Managing Complexity in Modern High-End Scientific Computing through Component-Based Software Engineering

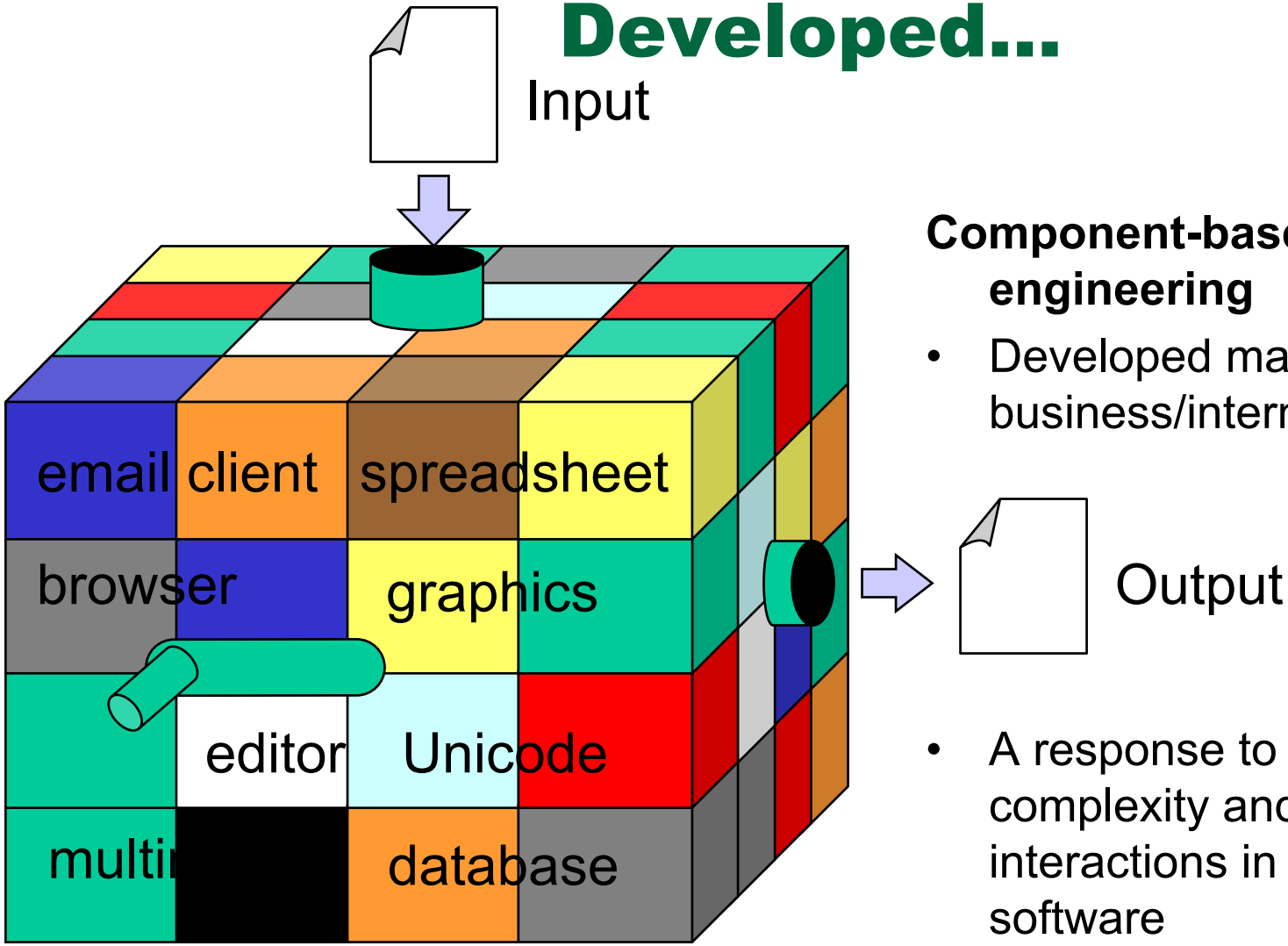
David E. Bernholdt
Oak Ridge National Laboratory

Robert C. Armstrong and Benjamin A. Allan
Sandia National Laboratories

The Recent History of HPC Scientific Computing

- Focus of software development has been…
  - Performance (time to solution)
  - Fidelity (sophistication and accuracy of model)
  - Problem size
- Combined with increasing complexity of HPC hardware, and…
- Overall success of computational simulation, has produced…
- An explosion in the complexity of HPC scientific software
  - “Grin and bear it” approach to managing complexity typical
  - Obvious impacts on productivity

Meanwhile, CBSE was Being Developed…



Component-based software engineering

- Developed mainly in business/internet community
- A response to increasing complexity and breadth of interactions in business software

Basic Principles of Component-Based Software Engineering

- Encapsulate software functionality in *components*
- Treat components as black boxes
  - Implementation is opaque to other components
- Components interact only through well-defined *interfaces*
- Components can be assembled into applications by connecting their interfaces appropriately within a *framework*

CBSE, Productivity, and Performance In General

- **Software productivity**
  - Provides a "plug and play" application development environment
  - Many components available "off the shelf"
  - Abstract interfaces facilitate reuse and interoperability of software
- *"The best software is code you don't have to write" [Jobs]*
- **Software complexity**
  - Components encapsulate much complexity into "black boxes"
  - Plug and play approach simplifies applications & adaptation
  - Model coupling is natural in component-based approach
- **Software performance** (indirect)
  - Plug and play approach and rich "off the shelf" component library simplify changes to accommodate different platforms

The Common Component Architecture

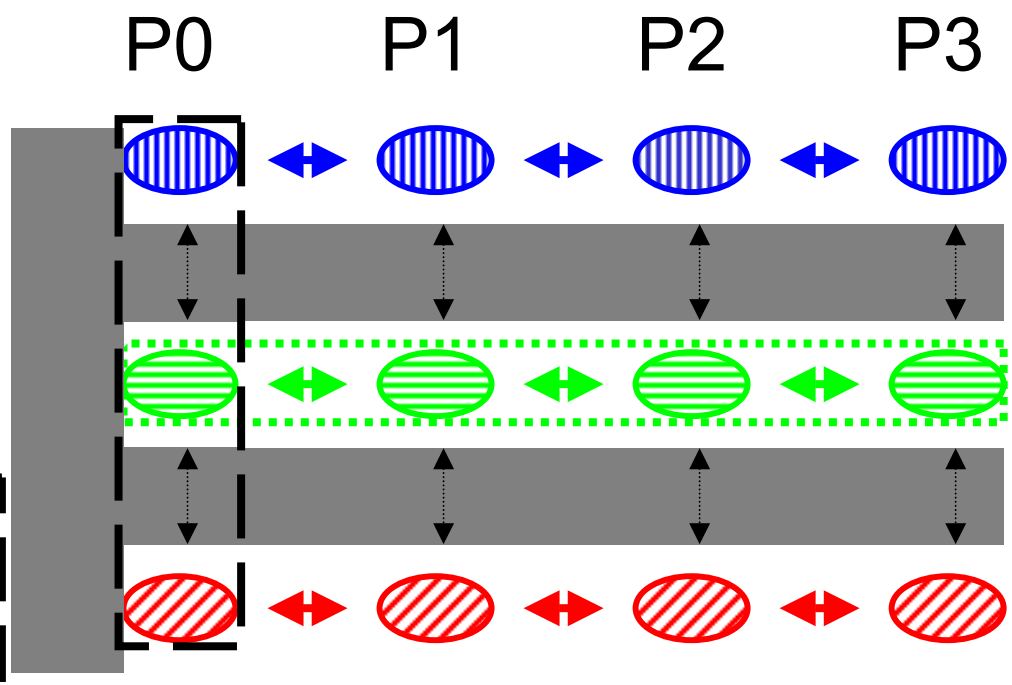
- A component model specifically designed for high-performance scientific computing
  - "Commodity" component models (i.e. CORBA, COM, EJB) aren't adequate
- Supports both parallel and distributed applications
  - Commodity models focus on distributed only
- Designed to be implementable without sacrificing performance
  - Distributed timescales *O(ms-s)*; parallel timescale *O(ns-μs)*
- Minimalist approach makes it easier to componentize existing software
  - CCA requires just one new method, minimal modifications elsewhere to be a useful component

Local High Performance: Direct Connection

- Calls *between* components equivalent to a C++ virtual function call: lookup function location, invoke it
  - Cost equivalent of ~2.8 F77 or C function calls
  - ~48 ns vs 17 ns on 500 MHz Pentium III Linux box
- Language interoperability can impose additional overheads
  - Some arguments require conversion
  - Costs vary, but small for typical scientific computing needs
- Calls within components have no CCA-imposed overhead
- **Implications**
  - Be aware of costs
  - Design so inter-component calls do enough work that overhead is negligible

High Performance and Parallelism

- Single component multiple data (SCMD) model is component analog of widely used SPMD model
- Each process loaded with the same set of components wired the same way
- Different components in same process "talk to each" other via ports and the framework
- ***Same component in different processes talk to each other through their favorite communications layer (i.e. MPI, PVM, GA)***



Components: Blue, Green, Red

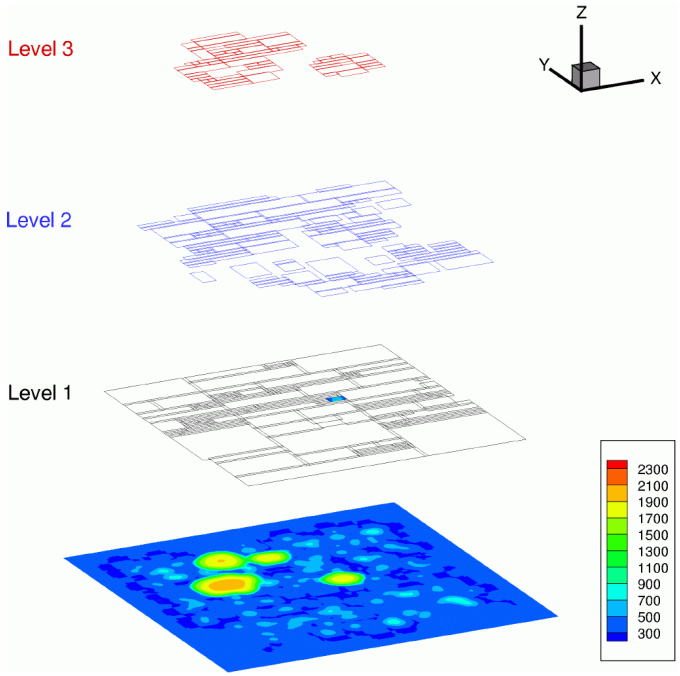
Framework: Gray

MCMD/MPMD also supported
Other component models ignore parallelism entirely

Complexity in CCA-Based Applications

- Components *are* a means to encapsulate complexity, but …
- *Assemblies* of components can become large and complex, especially if components are fine-grained
- Visual application assembly also has screen real estate constraints

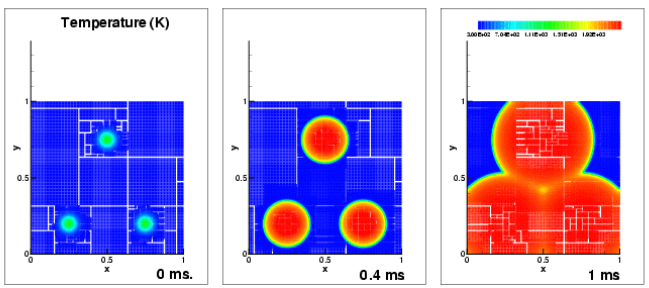
Computational Facility for Reacting Flow Science (CFRFS)



- SciDAC BES project, H. Najm PI
- **Investigators:** Sofia Lefantzi (SNL), Jaideep Ray (SNL), Sameer Shende (Oregon)
- **Goal:** A “plug-and-play” toolkit environment for flame simulations
- **Problem Domain:** Structured adaptive mesh refinement solutions to reaction-diffusion problems

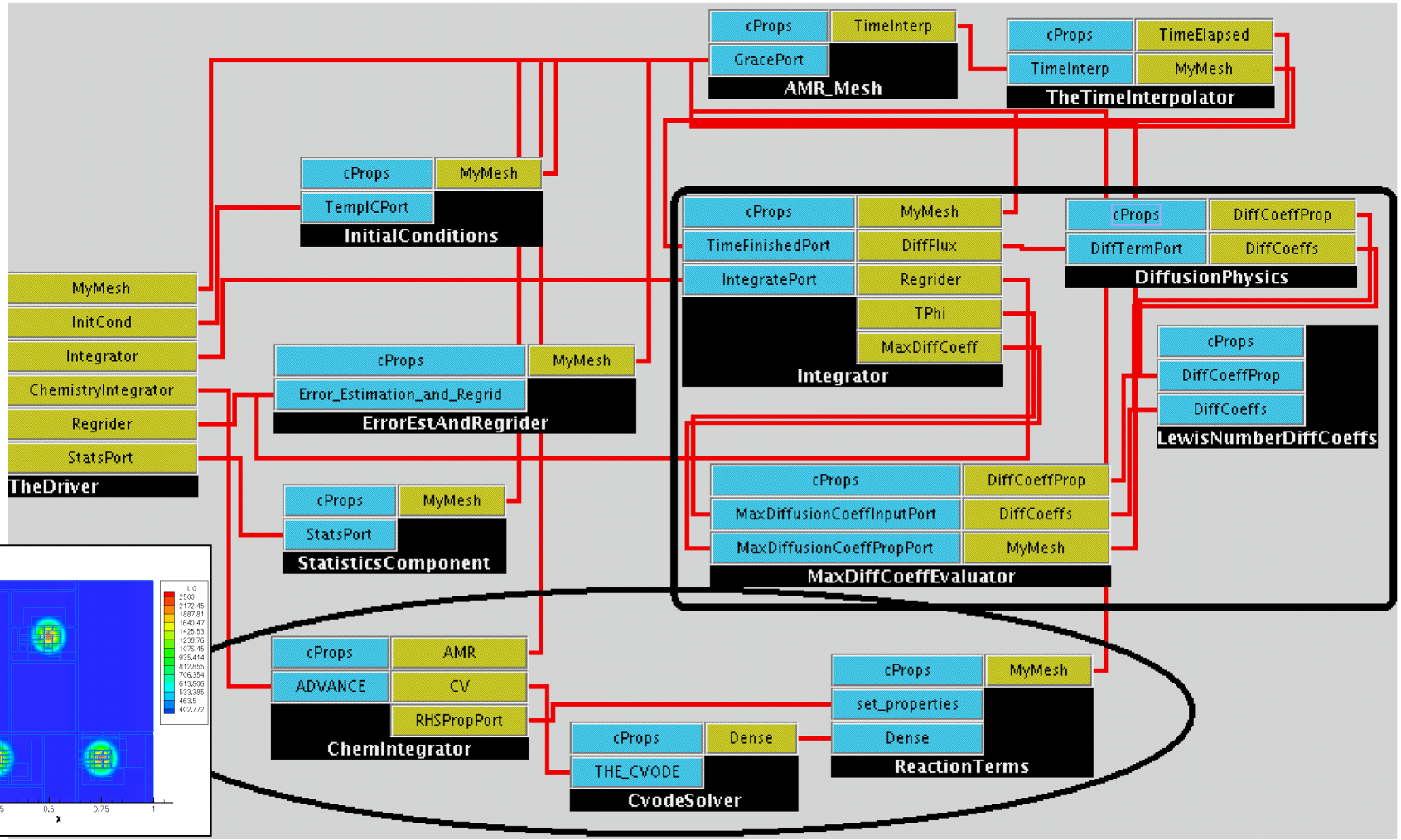
Scientific and Technical Summary

- H_2-Air ignition on a structured adaptive mesh, with an operator-split formulation
- RKC for non-stiff terms, BDF for stiff
- 9-species, 19-reactions, stiff mechanism
- 1cm x 1cm domain; max resolution = 12.5 microns
- Kernel for a 3D, adaptive mesh low Mach number flame simulation capability in SNL



- Components are usually in C++ or wrappers around old F77 code
- Developed numerous components
  - Integrator, spatial discretizations, chemical rates evalutator, transport property models, timers etc.
  - Structured adaptive mesh, load-balancers, error-estimators (for refining/coarsening)
  - In-core, off-machine, data transfers for post-processing
- TAU for timing (Oregon)
- CVODES integrator (LLNL)

"Wiring Diagram" for Typical CFRFS Application



BuilderService: A Standard CCA Framework Service

- Primary purpose of BuilderService is to allow programmatic assembly of applications
- Supports instantiating and destroying components and connecting ports of components
- Allows dynamically programmable behavior during application execution
- Can be used to create containers to aggregate components

Aggregation with BuilderService

Step1: Container gets BuilderService port

```
svc.registerUsesPort("gov.cca.BuilderService",
                                myBuilderService, properties);

gov.cca.BuilderServices bSvc = getPort("myBuilderService");
```

ChemSolver

Aggregation with BuilderService

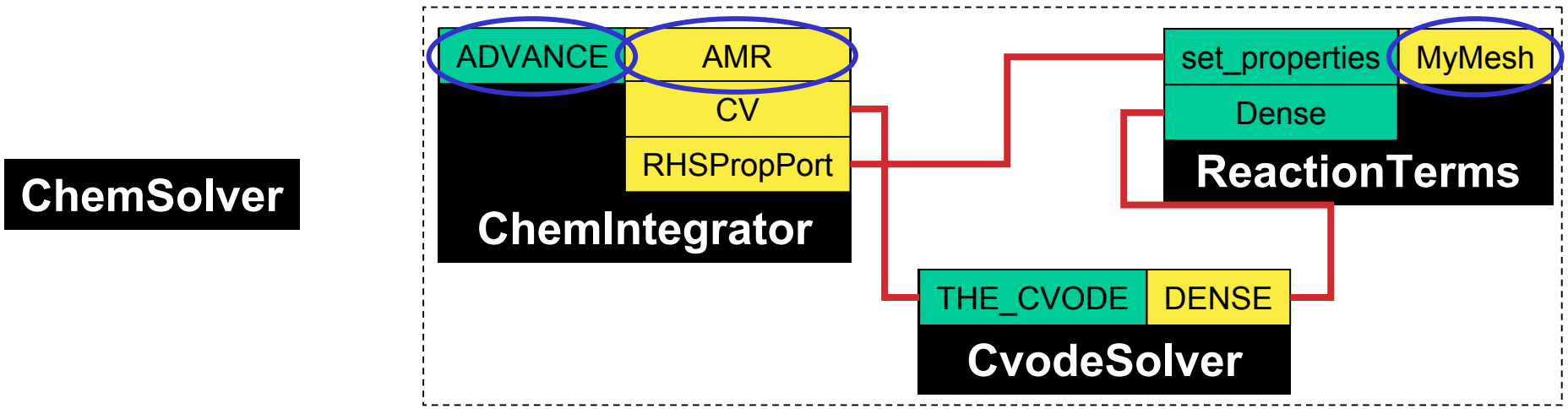
Step 2: Container instantiates and connects containees

```
gov.cca.ComponentID cmpCI = bSvc.createInstance(
        "ChemIntegrator", "CI_Type", properties);

                                    …

bSvc.connect("ChemIntegrator", "CV",
        "CvodeSolver", "THE_CVODE");

                                    …
```

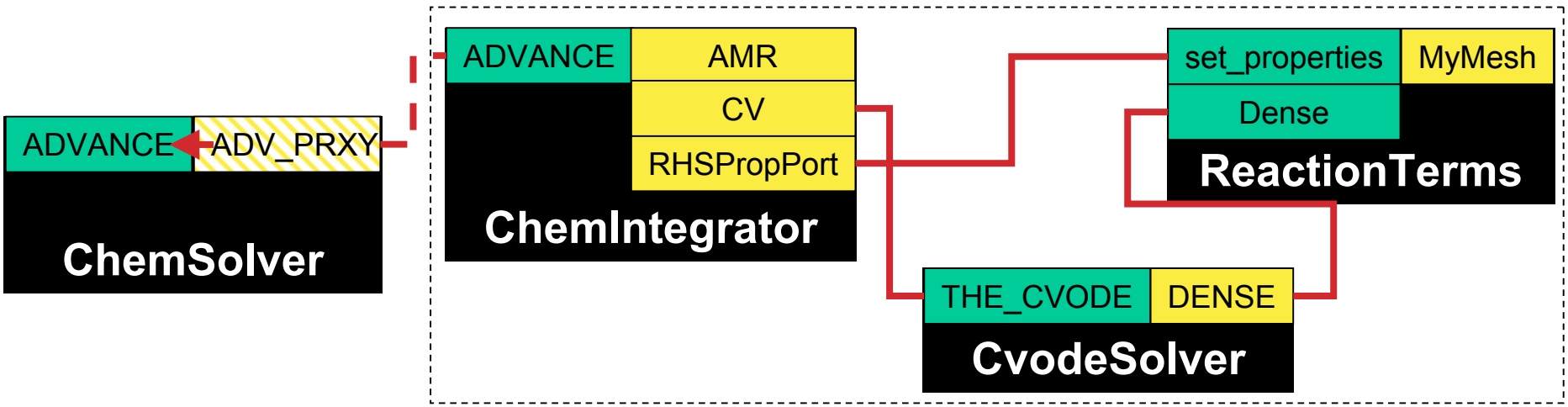


Aggregation with BuilderService

Step 3a: Container exports unconnected *provides* ports

```
svc.registerUsesPort("ADV_PRXY", "ADV_Type", properties);
gov.cca.ComponentID myID = svc.getComponentID();
bSvc.connect(myID, "ADV_PRXY", cmptCI, "ADVANCE");

gov.cca.Port advProxy = svc.getPort("ADV_PRXY");
svc.addProvidesPort(advProxy, "ADVANCE", "ADV_TYPE", properties);
```



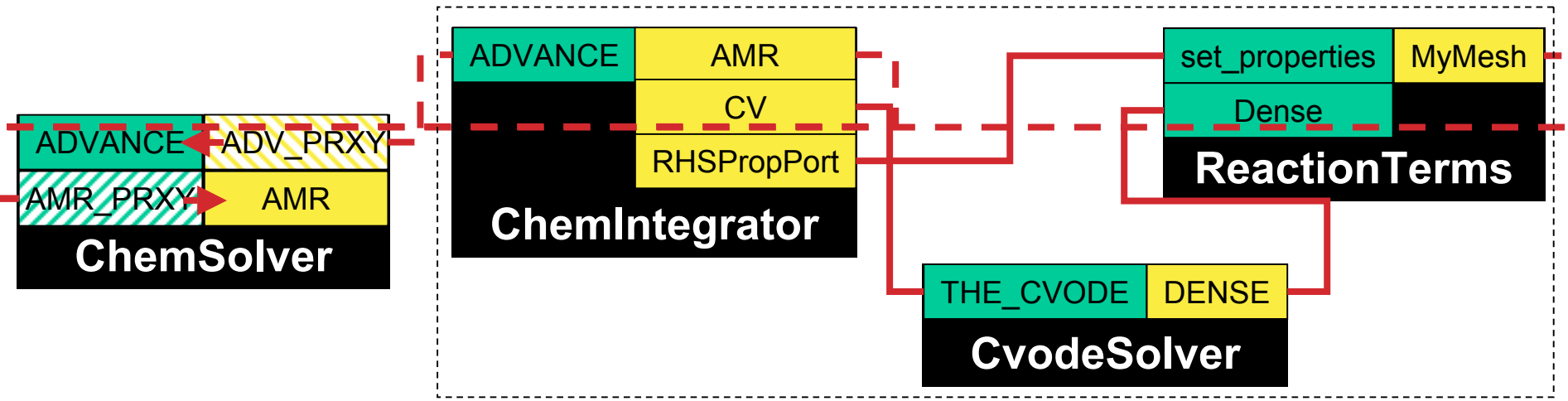
Aggregation with BuilderService

Step 3b: Container exports unconnected *uses* ports

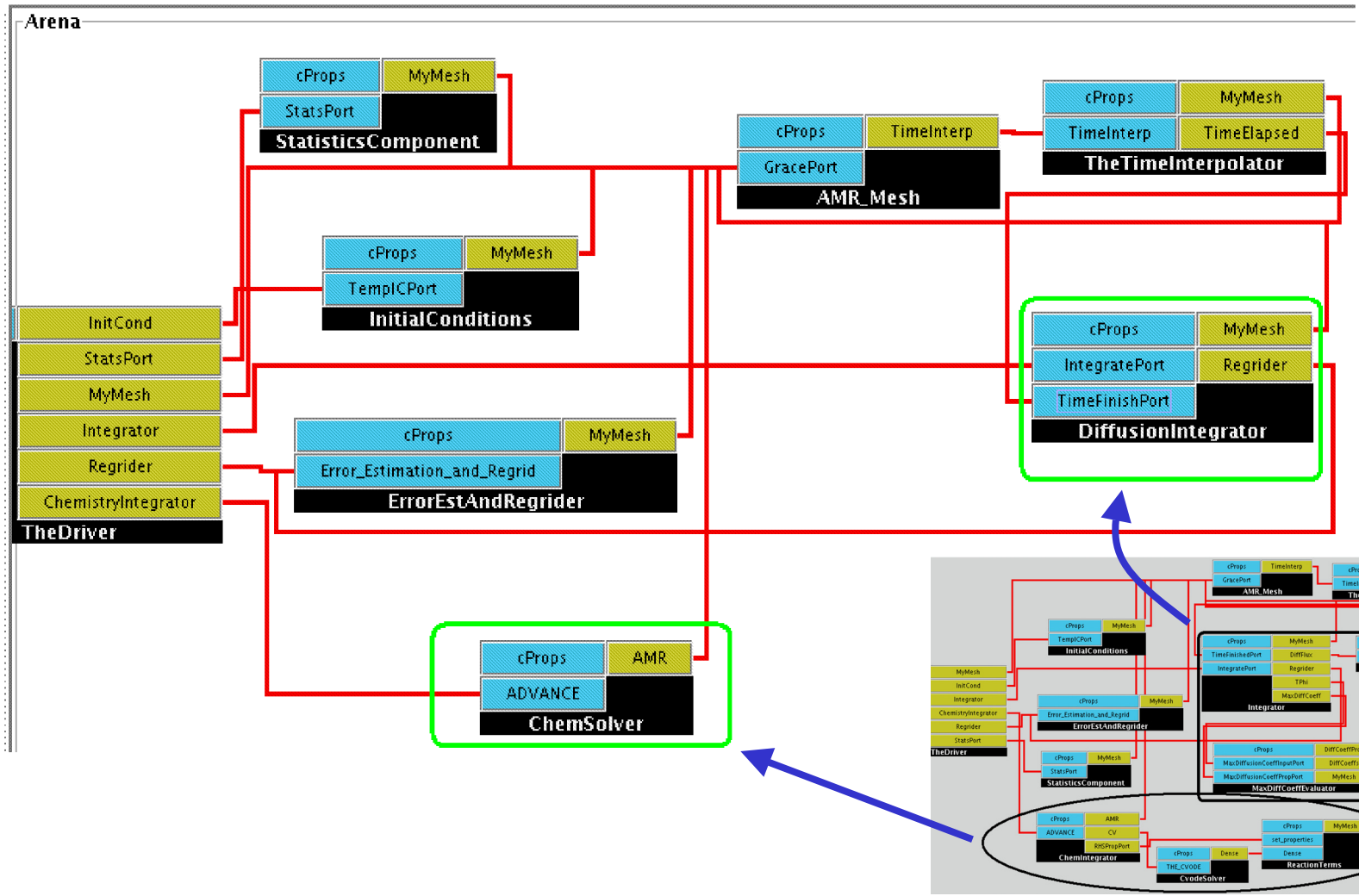
```
svc.registerUsesPort("AMR", "AMR_Type", properties);

// Safe only after our uses AMR port is connected externally
gov.cca.Port amrPort = svc.getPort("AMR");
svc.addProvidesPort(amrPort, "AMR_PRXY", "AMR_Type", properties);

bSvc.connect(cmptCI, "AMR", myID, "AMR_PRXY");
bSvc.connect(cmptRT, "MyMesh", myID, "AMR_PRXY");
```



Simplified CFRFS Application with Containers



Summary

- CCA is intended to improve productivity for a wide range of application areas
  - Simplify creation of large, complex simulation applications
  - Facilitate software interoperability and reuse
  - Maintain performance of non-component code
- BuilderService provides
  - Programmability of component assembly functions
  - Dynamic behavior during execution
  - Aggregation of complex component sub-assemblies
- AbstractFramework allows arbitrary code to act like a CCA framework
- CCA's design reuses basic principles to provide higher-level mechanisms to manage and hide complexity

More Info & Acknowledgements

- Web: http://www.cca-forum.org
- Email: *bernholdtde@ornl.gov*

- Special thanks to Jaideep Ray and Sophia Lefantzi (SNL) for the original CFRFS toolkit components

- Work supported by the US Dept. of Energy Office of Mathematical, Information and Computational Sciences (MICS), Scientific Discovery through Advanced Computing (SciDAC) program
- Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the US Dept. of Energy under contract DE-AC-05-00OR22725.