

# Tutorial for PETSc

Barry Smith

Argonne National Laboratory

February 2006

# Presentation Objectives

- Who?
- When?
- Where
- Why?
- What is in PETSc?
- How to use PETSc?

# PETSc Development Team



Satish  
Balay



Victor  
Eijkhout



Matt  
Knepley

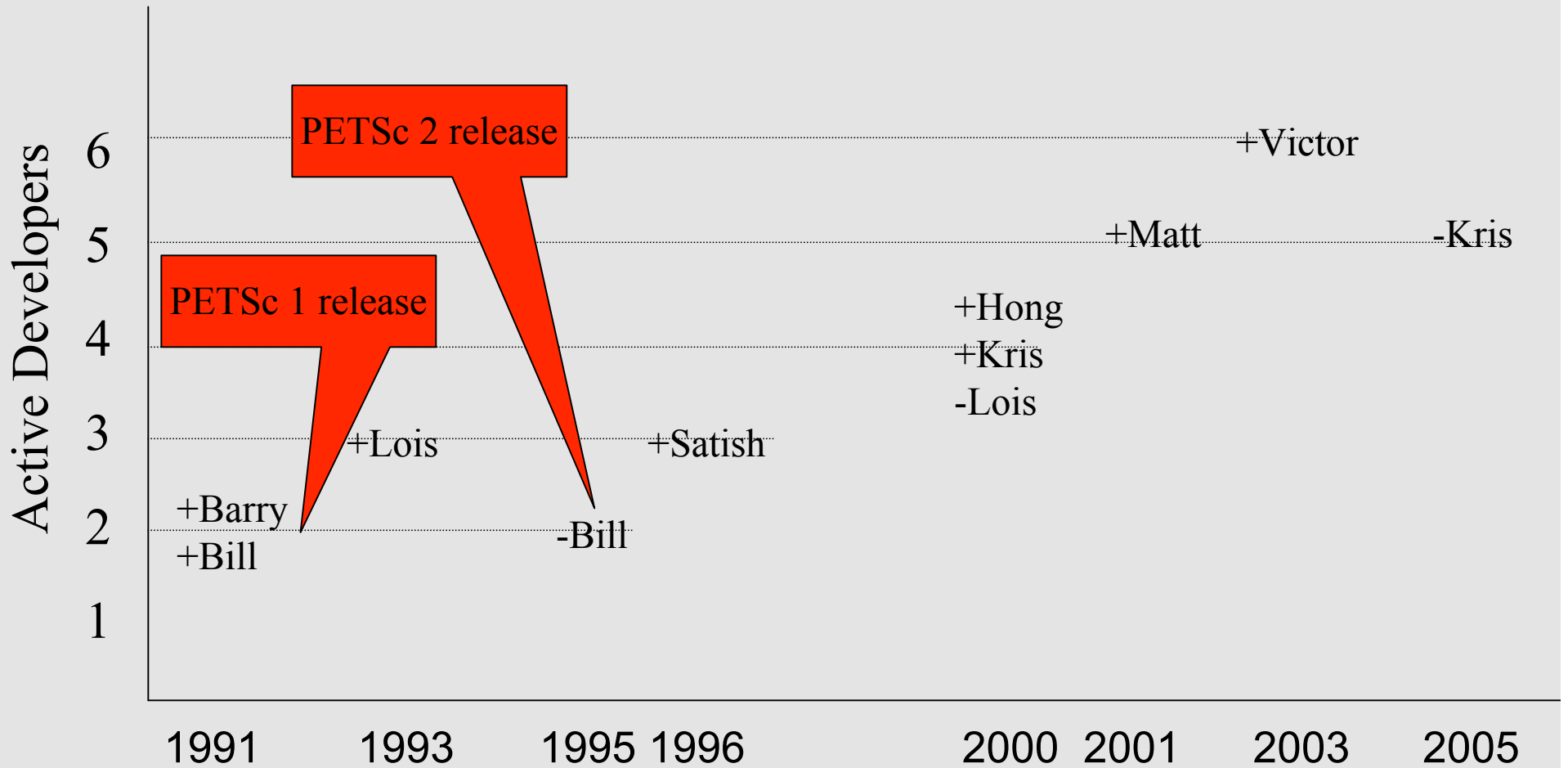


Barry  
Smith



Hong  
Zhang

# Timeline





# Where



# Why? Original Goals of PETSc

- Provide software for the **scalable** (parallel) solution of **algebraic systems** arising from **partial differential equation simulations** (PDEs).
  - Algebraic systems inherit structure from the grid and the PDE
- Original goals still strongly affect current design and functionality

# What is PETSc?

- Supported “research” code
  - Free for everyone, including industrial users
  - Hyperlinked documentation and manual pages for all routines
  - Many tutorial-style examples
  - Support via email: [petsc-maint@mcs.anl.gov](mailto:petsc-maint@mcs.anl.gov)
  - Usable from Fortran 77/90, C, and C++
- Portable(?) to virtually any system
  - Tightly coupled systems
    - Cray, SGI, IBM, HP, Sun
  - Loosely coupled systems, e.g., networks of workstations
    - Compaq, HP, IBM, SGI, Sun
    - Linux or Windows, Apple
- Funded largely by the US Department of Energy

# Features

- Support for/uses many numerical packages
- Many (parallel) vector/array operations
- Numerous (parallel) matrix formats
- **Numerous** linear solvers
- Nonlinear solvers
- Limited ODE integrators

# Support for/uses many numerical packages

- Packages can be automatically
  - downloaded
  - configured and built
  - installed in PETSc

# Interfaced Packages

## 1. LU (Sequential)

- SuperLU (Demmel and Li, LBNL)
- ESSL (IBM)
- Matlab
- LUSOL (from MINOS - Michael Saunders, Stanford)
- LAPACK
- PLAPACK (van de Geijn, UT Austin)
- UMFPACK (Timothy A. Davis)

## 2. Parallel LU

- SuperLU\_dist (Demmel and Li, LBNL)
- SPOOLES (Ashcroft, Boeing, funded by ARPA)
- MUMPS (European)
- PLAPACK (van de Geijn, UT Austin)

## 3. Parallel Cholesky –

- DSCPACK (Raghavan, Penn. State)
- SPOOLES (Ashcroft, Boeing, funded by ARPA)
- PLAPACK (van de Geijn, UT Austin)

# Interfaced Packages

4. **XYTlib** – parallel direct solver (Fischer and Tufo, ANL)
5. **SPAI** – Sparse approximate inverse (parallel)
  - Parasails (Chow, part of Hypre, LLNL)
  - SPAI 3.0 (Grote/Barnard)
6. **Algebraic multigrid**
  - Parallel BoomerAMG (part of Hypre, LLNL)
  - ML (part of Trilinos, SNL)
7. **Parallel ICC(0)** – BlockSolve95 (Jones and Plassman, ANL)
8. **Parallel ILU**
  - BlockSolve95 (Jones and Plassman, ANL)
  - PILUT (part of Hypre, LLNL)
  - EUCLID (Hysom – also part of Hypre, ODU/LLNL)
9. **Sequential ILU-DT** (SPARSEKIT2- Y. Saad, U of MN)

# Interfaced Packages

## 10. Parititioning

- Parmetis
- Chaco
- Jostle
- Party
- Scotch

## 11. ODE integrators

- Sundials (LLNL)

## 12. Eigenvalue solvers

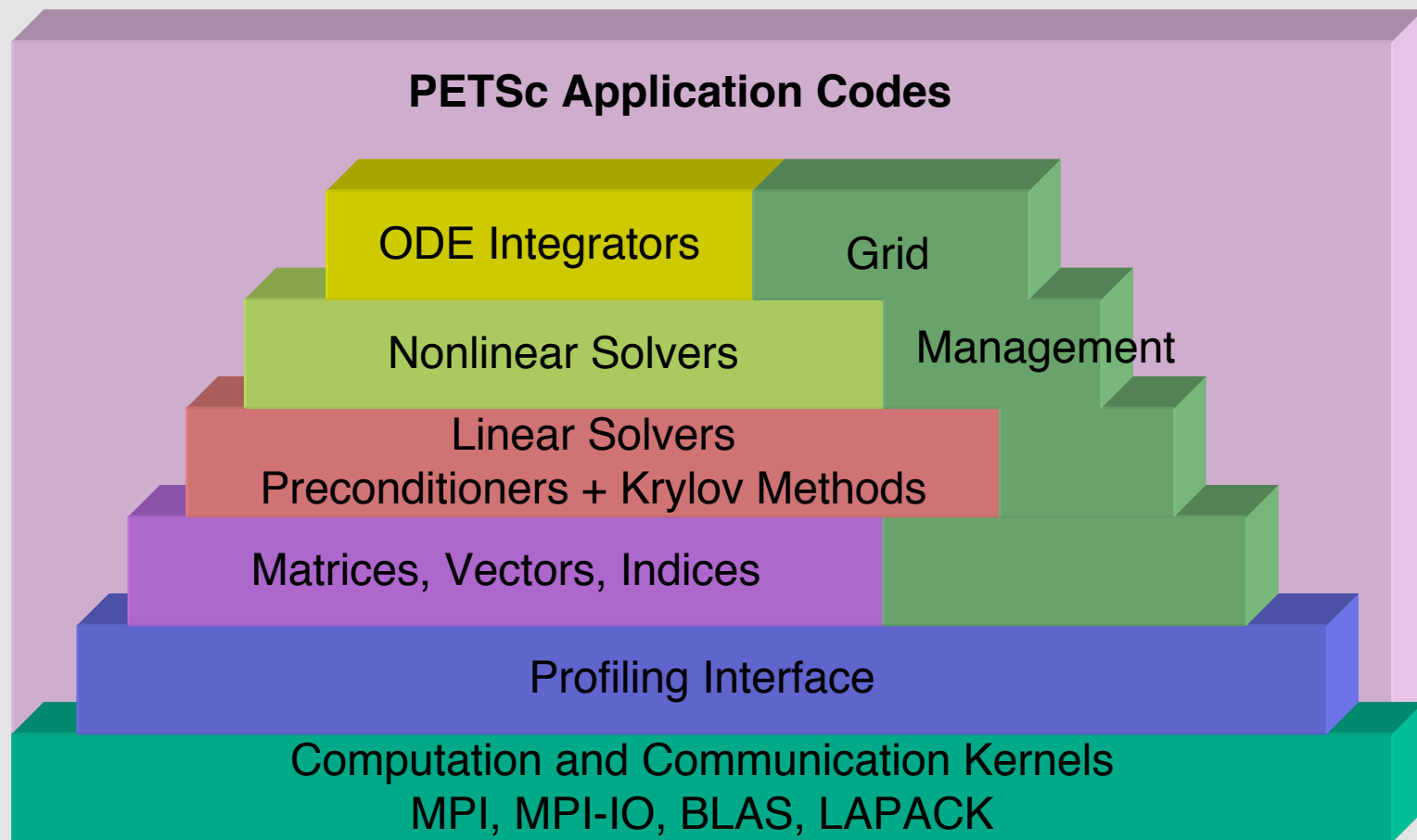
- BLOPEX (developed by Andrew Knyazev)



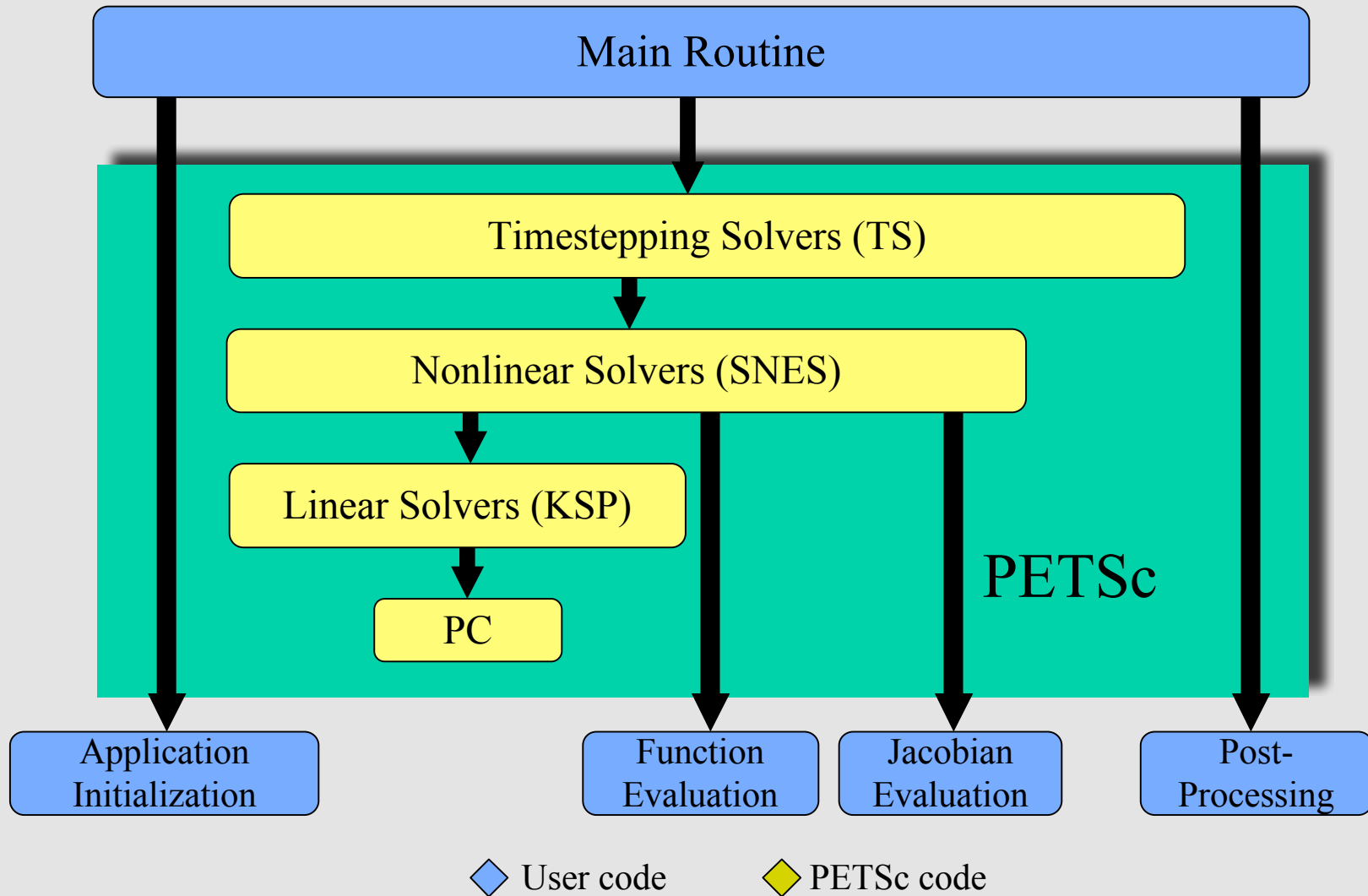
# Limit on Size?

- PETSc has done problem with **500 million unknowns**  
<http://www.scconference.org/sc2004/schedule/pdfs/pap111.pdf>
- PETSC has used a bit over **6,000 processors**  
[ftp://info.mcs.anl.gov/pub/tech\\_reports/reports/P776.ps.Z](ftp://info.mcs.anl.gov/pub/tech_reports/reports/P776.ps.Z)
- PETSC applications are closing in on the **tera-flop**

# Structure of PETSc



# Flow of Control for PDE Solution



PETSc Structure

# Levels of Abstraction in Mathematical Software

- Application-specific interface
  - Programmer manipulates objects associated with the application
- High-level mathematics interface
  - Programmer manipulates mathematical objects
    - Weak forms, boundary conditions, meshes
- Algorithmic and discrete mathematics interface
  - Programmer manipulates mathematical objects
    - Sparse matrices, nonlinear equations
  - Programmer manipulates algorithmic objects
    - Solvers
- Low-level computational kernels
  - BLAS-type operations
  - FFT



*PETSc  
emphasis*

# OO Programming & Design

**Design** based not on the **data** in object, but instead based on **operations** you perform with or on the data

**For example** a vector is not a 1d array of numbers but an abstract object where addition and scalar multiplication (and likely many more) are defined

**Added difficulty** is the **efficient** use of the computer

# The PETSc Programming Model

- **Goals**

- Portable, runs everywhere
- Performance, Performance, Performance
- Scalable parallelism

- **Approach**

- Distributed memory, “shared-nothing”
  - Requires only a compiler (single node or processor)
  - Access to data on remote machines through MPI
- Can still exploits “compiler discovered” parallelism on each node (e.g., SMP)
- Hide within objects the details of the communication
- User orchestrates communication at a higher abstract level

# Collectivity

- MPI communicators (MPI\_Comm) specify collectivity
  - Processes involved in a computation
- PETSc constructors are collective over a communicator
  - VecCreate(MPI\_Comm comm, int m, int M, Vec \*x)
  - Use **PETSC\_COMM\_WORLD** for all processes (like MPI\_COMM\_WORLD, but allows the same code to work when PETSc is started with a smaller set of processes)
- Some operations are collective, while others are not
  - collective: VecNorm()
  - not collective: VecGetLocalSize()
- If a sequence of collective routines is used, they **must** be called in the same order by each process.

# Design Principles

- Principle of Fairness
  - “If you can do it, your users will want to do it”
- Principle of Contrariness
  - “If you do it, your users will want to undo it”
- Both principles point to *symmetric* interfaces
  - Creation and query interfaces should be paired



# What is not in PETSc?

- Higher level representations of mathematical objects
- Operator overloading
- Automatic load balancing
- Sophisticated visualization capabilities
- Optimization and sensitivity, **BUT**

# Child Packages of PETSc

- **SIPs** - Shift-and-Invert Parallel Spectral Transformations
- **SLEPc** - scalable eigenvalue/eigenvector solver packages.
- **TAO** - scalable optimization algorithms

All have PETSc's style of programming

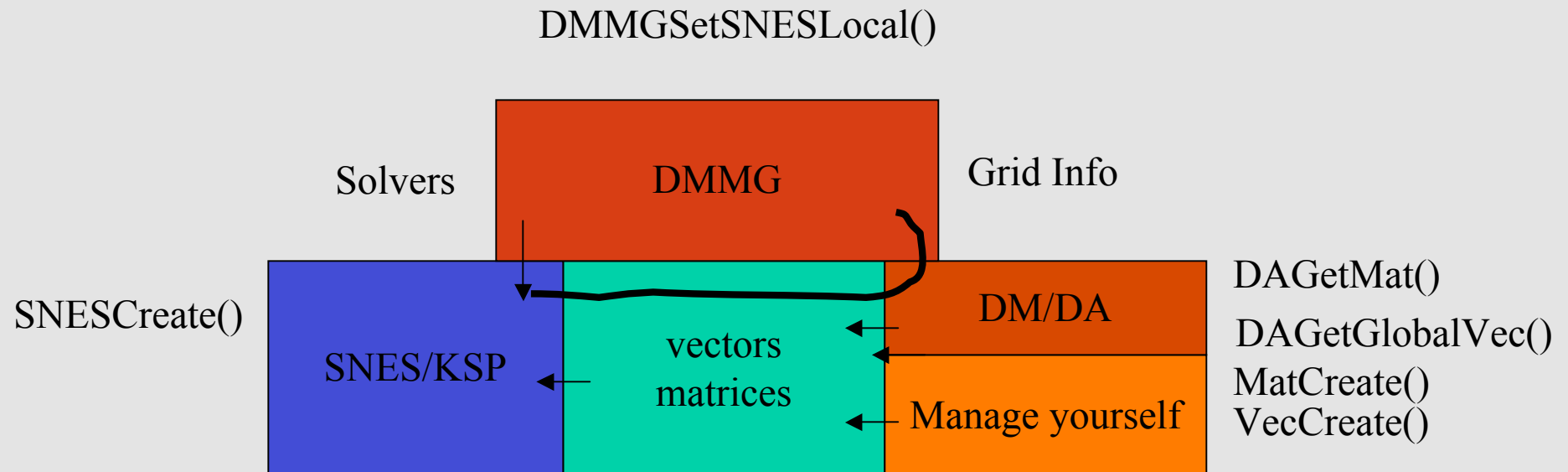
# Application Interaction

- Be willing to experiment with algorithms
  - Optimality is rarely achieved without interplay between numerical problems and algorithmics
- Adopt flexible, extensible programming
  - Algorithms and data structures not hardwired
- Be willing to play with the real code
- If possible, **profile** before seeking help
  - Automatic in PETSc

# Integration

- PETSc is **merely** a set of library interfaces
  - We do not seize main()
  - We do not control output
  - We propagate errors from underlying packages
  - We present (largely) the same interfaces in:
    - C
    - C++
    - F77
    - F90

# Three Ways of Using PETSc



# Initialization

- Call `PetscInitialize()`
  - Sets up static data and services
  - Sets up MPI if it is not already
- Call `PetscFinalize()`
  - Calculates logging summary
  - Shuts down and release resources

# Profiling

- `-log_summary` for a performance profile
  - Event timing
  - Memory usage
  - MPI messages
- Call `PetscLogStagePush/Pop()`
  - User can add new stages
- Call `PetscLogEventBegin/End()`
  - User can add new events

# Command Line Processing

- Check for an option
  - PetscOptionsHasName()
- Retrieve a value
  - PetscOptionsGetInt(), PetscOptionsGetIntArray()
- Set a value
  - PetscOptionsSetValue()
- Clear, alias, reject, etc.



# Linear Algebra I

- Vectors
  - Has a direct interface to the values
  - Supports all vector space operations
    - VecDot(), VecNorm(), VecScale()
  - Also unusual ops, e.g. VecSqrt()
  - Automatic communication during assembly
  - Customizable communication (scatters)

# PETSc Numerical Components

Nonlinear Solvers		
Newton-based Methods		Other
Line Search	Trust Region	

Time Steppers			
Euler	Backward Euler	Pseudo Time Stepping	Other

Krylov Subspace Methods							
GMRES	CG	CGS	Bi-CG-STAB	TFQMR	Richardson	Chebyshev	Other

Preconditioners						
Additive Schwartz	Block Jacobi	Jacobi	ILU	ICC	LU (Sequential only)	Others

Matrices						
Compressed Sparse Row (AIJ)	Blocked Compressed Sparse Row (BAIJ)	Block Diagonal (BDIAG)	Dense	Matrix-free	Other	

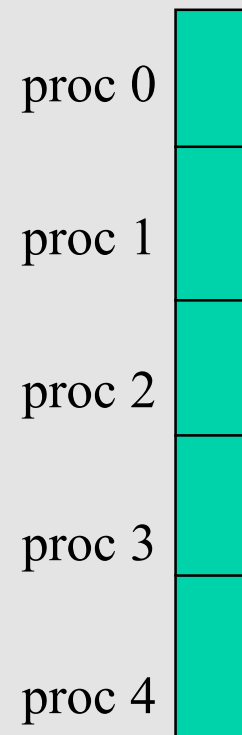
## Distributed Arrays

## Vectors

Index Sets			
Indices	Block Indices	Stride	Other

# Vectors

- What are PETSc vectors?
  - Fundamental objects for storing field solutions, right-hand sides, etc.
  - Each process locally owns a subvector of contiguously numbered global indices
- Create vectors via
  - `VecCreate(MPI_Comm, Vec *)`
    - `MPI_Comm` - processes that share the vector
  - `VecSetSizes( Vec, int, int )`
    - number of elements local to this process
    - or total number of elements
  - `VecSetType(Vec, VecType)`
    - Where `VecType` is
      - `VEC_SEQ`, `VEC_MPI`, or `VEC_SHARED`
    - `VecSetFromOptions(Vec)` lets you set the type at *runtime*



data objects:  
vectors

# Creating a vector

```
Vec x;  
int N;  
...
```

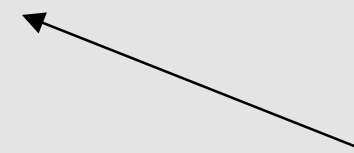
```
PetscInitialize(& argc, & argv, (char*)0, help);  
PetscOptionsGetInt(PETSC_NULL, "-n", & N, PETSC_NULL);  
...
```

```
VecCreate(PETSC_COMM_WORLD, & x);  
VecSetSizes(x, PETSC_DECIDE, N);  
VecSetType(x, VEC_MPI);  
VecSetFromOptions(x);
```

Use PETSc to get value  
from command line



Global size



PETSc  
determines local  
size

data objects:  
vectors

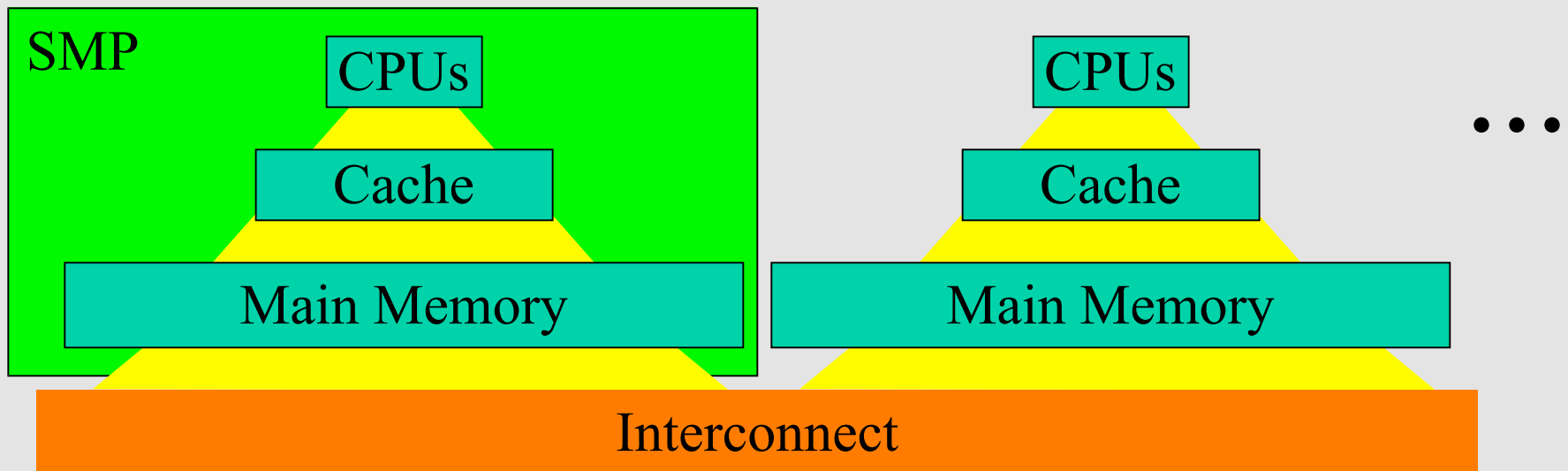
# How Can We Use a PETSc Vector

- PETSc supports “data structure-neutral” objects
  - distributed memory “shared nothing” model
  - single processors and shared memory systems
- PETSc vector is a “handle” to the real vector
  - Allows the vector to be distributed across many processes
  - To access the *elements* of the vector, we **cannot** simply do  
for (i=0; i<N; i++) v[i] = i;
  - We do not *require* that the programmer work only with the “local” part of the vector; we permit operations, such as setting an element of a vector, to be performed globally
- Recall how data is stored in the distributed memory programming model...

data objects:  
vectors

# Sample Parallel System Architecture

- Systems have an increasingly deep memory hierarchy (1, 2, 3, and more levels of cache)
- Time to reference main memory 100's of cycles
- Access to shared data requires synchronization
  - Better to ensure data is local and unshared if possible

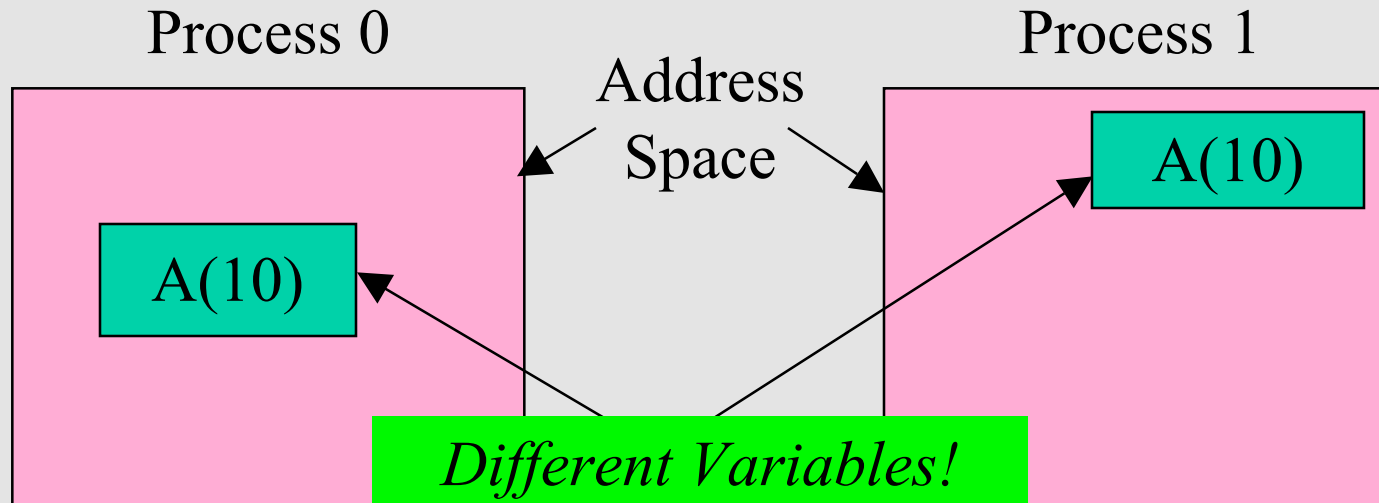


# How are Variables in Parallel Programs Interpreted?

- Single process (address space) model
  - OpenMP and threads in general
  - Fortran 90/95 and compiler-discovered parallelism
  - System manages memory and (usually) thread scheduling
  - Named variables refer to the *same* storage
- Single name space model
  - HPF
  - Global Arrays
  - Data distribution part of the language, but programs still written as if there is a single name space
- Distributed memory (shared nothing)
  - Message passing
  - Names variables in different processes are *unrelated*

data objects:  
vectors

# Distributed Memory Model



- Integer A(10)

...

**print \*, A**



This A is completely different from this one

- Integer A(10)

```
do i=1,10
```

```
  A(i) = i
```

```
enddo
```

...





# Vector Assembly

- A three step process
  - Each process tells PETSc what values to set or add to a vector component. Once *all* values provided,
  - Begin communication between processes to ensure that values end up where needed
  - (allow other operations, such as some computation, to proceed)
  - Complete the communication
- `VecSetValues(Vec,...)`
  - number of entries to insert/add
  - indices of entries
  - values to add
  - mode: [INSERT\_VALUES,ADD\_VALUES]
- `VecAssemblyBegin(Vec)`
- `VecAssemblyEnd(Vec)`

data objects:  
vectors

# Parallel Matrix and Vector Assembly

- Processes may generate any entries in vectors and matrices
- Entries need not be generated on the process on which they ultimately will be stored
- **PETSc automatically moves data during the assembly process if necessary**

data objects:  
vectors

# One Way to Set the Elements of A Vector

```
VecGetSize(x,& N); /* Global size */  
MPI_Comm_rank(PETSC_COMM_WORLD, &rank);  
if (rank == 0) {  
    for (i=0; i<N; i++)  
        VecSetValues(x, 1, &i, &i, INSERT_VALUES);  
}
```

Vector index

Vector value

/\* These two routines ensure that the data is distributed to  
the other processes \*/

```
VecAssemblyBegin(x);  
VecAssemblyEnd(x);
```

data objects:  
vectors

# A Parallel Way to Set the Elements of A Distributed Vector

```
VecGetOwnershipRange(x,&low,&high);  
for (i=low; i<high; i++)  
VecSetValues(x,1,&i,&i,INSERT_VALUES);
```

*/\* These two routines must be called (in case some other process contributed a value owned by another process) \*/*

```
VecAssemblyBegin(x);  
VecAssemblyEnd(x);
```

data objects:  
vectors

# Selected Vector Operations

Function Name	Operation
VecAXPY(Scalar *a, Vec x, Vec y)	$y = y + a*x$
VecAYPX(Scalar *a, Vec x, Vec y)	$y = x + a*y$
VecWAXPY(Scalar *a, Vec x, Vec y, Vec w)	$w = a*x + y$
VecScale(Scalar *a, Vec x)	$x = a*x$
VecCopy(Vec x, Vec y)	$y = x$
VecPointwiseMult(Vec x, Vec y, Vec w)	$w_i = x_i * y_i$
VecMax(Vec x, int *idx, double *r)	$r = \max x_i$
VecShift(Scalar *s, Vec x)	$x_i = s + x_i$
VecAbs(Vec x)	$x_i =  x_i $
VecNorm(Vec x, NormType type, double *r)	$r =   x  $

data objects:  
vectors

# A Complete PETSc Program

```
#include petscvec.h
int main(int argc,char **argv)
{
    Vec x;
    int n = 20,ierr;
    PetscTruth flg;
    PetscScalar one = 1.0, dot;

    PetscInitialize(& argc,& argv,0,0);
    PetscOptionsGetInt(PETSC_NULL,"-n",&n,PETSC_NULL);
    VecCreate(PETSC_COMM_WORLD,&x);
    VecSetSizes(x,PETSC_DECIDE,n);
    VecSetFromOptions(x);
    VecSet(& one,x);
    VecDot(x,x,& dot);
    PetscPrintf(PETSC_COMM_WORLD,"Vector length % dn",(int)dot);
    VecDestroy(x);
    PetscFinalize();
    return 0;
}
```

data objects:  
vectors

# Working With Local Vectors

- It is sometimes more efficient to directly access the storage for the local part of a PETSc Vec.
  - E.g., for finite difference computations involving elements of the vector
- PETSc allows you to access the local storage with
  - `VecGetArray(Vec, double *[])`
- You must return the array to PETSc when you finish
  - `VecRestoreArray(Vec, double *[])`
- Allows PETSc to handle data structure conversions
  - For most common uses, these routines are inexpensive and do *not* involve a copy of the vector.

data objects:  
vectors

# Example of VecGetArray

```
Vec    vec;
```

```
Double *avec;
```

```
...
```

```
VecCreate(PETSC_COMM_SELF,&vec);
```

```
VecSetSizes(vec,PETSC_DECIDE,N);
```

```
VecSetFromOptions(vec);
```

```
VecGetArray(vec,&avec);
```

```
/* compute with avec directly, e.g., */
```

```
PetscPrintf(PETSC_COMM_WORLD,
```

```
    "First element of local array of vec in each process is %f\n",  
    avec[0] );
```

```
VecRestoreArray(vec,&avec);
```

data objects: vectors
--------------------------



# Indexing

- Non-trivial in parallel
- PETSc IS object, generalization of
  - {0,3,56,9}
  - 1:4:55
  - Indexing by block

# Linear Algebra II

- Matrices
  - Must use MatSetValues()
    - Automatic communication
  - Supports many data types
    - AIJ, Block AIJ, Symmetric AIJ, Block Diagonal, etc.
  - Supports structures for many packages
    - Spooles, MUMPS, SuperLU, UMFPack, DSCPack

# Matrices

- What are PETSc matrices?
  - Fundamental objects for storing linear operators (e.g., Jacobians)
- Create matrices via
  - MatCreate(...,Mat \*)
    - MPI\_Comm - processes that share the matrix
    - number of local/global rows and columns
  - MatSetType(Mat,MatType)
    - where MatType is one of
      - default sparse AIJ: MPIAIJ, SEQAIJ
      - block sparse AIJ (for multi-component PDEs): MPIAIJ, SEQAIJ
      - symmetric block sparse AIJ: MPISBAIJ, SAEQSBAIJ
      - block diagonal: MPIBDIAG, SEQBDIAG
      - dense: MPIDENSE, SEQDENSE
      - matrix-free
      - etc.
    - MatSetFromOptions(Mat) lets you set the MatType at *runtime*.

data objects:  
matrices

# Matrices and Polymorphism

- Single user interface, e.g.,
  - Matrix assembly
    - MatSetValues()
  - Matrix-vector multiplication
    - MatMult()
  - Matrix viewing
    - MatView()
- Multiple underlying implementations
  - AIJ, block AIJ, symmetric block AIJ, block diagonal, dense, matrix-free, etc.
- A matrix is defined by its *interface*, the operations that you can perform with it.
  - Not by its data structure

data objects:  
matrices

# Matrix Assembly

- Same form as for PETSc Vectors:
- **MatSetValues(Mat,...)**
  - number of rows to insert/add
  - indices of rows and columns
  - number of columns to insert/add
  - values to add
  - mode:  
  [INSERT\_VALUES,ADD\_VALUES]
- **MatAssemblyBegin(Mat)**
- **MatAssemblyEnd(Mat)**

data objects:  
matrices

# Matrix Assembly Example

simple 3-point stencil for 1D discretization

```
Mat A;
int column[3], i;
double value[3];
...
MatCreate(PETSC_COMM_WORLD,&A);
MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,N,N);
MatSetFromOptions(A);
/* mesh interior */
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
if (rank == 0) { /* Only one process creates matrix entries */
  for (i=1; i<N-2; i++) {
    column[0] = i-1; column[1] = i; column[2] = i+1;
    MatSetValues(A,1,&i,3,column,value,INSERT_VALUES);
  }
}
/* also must set boundary points (code for global row 0 and N-1 omitted) */
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

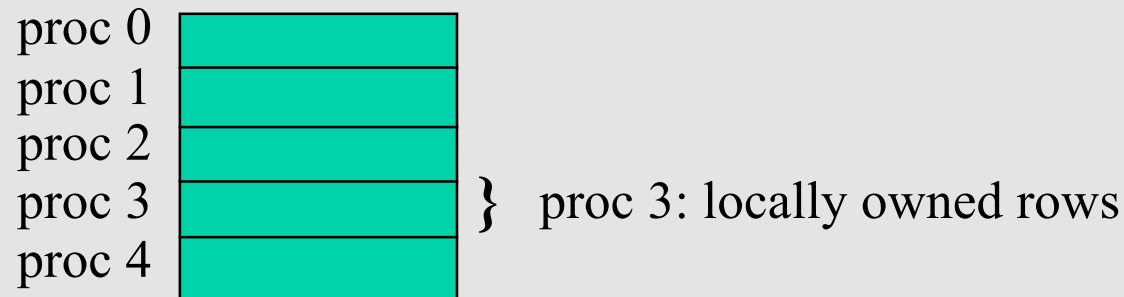
Choose the global size of the matrix

Let PETSc decide how to allocate matrix across processes

data objects:  
matrices

# Parallel Matrix Distribution

Each process locally owns a submatrix of contiguously numbered global rows.



`MatGetOwnershipRange(Mat A, int *rstart, int *rend)`

- `rstart`: first locally owned row of global matrix
- `rend - 1`: last locally owned row of global matrix

data objects:  
matrices

# Matrix Assembly Example With Parallel Assembly

simple 3-point stencil for 1D discretization

```
Mat A;
int column[3], i, start, end, istart, iend;
double value[3];
...
MatCreate(PETSC_COMM_WORLD, &A);
MatSetSizes(A, PETSC_DECIDE, PETSC_DECIDE, N, N);
MatSetFromOptions(A);
MatGetOwnershipRange(A, &start, &end);
/* mesh interior */
istart = start; if (start == 0) istart = 1;
iend = end; if (iend == n-1) iend = n-2;
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
for (i=istart; i<iend; i++) { /* each processor generates some of the matrix values */
    column[0] = i-1; column[1] = i; column[2] = i+1;
    MatSetValues(A, 1, &i, 3, column, value, INSERT_VALUES);
}
/* also must set boundary points (code for global row 0 and n-1 omitted) */
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

data objects:  
matrices



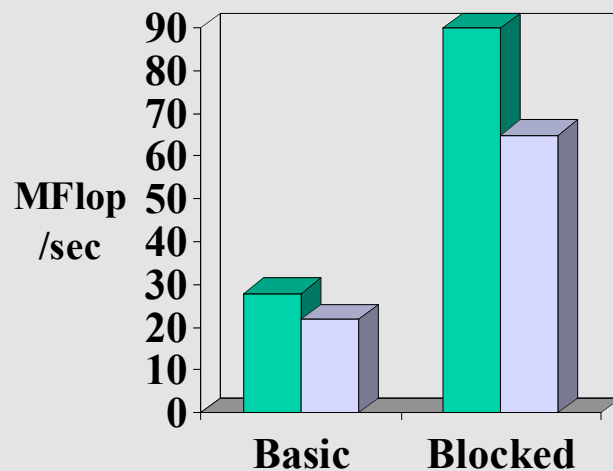
# Why Are PETSc Matrices The Way They Are?

- No one data structure is appropriate for all problems
  - Blocked and diagonal formats provide significant performance benefits
  - PETSc provides a large selection of formats and makes it (relatively) easy to extend PETSc by adding new data structures
- Matrix assembly is difficult enough without being forced to worry about data partitioning
  - PETSc provides parallel assembly routines
  - Achieving high performance still requires making most operations local to a process but programs can be incrementally developed.
- Matrix decomposition by consecutive rows across processes, **for sparse matrices**, is simple and makes it easier to work with other codes.
  - For applications with other ordering needs, PETSc provides “Application Orderings” (AO), described later.

data objects:  
matrices

# Blocking: Performance Benefits

More issues discussed in full tutorials available via PETSc web site.

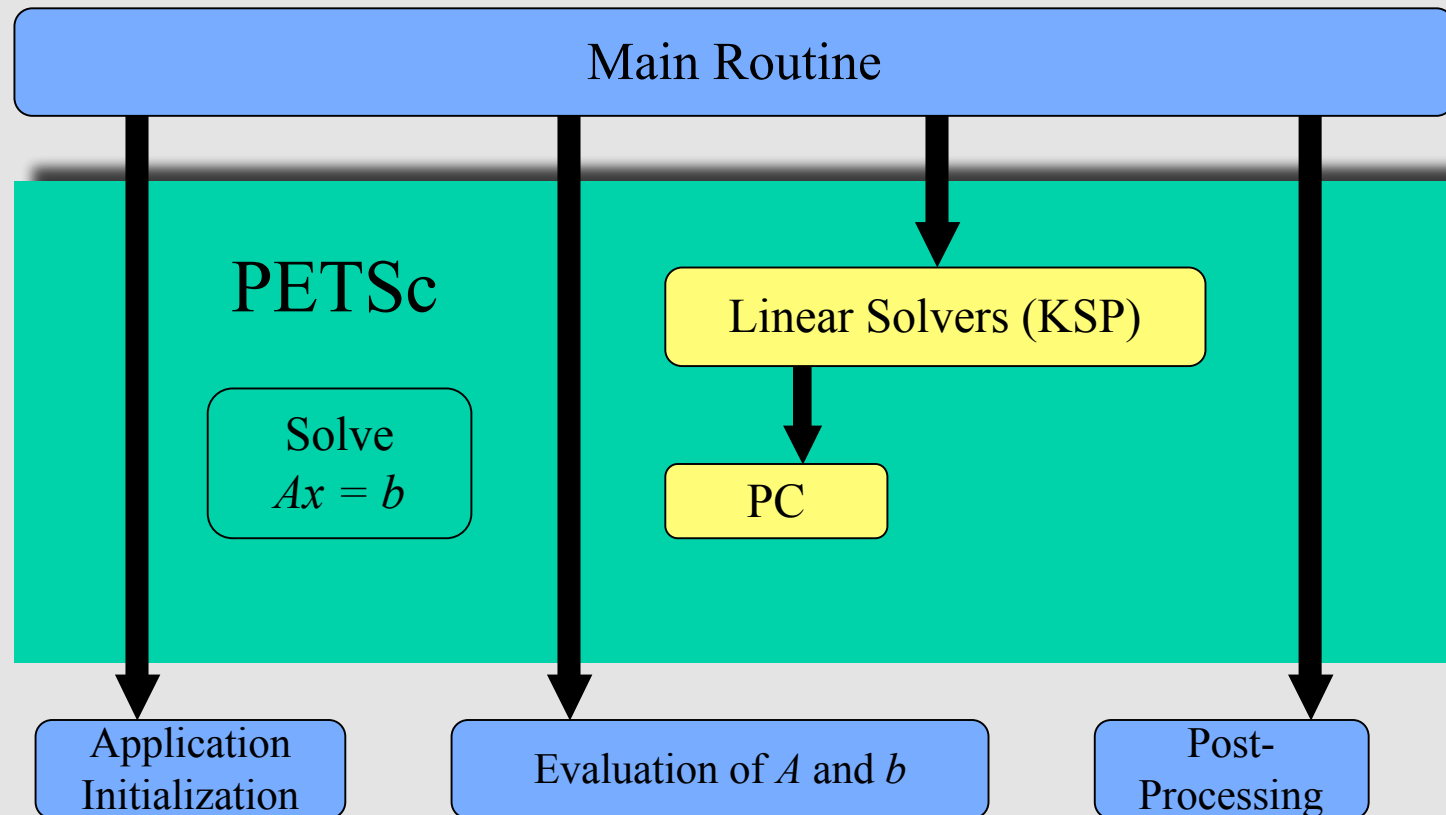


■ Matrix-vector products  
■ Triangular solves

- 3D compressible Euler code
- Block size 5
- IBM Power2

data objects:  
matrices

# Linear Solution



◆ User code

◆ PETSc code

beginner

solvers:  
linear

# Context Variables

- Are the key to solver organization
- Contain the complete state of an algorithm, including
  - parameters (e.g., convergence tolerance)
  - functions that run the algorithm (e.g., convergence monitoring routine)
  - information about the current state (e.g., iteration number)

beginner

solvers:  
linear

# Creating the KSP Context

- C/C++ version  
`ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);`
- Fortran version  
`call KSPCreate(PETSC_COMM_WORLD,ksp,ierr)`
- Provides an **identical** user interface for all linear solvers
  - uniprocess and parallel
  - real and complex numbers

beginner

solvers:  
linear

# KSP Structure

- Each KSP object actually contains two parts:
  - Krylov Space Method
    - The iterative method
    - The context contains information on method parameters (e.g., GMRES search directions), work spaces, etc
  - PC — Preconditioners
    - Knows how to apply a preconditioner
    - The context contains information on the preconditioner, such as what routine to call to apply it

# Linear Solvers in PETSc

## Krylov Methods (KSP)

- Conjugate Gradient
- GMRES
- CG-Squared
- Bi-CG-stab
- Transpose-free QMR
- etc.

## Preconditioners (PC)

- Block Jacobi
- Overlapping Additive Schwarz
- ICC, ILU via BlockSolve95
- ILU(k), LU (direct solve, sequential only)
- Arbitrary matrix
- etc.

beginner

solvers:  
linear

# Basic Linear Solver Code (C/C++)

```
KSP ksp;          /* linear solver context */
Mat  A;           /* matrix */
Vec  x, b;        /* solution, RHS vectors */
int  n, its;      /* problem dimension, number of iterations */
```

```
MatCreate(PETSC_COMM_WORLD,PETSC_DECIDE,PETSC_DECIDE,n,n,&A);
```

```
MatSetFromOptions(A);
```

```
/* (code to assemble matrix not shown) */
```

```
VecCreate(PETSC_COMM_WORLD,&x);
```

```
VecSetSizes(x,PETSC_DECIDE, n);
```

```
VecSetFromOptions(x);
```

```
VecDuplicate(x,&b);
```

```
/* (code to assemble RHS vector not shown)*/
```

```
KSPCreate(PETSC_COMM_WORLD,&ksp);
```

```
KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN);
```

```
KSPSetFromOptions(ksp);
```

```
KSPSolve(ksp,b,x,&its);
```

```
KSPDestroy(ksp);
```

Indicate whether the preconditioner has the same nonzero pattern as the matrix *each time a system is solved*. This default works with *all* preconditioners. Other values (e.g., SAME\_NONZERO\_PATTERN) can be used for particular preconditioners. Ignored when solving only one system

beginner

solvers:  
linear



# Basic Linear Solver Code (Fortran)

```
KPS    ksp
Mat     A
Vec     x, b
integer n, its, ierr
```

```
call MatCreate( PETSC_COMM_WORLD,A,ierr )
call MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,n,n,ierr)
call MatSetFromOptions( A, ierr )
call VecCreate( PETSC_COMM_WORLD,x,ierr )
call VecSetSizes( x, PETSC_DECIDE, n, ierr )
call VecSetFromOptions( x, ierr )
call VecDuplicate( x,b,ierr )
```

C then assemble matrix and right-hand-side vector

```
call KSPCreate(PETSC_COMM_WORLD,ksp,ierr)
call KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN,ierr)
call KSPSetFromOptions(ksp,ierr)
call KSPSolve(ksp,b,x,its,ierr)
KSPDestroy(ksp,ierr)
```

beginner

solvers:  
linear

# Customization Options

- Command Line Interface
  - Applies same rule to all queries via a database
  - Enables the user to have complete control at runtime, with no extra coding
- Procedural Interface
  - Provides a great deal of control on a usage-by-usage basis inside a single code
  - Gives full flexibility inside an application

beginner

solvers:  
linear

# Setting Solver Options at Runtime

- -ksp\_type [cg, gmres, bcgs, tfqmr, ...]
- -pc\_type [lu, ilu, jacobi, sor, asm, ...]

1

- -ksp\_max\_it <max\_iters>
- -ksp\_gmres\_restart <restart>
- -pc\_asm\_overlap <overlap>
- -pc\_asm\_type [basic, restrict, interpolate, none]
- etc ...

2

1


2


beginner

intermediate

solvers:  
linear

# Linear Solvers: Monitoring Convergence

- `-ksp_monitor` - Prints preconditioned residual norm 
- `-ksp_xmonitor` - Plots preconditioned residual norm

- `-ksp_truemonitor` - Prints true residual norm  $\|b - Ax\|$  
- `-ksp_xtruemonitor` - Plots true residual norm  $\|b - Ax\|$

- User-defined monitors, using callbacks 

 1

 2

 3

beginner

intermediate

advanced

solvers:  
linear

# Setting Solver Options within Code

- `KSPSetType(KSP ksp,KSPTType type)`
- `KSPSetTolerances(KSP ksp,PetscReal rtol, PetscReal atol,PetscReal dtol, int maxits)`
- etc....
- `KSPGetPC(KSP ksp,PC *pc)`
  - `PCSetType(PC pc,PCType)`
  - `PCASMSetOverlap(PC pc,int overlap)`
  - etc....

# Recursion: Specifying Solvers for Schwarz Preconditioner Blocks

- Specify KSP solvers and options with “-sub” prefix, e.g.,
  - Full or incomplete factorization
    - sub\_pc\_type lu
    - sub\_pc\_type ilu -sub\_pc\_ilu\_levels <levels>
  - Can also use inner Krylov iterations, e.g.,
    - sub\_ksp\_type gmres -sub\_ksp\_rtol <rtol>
    - sub\_ksp\_max\_it <maxit>

# KSP: Review of Basic Usage

- `KSPCreate( )` - Create solver context
- `KSPSetOperators( )` - Set linear operators
- `KSPSetFromOptions( )` - Set runtime solver options for `[KSP,PC]`
- `KSPSolve( )` - Run linear solver
- `KSPView( )` - View solver options actually used at runtime (alternative: `-ksp_view`)
- `KSPDestroy( )` - Destroy solver

beginner

solvers:  
linear

# KSP: Review of Selected Preconditioner Options

Functionality	Procedural Interface	Runtime Option	
Set preconditioner type	PCSetType( )	-pc_type [lu,ilu,jacobi,sor,asm,...]	1
Set level of fill for ILU	PCILUSetLevels( )	-pc_ilu_levels <levels>	2
Set SOR iterations	PCSORSetIterations( )	-pc_sor_its <its>	
Set SOR parameter	PCSORSetOmega( )	-pc_sor_omega <omega>	
Set additive Schwarz variant	PCASMSetType( )	-pc_asm_type [basic, restrict,interpolate,none]	
Set subdomain solver options	PCGetSubKSP( )	-sub_pc_type <pctype> -sub_ksp_type <ksptype> -sub_ksp_rtol <rtol>	

1

2

*And many more options...*

beginner

intermediate

solvers: linear:  
preconditioner  
s



# Review of Selected Krylov Method Options

Functionality	Procedural Interface	Runtime Option
Set Krylov method	KSPSetType( )	-ksp_type [cg,gmres,bcgs,tfqmr,cgs,...]
Set monitoring routine	KSPSetMonitor( )	-ksp_monitor, -ksp_xmonitor, -ksp_truemonitor, -ksp_xtruemonitor
Set convergence tolerances	KSPSetTolerances( )	-ksp_rtol <rt> -ksp_atol <at> -ksp_max_its <its>
Set GMRES restart parameter	KSPGMRESRestart( )	-ksp_gmres_restart <restart>
Set orthogonalization routine for GMRES	KSPGMRESOrthogonalization( )	-ksp_unmodifiedgramschmidt -ksp_iorthog

*And many more options...*

1

2

beginner

intermediate

solvers: linear:  
Krylov methods

# Why Polymorphism?

- Programs become independent of the choice of algorithm
- Consider the question:
  - What is the best combination of iterative method and preconditioner for my problem?
- How can you answer this experimentally?
  - Old way:
    - Edit code. Make. Run. Edit code. Make. Run. Debug. Edit. ...
  - New way:....

# KSP: Runtime Script Example

```
emacs@lava.mcs.anl.gov
Buffers Files Tools Edit Search Insert Help
#!/bin/csh
#
# Sample script: Experimenting with linear solver options.
# Can be used with, e.g., petsc/src/sles/examples/tutorials/ex2.c
#
foreach np (1 2 4 8)                # number of processors
  foreach ksptype (gmres bcgs tfqmr) # Krylov solver
    foreach pctype (bjacobi asm)    # preconditioner
      foreach subpctype (jacobi sor ilu) # subdomain solver
        if ($subpctype == ilu) then
          foreach level (0 1 2)      # level of fill for ILU(k)
            echo '***** Beginning new run *****'
            mpirun -np $np ex2 -pc_type $pctype -ksp_type $ksptype \
              -sub_ksp_type preonly sub_pc_type $subpctype \
              -sub_pc_ilu_levels $level \
              -ksp_monitor -sles_view -optionsleft
          else
            echo '***** Beginning new run *****'
            mpirun -np $np ex2 -pc_type $pctype -ksp_type $ksptype \
              -sub_ksp_type preonly sub_pc_type $subpctype \
              -ksp_monitor -sles_view -optionsleft
          endif
        end
      end
    end
  end
end
-----Emacs: script1 (Shell-script)--L1--Top-----
```

intermediate

solvers:  
linear

# Viewing KSP Runtime Options

```
emacs@lava.mcs.anl.gov
Buffers Files Tools Edit Search Help
[[lava] ex2 -ksp_monitor -pc_ilu_levels 1 -sles_view > out.5
0 KSP Residual norm 5.394188560416e+00
1 KSP Residual norm 1.238309089931e+00
2 KSP Residual norm 1.104133215450e-01
3 KSP Residual norm 6.609740098311e-03
4 KSP Residual norm 2.732911209560e-04
KSP Object:
  method: gmres
    GMRES: restart=30, using Modified Gram-Schmidt Orthogonalization
    maximum iterations=10000, initial guess is zero
    tolerances: relative=0.000138889, absolute=1e-50, divergence=10000
    left preconditioning
PC Object:
  method: ilu
    ILU: 1 level of fill
        out-of-place factorization
        matrix ordering: natural
    linear system matrix = precond matrix:
Matrix Object:
  type=MATSEQAIJ, rows=56, cols=56
  total: nonzeros=250, allocated nonzeros=560
Norm of error 0.000280658 iterations 4
-----Emacs: out.5 (Nroff) --L1--All-----
```

intermediate

solvers:  
linear

# Providing Different Matrices to Define Linear System and Preconditioner

Solve  $Ax=b$

Precondition via:  $M_L^{-1} A M_R^{-1} (M_R x) = M_L^{-1} b$

- Krylov method: Use  $A$  for matrix-vector products
- Build preconditioner using either
  - $A$  - matrix that defines linear system
  - or  $P$  - a different matrix (cheaper to assemble)
- `KSPSetOperators(KSP ksp,`
  - Mat  $A$ ,
  - Mat  $P$ ,
  - MatStructure flag)

advanced

solvers:  
linear

# Matrix-Free Solvers

- Use “shell” matrix data structure
  - `MatCreateShell(..., Mat *mfctx)`
- Define operations for use by Krylov methods
  - `MatShellSetOperation(Mat mfctx,`
    - `MatOperation MATOP_MULT,`
    - `(void *) int (UserMult)(Mat,Vec,Vec)`
- Names of matrix operations defined in `petsc/include/mat.h`
- Some defaults provided for nonlinear solver usage

advanced

solvers:  
linear

# PETSc Programming Aids

- Correctness Debugging
  - Automatic generation of tracebacks
  - Detecting memory corruption and leaks
  - Optional user-defined error handlers
- Performance Debugging
  - Integrated profiling using **-log\_summary**
  - Profiling by stages of an application
  - User-defined events

# Debugging

## Support for parallel debugging

- `-start_in_debugger` [gdb,dbx,noxterm]
- `-on_error_attach_debugger` [gdb,dbx,noxterm]
- `-on_error_abort`
- `-debugger_nodes` 0,1
- `-display machinename:0.0`

When debugging, it is often useful to place a breakpoint in the function `PetscError( )`.



# Sample Error Traceback

Breakdown in ILU factorization due to a zero pivot

```
xterm
Buffers Files Tools Edit Search Mule Help
[dreamcast] mpirun -np 1 ex1
-----
PETSc Version 2.1.0, Released April 11, 2001
  The PETSc Team   petsc-maint@mcs.anl.gov
  http://www.mcs.anl.gov/petsc/

See docs/copyright.html for copyright information.
See docs/changes.html for recent updates.
See docs/troubleshooting.html for hints about trouble shooting.
See docs/manualpages/index.html for manual pages.
-----
ex1 on a linux named dreamcast.mcs.anl.gov by balay Thu Oct  4 15:25:11 2001
Libraries linked from /home/balay/software/petsc-2.1.0/lib/libg/linux
-----
[0]PETSC ERROR: MatLUFactorNumeric_SeqAIJ() line 508 in src/mat/impls/aij/seq/aijfact.c
[0]PETSC ERROR:   Detected zero pivot in LU factorization!
[0]PETSC ERROR:   Zero pivot row 0!
[0]PETSC ERROR: MatLUFactorNumeric() line 1575 in src/mat/interface/matrix.c
[0]PETSC ERROR: PCSetUp_ILU() line 646 in src/sles/pc/impls/ilu/ilu.c
[0]PETSC ERROR: PCSetUp() line 783 in src/sles/pc/interface/precon.c
[0]PETSC ERROR: SLESSetUp() line 382 in src/sles/interface/sles.c
[0]PETSC ERROR: SLESSolve() line 483 in src/sles/interface/sles.c
[0]PETSC ERROR: main() line 195 in test/ex1.c
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_5469: p4_error: : 71
--1-:---F1 logs (Text)--L3-- 2%-----
```

debugging and errors

# Sample Memory Corruption Error

```
xterm
Buffers Files Tools Edit Search Mule Help
[dreamcast] mpirun -np 1 ex2 -trmalloc_off
[dreamcast] mpirun -np 1 ex2 -trmalloc

-----
PETSc Version 2.1.0, Released April 11, 2001
  The PETSc Team   petsc-maint@mcs.anl.gov
  http://www.mcs.anl.gov/petsc/

See docs/copyright.html for copyright information.
See docs/changes.html for recent updates.
See docs/troubleshooting.html for hints about trouble shooting.
See docs/manualpages/index.html for manual pages.

-----
ex2 on a linux named dreamcast.mcs.anl.gov by balay Thu Oct  4 15:35:29 2001
Libraries linked from /home/balay/software/petsc-2.1.0/lib/libg/linux

-----
PetscTrFreeDefault called from main() line 51 in test/ex2.c
Block [id=0(14)] at address 0x81152d8 is corrupted (probably write past end)
Block allocated in main() line 49 in test/ex2.c
[0]PETSC ERROR: PetscTrFreeDefault() line 363 in src/sys/src/memory/mtr.c
[0]PETSC ERROR:   Memory corruption!
[0]PETSC ERROR:   Corrupted memory!
[0]PETSC ERROR: main() line 51 in test/ex2.c
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_5691: p4_error: : 78
--1:--F1 logs (Text)--L32--27%
```

debugging and errors

# Sample Out-of-Memory Error

```
xterm
Buffers Files Tools Edit Search Mule Help
[dreamcast] mpirun -np 1 ex3
-----
PETSc Version 2.1.0, Released April 11, 2001
  The PETSc Team   petsc-maint@mcs.anl.gov
  http://www.mcs.anl.gov/petsc/

See docs/copyright.html for copyright information.
See docs/changes.html for recent updates.
See docs/troubleshooting.html for hints about trouble shooting.
See docs/manualpages/index.html for manual pages.
-----
ex3 on a linux named dreamcast.mcs.anl.gov by balay Thu Oct  4 15:51:46 2001
Libraries linked from /home/balay/software/petsc-2.1.0/lib/libg/linux
-----
[0]PETSC ERROR: PetscMallocAlign() line 59 in src/sys/src/memory/mal.c
[0]PETSC ERROR:   Out of memory. This could be due to allocating
[0]PETSC ERROR:   too large an object or bleeding by not properly
[0]PETSC ERROR:   destroying unneeded objects.
[0]PETSC ERROR:   Memory allocated -2044966576 Memory used by process 0
[0]PETSC ERROR:   Try running with -trdump or -trmalloc_log for info.
[0]PETSC ERROR:   Memory requested 500000296!
[0]PETSC ERROR: main() line 51 in test/ex3.c
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_6291: p4_error: : 55
--1:--F1 logs (Text)--L60--49%-----
```

debugging and errors

# Sample Floating Point Error

```
xterm
Buffers Files Tools Edit Search Mule Help
[maple] mpirun -np 1 ex4 -fp_trap
-----
ex4 on a solaris named maple.mcs.anl.gov by balay Thu Oct  4 16:08:19 2001
Libraries linked from /homes/balay/spetsc/lib/libg/solaris
-----
----- Stack Frames -----
Note: The EXACT line numbers in the stack are not available,
      INSTEAD the line number of the start of the function
      is given.
[0] CreateError line 12 tests/ex4.c
-----
[0]PETSC ERROR: unknownfunction() line 0 in Unknown directoryUnknown file
[0]PETSC ERROR:   Signal received!
[0]PETSC ERROR:   Caught signal FPE:
PETSC ERROR: Floating Point Exception,probably divide by zero
PETSC ERROR: Try option -start_in_debugger or -on_error_attach_debugger to
PETSC ERROR: determine where problem occurs
PETSC ERROR: likely location of problem given above in stack
!
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_20924: p4_error: : 59
-----
--1:--F1 logs (Text)--L88--Bot-----
```

debugging and errors

# Profiling and Performance Tuning

## **Profiling:**

- Integrated profiling using `-log_summary`
- User-defined events
- Profiling by stages of an application

## **Performance Tuning:**

- Matrix optimizations
- Application optimizations
- Algorithmic tuning

profiling and  
performance tuning

# Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- Active if PETSc was compiled with `-DPETSC_LOG` (default)
  - Can also profile application code segments
- Print summary data with option: `-log_summary`
- Print redundant information from PETSc routines: `-log_info`
- Print the trace of the functions called: `-log_trace`

profiling and  
performance tuning

# Sample -log\_summary

Event	Count		Time (sec)		Flops/sec		Mess	Avg len	Reduct	--- Global ---					--- Stage ---					Total Mflop/s
	Max	Ratio	Max	Ratio	Max	Ratio				%T	%F	%M	%L	%R	%T	%F	%M	%L	%R	
--- Event Stage 0: Main Stage																				
PetscBarrier	2	1.0	1.1733e-05	1.0	0.00e+00	0.0	0.0e+00	0.0e+00	0.0e+00	0	0	0	0	0	0	0	0	0	0	
--- Event Stage 1: SetUp																				
VecSet	2	1.0	9.3448e-04	1.0	0.00e+00	0.0	0.0e+00	0.0e+00	0.0e+00	0	0	0	0	0	0	0	0	0	0	
MatMultTranspose	1	1.0	1.8022e-03	1.0	1.85e+08	1.0	0.0e+00	0.0e+00	0.0e+00	0	0	0	0	0	0	57	0	0	0	185
MatAssemblyBegin	3	1.0	1.0057e-05	1.0	0.00e+00	0.0	0.0e+00	0.0e+00	0.0e+00	0	0	0	0	0	0	0	0	0	0	0
MatAssemblyEnd	3	1.0	2.0356e-02	1.0	0.00e+00	0.0	0.0e+00	0.0e+00	0.0e+00	0	0	0	0	0	5	0	0	0	0	0
MatFDColorCreate	2	1.0	1.5341e-01	1.0	0.00e+00	0.0	0.0e+00	0.0e+00	4.6e+01	1	0	0	0	16	36	0	0	0	74	0
--- Event Stage 2: Solve																				
VecDot	2	1.0	3.2985e-03	1.0	9.56e+07	1.0	0.0e+00	0.0e+00	2.0e+00	0	0	0	0	1	0	0	0	0	2	96
VecMDot	45	1.0	9.3093e-02	1.0	1.59e+08	1.0	0.0e+00	0.0e+00	1.5e+01	0	0	0	0	5	1	1	0	0	19	159
VecNorm	112	1.0	2.0851e-01	1.0	8.47e+07	1.0	0.0e+00	0.0e+00	5.2e+01	1	1	0	0	18	2	1	0	0	64	85
MatMultTranspose	1	1.0	1.8022e-03	1.0	1.85e+08	...														
VecNorm	112	1.0	2.0851e-01	1.0	8.47e+07	...	5.2e+01	...												

# More -log\_summary

Memory usage is given in bytes:

Object Type	Creations	Destructions	Memory	Descendants' Mem.
-------------	-----------	--------------	--------	-------------------

--- Event Stage 0: Main Stage

--- Event Stage 1: SetUp

Distributed array	4	0	0	2.37475e+06
Index Set	104	24	2376480	0
Map	40	10	2000	0
Vec	36	10	2846384	0
Vec Scatter	12	0	0	0
IS Local to global mapping	8	0	0	0
Matrix	8	0	0	0
Matrix FD Coloring	4	0	0	0
SNES	4	0	0	0
Krylov Solver	10	0	0	0
Preconditioner	10	0	0	0

--- Event Stage 2: Solve

Distributed array	0	4	822496	3.16488e+06
Index Set	20	100	3578544	0
Map	26	56	11200	0
Vec	160	186	92490656	2864
Vec Scatter	0	12	2374784	0



# Still more –log\_summary

```
=====
Average time to get PetscTime(): 1.13389e-08
Compiled without FORTRAN kernels
Compiled with double precision matrices (default)
sizeof(short) 2 sizeof(int) 4 sizeof(long) 4 sizeof(void*) 4
Libraries compiled on Fri May 28 01:39:58 PDT 2004 on MBuschel
Machine characteristics: CYGWIN_NT-5.1 MBuschel 1.5.9(0.112/4/2) 2004-03-18 23:05
Using PETSc directory: /home/Kris/petsc/petsc-dev
Using PETSc arch: cygwin
-----
Using C compiler: gcc -Wall -O -fomit-frame-pointer -Wno-strict-aliasing -I/home/K
c-dev/bmake/cygwin -I/home/Kris/petsc/petsc-dev/include -I/software/MPI/mpich-nt

EXTERN_CXX -D__SDIR__='.' '
C Compiler version:
gcc (GCC) 3.3.1 (cygming special)\nCopyright (C) 2003 Free Software Foundation,
```

# Adding a new Stage

```
int stageNum;  
PetscLogStageRegister (&stageNum, "name");
```

```
PetscLogStagePush (stageNum);  
[code to monitor]  
PetscLogStagePop ();
```

profiling and  
performance tuning

# Adding a new Event

```
static int USER_EVENT;  
PetscLogEventRegister(&USER_EVENT, "name", CLASS_COOKIE);  
  
PetscLogEventBegin(USER_EVENT, 0, 0, 0, 0);  
[code to monitor]  
PetscLogFlops(user_evnet_flops);  
PetscLogEventEnd(USER_EVENT, 0, 0, 0, 0);
```

profiling and  
performance tuning

# Adding a new Class

- Cookie identifies a class uniquely

```
static int CLASS_COOKIE;
```

```
PetscLogClassRegister(&CLASS_COOKIE, "name");
```

- This initialization must happen before any objects of this type are created

# Performance Requires Managing Memory

- Real systems have many levels of memory
  - Programming models try to hide memory hierarchy
    - Except C—register
- Simplest model: Two levels of memory
  - Divide at largest (relative) latency gap
  - Processes have their own memory
    - Managing a processes memory is known (if unsolved) problem
  - Exactly matches the distributed memory model

profiling and  
performance tuning

# Sparse Matrix-Vector Product

- Common operation for optimal (in floating-point operations) solution of linear systems

- Sample code:

```
for row=0,n-1
    m    = i[row+1] - i[row];
    sum  = 0;
    for k=0,m-1
        sum += *a++ * x[*j++];
    y[row] = sum;
```

- Data structures are  $a[nnz]$ ,  $j[nnz]$ ,  $i[n]$ ,  $x[n]$ ,  $y[n]$

profiling and  
performance tuning

# Simple Performance Analysis

- Memory motion:
  - $\text{nnz} (\text{sizeof}(\text{double}) + \text{sizeof}(\text{int})) + n (2 * \text{sizeof}(\text{double}) + \text{sizeof}(\text{int}))$
  - Perfect cache (never load same data twice)
- Computation
  - nnz multiply-add (MA)
- Roughly 12 bytes per MA
- Typical WS node can move  $\frac{1}{2}$ -4 bytes/MA
  - *Maximum* performance is 4-33% of peak

# More Performance Analysis

- Instruction Counts:
  - $\text{nnz} (2 * \text{load-double} + \text{load-int} + \text{mult-add}) + n (\text{load-int} + \text{store-double})$
- Roughly 4 instructions per MA
- Maximum performance is 25% of peak (33% if MA overlaps one load/store)
- Changing matrix data structure (e.g., exploit small block structure) allows reuse of data in register, eliminating some loads (x and j)
- Implementation improvements (tricks) cannot improve on these limits

profiling and  
performance tuning



# Alternative Building Blocks

- Performance of sparse matrix - multi-vector multiply:

<i>Format</i>	<i>Number of Vectors</i>	<i>Mflops</i>	
		<i>Ideal</i>	<i>Achieved</i>
<b>AIJ</b>	1	49	45
<b>AIJ</b>	4	182	120
<b>BAIJ</b>	1	64	55
<b>BAIJ</b>	4	236	175

- Results from 250 MHz R10000 (500 MF/sec peak)
- BAIJ is a block AIJ with blocksize of 4
- Multiple right-hand sides can be solved in nearly the same time as a single RHS

profiling and  
performance tuning

# Matrix Memory Pre-allocation

- PETSc sparse matrices are dynamic data structures. Can add additional nonzeros freely
- Dynamically adding many nonzeros
  - requires additional memory allocations
  - requires copies
  - can kill performance
- Memory pre-allocation provides the freedom of dynamic data structures plus good performance

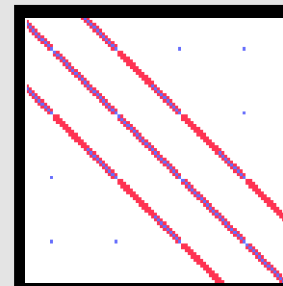
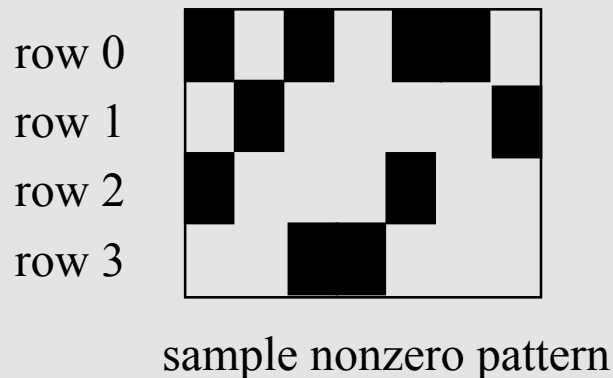
profiling and  
performance tuning

# Indicating Expected Nonzeros

## Sequential Sparse Matrices

`MatCreateSeqAIJ(..., int *nnz, Mat *A)`

- `nnz[0]` - expected number of nonzeros in row 0
- `nnz[1]` - expected number of nonzeros in row 1



another sample  
nonzero pattern

profiling and  
performance tuning

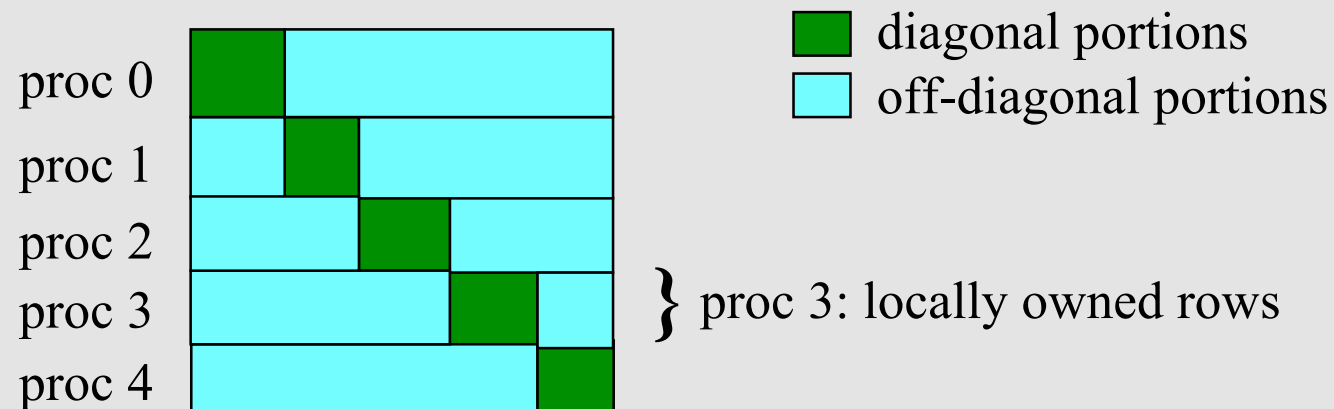
# Symbolic Computation of Matrix Nonzero Structure

- Create matrix with `MatCreate()`
- Set type with `MatSetType()`
- Form the nonzero structure of the matrix
  - loop over the grid for finite differences
  - loop over the elements for finite elements
  - etc.
- Preallocate matrix
  - `MatSeqAIJSetPreallocation()`
  - `MatMPIAIJSetPreallocation()`

profiling and  
performance tuning

# Parallel Sparse Matrices

- Each process locally owns a submatrix of contiguously numbered global rows.
- Each submatrix consists of **diagonal** and **off-diagonal** parts.



profiling and  
performance tuning

# Indicating Expected Nonzeros Parallel Sparse Matrices

```
MatMPIAIJSetPreallocation(Mat A,  
                           int d_nz, int *d_nnz,  
                           int o_nz, int *o_nnz)
```

- `d_nnz[]` - expected number of nonzeros per row in diagonal portion of local submatrix
- `o_nnz[]` - expected number of nonzeros per row in off-diagonal portion of local submatrix

profiling and  
performance tuning

# Verifying Predictions

- Use runtime option: `-log_info`
- Output:
  - [proc #] Matrix size: %d X %d; storage space: %d  
unnneeded, %d used
  - [proc #] Number of mallocs during MatSetValues( ) is %d

```
[merlin] mpirun ex2 -log_info
[0]MatAssemblyEnd_SeqAIJ:Matrix size: 56 X 56; storage space:
[0] 310 unnneeded, 250 used
[0]MatAssemblyEnd_SeqAIJ:Number of mallocs during MatSetValues() is 0
[0]MatAssemblyEnd_SeqAIJ:Most nonzeros in any row is 5
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routines
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routines
Norm of error 0.000156044 iterations 6
[0]PetscFinalize:PETSc successfully ended!
```

profiling and  
performance tuning

# Extending PETSc



# Linking to PETSc

- Application program or framework
- Nothing but the libraries
  - Custom link by user
- Using the PETSc build variables
  - Include `bmake/common/variables`
- Using the PETSc build rules
  - Include `bmake/common/base`
  - Also makes available 3<sup>rd</sup> party packages

# Customizing the Framework

- Adding additional logging
  - Pushing new stages
  - Registering new events and logging classes
- Adding 3<sup>rd</sup> party PETSc libraries
  - Enlarging the dynamic link path

# Adding an Implementation

- See `src/ksp/pc/impls/jacobi/jacobi.c`
- Implement the interface methods
  - For Jacobi, `PCSetUp()`, `PCApply()`,...
- Define a constructor
  - Allocate and initialize the class structure
  - Fill in the function table
  - Must have C linkage
- Register the constructor
  - See `src/ksp/ksp/interface/dlregis.c`
  - Maps a string (class name) to the constructor
  - Usually uses `PetscFListAdd()`

# Adding a Wrapper

- See `src/ts/impls/implicit/pvode/petscpvode.c`
- Just like an Implementation
  - Methods dispatch to 3<sup>rd</sup> party software
- Need to alter local makefile
  - Add a `requirespackage` line
  - Add include variable to `CPPFLAGS`
- Also need usual configure build additions

# Adding a Subtype

- See `src/mat/impls/aij/seq/umfpack/umfpack.c`
- Have to virtualize methods by hand
- Define a constructor
  - Change type name first to correct name
  - Call `MatSetType()` for base type
  - Replace (and save) overridden methods
  - Construct any specific data
- Must also define a conversion to the base type
  - Only called in destructor right now

# Adding a Type I

- See `src/ksp/ksp/kspimpl.h`
- Define a methods structure (interface)
  - A list of function pointers
- Define a type structure
  - First member is `PETSCHEADER(struct _Ops)`
  - Possibly other data members common to the type
  - A `void *data` for implementation structures

# Adding a Type II

- See `src/ksp/ksp/interface/dlregis.c`
- Define a package initializer (`PetscDLLibraryRegister`)
  - Called when the DLL is loaded
    - Also called from generic create if linking statically
  - Registers classes and events (see below)
  - Registers any default implementation constructors
  - Setup any info or summary exclusions

# Adding a Type III

- See `src/ksp/ksp/interface/itcreate.c`
- Define a generic create
  - Call package initializer if linking statically
  - Call `PetscHeaderCreate()`
  - Initialize any default data members
- Define a `setType()` method
  - Call the destructor of any current implementation
  - Call the constructor of the given implementation
  - Set the type name



# Adding a Type IV

- Things swept under the rug
  - Want a `setFromOptions()` which allows selection of implementations at runtime
  - Have to manage the database of registered constructors
  - View and destroy functions handled a little funny due to historical reasons

# Nonlinear Solvers (SNES)

*SNES: Scalable Nonlinear Equations Solvers*

- Application code interface
- Choosing the solver
- Setting algorithmic options
- Viewing the solver
- Determining and monitoring convergence
- Matrix-free solvers
- User-defined customizations

solvers:  
nonlinear

# Nonlinear Solvers

**Goal:** For problems arising from PDEs, support the general solution of  $F(u) = 0$

User provides:

- Code to evaluate  $F(u)$
- Code to evaluate Jacobian of  $F(u)$  (optional)
  - or use sparse finite difference approximation
  - or use automatic differentiation
    - AD support via collaboration with P. Hovland and B. Norris
    - Coming in next PETSc release via automated interface to ADIFOR and ADIC (see <http://www.mcs.anl.gov/autodiff>)

solvers:  
nonlinear

# Nonlinear Solvers (SNES)

- Newton-based methods, including
  - Line search strategies
  - Trust region approaches (using TAO)
  - Pseudo-transient continuation
  - Matrix-free variants
- Can customize all phases of solution process

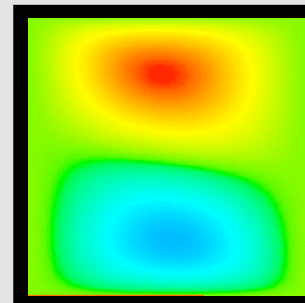
solvers:  
nonlinear

# Sample Nonlinear Application: Driven Cavity Problem

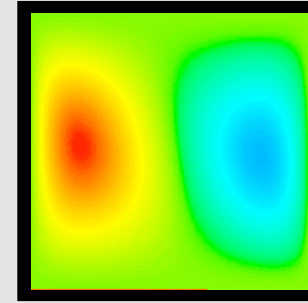
- Velocity-vorticity formulation
- Flow driven by lid and/or bouyancy
- Logically regular grid, parallelized with DAs
- Finite difference discretization
- source code:

[petsc/src/snes/examples/tutorials/ex19.c](https://petsc.org/src/snes/examples/tutorials/ex19.c)

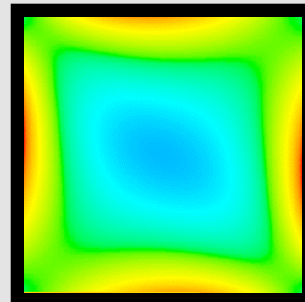
## Solution Components



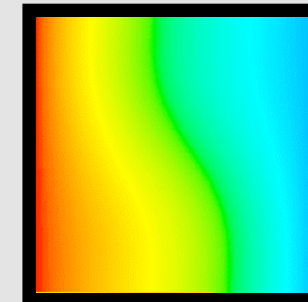
velocity:  $u$



velocity:  $v$



vorticity:  $\zeta$



temperature:  $T$

*Application code author: D. E. Keyes*

solvers:  
nonlinear

# Basic Nonlinear Solver Code (C/C++)

```
SNES snes;           /* nonlinear solver context */
Mat J;              /* Jacobian matrix */
Vec x, F;           /* solution, residual vectors */
int n, its;         /* problem dimension, number of iterations */
ApplicationCtx usercontext; /* user-defined application context */
```

...

```
MatCreate(PETSC_COMM_WORLD, &J);
MatSetSizes(J, PETSC_DECIDE, PETSC_DECIDE, n, n)
MatSetFromOptions(J);
VecCreate(PETSC_COMM_WORLD, &x);
VecSetSizes(x, PETSC_DECIDE, n);
VecSetFromOptions(x);
VecDuplicate(x, &F);

SNESCreate(PETSC_COMM_WORLD, &snest);
SNESSetFunction(snest, F, EvaluateFunction, usercontext);
SNESSetJacobian(snest, J, J, EvaluateJacobian, usercontext);
SNESSetFromOptions(snest);
SNESolve(snest, x, &its);
SNESDestroy(snest);
```

solvers:  
nonlinear

# Solvers Based on Callbacks

- User provides routines to perform actions that the library requires. For example,
  - `SNESSetFunction(SNES,...)`
    - `uservector` - vector to store function values
    - `userfunction` - name of the user's function
    - `usercontext` - pointer to private data for the user's function
- Now, whenever the library needs to evaluate the user's nonlinear function, the solver may call the application code directly with its own local state.
- **usercontext**: serves as an application context object. Data are handled through such opaque objects; the library never sees irrelevant application data.



important concept

solvers:  
nonlinear

# Sample Application Context: Driven Cavity Problem

```
typedef struct {  
    /* ----- basic application data ----- */  
    double    lid_velocity, prandtl;    /* problem parameters */  
    double    grashof;                 /* problem parameters */  
    int       mx, my;                 /* discretization parameters */  
    int       mc;                     /* number of DoF per node */  
    int       draw_contours;          /* flag - drawing contours */  
    /* ----- parallel data ----- */  
    MPI_Comm comm;                    /* communicator */  
    DA        da;                      /* distributed array */  
    Vec       localF, localX;          /* local ghosted vectors */  
} AppCtx;
```

solvers:  
nonlinear



# Sample Function Evaluation Code: Driven Cavity Problem

```
UserComputeFunction(SNES snes, Vec X, Vec F, void *ptr)
{
  AppCtx *user = (AppCtx *) ptr; /* user-defined application context */
  int      istart, iend, jstart, jend; /* local starting and ending grid points */
  Scalar   *f; /* local vector data */
  ....
  /* (Code to communicate nonlocal ghost point data not shown) */
  VecGetArray( F, &f );
  /* (Code to compute local function components; insert into f[ ] shown on
  next slide) */
  VecRestoreArray( F, &f );
  ....
  return 0;
}
```

solvers:  
nonlinear

# Sample Local Computational Loops: Driven Cavity Problem

```

....
for ( j = jstart; j<jend; j++ ) {
    row = (j - gys) * gxm + istart;
    for ( i = istart; i<iend; i++ ) {
        row++;    u = x[row].u;
        uxx = (two * u - x[row-1].u - x [ row+1 ].u ) * hyd hx;
        uyy = (two * u - x[row-gxm].u - x [row+gx m].u * hxdhy;
        f [row].u = uxx + uyy -
            p5 * (x [row+gx m].omega- x [row-gxm].omega) * hx;
        .....
    }
}
....

```

- The PDE's 4 components (U,V,Omega,Temp) are interleaved in the unknown vector.
- `gxs` and `gys` statements provide easy access to each component.

solvers:  
nonlinear

# Finite Difference Jacobian Computation

- Compute and explicitly store Jacobian via 1<sup>st</sup>-order FD
  - Dense: `-snes_fd`, `SNESDefaultComputeJacobian()`
  - Sparse via colorings: `MatFDColoringCreate()`, `SNESDefaultComputeJacobianColor()`
- Matrix-free Newton-Krylov via 1<sup>st</sup>-order FD, no preconditioning unless specifically set by user
  - `-snes_mf`
- Matrix-free Newton-Krylov via 1<sup>st</sup>-order FD, user-defined preconditioning matrix
  - `-snes_mf_operator`

solvers:  
nonlinear

# Uniform access to all linear and nonlinear solvers

- -ksp\_type [cg,gmres,bcgs,tfqmr,...]
- -pc\_type [lu,ilu,jacobi,sor,asm,...]
- -snes\_type [ls,...]

1

- -snes\_line\_search <line search method>
- -sles\_ls <parameters>
- -snes\_convergence <tolerance>
- etc...

2

solvers:  
nonlinear

# Customization via Callbacks:

## Setting a user-defined line search routine

```
SNESSetLineSearch(SNES  
  snes,int(*ls)(...),void *lsctx)
```

Available line search routines `ls( )` include:

- `SNESCubicLineSearch( )` - cubic line search
- `SNESQuadraticLineSearch( )` - quadratic line search
- `SNESNoLineSearch( )` - full Newton step
- `SNESNoLineSearchNoNorms( )` - full Newton step but calculates no norms (faster in parallel, useful when using a fixed number of Newton iterations instead of usual convergence testing)
- `YourOwnFavoriteLineSearchRoutine( )`

solvers:  
nonlinear

# SNES: Review of Basic Usage

- `SNESCreate( )` - Create SNES context
- `SNESSetFunction( )` - Set function eval.  
routine
- `SNESSetJacobian( )` - Set Jacobian eval.  
routine
- `SNESSetFromOptions( )` - Set runtime solver options  
for [SNES,SLES, KSP,PC]
- `SNESSolve( )` - Run nonlinear solver
- `SNESView( )` - View solver options  
actually used at runtime  
(alternative: `-snes_view`)
- `SNESDestroy( )` - Destroy solver

solvers:  
nonlinear

# SNES: Review of Selected Options

Functionality	Procedural Interface	Runtime Option
Set nonlinear solver	SNESsetType( )	-snes_type [ls,tr,uuls,umtr,...]
Set monitoring routine	SNESsetMonitor( )	-snes_monitor -snes_xmonitor, ...
Set convergence tolerances	SNESsetTolerances( )	-snes_rtol <rt> -snes_atol <at> -snes_max_its <its>
Set line search routine	SNESsetLineSearch( )	-snes_eq_ls [cubic,quadratic,...]
View solver options	SNESView( )	-snes_view
Set linear solver options	SNESgetSLES( ) SLESgetKSP( ) SLESgetPC( )	-ksp_type <ksptype> -ksp_rtol <krt> -pc_type <pctype> ...

*And many more options...*

solvers:  
nonlinear

# Parallel Data Layout and Ghost Values: Usage Concepts

*Managing **field data layout** and required **ghost values** is the key to high performance of most PDE-based parallel programs.*

## Mesh Types

- Structured
  - DA objects
- Unstructured
  - VecScatter objects

## Usage Concepts

- Geometric data
- Data structure creation
- Ghost point updates
- Local numerical computation



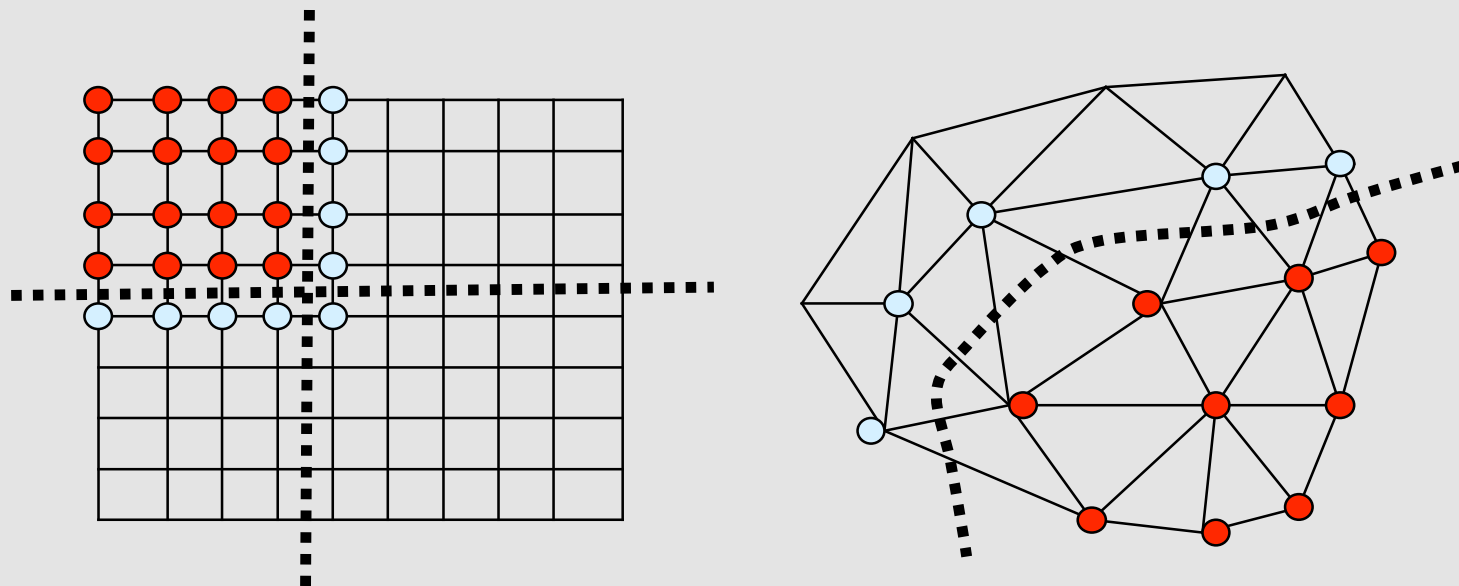
important concepts

data layout



# Ghost Values

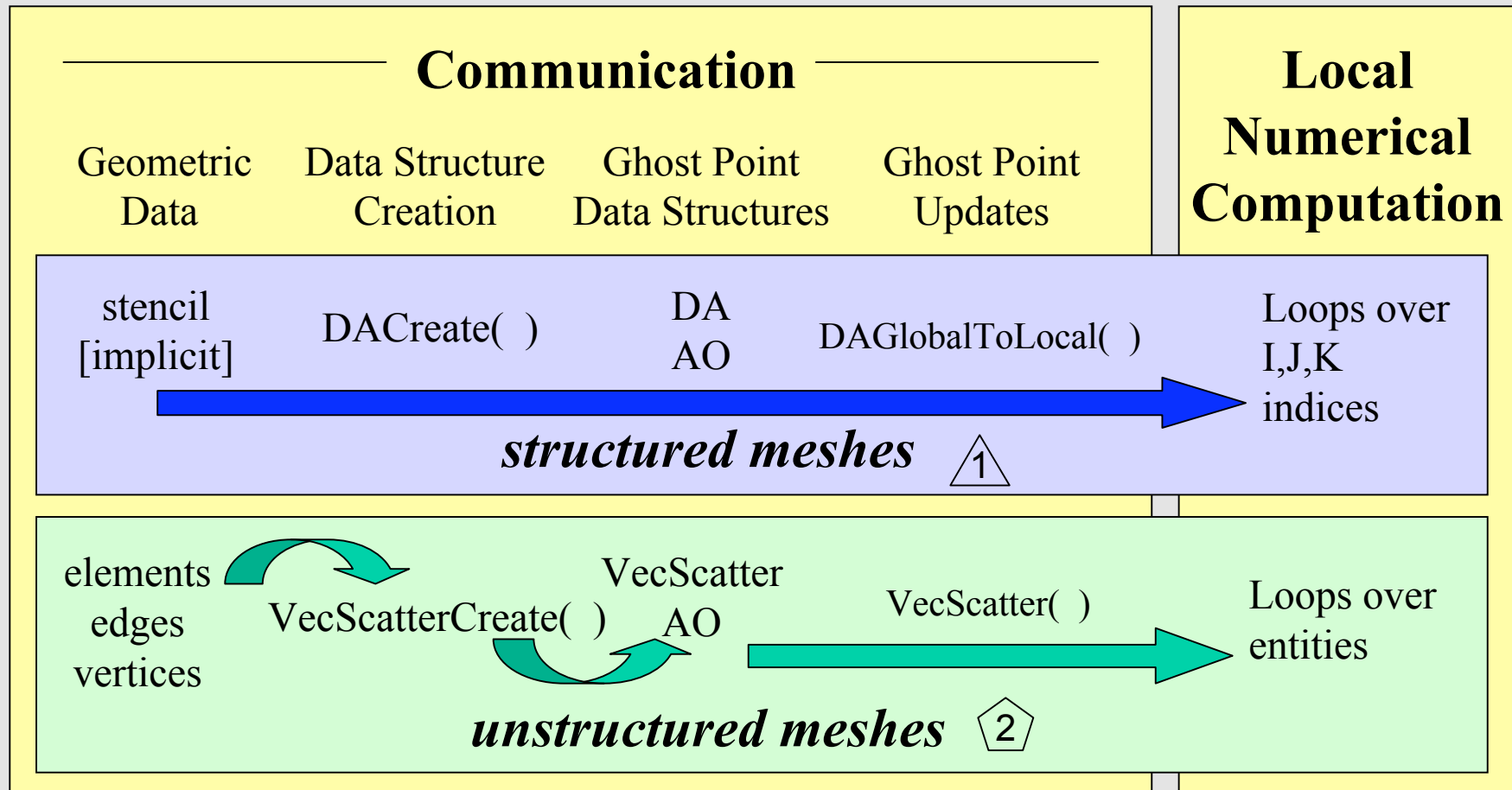
● Local node      ○ Ghost node



**Ghost values:** To evaluate a local function  $f(x)$ , each process requires its local portion of the vector  $x$  as well as its **ghost values** – or bordering portions of  $x$  that are owned by neighboring processes.

data layout

# Communication and Physical Discretization



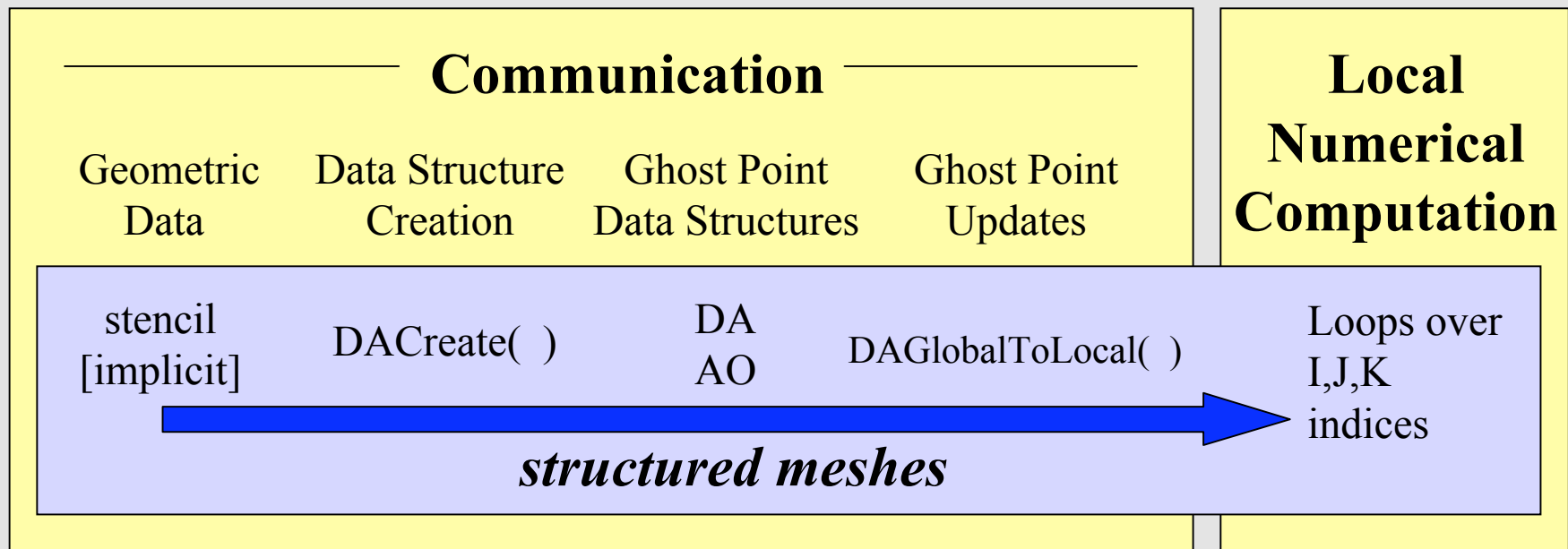
data layout

# DA: Parallel Data Layout and Ghost Values for Structured Meshes

- Local and global indices
- Local and global vectors
- DA creation
- Ghost point updates
- Viewing

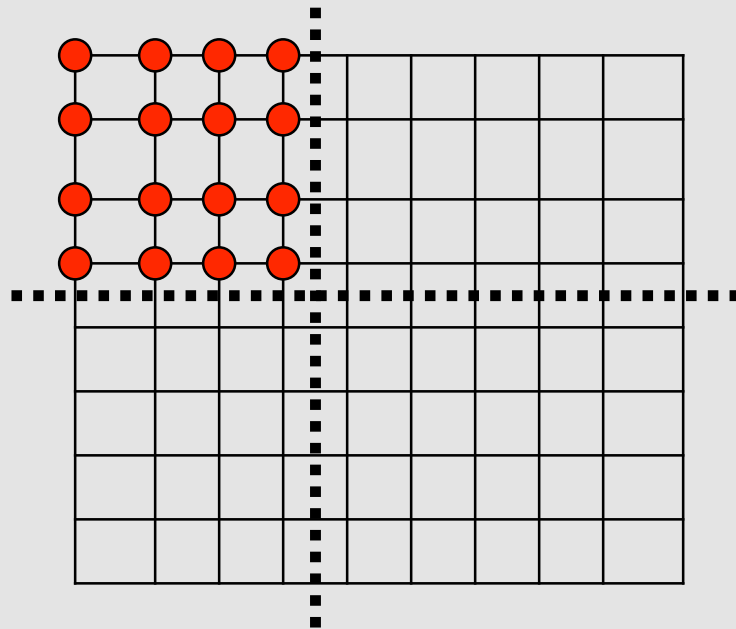
data layout:  
distributed arrays

# Communication and Physical Discretization: Structured Meshes



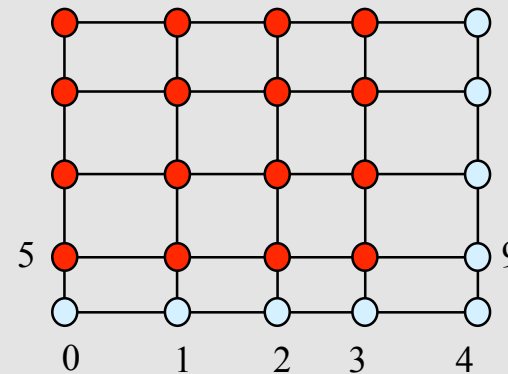
data layout:  
distributed arrays

# Global and Local Representations



**Global:** each process stores a unique local set of vertices (and each vertex is owned by exactly one process)

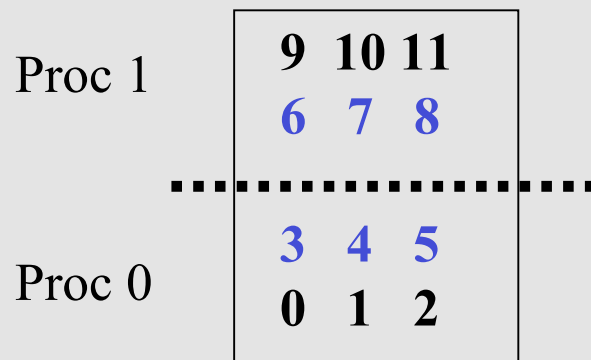
- Local node
- Ghost node



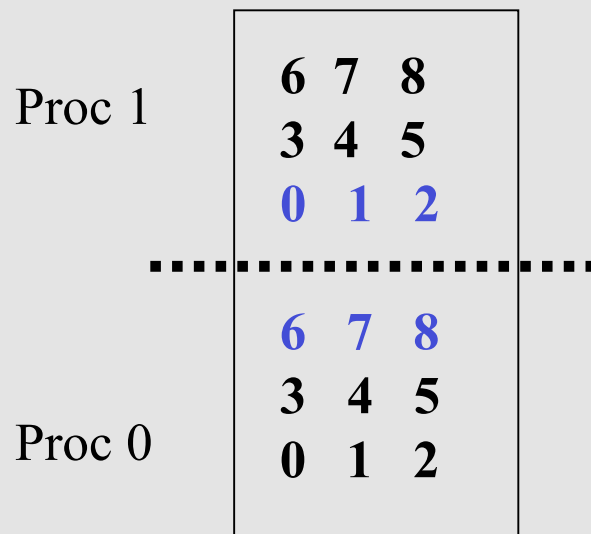
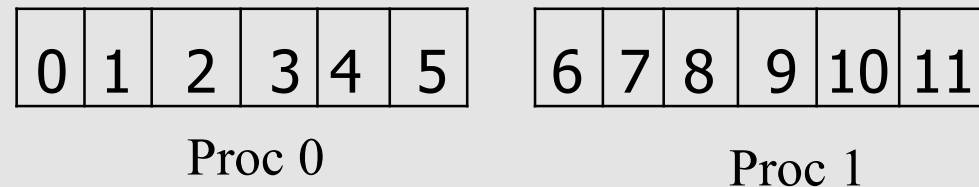
**Local:** each process stores a unique local set of vertices *as well as* ghost nodes from neighboring processes

data layout:  
distributed arrays

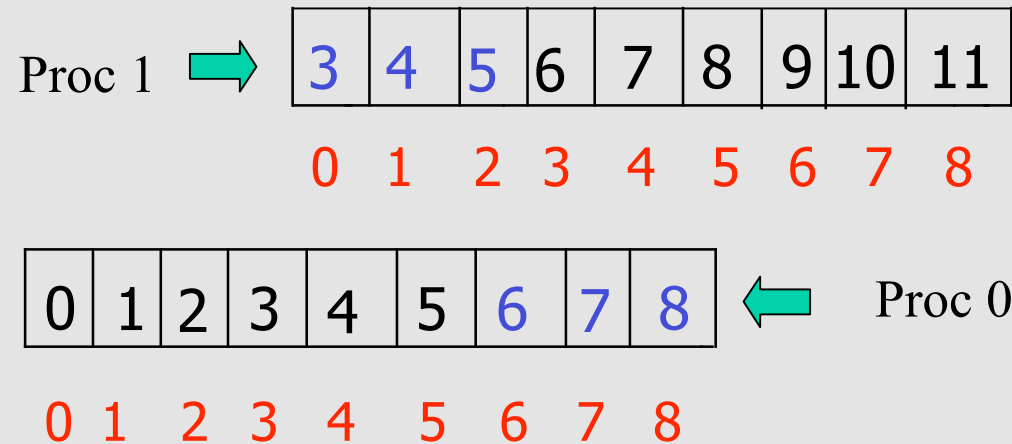
# Global and Local Representations (cont.)



## Global Representation:



## Local Representations:



data layout:  
distributed arrays

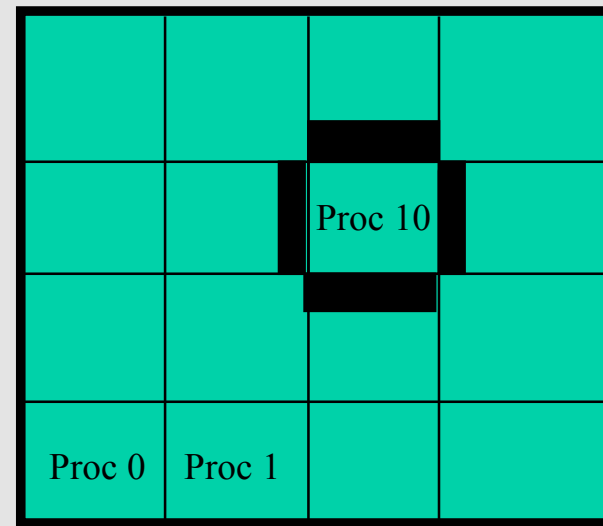
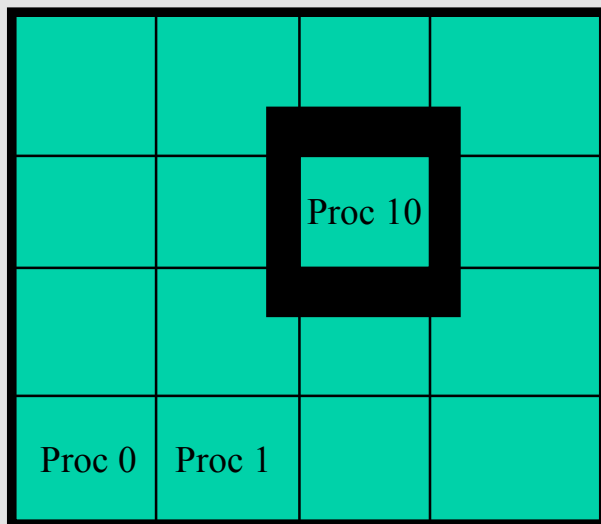
# Logically Regular Meshes

- **DA - Distributed Array**: object containing information about vector layout across the processes and communication of ghost values
- Form a DA
  - `DACreate1d(..., DA *)`
  - `DACreate2d(..., DA *)`
  - `DACreate3d(..., DA *)`
- Create the corresponding PETSc vectors
  - `DACreateGlobalVector( DA, Vec *)` or
  - `DACreateLocalVector( DA, Vec *)`
- Update ghostpoints (scatter global vector into local parts, including ghost points)
  - `DAGlobalToLocalBegin(DA, ...)`
  - `DAGlobalToLocalEnd(DA, ...)`

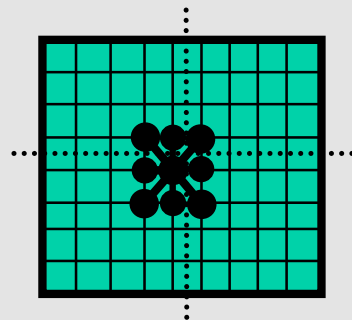
data layout:  
distributed arrays

# Distributed Arrays

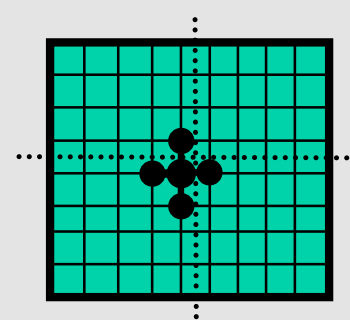
Data layout and ghost values



*Box-type  
stencil*



*Star-type  
stencil*



data layout:  
distributed arrays



# Vectors and DAs

- The DA object contains information about the data layout and ghost values, but **not** the actual field data, which is contained in PETSc vectors
- Global vector: parallel
  - each process stores a unique local portion
  - `DACreateGlobalVector(DA da, Vec *gvec);`
- Local work vector: sequential
  - each process stores its local portion plus ghost values
  - `DACreateLocalVector(DA da, Vec *lvec);`
  - uses “natural” local numbering of indices  $(0, 1, \dots, n_{local}-1)$

data layout:  
distributed arrays

# DACreate1d(..., \*DA)

- D A Create1d(M P I \_ C o m m c o m m , D A P e r i o d i c T y p e w r a p ,  
                  i n t M , i n t d o f , i n t s , i n t \* l c , D A \* i n r a )
  - M P I \_ C o m m — p r o c e s s e s c o n t a i n i n g a r r a y
  - D A \_ [ N O N P E R I O D I C , X P E R I O D I C ]
  - n u m b e r o f g r i d p o i n t s i n x - d i r e c t i o n
  - d e g r e e s o f f r e e d o m p e r n o d e
  - s t e n c i l w i d t h
  - N u m b e r o f n o d e s f o r e a c h d o m a i n
    - U s e P E T S C \_ N U L L f o r t h e d e f a u l t

data layout:  
distributed arrays

# DACreate2d(...,\*DA)

- D A Create2d(M P I \_ C o m m c o m m , D A P e r i o d i c T y p e w r a p ,  
D A S t e n c i l T y p e s t e n c i l \_ t y p e , i n t M , i n t N ,  
i n t m , i n t n , i n t d o f , i n t s , i n t \* l x , i n t \* l y , D A \* i n r a )
  - D A \_ [ N O N , X , Y , X Y ] P E R I O D I C
  - D A \_ S T E N C I L \_ [ S T A R , B O X ]
  - number of grid points in x- and y-directions
  - processes in x- and y-directions
  - degrees of freedom per node
  - stencil width
  - Number of nodes for each domain
    - Use P E T S C \_ N U L L for the default

data layout:  
distributed arrays

# Updating the Local Representation

Two-step process enables overlapping computation and communication

- `DAGlobalToLocalBegin(DA, global_vec, insert, local_vec )`
  - `global_vec` provides data
  - Insert is either `INSERT_VALUES` or `ADD_VALUES`
    - specifies how to update values in the local vector
  - `local_vec` is a pre-existing local vector
- `DAGlobalToLocal End(DA,...)`
  - Takes same arguments

data layout:  
distributed arrays

# Ghost Point Scatters: Burger's Equation Example

```
call DAGlobalToLocalBegin(da,u_global,INSERT_VALUES,ulocal,ierr)
call DAGlobalToLocalEnd(da,u_global,INSERT_VALUES,ulocal,ierr)
```

```
call VecGetArray( ulocal, uv, ui, ierr )
```

```
#define u(i) uv(ui+i)
```

```
C Do local computations (here u and f are local vectors)
```

```
do 10, i=1,localsize
```

```
    f(i) = (.5/h)*u(i)*(u(i+1) - u(i-1)) + (e/(h*h))*(u(i+1) - 2.0*u(i) + u(i-1))
```

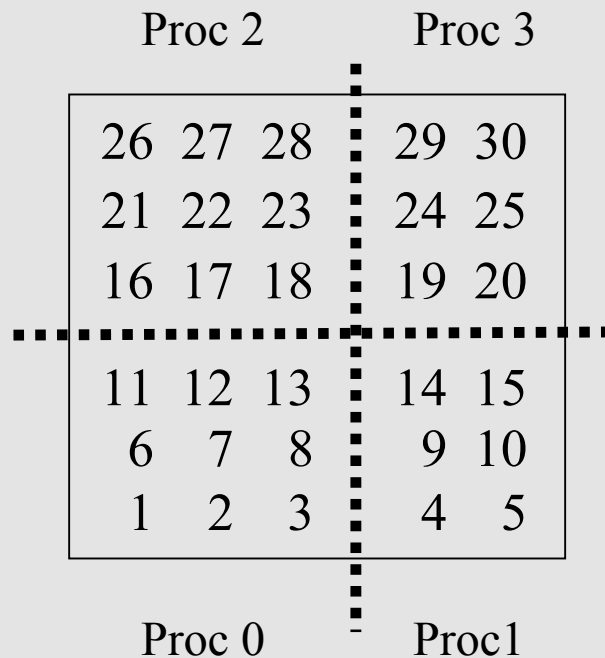
```
10 continue
```

```
call VecRestoreArray( ulocal, uv, ui, ierr )
```

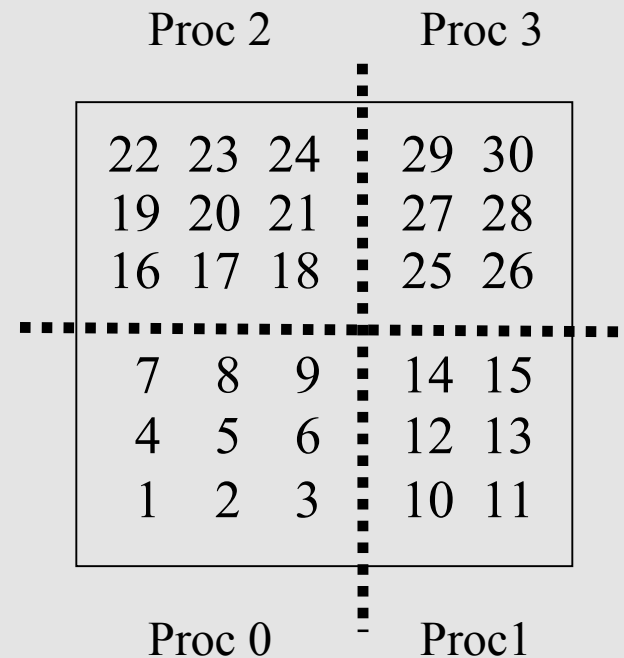
```
call DALocalToGlobal(da,f,INSERT_VALUES,f_global,ierr)
```

data layout: distributed arrays
------------------------------------

# Global Numbering used by DAs



Natural numbering, corresponding to the entire problem domain



PETSc numbering used by DAs

data layout:  
distributed arrays

# Mapping Between Global Numberings

- Natural global numbering
  - convenient for visualization of global problem, specification of certain boundary conditions, etc.
- Can convert between various global numbering schemes using AO (Application Orderings)
  - `DAGetAO(DA da, AO *ao);`
  - AO usage explained in next section
- Some utilities (e.g., `VecView()`) automatically handle this mapping for global vectors attained from DAs

data layout:  
distributed arrays

# Distributed Array Example

- Files: `src/snes/examples/tutorial/ex5.c`, `ex5f.F`
  - Functions that construct vectors and matrices use a naturally associated DA
    - `DAGetMatrix()`
    - `DASetLocalFunction()`
    - `DASetLocalJacobian()`

data layout:  
distributed arrays



# The Bratu Equation I

## SNES Example 5

- Create SNES and DA
- Use `DASetLocalFunction()`
  - Similarly `DASetLocalJacobian()`
- Use `SNESDAFormFunction()` for SNES
  - Could also use `FormFunctionMatlab()`
- Similarly `SNESDAComputeJacobian()`
  - Use `DAGetMatrix()` for SNES Jacobian
  - Could also use `SNESDefaultComputeJacobian()`

# The Bratu Equation II

## SNES Example 5

- ```
int FormFunctionLocal(DALocalInfo *info,
                    PetscScalar **x, PetscScalar **f,
                    void *ctx)

for(j = info->ys; j < info->ys + info->ym; j++) {
  for(i = info->xs; i < info->xs + info->xm; i++) {
    if (i == 0 || j == 0 || i == info->mx-1 || j == info->my-1) {
      f[j][i] = x[j][i];
    } else {
      u      = x[j][i];
      u_xx   = -(x[j][i+1] - 2.0*u + x[j][i-1])*(hy/hx);
      u_yy   = -(x[j+1][i] - 2.0*u + x[j-1][i])*(hx/hy);
      f[j][i] = u_xx + u_yy - hx*hy*lambda*PetscExpScalar(u);
    }
  }
}
```

# The Bratu Equation III

## SNES Example 5

- `int FormJacobianLocal(DALocalInfo *info, PetscScalar **x, Mat jac, void *ctx)`

```
for(j = info->ys; j < info->ys + info->ym; j++) {
  for(i = info->xs; i < info->xs + info->xm; i++) {
    row.j = j; row.i = i;
    if (i == 0 || j == 0 || i == info->mx-1 || j == info->my-1) {
      v[0] = 1.0;
      MatSetValuesStencil(jac, 1, &row, 1, &row, v, INSERT_VALUES);
    } else {
      v[0] = -(hx/hy); col[0].j = j-1; col[0].i = i;
      v[1] = -(hy/hx); col[1].j = j; col[1].i = i-1;
      v[2] = 2.0*(hy/hx+hx/hy) - hx*hy*lambda*PetscExpScalar(x[j][i]);
      v[3] = -(hy/hx); col[3].j = j; col[3].i = i+1;
      v[4] = -(hx/hy); col[4].j = j+1; col[4].i = i;
      MatSetValuesStencil(jac, 1, &row, 5, col, v, INSERT_VALUES);
    }
  }
}
```

# VecScatter: Vector Scatters & Gathers

*Parallel data layout and ghost values for unstructured meshes*

intermediate

intermediate

intermediate

intermediate

intermediate

- Mapping between global numberings
- Local and global indices
- VecScatter creation
- Ghost point updates
- Setting object data using local indices

tutorial outline:  
data layout:  
vector scatters

# Unstructured Meshes

- Setting up communication patterns is much more complicated than the structured case due to
  - mesh dependence
  - discretization dependence

beginner

data layout:  
vector scatters

# Sample Differences Among Discretizations

- Cell-centered
- Vertex-centered
- Cell and vertex centered (e.g., staggered grids)
- Mixed triangles and quadrilaterals

beginner

data layout:  
vector scatters

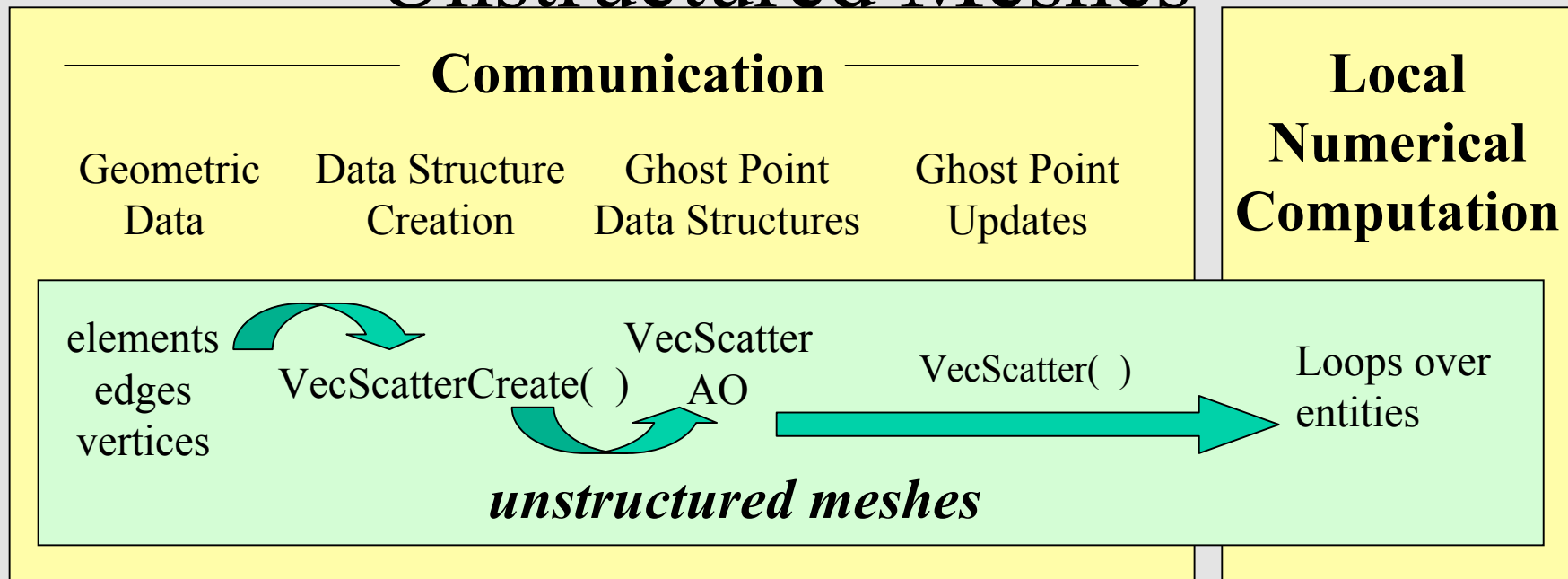
# Unstructured Mesh Concepts

- **AO: Application Orderings**
  - map between various global numbering schemes
- **IS: Index Sets**
  - indicate collections of nodes, cells, etc.
- **VecScatter:**
  - update ghost points using vector scatters/gathers
- **ISLocalToGlobalMapping**
  - map from a local (on-processor) numbering of indices to a global (across-processor) numbering

intermediate

data layout:  
vector scatters

# Communication and Physical Discretization: Unstructured Meshes



intermediate

data layout:  
vector scatters



# Setting Up the Communication Pattern

(the steps in creating VecScatter)

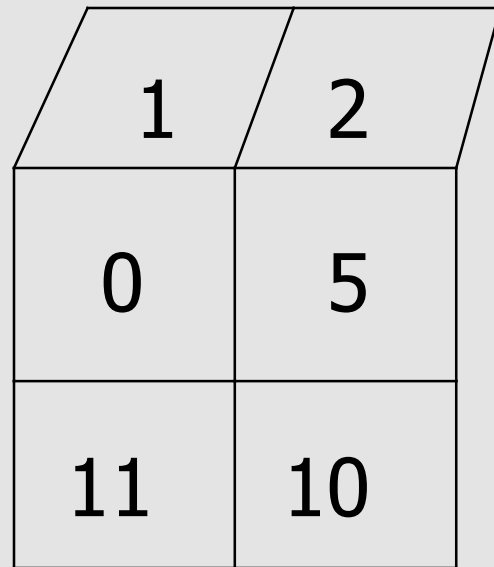
- Renumber objects so that contiguous entries are adjacent (AO)
- Determine needed neighbor values
- Generate local numbering
- Generate local and global vectors
- Create communication object (VecScatter)

intermediate

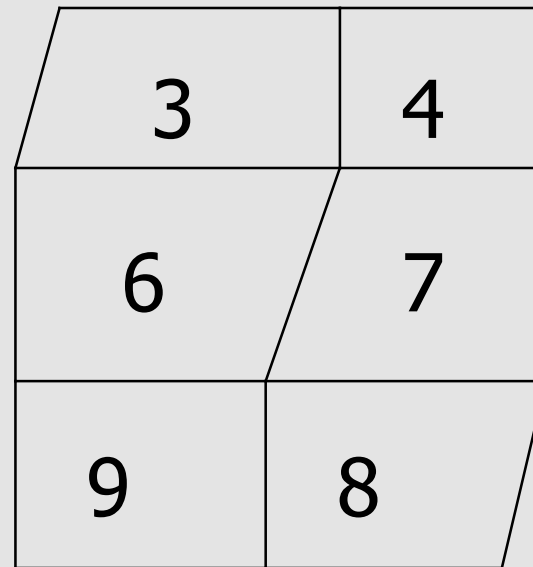
data layout:  
vector scatters

# Cell-based Finite Volume Application Numbering

Proc 0



Proc1



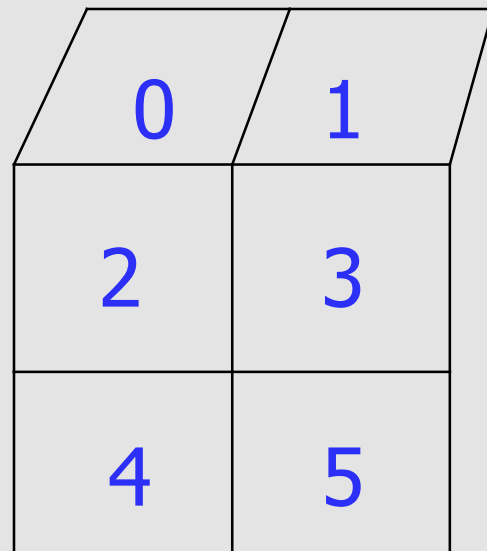
global indices defined by application

intermediate

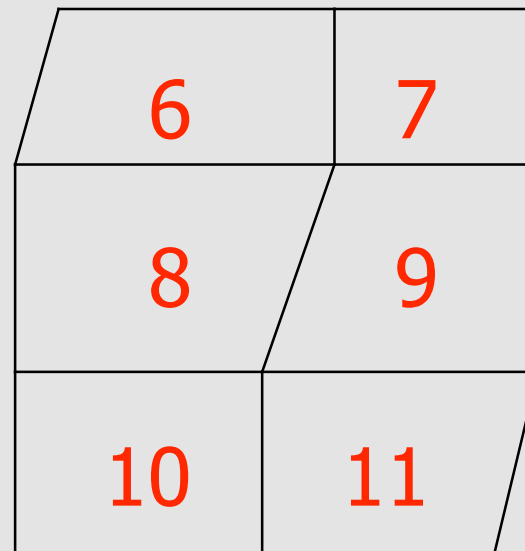
data layout:  
vector scatters

# PETSc Parallel Numbering

Proc 0



Proc 1



global indices numbered contiguously on each processor

intermediate

data layout:  
vector scatters

# Remapping Global Numbers: An Example

- Processor 0

- nlocal: 6

- app\_numbers: {1,2,0,5,11,10}

- petsc\_numbers: {0,1,2,3,4,5}

PETSc  
numbers

| Proc 0 | Proc 1 |
|--------|--------|
| 0 1    | 6 7    |
| 2 3    | 8 9    |
| 4 5    | 10 11  |

- Processor 1

- n\_local: 6

- app\_numbers: {3,4,6,7,9,8}

- petsc\_numbers: {6,7,8,9,10,11}

application  
numbers

|       |     |
|-------|-----|
| 1 2   | 3 4 |
| 0 5   | 6 7 |
| 11 10 | 9 8 |

intermediate

data layout:  
vector scatters

# Remapping Numbers (1)

- Generate a parallel object (AO) to use in mapping between numbering schemes
- `AOCreatBasic(MPI_Comm comm,`
  - `int nlocal,`
  - `int *app_numbers,`
  - `int *petsc_numbers,`
  - `AO *ao);`

intermediate

data layout:  
vector scatters

## Remapping Numbers (2)

- `AOApplicationToPetsc(AO *ao,`
  - `int number_to_convert,`
  - `int *indices);`
- For example, if `indices[ ]` contains the cell neighbor lists in an application numbering, then apply `AO` to convert to new numbering

intermediate

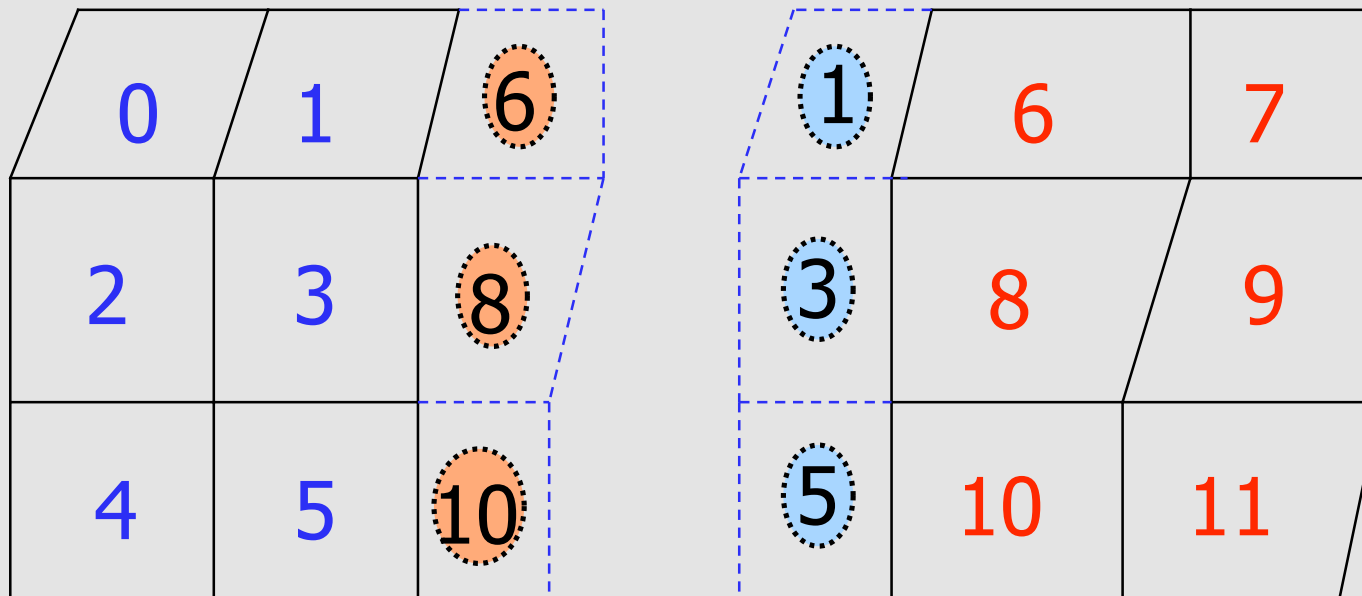
data layout:  
vector scatters

# Neighbors

using global PETSc numbers

Proc 0

Proc1



intermediate

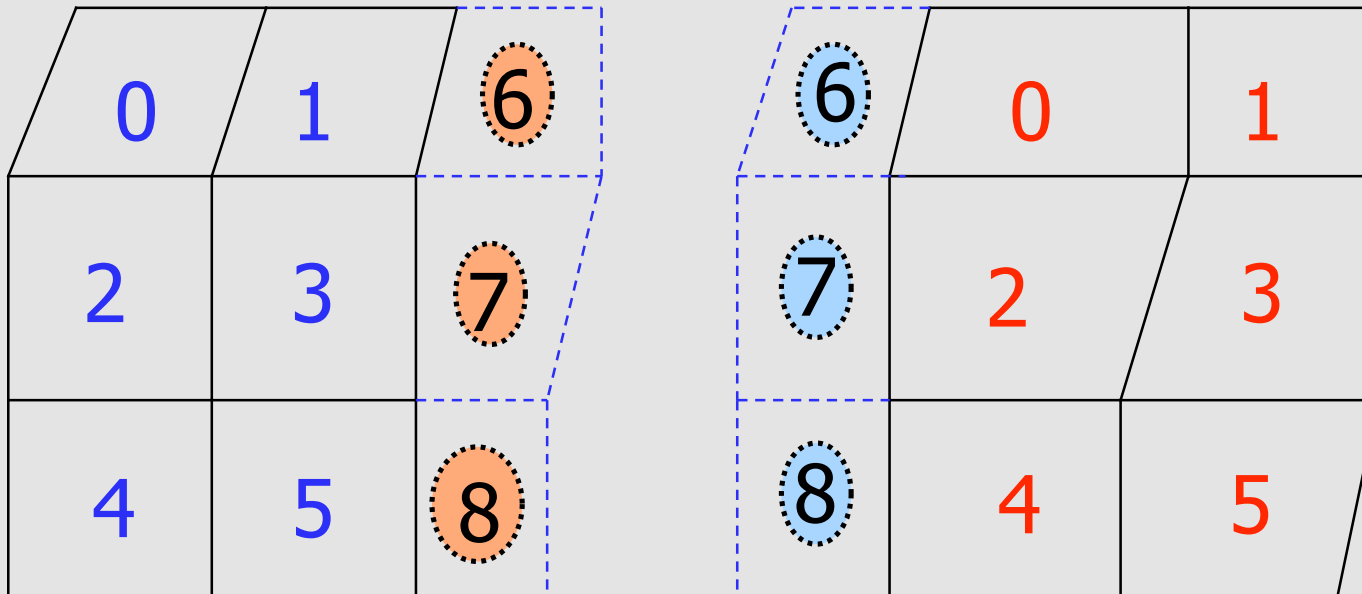
ghost cells

data layout:  
vector scatters

# Local Numbering

Proc 0

Proc 1



intermediate

ghost cells

data layout:  
vector scatters



# Global and Local Representations

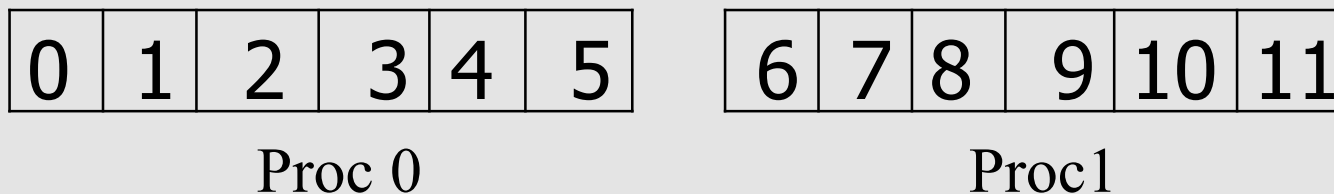
- Global representation
  - parallel vector with no ghost locations
  - suitable for use by PETSc parallel solvers
- Local representation
  - sequential vectors with room for ghost points
  - used to evaluate functions, Jacobians, etc.

intermediate

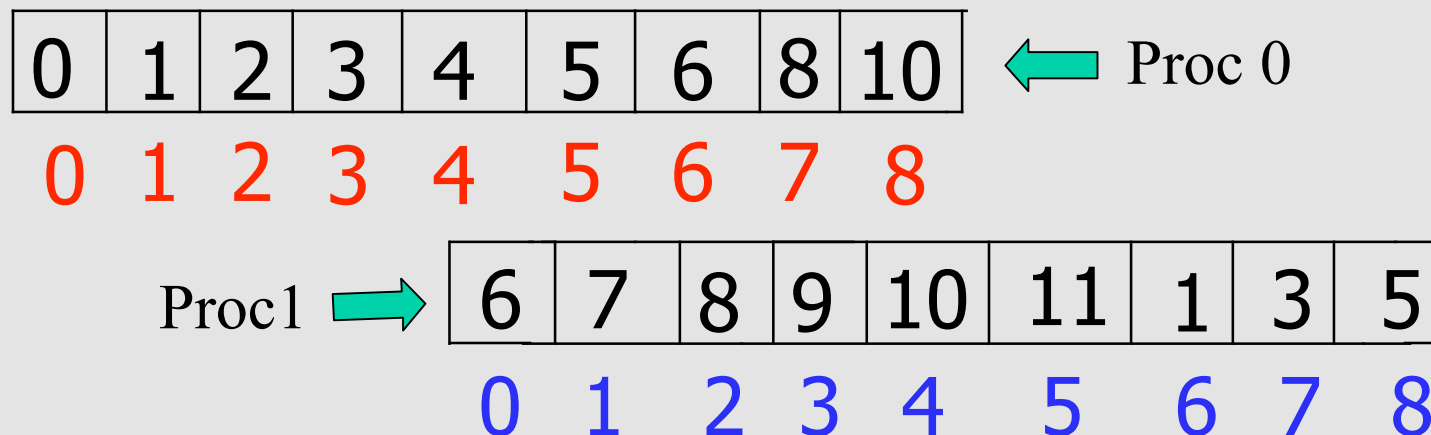
data layout:  
vector scatters

# Global and Local Representations

Global representation:



Local Representations:



intermediate

data layout:  
vector scatters

# Creating Vectors

- Sequential

```
VecCreateSeq(PETSC_COMM_SELF,  
             9,Vec *lvec);
```

- Parallel

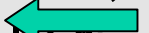
```
VecCreateMPI(PETSC_COMM_WORLD,  
             6,PETSC_DETERMINE,Vec  
*gvec)
```

intermediate

data layout:  
vector scatters

# Local and Global Indices

- Processor 0

- `ISCreateGeneral(PETSC_COMM_WORLD, 9, {0, 1, 2, 3, 4, 5, 6, 8, 10}, IS* isg);`  `ISCreateGeneral()` - Specify *global* numbers of locally owned cells, including ghost cells
- `ISCreateStride(PETSC_COMM_SELF, 9, 0, 1, IS* isl);`

- Processor 1

- `ISCreateGeneral(PETSC_COMM_WORLD, 9, {6, 7, 8, 9, 10, 11, 1, 3, 5}, IS* isg);`  `ISCreateStride()` Specify *local* numbers of locally owned cells, including ghost cells
- `ISCreateStride(PETSC_COMM_SELF, 9, 0, 1, IS* isl);` 

intermediate

data layout:  
vector scatters

# Creating Communication Objects

- VecScatterCreate(Vec gvec,
  - IS gis,
  - Vec lvec,
  - IS lis
  - VecScatter gtol);
- Determines all required messages for mapping data from a global vector to local (ghosted) vectors

intermediate

data layout:  
vector scatters

# Performing a Global-to-Local Scatter

Two-step process that enables overlapping computation and communication

- `VecScatterBegin(VecScatter gtol,`
  - `Vec gvec,`
  - `Vec lvec,`
  - `INSERT_VALUES`
  - `SCATTER_FORWARD);`
- `VecScatterEnd(...);`

intermediate

data layout:  
vector scatters

# Performing a Local-to-Global Scatter

- `VecScatterBegin(VecScatter gtol,`
  - `Vec lvec,`
  - `Vec gvec,`
  - `ADD_VALUES,`
  - `SCATTER_REVERSE);`
- `VecScatterEnd(...);`

intermediate

data layout:  
vector scatters

# Setting Values in Global Vectors and Matrices using a Local Numbering

- Create mapping
  - ISLocalToGlobalMappingCreateIS(IS gis,  
ISLocalToGlobalMapping \*lgmap);
- Set mapping
  - VecSetLocalToGlobalMapping(Vec gvec,  
ISLocalToGlobalMapping lgmap);
  - MatSetLocalToGlobalMapping(Mat gmat,  
ISLocalToGlobalMapping lgmap);
- Set values with local numbering
  - VecSetValuesLocal(Vec gvec,int ncols, int localcolumns,  
Scalar \*values, ...);
  - MatSetValuesLocal(Mat gmat, ...);
  - MatSetValuesLocalBlocked(Mat gmat, ...);

intermediate

data layout:  
vector scatters



# Sample Function Evaluation

```
int FormFunction(SNES snes, Vec Xglobal, Vec Fglobal, void *ptr)
{
  AppCtx *user = (AppCtx *) ptr;
  double x1, x2, f1, f2, *x, *f;
  int     *edges = user->edges;

  VecScatterBegin(user->scatter, Xglobal, Xlocal, SCATTER_FORWARD,
    INSERT_VALUES);
  VecScatterEnd(user->scatter, Xglobal, Xlocal, SCATTER_FORWARD,
    INSERT_VALUES);

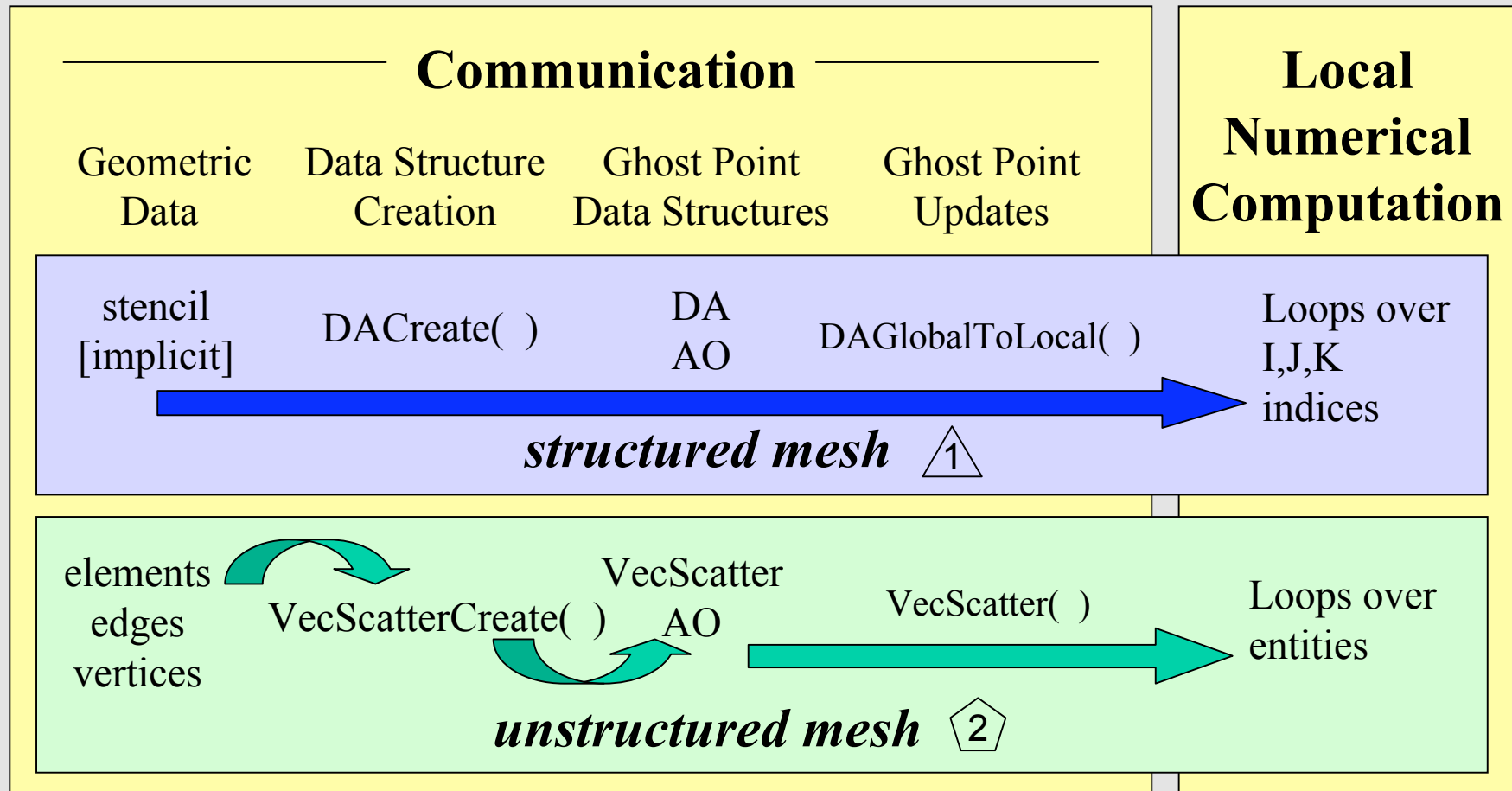
  VecGetArray(Xlocal,&X);  VecGetArray(Flocal,&F);
  for (i=0; i < user->nlocal; i++) {
    x1 = X[edges[2*i]]; x2 = X[edges[2*i+1]]; /* then compute f1, f2 */
    F[edges[2*i]] += f1; F[edges[2*i+1]] += f2;
  }
  VecRestoreArray(Xlocal,&X);  VecRestoreArray(Flocal,&F);

  VecScatterBegin(user->scatter, Flocal, Fglobal, SCATTER_REVERSE,
    INSERT_VALUES);
  VecScatterEnd(user->scatter, Flocal, Fglobal, SCATTER_REVERSE,
    INSERT_VALUES);
  return 0;
}
```

intermediate

data layout:  
vector scatters

# Communication and Physical Discretization



1

2

beginner | intermediate

data layout

# Interfacing to 3<sup>rd</sup> party Packages

## Using PETSc with Other Packages: Linear Solvers

- Interface Approach

- External linear solvers typically use a variant of CSR matrix
- Each package has a matrix subclass with overridden methods

- Usage

- Set preconditioners via the usual approach
  - Procedural interface: `PCSetType(pc,"spai")`
  - Runtime option: `-pc_type spai`
- Set preconditioner-specific options via the usual approach
  - `PCSPAISetEpsilon()`, `PCSPAISetVerbose()`, etc.
  - `-pc_spai_epsilon <eps> -pc_spai_verbose etc.`

Using PETSc with Other Packages:  
**HYPRE - Preconditioners**

- Several preconditioners
  - `-pc_type hypre`
  - `-pc_hypre_type [pilut,parasails,boomeramg,euclid]`
- Specialize preconditioner
  - `-pc_hypre_boomeramg_max_levels <num>`
  - `-pc_hypre_boomeramg_grid_sweeps <fine,down,up,coarse>`
- Options can be displayed with `-help`
  - Only for selected types

## Using PETSc with Other Packages: TAO – Optimization Software

- TAO - Toolkit for Advanced Optimization
  - Software for large-scale optimization problems
  - S. Benson, L. McInnes, and J. Moré
  - <http://www.mcs.anl.gov/tao>
- Initial TAO design uses PETSc for
  - Low-level system infrastructure - managing portability
  - Parallel linear algebra tools (SLES)
    - Veltisto (library for PDE-constrained optimization by G. Biros, see <http://www.cs.nyu.edu/~biros/veltisto>) – uses a similar interface approach
- TAO is evolving toward
  - CCA-compliant component-based design (see <http://www.cca-forum.org>)

# TAO Interface

```
TAO_SOLVER    tao;           /* optimization solver */  
Vec            x, g;         /* solution and gradient vectors */  
ApplicationCtx usercontext; /* user-defined context */
```

**TaoInitialize();**

```
/* Initialize Application -- Create variable and gradient vectors x and g */  
...
```

```
TaoCreate(MPI_COMM_WORLD,"tao_lmvm",&tao);  
TaoSetFunctionGradient(tao,x,g, FctGrad,(void*)&usercontext);
```

**TaoSolve**(tao);

```
/* Finalize application -- Destroy vectors x and g */ ...
```

```
TaoDestroy(tao);  
TaoFinalize();
```

Similar Fortran interface, e.g., **call TaoCreate(...)**

software  
interfacing:  
TAO

# Using PETSc with Other Packages: Matlab

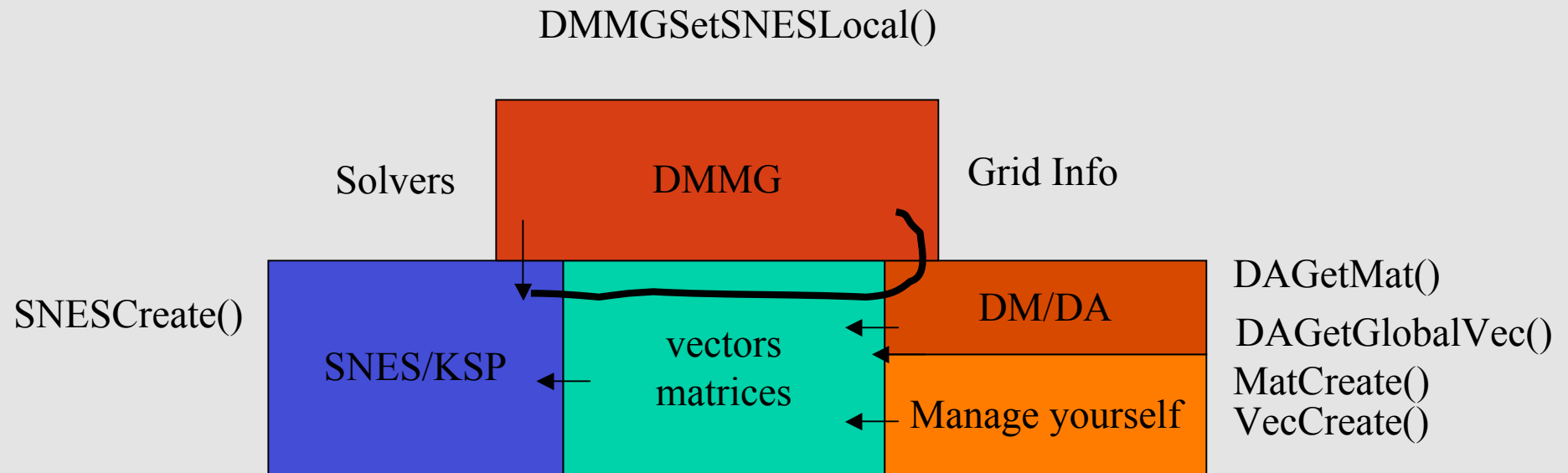
- Matlab
  - <http://www.mathworks.com>
- Interface Approach
  - PETSc socket interface to Matlab
    - Sends matrices and vectors to interactive Matlab session
  - PETSc interface to MatlabEngine
    - MatlabEngine – Matlab library allows C/Fortran programmers to use Matlab functions
    - PetscMatlabEngine – unwraps PETSc vectors and matrices for MatlabEngine
- Usage
  - PetscMatlabEngineCreate(MPI\_Comm,machinename,PetscMatlabEngine eng)
  - PetscMatlabEnginePut(eng,PetscObject obj)
    - Vector
    - Matrix
  - PetscMatlabEngineEvaluate(eng,"R = QR(A);")
  - PetscMatlabEngineGet(eng,PetscObject obj)



# Using PETSc with Other Packages: ParMETIS – Graph Partitioning

- ParMETIS
  - Parallel graph partitioning
  - G. Karypis (Univ. of Minnesota)
  - <http://www.cs.umn.edu/~karypis/metis/parmetis>
- Interface Approach
  - Use PETSc MatPartitioning() interface and MPIAIJ or MPIAdj matrix formats
- Usage
  - MatPartitioningCreate(MPI\_Comm, MatPartitioning ctx)
  - MatPartitioningSetAdjacency(ctx, matrix)
  - Optional – MatPartitioningSetVertexWeights(ctx, weights)
  - MatPartitioningSetFromOptions(ctx)
  - MatPartitioningApply(ctx, IS \*partitioning)

# Three Ways of Using PETSc



# DMMG

data management multigrid

- Creates the vectors, matrices,... appropriate for your grid.
- Connects them to a (hierarchy of) solvers
- User provides “local physics”

# References

- Documentation: <http://www.mcs.anl.gov/petsc/docs>
  - PETSc Users manual
  - Manual pages
  - Many hyperlinked examples
  - FAQ, Troubleshooting info, installation info, etc.
- Publications: <http://www.mcs.anl.gov/petsc/publications>
  - Research and publications that make use PETSc
- MPI Information: <http://www.mpi-forum.org>
- *Using MPI* (2<sup>nd</sup> Edition), by Gropp, Lusk, and Skjellum
- *Domain Decomposition*, by Smith, Bjorstad, and Gropp