An Overview of Components and the Common Component Architecture

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

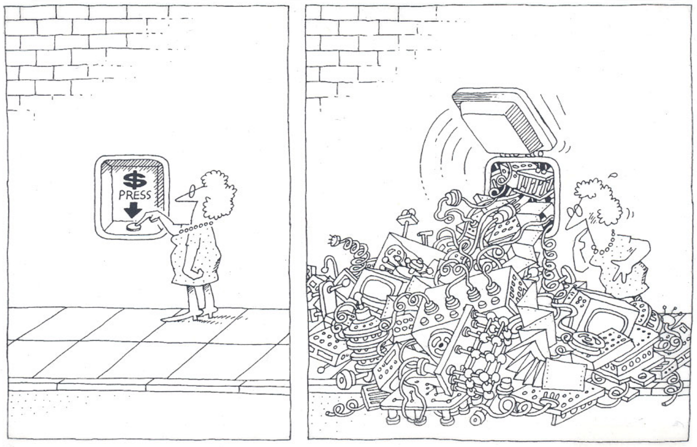
Outline

- **Why** do we need components?
- **What** are components?
- What are **CCA** components?

Why Components

- In “Components, The Movie”
  - Interoperability across multiple languages
  - Interoperability across multiple platforms
  - Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)
- Complexity

Why Components



The task of the software development team is to engineer the illusion of simplicity *[Booch]*.

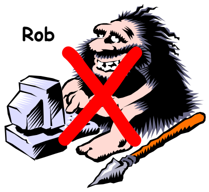
Software Complexity

- Software crisis
  - “Our failure to master the complexity of software results in projects that are late, over budget, and deficient in their stated requirements” *[Booch]*
- Can’t escape it
  - “The complexity of software is an essential property, not an accidental one” *[Brooks]*
- Help is on the way…
  - “A complex system that works is invariably found to have evolved from a simple system that worked… A complex system designed from scratch never works and cannot be patched up to make it work.” *[Gall]*
  - “Intracomponent linkages are generally stronger than intercomponent linkages” *[Simon]*
  - “Frequently, complexity takes the form of a hierarchy” *[Courtois]*

The Good the Bad and the Ugly

- An example of what can lead to a crisis in software:
- At least 41 different Fast Fourier Transform (FFT) libraries:
  - see, http://www.fftw.org/benchfft/doc/ffts.html
- Many (if not all) have different interfaces
  - different procedure names and different input and output parameters
- SUBROUTINE FOUR1(DATA, NN, ISIGN)
  - Replaces DATA by its discrete Fourier transform (if ISIGN is input as 1) or replaces DATA by NN times its inverse discrete Fourier transform (if ISIGN is input as -1). DATA is a complex array of length NN or, equivalently, a real array of length 2*NN. NN MUST be an integer power of 2 (this is not checked for!).

Components Promote Reuse



Hero programmer producing single-purpose, monolithic, tightly-coupled parallel codes

- Components promote software reuse
  - "The best software is code you don't have to write" *[Steve Jobs]*
- Reuse, through cost amortization increases software quality
  - thoroughly tested code
  - highly optimized code
  - improved support for multiple platforms
  - developer team specialization

What Are Components

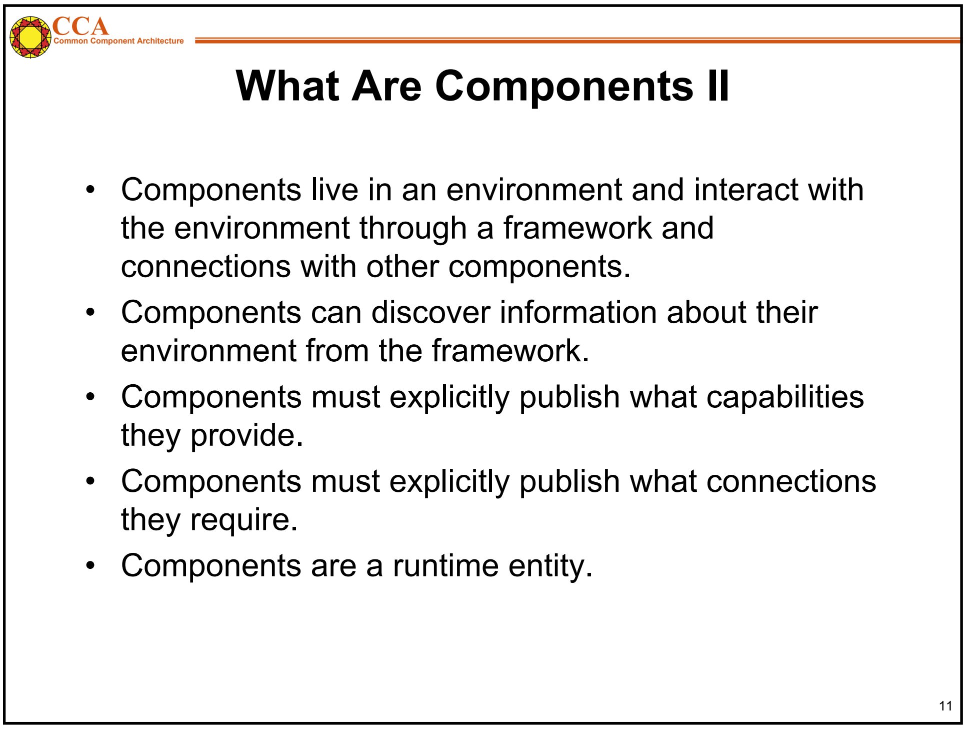
- **Why** do we need components?
- **What** are components?
- What are **CCA** components?

What Are Components *[Szyperski]*

- A component is a binary unit of independent deployment
  - well separated from other components
    - fences make good neighbors
  - can be deployed independently
- A component is a unit of third-party composition
  - is composable (even by physicists)
  - comes with clear specifications of what it requires and provides
  - interacts with its environment through well-defined interfaces
- A component has no persistent state
  - temporary state set only through well-defined interfaces
  - throw away that dependence on global data (common blocks)
- Similar to Java packages and Fortran 90 modules (with a little help)

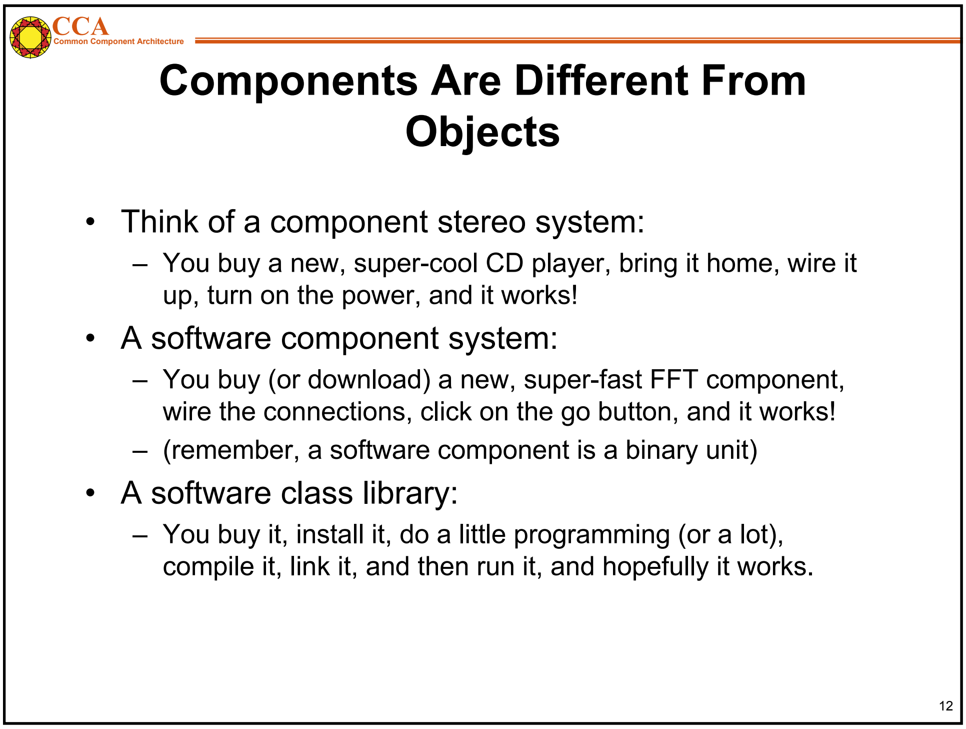
What Does This Mean

- Once again
  - A component is a binary unit of independent deployment
  - A component is a unit of third-party composition
  - A component has no persistent state
- So what does this mean
  - Components are “plug and play”
  - Components are reusable
  - Component applications are evolvable



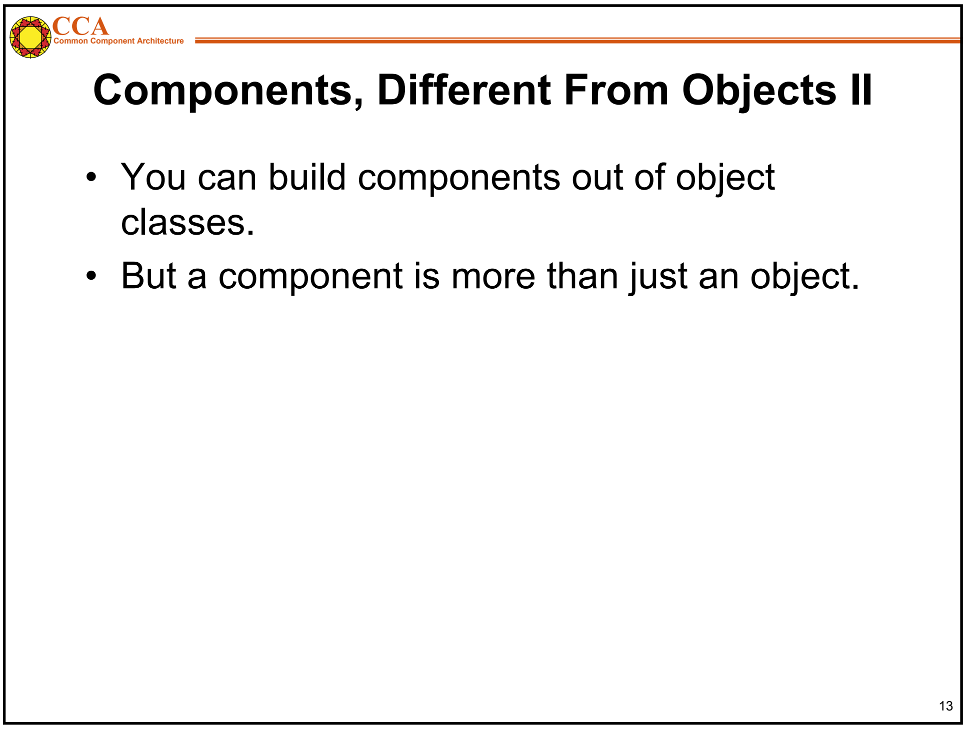
What Are Components II

- Components live in an environment and interact with the environment through a framework and connections with other components.
- Components can discover information about their environment from the framework.
- Components must explicitly publish what capabilities they provide.
- Components must explicitly publish what connections they require.
- Components are a runtime entity.



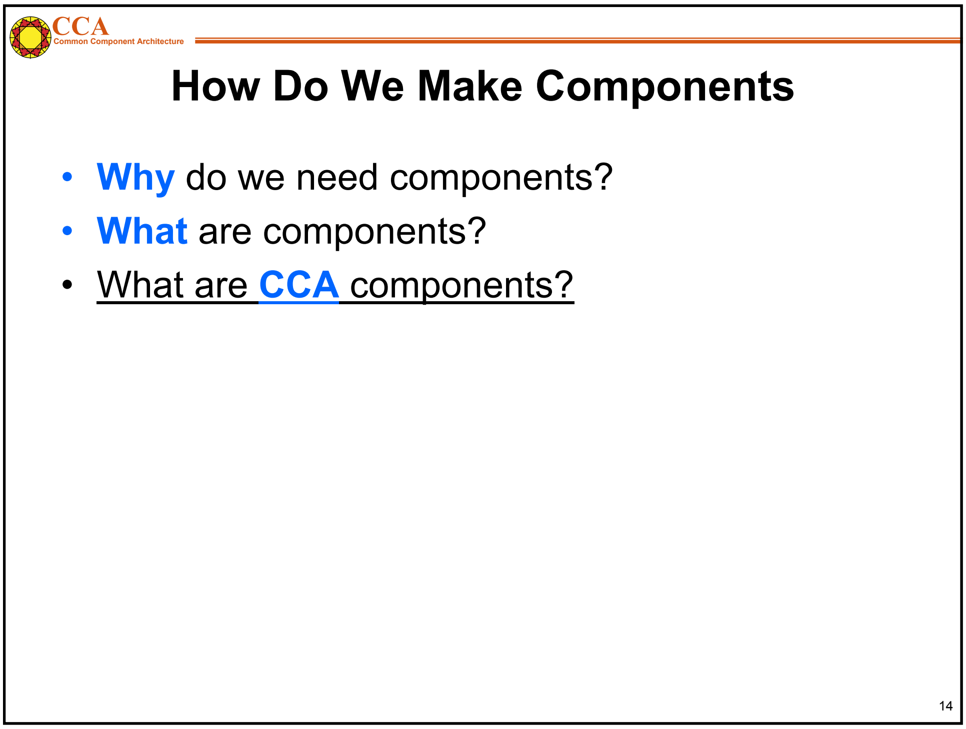
Components Are Different From Objects

- Think of a component stereo system:
  - You buy a new, super-cool CD player, bring it home, wire it up, turn on the power, and it works!
- A software component system:
  - You buy (or download) a new, super-fast FFT component, wire the connections, click on the go button, and it works!
  - (remember, a software component is a binary unit)
- A software class library:
  - You buy it, install it, do a little programming (or a lot), compile it, link it, and then run it, and hopefully it works.



Components, Different From Objects II

- You can build components out of object classes.
- But a component is more than just an object.



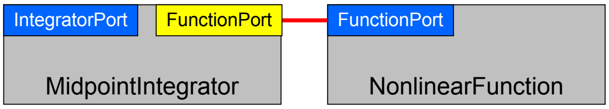
How Do We Make Components

- **Why** do we need components?
- **What** are components?
- <u>What are **CCA** components?</u>

Features of the Common Component Architecture

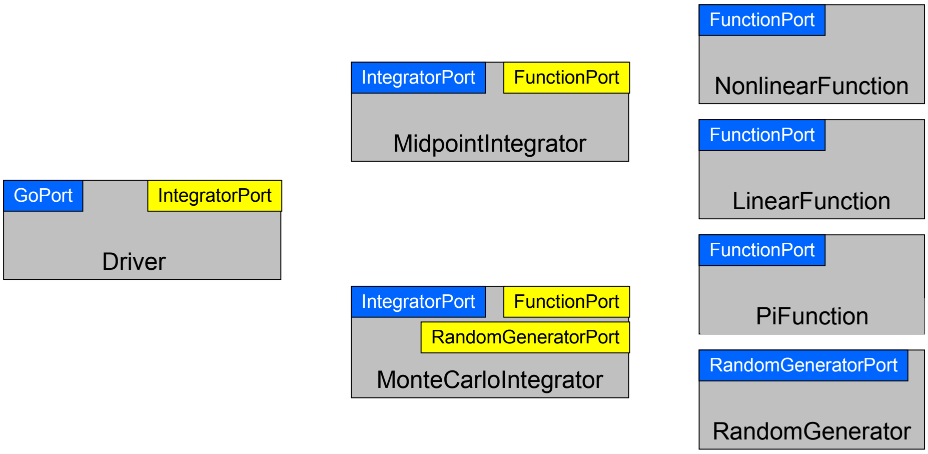
- A component model specifically designed for high-performance computing
  - Support HPC languages *(Babel)*
  - Support parallel as well as distributed execution models
  - Minimize performance overhead
- Minimalist approach makes it easier to componentize existing software
- Component interactions are *not* merely dataflow
- Components are peers
  - No particular component assumes it is “in charge” of the others.
  - Allows the application developer to decide what is important.

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
- Components may *use* ports – call methods or subroutines in the port
- Links denote a caller/callee relationship, ***not dataflow!***
  - e.g., FunctionPort could contain: *evaluate(in Arg, out Result)*

Components and Ports in the Integrator Example



FunctionPort
NonlinearFunction

IntegratorPort
FunctionPort
MidpointIntegrator

FunctionPort
LinearFunction

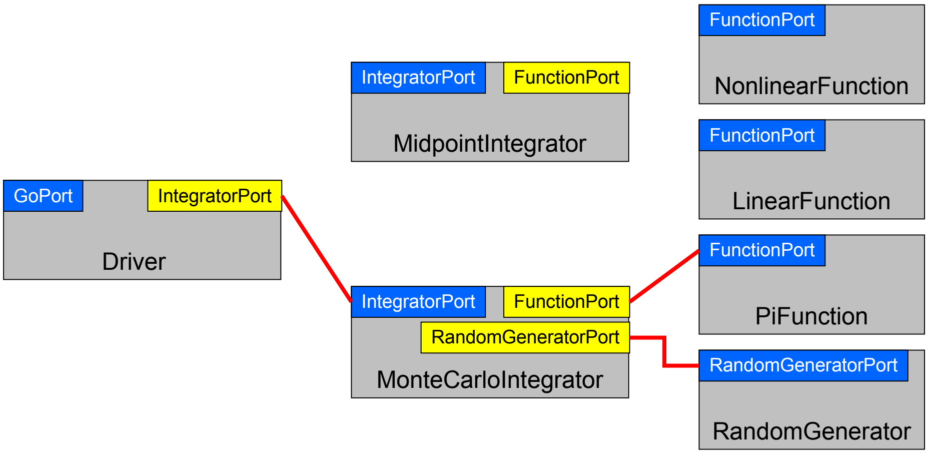
GoPort
IntegratorPort
Driver

FunctionPort
PiFunction

IntegratorPort
FunctionPort
RandomGeneratorPort
MonteCarloIntegrator

RandomGeneratorPort
RandomGenerator

An Application Built from the Example Components



FunctionPort
NonlinearFunction

IntegratorPort
FunctionPort
MidpointIntegrator

FunctionPort
LinearFunction

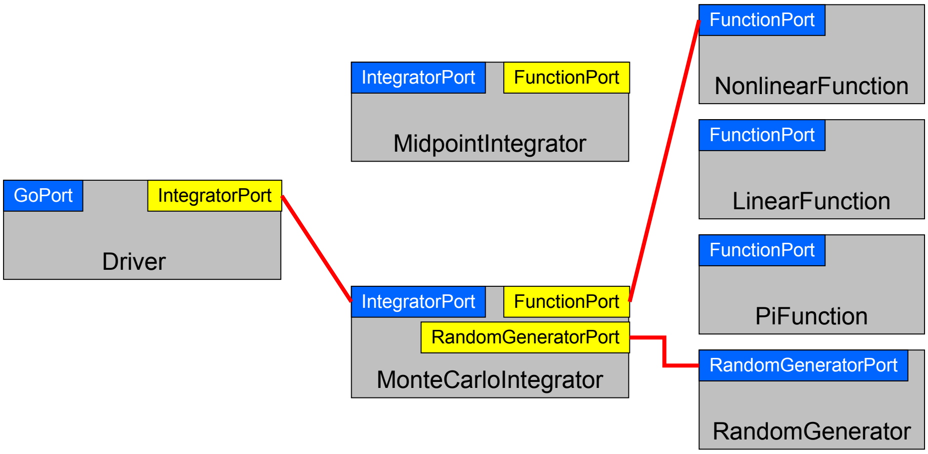
GoPort
IntegratorPort
Driver

FunctionPort
PiFunction

IntegratorPort
FunctionPort
RandomGeneratorPort
MonteCarloIntegrator

RandomGeneratorPort
RandomGenerator

Another Application…



GoPort
IntegratorPort
Driver

IntegratorPort
FunctionPort
MidpointIntegrator

IntegratorPort
FunctionPort
RandomGeneratorPort
MonteCarloIntegrator

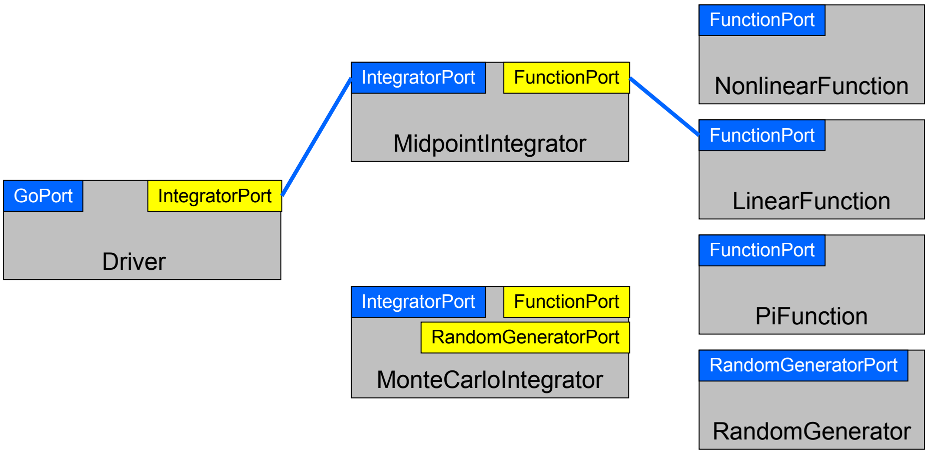
FunctionPort
NonlinearFunction

FunctionPort
LinearFunction

FunctionPort
PiFunction

RandomGeneratorPort
RandomGenerator

Application 3…



GoPort
IntegratorPort
Driver

IntegratorPort
FunctionPort
MidpointIntegrator

IntegratorPort
FunctionPort
RandomGeneratorPort
MonteCarloIntegrator

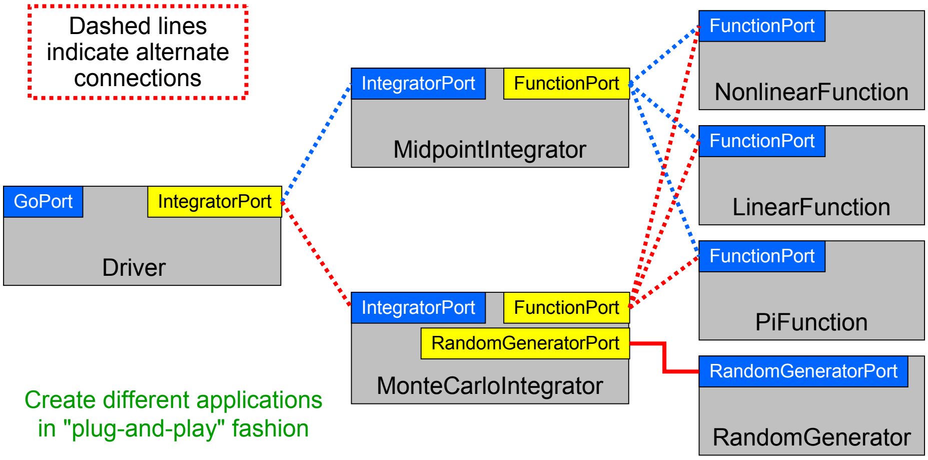
FunctionPort
NonlinearFunction

FunctionPort
LinearFunction

FunctionPort
PiFunction

RandomGeneratorPort
RandomGenerator

And Many More…



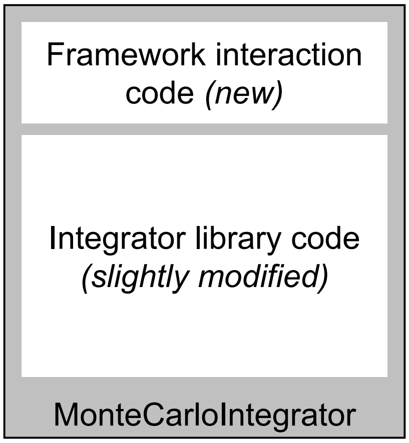
Create different applications in "plug-and-play" fashion

Ports, Interoperability, and Reuse

- Ports (interfaces) define how components interact
- Generality, quality, robustness of ports is up to designer/architect
  - "Any old" interface is easy to create, but…
  - Developing a robust domain "standard" interface requires thought, effort, and cooperation
- General "plug-and-play" interoperability of components requires multiple implementations conforming to the same interface
- Designing for interoperability and reuse requires "standard" interfaces
  - Typically domain-specific
  - "Standard" need not imply a formal process, may mean "widely used"

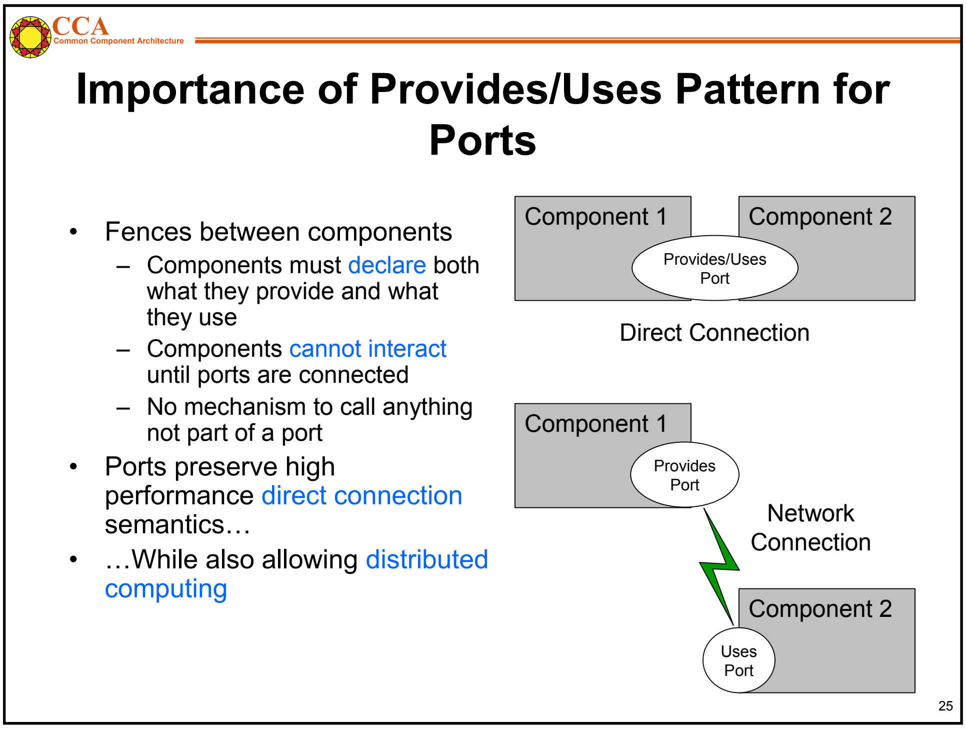
Components vs Libraries

- Component environments rigorously enforce interfaces
- Can have several versions of a component loaded into a single application
- Component needs add'l code to interact w/ framework
  - Constructor and destructor methods
  - Tell framework what ports it *uses* and *provides*
- Invoking methods on other components requires slight modification to "library" code


Framework interaction code *(new)*
Integrator library code *(slightly modified)*
MonteCarloIntegrator

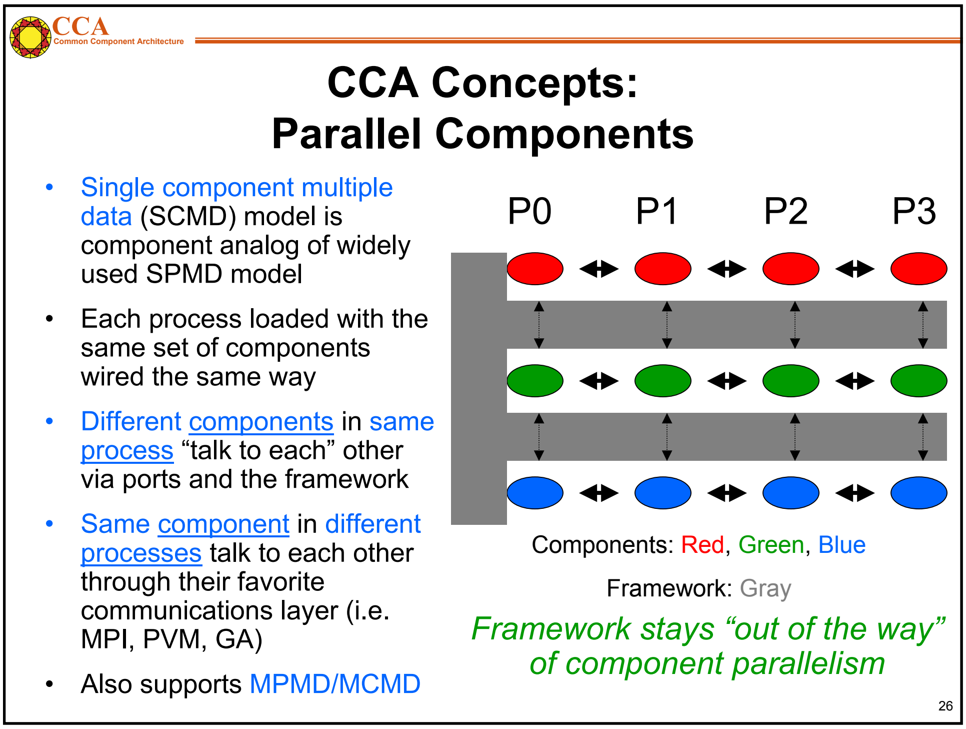
CCA Concepts: Frameworks

- The framework provides the means to "hold" components and compose them into applications
  - The framework is often application's "main" or "program"
- Frameworks allow exchange of ports among components without exposing implementation details
- Frameworks provide a small set of standard services to components
  - BuilderServices allow programs to compose CCA apps
- Frameworks may make themselves appear as components in order to connect to components in other frameworks
- *Currently:* specific frameworks support specific computing models (parallel, distributed, etc.). *Future:* full flexibility through integration or interoperation



Importance of Provides/Uses Pattern for Ports

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



CCA Concepts: Parallel Components

- 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)
- Also supports MPMD/MCMD

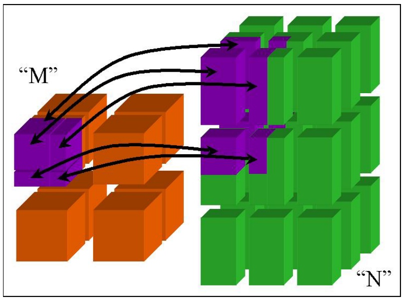
Components: Red, Green, Blue

Framework: Gray

Framework stays "out of the way" of component parallelism

CCA Concepts: MxN Parallel Data Redistribution

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

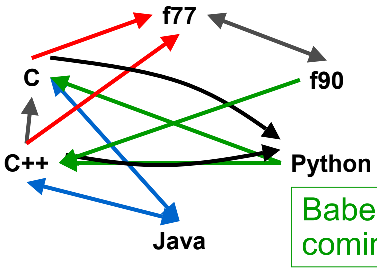


Research area -- tools under development

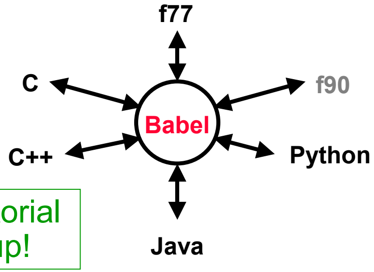
CCA Concepts: Language Interoperability

- 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



Babel tutorial coming up!



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
- MxN Parallel Data Redistribution
  - Model coupling, visualization, etc.
- Language Interoperability
  - Babel, Scientific Interface Definition Language (SIDL)