ForTrilinos Tutorial

Damian Rouson, Nicole Lemaster
Sandia National Laboratories

Karla Morris, Xiaofeng Xu
City University of New York

Salvatore Filippone
University of Rome

Jim Xia
IBM

Sponsors: DOE/OASCR, ONR

November 2, 2010
SAND2010-7764C
Unlimited Release

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Outline

- Motivation and objectives
- Methodology
  - Shadow object interface (Gray et al. 1999)
  - ForTrilinos application software stack
- Application prototype
  - Simple main
  - Prototypical PDE solver

Motivation

Fortran Apps Currently Use Custom Wrappers for Trilinos

- High Order Method Modeling Environment (HOMME) atmospheric dynamics solver
- Parallel Ocean Program (POP)
- Glimmer ice sheet model
- Multiphase Flow with Interphase eXchanges (MFIX)

Customized Procedural Interface

Advantages

- Gets the job done
- Does not required extensive lines of additional code

Disadvantages

- Requires flattening the data (reduced to intrinsic types & 1D arrays thereof)
- Requires flattening the functions that act on the data
- Requires hardwiring assumptions into receiving code
- Increases software complexity
- Low portability, reusability & maintainability

Interface Construction Approach

Specific Language Pairing

- Simplified Wrapper and Interface Generator (SWIG):
  - Parses C/C++ code to create bindings in scripting language
  - Interfaces C/C++ code with high level languages
  - No Fortran interoperability

Scientific Interface Definition Language (SIDL)

- Babel
  - Developer must define application programming interfaces (API) for given source code
  - Generates code stubs for code interoperability
  - No extensible derived types in Fortran 2003

Interface Construction Approach (cont.)

- Chasm:
  - Parses C/C++ and Fortran source code to generate its XML description
  - Uses XML stylesheet transformation (XSLT) to create binding code
  - No Fortran 2003 support

- Open Fortran Parser:
  - Parses Fortran 2003 code and automatically generates bindings for Fortran and C
  - No C++ support

Objectives

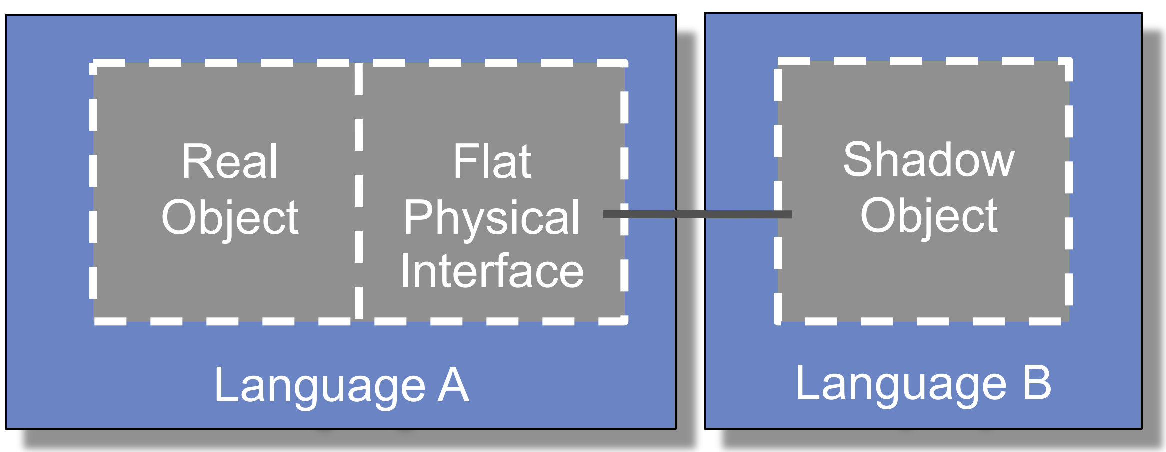
- To increase the adoption of Trilinos throughout DOE research communities that principally write Fortran, e.g. climate & combustion researchers.
- To maintain the OOP philosophy of the Trilinos project while using idioms that feel natural to Fortran programmers.

Outline

- Motivation and objectives
- Methodology
  - Shadow object interface (Gray et al. 1999)
  - ForTrilinos application software stack
- Application prototype
  - Sample main
  - Prototypical PDE solver

Gray's Shadow Object Interface

- Interface OO C++ and object based Fortran 95
- Flat interface exports real object behavior
- Shadow object is a logical interface, can be treated as a native object



Gray's Shadow Object Interface (cont.)

Constructing a physical interface requires:

- Unmangling procedure names
  - Limit C++ physical interfaces to extern "C"{}-bracketed C++ procedures with lower case names and Fortran external procedure
- Flattening interfaces with interoperable built-in data types
  - Pass only matching intrinsic data types
  - C++ arguments are passed as instances of an opaque pointer class template
- Ensuring code initialization
  - C/C++ `extern const` and `static` objects
  - Fortran `save` attribute objects

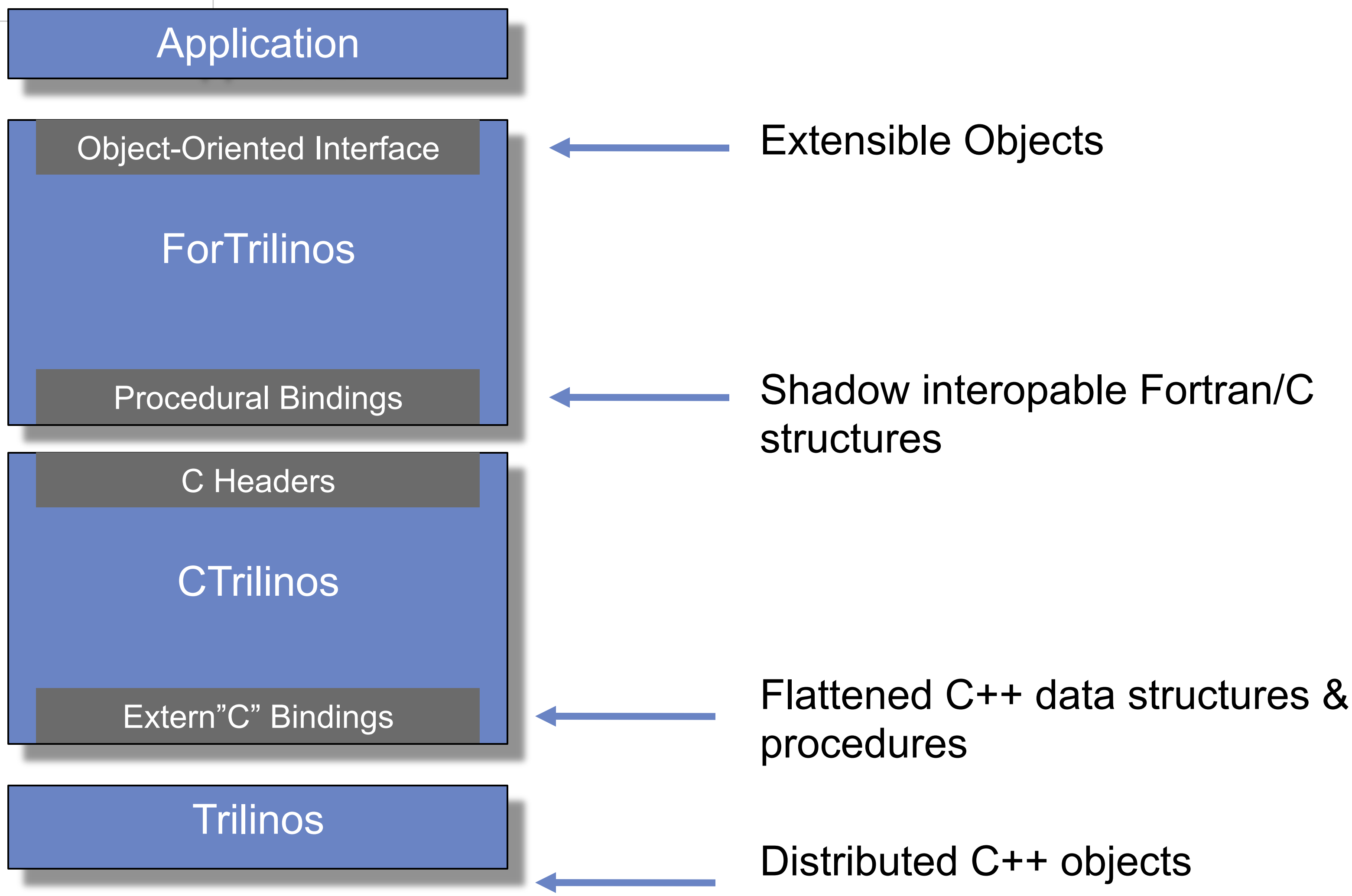
ForTrilinos/CTrilinos Shadow Object Interface

- No type-system ambiguity
- No case restrictions on C++ and the external procedures on Fortran
- No required passing by reference
- No OOP limitation

Features of Fortran 2003

- Object Oriented Support
  - Inheritance
  - Polymorphism (static and dynamic)
  - Encapsulation & information hiding
  - Interfaces (abstract derived types)
- C interoperability
- Enumerators

Application Software Stack



Sample Fortran Main

```
program main
  use iso_c_binding, only : c_double
  use Fepetra_MpiComm, only : epetra_mpicomm
  use FEpetra_Map, only : epetra_map
  use FEpetra_Vector, only : epetra_vector
  implicit none
  type(epetra_mpicomm) :: comm
  type(epetra_map) :: map
  type(epetra_vector) :: b
  ! MPI startup lines ommitted
  comm = epetra_mpicomm()
  map = epetra_map(numGlobalElements=64,IndexBase=1,comm=comm)
  b = epetra_vector(map)
  call b%PutScalar(2.0_c_double)
  print *, "L2 norm of b = ",b%Norm2()
  ! MPI shutdown lines ommitted
end program
```

Sandia National Laboratories

Procedural Bindings

- Trilinos `Epetra_MultiVector` object

```
int Random();
```

- Ctrilinos exports:
  - C wrapper prototype

```
int Epetra_MultiVector_Random(CT_Epetra_MultiVector_ID_t
  selfID);
```

  - Corresponding Fortran interface block

```
interface
  integer(c_int) function Epetra_MultiVector_Random(selfID) &
  bind(C,name='Epetra_MultiVector_Random')
  import :: c_int ,FT_Epetra_MultiVector_ID_t
  type(FT_Epetra_MultiVector_ID_t), intent(in), value :: selfID
  end function
end interface
```

Procedural Bindings (cont.)

- ForTrilinos derived type definition

```
type, bind(C) :: FT_Epetra_MultiVector_ID_t
  private
  integer(ForTrilinos_Table_ID_t) :: table
  integer(c_int) :: index
  integer(FT_boolean_t) :: is_const
end type
```

- CTrilinos corresponding C struct

```
typedef struct {
  CTrilinos_Table_ID_t table;/*Table holding object reference*/
  int index;                 /*Array index of the object*/
  boolean is_const;          /*Whether object was declared const*/
}CT_Epetra_MultiVector_ID_t;petra_MultiVector_ID_t;
```

Procedural Bindings

```
module ForTrilinos_enums
  use iso_c_binding ,only : c_int
  enum ,bind(C) enumerator ::
    FT_Epetra_SerialComm_ID, &
    FT_Epetra_Comm_ID, &
    FT_Epetra_MpiComm_ID
  end enum
  integer(kind(c_int)) ,parameter :: ForTrilinos_Type_ID_t= c_int
  integer(kind(c_int)) ,parameter :: FT_boolean_t = c_int
  integer(FT_boolean_t) ,parameter :: FT_FALSE = 0
  integer(FT_boolean_t) ,parameter :: FT_TRUE = 1
  type ,bind(C) :: ForTrilinos_Object_ID_t
    integer(ForTrilinos_Type_ID_t) :: table ! Object data type
    integer(c_int) :: index ! Array index
    integer(FT_boolean_t) :: is_const ! Const status
   end type
! Additional structurally equivalent types…
end module
```

CTrilinos

Capabilities not supported by C:

- Object method invocation
- Inheritance
- Polymorphism
- Overloaded function names and operators
- Default argument values
- Class and function templates
- String and bool types
- Exception handling
- Safe type-casting

CTrilinos

Invoking object methods from C:

- C++ Trilinos code

```
Epetra_MultiVector mv(...);

mv.Random();
```

- C wrapper possible code

```
Code in C using raw pointers ...
```

STOP EXTREMELY DANGEROUS MANEUVER

STOP EXTREMELY DANGEROUS MANEUVER

STOP EXTREMELY DANGEROUS MANEUVER

- Ctrilinos wrapper

```
Epetra_MultiVector_Random(mv_id);
```

Sandia National Laboratories

CTrilinos

Object Construction:

- Type-specific struct ID

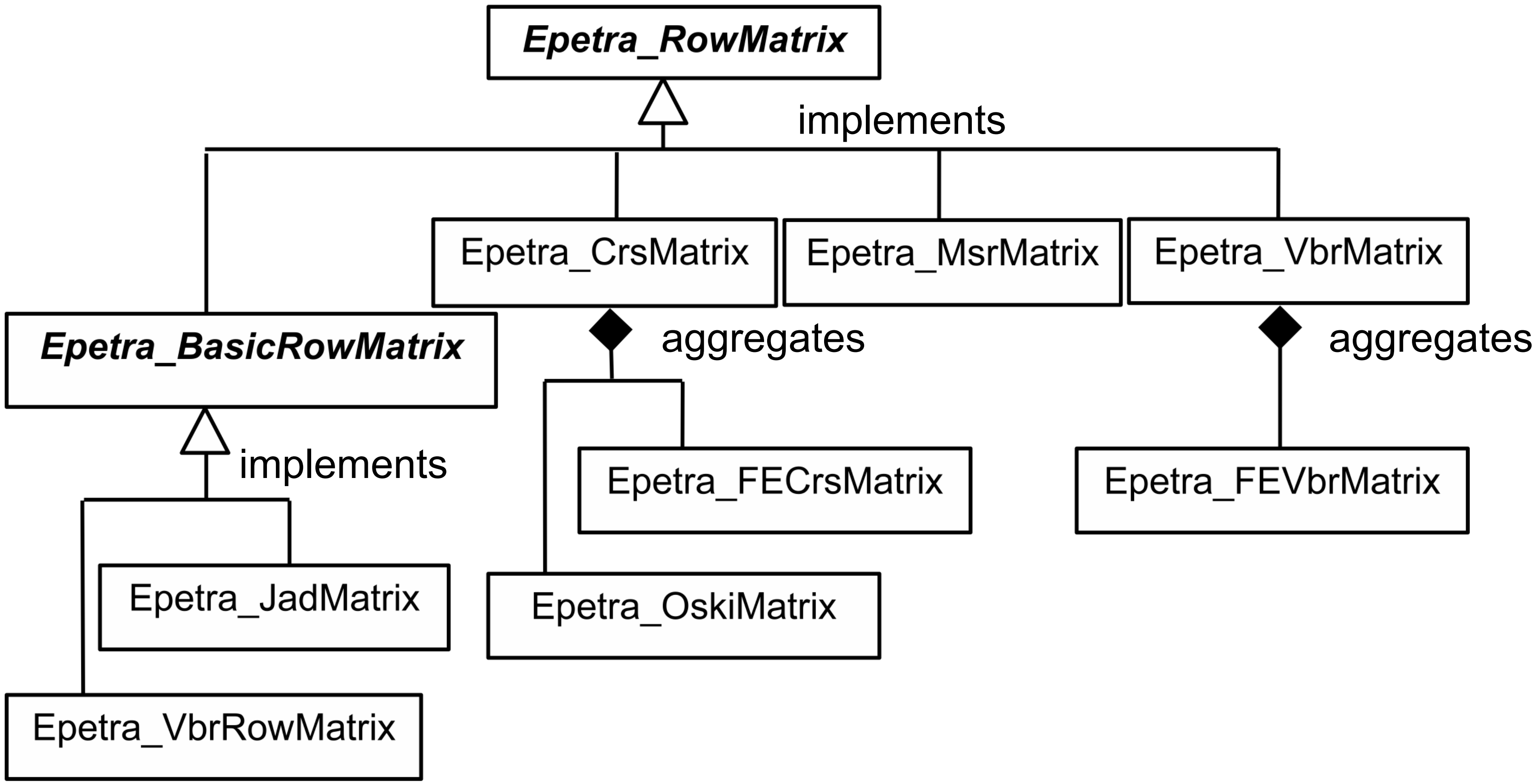
```
typedef struct {
  CTrilinos_Table_ID_t table;
  int index
  boolean is_const;
} CTrilinos_Universal_ID_t;
```

  - table: indentifies table holding reference to the object
  - index: entry in table for specific object
  - is_const: states if the object is constant or not

Object Destruction:

```
Epetra_MultiVector_Destroy(&mv_id)
```

Inheritance Hierarchy for Epetra_RowMatrix Class



CTrilinos

Function Overloading and Inheritance:

- C++ Trilinos code

```
Epetra_CrsMatrix *A = new Epetra_CrsMatrix(...);
Epetra_JadMatrix *B = new Epetra_JadMatrix(...);
A->TwoRowMatrixOp(B);
```

- C wrapper possible code

```
CT_Epetra_CrsMatrix_ID_t A;
CT_Epetra_JadMatrix_ID_t B;
A = Epetra_CrsMatrix_Create(...);
B = Epetra_JadMatrix_Create(...);
Epetra_RowMatrix_TwoRowMatrixOp_Crs_Jad(A,B);
```

CTrilinos

Union type definition:

```
typedef union{
        CTrilinos_Universal_ID_t universal;
        CT_Epetra_CrsMatrix_ID_t Epetra_CrsMatrix;
        CT_Epetra_BLAS_ID_t Epetra_BLAS;
        CT_Epetra_CompObject_ID_t Epetra_CompObject;
        CT_Epetra_DistObject_ID_t Epetra_DistObject;
        CT_Epetra_Object_ID_t Epetra_Object;
        CT_Epetra_Operator_ID_t Epetra_Operator;
        CT_Epetra_RowMatrix_ID_t EPetra_RowMatrix;
        CT_Epetra_SrcDistObject_ID_t Epetra_SrcDistObject;
} CT_Epetra_CrsMatrix_ID_Flex_t;
```

CTrilinos

Function Overloading and Inheritance:

- CTrilinos wrapper

```
CT_Epetra_CrsMatrix_ID_Flex_t A;
CT_Epetra_JadMatrix_ID_Flex_t B;
A.Epetra_CrsMatrix = Epetra_CrsMatrix_Create(...);
B.Epetra_JadMatrix = Epetra_JadMatrix_Create(...);
Epetra_RowMatrix_TwoRowMatrixOp
                          (A.Epetra_RowMatrix,B.Epetra_RowMatrix);
```

ForTrilinos

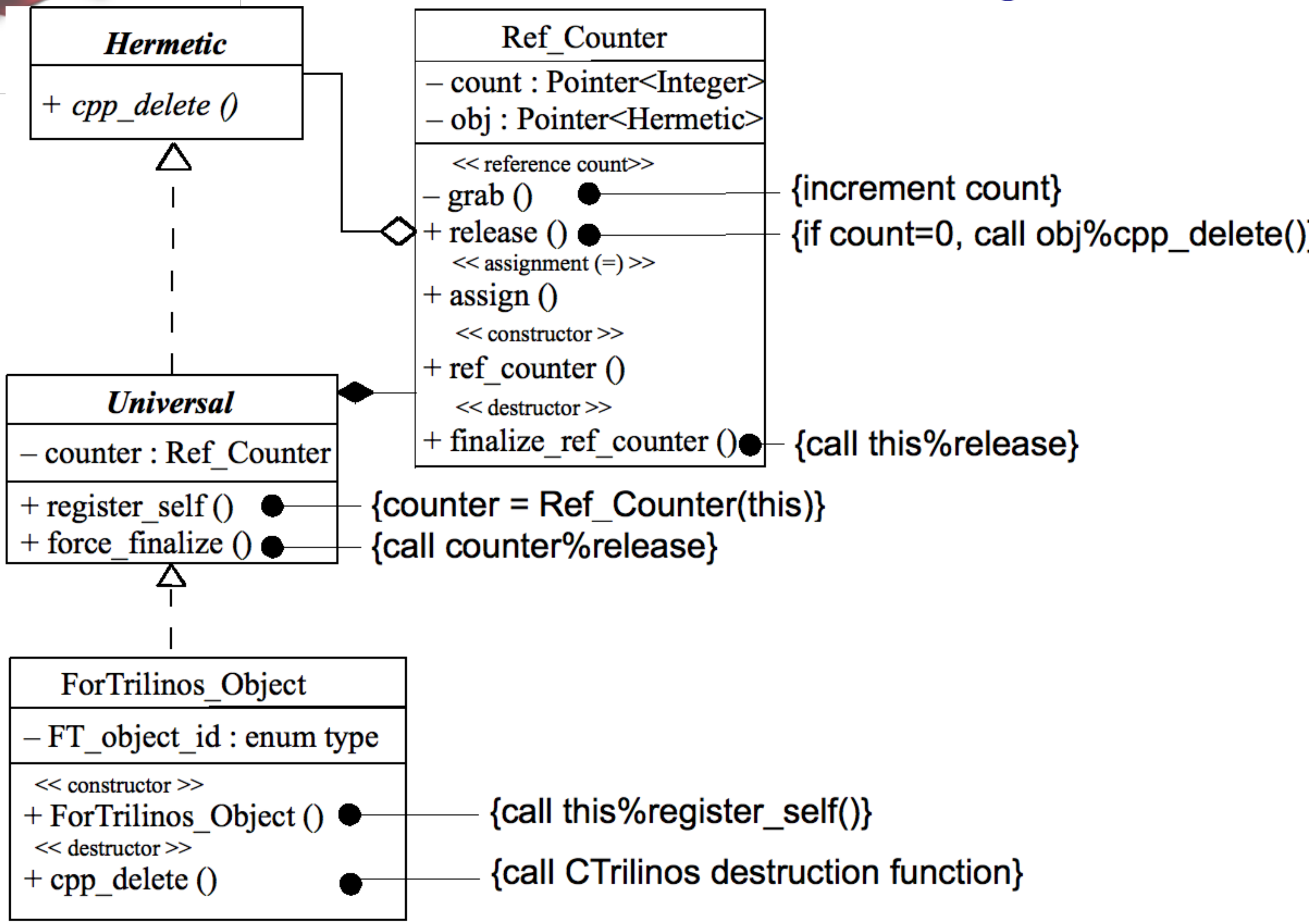
- Re-introduces OOP capabilities
  - Information hiding
  - Encapsulation
  - Inheritance
  - Static and dynamic polymorphism
- Fortran 2003 support allows
  - Abstract derived types
  - User-defined assignments and operators

ForTrilinos

Object construction/destruction in Fortran 2003:

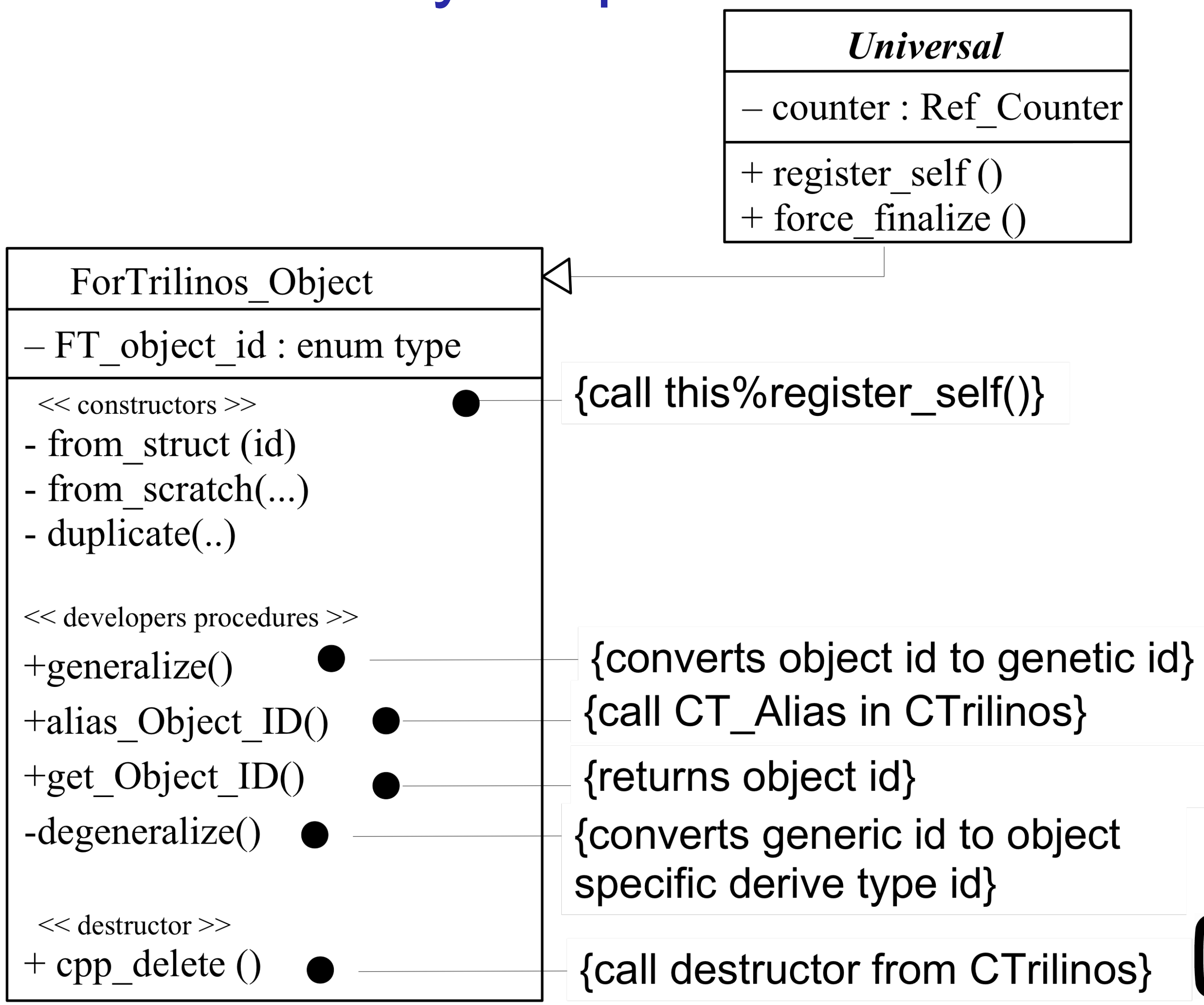
```
program main
  use field_module ,only : field
  implicit none
  type(field) :: u
  u = field()
end
```

Reference Counting



ForTrilinos

Inheritance and Polymorphism:



ForTrilinos

Inheritance:

```
module FEpetra_Vector
  use FEpetra_MultiVector, only: FEpetra_MultiVector
  private                         ! Hide everything by default
  public :: Epetra_Vector         ! Expose type/constructors/methods
  implicit none
  type ,extends(Epetra_MultiVector) :: Epetra_Vector
    private
    type(FT_Epetra_Vector_ID_t)  :: vector_id
  contains
    procedure  :: ctrilinos_delete =>ctrilinos_delete_EpetraVector
    procedure  :: get_EpetraVector_ID
    procedure ,nopass :: alias_EpetraVector_ID
    procedure  :: generalize
         ! ( other type bound procedures )
```

ForTrilinos

Inheritance:

```
interface Epetra_Vector ! constructors
  module procedure from_id, from_scratch
end interface
contains
type(Epetra_Vector) function from_id(id)
  type(FT_Epetra_Vector_ID_t) ,intent(in) :: id
  from_id%vector_id = id
  from_id%Epetra_MultiVector=Epetra_MultiVector(    &
   from_struct%alias_EpetraMultiVector_ID(from_struct%generalize
 ()))
  call from_id%register_self
end function
    ! . . .
```

ForTrilinos

Inheritance:

```
  type(Epetra_Vector) function from_scratch(...)
    type(FT_Epetra_Vector_ID_t) :: from_scratch_id
    from_scratch_id = Epetra_Vector_Create(...)     ! C wrapper
    from_scratch = from_id(from_scratch_id)
  end function
  subroutine ctrilinos_delete_EpetraVector(this)
    class(Epetra_Vector) ,intent(inout) :: this
    call Epetra_Vector_Destroy(this%vector_id)
  end subroutine
   ! . . .
end module
```

ForTrilinos

Inheritance:

```
program main
   use FEpetra_Vector, only : Epetra_Vector
   use iso_c_binding, only : c_double
   type(Epetra_Vector) :: A
    real(c_double), allocatable :: Anorm
    A=Epetra_Vector(...)
    Anorm=A%Norm2()
end main
```

ForTrilinos

Polymorphism:

```
type(Epetra_CrsMatrix) :: A
type(Epetra_JadMatrix) :: B
A=Epetra_CrsMatrix(...)
B=Epetra_JadMatrix(...)
call A%TwoRowMatrixOp(B)
```

Object-Oriented Fortran Interface

```
module FEpetra_Vector
  use forepetra ! Procedural bindings
  !...
  private ! Hide everything by default
  public :: Epetra_Vector ! Expose type/constructors/methods
  implicit none
  type ,extends(Epetra_MultiVector) :: Epetra_Vector
    private
    type(FT_Epetra_Vector_ID_t) :: vector_id
  contains
    procedure :: ReplaceGlobalValues
    procedure :: ExtractCopy_EpetraVector
    generic :: ExtractCopy => ExtractCopy_EpetraVector
    procedure :: get_element_EpetraVector
    generic :: get_Element => get_element_EpetraVector
  end type
  interface Epetra_MultiVector
    ! Specific constructor names
  end interface
contains ! Method impelementations
```

Sandia National Laboratories

Outline

- Motivation and objectives
- Methodology
  - Shadow object interface (Gray et al. 1999)
  - ForTrilinos application software stack
- Application prototype
  - Simple main
  - Prototypical PDE solver

ForTrilinos Application Prototype

Burgers Equation: $u_t + uu_x = \nu u_{xx}$

$u = u(x,t)$: velocity field

ν : diffusion coefficient

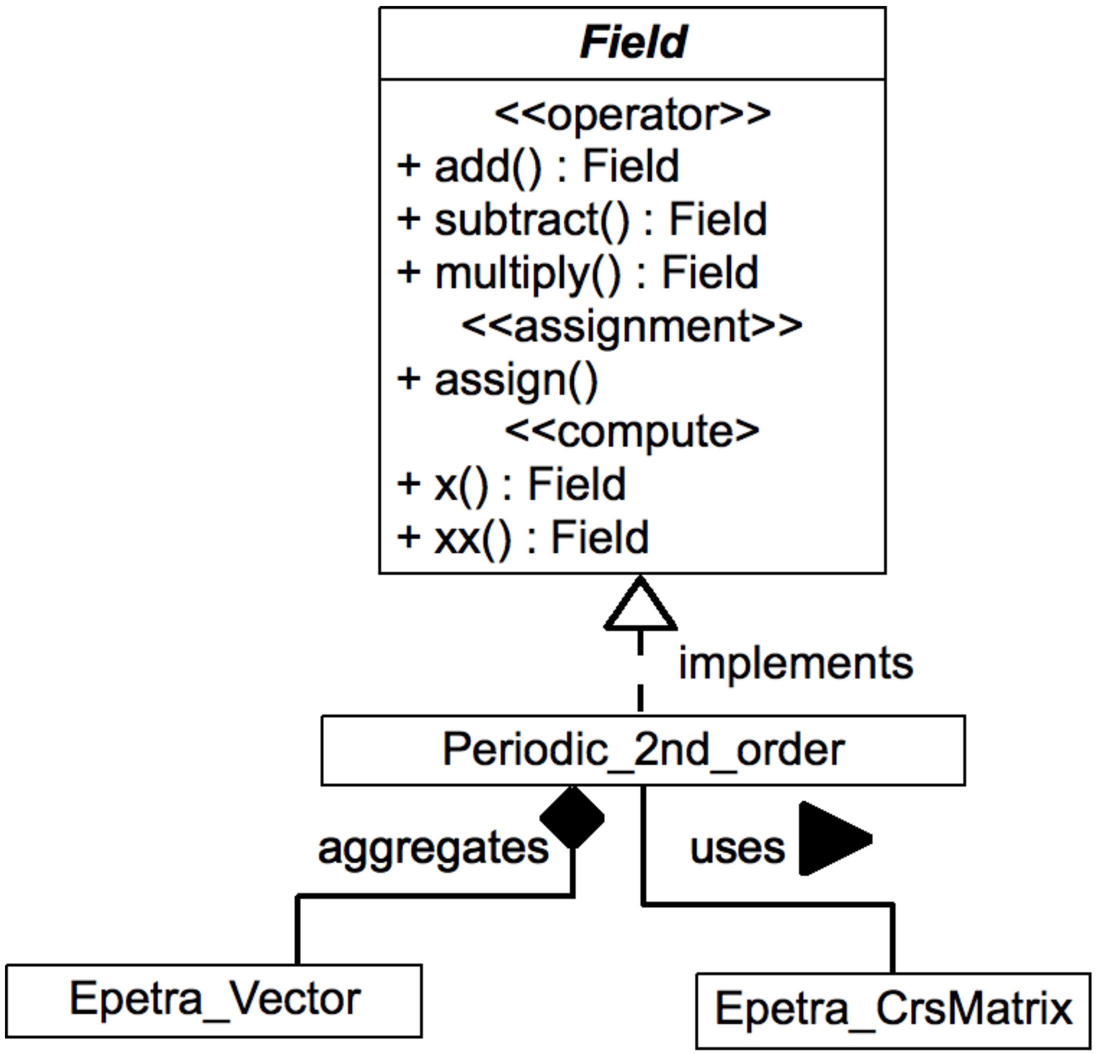
Abstract Calculus Pattern:

```
du_dt =  nu*u%xx() - u*u%x()
```

Synchronization

Asynchronous, purely functional operators and methods.

Burgers Solver Architecture



Sandia National Laboratories

Burgers Equation Solver

```
program main
#include "ForTrilinos_config.h"
#ifdef HAVE_MPI
  use mpi
  use FEpetra_MpiComm,only:Epetra_MpiComm
#else
  use FEpetra_SerialComm,only:Epetra_SerialComm
#endif
  use ForTrilinos_utils, only : valid_kind_parameters
  use iso_c_binding, only : c_int,c_double
  use field_module, only:initial_field
  use periodic_2nd_order_module, only : periodic_2nd_order
  use initializer ,only : u_initial,zero
  implicit none
   ! . . .
```

Burgers Equation Solver (cont.)

```
 #ifdef HAVE_MPI
  type(Epetra_MpiComm) :: comm
#else
  type(Epetra_SerialComm) :: comm
#endif
  type(periodic_2nd_order)          :: u,half_uu,u_half
  procedure(initial_field) ,pointer :: initial
  ! . . .
#ifdef HAVE_MPI
  call MPI_INIT(ierr)
  comm = Epetra_MpiComm(MPI_COMM_WORLD)
#else
  comm = Epetra_SerialComm()
#endif
```

Burgers Equation Solver (cont.)

```
initial => u_initial
u = periodic_2nd_order(initial,grid_resolution,comm)
initial => zero
half_uu = periodic_2nd_order(initial,grid_resolution,comm)
u_half = periodic_2nd_order(initial,grid_resolution,comm)
do tstep=1,1000 !2nd-order Runge-Kutta:
 dt = u%runge_kutta_2nd_step(nu ,grid_resolution)
 half_uu = u*u*half
 u_half = u + (u%xx()*nu - half_uu%x())*dt*half ! 1st substep
 half_uu = u_half*u_half*half
 u  = u + (u_half%xx()*nu - half_uu%x())*dt     ! 2nd substep
 t = t + dt
end do
```

Burgers Equation Solver (cont.)

```
  call half_uu%force_finalize
  call u_half%force_finalize
  call u%force_finalize
  call comm%force_finalize
#ifdef HAVE_MPI
  call MPI_FINALIZE(rc)
#endif
end program
```

Sample Operator

```
function multiple(lhs,rhs)
  class(periodic_2nd_order) ,intent(in) :: lhs
  real(c_double) ,intent(in)  :: rhs
  class(field) ,allocatable :: multiple
  type(periodic_2nd_order),allocatable ::local_multiple
  type(error) :: ierr
  allocate(periodic_2nd_order::local_multiple)
  local_multiple%f=Epetra_Vector(map,.true.)
  call local_multiple%f%Scale(rhs,lhs%f,ierr)
  ! ...
  call move_alloc(local_multiple,multiple)
end function
```

ForTrilinos Application Prototype

Burgers equation solver performance