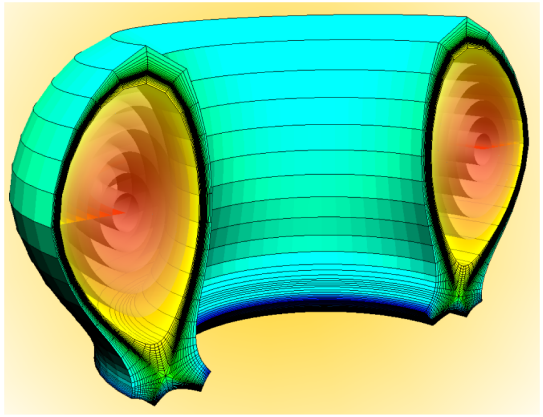
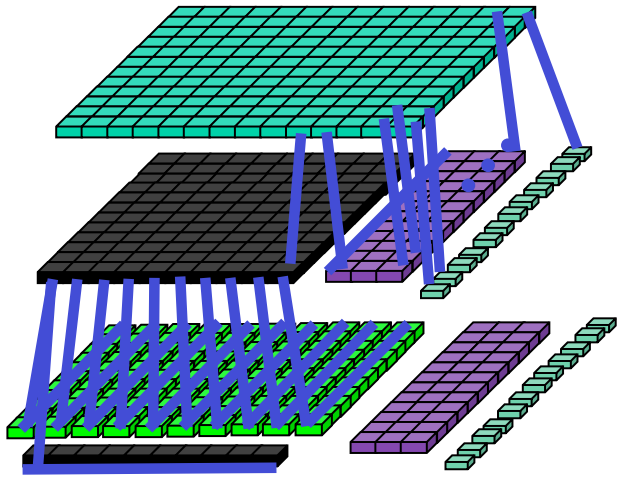
FACETS - A Tightly-coupled Framework for Integrated Fusion Modeling

3 December 2010





ANL (solvers): McInnes, Zhang, Balay, Farley, McCourt

CSU (sensitivity research): Estep, Tavener, Sheehan

GA (exp, GYRO): Groebner, Candy

Lehigh (core modeling, SBIR subcontract): Pankin

LLNL (edge physics): Cohen, Rognlien. Lodestro

LLNL (interlanguage): Epperly

ORNL (modeling, user interaction): Cobb

ParaTools (performance analysis): Malony, Spear, Shende

PPPL (core sources, algorithms): McCune, Indireshkumar, Hammett

UCSD (wall): Pigarov

Tech-X (framework, core): Cary, Carlsson, Hakim, Kruger, Miah, Pletzer, Shasharina, Vadlamani, Durant, Alexander Green

https://www.facetsproject.org/

FACETS especially thanks its collaborators

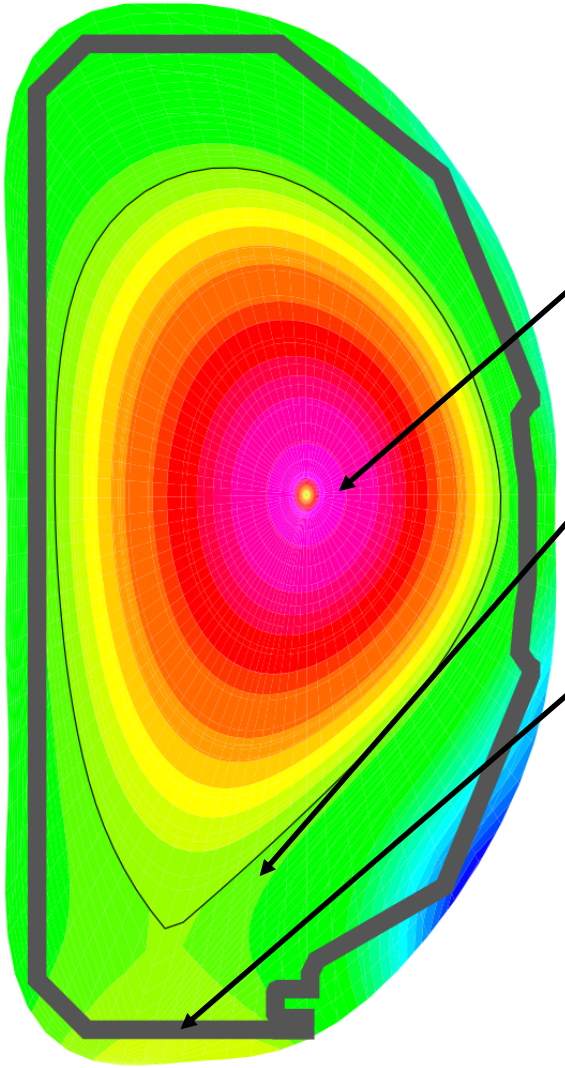
- SWIM collaboration has led to component improvements that have been exchanged
- CPES: Beginning collaboration with ADIOS
- PETSc/TOPS collaboration has led to algorithmic improvements
- VACET collaboration critical to developing visualization

- Important input from other unfunded collaborators
  - Rich Groebner
  - Alexei Pankin

Outline

- Overview of FACETS
- Framework in Detail

FACETS goal: tight coupling framework for core-edge-wall



Hot central plasma: nearly completely ionized, magnetic lines lie on flux surfaces, 3D turbulence embedded in **1D** transport

Cooler edge plasma: atomic physics important, magnetic lines terminate on material surfaces, 3D turbulence embedded in **2D** transport

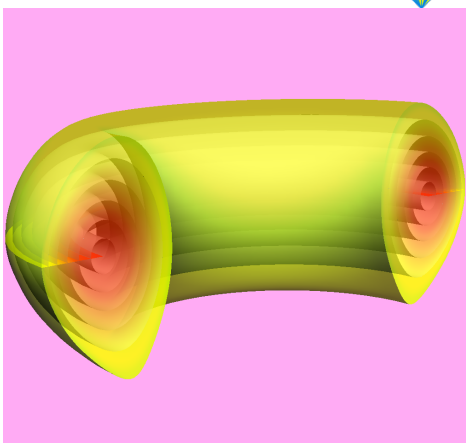
Material walls, embedded hydrogenic species, recycling

- **Coupling on short time scales**
- **Implicit coupling**
- **Inter-processor with MPI and in-memory communication**

FACETS Approach: couple physics components

- A fusion plasma finds a self-consistent, core-edge-wall state
  - Energy into the edge determines the pedestal (pressure) height, while the pedestal height is a dominant determiner of interior temperature, and so fusion power
  - Particle recycling involves wall loading/discharging, with the particles from the wall determining plasma density which then determines flux into the wall
- Coupling components (as opposed to one monolithic code) exploits space and time scale disparities and makes use of proven techniques for incorporating important physics in each region
- Plus -- it is not possible to cover all scales for all times (ITER is 20k ρ_e across)

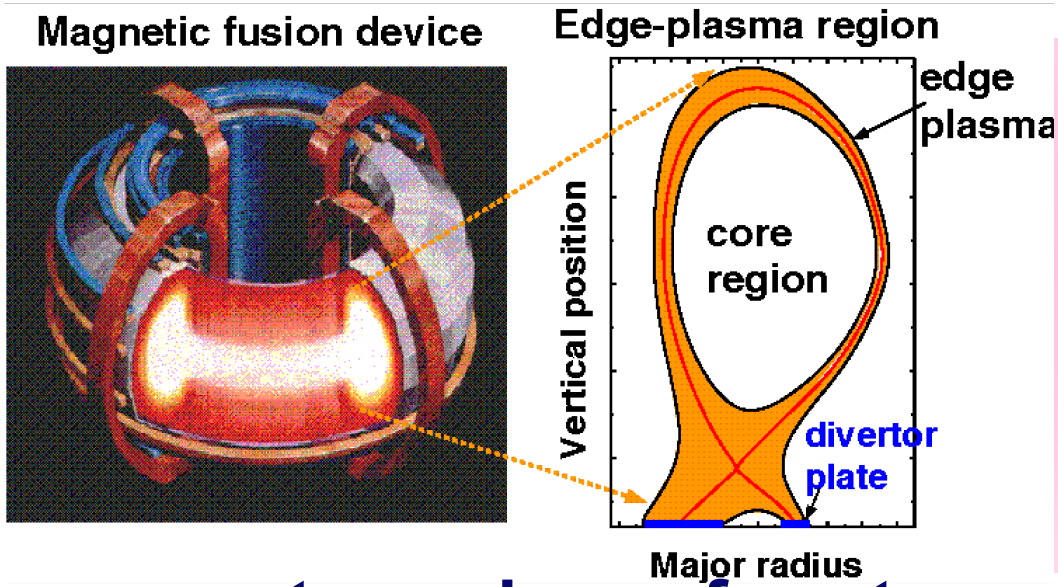
Plasma core: hot, 3D within 1D

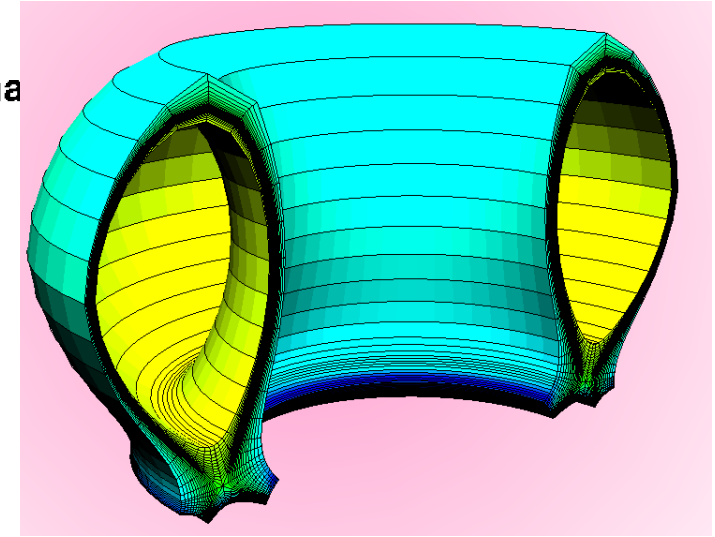


- Plasma core is the region well inside the separatrix
- Transport along field lines >> perpendicular transport leading to homogenization in poloidal direction
- 1D core equations in conservative form:
  - q = {plasma density, electron energy density, ion energy density}
  - F = highly nonlinear fluxes incl. neoclassical diffusion, electron/ion temperature gradient induced turbulence, etc., discussed later
  - S = particle and heating sources and sinks

$$\frac{\partial q}{\partial t} + \nabla \cdot F = S$$

Plasma edge: balance between transport within and across flux surfaces

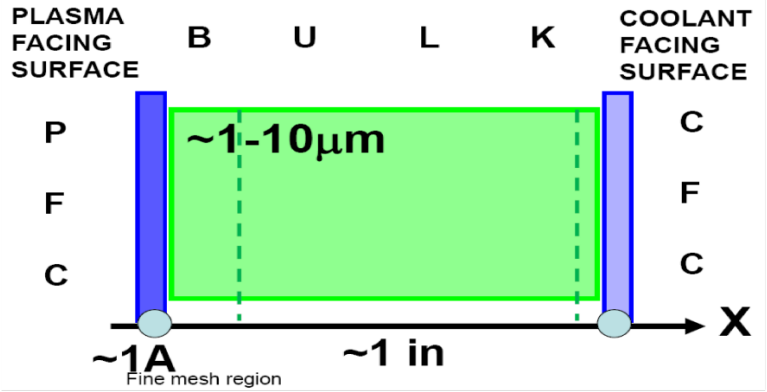




- Narrow: strong in-surface transport matched by slower cross-field transport
- Contacts divertor plates, determines wall loads
- Neutrals transport from edge to fuel core
- First component is UEDGE, multispecies, fluid plasma transport code

The wall is a dynamical medium, charging, discharging, eroding…

- **Challenges**
  - Dynamic and static retention of hydrogen in walls during and between the plasma shots
  - Hydrogen recycling, wall pumping/ outgassing and their effect on plasma performance
  - Heat exhaust by wall
  - Erosion/deposition and wall components lifetime
- **First component: WallPSI**
  - Distinguish mobile, adsorbed and trapped hydrogen on surfaces and in the bulk
  - Wall segment covered with non-uniform mesh with 1 A resolution near surfaces

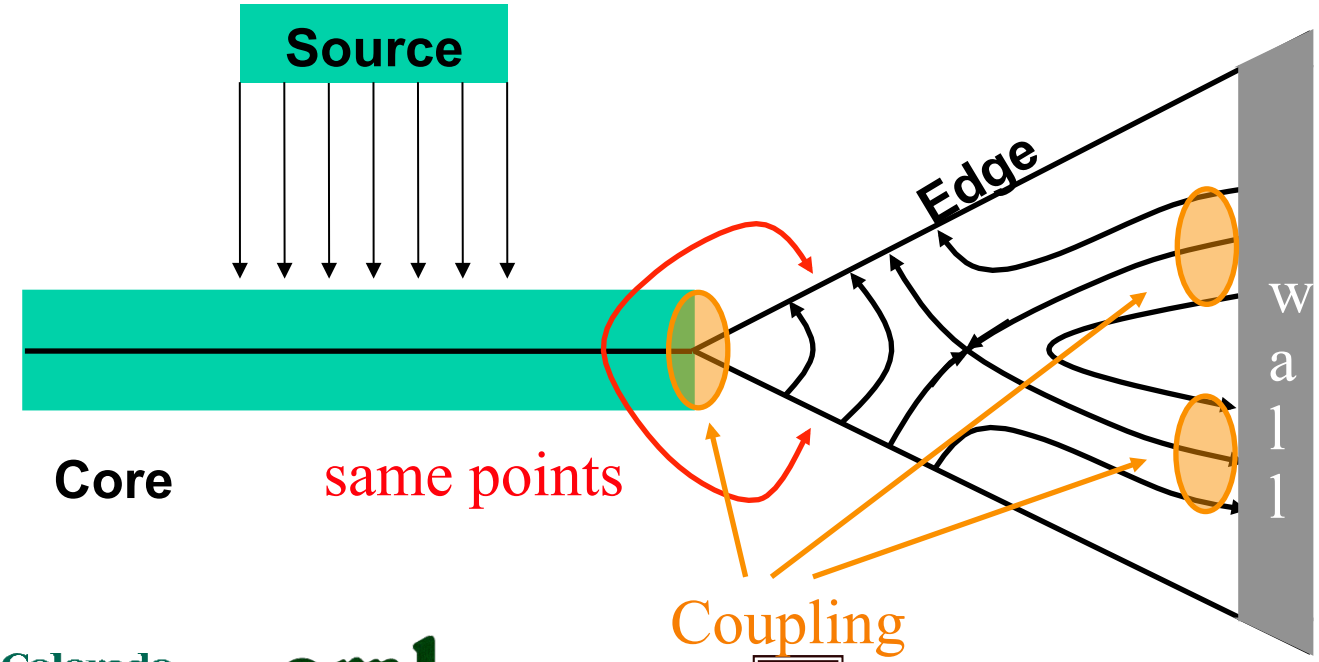


Large projects require distribution and connectivity

- Distribution/connectivity of physics (spatially)
- Distribution/connectivity of computing
- Distribution/connectivity of people

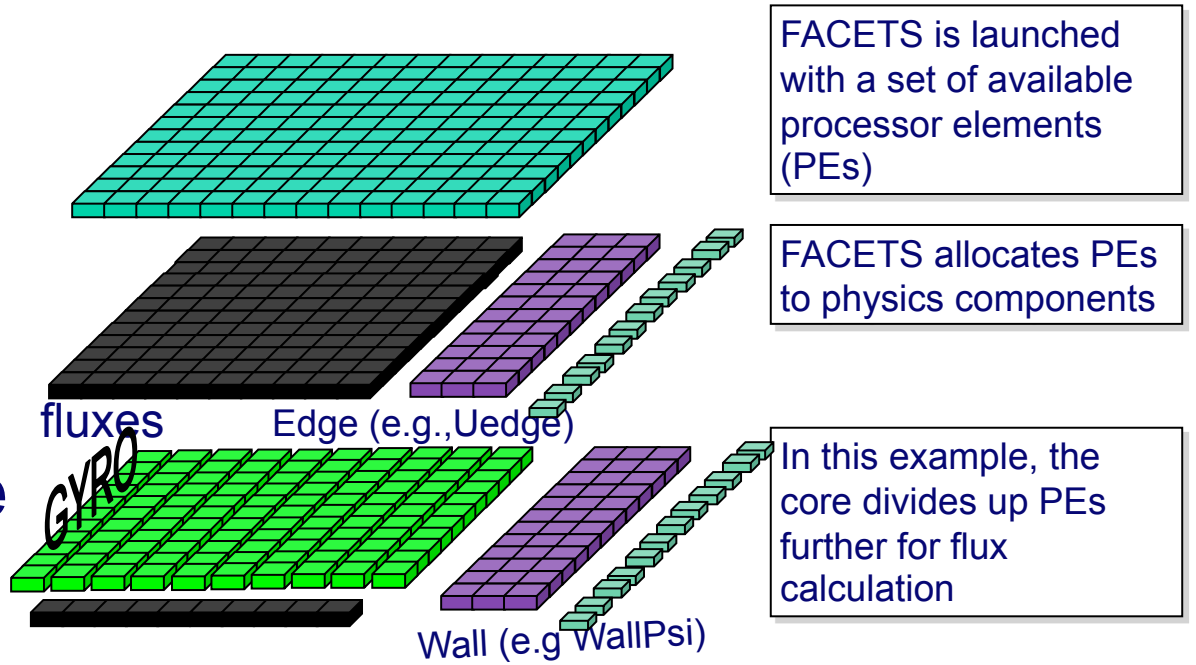
Core-Edge-Wall physics requires four different types of coupling

- **Core (1D)/Edge (2D) coupling involves point coupling**
- **Edge (2D)/Wall (1D) coupling involves point coupling between edge and multiple wall instances**
- **Source (2D)/Core(1D) involves volumetric coupling from source to core**
- **Equilibrium(1D)/Core(1D)/Edge(1D) involves volumetric coupling**



2. FACETS created a recursive communicator splitting framework to allow for such distributions/connectivities

- **Core contains fluxes, sources**
- **Sources can live across core and edge**



Software general enough to work in many situations

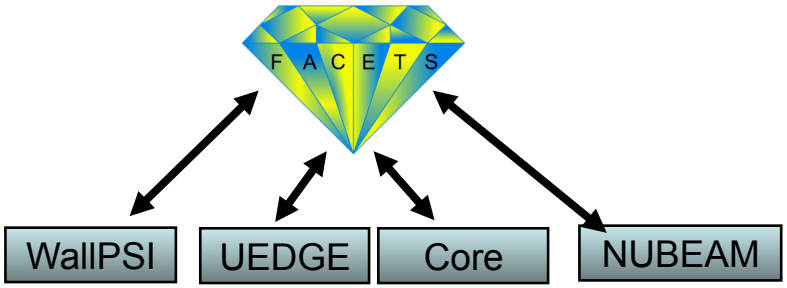
Processor decomposition can occur in two stages

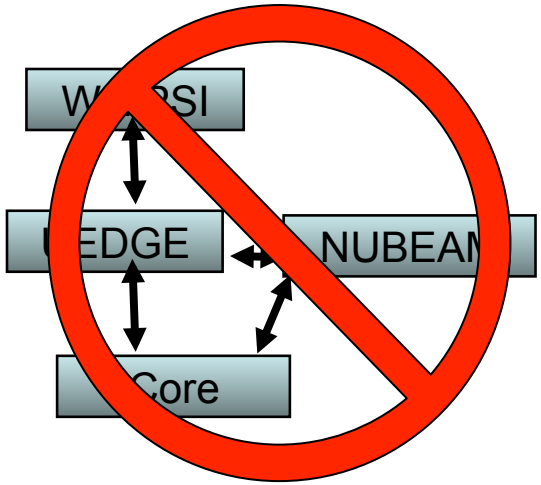
- Stage 1: Each component is give a set of PEs based on estimated work load
  - For each component a new MPI communicator is created using MPI_Comm_Split()
  - A simple round-robin scheme is used to divide the PEs

- Stage 2: A component may need to decompose its PEs further into tasks: e.g for computing turbulent fluxes for use in core or edge transport equations
  - Each task is given its own MPI communicator
  - The rank-0s of each task communicator are collected in another "messaging" communicator
  - We have developed a Mixed Integer Linear-Programming algorithm to do the PE decomposition efficiently

With this new infrastructure FACETS is now moving to full scale simulations on LCFs. Initial embedded core flux calculations show weak-scaling to 64K processors on Intrepid. Core solver enhancements in progress.

On-HPC "framework" mediates all communication





- Components do not talk among themselves
- Allows flexible composition of components
- Allows generic coupling schemes to be written (explicit, implicit) without changing component code
- Frees component developers from worrying about every possible coupling scenario and scheme

FACETS interface standards:

FACETS addressing a number of challenges

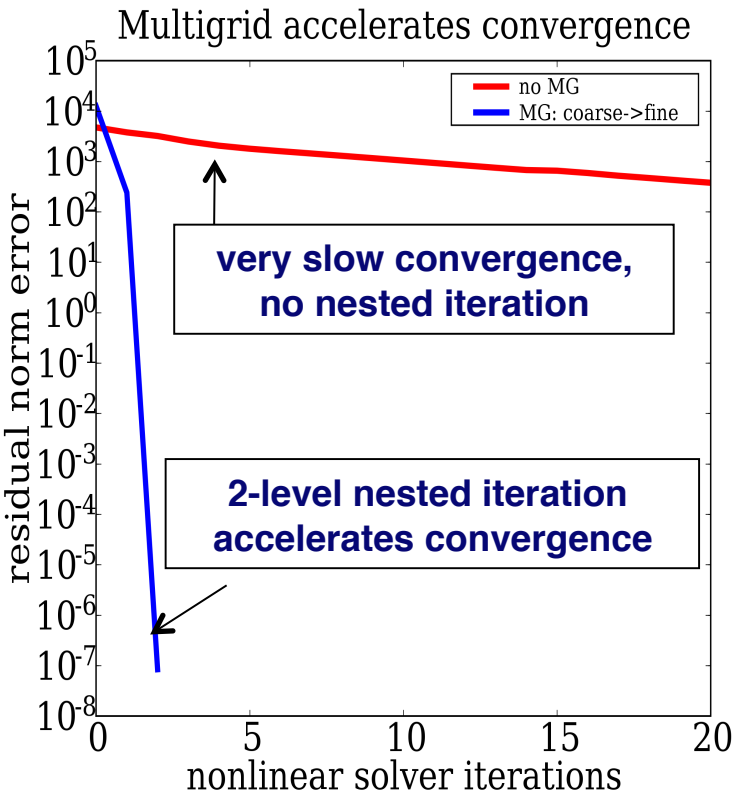
- Development complexity: FACETS is developing and/or composing 2.9M lines of code excluding external libraries
- Social complexity: FACETS researchers and developers come from a multitude of institutions with varying cultures about the best way to work
- Fusion community demographics: funding loss in the 80's and 90's led to a "mature" group, very much set in their ways. FACETS introduced subversion, wikis, build systems, metadata, …

FACETS has made a number of accomplishments

- Highly portable, robust multi-component "on-HPC" framework
- Methodology for ensuring robustness
- Verification (cross comparison with ASTRA)
- Implicit, nested iteration core solver, quasi-Newton core-edge coupling (generic, more than core-edge)
- Componentization of UEDGE, NUBEAM and WallPSI, GLF23, GYRO, TGLF, NEO, NCLASS
- Robust workflow (multi-platform, multi-parallel architecture)
- Uniform visualization using standard tools (VisIt, matplotlib) relying on metadata standards: VizSchema
- First physics: test of edge predictability

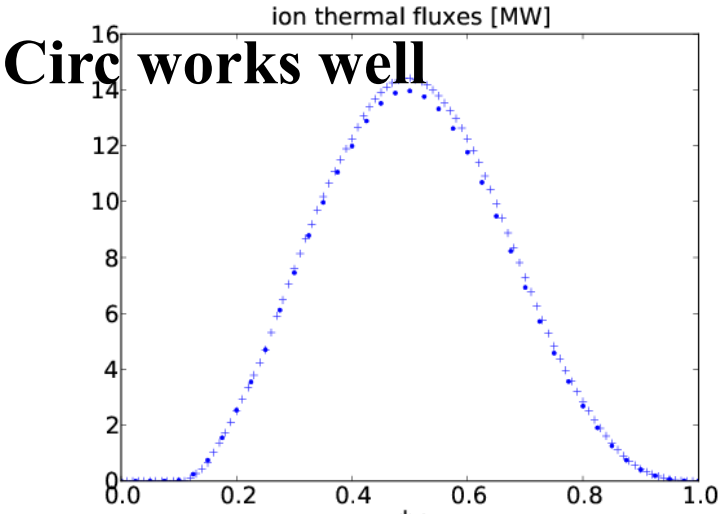
New core solver used allows larger stable time steps

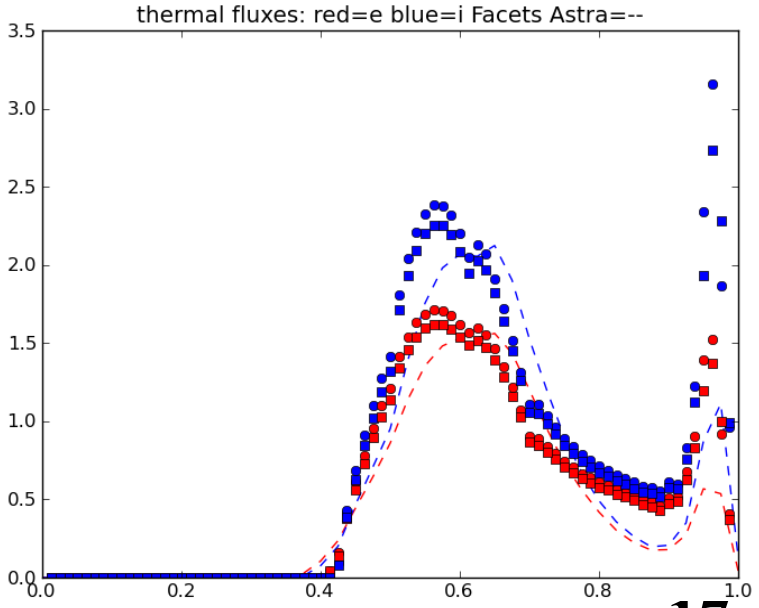
- Core solvers, even though 1D, are notoriously difficult
  - "Stiff (highly nonlinear) transport fluxes as a function of local values and gradients
- Nested iteration allows large time steps



FACETS core component verified by comparison with ASTRA

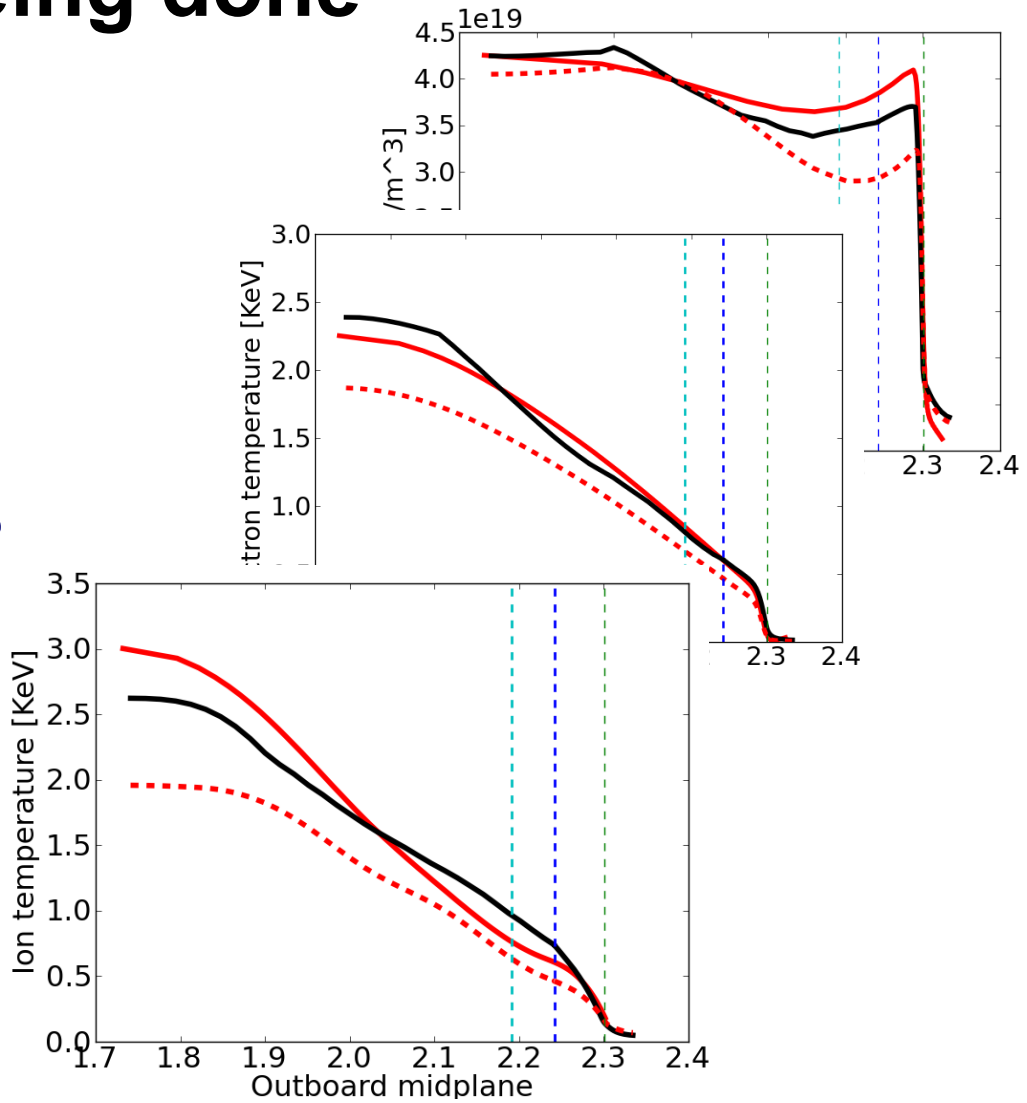
- Benchmark was GLF23 fluxes only for 10 MS
- Ion temperature differs at r=0.7 by 10% between ASTRA and FACETS
- Difference have been traced to differences in calculation of equilibrium quantities. Ongoing work.





Core-edge modeling with interpretive edge now being done

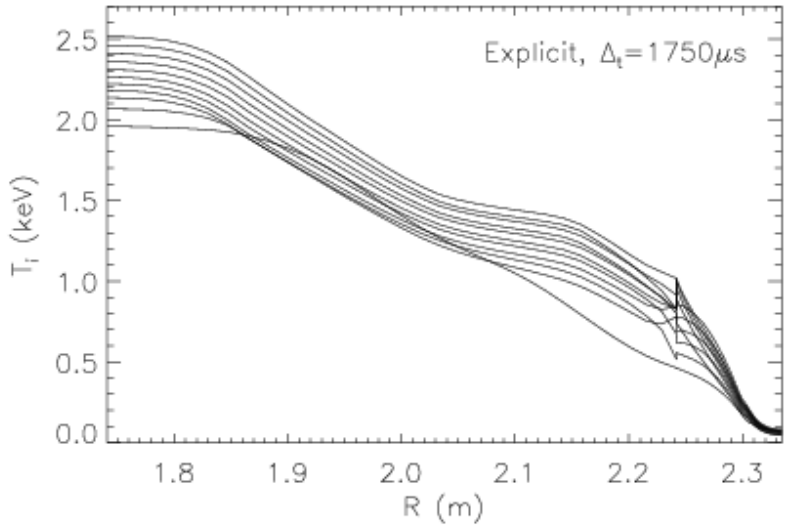
- D3D 118897
- Core profiles reasonably well-described by GLF23 or TGLF
- Edge region lacks predictive models
- Procedure: Use experimental data to determine coefficients required to give profiles over 35 ms with just edge region
- Use profiles in fully coupled simulation
- For core region: Turn of GLF23 over a spatial region to ensure continuity of fluxes
- Sources and impurities come from interpretive transport simulation

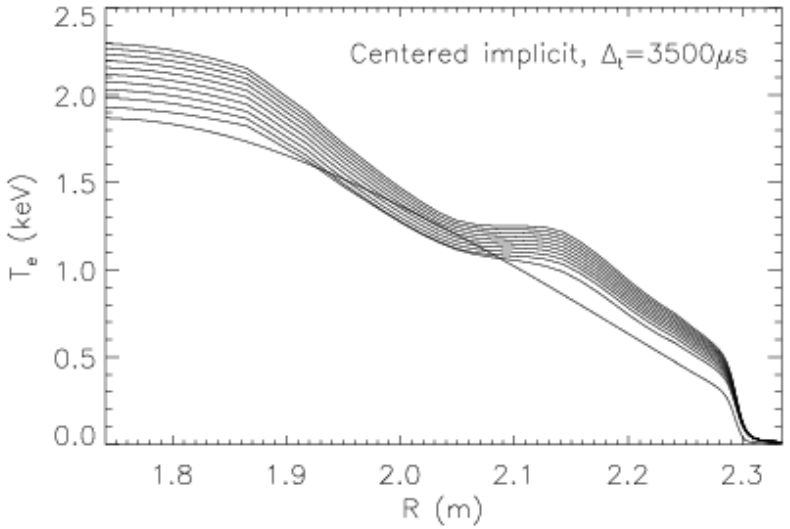


Latest results from coupled core-edge simulations of pedestal buildup in the DIII-D tokamak using the FACETS code, Hakim, GP9.00132

Implicit coupling allows faster computations

- Prior to this year, calculations were explicit
  - Edge passes matching temperature to the core
  - Core passes flux to the edge
  - Lag between values visible at large time step
  - Time step limited to 350 μs before surface oscillations present
- Implicit coupling eliminates the above disadvantages, allows 10x time step, 3x computational savings
- Limitation was due to individual components



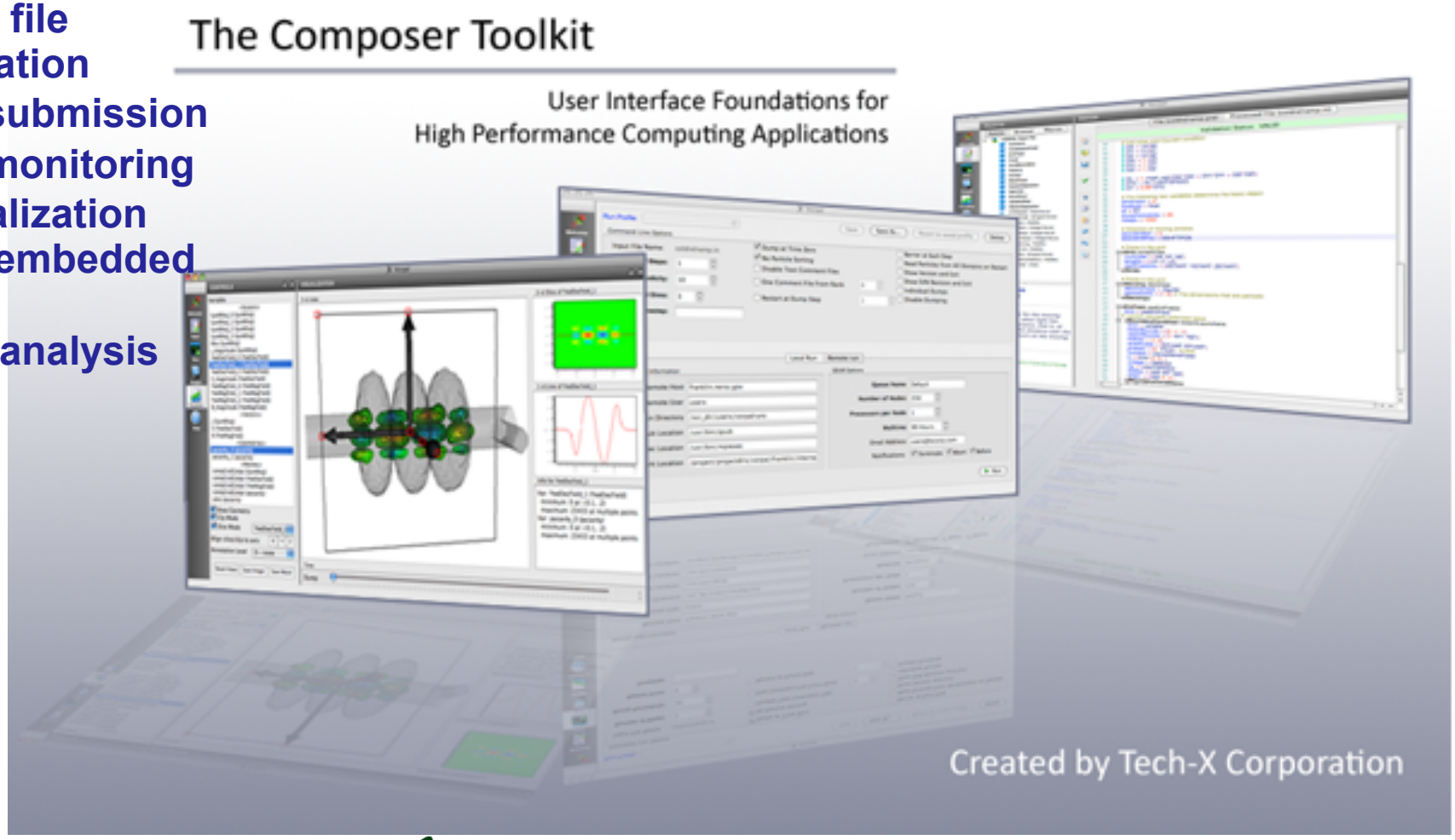


FACETS in now in beta at US D.O.E. Leadership Class Facilities (LCFs)

- NERSC (Berkeley) installations:
  - /project/projectdirs/facets/franklin/internal-path-3.2/facets
  - /project/projectdirs/facets/franklin/internal-pgi-10.5/facets
  - /project/projectdirs/facets/hopper/internal-pgi-10.3/facets
  - /project/projectdirs/facets/carver/internal-gcc-4.4/facets
  - svn version 3294
- ALCF (Argonne) installations
  - /home/projects/facets/intrepid/internal/facets
  - svn version 3031
- Executable(s) in the bin subdirectory
- Running is "simply" submitting a job with the facets executable and an input file (but usually with a script to orchestrate env, profiles, …
- Input file describes the components to be included, hierarchically, and the connections between them

FACETS Composer will facilitate doing simulations

- Native (Windows, Mac, Linux) application
- Input file validation
- Job submission
- Job monitoring
- Visualization with embedded VisIt
- Data analysis



Written generally enough to work variety of applications

Summary and future directions

- A parallel component approach works well for whole-device modeling including the edge
- Core fluxes and solver have been validated
- Verification studies have shown ability to predict pedestal buildup with interpretive coefficients
- Embedded turbulence computations demonstrated (not shown)

Next steps

- Dynamic equilibrium (theory needed for edge)
- Wall
- Predictive edge (BOUT++)

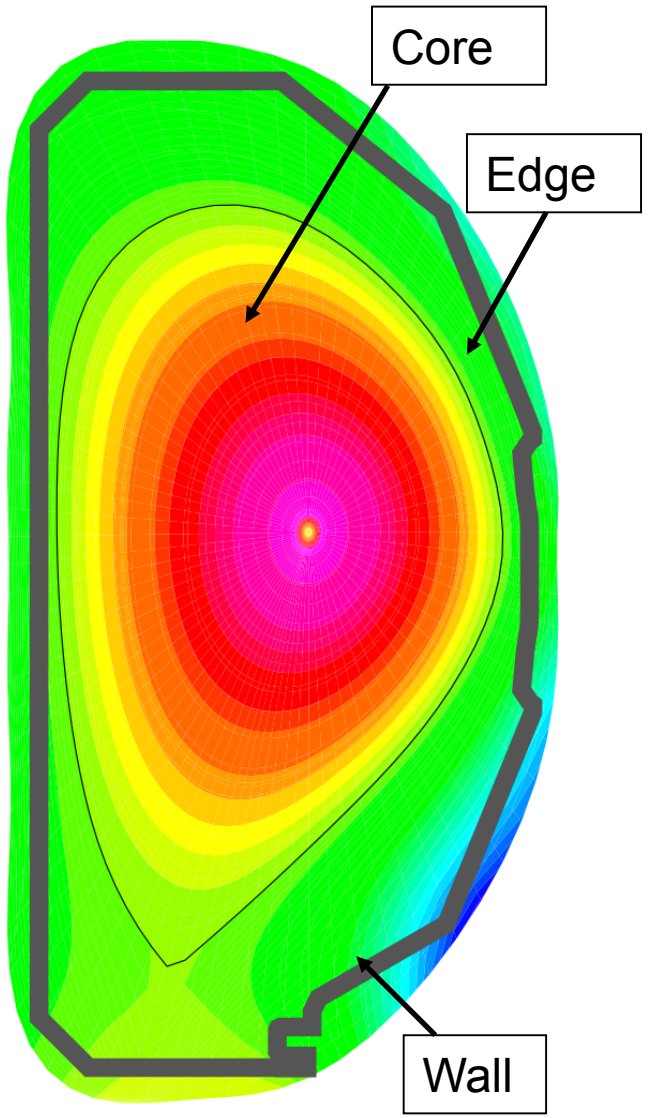
Outline

- Overview of FACETS
- Framework in Detail

Framework will enable distributed multiphysics calculations on HPC machines

Framework provides extensive infrastructure to wrap external physics components and create new components and perform massively parallel, cuncurrent multiphysics simulations.

- **Run on laptops to LCFs**
  - **Provide sequential execution and make efficient use of LCFs: good load balancing**
- **Support low to high fidelity physics**
  - **Swap physics without recompiliation**
- **Support low latency data exchange between components**
  - **Avoid disk files if possible**
  - **Use MPI or other processor-to-processor communication**
- **Support multiple coupling schemes: explicit, implicit, …**
- **Support creation of components written in different languages and different eras (WallPsi is C, UEDGE is Python/Fortran, Nubeam is Fortran 90).**
- **Use existing best practices when possible**
- **Allow for concurrent and sequential execution from a single executable or multiple executables if needed**

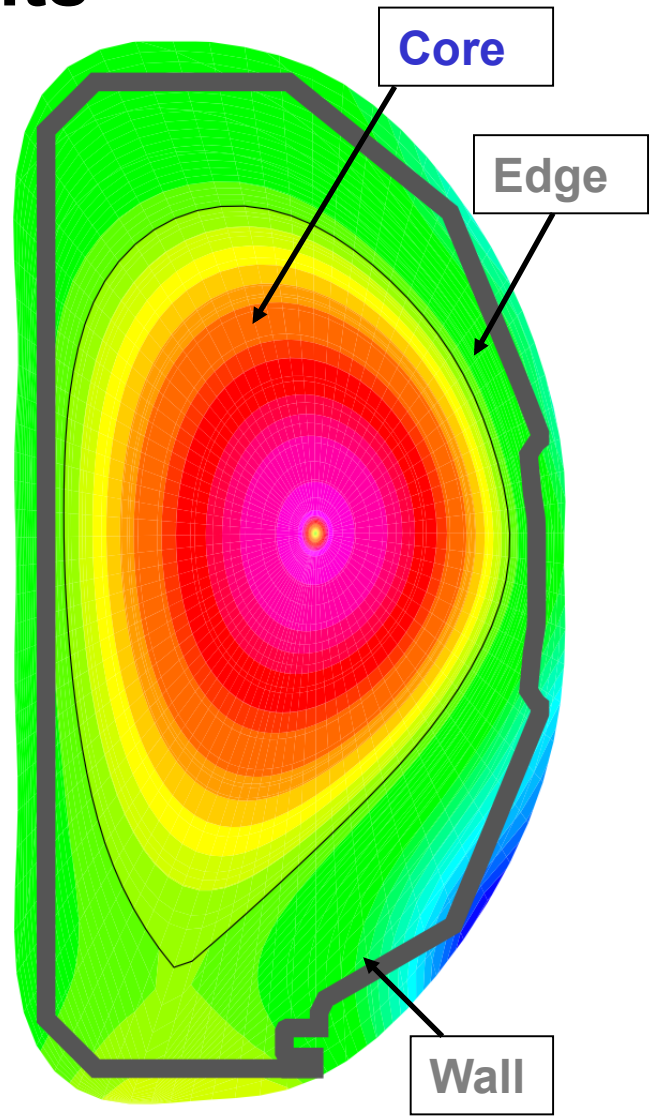


Framework orchestrates all aspects of the simulations

- Creates integrated simulation: single executable, but nothing precludes multiple-executables
  - Allocates parallel resources for components
  - Instantiates components, allowing them to read input files, set initial conditions, etc
  - Provides direct memory access to component
- Advances the simulation: i.e. is responsible for the main loop, and calling out to I/O routines
- Allows for creation of coupling mechanism and provides API for exchanging data
- Provides extensive infrastructure (arrays, grids, algorithms) for creating new solvers or coupling strategies.

FACETS supports both external and internal components

- Core component is an internal component: i.e. written using FACETS provides infrastructure
- Edge component (UEDGE) is an external component written in a version of F90 wrapped using Babel
- Wall component (WallPSI) is an external component written in C and hand-wrapped
- Beam heating component (Nubeam) is external component written in Fortran and hand-wrapped



All coupling algorithms (explicit, implicit) are provided by the framework itself

Components must obey well-defined interfaces for inclusion into FACETS

Initialization/Finalization Interface
`int setLogFile(const string& lf);`
`int setMpiComm(long comm);`
`int readParams(const string& infile);`
`int buildData();`
`int initialize();`
`int finalize();`

Update Interface
`int update(double t);`
`int revert();`

Dump Restore Interface
`int dumpToFile(const string& file) const;`
`int restoreFromFile(const std::string& file);`

Full interface available at https://www.facetsproject.org/wiki/InterfacesAndNamingScheme

Data access enabled by set/get methods. Allows run-time flexibility.

Data Access Interfaces
`int set0dDouble(const string& name, double val);`
`int get0dDouble(const string& name, double& ret) const;`
`int setNdDouble(const string& name, const std::vector<int>& shape,const std::vector<double>& data);`
`int getNdDouble(const string& name, std::vector<int>& shape, std::vector<double>& data) const;`
`int getRankOfInterface(const string& name, size_t& ret)`

Simplified interfaces with standard C++ structures

Full interface available at https://www.facetsproject.org/wiki/InterfacesAndNamingScheme

Data-exchange is through get/set string interface

quantity_Interface_ElementsymMassnumIonstate_class **(name should be legal C/Python identifier and not underscores in except as separators)**

Quantity: e.g. density [m^3], energyDensity [J/m^3], ptclFlux[$1/m^2/s$], temperature [eV]

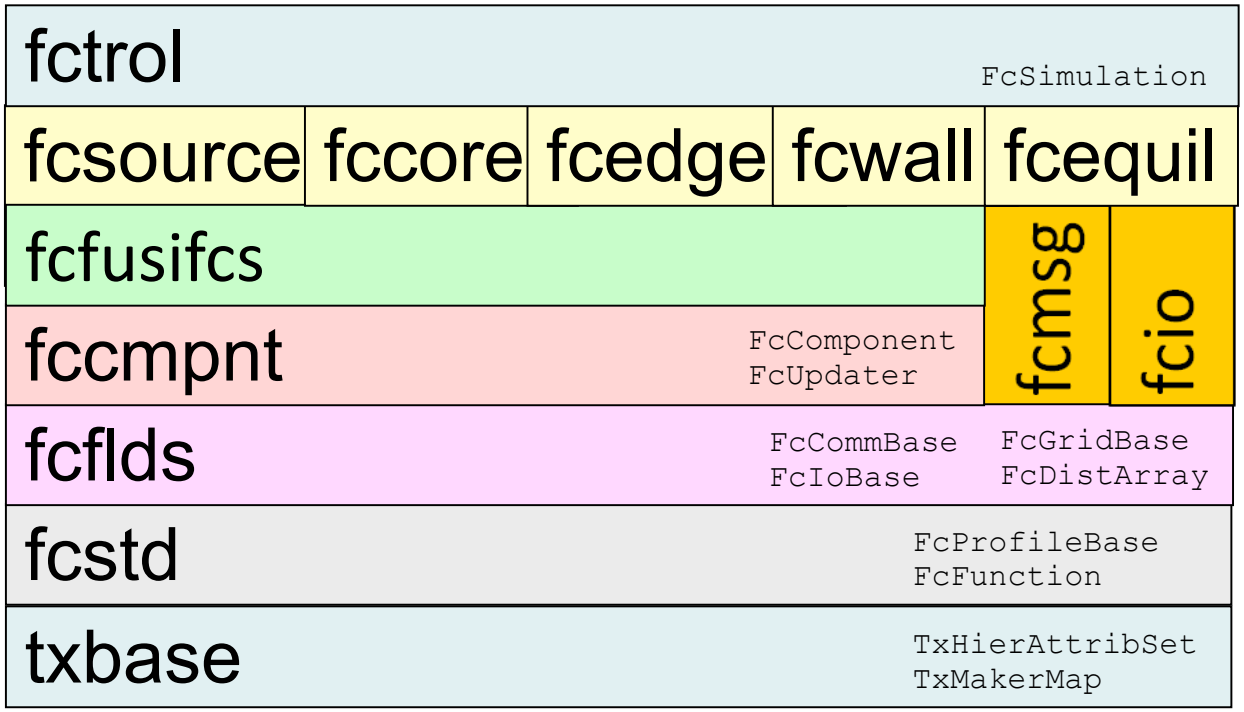
Interface: CE → Core-Edge, EW → Edge-Wall, CS → Core-Source

Species: e.g. O16p8 → fully-ionized oxygen, H2p1 → deuterium, C12p3 → Carbon-12 with 3rd ionization state

Examples:
density_CE_C12p5, density_CS_He4p2_fusionprod, ptclFlux_EW_H2p1

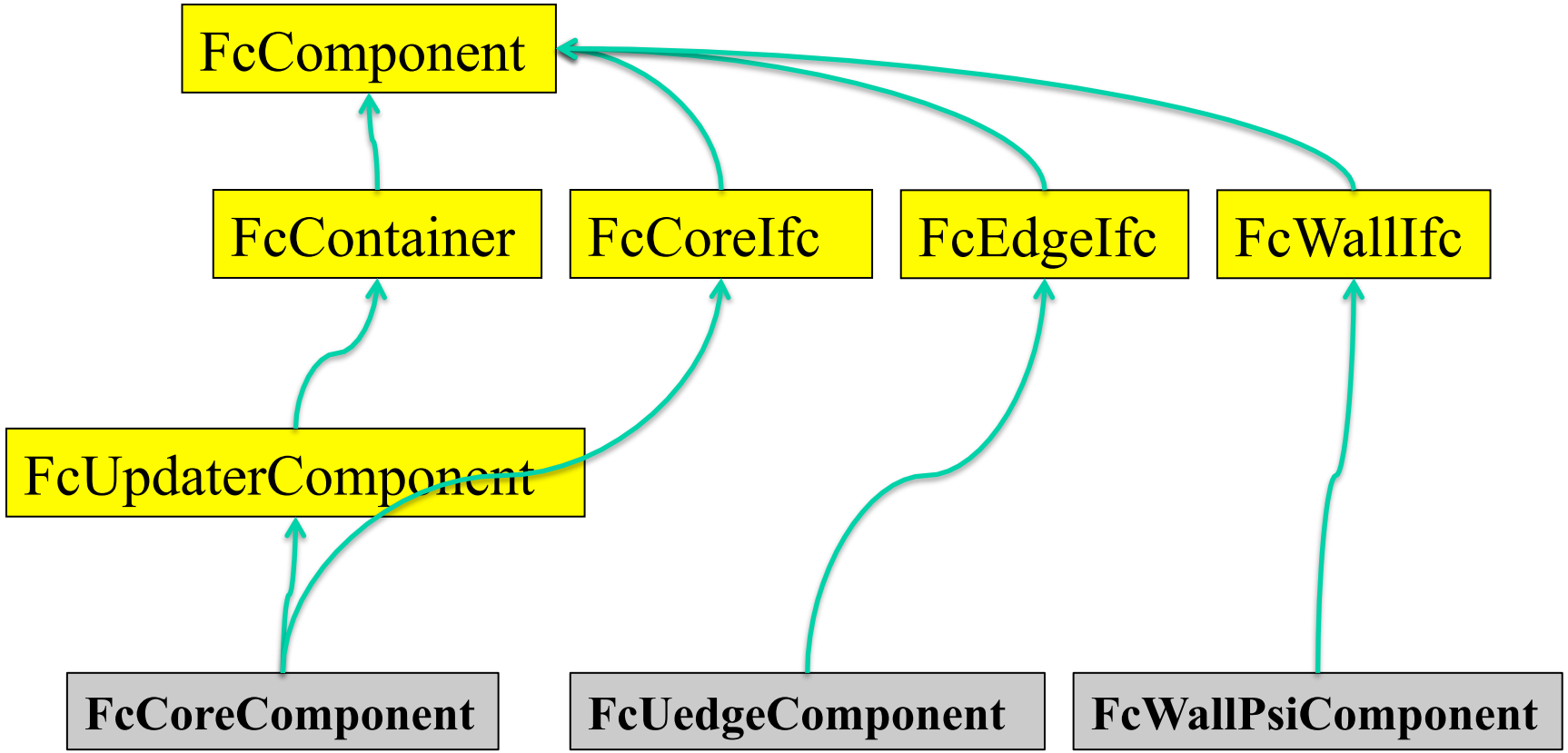
Units are in SI except for temperature in eV. Names and API determined after long discussion with FACETS team.

FACETS framework is written in C++ and uses careful layering and dynamic and static polymorphism

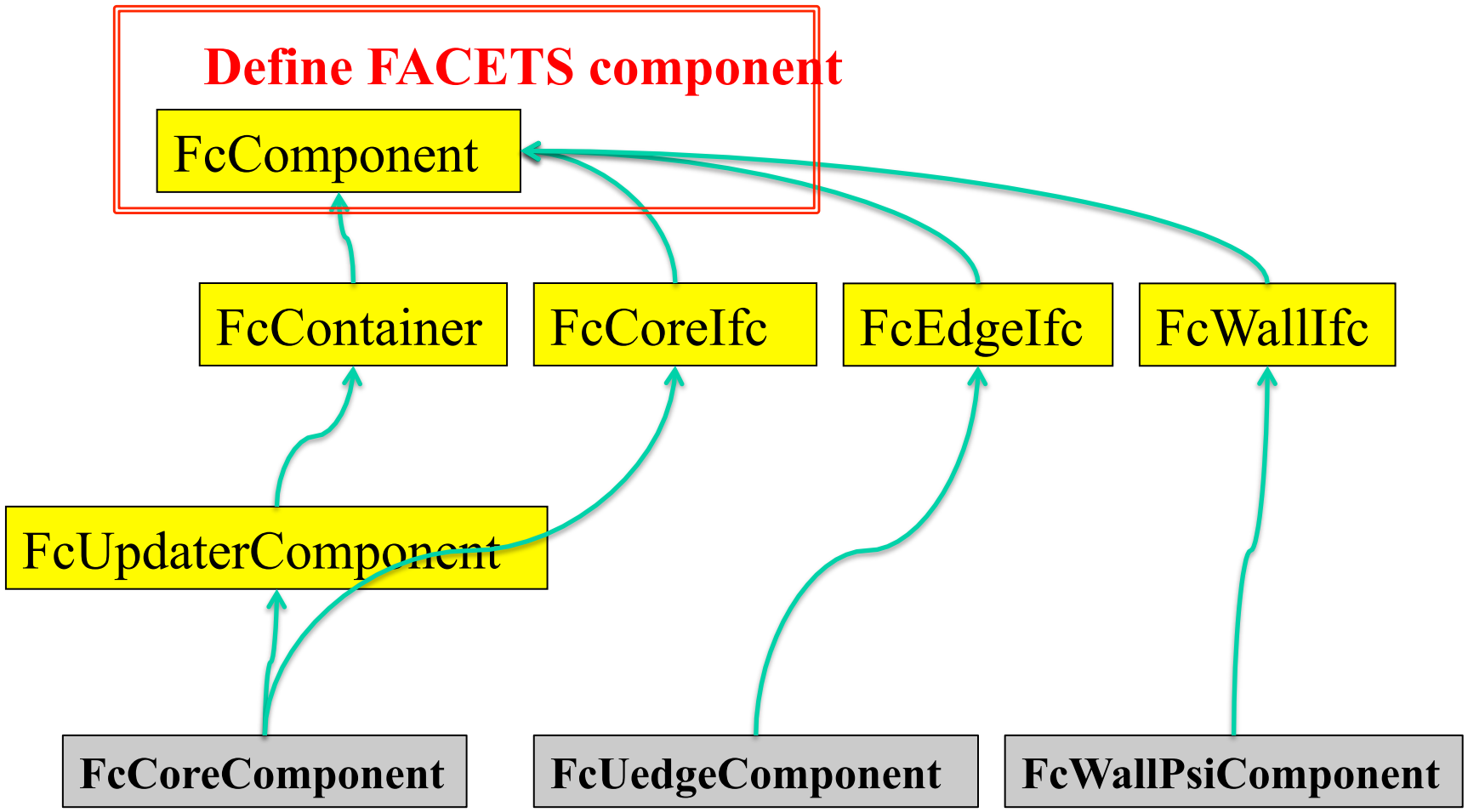


Lines of code	69135
Number source files	486
Total managed code	Over 2.9 million, 3 million generated

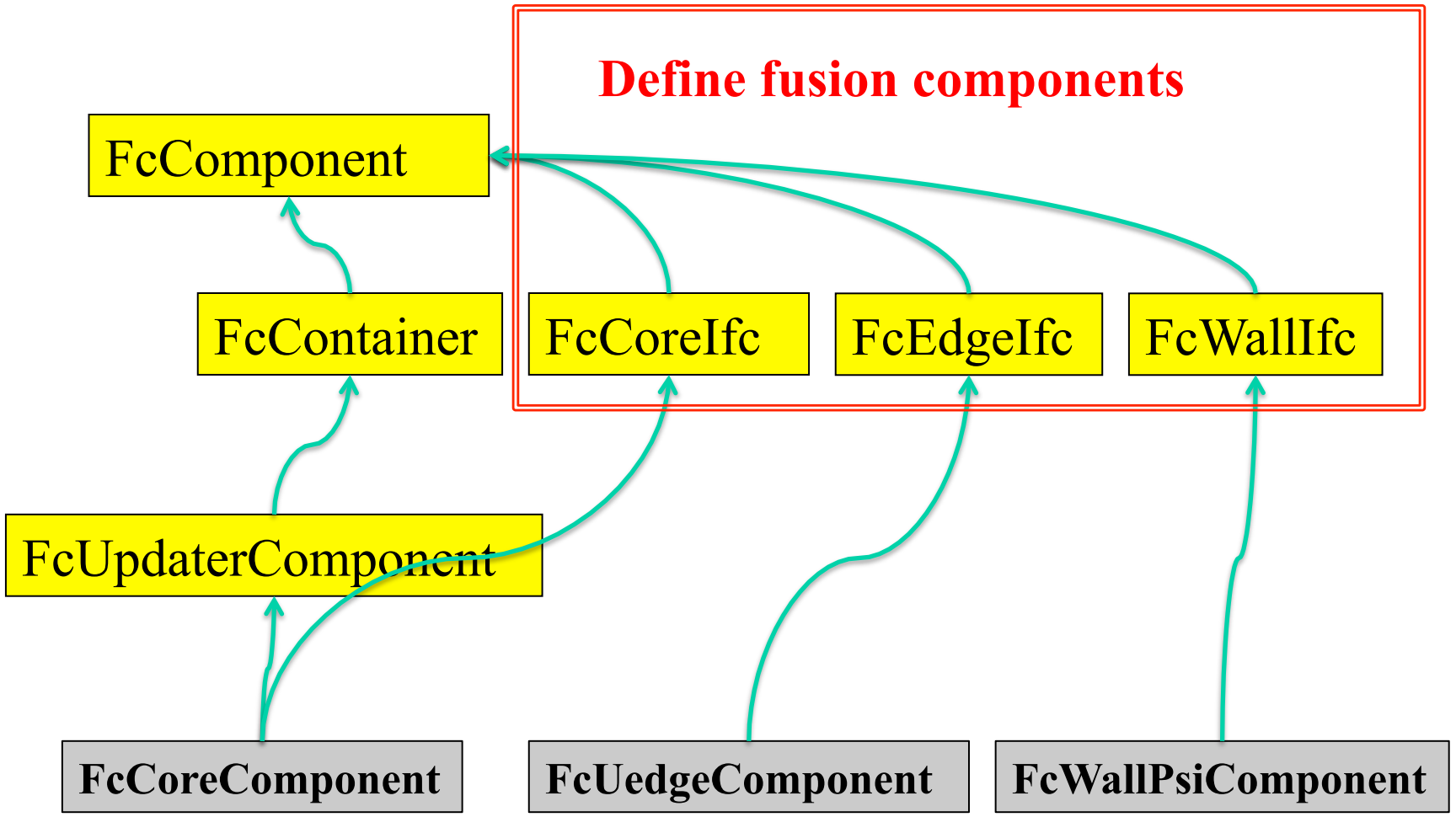
C++ framework hierarchy permits determination of component type



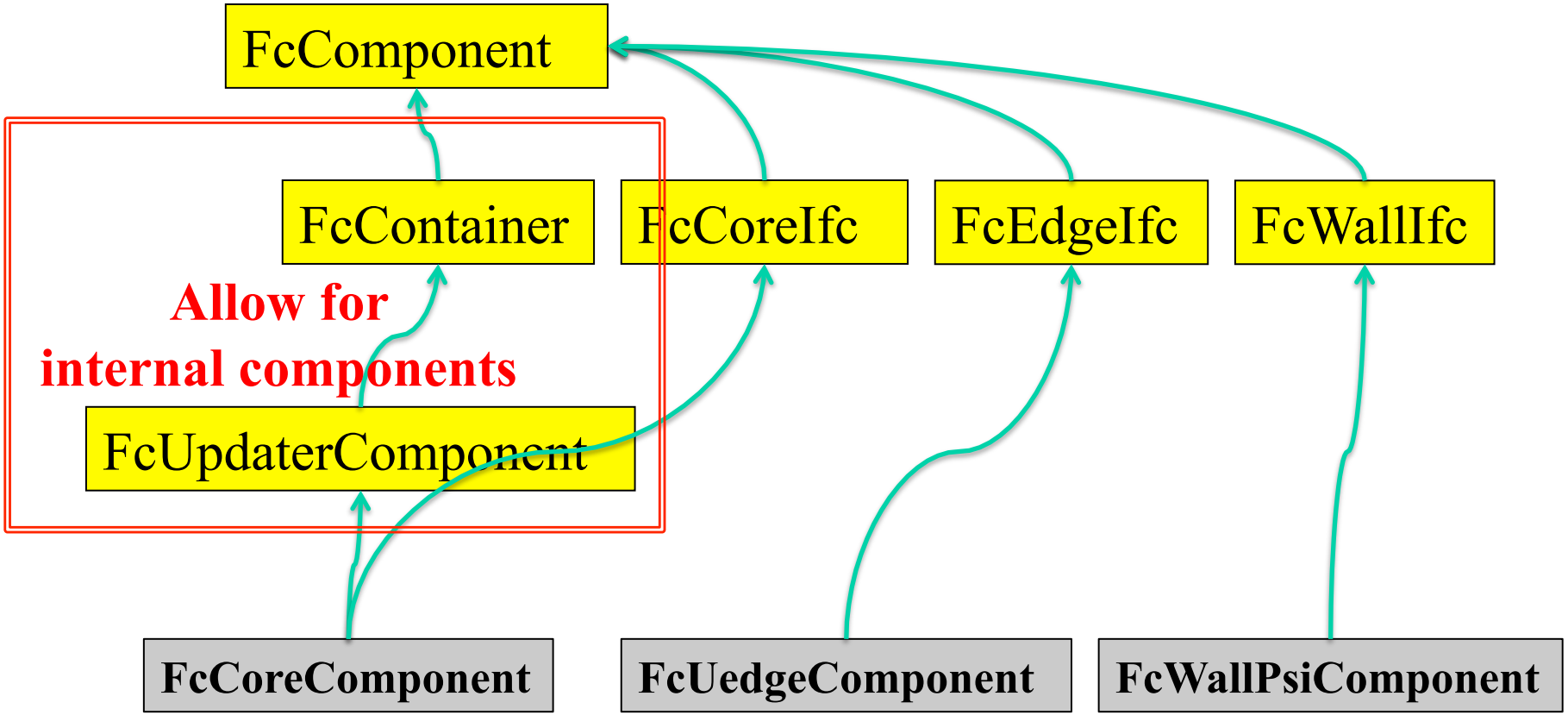
Hierarchy permits determination of component type



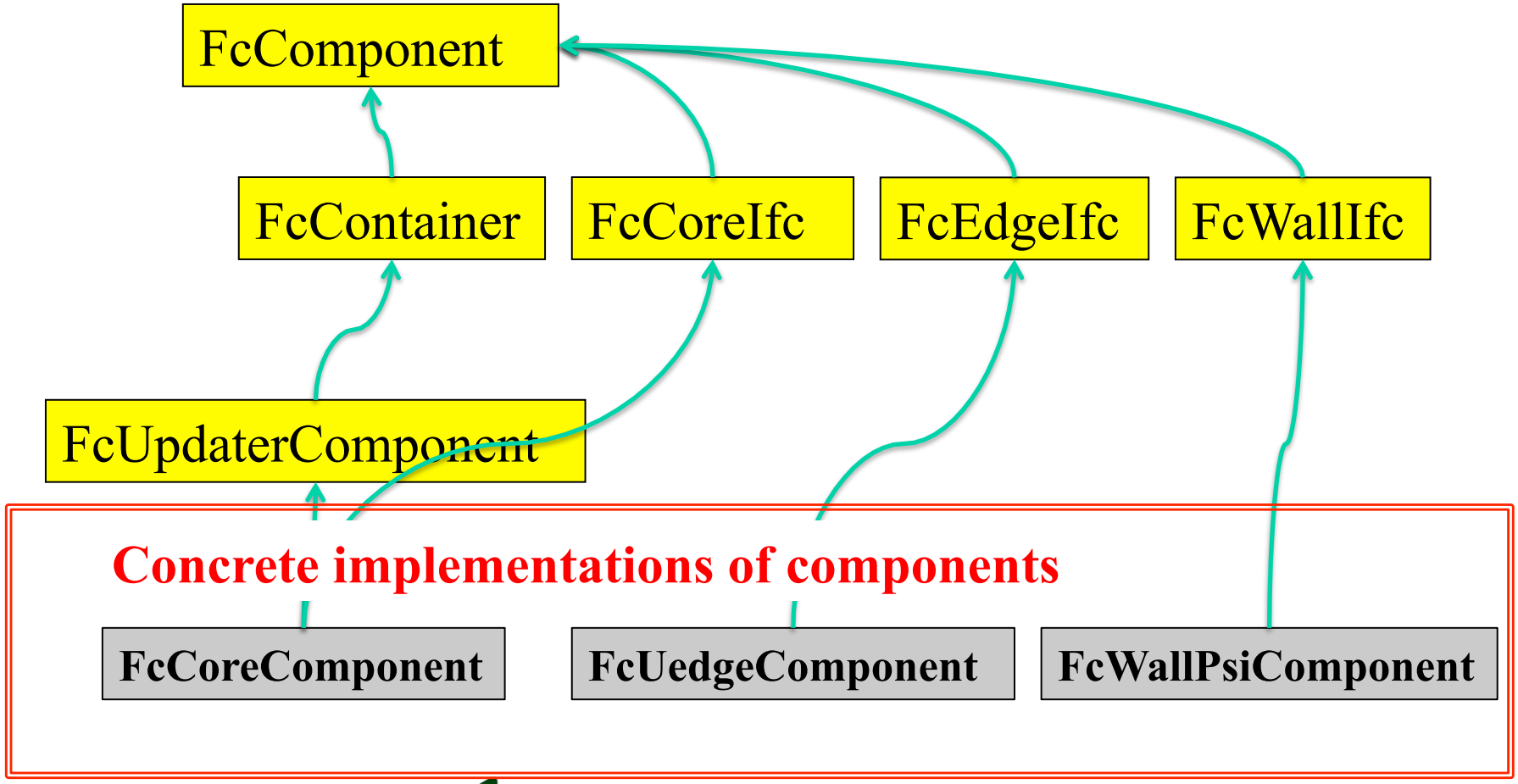
Hierarchy permits determination of component type



Hierarchy permits determination of component type



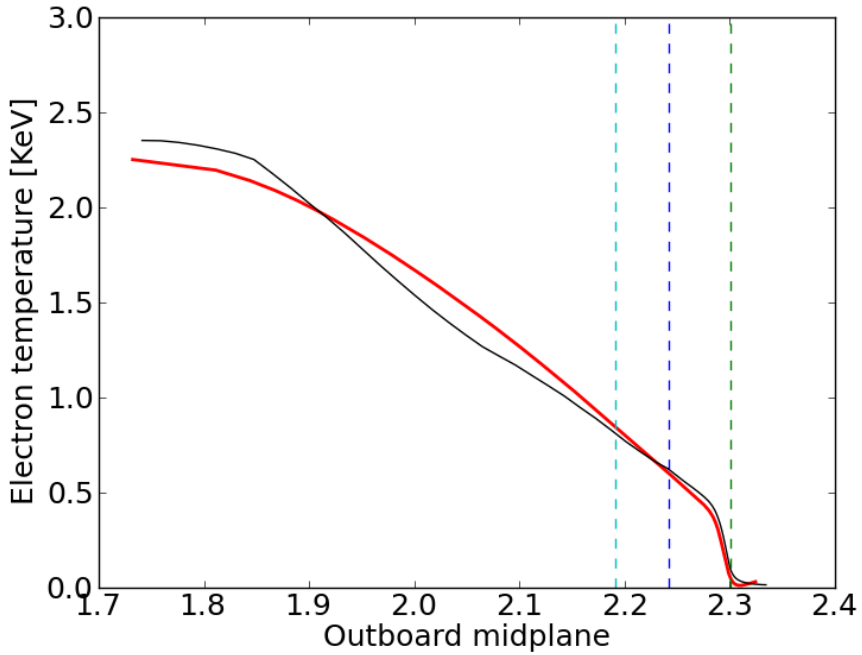
Hierarchy permits determination of component type



Separation of data from algorithms provides run-time flexibility and reuse

- Framework provides extensive infrastructure
  - Updaters: Encapsulate algorithms
  - DataStructs: Encapsulate data (distributed arrays, pre-processor arrays, …)
  - Grids: Encapsulate computational domain (Cartesian, body-fitted, unstructured, …)
- This infrastructure can be used to
  - Write brand new (non-legacy) algorithm: e.g. FACETS core solver, implicit solvers for non-linear systems, fluid solvers, ….
  - Write coupling algorithms: e.g: Explicit coupling, quasi-Newton coupling, …

Example updater: explicit core-edge coupling



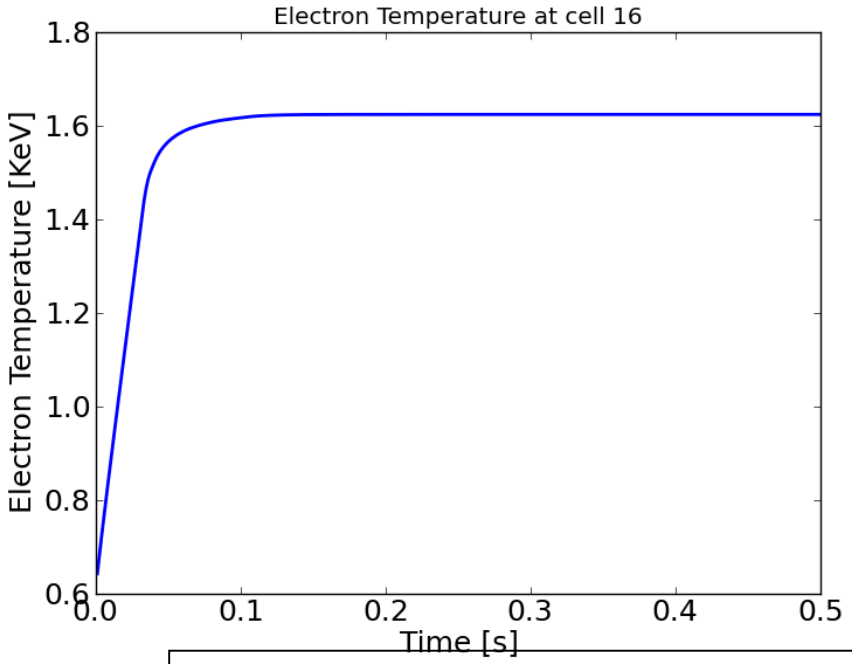

```
<Updater myCoreEdgeUpdater>
   kind = containerUpdater
# names of components to run
   components = [core, edge]
</Updater>
```

```
<Updater coupleCore2Edge>
   kind = bcDataTransferUpdater
# name of source component
   fromComponent = core
# name of destination component
   toComponent = edge
# name of interface
   interface = CE
# list of variables to get
   getNames = ["energyFlux_CE_H2p1”]
# list of variables to set
   setNames = ["energyFlux_CE_H2p1"]
</Updater>
```

```
<UpdateStep coreEdgeStep>
# updaters to run
  updaters = [myCoreEdgeUpdater,
     coupleCore2Edge, coupleEdge2Core]
</UpdateStep>
```

Special DataStruct can be used to record simulation diagnostics

```
<DataStruct cellElcTemp>
    kind = dynVector
    numComponents = 1
</DataStruct>
```



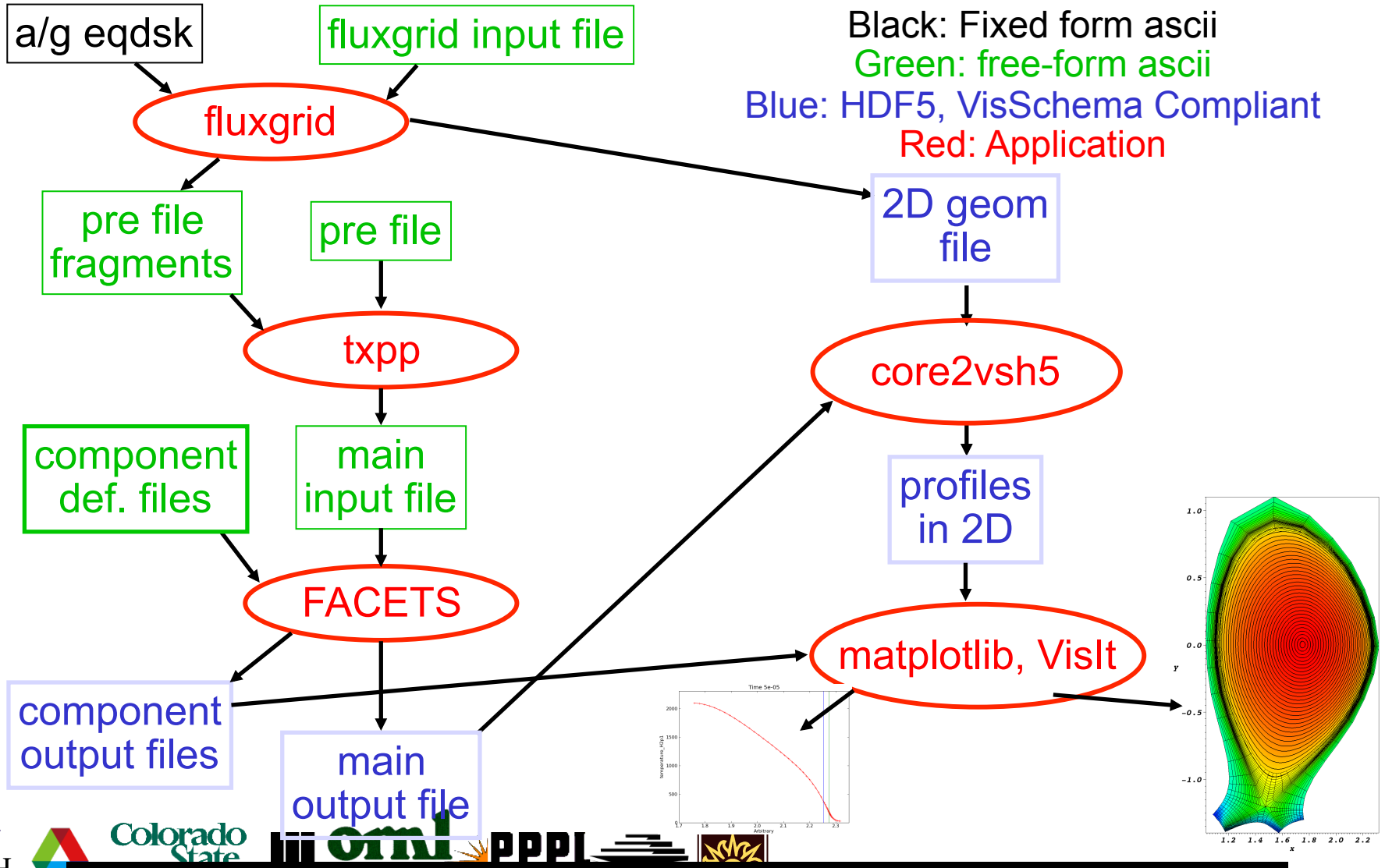

```
<Updater recordElcTemp>
    kind = recordFieldAtIndex1D
    onGrid = domain
 # input array name
    in = [temperature_electron]
# output dynVector name
    out = [cellElcTemp]
# index in input array to record
    index = [16]
# component in input array to record
    inpIndices = [0]
</Updater>
<UpdateStep recordElcTempStep>
# updaters to run
  updaters = [recordElcTemp]
</UpdateStep>
```

Recorded data can be used to "replay" a simulation without actually running the component that produced the data

FACETS uses shell scripts for workflow management

- Workflow: everything that can be done off HPC
  - Pre and post processing of input and experimental data
  - Visualization
- Why shell scripts:
  - With complex workflows (data from experiments, geometry parsing, …) portability and programmability is desired
  - Programming with a GUI language can get tedious: one does not want to write compiled code to execute utilities
- FACETS uses a single input file resulting from preprocessing
  - External components may use their own input file format (e.g. UEDGE) but are told what particular file to use.
- All preprocessing is done using Python or compiled utilities

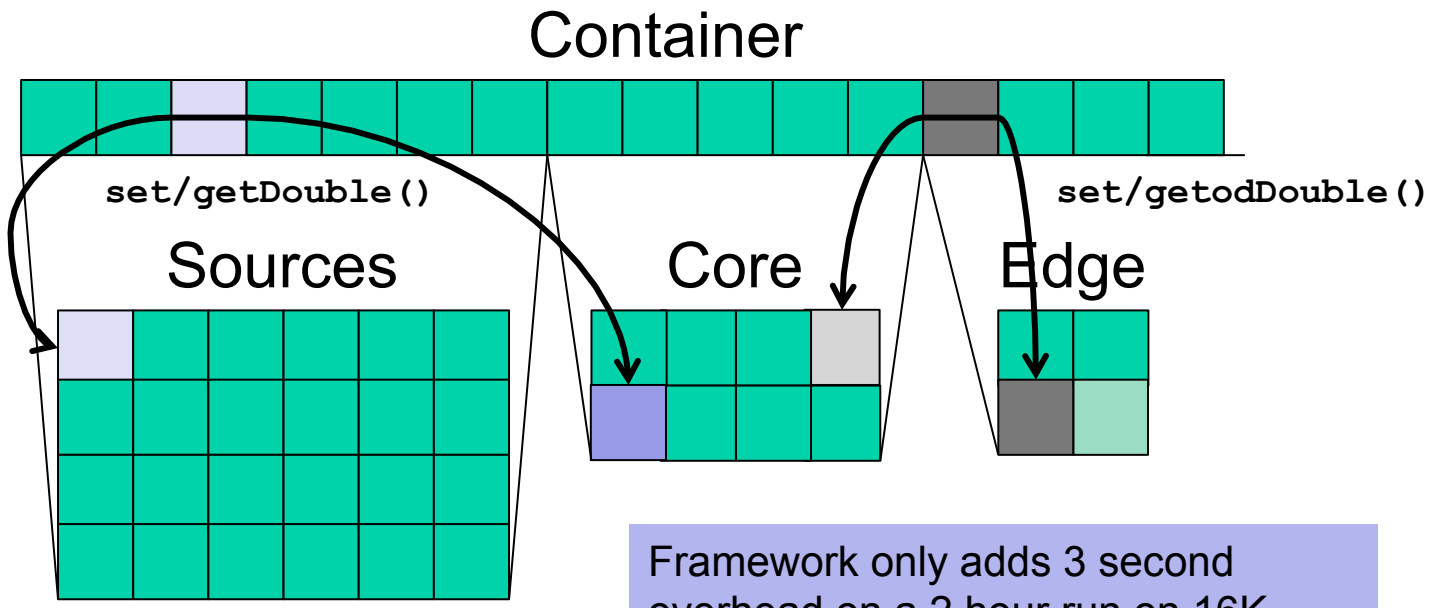
Workflow can be complex for real problems



https://www.facetsproject.org/facets/wiki/CoreEdgeWorkflowDetails

Framework communicates with interface ranks of each component

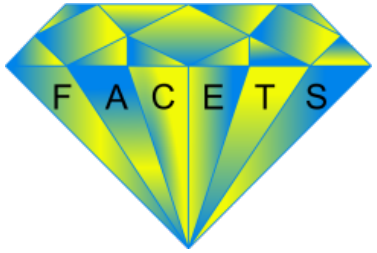
The framework is in control of all coupling. The communication will initiate with a call to `getOdDouble()`. The parent will move the returned data to the appropriate rank, then call `setOdDouble()` accordingly.



Framework only adds 3 second overhead on a 2 hour run on 16K processors.

Conclusion: FACETS has developed a sophisticated parallel HPC framework for multiphysics simul ations

- FACETS framework orchestrates all aspects of the simulation life-time
- Framework provides infrastructure to write new physics components and algorithms and bring-in legacy components
- Framework provides well-defined API for components and coupling.
- Sophisticated processor decomposition stratergy is being integrated into FACETS to make efficient use of LCFs
- New physics (both fusion and other plasma physics) has been enabled by the framework



Extra slides