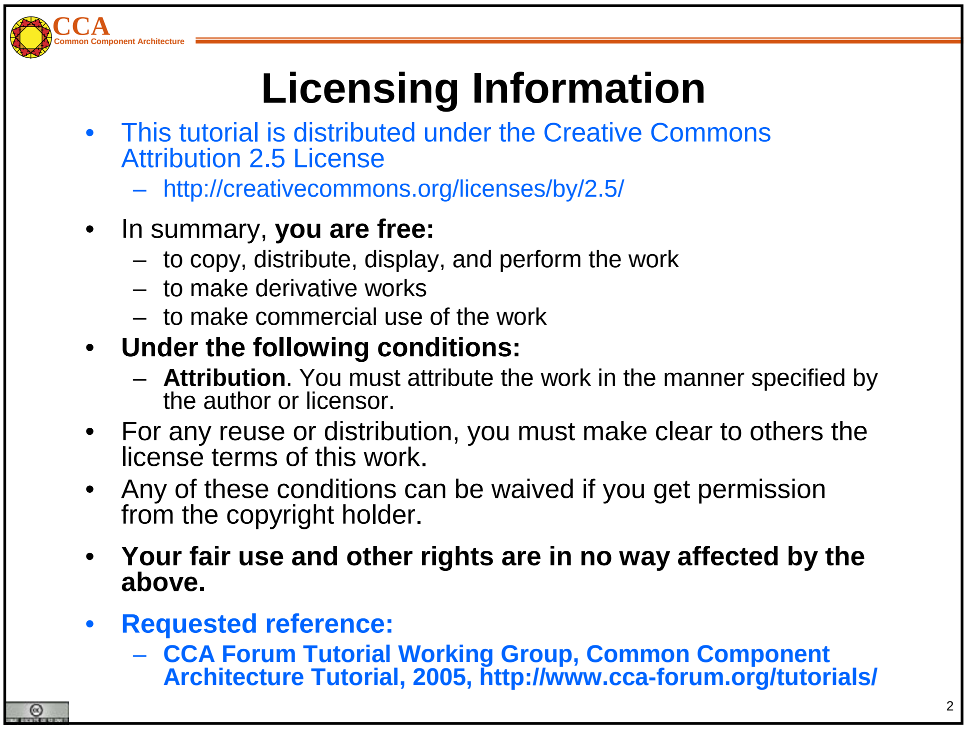
CCA
Common Component Architecture

Welcome to the Common Component Architecture Tutorial

ACTS Workshop

26 August 2005

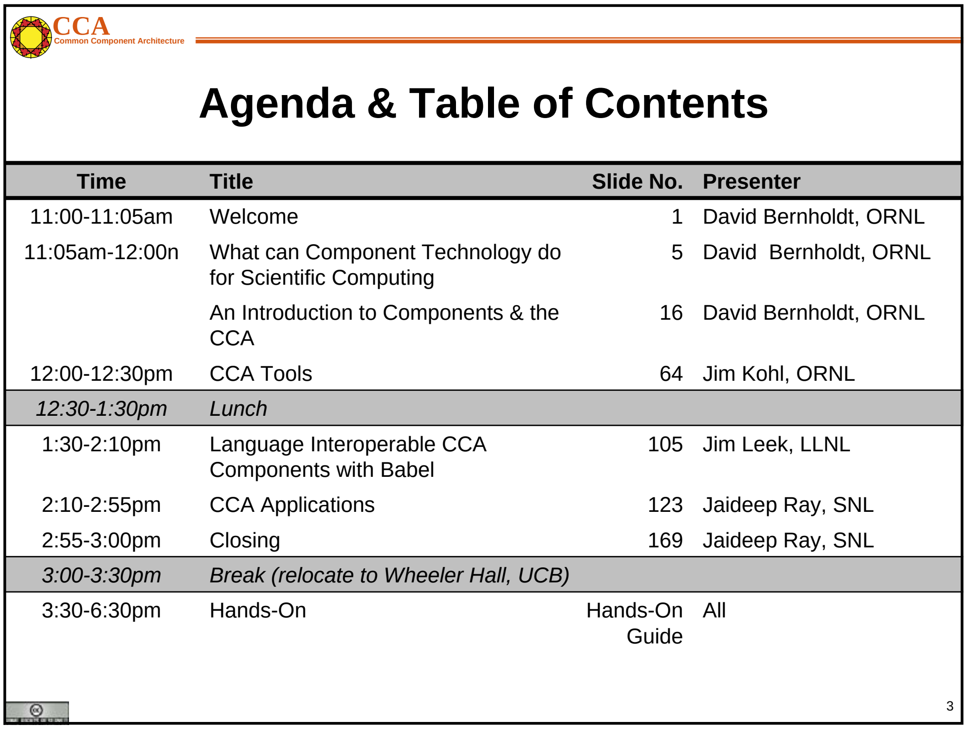
CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org



CCA
Common Component Architecture

Licensing Information

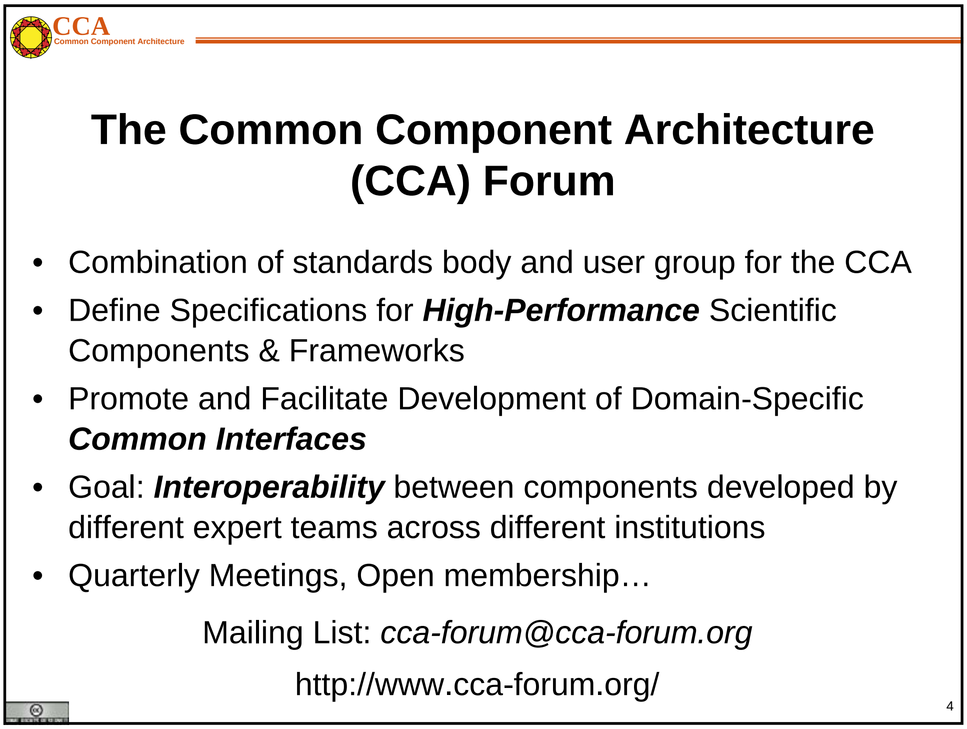
- This tutorial is distributed under the Creative Commons Attribution 2.5 License
  - http://creativecommons.org/licenses/by/2.5/
- In summary, **you are free:**
  - to copy, distribute, display, and perform the work
  - to make derivative works
  - to make commercial use of the work
- **Under the following conditions:**
  - **Attribution**. You must attribute the work in the manner specified by the author or licensor.
- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.
- **Your fair use and other rights are in no way affected by the above.**
- **Requested reference:**
  - **CCA Forum Tutorial Working Group, Common Component Architecture Tutorial, 2005, http://www.cca-forum.org/tutorials/**



CCA Common Component Architecture

Agenda & Table of Contents

Time	Title	Slide No.	Presenter
11:00-11:05am	Welcome	1	David Bernholdt, ORNL
11:05am-12:00n	What can Component Technology do for Scientific Computing	5	David Bernholdt, ORNL
	An Introduction to Components & the CCA	16	David Bernholdt, ORNL
12:00-12:30pm	CCA Tools	64	Jim Kohl, ORNL
12:30-1:30pm	*Lunch*		
1:30-2:10pm	Language Interoperable CCA Components with Babel	105	Jim Leek, LLNL
2:10-2:55pm	CCA Applications	123	Jaideep Ray, SNL
2:55-3:00pm	Closing	169	Jaideep Ray, SNL
3:00-3:30pm	*Break (relocate to Wheeler Hall, UCB)*		
3:30-6:30pm	Hands-On	Hands-On Guide	All



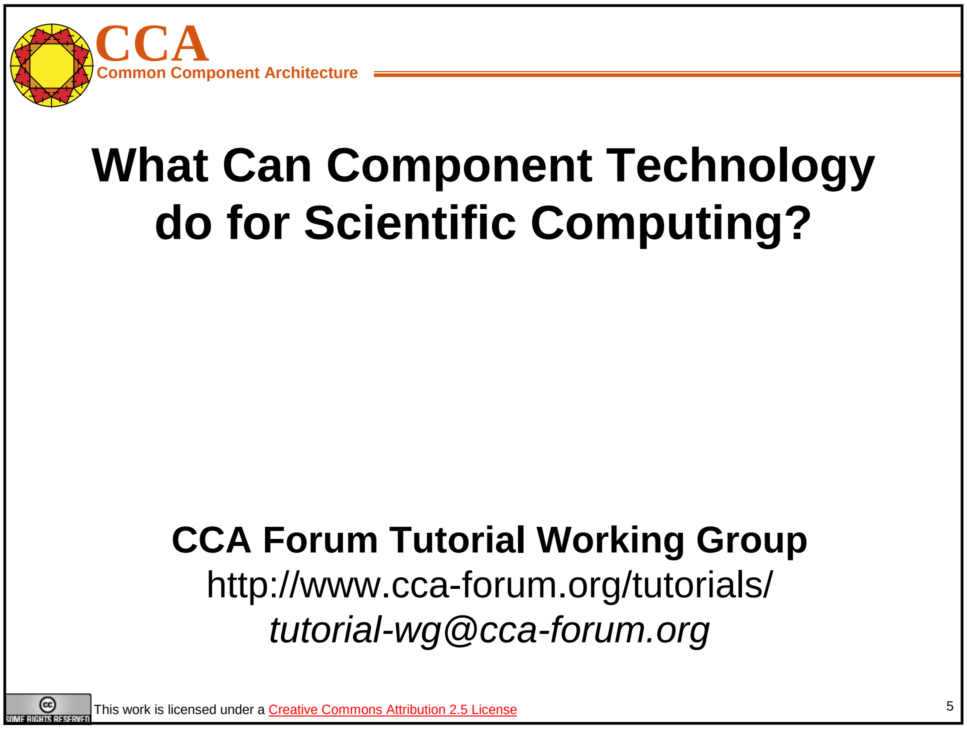
CCA Common Component Architecture

The Common Component Architecture (CCA) Forum

- Combination of standards body and user group for the CCA
- Define Specifications for ***High-Performance*** Scientific Components & Frameworks
- Promote and Facilitate Development of Domain-Specific ***Common Interfaces***
- Goal: ***Interoperability*** between components developed by different expert teams across different institutions
- Quarterly Meetings, Open membership…

Mailing List: *cca-forum@cca-forum.org*

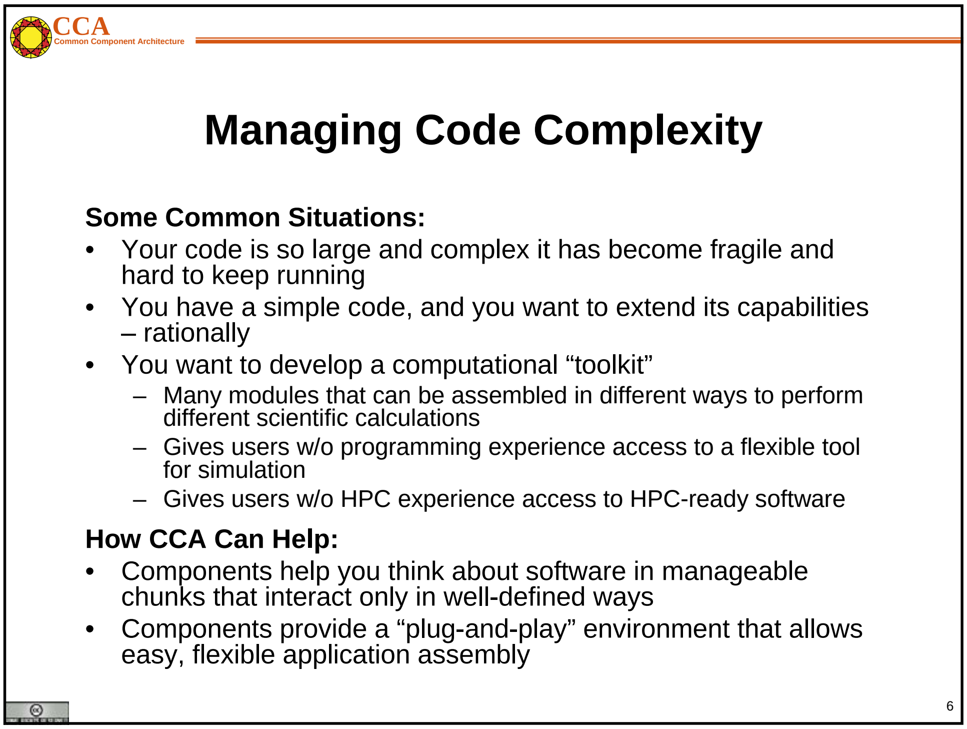
http://www.cca-forum.org/



CCA
Common Component Architecture

What Can Component Technology do for Scientific Computing?

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org



CCA
Common Component Architecture

Managing Code Complexity

Some Common Situations:

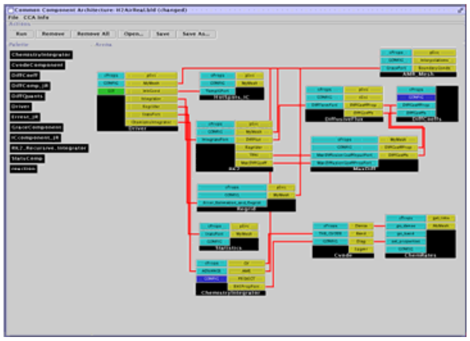
- Your code is so large and complex it has become fragile and hard to keep running
- You have a simple code, and you want to extend its capabilities – rationally
- You want to develop a computational "toolkit"
  - Many modules that can be assembled in different ways to perform different scientific calculations
  - Gives users w/o programming experience access to a flexible tool for simulation
  - Gives users w/o HPC experience access to HPC-ready software

How CCA Can Help:

- Components help you think about software in manageable chunks that interact only in well-defined ways
- Components provide a "plug-and-play" environment that allows easy, flexible application assembly

Example: 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

Helping Groups Work with Software

Some Common Situations:

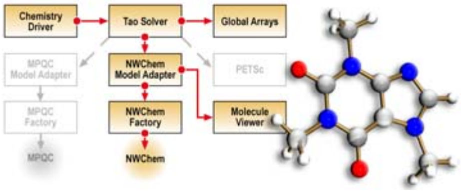
- Many (geographically distributed) developers creating a large software system
  - Hard to coordinate, different parts of the software don't work together as required
- Groups of developers with different specialties
- Forming communities to standardize interfaces or share code

How CCA Can Help:

- Components are natural units for
  - Expressing software architecture
  - Individuals or small groups to develop
  - Encapsulating particular expertise
- Some component models (including CCA) provide tools to help you think about the *interface* separately from the *implementation*

Example: Quantum Chemistry

- Integrated state-of-the-art optimization technology into two quantum chemistry packages to explore effectiveness in chemistry applications
- Geographically distributed expertise:
  - California - chemistry
  - Illinois - optimization
  - Washington – chemistry, parallel data management
- Effective collaboration with minimal face-to-face interaction



Schematic of CCA-based molecular structure determination quantum chemistry application.

Components based on: MPQC, NWChem (quantum chem.), TAO (optimization), Global Arrays, PETSc (parallel linear algebra)

CCA tools used: Babel, Ccaffeine, and ccafe-gui

Languages: C, C++, F77, Python

Example: TSTT Unstructured Mesh Tool Interoperability

- Common interface for unstructured mesh geometry and topology
  - 7 libraries: FMDB, Frontier, GRUMMP, Mesquite, MOAB, NWGrid, Overture
  - 6 institutions: ANL, BNL/SUNY-Stony Brook, LLNL, PNNL, RPI, SNL
- Reduces need for N^2 pairwise interfaces to just *N*

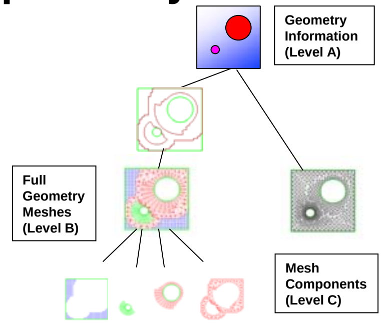
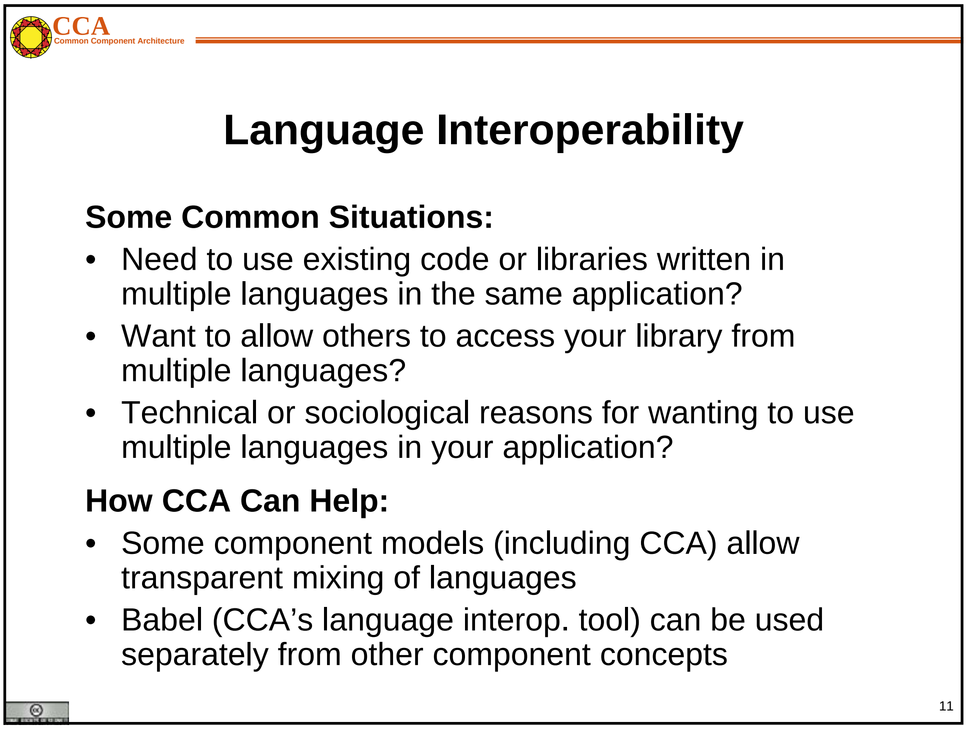

Geometry Information (Level A)
Full Geometry Meshes (Level B)
Mesh Components (Level C)

Illustration of geometry domain hierarchy used in TSTT mesh interface.

CCA tools used: Babel (SIDL), Chasm

Library languages: C, C++, F77, F90



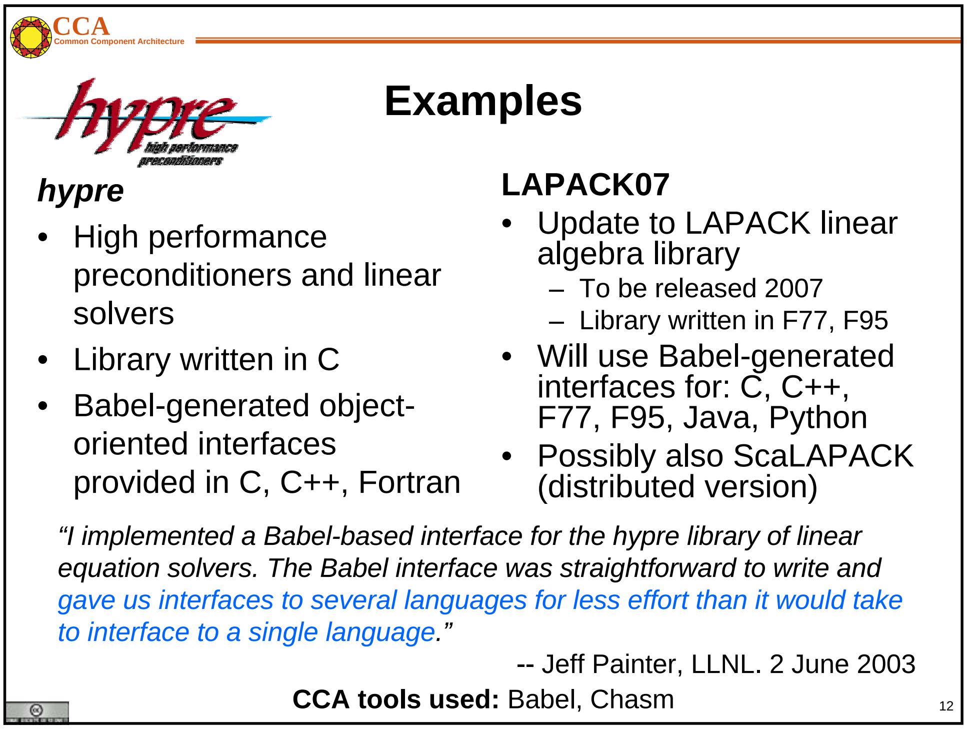
Language Interoperability

Some Common Situations:

- Need to use existing code or libraries written in multiple languages in the same application?
- Want to allow others to access your library from multiple languages?
- Technical or sociological reasons for wanting to use multiple languages in your application?

How CCA Can Help:

- Some component models (including CCA) allow transparent mixing of languages
- Babel (CCA's language interop. tool) can be used separately from other component concepts



Examples

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

Coupling Codes

Some Common Situations:

- Your application makes use of numerous third-party libraries
  - Some of which interact (version dependencies)
- You want to develop a simulation in which your code is coupled with others
  - They weren't designed with this coupling in mind
  - They must remain usable separately too
  - They are all under continual development, individually
  - They're all parallel and need to exchange data frequently

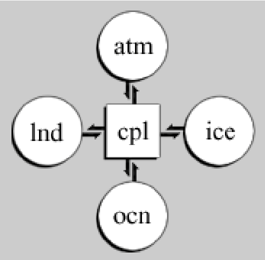
How CCA Can Help:

- Components are isolated from one another
  - Interactions via well-defined interfaces
  - An application can include multiple versions of a component
- Components can be composed flexibly, hierarchically
  - Standalone application as one assembly, coupled simulation as another
- CCA can be used in SPMD, MPMD, and distributed styles of parallel computing
- CCA is developing technology to facilitate data and functional coupling of parallel applications

Example: Global Climate Modeling and the Model Coupling Toolkit (MCT)

- MCT is the basis for Community Climate System Model (CCSM3.0) coupler (cpl6)
- Computes interfacial fluxes and manages redistribution of data among parallel processes
- Written in F90, Babel-generated bindings for C++, Python
- ***CCA tools used:*** *Babel, Chasm*

mct



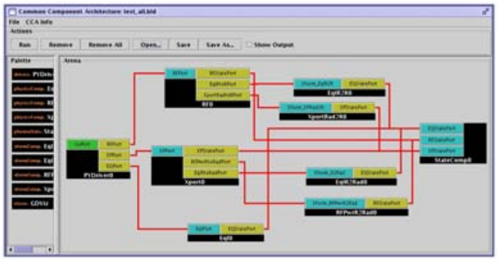
Schematic of CCSM showing coupler managing data exchanges between atmosphere, sea ice, ocean, and land models.

(From http://www.ccsm.ucar.edu/models/ccsm3.0/cpl6/)

CCA
Common Component Architecture

Example: Integrated Fusion Simulation

- Proof-of-principle of using CCA for integrated whole-device modeling needed for the ITER fusion reactor
- Couples radio frequency (RF) heating of plasma with transport modeling
- Coarse-grain encapsulation of pre-existing programs
- Follow-on plans for RF, transport, and magneto-hydrodynamics



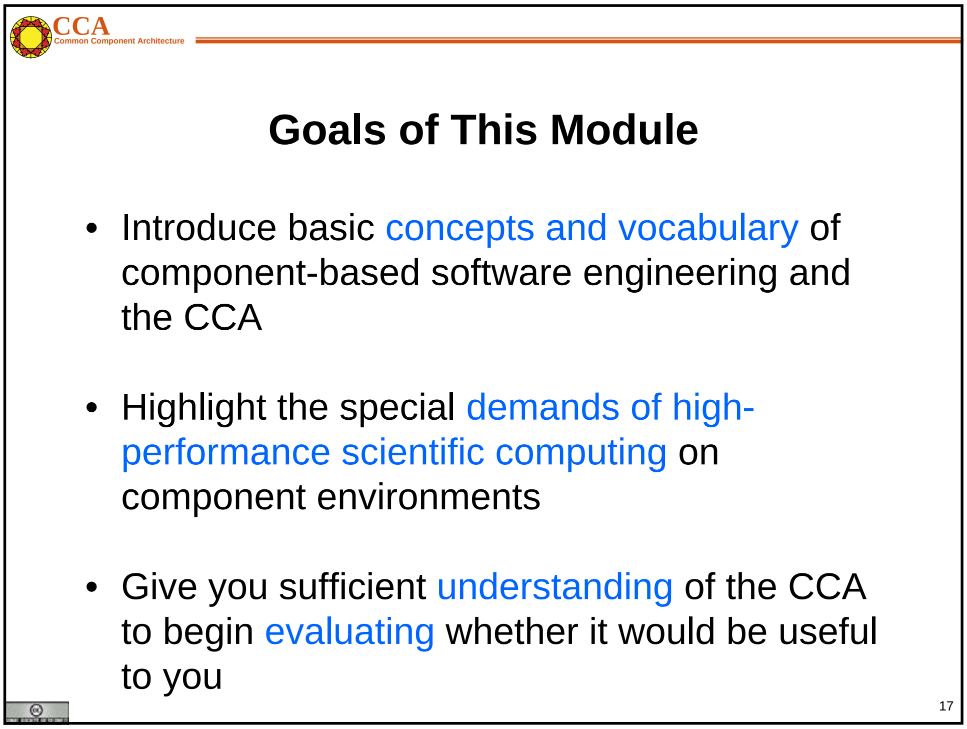
"Wiring diagram" for integrated fusion simulation.

Components based on: AORSA, Houlberg's transport library
New components: Driver, State
CCA tools used: Babel, Chasm, Ccaffeine, ccafe-gui
Languages: C++, F90, Python

CCA
Common Component Architecture

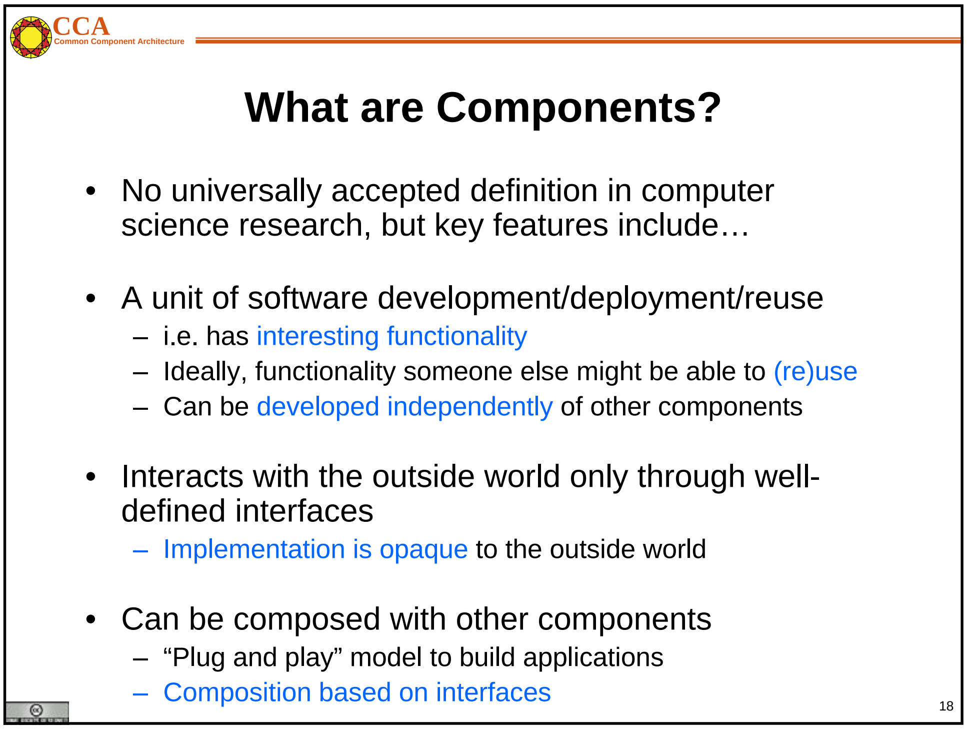
An Introduction to Components and the Common Component Architecture

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org



Goals of This Module

- Introduce basic concepts and vocabulary of component-based software engineering and the CCA
- Highlight the special demands of high-performance scientific computing on component environments
- Give you sufficient understanding of the CCA to begin evaluating whether it would be useful to you



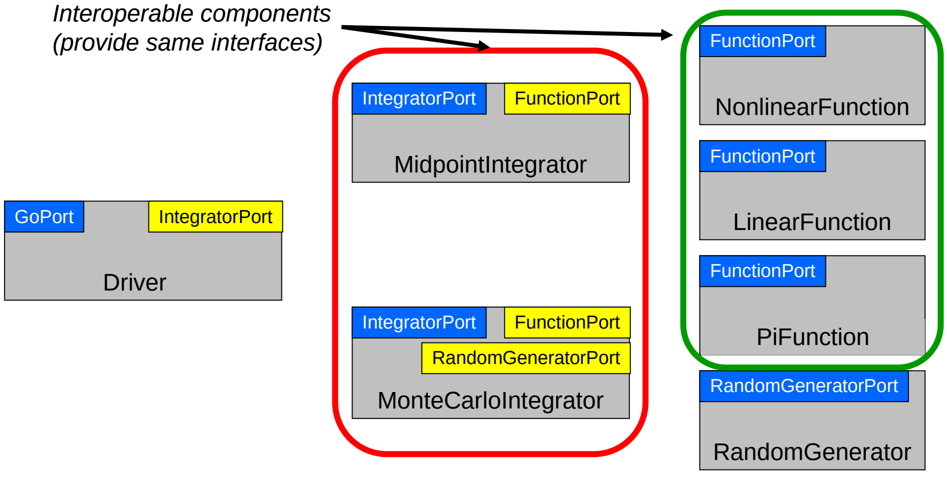
What are Components?

- 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

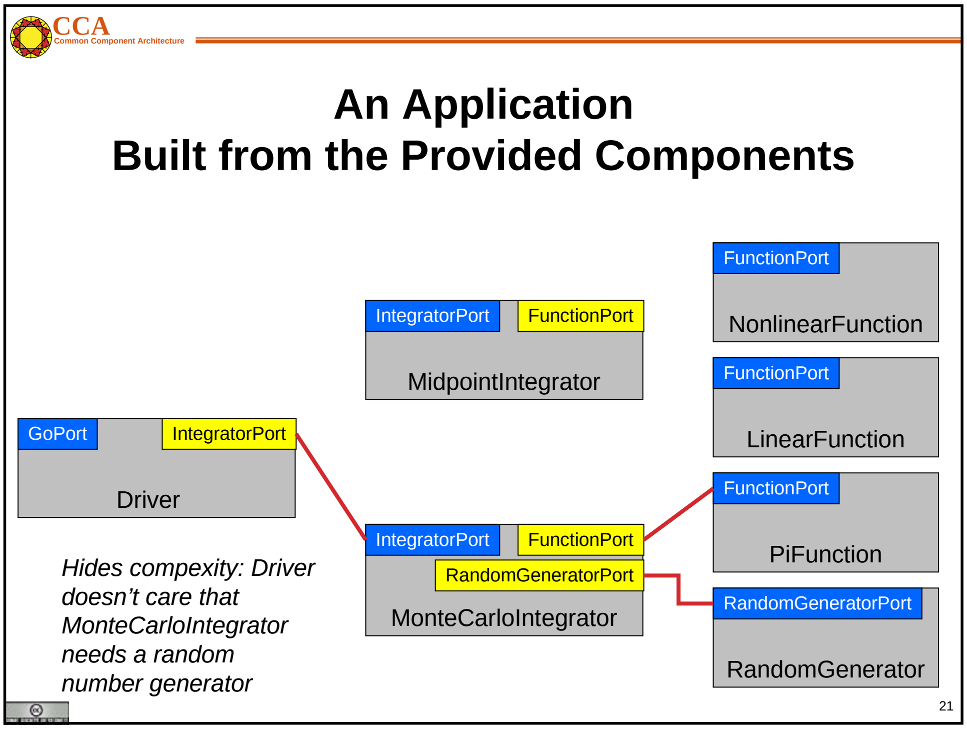
What is a Component Architecture?

- 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

A Simple Example: Numerical Integration Components



Interoperable components (provide same interfaces)



An Application Built from the Provided Components

GoPort IntegratorPort Driver

IntegratorPort FunctionPort MidpointIntegrator

IntegratorPort FunctionPort RandomGeneratorPort MonteCarloIntegrator

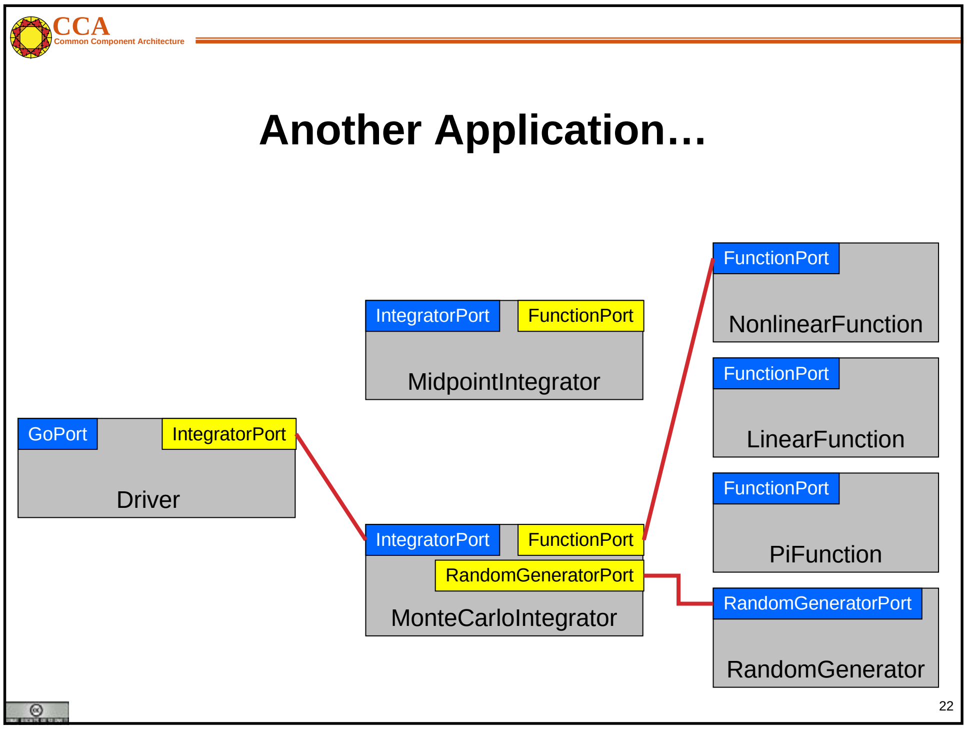
FunctionPort NonlinearFunction

FunctionPort LinearFunction

FunctionPort PiFunction

RandomGeneratorPort RandomGenerator

Hides compexity: Driver doesn't care that MonteCarloIntegrator needs a random number generator



Another Application…

GoPort IntegratorPort Driver

IntegratorPort FunctionPort MidpointIntegrator

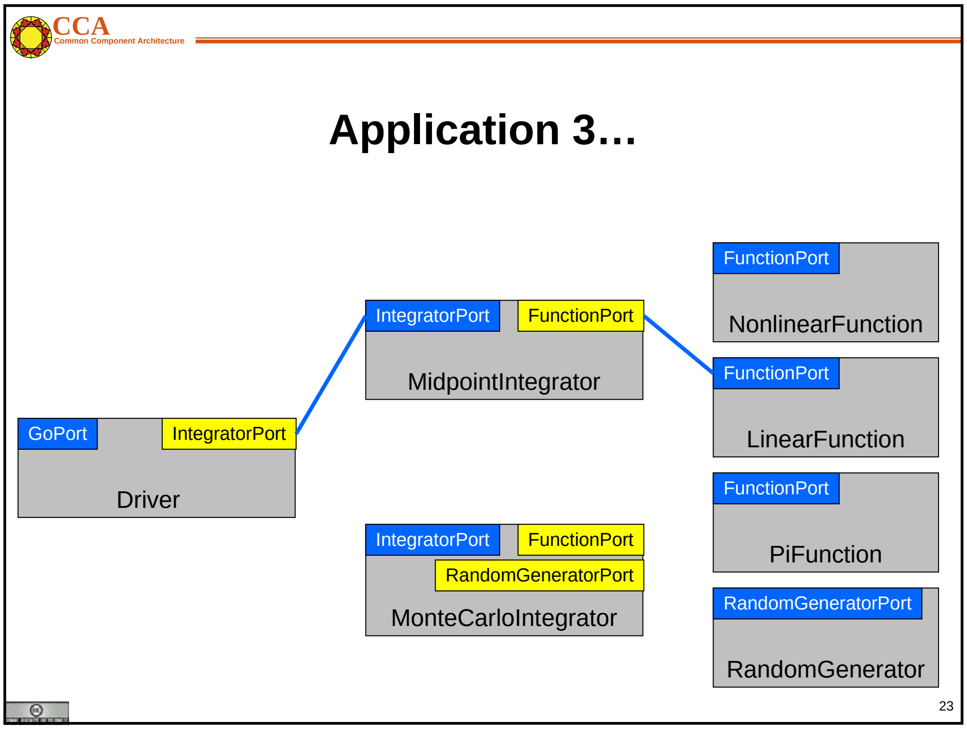
IntegratorPort FunctionPort RandomGeneratorPort MonteCarloIntegrator

FunctionPort NonlinearFunction

FunctionPort LinearFunction

FunctionPort PiFunction

RandomGeneratorPort RandomGenerator



CCA
Common Component Architecture

Application 3…

FunctionPort
NonlinearFunction

IntegratorPort
FunctionPort
MidpointIntegrator

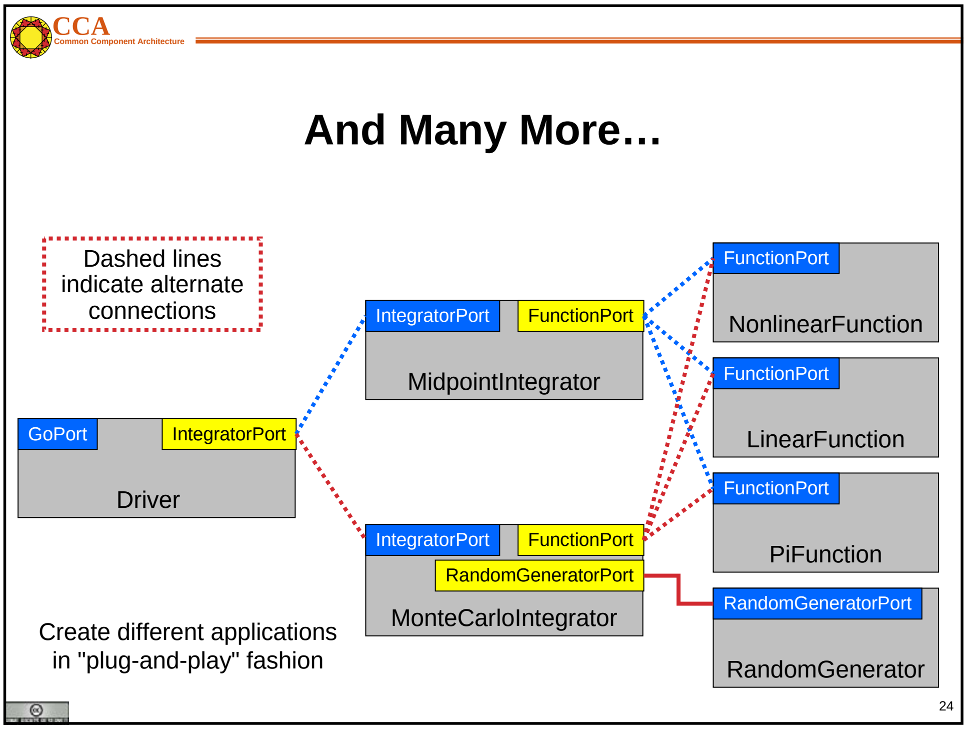
FunctionPort
LinearFunction

GoPort
IntegratorPort
Driver

FunctionPort
PiFunction

IntegratorPort
FunctionPort
RandomGeneratorPort
MonteCarloIntegrator

RandomGeneratorPort
RandomGenerator



CCA
Common Component Architecture

And Many More…

Dashed lines indicate alternate connections

FunctionPort
NonlinearFunction

IntegratorPort
FunctionPort
MidpointIntegrator

FunctionPort
LinearFunction

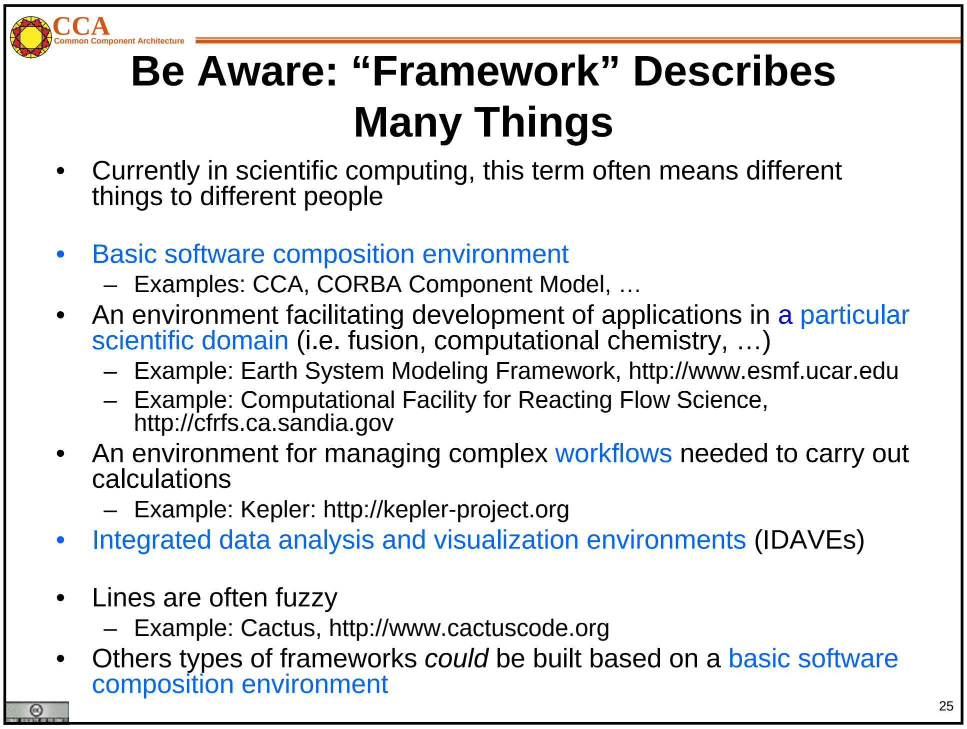
GoPort
IntegratorPort
Driver

FunctionPort
PiFunction

IntegratorPort
FunctionPort
RandomGeneratorPort
MonteCarloIntegrator

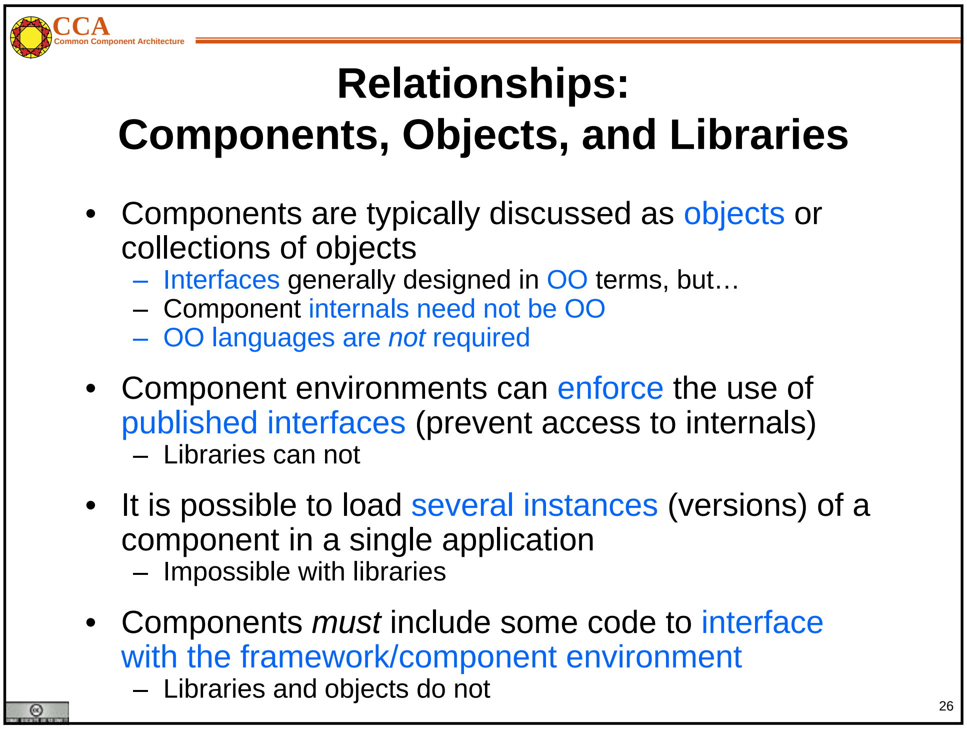
RandomGeneratorPort
RandomGenerator

Create different applications in "plug-and-play" fashion



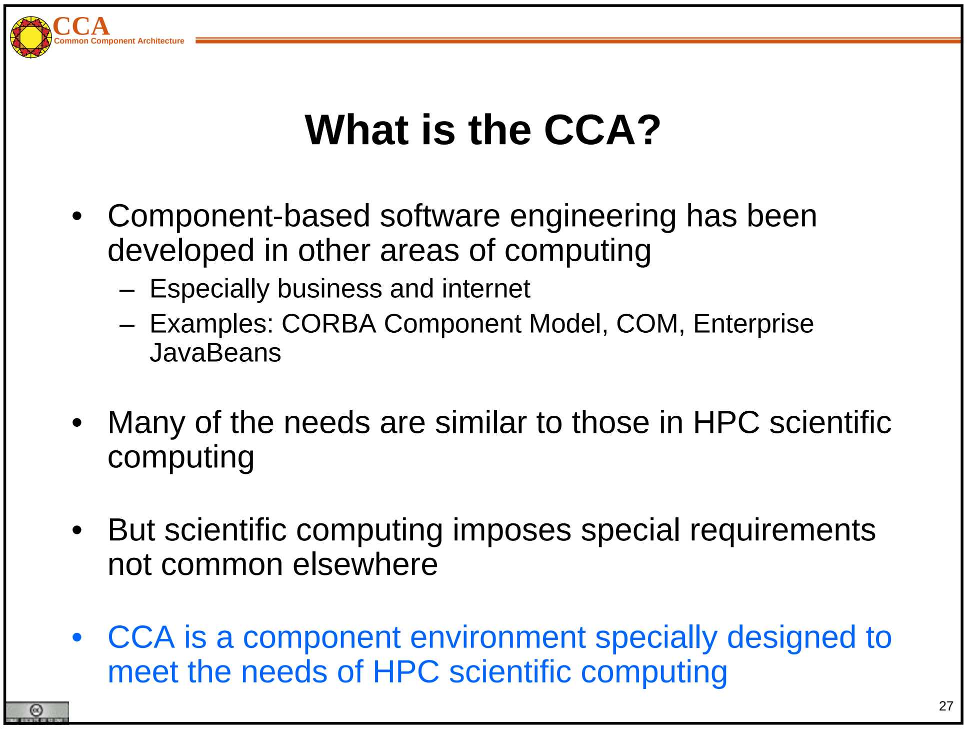
Be Aware: “Framework” Describes Many Things

- Currently in scientific computing, this term often means different things to different people
- Basic software composition environment
  - Examples: CCA, CORBA Component Model, …
- An environment facilitating development of applications in a particular scientific domain (i.e. fusion, computational chemistry, …)
  - Example: Earth System Modeling Framework, http://www.esmf.ucar.edu
  - Example: Computational Facility for Reacting Flow Science, http://cfrfs.ca.sandia.gov
- An environment for managing complex workflows needed to carry out calculations
  - Example: Kepler: http://kepler-project.org
- Integrated data analysis and visualization environments (IDAVEs)
- Lines are often fuzzy
  - Example: Cactus, http://www.cactuscode.org
- Others types of frameworks *could* be built based on a basic software composition environment



Relationships: Components, Objects, and Libraries

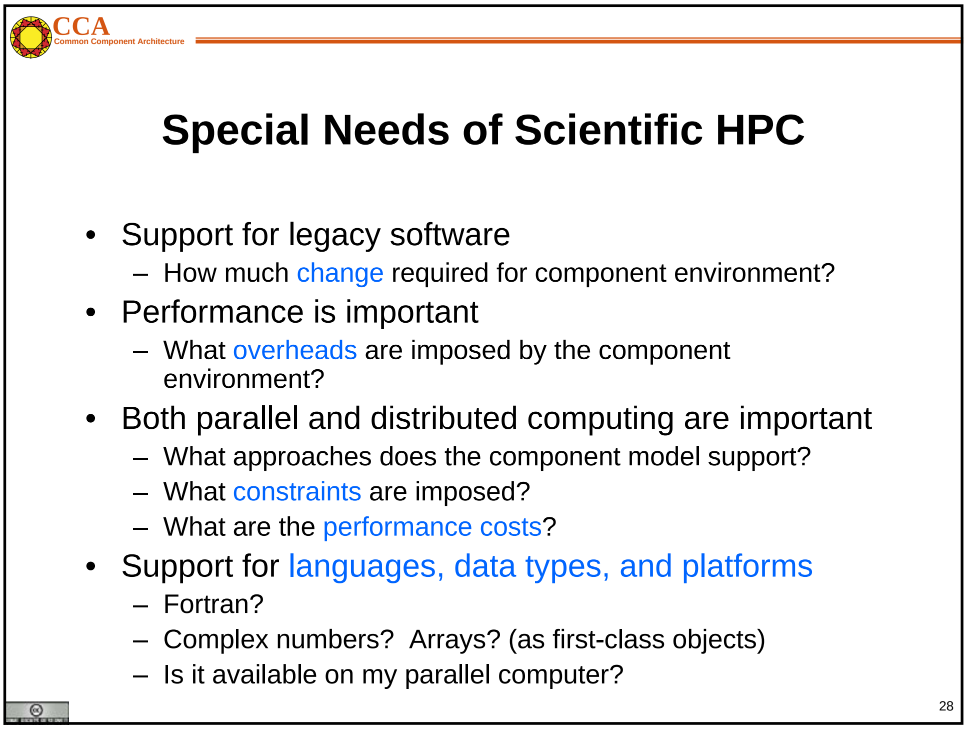
- Components are typically discussed as objects or collections of objects
  - Interfaces generally designed in OO terms, but…
  - Component internals need not be OO
  - OO languages are *not* required
- Component environments can enforce the use of published interfaces (prevent access to internals)
  - Libraries can not
- It is possible to load several instances (versions) of a component in a single application
  - Impossible with libraries
- Components *must* include some code to interface with the framework/component environment
  - Libraries and objects do not



CCA
Common Component Architecture

What is the CCA?

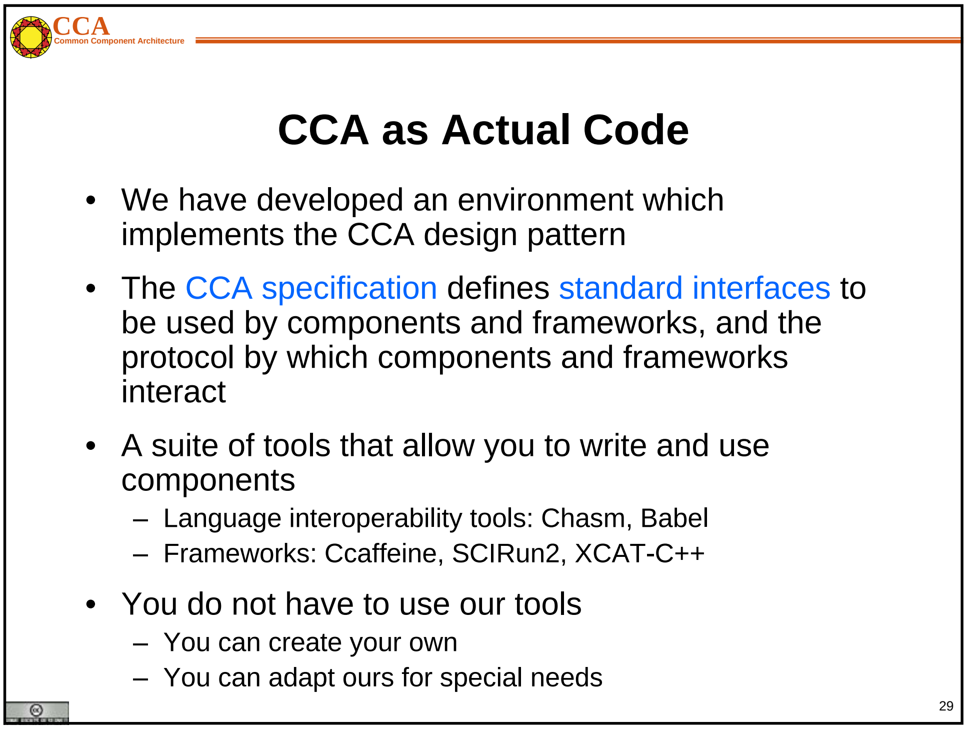
- Component-based software engineering has been developed in other areas of computing
  - Especially business and internet
  - Examples: CORBA Component Model, COM, Enterprise JavaBeans
- Many of the needs are similar to those in HPC scientific computing
- But scientific computing imposes special requirements not common elsewhere
- CCA is a component environment specially designed to meet the needs of HPC scientific computing



CCA
Common Component Architecture

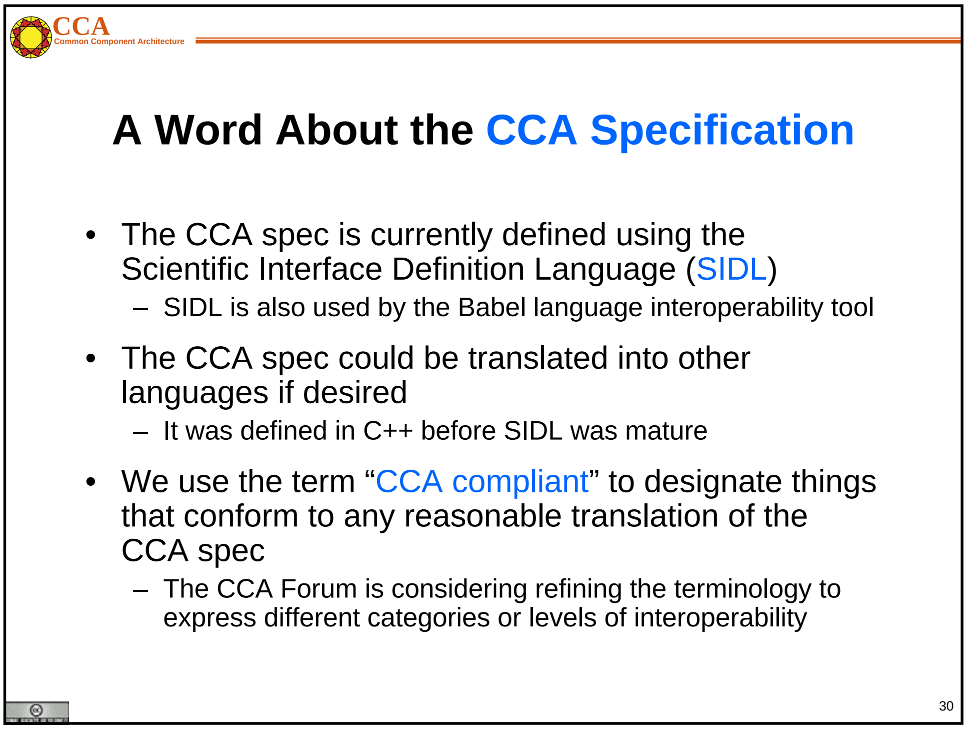
Special Needs of Scientific HPC

- 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?



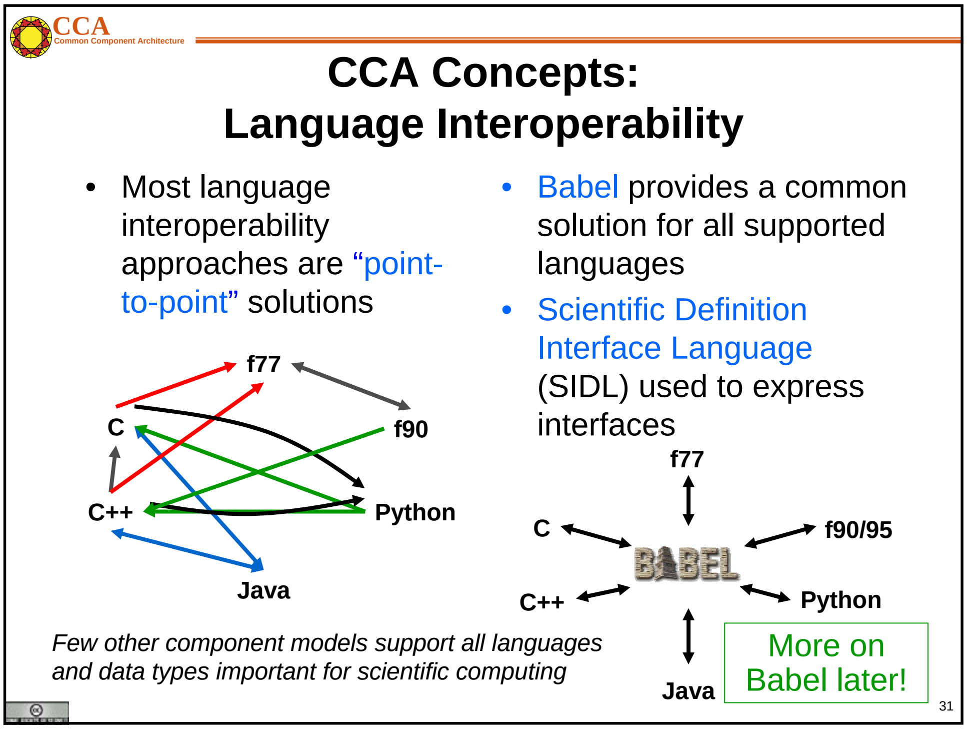
CCA as Actual Code

- We have developed an environment which implements the CCA design pattern
- The CCA specification defines standard interfaces to be used by components and frameworks, and the protocol by which components and frameworks interact
- A suite of tools that allow you to write and use components
  - Language interoperability tools: Chasm, Babel
  - Frameworks: Ccaffeine, SCIRun2, XCAT-C++
- You do not have to use our tools
  - You can create your own
  - You can adapt ours for special needs



A Word About the CCA Specification

- The CCA spec is currently defined using the Scientific Interface Definition Language (SIDL)
  - SIDL is also used by the Babel language interoperability tool
- The CCA spec could be translated into other languages if desired
  - It was defined in C++ before SIDL was mature
- We use the term “CCA compliant” to designate things that conform to any reasonable translation of the CCA spec
  - The CCA Forum is considering refining the terminology to express different categories or levels of interoperability



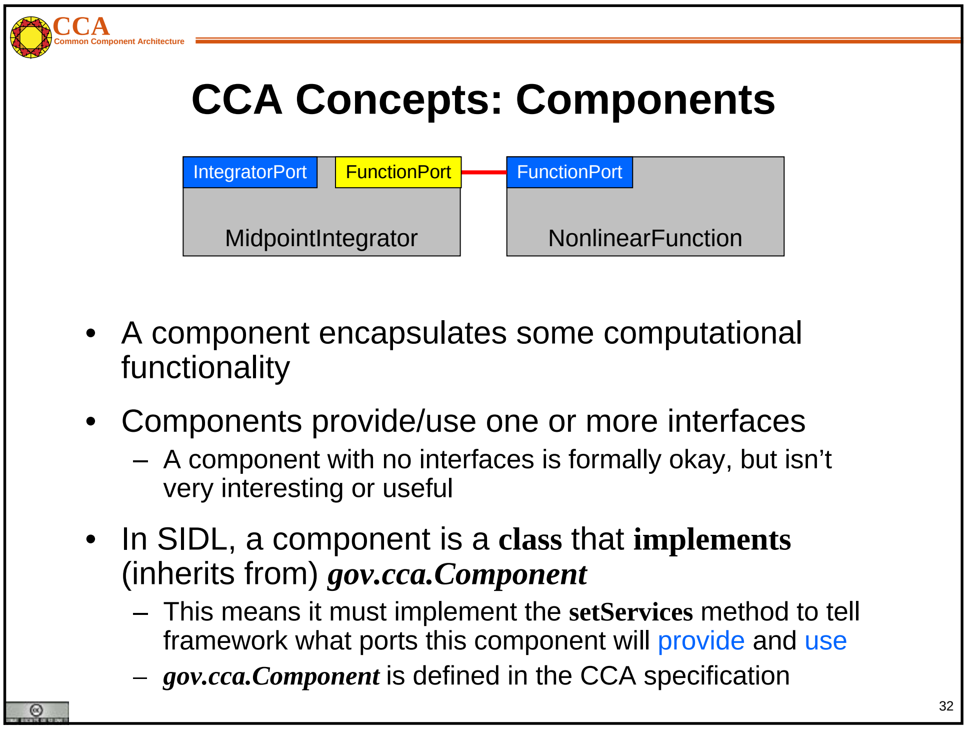
CCA Concepts: Language Interoperability

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

- Babel provides a common solution for all supported languages
- Scientific Definition Interface Language (SIDL) used to express interfaces

Few other component models support all languages and data types important for scientific computing

More on Babel later!

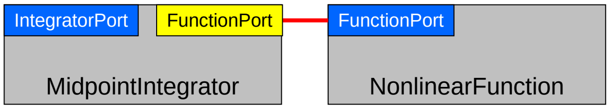


CCA Concepts: Components

- 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
Common Component Architecture

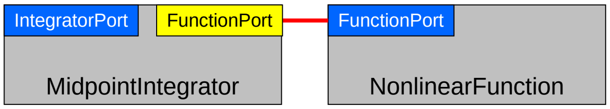
CCA Concepts: Ports


IntegratorPort
FunctionPort
FunctionPort
MidpointIntegrator
NonlinearFunction

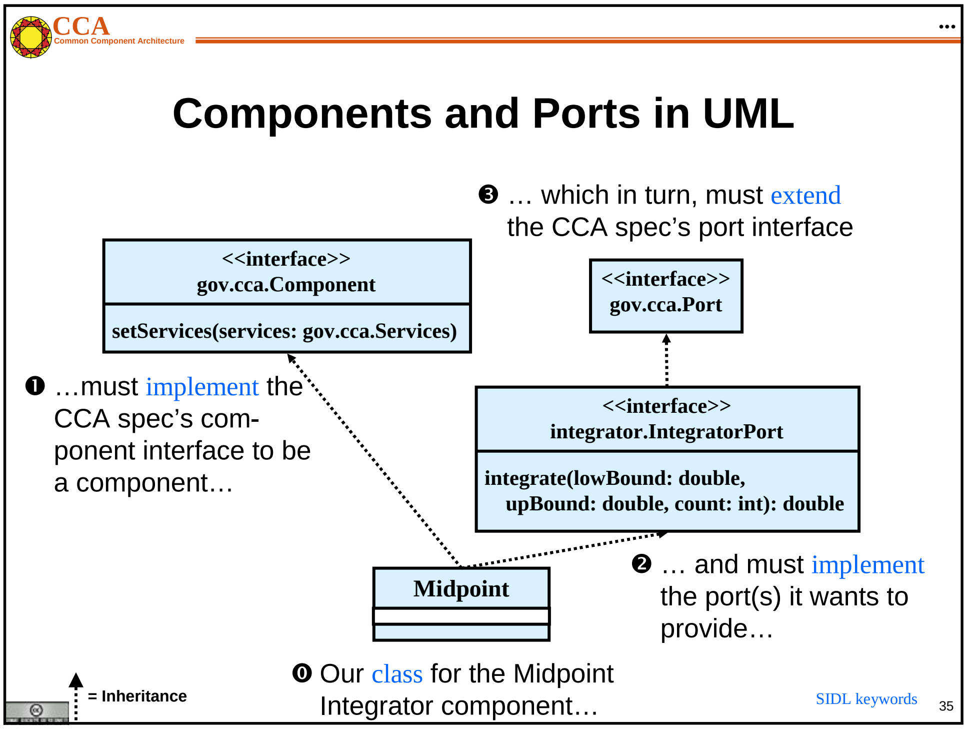
- Components interact through well-defined interfaces, or *ports*
  - A port expresses some computational functionality
  - In Fortran, a port is a bunch of subroutines or a module
  - In OO languages, a port is a abstract class or interface
- Ports and connections between them are a procedural (caller/callee) relationship, ***not dataflow!***
  - e.g., *FunctionPort* could contain: *evaluate(in Arg, out Result)*
- A single component can implement multiple ports
  - Different “views” of the same functionality
  - Related pieces of functionality
- Multiple components can implement the same port
  - Different implementations of the same functionality
  - Basis for interoperability of component software

CCA
Common Component Architecture

CCA Concepts: *Provides* Ports


IntegratorPort
FunctionPort
FunctionPort
MidpointIntegrator
NonlinearFunction

- Components may *provide* ports – implement the class or subroutines of the port (“Provides” Port)
- In SIDL, ports are interfaces that extend (inherit from) **gov.cca.Port**
  - ***gov.cca.Port*** is defined in the CCA spec and has no methods – simply allows manipulation as a port, enables plug-and-play
- In SIDL, a component that *provides* a port must **implement** (inherit from) the port’s interface
  - These are usually defined by software developers, *not* in the CCA specification.
  - ***gov.cca.ports.GoPort*** and ***gov.cca.ComponentRelease*** are two special cases – they are defined by the CCA specification, and thus have special meaning



…

Components and Ports in UML

❸ … which in turn, must extend the CCA spec's port interface

<<interface>>
gov.cca.Component

setServices(services: gov.cca.Services)

<<interface>>
gov.cca.Port

❶ …must implement the CCA spec's component interface to be a component…

<<interface>>
integrator.IntegratorPort

integrate(lowBound: double, upBound: double, count: int): double

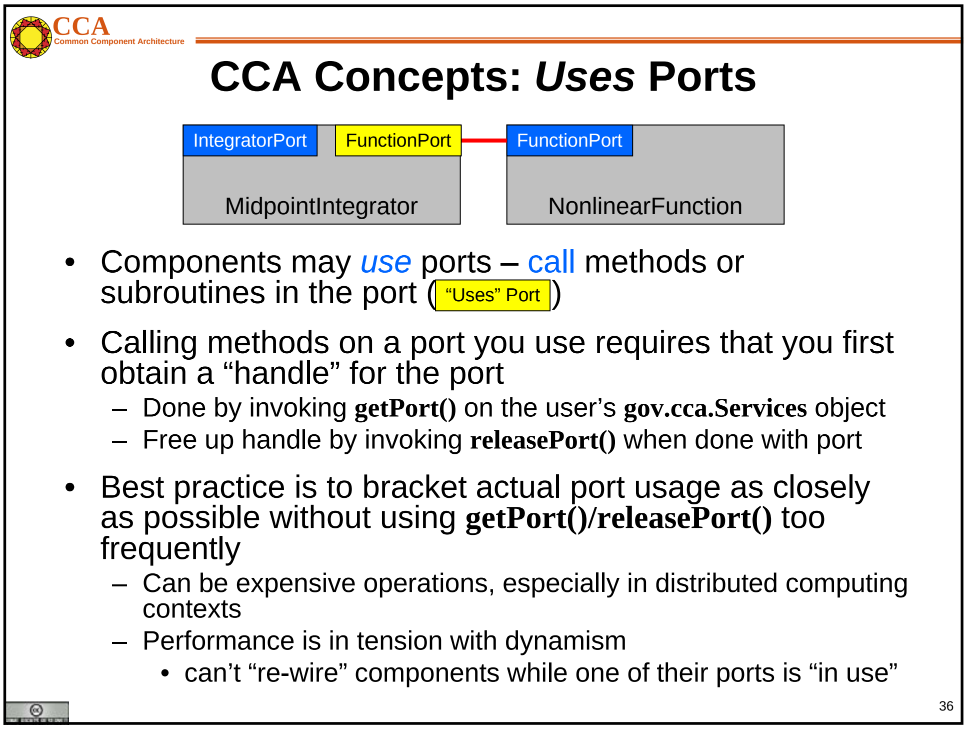
Midpoint

❷ … and must implement the port(s) it wants to provide…

= Inheritance

⓿ Our class for the Midpoint Integrator component…

SIDL keywords

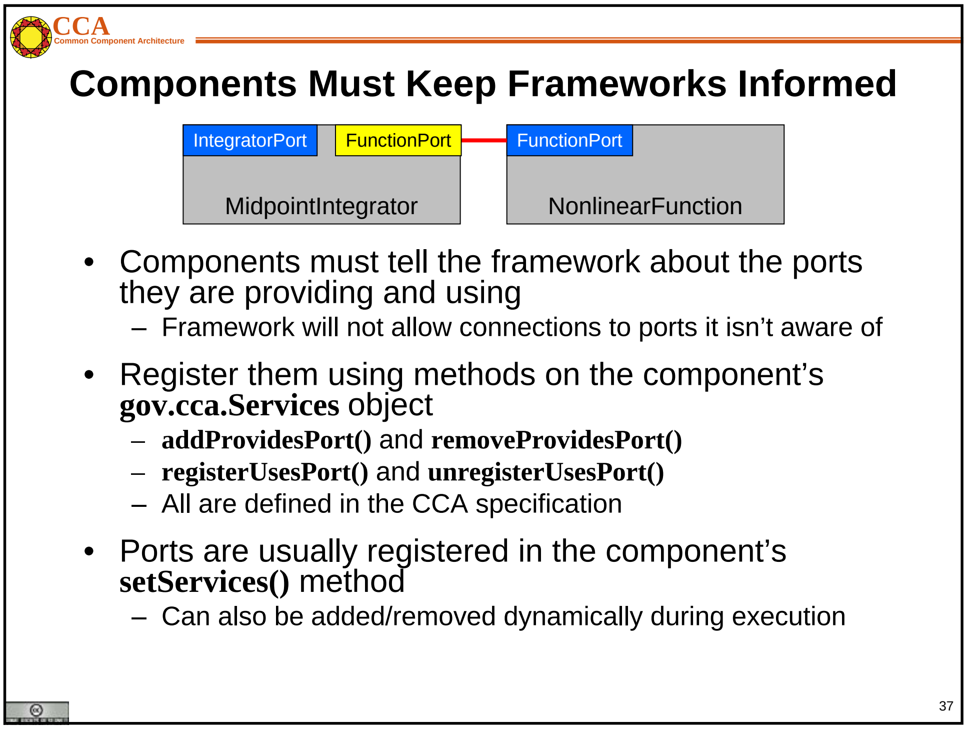


CCA Concepts: *Uses* Ports

IntegratorPort FunctionPort
MidpointIntegrator

FunctionPort
NonlinearFunction

- Components may *use* ports – call methods or subroutines in the port ("Uses" Port)
- Calling methods on a port you use requires that you first obtain a "handle" for the port
  - Done by invoking **getPort()** on the user's **gov.cca.Services** object
  - Free up handle by invoking **releasePort()** when done with port
- Best practice is to bracket actual port usage as closely as possible without using **getPort()/releasePort()** too frequently
  - Can be expensive operations, especially in distributed computing contexts
  - Performance is in tension with dynamism
    - can't "re-wire" components while one of their ports is "in use"



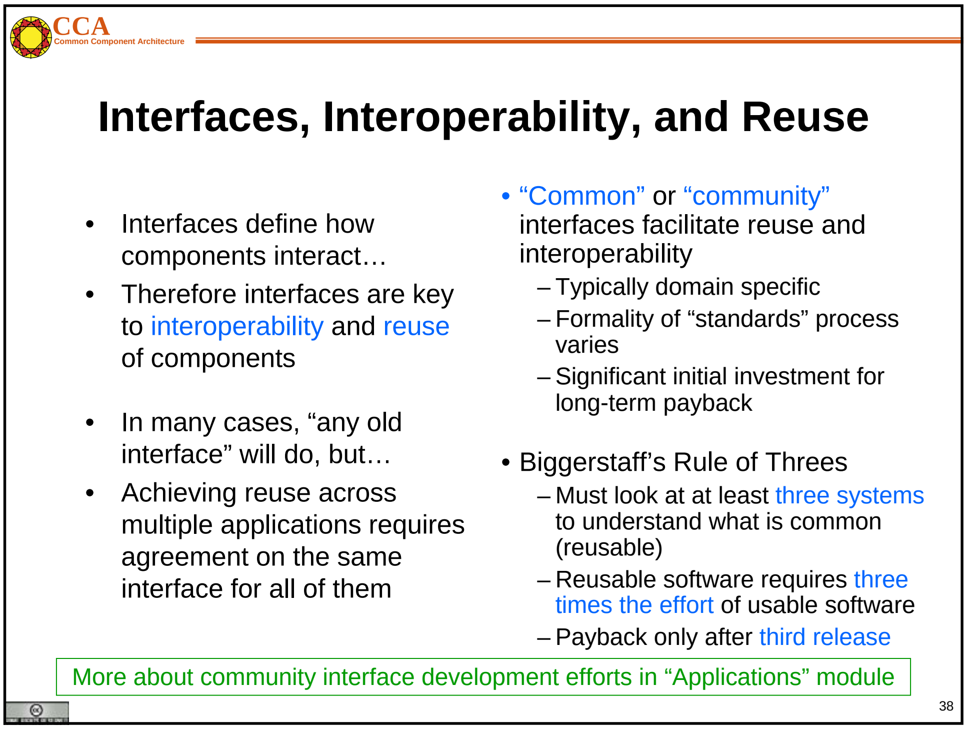
CCA
Common Component Architecture

Components Must Keep Frameworks Informed

IntegratorPort FunctionPort — FunctionPort

MidpointIntegrator NonlinearFunction

- Components must tell the framework about the ports they are providing and using
  - Framework will not allow connections to ports it isn't aware of
- Register them using methods on the component's **gov.cca.Services** object
  - **addProvidesPort()** and **removeProvidesPort()**
  - **registerUsesPort()** and **unregisterUsesPort()**
  - All are defined in the CCA specification
- Ports are usually registered in the component's **setServices()** method
  - Can also be added/removed dynamically during execution



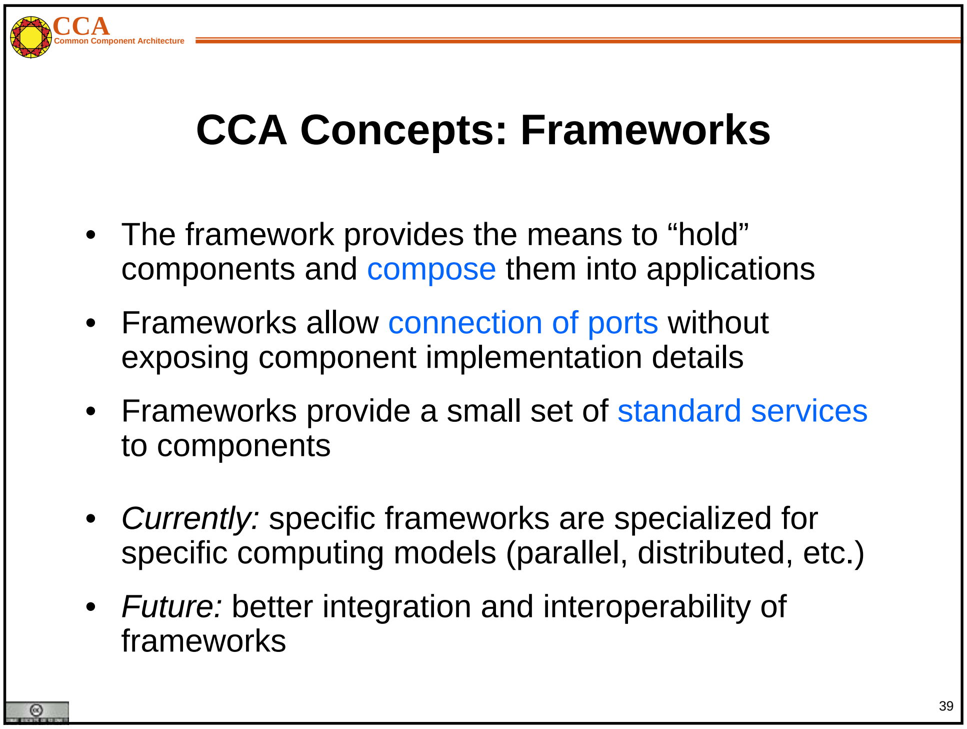
CCA
Common Component Architecture

Interfaces, Interoperability, and Reuse

- Interfaces define how components interact…
- Therefore interfaces are key to interoperability and reuse of components
- In many cases, "any old interface" will do, but…
- Achieving reuse across multiple applications requires agreement on the same interface for all of them

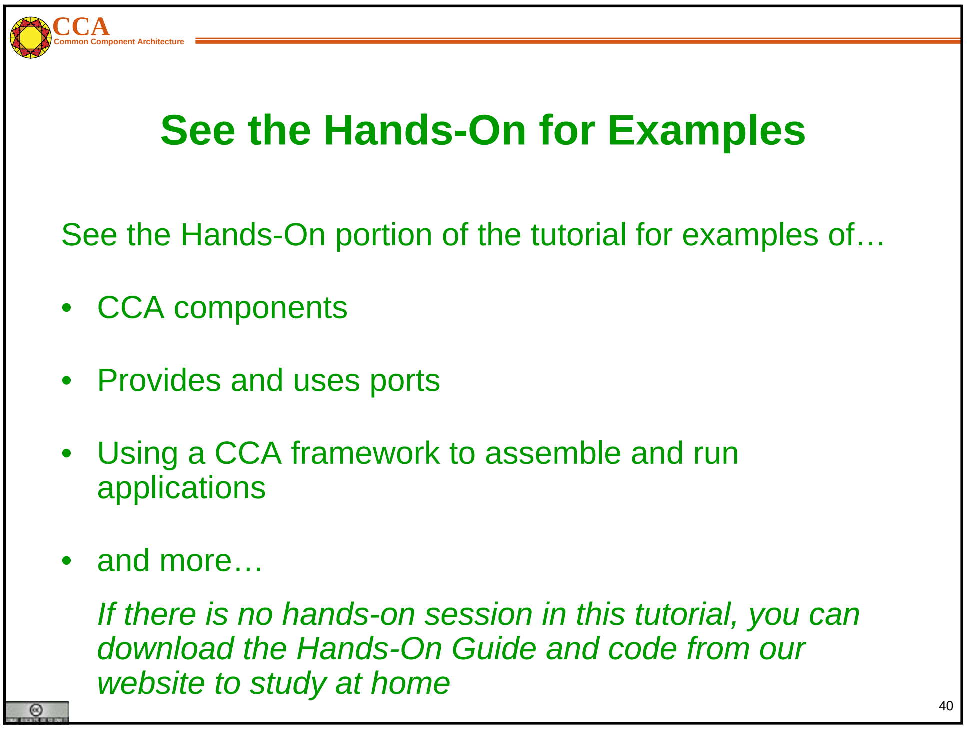
- "Common" or "community" interfaces facilitate reuse and interoperability
  - Typically domain specific
  - Formality of "standards" process varies
  - Significant initial investment for long-term payback
- Biggerstaff's Rule of Threes
  - Must look at at least three systems to understand what is common (reusable)
  - Reusable software requires three times the effort of usable software
  - Payback only after third release

More about community interface development efforts in "Applications" module



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
- *Currently:* specific frameworks are specialized for specific computing models (parallel, distributed, etc.)
- *Future:* better integration and interoperability of frameworks

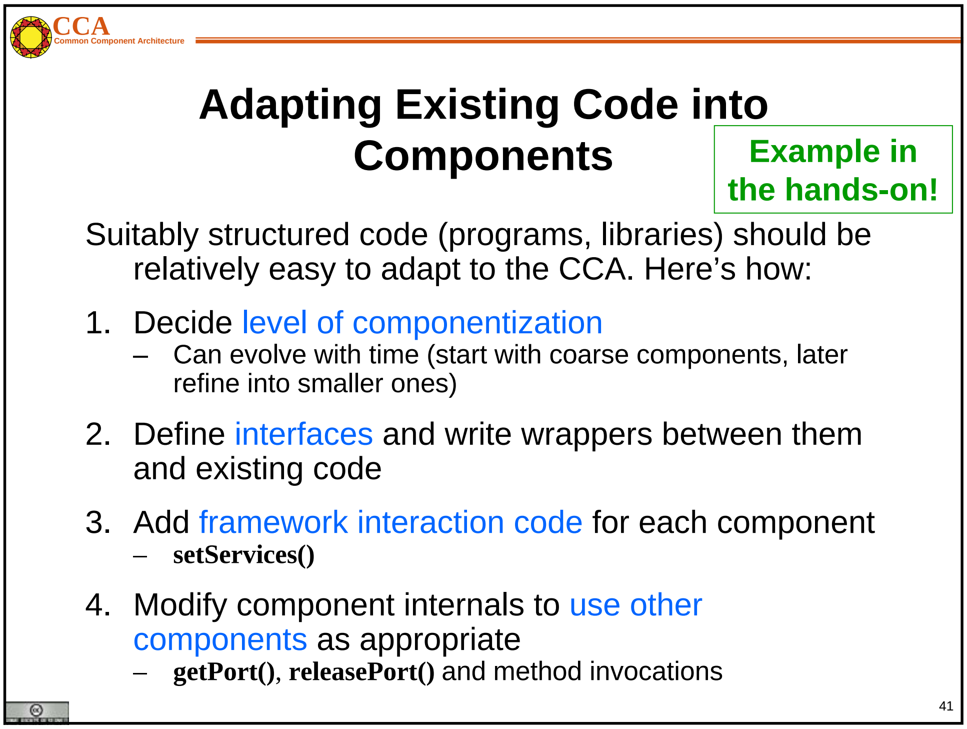


See the Hands-On for Examples

See the Hands-On portion of the tutorial for examples of…

- CCA components
- Provides and uses ports
- Using a CCA framework to assemble and run applications
- and more…

  If there is no hands-on session in this tutorial, you can download the Hands-On Guide and code from our website to study at home



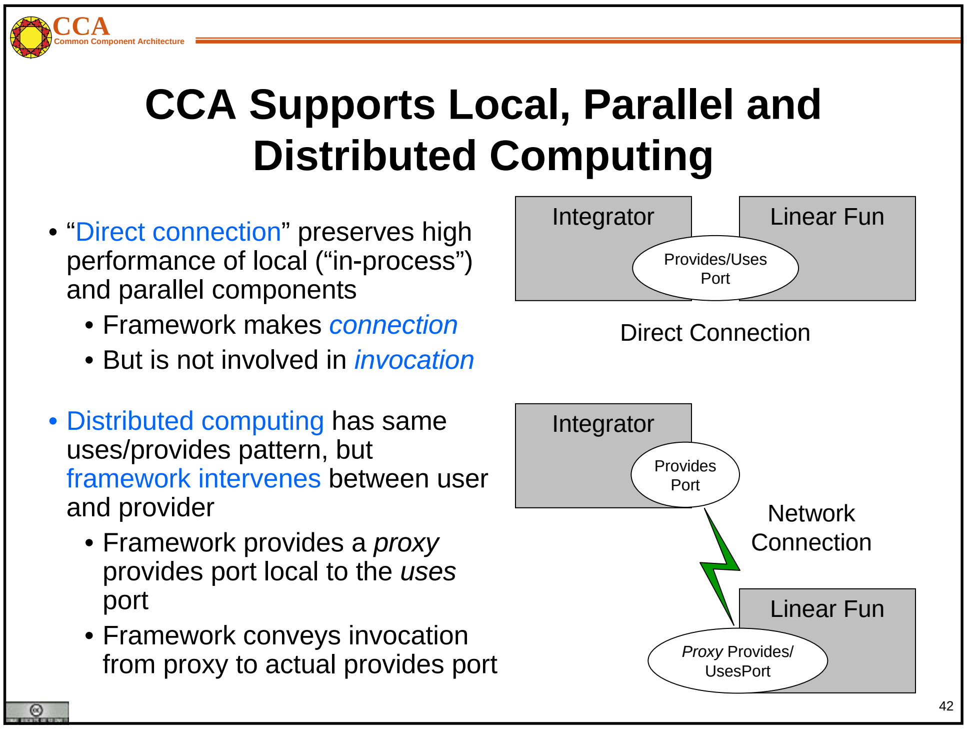
CCA Common Component Architecture

Adapting Existing Code into Components

Example in the hands-on!

Suitably structured code (programs, libraries) should be relatively easy to adapt to the CCA. Here's how:

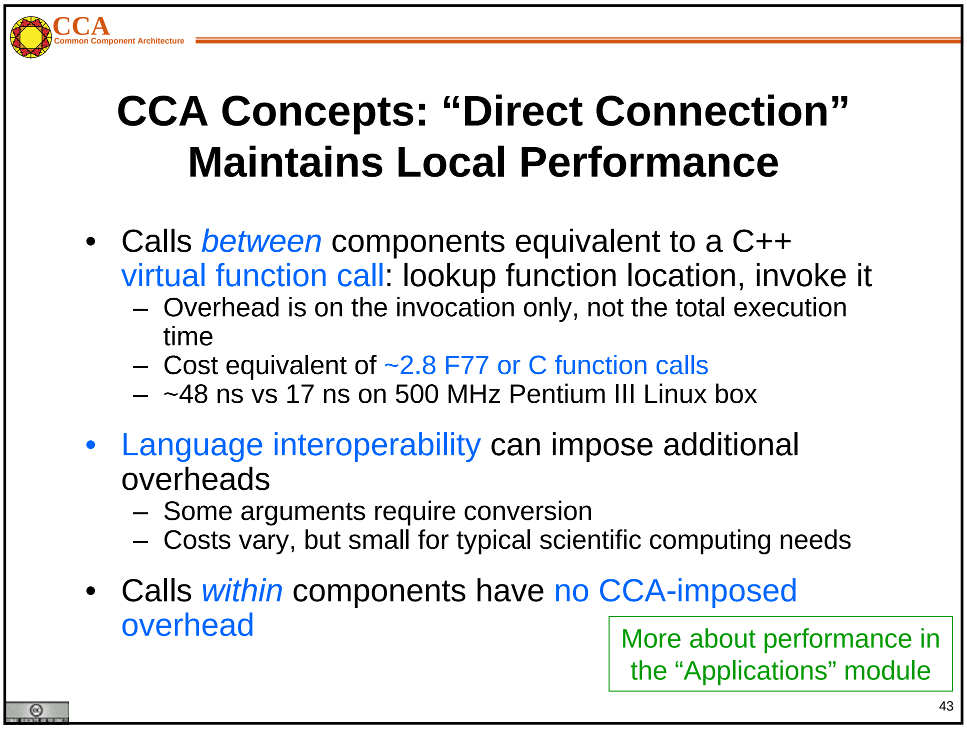
1. Decide level of componentization
   - Can evolve with time (start with coarse components, later refine into smaller ones)
2. Define interfaces and write wrappers between them and existing code
3. Add framework interaction code for each component
   - **setServices()**
4. Modify component internals to use other components as appropriate
   - **getPort()**, **releasePort()** and method invocations



CCA Common Component Architecture

CCA Supports Local, Parallel and Distributed Computing

- "Direct connection" preserves high performance of local ("in-process") and parallel components
  - Framework makes *connection*
  - But is not involved in *invocation*
- Distributed computing has same uses/provides pattern, but framework intervenes between user and provider
  - Framework provides a *proxy* provides port local to the *uses* port
  - Framework conveys invocation from proxy to actual provides port

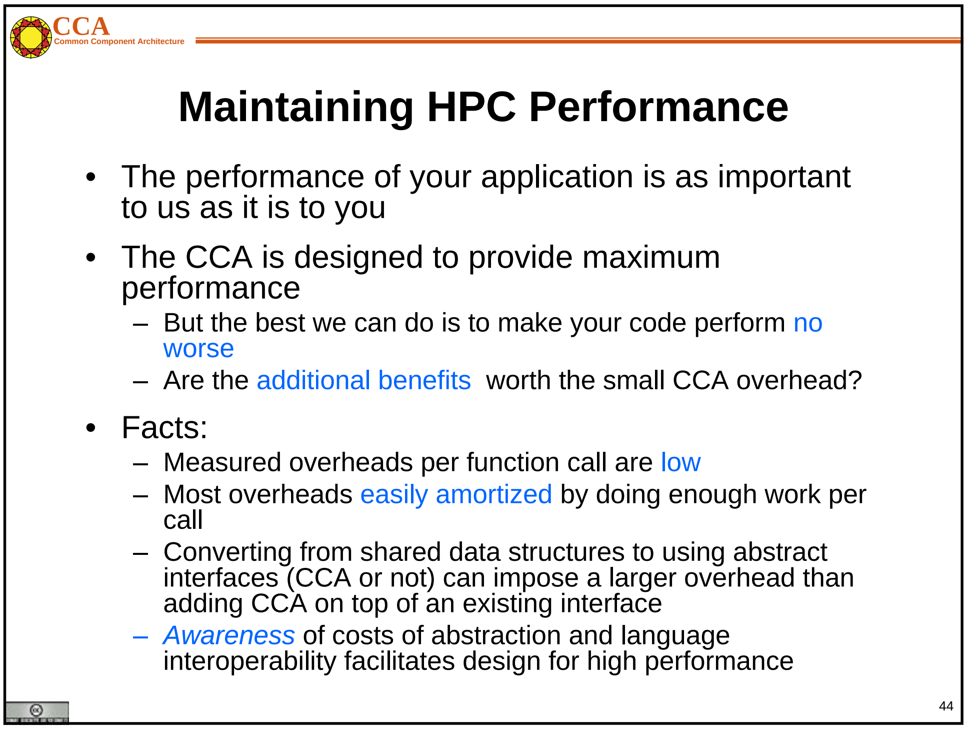


CCA
Common Component Architecture

CCA Concepts: "Direct Connection" Maintains Local Performance

- Calls *between* components equivalent to a C++ virtual function call: lookup function location, invoke it
  - Overhead is on the invocation only, not the total execution time
  - 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

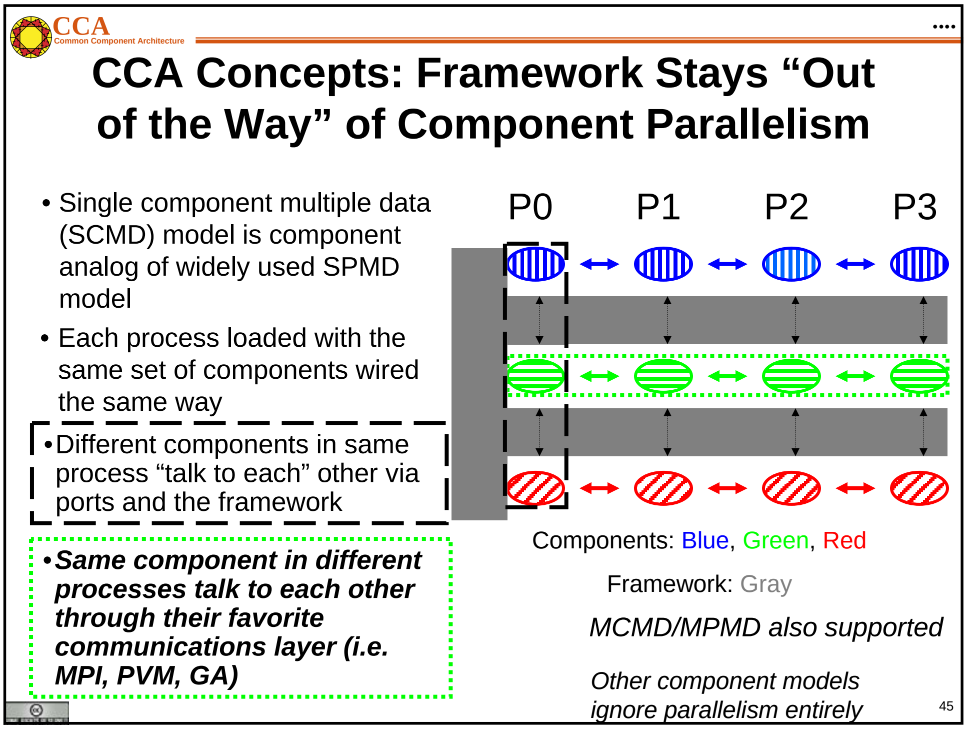
More about performance in the "Applications" module



CCA
Common Component Architecture

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
  - Are the additional benefits worth the small CCA overhead?
- Facts:
  - Measured overheads per function call are low
  - Most overheads easily amortized by doing enough work per call
  - Converting from shared data structures to using abstract interfaces (CCA or not) can impose a larger overhead than adding CCA on top of an existing interface
  - *Awareness* of costs of abstraction and language interoperability facilitates design for high performance



CCA Concepts: Framework Stays "Out of the Way" of Component Parallelism

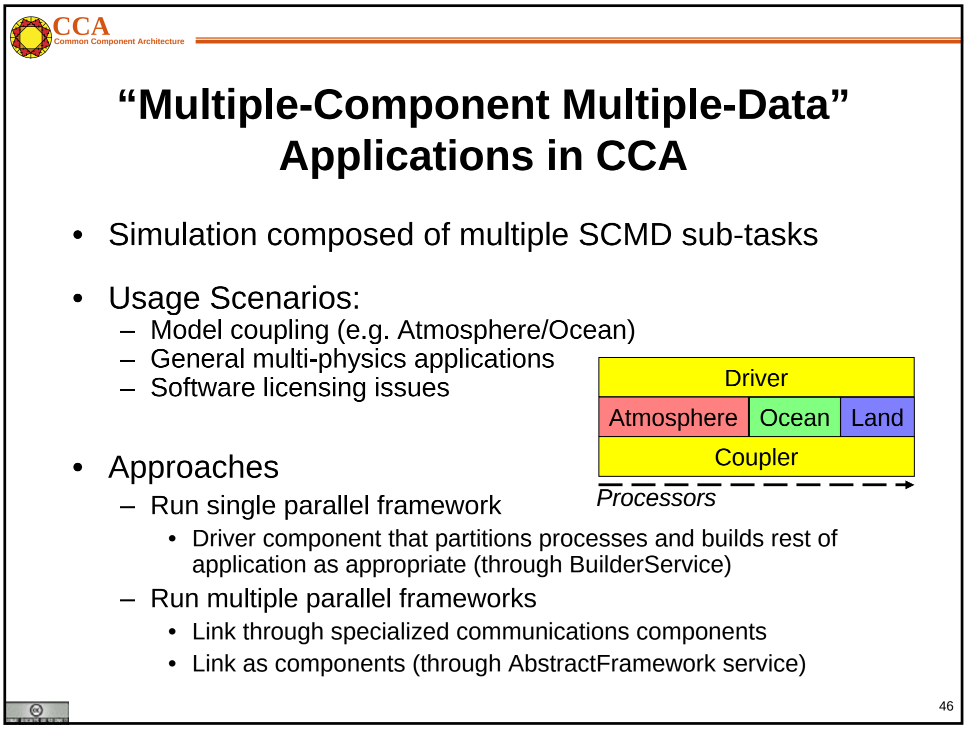
- Single component multiple data (SCMD) model is component analog of widely used SPMD model
- Each process loaded with the same set of components wired the same way
- Different components in same process "talk to each" other via ports and the framework
- ***Same component in different processes talk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

Components: Blue, Green, Red

Framework: Gray

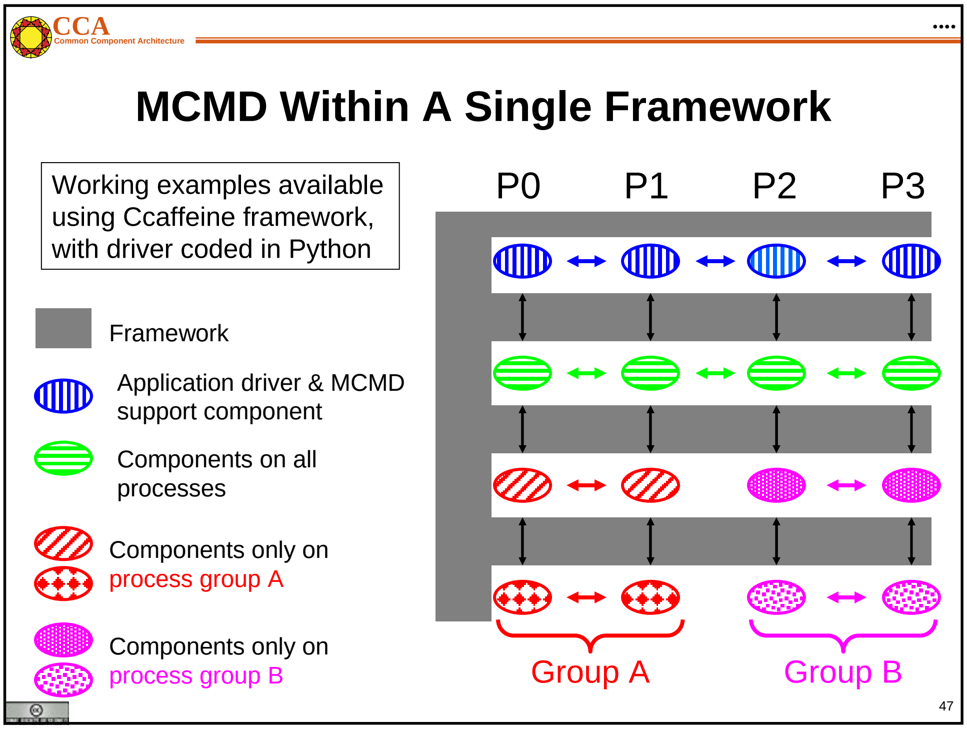
MCMD/MPMD also supported

Other component models ignore parallelism entirely



"Multiple-Component Multiple-Data" Applications in CCA

- Simulation composed of multiple SCMD sub-tasks
- Usage Scenarios:
  - Model coupling (e.g. Atmosphere/Ocean)
  - General multi-physics applications
  - Software licensing issues
- Approaches
  - Run single parallel framework
    - Driver component that partitions processes and builds rest of application as appropriate (through BuilderService)
  - Run multiple parallel frameworks
    - Link through specialized communications components
    - Link as components (through AbstractFramework service)



MCMD Within A Single Framework

Working examples available using Ccaffeine framework, with driver coded in Python

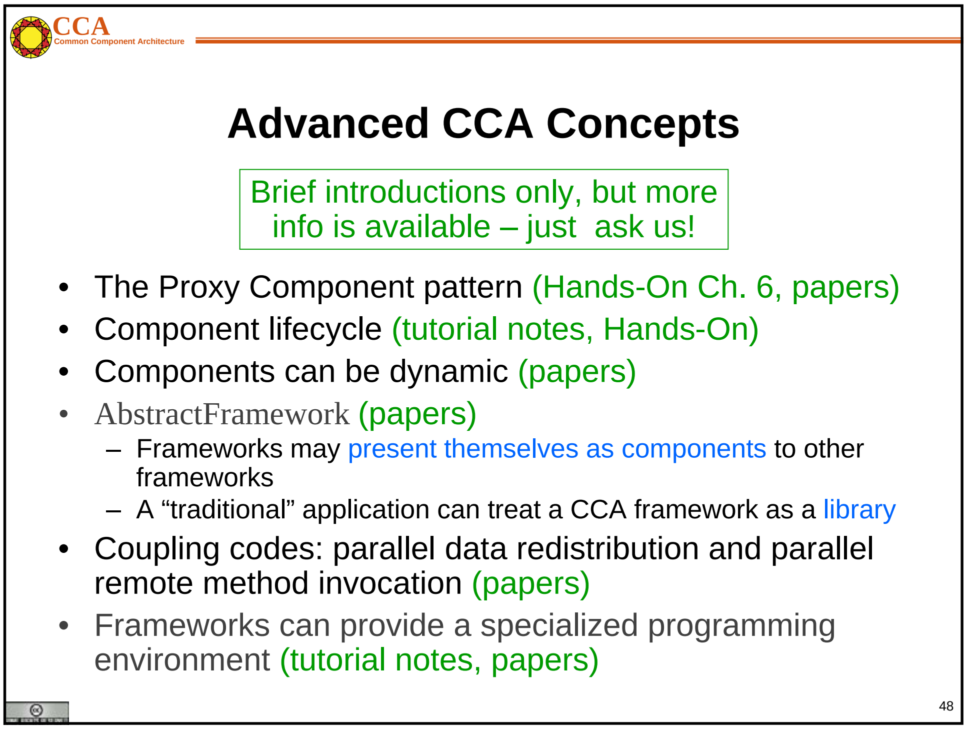
▩ Framework

▥ Application driver & MCMD support component

▤ Components on all processes

▨ Components only on process group A

▦ Components only on process group B

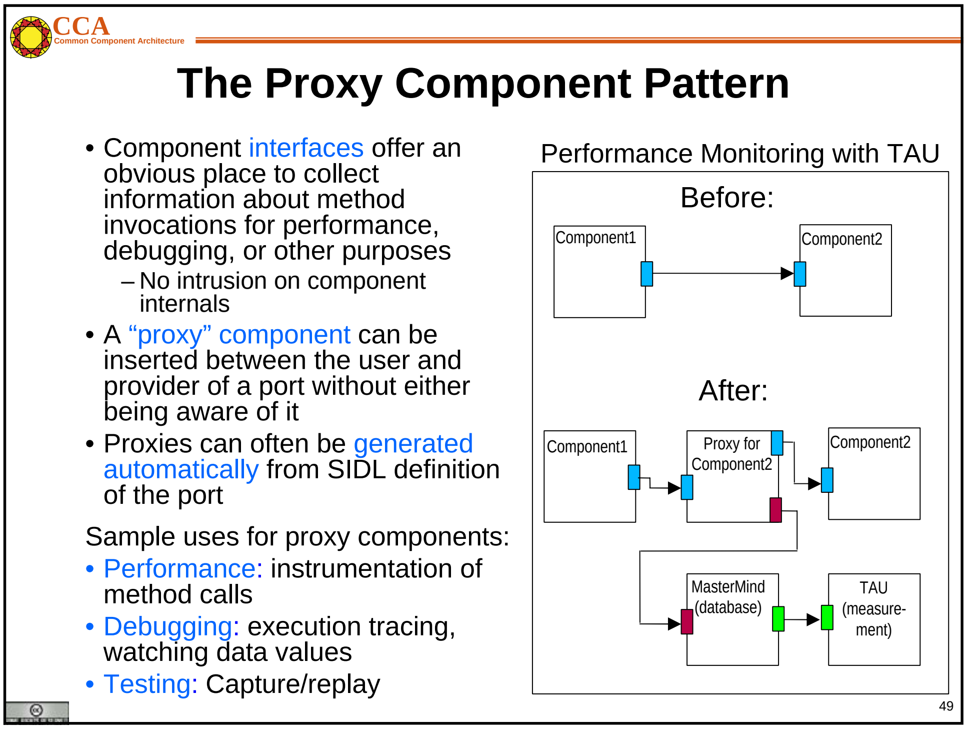


CCA
Common Component Architecture

Advanced CCA Concepts

Brief introductions only, but more info is available – just ask us!

- The Proxy Component pattern (Hands-On Ch. 6, papers)
- Component lifecycle (tutorial notes, Hands-On)
- Components can be dynamic (papers)
- AbstractFramework (papers)
  - Frameworks may present themselves as components to other frameworks
  - A “traditional” application can treat a CCA framework as a library
- Coupling codes: parallel data redistribution and parallel remote method invocation (papers)
- Frameworks can provide a specialized programming environment (tutorial notes, papers)



CCA Common Component Architecture

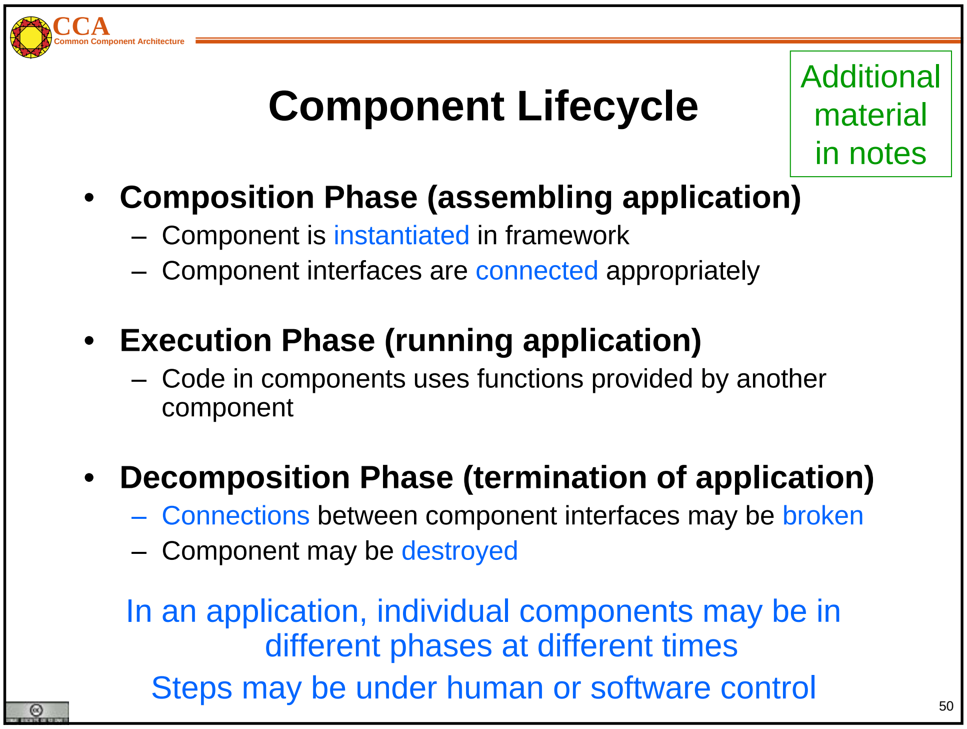
The Proxy Component Pattern

- Component interfaces offer an obvious place to collect information about method invocations for performance, debugging, or other purposes
  - No intrusion on component internals
- A "proxy" component can be inserted between the user and provider of a port without either being aware of it
- Proxies can often be generated automatically from SIDL definition of the port

Sample uses for proxy components:

- Performance: instrumentation of method calls
- Debugging: execution tracing, watching data values
- Testing: Capture/replay

Performance Monitoring with TAU



CCA Common Component Architecture

Component Lifecycle

Additional material in notes

- **Composition Phase (assembling application)**
  - Component is instantiated in framework
  - Component interfaces are connected appropriately
- **Execution Phase (running application)**
  - Code in components uses functions provided by another component
- **Decomposition Phase (termination of application)**
  - Connections between component interfaces may be broken
  - Component may be destroyed

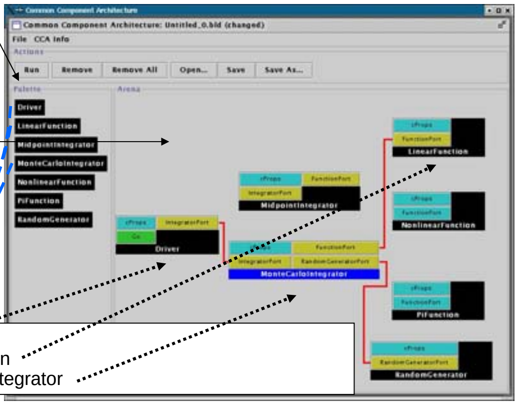
In an application, individual components may be in different phases at different times

Steps may be under human or software control

User Viewpoint: Loading and Instantiating Components

- Components are code + metadata
- Using metadata, a **Palette** of available components is constructed
- Components are instantiated by user action (i.e. by dragging from **Palette** into **Arena**)
- Framework calls component's constructor, then setServices

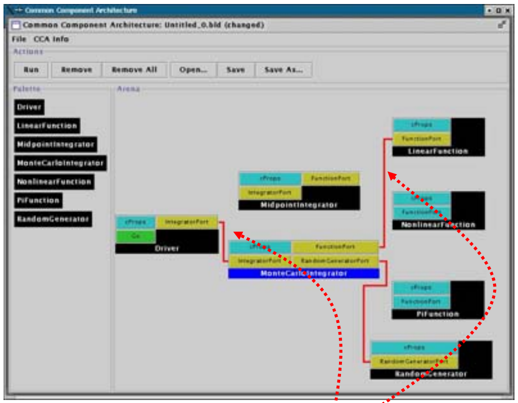
- Details are framework-specific!
- Ccaffeine currently provides both command line and GUI approaches



```
create     Driver                   Driver
create     LinearFunction           LinearFunction
create     MonteCarloIntegrator     MonteCarloIntegrator
```

User Connects Ports

- Can only connect uses & provides
  - Not uses/uses or provides/provides
- Ports connected by type, not name
  - Port names must be unique within component
  - Types must match across components
- Framework puts info about *provider* of port into *using component's* Services object

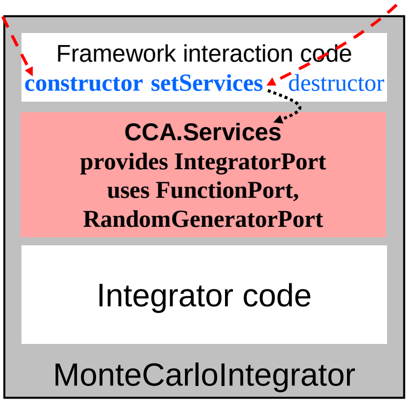


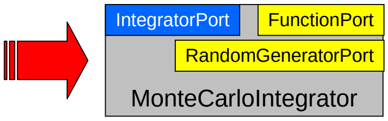
```
connect    Driver                  IntegratorPort    MonteCarloIntegrator    IntegratorPort
connect    MonteCarloIntegrator    FunctionPort      LinearFunction          FunctionPort
…
```

Component's View of Instantiation

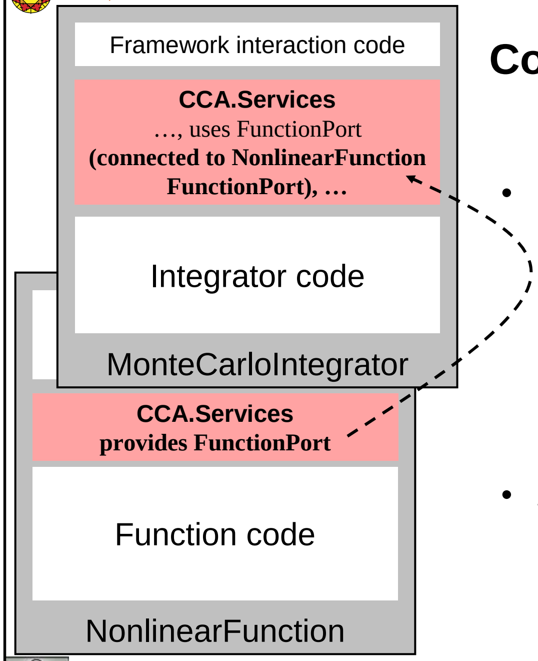
- Framework calls component's constructor
- Component initializes internal data, etc.
  - Knows *nothing* outside itself

- Framework calls component's setServices
  - Passes setServices an object representing everything "outside"
  - setServices declares ports component *uses* and *provides*
- Component *still* knows nothing outside itself
  - But Services object provides the means of communication w/ framework
- Framework now knows how to "decorate" component and how it might connect with others

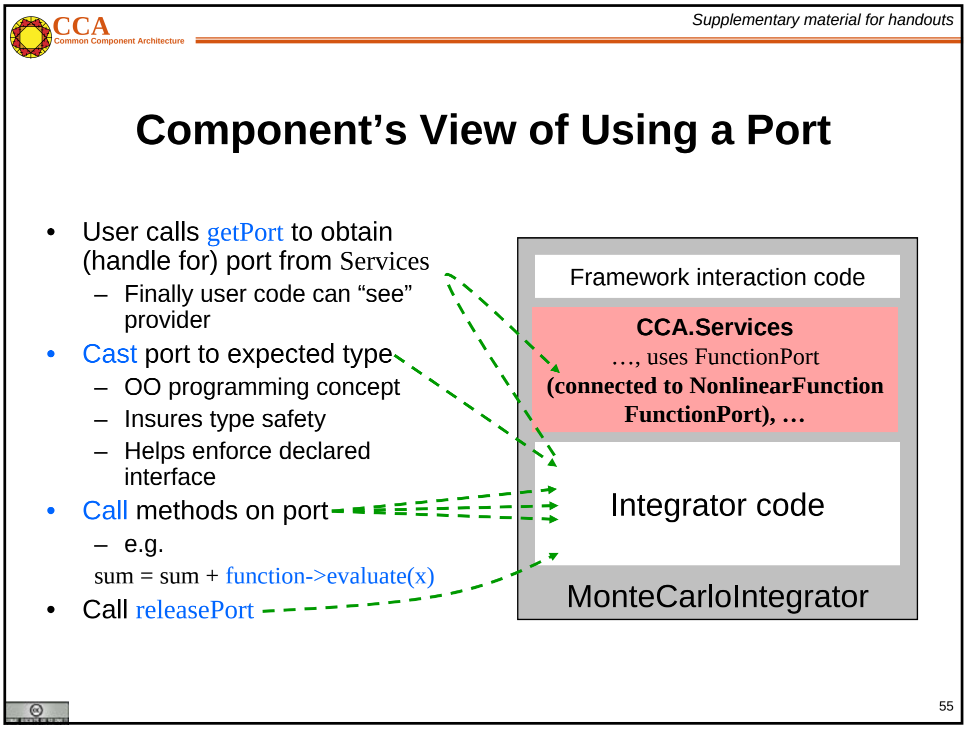

Framework interaction code
constructor setServices destructor
CCA.Services
provides IntegratorPort
uses FunctionPort,
RandomGeneratorPort
Integrator code
MonteCarloIntegrator



Component's View of Connection


Framework interaction code
CCA.Services
…, uses FunctionPort
(connected to NonlinearFunction FunctionPort), …
Integrator code
MonteCarloIntegrator
CCA.Services
provides FunctionPort
Function code
NonlinearFunction

- Framework puts info about provider into user component's Services object
  - ***MonteCarloIntegrator***'s Services object is aware of connection
  - ***NonlinearFunction*** is not!
- ***MCI***'s integrator code cannot yet call functions on FunctionPort



CCA Common Component Architecture

Supplementary material for handouts

Component's View of Using a Port

- User calls getPort to obtain (handle for) port from Services
  - Finally user code can "see" provider
- Cast port to expected type
  - OO programming concept
  - Insures type safety
  - Helps enforce declared interface
- Call methods on port
  - e.g.

  sum = sum + function->evaluate(x)
- Call releasePort

Framework interaction code

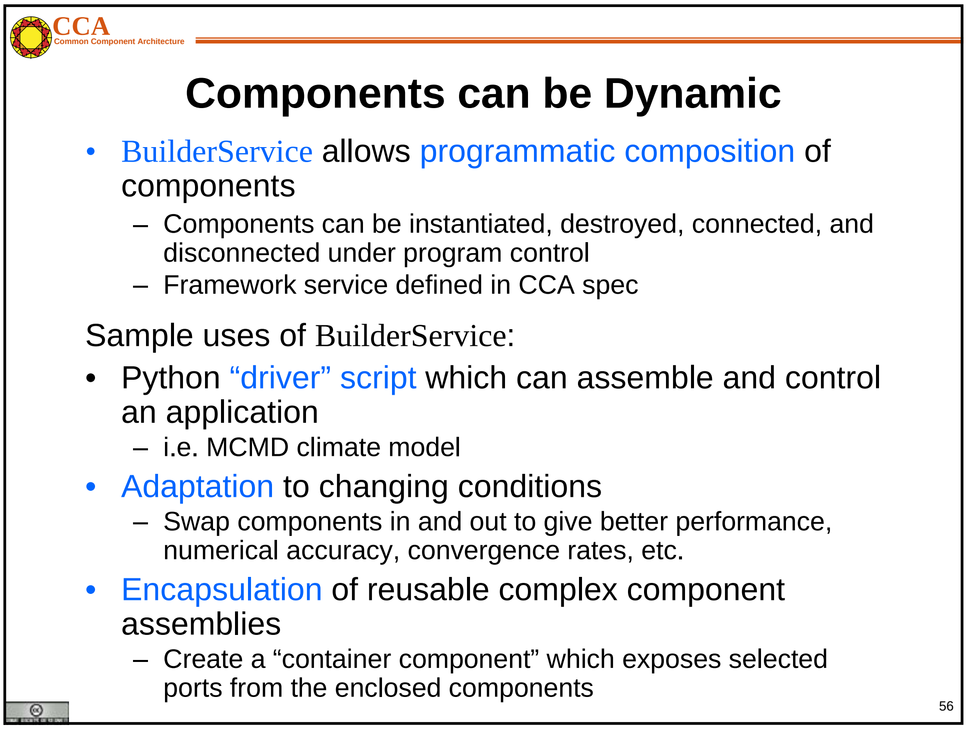
CCA.Services

…, uses FunctionPort

(connected to NonlinearFunction FunctionPort), …

Integrator code

MonteCarloIntegrator



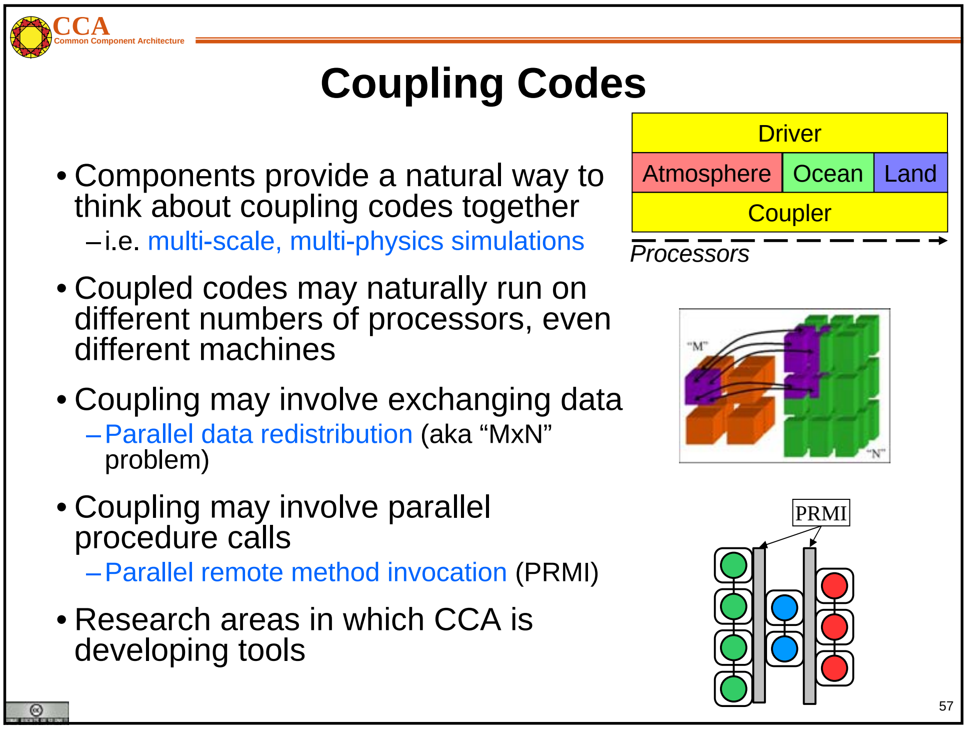
CCA Common Component Architecture

Components can be Dynamic

- BuilderService allows programmatic composition of components
  - Components can be instantiated, destroyed, connected, and disconnected under program control
  - Framework service defined in CCA spec

Sample uses of BuilderService:

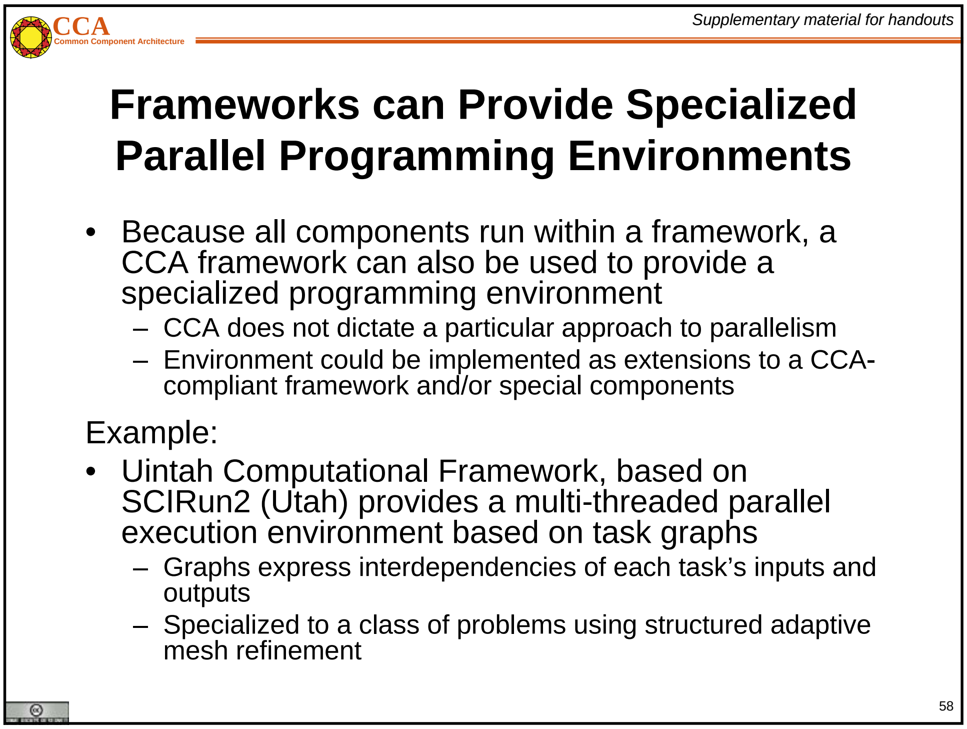
- Python "driver" script which can assemble and control an application
  - i.e. MCMD climate model
- Adaptation to changing conditions
  - Swap components in and out to give better performance, numerical accuracy, convergence rates, etc.
- Encapsulation of reusable complex component assemblies
  - Create a "container component" which exposes selected ports from the enclosed components



CCA
Common Component Architecture

Coupling Codes

- Components provide a natural way to think about coupling codes together
  - i.e. multi-scale, multi-physics simulations
- Coupled codes may naturally run on different numbers of processors, even different machines
- Coupling may involve exchanging data
  - Parallel data redistribution (aka “MxN” problem)
- Coupling may involve parallel procedure calls
  - Parallel remote method invocation (PRMI)
- Research areas in which CCA is developing tools



CCA
Common Component Architecture

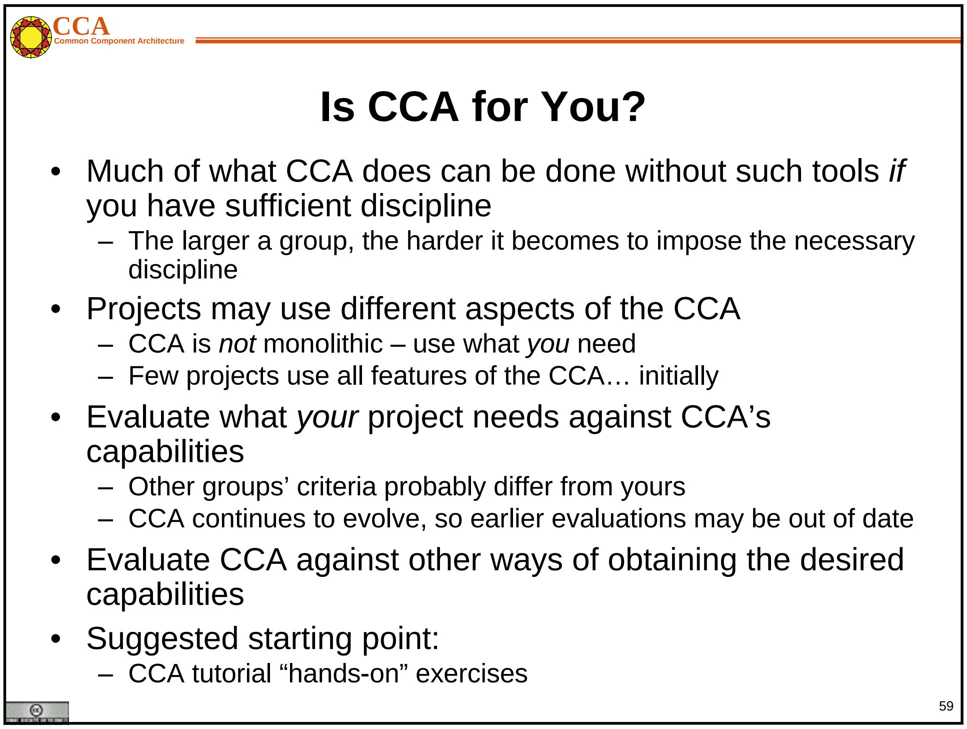
Supplementary material for handouts

Frameworks can Provide Specialized Parallel Programming Environments

- Because all components run within a framework, a CCA framework can also be used to provide a specialized programming environment
  - CCA does not dictate a particular approach to parallelism
  - Environment could be implemented as extensions to a CCA-compliant framework and/or special components

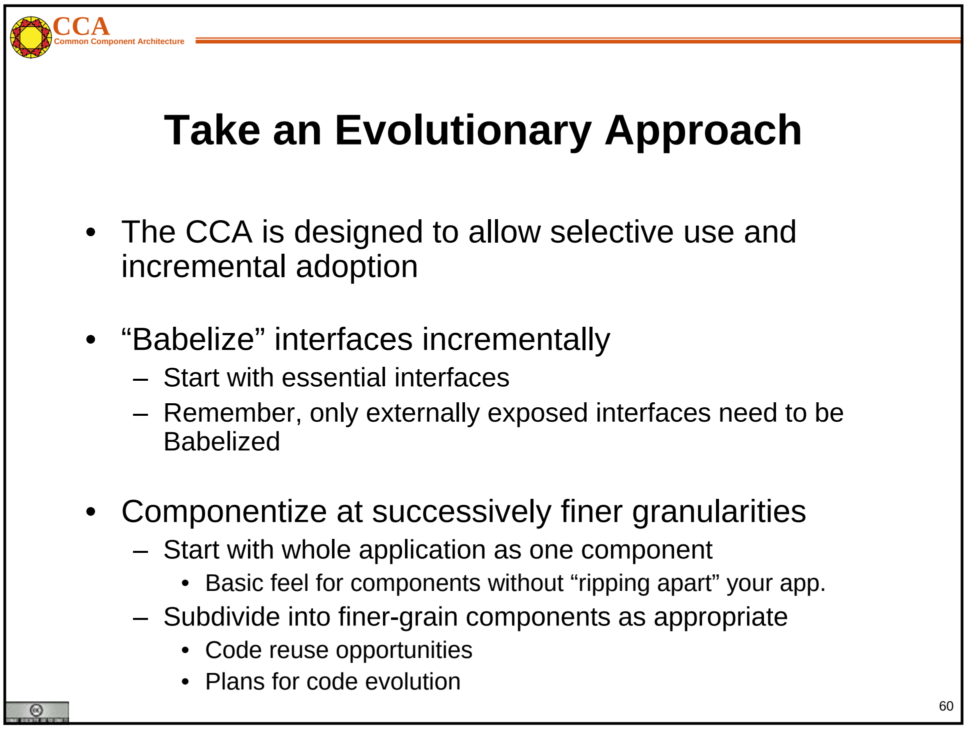
Example:

- Uintah Computational Framework, based on SCIRun2 (Utah) provides a multi-threaded parallel execution environment based on task graphs
  - Graphs express interdependencies of each task’s inputs and outputs
  - Specialized to a class of problems using structured adaptive mesh refinement



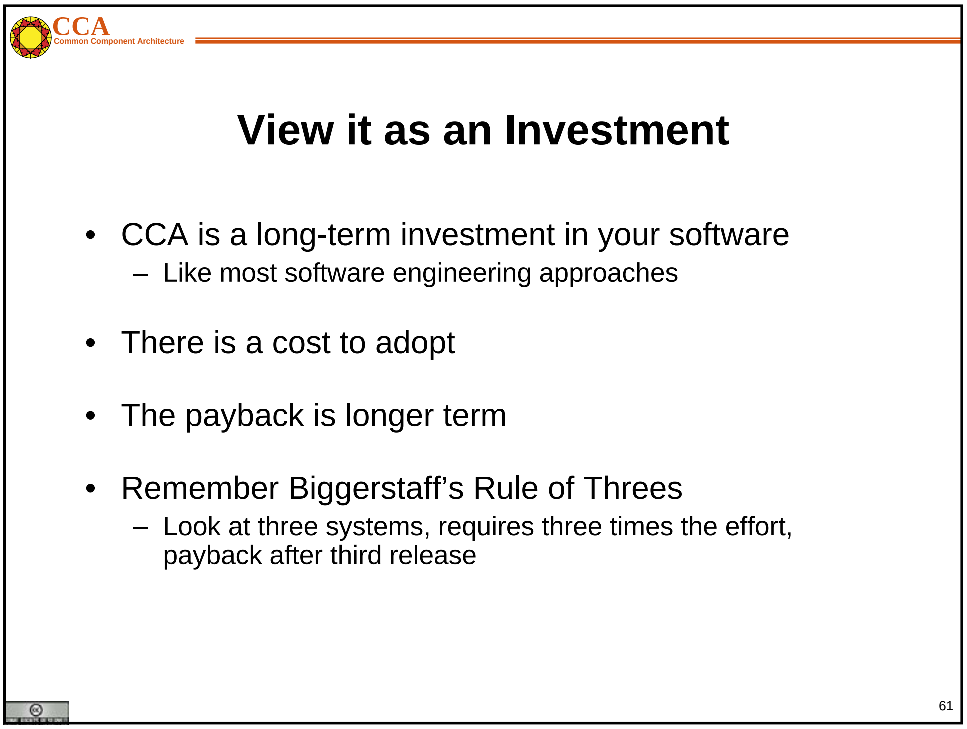
Is CCA for You?

- 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
  - Other groups' criteria probably differ from yours
  - CCA continues to evolve, so earlier evaluations may be out of date
- Evaluate CCA against other ways of obtaining the desired capabilities
- Suggested starting point:
  - CCA tutorial "hands-on" exercises



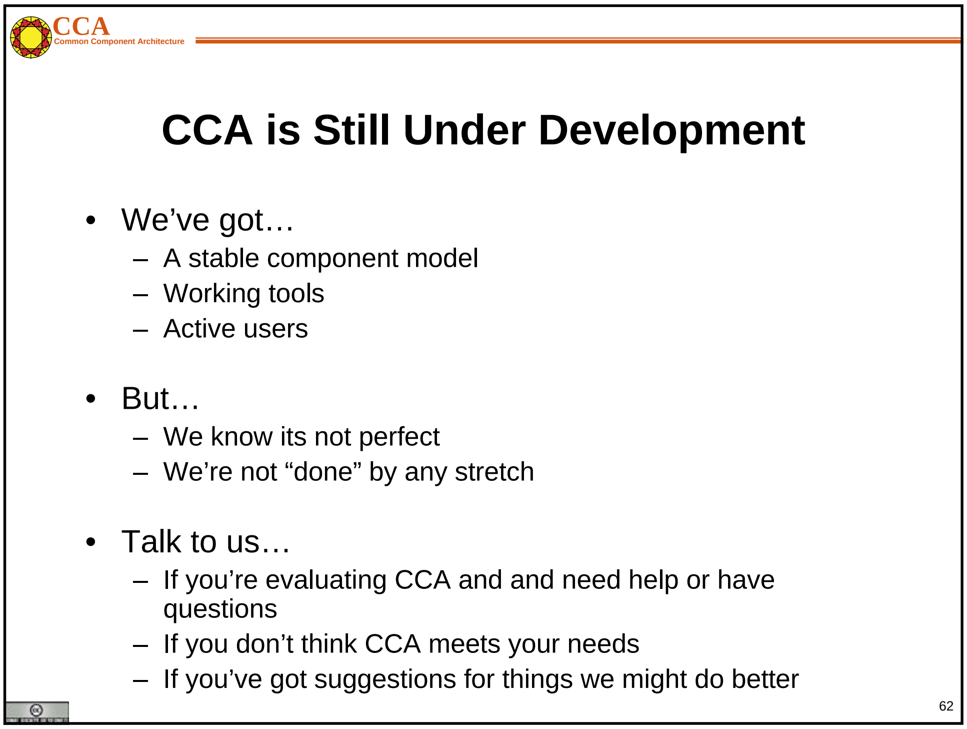
Take an Evolutionary Approach

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



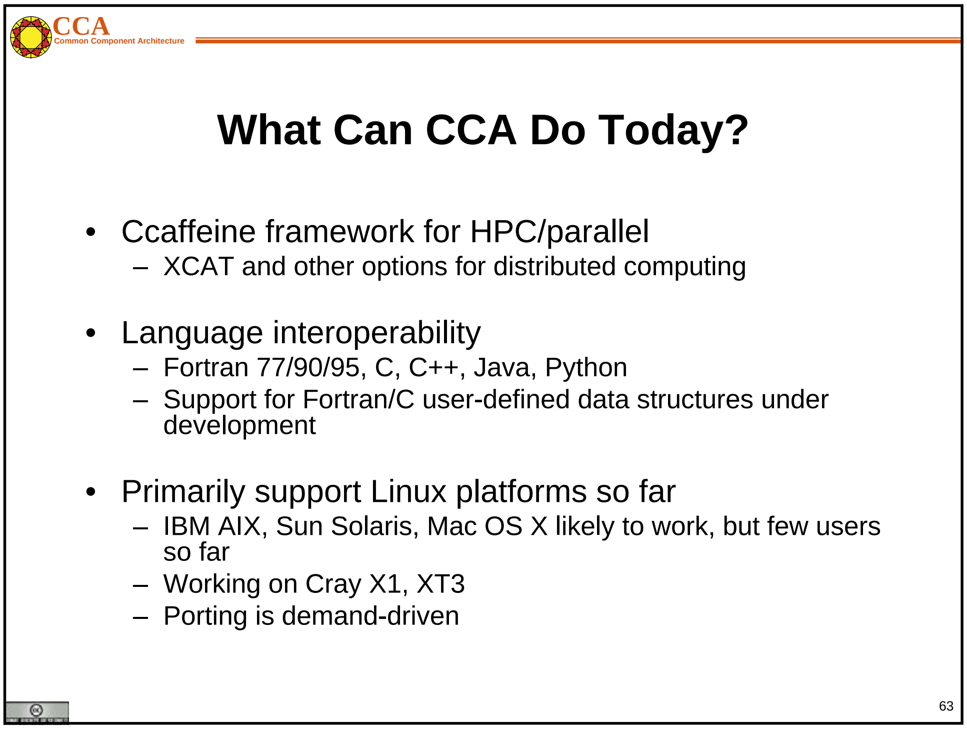
View it as an Investment

- 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
- Remember Biggerstaff's Rule of Threes
  - Look at three systems, requires three times the effort, payback after third release



CCA is Still Under Development

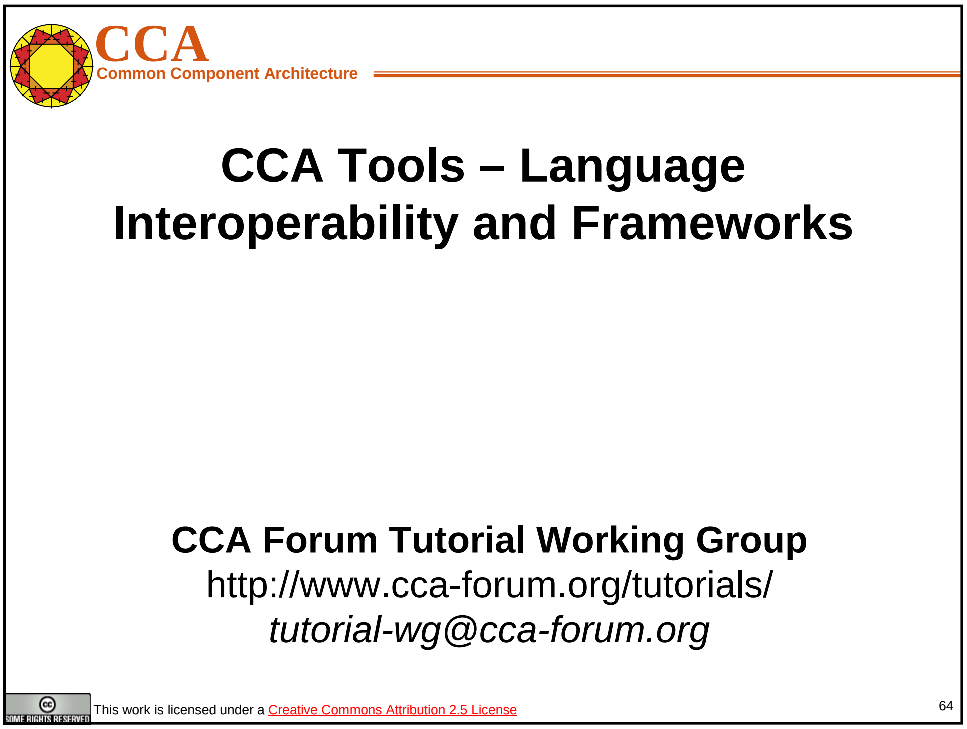
- We've got…
  - A stable component model
  - Working tools
  - Active users
- But…
  - We know its not perfect
  - We're not "done" by any stretch
- Talk to us…
  - If you're evaluating CCA and and need help or have questions
  - If you don't think CCA meets your needs
  - If you've got suggestions for things we might do better



CCA
Common Component Architecture

What Can CCA Do Today?

- Ccaffeine framework for HPC/parallel
  - XCAT and other options for distributed computing
- Language interoperability
  - Fortran 77/90/95, C, C++, Java, Python
  - Support for Fortran/C user-defined data structures under development
- Primarily support Linux platforms so far
  - IBM AIX, Sun Solaris, Mac OS X likely to work, but few users so far
  - Working on Cray X1, XT3
  - Porting is demand-driven

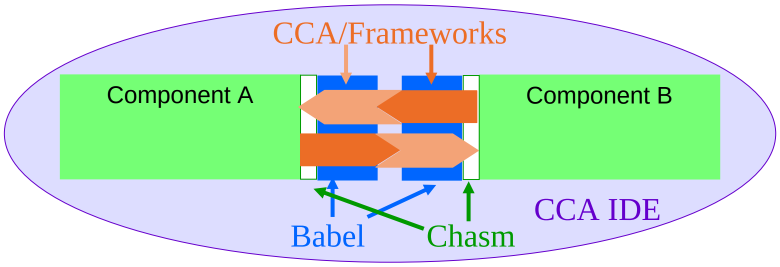


CCA
Common Component Architecture

CCA Tools – Language Interoperability and Frameworks

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org

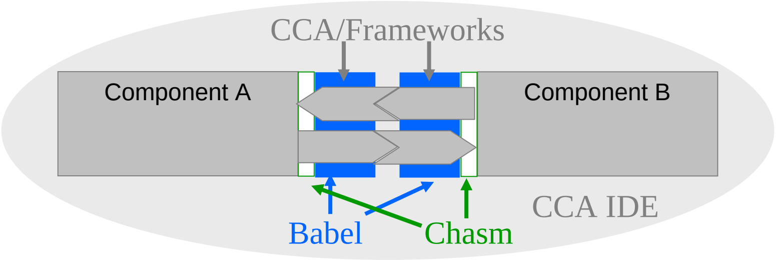
Goal of This Module



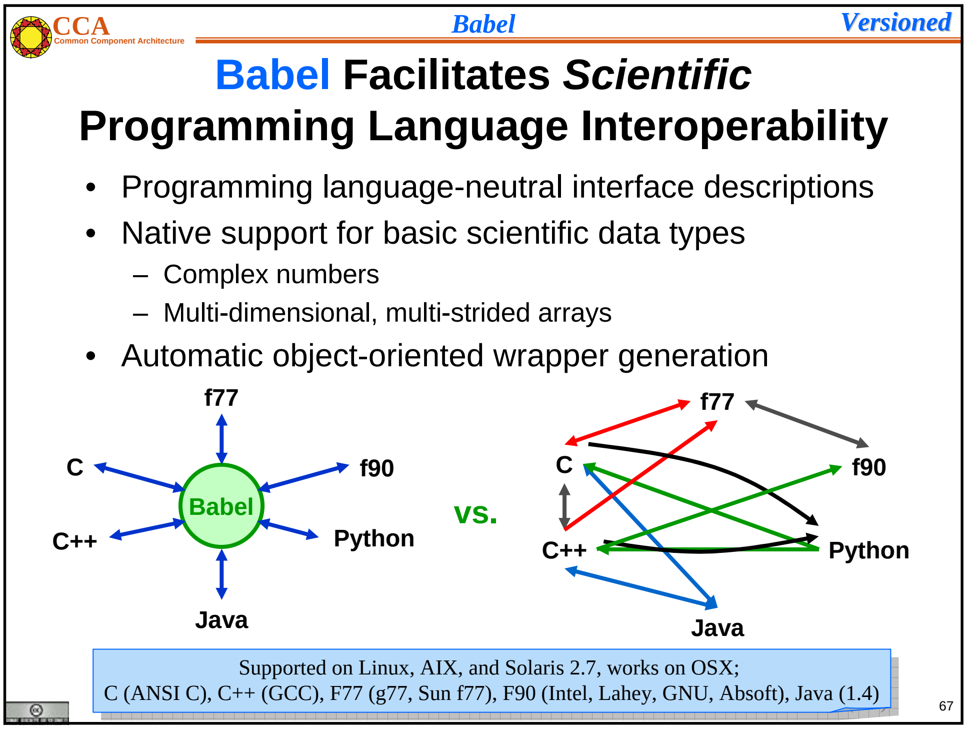
- Describe tools for multi-lingual, scientific component ‘plug-and-play’

IDE = Interactive Development Environment

Tools Module Overview



- Language interoperability tools
- Frameworks
- CCA Interactive Development Environment

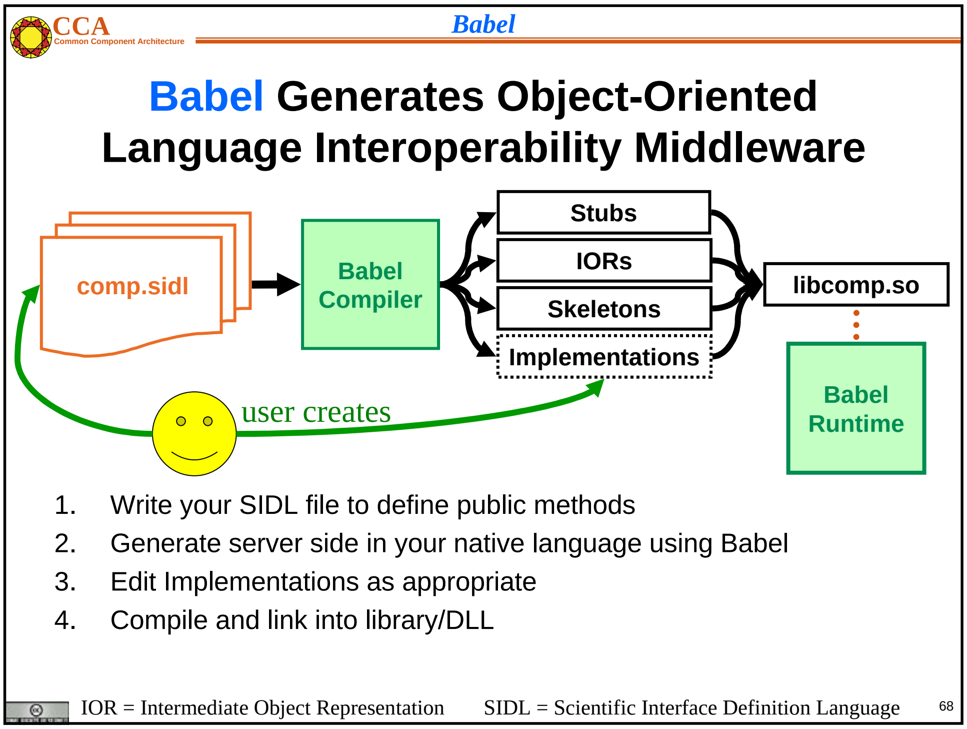


CCA Common Component Architecture — *Babel* — *Versioned*

Babel Facilitates *Scientific* Programming Language Interoperability

- Programming language-neutral interface descriptions
- Native support for basic scientific data types
  - Complex numbers
  - Multi-dimensional, multi-strided arrays
- Automatic object-oriented wrapper generation

Supported on Linux, AIX, and Solaris 2.7, works on OSX;
C (ANSI C), C++ (GCC), F77 (g77, Sun f77), F90 (Intel, Lahey, GNU, Absoft), Java (1.4)

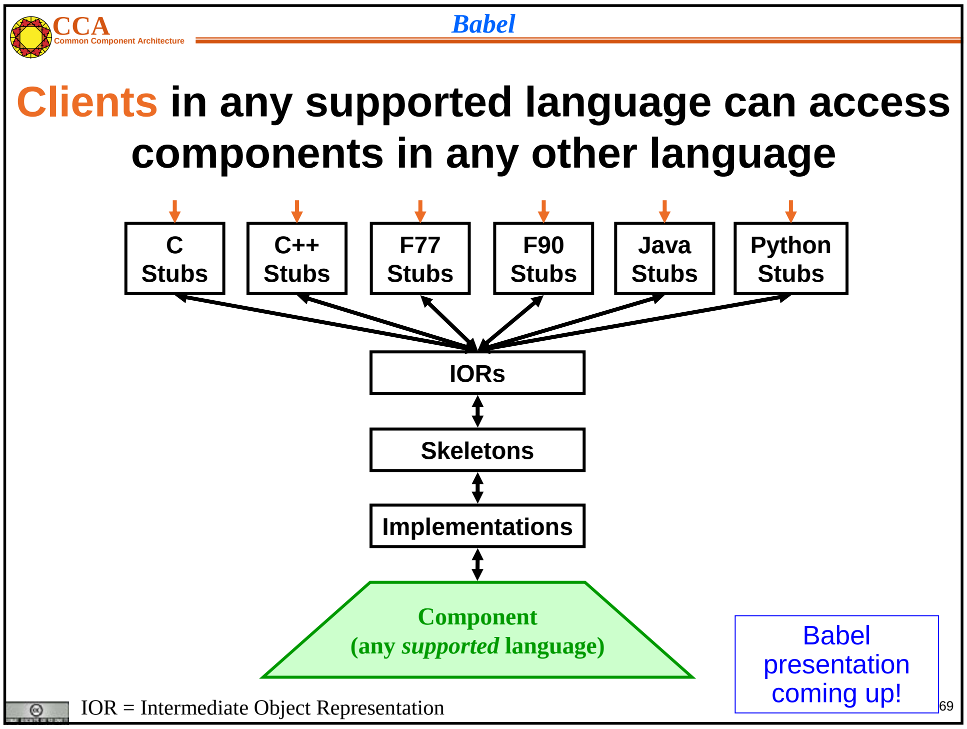


CCA Common Component Architecture — *Babel*

Babel Generates Object-Oriented Language Interoperability Middleware

1. Write your SIDL file to define public methods
2. Generate server side in your native language using Babel
3. Edit Implementations as appropriate
4. Compile and link into library/DLL

IOR = Intermediate Object Representation SIDL = Scientific Interface Definition Language



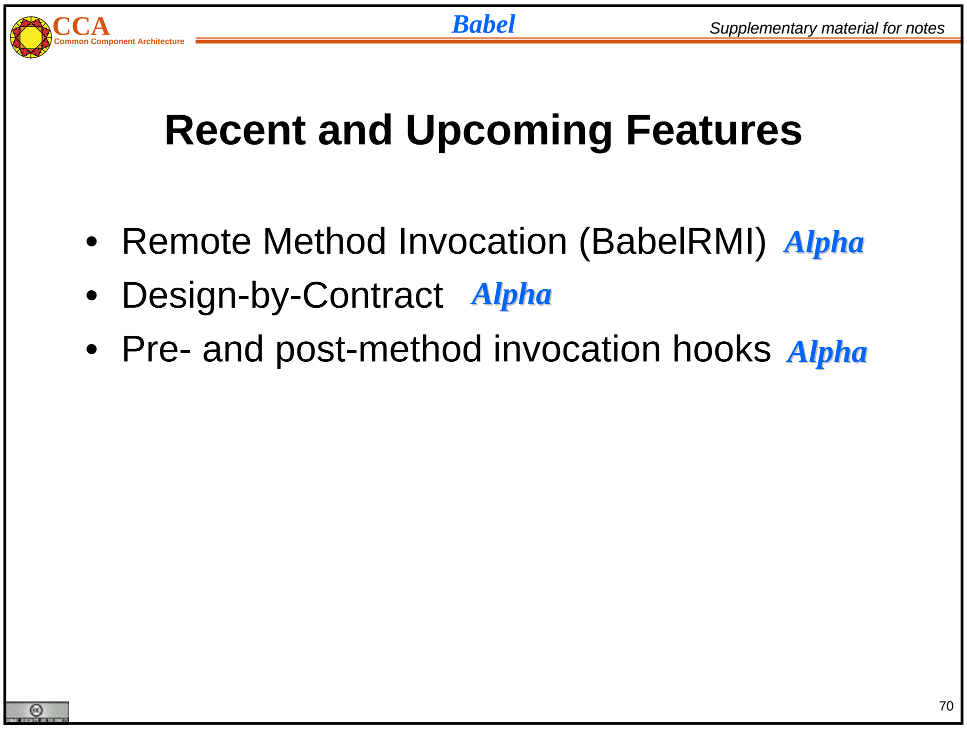
CCA Common Component Architecture

Babel

Clients in any supported language can access components in any other language

Babel presentation coming up!

IOR = Intermediate Object Representation



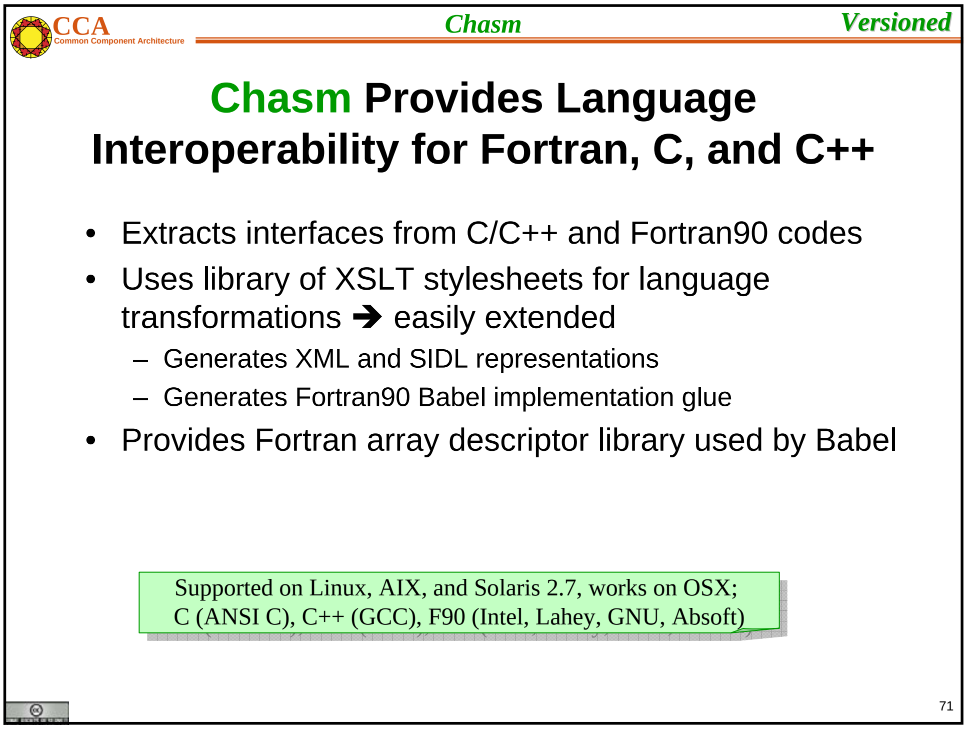
CCA Common Component Architecture

Babel

Supplementary material for notes

Recent and Upcoming Features

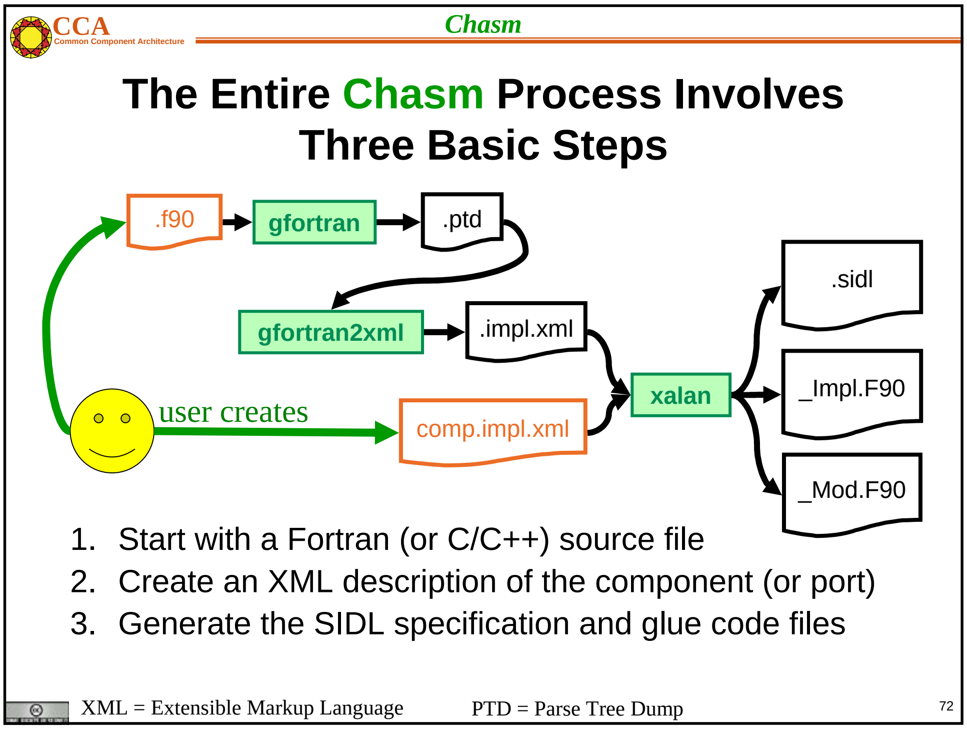
- Remote Method Invocation (BabelRMI) ***Alpha***
- Design-by-Contract ***Alpha***
- Pre- and post-method invocation hooks ***Alpha***



Chasm Provides Language Interoperability for Fortran, C, and C++

- Extracts interfaces from C/C++ and Fortran90 codes
- Uses library of XSLT stylesheets for language transformations ➔ easily extended
  - Generates XML and SIDL representations
  - Generates Fortran90 Babel implementation glue
- Provides Fortran array descriptor library used by Babel

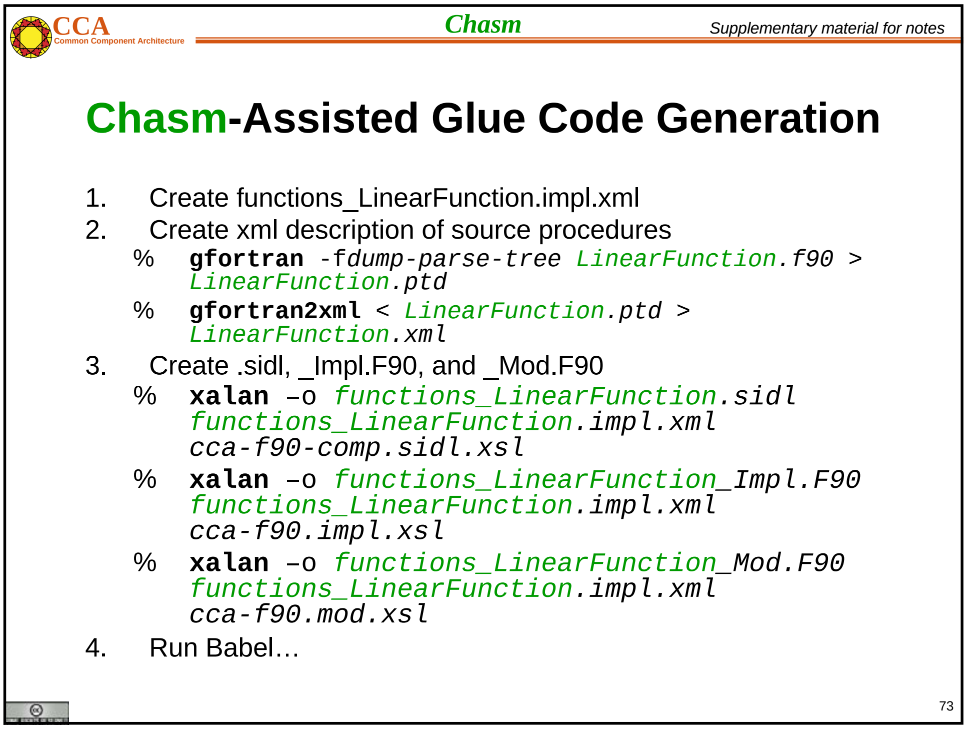
Supported on Linux, AIX, and Solaris 2.7, works on OSX;
C (ANSI C), C++ (GCC), F90 (Intel, Lahey, GNU, Absoft)



The Entire Chasm Process Involves Three Basic Steps

1. Start with a Fortran (or C/C++) source file
2. Create an XML description of the component (or port)
3. Generate the SIDL specification and glue code files

XML = Extensible Markup Language PTD = Parse Tree Dump



Chasm-Assisted Glue Code Generation

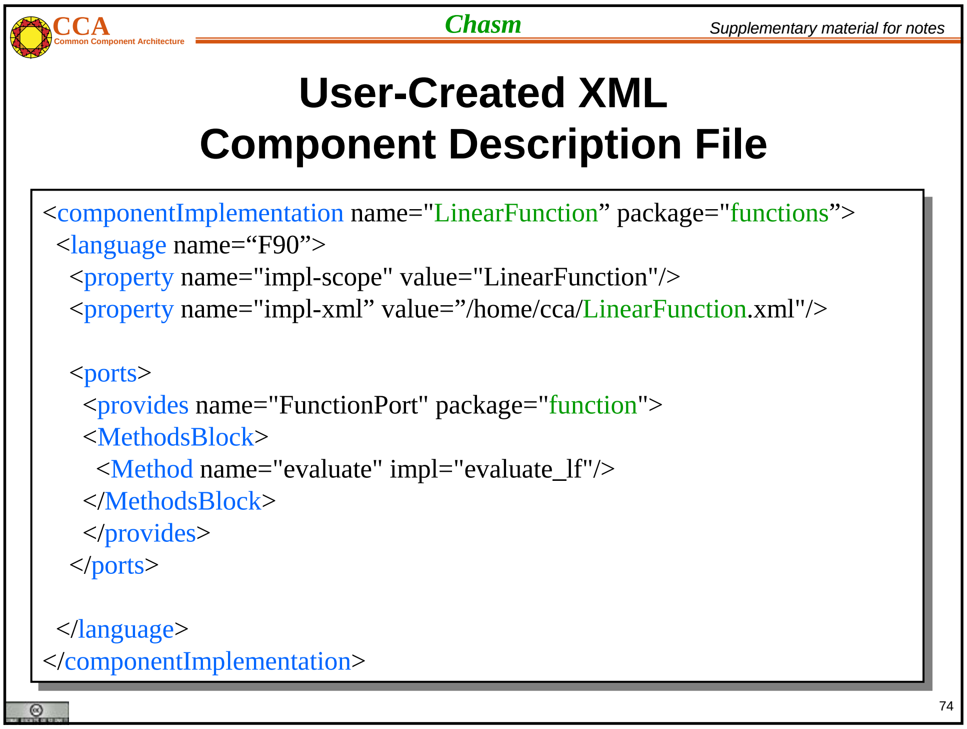
1. Create functions_LinearFunction.impl.xml
2. Create xml description of source procedures

```
% gfortran -fdump-parse-tree LinearFunction.f90 >
  LinearFunction.ptd
% gfortran2xml < LinearFunction.ptd >
  LinearFunction.xml
```

3. Create .sidl, _Impl.F90, and _Mod.F90

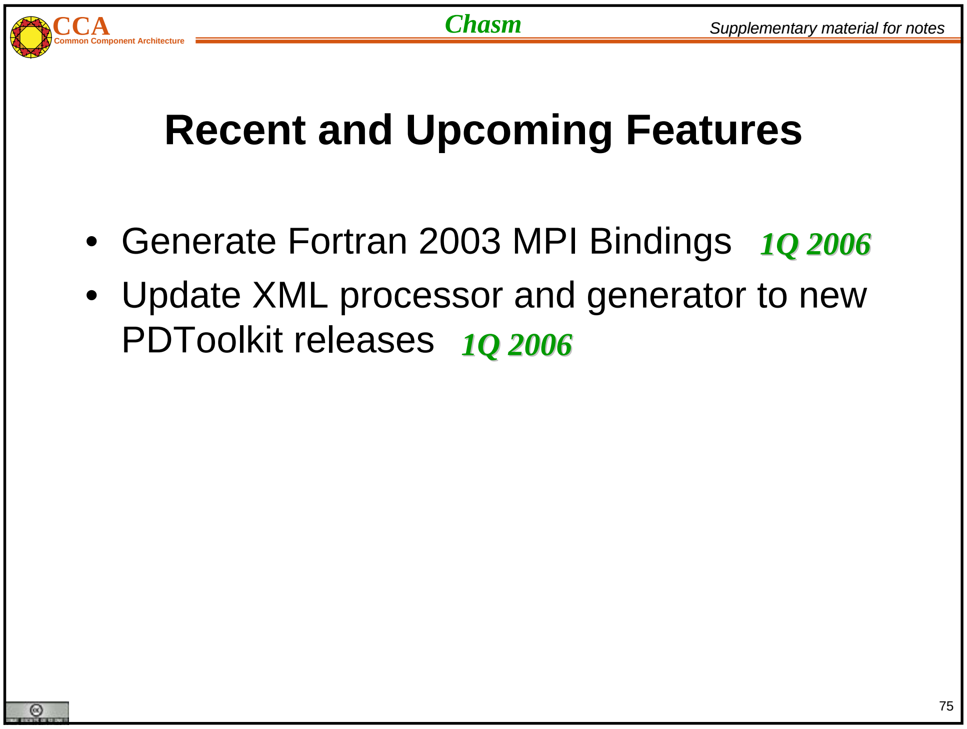
```
% xalan –o functions_LinearFunction.sidl
  functions_LinearFunction.impl.xml
  cca-f90-comp.sidl.xsl
% xalan –o functions_LinearFunction_Impl.F90
  functions_LinearFunction.impl.xml
  cca-f90.impl.xsl
% xalan –o functions_LinearFunction_Mod.F90
  functions_LinearFunction.impl.xml
  cca-f90.mod.xsl
```

4. Run Babel…



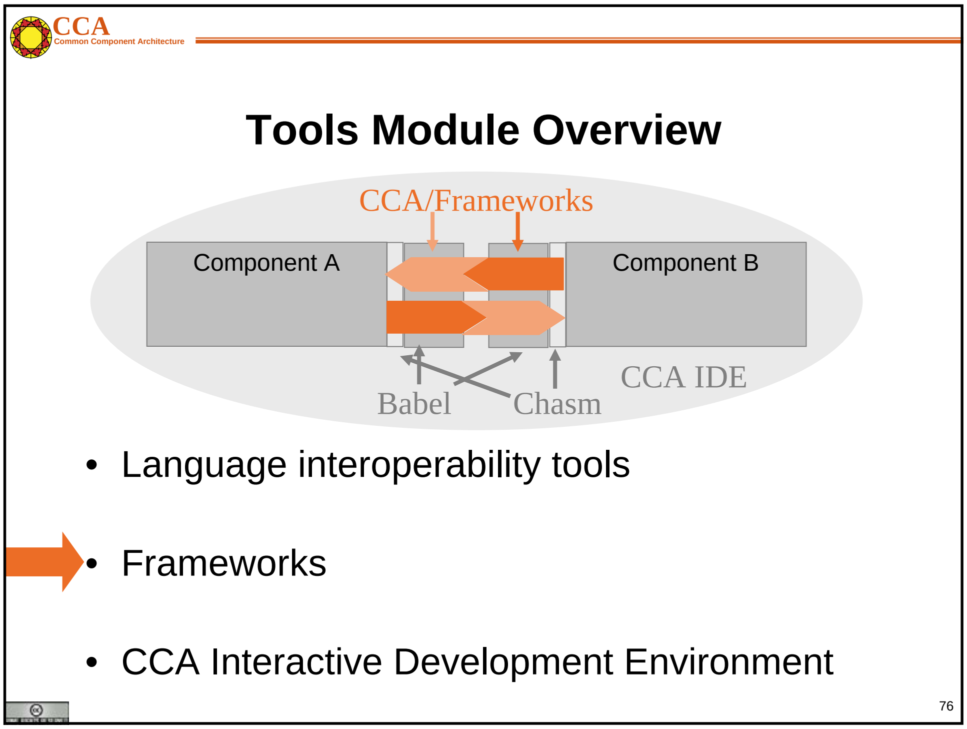
User-Created XML Component Description File

```
<componentImplementation name="LinearFunction” package="functions”>
  <language name=“F90”>
    <property name="impl-scope" value="LinearFunction"/>
    <property name="impl-xml” value=”/home/cca/LinearFunction.xml"/>

    <ports>
      <provides name="FunctionPort" package="function">
      <MethodsBlock>
        <Method name="evaluate" impl="evaluate_lf"/>
      </MethodsBlock>
      </provides>
    </ports>

  </language>
</componentImplementation>
```



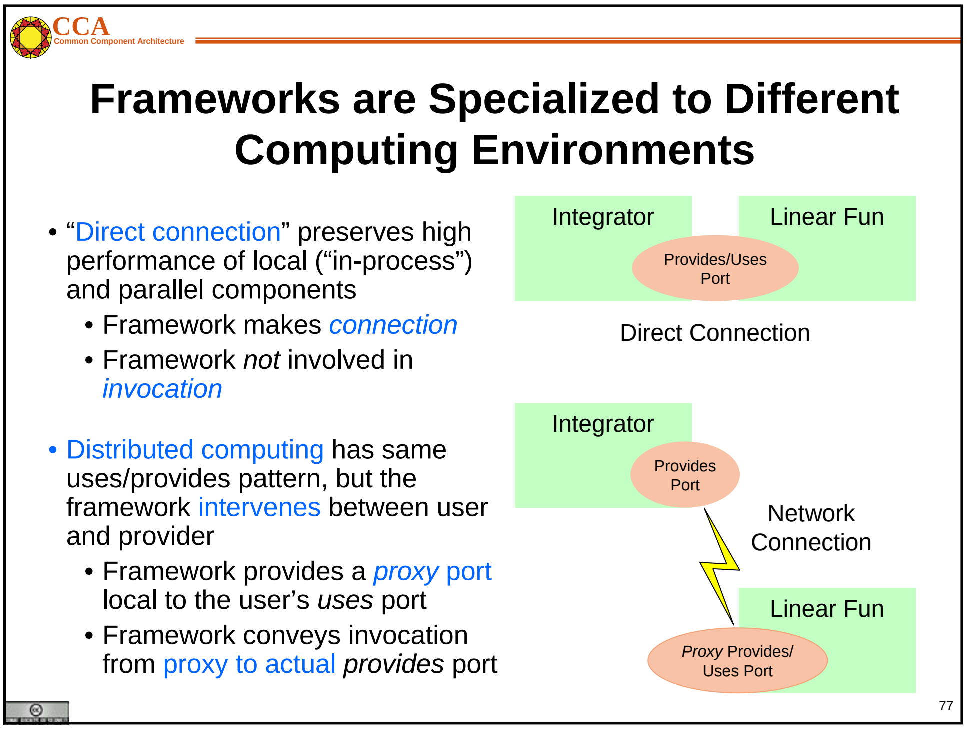
Recent and Upcoming Features

- Generate Fortran 2003 MPI Bindings ***1Q 2006***
- Update XML processor and generator to new PDToolkit releases ***1Q 2006***



Tools Module Overview

- Language interoperability tools
- Frameworks
- CCA Interactive Development Environment



CCA
Common Component Architecture

Frameworks are Specialized to Different Computing Environments

- "Direct connection" preserves high performance of local ("in-process") and parallel components
  - Framework makes *connection*
  - Framework *not* involved in *invocation*
- Distributed computing has same uses/provides pattern, but the framework intervenes between user and provider
  - Framework provides a *proxy* port local to the user's *uses* port
  - Framework conveys invocation from proxy to actual *provides* port

Integrator

Linear Fun

Provides/Uses Port

Direct Connection

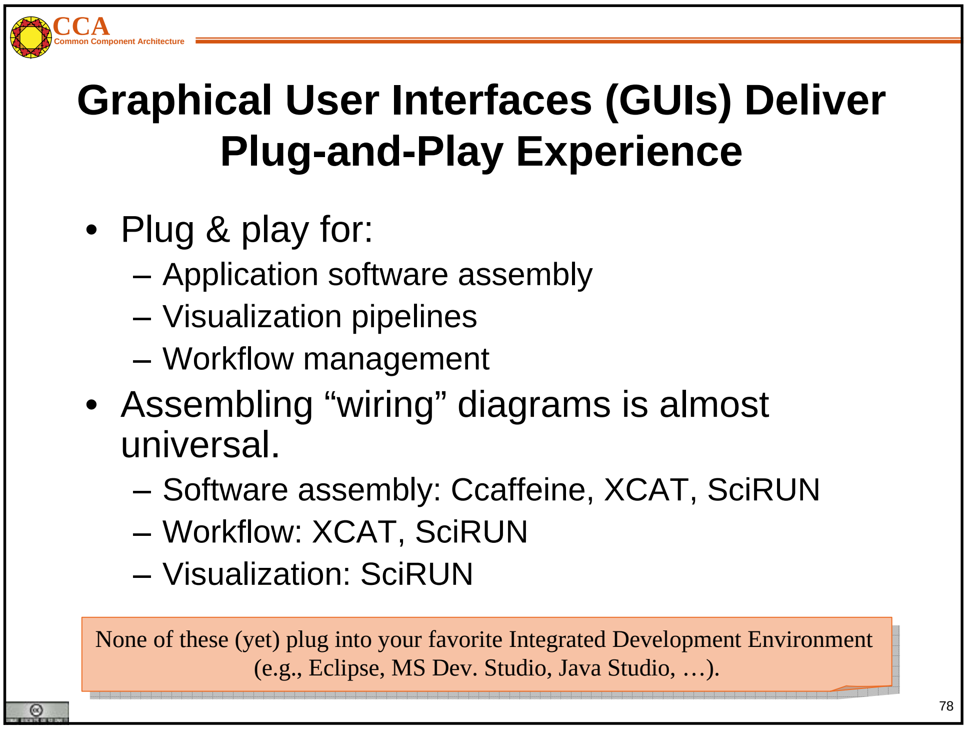
Integrator

Provides Port

Network Connection

Linear Fun

Proxy Provides/ Uses Port

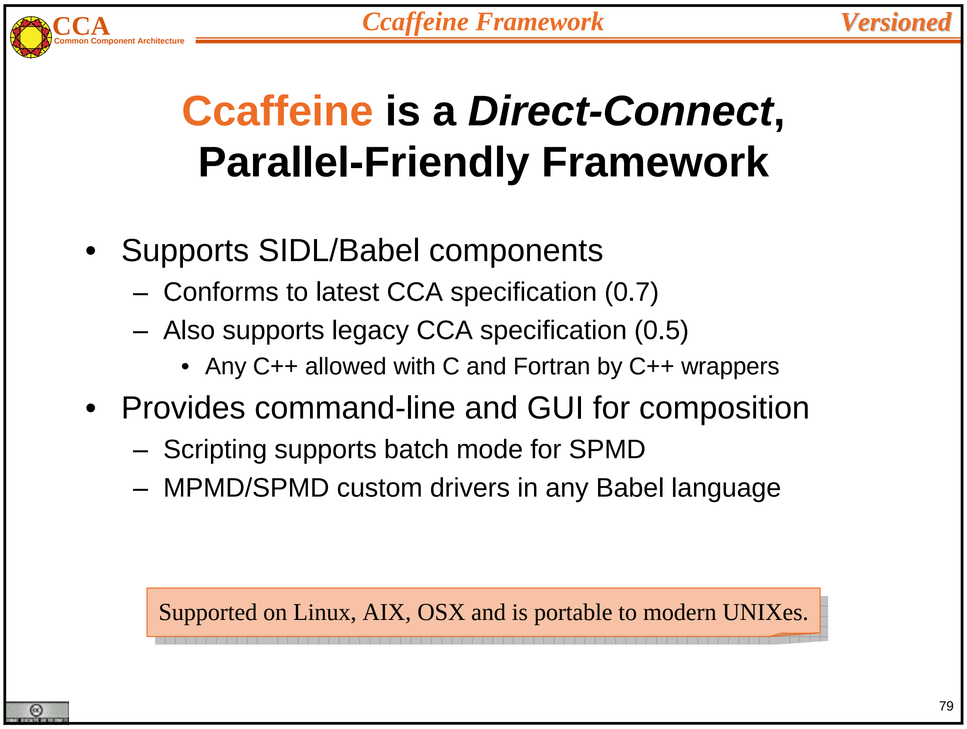


CCA
Common Component Architecture

Graphical User Interfaces (GUIs) Deliver Plug-and-Play Experience

- Plug & play for:
  - Application software assembly
  - Visualization pipelines
  - Workflow management
- Assembling "wiring" diagrams is almost universal.
  - Software assembly: Ccaffeine, XCAT, SciRUN
  - Workflow: XCAT, SciRUN
  - Visualization: SciRUN

None of these (yet) plug into your favorite Integrated Development Environment (e.g., Eclipse, MS Dev. Studio, Java Studio, …).

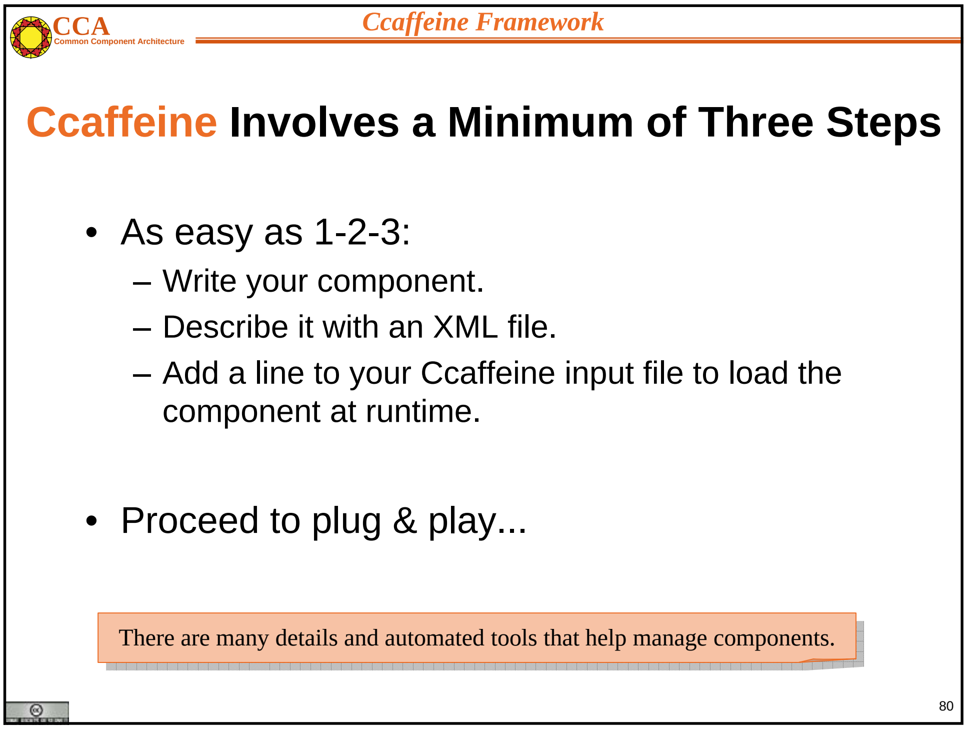


CCA Common Component Architecture — *Ccaffeine Framework* — *Versioned*

Ccaffeine is a *Direct-Connect*, Parallel-Friendly Framework

- Supports SIDL/Babel components
  - Conforms to latest CCA specification (0.7)
  - Also supports legacy CCA specification (0.5)
    - Any C++ allowed with C and Fortran by C++ wrappers
- Provides command-line and GUI for composition
  - Scripting supports batch mode for SPMD
  - MPMD/SPMD custom drivers in any Babel language

Supported on Linux, AIX, OSX and is portable to modern UNIXes.



CCA Common Component Architecture — *Ccaffeine Framework*

Ccaffeine Involves a Minimum of Three Steps

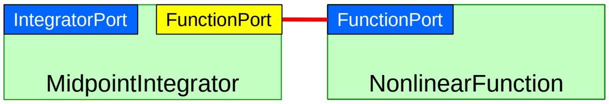
- As easy as 1-2-3:
  - Write your component.
  - Describe it with an XML file.
  - Add a line to your Ccaffeine input file to load the component at runtime.
- Proceed to plug & play...

There are many details and automated tools that help manage components.

Optional

Ccaffeine GUI

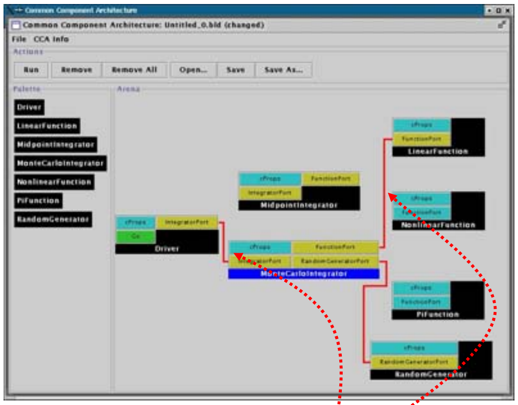
- Process
  - User input is broadcast SPMD-wise from Java.
  - Changes occur in GUI *after* the C++ framework replies.
  - If your components are computing, GUI changes are blocked.
- Components interact through *port connections*
  - *provide* ports implement class or subroutines — "Provides" Port
  - *use* ports call methods or subroutines in the port. — "Uses" Port
  - Links denote caller/callee relationship *not* data flow



Java is required.

User Connects Ports

- Can only connect *uses* & *provides*
  - *Not* uses/uses
  - *Not* provides/provides
- Ports connected by type not name
  - Port names must be unique within component
  - Types must match across components
- Framework puts info about *provider* of port into *using component's* Services object



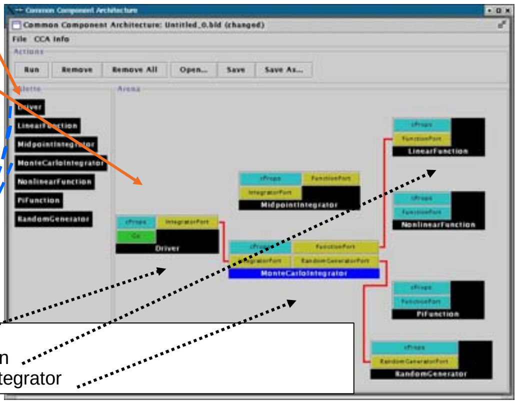
```
connect   Driver               IntegratorPort   MonteCarloIntegrator   IntegratorPort
connect   MonteCarloIntegrator FunctionPort     LinearFunction         FunctionPort
…
```

CCA Common Component Architecture

Ccaffeine Framework (ccafe-gui)

Building an Application (1 of 2)

- Components are code + XML metadata
- Using metadata, a **Palette** of available components is constructed.
- Components are instantiated by user action (i.e. by dragging from **Palette** into **Arena**).
- Framework calls component's constructor, then setServices



create	Driver	Driver
create	LinearFunction	LinearFunction
create	MonteCarloIntegrator	MonteCarloIntegrator

CCA Common Component Architecture

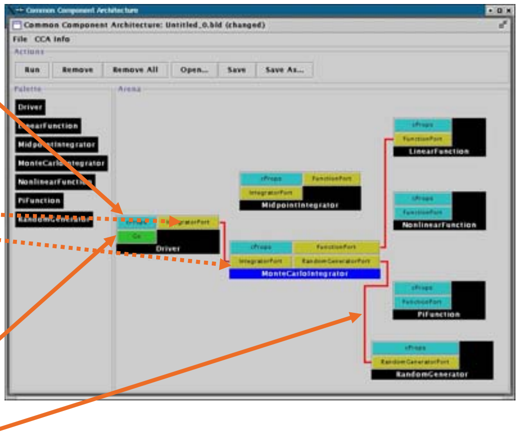
Ccaffeine Framework (ccafe-gui)

Building an Application (2 of 2)

1. Click *Configure* port to start parameter input dialogue.

2. *For each connection*: click a *uses* port then click a *provides* port to establish a connection.

3. Click *Go* port to start the application.

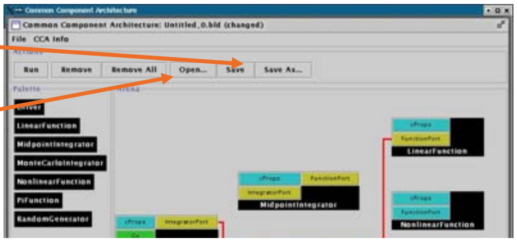


Right-clicking a connection line breaks the connection -- enabling component substitution.

Application Configurations can be Re-used

1. Click *Save* or *Save As…* to save actions.

2. Click *Open* to replay actions.



- Script optimization
  % **simplify-bld** *saved_file.bld* > *faster_file.bld*

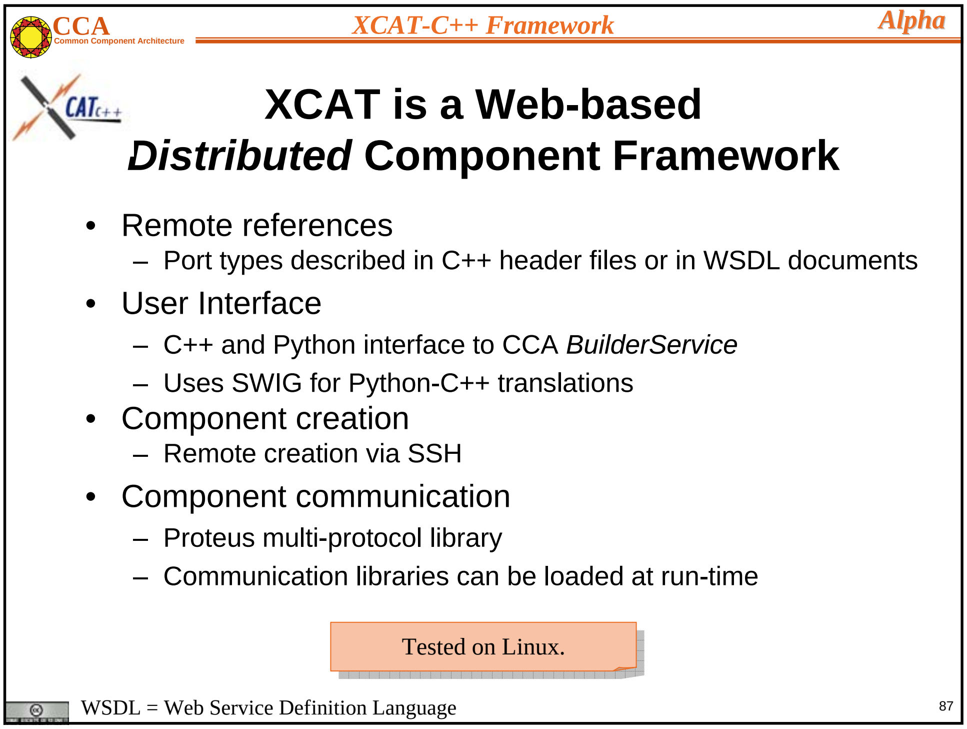
- Batch conversion
  % **bld2rc** *faster_file.bld* > *faster_file.batch*

- C++ stand-alone execution
  % **bld2babel-cpp** *faster_file.bld faster_file_babel outdir*

  or % **bld2neo** *faster_file.bld faster_file.batch outdir*

Recent and Upcoming Features

- Interoperate with other CCA frameworks
  - Via Babel RMI **2H 2006**



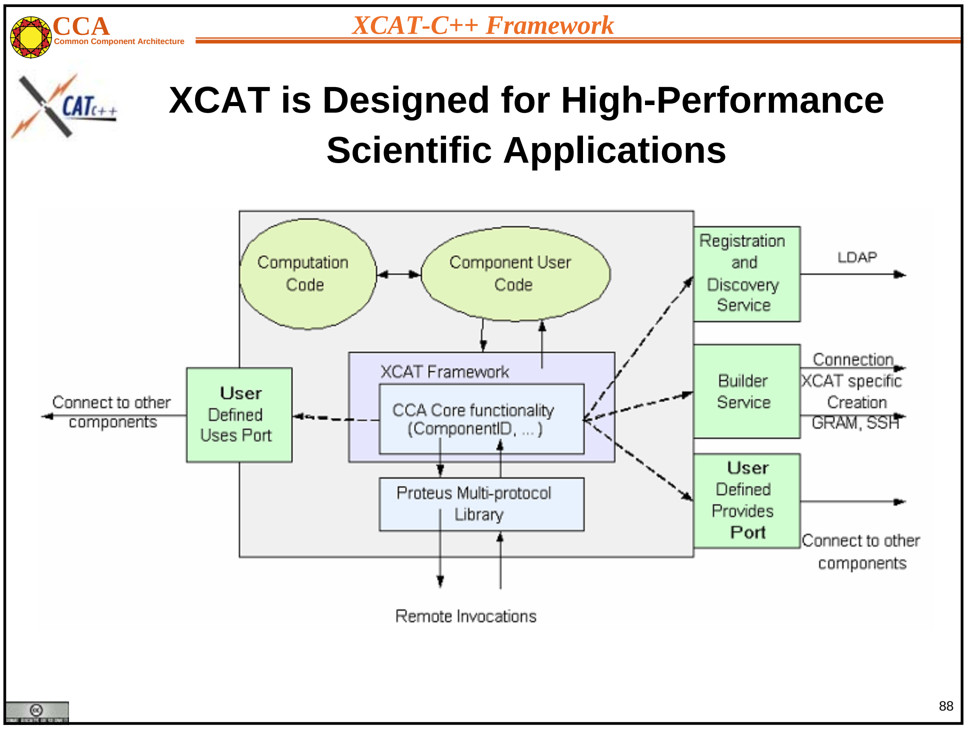
CCA Common Component Architecture — *XCAT-C++ Framework* — *Alpha*

XCAT is a Web-based *Distributed* Component Framework

- Remote references
  - Port types described in C++ header files or in WSDL documents
- User Interface
  - C++ and Python interface to CCA *BuilderService*
  - Uses SWIG for Python-C++ translations
- Component creation
  - Remote creation via SSH
- Component communication
  - Proteus multi-protocol library
  - Communication libraries can be loaded at run-time

Tested on Linux.

WSDL = Web Service Definition Language



CCA Common Component Architecture — *XCAT-C++ Framework*

XCAT is Designed for High-Performance Scientific Applications

Computation Code
Component User Code
Registration and Discovery Service
LDAP
XCAT Framework
CCA Core functionality (ComponentID, …)
Builder Service
Connection
XCAT specific Creation
GRAM, SSH
Connect to other components
User Defined Uses Port
Proteus Multi-protocol Library
User Defined Provides Port
Connect to other components
Remote Invocations

Recent and Upcoming Features

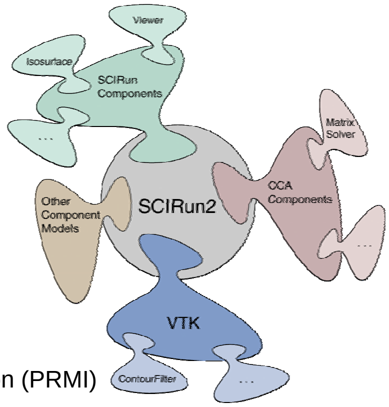
- Support GRAM for component creation **1H 2006**
  - Allow use of grid resources
- Automated component registration and discovery **2H 2006**
- Support new protocols such as UDT (in Proteus) **1H 2006**
- Support Babel's Remote Method Invocation **2H 2006**
  - Allows access to Babel objects through remote Babel stubs
  - Provides direct support for SIDL in distributed applications
  - Leverages Proteus

GRAM = Grid Resource Allocation Management UDT = UDP-based Data Transfer protocol

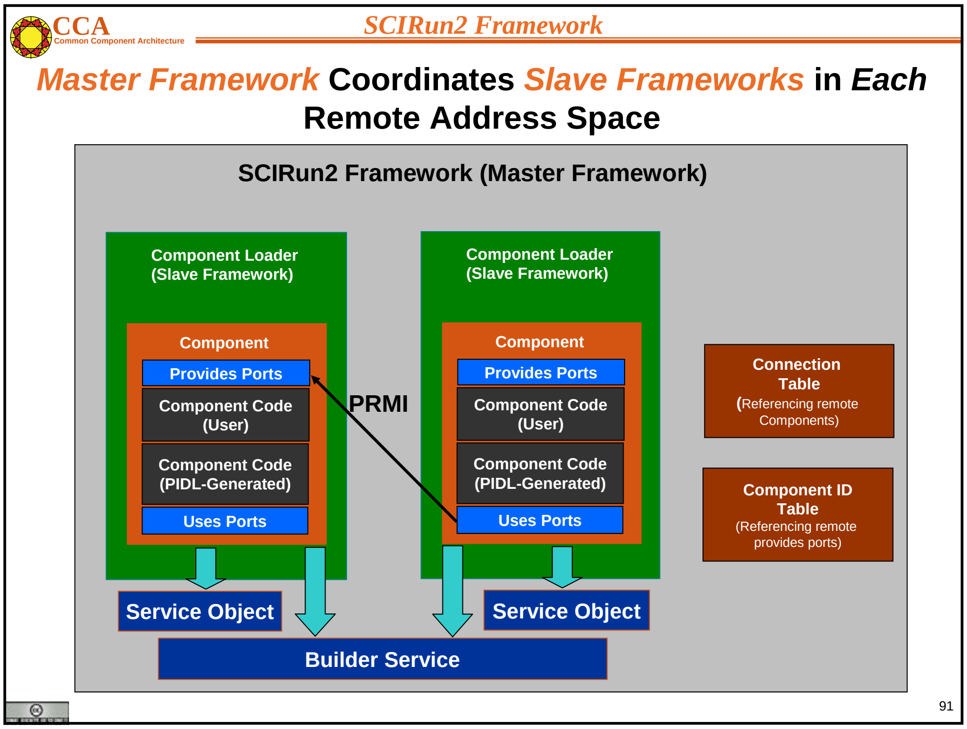
Alpha

SCIRun2 is a Cross-Component Model, *Distributed* Component Framework

- Semi-automatically bridges component *models*
  - Templated components connected at run-time generate bridges
- Parallel Interface Definition Language (PIDL) – a SIDL variant
- User interface – GUI and textual
  - Dynamic composition
- Component and framework creation
  - Remote via SSH
- Component communication
  - C++ RMI with co-location optimization
  - MPI/ Parallel Remote Method Invocation (PRMI)



Supported on Linux.



Master Framework Coordinates *Slave Frameworks* in *Each* Remote Address Space

SCIRun2 Framework (Master Framework)

Component Loader (Slave Framework)

Component

Provides Ports

Component Code (User)

Component Code (PIDL-Generated)

Uses Ports

PRMI

Component Loader (Slave Framework)

Component

Provides Ports

Component Code (User)

Component Code (PIDL-Generated)

Uses Ports

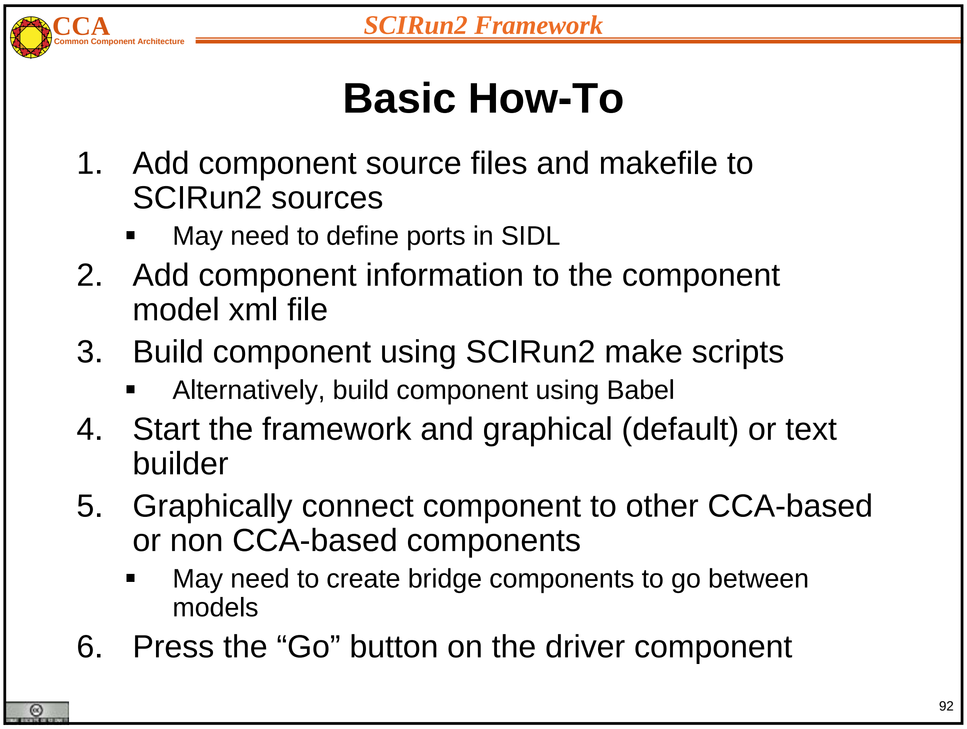
Connection Table (Referencing remote Components)

Component ID Table (Referencing remote provides ports)

Service Object

Service Object

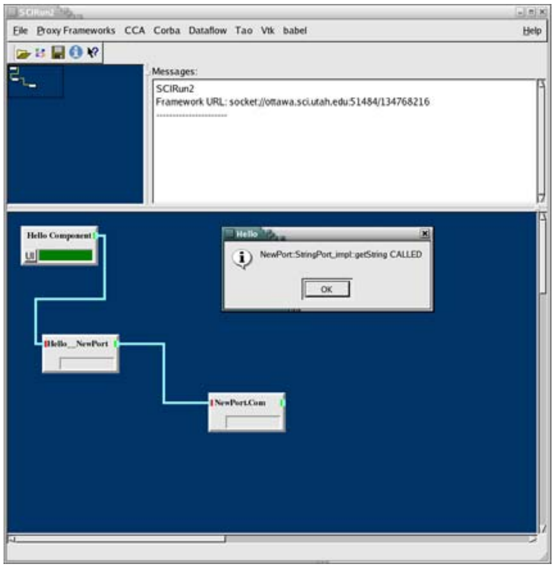
Builder Service



Basic How-To

1. Add component source files and makefile to SCIRun2 sources
   - May need to define ports in SIDL
2. Add component information to the component model xml file
3. Build component using SCIRun2 make scripts
   - Alternatively, build component using Babel
4. Start the framework and graphical (default) or text builder
5. Graphically connect component to other CCA-based or non CCA-based components
   - May need to create bridge components to go between models
6. Press the “Go” button on the driver component

Simple SCIRun2 CCA (PIDL) and Babel Bridge



```
SCIRun2
Framework URL: socket://ottawa.sci.utah.edu:51484/134768216
----------------------
```

NewPort:StringPort_impl:getString CALLED

Recent and Upcoming Features

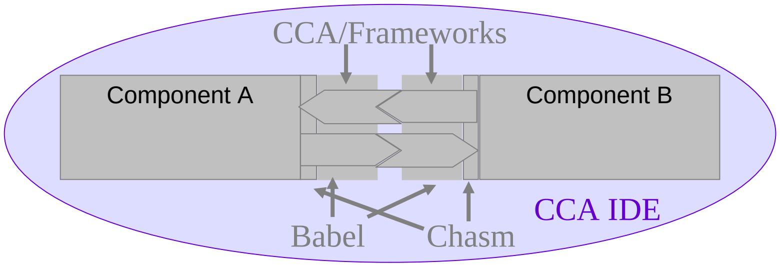
- Merge PIDL with SIDL/Babel *1H 2005*
- Support additional component models
  - Kepler workflows *1H 2006*
- Support Babel's Remote Method Invocation PRMI *2H 2006*
- Automate bridging *On-going*

Experimental Frameworks

Framework	Purpose	Summary
Distributed CCA Framework (DCA)	MxN research	• Goal: explore MxN Parallel-Remote Method Invocation (PRMI) using MPI • Parallel data transfer and redistribution integrated into port invocation mechanism
LegionCCA	Grid-based research	• Goal: allow component-based CCA applications to run in Grid-scale environments using Legion • Supports creation, scheduling, persistence, migration, and fault notification; relies on Legion's built-in RPC mechanism (~Unix sockets)
XCAT-Java Versioned	Globus-based Grid research	• Goal: explore web interface for launching distributed applications • This (alpha) version compatible with latest CCA specification and provides built-in seamless compatibility with OGSI.

OGSI = Open Grid Services Infrastructure

Tools Module Overview



- Language interoperability tools
- Frameworks
- CCA Interactive Development Environment

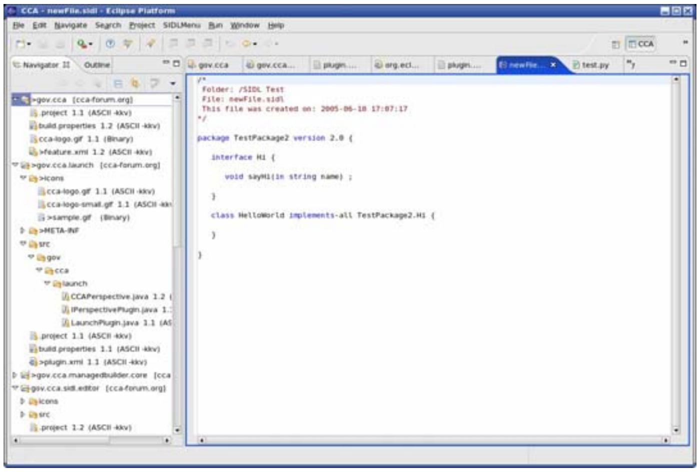
Component Development Environment Provided via Eclipse Plug-ins

- Provides a high-level graphical environment
  - Creating new SIDL-based components
  - Componentizing legacy codes
    - C, C++, Java and Fortran
- Automatic code generation

Supported on Linux, Windows, MacOS.

Component Development Environment Starts at the Eclipse Platform Level

Plug-ins for:
- SIDL Editor
- Wizards
- Preliminary automated build support

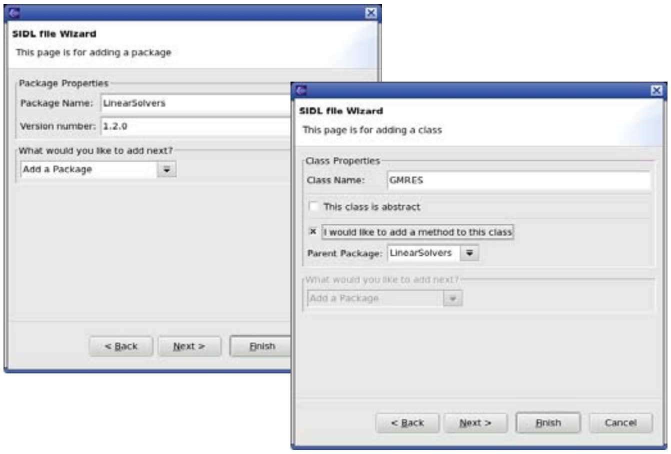

CCA - newFile.sidl - Eclipse Platform
File Edit Navigate Search Project SIDLMenu Run Window Help
CCA
Navigator
Outline
>gov.cca [cca-forum.org]
.project 1.1 (ASCII -kkv)
build.properties 1.2 (ASCII -kkv)
cca-logo.gif 1.1 (Binary)
>feature.xml 1.2 (ASCII -kkv)
>gov.cca.launch [cca-forum.org]
>icons
cca-logo.gif 1.1 (ASCII -kkv)
cca-logo-small.gif 1.1 (ASCII -kk
>sample.gif (Binary)
>META-INF
src
gov
cca
launch
CCAPerspective.java 1.2 (
IPerspectivePlugin.java 1.
LaunchPlugin.java 1.1 (AS
.project 1.1 (ASCII -kkv)
build.properties 1.1 (ASCII -kkv)
>plugin.xml 1.1 (ASCII -kkv)
>gov.cca.managedbuilder.core [cca
gov.cca.sidl.editor [cca-forum.org]
icons
src
.project 1.2 (ASCII -kkv)
gov.cca
gov.cca...
plugin....
org.ecl...
plugin....
newFile...
test.py

```
/*
 Folder: /SIDL Test
 File: newFile.sidl
 This file was created on: 2005-06-10 17:07:17
*/

package TestPackage2 version 2.0 {

   interface Hi {

      void sayHi(in string name) ;

   }

   class HelloWorld implements-all TestPackage2.Hi {

   }

}
```

Imperative that you start by creating a new project!

Wizards are Available for Adding Packages and Classes or Generating SIDL from Legacy Codes

- Intuitive interfaces to port and component definition
- Helper wizards for setting port, component and (in the future) application properties



SIDL file Wizard

This page is for adding a package

Package Properties

Package Name: LinearSolvers

Version number: 1.2.0

What would you like to add next?

Add a Package

SIDL file Wizard

This page is for adding a class

Class Properties

Class Name: GMRES

□ This class is abstract

☒ I would like to add a method to this class

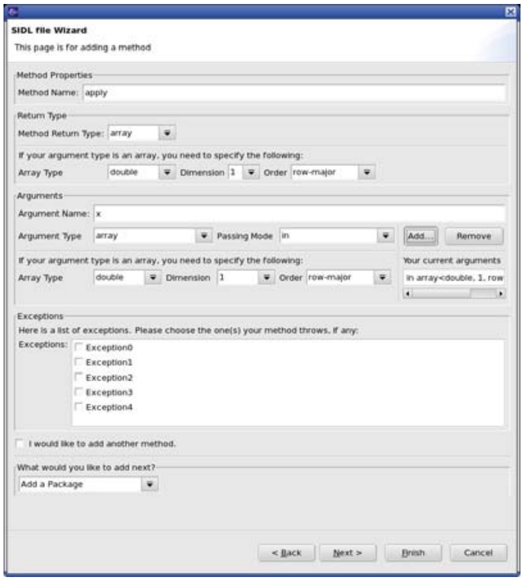
Parent Package: LinearSolvers

What would you like to add next?

Add a Package

Supplementary material for notes

A Wizard is also Available for Adding Methods



SIDL file Wizard

This page is for adding a method

Method Properties

Method Name: apply

Return Type

Method Return Type: array

If your argument type is an array, you need to specify the following:

Array Type double Dimension 1 Order row-major

Arguments

Argument Name: x

Argument Type array Passing Mode in

If your argument type is an array, you need to specify the following:

Array Type double Dimension 1 Order row-major

Your current arguments

in array<double, 1, row

Exceptions

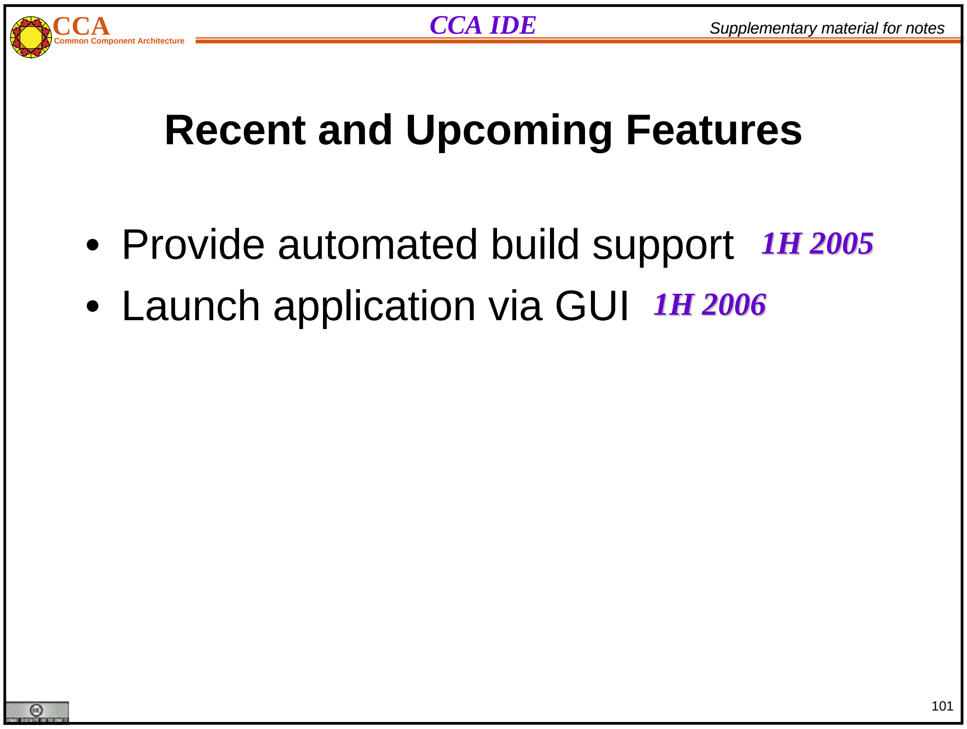
Here is a list of exceptions. Please choose the one(s) your method throws, if any:

Exceptions:
□ Exception0
□ Exception1
□ Exception2
□ Exception3
□ Exception4

□ I would like to add another method.

What would you like to add next?

Add a Package



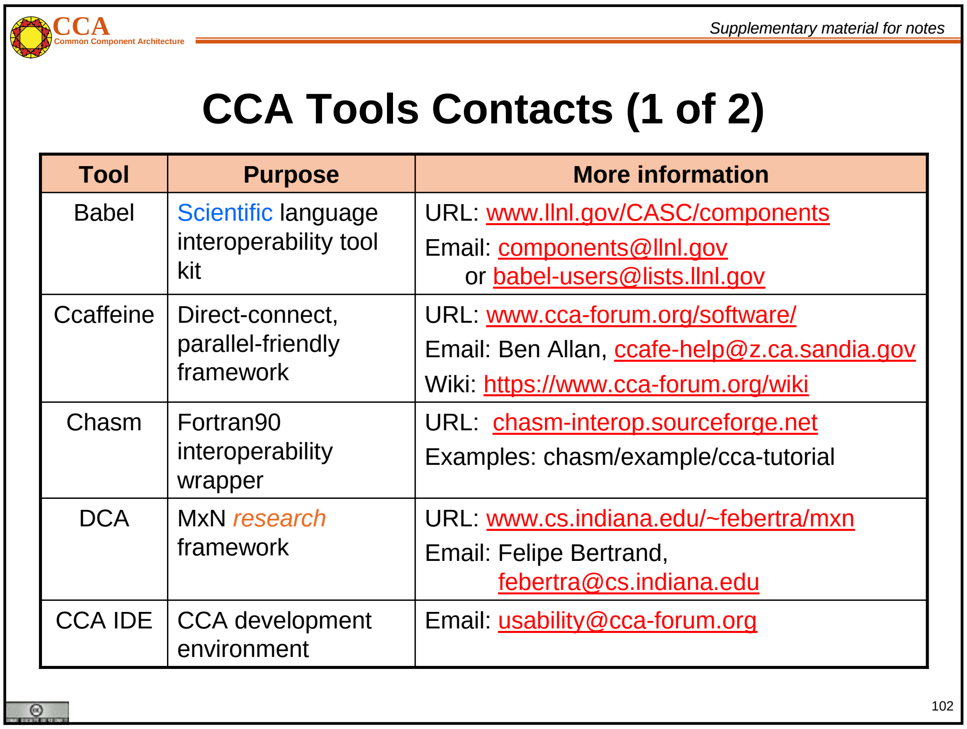
CCA IDE

Supplementary material for notes

Recent and Upcoming Features

- Provide automated build support ***1H 2005***
- Launch application via GUI ***1H 2006***

101

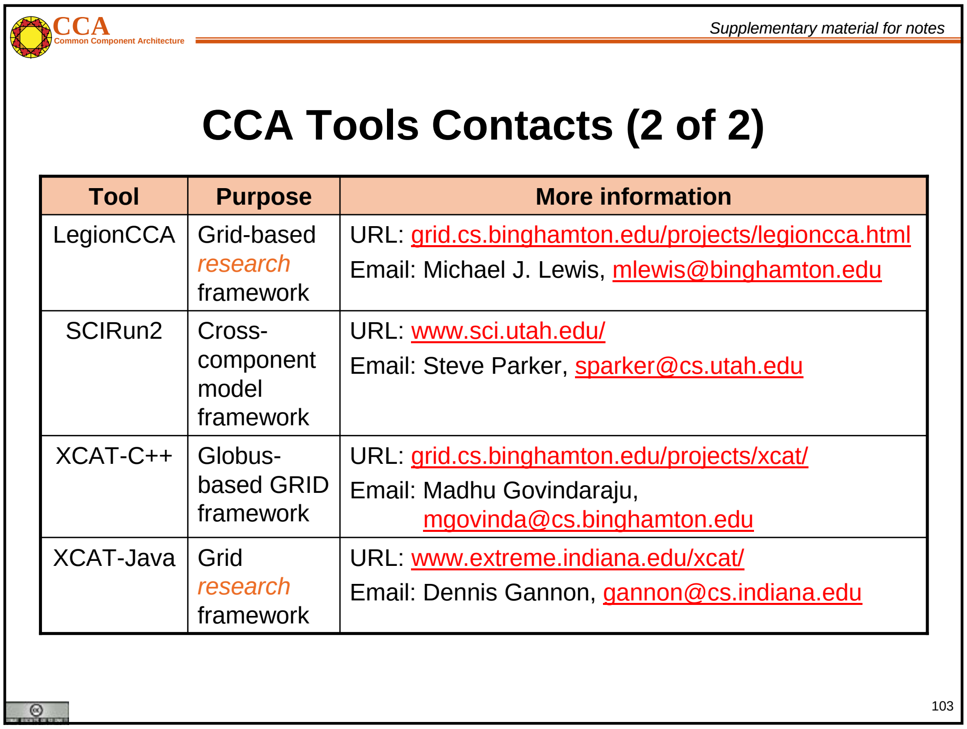


Supplementary material for notes

CCA Tools Contacts (1 of 2)

Tool	Purpose	More information
Babel	Scientific language interoperability tool kit	URL: www.llnl.gov/CASC/components Email: components@llnl.gov or babel-users@lists.llnl.gov
Ccaffeine	Direct-connect, parallel-friendly framework	URL: www.cca-forum.org/software/ Email: Ben Allan, ccafe-help@z.ca.sandia.gov Wiki: https://www.cca-forum.org/wiki
Chasm	Fortran90 interoperability wrapper	URL: chasm-interop.sourceforge.net Examples: chasm/example/cca-tutorial
DCA	MxN *research* framework	URL: www.cs.indiana.edu/~febertra/mxn Email: Felipe Bertrand, febertra@cs.indiana.edu
CCA IDE	CCA development environment	Email: usability@cca-forum.org

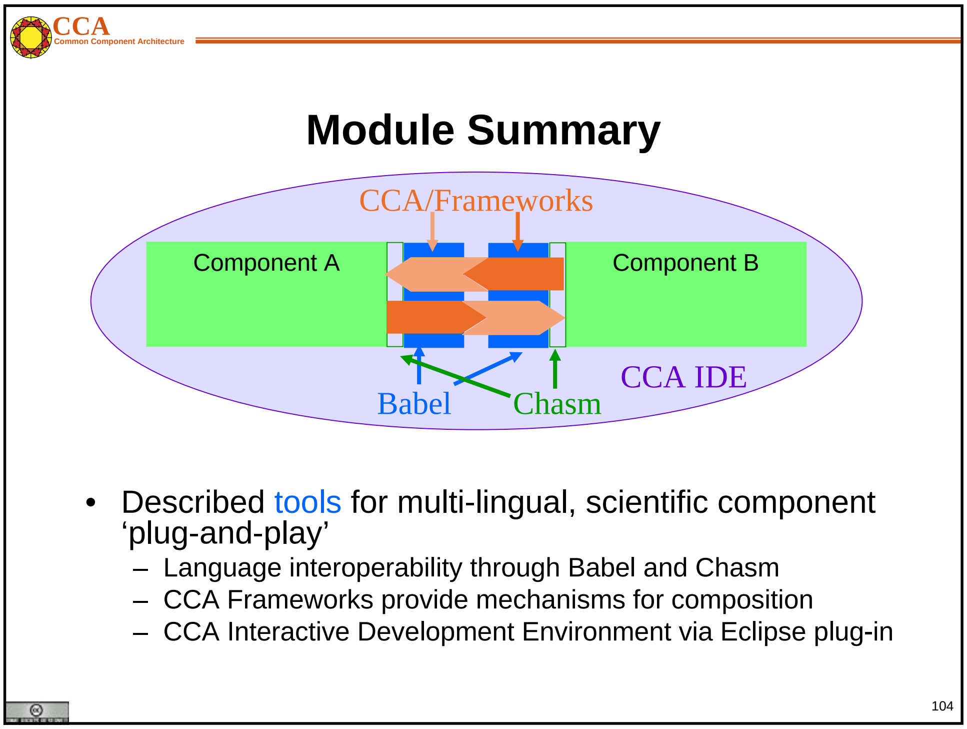
102



CCA Common Component Architecture

CCA Tools Contacts (2 of 2)

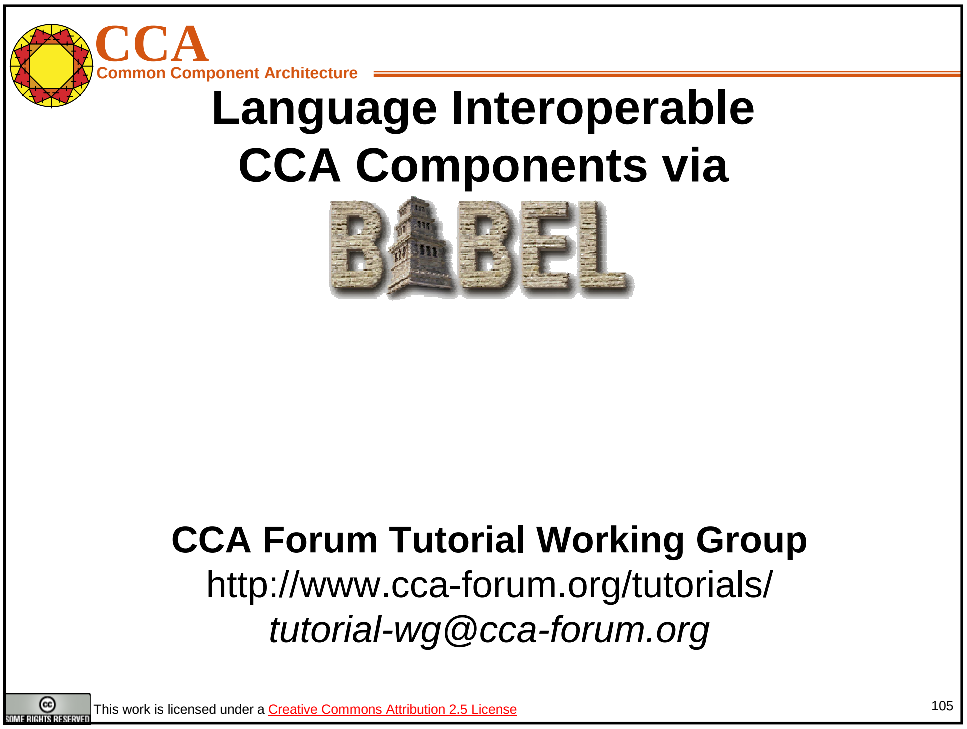
Tool	Purpose	More information
LegionCCA	Grid-based *research* framework	URL: grid.cs.binghamton.edu/projects/legioncca.html Email: Michael J. Lewis, mlewis@binghamton.edu
SCIRun2	Cross-component model framework	URL: www.sci.utah.edu/ Email: Steve Parker, sparker@cs.utah.edu
XCAT-C++	Globus-based GRID framework	URL: grid.cs.binghamton.edu/projects/xcat/ Email: Madhu Govindaraju, mgovinda@cs.binghamton.edu
XCAT-Java	Grid *research* framework	URL: www.extreme.indiana.edu/xcat/ Email: Dennis Gannon, gannon@cs.indiana.edu



CCA Common Component Architecture

Module Summary

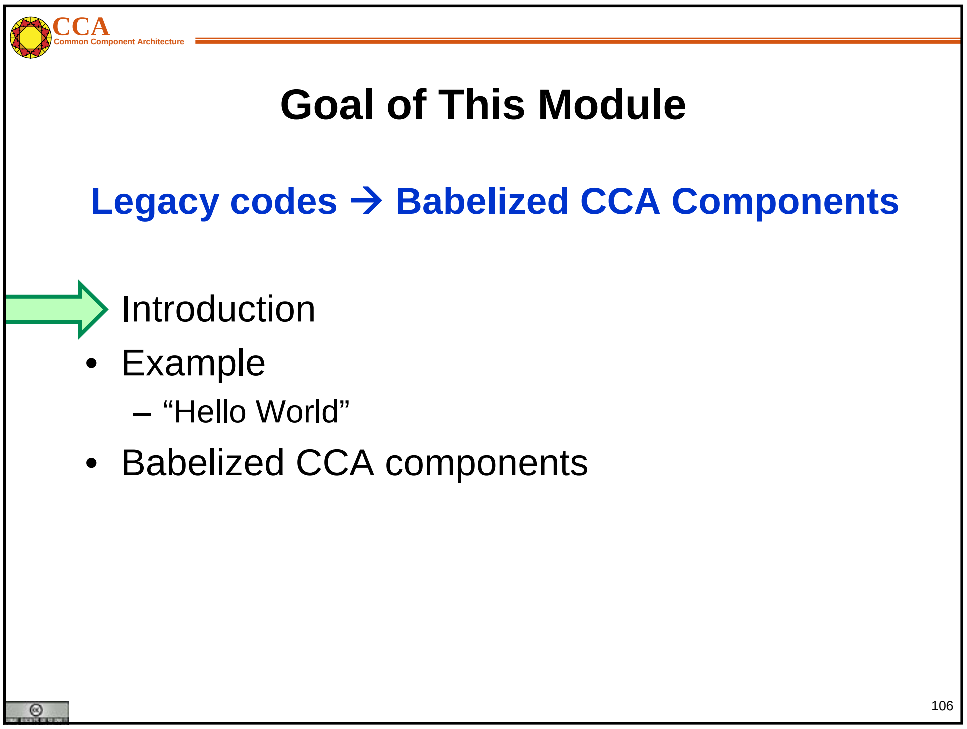
- Described tools for multi-lingual, scientific component 'plug-and-play'
  - Language interoperability through Babel and Chasm
  - CCA Frameworks provide mechanisms for composition
  - CCA Interactive Development Environment via Eclipse plug-in



CCA
Common Component Architecture

Language Interoperable CCA Components via BABEL

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org



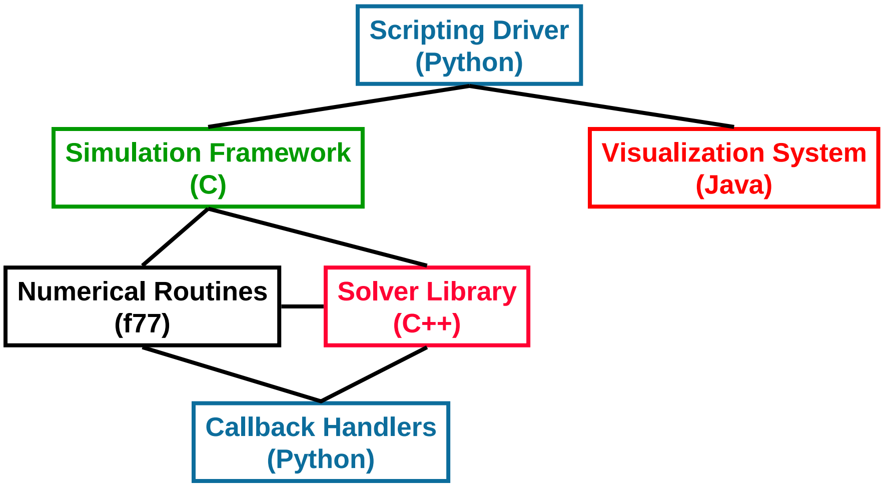
CCA
Common Component Architecture

Goal of This Module

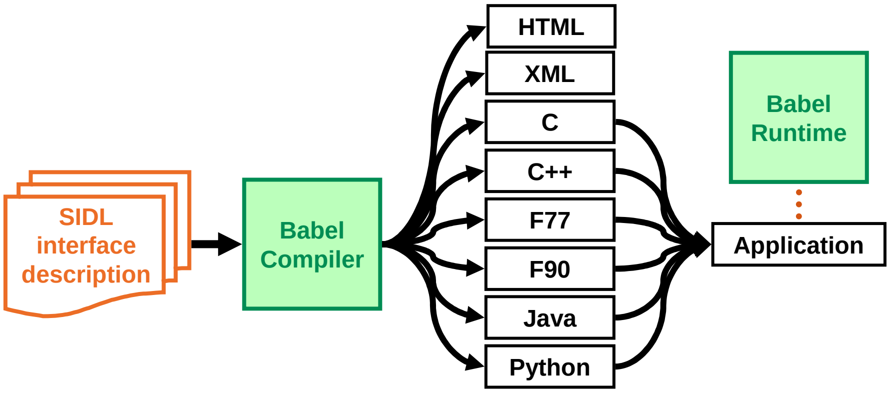
Legacy codes → Babelized CCA Components

- Introduction
- Example
  - “Hello World”
- Babelized CCA components

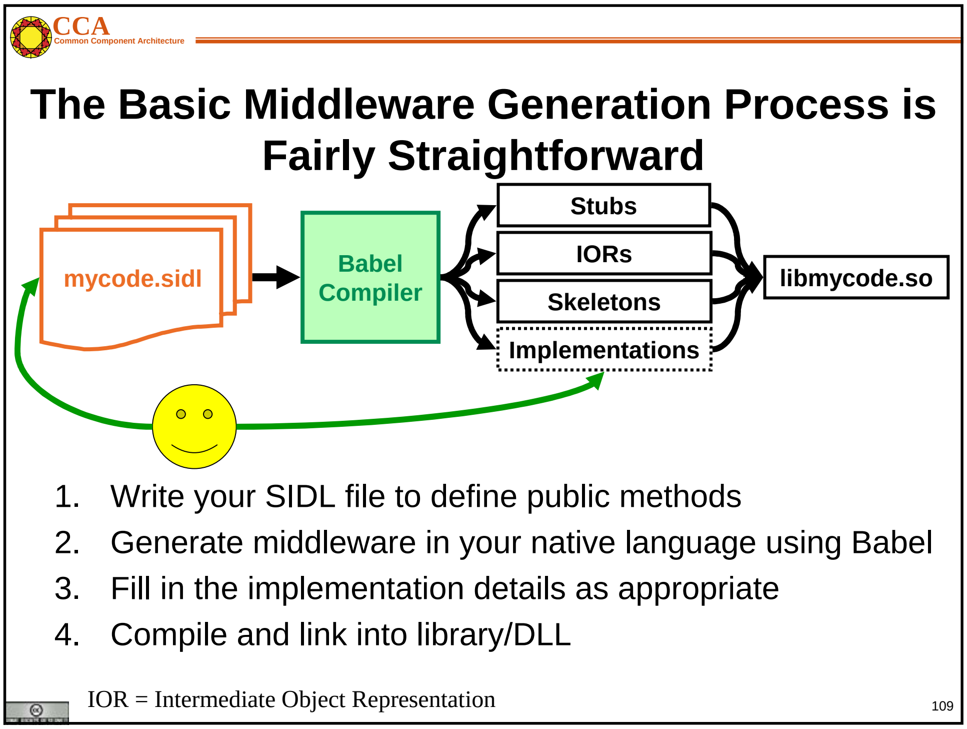
Language Interoperability Facilitates Building Multiple Language Applications


Scripting Driver (Python)
Simulation Framework (C)
Visualization System (Java)
Numerical Routines (f77)
Solver Library (C++)
Callback Handlers (Python)

Babel Toolkit Consists of Two Parts: Code Generator + Runtime Library



SIDL = Scientific Interface Definition Language

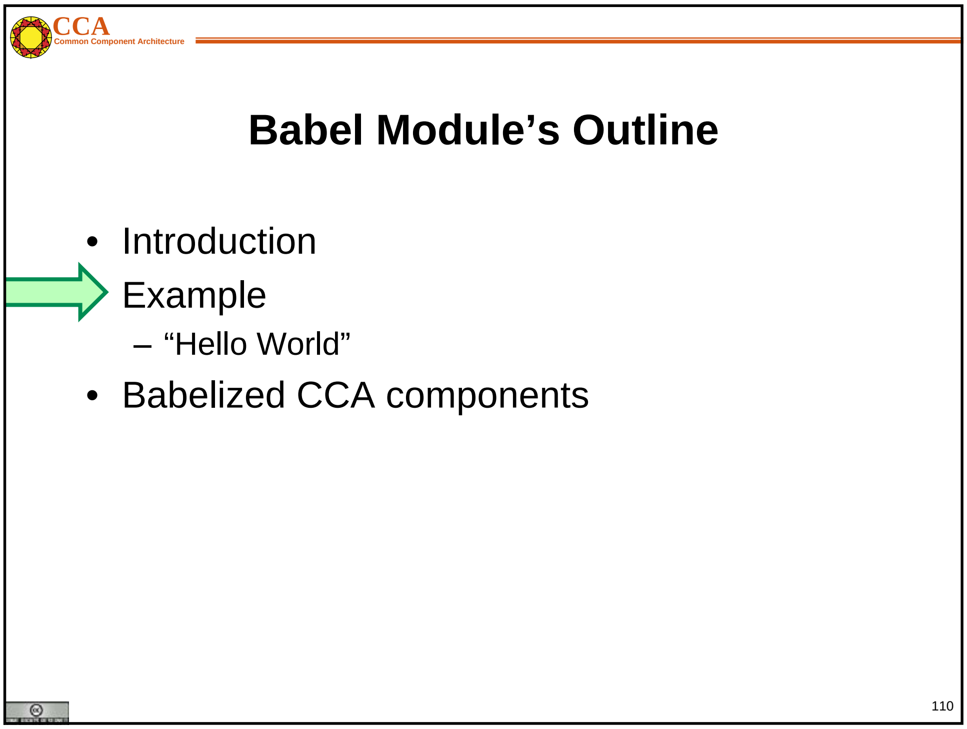


The Basic Middleware Generation Process is Fairly Straightforward

mycode.sidl
Babel Compiler
Stubs
IORs
Skeletons
Implementations
libmycode.so

1. Write your SIDL file to define public methods
2. Generate middleware in your native language using Babel
3. Fill in the implementation details as appropriate
4. Compile and link into library/DLL

IOR = Intermediate Object Representation



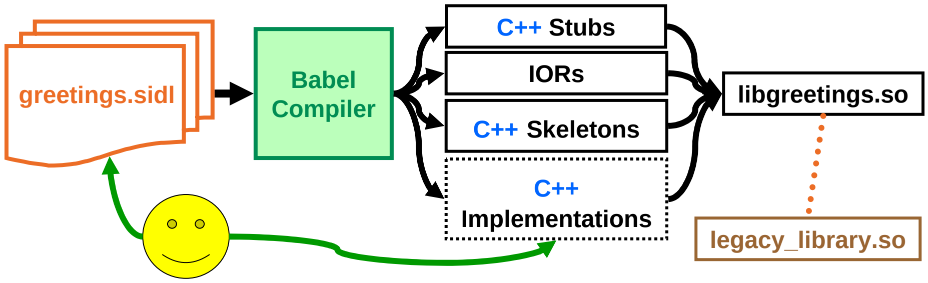
Babel Module’s Outline

- Introduction
- Example
  - “Hello World”
- Babelized CCA components

greetings.sidl: A Sample SIDL File for our "Hello World" Example

```
package greetings version 1.0 {
    interface Hello {
        void setName( in string name );
        string sayIt ( );
    }
    class English implements-all Hello {  }
}
```

At a Minimum, the Library Developer Generates Single-Language Middleware



1. Execute `babel --server=C++ *greetings.sidl*`
2. Fill in the implementation details *to dispatch to legacy code*
3. Compile and link into a library/DLL

CCA
Common Component Architecture

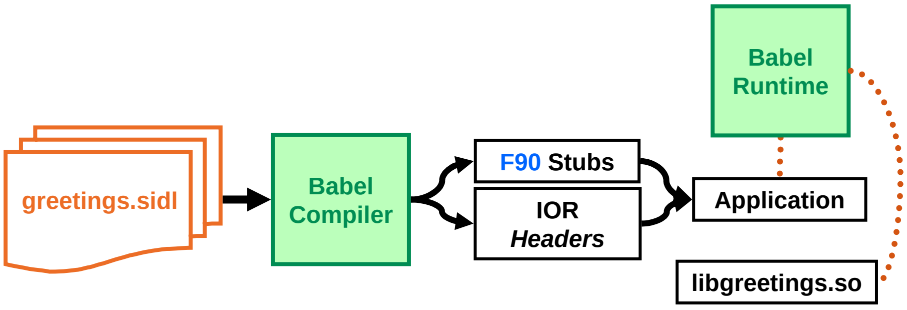
Implementation Details Must be Filled in Between Splicer Blocks

```
namespace greetings {
class English_impl {
  private:
    // DO-NOT-DELETE splicer.begin(greetings.English._impl)
    string d_name;
    // DO-NOT-DELETE splicer.end(greetings.English._impl)
```

```
string
greetings::English_impl::sayIt()
throw ()
{
  // DO-NOT-DELETE splicer.begin(greetings.English.sayIt)
  string msg(“Hello “);
  return msg + d_name + “!”;
  // DO-NOT-DELETE splicer.end(greetings.English.sayIt)
}
```

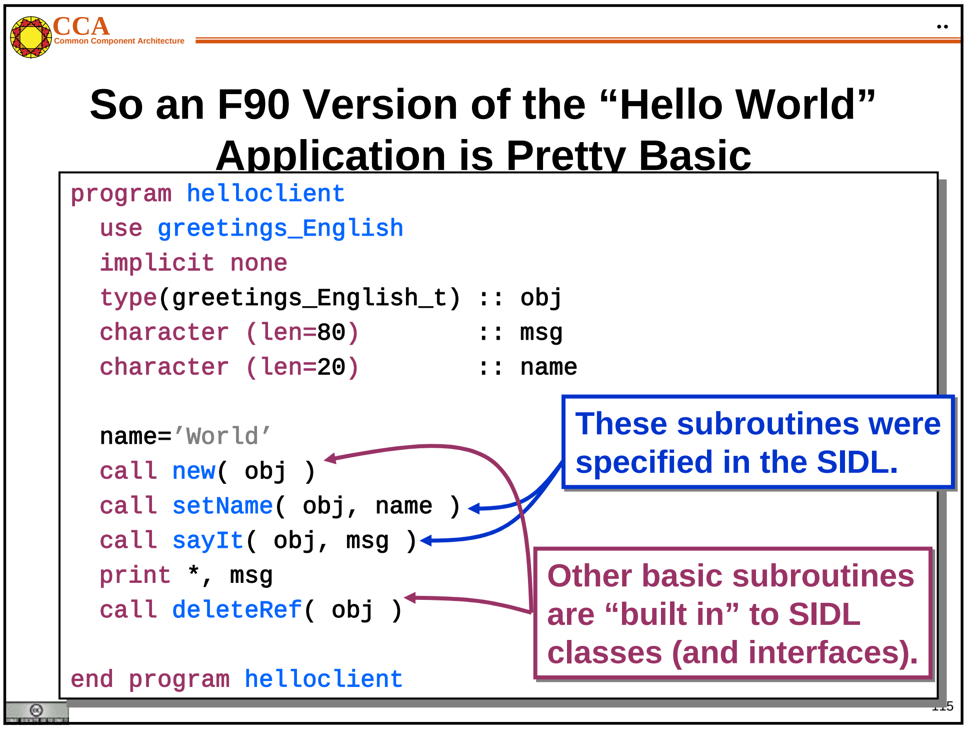
CCA
Common Component Architecture

To Allow the Library User To Use Another Language, the Library Developer Must Generate Stubs



1. Execute `babel --client=*F90 greetings.sidl*`
2. Invoke stub versions of the methods
3. Compile and link with generated code, library, & Runtime
4. Place DLL in suitable location

Could be done by the User!



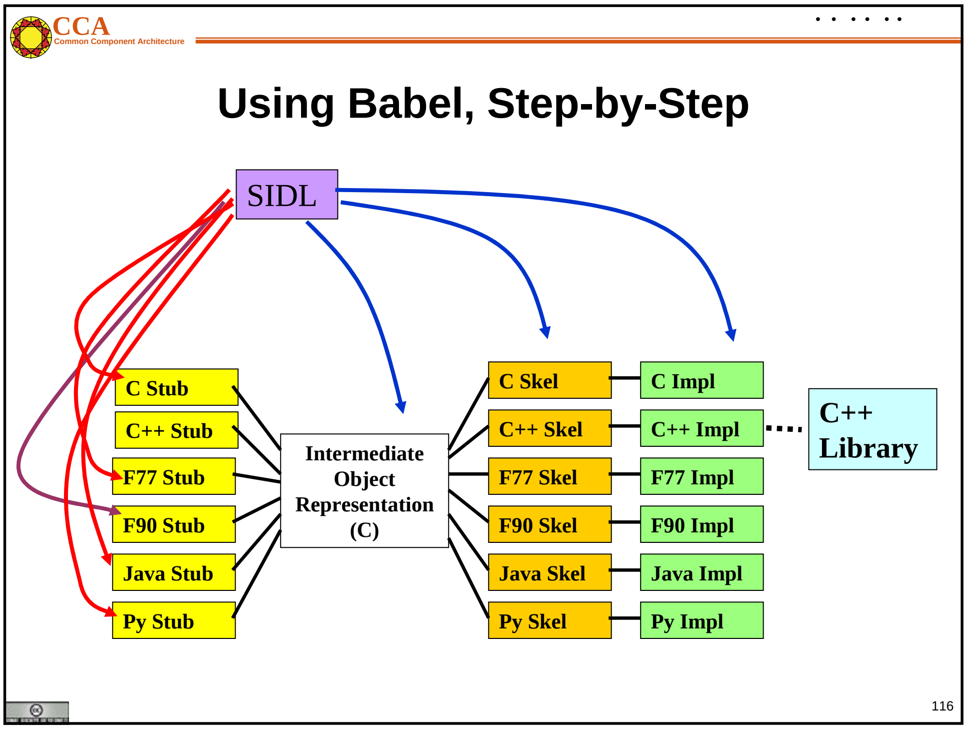
CCA
Common Component Architecture

So an F90 Version of the “Hello World” Application is Pretty Basic

```
program helloclient
  use greetings_English
  implicit none
  type(greetings_English_t) :: obj
  character (len=80)        :: msg
  character (len=20)        :: name

  name='World'
  call new( obj )
  call setName( obj, name )
  call sayIt( obj, msg )
  print *, msg
  call deleteRef( obj )

end program helloclient
```

These subroutines were specified in the SIDL.

Other basic subroutines are “built in” to SIDL classes (and interfaces).



CCA
Common Component Architecture

Using Babel, Step-by-Step

SIDL

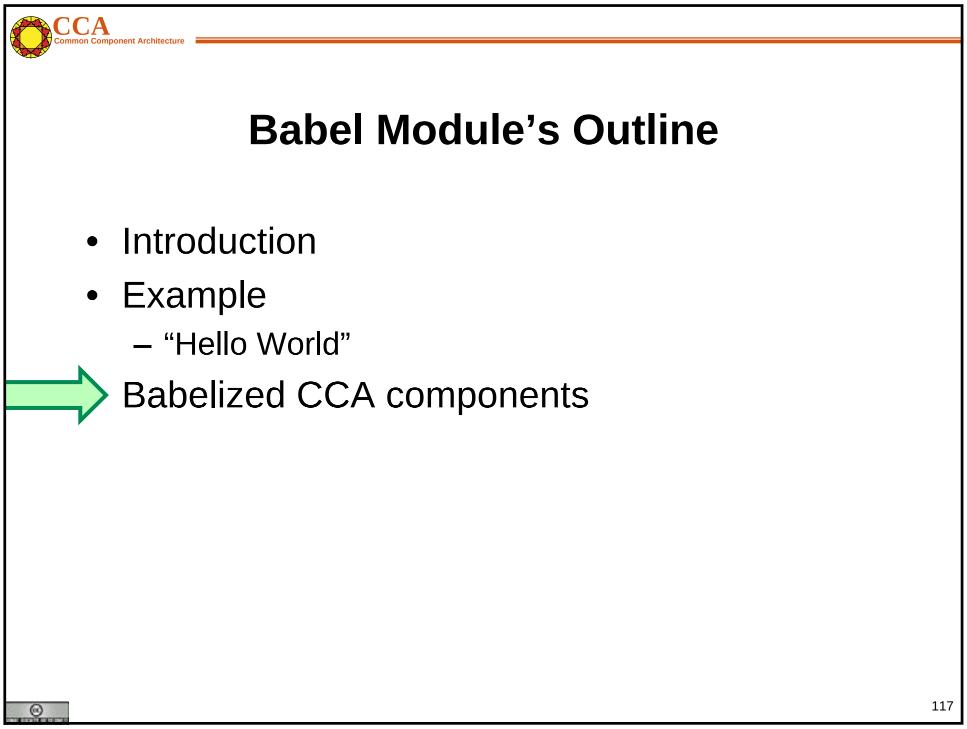
C Stub
C++ Stub
F77 Stub
F90 Stub
Java Stub
Py Stub

Intermediate Object Representation (C)

C Skel
C++ Skel
F77 Skel
F90 Skel
Java Skel
Py Skel

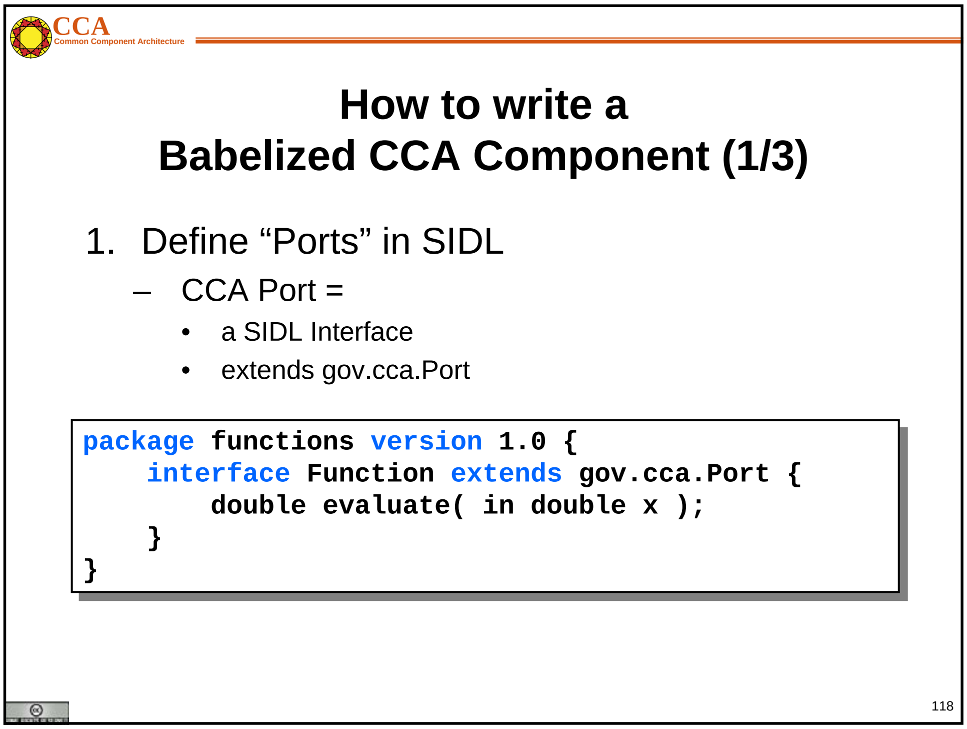
C Impl
C++ Impl
F77 Impl
F90 Impl
Java Impl
Py Impl

C++ Library



Babel Module’s Outline

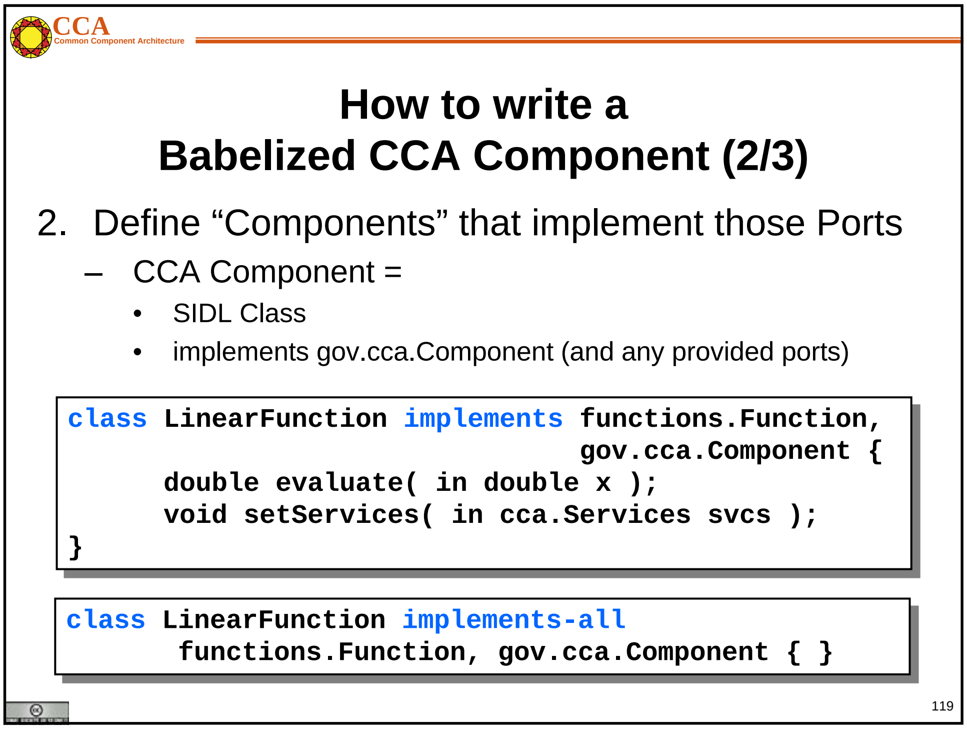
- Introduction
- Example
  - – “Hello World”
- Babelized CCA components



How to write a Babelized CCA Component (1/3)

1. Define “Ports” in SIDL
   - – CCA Port =
     - a SIDL Interface
     - extends gov.cca.Port

```
package functions version 1.0 {
    interface Function extends gov.cca.Port {
        double evaluate( in double x );
    }
}
```



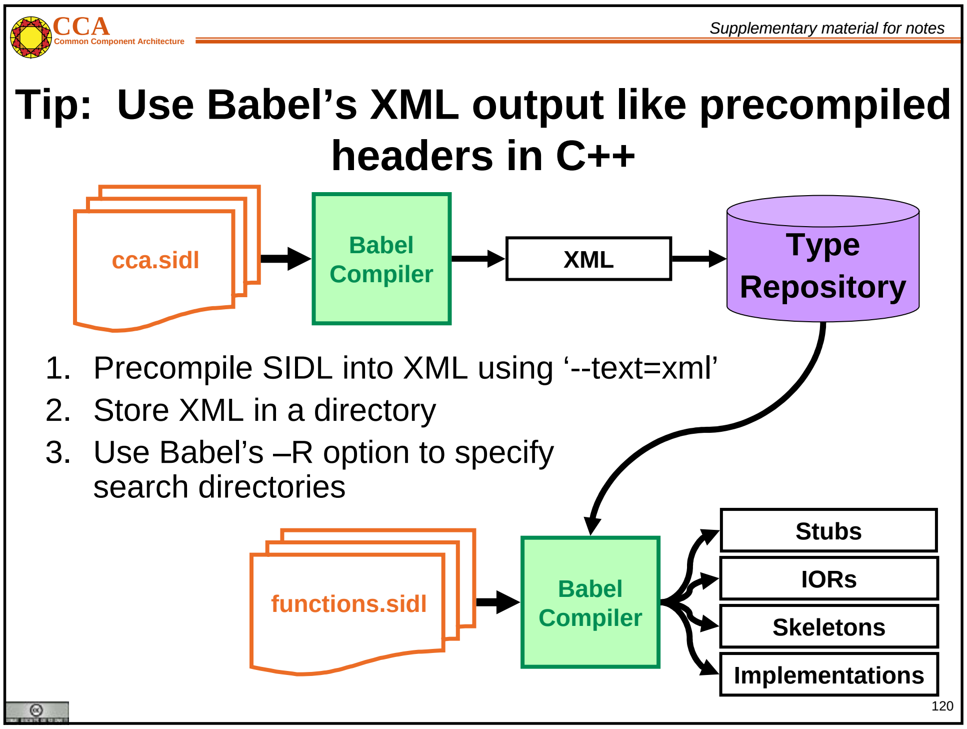
CCA Common Component Architecture

How to write a Babelized CCA Component (2/3)

2. Define “Components” that implement those Ports
   - CCA Component =
     - SIDL Class
     - implements gov.cca.Component (and any provided ports)

```
class LinearFunction implements functions.Function,
                                gov.cca.Component {
      double evaluate( in double x );
      void setServices( in cca.Services svcs );
}
```

```
class LinearFunction implements-all
       functions.Function, gov.cca.Component { }
```



CCA Common Component Architecture

Supplementary material for notes

Tip: Use Babel’s XML output like precompiled headers in C++

cca.sidl

Babel Compiler

XML

Type Repository

1. Precompile SIDL into XML using ‘--text=xml’
2. Store XML in a directory
3. Use Babel’s –R option to specify search directories

functions.sidl

Babel Compiler

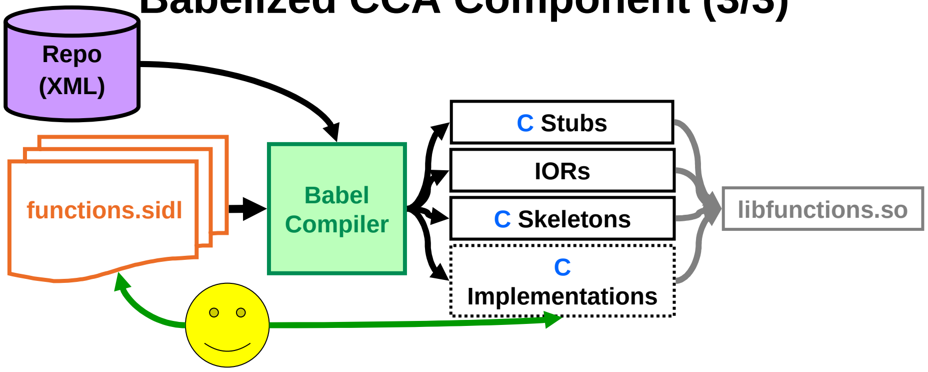
Stubs

IORs

Skeletons

Implementations

How to write a Babelized CCA Component (3/3)



3. Use Babel to generate the interoperability glue
   - Execute `babel --server=*C* –R*Repo functions.sidl*`
4. Fill in Implementations as needed

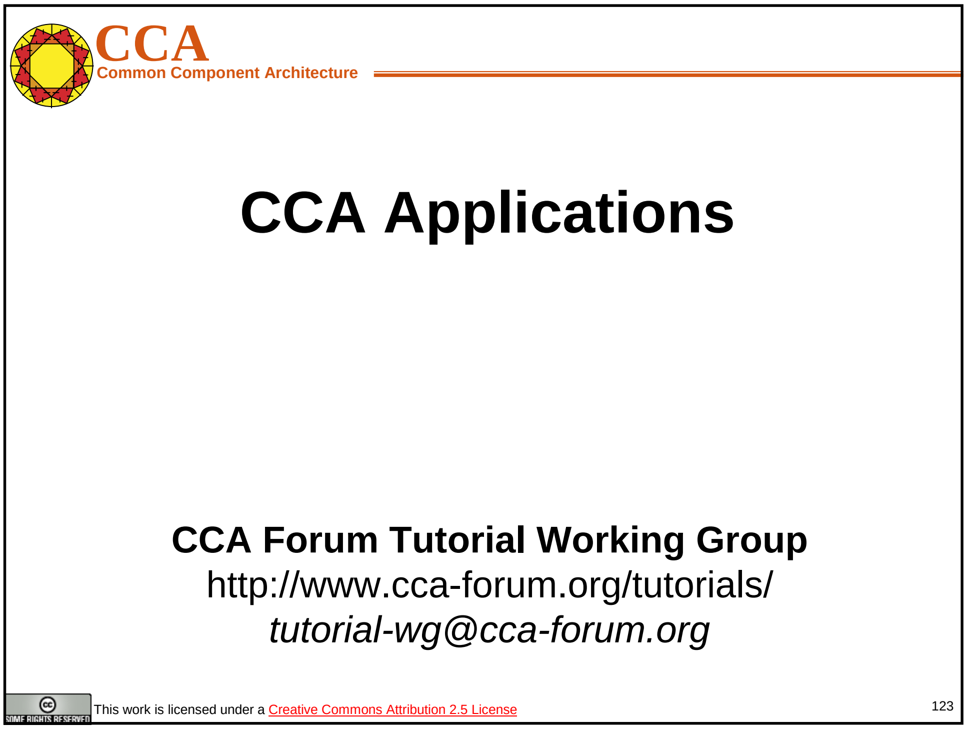
Contact Information

- Project: http://www.llnl.gov/CASC/components
  - Babel: language interoperability tool
  - Alexandria: component repository
  - Quorum: web-based parliamentary system
  - Gauntlet: testing framework
- Project Team Email: components@llnl.gov
- Mailing Lists: majordomo@lists.llnl.gov

  subscribe babel-users *[email address]*

  subscribe babel-announce *[email address]*
- Bug Tracking: https://www.cca-forum.org/bugs/babel/

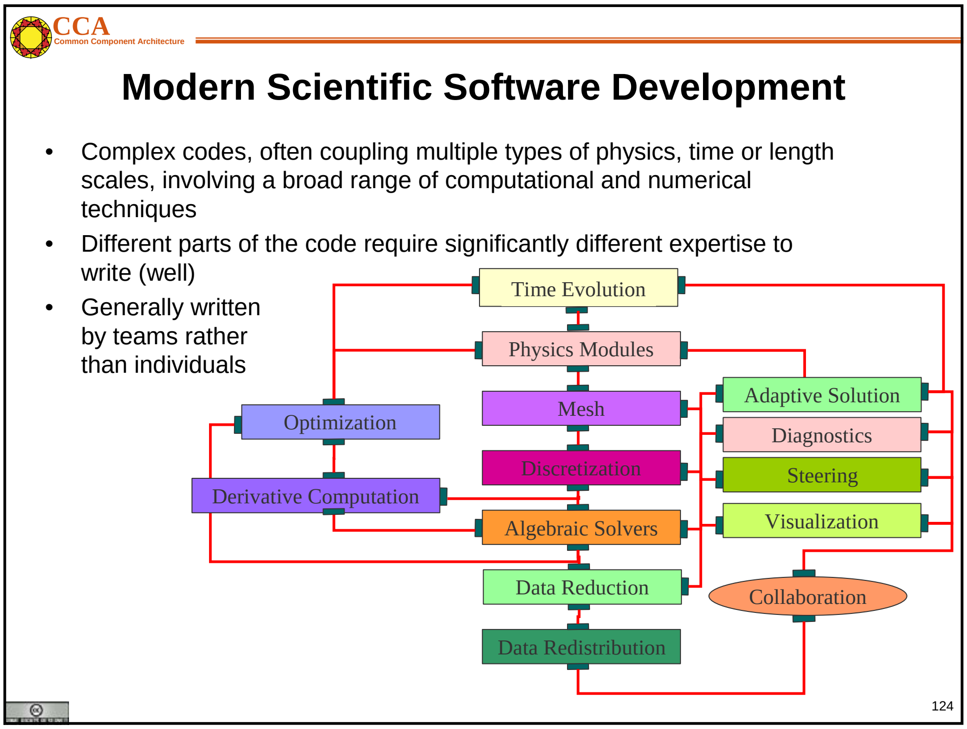
  or email to babel-bugs@cca-forum.org



CCA
Common Component Architecture

CCA Applications

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org



CCA
Common Component Architecture

Modern Scientific Software Development

- Complex codes, often coupling multiple types of physics, time or length scales, involving a broad range of computational and numerical techniques
- Different parts of the code require significantly different expertise to write (well)
- Generally written by teams rather than individuals

Time Evolution
Physics Modules
Optimization
Mesh
Adaptive Solution
Diagnostics
Discretization
Steering
Derivative Computation
Algebraic Solvers
Visualization
Data Reduction
Collaboration
Data Redistribution

Overview

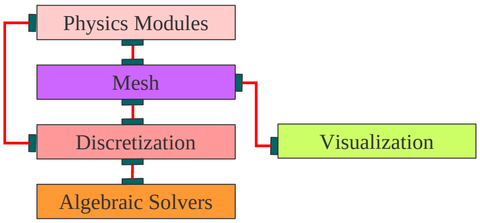
- Examples (scientific) of increasing complexity
  - Laplace equation
  - Time-dependent heat equation
  - Nonlinear reaction-diffusion system
  - Quantum chemistry
  - Climate simulation
- Tools
  - MxN parallel data redistribution
  - Performance measurement, modeling and scalability studies
- Community efforts & interface development
  - TSTT Mesh Interface effort
  - CCTTSS's Data Object Interface effort

Laplace Equation

$$\nabla^2\varphi\,(x,y) = 0 \in [0,1] \times [0,1]$$

$$\varphi(0,y)=0 \qquad \varphi(1,y)=\sin\,(2\pi y)$$

$$\delta\varphi/\delta y(x,0) = \delta\varphi/\delta y(x,1) = 0$$



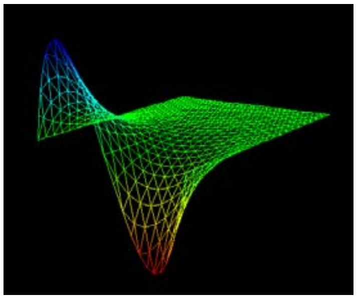
Physics Modules

Mesh

Discretization

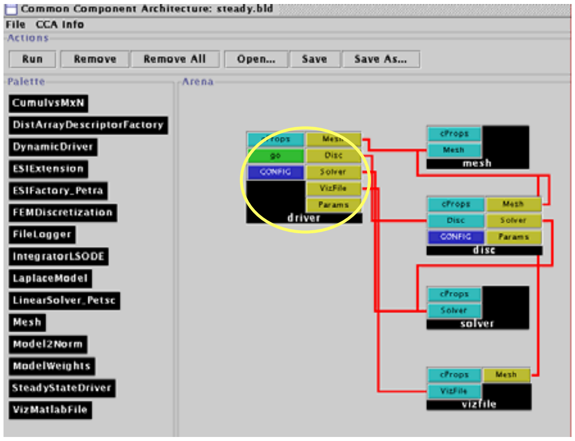
Visualization

Algebraic Solvers



Laplace Equation with Components

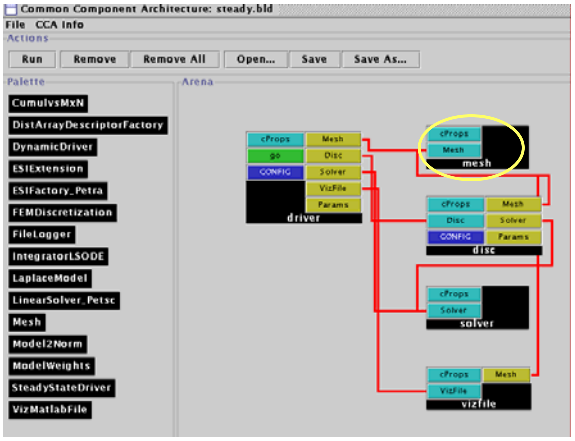
- The Driver Component
  - Responsible for the overall application flow
  - Initializes the mesh, discretization, solver and visualization components
  - Sets the physics parameters and boundary condition information



Laplace Equation with Components

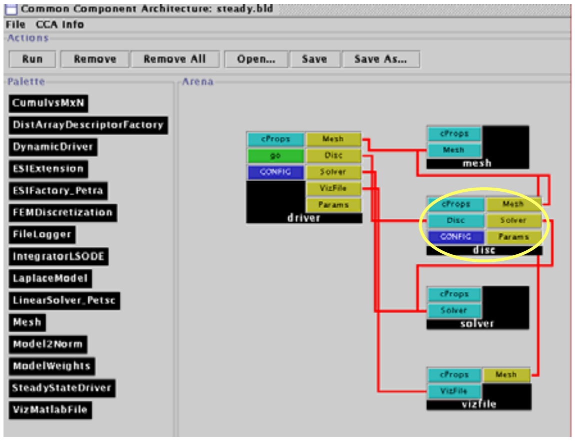
- The Driver

- The Mesh Component
  - Provides geometry, topology, and boundary information
  - Provides the ability to attach user defined data as tags to mesh entities
  - Is used by the driver, discretization and visualization components



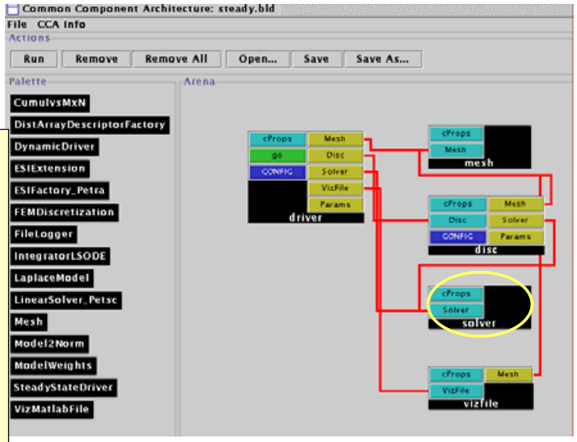
Laplace Equation with Components

- The Driver
- The Mesh
- The Discretization Component
  - Provides a finite element discretization of basic operators (gradient, Laplacian, scalar terms)
  - Driver determines which terms are included and their coefficients
  - Boundary conditions, assembly etc



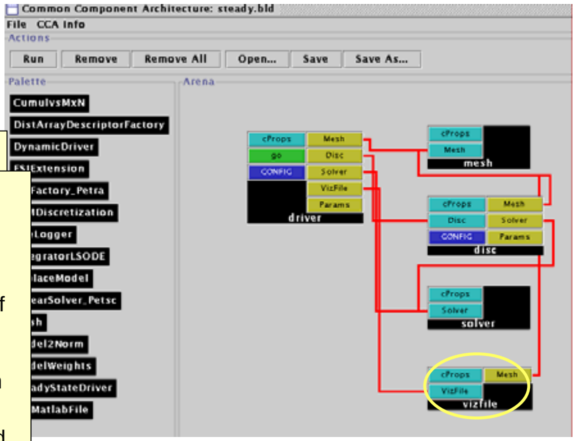
Laplace Equation with Components

- The Driver
- The Mesh
- The Discretization
- The Solver Component
  - Provides access to vector and matrix operations (e.g., create, destroy, get, set)
  - Provides a "solve" functionality for a linear operator



Laplace Equation with Components

- The Driver
- The Mesh
- The Discretization
- The Solver
- The Visualization Component
  - Uses the mesh component to print a vtk file of φ on the unstructured triangular mesh
  - Assumes user data is attached to mesh vertex entities



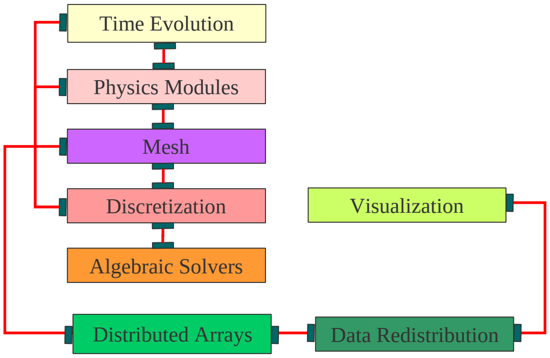
Time-Dependent Heat Equation

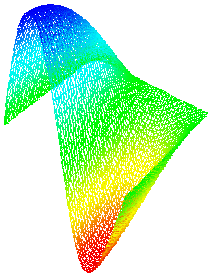
$$\delta\varphi/\delta t = \nabla^2\varphi \ (x,y,t) \in [0,1] \times [0,1]$$

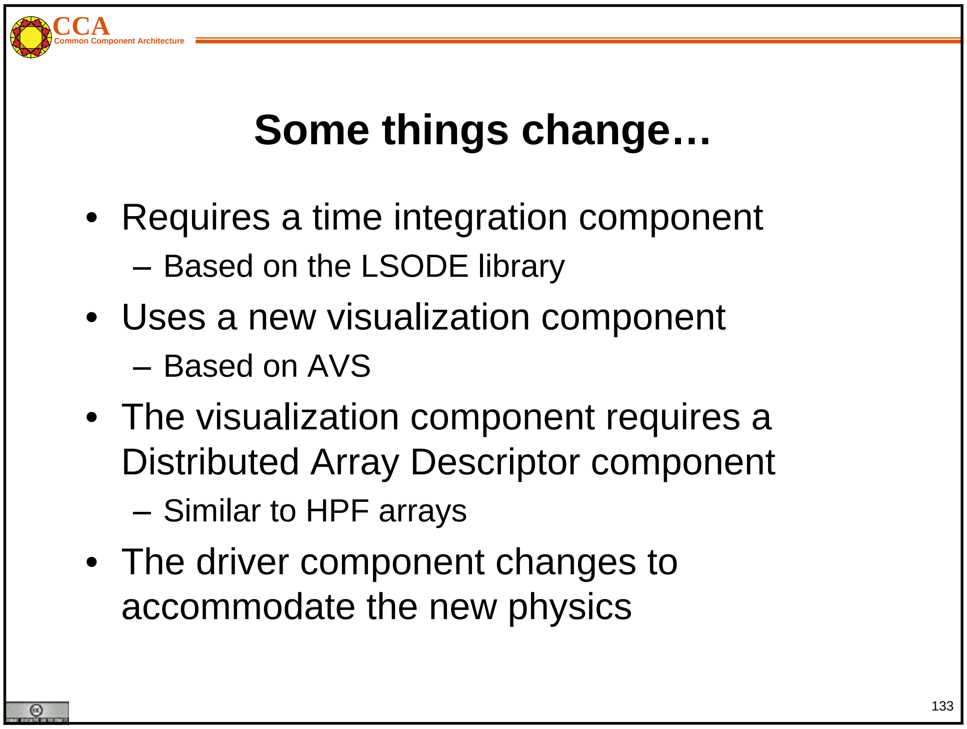
$$\varphi(0,y,t)=0 \qquad \varphi(1,y,t)=.5\sin(2\pi y)\cos(t/2)$$

$$\delta\varphi/\delta y(x,0) = \delta\varphi/\delta y(x,1) = 0$$

$$\varphi(x,y,0)=\sin(.5\pi x)\ \sin(2\pi y)$$

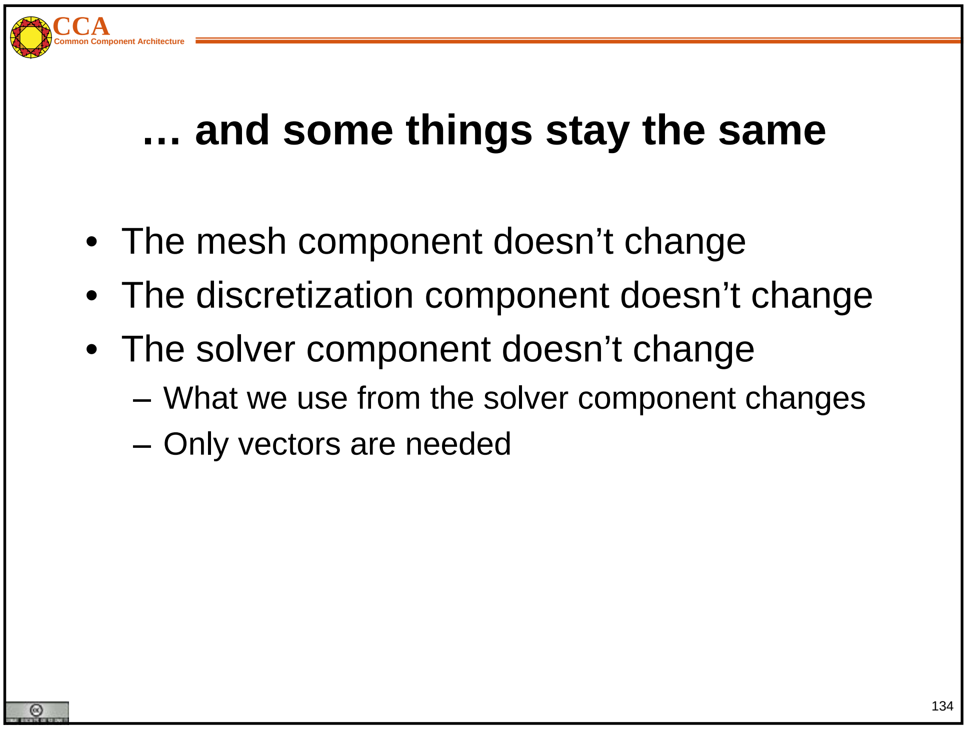






Some things change…

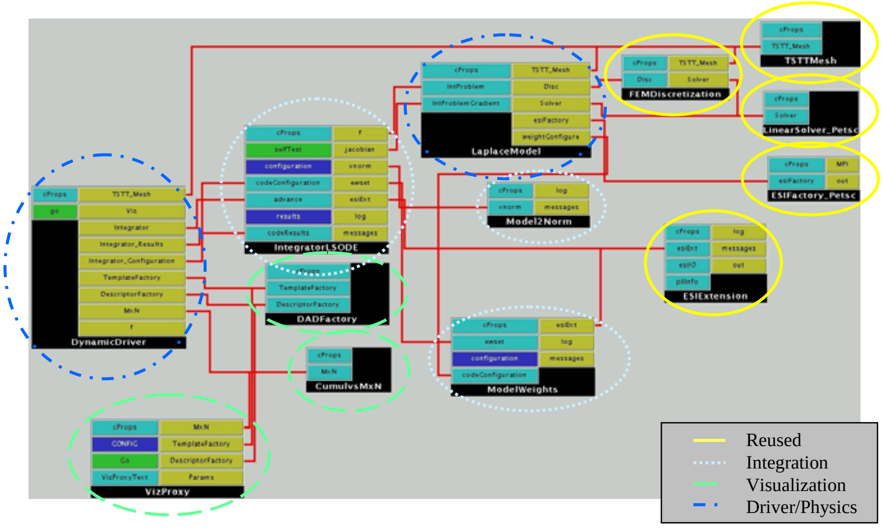
- Requires a time integration component
  - Based on the LSODE library
- Uses a new visualization component
  - Based on AVS
- The visualization component requires a Distributed Array Descriptor component
  - Similar to HPF arrays
- The driver component changes to accommodate the new physics



… and some things stay the same

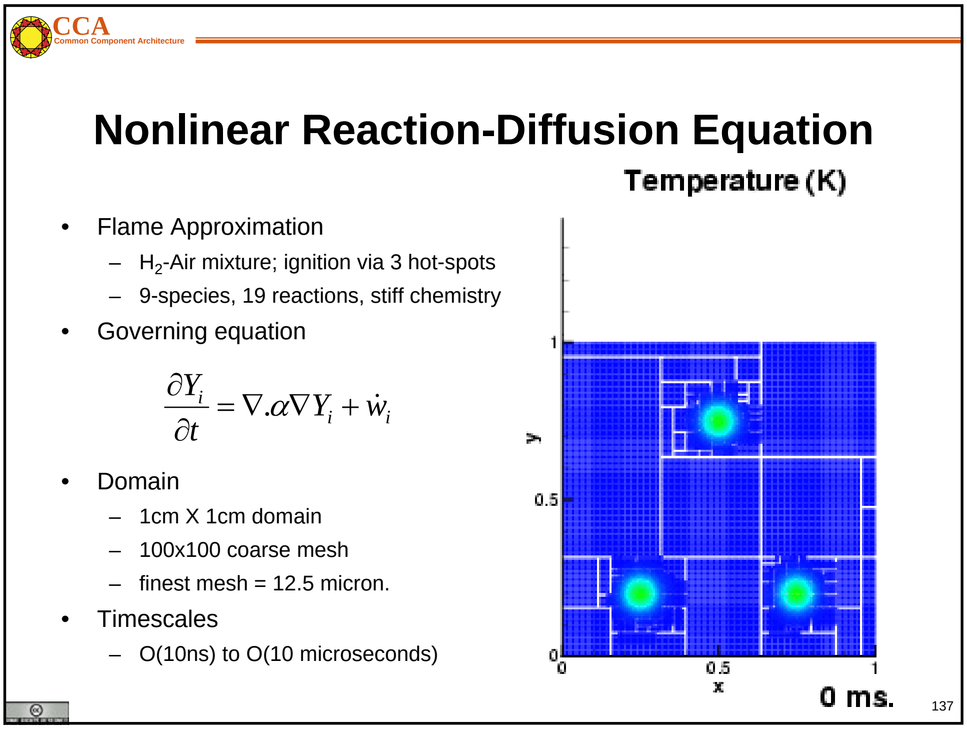
- The mesh component doesn’t change
- The discretization component doesn’t change
- The solver component doesn’t change
  - What we use from the solver component changes
  - Only vectors are needed

Heat Equation Wiring Diagram



What did this exercise teach us?

- Easy to incorporate the functionalities of components developed at other labs and institutions given a well-defined interface.
  - In fact, some components (one uses and one provides) were developed simultaneously across the country from each other after the definition of a header file.
  - Amazingly enough, they usually “just worked” when linked together (and debugged individually).
- In this case, the complexity of the component-based approach was higher than the original code complexity.
  - Partially due to the simplicity of this example
  - Partially due to the limitations of the some of the current implementations of components



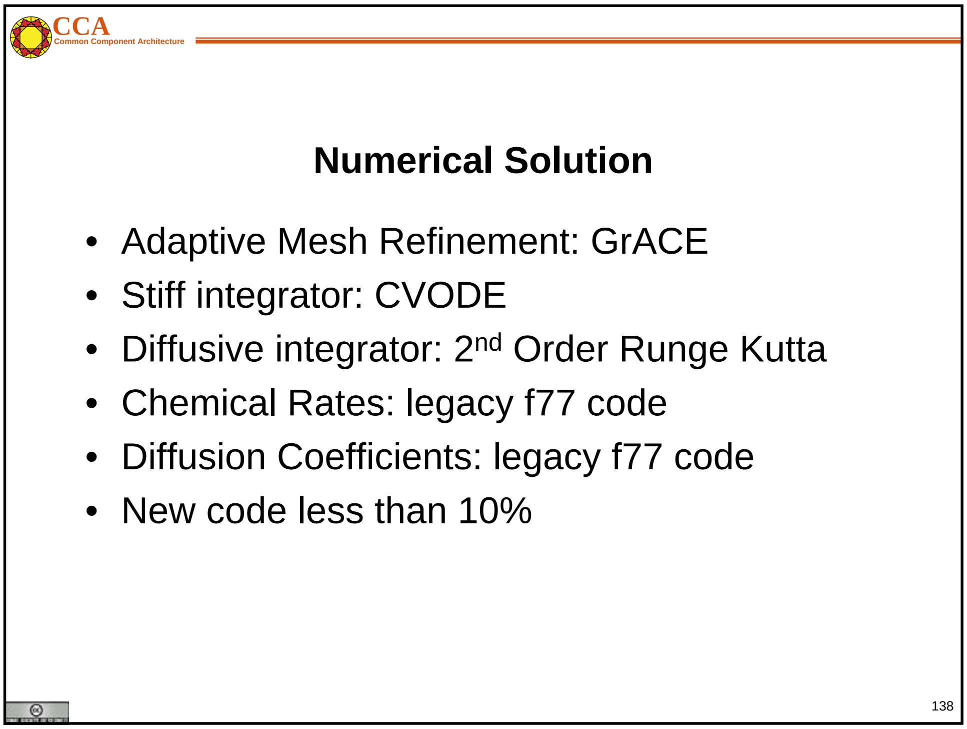
CCA Common Component Architecture

Nonlinear Reaction-Diffusion Equation

- Flame Approximation
  - H_2-Air mixture; ignition via 3 hot-spots
  - 9-species, 19 reactions, stiff chemistry
- Governing equation

$$\frac{\partial Y_i}{\partial t} = \nabla.\alpha\nabla Y_i + \dot{w}_i$$

- Domain
  - 1cm X 1cm domain
  - 100x100 coarse mesh
  - finest mesh = 12.5 micron.
- Timescales
  - O(10ns) to O(10 microseconds)

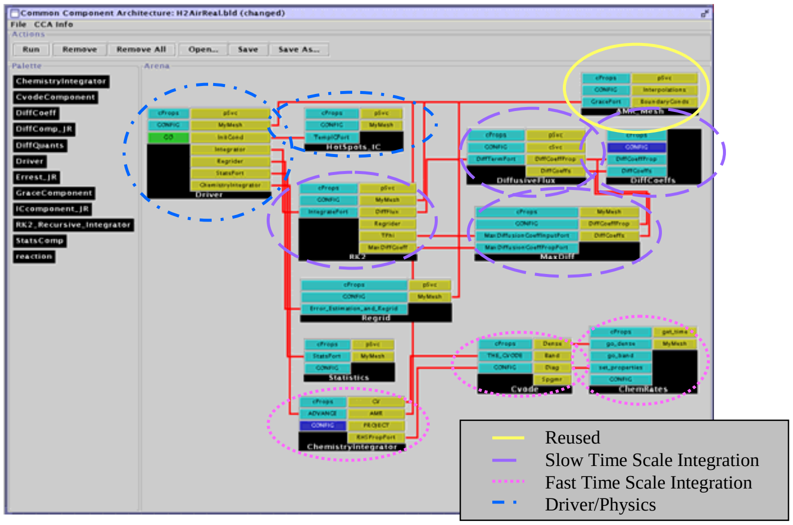


CCA Common Component Architecture

Numerical Solution

- Adaptive Mesh Refinement: GrACE
- Stiff integrator: CVODE
- Diffusive integrator: 2nd Order Runge Kutta
- Chemical Rates: legacy f77 code
- Diffusion Coefficients: legacy f77 code
- New code less than 10%

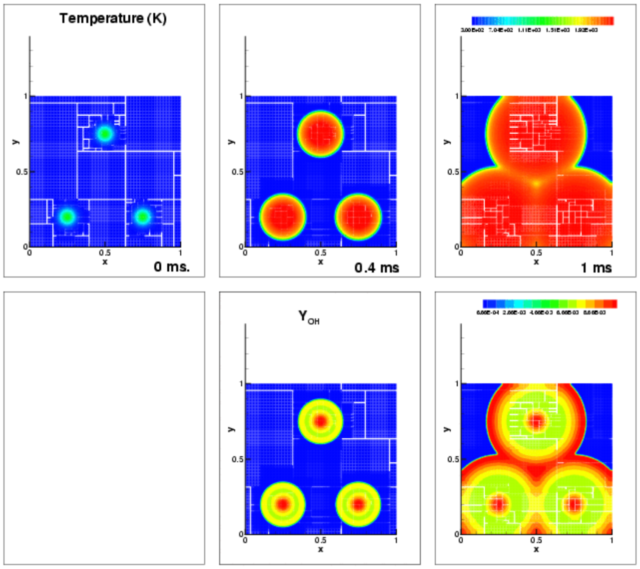
Reaction-Diffusion Wiring Diagram

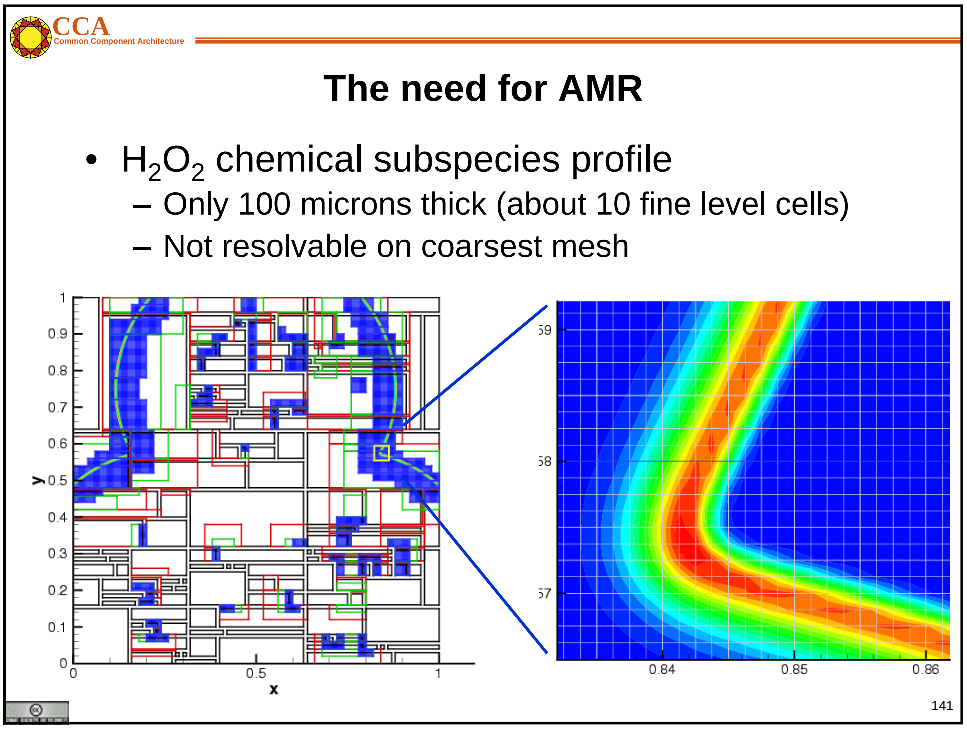


Evolution of the Solution

Temperature

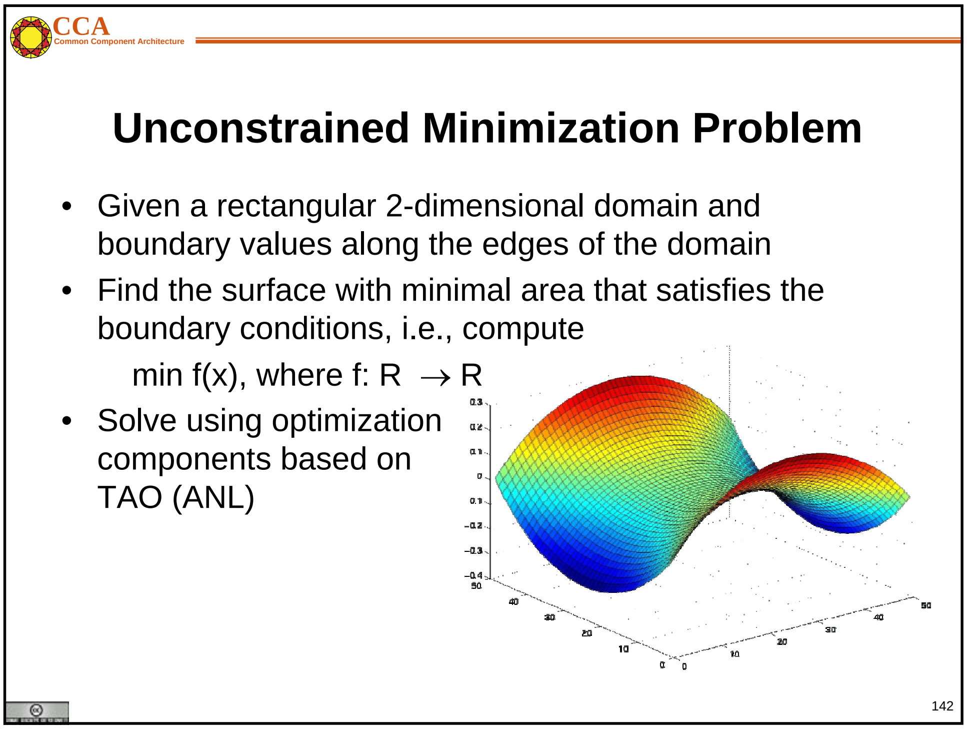
OH Profile





The need for AMR

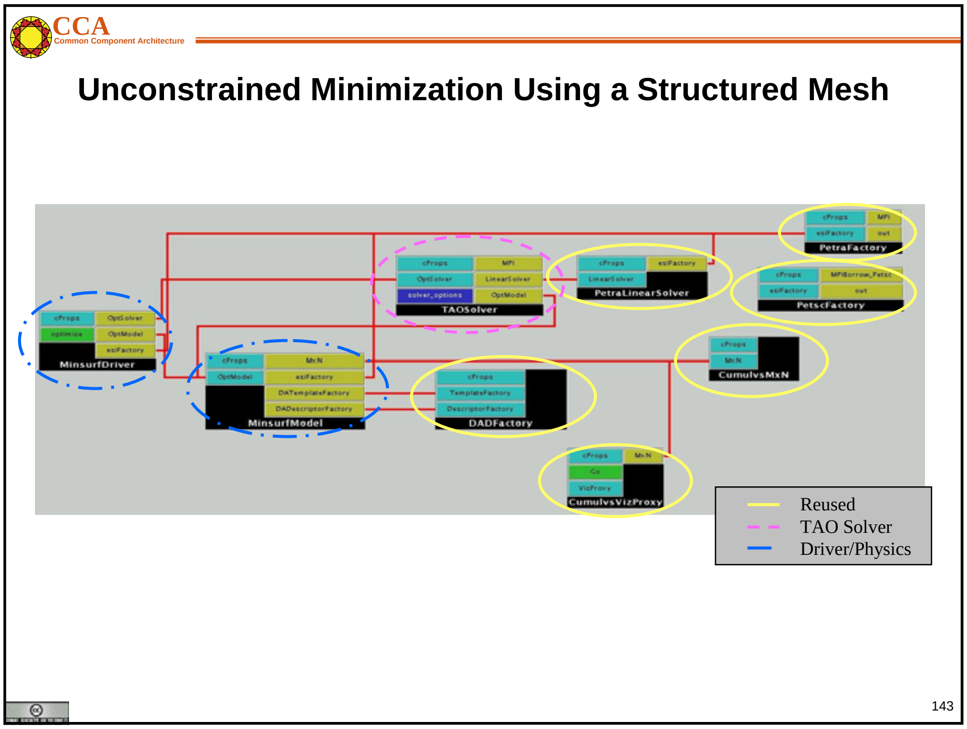
- H_2O_2 chemical subspecies profile
  - Only 100 microns thick (about 10 fine level cells)
  - Not resolvable on coarsest mesh



Unconstrained Minimization Problem

- Given a rectangular 2-dimensional domain and boundary values along the edges of the domain
- Find the surface with minimal area that satisfies the boundary conditions, i.e., compute

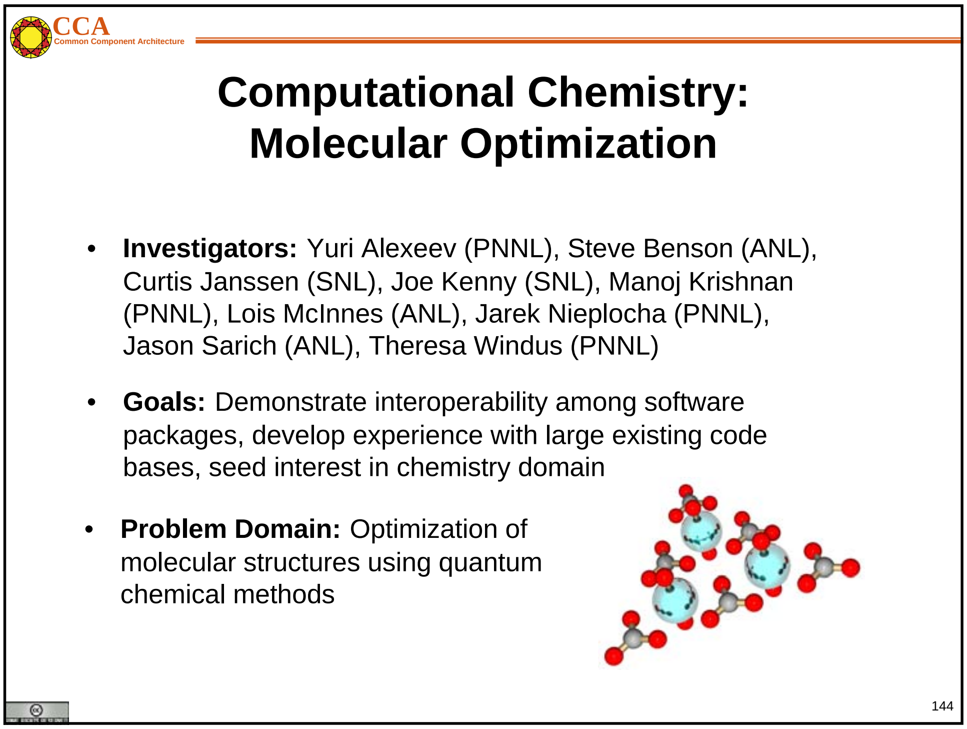
  min f(x), where f: R → R
- Solve using optimization components based on TAO (ANL)



CCA
Common Component Architecture

Unconstrained Minimization Using a Structured Mesh

🟨	Reused
🟪	TAO Solver
🟦	Driver/Physics



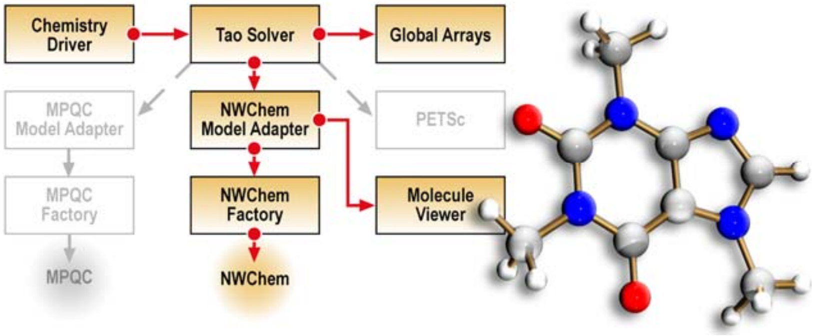
CCA
Common Component Architecture

Computational Chemistry: Molecular Optimization

- **Investigators:** Yuri Alexeev (PNNL), Steve Benson (ANL), Curtis Janssen (SNL), Joe Kenny (SNL), Manoj Krishnan (PNNL), Lois McInnes (ANL), Jarek Nieplocha (PNNL), Jason Sarich (ANL), Theresa Windus (PNNL)
- **Goals:** Demonstrate interoperability among software packages, develop experience with large existing code bases, seed interest in chemistry domain
- **Problem Domain:** Optimization of molecular structures using quantum chemical methods

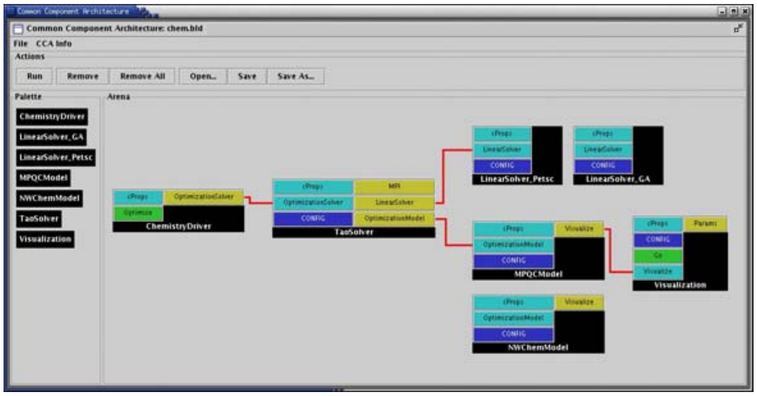
Molecular Optimization Overview

- Decouple geometry optimization from electronic structure
- Demonstrate interoperability of electronic structure components
- Build towards more challenging optimization problems, e.g., protein/ligand binding studies



Components in gray can be swapped in to create new applications with different capabilities.

Wiring Diagram for Molecular Optimization

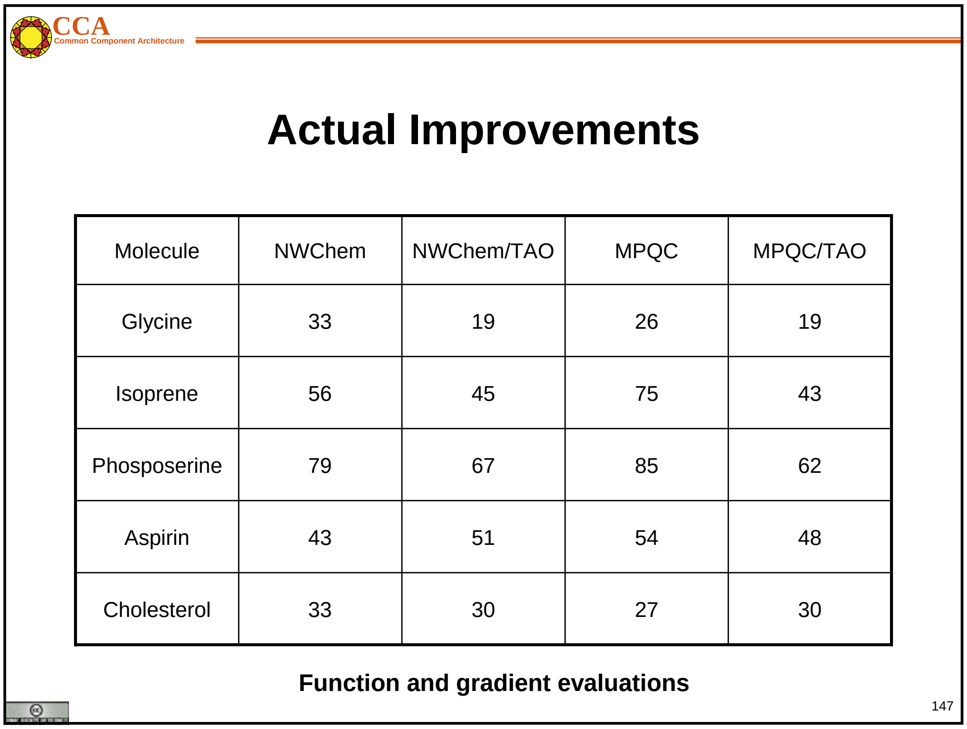


- Electronic structures components:
  - MPQC (SNL)

    http://aros.ca.sandia.gov/~cljanss/mpqc
  - NWChem (PNNL)

    http://www.emsl.pnl.gov/pub/docs/nwchem
- Optimization components: TAO (ANL) http://www.mcs.anl.gov/tao
- Linear algebra components:
  - Global Arrays (PNNL) http://www.emsl.pnl.gov:2080/docs/global/ga.html
  - PETSc (ANL)

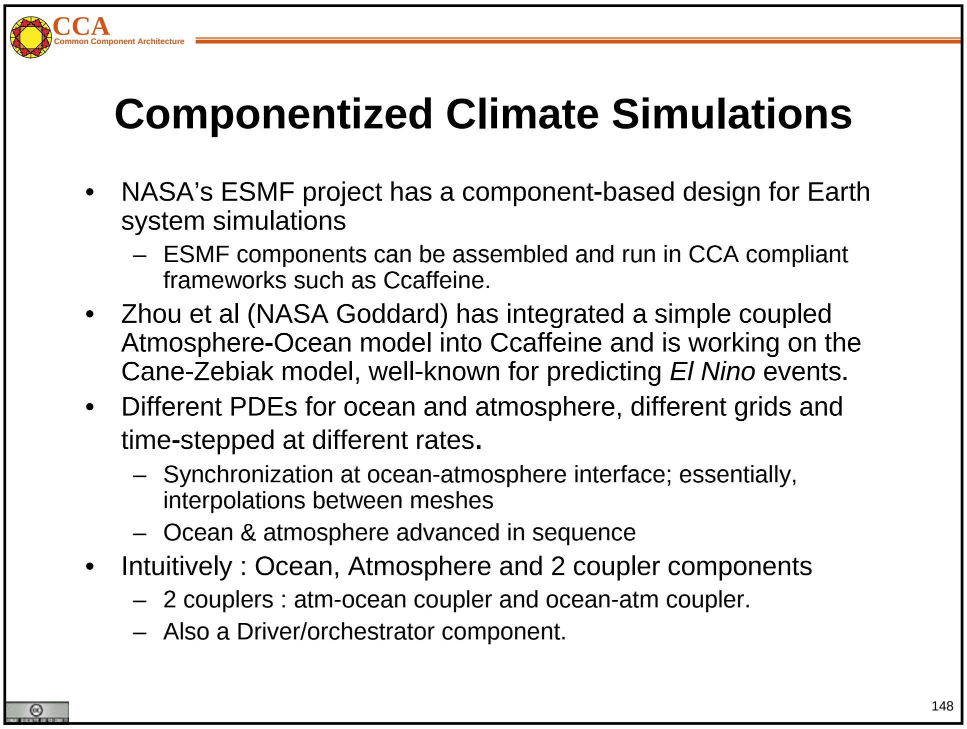
    http://www.mcs.anl.gov/petsc



Actual Improvements

Molecule	NWChem	NWChem/TAO	MPQC	MPQC/TAO
Glycine	33	19	26	19
Isoprene	56	45	75	43
Phosposerine	79	67	85	62
Aspirin	43	51	54	48
Cholesterol	33	30	27	30

Function and gradient evaluations



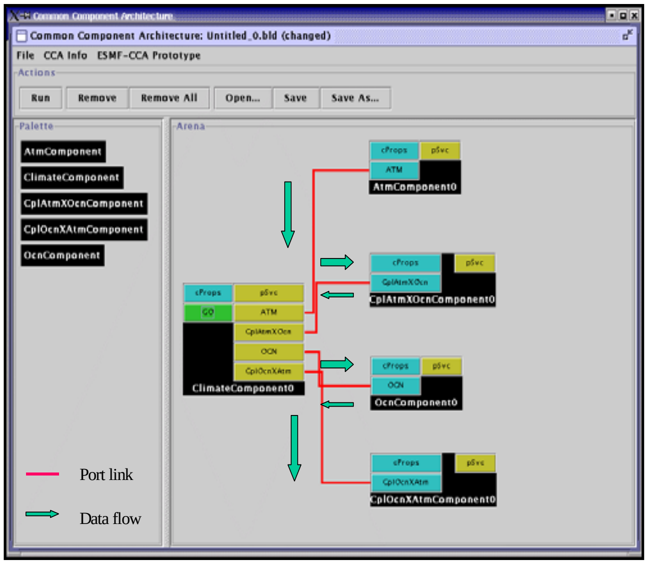
Componentized Climate Simulations

- NASA’s ESMF project has a component-based design for Earth system simulations
  - ESMF components can be assembled and run in CCA compliant frameworks such as Ccaffeine.
- Zhou et al (NASA Goddard) has integrated a simple coupled Atmosphere-Ocean model into Ccaffeine and is working on the Cane-Zebiak model, well-known for predicting *El Nino* events.
- Different PDEs for ocean and atmosphere, different grids and time-stepped at different rates.
  - Synchronization at ocean-atmosphere interface; essentially, interpolations between meshes
  - Ocean & atmosphere advanced in sequence
- Intuitively : Ocean, Atmosphere and 2 coupler components
  - 2 couplers : atm-ocean coupler and ocean-atm coupler.
  - Also a Driver/orchestrator component.

CCA Common Component Architecture

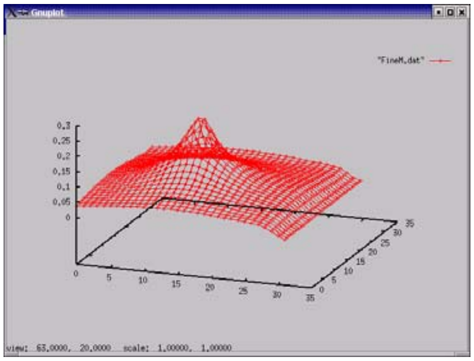
Coupled Atmosphere-Ocean Model Assembly

- **Climate Component :**
  - Schedule component coupling
- **Data flow is via pointer NOT data copy.**
  - All components in C++; run in Ccaffeine
- **Multiple ocean models with the same interface**
  - Can be selected by a user at runtime



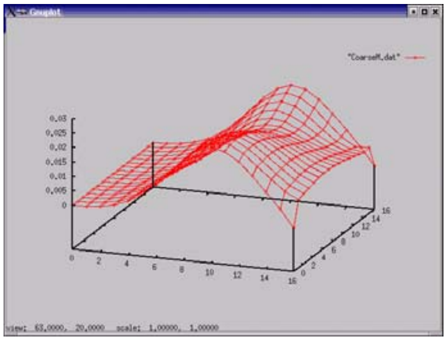
CCA Common Component Architecture

Simulation Results



A non-uniform ocean field variable (e.g., current)

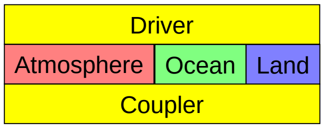
…changes a field variable (e.g.,wind) in the atmosphere !



CCA
Common Component Architecture

Concurrency At Multiple Granularities

- Certain simulations need multi-granular concurrency
  - Multiple Component Multiple Data, multi-model runs
- Usage Scenarios:
  - Model coupling (e.g. Atmosphere/Ocean)
  - General multi-physics applications
  - Software licensing issues
- Approaches
  - Run single parallel framework
    - Driver component that partitions processes and builds rest of application as appropriate (through BuilderService)
  - Run multiple parallel frameworks
    - Link through specialized communications components
    - Link as components (through AbstractFramework service; highly experimental at present)



Driver

Atmosphere | Ocean | Land

Coupler

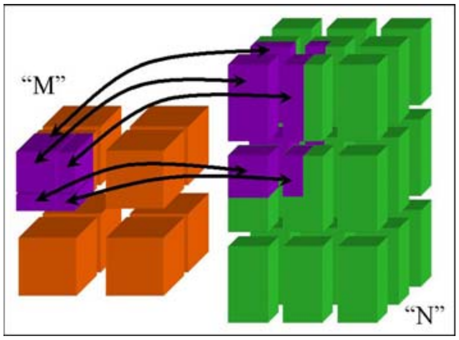
CCA
Common Component Architecture

Overview

- Examples (scientific) of increasing complexity
  - Laplace equation
  - Time-dependent heat equation
  - Nonlinear reaction-diffusion system
  - Quantum chemistry
  - Climate simulation
- Tools
  - MxN parallel data redistribution
  - Performance measurement, modeling and scalability studies
- Community efforts & interface development
  - TSTT Mesh Interface effort
  - CCTTSS's Data Object Interface effort

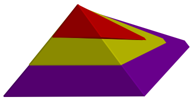
"MxN" Parallel Data Redistribution: The Problem…

- Create complex scientific simulations by coupling together multiple parallel component models
  - Share data on "M" processors with data on "N"
    - M != N ~ Distinct Resources (Pronounced "M by N")
  - Model coupling, e.g., climate, solver / optimizer
  - Collecting data for visualization
    - Mx1; increasingly MxN (parallel rendering clusters)
- Define common interface
  - Fundamental operations for any parallel data coupler
    - Full range of synchronization and communication options

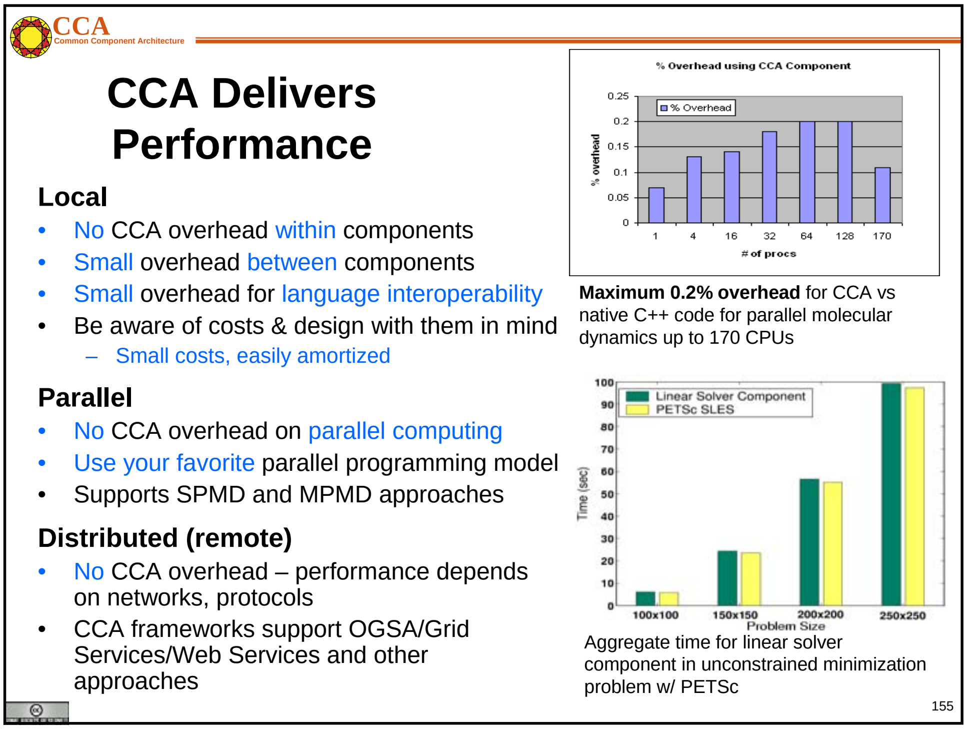


Hierarchical MxN Approach

- Basic MxN Parallel Data Exchange
  - Component implementation
  - Initial prototypes based on CUMULVS & PAWS
    - Interface generalizes features of both
- Higher-Level Coupling Functions
  - Time & grid (spatial) interpolation, flux conservation
  - Units conversions…
- "Automatic" MxN Service via Framework
  - Implicit in method invocations, "parallel RMI"



http://www.csm.ornl.gov/cca/mxn/



CCA Delivers Performance

Local

- No CCA overhead within components
- Small overhead between components
- Small overhead for language interoperability
- Be aware of costs & design with them in mind
  - Small costs, easily amortized

Parallel

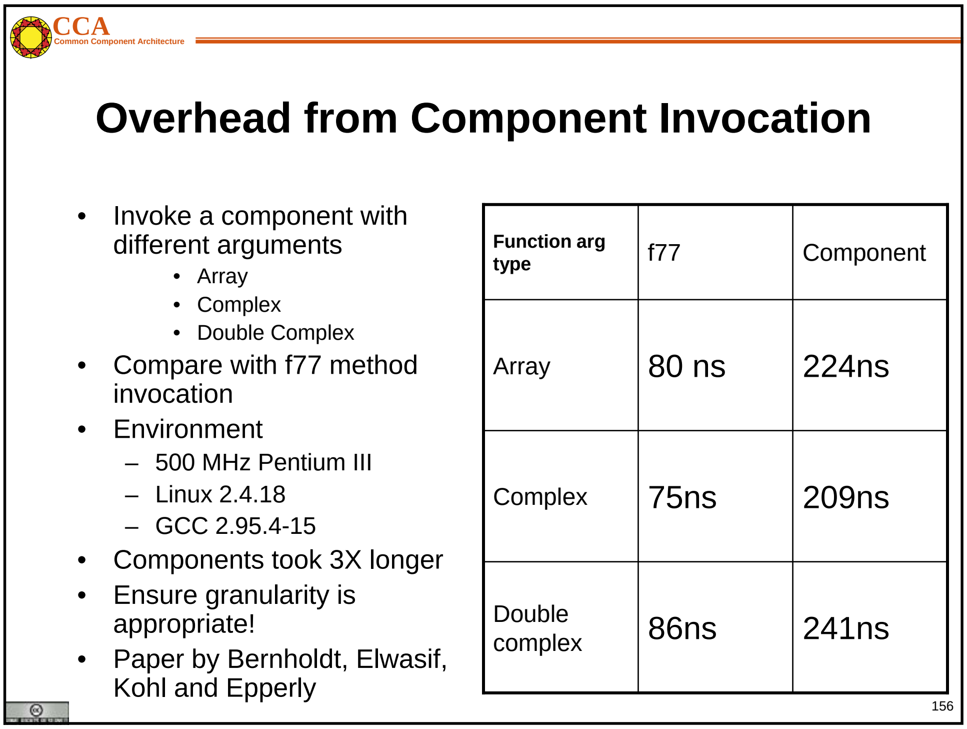
- No CCA overhead on parallel computing
- Use your favorite parallel programming model
- Supports SPMD and MPMD approaches

Distributed (remote)

- No CCA overhead – performance depends on networks, protocols
- CCA frameworks support OGSA/Grid Services/Web Services and other approaches

Maximum 0.2% overhead for CCA vs native C++ code for parallel molecular dynamics up to 170 CPUs

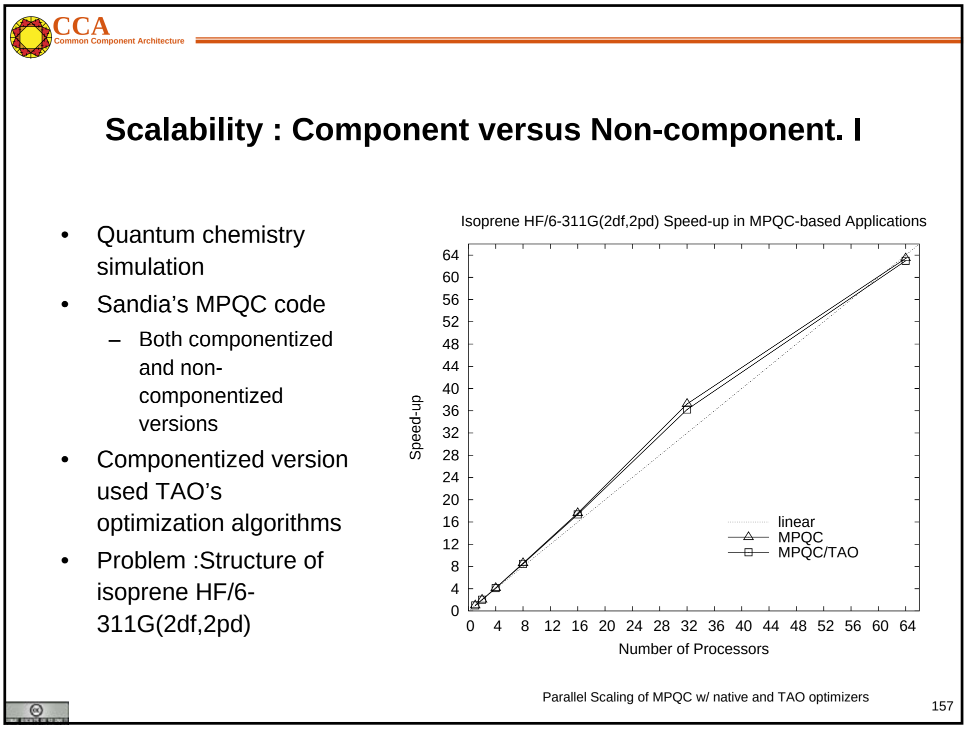
Aggregate time for linear solver component in unconstrained minimization problem w/ PETSc



Overhead from Component Invocation

- Invoke a component with different arguments
  - Array
  - Complex
  - Double Complex
- Compare with f77 method invocation
- Environment
  - 500 MHz Pentium III
  - Linux 2.4.18
  - GCC 2.95.4-15
- Components took 3X longer
- Ensure granularity is appropriate!
- Paper by Bernholdt, Elwasif, Kohl and Epperly

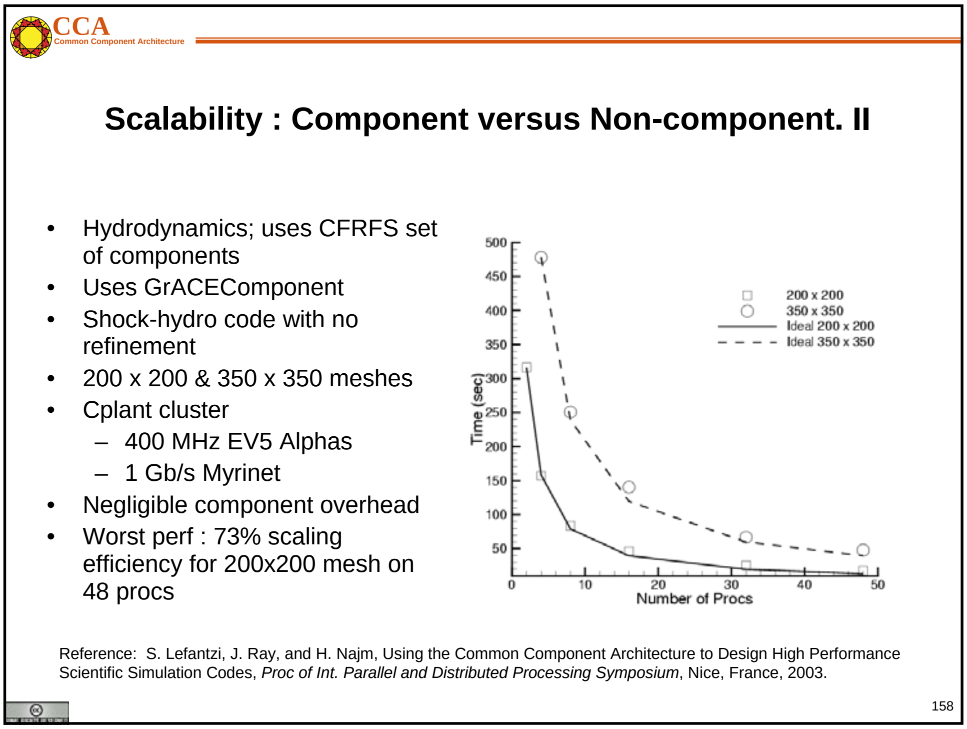
Function arg type	f77	Component
Array	80 ns	224ns
Complex	75ns	209ns
Double complex	86ns	241ns



Scalability : Component versus Non-component. I

- Quantum chemistry simulation
- Sandia's MPQC code
  - Both componentized and non-componentized versions
- Componentized version used TAO's optimization algorithms
- Problem :Structure of isoprene HF/6-311G(2df,2pd)

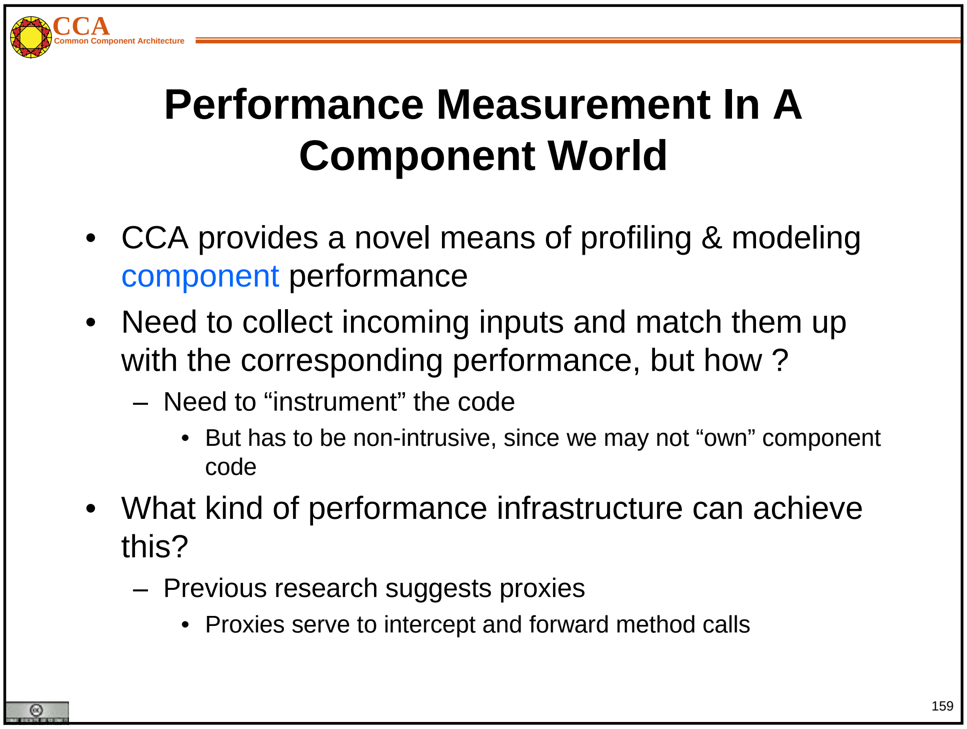
Parallel Scaling of MPQC w/ native and TAO optimizers



Scalability : Component versus Non-component. II

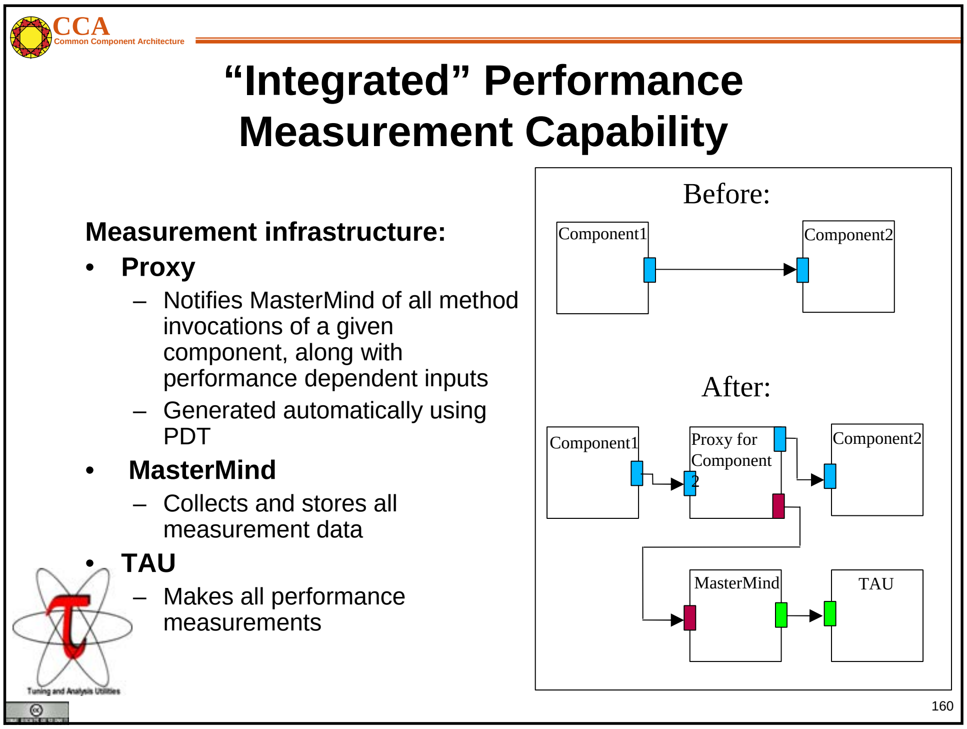
- Hydrodynamics; uses CFRFS set of components
- Uses GrACEComponent
- Shock-hydro code with no refinement
- 200 x 200 & 350 x 350 meshes
- Cplant cluster
  - 400 MHz EV5 Alphas
  - 1 Gb/s Myrinet
- Negligible component overhead
- Worst perf : 73% scaling efficiency for 200x200 mesh on 48 procs

Reference: S. Lefantzi, J. Ray, and H. Najm, Using the Common Component Architecture to Design High Performance Scientific Simulation Codes, *Proc of Int. Parallel and Distributed Processing Symposium*, Nice, France, 2003.



Performance Measurement In A Component World

- CCA provides a novel means of profiling & modeling component performance
- Need to collect incoming inputs and match them up with the corresponding performance, but how ?
  - Need to “instrument” the code
    - But has to be non-intrusive, since we may not “own” component code
- What kind of performance infrastructure can achieve this?
  - Previous research suggests proxies
    - Proxies serve to intercept and forward method calls

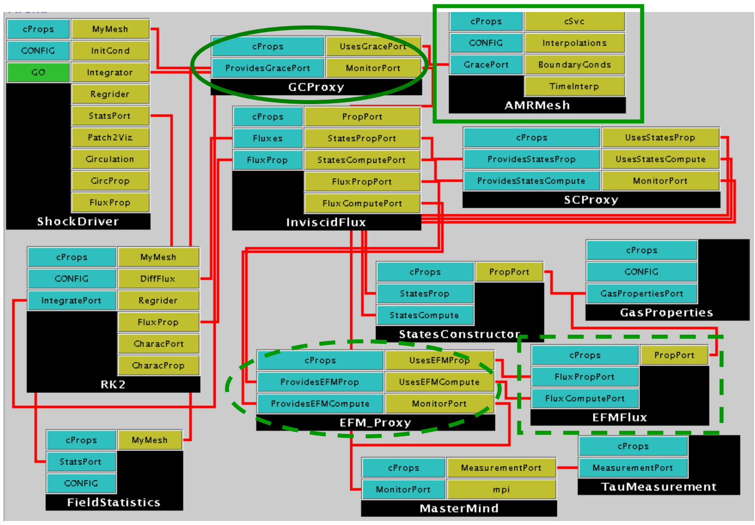


“Integrated” Performance Measurement Capability

Measurement infrastructure:

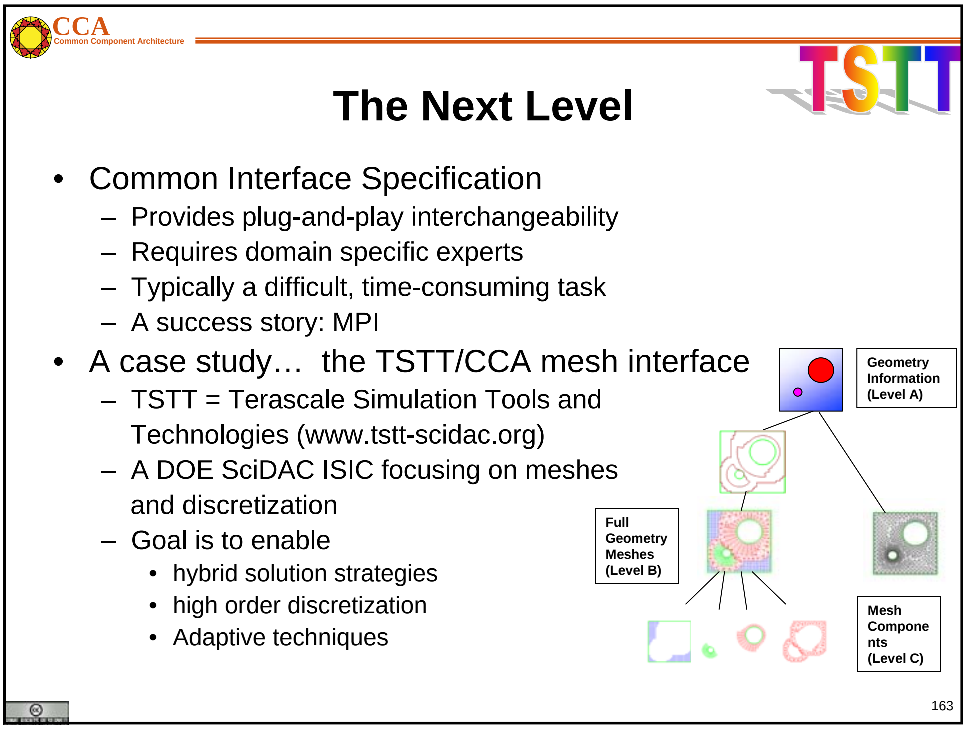
- **Proxy**
  - Notifies MasterMind of all method invocations of a given component, along with performance dependent inputs
  - Generated automatically using PDT
- **MasterMind**
  - Collects and stores all measurement data
- **TAU**
  - Makes all performance measurements

Component Application With Proxies



Overview

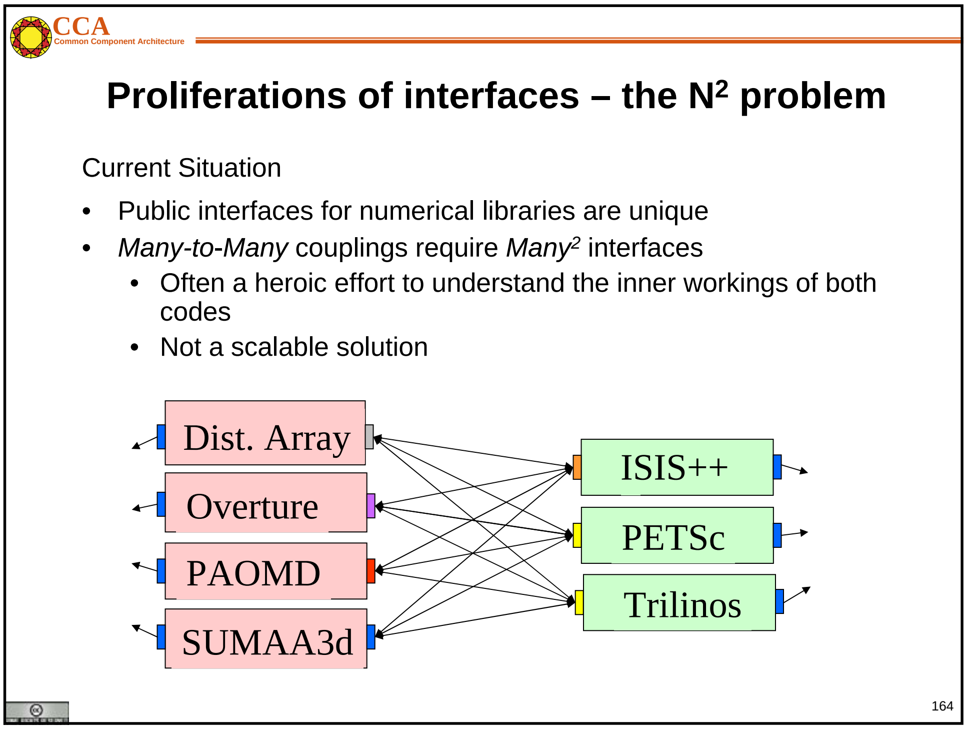
- Examples (scientific) of increasing complexity
  - Laplace equation
  - Time-dependent heat equation
  - Nonlinear reaction-diffusion system
  - Quantum chemistry
  - Climate simulation
- Tools
  - MxN parallel data redistribution
  - Performance measurement, modeling and scalability studies
- ➡ Community efforts & interface development
  - TSTT Mesh Interface effort
  - CCTTSS's Data Object Interface effort



The Next Level

- Common Interface Specification
  - Provides plug-and-play interchangeability
  - Requires domain specific experts
  - Typically a difficult, time-consuming task
  - A success story: MPI
- A case study… the TSTT/CCA mesh interface
  - TSTT = Terascale Simulation Tools and Technologies (www.tstt-scidac.org)
  - A DOE SciDAC ISIC focusing on meshes and discretization
  - Goal is to enable
    - hybrid solution strategies
    - high order discretization
    - Adaptive techniques

Geometry Information (Level A)
Full Geometry Meshes (Level B)
Mesh Compone nts (Level C)



Proliferations of interfaces – the N^2 problem

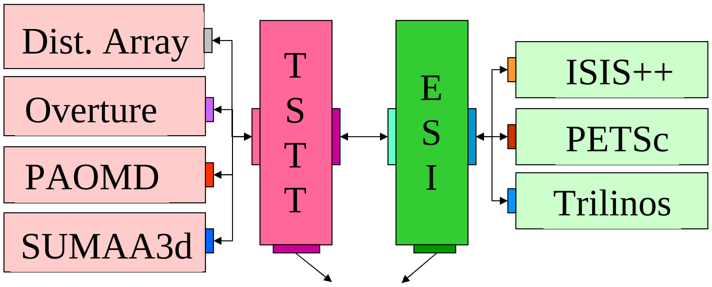
Current Situation

- Public interfaces for numerical libraries are unique
- *Many-to-Many* couplings require *Many*^{2} interfaces
  - Often a heroic effort to understand the inner workings of both codes
  - Not a scalable solution

Common Interface Specification

Reduces the *Many-to-Many* problem to a *Many-to-One* problem

- Allows interchangeability and experimentation
- Challenges
  - Interface agreement
  - Functionality limitations
  - Maintaining performance



Dist. Array

Overture

PAOMD

SUMAA3d

TSTT

ESI

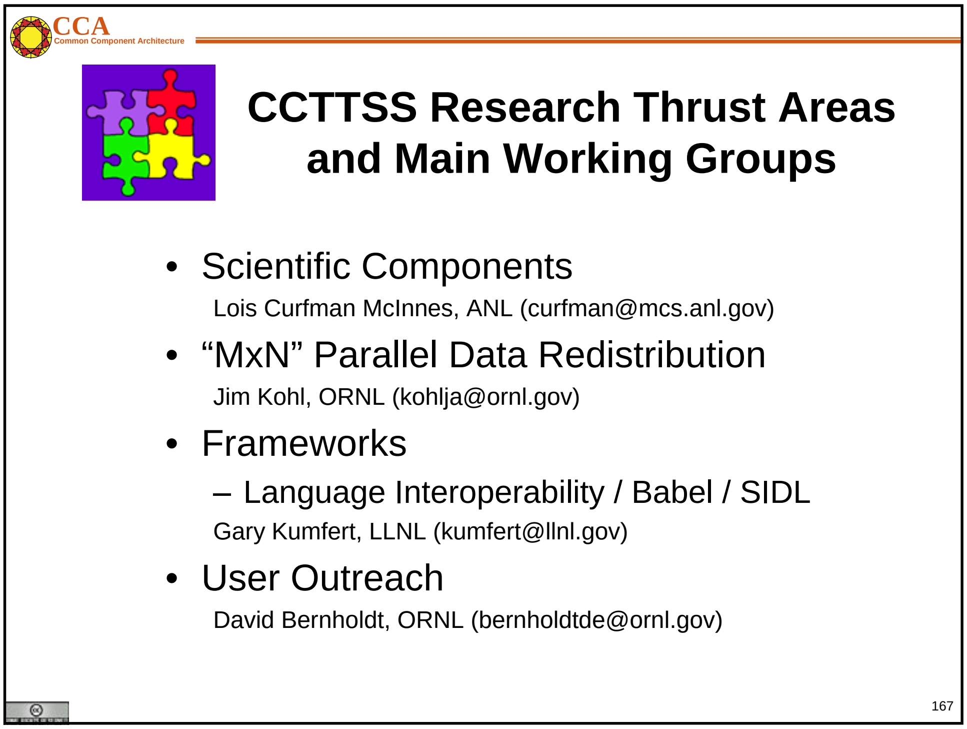
ISIS++

PETSc

Trilinos

TSTT Philosophy

- Create a small set of interfaces that existing packages can support
  - AOMD, CUBIT, Overture, GrACE, …
  - Enable both interchangeability and interoperability
- Balance performance and flexibility
- Work with a large tool provider and application community to ensure applicability
  - Tool providers: TSTT and CCA SciDAC centers
  - Application community: SciDAC and other DOE applications



CCA
Common Component Architecture

CCTTSS Research Thrust Areas and Main Working Groups

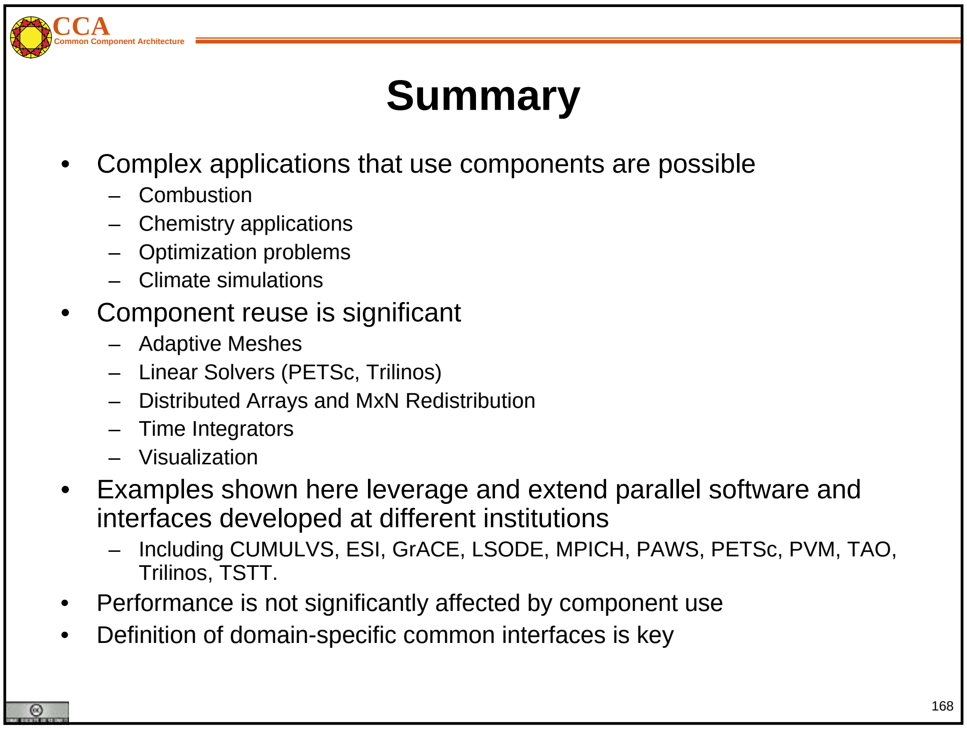
- Scientific Components

  Lois Curfman McInnes, ANL (curfman@mcs.anl.gov)
- “MxN” Parallel Data Redistribution

  Jim Kohl, ORNL (kohlja@ornl.gov)
- Frameworks
  - Language Interoperability / Babel / SIDL

  Gary Kumfert, LLNL (kumfert@llnl.gov)
- User Outreach

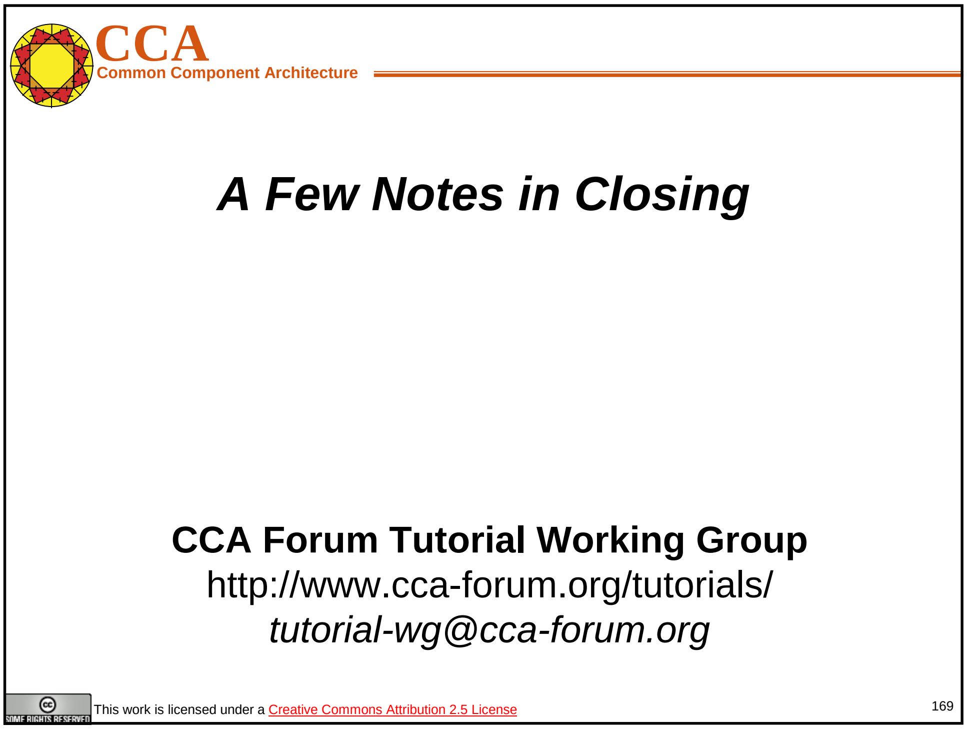
  David Bernholdt, ORNL (bernholdtde@ornl.gov)



CCA
Common Component Architecture

Summary

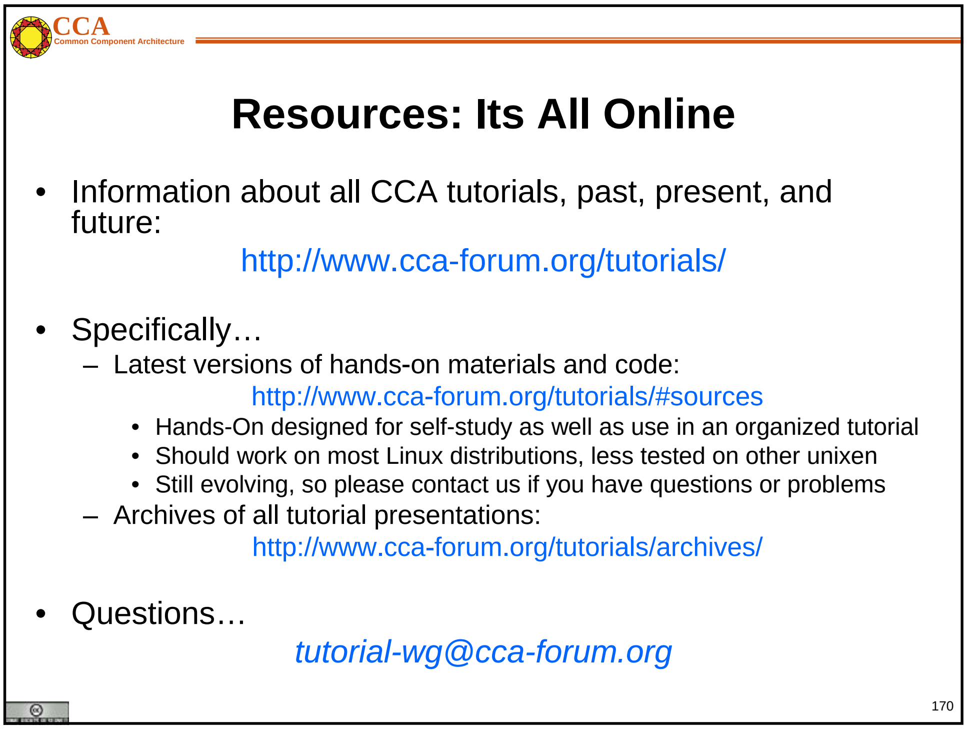
- Complex applications that use components are possible
  - Combustion
  - Chemistry applications
  - Optimization problems
  - Climate simulations
- Component reuse is significant
  - Adaptive Meshes
  - Linear Solvers (PETSc, Trilinos)
  - Distributed Arrays and MxN Redistribution
  - Time Integrators
  - Visualization
- Examples shown here leverage and extend parallel software and interfaces developed at different institutions
  - Including CUMULVS, ESI, GrACE, LSODE, MPICH, PAWS, PETSc, PVM, TAO, Trilinos, TSTT.
- Performance is not significantly affected by component use
- Definition of domain-specific common interfaces is key



CCA
Common Component Architecture

A Few Notes in Closing

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org



CCA
Common Component Architecture

Resources: Its All Online

- Information about all CCA tutorials, past, present, and future:

  http://www.cca-forum.org/tutorials/

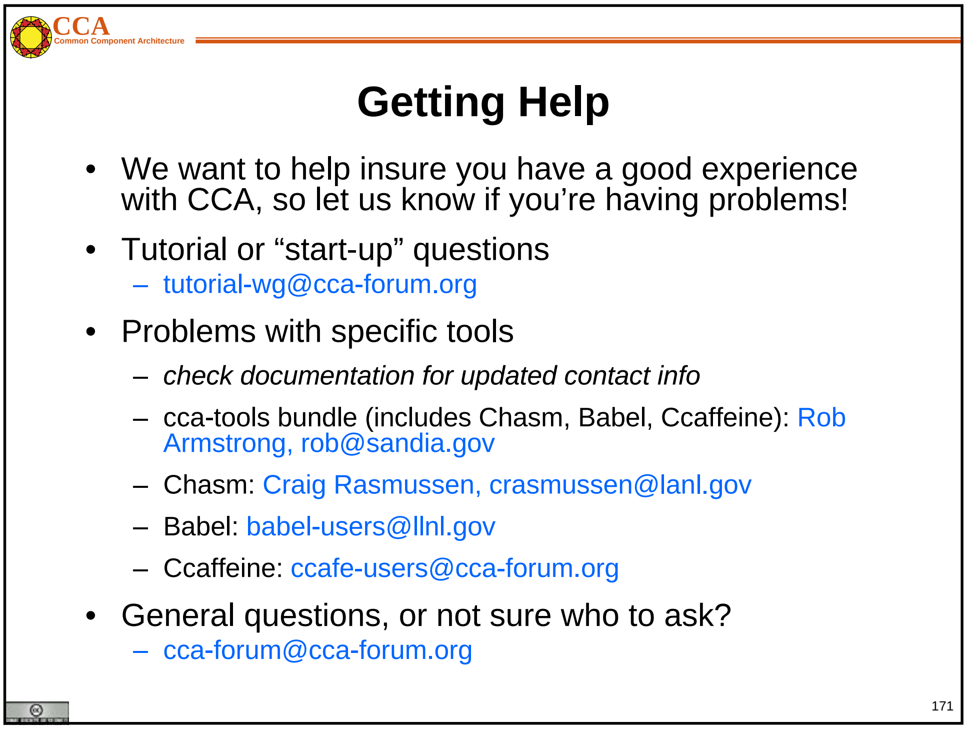
- Specifically…
  - Latest versions of hands-on materials and code:

    http://www.cca-forum.org/tutorials/#sources
    - Hands-On designed for self-study as well as use in an organized tutorial
    - Should work on most Linux distributions, less tested on other unixen
    - Still evolving, so please contact us if you have questions or problems
  - Archives of all tutorial presentations:

    http://www.cca-forum.org/tutorials/archives/

- Questions…

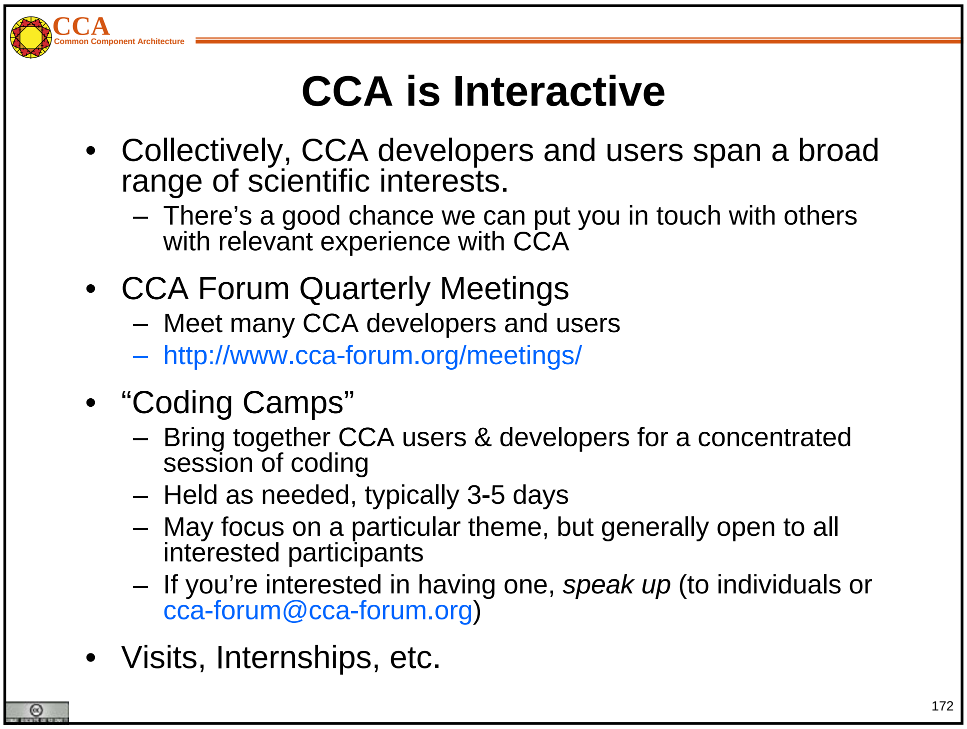
  tutorial-wg@cca-forum.org



CCA
Common Component Architecture

Getting Help

- We want to help insure you have a good experience with CCA, so let us know if you're having problems!
- Tutorial or "start-up" questions
  - tutorial-wg@cca-forum.org
- Problems with specific tools
  - *check documentation for updated contact info*
  - cca-tools bundle (includes Chasm, Babel, Ccaffeine): Rob Armstrong, rob@sandia.gov
  - Chasm: Craig Rasmussen, crasmussen@lanl.gov
  - Babel: babel-users@llnl.gov
  - Ccaffeine: ccafe-users@cca-forum.org
- General questions, or not sure who to ask?
  - cca-forum@cca-forum.org



CCA
Common Component Architecture

CCA is Interactive

- Collectively, CCA developers and users span a broad range of scientific interests.
  - There's a good chance we can put you in touch with others with relevant experience with CCA
- CCA Forum Quarterly Meetings
  - Meet many CCA developers and users
  - http://www.cca-forum.org/meetings/
- "Coding Camps"
  - Bring together CCA users & developers for a concentrated session of coding
  - Held as needed, typically 3-5 days
  - May focus on a particular theme, but generally open to all interested participants
  - If you're interested in having one, *speak up* (to individuals or cca-forum@cca-forum.org)
- Visits, Internships, etc.

CCA
Common Component Architecture

Acknowledgements: Tutorial Working Group

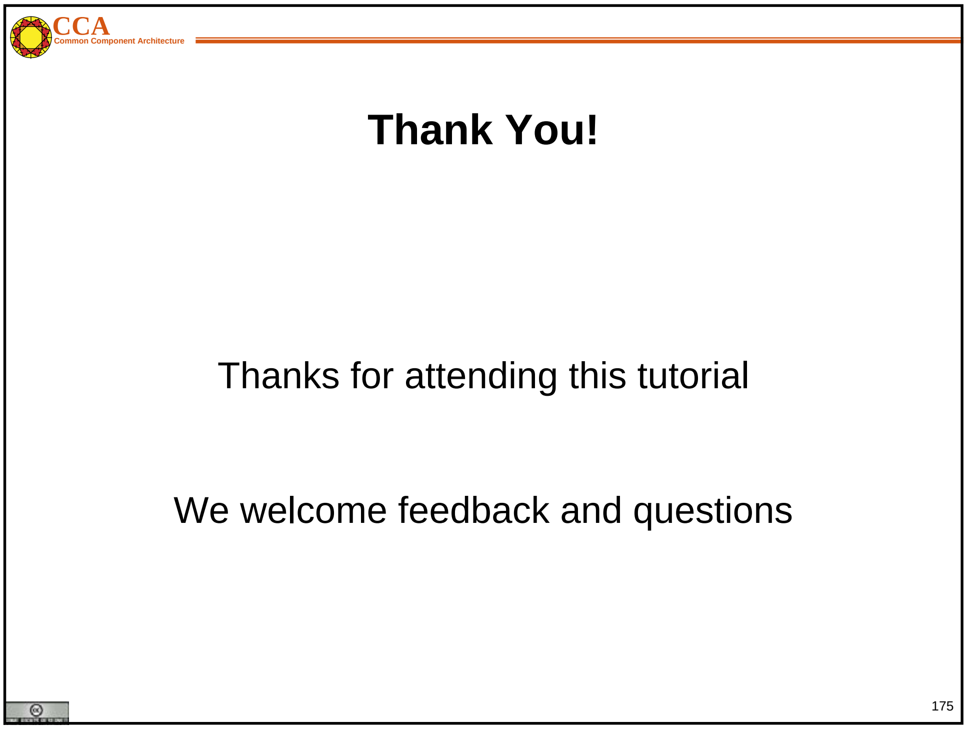
- **People:** Benjamin A. Allan, Rob Armstrong, David E. Bernholdt, Randy Bramley, Tamara L. Dahlgren, Lori Freitag Diachin, Wael Elwasif, Tom Epperly, Madhusudhan Govindaraju, Ragib Hasan, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Alan Morris, Boyana Norris, Craig Rasmussen, Jaideep Ray, Sameer Shende, Torsten Wilde, Shujia Zhou
- **Institutions:** ANL, Binghamton U, Indiana U, JPL, LANL, LLNL, NASA/Goddard, ORNL, SNL, U Illinois, U Oregon
- **Computer facilities** provided by the Computer Science Department and University Information Technology Services of Indiana University, supported in part by NSF grants CDA-9601632 and EIA-0202048.

CCA
Common Component Architecture

Acknowledgements: The CCA

- **ANL** –Steve Benson, Jay Larson, Ray Loy, Lois Curfman McInnes, Boyana Norris, Everest Ong, Jason Sarich…
- **Binghamton University** - Madhu Govindaraju, Michael Lewis, …
- **Indiana University** - Randall Bramley, Dennis Gannon, …
- **JPL** – Dan Katz, …
- **LANL** - Craig Rasmussen, Matt Sotille, …
- **LLNL** – Tammy Dahlgren, Lori Freitag Diachin, Tom Epperly, Scott Kohn, Gary Kumfert, …
- **NASA/Goddard** – Shujia Zhou
- **ORNL** - David Bernholdt, Wael Elwasif, Jim Kohl, Torsten Wilde, …
- **PNNL** - Jarek Nieplocha, Theresa Windus, …
- **SNL** - Rob Armstrong, Ben Allan, Lori Freitag Diachin, Curt Janssen, Jaideep Ray, …
- **Tech-X Corp. –** Johan Carlsson, Svetlana Shasharina, Ovsei Volberg, Nanbor Wang
- **University of Oregon** – Allen Malony, Sameer Shende, …
- **University of Utah** - Steve Parker, …

and many more… without whom we wouldn't have much to talk about!



CCA
Common Component Architecture

Thank You!

Thanks for attending this tutorial

We welcome feedback and questions