High-Performance Component-Based Scientific Software Engineering

Boyana Norris
Argonne National Laboratory
http://www.mcs.anl.gov/~norris

CSDMS Meeting: "Modeling for Environmental Change"
October 15, 2010

Acknowledgments

- Common Component Architecture Forum
- CSDMS
- DOE Office of Science

What are components?

- No universally accepted definition in CS; some features:

- 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

- Requires a managed execution environment; can be composed with other components dynamically
  - "Plug and play" model to build applications
  - Composition based on interfaces

What is a component architecture?

- A set of standards that allows:
  - Multiple groups to write units of software (components)…
  - And have confidence that their components will work with other components written in the same architecture
- These standards define…
  - The rights and responsibilities of a component
  - How components express their interfaces
  - The environment in which components are composed to form an application and executed (framework)
  - The rights and responsibilities of the framework

Object-oriented vs component-oriented development

- Components can be implemented using OO techniques
- Component-oriented development can be viewed as augmenting OOD with certain policies, e.g., require that certain abstract interfaces be implemented
- Components, once compiled, may require a special execution environment
- COD focuses on higher levels of abstraction, not particular to a specific OO language
  - abstract (*common*) interfaces
  - dynamic composability
  - language interoperability
- Can convert from OO to CO specific to a given framework, possibly with some level of automation

What is the CCA?

- Component-based software engineering has been developed in other areas of computing
  - Especially business and internet
  - Examples: CORBA Component Model, COM, Enterprise JavaBeans
- Many of the needs are similar to those in HPC scientific computing but scientific computing imposes special requirements not common elsewhere
- CCA is a component environment specially designed to meet the needs of HPC scientific computing
  - The CCA Forum (open to all) was formed in 1999
  - Has been supported through multiple DOE projects since 2000

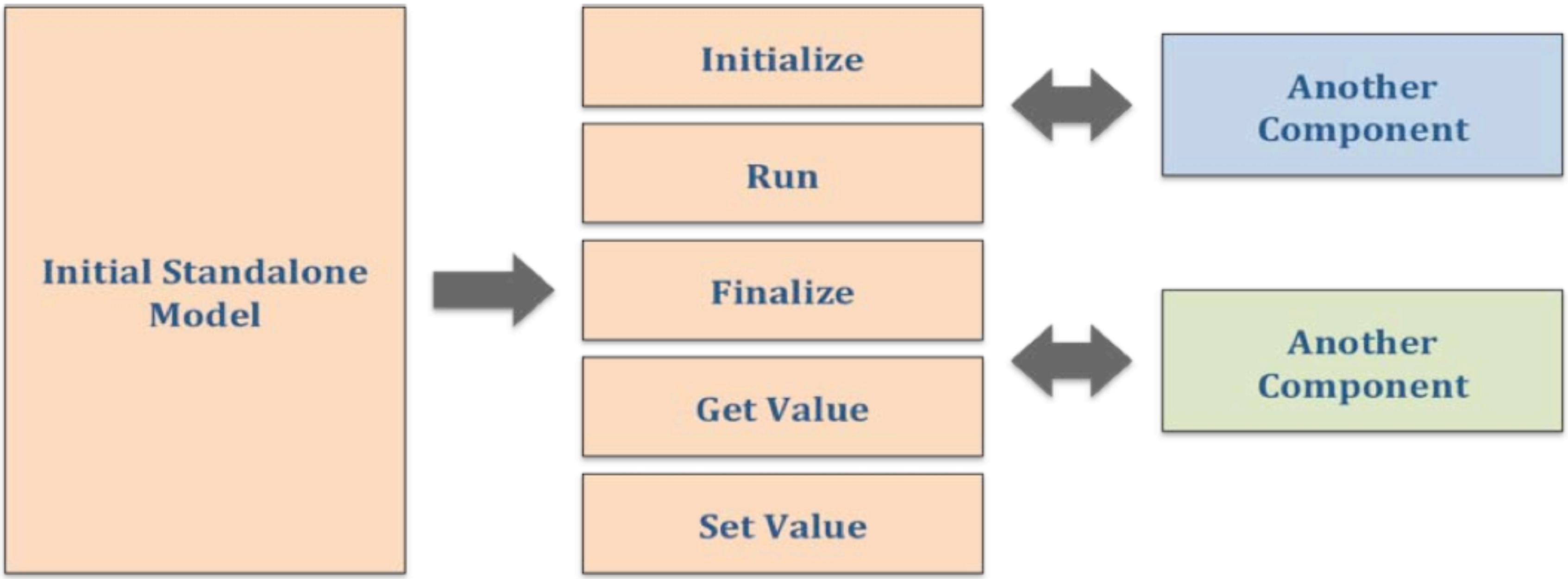
Special needs of scientific HPC

- 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?

Some examples

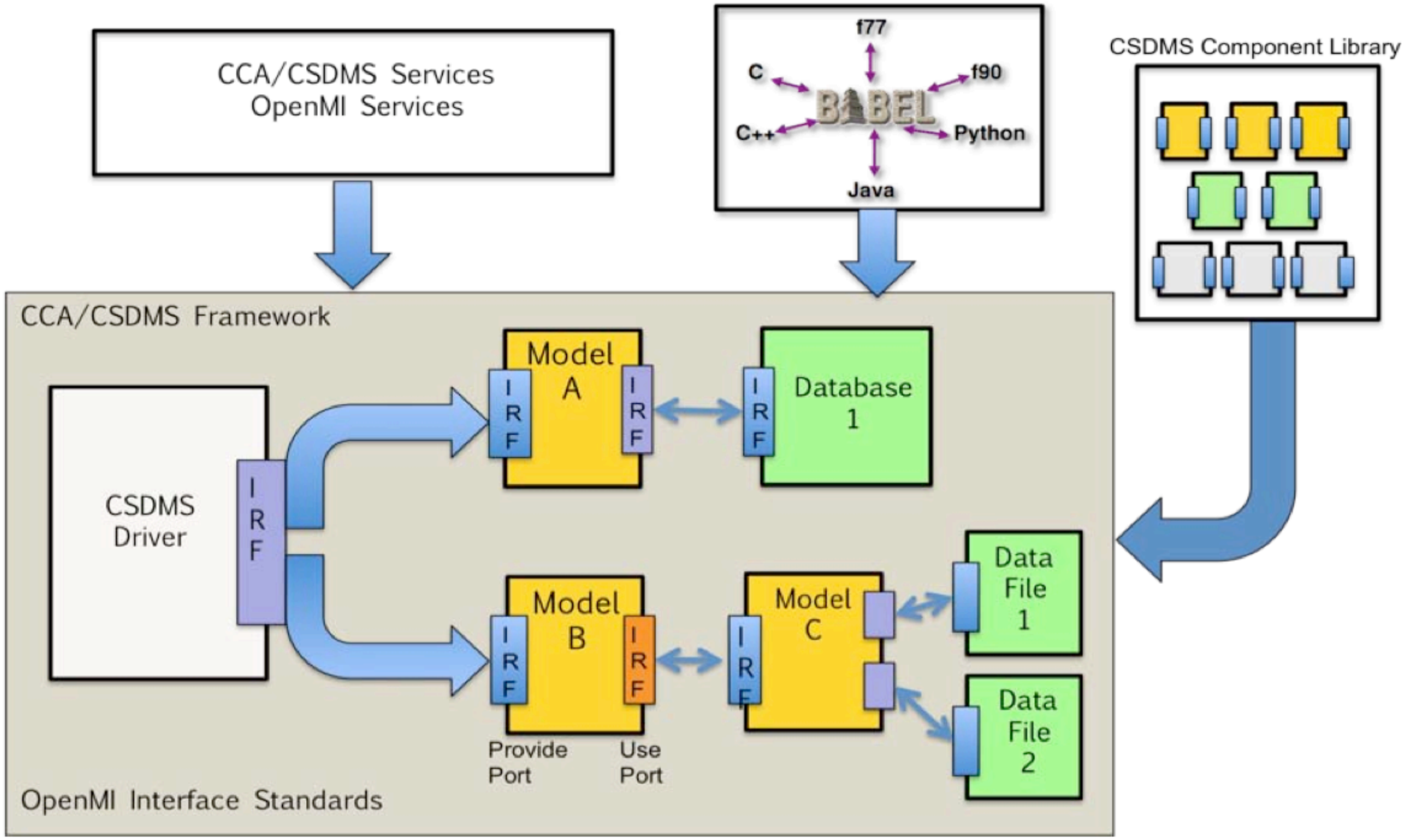
- Different component granularity
- Few vs many interfaces

CSDMS interfaces

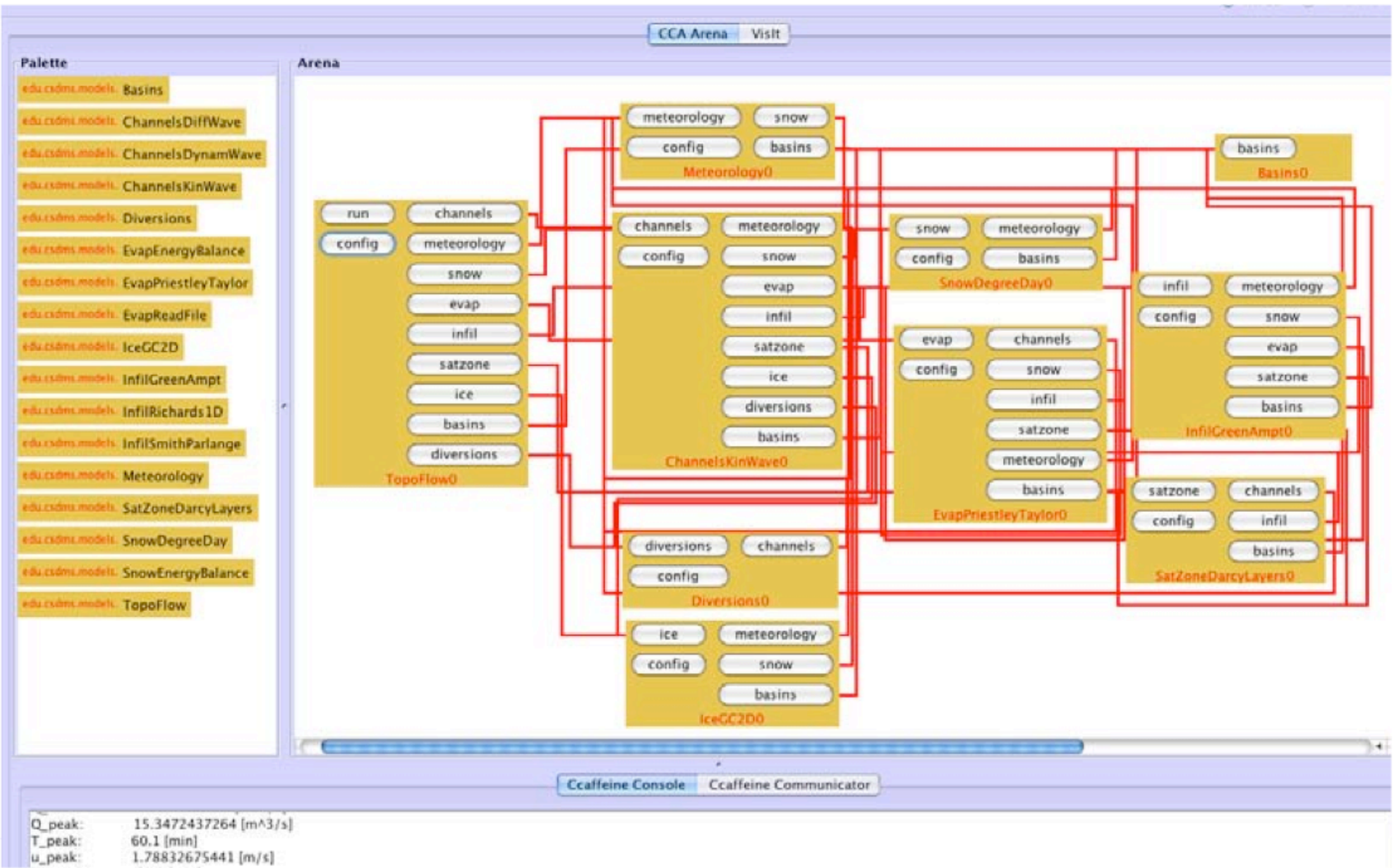


- 2009 CSDMS Annual Report

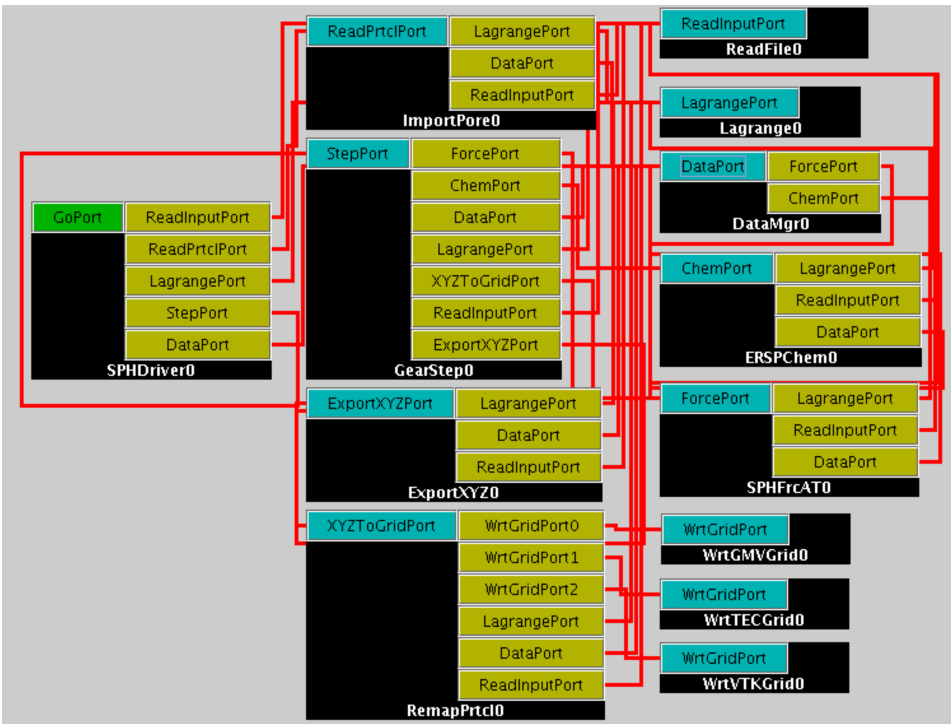
CSDMS Framework

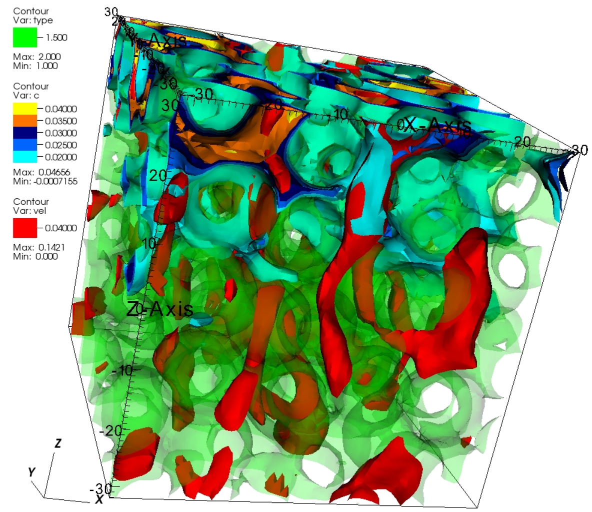


CSDMS Integrated Component Simulation



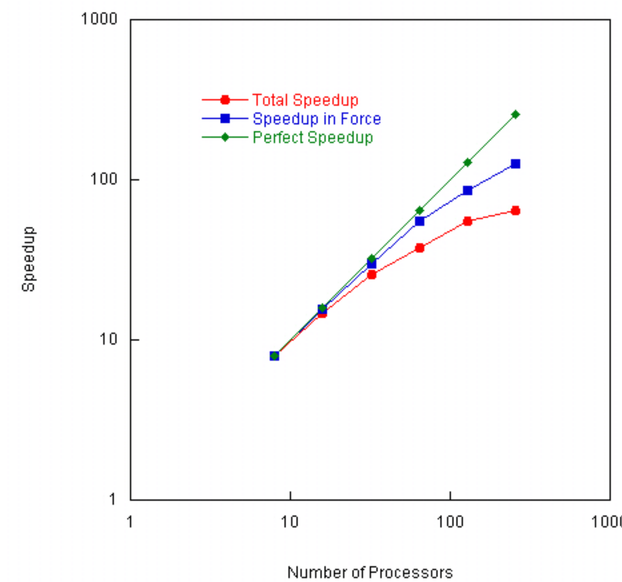
SPH Groundwater Modeling Framework





CCA
Common Component Architecture

VACET



- Bruce Palmer, PNNL

SWIM: Integrated Plasma Simulation Framework

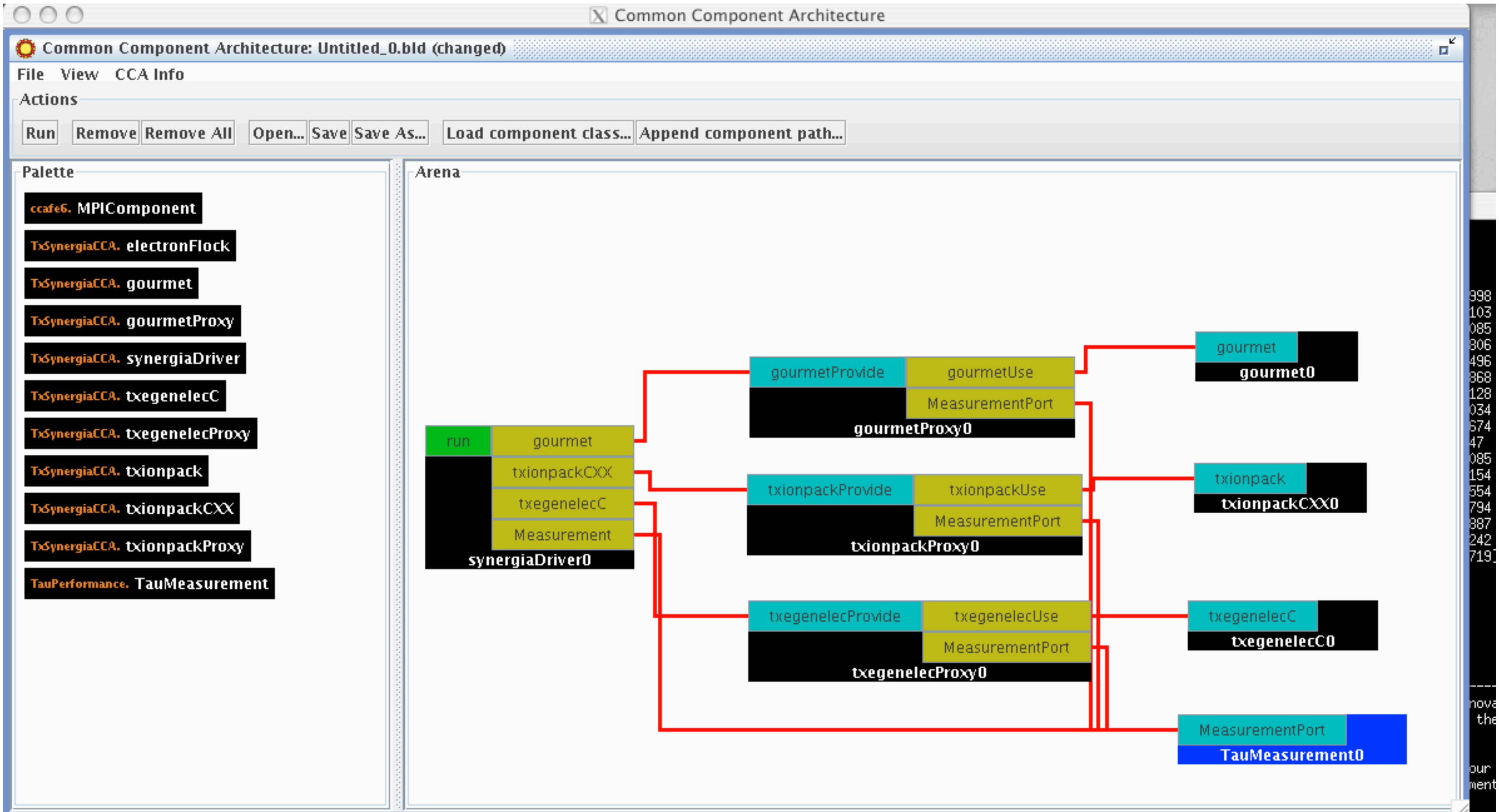
A flexible, extensible computational framework capable of coupling state-of-the-art models for energy and particle sources, transport, and stability for tokamak core plasma. www.cswim.org -Don Batchelor et al.

- CCA implementation
- File-based communication between components

SWIM: Integrated Plasma Simulation Framework Component Interface

```
from  component import Component

class HelloDriver(Component):
  def __init__(self, services, config):
    Component.__init__(self, services, config)
    print 'Created %s' % (self.__class__)

  def init(self, timeStamp=0.0):
    return

  def step(self, timeStamp=0.0):
    return

  def finalize(self, timeStamp=0.0):
    return
```

Accelerator Simulations



- ❑ Source: Tech-X Corporation

Benefits of Component-Based Software Development

- Software evolution and maintainability
- Encourage (force?) people to reach agreements in order to define interfaces
- Enable independent development
- Plug-and-play application composition

The Dark Side

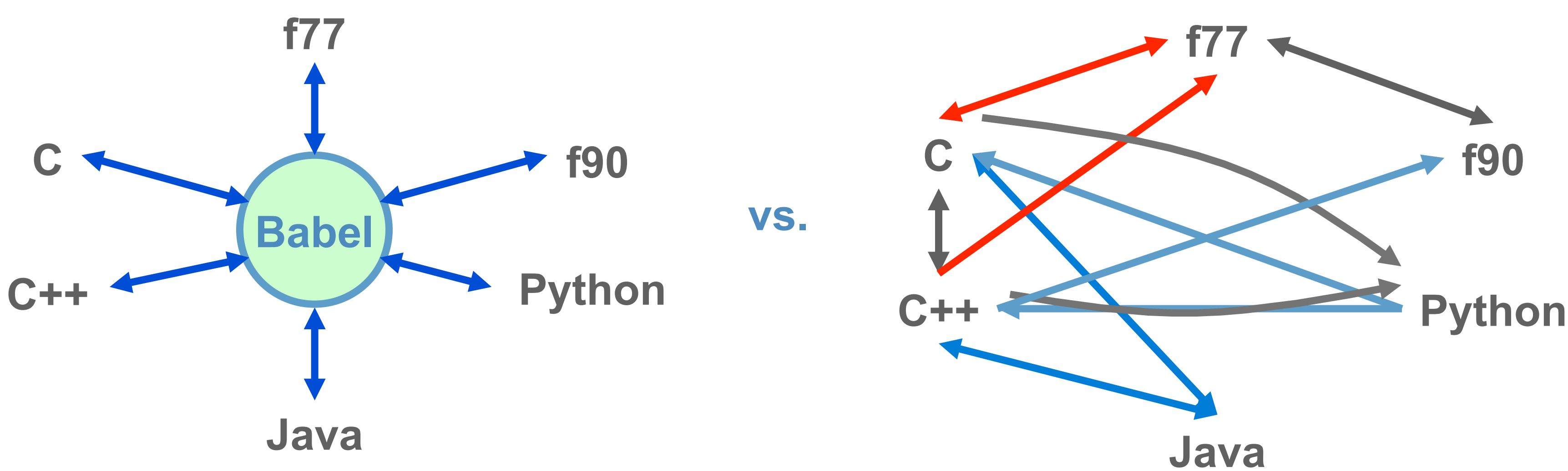
- Identifying components and designing clean interfaces is harder than writing less modular code
- Components require extra code
- Multiple abstraction layers add extra runtime overhead
- Complications of interoperability
  - Different grids, boundary conditions, etc.

On the Bright Side

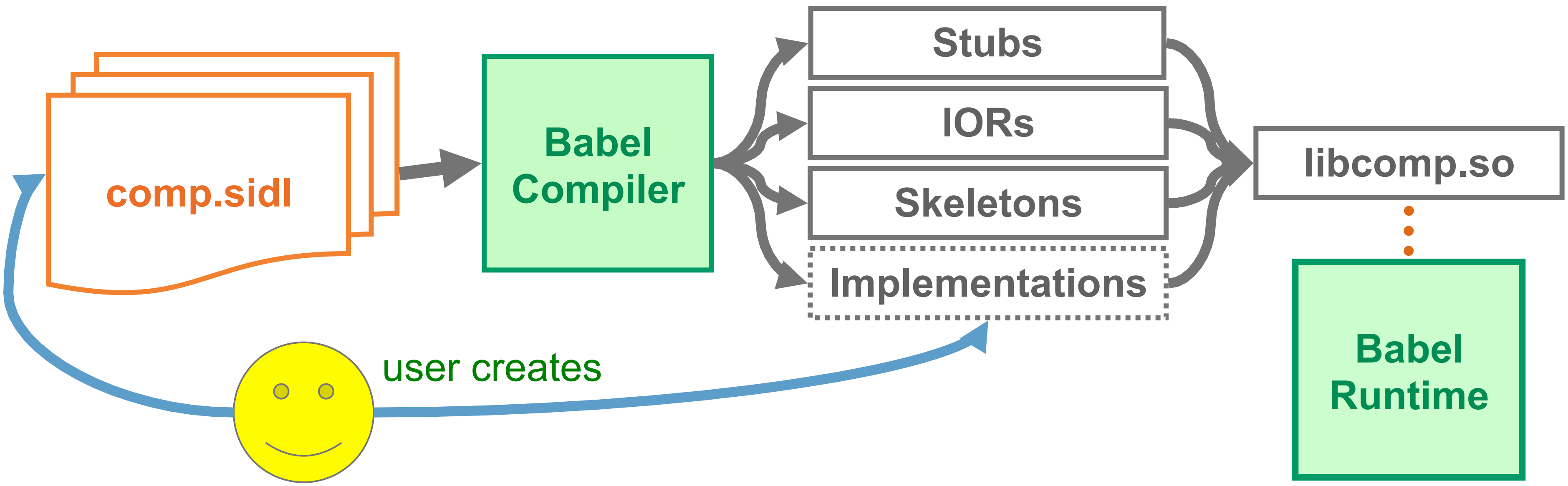
- Better code reuse including *outside* a project
  - Less software to develop from scratch
  - Ability to leverage expert knowledge/division of labor
- Huge potential for usability-enhancing automation
  - Development processes (software creation, modification, builds, testing)
  - Code generation
  - Link- and run-time optimizations
- Community-specific frameworks that make component development highly productive
- Language interoperability

Language interoperability with Babel

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



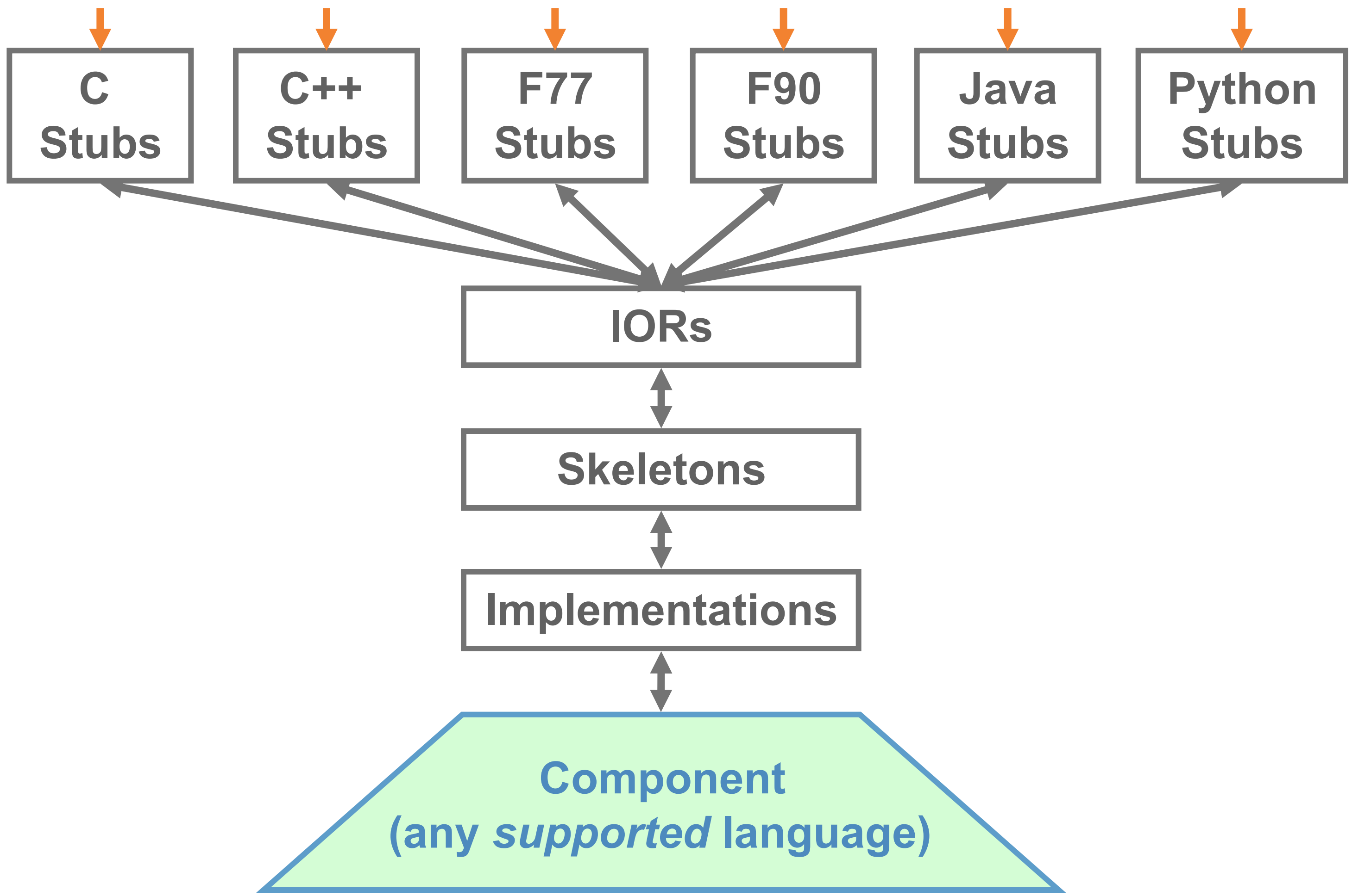
Babel Generates object-oriented language interoperability middleware



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

IOR = Intermediate Object Representation SIDL = Scientific Interface Definition Language

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



IOR = Intermediate Object Representation

An example of usability improvement: Managing projects with Bocca

- The interoperability, connectivity, and modularity of components is independent of their function.
- Bocca creates and manages a graph representation of interface and component dependencies, which is used to
  - Generate the build system
  - Generate the “glue” code for language interoperability
  - Generate component metadata
  - Generate tests

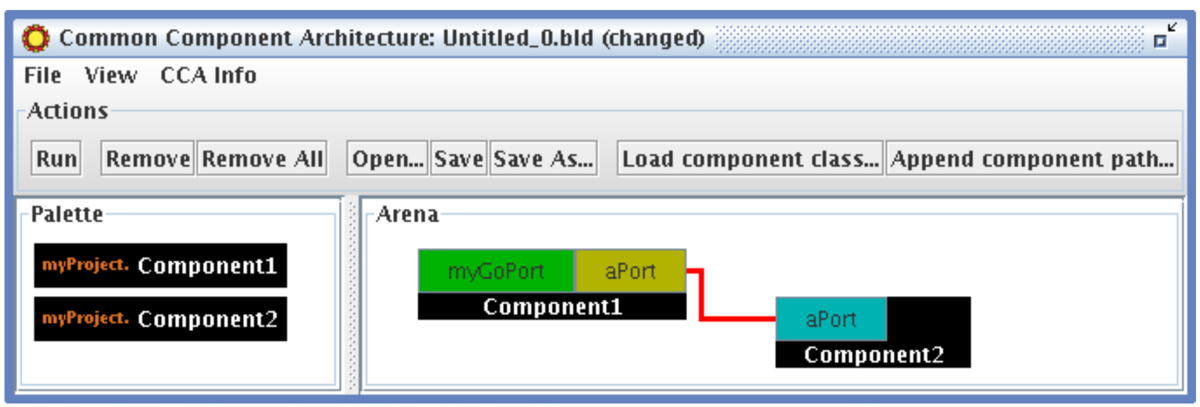
Bocca command-line example

- Create some components with ports:

Port Type
Port Name
Component Type

```
$ bocca create project myProject
$ cd myProject
$ bocca create port myPort
$ bocca create component --uses=myPort@aPort --go=myGoPort Component1
$ bocca create component --provides=myPort@aPort  Component2
```

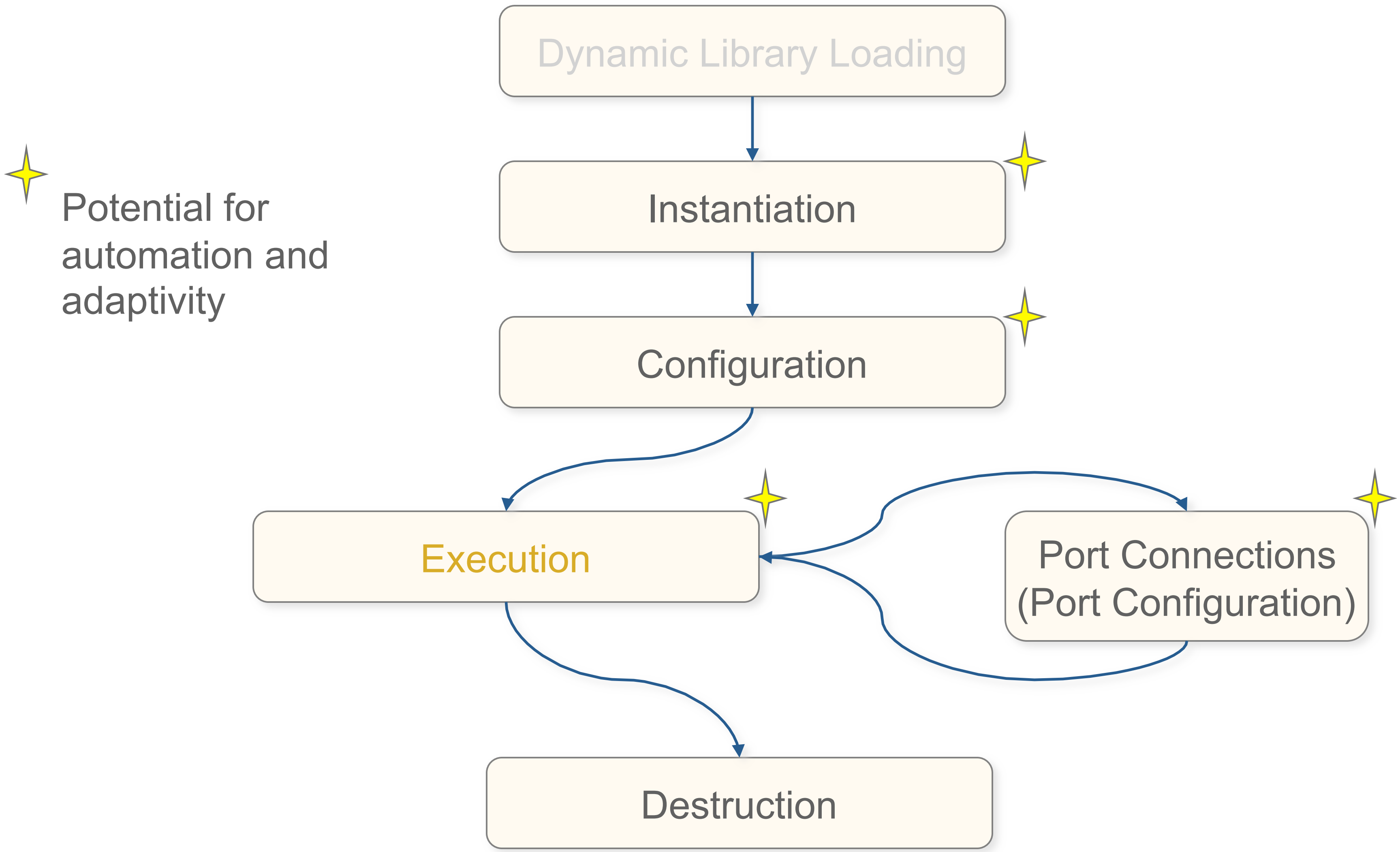
- From the CCA perspective these are fully functional components (no implementation however):



Adapt component applications

- Leverage the fact that components are plug-and-play
- Automate the configuration and runtime adaptation of high-performance component applications, through the so called Computational Quality of Service (CQoS) infrastructure
  - Instrumentation of component interfaces
  - Performance data gathering
  - Performance analysis
  - Adaptive algorithm support

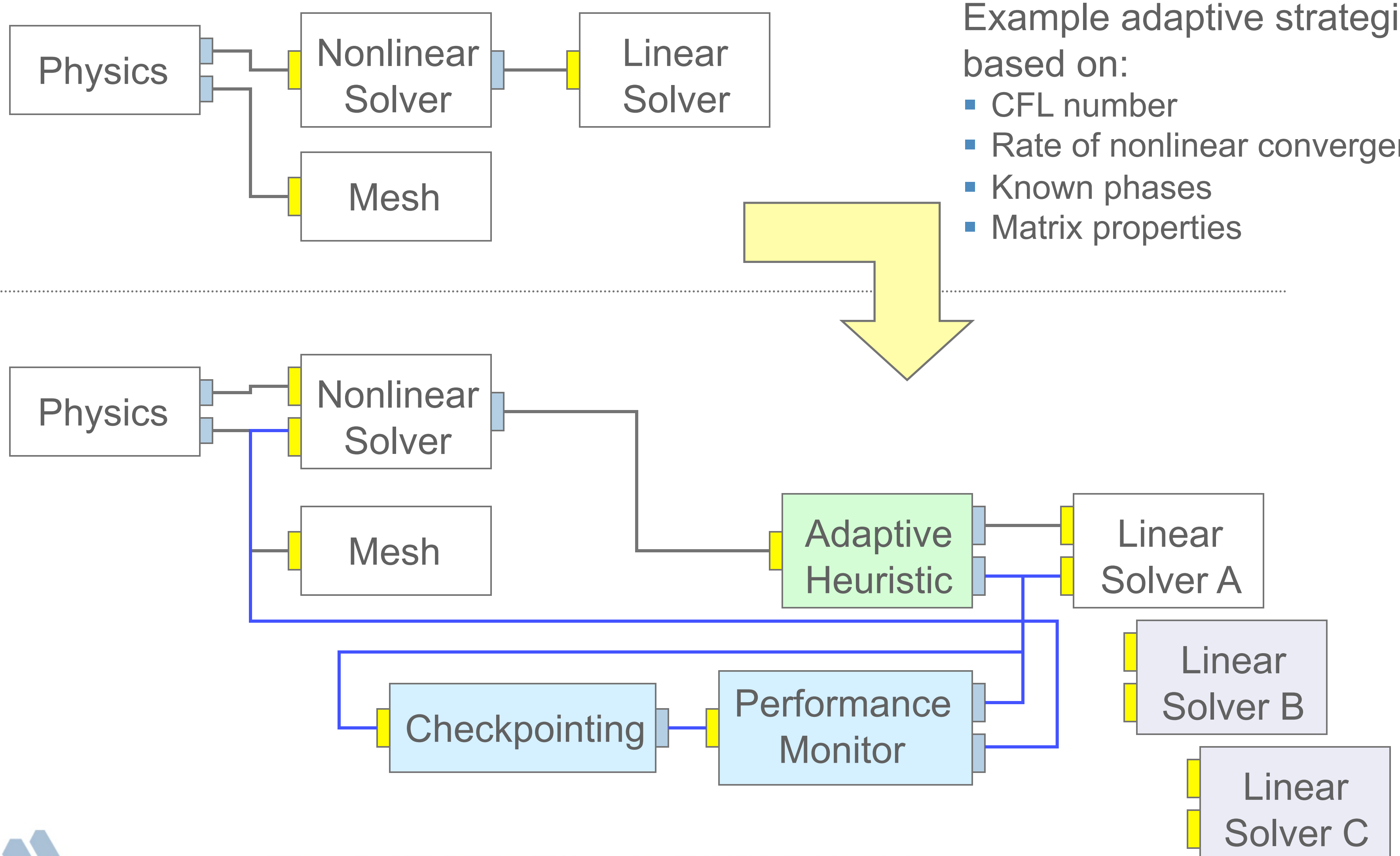
Major events in a component's lifetime



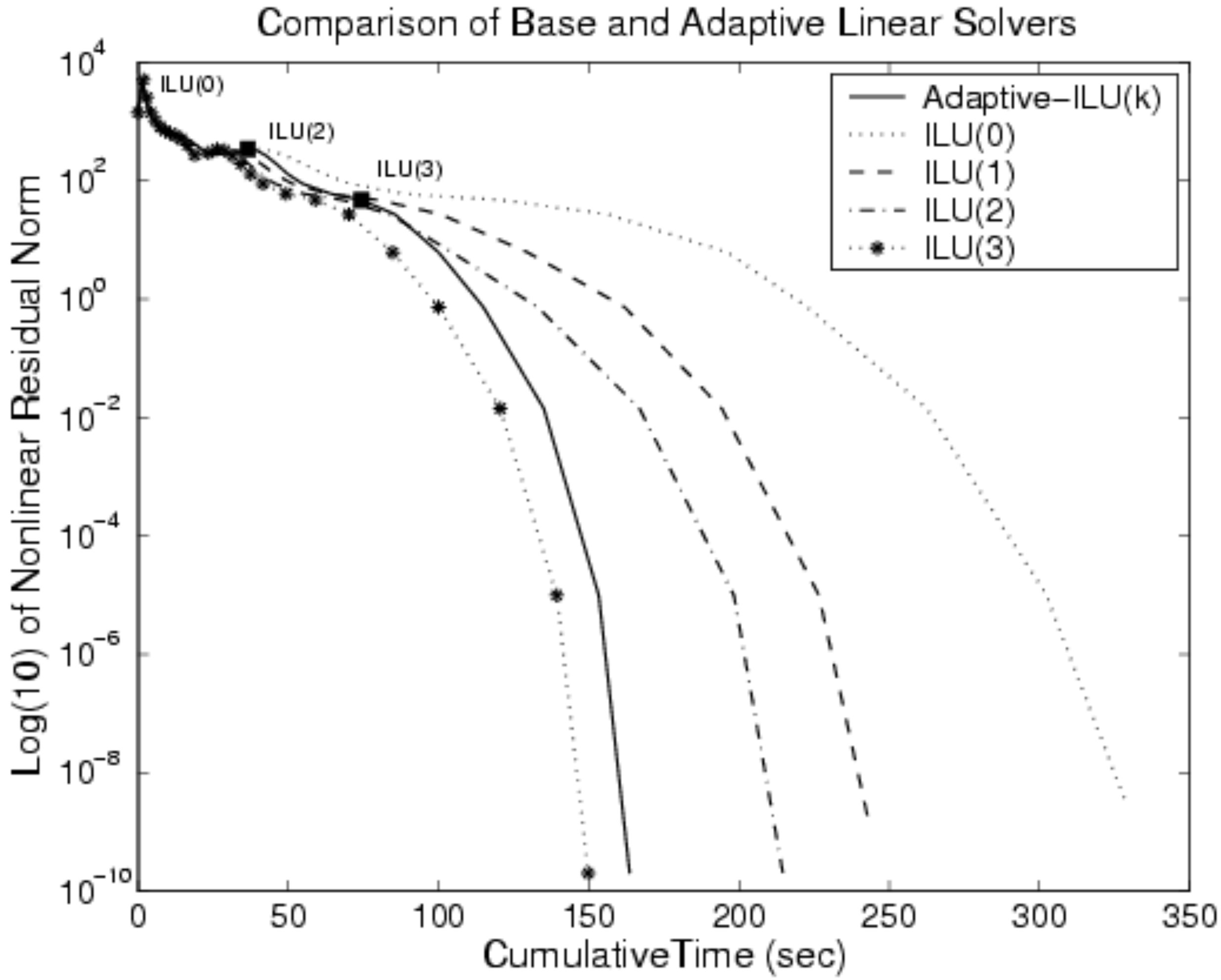
Example: Multimethod linear solver components in nonlinear PDE solution

Example adaptive strategies based on:

- CFL number
- Rate of nonlinear convergence
- Known phases
- Matrix properties



Example 1: 2D Driven Cavity Flow[1]

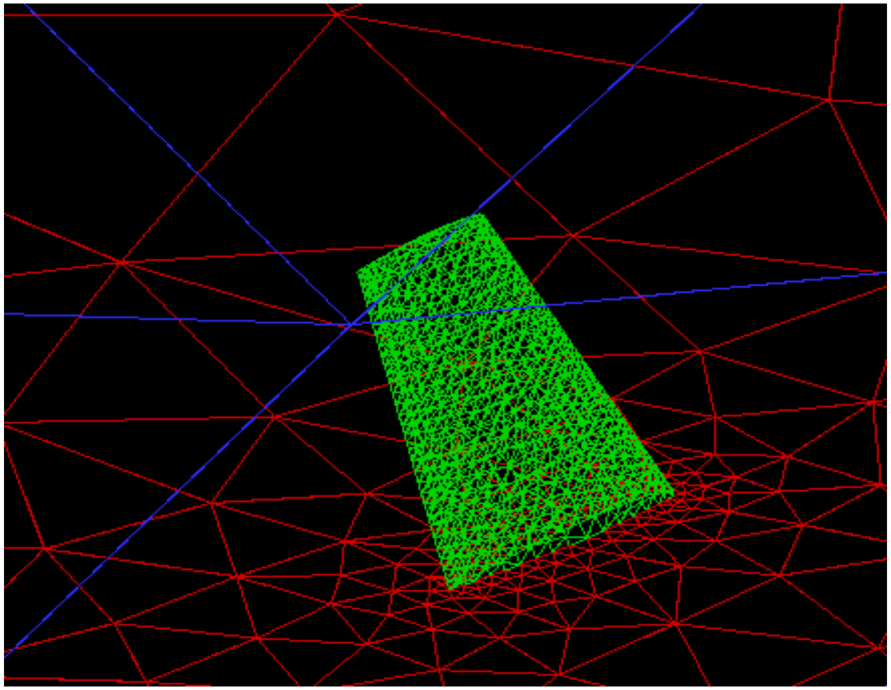


- Driven cavity flow, which combines lid-driven flow and buoyancy-driven flow in a two-dimensional rectangular cavity. We use a velocity=vorticity formulation of the Navier-Stokes and energy equations, discretized on a uniform Cartesian mesh.

[1] T. S. Coffey, C.T. Kelley, and D.E. Keyes. Pseudo-transient continuation and differential algebraic equations. *SIAM J. Sci. Comp*, 25:553–569, 2003.

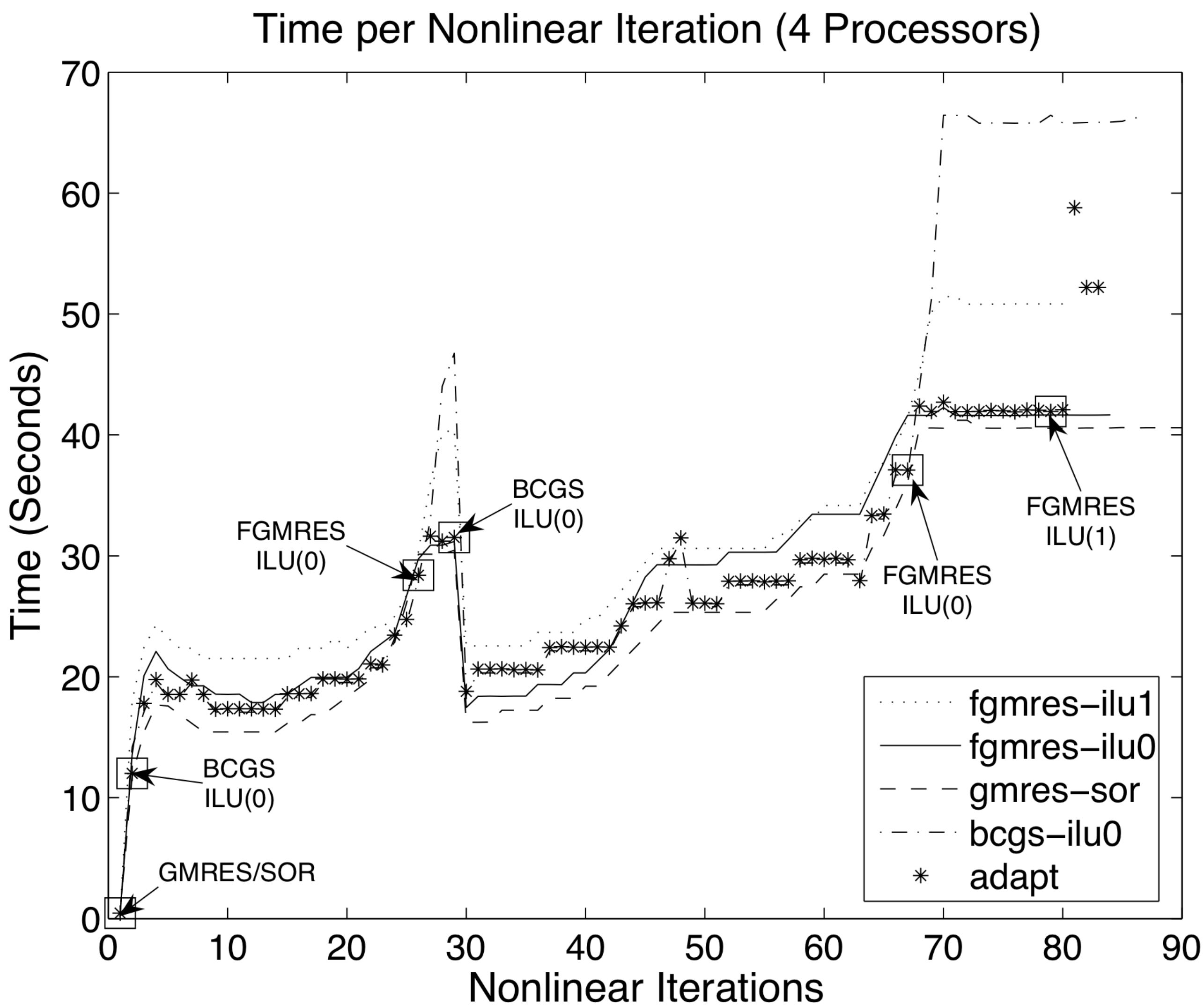
Example 2: PETSc-FUN3D

- 3D compressible Euler (used in this work; also supports incompressible Navier-Stokes)
- Fully implicit, steady-state
- Developed by D. Kaushik et al. (ANL)
- Based on FUN3D (developed by W.K. Anderson, NASA Langley)
  - Tetrahedral, vertex-centered unstructured mesh
  - Discretization: 1st or 2nd order Roe for convection and Galerkin for diffusion
- Pseudo-transient continuation
  - backward Euler for nonlinear continuation toward steady-state solution
  - Switched Evolution/relaxation (SER) approach of Van Leer and Mulder



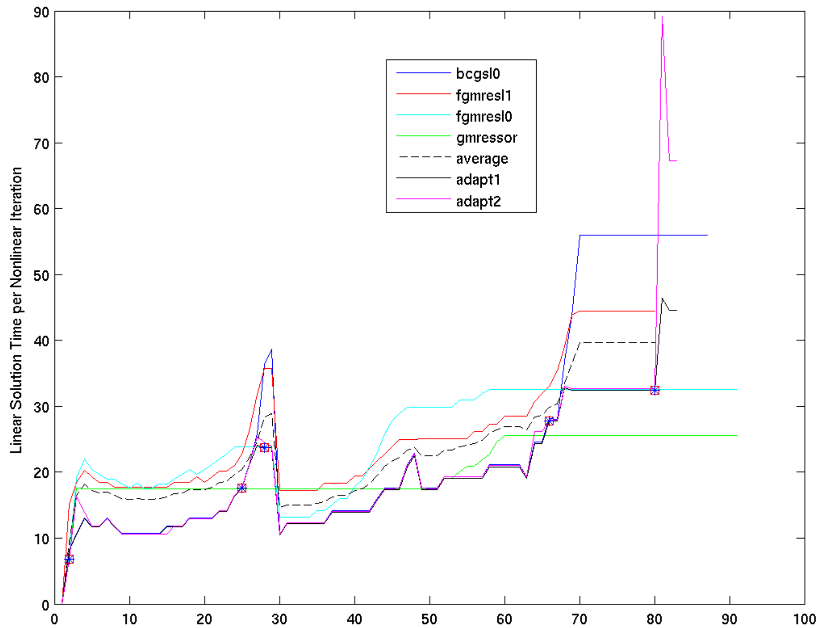
- Newton-Krylov nonlinear solver
  - Matrix-Free (2nd order FD)
  - Preconditioner (1st order analytical)
- Won Gordon Bell prize at SC99; ongoing enhancements and performance tuning

Adaptive linear solver components



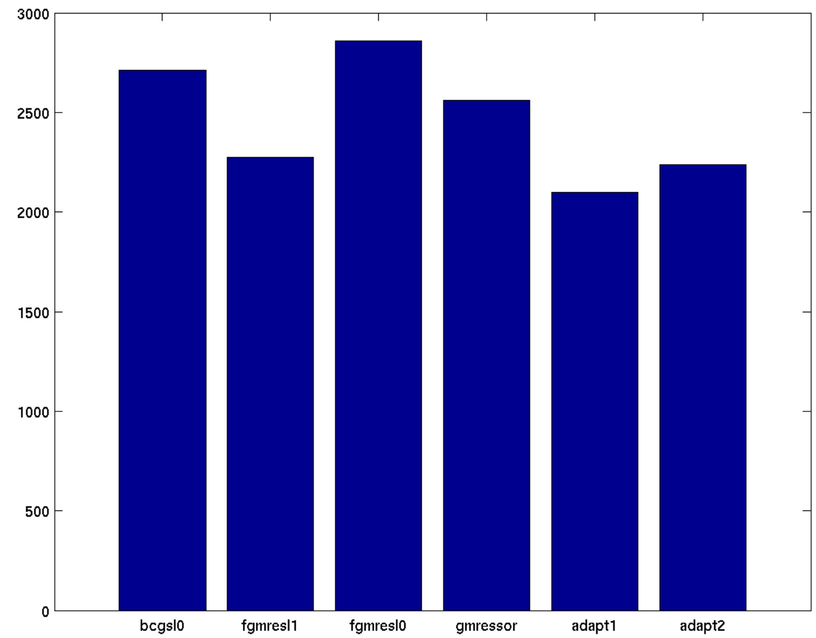
Example 2: FUN3D

Linear solution time per nonlinear iteration



Nonlinear Iteration Number

Cumulative time (seconds)



base methods adaptive

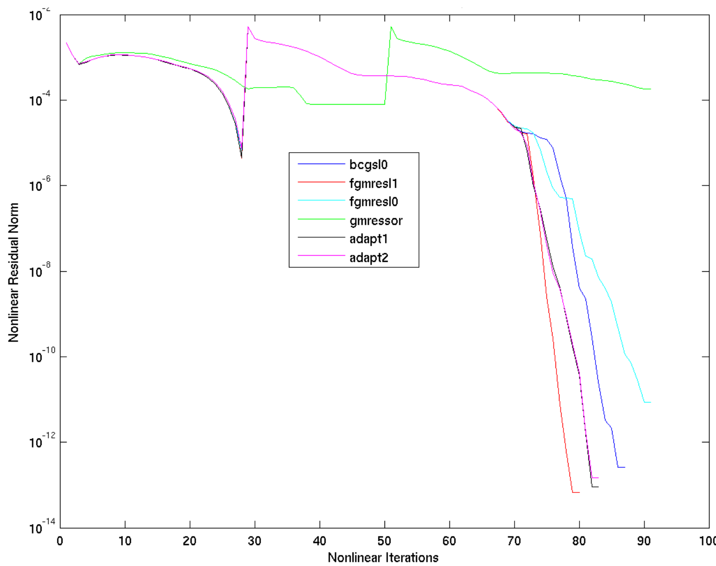
Adapt1: (1) 1st order: BCGS / BJacobi with ILU(0)
(25) 1st order: FGMRES(30) / BJacobi with ILU(0)
(28) 2nd order: BCGS / Bjacobi with ILU(0)
(66) 2nd order: FGMRES(30) / BJacobi with ILU(0)
(80) 2nd order: FGMRES(30) / BJacobi with ILU(1)

Adapt2: (1) 1st order: GMRES(30) / Bjacobi with SOR
(2) 1st order: BCGS / BJacobi with ILU(0)
(25) 1st order: FGMRES(30) / BJacobi with ILU(0)
(28) 2nd order: BCGS / Bjacobi with ILU(0)
(66) 2nd order: FGMRES(30) / BJacobi with ILU(0)
(80) 2nd order: FGMRES(30) / BJacobi with ILU(1)

Example 2: FUN3D (cont.)

- Comparison of traditional fixed linear solvers and an adaptive scheme, which uses a different preconditioner during each of the phases of the pseudo-transient Newton-Krylov algorithm
- MCS Jazz cluster (2.4 GHz Pentium Xeon with 1 or 2 GB RAM), 4 nodes

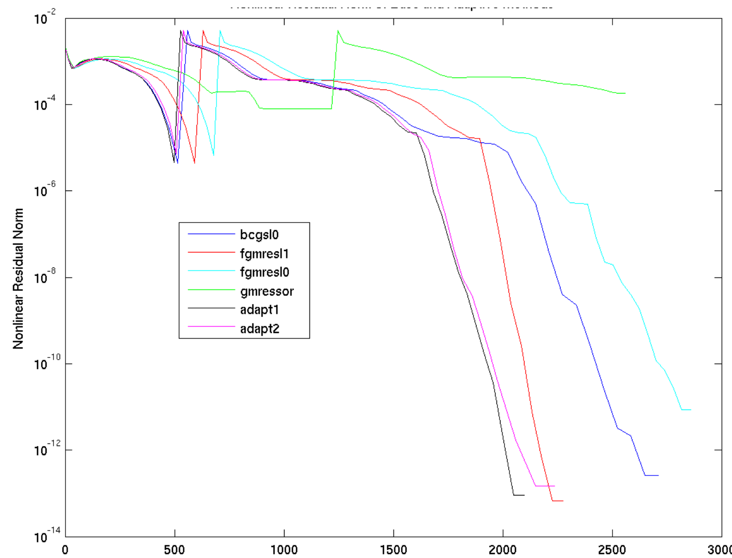
Convergence rates of base and adaptive methods



Nonlinear iterations

Adapt1: (1) 1st order: BCGS / BJacobi with ILU(0)
(25) 1st order: FGMRES(30) / BJacobi with ILU(0)
(28) 2nd order: BCGS / Bjacobi with ILU(0)
(66) 2nd order: FGMRES(30) / BJacobi with ILU(0)
(80) 2nd order: FGMRES(30) / BJacobi with ILU(1)

Convergence rates of base and adaptive methods

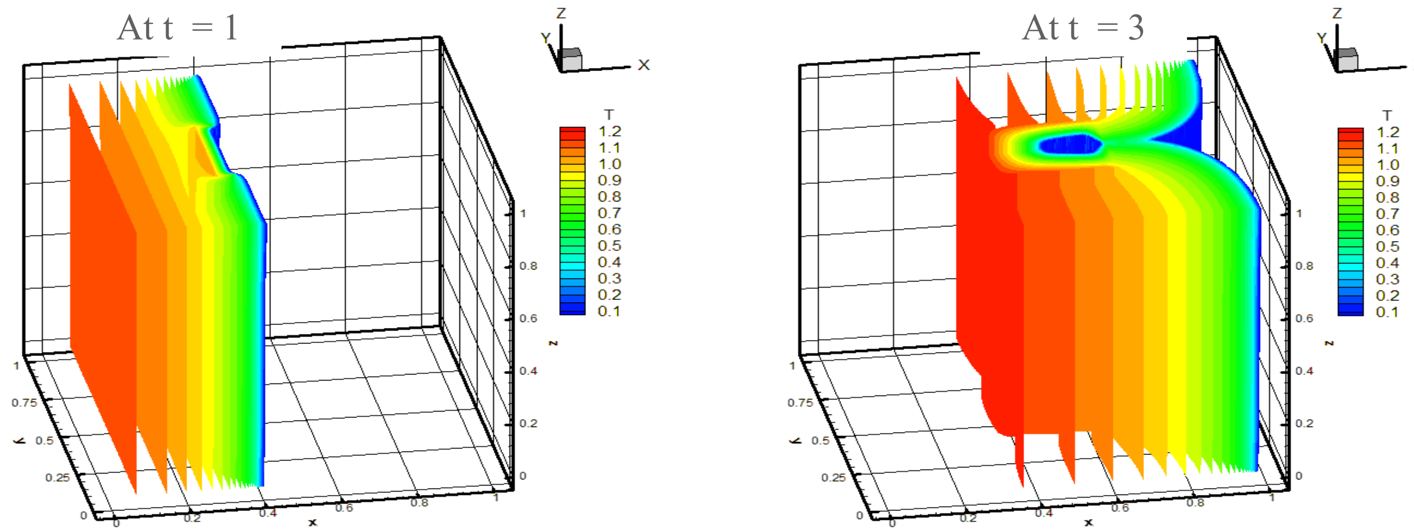


Cumulative time (seconds)

Adapt2: (1) 1st order: GMRES(30) / Bjacobi with SOR
(2) 1st order: BCGS / BJacobi with ILU(0)
(25) 1st order: FGMRES(30) / BJacobi with ILU(0)
(28) 2nd order: BCGS / Bjacobi with ILU(0)
(66) 2nd order: FGMRES(30) / BJacobi with ILU(0)
(80) 2nd order: FGMRES(30) / BJacobi with ILU(1)

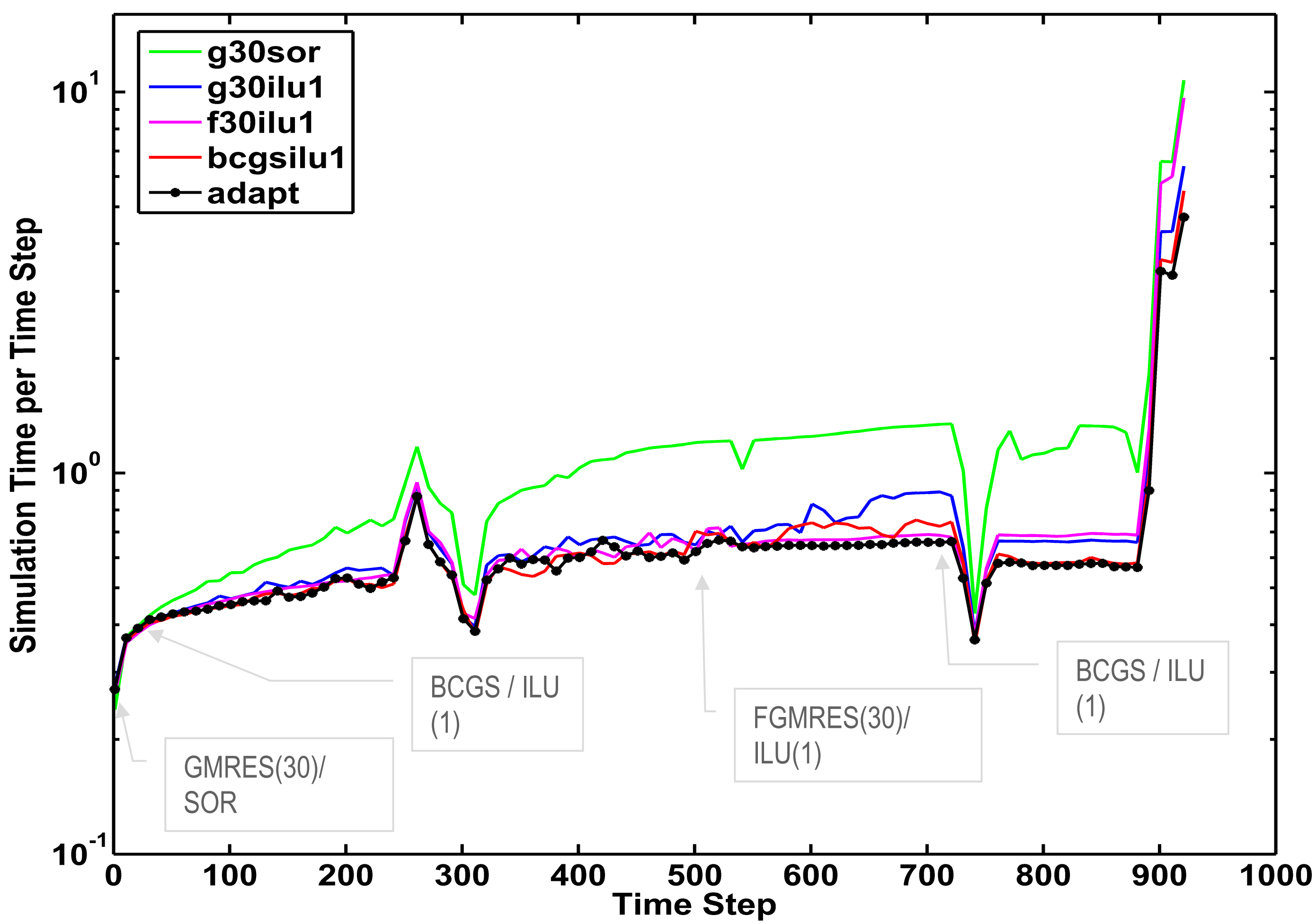
- Adaptive methods provide **reliable and robust** solution and reduced the number of nonlinear iterations and overall time to solution.

Example 3: Radiation Transport



- Based on Mousseau, Knoll, and Rider (LA-UR-99-4230)
- Govern the evolution of photon radiation in an optically thick medium
- Derived by integrating over all energy frequencies, assuming
  - Isotropy (angle dependence averaged out)
  - Small mean-free photon paths
- Very important in the simulation of forest fires, inertial confinement fusion (http://fusion.gat.com/icf), astrophysical phenomena

Example 3: Radiation Transport



Future directions

- Many possibilities for usability enhancements
- More automation
  - Component creation from existing code
  - CQoS
- Community contributions

Summary

- If defined and used properly, components are a powerful software development approach that enable diverse codes and developers collaborate effectively
  - 50% social interaction + 30% discipline + 20% code development