SAND2016-10687C

Epetra & Tpetra (Sparse linear algebra) overview

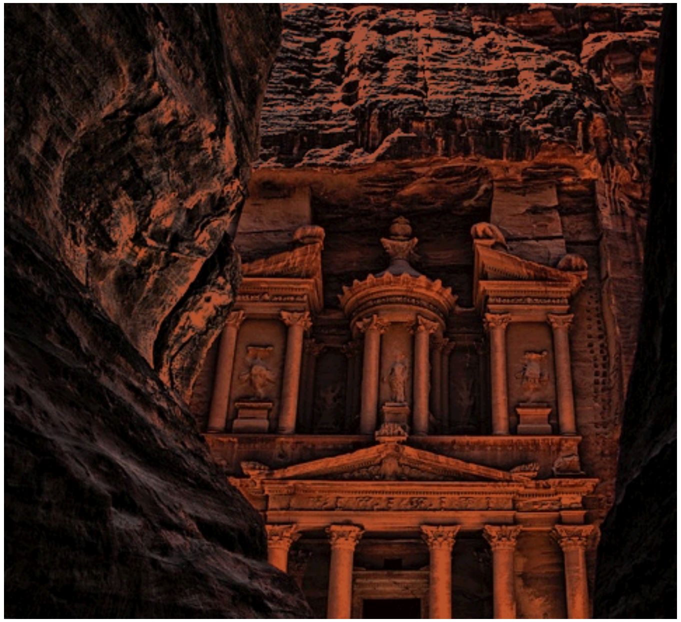
Mark Hoemmen & Alicia Klinvex
Sandia National Laboratories

24 Oct 2016

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.

Trilinos' Common Language: Petra

- "Common language" for distributed sparse linear algebra
- Petra[1] provides parallel…
  - Sparse graphs & matrices
  - Dense vectors & multivectors
  - Data distributions & redistribution
- "Petra Object Model":
  - Describes objects & their relationships abstractly, independent of language or implementation
  - Explains how to construct, use, & redistribute parallel graphs, matrices, & vectors
- We maintain 2 implementations



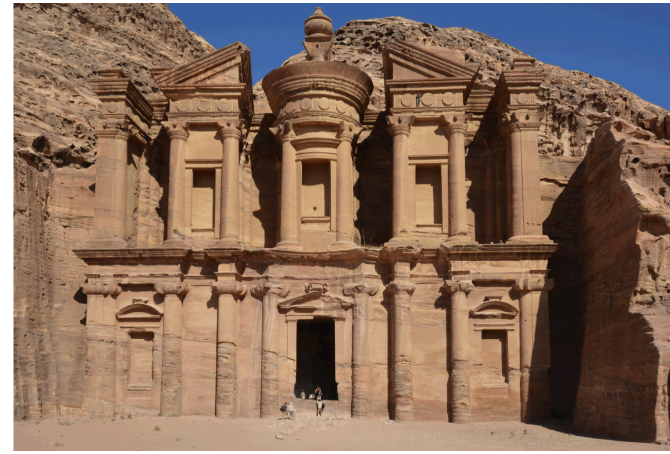
Al Khazneh ("The Treasury"), in the ancient city of Petra, in modern Jordan.

[1]Petra (πέτρα) is Greek for "foundation."

Sandia National Laboratories

Petra Implementations

- Epetra (Essential Petra):
  - Earliest & most heavily used
  - C++ <= 1998 (“C+/- compilers” OK)
  - Real, double-precision arithmetic
  - C & Fortran interfaces
  - MPI only (very little OpenMP support)
  - Some support for problems with over two billion unknowns (“Epetra64”)
- Tpetra (Templated Petra):
  - Supports & requires C++11 (as of 11.14)
  - Real, complex, extended-precision, automatic differentiation, etc. types
  - Can solve problems with > 2B unknowns
  - “MPI+X” (shared-memory parallel)



Al Deir (“The Monastery”) at Petra.

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

Sandia National Laboratories

Two "software stacks": Epetra & Tpetra

- Many packages were built on Epetra's interface
- Users want features that break interfaces
  - Support for solving huge problems (> 2B entities)
  - Arbitrary & 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
- Some packages can work with either Epetra or Tpetra
  - Iterative linear solvers & eigensolvers (Belos, Anasazi)
  - Multilevel preconditioners (MueLu), sparse direct (Amesos2)
- Which do I use?
  - Epetra is more stable; Tpetra is more forward-looking
  - For MPI only, their performance is comparable
  - For MPI+X, Tpetra will be the only path forward

Kokkos: Thread-parallel programming model & more

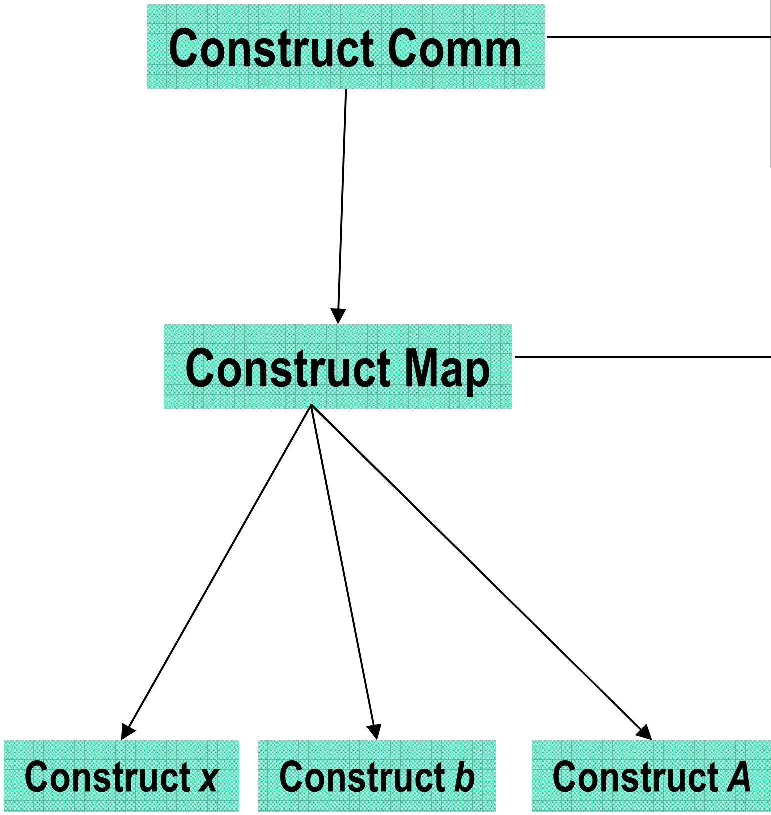
- Performance-portable abstraction over many different thread-parallel programming models: OpenMP, CUDA, Pthreads, …
  - Avoid risk of committing code to hardware or programming model
  - C++ library: Widely used, portable language with good compilers
- Abstract away physical data layout & target it to the hardware
  - Solve "array of structs" vs. "struct of arrays" problem
- Expose different memory & execution spaces
- Data structures & idioms for thread-scalable parallel code
  - Multi-dimensional arrays, hash table, sparse graph & matrix
  - Automatic memory management, atomic updates, vectorization, ...
- Stand-alone; does not require other Trilinos packages
  - Used in LAMMPS molecular dynamics code
  - Growing use in Trilinos; other apps starting too

Carter Edwards, Christian Trott, Dan Sunderland, Jim Foucar, Mark Hoemmen

Trilinos

Petra distributed object model

Solving Ax = b:
Typical Petra Object Construction Sequence



- Comm: Assigns ranks to processes
- Any number of Comm objects can exist
- Comms can be nested (e.g., serial within MPI)

- Maps describe a parallel layout
- Multiple objects can share the same Map
- Two Maps (source & target) define a communication pattern (Export or Import)

- Computational objects
- Compatibility assured via common Map

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);
```

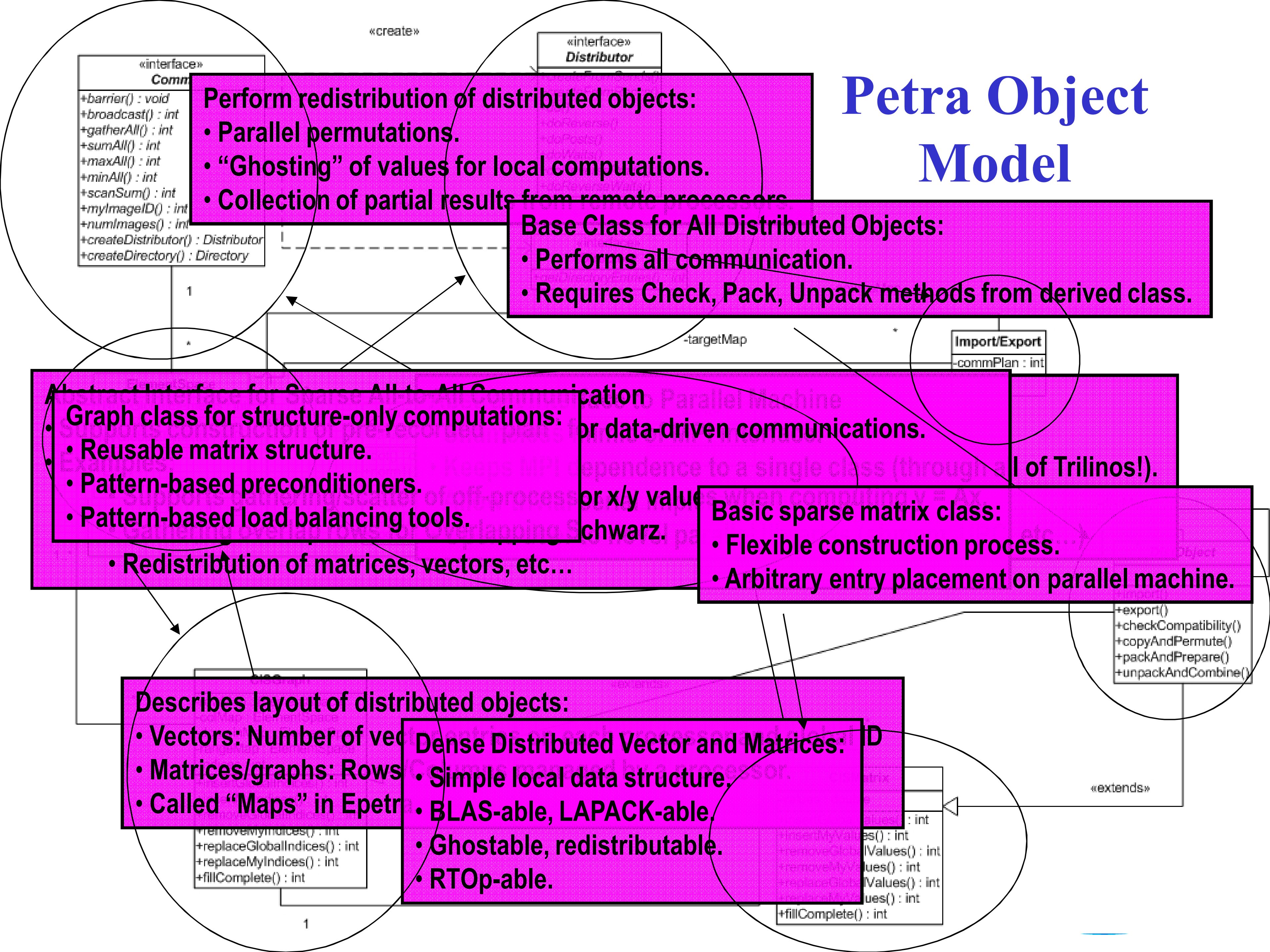
```
  // ***** Create/define AztecOO instance, solve *****
  AztecOO solver(problem);
  solver.SetAztecOption(AZ_precond, AZ_Jacobi);
  solver.Iterate(1000, 1.0E-8);
```

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


  return 0;
}
```

Sandia National Laboratories

Petra Object Model



A Map describes a data distribution

- A Map…
  - has a Comm(unicator)
  - is like a vector space
  - assigns entries of a data structure to (MPI) processes
- Global vs. local indices
  - You care about global indices (independent of # processes)
  - Computational kernels care about local indices
  - A Map "maps" between them
- Parallel data redistribution = function betw. 2 Maps
  - That function is a "communication pattern"
  - {E,T}petra let you precompute (expensive) & apply (cheaper) that pattern repeatedly to different vectors, matrices, etc.

1-to-1 Maps

- A Map is 1-to-1 if…
  - Each global index appears only once in the Map
  - (and is thus associated with only a single process)
- For data redistribution, {E,T}petra cares whether source or target Map is 1-to-1
  - "Import": source is 1-to-1
  - "Export": target is 1-to-1
- This (slightly) constraints Maps of a matrix:
  - Domain Map must be 1-to-1
  - Range Map must be 1-to-1

2D Objects: Four Maps

- Epetra 2D objects: graphs and matrices

Typically a 1-to-1 map

Typically NOT a 1-to-1 map

- Have four maps:
  - **Row Map**: On each process, 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 $y = Ax$).

Must be 1-to-1 maps!!!

Sandia National Laboratories

Sample Problem

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

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

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

- Row Map = {0, 1}
- Column Map = {0, 1, 2}
- Domain Map = {0, 1}
- Range Map = {0, 1}

PE 1 Contents

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

- Row Map = {2}
- Column Map = {1, 2}
- Domain Map = {2}
- Range Map = {2}

Original Problem

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

Notes:

- Rows are wholly owned.
- Row Map = Domain = Range (all 1-to-1).
- Column Map 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

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

- Row Map = {0, 1}
- Column Map = {0, 1, 2}
- Domain Map = {1, 2}
- Range Map = {0}

PE 1 Contents

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

- Row Map = {2}
- Column Map = {1, 2}
- Domain Map = {0}
- Range Map = {1, 2}

Original Problem

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

Notes:

- Rows are wholly owned.
- Row Map NOT = Domain Map
  NOT = Range Map (all 1-to-1).
- Column Map 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

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

- Row Map = {0, 1}
- Column Map = {0, 1}
- Domain Map = {1, 2}
- Range Map = {0}

PE 1 Contents

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

- Row Map = {1, 2}
- Column Map = {1, 2}
- Domain Map = {0}
- Range Map = {1, 2}

Original Problem

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

Notes:

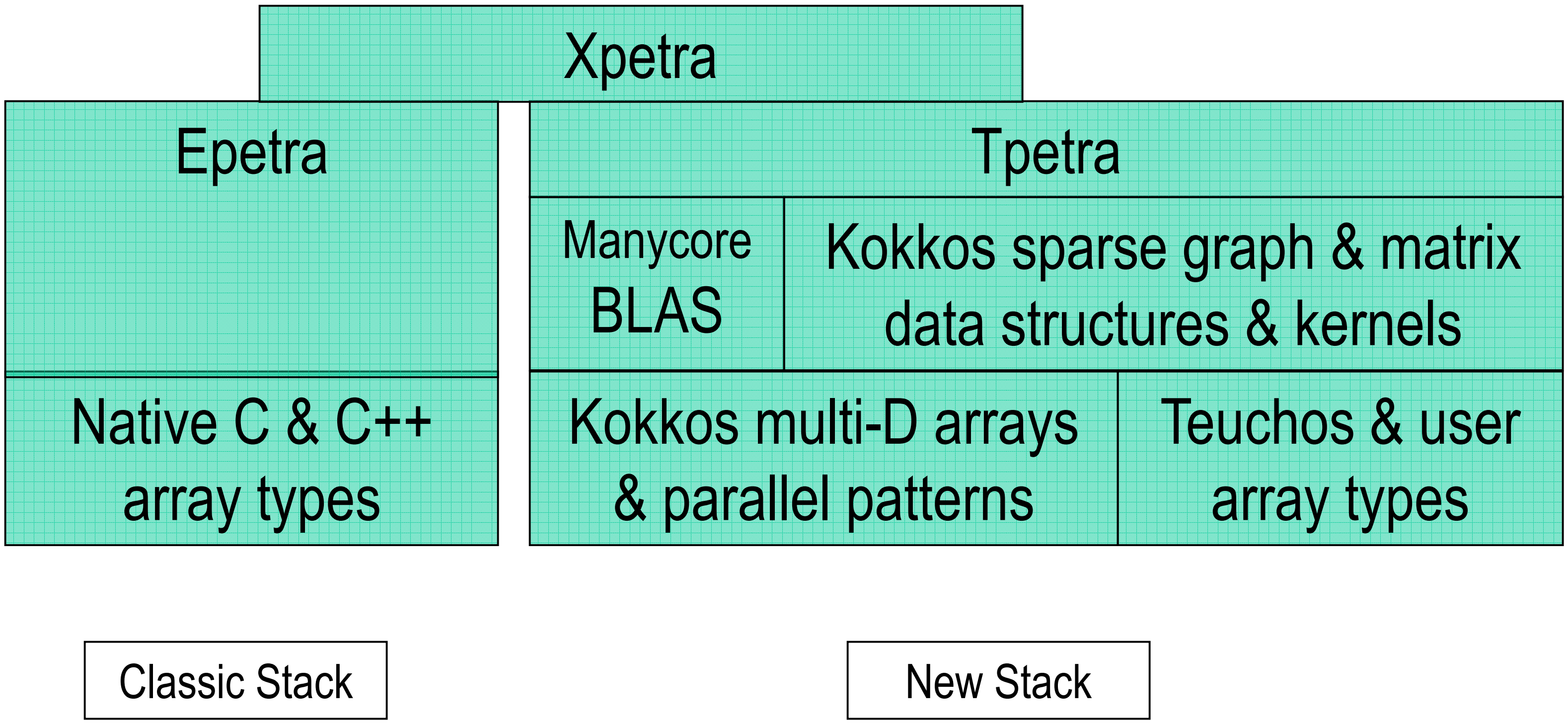
- Rows are NOT wholly owned.
- Row Map NOT = Domain Map
  NOT = Range Map (all 1-to-1).
- Row Map and Column Map NOT 1-to-1.
- Call to fillComplete:
  A.fillComplete(domainMap, rangeMap);

What does fillComplete do?

- Signals you're done
  - Defining graph structure of the matrix
  - Modifying the matrix's values
- Creates communication patterns for distributed sparse matrix-vector multiply:
  - If Column Map ≠ Domain Map, create Import
  - If Row Map ≠ Range Map, create Export
- A few rules:
  - Non-square matrices will *always* require:

    ```
    A.fillComplete(domainMap,rangeMap);
    ```
  - Domain Map & Range Map *must be 1-to-1*

Data Classes Stacks



Trilinos

Questions?

Sandia National Laboratories