Component-Based Software for High-Performance Computing: An Introduction to the Common Component Architecture

David E. Bernholdt

Computer Science and Mathematics Division, Oak Ridge National Laboratory

and

Center for Component Technology for Terascale Simulation Software

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

Acknowledgements

- Members of the CCTTSS, especially
  - Rob Armstrong, SNL
  - Lori Frietag, ANL
  - Jim Kohl, ORNL
  - Lois Curfman McInnes, ANL
  - Boyanna Norris, ANL
  - Jaideep Ray, SNL
- This work is sponsored by…
  - US Dept. of Energy
    - Office of Science
      - Advanced Scientific Computing Research (ASCR)
        - Mathematics, Information and Computer Science (MICS)
      - Scientific Discovery through Advanced Computing (SciDAC) program

Challenges in Modern Scientific Software Development

- Desire for rapid development cycle
- Diversity of languages and tools
  - What can be used together?
  - Ability to reuse code from inside or outside organization
- Parallel computing significantly increases algorithmic complexity
  - Where do you find the expertise in all disciplines your code might require?
  - Architectural complexity and diversity of modern MPPs
- Increasing capability of computers leads to new expectations of simulation
  - Higher fidelity, greater sophistication
  - Extension to multi-scale and multi-physics simulations
  - Coupling of simulations

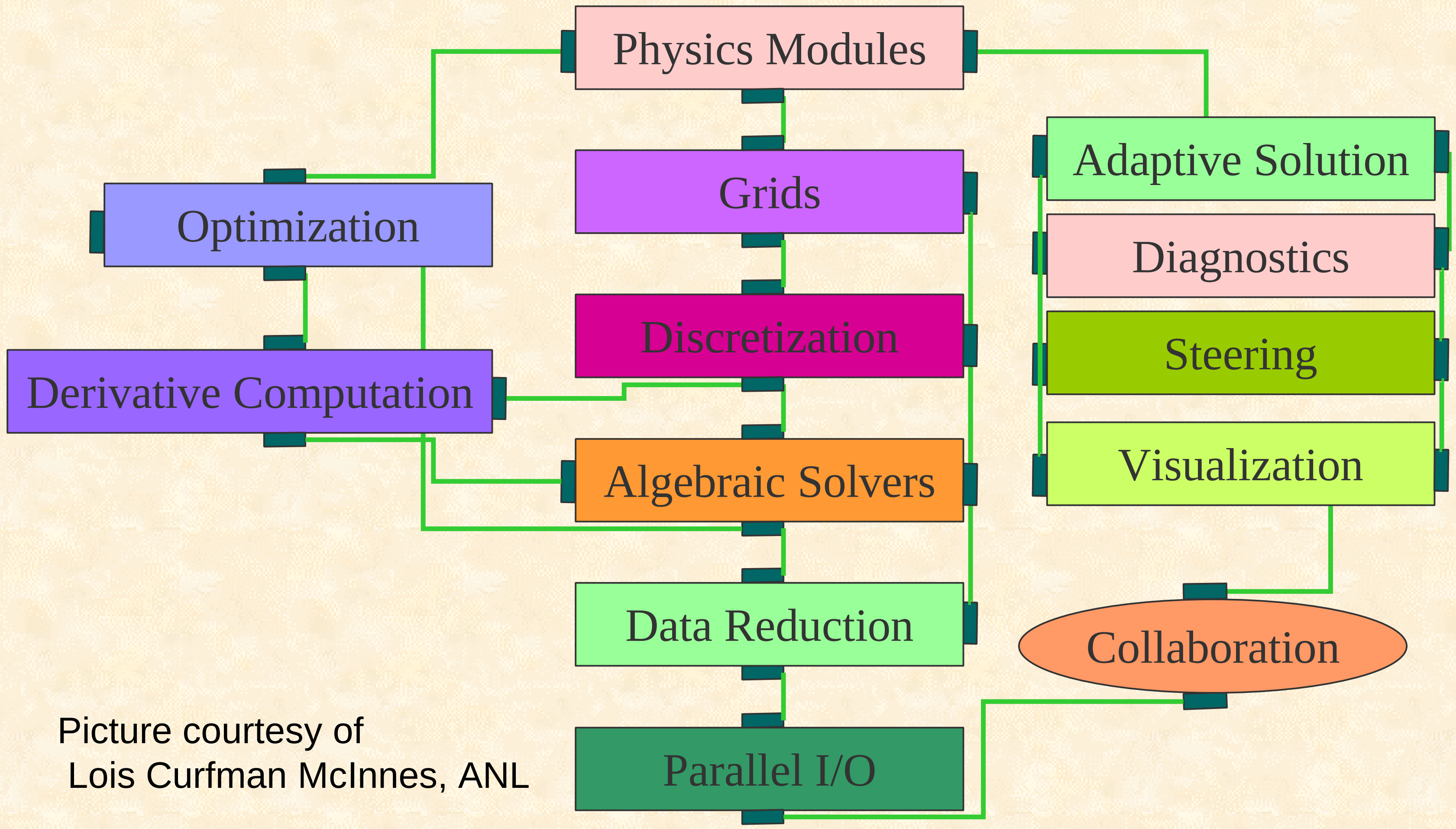
Managing Software Complexity: Object-Oriented Programming

- OO design is a well-established, popular, and widely used methodology
  - Often sold as a means for managing software complexity
  - Fundamental ideas: Modularity, Abstraction, Encapsulation, Hierarchy
- OO programming does not *require* OO languages, but langauge support can be very helpful
  - Support for encapsulation, hierarchy, etc.
- Objects can simplify abstraction and encapsulation, facilitating reuse
  - But deep object hierarchies make reuse harder
- OO languages are still rather immature
  - Reliance on OO language features can make language interoperability hard

Component-Based Programming

- Based on OO ideas, but at a coarser level
- Components encapsulate well-defined units of reusable functionality (in OO sense, often a collection of objects)
- They interact through well-defined interfaces
  - Separates interface from its implementation: “Good fences make good neighbors”
- Components improve modularity and reuse
  - Provides for “plug and play” HPC code.
  - Facilitates exchange of components between groups
  - Facilitates interdisciplinary collaboration in a single app.
  - Allows you to focus on your area of interest/expertise
- Intended to make it easier to *compose* software into a working application
- *Not* a magic bullet
  - Does not alter algorithms: does not speed up their development, or eliminate bugs. Does not make *programming* easier

21st Century Application



Picture courtesy of
Lois Curfman McInnes, ANL

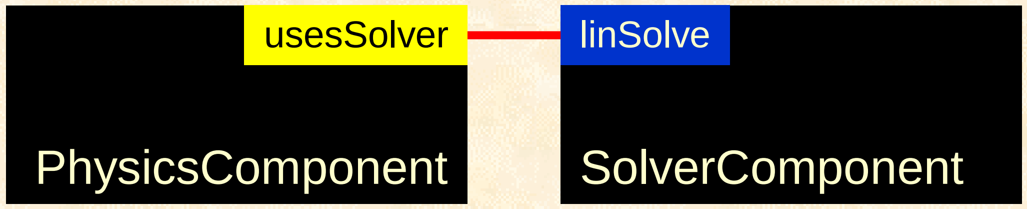
Commodity Component Models

- Component models are common in visualization environments
  - AVS, Data Explorer
  - dataflow-based
- Component-based software development has taken the commercial/industrial world by storm.
  - CORBA, COM/DCOM, Enterprise JavaBeans
- Unfortunately, these systems are not generally suitable for high-performance scientific computing
  - Focus exclusively on local & distributed computing
  - No concept of parallelism
  - High overheads, even for local operation
  - Often platform or language-specific

The Common Component Architecture

- A component model specifically designed for high-performance computing
- Supports both parallel and distributed applications
- Designed to be implementable without sacrificing performance
- Minimalist approach makes it easier to *componentize* existing software

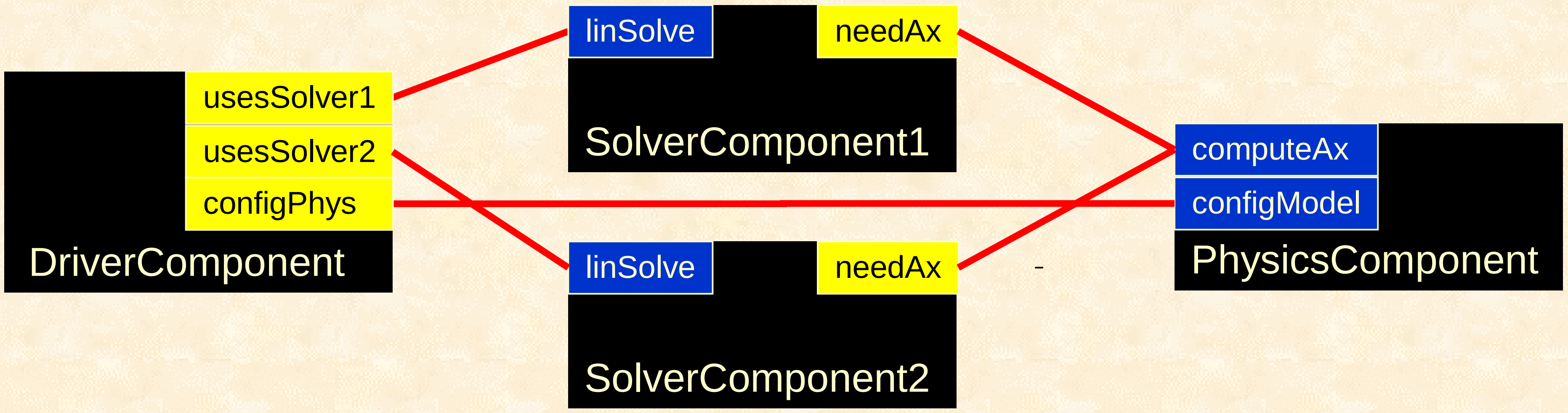
CCA Concepts: Ports



- Components interact through well-defined interfaces, or *ports*
  - In OO language, a port is a class
  - In Fortran languages, a port is a bunch of subroutines
- A given component may *provide* a port – implement the class or subroutines
- Another component may *use* that port – call methods or subroutines in the port.
- Links denote a caller/callee relationship, ***not dataflow!***
  - e.g., linSolve port might contain: *solve(<u>in</u> A, <u>out</u> x, <u>in</u> b)*

A More Complex Example

- A component may *provide* and/or *use* multiple ports
- A component may *provide* some ports and *use* others
- A component may *provide* or *use* multiple instances of the same port

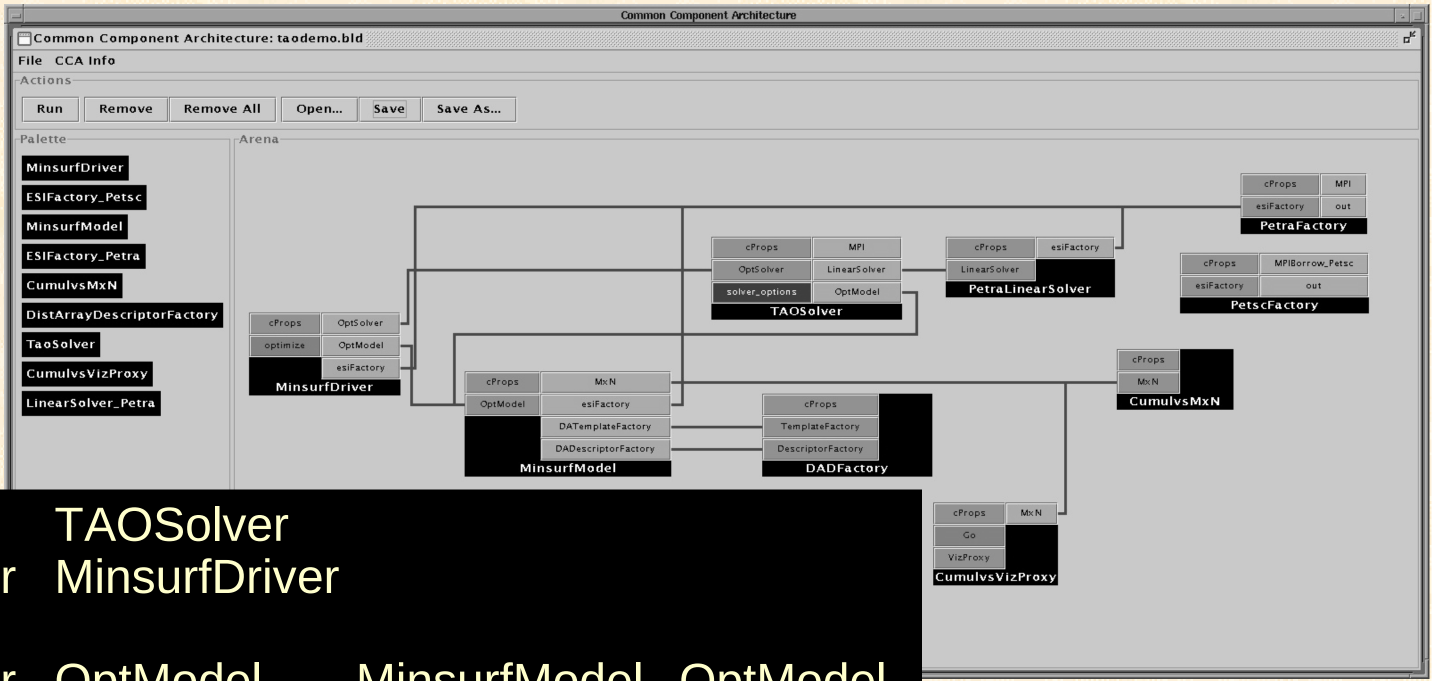


CCA Concepts: Frameworks

- The framework provides the means to “hold” components and compose them into applications
- The framework is the application’s “main” or “program”
- Frameworks allow exchange ports among components without exposing implementation details
- Frameworks may support sequential, distributed, or parallel execution models, or any combination they choose
- Frameworks provide a small set of standard services to components
  - BuilderServices allow programs to compose CCA apps
- Frameworks may make themselves appear as components in order to connect to components in other frameworks

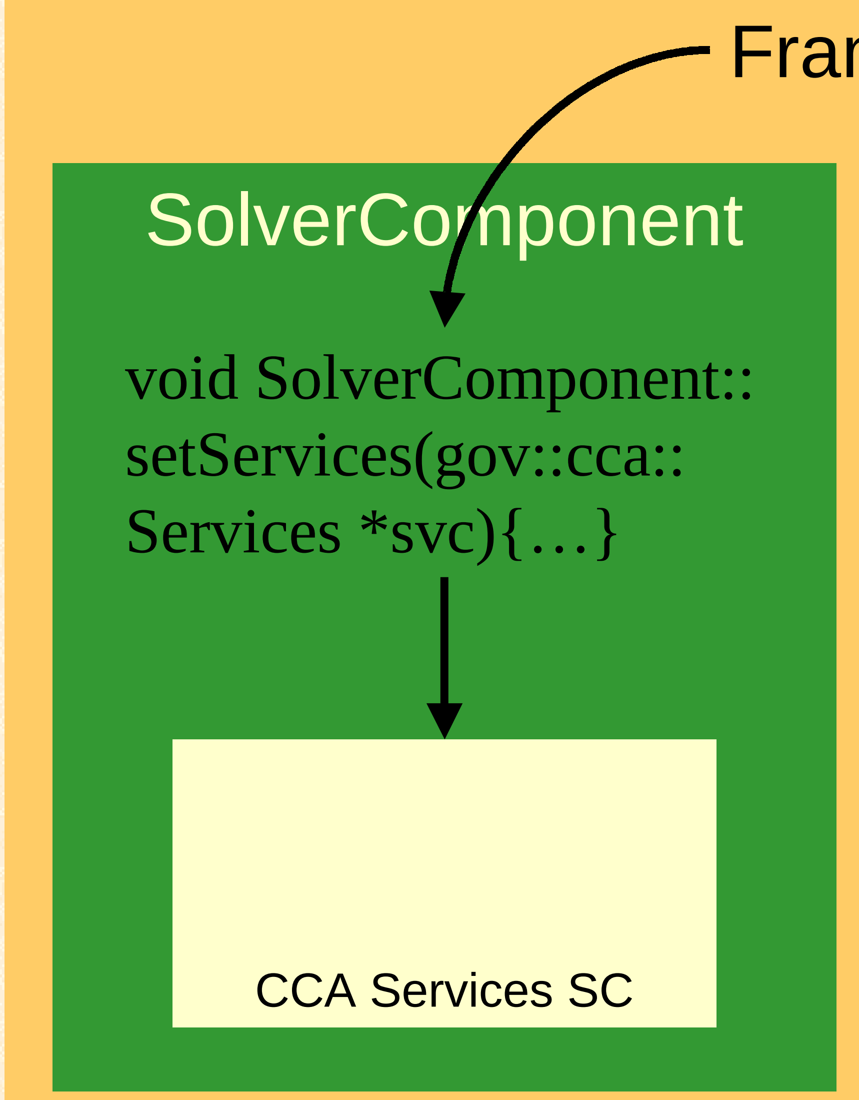
What Does This Look Like?

- Launch framework (w/ or w/o GUI)
- Instantiate components required for app.
- Connect appropriate provided and used ports
- Start application (i.e. click Go port)



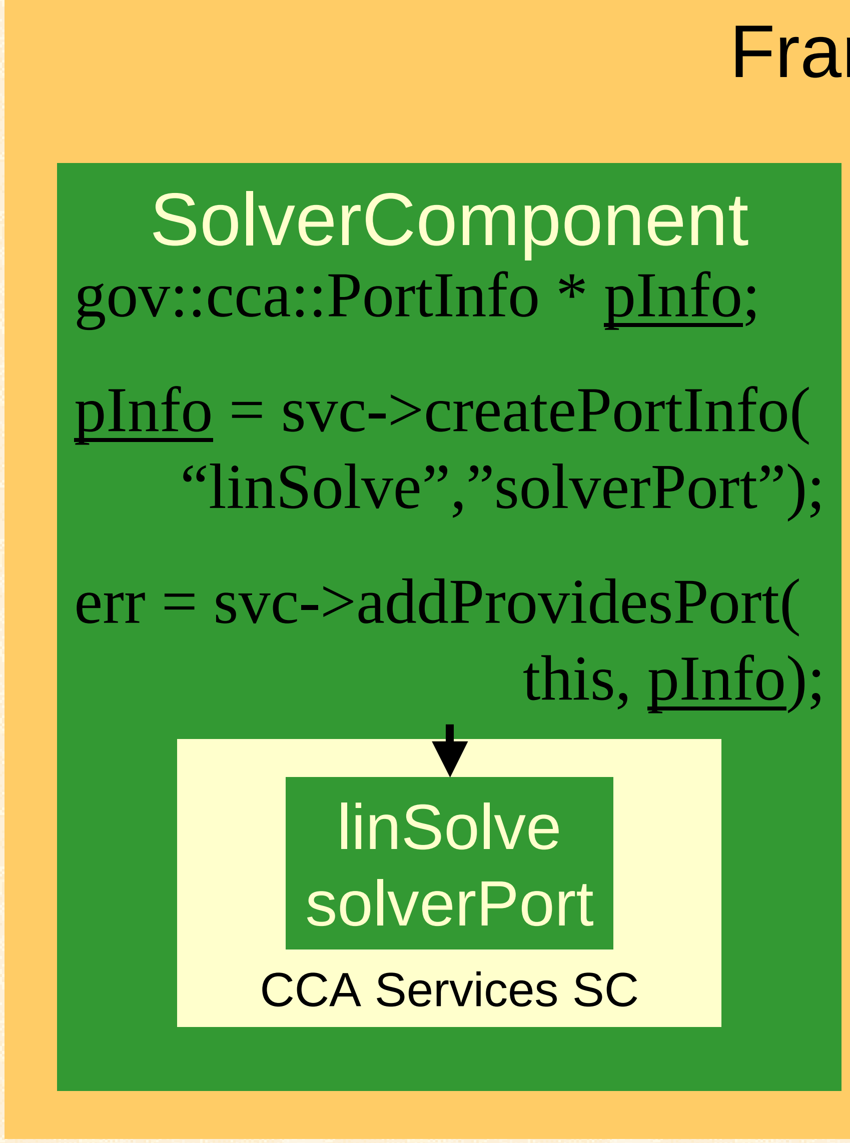
```
create   TaoSolver      TAOSolver
create   MinsurfDriver  MinsurfDriver
…
connect  MinsurfDriver  OptModel      MinsurfModel  OptModel
connect  MinsurfDriver  OptSolver     TAOSolver     OptSolver
…
```

What Makes a CCA Component?



- User instantiates ***SolverComponent***
- Framework calls ***SC***'s setServices method with a CCA Services object – the component's window on the external world

SC Provides a solverPort



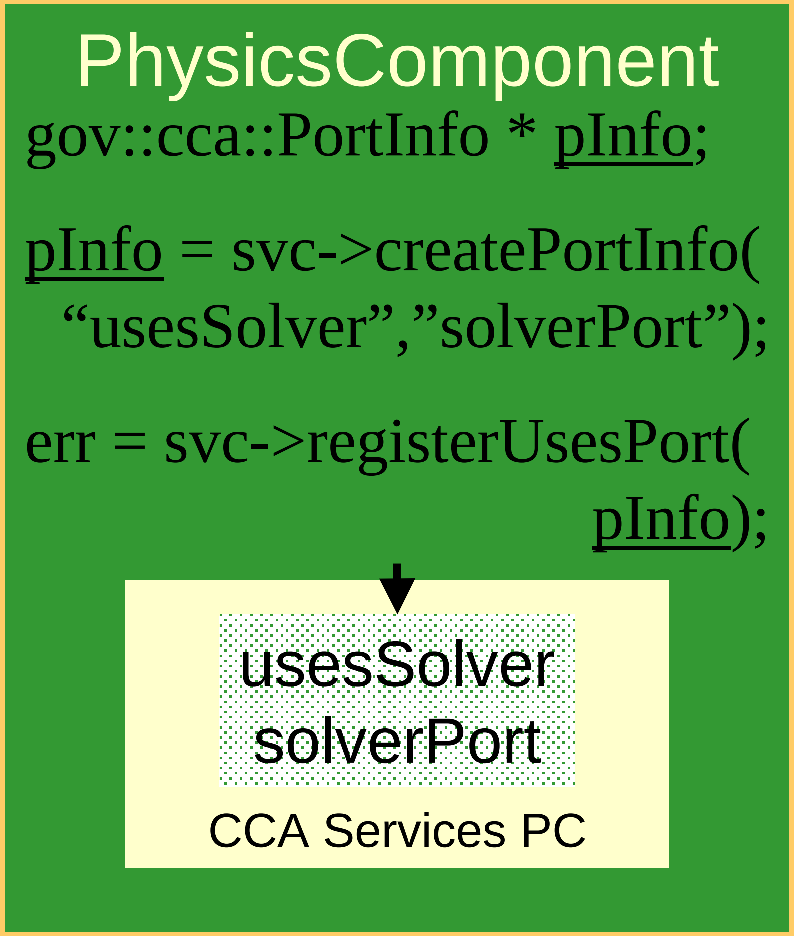
```
gov::cca::PortInfo * pInfo;

pInfo = svc->createPortInfo(
      “linSolve”,”solverPort”);

err = svc->addProvidesPort(
                       this, pInfo);
```

- Within setServices, component declares ports it *provides* and *uses*
- addProvidesPort places port in CCA Services – makes it visible the *framework*
- Other components cannot yet use solverPort!
- setServices completes and control returns to framework to instantiate other components

PC Wants to Use a solverPort

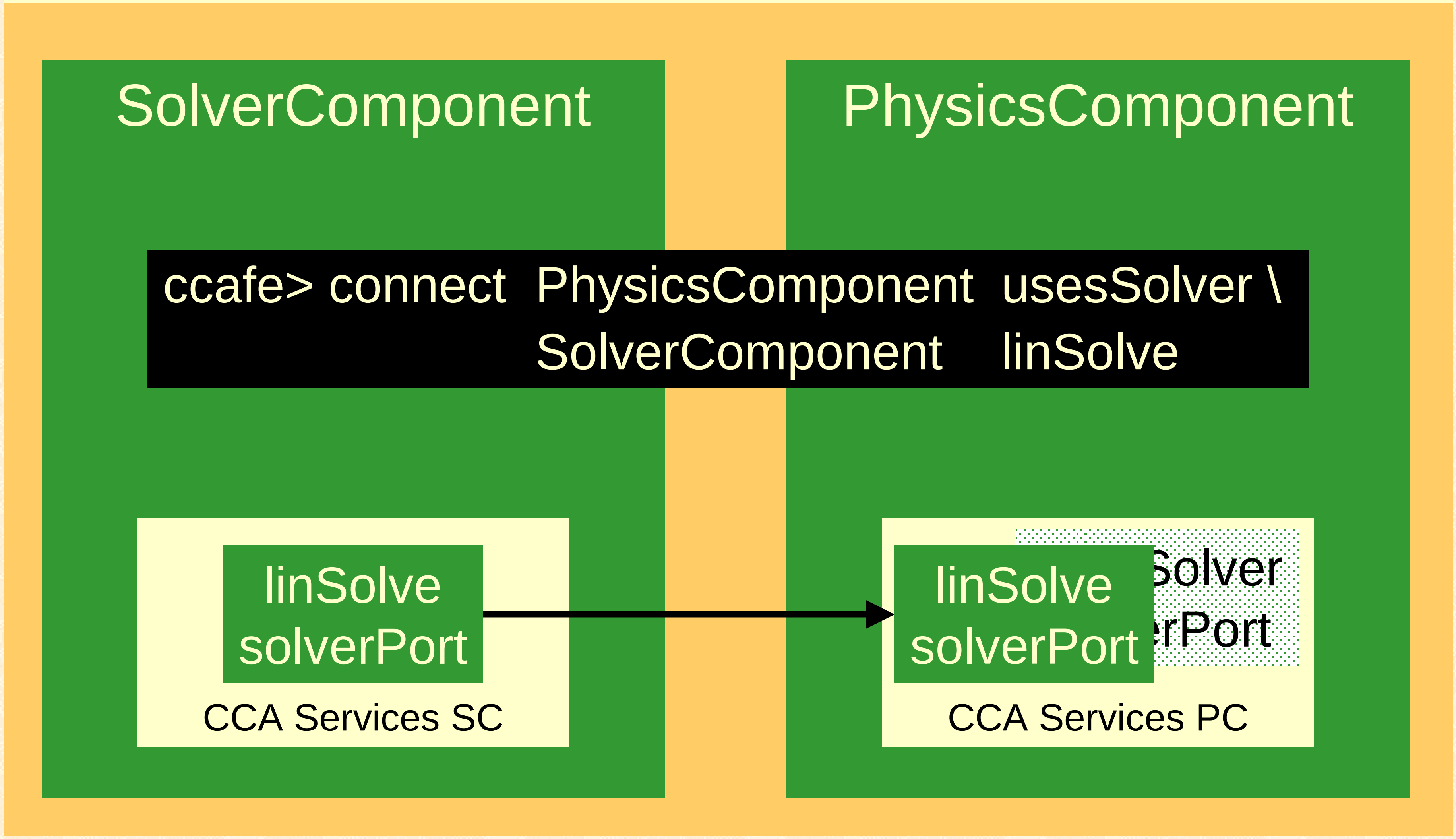
Framework

- User instantiates ***PhysicsComponent***
- Framework calls ***PC***'s setServices
- registerUsesPort informs the *framework* that we want to connect to a component providing a solverPort
- setServices completes, control returns to framework
- PC cannot see ***SolverComponent*** or the solverPort it provides



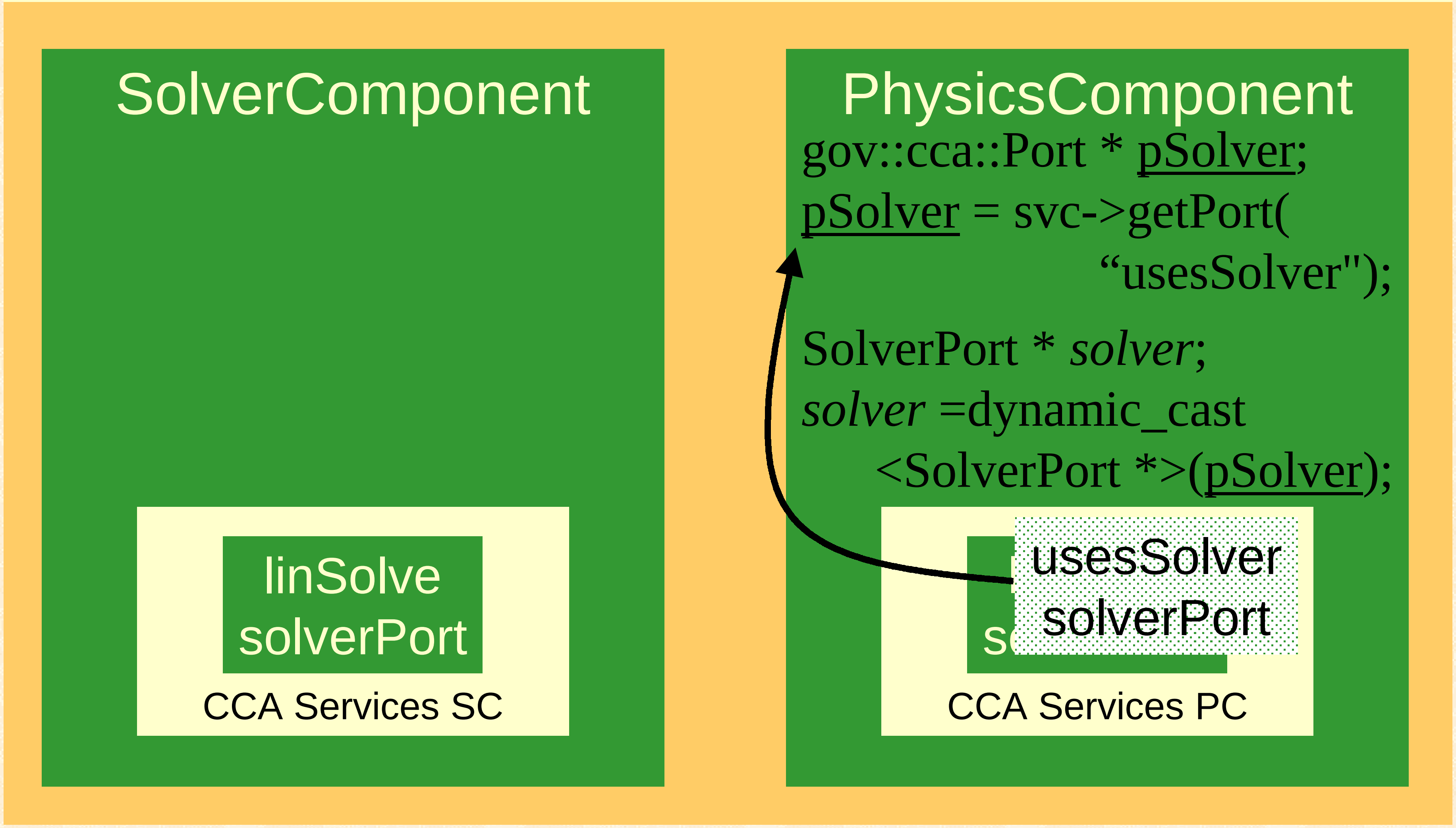
```
gov::cca::PortInfo * pInfo;

pInfo = svc->createPortInfo(
  "usesSolver","solverPort");

err = svc->registerUsesPort(
                          pInfo);
```

User Instructs Framework to Connect solverPort between *User* and *Provider*



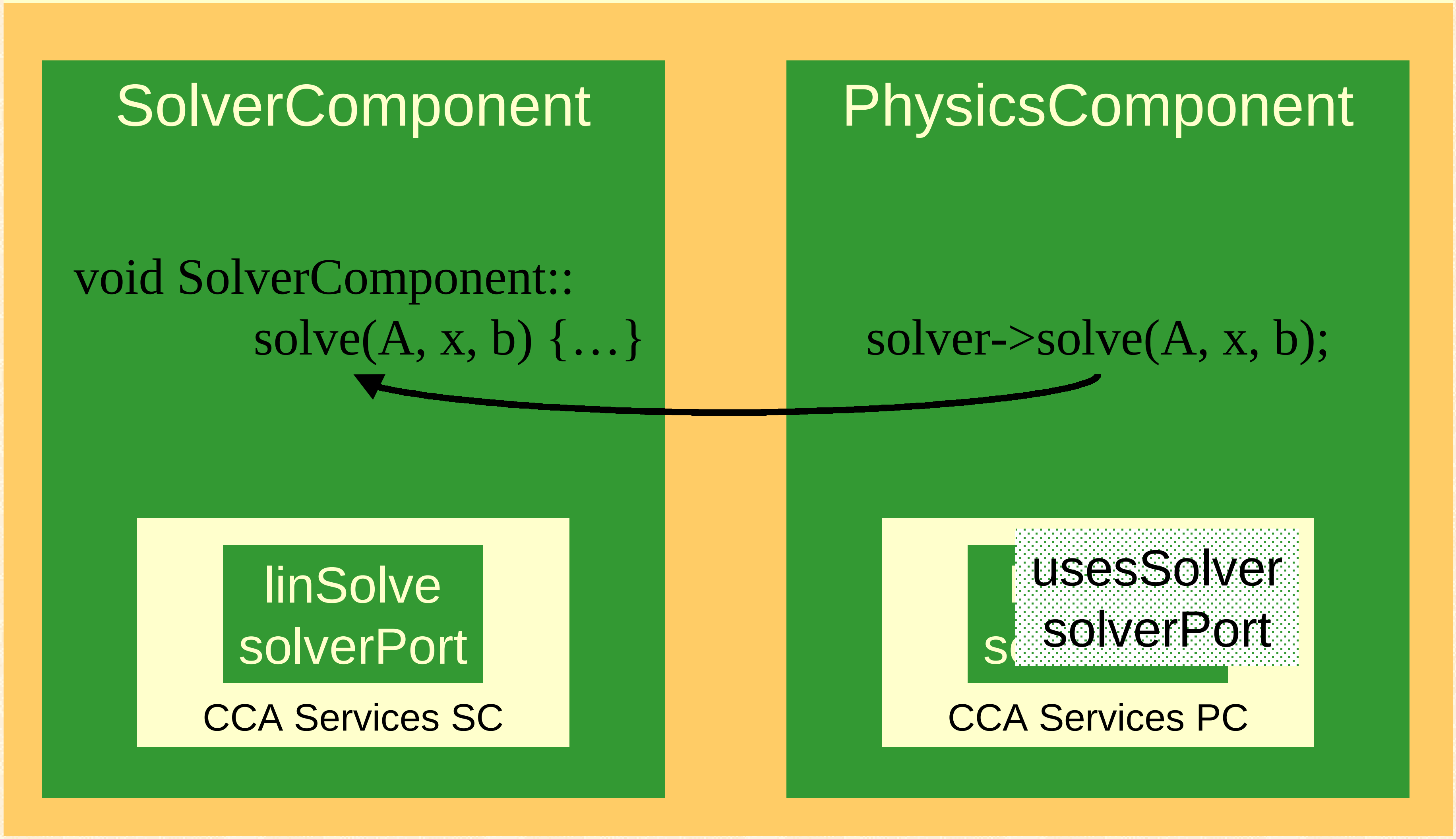
```
ccafe> connect  PhysicsComponent  usesSolver \
                SolverComponent   linSolve
```

PC Gets the Port from its CCA Services



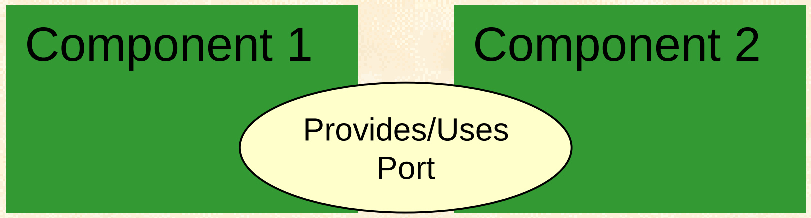
```
gov::cca::Port * pSolver;
pSolver = svc->getPort(
                    “usesSolver");
SolverPort * solver;
solver =dynamic_cast
     <SolverPort *>(pSolver);
```

PC Can Finally Call solve on SC's solverPort

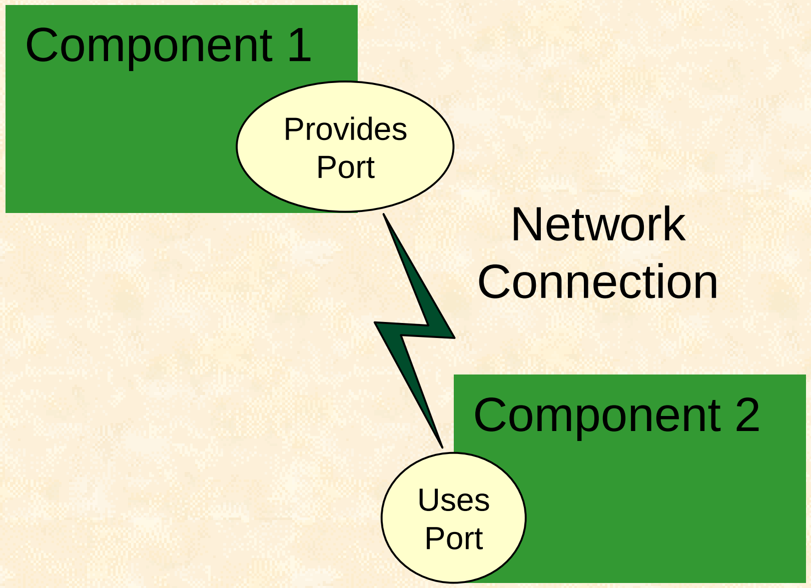


Important Features of CCA Component Model

- Fences between components
  - Components must declare both what they provide and what they use
  - Components cannot interact until ports are connected
  - No mechanism to call anything not part of a port
- Ports preserve high performance direct connection semantics…
- …While also allowing distributed computing



Direct Connection



Prototype CCA Frameworks

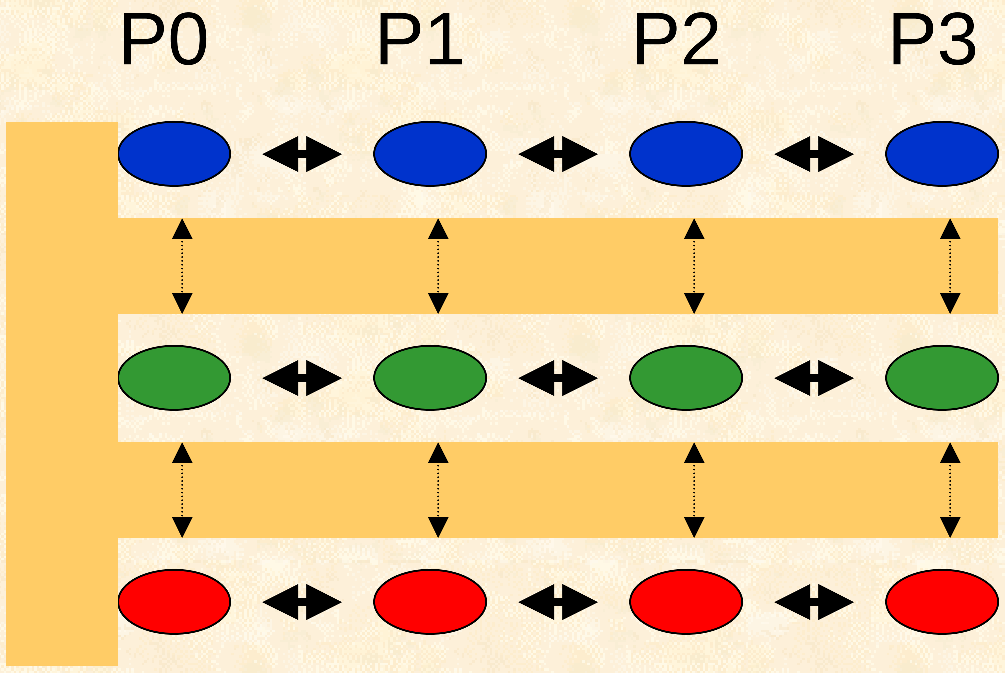
- CCAT, Indiana University, Dennis Gannon
  - Distributed
  - Network connection
- CCAFFEINE, Sandia National Laboratories, Rob Armstrong
  - SPMD/SCMD parallel
  - Direct connection
- SCIRun/Uintah, University of Utah, Steve Parker
  - Parallel, multithreaded
  - Direct connection

CCA Concepts: Direct Connection

- Components loaded into separate *namespaces* in same *address space* (process) from shared libraries
- getPort call returns a pointer to the port's function table
- Invoking a method on a port is equivalent to a C++ virtual function call: lookup function, invoke
- Maintains performance (lookup can be cached)

CCA Concepts: SCMD

- 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)



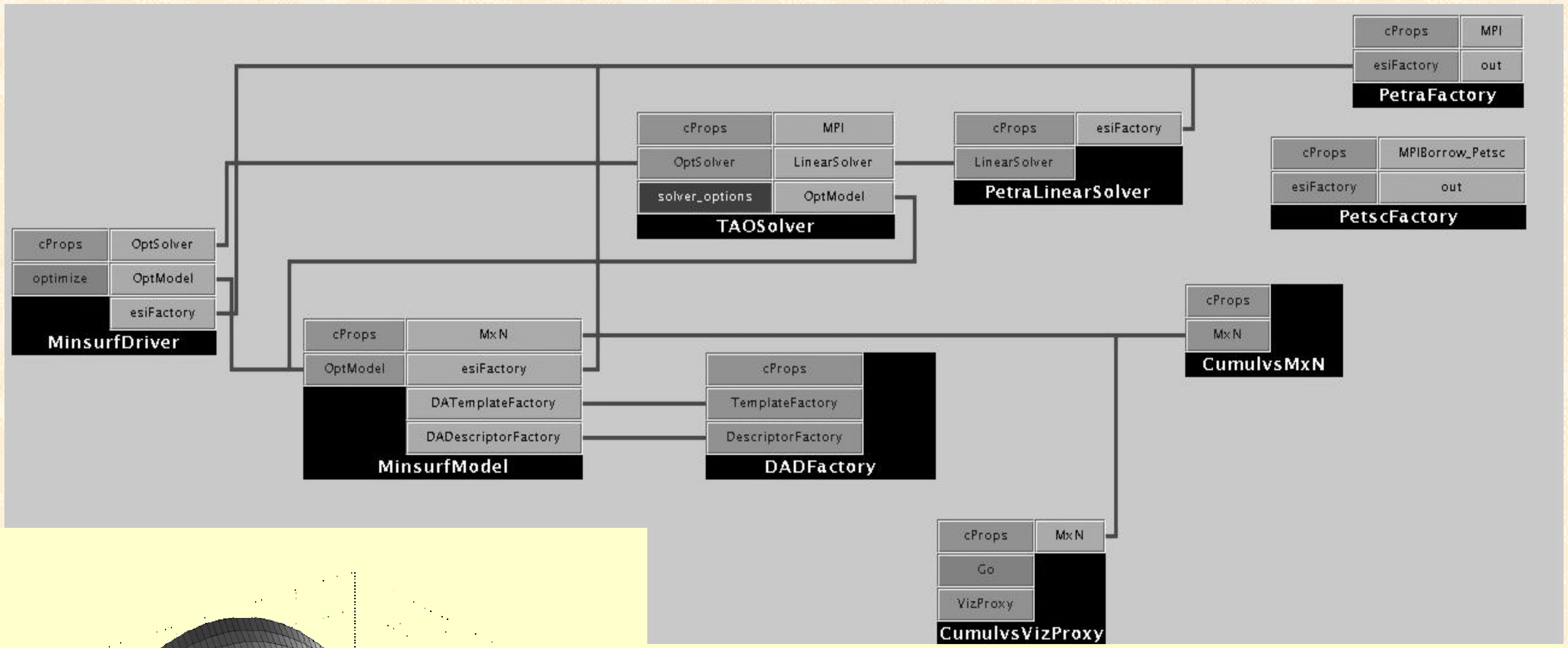
Components: Blue, Green, Red

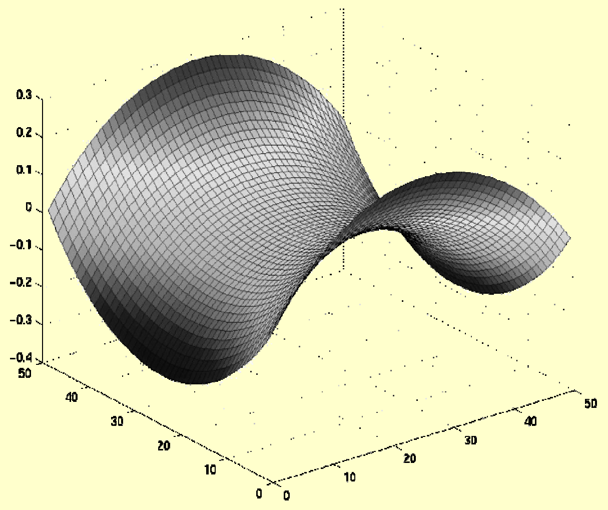
Framework: Beige

Current Status of CCA

- Specification version 0.5
- Working prototype frameworks
- Working multi-component parallel and distributed demonstration applications
- Draft specifications for
  - Basic scientific data objects
  - MxN parallel data redistribution
- SC01 demonstrations
  - four different “direct connect” applications, add’l distributed
  - DC demos: 31 distinct components, up to 17 in any single application, 6 used in more than one application
  - Components leverage and extend parallel software tools including CUMULVS, GrACE, LSODE, MPICH, PAWS, PETSc, PVM, SUMAA3d, TAO, and Trilinos.

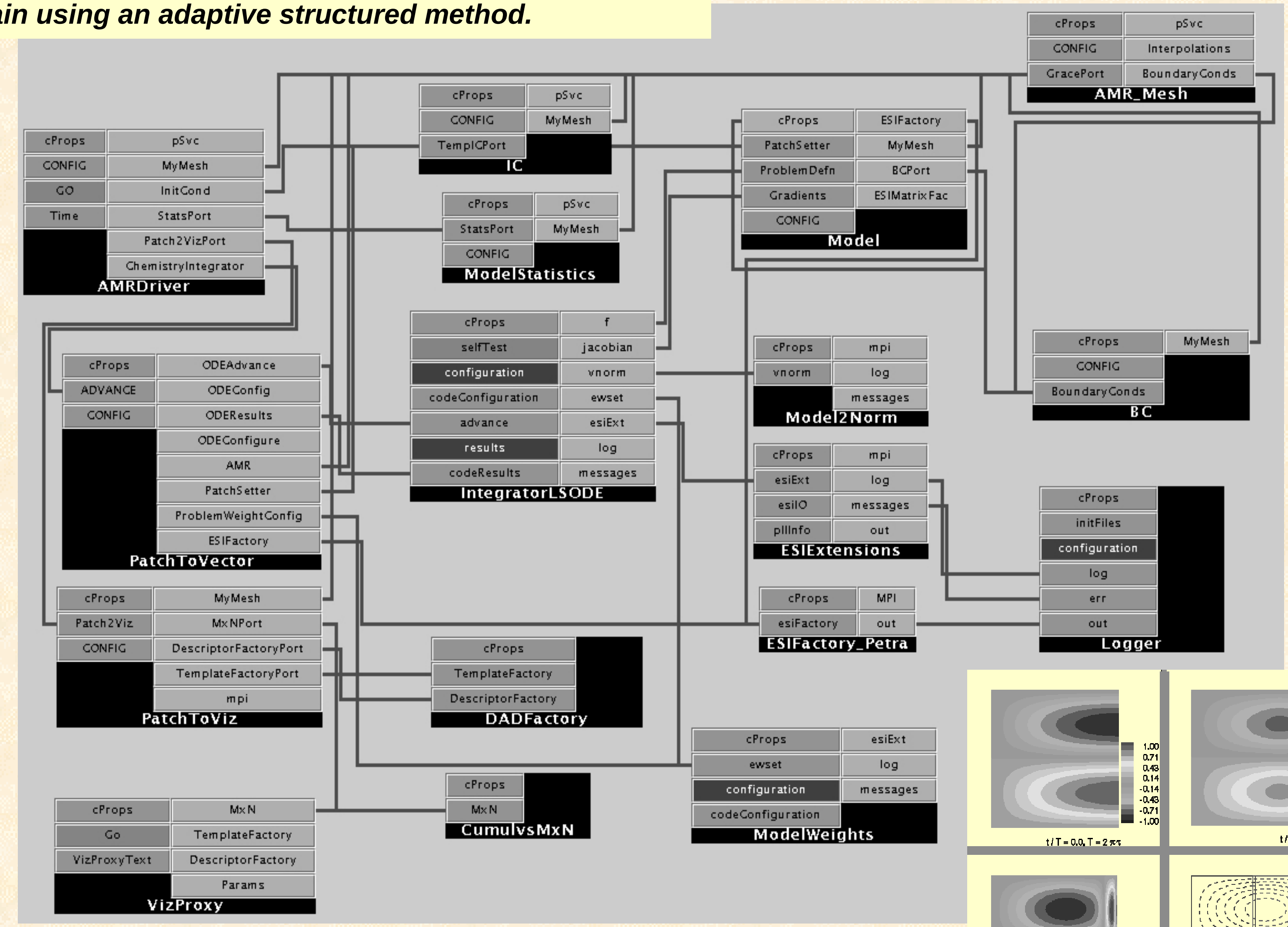
Solution of an unconstrained minimization problem (determining minimal surface area given boundary constraints) using the TAOSolver optimization component

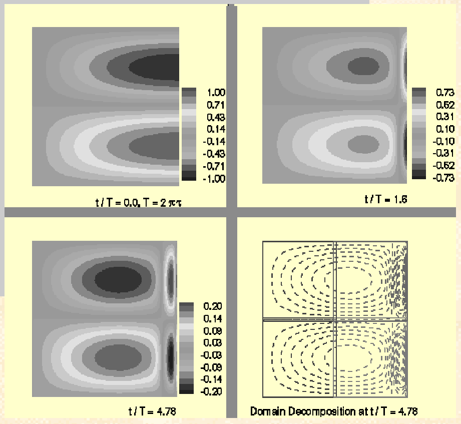




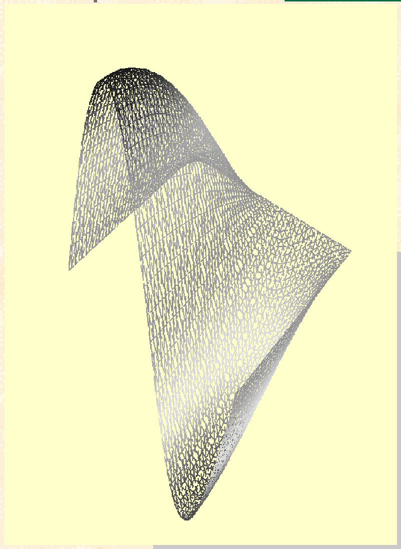
TAOSolver uses linear solver components that incorporate abstract interfaces under development by the Equation Solver Interface (ESI) working group; underlying implementations are provided via the new ESI interfaces to parallel linear solvers within the PETSc and Trilinos libraries. These linear solver components are employed in the other two applications as well.

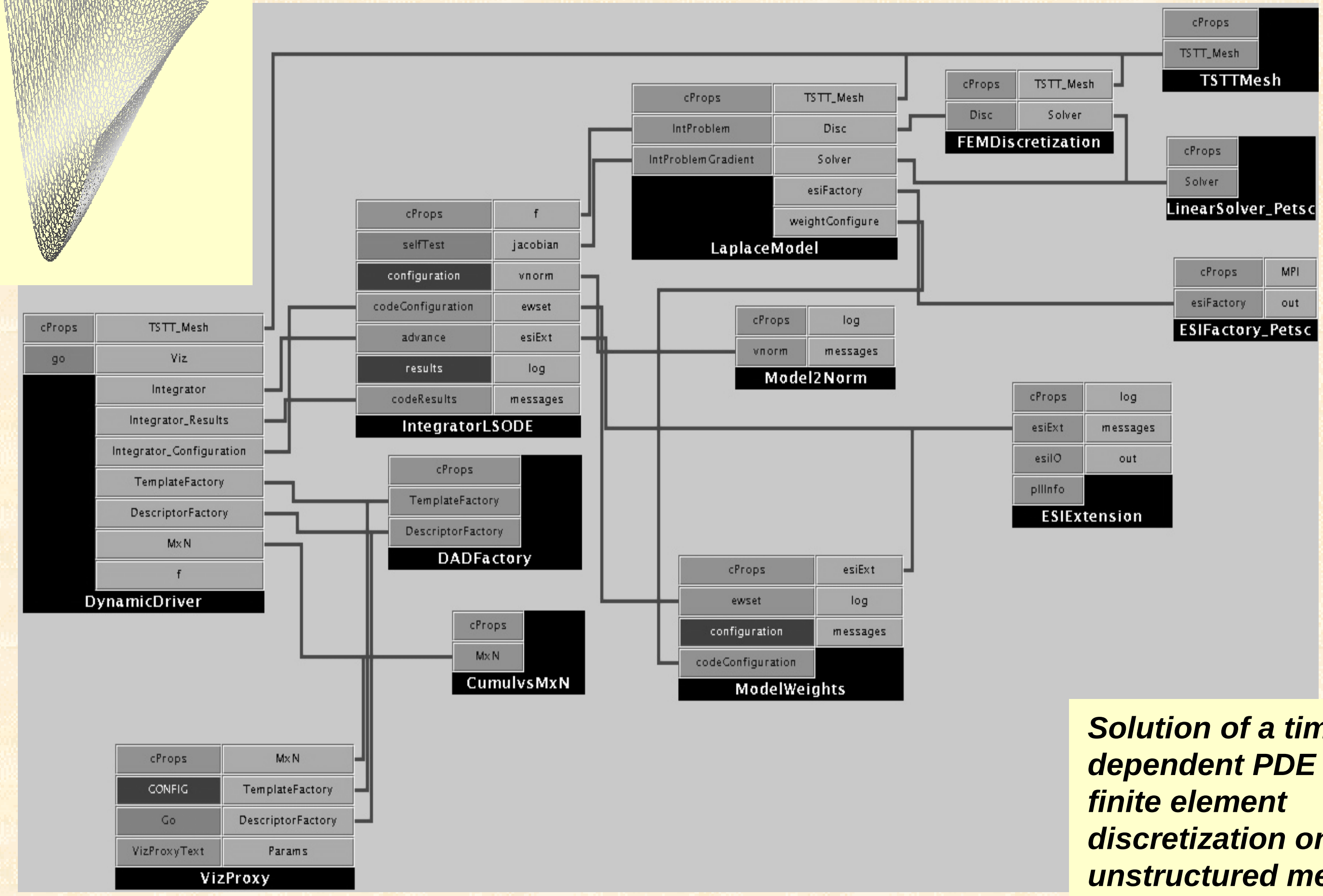
Solution of a two-dimensional heat equation on a square domain using an adaptive structured method.


AMR_Mesh: cProps, pSvc, CONFIG, Interpolations, GracePort, BoundaryConds
IC: cProps, pSvc, CONFIG, MyMesh, TempICPort
AMRDriver: cProps, pSvc, CONFIG, MyMesh, GO, InitCond, Time, StatsPort, Patch2VizPort, ChemistryIntegrator
ModelStatistics: cProps, pSvc, StatsPort, MyMesh, CONFIG
Model: cProps, ESIFactory, PatchSetter, MyMesh, ProblemDefn, BCPort, Gradients, ESIMatrixFac, CONFIG
IntegratorLSODE: cProps, f, selfTest, jacobian, configuration, vnorm, codeConfiguration, ewset, advance, esiExt, results, log, codeResults, messages
Model2Norm: cProps, mpi, vnorm, log, messages
BC: cProps, MyMesh, CONFIG, BoundaryConds
PatchToVector: cProps, ODEAdvance, ADVANCE, ODEConfig, CONFIG, ODEResults, ODEConfigure, AMR, PatchSetter, ProblemWeightConfig, ESIFactory
ESIExtensions: cProps, mpi, esiExt, log, esiIO, messages, pllInfo, out
Logger: cProps, initFiles, configuration, log, err, out
PatchToViz: cProps, MyMesh, Patch2Viz, MxNPort, CONFIG, DescriptorFactoryPort, TemplateFactoryPort, mpi
ESIFactory_Petra: cProps, MPI, esiFactory, out
DADFactory: cProps, TemplateFactory, DescriptorFactory
CumulvsMxN: cProps, MxN
ModelWeights: cProps, esiExt, ewset, log, configuration, messages, codeConfiguration
VizProxy: cProps, MxN, Go, TemplateFactory, VizProxyText, DescriptorFactory, Params



IntegratorLSODE provides a second-order implicit time integrator, and ***Model*** provides a discretization. The remaining components are essentially utilities that construct the global ODE system or adaptors that convert the patch-based data structures of the mesh to the globally distributed array structure used for runtime visualization.





Solution of a time-dependent PDE using a finite element discretization on an unstructured mesh

IntegratorLSODE provides a second-order implicit time integrator, and ***FEMDiscretization*** provides a discretization.

This application (and the other two applications as well) use the ***DADFactory*** component to describe the parallel data layout so that the ***CumulsMxN*** data redistribution component can then collate the data from a multi-processor run to a single processor for runtime visualization.

The CCA Forum

- The Common Component Architecture is merely a *specification* for what is required to be a CCA component and a CCA framework
- Specification determined by the CCA Forum
  - Open membership
  - Face-to-face meetings quarterly
  - Voting privileges based on attendance at recent meetings
- CCA Forum has been meeting regularly since January 1998
  - Next meeting: Bishop's Lodge, Santa Fe, NM
    - 9 January 2002 – working groups & tutorial,
    - 10-11 January 2002 – general Forum meeting

The Center for Component Technology for Terascale Simulation Software (CCTTSS)

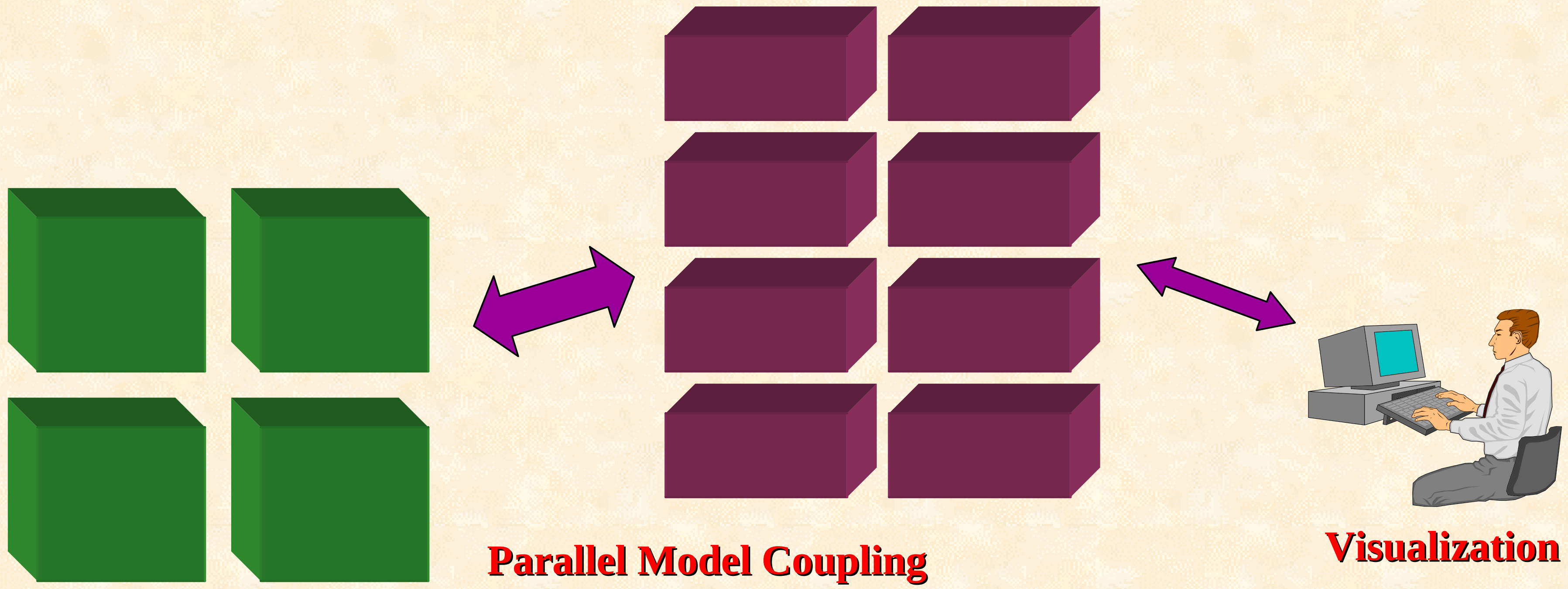
- A SciDAC Integrated Software Infrastructure Center (ISIC)
- *Mission:* Advance research in high-performance component technology and bring CCA from a conceptual prototype to a full-fledged production-quality environment
- *Participants:*
  - Argonne, Livermore, Los Alamos, Oak Ridge, Pacific Northwest, and Sandia National Laboratories;
  - Indiana University, and University of Utah

CCTTSS's R&D Agenda

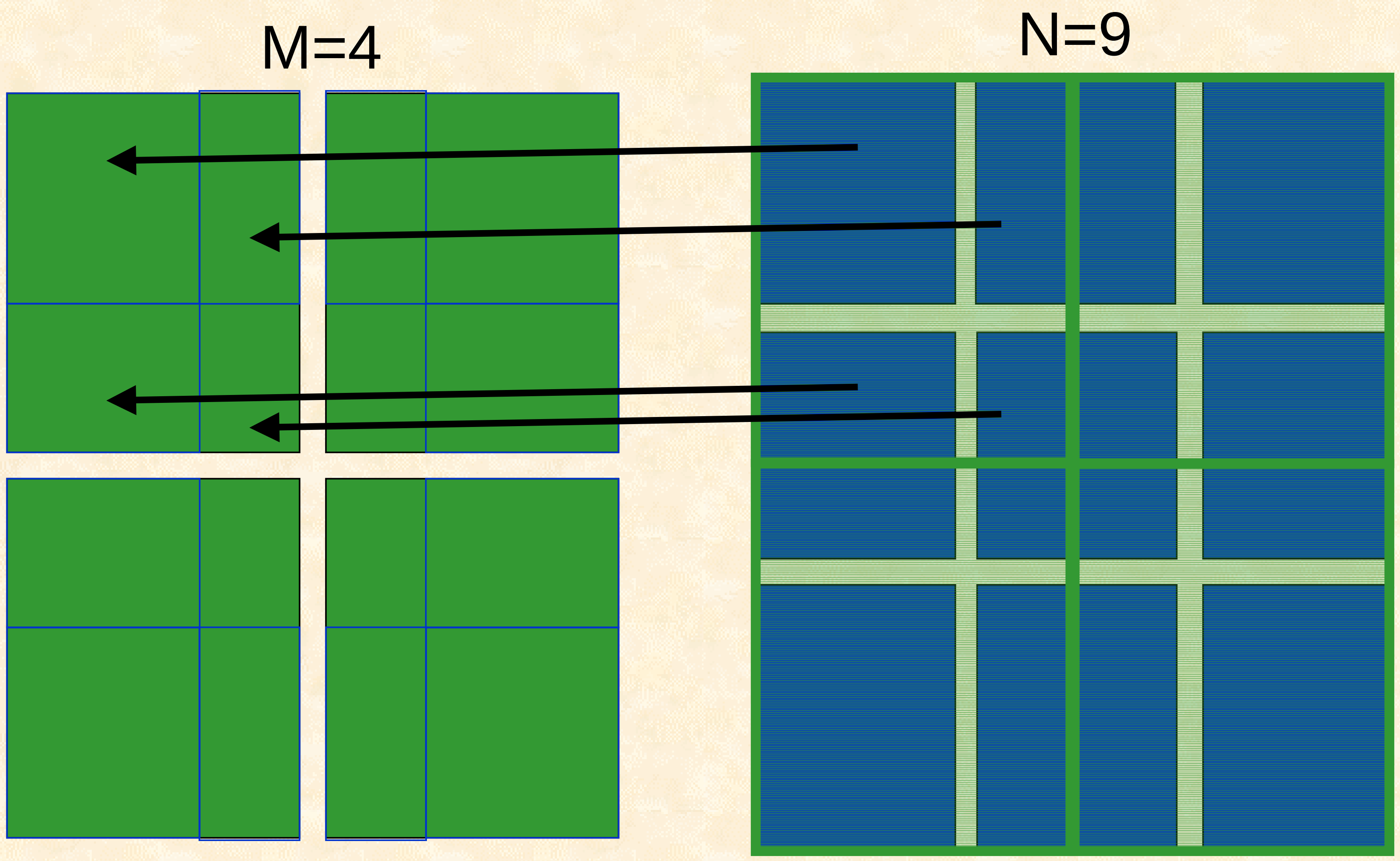
- Frameworks
  - Integration of prototype frameworks
  - Language interoperability tools
  - Component deployment
- Scientific Components
  - Abstract interfaces and component implementations
  - Scientific data; Linear, nonlinear, and optimization solvers; Steering and visualization; Multi-threading and load redistribution; Fault tolerance
  - Quality of service research
- MxN Parallel Data Redistribution
- Applications Integration
  - Chemistry and Climate work within CCTTSS
  - Close collaboration with other SciDAC infrastructure projects (especially TOPS, TSTT)
  - Strong liaison with adopting groups

CCA Concepts: MxN Parallel Data Redistribution

- Share Data Among Coupled Parallel Models
  - Disparate Parallel Topologies (M processes vs. N)
  - e.g. Ocean & Atmosphere, Solver & Optimizer…
  - e.g. Visualization (Mx1, increasingly, MxN)



MxN: The Problem



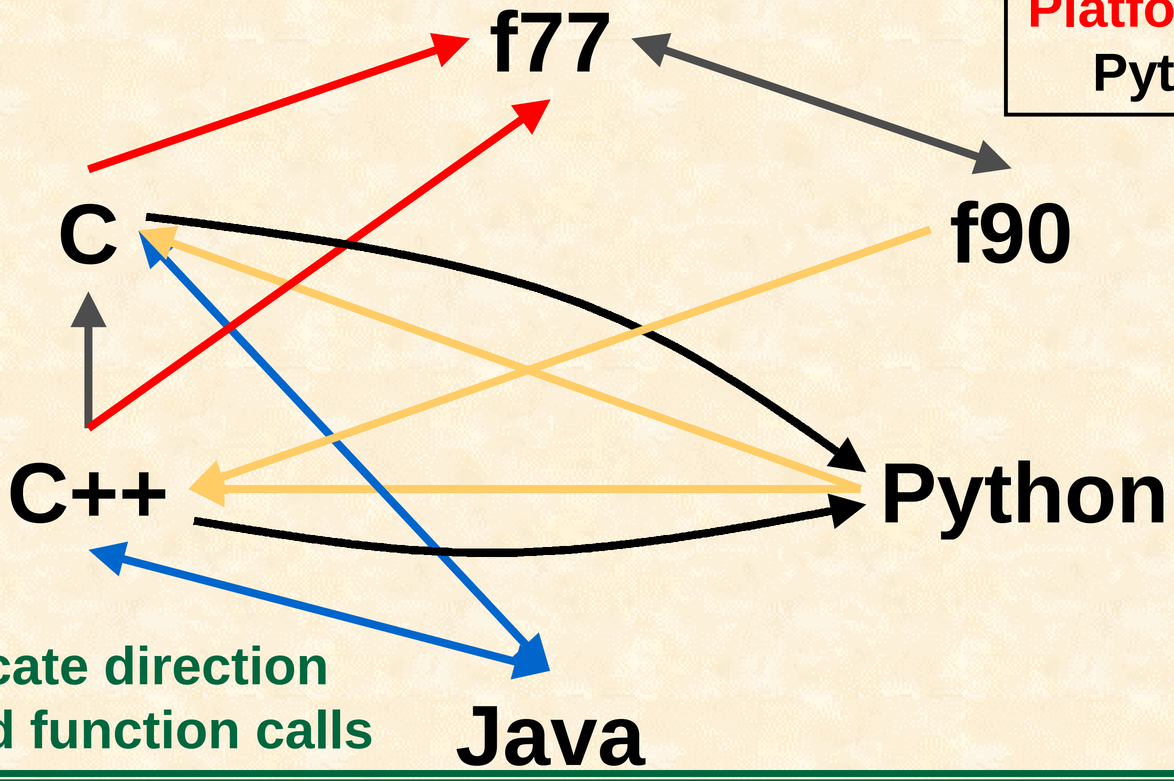
MxN Research Activities

- Distributed data descriptors
  - Uniform langauge for expressing distribution
  - Draft specification for dense multi-dimensional arrays
- MxN port
  - Draft specification completed, implemented
  - Minimal intrusion to “instrument” component, third-party control possible
  - Multiple data fields, exchange in either direction
  - One-shot or repeated transfer
- Future
  - Framework-based solutions
  - Higher-level coupling, account for units of data, spatial and termporal interpolation, etc.

CCA Concepts: Language Interoperability

- Existing language interoperability approaches are "point-to-point" solutions

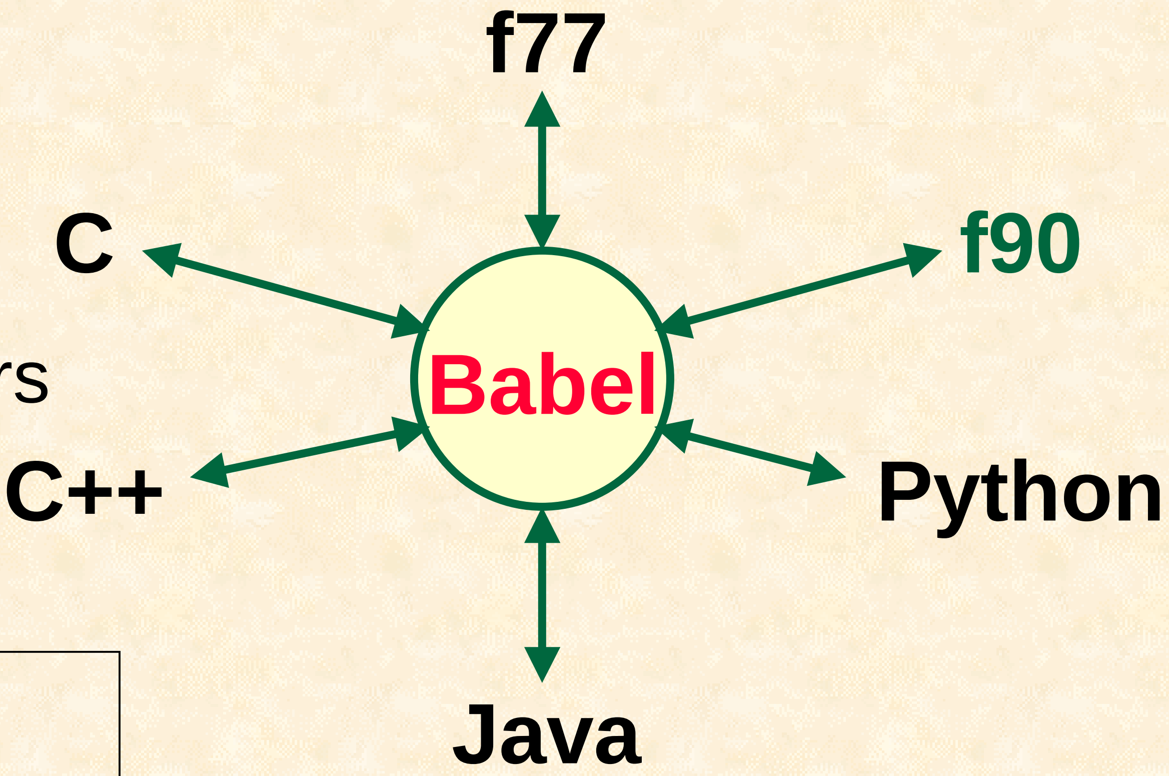
Java JNI
Native
SWIG/SILOON
Platform Dependent
Python Library



Arrows indicate direction of supported function calls

Language Interoperability w/ Babel

- Babel provides a unified approach in which all languages are considered peers

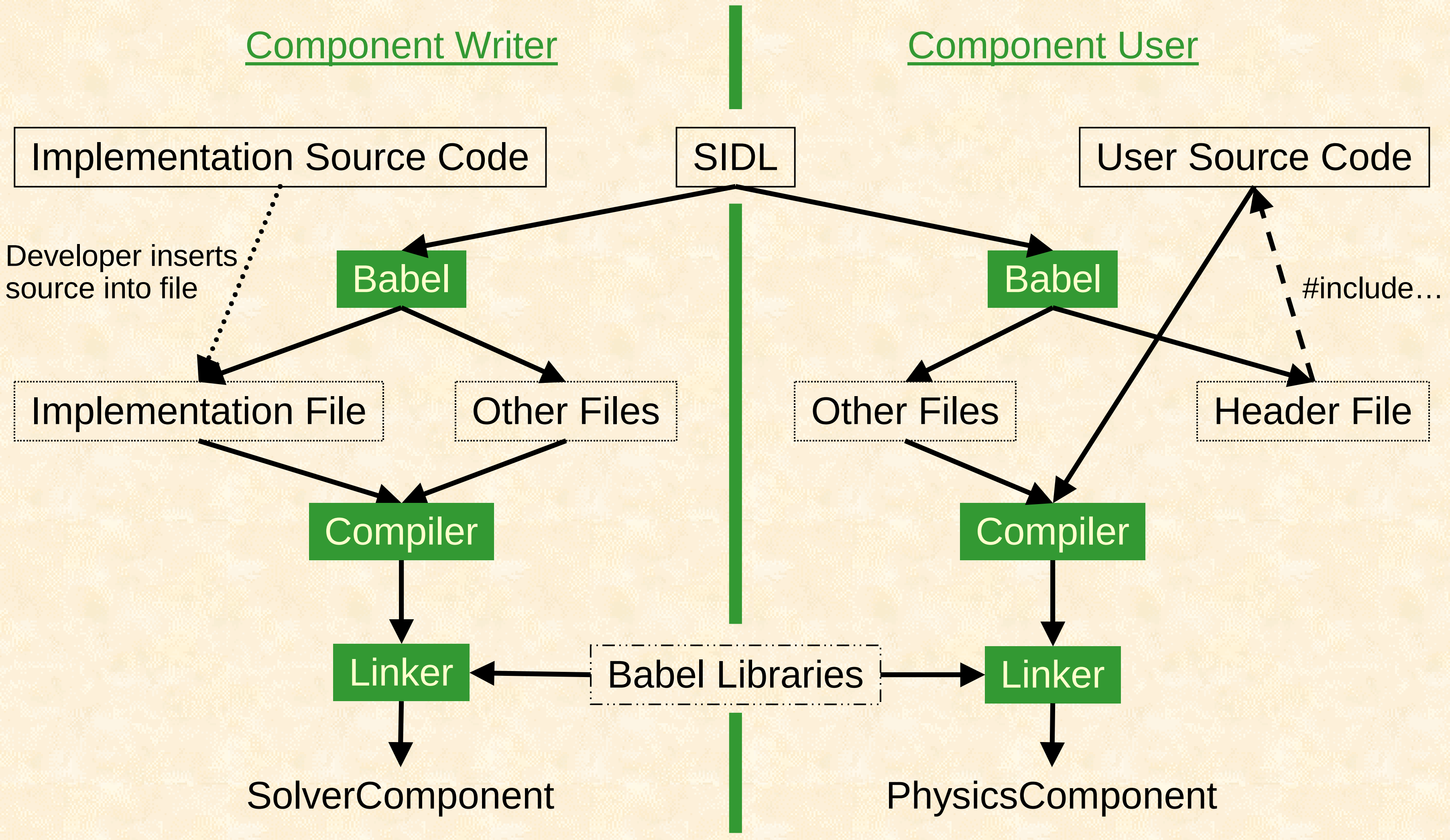


solverPort.sidl

```
version solverPort 1.0;
package solverPort {
   class Solver {
      int solve( in   array<double,2>  A,
                 out  array<double,1>  x,
                 in   array<double, 1> b);
   }
}
```

Somewhat similar to the CORBA approach in the business domain

Using Babel


Component Writer
Component User
Implementation Source Code
SIDL
User Source Code
Developer inserts source into file
Babel
Babel
#include…
Implementation File
Other Files
Other Files
Header File
Compiler
Compiler
Linker
Babel Libraries
Linker
SolverComponent
PhysicsComponent

Summary

- Components promote modularity & reuse, allow developers to focus on their areas of expertise
- CCA is a component model targeted specifically to the needs of high-performance computing – supports *direct connection* of components (as well as distributed computing)
- Components exchange *ports* following a *uses-provides* design pattern.
- Specification intentionally places minimal requirements on components
  - 1 additional method to become a component
  - 2 calls to declare a used or provided port
  - 2 calls required to get a port for use
- Useful prototypes exist, applications being developed
- CCTTSS mission to bring CCA from prototype to production-quality system

Information Pointers

- http://www.cca-forum.org
- http://www.cca-forum.org/ccttss
- Mailing list: cca-forum@z.ca.sandia.gov (majordomo)
- CCTTSS contacts:

Lead PI	Rob Armstrong	SNL	rob@sandia.gov
Frameworks	Scott Kohn	LLNL	skohn@llnl.gov
Scientific Data Components	Lois McInnes	ANL	mcinnes@anl.gov
MxN Data Redistribution	Jim Kohl	ORNL	kohlja@ornl.gov
Applications Integration	David Bernholdt	ORNL	bernholdtde@ornl.gov
PNNL PI	Jarek Nieplocha	PNNL	j_nieplocha@pnl.gov