HPC Resource Management: View to the Future

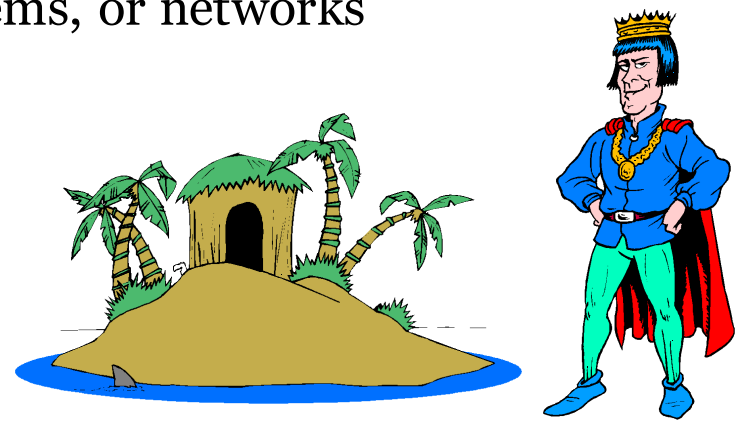
Ralph H. Castain
Jan 27, 2016

Definition: RM

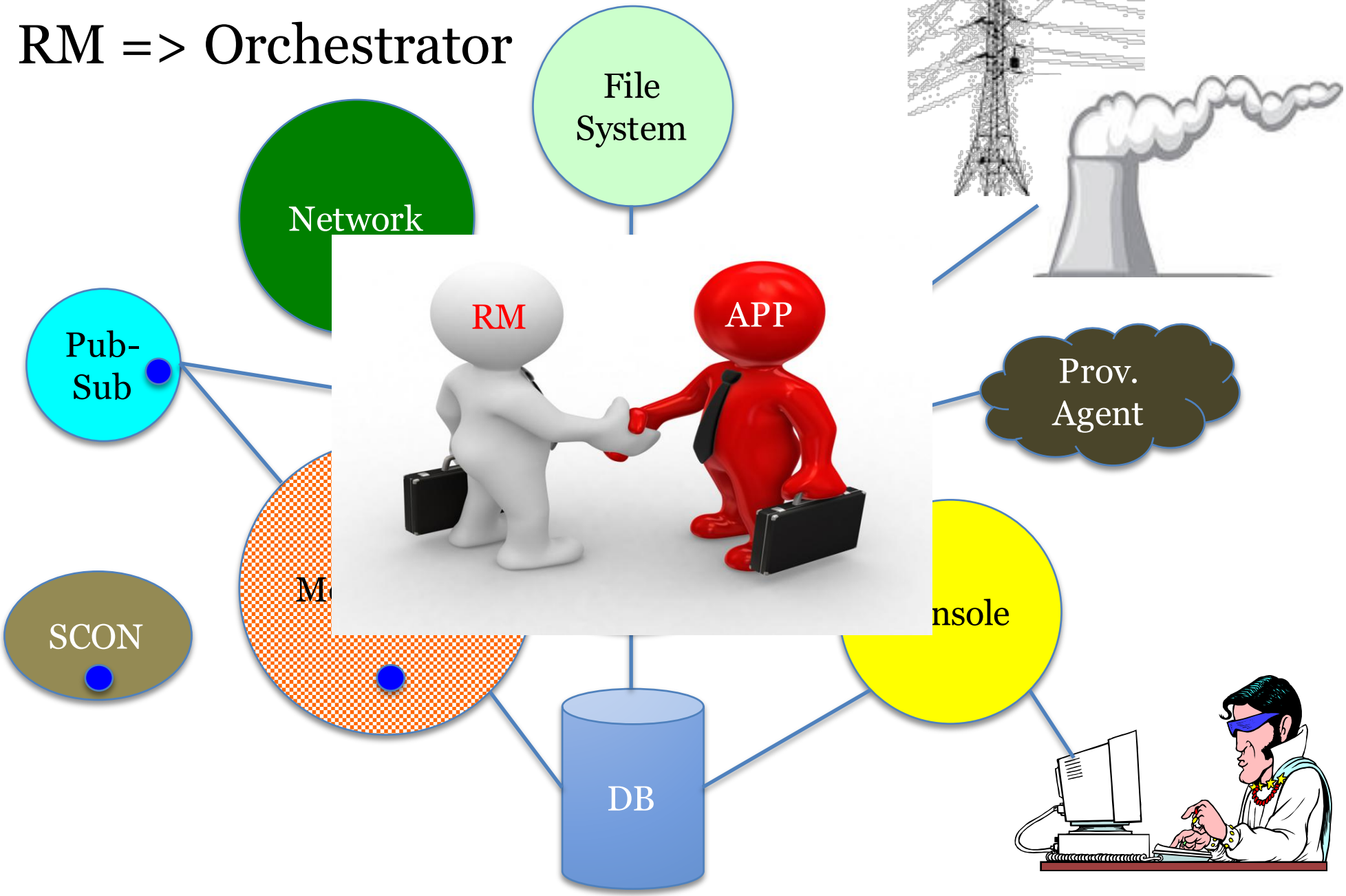
- Scheduler/Workload Manager (WLM)
  - Allocates resources to session
  - Interactive and batch
- Run-Time Environment
  - Launch and monitor applications
  - Support inter-process communication wireup
  - Serve as intermediary between applications and WLM
    - Dynamic resource requests
    - Error notification
  - Implement failure policies

Current State

- Provided as complete package (WLM+RTE)
  - Sometimes offer hook to replace one
  - Mostly proprietary
  - Open source often GPL (integration difficult)
- Programming model specific, static
- Independent island
  - Not integrated with monitoring, file systems, or networks
- Limited fault tolerance support
  - Restart failed job



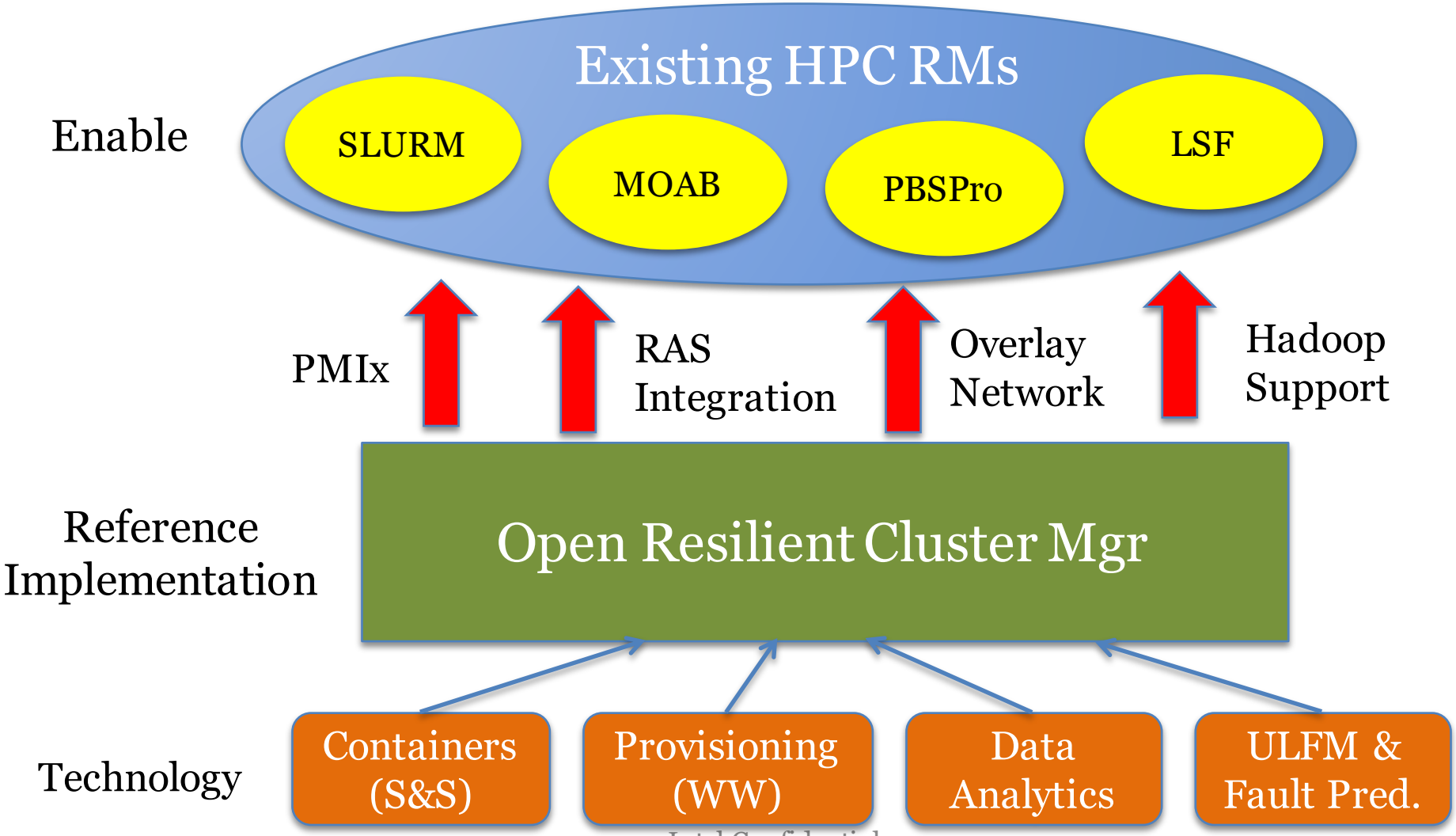
RM => Orchestrator



Resource Management Emerging Needs

- Emerging Application Needs
  - Dynamic resource management
  - Application involvement in decisions, including fault notification
  - Workflow orchestration
- Emerging System Needs
  - Scalable operations
  - Cross-subsystem integration (RM, RAS, site utilities, …)
  - Data staging, Burst Buffers, persistent memory management
  - System notification requests, topology, QoS
  - Power management

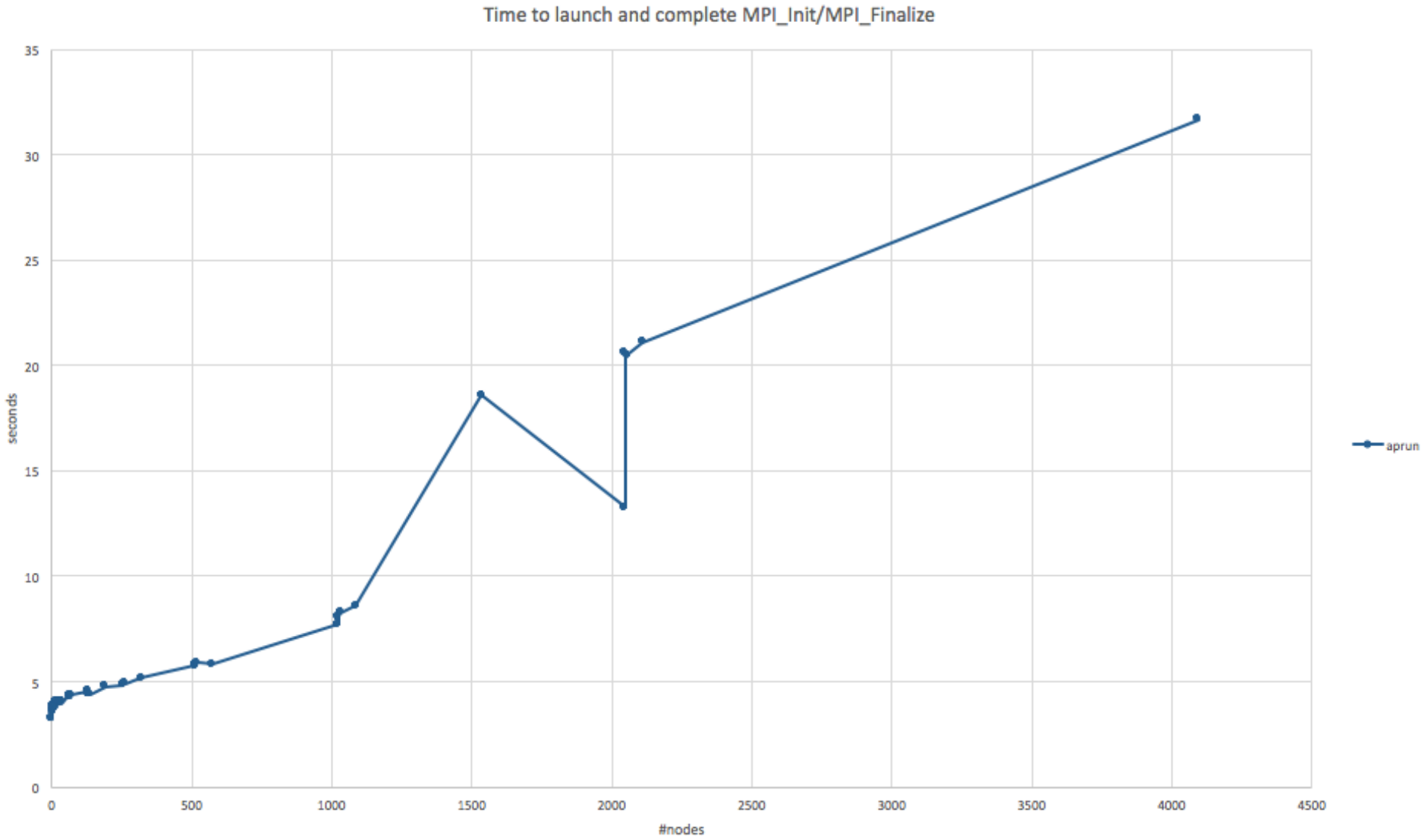
Multi-Tiered Strategy

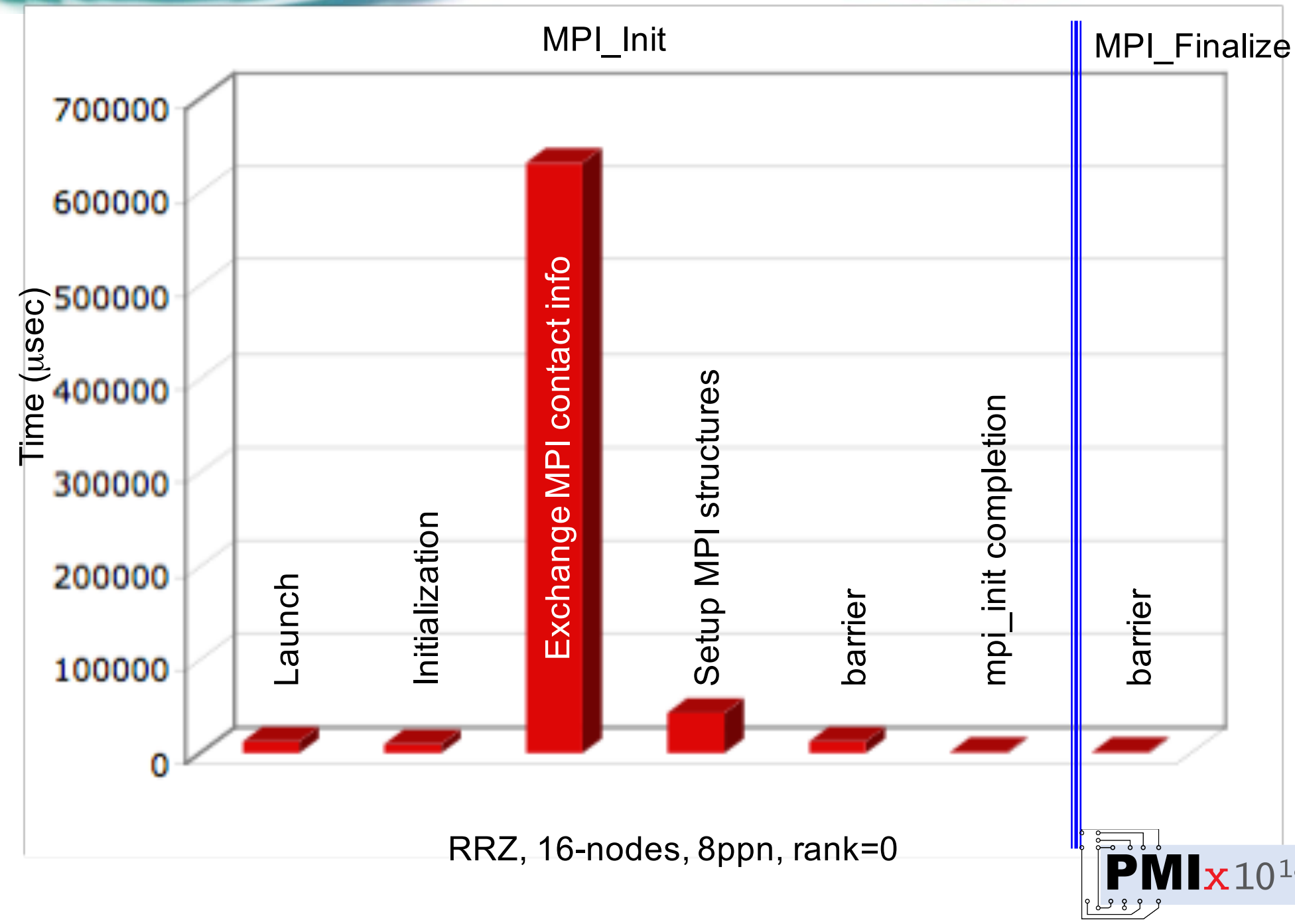


Reference Implementation: ORCM

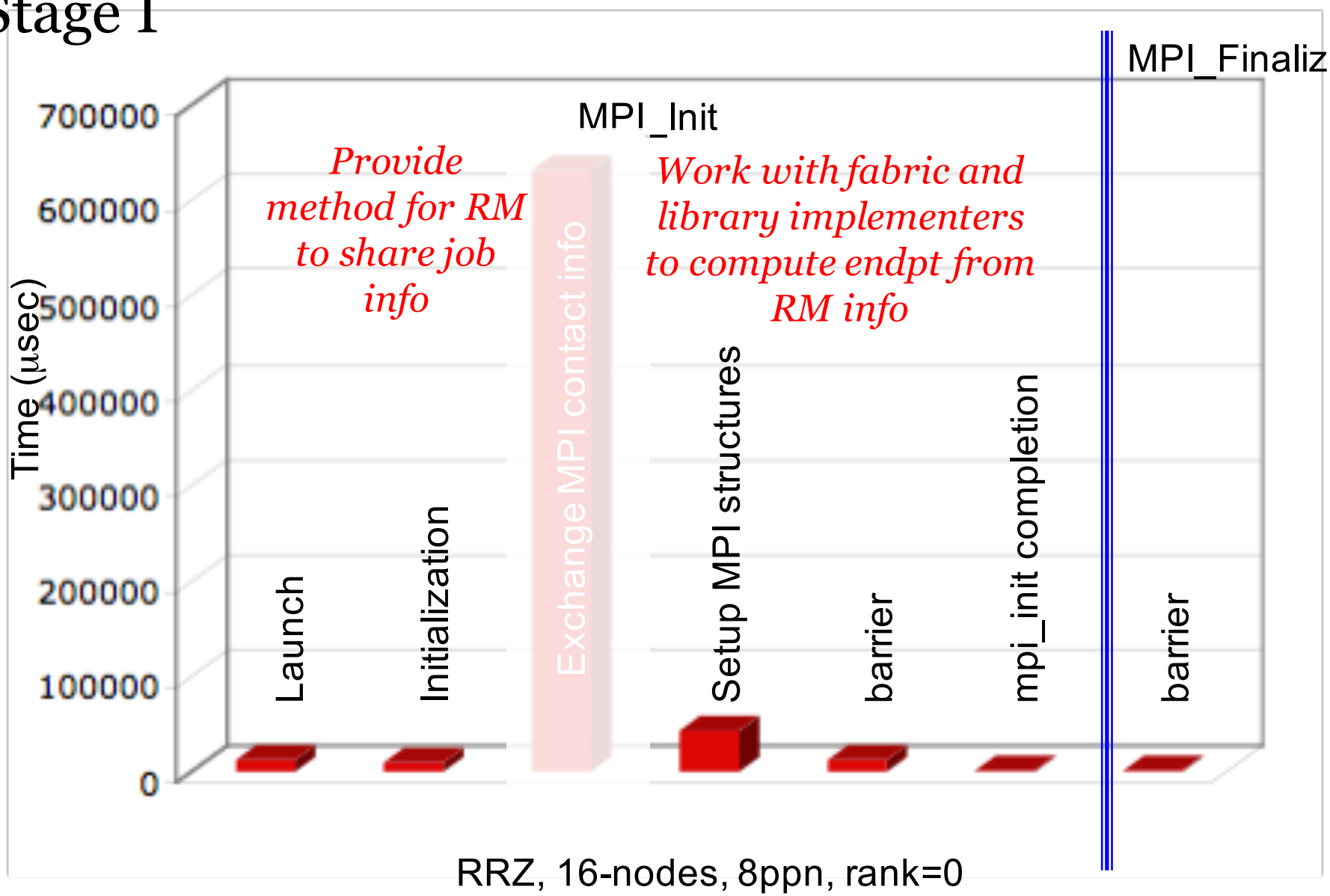
- Scalable to exascale levels & beyond
  - Better-than-linear scaling
  - Constrained memory footprint
- Dynamically configurable
  - Sense and adapt, user-directable
  - On-the-fly updates
  - Fully utilize underlying hw
- Open source (non-GPL)
  - Support proprietary add-ons
- Maintainable, flexible
  - Plugin architecture
  - Easy extension for R&D
- Resilient
  - Self-heal around failures
  - Reintegrate recovered resources

Launch Scaling: Core Capability

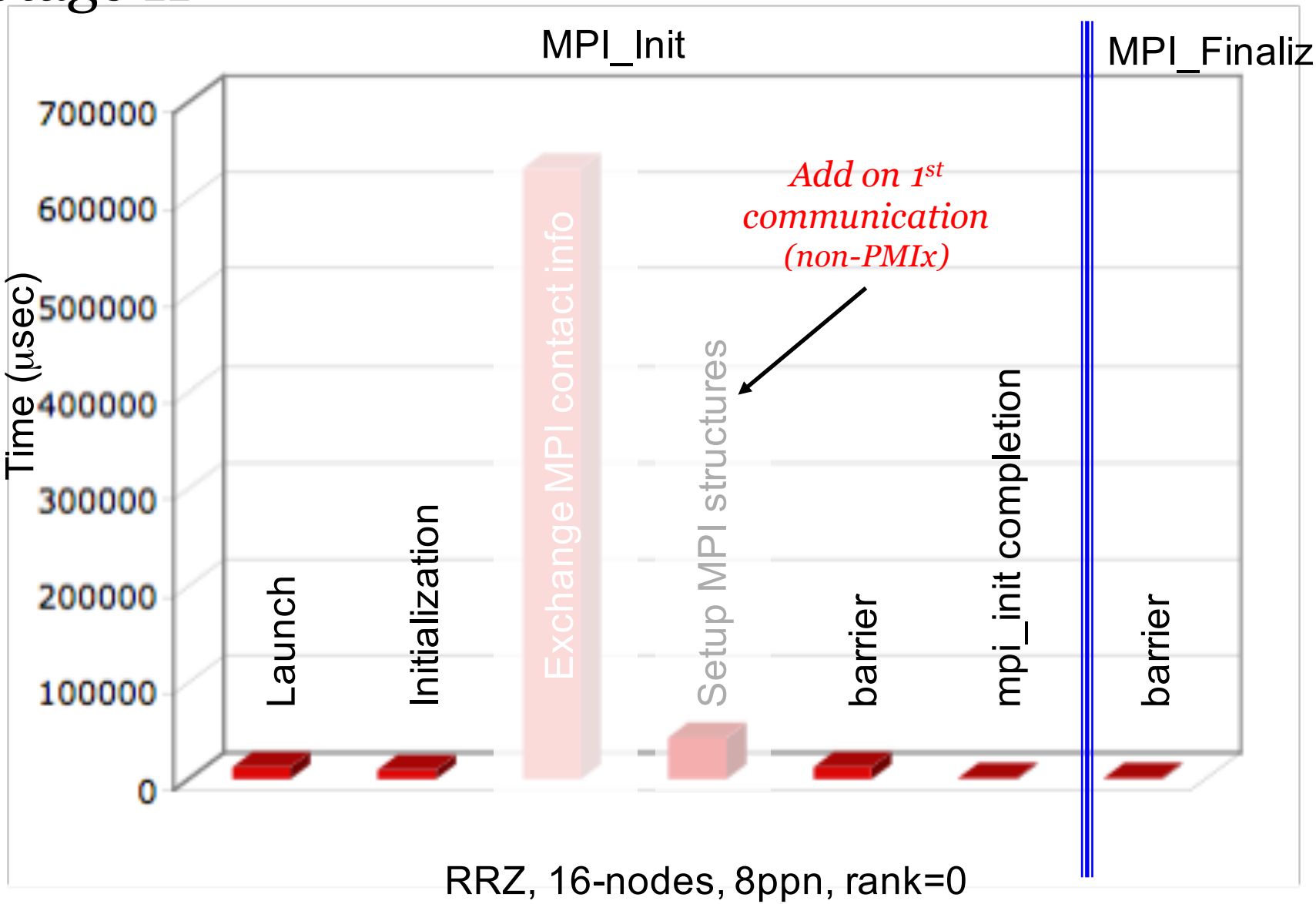



MPI_Init
MPI_Finalize
Time (µsec)
700000
600000
500000
400000
300000
200000
100000
0
Launch
Initialization
Exchange MPI contact info
Setup MPI structures
barrier
mpi_init completion
barrier
RRZ, 16-nodes, 8ppn, rank=0

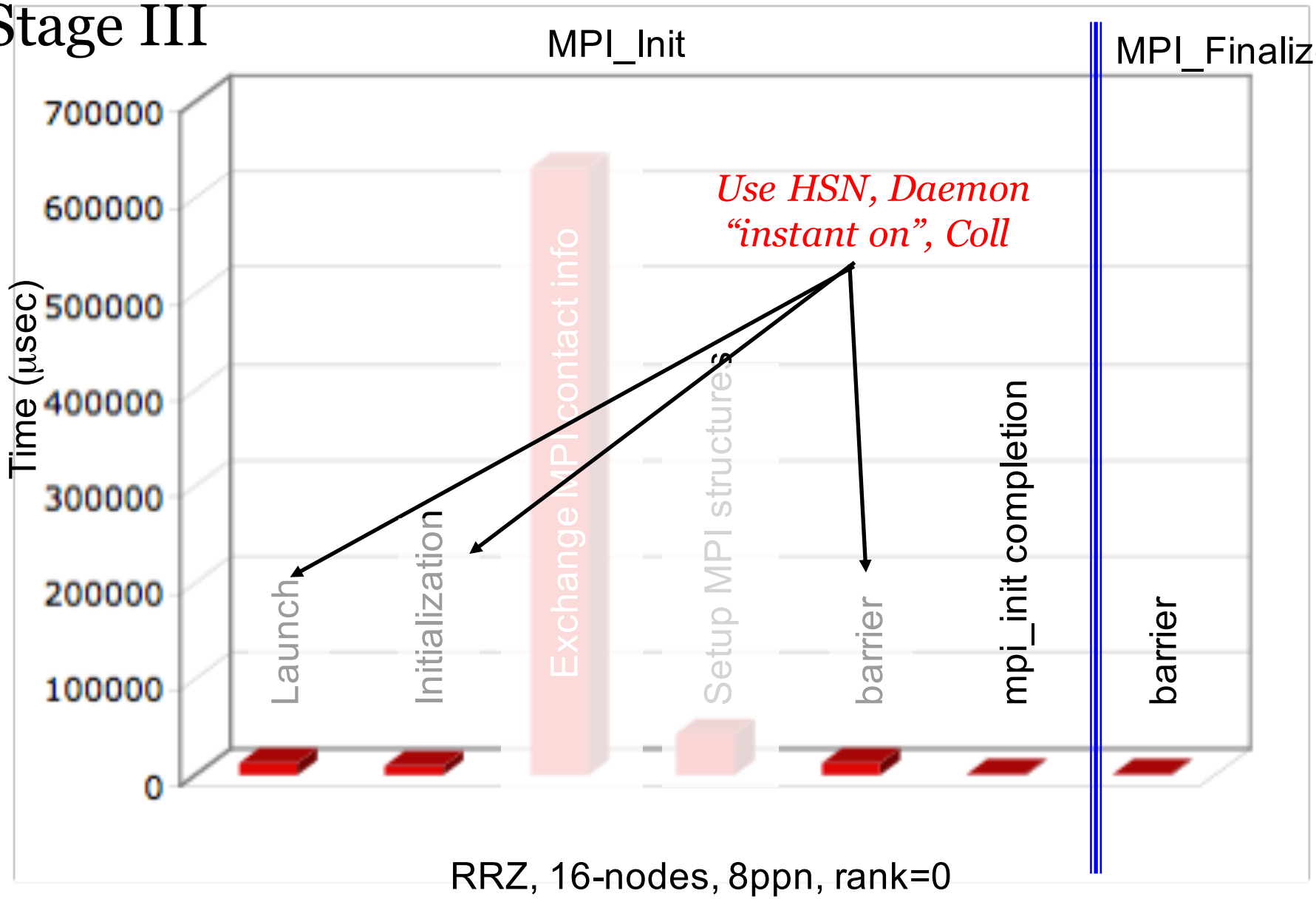
Stage I


MPI_Finalize
MPI_Init
Provide method for RM to share job info
Exchange MPI contact info
Work with fabric and library implementers to compute endpt from RM info
Time (µsec)
700000
600000
500000
400000
300000
200000
100000
0
Launch
Initialization
Setup MPI structures
barrier
mpi_init completion
barrier
RRZ, 16-nodes, 8ppn, rank=0

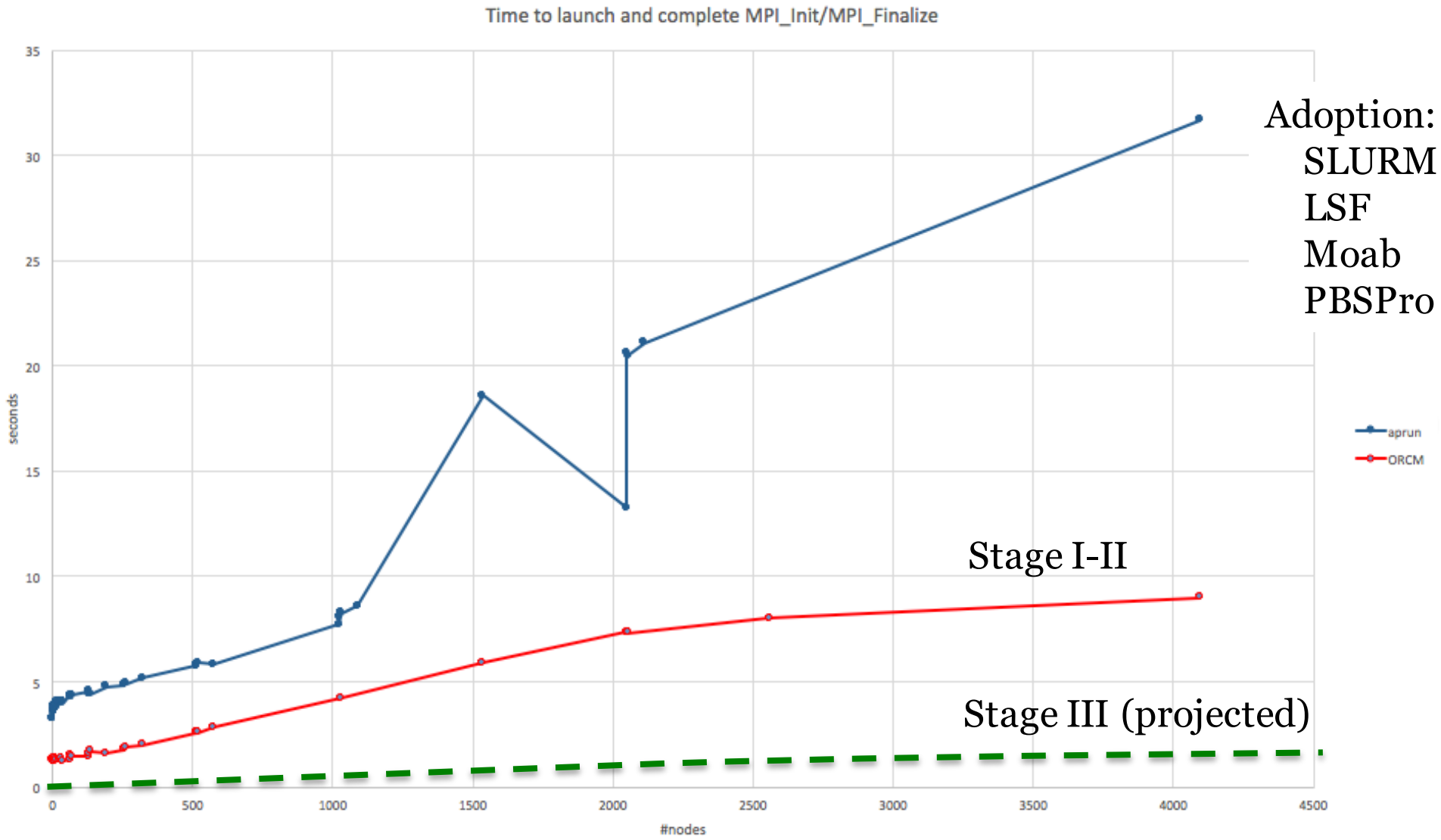
Stage II


MPI_Init
MPI_Finalize
Time (µsec)
700000
600000
500000
400000
300000
200000
100000
0
Launch
Initialization
Exchange MPI contact info
Setup MPI structures
barrier
mpi_init completion
barrier
Add on 1st communication (non-PMIx)
RRZ, 16-nodes, 8ppn, rank=0

Stage III


MPI_Init
MPI_Finalize
Time (µsec)
700000
600000
500000
400000
300000
200000
100000
0
Use HSN, Daemon "instant on", Coll
Launch
Initialization
Exchange MPI contact info
Setup MPI structures
barrier
mpi_init completion
barrier
RRZ, 16-nodes, 8ppn, rank=0

Current Status



Adoption:
- SLURM
- LSF
- Moab
- PBSPro

Flexible Allocation Support

- Request additional resources
  - Compute, memory, network, NVM, burst buffer
  - Immediate, forecast
  - Expand existing allocation, separate allocation
- Return extra resources
  - No longer required
  - Will not be used for some specified time, reclaim (handshake) when ready to use
- Notification of preemption
  - Provide opportunity to cleanly pause

I/O Support

- Asynchronous operations
  - Anticipatory data fetch, staging
  - Advise time to complete
  - Notify upon available
- Storage policy requests
  - Hot/warm/cold data movement
  - Desired locations and striping/replication patterns
  - Persistence of files, shared memory regions across jobs, sessions
  - ACL to generated data across jobs, sessions

Spawn Support

- Staging support
  - Files, libraries required by new apps
  - Allow RM to consider in scheduler
    - Current location of data
    - Time to retrieve and position
    - Schedule scalable preload
    - Specified topology, min/max procs, …
- Provisioning requests
  - Allow RM to consider in selecting resources, minimize startup time due to provisioning
  - Desired image, packages

Network Integration

- Quality of service requests
  - Bandwidth, traffic priority, power constraints
  - Multi-fabric failover, striping prioritization
  - Security requirements
    - Network domain definitions, ACLs
- Notification requests
  - State-of-health
  - Update process endpoint upon fault recovery
- Topology information
  - Torus, dragonfly, …

Power Control/Management

- Scheduler analytics
  - Predict power usage, advise updates
- Application requests
  - Advise of changing workload requirements
  - Request changes in policy
  - Specify desired policy for spawned applications
- RM notifications
  - Need to change power policy
  - Preemption notification
    - Allow application to accept, request pause

Fault Tolerance

- Notification
  - App can register for error notifications, incipient faults
    - RM-app negotiate to determine response
  - App can notify RM of errors
    - RM will notify specified, registered procs
- Rapid application-driven checkpoint
  - Local on-node NVRAM, auto-stripe checkpoints
  - Policy-driven “bleed” to remote burst buffers and/or global file system
- Restart support
  - Specify source (remote NVM checkpoint, global filesystem, etc)
  - Location hints/requests
  - Entire job, specific processes

Workflow Orchestration

- Growing range of workflow patterns
  - Traditional HPC: bulk synchronous, single application
  - Analytics: asynchronous, staged
  - Parametric: large number of simultaneous independent jobs
- Ability to dynamically spawn a job
  - Need flexibility – min/max size, rolling allocation, …
  - Events between jobs and cross-linking of I/O
  - Queuing of spawn requests
- Tool interfaces that work in these environments

Workload Manager: Job Description Language

- Complexity of describing job is growing
  - Power, file/lib positioning
  - Performance vs capacity, programming model
  - System, project, application-level defaults
- Provide templates?
  - System defaults, with modifiers
    - --hadoop:mapper=foo,reducer=bar
  - User-defined
    - Application templates
    - Shared, group templates
  - Markup language definition of behaviors, priorities

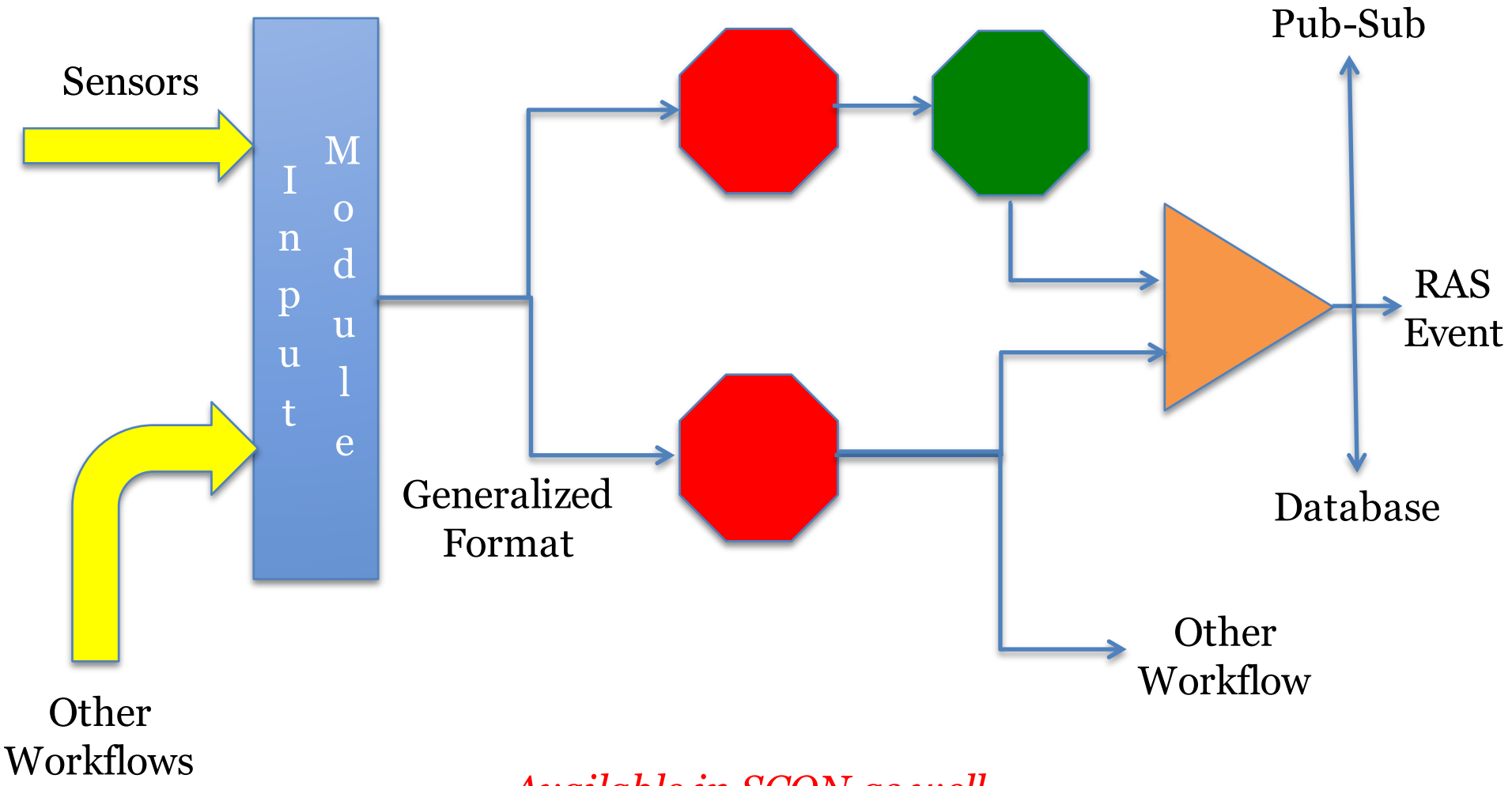
Flexible Architecture

- Each tool built on top of same plugin system
  - Different combinations of frameworks
  - Different plugins activated to play different roles
  - Example: orcmd on compute node vs on rack/row controllers
- Designed for distributed, centralized, hybrid operations
  - Centralized for small clusters
  - Hybrid for larger clusters
  - Example: centralized scheduler, distributed "worker-bees"
- Accessible to users for interacting with RM
  - Add shim libraries (abstract, public APIs) to access framework APIs
  - Examples: SCON, pub-sub, in-flight analytics

Breaking it Down

- Workload Manager
  - Dedicated framework
  - Plugins for two-way integration to external WM (Moab, Cobalt)
  - Plugins for implementing internal WM (FIFO)
- Run-Time Environment
  - Broken down into functional blocks, each with own framework
    - Loosely divided into three general categories
    - Messaging, launch, error handling
    - One or more frameworks for each category
  - Knitted together via “state machine”
    - Event-driven, async
    - Each functional block can be separate thread
    - Each plugin within each block can be separate thread(s)

Analytics Workflow Concept



Available in SCON as well

Workflow Elements

- Average (window, running, etc.)
- Rate (convert incoming data to events/sec)
- Threshold (high, low)
- Filter
  - Selects input values based on provided params
- RAS event
  - Generates a RAS event corresponding to input description
- Publish data

Analytics

- Execute on aggregator nodes for in-flight reduction
  - Sys admin defines, user can define (if permitted)
- Event-based state machine
  - Each workflow in own thread, own instance of each plugin
  - Branch and merge of workflow
  - Tap stream between workflow steps
  - Tap data streams (sensors, others)
- Event generation
  - Generate events/alarms
  - Specify data to be included (window)

Distributed Architecture

- Hierarchical, distributed approach for unlimited scalability
  - Utilize daemons on rack/row controllers
- Analysis done at each level of the hierarchy
  - Support rapid response to critical events
  - Distribute processing load
  - Minimize data movement
- RM's error manager framework controls response
  - Based on specified policies

Fault Diagnosis

- Identify root cause and location
  - Sometimes obvious – e.g., when direct measurement
  - Other times non-obvious
    - Multiple cascading impacts
    - Cause identified by multi-sensor correlations (indirect measurement)
    - Direct measurement yields early report of non-root cause
    - Example: power supply fails due to borderline cooling + high load
- Estimate severity
  - Safety issue, long-term damage, imminent failure
- Requires in-depth understanding of hardware

Fault Prediction: Methodology

- Exploit access to internals
  - Investigate optimal location, number of sensors
  - Embed intelligence, communications capability
- Integrate data from all available sources
  - Engineering design tests
  - Reliability life tests
  - Production qualification tests
- Utilize learning algorithms to improve performance
  - Both embedded, post process
  - Seed with expert knowledge

Fault Prediction: Outcomes

- Continuous update of mean-time-to-preventative-maintenance
  - Feed into projected downtime planning
  - Incorporate into scheduling algo
- Alarm reports for imminent failures
  - Notify impacted sessions/applications
  - Plan/execute preemptive actions
- Store predictions
  - Algorithm improvement

HPC Controls