An Exascale Operating System and Runtime Software Research & Development Project

Developing vendor neutral, open-source OS/R software

ANL: **Pete Beckman** (*PI*), Marc Snir (*Chief Scientist*), Pavan Balaji, Rinku Gupta, Kamil Iskra, Franck Cappello, Rajeev Thakur, Kazutomo Yoshii
LLNL: Maya Gokhale, Edgar Leon, Barry Rountree, Martin Schulz, Brian Van Essen
PNNL: Sriram Krishnamoorthy, Roberto Gioiosa
UC: Henry Hoffmann
UIUC: Laxmikant Kale, Eric Bohm, Ramprasad Venkataraman
UO: Allen Malony, Sameer Shende, Kevin Huck
UTK: Jack Dongarra, George Bosilca, Thomas Herault

See http://www.argo-osr.org/ for more information

Terse Intro

- ASCR FastOS was a forum back in 2002, and funded research in 2004
- OS/R work usually involves shared resources
- Separation of *Mechanism* and *Policy*
- OS/R research is ***not*** fixated on node kernels
  - (*even if that's all you ever hear about*)
  - Many (most?) OS/R components are user-space
  - When Linux kernel extensions are needed, patches are common and easy, adoption straightforward
    - PAPI, Lustre, libmsr-safe, BLCR, on-demand paging (for 0-copy HPC NIC), cross memory attach (some extensions migrate to the stock Linux kernel with help from vendors)

What is Argo building?
New System Software Components:

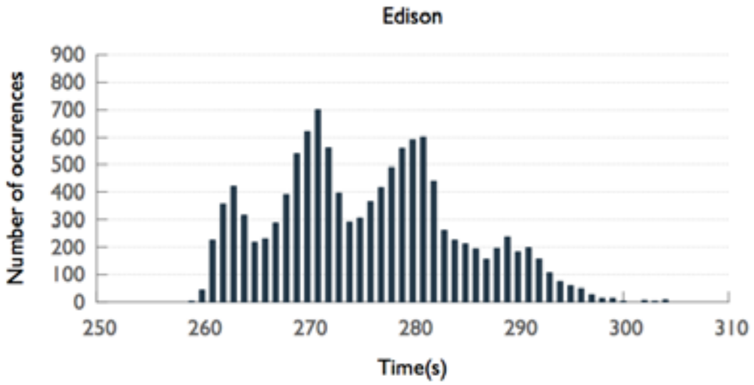
- Improve application performance
  - Argobots lightweight thread/task layer
    - Improve performance of MPI+OpenMP, math libraries: PLASMA, etc.
    - Support new, more dynamic / load-balanced programming models
  - Argo Backplane hierarchical pub/sub backplane
    - Provide APIs to build application resilience with out-of-band events
- Support new application modes
  - Argo Containers manage cores, memory, and power within a node
    - Improve resource mgmt in support of in-situ analysis & burst buffers, etc.
  - Argo Backplane: in-situ data reduction, analysis, and introspection
- Provide new capabilities to applications
  - DIMMAP: provides new programmer interfaces for NVRAM
  - Argo Power: provides APIs; enables machine-learning & adaptation
  - Argo Global OS/R: support for new workflows, coupled apps, etc.

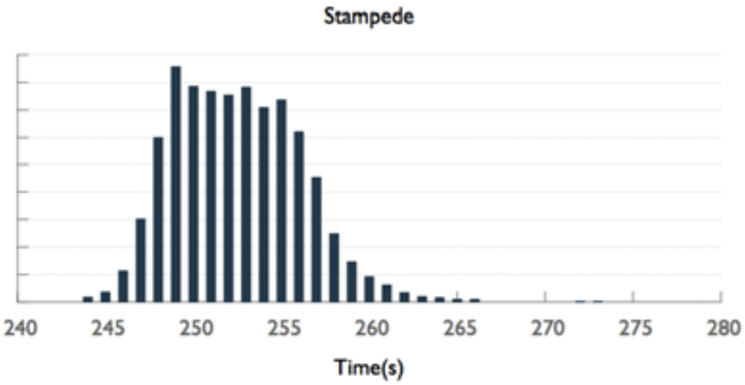
Is Current System Software Sufficient? What Gaps *Must* We Address?

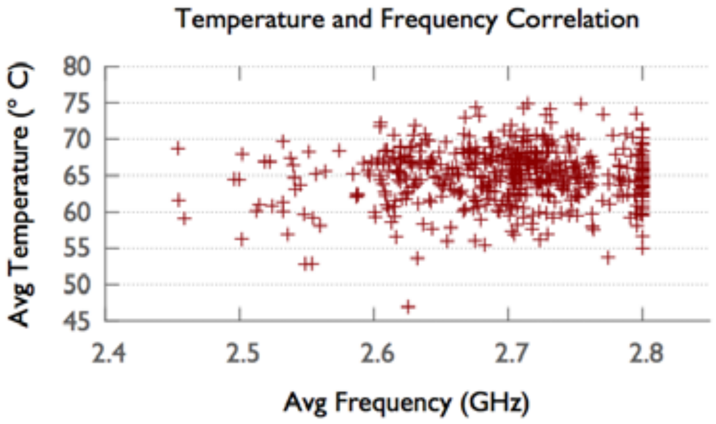
- **Extreme in-node parallelism**
  - Poor mechanisms for precise resource management (cores, power, memory, network)
  - Legacy threads/tasks implementations perform poorly at scale
- **Dynamic variability of platform; Power is constrained**
  - Poor runtime mechanisms for managing dynamic overclocking, provisioning power, adjusting workloads
  - No mechanisms for managing power dynamically, globally, and in cooperation with user-level runtime layers
- **Hierarchical memory**
  - Poor interfaces / strategies for managing deepening memory
- **New modes for HPC**
  - No portable interfaces for easily building workflows, in-situ analysis, coupled physics, advanced I/O, application resilience

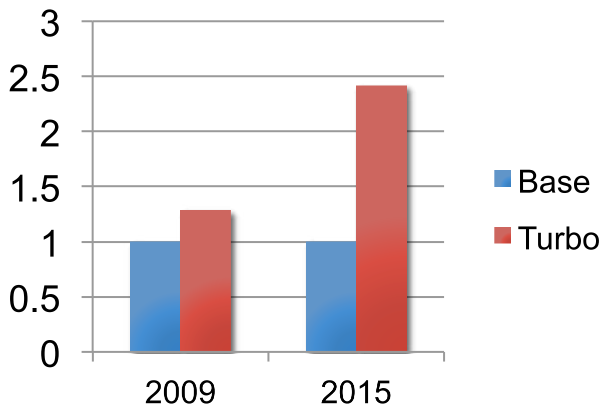
Dynamic Node Performance is Increasing

- To improve performance under power constraints, chips use dynamic overclocking
- Chips have ***increasing*** silicon process variability impacting power constraints







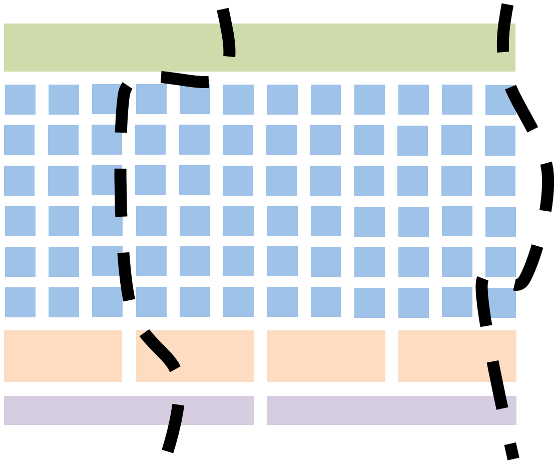


Increasing Dynamism

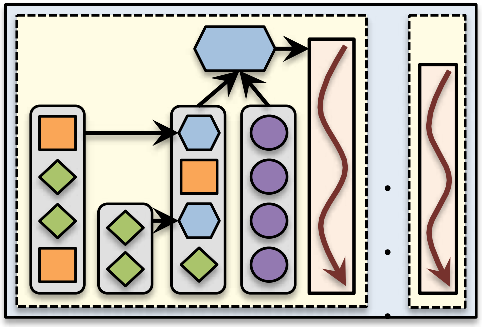
"...Execution time difference of up to 16% among processors within a 512 node allocation on Edison and Stampede"

Argo Innovations to Address Exascale Gaps

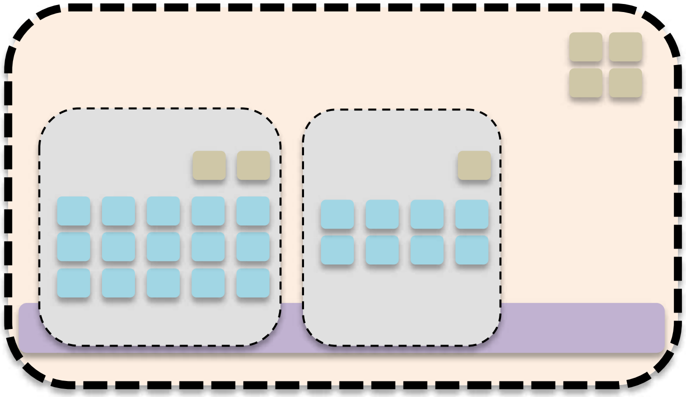
(starting with the key abstractions)



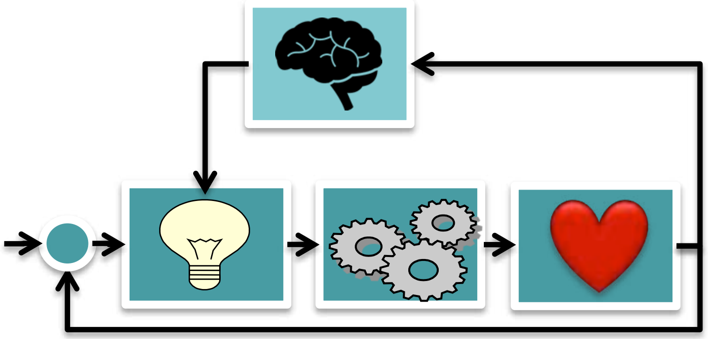
Elastic intranode containers with resource knobs



Lightweight thread/tasks designed for containers, messaging, and memory hierarchy

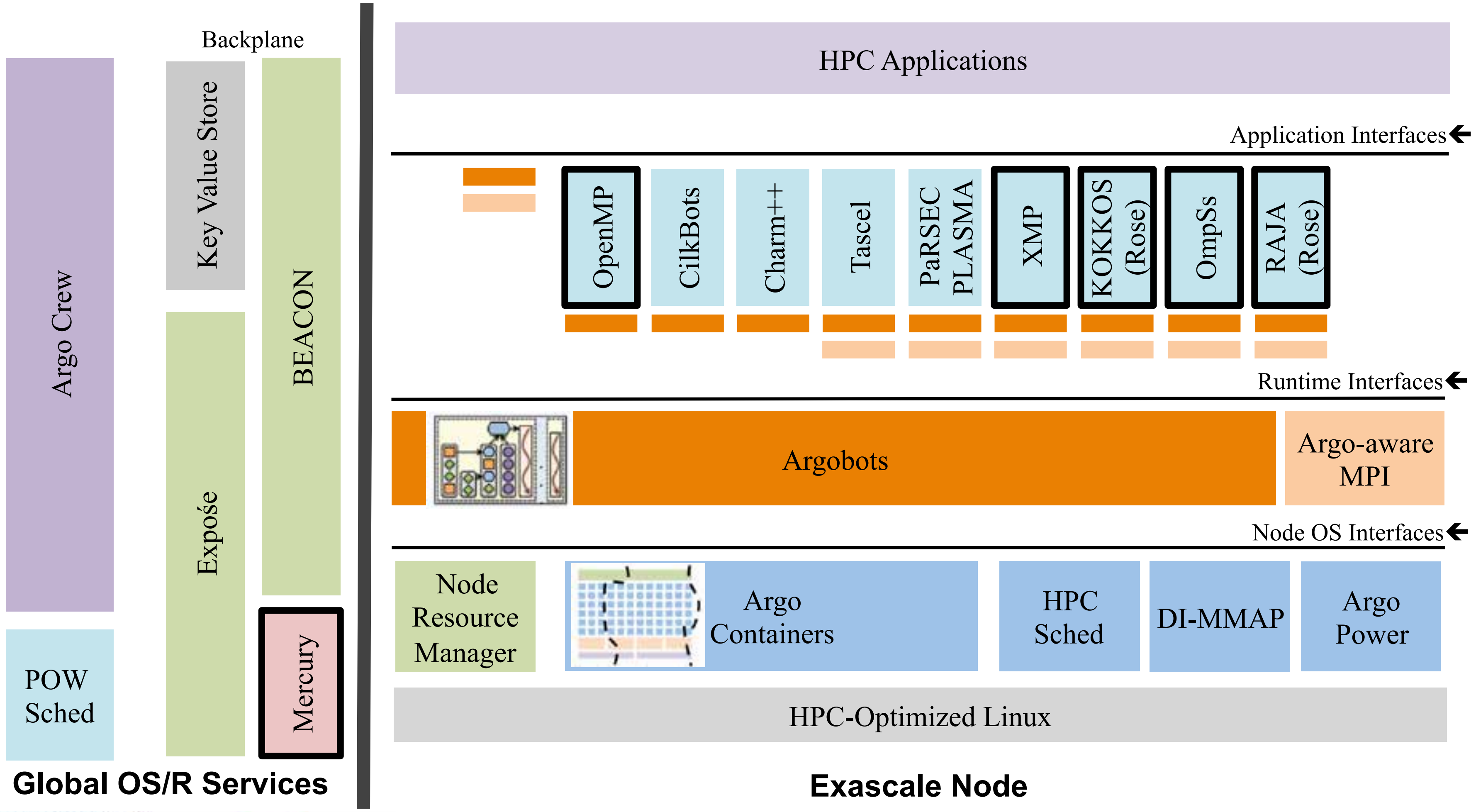


Hierarchy of *Enclaves* connected via a *Backplane*



Adaptive, learning, integrated control system

The Argo Architecture



Backplane

Argo Crew

Key Value Store

BEACON

Exposé

Mercury

POW Sched

Global OS/R Services

HPC Applications

Application Interfaces

OpenMP

CilkBots

Charm++

Tascel

PaRSEC PLASMA

XMP

KOKKOS (Rose)

OmpSs

RAJA (Rose)

Runtime Interfaces

Argobots

Argo-aware MPI

Node OS Interfaces

Node Resource Manager

Argo Containers

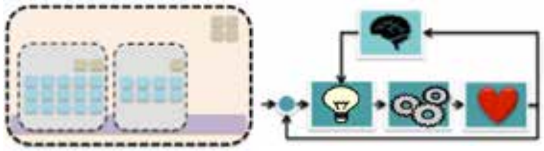
HPC Sched

DI-MMAP

Argo Power

HPC-Optimized Linux

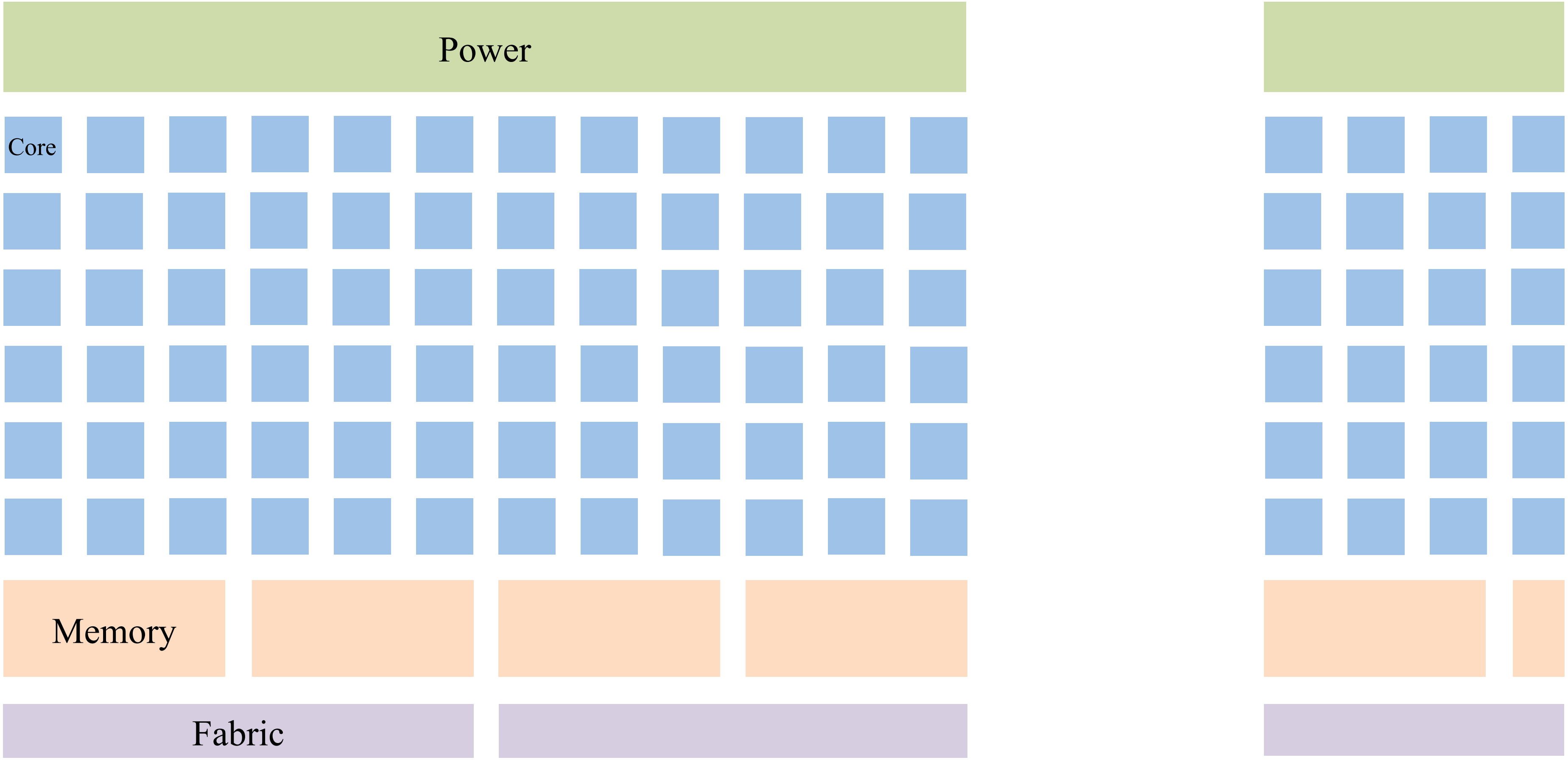
Exascale Node



□ = External Collaborations

Argo NodeOS/R: What are we building?

New set of **Linux** tools & extensions focused on *integrated resource management* for Exascale



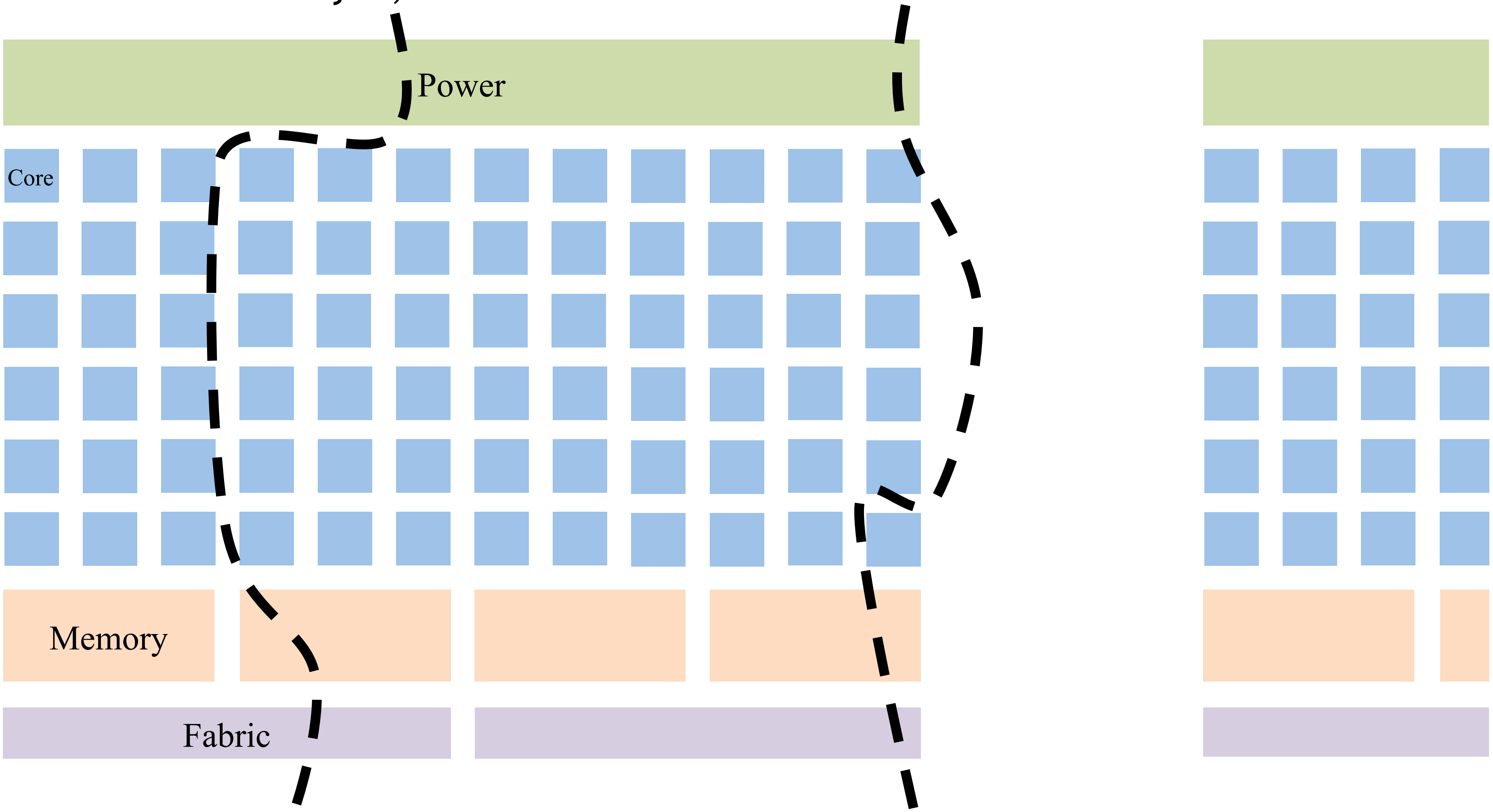
Why Containers?

HPC nodes are complex and powerful

Programmers need predictability (or not…)

Burst Buffer, in-situ Analysis, Coupled Codes, Advanced I/O, Remote methods/async, ServiceOS

Share or Partition?



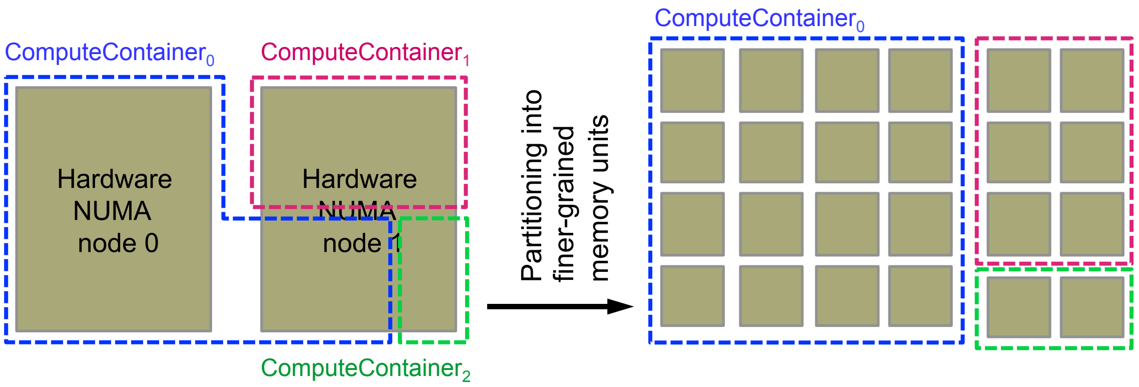
Argo Containers

- Leverages Google work on `cgroups` from 2007
- Adapts and extends concept for HPC and Exascale
  - Provides performance isolation and control knobs
- Allow HPC SW to manage resources more directly
  - Cores, Memory, Priority, I/O, Interconnect, Power
- Enable co-scheduling of different physical resources to provide optimal resource configuration and utilization
- Logical partitioning provides defined resources to:
  - Service OS, Simulation
  - Burst Buffer / Background Checkpoint Restart drain
  - In-situ analysis and data reduction
  - Advanced I/O, compression, etc.
- **KEY FEATURE:** Dynamic adjustment is easy
  - Enables machine learning (autotuning) to find optimal resource balance based on goals

```
$ argo_nodeos_config --create_service_os=\
"cpus:[0,24] mems:[0,24]"
$ argo_nodeos_config --create_container=\
"name:compute0 cpus:[2-10] mems:[1-23]"
$ argo_nodeos_config --alter_container=\
"name:compute0 -cpus:[8-10]”
[…]
$ argo_nodeos_config --show_config
======SERVICE_OS======
-Hardware threads: 0 24
-Memory nodes: 0 24
-  503 tasks
======================
----- COMPUTE CONTAINER -----
-Name: compute0
-Owner: judi (1001)
-Hardware threads(exclusive): 2 3 4 5 6 7
-Memory nodes(non exclusive): 1 2 3 4 5 6
7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23
-Tasks: 9378
-Not balancing load
```

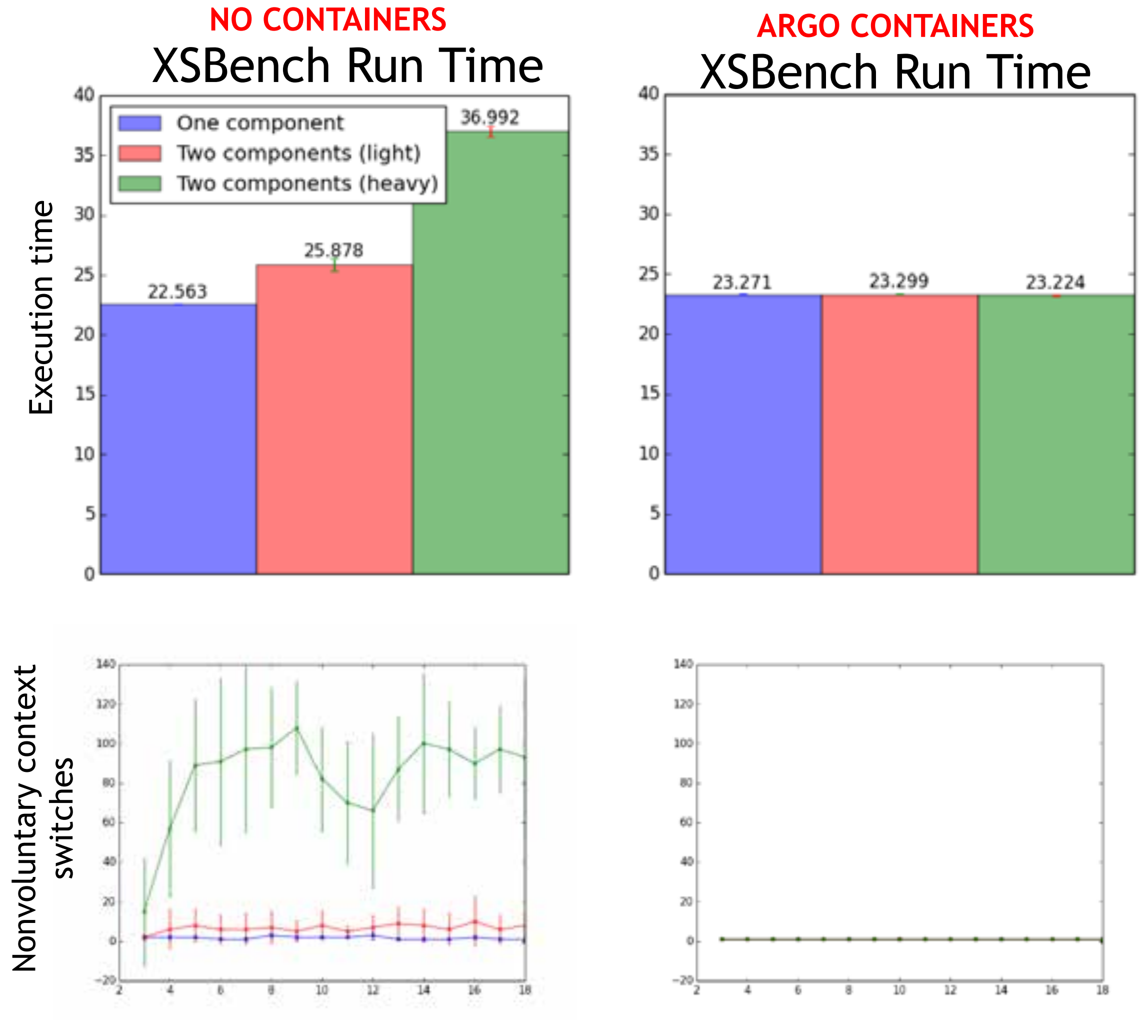
Argo Container
Memory Management

- Argo adds new capabilities to support complex memory hierarchies
- Our *Finer-Grained Memory* mechanism improves memory management
  - Linux memory partitioning works at NUMA node granularity
  - Argo adapts NUMA to provide logical blocks
    - Arbitrary block size, control over physical location
    - Provides first-level abstraction for managing deeper memory hierarchies
      - different partitioning granularities at different hierarchy levels



Initial version well suited for "coherence islands" and can be extended to even deeper layers (*NDA)

Impact of Argo Containers



The Test:

- Dual 12-core Intel Haswell
  - (48 HW threads)
- Test 1: XSBench on 46 threads
- Test 2: XSBench/46 + Other/4
- Test 3: XSBench/46 + Other/40

Argo Container Observations:

- Shows fantastic promise!
- Easy to configure and have low overhead

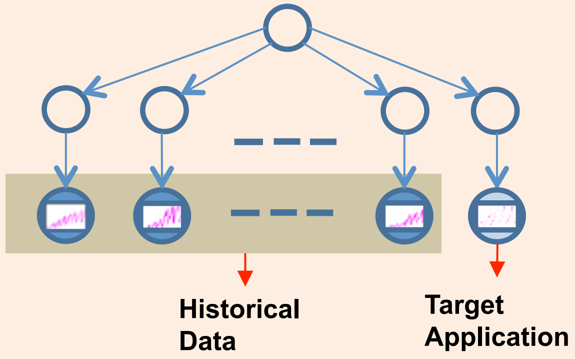
Future:

- Add controls for power, fabric
- Provide different execution environments in each container (kernel features, namespaces – think Docker/Shifter)

Argo Power: NodeOS/R
What are we building?

- *Mechanism*:
  - Tools & APIs for measuring, controlling, and managing power for HPC apps
    - HPC benefits from precise resource management
    - Coordinate app needs and RTS with low-level knobs
- *Policy*:
  - Learning, prediction, and autotuning
  - Goal based:
    - HPC App performance
    - Power

Hierarchical Bayesian Learning "convexifies" space into Pareto-optimal configurations

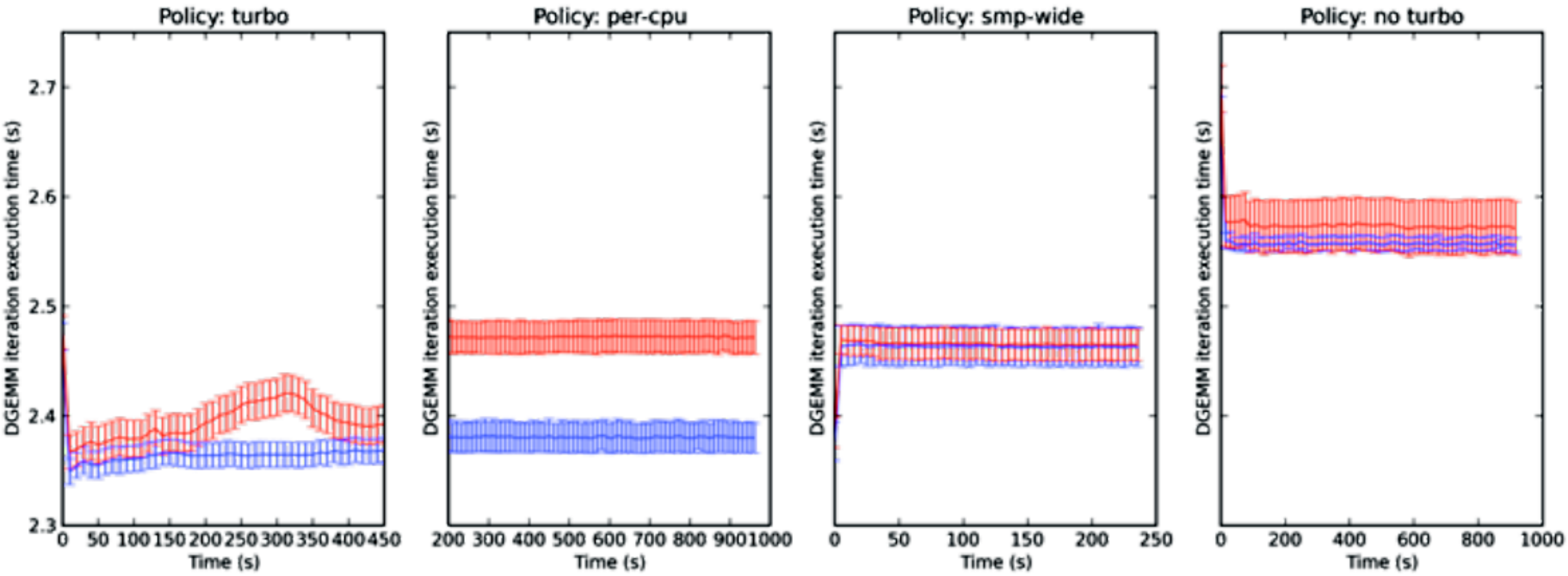


Why is this important?:

- Process & power variation increasing
- Imbalance wastes power (speed)
- Turbo Boost algorithm "greedy", can't see future syncs.
  - Co-design would help

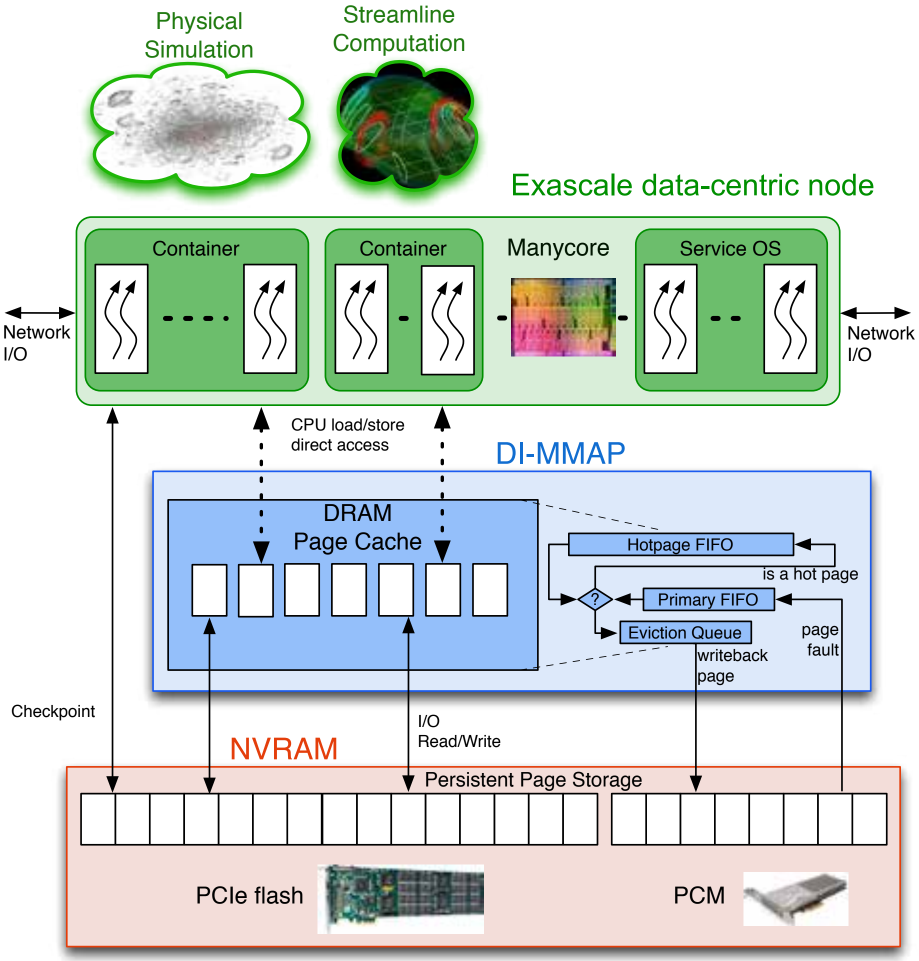
Argo Power: Node OS/R Impact

- Enable machine-learning; finding right knob settings at runtime.
- Low-level monitoring and control linked to higher-level application runtime provide real performance improvements
  - Enables thermal-aware thread management
- Glue layers to existing APIs such as PAPI, TAU
  - Application can explore power and performance via friendly APIs
- User-friendly control knobs for variability
  - Higher performance / higher variability vs. lower variability / lower performance



DI-MMAP and Looking to Hierarchical Memory for Exascale

- Non-volatile memory can extend main memory by orders of magnitude with a suitably optimized DRAM cache
- Standard Linux page cache is not optimized for HPC use cases
  - high memory pressure
  - substandard performance

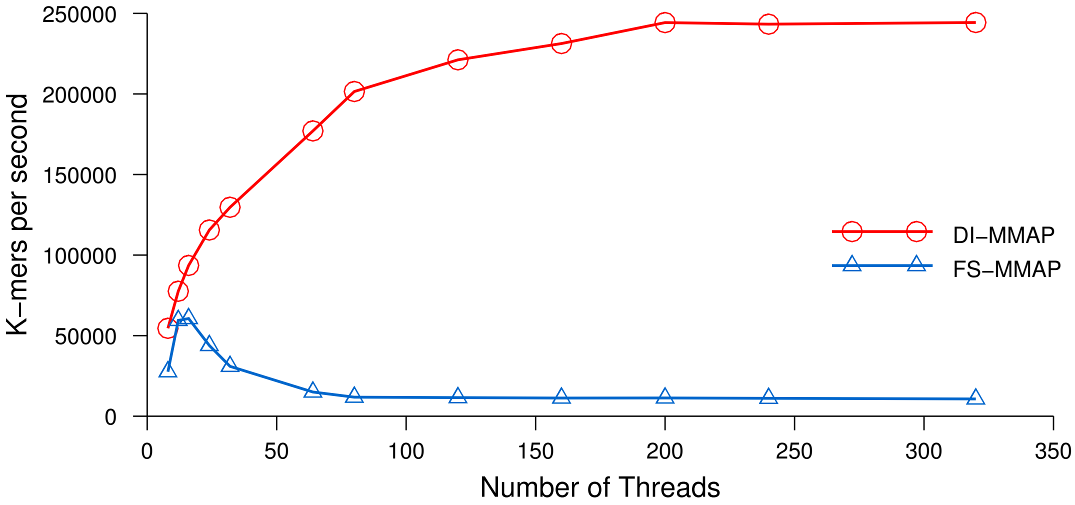


DI-MMAP: What are we Building?

- Scalable, HPC implementation of DRAM cache for NVRAM
- Concept can be leveraged for multiple RAM layers
- Integrates non-volatile random access memory into the HPC memory architecture.
- Enable scalable out-of-core computations for data-intensive workloads.
- Optimized for exascale compute node architecture: massively parallel asynchronous threads
  - custom policies for allocation and distribution of DRAM cache (size, NUMA placement)
  - Prefetch to support user-level threads (Argobots)

DI-MMAP – Impact

- Significant performance improvements over Linux mmap with out-of-core data intensive workloads
  - 3-4x on Livermore Metagenomics Analysis Toolkit
  - 2.4x on Graph500 Scale 40
- Transparent support eases portability for many existing HPC applications
- **Future:**
  - Multiple caching policies customized to HPC applications
  - Application-tailored prefetch



Bioinformatics database query:
DI-MMAP vs linux mmap

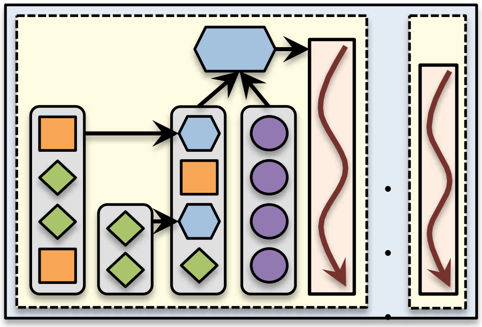
Argo Node OS/R Adoption and Deployment

- All code is Open Source
  - Argo Containers: kernel patches, user-space tools (to be officially released)
  - Argo Power: kernel modules, user-space tools, libraries `https://github.com/scalability-llnl/`, `https://github.com/coolr-hpc/`
  - DI-MMAP: kernel module, user-space tools `https://bitbucket.org/vanessen/di-mmap`
  - HPC-Sched: kernel patches (to be officially released)
- Deployment
  - Success working with CESAR to improve Node OS/R performance
  - Some components already in use by applications (DI-MMAP)
  - Argo Power components from LLNL (libmsr) deployed in production
  - We expect to test all components on CORAL systems
- Vendor Collaboration
  - Some Argo Power components to be included in RedHat. PAPI too

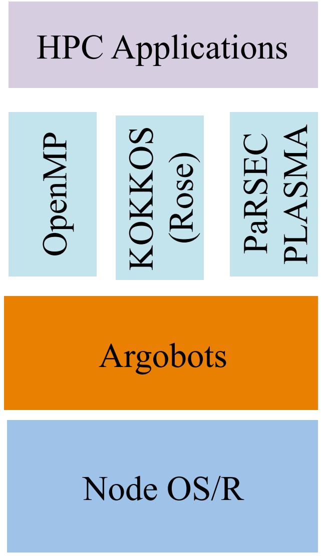
Argo Node OS/R Roadmap

- 2016:
  - Centralized Node Resource Manager co-scheduling CPU, memory, network, and power resources
- 2017:
  - Optimized support for communication libraries (optimal memory mappings for put/get, fast thread wakeup)
  - Containers with dynamic power budgets
  - Callbacks to Fault Manager in user space on system fault events
- 2018
  - Integrated, hierarchical memory management including on-package and off-package DRAM, memory on GPU, NVRAM
    - including partitioning, software-based caching and prefetching, callbacks into runtime for latency hiding
- 2019:
  - System call forwarding and optimized support for on-node storage, including draining policies of burst buffers
  - Ensure optimal execution of workflows
    - caching/prefetching of executables, inputs, outputs; result coalescence, etc.

Argobots: What are we building?



- Lightweight, integrated, thread & task rts for 100X increase in on-node parallelism
- Built for hierarchical memory domains
- Elastic by design – adjusting OS/R resources (containers, memory, etc.) on the fly.
- Designed for tight integration with HPC interconnect
- Designed as middleware
- “I can help solve your on-node AMR adaptive/dynamic parallelism”



HPC Applications

OpenMP

KOKKOS (Rose)

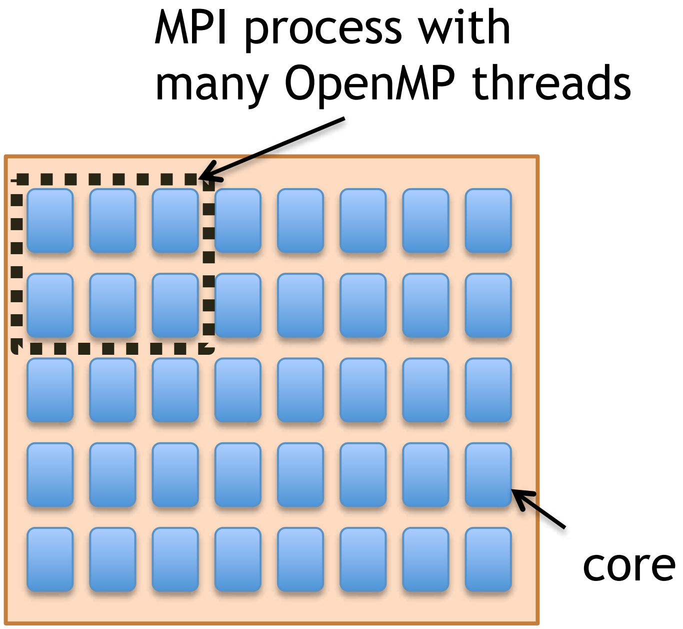
PaRSEC PLASMA

Argobots

Node OS/R

Today: Massive On-node Parallelism

- MPI+OpenMP is sufficient for many apps, but implementation is poor
  - Today MPI+OpenMP == MPI+Pthreads
- Pthread abstraction is too generic, not suitable for HPC
  - Lack of fine-grained scheduling, memory management, network management, signaling, etc.
- New runtime will significantly improve MPI+OpenMP performance *AND* support emerging programming models



Current situation:

- One or more MPI processes per node
- Each MPI process has limited internal parallelism typically with OpenMP
- MPI Process communication is often serialized

What are the Shortcomings today?: Pthreads (1/2)

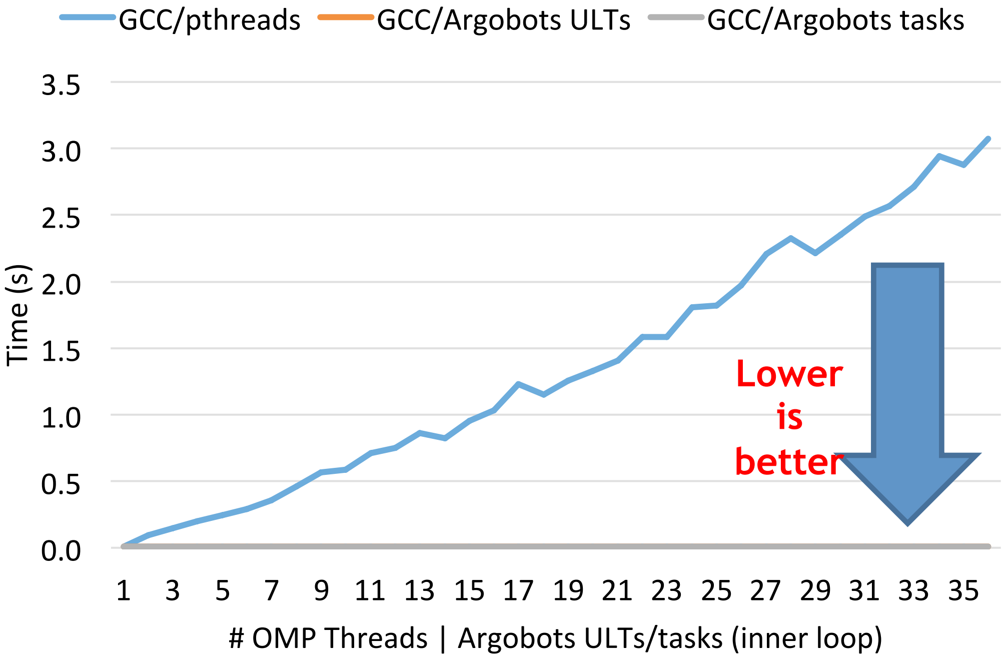
Nesting

```
int in[100][100], out[100][100];

#pragma omp parallel for
for (i = 0; i < 100; i++) {
   petsc_voodoo(i);
}

petsc_voodoo(int x)
{
   #pragma omp parallel for
   for (j = 0; j < 100; j++)
      out[x][j] = cosine(in[x][j]);
}
```

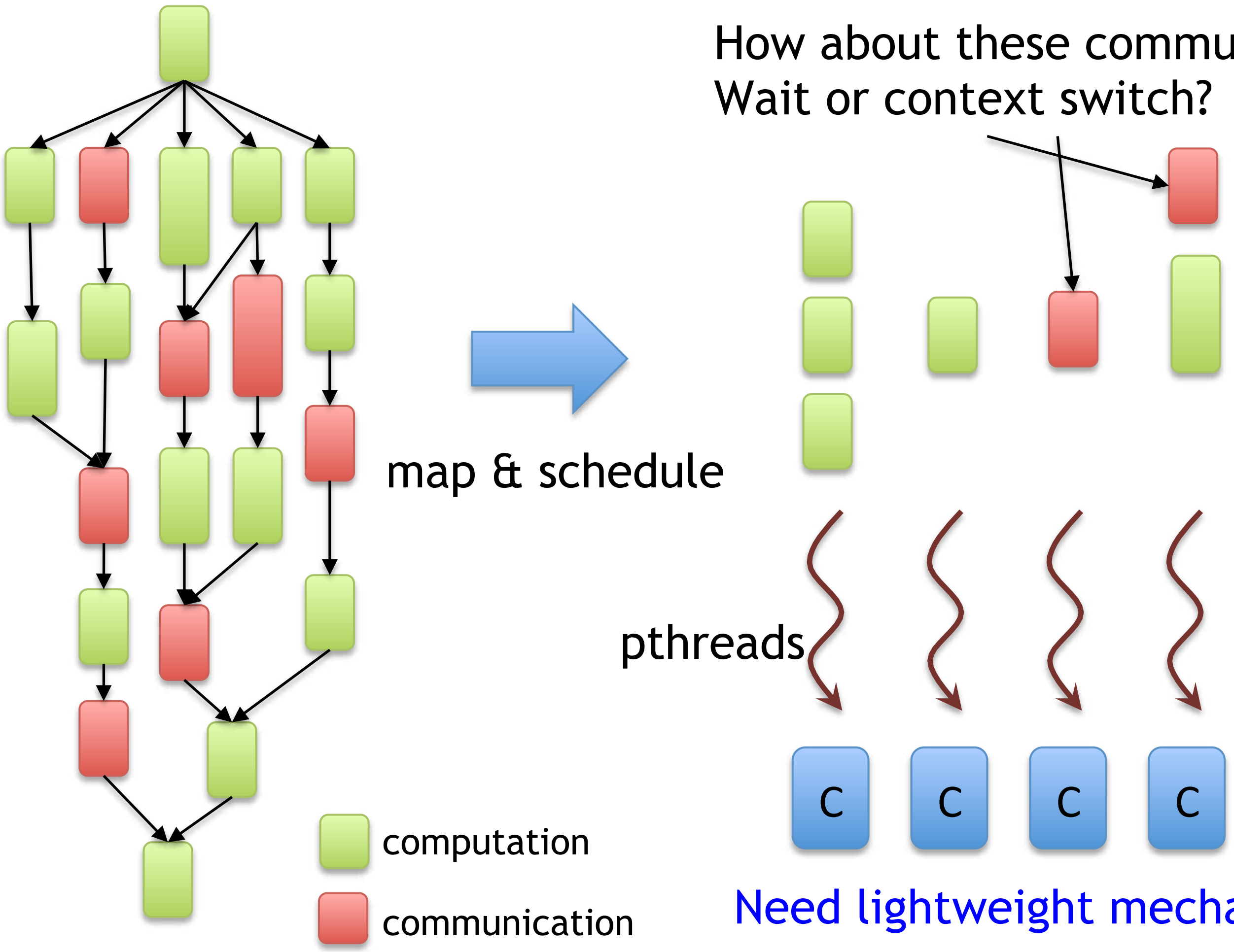
Execution time for 36 threads in the outer loop



Why is traditional OpenMP's performance so bad? The compiler cannot analyze petsc_voodoo to know whether the function might ever block or yield, so it has to assume that it might. Therefore a stack is needed to facilitate it. Creating additional pthreads for each nesting is the simplest way to achieve this.

What are the Shortcomings today?: Pthreads (2/2)

Tasks of application mapped to a group of pthreads



How about these communications?
Wait or context switch?

🟩 computation

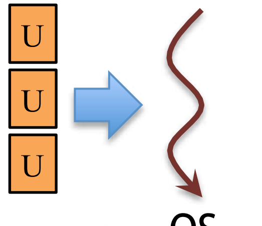
🟥 communication

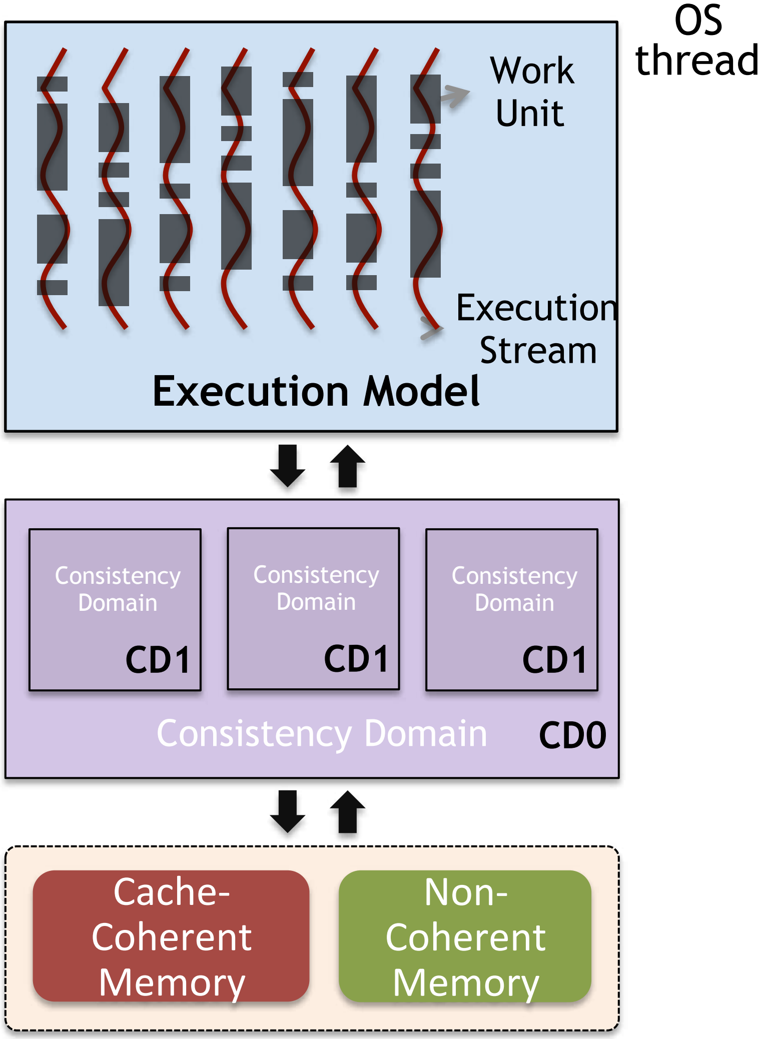
Work units intermixed with blocking calls (such as communication calls) can cause idle cores

Need lightweight mechanisms to switch tasks!

Our Approach: Argobots

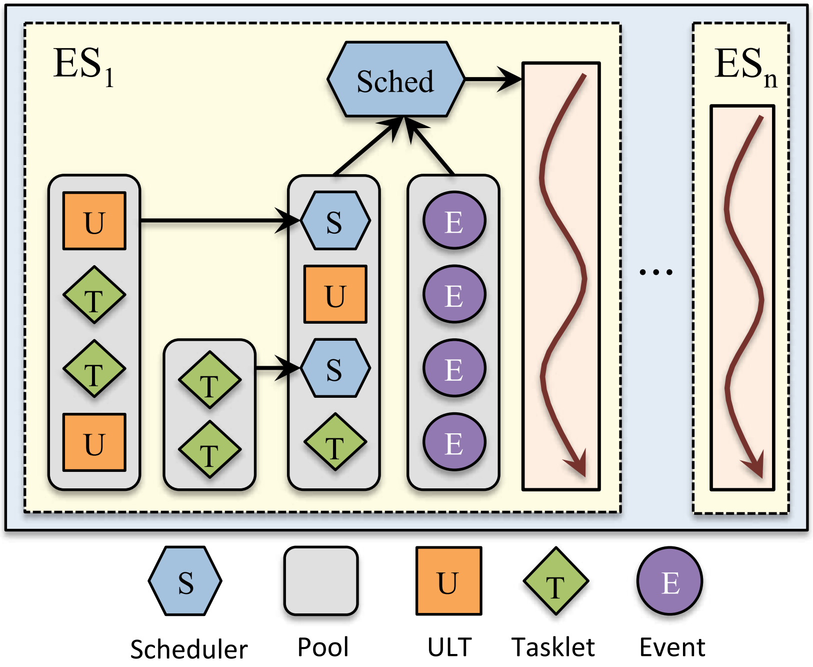
- **Lightweight Low-level Threading/ Tasking Framework**
- Massive parallelism
  - **Exec. Streams** guarantee progress
  - **Work Units** execute to completion
- Separate *mechanism* from *policy*
  - Users can write their own scheduler
  - OpenMP knows more… use it…
- Clearly defined memory semantics
  - Consistency domains
  - Software can manage consistency
  - Support explicit memory placement and movement





Argobots Execution Model

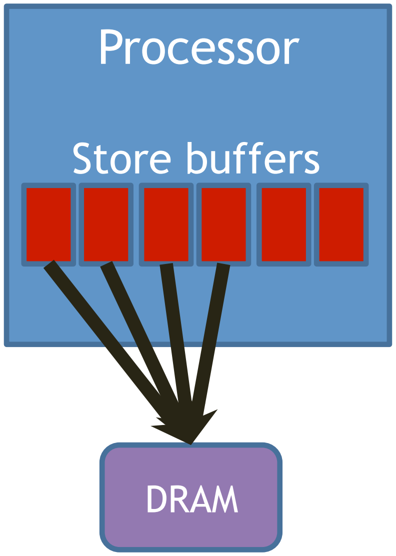
- **Execution Streams (ES)**
  - Sequential instruction stream
    - Can consist of one or more work units
  - Mapped efficiently to a hardware resource
  - Implicitly managed progress semantics
    - One blocked ES cannot block other ESs
- **User-level Threads (ULTs)**
  - Independent execution units in user space
  - Associated with an ES when running
  - Yieldable and migratable
  - Can make blocking calls
- **Tasklets (Intra-node)**
  - Atomic units of work
  - Asynchronous completion via notifications
  - Not yieldable, migratable before execution
  - Cannot make blocking calls



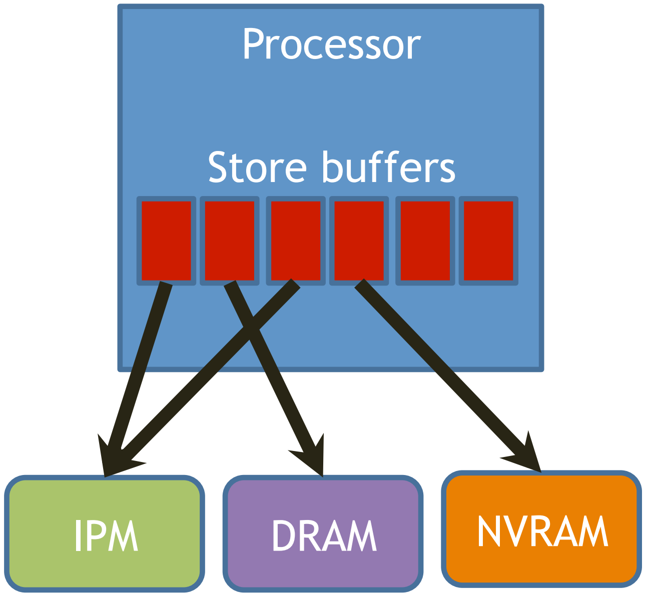
- **Scheduler**
  - Stackable scheduler with pluggable strategies
- **Sync primitives**
  - Mutex, condition variable, future
- **Events**
  - Communication triggers

Argobots Memory Model

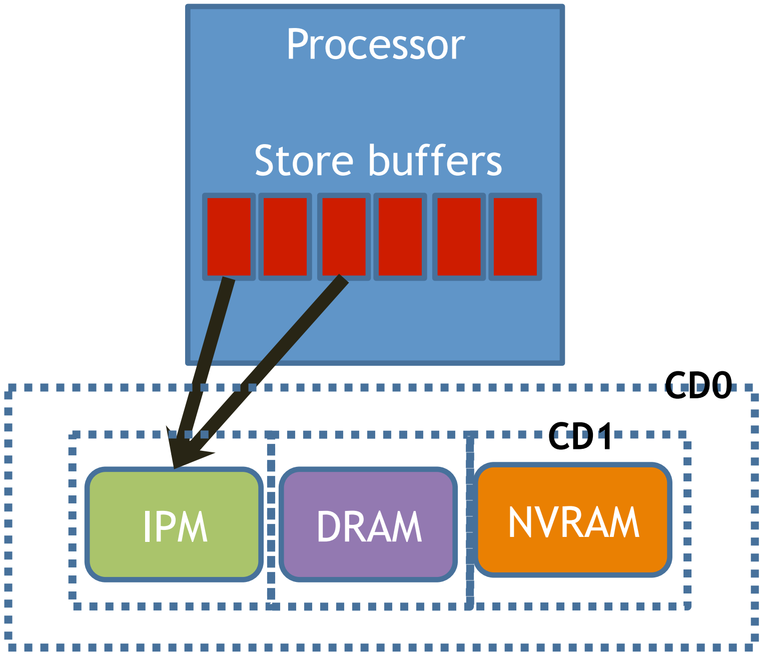
- Memory operation ordering consistency
  - Needed for correctly exposing visibility of data to other threads/cores
  - Either through explicit memory fences or implicitly through any atomic operation



Today's processors do not distinguish memory operations. A memory consistency operation would flush all stores to memory.

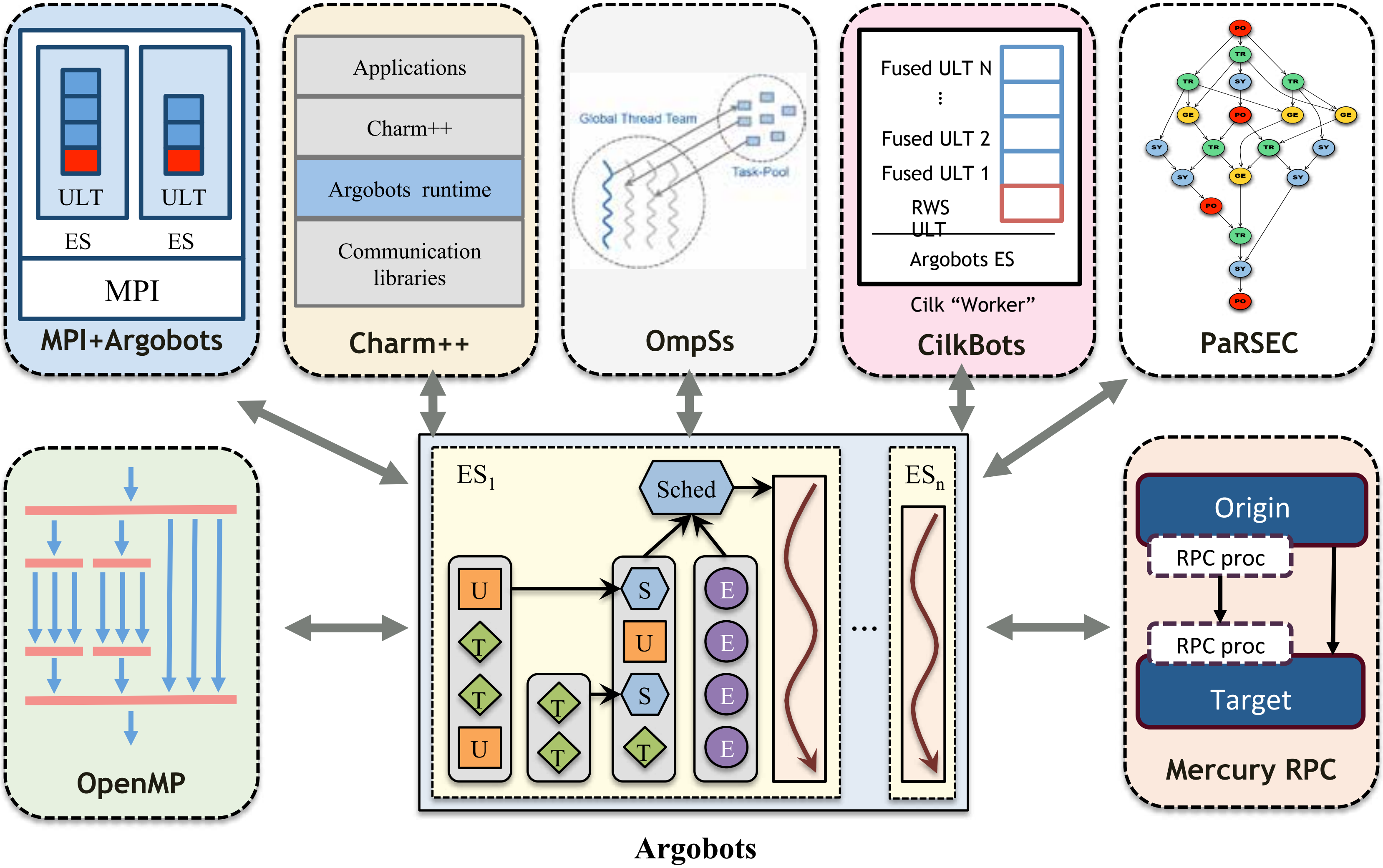


Scaling today's technology to future heterogeneous memory would result in any memory consistency operation to be bound by the slowest memory.



Argo Approach

Argobots Ecosystem



External Connections

GridFTP, Kokkos, RAJA, ROSE, TASCEL, XMP, etc.

Integrating OpenMP with Argobots

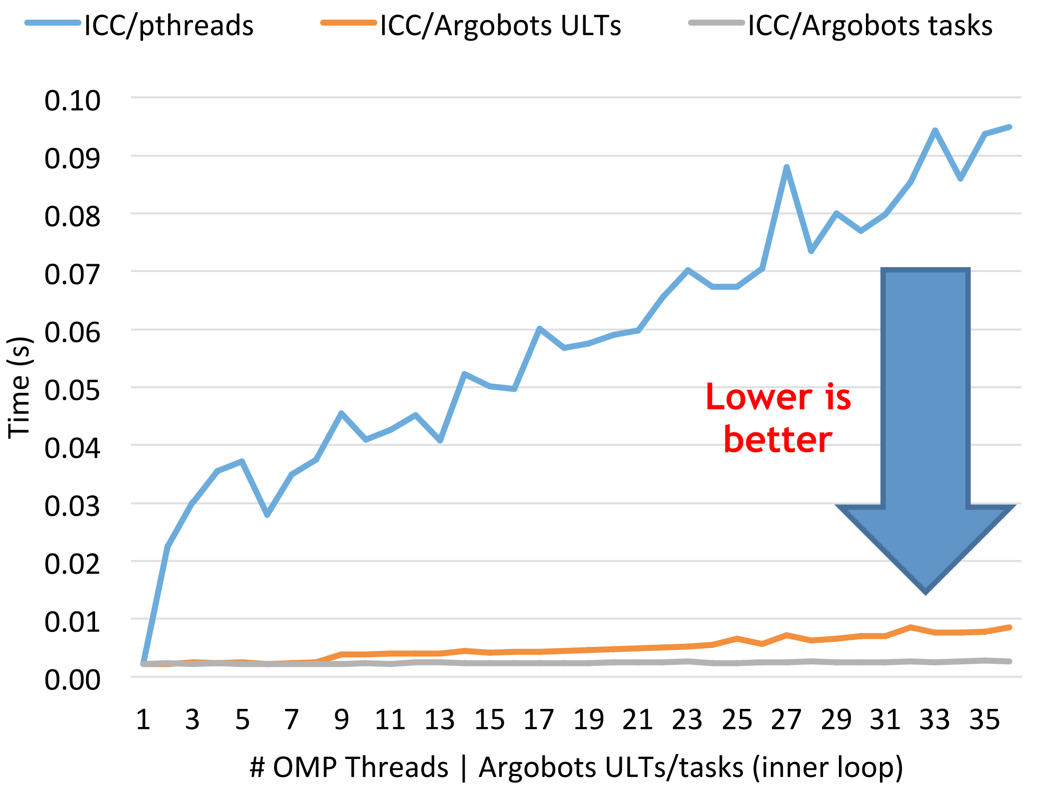
Nesting

```
int in[100][100], out[100][100];

#pragma omp parallel for
for (i = 0; i < 100; i++) {
   petsc_voodoo(i);
}

petsc_voodoo(int x)
{
   #pragma omp parallel for
   for (j = 0; j < 100; j++)
      out[x][j] = cosine(in[x][j]);
}
```

Execution time for 36 threads in the outer loop



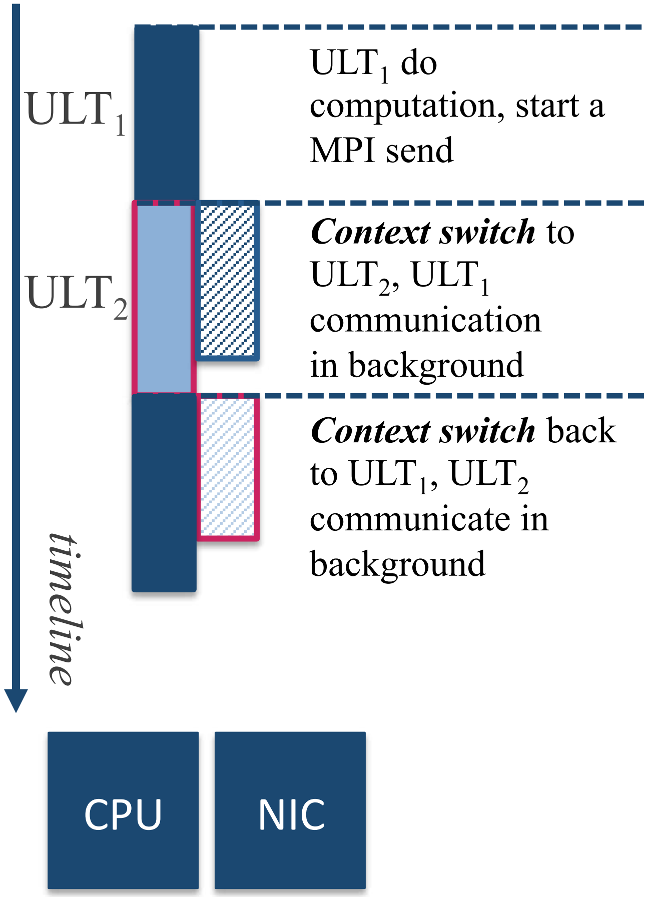
Lower is better

Some overhead is added by creating ULTs instead of Tasks

MPI interoperation with Argobots

Overlap communication with computation using ULT

- Lightweight
  - ULT does not execute concurrently using additional hardware resource, but takes turn to run by context switching
  - No lock needed between two ULTs in the same kernel thread
- Asynchronous communication
  - Helps turn an MPI blocking call to a nonblocking one
  - Decouples the operation of “send start” and “send complete”
- Dynamic Tasking
  - Providing automatic overlap based on task-graph dependencies

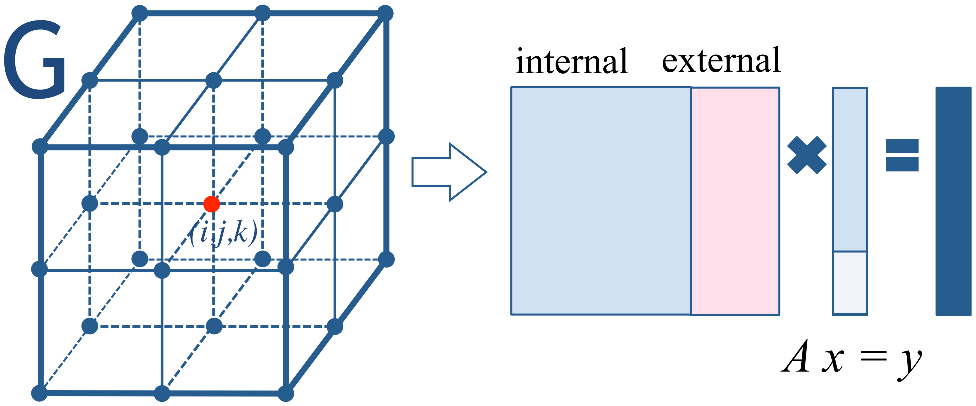


ULT$_1$ do computation, start a MPI send

Context switch to ULT$_2$, ULT$_1$ communication in background

Context switch back to ULT$_1$, ULT$_2$ communicate in background

Application: HPCG



- High Performance Conjugate Gradient (HPCG)
  - Solves $Ax=b$, large and sparse matrix
- Hiding Global Collective Communication
  - overlap communication and computation between iterations
  - fork a ULT to do ult_ddot and join in the next iteration
- Hiding Neighborhood Communication
  - for each neighbor, fork a ULT to do halo exchange and a small part of SpMV (communication)
  - main ULT computes local spmv (computation)

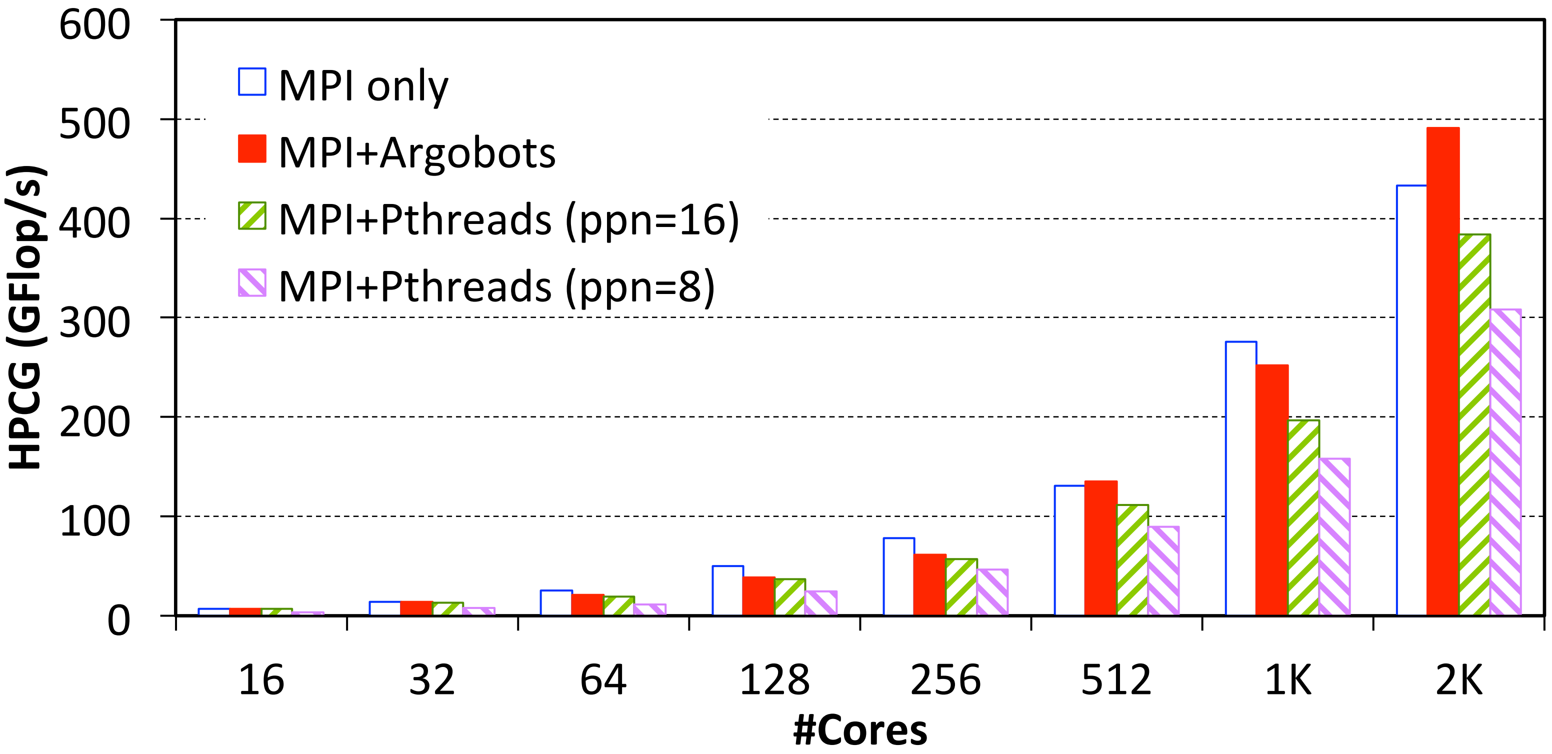
HPCG

```
for k = [1: max_iter]:
    MG(A, r, z);
    if k > 1:
        ult_join (thread);
        if (normr <= tolerance) break;
    ……
    ult_fork(ult_ddot, &param, &thread)
```

SpMV

```
SpMV(A, x, &y):
    for each neighbor:
        ult_fork(es, ult_spmv, &t[i]);
    for i in [0: nRows]:
        ult_yield();
        for each j in row i:
            y[i] += val[j] * x[idx[j]];
    for each neighbor i:
        ult_join(t[i]);
```

Preliminary Results: HPCG

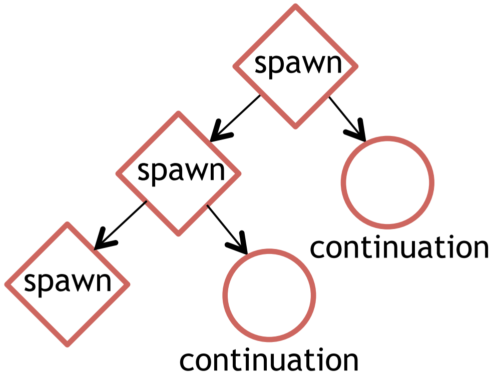


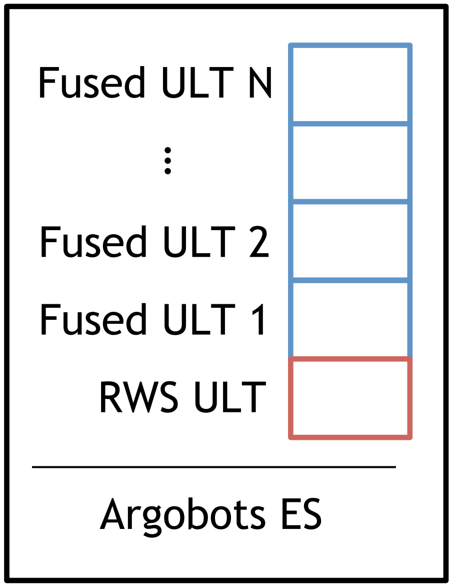
- On 2,048 cores, HPCG using MPI+Argobots shows performance improvement of **13.4%** over MPI-only version, or **27.4%** over MPI+Pthreads version.
  - As core number increases, the benefit of communication hiding begins to reveal. DDOT% increases from 0.62% on 16 cores to 36.8% on 2,048 cores.

Cilk over Argobots (Cilkbots)

- Cilk built on Argobots
  - Worker (previously pthread) is now an Argobots ES
  - Cilk work stealing scheduler runs in a Argobots ULT
  - **Modify Cilk compiler to generate Argobots code**
- Fuse multiple spawn trees to improve locality
  - Distinct spawn trees require their own stack
- Specialized Locality-aware Argobots scheduler

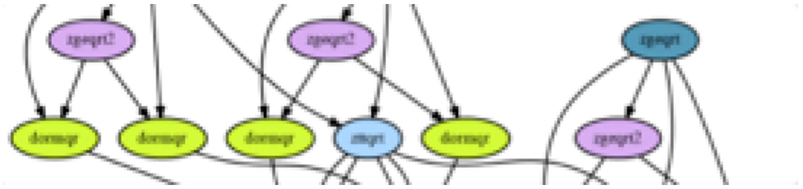
- Key Message: Argobots can improve BOTH existing OpenMP and *NEW* programming models



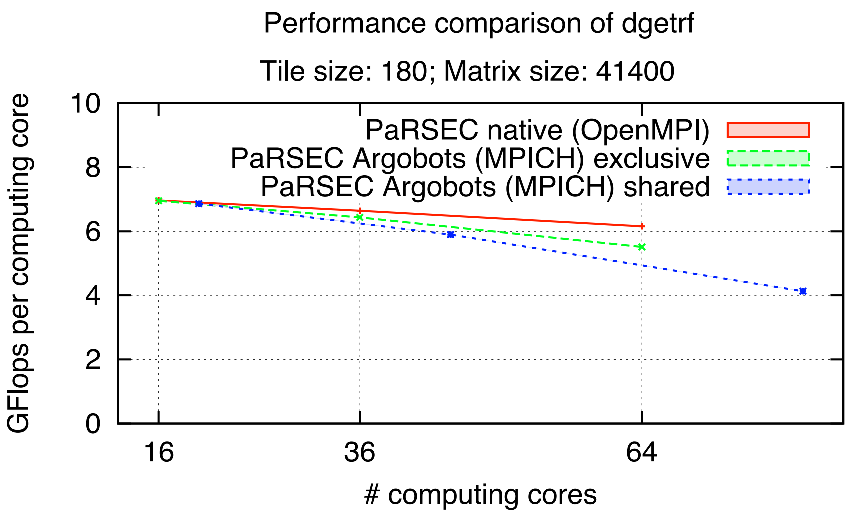


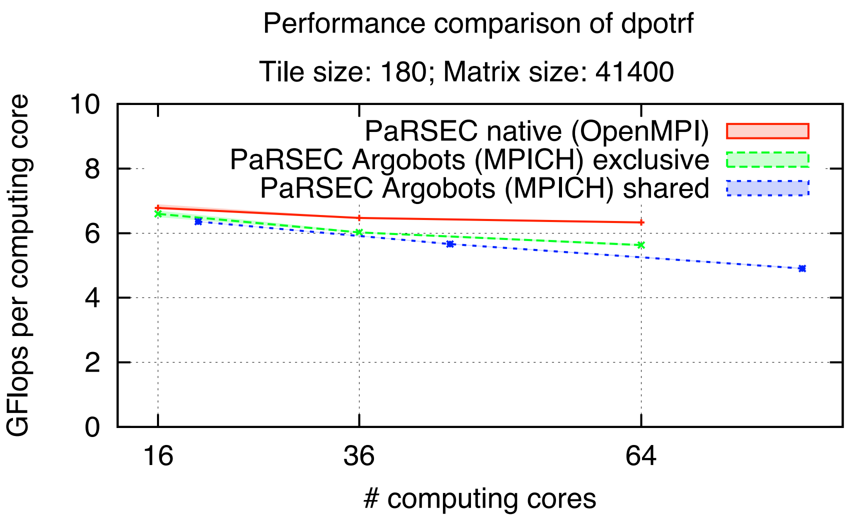
Cilk “Worker”

PLASMA/PaRSEC with Argobots



- PaRSEC: framework for architecture-aware scheduling of micro-tasks on many-core
  - Compiler optimizes tasks; Developer describes dependencies
  - Separate algorithms from data distribution





Argobots Adoption and Deployment

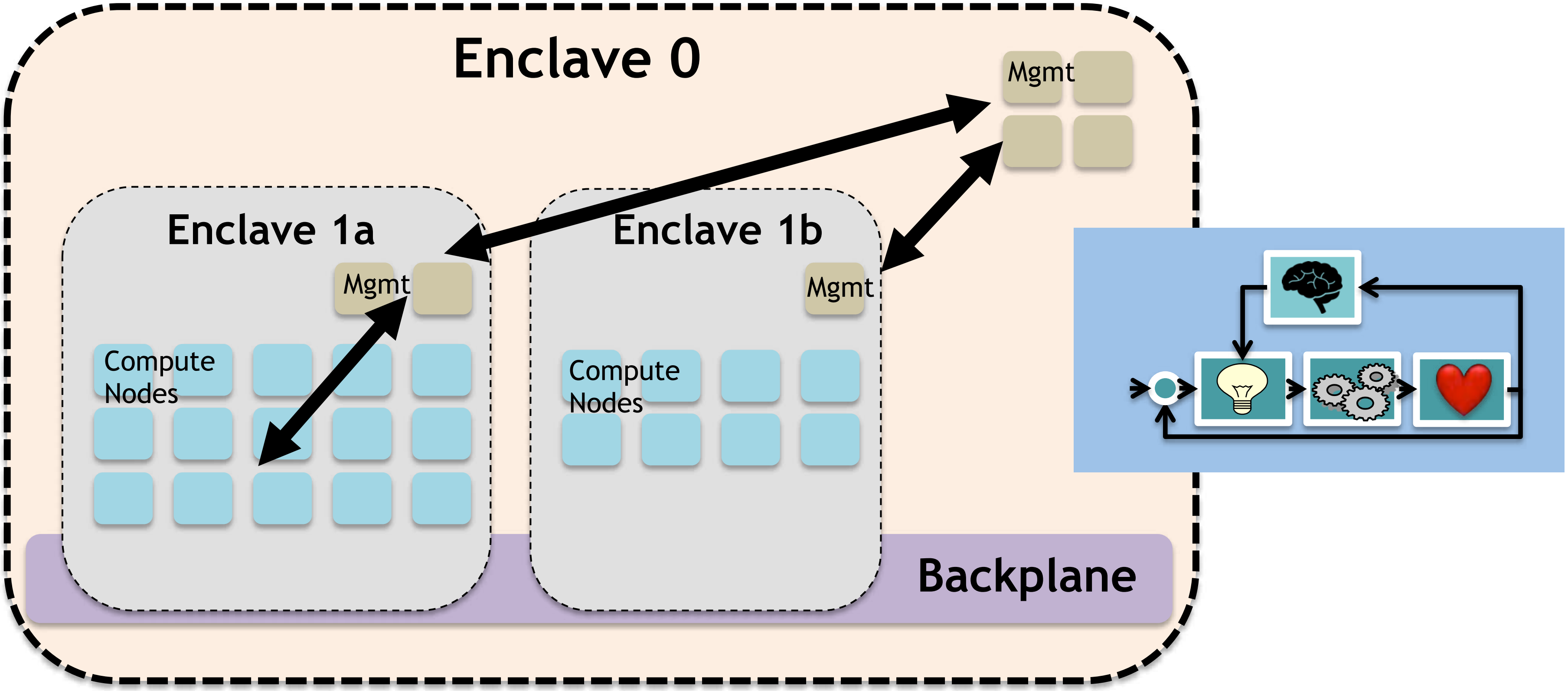
- All code is Open Source
  - `http://git.mcs.anl.gov/argo/argobots.git`
- Deployment: (remember that customers are high-level Runtimes…)
  - Success working with CESAR to improve XSBench
  - In Testing: OpenMP, PLASMA (PaRSEC), Charm++, Cilk,
  - In Future: KOKKOS/RAJA, OmpSs, etc.
- Other activities:
  - Exploring a broad community effort to standardize interfaces
    - Improved interoperability between MPI and OpenMP
    - Improved interoperability with emerging communication standards (OFI, UCX)
  - An open-source LLVM derived OpenMP reference implementation over Argobots

Argobots Roadmap

- 2016:
  - OpenMP integration and deployment
  - MPI integration
- 2017:
  - Network integration, lightweight thread activation
  - Performance/correctness tools
  - Open source LLVL OpenMP to Argobots
- 2018:
  - Extend hierarchical memory model, heterogeneous memory, NVRAM
  - Progress estimation for power management
  - Heterogeneous hardware (Big/Little cores, CPU/GPU)
- 2019:
  - Machine learning / autotuning for memory hierarchy, power, and concurrency in Argobots

A Hierarchical Exascale System

Enclaves and *Backplane and the Global OS/R*



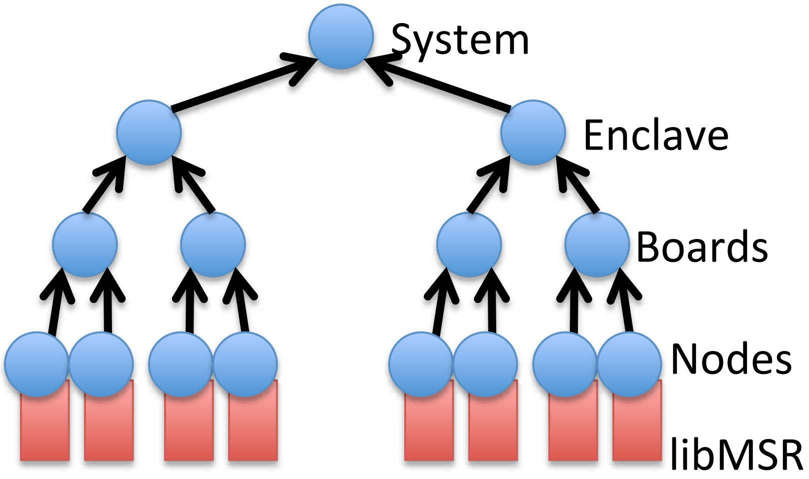
- Recursive hierarchy enables new capabilities and workflows
- Machine learning / autotuning for closed-loop control system
- Enables writing meta-programs for enclaves:
  - task/load manager, many-task workflow engines, power management, resilience response, coordination of coupled components, etc.

The Argo Backplane

- Benefits:
  - **Applications:** *Mechanisms* for handling resilience, live performance data (introspection), and responding to dynamism
  - **System**: Mechanisms for hierarchical management of power, workflows, NVRAM, etc.
- Key Features:
  - Provides (scalable) global view
  - Provide aggregation services: Reduction, Filter, etc.
  - Uses Pub/Sub
  - Well suited for Application-based resilience
  - Key-Value Store and Dynamic Trees for implementing the match-making and diffusion, Reduction of events.

Adaptive Power Management

- Fixed power budget for a machine
- System must optimize HPC performance
- Hierarchical and adaptive control



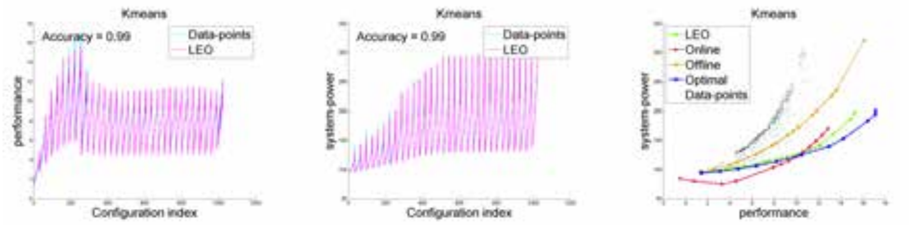
System

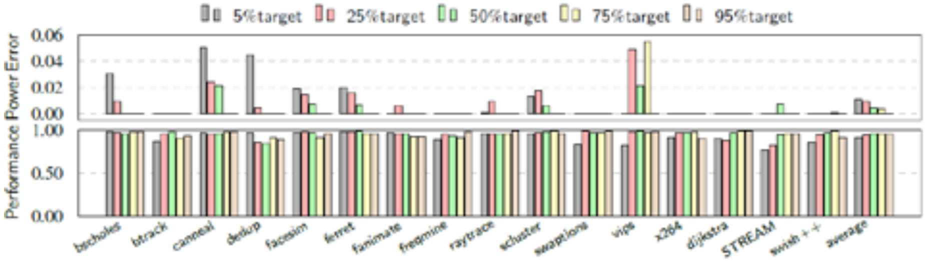
Enclave

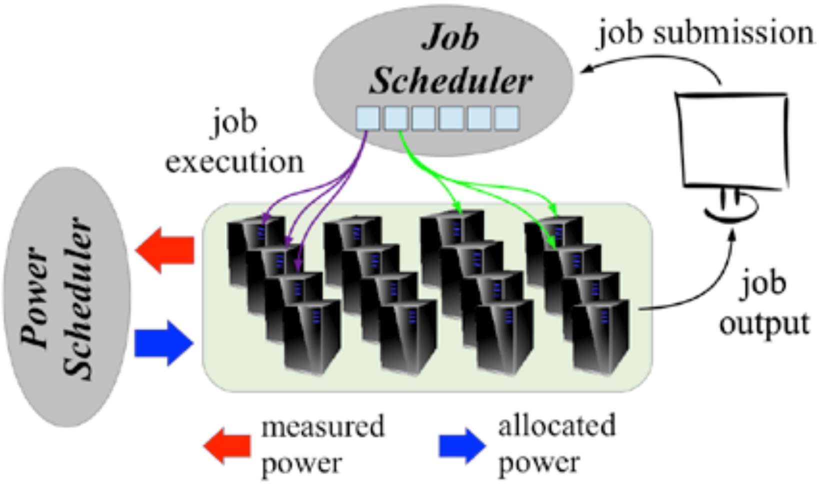
Boards

Nodes

libMSR







Approach

- Publish power on Argo backplane
- Adaptive decision algorithm at each layer of the tree
  - Collect unused power and propagate up
  - Detect power constrained executions and distributed power down
- Policy guidance at each level

Results on Sandy Bridge Server

- Learning finds pareto-optimal tradeoffs and control provides power guarantees

The POW Scheduler: Power Control through the Backplane

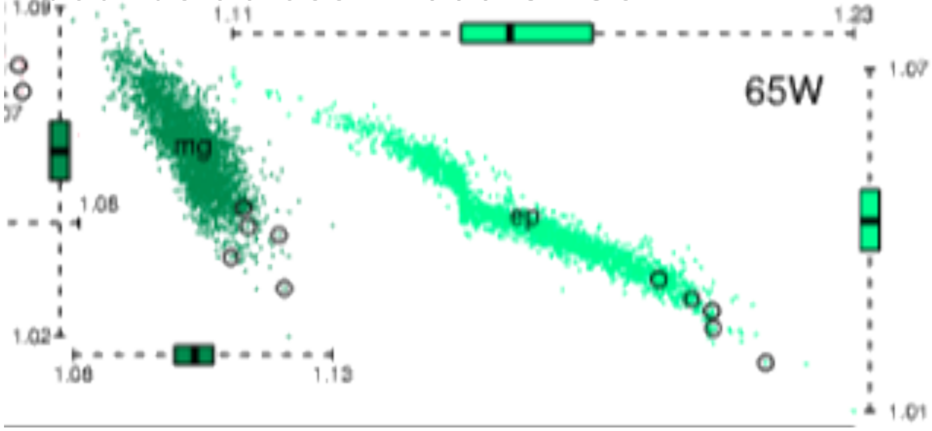
libMSR: A node OS level API for Power/Thermal data collection

- Read thermal and power MSRs
- Matching kernel driver (msr-safe)
- Active processor/DRAM power control through RAPL
- NodeOS level API for ARGO

Status

- Support for Sandy/Ivy-Bridge
- Port to Haswell nearly complete
- https://github.com/scalability-llnl/libmsr

Study: processor variation under power bounds across ~2000 CPUs

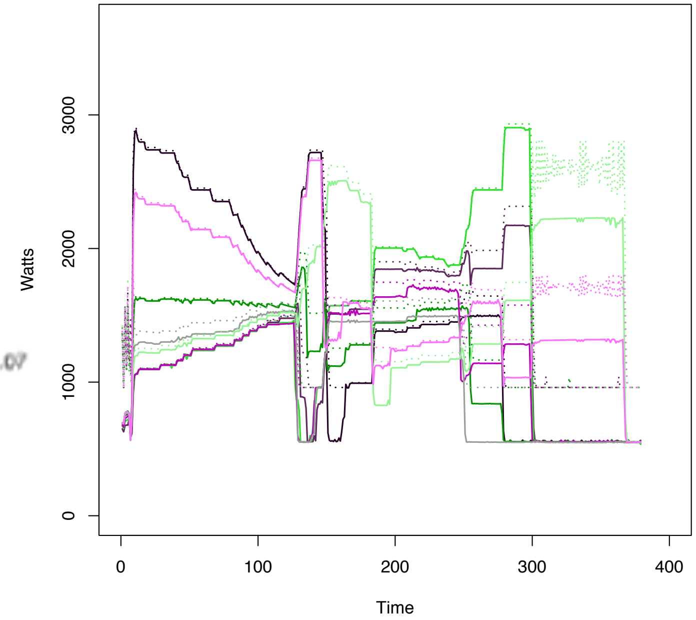


POW: Global Power Scheduling

- Data collected via libMSR/msr-safe
  - Publish data into BEACON/EXPOSE
  - Online aggregation through overlay
- Hierarchical control algorithm to control node local power caps

Status

- Working prototype
- Successful used on LLNL's cab cluster



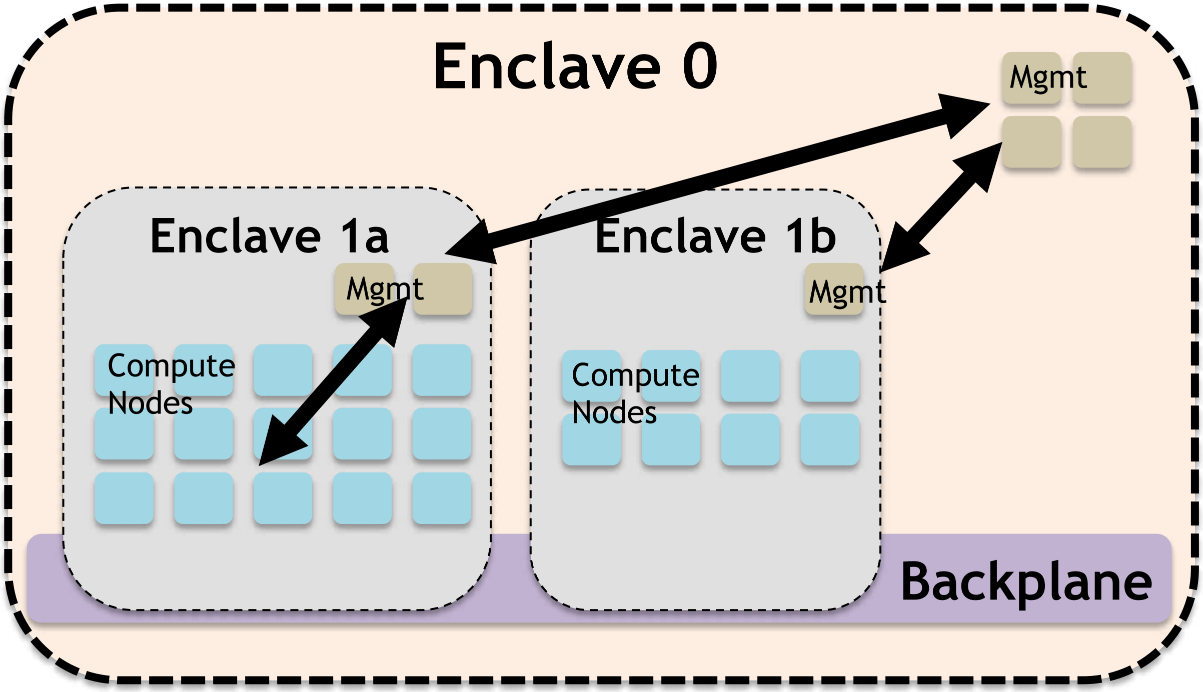
Power usage across eight enclaves (ach color shows data for one enclave) under a global power bound and shifting power between enclaves

Argo Backplane & Global OS/R Adoption and Deployment

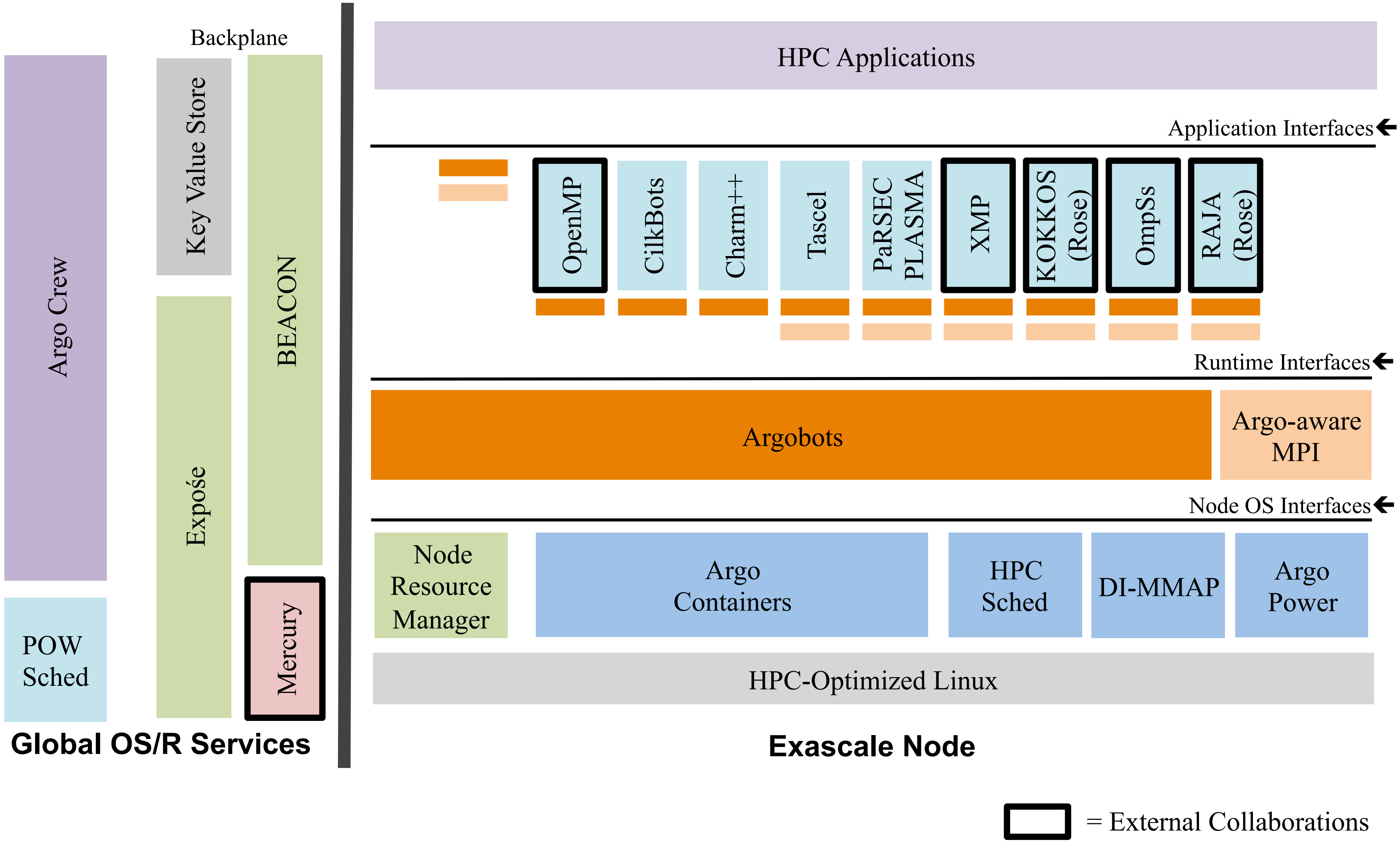
- All code is Open Source
  - `http://git.mcs.anl.gov/argo/argobots.git, leo.cs.uchicago.edu, poet.cs.uchicago.edu`
  - https://github.com/scalability-llnl/libmsr
  - Backplane pieces: Expose and BEACON being released

Global OS/R Example: Power (SC2015)

- Set power budgets per enclave (Global Resource Manager)
- Enclave gives each node a power level (Enclave Resource Manager)
- Nodes adjust Threads & Cores to meet goals (Node Resource Manager)
- Entire system is dynamic, and automatically adjusts



The Argo Architecture



Exploring ECP…

ECP

- Argo Node OS/R
  - Containers
  - Power
  - HPC Sched
  - DI-MMAP
- Argobots
  - OpenMP LLVM
  - Kokkos/Raja
  - PaRSEC
- Backplane
  - Resilience Events
  - (parts…)
  - Enclave Support
- Global OS/R
  - Power Sched
  - (parts…)

Let's Talk

- Argo Node OS/R
  - I/O Forwarding
  - Hierarchical memory SW mgmt
  - Simple opt.
- Argobots
  - Big/Little cores
  - Hetero cores
  - Charm++
- Backplane
  - Workflow
  - FTI
  - Efficient Enclaves
  - (parts…)
- Global OS/R
  - (parts...)

Research

- Argo Node OS/R
  - Machine Learning Optimization
- Argobots
  - Cilkbots, TASCEL
  - OmpSs
- Backplane
  - (parts…)
- Global OS/R
  - (parts...)

Questions

ECP questions:

1. The goals of your project and its current status (see next slides for sub-points)

2. What are the specific ties to identified requirements of the applications, other software components?

3. Will the developed software technologies be mature enough to be part of the software stack on exascale systems expected to be selected in 2019 and installed in 2023?

4. What do you feel are the key challenges posed and opportunities offered by exascale systems for your specific area?

5. What is the R&D that you would like to carry out within the ECP?

6. What research remains for your project's outcomes to benefit key DOE applications?

7. How would the proposed activities build on the research you have been carrying out with ASCR Research funding?

8. What are the proposed activities that you believe would contribute to the ECP?

9. Your roadmap/timeline for maturing the software technologies and deploying them on exascale platforms, with a few intermediate milestones or decision points (forks in the roadmap). The timeline is of particular importance in selection what the ECP will include in the development plans.

10. Highlight your X-stack or OS/R activities that would help DOE exascale apps achieve ECP performance, efficiency and resilience performance goals on 2023 hardware and system architectures selecting what the ECP will include in the development plans.

Project Goal and Status (Q1)

- Do you release your software as open source? YES
  - Open Source is in our DNA
  - Many components released, some preparing for final approval
- Do you have DOE/NNSA users of your software? YES
  - Argo Power components, DI-MMAP, etc.
- Have facilities, vendors, or ISVs picked up your software? YES
  - DI-MMAP
  - Argo Power components from LLNL
  - Collaboration with vendors on integration plans:
    - Argo OS/R: memory management improvements: IBM (Sexton), Intel (Wisniewski)
    - Backplane: Intel's new "PMIx Error Handling infrastructure" is based directly on the GIB concept with a similar API
    - Globalview: Part of Cray and Intel and OpenStack working group. Overall design has already impacted Intel's work – in particular, for power management and Global Information Bus
  - Adoption plan:
    - Make available as an option on CORAL systems in collaboration with LCFs
    - Standardization effort (Argobots):
      - A standardized user-level threads interface in cooperation with the broader community
      - Argobots as an implementation of the standardized user-level threads
      - An open-source LLVM derived OpenMP implementation over Argobots
    - Transfer to the community
      - Will organize Tutorials at SC and other major event in US (Cluster, HPDC when in USA)

Project Goal and Status (Q1)

- What is the support model for your software?
  - Supportedby research group
  - Our support model includes/will includes regular releases, bug fix releases and providing support to users via project mailing lists
- Are there any applications in particular that the outcomes of your project are targeting? YES
  - CESAR, HACC, ACME, NEK5000, FLASH, etc.

Question Q2 and Q10

Q2) What are the specific ties to identified requirements of the applications, other software components?

- All applications and workflows need monitoring and management of execution dynamisms
- Argo components will help support new application modes (Argo container, Exposé), and provide new capabilities for NVRAM, Power and workflow management (DIMMAP, Argo Power and Argo Crew)

Q10) Highlight your X-stack or OS/R activities that would help DOE exascale apps achieve ECP performance, efficiency and resilience performance goals on 2023 hardware and system architectures

- Argo components will help improve performance (Argobot, Backplane, NodeOS)
- Backplane will transport notifications and commands related to resilience
- New capability for power management will help improve efficiency (Argo Power)

Question Q3 to Q6

- Q3) Will the developed software technologies be mature enough to be part of the software stack on exascale systems expected to be selected in 2019 and installed in 2023? YES
- Q4) What do you feel are the key challenges posed and opportunities offered by exascale systems for your specific area?
  - Monitoring, modeling managing the dynamisms (power management, faults) in hardware/software
  - Proposing relevant interface to provide dynamisms information and enable management capabilities
- Q5) What is the R&D that you would like to carry out within the ECP?
  - See previous slides
- Q6) What research remains for your project's outcomes to benefit key DOE applications?
  - See previous slides

Project Goal and Status (Q1)

Q7) How would the proposed activities build on the research you have been carrying out with ASCR Research funding?

- Build on the principles and design of the Argo components
- Leverage experiment results on prototype software to spot limitations and identify gaps
- Improve the prototype software accrodingly

Q8) What are the proposed activities that you believe would contribute to the ECP?

- See previous slides

Q9) Your roadmap/timeline for maturing the software technologies and deploying them on exascale platforms, with a few intermediate milestones or decision points (forks in the roadmap). The timeline is of particular importance in selecting what the ECP will include in the development plans.

- In the previous slides