Transport Simulation with High Resolution RF Analysis Using the Common-Component Architecture

Lee A. Berry

US-Japan Workshop on

Integrated Modeling

PPPL

September 21-23, 2004

Outline

- Why components?
- A “proof” of principle” project to gain experience.
- Overview of the Common-Component Architecture (CCA).
- Summary of project status.
- Experience.

Acknowledgements I: 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** – 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!

Excerpts from CCA tutorials were used for much of this talk.

Acknowledgements II: Project Team

- Physics:
  - Fred Jaeger, Wayne Houlberg, Don Batchelor.
- Applied math—algorithms:
  - Ed D'Azevedo.
- Computer Science:
  - David Bernholdt, Wael Elwasif, James Kohl.

This is a two-year, ~ 2-py/y effort with the goal of exploring the advantages/disadvantages of the CCA for fusion simulation

Needs of large simulations

- High performance.
- Rapid development cycle.
- Language interoperability, ready use of legacy code.
- Multiple third-party libraries.
- Range of applications with common elements.
- Efficient implementation with large teams.

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 ***"Standard" Interfaces.***
- Goal: ***Interoperability*** between components developed by different expert teams across different institutions.
- Quarterly Meetings, Open membership….

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

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

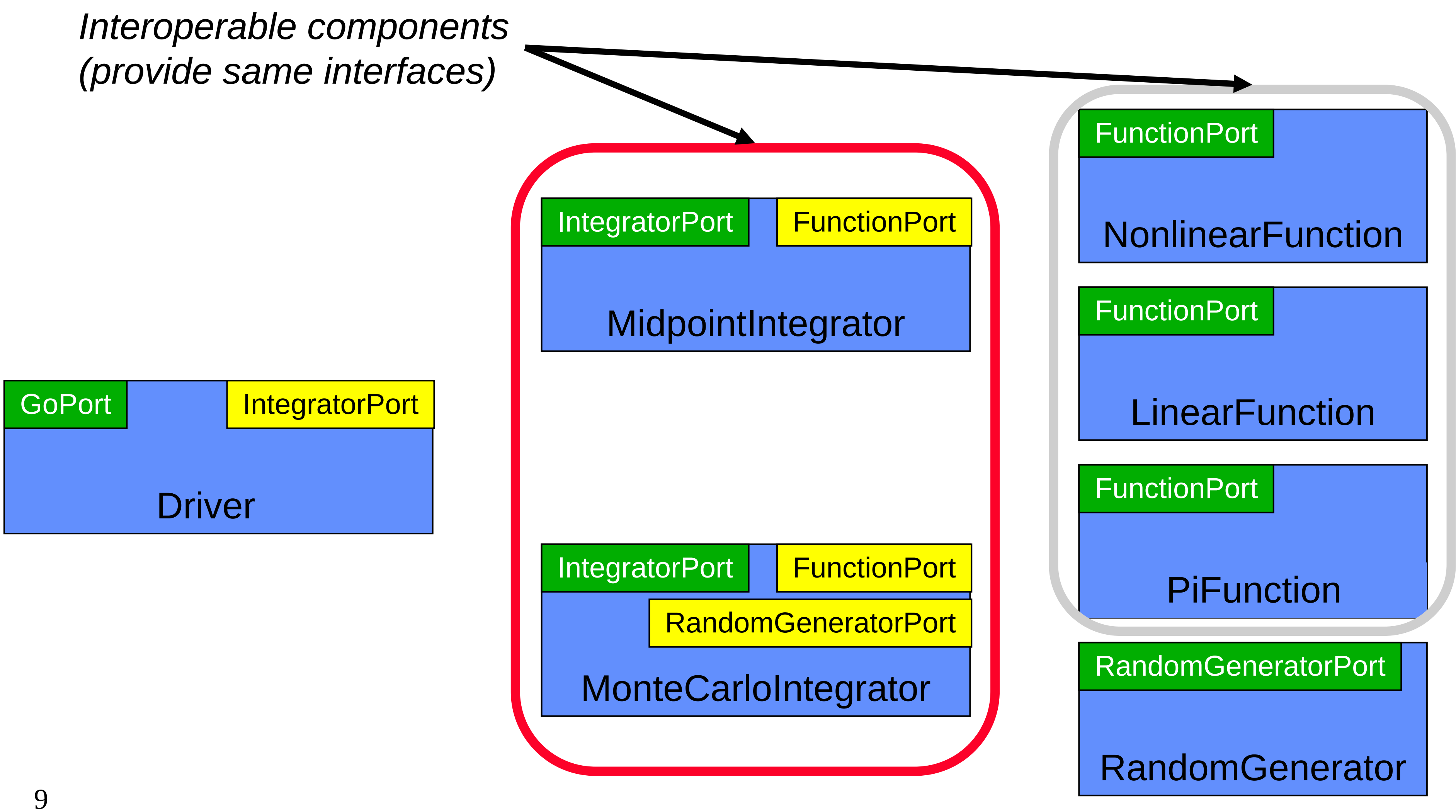
What *are* Components?

- **A unit of software deployment/reuse:**
  - Ideally, has functionality that someone else might be able to (re)use;
  - Can be developed independently of other components;
  - Has significant computational work to pay for overhead.
- **Interacts with the outside world *only* through well-defined interfaces:**
  - Implementation is opaque to the outside world;
  - Components *may* maintain state information;
  - But external access to state info must be through an interface.
  - File-based interactions can be recast using an “I/O component”.
- **Can be composed with other components:**
  - “Plug and play” model to build applications;
  - Composition based on interfaces.

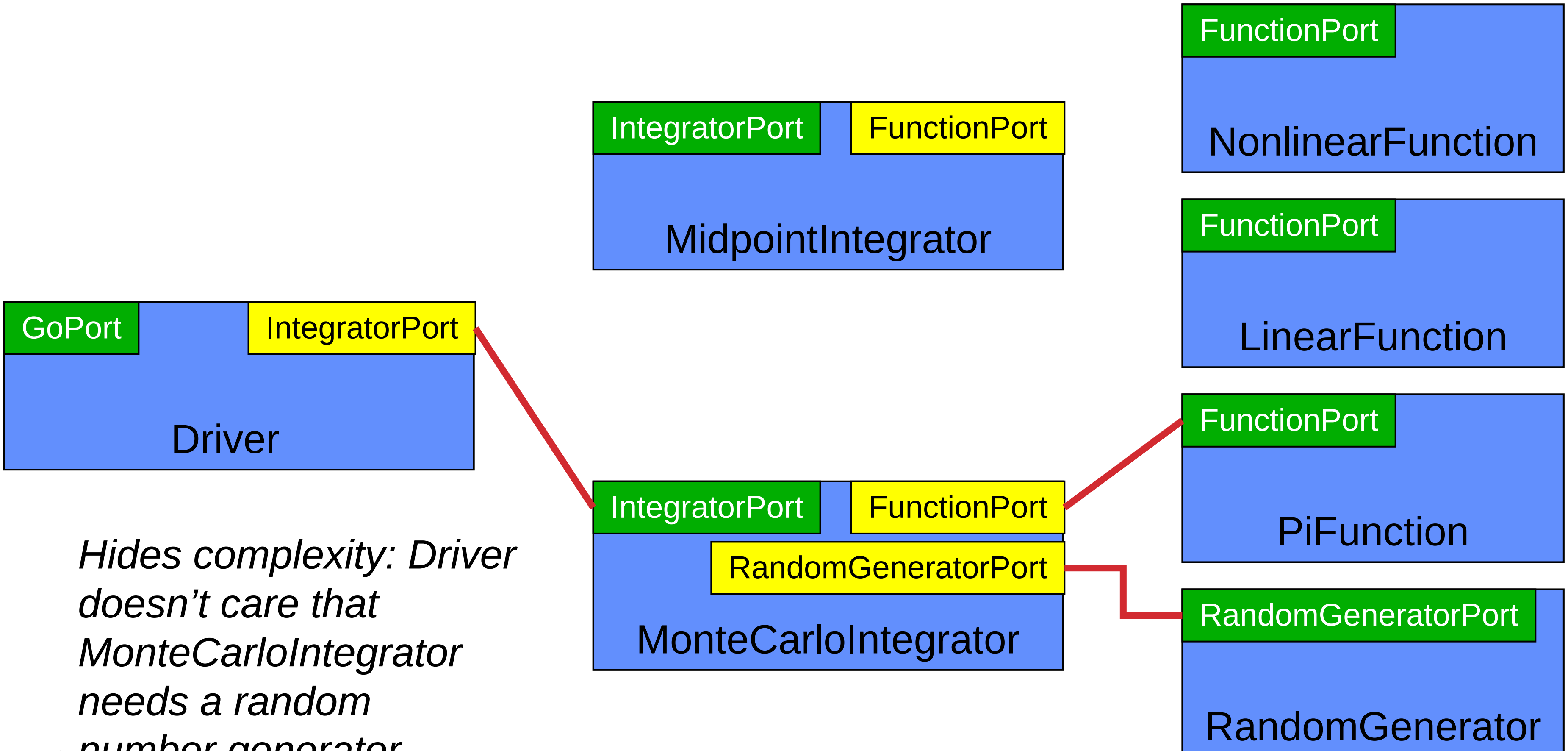
What is a Component Architecture?

- 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 are composed to form an application and executed (framework);
  - the rights and responsibilities of the framework.

A Simple Example: Numerical Integration Components

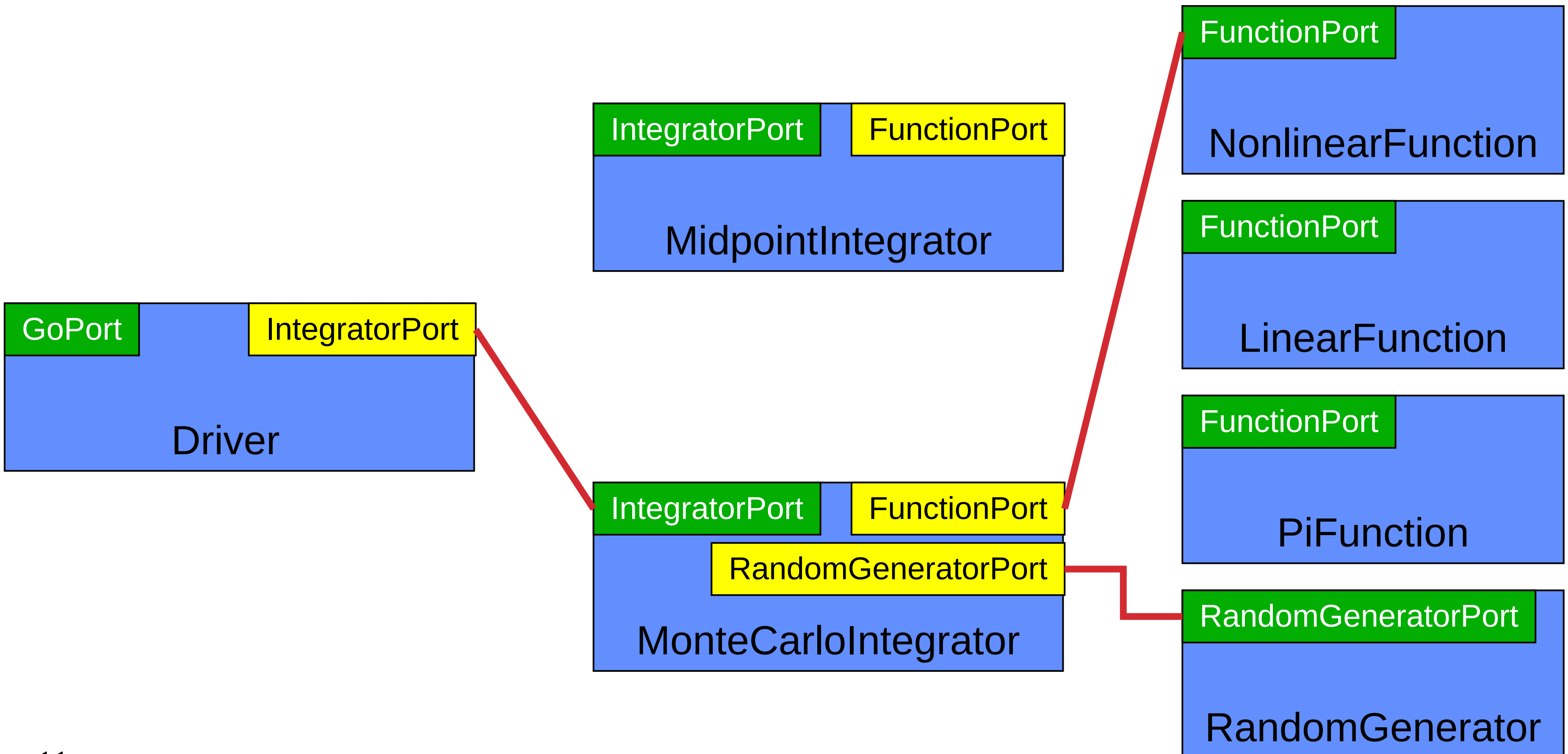


An Application Built from the Provided Components

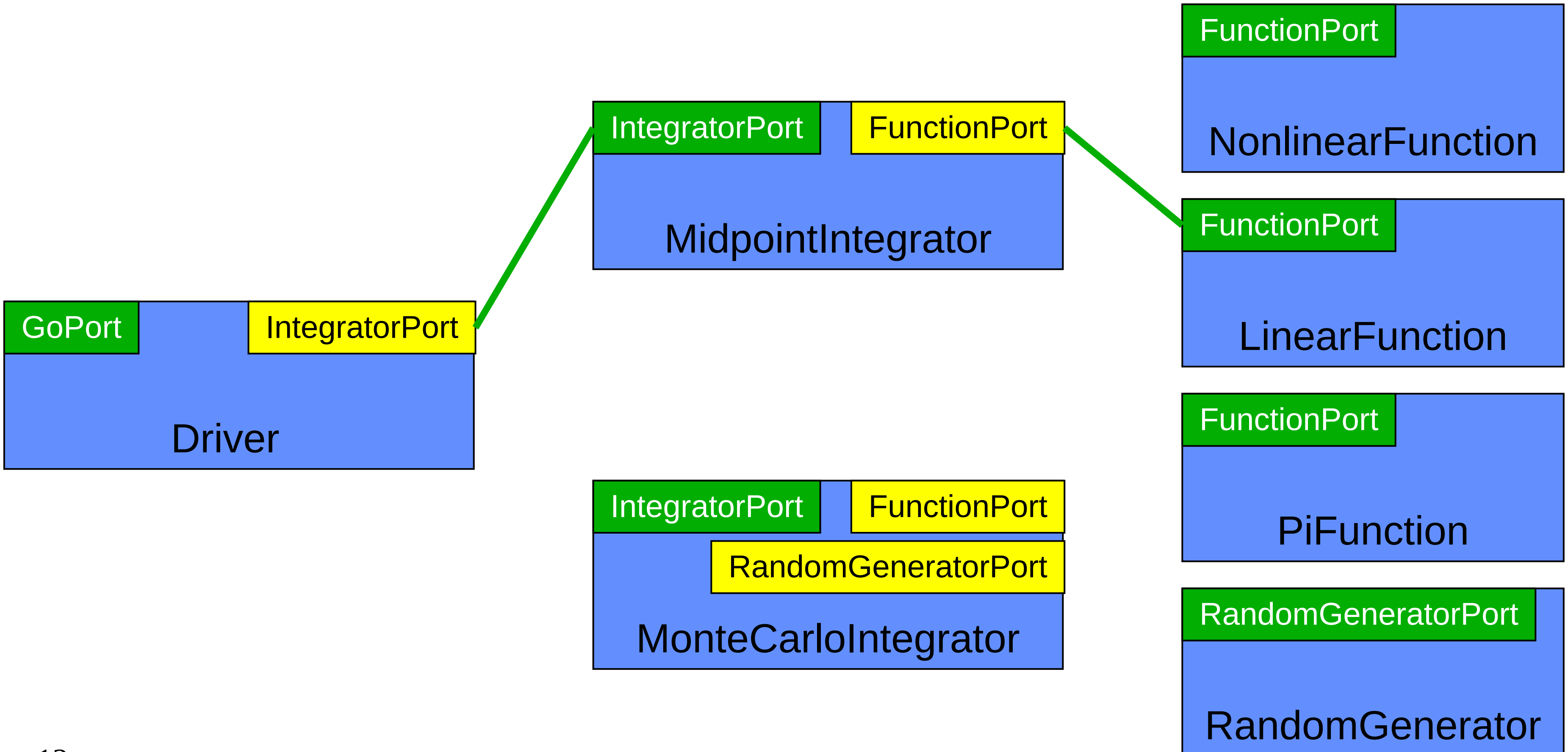


Hides complexity: Driver doesn't care that MonteCarloIntegrator needs a random number generator

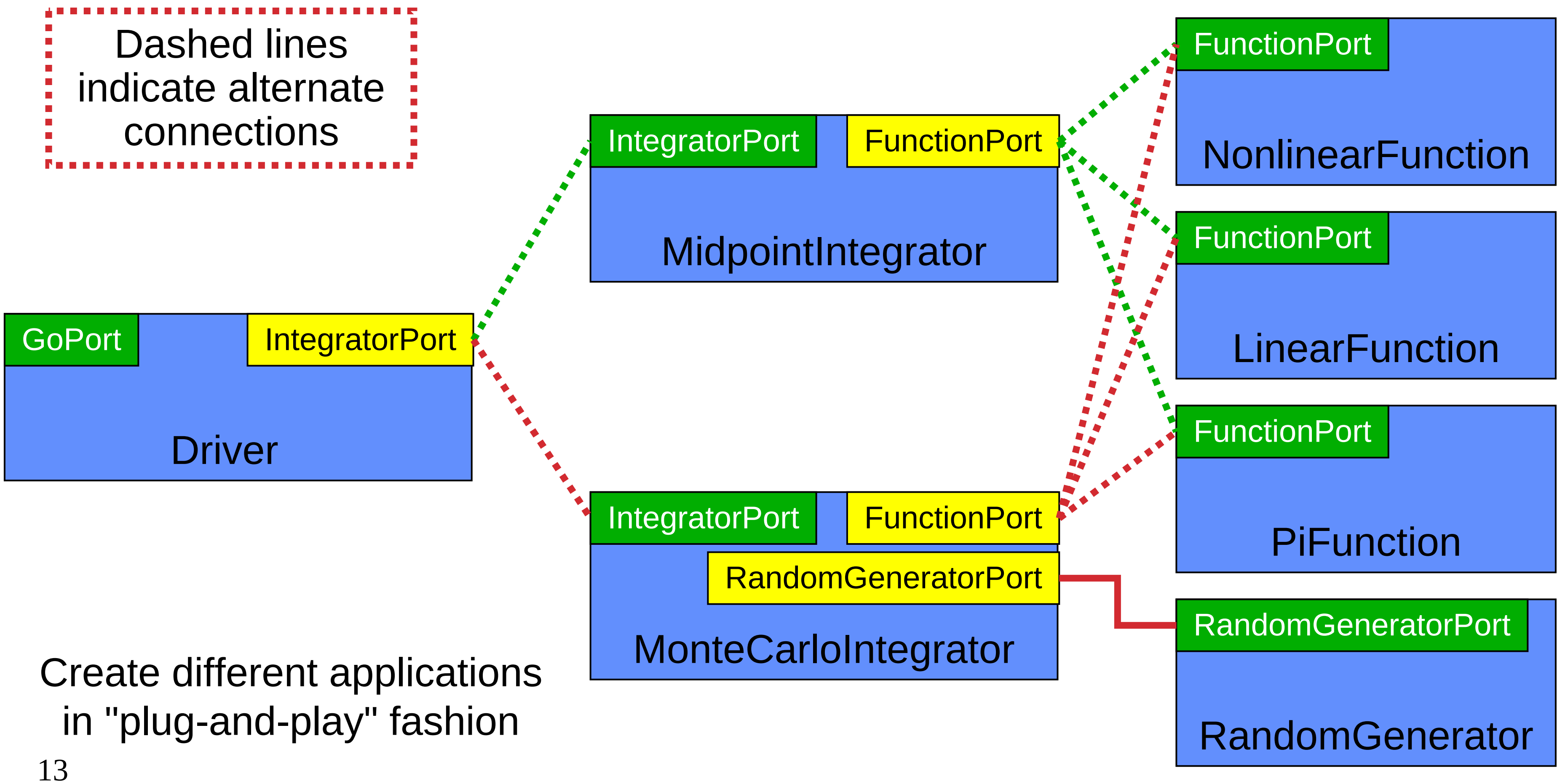
Another Application…



Application 3…

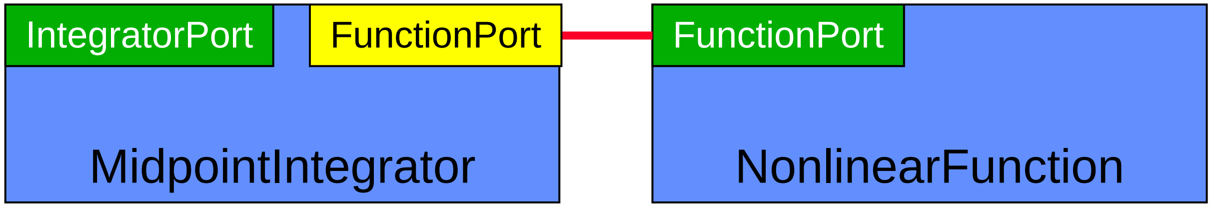


And Many More…



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

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).*

Special Needs of Scientific HPC

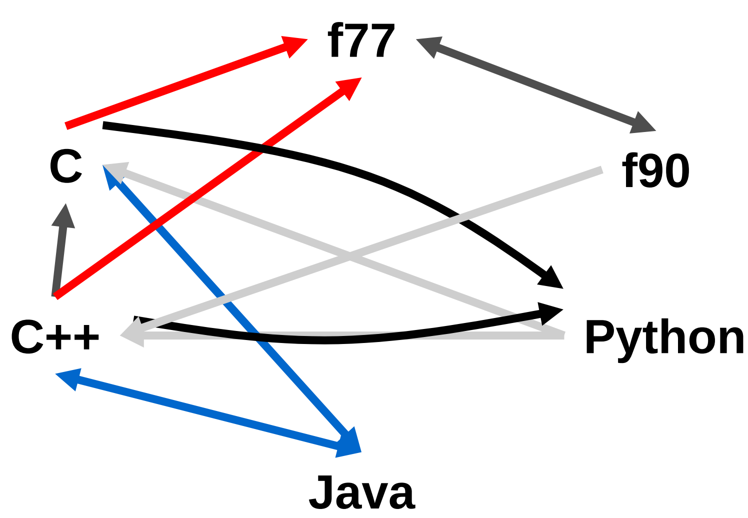
- Support for legacy software
  - How much change required for component environment?
- Performance is important
  - What overheads are imposed by the component environment?
- Both parallel and distributed computing are important
  - What approaches does the component model support?
  - What constraints 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?

Commodity Component Models

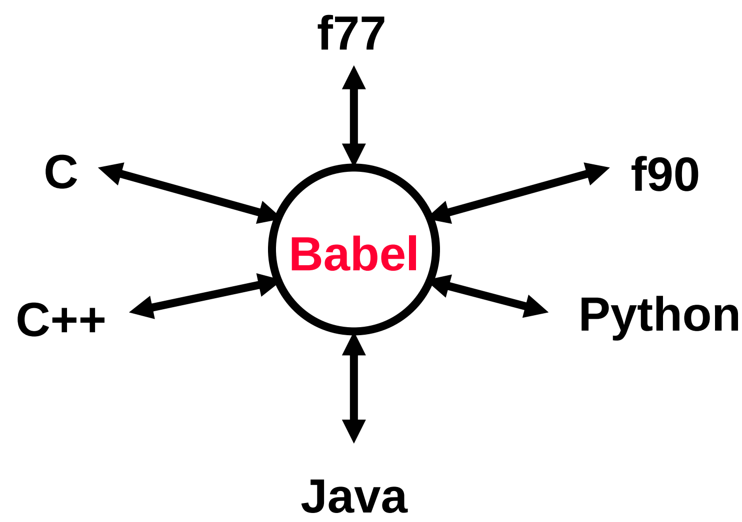
- CORBA Component Model (CCM), COM, Enterprise JavaBeans:
  - arise from business/internet software world.
- Componentization requirements can be high.
- Can impose significant performance overheads.
- No recognition of tightly-coupled parallelism.
- May be platform specific.
- May have language constraints.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

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

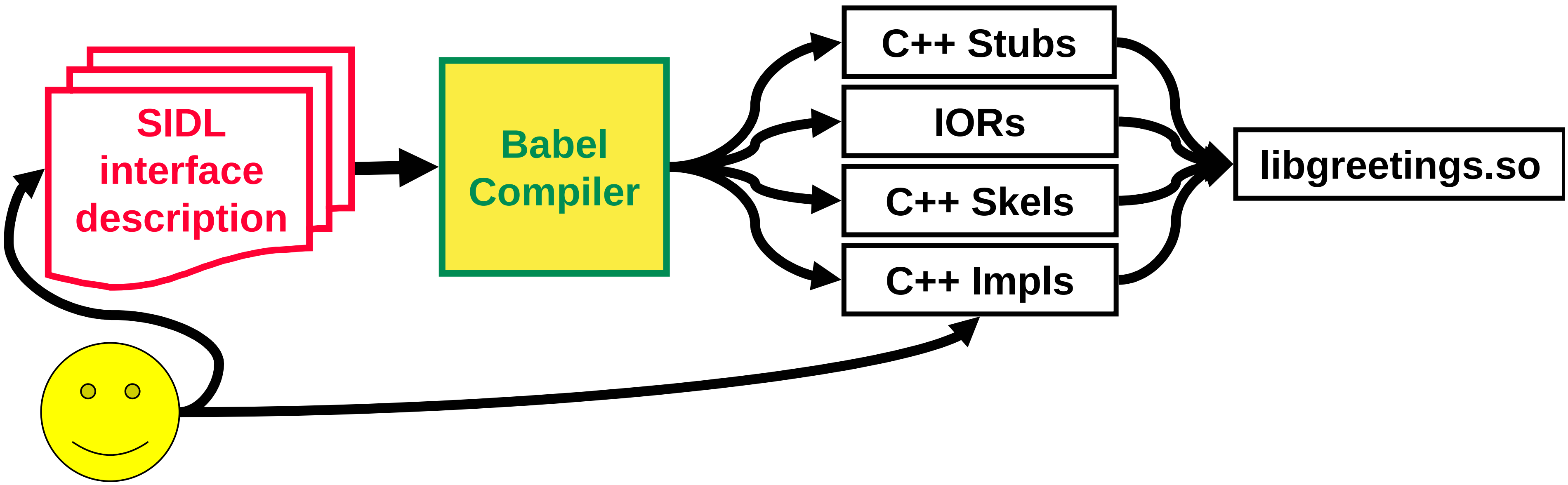


Babel is a compiler that processes Scientific Interface Definition Language (SIDL) files.

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 {  }
}
```

Library Developer Does This...



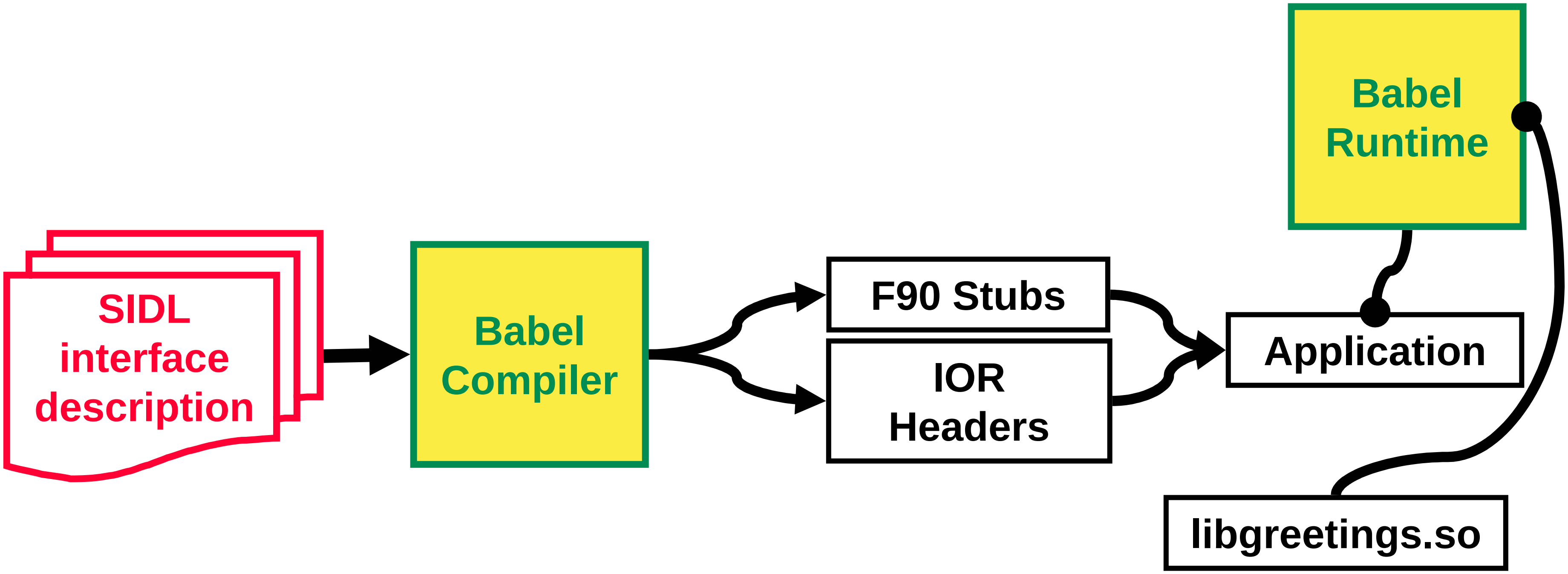
1. `babel --server=C++ greetings.sidl`
2. Add implementation details
3. Compile & Link into Library/DLL

Adding the Implementation

```
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)
// Skip to impl for setName()
```

```
// Implementation for setName() above
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)
}
```

Library User Does This...



1. `babel --client=F90 greetings.sidl`
2. Compile & Link generated Code & Runtime
3. Place DLL in suitable location

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 come from directly from the SIDL

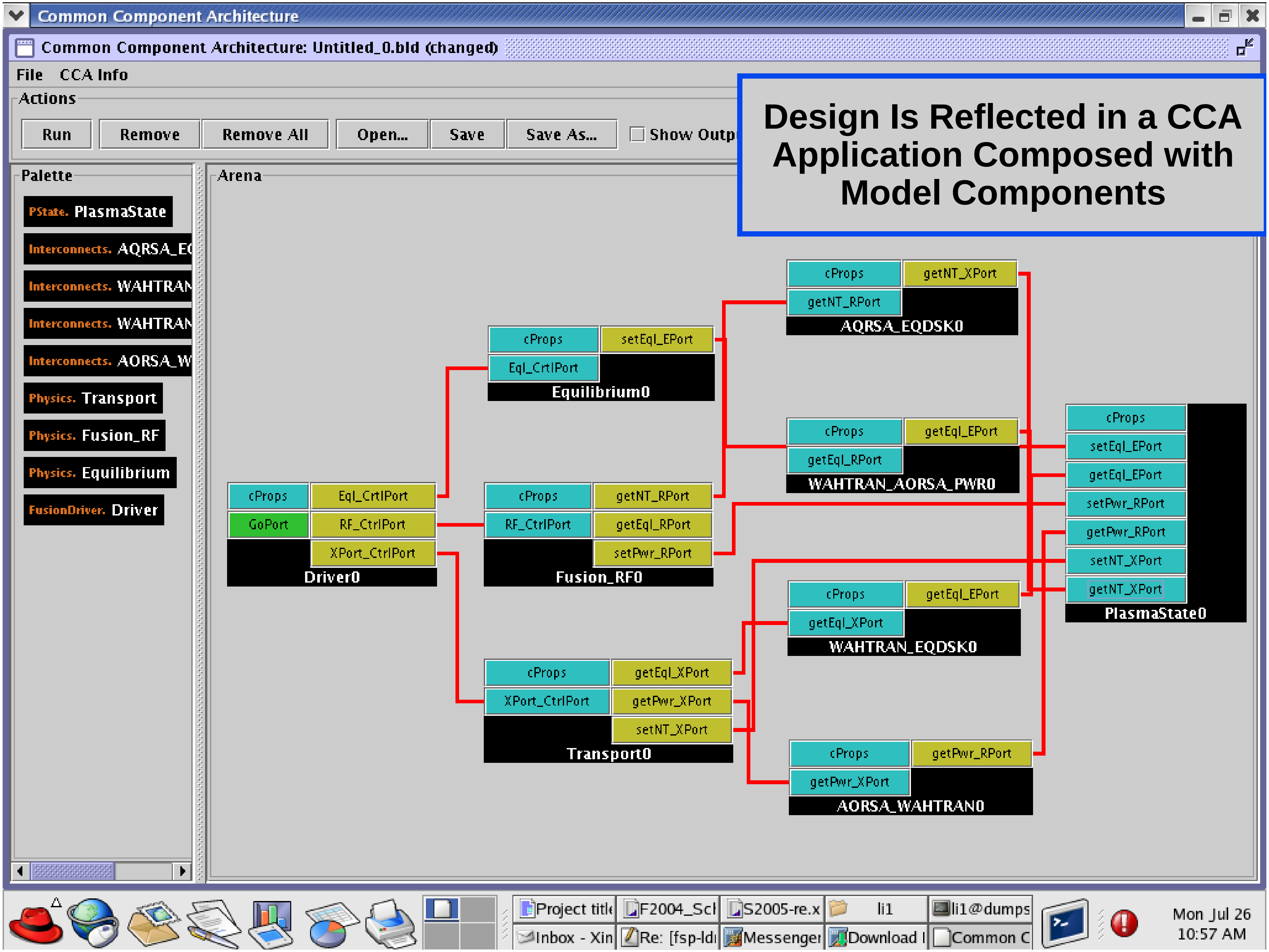
Some other subroutines are "built in" to every SIDL class/interface

Step I: Develop the design

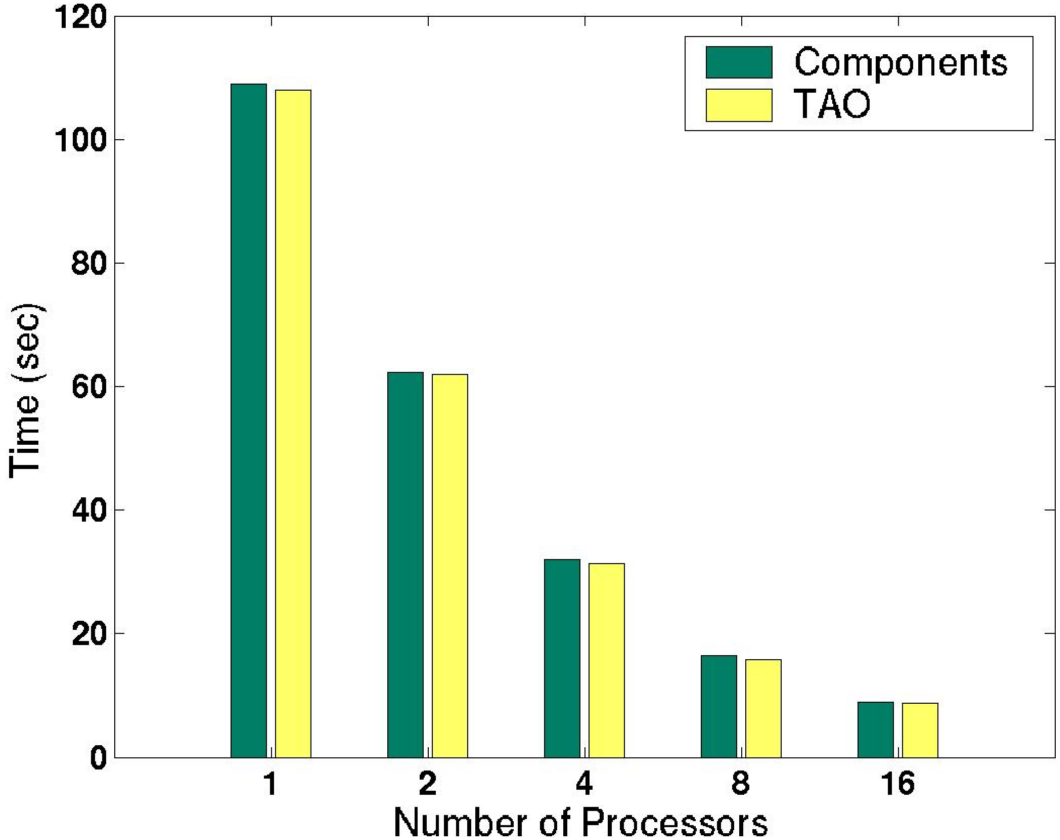
```
package fusion version 0.1 {
   package physics {
      interface XPort {
         // allocates storage for routines/methodes in XPORT component
         int XPInit(in string XportIn,                    // name of control file for Xport
                    out string XInitMessage,              // file open, not enough values, etc.
                    in int n_i,                           // number of ion species
                    in int n_r);                          // number of radial grid points
```

- Interfaces for components and their methods (ports) SIDL:
  - forces team design;
  - provides a language independent design;
  - focuses attention on data exchange and functionality.

Design Is Reflected in a CCA Application Composed with Model Components



Scalability on a Linux Cluster



Total execution time for the minimum surface minimization problem using a fixed-sized 250x250 mesh.

- Newton method with line search
- Solve linear systems with the conjugate gradient method and block Jacobi preconditioning (with no-fill incomplete factorization as each block's solver, and 1 block per process)
- Negligible component overhead; good scalability

Experience I
The good news.

- What you do inside of a component is your own business.
- Interface design is a necessary part of CCA.
- Babel, SIDL use is growing, features are being added.

Experience II
Characteristics you can live with.

- Data types
  - int, long int, bool, char;
  - single/double, complex single/double;
  - arrays of the above;
  - strings, opaque.
- Arrays are a structure.
- No optional arguments.
- Row/column ordering is not fixed.

Experience III
Issues

- Project must provide CompSci support:
  - physics/applied math staff provide modules, procedures etc., experts convert to components.
- Babel is operating system, compiler, dependent. (No storage standard for F90, varied .so/.dll imlementation.)
- File, build environment is complex—expert maintenance required. Stability is key.
- Long-term existence of support/development is not guaranteed.

Summary

- A CCA based project must be of sufficient size to justify necessary superstructure.
- Language interoperability works, eliminates endless discussions.
- Interface design is painful, but akin to design requirements, interface documents for fabrication projects.
- Component-based methodology should address need for wide range of uses for physics packages.
- We will be getting to some physics this year.