SAND2014-18749PE

An Overview of Trilinos

Mark Hoemmen
Sandia National Laboratories
30 June 2014

Sandia is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U. S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Schedule

Session 1: Trilinos overview
Session 2: Hands-on Trilinos tutorial
Session 3: Kokkos overview and tutorial
Session 4: Hands-on, audience-directed topics

Outline

- What can Trilinos do for you?
- Trilinos' software organization
- Whirlwind tour of Trilinos packages
- Getting started: "How do I…?"
- Preparation for hands-on tutorial

What can Trilinos do for you?

What is Trilinos?

- Object-oriented software framework for…
- Solving big complex science & engineering problems
- More like LEGO™ bricks than Matlab™

Applications

- All kinds of physical simulations:
  - Structural mechanics (statics and dynamics)
  - Circuit simulations (physical models)
  - Electromagnetics, plasmas, and superconductors
  - Combustion and fluid flow (at macro- and nanoscales)
- Coupled / multiphysics models
- Data and graph analysis
  - Even gaming!

Target platforms:
Any and all, current and future

- Laptops and workstations
- Clusters and supercomputers
  - Multicore CPU nodes
  - Hybrid CPU / GPU nodes
- Parallel programming environments
  - MPI, OpenMP, Pthreads, Intel TBB
  - CUDA (for NVIDIA GPUs)
  - Combinations of the above
- User “skins”
  - C++ (primary language)
  - C, Fortran, Python
  - Web (Hands-on demo)

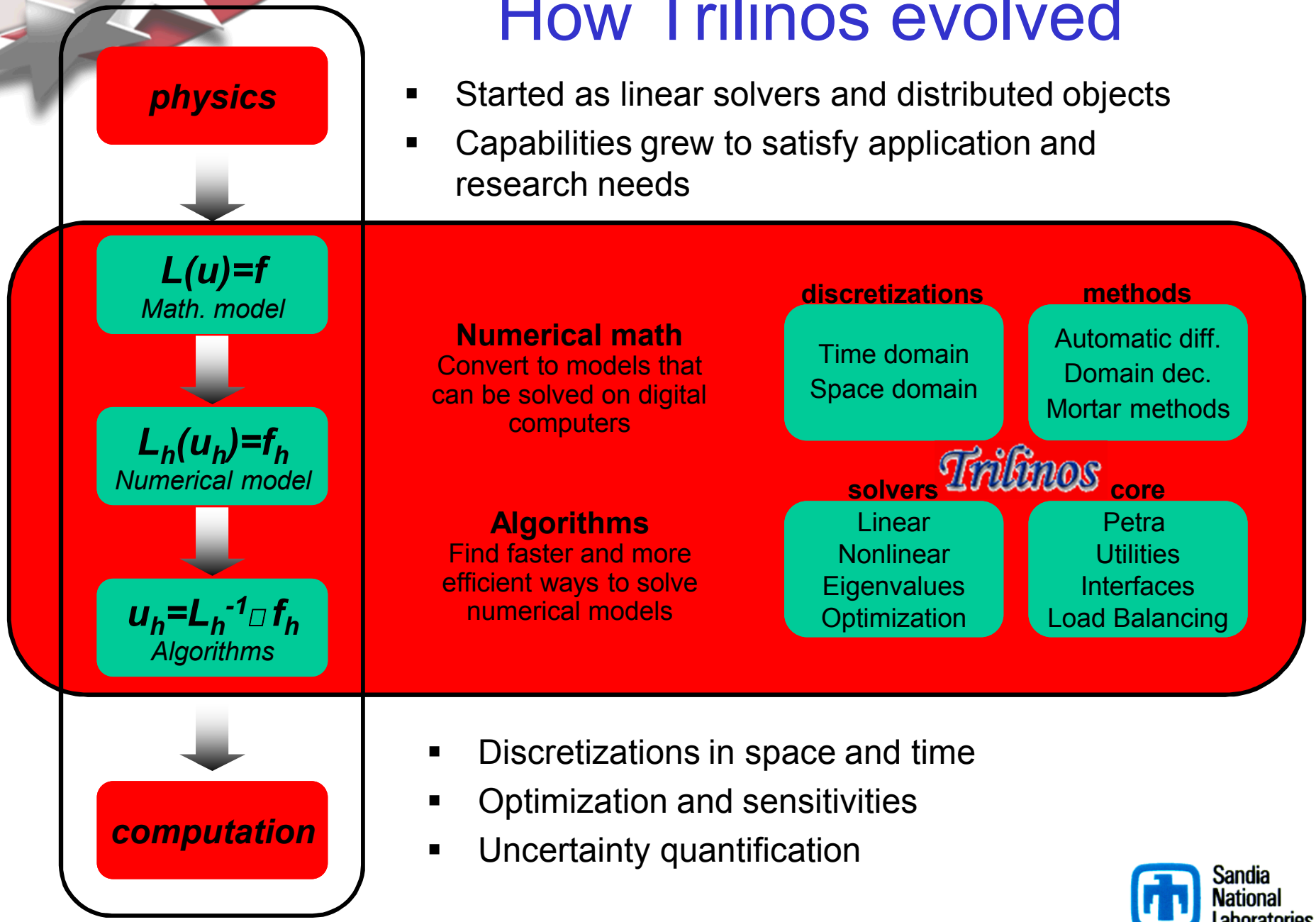
Sandia National Laboratories

Unique features of Trilinos

- Huge library of algorithms
  - Linear and nonlinear solvers, preconditioners, …
  - Optimization, transients, sensitivities, uncertainty, …
- Support for huge (> 2B unknowns) problems
- Support for mixed and arbitrary precisions
- Growing support for hybrid (MPI+X) parallelism
  - X: CPU threads (multicore or Intel Xeon Phi) or Nvidia GPU
  - Built *on* a unified shared-memory parallel programming model: Kokkos (see Session 3)
  - Support currently limited, but growing fast.

How Trilinos evolved

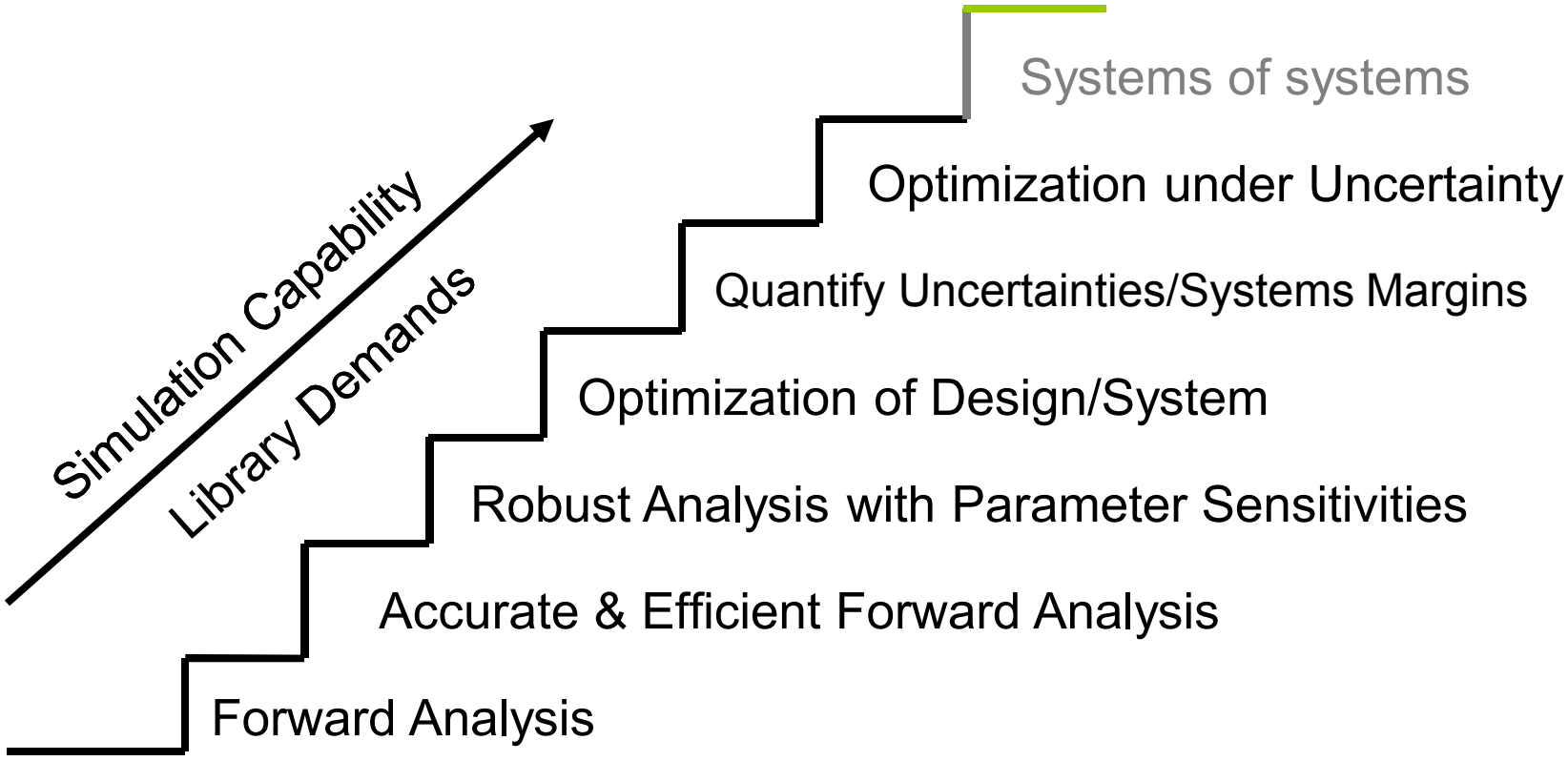
- Started as linear solvers and distributed objects
- Capabilities grew to satisfy application and research needs



- Discretizations in space and time
- Optimization and sensitivities
- Uncertainty quantification

Sandia National Laboratories

From Forward Analysis, to Support for High-Consequence Decisions



Each stage requires *greater performance* and *error control* of prior stages:
Always will need: more accurate and scalable methods.
more sophisticated tools.

Trilinos strategic goals

- Algorithmic goals
  - *Scalable* computations (at all levels of parallelism)
  - *Hardened* computations
    - Fail only if problem intractable
    - Diagnose failures & inform the user
  - Full vertical coverage
    - Problem construction, solution, analysis, & optimization
- Software goals
  - Universal interoperability (within & outside Trilinos)
  - Universal accessibility
    - Any hardware & operating system with a C++ compiler
    - Including programming languages besides C++
  - “Self-sustaining” software
    - Clean design & implementation
    - Sufficient testing & documentation for confident refactoring

Trilinos' software organization

Trilinos is made of packages

- Not a monolithic piece of software
  - Like LEGO™ bricks, not Matlab™
- Each package
  - Has its own development team and management
  - Makes its own decisions about algorithms, coding style, etc.
  - May or may not depend on other Trilinos packages
  - May even have a different license (most are BSD) or release status
  - Benefits from Trilinos build and test infrastructure
- Trilinos is not “indivisible”
  - You don’t need all of Trilinos to get things done
  - Don’t feel overwhelmed by large number (~55) of packages!
  - Any subset of packages can be combined and distributed
- Trilinos top layer framework (TriBITS)
  - Manages package dependencies
  - Runs packages’ tests nightly, and on every check-in
  - Useful: spun off from Trilinos into a separate project

Why packages?

- Users decide how much of Trilinos they want to use
  - Only use and build the packages you need
  - Mix and match Trilinos components with your own, e.g.,
    - Trilinos sparse matrices with your own linear solvers
    - Your sparse matrices with Trilinos' linear solvers
    - Trilinos sparse matrices & linear solvers with your nonlinear solvers
    - …
- Popular packages keep their "brand"
  - Well-known packages like ML & Zoltan still stand alone
  - But benefit from Trilinos build & test infrastructure
- Reflects organization of research / development teams
  - Easy to turn a research code into a new package
  - Keeps team sizes small for more productivity
  - "Multifrontal development" (minimizes interference)

Interoperability vs. Dependence

("Can Use") ("Depends On")

- Packages have minimal required dependencies…
- But interoperability makes them useful:
  - NOX (nonlinear solver) needs linear solvers
    - Can use any of {AztecOO, Belos, LAPACK, …}
  - Belos (linear solver) needs preconditioners, matrices, and vectors
    - Matrices and vectors: any of {Epetra, Tpetra, Thyra, …, PETSc}
    - Preconditioners: any of {IFPACK, ML, Ifpack2, MueLu, Teko, …}
- Interoperability is enabled at configure time
  - Each package declares its list of interoperable packages
  - Trilinos' build system automatically hooks them together
  - You can ask to build only specific packages
    - Trilinos will enable only those packages needed to make them build
- External compatible development
  - DTK (https://github.com/CNERG/DataTransferKit)
  - Parallel distributed data transfer engine, Stuart Slattery

Capability areas and leaders

- Capability areas:
  - Framework, Tools & Interfaces (Jim Willenbring)
  - Software Engineering Technologies and Integration (Ross Bartlett)
  - Discretizations (Pavel Bochev)
  - Geometry, Meshing & Load Balancing (Karen Devine)
  - Scalable Linear Algebra (Mike Heroux)
  - Linear & Eigen Solvers (Jonathan Hu)
  - Nonlinear, Transient & Optimization Solvers (Andy Salinger)
  - Scalable I/O (Ron Oldfield)
  - User Experience (Bill Spotz)
- Each area includes one or more Trilinos packages
- Each leader provides strategic direction within area

Whirlwind Tour of Packages

Full Vertical Solver Coverage

Trilinos

Optimization **Unconstrained:** **Constrained:**	Find $u \in \Re^n$ that minimizes $g(u)$ Find $x \in \Re^m$ and $u \in \Re^n$ that minimizes $g(x,u)$ s.t. $f(x,u) = 0$	**Sensitivities (Automatic Differentiation: Sacado)**	**MOOCHO**
Bifurcation Analysis	Given nonlinear operator $F(x,u) \in \Re^{n+m}$ For $F(x,u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$		**LOCA**
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	Given nonlinear operator $F(x) \in \Re^m \rightarrow \Re$ Solve $F(x) = 0$ $x \in \Re^n$		**NOX**
Linear Problems **Linear Equations:** **Eigen Problems:**	Given Linear Ops (Matrices) $A, B \in \Re^{m \times n}$ Solve $Ax = b$ for $x \in \Re^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \Re^n$, $\lambda \in$		**AztecOO** **Belos** **Ifpack, ML, etc...** **Anasazi**
Distributed Linear Algebra **Matrix/Graph Equations** **Vector Problems:**	Compute $y = Ax; A = A(G); A \in \Re^{m \times n}, G \in \Im^{m \times n}$ Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \Re^n$		**Epetra** **Tpetra** **Kokkos**

Trilinos Package Summary

	Objective	Package(s)
Discretizations	Meshing & Discretizations	STKMesh, Intrepid, Pamgen, Sundance, Mesquite
	Time Integration	Rythmos
Methods	Automatic Differentiation	Sacado
	Mortar Methods	Moertel
Services	Linear algebra objects	Epetra, Tpetra
	Interfaces	Xpetra, Thyra, Stratimikos, RTOp, FEI, Shards
	Load Balancing	Zoltan, Isorropia, Zoltan2
	“Skins”	PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika
	Utilities, I/O, thread API	Teuchos, EpetraExt, Kokkos, Triutils, ThreadPool, Phalanx
Solvers	Iterative linear solvers	AztecOO, Belos, Komplex
	Direct sparse linear solvers	Amesos, Amesos2, ShyLU
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi
	Incomplete factorizations	AztecOO, IFPACK, Ifpack2
	Multilevel preconditioners	ML, CLAPS, MueLu
	Block preconditioners	Meros, Teko
	Nonlinear solvers	NOX, LOCA
	Optimization	MOOCHO, Aristos, TriKota, Globipack, Optipack
	Stochastic PDEs	Stokhos

Two solver stacks: Epetra & Tpetra

- Many packages built on Epetra linear algebra interface
  - Common “solver stack” for Epetra sparse matrices & vectors
- Users want features that break interfaces
  - Support for solving huge problems (> 2B entities)
  - Arbitrary and mixed precision
  - Hybrid (MPI+X) parallelism (← most radical interface changes)
- Users also value backwards compatibility
- We decided to build a (partly) new stack using Tpetra
  - MPI+X – friendly interfaces go into Tpetra stack
  - Epetra got some support for huge problems (“Epetra64”)
  - Some packages can work with either Epetra or Tpetra
    - Iterative linear solvers & eigensolvers (Belos, Anasazi)
    - Multilevel preconditioning (MueLu), sparse direct (Amesos2)

Whirlwind Tour of Packages

Discretizations Methods Core Solvers

Teuchos

- Portable utility package of commonly useful tools
  - ParameterList: nested (key, value) database (more later)
  - Generic LAPACK and BLAS wrappers
  - Local dense matrix and vector classes
  - Memory management classes (more later)
  - Scalable parallel timers and statistics
  - Support for generic algorithms (traits classes)
- Help make Trilinos work on as many platforms as possible
  - Protect algorithm developers from platform differences
  - Not all compilers could build Boost in the mid-2000s
  - BLAS and LAPACK Fortran vs. C calling conventions
  - Different sizes of integers on different platforms
- You'll see this package a lot

Package lead: Roscoe Barlett (many developers)

Trilinos Common Language: Petra

- "Common language" for distributed linear algebra objects (operator, sparse matrix, dense vectors)

- Petra[1] provides distributed matrix and vector services
- Object model
  - Describes basic user and support classes in UML, independent of language/implementation
  - Describes objects and relationships to build and use matrices, vectors and graphs
- Has 2 implementations under active development

[1]Petra is Greek for "foundation".

Petra Implementations

- Epetra (Essential Petra):
  - Earliest and most heavily used
  - C++ circa 1998 (“C+/- compilers” OK)
  - Real, double-precision arithmetic
  - Interfaces accessible to C and Fortran users

- Tpetra (Templated Petra):
  - C++ circa mid-2000s (no C++11)
  - Supports arbitrary scalar and index types via templates
    - Arbitrary- and mixed-precision arithmetic
    - 64-bit indices for solving problems with >2 billion unknowns
  - Hybrid MPI / shared-memory parallel
    - Supports multicore CPU and hybrid CPU/GPU
    - Built on Kokkos shared-memory parallel programming model

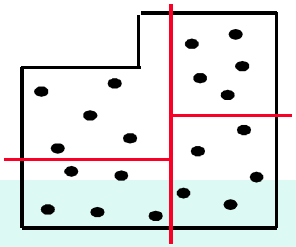
Package leads: Mike Heroux, Mark Hoemmen (many developers)

EpetraExt: Extensions to Epetra

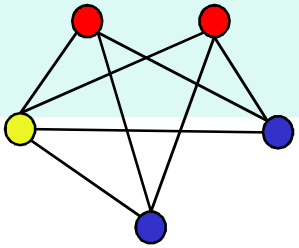
- Library of useful classes not needed by everyone
- Most classes are types of “transforms”.
- Examples:
  - Graph/matrix view extraction.
  - Epetra/Zoltan interface.
  - Explicit sparse transpose.
  - Singleton removal filter, static condensation filter.
  - Overlapped graph constructor, graph colorings.
  - Permutations.
  - Sparse matrix-matrix multiply.
  - Matlab, MatrixMarket I/O functions.
- Most classes are small, useful, but non-trivial to write.
- Migrating to 64-bit ints

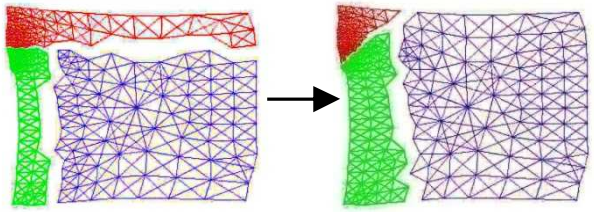
Developer: Robert Hoekstra, Alan Williams, Mike Heroux, Chetan Jhurani

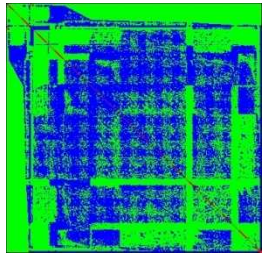
Zoltan(2)



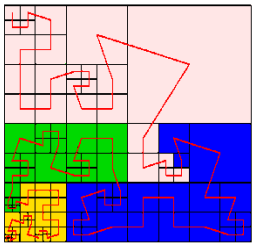
- Data Services for Dynamic Applications
  - Dynamic load balancing
  - Graph coloring
  - Data migration
  - Matrix ordering



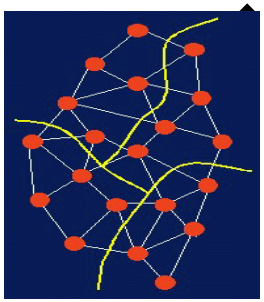




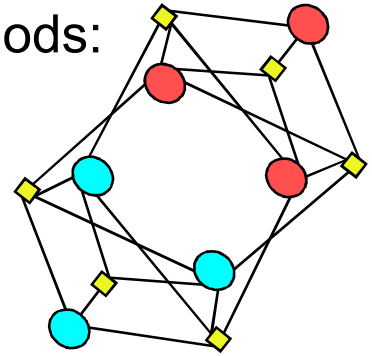
- Partitioners:



Geometric (coordinate-based) methods:
- Recursive Coordinate Bisection (Berger, Bokhari)
- Recursive Inertial Bisection (Taylor, Nour-Omid)
- Space Filling Curves (Peano, Hilbert)
- Refinement-tree Partitioning (Mitchell)



Hypergraph and graph (connectivity-based) methods:
- Hypergraph Repartitioning PaToH (Catalyurek)
- Zoltan Hypergraph Partitioning
- ParMETIS (U. Minnesota)
- Jostle (U. Greenwich)



Isorropia package: interface to Epetra objects

Developers: Karen Devine, Erik Boman, Siva Rajamanickam, Michael Wolf

Sandia National Laboratories

Thyra

- Abstract linear algebra interfaces
- Offers flexibility through abstractions to algorithm developers
- Linear solvers (Direct, Iterative, Preconditioners)
  - Use abstraction of basic matrix & vector operations
  - Can use any concrete linear algebra library (Epetra, Tpetra, …)
- Nonlinear solvers (Newton, etc.)
  - Use abstraction of linear solve
  - Can use any concrete linear solver library and preconditioners
- Transient/DAE solvers (implicit)
  - Use abstraction of nonlinear solve
- Thyra is how Stratimikos talks to data structures & solvers

Developers: Roscoe Bartlett, Kevin Long

"Skins"

- PyTrilinos provides Python access to Trilinos packages
- Uses SWIG to generate bindings.
- Support for many packages

Developer: Bill Spotz

- CTrilinos: C wrapper (mostly to support ForTrilinos).
- ForTrilinos: OO Fortran interfaces.

Developers: Nicole Lemaster, Damian Rouson

- WebTrilinos: Web interface to Trilinos
- Generate test problems or read from file.
- Generate C++ or Python code fragments and click-run.
- Hand modify code fragments and re-run.
- Will use during hands-on.

Developers: Ray Tuminaro, Jonathan Hu, Marzio Sala, Jim Willenbring

Whirlwind Tour of Packages

Discretizations Methods Core Solvers

Kokkos: Node-level Data Classes

- Manycore/Accelerator data structures & kernels
- Epetra is MPI-only, Tpetra is MPI[+X[+Y]].
- Kokkos parallel dispatch and data structures
  - See Session 3 for details
  - Multi-dimensional arrays, hash table, sparse graph & matrix
  - Hide physical data layout & target it to the hardware
- "Pretty Good Kernels"
  - Computational kernels for sparse and dense matrices
  - Written generically to Kokkos, not to specific hardware
    - "How I learned to stop worrying and love CSR"
  - Replaceable with vendor-optimized libraries

Developer: Carter Edwards, Mark Hoemmen, Dan Sunderland, Christian Trott

Amesos

- Interface to direct solvers for distributed sparse linear systems (KLU, UMFPACK, SuperLU, MUMPS, ScaLAPACK)

- Challenges:
  - No single solver dominates
  - Different interfaces and data formats, serial and parallel
  - Interface often changes between revisions
- Amesos offers:
  - A single, clear, consistent interface, to various packages
  - Common look-and-feel for all classes
  - Separation from specific solver details
  - Use serial and distributed solvers; Amesos takes care of data redistribution
  - Native solvers: KLU and Paraklete

Developers: Ken Stanley, Marzio Sala, Tim Davis

Amesos2

- Second-generation sparse direct solvers package

- Unified interface to multiple solvers, just like Amesos
- Supports matrices of arbitrary scalar and index types
- Path to multicore CPU and hybrid CPU/GPU solvers
- Multiple solvers can coexist in the same MPI process
  - Supports new “hybrid / hybrid” direct / iterative solver ShyLU
- Abstraction from specific sparse matrix representation
  - Accepts Epetra or Tpetra sparse matrices
  - Extensible to other matrix types

Developers: Eric Bavier, Erik Boman, and Siva Rajamanickam

AztecOO

- Krylov subspace solvers: CG, GMRES, BiCGSTAB,…
- Incomplete factorization preconditioners

- Aztec was Sandia's workhorse solver:
  - Extracted from the MPSalsa reacting flow code
  - Installed in dozens of Sandia apps
  - 1900+ external licenses
- AztecOO improves on Aztec by:
  - Using Epetra objects for defining matrix and vectors
  - Providing more preconditioners/scalings
  - Using C++ class design to enable more sophisticated use
- AztecOO interface allows:
  - Continued use of Aztec for functionality
  - Introduction of new solver capabilities outside of Aztec

Developers: Mike Heroux, Alan Williams, Ray Tuminaro

Belos

- Next-generation linear iterative solvers

- Decouples algorithms from linear algebra objects
  - Linear algebra library has full control over data layout and kernels
  - Improvement on “reverse communication” interface of Aztec
  - Essential for hybrid (MPI+X) parallelism
- Solves problems that apps really want to solve, faster:
  - Multiple right-hand sides: AX=B
  - Sequences of related systems: $(A + \Delta A_k) X_k = B + \Delta B_k$
- Many advanced methods for these types of systems
  - Block methods: Block GMRES and Block CG
  - Recycling solvers: GCRODR (GMRES) and CG
  - “Seed” solvers (hybrid GMRES)
  - Block orthogonalizations (TSQR)
- Supports arbitrary and mixed precision, and complex

Developers: Heidi Thornquist, Mike Heroux, Mark Hoemmen, Mike Parks, …

Sandia National Laboratories

Ifpack(2): Algebraic preconditioners

- Preconditioners:
  - Overlapping domain decomposition
  - Incomplete factorizations (within an MPI process)
  - (Block) relaxations & Chebyshev
- Accepts user matrix via abstract matrix interface
- Uses Epetra for basic matrix/vector calculations
- Perturbation stabilizations & condition estimation
- Can be used by NOX, ML, AztecOO, Belos, …
- Ifpack2: Tpetra version of Ifpack
  - Supports arbitrary precision & complex arithmetic
  - Path forward to hybrid-parallel factorizations

Developers: Mike Heroux, Mark Hoemmen, Siva Rajamanickam, Marzio Sala, Alan Williams, etc.

: Multi-level Preconditioners

- Smoothed aggregation, multigrid and domain decomposition preconditioning package

- Critical technology for scalable performance of many apps
- ML compatible with other Trilinos packages:
  - Accepts user data as Epetra_RowMatrix object (abstract interface). Any implementation of Epetra_RowMatrix works.
  - Implements the Epetra_Operator interface. Allows ML preconditioners to be used with AztecOO, Belos, Anasazi.
- Can also be used independent of other Trilinos packages
- Next-generation ML package: MueLu
  - Works with Epetra or Tpetra objects (via Xpetra interface)

Developers: Ray Tuminaro, Jeremie Gaidamour, Jonathan Hu, Marzio Sala, Chris Siefert

MueLu: Next-gen algebraic multigrid

- Motivation for replacing ML
  - Improve maintainability & ease development of new algorithms
  - Decouple computational kernels from algorithms
    - ML mostly monolithic (& 50K lines of code)
    - MueLu relies more on other Trilinos packages
  - Exploit Tpetra features
    - MPI+X (Kokkos programming model mitigates risk)
    - 64-bit global indices (to solve problems with >2B unknowns)
    - Arbitrary Scalar types (Tramonto runs MueLu w/ double-double)
- Works with Epetra or Tpetra (via Xpetra common interface)
- Facilitate algorithm development
  - Energy minimization methods
  - Geometric or classic algebraic multigrid; mix methods together
- Better support for preconditioner reuse
  - Explore options between “blow it away” & reuse without change

Developers: Andrey Prokopenko, Jonathan Hu, Chris Siefert, Ray Tuminaro, Tobias Wiesner

Anasazi

- Next-generation iterative eigensolvers

- Decouples algorithms from linear algebra objects
  - Like Belos, except that Anasazi came first
- Block eigensolvers for accurate cluster resolution
- Can solve
  - Standard ($AX = \Lambda X$) or generalized ($AX = BX\Lambda$)
  - Hermitian or not, real or complex
- Algorithms available
  - Block Krylov-Schur (most like ARPACK's IR Arnoldi)
  - Block Jacobi-Davidson; TraceMin in progress
  - Locally Optimal Block-Preconditioned CG (LOBPCG)
  - Implicit Riemannian Trust Region solvers
  - Scalable orthogonalizations (e.g., TSQR, SVQB)

Developers: Heidi Thornquist, Mike Heroux, Chris Baker, Rich Lehoucq, Ulrich Hetmaniuk, Mark Hoemmen

Sandia National Laboratories

NOX: Nonlinear Solvers

- Suite of nonlinear solution methods

Broyden's Method

$$M_B = f(x_c) + B_c d$$

Newton's Method

$$M_N = f(x_c) + J_c d$$

Tensor Method

$$M_T = f(x_c) + J_c d + \frac{1}{2} T_c d d$$

Jacobian Estimation
- Graph Coloring
- Finite Difference
- Jacobian-Free Newton-Krylov

Globalizations

Line Search	Trust Region
Interval Halving	Dogleg
Quadratic	Inexact Dogleg
Cubic	
More'-Thuente	

Implementation
- Parallel
- OO-C++
- Independent of the linear algebra package!

http://trilinos.sandia.gov/packages/nox

Developers: Tammy Kolda, Roger Pawlowski

Sandia National Laboratories

LOCA

- Library of continuation algorithms

- Provides
  - Zero order continuation
  - First order continuation
  - Arc length continuation
  - Multi-parameter continuation (via Henderson's MF Library)
  - Turning point continuation
  - Pitchfork bifurcation continuation
  - Hopf bifurcation continuation
  - Phase transition continuation
  - Eigenvalue approximation (via ARPACK or Anasazi)

Developers: Andy Salinger, Eric Phipps

MOOCHO & Aristos

- MOOCHO: Multifunctional Object-Oriented arCHitecture for Optimization
  - Large-scale invasive simultaneous analysis and design (SAND) using reduced space SQP methods.

Developer: Roscoe Bartlett

- Aristos: Optimization of large-scale design spaces
  - Invasive optimization approach
  - Based on full-space SQP methods
  - Efficiently manages inexactness in the inner linear solves

Developer: Denis Ridzal

Whirlwind Tour of Packages

Core Utilities

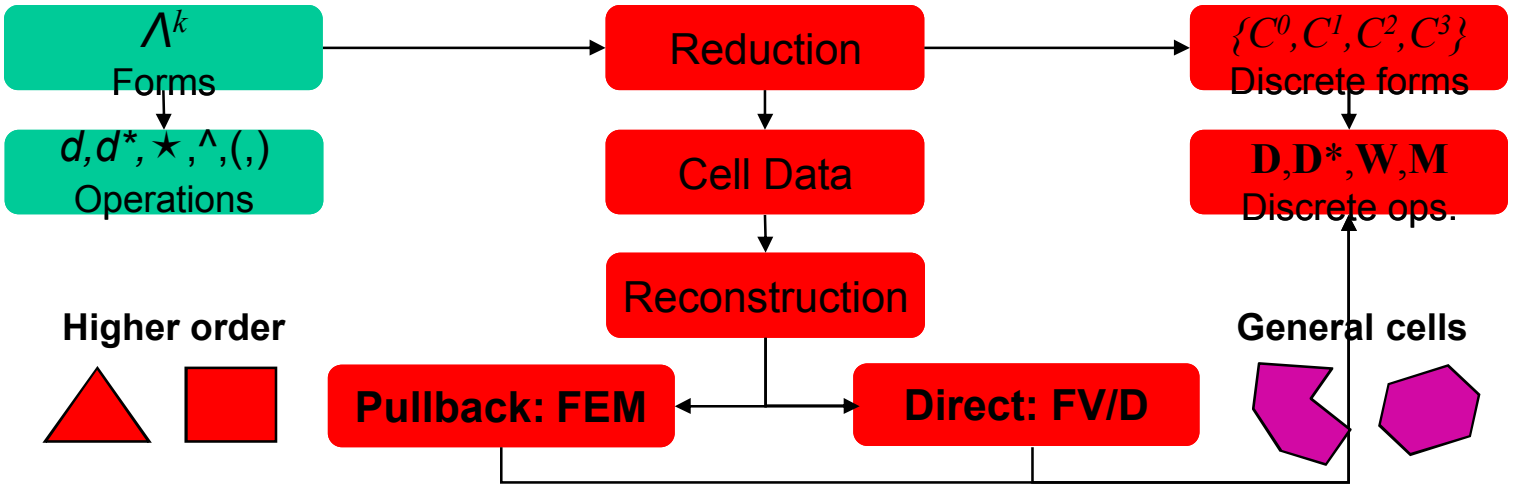
Discretizations Methods Solvers

Intrepid

Interoperable Tools for Rapid Development of Compatible Discretizations

Intrepid offers an **innovative software design** for compatible discretizations:

- Access to finite {element, volume, difference} methods using a common API
- Supports hybrid discretizations (FEM, FV and FD) on unstructured grids
- Supports a variety of cell shapes:
  - Standard shapes (e.g., tets, hexes): high-order finite element methods
  - Arbitrary (polyhedral) shapes: low-order mimetic finite difference methods
- Enables optimization, error estimation, V&V, and UQ using fast invasive techniques (direct support for cell-based derivative computations or via automatic differentiation)



Developers: Pavel Bochev and Denis Ridzal

Rythmos

- Suite of time integration (discretization) methods
- Supported methods include
  - Backward and Forward Euler
  - Explicit Runge-Kutta
  - Implicit BDF
- Operator splitting methods & sensitivities
- Revived by Glen Hansen and under active development

Developers: Glen Hansen, Roscoe Bartlett, Todd Coffey

Whirlwind Tour of Packages

Discretizations Methods Core Solvers

Sacado: Automatic Differentiation

- Automatic differentiation tools optimized for element-level computation

- Applications of AD: Jacobians, sensitivity and uncertainty analysis, …
- Uses C++ templates to compute derivatives
  - You maintain one templated code base; derivatives don't appear explicitly
- Provides three forms of AD
  - Forward Mode: $(x,\ V) \longrightarrow \left(f,\ \frac{\partial f}{\partial x} V\right)$
    - Propagate derivatives of intermediate variables w.r.t. independent variables forward
    - Directional derivatives, tangent vectors, square Jacobians, ∂f / ∂x when **m ≥ n**
  - Reverse Mode: $(x,\ W) \longrightarrow \left(f,\ W^T \frac{\partial f}{\partial x}\right)$
    - Propagate derivatives of dependent variables w.r.t. intermediate variables backwards
    - Gradients, Jacobian-transpose products (adjoints), ∂f / ∂x when **n > m**.
  - Taylor polynomial mode: $x(t) = \sum_{k=0}^{d} x_k t^k \longrightarrow \sum_{k=0}^{d} f_k t^k = f(x(t)) + O(t^{d+1}),\quad f_k = \frac{1}{k!}\frac{d^k}{dt^k} f(x(t))$
  - Basic modes combined for higher derivatives

Developers: Eric Phipps, David Gay

Sandia National Laboratories

Solver collaborations: Abstract interfaces and applications

Categories of Abstract Problems and Abstract Algorithms

Trilinos Packages

- Linear Problems: Given linear operator (matrix) $A \in \mathbf{R}^{n \times n}$
  - Linear equations: Solve $Ax = b$ for $x \in \mathbf{R}^n$ — Belos
  - Eigen problems: Solve $Av = \lambda v$ for (all) $v \in \mathbf{R}^n$ and $\lambda \in \mathbf{R}$ — Anasazi
- Nonlinear Problems: Given nonlinear operator $c(x, u) \in \mathbf{R}^{n+m} \rightarrow \mathbf{R}^n$
  - Nonlinear equations: Solve $c(x) = 0$ for $x \in \mathbf{R}^n$ — NOX
  - Stability analysis: For $c(x, u) = 0$ find space $u \in \mathcal{U}$ such that $\frac{\partial c}{\partial x}$ is singular — LOCA
- Transient Nonlinear 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 $x(t) \in \mathbf{R}^n, t \in [0, T]$ — Rythmos
- Optimization Problems:
  - Unconstrained: Find $u \in \mathbf{R}^n$ that minimizes $f(u)$ — MOOCHO
  - Constrained: Find $y \in \mathbf{R}^m$ and $u \in \mathbf{R}^n$ that: minimizes $f(y, u)$ such that $c(y, u) = 0$ — Aristos

Sandia National Laboratories

Abstract Numerical Algorithms

An **abstract numerical algorithm** (ANA) is a numerical algorithm that can be expressed solely in terms of vectors, vector spaces, and linear operators

Example Linear ANA (LANA) : Linear Conjugate Gradients

Given:

$A \in \mathcal{X} \to \mathcal{X}$: s.p.d. linear operator

$b \in \mathcal{X}$: right hand side vector

Find vector $x \in \mathcal{X}$ that solves $Ax = b$

- ANAs can be very mathematically sophisticated!
- ANAs can be extremely reusable!

Linear Conjugate Gradient Algorithm

Compute $r^{(0)} = b - Ax^{(0)}$ for the initial guess $x^{(0)}$.

for $i = 1, 2, \ldots$

$\rho_{i-1} = \left\langle r^{(i-1)}, r^{(i-1)} \right\rangle$

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ $(\beta_0 = 0)$

$p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)}$ $(p^{(1)} = r^{(1)})$

$q^{(i)} = Ap^{(i)}$

$\gamma_i = \left\langle p^{(i)}, q^{(i)} \right\rangle$

$\alpha_i = \rho_{i-1} / \gamma_i$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence; continue if necessary

end

Types of operations

- linear operator applications
- vector-vector operations
- scalar operations
- scalar product <x,y> defined by vector space

Types of objects

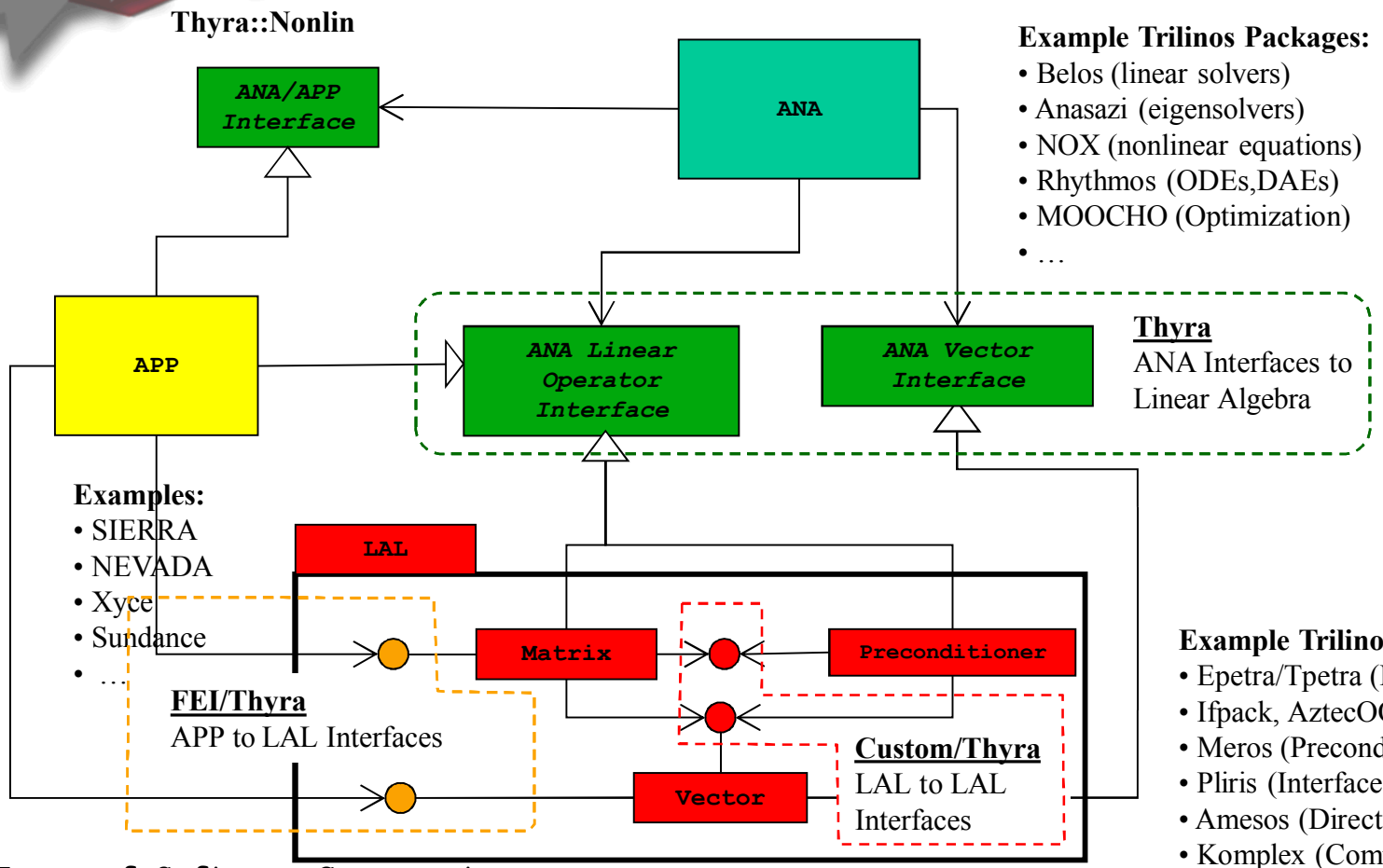
Linear Operators
- A

Vectors
- r, x, p, q

Scalars
- ρ, β, γ, α

Vector spaces?
- $\mathcal{X}$

Solver Software Components and Interfaces



Types of Software Components

1) ANA : Abstract Numerical Algorithm (e.g. linear solvers, eigensolvers, nonlinear solvers, stability analysis, uncertainty quantification, transient solvers, optimization etc.)

2) LAL : Linear Algebra Library (e.g. vectors, sparse matrices, sparse factorizations, preconditioners)

3) APP : Application (the model: physics, discretization method etc.)

Sandia National Laboratories

Stratimikos package

- Greek **στρατηγική** (strategy) + **γραμμικός** (linear)
- Uniform run-time interface to many different packages
  - Linear solvers: Amesos, AztecOO, Belos, …
  - Preconditioners: Ifpack, ML, …
- Defines common interface to create and use linear solvers
  - Thyra::DefaultLinearSolverBuilder
- Reads in options through a Teuchos::ParameterList
  - Can change solver and its options at run time
  - Can validate options, & read them from a string or XML file
- Accepts any linear system objects that provide
  - Epetra_Operator / Epetra_RowMatrix view of the matrix
  - Vector views (e.g., Epetra_MultiVector) for right-hand side and initial guess
- Increasing support for Tpetra objects

Stratimikos Parameter List and Sublists

```
<ParameterList name="Stratimikos">
  <Parameter name="Linear Solver Type" type="string" value="AztecOO"/>
  <Parameter name="Preconditioner Type" type="string" value="Ifpack"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="Amesos">
      <Parameter name="Solver Type" type="string" value="Klu"/>
      <ParameterList name="Amesos Settings">
        <Parameter name="MatrixProperty" type="string" value="general"/>
        ...
        <ParameterList name="Mumps"> ... </ParameterList>
        <ParameterList name="Superludist"> ... </ParameterList>
      </ParameterList>
    </ParameterList>
    <ParameterList name="AztecOO">
      <ParameterList name="Forward Solve">
        <Parameter name="Max Iterations" type="int" value="400"/>
        <Parameter name="Tolerance" type="double" value="1e-06"/>
        <ParameterList name="AztecOO Settings">
          <Parameter name="Aztec Solver" type="string" value="GMRES"/>
          ...
        </ParameterList>
      </ParameterList>
      ...
    </ParameterList>
    <ParameterList name="Belos"> ... </ParameterList>
  </ParameterList>
<ParameterList name="Preconditioner Types">
    <ParameterList name="Ifpack">
      <Parameter name="Prec Type" type="string" value="ILU"/>
      <Parameter name="Overlap" type="int" value="0"/>
      <ParameterList name="Ifpack Settings">
        <Parameter name="fact: level-of-fill" type="int" value="0"/>
        ...
      </ParameterList>
    </ParameterList>
    <ParameterList name="ML"> ... </ParameterList>
  </ParameterList>
</ParameterList>
```

Top level parameters

Linear Solvers

Preconditioners

Sublists passed on to package code!

Every parameter and sublist is handled by Thyra code and is fully validated!

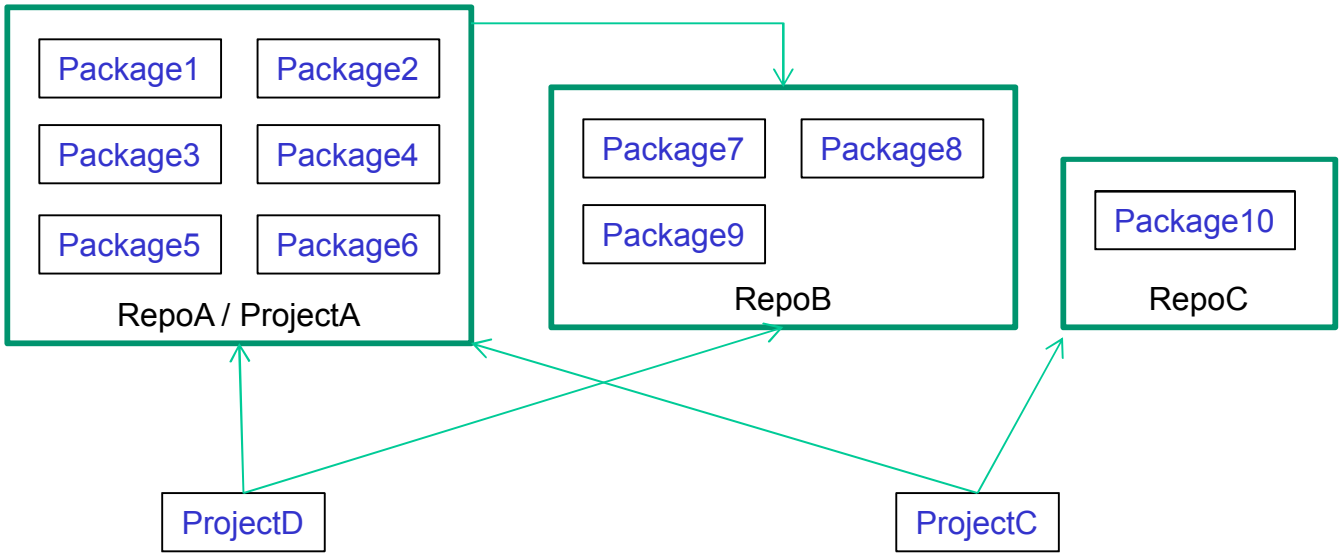
Sandia National Laboratories

Trilinos integrated into other libraries or applications

TriBITS: Trilinos/Tribal Build, Integrate, Test System

- Based on CMake, CTest, & CDash (Kitware open-source toolset)
  - Developed during Trilinos' move to CMake
  - Later extended for use in CASL projects (e.g., VERA) & SCALE
- Partitions a project into packages
  - Common CMake build and test infrastructure across packages
  - Handles dependencies between packages
- Integrated support for MPI, CUDA, & third-party libraries (TPLs)
- Multi-repository development
  - Can depend on packages in external repositories
  - Handy for mixing open-source & closed-source packages
- Test driver:
  - Partitions output per package to CDash
  - Failed packages don't propagate errors to downstream packages
  - Integrated coverage and memory testing (shows up on CDash)
  - Nightly and continuous integration (CI) test driver
- Pre-push synchronous continuous integration testing
  - Developers must use Python checkin-test.py script to push
  - It enables dependent packages, & builds & runs tests
  - Also automates asynchronous continuous integration tests
- Plus: TribitsDashboardDriver system, download-cmake.py and numerous other tools

TriBITS: Meta Project, Repository, Packages



Current state of TriBITS

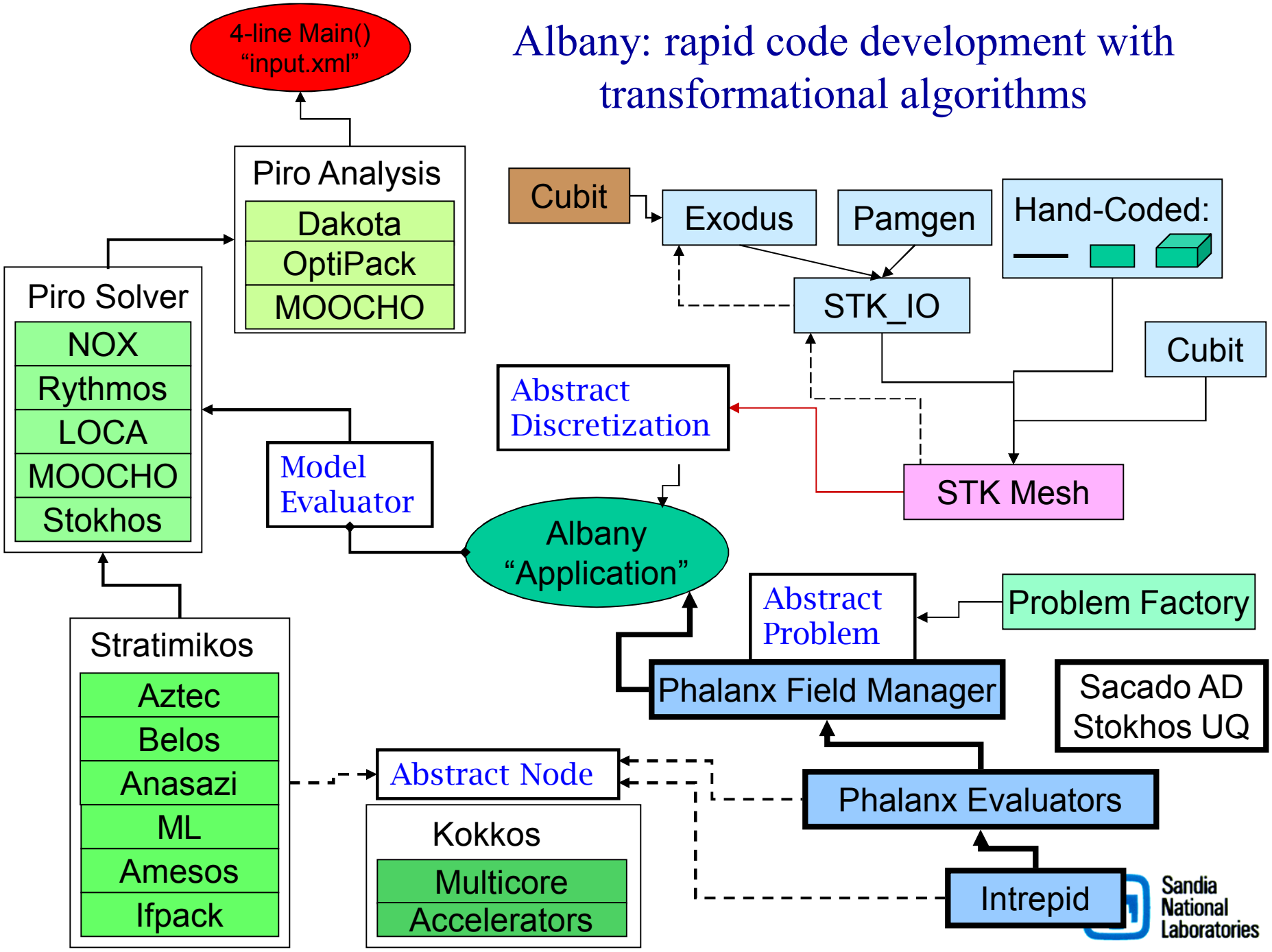
- Flexible aggregation of Packages from different Repositories into meta Projects
- TriBITS directory can be snapshoted out of Trilinos into stand-alone projects (independent of Trilinos)
- Being used by CASL VERA software, and several other CASL-related software packages
- egdist: Managing multiple repositories

Future changes/additions to TriBITS

- Combining concepts of TPLs and Packages to allow flexible configuration and building
- TribitsExampleProject
- Trilinos-independent TriBITS documentation
- Provide open access to TribitsExampleProject and therefore TriBITS

Sandia National Laboratories

Albany: rapid code development with transformational algorithms



Software interface idioms

Idioms: Common “look and feel”

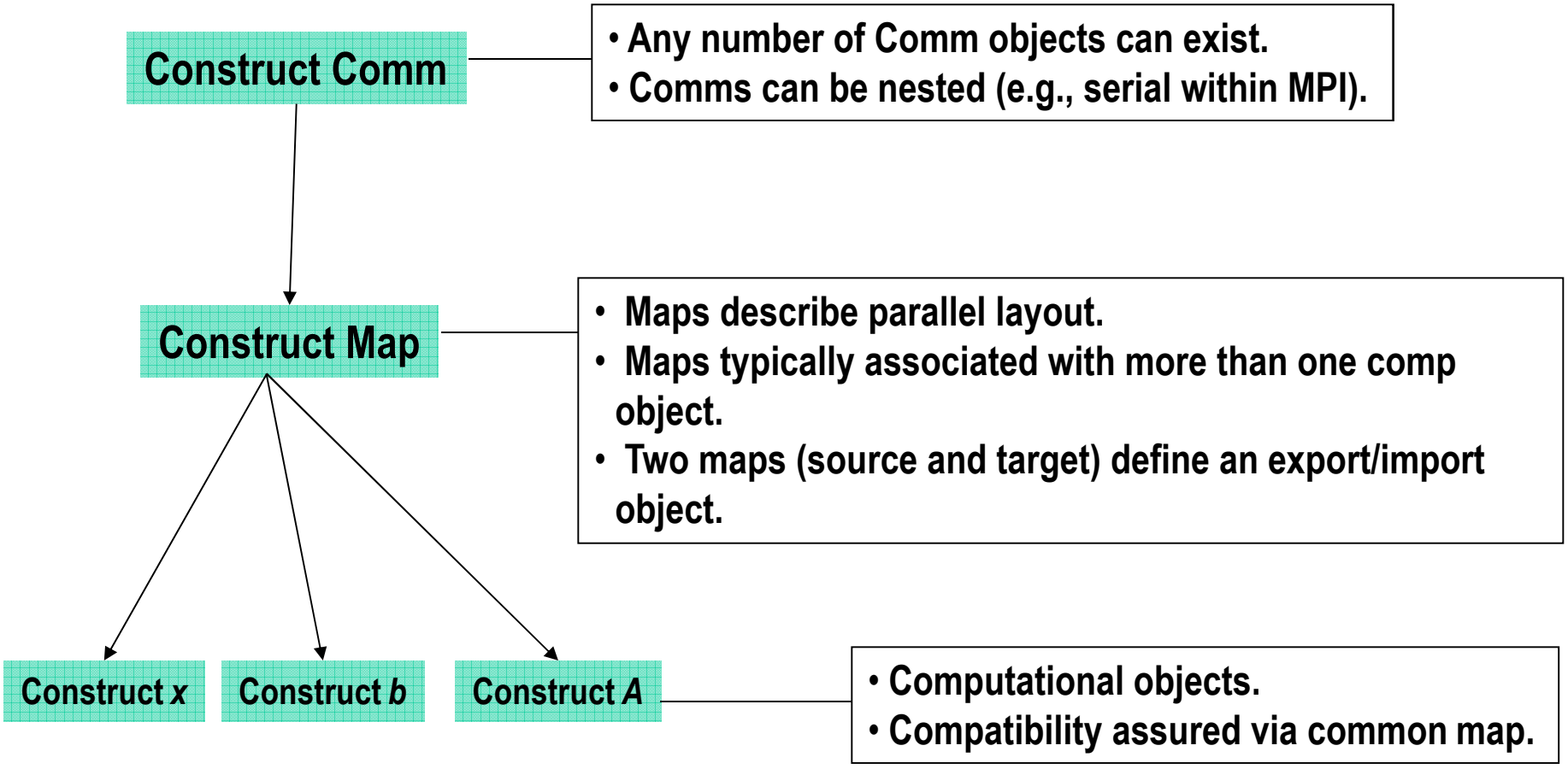
- Petra distributed object model
  - Provided by Epetra & Tpetra
  - Common “language” shared by many packages
- Kokkos shared-memory parallel programming model
  - Multidimensional arrays (with device-optimal layout)
  - Parallel operations (for, reduce, scan): user specifies kernel
  - Thread-parallel hash table, sparse graph, & sparse matrix
- Teuchos utilities package
  - Hierarchical “input deck” (ParameterList)
  - Memory management classes (RCP, ArrayRCP)
    - Safety: Manage data ownership & sharing
    - Performance: Avoid deep copies
  - Performance counters (e.g., TimeMonitor)

Petra Distributed Object Model

Sandia National Laboratories

Solving *Ax = b:*

Typical Petra Object Construction Sequence



Petra Implementations

- Epetra (Essential Petra):
  - Current production version
  - Uses stable core subset of C++ (circa 2000)
  - Restricted to real, double precision arithmetic
  - Interfaces accessible to C and Fortran users

- Tpetra (Templated Petra):
  - Next-generation version
  - C++ compiler can't be too ancient (no need for C++11 but good to have)
  - Supports arbitrary scalar and index types via templates
    - Arbitrary- and mixed-precision arithmetic
    - 64-bit indices for solving problems with >2 billion unknowns
  - Hybrid MPI / shared-memory parallel
    - Supports multicore CPU and hybrid CPU/GPU
    - Built on Kokkos manycore node library

Package leads: Mike Heroux, Mark Hoemmen (many developers)

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
```

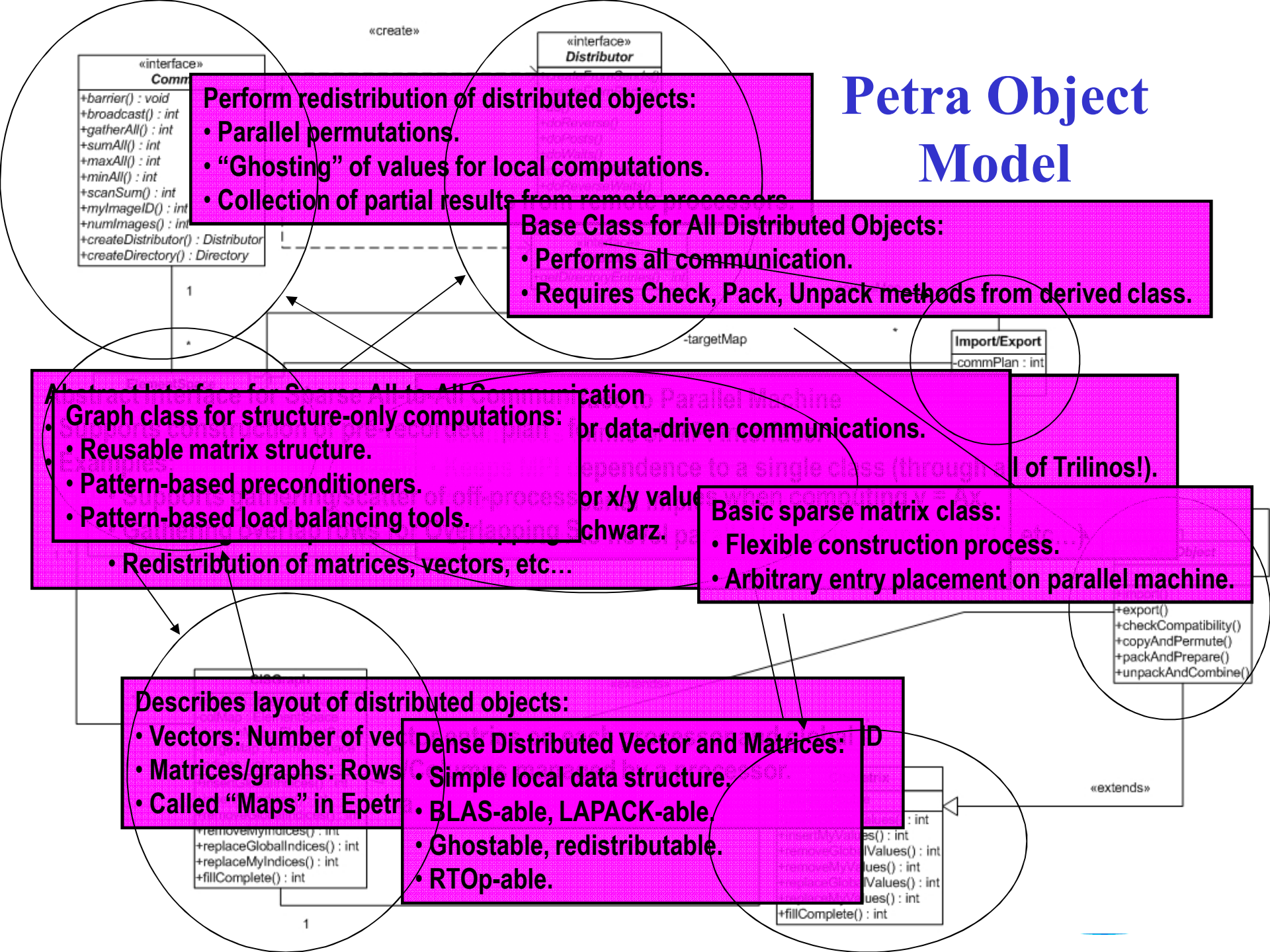
```
// ***** Create Linear Problem *****
  Epetra_LinearProblem problem(&A, &x, &b);
```

```
  // ***** 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;
}
```

Sandia National Laboratories

Petra Object Model



Details about Epetra Maps

- Note: Focus on Maps (not BlockMaps).
- Getting beyond standard use case…

- **Note: All of the concepts presented here for Epetra carry over to Tpetra!**

1-to-1 Maps

- A map is 1-to-1 if…
  - Each global ID appears only once in the map
  - (and is thus associated with only a single process)
- Certain operations in parallel data repartitioning require 1-to-1 maps:
  - Source map of an import must be 1-to-1.
  - Target map of an export must be 1-to-1.
  - Domain map of a 2D object must be 1-to-1.
  - Range map of a 2D object must be 1-to-1.

2D Objects: Four Maps

- Epetra 2D objects:
  - CrsMatrix, FECrsMatrix
  - CrsGraph
  - VbrMatrix, FEVbrMatrix
- Have four maps:
  - **Row Map**: On each processor, the global IDs of the **rows** that process will "manage."
  - **Column Map**: On each processor, the global IDs of the **columns** that process will "manage."
  - **Domain Map**: The layout of domain objects (the *x* (multi)vector in $y = Ax$).
  - **Range Map**: The layout of range objects (the *y* (multi)vector in $y = Ax$).

Typically a 1-to-1 map

Typically NOT a 1-to-1 map

Must be 1-to-1 maps!!!

Sandia National Laboratories

Sample Problem

$$\underset{\boldsymbol{y}}{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}} = \underset{\boldsymbol{A}}{\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}} \underset{\boldsymbol{x}}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}$$

Case 1: Standard Approach

- First 2 rows of A, elements of y and elements of x, kept on PE 0.
- Last row of A, element of y and element of x, kept on PE 1.

PE 0 Contents

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \ldots A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \end{bmatrix}, \ldots x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- RowMap = {0, 1}
- ColMap = {0, 1, 2}
- DomainMap = {0, 1}
- RangeMap = {0, 1}

PE 1 Contents

$$y = \begin{bmatrix} y_3 \end{bmatrix}, \ldots A = \begin{bmatrix} 0 & -1 & 2 \end{bmatrix}, \ldots x = \begin{bmatrix} x_3 \end{bmatrix}$$

- RowMap = {2}
- ColMap = {1, 2}
- DomainMap = {2}
- RangeMap = {2}

Original Problem

$$\begin{matrix} y \\ \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \end{matrix} = \begin{matrix} A \\ \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \end{matrix} \begin{matrix} x \\ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \end{matrix}$$

Notes:

- Rows are wholly owned.
- RowMap=DomainMap=RangeMap (all 1-to-1).
- ColMap is NOT 1-to-1.
- Call to FillComplete: A.FillComplete(); // Assumes

Sandia National Laboratories

Case 2: Twist 1

- First 2 rows of A, first element of y and last 2 elements of x, kept on PE 0.
- Last row of A, last 2 element of y and first element of x, kept on PE 1.

PE 0 Contents

$$y = \begin{bmatrix} y_1 \end{bmatrix}, \ldots A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \end{bmatrix}, \ldots x = \begin{bmatrix} x_2 \\ x_3 \end{bmatrix}$$

- RowMap = {0, 1}
- ColMap = {0, 1, 2}
- DomainMap = {1, 2}
- RangeMap = {0}

PE 1 Contents

$$y = \begin{bmatrix} y_2 \\ y_3 \end{bmatrix}, \ldots A = \begin{bmatrix} 0 & -1 & 2 \end{bmatrix}, \ldots x = \begin{bmatrix} x_1 \end{bmatrix}$$

- RowMap = {2}
- ColMap = {1, 2}
- DomainMap = {0}
- RangeMap = {1, 2}

Original Problem

$$\begin{matrix} y \\ \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \end{matrix} = \begin{matrix} A \\ \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \end{matrix} \begin{matrix} x \\ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \end{matrix}$$

Notes:

- Rows are wholly owned.
- RowMap is NOT = DomainMap
  is NOT = RangeMap (all 1-to-1).
- ColMap is NOT 1-to-1.
- Call to FillComplete:
  A.FillComplete(DomainMap, RangeMap);

Sandia National Laboratories

Case 2: Twist 2

- First row of A, part of second row of A, first element of y and last 2 elements of x, kept on PE 0.
- Last row, part of second row of A, last 2 element of y and first element of x, kept on PE 1.

PE 0 Contents

$$y = [y_1], \ldots A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 1 & 0 \end{bmatrix}, \ldots x = \begin{bmatrix} x_2 \\ x_3 \end{bmatrix}$$

- RowMap = {0, 1}
- ColMap = {0, 1}
- DomainMap = {1, 2}
- RangeMap = {0}

PE 1 Contents

$$y = \begin{bmatrix} y_2 \\ y_3 \end{bmatrix}, \ldots A = \begin{bmatrix} 0 & 1 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \ldots x = [x_1]$$

- RowMap = {1, 2}
- ColMap = {1, 2}
- DomainMap = {0}
- RangeMap = {1, 2}

Original Problem

$$\begin{matrix} y \\ \end{matrix} \quad \begin{matrix} A \\ \end{matrix} \quad \begin{matrix} x \\ \end{matrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Notes:

- Rows are NOT wholly owned.
- RowMap is NOT = DomainMap
  is NOT = RangeMap (all 1-to-1).
- RowMap and ColMap are NOT 1-to-1.
- Call to FillComplete:
  A.FillComplete(DomainMap, RangeMap);

Sandia National Laboratories

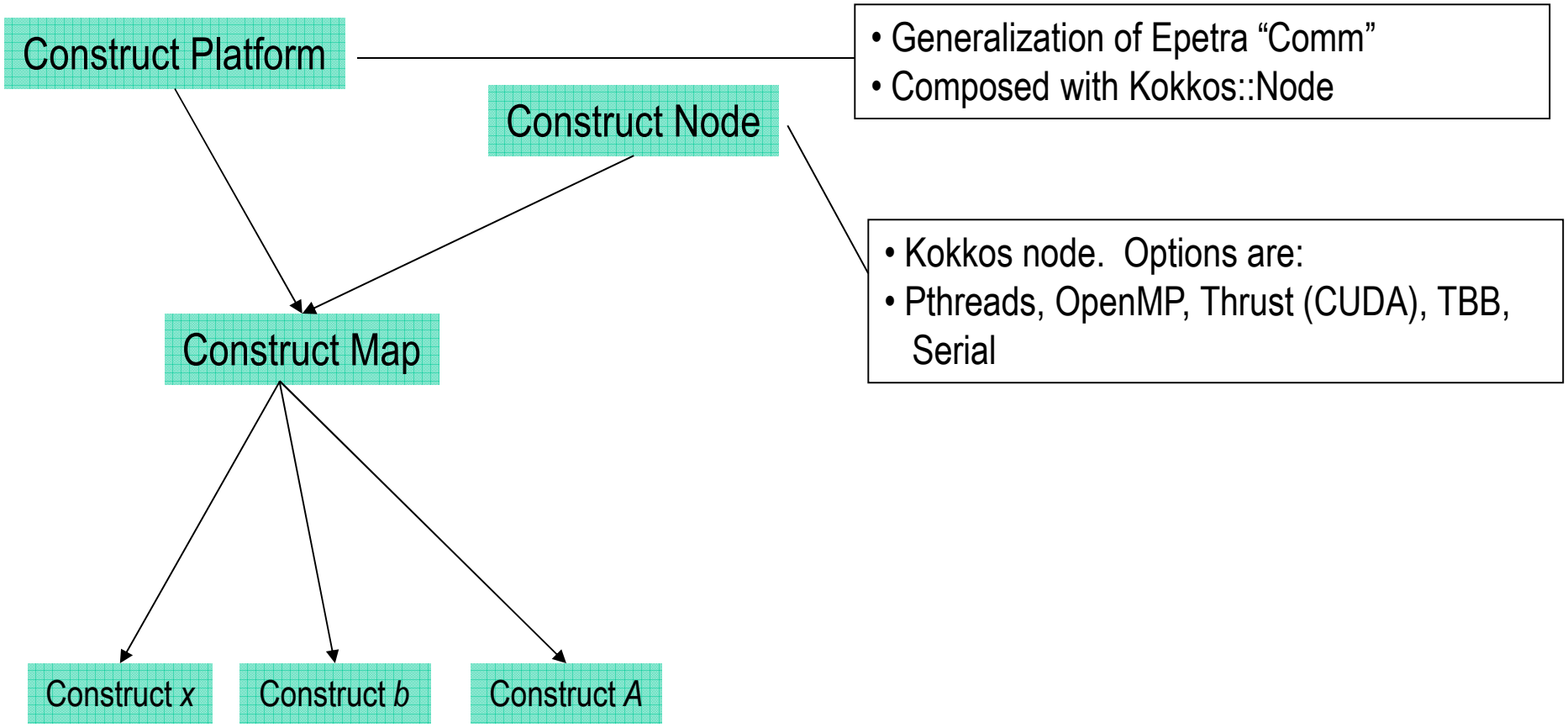
What does FillComplete do?

- Signals you're done defining matrix structure
- Does a bunch of stuff
- Creates communication patterns for distributed sparse matrix-vector multiply:
  - If ColMap ≠ DomainMap, create Import object
  - If RowMap ≠ RangeMap, create Export object
- A few rules:
  - Non-square matrices will *always* require:

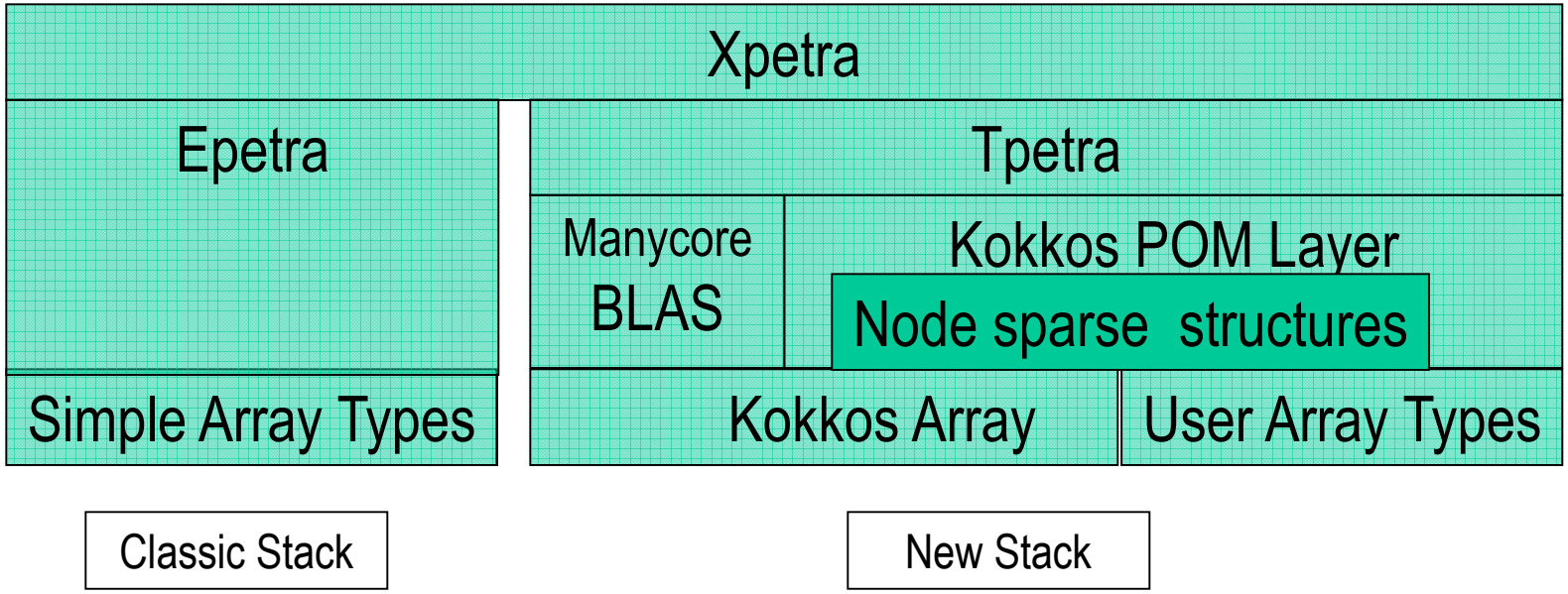
    ```
    A.FillComplete(DomainMap,RangeMap);
    ```

  - DomainMap and RangeMap *must be 1-to-1*

Typical Flow of Tpetra Object Construction



Data Classes Stacks



Tpetra-Xpetra Diff

```
< #include <Tpetra_Map.hpp>
< #include <Tpetra_CrsMatrix.hpp>
< #include <Tpetra_Vector.hpp>
< #include <Tpetra_MultiVector.hpp>
---
> #include <Xpetra_Map.hpp>
> #include <Xpetra_CrsMatrix.hpp>
> #include <Xpetra_Vector.hpp>
> #include <Xpetra_MultiVector.hpp>
>
> #include <Xpetra_MapFactory.hpp>
> #include <Xpetra_CrsMatrixFactory.hpp>
67c70,72
<   RCP<const Tpetra::Map<LO, GO> > map = Tpetra::createUniformContigMap<LO, GO>(numGlobalElements, comm);
---
>   Xpetra::UnderlyingLib lib = Xpetra::UseTpetra;
>
>   RCP<const Xpetra::Map<LO, GO> > map = Xpetra::MapFactory<LO, GO>::createUniformContigMap(lib, numGlobalEler
72c77
<   RCP<Tpetra::CrsMatrix<Scalar, LO, GO> > A = rcp(new Tpetra::CrsMatrix<Scalar, LO, GO>(map, 3));
---
>   RCP<Xpetra::CrsMatrix<Scalar, LO, GO> > A =  Xpetra::CrsMatrixFactory<Scalar, LO, GO>::Build(map, 3);
97d101
```

LO – Local Ordinal
GO – Global Ordinal

Sandia National Laboratories

ParameterList: Trilinos' "input deck"

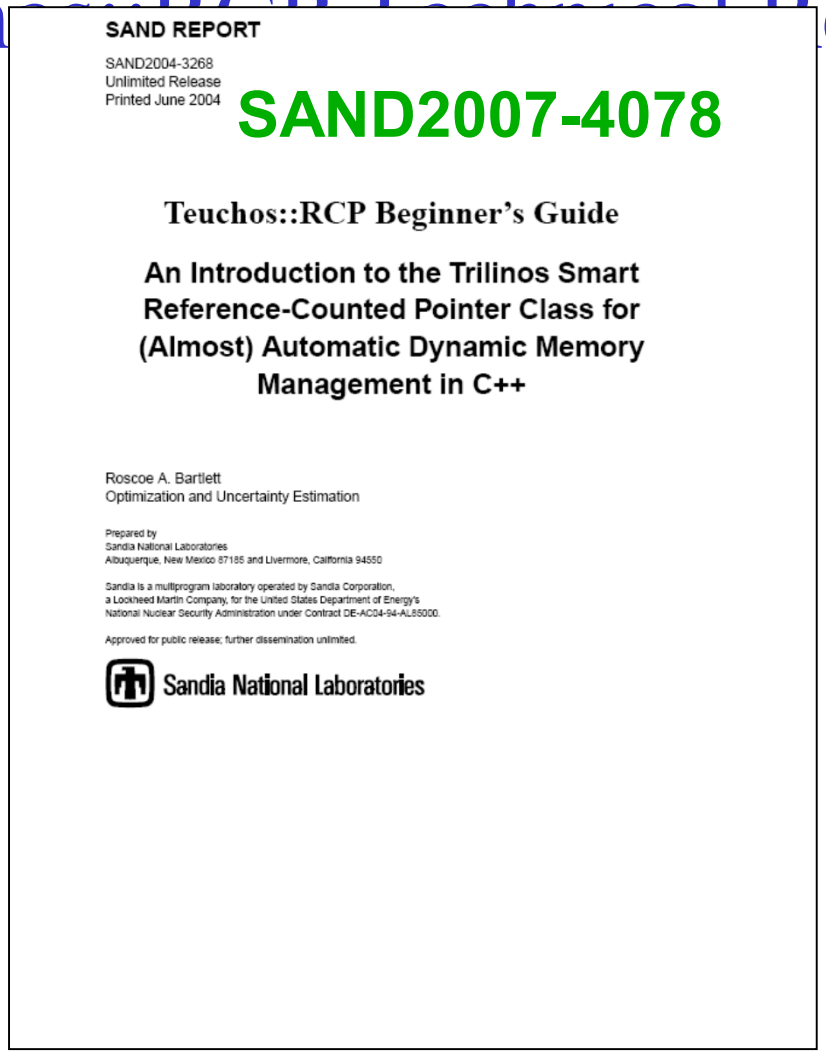
- Simple key/value pair database, but nest-able
  - Naturally hierarchical, just like numerical algorithms or software
  - Communication protocol between application layers
- Reproducible runs: save to XML, restore configuration
- Can express constraints and dependencies
- Optional GUI (Optika): lets novice users run your app

```
Teuchos::ParameterList p;
p.set("Solver", "GMRES");
p.set("Tolerance", 1.0e-4);
p.set("Max Iterations", 100);

Teuchos::ParameterList& lsParams = p.sublist("Solver Options");
lsParams.set("Fill Factor", 1);

double tol = p.get<double>("Tolerance");
int fill = p.sublist("Solver Options").get<int>("Fill Factor");
```

Memory management classes

- Scientific computation: Lots of data, big objects
  - Avoid copying and share data whenever possible
  - Who “owns” (deallocates) the data?
- Manual memory management (void*) not an option
  - Results in buggy and / or conservative code
- Reference-counted pointers (RCPs) and arrays
  - You don’t have to deallocate memory explicitly
  - Objects deallocated when nothing points to them anymore
  - Almost no performance cost for large objects

Teuchos::RCP Technical Report



SAND REPORT

SAND2004-3268
Unlimited Release
Printed June 2004

SAND2007-4078

Teuchos::RCP Beginner's Guide

An Introduction to the Trilinos Smart Reference-Counted Pointer Class for (Almost) Automatic Dynamic Memory Management in C++

Roscoe A. Bartlett
Optimization and Uncertainty Estimation

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

http://trilinos.sandia.gov/documentation.html

Trilinos/doc/RCPbeginnersGuide

Sandia National Laboratories

“But I don’t want RCPs!”

- They do add some keystrokes:
  - `RCP<Matrix>` vs. `Matrix*`
  - `ArrayRCP<double>` vs. `double[]`
- BUT: Run-time cost is none or very little
  - We have automated performance tests
- Debug build → useful error checking
  - More than Boost’s / C++11’s shared_ptr
  - Which we couldn’t use for historical reasons
- Not every Trilinos package exposes them
  - Some packages hide them behind handles or typedefs
  - Python “skin” hides them; Python is garbage-collected
- RCPs part of interface between packages
  - Trilinos like LEGO™ blocks
  - Packages don’t have to worry about memory management
    - Easier for them to share objects in interesting ways

TimeMonitor

- Timers that keep track of:
  - Runtime
  - Number of calls
- Time object associates a string name to the timer:

```
RCP<Time> stuffTimer =
  TimeMonitor::getNewCounter ("Do Stuff");
```

- TimeMonitor guard controls timer in scope-safe way

```
{
  TimeMonitor tm (*stuffTimer);
  doStuff ();
}
```

- Automatically takes care of recursive / nested calls
- Scalable (O(log P)), safe parallel timer statistics summary
  - `TimeMonitor::summarize ();`

Getting started: “How do I…?”

“How do I…?”

- Build my application with Trilinos?
- Learn about common Trilinos programming idioms?
- Download / find an installation of Trilinos?
- Find documentation and help?

Building your app with Trilinos

If you are using Makefiles:

- Makefile.export system

If you are using CMake:

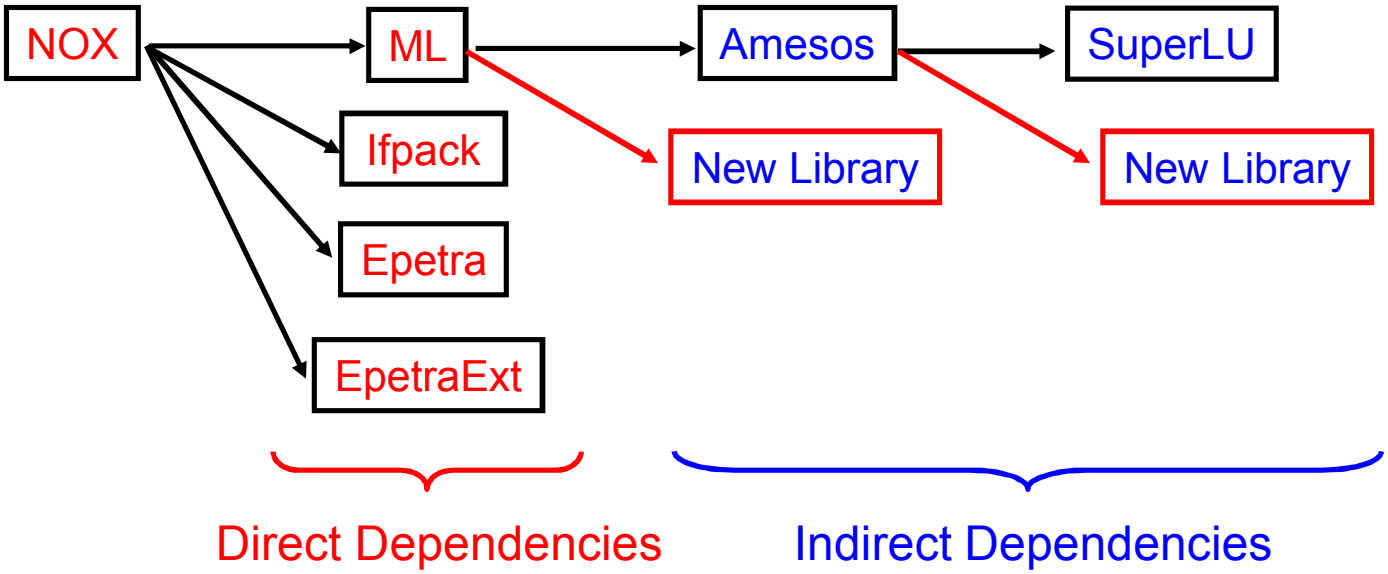
- CMake FIND_PACKAGE

Trilinos helps you link it with your application

- Library link order
  - -lnoxepetra -lnox –lepetra –lteuchos –lblas –llapack
  - Order matters!
  - Optional package dependencies affect required libraries
- Using the same compilers that Trilinos used
  - g++ or icc or icpc or …?
  - mpiCC or mpCC or mpicxx or … ?
- Using the same libraries that Trilinos used
  - Using Intel's MKL requires a web tool to get the link line right
  - Trilinos remembers this so you don't have to
- Consistent build options and package defines:
  - g++ -g –O3 –D HAVE_MPI –D _STL_CHECKED
- You don't have to figure any of this out! Trilinos does it for you!
  - Please don't try to guess and write a Makefile by hand!
  - This leads to trouble later on, which I've helped debug.

Why let Trilinos help?

- Trilinos has LOTS of packages
- As package dependencies (especially optional ones) are introduced, more maintenance is required by the top-level packages:



NOX either must:

- Account for the new libraries in its configure script (unscalable), or
- Let Trilinos' build system tell it about direct and indirect dependencies

Using CMake to build with Trilinos

- CMake: Cross-platform build system
  - Similar function as the GNU Autotools
- Building Trilinos requires CMake
- You don't have to use CMake to use Trilinos
- But if you do: FIND_PACKAGE(Trilinos …)
  - Example: https://code.google.com/p/trilinos/wiki/CMakeFindPackageTrilinosExample
- I find this much easier than writing Makefiles

Using the Makefile.export system

```
#
# A Makefile that your application can use if you want to build with Epetra.
#
# You must first set the TRILINOS_INSTALL_DIR variable.

# Include the Trilinos export Makefile for the Epetra package.
include $(TRILINOS_INSTALL_DIR)/include/Makefile.export.Epetra

# Add the Trilinos installation directory to the library and header search paths.
LIB_PATH = $(TRILINOS_INSTALL_DIR)/lib
INCLUDE_PATH = $(TRILINOS_INSTALL_DIR)/include $(CLIENT_EXTRA_INCLUDES)

# Set the C++ compiler and flags to those specified in the export Makefile.
# This ensures your application is built with the same compiler and flags
# with which Trilinos was built.
CXX = $(EPETRA_CXX_COMPILER)
CXXFLAGS = $(EPETRA_CXX_FLAGS)

# Add the Trilinos libraries, search path, and rpath to the
# linker command line arguments
LIBS = $(CLIENT_EXTRA_LIBS) $(SHARED_LIB_RPATH_COMMAND) \
  $(EPETRA_LIBRARIES) \
  $(EPETRA_TPL_LIBRARIES) \
  $(EPETRA_EXTRA_LD_FLAGS)

#
# Rules for building executables and objects.
#
%.exe : %.o $(EXTRA_OBJS)
        $(CXX) -o $@ $(LDFLAGS) $(CXXFLAGS) $< $(EXTRA_OBJS) -L$(LIB_PATH) $(LIBS)

%.o : %.cpp
        $(CXX) -c -o $@ $(CXXFLAGS) -I$(INCLUDE_PATH) $(EPETRA_TPL_INCLUDES) $<
```

How do I learn more?

How do I learn more?

- Documentation:
  - Trilinos Wiki with many runnable examples: https://code.google.com/p/trilinos/wiki/
  - Per-package documentation: http://trilinos.sandia.gov/packages/
  - Other material on Trilinos website: http://trilinos.sandia.gov/
- E-mail lists: http://trilinos.sandia.gov/mail_lists.html
- Annual user meetings and other tutorials:
  - Trilinos User Group (TUG) meeting and tutorial
    - Late October, or early November at SNL / NM
    - Talks available for download (slides and video):
      - http://trilinos.sandia.gov/events/trilinos_user_group_201<N>
      - Where N is 0, 1, 2, 3
  - European TUG meetings (once yearly, in summer)
    - Next: CSCS, Lugano, Switzerland, June 30 – July 1, 2014.
    - Also (tentative): Paris-Saclay, early March 2015.

How do I get Trilinos?

- Current release (11.6) available for download
  - http://trilinos.sandia.gov/download/trilinos-11.6.html
  - Source tarball with sample build scripts
  - 11.8 Freeze was last week.
- Public (read-only) git repository
  - http://trilinos.sandia.gov/publicRepo/index.html
  - Development version, updated ~ nightly
- Cray packages recent releases of Trilinos
  - http://www.nersc.gov/users/software/programming-libraries/math-libraries/trilinos/
  - `$ module load trilinos`
  - Recommended for best performance on Cray machines
- Most packages under BSD license
  - A few packages are LGPL

How do I build Trilinos?

- Need C and C++ compiler and the following tools:
  - CMake (version >= 2.8)
  - LAPACK and BLAS
- Optional software:
  - MPI (for distributed-memory parallel computation)
  - Many other third-party libraries
- You may need to write a short configure script
  - Sample configure scripts in sampleScripts/
  - Find one closest to your software setup, & tweak it
- Build sequence looks like GNU Autotools
  1. Invoke your configure script, that invokes CMake
  2. `make`
  3. `make install`
- Documentation:
  - http://trilinos.sandia.gov/Trilinos11CMakeQuickstart.txt
  - Ask me at the hands-on if interested

Hands-on tutorial

- Two ways to use Trilinos
  - Student shell accounts
  - WebTrilinos
- Student shell accounts
  - Pre-built Trilinos with Trilinos_tutorial (Github repository)
  - Github: branch, send pull requests, save commits / patches!
  - Steps (we may do some for you in advance):
    1) Log in to student account on paratools07.rrt.net
    2) git clone https://github.com/jwillenbring/Trilinos_tutorial.git
    3) cd Trilinos_tutorial && source ./setup.sh (load modules)
    4) cd cmake_build && ./live-cmake (build all examples)
    5) Change into build subdirectories to run examples by hand
- WebTrilinos
  - Build & run Trilinos examples in your web browser!
  - Need username & password (will give these out later)
  - https://code.google.com/p/trilinos/wiki/TrilinosHandsOnTutorial
  - Example codes: https://code.google.com/p/trilinos/w/list

Other options to use Trilinos

- Virtual machine
  - Install VirtualBox, download VM file, and run it
  - Same environment as student shell accounts
  - We won't cover this today, but feel free to try it
- Build Trilinos yourself on your computer
  - We may cover this later, depending on your interest
  - Prerequisites:
    - C++ compiler, Cmake version >= 2.8, BLAS & LAPACK, (MPI)
    - Download Trilinos: trilinos.org -> Download
  - Find a configuration script suitable for your computer
    - https://code.google.com/p/trilinos/wiki/BuildScript
    - Trilinos/sampleScripts/
  - Modify the script if necessary, & use it to run CMake
  - make –jN, make –jN install
  - Build your programs against Trilinos
    - Use CMake with FIND_PACKAGE(Trilinos …), or
    - Use Make with Trilinos Makefile.export system

Manycore/Acclerator Capabilities: A *very* brief introduction

Full tutorial:

http://trilinos.sandia.gov/events/trilinos_user_group_2013/presentations/2013-11-TUG-Kokkos-Tutorial.pdf

"MPI+X" programming model

- Modern HPC environment has at least 2 levels of parallelism:

MPI
- (1) distributed memory parallelism typically supported through a Message Passing Interface (MPI) library and

X
- (2) shared memory parallelism supported through one of the many thread-level programming models (CUDA, OpenMP, OpenACC, OpenCL, pthreads etc)

“X” in “MPI+X”

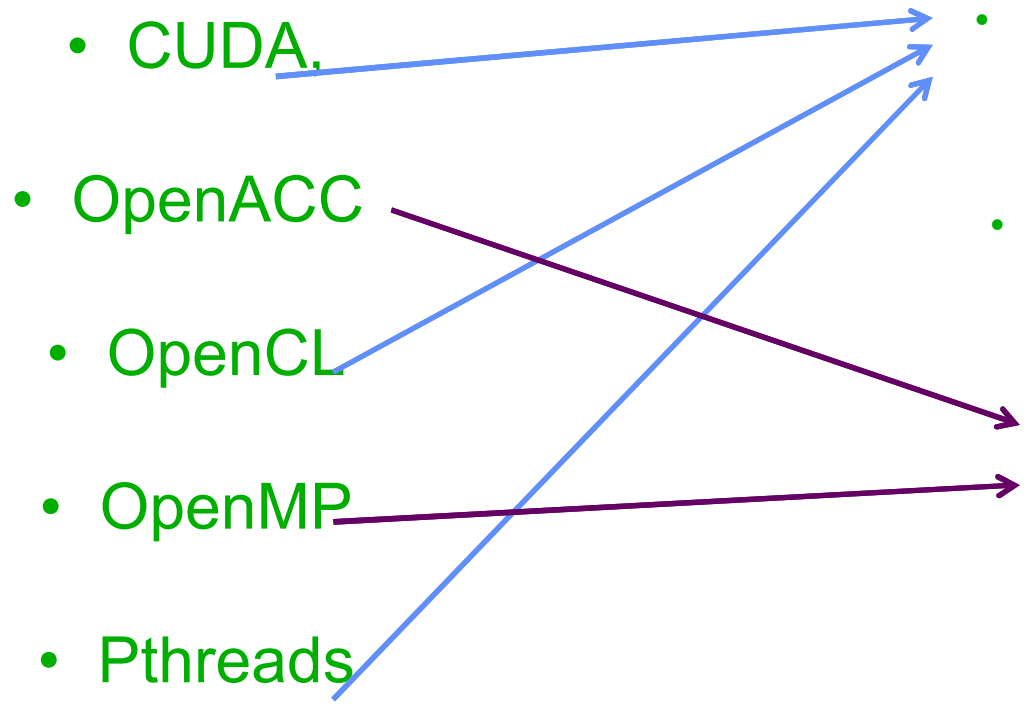
- CUDA,
- OpenACC
- ~~OpenCL~~ Incompatible with C++
- OpenMP
- Pthreads
- ….

NVIDIA GPUs

AMD GPUs, DSPs, and FPGAs

Multicore CPUs, MICs (Intel’s Xeon Phi)

"X" in "MPI+X"



- Low-level control and optimizations
  - Not easy to program
  - Require significant code modifications
- Pthreads: can't be used on GPUs

- Architecture-specific,
- OpenACC: doesn't provide low-level control or of e.g., memory residence

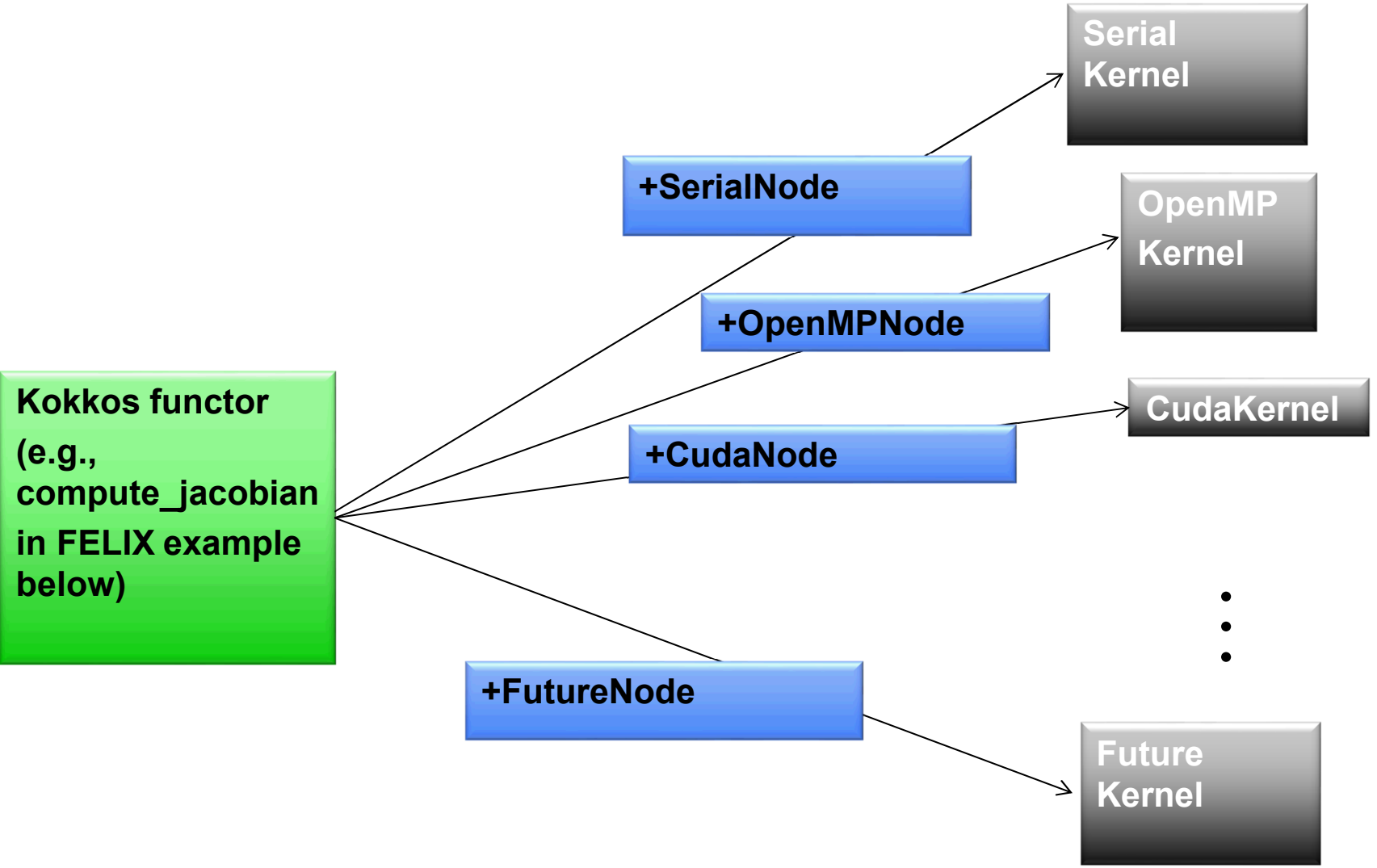
Kokkos - a programming model that enables performance portability across diverse and evolving manycore devices.

Struct-of-Arrays vs. Array-of-Structs

A False Dilemma

With C++ as your hammer,
everything looks like your thumb.

Compile-time Polymorphism



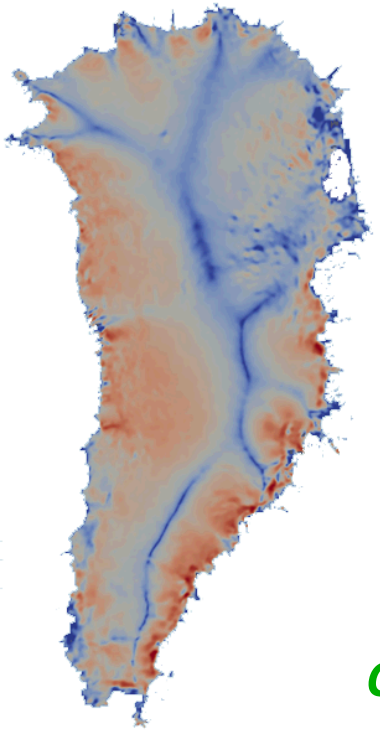
C++ Approach: Trilinos/Kokkos Array

- Challenge: Manycore Portability with Performance
  - Multicore-CPU and manycore-accelerator (e.g., NVIDIA)
  - Diverse memory access patterns, shared memory utilization, …
- Via a Library, not a language
  - C++ with template meta-programming
  - In the *spirit* of Thrust or Threading Building Blocks (TBB)
  - Concise and simple API: functions and multidimensional arrays
- Data Parallel Functions
  - Deferred task parallelism, pipeline parallelism, ...
  - Simple parallel_for and parallel_reduce semantics
- Multidimensional Arrays
  - versus “arrays of structs” or “structs of arrays”

Kokkos Array Abstractions

- Manycore Device
  - Has many threads of execution sharing a memory space
  - Manages a memory space separate from the host process
    - Physically separate (GPU) or logically separate (CPU)
    - or with non-uniform memory access (NUMA)
- Data Parallel Function
  - Created in the host process, executed on the manycore device
  - Performance can be dominated by memory access pattern
    - E.g., NVIDIA coalesced memory access pattern
- Multidimensional Array
  - ➢ Map array data into a manycore device's memory
  - Partition array data for data parallel work
  - Function + parallel partition + map -> memory access pattern

Albany Greenland Ice Sheet Model (FELIX project)



- An unstructured-grid finite element ice sheet code for land-ice modeling.

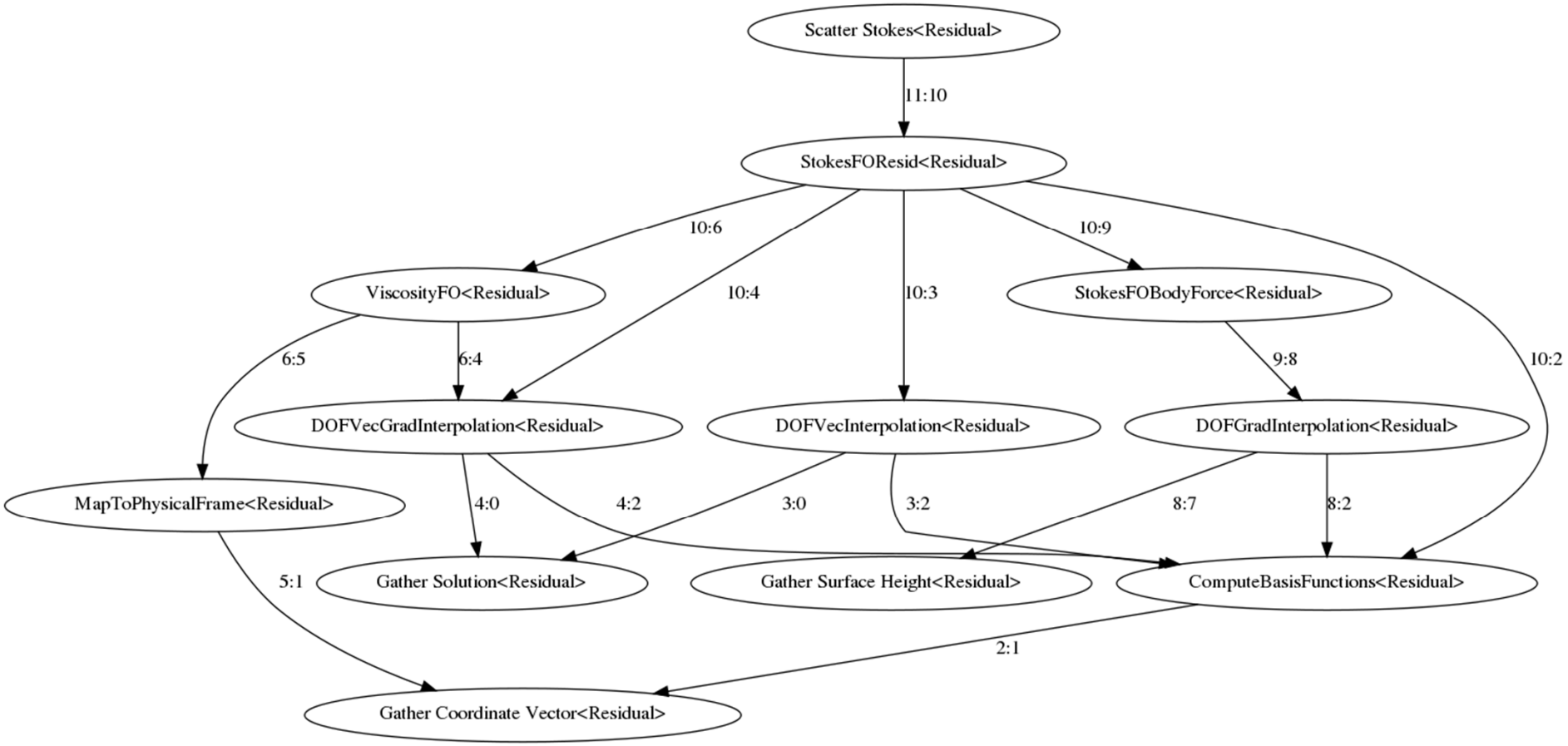
- Project objective:
  - Provide sea level rise prediction
  - Run on new architecture machines (hybrid systems).

Funding Source: SciDAC

Collaborators: SNL, ORNL, LANL, LBNL, UT, FSU, SC, MIT, NCAR

Sandia Staff: A. Salinger, I. Kalashnikova, M. Perego, R. Tuminaro, J. Jakeman, M. Eldred

Greenland Ice-Sheet model



Sandia National Laboratories

Kokkos implementation algorithm:

1) Replace array allocations with Kokkos::Views (in Host space)

2) Replace array access with Kokkos::Views

3) Replace functions with Functors, run in parallel on Host

4) Set device to ‘Cuda’, ‘OpenMP’ or ‘Threads’ and run on specified Device

General code structure for a Kokkos implementation

1. void main()
2. {
3. Device::initialize()
4. Allocate Kokkos::View (arrays)
5. …
6. Kokkos::deep_copy (..) //Copy data to device
7. …
8. Kokkos::parallel_for/ parallel_reduce (over the number of iterations to be performed in parallel, Kokkos_Functor)
9. …
10. Kokkos::deep_copy (..) //Copy data to host
11. …
12. Device::finalize()
13. }

Kokkos_functor example: compute_jacobian

```
for(int cell = 0; cell < worksetNumCells; cell++) {
    for(int qp = 0; qp < numQPs; qp++) {
      for(int row = 0; row < numDims; row++){
        for(int col = 0; col < numDims; col++){
          for(int node = 0; node < numNodes; node++){
            jacobian(cell, qp, row, col) +=

                coordVec(cell, node, row)
                        *basisGrads(node, qp, col);
          } // node
        } // col
      } // row
    } // qp
  } // cell
```

```
Kokkos::parallel_for ( worksetNumCells,
 compute_jacobian<ScalarT, Device, numQPs, numDims,
numNodes> (basisGrads, jacobian, coordVec));
```

```
template < typename ScalarType, clas DeviceType, int numQPs_,
                                    int numDims_, int numNodes_ >
class compute_jacobian  {
  Array3 basisGrads_;
  Array4 jacobian_;
  Array3_const coordVec_;
public:
  typedef DeviceType device_type;
  compute_jacobian(Array3 &basisGrads, Array4 &jacobian,
          Array3 &coordVec)
   : basisGrads_(basisGrads)
   , jacobian_(jacobian)
   , coordVec_(coordVec){}

KOKKOS_INLINE_FUNCTION
 void operator () (const std::size_t i) const
 {
  for(int qp = 0; qp < numQPs_; qp++) {
       for(int row = 0; row < numDims_; row++){
         for(int col = 0; col < numDims_; col++){
           for(int node = 0; node < numNodes_; node++){
             jacobian_(i, qp, row, col) += coordVec_(i, node, row)
                                          *basisGrads_(node, qp, col);
           } // node
         } // col
       } // row
    } // qp
 }
 };
```

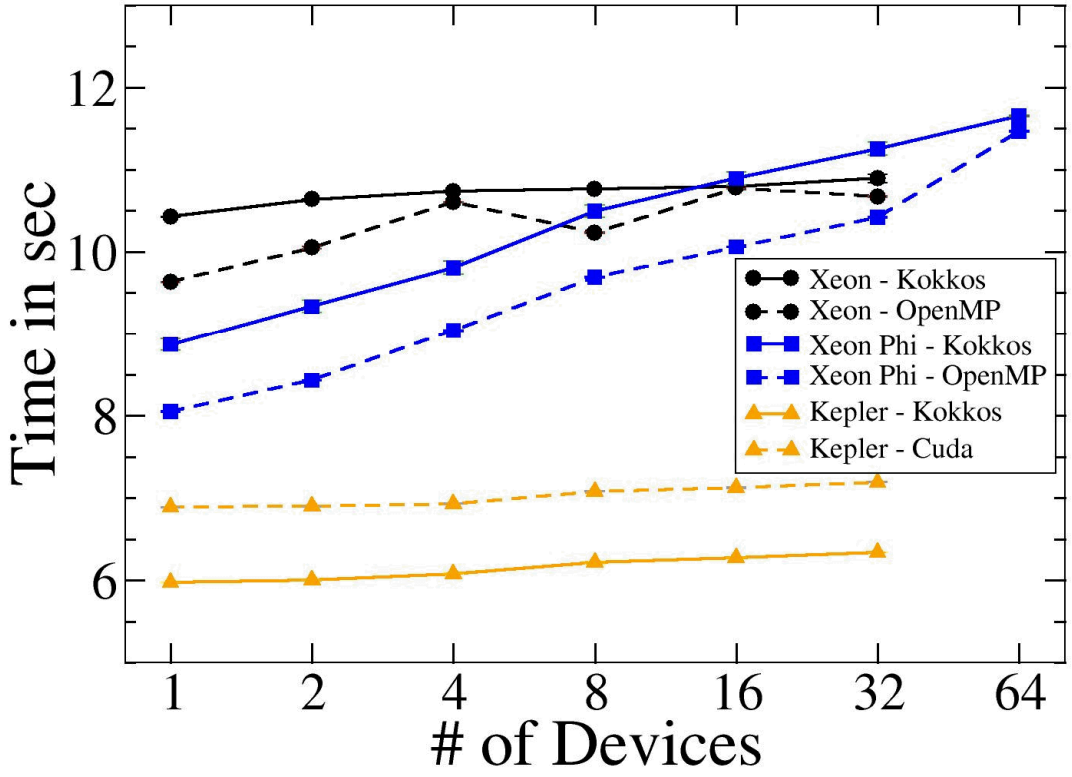
Performance Evaluation

- Using Sandia Computing Research Center Testbed Clusters
  - Compton: 32nodes
    - 2x Intel Xeon E5-2670 (Sandy Bridge), hyperthreading enabled
    - 2x Intel Xeon Phi 57core (pre-production)
    - ICC 13.1.2, Intel MPI 4.1.1.036
  - Shannon: 32nodes
    - 2x Intel Xeon E5-2670, hyperthreading disabled
    - 2x NVidia K20x
    - GCC 4.4.5, Cuda 5.5, MVAPICH2 v1.9 with GPU-Direct
- Absolute performance “unit” tests
  - Evaluate parallel dispatch/synchronization efficiency
  - Evaluate impact of array access patterns and capabilities
- Mini-application : Kokkos vs. ‘native’ implementations
  - Evaluate cost of portability

MPI+X Performance Test: MiniFE

Conjugate Gradient Solve of a Finite Element Matrix

- Comparing X = Kokkos, OpenMP, Cuda (GPU-direct via MVAPICH2)
- Weak scaling with one MPI process per device
  - Except on Xeon: OpenMP requires one process/socket due to NUMA
  - 8M elements/device
- Kokkos performance
  - 90% or better of "native"
  - Improvements ongoing



Linear System Solves

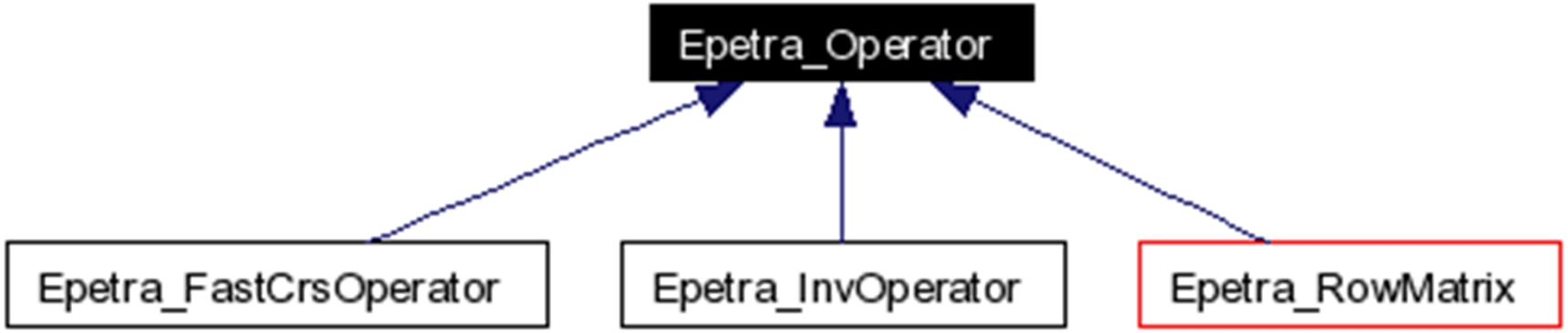
AztecOO

- Aztec is the previous workhorse solver at Sandia:
  - Extracted from the MPSalsa reacting flow code.
  - Installed in dozens of Sandia apps.
- AztecOO leverages the investment in Aztec:
  - Uses Aztec iterative methods and preconditioners.
- AztecOO improves on Aztec by:
  - Using Epetra objects for defining matrix and RHS.
  - Providing more preconditioners/scalings.
  - Using C++ class design to enable more sophisticated use.
- AztecOO interfaces allows:
  - Continued use of Aztec for functionality.
  - Introduction of new solver capabilities outside of Aztec.
- Belos is coming along as alternative.
  - AztecOO will not go away.
  - Will encourage new efforts and refactorings to use Belos.

AztecOO Extensibility

- AztecOO is designed to accept externally defined:
  - **Operators** (both *A* and *M*):
    - The linear operator *A* is accessed as an Epetra_Operator.
    - Users can register a preconstructed preconditioner as an Epetra_Operator.
  - **RowMatrix**:
    - If *A* is registered as a RowMatrix, Aztec's preconditioners are accessible.
    - Alternatively *M* can be registered separately as an Epetra_RowMatrix, and Aztec's preconditioners are accessible.
  - **StatusTests**:
    - Aztec's standard stopping criteria are accessible.
    - Can override these mechanisms by registering a StatusTest Object.

AztecOO understands Epetra_Operator



- AztecOO is designed to accept externally defined:
  - Operators (both *A* and *M*).
  - RowMatrix (Facilitates use of AztecOO preconditioners with external *A*).
  - StatusTests (externally-defined stopping criteria).

Belos and Anasazi

- Next generation linear solver / eigensolver library, written in templated C++.
- Provide a generic interface to a collection of algorithms for solving large-scale linear problems / eigenproblems.
- Algorithm implementation is accomplished through the use of traits classes and abstract base classes:
  - e.g.: MultiVecTraits, OperatorTraits
  - e.g.: SolverManager, Eigensolver / Iteration, Eigenproblem/ LinearProblem, StatusTest, OrthoManager, OutputManager
- Includes block linear solvers / eigensolvers:
  - Higher operator performance.
  - More reliable.
- Solves:
  - $AX = X\Lambda$ or $AX = BX\Lambda$ (Anasazi)
  - $AX = B$ (Belos)

Why are Block Solvers Useful?

- Block Solvers (in general):
  - Achieve better performance for operator-vector products.
- Block Eigensolvers (Op(A)X = LX):
  - Reliably determine multiple and/or clustered eigenvalues.
  - Example applications: Modal analysis, stability analysis, bifurcation analysis (LOCA)
- Block Linear Solvers (Op(A)X = B):
  - Useful for when multiple solutions are required for the same system of equations.
  - Example applications:
    - Perturbation analysis
    - Optimization problems
    - Single right-hand sides where A has a handful of small eigenvalues
    - Inner-iteration of block eigensolvers

Linear / Eigensolver Software Design

Belos and Anasazi are solver libraries that:

1. Provide an abstract interface to an operator-vector products, scaling, and preconditioning.
2. Allow the user to enlist any linear algebra package for the elementary vector space operations essential to the algorithm. (Epetra, PETSc, etc.)
3. Allow the user to define convergence of any algorithm (a.k.a. status testing).
4. Allow the user to determine the verbosity level, formatting, and processor for the output.
5. Allow these decisions to be made at runtime.
6. Allow for easier creation of new solvers through “managers” using “iterations” as the basic kernels.

Nonlinear System Solves

NOX/LOCA: Nonlinear Solver and Analysis Algorithms

NOX and LOCA are a combined package for solving and analyzing sets of nonlinear equations.

- NOX: Globalized Newton-based solvers.
- LOCA: Continuation, Stability, and Bifurcation Analysis.

We define the nonlinear problem:

$$\text{given } F : \mathbb{R}^n \rightarrow \mathbb{R}^n,$$

$$\text{find } x_* \in \mathbb{R}^n \text{ such that } F(x_*) = 0 \in \mathbb{R}^n$$

F is the residual or function evaluation

x is the solution vector

$J \in \mathbb{R}^{n \times n}$ is the Jacobian Matrix defined by: $J_{ij} = \dfrac{\partial F_i}{\partial x_j}$

Nonlinear Solver Algorithms

Tensor Method

$$M_T = f(x_c) + J_c d + \frac{1}{2} T_c dd$$

Newton's Method

$$M_N = f(x_c) + J_c d$$

Broyden's Method

$$M_B = f(x_c) + B_c d$$

Globalizations

Line Search
Interval Halving
Quadratic
Cubic
More'-Thuente
Curvilinear (Tensor)

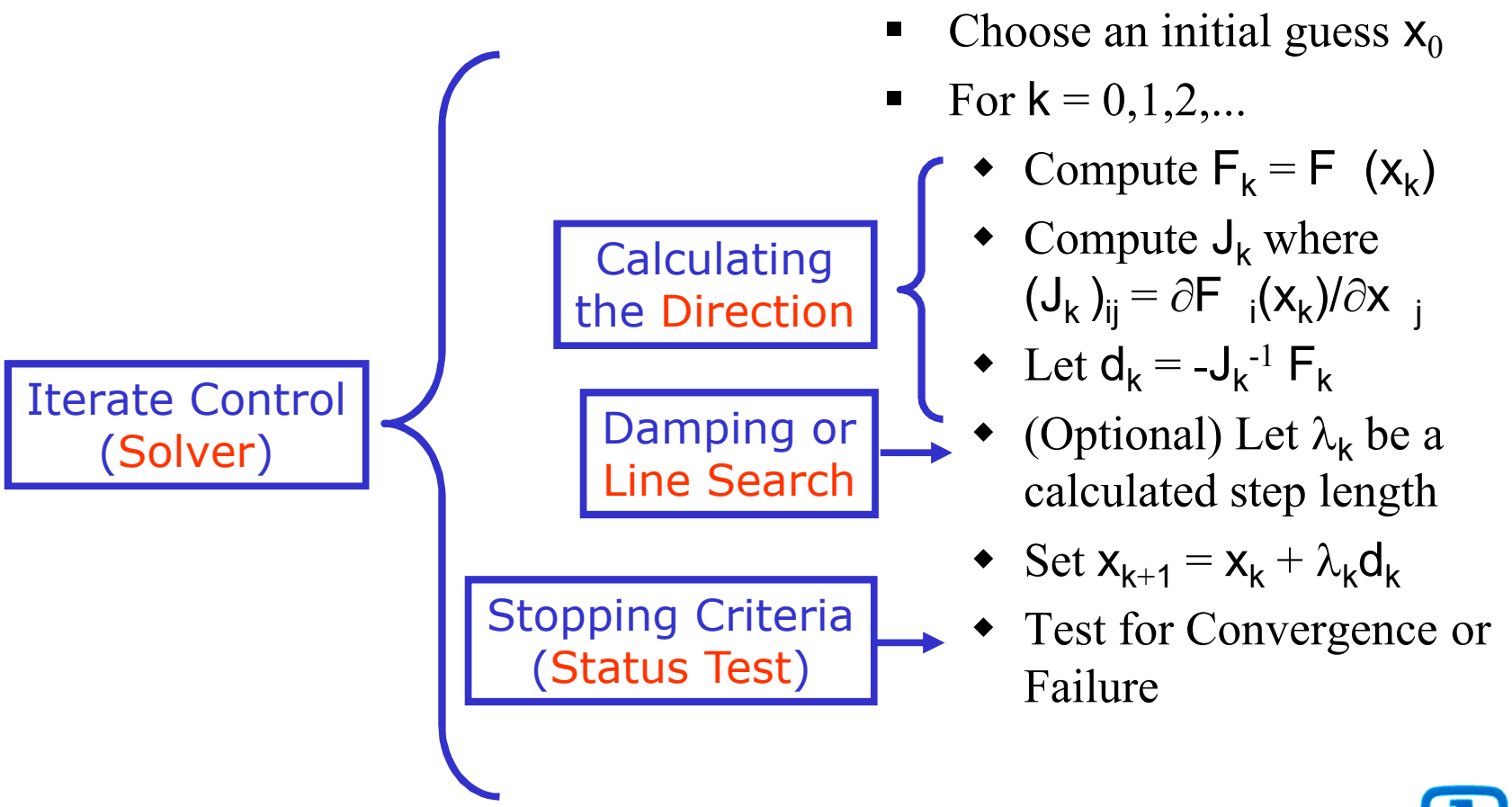
Trust Region
Dogleg
Inexact Dogleg

Homotopy
Artificial Parameter Continuation
Natural Parameter Continuation

Iterative Linear Solvers: Adaptive Forcing Terms
Jacobian-Free Newton-Krylov
Jacobian Estimation: Colored Finite Difference

Building Blocks of NOX

Example: Newton's Method for $F(x) = 0$



Sandia National Laboratories

Stopping Criteria
(StatusTests)

Highly Flexible Design: Users build a convergence test hierarchy and registers it with the solver (via solver constructor or reset method).

- Norm F: {Inf, One, Two} {absolute, relative} $\|F\| \leq \text{tol}$
- Norm Update ΔX: {Inf, One, Two} $\|x_k - x_{k-1}\| \leq \text{tol}$
- Norm Weighted Root Mean Square (WRMS):

$$C\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i^k - x_i^{k-1}}{\text{RTOL}|x_i^{k-1}| + \text{ATOL}_i}\right)^2} \leq \text{tol}$$

- Max Iterations: Failure test if solver reaches max # iters
- FiniteValue: Failure test that checks for NaN and Inf on $\|F\|$
- Stagnation: Failure test that triggers if the convergence rate fails a tolerance check for n consecutive iterations.

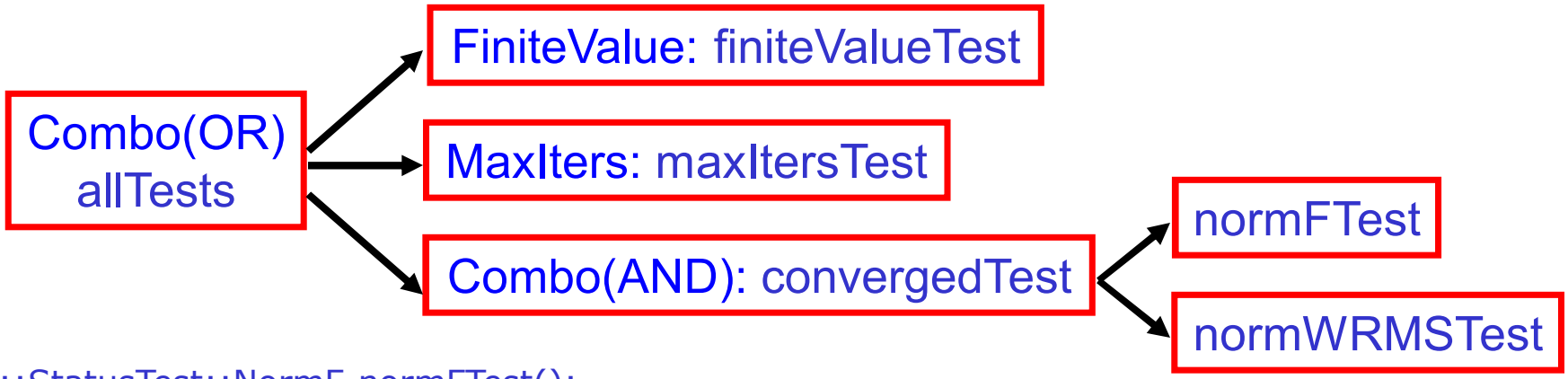
$$\frac{\|F_k\|}{\|F_{k-1}\|} \geq \text{tol}$$

- Combination: {AND, OR}
- Users Designed: Derive from NOX::StatusTest::Generic

Sandia National Laboratories

Building a Status Test

- Converge if both: $\|F\| \leq 1.0E-6 \quad \|\delta x\|_{WRMS} \leq 1.0$
- Fail if value of $\|F\|$ becomes Nan or Inf
- Fail if we reach maximum iterations




```
NOX::StatusTest::NormF normFTest();
NOX::StatusTest::NormWRMS normWRMSTest();
NOX::StatusTest::Combo convergedTest(NOX::StatusTest::Combo::AND);
convergedTest.addStatusTest(normFTest);
convergedTest.addStatusTest(normWRMSTest);
NOX::StatusTest::FiniteValue finiteValueTest;
NOX::StatusTest::MaxIters maxItersTest(200);
NOX::StatusTest::Combo allTests(NOX::StatusTest::Combo::OR);
allTests.addStatusTest(finiteValueTest);
allTests.addStatusTest(maxItersTest);
allTests.addStatusTest(convergedTest);
```

Sandia National Laboratories

Status Tests Continued

```
-- Final Status Test Results --
Converged....OR Combination ->
  Converged....AND Combination ->
    Converged....F-Norm = 3.567e-13 < 1.000e-08
                  (Length-Scaled Two-Norm, Absolute Tolerance)
    Converged....WRMS-Norm = 1.724e-03 < 1
                  (Min Step Size:  1.000e+00 >= 1)
                  (Max Lin Solv Tol:  4.951e-14 < 0.5)
  ??..........Finite Number Check (Two-Norm F) = Unknown
  ??..........Number of Iterations = -1 < 200
```

User Defined are Derived from NOX::StatusTest::Generic

```
NOX::StatusTest::StatusType checkStatus(const NOX::Solver::Generic &problem)

NOX::StatusTest::StatusType
checkStatusEfficiently(const NOX::Solver::Generic &problem,
                       NOX::StatusTest::CheckType checkType)

NOX::StatusTest::StatusType getStatus() const

ostream& print(ostream &stream, int indent=0) const
```

NOX Interface

NOX solver methods are ANAs, and are implemented in terms of group/vector abstract interfaces:

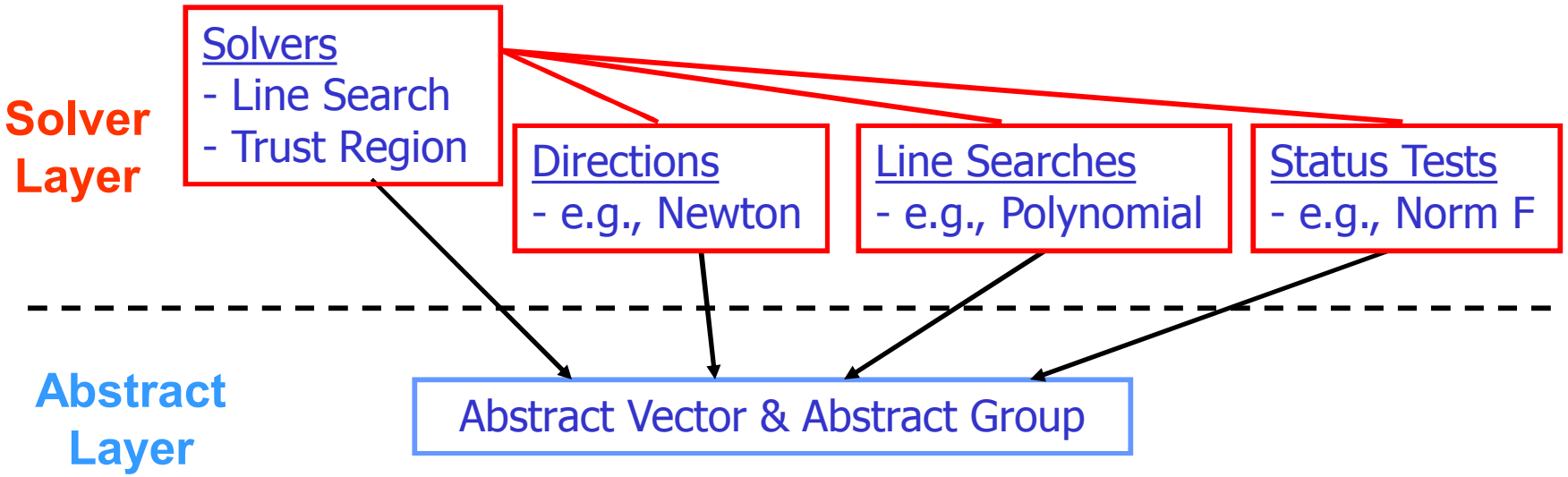
Group	Vector
`computeF()`	`innerProduct()`
`computeJacobian()`	`scale()`
`applyJacobianInverse()`	`norm()`
	`update()`

NOX solvers will work with any group/vector that implements these interfaces.

Four concrete implementations are supported:

1. LAPACK
2. EPETRA
3. PETSc
4. Thyra (Release 8.0)

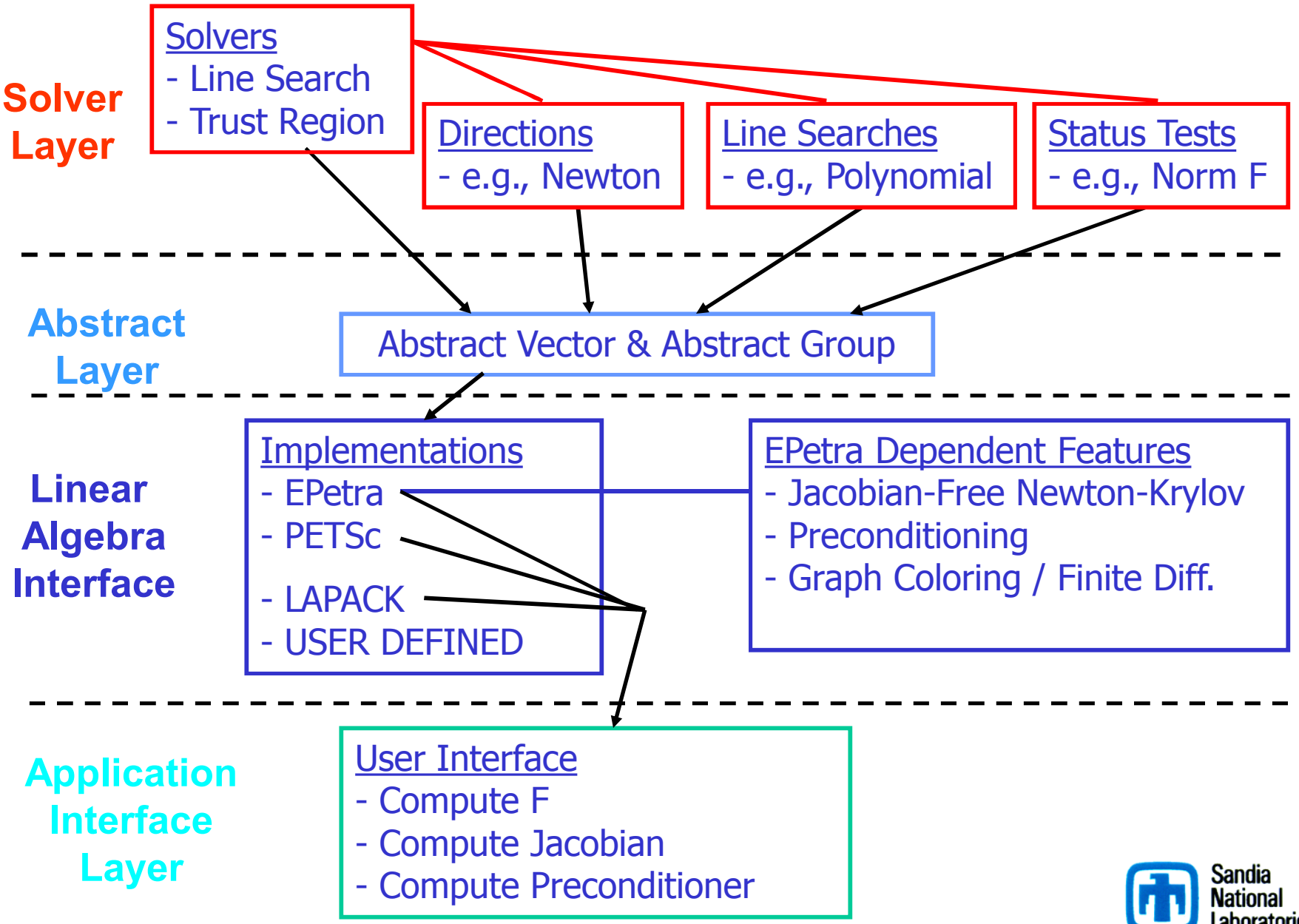
NOX Interface



- Don't need to directly access the vector or matrix entries, only manipulate the objects.
- NOX uses an abstract interface to manipulate linear algebra objects.
- Isolate the Solver layer from the linear algebra implementations used by the application.
- This approach means that NOX does NOT rely on any specific linear algebra format.
- Allows the apps to tailor the linear algebra to their own needs!
  - Serial or Parallel
  - Any Storage format: User Defined, LAPACK, PETSc, Epetra

Sandia National Laboratories

NOX Framework



Sandia National Laboratories

The Epetra "Goodies"

- Matrix-Free Newton-Krylov Operator
  - Derived from Epetra_Operator
  - Can be used to estimate Jacobian action on a vector
  - NOX::Epetra::MatrixFree

$$Jy = \frac{F(x + y\delta) - F(x)}{\delta}$$

- Finite Difference Jacobian
  - Derived from an Epetra_RowMatrix
  - Can be used as a preconditioner matrix
  - NOX::Epetra::FiniteDifference

$$J_j = \frac{F(x + \delta e_j) - F(x)}{\delta}$$

- Graph Colored Finite Difference Jacobian
  - Derived from NOX::Epetra::FiniteDifference
  - Fast Jacobian fills – need connectivity/coloring graph
  - (NOX::Epetra::FiniteDifferenceColoring)

- Full interface to AztecOO using NOX parameter list
- Preconditioners: internal AztecOO, Ifpack, User defined
- Scaling object

Sandia National Laboratories

Next Generation Multigrid: MueLu

Andrey Prokopenko, Jonathan Hu, Chris Siefert, Ray Tuminaro, Tobias Wiesner

Sandia National Laboratories

Motivation for a New Multigrid Library

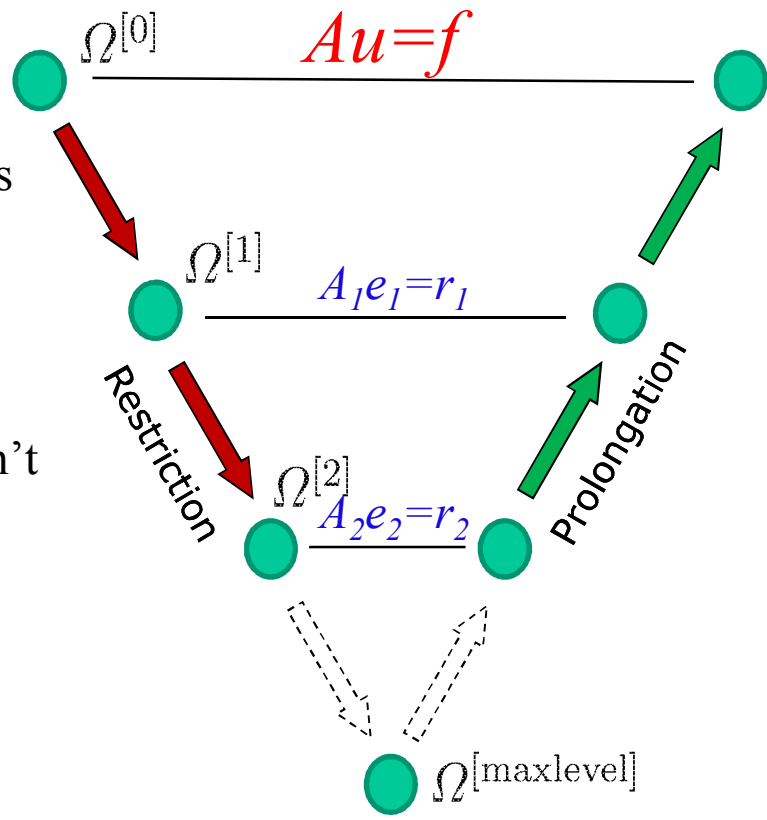
- Trilinos already has mature multigrid library, ML
  - Algorithms for Poisson, Elasticity, H(curl), H(div)
  - Algorithms extensively exercised in practice
  - Broad user base with hard problems
- However …
  - Poor links to other Trilinos capabilities (e.g., smoothers)
  - C-based, only scalar type "double" supported explicitly
  - Over 50K lines of source code
    - Hard to add cross-cutting features like MPI+X
    - Optimizations & semantics are poorly documented

Objectives for New Multigrid Framework

- Templating on scalar, ordinal types
  - Scalar: Complex; extended precision
  - Ordinal: Support 64-bit global indices for huge problems
- Advanced architectures
  - Kokkos support for various compute node types (MPI, MPI+threads, MPI+GPU)
- Extensibility
  - Facilitate development of other algorithms
    - Energy minimization methods
    - Geometric, classic algebraic multigrid, …
  - Ability to combine several types of multigrid
- Preconditioner reuse
  - Reduce setup expense

AMG

- Two main components
  - Smoothers
    - Approximate solves on each level
    - "Cheaply" reduces particular error components
    - On coarsest level, smoother = A_i^{-1} (usually)
  - Grid Transfers
    - Moves data between levels
    - Must represent components that smoothers can't reduce
- Algebraic Multigrid (AMG)
  - AMG generates grid transfers
  - AMG generates coarse grid A_i's



Sandia National Laboratories

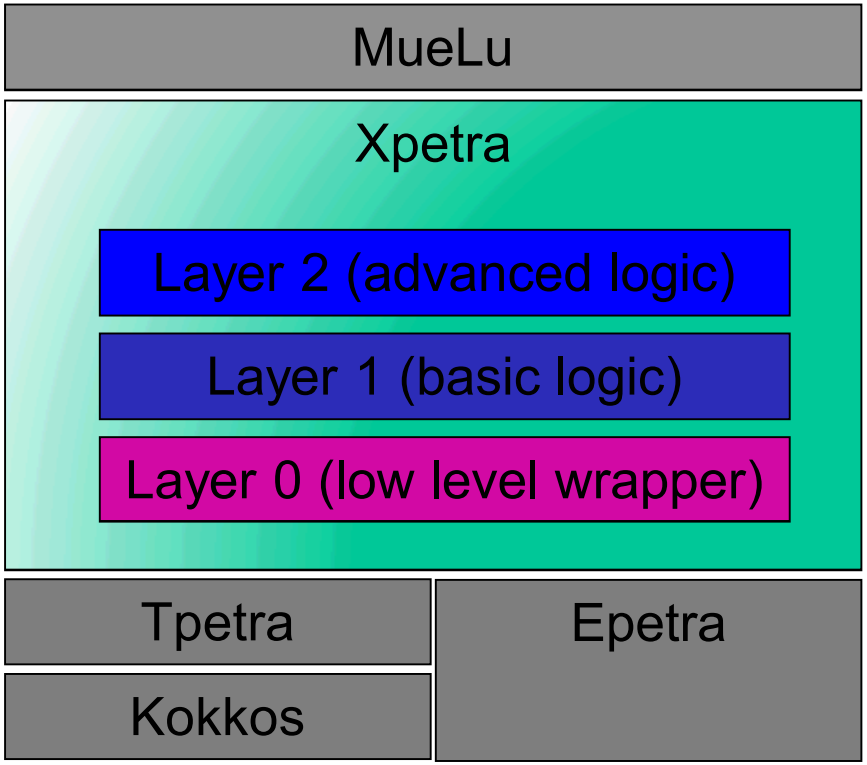
Current MueLu Capabilities

- Transfer operators
  - Smoothed aggregation
  - Nonsmoothed aggregation
  - Petrov Galerkin
  - Energy minimization
- Smoothers and direct solvers
  - Ifpack/Ifpack2 (Jacobi, Gauss-Seidel, ILU, polynomial, …)
  - Amesos/Amesos2 (KLU, Umfpack, Superlu, …)
  - Block smoothers (Braess Sarazin, …)

We support both Epetra and Tpetra!

Xpetra

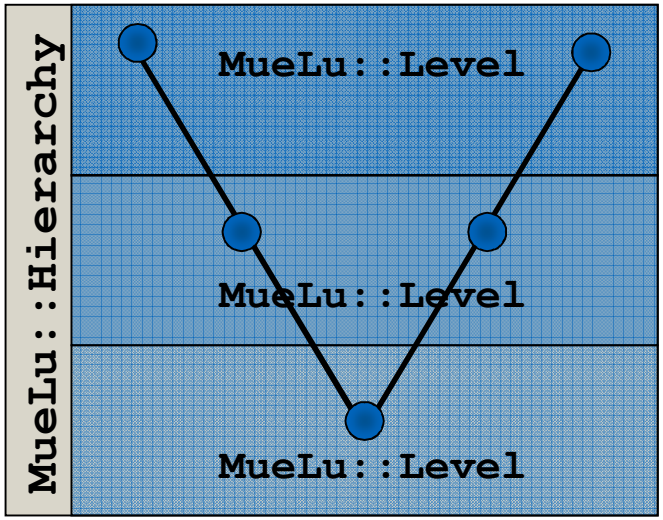
- Wrapper for Epetra and Tpetra
  - Based on Tpetra interfaces
  - Allows unified access to either linear algebra library
- Layer concept:
  - Layer 2: blocked operators
  - Layer 1: operator views
  - Layer 0: low level E/Tpetra wrappers (automatically generated code)
- MueLu algorithms are written using Xpetra



Design overview

Design

- Hierarchy
  - Generates and stores data
  - Provides multigrid cycles
- Factory
  - Generates data
- FactoryManager
  - Manages dependencies among factories



Preconditioner is created by linking together factories (constructing FactoryManager) and generating Hierarchy data using that manager.

User is not required to specify these dependencies.

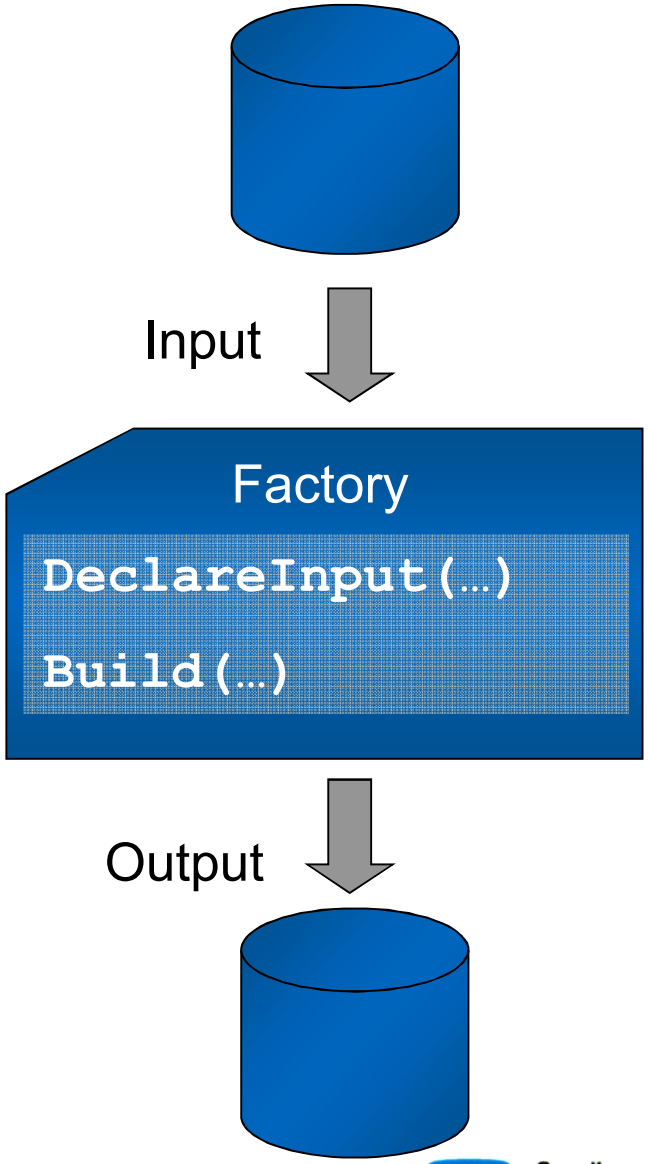
Factories

- **Factory processes input data (from Level) and generates some output data (stored in Level)**

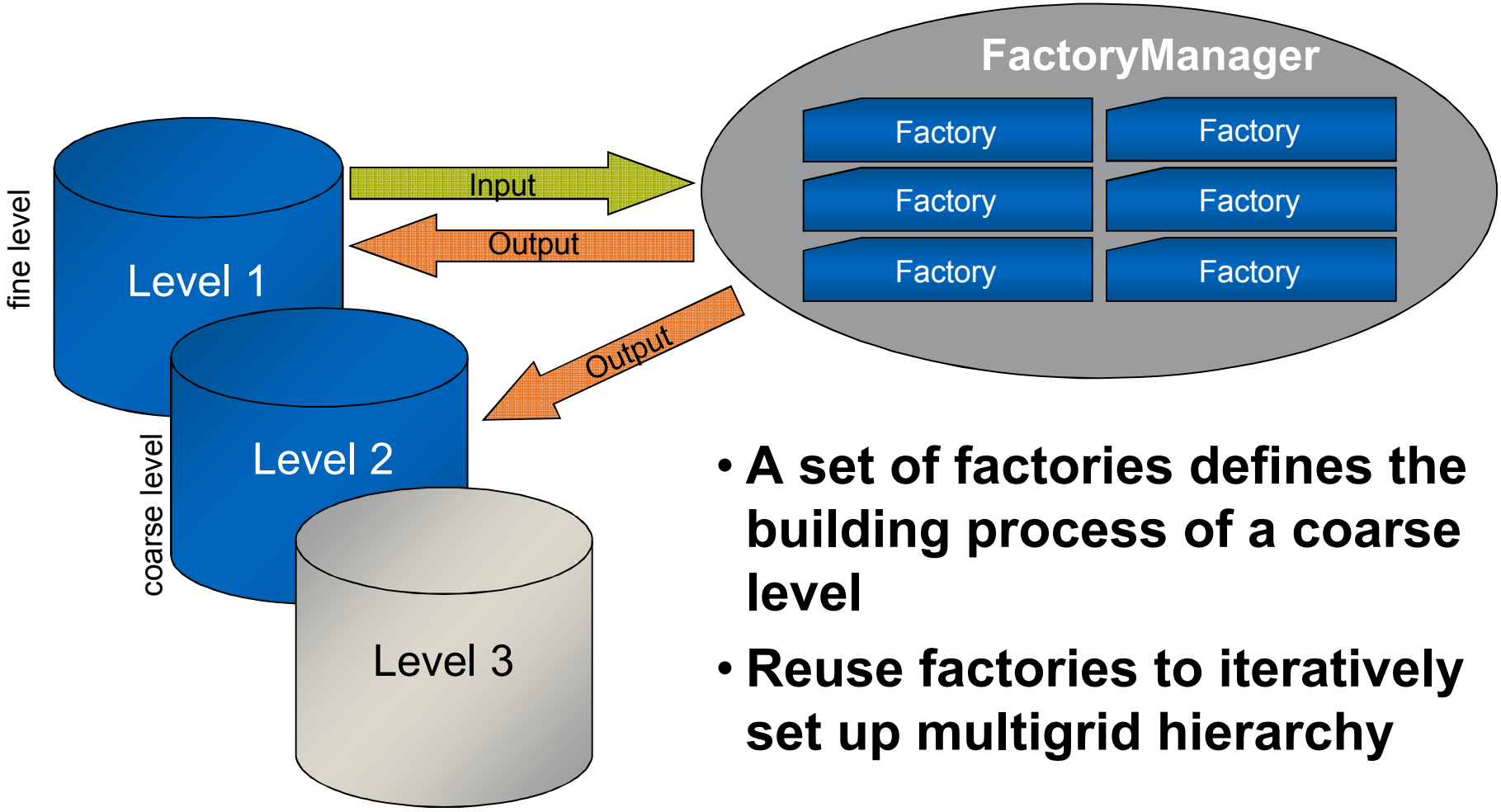
- **Two types of factories**
  - **Single level (smoothers, aggregation, ...)**
  - **Two level (prolongators)**

    Output is stored on next coarser level

Factory can generate more multiple output variables (e.g. „Ptent“ and „Nullspace“)



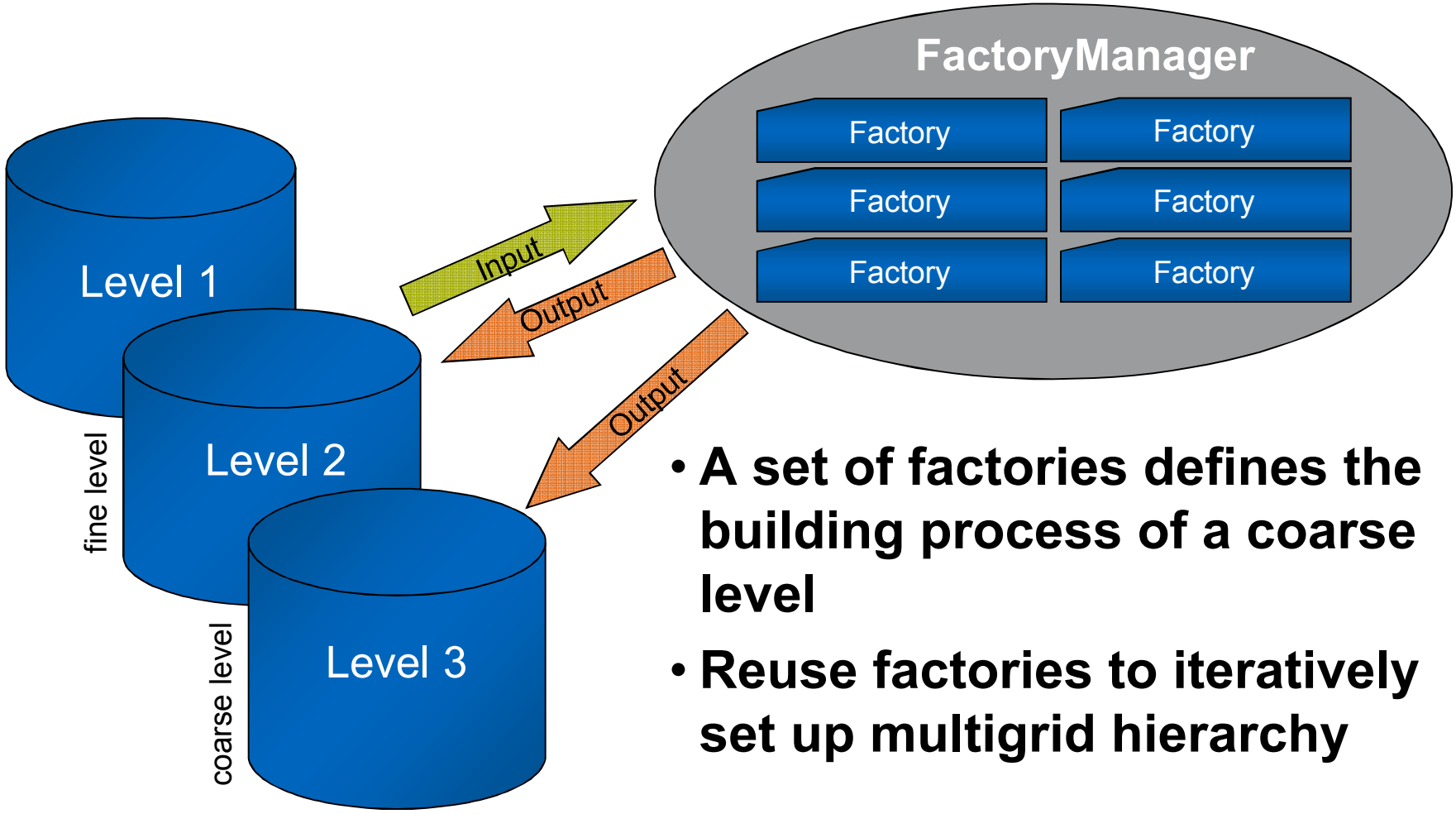
Multigrid hierarchy



- **A set of factories defines the building process of a coarse level**
- **Reuse factories to iteratively set up multigrid hierarchy**

Sandia National Laboratories

Multigrid hierarchy

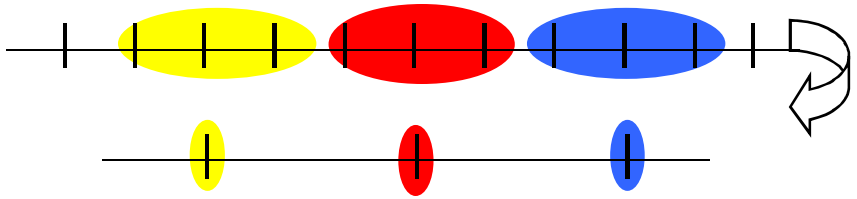


- **A set of factories defines the building process of a coarse level**
- **Reuse factories to iteratively set up multigrid hierarchy**

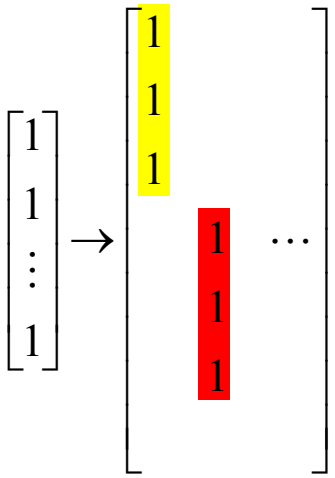
Sandia National Laboratories

Smoothed Aggregation Setup

- Group fine unknowns into *aggregates* to form coarse unknowns



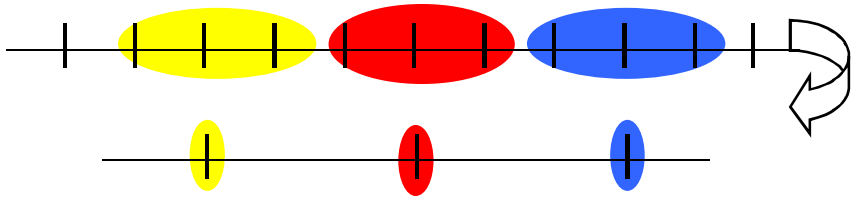
- Partition given nullspace $B_{(h)}$ across aggregates to have local support



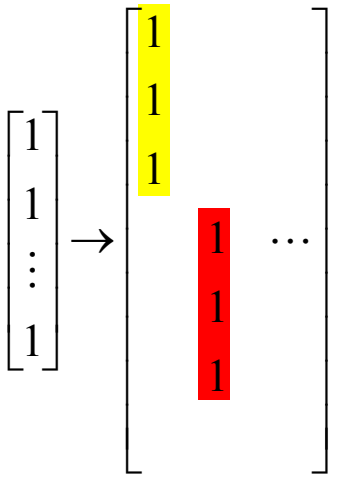
Sandia National Laboratories

Smoothed Aggregation Setup

- Group fine unknowns into *aggregates* to form coarse unknowns

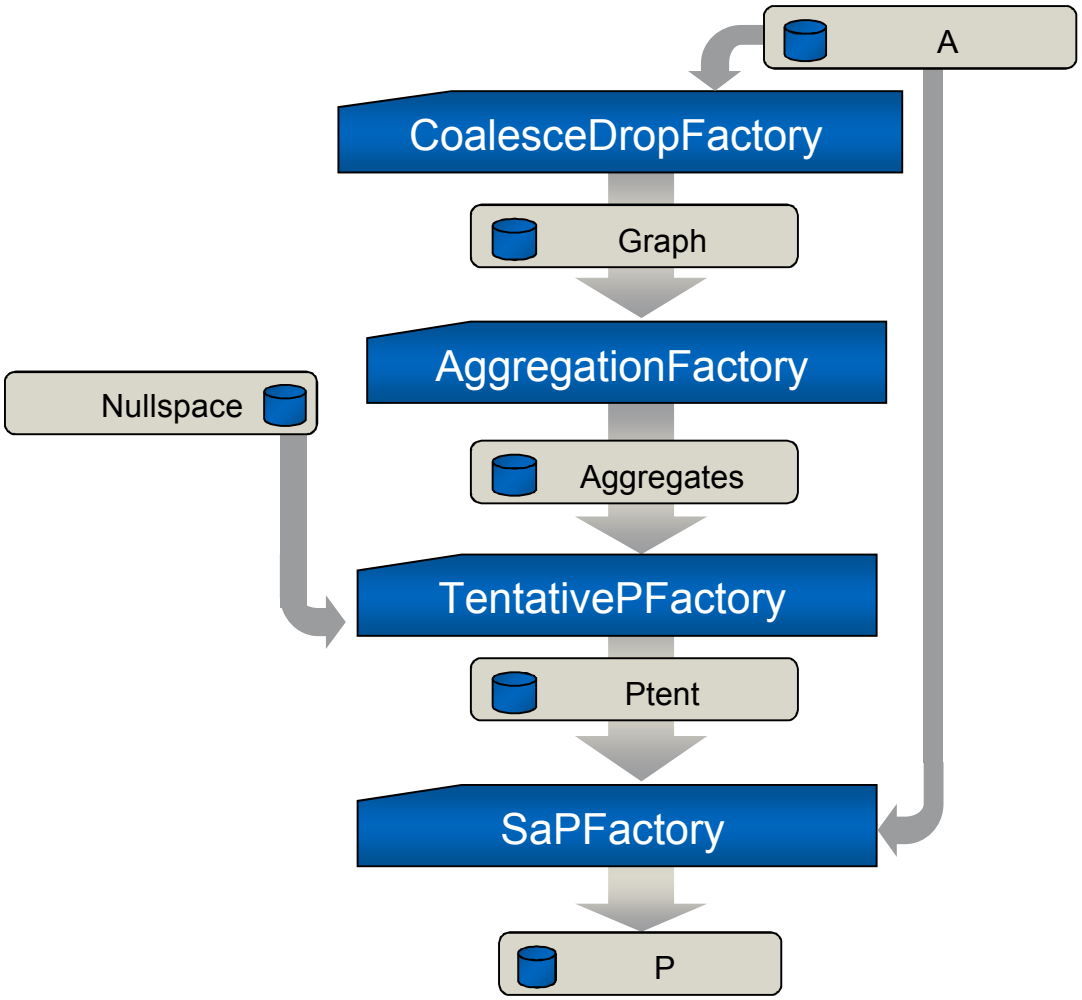


- Partition given nullspace $B_{(h)}$ across aggregates to have local support
- Calculate $QR=B_{(h)}$ to get initial prolongator P^{tent} (=Q) and coarse nullspace (R).

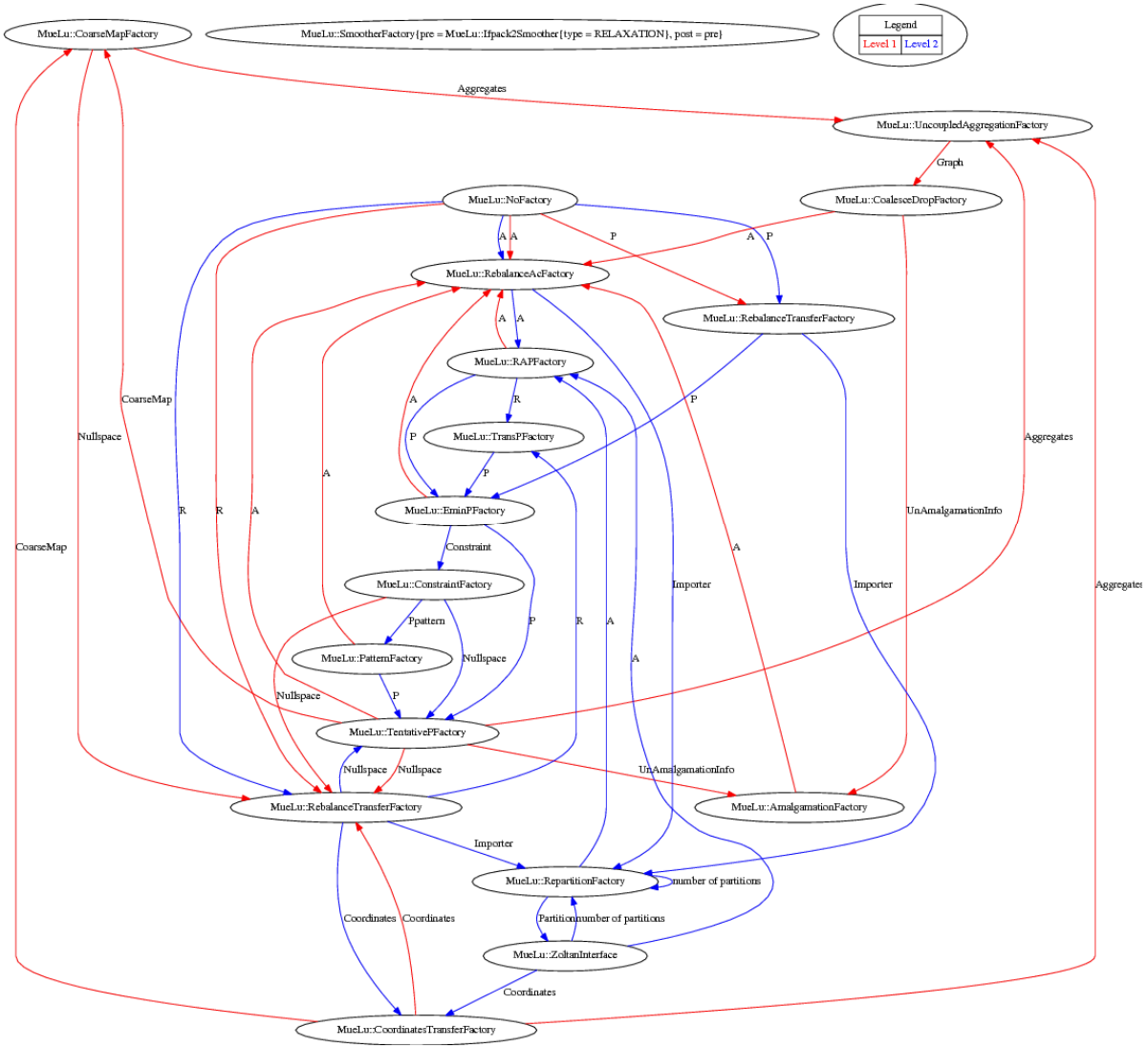


- **Form final prolongator** $P^{sm} = (I - \omega D^{-1} A) P^{tent}$

Linking factories



Linking factories



Sandia National Laboratories

Advantages of Data Management on Level

- Level manages data deallocation once all requests satisfied
- Generating factory does not need to know what other factories require data
- Data reuse
  - Any data (aggregates, *P*, …) can be retained by user request for reuse in later runs.
  - Data can be retained for later analysis.
  - Almost any reuse granularity is possible.

Muelu User interfaces

MueLu – User Interfaces

- MueLu can be customized as follows:
  - XML input files
  - Parameter lists (key-value pairs)
  - Directly through C++ interfaces
- New/casual users
  - Minimal interface
  - Sensible defaults provided automatically
- Advanced users
  - Can customize or replace any component of multigrid algorithm.

C++: smoothed aggregation

```
1  Hierarchy  H(fineA);      // generate hierarchy using fine level matrix
2
3  H.Setup();               // call multigrid setup (create hierarchy)
4
5  H.Iterate (B, nIts, X);  // perform nIts iterations with multigrid
6                           // algorithm (V-Cycle)
```

- Generates smoothed aggregation AMG
- Uses reasonable defaults.
- Every component can be easily changed

C++: unsmoothed aggregation

```
1  Hierarchy  H(fineA);          // generate hierarchy using fine level matrix
2
3  RCP<TentativePFactory  > PFact  = rcp(new  TentativePFactory  ());
4  FactoryManager   M;           // construct factory manager
5  M.SetFactory ("P", PFact);    // define tentative prolongator factory
6                                // as default factory for generating P
7
8  H.Setup(M);                   // call multigrid setup (create hierarchy)
9
10 H.Iterate (B, nIts , X);      // perform nIts iterations with multigrid
11                               // algorithm (V-Cycle)
```

- Generates unsmoothed prolongator

Sandia National Laboratories

C++: unsmoothed aggregation

```
1  Hierarchy  H(fineA);          // generate hierarchy using fine level matrix
2
3  RCP<TentativePFactory  > PFact  = rcp(new  TentativePFactory  ());
4  FactoryManager   M;           // construct factory manager
5  M.SetFactory ("P", PFact);    // define tentative prolongator factory
6                                // as default factory for generating P
7
8  H.Setup(M);                   // call multigrid setup (create hierarchy)
9
10 H.Iterate (B, nIts , X);      // perform nIts iterations with multigrid
11                               // algorithm (V-Cycle)
```

- Generates unsmoothed prolongator

C++: polynomial smoother

```
1  Hierarchy  H(fineA);      // generate hierarchy using fine level matrix
2
3  Teuchos::ParameterList   smootherParams;
4  smootherParams.set("chebyshev: degree  ", 3);
5
6  RCP<SmootherPrototype> smooProto =
7      rcp(new TrilinosSmoother ("Chebyshev ", smootherParams));
8
9  RCP<SmootherFactory> smooFact =
10     rcp(new SmootherFactory (smooProto));
11
12 FactoryManager  M;
13 M.SetFactory ("Smoother ", smooFact);
14
15 H.Setup(M);              // call multigrid setup (create hierarchy)
16
17 H.Iterate (B, nIts, X);  // perform nIts iterations with multigrid
18                          // algorithm (V-Cycle)
```

- Uses degree 3 polynomial smoother

Sandia National Laboratories

XML: creating hierarchy

```
1 ParameterListInterpreter    mueluFactory (xmlFile );
2 RCP<Hierarchy > H = mueluFactory .CreateHierarchy ();
3 H->GetLevel (0)->Set("A", fineA );
4
5 mueluFactory .SetupHierarchy (*H);
6
7 H->Iterate (B, nIts , X);
```

Sandia National Laboratories

XML: smoothed aggregation

```xml
<ParameterList   name="MueLu" >
    <Parameter   name="verbosity"   type="string"  value="high" />

    <Parameter   name="max levels"   type="int"  value="10" />
    <Parameter   name="coarse: max size"   type="int"  value="2000" />

    <Parameter   name="number of equations"   type="int"  value="1" />

    <Parameter   name="algorithm"   type="string"  value="sa" />

</ParameterList>
```

- Generates smoothed aggregation AMG
- Uses reasonable defaults

XML: unsmoothed aggregation

```
1  <ParameterList   name="MueLu" >
2      <Parameter   name="verbosity"   type="string"  value="high" />
3
4      <Parameter   name="max levels"   type="int"  value="10" />
5      <Parameter   name="coarse: max size"    type="int"  value="2000" />
6
7      <Parameter   name="number of equations"    type="int"  value="1" />
8
9      <Parameter   name="algorithm"   type="string"  value="unsmoothed"  />
10
11 </ParameterList>
```

- Generates unsmoothed prolongator

XML: polynomial smoother

```
1  <ParameterList   name="MueLu" >
2      <Parameter   name="verbosity"   type="string"  value="high" />
3
4      <Parameter   name="max levels"   type="int"  value="10" />
5      <Parameter   name="coarse: max size"    type="int"  value="2000" />
6
7      <Parameter   name="number of equations"    type="int"  value="1" />
8
9      <Parameter   name="algorithm"   type="string"  value="sa" />
10
11     <Parameter           name="smoother: type"   type="string"  value="CHEBYSHEV" />
12     <ParameterList       name="smoother: params"  >
13         <Parameter         name="chebyshev: degree"    type="int"  value="3" />
14     </ParameterList>
15
16 </ParameterList>
```

- Uses degree 3 polynomial smoother

Sandia National Laboratories

XML: polynomial smoother only for level 2

```
1  <ParameterList   name="MueLu" >
2      <Parameter   name="verbosity"   type="string"  value="high" />
3
4      <Parameter   name="max levels"   type="int"  value="10" />
5      <Parameter   name="coarse: max size"    type="int"  value="2000" />
6
7      <Parameter   name="number of equations"    type="int"  value="1" />
8
9      <Parameter   name="algorithm"   type="string"  value="sa" />
10
11     <ParameterList   name="level 2" >
12         <Parameter   name="smoother: type"   type="string"  value="CHEBYSHEV" />
13         <ParameterList   name="smoother: params"  >
14             <Parameter   name="chebyshev: degree"    type="int"  value="3" />
15         </ParameterList>
16     </ParameterList>
17
18 </ParameterList>
```

- Uses degree 3 polynomial smoother for level 2
- Uses default smoother (Gauss-Seidel) for all other levels

Summary

- Current status
  - Part of publicly available Trilinos anonymous clone
  - We still support ML.
- Ongoing/Future work
  - Preparing for public release
    - Improving documentation
    - Improving application interfaces
  - Improving performance
  - Integrating existing algorithms
  - Developing new algorithms