

Managing Power Efficiency of HPC Applications with Variorum and GEOPM

ECP Tutorial

Feb 4, 2020 2:30PM-6:00 PM

Tapasya Patki, Stephanie Brink, Aniruddha Marathe, Barry Rountree
David Lowenthal (U. Arizona), Jonathan Eastep (Intel)



Agenda

- Part I: Overview of GEOPM (15 minutes)
 - High-level design
 - User-facing, application-context markup API
 - Demonstrations (10 minutes)
- Part II: Plug-ins to extend GEOPM algorithm and platform support (30 minutes)
 - **Agent**: Run-time tuning extension
 - **PlatformIO**: Platform-specific support extension
 - Demonstrations (10 minutes)
- Part III: ECP Argo Contributions (30 minutes)
 - **ConductorAgent**: Transparent, performance-optimizing configuration selection
 - **IBM PlatformIO plugin**: Port of GEOPM to IBM Power9 + Nvidia platform
- Questions/Discussion (10 minutes)

Part I: Hands-on Tutorial on GEOPM



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

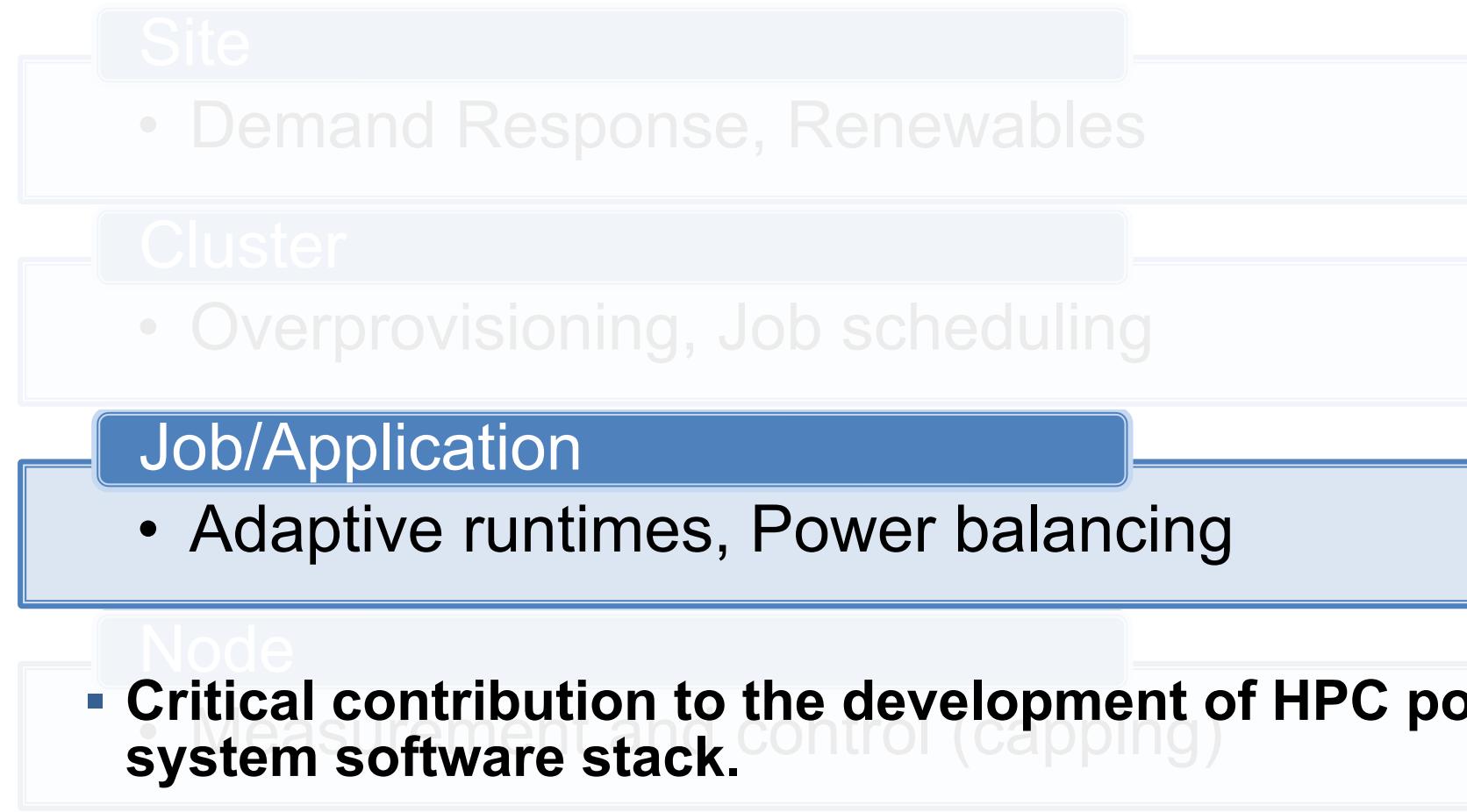
RMAP,
P-SLURM,
PowSched

GEOPM,
Conductor,
Pshifter,...

Libmsr,
msr-safe

Background: System Software Stack for Power Management

Inherited Power Bounds



Software

Dashboards

RMAP,
P-SLURM,
PowSched

GEOPM,
Conductor,
PShifter,...

LibMSR,
MSI-Safe

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

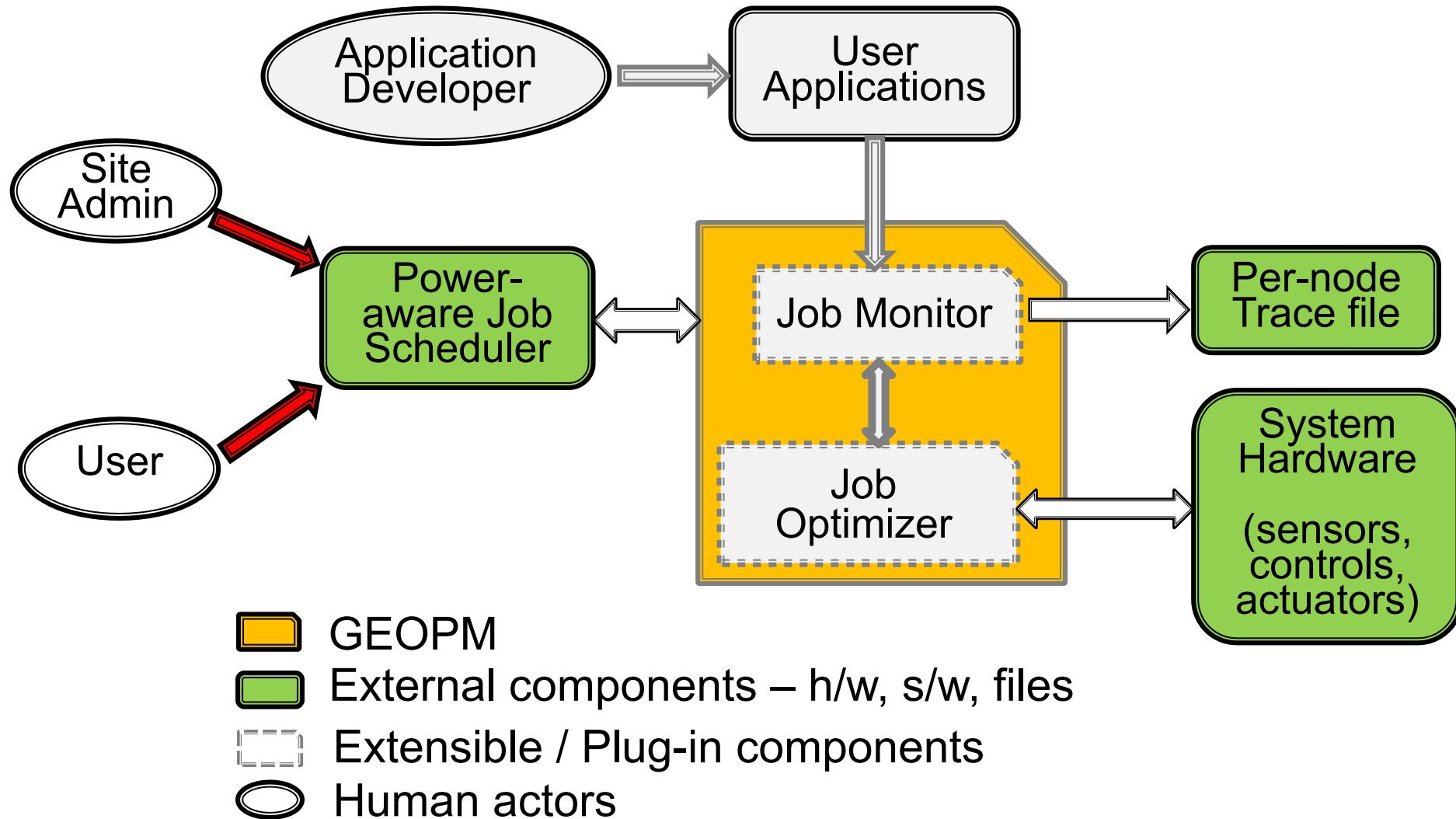
- Power-aware runtime system for large-scale HPC systems
- Intel developed a production-grade, scalable, open-source job-level extensible runtime and framework
 - Extensibility through plug-ins + advanced default functionality
- Limitations of existing runtimes
 - Research-based codes addressed specific 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 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

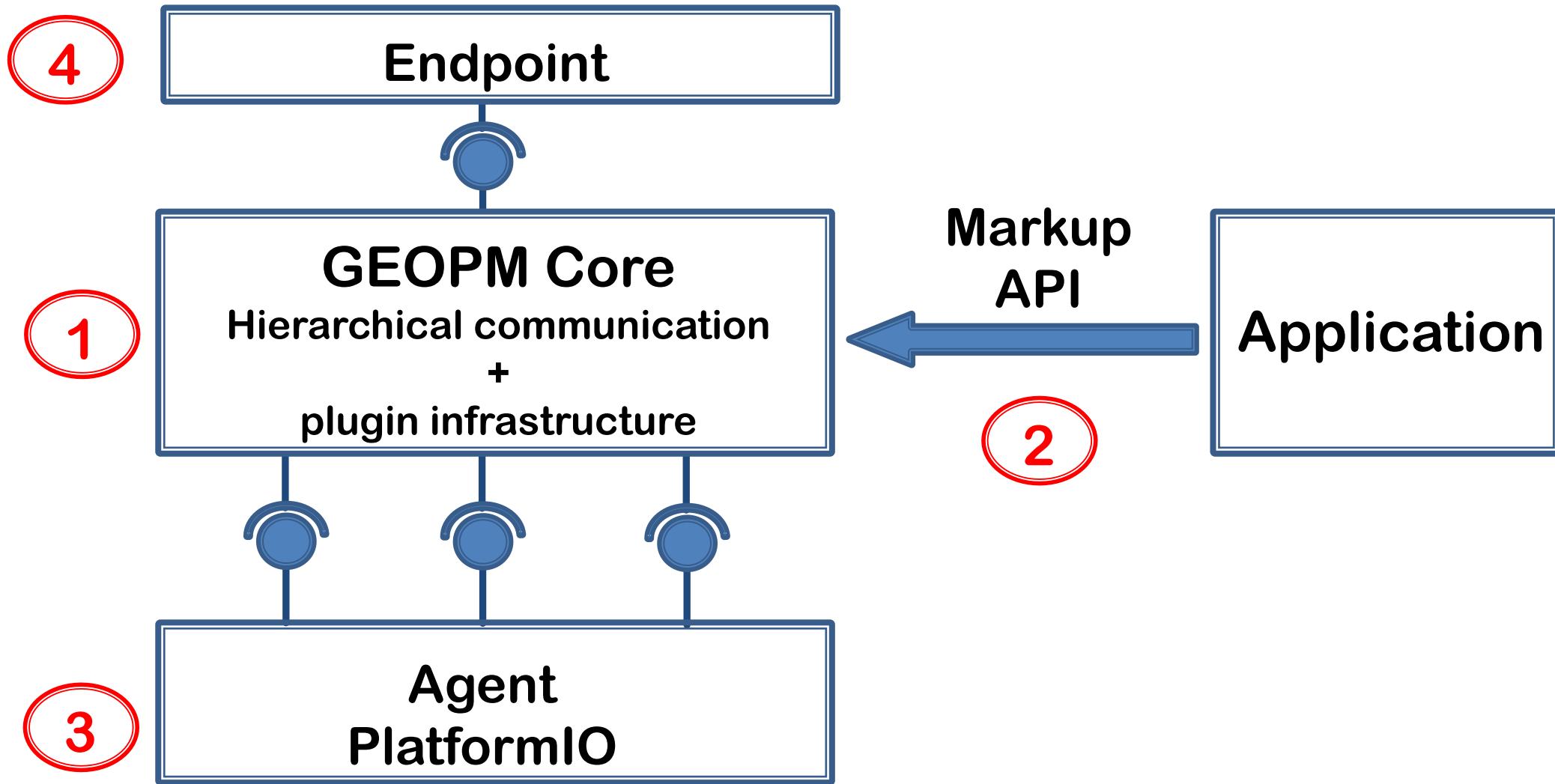
GEOPM System Model



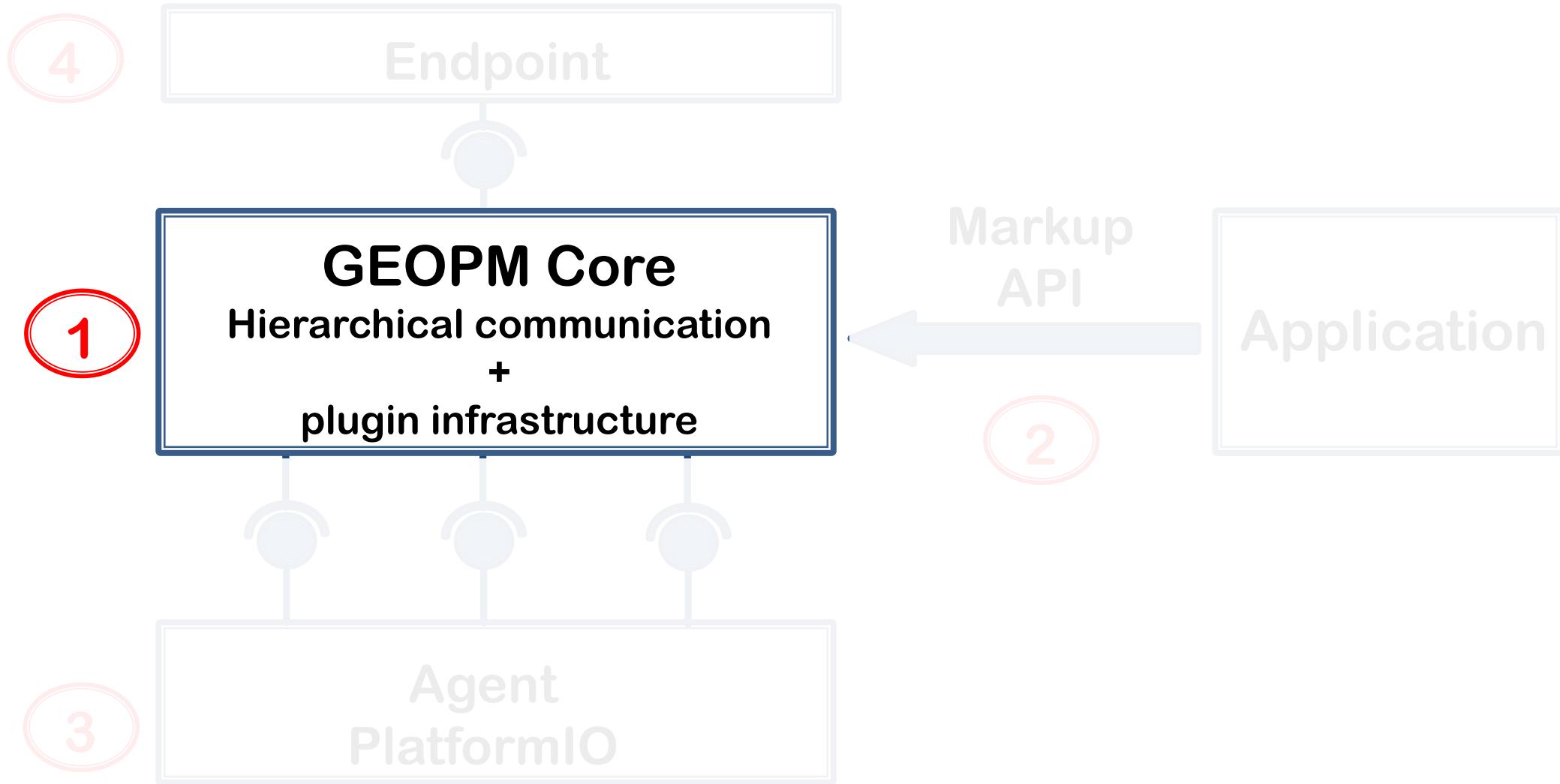
GEOPM: Capabilities

- 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

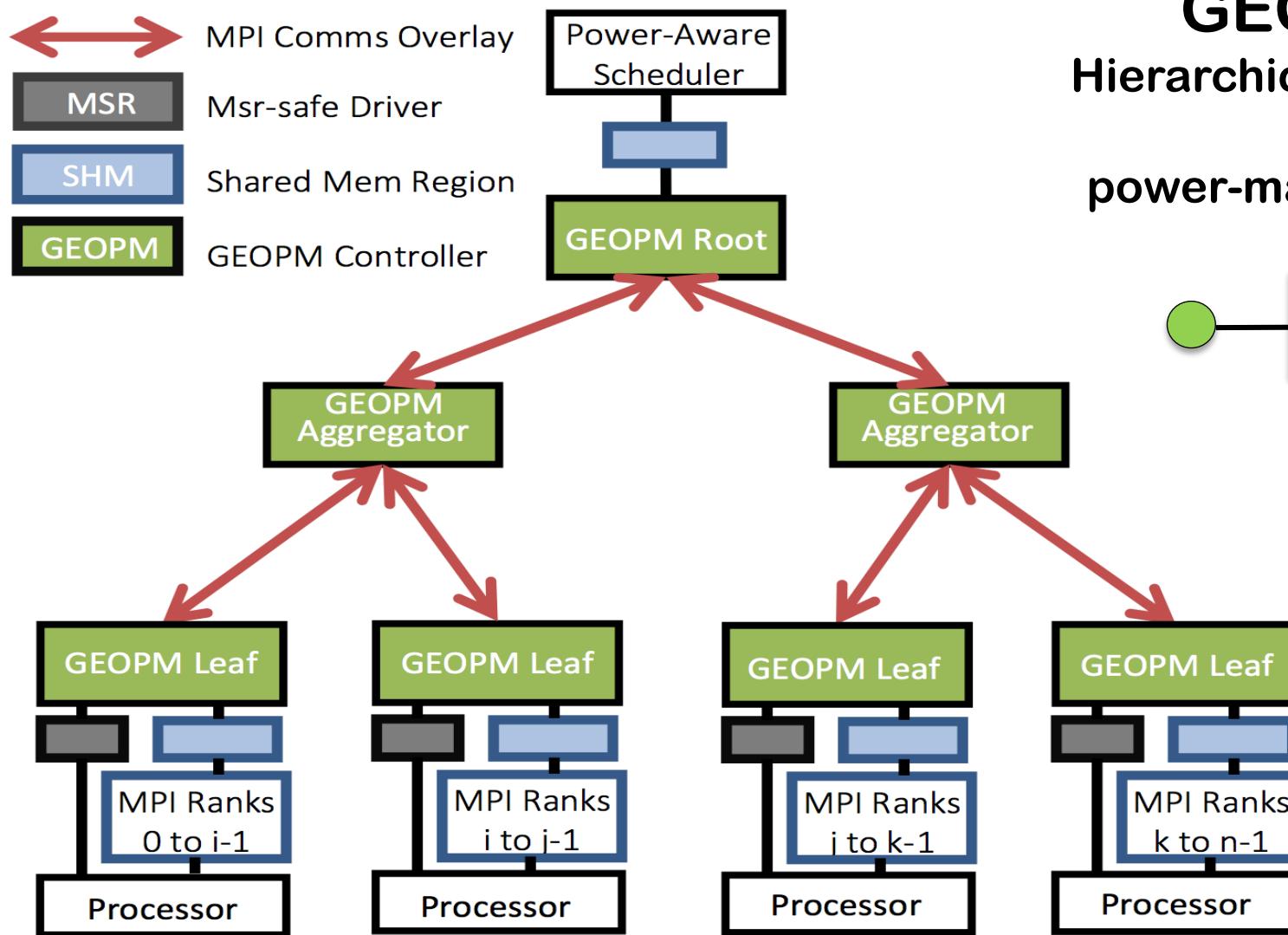
GEOPM Components of Interest



GEOPM Components of Interest



GEOPM Infrastructure



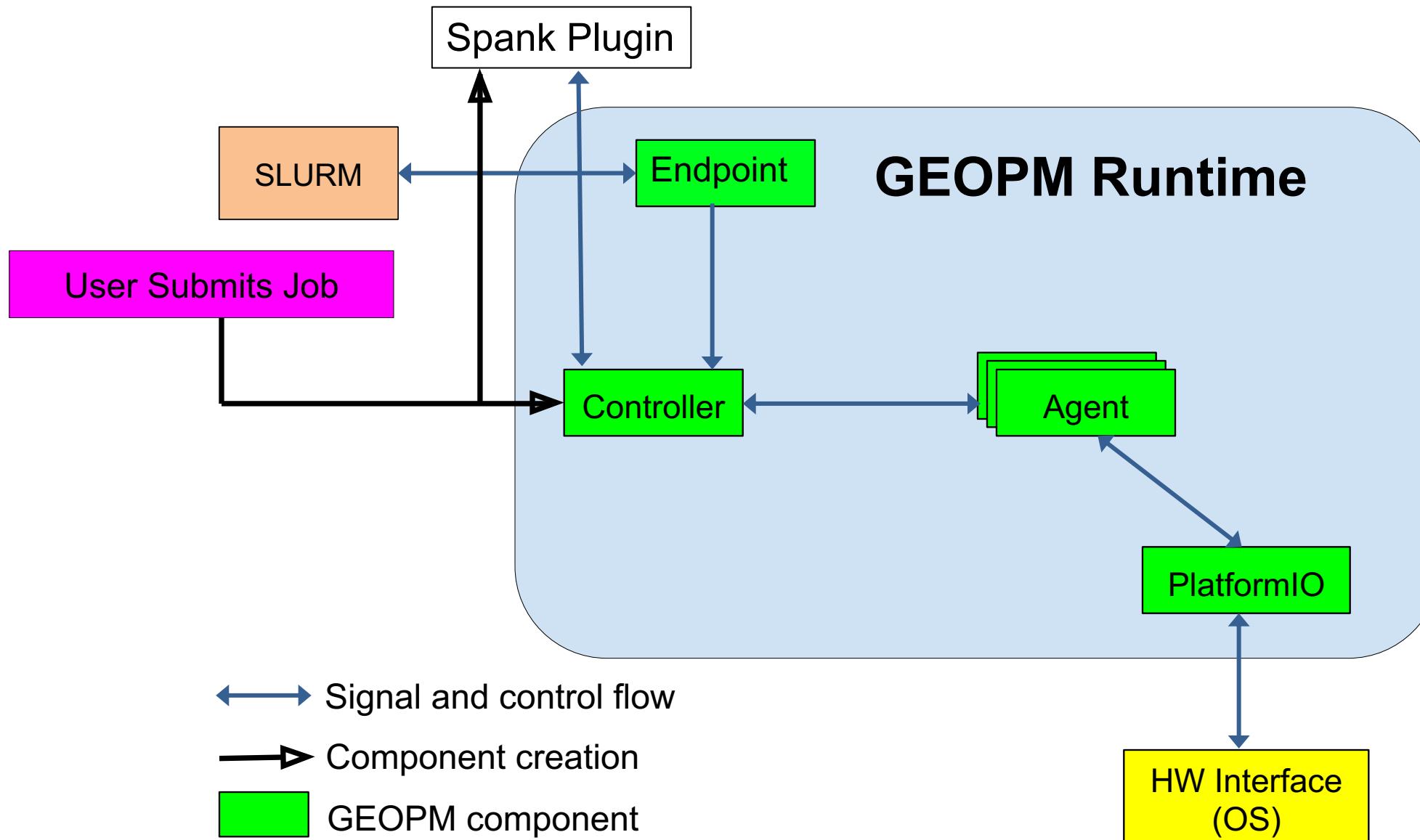
GEOPM Core
Hierarchical communication
+
power-management plugin

GEOPM Infrastructure

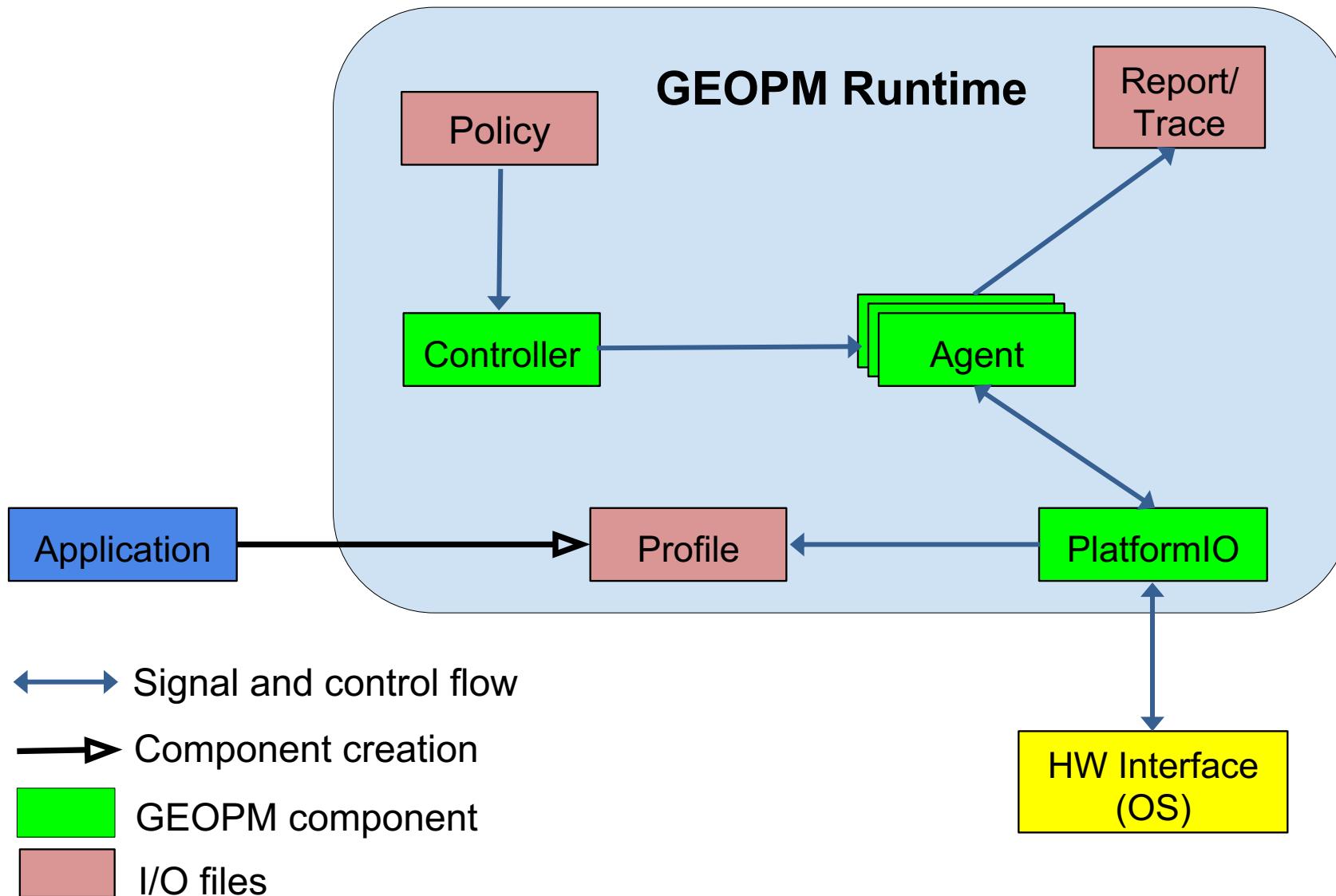
- GEOPM [Source repository](#) navigation
 - Branches, directories, releases
 - [GEOPM Wiki](#)
- Build process
 - Dependencies
 - Build configuration
- GEOPM core infrastructure source
 - Overview of important classes
 - Plug-in source
 - Tutorials and examples
 - Test coverage



GEOPM: Component Communication



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 <> \  
  --geomp-ctl=process \  
  --geomp-agent=AGENT_NAME \  
  --geomp-policy=POLICY_FILE.json \  
  --geomp-report=REPORT_FILE.txt \  
  --geomp-trace=TRACE_FILE.csv \  
  -- APP_EXEC APP_OPTIONS
```

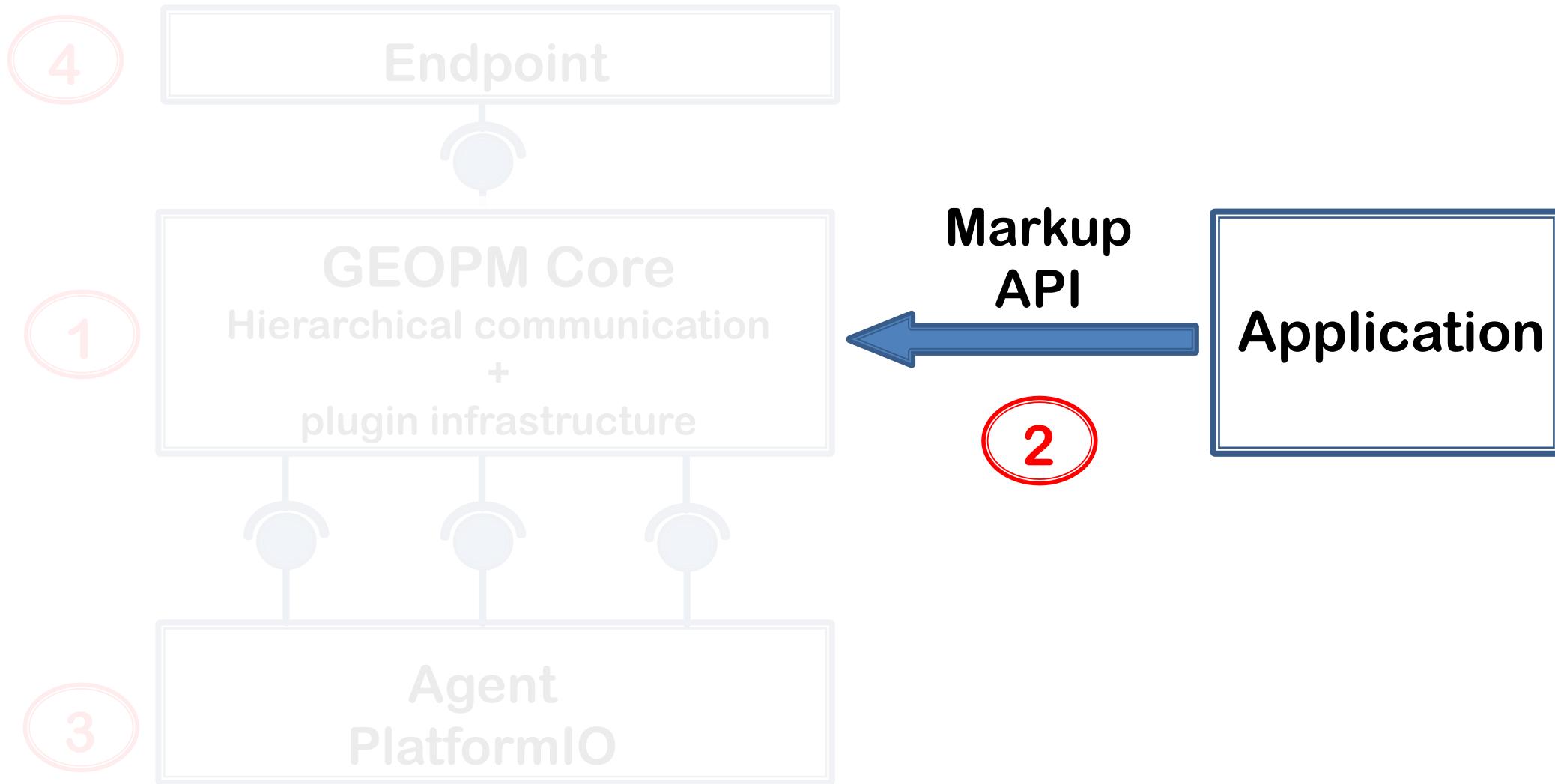
Example:

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

Demo: Running Application with GEOPM



GEOPM Components of Interest





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
 - `geomp_prof_region(name, hint, ID)`
 - `geomp_prof_enter(ID)`
 - `geomp_prof_exit(ID)`
- Marking region progress
 - `geomp_prof_progress(ID, %progress)`
- Marking a timestep
 - `geomp_prof_epoch()`

OpenMP Region

- Marking up regions of interest
 - `geomp_tprof_init(num_work_unit)`
 - `geomp_tprof_init_loop(num_thread, thread ID, num_iter, chunk_size)`
- Marking region progress
 - `geomp_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

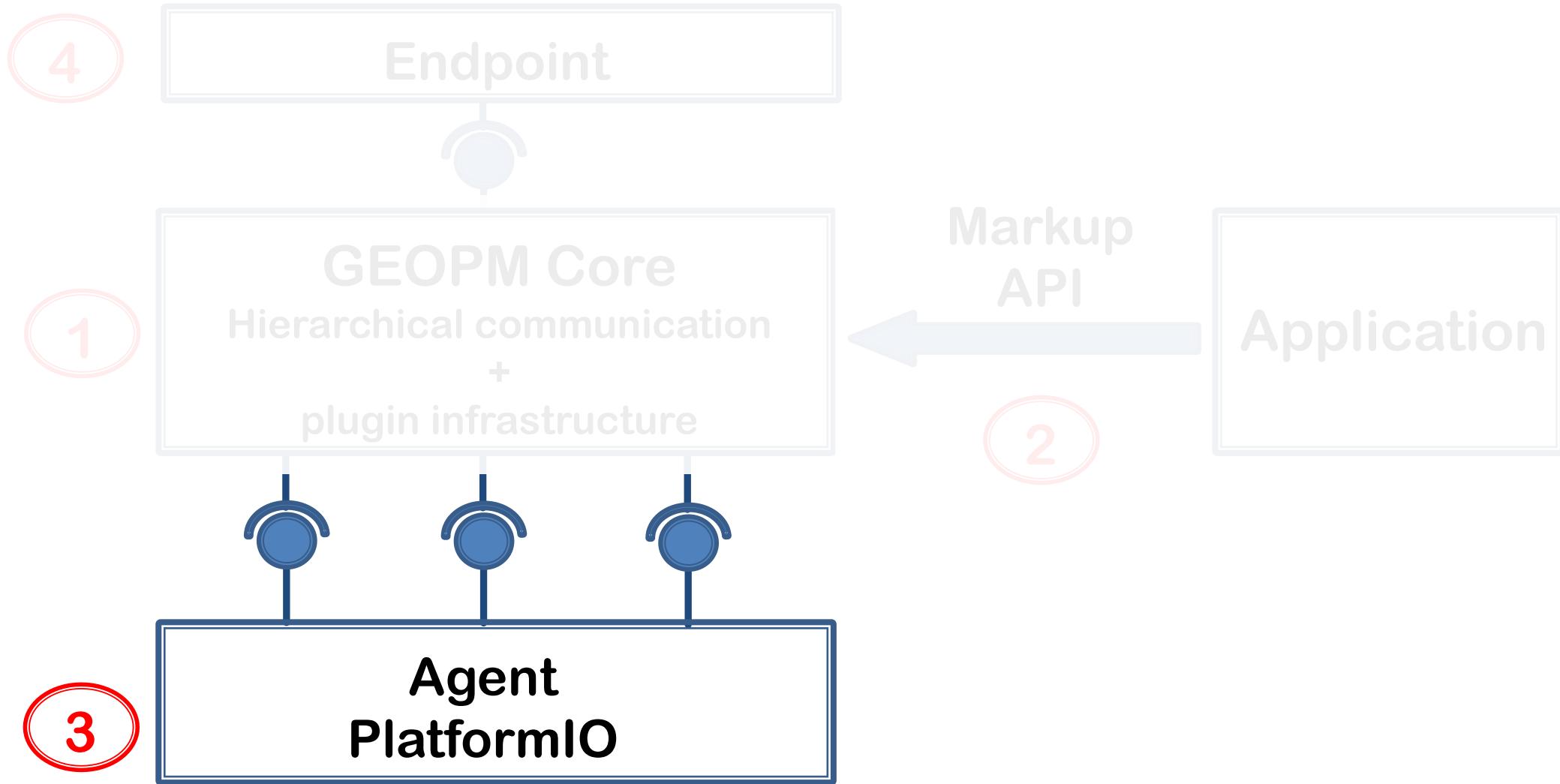
- Governed policy
 - Node-level assignment
- Balanced policy
 - Cluster-level assignment

Extension Interfaces

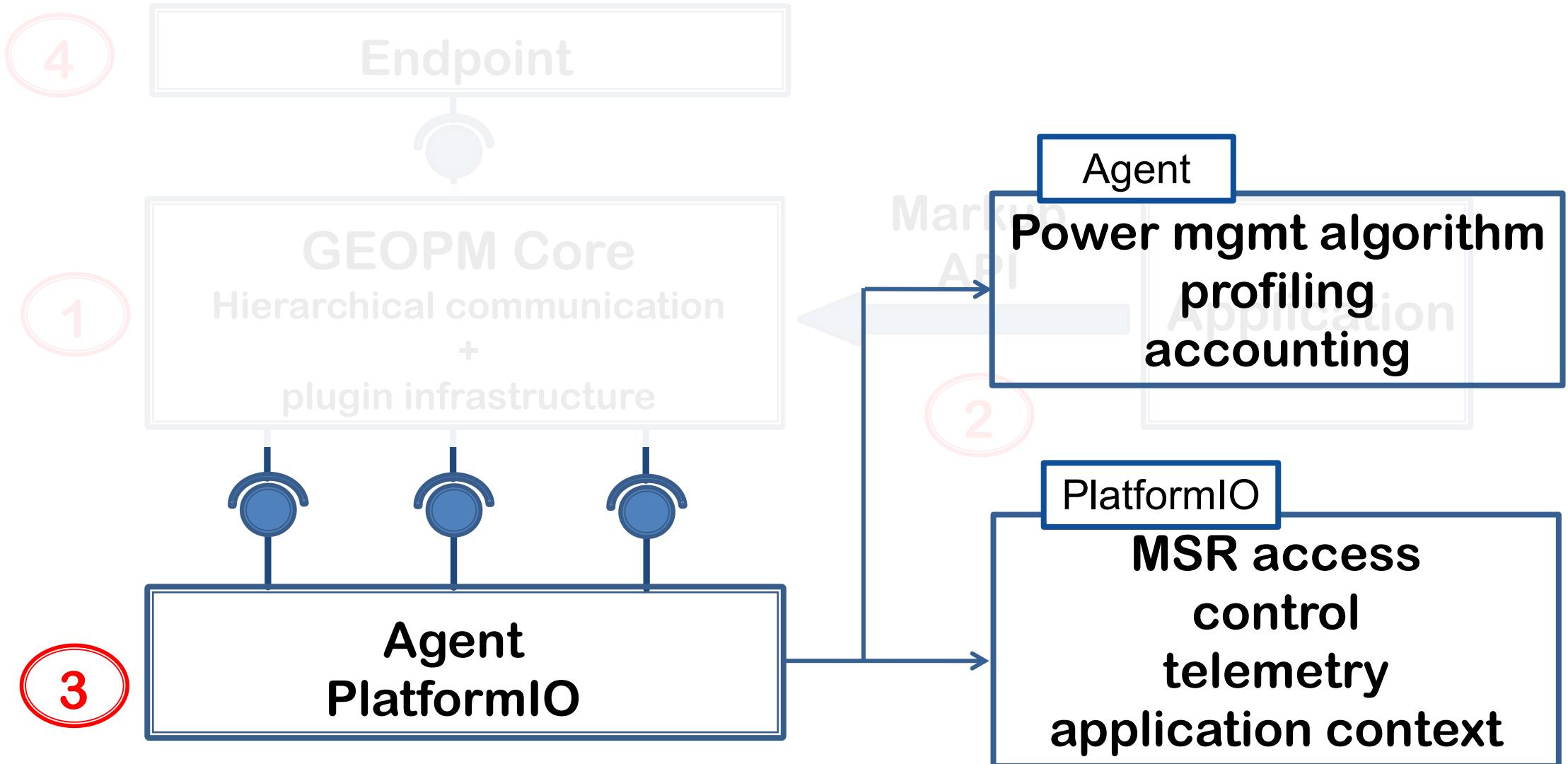
- New Agent plugin: ConductorAgent
- New PlatformIO plugin: IBM port of GEOPM

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
 - EnergyEfficientAgent
- Example PlatformIO plugins
 - MSRIOGroup
 - KNLIOGroup
- Tutorial plugins: [ExampleAgent](#) and [ExampleIOGroup](#)
 - Key methods and code blocks
 - Policy description interface

Demo: GEOPM Agent Example

Part III: ECP Argo Contributions

ECP Argo: 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)

ECP Argo Contributions: 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

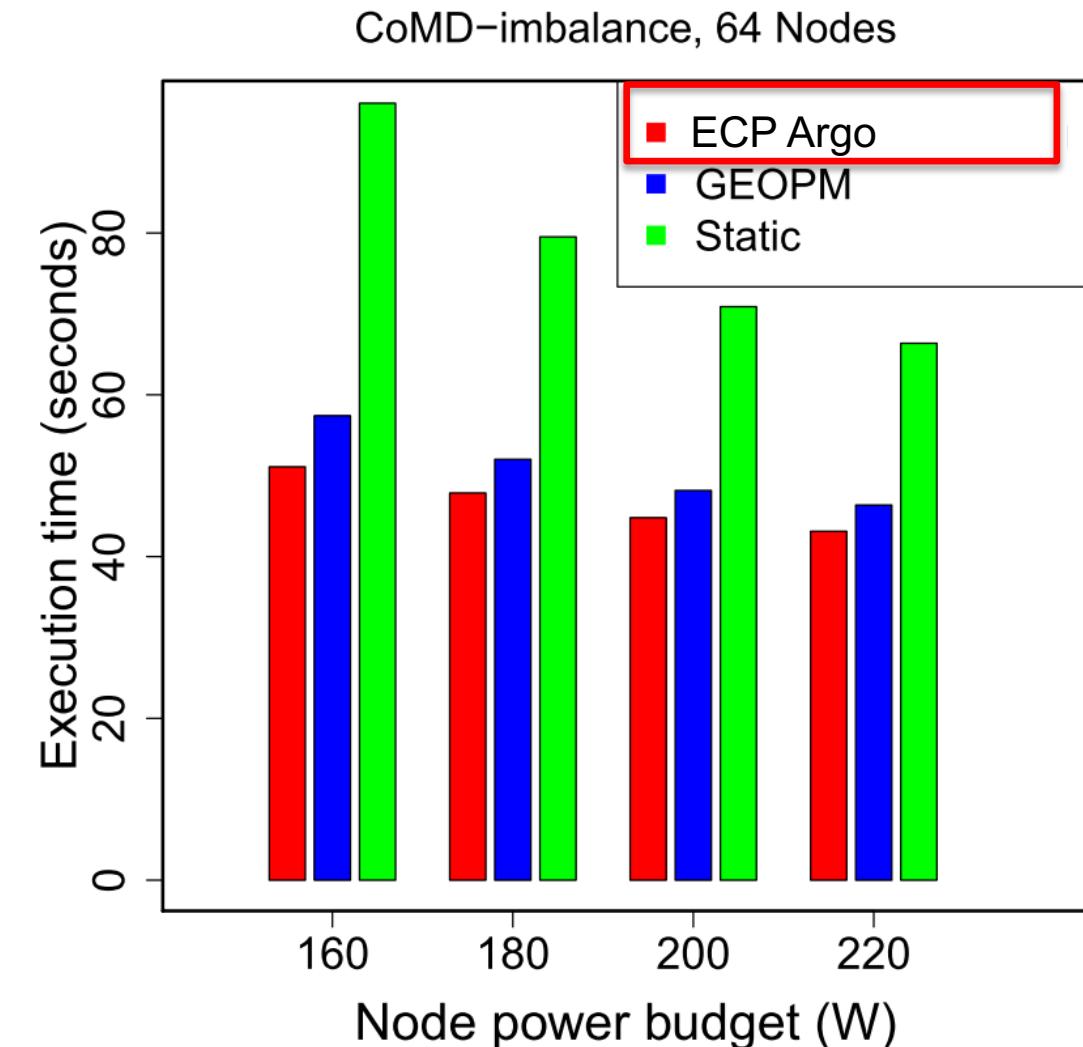
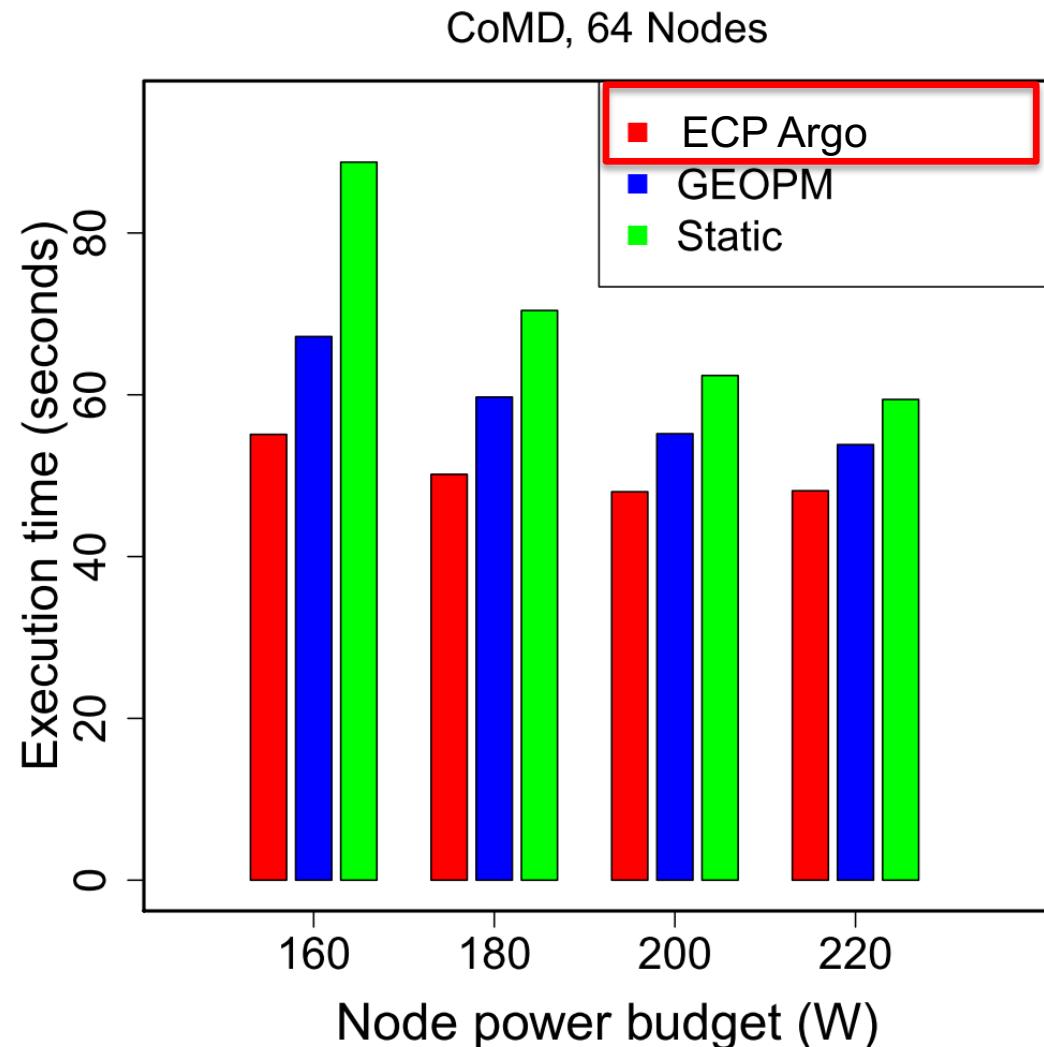
- Governed policy
 - Node-level assignment
- Balanced policy
 - Cluster-level assignment

Extension Interfaces

- New policy agent plugin: ConductorAgent
- New PlatformIO plugin: IBM port of GEOPM



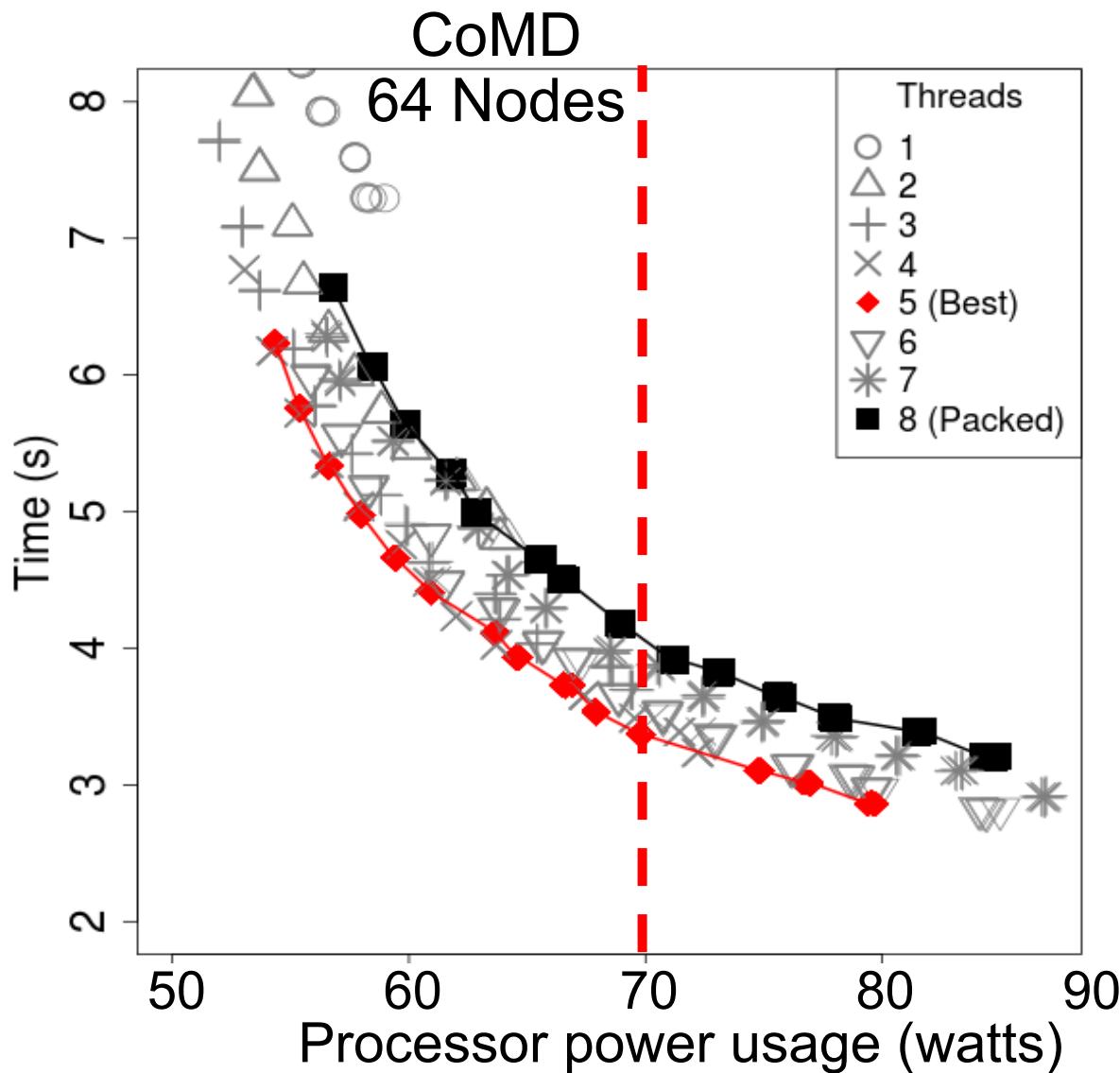
ECP Argo: How Much Do We Gain With Configuration Tuning?



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



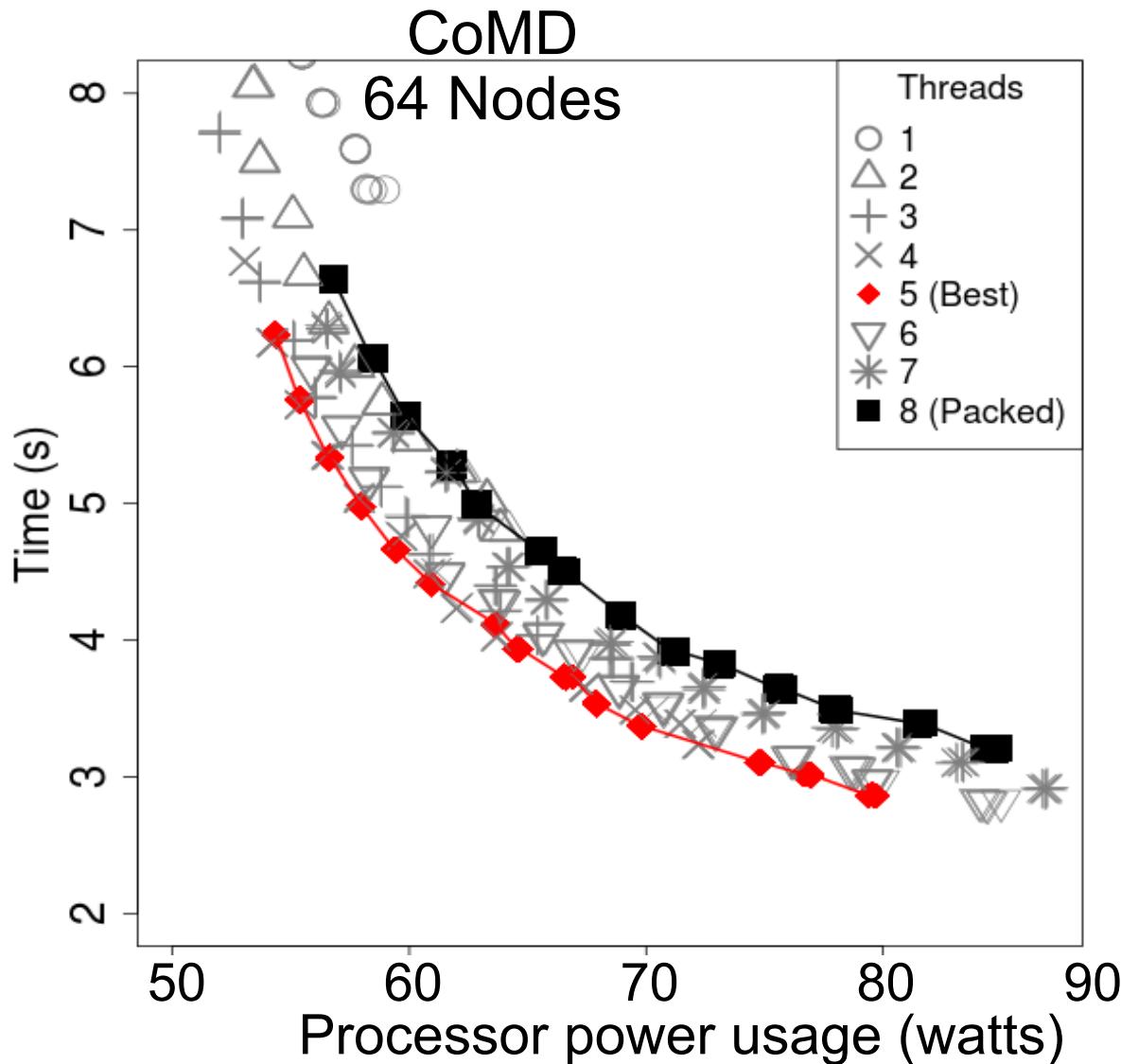
1. Trivial node-level configurations may be inefficient

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

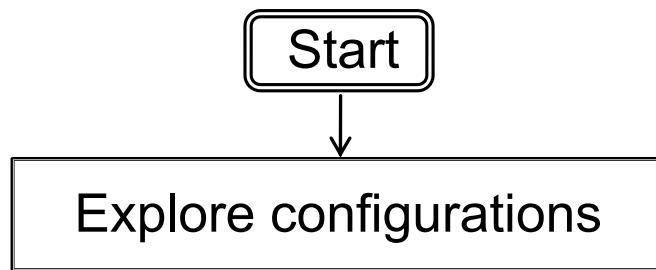


1. Trivial node-level configurations may be inefficient
 - Input: {# cores, frequency/power limit}
 - Output: {Execution time, power usage}
 - Up to 30% slower than the optimal configuration
 - Needs prohibitively large number of runs of the application
2. Portion of power left unused with load-imbalanced applications (up to 40%)

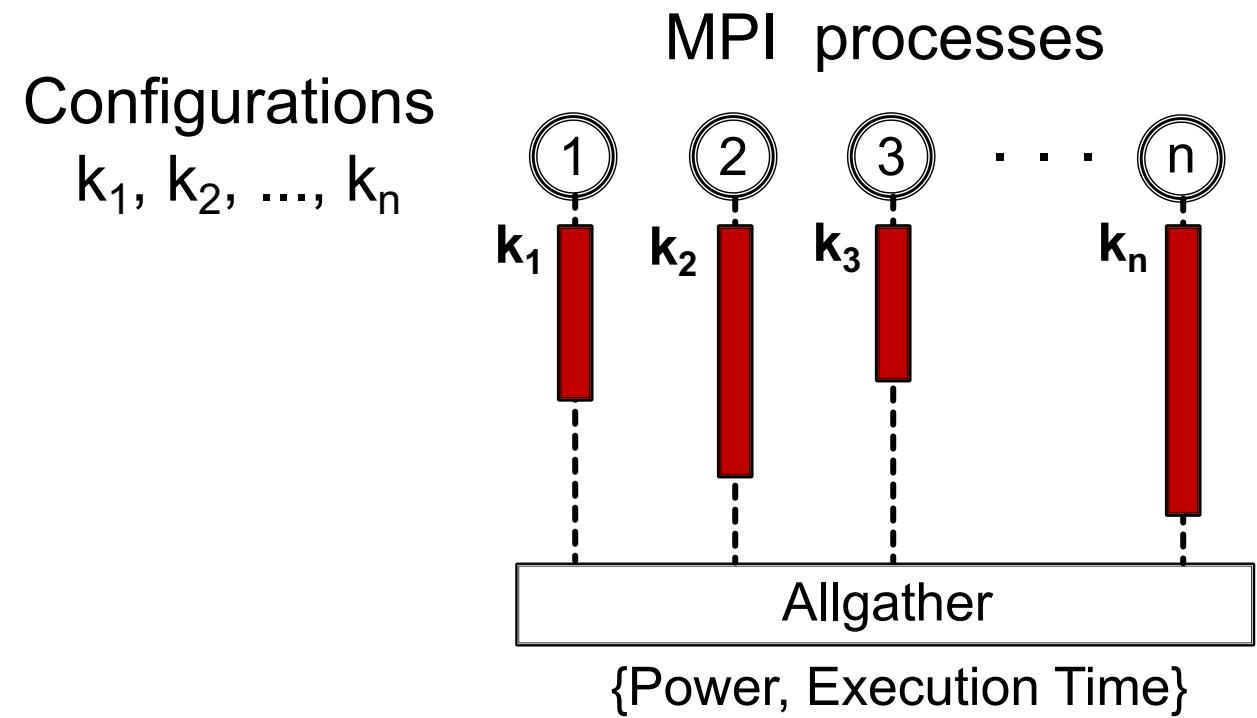
Conductor: Dynamic Configuration and Power Management

- 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

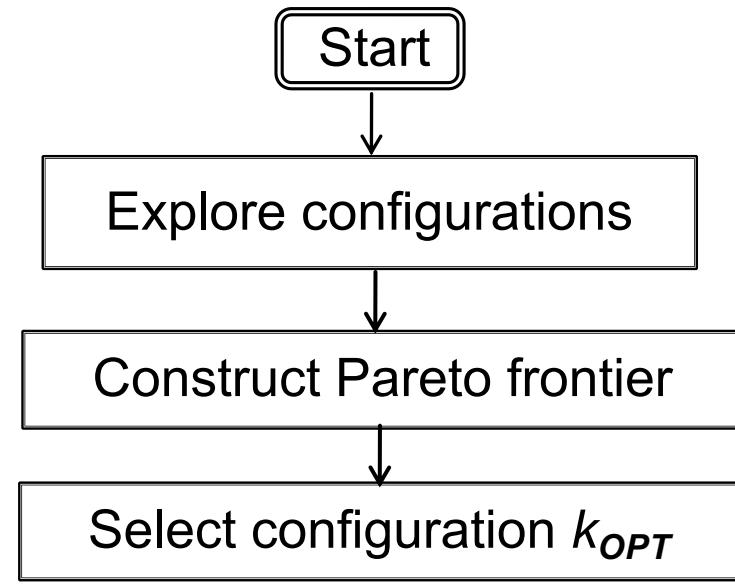
ConductorAgent Algorithm



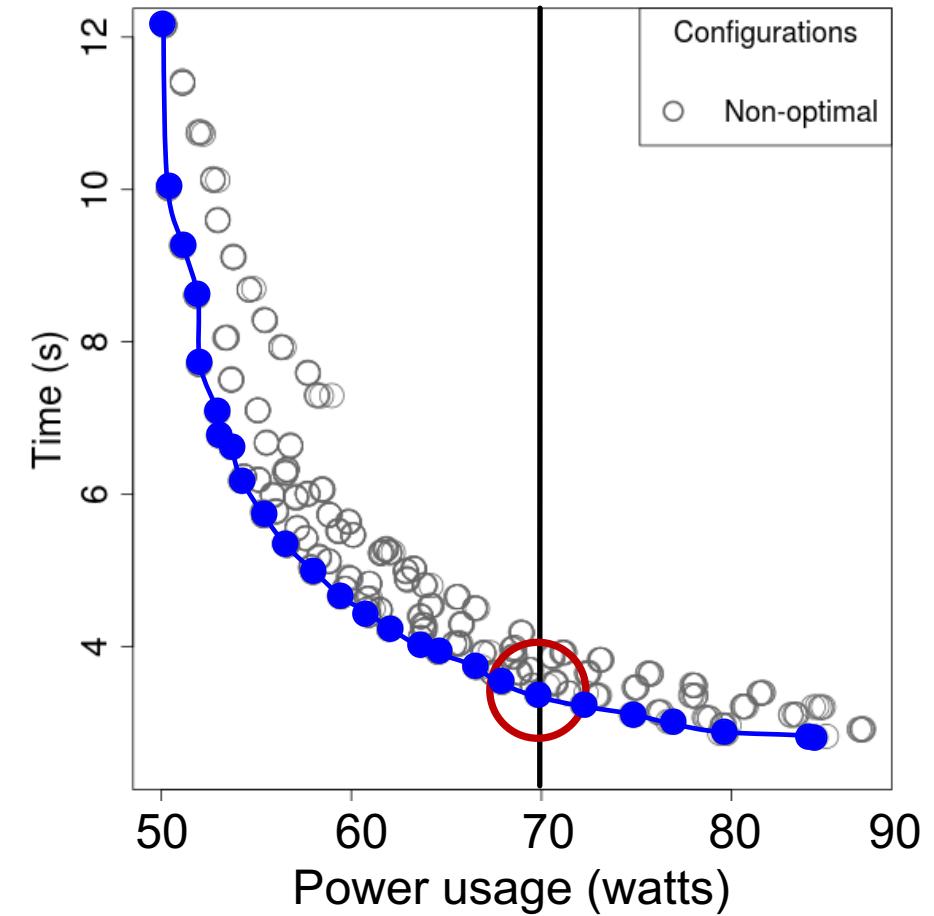
Step 1: Configuration Exploration



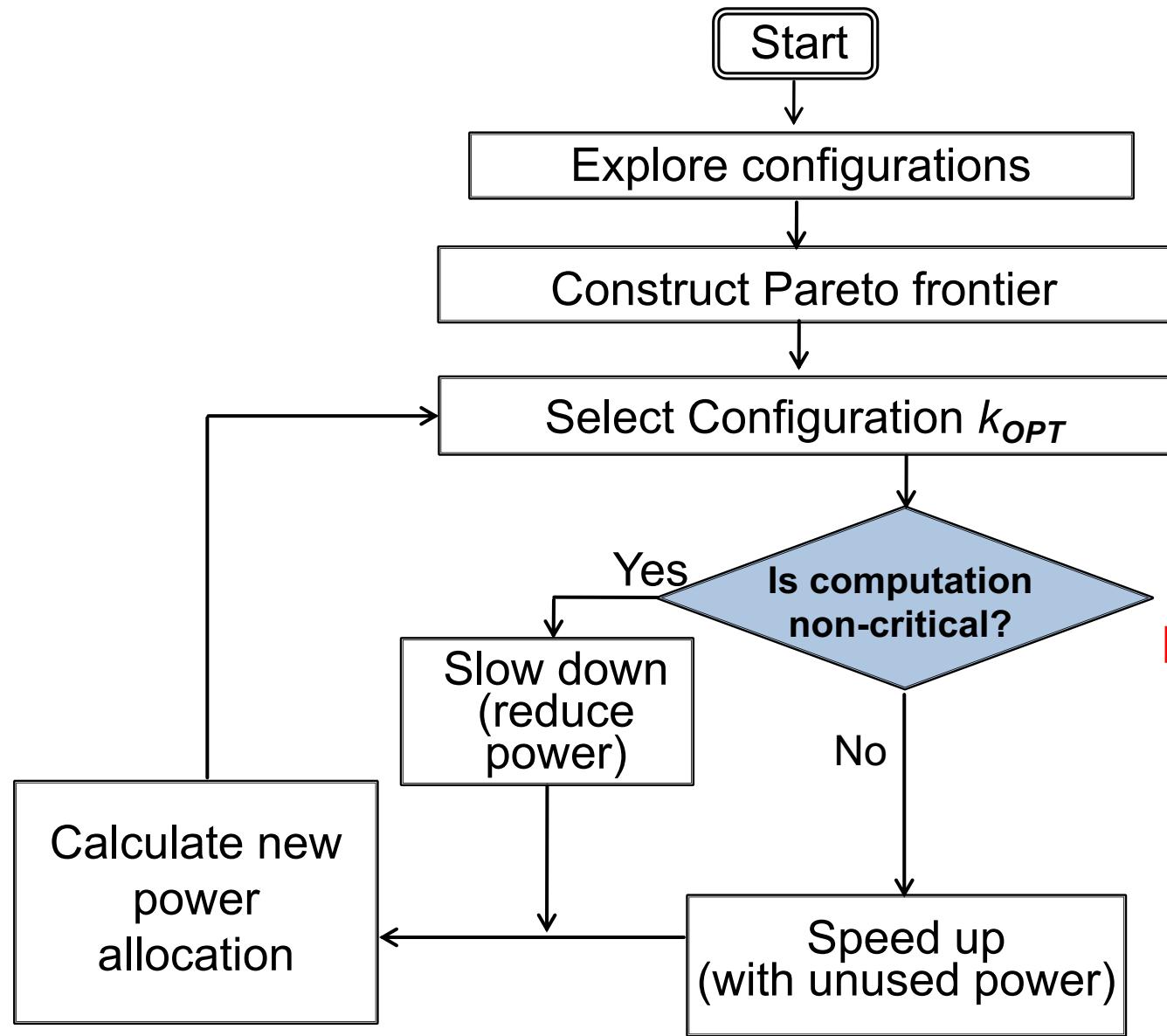
ConductorAgent Algorithm



Step 1: Configuration Exploration

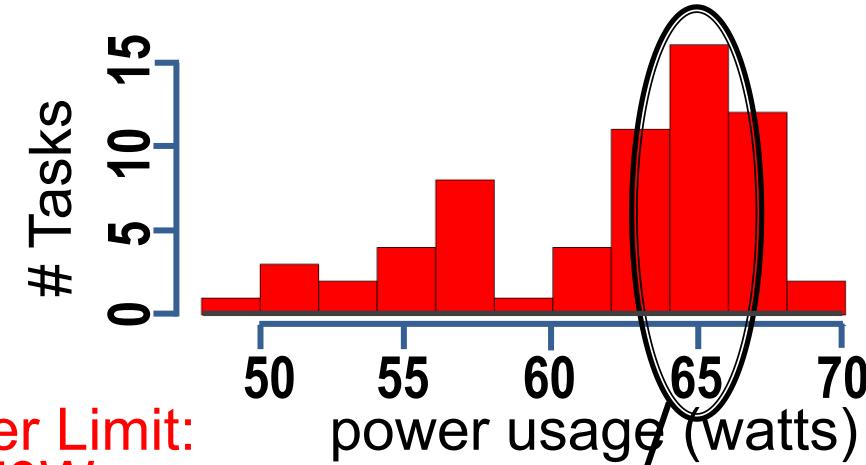


ConductorAgent Algorithm



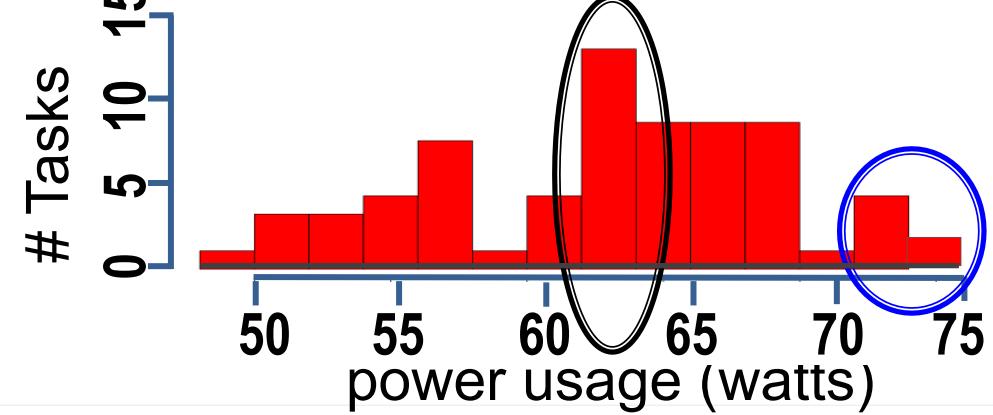
Step 2: Power Re-allocation

ParaDiS: Before power re-allocation



Power Limit:
70W

ParaDiS: After power re-allocation



Conductor: Integration into GEOPM

- **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

GEOPM::
SharedMemory

GEOPM Controller

ConductorAgent

Init
Handshake

Application Process

OMPT

Profiler

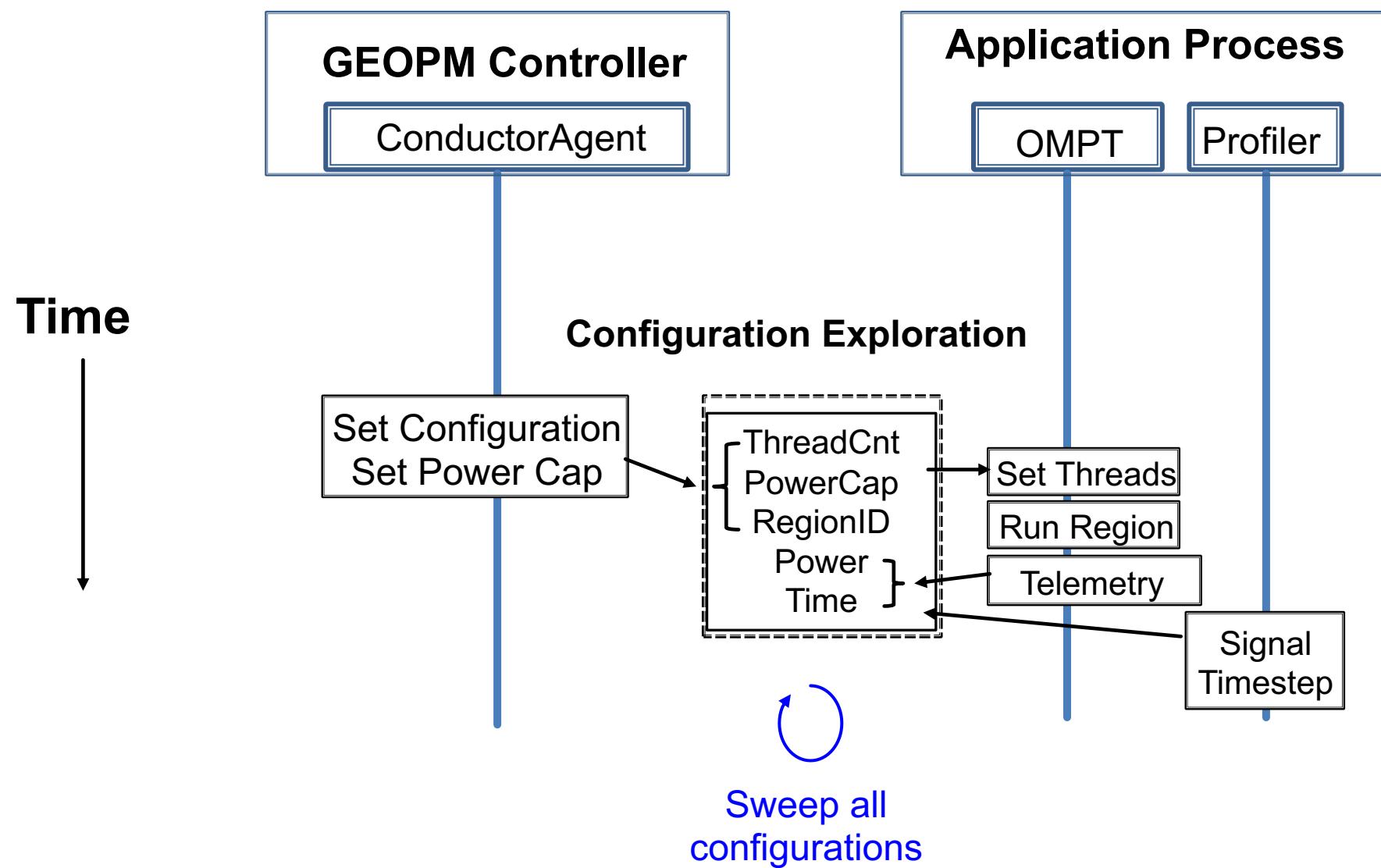
GEOPM::
SharedMemoryUser

Time

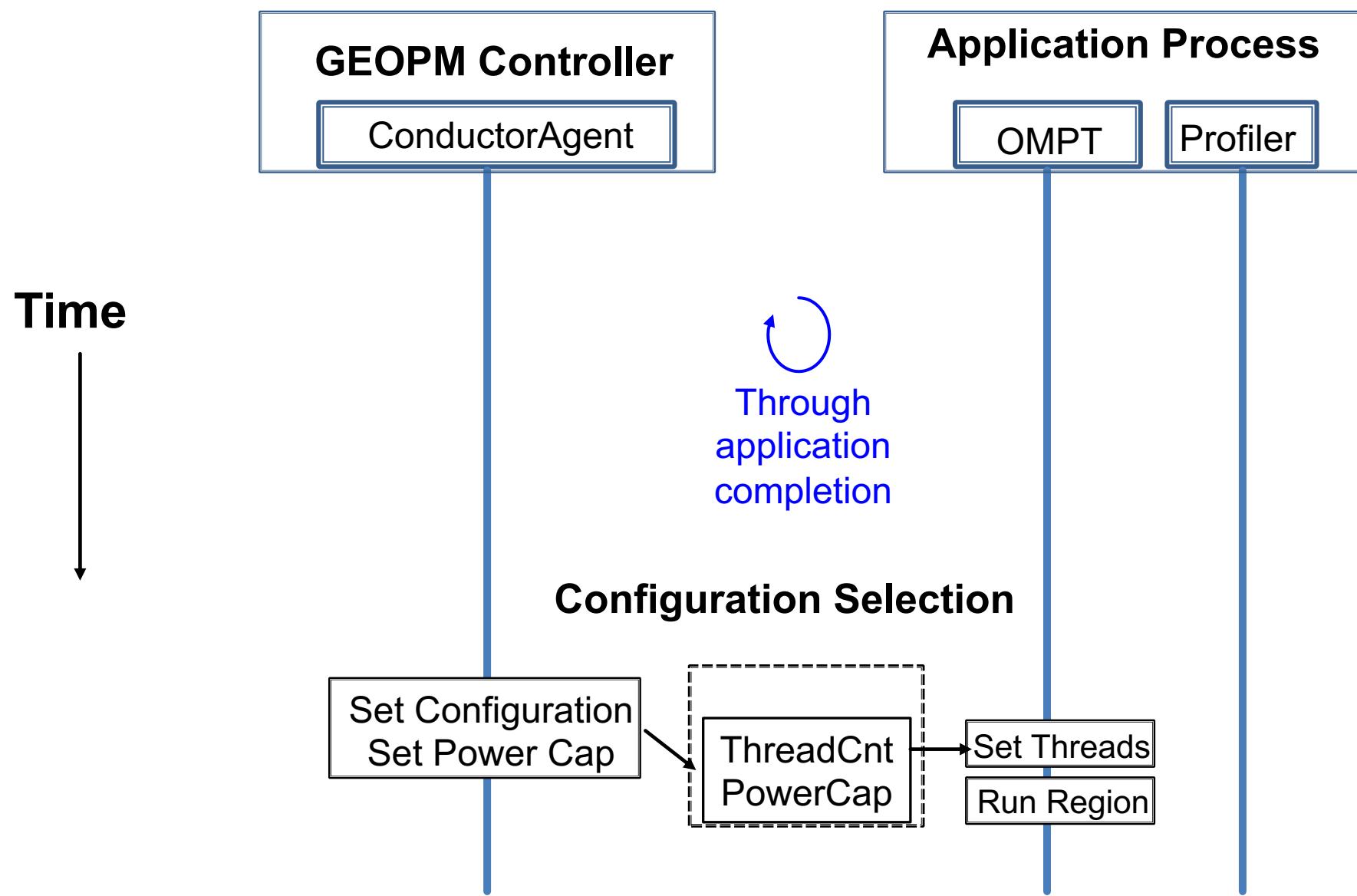
Shared
memory
space

Initialize control
and telemetry

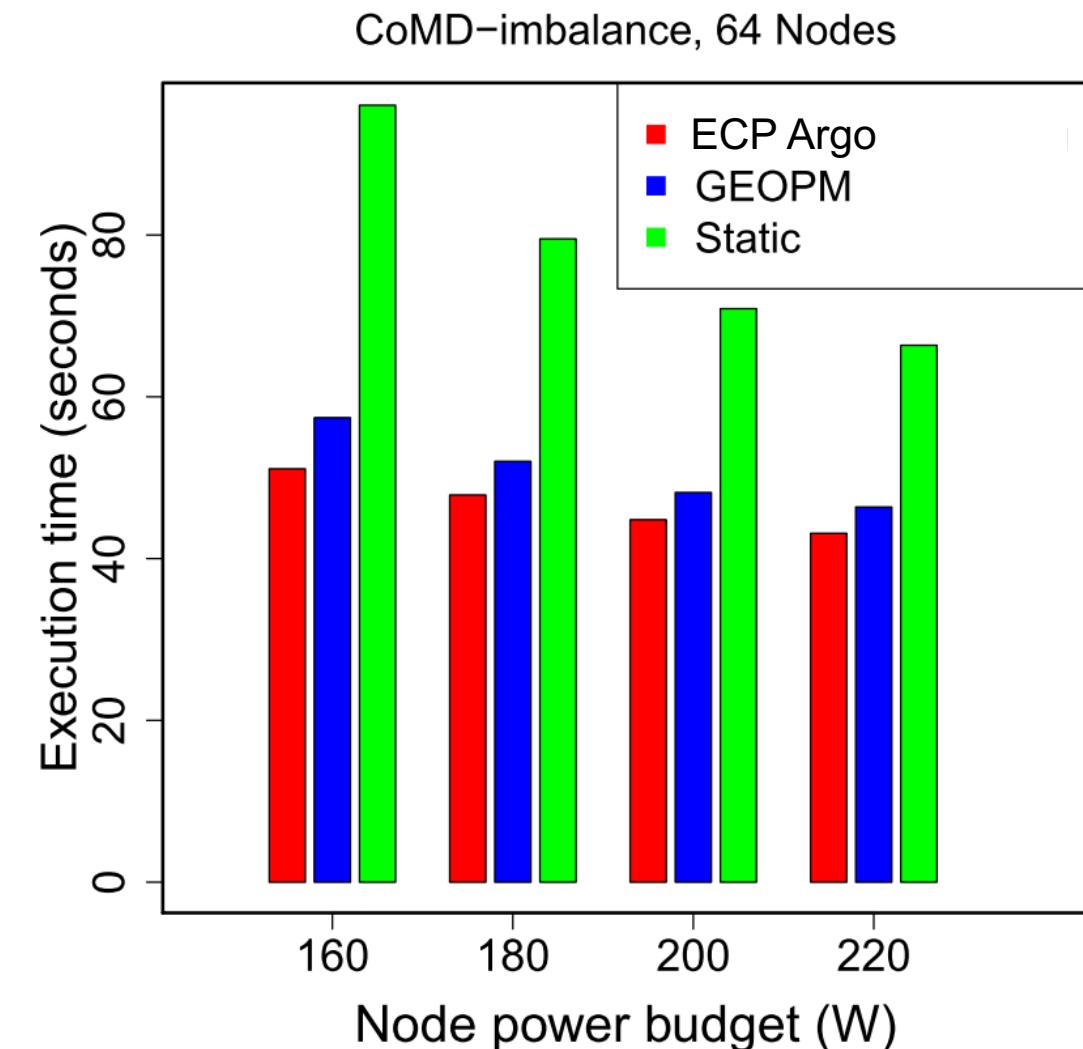
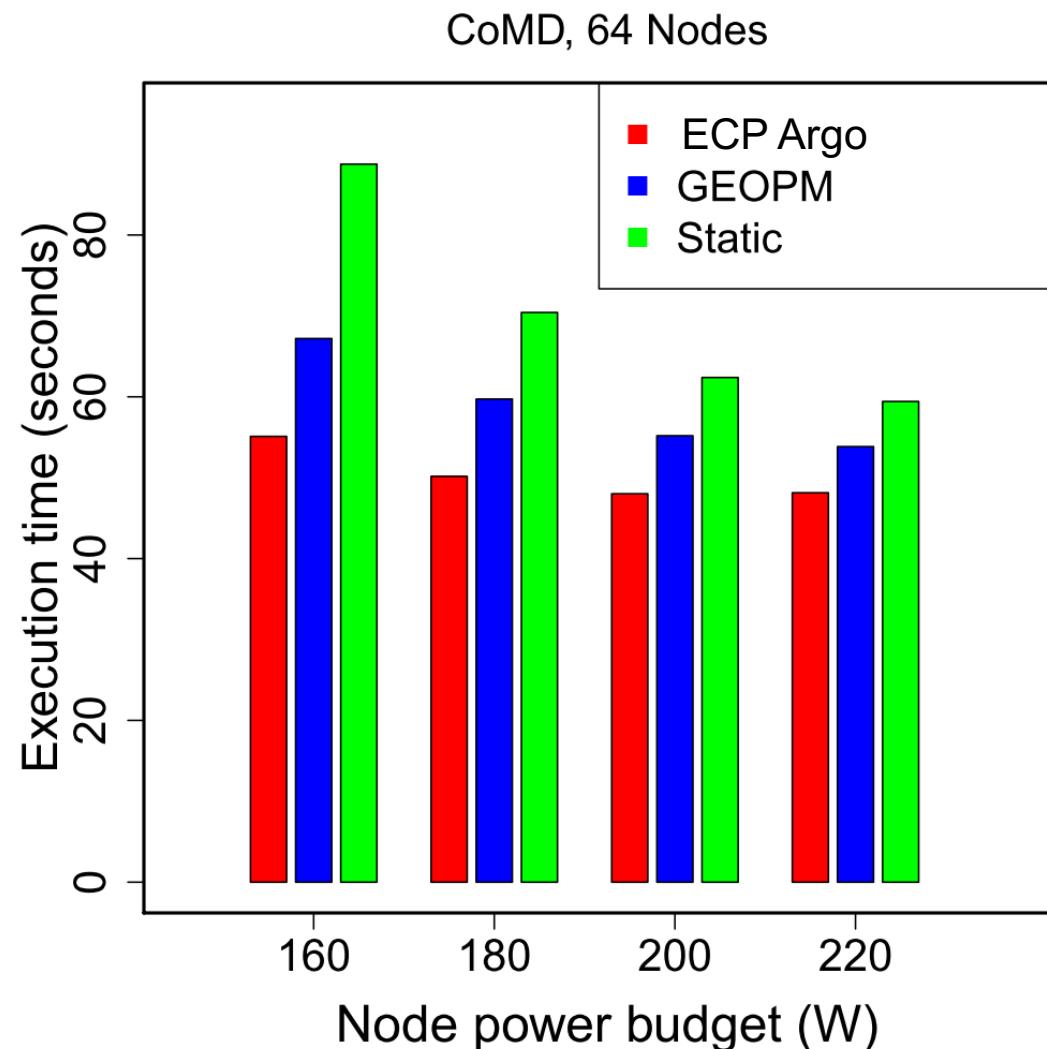
Configuration Exploration: Set Configuration, Collect Telemetry



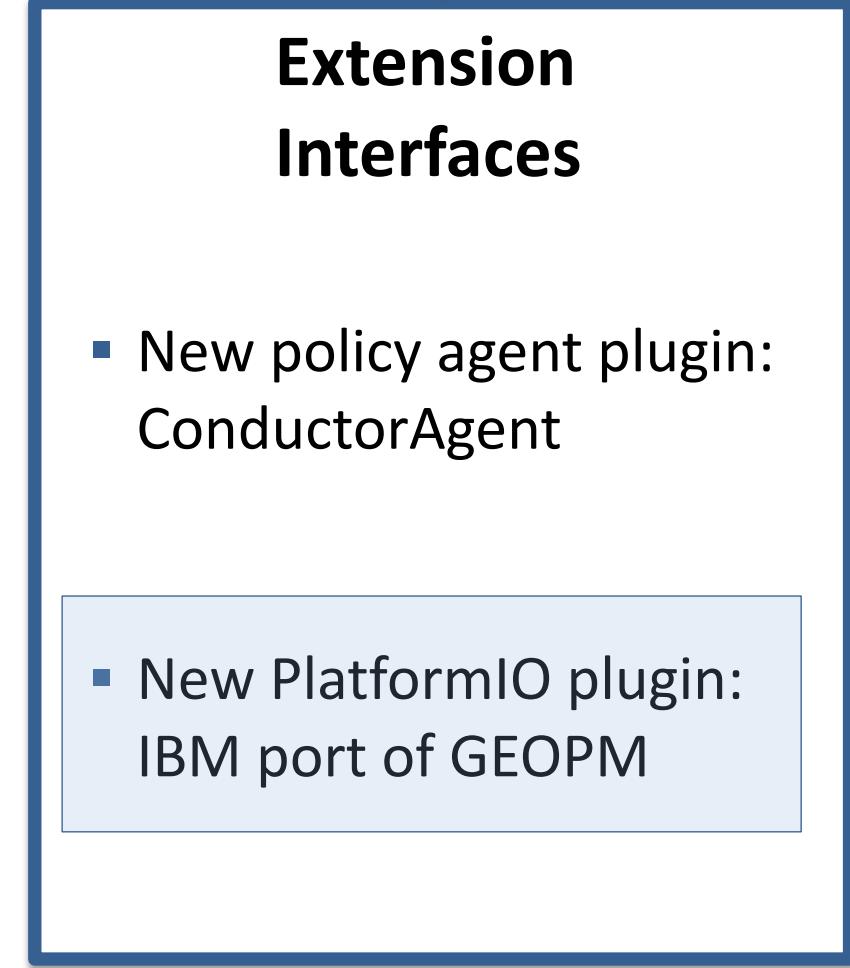
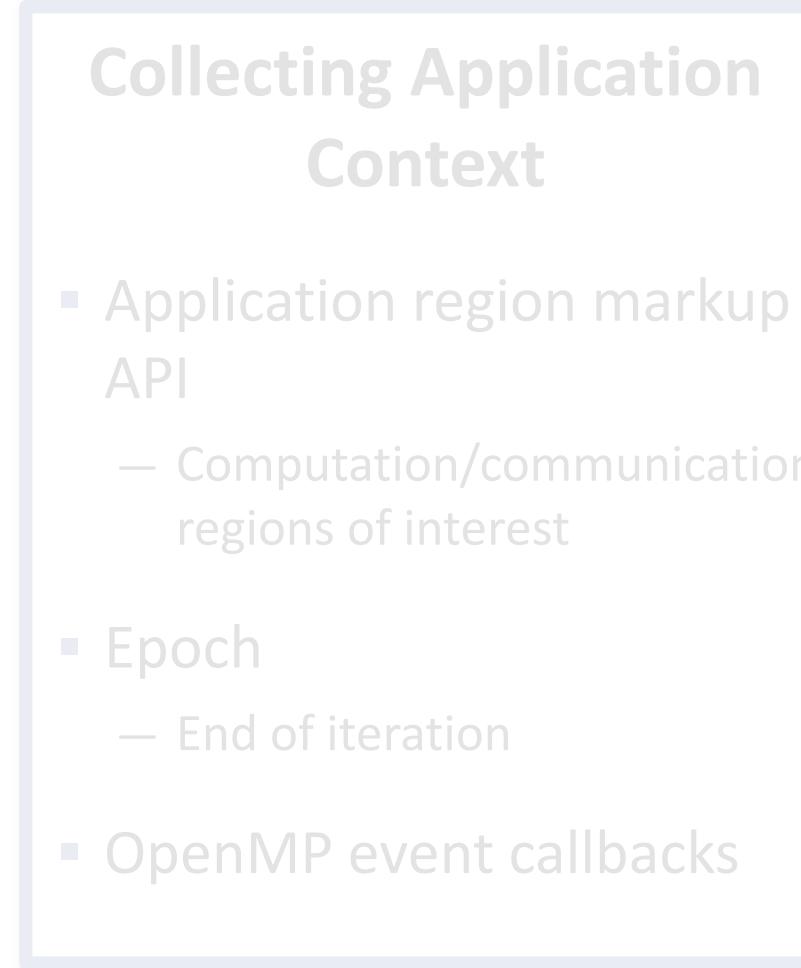
Configuration Selection: Pick Power-Efficient Configurations



ECP Argo: End Result



ECP Argo Contributions: Components and Interfaces



GEOPM Port: Migration to New GEOPM IOGroup Interface

Purpose	Old: PlatformImp interface	New: IOGroup interface
Get platform information on POWER9	PowerPlatformImp : extends <i>PlatformImp</i>	PowerIOGroup : extends <i>IOGroup</i>
RAPL-like monitoring and control on POWER9	OCCPlatform : extends <i>Platform</i>	PowerIO : Direct CPU monitoring/control interface
Get platform information on GPUs	NVMLPlatformImp : extends <i>PlatformImp</i>	NVMLIOGroup : extends <i>IOGroup</i>
RAPL-like monitoring and control on GPUs	NVMLPlatform : extends <i>Platform</i>	NVMLIO : Direct GPU monitoring/control

*Additional modifications in GEOPM Agent implementations to fully support GEOPM power management on POWER9 dual socket + Nvidia Volta with NVLink

GEOPM IBM Port: IBM “Witherspoon” Node

System configuration & interfaces

- CPU ID: PowerNV 8335-GTH, 2.2
- Number of cores: 160 4-way SMT, 3.7 GHz
- System memory: 66 GB
- GPU: Nvidia Tesla V100-SXM2
- Software: RHEL, GNU C/C++, GNU Fortran, MPICH2

We use linear regression-based model to predict power usage at a given CPU frequency

$$P_{CPU} = \alpha \cdot F + C$$

where, P_{CPU} : P9 CPU power usage (watts),

F : CPU frequency (GHz)

α : Coefficient of frequency scaling

C : Constant offset base frequency <-> power correlation

Telemetry

Control



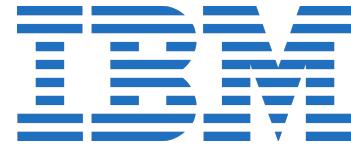
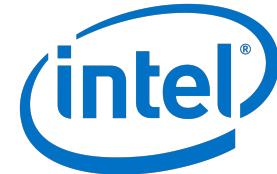
ECP Argo: Github Contributions

- Conductor integration and IBM platform plugin:
 - <https://github.com/geopm/geopm/pull/757>
- GEOPM integration with Caliper:
 - <https://github.com/LLNL/Caliper/pull/213>

GEOPM Team and 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



LLNL Team

Aniruddha Marathe
Tapasya Patki
Stephanie Brink
Barry Rountree

Questions?

Github links

Configuration Exploration: <https://github.com/amarathe84/geopm/tree/master>

IBM Port: <https://github.com/amarathe84/geopm/tree/ibm-port>



ECP Future Work

Done

- ECP Phase I: GEOPM extensions, Power-aware SLURM, Legion extensions
- ECP Phase II: Power control and monitoring through variorum, co-scheduling and workflows

WIP

- Deliver initial version of PowerStack
- Integrate GEOPM and Variorum
- Include node-level power capping after OPAL firmware update

Future

- Extend configuration exploration to include CPU-GPU configuration space
- Power/Performance models for co-scheduling and workflows
- Port variorum to ARM, HPE, and other architectures
- PowerStack Consortium and industry integration





**Lawrence Livermore
National Laboratory**

GEOPM Terms and Concepts

Launcher - Wrapper of system application launch (e.g. srun or aprun) to start GEOPM runtime

Endpoint - Shared memory interface between resource manager and GEOPM runtime

Agent - Extensible object determining samples/policies that interfaces with the PlatformIO

Policy - Array of floating-point settings for Agent specific control parameters in SI units

Sample - Array of floating-point values providing Agent specific runtime data in SI units

Controller - Thread on each compute node loads plugins and runs GEOPM algorithm

Report - Text file containing summary of aggregated stats collected during application run

Trace - Time series of signals collected over application run in pipe separated ASCII table

PlatformIO - High level interface for signals and controls enabling low level extension

Signal - Named parameter in SI units that can be measured using PlatformIO

Control - Named parameter in SI units that can be set using PlatformIO

Profile - Interface for annotating compute application; provides PlatformIO region signals

Supplementary Slides



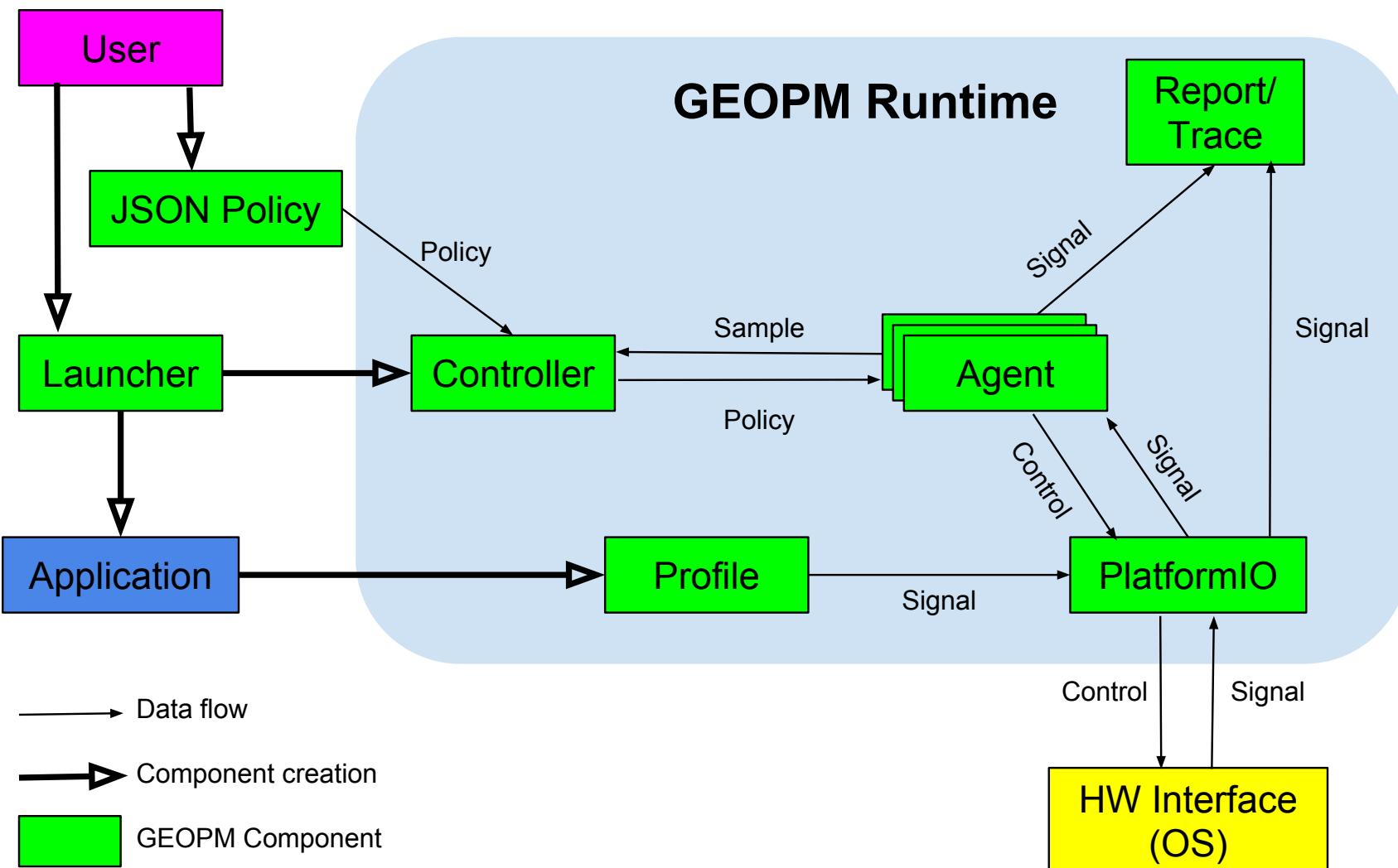
Future Work

- Short-term:
 1. Include node-level power capping after OPAL firmware update
 - Make the agent implementations aware of node-level power controls
 2. Algorithm for explicit power assignment of CPU and GPU components and compare with the node-level power assignment implementation
- Medium-to-long term:
 1. Enable user-level access to power control and monitoring through **Libvariorum**
 2. Extend configuration exploration to include CPU-GPU configuration space
 3. Evaluate the port for realistic, load-imbalanced applications.

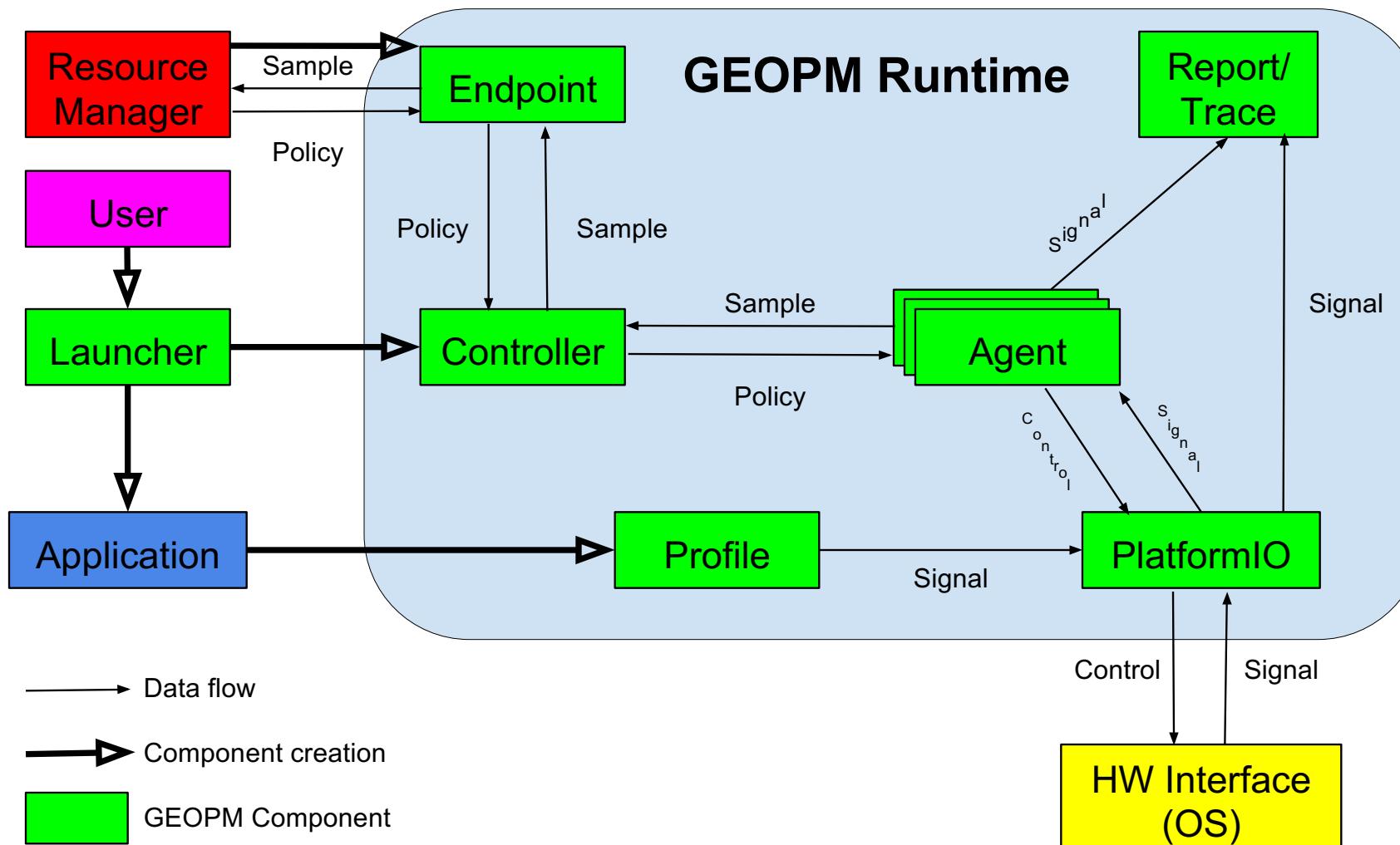
GEOPM IBM Port: Evaluation

- Single-node tests:
 - The PowerIO and NVMLIO assigned component power as expected
 - Governed policy enforced power budget on CPU and GPU for a synthetic workload
- On a 4-node setup with no power control (monitoring-only mode):
 - GEOPM port generated CPU power usage traces for synthetic workload

GEOPM Runtime Diagram: User Driven



GEOPM Runtime Diagram: Resource Manager Driven



GEOPM Profiling Support

- Report
 - Generates aggregated report at end of runtime (one file per run)
 - Sections in report broken down per compute node and per application region
 - Agent can customize each section of the report for data it aggregates
- Trace
 - Time series of application profile throughout runtime (one file per node)
 - Samples created periodically based on the Agent::wait() method
 - Each sample contains hardware counters, application feedback and Agent signals
 - Traced signals can be extended with the --geopm-trace-signals launcher option
 - User can select any valid PlatformIO supported signal including IOGroup plugins
 - More information in [geopm\(7\)](#)
- Endpoint (in-progress)
 - Live data aggregated across the entire set of compute nodes used by application
 - Updated with Agent specific signal aggregation technique to produce samples



GEOPM Profile Analysis

- The geopmio module parses reports and traces to pandas dataframe
- Visualization: [geopmplotter\(1\)](#)
- Deeper analysis: [geopmanalysis\(1\)](#) combines app launch and data visualization
- User-customized interactive sessions can manipulate and visualize collected data
- Feature tests of custom Agents can be easily created with python infrastructure

GEOPM IBM Port: Migration to New GEOPM IOGroup Interface

Purpose	Old: PlatformImp interface	New: IOGroup interface
Get platform information on POWER9	PowerPlatformImp : extends <i>PlatformImp</i>	PowerIOGroup : extends <i>IOGroup</i>
RAPL-like monitoring and control on POWER9	OCCPlatform : extends <i>Platform</i>	PowerIO : Direct CPU monitoring/control interface
Get platform information on GPUs	NVMLPlatformImp : extends <i>PlatformImp</i>	NVMLIOGroup : extends <i>IOGroup</i>
RAPL-like monitoring and control on GPUs	NVMLPlatform : extends <i>Platform</i>	NVMLIO : Direct GPU monitoring/control

*Additional modifications in GEOPM Agent implementations to fully support GEOPM power management on POWER9 dual socket + Nvidia Volta with NVLink

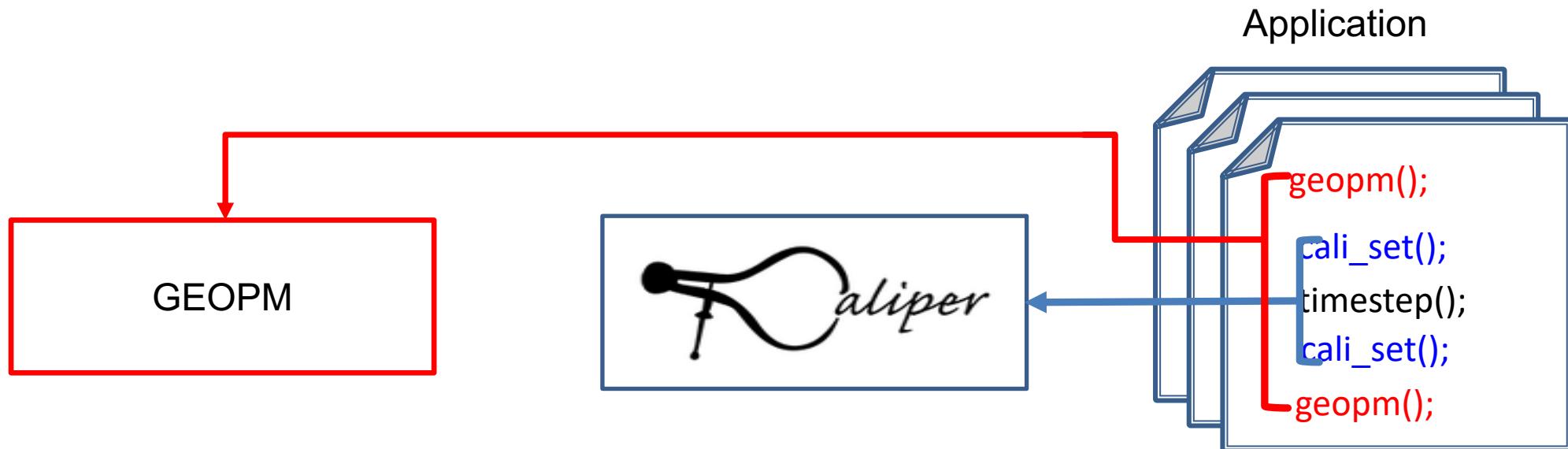
GEOPM IBM Port: IBM “Witherspoon” Node

System configuration & interfaces

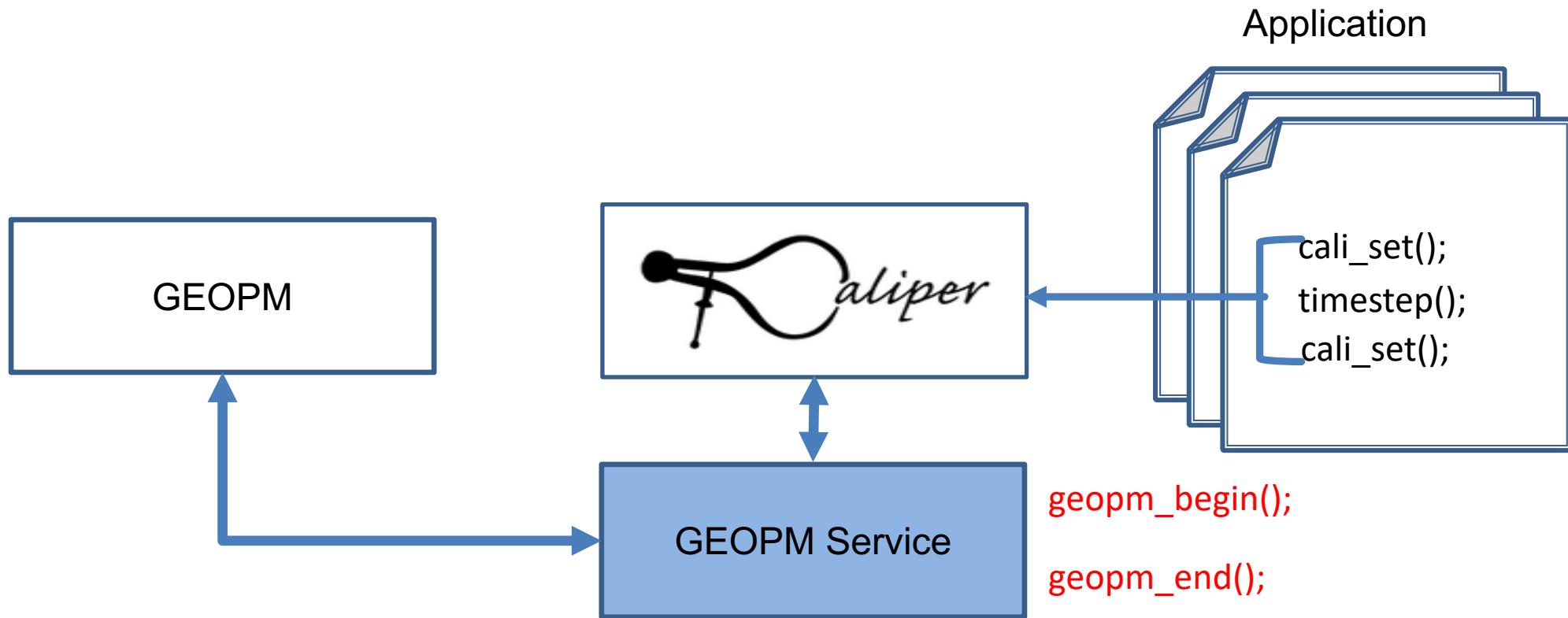
- CPU ID: PowerNV 8335-GTH, 2.2
- Number of cores: 160 4-way SMT, 3.7 GHz
- System memory: 66 GB
- GPU: Nvidia Tesla V100-SXM2
- Software: RHEL, GNU C/C++, GNU Fortran, MPICH2

Telemetry	CPU frequency	<ul style="list-style-type: none">▪ <code>/sys/devices/system/cpu/cpufreq/policy*/scaling_cur_freq</code>
	CPU Sensors	<ul style="list-style-type: none">▪ <code>/sys/firmware/opal/exports/occ_inband_sensors</code>▪ Performance Monitoring library (perfmon2): libpfm4
	GPU information	<ul style="list-style-type: none">▪ NVML :: *
Control	CPU frequency	<ul style="list-style-type: none">▪ <code>/sys/devices/system/cpu/cpufreq/policy*/scaling_setspeed</code>
	GPU power limit	<ul style="list-style-type: none">▪ NVML :: <code>nvmIDeviceSetPowerManagementLimit()</code>
	Node-level power capping	<ul style="list-style-type: none">▪ <code>/sys/firmware/opal/powercap/system-powercap/powercap-current</code>

ECP Argo: Software Components

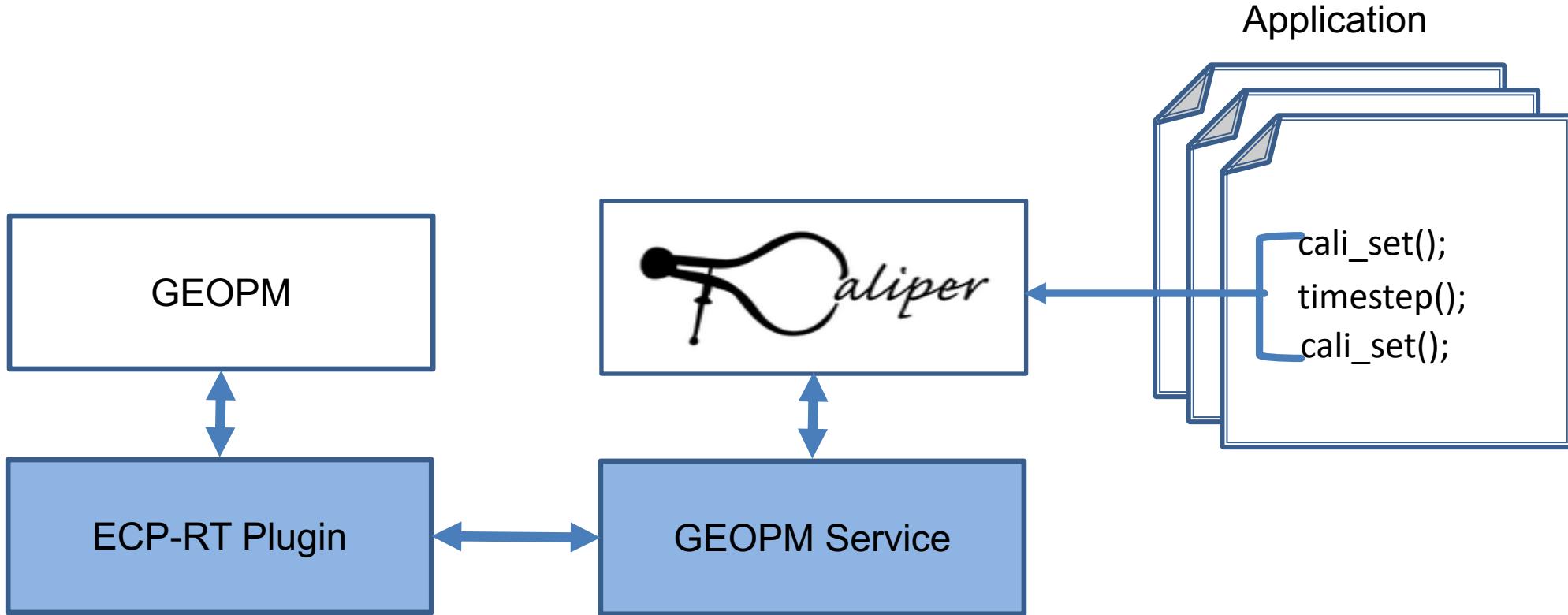


ECP Argo: Software Components



1. Integrate **Caliper** framework into the ECP software stack
 - Avoid duplication of markups
 - Expose application context in Caliper to GEOPM for power management

ECP Argo: Software Components



2. Integrate core functionalities of [Conductor](#) with GEOPM through the Caliper service and GEOPM plugin interface

ECP Argo: Selecting Power-Optimizing Configuration

- **Objective:** Fine-tune application configuration under a power cap
- **ECP Argo Solution:**
 - Augment GEOPM's algorithm with performance-optimizing application configurations: # threads, Frequency, etc.
 - Port GEOPM to IBM POWER9 (support for LLNL Sierra)

ECP Argo: Configuration Exploration

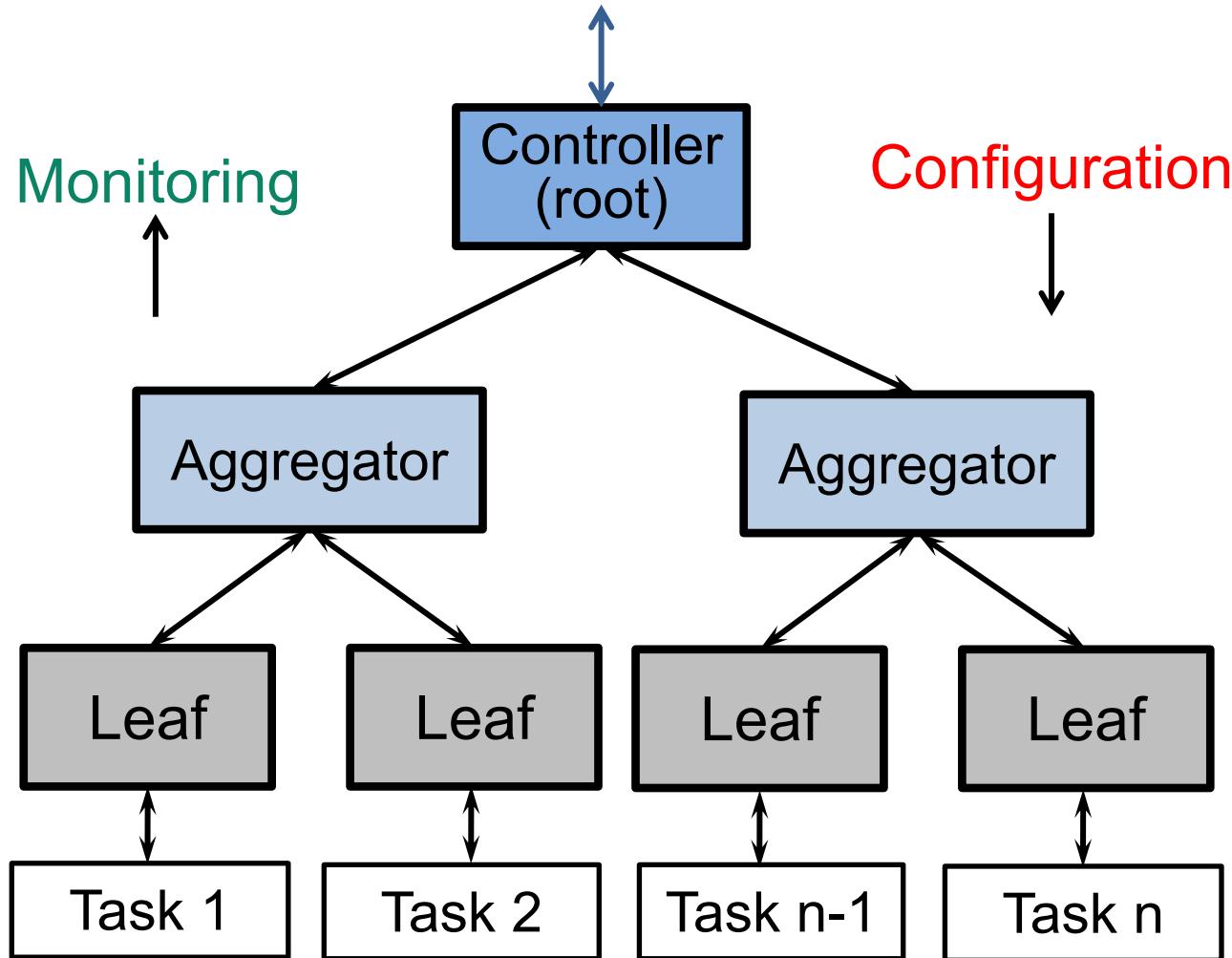
1. Interface **Caliper** framework with GEOPM in the ECP software stack
 2. Migrate core functionalities of **Conductor** (a research-based code) into GEOPM as plugins.
- Components
 - **Caliper**: Application context-aware monitoring library (David Boehme, SC'16) ECP SPCM147.
 - **Conductor**: Power-optimizing runtime system by LLNL (Marathe et al., ISC'15)

Mapping GEOPM and Caliper Markups

- Most one-to-one mapping of attribute are possible
- Need to follow naming conventions for non-supported attributes
 - For example, GEOPM region ‘hint’
- Developed a mapping document to formalize the mapping process with agreement between GEOPM and Caliper team
- [Mapping](#) between GEOPM and Caliper markups
- Components: [Caliper component](#)
[GEOPM component](#)

Intel GEOPM: Global Extensible Open Power Manager

Interface with resource scheduler

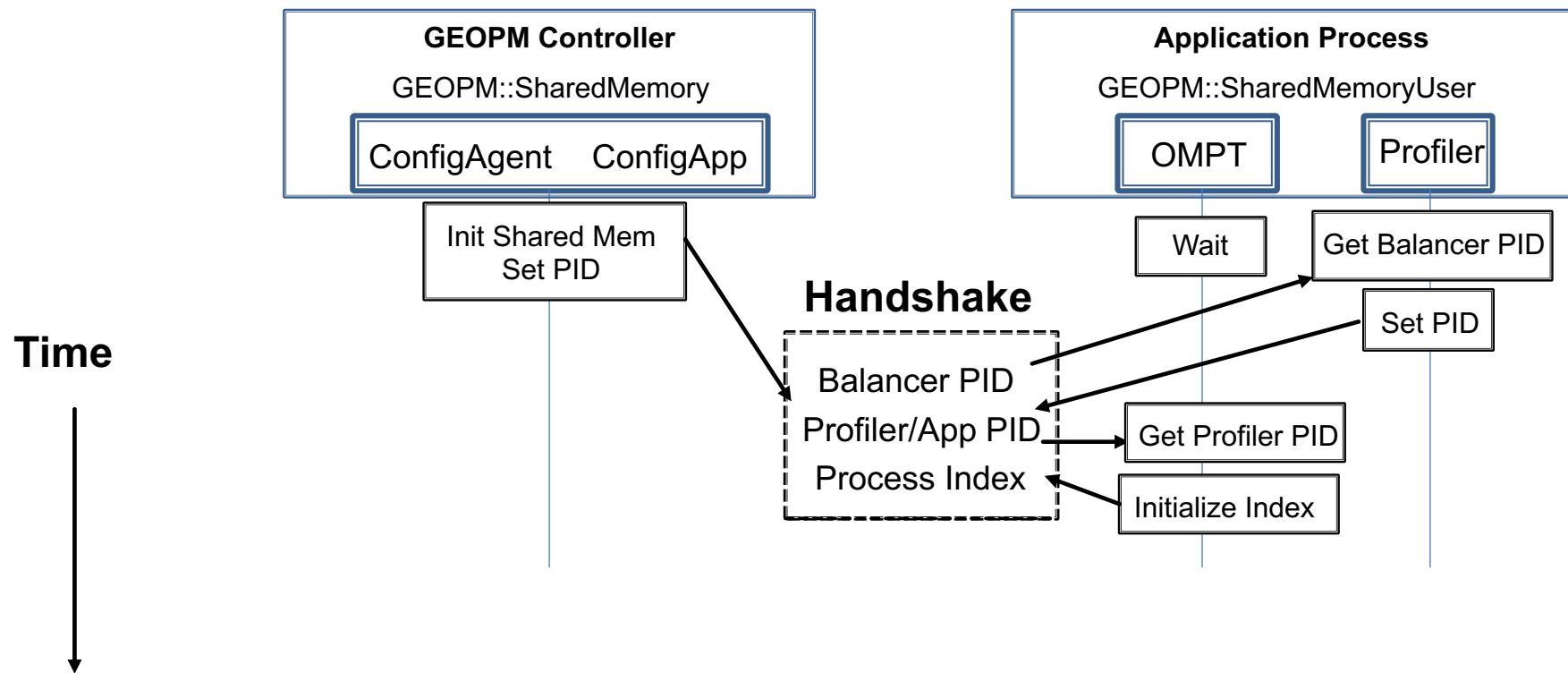


- Important components
 - **Caliper**: Application context-aware monitoring library (Boehme et al. SC'16)
 - **GEOPM**: Intel's open-source, hierarchical runtime system
1. Interface Caliper framework with GEOPM in the ECP software stack
 2. Interface ECP Argo with power-aware scheduler.

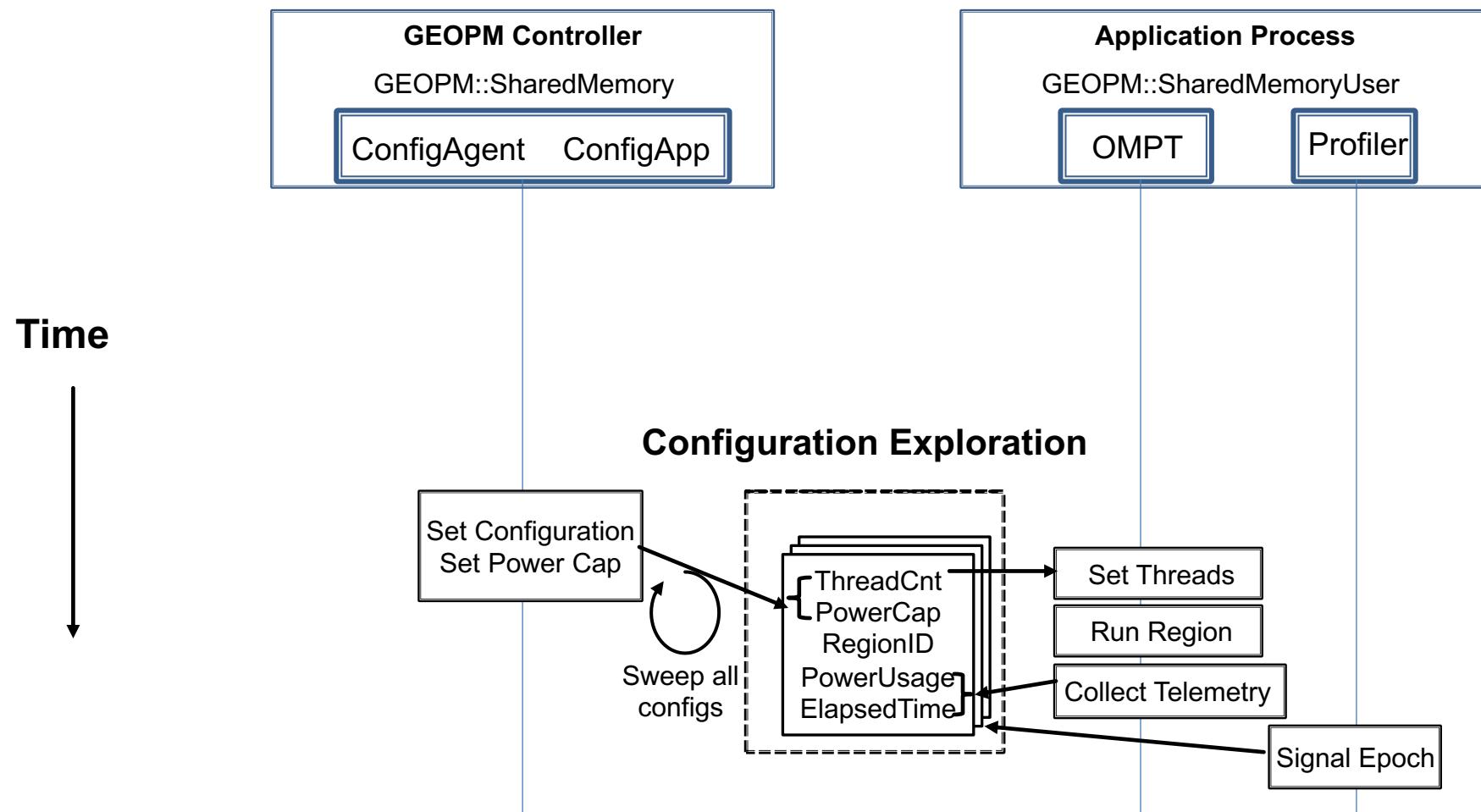
GEOPM 1.0 Interfaces

Purpose	IOGroup/PlatformIO	Agent
Get platform information	MSRIOGroup : extends <i>IOGroup</i>	
Power monitoring and control (hardware interface)	MSRIO : Direct CPU monitoring/control interface	
Interfacing with the resource scheduler	JSON : user-specified configuration at launch Endpoint : interface with resource-scheduler	

Initialization: GEOPM, Application Handshake



Configuration Exploration: Collect Phase-level Profiles



Configuration Selection: Select Power-Optimizing Configuration

