CCA
Common Component Architecture

Welcome to the Common Component Architecture Tutorial

ACTS Collection Workshop

20 August 2010

CCA Forum Tutorial Working Group

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

tutorial-wg@cca-forum.org

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, 2010, http://www.cca-forum.org/tutorials/**

About the Printed Notes

- The printed (or PDF) version of these presentations includes additional slides marked *“Supplementary material for handouts”*
- Additional material to address questions sometimes raised, or provide more detail on a topic
- We are happy to discuss this material if asked

Introductions

- David E. Bernholdt (Oak Ridge National Laboratory)
- Dietmar Ebner (Lawrence Livermore National Laboratory)
- Wael R. Elwasif (Oak Ridge National Laboratory)
- Sameer Shende (U. Oregon)

Who We Are: 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 tools for component-based software development, components, and component applications
- Open membership, quarterly meetings…

General mailing list: *cca-forum@cca-forum.org*

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

CCA
Common Component Architecture

Agenda & Table of Contents

These notes are available online at:
http://www.cca-forum.org/tutorials/archives/2010/tutorial-2010-08-20

What You Should Take Away from this Tutorial

- The concepts of component-based software development (CBSD)
  - Help organize, design, and manage large, complex software systems
- The Common Component Architecture (CCA) is a CBSD approach specifically designed for high-performance computational science and engineering
  - The CCA is not the *only* way to do CBSD
- Hands-on experience using and creating simple components using the CCA tools

Introduction to HPC Component Software

CCA Forum Tutorial Working Group

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

tutorial-wg@cca-forum.org

Managing Code Complexity

Some Common Situations:

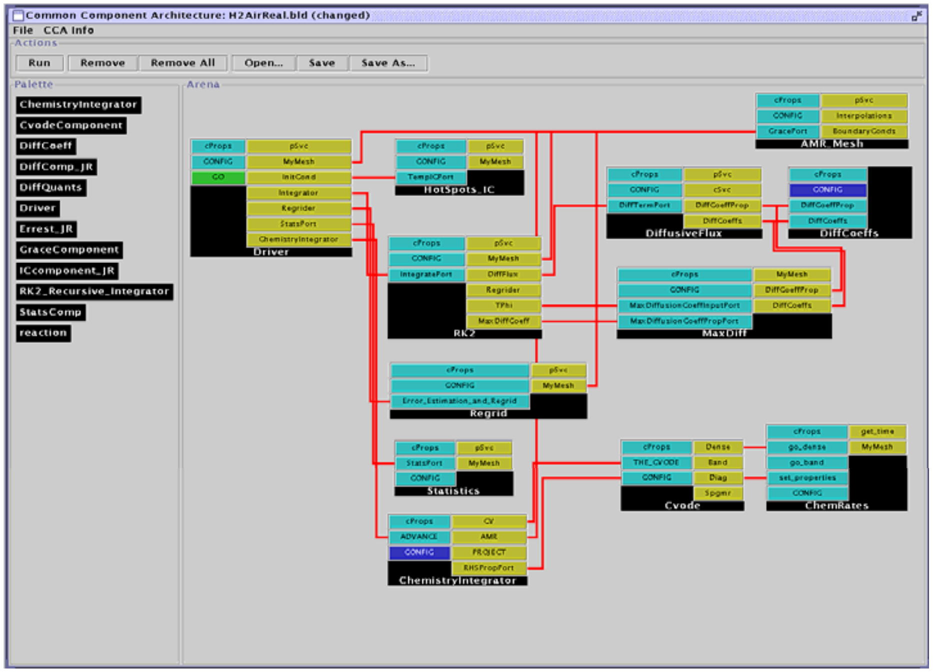
- Your code is so large and complex it has become fragile and hard to keep running
- You have a simple code, and you want to extend its capabilities – rationally
- You want to develop a computational "toolkit"
  - Many modules that can be assembled in different ways to perform different scientific calculations
  - 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:

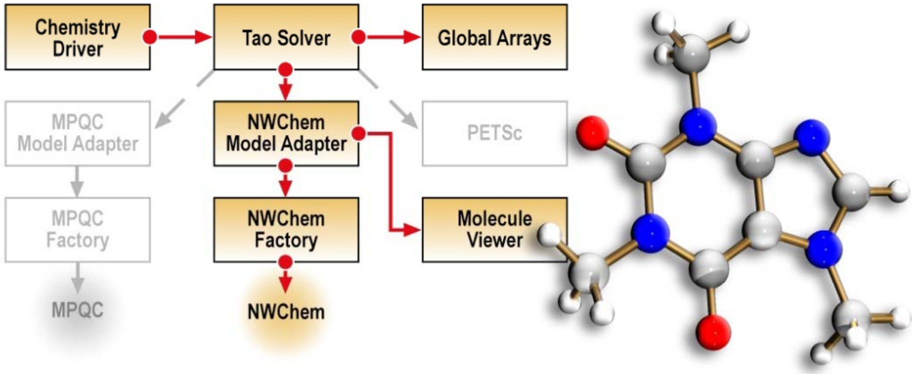
- Many (geographically distributed) developers creating a large software system
  - Hard to coordinate, different parts of the software don't work together as required
- Groups of developers with different specialties
- Forming communities to standardize interfaces or share code

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

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:

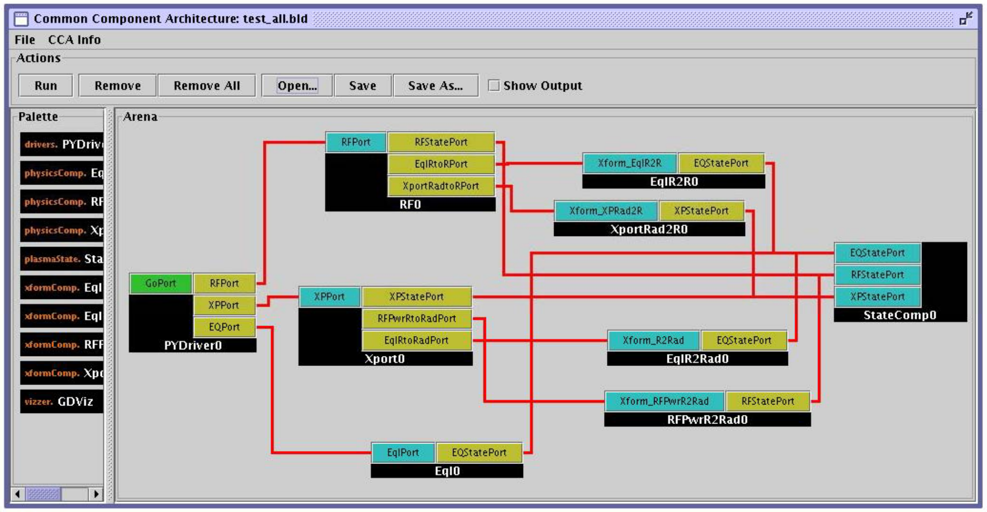
- Your application makes use of numerous third-party libraries
  - Some of which interact (version dependencies)
- You want to develop a simulation in which your code is coupled with others
  - They weren't designed with this coupling in mind
  - They must remain usable separately too
  - They are all 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: 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

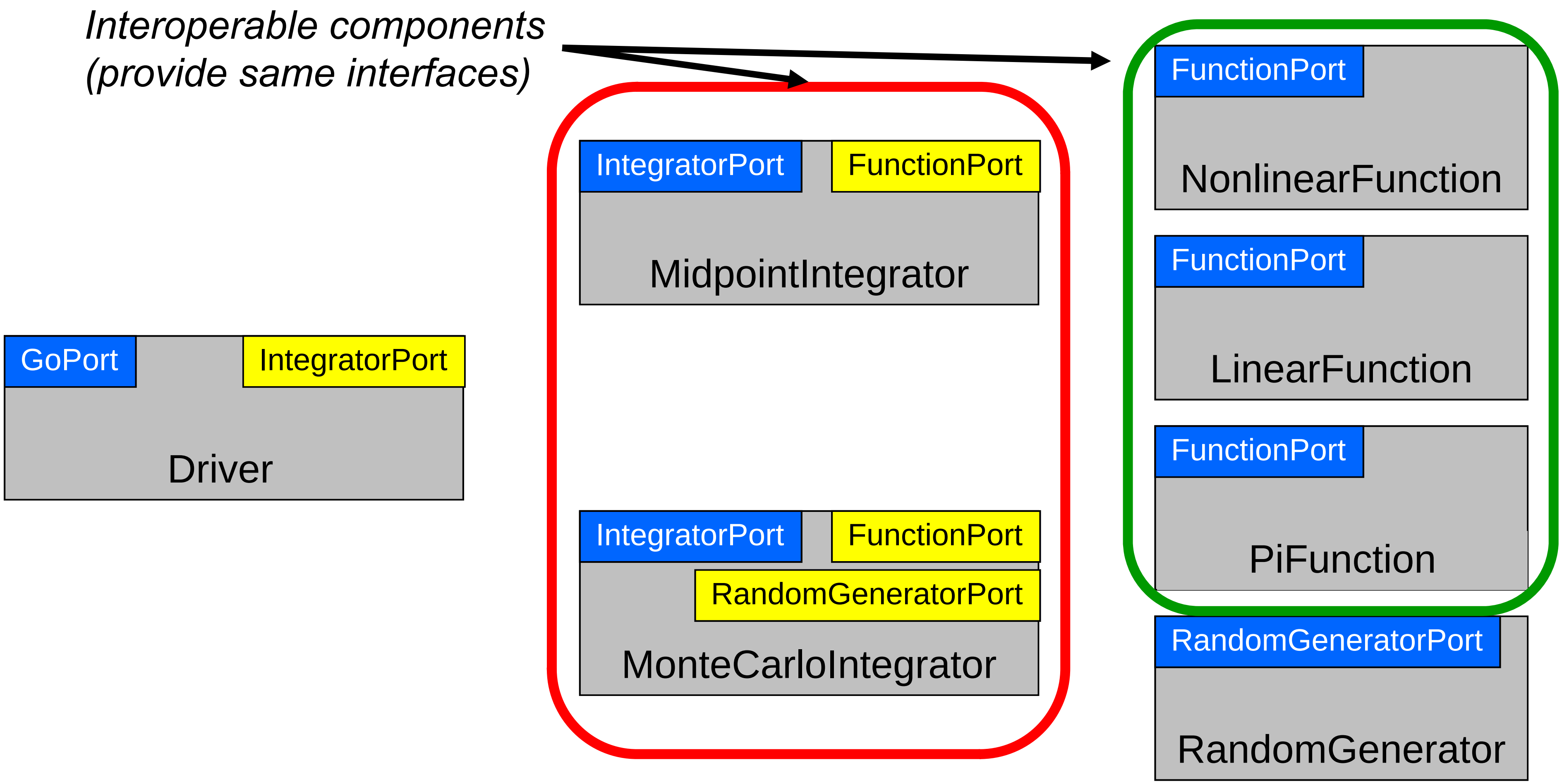
What are Components?

- 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

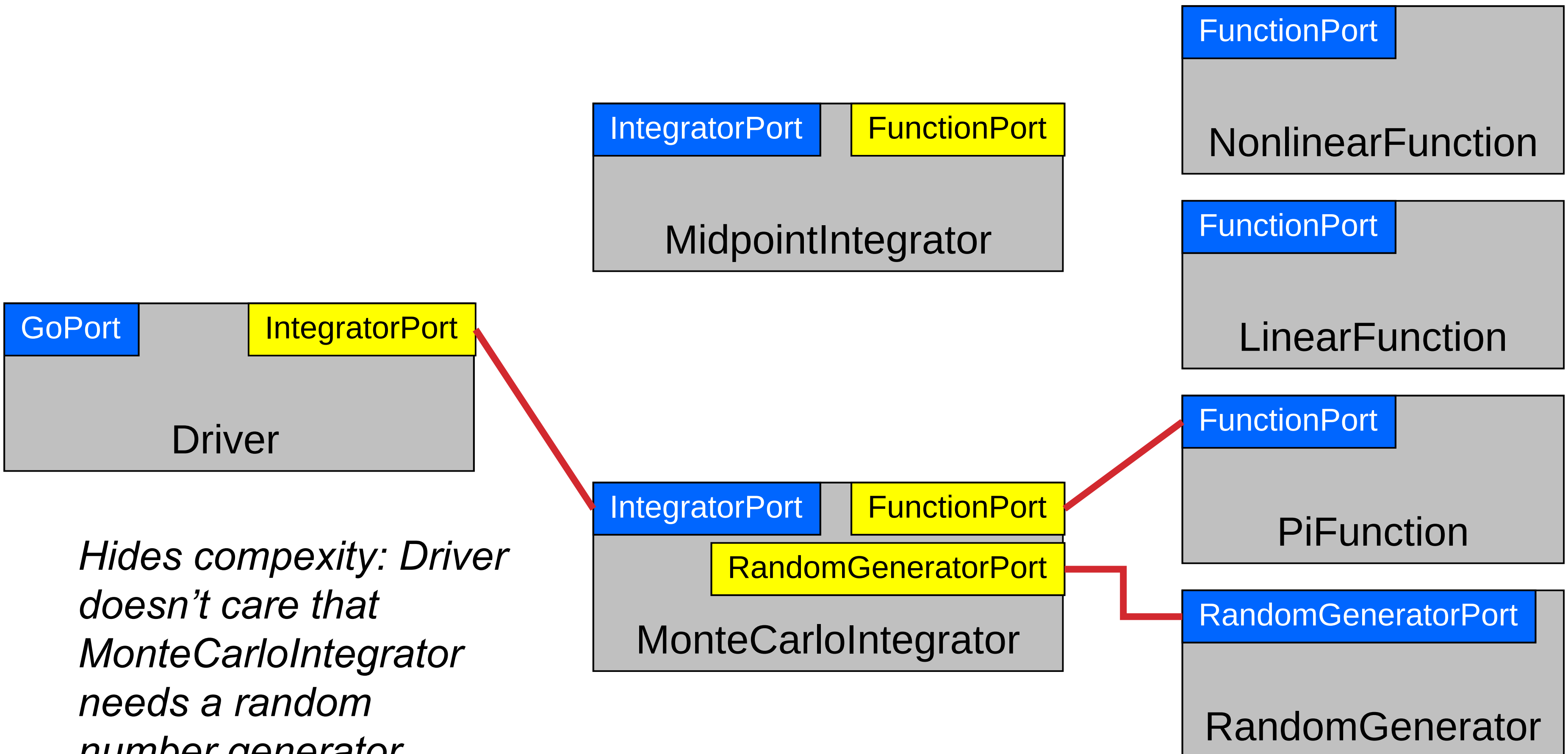
What is a Component Architecture?

- 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

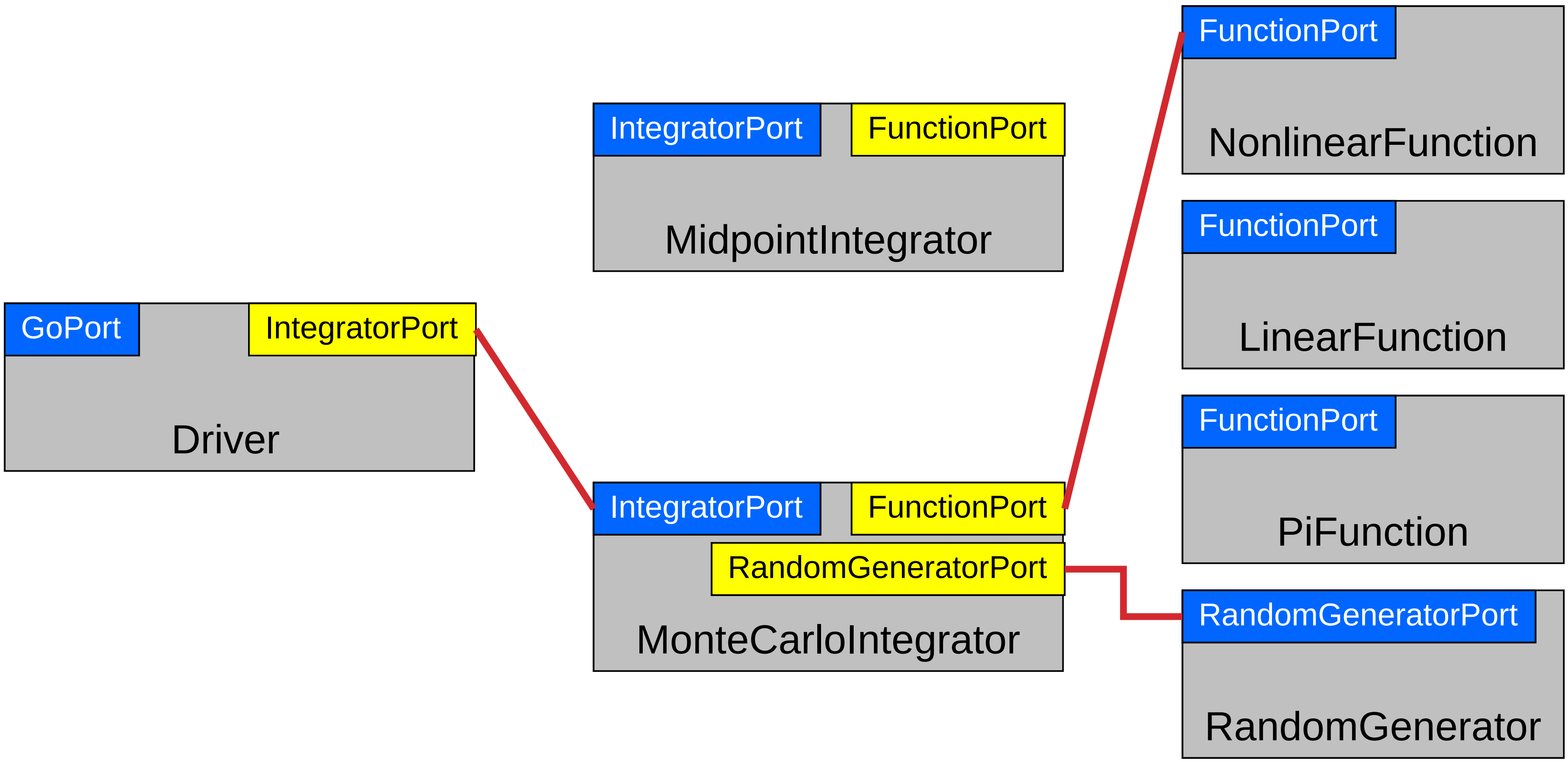


An Application
Built from the Provided Components



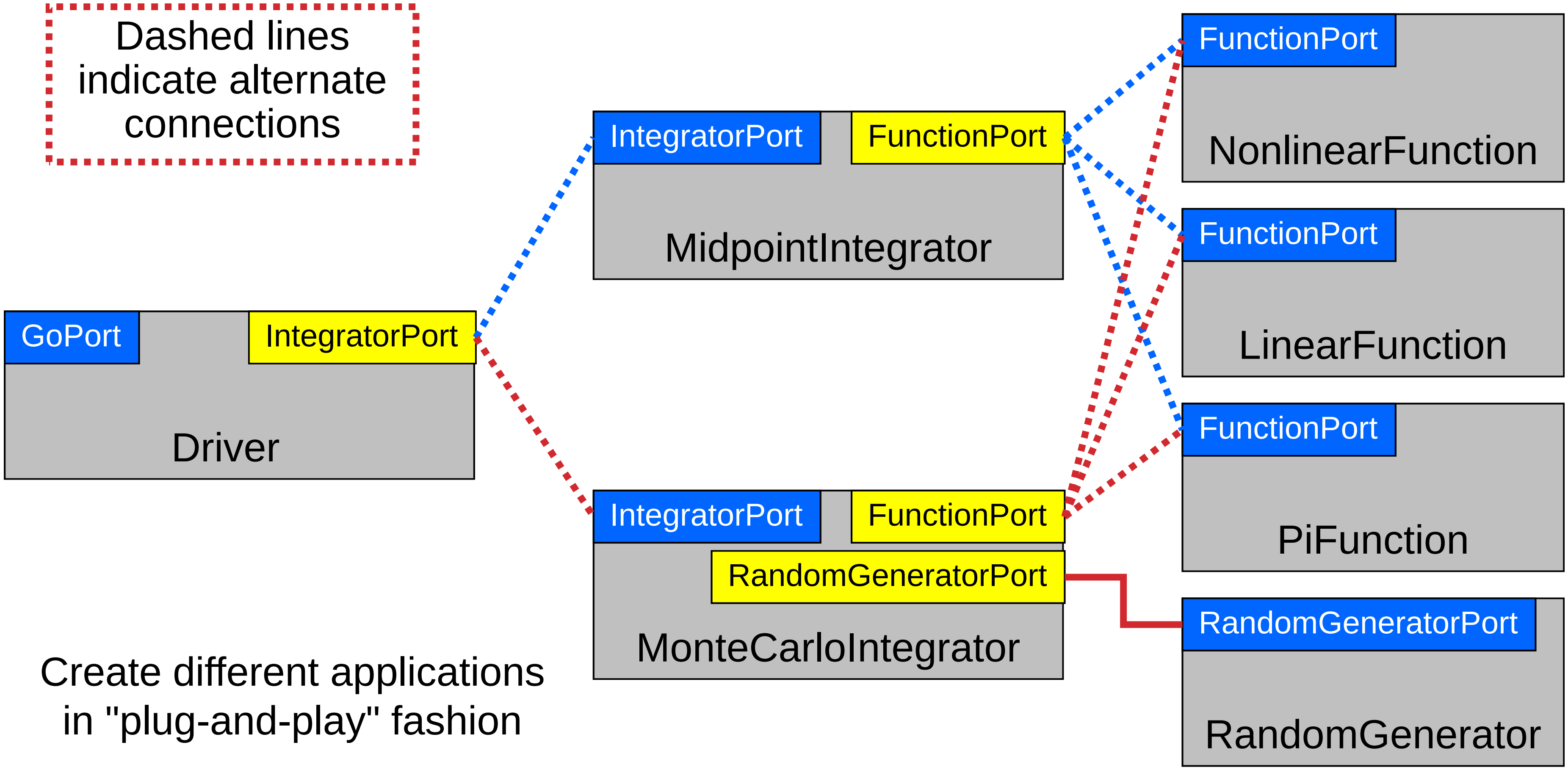
Hides compexity: Driver doesn't care that MonteCarloIntegrator needs a random number generator

Another Application…



CCA Common Component Architecture

And Many More…



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

Comparison of Application Development Approaches

Characteristics	Monolithic Simulation Code	Simulation Frameworks	Library -Based	Component -Based
Support for specific workflows and information flows	High	High	Low	Low
Flexibility w.r.t. workflow and information flow	Low	Medium	High	High
User-level extensibility	Low	Medium	High	High
Ease of incorporation of outside code (code reuse)	Low	Low-Medium	Medium	High
Ease of experimentation	Low	Medium	Medium	High
Amount of new code required to create a complete simulation	Low	Medium	High	High (reuse can reduce)
Breadth of current “ecosystem” for “plugins”	Low	Medium	High	Low (but growing)
Ease of coupling simulations	Low	Low	Medium	High

Be Aware: "Framework" Describes Many Things

- Currently in scientific computing, this term 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

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

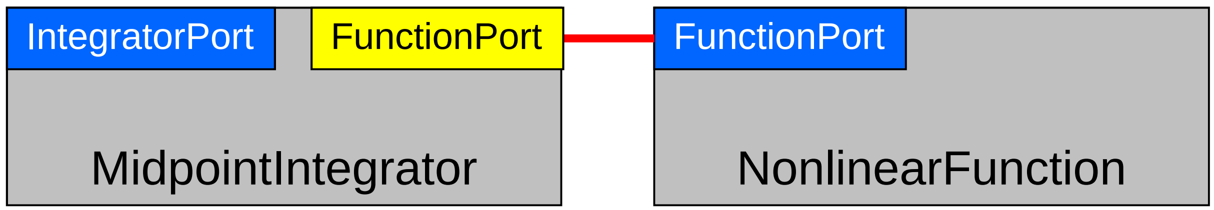
Special Needs of Scientific HPC

- 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: Concept and Practice

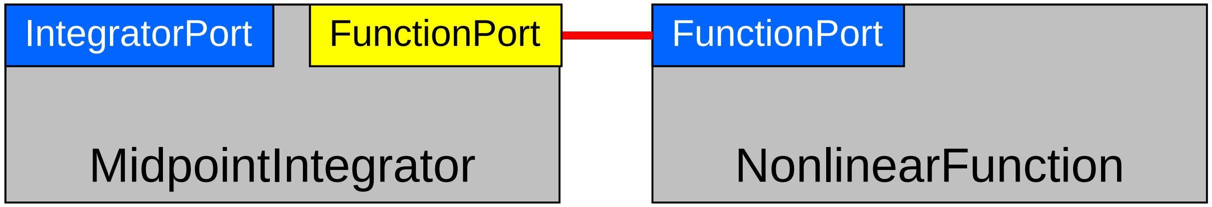
- In the following slides, we explain important concepts of component-based software from the CCA perspective
- We also sketch how these concepts are manifested in code (full details in the Hands-On)
- The CCA Specification is the mapping between concept and code
  - A standard established by the CCA Forum
  - Expressed in the Scientific Interface Definition Language (SIDL) for language neutrality (syntax similar to Java)
  - SIDL can be translated into bindings for specific programming languages using, e.g., the Babel language interoperability tool

CCA Concepts: Components



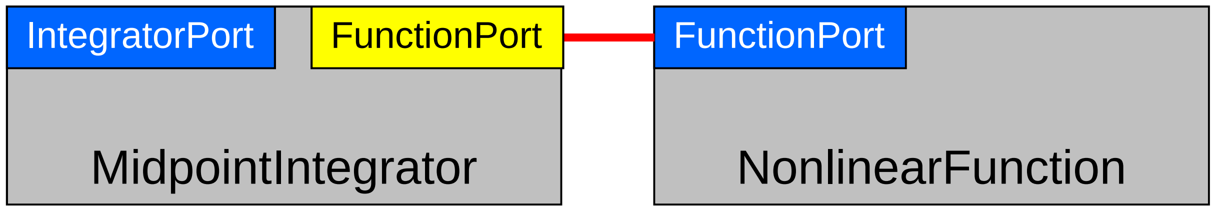
- A component encapsulates some computational functionality
- Components provide/use one or more interfaces
  - A component with no interfaces is formally okay, but isn't very interesting or useful
- In SIDL, a component is a **`class`** that **`implements`** (inherits from) ***`gov.cca.Component`***
  - 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 Concepts: Ports



- Components interact through well-defined interfaces, or *ports*
  - A port expresses some computational functionality
  - In Fortran, a port is a bunch of subroutines or a module
  - In OO languages, a port is an abstract class or interface
- Ports and connections between them are a procedural (caller/callee) relationship, not dataflow!
  - e.g., *`FunctionPort`* could contain a method like `evaluate(in Arg, out Result)` with data flowing both ways

CCA Concepts: *Provides* and *Uses* Ports



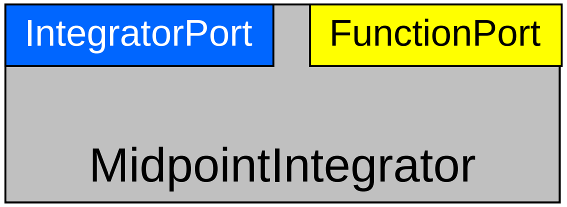
- Components may *provide* ports – implement the class or subroutines of the port ("Provides" Port)
  - *Providing* a port implies certain inheritance relationships between the component and the abstract definition of the interface (more details shortly)
  - A component can *provide* multiple ports
    - Different "views" of the same functionality, or
    - Related pieces of functionality
- Components may *use* ports – call methods or subroutines in the port ("Uses" Port)
  - *Use* of ports is just like calling a method normally except for a little additional work due to the "componentness" (more details shortly)
  - No inheritance relationship implied between caller and callee
  - A component can *use* multiple ports

Components and Ports (in SIDL)

```
package gov.cca {
  interface Component {
    void setServices(…);
} }
```

```
package gov.cca {
  interface Port {
} }
```

```
package integrators {
  interface IntegratorPort
        extends gov.cca.Port
{
    double integrate(…};
} }
```




```
package integrators {
  class Midpoint implements
        gov.cca.Component,
        integrator.IntegratorPort
{
  double integrate(…);
  void setServices(…);
} }
```

Key:

= Inheritance

SIDL inheritance keywords

Components and Ports (in UML)

Note that only the provides ports appear in the component's inheritance hierarchy. Uses ports do not.

A port must extend the CCA spec's port interface

<<interface>>
gov.cca.Component

setServices(services: gov.cca.Services)

<<interface>>
gov.cca.Port

A component must implement the CCA spec's component interface

<<interface>>
integrator.IntegratorPort

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

IntegratorPort FunctionPort

MidpointIntegrator

Midpoint

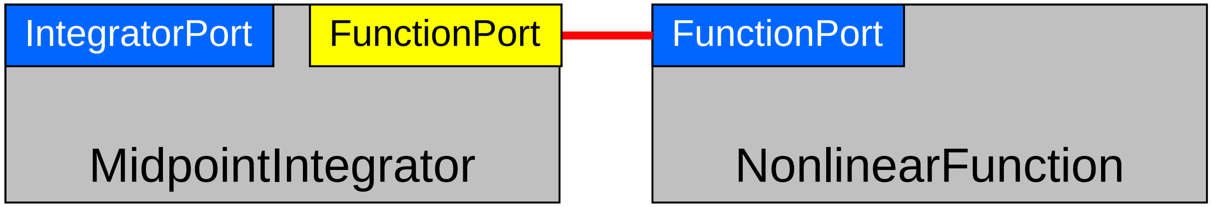
A component must implement the port(s) it provides

Class for Midpoint Integrator component

Key: = Inheritance

SIDL keywords

Using Ports



- 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” a ports that is “in use”

Where Do Ports Come From?

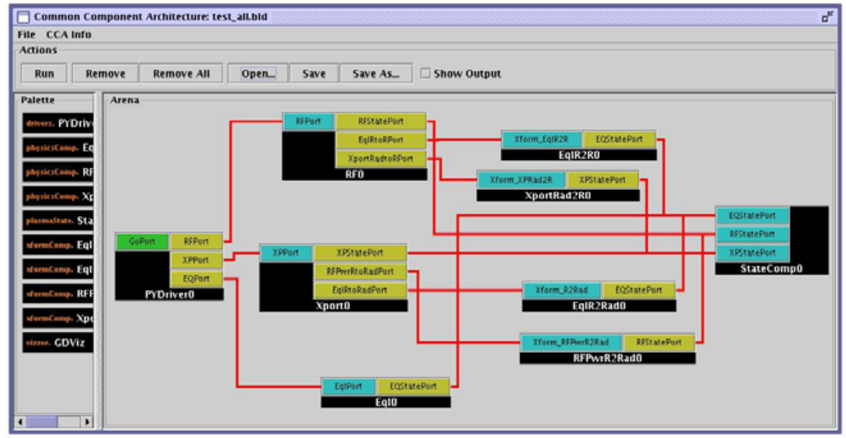
- Most ports are designed and implemented by users of CCA
  - May be specific to an application or used more broadly (i.e. community-wide)
- The CCA specification defines a *small* number of ports
  - Most are services CCA frameworks must provide for use by components
  - Some are intended for users to implement in their components, and have a special meaning recognized by the framework
    - *E.g.* *`gov.cca.ports.GoPort`* provides a very simple protocol to start execution of component-based applications

Interfaces are an Investment

- The larger the community, the greater the time & effort required to obtain agreement
  - Equally true in component and non-component environments
    - MPI 1.0 (well understood at the start) took 8 months, meeting every six weeks
    - MPI 2.0 (not well understood at the start) took 1.5 years, meeting every six weeks
  - Convenient communities are often “project” and “scientific domain”
- Formality of “standards” process varies
- 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

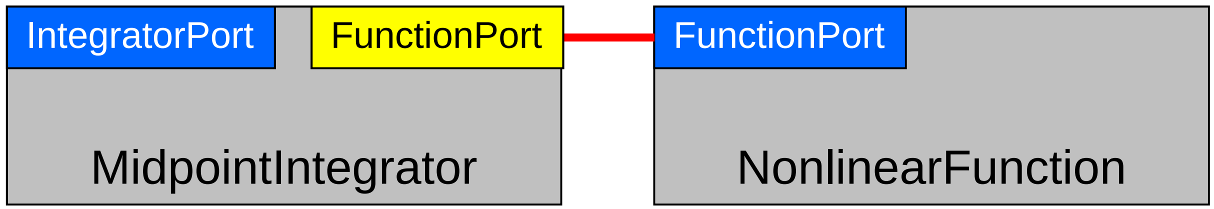
CCA
Common Component Architecture

CCA Concepts: Frameworks



- The framework provides the means to “hold” components and compose them into applications
- Frameworks allow connection of ports without exposing component implementation details
- Frameworks provide a small set of standard services to components
  - Framework services are CCA ports, just like on components
  - Additional (non-standard) services can also be offered
  - Components can register ports as services using the `ServiceProvider` port
- *Currently:* specific frameworks are specialized for specific computing models (parallel, distributed, etc.)
- *Future:* better integration and interoperability of frameworks

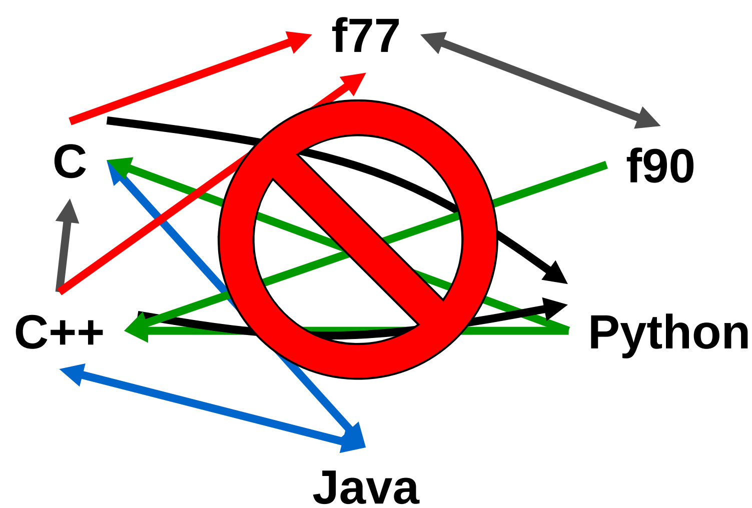
Components Must Keep Frameworks Informed

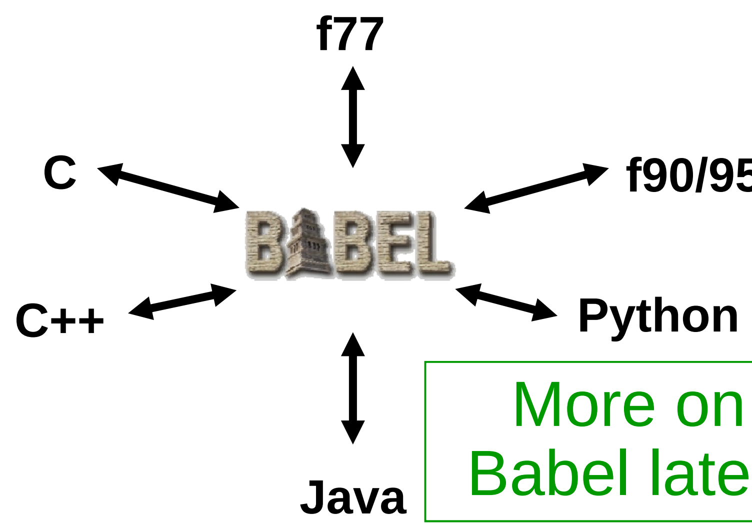


- Components must tell the framework about the ports they are providing and using
  - Framework will not allow connections to ports it isn't aware of
- Register them using methods on the component's **`gov.cca.Services`** object
  - **`addProvidesPort()`** and **`removeProvidesPort()`**
  - **`registerUsesPort()`** 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 Concepts: Language Interoperability

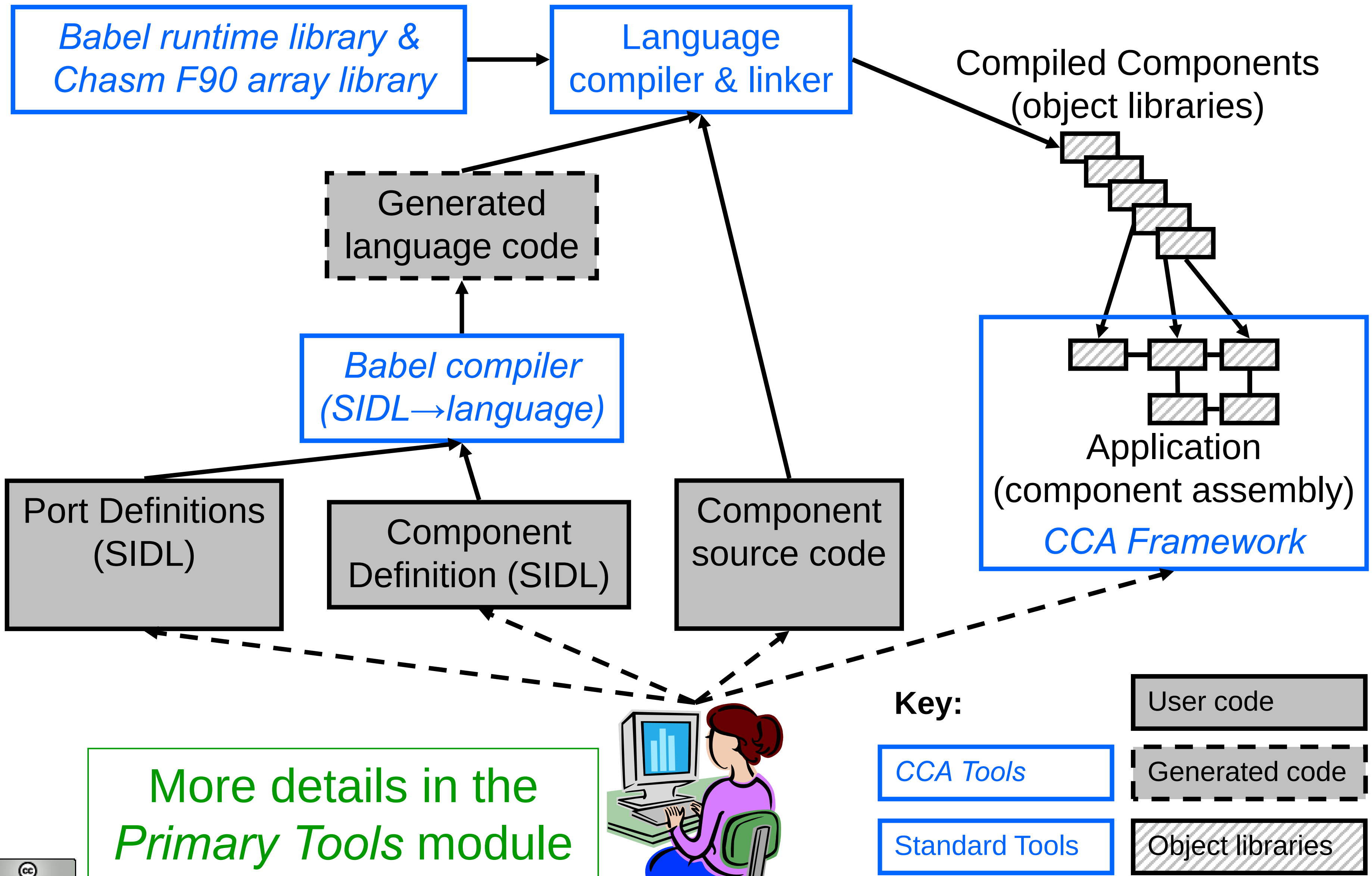
- Scientific software is increasingly diverse in use of programming languages
- In a component environment, users should not care what language a component is implemented in
- “Point-to-point” solutions to language interoperability are not suitable for a component environment
- The Babel language interoperability tool provides a common solution for all supported languages
- Scientific Interface Definition Language provides language-neutral way of expressing interfaces





More on Babel later!

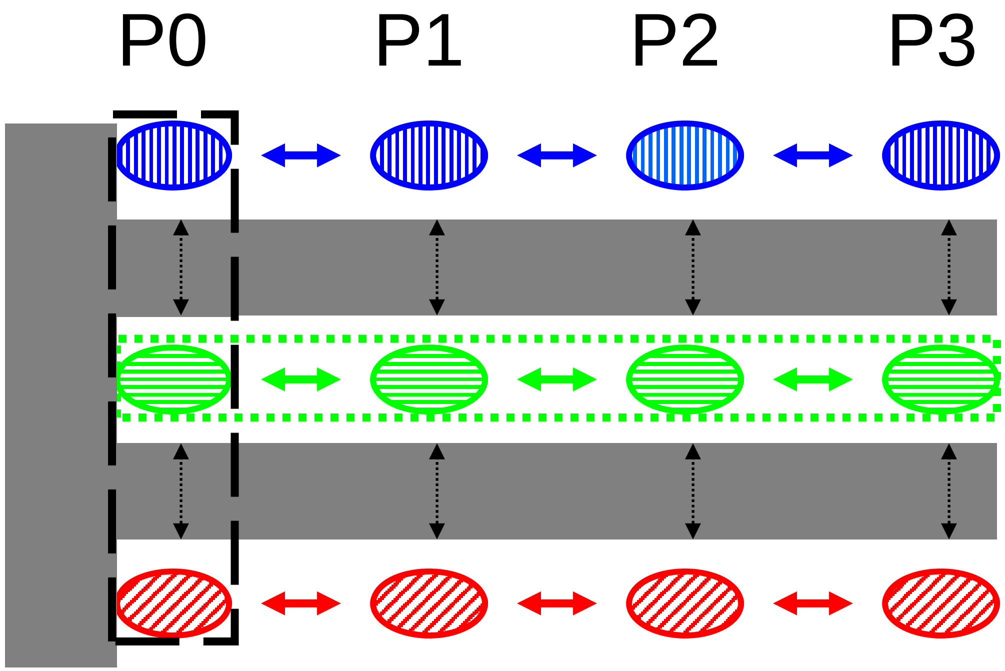
Coding in a CCA Environment



More details in the *Primary Tools* module

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

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



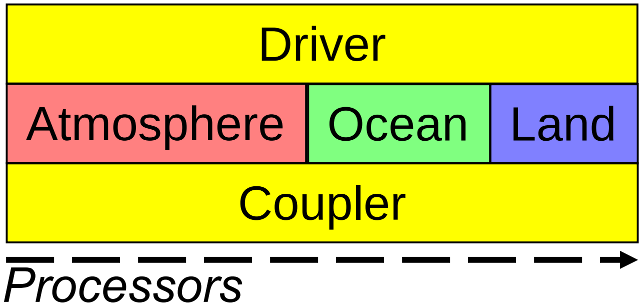
Components: Blue, Green, Red

Framework: Gray

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

"Multiple-Component Multiple-Data" Applications in CCA

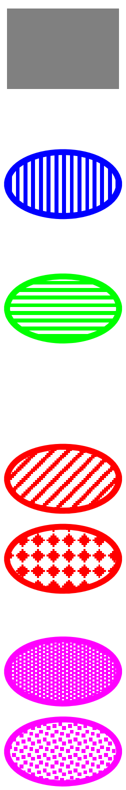
- Simulation composed of multiple SCMD sub-tasks
- Usage Scenarios:
  - Model coupling (e.g. Atmosphere/Ocean)
  - General multi-physics applications
  - Software licensing issues
    - i.e. limited number of instances



- 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

See example in the *Using CCA* module (multilevel parallelism in quantum chemistry)



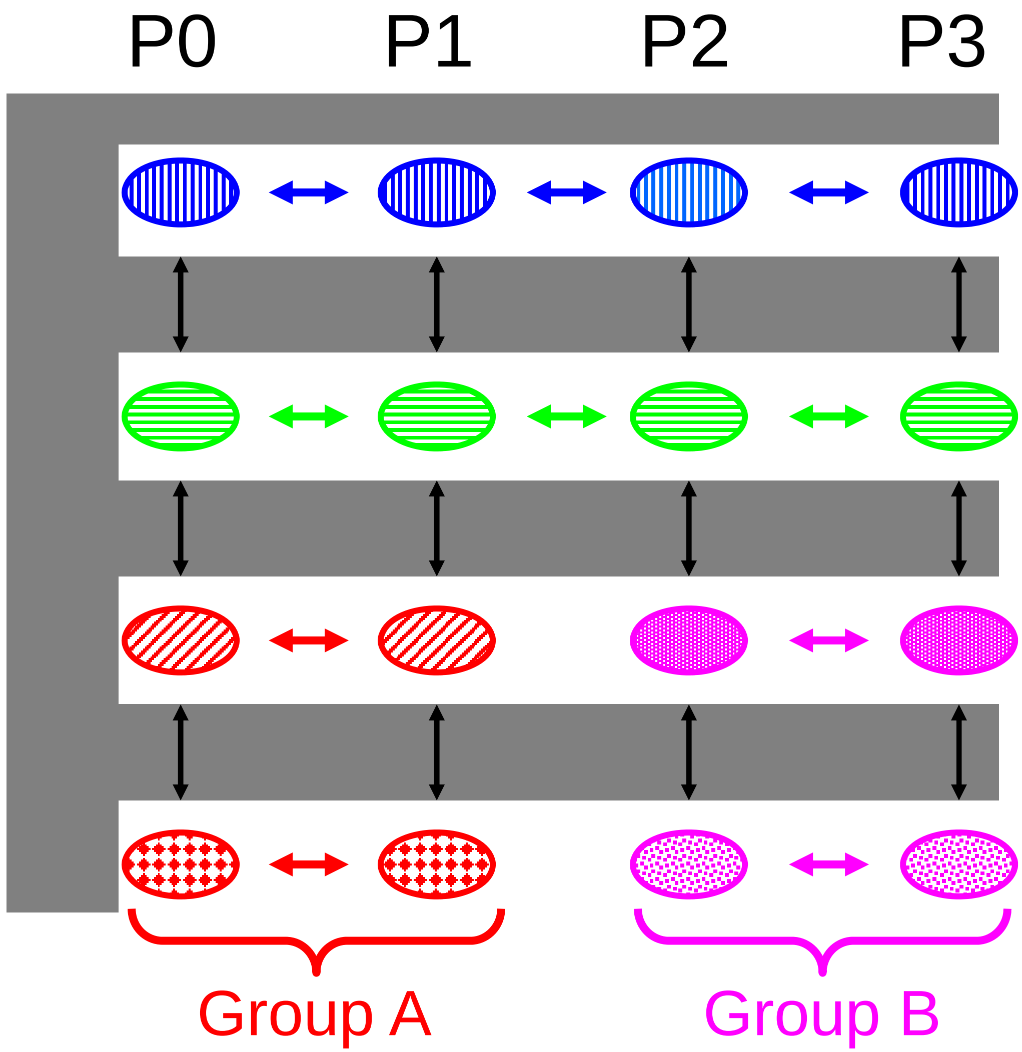
Framework

Application driver & MCMD support component

Components on all processes

Components only on process group A

Components only on process group B



"Direct Connection" Details

- Directly connected components are in the same address space
  - Data can be passed by reference instead of copying
  - Just like "traditional" programs
  - Framework involved in connecting components, but not invocations on ports
- Cost of "CCAness" in a direct connect environment is one level of indirection on calls *between* components
  - Equivalent to a C++ virtual function call: lookup function location, invoke it
  - Overhead is on the invocation only (i.e. latency), 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

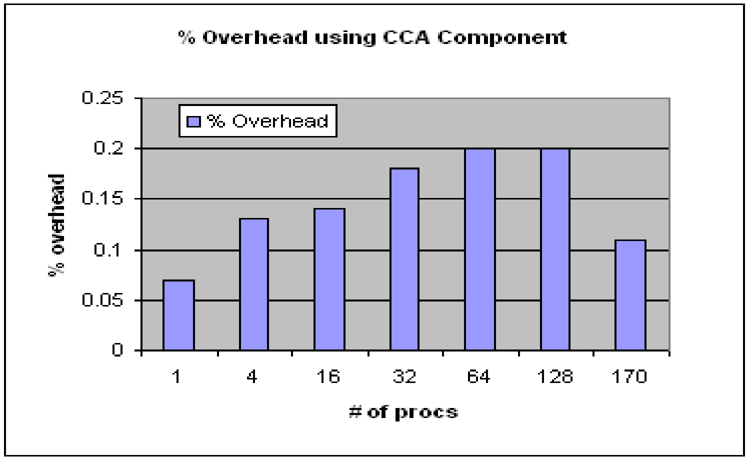
Maintaining HPC Performance

More about performance in notes

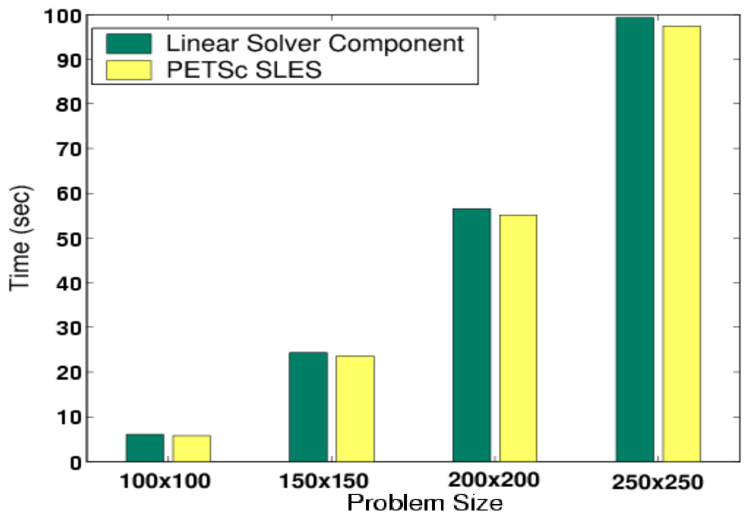
- The performance of your application is as important to us as it is to you
- The CCA is designed to provide maximum performance
  - But the best we can do is to make your code perform no worse, *unless we give easy access to new algorithms*.
- Facts:
  - Measured overheads per function call are low
  - Most overheads easily amortized by doing enough work per call
  - Other changes made during componentization may also have performance impacts
  - *Awareness* of costs of abstraction and language interoperability facilitates design for high performance

Some Performance Results and References

- Lois Curfman McInnes,et al. **Parallel PDE-Based Simulations Using the Common Component Architecture**. In Are Magnus Bruaset, Petter Bjorstad, and Aslak Tveito, editors, *Numerical Solution of PDEs on Parallel Computers*. Springer-Verlag, 2005. Invited chapter, in press.
- S. Benson, et al. **Using the GA and TAO Toolkits for Solving Large-Scale Optimization Problems on Parallel Computers**. Technical report ANL/MCS-P1084-0903, Argonne National Laboratory, September 2003.
- Boyana Norris, et al. **Parallel Components for PDEs and Optimization: Some Issues and Experiences**. *Parallel Computing*, 28:1811--1831, 2002.
- David E. Bernholdt, et al. **A Component Architecture for High-Performance Computing**. In *Proceedings of the Workshop on Performance Optimization via High-Level Languages and Libraries (POHLL-02)*, 2002.



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

Advanced CCA Concepts

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

- Leveraging the component environment to provide additional capabilities to software developers
- The Proxy Component pattern (Hands-On, papers)
- Component lifecycle (tutorial notes, Hands-On)
- Components can be dynamic (papers)
- Improving the quality of component software (papers)
- Support for advanced parallel/high-performance computing (papers)

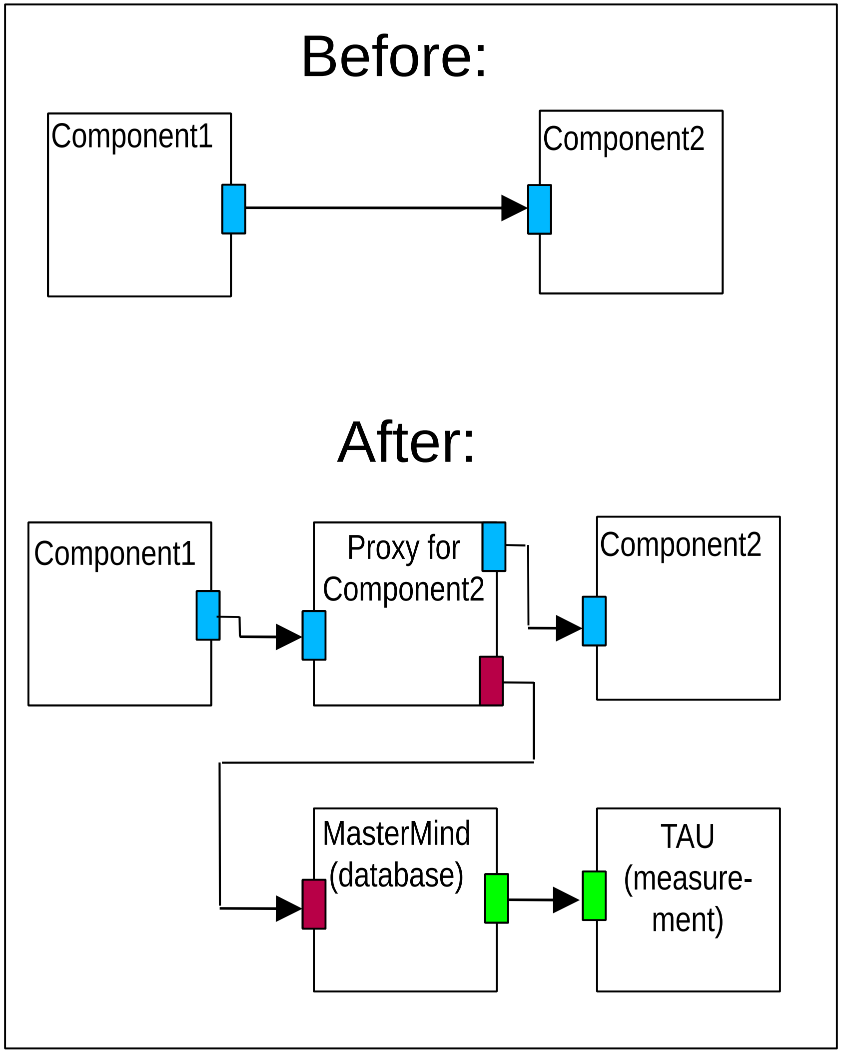
The Proxy Component Pattern

- A "proxy" component can be inserted between the user and provider of a port without either being aware of it (non-invasive)
- Proxy can observe or act on all invocations of the interface
- Similar to aspect-oriented programming
- For many purposes, proxies can 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



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

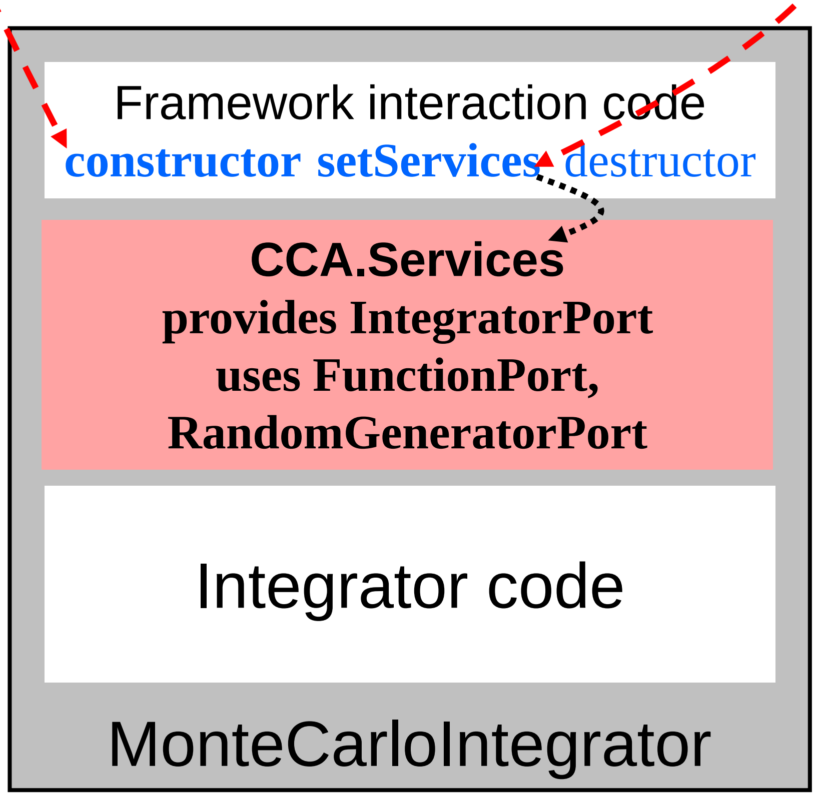
- **Decomposition Phase (termination of application)**
  - Connections between component interfaces may be broken
  - Component may be destroyed

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

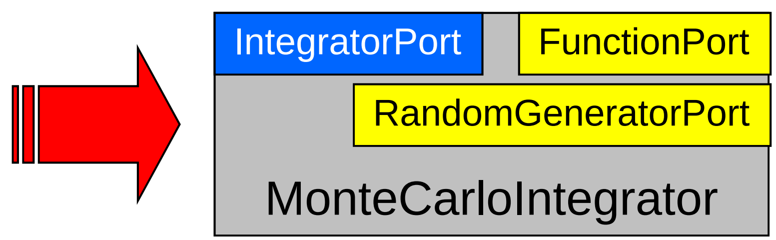
Steps may be under human or software control

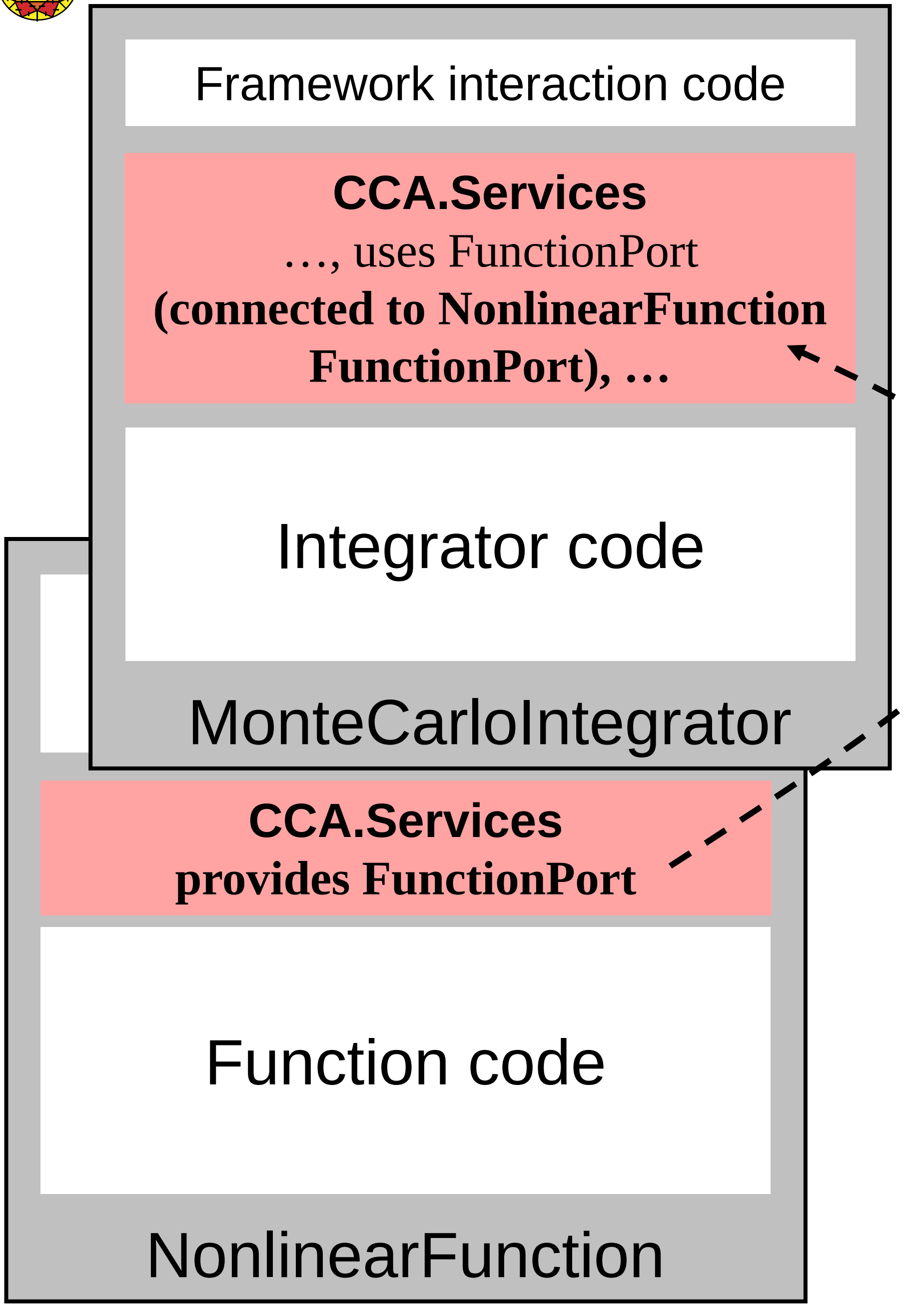
Component's View of Instantiation

- 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





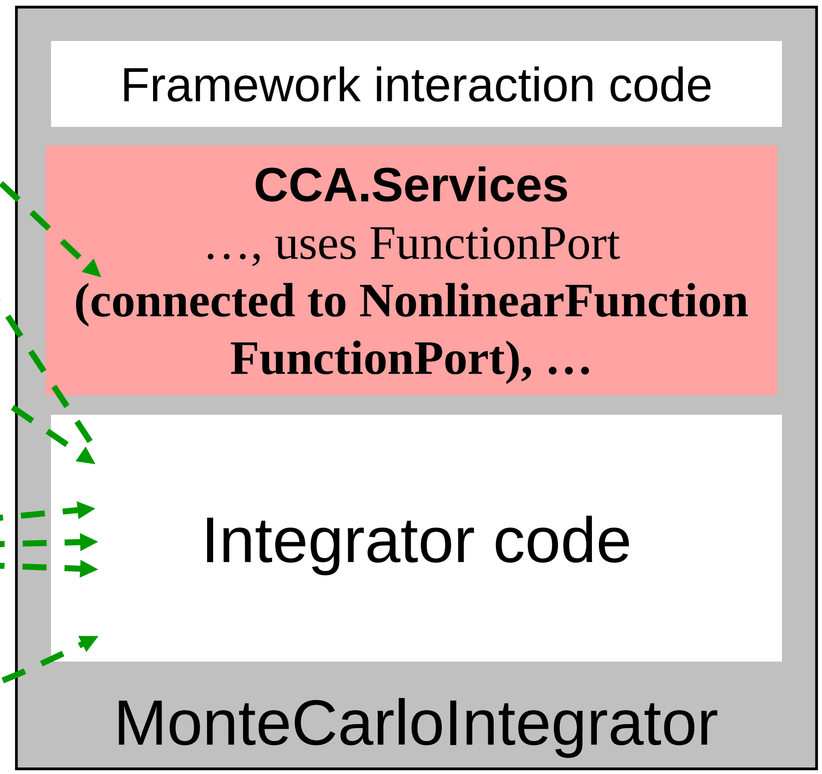
Component's View of Connection

- 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

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



Dynamic Component Assemblies

- ***`gov.cca.BuilderService`*** allows programmatic composition of components
  - Components can be instantiated/destroyed, and connected/disconnected under program control

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.
  - TASCS project “Computational Quality of Service” activity

Enhancing Software Quality

- Current component architectures define syntax of interfaces
- Extend interface to include semantics (behavior) for more complete definition
  - "Design by contract"
  - Help ensure component performs correctly
  - Help ensure component is used correctly
- Selective enforcement to control impact
- TASCS project "Software Quality and Verification" activity

```
package vector version 1.0 {
  class Utils { …
    static double norm(in array<double> u,
                       in double tol,
                       in int badLevel)
      require    /* Preconditions */
        not_null : u != null;
        u_is_1d : dimen(u) == 1;
        non_negative_tolerance : tol >= 0.0;
      ensure    /* Postconditions */
        no_side_effects : is pure;
        non_negative_result : result >= 0.0;
        nearEqual(result, 0.0, tol)
          iff isZero(u, tol);
 … }
}
```

Supporting Emerging HPC Hardware Environments

- CCA does not dictate a specific approach to parallelism
- Different approaches and tools can be provided via components and custom frameworks

Examples…

- Uintah Computational Framework (Utah) provides a multi-threaded parallel execution environment based on task graphs
  - Specialized to certain structured adaptive mesh refinement problems
- TASCS developing services to manage groups of parallel components/tasks (MCMD)

Also…

- TASCS developing support for heterogeneous processor environments
  - FPGAs, GP-GPUs, accelerators, and other co-processors
  - Accelerator code encapsulated as components, interacting w/ components on primary processors
- Integration of fault tolerance capabilities with CCA under development (CIFTS-TASCS collaboration)

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
- “SIDLize” interfaces incrementally
  - Start with essential interfaces
  - Remember, only externally exposed interfaces need to be Babelized
- Componentize at successively finer granularities
  - Start with whole application as one component
    - Basic feel for components without “ripping apart” your app.
  - Subdivide into finer-grain components as appropriate
    - Code reuse opportunities
    - Plans for code evolution

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

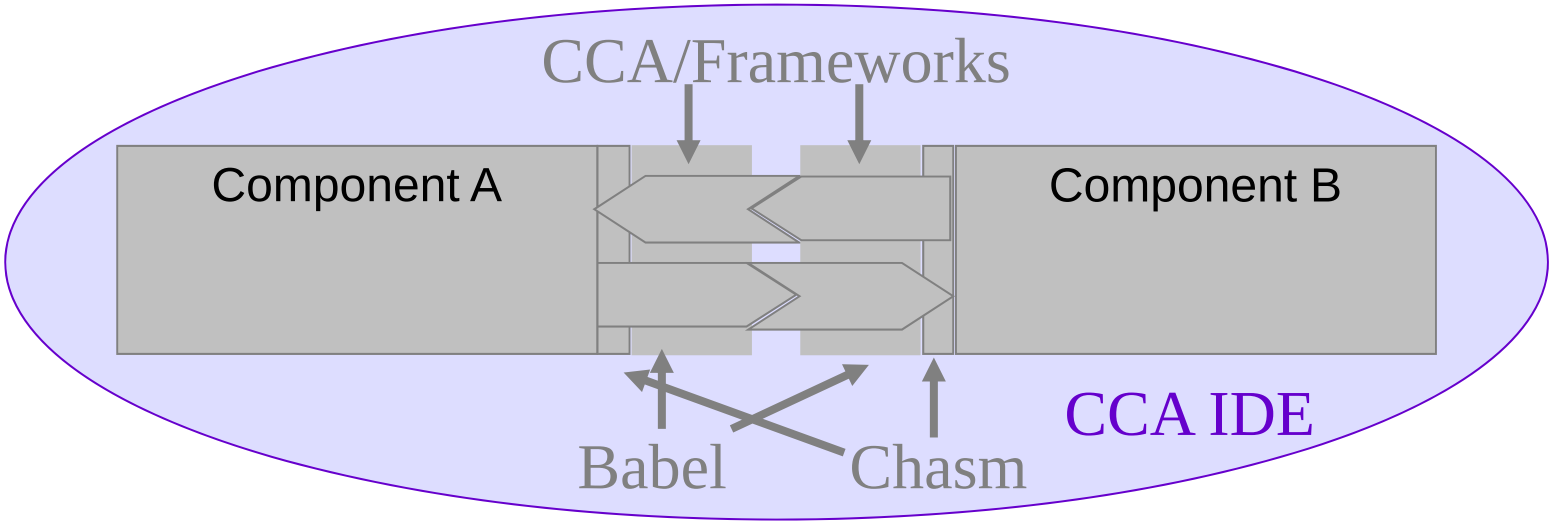
The Primary Tools

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

The tools

- Bocca – project environment
- Ccaffeine – framework
- SIDL – interoperability language
- Babel – HPC language binding generator
- CCA – specification for components, frameworks

Tools Module Overview



- CCA Development Environment
- Frameworks
- Language interoperability tools

Bocca Development Environment

- Provides a text-based, portable environment
  - Create or import SIDL and CCA based codes.
  - Automatic build system maintenance.
  - Easy to adopt or abandon while preserving code, build.
- No GUI required.
- Basis for common CCA toolkit installation.
- Manages components in all Babel-supported languages (C, C++, Fortran, Java, Python).

Bocca Creates Skeletons for CCA

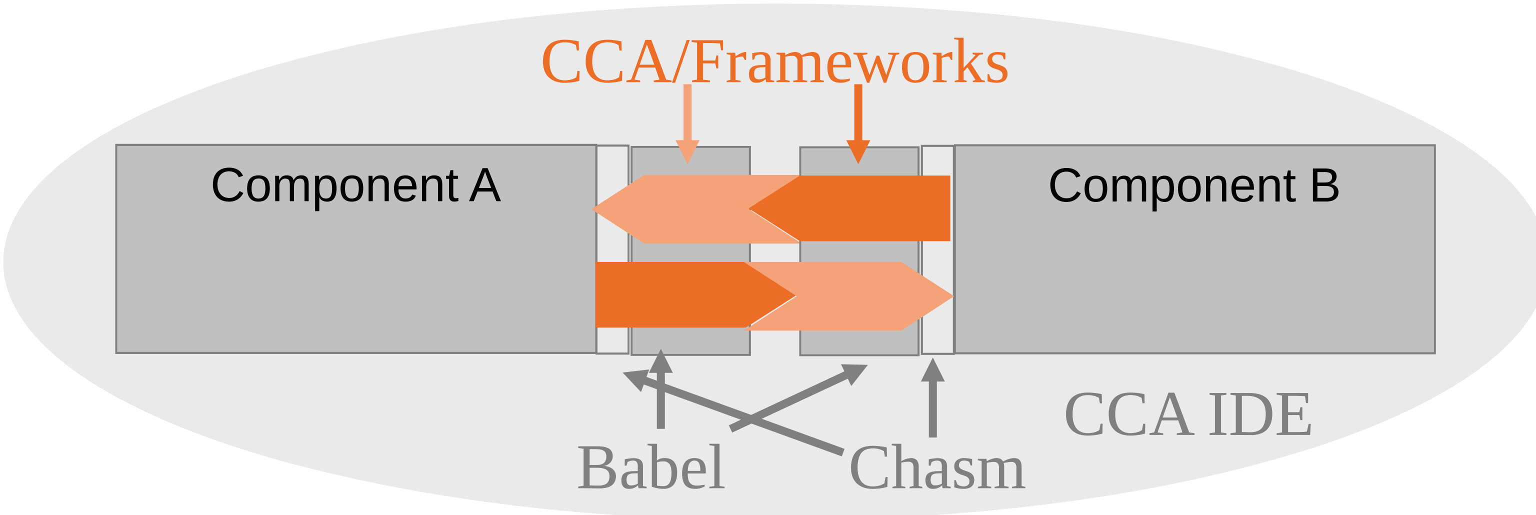
- Including ports and interfaces
  - Give the SIDL name and an empty port or interface is created.
- Including components and classes
  - Give the name and an empty component or class is created.
  - Some extra options: the component uses/provides ports, implemented interfaces or extended classes
- Including build system
  - For all ports/components in the project
  - Implemented in any CCA supported language
- Create applications with Ccaffeine GUI (today)
- Including application composition (coming soon)

CCA
Common Component Architecture

Bocca Example

```
# create an empty but buildable CCA skeleton
bocca create project myproj
cd myproj
./configure

bocca create port myJob
bocca create component myWorker --provides=myJob:job1

# fill in public functionality
bocca edit port myJob

# fill in implementation
bocca edit component –i myWorker

# compile application
make
```

Tools Module Overview



- CCA Interactive Development Environment
- Frameworks
- Language interoperability tools

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

- Supports SIDL/Babel components
  - Conforms to latest CCA specification (0.8)
  - 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

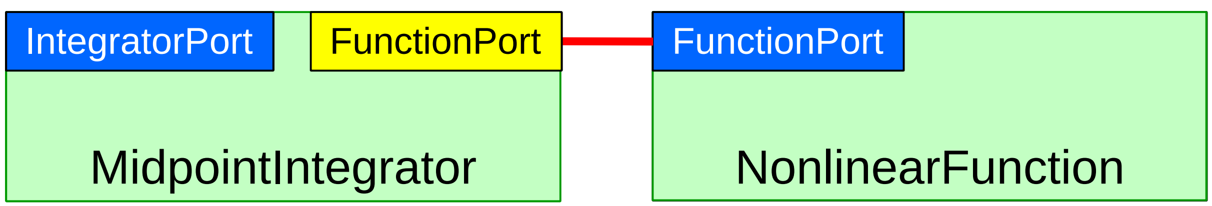
Supported on Linux, AIX, OSX and is portable to modern UNIXes.

Optional

Ccaffeine GUI

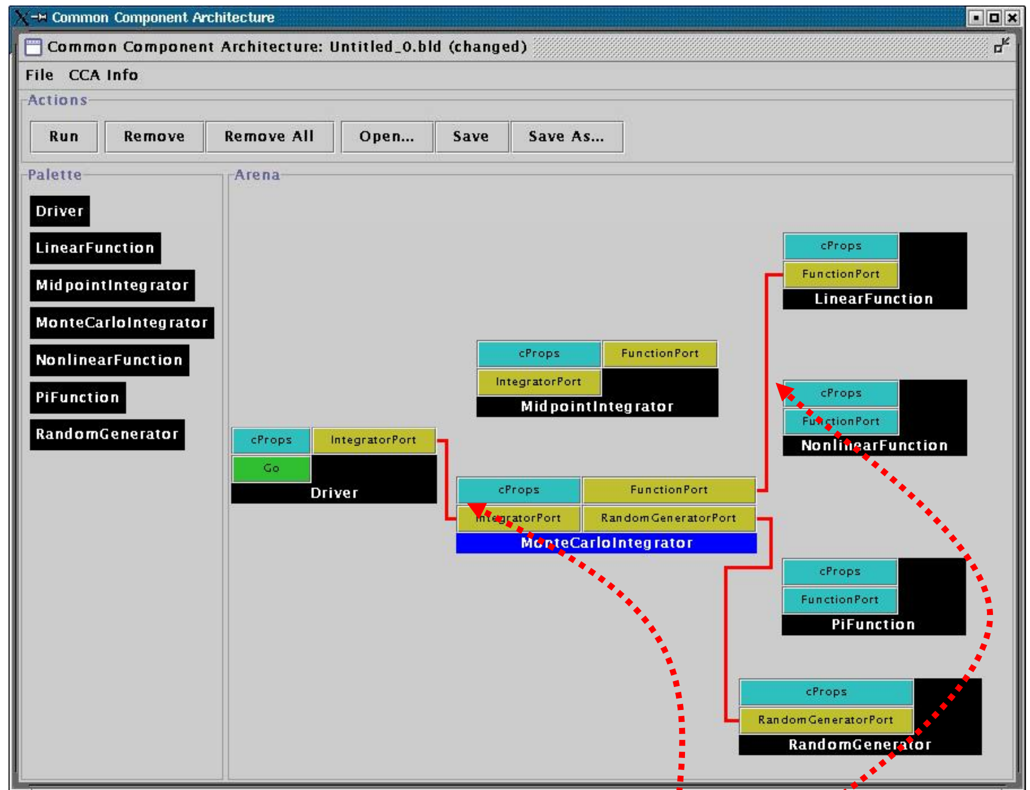
- Process
  - User input is broadcast SPMD-wise from Java.
  - Changes occur in GUI *after* the C++ framework replies.
  - If your components are computing, GUI changes are blocked.

- 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



User Connects Ports

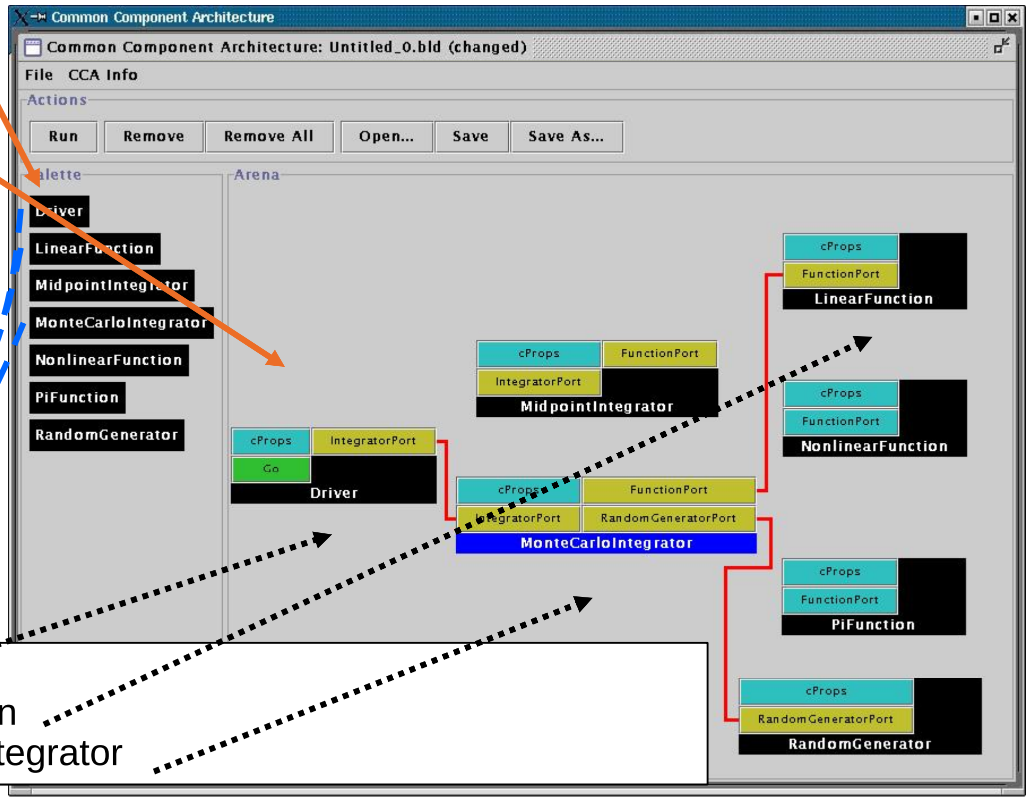
- Can only connect *uses* & *provides*
  - *Not* uses/uses
  - *Not* provides/provides
- Ports connected by type not name
  - Port names must be unique within a component
  - Types must match across components
- Framework puts info about *provider* of port into *using component's* Services object



connect	Driver	IntegratorPort	MonteCarloIntegrator	IntegratorPort
connect	MonteCarloIntegrator	FunctionPort	LinearFunction	FunctionPort
…				

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



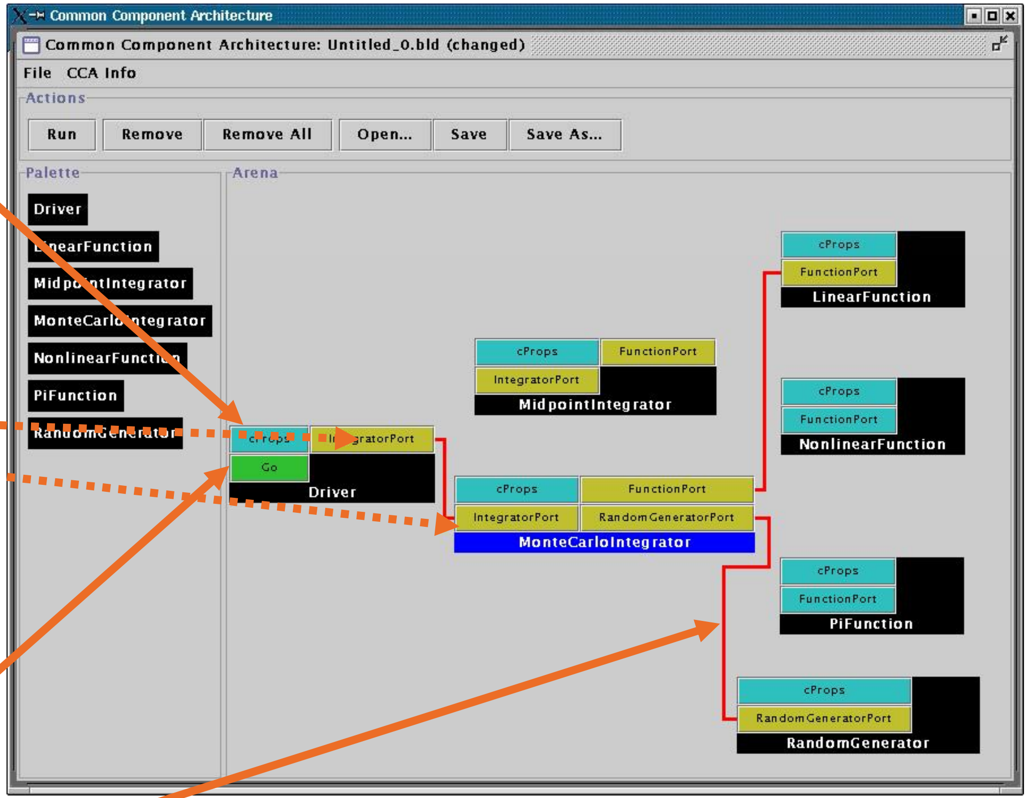
create	Driver	Driver
create	LinearFunction	LinearFunction
create	MonteCarloIntegrator	MonteCarloIntegrator

Building an Application (2 of 2)

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

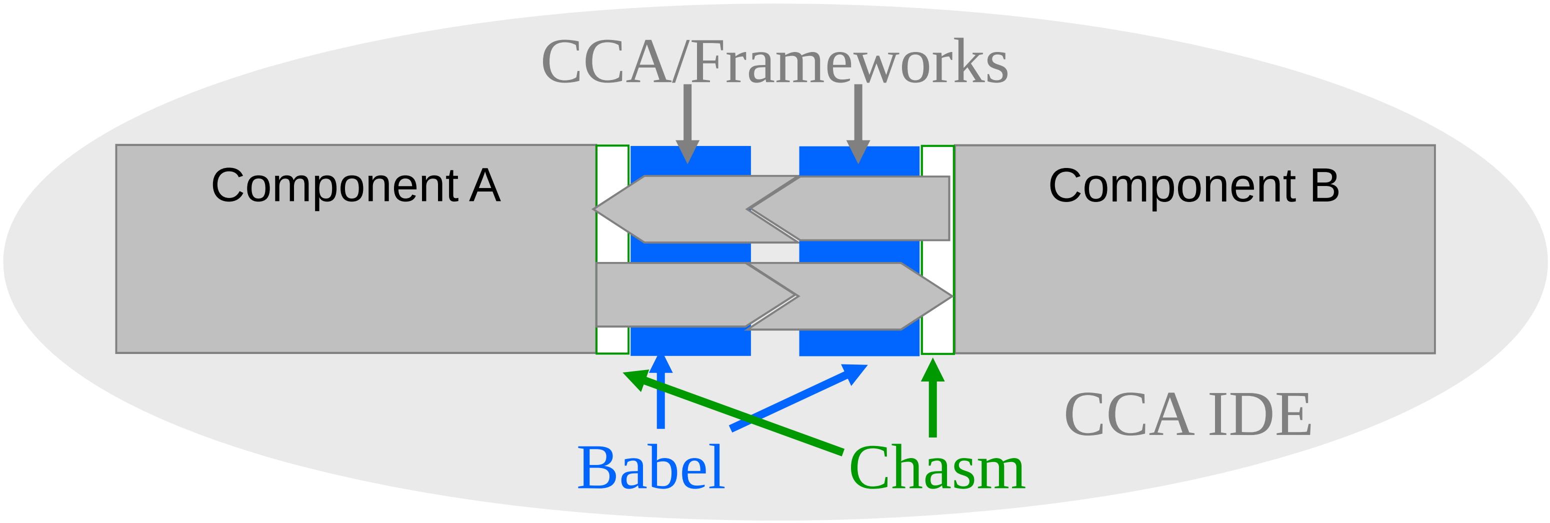
2. *For each connection*: click a *uses* port then click a *provides* port to establish a connection.

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



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

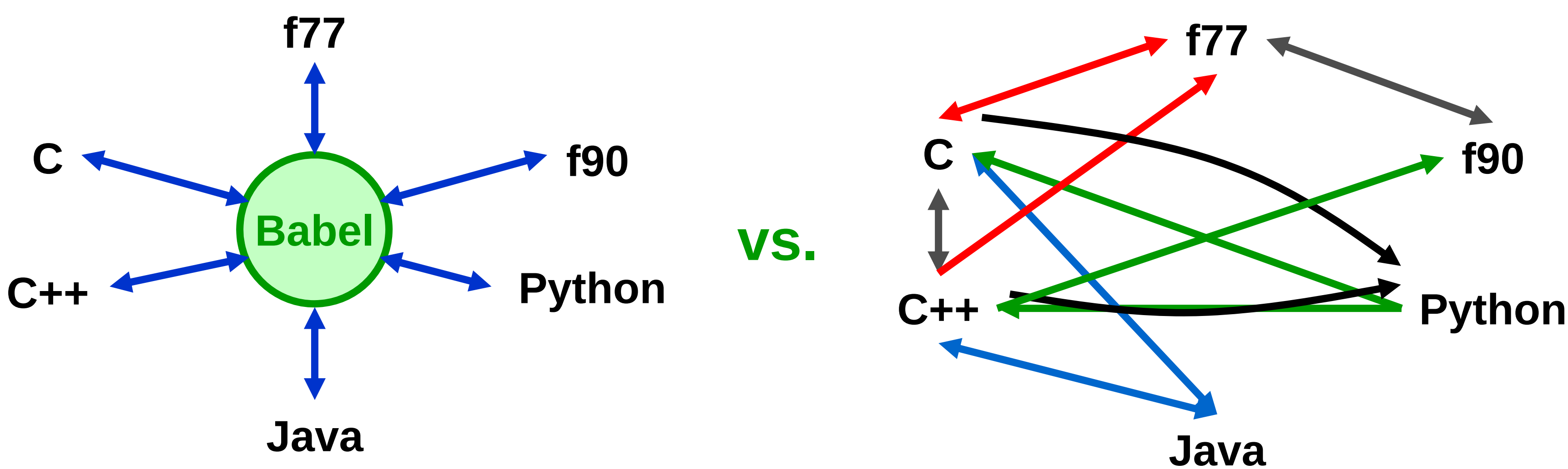
Tools Module Overview



- CCA Interactive Development Environment
- Frameworks
- Language interoperability tools

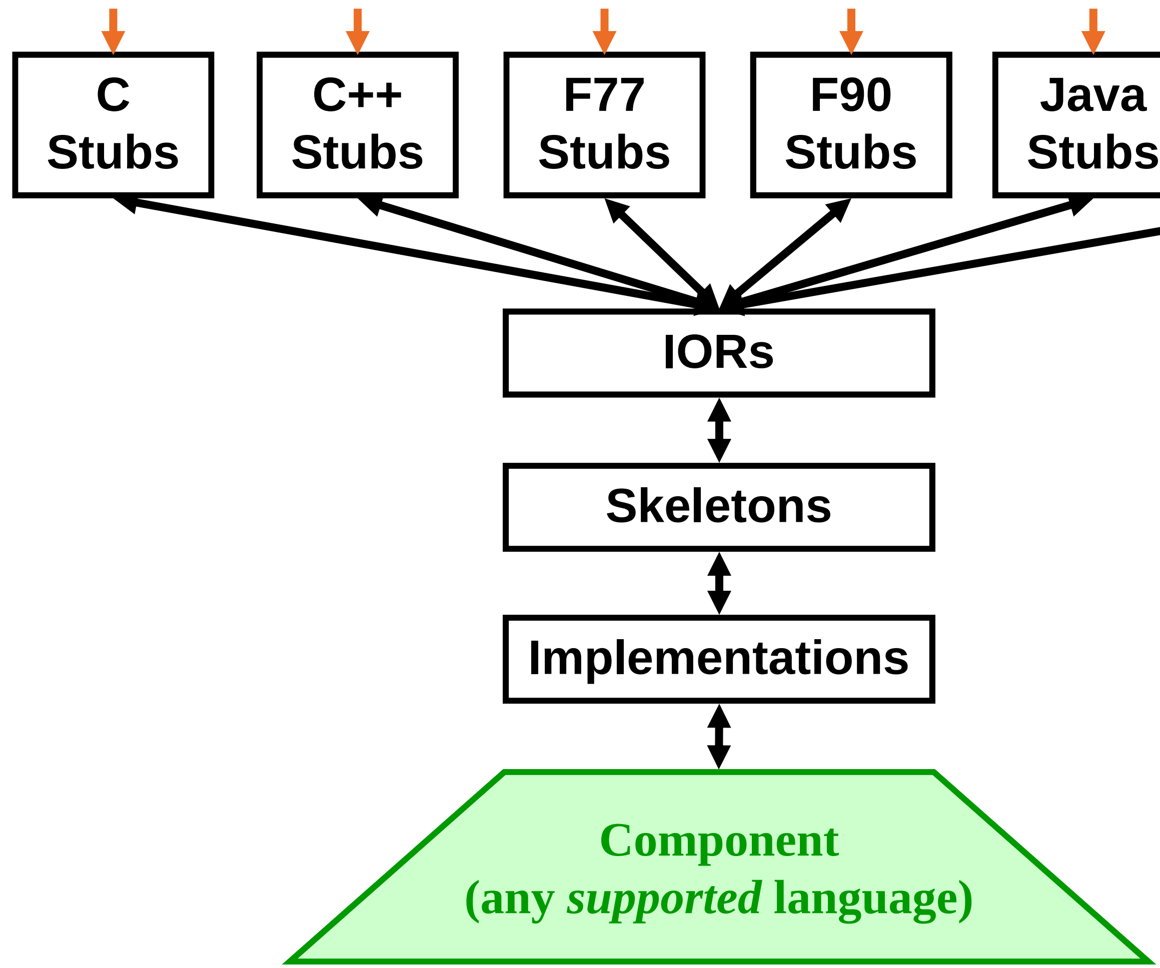
SIDL Facilitates *Scientific* Programming Language Interoperability

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



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

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



IOR = Intermediate Object Representation

The SIDL File that defines the "greetings.English" type

```
①② package greetings version 1.0 {
③④     interface Hello {
⑤          void setName( in string name );
           string sayIt ( );
        }
⑥       class English implements-all Hello {  }
    }
```

Handout Material: Code Notes

① Packages contain user-defined types and are used to reduce naming collisions. Packages can be nested.

② Packages can be versioned. User defined types must be nested inside a versioned package and gain the same version number as the innermost versioned package

③ SIDL has a inheritance model similar to Java and Objective C. Classes can inherit multiple interfaces, but at most one implementation (other class).

④ An interface describes an API, but doesn’t name the implementation.

⑤ Note that arguments have mode, type, and name. Mode can be one of “in”, “out”, and “inout”. These SIDL modes have slightly different semantics than Fortran90 “intents”.

⑥ This class generates English greetings. One could imagine a strategy for internationalization that uses the Hello interface everywhere, but loads in English, French, or whatever classes based on user’s preference.

Working Code: "Hello World" in F90 Using a Babel Type

```
program helloclient
① use greetings_English
② use sidl_BaseInterface
  implicit none
③ type(greetings_English_t) :: obj
④ type(sidl_BaseInterface_t):: exc
  character (len=80)        :: msg
  character (len=20)        :: name
⑤ name='World'
⑥ call new( obj, exc )
⑦ call setName( obj, name, exc )
⑧ call sayIt( obj, msg, exc )
⑨ call deleteRef( obj, exc )
  print *, msg
end program helloclient
```

Handout Material: Code Notes

① Use statement for the greetings.English type

② Use statement for the sidl.BaseInterface type

③ Obj is a F90 derived type we get from the using statement, note the "_t" extension that prevents it from colliding with the using statement.

④ Exc is used to hold exceptions thrown by methods

⑤ In C/C++ examples, this variable would be initialized by a the command-line variable "argv[1]", but its trickier to do portably in F90 and too long, so I just initialize the name to "World".

⑥ Obj is not yet initialized. The Babel idiom in F90 is to call new() to initialize the Babel type. In other languages its _create(). NOTE: good code would add error checking.

⑦ setName() puts data into the obj. It sets its state.

⑧ sayIt() returns the entire greeting including the aforementioned name.

⑨ deleteRef() is a subroutine that all Babel types inherit from a parent class. All Babel objects are reference counted. When there are no more outstanding references, the object is told to clean up after itself.

Working Code: “Hello World” in F90 Using a Babel Type

```
program helloclient
  use greetings_English
  use sidl_BaseInterface
  implicit none
  type(greetings_English_t) :: obj
  type(sidl_BaseInterface_t):: exc
  character (len=80)         :: msg
  character (len=20)         :: name
  name='World'
  call new( obj, exc )
  call setName( obj, name, exc )
  call sayIt( obj, msg, exc )
  call deleteRef( obj, exc )
  print *, msg
end program helloclient
```

Looks like a native F90 derived type

These subroutines were specified in the SIDL.

Other basic subroutines are “built in” to all Babel types.

Question: What language is “obj” really implemented in?

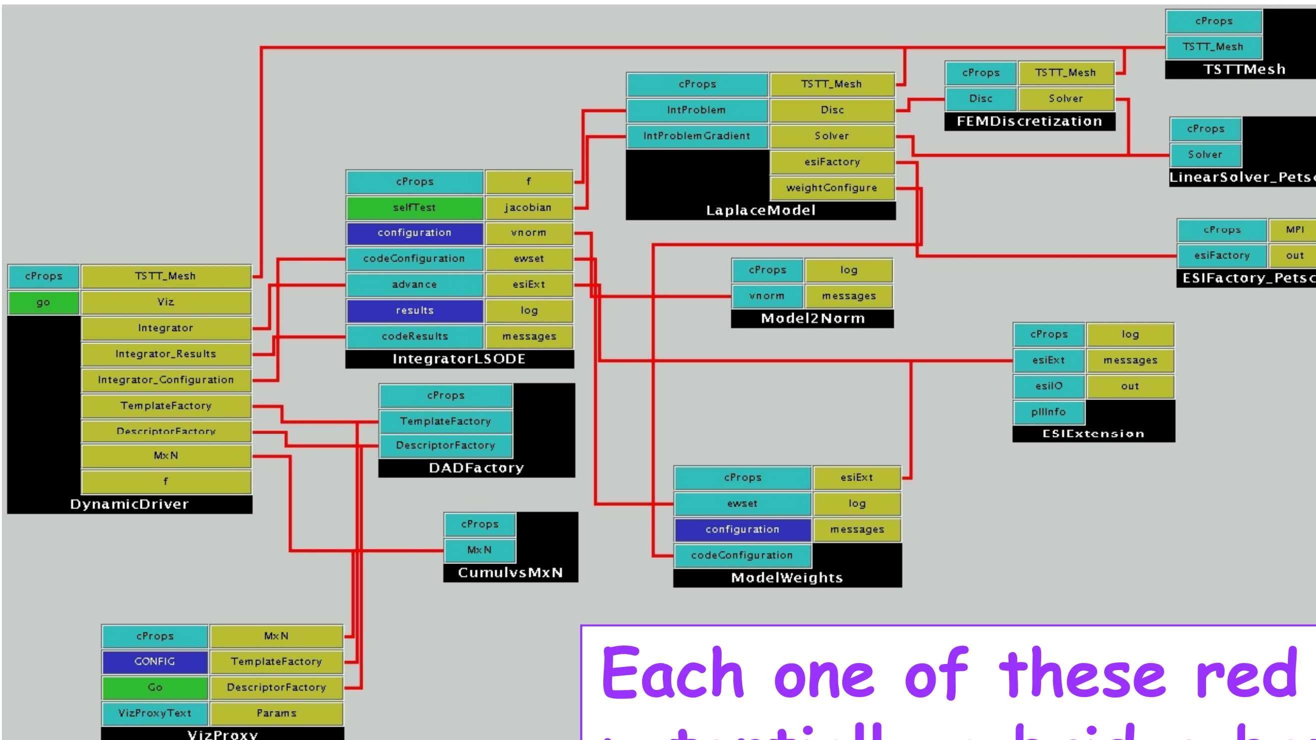
```
program helloclient
  use greetings_English
  use sidl_BaseInterface
  implicit none
  type(greetings_English_t) :: obj
  type(sidl_BaseInterface_t):: exc
  character (len=80)         :: msg
  character (len=20)         :
  name='World'
  call new( obj, exc )
  call setName( obj, name, ex
  call sayIt( obj, msg, exc )
  call deleteRef( obj, exc )
  print *, msg
end program helloclient
```

Answer: Can't Know!

With Babel, it could be C, C++, Python, Java, Fortran77, or Fortran90/95

In fact, it could change on different runs without recompiling this code!

CCA uses Babel for high-performance n-way language interoperabilty



Each one of these red lines, is potentially a bridge between two languages. No telling which language your component will be connected to when you write it.

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 uses SIDL to specify APIs and Type Hierarchy for Frameworks, Services, Components, & Ports

- **A CCA framework must**
  - **implement `gov.cca.AbstractFramework`,**
  - **provide a `gov.cca.ports.BuilderService`,**
  - **etc.**
- **A CCA port must**
  - **be a SIDL interface extending `gov.cca.Port`**
- **A CCA component must**
  - **be a SIDL class implementing `gov.cca.Component`**

The CCA Specification is a SIDL file.

How to write a Babelized CCA Component (1/2)

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 );
    }
}
```

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

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 { }
```

Using the CCA: Approaches & Experience

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

Components in the Small: Impacts within a Project

Benefits include:

- Rapid testing, debugging, and benchmarking
- Wrapped legacy portions need not be reimplemented or reverified
- Interface change effects across components are clear and usually automatically found by compilers if overlooked
- Object-orientation made simpler for C and Fortran
- Support for implementation-hiding discipline
- Coordination of independent workers while separating concerns (scientific specialty, development style, etc.)
- Work of transient contributors remains as well-defined, lasting components

Components in the Large: Connecting Multiple Projects

Benefits include:

- SIDL can be used to facilitate the interface consensus processes
- Different sub-projects do not have to agree on one implementation language
- Developers who never meet in person have an excellent chance of code integration working on the first try

Costs include:

- Consensus can be expensive to obtain
- Writing code for others to use is more difficult than writing it just for yourself

CCA
Common Component Architecture

Application Decomposition Strategies

- Conceptually decompose the application into
  - cutting-edge areas (less stable) *and*
  - areas that can employ existing component-based libraries (more stable)
- Decompose each area into components for
  - physics
  - mathematics
  - data management

  as dictated by the application; sketch a typical component layout
- Many components will encapsulate algorithmic logic only, with little or no private data
- Most HPC applications will have a central data abstraction that provides data memory management and parallel communication
- In a multilanguage application, all I/O *may* need to be isolated into components written in a single common language (file based I/O should not be affected)
- Component boundaries (and port interfaces) may be set to isolate proprietary code or difficult contributors

Interface Design: Technical Factors

- Do we make a single large port look like the underlying library or divide functions into groups on separate ports?
- Should a function with many optional arguments be split into several alternative functions with simpler usage?
- Do we make the ports more general than the existing code?
- Do we require the ports to work across languages? Across networks?
  - If not, gains in efficiency or coding ease might be had
  - If so, memory management and I/O challenges may arise

Interface Design: Social Factors
(Defining Ports to Wrap Existing Code)

- Will the port hide just one implementation, or will there need to be plug compatibility with other implementations? From other teams?
- Who defines the interface and maintains it?
  1. Project dictator? (*Fast…*)
  2. The owner of the legacy functionality? (*Slow, if not you…*)
  3. A standards committee? (*Really slow…*)
- How many iterations of redefining the ports will the customers tolerate?

Implementation Issues in Wrapping

- Do we split large libraries into several components?
  - Splitting is difficult to do if global variables or common blocks are widely used.
- Do we expect more than one implementation instance of a port in a single run-time?
  - If not, interface contracts may include global side effects
- Do we integrate the wrapper code in the existing code's development and build processes?
  - If not, how do we ensure build consistency and on-going wrapper support?
- Code bases with large interfaces need automated wrapping tools

Benefits of Wrapping Code Using CCA

- Setting a language-neutral interface definition (SIDL) can greatly clarify design discussions
- Provides a chance to reorganize the interface and hide globals
- Allows testing of alternate versions if doing performance studies
- Allows easy “experimentation” with new algorithms
- Software discipline is enforced, not optional
- Implementation decisions (to split libraries, etc) can be easily revised over time if interfaces remain constant (possibly with the addition of new interfaces)

Interface Design for *New* Code

- Write SIDL for each connection (port) in the sketched component layout
- If two ports must always be used together, consider merging them
- Review SIDL drafts for near-duplication of ports
- Avoid creating interface contracts that require using hidden global data
- Consider exporting tuning and/or configuration parameter inputs as a port
- All the design issues from wrapping existing code apply, also
- ***Interfaces will change.***

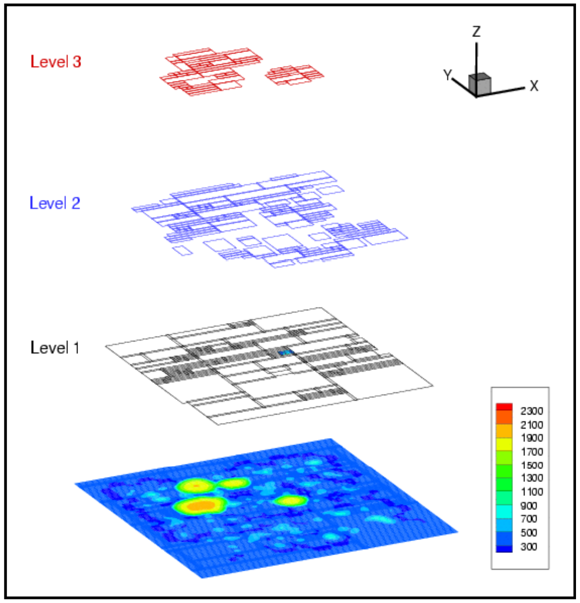
Recommended Implementation Patterns

- Expect to decompose initial components further as work progresses and requirements expand
- Build systems (i.e. `make`) should be kept as simple as possible
  - Keep a subdirectory for port definitions and any implementation-independent glue code derived from the ports
  - Keep each component (and any wrapped code) in its own subdirectory
  - Keep application-wide flags in a configure script or an include file common to all components and ports
  - Consistency is key. Extract build flags from `cca-spec-babel-config` and if possible compile & link with `babel-libtool`

The bocca tool generates a build system consistent with these recommendations

Case Study: Combustion Modeling

- Computational Facility for Reacting Flow Science (CFRFS)
  - http://cfrfs.ca.sandia.gov
  - Funded via SciDAC initiative (PI: H. Najm)
- Focus: A toolkit to perform simulations of lab-sized unsteady flames
  - Solve the Navier-Stokes w/detailed chemistry
  - Various mechanisms up to ~50 species, 300 reactions



- Consequently:
  - Disparity of length scales :
    - use structured adaptively refined meshes
  - Disparity of time scales (transport versus chemistry) :
    - use an operator-split construction and solve chemistry implicitly
    - adaptive chemistry: use computational singular perturbation to find and follow low dimensional chemical manifolds

J. Ray, S. Lefantzi, J. Lee, C. Kennedy, W. Ashurst, K. Smith, M. Liu, N. Trebon, J. Ortega, C. Safta, S. Chandra, H. Johansson

Why Use CCA in the CFRFS Toolkit?

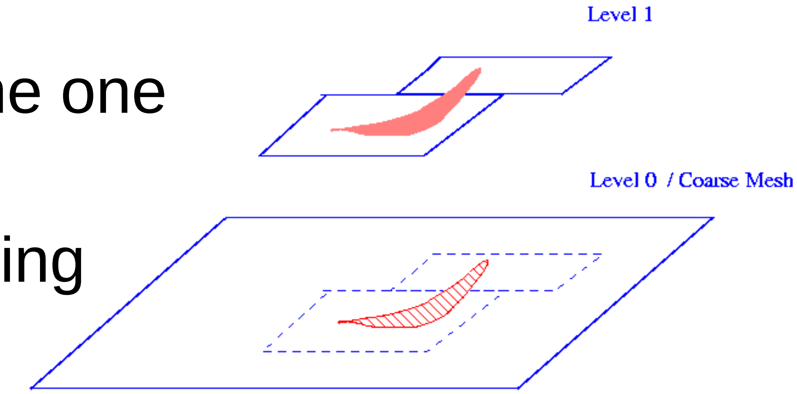
- Separate clearly the physics models, numerical algorithms, and the “CS” parts of the toolkit
  - Strictly functional!
- Realize the separation in software
- Tame *software* complexity
- Separate contributions by transient contributors
  - Form the bulk of the developers
- Create “chunks” of well-defined functionality that can be developed by experts (usually numerical analysts and combustion researchers)

Design Principles of the Toolkit / 1

- **Principal Aim: Reduce software complexity**
  - We can deal with the rest
- Functional decomposition into components
  - “Data Object” and Mesh components
  - (Large) set of numerical algorithmic components (integrators, linear/nonlinear solvers, etc.)
  - (Large) set of physical models components (gas-phase combustion chemistry, thermodynamics, fluid dynamic quantities, e.g. viscous stress tensor)
  - Handful of adaptors

Design Principles of the Toolkit / 2

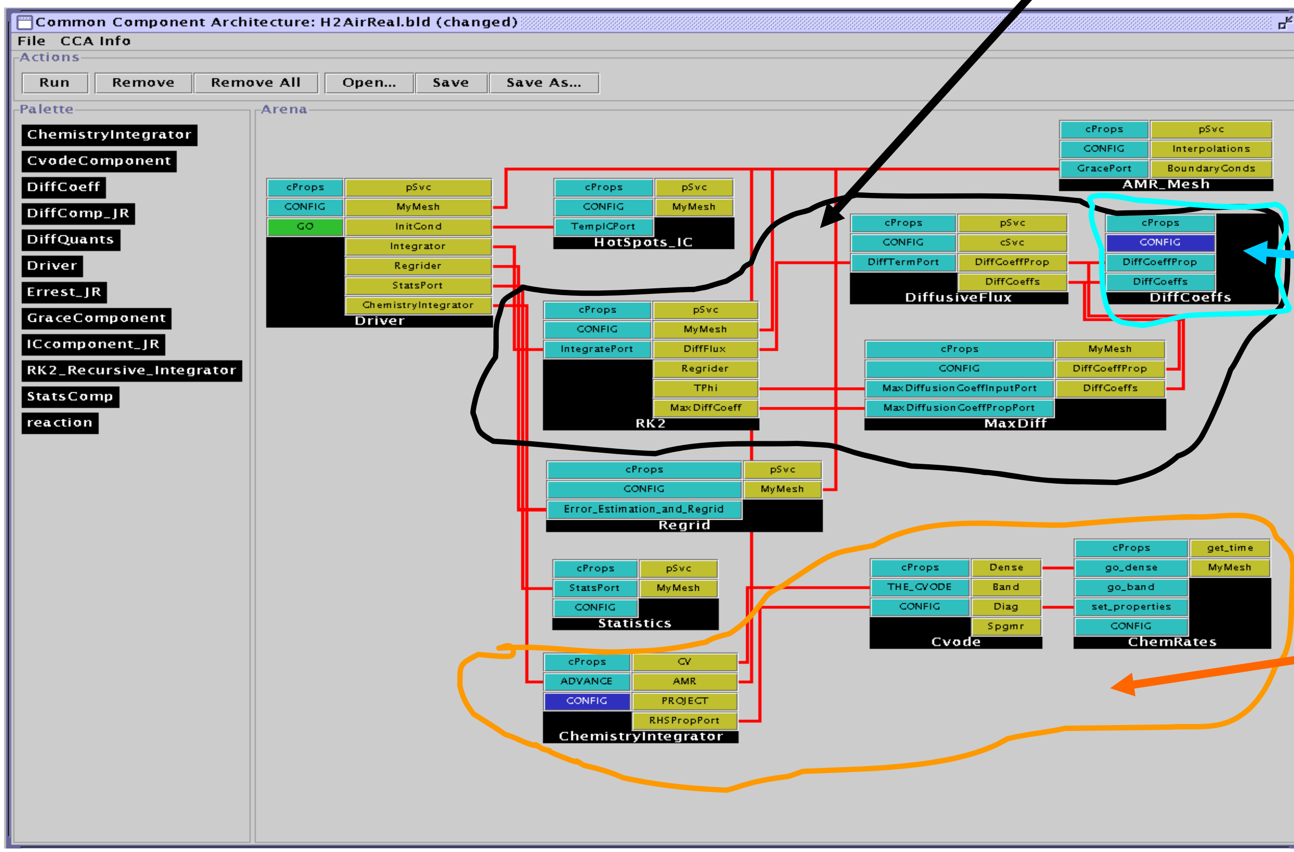
- Decomposition reflected in the port design and implementation
  - Most re-implemented port is the one that exchanges a rectangular sub-domain's data for processing by components



- Sparse connectivity between components
  - i.e., components communicate with a few others
  - Large apps (component assemblies) are composed by assembling smaller, largely independent sub-assemblies
    - Sub-assemblies usually deal with a certain physics
  - Intuitive way to assemble a *multiphysics* code

The Code

Transport subassembly



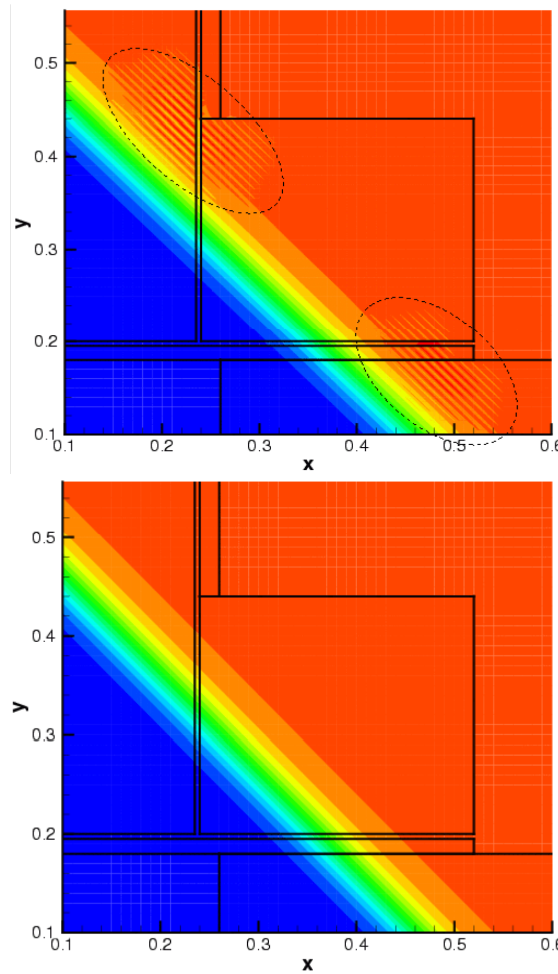
Diffusion coefficients

Chemistry reaction subassembly

Separate component subsystems for transport (black) and for reaction (orange) in a reaction-diffusion code. They two are coupled at a relatively high level.

CFRFS Toolkit Status

- Started in 2001
- 100+ components today, all peers, independent, mixed and matched for combustion and shock hydrodynamics
- 8 external libraries
- Contributors: 13 in all, including 3 summer students
- Only 3 of the 13 contributors are at SNL today



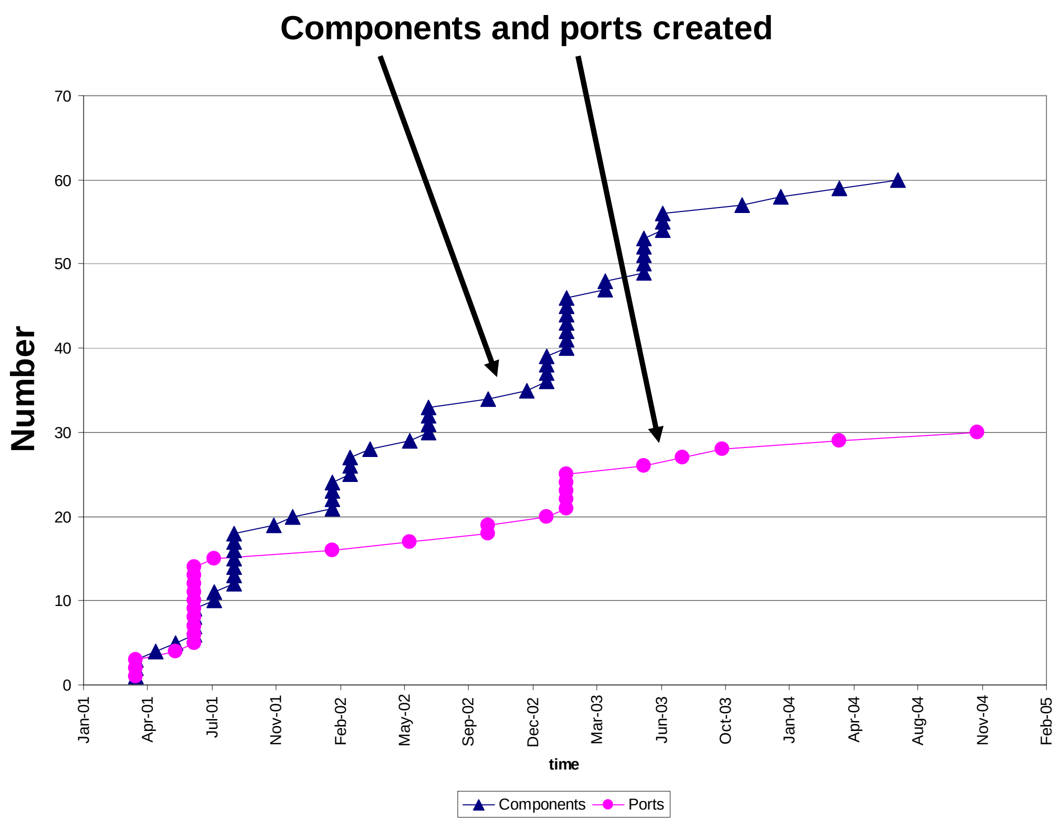
A Fitzhugh-Nagumo equation being solved on a block-structured adaptively refined mesh. The top image illustrates Runge phenomena at coarse-fine interfaces (dashed ovals) when using high-order schemes (6^{th} order interpolations with 4^{th} order discretizations). Filtering them with an 8^{th} order filter removes them (bottom).

Has the Toolkit Approach Helped Tame Software Complexity?

- How has the code evolved?
  - How often have new ports been added?
  - How many rewrites have been done?
- How large are the components?
- How many ports do they have?
  - How large are the ports?
- How many ports exist?
  - i.e., Is the design general enough to support many implementations?
- What is the connectivity of components in application codes?

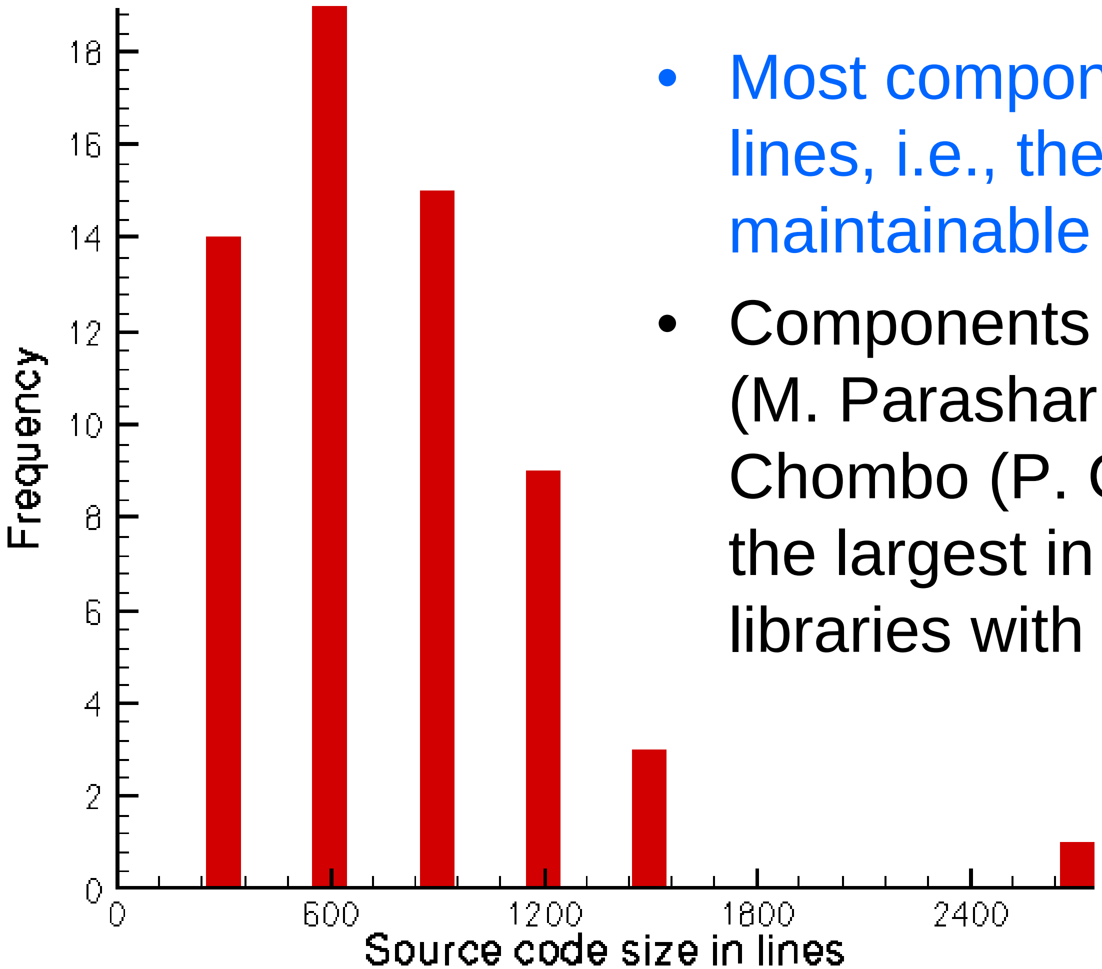
CCA
Common Component Architecture

Scalability: Capability Growth without Rewrites



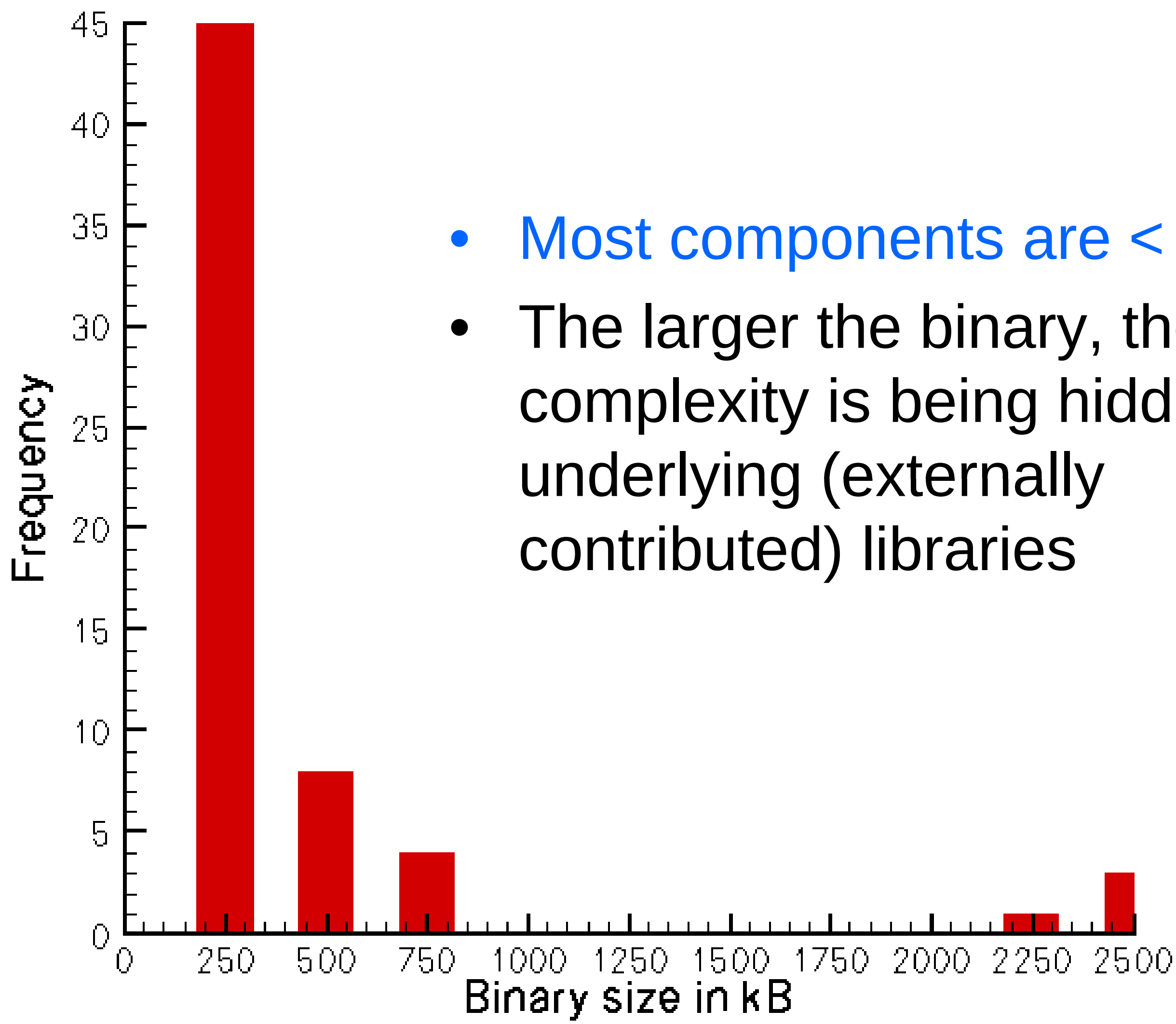
- Port designs typically occur in spurts followed by long component development times.
- Ports may have multiple implementations; hence the number of ports is typically less than the number of components.
- As the toolkit has matured, the number of ports is seen to be asymptoting to a slow growth rate.

Taming Complexity: Lines of Code



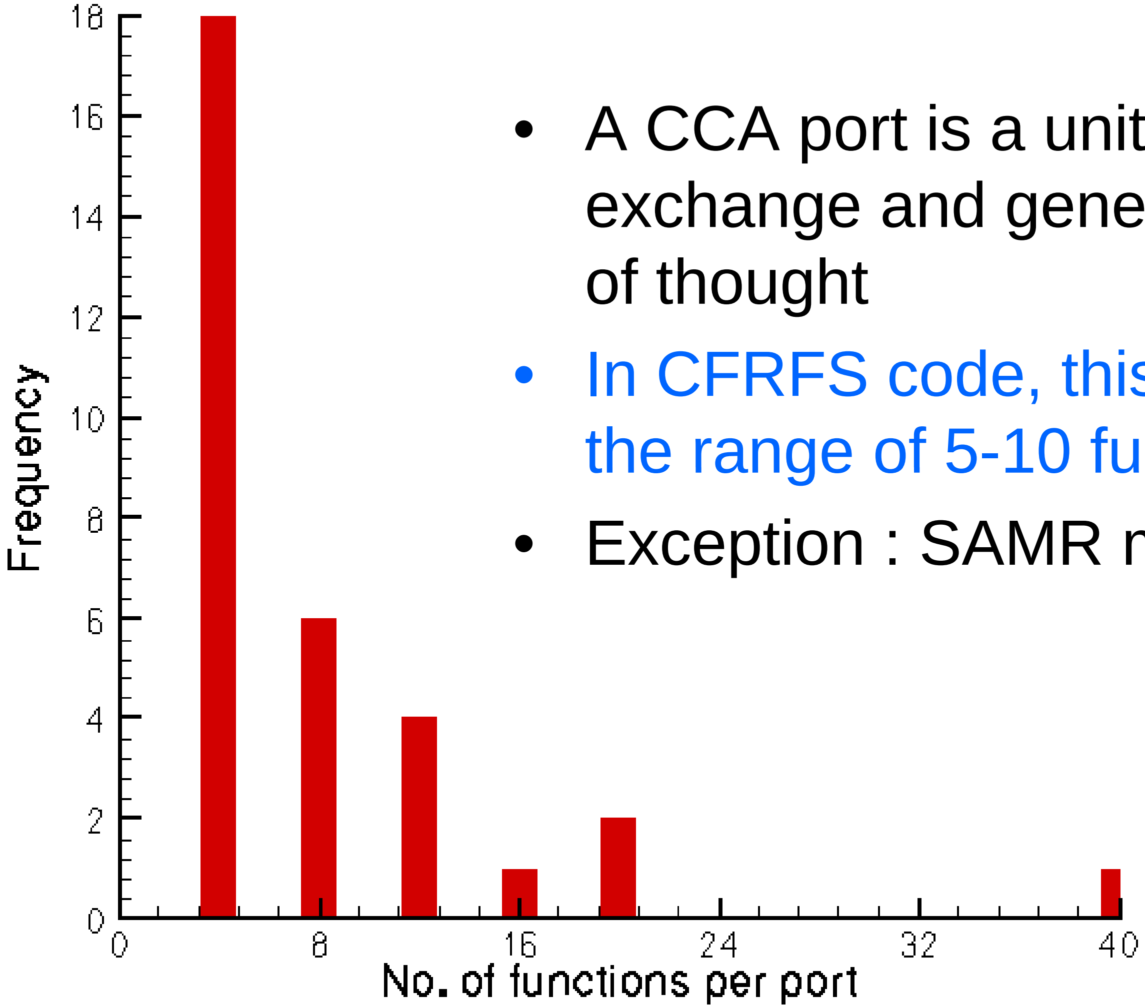
- Most components are < 1000 lines, i.e., they are easily maintainable
- Components based on GrACE (M. Parashar, Rutgers) and Chombo (P. Colella, LBNL) are the largest in size: parallel mesh libraries with load-balancers

Taming Complexity: Code Size



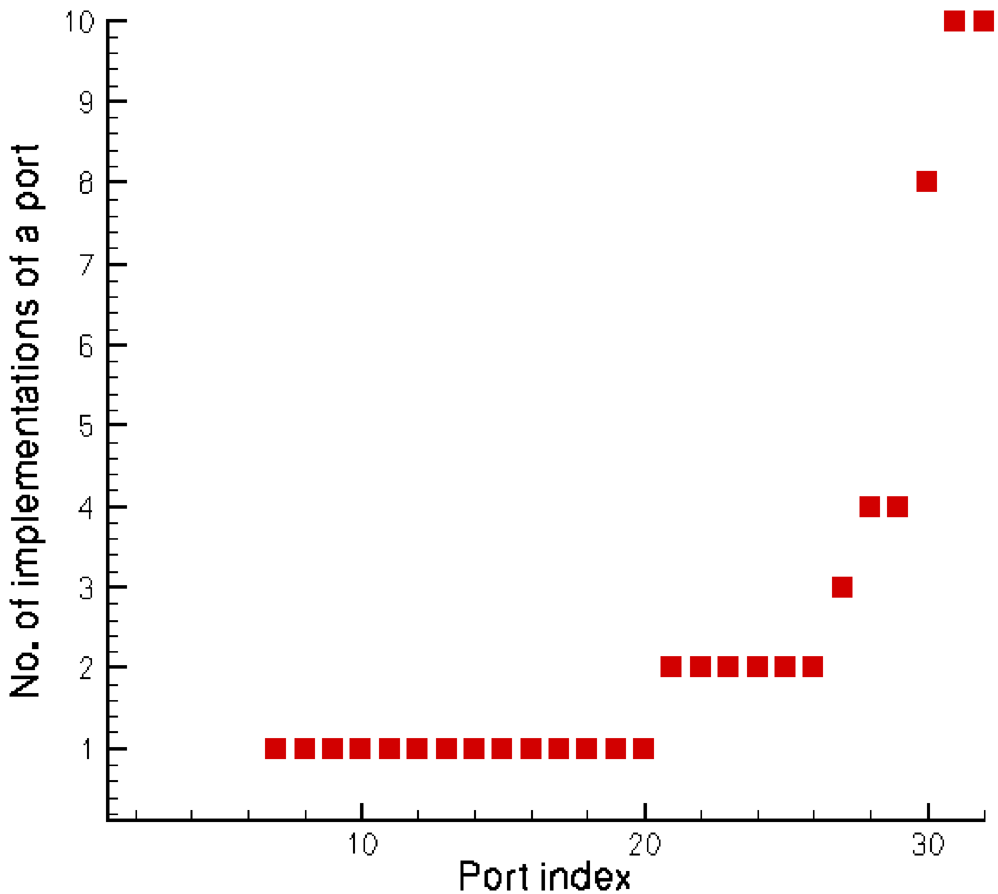
- Most components are < 250 kB
- The larger the binary, the more complexity is being hidden in underlying (externally contributed) libraries

Taming Complexity: Interface Size



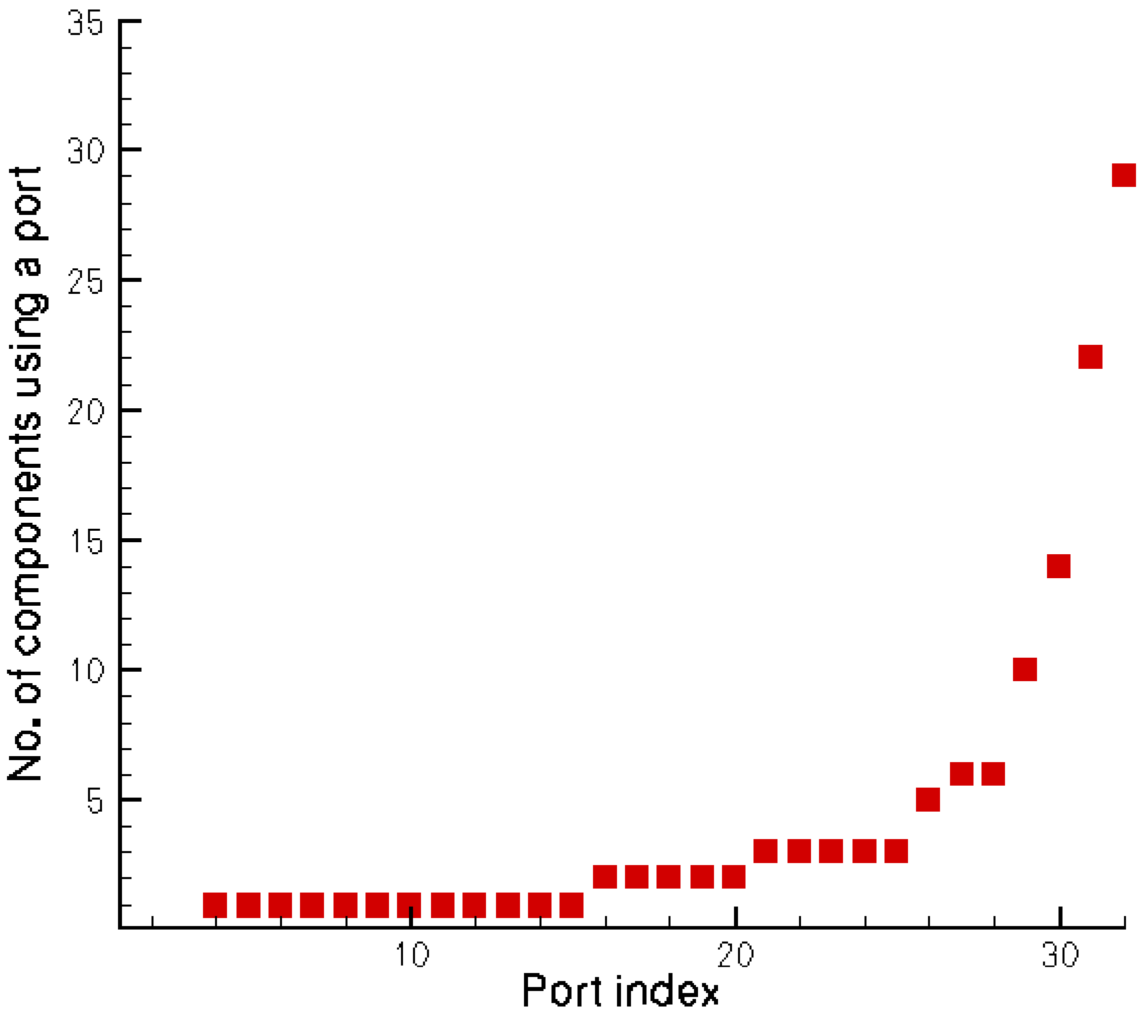
- A CCA port is a unit of task exchange and generally also a unit of thought
- In CFRFS code, this is typically in the range of 5-10 functions
- Exception : SAMR mesh data port

Taming Complexity: Implementations



- CFRFS ports may have just one or many imple-mentations, as needed, but ...
- Most ports have 1 or 2 implementations
- High-utility ports exist, e.g., for exchanging a patch's worth of data

Taming Complexity: Callers

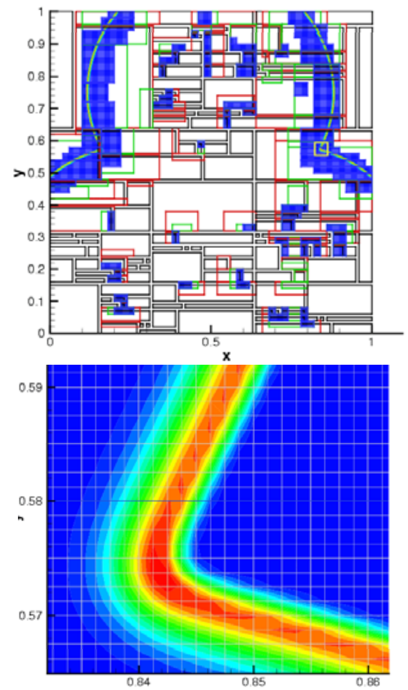


- Most CFRFS ports are used by only a few clients, but …
- Key ports are used by many components

CCA Common Component Architecture

Scientific Productivity

- Conventional Measures (May 2008)
  - 5 journal papers in CFD/Numerics
  - 4 software-oriented journal papers, 1 book chapter
  - Over 15 conference papers, including best paper award
  - Over 60 presentations
  - 1 MS and 3 PhD theses
  - 6 test applications
  - See papers at: http://cfrfs.ca.sandia.gov
- Unconventional Measures
  - Did the toolkit spawn new research in app-focused CS (e.g., performance evaluation/enhancement/modeling?)
  - Can the design accommodate software which were themselves designed to be *frameworks* and *not components* ?



H_2O_2 chemical subspecies profile and associated AMR mesh

CCA
Common Component Architecture

Adaptive Codes / Computational Quality of Service

- Can codes be optimized during runtime ?
  - CCA code is an assembly of components
  - Component applications can be non-optimal because
    - The mathematical characteristics of the simulation are different from those of the component
    - The component is badly implemented
- So, can one dynamically “re-arrange” an assembly to improve performance?
- Simplistically, 2 approaches:
  - Create performance model per component, use best one
  - Create expert system that analyzes problem and picks good solution strategy

Requirements for CQoS codes

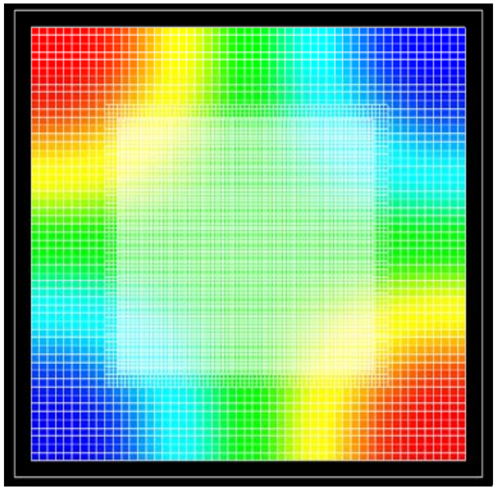
- A software framework within which ordinary components can be "CQoS-ified"
  - By "decorating" a given component-network with extra CQoS components
- The 'extra" components form a control system which monitors, tweaks and/or replaces components commensurate with the problem and the computer at hand.
  - Requires continuous performance monitoring
    - How do you monitor a component non-invasively?
  - Requires continuous querying of a performance database to analyze current situation and make predictions
- The control infrastructure/framework/software is generic, the control law is domain specific.

Who's working on CQoS ?

- Control law work:
  - Computational chemistry: evaluation of integrals
  - Partitioning of block-structured adaptive meshes
  - Iterative sparse linear algebra: mix-and-match accelerators and preconditioners
- Infrastructure work
  - Performance measurement: Automatic performance data (statistical) analysis
  - Design of a CQoS API (to enable extraction & storage of performance data)
  - Quick-and-dirty experimentation in a CQoS world (SciPY based)
  - Infrastructural components – databases, regression-ers etc

Incorporating Other Frameworks

- Chombo (by P. Colella, LBNL) has solvers (AMRGodunov, AMRElliptic, etc.) that:
  - Work on block structured adaptive meshes
  - Accept Chombo-specific data structures
  - But fundamentally require:
    - A double pointer, an array, where variables on a patch (box) are stored in blocked format
    - Bunch of integer arrays that describe the array



Using Chombo to solve the Poisson equation (needed for CFRFS flame simulations).

- Challenge: Can Chombo be used within CCA?
  - Currently under testing, as part of CFRFS Toolkit
  - No existing CFRFS components were modified
  - No data copies

Using CCA: Summary

- Review of guidelines for developing high-performance scientific components (both new code and wrappers for existing code)
- CCA is an enabling technology for scientific applications
  - Has enabled mathematicians, chemists, combustion scientists, and computer scientists to contribute new strategies that are shrink-wrapped for easy re-use
  - Apart from ***science research***, also spawned ***new research directions in CS***
  - Has enabled research scientists to investigate unconventional approaches, for example multilevel parallelism and dynamic adaptivity
- For more info on the CCA applications/case studies, see:
  - Chemistry: http://www.cca-forum.org/~cca-chem
  - Combustion: http://cfrfs.ca.sandia.gov
- **Different facets of CCA components may be useful within different projects … What are your needs and priorities?**

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…

  help@cca-forum.org or cca-tutorial@cca-forum.org

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
  - help@cca-forum.org or cca-tutorial@cca-forum.org
- Problems with specific tools
  - help@cca-forum.org
  - cca-tools bundle (includes Chasm, Babel, Ccaffeine): Rob Armstrong, cca-tools@cca-forum.org
  - Bocca: bocca-dev@cca-forum.org
  - Babel: babel-users@llnl.gov
  - Ccaffeine: ccafe-users@cca-forum.org
- General questions, or not sure who to ask?
  - help@cca-forum.org

Acknowledgements: Tutorial Working Group

- **People:** Benjamin A. Allan, Rob Armstrong, David E. Bernholdt, Randy Bramley, Tamara L. Dahlgren, Lori Freitag Diachin, Tony Drummond, Wael Elwasif, Tom Epperly, Madhusudhan Govindaraju, Ragib Hasan, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Alan Morris, Stefan Muszala, Boyana Norris, Craig Rasmussen, Jaideep Ray, Sameer Shende
- **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
- **Supported** in part by the Scientific Discovery through Advanced Computing (SciDAC) program of the U.S. Dept. of Energy Office of Science, Office of Advanced Scientific Computing Research

Acknowledgements: The CCA

- **Ames Lab –** Masha Sosonkina, …
- **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, …
- **Iowa State University** - Theresa Windus, …
- **LANL** - Craig Rasmussen, …
- **LLNL** – Tammy Dahlgren, Lori Freitag Diachin, Tom Epperly, Scott Kohn, Gary Kumfert, …
- **Louisiana State University** – Dan Katz, …
- **NASA/Goddard** – Shujia Zhou
- **ORNL** - David Bernholdt, Wael Elwasif, Jim Kohl, Torsten Wilde, …
- **PNNL** – Manoj Krishnan, Jarek Nieplocha, Bruce Palmer, …
- **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, Matt Sottile…
- **University of Utah** – Koasta Damevski, Steve Parker, …

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

Thank You!

Thanks for attending this tutorial

We welcome feedback and questions