Managing Power Efficiency of HPC Applications with Variorum and GEOPM

ECP Tutorial

Feb 4, 2020 2:30PM-6PM

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

LLNL-PRES-804125

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

Lawrence Livermore National Laboratory

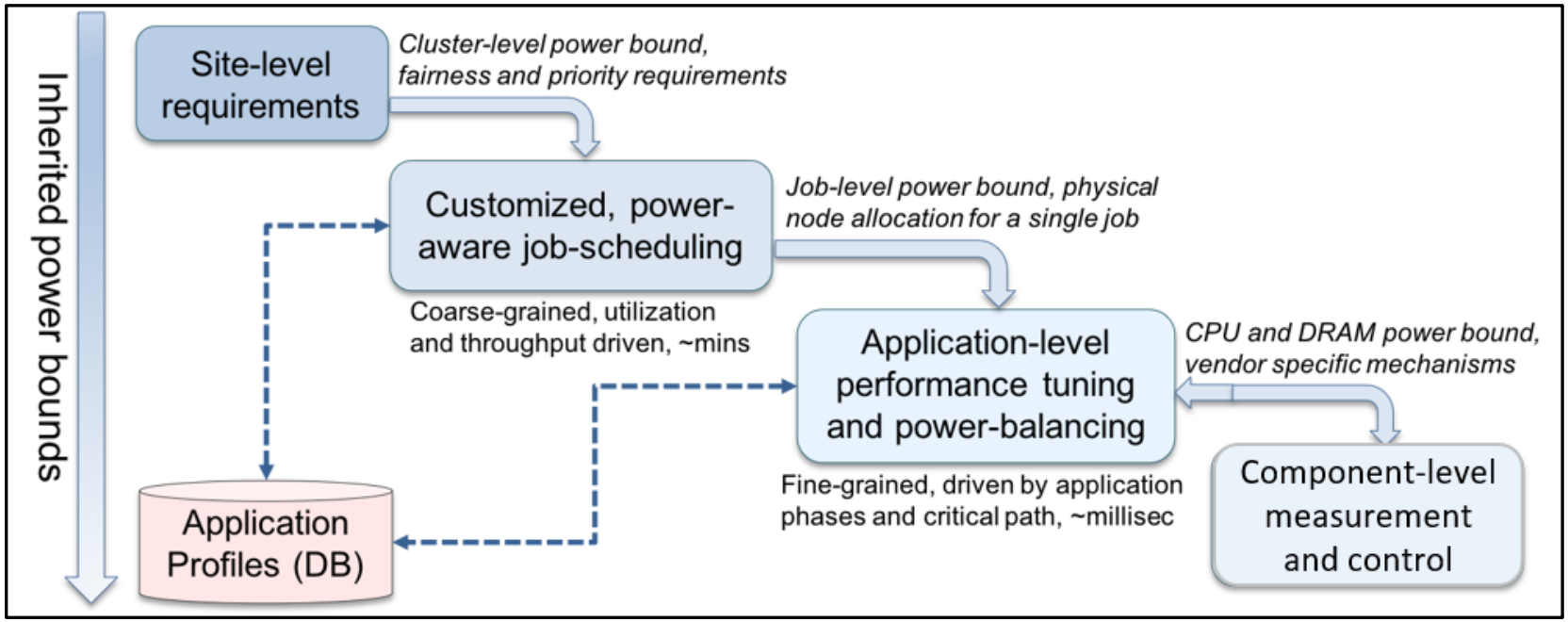
Welcome: HPC Systems and Power Management

- Compute Nodes, I/O Nodes, Network
- Static and Dynamic Power Management
- DOE's ongoing project: ECP Argo

Agenda

Topic	Time Slot	Presenters
PowerStack Introduction and ECP Challenges	2:30 – 2:50	Tapasya Patki
Workflows and site-level power management	2:50 – 3:10	Tapasya Patki
Power Control Knobs on Intel and IBM systems	3:10 – 3:30	Tapasya Patki
BREAK	***3:30 – 4:00***	
Variorum	4:00 – 4:30	Stephanie Brink
Hands-on Tutorial on GEOPM	4:30 – 4:45	Aniruddha Marathe
GEOPM Agent/Platform API	4:45 – 5:15	Aniruddha Marathe
LLNL's advanced plugins (DVFS-based IBM plugin, configuration selection plugin)	5:15 – 5:50	Aniruddha Marathe
Wrap up	5:50 – 6:00	All

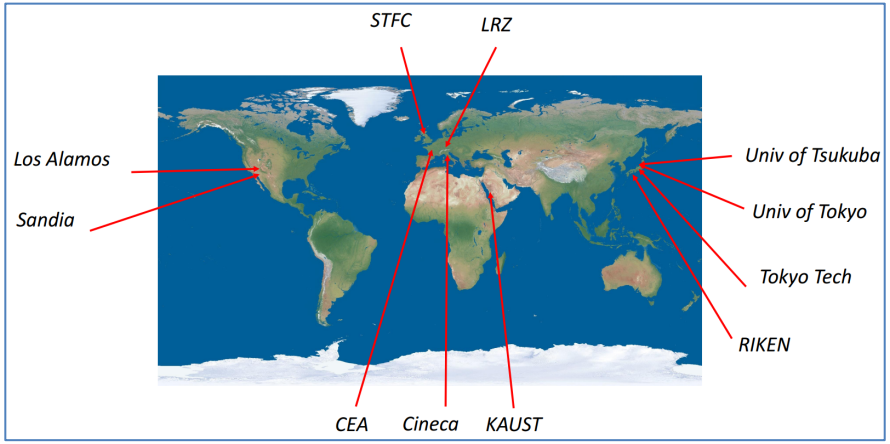
PowerStack: System-wide, dynamic power management



https://hpcpowerstack.github.io/

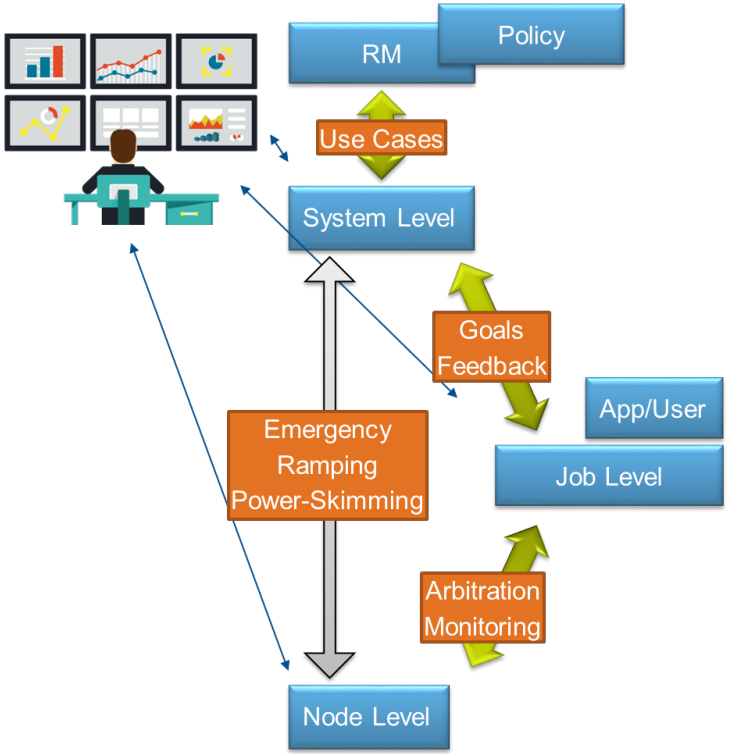
PowerStack: Stakeholders

- Current industry collaborators: Intel, IBM, AMD, ARM, Cray, HPE, Fujitsu, Altair, ATOS/Bull
- 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 will be developed as part of ECP using SLURM, GEOPM, libmsr/msr-safe (close collaboration with Intel)



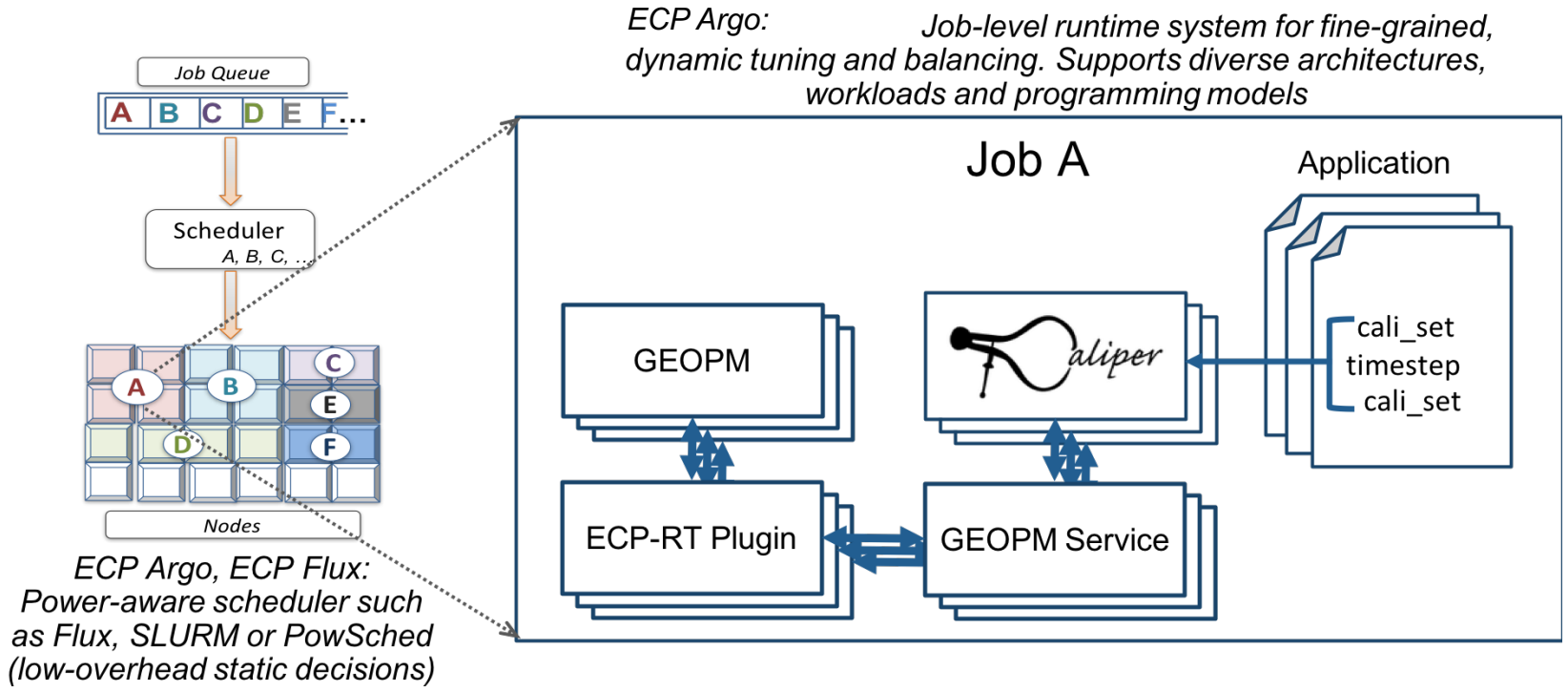
EEHPC-WG's insight into sites investing in Energy- and Power-aware Job Scheduling and Resource Management (EPA-JSRM)

PowerStack: Layers

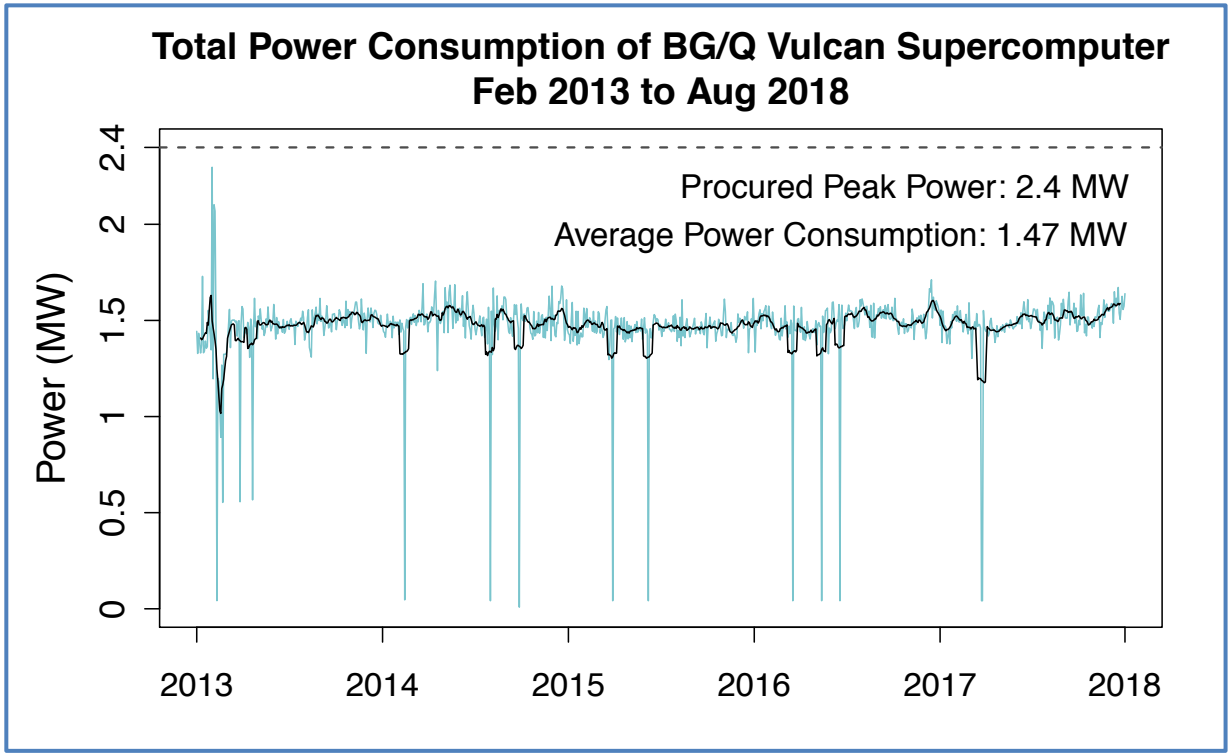


- Site-level
  - Policies, objectives
- System-level
  - Scheduling, runtimes, application-awareness
- Platform-level
  - Hardware interaction, PowerAPI
- Idle and Emergency Management

ECP Argo uses SLURM/Flux, GEOPM and variorum as vehicles for power management at exascale

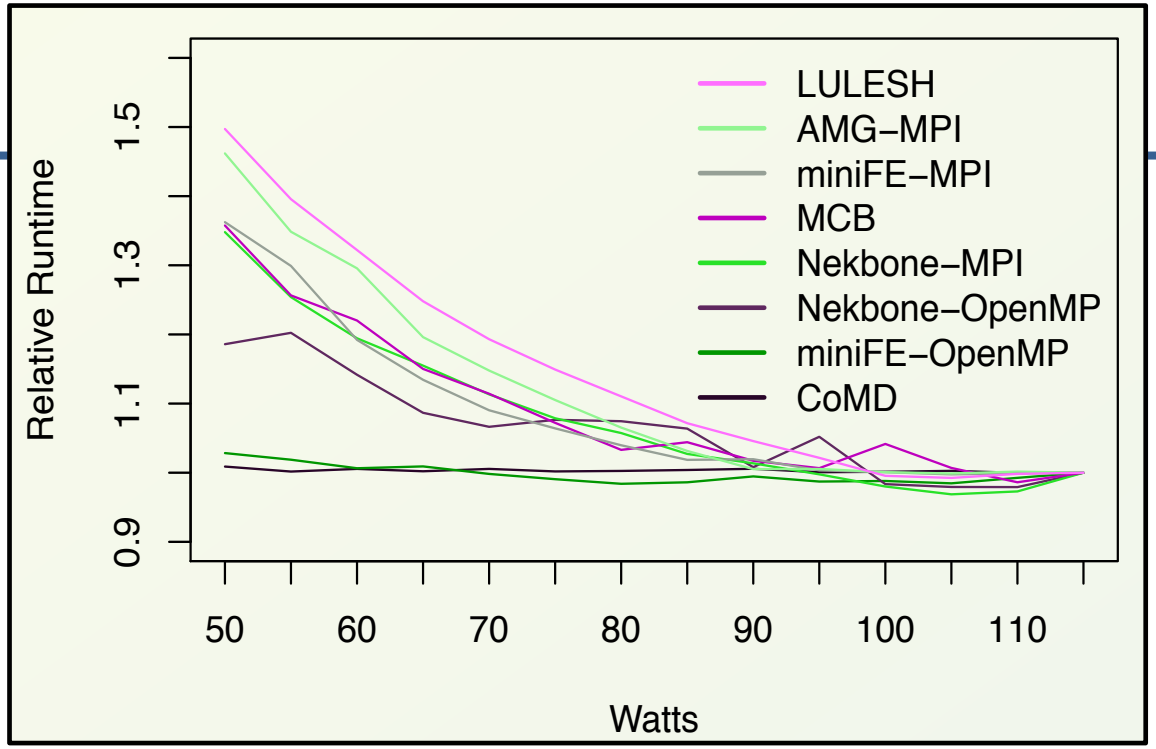


Unused Power: 40%



Why is it so?

- Applications have different memory, communication, I/O requirements and phase behaviors
- Applications don't utilize all of the allocated power, thus allocating more power doesn't always improve performance

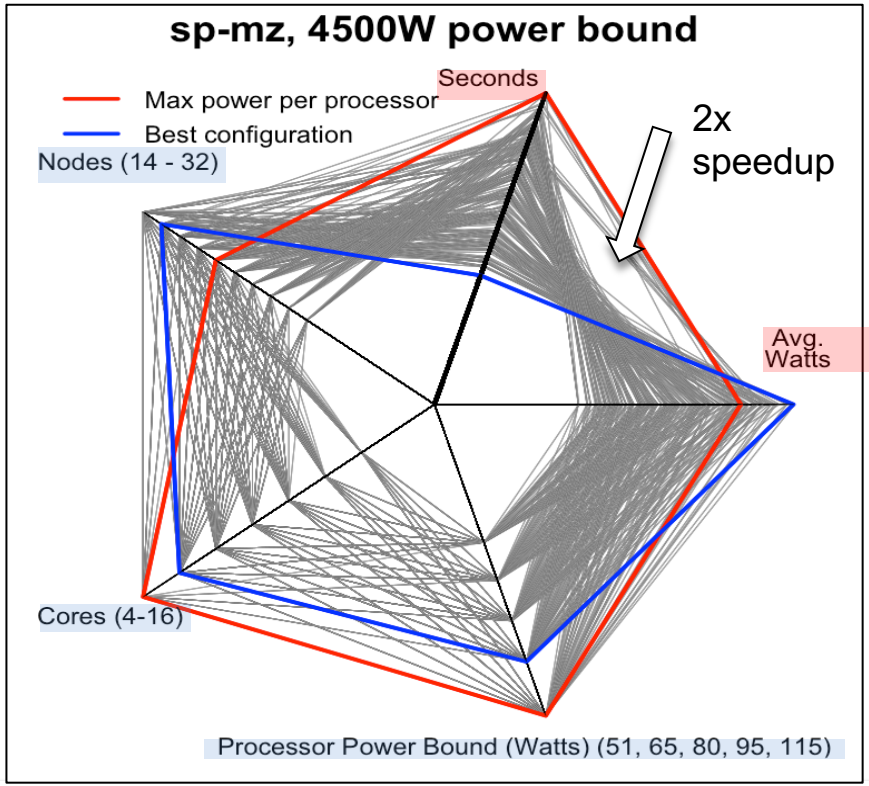


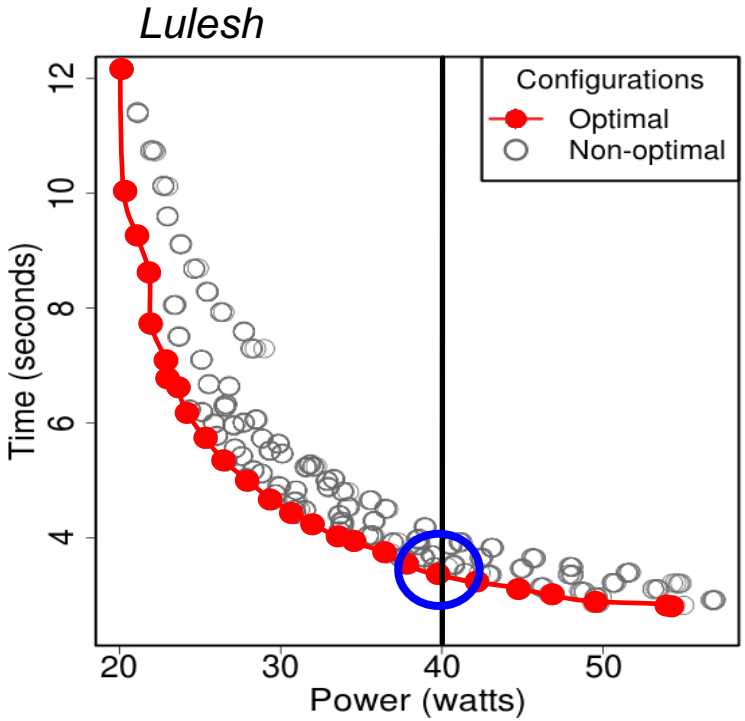
Intel Sandy Bridge, 8 nodes, (2 sockets, 8 cores)

Min: 51 W, Max: 115 W

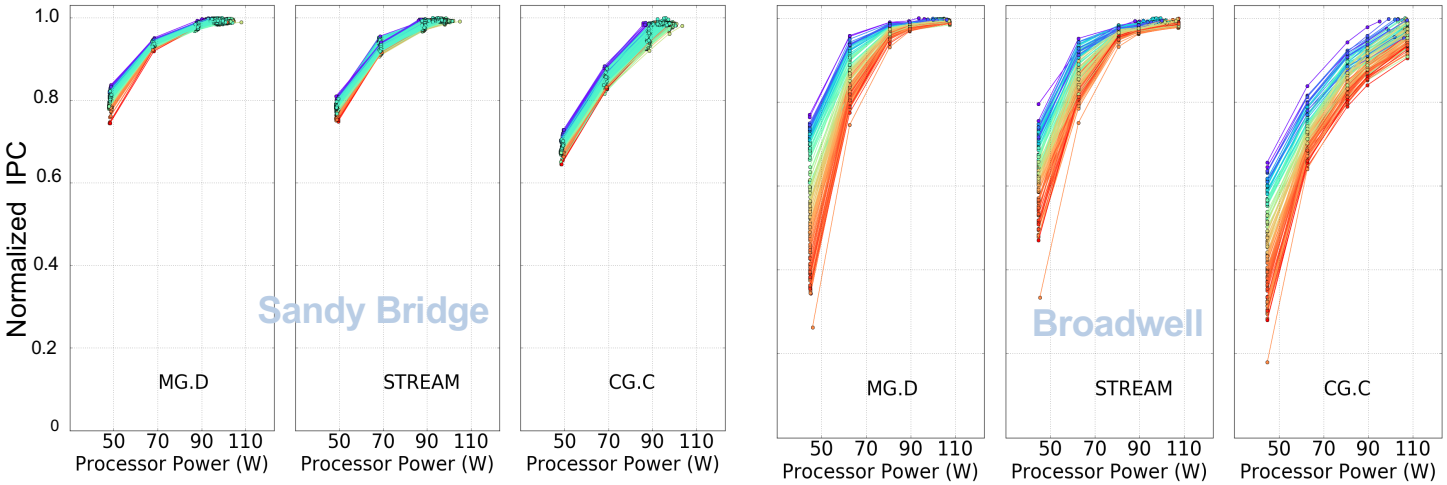
Graph Source: Daniel Ellsworth, University of Oregon

Automatic configuration selection is crucial for performance and energy efficiency





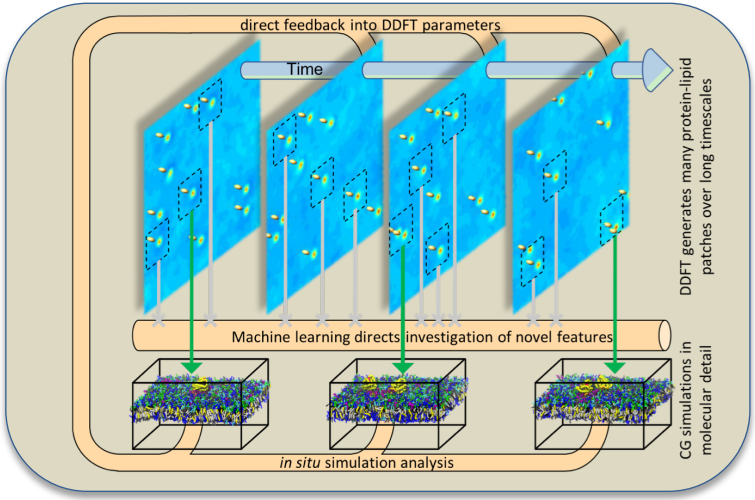
Performance variability can result from processor manufacturing process

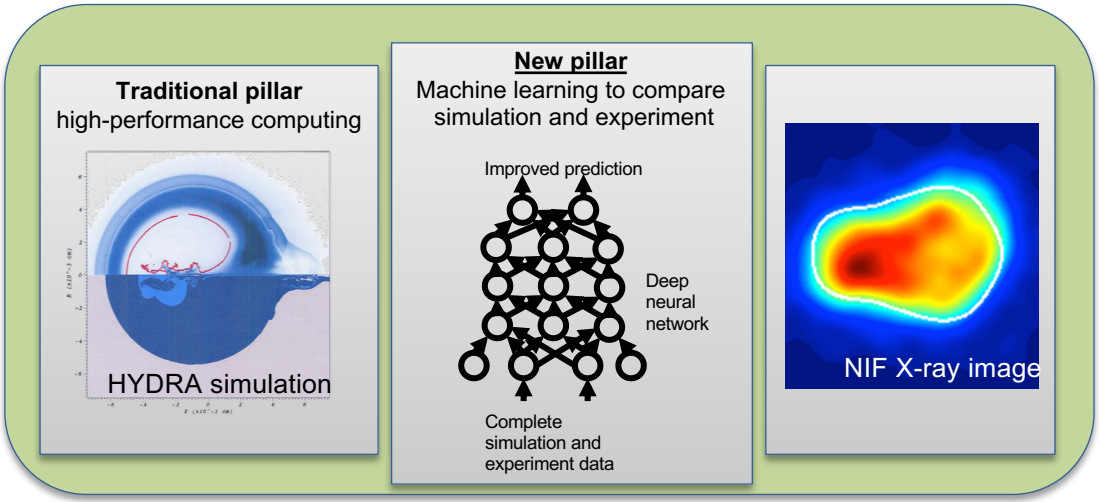


- Manufacturing variability in hardware continues to increase, and will worsen with heterogeneous systems
- 4x difference between two generations of Intel processors, needs advanced runtime options for mitigation

Demo of Application Performance and Processor Variability

Workflows on high-end HPC systems are undergoing significant changes.

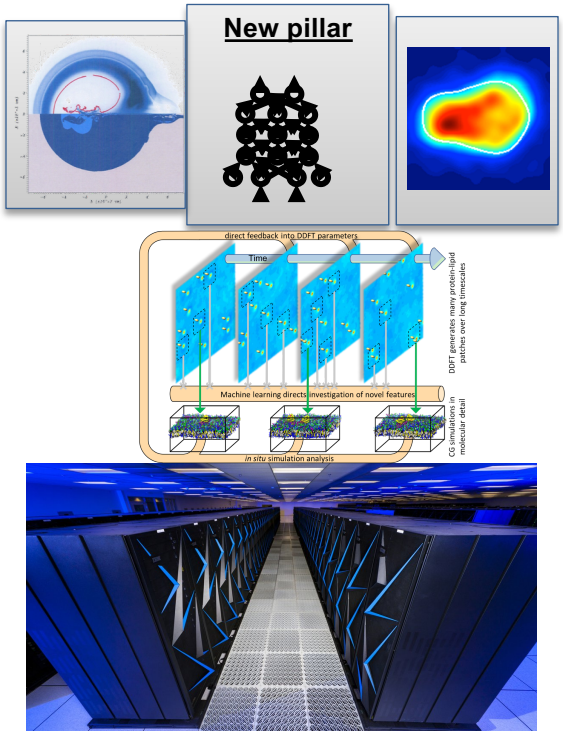




- Cancer Moonshot Pilot2 – co-schedule many elements and ML continuously schedules, de-schedules and executes MD jobs.
- In-situ analytics modules
- ~7,500 jobs simultaneously running

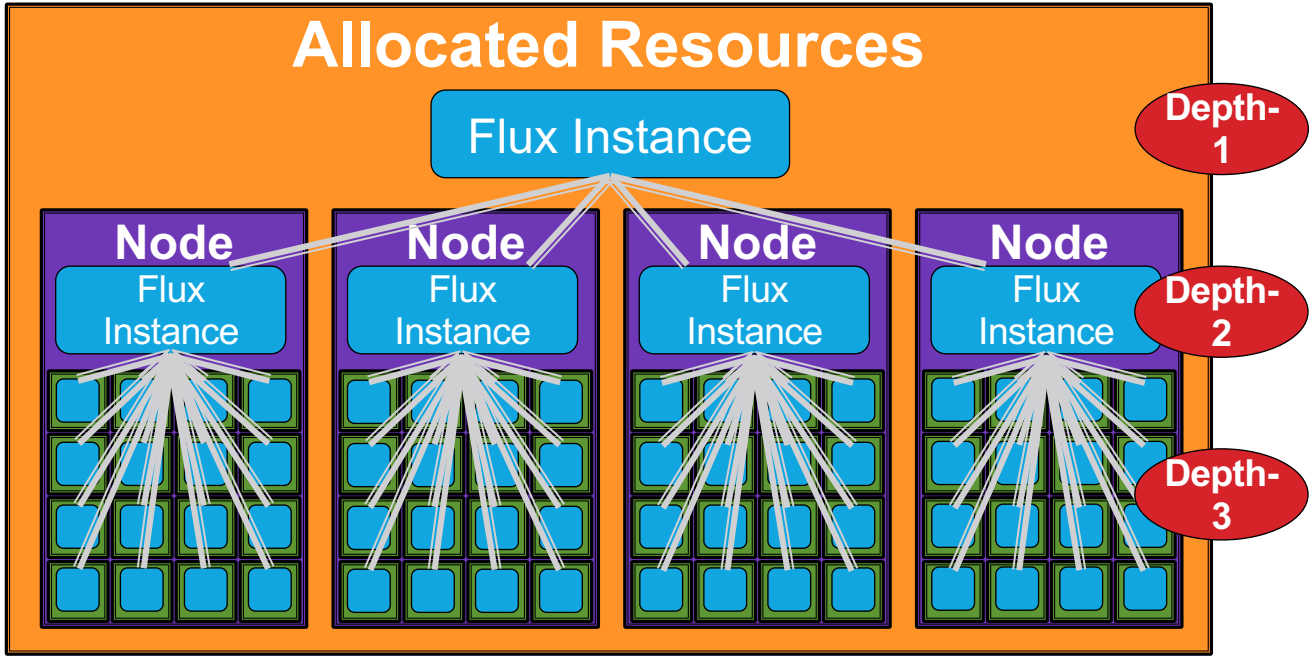
- Machine Learning Strategic Initiative (MLSI) – 1 billion short-running jobs!
- Similar needs for co-scheduling heterogenous components

Key challenges in emerging workflow scheduling include…



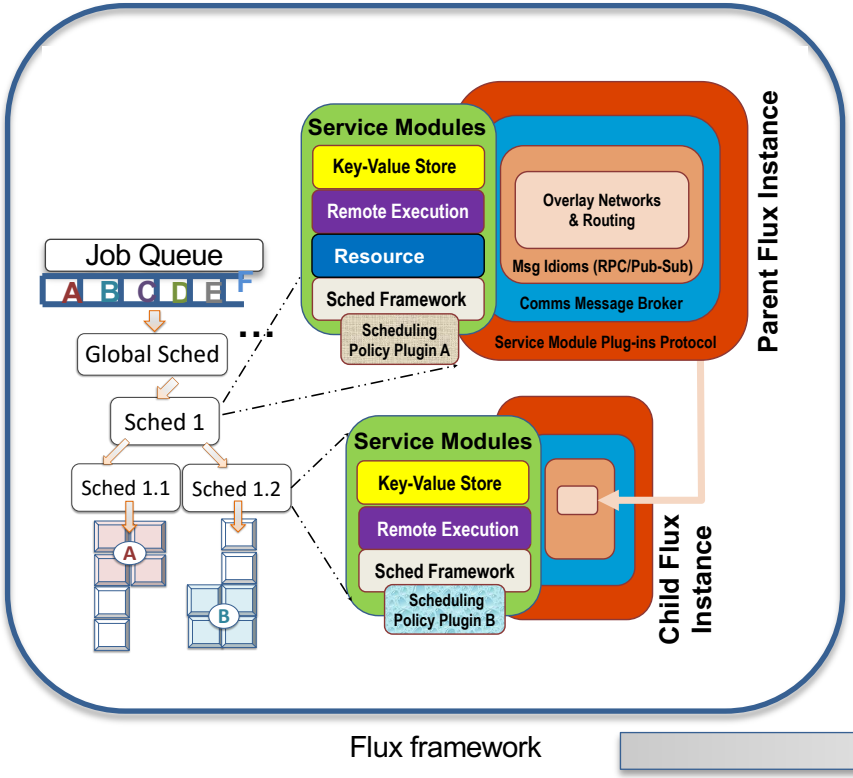
- Co-scheduling challenge
- Job throughput challenge
- Job communication/coordination challenge
- Portability challenge

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



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

Flux is specifically designed to embody our fully hierarchical scheduling model.



Techniques

Challenges

Scheduler Specialization

Scheduler Parallelism

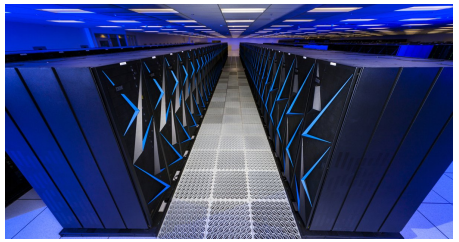
Rich API set

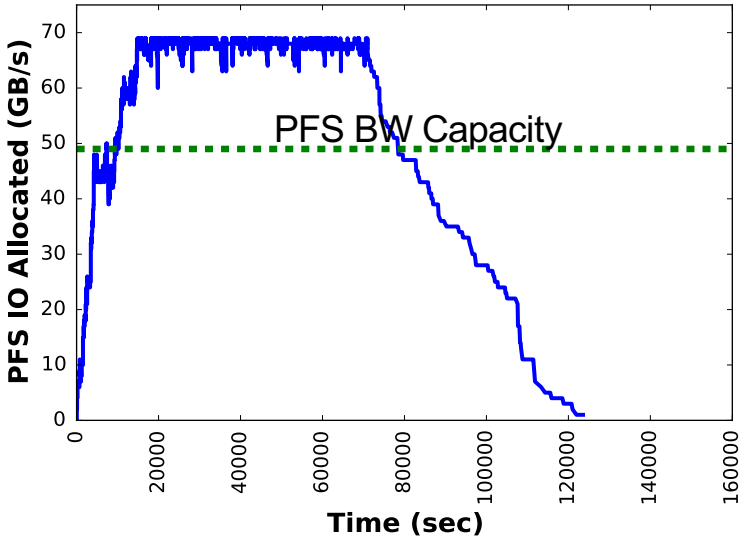
Consistent API set

- Co-scheduling challenge
- Job throughput challenge
- Job communication/coordination challenge
- Portability challenge

The changes in resource types are equally challenging.

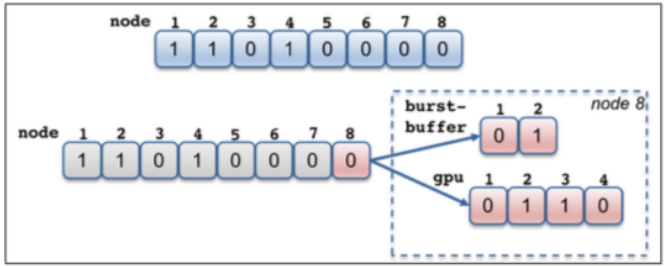
- Problems are not just confined to the workload/workflow challenge.
- Resource types and their relationships are also becoming increasingly complex.
- Much beyond compute nodes and cores…
  - GPGPUs
  - Burst buffers
  - I/O and network bandwidth
  - Network locality
  - Power





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

- Designed when the systems are much simpler
  - Node-centric models
  - Bitmaps to represent a set of compute nodes

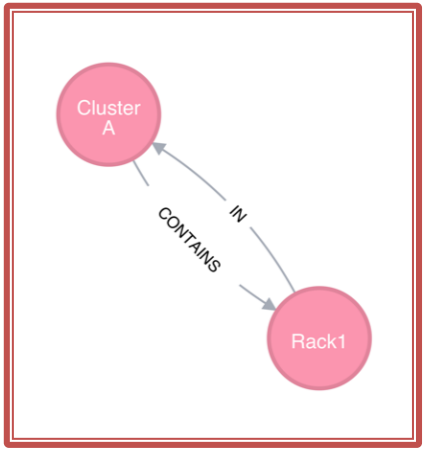


- HPC has become far more complex
  - Evolutionary approach to cope with the increased complexity
  - E.g., add auxiliary data structures on top of the node-centric data model

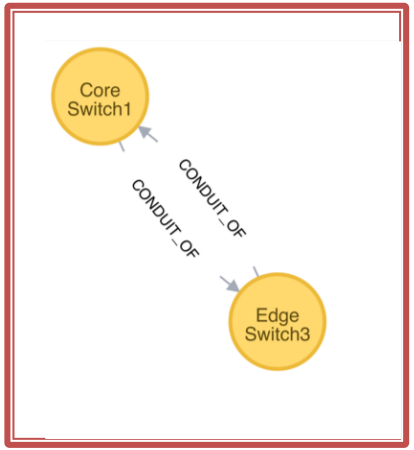
- Can be quickly unwieldy
  - Every new resource type requires new a user-defined type
  - A new relationship requires a complex set of pointers cross-referencing different types.

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



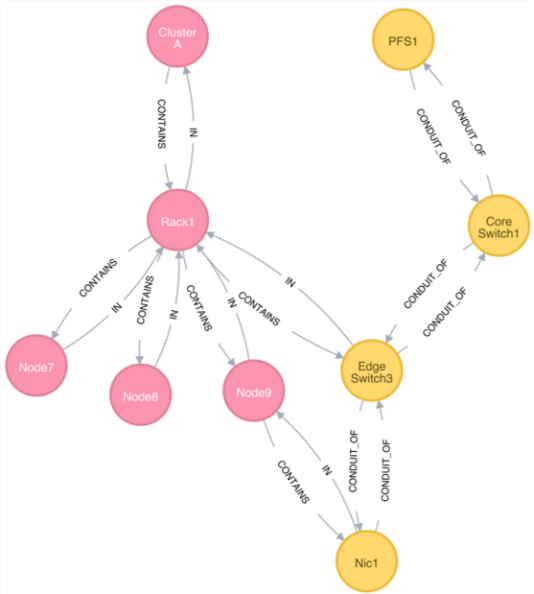
Containment subsystem



Network connectivity subsystem

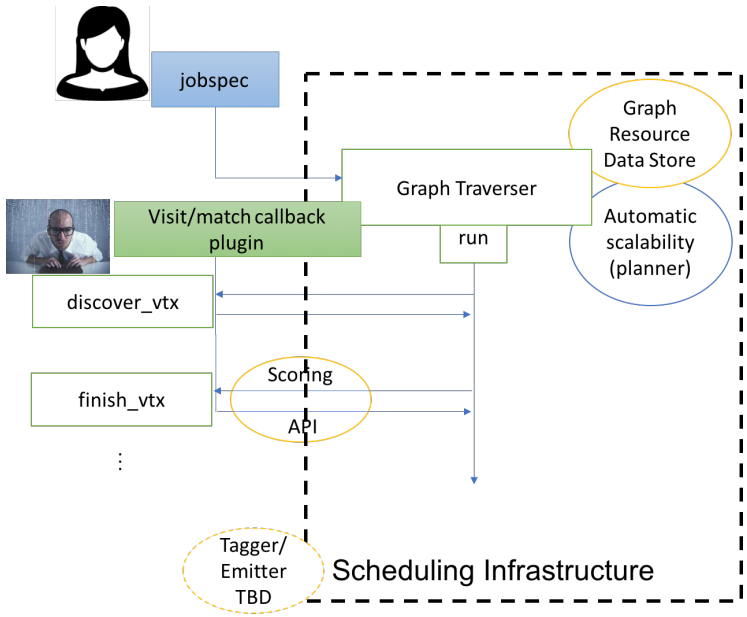
Our graph-based resource data model reduces the complexity of our scheduler.

- Highly composable to support a graph with arbitrary complexity
  - Simply add new vertices and connectivity to compose
- The scheduler remains to be a highly generic graph code.
  - No code change is required on new resource types and relationships.



Flux architecture embodies our graph matching scheme with good separation of concerns and extensibility.

- Allow our resource module to populate and maintain the total graph of the instance.
  - Graph resource data store
- On receiving each job-spec object:
  - Start to traverse the graph with a known visit type (currently DFV and DFU).
  - On each visit event (e.g., post-order visit)
    - a policy match callback is invoked to score the visiting vertex
    - high- or low-id first, locality-aware, and performance variation-aware policies
- Best matching resources are selected at the end of the traversal.

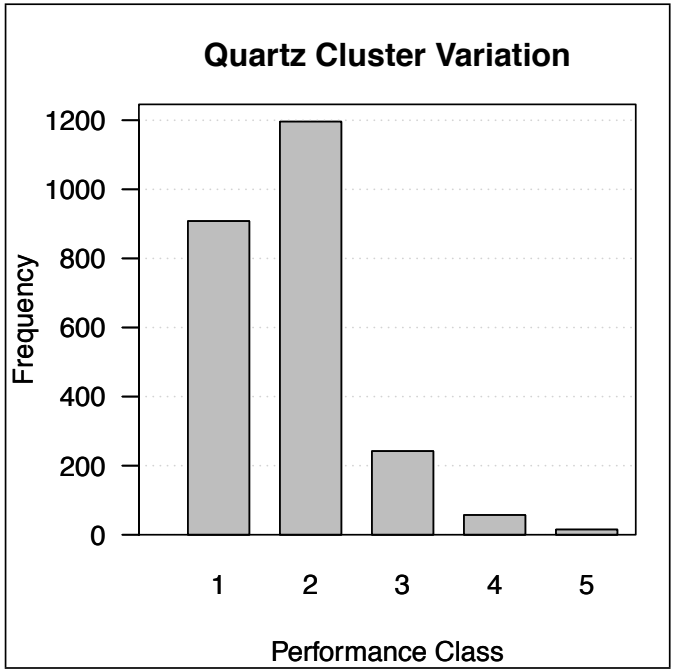


Statically determining node performance classes: 2469 nodes of Quartz

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

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

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

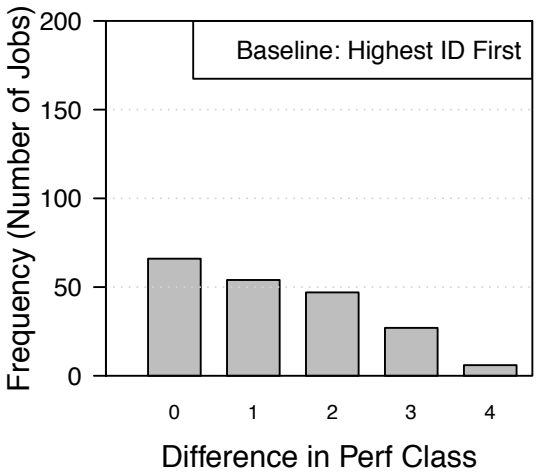


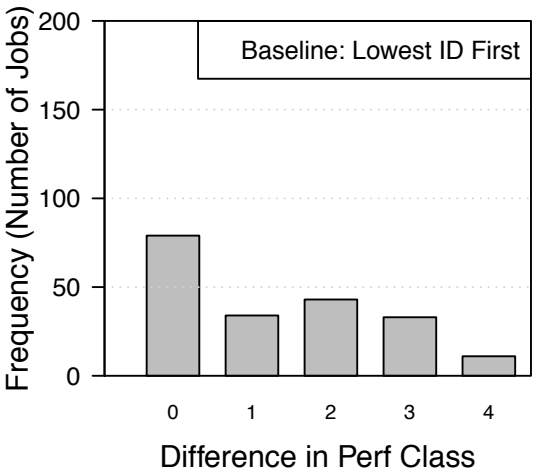
Measuring impact of variation-aware scheduling

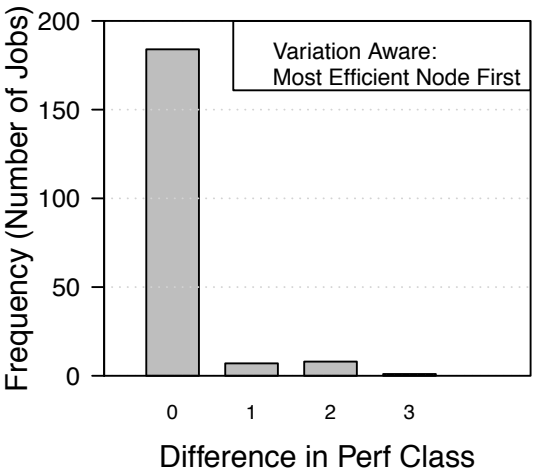
$$P_j := \{p_a | a \in n \wedge allocated(a, j)\}$$
$$fom_j = max(P_j) - min(P_j)$$

- *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







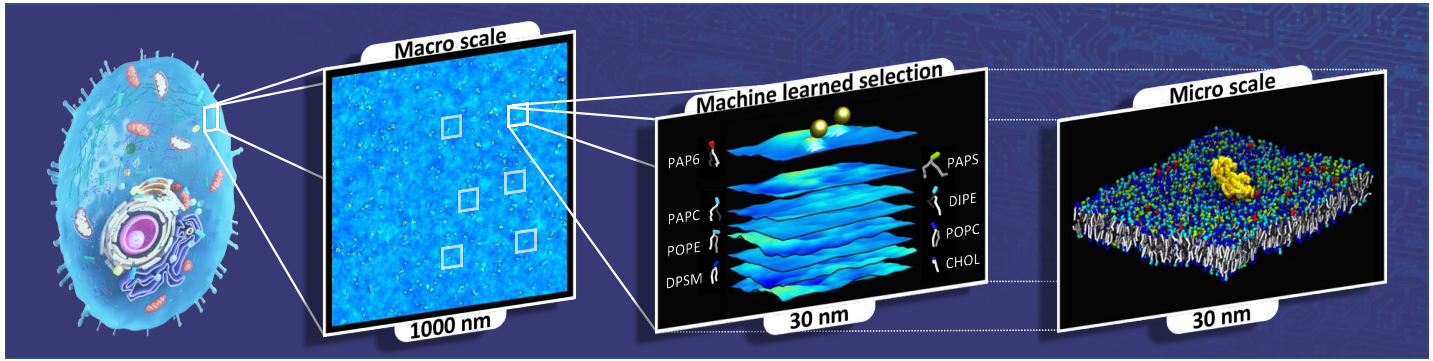
Performance analysis of modern workflows is critical to the procurement process of next-gen systems

- HPC systems are becoming extremely heterogenous (Sierra, Summit…)
  - Increased concurrency, multiple per-node GPUs, burst buffers, etc.
  - Maintaining high utilization of resources is challenging
  - Efficient design points are harder to obtain: e.g. power constraints, network/IO bandwidth…
  - How do we measure efficiency in heterogeneous environments when procuring future systems?

- Modern scientific workflows are complex
  - Multiple binaries for different devices and tasks
  - Both serial and parallel components: MPI, CUDA, OpenMP
  - Combinations of compilers and workflow management tools
  - Co-scheduled dependencies

Scalable performance tools
don’t exist for end-to-end analysis of workflows!

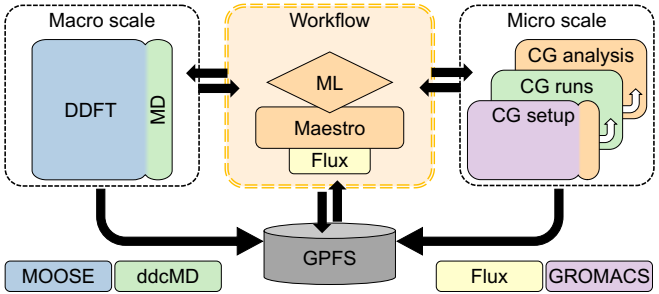
Multiscale Machine-Learned Modeling Infrastructure (MuMMI) is designed to understand a cancer-signaling mechanism



- Mutated RAS protein is implicated in **a third of all human cancers**
- Understanding the role of RAS requires exploration at multiple length- and time-scales

Multiscale Machine-Learned Modeling Infrastructure (MuMMI) is designed to leverage massive heterogenous supercomputers

Type of code	Programming language	# of concurrent executions	Resources per execution
Macro Scale (continuum)	C++ (with MPI)	1	24 CPU cores
Micro Scale (CG molecular dynamics)	C++ (with CUDA)	O(10,000)	1 CPU + 1 CPU core
CG setup	Python	O(10,000)	20 CPU cores
CG analysis (in situ)	Python	O(10,000)	3 CPU cores
Workflow and ML	Python	1	1 computational node



First step in power analysis of workflows: MuMMI workflow and GPU's

- Pilot2 Multiscale Machine-Learned Modeling Infrastructure (MuMMI)
  - Significant science advancements in cancer biology
  - Significant computing advancement with co-scheduled GPU components
  - Scales to massive supercomputers such as Sierra and Summit

- GPU Power and Performance analysis:
  - How power efficient are GPU workflow components?
  - How do GPU power caps and frequency caps impact performance?
  - What tools can be used at scale and across HPC systems/architectures?

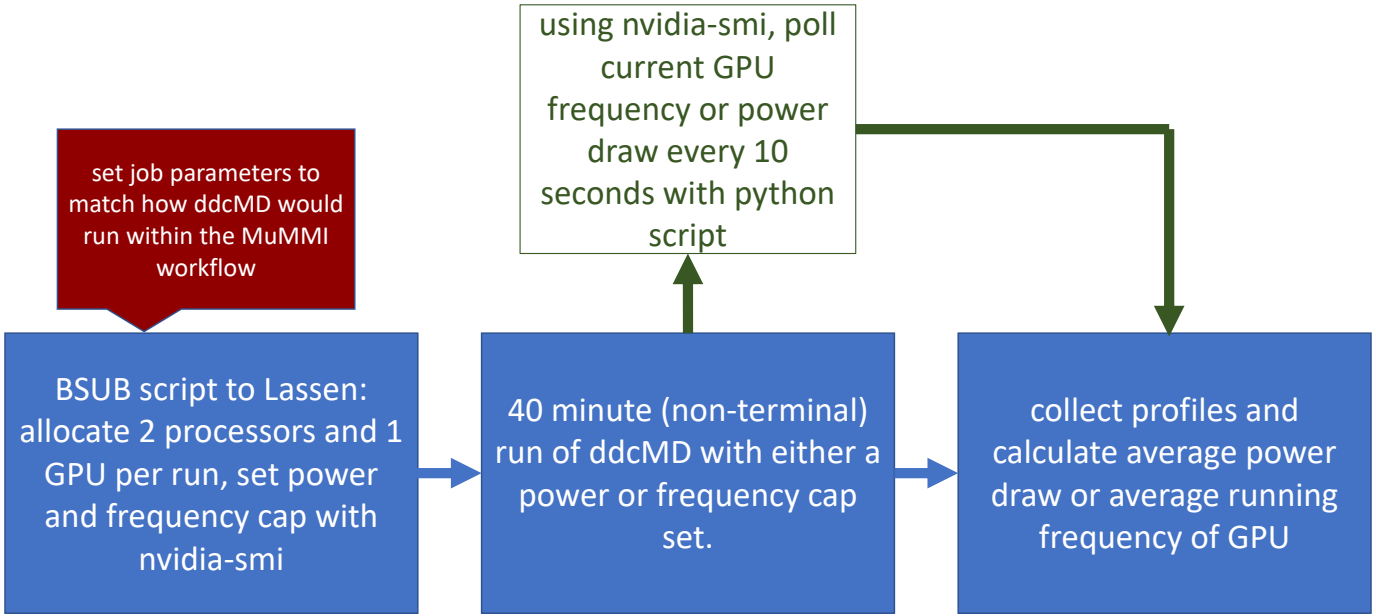
Our HPC environment for MuMMI analysis

- LLNL's Lassen Cluster (mini-Sierra)
- IBM Power 9 Architecture
- 795 nodes consisting of:
  - IBM Power9, 40 cores per node
  - 4 Nvidia Volta Tesla V100 (Volta) GPUs
  - 256 GB of DDR4 main memory
- LSF is the main job scheduler
  - Flux and Maestro are two other scheduling components used for workflows
- Listed at 10 on the top 500

Power and frequency capping tools

- We did not have permissions to limit entire node power on Power9
  - Future research will enforce limits on Power9 CPUs and GPUs
- Nvidia provides tool for GPUs: nvidia-smi
  - *Nvidia-smi –ac* : set frequency caps
    - An upper limit that the frequency cannot exceed
  - *Nvidia-smi –pl* : sets power caps
    - An upper limit that power draw from the GPU cannot exceed

Power management and variation in workflows: data collection workflow

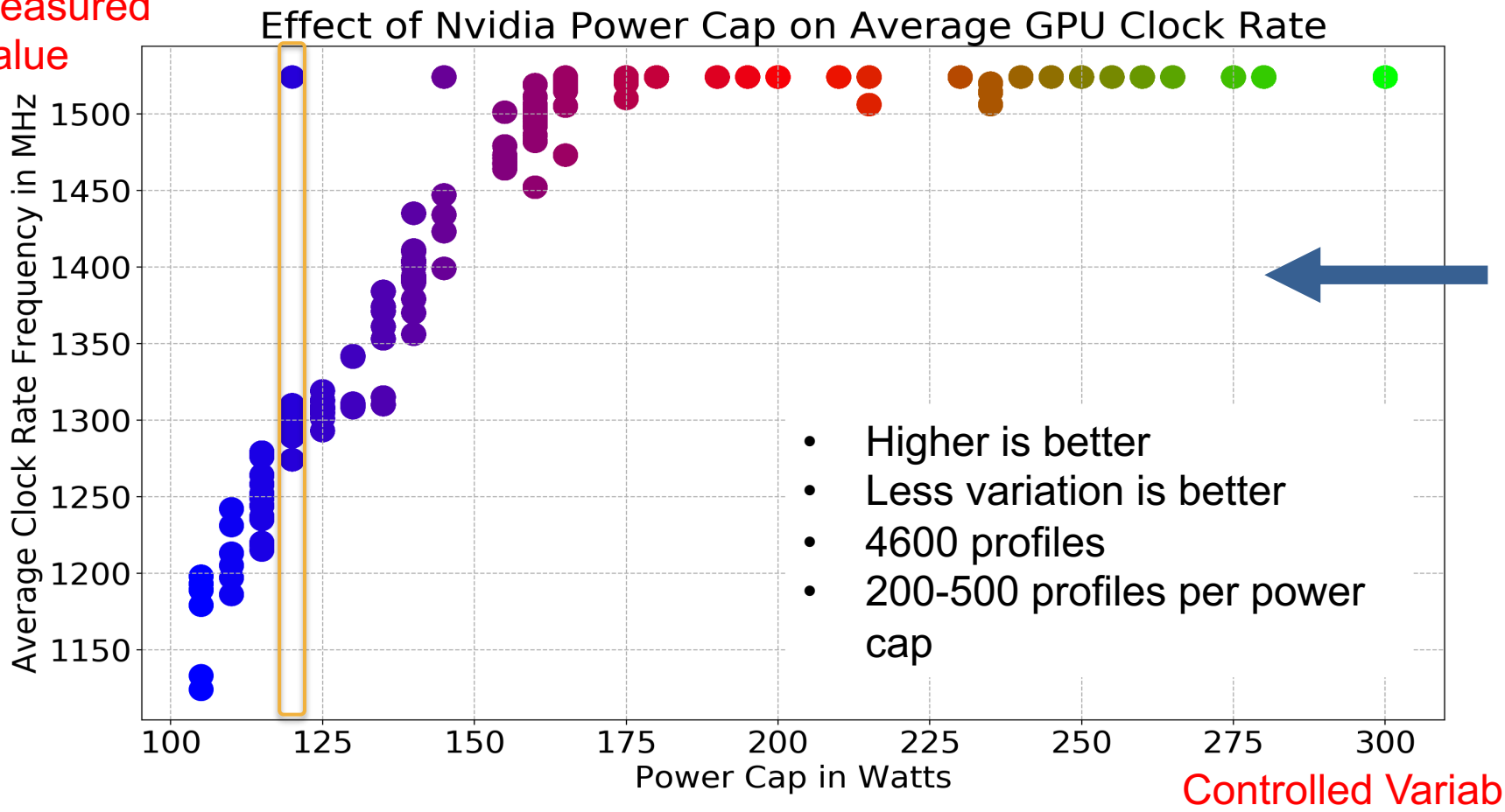


General data numbers we collected

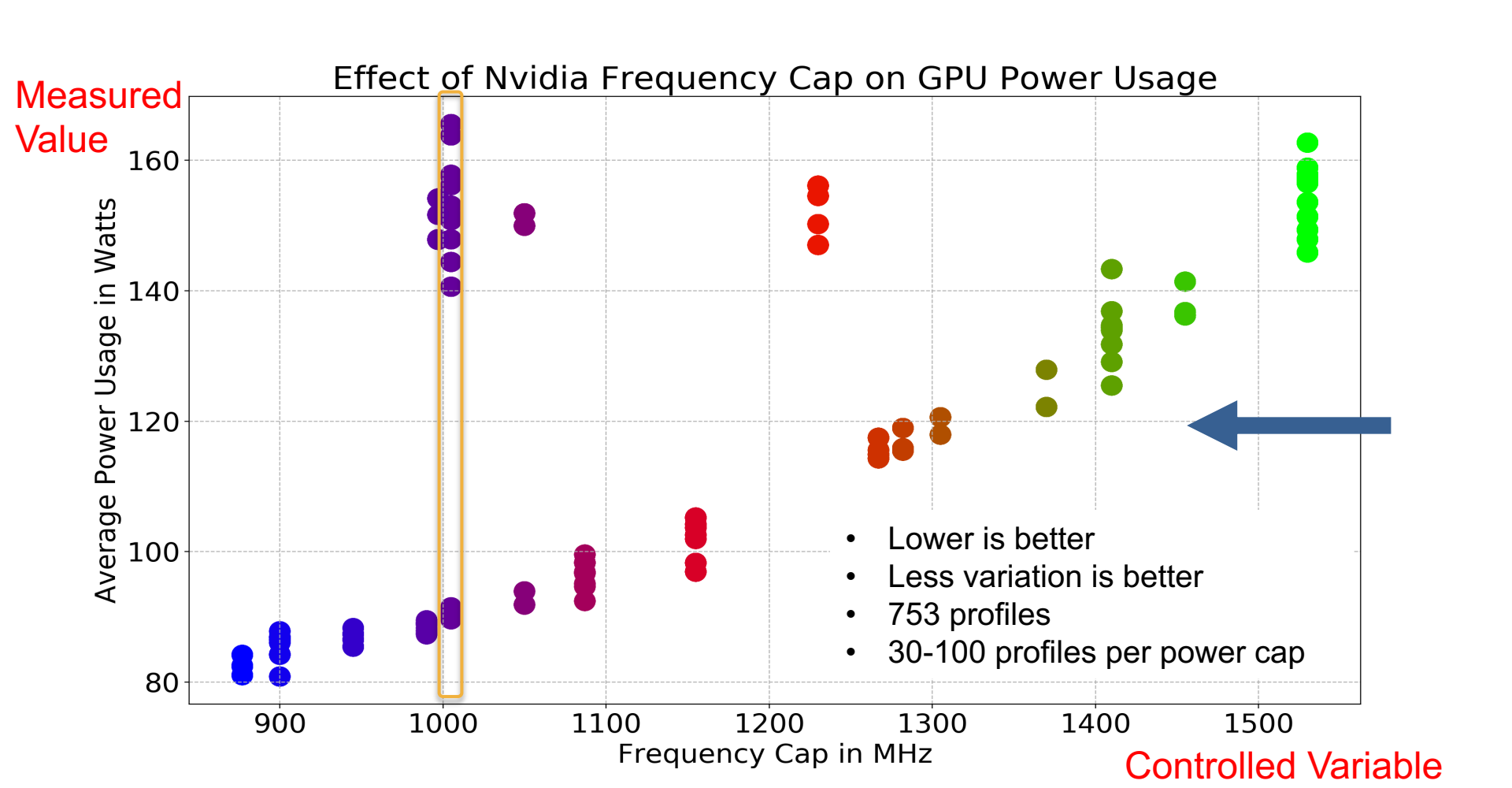
- 5353 ddcmd profiles were collected across two experiments
- Experimental control variables: power or frequency limitation

Power Limit Testing	Frequency Limit Testing
4600 profiles generated with power caps	753 profiles were generated with frequency caps
Across 30 different power caps	Across 17 different profiles
From 105W to 300W	From 877 MHz to 1530 MHz

Measured Value



Controlled Variable


Measured Value
Effect of Nvidia Frequency Cap on GPU Power Usage
Average Power Usage in Watts
160
140
120
100
80
900
1000
1100
1200
1300
1400
1500
Frequency Cap in MHz
• Lower is better
• Less variation is better
• 753 profiles
• 30-100 profiles per power cap
Controlled Variable

Conclusions for the MuMMI workflow can apply across the board: more runtime analysis is necessary

- 120W savings in power without slowdown in performance per GPU
- Cluster wide savings of 382kW
  - 254.6 kWh energy savings
  - $43k in savings per day @ 7c per kWh
- Limiting power from 300W to 105W
  - Measured slowdown of 17% In ddcMD performance
- GPU frequency does not equate to general performance of workflows
- Frequency capping proved to be an unreliable way of improving efficiency

So, how do we even manage power? (measurement vs control)

Two primary mechanisms in hardware for control:

- Dynamic voltage and frequency scaling
- Power capping, such as RAPL (Intel) or OPAL (IBM)

- Automatic tuning through Intel Turbo Boost or IBM UltraTurbo

- ACPI defines P-states, which are voltage/frequency pairs
- DVFS: `cpufreq`, '`userspace`' governor
- RAPL:
  - Low-level register programming, supported with `msr` kernel module along with `msr-tools`
  - LLNL's `msr-safe` and `variorum` provide safe and clean access from user space across architectures
- OPAL:
  - Firmware layer providing in band monitoring with sensors and out of band power capping

Introduction to Intel's RAPL

- Intel provides programmable, machine-specific registers for power, energy and thermal management (power capping)

- MSR Domains for server architectures:
  - Package – represents processor cores, caches, and other things on the socket
  - DRAM – represents memory components
  - Other uncore registers also exist

Intel SDM: Vol 3, Chapter 14.9, https://software.intel.com/en-us/articles/intel-sdm

Deep dive into the MSR_PKG_POWER_LIMIT, (0x610h,rw)

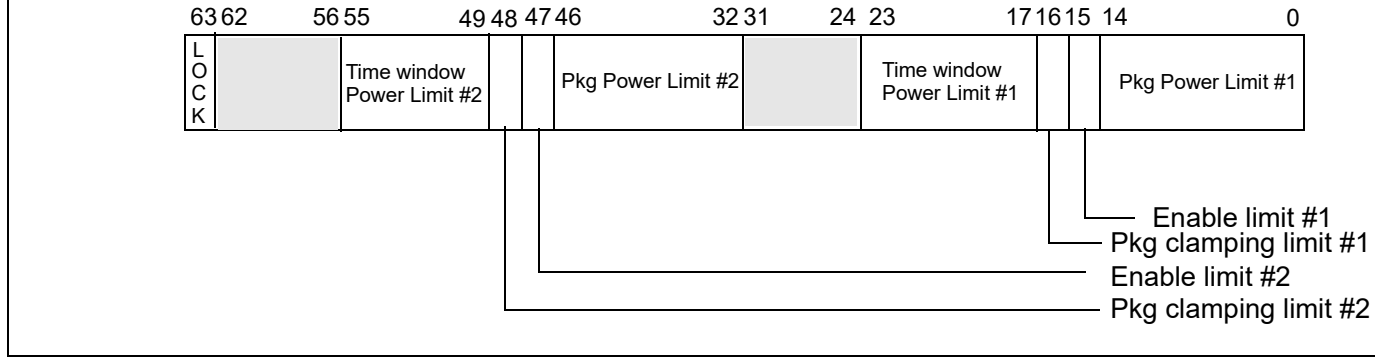


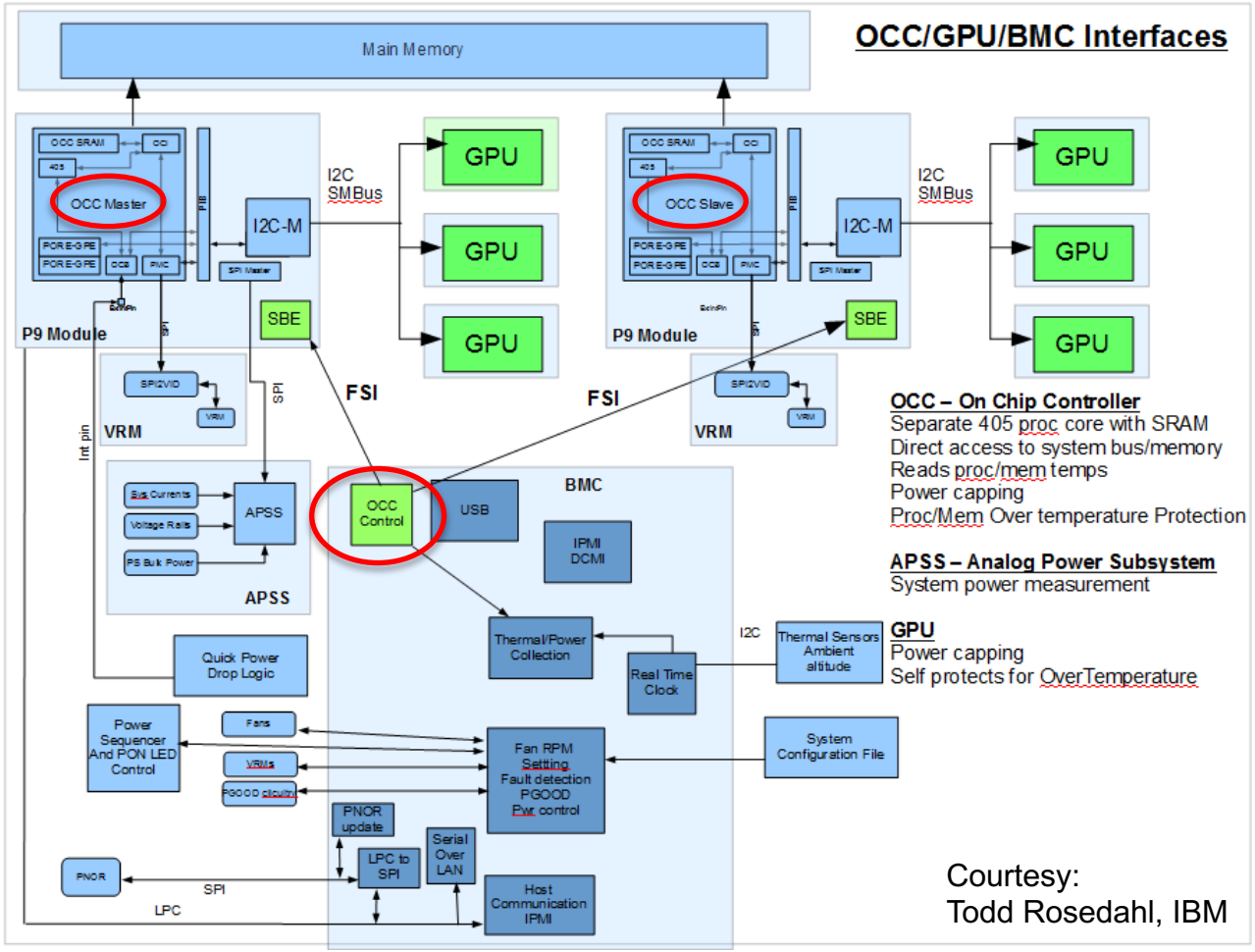
Figure 14-32. MSR_PKG_POWER_LIMIT Register

- Pkg Power Limit#1, bits 14:0, sets average power limit corresponding for window 1, unit 0.125 W
- Enable Limit#1: bit 15, 0 = disabled
- Pkg Clamping Limit#1, bit 16, allow going below OS requested P/T state
- Time Window Limit#1, bits 23:17, minimum is typically 1 millisec
- Pkg Limit #2 is typically not used

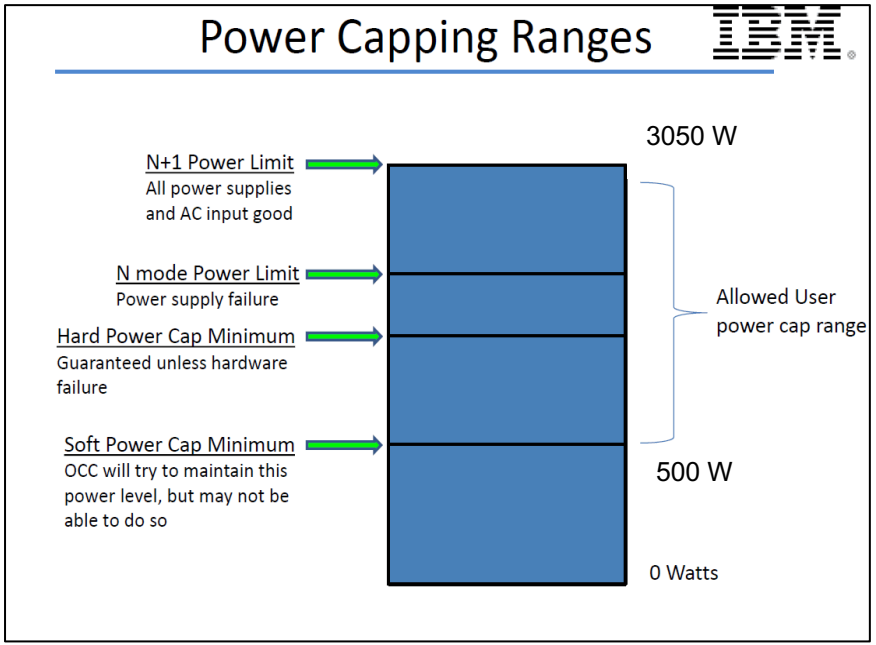
Demo of `msr-tools` and introduction to `msr-safe`

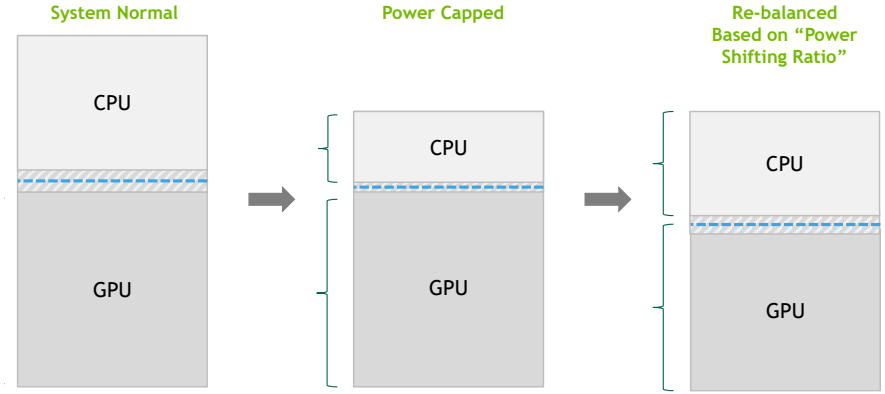
Power9 OCCs have a Primary-Secondary Design

- OCC collects thermal data every 250us
- Power knobs are exposed with the Open Power Abstraction Layer (OPAL)
- When power button is pressed, BMC selects master chip and releases SBEs, OPAL firmware is loaded



Capping Node and GPU power





- Power-shifting ratio determines distribution between CPU and GPU
- Per-socket cap and memory cap cannot be specified, master OCC has control

OPAL interfaces for in-band sensors and power capping

Requires OPAL firmware v6.0.18 – includes fix for a firmware bug we found based on soft power cap

Read-only access:

- `/sys/firmware/opal/exports/occ_inband_sensors`
  - Over 336 sensors reported on `alehouse`
  - Including power, temperature, frequencies for CPU, Memory, GPU (1ms granularity)

Read/write acces:

- `/sys/firmware/opal/powercap/system-powercap/powercap-current`
- `/sys/firmware/opal/psr/cpu_to_gpu_*` (per socket)


Lawrence Livermore
National Laboratory

Agenda

Topic	Time Slot	Presenters
PowerStack Introduction and ECP Challenges	*2:30 – 2:50*	*Tapasya Patki*
Workflows and site-level power management	*2:50 – 3:10*	*Tapasya Patki*
Power Control Knobs on Intel and IBM systems	*3:10 – 3:30*	*Tapasya Patki*
BREAK	***3:30 – 4:00***	
Variorum	4:00 – 4:30	Stephanie Brink
Hands-on Tutorial on GEOPM	4:30 – 4:45	Aniruddha Marathe
GEOPM Agent/Platform API	4:45 – 5:15	Aniruddha Marathe
LLNL's advanced plugins (DVFS-based IBM plugin, configuration selection plugin)	5:15 – 5:50	Aniruddha Marathe
Wrap up	5:50 – 6:00	All

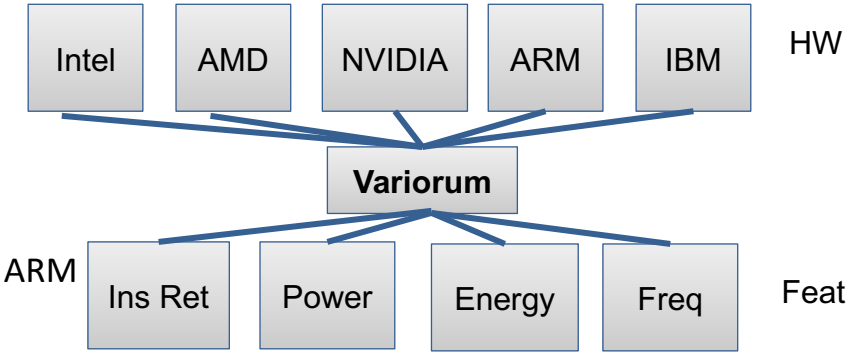
Power management capabilities differ across vendors

- `libmsr` (~2012) has been quite successful in the community
  - Provided simpler monitor/control interfaces for translating bit fields into 64-bit MSRs
  - But, was Intel specific

- Interfaces, domains, latency, capabilities
  - But, may also vary across generations within the same vendor

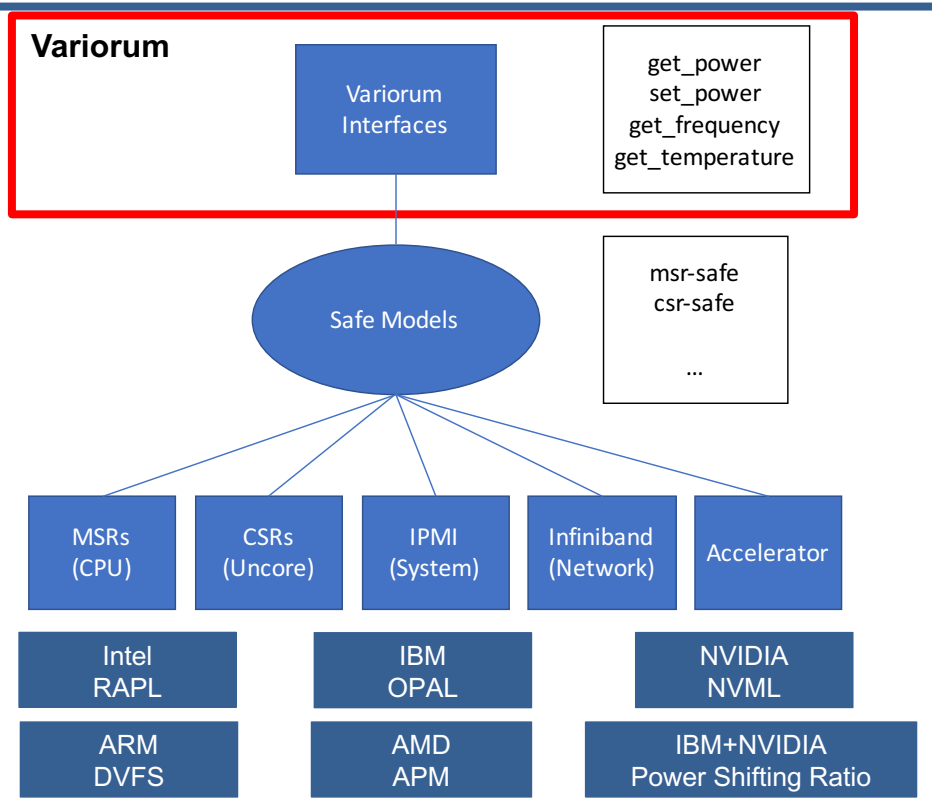
- Our goals:
  - Target 95% of users (friendly APIs)
  - More devices
    - *i.e.*, CPUs, Accelerators, IPMI, PCIe (CSRs, MMIO)
  - More platforms
    - *i.e.*, Intel Skylake, IBM P9, NVIDIA Volta, AMD Epyc, ARM
  - More hardware knobs and controls

Feature	Ivy Bridge	Haswell
CPU Freq	Per-socket implementation	Per-core implementation
Energy Status	Uses energy unit fused into MSR_POWER_UNIT for conversion	Uses standard 15.3 micro-Joules for conversion
Power Limit	Same implementation	



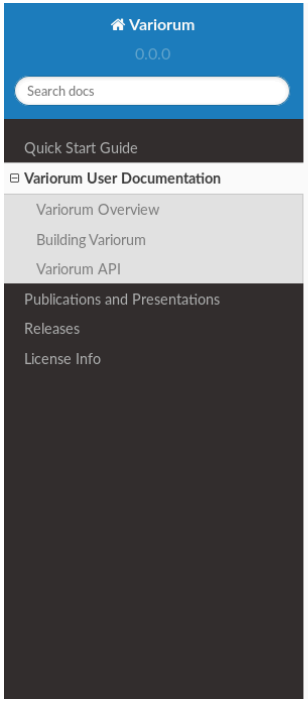
Variorum:
Vendor-neutral user space library for hardware control knobs

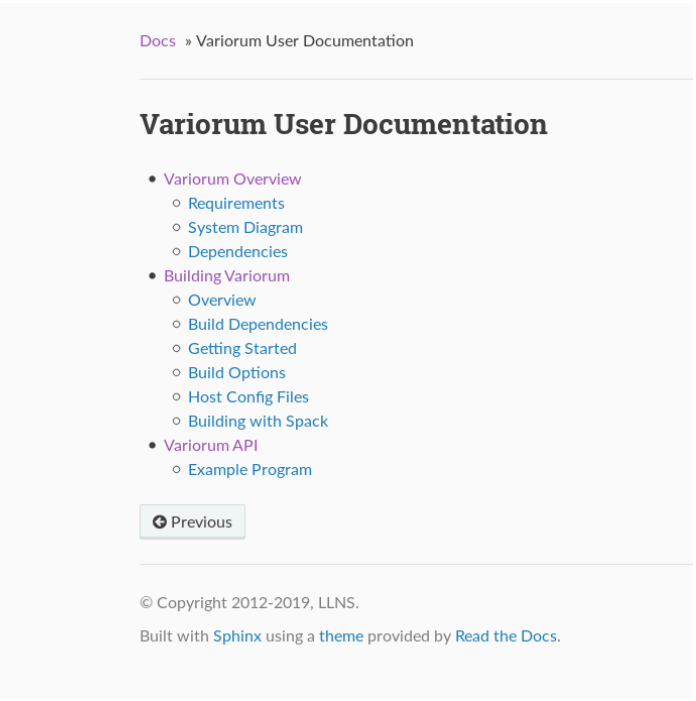
- Platform-agnostic simple front-facing APIs
- Security layer provided to ensure safe, reliable operation
- Batching interfaces minimizing overheads of reading/writing many MSRs
- Production version of `libmsr`, which targeted Intel architectures
- C-based library
- Function pointers to specific implementation for target architecture



Variorum v0.1.0 Release

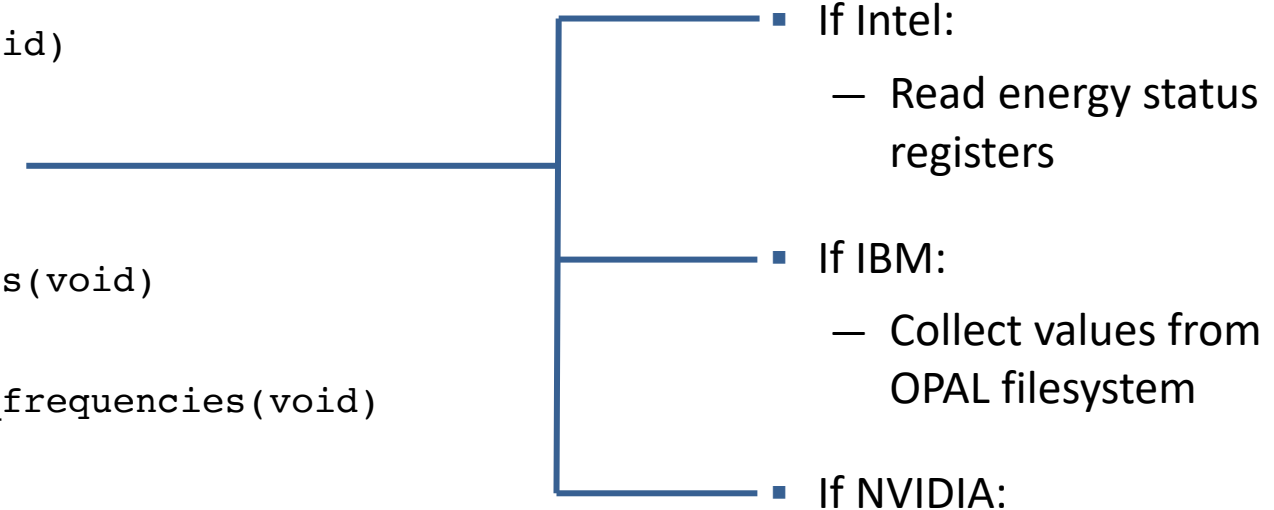
- Released Nov 11, 2019!
  - https://github.com/llnl/variorum
- Resources
  - Spack package for installation
    - under review
  - readthedocs webpages
    - https://variorum.readthedocs.io/
- License: Permissive (BSD-3)
- Current support for Intel Skylake (and older) and IBM Power9
- On the immediate roadmap: NVIDIA+IBM, ARM, AMD, advanced user APIs





Variorum APIs / Examples (Subset)

- `int dump_thermals(void)`
- `int dump_power(void)`
- `int dump_power_limits(void)`
- `int print_available_frequencies(void)`
- `int set_power_limits(val)`
- `int disable_turbo(void)`



Demo of `variorum`


Lawrence Livermore
National Laboratory