# Shasta Software Workshop

CUG 2019 – Montreal, Canada

lkaplan@cray.com

# Shasta Software Workshop – Agenda

- Shasta Software Stack overview
  - Themes and strategy
  - Architecture
  - System Management, Linux, PE
- System Management
  - Levels and components
  - Service infrastructure/APIs
  - Image and config management/Boot
  - Security
  - Monitoring
  - Network management
- Discussion
- Break
- Linux
  - Components
  - Slingshot
- User Environment
  - User Access Service/Nodes
  - WLMs
  - Containers for users
  - Cray Programming Environment
  - Analytics
- Storage
- Software Status
- Discussion
- End

# Presenters (in order of appearance)

- Larry Kaplan – Chief Software Architect
- Harold Longley – Manager, management systems
- Jason Rouault – Director, management systems

- Matt Haines – VP, system management and cloud software
- Jonathan “Bill” Sparks – Staff Engineer, cloud hosting
- John Fragalla – Principal Engineer, storage pre-sales
- Dave Poulsen – Senior Program Manager,  strategic customer engagements

# Overview

Larry Kaplan

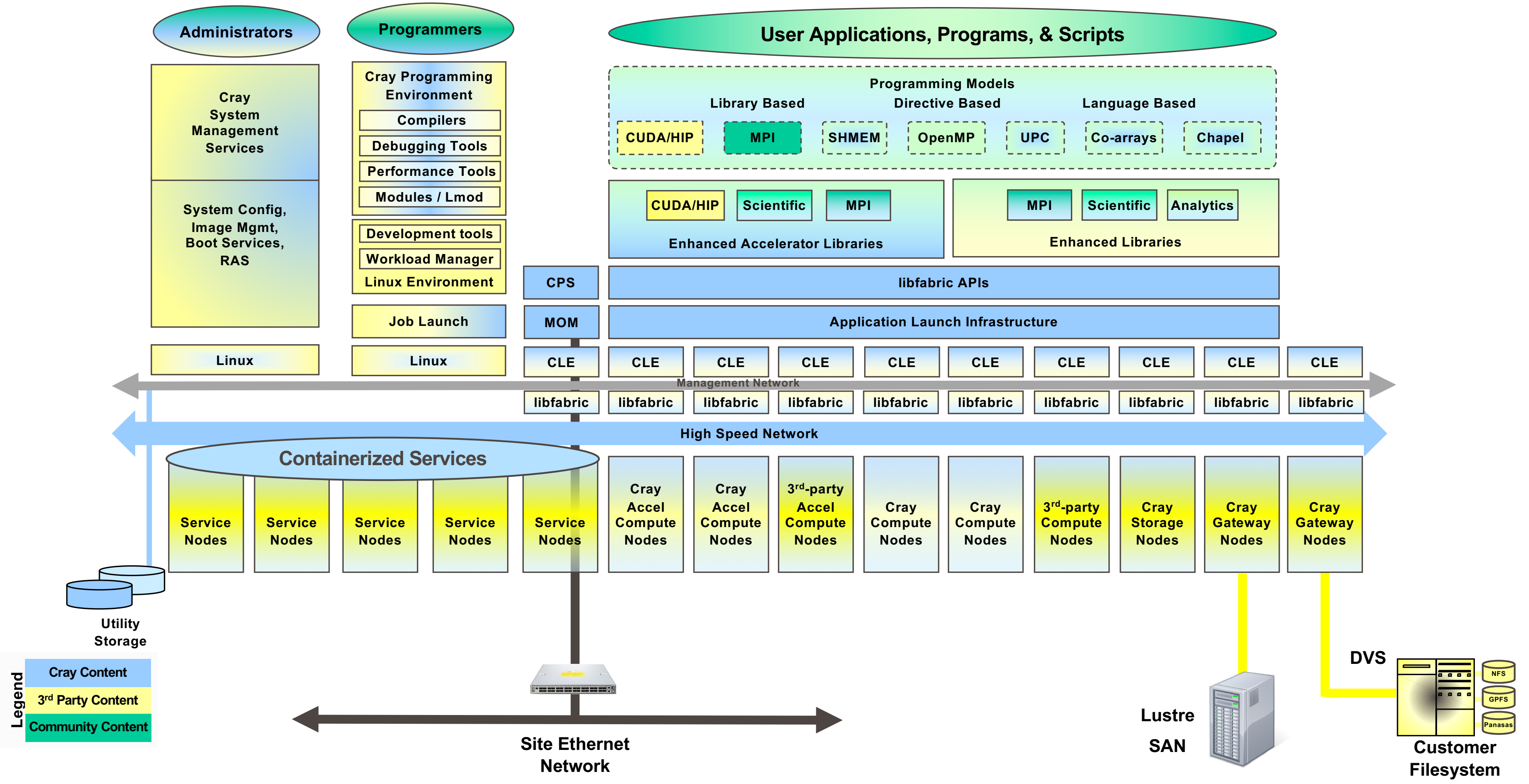
# Shasta Software Themes

| | |
|---|---|
| Scaling to exascale | • Building on current management and Linux scalability enhancements<br>• MPI scalability across full systems |
| Toward zero downtime | • Separate management and operating environments<br>• Concurrent maintenance<br>• Health and resiliency support |
| Run any workflow | • Customer choice of operating environment<br>• Broad container support<br>• Workload management and orchestration |
| Modularity | • Clean APIs between software components<br>• Customizable with easy integration |

# Shasta Software Strategy

- **Evolution of proven XC software stack**
  - CLE managed ecosystem and image management
  - Resilient services and other reliability features
  - High performance networking software
- **Emphasis on modularization and APIs**
  - Supports separation of software components
  - APIs will be published
  - Flexibility allows customers to engage with the software stack in new way
- **Leverage Open Software**
  - Use existing open solutions where Cray differentiation not needed

# Cray Shasta System



Administrators

Programmers

User Applications, Programs, & Scripts

Cray System Management Services

System Config, Image Mgmt, Boot Services, RAS

Linux

Cray Programming Environment

Compilers

Debugging Tools

Performance Tools

Modules / Lmod

Development tools

Workload Manager

Linux Environment

Job Launch

Linux

Programming Models

Library Based

Directive Based

Language Based

CUDA/HIP

MPI

SHMEM

OpenMP

UPC

Co-arrays

Chapel

CUDA/HIP

Scientific

MPI

Enhanced Accelerator Libraries

MPI

Scientific

Analytics

Enhanced Libraries

CPS

libfabric APIs

MOM

Application Launch Infrastructure

CLE | CLE | CLE | CLE | CLE | CLE | CLE | CLE | CLE | CLE

Management Network

libfabric | libfabric | libfabric | libfabric | libfabric | libfabric | libfabric | libfabric | libfabric | libfabric

High Speed Network

Containerized Services

Service Nodes | Service Nodes | Service Nodes | Service Nodes | Service Nodes

Cray Accel Compute Nodes | Cray Accel Compute Nodes | 3rd-party Accel Compute Nodes | Cray Compute Nodes | Cray Compute Nodes | 3rd-party Compute Nodes | Cray Storage Nodes | Cray Gateway Nodes | Cray Gateway Nodes

Utility Storage

Site Ethernet Network

Lustre SAN

DVS

NFS

GPFS

Panasas

Customer Filesystem

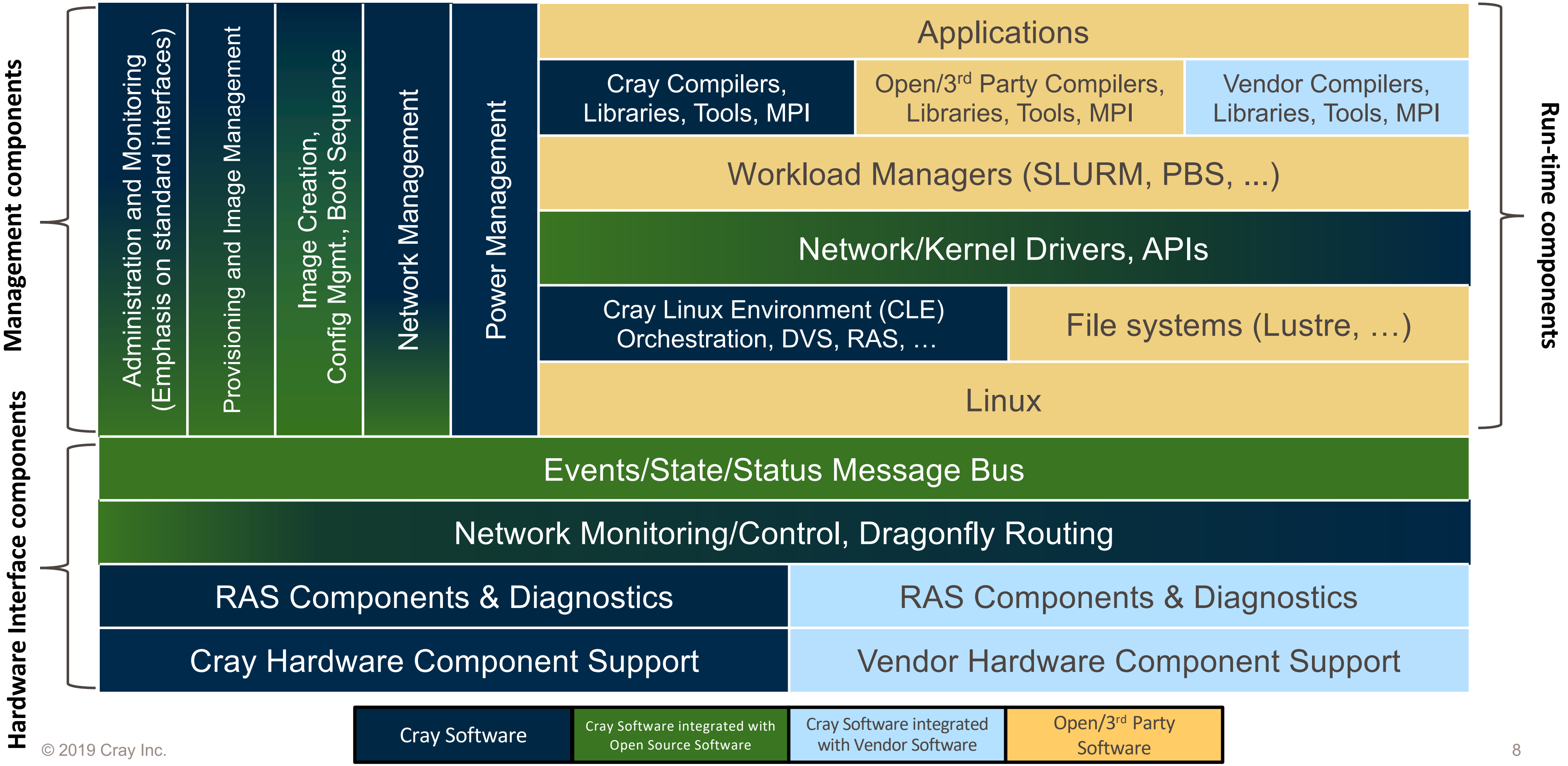
Legend

Cray Content

3rd Party Content

Community Content

# Shasta Software – Slingshot



Management components

Administration and Monitoring (Emphasis on standard interfaces)

Provisioning and Image Management

Image Creation, Config Mgmt., Boot Sequence

Network Management

Power Management

Applications

Cray Compilers, Libraries, Tools, MPI

Open/3rd Party Compilers, Libraries, Tools, MPI

Vendor Compilers, Libraries, Tools, MPI

Workload Managers (SLURM, PBS, ...)

Network/Kernel Drivers, APIs

Cray Linux Environment (CLE) Orchestration, DVS, RAS, …

File systems (Lustre, …)

Linux

Run-time components

Hardware Interface components

Events/State/Status Message Bus

Network Monitoring/Control, Dragonfly Routing

RAS Components & Diagnostics

RAS Components & Diagnostics

Cray Hardware Component Support

Vendor Hardware Component Support

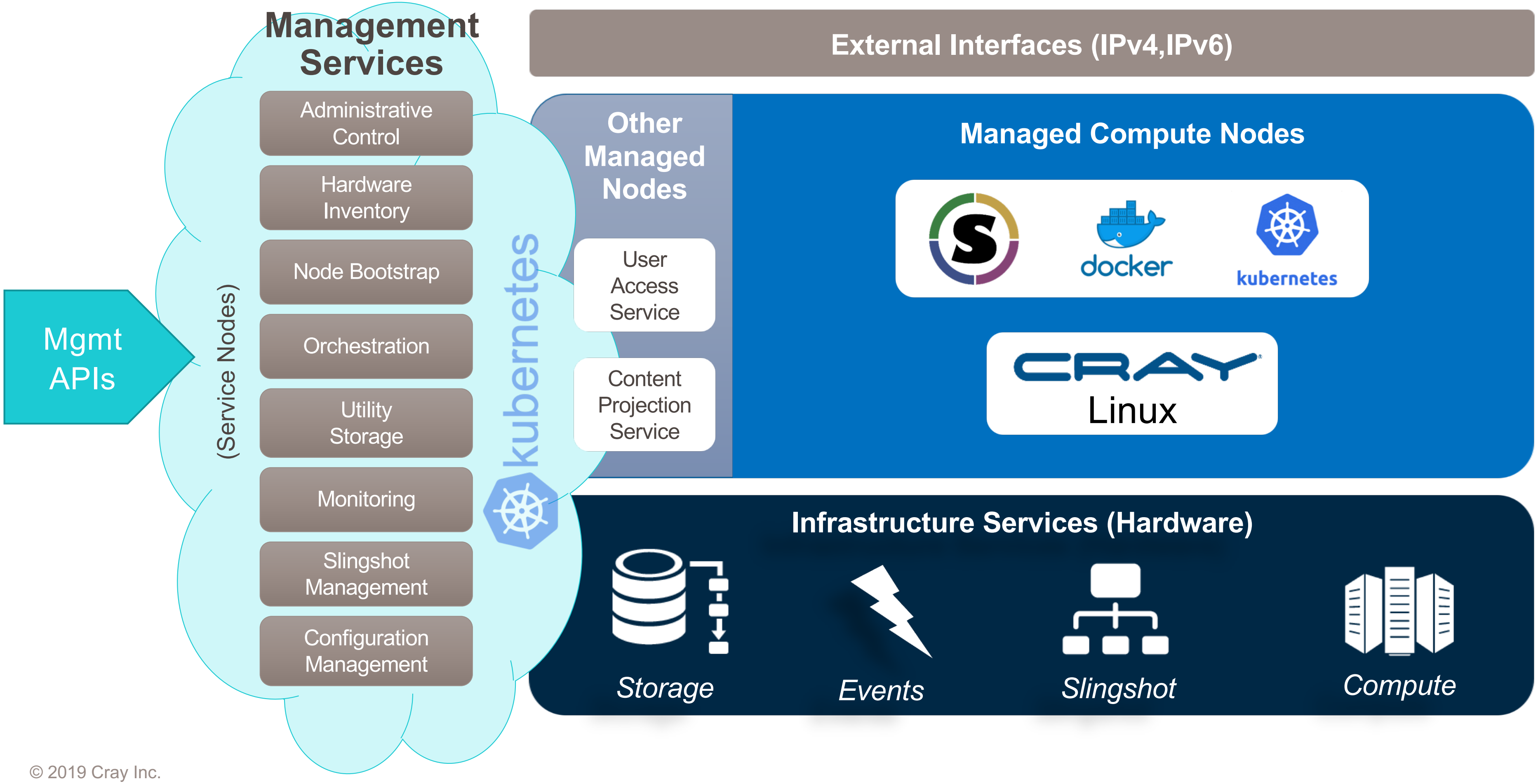
Cray Software

Cray Software integrated with Open Source Software

Cray Software integrated with Vendor Software

Open/3rd Party Software

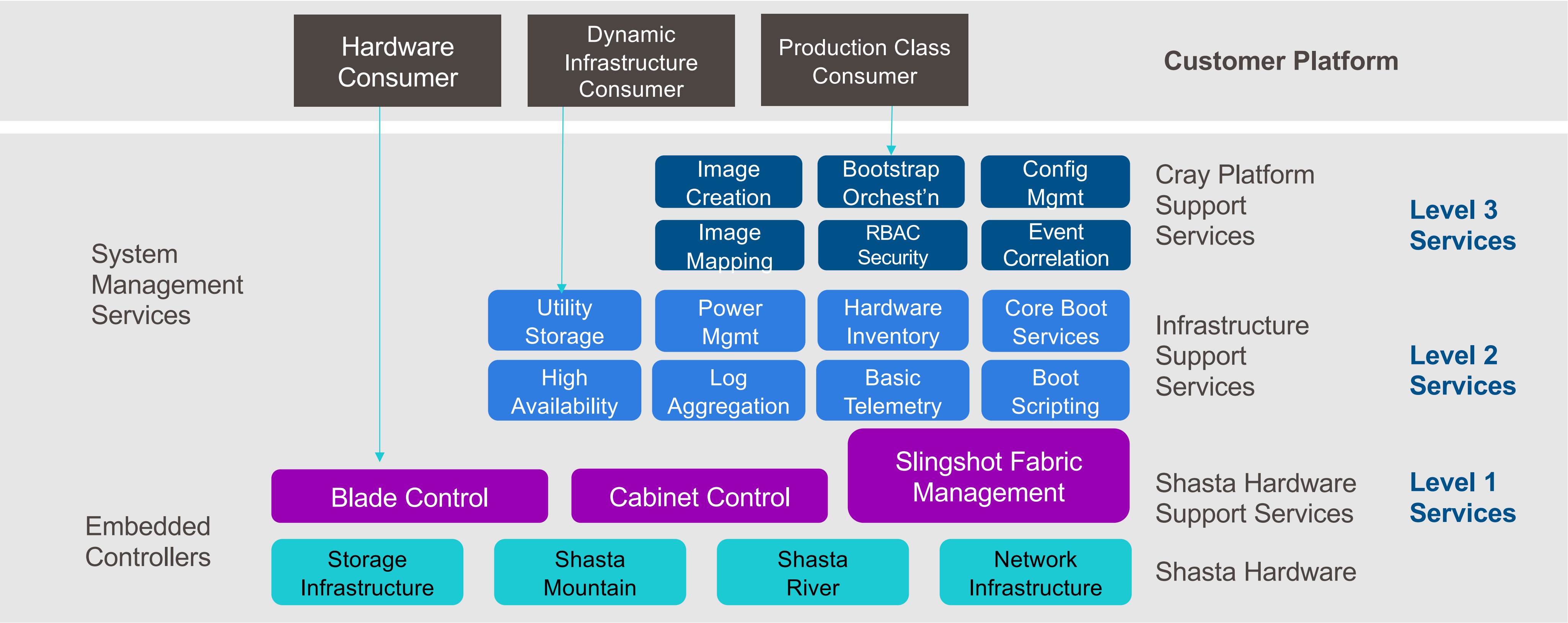
# Shasta System Software



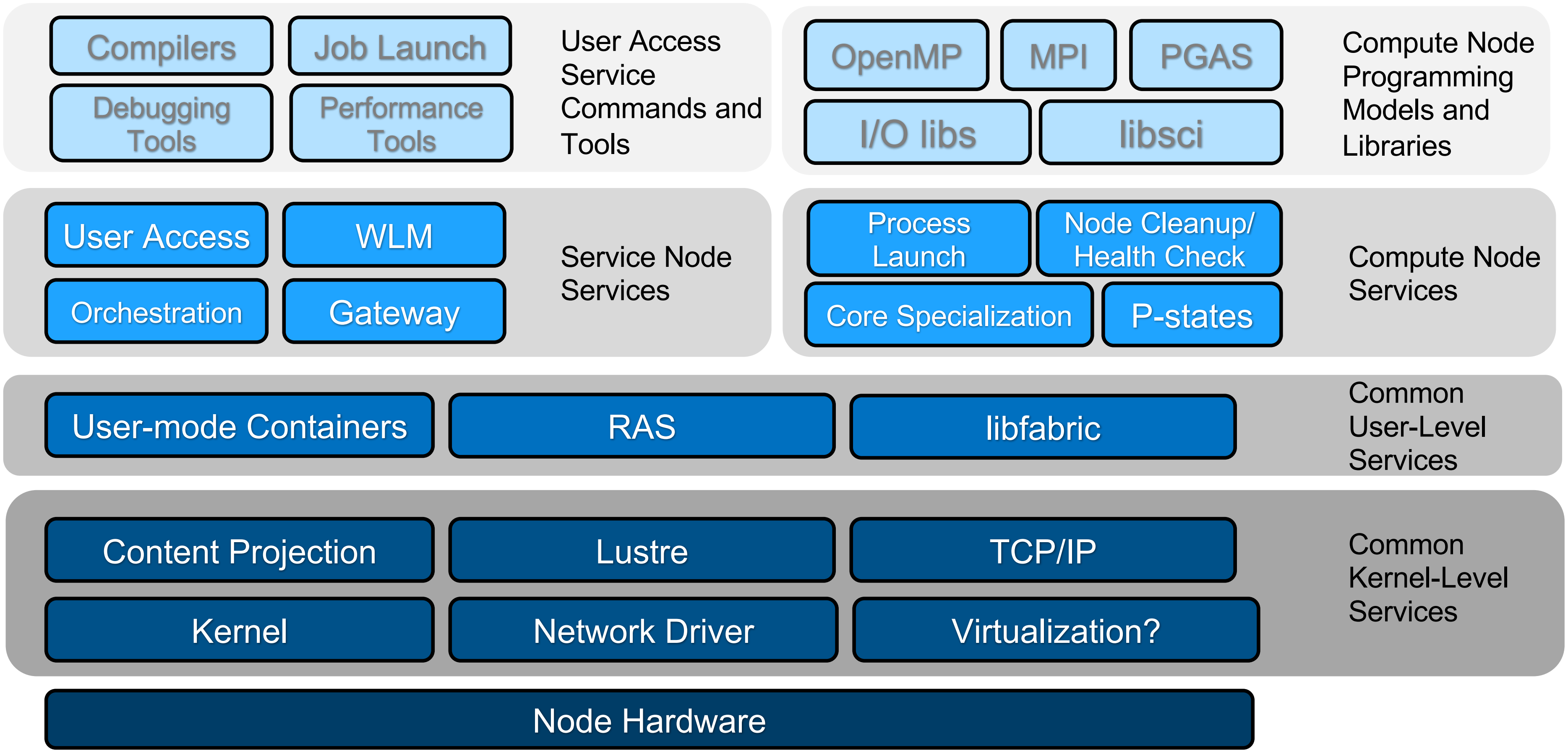
# Separation, Containers, and Orchestration

- **Separate “Management Services” from platform-centric “Managed Services”**
  - E.g. boot service is platform independent but Netroot service is specific to CLE
- **Orchestrated containerized services**
  - Both management and managed
- **Advantages**
  - Supports deployment and upgrade of unique software stacks
  - Supports independent scale-out and resiliency for services
  - Clear distinction between infrastructure and platform/ecosystem

# Shasta System Management



# CLE Software Components



Compilers
Job Launch
Debugging Tools
Performance Tools
User Access Service Commands and Tools

OpenMP
MPI
PGAS
I/O libs
libsci
Compute Node Programming Models and Libraries

User Access
WLM
Orchestration
Gateway
Service Node Services

Process Launch
Node Cleanup/ Health Check
Core Specialization
P-states
Compute Node Services

User-mode Containers
RAS
libfabric
Common User-Level Services

Content Projection
Lustre
TCP/IP
Kernel
Network Driver
Virtualization?
Common Kernel-Level Services

Node Hardware

# Shasta Development Environment

## Programming Languages

### Languages

- Fortran
- C
- C++
- Chapel
- Python
- R

## Programming Models

### Distributed Memory

- Cray MPI<br>SHMEM

### Shared Memory / GPU

- OpenMP

### PGAS & Global View

- UPC<br>Fortran coarrays<br>Coarray C++<br>Chapel

## Programming Environments

### ProgEnv-

- Cray Compiling Environment PrgEnv-cray
- GNU PrgEnv-gnu
- 3rd Party compilers (AMD, Intel, PGI, etc) PrgEnv-???

## Optimized Libraries

### Scientific Libraries

- LAPACK
- ScaLAPACK
- BLAS
- Iterative Refinement Toolkit
- FFTW

### I/O Libraries

- NetCDF
- HDF5

## Tools

### Environment setup

- Modules / Lmod
- Tool Enablement (supports Spack, CMake, etc.)

### Performance Analysis

- CrayPAT
- Cray Apprentice$^2$

### Porting

- Reveal
- CCDB

## Tools (continued)

### Debuggers

- gdb4hpc
- TotalView
- DDT

### Debugging Support

- Abnormal Termination Processing (ATP)
- STAT
- Valgrind4hpc

## Analytics / AI **

### AI Toolboxes

- Cray Urika AI - Analytics
- Chapel AI

### DL Frameworks

- Cray PE DL Scalability Plugin

🟦 Cray Developed

🟪 Cray added value to 3rd party

🔷 3rd party packaging

⬛ Licensed ISV SW

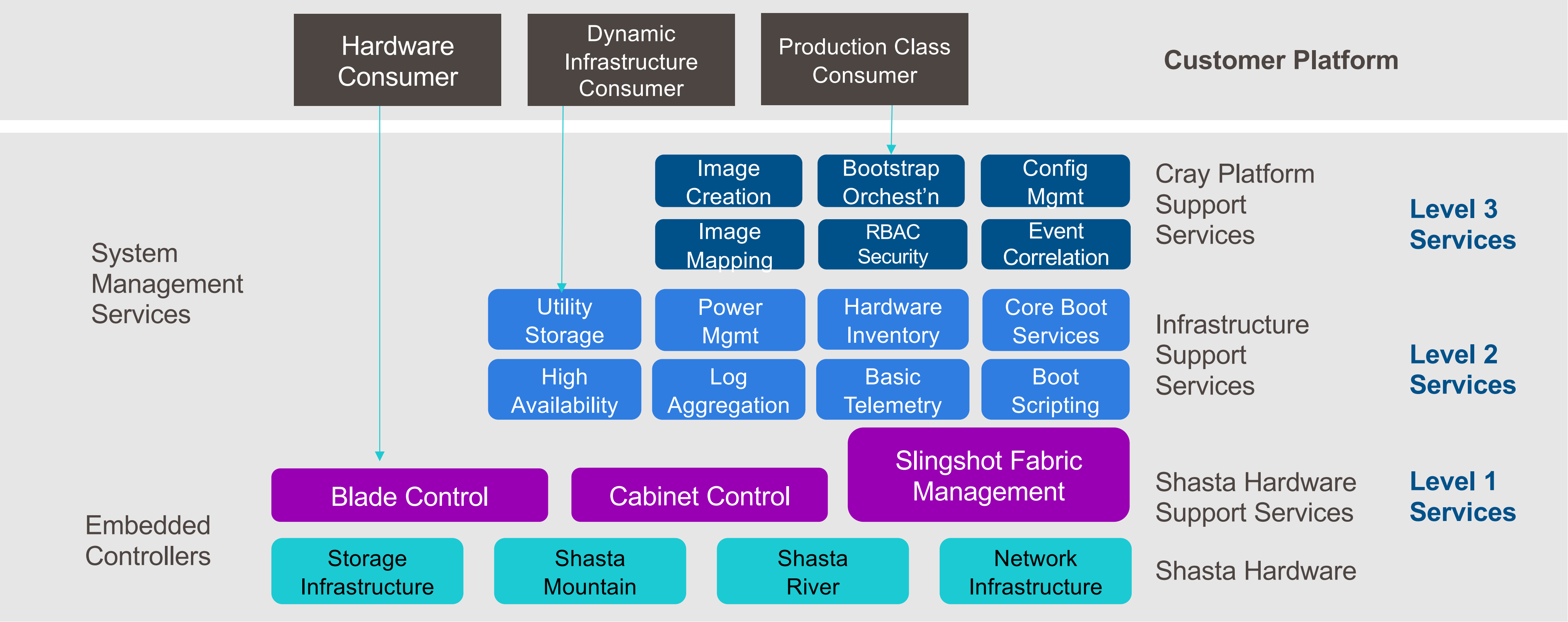
** Not PE dependent

# System Management

Harold Longley

Jason Rouault

# Shasta System Management



Hardware Consumer

Dynamic Infrastructure Consumer

Production Class Consumer

**Customer Platform**

System Management Services

Image Creation

Bootstrap Orchest'n

Config Mgmt

Image Mapping

RBAC Security

Event Correlation

Cray Platform Support Services

**Level 3 Services**

Utility Storage

Power Mgmt

Hardware Inventory

Core Boot Services

High Availability

Log Aggregation

Basic Telemetry

Boot Scripting

Infrastructure Support Services

**Level 2 Services**

Blade Control

Cabinet Control

Slingshot Fabric Management

Shasta Hardware Support Services

**Level 1 Services**

Embedded Controllers

Storage Infrastructure

Shasta Mountain

Shasta River

Network Infrastructure

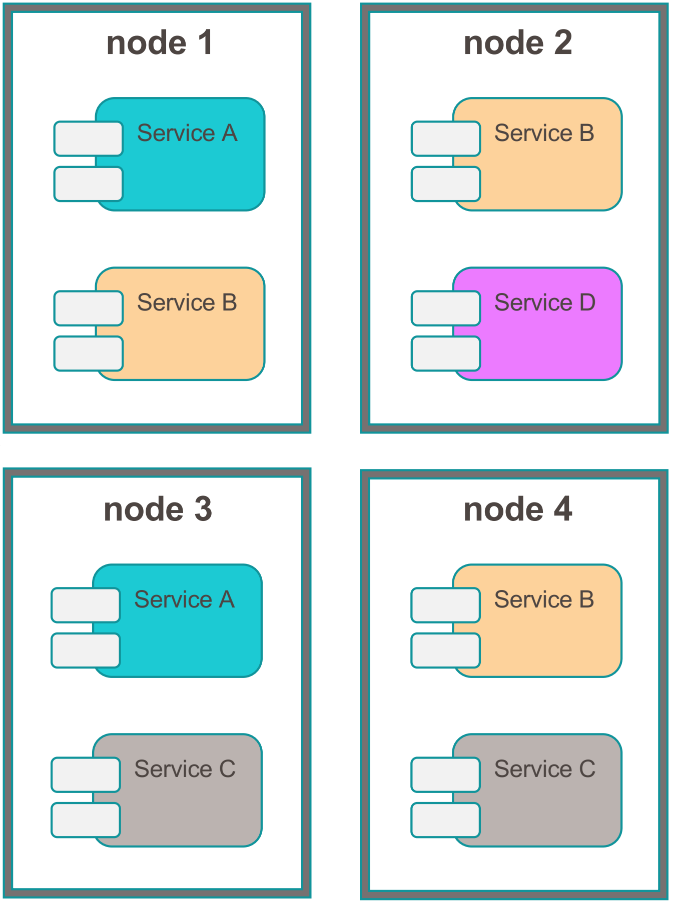
Shasta Hardware

# Service Based Architecture

- Services
  - Represent a logical activity within the system
  - Are self-contained
  - Only expose interfaces (or APIs) for communication with other services and components
- Modular approach
  - Decouples the services from each other
  - Allows for greater ease of maintenance and replacement of the components within each service
  - As long as the API behaves the same, there is no need for another service or component that relies on it to know its internal structure or implementation

# Distributed Services

- Compose a service or tool by integrating distributed, separately-maintained, and deployed software components
- Enabled by technologies and standards that make it easier for components to communicate and cooperate over a network
- Increases the reliability, availability, and scalability of the management functions
- Enables scaling across multiple hosts
- Allows the system management requests to be load balanced across a distributed system for automatic scalability and reliability



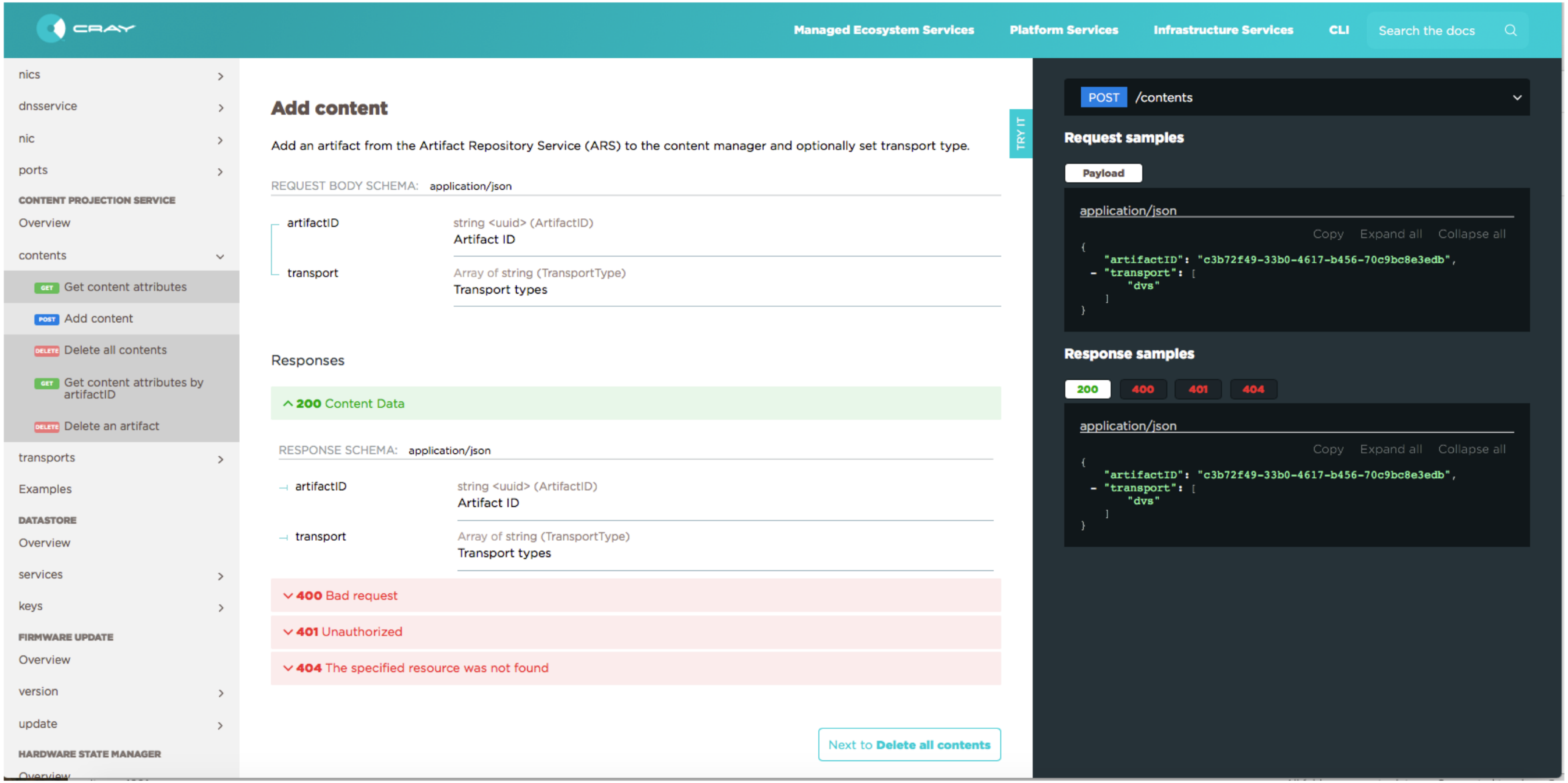
Multiple non-compute nodes distribute service load

# REST API

- A RESTful API is an application program interface (API) that uses HTTP requests
  - GET, DELETE, PUT, PATCH, POST

- REST API specification (swagger/OpenAPI 3.0) for Cray microservices used to generate
  - API documentation
    - Provided in docker image and in tarball for webserver
  - API server stubs for the microservice
  - API client code for the Cray CLI framework

# API Documentation from REST API Specification



## Add content

Add an artifact from the Artifact Repository Service (ARS) to the content manager and optionally set transport type.

REQUEST BODY SCHEMA: application/json

| | |
|---|---|
| artifactID | string <uuid> (ArtifactID)<br>Artifact ID |
| transport | Array of string (TransportType)<br>Transport types |

### Responses

200 Content Data

RESPONSE SCHEMA: application/json

| | |
|---|---|
| artifactID | string <uuid> (ArtifactID)<br>Artifact ID |
| transport | Array of string (TransportType)<br>Transport types |

400 Bad request

401 Unauthorized

404 The specified resource was not found

POST /contents

### Request samples

Payload

application/json

```
{
    "artifactID": "c3b72f49-33b0-4617-b456-70c9bc8e3edb",
  - "transport": [
        "dvs"
    ]
}
```

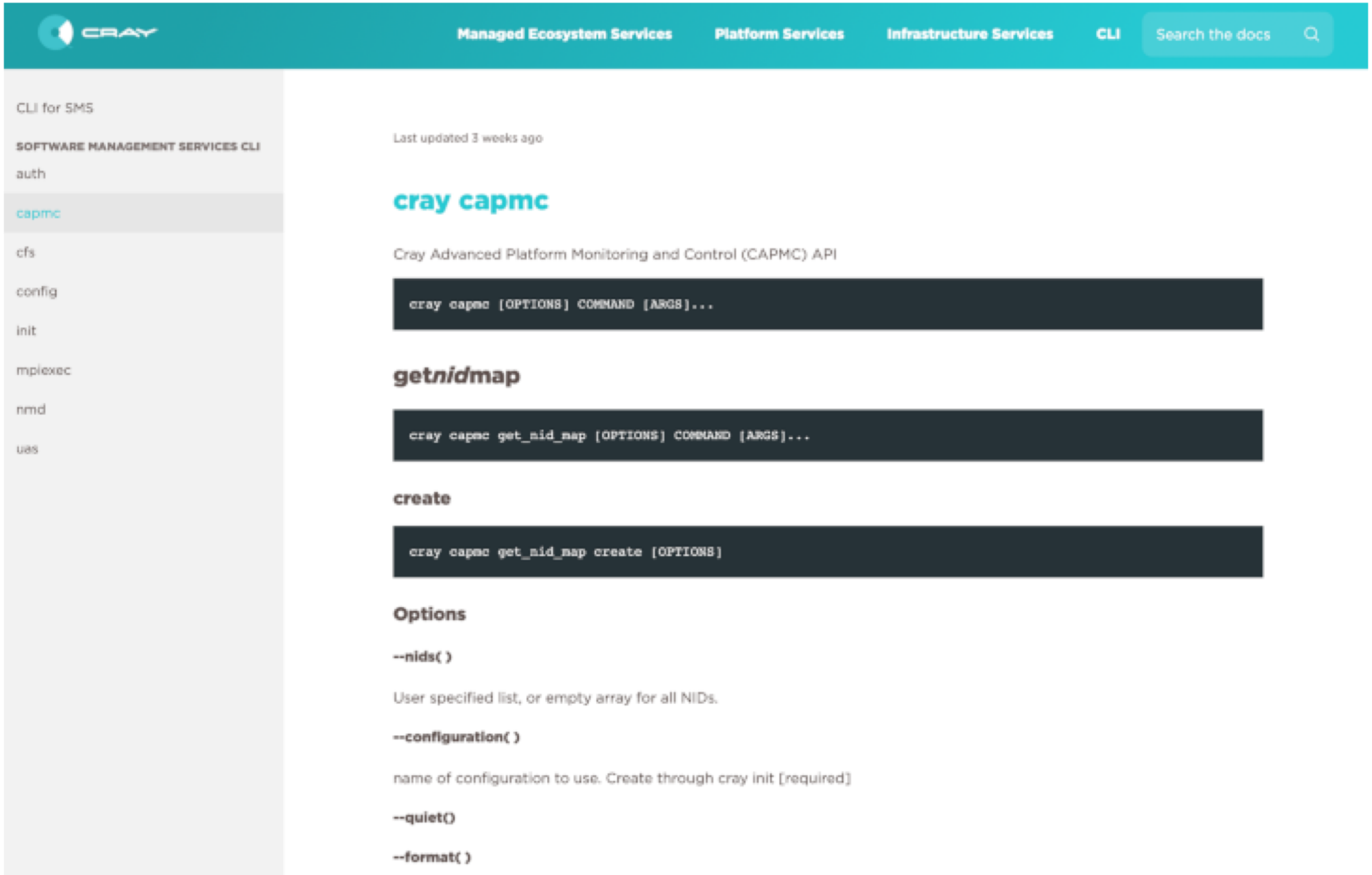
### Response samples

200 400 401 404

application/json

```
{
    "artifactID": "c3b72f49-33b0-4617-b456-70c9bc8e3edb",
  - "transport": [
        "dvs"
    ]
}
```

# CLI Documentation from REST API Specification



Last updated 3 weeks ago

## cray capmc

Cray Advanced Platform Monitoring and Control (CAPMC) API

```
cray capmc [OPTIONS] COMMAND [ARGS]...
```

### get*nid*map

```
cray capmc get_nid_map [OPTIONS] COMMAND [ARGS]...
```

### create

```
cray capmc get_nid_map create [OPTIONS]
```

### Options

**--nids( )**

User specified list, or empty array for all NIDs.

**--configuration( )**

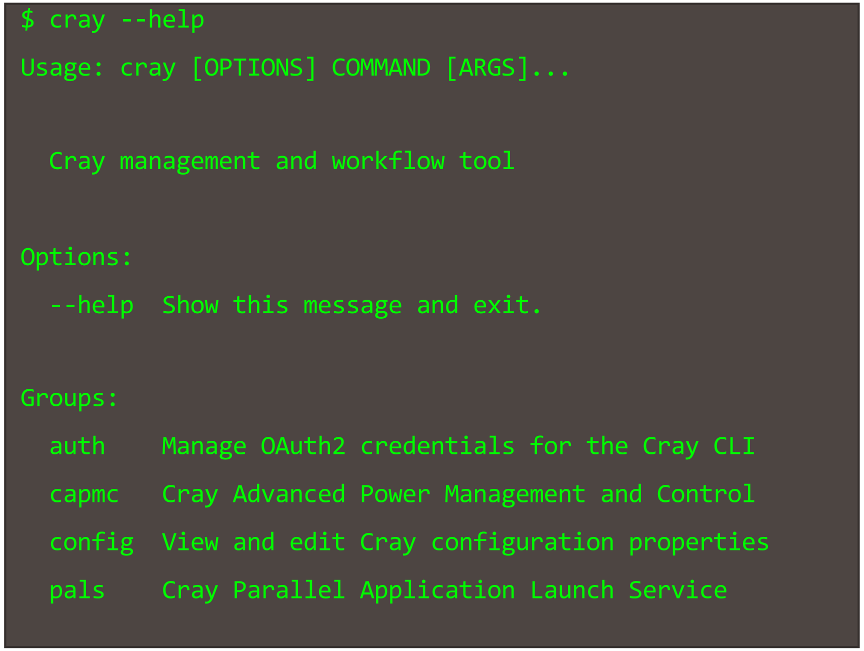
name of configuration to use. Create through cray init [required]

**--quiet()**

**--format( )**

# CLI Framework from REST API Specification

- New CLI for interacting with Shasta Management
  - Based on REST APIs and minimal code
  - Generated CLI
  - Built on a set of open standards
  - REST for all control



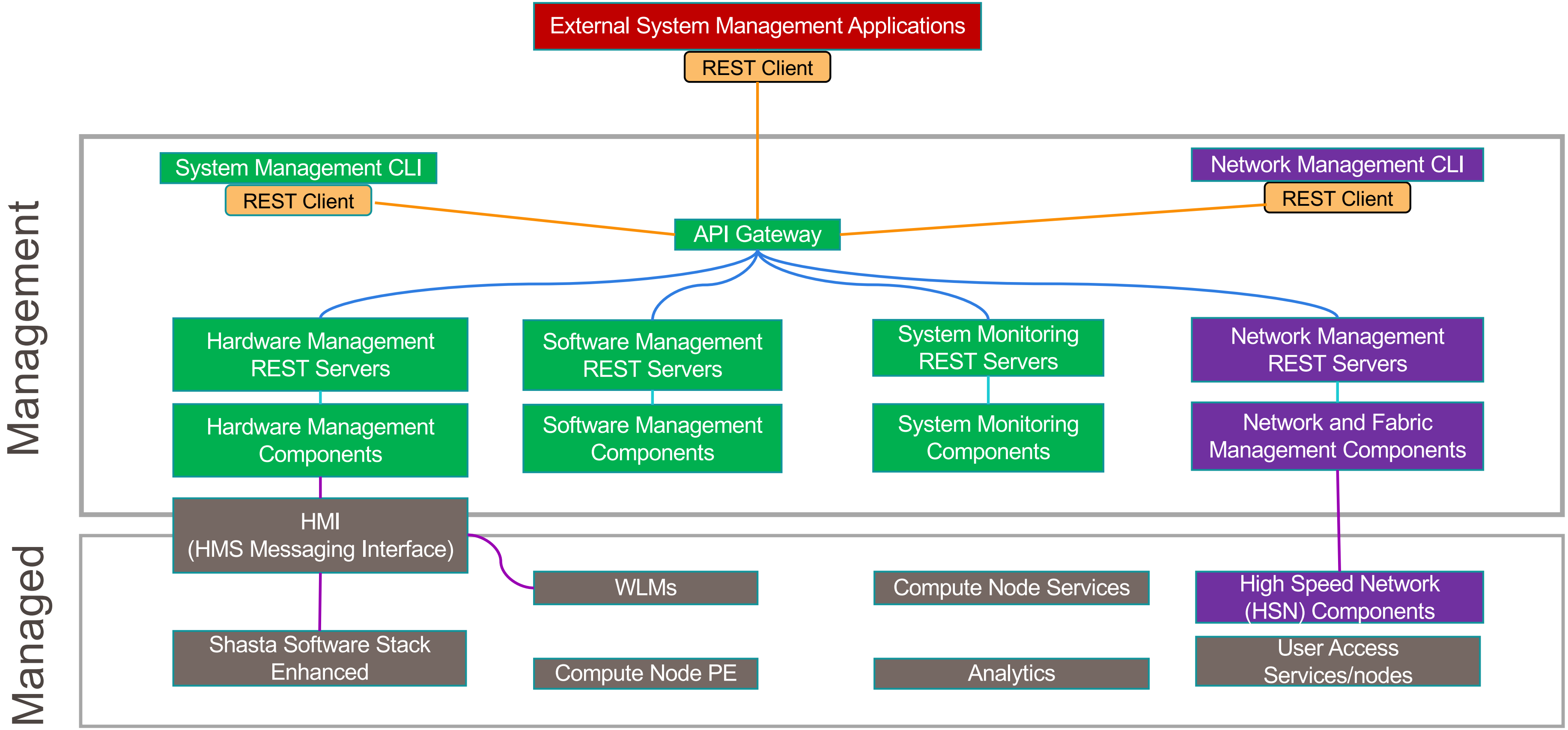
```
$ cray --help
Usage: cray [OPTIONS] COMMAND [ARGS]...

  Cray management and workflow tool

Options:
  --help  Show this message and exit.

Groups:
  auth    Manage OAuth2 credentials for the Cray CLI
  capmc   Cray Advanced Power Management and Control
  config  View and edit Cray configuration properties
  pals    Cray Parallel Application Launch Service
```

# System Management API Gateway



External System Management Applications

REST Client

System Management CLI

REST Client

Network Management CLI

REST Client

API Gateway

Hardware Management REST Servers

Software Management REST Servers

System Monitoring REST Servers

Network Management REST Servers

Hardware Management Components

Software Management Components

System Monitoring Components

Network and Fabric Management Components

HMI (HMS Messaging Interface)

WLMs

Compute Node Services

High Speed Network (HSN) Components

Shasta Software Stack Enhanced

Compute Node PE

Analytics

User Access Services/nodes

# Docker and Kubernetes

- Docker
  - Docker container runtime
  - Docker execution environment
    - Standardizes the management and interfaces
  - Configuration data passed into the container modules
    - Code that provides the networking is the same for every container
- Kubernetes
  - Manages the life cycle of containers within the service infrastructure
  - Scheduling of containers to run across a set of hosts
  - Controlling where to run a service based on requirements of the service
  - DNS and networking support between containers in a system
  - Automatic scaling and health monitoring
  - Upgrade strategies

# Image and Configuration Management and Boot Orchestration

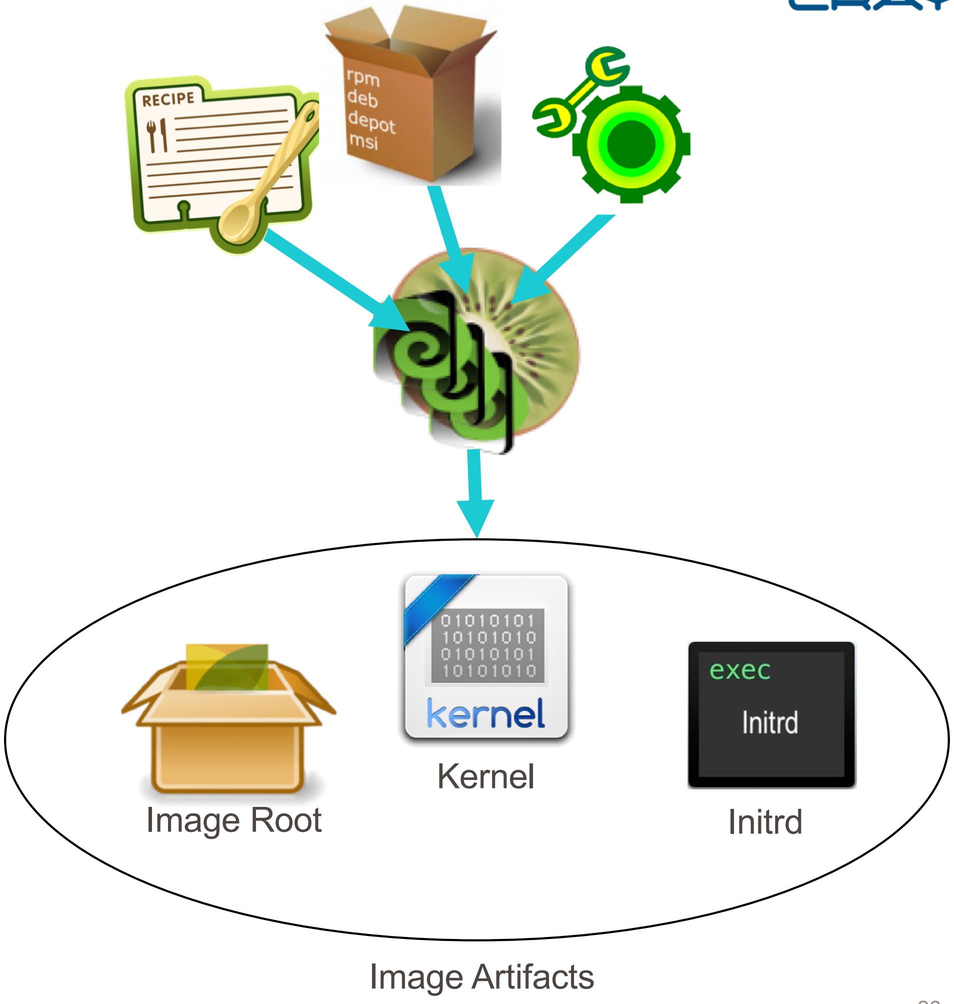
Harold Longley

# Image Management

- Prescriptive recipes create image artifacts used to boot nodes
- RESTful services for image management
  - Package Repository Service (PRS)
    - Define zypper/yum package repositories and provide the RPM content, at scale, for installing and updating software for nodes in the system
  - Image Management Service (IMS)
    - Build images from kiwi-ng recipes and customize images
    - Multiple Linux distributions supported
    - Uses kiwi-ng in a docker container
    - Uses Kubernetes Job workflow
  - Artifact Repository Service (ARS)
    - Store and retrieve artifacts (recipe, kernel, initrd, image root)
- Interact with these services using the REST API or Cray CLI
- **CUG 2019 presentation**
  - **Reimagining Image Management in the New Shasta Environment**

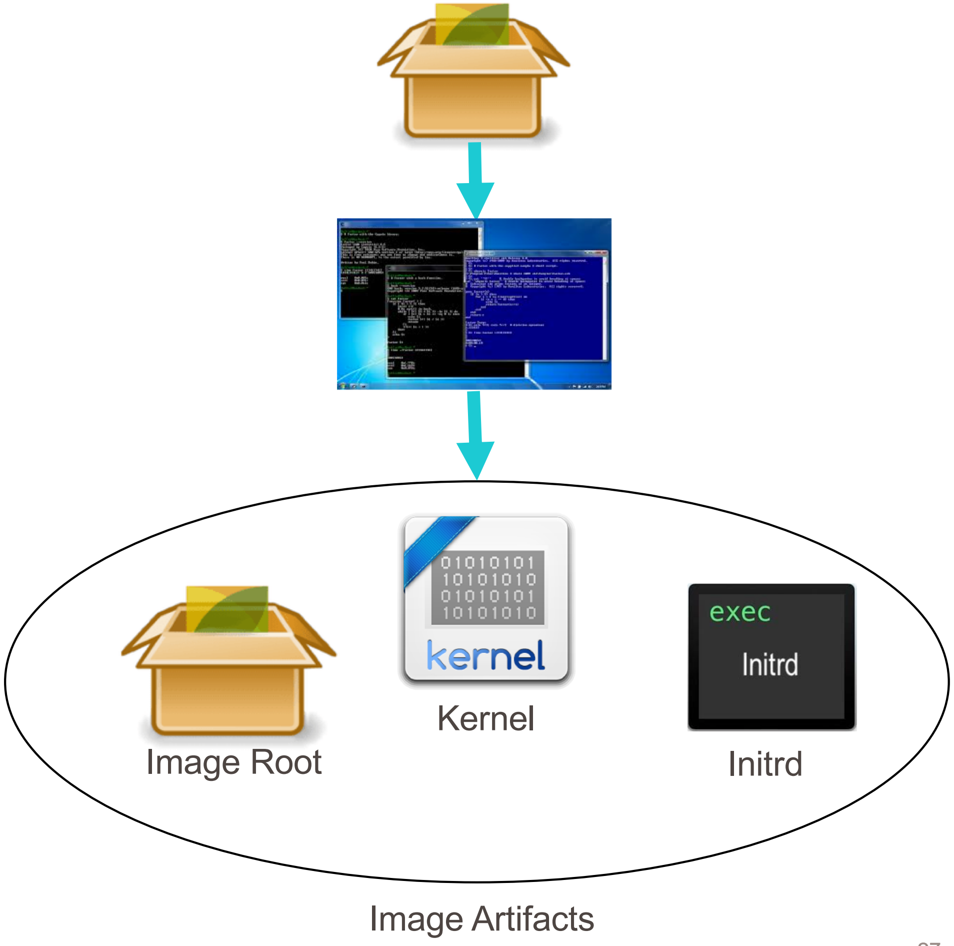
# Creating an Image

- Admin submits a “create job” to IMS
  - IMS establishes new Kubernetes pod to build image
  - Recipe downloaded from ARS and passed to kiwi-ng running in new pod
    - kiwi-ng installs RPM packages listed in recipe
    - RPMs retrieved from repos setup by the Package Repository Service (PRS)
    - After rpms installed, kiwi-ng runs scripts specified in recipe on image root
  - When kiwi-ng completes, image artifacts collected and stored in ARS

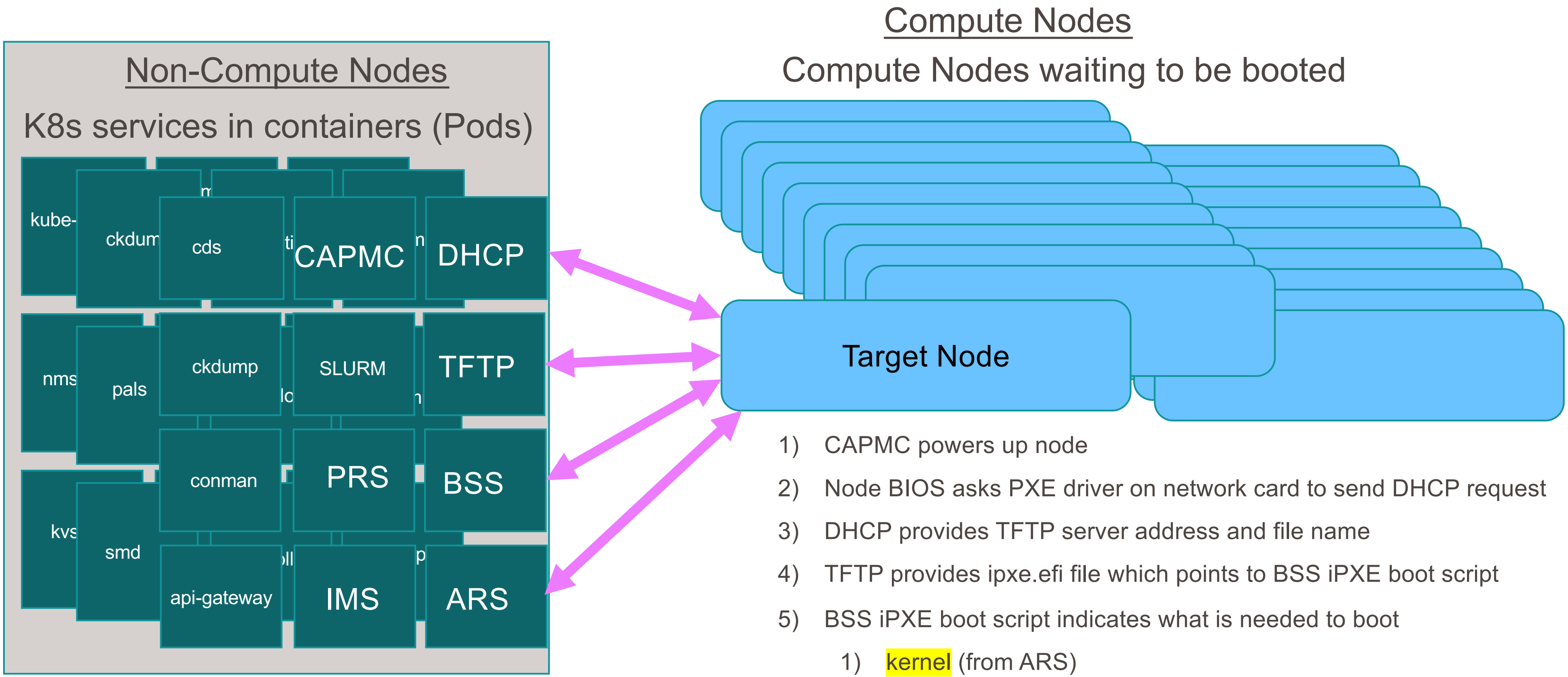


# Customizing an Image

- Admin submits a “customize job” to IMS
    - IMS establishes new Kubernetes pod to customize the image
    - Existing image is downloaded from ARS and uncompressed
    - SSH environment is established where admin can access the image root and make any required changes
    - When admin is done, image artifacts are collected and stored in ARS as new artifacts



# Boot Process Flow Needs Image Artifacts



Non-Compute Nodes

K8s services in containers (Pods)

Compute Nodes

Compute Nodes waiting to be booted

1) CAPMC powers up node
2) Node BIOS asks PXE driver on network card to send DHCP request
3) DHCP provides TFTP server address and file name
4) TFTP provides ipxe.efi file which points to BSS iPXE boot script
5) BSS iPXE boot script indicates what is needed to boot
    1) kernel (from ARS)
    2) initrd (from ARS)
    3) Kernel parameters (including the image root from ARS)

# Boot Orchestration

- Booting compute nodes requires coordination of several services
  - Hardware State Manager (HSM) – Inventory of nodes and their attributes
  - Artifact Repository (ARS) – Stores boot artifacts (kernel, initrd, image root)
  - Image Management (IMS) – Stores image record (a triple of kernel, initrd, image root)
  - Boot Script (BSS) – Stores per-node information about iPXE boot script
  - Cray Advanced Platform Management Control (CAPMC) – Powers control for node(s)
  - Hardware Message Interface (HMI) – Manages heartbeat messages and state in HSM
  - Version Control (VCS) – Stores configuration data and code with versioning
  - Configuration Framework (CFS) – Configures node(s) using configuration framework
- Boot Orchestration Service (BOS)
  - Coordinates these services
  - Tracks status

# Configuration Framework

- Provides a configuration framework for Cray and customers which integrates industry-standard configuration management tooling with Cray services
- Flexible workflow
  - pre-boot image customization
  - post-boot node personalization
  - post-boot re-configuration
- Provides dynamic inventory plugins to target Cray nodes for config
- Provides versioned config data management which enables upgrade, rollback, and test

# Configuration Tools

- What tools can be used to change and track changes?
  - Customize images or personalize nodes with Ansible
    - Ansible will be used for remote execution
      - https://docs.ansible.com/
    - Ansible "push" mode
      - https://www.ansible.com/overview/how-ansible-works
    - System administrators are familiar with Ansible concepts
      - playbooks, roles, modules, variable precedence, inventory, etc.
  - Change management and version control
    - System administrators/DevOps are familiar with git
      - https://git-scm.com/
  - Any customer provided methods to customize image or personalize nodes

# Configuration Options

- Image customization options (pre-boot)
  - IMS via manual SSH configuration environment
  - IMS via automatic Ansible plays in SSH configuration environment
- Node personalization options (post-boot)
  - Node personalization via Ansible plays on booted node
  - Node personalization via manual configuration
  - Live Update (post-boot) zypper/yum updates rpm on booted node
- Reconfiguration of node (without rebooting)
  - Same methods as node personalization
- Any customer provided methods for image customization, node personalization, or reconfiguration

# Security

Jason Rouault

# Measuring Ourselves

- Many published standards for security
  - Related to the day to day activities of hardware and software development
- Shasta platform is designed for multiple consumers, use cases, and deployment models
  - Cray cannot rely solely on a single standard to meet our objectives
- A collection of standards will be used
  - Assures we are working towards effective postures that apply to the scenarios for our platform
- These include:

ISO

CIS Center for Internet Security®

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

CSA cloud security alliance®

# Shasta Priorities

## Internal Controls

- Vulnerability scanning, static/dynamic analysis, and code signing as part of the CI/CD pipeline
- Management of OSS ingest, specifically for base OS and container images

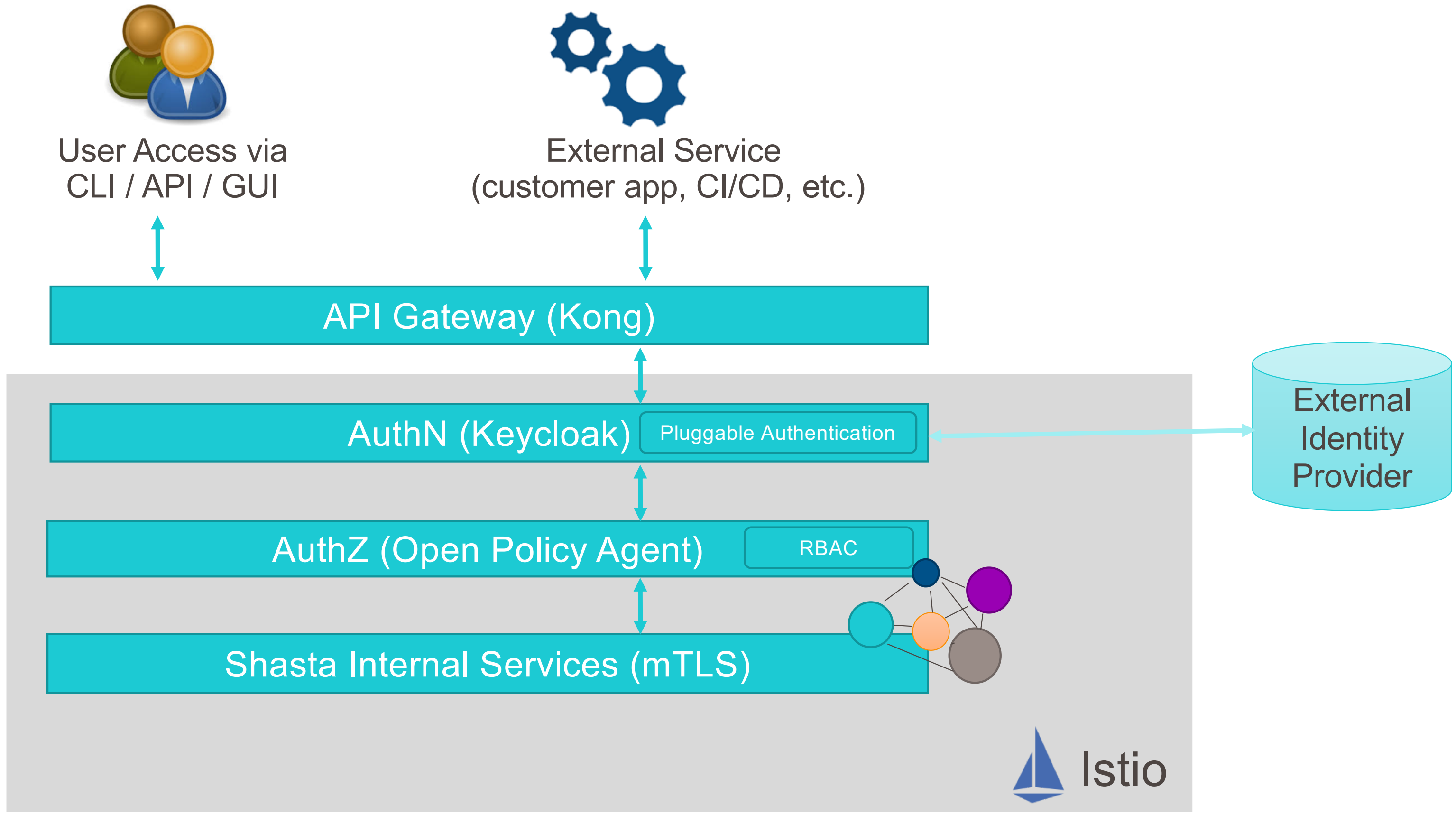
## Shasta Management Services

- Applying best practice configurations to our core platform (CIS, etc.)
- Centralized CA and tooling to allow customers to use their internal certs
- Flexible AuthN / AuthZ architecture across the management services
- Centralized credential/secret/key management for services
- Integration with customer internal processes for SIEM, audit, etc. (logging)

## Validation / On-going test

- Formal assessment (pentest, etc.) of management services and identification of security gaps for remediation on a periodic basis as change dictates
- Build security scanning into our test plan/automation

# Simplified AuthN/AuthZ Flow



User Access via CLI / API / GUI

External Service (customer app, CI/CD, etc.)

API Gateway (Kong)

AuthN (Keycloak)

Pluggable Authentication

AuthZ (Open Policy Agent)

RBAC

Shasta Internal Services (mTLS)

External Identity Provider

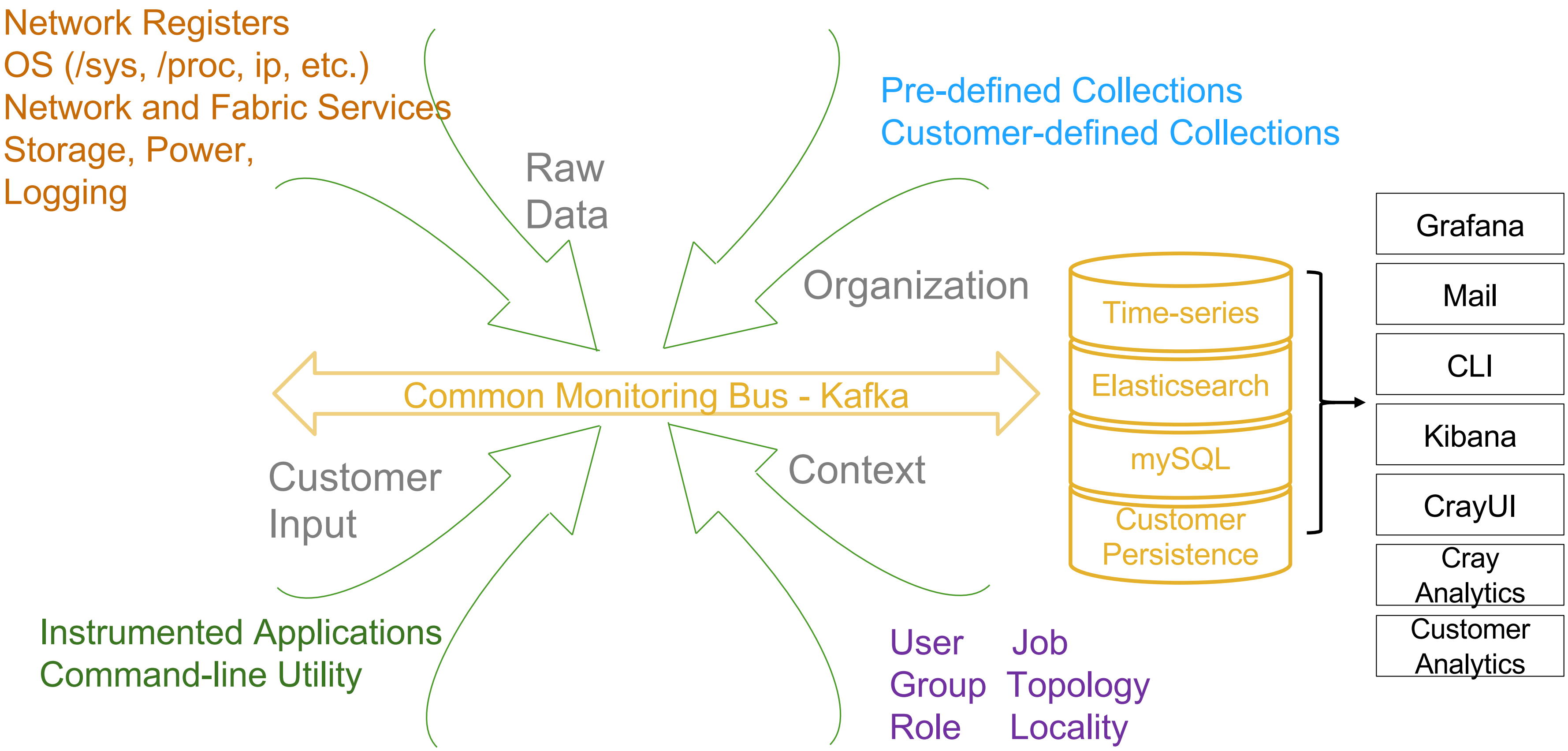
# Monitoring

Larry Kaplan

# System Monitoring Framework (SMF)

- Tightly-integrated monitoring system
- Provides detailed telemetry information from multiple subsystems:
  - Fabric
  - Network
  - Job Management
  - Storage
  - Power
  - User Applications
  - Messaging Libraries
  - Operating Systems
- Incorporates the context necessary to understand telemetry data
- Feeds into a common message bus, persistence, and UI infrastructure
- SMF is based upon Cray View for ClusterStor, but expanded to cover the entire system

# System Monitoring Framework Flows



Network Registers
OS (/sys, /proc, ip, etc.)
Network and Fabric Services
Storage, Power,
Logging

Raw Data

Pre-defined Collections
Customer-defined Collections

Organization

Common Monitoring Bus - Kafka

Customer Input

Context

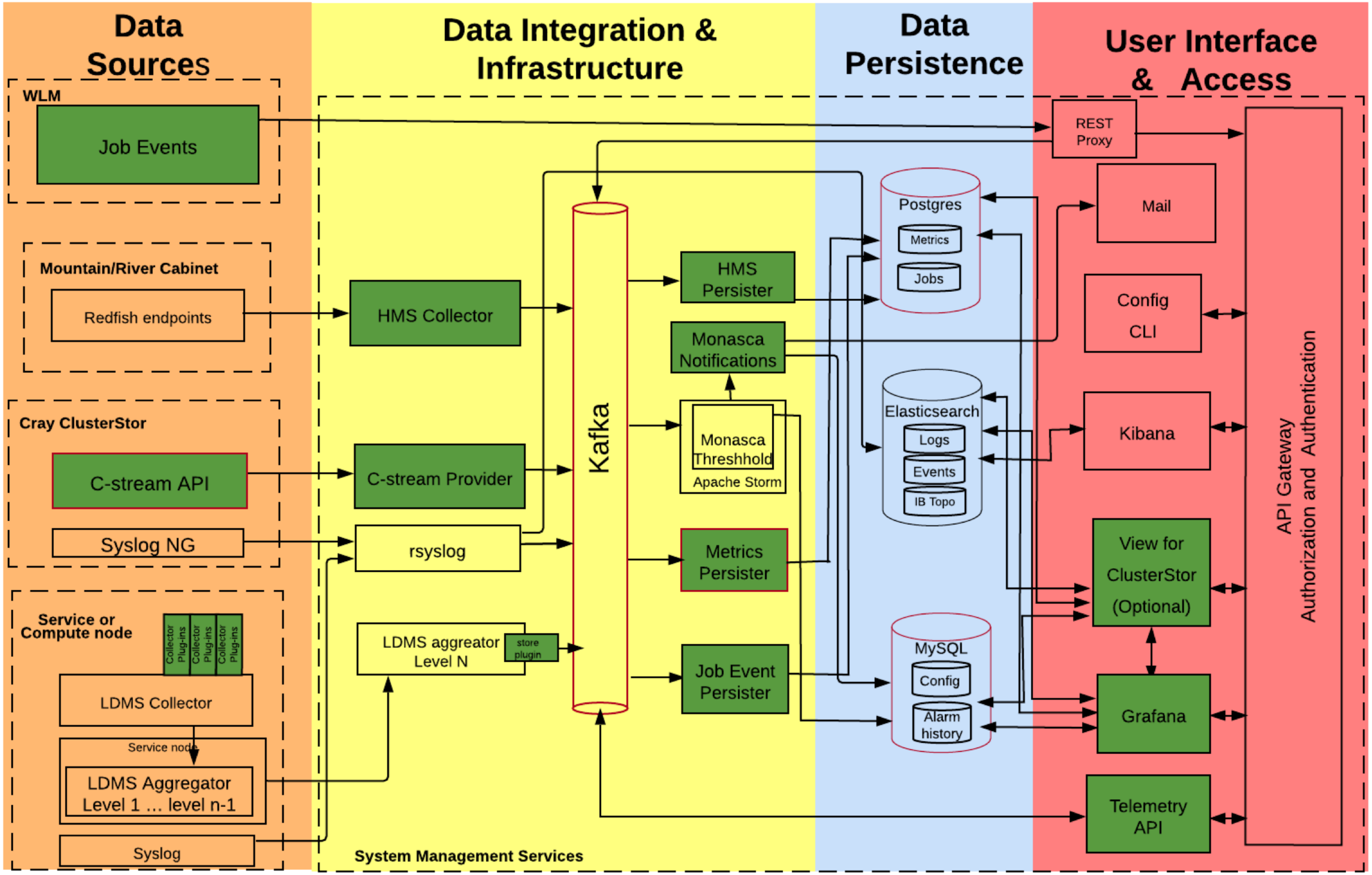
Instrumented Applications
Command-line Utility

User Job
Group Topology
Role Locality

Time-series
Elasticsearch
mySQL
Customer Persistence

Grafana
Mail
CLI
Kibana
CrayUI
Cray Analytics
Customer Analytics

# System Monitoring Framework



# RAS Events and Telemetry

- RAS related information is available in the system telemetry streams/topics
  - Includes logs, log analysis, change notifications, and system events
- As much as practical, this information is used to enable automated handling of many scenarios
  - Examples include responding to machine checks and other node health events, network failures, and some forms of failover handling
- All events and logs use system coordinated time
  - PTP on the HSN and NTP on the mgmt networks – synced to each other
- APIs are available for both streaming and historical access
  - History provided by SMS limited to 30 days

# Network Management

Larry Kaplan

# Fabric vs. Network

**Fabric** is:

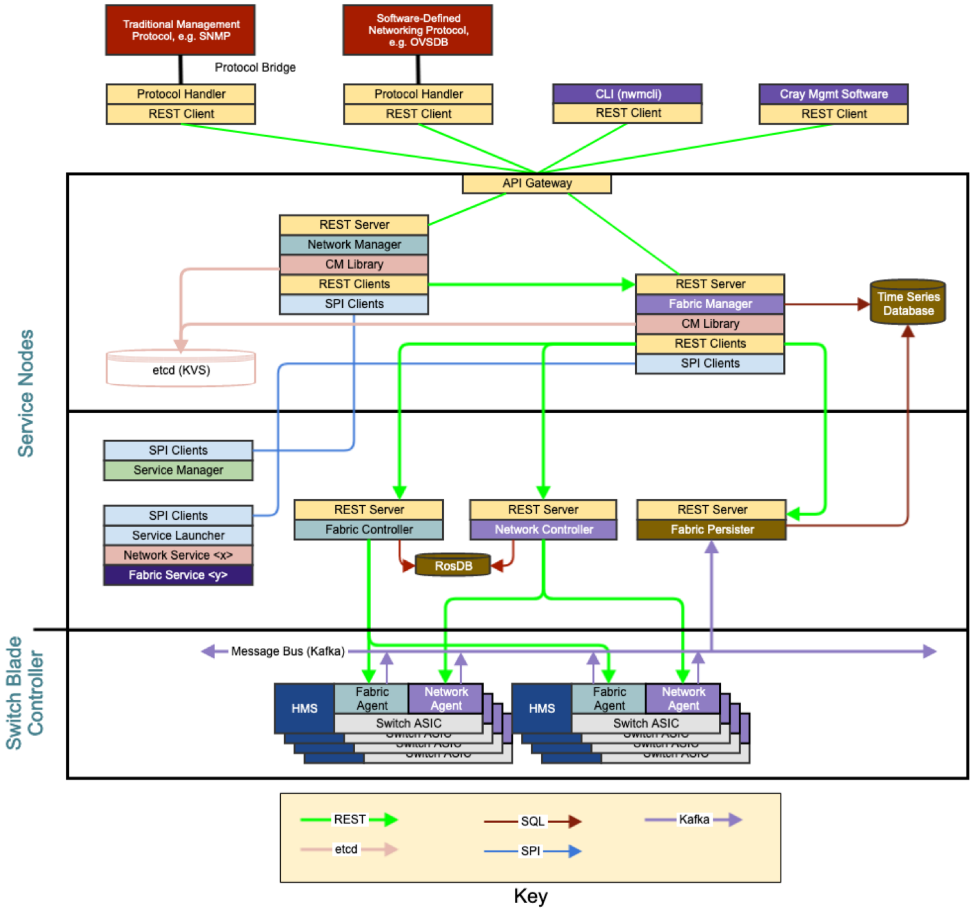
- The infrastructure, including:
  - Switches
  - Links (cables or traces)
  - Ports (and attached NICs/MACs)
- Common settings
  - Traffic Classes
- Pool of Common Resources
  - E.g. VLANs

**Networks** are:

- Logical constructs on top of the Fabric
- Ethernet configuration
  - IP Address Ranges
  - DHCP Settings
  - DNS Settings
- Services
  - Protocol support
  - Scalability

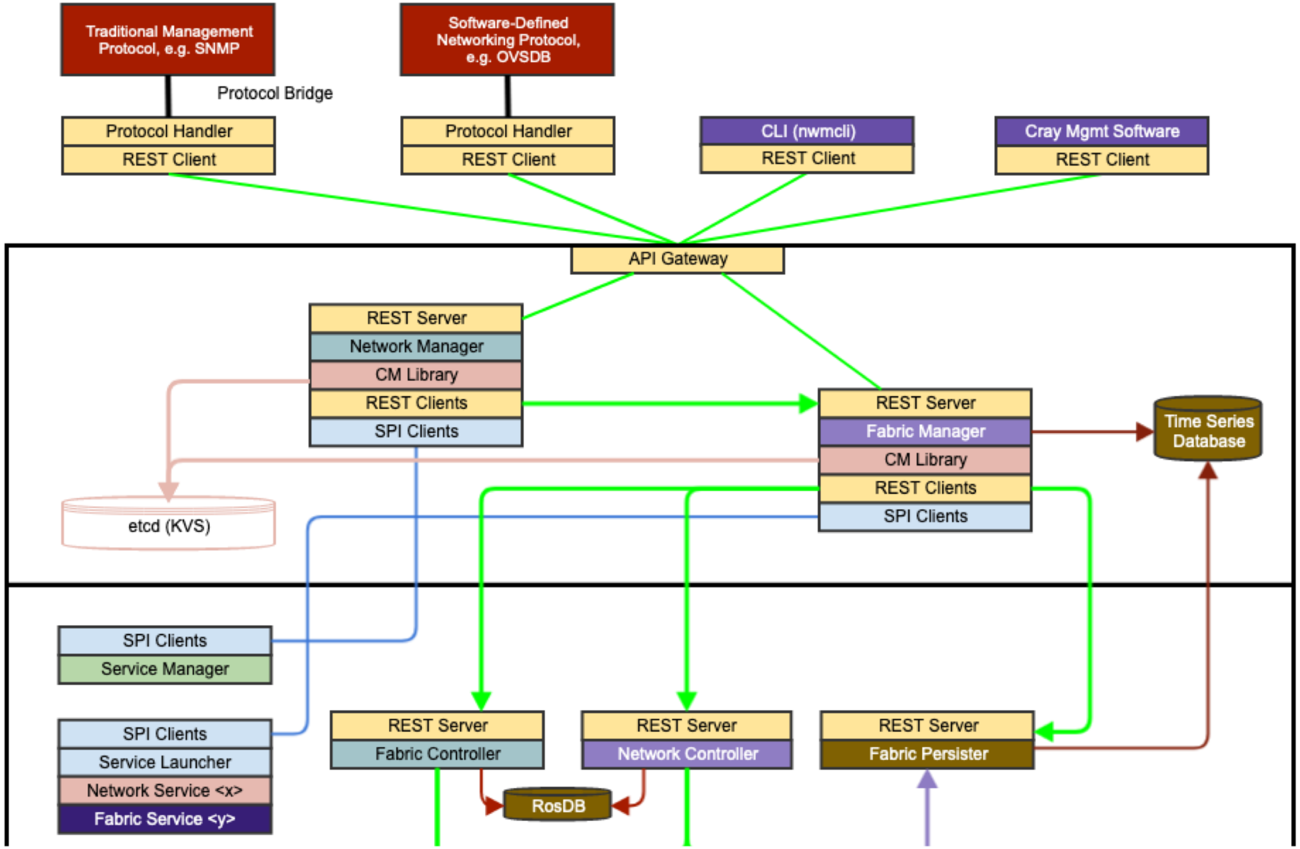
# Fabric and Network Administrators

- Fabric and Network Management Stack are modular
    - Specific components support Fabric and Network activities
- Stack is aligned with Cray System Management's Role-Based Access Controls (RBAC)
    - Fabric and Network admins own specific responsibilities



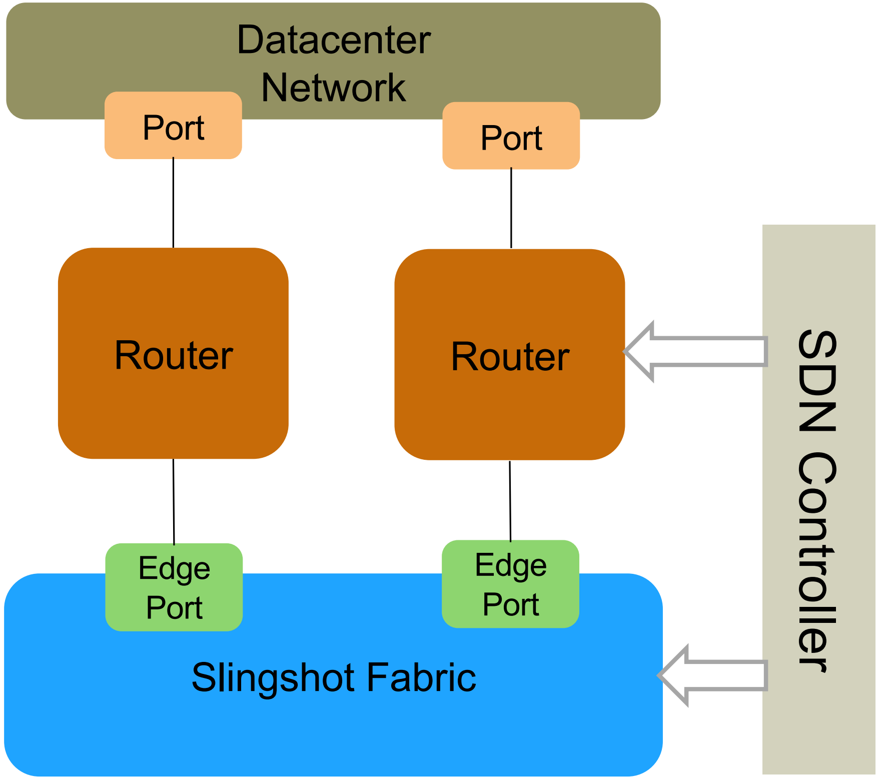
# Fabric and Network Management Access

- All command and control traffic is through REST APIs
  - Published but proprietary
- Standard network management protocols are supported through protocol bridges



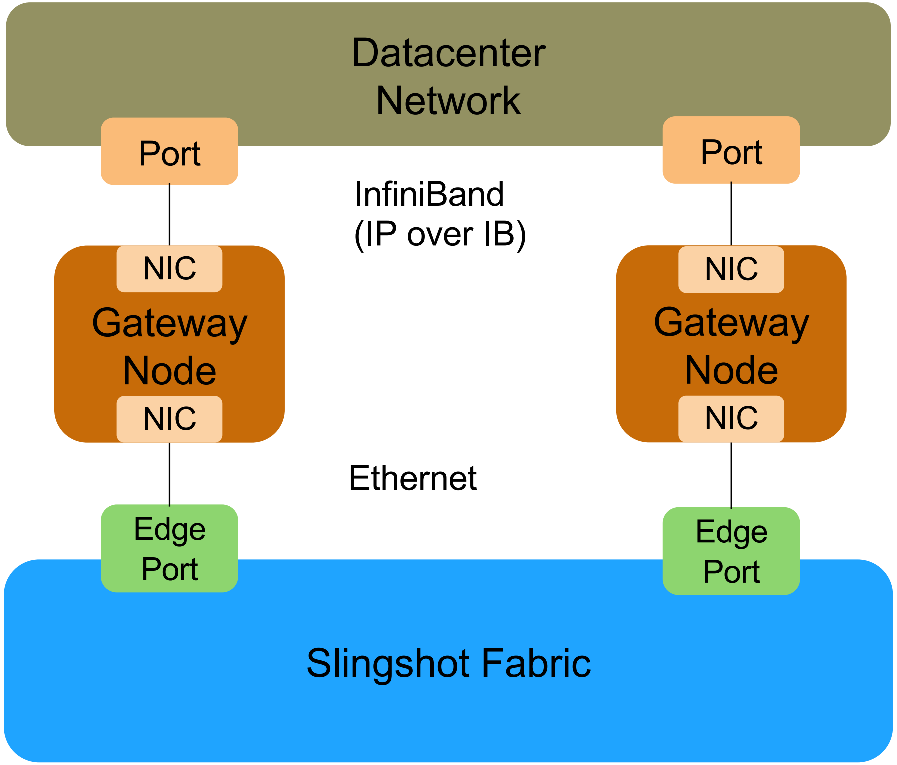
# High Throughput 3rd Party Router

- **Qualified by Cray**
- **Managed by SDN Controller**
  - Simplified controller based on OVS protocol to configure interfaces, NAT, and Firewall rules
  - Support one of standard controllers: OpenDaylight or RYU



# Bridging Networks

- **Routing service can provide bridging function**
  - Ethernet to IPoIB (or other non-ethernet physical transport)



# BREAK

QUESTIONS?

# Linux (Managed Ecosystem)

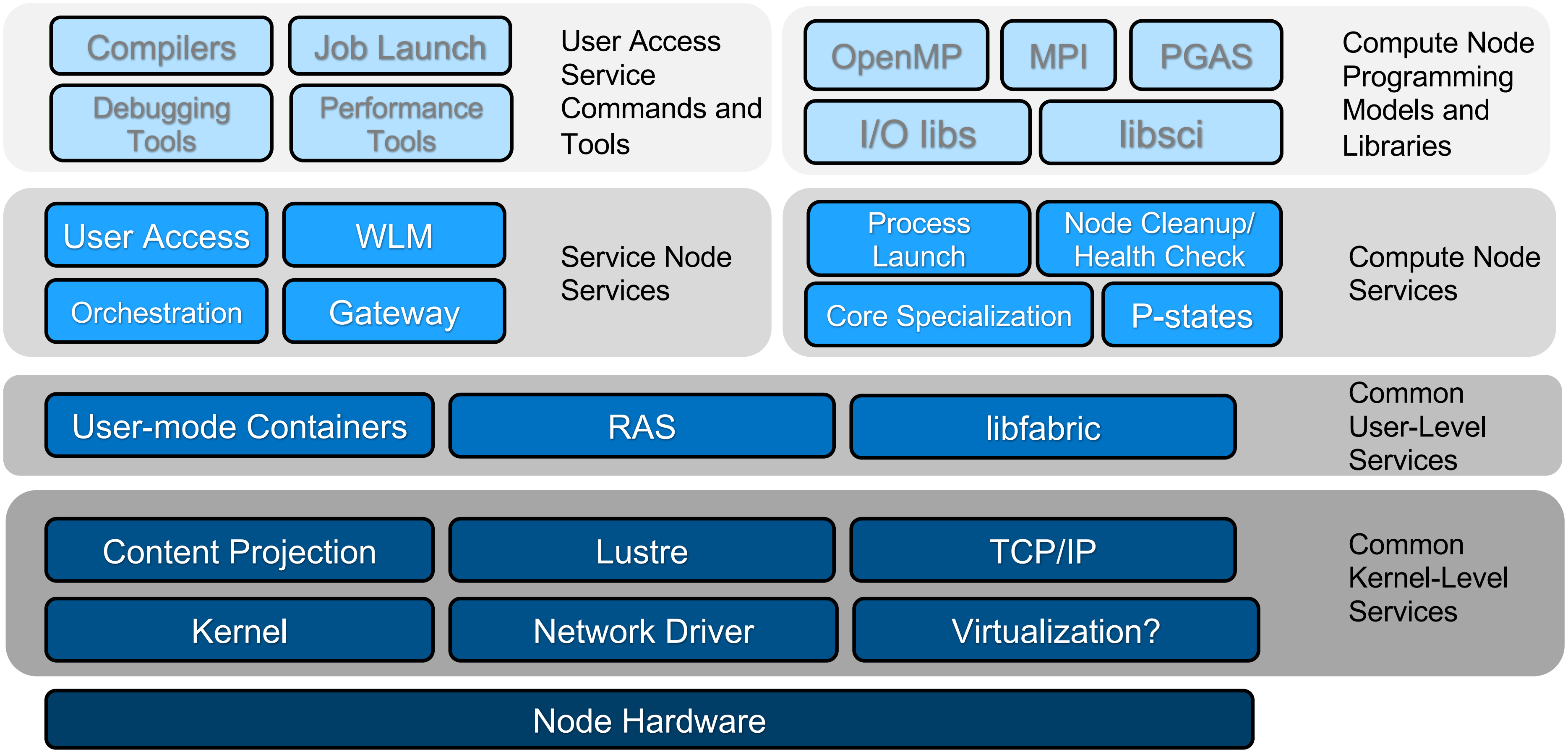
Larry Kaplan

# Shasta Linux Software Stack

- **Flexibility for Cray to meet customer needs**
  - Fully optimized Linux for high-end HPC, based on SLES
    - Corresponds to current CLE software stack
  - Provision for standard Linux distros with Cray network software
    - Possibilities include SLES, CentOS, Red Hat
    - Pricing and support model TBD
  - Also considering a middle ground with some Cray enhancements

- **Individual Cray Software Components**
  - Distro agnostic
  - Less intrusive, better interoperability with site software stack
  - Enables faster response time for updates

# CLE Software Components



Compilers

Job Launch

Debugging Tools

Performance Tools

User Access Service Commands and Tools

OpenMP

MPI

PGAS

I/O libs

libsci

Compute Node Programming Models and Libraries

User Access

WLM

Orchestration

Gateway

Service Node Services

Process Launch

Node Cleanup/ Health Check

Core Specialization

P-states

Compute Node Services

User-mode Containers

RAS

libfabric

Common User-Level Services

Content Projection

Lustre

TCP/IP

Kernel

Network Driver

Virtualization?

Common Kernel-Level Services

Node Hardware

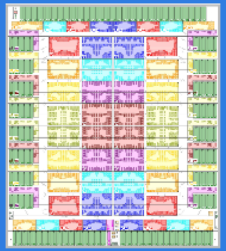
# Slingshot

Larry Kaplan

# Slingshot Components

*Rosetta*

- Multiple QoS levels
- Aggressive adaptive routing
- Advanced congestion control
- Very low average *and* tail latency



64 ports x 200 Gbps

*NIC*

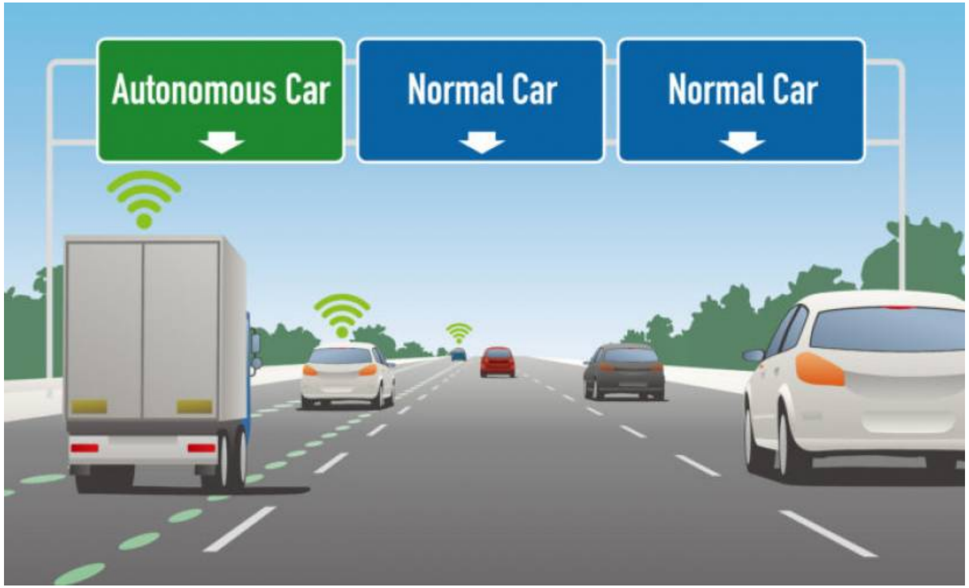
- Cray MPI stack
- Ethernet functionality
- RDMA offload
- ~50M MPI messages/sec

# Traffic Classification

- Application traffic association by packet marking
  - Packet header field carries a Differentiated Services Code Point (DSCP)
    - DSCP field of IP header
    - PCP field in VLAN tag of Ethernet header
  - Code Point indicates preferred network behavior
    - Not guaranteed
    - Aggregation is possible
- Network-wide, predefined classification mappings
  - Specifies network properties and characteristic
    - Manipulates underlying hardware resources
  - Defines Code Point association

# Rosetta Traffic Classes

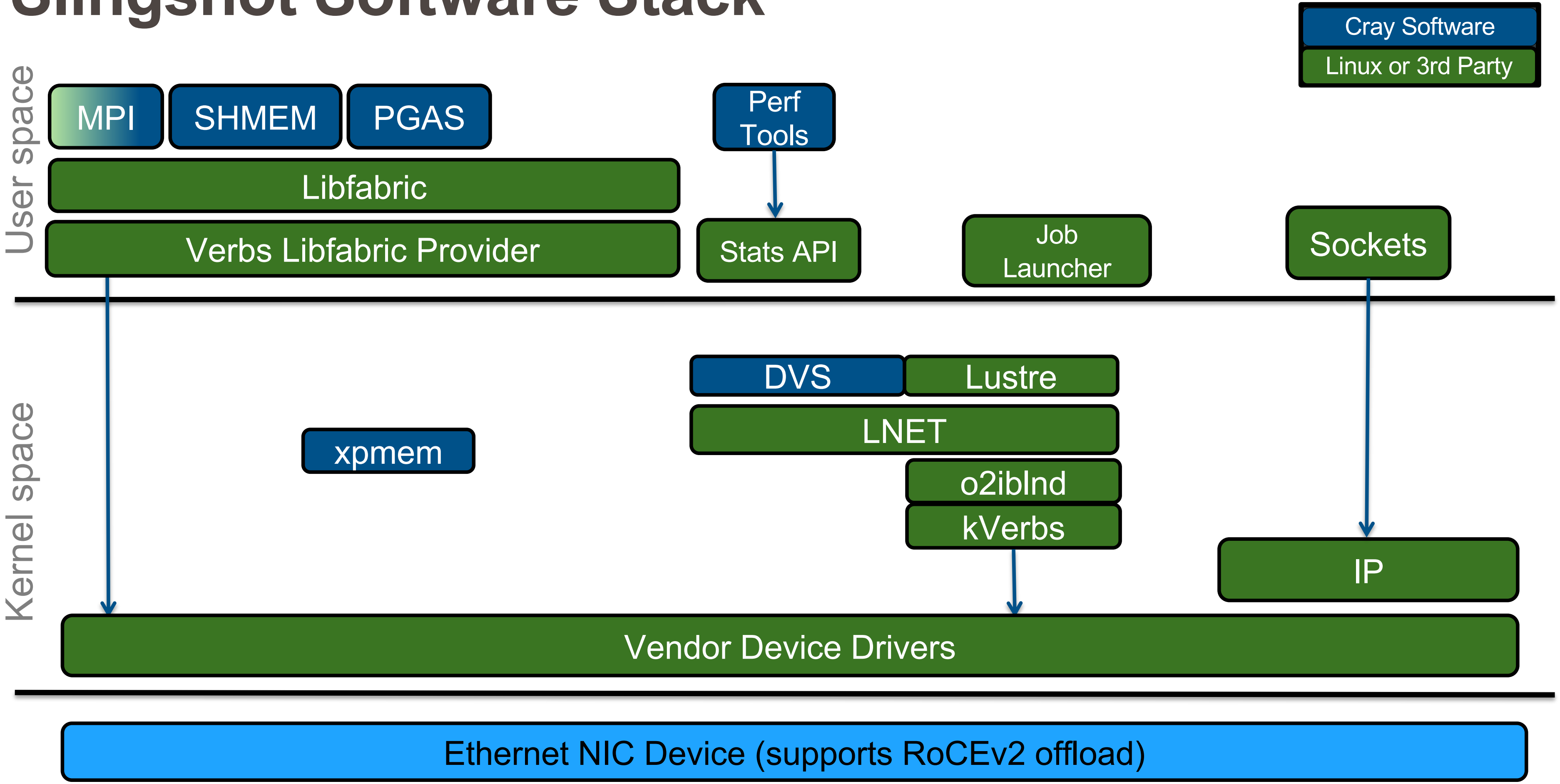
- Example Traffic Classes
  - Priority – low latency queries, barriers, etc.
  - I/O – tuned for isolating large high-bandwidth transfers
  - Dedicated – reserve bandwidth to minimize variations between runs of the same job
  - Best effort – default for non-critical applications
  - Scavenger – background, lossy traffic, monitoring
- Establish 'best practice'
  - Default settings for each site or system
  - Expect configuration varies between systems



# Accessing Traffic Classes

- Differentiated Services Code Points (DSCP) provide TC mechanism
- Allows both standard DSCP and the HPC classes to be used where appropriate
- Cray also will propose libfabric based access
- Jobs granted access to TCs via WLM
  - WLM gets info on what is configured from network manager
  - Executes access policies determined by site
- Applications can then use them in several ways
  - Single TC – for the entire application (possibly dedicated)
  - Two TCs – one for low bandwidth/low latency (priority), another for all other traffic
  - Multiple TCs – fuller control, potentially on a per transfer basis
  - Note that ordering is NOT maintained across TCs

# Slingshot Software Stack



# Verbs libfabric Provider

- Cray is moving to libfabric for our low-level communication interface (LLCI)
  - Community created and supported
  - Geared towards network clients rather than network hardware
- Provider needed to be both performant and scalable
- Existing ethernet providers have challenges, Verbs-based providers seemed best
  - Others had more severe scaling issues (such as sockets-based provider)
- Choose between:
  - OFI-RXM layered on Verbs Messaging Endpoints
  - OFI-RXD layered on Verbs Datagram Endpoints
  - a native RDM implementation within the Verbs core provider
- Selected #1 based on evaluation of performance, ease of enhancement, and maintainability
  - Implemented enhanced eXtended Reliable Connection (XRC) for scalability
- Results are being committed back to the community

# User Environment

Matt Haines

# User Access on Containers?

## Advantages 😀

- Load balanced and HA access
- Different OS per user
- Custom images per user
- Easy to test new OS/images
- Resource limits by role/profile
- Process space isolation
- Cloud-like “cattle” model for throw-away and replace usage
- Hardware affinity by role/profile
- "User-access-to-go"

# User Access on Containers (cont) ?

Thanks for the feedback

## Advantages

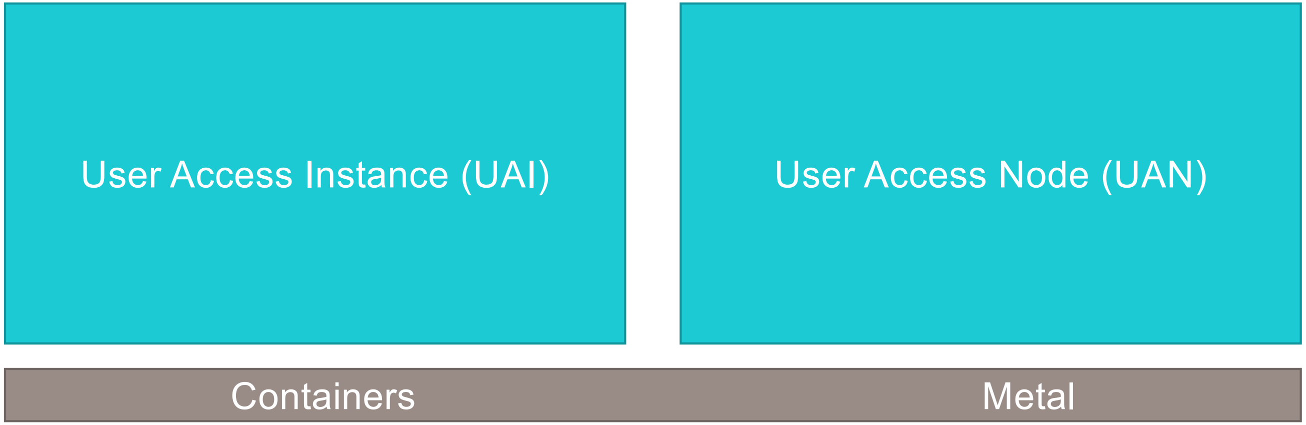
- Load balanced and HA access
- Different OS per user
- Custom images per user
- Easy to test new OS/images
- Resource limits by role/profile
- Process space isolation
- Cloud-like “cattle” model for throw-away and replace usage
- Hardware affinity by role/profile
- "User-access-to-go"

## Challenges

- Access to special hardware features
- Swap space support
- Interesting deployments
- Specialized security & access controls
- Sharing instances between users raises security concerns
- Admin access for debugging and support
- Kubernetes networking

# User Access Implementation Space (Internal)

Goal to support both!



User Access Instance (UAI)

User Access Node (UAN)

Containers

Metal

# User Access and Login

## UAI

- Create UAI
  - Can have timeout or be persistent
- ssh to UAI IP address
  - Nonstandard port (for now)
- Native Kubernetes support for load balancing UAIs across nodes

## UAN

- ssh to UAN IP address
  - Standard port (22)
- No native load balancing
  - LB can be added by customer for a single IP across multiple UANs

# User Access and Job Launch

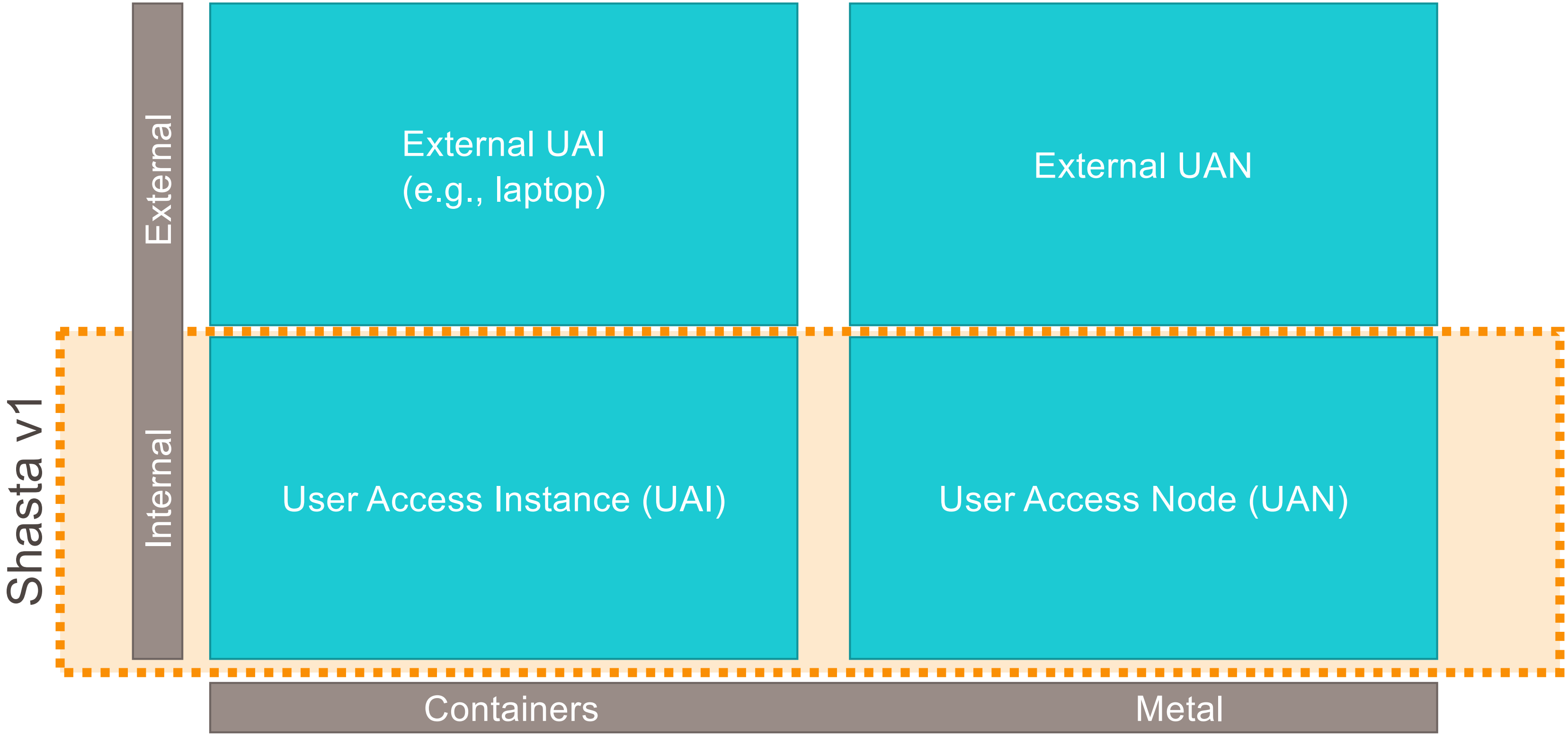
## UAI

- WLM clients are installed local to the user access instance (UAI)
  - Commands executed as WLM vendor intended, not proxied
  - No escaping or special handling of the environment
- Access to Lustre mount for job scripts, binaries, and results
  - All UAIs default to `/lus` mount
- Networking handled by Kubernetes

## UAN

- WLM clients are installed local to the user access node (UAN)
  - Commands executed as WLM vendor intended, not proxied
  - No escaping or special handling of the environment
- Access to Lustre mount for job scripts, binaries, and results
  - All UANs default to /lus mount
- Networking handled by base OS

# User Access Implementation Space



External UAI
(e.g., laptop)

External UAN

User Access Instance (UAI)

User Access Node (UAN)

# Workload Management

## SLURM & PBS PRO

- Actively working with SchedMD and Altair on Shasta check-out and new APIs
- Cray providing integration through a new set of services and APIs
- Both WLMs supported for FCS
- Other WLMs can also use the same APIs

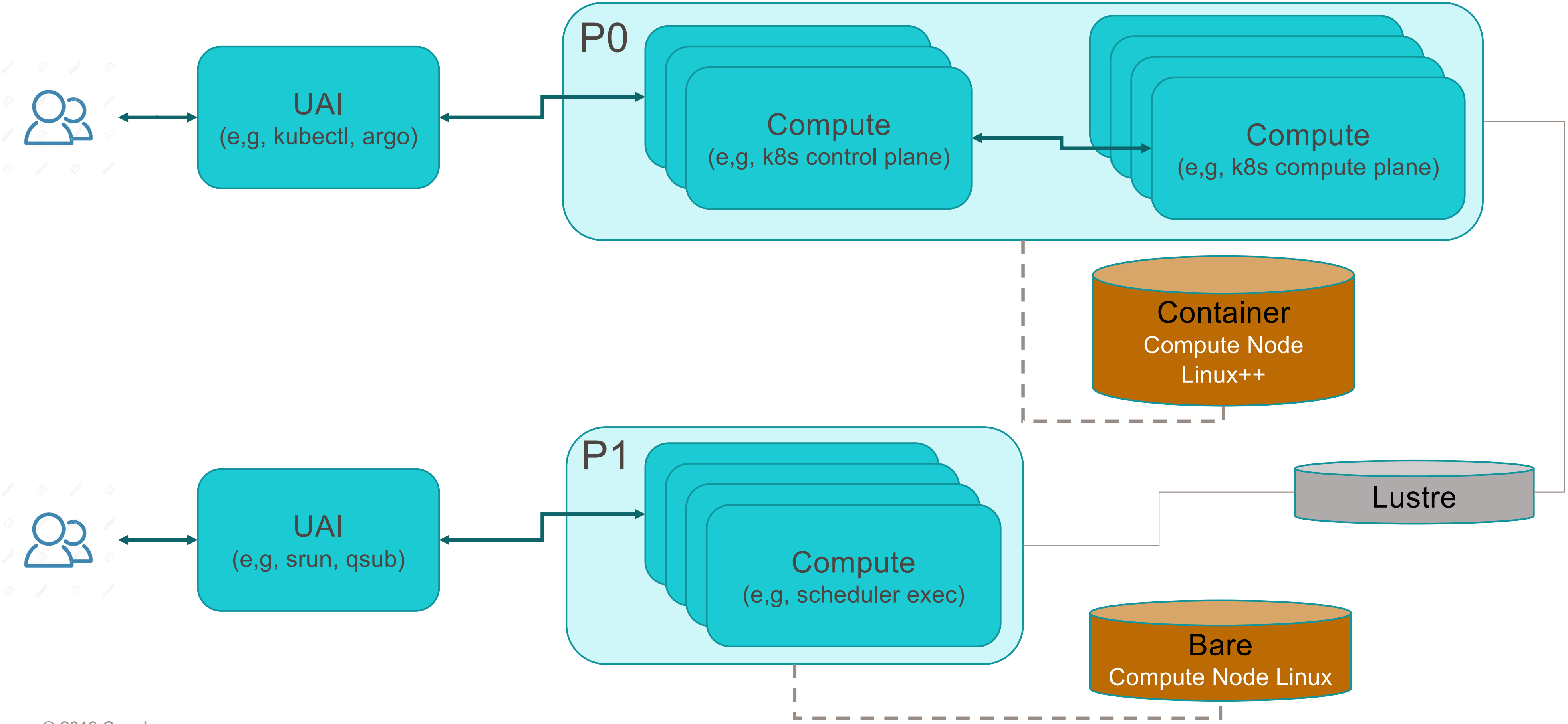
## CRAY WLM/RM SERVICES

- PALS – Parallel application launch service
- JACS – Job and application configuration services
- HATS – Health analysis test service
- JARS – Job and application reporting service

# Containers for Users

Jonathan “Bill” Sparks

# Orchestration & Scheduling



# User Interactions

- Orchestration/containers

```
[root@ncn-005 ~]# kubectl get nodes
NAME      STATUS ROLES       AGE VERSION
nid000001 Ready  master,node 11h v1.13.3
nid000002 Ready  master,node 11h v1.13.3
nid000003 Ready  master,node 11h v1.13.3
nid000004 Ready  node        10h v1.13.3
```

- Batch

```
[root@ncn-005 ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
workq*       up   infinite      4   idle nid[000001-000004]
```

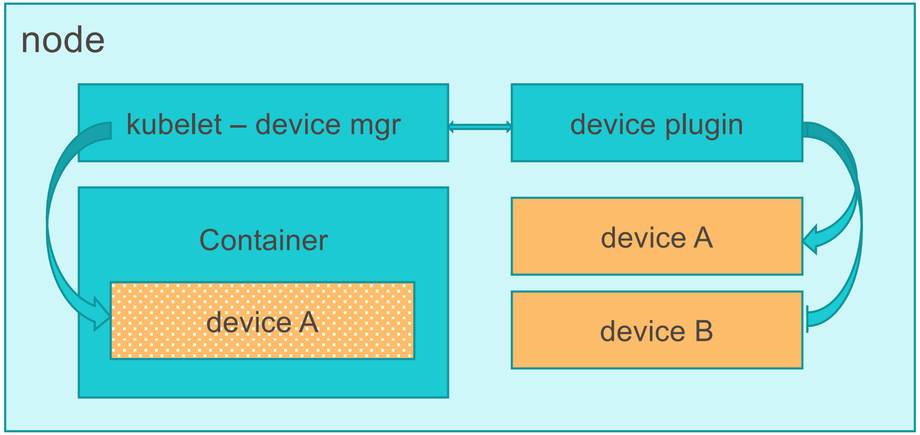
# Shasta Container Strategy

- **HPC Containers**
  - Cray compute OS is container runtime agnostic
    - Support for Docker and Singularity
    - Bring your own container runtime environment via CMS/IMS
  - Runtime choice depends on orchestration/scheduler
    - Docker for use with Kubernetes – AI/ML/cloud-native
      - Direct Docker engine access will be protected via authentication
    - Singularity for use with Workload Manager (PBS, Slurm, ...)
- **Communications for MPI have several options**
  - MPICH ABI compatible applications can use Cray MPI
  - Libfabric enabled MPI can use Cray libfabric (late binding)
  - Verbs based MPI can use standard Linux Verbs over Ethernet

# Kubernetes Host Resource Access

- Network (RDMA): Network device plugin
- Accelerators (GPU): Device plugin
- Benefits:
  - Framework provides monitoring and management of plugin
  - Device plugins execute privileged, whereas the user containers run unprivileged

- Plugin advertise devices to kubelet
- k8s allocate plugin device with mgr.
- Kubelet exports device to container



https://github.com/kubernetes/community/blob/master/contributors/design-proposals/resource-management/device-plugin.md

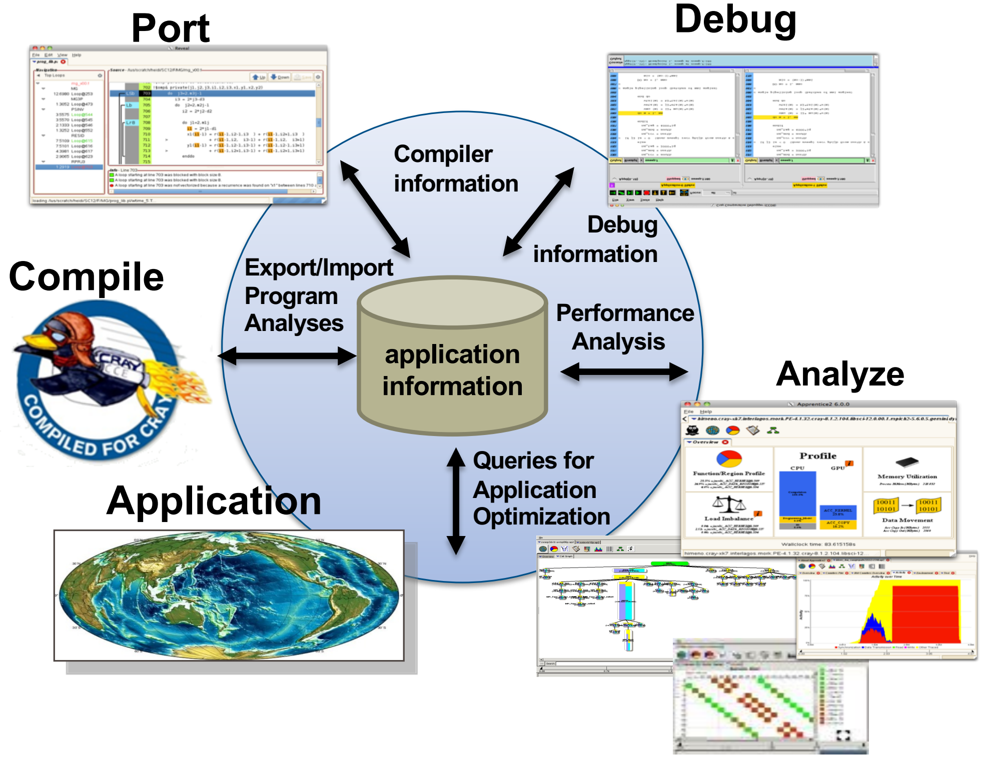
# Programming Environment

Larry Kaplan

# The Cray Programming Environment Mission

- The Cray PE is designed to **drive maximum computing performance** while **focusing on programmability and portability**

- Provide the best environment to develop, debug, analyze, and optimize applications for **production supercomputing** with **tightly coupled compilers, libraries, and tools**
  - Address issues of scale and complexity of HPC systems
  - Intuitive behavior and best performance with the least amount of effort
  - Target **ease of use** with extended **functionality** and increased **automation**
  - Close **interaction with users**



# The Cray Compiling Environment on Shasta

- Cray technology **designed for real scientific applications**, not just for benchmarks

- Fully integrated **heterogeneous optimization** capability

- Focus on standards compliance for **application portability** and **investment protection**

C++ 17

Fortran 2008

OpenMP 4.5

C11

UPC 1.3

# Cray Programming Environment for Shasta

- **Fortran, C, and C++ compilers**
  - **OpenMP directives to drive compiler optimization**
  - Compiler optimizations for multi-core processors and SIMD/vectors

- **Cray Reveal**
  - **Scoping analysis** tool to assist user in understanding their code and taking full advantage of both software and hardware in the system

- **Cray Performance Measurement and Analysis toolkit**
  - Single tool for CPU performance analysis with statistics for the whole application

- **Parallel debugger support with Totalview, DDT, and Cray CCDB**

- **Auto-tuned Scientific Libraries support**
  - Getting performance from the system … **no assembly required**

# Shasta Development Environment

## Programming Languages

### Languages

- Fortran
- C
- C++
- Chapel
- Python
- R

## Programming Models

### Distributed Memory

- Cray MPI<br>SHMEM

### Shared Memory / GPU

- OpenMP

### PGAS & Global View

- UPC<br>Fortran coarrays<br>Coarray C++<br>Chapel

## Programming Environments

### ProgEnv-

- Cray Compiling Environment PrgEnv-cray
- GNU PrgEnv-gnu
- 3rd Party compilers (AMD, Intel, PGI, etc) PrgEnv-???

## Optimized Libraries

### Scientific Libraries

- LAPACK
- ScaLAPACK
- BLAS
- Iterative Refinement Toolkit
- FFTW

### I/O Libraries

- NetCDF
- HDF5

## Tools

### Environment setup

- Modules / Lmod
- Tool Enablement (supports Spack, CMake, etc.)

### Performance Analysis

- CrayPAT
- Cray Apprentice$^2$

### Porting

- Reveal
- CCDB

## Tools (continued)

### Debuggers

- gdb4hpc
- TotalView
- DDT

### Debugging Support

- Abnormal Termination Processing (ATP)
- STAT
- Valgrind4hpc

## Analytics / AI **

### AI Toolboxes

- Cray Urika AI - Analytics
- Chapel AI

### DL Frameworks

- Cray PE DL Scalability Plugin

■ Cray Developed
🟪 Cray added value to 3rd party
🟦 3rd party packaging
⬛ Licensed ISV SW

** Not PE dependent

# Further Details

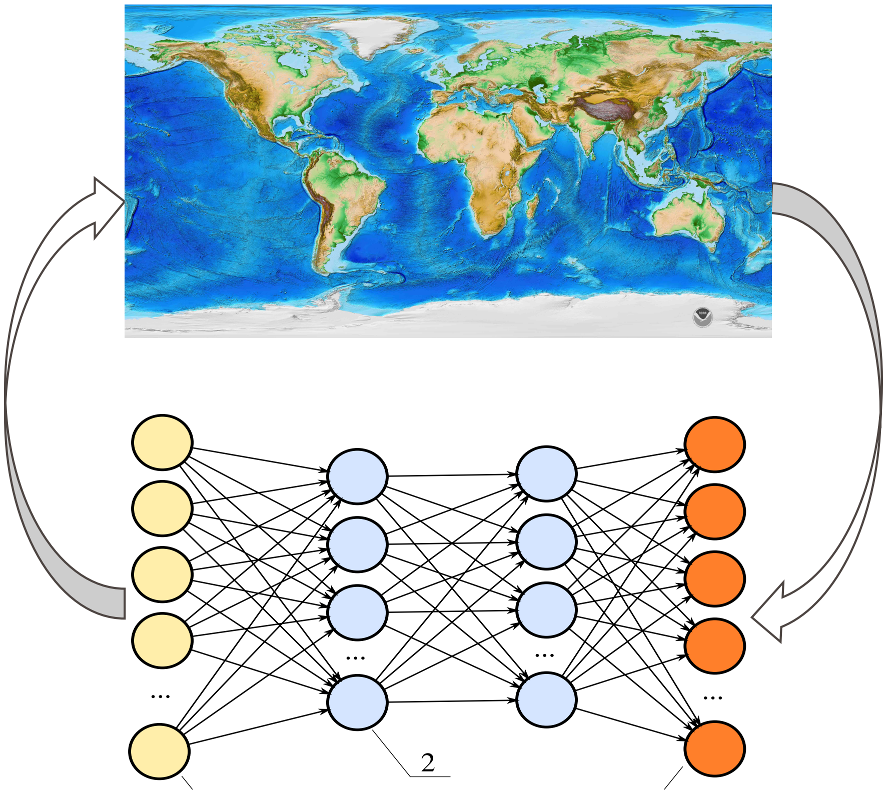
- Programming Environments, Applications, and Documentation (PEAD)
  - Special Interest Group (SIG) meeting
  - Today 4:40pm-6pm  BoF 3B

# Shasta Analytics

Larry Kaplan

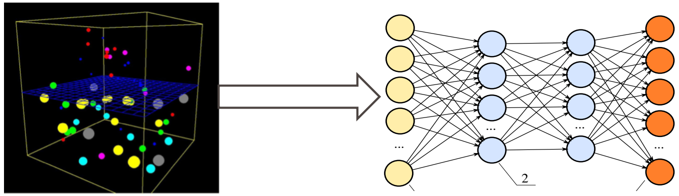
# Convergence of AI, Analytics, and Simulation

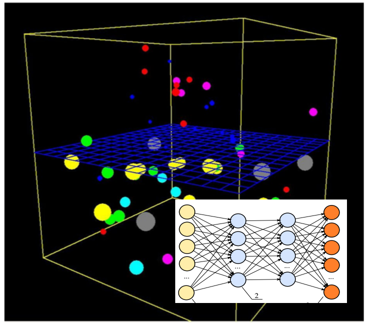
- **How can AI help simulation, and how can simulation help AI?**
  - Trained models to replace expensive computations with "good enough" approximations
  - Training models on simulated results
  - Machine learning to choose optimal simulation parameters ("tuning knobs")
- **Leverage full capabilities of hardware**
  - Increase utilization
  - Reduce data movement
  - Simplify workflows

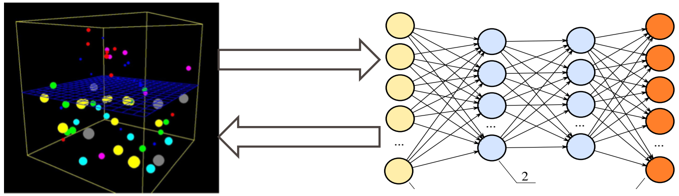


# Cray Vision: Tools and Expertise

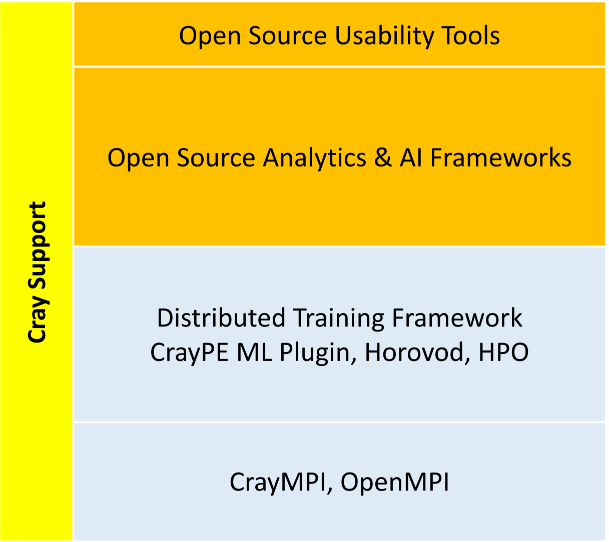
- **Flexible tools to enable creation and exploration of converged workflows**
  - Learning outside
  - Learning inside
  - Learning on-the-side
- **Interoperates with popular open source ML/DL and Analytics frameworks and libraries**







# Urika – Shasta



Cray Support

Open Source Usability Tools

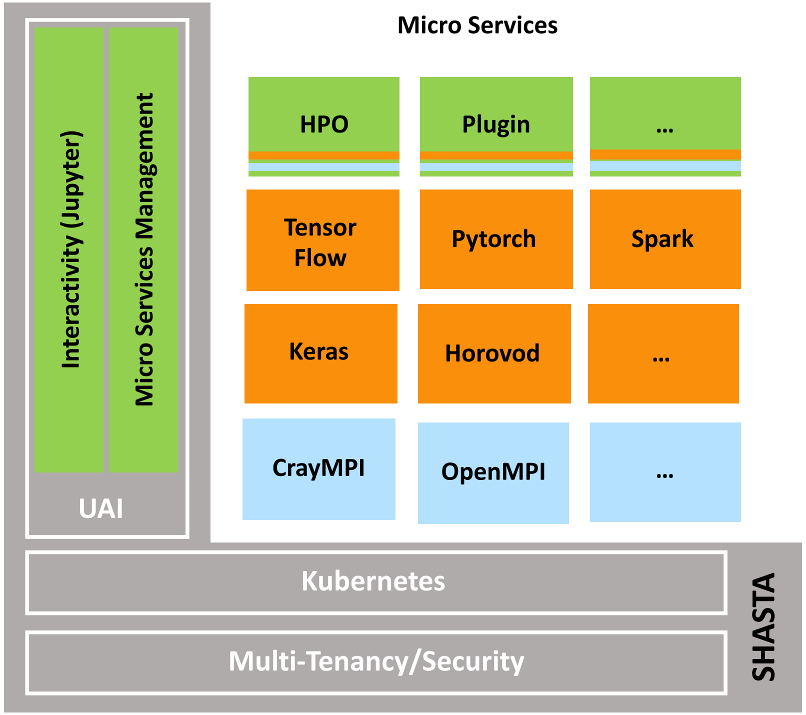
Open Source Analytics & AI Frameworks

Distributed Training Framework
CrayPE ML Plugin, Horovod, HPO

CrayMPI, OpenMPI

Urika (XC, CS)

5 Course Dinner, Prix Fixe



Interactivity (Jupyter)

Micro Services Management

UAI

Micro Services

HPO

Plugin

...

Tensor Flow

Pytorch

Spark

Keras

Horovod

...

CrayMPI

OpenMPI

...

Kubernetes

Multi-Tenancy/Security

SHASTA

Urika – Shasta

A la Carte

# Urika-Shasta – Overview

- Based on community frameworks
- Cray additions leverage these frameworks
- Frameworks, libraries, and other components containerized as micro-services
  - Micro-services management eases deployment
- Interactivity via Jupyter
- Leverage Shasta features
  - Image Management
  - Containers and Kubernetes
  - Security
  - Development pipelines

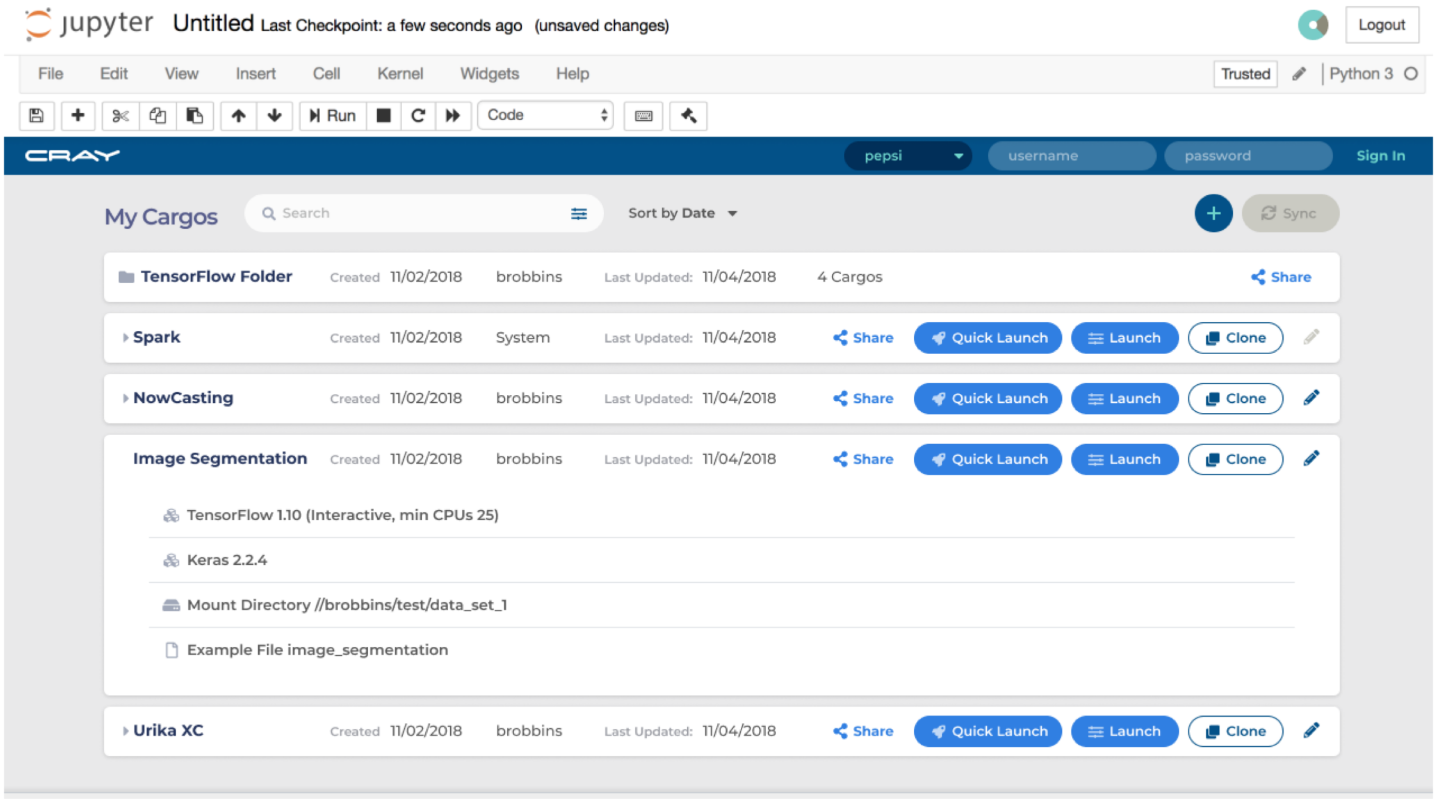
# Urika-Shasta – Frameworks & Libraries

- Community
  - TensorFlow
  - Keras
  - PyTorch
  - TensorBoard
  - Jupyter Notebooks
  - Alchemist
  - Python
  - R
  - DASK
  - pbdR

- Cray
  - Distributed Training Plugin
  - Hyper Parameter Optimization (HPO)
  - MPI
  - Integration

- Others can also be added!

# Urika-Shasta – Dynamic Environments



| | | | | | | |
|---|---|---|---|---|---|---|
| TensorFlow Folder | Created 11/02/2018 | brobbins | Last Updated: 11/04/2018 | 4 Cargos | Share | |
| Spark | Created 11/02/2018 | System | Last Updated: 11/04/2018 | | Share | Quick Launch · Launch · Clone |
| NowCasting | Created 11/02/2018 | brobbins | Last Updated: 11/04/2018 | | Share | Quick Launch · Launch · Clone |
| Image Segmentation | Created 11/02/2018 | brobbins | Last Updated: 11/04/2018 | | Share | Quick Launch · Launch · Clone |
| Urika XC | Created 11/02/2018 | brobbins | Last Updated: 11/04/2018 | | Share | Quick Launch · Launch · Clone |

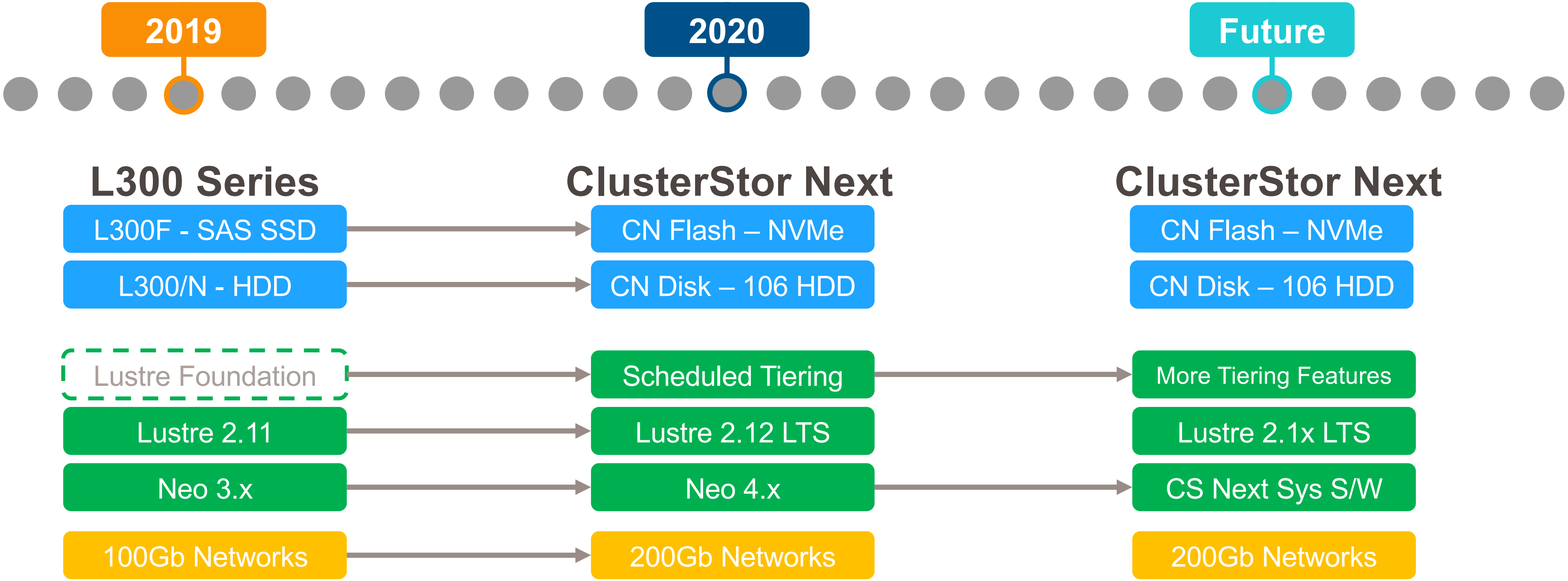
Image Segmentation:

- TensorFlow 1.10 (Interactive, min CPUs 25)
- Keras 2.2.4
- Mount Directory //brobbins/test/data_set_1
- Example File image_segmentation

# Storage

John Fragalla

# ClusterStor Product Transitions



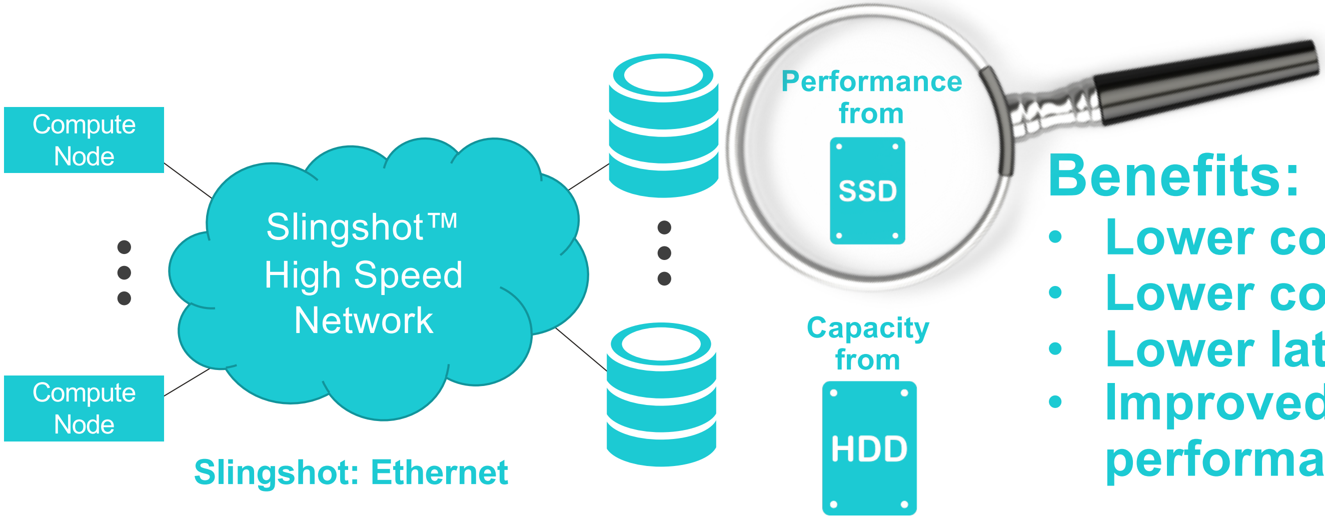
| 2019 | 2020 | Future |
|---|---|---|
| L300 Series | ClusterStor Next | ClusterStor Next |
| L300F - SAS SSD | CN Flash – NVMe | CN Flash – NVMe |
| L300/N - HDD | CN Disk – 106 HDD | CN Disk – 106 HDD |
| Lustre Foundation | Scheduled Tiering | More Tiering Features |
| Lustre 2.11 | Lustre 2.12 LTS | Lustre 2.1x LTS |
| Neo 3.x | Neo 4.x | CS Next Sys S/W |
| 100Gb Networks | 200Gb Networks | 200Gb Networks |

# ClusterStor Next – Flexibility

| | Extreme Performance | Hybrid Flexibility | HDD Performance | HDD Capacity |
|---|---|---|---|---|
| SSD Performance (write) | 60 GB/s | 60 GB/s | | |
| SSD Usable Capacity (3.2 TB) | 55.3 TB | 55.3 TB | | |
| HDD Performance | | 15 GB/s | 30 GB/s | 30 GB/s |
| HDD Usable Capacity (14TB) | | 1.07 PB | 2.14 PB | 4.27 PB |
| Network ports | 6 x 200 Gbps | 4 x 200 Gbps | 2 x 200 Gbps | 2 x 200 Gbps |
| Height Rack Units | 2 | 6 | 10 | 18 |
| Compared 2 x L300N (10RU) | 15 times faster | 15 times (flash), 0.7 (HDD) | 50% faster | 50% faster |

# ClusterStor Next – Directly on Slingshot™ HSN

Shasta™/ “ClusterStor Next”



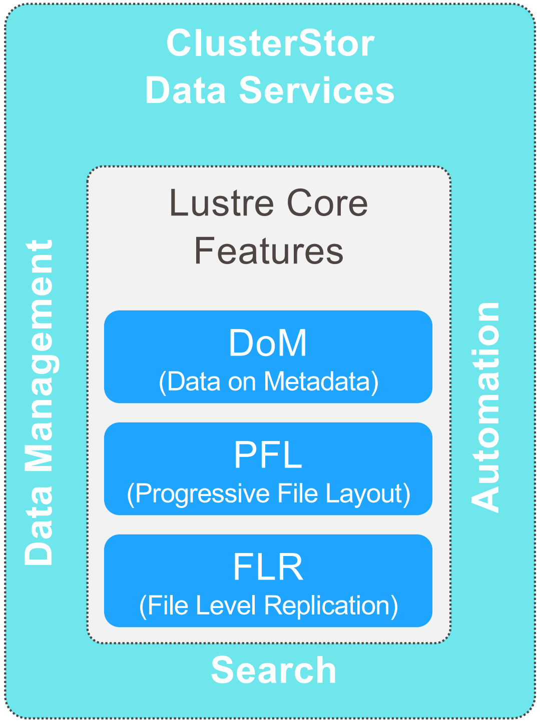
Slingshot: Ethernet

## Benefits:

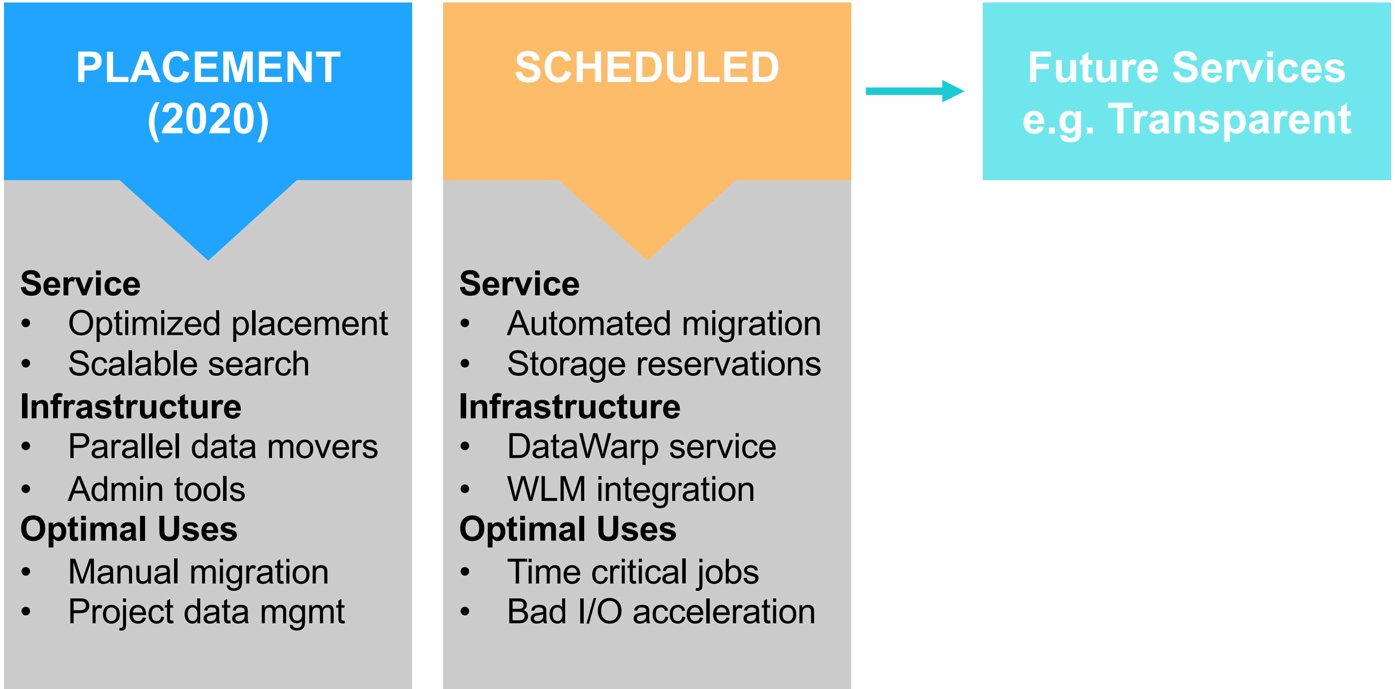
- Lower cost
- Lower complexity
- Lower latency
- Improved small I/O performance

# ClusterStor Data Services

- Cohesiveness
  - Reduce complexity for customers
- Scale
  - Move beyond scale limits of Robinhood
  - Target petascale to exascale
- Integration
  - Direct integration with ClusterStor
  - Built-in management and monitoring
  - Workflow integration through workload managers



# Data Services Progression



**PLACEMENT (2020)**

**Service**
- Optimized placement
- Scalable search

**Infrastructure**
- Parallel data movers
- Admin tools

**Optimal Uses**
- Manual migration
- Project data mgmt

**SCHEDULED**

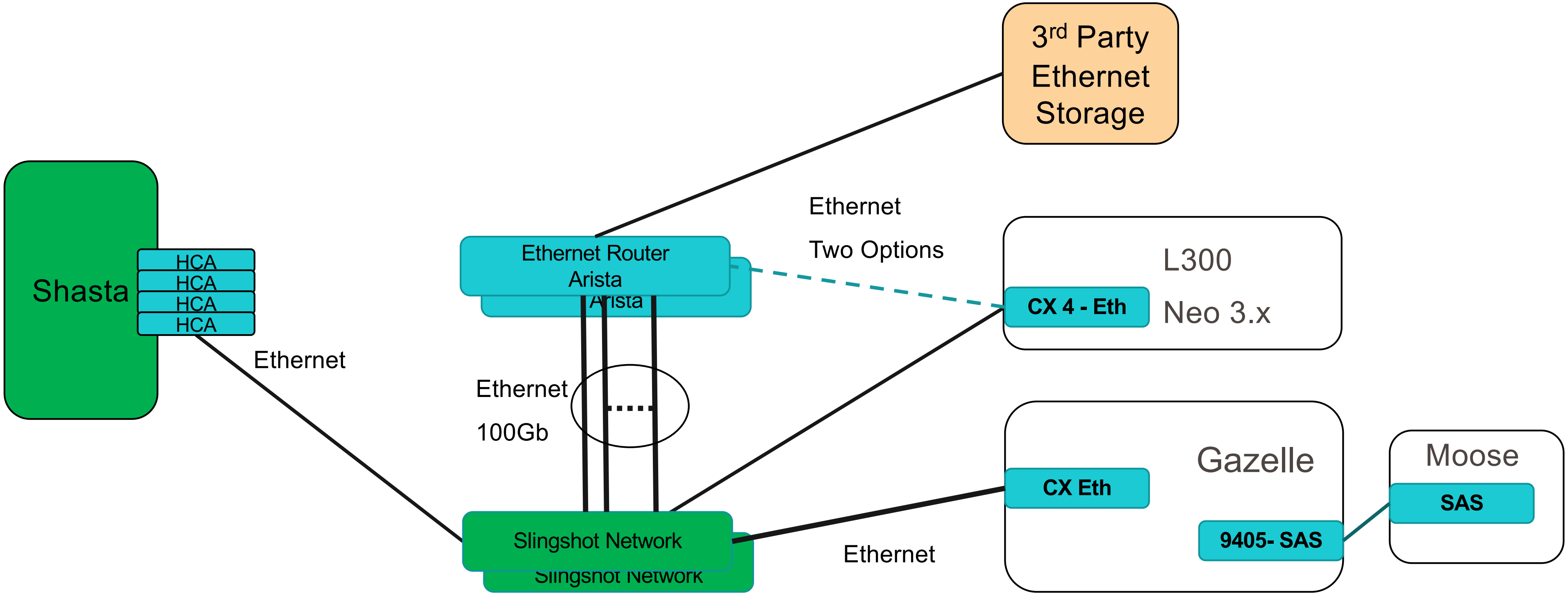
**Service**
- Automated migration
- Storage reservations

**Infrastructure**
- DataWarp service
- WLM integration

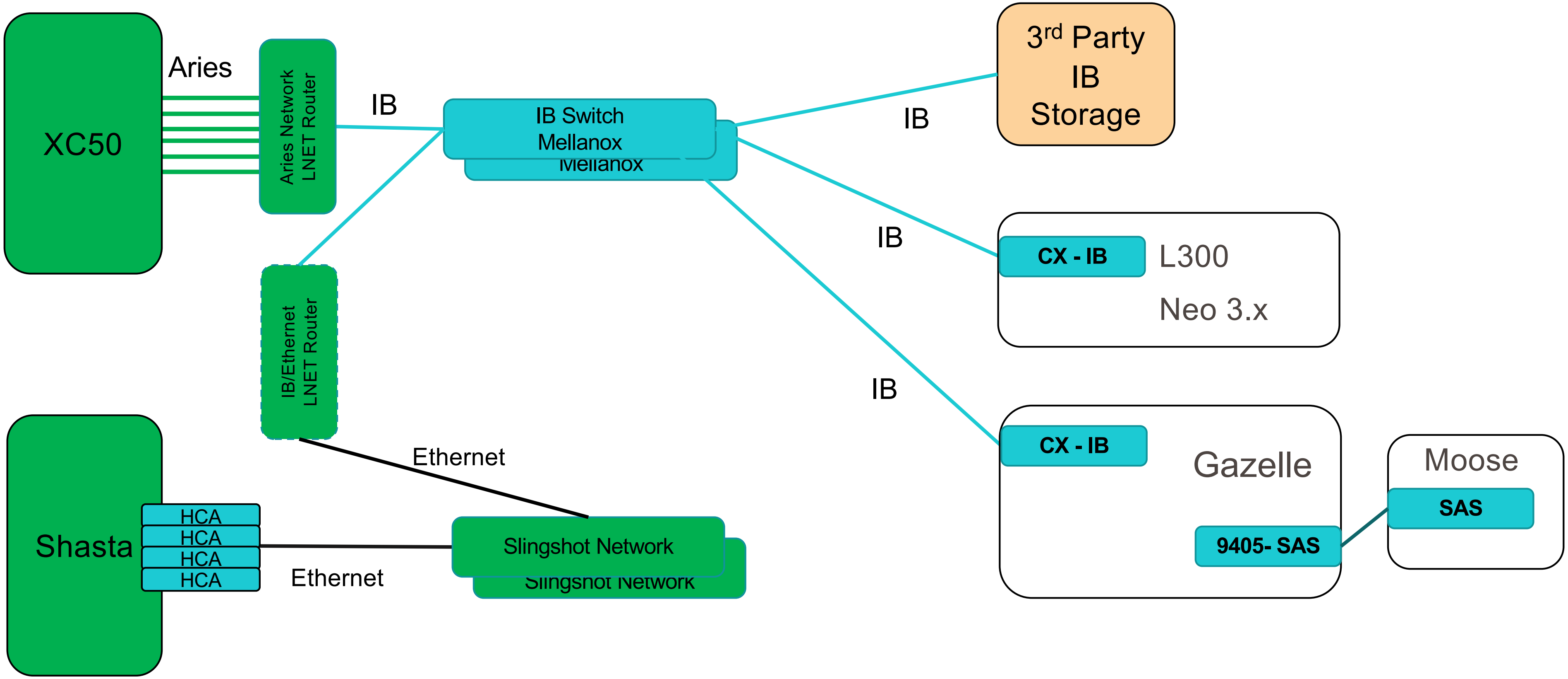
**Optimal Uses**
- Time critical jobs
- Bad I/O acceleration

**Future Services e.g. Transparent**

# Storage Data Paths – Ethernet



# Storage Data Paths – InfiniBand



# Status Update

Dave Poulsen

# Shasta Status & Early Customer Experience

- Cray R&D has engaged with (limited) customers around Shasta for some time
  - Collaboration group
  - Early previews of Shasta software
- Early results have been very encouraging!
  - Much work to be done
    - But starting earlier, and communicating more, is better
  - Increased confidence in Shasta v1
    - Collaboration has focused Cray on designing to meet customers’ needs

# Shasta v1 (Pre-)Release Cadence

| 2018 | 2019 | | | |
|---|---|---|---|---|
| Q4 | Q1 | Q2 | Q3 | Q4 |
| **Pre-Release 1**<br>*Installable, functional first release*<br>• **COTS** hardware<br>• Basic installer<br>• 1st system mgmt. (services & APIs)<br>• Kubernetes (K8s) orchestration<br>• Compile & launch basic MPI jobs | **Pre-Release 2**<br>*Solidified infrastructure, plus initial new features*<br>• **COTS** hardware<br>• Resilient K8s<br>• Common logging<br>• 1st CLIs for APIs<br>• Infrastructure work:<br>  • Pkg. & install<br>  • System mgmt.<br>  • User access<br>  • … | **Pre-Release 3**<br>*Considerable new v1 functionality*<br>• **COTS** hardware<br>• SLES15 CNOS<br>• UAS & end-user workflow, SLURM<br>• System mgmt.<br>• More PE<br>• Analytics | **Pre-Release 4**<br>*Feature-completeness for Shasta v1*<br>• **COTS** hardware<br>• SLES15 CNOS<br>• Install & upgrade<br>• System mgmt.<br>• UAS & WLM<br>• Cray PE<br>• Analytics | **Shasta v1 GA**<br>*Fully-validated v1 release, to be used in initial Shasta acceptances*<br>• **Shasta** hardware<br>• AMD Rome<br>• Rosetta<br>• SLES15 OS<br>• ... |

**Ongoing Shasta hardware enabling + scale-out readiness work…**

# Shasta v1 GA

- 1st production Shasta SW release is on track for later this year
  - Will be used in initial Shasta acceptances
- Validated, production-ready set of Shasta v1 GA features
  - *(see previous slide…)*
- Maturing internal R&D processes
  - Agile planning & SW devel.
  - Broad use of CI/CD/CT
  - DevOps best-practices

- Further development will occur post-v1
  - Hardware enabling
  - Scale-out & hardening
  - Merged system management & administration *(Shasta + Storage)*
  - System mgmt. & security enhancements
  - OS upgrades & enhancements
  - *And more new features…*

# Customer Feedback

- Early customer interactions with Cray R&D
  - Customers: early view of Shasta architecture & design ideas
  - Cray: validate Shasta design, get customer feedback
- Pre-release software has been a useful vehicle
  - Customers: early experience with Shasta SW
  - Cray: creates opportunities for collecting (specific) feedback
    - And has accelerated CI/CD/CT infrastructure development!
- Customer requests:
  - Seek architectural input / feedback, even before features are “fully baked”
  - Show how Shasta design addresses customers’ particular use-cases / needs
  - Educate Cray teams on customers’ perspectives & requirements

# SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

# THANK YOU

QUESTIONS?

✉ lkaplan@cray.com

cray.com
@cray_inc
linkedin.com/company/cray-inc-/