

# Designing Large Multiphysics Applications

Experiences from SHARP,  
FLASH, and QGMG

Andrew Siegel  
Argonne National Lab

# Premise

- Considerable advances in industrial software engineering/process over past 10-20 years
- Generally speaking, many of these advances have not found their way into scientific “codes”
  - More of a cottage industry
  - Physicists not formally trained in software engineering
  - Numerical models not “robust” -- tendency to focus on numerical algorithms
  - (Less true for middleware)
- Ambitious integrated, multi-physics, multidisciplinary, million-line code efforts require modern software practices
- Scientific software has unique properties which complicate application of industry software ideas ... we will discuss

# What has changed

- Move to increased realism, simulation of complete system
  - Multiply physical regimes, effects, etc.
- Move to “community codes”
  - To minimize duplication of effort
  - Documentation, testing, versioning, training, etc. become important
- Emphasis on performance
  - Extreme scalability
  - Single processor optimization
  - Both are research topics in their own right
- Distributed development teams
  - “can I add my module to your code”, etc.

# By contrast

“Acting on this conclusion, in 1961 I asked R.W. Bray, then a research student at Cambridge, to make a study of the feasibility of integrating numerically the Navier-Stokes equations for 2d homogeneous turbulence over a large enough time interval to reveal settled statistical properties which were not dependent on the form of the initial conditions. The digital computer available at Cambridge at that time (ESDAC 2) was a rather small one by today’s standards, and the results of Bray’s numerical investigation ... were not thought to be sufficiently extensive or decisive to warrant publication. Limited though the results were, they did show the potentialities of the approach, and I hope that the brief description of them to be given here will induce someone with a larger computer to take the investigation further.”

G. K. Batchelor, 1969 “Computation of Energy Spectrum in Homogeneous 2d Turbulence”  
*High-Speed Computing in Fluid Dynamics, Phy. Fluids Supplement*

# Our recent large code projects

- SHARP (2005-)
  - Simulation-Based High-Efficiency Advanced Reactor Prototyping
  - Goal: Use simulation to improve reactor design
    - Enhanced safety, economics, reduced development time
  - Funded: DOE-NE (so far)
    - Target: \$5-10 million per year
  - Team:
    - Nuclear engineers : Designers, modelers, safety analyzers, experimentalists
    - Comput\* Scientists: Includes applied mathematicians

# *Flash Code*

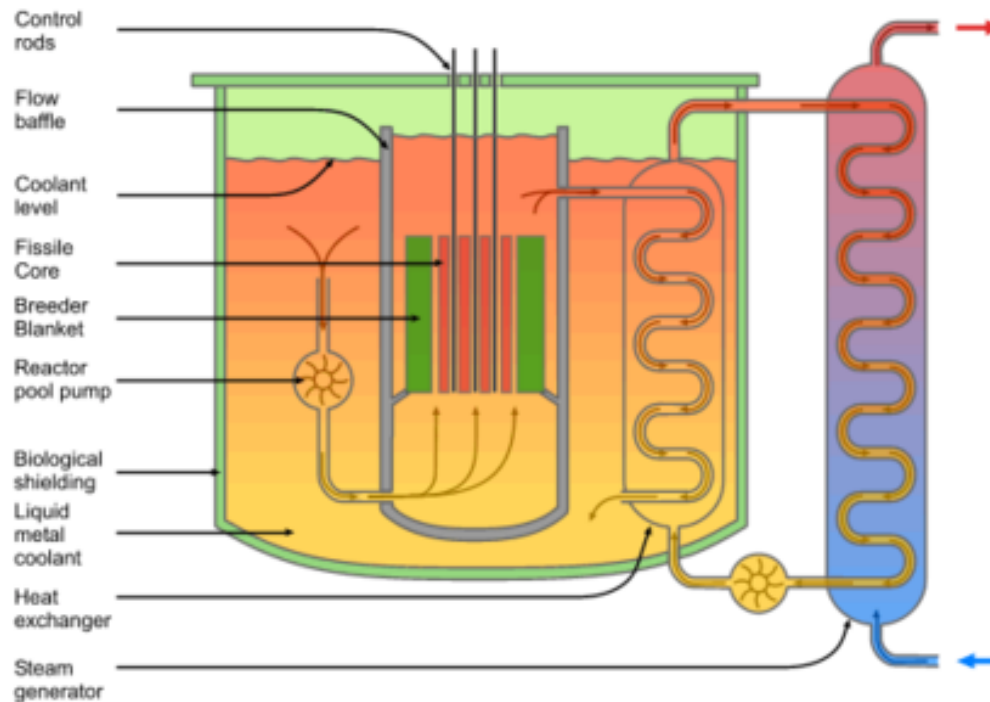
- FLASH (1998-)
  - Goal: Use simulation study fundamental physics of thermonuclear flashes, supernovae, etc.
  - Funded: NNSA (ASC Alliance)
    - \$4-5 million/year
  - Open source community code
    - Hundreds of downloads
    - Close to 1.e6 lines of code
    - Significant emphasis on use of leadership class architectures
  - Team:
    - Astrophysicists : theorists, modeler
    - Experimentalists
    - Comput\* Scientists: Includes applied mathematicians

# Cartoon of fast reactor

fig:01



## Liquid Metal cooled Fast Breeder Reactors (LMFBR) "Pool" Design



### Physics

- thermal hydraulics
- neutronics
- fuel behavior □ □
- structural mechanics

### Code requirements

- multiple implementations
- Complicated meshes
- Huge resolution (petascale)
- vis, i/o
- SGS modeling
- nonlinear coupling
- community code
- validation
- verification
- uncertainty analysis

# What do we hope to learn?

SciFi

“Virtual Experimental Facility”  
(run a reactor to study anything of interest)

- First-principles physics
- Replace experiment
- Virtual prototyping

Future

Predict Specific Global  
Properties  
(mixing, mean temp profile)

- Some modeling required
- Imperfect physics □ □
- Huge range in scales

Traditional

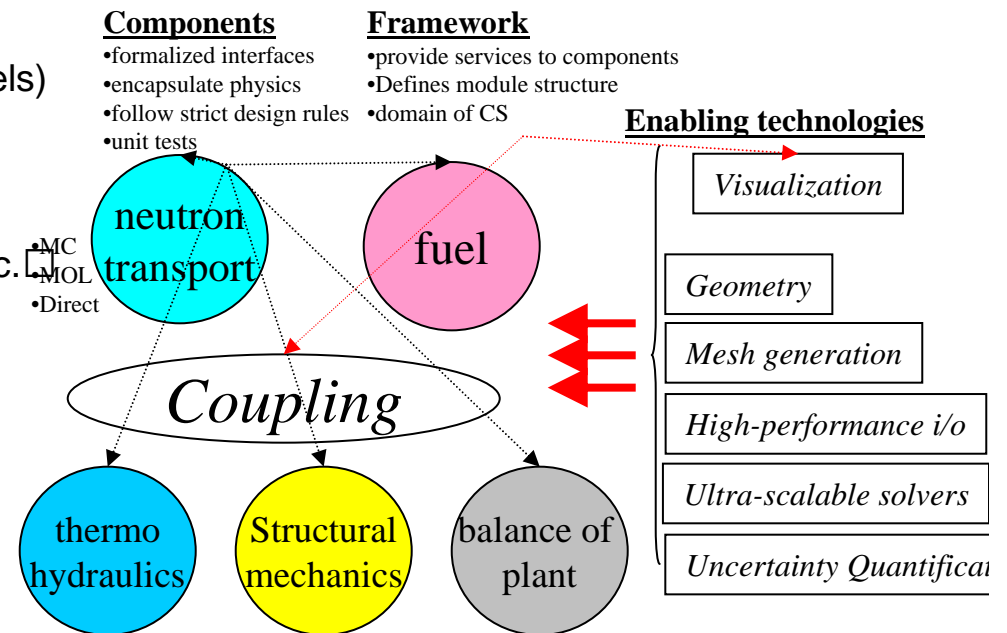
Extrapolate Empirical  
Correlations

- Based on experimental data
- Simple e.g. 1d physics

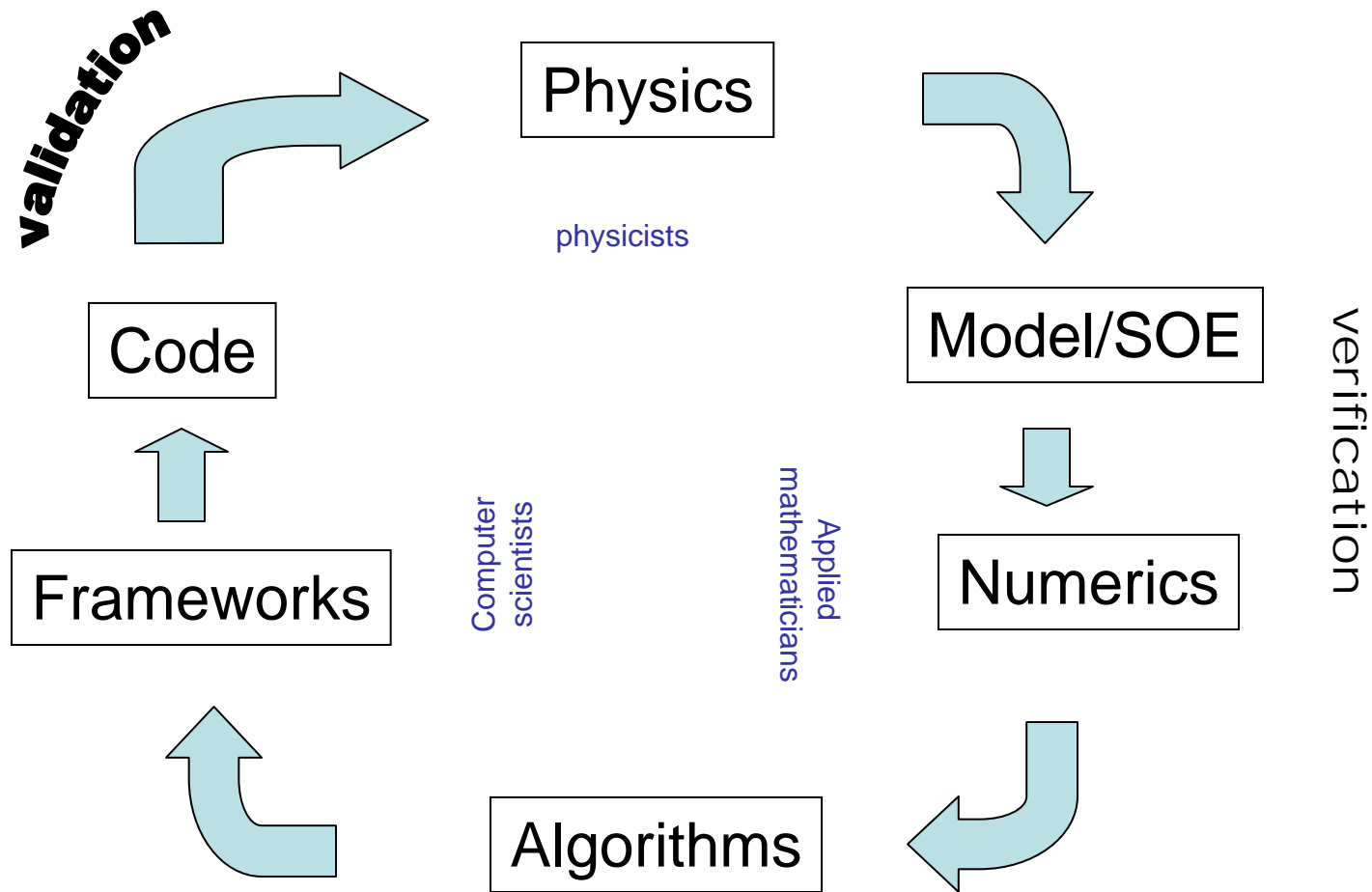
Fundamental insight

# What are key challenges?

- Major software challenge to design flexible architecture combining
  - Both strong and weak inter-component coupling
  - Different (overlapping) meshes for different physics
  - Multiple “swappable” implementations of same physics
  - Portability (from desktop to petascale architectures)
  - Built-in uncertainty analysis/validation
  - Reasonable UI layer for designer ease-of-use
- Individual module model validation/improvement still #1 issue
  - “first-principles” approach still requires deep modeling insight
  - areas of poorly understood physics (e.g. fuels)
- We are not starting from scratch
  - Ten years of ASC program
  - Leverage other DOE tools from SciDAC, etc.
    - *PETSc, CUBIT, VISIT, Nek, ...*
- Tightly integrated teams of designers with computational scientists, etc.
  - End users must be deeply involved throughout the entire process
  - Meet daily...



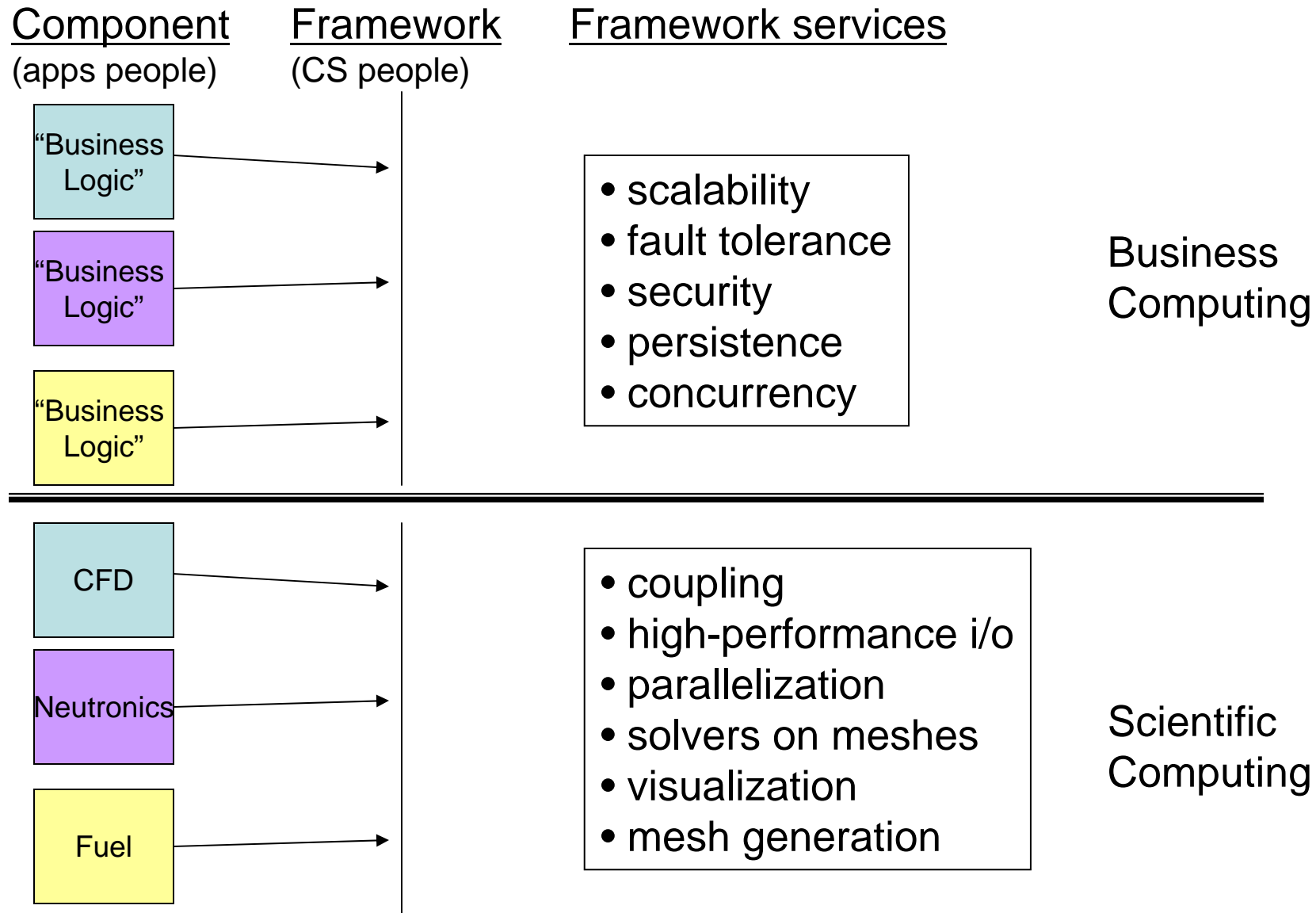
# This requires a range of expertise



# How to manage complexity

- Complexity --> (number of lines)<sup>2</sup>
- Modern multi-physics simulation ~ 1.e6 lines of code
- How to manage this
  - divide and conquer (oldest strategy in SE)
    - Define separate autonomous software entities that interact through well defined interfaces
    - Teams focus on their own domain and do not need to be aware of internal details of other parts
    - Code is “architected” at high-level before individual components are designed -- this is where interfaces and points of coupling are defined
- Question: how realistic is this for physics?

# Basic idea of components and frameworks



# Has this model succeeded?

- It's a mixed story for scientific computing
- Some aspects decouple more naturally than others
  - I/O, visualization, meshing, messaging framework, testing framework, solvers
  - Many modern scientific program leverage 3rd party tools in these areas and/or can divide labor internally
- Others aspects more difficult to separate
  - Physics, parallelization, performance
  - Off the shelf components not used
  - Few successes even within a software project

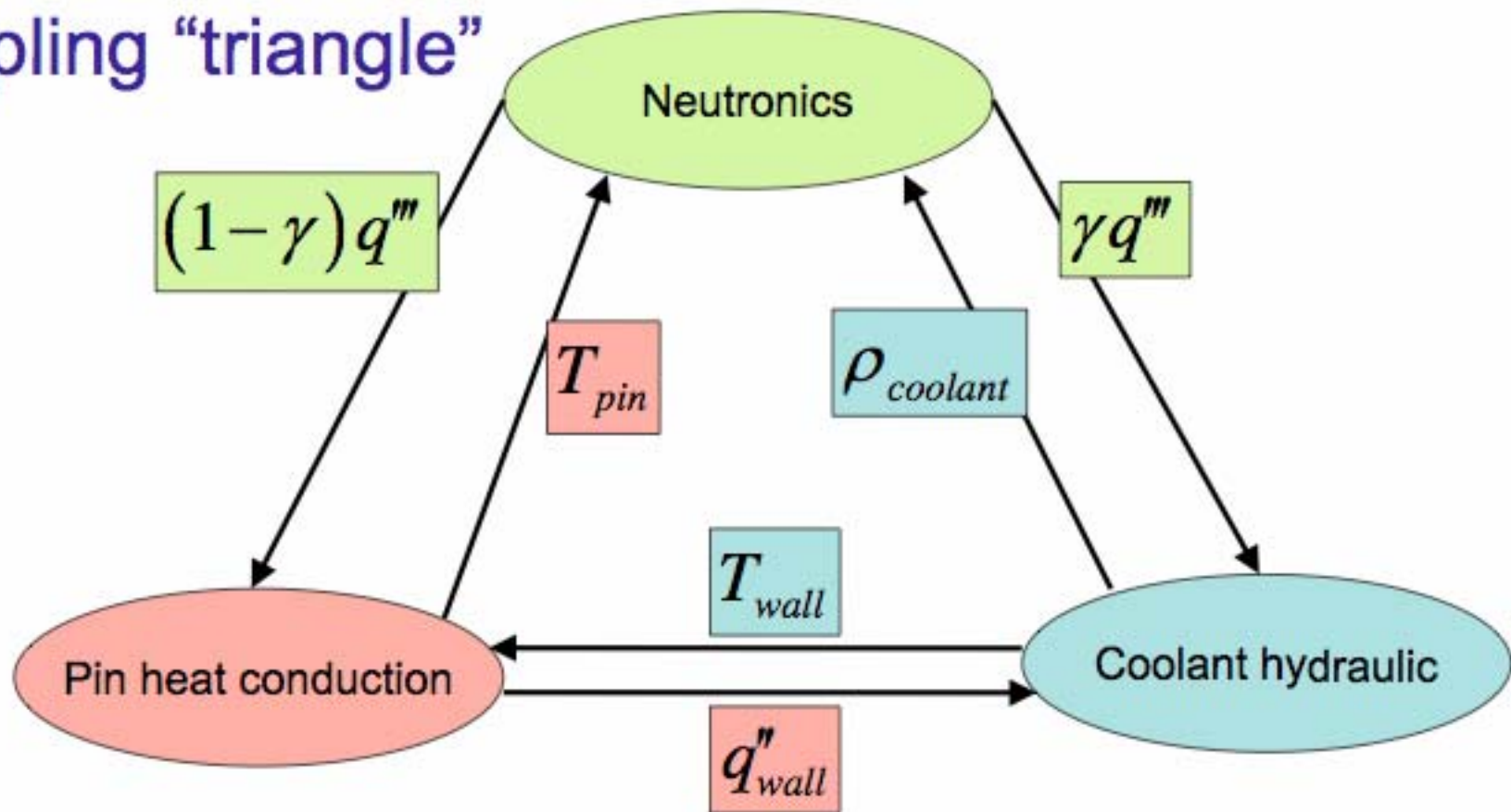
# Why decompose physics?

- Other reasons to decompose physics
  - Leverage existing “legacy” codes
  - Leverage concentrated domain expertise
  - Mix and match for general simulation tool
  - External collaborators can more easily build-in their components
  - Easier to run in single-physics mode
  - Easier to learn for new developers

# Challenges of coupling

- This brings us to the concept of multi-physics coupling
- Key classifications that affect software decisions
  - Is coupling co-located or through interface?
  - Are different meshes required in case 1?
  - Is system nonlinear?
  - What is system stiffness?
  - Where are interfaces likely to change?
  - How much data must be exchanged compared to memory storage?
  - How to do most flexibly/efficiently in parallel

# Coupling "triangle"



$q'''$  = heat generation rate density ( $\text{W}/\text{m}^3$ )  
 $\gamma$  = fraction of heat generated directly in the coolant

$q''_{wall}$  = heat flux at the pin wall ( $\text{W}/\text{m}^2$ )  
 $T_{pin}$  = temperature distribution in the fuel pin ( $^{\circ}\text{C}$  or  $\text{K}$ )

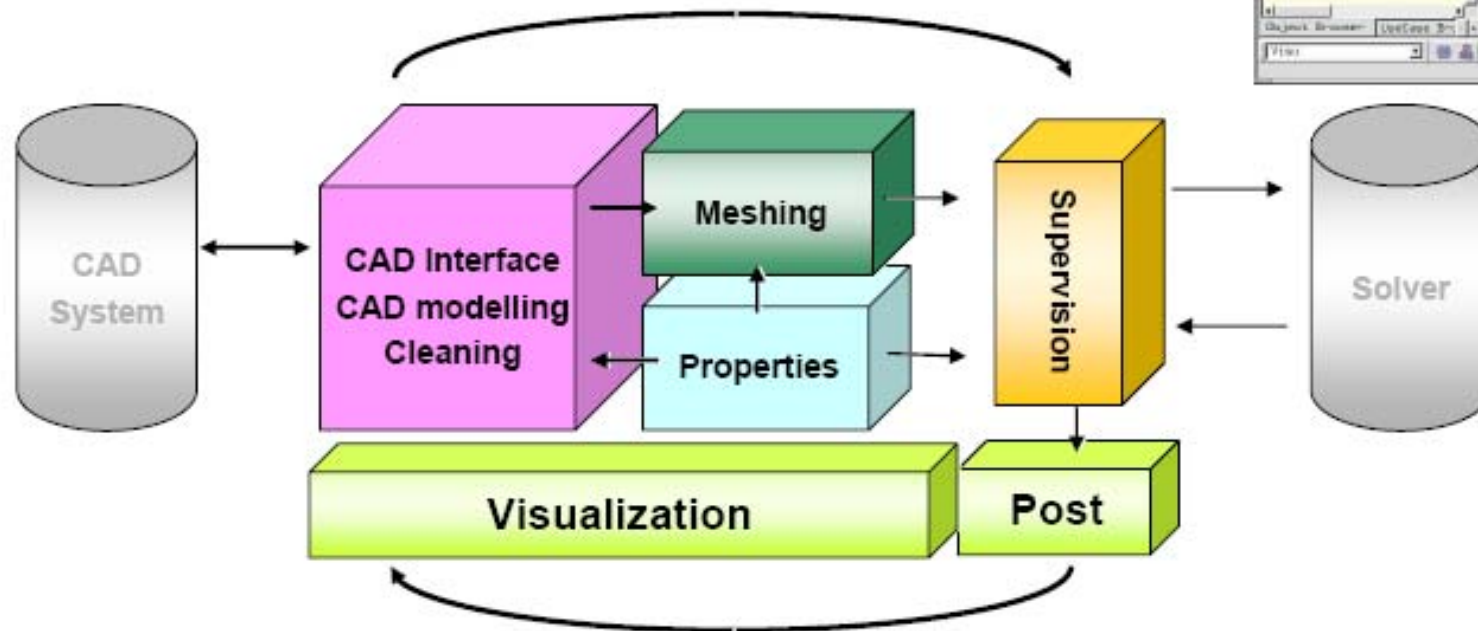
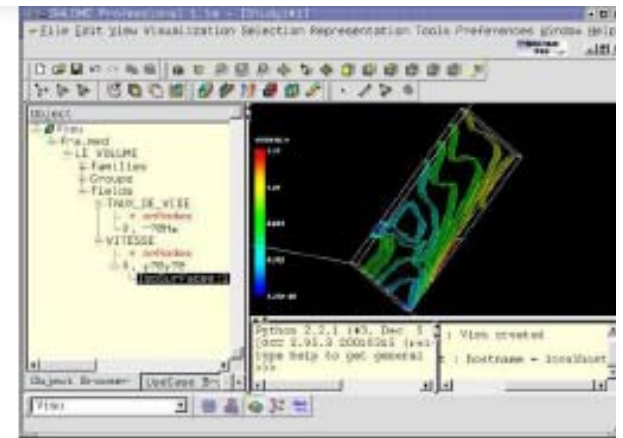
$T_{wall}$  = temperature distribution at the wall ( $^{\circ}\text{C}$  or  $\text{K}$ )  
 $\rho_{coolant}$  = density of the coolant ( $\text{kg}/\text{m}^3$ )

# Other examples

- Coupling of atmosphere-ocean model
  - Heat/momentum exchange at interface
- Coupling of structural mechanism shock-capturing fluids
  - Pressure transfer at boundary
- Coupling of nuclear synthesis to shock-capturing fluids for thermonuclear flashes
- Single phase to multi-phase fluid coupling

# French have been working on this ... Salome Toolkit

**A generic development platform for  
pre/post processing and code coupling  
for numerical simulation**



**Open-source project by the SALOME RNTL network :**

<http://www.salome-platform.org>

# Designing the team

- This is still a good model
- Some success in business computing
- More complicated for scientific computing -- why?

# Early Results

- In research projects, typically need to show sponsor early science results (e.g. within one year)
- This puts a constraint on design process
  - Must take path for short-term gains
  - These design choices are very hard to undo
- In industrial software one sees much more of an emphasis/appreciation of the complexity/importance of the overall architectural design phase.
- That said ...

# Unknown Requirements

- Scientific software often has much more of a research component and all of the requirements cannot simply be laid out ahead of time
- Need to explore physics/numerics early to determine the best approach
  - What numerical techniques work well in physically/geometrically complex regions?
  - Where is higher order multiphysics coupling needed?
  - How much data will I need to dump for certain analyses?
- Ex:
  - SGS models for liquid metals in fast reactor core
  - Treatment of void regions in reactor core
  - Sensitivity of neutron flux to accuracy of input cross section data
- Integrate rapid prototyping tools
  - FreeFEM, MatLab

# Numerics

- Each module box on powerpoint slide not really independent
  - Represents an equation, not necessarily a module
  - Part of integrated, usually nonlinear system
- Can easily separate boxes with time splitting and well defined interfaces (even though mesh can complicate this)
- However, time splitting can produce large inaccuracies
  - E.g. for fast reactor transients (rod ejection)
- Advanced numerical methods for coupling
  - JFNK methods for coupling
  - Accelerated fixed point methods

# Performance

- Computer scientists prefer general solutions
- Sophisticated code with many options must perform almost as well as any single-purpose code
- Performance often competes with modularity - specific solutions typically faster

Computer scientists like to do things like this:

```
fnew = f.getVal(i,j) + f.getVal(i,j+1) + f.getVal(i+1,j) + f.getVal(i-1,j) - 4*f.getVal(i,j);  
f.setVal(i,j,fnew);
```

- Architect at a high-level where performance losses are negligible -> use familiar “efficient” languages for intensive operations

# Mutual research

- Research environment is unique
- Computer scientists cannot become code hackers
- Most PI's not managers in industry sense
- Scientists resist using their code as testbed for new CS concepts
- Physics must drive design decisions

# Rapidly changing computing environment

- Need to remain flexible to leverage leadership-class computing facilities
  - Vector vs. cache-based
  - Global metadata on BG/L
  - Topology-aware algorithms
- Performance model key

# Developers=Users

- Typically, developers and main users of code are the same group
- This means that, when a new major version is created, important bug fixes and features need to continuously be added to old version
  - This ensures continuity of science research
  - Decrees to freeze the source don't work well
- Must devise strategy to get new code  $\geq$  old version capabilities within a month, maximum

# Summary

- Large R&D multiphysics software projects are unlikely to succeed both
  - using “conventional” development models derived strictly from other industries
  - using an ad hoc development process appropriate for smaller projects
- CCA work taking key steps toward providing tools/standards directed at HPC

# Role of the Code Group

Astro      Astrophysics Group  
CP         Computational Physics Group  
Code       Code Group  
CS         Computer Science Group

