The Common Component Architecture: Building Frameworks for Computational Science

David E. Bernholdt

Oak Ridge National Laboratory

bernholdtde@ornl.gov

http://www.cca-forum.org

Acknowledgements

- The Common Component Architecture is a true community effort, and therefore the represents the collective efforts of many people

- The CCA Forum is the standards body/user group

- Two large DOE/SC-sponsored projects have led the development
  - 2001-2006: Center for Component Technology for Terascale Simulation Software (CCTTSS)
  - 2006-2011: Center for Technology for Advanced Scientific Component Software (TASCS)

- Many other projects with a wide range of sponsors have also made significant contributions
  - DoD, DOE/NNSA, EU, NASA, NIH, probably others…

- Users from a wide range of scientific disciplines

What are Components?

- No universally accepted definition in computer science research, but key features include…

- A unit of software development/deployment/reuse
    - i.e. has interesting functionality
    - Ideally, functionality someone else might be able to (re)use
    - Can be developed independently of other components

- Interacts with the outside world only through well-defined interfaces
    - Implementation is opaque to the outside world

- Can be composed with other components
    - "Plug and play" model to build applications
    - Composition based on interfaces

What is a Component Architecture?

- A set of standards that allows:
  - Multiple groups to write units of software (components)…
  - And have confidence that their components will work with other components written in the same architecture

- These standards define…
  - The rights and responsibilities of a component
  - How components express their interfaces
  - The environment in which components are composed to form an application and executed (framework)
  - The rights and responsibilities of the framework

What is the CCA?

- Component-based software engineering has been developed primarily in other areas of computing
  - Especially business and internet
  - Examples: CORBA Component Model, COM, Enterprise JavaBeans

- Many of the needs are similar to those in HPC scientific computing

- But scientific computing imposes special requirements not common elsewhere

- CCA is a component environment specially designed to meet the needs of HPC scientific computing

Special Needs of Scientific HPC

- ## Support for legacy software
  - How much change required for component environment?

- ## Performance is important
  - What overheads are imposed by the component environment?

- ## Both parallel and distributed computing are important
  - What approaches does the component model support?
  - What constraints are imposed?
  - What are the performance costs?

- ## Support for languages, data types, and platforms
  - Fortran?
  - Complex numbers? Arrays? (as first-class objects)
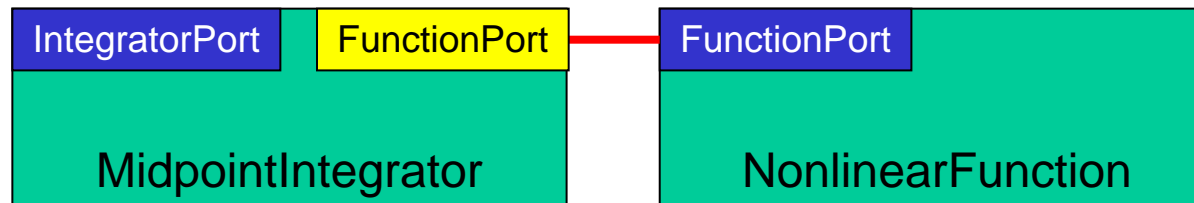 - Is it available on my parallel computer?

Computational Science Frameworks

- *This is my personal working definition*

- Frameworks provide an environment for the development and execution of simulations in a given scientific domain or class of problem

- Provide utility routines specialized to the domain
  - Simplifies the task of developing simulations in that domain

- Usually define a simulation workflow
  - Definition may be implicit in framework structure
  - May provide some flexibility, but not complete freedom

# Relationship between Components and Science Frameworks

- Component environments are much more basic than computational science frameworks
  - They provide a way to hook pieces of code together
  - They don't (of themselves) provide anything domain-specific

- Component environments can be used as the basis for the construction of computational science frameworks
  - A rich component "ecosystem" is likely to have many relevant things already available in component form

- This approach has important benefits in flexibility and openness

- Even without actually using a component environment, framework design and development can benefit from component concepts

- *Caveat:* the component world (including CCA) also uses the term framework
  - More about CCA frameworks later

CCA Concepts: Components

| IntegratorPort | FunctionPort | | FunctionPort |
| MidpointIntegrator | | | NonlinearFunction |

- A component encapsulates some computational functionality

- Components provide/use one or more interfaces
  - A component with no interfaces is formally okay, but isn't very interesting or useful

- In SIDL, a component is a `class` that `implements` (inherits from) *gov.cca.Component*
  - This means it must implement the `setServices` method to tell framework what ports this component will provide and use
  - *gov.cca.Component* is defined in the CCA specification

CCA Concepts: Ports

| IntegratorPort	FunctionPort	—	FunctionPort
MidpointIntegrator			NonlinearFunction
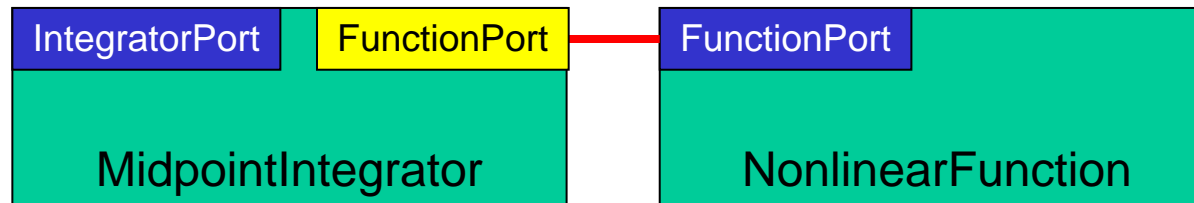
- Components interact through well-defined interfaces, or *ports*
  - A port expresses some computational functionality
  - In Fortran, a port is a bunch of subroutines or a module
  - In OO languages, a port is an abstract class or interface

- Ports and connections between them are a procedural (caller/callee) relationship, not dataflow!
  - e.g., *FunctionPort* could contain a method like `evaluate(in Arg, out Result)` with data flowing both ways

CCA Concepts: *Provides* and *Uses* Ports

IntegratorPort	FunctionPort		FunctionPort
MidpointIntegrator			NonlinearFunction
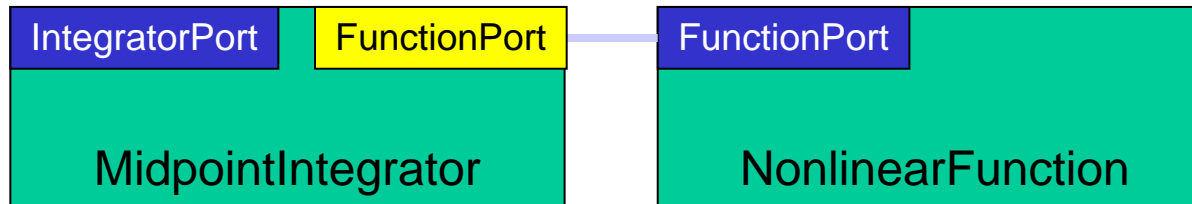
- Components may *provide* ports – implement the class or subroutines of the port ("Provides" Port)
  - *Providing* a port implies certain inheritance relationships between the component and the abstract definition of the interface
  - A component can *provide* multiple ports
    - Different "views" of the same functionality, or
    - Related pieces of functionality

- Components may *use* ports – call methods or subroutines in the port ("Uses" Port)
  - *Use* of ports is just like calling a method normally except for a little additional work due to the "componentness"
  - No inheritance relationship implied between caller and callee
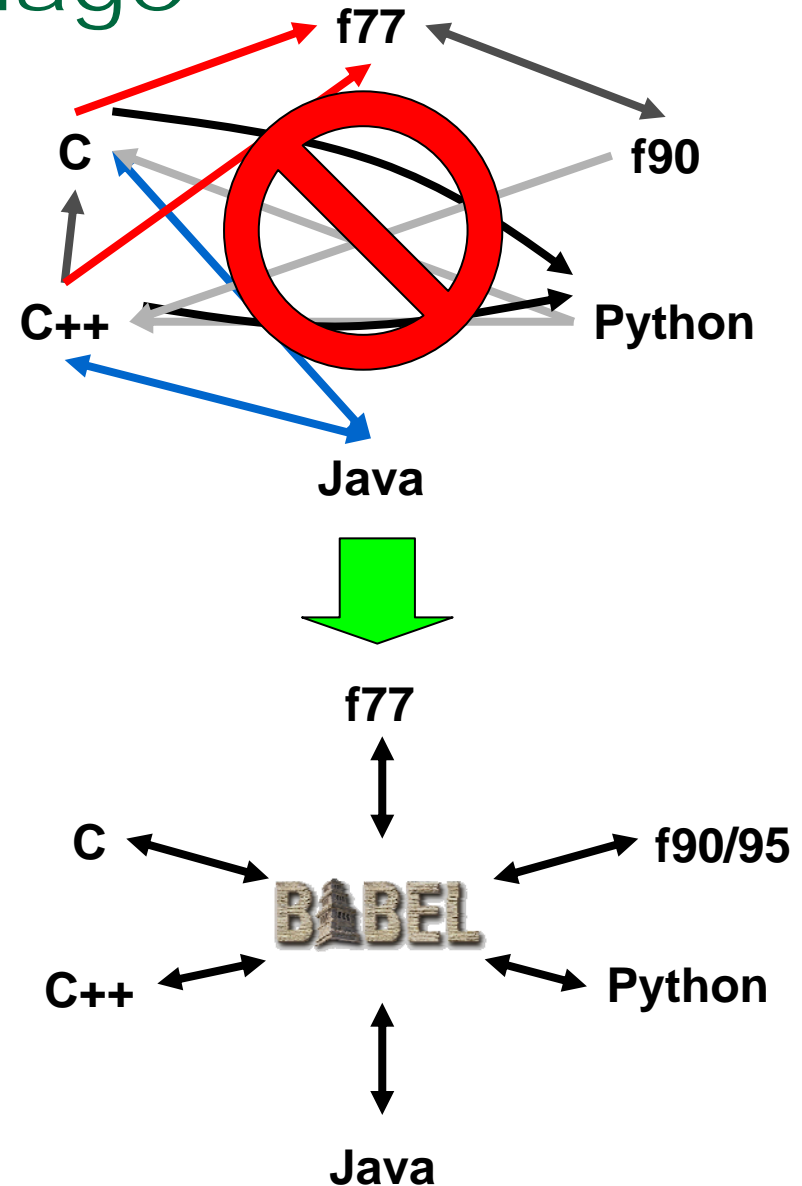 - A component can *use* multiple ports
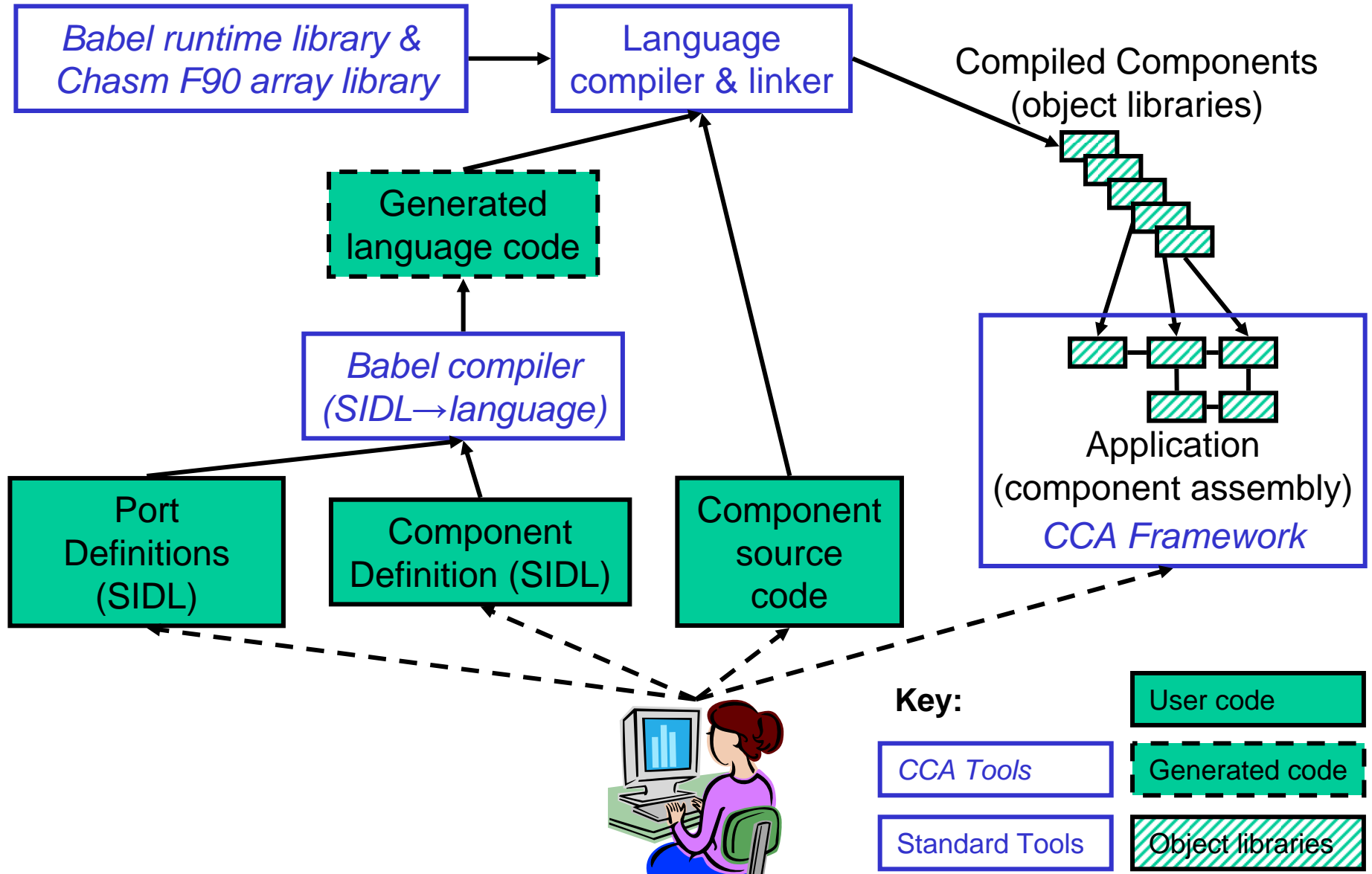
CCA Concepts: Frameworks
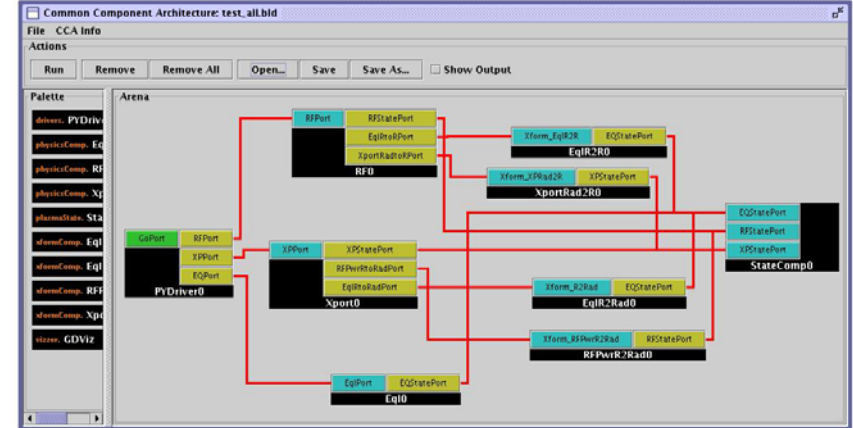
- The framework provides the means to "hold" components and compose them into applications

- Frameworks allow connection of ports without exposing component implementation details

- Frameworks provide a small set of standard services to components
  - Framework services are CCA ports, just like on components
  - Additional (non-standard) services can also be offered
  - Components can register ports as services using the *ServiceProvider* port

- *Currently:* specific frameworks are specialized for specific computing models (parallel, distributed, etc.)

- *Future:* better integration and interoperability of frameworks

CCA Concepts: Language Interoperability

- Scientific software is increasingly diverse in use of programming languages

- In a component environment, users should not care what language a component is implemented in

- "Point-to-point" solutions to language interoperability are not suitable for a component environment

- The Babel language interoperability tool provides a common solution for all supported languages

- Scientific Interface Definition Language provides language-neutral way of expressing interfaces

Coding in a CCA Environment

Babel runtime library & Chasm F90 array library → Language compiler & linker → Compiled Components (object libraries)

Generated language code

Babel compiler (SIDL→language)

Port Definitions (SIDL)

Component Definition (SIDL)

Component source code

Application (component assembly)

CCA Framework

Key:

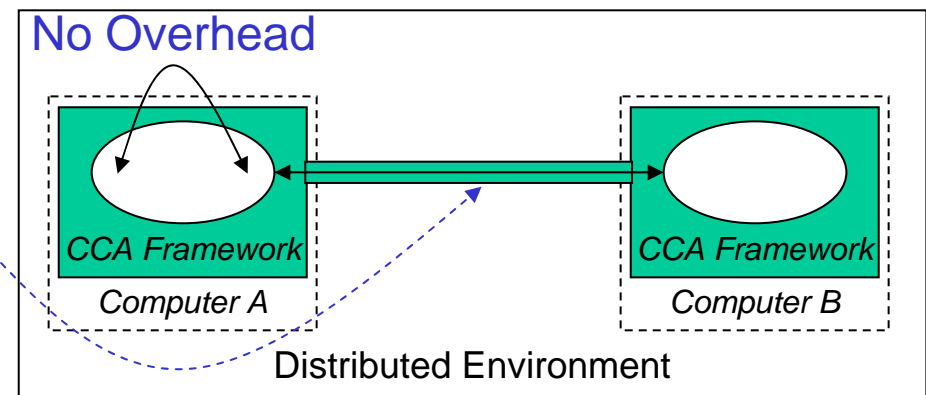|	
	User code
CCA Tools	Generated code
Standard Tools	Object libraries

Performance, the Big Picture

Direct-Connect, Parallel

- No CCA overhead on…
  - calls within component
  - parallel communications across components

- Small overheads on invocations on ports
  - Virtual function call (CCAness)
  - Language Interoperability (some data types)
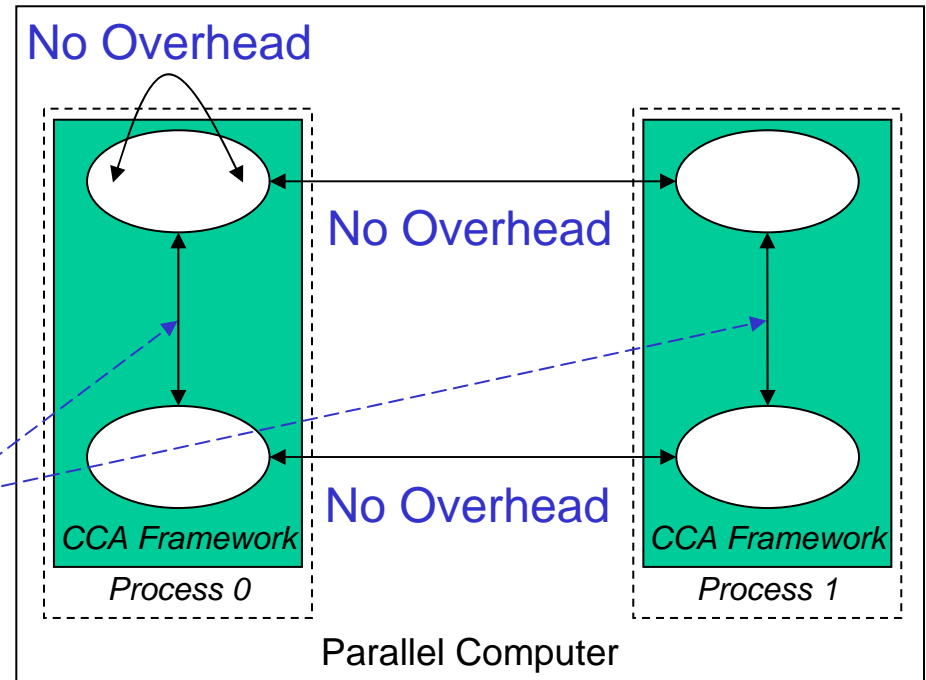
Distributed

- No CCA overhead on calls within component

- Overheads on invocations on ports
  - Language interoperability (some data types)
  - Framework
  - (Wide area) network

No Overhead

No Overhead

No Overhead

CCA Framework

Process 0

CCA Framework

Process 1

Parallel Computer

No Overhead

CCA Framework

Computer A

CCA Framework

Computer B

Distributed Environment

CCA Supports Parallelism -- by "Staying Out of the Way" of it

- 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)*
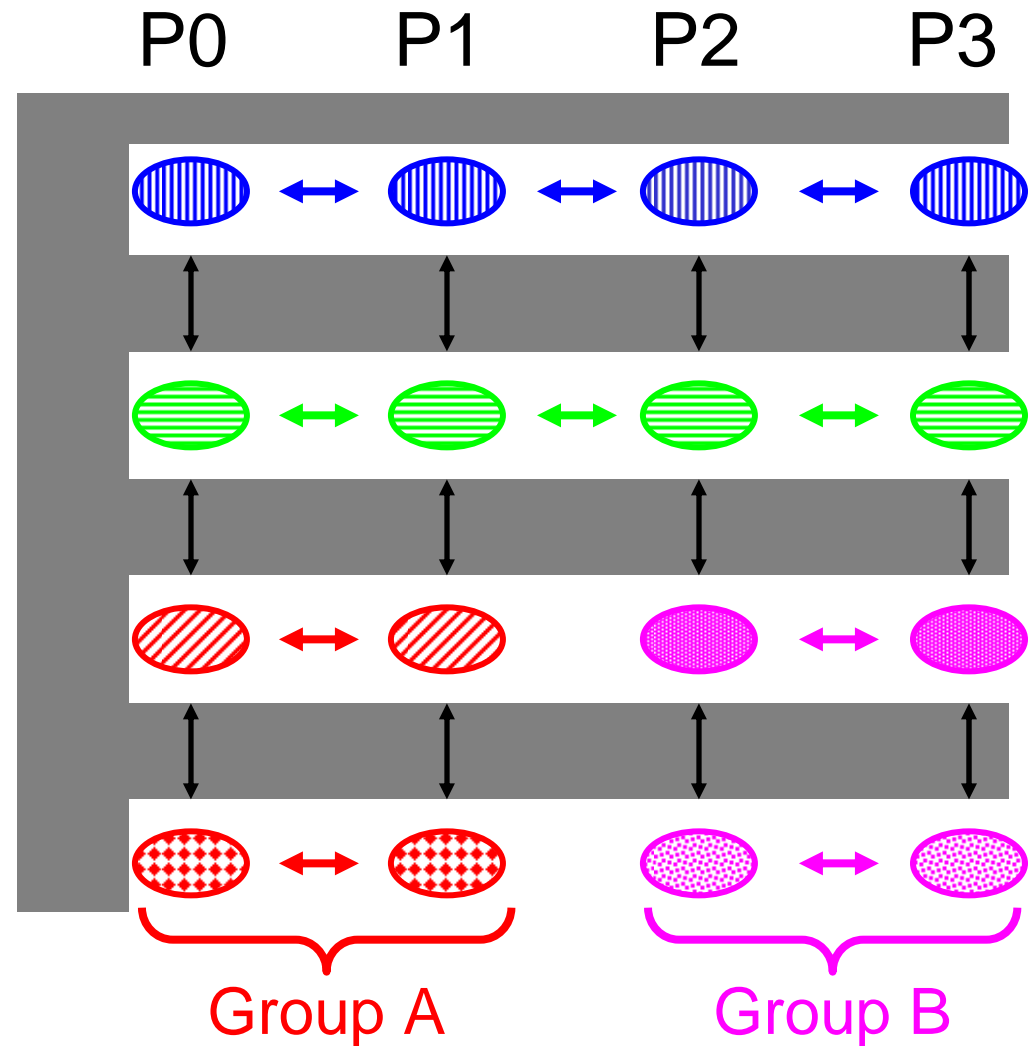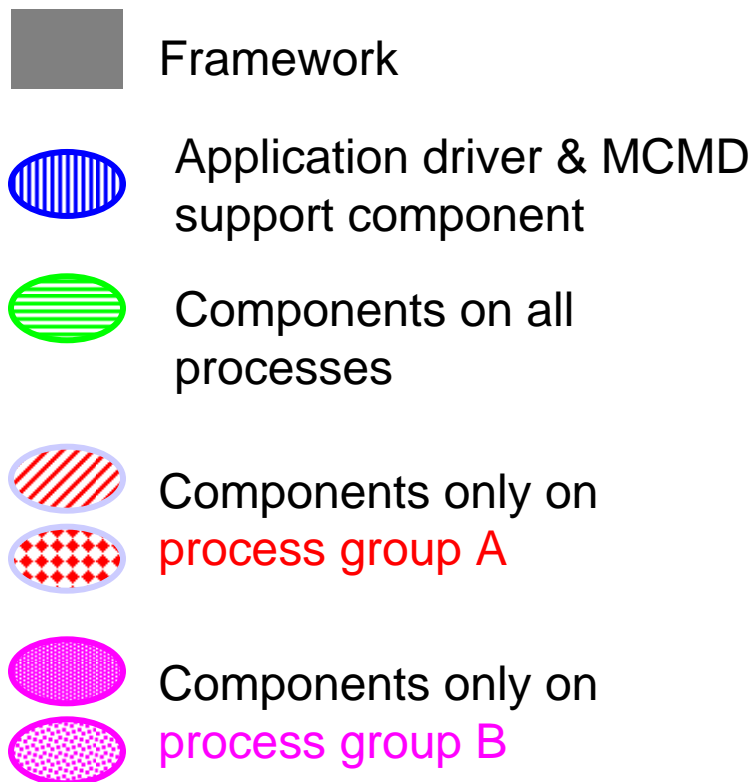
P0   P1   P2   P3
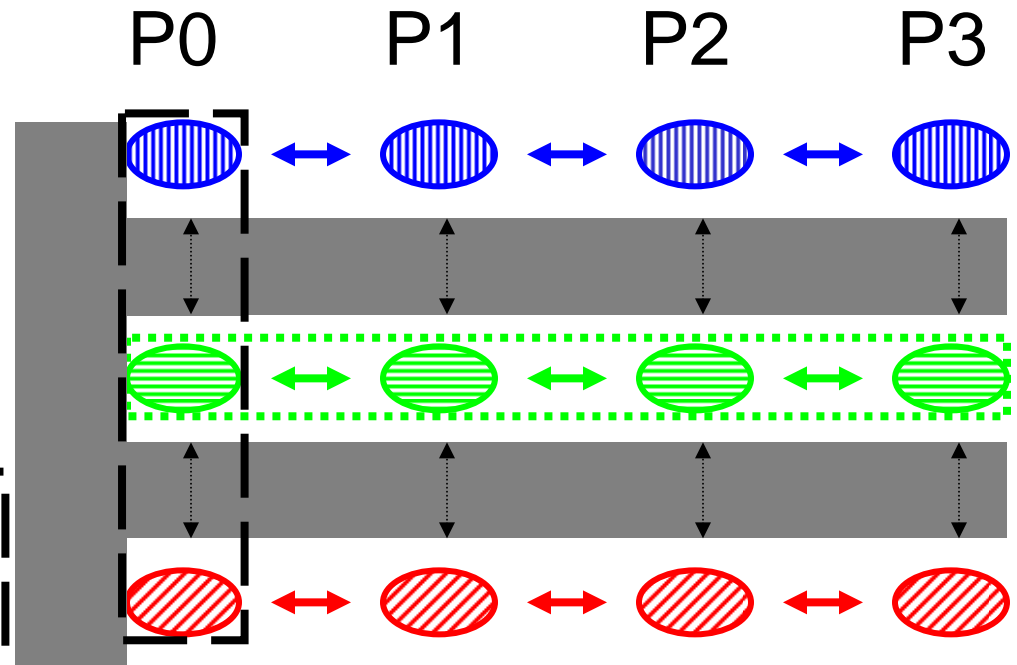
Components: Blue, Green, Red

Framework: Gray

Any parallel programming environments that can be mixed outside of CCA can be mixed inside

Multiple Component Multiple Data (MCMD) Parallelism

Useful for coupled parallel simulations, multi-level parallel algorithms, etc.

Framework

Application driver & MCMD support component

Components on all processes

Components only on process group A

Components only on process group B

P0   P1   P2   P3

Group A   Group B

Maintaining HPC Performance

- The performance of your application is as important to us as it is to you

- The CCA is designed to provide maximum performance
  - But the best we can do is to make your code perform no worse

- Facts:
  - Measured overheads per function call are low
  - Most overheads easily amortized by doing enough work per call
  - Other changes made during componentization may also have performance impacts
  - *Awareness* of costs of abstraction and language interoperability facilitates design for high performance

How Is the CCA Being Used Today?

- Many different application domains and interests (in CCA)
  - Combustion, quantum chemistry, radio astronomy, materials, fusion, particle physics, subsurface transport, cell biology, …

- To manage code complexity

- To facilitate collaborative software development

- To build computational toolkits and frameworks

- Multi-language interfaces for libraries

- Defining common interfaces to facilitate interoperability and reuse of software (libraries)

- Coupling of simulations

Computational Facility for Reacting Flow Science (CFRFS)

- A toolkit to perform simulations of unsteady flames

- Solve the Navier-Stokes with detailed chemistry
  - Various mechanisms up to ~50 species, 300 reactions
  - Structured adaptive mesh refinement

- CFRFS today:
  - 61 components
 - 7 external libraries
  - 9 contributors
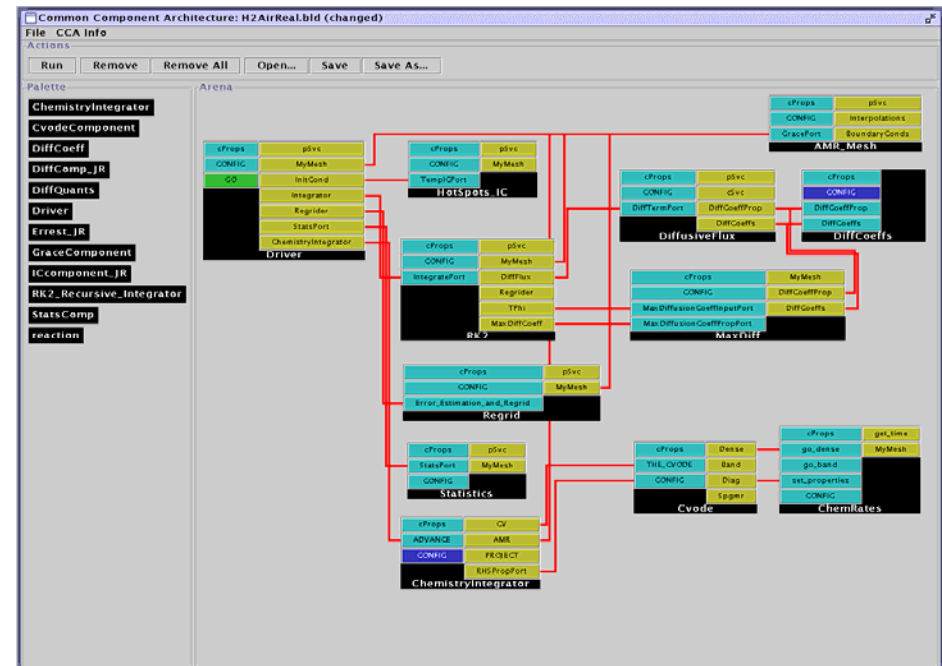
"Wiring diagram" for a typical CFRFS simulation, utilizing 12 components.

CCA tools used: Ccaffeine, and ccafe-gui
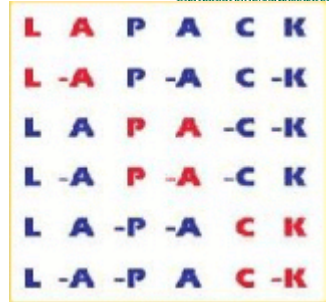**Languages:** C, C++, F77

TSTT Unstructured Mesh Tool Interoperability

- **Interoperability** -- multiple implementations conforming to the same interface

- **Reuse** – ability to use a component in many applications

- The larger the community that agrees to the interface, the greater the opportunity for interoperability and reuse
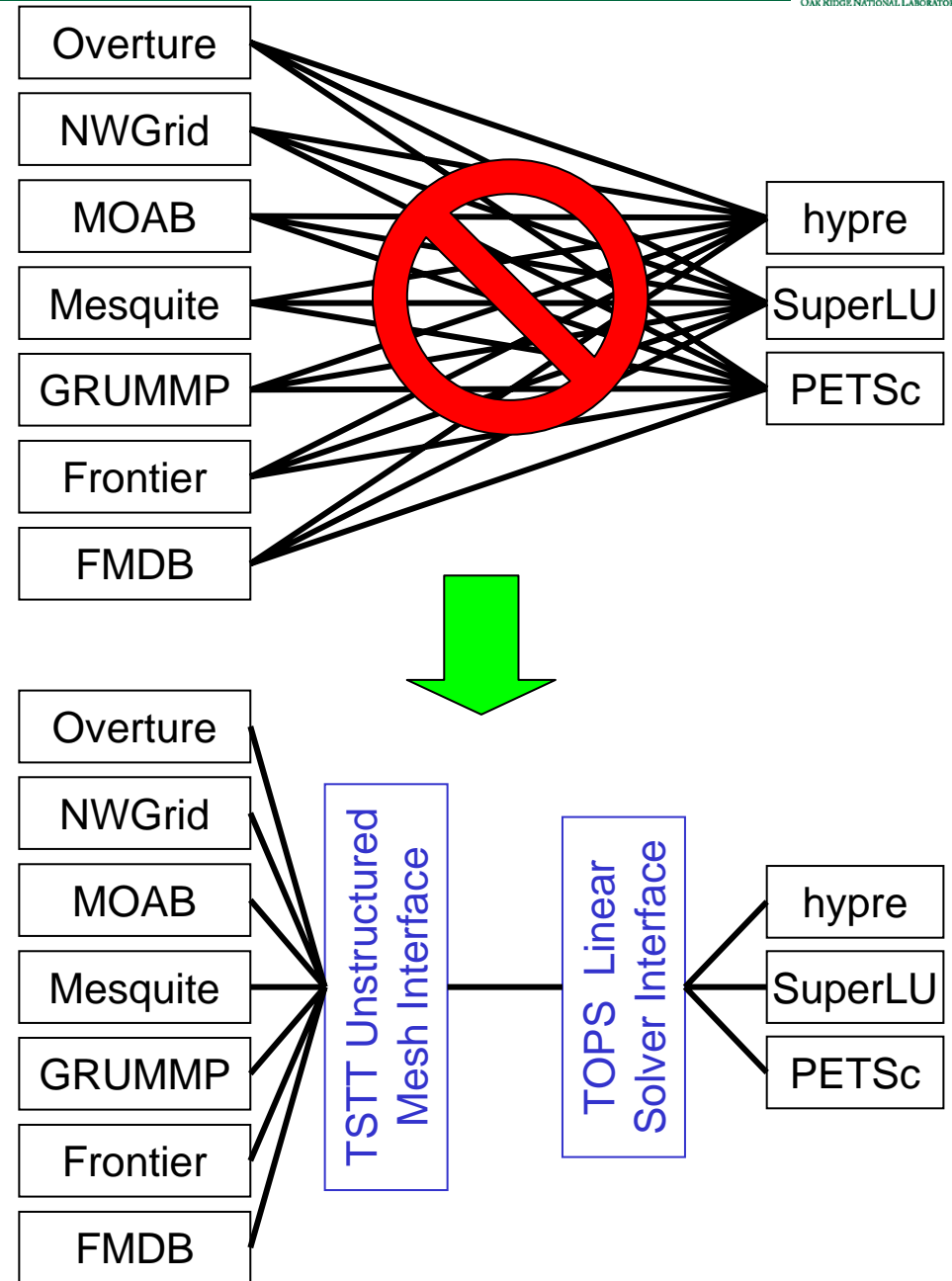
Language Interoperability

hypre

- High performance preconditioners and linear solvers

- Library written in C

- Babel-generated object-oriented interfaces provided in C, C++, Fortran

LAPACK07

- Update to LAPACK linear algebra library
  - To be released 2007
  - Library written in F77, F95

- Will use Babel-generated interfaces for: C, C++, F77, F95, Java, Python

- Possibly also ScaLAPACK (distributed version)

"I implemented a Babel-based interface for the hypre library of linear equation solvers. The Babel interface was straightforward to write and gave us interfaces to several languages for less effort than it would take to interface to a single language."

-- Jeff Painter, LLNL. 2 June 2003

CCA tools used: Babel, Chasm

Is CCA for You?

- Much of what CCA does can be done without such tools *if* you have sufficient discipline
  - The larger a group, the harder it becomes to impose the necessary discipline

- Projects may use different aspects of the CCA
  - CCA is *not* monolithic – use what *you* need
  - Few projects use all features of the CCA… initially

- Evaluate what *your* project needs against CCA's capabilities
  - CCA targets complex software development challenges
  - Not all projects are that complex

- Evaluate CCA against other ways of obtaining the desired capabilities

- Suggested starting point:
  - CCA tutorial "hands-on" exercises

Take an Evolutionary Approach

- The CCA is designed to allow selective use and incremental adoption

- "SIDLize" interfaces incrementally
  - Start with essential interfaces
  - Remember, only externally exposed interfaces need to be Babelized

- Componentize at successively finer granularities
  - Start with whole application as one component
    - Basic feel for components without "ripping apart" your app.
  - Subdivide into finer-grain components as appropriate
    - Code reuse opportunities
    - Plans for code evolution

Components in the Small: Impacts within a Project

Benefits include:

- Rapid testing, debugging, and benchmarking

- Support for implementation-hiding discipline

- Coordination of independent workers

- Interface change effects across components are clear and usually automatically found by compilers if overlooked

- Object-orientation made simpler for C and Fortran

Components in the Large: Connecting Multiple Projects

Benefits include:

- SIDL can be used to facilitate the interface consensus processes

- Different sub-projects do not have to agree on one implementation language

- Developers who never meet in person have an excellent chance of code integration working on the first try

Costs include:

- Consensus can be expensive to obtain

- Writing code for others to use is more difficult than writing it just for yourself

View it as an Investment

- CCA is a long-term investment in your software
  - Like most software engineering approaches

- There is a cost to adopt

- The payback is longer term

- Biggerstaff's Rule of Threes
  - Must look at at least three systems to understand what is common (reusable)
  - Reusable software requires three times the effort of usable software
  - Payback only after third release

Where is CCA Now?

- CCA development has been driven in large part by the needs of our application partners
  - Many useful and interesting ideas/needs we haven't had time/funding to address
  - Happy to collaborate to do new things

- CCA specification stable, and near "1.0"

- CCA tools available and working
  - Ccaffeine framework, Babel, Chasm lang. interop.
  - Distributed framework lags somewhat
  - Build procedures somewhat cumbersome
  - Need to be more widely ported & tested
  - Need additional tools to automate tedious aspects of software development process

- Many applications in many different fields of science are using CCA
  - Combustion, quantum chemistry, radio astronomy, materials, fusion, particle physics, subsurface transport, cell biology, …

- We do *not* yet have a large suite of ready to use "off the shelf" components

Looking Forward

- Center for Technology for Advanced Scientific Component Software (TASCS)
  - ANL, LANL, LLNL, ORNL, PNNL, SNL
  - Binghamton, Indiana, Maryland, Utah
  - Tech-X Corp.

- Funded by DOE SciDAC program 2006-2011

- $3M/year

TASCS Component Technology Development Initiatives

- ## Computational Quality of Service (CQoS)
  - Dynamic adaptation of running applications in response to performance, numerical, or other criteria

- ## Software Quality and Verification
  - Semantic annotations on component interfaces, performance-sensitive enforcement of assertions

- ## Emerging HPC Paradigms
  - Using component environments to facilitate programming coming massively parallel and heterogeneous systems

TASCS Support for the CCA Environment

- Maintaining, supporting, and porting the core tools

- Enhancing the CCA environment
  - Completing and extending the CCA specification
  - Interoperability with other component-like environments
  - SIDL/Babel enhancements

- Usability
  - "CCA Lite"
  - Debugging and Testing

TASCS Component Toolkit

- **Component development tools**
  - Command line
  - IDE

- **CCA component collection**

- **Community interface development**

- **Component repository**

TASCS Outreach and Support

- Application support

- User outreach and support
  - Tutorials, coding camps, etc.

- Community outreach

The CCA is Not Just About Tools

- Use the *ideas* of component technology, software architecture, and software engineering

- Think about the software in small, manageable pieces (components)

- Think about how the pieces interact (interfaces)

- Think about how to organize the pieces into the application

- Think about software reuse, interoperability, and common interfaces

Even the (Component) Ideas Can't Solve all Problems

- Coupled simulation especially involves issues that component concepts can't address
  - New physics exposed by the coupling
  - Mathematical/numerical issues

- The social and sociological aspects are often at least as important as the scientific or technical to success

More Information

- ## CCA Forum
  - Web site: http://www.cca-forum.org
  - Mailing list: cca-forum@cca-forum.org
  - In person: quarterly meetings (next: 5-6 Oct @ ANL)

- ## Babel
  - Web site: http://www.llnl.gov/CASC/components

- ## Papers
  - BA Allan, et al., A Component Architecture for High-Performance Scientific Computing, *Intl. J. High-Perf. Computing Appl.* **20**, 163 (2006)
  - LC McInnes, et al., Parallel PDE-Based Simulations Using the Common Component Architecture, in Are Magnus Bruaset and Aslak Tveito, editors, *Numerical Solution of PDEs on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering (LNCSE)*, pages 327-384, Springer-Verlag, 2006

- ## Me: bernholdtde@ornl.gov