



**CCA**  
Common Component Architecture

# **Welcome to the Common Component Architecture Tutorial**

HPCS Team Meeting  
30 June 2005

**CCA Forum Tutorial Working Group**  
[http://www.cca-forum.org/tutorials/  
tutorial-wg@cca-forum.org](http://www.cca-forum.org/tutorials/tutorial-wg@cca-forum.org)

1



**CCA**  
Common Component Architecture

## **Agenda & Table of Contents**

<b>Time</b>	<b>Title</b>	<b>Slide No.</b>	<b>Presenter</b>
9:00-9:05am	Welcome	1	Wael Elwasif, ORNL
9:05-10:00am	A Pictorial Introduction to Components in Scientific Computing	7	Dan Katz, JPL
	An Introduction to Components & the CCA	27	Dan Katz, JPL
10:00-10:15am	Distributed Computing with the CCA	68	Wael Elwasif, ORNL
10:15-10:30am	<i>Break</i>		
10:30-11:20am	CCA Applications	85	Wael Elwasif, ORNL
11:20-11:45am	Language Interoperable CCA Components with Babel	131	Tammy Dahlgren, LLNL
11:45am-12:00n	Questions		All

Note: Due to time constraints, this presentation is slightly condensed relative to the notes provided.

2



## 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](mailto:cca-forum@cca-forum.org)

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

3



## Acknowledgements: Tutorial Working Group

- **People:** Rob Armstrong, David Bernholdt, Randy Bramley, Wael Elwasif, Tammy Dahlgren, Lori Freitag Diachin, Madhusudhan Govindaraju, Ragib Hasan, Dan Katz, Jim Kohl, Gary Kumpfert, Lois Curfman McInnes, 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.

4



## 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, ...
  - **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!

5



## 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](mailto:tutorial-wg@cca-forum.org)

6



**CCA**  
Common Component Architecture

## A Pictorial Introduction to Components in Scientific Computing

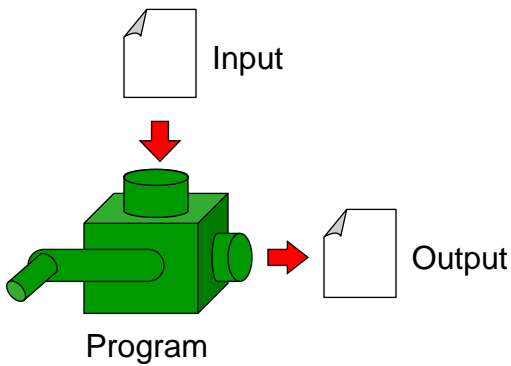
**CCA Forum Tutorial Working Group**  
[http://www.cca-forum.org/tutorials/  
tutorial-wg@cca-forum.org](http://www.cca-forum.org/tutorials/tutorial-wg@cca-forum.org)

7




**CCA**  
Common Component Architecture

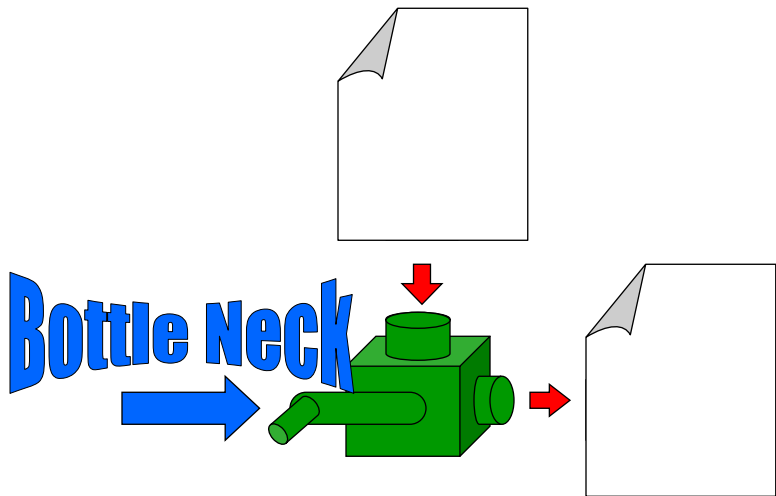
Once upon a time...



8


 **CCA**  
Common Component Architecture

### As Scientific Computing grew...

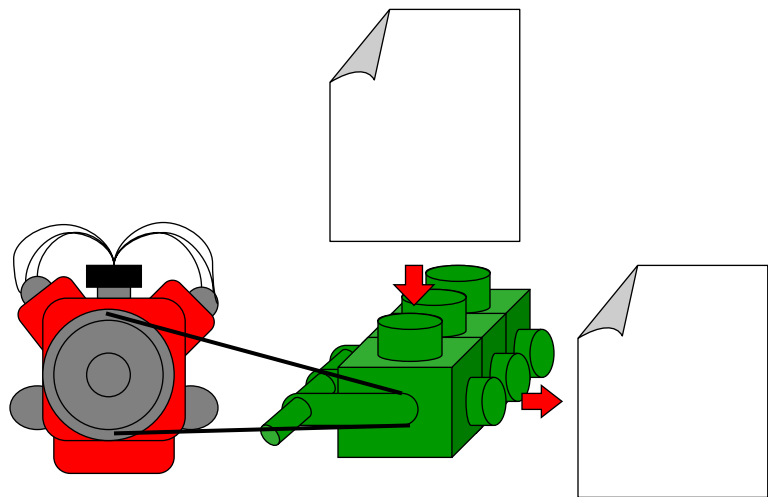


The diagram illustrates a workflow where a document (input) is processed by a green mechanical assembly. A blue arrow labeled "Bottle Neck" points to the assembly, indicating a bottleneck in the process. A red arrow points from the assembly to a second document (output).

9

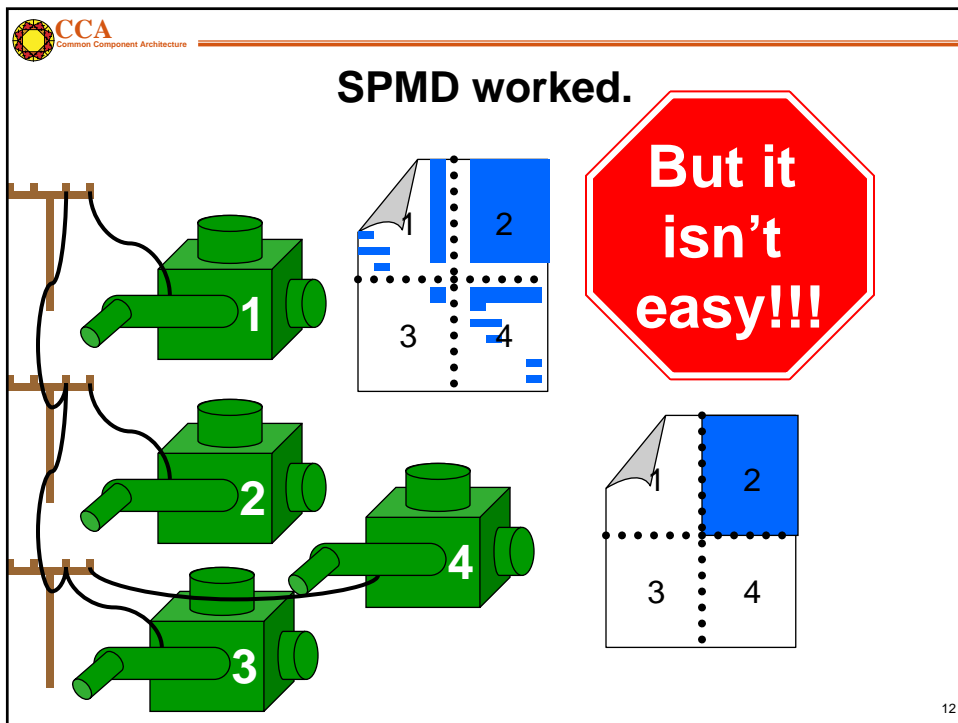
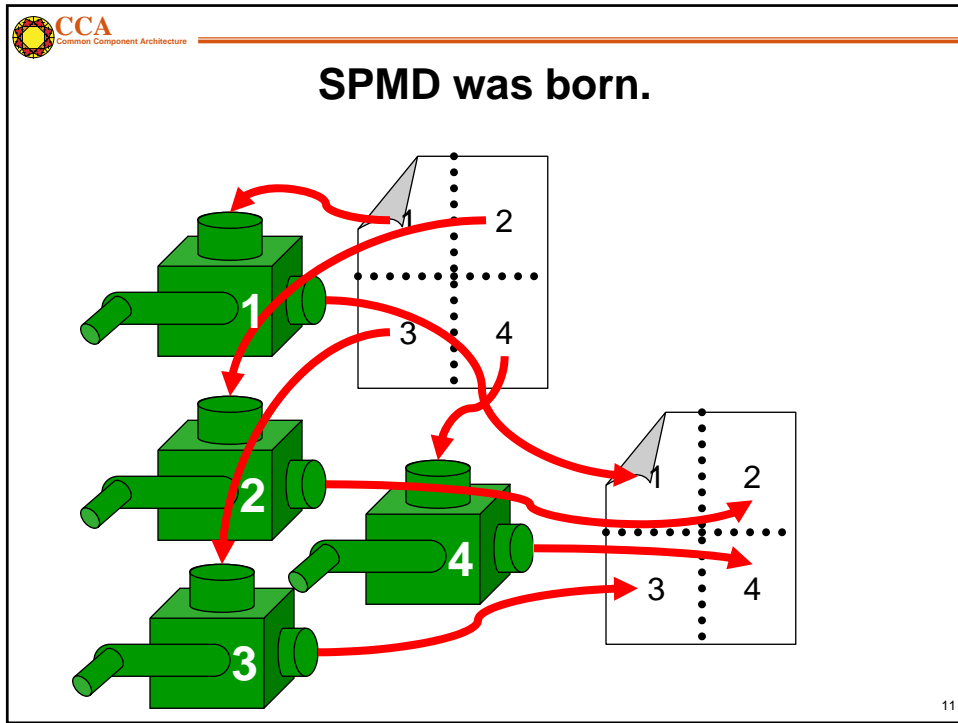
 **CCA**  
Common Component Architecture

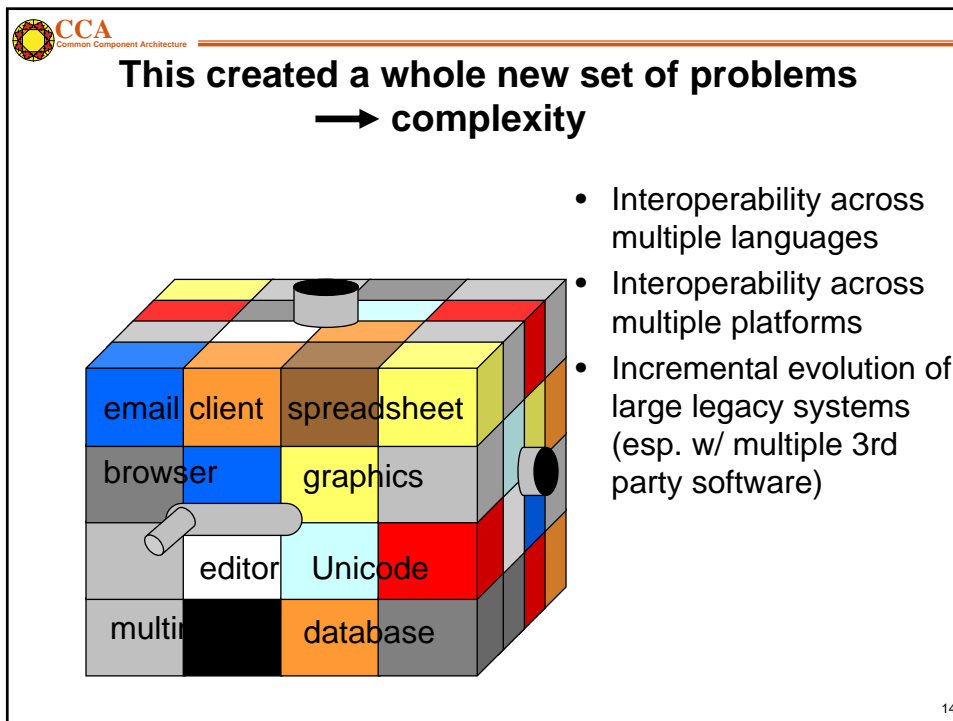
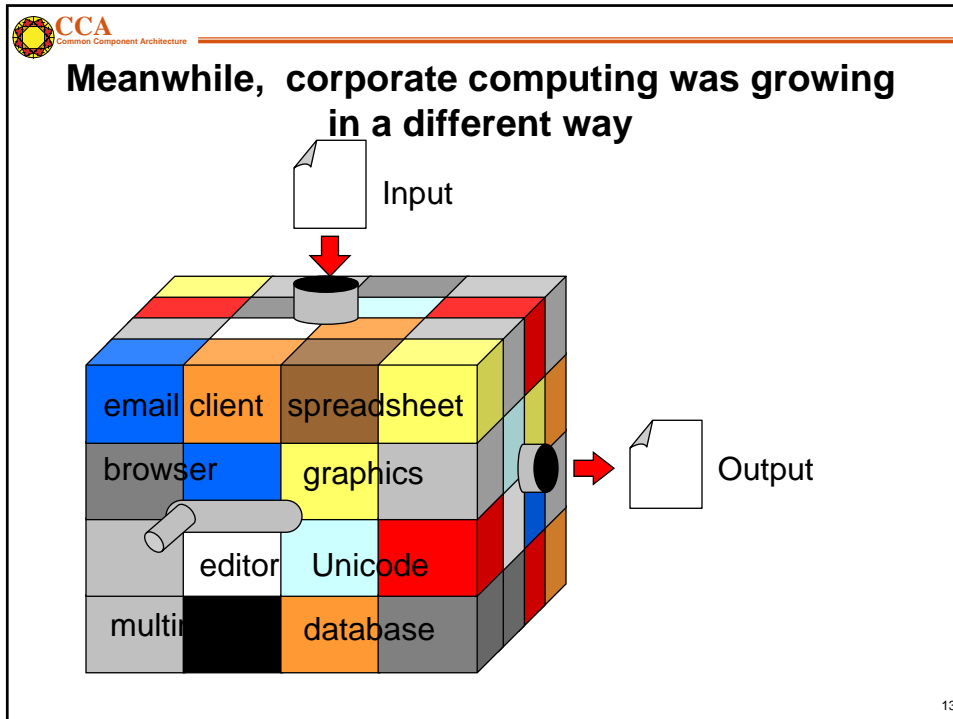
### Tried to ease the bottle neck

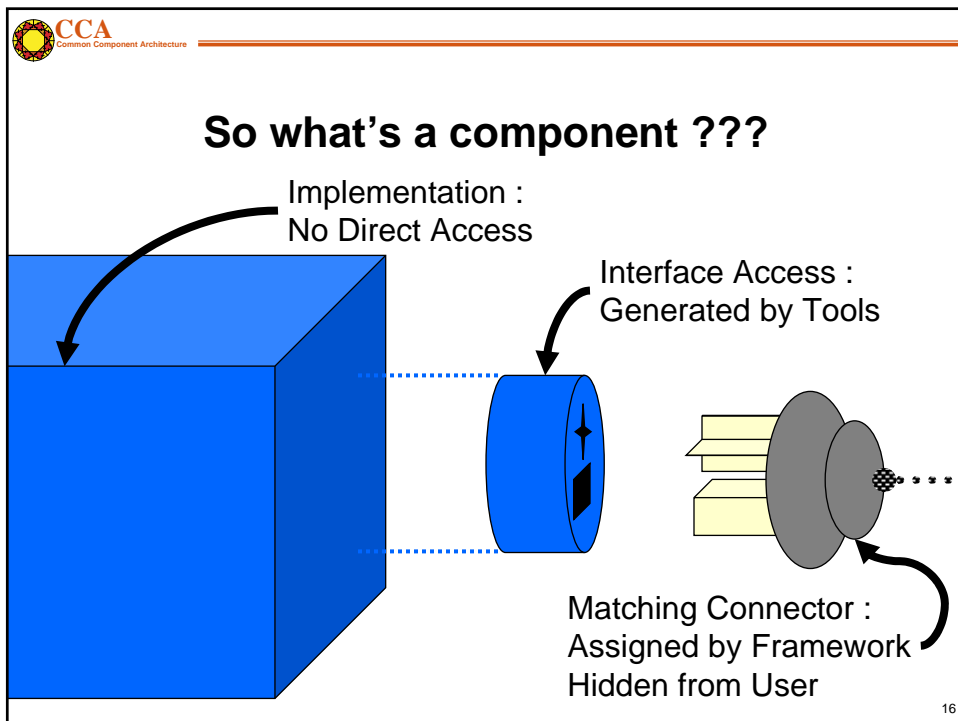
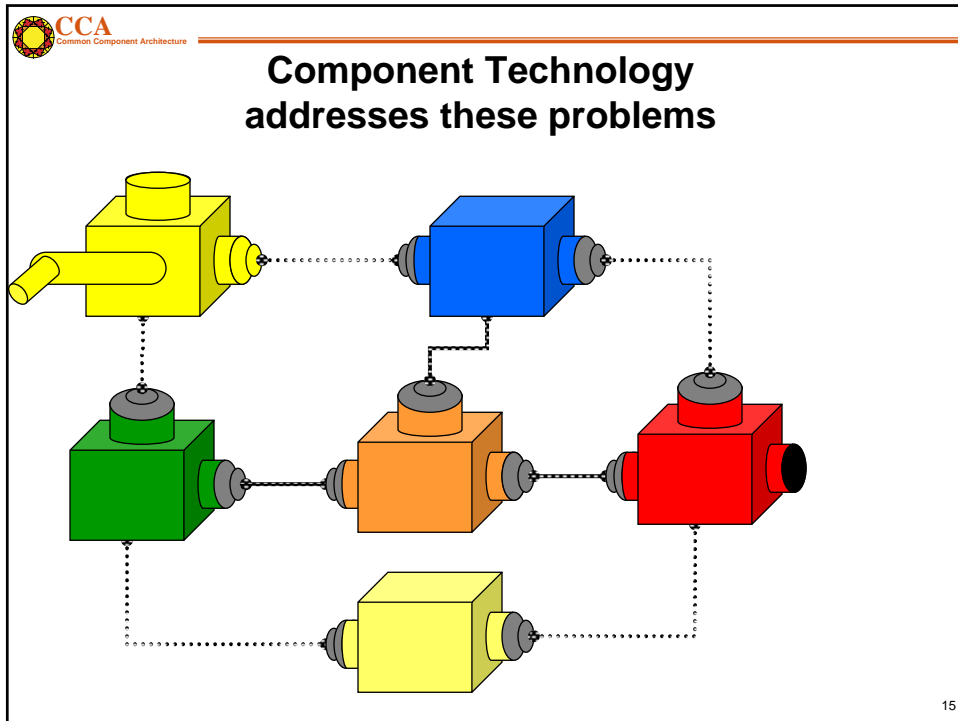


The diagram illustrates an attempt to ease the bottleneck by adding a red engine to the green mechanical assembly. A black line connects the engine to the assembly, suggesting a power source or drive mechanism. A red arrow points from the assembly to a document icon, representing the output.

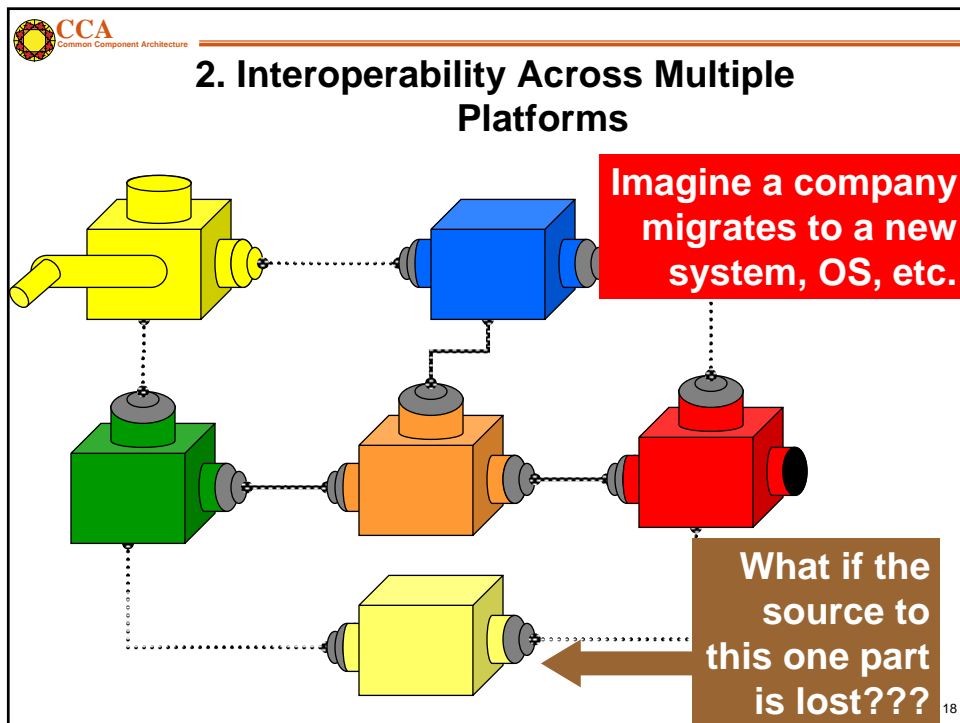
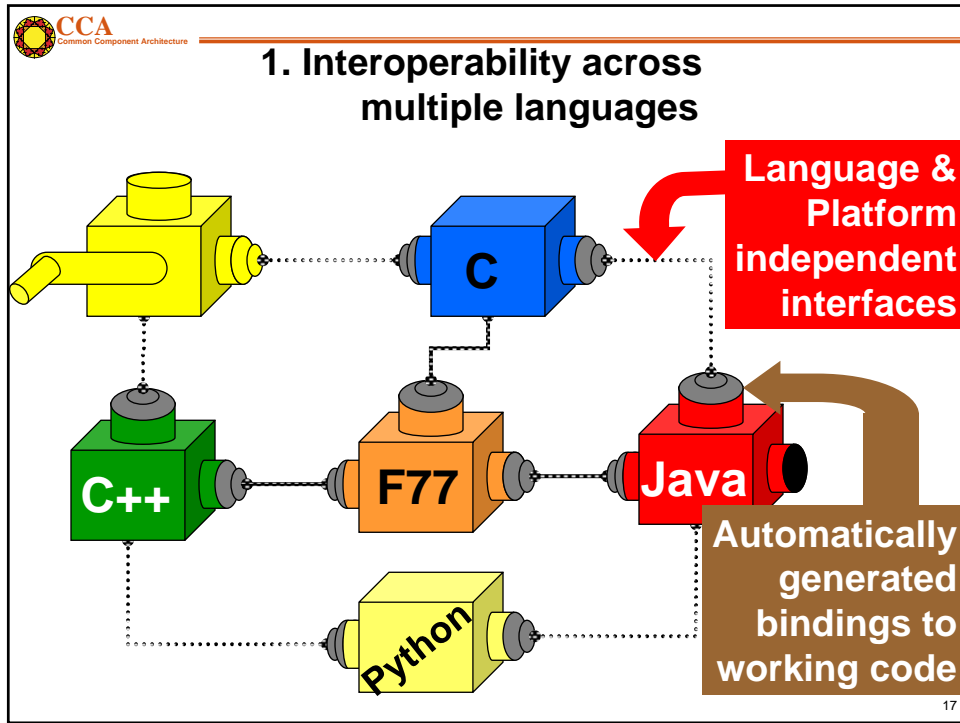
10

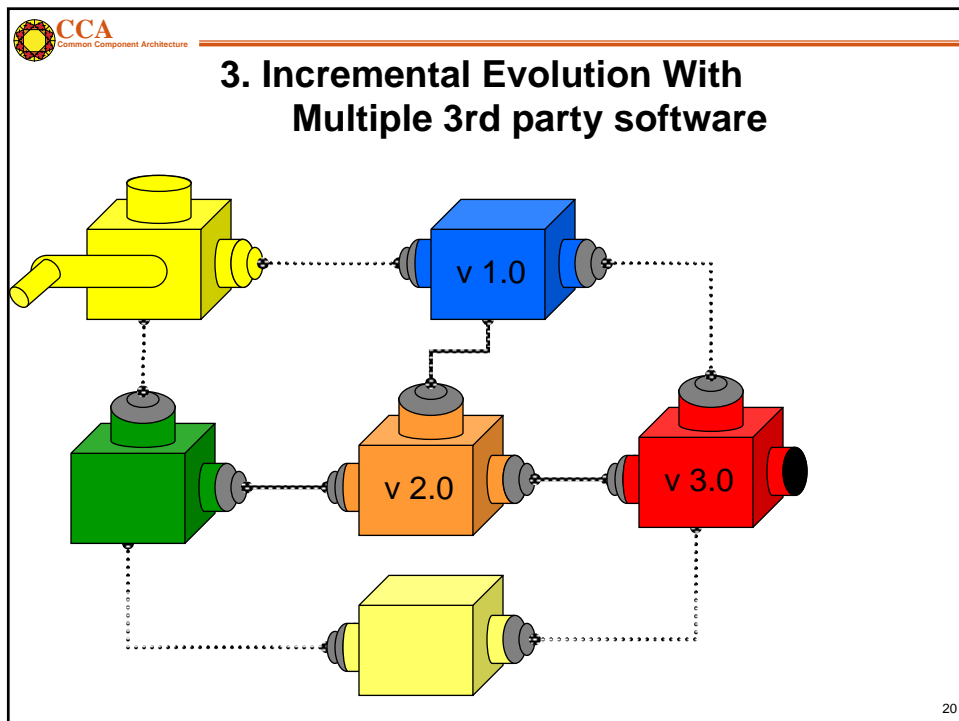
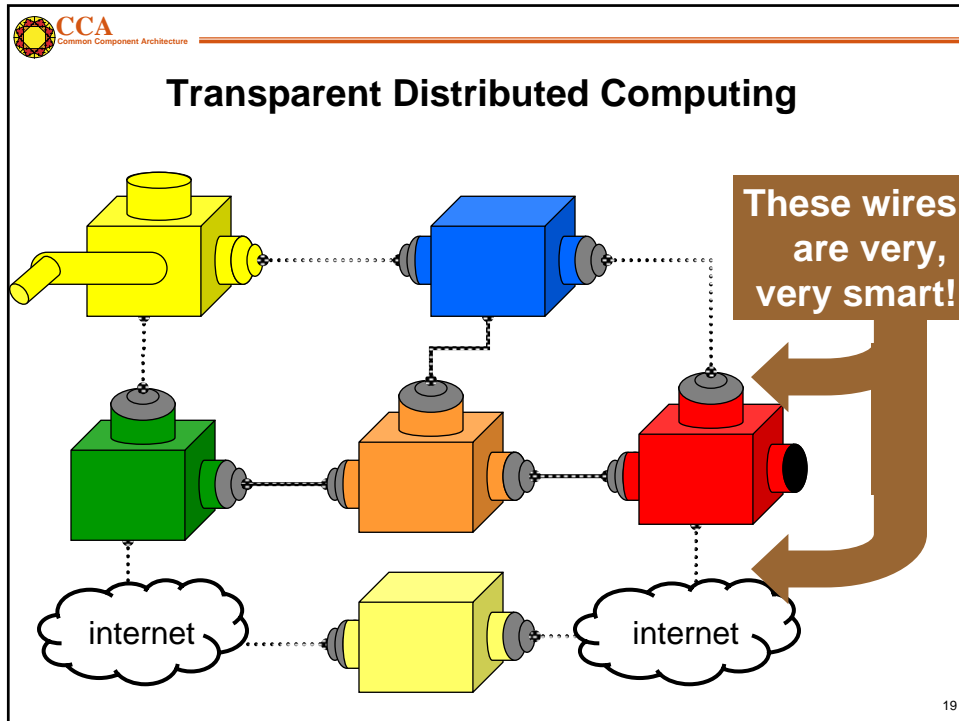


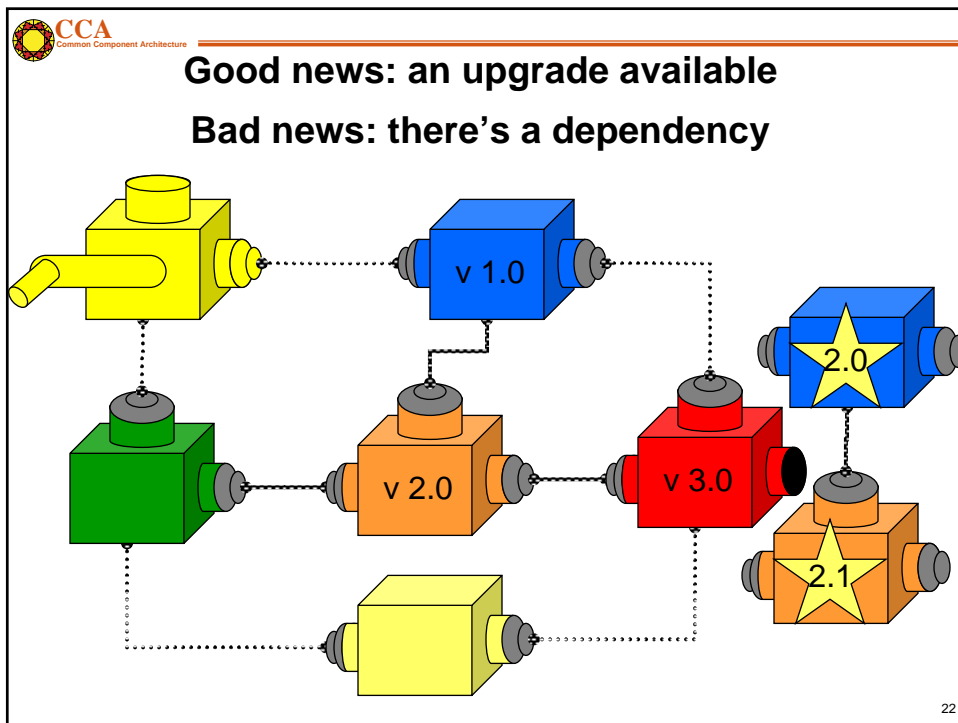
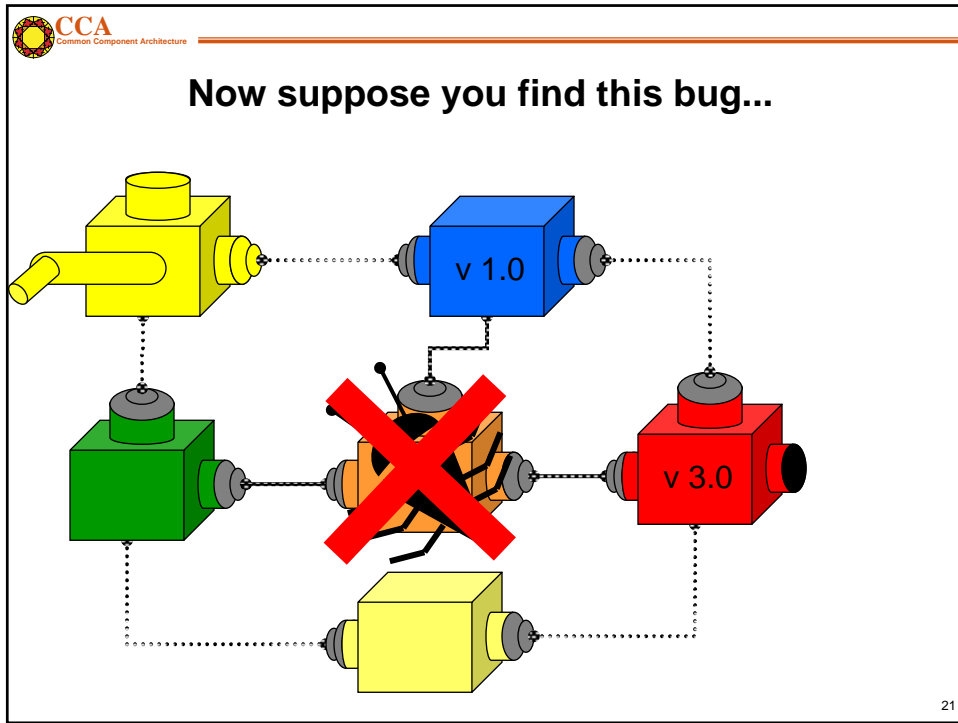


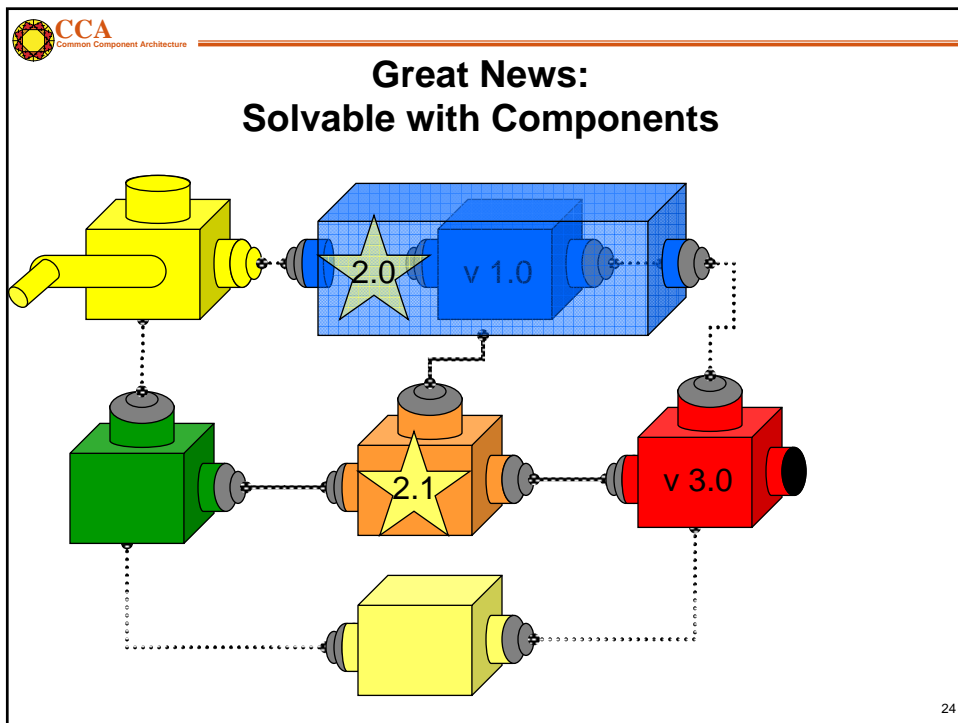
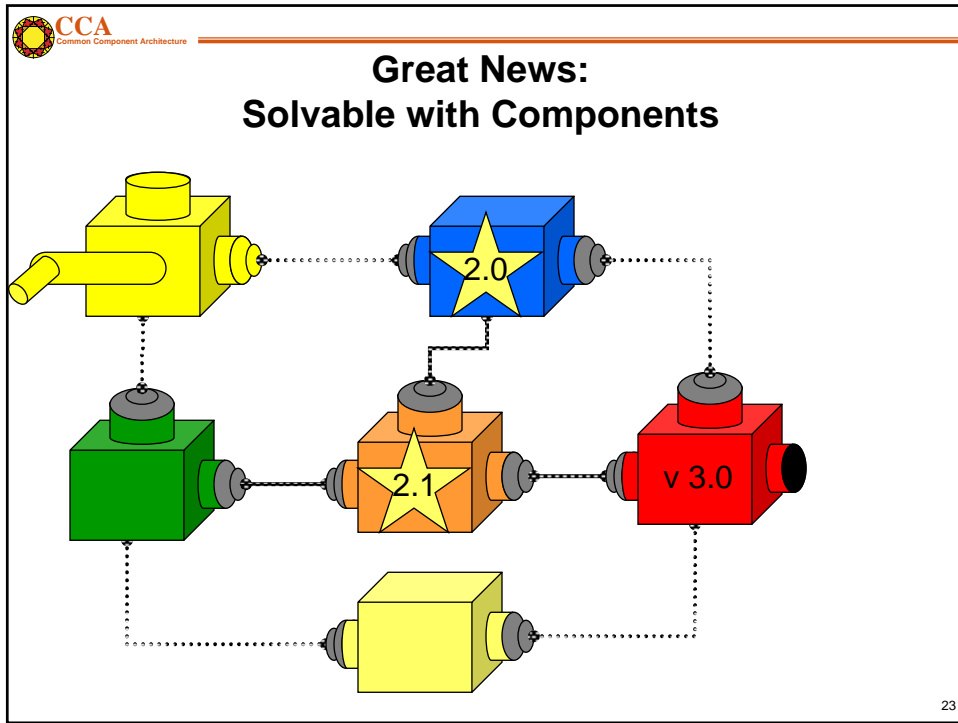












**CCA**  
Common Component Architecture

## Why Components for Scientific Computing → Complexity

- Interoperability across multiple languages
- Interoperability across multiple platforms
- Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)

25

**CCA**  
Common Component Architecture

## The Model for Scientific Component Programming

Industry

Science

CCA

26



# 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*

27



## 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

28



## Component-Based Software Engineering

- CBSE methodology is an emerging approach to software development
  - Both in research and in practical application
  - Especially popular in business and internet areas
- Addresses software **complexity** issues
- Increases software **productivity**

29



## Motivation: For Library Developers

- People want to use your software, but need wrappers in languages you don't support
  - Many component models provide language interoperability
- Discussions about standardizing interfaces are often sidetracked into implementation issues
  - Components separate interfaces from implementation
- You want users to stick to your published interface and prevent them from stumbling (prying) into the implementation details
  - Most component models actively enforce the separation

30



## Motivation: For Application Developers and Users

- You have difficulty managing **multiple third-party libraries** in your code
- You (want to) use **more than two languages** in your application
- Your code is **long-lived** and different pieces **evolve** at different rates
- You want to be able to **swap** competing implementations of the same idea and **test** without modifying any of your code
- You want to **compose** your application with some other(s) that weren't originally designed to be combined

31



## What are Components?

- No universally accepted definition in computer science research ...yet
- 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**

32



**CCA**  
Common Component Architecture

## 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 are composed to form an application and executed (**framework**)
  - The rights and responsibilities of the framework

33

**CCA**  
Common Component Architecture

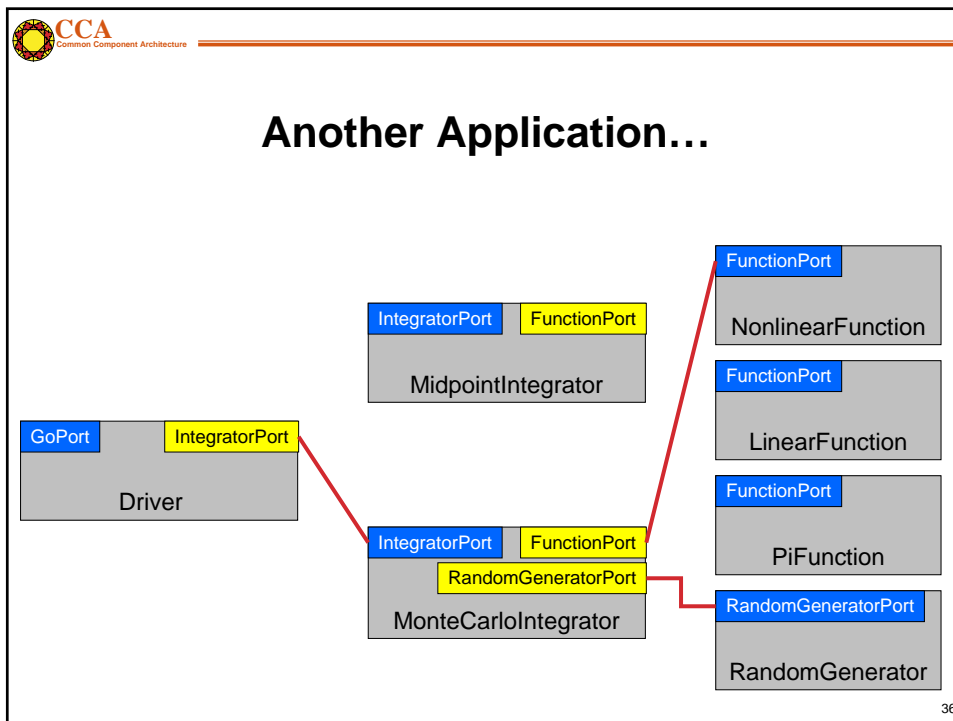
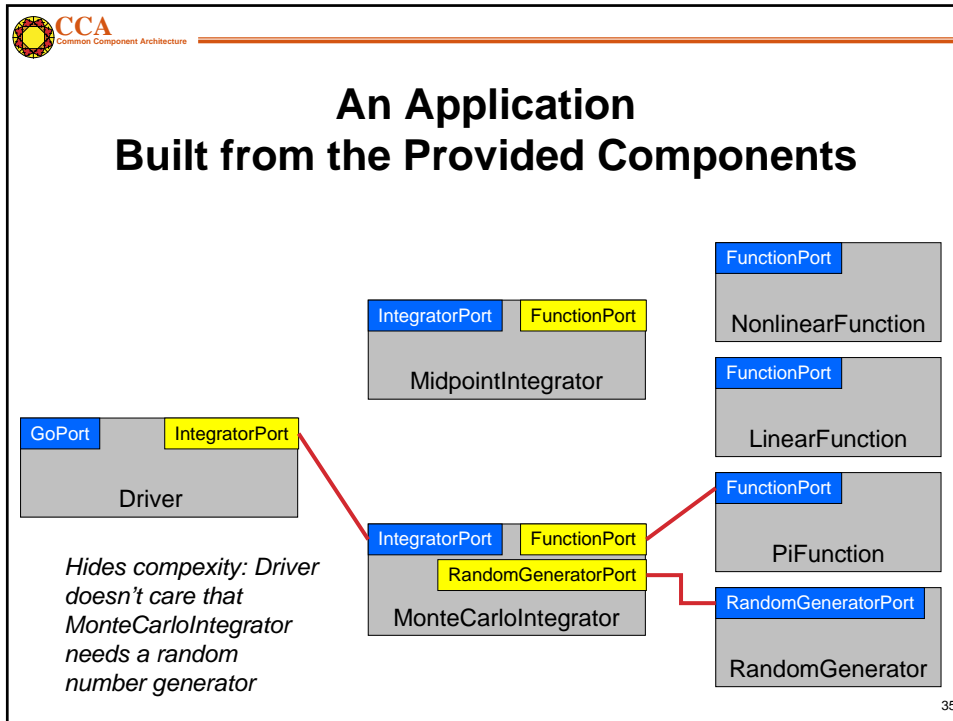
## A Simple Example: Numerical Integration Components

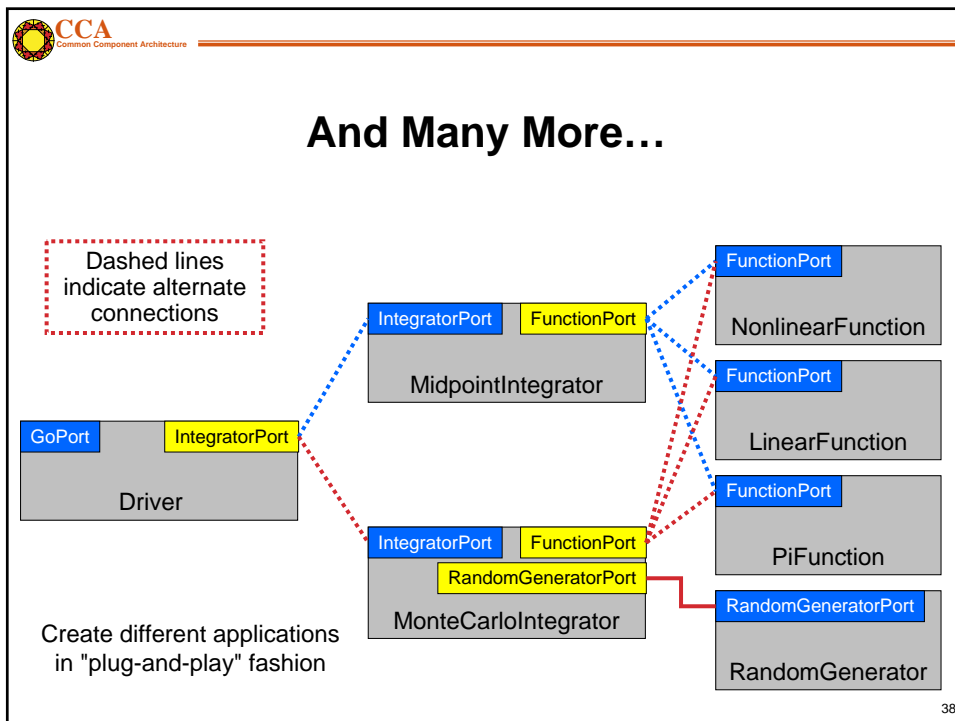
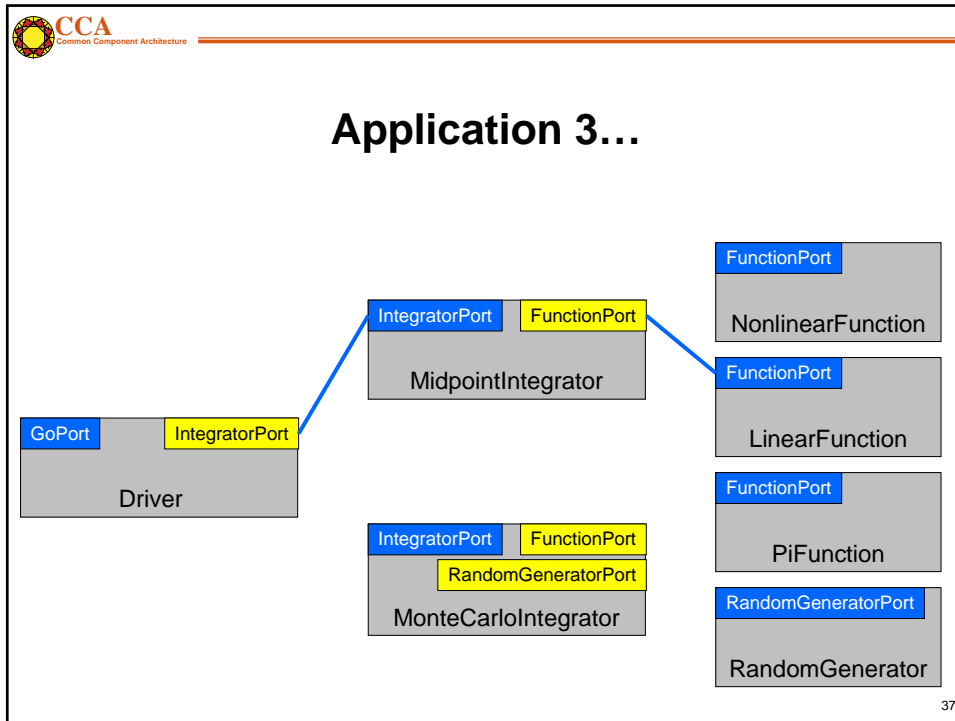
*Interoperable components  
(provide same interfaces)*


The diagram illustrates the following components and their interfaces:

- Driver**: Contains **GoPort** and **IntegratorPort**.
- MidpointIntegrator**: Contains **IntegratorPort** and **FunctionPort**.
- MonteCarloIntegrator**: Contains **IntegratorPort**, **FunctionPort**, and **RandomGeneratorPort**.
- NonlinearFunction**: Contains **FunctionPort**.
- LinearFunction**: Contains **FunctionPort**.
- PiFunction**: Contains **FunctionPort**.
- RandomGenerator**: Contains **RandomGeneratorPort**.

34








**CCA**  
Common Component Architecture

---

## 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

39




**CCA**  
Common Component Architecture

---

## Domain-Specific Frameworks vs Generic Component Architectures

<b>Domain-Specific</b>	<b>Generic</b>
<ul style="list-style-type: none"><li>• Often known as “frameworks”</li><li>• Provide a significant software infrastructure to support applications in a <b>given domain</b><ul style="list-style-type: none"><li>– Often attempts to generalize an existing large application</li></ul></li><li>• Often hard to adapt to use outside the original domain<ul style="list-style-type: none"><li>– Tend to assume a <b>particular structure/workflow</b> for application</li></ul></li><li>• Relatively <b>common</b><ul style="list-style-type: none"><li>– E.g. Cactus, ESMF, PRISM</li><li>– Hypre, Overture, PETSc, POOMA</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Provide the infrastructure to <b>hook components</b> together<ul style="list-style-type: none"><li>– Domain-specific infrastructure can be built as components</li></ul></li><li>• Usable in <b>many domains</b><ul style="list-style-type: none"><li>– Few assumptions about application</li><li>– <b>More opportunities for reuse</b></li></ul></li><li>• Better supports <b>model coupling</b> across traditional domain boundaries</li><li>• Relatively <b>rare</b> at present<ul style="list-style-type: none"><li>– e.g. CCA</li></ul></li></ul>

40



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

41



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?

42



## Commodity Component Models

- CORBA Component Model (CCM), COM, Enterprise JavaBeans
  - Arise from business/internet software world
- Componentization **requirements** can be **high**
- Can impose significant **performance overheads**
- No recognition of **tightly-coupled parallelism**
- May be **platform specific**
- May have **language constraints**
- May not support common scientific **data types**


43



## What is the CCA?

- CCA is a **specification of a component environment** designed for **high performance scientific computing**
  - Specification is decided by the **CCA Forum**
    - CCA Forum membership open to all
  - “**CCA-compliant**” just means conforming to the specification
    - Doesn't require using any of our code!
- A **tool** to enhance the productivity of scientific programmers
  - Make the hard things easier, make some intractable things tractable
  - Support & promote reuse & interoperability
  - **Not a magic bullet**

44


 **CCA**  
Common Component Architecture

---

## CCA Philosophy and Objectives

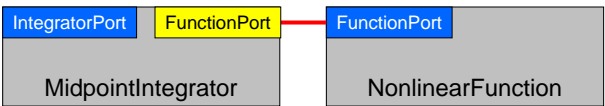
- **Local and remote components**
  - Support local, HPC parallel, and distributed computing
- **High Performance**
  - Design should support high-performance mechanisms wherever possible (i.e. minimize copies, extra communications, extra synchronization)
  - Support SPMD and MPMD parallelism
  - Allow user to choose parallel programming models
- **Heterogeneity**
  - Multiple architectures, languages, run-time systems used simultaneously in an application
- **Integration**
  - Components should be easy to make and easy to use
- **Openness and simplicity**
  - CCA spec should be open & usable with open software

45

 **CCA**  
Common Component Architecture

---

## CCA Concepts: Components



```
graph LR; MI[MidpointIntegrator] --- IP[IntegratorPort]; MI --- FP1[FunctionPort]; NF[NonlinearFunction] --- FP2[FunctionPort]; FP1 --- FP2;
```

- Components provide/use one or more **ports**
  - A component with no ports isn't very interesting
- Components include some **code which interacts with a CCA framework**

46

CCA  
Common Component Architecture

## CCA Concepts: Ports

- Components interact through well-defined **interfaces**, or **ports**
  - In OO languages, a port is a **class** or **interface**
  - In Fortran, a port is a bunch of subroutines or a **module**
- Components may **provide** ports – **implement** the class or subroutines of the port ( **"Provides" Port** )
- Components may **use** ports – **call** methods or subroutines in the port ( **"Uses" Port** )
- Links between ports denote a procedural (caller/callee) relationship, **not dataflow!**
  - e.g., FunctionPort could contain: *evaluate(in Arg, out Result)*

47


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
- *Currently:* specific frameworks support specific computing models (parallel, distributed, etc.)
- *Future:* full flexibility through integration or interoperation

48





CCA  
Common Component Architecture


---

## Writing Components

- Components...
  - Inherit from **gov.cca.Component**
    - Implement `setServices` method to register ports this component will **provide** and **use**
  - Implement the ports they provide
  - Use ports on other components
    - `getPort/releasePort` from framework `Services` object
- Interfaces (ports) extend **gov.cca.Port**

Full details in the hands-on!

49



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


50

 **CCA**  
Common Component Architecture

## Writing Frameworks

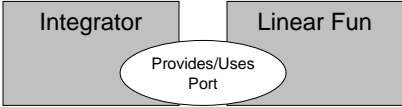
- *There is no reason for most people to write frameworks – just use the existing ones!*
- Frameworks must provide certain ports...
  - **ConnectionEventService**
    - Informs the component of connections
  - **AbstractFramework**
    - Allows the component to *behave as a framework*
  - **BuilderService**
    - Instantiate components & connect ports
  - **ComponentRepository**
    - A default place where components are found
- Frameworks must be able to load components
  - Typically shared object libraries, can be statically linked
- Frameworks must provide a way to compose applications from components

51

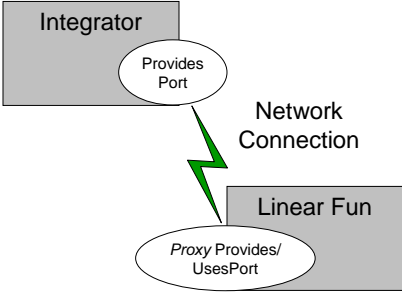
 **CCA**  
Common Component Architecture

## CCA Supports Local, Parallel and Distributed Computing

- “**Direct connection**” preserves high performance of local (“in-process”) 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



Direct Connection



Network Connection

52

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

More about performance in the “Applications” module

53

**CCA**  
Common Component Architecture

## 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)**

P0      P1      P2      P3

Components: Blue, Green, Red

Framework: Gray

*MCMD/MPMD also supported*

*Other component models ignore parallelism entirely*

54

**CCA**  
Common Component Architecture

## “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; highly experimental at present)

55

**CCA**  
Common Component Architecture

## 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

56

**CCA**  
Common Component Architecture

## CCA Concepts: Language Interoperability

- Existing language interoperability approaches are “point-to-point” solutions
- Babel provides a unified approach in which all languages are considered peers
- Babel used primarily at interfaces

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

Babel presentation coming up!

57


**CCA**  
Common Component Architecture

## Advanced CCA Concepts

- Frameworks provide a **BuilderService** which allows **programmatic composition** of components
- Frameworks may **present themselves as components** to other frameworks
- A “traditional” application can treat a CCA framework as a **library**
- Meta-component models** enable bridging between CCA components and other component(-like) environments
  - e.g. SCIRun Dataflow, Visualization Toolkit (VTK), ...

No time to go into detail on these, but ask us for more info after the tutorial

58



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

59



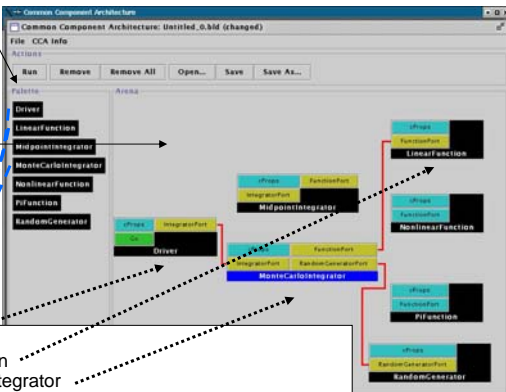
CCA  
Common Component Architecture

Supplementary material for handouts

## 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 LinearFunction MonteCarloIntegrator

create Driver LinearFunction MonteCarloIntegrator

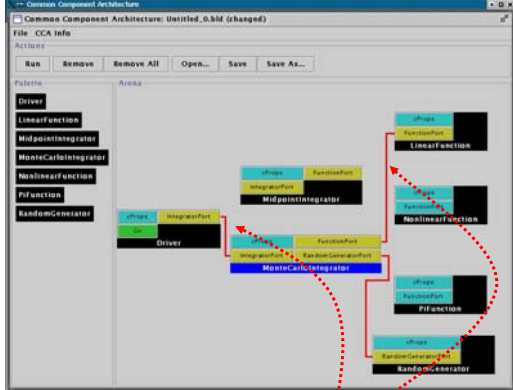
60

Supplementary material for handouts


Common Component Architecture

## 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	FunctionPort
connect	MonteCarloIntegrator	FunctionPort	LinearFunction	IntegratorPort	FunctionPort
...					

61

Supplementary material for handouts


Common Component Architecture

## 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

➔

IntegratorPort

FunctionPort

RandomGeneratorPort

MonteCarloIntegrator

62

Supplementary material for handouts

The diagram illustrates the interaction between two components. On the left, a box represents the **MonteCarloIntegrator**, which contains **Integrator code** and **CCA.Services** (noting that it uses **FunctionPort** and is connected to **NonlinearFunction**). On the right, a box represents the **NonlinearFunction**, which contains **Function code** and **CCA.Services** (noting that it provides **FunctionPort**). A dashed arrow points from the **CCA.Services** in the **NonlinearFunction** to the **CCA.Services** in the **MonteCarloIntegrator**.

## 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**

63

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)`
- **Release** port

The diagram shows the **MonteCarloIntegrator** component box from the previous slide. Dashed green arrows indicate the flow of information and actions: one arrow points from the **CCA.Services** to the **Integrator code**; another points from the **Integrator code** to the **CCA.Services**; a third points from the **Integrator code** to the **Function code** in the **NonlinearFunction** component; and a fourth points from the **Integrator code** back to the **CCA.Services** in the **MonteCarloIntegrator**.

64





## What the CCA isn't...

- CCA doesn't specify who owns "main"
  - CCA components are peers
  - Up to application to define component relationships
    - "Driver component" is a common design pattern
- CCA doesn't specify a parallel programming environment
  - Choose your favorite
  - Mix multiple tools in a single application
- CCA doesn't specify I/O
  - But it gives you the infrastructure to create I/O components
  - Use of stdio may be problematic in mixed language env.
- CCA doesn't specify interfaces
  - But it gives you the infrastructure to define and enforce them
  - CCA Forum supports & promotes common interface efforts
- CCA doesn't require (but does support) separation of algorithms/physics from data
  - Generic programming

65



## What the CCA is...

- CCA is a *specification* for a component environment
  - Fundamentally, a design pattern
  - Multiple "reference" implementations exist
  - Being used by applications
- CCA is designed for interoperability
  - Components within a CCA environment
  - CCA environment with other tools, libraries, and frameworks
- CCA provides an environment in which domain-specific application frameworks can be built
  - While retaining opportunities for software reuse at multiple levels

66



## Concept Review

- **Ports**
  - Interfaces between components
  - Uses/provides model
- **Framework**
  - Allows assembly of components into applications
- **Direct Connection**
  - Maintain performance of local inter-component calls
- **Parallelism**
  - Framework stays out of the way of parallel components
- **Language Interoperability**
  - Babel, Scientific Interface Definition Language (SIDL)


67



## Distributed Computing with the CCA


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

68

 **CCA**  
Common Component Architecture

## Component Composition


- Components can be linked along shared interfaces (ports) where one component invokes the services of another
  - Two types of **Ports**
    - Provides Ports – implements a remote interface
    - Uses Ports – uses a remote interface
  - A user and a provider of the same type can be linked
  - Details of run-time substrate shielded in **stubs and skeletons**
    - Similar in concept to the files generated by Babel



**Uses port -**  
a call site for  
a remote function  
invocation

**Provides Port -**  
A set of functions  
which can be  
invoked remotely

69

 **CCA**  
Common Component Architecture

## How Distributed Frameworks are Different

<p><b>Remote Creation</b></p> <ul style="list-style-type: none"><li>• Launch components in remote address spaces</li><li>• Heterogeneity management</li><li>• Use resource managers to service requests on each remote resource</li><li>• Store, move and replicate component binaries</li></ul>	<p><b>Remote Invocation</b></p> <ul style="list-style-type: none"><li>• Need <b>global pointers</b> and not local pointers</li><li>• Invoke methods across machine boundaries</li><li>• Need <b>global namespace</b> for names of components and services</li><li>• Mechanism for implementing <b>remote method invocation (RMI)</b></li><li>• Introspection mechanisms to allow ports and services to be discovered and accessed</li></ul>
--	---

70



## CCA Concepts that Influence Design of Distributed Frameworks (1)

- Ports
  - References to *provides* ports can move across address spaces
  - *Uses* ports are local to each component
- Services Object is present in each component
  - Manages all the ports
  - Hides details of framework-specific bindings for ports
- ComponentID: opaque handle to the component
  - Should be [serializable](#) and [deserializable](#)
  - Usually points to the [services](#) object


71



## CCA Concepts that Influence Design of Distributed Frameworks (2)

- Builder Service: charged with following operations
  - Create Components in remote address spaces
    - Return a [ComponentID](#) of instantiated components
    - Hide details of heterogeneous remote environments
  - Connect ports of components
    - Facilitate connection between uses and provides ports
      - Only if they are of the same SIDL type
    - Place provides port reference in the uses port table
- Introspection
  - Allow remote querying of a component
    - How many and what type of ports does the component have?

72




CCA  
Common Component Architecture

---

## Key Design Choices for Distributed CCA Frameworks (1)

- How is the CCA **ComponentID** represented in a distributed environment?
  - Handle that can be passed to remote components
  - Serialize and deserialize ComponentID
  - Belong to a namespace understood in the entire framework
  - Should enable optimized communication for co-located components
- How is the **PortType** represented?
  - *Provides* port should be designed as a remote service
  - *Uses* port should be a local object

73



CCA  
Common Component Architecture

---

## Key Design Choices for Distributed CCA Frameworks (2)

- Where can the key CCA functions be called from? Are they remote or local?
  - `getPort()` call on the services object
    - Should return a remote reference for provides ports
    - Note that the same call in the **Ccaffeine** framework returns a local object
  - Details of remote and local calls should be hidden
    - Framework should internally distinguish local and remote calls
- How can components be connected?
  - Need internal mechanism for uses port to obtain remote reference of the provides port
    - Information can be stored in a central table, facilitate development of GUIs to show component assembly
    - Distributed across components so they are aware of who they are connected to

74



## Key Design Choices for Distributed CCA Frameworks (3)

- Should Builder Service be centralized or distributed?
  - A component can have its own builder service if
    - The builder service is lightweight
    - The components has special create/connect requirements



## Current CCA Distributed Frameworks

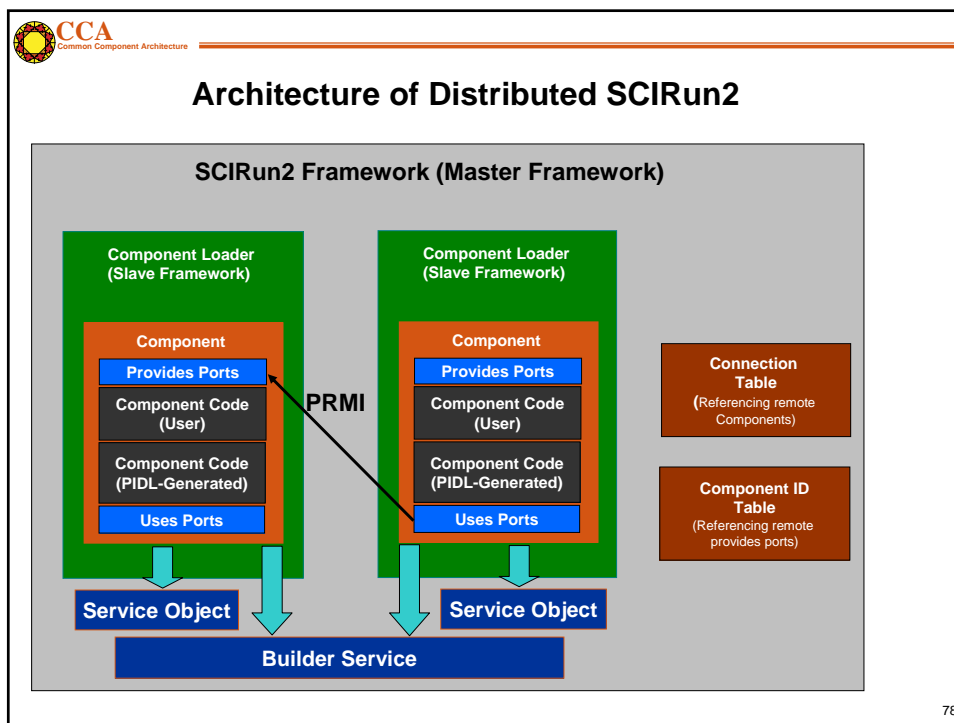
- SCIRun2
  - University of Utah
- LegionCCA
  - Binghamton University - State University of New York (SUNY)
- XCAT (Java and C++)
  - Indiana University and Binghamton University
- DCA
  - Indiana University
  - A research framework for MXN
- Frameworks address the design questions in different ways
  - Each has a different set of capabilities
  - Specialized for different kinds of applications


CCA  
Common Component Architecture

## SCIRun2

- Communication
  - C++ RMI that uses an in-house SIDL compiler
  - Co-location optimization
- Remote creation of distributed components
  - A **slave framework** resides in each remote address space
  - Uses **ssh** to start the slave framework
  - CCA BuilderService communicates with **master framework** which coordinates **slave frameworks**
- Support for distributed and parallel components
  - Can launch MPI-parallel components
    - Components interact via **Parallel Remote Method Invocation**
    - Each MPI process may contain multiple threads

77

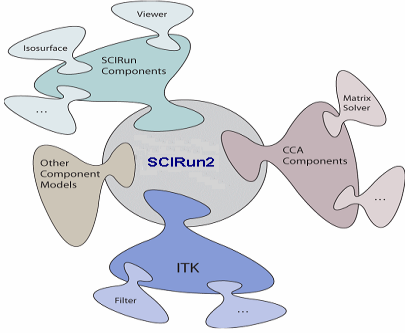


 **CCA**  
Common Component Architecture

---


## SCIRun2 Meta-Component Model

- In the same way that components plug into a CCA framework, component models (such as CCA) plug into SCIRun2
- Allows components of several different families to be used together
- Currently supports: CCA (Babel), SCIRun Dataflow, Visualization Toolkit (Vtk); others coming...
- Bridging between components of different models is semi-automatic; current research is defining a more automatic form of bridging



Research Area

79

 **CCA**  
Common Component Architecture

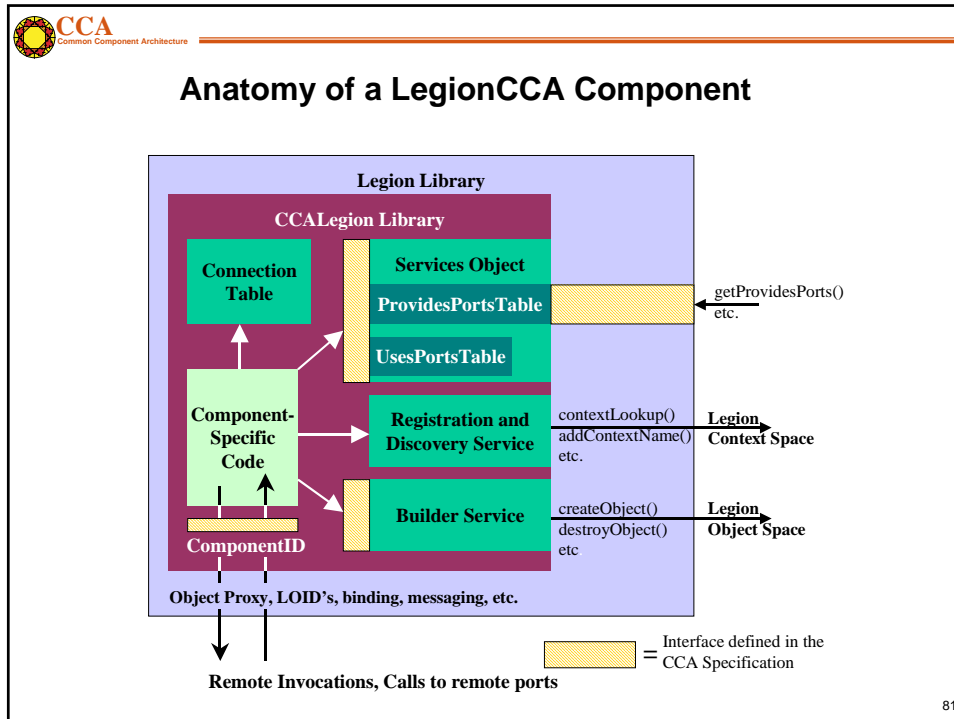
---

## LegionCCA


- Legion is a collection of software services for the Grid
  - Provides illusion of a virtual machine for geographically-distributed resources
- LegionCCA: models CCA components as Legion objects
- Component Communication
  - Uses Legion's built-in RPC mechanisms, based on Unix sockets
- ComponentID: based on Legion LOID
  - LOID: globally unique object id

80



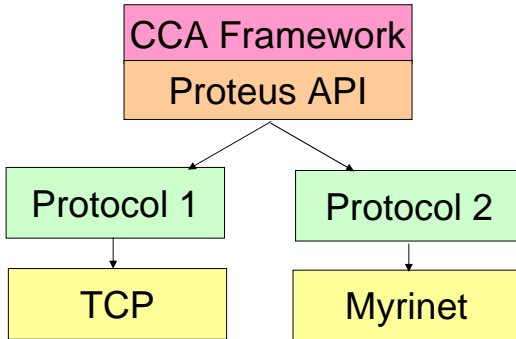


- ## XCAT
- Based on Web Services Standards
    - Remote reference format is WSDL
    - Remote Communication is based on XSOAP
      - An implementation of the [SOAP](#) protocol from Indiana Univ.
  - Remote creation of distributed components
    - Creation can currently be done via [GRAM](#) or [SSH](#)
      - GRAM: Grid Resource Allocation and Management
  - XCAT-Java
    - Consistent with standards in Grid Web Services
  - XCAT-C++
    - Uses [Proteus](#) for high performance remote communication
      - Proteus: multi-protocol library for messaging and RMI
      - Currently has two protocols: binary and SOAP
- 82

 **CCA**  
Common Component Architecture

## Proteus: Multi-Protocol Library


- One protocol does not suit all applications
- Proteus provides single-protocol abstraction to components
  - Allows users to dynamically switch between protocols
    - Example: Protocol 1 & Protocol 2, in the picture
  - Facilitates use of specialized implementations of [serialization](#) and [deserialization](#)



```
graph TD; CCA[CCA Framework] --- API[Proteus API]; API --- P1[Protocol 1]; API --- P2[Protocol 2]; P1 --- TCP[TCP]; P2 --- Myr[Myrinet];
```

The diagram illustrates the Proteus Multi-Protocol Library architecture. At the top is the CCA Framework, which contains the Proteus API. The Proteus API is designed to abstract multiple protocols. It branches into Protocol 1 and Protocol 2. Protocol 1 is implemented using TCP, and Protocol 2 is implemented using Myrinet. This structure allows users to dynamically switch between different network protocols through a single API.

83


 **CCA**  
Common Component Architecture

## Babel RMI

**Research!**

- Allows Babel objects to be accessed through remote Babel stubs.
- Underlying RMI uses Proteus.
- Objects that can be transmitted (serializable) inherent from Serializable.
- Actual implementation of serialization functions is by users, since only they know what needs to be serialized.

84


 **CCA**  
Common Component Architecture

---

# CCA Applications

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

85


 **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


86

 **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
  - CCTSS's Data Object Interface effort

87

 **CCA**  
Common Component Architecture

## Laplace Equation

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

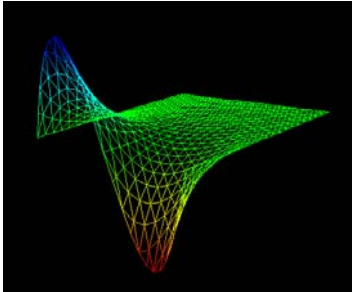
Physics Modules

Mesh

Discretization

Algebraic Solvers

Visualization



88

**CCA**  
Common Component Architecture

## 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

The screenshot shows the CCA software interface for a file named 'steady.bl'. On the left is a 'Palette' of components including CumulvsMxN, DistArrayDescriptorFactory, DynamicDriver, ESiExtension, ESiFactory\_Petra, FEMDiscretization, FileLogger, IntegratorLSODE, LaplaceModel, LinearSolver\_Petsc, Mesh, Model2Norm, ModelWeights, SteadyStateDriver, and VizMatlabFile. The main 'Arena' displays a diagram where the 'driver' component is circled in yellow. Red lines connect the driver to other components like 'mesh', 'disc', and 'solver'.

89

**CCA**  
Common Component Architecture

## 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

The screenshot shows the CCA software interface for a file named 'steady.bl'. On the left is a 'Palette' of components including CumulvsMxN, DistArrayDescriptorFactory, DynamicDriver, ESiExtension, ESiFactory\_Petra, FEMDiscretization, FileLogger, IntegratorLSODE, LaplaceModel, LinearSolver\_Petsc, Mesh, Model2Norm, ModelWeights, SteadyStateDriver, and VizMatlabFile. The main 'Arena' displays a diagram where the 'mesh' component is circled in yellow. Red lines connect the mesh to other components like 'driver', 'disc', and 'solver'.

90

**CCA**  
Common Component Architecture

## 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

Common Component Architecture: steady.bid

File CCA Info

Actions Run Remove Remove All Open... Save Save As...

Palette Arena

CumulvsMxN  
DistArrayDescriptorFactory  
DynamicDriver  
ESiExtension  
ESiFactory\_Petra  
FEMDiscretization  
FileLogger  
IntegratorLSODE  
LaplaceModel  
LinearSolver\_Petsc  
Mesh  
Model2Norm  
ModelWeights  
SteadyStateDriver  
VizMatlabFile

91

**CCA**  
Common Component Architecture

## 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

Common Component Architecture: steady.bid

File CCA Info

Actions Run Remove Remove All Open... Save Save As...

Palette Arena

CumulvsMxN  
DistArrayDescriptorFactory  
DynamicDriver  
ESiExtension  
ESiFactory\_Petra  
FEMDiscretization  
FileLogger  
IntegratorLSODE  
LaplaceModel  
LinearSolver\_Petsc  
Mesh  
Model2Norm  
ModelWeights  
SteadyStateDriver  
VizMatlabFile

92

**CCA**  
Common Component Architecture

## 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  $\phi$  on the unstructured triangular mesh
  - Assumes user data is attached to mesh vertex entities

Common Component Architecture: steady.bld

File CCA Info

Actions Run Remove Remove All Open... Save Save As...

Palette Arena

CumulusMxN  
DistArrayDescriptorFactory  
DynamicDriver  
Extension  
Factory\_Petra  
IDiscretization  
Logger  
egratorLSODE  
IfaceModel  
earSolver\_Petsc  
sh  
Jel2Norm  
JelWeights  
ndyStateDriver  
MatlabFile

cProps Mesh  
gc Disc  
CONFIG Solver  
Params  
driver

cProps Mesh  
mesh

cProps Mesh  
Disc Solver  
CONFIG Params  
disc

cProps Solver  
solver

cProps Mesh  
Vizfile  
vizfile

93

**CCA**  
Common Component Architecture

## Time-Dependent Heat Equation

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

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

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

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

Time Evolution

Physics Modules

Mesh

Discretization

Algebraic Solvers

Distributed Arrays

Data Redistribution

Visualization

94



## 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

95

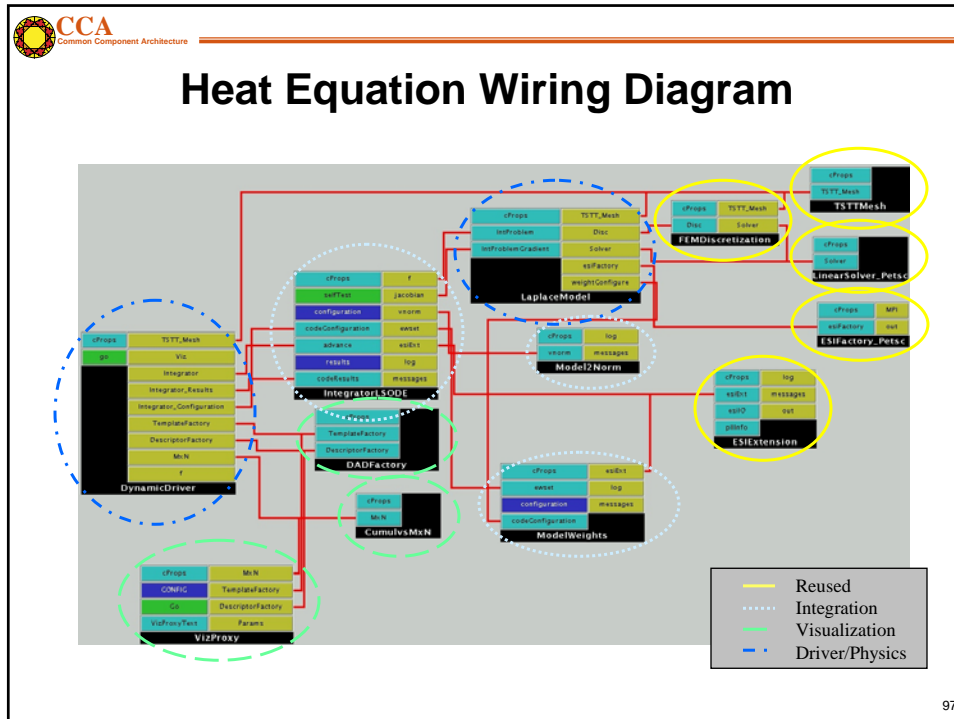


## ... and some things stay the same

- 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

96






## 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

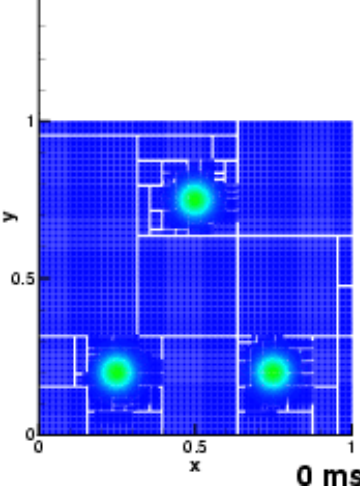
98

 **CCA**  
Common Component Architecture


## Nonlinear Reaction-Diffusion Equation

Temperature (K)

- Flame Approximation
  - H<sub>2</sub>-Air mixture; ignition via 3 hot-spots
  - 9-species, 19 reactions, stiff chemistry
- Governing equation
 
$$\frac{\partial Y_i}{\partial t} = \nabla \cdot \alpha \nabla Y_i + \dot{w}_i$$
- Domain
  - 1cm X 1cm domain
  - 100x100 coarse mesh
  - finest mesh = 12.5 micron.
- Timescales
  - O(10ns) to O(10 microseconds)



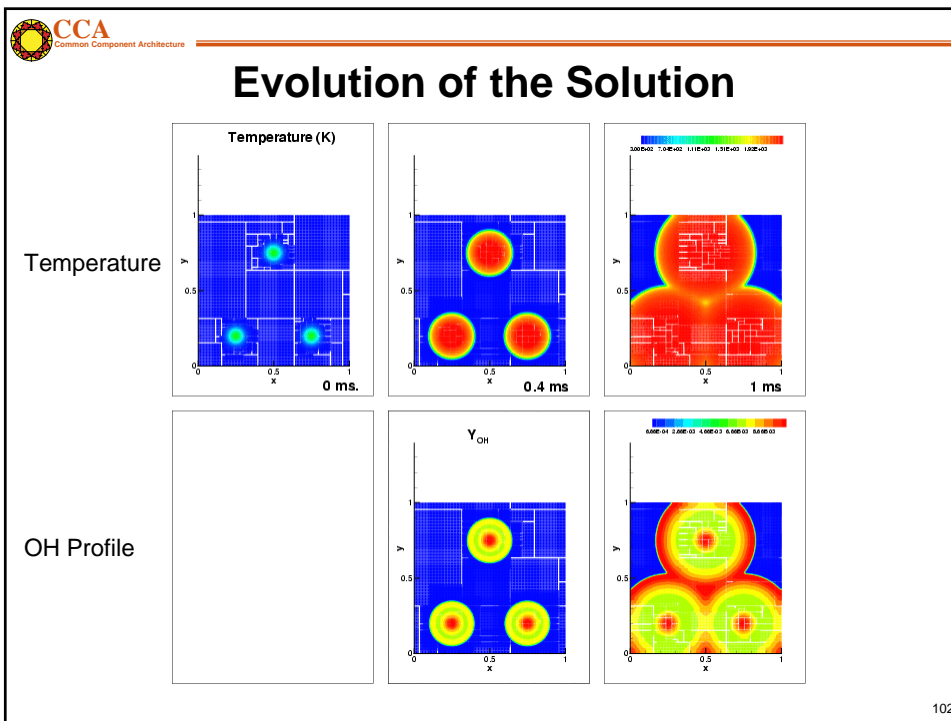
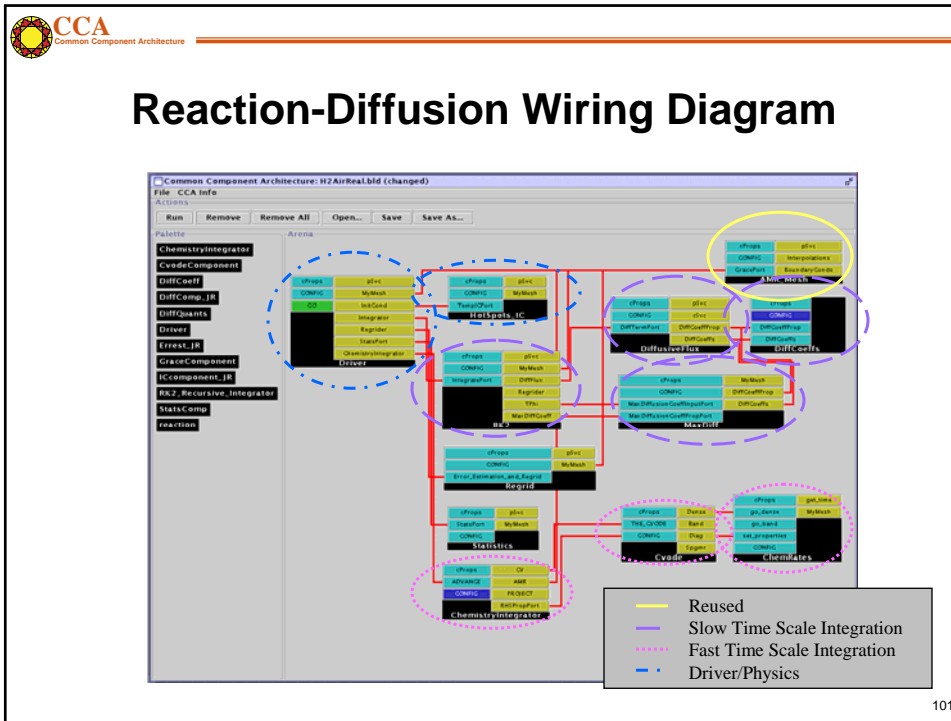
**0 ms.** 99

 **CCA**  
Common Component Architecture

## Numerical Solution

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

100



**CCA**  
Common Component Architecture

## The need for AMR

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

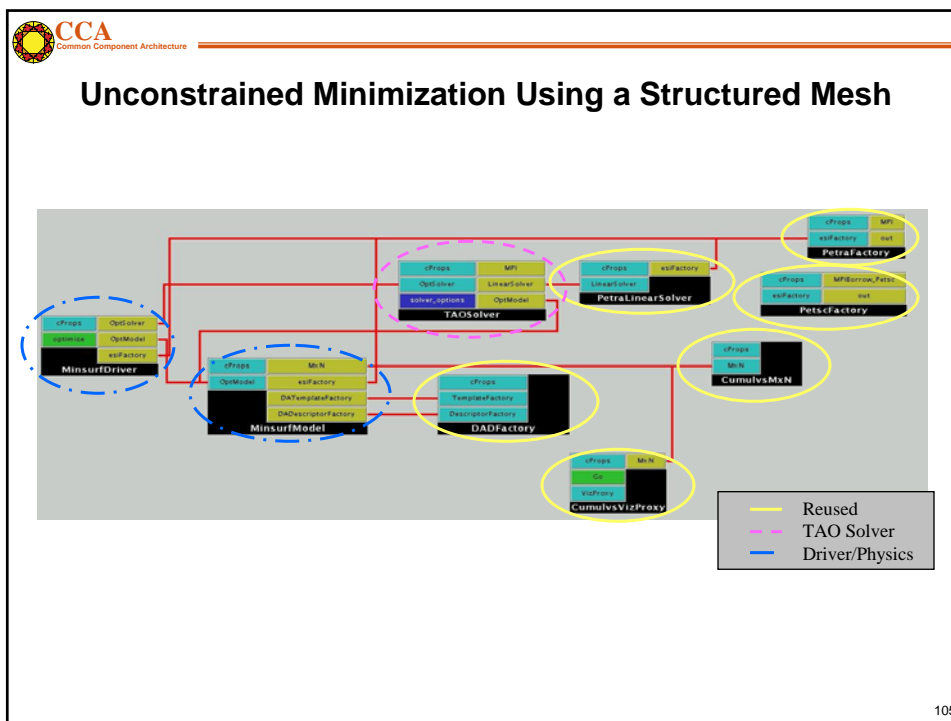
103

**CCA**  
Common Component Architecture

## 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
 
$$\min f(x), \text{ where } f: \mathbb{R} \rightarrow \mathbb{R}$$
- Solve using optimization components based on TAO (ANL)

104

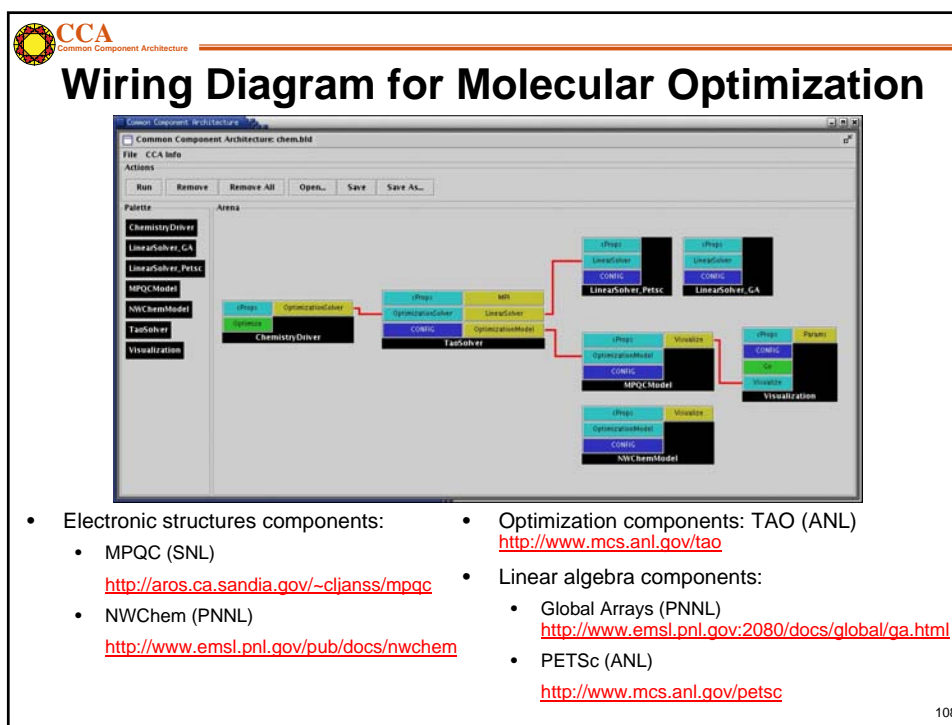
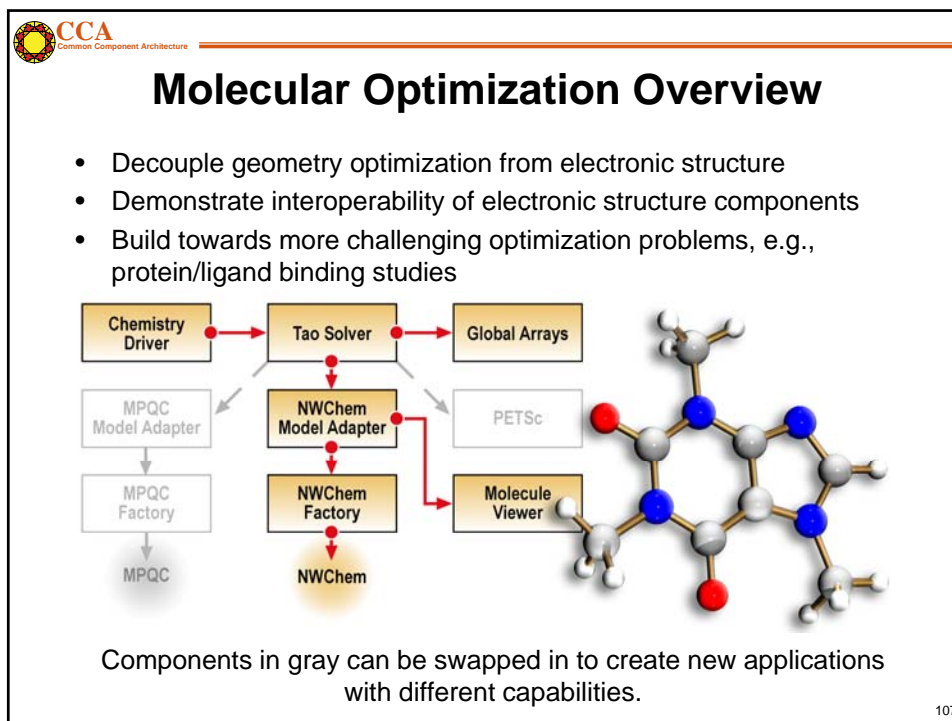



CCA  
Common Component Architecture

## 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

106






CCA  
Common Component Architecture

## 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**

109




CCA  
Common Component Architecture

## 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.

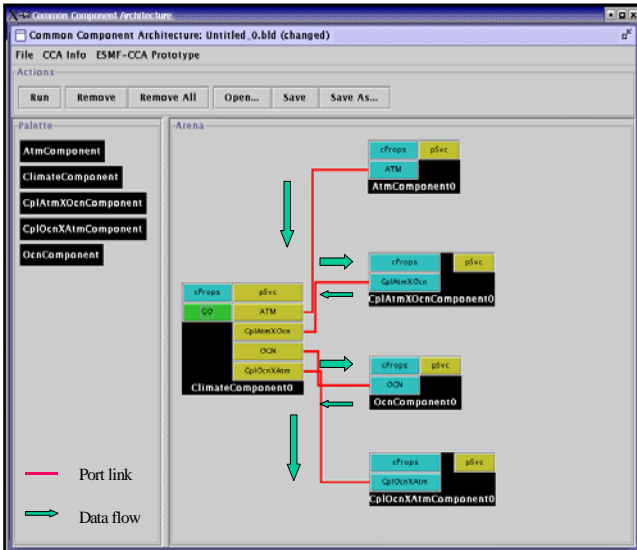
110




**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

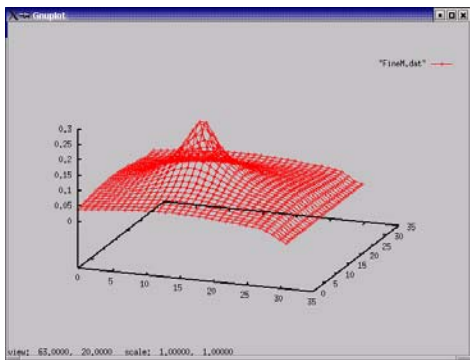


111



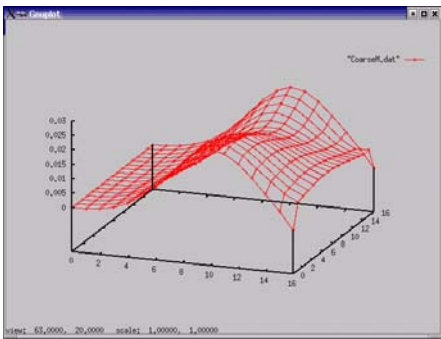
**CCA**  
Common Component Architecture

## Simulation Results




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

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



112

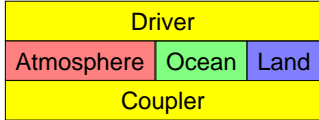


 **CCA**  
Common Component Architecture


---

## 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)



113


 **CCA**  
Common Component Architecture

---

## Overview

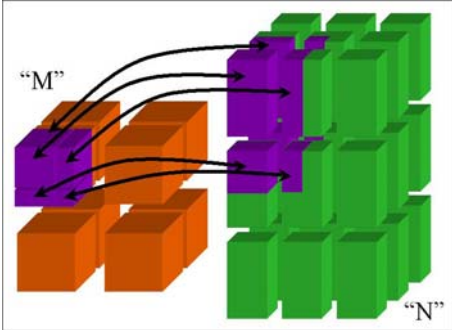
- Examples (scientific) of increasing complexity
  - Laplace equation
  - Time-dependent heat equation
  - Nonlinear reaction-diffusion system
  - Quantum chemistry
  - Climate simulation
- • Tools
  - MxN parallel data redistribution
  - Performance measurement, modeling and scalability studies
- Community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

114


 **CCA**  
Common Component Architecture

## “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”
    - $M \neq N$  ~ 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




115

 **CCA**  
Common Component Architecture


## 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/>

116



**CCA**  
 Common Component Architecture

## CCA Delivers Performance

**Local**

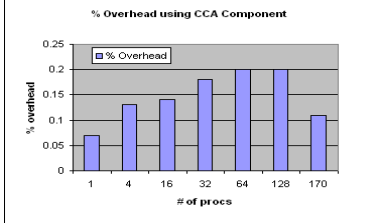
- No CCA overhead **within** components
- Small overhead **between** components
- Small overhead for **language interoperability**
- Be aware of costs & design with them in mind
  - Small costs, easily amortized

**Parallel**

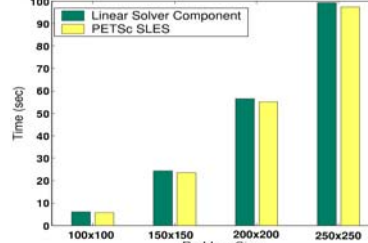
- No CCA overhead on **parallel computing**
- Use **your favorite** parallel programming model
- Supports SPMD and MPMD approaches

**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

117



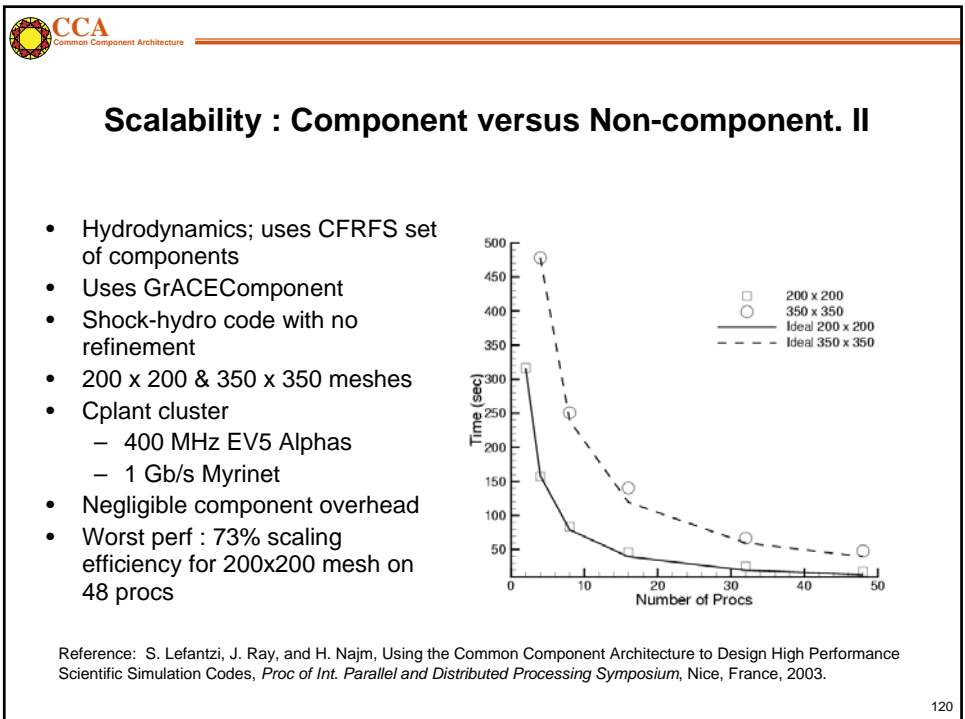
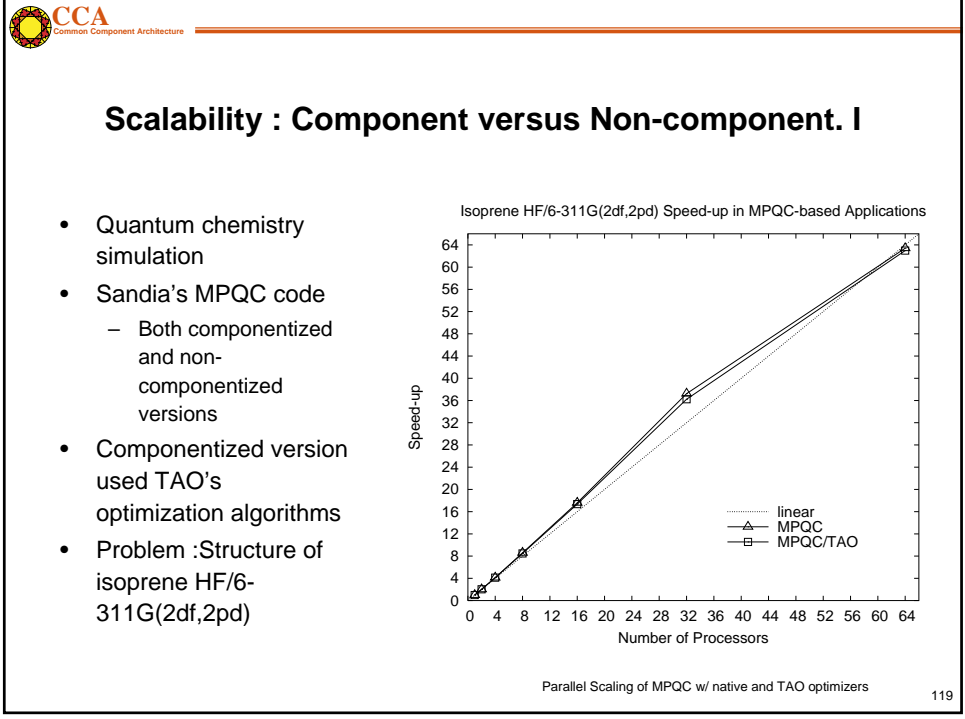
**CCA**  
 Common Component Architecture

## 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

118



**CCA**  
Common Component Architecture

## 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

121

**CCA**  
Common Component Architecture

## “Integrated” Performance Measurement Capability

**Measurement infrastructure:**

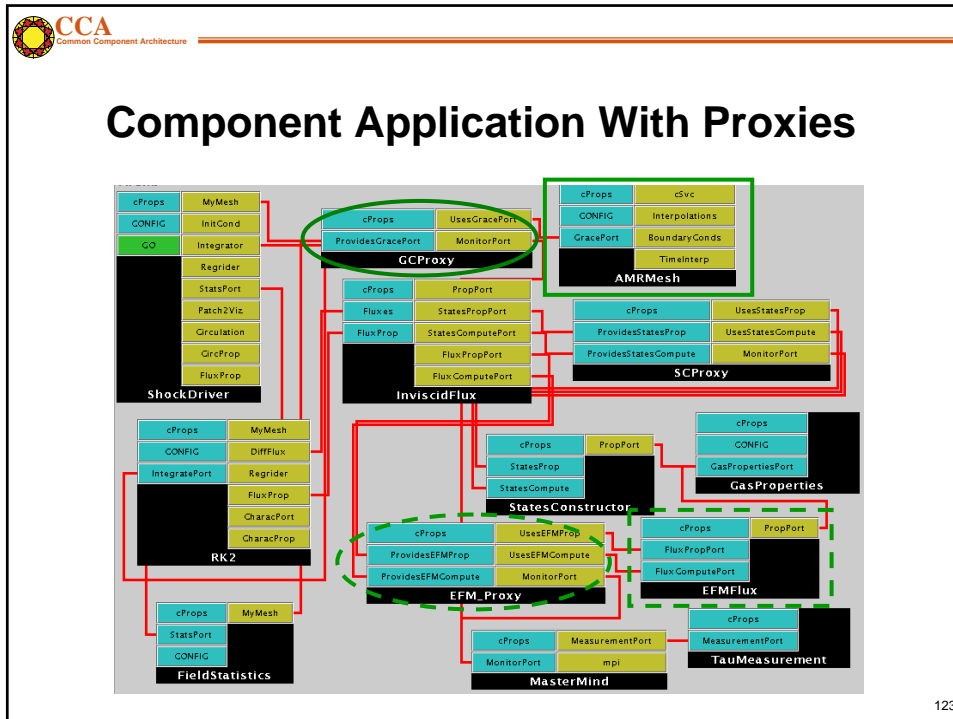
- **Proxy**
  - Notifies MasterMind of all method invocations of a given component, along with performance dependent inputs
  - Generated automatically using PDT
- **MasterMind**
  - Collects and stores all measurement data
- **TAU**
  - Makes all performance measurements

The diagram illustrates the integration of performance measurement infrastructure. It is divided into two parts: 'Before' and 'After'.

**Before:** A box labeled 'Component1' has a blue vertical bar on its right side. An arrow points from this bar to a blue vertical bar on the left side of a box labeled 'Component2'.

**After:** The 'Component1' box now has a blue vertical bar on its right side that connects to a blue vertical bar on the left side of a box labeled 'Proxy for Component'. The 'Proxy for Component' box has a blue vertical bar on its right side that connects to a blue vertical bar on the left side of the 'Component2' box. Below the 'Proxy for Component' box is a box labeled 'MasterMind'. A red vertical bar on the right side of the 'Proxy for Component' box connects to a red vertical bar on the left side of the 'MasterMind' box. Below the 'MasterMind' box is a box labeled 'TAU'. A green vertical bar on the right side of the 'MasterMind' box connects to a green vertical bar on the left side of the 'TAU' box.

122



- ## 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

**CCA**  
Common Component Architecture

**TSTT**

## 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](http://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

125


**CCA**  
Common Component Architecture

## Proliferations of interfaces – the $N^2$ problem

Current Situation

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

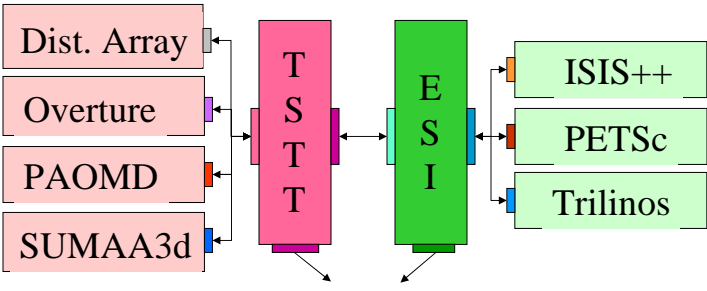
126

 **CCA**  
Common Component Architecture


## 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



127


 **CCA**  
Common Component Architecture

## 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

128






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)

129



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

130




**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](http://www.cca-forum.org/tutorials/tutorial-wg@cca-forum.org)

131



**CCA**  
Common Component Architecture

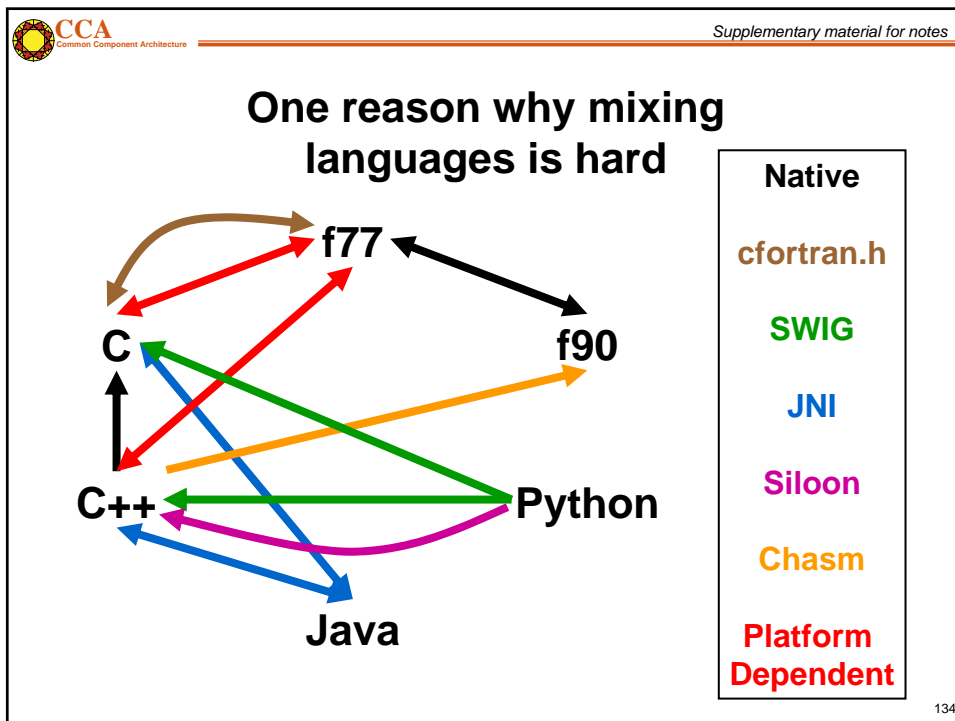
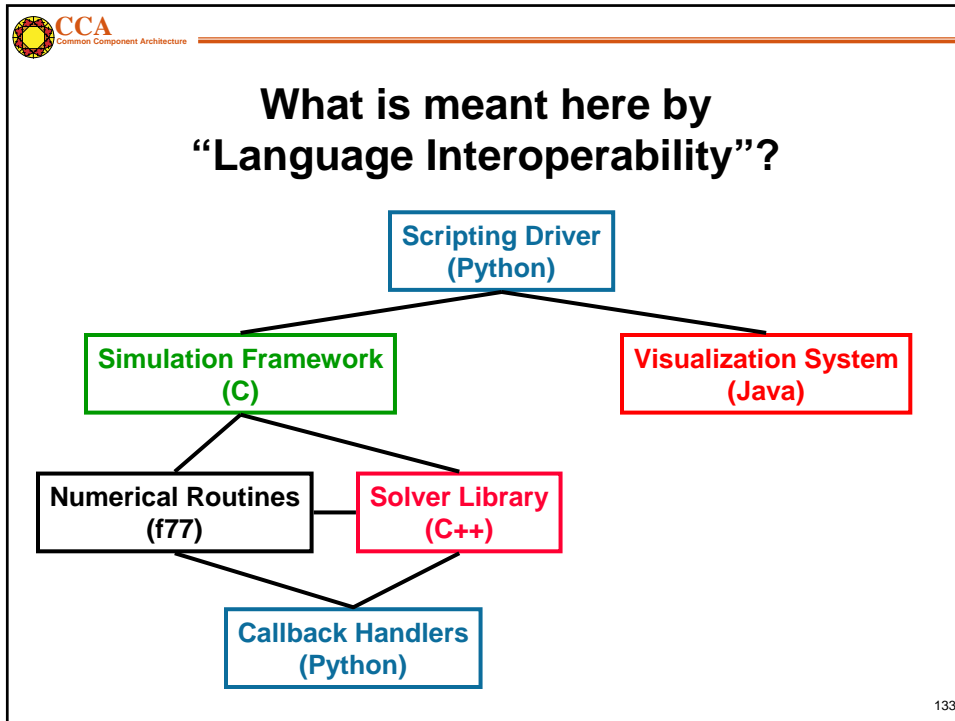
---

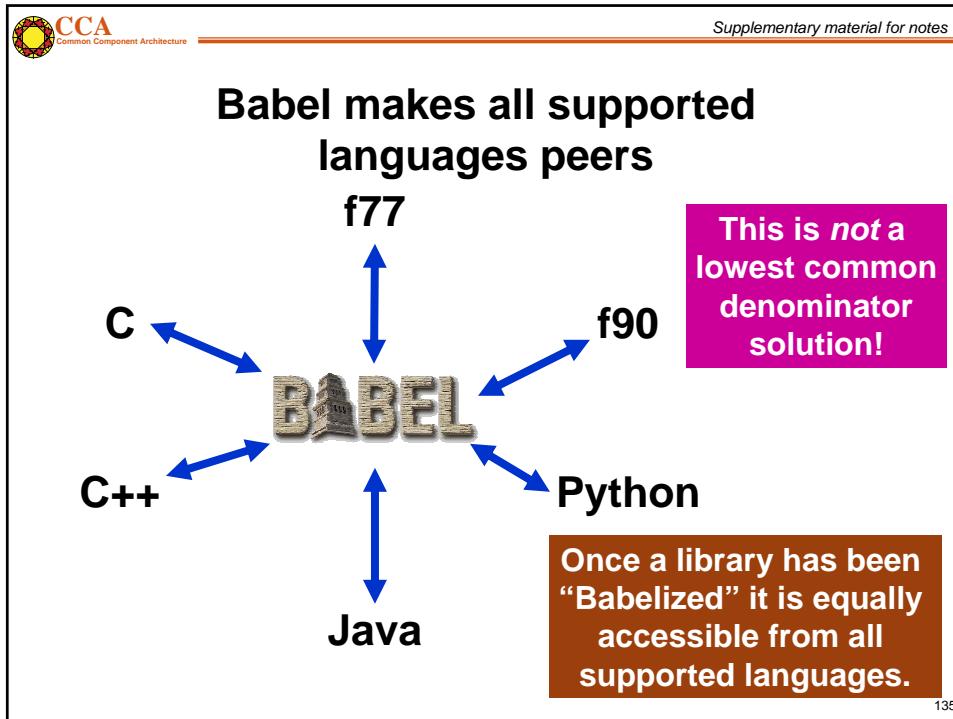
## Goal of This Module


**Legacy codes → Babelized CCA Components**

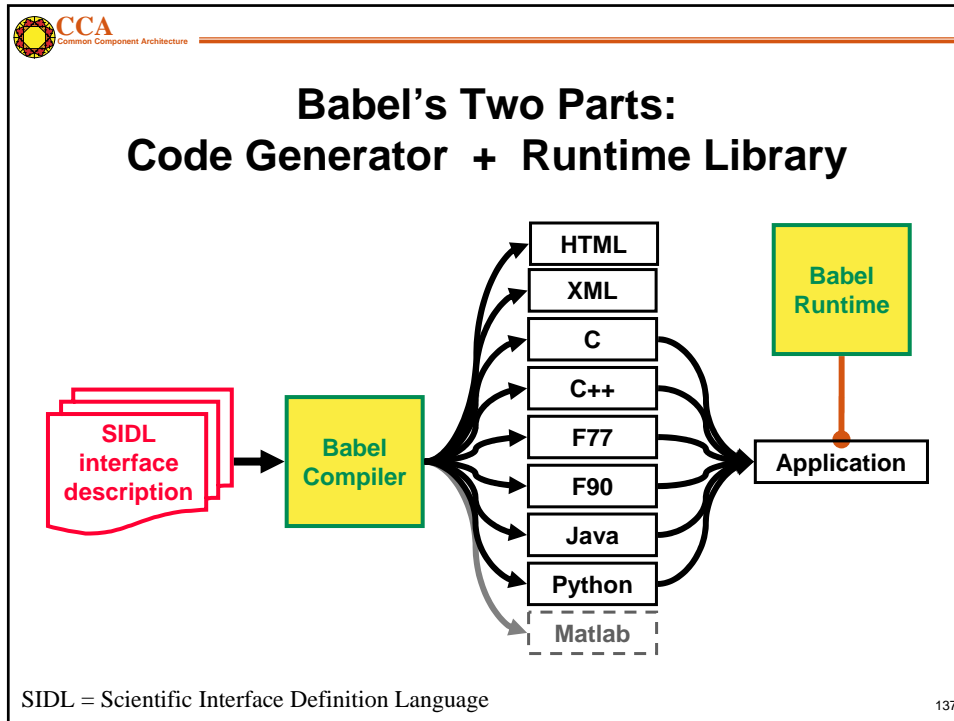
- Introduction
- Babel
  - “Hello World” example
  - Legacy Code
- Babelizing CCA components

132





- CCA Common Component Architecture
- ### Babel Module's Outline
- Introduction
  -  Babel
    - “Hello World” example
    - Legacy Code
  - Babelizing CCA components
- 136




**CCA**  
Common Component Architecture

### greetings.sidl: A Sample SIDL File

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

138



CCA  
Common Component Architecture


Supplementary material for notes

## SIDL Grammar (1/3): Packages and Versions

- Packages can be nested

```
package foo version 0.1 { package bar { ... } }
```
- Versioned Packages
  - Defined as packages with *explicit version number* OR packages enclosed by a versioned package
  - Reentrant by default, but can be declared final
  - May contain interfaces, classes, or enums
- Unversioned Packages
  - Can only enclose more packages, not types
  - *Must* be re-entrant; *Cannot* be declared final

139



CCA  
Common Component Architecture

Supplementary material for notes

## SIDL Grammar (2/3): Classes & Interfaces

- SIDL has 3 user-defined objects
  - **Interfaces** – APIs only, no implementation
  - **Abstract Classes** – 1 or more methods unimplemented
  - **Concrete Classes** – All methods are implemented
- Inheritance (like Java/Objective C)
  - Interfaces may **extend** Interfaces
  - Classes **extend** no more than one Class
  - Classes can **implement** multiple Interfaces
- Only concrete classes can be instantiated

140

CCA  
Common Component Architecture

Supplementary material for notes

## SIDL Grammar (3/3): Methods and Arguments

- Methods are **public virtual** by default
  - **static** methods are not associated with an object instance
  - **final** methods can not be overridden
- Arguments have 3 parts
  - Mode: can be **in**, **out**, or **inout** (like CORBA, but semantically different than F90)
  - Type: one of (bool, char, int, long, float, double, fcomplex, dcomplex, array<Type,Dimension>, enum, interface, class )
  - Name

141

CCA  
Common Component Architecture

## Library Developer Does This...

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

142

CCA  
Common Component Architecture

## Adding the Implementation

```
namespace greetings {
class EnglishImpl {
private:
// DO-NOT-DELETE spl i cer. begi n(greeti ngs. Engl i sh. _i mpl )
string d_name;
// DO-NOT-DELETE spl i cer. end(greeti ngs. Engl i sh. _i mpl )

string
greeti ngs::Engl i sh_i mpl ::sayl t()
throw ()
{
// DO-NOT-DELETE spl i cer. begi n(greeti ngs. Engl i sh. sayl t)
string msg("Hello ");
return msg + d_name + "!";
// DO-NOT-DELETE spl i cer. end(greeti ngs. Engl i sh. sayl t)
}
}
```

143

CCA  
Common Component Architecture

## Library User Does This...

```

graph LR
    A[greetings.sidl] --> B[Babel Compiler]
    B --> C[F90 Stubs]
    B --> D[IOR Headers]
    C --> E[Application]
    D --> E
    E --- F[libgreetings.so]
    G[Babel Runtime] --> E
    G --- F
  
```

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

IOR = Intermediate Object Representation

144



CCA  
Common Component Architecture

## F90/Babel "Hello World" Application

```

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 )
  call deleteRef( obj )
  print *, msg
end program helloclient
    
```

These subroutines were specified in the SIDL.

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

145

CCA  
Common Component Architecture

## Babelizing Legacy Code

1. Write your SIDL file to define public methods
2. Generate server side in your native language using Babel
3. Edit Implementations to dispatch to your code  
*Do NOT need to modify the legacy library itself!*
4. Compile and link into library/DLL









146

CCA  
Common Component Architecture

Supplementary material for notes

## Known Projects Using Babel

(see [www.llnl.gov/CASC/components/gallery.html](http://www.llnl.gov/CASC/components/gallery.html) for more)

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.

147

CCA  
Common Component Architecture


Supplementary material for notes

## Investing in Babelization can *improve* the interface to the code.

*“When Babelizing LEOS [an equation of state library at LLNL], I completely ignored the legacy interface and wrote the SIDL the way I thought the interface should be. After running Babel to generate the code, I found all the hooks I needed to connect LEOS without changing any of it. Now I’ve got a clean, new, object-oriented python interface to legacy code. Babel is doing much more than just wrapping here.”*

-- Charlie Crabb, LLNL  
(conversation)

148




CCA  
Common Component Architecture

---

## Babel Module's Outline

- Introduction
- Babel
  - “Hello World” example
  - Legacy Code
- Babelizing CCA components

149



CCA  
Common Component Architecture

---

## How to write Babelized CCA Components

1. Define “Ports” in SIDL
2. Define “Components” in SIDL
  - Used to implement ports defined in step 1.*
3. Use Babel to generate the interoperability glue
4. Fill in the Implementations as needed

150



## 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 );
  }
}
```

151



## 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 { }
```

152

CCA Common Component Architecture Supplementary material for notes

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

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


153

CCA Common Component Architecture

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

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

154

 **CCA**  
Common Component Architecture


Supplementary material for notes

---

## Limitations of Babel's Approach to Language Interoperability

- Babel is a code generator
  - Do obscure tricks no one would do by hand
  - Don't go beyond published language standards
- Customized compilers / linkers / loaders beyond our scope
  - E.g. icc and gcc currently don't mix on Linux
  - E.g. No C++-style templates in SIDL. (Would require special linkers/loaders to generate code for template instantiation, like C++ does.)
- Babel makes language interoperability feasible, but not trivial
  - Build tools severely underpowered for portable multi-language codes

155

 **CCA**  
Common Component Architecture

---

## 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](mailto:components@llnl.gov)
- Mailing Lists: [majordomo@lists.llnl.gov](mailto:majordomo@lists.llnl.gov)
  - subscribe [babel-users](mailto:babel-users) [email address]
  - subscribe [babel-announce](mailto:babel-announce) [email address]
- Bug Tracking: <https://www.cca-forum.org/bugs/babel/>  
or email to [babel-bugs@cca-forum.org](mailto:babel-bugs@cca-forum.org)

156