

Pyre: a distributed component framework

Michael Aivazis
Caltech

DANSE Developers Workshop
January 22-23, 2007

Overview

- ⊕ What is a distributed component framework?
 - ⊕ *what is a component?*
 - ⊕ *what is a framework?*
 - ⊕ *why be distributed?*
- ⊕ Why bother building a framework?
 - ⊕ *is it the solution to any **relevant** problem?*
 - ⊕ *is it the right solution?*
- ⊕ High level description of the specific solution provided by ***pyre***

Pyre overview

✦ Projects

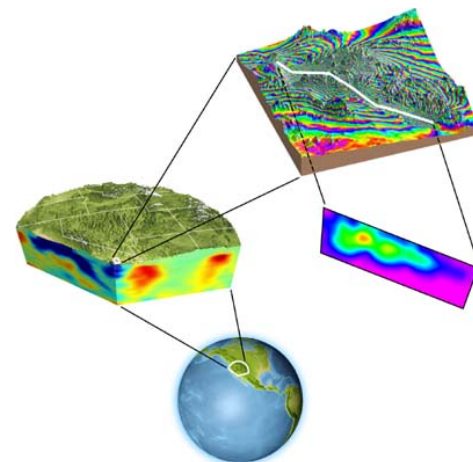
- ✦ *Caltech ASC Center (DOE)*
- ✦ *Computational Infrastructure in Geodynamics (NSF):*
- ✦ *DANSE (NSF)*

✦ Portability:

- ✦ *languages: C, C++, F77, F90*
- ✦ *compilers: all native compilers on supported platforms, gcc, Absoft, PGI*
- ✦ *platforms: all common Unix variants, OSX, Windows*

✦ Statistics:

- ✦ *1200 classes, 75,000 lines of Python, 30,000 lines of C++*
- ✦ *Largest run: **nirvana** at LANL, 1764 processors for 24 hrs, generated 1.5 Tb*



Flexibility through the use of scripting

- ✦ Scripting enables us to
 - ✦ *Organize the large number of simulation parameters*
 - ✦ *Allow the simulation environment to discover new capabilities without the need for recompilation or relinking*
- ✦ The python interpreter
 - ✦ *The interpreter*
 - ✦ *modern object oriented language*
 - ✦ *robust, portable, mature, well supported, well documented*
 - ✦ *easily extensible*
 - ✦ *rapid application development*
 - ✦ *Support for parallel programming*
 - ✦ *trivial embedding of the interpreter in an MPI compliant manner*
 - ✦ *a python interpreter on each compute node*
 - ✦ *MPI is fully integrated: bindings + OO layer*
 - ✦ *No measurable impact on either performance or scalability*

User stereotypes

- ✦ End-user
 - ✦ *occasional user of prepackaged and specialized analysis tools*
- ✦ Application author
 - ✦ *author of prepackaged specialized tools*
- ✦ Expert user
 - ✦ *investigator with a specific scientific goal*
- ✦ Domain expert
 - ✦ *author of analysis, modeling or simulation software*
- ✦ Software integrator
 - ✦ *responsible for extending software with new technology*
- ✦ Framework maintainer
 - ✦ *responsible for maintaining and extending the infrastructure*

Facilitating common tasks

Tasks performed by a typical user

Customize timestepping behavior

Access and control of the Grid data structures

Customize patch integrators

Customize output file formats

Customize initial / boundary conditions

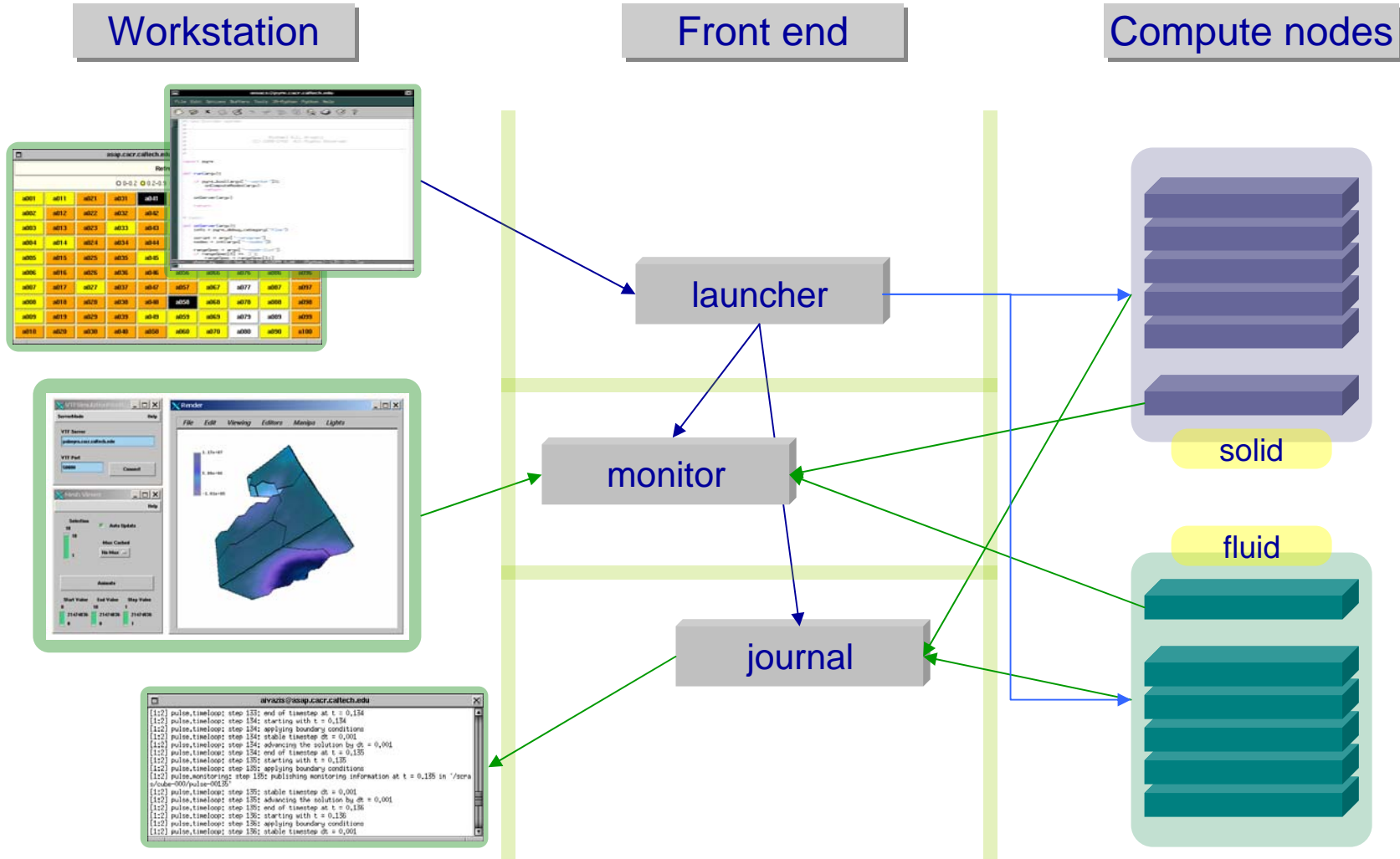
Customize refinement criteria

Run demo applications. Change problem parameters such as geometry, mesh sizes, output frequencies

Increasing sophistication / difficulty

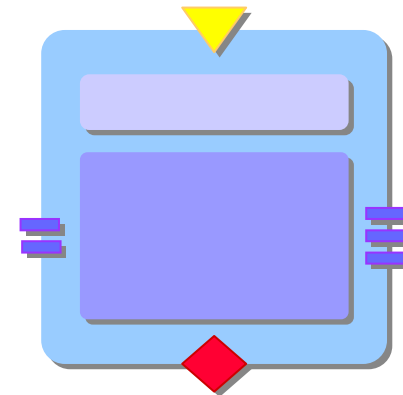
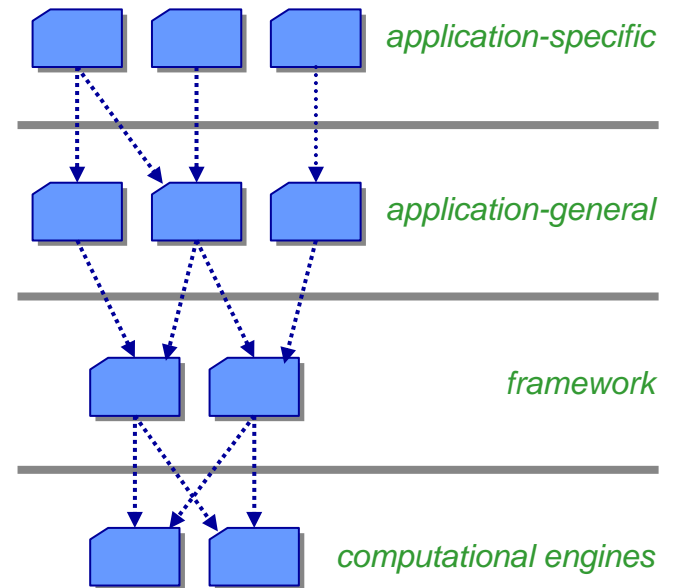
- ⊕ Interchangeable components; e.g. create and initialize a fluid mesh by
 - ⊕ *reading some geometry input description*
 - ⊕ *reading a checkpoint file*
 - ⊕ *invoking a user provided callback for setting initial conditions*

Distributed services

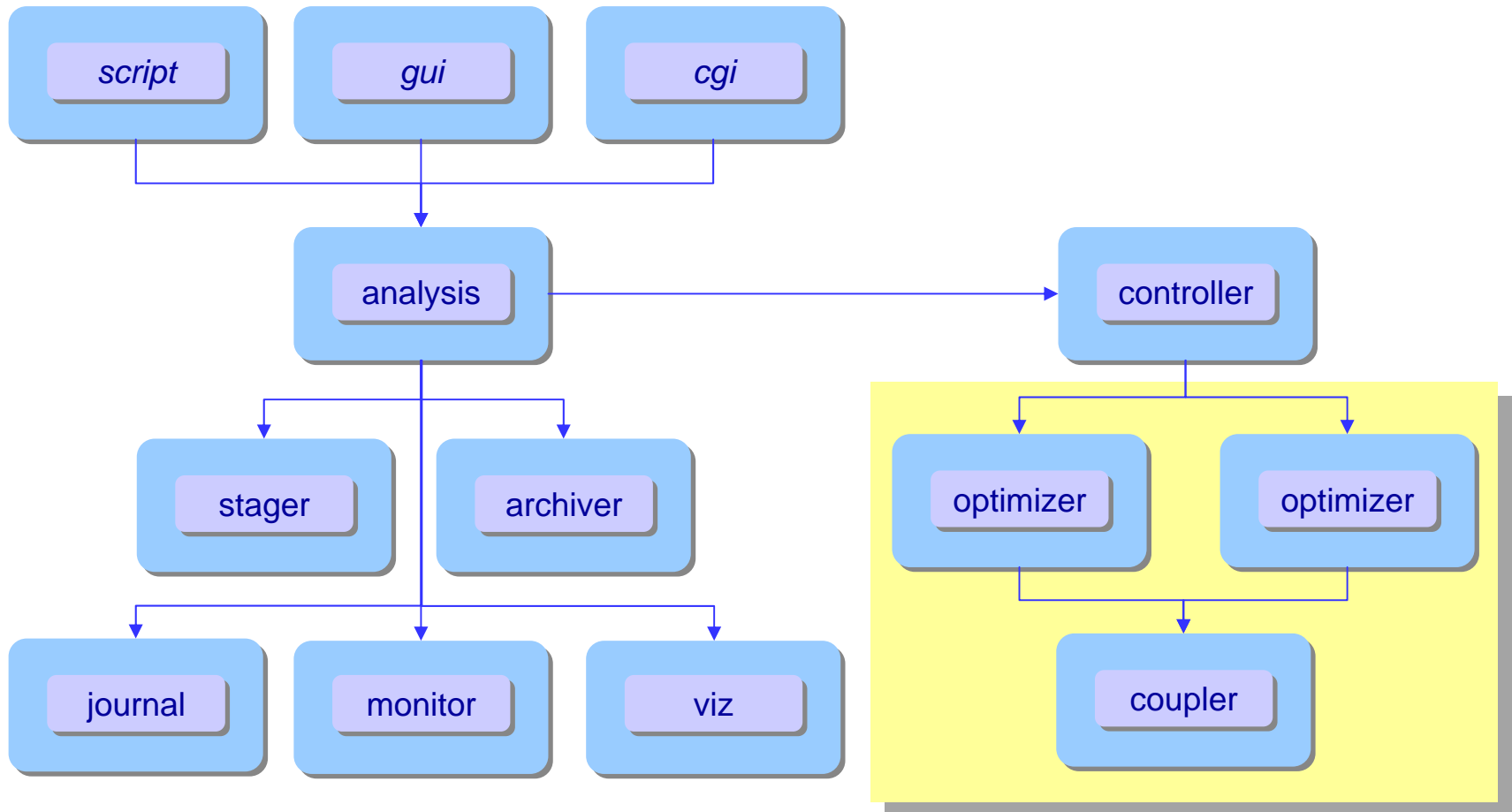


Pyre: the integration architecture

- ✦ Pyre is a *software architecture*:
 - ✦ a specification of the organization of the software system
 - ✦ a description of the crucial structural elements and their interfaces
 - ✦ a specification for the possible collaborations of these elements
 - ✦ a strategy for the composition of structural and behavioral elements
- ✦ Pyre is multi-layered
 - ✦ flexibility
 - ✦ complexity management
 - ✦ robustness under evolutionary pressures
- ✦ Pyre is a *component framework*

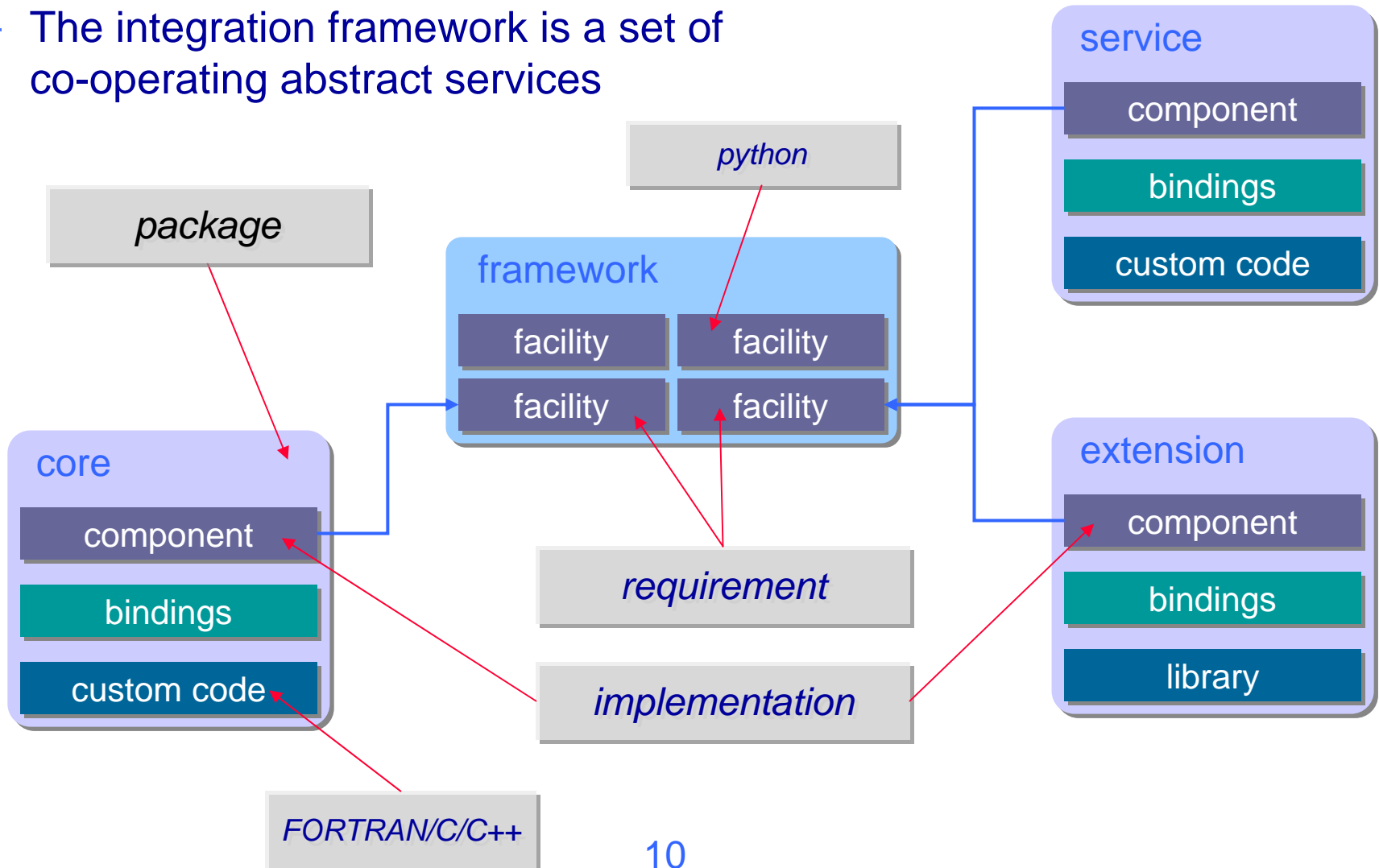


Example application



Component architecture

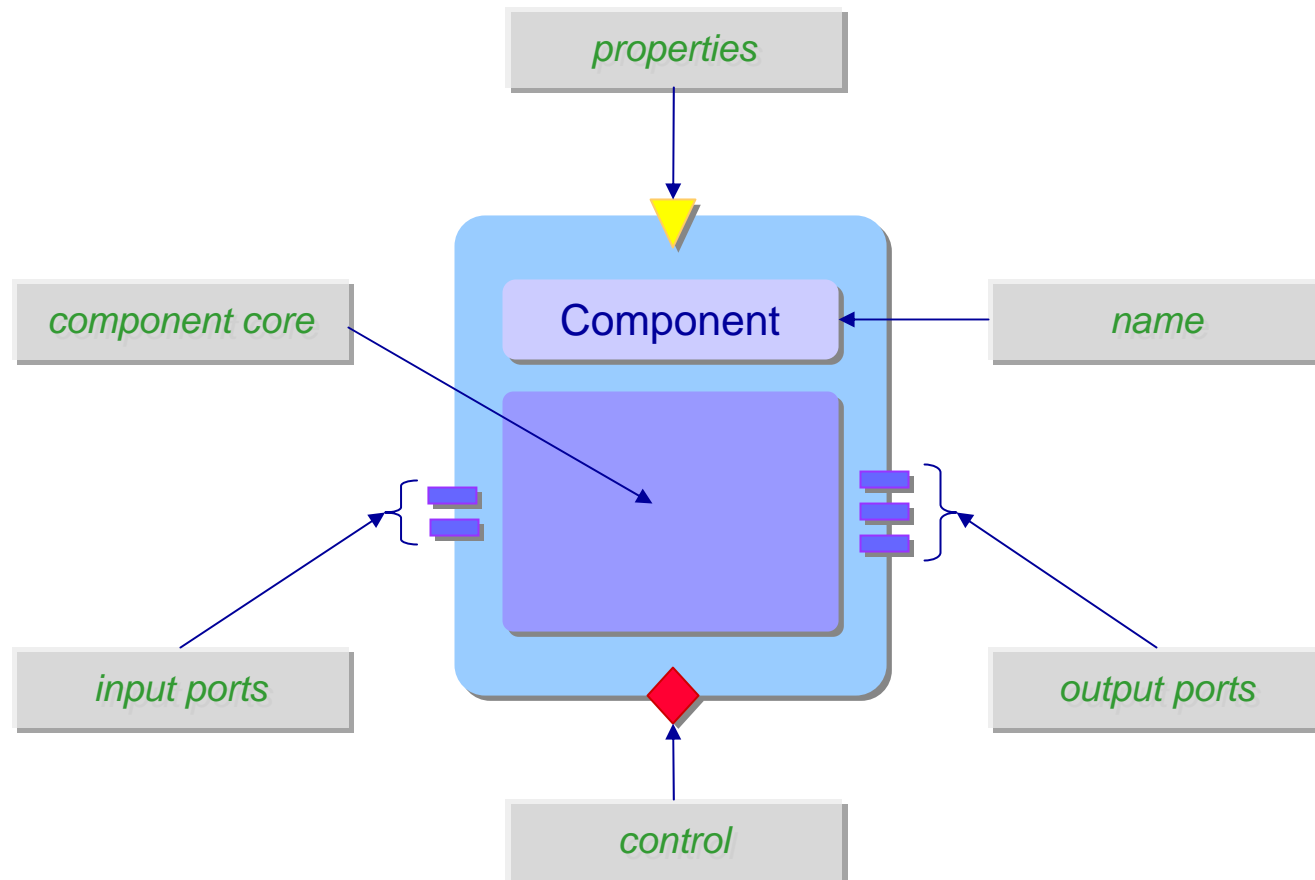
- ✦ The integration framework is a set of co-operating abstract services



Encapsulating critical technologies

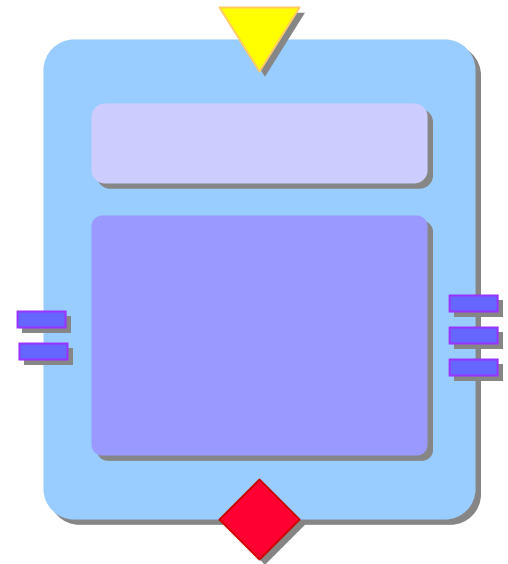
- ✦ Extensibility
 - ✦ *new algorithms and analysis engines*
 - ✦ *technologies and infrastructure*
- ✦ High-end computations
 - ✦ *visualization*
 - ✦ *easy access to large data sets*
 - ✦ *single runs, backgrounds, archived data*
 - ✦ *metadata*
 - ✦ *distributed computing*
 - ✦ *parallel computing*
- ✦ Flexibility:
 - ✦ *interactivity: web, GUI, scripts*
 - ✦ *must be able to debug almost everything on a laptop*

Component schematic



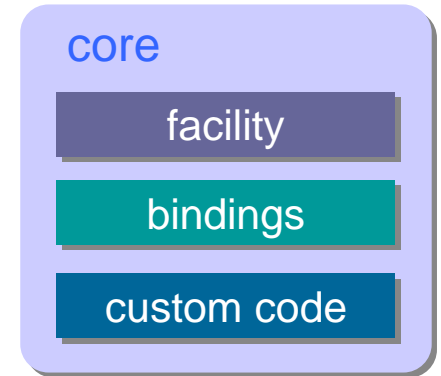
Component anatomy

- ✦ Core: encapsulation of computational engines
 - ✦ *middleware that manages the interaction between the framework and codes written in low level languages*
- ✦ Harness: an intermediary between a component's core and the external world
 - ✦ *framework services:*
 - ✦ *control*
 - ✦ *port deployment*
 - ✦ *core services:*
 - ✦ *deployment*
 - ✦ *launching*
 - ✦ *teardown*



Component core

- ✦ Three tier encapsulation of access to computational engines
 - ✦ *engine*
 - ✦ *bindings*
 - ✦ *facility implementation by extending abstract framework services*
- ✦ Cores enable the lowest integration level available
 - ✦ *suitable for integrating large codes that interact with one another by exchanging complex data structures*
 - ✦ *UI: text editor*

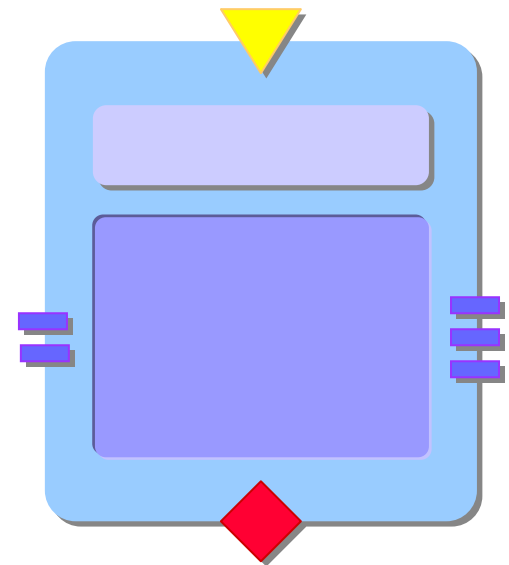


Computational engines

- ✦ Normal engine life cycle:
 - ✦ *deployment*
 - ✦ *staging, instantiation, static initialization, dynamic initialization, resource allocation*
 - ✦ *launching*
 - ✦ *input delivery, execution control, hauling of output*
 - ✦ *teardown*
 - ✦ *resource de-allocation, archiving, execution statistics*
- ✦ Exceptional events
 - ✦ *core dumps, resource allocation failures*
 - ✦ *diagnostics: errors, warnings, informational messages*
 - ✦ *monitoring: debugging information, self consistency checks*
- ✦ Distributed computing
- ✦ Parallel processing

Component harness

- ✦ The harness
 - ✦ *collects and delivers user configurable parameters*
 - ✦ *interacts with the data transport mechanisms*
 - ✦ *guides the core through the various stages of its lifecycle*
 - ✦ *provides monitoring services*
- ✦ Parallelism and distributed computing are achieved by specialized harness implementations
- ✦ The harness enables the second level of integration
 - ✦ *adding constraints makes code interaction more predictable*
 - ✦ *provides complete support for an application generic interface*



Support for concurrent applications

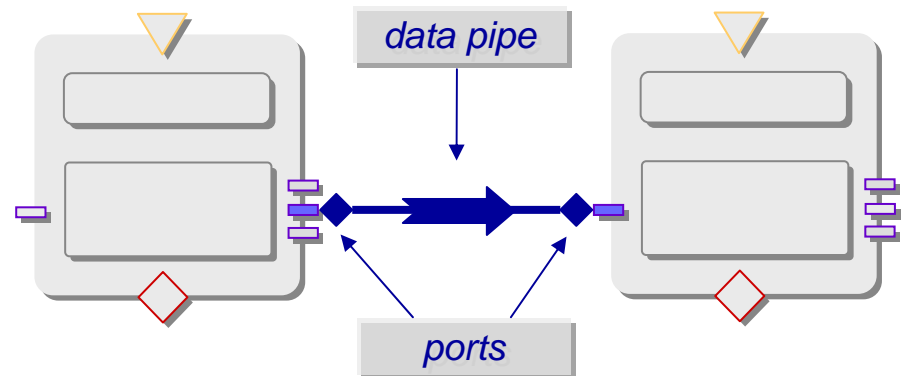
- ✦ Python as the driver for concurrent applications that
 - ✦ *are embarrassingly parallel*
 - ✦ *have custom communication strategies*
 - ✦ *sockets, ICE, shared memory*
- ✦ Excellent support for MPI
 - ✦ **mpipython.exe**: *MPI enabled interpreter (needed only on some platforms)*
 - ✦ **mpi**: *package with python bindings for MPI*
 - ✦ *support for staging and launching*
 - ✦ *communicator and processor group manipulation*
 - ✦ *support for exchanging python objects among processors*
 - ✦ **mpi.Application**: *support for launching and staging MPI applications*
 - ✦ *descendant of **pyre.application.Application***
 - ✦ *auto-detection of parallelism*
 - ✦ *fully configurable at runtime*
 - ✦ *used as a base class for user defined application classes*

Support for distributed computing

- ✦ We are in the process of migrating the existing support for distributed processing into `gs1`, a new package that completely encapsulates the middleware
- ✦ Provide both user space and grid-enabled solution
- ✦ User space:
 - ✦ *ssh, scp*
 - ✦ *pyre service factories and component management*
- ✦ Web services
 - ✦ *pyGridWare from Keith Jackson's group*
- ✦ Advanced features
 - ✦ *dynamic discovery for optimized deployment*
 - ✦ *reservation system for computational resources*

Ports and pipes

- ✦ Ports further enable the physical decoupling of components by encapsulating data exchange
- ✦ Runtime connectivity implies a two stage negotiation process
 - ✦ *when the connection is first established, the io ports exchange abstract descriptions of their requirements*
 - ✦ *appropriate encoding and decoding takes place during data flow*
- ✦ Pipes are data transport mechanisms chosen for efficiency
 - ✦ *intra-process or inter-process*
 - ✦ *components need not be aware of the location of their neighbors*
- ✦ Standardized data types obviate the need for a complicated runtime typing system
 - ✦ *meta-data in a format that is easy to parse (XML)*
 - ✦ *tables*
 - ✦ *histograms*



Component implementation strategy

- ✦ Write engine
 - ✦ *custom code, third party libraries*
 - ✦ *modularize by providing explicit support for life cycle management*
 - ✦ *implement handling of exceptional events*
- ✦ Construct python bindings
 - ✦ *select entry points to expose*
- ✦ Integrate into framework
 - ✦ *construct object oriented veneer*
 - ✦ *extend and leverage framework services*
- ✦ Cast as a component
 - ✦ *provide object that implements component interface*
 - ✦ *describe user configurable parameters*
 - ✦ *provide meta data that specify the IO port characteristics*
 - ✦ *code custom conversions from standard data streams into lower level data structures*
- ✦ All steps are well localized!

Cost/benefit

+ Drawbacks

- + *some reengineering required*
- + *paradigm shift*
- + *learning curve – not helped by the lack of documentation...*

+ Benefits

- + *clear path forward for “legacy” applications*
- + *easy, normalized access to large number of facilities*
- + *structured way for enabling engines in modern computational environments*
- + *rigorous separation of UI from computational engines*
- + *easy re-hosting of compliant application*

Status update

- ✦ Database access
 - ✦ *backend access*
 - ✦ *data types*
 - ✦ *SQL queries*
- ✦ Application hosting (user interfaces)
 - ✦ *GUI*
 - ✦ *web portals*
 - ✦ *web services*
- ✦ Distributed services
 - ✦ *semi-asynchronous, fully asynchronous*
 - ✦ *authentication, GUIDs, session management, monitoring*
 - ✦ *distributed control layer*
 - ✦ *“steering”*

Wrap up

- ✦ Expect **pyre 1.0** early 2007
 - ✦ *largely a documentation effort*
 - ✦ *minor re-design of some internals*
 - ✦ *re-examination of the component-inventory coupling*
- ✦ There is a lot of material on the web
 - ✦ *under extensive reorganization*
 - ✦ *currently at <http://www.cacr.caltech.edu/projects/pyre>*
 - ✦ *soon to be at <http://pyre.caltech.edu>*
- ✦ Contact info
 - ✦ aivazis@caltech.edu
 - ✦ pyre@cacr.caltech.edu