An Overview of the Morfeus Project

Damian Rouson
Sandia National Laboratories

Joel Koplik, Karla Morris, Xiaofeng Xu
City University of New York

Sponsors:
U.S. Office of Naval Research
U.S. Department of Energy (OASCR)

Sandia National Laboratories

Outline

- Introduction
- Methodology
- Results
- Conclusion

Sandia National Laboratories

Outline

- Introduction
  - Motivation
  - Objectives
  - Previous work
- Methodology
- Results
- Conclusion

Sandia National Laboratories

Motivation

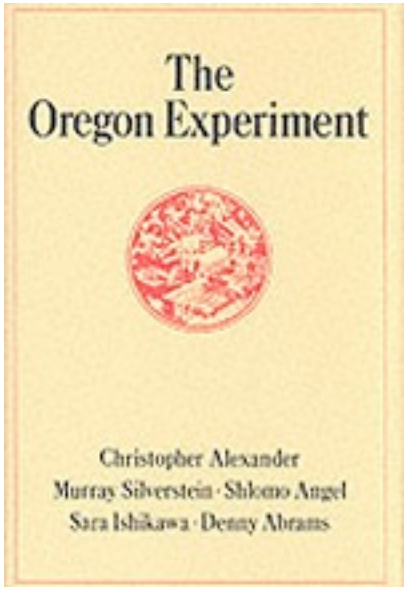
- Published OOD patterns in scientific programming are rare – especially in Fortran 2003/2008.
- Conventional wisdom suggests that the higher-level abstractions characteristic of OOP impose performance penalties.

Objectives

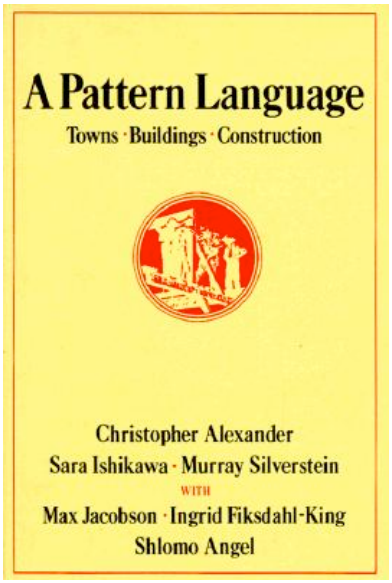
- To move scientific programmers to higher-level, platform-agnostic yet scalable abstractions.
- To demonstrate general OOD patterns & distill new domain-specific patterns from multiphysics applications in Fortran.
- To construct an open-source framework that encourages the use of the demonstrated patterns.

Previous Work: Patterns

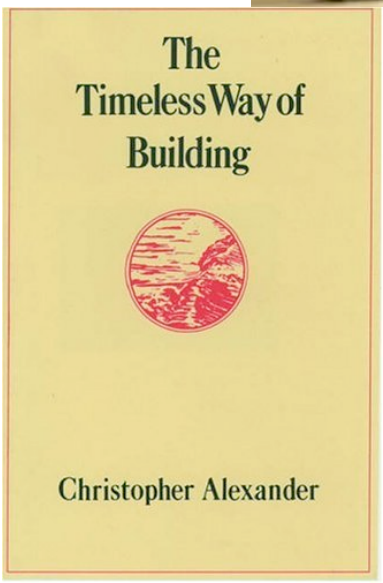
Building architecture: Alexander et al.



Vol. III (1975)



Vol. II (1977)



Vol. I (1979)

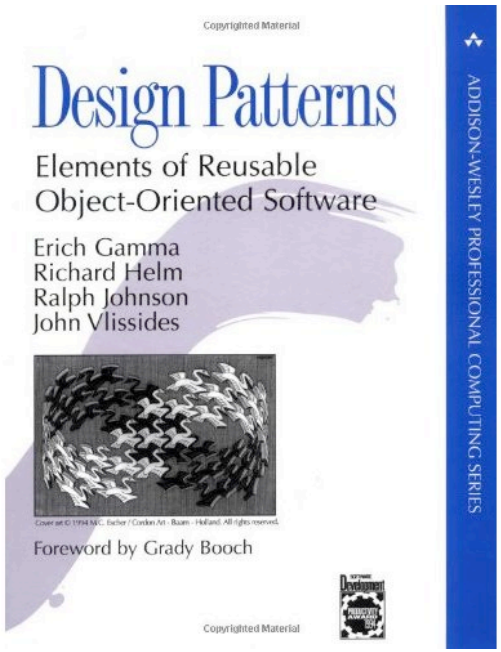
Positive Outdoor Space
Julian Street Inn
Shelter for the Homeless
San Jose, CA

Sandia National Laboratories

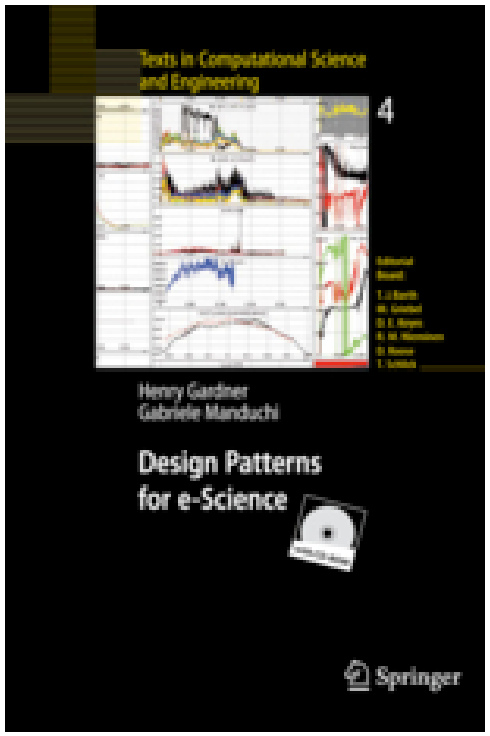
Previous Work: OOD Patterns

Software architecture: Gamma et al. (1995)

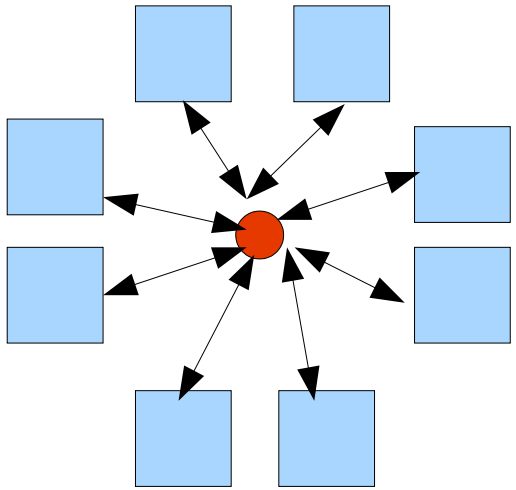
Scientific software architecture:
Gardner & Manduchi (2007)



1995



2007

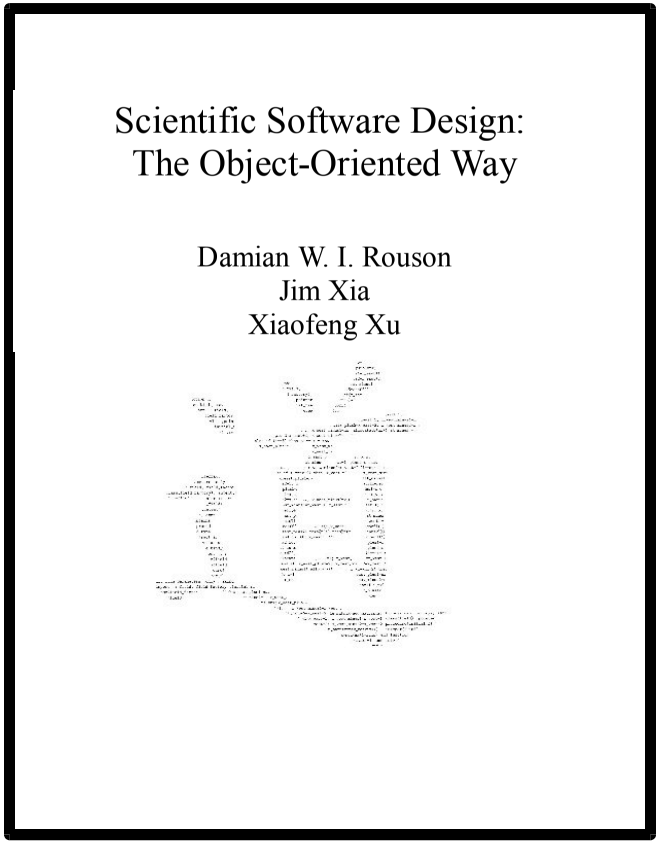


Mediator

Sandia National Laboratories

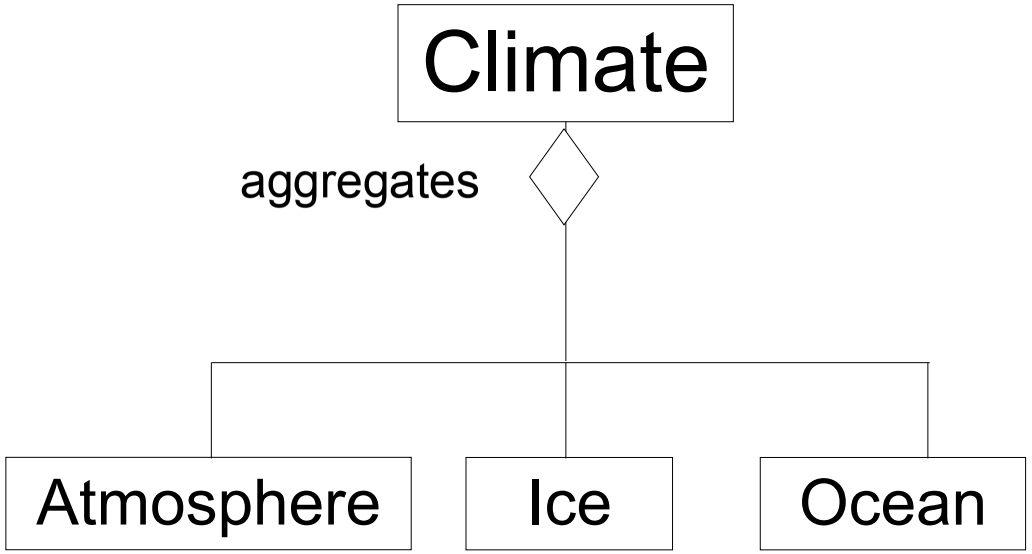
Previous Work: Patterns in Fortran

- F95: Decyk & Gardner (2006-'07)
- F03: Markus (2008)
- F03: Rouson, Adalsteinsson & Xia (2010)



2011

Puppeteer:



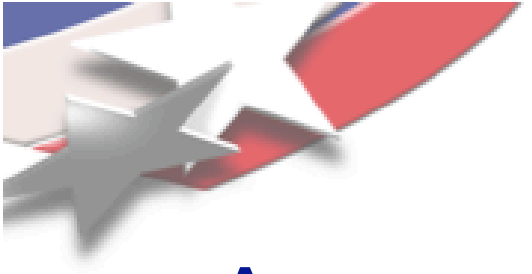
Sandia National Laboratories

Outline

- Introduction
- Methodology
  - Language selection
  - Pattern definition
- Results
- Conclusion

Why Fortran?

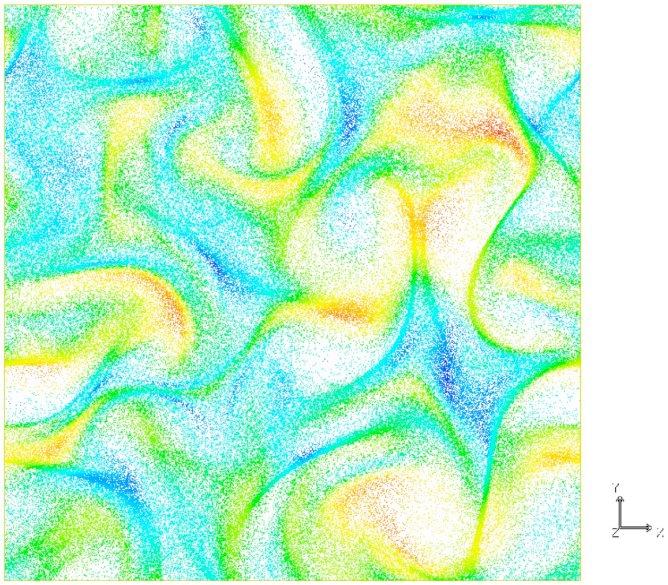
- Scientific programmers rule!
- Mathematical expressiveness:
  - User-defined operators:
    `.div.`, `.grad.`, `.curl.`, etc.
  - Multidimensional arrays
  - Array operations
- Platform-agnostic parallelism (Fortran 2008)
- Automatic memory management:
  - Automatic re-sizing of arrays
  - Automatic destruction of dynamically allocated arrays/objects.



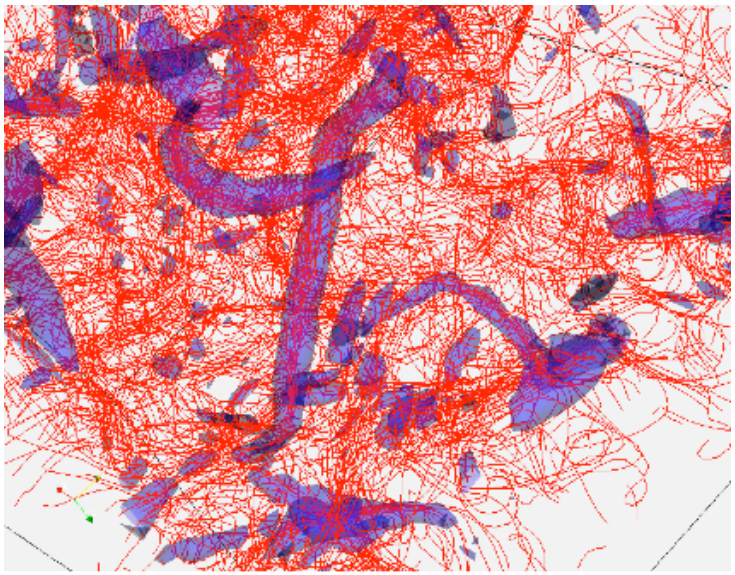
What's a pattern?

- A common solution to a recurring OOD problem.
- Four essential elements:
  - The name
  - The problem
  - The solution
  - The consequences
- Additional elements:
  - Also known as...
  - Known uses.
  - Related patterns.
  - Sample implementation.

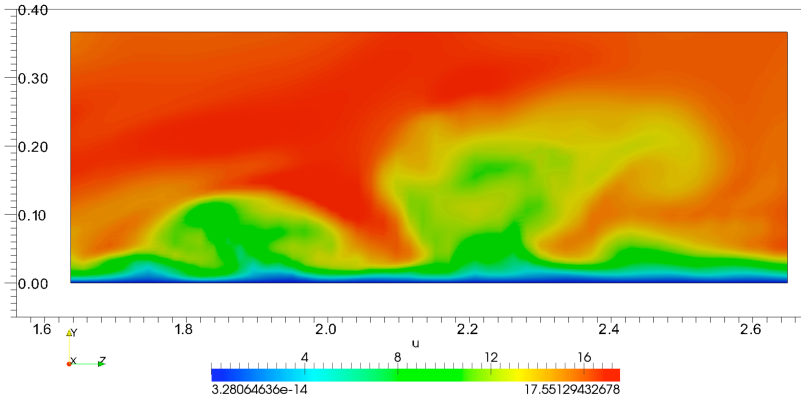
Multiphysics with Morfeus



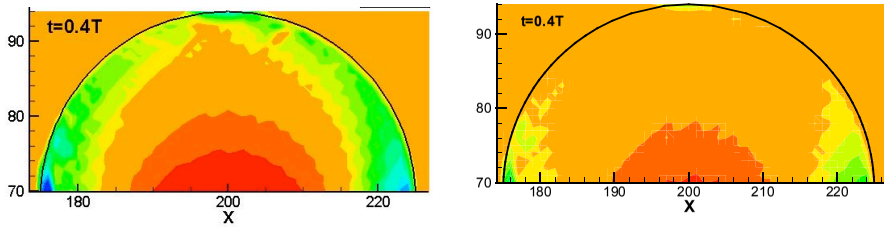
Particles in liquid metal MHD (red=fastest, blue=slowest) [Rouson et al., *PoF* 2008]



Quantum vortices (red) & classical vortices (blue) in superfluid helium. [Morr is et al., *PRL* 2008]



Optical turbulence in the ABL. [Morris et al., *JoT* sub. 2010]



Lattice Boltzman blood flow with & without stent. [Xu & Lee *IJNMF* 2008]

Software Stack

Concrete classes.

Abstract class framework.

} Procedural bindings

Distributed C++ objects

Trilinos

- Heroux et al. (1995) “An Overview of the Trilinos Project,” *ACM TOMS*.
- Over 50 packages: linear, nonlinear, & eigensolvers; optimization; automatic differentiation; load balancing,...
- Establishes consistent, professional software engineering practices for over 50 packages:
  - Automated nightly multi-platform build (CMake)
  - Automated test/notification/dashboard (CDash)
  - State-of-the-art repository (Git)
  - Mailing lists (Mailman)
  - Web-based issue tracking (Bugzilla)
  - Automated documentation (Doxygen)

Outline

- Introduction
- Methodology
- Results
  - Abstract Calculus
  - Abstract Factory & Factory Method
  - Object & Surrogate
- Conclusion

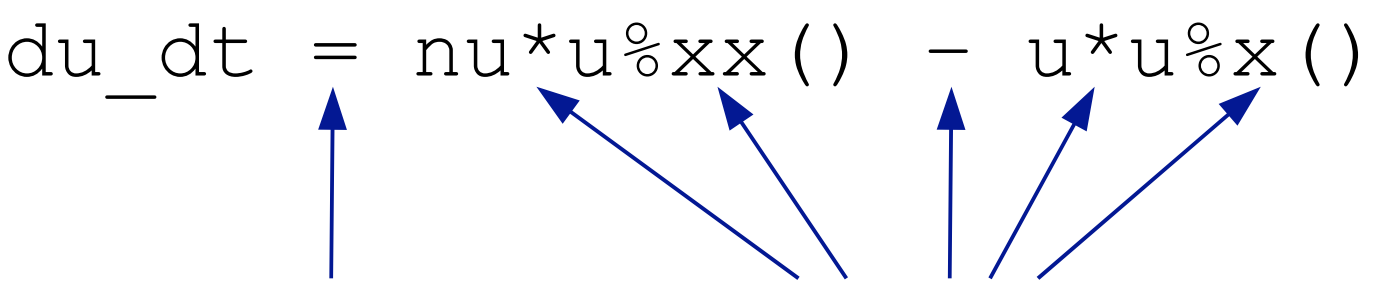
Abstract Calculus Pattern

"Software abstractions should resemble blackboard abstractions."
Kevin Long, Texas Tech. U.

Blackboard abstraction	Software abstraction
$u = u(x,t)$	`class(field),pointer::u,du_dt`
$u(x=0,t)=u_o$	`call u%boundary(x,0,u0)`
$u_t \equiv \partial u / \partial t$	`u%t()`
$u^{n+1} = u^n + u_t^n \Delta t$	`u = u + u%t()*dt`
$u_t = \nu u_{xx} - u u_x$	`du_dt = nu*u%xx() - u*u%x()`

Sandia National Laboratories

Scalability



Synchronization

Purely functional operators and methods.

⇒ Highly asynchronous.

⇒ Blurs the boundary between task & data parallelism.

Factory Patterns

The problem:

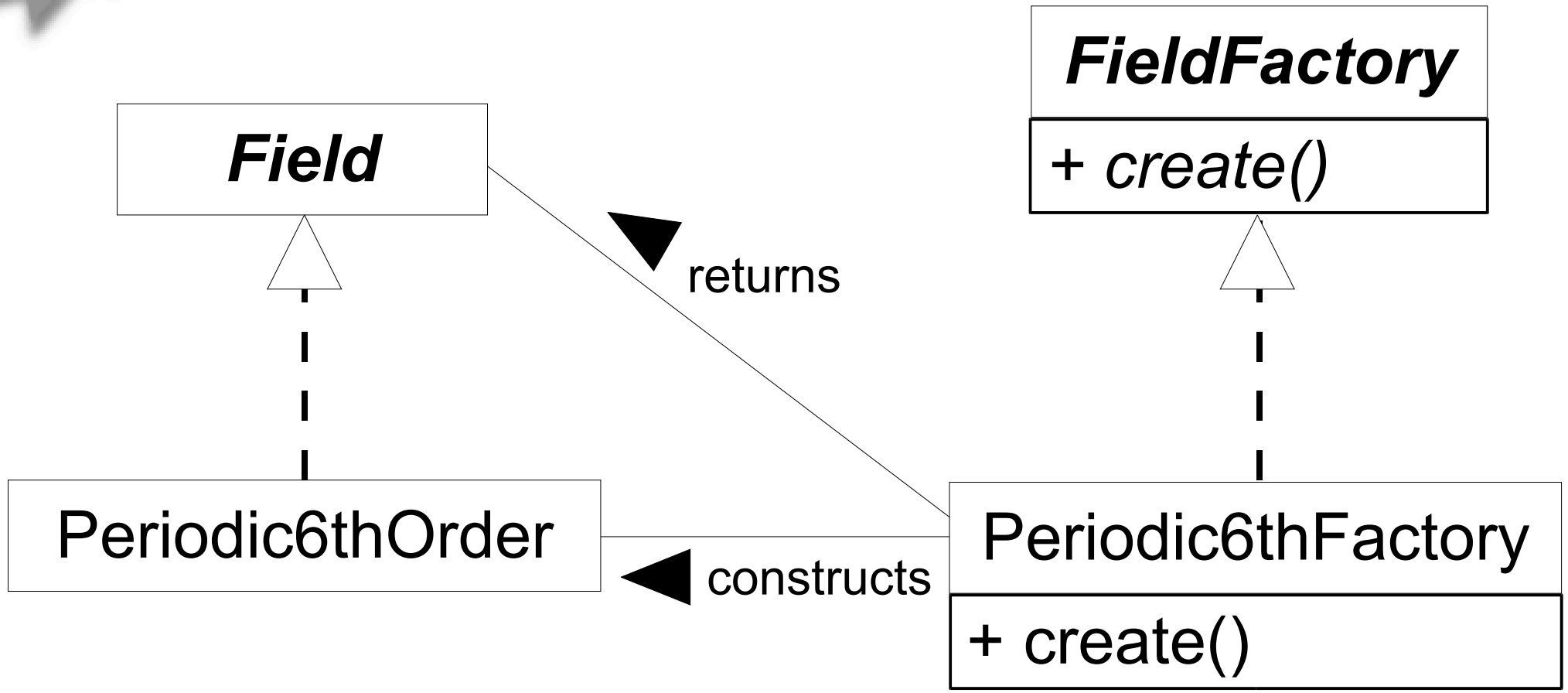
- GoF Philosophy:
  "Program to an interface, not an implementation."
- Abstract classes model interfaces.
- Opposing force:
  Abstract classes cannot be instantiated.

Factory Patterns

- The Abstract Factory solution:
  - “Provide an interface for creating families of related or dependent objects without specifying their concrete classes.” GoF

- The Factory Method solution:
  - “Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses.” GoF

Class Diagram



Legend

"Implements" – –▷

Public +

Abstract Type

deferred_binding()

Sandia National Laboratories

Trilinos-Based Client

```
program main
!...
#ifdef HAVE_MPI
  type(Epetra_MpiComm) :: comm
#else
  type(Epetra_SerialComm) :: comm
#endif
  class(field), pointer :: u
  class(field), pointer :: u
  class(field_factory), allocatable :: field_creator
  allocate(periodic_6th_factory::field_creator)
```

Trilinos-Based Client

```
#ifdef HAVE_MPI
  call MPI_INIT(ierr)
  comm = Epetra_MpiComm(MPI_COMM_WORLD)
#else
  comm = Epetra_SerialComm()
#endif
  initial => u_initial
  u => field_creator%create(initial,resolution)
!..
  do while t < t_final
    dt = u%euler_step(nu,grid_resolution)
    u  = u + (nu*u%xx()*nu - u*u%x())*dt
    t = t + dt
  end do
#ifdef HAVE_MPI
  call MPI_FINALIZE(rc)
#endif
  call u%force_finalize
end program
```

Object Pattern

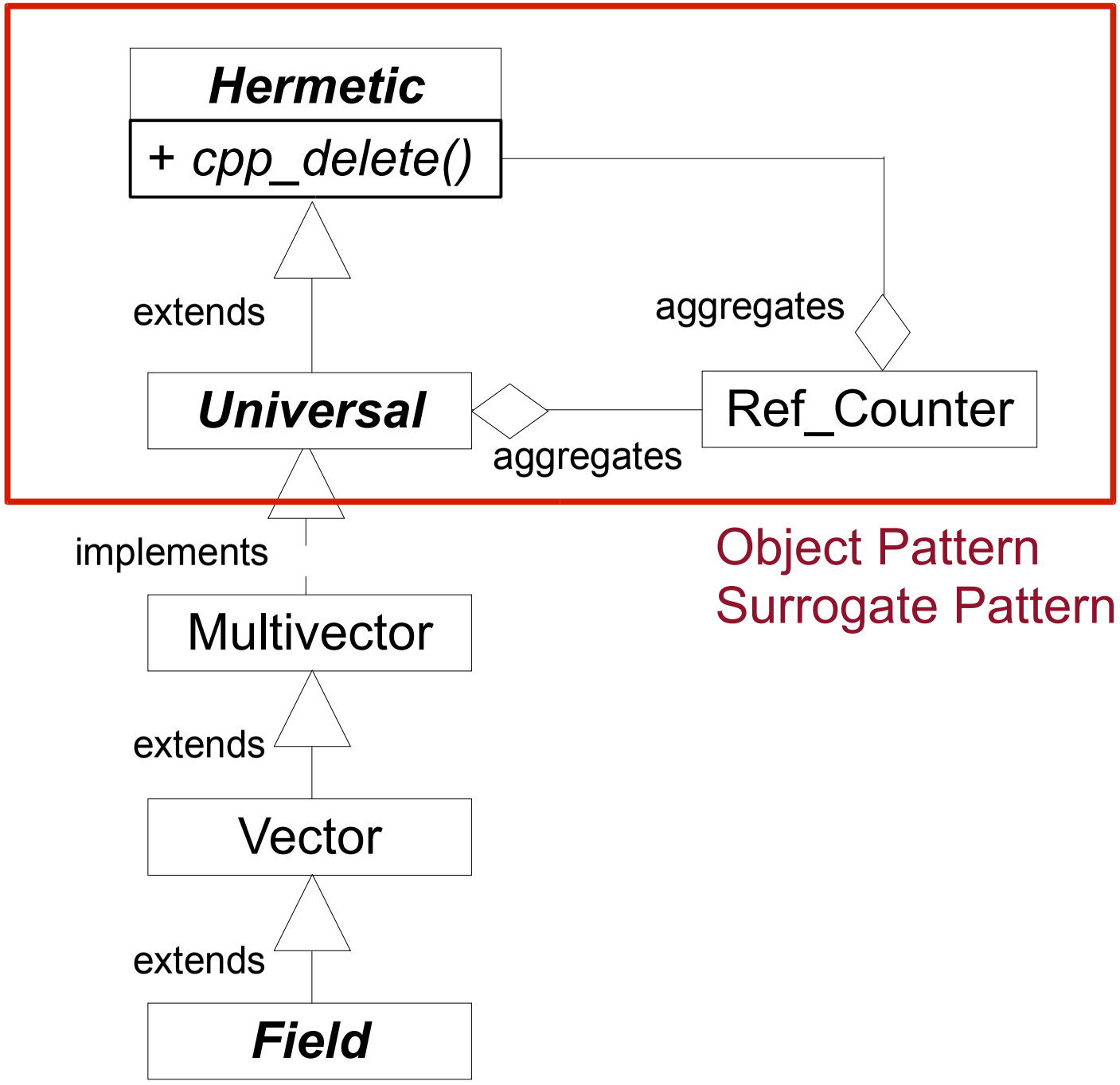
The problem:

- Robust design requires a degree of uniformity:
  - Consistent desirable behavior, e.g. leak-free memory management.
  - Universal referencing.
- Opposing force:
  - Too much uniformity feels like handcuffs.

Object Pattern

- The solution:
  - Define a universal parent class hierarchy from which every class in the package inherits the desired behavior/interface.
  - Use the Object class hierachy to ensure low-level services, e.g., those likely to be considered as candidates for future inclusion in the language.

Class Model



Object Pattern
Surrogate Pattern

Sandia National Laboratories

Outline

- Introduction
- Methodology
- Results
- Conclusion
  - Summary
  - Future work
  - Acknowledgements

Sandia National Laboratories

Summary

- Abstract Calculus illuminates a path toward highly asynchronous computing that blurs the task/data parallel distinction
- Fortran 2003 appears to have the expressiveness to support the general GoF design patterns in multiphysics applications.
- Several domain-specific and language-specific patterns emerge along the way.

Future Work

- Implement an Abstract Calculus based on Fortran 2008 coarray PGAS
- Unit testing
- Performance testing
- Public release:
  - http://trilinos.sandia.gov
- Demonstrate a scalable OO multiphysics solver in Fortran 2003/2008.

Acknowledgements

- Sandia National Laboratories:
  - Mike Heroux, Nicole Lemaster
- IBM:
  - Jim Xia

Sandia National Laboratories