Physics
The Integrated Plasma Simulator Framework For Loosely Coupled Simulations

Wael R. Elwasif

Oak Ridge National Laboratory

The origins of IPS: The SWIM Project

- Center for Simulation of RF Wave Interactions with Magnetohydrodynamics (SWIM).
- One of three DOE SCIDAC centers looking into coupled fusion simulations
  - Typically referred to as the proto-FSP projects.
- Primary Objective
  - Study the use of RF Waves to control the stability of burning plasma in a fusion tokamak.
- More info: http://cswim.org

Motivation and Background

- Systemic coupling of disparate fusion codes
  - Prelude to Fusion Simulation Project (FSP)
- Heavily used, mature, long-lived codes
  - Occasional two-way coupling
- Different characteristics and capabilities
  - Parallelism, data format, execution work flow,..
- No mandate to re-factor major codes
  - Beyond the scope of the project.
- Codes ***WILL*** change during the project lifetime
  - Avoid forking and loss of new features.

Computing Philosophy & Approach

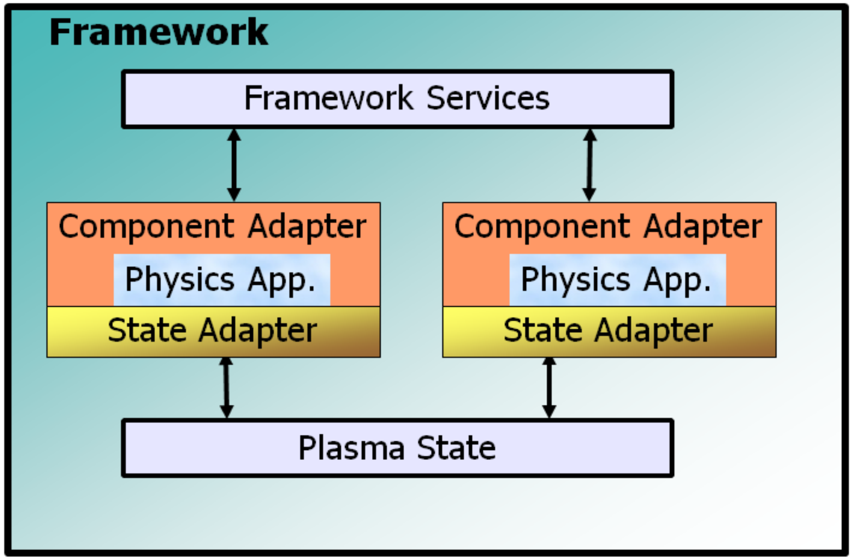
- Minimize level of effort to bring in physics codes
  - Avoid bifurcation of physics modules – not different SWIM/stand-alone versions
  - Wrappers around unmodified codes
  - Use application native I/O, transform to shared data using ***state adapters***
- Design for broader range of integrated simulation than required
  - Prototype for FSP framework needs - ***generalizability***
  - Target loose coupling initially, but with concepts that “scale” to stronger coupling

Computing Philosophy & Approach (2)

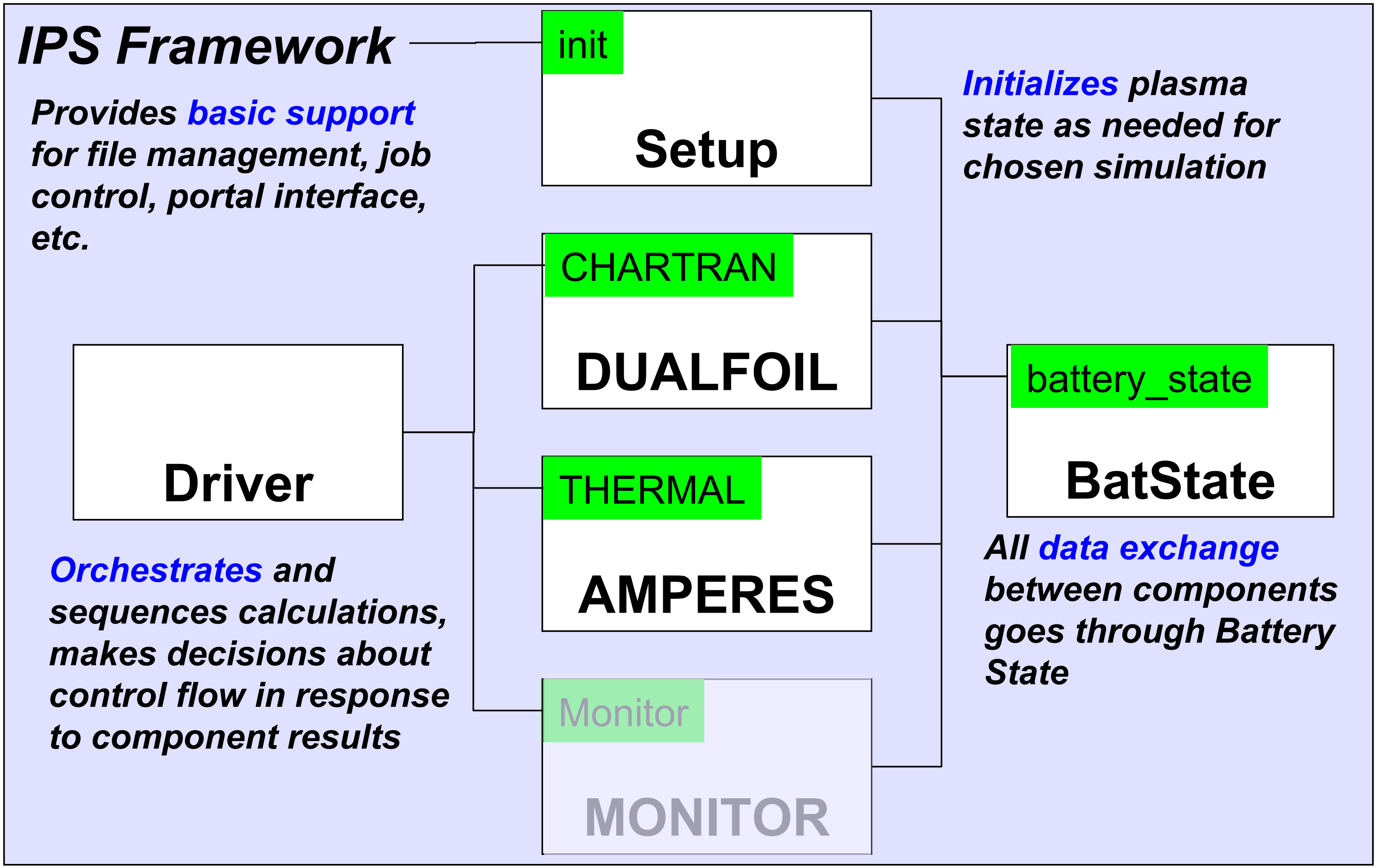
- Design for multiple implementations of each physics component
  - Better definitions of interfaces
  - Accommodate reduced models, inter-comparisons (V&V), etc.
- Component Approach
  - Based on Common Component Architecture concepts
  - Simplified implementation, focusing on concepts, key features

The Integrated Plasma Simulator (IPS): Design Features

- Simulation framework
  - Light weight, Python-based implementation (***4332 LOC***)
  - Adaptability, extensibility, and flexibility
  - Provide ***services*** to connected components
- Pluggable components:
  - Python and Python-wrapped functional units
  - Use framework services to coordinate execution
- Plasma state layer
  - Data repository, conduit for inter-component data exchange
- File-Based data exchange
  - No change to underlying codes
  - Simplify *"**unit testing**"*

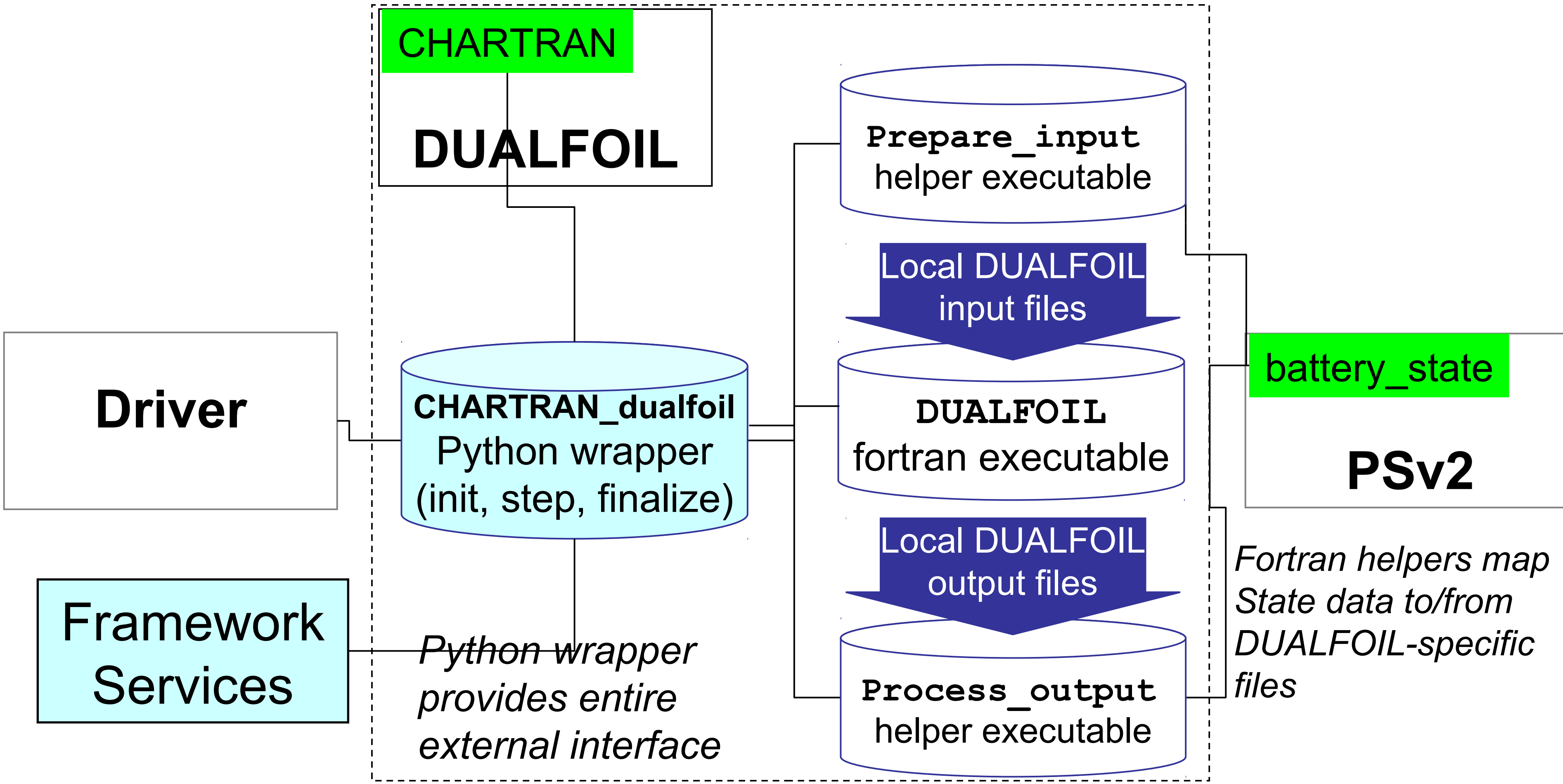


Schematic of an IPS Application



Components implement (one or more) specific interfaces.
A given interface may have multiple implementations.

Drilling Down: Typical Component Structure

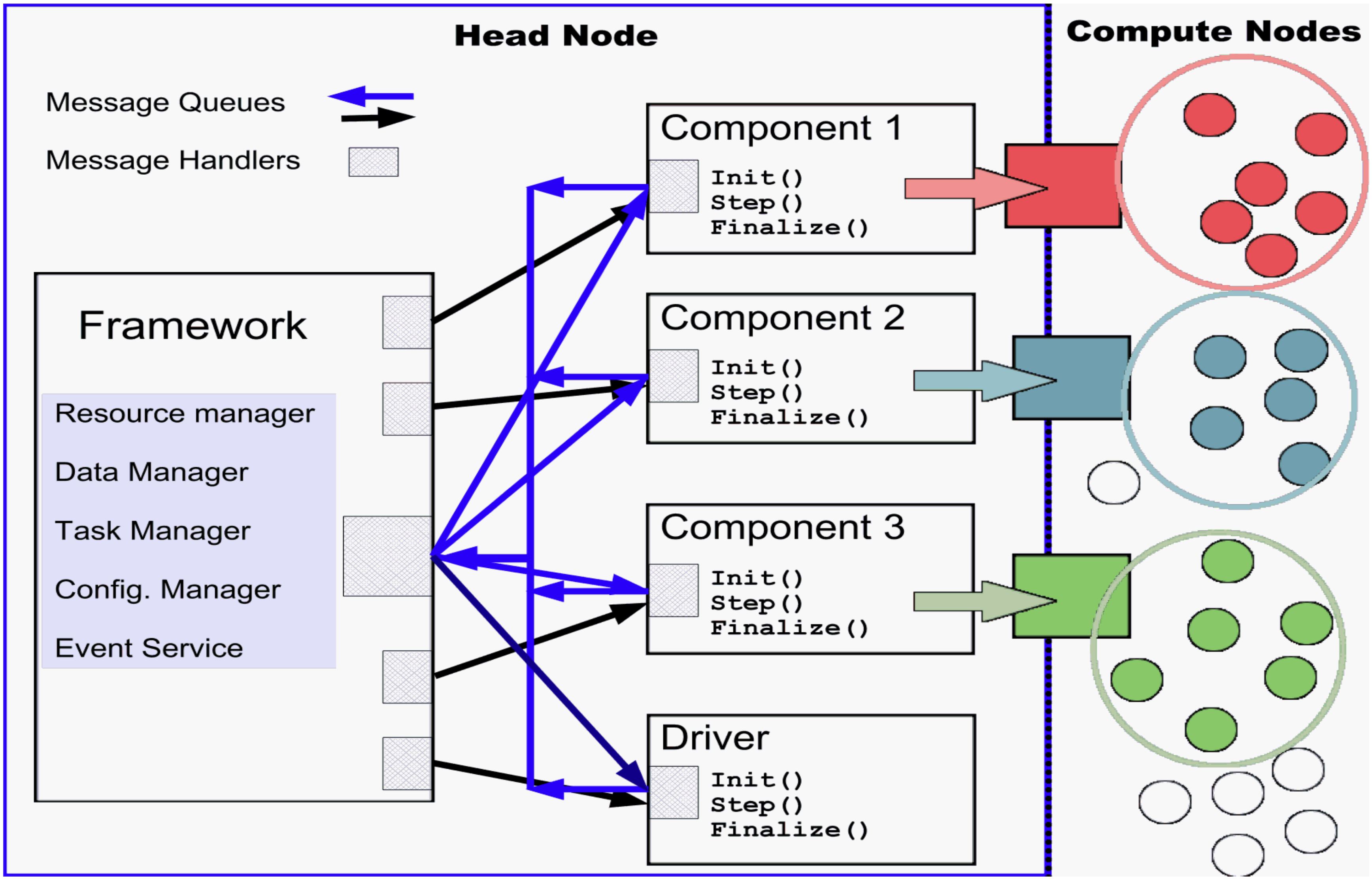


IPS design/specifications say nothing about internal implementation of components.

Hooking it All Up – IPS Framework Services

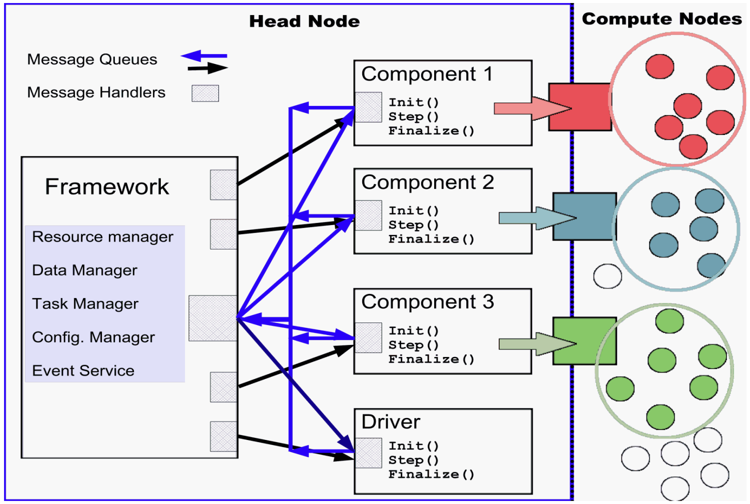
- Configuration management
  - Simulation configuration
  - Component instantiation and configuration
- Task management
  - Mediate inter-component method invocation
  - Manage execution of underlying applications
- Data management
  - Input/output data staging
  - Mediate concurrent access to plasma state files
  - Manage data for checkpoint and restart (framework level)
- Resource management
  - Manages pool of resources provided to batch job in which IPS is running
  - Concurrent access to shared simulation resources (mainly compute nodes)
- Event management
  - Asynchronous publish/subscribe event model for inter-component information exchange
- Simulation monitoring
  - Progress monitoring via SWIM web portal

IPS Execution Environment



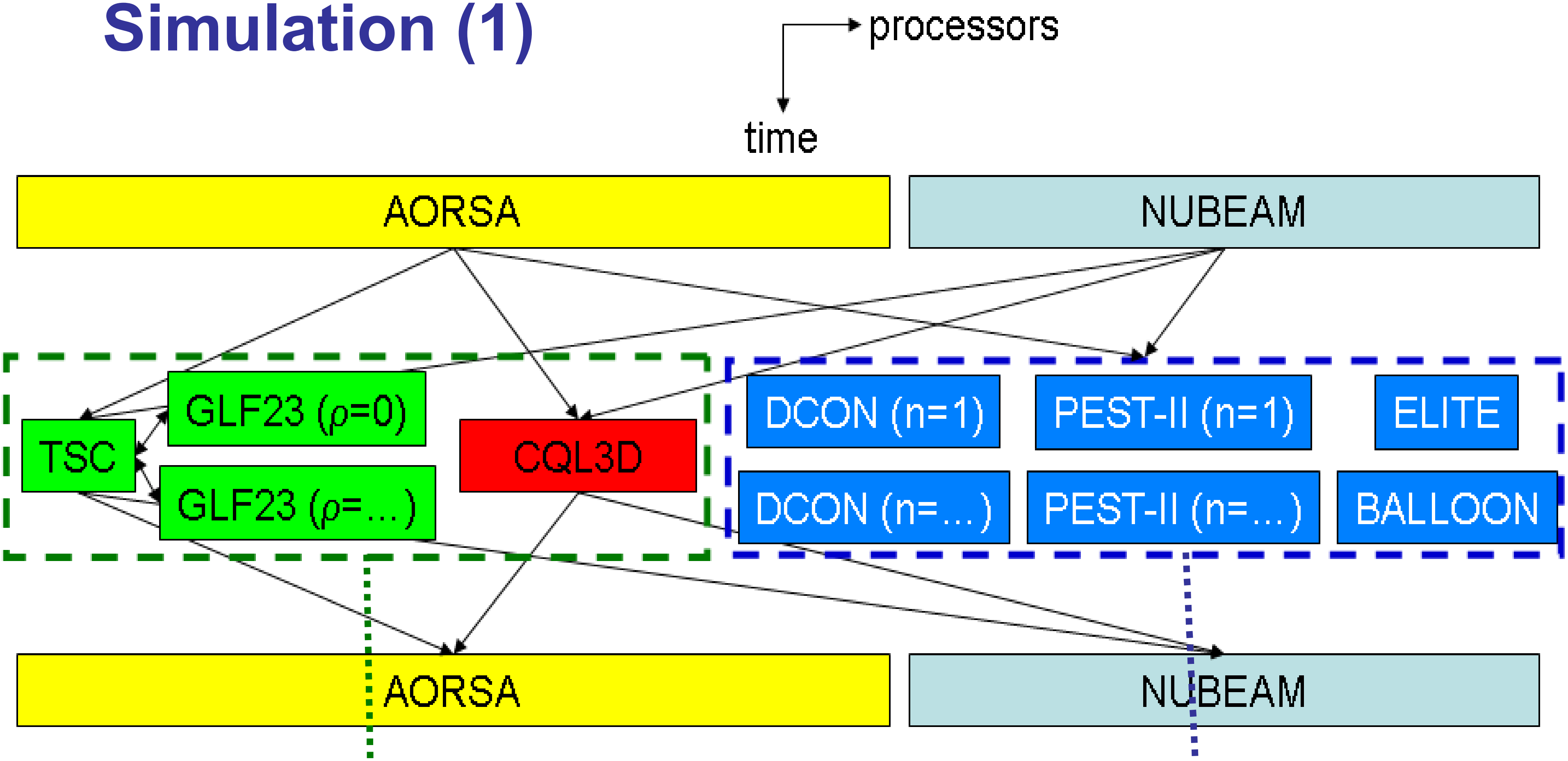
IPS Supports *Four* Levels of Parallelism

- ***Parallel Tasks*** (physics applications)
  - Used routinely – Physics applications vary in parallelism
- ***Concurrent task execution***
  - A component can launch multiple concurrent tasks
  - Useful to (for example) localized sensitivity analysis or parameter sweep



- ***Concurrent component*** method execution
  - Also known as concurrent multitasking or multiple-component multiple-data (MCMD) execution
  - As long as data dependencies are respected, many components can be run concurrently
  - Exposes more parallelism; can improve resource utilization, time to solution
- ***Multiple independent simulations*** can be executed in a single IPS invocation
  - Simple extension of concurrent multitasking
  - Exposes more parallelism; can improve resource utilization, time to solution
  - Basis for parameter sweeps using the IPS

Concurrent Multitasking for a Complex Simulation (1)

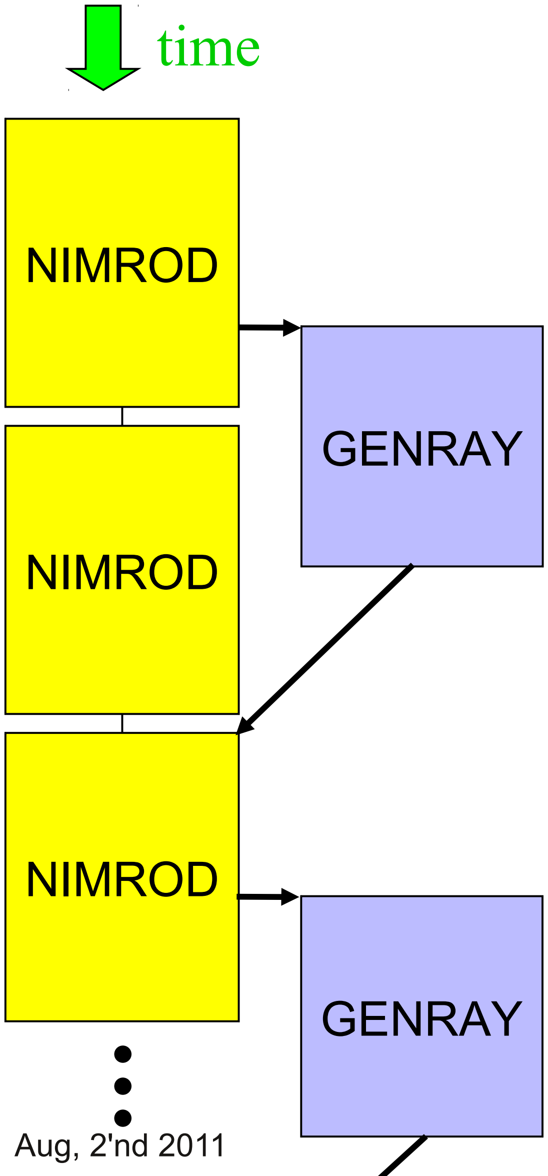


Equilibrium and profile advance for step t, including parallel anomalous transport tasks for each flux surface, all running concurrently with the Fokker Planck component.

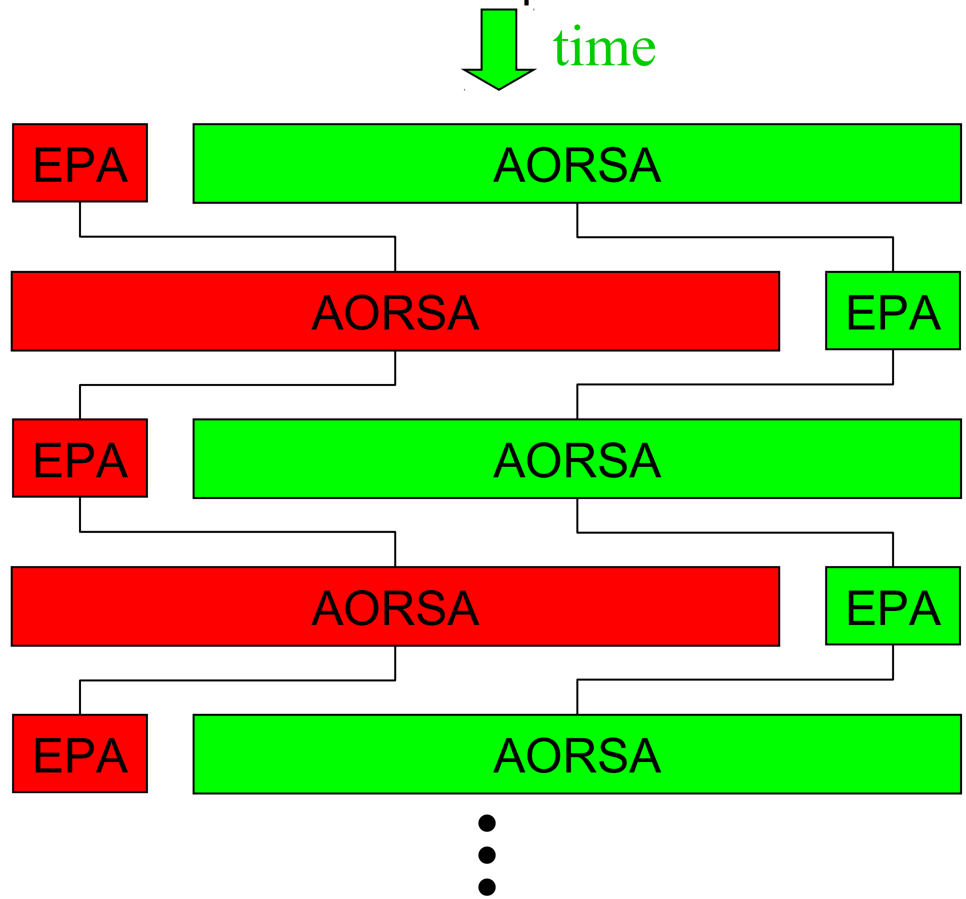
Multiple stability analysis components running on multiple toroidal modes, all running concurrently on t-1 results.

Concurrent Multitasking (2) and Multiple Simulations

Initial Slow MHD Scenario



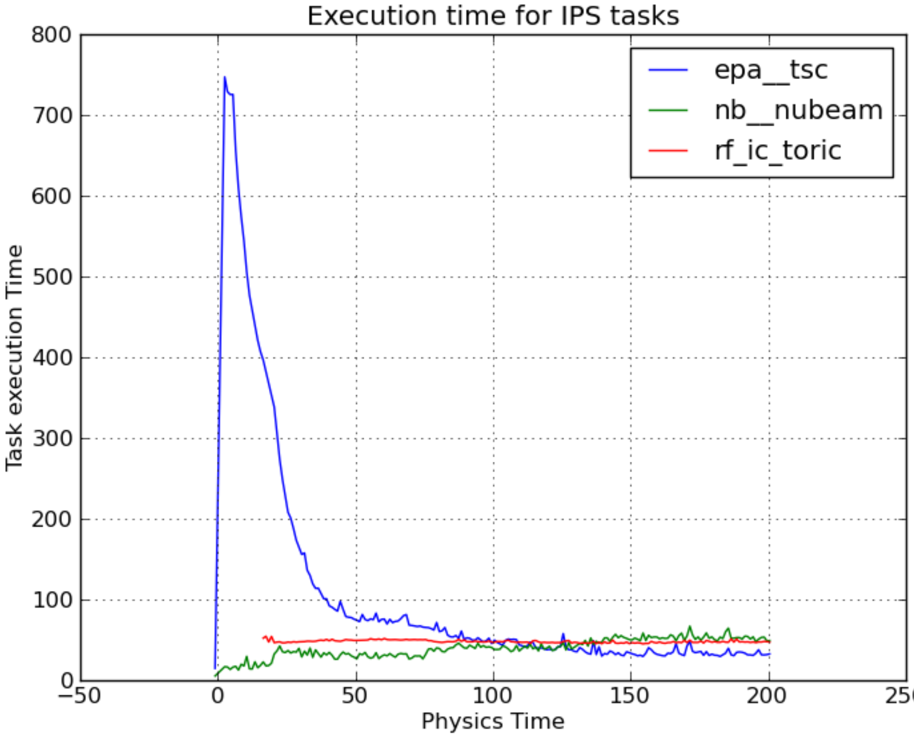
"Ensemble" of Coupled Simulations



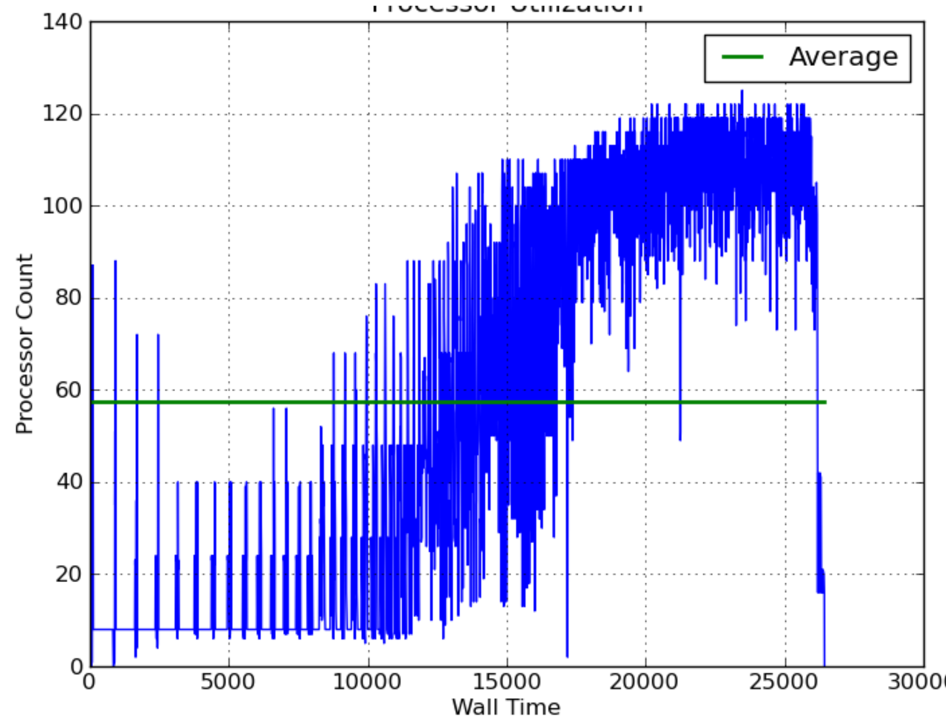
Two (or more) simulations (i.e. multiple pedastal heights) share the same processor allocation, running out of phase to maximize utilization.

Multiple Simulations In Action On the Cray XT5 at NERSC

Execution time for IPS tasks



Processor utilization for 9-simulation parameter scan

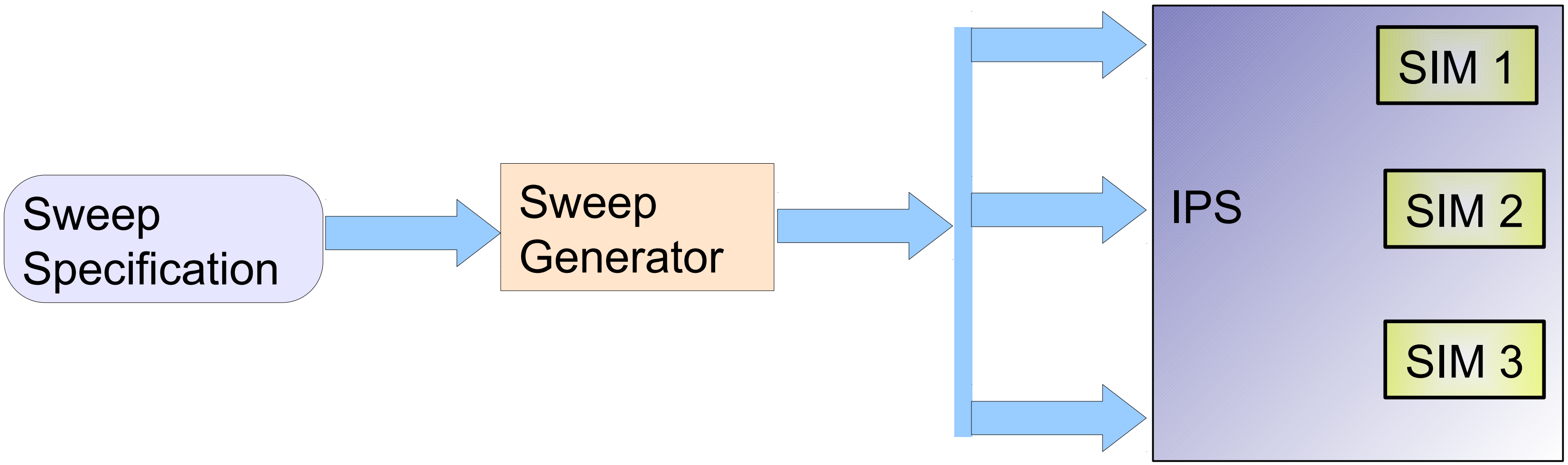


- Average processor usage for first 200 sec of simulation is about 58%. Is this good?
- How can I know how many simultaneous simulations to run and how many cores to use?

Flexible Task Parallelism in IPS Components

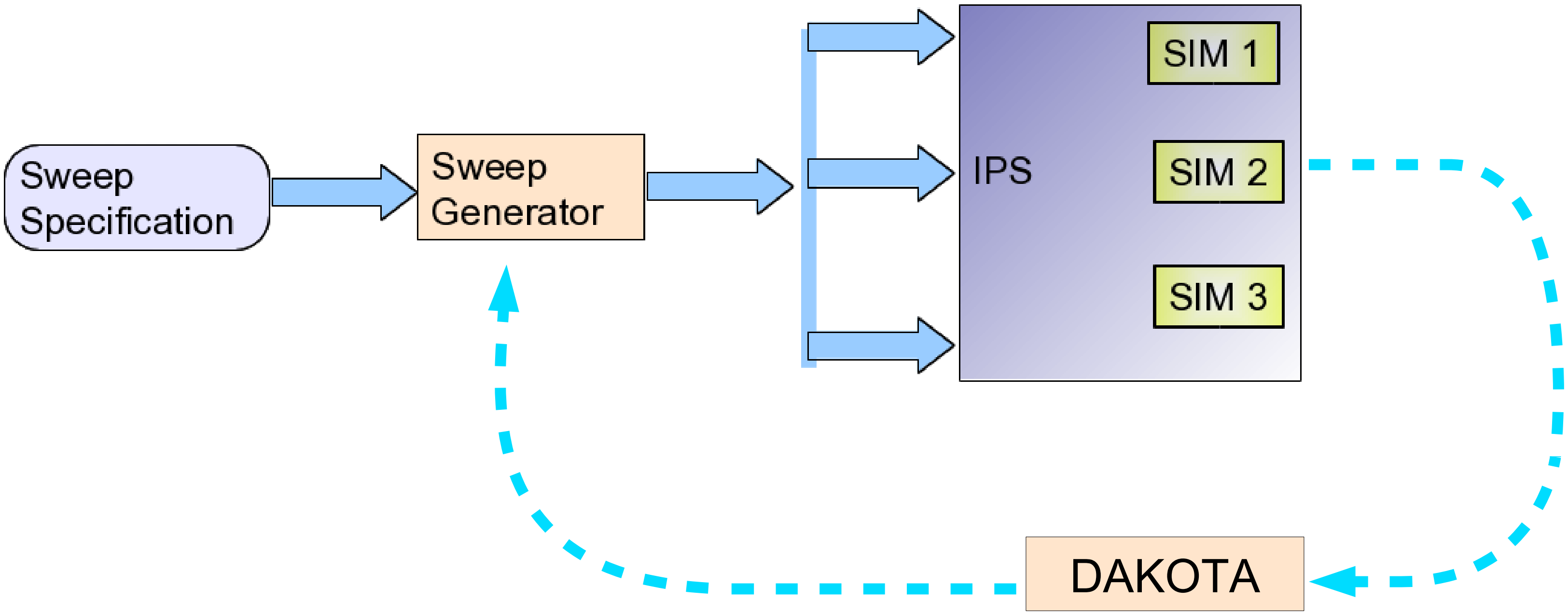
- Single blocking and non-blocking task invocation.
  - Component manages outstanding tasks.
- **Task Pools:**
  - Create n>1 tasks to be managed by the framework.
  - Framework manages scheduling, resource allocation, and task execution for all tasks in the pool.
    - Blocking: Wait for all of them to finish.
    - Non-Blocking: Query for finished tasks periodically.
- Can be used to implement localized parameter sweeps or other pleasantly parallel component tasks.

Parameter Sweep using the IPS – Phase 1



Pre-defined parameter set that covers the parameter space for all components in the simulation.

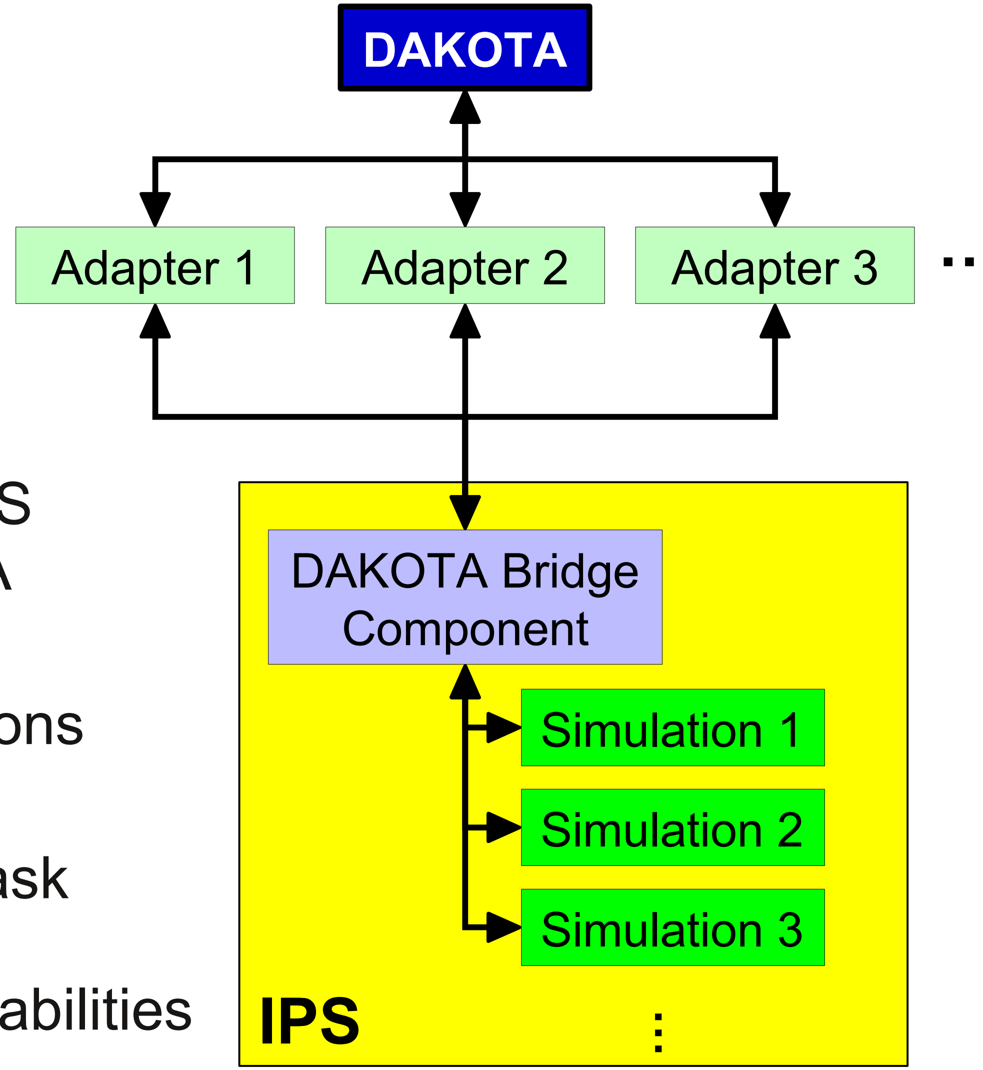
Parameter Sweep using the IPS – Phase 2



Dynamic generation for design optimization using the DAKOTA tool kit (from Sandia National Lab)

Integrating DAKOTA with the IPS

- DAKOTA Toolkit
  - Optimization
  - Uncertainty Quantification
  - Parameter Estimation
  - Parameter Studies
- Dynamic instantiation of IPS simulations under DAKOTA control
- Adapters map IPS simulations to DAKOTA specs
- Exploit IPS resource and task management and multiple concurrent simulations capabilities



Highlights of DAKOTA Integration

- Single Instance of the IPS framework.
- Dynamic instantiation of an entire coupled simulation, as directed by DAKOTA.
- Multiple simulations share available framework resources, mediated through the resource manager.
- Concurrency in DAKOTA-IPS runs:
  - User specific: for parameter sweeps.
  - User+algorithm specific: for optimization
  - Concurrency may depend on the number of parameters.

Summary

- IPS provides a highly flexible, robust environment for coupled simulations.
- Light weight, portable environment that works on platforms from laptops to petascale machines.
- Adapting stand-alone codes for us in the IPS is fairly straight forward
  - Greatly simplifies debugging for coupled simulations.
- Support for concurrent simulations enable efficient deployment of parameter sweeps and optimization regiments.