Lab 05 – Mini-Lecture
Third Party Libraries

Prof. Brendan Kochunas

NERS/ENGR 570 - Methods and Practice of Scientific Computing (F20)

COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Outline

- BLAS/LAPACK
- PETSc
- Trilinos

Learning Objectives: By the end of Today's Lecture you should be able to

- (Knowledge) quickly navigate documentation for BLAS, LAPACK, PETSc, and Trilinos to find more information about a particular capability of the library

Scientific Computing Libraries

BLAS, LAPACK, PETSc, Trilinos, and Others

BLAS (Basic Linear Algebra Subprograms)

- BLAS is a (Fortran) programming interface to low-level linear algebra routines.
- Examples of basic linear algebra operations:
  - Dot products, addition of vectors, scalar multiplication of vectors.
  - Matrix-vector multiplications.
  - Matrix-matrix multiplications.
- Why BLAS?
  - Handwritten simple linear algebra routines can run at widely varying speeds.
  - Loop unrolling and finding correct compiler flags is key and sometimes becomes difficult
  - when using a new compiler or computer.
  - Basic linear algebra routines form the backbone of many sophisticated solvers and BLAS package tries to provide most optimized version of basic linear algebra routines.
  - Linking to BLAS, we get code that runs much faster than hand-written version of code.

BLAS (Basic Linear Algebra Subprograms)

- BLAS-1 operations:
  - Routines which involve only vector operations: dot-products, vector norms.
  - S – Single Precision, D - Double Precision, C – Complex, Z – Double precision complex

```
_SWAP      x <---> y                  S, D
_SCAL      x <-- alpha * x            S, D, C, Z, CS, ZD
_COPY      x <-- y                    S, D, C, Z
_AXPY      y <-- alpha * x + y        S, D, C, Z
_DOT       dot <--                    S, D, DS
_DOTU      dot <-- x^T*y              C, Z
_DOTC      dot <-- x^H*y              C, Z
_NRM2      nrm2 <-- || x ||_2         S, D, SC, DZ
_ASUM      nrm1 <-- || x ||_1         S, D, SC, DZ
```

BLAS (Basic Linear Algebra Subprograms)

- BLAS-2 operations:
  - This level contains matrix-vector operations including, among other things, a generalized matrix-vector multiplication (gemv):

```
_GEMV      y <-- alpha*A*x + beta*y, y <-- alpha*A^T*x + beta*y  (General Real Matrix)
_GBMV      y <-- alpha*A*x + beta*y, y <-- alpha*A^T*x + beta*y  (General Banded)
_HEMV      y <-- alpha*A*x + beta*y                              (Hermitian (complex))
_HBMV      y <-- alpha*A*x + beta*y                              (Hermitian (banded))
_SYMV      y <-- alpha*A*x + beta*y                              (Symmetric)
_SBMV      y <-- alpha*A*x + beta*y                              (Symmetric banded)
_TRMV      y <-- A*x , x <-- A^T*x                               (Triangular)
_TBMV      y <-- A*x , x <-- A^T*x                               (Triangular banded)
_TRSV      y <-- inv(A)*x , x <-- inv(A^T) * x
```

BLAS (Basic Linear Algebra Subprograms)

- BLAS-3 Matrix-matrix operations:
  - This level operations are matrix-matrix multiplications.

BLAS API

```
_GEMM  C <-- alpha op(A) op(B) + beta C
_SYMM  C <-- alpha AB + beta C
_HEMM  C <-- alpha AB + beta C
_SYRK  C <-- alpha A A^T + beta C
_HERK  C <-- alpha A A^H + beta C
_SYRK2 C <-- alpha A B^T + alpha B A^T + beta C
_TRMM  B <-- alpha op(A) B
_TRSM  B <-- alpha op(inv(A)) B
```

BLAS-Example Code

[API]

Sparse blas

CBLAS

PSBLAS

Reference

http://www.netlib.org/lapack/explore-html/

```
*  ===================================================================
      DOUBLE PRECISION FUNCTION ddot(N,DX,INCX,DY,INCY)
*
*  -- Reference BLAS level1 routine (version 3.4.0) --
*  -- Reference BLAS is a software package provided by Univ. of Tennessee,    --
*  -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
*     November 2011
*
*     .. Scalar Arguments ..
      INTEGER INCX,INCY,N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION DX(*),DY(*)
*     ..
*
*  =====================================================================
*
*     .. Local Scalars ..
      DOUBLE PRECISION DTEMP
      INTEGER I,IX,IY,M,MP1
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC mod
*     ..
      ddot = 0.0d0
      dtemp = 0.0d0
      IF (n.LE.0) RETURN
      IF (incx.EQ.1 .AND. incy.EQ.1) THEN
*
*        code for both increments equal to 1
*
*
*        clean-up loop
*
         m = mod(n,5)
         IF (m.NE.0) THEN
            DO i = 1,m
               dtemp = dtemp + dx(i)*dy(i)
            END DO
            IF (n.LT.5) THEN
               ddot=dtemp
            RETURN
            END IF
         END IF
         mp1 = m + 1
         DO i = mp1,n,5
          dtemp = dtemp + dx(i)*dy(i) + dx(i+1)*dy(i+1) +
     $            dx(i+2)*dy(i+2) + dx(i+3)*dy(i+3) + dx(i+4)*dy(i+4)
         END DO
      ELSE
*
```

f77

f90

LAPACK

- LA Pack is a collection of Fortran functions that can help you solve Linear Algebra related problems based on BLAS routines.
- LAPACK provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.
- The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided,
- Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.
- Online Reference: http://www.netlib.org/lapack

LAPACK Routines

- Simple driver routines: Simple driver routines solve a linear algebra problem
- Eg: Finding eigenvalues of a matrix, solving a set of linear equations etc.,
- Expert driver routines: Expert driver routines do the same things as simple driver routines, but will provide more options or information to the user.
- Eg: SGESV is used to solve linear systems whereas the expert driver SGESVX not only solves the linear system but will also provide the estimate of the condition number of input matrix.
- Computational routines: Routines are mainly for internal use by LA Pack itself and called by driver routines.
- Eg: LU, QR and other factorizations or reduction of symmetric matrix to tridiagonal form.

LAPACK Naming conventions

- LAPACK functions are usually named in the form XYYZZZ

X = type of problem that the routine solves
S Single precision real
D Double precision real
C Single complex
Z Double precision complex

YY = matrix types
GE General
BD Bidiagonal
HE Hermitian
HB Hermitian Band
SB Symmetric Band

ZZZ = indicate the computation performed
Eg: SV = Solve, SVX = Solve Expert

SVD

PETSc

- **P**ortable **E**xtensible **T**oolkit for **S**cientific computing:

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort. PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet. — Barry Smith

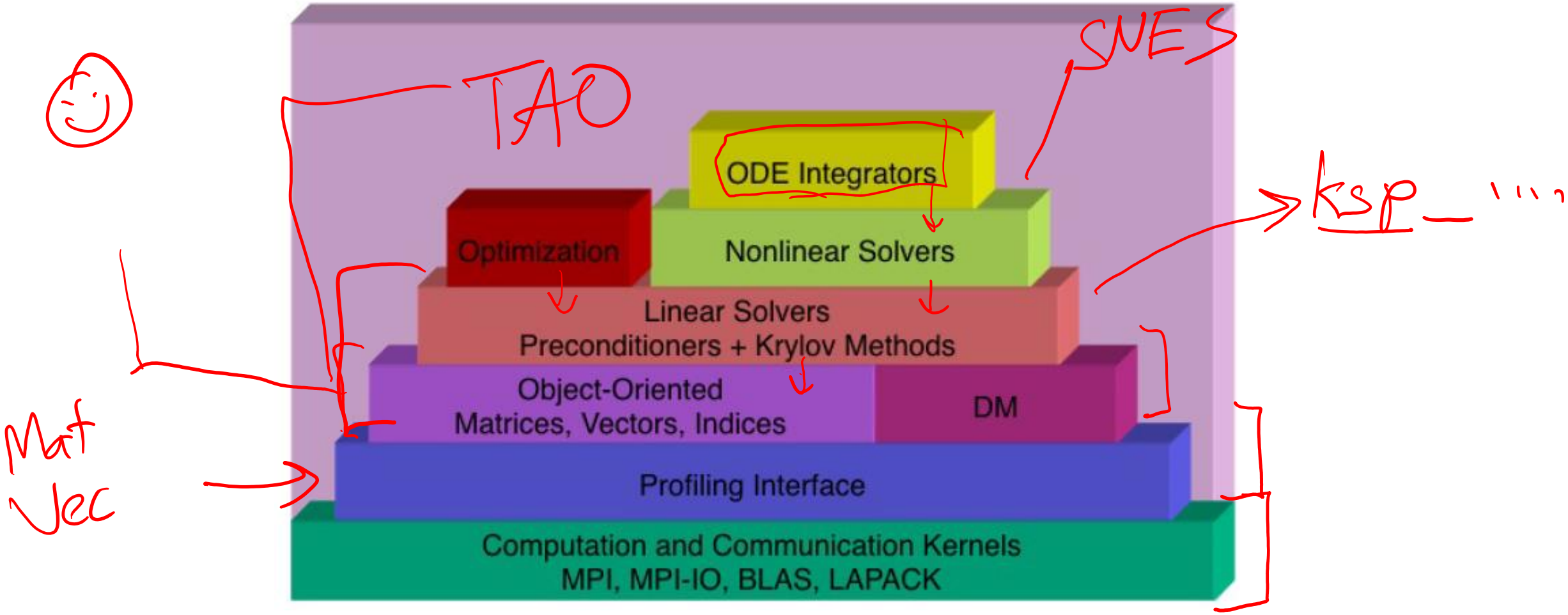
- Philosophy: Everything has a plugin architecture
  - Vectors, Matrices, Partitioning algorithms
  - Preconditioners, Krylov accelerators
  - Nonlinear solvers, Time integrators
  - Spatial discretizations
  - Application user loads plugin at run, no source code in sight. PETSc tries to keep solvers independent of physics and discretization.

$Ax = b$ solves

Values of A

determines structure of A

PETSc Structure



Basic PETSc object usage

Every object in PETSc supports a basic interface

Function	Operation
`Create()`	create the object
`Get/SetName()`	name the object
`Get/SetType()`	set the implementation type
`Get/SetOptionsPrefix()`	set the prefix for all options
`SetFromOptions()`	customize object from the command line
`SetUp()`	preform other initialization
`View()`	view the object
`Destroy()`	cleanup object allocation

Also, all objects support the `-help` option.

PETSc Vectors

- PETSc vectors are fundamental datatypes of PETSc that are used represent field solutions, right-hand sides etc.
- Each process locally owns a subvector of contiguous global data.
- Creating PETSc vectors:
  - `VecCreate(MPI_Comm, Vec *)`
  - `VecSetSizes(Vec, int n, int N)`
  - `VecSetTypes(Vec, VecType typename)`
  - `VecSetFromOptions(Vec)`
- Supports all vector space operations `VecDot()`, `VecNorm()`, `VecScale()`

PETSc Vectors

- Inserting entries into PETSc vectors:
  - Each process sets or add values and begins communications to send values to correct process and complete the communication.
  - `VecSetValues(Vec v, int n, int rows[], PetscScalar values[], mode)`
  - `VecAssemblyBegin(Vec v)`
  - `VecAssemblyEnd(Vec v)` → PETSc I'm all done
- PETSc allows you to access the local storage with `VecGetArray()` functions.

PETSc Matrices

- PETSc handles both sparse and dense matrix formats in parallel.

```
MatCreate(MPI_Comm, Mat *)
MatSetSizes(Mat, int m, int n, int M, int N)
MatSetType(Mat, MatType typeName)
MatSetFromOptions(Mat)
```

Can set the type at runtime

```
MatMPIBAIJSetPreallocation(Mat,...)
```

important for assembly performance

```
MatSetBlockSize(Mat, int bs)
```

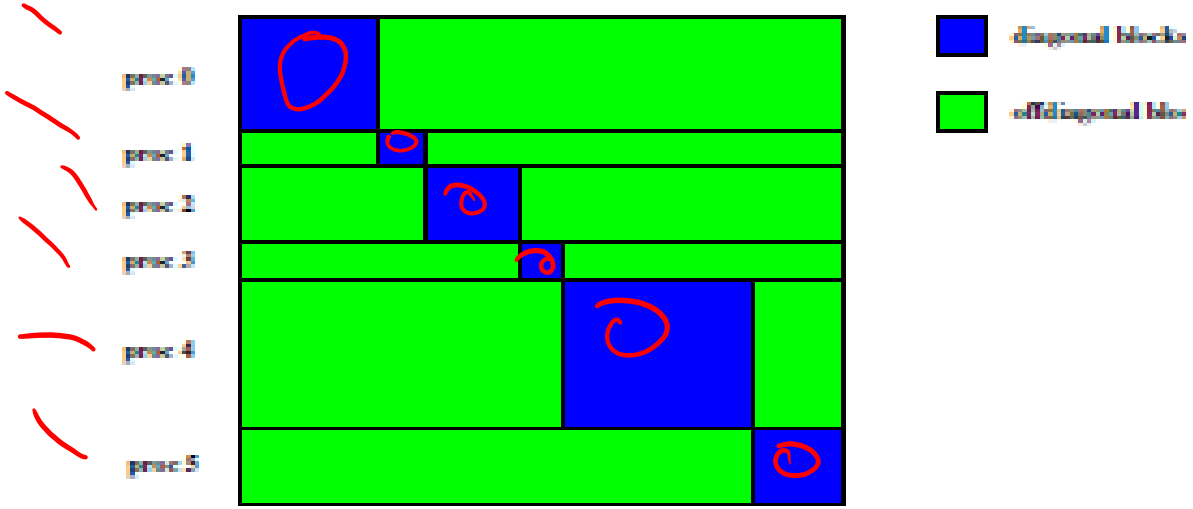
for vector problems

```
MatSetValues(Mat,...)
```

MUST be used, but does automatic communication
MatSetValuesLocal, MatSetValuesStencil,
MatSetValuesBlocked

Parallel sparse matrix in PETSc

Each process locally owns a submatrix of contiguous global rows
Each submatrix consists of diagonal and off-diagonal parts




```
MatGetOwnershipRange(Mat A, int *start, int *end)
```

`start`: first locally owned row of global matrix
`end-1`: last locally owned row of global matrix

Other useful features of PETSc

- Iterative solvers:
  - Linear solvers in PETSc KSP: Conjugate Gradient, Bi Conjugate Gradient, GMRES, etc.
  - Lots of sophisticated Preconditioners like SOR, block-ILU, multigrid, field-split, etc.
  - Nonlinear solvers (SNES):
    - Newton type with line search and trust-region
    - Quasi Newton methods
    - Nonlinear conjugate gradients
    - User-defined methods.
- Time Integration strategies

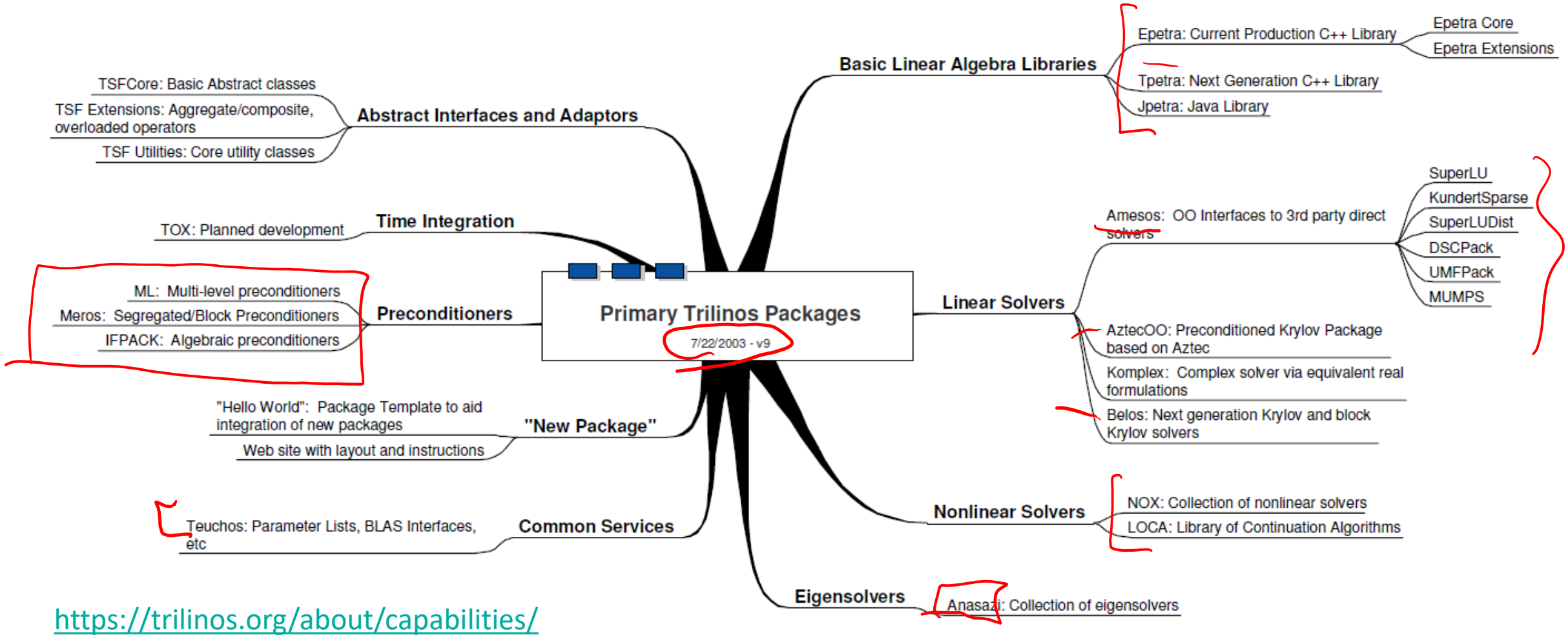
TRILINOS

- The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems. A unique design feature of Trilinos is its focus on packages.
- More extensive than PETSc and more complex to use.
- Trilinos tries to provide an environment for solving FEM problems and PETSc provides an environment for solving sparse linear algebra problems.

Capabilities

- Trilinos – Greek for "String of Pearls"
  - Most package names are Greek
  - Duplication of capability
  - Some are deprecated

User Experience	Parallel Programming Environments	I/O Support
Mesh & Geometry	Framework & Tools	Discretization
Linear Algebra Services	Linear & Eigen Solvers	Embedded Nonlinear Analysis
Software Engineering		



https://trilinos.org/about/capabilities/

FASTMATH

- Contains Tools for:
  - Problem Discretization
  - Solution of Algebraic Systems
  - Uncertainty Quantification
  - Numerical Optimization

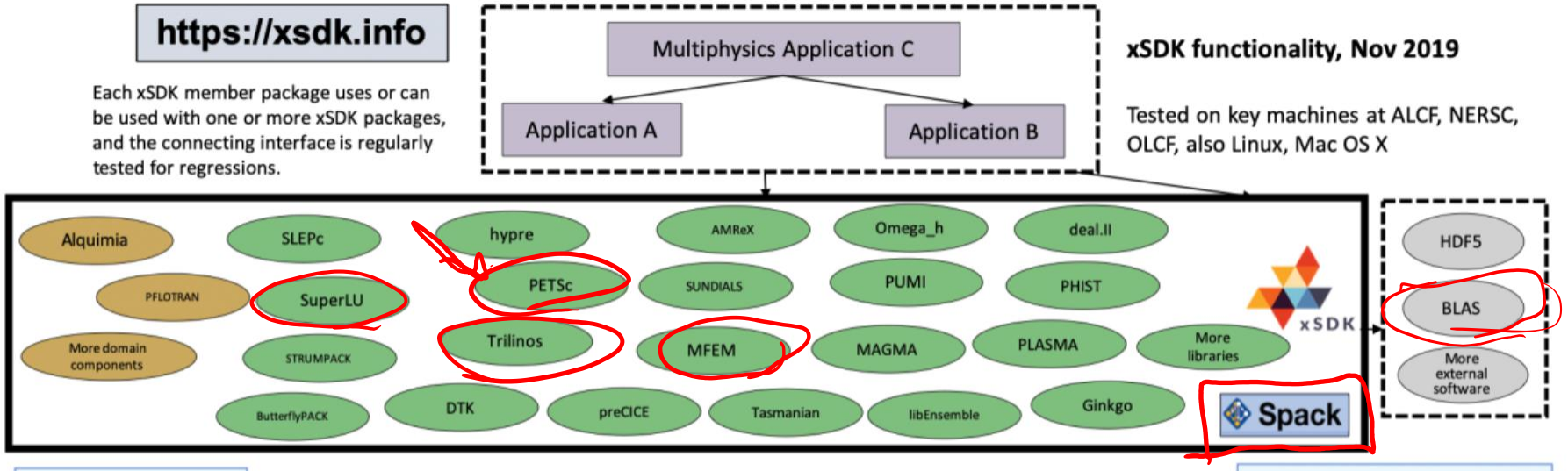
https://fastmath-scidac.llnl.gov/software-catalog.html

Integrating Libraries with your Code
eXstreme-scale Software Development Kit

xSDK Version 0.5.0: November 2019

https://xsdk.info

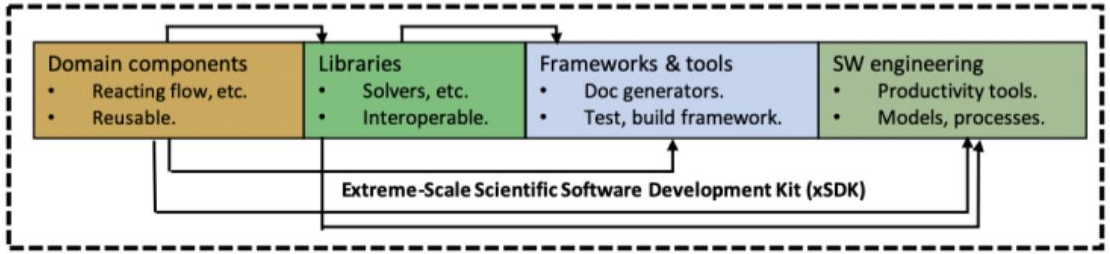
Each xSDK member package uses or can be used with one or more xSDK packages, and the connecting interface is regularly tested for regressions.



xSDK functionality, Nov 2019

Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X

November 2019
- 21 math libraries
- 2 domain components
- 16 mandatory xSDK community policies
- Spack xSDK installer



Impact: Improved code quality, usability, access, sustainability

Foundation for work on performance portability, deeper levels of package interoperability