# The SWIM Plasma State in Multi-Physics Fusion Simulations


**Don Batchelor, Doug McCune**


**Integrated Modeling Technology workshop**

**June 9, 2011**

**ITER Site, Cadarache FR**


**D. B. Batchelor, L. A. Berry, E. F. Jaeger** – *ORNL Fusion Energy*

**D. E. Bernholdt, E. D'Azevedo, W. Elwasif, S. Foley** – *ORNL Computer Science*

**S. C. Jardin, E. Feibush, D. McCune, J. Chen, M. Chance, J. Breslau** – *PPPL*

**G. Abla, D. P. Schissel** – *General Atomics ,* **R. W. Harvey** – *CompX*

**R. Bramley**– *Indiana University,* **D. Keyes** – *Columbia University,* **D. Schnack** – *U. Wisconsin*

**P. T. Bonoli, J. Ramos, J.Wright** – *MIT,* **S. Kruger, T. Jenkins** – *TechX*, **G. Batemar**

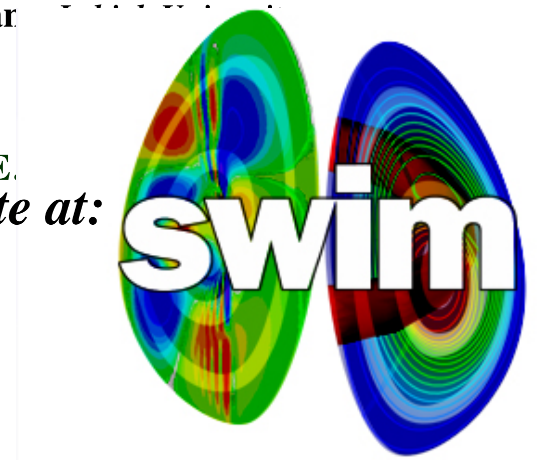
**Unfunded participants:**

**L. Sugiyama** – *MIT,* **J. D. Callen, C. C. Hegna, C. Sovinec** – *University of Wisconsin,* **E.**

**H. St. John** – *General Atomics,* **A. Kritz** – *Lehigh Univ.*


*See our fun website at:*

**www.cswim.org**

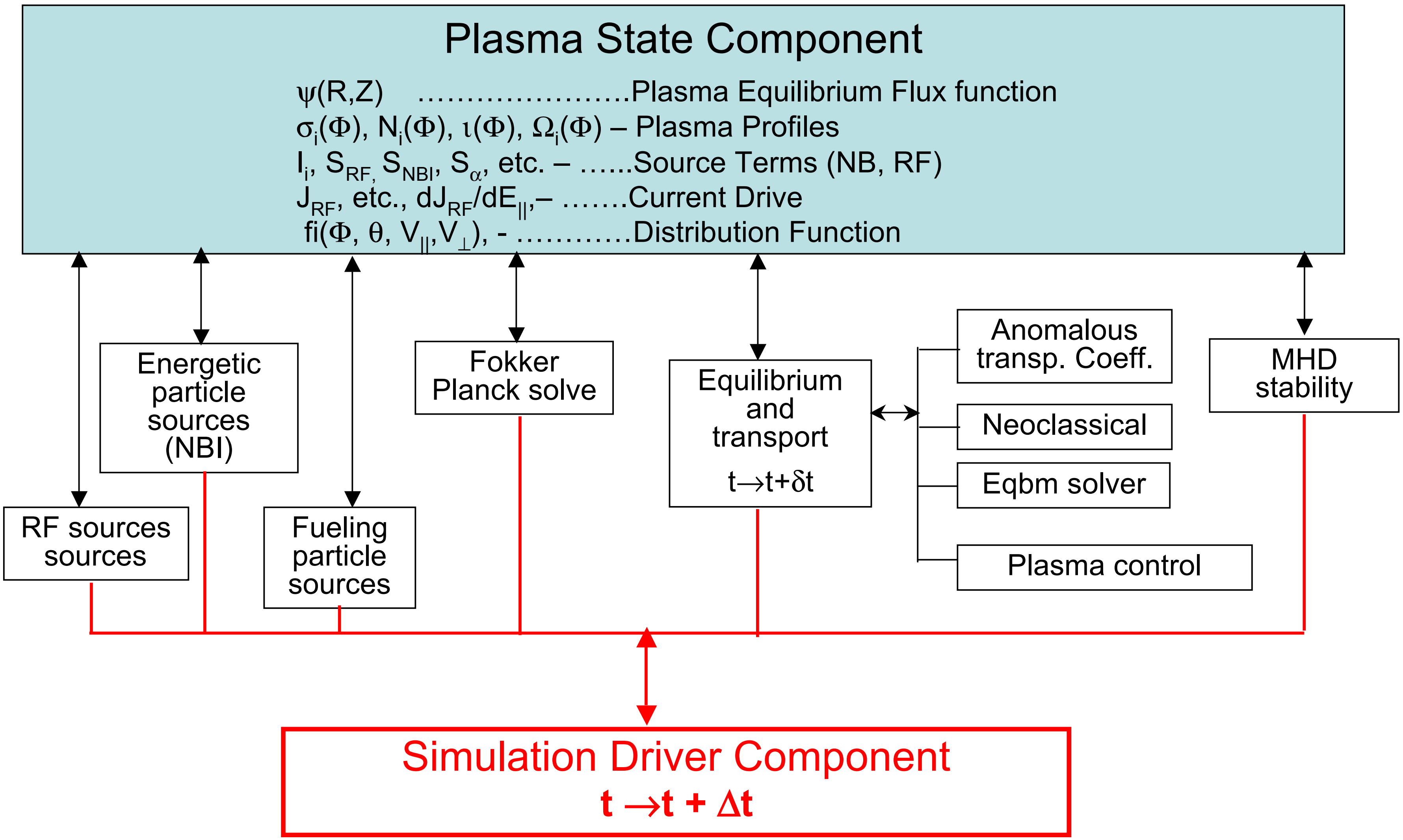


# All Simulation data exchanged between components goes through Plasma State

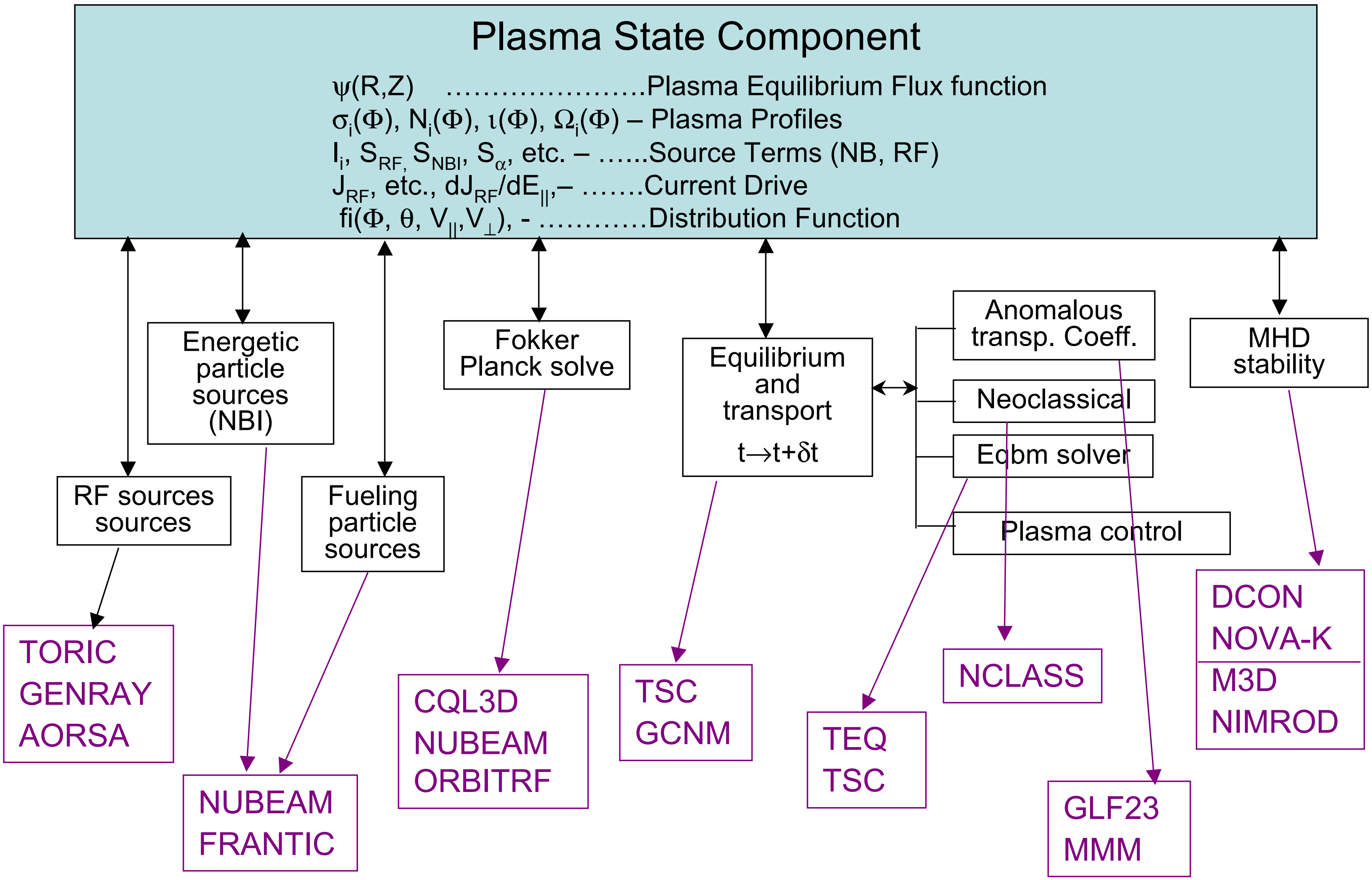
- **Fortran 2003 Module – supports in-memory or file-based data exchange (netCDF)**
- **Very simple user interface → functions: get, store, commit, merge partial**
- **Other powerful functions available, but not required → e.g. grid interpolation**
- **Supports multiple state instances, partial states**
- **Code is automatically generated from state specification text file → ease and accuracy of update**
- **Some types of data we haven't dealt with yet → distribution functions are just code dependent filenames → keep plasma state objects small**
- **Being shared with other projects**
  - **Component-to-component data exchange in TRANSP and PTRANSP**
  - **Coupling of neutral beam and fusion product sources to FACETS C/C++ transport driver**
  - **Anomalous transport data TGYRO**

**More generally "*plasma state*" consists of a set of files that are managed and transported as group by the framework – eg eqdsk files, distribution functions**

# Integrated Plasma Simulator design – Physicists view



# Integrated Plasma Simulator design – Components are implemented by mature, well-validated codes



# Plasma State Object Contents

- **Member elements are scalars and arrays of:**
  - **REAL(KIND=rspec), equivalent to REAL*8.**
  - **INTEGER.**
  - **CHARACTER*nnn – strings of various length.**
- **Flat structure, scalars and allocatable arrays:**
  - **All object members are primitive fortran types.**
- **Maximum element identifier length = 19**
  - **Alphabetic 1st character; then alphanumeric + "_"**
  - **26*37**18 = 4.39*10**29 possible element names**
- **Semantic elements (constituted by one or more primitive PS object data elements):**
  - **Item lists (for example: list of neutral beams).**
  - **Species lists (for example: list of beam species).**
  - **Grids (for example: radial grid for neutral beam physics component).**

# Plasma State Physics Components – 13 at present

**Each data element is assigned to a physics component – but not write restricted**

- **Plasma (pertaining to thermal species profiles)**
- **EQ (pertaining to MHD equilibrium)**
- **Heating components: NBI, IC, LH, EC**
- **FUS (fusion products)**
- **RAD (radiated power)**
- **GAS (neutral species)**
- **RUNAWAY**
- **LMHD**
- **RIPPLE**
- **ANOM**

# Plasma State Sections

- Machine_Description
  - Time invariant, shot invariant for tokamak-epoch
- Shot_Configuration
  - Time invariant (e.g. species lists).
- Simulation_Init
  - Time invariant (e.g. grids & derived species lists).
- State_Data – non-gridded scalars and arrays.
- State_Profiles – arrays of gridded profiles.

# For Example: NBI Component

- **Machine description:**
  - List of neutral beams: Names, detailed geometry, energy fraction tables.
- **Shot configuration:**
  - Injection species for each neutral beam.
- **Simulation initialization:**
  - Beam species list, derived from shot configuration.
  - Radial grid for NBI profile outputs.
- **State Data**
  - Neutral beam injector powers and voltages.
  - Injection fractions (full/half/third energy beam current fractions).
- **State Profiles**
  - Beam ion densities, <Eperp>, <Epll>.
  - Main Heating: Pbe, Pbi, Pbth.
  - Main Torques: Tqbe, Tqbi, TqbJxB, Tqbth.
  - Particle source profiles, all thermal species.
  - Current drive, beam deposition halo profiles, etc.

# PS: What's in and What's not

## Included in Plasma State:

- Physics data shared between components:
  - E.g. neutral beam powers set by plasma model.
  - Profiles returned by NBI, used by plasma model.
- Common static data
  - Machine description data
  - Metadata – simulation name, shot number

## Not included in Plasma state

- Anything related to specific code implementation:
  - Code algorithm switches or internal grids
  - E.g. NPTCLS for NUBEAM implementation of NBI.
- Data specific to a single component only
  - E.g. Monte Carlo code state as particle lists.
- So far profiles of rank > 2 have not been used.

# PS Interpolation Services

- Components provide data on their native grids.
- Interpolation typically required for use.
- Plasma State definition provides “recommended” interpolation method.
  - Spline, Hermite, piecewise linear, zone step functions
  - Conservative “rezoning” of profiles
    - For densities & sources conserve #, #/sec, Watts, …
    - For temperatures conserve volume integrated n*T.
- Fortran implementation: xplasma, pspline (NTCC).

# PS I/O Services

- **PS_get_Plasma_State – read all from NetCDF**
- **PS_store_Plasma_state – write all to NetCDF**
- **PS_read_update_file – read a Plasma State update from a specified component**
  **→ read part of state**
- **PS_write_update_file – a Plasma State update from a specified component**
  **→ write part of state**

**Partial state read/write enables  non-overlaping components to run concurrently and can reduce volume of data traffic**

**Hash-code facility enables writing only data that has changed and comparison between two different Plasma State objects**

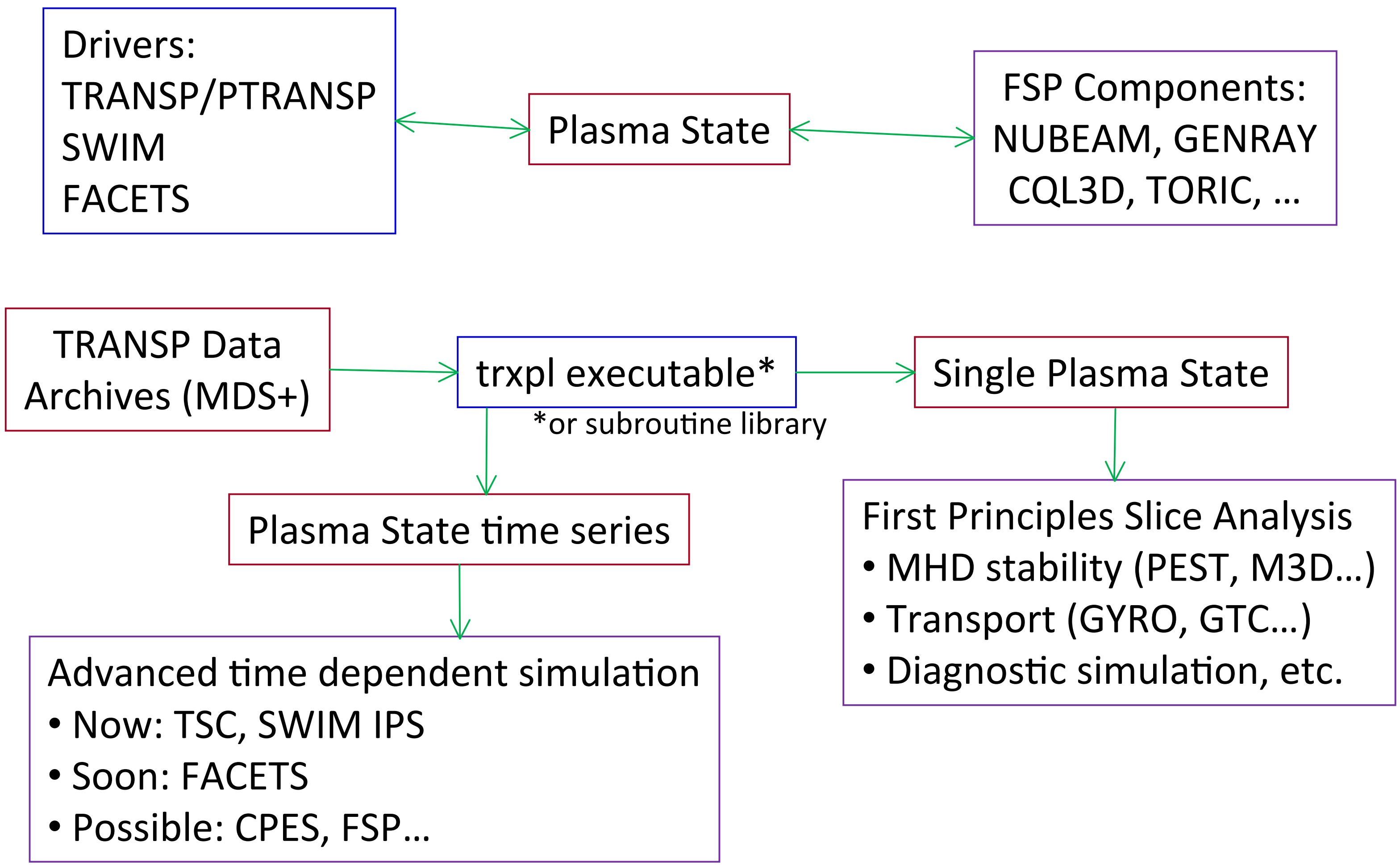
## All Plasma State code written and maintained by Python code generator

- **Edit the specification file.**
- **Run the Python code generator.**
- **Run compatibility tests.**
- **Commit to repository**
- **Current released SWIM version: 2.032**

## Version compatibility

- **All version 2.xxx states compatible**
- **Code linked to newer PS software can read old version state file → some data items missing**
- **Code linked to older PS software can read new version state file → some data items not used**

# Utilization of Plasma State



# Performance Considerations

- Plasma State I/O is serial overhead.
  - But Plasma State aggregate sizes are usually small;
  - ~500 scalar lists and low rank profile elements;
  - 0.5-5Mbytes as NetCDF, modestly larger memory footprint due to interpolation data;
  - Not a limiting factor in present day applications.
- But this could change quickly if PS is ever extended to include rank 3 or higher profiles.
  - Domain decompositions not yet considered.

# Functional Parallelism

- SWIM has demonstrated functional parallelism by using Plasma State update I/O:
  - NUBEAM (NBI & FUS) component on 500p;
  - AORSA (IC) component concurrently on 2000p;
  - Each writes PS update files when done;
  - Driver waits for individual component completions and reads updates as available.
  - Result is single PS object combining all updates.
- Simple, but it works.

# Areas for improvement

- Have not yet dealt with data protection
- Intrinsic data types only.
- Presently supports arrays up to 2D
- Use experience limited to small aggregate data sizes – large data structures transferred as references to separate files (so far)
- Code dependencies (NTCC modules) make build on new platforms complicated
- Simulation use strategies have to be carefully planned – Well defined, step by step initialization process
- Dynamic regridding requires re-initialization of state object with interpolation → Code generator can make this convenient when needed (hasn't happened yet).

# Strengths

- Simple
  - Simple interface supported by extensive optional services
  - Flat structure – below Plasma State derived type or C++ object only intrinsic data types
- Small
  - Typically 0.5 to 5 MB
  - Can read/write partial Plasma States
- Persistent storage in standard file format (NETCDF)
  - Accessible from any standard programming language – fortran, C/C++, Python
  - Readable by common utilities – ncdump, VisIt (graphics)…
- Fairly broad experience of use inside and outside of project
  - Time dependent simulation
  - Access to experimental data within simulations – interpretive simulation
  - Exchange of experimental data

## Summary

- The Plasma State provides a simple, yet powerful data standard for time dependent multiphysics simulation.
- Application so far in realm of 1.5d transport codes.
- Provides for standardization of communications with physics components.
- Provides for access to archived TRANSP results.