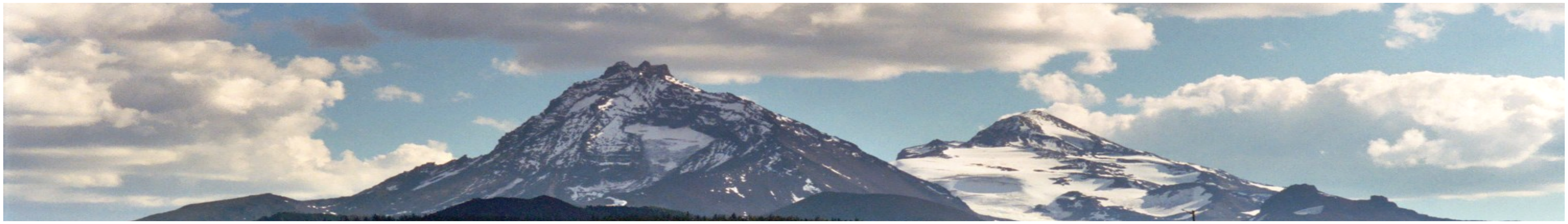


Lecture 9: Parallel Performance Tools Future

Allen D. Malony

Department of Computer and Information Science

UNIVERSITY OF OREGON

Perspective – Parallel Tools Experience

- *Performance* has been the fundamental driving concern for parallel computing ⇨ high-performance computing
  - *Performance observation*, *modeling*, and *engineering* are key methodologies to deliver the potential of parallel, HPC machines
- However, there has always been a strained relationship between *performance tools* and the parallel computing
  - Performance considered necessary, but an afterthought
- There are compelling reasons for performance methods and technologies to be integrated throughout parallel computing
  - To obtain, learn, and carry forward performance knowledge
  - To address increasing scalability and complexity issues
  - To bridge between programming semantics (computation model) and execution model operation

Parallel Performance Engineering

- Scalable, optimized applications deliver HPC promise
- Optimization through *performance engineering process*
  - Understand performance complexity and inefficiencies
  - Tune application to run optimally on high-end machines
- How to make the process more effective and productive?
  - What is the nature of the performance problem solving?
  - What is the performance technology to be applied?
- Performance tool efforts have been focused on performance observation, analysis, problem diagnosis
  - Application development and optimization productivity
  - Programmability, reusability, portability, robustness
  - Performance technology part of larger programming system
- Parallel systems evolution will change process, technology, and use of tools

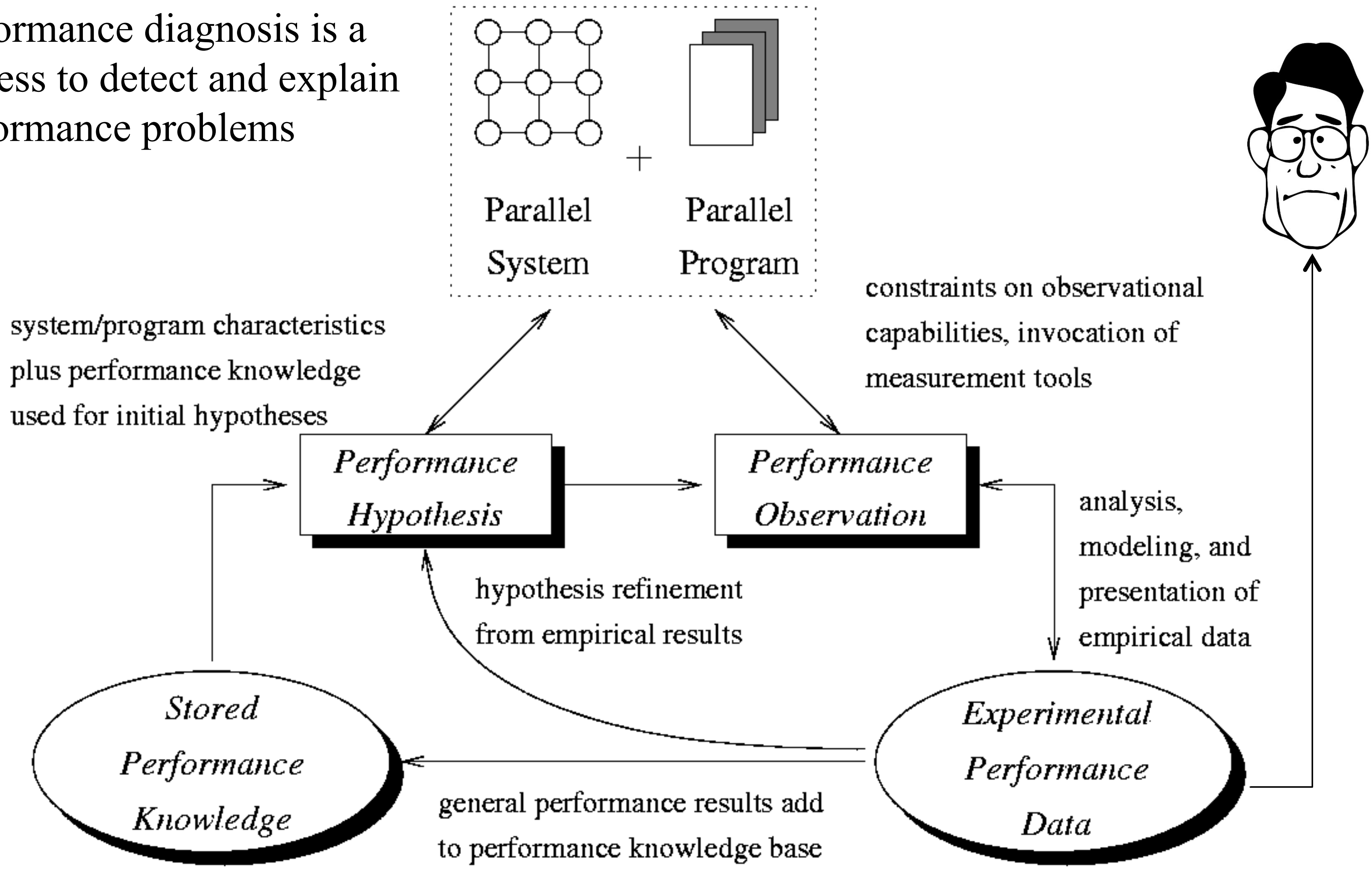
Retrospective (1991) – Performance Observability

- Performance evaluation problems define the requirements for performance measurement and analysis methods
- *Performance observability* is the ability to "accurately" capture, analyze, and present understand (collectively *observe*) information about parallel software and system
- Tools for performance observability must balance the *need* for performance data against the *cost* of obtaining it (environment complexity, performance intrusion)
  - Too little performance data makes analysis difficult
  - Too much data perturbs the measured system
- Important to understand performance observability complexity and develop technology to address it

UNIVERSITY OF OREGON

Retrospective (1998-2007) – Performance Diagnosis

Performance diagnosis is a process to detect and explain performance problems



UNIVERSITY OF OREGON

Performance Diagnosis Projects

- ***APART* – Automatic Performance Analysis - Real Tools**
  - Problem specification and identification
- ***Poirot*** – theory of performance diagnosis processes
  - Compare and analyze performance diagnosis systems
  - Use theory to create system that is automated / adaptable
    - *Heuristic classification:* match to characteristics
    - *Heuristic search:* look up solutions with problem knowledge
  - Problems: low-level feedback, lack of explanation power
- ***Hercule*** – knowledge-based (model-based) diagnosis
  - Capture knowledge about performance problems
  - Capture knowledge about how to detect and explain them
  - Knowledge comes from *parallel computational models*
    - associate computational models with performance models

Retrospective (2008-2012) – Performance Complexity

- Performance tools have evolved incrementally to serve the dominant architectures and programming models
  - Reasonably stable, static parallel execution models
  - Allowed application-level observation focus
- Observation by ***first person*** measurement model:
  - Performance measurement can be made locally (per thread)
  - Performance data collected at the end of the execution
  - Post-mortem analysis and presentation of performance results
  - Offline performance engineering
- Increasing performance complexity
  - Factors: core counts, hierarchical memory architecture, interconnection technology, heterogeneity, and scale
- Focus on performance technology integration

Evolution

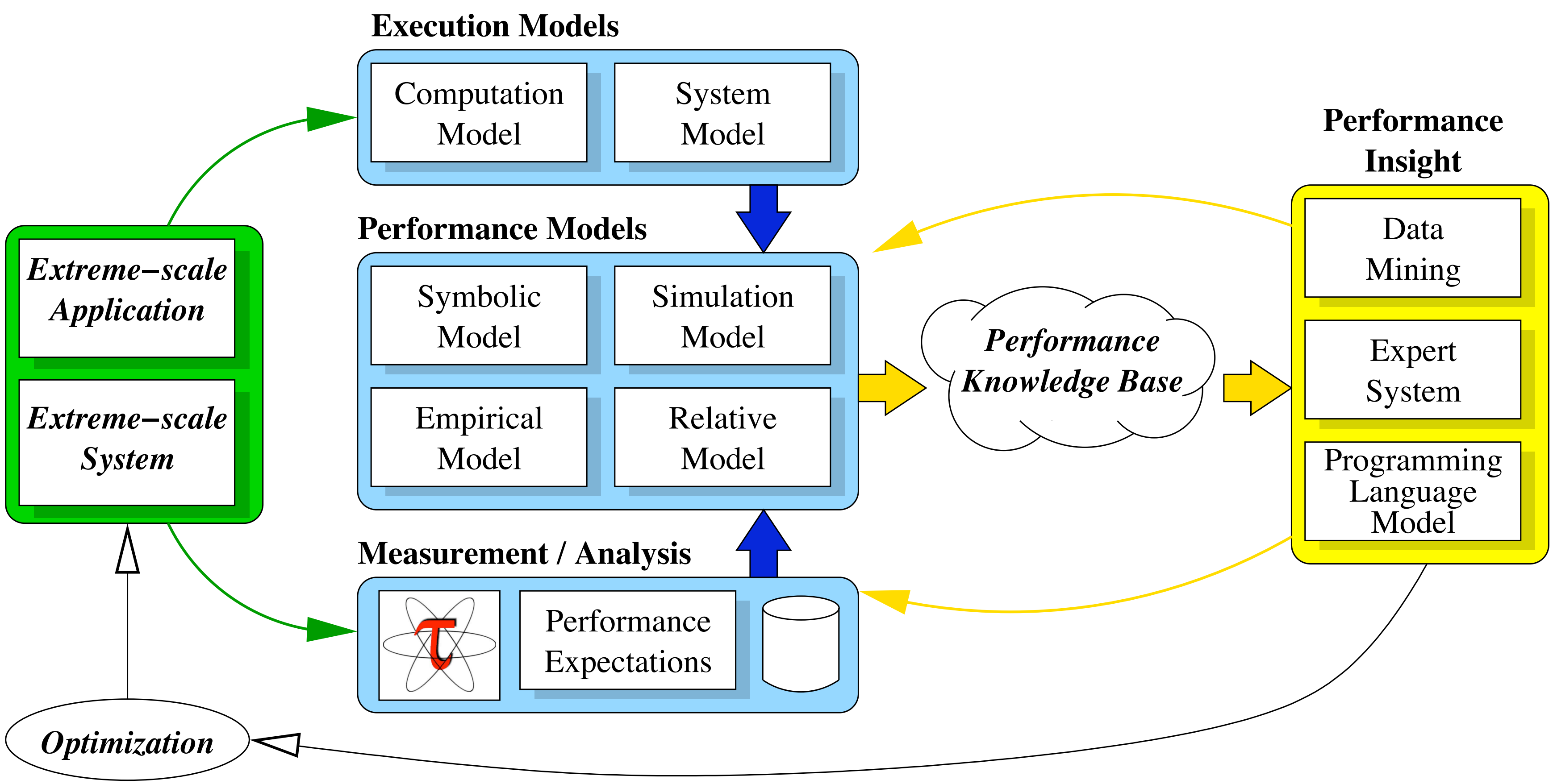
- Increased performance complexity and scale forces the engineering process to be more intelligent and automated
  - Automate performance data analysis / mining / learning
  - Automated performance problem identification
- Even with intelligent and application-specific tools, the decisions of what to analyze are difficult
  - Performance engineering tools and practice must incorporate a performance knowledge discovery process
- Model-oriented knowledge
  - Computational semantics of the application
  - Symbolic models for algorithms
  - Performance models for system architectures / components
- Application developers can be more directly involved in the performance engineering process

Need for Whole Performance Evaluation

- Extreme scale performance is an optimized orchestration
  - Application, processor, memory, network, I/O
- Reductionist approaches to performance will be unable to support optimization and productivity objectives
- Application-level only performance view is myopic
  - Interplay of hardware, software, and system components
  - Ultimately determines how performance is delivered
- Performance should be evaluated *in toto*
  - Application and system components
  - Understand effects of performance interactions
  - Identify opportunities for optimization across levels
- Need *whole performance evaluation practice*

"Extreme" Performance Engineering

- Empirical performance data evaluated with respect to performance expectations at levels of abstraction



UNIVERSITY OF OREGON

Revolution (for Exascale)

- First person measurement (post-mortem analysis) not viable
  - Highly concurrent and dynamic execution model
    - post-mortem analysis of low-level data prohibitive
  - Interactions with scarce and shared resources
    - introduces bottlenecks and queues on chip/node and between nodes
  - Multiple objectives (performance, energy, resilience, …)
  - Dynamic variability motivates need for runtime adaptation
- ***Third person*** measurement model required (in addition)
  - Focus is on system activity characterization at different levels
    - system resource usage is a primary concern
  - Measurements are analyzed relative to contributors
  - Online analysis and availability of performance allows introspective adaptation for objective evaluation and tuning

A New “Performance” Observability

- Exascale requires a fundamentally different “performance” observability paradigm
  - Designed specifically to support introspective adaptation
  - Reflects computation model mapped to execution model
  - Aware of multiple objectives (“performance”)
  - In-situ analysis of performance state and objectives
- Key parallel “performance” abstraction
  - Inherent state of exascale execution is dynamic
  - Embodies non-stationarity of “performance”
  - Constantly shaped by the adaptation of resources to meet computational needs and optimize execution objectives

UNIVERSITY OF OREGON

Needs Integration in Exascale Software Stack

- Exascale programming methodology can include observability awareness and adaptability
  - Programming system exposes alternatives
    - parameters, algorithms, parallelism control, …
  - Runtime state awareness can be coupled with application knowledge for self-adaptive, closed-loop runtime tuning
    - richer contextualization and attribution of performance state
- Enables *top-down* application transformations to be optimized for runtime and system layers
- Feeding back dynamic information about HW/SW software resources from *bottom-up*
- Performance introspection and dynamic adaptivity
- Requires support in OS and runtime environment

Willing Suspension of (Observability) Disbelief

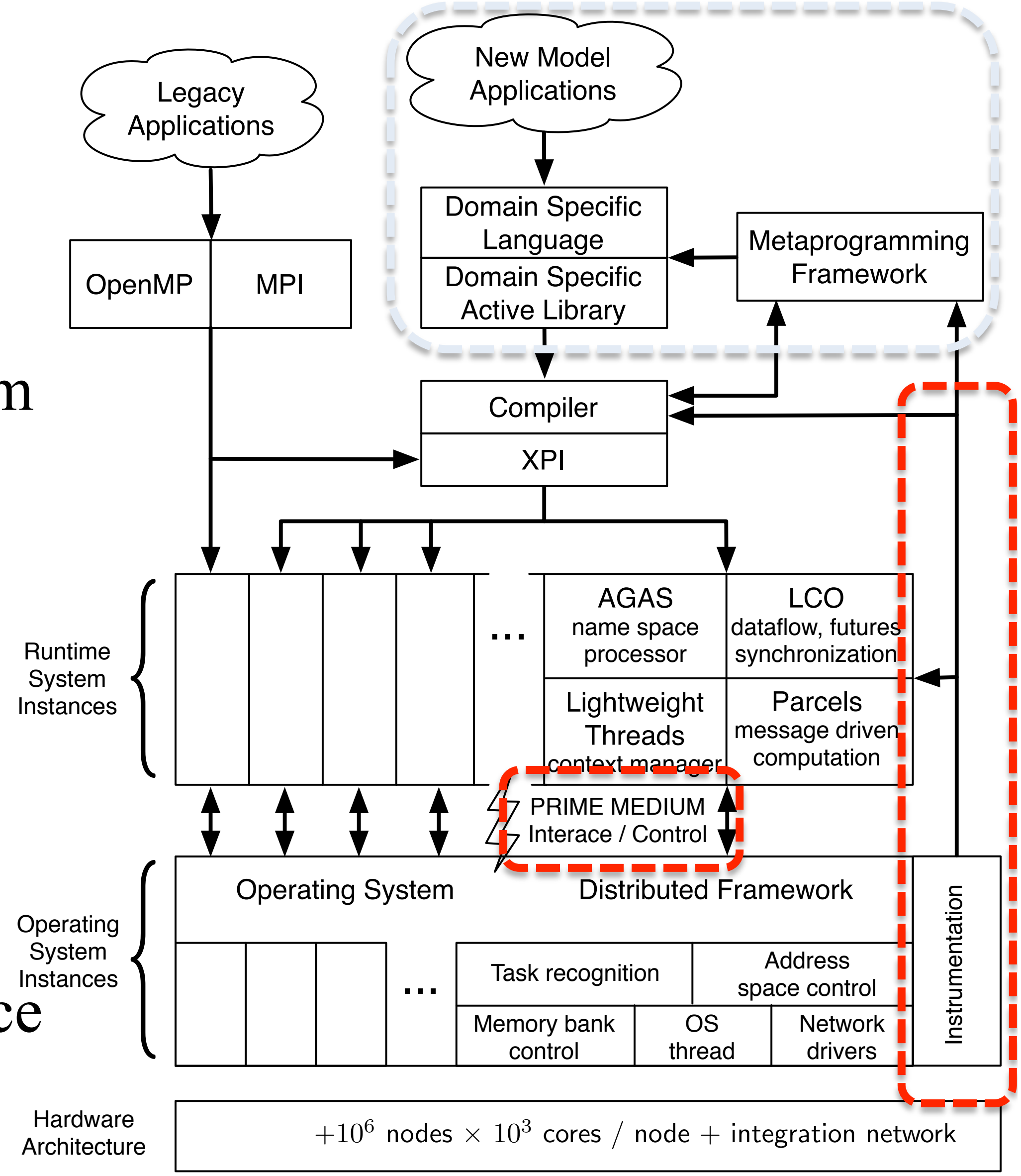
- Suppose that full knowledge of the state of the exascale system were available
  - Can observe it at any time
  - No cost to produce it, access it, or analyze it
  - No cost to provide it to wherever it might be used
- How/where could it be used in an exascale system?
- Would it fundamentally change how an exascale system is programmed?
- How would you incorporate such a capability in the exascale software stack?

XPRESS Project (DOE X-Stack)

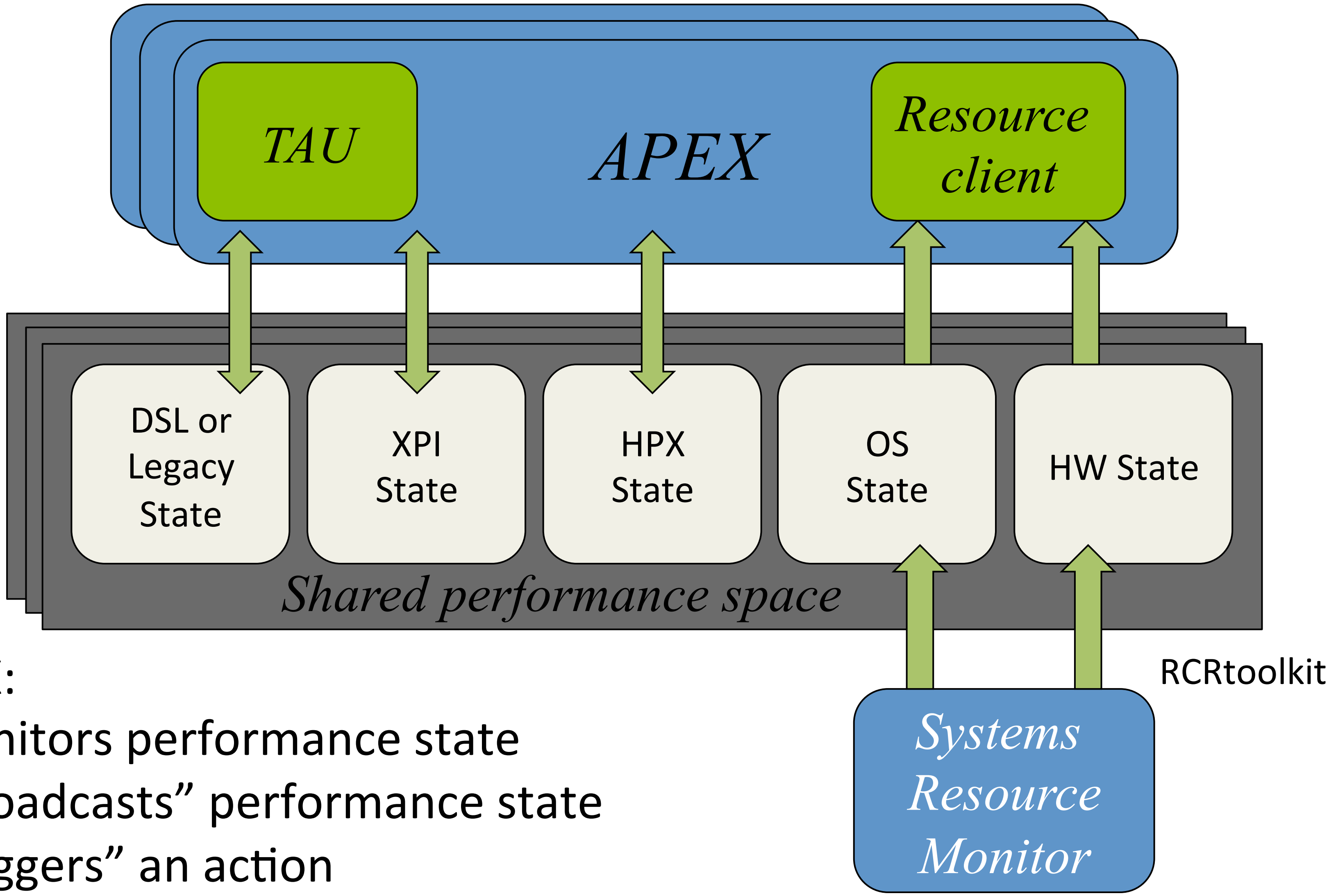
- Design and development of exascale software stack to support the ParalleX execution model
  - Highly concurrent
  - Asynchonous
  - Message driven
  - Global address space
- OpenX
  - XPI programming API
  - HPX runtime system
  - RIOS interface to OS
  - APEX performance system
- Team
  - Universities: IU, LSU, UH, UNC/RENCI, UO
  - Laboratories: SNL, LBNL, ORNL

Integrated Software Stack for ParalleX

- OpenX
  - XPI programming API
  - HPX runtime system
  - RIOS interface to OS
  - APEX performance system
- APEX
  - OS (LXK) tracks system-level resources
  - Runtime (HPX) tracks threads, queues, parcels, remote ops, memory, concurrency
  - XPI allows, allow language-level performance semantics to be measured



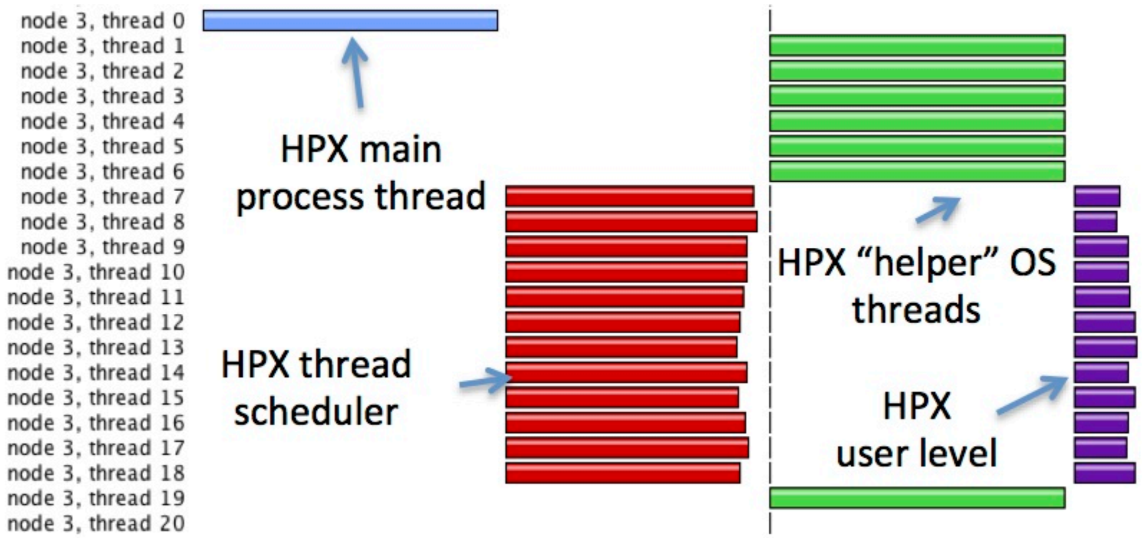
APEX Prototype Approach

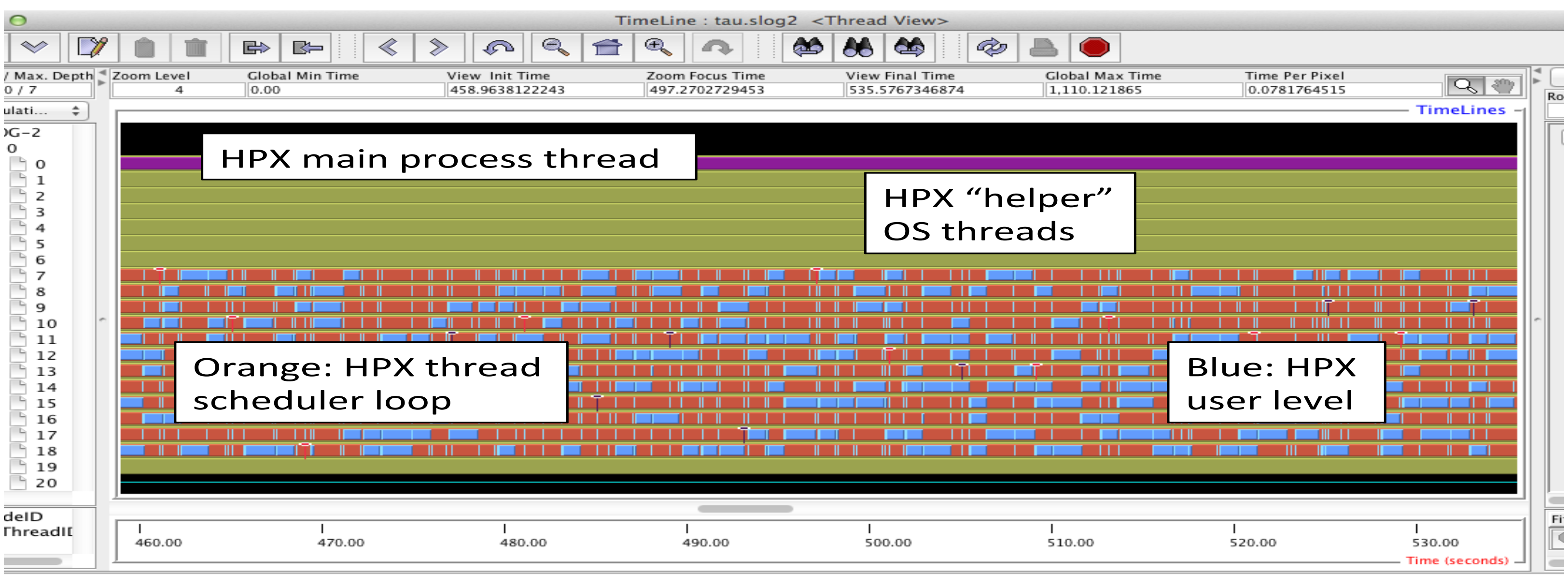


APEX:

- monitors performance state
- "broadcasts" performance state
- "triggers" an action

APEX Prototyping with HPX-3 and TAU





UNIVERSITY OF OREGON

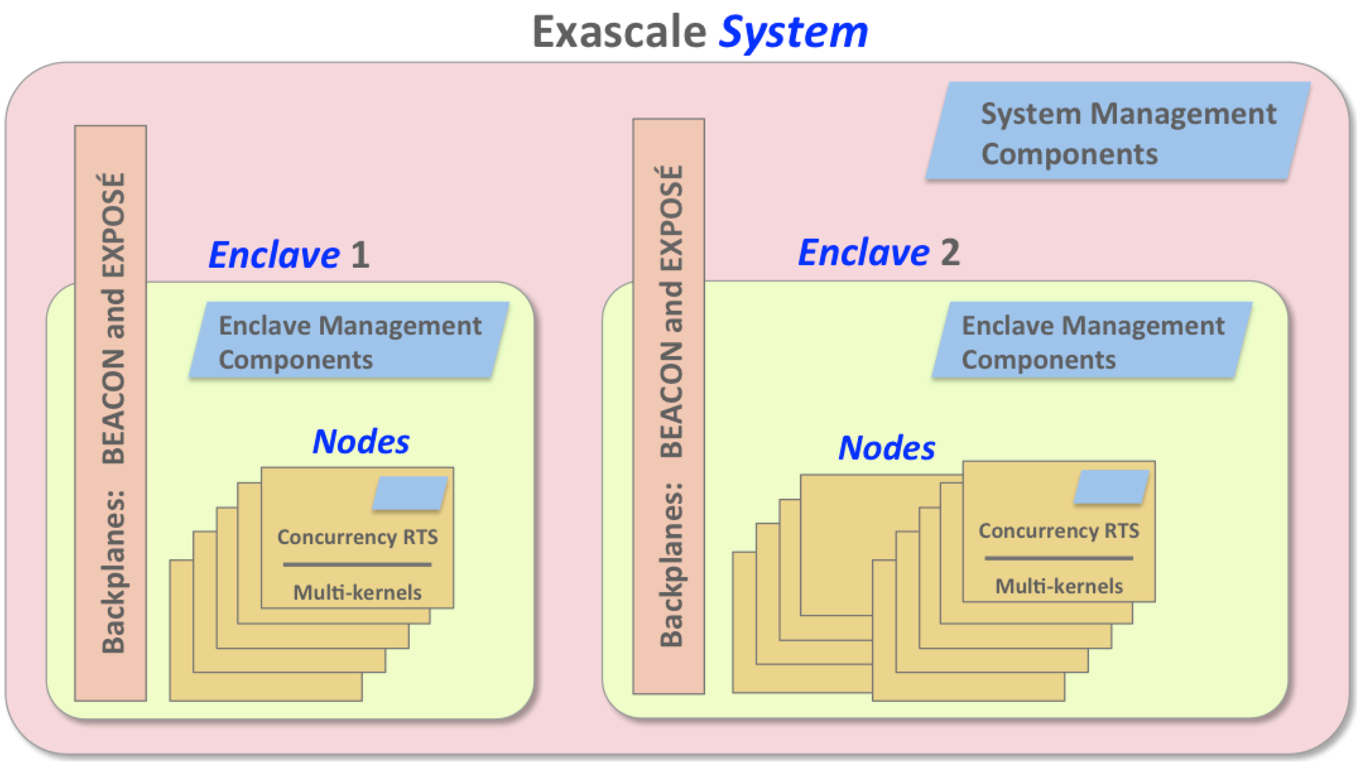
Argo DOE ExaOSR Project

- Exascale OS and runtime research project
- Team
  - Labs: ANL, LLNL, PNL
  - Universities: BU, UC, UIUC, UO, UTK
- Philosophy
  - Whole-system view
    - dynamic user environment (functionality, dynamism, flexibility)
    - first-class managed resources (performance, power, …)
    - hierarchical response to faults
  - Massive concurrency support
- Key ideas relevant to ESPT
  - Hierarchical (control, communication, goals, data resolution)
  - Embedded (performance, power, …) feedback and response
  - Global system support

UNIVERSITY OF OREGON

Argo Exascale System View

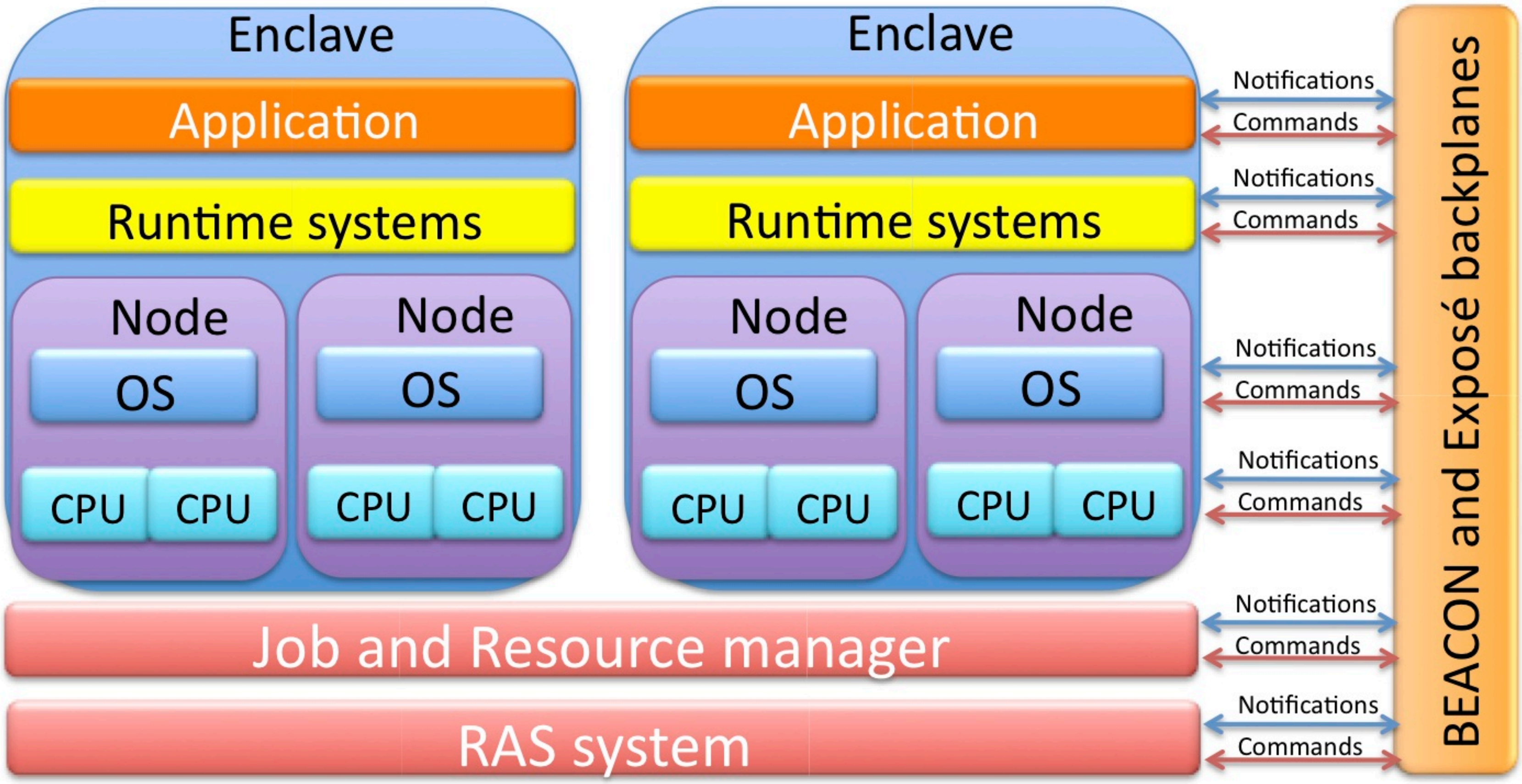
- Node OSR
- Lightweight runtime for concurrency
- Event, control, and performance backplane
- Global optimization / global view



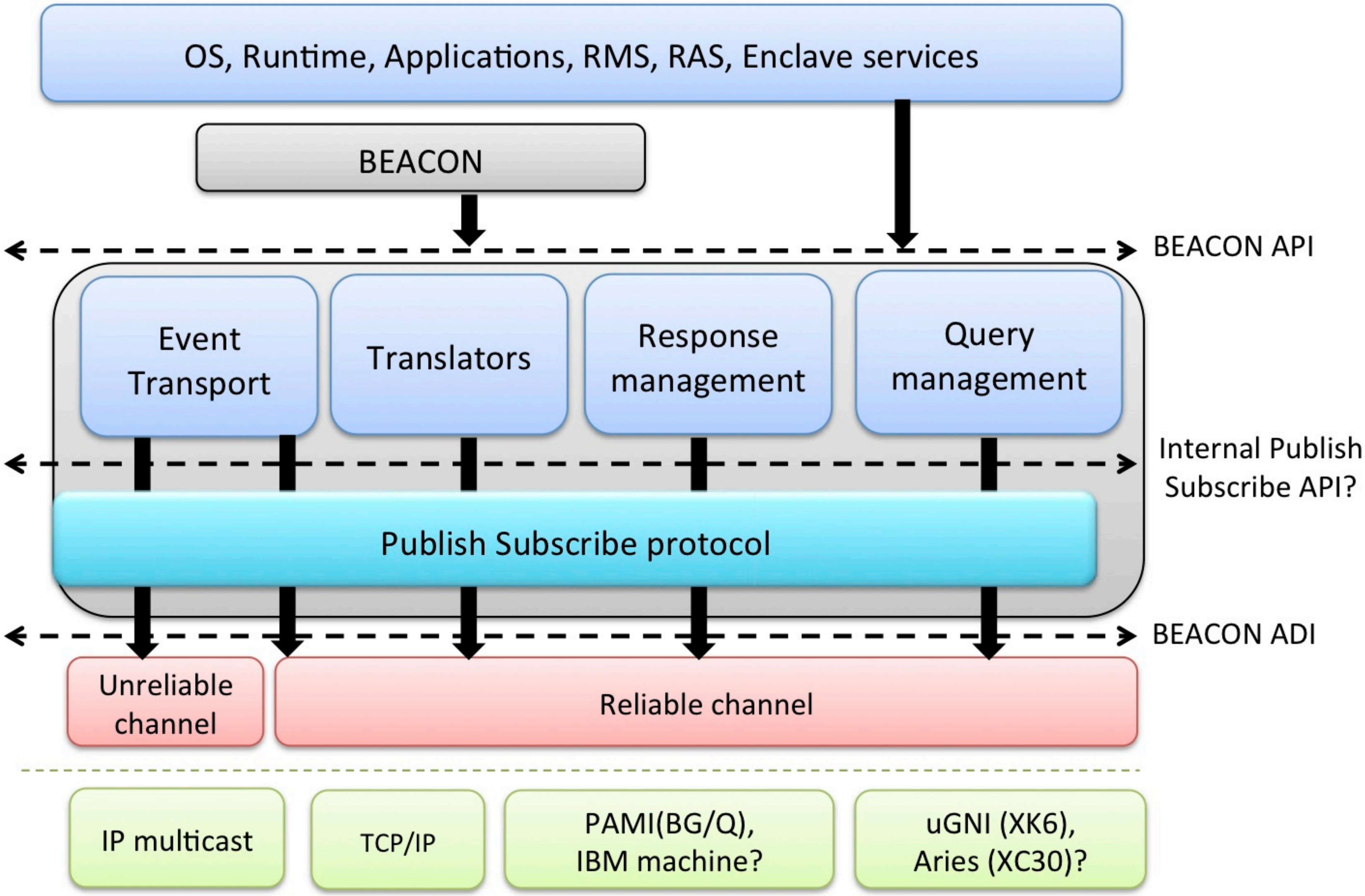
UNIVERSITY OF OREGON

Argo Global Information Backplane

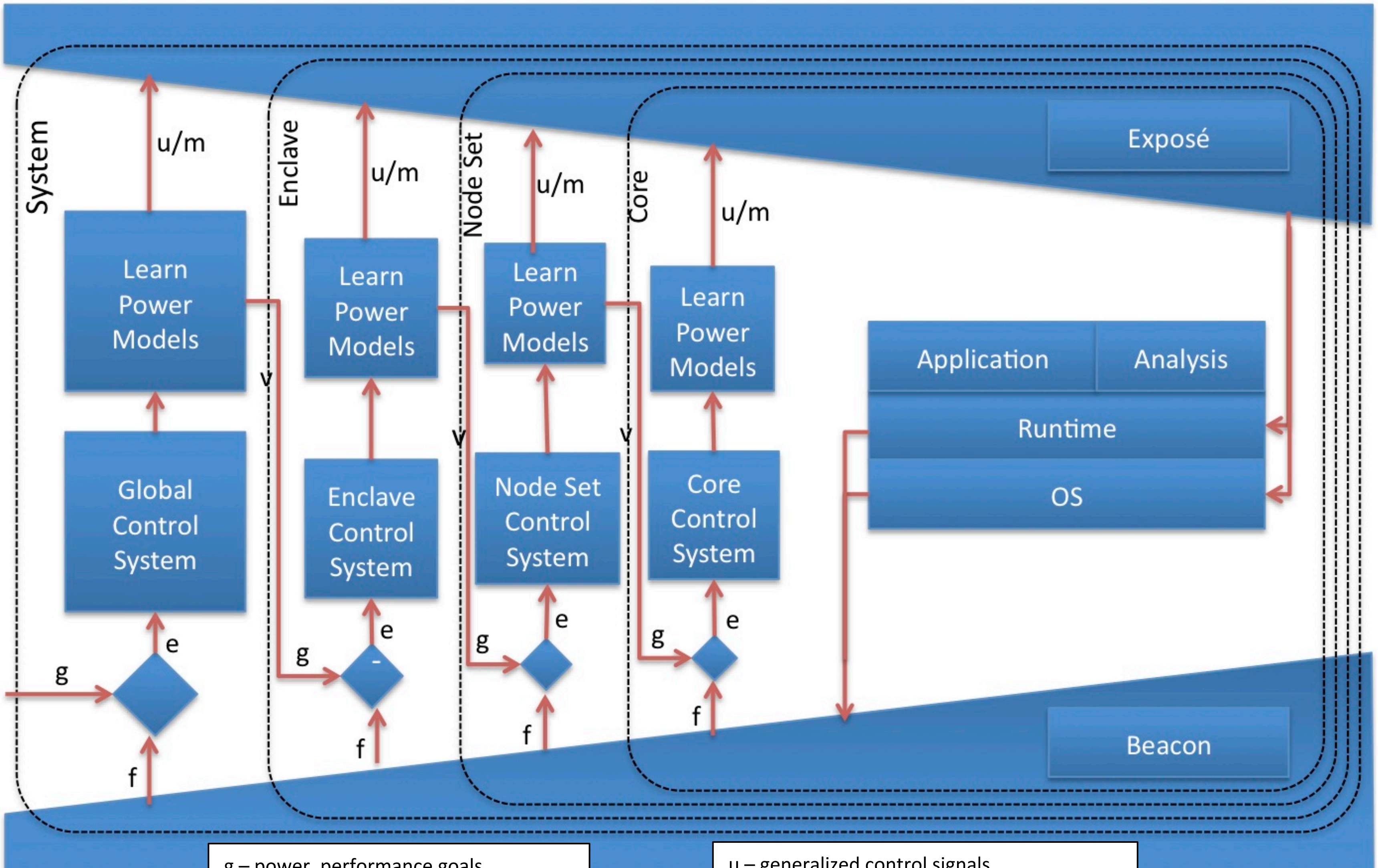
- BEACON: event/action/control notification
- Exposé: performance observability system



BEACON Architecture



Global Optimization View



g – power, performance goals
f – feedback from BEACON and EXPOSÉ
e – error between goals and feedback

u – generalized control signals
v – settings for next level
m – power/resilience models & mechanisms

Research Projects

- *PRIMA-X* (DOE ASCR, UO, Research Centre Juelich, Aachen) E (R)
  - Refactoring core performance measurement infrastructure
- *Vancouver* (DOE X-Stack, ORNL, UO, GT, UIUC) (in renewal) E
  - Heterogeneous performance measurement (with GPUs)
- *SUPER* (DOE SciDAC Institute, many partners) E
  - Performance tools integration for end-to-end analysis, autotuning, energy, and resilience
- *GlassBox* (NSF OCI SI^2 program, GT, UH, UO) E (R)
  - Open interactions in HPC toolchain (compiler, runtime, I/O)
  - Integrated performance observation and knowledge sharing
- *XPRESS* (DOE X-Stack, SNL, IU, UNC, UH, UO, LSU) R
  - Integrated software ecosystem for ParalleX
  - Online, adaptive optimization and control
- Argo (DOE ExaOSR, ANL, PNNL, LLNL, UO, UC, UIUC) R
  - OS and runtime technology for exascale
  - Scalable signaling, introspection, feedback

Evolution
Revolution

UNIVERSITY OF OREGON

Performance as Collective Behavior

- New performance observability paradigm can be thought of as providing *collective awareness* for exascale
- It allows the exascale system to be *performance-aware* and *performance-reactive*, able to observe performance state wholistically and holistically and to couple it with application knowledge for self-adaptive, runtime control

- Exascale systems might be considered to then be functioning as an self-organizing performance collective whose behavior is a natural consequence of seeking highly efficient operational forms optimizing exascale objectives