A Dynamical Systems Approach to Physical Oceanography

Jonas Thies [1] Fred Wubs [1] Henk Dijkstra [2]

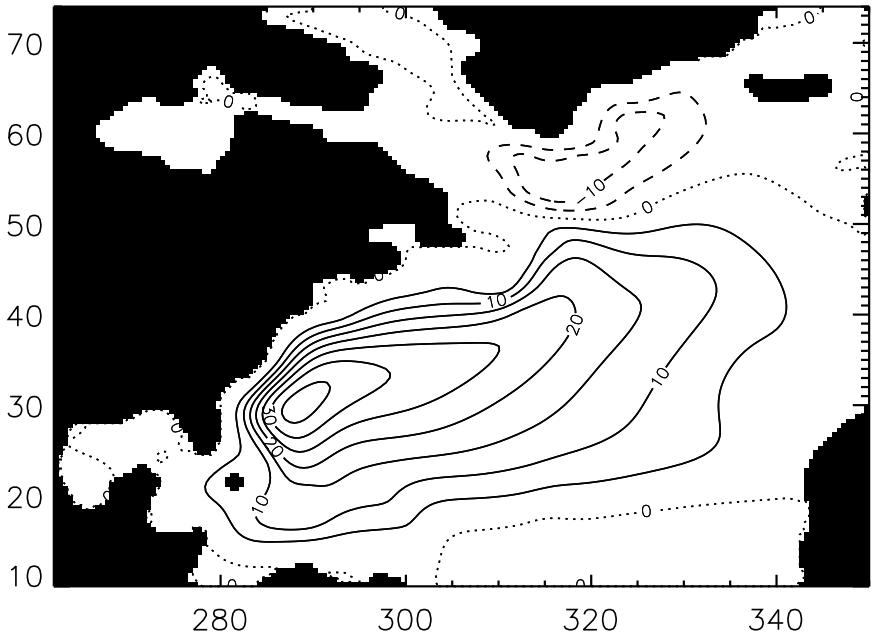
[1] Department of Mathematics and Computer Science
University of Groningen, the Netherlands

[2] Institute for Marine and Atmospheric Studies (IMAU)
University of Utrecht

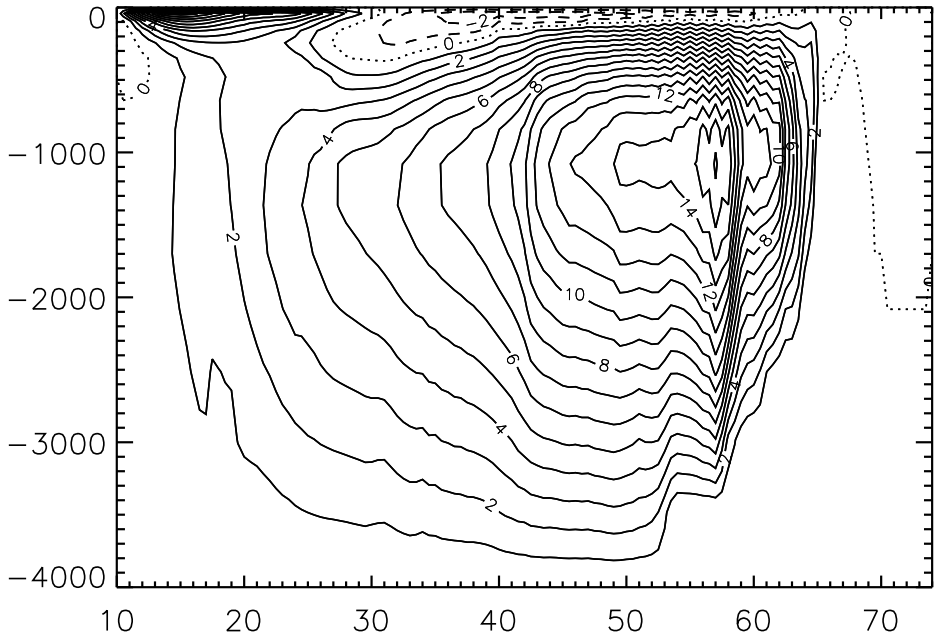
Trilinos User Group Meeting, October 22nd, 2008

Example: Breakdown of the North Atlantic MOC

- Wind-driven circulation



- Meridional overturning circulation



Quantitative and qualitative parameter sensitivity

(e.g global warming, arctic sea-ice melting)

Why Implicit Ocean Models?

Why Implicit Ocean Models?

- Phase space trajectories (**continuation**),

Why Implicit Ocean Models?

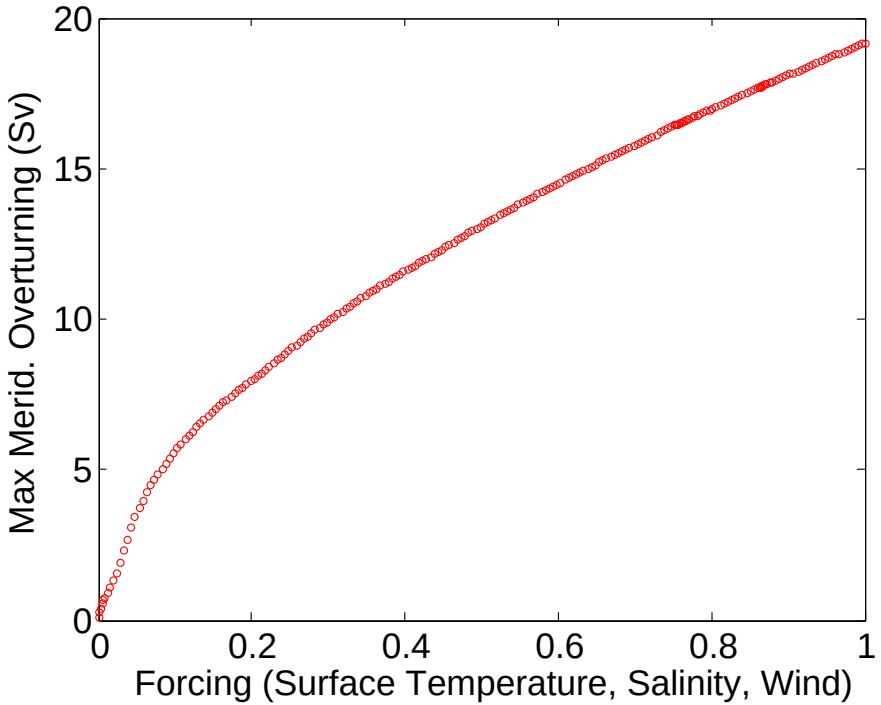
- Phase space trajectories (**continuation**),
- Eigenmodes and bifurcations (linear stability analysis)

Why Implicit Ocean Models?

- Phase space trajectories (**continuation**),
- Eigenmodes and bifurcations (linear stability analysis)
- Integrate long time scales ('spin-up')

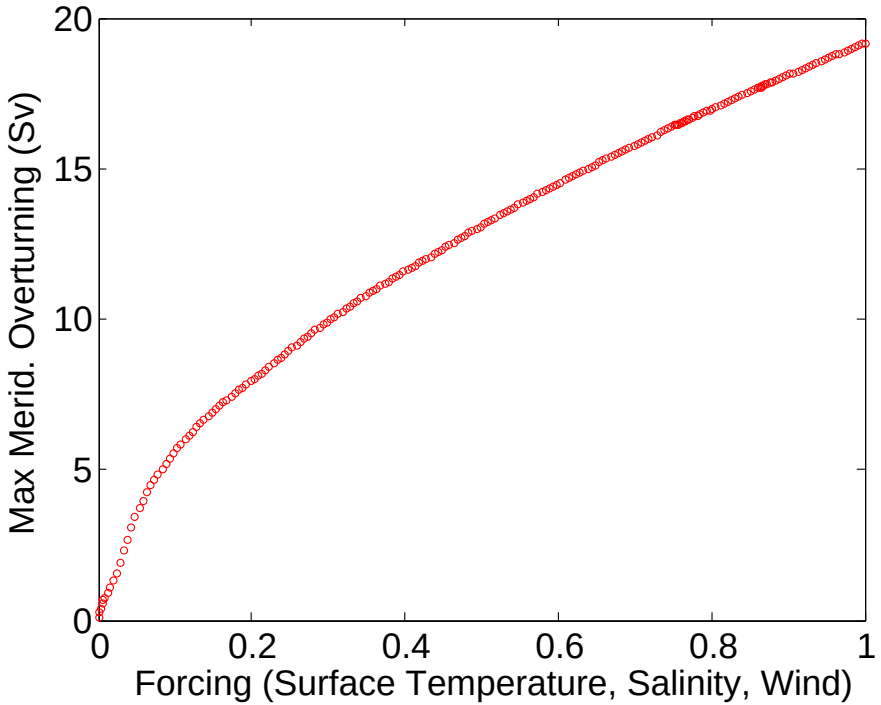
Why Implicit Ocean Models?

- Phase space trajectories (**continuation**),
- Eigenmodes and bifurcations (linear stability analysis)
- Integrate long time scales ('spin-up')



Why Implicit Ocean Models?

- Phase space trajectories (**continuation**),
- Eigenmodes and bifurcations (linear stability analysis)
- Integrate long time scales ('spin-up')



$\Longrightarrow$ **Need to solve many very large nonlinear systems**

Ocean Primitive Equations

- $\boldsymbol{\nabla_h \cdot \vec{u} + \frac{\partial w}{\partial z} = 0}$ Continuity equation
- $\boldsymbol{\frac{D\vec{u}}{Dt} = -\nabla_h p + A_h \nabla^2 \vec{u} + \Omega \times \vec{u}}$ Horizontal momentum
- $\boldsymbol{\frac{\partial p}{\partial z} = g\rho(p, T, S)}$ Hydrostatic balance
- $\boldsymbol{\frac{DT}{Dt} = \kappa \nabla^2 T + SGS^\star}$ Tracer transport (temp., salt)

Note: some metric terms have been skipped for readability

* Tracer mixing and 'convective adjustment'
(strong, local, anisotropic 'density diffusion')

Structure of the Linear Systems

$$\begin{pmatrix} & \end{pmatrix} \begin{pmatrix} u_{uv} \\ u_w \\ u_p \\ u_{TS} \end{pmatrix} = \begin{pmatrix} b_{uv} \\ b_w \\ b_p \\ b_{TS} \end{pmatrix}$$

Structure of the Linear Systems

$$\begin{pmatrix} A_{uv} & E_{uv} & G_{uv} & 0 \\ & & & \end{pmatrix} \begin{pmatrix} u_{uv} \\ u_w \\ u_p \\ u_{TS} \end{pmatrix} = \begin{pmatrix} b_{uv} \\ b_w \\ b_p \\ b_{TS} \end{pmatrix}$$

- A_{uv}: convection, diffusion, Coriolis
- G_*: discrete gradient

Structure of the Linear Systems

$$\begin{pmatrix} \mathbf{A_{uv}} & \mathbf{E_{uv}} & G_{uv} & 0 \\ 0 & 0 & G_w & \mathbf{B_{TS}} \\ & & & \end{pmatrix} \begin{pmatrix} u_{uv} \\ u_w \\ u_p \\ u_{TS} \end{pmatrix} = \begin{pmatrix} b_{uv} \\ b_w \\ b_p \\ b_{TS} \end{pmatrix}$$

- $\mathbf{A_{uv}}$: convection, diffusion, Coriolis
- G_*: discrete gradient
- $\mathbf{B_{TS}}$: buoyancy term

Structure of the Linear Systems

$$\begin{pmatrix} \mathbf{A_{uv}} & \mathbf{E_{uv}} & G_{uv} & 0 \\ 0 & 0 & G_w & \mathbf{B_{TS}} \\ D_{uv} & D_w & 0 & 0 \end{pmatrix} \begin{pmatrix} u_{uv} \\ u_w \\ u_p \\ u_{TS} \end{pmatrix} = \begin{pmatrix} b_{uv} \\ b_w \\ b_p \\ b_{TS} \end{pmatrix}$$

- $\mathbf{A_{uv}}$: convection, diffusion, Coriolis
- G_*: discrete gradient
- D_*: discrete divergence
- $\mathbf{B_{TS}}$: buoyancy term

Structure of the Linear Systems

$$\begin{pmatrix} \mathbf{A_{uv}} & \mathbf{E_{uv}} & G_{uv} & 0 \\ 0 & 0 & G_w & \mathbf{B_{TS}} \\ D_{uv} & D_w & 0 & 0 \\ C_{uv} & C_w & 0 & \mathbf{A_{TS}} \end{pmatrix} \begin{pmatrix} u_{uv} \\ u_w \\ u_p \\ u_{TS} \end{pmatrix} = \begin{pmatrix} b_{uv} \\ b_w \\ b_p \\ b_{TS} \end{pmatrix}$$

- $\mathbf{A_{uv}}$: convection, diffusion, Coriolis
- G_*: discrete gradient
- D_*: discrete divergence
- $\mathbf{B_{TS}}$: buoyancy term
- C_*: tracer advection
- $\mathbf{A_{TS}}$: tracer advection, diffusion, subgrid scale mixing

Block Preconditioning

Block Preconditioning

- Let $p = \bar{p} + \tilde{p}$, $\bar{p} = Mp, M^T = \text{null}(G_w)$.

Block Preconditioning

- Let $p = \bar{p} + \tilde{p}$, $\bar{p} = Mp, M^T = \text{null}(G_w)$.
- rearrange Jacobian

$$\begin{pmatrix} A_p & 0 & 0 & B_{TS} \\ G_{uv} & A_{uv} \quad G_{uv}M^T \\ & MD_{uv} \quad 0 & 0 & 0 \\ 0 & D_{uv} & A_w & 0 \\ 0 & C_{uv} & C_w & A_{TS} \end{pmatrix} \begin{pmatrix} u_{\tilde{p}} \\ u_{uv} \\ u_{\bar{p}} \\ u_w \\ u_{TS} \end{pmatrix} = \begin{pmatrix} b_w \\ b_{uv} \\ b_{\bar{p}} \\ b_{\tilde{p}} \\ b_{TS} \end{pmatrix}$$

Block Preconditioning

- Let $p = \bar{p} + \tilde{p}$, $\bar{p} = Mp, M^T = \mathrm{null}(G_w)$.
- rearrange Jacobian

$$\begin{pmatrix} A_p & 0 & 0 & B_{TS} \\ G_{uv} & \begin{matrix} A_{uv} \\ MD_{uv} \end{matrix} & \begin{matrix} G_{uv}M^T \\ 0 \end{matrix} & \begin{matrix} 0 \\ 0 \end{matrix} \\ 0 & D_{uv} & A_w & 0 \\ 0 & C_{uv} & C_w & A_{TS} \end{pmatrix} \begin{pmatrix} u_{\tilde{p}} \\ u_{uv} \\ u_{\bar{p}} \\ u_w \\ u_{TS} \end{pmatrix} = \begin{pmatrix} b_w \\ b_{uv} \\ b_{\bar{p}} \\ b_{\tilde{p}} \\ b_{TS} \end{pmatrix}$$

- A_p and A_w are lower/upper triangular

Block Preconditioning

- Let $p = \bar{p} + \tilde{p}$, $\bar{p} = Mp, M^T = \mathrm{null}(G_w)$.
- rearrange Jacobian

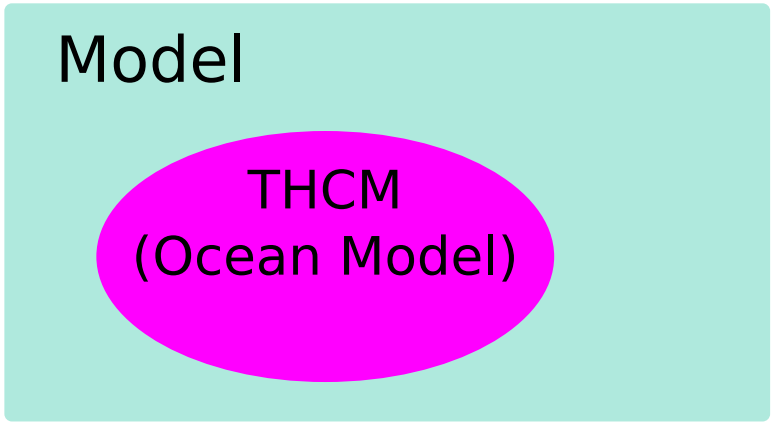
$$\begin{pmatrix} \mathbf{A_p} & 0 & 0 & \mathbf{B_{TS}} \\ G_{uv} & \begin{matrix} A_{uv} \\ MD_{uv} \end{matrix} & \begin{matrix} G_{uv}M^T \\ 0 \end{matrix} & \begin{matrix} 0 \\ 0 \end{matrix} \\ 0 & D_{uv} & \mathbf{A_w} & 0 \\ 0 & C_{uv} & C_w & \mathbf{A_{TS}} \end{pmatrix} \begin{pmatrix} u_{\tilde{p}} \\ u_{uv} \\ u_{\bar{p}} \\ u_w \\ u_{TS} \end{pmatrix} = \begin{pmatrix} b_w \\ b_{uv} \\ b_{\bar{p}} \\ b_{\tilde{p}} \\ b_{TS} \end{pmatrix}$$

- $\mathbf{A_p}$ and $\mathbf{A_w}$ are lower/upper triangular
- Drop $\mathbf{B_{TS}}$, $\Longrightarrow$ Block-Gauss-Seidel preconditioner

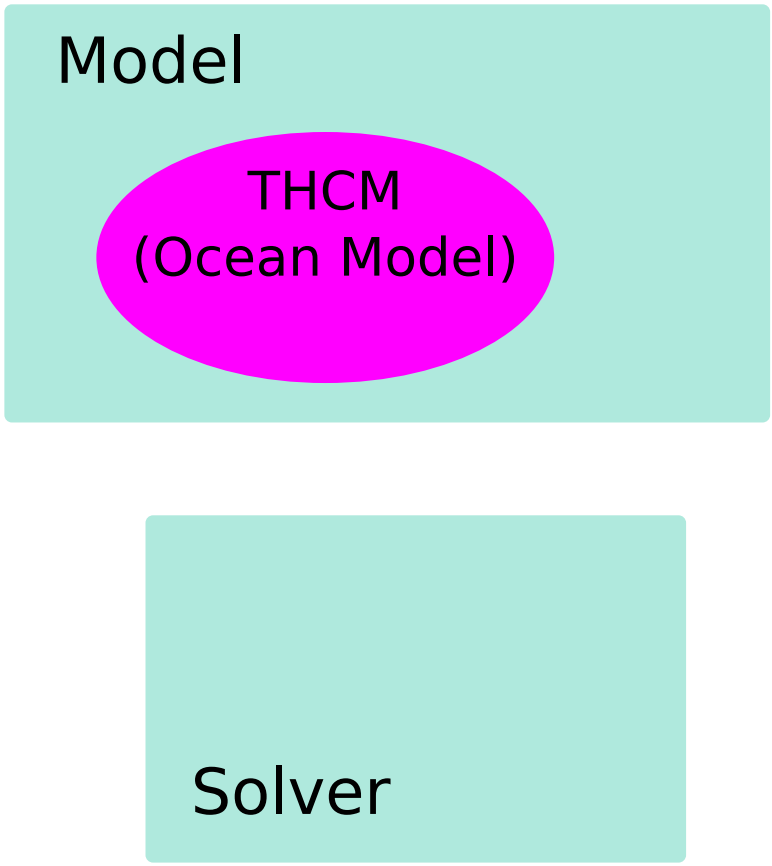
Trilinos in Action

Model

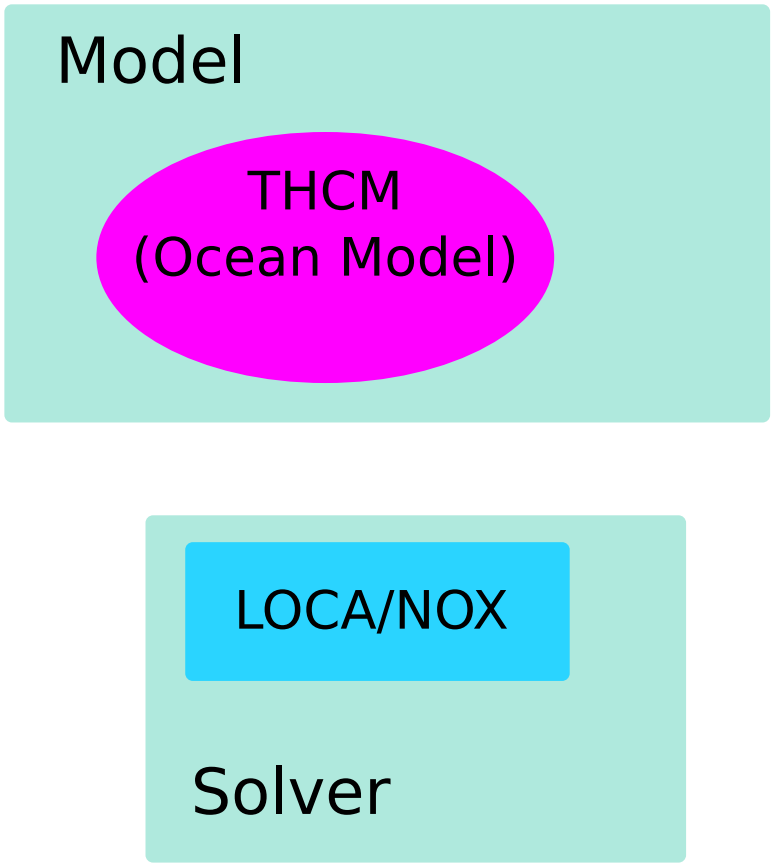
Trilinos in Action



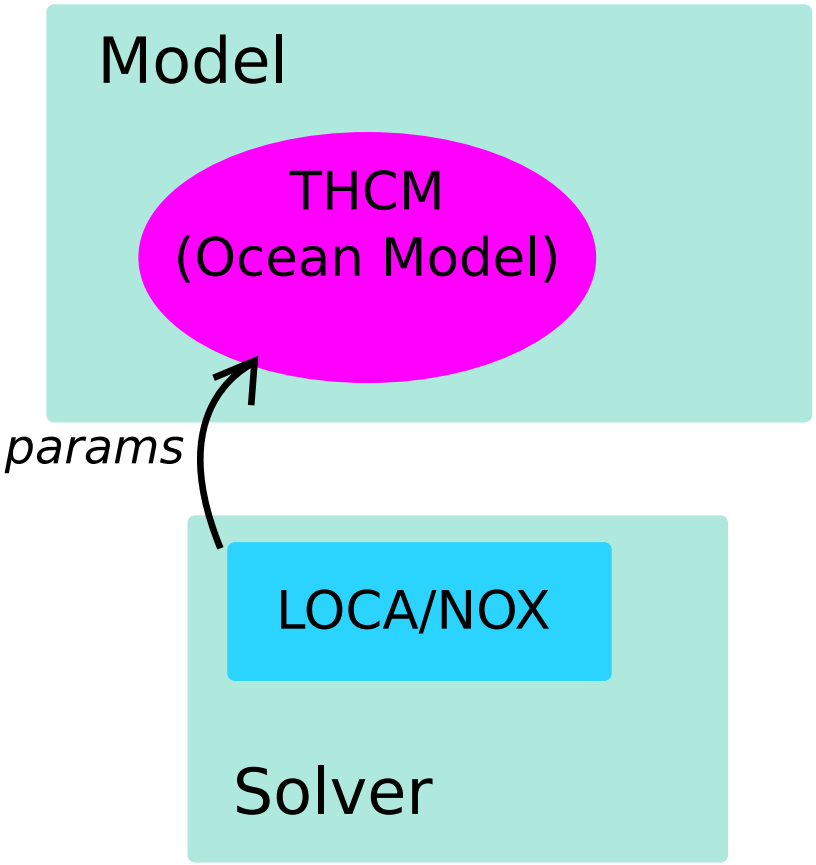
Trilinos in Action



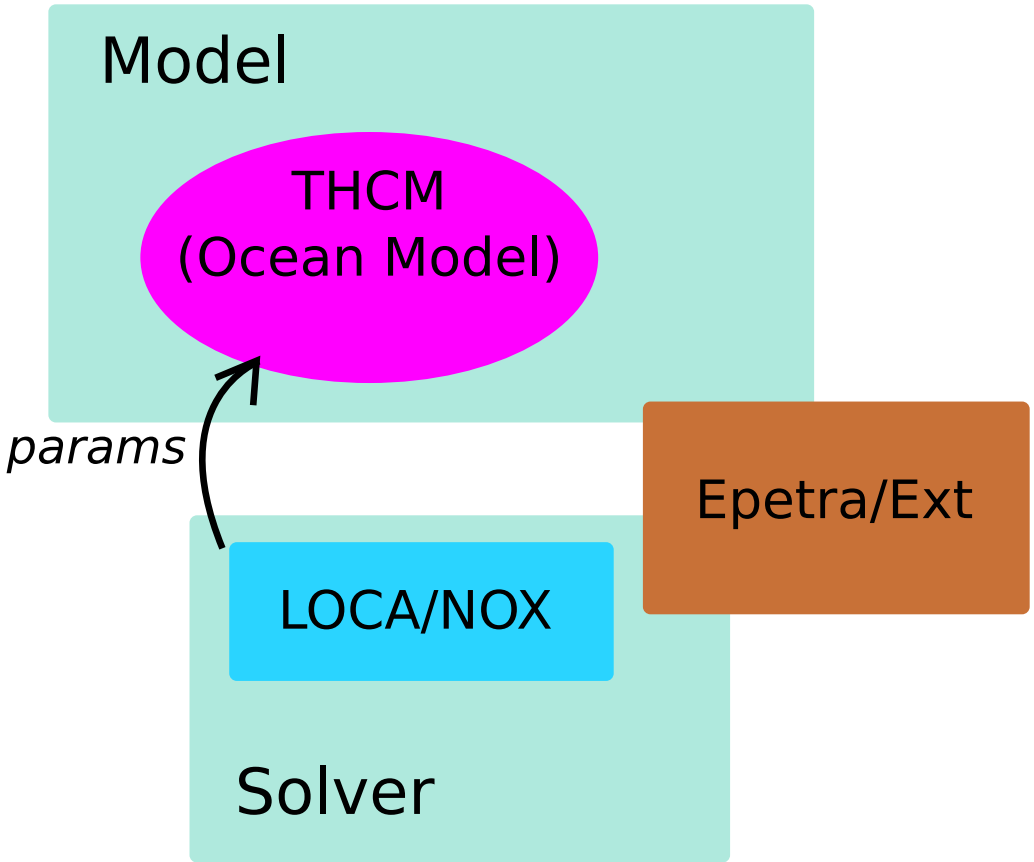
Trilinos in Action



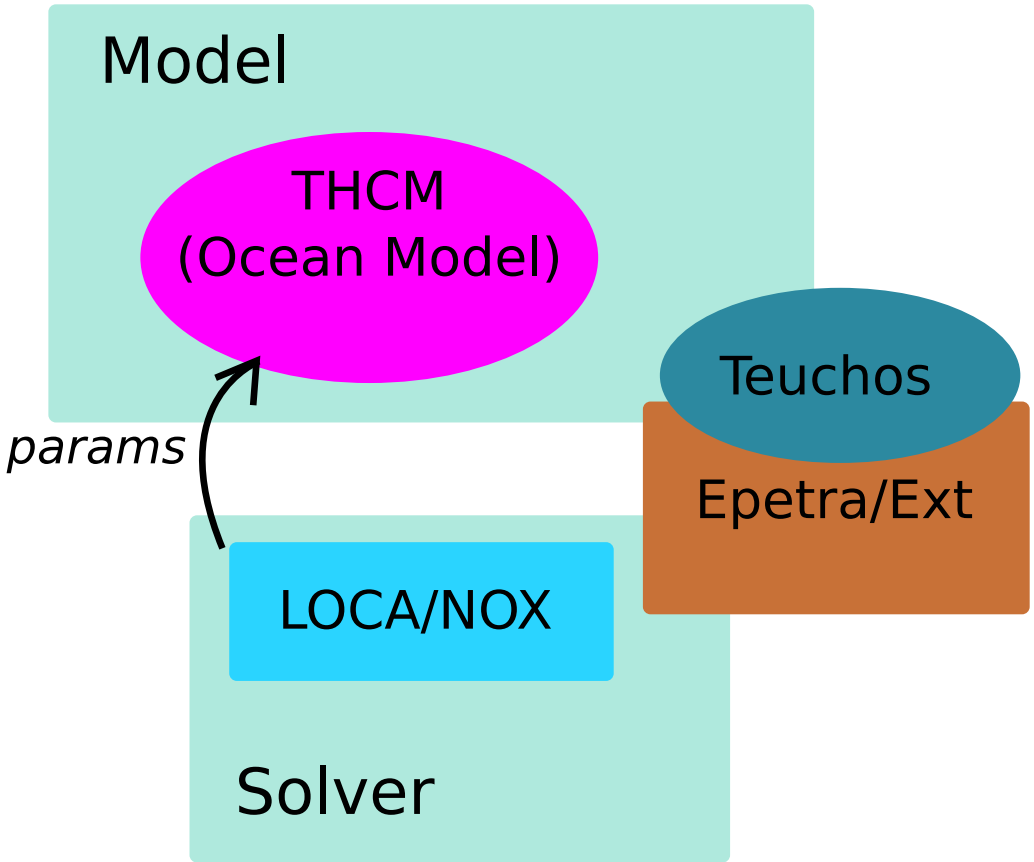
Trilinos in Action



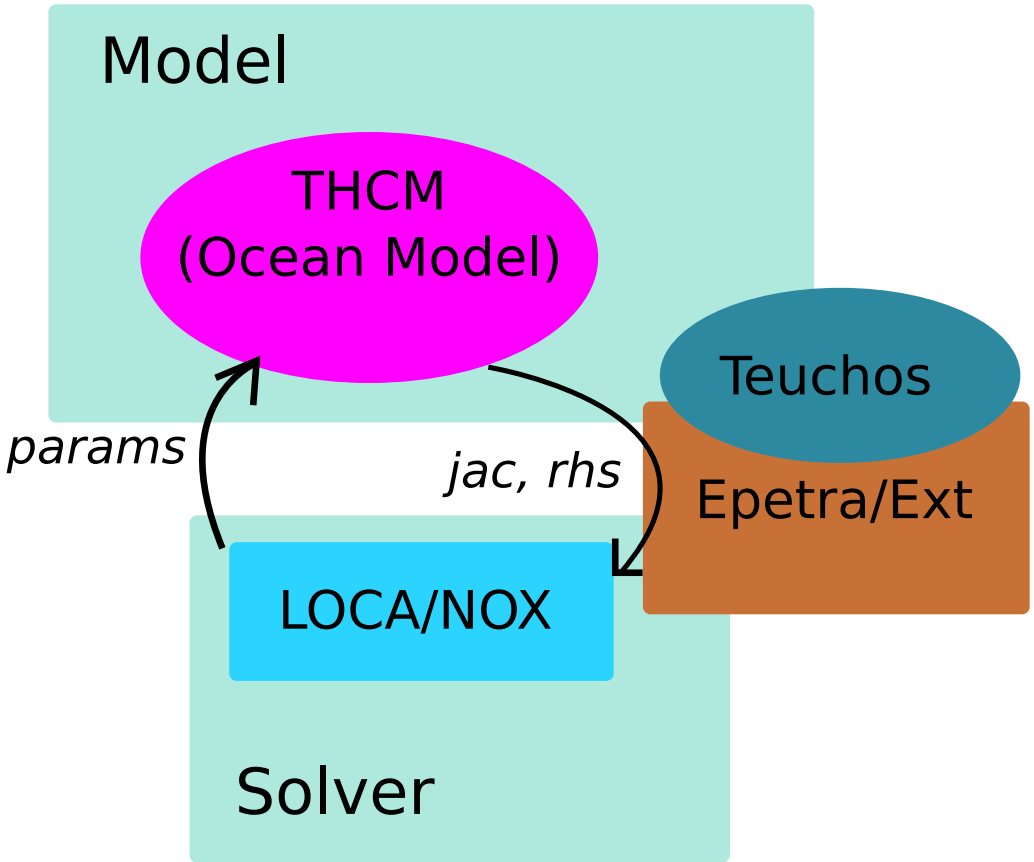
Trilinos in Action



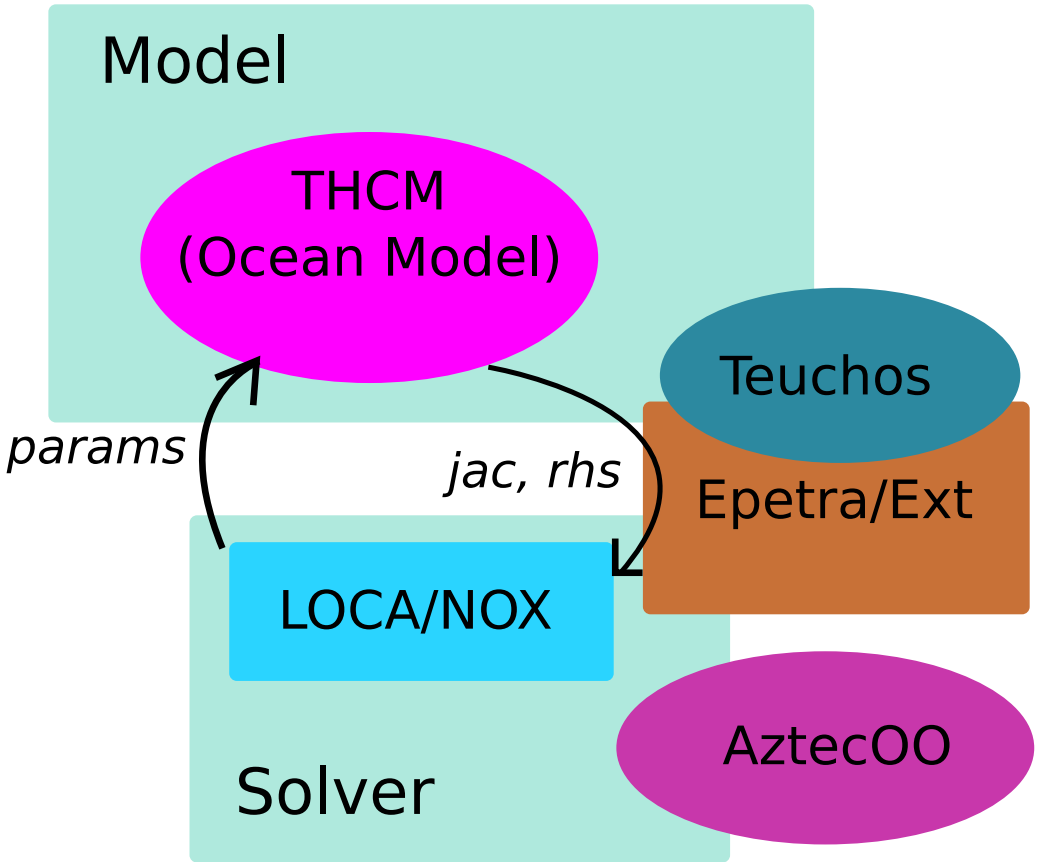
Trilinos in Action



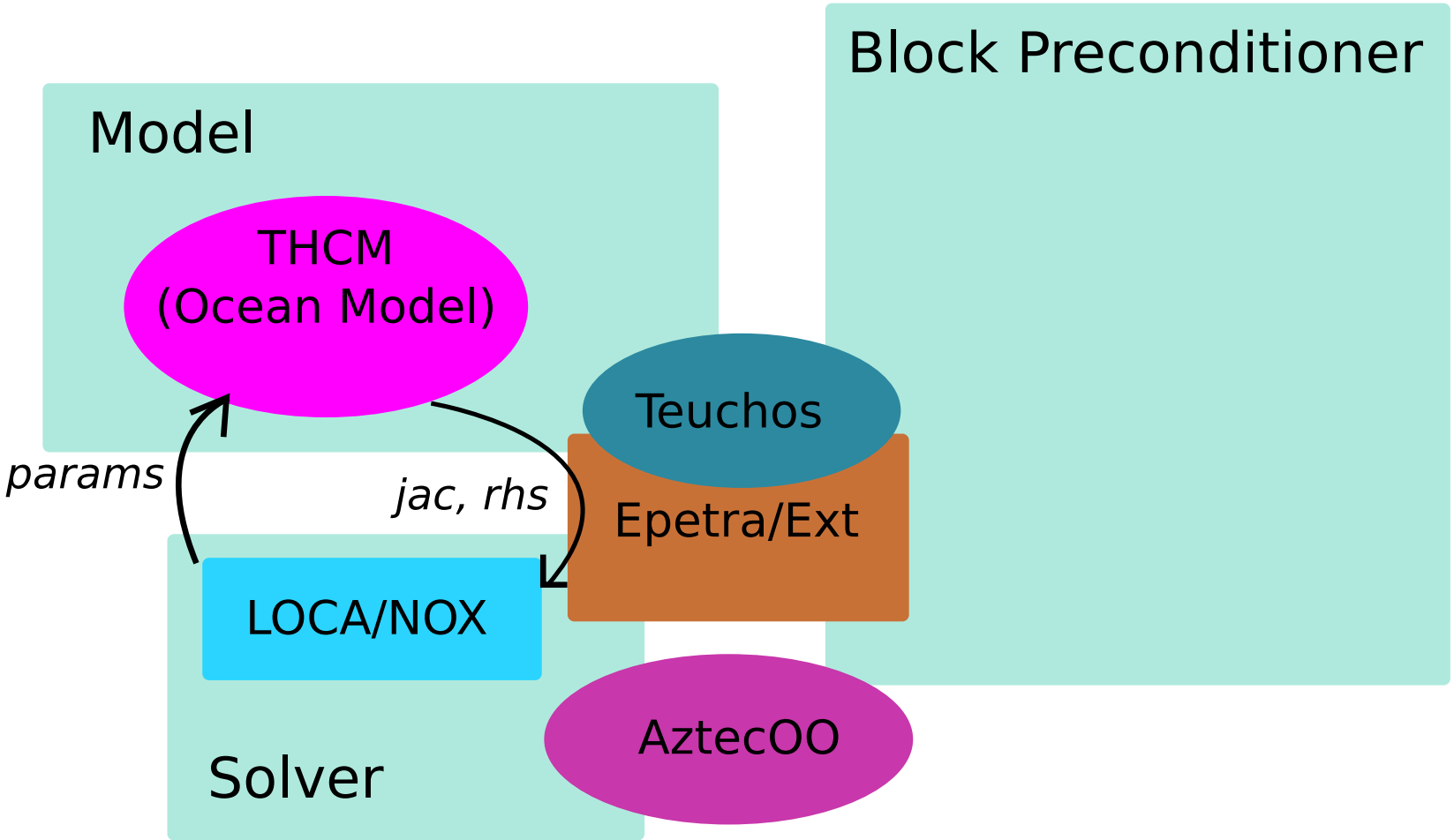
Trilinos in Action



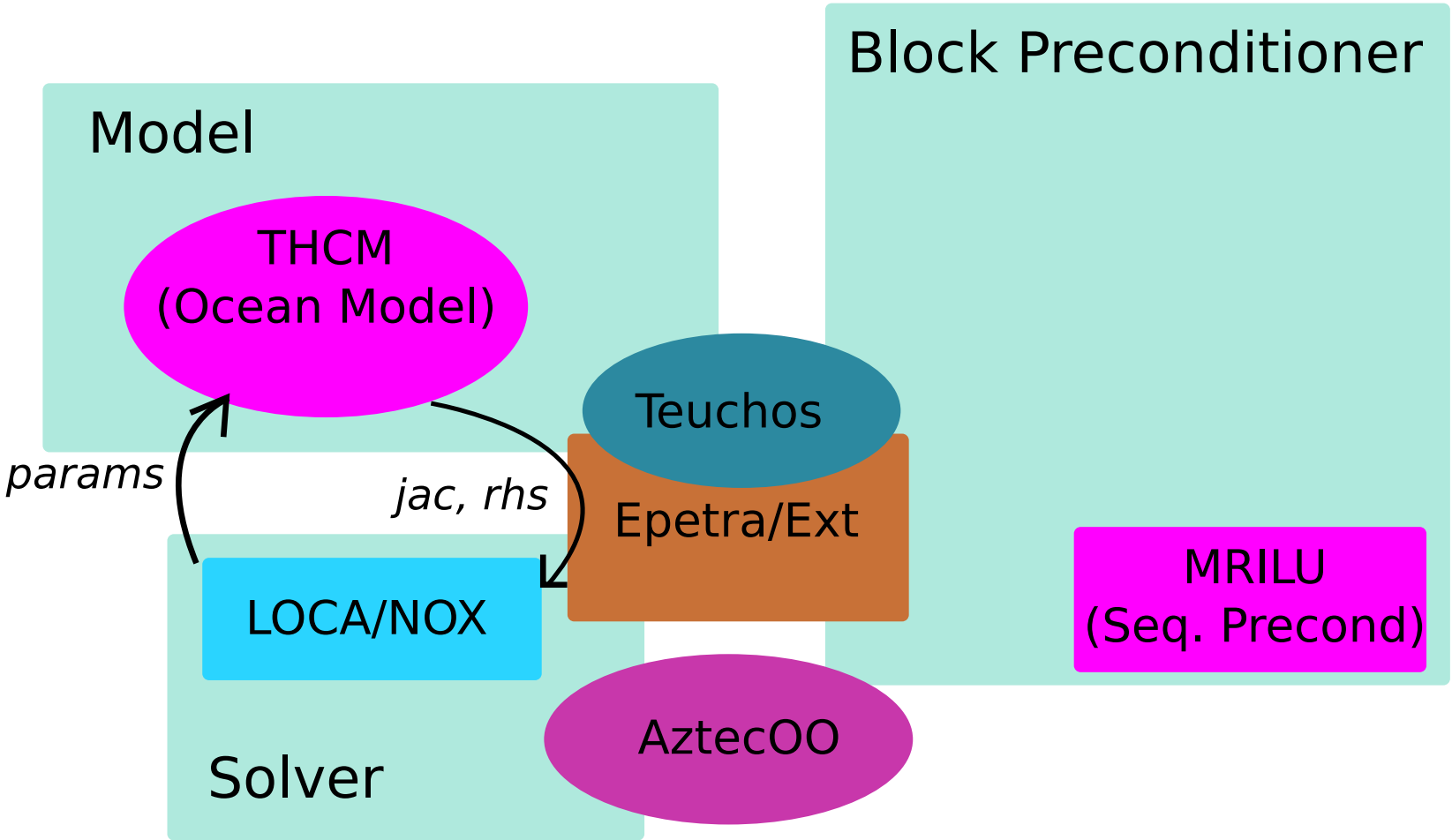
Trilinos in Action



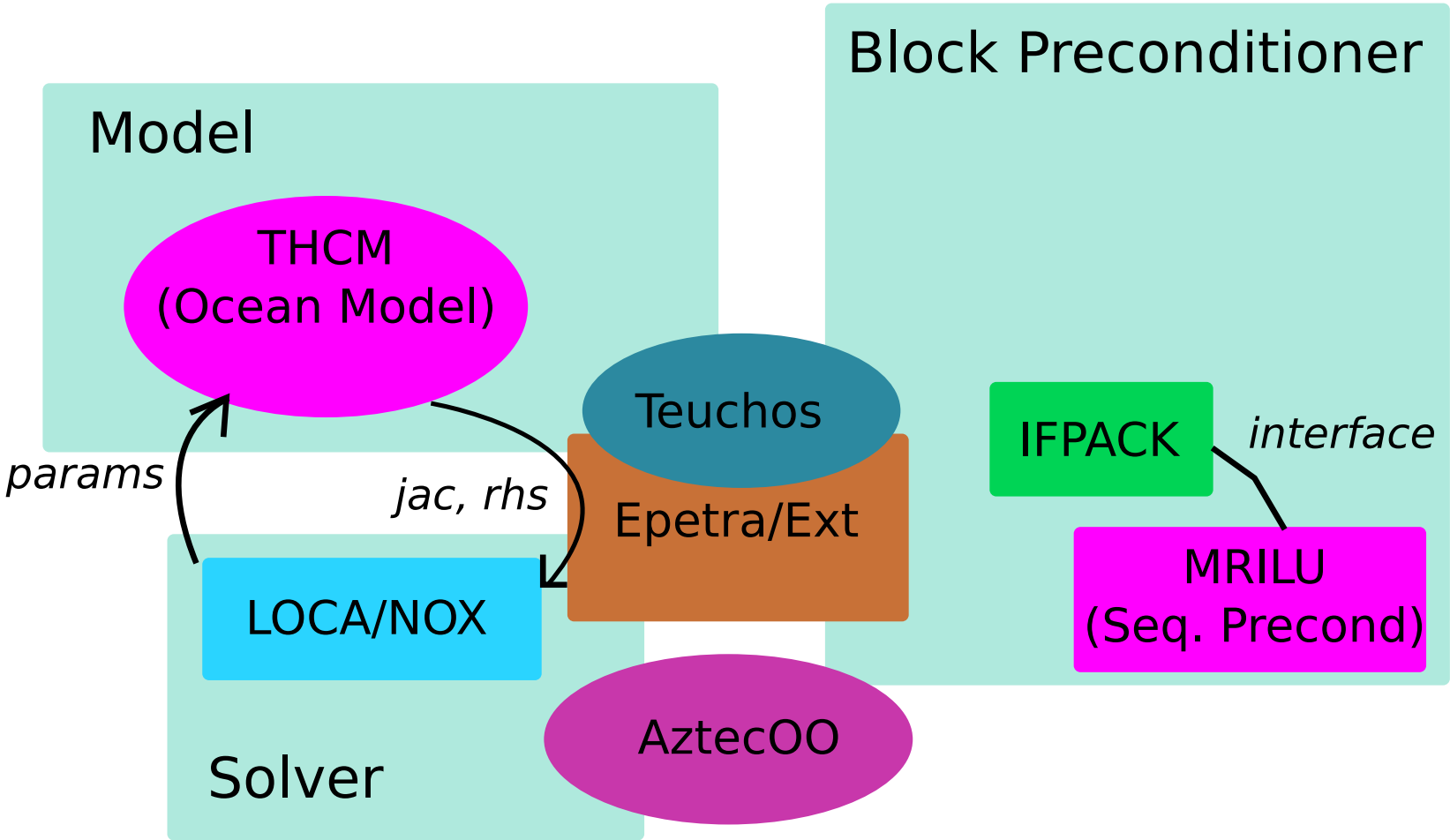
Trilinos in Action



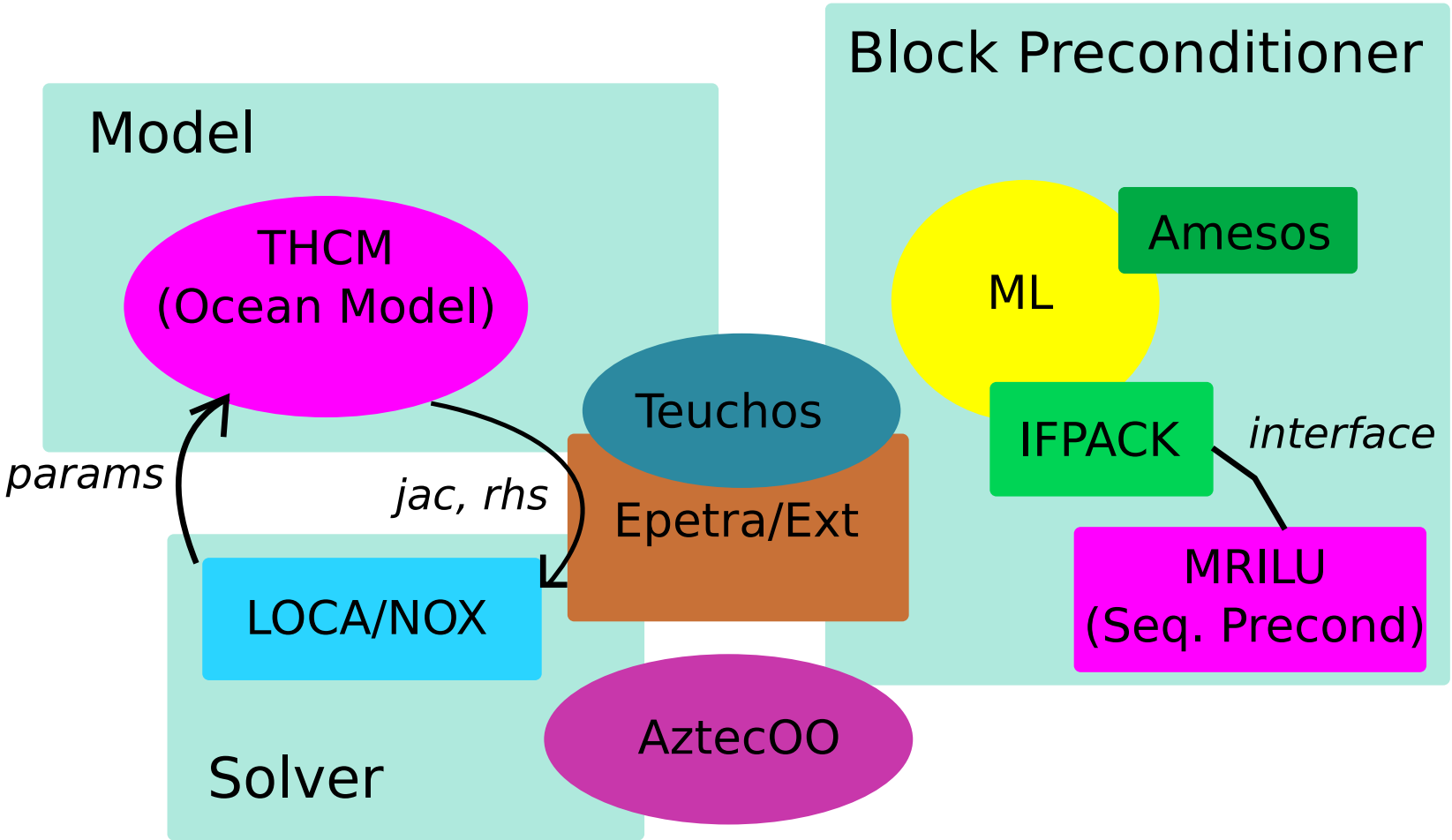
Trilinos in Action



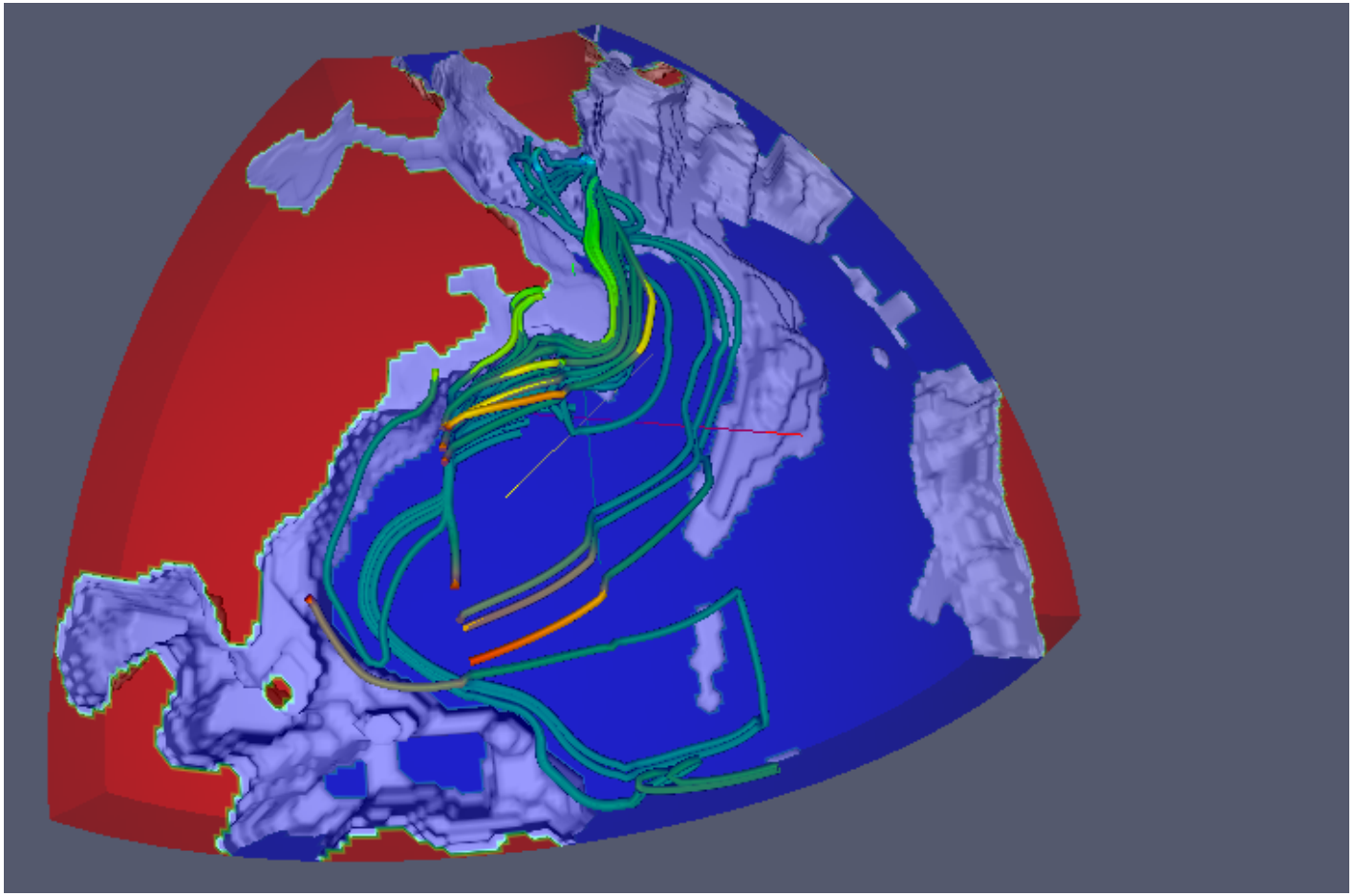
Trilinos in Action



Trilinos in Action



Discussion



Thanks for having me!

The Petra Object Model: Useful Parallel Abstractions

Michael A. Heroux
Sandia National Laboratories

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.

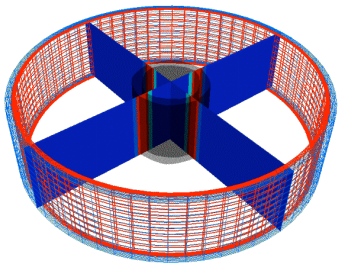
Sandia National Laboratories

Alternate Title

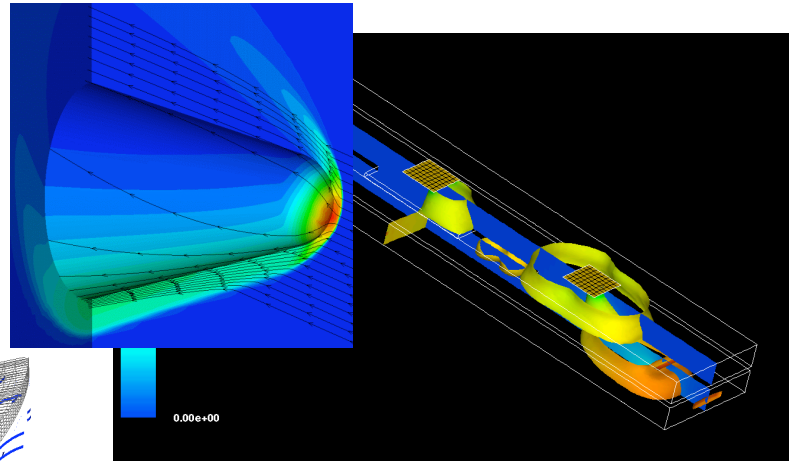
"In absence of a portable parallel language and in the presence of recurring yet evolving system architectures, while in the meantime trying to get work done with the current tools and solve problems of interest, what we have developed to solve large-scale systems of equations on large-scale computers and allow for adapting in the future."

Sandia National Laboratories

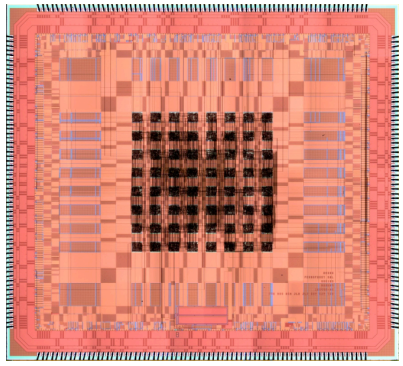
Target Problems: PDES and more…

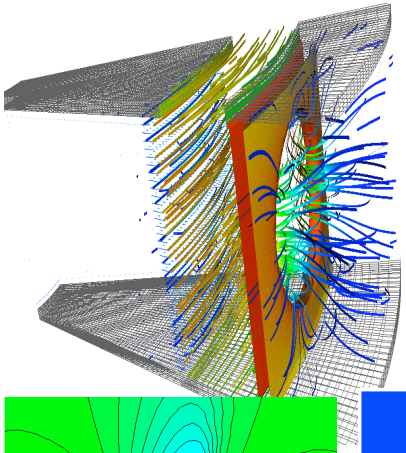


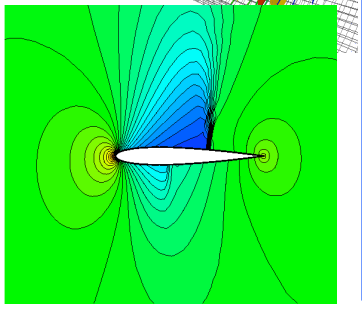
PDES

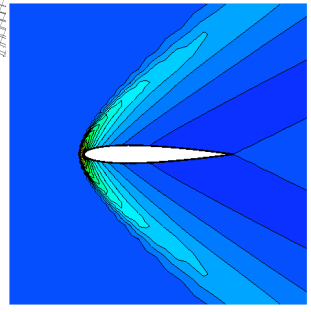


Circuits



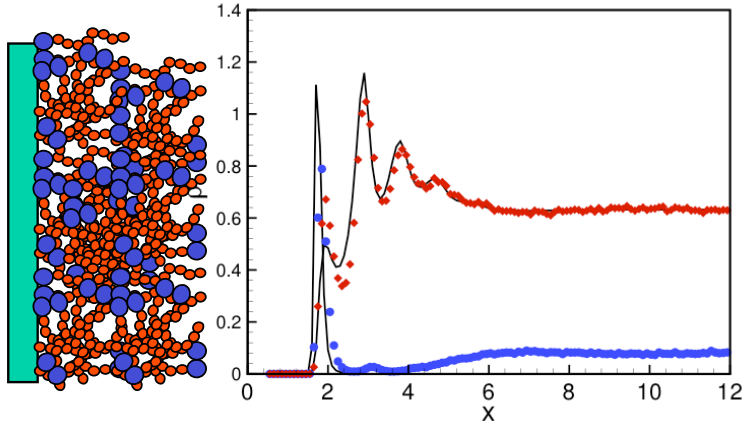






And More…

Inhomogeneous Fluids



Trilinos Packages

- Trilinos is a collection of *Packages*.
- Each package is:
  - Focused on important, state-of-the-art algorithms in its problem regime.
  - Developed by a small team of domain experts.
  - Self-contained: No explicit dependencies on any other software packages (with some special exceptions).
  - Configurable/buildable/documented on its own.
- Sample packages: NOX, AztecOO, ML, IFPACK, Meros.
- Special package collections:
  - Petra (Epetra, Tpetra, Jpetra): Concrete Data Objects
  - Thyra: Abstract Conceptual Interfaces
  - Teuchos: Common Tools.
  - New_package: Jumpstart prototype.

Sandia National Laboratories

Full "Vertical" Solver Coverage

Trilinos Packages

· Optimization Problems: · Unconstrained: · Constrained:	Find $u \in \Re^n$ that minimizes $f(u)$ Find $y \in \Re^m$ and $u \in \Re^n$ that minimizes $f(y,u)$ s.t. $c(y,u) = 0$	MOOCHO
· Transient Problems: · DAEs/ODEs:	Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0,T],\ x(0) = x_0,\ \dot{x}(0) = x_0'$ for $x(t) \in \Re^n,\ t \in [0,T]$	Rythmos
· Nonlinear Problems: · Nonlinear equations: · Stability analysis:	Given nonlinear op $c(x,u) \in \Re^{n+m} \rightarrow \Re^n$ Solve $c(x) = 0$ for $x \in \Re^n$ For $c(x,u) = 0$ find space $u \in U \ni \frac{\partial c}{\partial x}$ singular	NOX LOCA
· Implicit Linear Problems: · Linear equations: · Eigen problems:	Given linear ops (matrices) $A, B \in \Re^{n \times n}$ Solve $Ax = b$ for $x \in \Re^n$ Solve $Av = \lambda Bv$ for (all) $v \in \Re^n,\ \lambda \in \Re$	AztecOO, Belos, Ifpack,ML,etc. Anasazi
· Explicit Linear Problems: · Matrix/graph equations: · Vector problems:	Compute $y = Ax;\ A = A(G);\ A \in \Re^{m \times n}, G \in$ □ $^{m \times n}$ Compute $y = \alpha x + \beta w;\ \alpha =< x, y >;\ x, y \in \Re^n$	Epetra, Tpetra

Observations

- 30-80% of application runtime spent in solvers or:
- Solvers are not applications but can drive app performance.

- Vast majority of solver time spent in kernels or:
- Kernels are not solvers but can drive solver performance.

Sandia National Laboratories

A Simple Epetra/AztecOO Program

```
// Header files omitted…
int main(int argc, char *argv[]) {
Epetra_SerialComm Comm();
```

```
// ***** Map puts same number of equations on each pe *****

  int NumMyElements = 1000 ;
  Epetra_Map Map(-1, NumMyElements, 0, Comm);
  int NumGlobalElements = Map.NumGlobalElements();
```

```
// ***** Create an Epetra_Matrix  tridiag(-1,2,-1) *****

  Epetra_CrsMatrix A(Copy, Map, 3);
  double negOne = -1.0; double posTwo = 2.0;

  for (int i=0; i<NumMyElements; i++) {
    int GlobalRow = A.GRID(i);
    int RowLess1 = GlobalRow - 1;
    int RowPlus1 = GlobalRow + 1;
    if (RowLess1!=-1)
      A.InsertGlobalValues(GlobalRow, 1, &negOne, &RowLess1);
    if (RowPlus1!=NumGlobalElements)
      A.InsertGlobalValues(GlobalRow, 1, &negOne, &RowPlus1);
    A.InsertGlobalValues(GlobalRow, 1, &posTwo, &GlobalRow);
  }
A.FillComplete(); // Transform from GIDs to LIDs
```

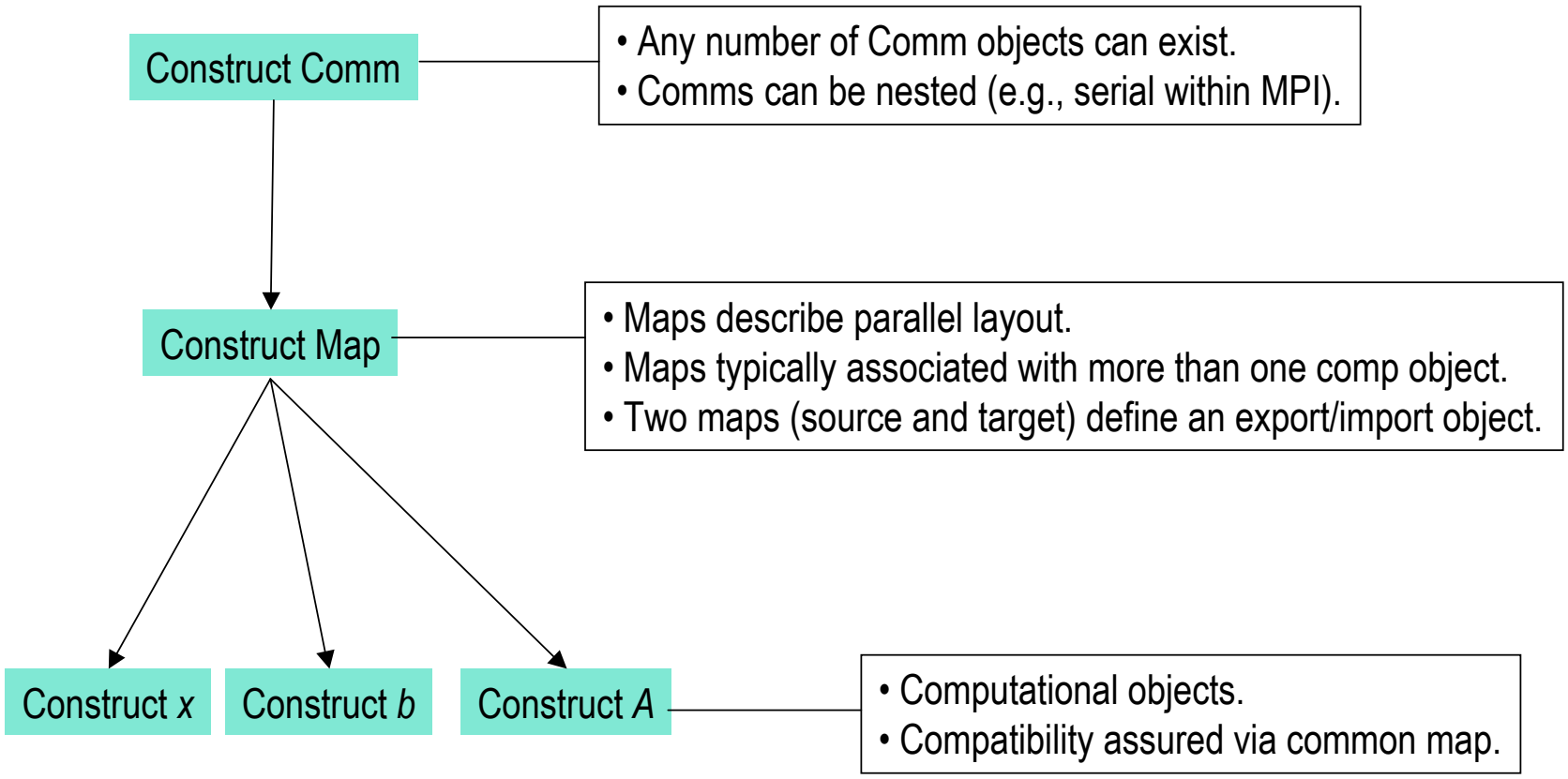
```
  // ***** Create x and b vectors *****
  Epetra_Vector x(Map);
  Epetra_Vector b(Map);
  b.Random(); // Fill RHS with random #s
```

```
// ***** Create Linear Problem *****
  Epetra_LinearProblem problem(&A, &x, &b);
```

```
  // ***** Create/define AztecOO instance, solve *****
  AztecOO solver(problem);
  solver.SetAztecOption(AZ_precond, AZ_Jacobi);
  solver.Iterate(1000, 1.0E-8);
```

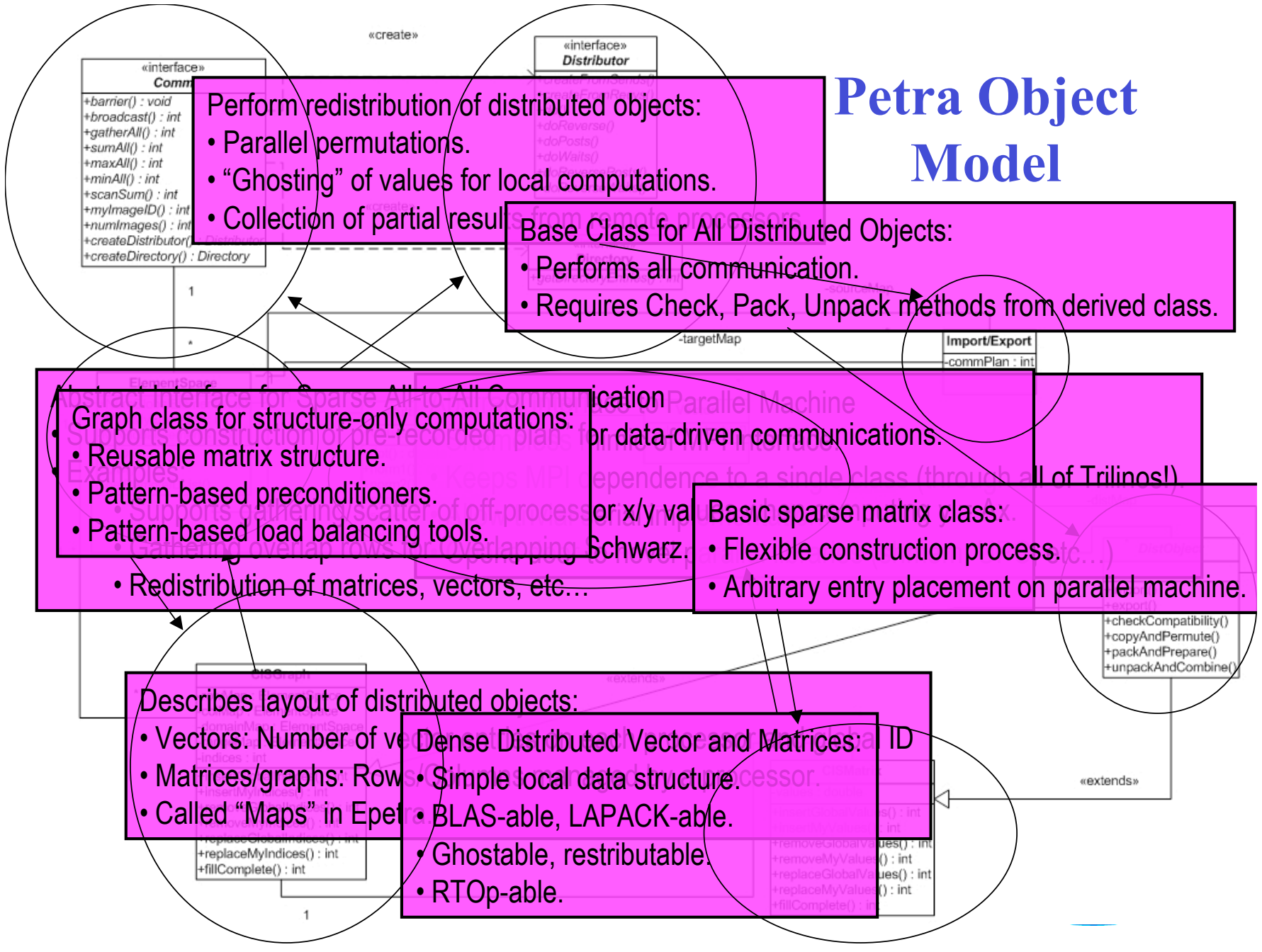
```
// ***** Report results, finish **********************
  cout << "Solver performed " << solver.NumIters()
       << " iterations." << endl
       << "Norm of true residual = "
       << solver.TrueResidual()
       << endl;


  return 0;
}
```

Typical Flow of Epetra Object Construction



Sandia National Laboratories

Petra Object Model



Petra Implementations

- Three version under development:
- **Epetra** (Essential Petra):
  - Current production version.
  - Restricted to real, double precision arithmetic.
  - Uses stable core subset of C++ (circa 2000).
  - Interfaces accessible to C and Fortran users.
- **Tpetra** (Templated Petra):
  - Next generation C++ version.
  - Templated scalar and ordinal fields.
  - Uses namespaces, and STL: Improved usability/efficiency.
- **Jpetra** (Java Petra):
  - Pure Java. Portable to any JVM.
  - Interfaces to Java versions of MPI, LAPACK and BLAS via interfaces.

Sandia National Laboratories

First Useful Abstraction: Comm

Sandia National Laboratories

Epetra Communication Classes

- Epetra_Comm is a pure virtual class:
  - Has no executable code: Interfaces only.
  - Encapsulates behavior and attributes of the parallel machine.
  - Defines interfaces for basic services such as:
    - Collective communications.
    - Gather/scatter capabilities.
  - Allows multiple parallel machine implementations.
- Implementation details of parallel machine confined to Comm classes.
- In particular, rest of Epetra (and rest of Trilinos) has no dependence on MPI.

Sandia National Laboratories

Comm Methods

- CreateDistributor() const=0 [pure virtual]
- CreateDirectory(const Epetra_BlockMap & map) const=0 [pure virtual]
- Barrier() const=0 [pure virtual]
- Broadcast(double *MyVals, int Count, int Root) const=0 [pure virtual]
- Broadcast(int *MyVals, int Count, int Root) const=0 [pure virtual]
- GatherAll(double *MyVals, double *AllVals, int Count) const=0 [pure virtual]
- GatherAll(int *MyVals, int *AllVals, int Count) const=0 [pure virtual]
- MaxAll(double *PartialMaxs, double *GlobalMaxs, int Count) const=0 [pure virtual]
- MaxAll(int *PartialMaxs, int *GlobalMaxs, int Count) const=0 [pure virtual]
- MinAll(double *PartialMins, double *GlobalMins, int Count) const=0 [pure virtual]
- MinAll(int *PartialMins, int *GlobalMins, int Count) const=0 [pure virtual]
- MyPID() const=0 [pure virtual]
- NumProc() const=0 [pure virtual]
- Print(ostream &os) const=0 [pure virtual]
- ScanSum(double *MyVals, double *ScanSums, int Count) const=0 [pure virtual]
- ScanSum(int *MyVals, int *ScanSums, int Count) const=0 [pure virtual]
- SumAll(double *PartialSums, double *GlobalSums, int Count) const=0 [pure virtual]
- SumAll(int *PartialSums, int *GlobalSums, int Count) const=0 [pure virtual]
- ~Epetra_Comm() [inline, virtual]

Sandia National Laboratories

Second Useful Abstraction: ElementSpace aka Map

Sandia National Laboratories

Map Classes

- Epetra maps prescribe the layout of distributed objects across the parallel machine.
- Typical map: 99 elements, 4 MPI processes could look like:
  - Number of elements = 25 on PE 0 through 2,
= 24 on PE 3.
  - GlobalElementList = {0, 1, 2, …, 24} on PE 0,
= {25, 26, …, 49} on PE 1. … etc.
- Funky Map: 10 elements, 3 MPI processes could look like:
  - Number of elements = 6 on PE 0,
= 4 on PE 1,
= 0 on PE 2.
  - GlobalElementList = {22, 3, 5, 2, 99, 54} on PE 0,
= { 5, 10, 12, 24} on PE 1,
= {} on PE 2.

Note: Global elements IDs (GIDs) are only labels:

- Need not be contiguous range on a processor.
- Need not be uniquely assigned to processors.
- Funky map is not unreasonable, given auto-generated meshes, etc.
- Use of a "Directory" facilitates arbitrary GID support.

Sandia National Laboratories

ElementSpace/Map

- Critical classes for efficient parallel execution. Provides ability to:
  - Generate families of compatible objects.
  - Quickly test if two objects have compatible layout.
  - Redistribute objects to make compatible layout (using Import/Export).

Sandia National Laboratories

Example: Quick Testing of Compatible Distributions

```
int dft_PolyA22_Epetra_Operator::Apply (const Epetra_MultiVector& X,
                                        Epetra_MultiVector& Y) const {

  TEST_FOR_EXCEPT(!X.Map().SameAs(OperatorDomainMap()));
  TEST_FOR_EXCEPT(!Y.Map().SameAs(OperatorRangeMap()));
```

- Fragment of code from Epetra_Operator adapter.
- Test for compatibility of maps.
- Lack of this ability is critical flaw of most parallel languages to date.

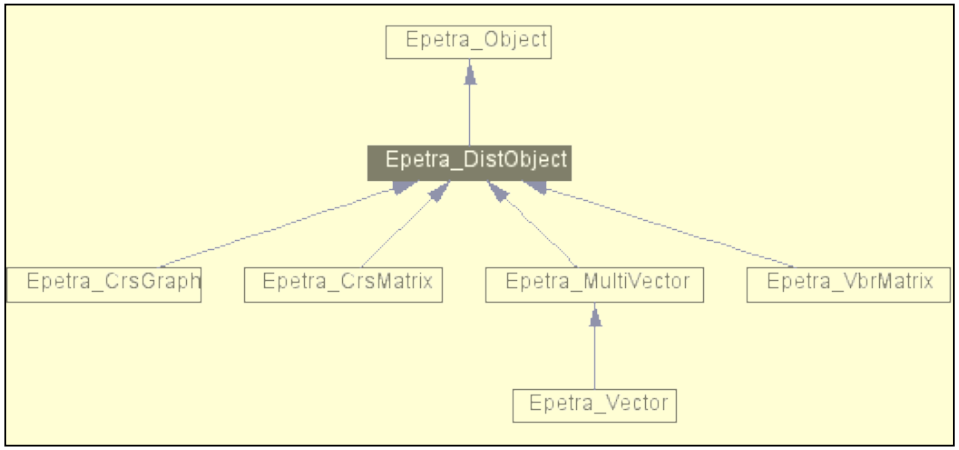
Sandia National Laboratories

Third Useful Abstraction: DistObject

Sandia National Laboratories

Epetra DistObject Base Class

- Some Epetra distributed object classes:
  - Vector
  - MultiVector
  - CrsGraph
  - CrsMatrix
  - VbrMatrix



- DistObject is a base class for all the above:
  - Construction of DistObject requires a Map (or BlockMap or LocalMap).
  - Has concrete methods for parallel data redistribution of an object.
  - Has virtual Pack/Unpack method that each derived class must implement.
- DistObject advantages:
  - Minimized redundant code.
  - Facilitates incorporation of other distributed objects in future.

Sandia National Laboratories

Epetra_DistObject Virtual Methods

virtual int CheckSizes (const Epetra_SrcDistObject &Source)=0

Allows the source and target (this) objects to be compared for compatibility, return nonzero if not.

virtual int CopyAndPermute (const Epetra_SrcDistObject &Source, int NumSameIDs,
int NumPermuteIDs, int *PermuteToLIDs, int *PermuteFromLIDs)=0

Perform ID copies and permutations that are on processor.

virtual int PackAndPrepare (const Epetra_SrcDistObject &Source, int NumExportIDs, int *ExportLIDs,
int Nsend, int Nrecv, int &LenExports, char *&Exports, int &LenImports,
char *&Imports, int &SizeOfPacket, Epetra_Distributor &Distor)=0

Perform any packing or preparation required for call to DoTransfer().

virtual int UnpackAndCombine (const Epetra_SrcDistObject &Source, int NumImportIDs, int *ImportLIDs,
char *Imports, int &SizeOfPacket, Epetra_Distributor &Distor,
Epetra_CombineMode CombineMode)=0

Perform any unpacking and combining after call to DoTransfer().

Sandia National Laboratories

Epetra_DistObject Import/Export Methods

int Import (const Epetra_SrcDistObject &A, const Epetra_Import &Importer, Epetra_CombineMode CombineMode)
Imports an Epetra_SrcDistObject using the Epetra_Import object.

int Import (const Epetra_SrcDistObject &A, const Epetra_Export &Exporter, Epetra_CombineMode CombineMode)
Imports an Epetra_SrcDistObject using the Epetra_Export object.

int Export (const Epetra_SrcDistObject &A, const Epetra_Import &Importer, Epetra_CombineMode CombineMode)
Exports an Epetra_SrcDistObject using the Epetra_Import object.

int Export (const Epetra_SrcDistObject &A, const Epetra_Export &Exporter, Epetra_CombineMode CombineMode)
Exports an Epetra_SrcDistObject using the Epetra_Export object.

Sandia National Laboratories

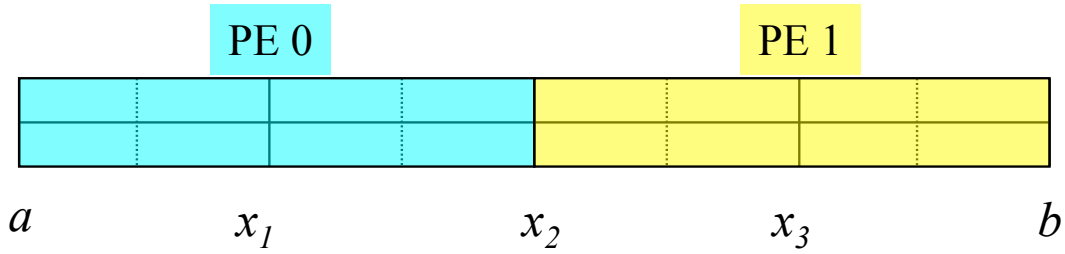
Import, Export and DistObject

- Work with any class that *isa* DistObject (or SrcDistObject).
- Abstraction of these operations to classes simplifies:
  - Optimal implementation on a variety of systems.
  - Reuse of complex parallel data redistribution:
    - DistObject has most of the complexity.

Sandia National Laboratories

Example: 1D Matrix Assembly

$$-u_{xx} = f$$
$$u(a) = \gamma_0$$
$$u(b) = \gamma_1$$



- 3 Equations: Find u at x_1, x_2 and x_3
- Equation for u at x_2 gets a contribution from PE 0 and PE 1.
- Would like to compute partial contributions independently.
- Then combine partial results.

Sandia National Laboratories

Two Maps

- We need two maps:
  - Assembly map:
    - PE 0: { 1, 2 }.
    - PE 1: { 2, 3 }.
  - Solver map:
    - PE 0: { 1, 2 } (we arbitrate ownership of 2).
    - PE 1: { 3 }.

Sandia National Laboratories

End of Assembly Phase

- At the end of assembly phase we have AssemblyMatrix:

  AssemblyMatrix *AssemblyRhs*

  - On PE 0:

    Equation 1: $\begin{bmatrix} 2 & -1 & 0 \end{bmatrix}$ $\begin{bmatrix} 1 \end{bmatrix}$

    Equation 2: $\begin{bmatrix} -1 & 1 & 0 \end{bmatrix}$ $\begin{bmatrix} 1 \end{bmatrix}$

  - On PE 1:

    Equation 2: $\begin{bmatrix} 0 & 1 & -1 \end{bmatrix}$ $\begin{bmatrix} 1 \end{bmatrix}$

    Equation 3: $\begin{bmatrix} 0 & -1 & 2 \end{bmatrix}$ $\begin{bmatrix} 1 \end{bmatrix}$

- Want to assign all of Equation 2 to PE 0 for use with solver.

Sandia National Laboratories

Export Assembly Matrix to Solver Matrix

```
Epetra_Export Exporter(AssemblyMap, SolverMap);

Epetra_CrsMatrix SolverMatrix (Copy, SolverMap, 0);

SolverMatrix.Export(AssemblyMatrix, Exporter, Add);

SolverMatrix.TransformToLocal();
```

Sandia National Laboratories

End of Export Phase

- At the end of Export phase we have SolverMatrix:
  - On PE 0:

SolverMatrix *SolverRhs*

On PE 1:

Equation 1: $\begin{bmatrix} 2 & -1 & 0 \end{bmatrix}$ $\begin{bmatrix} 1 \end{bmatrix}$

Equation 2: $\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$ $\begin{bmatrix} 2 \end{bmatrix}$

Equation 3: $\begin{bmatrix} 0 & -1 & 2 \end{bmatrix}$ $\begin{bmatrix} 1 \end{bmatrix}$

- Each row is uniquely owned.

Sandia National Laboratories

Need for Import/Export

- Solvers for complex engineering applications need expressive, easy-to-use parallel data redistribution:
  - Allows better scaling for non-uniform overlapping Schwarz.
  - Necessary for robust solution of multiphysics problems.
- We have found import and export facilities to be a very natural and powerful technique to address these issues.

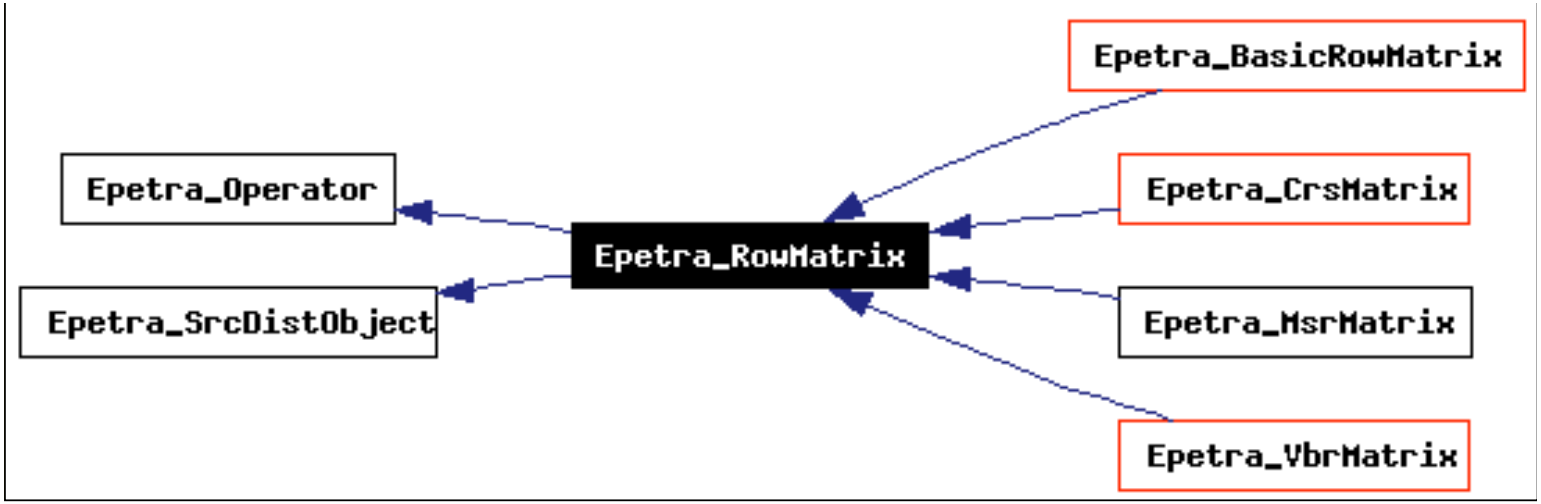
Sandia National Laboratories

Other uses for Import/Export

- In addition, import and export facilities provide a variety of other capabilities:
  - Communication needs of sparse matrix multiplication.
  - Parallel assembly: Shared nodes receive contributions from multiple processors, reverse operation replicates results back.
  - Higher order interpolations are easy to implement.
  - Ghost node distributions.
  - Changing work loads can be re-balanced.
  - Sparse matrix transpose become trivial to implement.
  - Allows gradual MPI-izing of an application.
  - Cached Overlapped distributed vectors (generalization of distributed sparse MV).
  - Rendezvous algorithms easy to implement.

Sandia National Laboratories

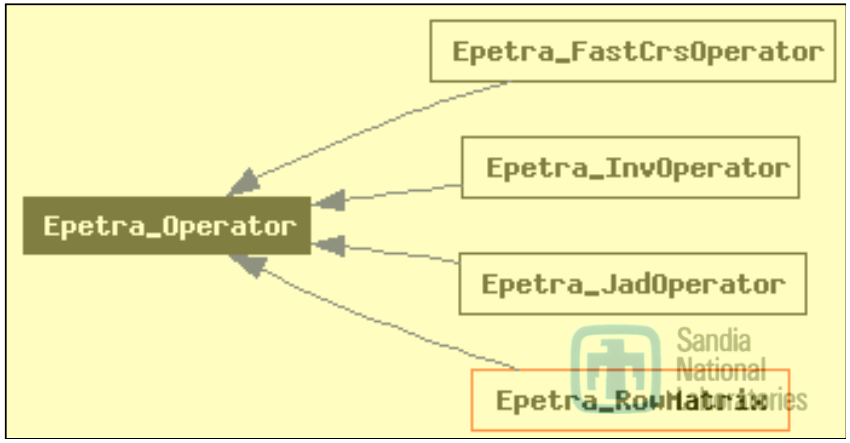
Fourth Useful Abstraction: Coarse-grain linear algebra objects



Notes:

- **All Trilinos solvers can access linear operators via:**
  - **Epetra_Operator if no coefficients needed.**
  - **Epetra_RowMatrix if coefficients.**
  - **All Epetra matrix and operator class implement these two interfaces.**

Epetra_Operator and Epetra_RowMatrix



Summary

- Abstract machine model provides flexibility:
  - Required for Portability.
  - Enables state retention.
  - Allows specialized extensibility.
- Abstraction of distribution is dssential:
  - Required for efficient loop execution.
  - Supports easy construction of compatibly distributed objects.
  - Enables efficient redistribution mechanisms.
- Coarse-grain abstraction enables use of a range of computer systems:
  - (Within SPMD model) abstractions like Epetra_Operator seem to give us a fighting chance on incremental and novel architectures.
- (In the solvers area) Nested Parallelism is a Dubious Option:
  - Been there, done that (circa 95-96).
  - Limited opportunity for improvement.
  - Many opportunities to hinder performance.
  - Could (potentially) help with load imbalance.

Sandia National Laboratories