The SWIM Integrated Plasma Simulator (IPS) Framework For Loosely Coupled Fusion Simulations

Wael R. Elwasif
Oak Ridge National Laboratory
And the SWIM Project Team

The SWIM Project

- Center for Simulation of RF Wave Interactions with Magnetohydrodynamics (SWIM).
- One of three US 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

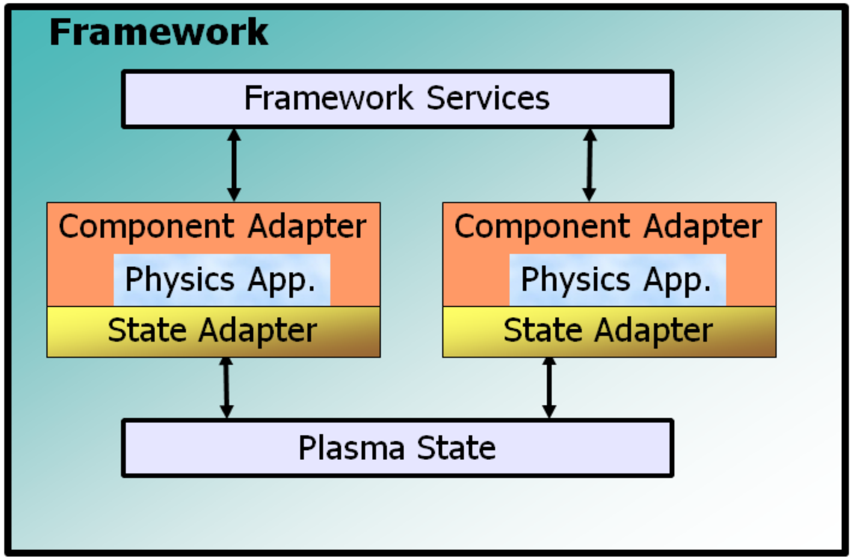
- Minimize level of effort to bring in physics codes
  - Avoid bifurcation of physics modules – not different SWIM/stand-alone versions
  - Wrappers around unmodified codes
  - Use application native I/O, transform to shared data using ***state adapters***
- Design for broader range of integrated simulation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

Computing Philosophy & Approach (2)

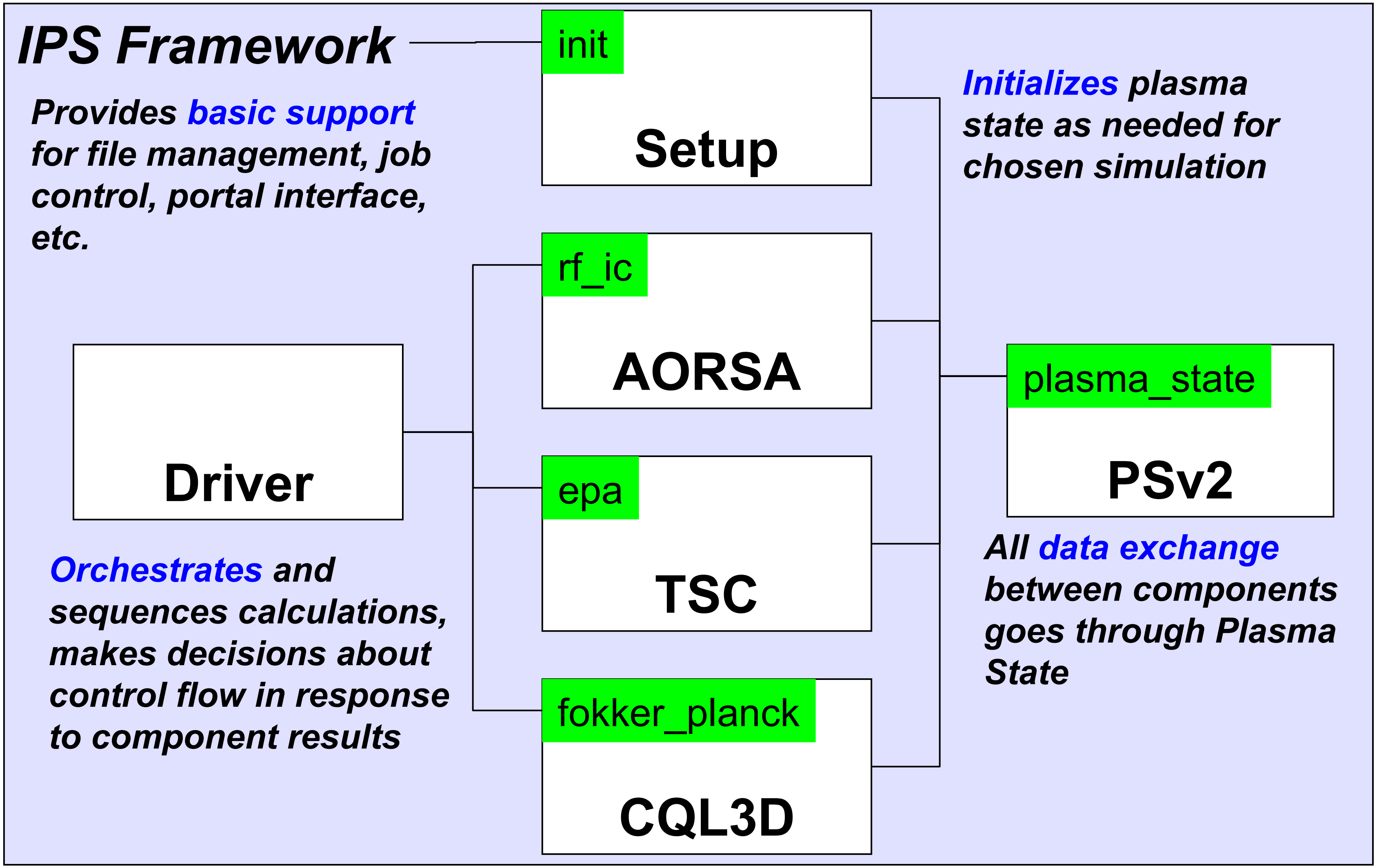
- Design for multiple implementations of each physics component
  - ***Code***-based interfaces vs ***Physics***-based interfaces
  - Accommodate reduced models, inter-comparisons (V&V), etc.
- Component Approach
  - Based on Common Component Architecture concepts
  - Simplified implementation, focusing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

The Integrated Plasma Simulator (IPS): Design Features

- Simulation framework
  - Light weight, Python-based implementation (***4328 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**"*

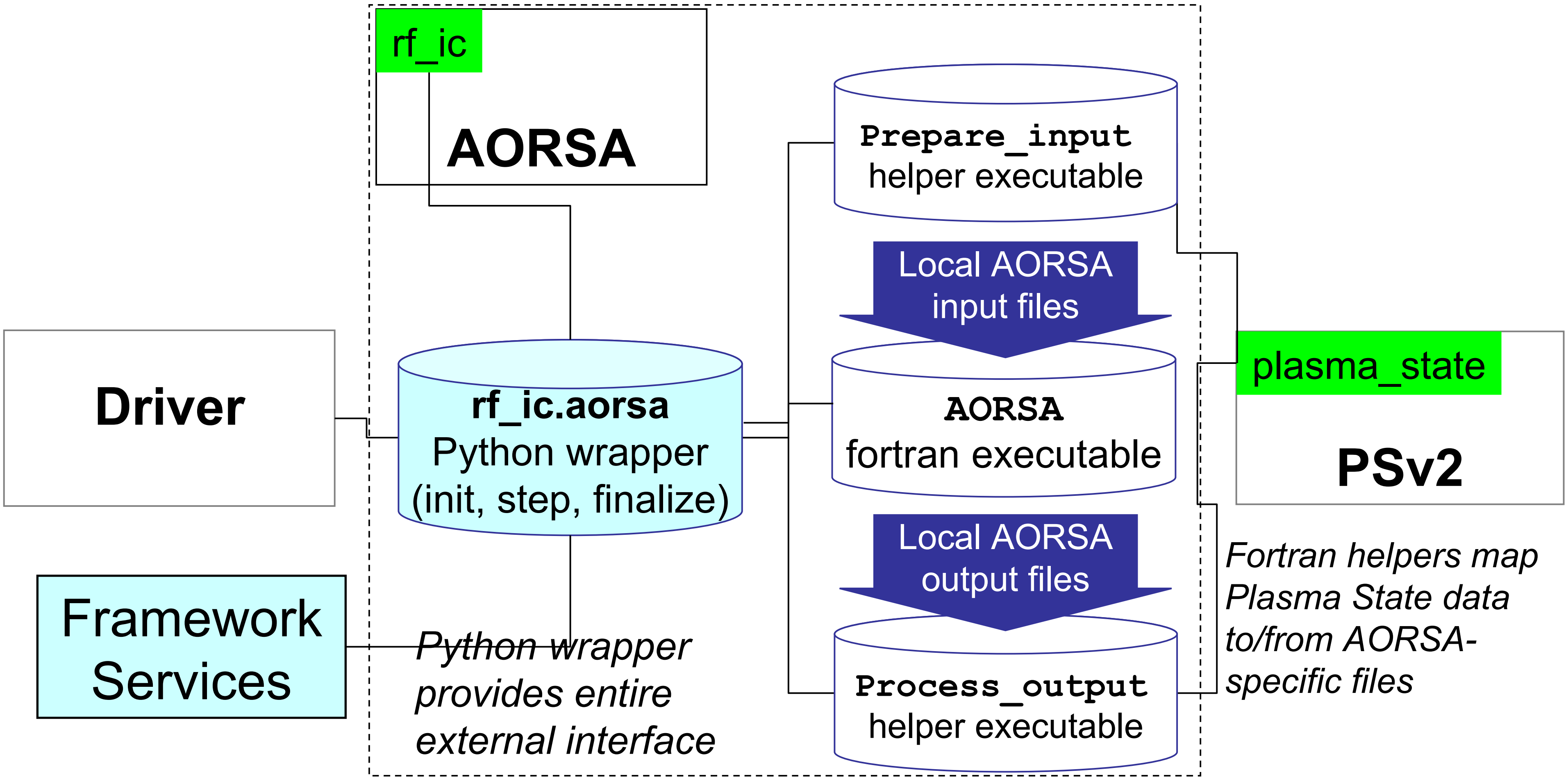


Schematic of a *Classical* IPS Application



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

Drilling Down: Typical Component Structure

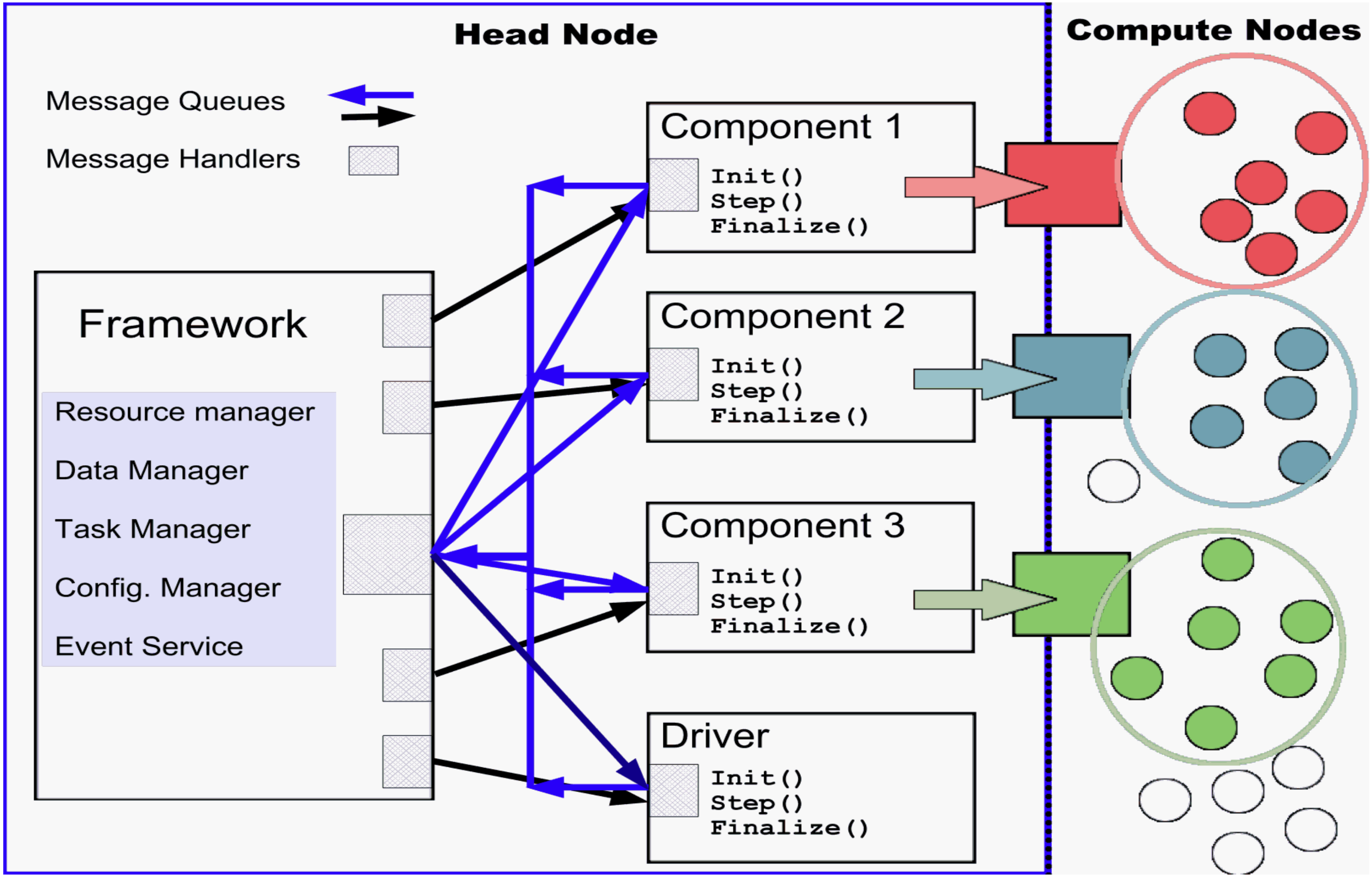


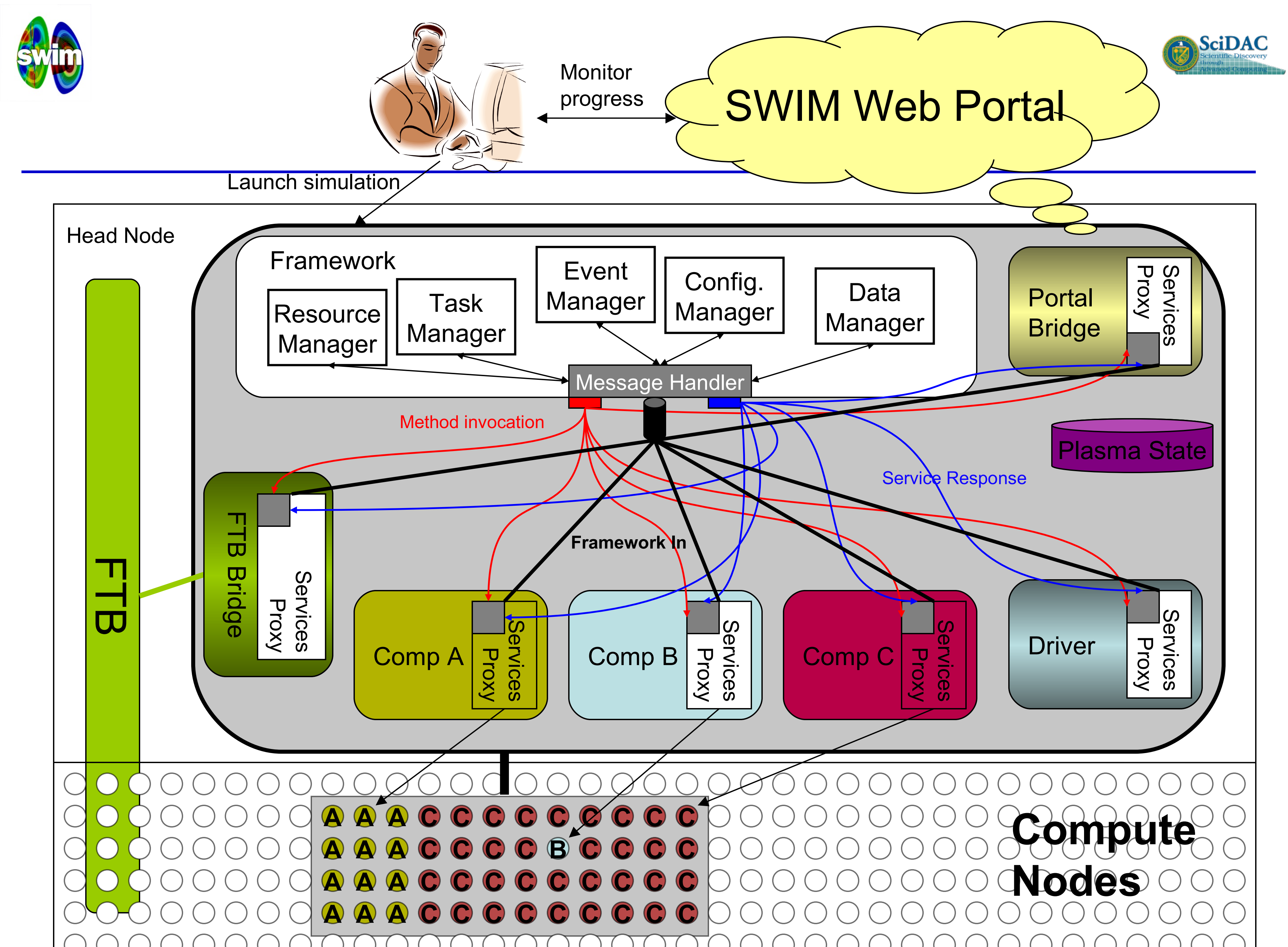
IPS design/specifications say nothing about internal implementation of components.

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

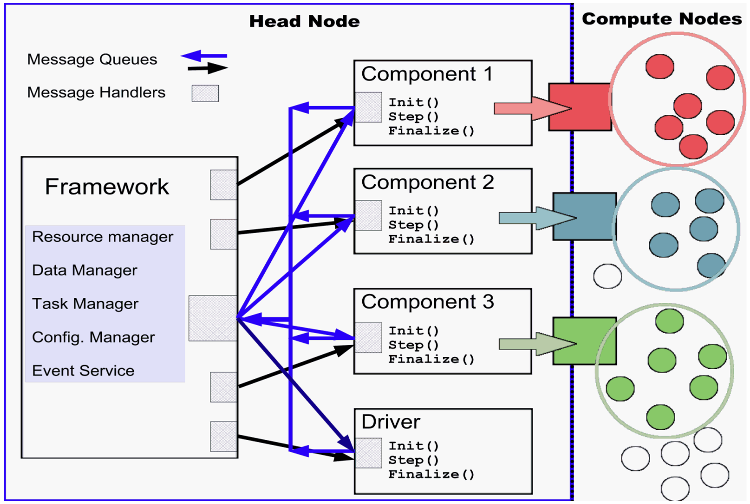
IPS Execution Environment





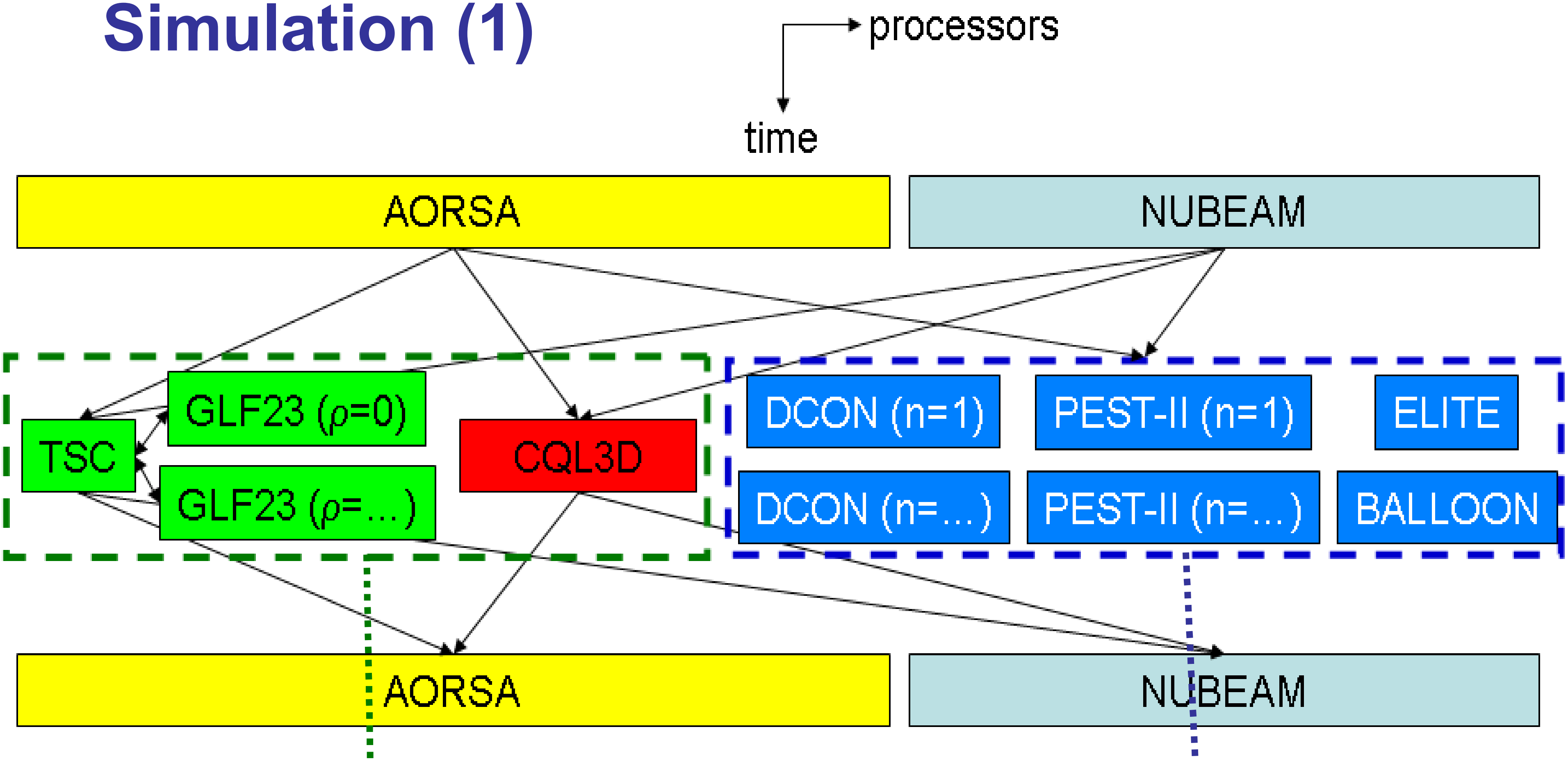
Monitor progress
SWIM Web Portal
Launch simulation
Head Node
Framework
Event Manager
Config. Manager
Data Manager
Resource Manager
Task Manager
Message Handler
Portal Bridge
Services Proxy
Method invocation
Plasma State
Service Response
FTB
FTB Bridge
Services Proxy
Framework In
Comp A
Services Proxy
Comp B
Services Proxy
Comp C
Services Proxy
Driver
Services Proxy
Compute Nodes

IPS Supports *Four* Levels of Parallelism



- ***Parallel Tasks*** (physics applications)
  - Used routinely – SWIM physics applications vary in parallelism
- ***Concurrent task execution***
  - A component can launch multiple concurrent tasks
  - Basis of ***Parareal*** implementation (discussed later)
  - Also useful to (for example) create a component that parallelizes over flux surfaces implemented with a physics code that treats one surface at a time
- ***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

Concurrent Multitasking for a Complex Simulation (1)

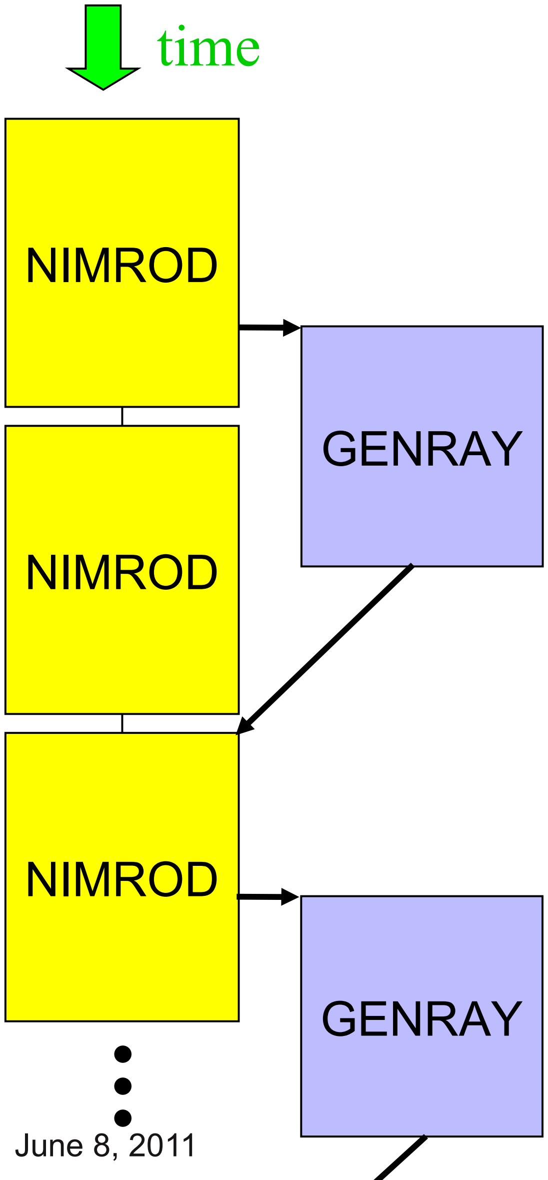


Equilibrium and profile advance for step t, including parallel anomalous transport tasks for each flux surface, all running concurrently with the Fokker Planck component.

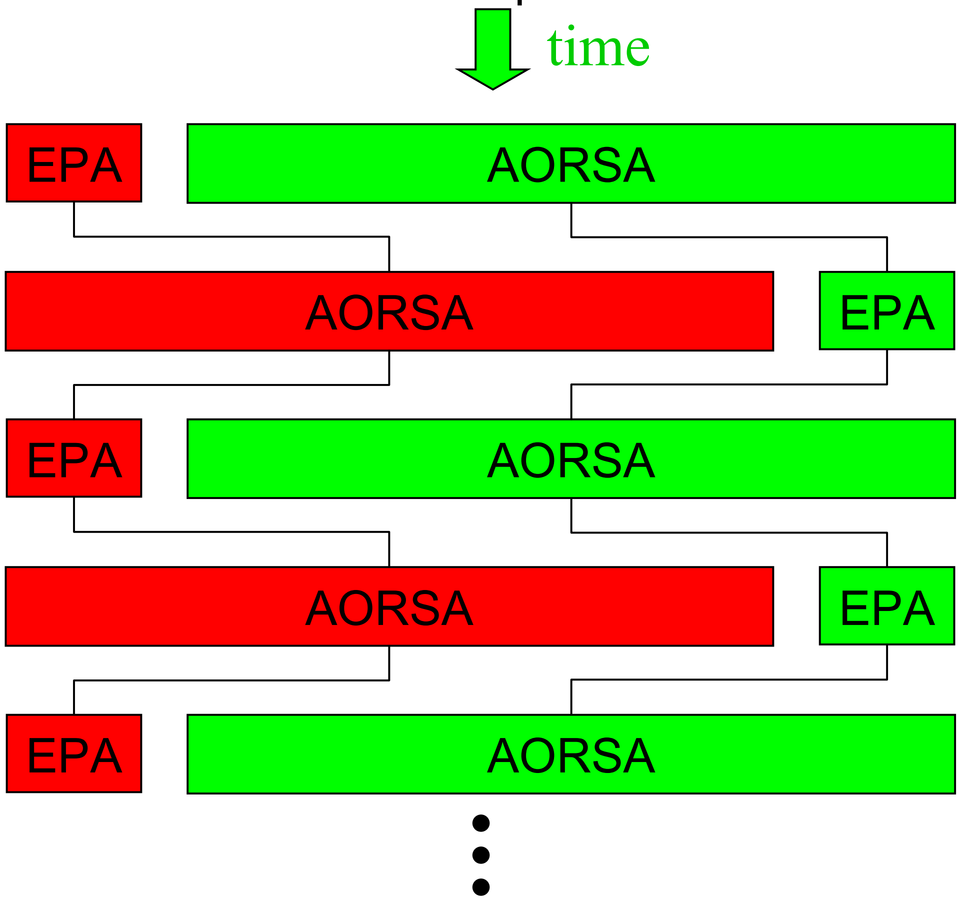
Multiple stability analysis components running on multiple toroidal modes, all running concurrently on t-1 results.

Concurrent Multitasking (2) and Multiple Simulations

Initial Slow MHD Scenario



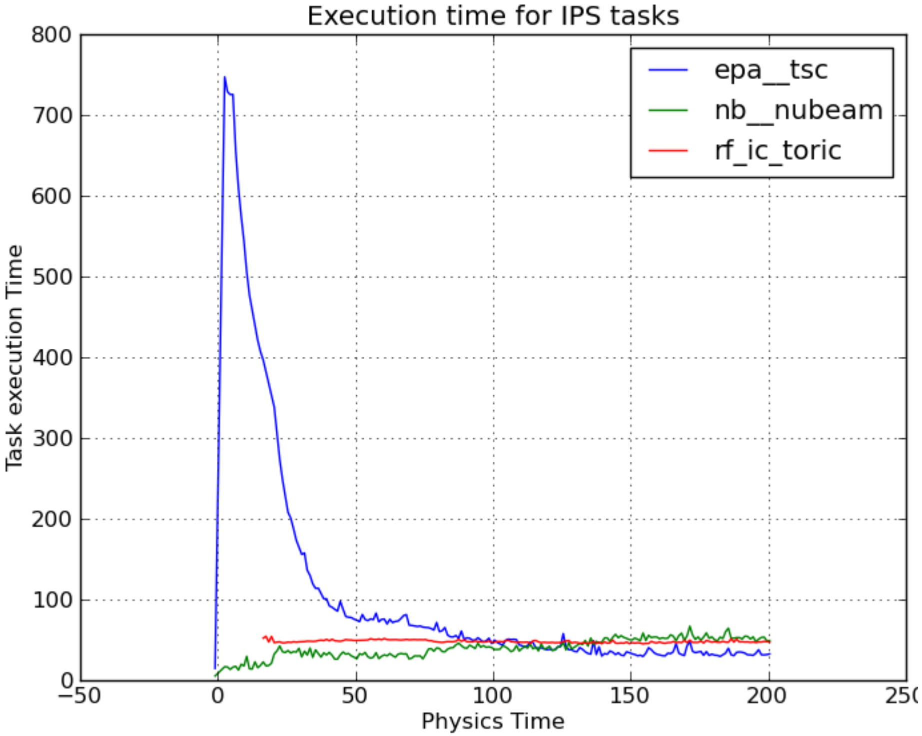
"Ensemble" of Coupled Simulations



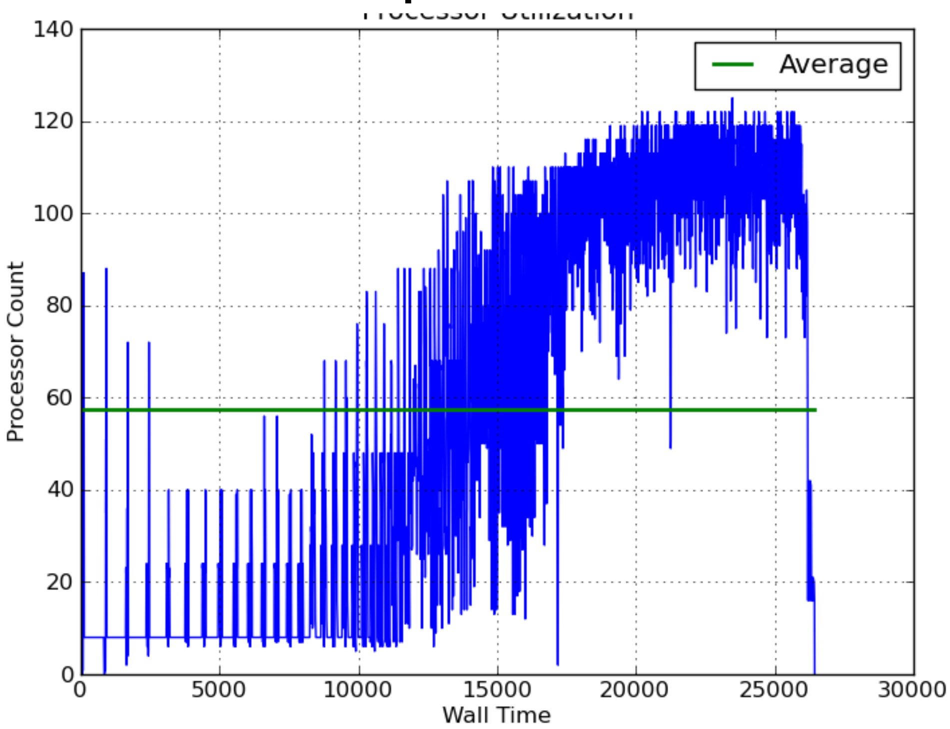
Two (or more) simulations (i.e. multiple pedastal heights) share the same processor allocation, running out of phase to maximize utilization.

Multiple Simulations In Action
Multiple Simulations Cray XT5 at NERSC

Execution time for IPS tasks



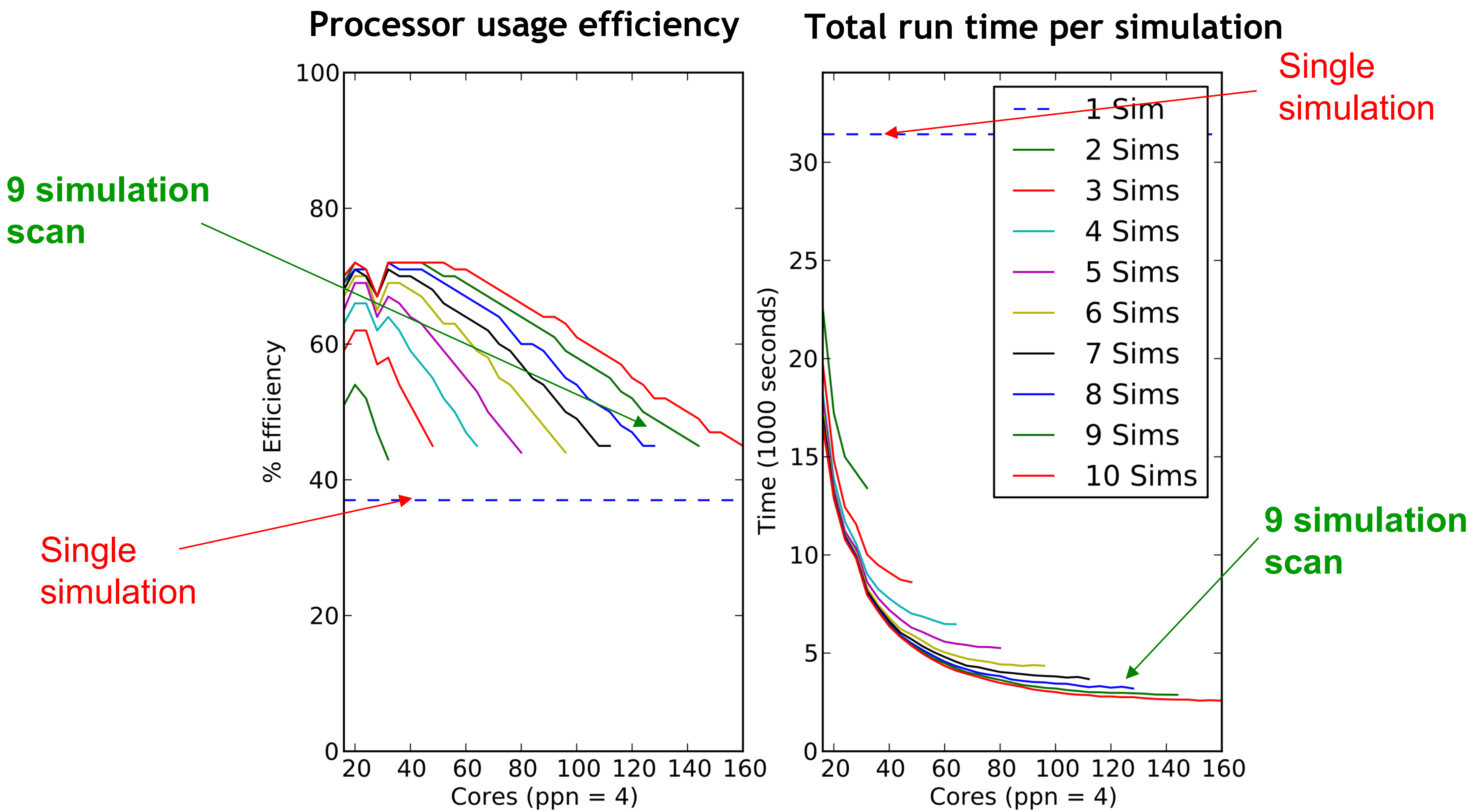
Processor utilization for 9-simulation parameter scan



- 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?

Simulating Resource Utilization

A model based on mean and standard deviation of task execution time in different simulation phases predicts performance



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.
- Used in “standard” implementation of Parareal

Event-Based Control Flow in IPS Simulations

- One (or more) components acting like "***servers***"
  - May require slight modification to underlying codes if they too will run as servers.
- Asynchronous events published to pre-defined topics (channels), when an event of interest occurs.
  - Topics and event payload agreed upon among participating entities.
- Components subscribe to topics of interest and periodically check for published events.
  - Pull model to avoid threading complications

Case Study: Parareal Using the IPS

- Parareal: Parallel In Time
  - Iterative parallelization (domain decomposition) of time in time-dependent problems.
- Requirements
  - A "f*ast*" coarse solver (***G***), and an "*accurate*" fine solver ***F***.
  - ***G*** should have enough physics, resolution, ..etc to propagate the "*essential*" physics forward in time.
- Also needed:
  - Convergence measure.
  - Operators to transform the states of ***G*** and ***F*** to inputs for ***F*** and ***G***, respectively.

Parareal – Prior Art and Current Work

- Y. Maday, G. Turinici, C. R. Acad. Sci. Paris, Ser. I 335 (2002) 387–392.
- L. Baffico, S. Bernard, Y. Maday, G. Turinici, G. Zérah, Parallel in time molecular dynamics simulations, Phys. Rev. E 66 (5) (2002) 057706.
- ...
- ***D. Samaddar, D.E. Newman, R. Sánchez, Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm, J Comp Phys 229(18) (2010) 6558***
- ***L. Berry, W. Elwasif, J. Reynolds-Barredo, D. Samadar, R. Sanchez, and D. E. Newman, Event-Based Parareal: A data-flow based implementation of Parareal, In Preparation.***

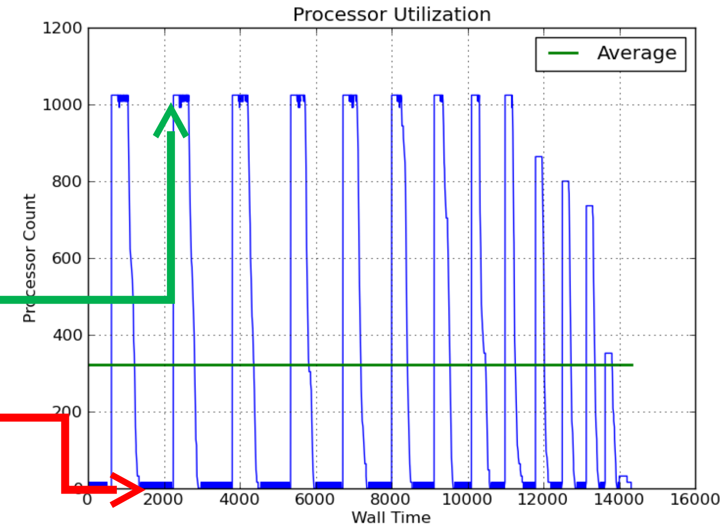
Classical Parareal using the IPS

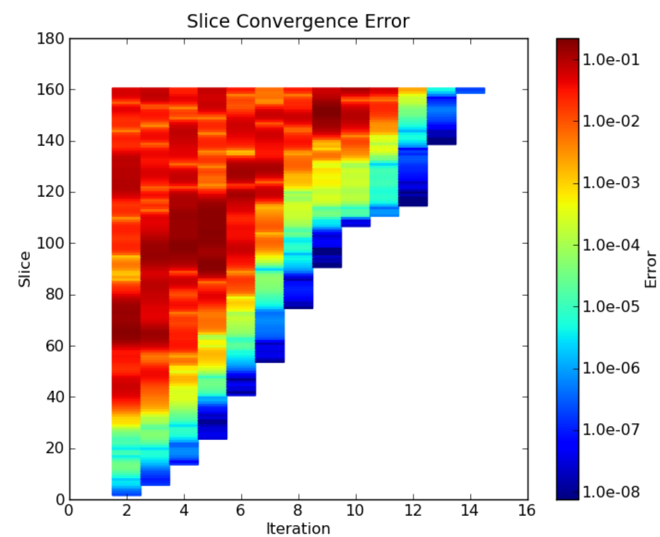
- Components: Driver, Fine Solver, Coarse Solver
- Driver: Flow control
- Coarse Component:
  - Evaluate (sequentially) coarse solution for the entire time domain.
- Fine Component:
  - Use IPS's ***task pool*** to evaluate fine solution in parallel for the entire time domain (as permitted by available compute resources).
- Framework manages resource allocation and dispatching of tasks in the task pool.

IPS Multi-Task Parallelism for Parareal

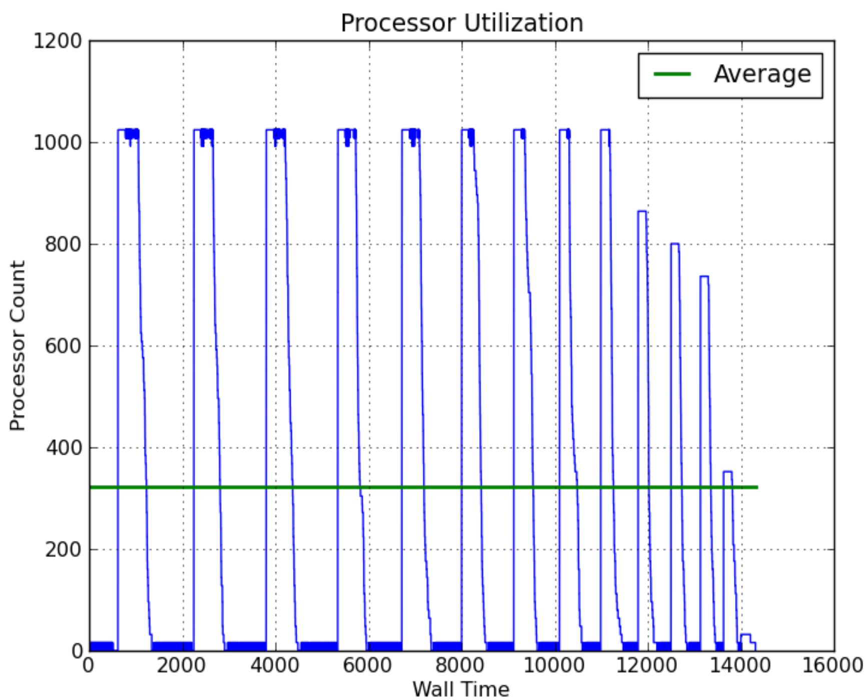
- Two levels of parallelism
- Fine solver executes a task for each u converged slice, concurrently (***forall***)

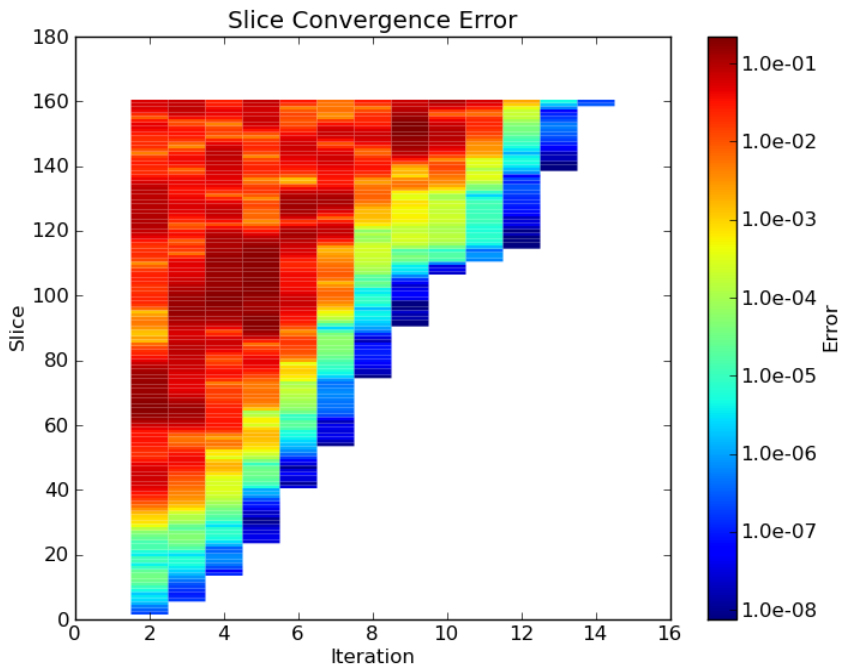
```
first_slice = 1
num_converged = 0
for iteration = 1, max_iterations
   for slice = first_slice..num_slices
      coarse_solve(iteration, slice)
   forall slice = first_slice..num_slices
      fine_solve(iteration, slice)
   for slice = first_slice..num_slices
      test_convergence(iteration, slice)
   num_converged +=
      first_non_converged_slice - first_slice
   if (num_converged == num_slices)
      end // SUCCESS
   else
      first_slice = first_non_converged_slice
end //Failed to converge in max_iteration
```





Classical Parareal using the IPS



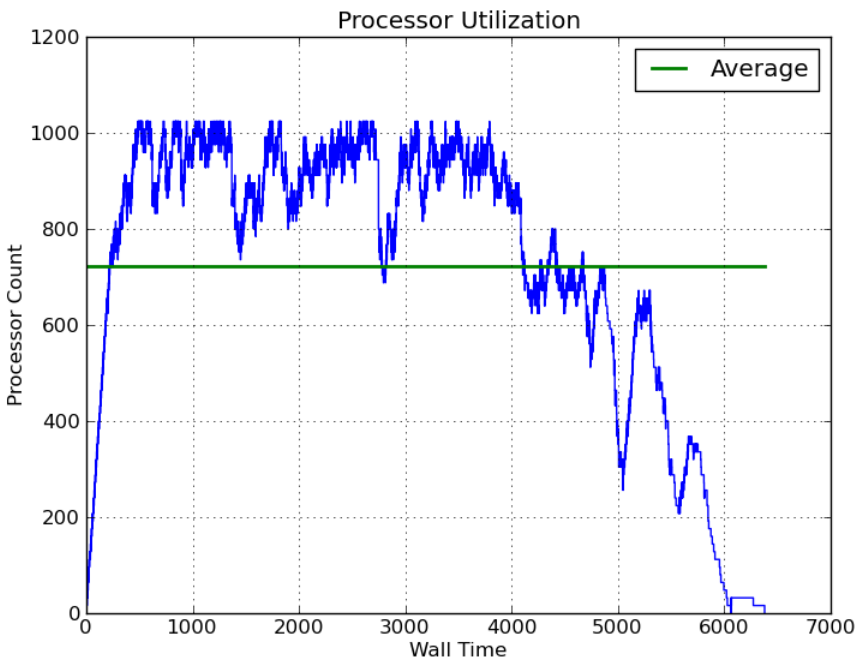


It is non-trivial to find really fast and “good enough” coarse solvers.
We can lower the barrier by dispatching tasks ***asynchronously***.

Asynchronous Event-Based Parareal

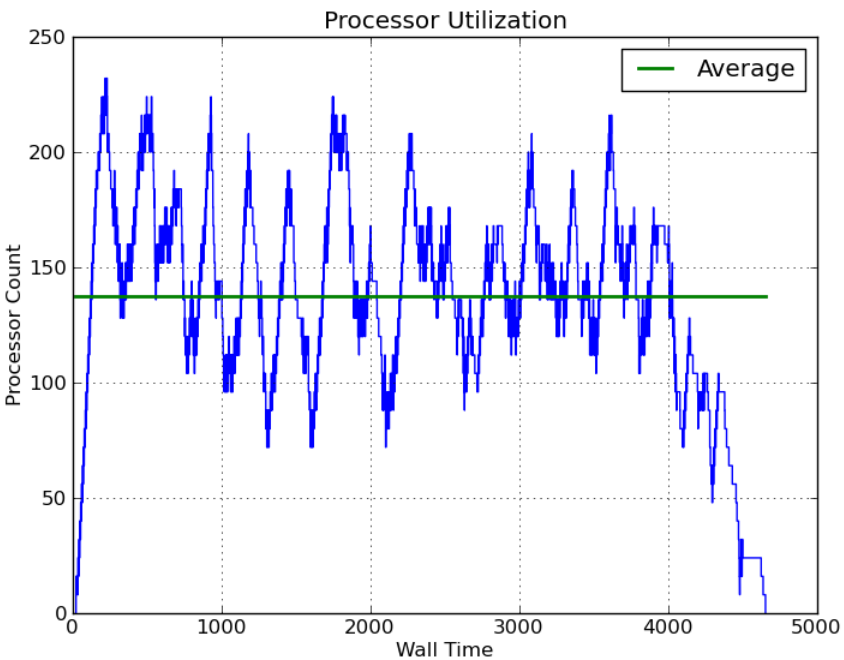
- Three levels of parallelism
  - Concurrent components, concurrent parallel tasks.
- Three "***server***" components:
  - Coarse, Fine, and Converge
- Driver component merely initiates the simulation.
- Implicit synchronization using IPS asynchronous events published to pre-defined topics (channels).
- A ***task*** in a time slice (fine solve, coarse solve, convergence check) is started as soon as its prerequisites are satisfied.
- Components manage re-launch of tasks when lack of resources prevent immediate execution.

Improved Utilization And Run Time



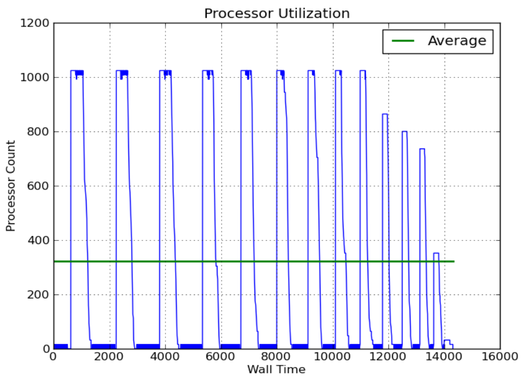
But why execute the coarse (and fine) tasks when "*the reach*" of the coarse solver is obviously diminished ??

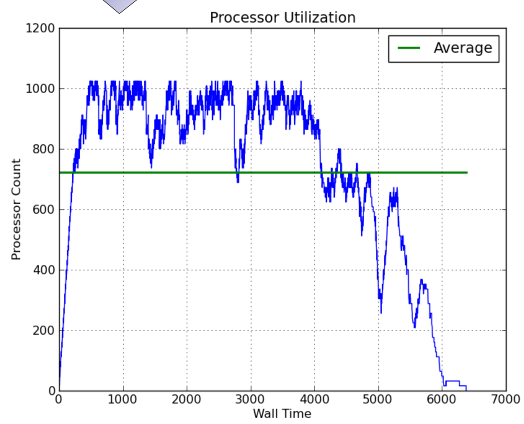
Dynamic Slice Addition In Parareal

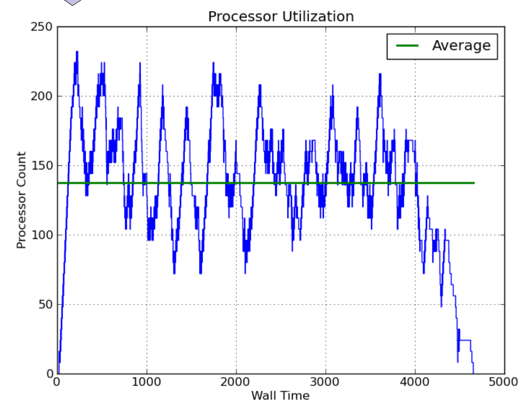


Significant reduction in required resources, while maintaining convergence properties.

IPS Flexible Task Parallelism Improves Utilization and Solution Time



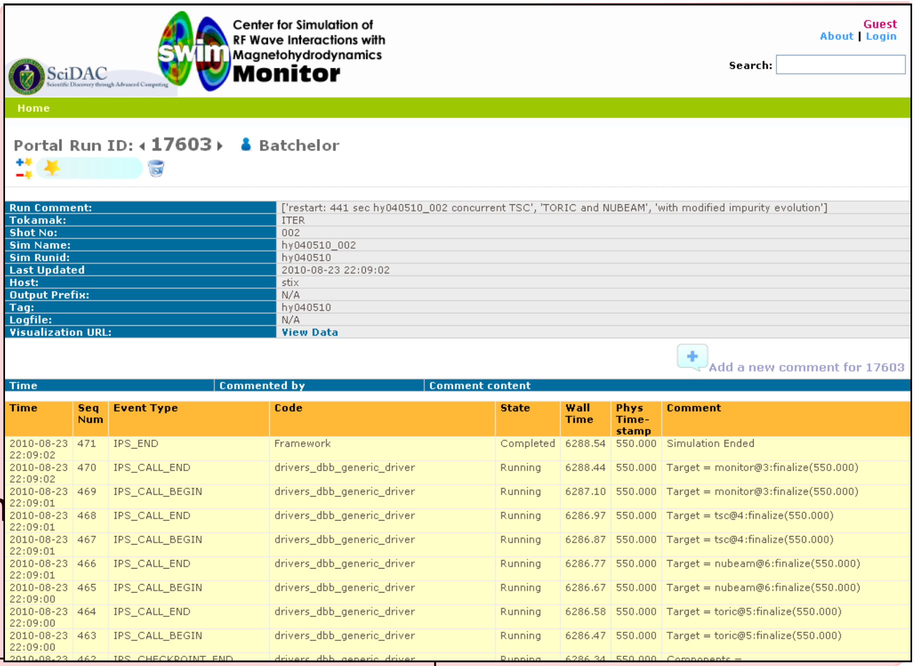




No change to the "core" solvers, only in the task model and execution flow control.

SWIM Portal – Real-Time Job Monitoring

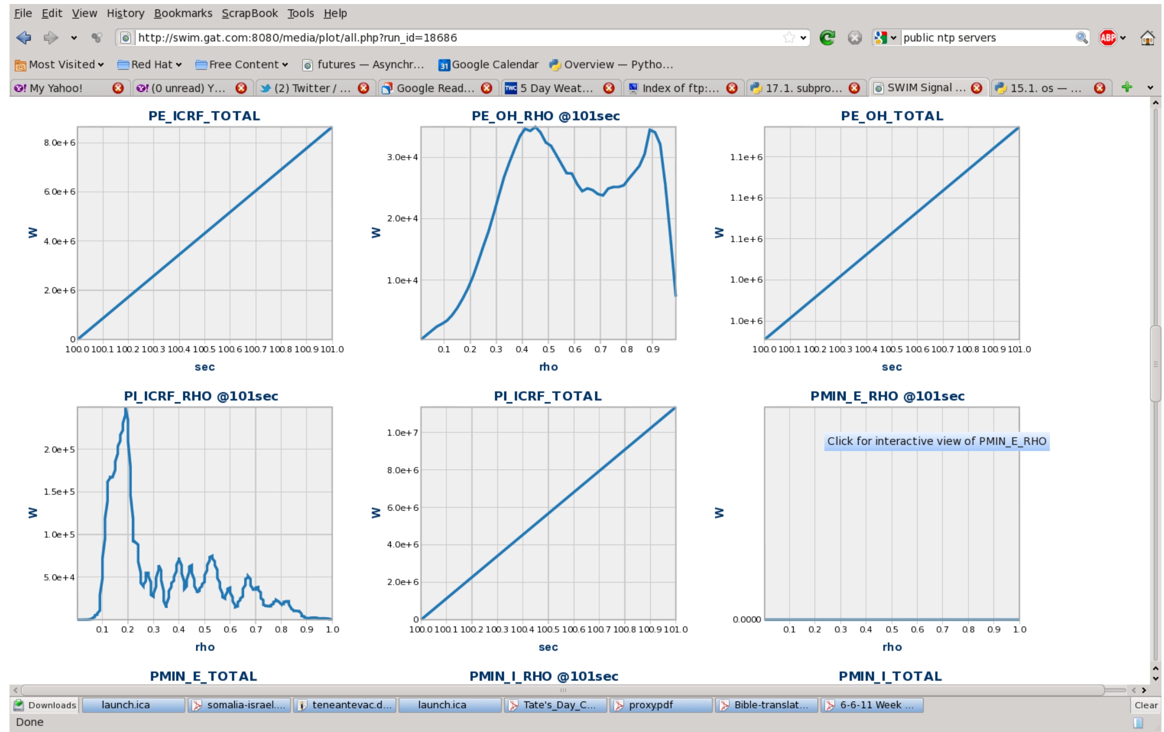
- http://swim.gat.com:8080/monitor
  - Server hosted at General Atomics
- Portal usage is completely optional – IPS jobs will run without it
- Real-time monitoring of job progress
- IPS framework instrumented to automatically provide portal with status information based on execution flow
  - Component method invocations
  - Data management operations
  - Task failures
  - Messages sent via simple http protocol
- Components can provide additional information to portal



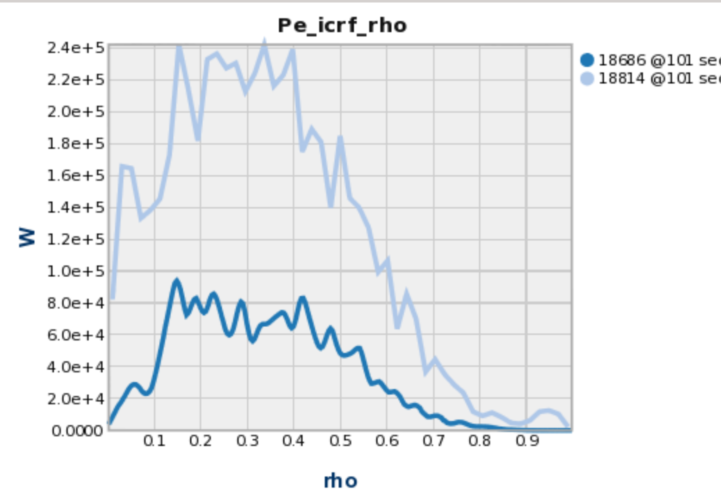
Run page provides history of all messages received by portal

Main page summarized lists all recent IPS runs with latest status information

SWIM Portal – Real-Time Monitoring & Analysis

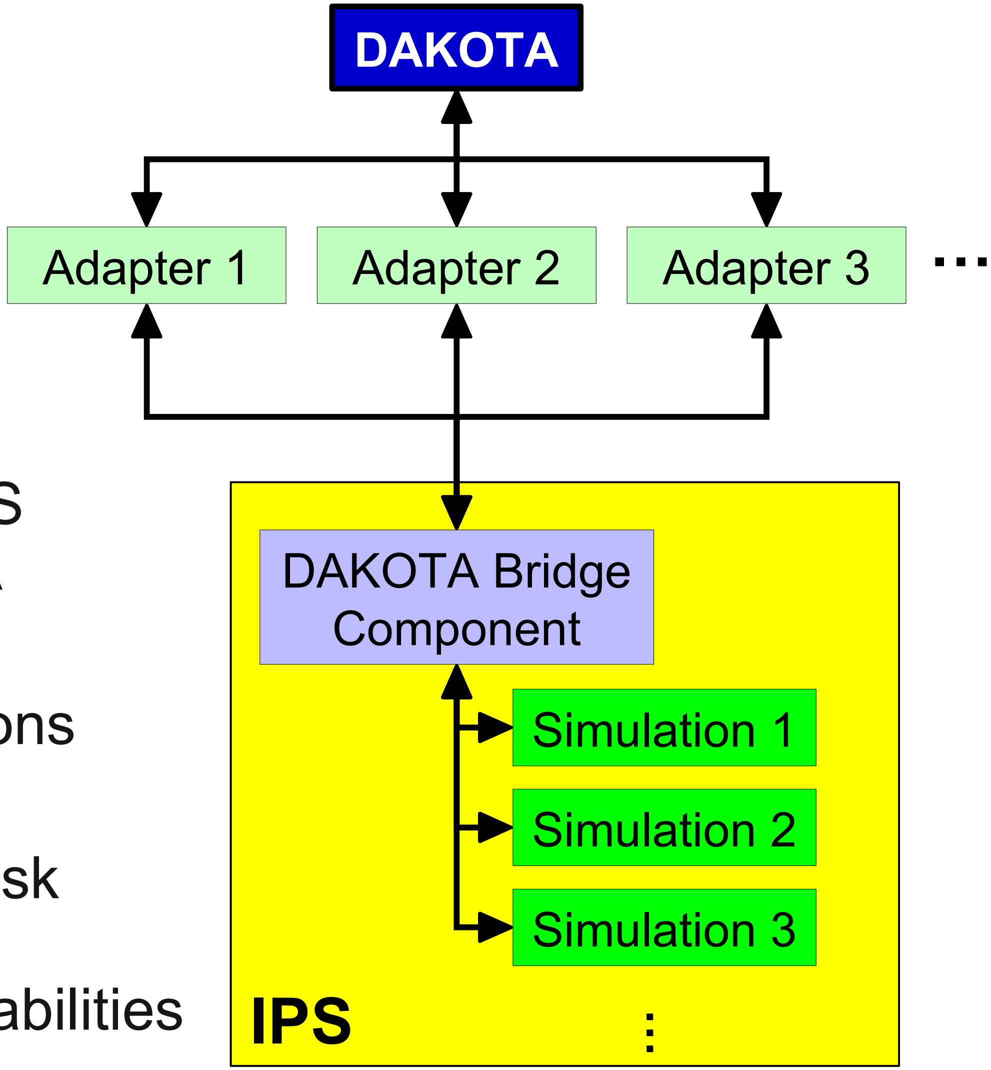


- Simulation monitoring and summary analysis via portal-based data store and web browser
- Monitoring component (in IPS) exports data of interest to NetCDF file
  - Separate from Plasma State, smaller
- Portal imports monitor file from web-accessible space on simulation platform
- Simulation summary uploaded to MDS+, available for analysis.
- HTML5 based summary graphs
- Comparisons (simulation & Experimental)



Parameter Sweeps & Optimization - DAKOTA

- 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



Summary

- IPS provides a simple, light-weight framework for loosely coupled, file-based, coupled simulations.
  - When data exchange size and frequency allow.
- Adapting stand-alone codes for us in the IPS is fairly straight forward
  - Greatly simplifies debugging for coupled simulations.
- Multiple levels of concurrency provides flexibility to exploit parallelism and improve resource utilization.
- Light weight, highly expressive Python environment simplifies component development
  - Total size of four Parareal components: ***913 LOC***

Some Meta Thoughts

- Trade off between ***flexibility*** and ***robustness*** should not be rigid
  - Different needs during different phasis in a project
- Incurring ***Technical Debt*** may be necessary
  - As long as the interest doesn't grow too high
- ***Really Big*** machines are around the corner
  - Today's supercomputer is tomorrow's cluster.
- ***Capability*** vs ***Capacity*** Computing
  - We will probably need both, maybe on the same machine.

Questions?