ParTec
MODULAR SUPERCOMPUTING

ParaStation MODULO

ParaStation HEALTHCHECKER

IO-SEA – Training – Infrastructure Monitoring Tools

May 10th, 2023

Patrick Küven, Support Analyst

Stephan Krempel, HPC Software Engineer

IO-SEA

PARASTATION MODULO SOFTWARE SUITE

ParTec
MODULAR SUPERCOMPUTING

Enabling HPC

- ParTec is a strong HPC specialist for more than two decades
  - ParaStation research project: 1995 (Univ. of Karlsruhe, Germany)
  - ParTec founded as a spin-off in 1999
  - HPC full *service* provider since 2004
  - HPC full *systems* provider since 2021
- Pioneering the Modular Supercomputing Architecture (MSA) for >10 years
- ParaStation Modulo is extensively used in production environments
  - Serves as the basis for co-design and co-development
  - Also enables ParTec Support services: on-site/remote system operations
- ParaStation Modulo serves as a platform for research activities
  - Used and further developed in Exascale-related projects like DEEP, {DEEP, RED, IO}-SEA, EUPEX
  - Also serves as a platform for MSA in Quantum- and AI-related projects like HPCQS, QSolid and CoE RAISE

ParaStation MODULO

DEEP Projects

DEEP-SEA

IO-SEA

RED-SEA

RAISE Center of Excellence

HPCQS

QSOLID

EUPEX European Pilot for Exascale

PARASTATION MODULO SOFTWARE SUITE

ParTec
MODULAR SUPERCOMPUTING

ParaStation CLUSTERTOOLS

Tools for Provisioning and Management

- System management CLI
- Image management
- Rolling updates
- Stateless & stateful booting
- Post-install configuration
- Slurm integration
- Distributed database for system configuration
- HealthChecker integration

ParaStation HEALTHCHECKER

Integrity of the Computing Environment

- Automated error detection & error handling
- Various hook-in points
- No interference with jobs
- TicketSuite integration
- Highly configurable

- 100+ tests (HW/SW):
- Node/System/Fabric level

ParaStation TICKETSUITE

Issue Tracking on System Level

- Manual and automatic ticket creation
- Prioritization
- Routing/Triage
- Documentation and central information hub
- Maintenance planning
- Interfaces with external ticketing systems

ParaStation MPI

Execution Environment and MPI Library

- MPI-4.0-compliant
- MPICH ABI compatible
- Supports multiple interconnects in parallel
- Modularity support
- Network bridging
- PMIx support
- Full Slurm integration

PARASTATION CLUSTER TOOLS

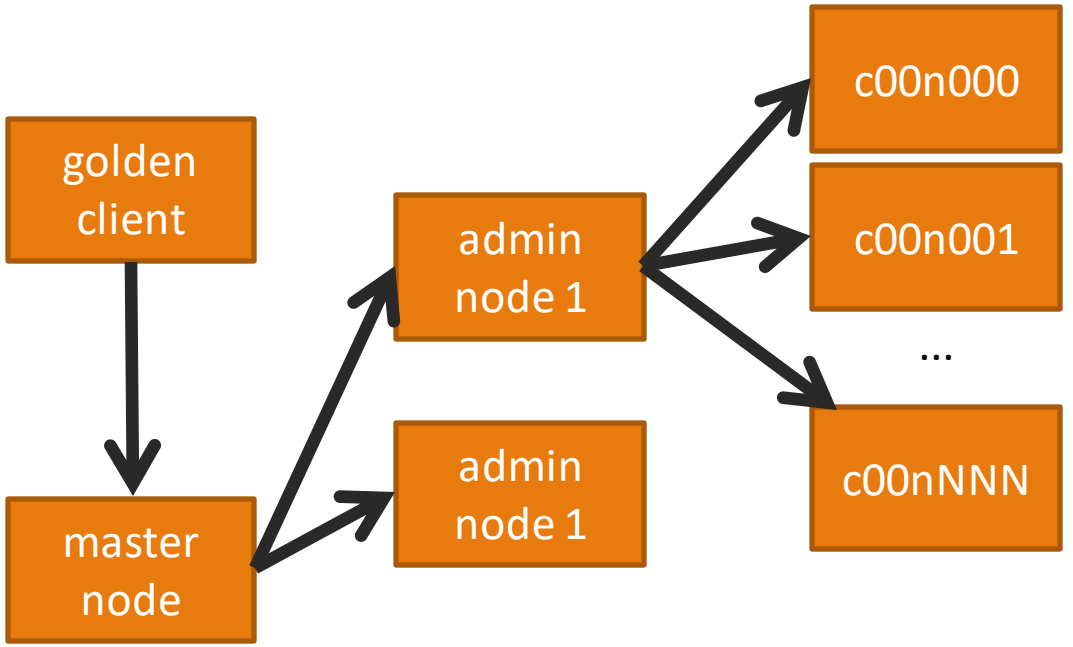
Overview

- Install / update all types of cluster nodes
  - On-disk, disk-less and container-based installation
- Image-based
  - Local modifications after cloning supported
  - Supports multiple images per node
  - Image may install on-disk, disk-less or in container
  - Rolling update supported
  - Synchronization for configuration changes
- Automatic PXE boot of nodes for
  - System install, or
  - System diagnostics / maintenance
- Set of CLI tools for system management
  - Configuration and image handling
  - Console redirection and other IPMI functionality
    - Redfish support under development
  - Provides uniform interface, hides away hardware differences
- Tight integration with HealthChecker and TicketSuite
  - High degree of automation

ParaStation CLUSTERTOOLS

Image Handling

- Image comprises:
  - Directory tree
  - Local modifications (post-install scripts, overrides)
  - Meta-information (name, timestamp, ...)
- Excludes (get/put)
  - Automatically excluded: Remote and non-disk, file systems, auto-generated files
  - Additionally, exclude files (global, per cluster, per image) can be specified
- Uses rsync to transfer images
- Image preparation
  - On golden client (can also be a VM or container)
  - Directly in image environment, using systemd-nspawn
  - Locking mechanism to ensure consistency
  - Support for admin-supplied changelog for each change



Retrieve image: `pscluster image get <IMAGE>`

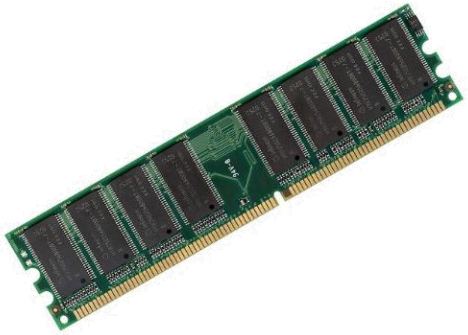
Roll out image: `pscluster image put <IMAGE>`

Node Installation (x86_64/aarch64)

- Requirements
  - PXE boot support (legacy BIOS or UEFI)
  - Remote control via IPMI enabled (BMC configured)
  - Redfish support under development
  - Association: node ↔ MAC address of NIC
  - Support for persistent installs (disk) and volatile installs (RAM)
- Process
  - PXE boots grub2 loader
  - Loader boots kernel + initramfs
  - Image kernel is used
  - Initramfs
    - Establish network connection
    - Sync config
    - Optionally: create disk layout
    - Sync full image
    - Run post-install scripts to configure the node

Node Installation

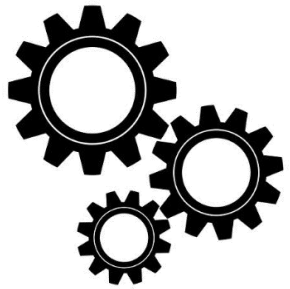
- Persistent installation to disk
  - Disk layout described
  - Support for LVM, SW RAID, different file system types …
- Volatile installation
  - Aka “disk-less”
  - Image is kept in-memory (tmpfs)
  - “Look and feel” similar to persistent installs: update image, install rpms, …
- Same image might be used for persistent and volatile installs
  - Special handling recommended for:
    - Syslog: forwarded to admin node
    - Kdump: forwarded/saved on admin node



PARASTATION CLUSTER TOOLS

Post-Install

- After image has been copied — node specifics need to be configured
  - Hostname
  - Network interfaces: management, IPoIB, file systems, special routes, name resolution, …
  - BMC: user, password, network config, time & date, …
  - Grub2 / boot configuration, initramfs,
- Solution: post install scripts, using psconfigure for common configuration tasks
  - Turns data base (psconfig) entries into
    - Commands (ipmitool lan set, nmcli, …)
    - Configuration files (ifcfg-en, …)
  - Parametrized, ready-made plugins already provided for common configuration taks



psconfigure / psconfig — Network Example

```
network:mngt
DevMTU     = "1500"
DevName    = "enp0s3"
DevNetMask = "255.255.255.0"
DevNetwork = "10.2.8.0"
```

```
network:ib
DevMTU     = "1500"
DevName    = "ib0"
DevNetMask = "255.255.255.0"
DevNetwork = "10.2.9.0"
```

```
host:cnode007
...
MngtNet                = ptr(["network:mngt"])
MngtNet.DevHWAddress   = "00:aa:bb:cc:dd:ee"
MngtNet.DevIPAddress   = "10.2.8.77"
IPoIBNet.              = ptr(["network:ib"])
IPoIBNet.DevIPAddress  = "10.2.9.77"
```

```
psconfigure … MngtNet …
```

```
DEVICE  = "enp0s3"
TYPE    = "ethernet"
IPADDR  = "10.2.8.77"
NETMASK = "255.255.255.0"
HWADDR  = "00:aa:bb:cc:dd:ee"
```

```
psconfigure … IPoIBNet …
```

```
DEVICE  = "ib0"
TYPE    = "infiniband"
IPADDR  = "10.2.9.77"
NETMASK = "255.255.255.0"
```

Container Installation

- pscluster supports installation of an image into a container
  - Same image as for persistent and volatile installs
  - Currently LXC containers are supported
- Install new container: `pscluster node reinstall`
  - Copy image to container directory on hosting node
  - Run post install scripts (container versions)
- Once the container is up and running:
  - Update container: `pscluster node update`
  - Get image: `pscluster image get`

Rolling Updates

- Image updates might be rolled out during production using a rolling update
  - Image has to be prepared
  - All related nodes are offlined at once with special note using:

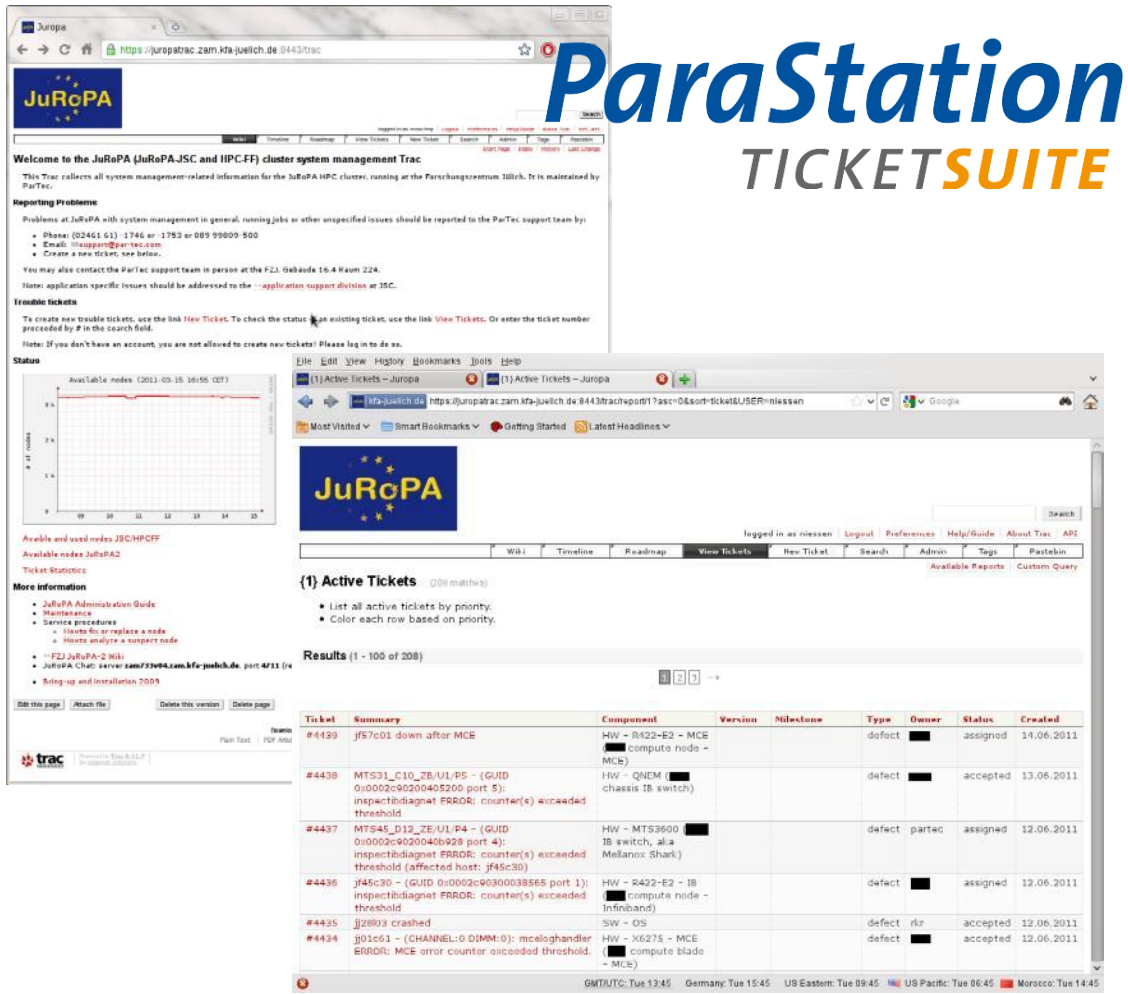
```
psmaintenance -r -t now jwc0[0-9]n[000-287]
```

  - As soon as the nodes become free they are handled by the checkbot:
    - Update node
    - Run post-update scripts
    - Check node
    - Online node
  - Updates are synchronized with batch system / job runs

PARASTATION TICKET SUITE

Central Information Hub

- Central location for problem management
  - Usually dedicated to a single system
  - Deployed in the early phase of the project
  - Accessible by all involved people
- Based on “Trac” open source software
  - Configured and further enhanced by ParTec with plugins and CLI tools
  - GitLab integration currently under development
- Various interfaces for easy integration and synchronization
- Versatile, user-friendly on-the fly generation of statistics

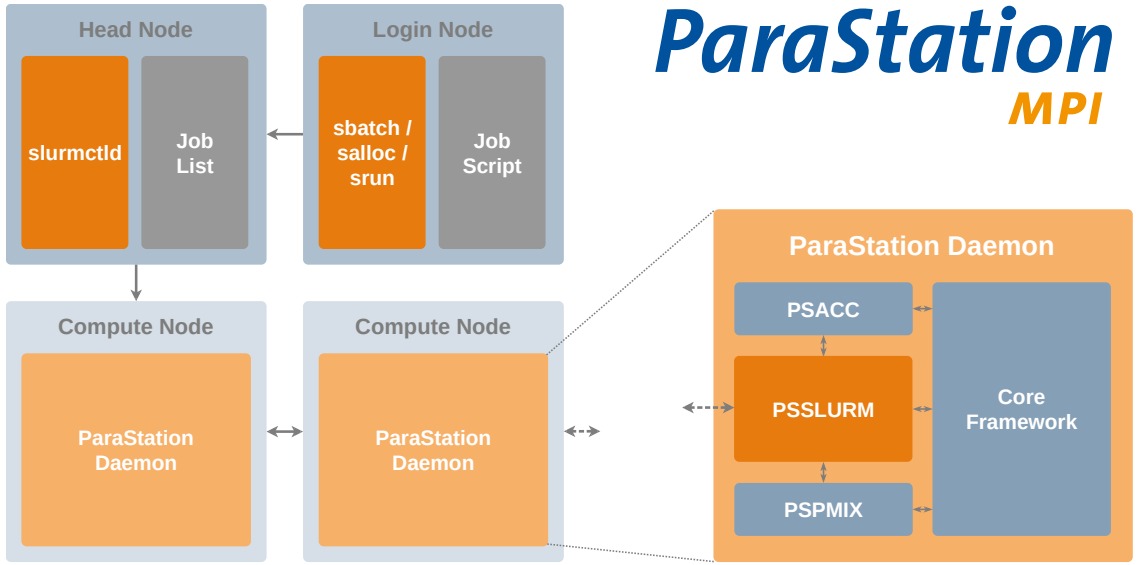


ParaStation TICKETSUITE

PARASTATION MPI (PROCESS MANAGER)

Scalable Process Manager

- Scalable network of MPI process management daemons
  - One instance running on each of the computational nodes
  - Responsible for process startup and control
  - Responsible for intra-job resource assignment
  - Provides precise resource monitoring
  - Provides a PMIx server to the application
  - Guarantees proper cleanup after jobs
- psslurm: Full integration for Slurm
  - Implemented as plugin (i.e., loadable shared library) to the ParaStation Management daemon
  - Replaces node-local Slurm daemons
  - Enforces resource limits
  - Collects misc. information, e.g., accounting, energy, file system usage, ... and forwards it to the slurmctld

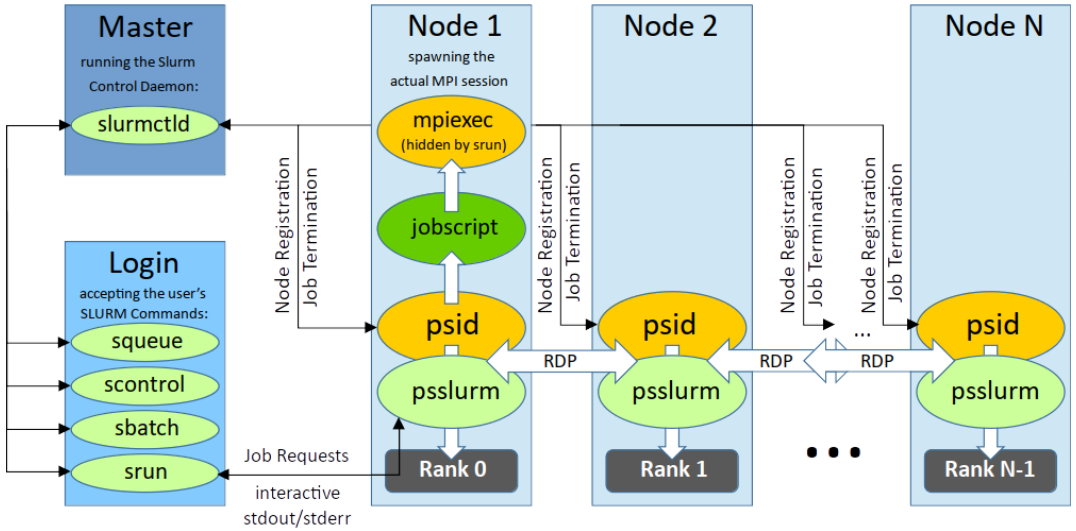


Source code available on GitHub:
https://github.com/ParaStation/psmgmt

PARASTATION MPI (PROCESS MANAGER)

Scalable Process Manager

- Advantages of the psslurm integration
  - Benefit from proven functionality, stability, and scalability of the ParaStation Process Manager for starting and controlling parallel application jobs
  - Benefit from extra features in heterogeneous environments
  - Reduce the number of daemons on the compute nodes
  - Integration with ParaStation HealthChecker via parallel prologue / epilogue
  - Fully controlled code base: Allows to quickly fix problems and to add unique features

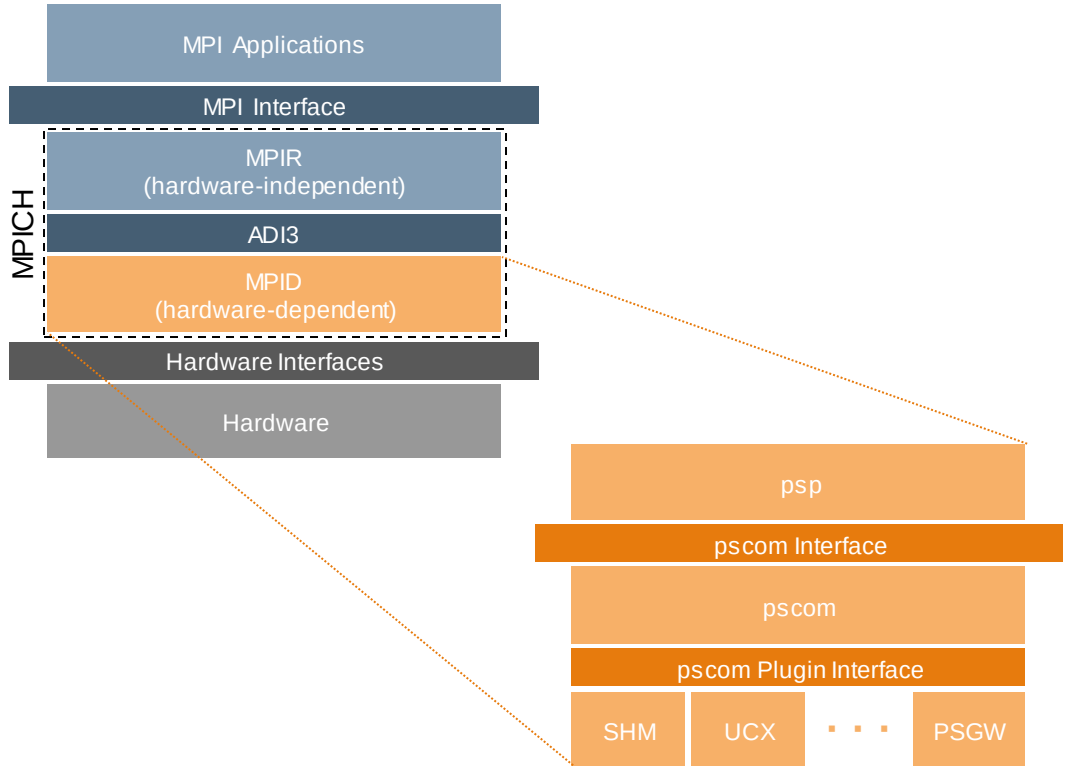


ParaStation MPI

PARASTATION MPI

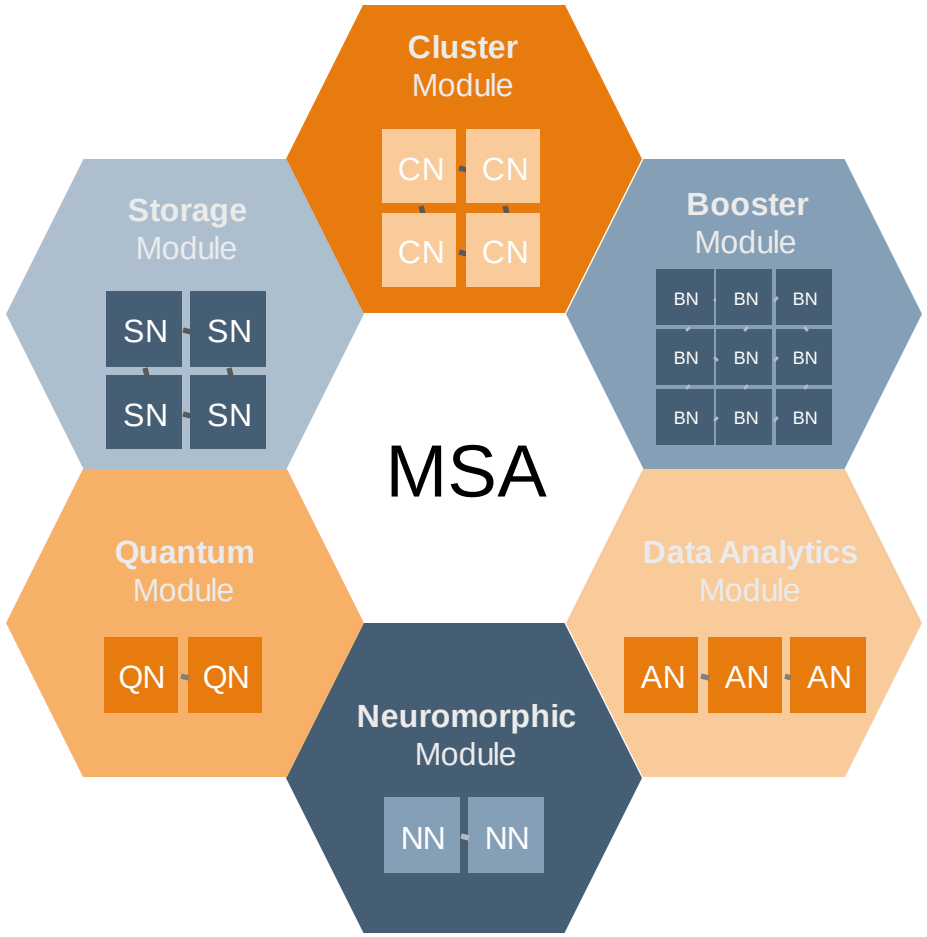
Architecture

- Based on MPICH 4.1
  - Support MPICH tools for tracing, debugging, etc.
  - Integrates into MPICH on the MPID layer by implementing an ADI3 device
  - The PSP Device is powered by pscom—a low-level point-to-point communication library
  - Support the MPICH ABI Compatibility Initiative
  - Tightly integrated with the ParaStation process manager (e.g., for the provision of process sets)
- Support for various transports / protocols via pscom plugins
  - Support for InfiniBand, Omni-Path, BXI, etc.
  - Concurrent usage of different transports
  - Transparent bridging between any pair of networks enabled by gateway capabilities
- Proven to scale up to ~3,500 nodes and more than 140,000 processes per job



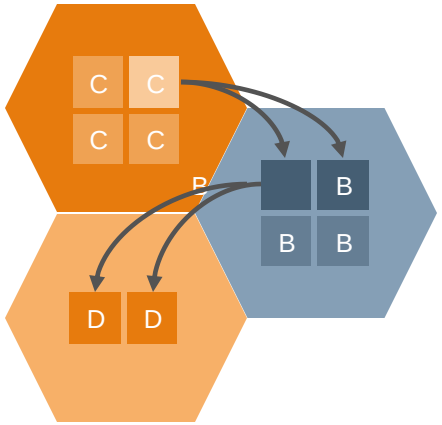
MODULAR SUPERCOMPUTING ARCHITECTURE

- Generalization of the Cluster-Booster Concept
  - Heterogeneity on the system level
  - Effective resource sharing
- Any number of (specialized) modules possible
  - Cost-effective scaling
  - Extensibility of existing modular systems by adding modules
- Fit application diversity
  - Large-scale simulations
  - Data analytics
  - Machine/Deep Learning, AI
  - Hybrid-quantum Workloads
- Achieve leading scalability and energy efficiency
  - Exascale-ready!
- Unified software environment for running across all modules
  - Enabled by the ParaStation Modulo software suite

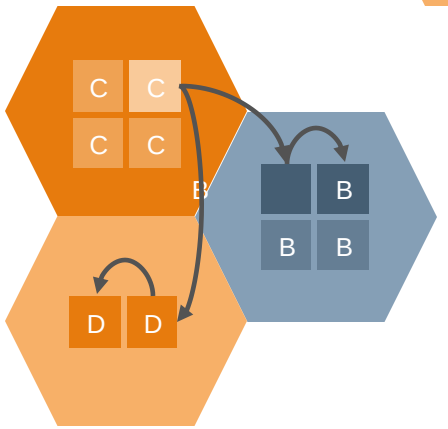


MSA AWARENESS

- Support for multi-level hierarchy-aware collectives
  - Optimize communication patterns to the topology of the MSA
  - Assumption: Inter-module communication is the bottleneck
  - Dynamically update the communication patterns (experimental)
- API extensions for accessing modularity information
  - New MPI split type for communicators (`MPIX_COMM_TYPE MODULE`)
  - Provide the module id via the `MPI_INFO_ENV` object
- MPI Network Bridging
  - Connect any pair of interconnect and protocol
  - Transparent to the application layer



Binary Tree
(default)



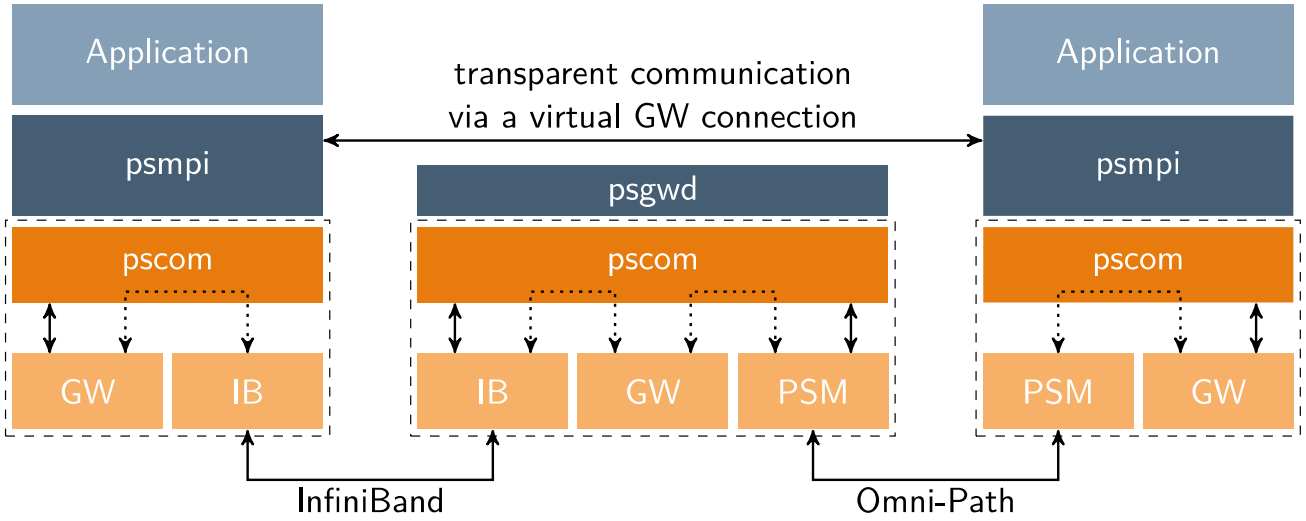
Hierarchical
(MSA-aware)

MPI NETWORK BRIDGING

- Transparent communication across networks
  - Use a gateway when two processes are not directly connected through the same network
  - Bridging between any pair of interconnects supported by pscom (e.g., InfiniBand, Omni-Path, BXI, etc.)

- Static routing
  - Use the same gateway for different destinations
  - Virtual GW connections provide full transparency to the application layer

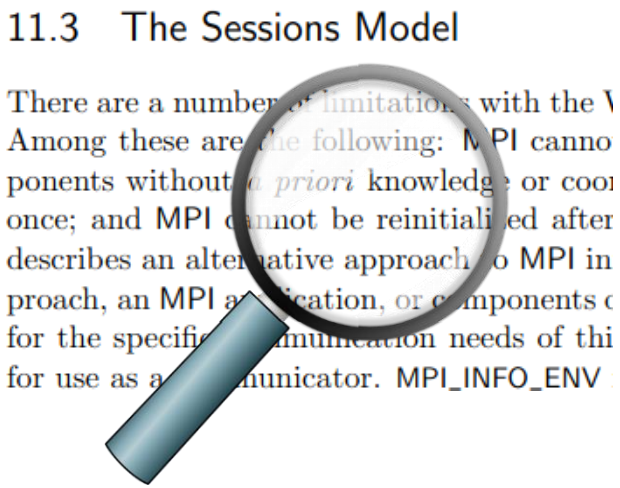
- Successfully deployed in production environments
  - Implemented first for the JURECA Cluster-Booster System
  - Bridging between Mellanox EDR and Intel Omni-Path



PARASTATION MPI

Malleability for MPI Applications

- Malleability features developed in the context of DEEP-SEA
  - Adding or removing of HPC resources during job run time
  - Support MPI-4 sessions in ParaStation MPI (extend MPI4 to support dynamic sessions)
  - Job-intiated: Job releases resources or asks the scheduler for more resources
  - Scheduler-initiated: Scheduler decides to re-organize resource usage, e.g. to optimize job queue
  - Externally initiated: Meta scheduler makes decisions, e.g., based on application models
- Triggering mechanism and protocol between scheduler and MPI application is subject to research and collaboration
  - Currently under discussion: employing and possibly extending PMIx
  - Changes to Slurm scheduler likely needed
- Standardization of MPI and PMIx extensions targeted



THE MALLEABILITY CHALLENGE

Application Challenge

- Checkpoint/Restart (CP/RS) with modified resource specs already works today for many apps
- Handshake with scheduler required for automation
  - Scheduler needs more information about jobs
- Dynamic MPI Session support is needed to be able to re-initialize MPI apps on-the-fly (avoiding CP/RS)
  - Application re-factoring required

Scheduler Challenge

- Bi-directional interface with MPI runtime is required
- Deal with rectangular node x walltime shapes that are compressible/expandable over time
  - Requires new scheduling algorithms & policies

Operational Challenge

- New policies ruling which jobs get compressed and which get expanded while balancing
  - System utilization
  - Science throughput
  - Turnaround times
  - Energy efficiency
- How to incentivate users to walk the extra mile?
  - Requires rewards mechanism in fair-share usage model

WHAT'S NEXT?

CURRENT AND FUTURE DEVELOPMENTS

OPTIMIZATION

- Performance optimizations (e.g., further improve BXI support)
- Expose low-level RMA for improved one-sided communication
- Extend support for hierarchical collectives (e.g., UCC support)

MPI-4

- Improve/extend MPI-4 support
- Tighter integration with the process manager (e.g., for the provision of psets)
- Bring developments upstream

MALLEABILITY

- Dynamic resizing of jobs
- Support for application-driven *(active)* and scheduler-driven *(passive)* malleability
- Leverage PMIx (e.g., `PMIx_Allocation_request`)
- Build upon the MPI Sessions interface

PARASTATION MODULO SOFTWARE SUITE

ParTec MODULAR SUPERCOMPUTING

ParaStation CLUSTERTOOLS

Tools for Provisioning and Management

- System management CLI
- Image management
- Rolling updates
- Stateless & stateful booting
- Post-install configuration
- Slurm integration
- Distributed database for system configuration
- HealthChecker integration

ParaStation HEALTHCHECKER

Integrity of the Computing Environment

- Automated error detection & error handling
- Various hook-in points
- No interference with jobs
- TicketSuite integration
- Highly configurable

- 100+ tests (HW/SW):
- Node/System/Fabric level

ParaStation TICKETSUITE

Issue Tracking on System Level

- Manual and automatic ticket creation
- Prioritization
- Routing/Triage
- Documentation and central information hub
- Maintenance planning
- Interfaces with external ticketing systems

ParaStation MPI

Execution Environment and MPI Library

- MPI-4.0-compliant
- MPICH ABI-compatible
- Supports multiple interconnects in parallel
- Modularity support
- Network bridging
- PMIx support
- Full Slurm integration

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Components

- Local Checking of Nodes
  - Core component
  - Flexible and expandable framework
- Global System Checking
  - Detecting problems not local to single nodes
- Associated Tool “Checkbot”
  - “Automatic admin”
  - Actually, part of ParaStation Cluster Tools

ParaStation HEALTHCHECKER

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Automated Health Checking

- Expandable framework
- Assess health
  - Hardware
  - Software
  - Configuration
- Non-destructive
- Multiple usage scenarios
  - On trigger, e.g., job driven (prologue/epilogue), on reboot, manual stress-test
  - Periodically, run at regular intervals
- Node-local vs. global checking
- **Aims at maximizing system usability**

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Local Checking of Nodes

- Autonomous local checks → Unlimited scalability
- Framework provides unlimited flexibility
- Parallel execution of tests
- Enabled to check remote conditions as well
  - Example: Network connectivity of the node
- Timeout handling at different levels
- PreActions (at test and test set level)
  - Perform clean-up actions
  - Fix transient problems
- PostActions (at test and test set level)
  - Offline nodes; reschedule jobs
  - Create tickets

ParaStation HEALTHCHECKER

The Parastation HealthChecker can

- put a node into a defined state;
- check a node for a defined state;
- take actions based on the result of these checks.

It's just a matter of configuration.

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

ParTec MODULAR SUPERCOMPUTING

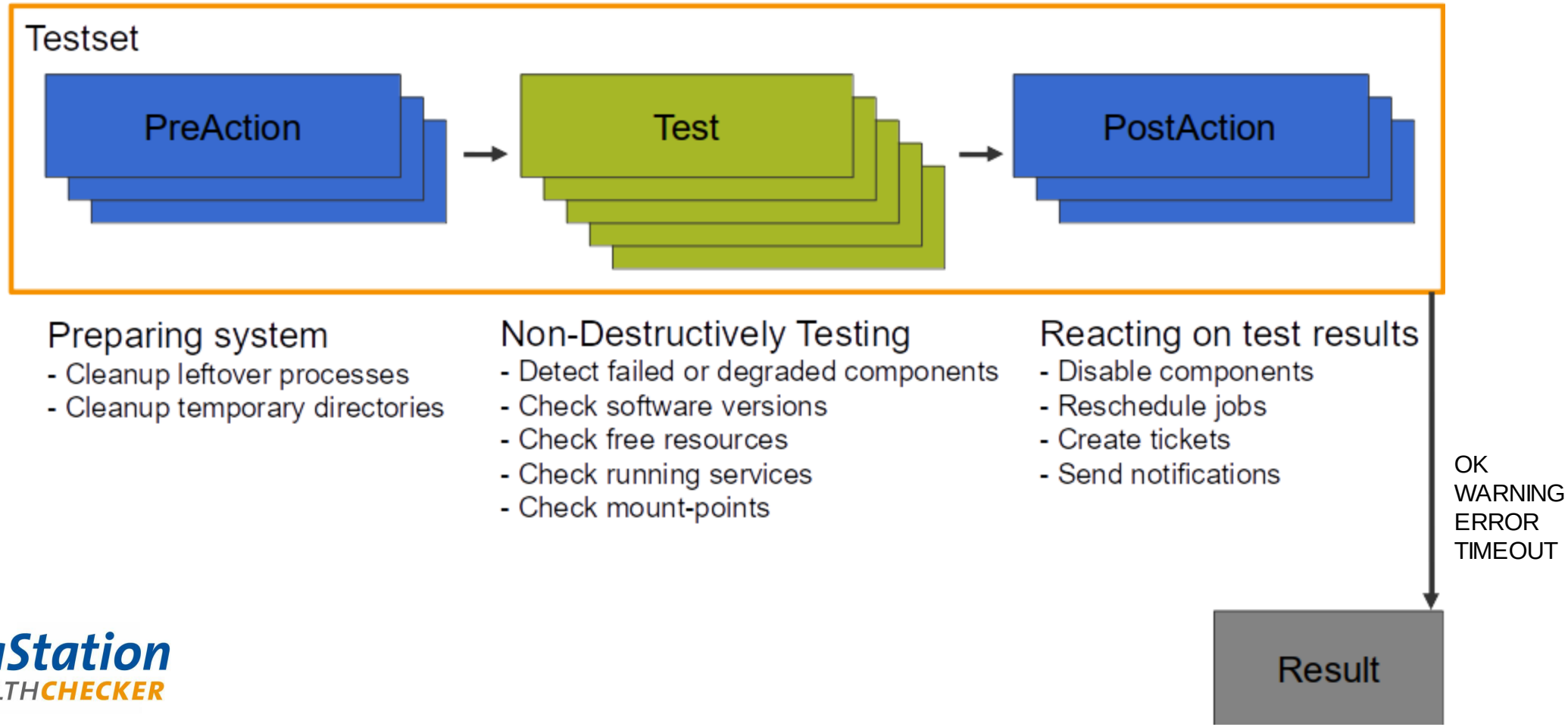
Terminology

Term	Description	Example
Check	Prepared configurable test unit, usually a script	cpu_speed.sh, memory_free.sh, userprocesses.sh
Test	Configured test unit, part of a **test set**, uses **check**	[cpu_speed_3GHz], [memory_free_8GB], [userprocesses]
Test Set	Several **tests** executed together	reboot, prologue, epilogue, stress-test
PreAction	Action executed before a (**test** or) **test set** run	process_cleanup.sh
PostAction	Action executed after a (**test** or) **test set** run	slurm_set_offline.sh

ParaStation HEALTHCHECKER

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Local Checking of Nodes



ParaStation HEALTHCHECKER

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Example: Kill User Processes

- **PreAction**: process_cleanup.sh
  `process_cleanup.sh root,pscd`
  kill all processes not owned by allowed users

- **Test**: [userprocesses] using check *userprocesses.sh*
  `userprocesses.sh root,pscd`
  check if there are processes not owned by allowed users

- **PostAction**: slurm_set_offline.sh
  `slurm_set_offline.sh`
  drain local node if test *userprocesses* failed

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Example: Test Set Configuration in psconfig

```
[hctestset:prologue]
PreActions = ["process_cleanup", "ipc_cleanup", "psid_cleanup", …]
Tests = ["kernel_modules", "cpu_count", "memory_free",
  "memory_badpages", "HCA_pcispeed", "infiniband_phy_state",
  "infiniband_state", "infiniband_speed", "infiniband_counters",
  "net_ping_ib", "daemons", "disk_free", "disk_linkspeed", …]
PostActions = ["node_set_offline"]
Timeout = "58"
TimeoutActionCommand = "/opt/parastation/lib/actions/set_offline.sh"
Break = "never"
```

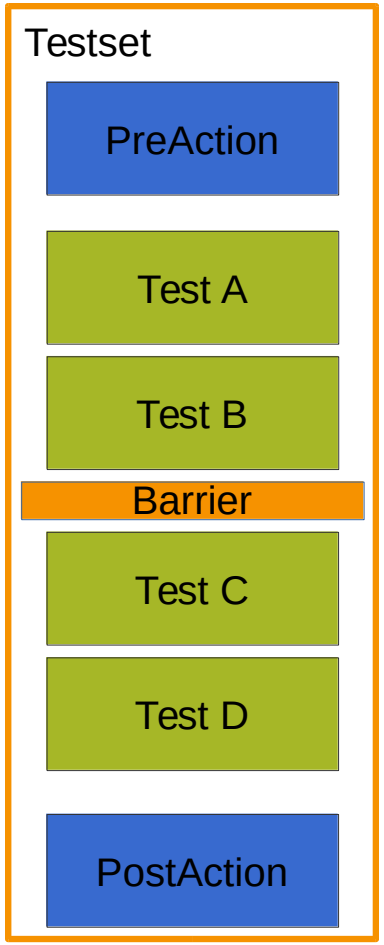
KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Example: Test Configuration in psconfig

```
[+hctest:psid]
.parents = ["class:hctest"]
Command = "/opt/parastation/lib/checks/psid.sh 510 510"
Hardware = ["*"]
NodeTypes = ["compute*", "master", "admin"]
Timeout = "10"
# Inherited keys:
# KillWaitTime = "1"
# NoRepeatTime = "0"
```

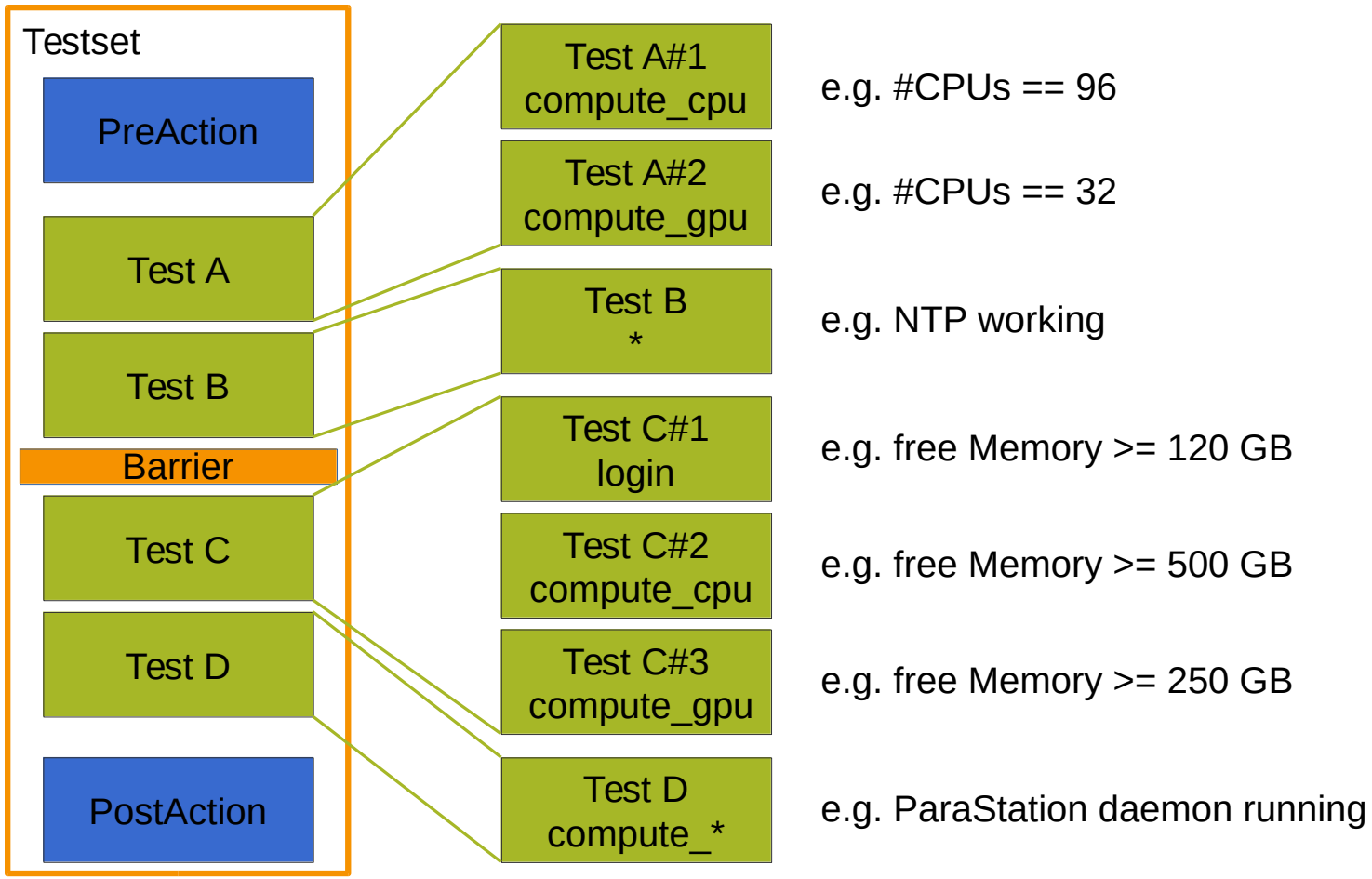
KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Local Checking of Nodes



KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Local Checking of Nodes



e.g. #CPUs == 96

e.g. #CPUs == 32

e.g. NTP working

e.g. free Memory >= 120 GB

e.g. free Memory >= 500 GB

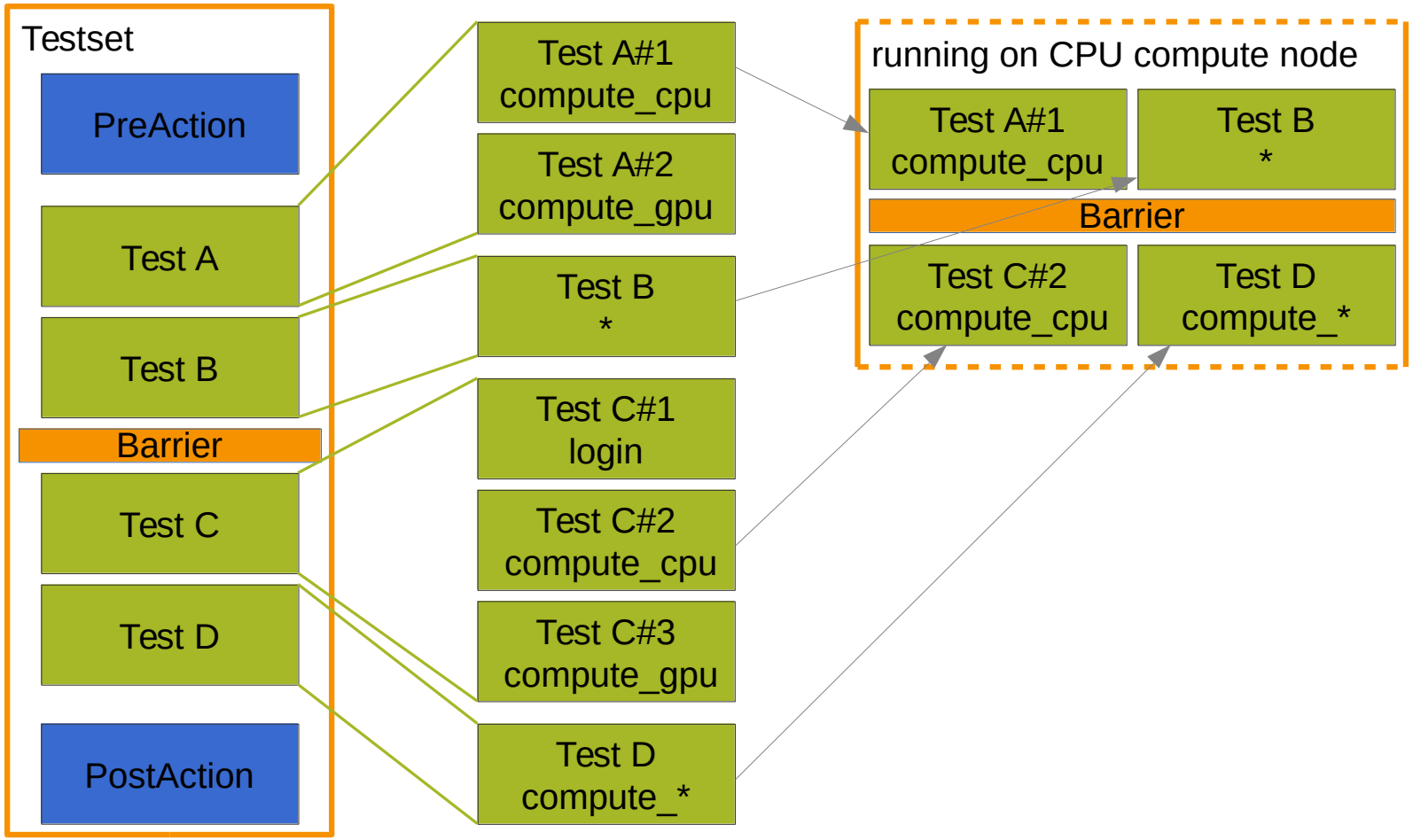
e.g. free Memory >= 250 GB

e.g. ParaStation daemon running

ParaStation HEALTHCHECKER

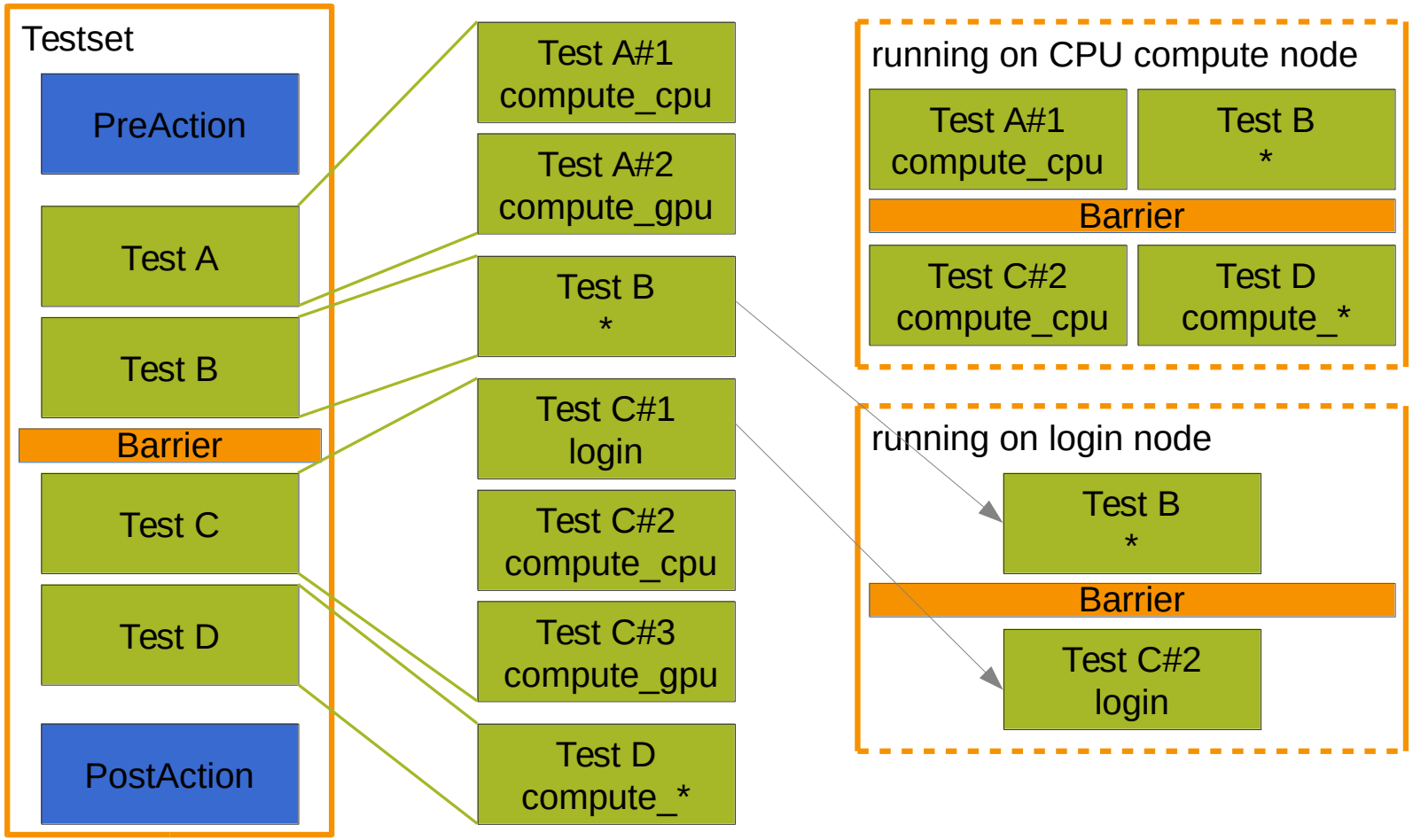
KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Local Checking of Nodes



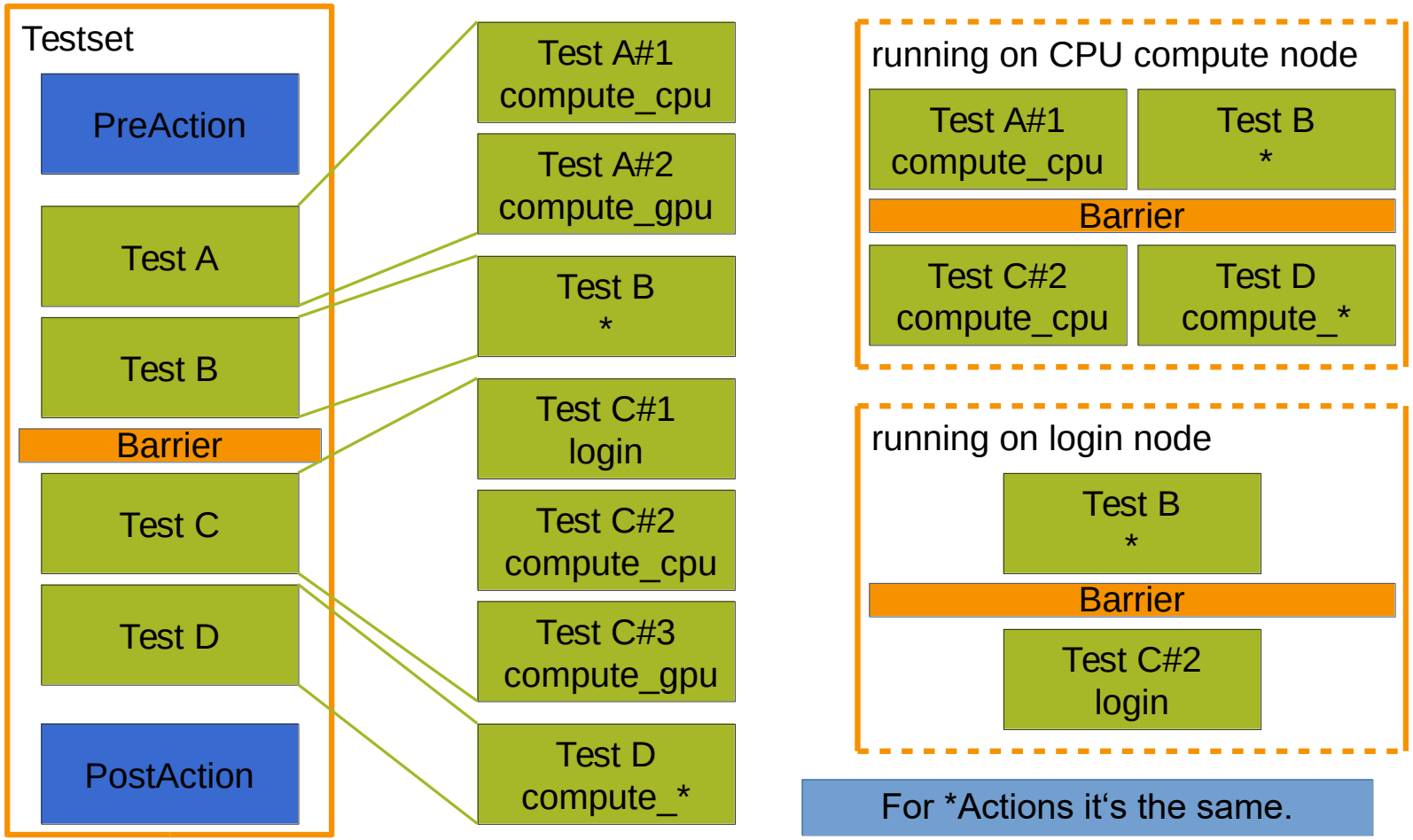
KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Local Checking of Nodes



KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Local Checking of Nodes



KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Design Advantages

- Only define one test set for one situation/trigger point
- Use predefined checks to easily define similar tests
- Limit and overload tests and actions
  to make the same test set suitable for different node types
- Benefit from many years of experience through a large collection of prepared checks

ParaStation HEALTHCHECKER

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

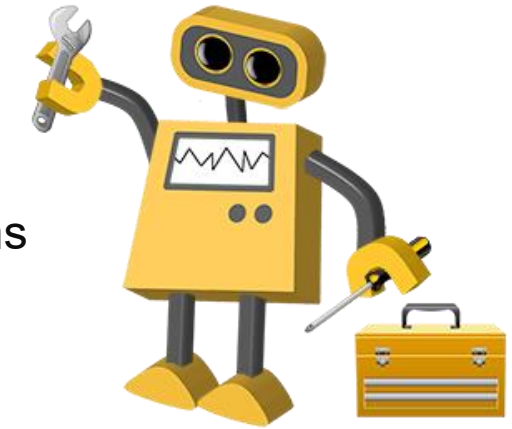
System Global Checking

- Continuous/periodic checks are performed on service nodes
  - No healthcheck actions on compute nodes during job run
- Checks health of other components, e.g.,
  - Monitoring of machine check events
  - Monitoring of log files
    - Compute nodes' logfiles are forwarded to admin nodes
    - OpenSM logs
  - Periodic runs of ibdiagnet
  - Interfacing other monitoring components

ParaStation HEALTHCHECKER

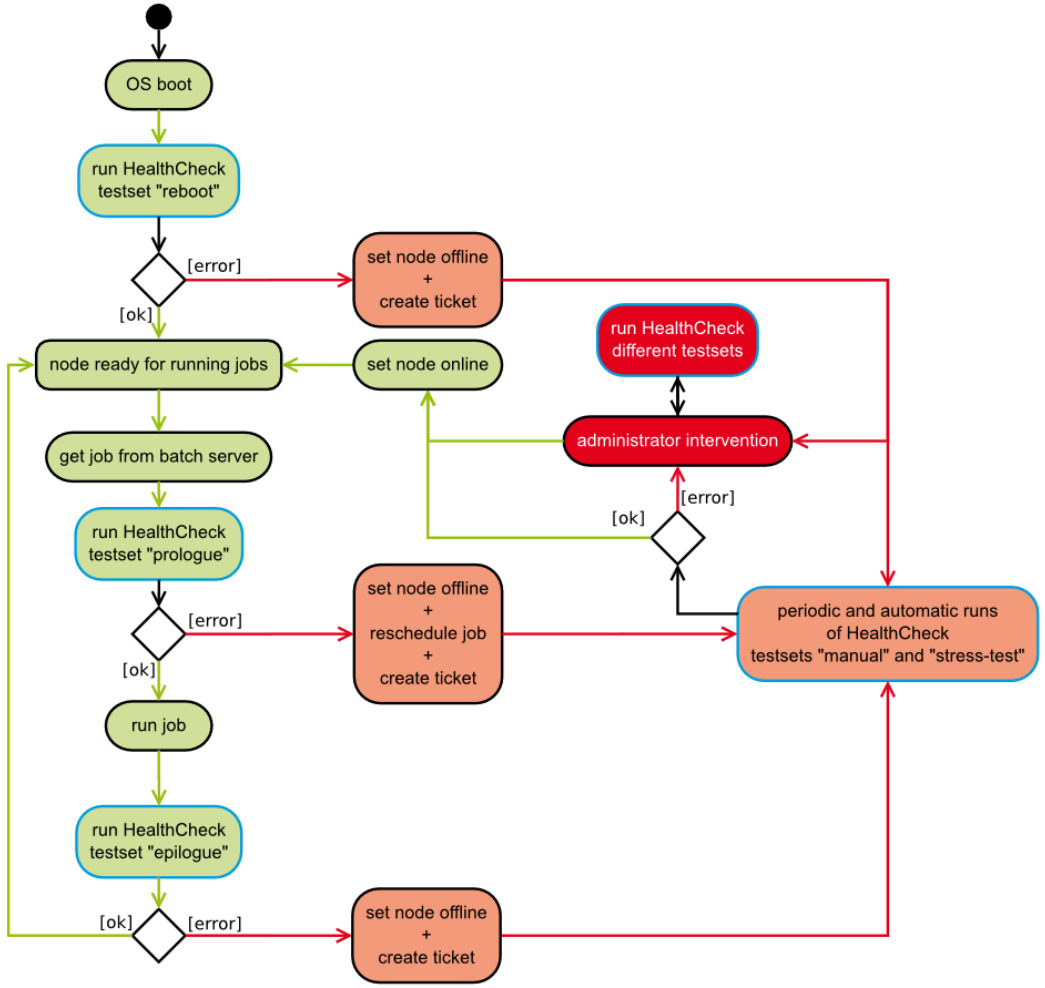
Checkbot

- Runs periodically on a master node for nodes offlined by pshealthcheck
- Can also be triggered manually for a list of nodes (e.g., repaired nodes)
- Actions
  - Powers on nodes (optional)
  - Updates nodes (optional)
  - Runs pshealthcheck
  - Runs “fix” scripts on nodes for failed checks → trying to fix transient problems
  - Takes “good” nodes back online
  - Updates all related tickets
  - Clears maintenance tag in database
  - Clears node “identify” LED



KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

Health Checking in Practice

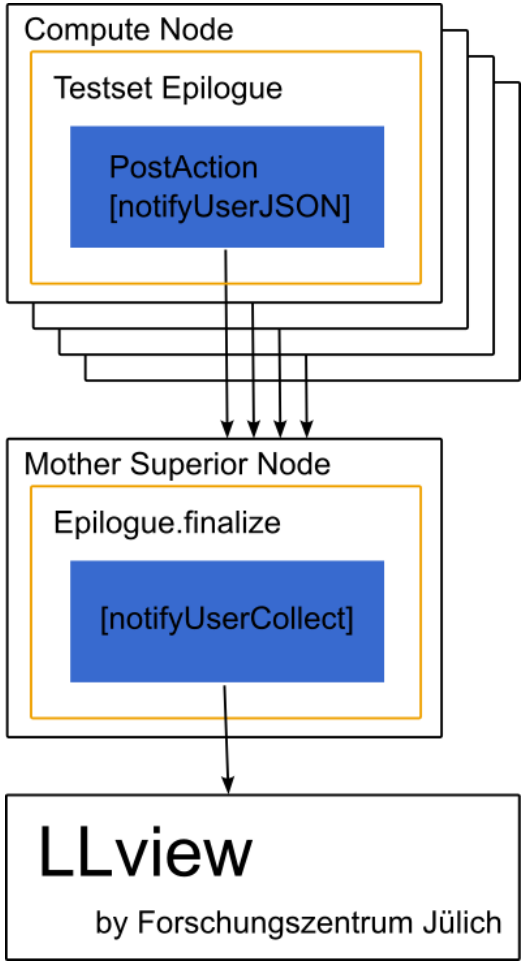


Tests, to address problematic aspects (non-exhaustive list):

- BIOS version and installation date
- Number of available CPUs
- CPU speed and type
- Running services (e.g., syslog, xinetd, systemd status,...)
- Disk space
- Disk health (SMART)
- InfiniBand bandwidth
- InfiniBand error counters and connectivity
- Kernel version
- Working LDAP
- Checksum of critical configuration files
- Free memory (kernel memory leakage!)
- Memory bus speed and size
- Mounted file systems
- DNS configuration and availability
- Network counters
- Ethernet connectivity and speed
- Software versions
- ...

ParaStation HEALTHCHECKER

KEEPING NODES HEALTHY – PARASTATION HEALTHCHECKER

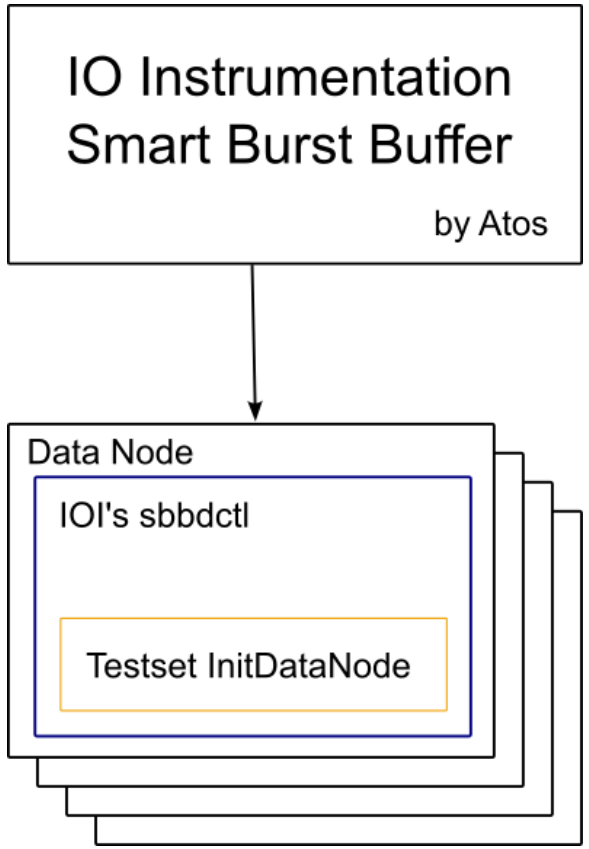


Monitoring Agent

ParaStation HEALTHCHECKER

in

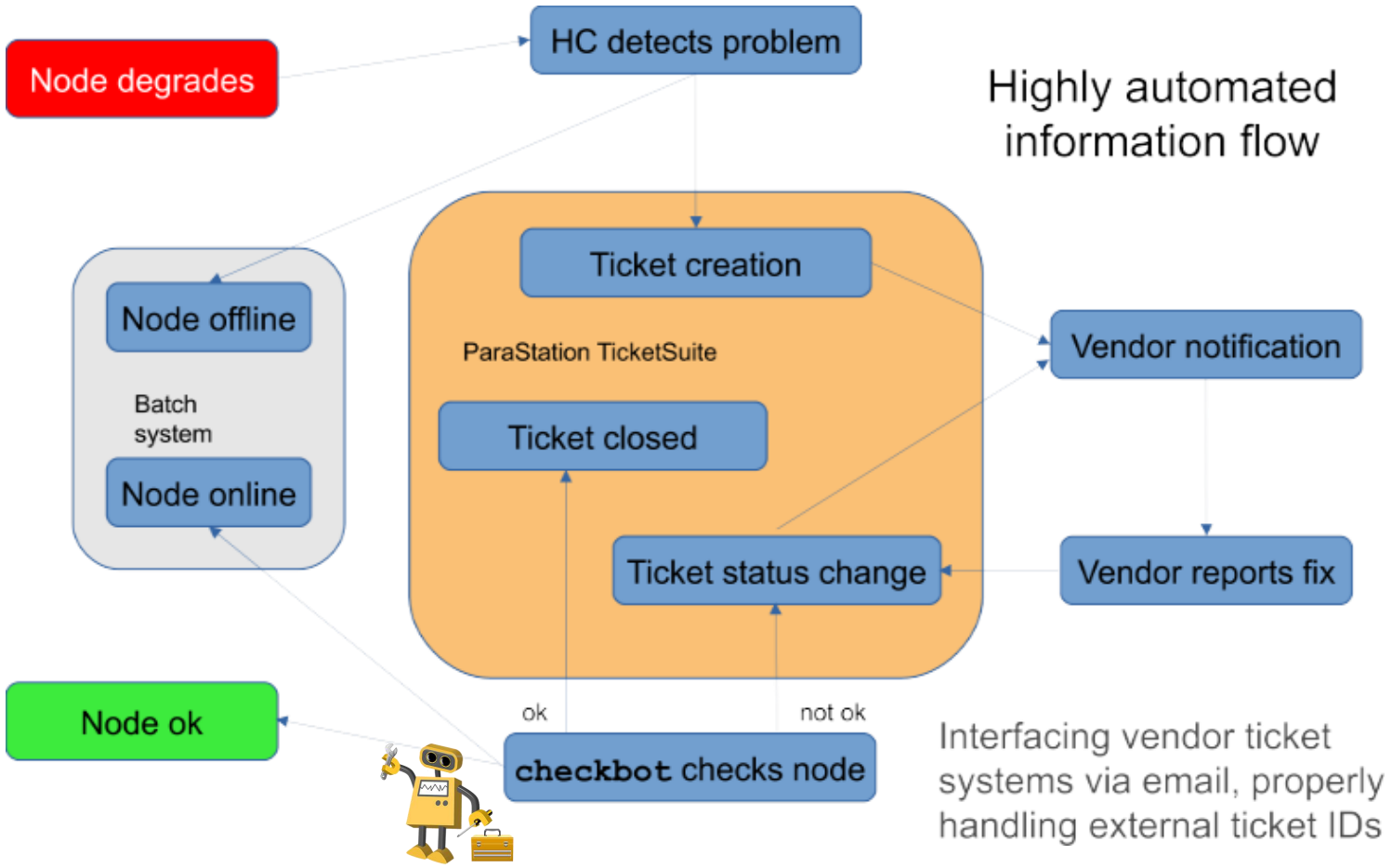
IO-SEA



Health Checking Data Nodes

DEMO

Workflow



THANK YOU FOR YOUR ATTENTION

QUESTIONS?