Integrating Variorum with System Software and Tools

Module 2 of 2, ECP Lecture Series

Tapasya Patki, Aniruddha Marathe,
Stephanie Brink, and Barry Rountree

13 August 2021 8:30AM-10:00AM PDT

27 August 2021 4:00PM-5:30PM PDT (Repeat)

LLNL-PRES-825680

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory
under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

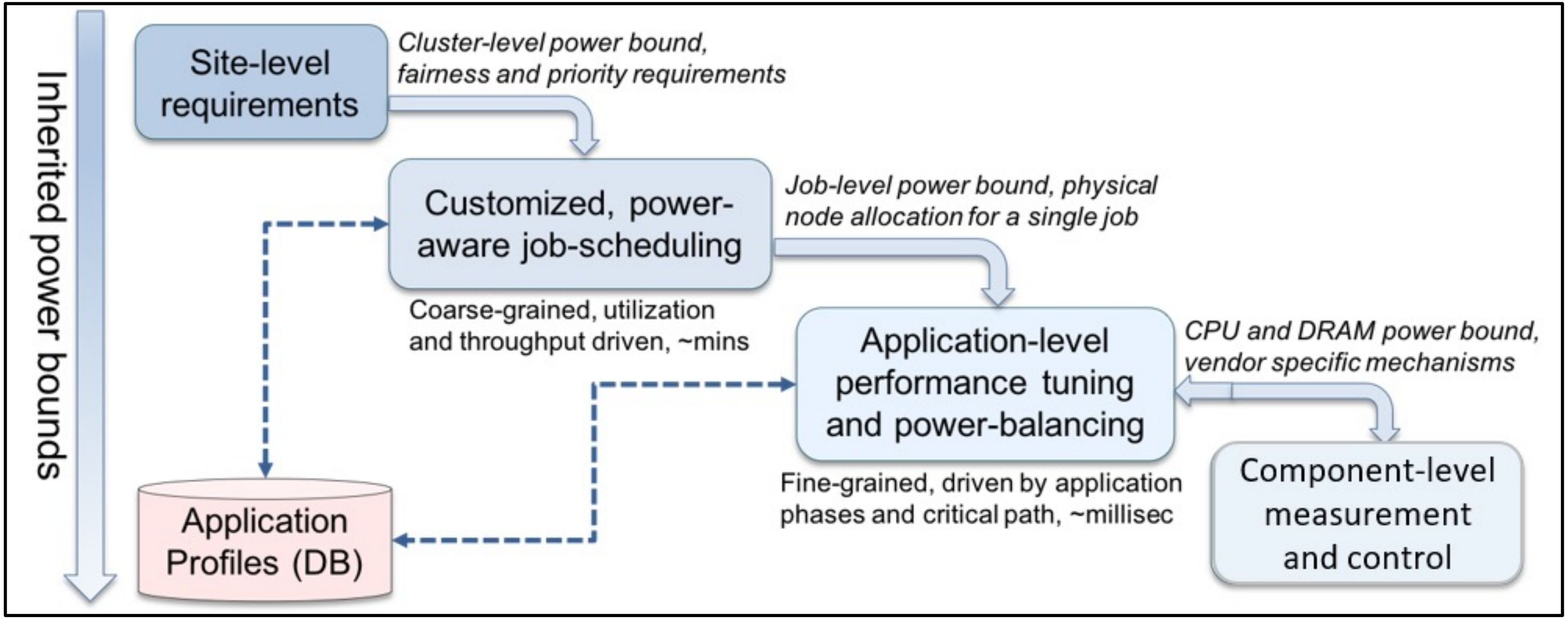
https://variorum.readthedocs.io/

Lawrence Livermore National Laboratory

Module 2 Agenda

- Recap module 1, revisit PowerStack and JSON API (15 minutes)
- Job-level power management: GEOPM (35 minutes)
- System-level power management:
  - SLURM (5 minutes)
  - Flux (20 minutes)
- Application and workflow power management: Kokkos and Caliper (5 minutes)
- Upcoming features in Variorum (5 minutes)
- Wrap up (5 minutes)

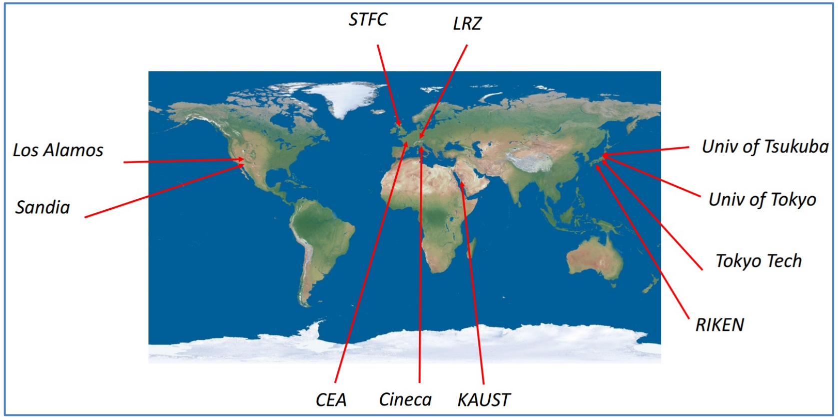
HPC PowerStack: Community Effort on System-wide, dynamic power management



https://hpcpowerstack.github.io/

PowerStack: Stakeholders

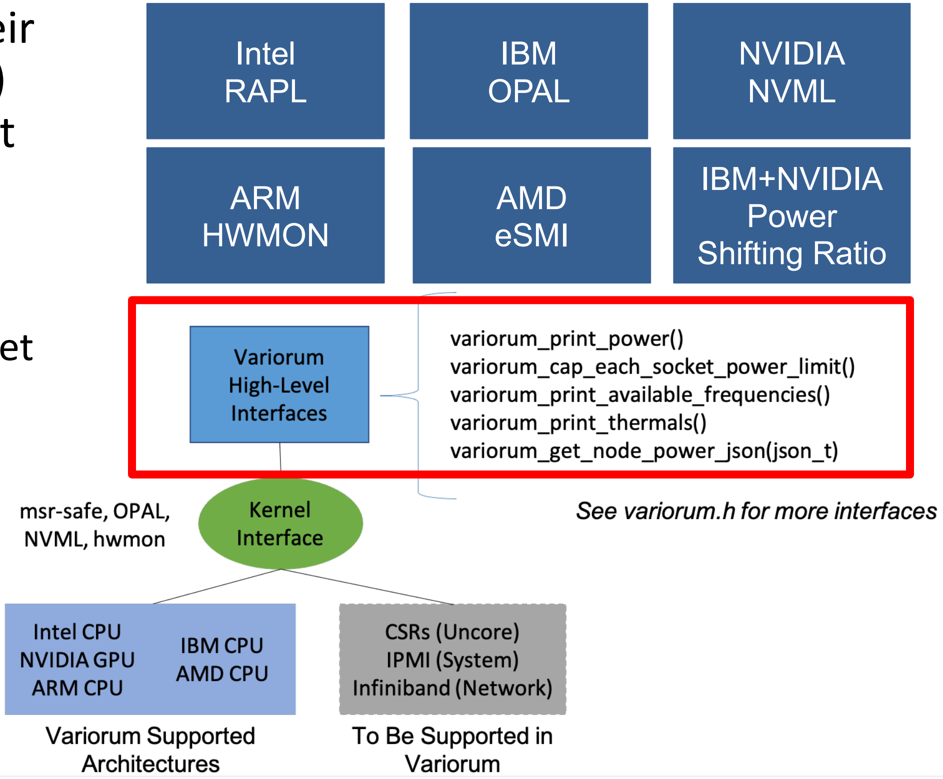
- Current industry collaborators: Intel, IBM, AMD, ARM, NVIDIA, Cray/HPE, Fujitsu, Altair, ATOS/Bull, and PowerAPI community standard
- Multiple academic and research collaborators across Europe, Asia, US
- Three working groups established
- Dynamic power management at all levels, along with prioritization of the critical path, application performance and throughput
- One of the prototypes developed as part of ECP using SLURM, GEOPM, Variorum/msr-safe (close collaboration with Intel)
- Additional software with Flux and Variorum underway



EEHPC-WG's insight into sites investing in Energy- and Power-aware Job Scheduling and Resource Management (EPA-JSRM)

Variorum: Vendor-neutral user space library for power management

- Power management capabilities (and their interfaces, domains, latency, capabilities) widely differ from one vendor to the next
- Variorum: Platform-agnostic vendor-neutral, simple front-facing APIs
  - Evolved from *libmsr*, and designed to target several platforms and architectures
  - Abstract away tedious and chaotic details of low-level knobs
  - Implemented in C, with function pointers to specific target architecture
  - Integration with higher-level power management software through JSON



https://variorum.readthedocs.io/

Variorum Current Support (as of v0.4.1)

- Initial v0.1.0 released Nov 11, 2019
  - Platforms and microarchitectures supported:
    - Intel: Kaby Lake, Skylake, Broadwell, Haswell, Ivy Bridge, Sandy Bridge
    - IBM: Power9
- Current release (April 2021), v0.4.1:
  - Platforms and microarchitectures supported:
    - Nvidia: Volta
    - ARM: Juno
    - AMD (under review)
  - JSON API to integrate with external tools (e.g., Kokkos, Caliper, GEOPM, Flux)

https://github.com/llnl/variorum

Adding a vendor-neutral JSON interface

- Many of Variorum's APIs are printing output to stdout for user to parse
  - While nice for providing a friendly interface to understanding the hardware-level metrics, this limits ability for Variorum to provide these metrics to an external tool

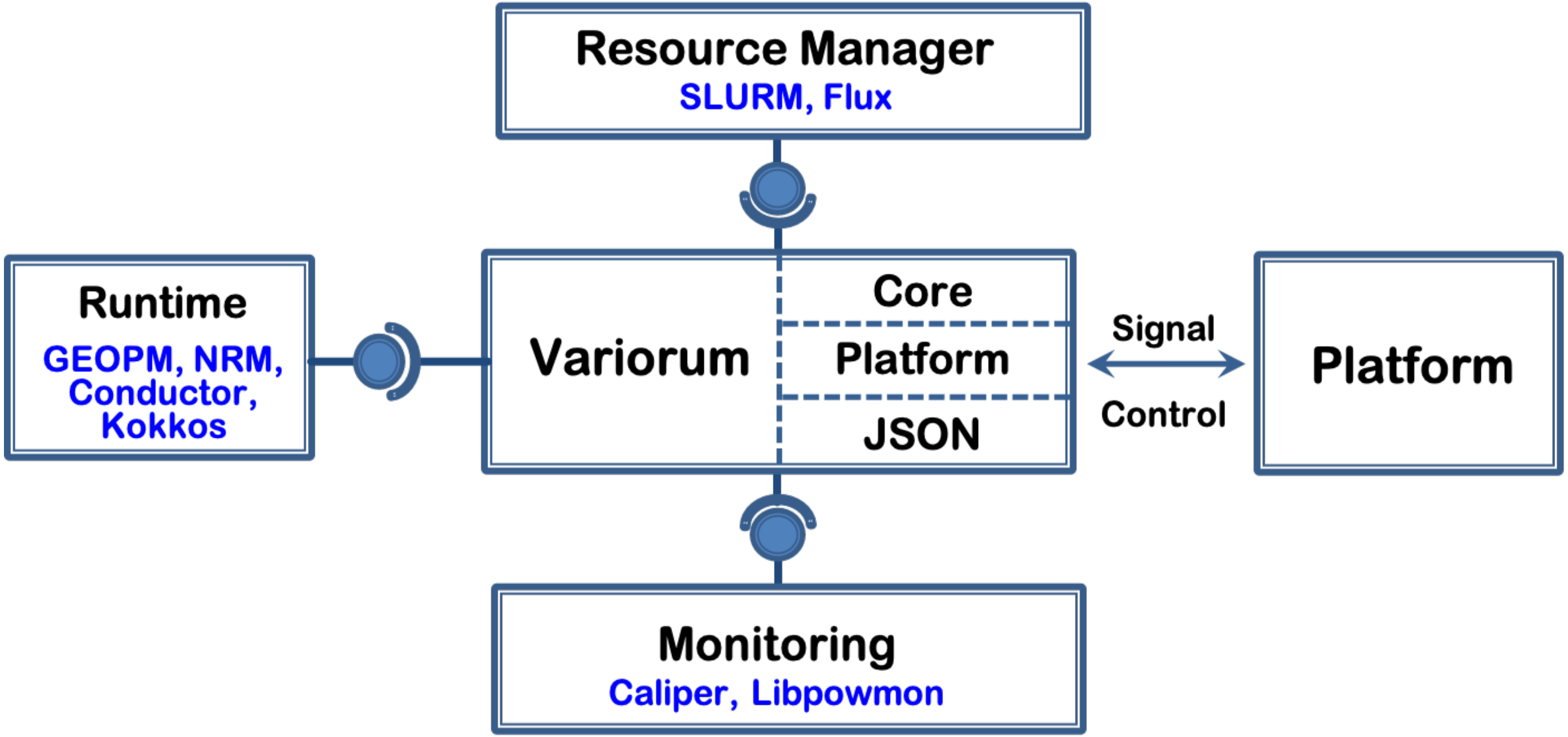
- Added int variorum_get_node_power_json(json_t *) to integrate variorum with other tools (*e.g.*, Flux and Kokkos)
  - { "hostname": (string),
  - "timestamp": (int),
  - "power_node": (int),
  - "power_cpu_socket_<id>": (int)
  - "power_mem_socket_<id>": (int)
  - "power_gpu_socket_<id>": (int) }

  JSON object keys

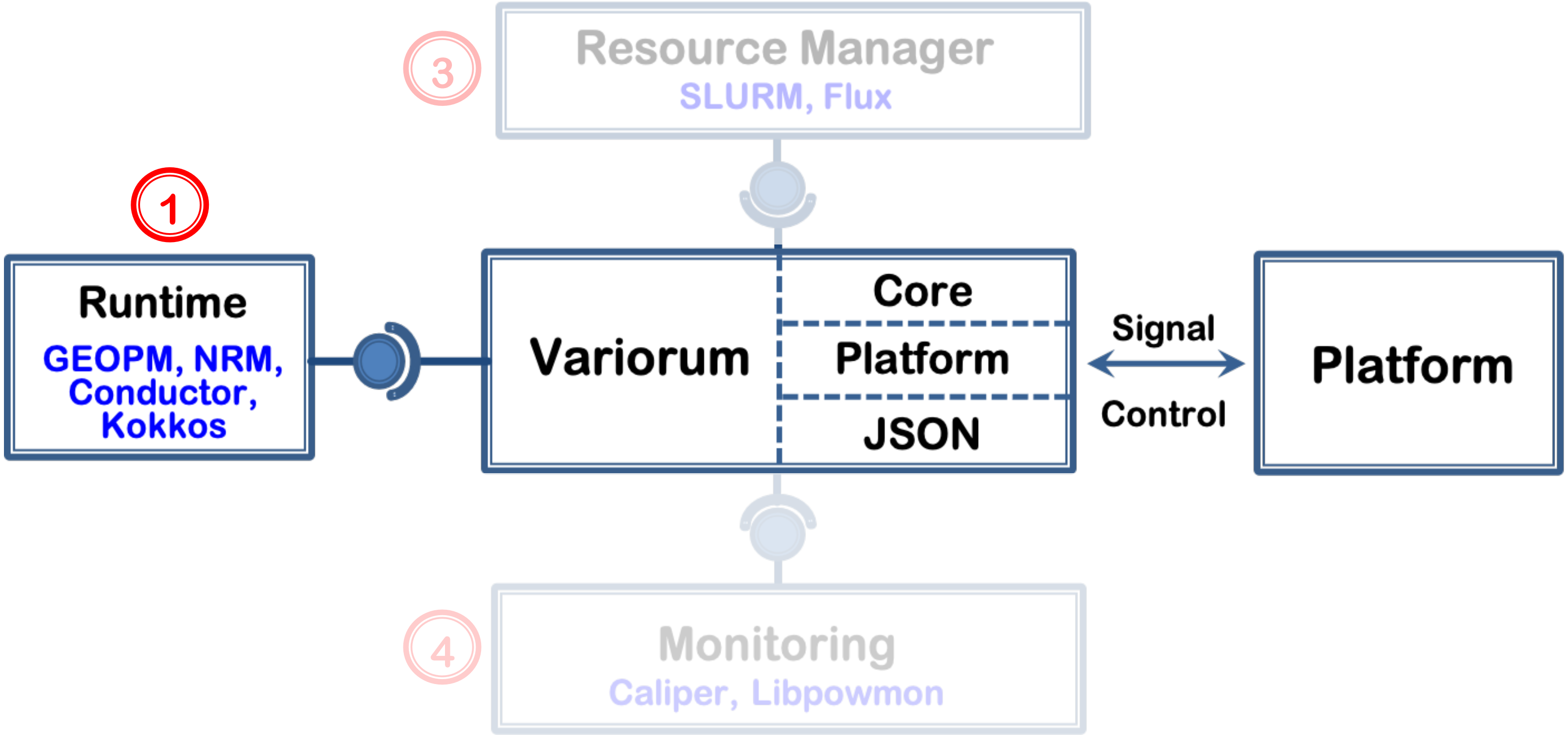
- Example: Reporting end-to-end power usage for Kokkos loops

- Example: Provide power-awareness to Flux scheduling model enabling resources to be assigned based on available power

Interfacing Variorum with PowerStack Components



Interfacing Variorum with PowerStack Components



Collaborations

GEOPM Core Team (Intel)

Jonathan Eastep (Project Lead)
Chris Cantalupo (Lead Developer)
Fede Ardanaz
Brad Geltz
Brandon Baker
Mohammad Ali
Siddhartha Jana
Diana Guttman

ANL Team

Pete Beckman
Kamil Iskra
Swann Perarnau
Florence Monna
Kazutomo Yoshii

LLNL Team

Aniruddha Marathe
Tapasya Patki
Stephanie Brink
Barry Rountree

Agenda: Integration with Runtime Systems

- Part I: Overview of GEOPM (5 minutes)
  - High-level design
  - User-facing, application-context markup API
- Part II: Plug-ins to extend GEOPM algorithm and platform support (10 minutes)
  - **Agent**: Run-time tuning extension
  - **PlatformIO**: Platform-specific support extension
  - Demonstrations (5 minutes)
- Part III: ECP Argo Contributions (10 minutes)
  - **VariorumIO:** Variorum plugin for GEOPM
  - **NRM integration:** Decentralizing job-level power management
  - **ConductorAgent**: Transparent, performance-optimizing configuration selection
  - **IBM PlatformIO plugin**: Port of GEOPM to IBM Power9 platform

Power-Constrained Performance-Optimization Problem

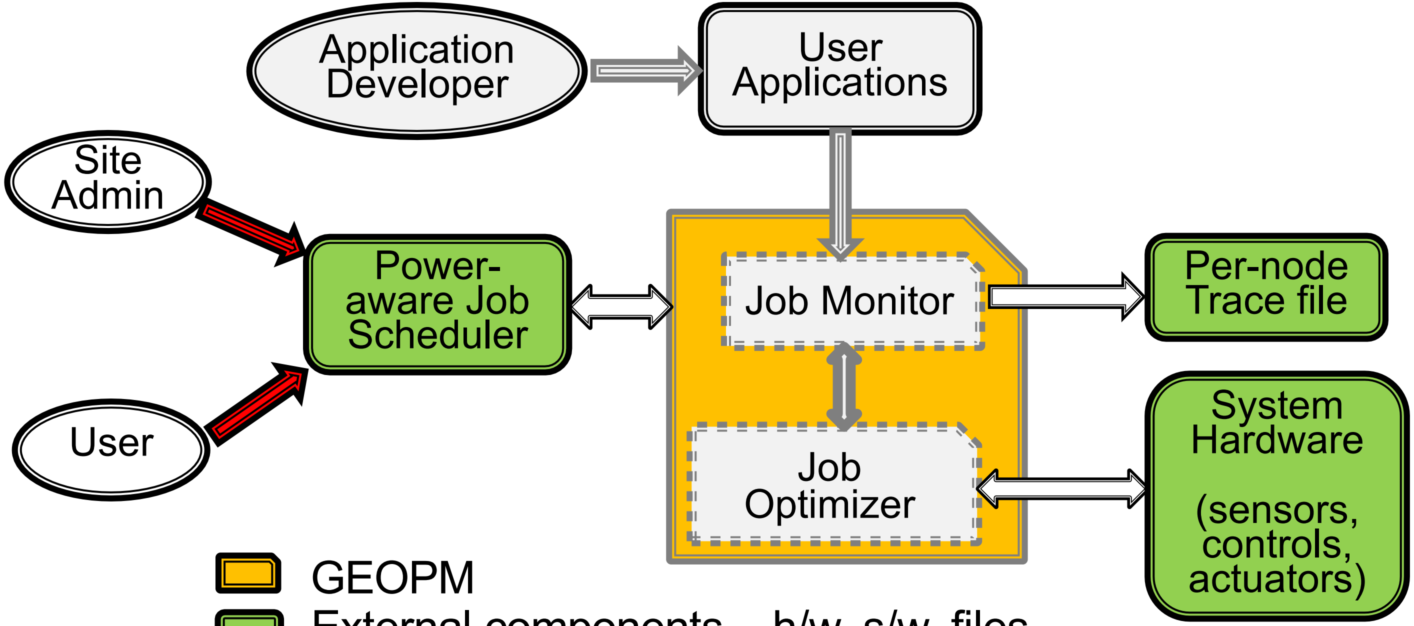
Problem definition

Given a job-level power constraint and number of nodes, how do we optimize application performance?

GEOPM: Global Extensible Open Power Manager

- Power-aware runtime system for large-scale HPC systems
- Intel developed a production-grade, scalable, open-source job-level extensible runtime and framework
  - Extensibility through plug-ins + advanced default functionality
- Limitations of existing runtimes
  - Research-based codes addressed specific needs and situations
  - Ad-hoc, targeted specific architecture, memory model
  - Suffered scalability issues
  - Reliance on empirical data
- Funded through a contract with Argonne National Laboratory

GEOPM System Model



GEOPM
External components – h/w, s/w, files
Extensible / Plug-in components
Human actors

Background: System Software Stack for Power Management

Inherited Power Bounds

Site
- Demand Response, Renewables

Cluster
- Overprovisioning, Job scheduling

Job/Application
- Adaptive runtimes, Power balancing

Node
- Measurement and control (capping)

Software
Dashboards
RMAP, P-SLURM, PowSched
GEOPM, Conductor, Kokkos,...
Libmsr, msr-safe

Background: System Software Stack for Power Management

Inherited Power Bounds

Site
- Demand Response, Renewables

Cluster
- Overprovisioning, Job scheduling

Job/Application
- Adaptive runtimes, Power balancing

Node
- Measurement and control (capping)

- **Critical contribution to the development of HPC power-aware system software stack.**

Software
Dashboards
RMAP, P-SLURM, PowSched
GEOPM, Conductor, Kokkos,...
Libmsr, msr-safe

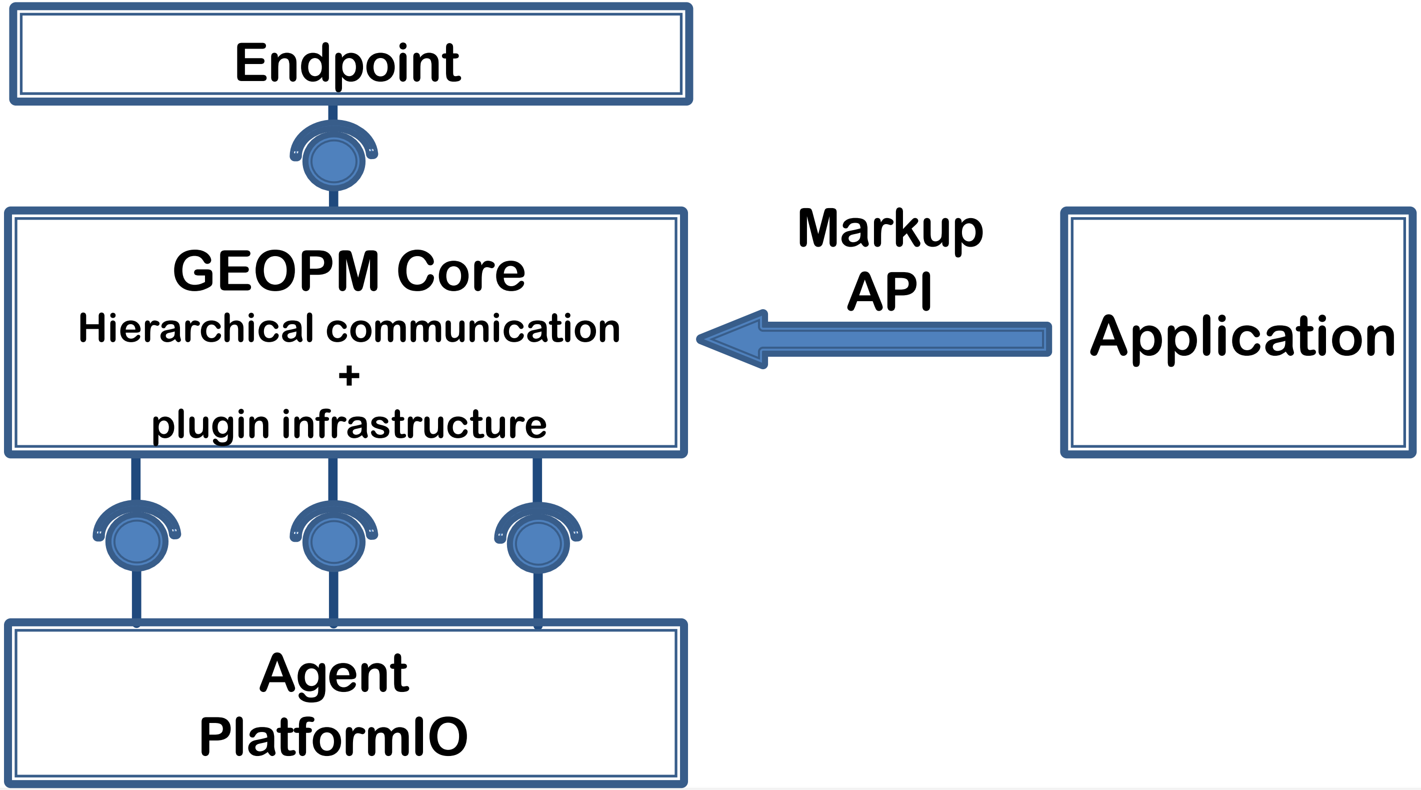
GEOPM Project Goals

- **Managing power**
  - Maximizing power efficiency or performance under a power cap
- **Managing manufacturing variation**
  - Power / frequency relationship is non-uniform across different processors of same type
- **Managing workload imbalance**
  - Divert power to CPUs with more work
- **Managing system jitter**
  - Divert power to CPUs interrupted or stalled by system noise
- **Application profiling**
  - Report application performance and power metrics
- **Runtime application tuning**
  - Extensible runtime control agent with plug-in architecture
- **Integration with MPI**
  - Automatic integration with MPI runtime through PMPI interface
- **Integration with OpenMP**
  - Automatic integration with OpenMP through OMPT interface

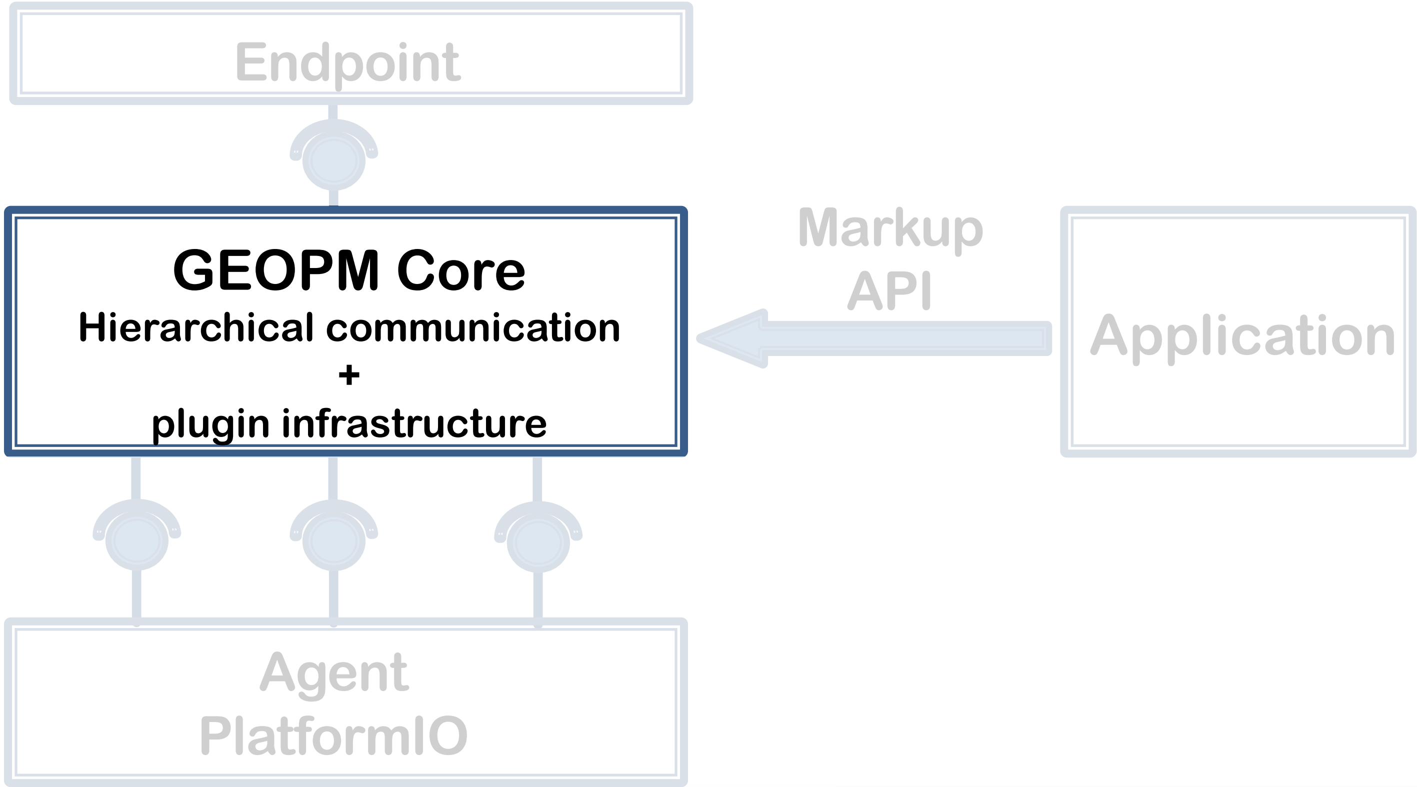
GEOPM: Capabilities

- Enables analysis and transparent tuning of distributed-memory applications
- **Feedback-guided optimization**: Leverages lightweight application profiling
- **Learns application phase patterns**: load imbalance across nodes, distinct computational phases within a node
- **Uses tuning parameters**: processor power limit, core frequency, etc.
- **Built-in optimization algorithms**: Static Power capping, energy reduction, load balancing, limiting synchronization costs

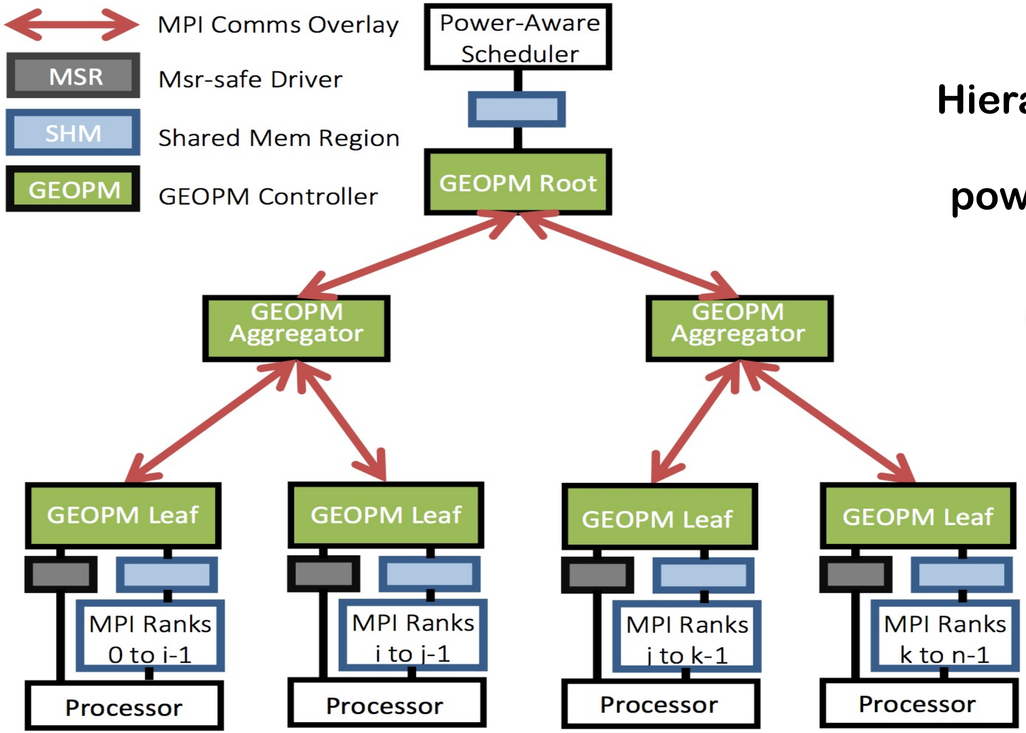
GEOPM Components of Interest



GEOPM Components of Interest



GEOPM Infrastructure

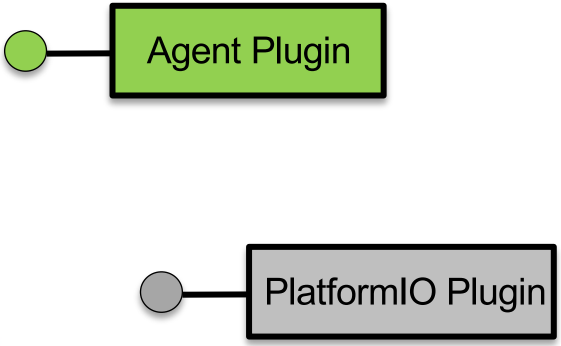


GEOPM Core

Hierarchical communication

+

power-management plugin



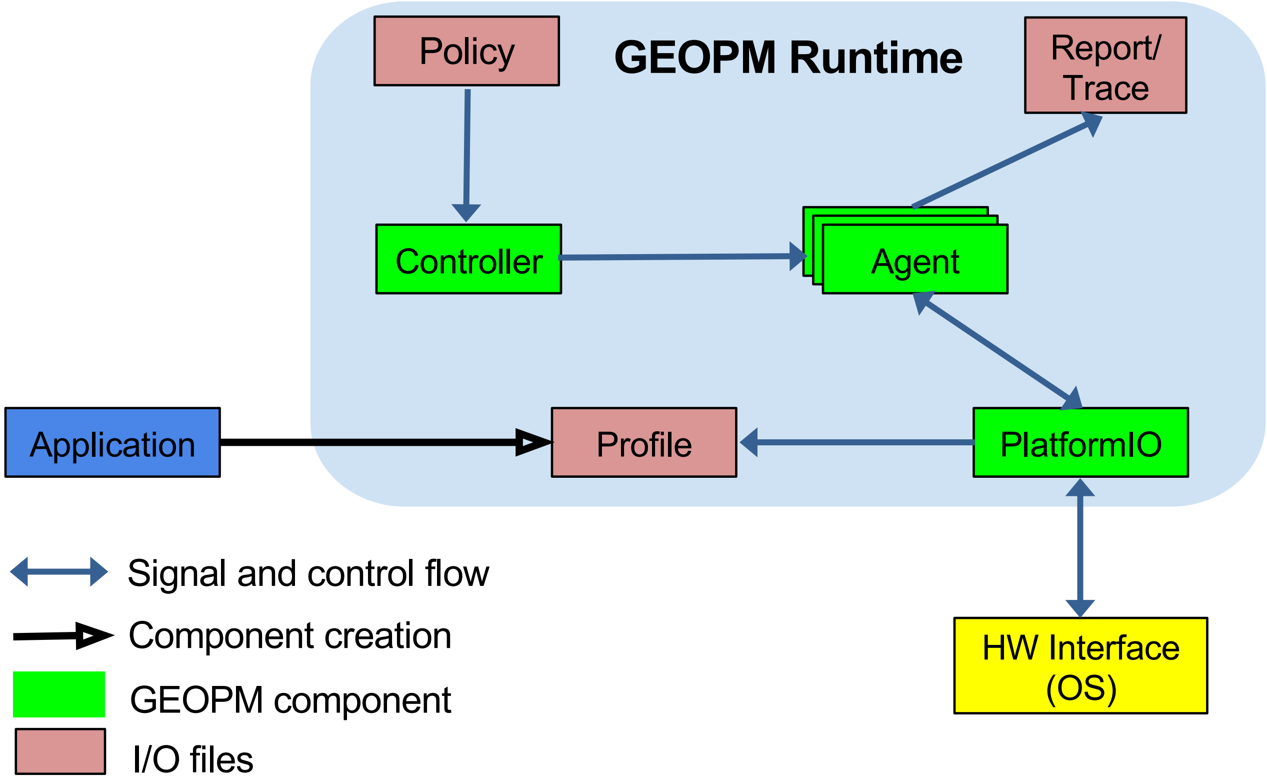
GEOPM Infrastructure

- GEOPM Source repository navigation
  - Branches, directories, releases
  - GEOPM Wiki

- Build process
  - Dependencies
  - Build configuration

- GEOPM core infrastructure source
  - Overview of important classes
  - Plug-in source
  - Tutorials and examples
  - Test coverage

GEOPM: Input/Output Files



GEOPM Configuration, Build and Launch

Building an Application with GEOPM

Step 1 : Set the environment

```
$> module load geopm
$> module load <intel compiler>
$> module load <MPI compiled with intel-c>
```

Step 2: Link the Application to GEOPM library

```
$> mpicc  APP_SRC.c -L$GEOPM_LIB -lgeopm \
          -o APP_EXEC \
          COMPILER_FLAGS
```

Example

```
$> mpicc helloworld.c -L$GEOPM_LIB -lgeopm -o a.out
```

Running an Application with GEOPM

Step 3: Generate a policy file

```
$> geopmagent --agent=AGENT_NAME --policy=INPUT_PARAMS > POLICY_FILE.json
```

Example:

```
$> geopmagent --agent=monitor --policy=None > monitor_policy.json
```

Step 4: Launch application with GEOPM launcher wrapper

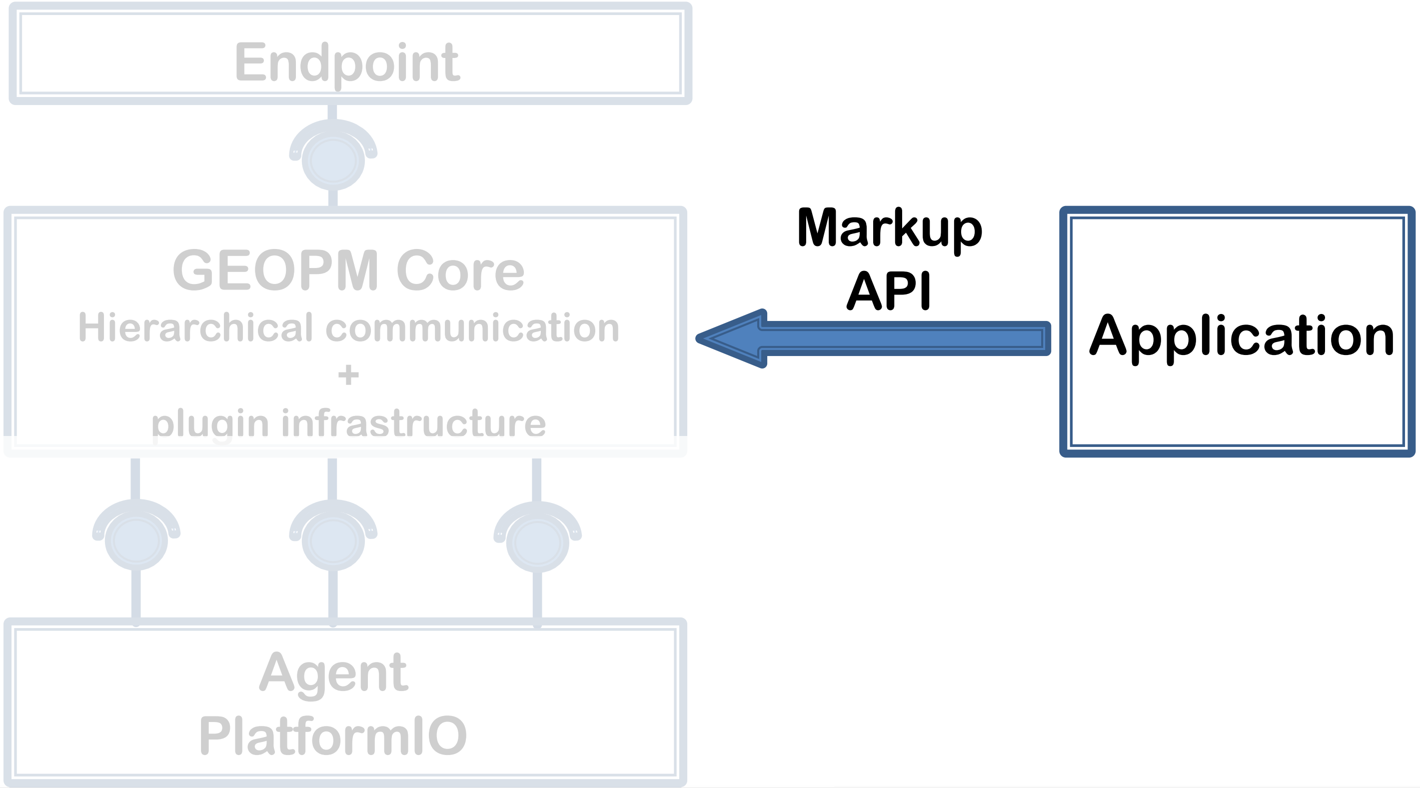
```
$> geopmlaunch srun -n < > -N < >\
        --geopm-ctl=process \
        --geopm-agent=AGENT_NAME \
        --geopm-policy=POLICY_FILE.json \
        --geopm-report=REPORT_FILE.txt \
        --geopm-trace=TRACE_FILE.csv \
        --    APP_EXEC   APP_OPTIONS
```

Example:

```
$> geopmlaunch srun -n 4 -N 1 \
        --geopm-ctl=process \
        --geopm-agent=monitor \
        --geopm-policy=monitor_policy.json \
        --geopm-report=report.txt \
        --geopm-trace=trace.csv  \
        --    a.out
```

Demo: Running Application with GEOPM

GEOPM Components of Interest



GEOPM: Components and Interfaces

Collecting Application Context

- Application region markup API
  - Computation/communication regions of interest
- Epoch
  - End of iteration
- OpenMP event callbacks

Power Assignment Policies

- Governed policy
  - Node-level assignment
- Balanced policy
  - Cluster-level assignment

Extension Interfaces

- New Agent plugin: ConductorAgent
- New PlatformIO plugin: IBM port of GEOPM

GEOPM Markup API: Purpose

- C interfaces provided in GEOPM that the application links against
  - Resemble typical profiler interfaces
- Annotation functions for programmers to provide information about application critical path and phases to GEOPM
  - Points where bulk synchronizations occur
  - Phase changes occur in an MPI rank (i.e. phase entry and exit)
  - Hints on whether phases will be compute-,memory-, or communication-intensive
  - How much progress each MPI rank has made in the phase (critical path)

Application Markup API

MPI/Sequential Region

- Marking up regions of interest
  - geopm_prof_region(name, hint, ID)
  - geopm_prof_enter(ID)
  - geopm_prof_exit(ID)
- Marking region progress
  - geopm_prof_progress(ID, %progress)
- Marking a timestep
  - geopm_prof_epoch()

OpenMP Region

- Marking up regions of interest
  - geopm_tprof_init(num_work_unit)
  - geopm_tprof_init_loop(num_thread, thread ID, num_iter, chunk_size)
- Marking region progress
  - geopm_tprof_post()

Demo: Using the GEOPM Markup API

Part II: Plug-ins to extend GEOPM algorithm and platform support

GEOPM: Policy plugins

Collecting Application Context

- Application region markup API
  - Computation/communication regions of interest
- Epoch
  - End of iteration
- OpenMP event callbacks

Power Assignment Policies

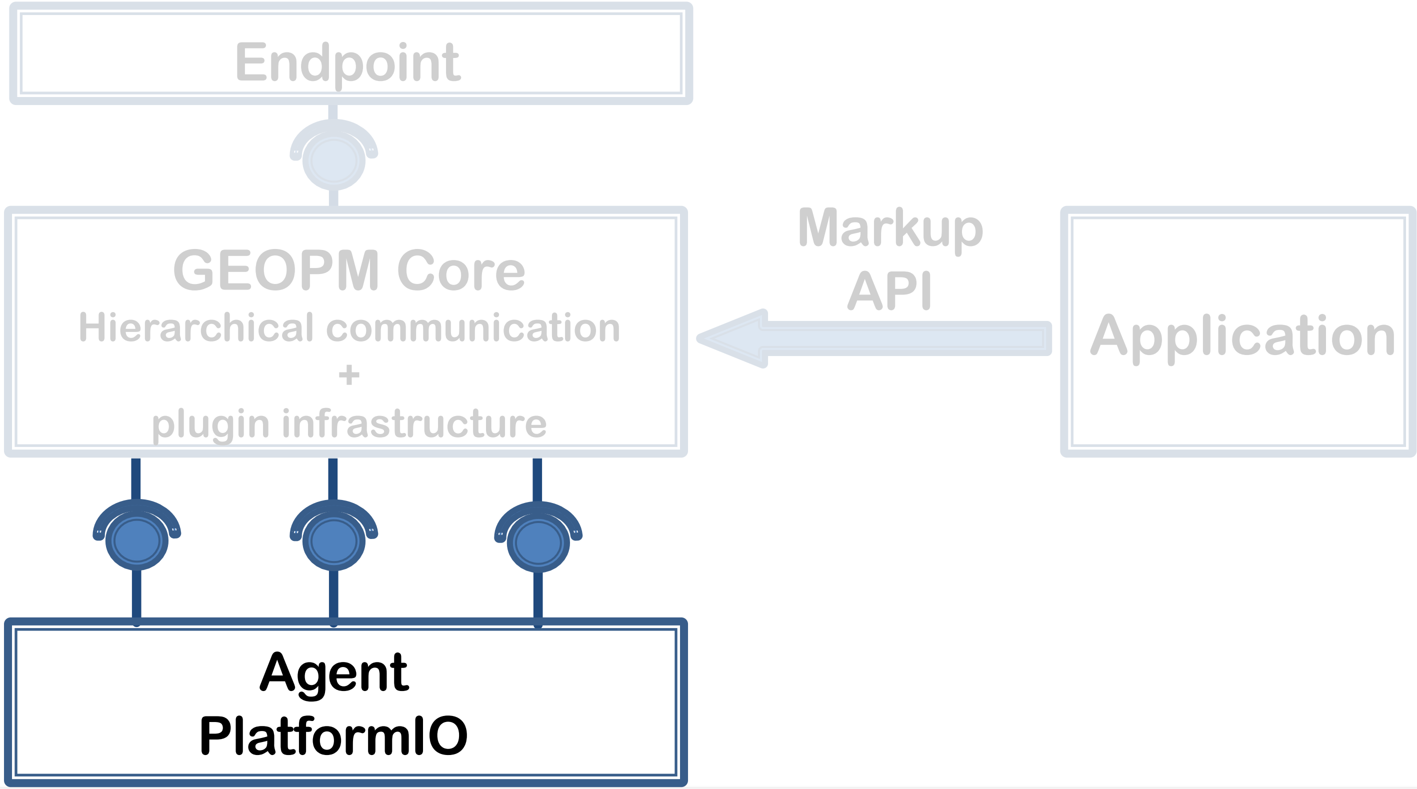
- Governed policy
  - Node-level assignment
- Balanced policy
  - Cluster-level assignment

Extension Interfaces

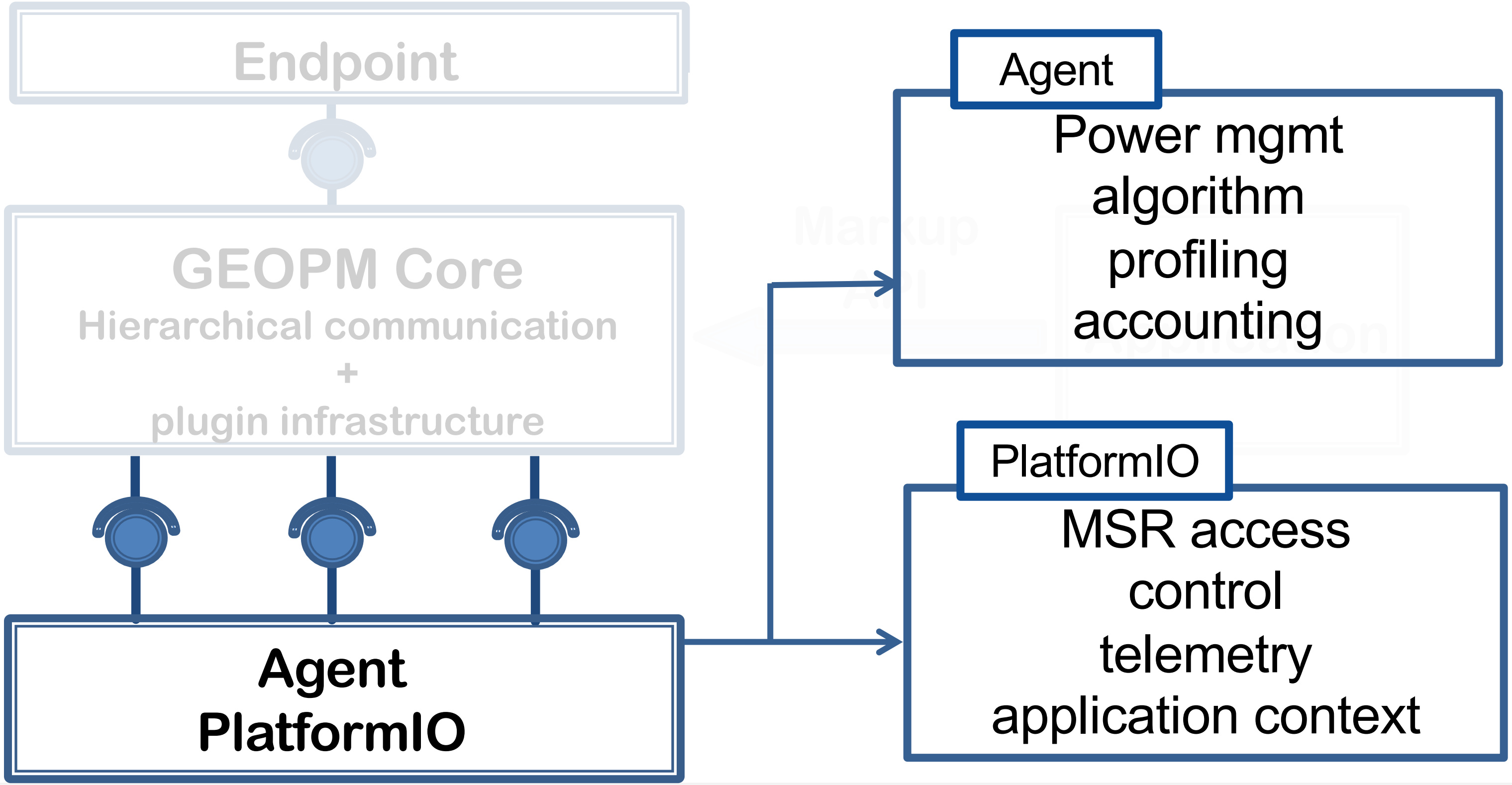
- New Agent plugin: ConductorAgent
- New PlatformIO plugin: IBM port of GEOPM

Demo: Using the Default GEOPM Policies

GEOPM Components of Interest



GEOPM Components of Interest



GEOPM Plugin Interface

- Two types of plugins: PlatformIO and Agent plugins
- Example **Agent** plugins
  - MonitorAgent
  - BalancerAgent
  - GoverningAgent

- Example **PlatformIO** plugins
  - VariorumIOGroup

- Tutorial plugins: ExampleAgent and ExampleIOGroup
  - Key methods and code blocks
  - Policy description interface

VariorumIO: Interfacing GEOPM with Variorum for Vendor Neutrality

- **Motivation**: GEOPM uses platform-specific interfaces for signals and controls on the target architecture
  - A PlatformIO plug-in interfacing with Variorum as the vendor-neutral lower-level API
- **Components**
  - VariorumIO plugin to map GEOPM-specific data structures to Variorum
  - Low-level API in Variorum to aggregate low-level signals and pass to GEOPM
- **Challenge:** Translate vendor-specific into vendor-agnostic signals and controls
- **On-going work:**
  - Integration with JSON API for capability query
  - Evaluation on several platforms

VariorumIO: Contributions to GEOPM and Variorum

- GEOPM: Added VariorumIO
- Code contributions:

https://github.com/amarathe84/geopm/pull/1

- Supported version: GEOPM v1.1

- Variorum: Added low-level API to aggregate platform signals and controls
- Code contributions:

https://github.com/LLNL/variorum/pull/126

- Supported version: Variorum v0.4.0

ConductorAgent: Selecting Power-Optimizing Configuration

- **Approach:** Hardware Overprovisioning with job-level power guarantees
  - More compute resources than you can power up at once
- **Objective**: Optimize job performance under a power constraint
- **Solution**: GEOPM – power-constrained performance optimization
- **ECP Argo Contributions**:
  - Augment GEOPM's algorithm with performance-optimizing application configurations: # threads, Frequency, etc.
  - Port GEOPM to IBM POWER9 (support for LLNL Sierra)

Extending GEOPM: Components and Interfaces

Collecting Application Context

- Application region markup API
  - Computation/communication regions of interest
- Epoch
  - End of iteration
- OpenMP event callbacks

Power Assignment Policies

- Governed policy
  - Node-level assignment
- Balanced policy
  - Cluster-level assignment

Extension Interfaces

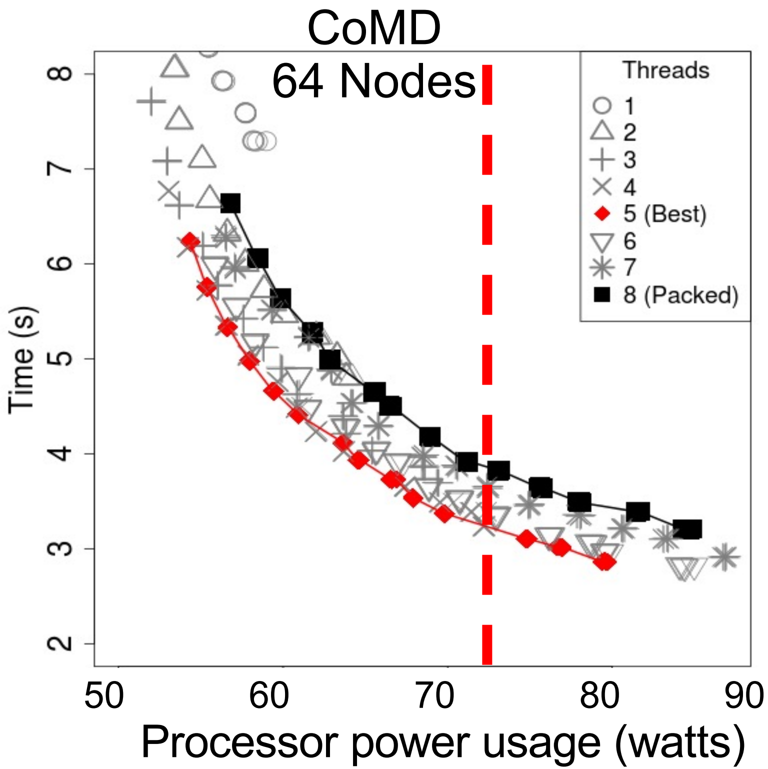
- New policy agent plugin: ConductorAgent
- New PlatformIO plugin: VariorumIO plugin

Naïve Scheme: Static Power Allocation

- Equally distribute and enforce power constraint over all nodes of a job
  - Uses Intel's Running Average Power Limit (RAPL) interface

- Statically select a *configuration* under the power constraint
  - Configuration: {Number of cores, Frequency/power limit}
  - Commonly used: *Packed* configuration
    - Maximum cores possible on the processor
    - Frequency or power limit as the control knob

Limitations of Static Power Allocation



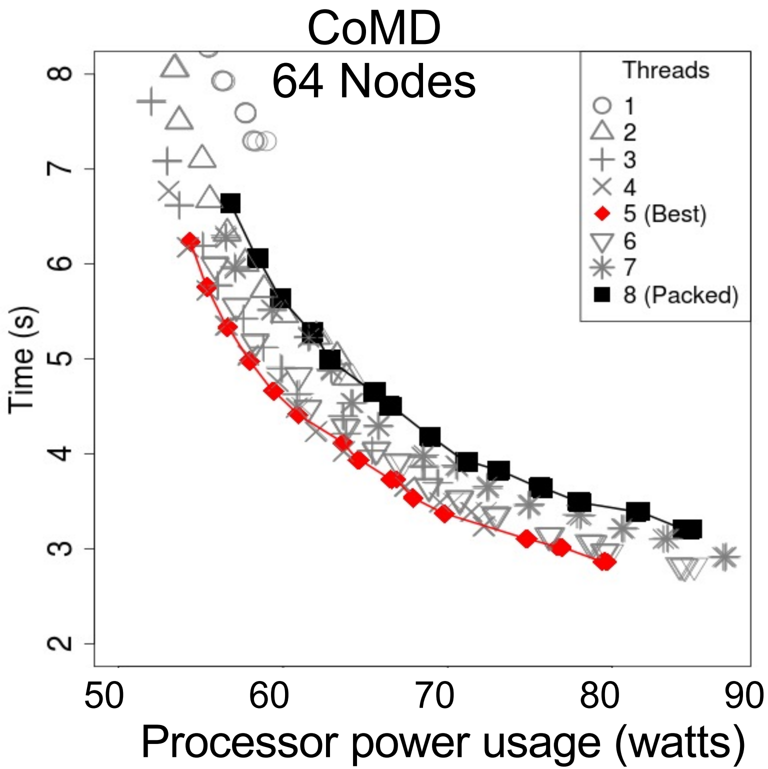
1. Trivial node-level configurations may be inefficient

   Input: {# cores, frequency/power limit}

   Output: {Execution time, power usage}

- Up to 30% slower than the optimal configuration
- Needs prohibitively large number of runs of the application

Limitations of Static Power Allocation



1. Trivial node-level configurations may be inefficient

   Input: {# cores, frequency/power limit}

   Output: {Execution time, power usage}

   - Up to 30% slower than the optimal configuration
   - Needs prohibitively large number of runs of the application

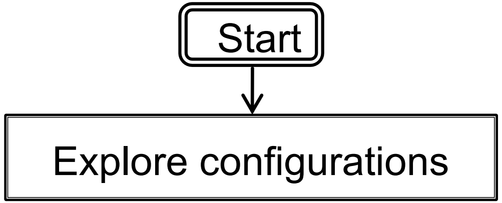
2. Portion of power left unused with load-imbalanced applications (up to 40%)

Conductor: Dynamic Configuration and Power Management

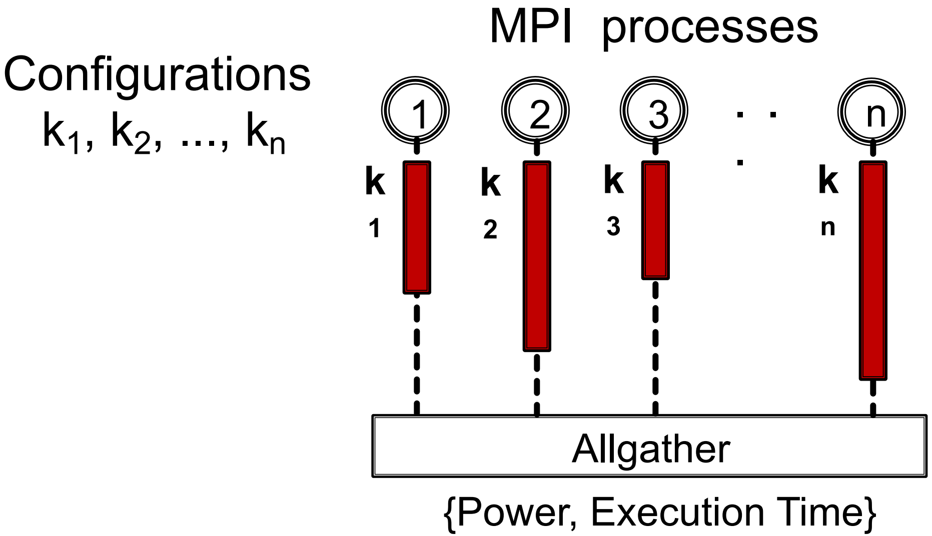
- Goals of ConductorAgent
  - Speed up computation on the critical path
  - Use power-efficient configuration

- Need to *dynamically* identify
  - Computation region potentially on the critical path
  - {execution time, power usage} profile for every computation on every processor

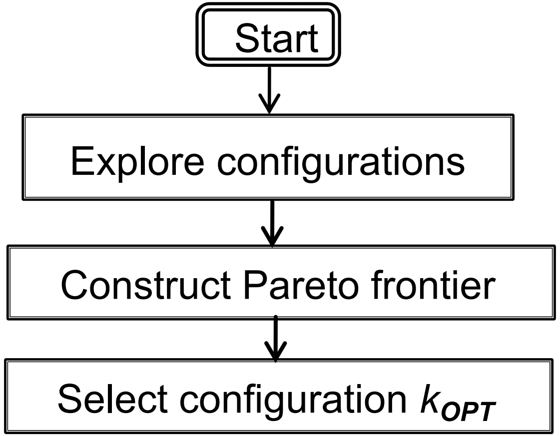
ConductorAgent Algorithm



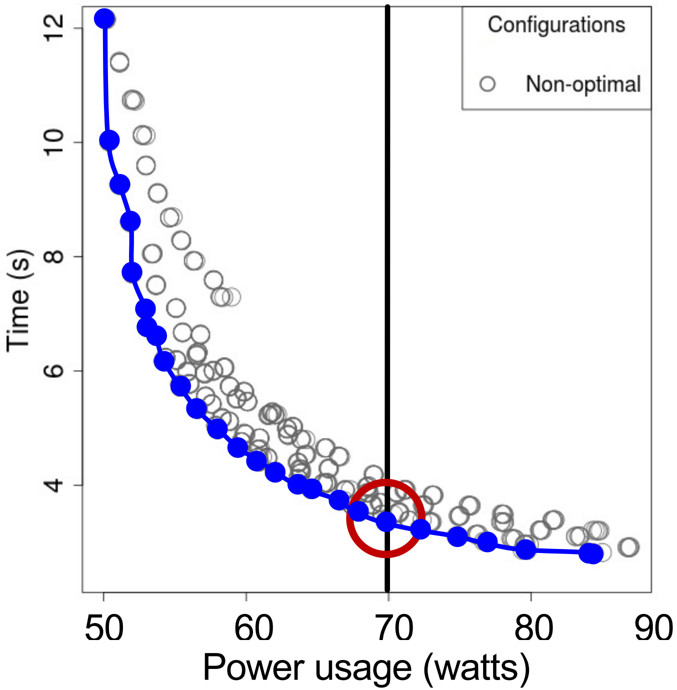
Step 1: Configuration Exploration



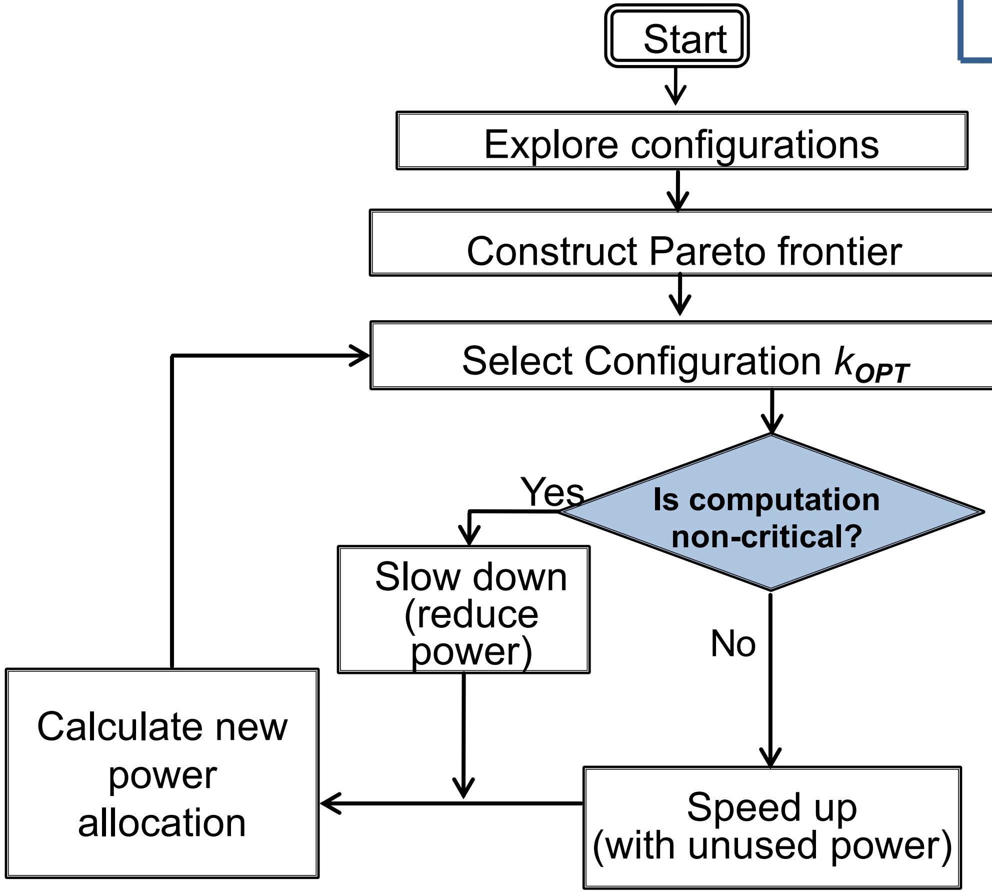
ConductorAgent Algorithm



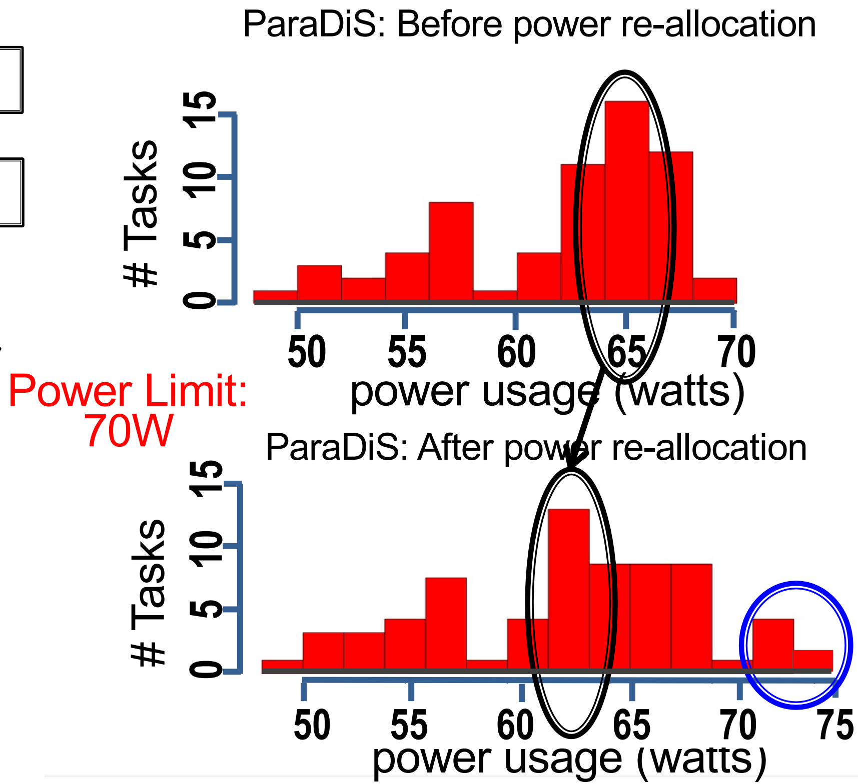
Step 1: Configuration Exploration



ConductorAgent Algorithm



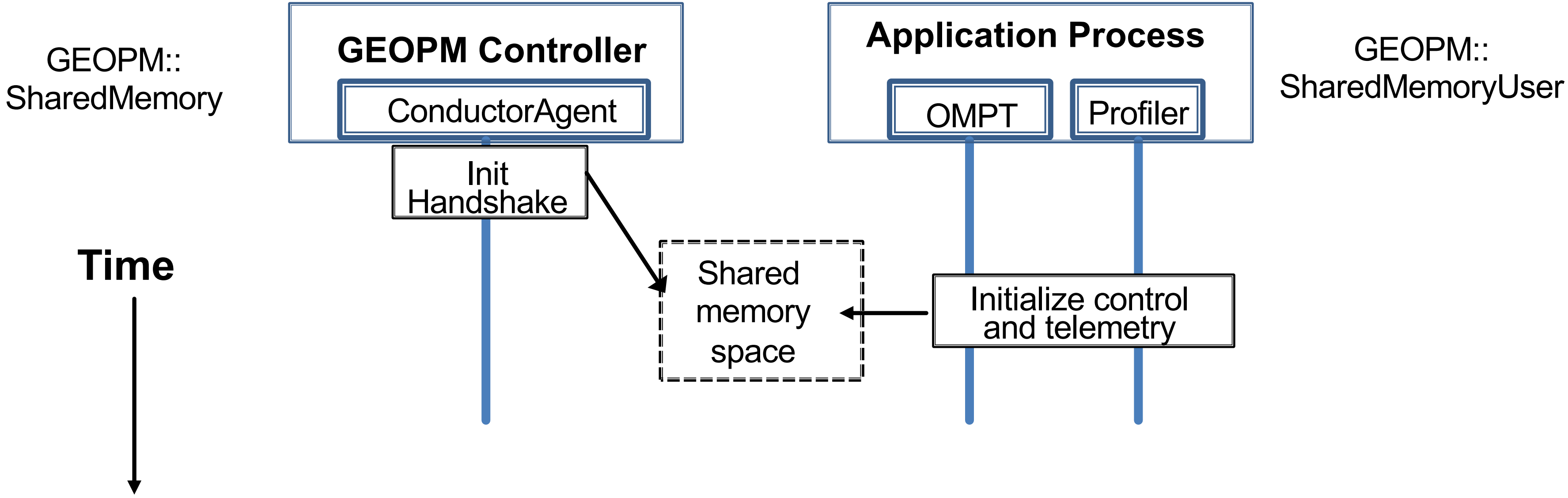
Step 2: Power Re-allocation



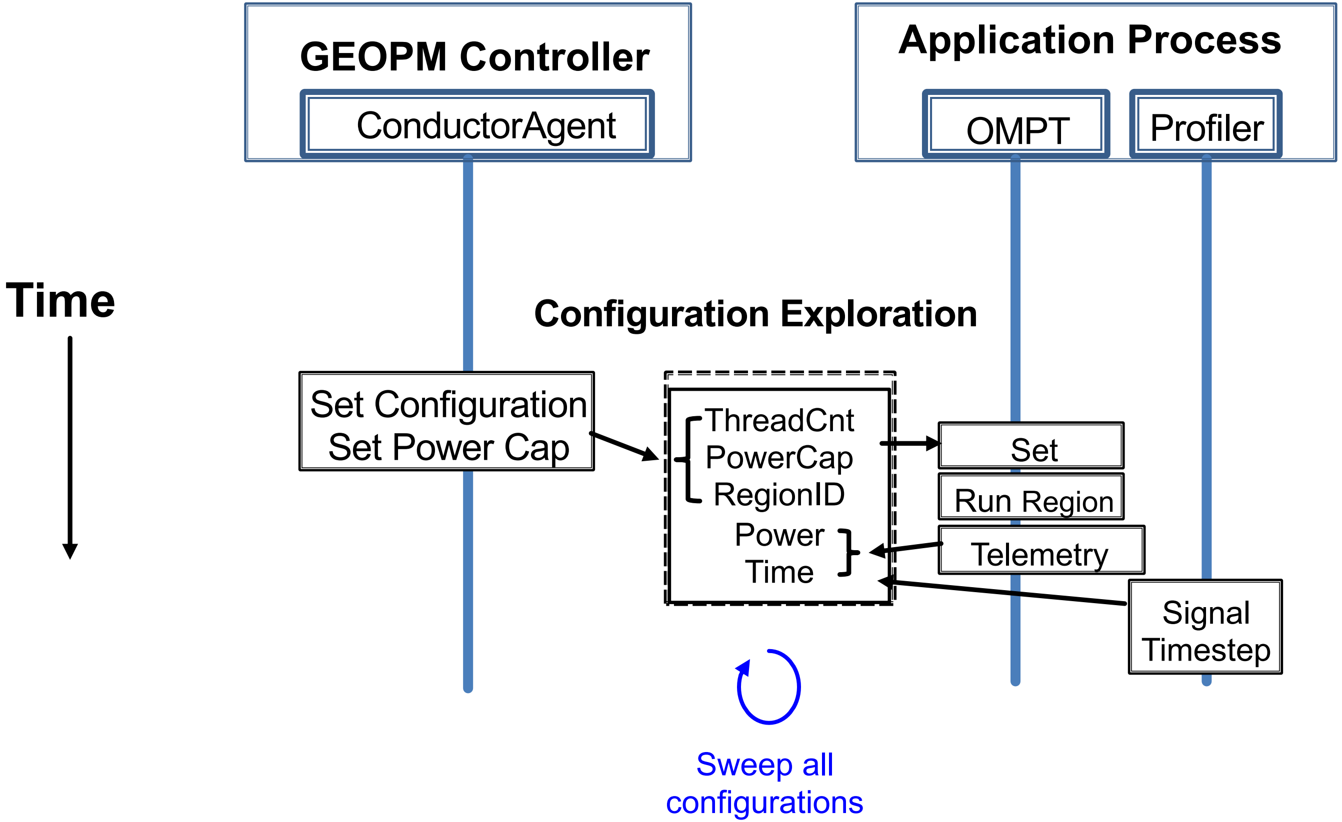
Conductor: Integration into GEOPM with Variorum

- **OMPT class**
  - Explore {OMP, Pcap} configurations during the exploration phase
  - Select power-efficient configuration during regular execution.
- **Profile class**
  - Report end of timestep (i.e., ‘epoch’), application and system telemetry to enable sweep of configuration at runtime.
- **ConfigApp class**
  - Perform profiling, generate pareto-optimal configurations.
- **ConfigAgent class**
  - Share telemetry with PowerBalancer agent, send configuration to OMPT.

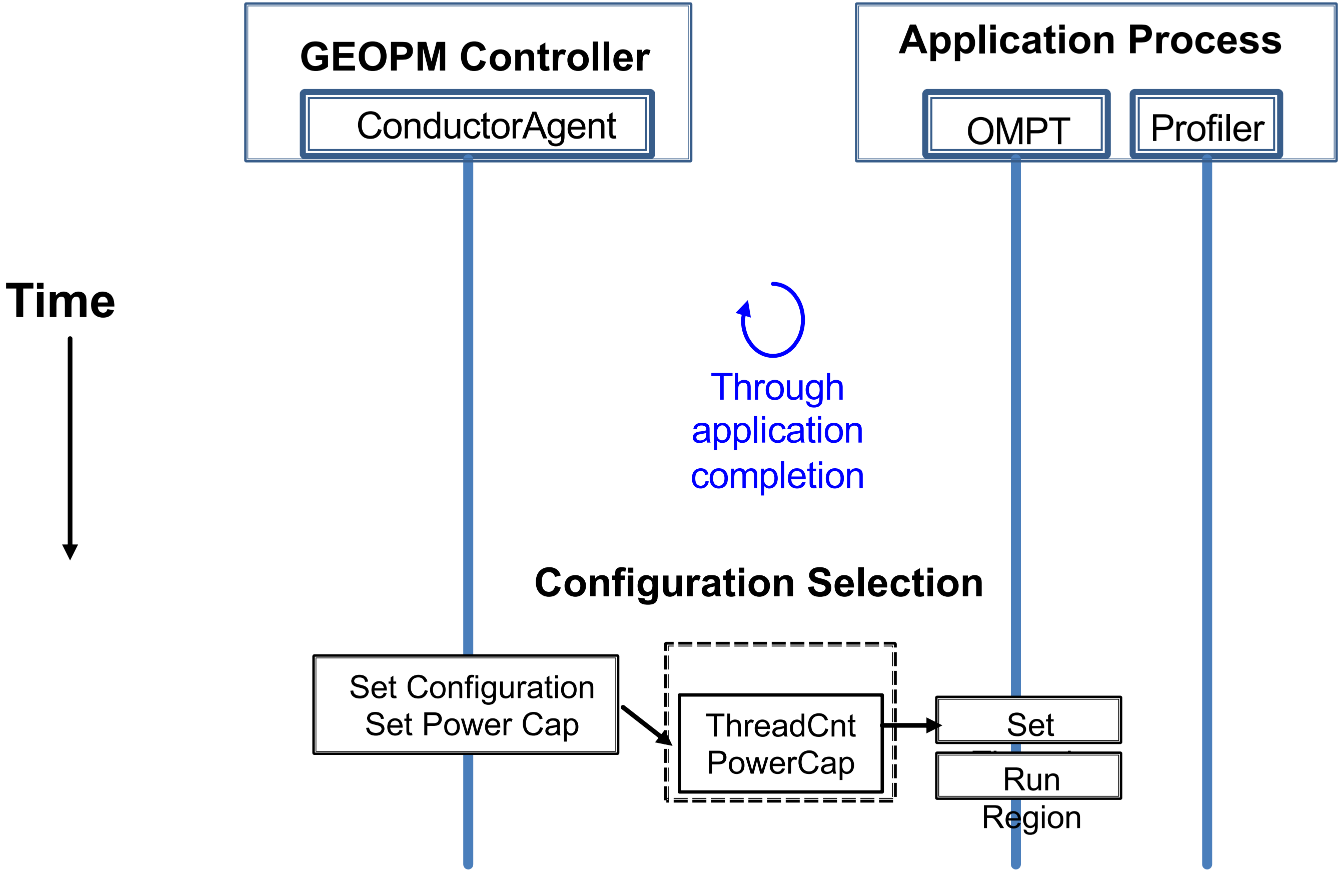
Initialization: GEOPM, Application Handshake



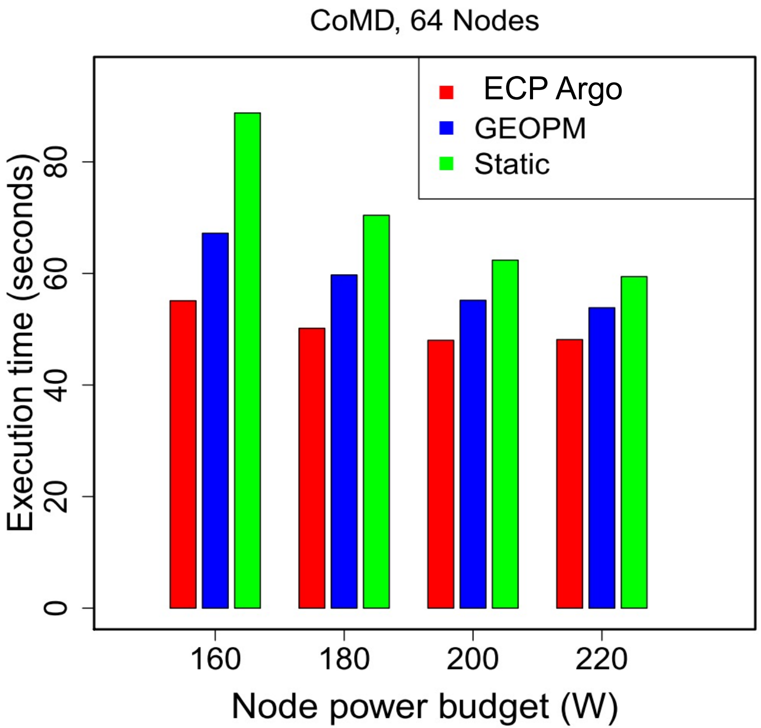
Configuration Exploration: Set Configuration, Collect Telemetry

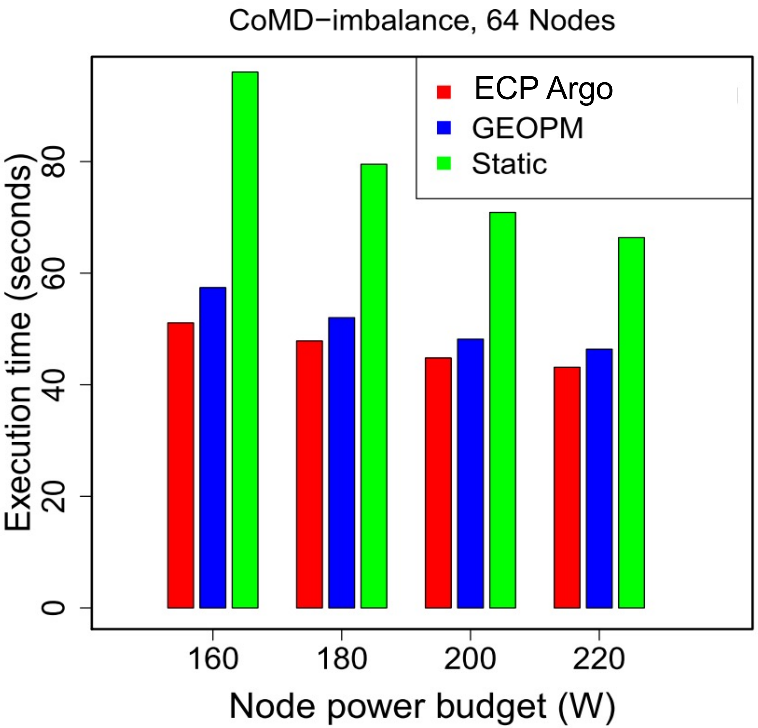


Configuration Selection: Pick Power-Efficient Configurations



Conductor Integration: Results





Conductor Integration: On-going Efforts

- Refresh the Conductor plugin to the latest GEOPM code
- Integration with JSON interface of Variorum
- Conductor integration:
  - https://github.com/geopm/geopm/pull/757
- GEOPM integration with Caliper:
  - https://github.com/LLNL/Caliper/pull/213

Extending GEOPM: Components and Interfaces

Collecting Application Context

- Application region markup API
  - Computation/communication regions of interest
- Epoch
  - End of iteration
- OpenMP event callbacks

Power Assignment Policies

- Governed policy
  - Node-level assignment
- Balanced policy
  - Cluster-level assignment

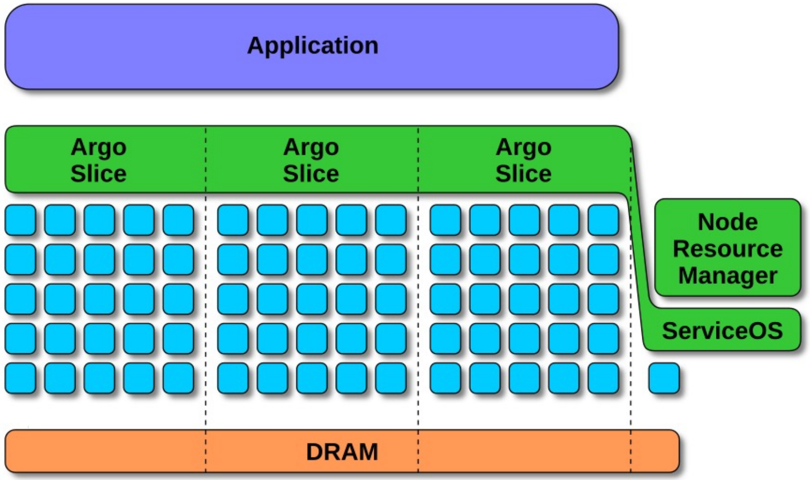
Extension Interfaces

- New policy agent plugin: ConductorAgent
- New PlatformIO plugin: VariorumIO plugin

Part III: Integration of NRM, GEOPM and Variorum

Node Resource Manager (NRM) Integration

- Adaemon running on the compute nodes. It centralizes node management activities
  - job management,
  - resource management, and
  - power management
- Uses slices for resource management
  - Physical resources divided into separate partitions
  - Used to separate individual components of workloads
  - Helps in improved performance isolation between components

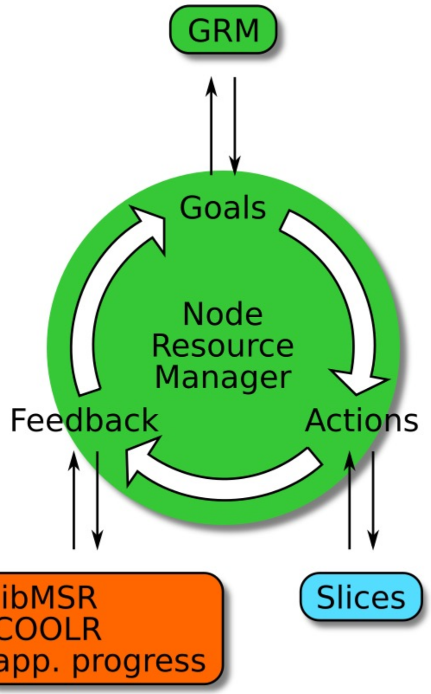


Node Resource Manager (NRM) Integration

- Slices can currently manage the following:
  - CPU cores (hardware threads)
  - Memory (including physical memory at sub-NUMA granularity with a patched Linux kernel)
  - Kernel task scheduling class: The physical resources are partitioned primarily by using the cgroups mechanism of the Linux kernel. Work is under way to extend the management to I/O bandwidth as well as to the partitioning of last-level CPU cache using Intel's Cache Allocation Technology.
- Meant to be transparent to applications
  - do not impede communication between application components,
  - also compatible with (and complementary to) container runtimes such as Docker, Singularity, or Shifter.

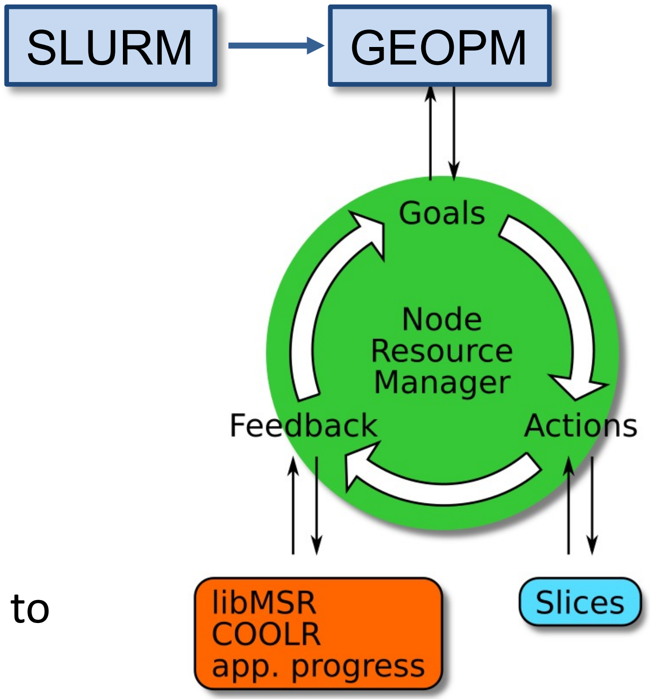
Node Resource Manager (NRM) Integration

- NRM Daemon
  - Manages power at the node level
  - Works in a closed control loop, obtaining goals (power limit) from the higher level entity
  - Acts on application workloads launched within slices by
- NRM Client
  - Launches and manages application runtime
  - Relies on self-reporting by applications
    - Feedback on the efficacy of its power policies,
    - Identification of the critical path

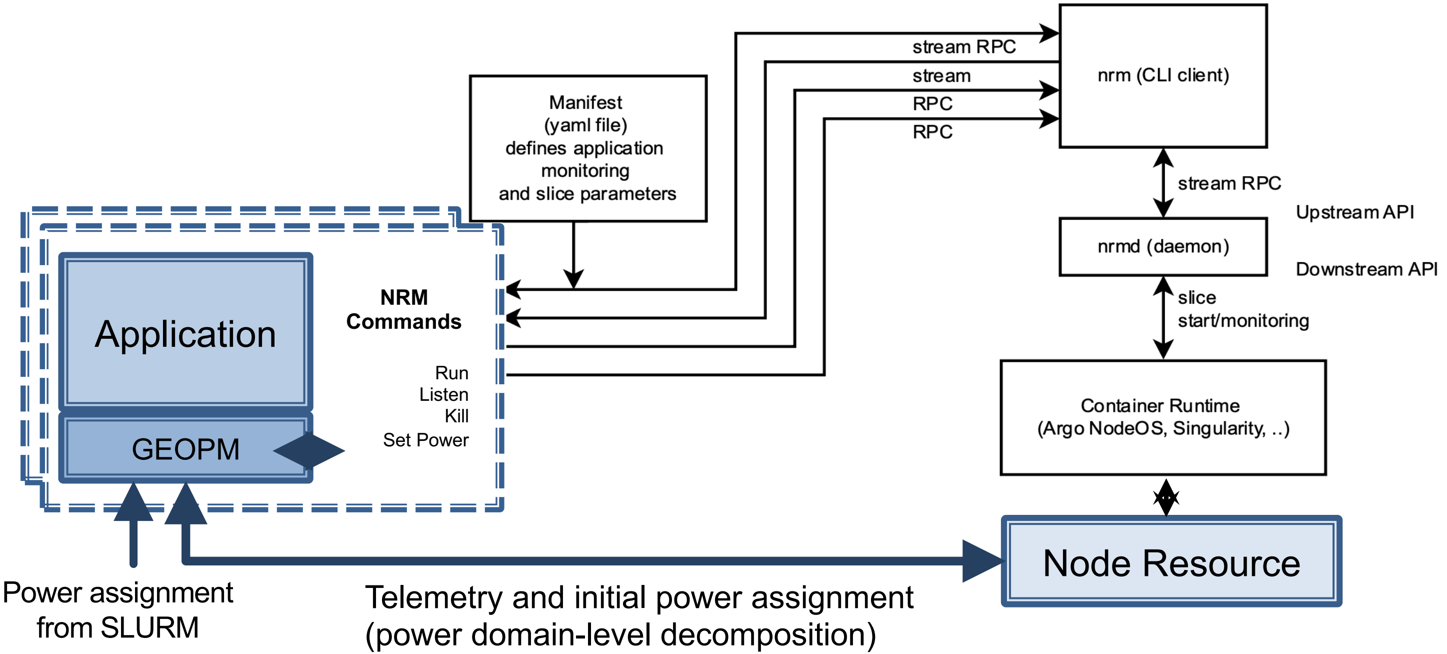


Motivation: NRM and GEOPM Integration

- Hierarchical assignment of power optimization goals along logical and physical boundaries
- Compartmentalization of the power optimization goals enables level-specific goals, for example, improving the time spent on the critical path (IPS) at the job and power efficiency at the node level (IPS/W).
- GEOPM can indirectly support containerized workflows
  - Limitation: power-assignment still at power domain boundaries.
- Leverage NRM's existing integration with ECP applications to include GEOPM and SLURM integration



First Attempt: NRM and GEOPM Integration



- The GEOPM launcher integrates with the NRM launcher to launch the application
  - GEOPM runs with a power budget assigned by SLURM
  - Hands off execution to NRM and application through a manifest and NRM JSON
  - NRM runs the application to completion

Build and Run Application with NRM and GEOPM

Step 1: Configure and build GEOPM

```
$> git clone https://github.com/amarathe84/geopm-nrm.git
$> ./autogen.sh
$> ./configure --prefix=$HOME/geopm/install-ecp \
        CC=<path to C compiler> \
        CXX=<path to C++ compiler> \
        F77=<path to Fortran compiler> \
            --enable-ompt
$> make
$> make install
```

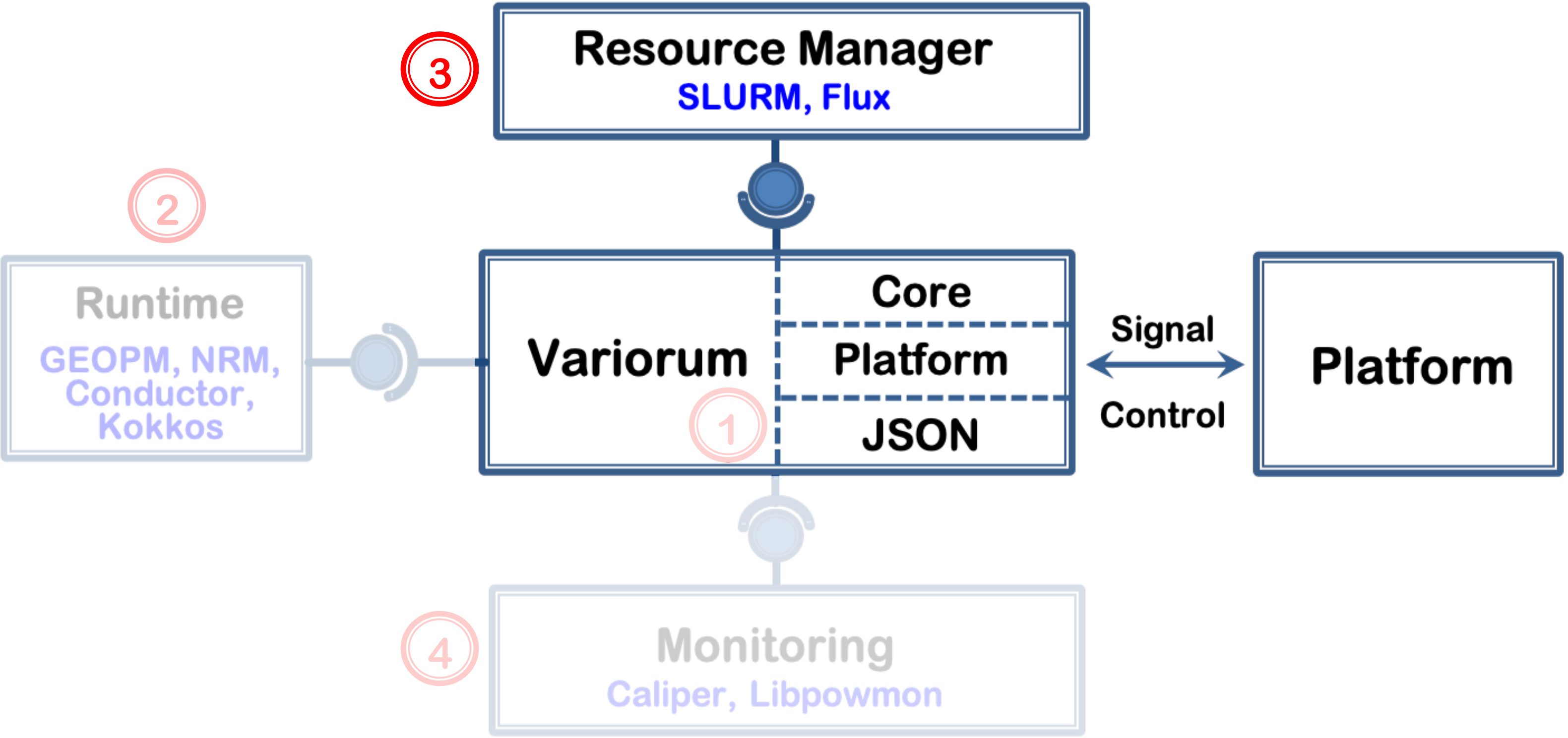
Step 2: Build NRM (needs nix-build/NixOS)

```
$> nix-build -A nrm
```

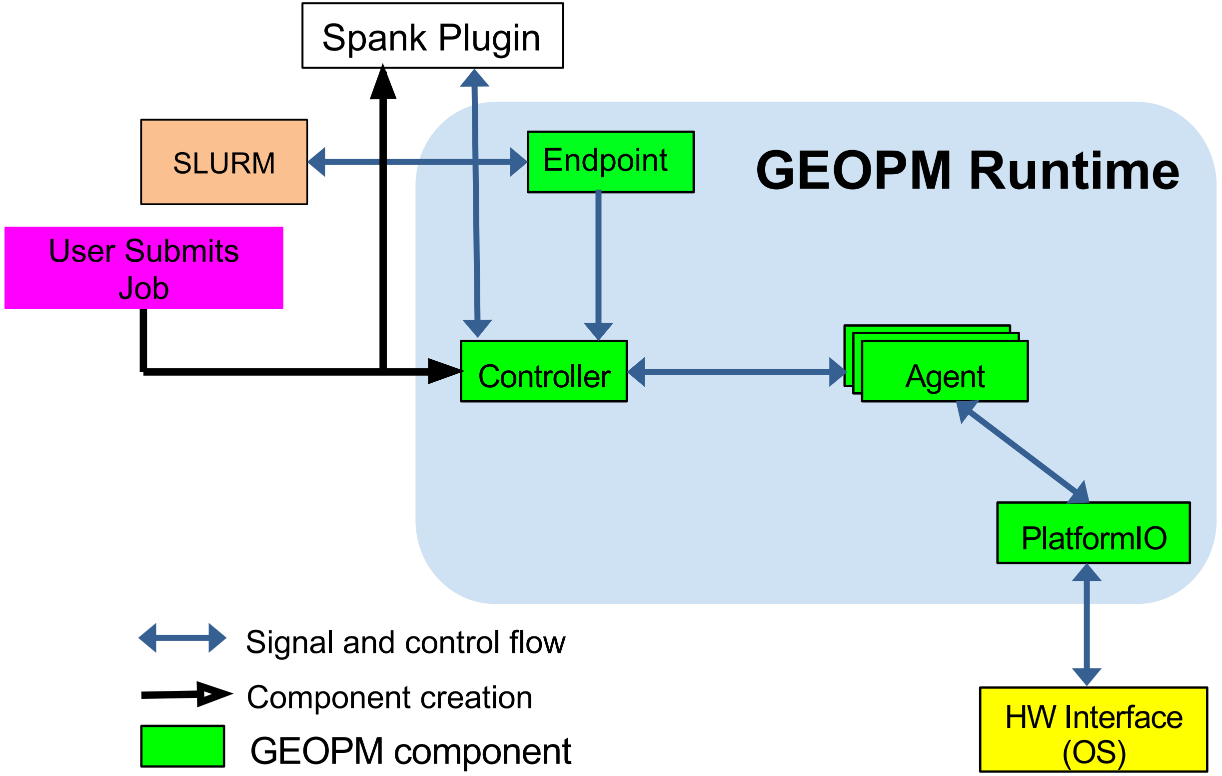
Step 3: Run GEOPM and NRM

```
$> OMP_NUM_THREADS=<num therads> \
    geopmnrmlaunch \
            --geopm-ctl=process \
            --geopm-policy=<JSON policy spec> \
            --geopm-report=report \
            --geopm-trace=trace \
            --geopm-agent=power_governor \
            -N <numnodes> -n <numtasks> -m block -l \
                -- \
                <application path>
```

Interfacing Variorum with PowerStack Components

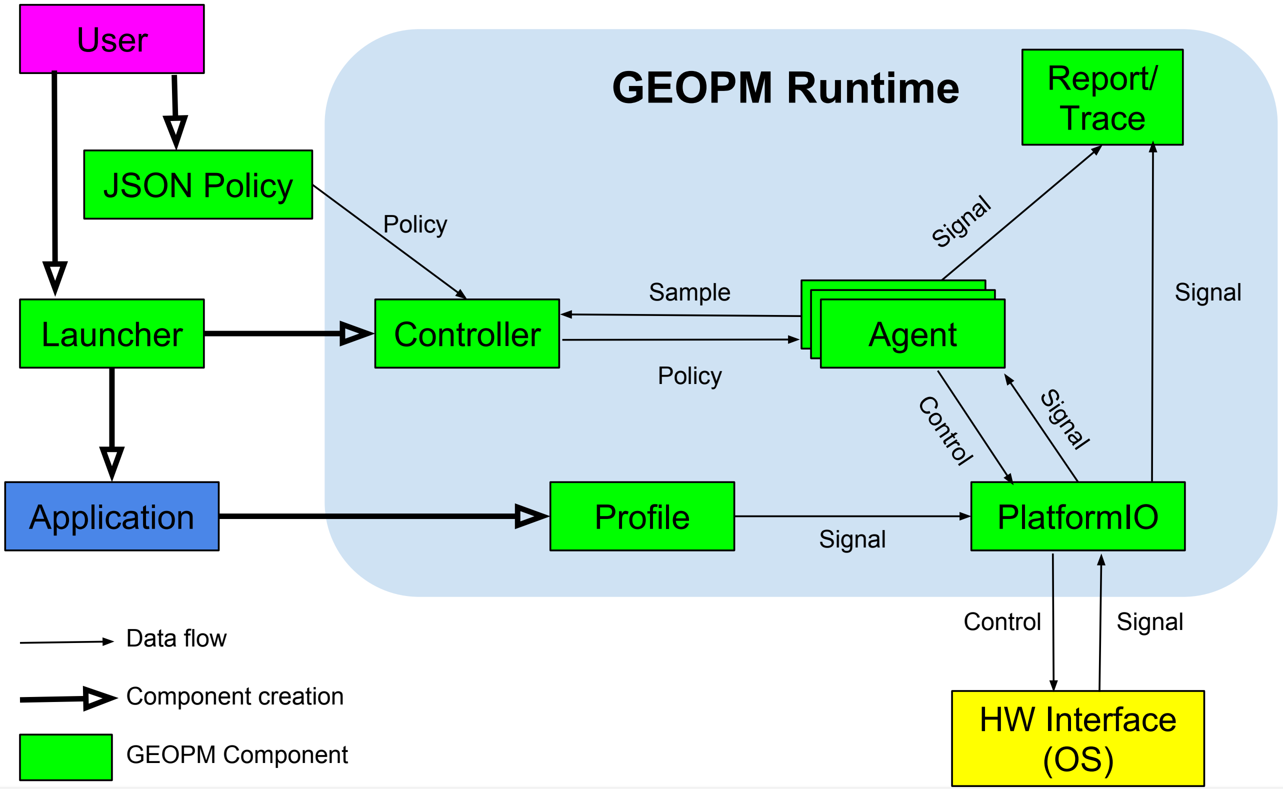


SLURM, GEOPM and Variorum Integration: Default Behavior



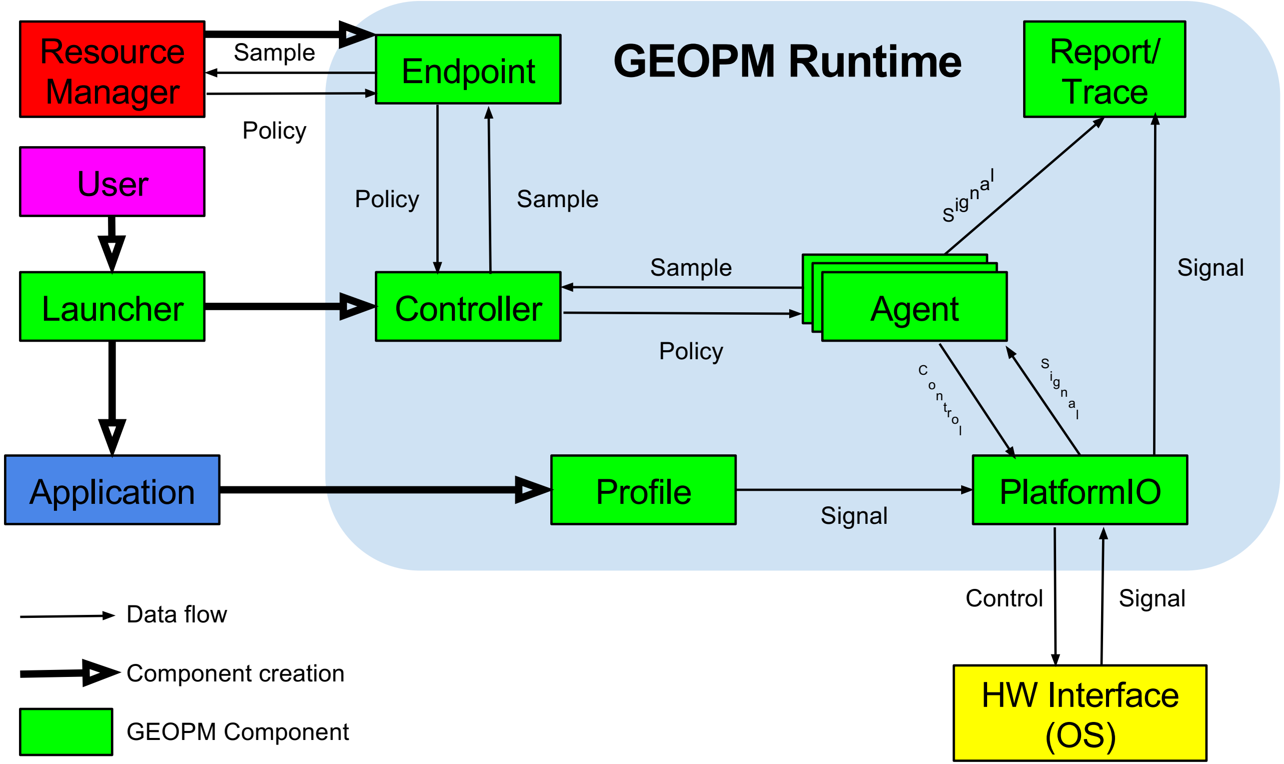
- SLURM allocates resources and runs the spank plugin on each node
- Spank plugin derives the default node power budget
- GEOPM PlatformIO picks up the assigned power budget and applies it to each socket
- GEOPM continues execution through completion with the assigned power budget

SLURM, GEOPM and Variorum Integration: User-Driven



- SLURM allocates resources and runs the spank plugin on each node
- Spank plugin derives the node power budget
  Based on user's request
- GEOPM PlatformIO picks up the assigned power budget and applies it to each socket
- GEOPM continues execution

SLURM, GEOPM and Variorum Integration: Resource Manager Driven



- SLURM allocates resources, derives a node power budget and runs the spank plugin on each node
- Spank plugin passes the node power budget to GEOPM
- GEOPM PlatformIO picks up the assigned power budget and applies it to each socket
- GEOPM continues execution

SLURM Integration with Variorum

Steps involved in applying the power budget

1. Allocate job resources (salloc/sbatch)
2. Invoke Variorum API to apply power limit
3. Instantiate application with GEOPM
4. Apply JSON-specified power budget with GEOPM (static)
5. Run application to completion

SLURM Integration: Verification/Testing

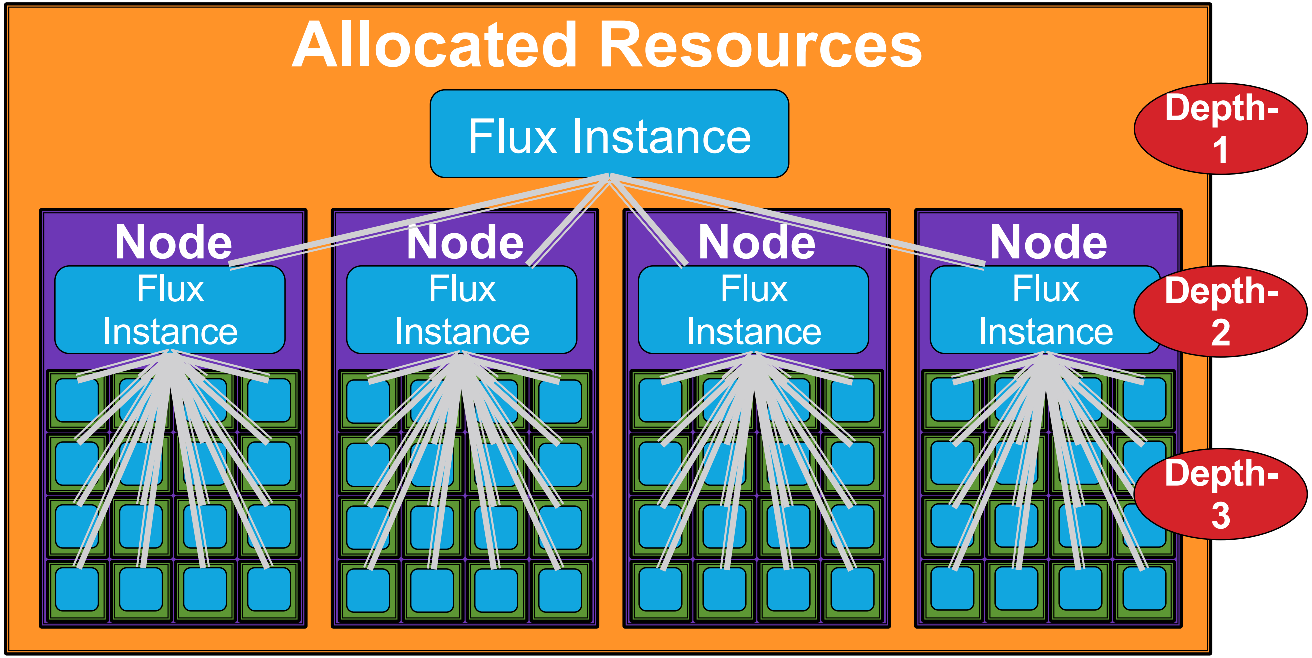
1. GEOPM Configurations: JSON
2. Applications
3. SPANK plugin configuration
4. Job configurations and outcomes
    1. MPI
    2. Non-MPI
    3. OpenMP
    4. MPI+OpenMP

Configuration files:
/etc/geopm/environment-default.json
/etc/geopm/environment-override.json

```
/etc/geopm/environment-override.json

{"GEOPM_AGENT": "power_balancer",
 "GEOPM_POLICY": ../ig/geopm_power_balancer.json”}
```

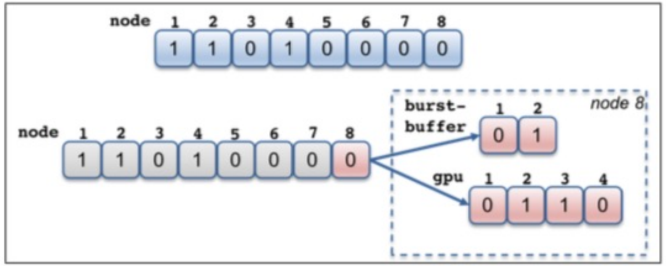
Flux provides a new hierarchical scheduling model to meet Exascale challenges – targeted on El Capitan



Our *"Fully Hierarchical Scheduling"* is designed to cope with many emerging workload challenges.

The traditional resource data models are largely ineffective to cope with the resource challenge.

- Designed when the systems are much simpler
  - Node-centric models
  - SLURM: bitmaps to represent a set of compute nodes
  - PBSPro: a linked-list of nodes

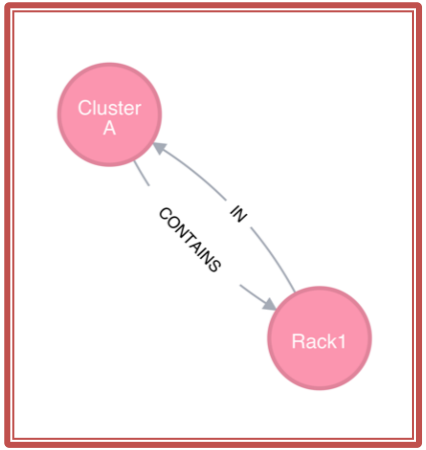


- HPC has become far more complex
  - Evolutionary approach to cope with the increased complexity
  - E.g., add auxiliary data structures on top of the node-centric data model

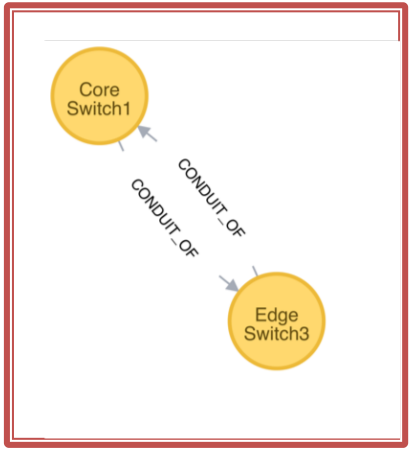
- Can be quickly unwieldy
  - Every new resource type requires new a user-defined type
  - A new relationship requires a complex set of pointers cross-referencing different types.

Flux uses a graph-based resource data model to represent schedulable resources and their relationships.

- A graph consists of a set of vertices and edges
  - Vertex: a resource
  - Edge: a relationship between two resources

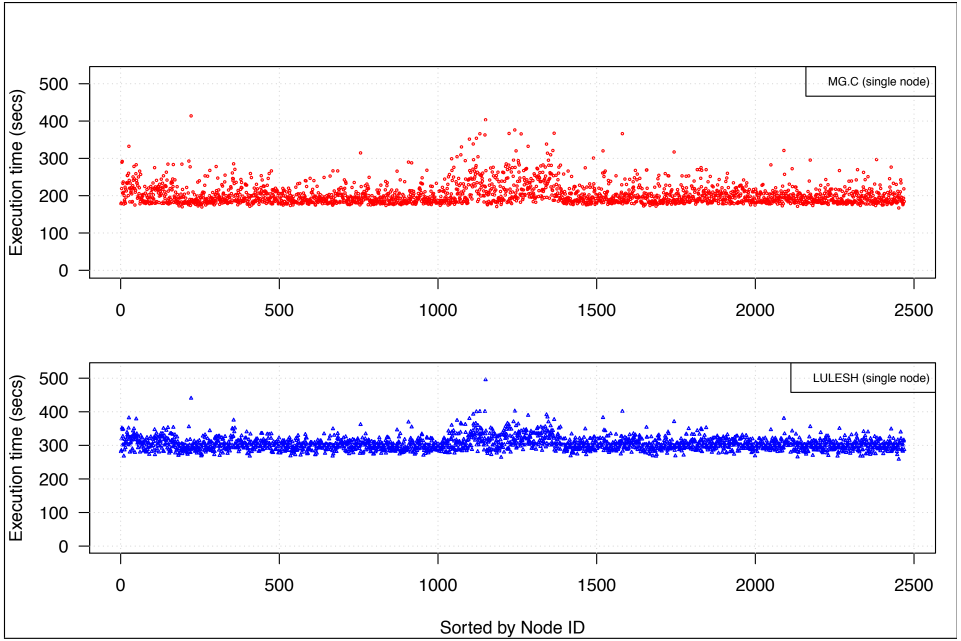


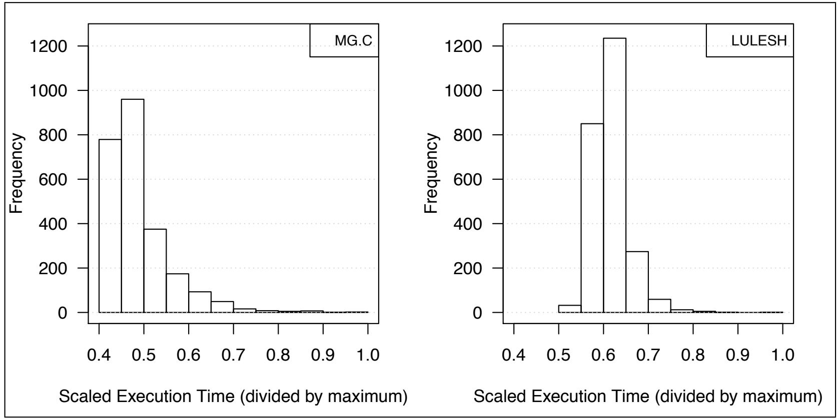
Containment subsystem



Network connectivity subsystem

Real world example of variation: Quartz cluster, 2469 nodes, 50 W CPU power per socket





- 2.47x difference between the slowest and the fastest node for MG
- 1.91x difference for LULESH.

https://github.com/flux-framework/flux-sched/tree/master/resource/policies

Statically determining node performance classes

- Ranking every processor is not feasible from point of view of accounting as well as application differences

- Statically create bins of processors with similar performance instead
  - Techniques for this can be simple or complex
  - How many classes to create, which benchmarks to use, which parameters to tweak
  - Our choice: 5 classes, LULESH and MG, 50 W power cap

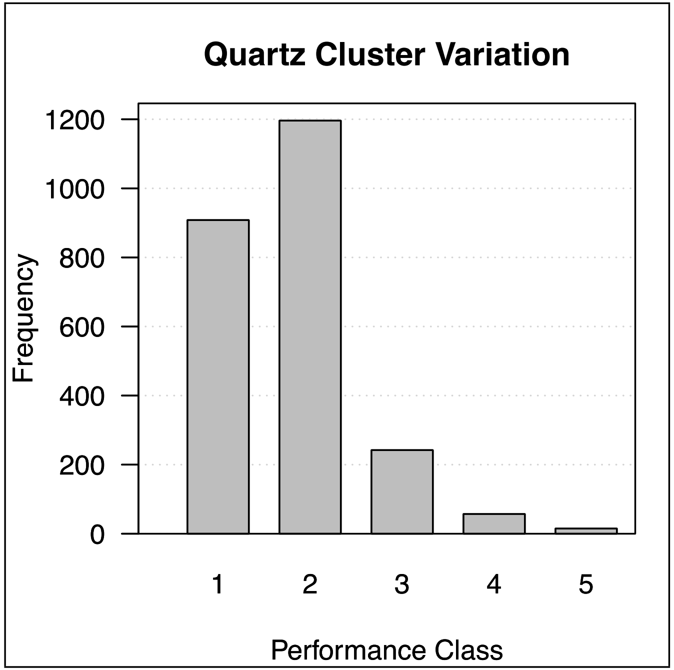
- Mitigation
  - Rank-to-rank: minimize spreading application across performance classes
  - Run-to-run: allocate nodes from same set performance classes to similar applications

Statically determining node performance classes: 2469 nodes of Quartz

$$t_{combined_i} = \frac{\frac{t_{MG_i}}{median(t_{MG_{1:n}})} + \frac{t_{LULESH_i}}{median(t_{LULESH_{1:n}})}}{2}$$

$$t_{norm_j} = \frac{t_{combined_j} - min(t_{combined_j})}{max(t_{combined_j}) - min(t_{combined_j})}$$

$$p = \begin{cases} 1, & \text{if } 0 \leq t_{norm_i} \leq 0.10 \\ 2, & \text{if } 0.10 < t_{norm_i} \leq 0.25 \\ 3, & \text{if } 0.25 < t_{norm_i} \leq 0.40 \\ 4, & \text{if } 0.40 < t_{norm_i} \leq 0.60 \\ 5, & \text{if } 0.60 < t_{norm_i} \leq 1.0 \end{cases}$$



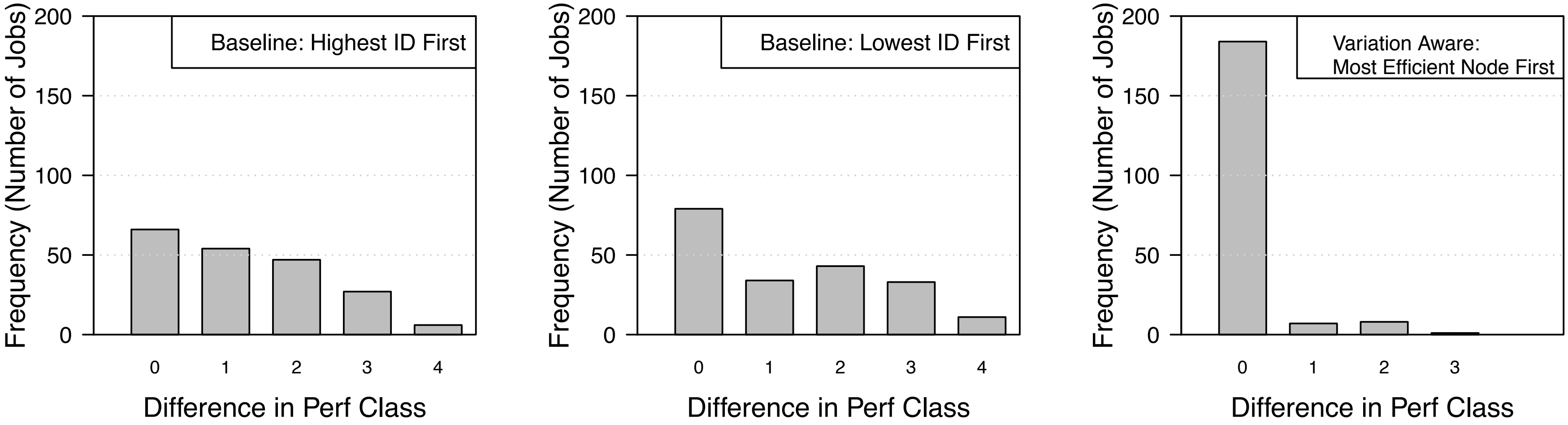
https://github.com/flux-framework/flux-sched/tree/master/resource/policies

Measuring impact of variation-aware scheduling

$$P_j := \{p_a | a \in n \wedge allocated(a, j)\}$$
$$fom_j = max(P_j) - min(P_j)$$

- *allocated(a,j)* returns true if node *a* has been allocated to job *j*
- P_j is the set of performance classes of the nodes allocated to job *j*
- Figure of merit, fom_j, is a measure of how widely the job is spread across different performance classes
- For a job trace, we will look for number of jobs with low figure of merit

Variation-aware scheduling results in 2.4x reduction in rank-to-rank variation in applications with Flux



Facilities Recap: Mitigating Power Swings on Sierra/Lassen with in-depth application analysis with Variorum

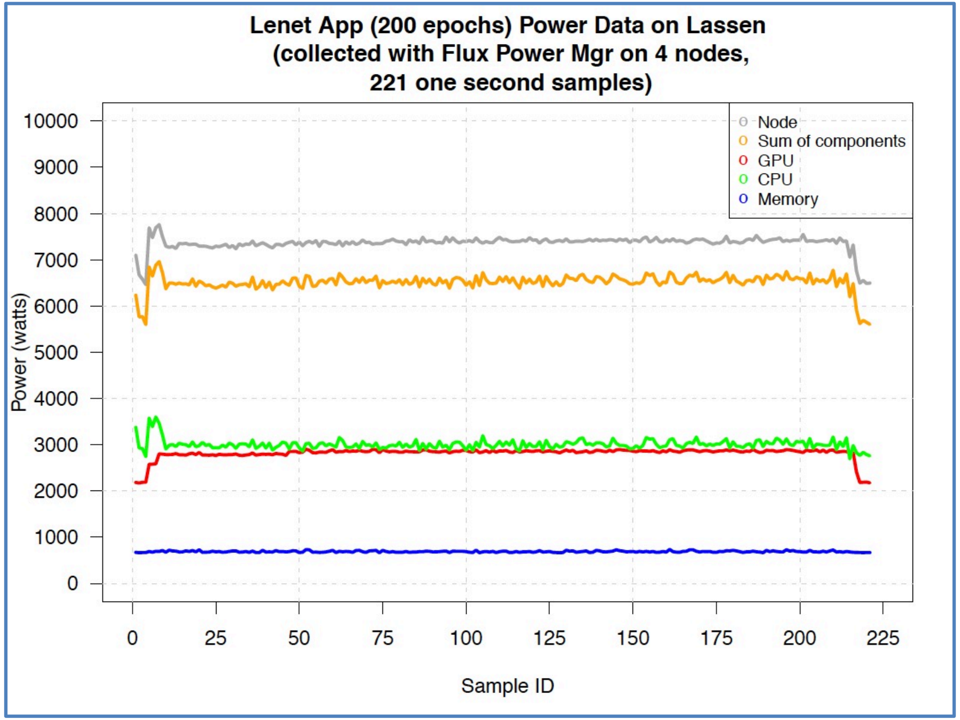
*Example: LBANN on **<u>Sierra</u>** at full scale has significant fluctuations impacting LLNL's electrical grid -- workload swings expected to worsen at exascale*

- Livermore Big Artificial Neural Network toolkit (LBANN) -- infrastructure used for deep learning in HPC
- LBANN utilizes all 4 GPUs per node
- Data shows 3 minute samples over 6 hours on Sierra with >200 KW swings
- Other workflows have similar trends with power fluctuations at scale
- Mitigation of power fluctuations is required to avoid electrical supply disruption
- Variorum + Flux can dynamically analyze applications and prevent future fluctuations

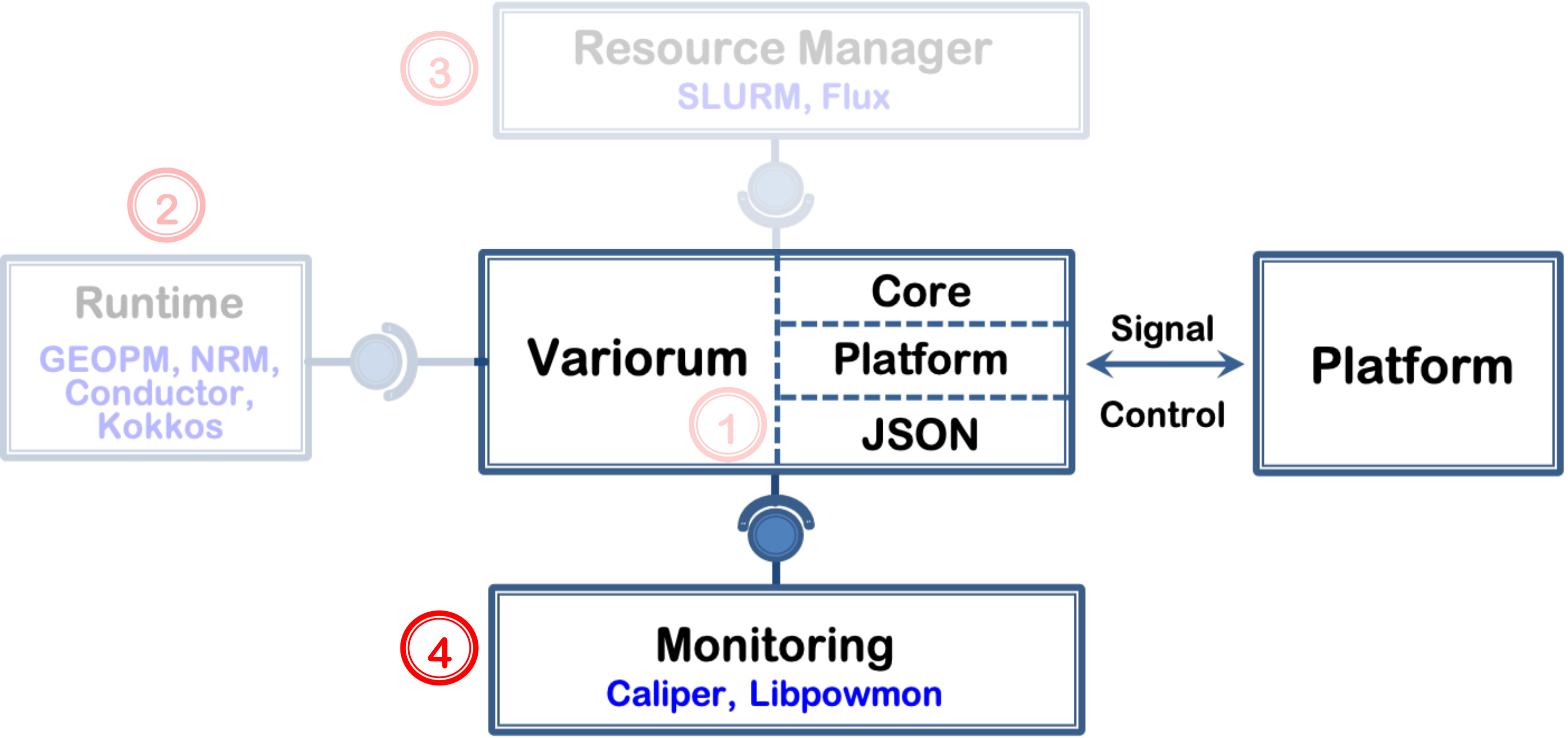
Combining previous research into upcoming production efforts: Flux + Variorum to monitor and manage power swings of workflows

- Flux Power Manager Module is underway for El Capitan
- Utilizes the Variorum JSON interface to develop a Flux system instance to monitor and control power
- Algorithms for detecting and managing power swings at scale are underway
- Example shows LBANN workflow's Lenet application being monitored

https://github.com/rountree/flux-power-mgr



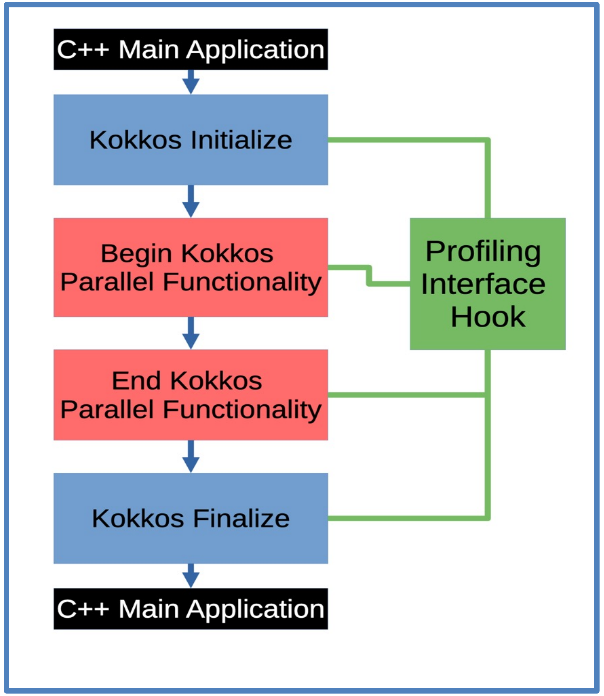
Interfacing Variorum with PowerStack Components



Enabling workflow power monitoring with the Kokkos and Caliper ports of Variorum

- Utilizes the Variorum JSON interface to allow for monitoring of integrated workflows
- Kokkos port has been merged into production (with kokkos-tools) and provides per-rank output
- Caliper service is under development
- Both ports will be tested for scalability with benchmarks and hardened in the upcoming Variorum release

https://github.com/kokkos/kokkos-tools/tree/develop/profiling/variorum-connector



Upcoming Variorum Next Steps

Development Efforts

- Upcoming release: last quarter of 2021
- Ports for AMD CPU (WIP), PowerAPI, AMD GPUs
- Advanced APIs
- CI and testing for ECP on exascale microarchitectures

Research Efforts

- Harden Caliper service for Variorum
- Workflow integration (MuMMI, E3SM, LBANN)
- MLPerf (GPU) characterization
- SLURM + GEOPM + Variorum: Extend it to use JSON

Thank you for attending our tutorial series!

- Both modules will be repeated on August 20 and August 23
  - Introduction to Variorum
  - Integrating Variorum with System Software and Tools

- Submit your issues or feature request: https://github.com/llnl/variorum/issues
- Documentation: https://variorum.readthedocs.io
- Join our mailing list (low traffic): variorum-users@llnl.gov
- Questions? Email us at variorum-maintainers@llnl.gov

CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory