

# Reproducible Benchmarking for High-Performance Computing Applications

PACT

Oct 19, 2026



Olga Pearce (LLNL), Doug Jacobsen (Google)

Prepared by LLNL under Contract DE-AC52-07NA27344.

# Tutorial Presenters



Olga Pearce  
LLNL



Doug Jacobsen  
Google

# Tutorial Agenda

|                                                                         |               |        |
|-------------------------------------------------------------------------|---------------|--------|
| Welcome and Introductions                                               | Olga Pearce   | 10 min |
| Benchpark Overview and Basics                                           | Olga Pearce   | 30 min |
| Ramble Overview and Basics                                              | Doug Jacobsen | 20 min |
| Hands on: Running an experiment, analyzing performance of scaling study | Olga Pearce   | 30 min |
| Break                                                                   | All           | 30 min |
| Hands on: Writing an Application specification in Ramble                | Doug Jacobsen | 30 min |
| Hands on: Writing an Experiment specification in Benchpark              | Olga Pearce   | 30 min |
| Tutorial Survey                                                         | All           | 10 min |
| Closing remarks, Q&A                                                    | All           | 20 min |

# Tutorial Materials

Find these slides and associated scripts here:

<http://software.llnl.gov/benchpark/interactive-tutorial/start-here.html>

We also have a channel room on Spack slack.

You can join here:

[slack.spack.io](https://slack.spack.io)

Join the **#benchpark-support** channel!

You can continue to ask questions here after the tutorial has ended.

# Tutorial Instances: <http://bit.ly/4kGQDlc>



- We have an AWS instance for the hands-on component of this tutorial
- The instance provides:
  - Pre-installed Benchmark and required dependencies
  - Job scheduler

When logging in to the instance:

- Please use a unique username to avoid resource allocation conflicts
  - First initial followed by last name (e.g., John Doe would be jdoe)
- PW: hpctutorial26

# We benchmark HPC systems for many reasons

- Procurements
  - Communicate datacenter workload to vendors
  - Co-design systems, monitor progress
  - System acceptance (contractual specification)
- Validation of software stack, tools
  - Compilers
  - Debuggers
  - Correctness tools
  - Performance tools
- Research
  - Programming models
  - Computational methods
  - Performance across architectures

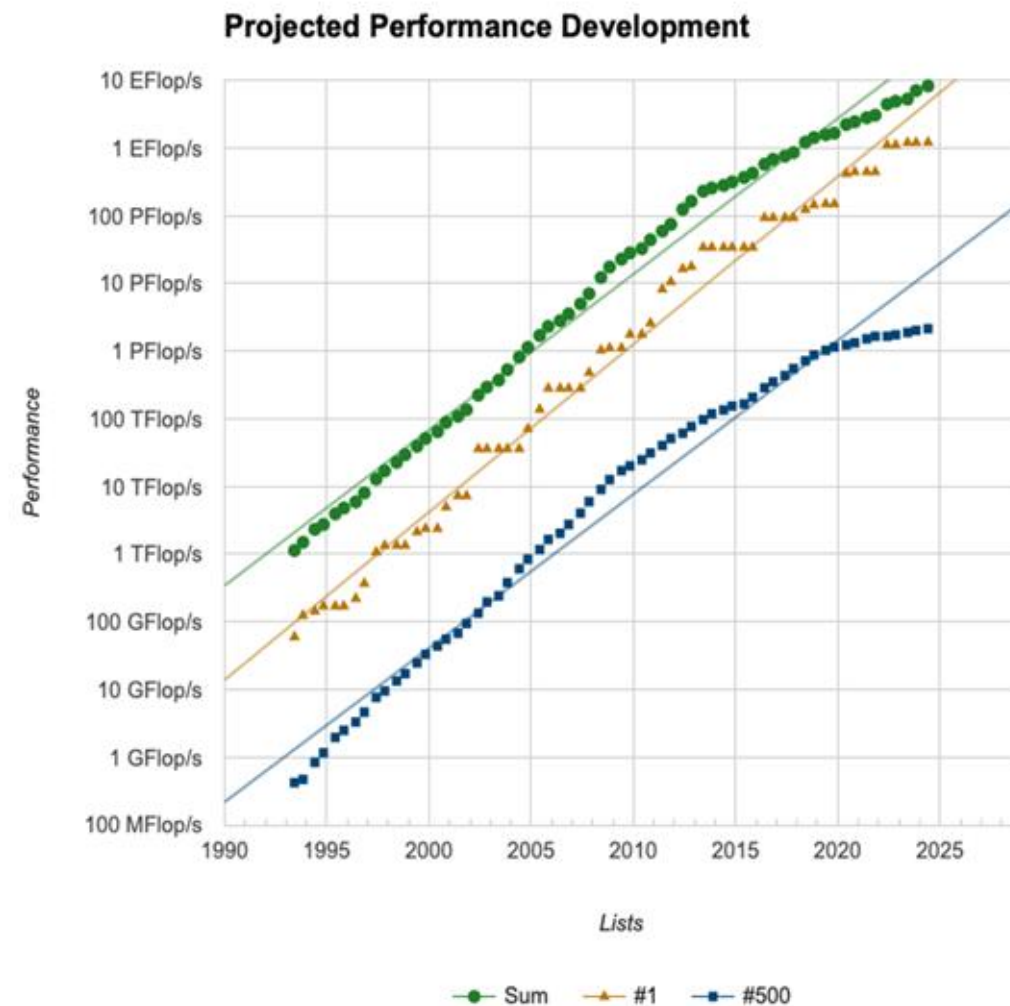


*Many stakeholders, communication is key to collaboration*

# Benchmarking is challenging

- What are we trying to characterize?
- Are we capturing the best the system can do?
- Is something else impacting performance?
- Did we build and run the code in the optimal and reproducible way?

Source: <https://www.top500.org/statistics/perfdevel/>



# HPC benchmarks run on diverse HPC hardware



Lawrence Berkeley Nat'l Lab  
**AMD Zen + NVIDIA**



**F u g a k u**  
RIKEN Fujitsu **ARM a64fx**



Lawrence Livermore Nat'l Lab  
**IBM Power9 + NVIDIA**



Lawrence Livermore Nat'l Lab  
**AMD Zen + Radeon**



Oak Ridge National Lab  
**AMD Zen + Radeon**



Argonne National Lab  
**Intel Xeon + Xe**

- Benchmark source code
  - Abstraction (OpenMP, RAJA, Kokkos)
  - Hardware-specific (CUDA, ROCm)
- Optimized code for the CPU and GPU
  - Must make effective use of the hardware
  - Can make 10-100x performance difference
- Rely heavily on system packages
  - Need to use optimized communication and MPI libraries that come with machines

# Writing benchmark source code is only the beginning



## State-of-the-practice:

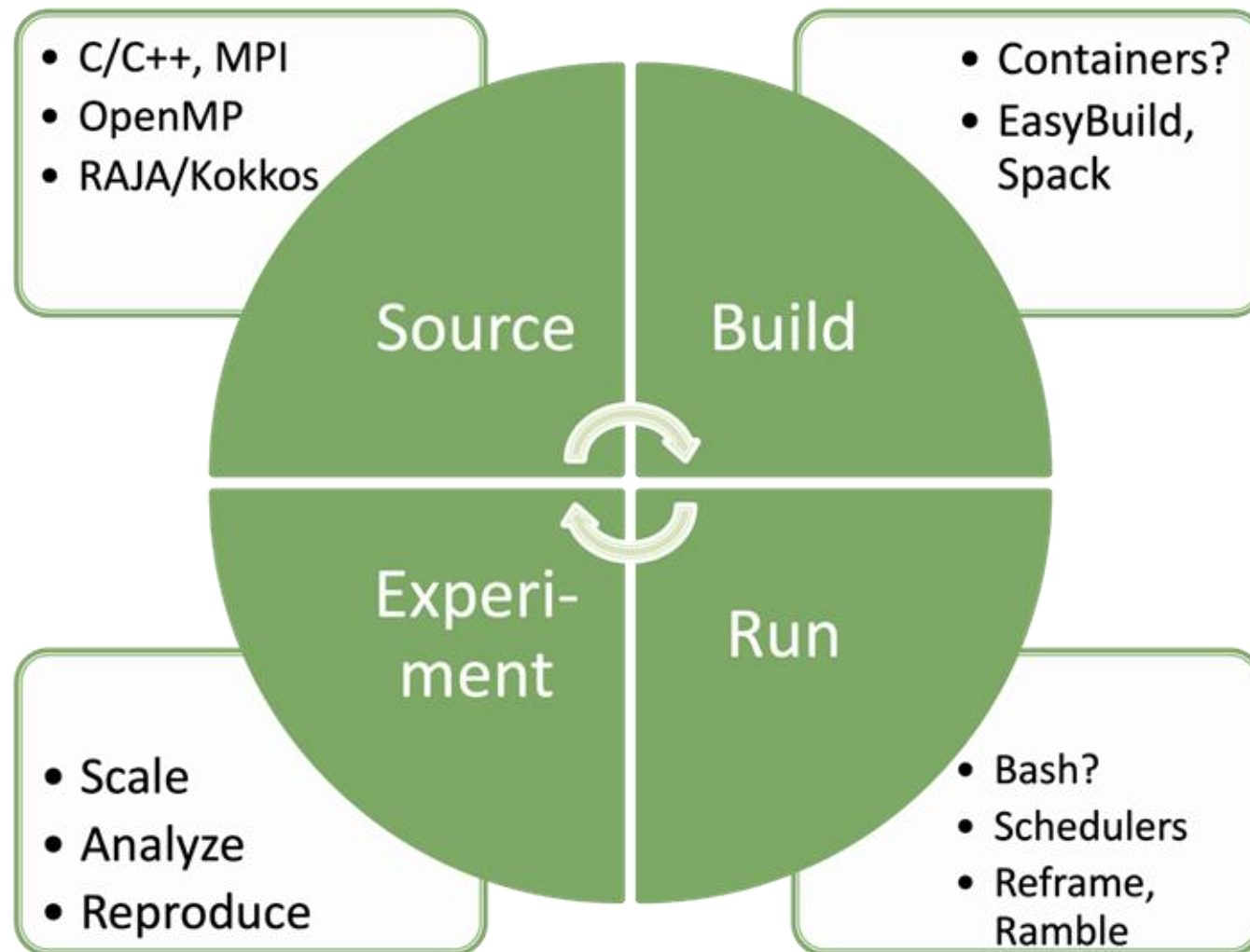
### HPC system benchmarking is manual!

- Building on each system is different, porting the builds to new systems is manual
- Running on each system is different, porting run scripts to new systems is manual
- Systems keep changing, requiring updates to how we build and run benchmarks
- Triggering builds and runs is manual: benchmark results don't stay up to date
- Performance analysis of results is manual

*Communicate technical specification via code, documentation, white papers*

# HPC benchmarks are HPC software

- Portability
- Maintenance
- Testing/CI
- Verification
- Reproducibility



All components must work for your system, focus on explainable performance

# Benchmarkpark enables complete specification of HPC benchmarks



Infrastructure-as-code benchmark specification codifies:

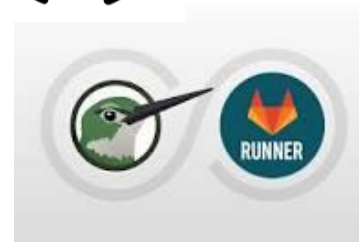
- Benchmark build and run instructions
- HPC Systems
- HPC Experiments

Leverage advances in HPC automation

- Source code
- Build specification
- Run specification
- CI



**github.com/llnl/benchmarkpark**



**Ramble**



**Spack**

Every part of the specification is codified: used to communicate, automate

# Benchpark enables reproducible specifications of benchmarks

## Specify

- How to build and run benchmarks on a system

## Run

- Run an experiment on a system

## Reproduce

- Re-run an experiment on a system

## Replicate

- Run an experiment on a new system

## Maintain

- CI: Run experiment on HPC systems

## Record

- Perf.measurements + *full spec* of experiment

Full specification enables reproducibility, replicability, and automation

# What is Ramble?

# What is Ramble?

Ramble is an open-source experimentation framework written in python.

Ramble's primary goals are to:

- **Accelerate productivity**
- **Improve experiment portability**
- **Increase reproducibility information**
- **Encoding domain knowledge**

Ramble is particularly good at generating parameterized “experiments”, but can be used for more than performance focused experiments.

# What is Ramble?

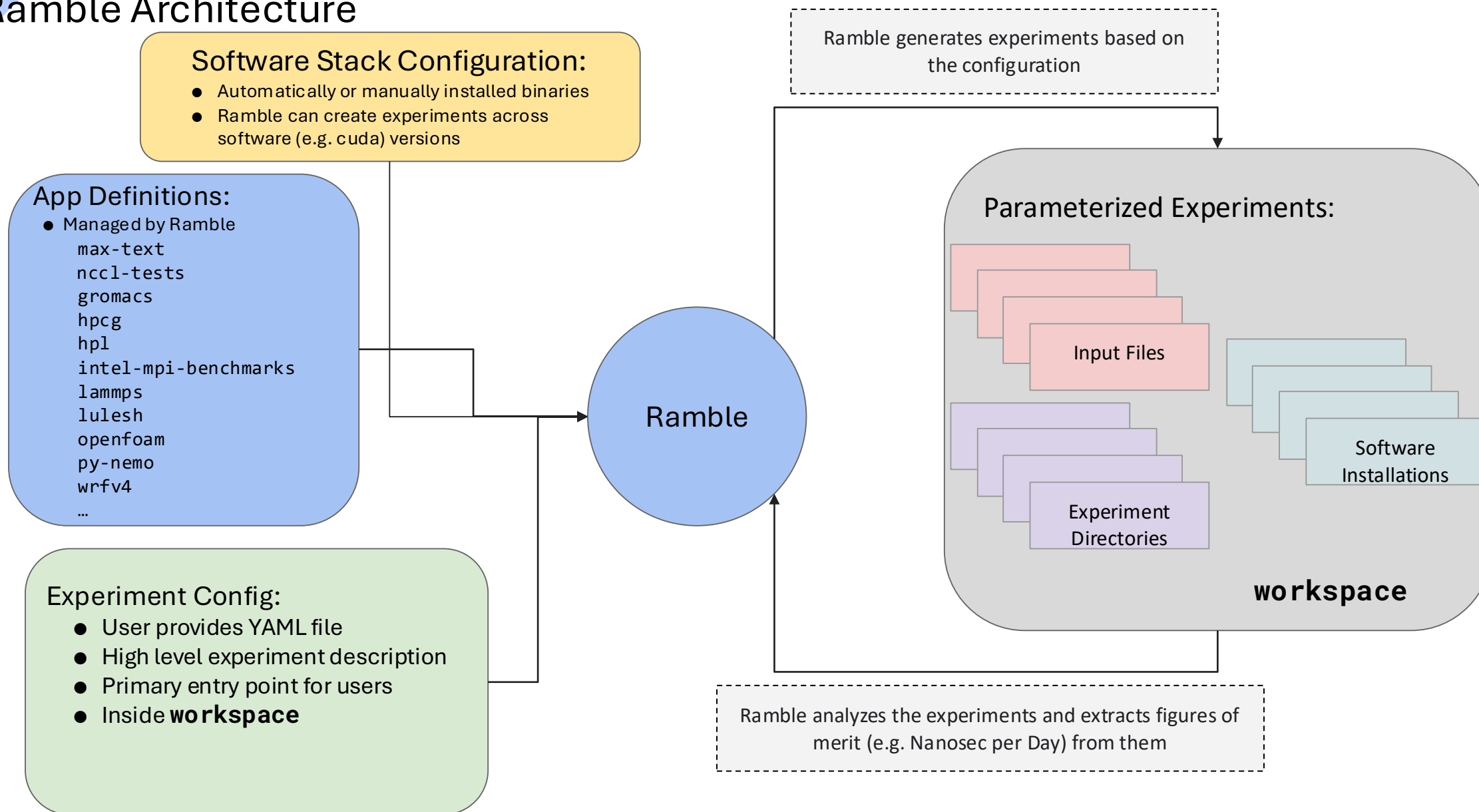
Ramble is:

- Written in python
- Multi-platform
- Heavily based on Spack (Written on top of Spack's infrastructure)
- An experimentation framework (Related to running experiments like Spack is to building software)
- Open Source: <https://github.com/GoogleCloudPlatform/ramble>

Ramble provides:

- A flexible templating engine
- A domain specific language for describing an application and its workloads
- Standardized definitions of how workloads are configured and executed

# Ramble Architecture



# Ramble Components

## Object Definitions

**Portable** definitions written in python, that represent:

Applications:

- Executables (commands to run)
- Input files (things to download)
- Software (things to install)
- Workloads (executables + input files)
- Figures of merit (things to analyze)

Package Managers (spack, eessi)

Workflow Managers (slurm, batch, gke)

Modifiers (lscpu, intel-aps, etc..)

Ramble supports **public** and **private** object definitions through configurable repos.

Does not include experiment or system details.

## Experiment Configuration (Workspace)

Self contained directory describing a set of **parameterized** experiments to execute.

Contains:

- YAML File, describes:
  - Experiments
  - Software stack
  - Execution workflow
- Template files

After setup, contains:

- Input files
- Experiment execution directories
- Software environments

## Experiment Directory

Each experiment has **their own** execution directory.

These contain:

- Rendered versions of the template files (which can then be executed)

After executing the might contain:

- Links to the input files
- Output files from the experiment to be analyzed

# Ramble Components

## Software Stack

Ramble supports using a predefined software stack, or generating a new software stack to produce experiments.

Supported software stack workflows in Ramble include:

- Pre-installed software (Considered “Existing”)
  - e.g. ISV applications installed in a static location
  - Software installed manually through another mechanism (yum, apt, pip, etc..)
- Driving the creation of a software stack:
  - Using a 3P package manager (currently supports [spack](#), [pip](#), [eessi](#), [environment-modules](#))
  - Can parameterize various aspects of the stack
  - Can use a binary cache to accelerate deployments, and provide binary objects to customers

When Ramble manages the software stack, it can parameterize portions of the stack to generate various studies. This includes:

- Changing compilers
- Changing dependencies (e.g. MPI impl., CUDA vs. ROCM)
- Package versions (e.g. CUDA versions)
- Optimization Targets (e.g. Icelake vs. Zen 3)
- etc...

Allows very flexible descriptions of the software environment

Spack is used to distribute [well defined software stacks](#).

Various organizations provide public spack caches, which can make it faster to deploy a complex software stack.

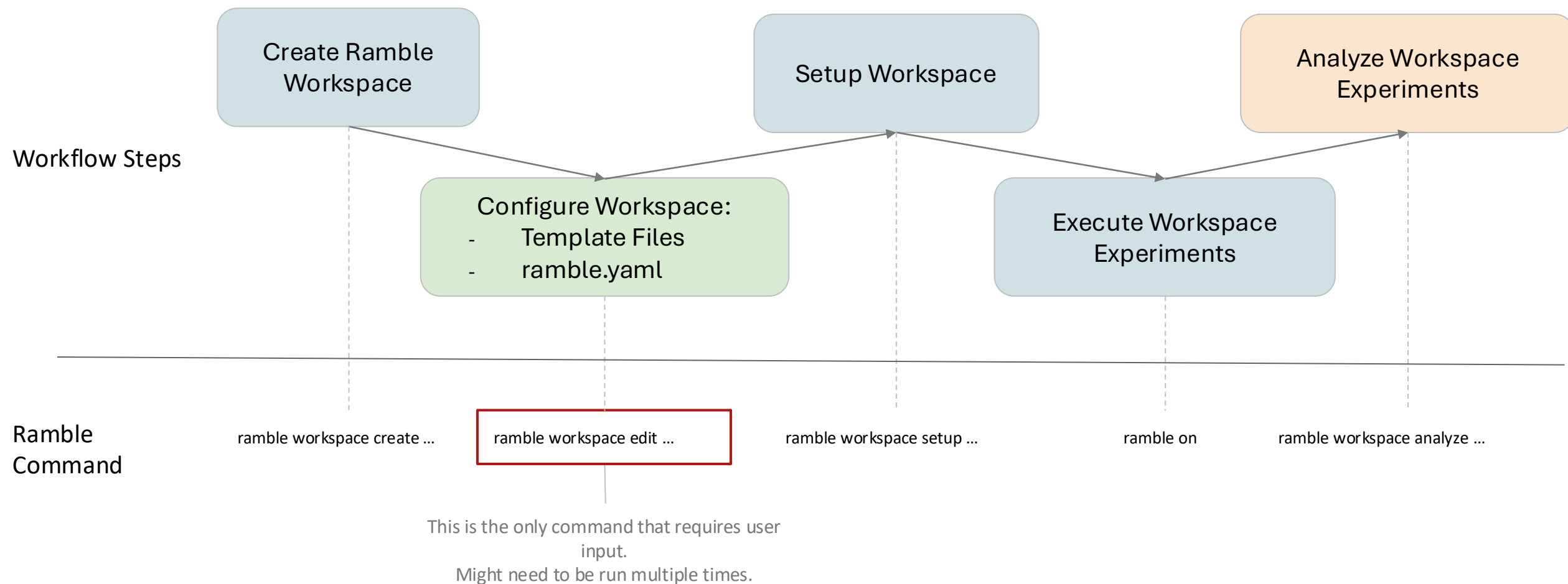
# How does Ramble work?

# What is Ramble's Workflow?

Ramble Controlled Step

User Input Step

Output Step



# Ramble Workspace:

A workspace is a self contained directory, that contains:

- software environments
- input files
- experiments

Workspaces are independent, and should represent a set of independent experiments you want to execute.

## Workspace: my\_experiments

```
configs:  
└─ ramble.yaml  
   execute_experiment.tpl
```

inputs

software

experiments

# Workspace Config:

Ramble's workspace config YAML can generate many experiments with only a little syntax.

applications:

wrfv3: ← Application Definition Name

n\_repeats: 5 } Define 5 repeats per experiment

variables:

processes\_per\_node: [30, 56]  
partition: ['c2', 'c2d'] } Define two vectors, of length 2.

n\_ranks: '{processes\_per\_node}\*{n\_nodes}'

workloads:

CONUS\_2p5km: ← Workload Name (from Application Definition)

experiments:

scaling\_study\_{partition}\_{nodes}nodes:

variables:

n\_nodes: [1, 2, 4, 8, 16, 32] } Define one vector, of length 6

matrix:  
- n\_nodes } Define one matrix, using the n\_nodes vector. Shape is 1x6.

- Vectors that are not used by a matrix are zipped together (must be the same length)
- Zip of vectors are crossed with any matrices
- Result is: 1x6 (matrix) x 2 (vectors) x 5 (repeats) = 60, where each index is a 4-element tuple: (processes\_per\_node, n\_ranks, n\_nodes, repeat\_idx)

# Workspace Config:

In addition to the YAML config, Ramble has a template engine to render scripts:

```
#!/bin/bash
```

```
{command}
```



```
#!/bin/bash
```

```
source <path/to/spack>  
spack env activate <path/to/env>  
cp <inputs>/* <experiment_dir>/.  
mpirun -n <n_ranks> wrf.exe
```

# Workspace Setup:

## Workspace: wrf-demo

```
configs:  
  - ramble.yaml  
  - execute_experiment.tpl
```

inputs

software

experiments

### Execute input phases:

- Determine which inputs are necessary
- Download them
- Define variables for referring to these inputs

### Execute software phases (Only used on SpackApplications currently):

- Install necessary compilers
- Create spack environments for the experiments
- Using spack, install the required software
- Define variables for referring to the spack environments

### Execute experiment creation phases:

- Create experiment execution directories
- Define variables specific to the experiment
- Render any \*.tpl files into the execution directories
- Append the experiment to \$workspace/all\_experiments

# Execute Experiments:

## Workspace: wrf-demo

```
configs:  
  └─ ramble.yaml  
     execute_experiment.tpl
```

```
software/  
  └─ wrfv3.CONUS_2p5km  
     └─ spack.yaml
```

```
inputs:  
  └─ ...
```

all\_experiments

```
experiments/  
  └─ wrfv3  
     └─ CONUS_2p5km  
        ├── scaling_study_c2_16nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2_1nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2_2nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2_32nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2_4nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2_8nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2d_16nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2d_1nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2d_2nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2d_32nodes  
        │   └─ execute_experiment  
        ├── scaling_study_c2d_4nodes  
        │   └─ execute_experiment  
        └─ scaling_study_c2d_8nodes  
           └─ execute_experiment
```

After setting up a workspace, the experiments can be executed using:

- `$workspace/all_experiments`
- `ramble on` (with an activated workspace)

Depending on the `ramble.yaml` and `execute_experiment.tpl` this:

- Executes experiments sequentially
- Submits experiments to a workload manager

# Analyze Workspace:

After experiments are executed, Ramble can extract their figures of merit.

This is done through:

```
ramble workspace analyze
```

Ramble will process the output files described in an experiment's `application.py`, and extract success criteria and figures of merit.

These are then written to a file (e.g. `results.txt`) in the provided format (can be text, yaml, or json).



# What other things can you do with Ramble?

# Parameterizing Software Definitions

A common workflow in optimizing software is exploring how modifications to a software stack impact performance. Ramble's support of these workflows can be seen below:

```
ramble:
```

```
...
applications:
  wrfv3:
    workloads:
      CONUS_2p5km:
        experiments:
          '{mpi}_{partition}_{n_nodes}nodes':
            variables:
              n_nodes: [1, 2, 4, 8, 16, 32]
              mpi: ['impi', 'ompi']
              env_name: 'wrf-{mpi}'
            matrix:
              - n_nodes
              - mpi
```

```
[ramble:]
```

```
...
software:
  packages:
    wrf:
      pkg_spec: wrf@3.9.1.1
    impi:
      pkg_spec: intel-oneapi-mpi@2021.9
    ompi:
      pkg_spec: openmpi@4.1.4
  environments:
    wrf-{mpi}:
      packages:
        - wrf
        - '{mpi}'
```

# Parameterizing Software Definitions

A common workflow in optimizing software is exploring how modifications to a software stack impact performance. Ramble's support of these workflows can be seen below:

ramble:

```
...
applications:
  wrfv3:
    workloads:
      CONUS_2p5km:
        experiments:
          '{mpi}_{partition}_{n_nodes}nodes':
            variables:
              n_nodes: [1, 2, 4, 8, 16, 32]
              mpi: ['impi', 'ompi']
              env_name: 'wrf-{mpi}'
            matrix:
              - n_nodes
              - mpi
```

[ramble:]

```
...
software:
  packages:
    wrf:
      pkg_spec: wrf@3.9.1.1
      impi:
      pkg_spec: intel-oneapi-mpi@2021.9
      ompi:
      pkg_spec: openmpi@4.1.4
    environments:
      wrf-{mpi}:
        packages:
          - wrf
          - '{mpi}'
```

Defines spack environments

Variable  
expansion

MPI  
packages

# Performance Analysis Tools

Modifiers all composable definitions to edit the experiment behavior

ramble:

```
...
modifiers:
- name: intel-aps
applications:
  wrfv3:
    workloads:
      CONUS_2p5km:
        experiments:
          '{mpi}_{partition}_{n_nodes}nodes':
            variables:
              n_nodes: [1, 2, 4, 8, 16, 32]
              mpi: ['impi', 'ompi']
              env_name: 'wrf-{mpi}'
            matrix:
              - n_nodes
              - mpi
```

[ramble:]

```
...
software:
  packages:
    wrf:
      pkg_spec: wrf@3.9.1.1
    impi:
      pkg_spec: intel-oneapi-mpi@2021.9
    ompi:
      pkg_spec: openmpi@4.1.4
  environments:
    wrf-{mpi}:
      packages:
        - wrf
        - '{mpi}'
```

# Performance Analysis Tools

Modifiers all composable definitions to edit the experiment behavior

ramble:

```
...
modifiers:
- name: intel-aps ←
applications:
  wrfv3:
    workloads:
      CONUS_2p5km:
        experiments:
          '{mpi}_{partition}_{n_nodes}nodes':
            variables:
              n_nodes: [1, 2, 4, 8, 16, 32]
              mpi: ['impi', 'ompi']
              env_name: 'wrf-{mpi}'
            matrix:
              - n_nodes
              - mpi
```

[ramble:]

```
...
software:
  packages:
    wrf:
      pkg_spec: wrf@3.9.1.1
    impi:
      pkg_spec: intel-oneapi-mpi@2021.9
    ompi:
      pkg_spec: openmpi@4.1.4
  environments:
    wrf-{mpi}:
      packages:
        - wrf
        - '{mpi}'
```

Performance Analysis  
Tool

# Hands On

Follow script at [http://software.llnl.gov/benchpark/interactive\\_tutorial/add-an-experiment.html](http://software.llnl.gov/benchpark/interactive_tutorial/add-an-experiment.html)

Tutorial Instances: <http://bit.ly/4kGQDlc>

- We have an AWS instance for the hands-on component of this tutorial
- The instance provides:
  - Pre-installed Benchpark and required dependencies
  - Job scheduler



When logging in to the instance:

- Please use a unique username to avoid resource allocation conflicts
  - First initial followed by last name (e.g., John Doe would be jdoe)
- PW: hpctutorial26

# Tutorial Materials

Find these slides and associated scripts here:

<http://software.llnl.gov/benchpark/interactive-tutorial/start-here.html>

We also have a channel room on Spack slack.

You can join here:

[slack.spack.io](https://slack.spack.io)

Join the **#benchpark-support** channel!

You can continue to ask questions here after the tutorial has ended.

# Who does Benchpark target

**People who want to use or distribute benchmarks for HPC!**

## **1. End Users of HPC Benchmarks**

- Install, run, analyze performance of HPC benchmarks

## **2. Benchmark Developers**

- People who want to share their benchmarks

## **3. Procurement teams at HPC Centers**

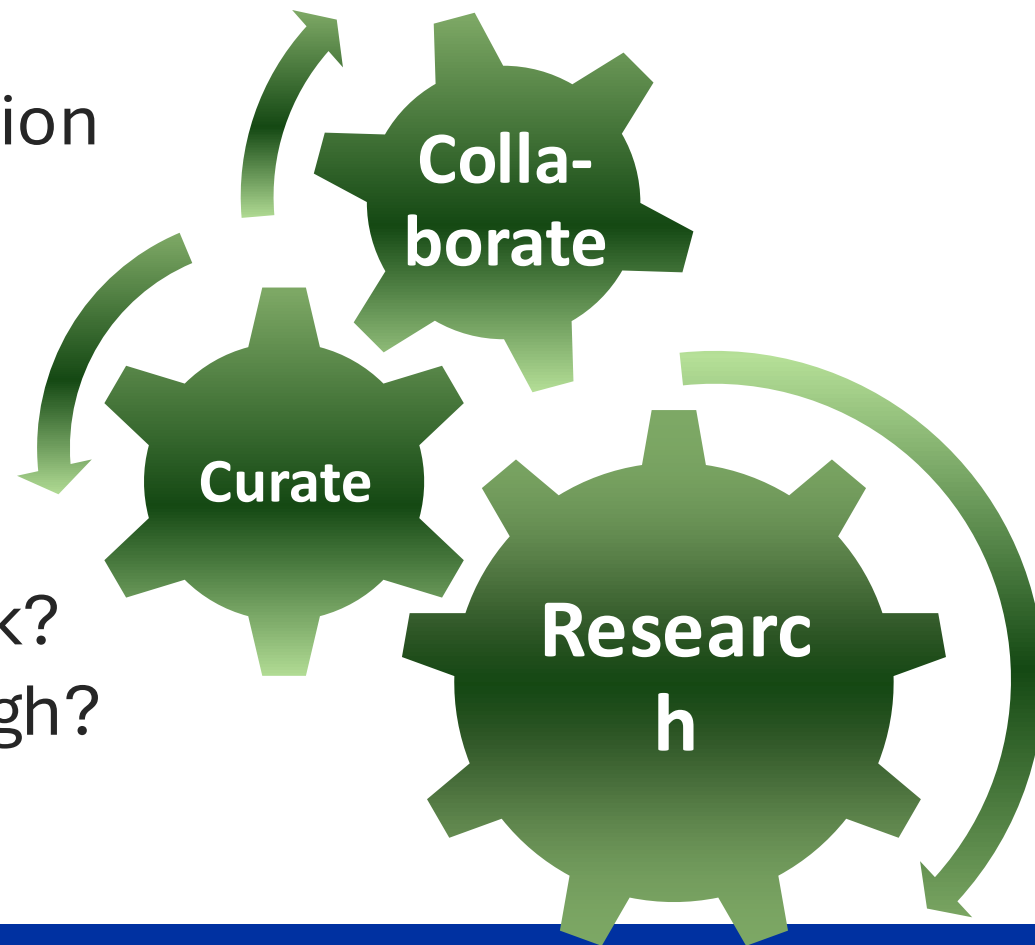
- Curate workload representation, evaluate and monitor system progress

## **4. HPC Vendors**

- Understand the curated workload of HPC centers, propose systems

# Catalogued library of *working* benchmarks

- Enables exploration of large configuration space
  - Architectural
  - Software stack
  - Temporal
- What architecture and system configuration is best for my benchmark?
- On my system, is OpenMPI good enough?
- What purpose will Benchpark help you address?



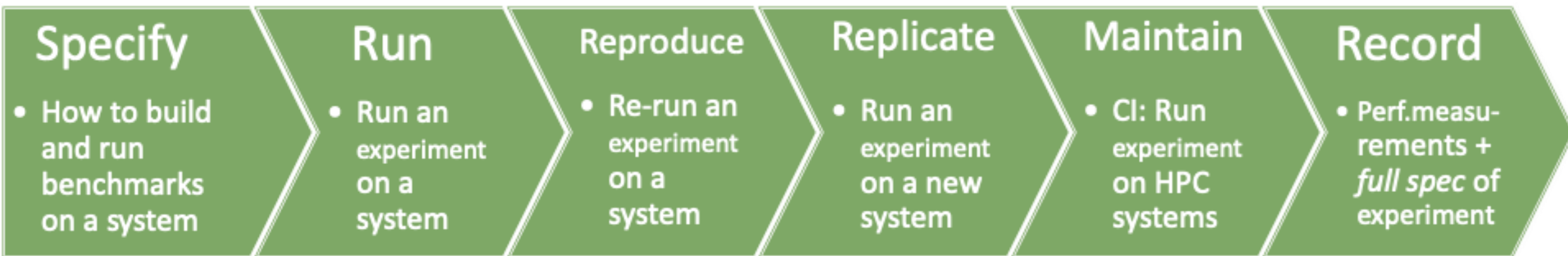
Building a community to contribute/benefit

# Benchmark: Open collaborative repository for reproducible specifications of HPC benchmarks

Infrastructure-as-code benchmark specification

enables **reproducibility, replicability, and automation**

- HPC Systems
- HPC Experiments



- Tagging, keywords for publications
- Performance metrics, metrics of usefulness
- Dashboards: Archive specs+results

- CI pipelines on PRs from GitHub at data centers across the world and in the cloud

\*Olga Pearce et al, Continuous Benchmarking, HPCTests SC|23

\*Olga Pearce et al, Repeat, Replicate, Reproduce, ACM REP 2025

Full specification enables reproducibility, replicability, and automation

# Benchpark roadmap and community engagement

## Future directions:

- Suite curation: Reproducible specification of an entire suite
- Tagging: Keywords for publications, finding benchmarks
- Metrics: Performance, usefulness
- Dashboards: Archive+share *specs*+results, Slices in configuration space
- CI pipelines at data centers across the world and in the cloud

## Community Engagement:

- Co-design / vendors on board, but incentive for app teams? (carrot or stick?)
- Who owns which parts of the specification and approves changes?
- Who finances R&D and maintenance?
- ROI for the people working on it? → think about post docs, researchers, etc.

Collaboration, reproducibility, fully-specified public results



# Connect with us after the tutorial!

Tutorial material:

<http://software.llnl.gov/benchpark/interactive-tutorial/start-here.html>

Connect with us on Spack slack

**#benchpark-support**

We want your  
feedback!



Contribute systems, benchmarks, experiments,  
and lectures on GitHub



[slack.spack.io](https://slack.spack.io)

**#benchpark-support**



★ Star us on GitHub!

[github.com/llnl/benchpark](https://github.com/llnl/benchpark)





# Backup Slides



#### Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

# Three Rs in HPC Benchmarking

1. Repeat
2. Replicate
3. Reproduce

# Three Rs in HPC Benchmarking

1. Repeat: run on same system

2. Replicate

3. Reproduce

| Component       | Repeat |
|-----------------|--------|
| (1) System H/W  | ✓      |
| (2) System S/W  | ✓      |
| (3) Application | ✓      |

- Verifying benchmark still builds and runs
- Handing over benchmarking tasks from one user to another
- Evaluating benchmark performance changes as system ages
- Running multiple trials of a benchmark for noise variability

# Three Rs in HPC Benchmarking

## 1. Repeat

## 2. Replicate: run with a different software stack

## 3. Reproduce

| Component       | Repeat | Replicate |
|-----------------|--------|-----------|
| (1) System H/W  | ✓      | ✓         |
| (2) System S/W  | ✓      | ≈         |
| (3) Application | ✓      | ✓         |

- Verifying benchmark still builds and runs with update system software
- Validating performance of updated system software
- Evaluating performance differences of different software substitutes (e.g., gcc or icc compilers)

# Three Rs in HPC Benchmarking

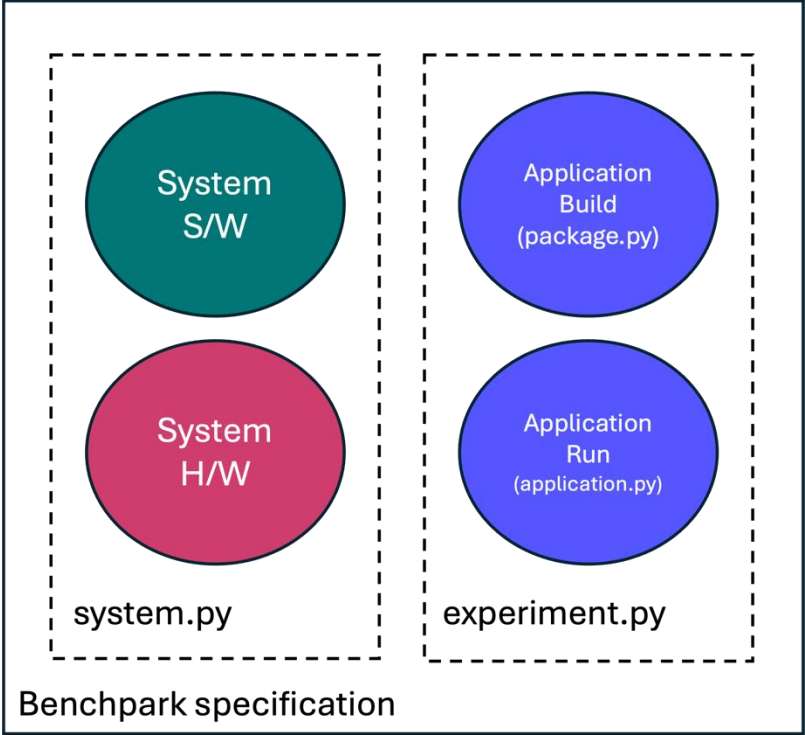
1. Repeat
2. Replicate
3. Reproduce: run on a different system

- Verifying benchmark builds and runs on diverse hardware
- Evaluating benchmark performance across different hardware
- Exploring potential benefits of hardware upgrades

| Component       | Repeat | Replicate | Reproduce |
|-----------------|--------|-----------|-----------|
| (1) System H/W  | ✓      | ✓         | X         |
| (2) System S/W  | ✓      | ≈         | X         |
| (3) Application | ✓      | ✓         | ≈         |

# Benchmark Specification for Three Rs

|     | Component   | Repeat | Replicate | Reproduce |
|-----|-------------|--------|-----------|-----------|
| (1) | System H/W  | ✓      | ✓         | X         |
| (2) | System S/W  | ✓      | ≈         | X         |
| (3) | Application | ✓      | ✓         | ≈         |



# Benchmark Specification for the Three Rs

|     | Component   | Repeat | Replicate | Reproduce |
|-----|-------------|--------|-----------|-----------|
| (1) | System H/W  | ✓      | ✓         | X         |
| (2) | System S/W  | ✓      | ≈         | X         |
| (3) | Application | ✓      | ✓         | ≈         |

Repeat as  
a different  
user

```
1 sysa:~ user1$ benchpark system init --dest=sysa intel-cluster  
cluster=sysa  
2 sysa:~ user1$ benchpark experiment init --dest=qs  
quicksilver +openmp +weak  
3 sysa:~ user1$ benchpark setup ./qs ./sysa workspace/
```

```
1 sysa:~ user2$ benchpark system init --dest=sysa intel-  
cluster cluster=sysa  
2 sysa:~ user2$ benchpark experiment init --dest=qs  
quicksilver +openmp +weak  
3 sysa:~ user2$ benchpark setup ./qs ./sysa workspace2/
```

Replicate  
with  
different  
SW stack

```
1 sysa:~ user1$ benchpark system init --dest=sysa1 intel-  
cluster cluster=sysa compiler=gcc1  
2 sysa:~ user1$ benchpark experiment init --dest=qs  
quicksilver +openmp +weak  
3 sysa:~ user1$ benchpark setup ./qs ./sysa1 workspace/
```

```
1 sysa:~ user2$ benchpark system init --dest=sysa2 intel-  
cluster cluster=sysa compiler=gcc2  
2 sysa:~ user2$ benchpark experiment init --dest=qs  
quicksilver +openmp +weak  
3 sysa:~ user2$ benchpark setup ./qs ./sysa2 workspace2/
```

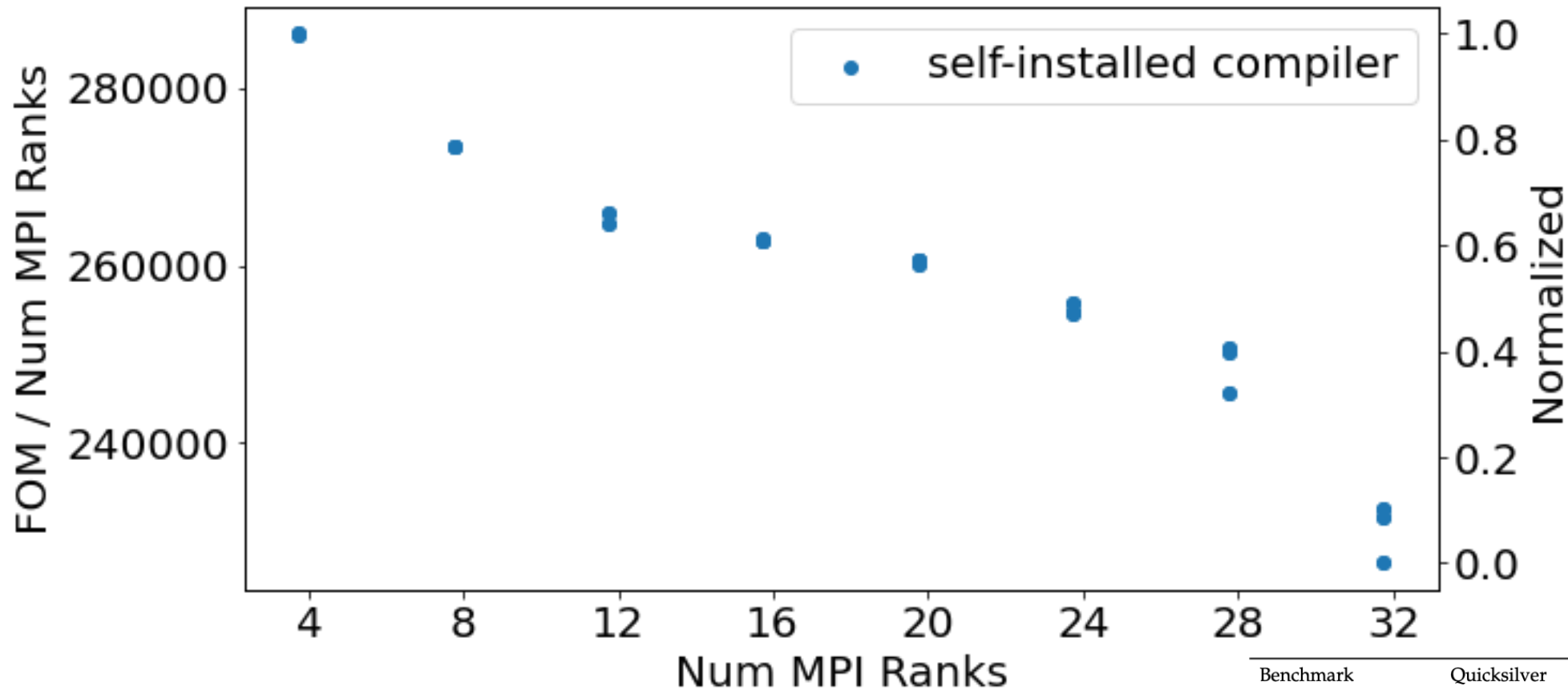
Reproduce  
on a  
different  
system

```
1 sysa:~ user1$ benchpark system init --dest=sysa intel-  
cluster cluster=sysa  
2 sysa:~ user1$ benchpark experiment init --dest=qs-openmp  
quicksilver +openmp +weak  
3 sysa:~ user1$ benchpark setup ./qs-openmp ./sysa worksp/
```

```
1 sysb:~ user2$ benchpark system init --dest=sysb ibm-cluster  
2 sysb:~ user2$ benchpark experiment init --dest=qs-cuda  
quicksilver +cuda +weak  
3 sysb:~ user2$ benchpark setup ./qs-cuda ./sysb work2/
```

# Repeat runs of Quicksilver on Ruby using *same compiler*

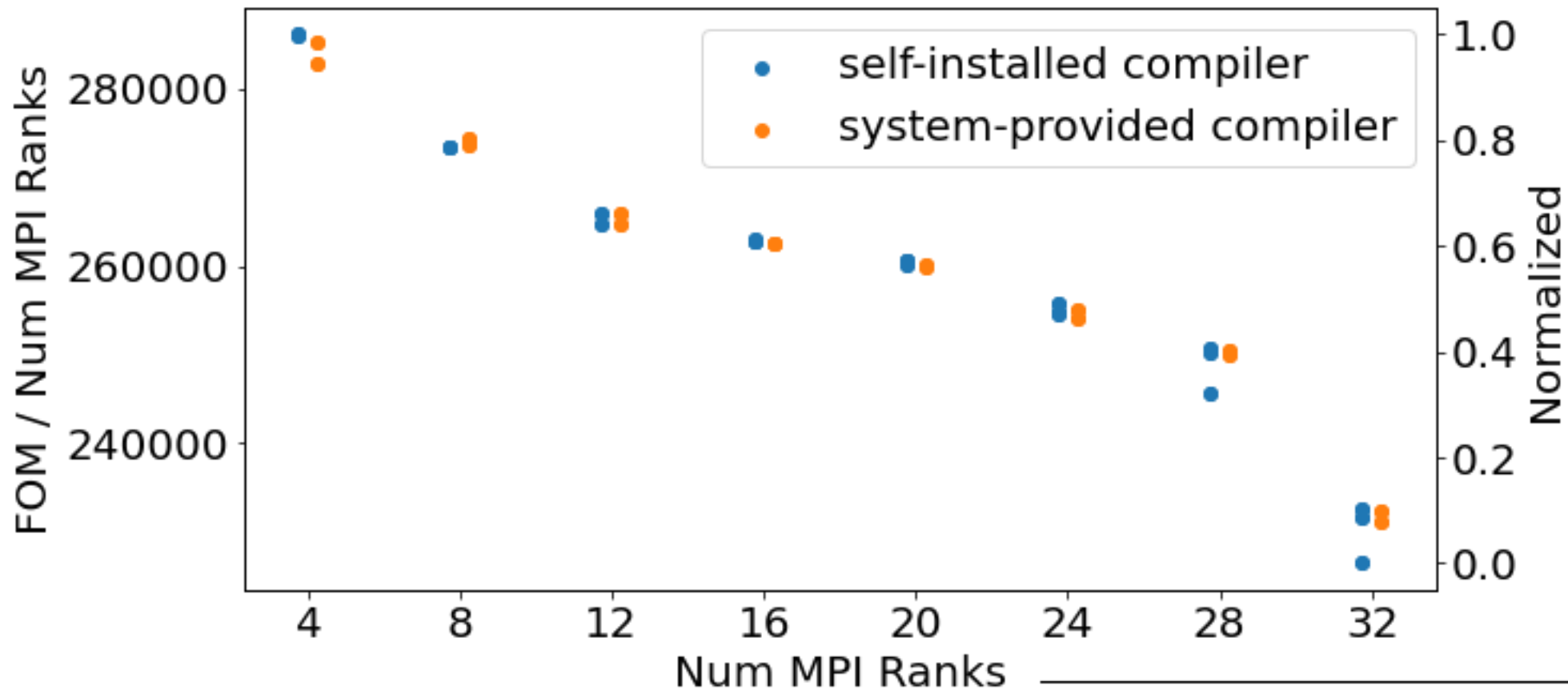
| Component       | Repeat | Replicate | Reproduce |
|-----------------|--------|-----------|-----------|
| (1) System H/W  | ✓      | ✓         | X         |
| (2) System S/W  | ✓      | ≈         | X         |
| (3) Application | ✓      | ✓         | ≈         |



| Benchmark   | Quicksilver                              |
|-------------|------------------------------------------|
| Variants    | +mpi, +openmp                            |
| Num Ranks   | 4, 8, 12, 16, 20, 24, 28, 32             |
| Num Nodes   | 1                                        |
| Scaling     | weak                                     |
| Repetitions | 4 (Increase as appropriate for analysis) |

# Replicate runs of Quicksilver on Ruby with *different compilers*

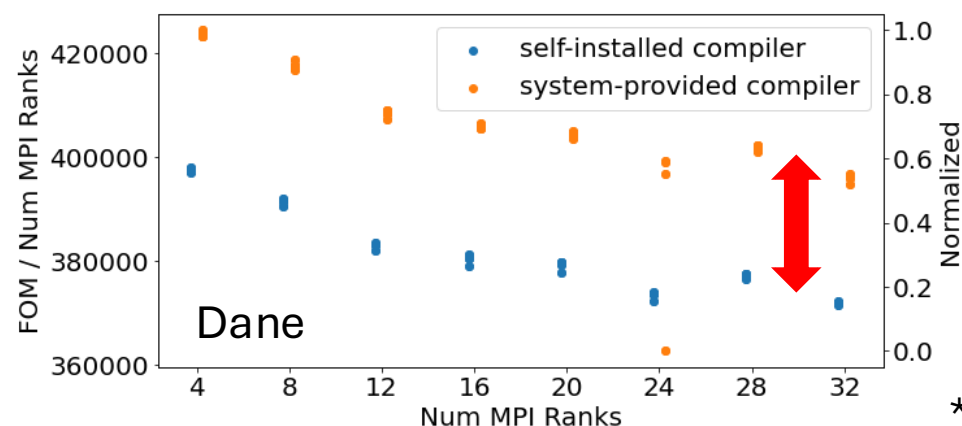
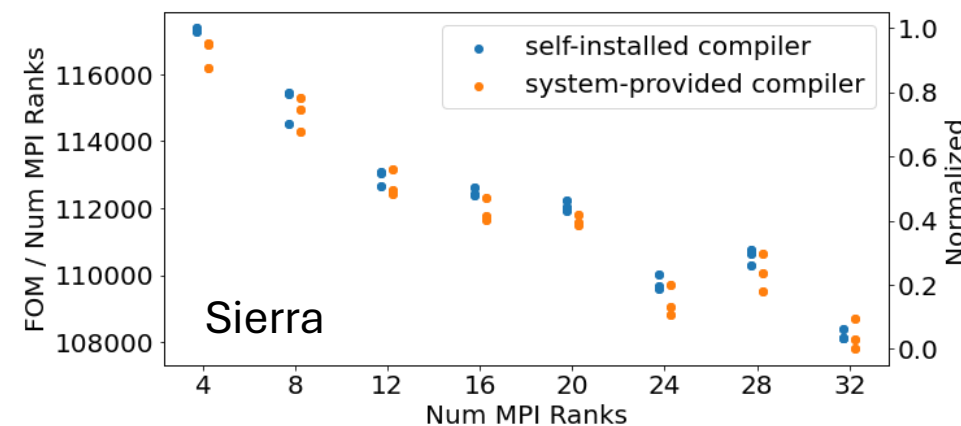
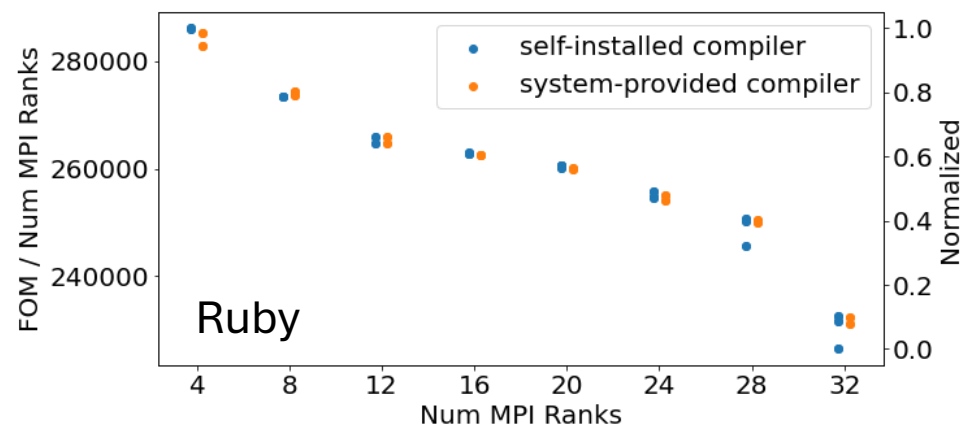
|     | Component   | Repeat | Replicate | Reproduce |
|-----|-------------|--------|-----------|-----------|
| (1) | System H/W  | ✓      | ✓         | X         |
| (2) | System S/W  | ✓      | ≈         | X         |
| (3) | Application | ✓      | ✓         | ≈         |



| Compiler | Version | How Installed   |
|----------|---------|-----------------|
| gcc      | 12.1.1  | system-provided |
| gcc      | 12.1.0  | self-installed  |

# Reproduce runs of Quicksilver across *different systems*

| Component       | Repeat | Replicate | Reproduce |
|-----------------|--------|-----------|-----------|
| (1) System H/W  | ✓      | ✓         | X         |
| (2) System S/W  | ✓      | ≈         | X         |
| (3) Application | ✓      | ✓         | ≈         |



| System | P-value    | Statistically Significant? |
|--------|------------|----------------------------|
| Ruby   | 0.490      | No                         |
| Sierra | 0.349      | No                         |
| Dane   | 0.00000105 | Yes                        |

| System Name | CPU Hardware          | Number of Cores |
|-------------|-----------------------|-----------------|
| Ruby        | Intel Cascade Lake    | 56              |
| Sierra      | IBM Power 9           | 44              |
| Dane        | Intel Sapphire Rapids | 112             |

\*Gap on Dane is statistically significant



# Analysis of calltrees to understand performance difference using Thicket

```

0.000 main
├─ 0.134 qs.mainloop
│   ├── 0.019 MPI_Allreduce
│   ├── 0.228 cycleFinalize
│   │   ├── 0.003 MPI_Allreduce
│   │   └─ 0.000 MPI_Wait
│   ├── 0.558 cycleInit
│   │   ├── 0.011 MPI_Allreduce
│   │   └─ 0.022 MPI_Barrier
│   └─ 0.022 cycleTracking
│       ├── 0.000 MPI_Cancel
│       ├── 0.001 MPI_Irecv
│       ├── 0.000 MPI_Wait
│       └─ 230.542 cycleTracking_Kernel
│           └─ 1.158 cycleTracking_MPI
│               ├── 0.018 MPI_Irecv
│               ├── 0.023 MPI_Isend
│               ├── 0.108 MPI_Test
│               ├── 0.046 MPI_Wait
│               └─ 0.001 cycleTracking_Test_Done
│                   └─ 20.566 MPI_Allreduce
└─ 0.000 cycleTracking_Test_Done
    └─ 0.000 MPI_Allreduce
    
```

Self-Installed Compiler

```

0.000 main
├─ 0.135 qs.mainloop
│   ├── 0.025 MPI_Allreduce
│   ├── 0.226 cycleFinalize
│   │   ├── 0.003 MPI_Allreduce
│   │   └─ 0.000 MPI_Wait
│   ├── 0.559 cycleInit
│   │   ├── 0.014 MPI_Allreduce
│   │   └─ 0.024 MPI_Barrier
│   └─ 0.022 cycleTracking
│       ├── 0.000 MPI_Cancel
│       ├── 0.001 MPI_Irecv
│       ├── 0.000 MPI_Wait
│       └─ 219.612 cycleTracking_Kernel
│           └─ 1.165 cycleTracking_MPI
│               ├── 0.019 MPI_Irecv
│               ├── 0.023 MPI_Isend
│               ├── 0.106 MPI_Test
│               ├── 0.046 MPI_Wait
│               └─ 0.001 cycleTracking_Test_Done
│                   └─ 19.478 MPI_Allreduce
└─ 0.000 cycleTracking_Test_Done
    └─ 0.000 MPI_Allreduce
    
```

System-Provided Compiler

Legend (Metric: Avg time/rank (exc))

|   |                 |
|---|-----------------|
| ■ | 207.49 – 230.54 |
| ■ | 161.38 – 207.49 |
| ■ | 115.27 – 161.38 |
| ■ | 69.16 – 115.27  |
| ■ | 23.05 – 69.16   |
| ■ | 0.00 – 23.05    |

# Takeaways

- Formalize how to repeat, replicate, and reproduce HPC benchmarking experiments
- Showcase how Benchpark enables reproducibility through specification of experiments and HPC systems
- Full record of the specification is preserved for introspection and reproducibility

# Performance Tools: Caliper



- Caliper is an performance profiling library
- Integrates a performance profiler into your program
  - Profiling is always available
  - Simplifies performance profiling for application end users
- Common instrumentation interface
  - Provides program context information for other tools
- Designed for HPC
  - Supports MPI, OpenMP, CUDA, HIP, Kokkos, RAJA
- [software.llnl.gov/Caliper](https://software.llnl.gov/Caliper)

# Performance Tools: Thicket



- Thicket is a toolkit for Exploratory Data Analysis
- Enables exploratory data analysis of multi-run data
- Compose data from diff. sources and types
  - Different scaling (e.g., strong, weak)
  - Different application parameters
  - Different compilers and optimization levels
  - Different hardware types (e.g., CPUs, GPUs)
  - Different performance tools
- Perform analysis on the thicket of runs
  - Manipulate the set of data
  - Visualize the dataset
  - Perform analysis on the data
  - Model data
  - Leverage third-party tools in the Python ecosystem

[thicket.readthedocs.io](https://software.llnl.gov/benchpark/tutorial-101.html)

# Ramble modifiers in Benchpark encapsulate reusable patterns to perform a specific configuration of an experiment

- Affinity modifier for pinning threads
- Allocation modifier to request resources
- Hwloc modifier to capture hardware topology
- Caliper modifier to profile an application

```
benchpark experiment init --dest=amg amg2023 --system=elcap --rocm workload=problem2 +strong caliper=mpi.time
```

# Ramble Tutorial: Writing an HPL application definition

- [https://ramble.readthedocs.io/en/latest/dev\\_guides/2\\_hpl\\_application\\_definition\\_tutorial.html](https://ramble.readthedocs.io/en/latest/dev_guides/2_hpl_application_definition_tutorial.html)