



# An Overview of Sandia National Laboratory's High Performance Computing Power Application Programming Interface (API)- Specification

John H. Laros III

Sandia National Laboratories

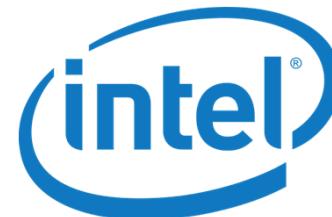
<http://powerapi.sandia.gov>

# Overview

- This will be a 10,000 foot view
  - The specification is necessarily broad in scope
  - Covering the specification in detail takes many hours
- A bit of history
- Collaboration from the start
- Important core principles
- Some higher level concepts
- Moving forward

# Who is Behind PowerAPI?

James H. Laros III, David DeBonis, Ryan Grant, Suzanne M. Kelly,  
Michael Levenhagen, Stephen Olivier, Kevin Pedretti

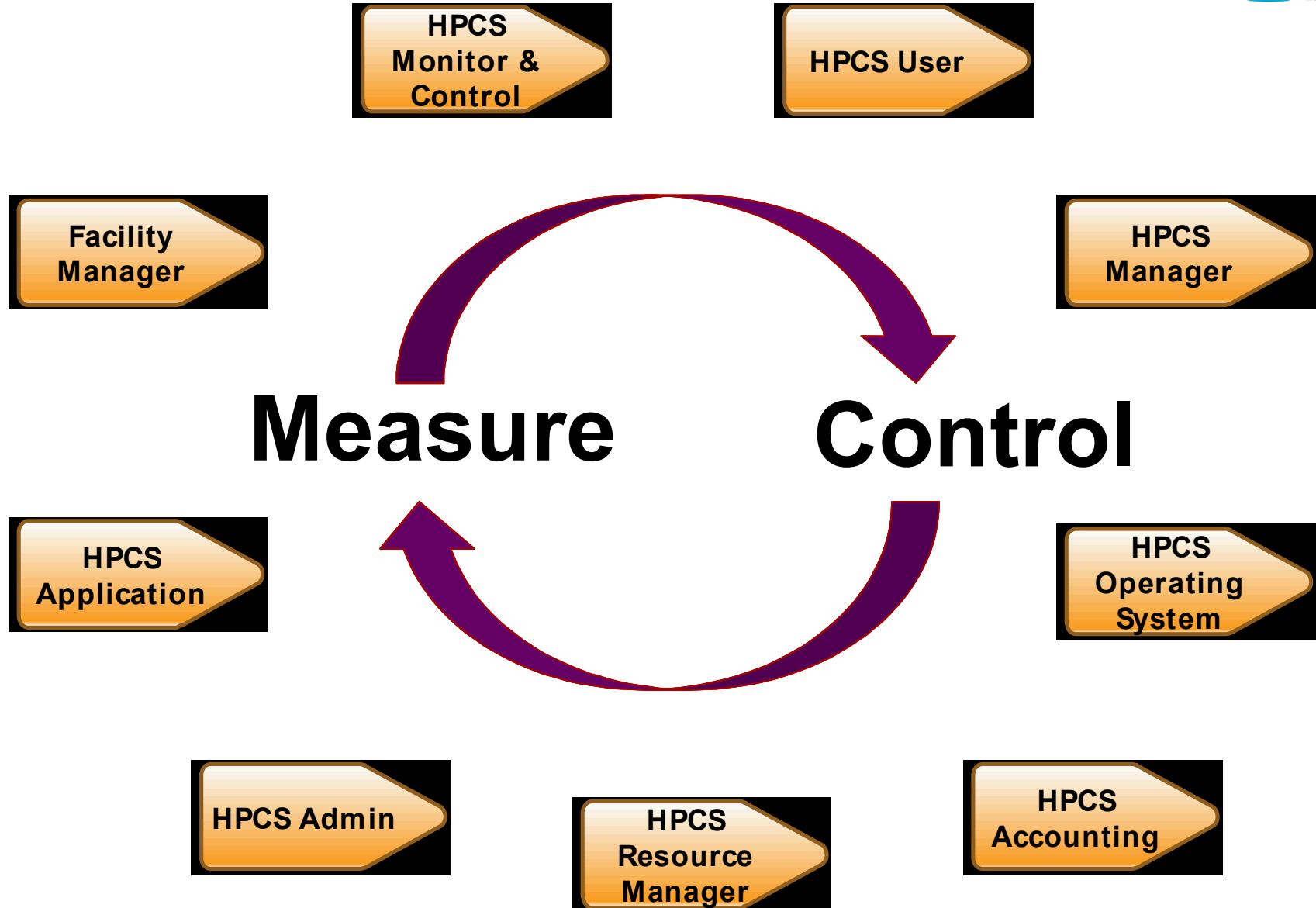


Sandia  
National  
Laboratories



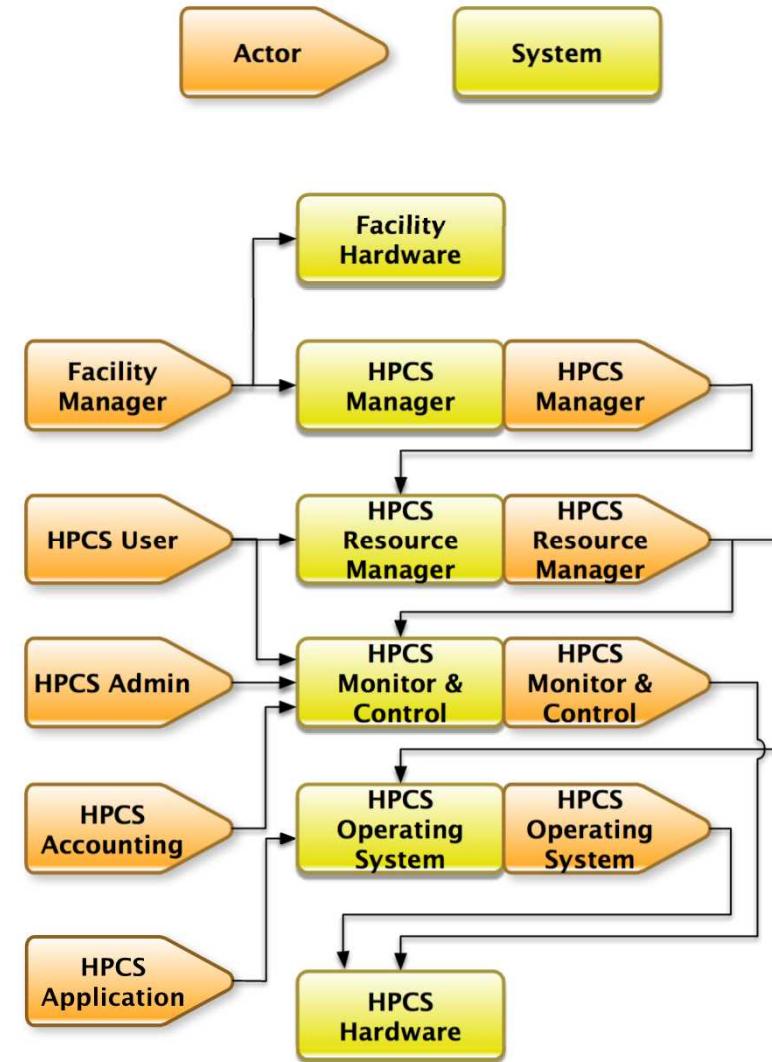
<Your logo here!>





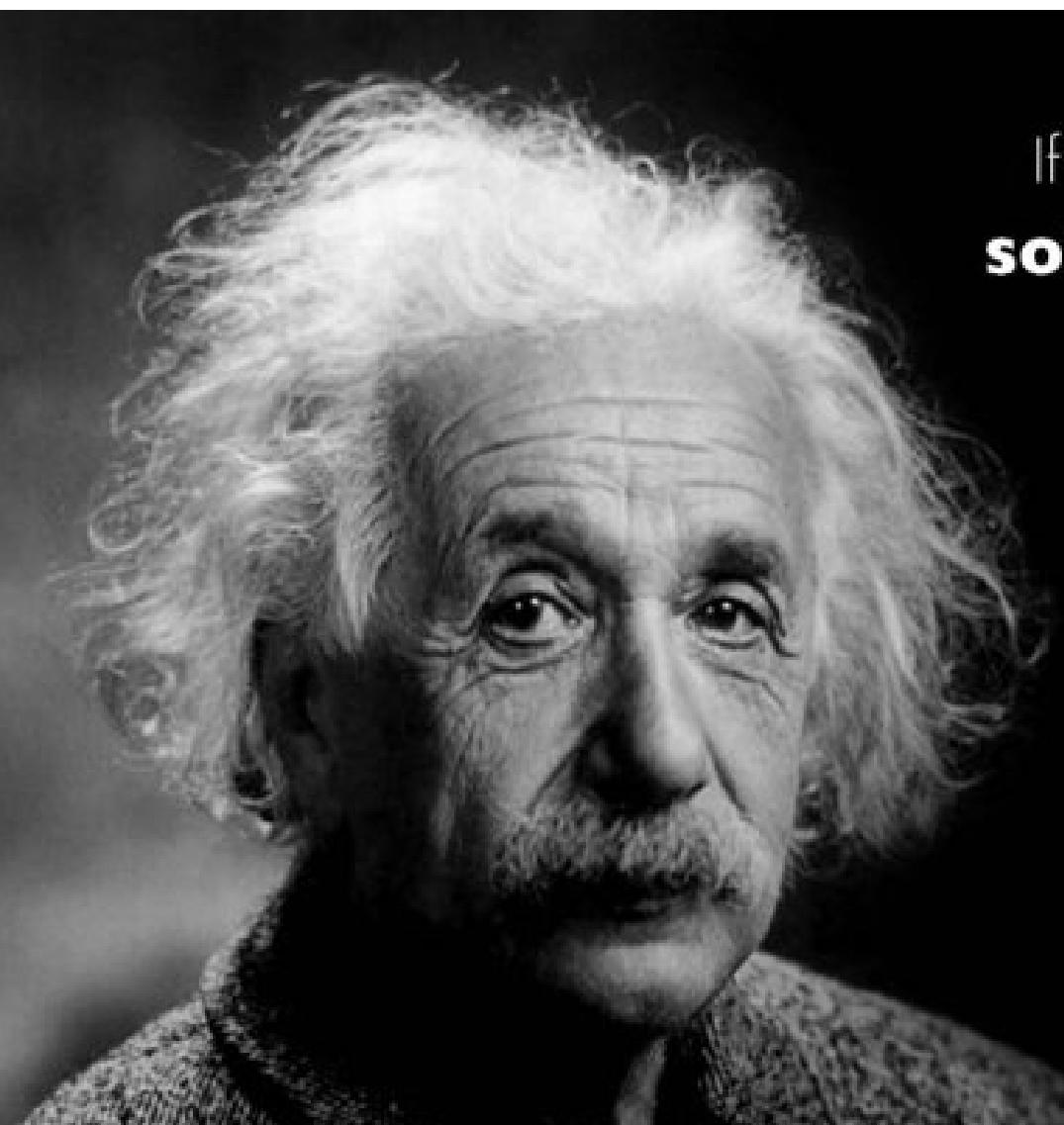
# A UML-ish Approach

- Diagram is the result of a UML study of the target space
  - Goal: Define Scope, Roles and Interfaces
- Arrows indicate interfaces or interaction between an Actor (Role) and System
  - Each interaction represents an interface that is defined in the specification
  - Specification is structured from the user or Role perspective
- Notice that an Actor (Role) can also be a System
- Cite use case document



# Goals

- Portability for the HPC community
  - Wouldn't it be nice to develop tools that worked on all your machines with little to no modification?
  - Same desire exists no matter what Role you play
    - More about Roles later
- Forecast emerging needs of HPC community
  - As a group, inform the vendors of how we want to use systems now and in the future
  - Specification acts as a basis of collaboration
- Expose new capabilities developed by vendors and community
  - Leverage vendor and community innovations in this and related spaces
  - E.g. Geo and Redfish
- Most important, want something out there to throw stones at
  - Need a starting point!



If I had an hour to  
**solve a problem** and my  
**life depended** on it,  
I would use the  
first 55 minutes  
determining the  
**proper questions to ask.**

Albert Einstein

# What is the Power API?

A comprehensive API for power **MEASUREMENT** and **CONTROL** of HPC platforms

- *Comprehensive = Facility to Component*
- *API = Define the interFACE not the mechanism*
- *HPC platforms = Facility (or datacenter) and all the platforms within*

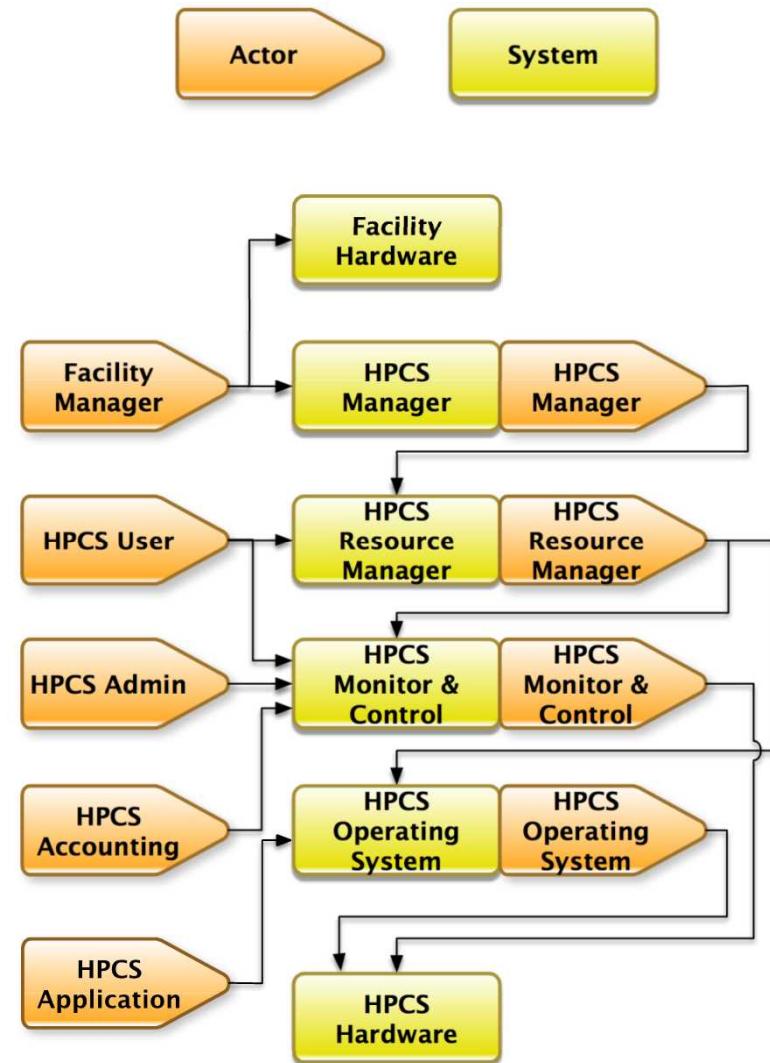
Considers all users of HPC platform - people and programs

- *Core (Common)*
  - *Common among all “users”*
    - *Includes: Roles, Initialization, Navigation, Objects and Groups, Attributes (Get/Set), Metadata and Statistics*
- *High-Level Common*
  - *Higher level of abstraction but still potentially common among multiple Roles*
- *Role/System Specific*
  - *Higher level abstraction specific to how Role interfaces with system*

# Roles

## PWR\_Role

```
typedef enum {  
    PWR_ROLE_APP, /* Application */  
    PWR_ROLE_MC, /* Monitor and Control */  
    PWR_ROLE_OS, /* Operating System */  
    PWR_ROLE_USER, /* User */  
    PWR_ROLE_RM, /* Resource Manager */  
    PWR_ROLE_ADMIN, /* Administrator */  
    PWR_ROLE_MGR, /* HPCS Manager */  
    PWR_ROLE_ACC /* Accounting */  
} PWR_Role;
```



# Roles

- **Application** – Application or application library executing on the compute resource; May include run-time components running in user space
- **Monitor and Control** -- Cluster management or Reliability Availability and Serviceability (RAS) systems, for example.
- **Operating System** -- Linux or specialized lightweight kernels and privileged portions of run-time systems. Privilege escalation layer.
- **User** -- The end user of the HPC platform.
- **Resource Manager** – Can include work load managers, schedulers, allocators and even portions of run-time systems that manage resources.
- **Administrator** – System administrator or day-to-day platform manager.
- **HPCS Manager** -- Individual(s) responsible for managing policy for the HPC platform, often through scheduling policy. Implements facility parameters.
- **Accounting** -- Individual or software that produces reports of metrics for the HPC platform.

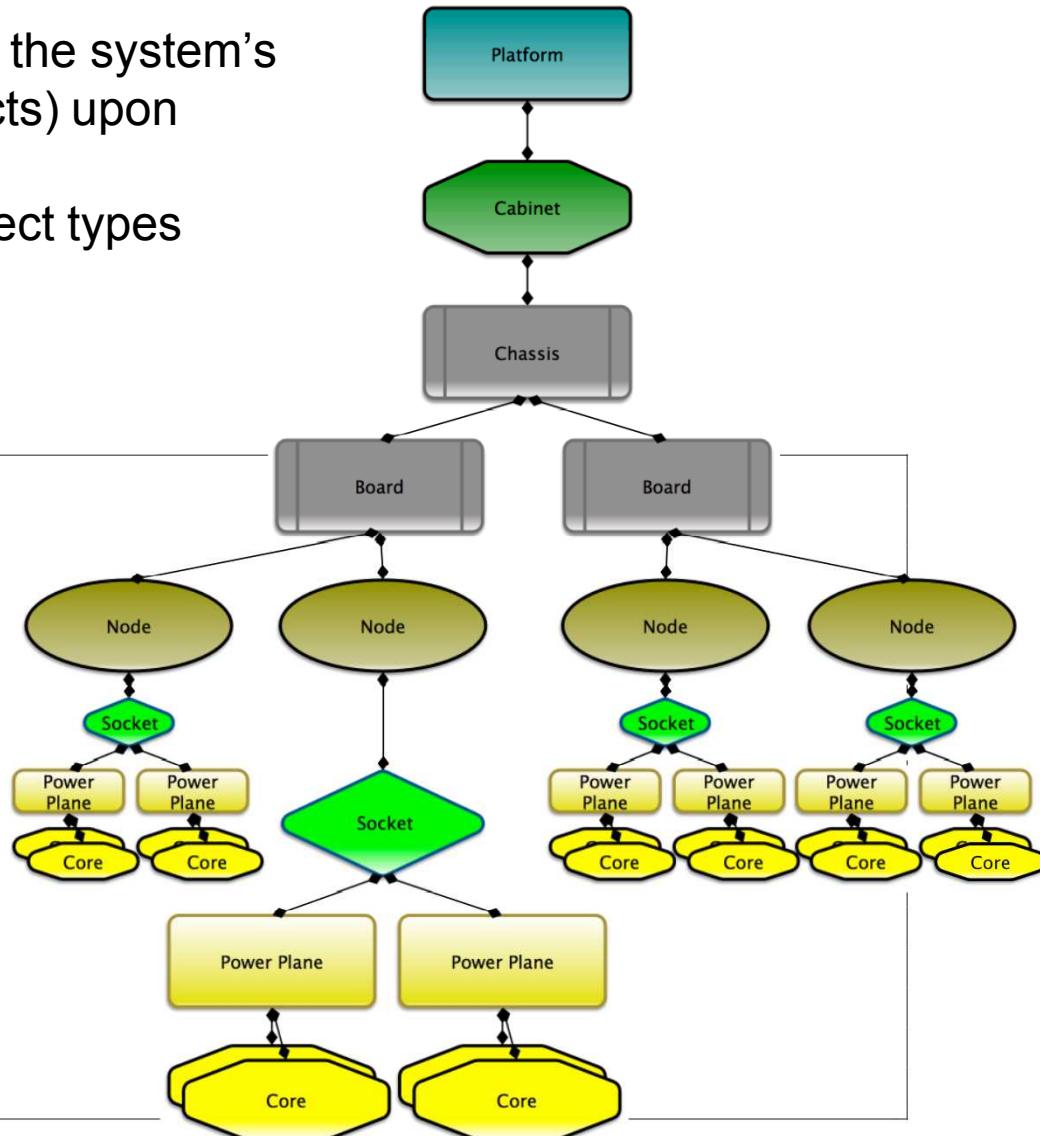
# System Description

Presents a navigable view of the system's hardware components (objects) upon initialization

- Can extend to custom object types
- Can be heterogeneous

## PWR\_ObjType

```
typedef enum {
    PWR_OBJ_PLATFORM = 0,
    PWR_OBJ_CABINET,
    PWR_OBJ_CHASSIS,
    PWR_OBJ_BOARD,
    PWR_OBJ_NODE,
    PWR_OBJ_SOCKET,
    PWR_OBJ_CORE,
    PWR_OBJ_POWER_PLANE,
    PWR_OBJ_MEM,
    PWR_OBJ_NIC,
    PWR_NUM_OBJ_TYPES,
    /* */
    PWR_OBJ_INVALID = -1,
    PWR_OBJ_NOT_SPECIFIED = -2
} PWR_ObjType;
```



# Objects

- **Objects and Groups**
  - Objects represent components of a system
    - Lots of flexibility in what a component is
  - System Description is the organization of these objects to represent the system
  - Representation may be dependent on Role (and other considerations)
  - Attributes (later) are associated with objects
  - Groups can be implementation (predefined) or user defined (long or short lived)
- **Navigate** the system description provided upon **Initialization**
  - Entry point may depend on Role
    - Node entry point for Application
    - Platform entry point for Administrator
  - Navigate up (parent object), down (child objects)
  - Navigation can also be thought of as Discovery

# Attribute Interface

```
typedef enum {
    PWR_ATTR_PSTATE = 0, /* uint64_t */
    PWR_ATTR_CSTATE, /* uint64_t */
    PWR_ATTR_CSTATE_LIMIT, /* uint64_t */
    PWR_ATTR_SSTATE, /* uint64_t */
    PWR_ATTR_CURRENT, /* double, amps */
    PWR_ATTR_VOLTAGE, /* double, volts */
    PWR_ATTR_POWER, /* double, watts */
    PWR_ATTR_POWER_LIMIT_MIN, /* double, watts */
    PWR_ATTR_POWER_LIMIT_MAX, /* double, watts */
    PWR_ATTR_FREQ, /* double, Hz */
    PWR_ATTR_FREQ_LIMIT_MIN, /* double, Hz */
    PWR_ATTR_FREQ_LIMIT_MAX, /* double, Hz */
    PWR_ATTR_ENERGY, /* double, joules */
    PWR_ATTR_TEMP, /* double, degrees Celsius */
    PWR_ATTR_OS_ID, /* uint64_t */
    PWR_ATTR_THROTTLED_TIME, /* uint64_t */
    PWR_ATTR_THROTTLED_COUNT, /* uint64_t */
    PWR_NUM_ATTR_NAMES,
    /* */
    PWR_ATTR_INVALID = -1,
    PWR_ATTR_NOT_SPECIFIED = -2
} PWR_AttrName;
```

# Attribute Interface

## MEASURE

```
int PWR_ObjAttrGetValue( PWR_Obj object,
                         PWR_AttrName attr,
                         void* buf,
                         PWR_Time* ts);
```

## CONTROL

```
int PWR_ObjAttrSetValue( PWR_Obj object,
                         PWR_AttrName attr,
                         void* buf );
```

*Symmetric calls available for operating on groups of objects*

# Attributes: Common Functionality

- **Attributes** (measure and control) of objects and groups of objects
  - Access dependent on Role and other implementation specific considerations
- **Get and Set operations** enable basic measurement and control for the exposed object attributes (and groups of objects)
- Attributes can represent generic measurement and control features
  - Power, Voltage, Current, Frequency
- Architecture specific features are permissible at the lowest levels
  - Pstate, Cstate may not be meaningful for all architectures
- An attribute, like power, can represent an instrumentation point or a summation of underlying instrumentation points
  - Power attribute of a CPU object
  - Power attribute of a Node object

# Metadata Interface

```

typedef enum {
  PWR_MD_NUM = 0, /* uint64_t */
  PWR_MD_MIN, /* either uint64_t or double, depending on attribute type */
  PWR_MD_MAX, /* either uint64_t or double, depending on attribute type */
  PWR_MD_PRECISION, /* uint64_t */
  PWR_MD_ACCURACY, /* double */
  PWR_MD_UPDATE_RATE, /* double */
  PWR_MD_SAMPLE_RATE, /* double */
  PWR_MD_TIME_WINDOW, /* PWR_Time */
  PWR_MD_TS_LATENCY, /* PWR_Time */
  PWR_MD_TS_ACCURACY, /* PWR_Time */
  PWR_MD_MAX_LEN, /* uint64_t, max strlen of any returned metadata string. */
  PWR_MD_NAME_LEN, /* uint64_t, max strlen of PWR_MD_NAME */
  PWR_MD_NAME, /* char *, C-style NULL-terminated ASCII string */
  PWR_MD_DESC_LEN, /* uint64_t, max strlen of PWR_MD_DESC */
  PWR_MD_DESC, /* char *, C-style NULL-terminated ASCII string */
  PWR_MD_VALUE_LEN, /* uint64_t, max strlen returned by PWR_MetaDataAtIndex */
  PWR_MD_VENDOR_INFO_LEN, /* uint64_t, max strlen of PWR_MD_VENDOR_INFO */
  PWR_MD_VENDOR_INFO, /* char *, C-style NULL-terminated ASCII string */
  PWR_MD_MEASURE_METHOD, /* uint64_t, 0/1 depending on real/model mesurement */
  PWR_NUM_META_NAMES,
  /* */
  PWR_MD_INVALID = -1,
  PWR_MD_NOT_SPECIFIED = -2
} PWR_MetaData;

```

# Metadata: Common Functionality

- **Metadata** interface provides information about quality, frequency, and other characteristics associated with attributes of objects
  - Can be specific for a particular attribute/object pair
    - All power sensors might not provide the same accuracy
  - Frequency of sample collection can help determine usefulness of data
  - Can also, in some cases, set metadata
    - Potentially to change how a device responds

# Statistics Interface

## PWR\_AttrStat

```
typedef enum {
    PWR_ATTR_STAT_MIN = 0,
    PWR_ATTR_STAT_MAX,
    PWR_ATTR_STAT_AVG,
    PWR_ATTR_STAT_STDEV,
    PWR_ATTR_STAT_CV,
    PWR_NUM_ATTR_STATS,
    /* */
    PWR_ATTR_STAT_INVALID = -1,
    PWR_ATTR_STAT_NOT_SPECIFIED = -2
} PWR_AttrStat;
```

# Statistics: Common Functionality

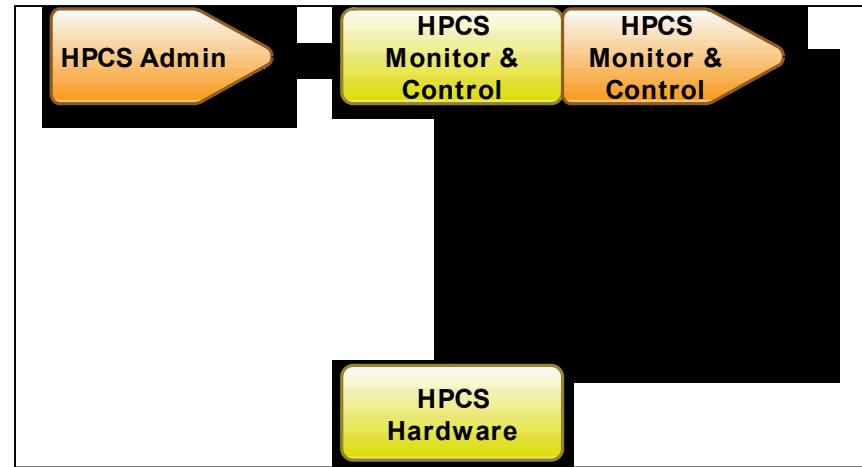
- **Statistics** interface gathers data on one or more attributes for an object or group of objects
  - Real time or historic statistics
    - Historic implies data retention (database for example)
  - Min, Max, Average, Standard Deviation, Coefficient of Variation
  - Reduction operation available
  - User provided statistic function on the to-do list
- Provides functions to...
  - Start, stop, and reset statistics gathering
  - Get the calculated value(s) for the object or group of objects
  - Reduce the values calculated for objects in a group into a single value



# High Level Interfaces by Role (1)

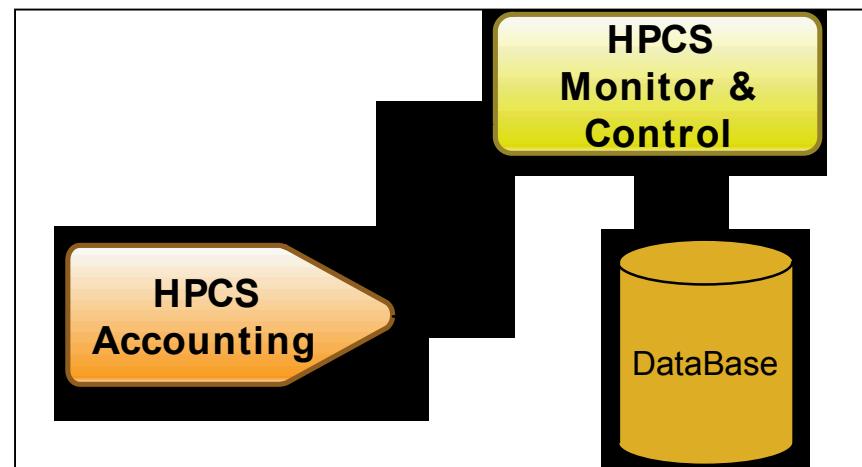
## Administrator:

- Apply Hard Power Limits based on Facility parameters
  - Bounds Power Aware Scheduling



## Accounting:

- Power/Energy Application profiling based on historic information
  - Feeds into Power Aware Scheduling



# High Level Interfaces by Role (2)

## Resource Manager

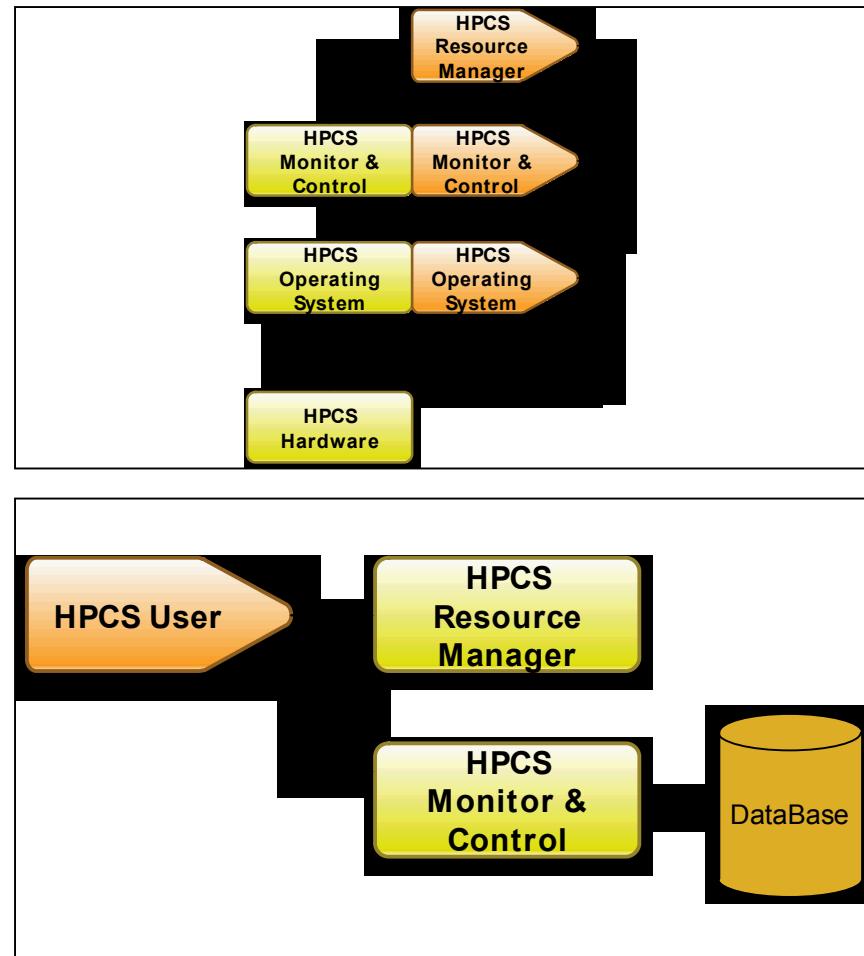
- Power Aware Scheduling
  - HPC Tetris

## User

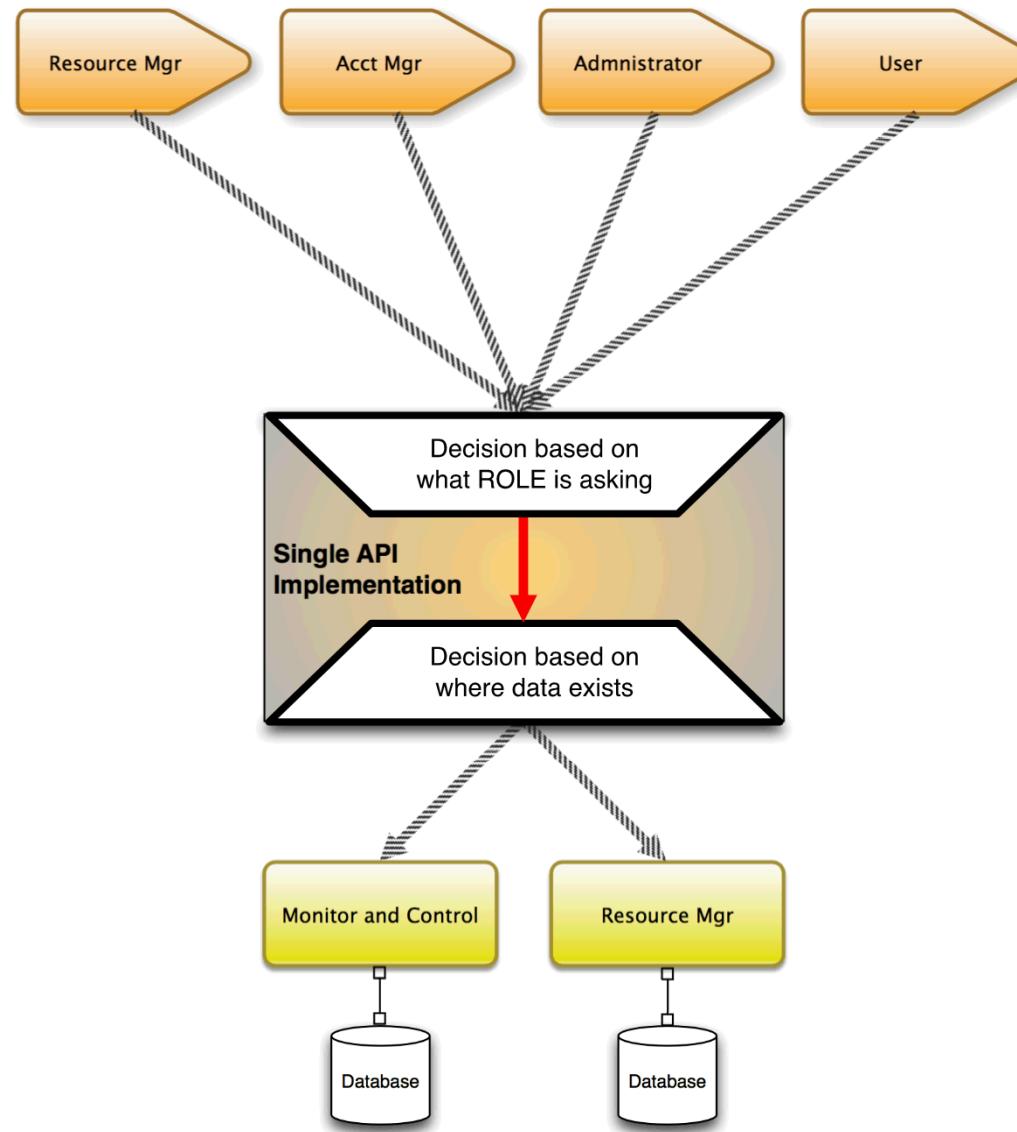
- Understands application  
Power and Energy  
Characteristics and Phases

## Application

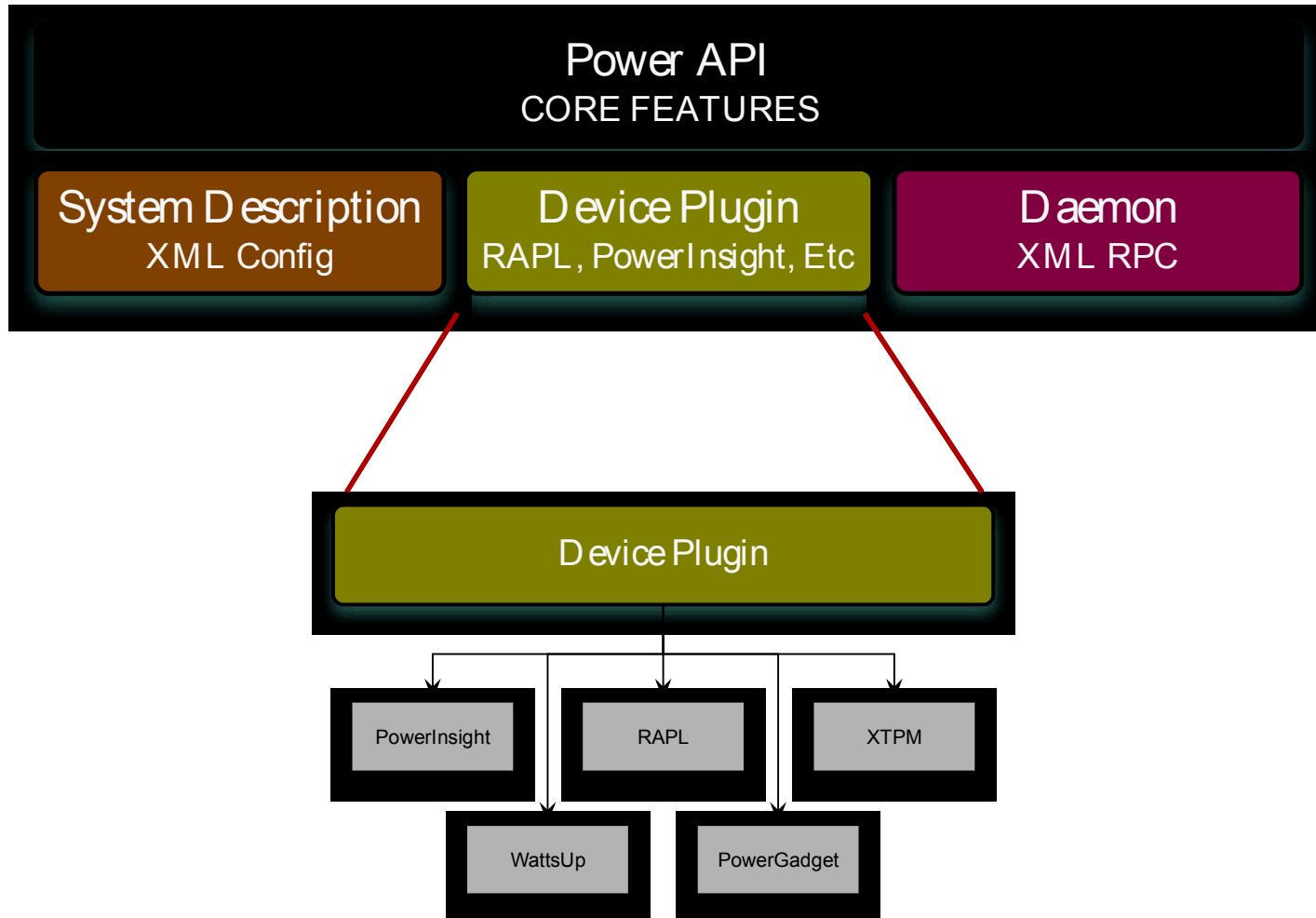
- Provide application hints  
based on profiling



# One Implementation Across Multiple Interfaces



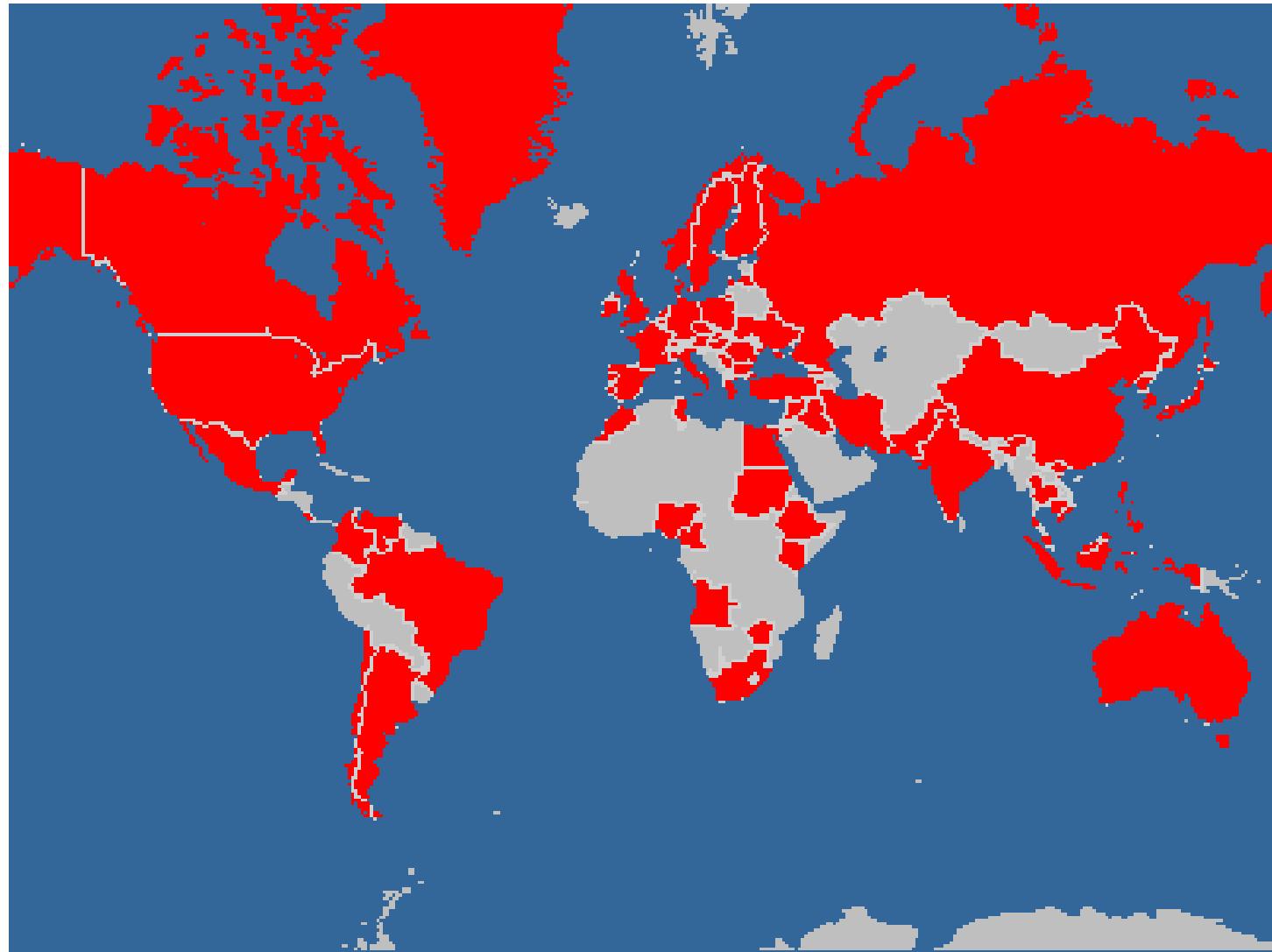
# Reference Implementation



Available online and open source: <http://github.com/pwrapi>

# Power API Timeline

- 2013: Use case document prepared by SNL and NREL and reviewed by partners
- July 2014: Draft specification review meeting with cross-vendor panel of experts
- Aug. 2014: **Specification v1.0** release (<http://powerapi.sandia.gov/>)
- Sept. 2014: Day-long community launch meeting with labs, industry, academia
- Jan. 2015: Prototype implementation release
- June 2015: **Reference implementation** release (<http://github.com/pwrapi>)
- Aug. 2015: Specification v1.1 release
- Oct. 2015: **Specification v1.1a** release (<http://powerapi.sandia.gov/>)



**81 Countries as of October 2015**

- DOE NNSA's Advanced Technology System (ATS-1)
  - >19,000 Nodes, <10MW
- Introduced the concept of funding Non-Recurring Engineering (NRE) projects to advance important technologies in conjunction with platform procurement
- Cray contracted to address Advanced Power Management
  - Implement portions of the Power API at scale

Targeting two areas of the API:

1. Implement interface to Power Management Database
  - Extend specification to include Python
  - Monitor and Control in our diagram
2. Compute node implementation (native C)

# Going Forward

- How do we move forward?
- What “standards” model to apply?
- Regular calls?
  - Frequency?
- Face to Face meetings
  - Frequency?
- Important to have broad community participation which includes vendor representatives

# Thank you – Questions?



<http://powerapi.sandia.gov/>



## Acknowledgments:

This work was funded through the Computational Systems and Software Environment sub-program of the Advanced Simulation and Computing Program funded by the National Nuclear Security Administration