



PETSc Tutorial

Numerical Software Libraries for the Scalable Solution of PDEs

Satish Balay, Kris Buschelman, Bill Gropp,
Dinesh Kaushik, Matt Knepley,
Lois Curfman McInnes, Barry Smith, Hong Zhang

Mathematics and Computer Science Division
Argonne National Laboratory

<http://www.mcs.anl.gov/petsc>

Intended for use with version 2.1.3 of PETSc

Tutorial Objectives

- Introduce the Portable, Extensible Toolkit for Scientific Computation (PETSc)
- Demonstrate how to write a complete parallel implicit PDE solver using PETSc
- Introduce PETSc interfaces to other software packages
- Explain how to learn more about PETSc

The Role of PETSc

- Developing parallel, non-trivial 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.

What is PETSc?

- A freely available and supported research code
 - Available via <http://www.mcs.anl.gov/petsc>
 - 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
 - Usable from Fortran 77/90, C, and C++
- Portable to any parallel system supporting MPI, including
 - Tightly coupled systems
 - Cray T3E, SGI Origin, IBM SP, HP 9000, Sun Enterprise
 - Loosely coupled systems, e.g., networks of workstations
 - Compaq, HP, IBM, SGI, Sun
 - PCs running Linux or Windows
- PETSc history
 - Begun in September 1991
 - Now: over 8,500 downloads since 1995 (versions 2.0 and 2.1)
- PETSc funding and support
 - Department of Energy: MICS Program, DOE2000, SciDAC
 - National Science Foundation, Multidisciplinary Challenge Program, CISE

The PETSc Team



Satish
Balay



Matt
Knepley



Kris
Buschelman



Lois
Curfman
McInnes



Bill
Gropp



Barry
Smith



Dinesh
Kaushik



Hong
Zhang

PETSc Concepts

- How to specify the mathematics of the problem
 - Data objects
 - vectors, matrices
- How to solve the problem
 - Solvers
 - linear, nonlinear, and time stepping (ODE) solvers
- Parallel computing complications
 - Parallel data layout
 - structured and unstructured meshes

Tutorial Topics (short course)

8:30-8:50 Lois
8:50-9:20 Satish
Demo
9:20-10:00 Lois
Demo
Break

Getting started

- motivating examples
- programming paradigm

Data objects

- vectors (e.g., field variables)
- matrices (e.g., sparse Jacobians)

Viewers

- object information
- visualization

Solvers

- Linear

Profiling and performance tuning

10:20-10:45 Lois
10:45-11:45 Satish
Demo
11:45-12:10 Lois

Solvers (cont.)

- nonlinear
- timestepping (and ODEs)

Data layout and ghost values

- structured and unstructured mesh problems

Putting it all together

- a complete example

Debugging and error handling

New features

Using PETSc with other software packages

Tutorial Topics (Long Course)

- **Getting started**
 - motivating examples
 - programming paradigm
- **Data objects**
 - vectors (e.g., field variables)
 - matrices (e.g., sparse Jacobians)
- **Viewers**
 - object information
 - visualization
- **Solvers**
 - Linear
- **Profiling and performance tuning**
- **Solvers (cont.)**
 - nonlinear
 - timestepping (and ODEs)
- **Data layout and ghost values**
 - structured and unstructured mesh problems
- **Putting it all together**
 - a complete example
- **Debugging and error handling**
- **New features**
- **Using PETSc with other software packages**

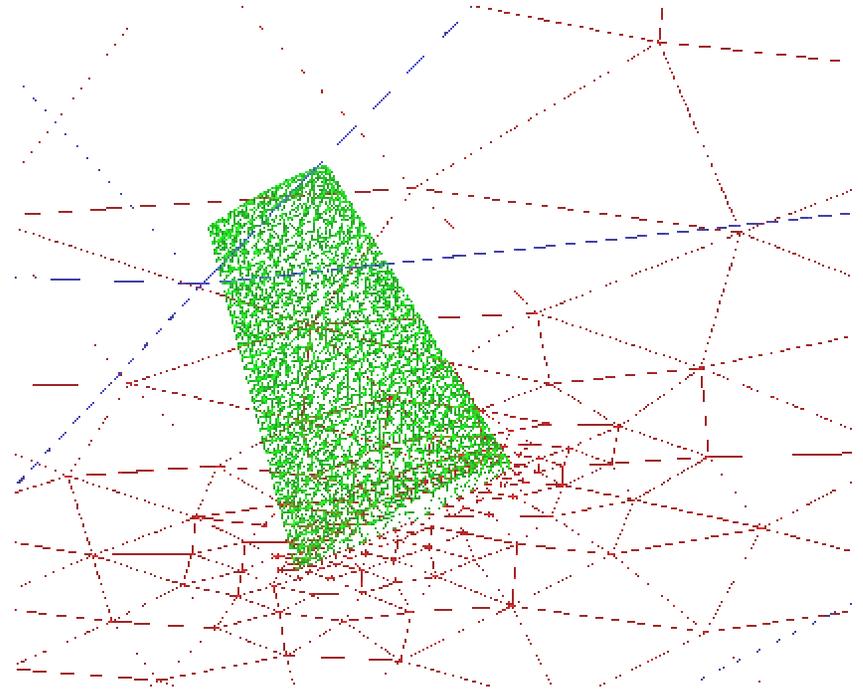
Tutorial Topics:

Using PETSc with Other Packages

- Linear solvers
 - AMG <http://www.mgnet.org/mgnet-codes-gmd.html>
 - BlockSolve95 <http://www.mcs.anl.gov/BlockSolve95>
 - ILUTP <http://www.cs.umn.edu/~saad/>
 - LUSOL <http://www.sbsi-sol-optimize.com>
 - SPAI <http://www.sam.math.ethz.ch/~grote/spai>
 - SuperLU <http://www.nersc.gov/~xiaoye/SuperLU>
- Mesh and discretization tools
 - Overture <http://www.llnl.gov/CASC/Overture>
 - SAMRAI <http://www.llnl.gov/CASC/SAMRAI>
 - SUMAA3d <http://www.mcs.anl.gov/sumaa3d>
- Optimization software
 - TAO <http://www.mcs.anl.gov/tao>
 - Veltisto <http://www.cs.nyu.edu/~biros/veltisto>
- ODE solvers
 - PVODE <http://www.llnl.gov/CASC/PVODE>
- Others
 - Matlab <http://www.mathworks.com>
 - ParMETIS <http://www.cs.umn.edu/~karypis/metis/parmetis>

CFD on an Unstructured Mesh

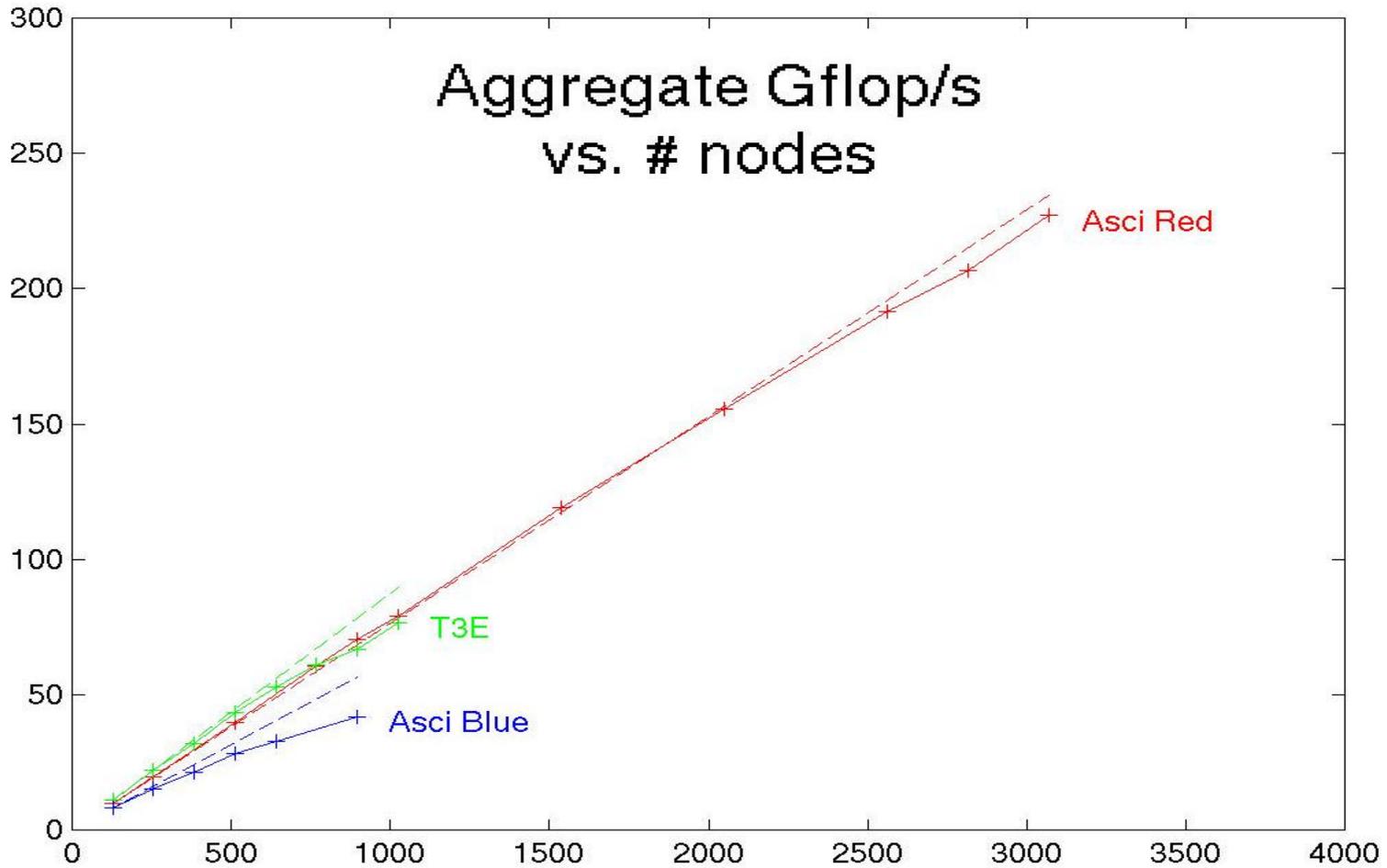
- 3D incompressible Euler
- Tetrahedral grid
- Up to 11 million unknowns
- Based on a legacy NASA code, FUN3d, developed by W. K. Anderson
- Fully implicit steady-state
- Primary PETSc tools: nonlinear solvers (SNES) and vector scatters (VecScatter)



Results courtesy of Dinesh Kaushik and David Keyes, Old Dominion Univ., partially funded by NSF and ASCI level 2 grant

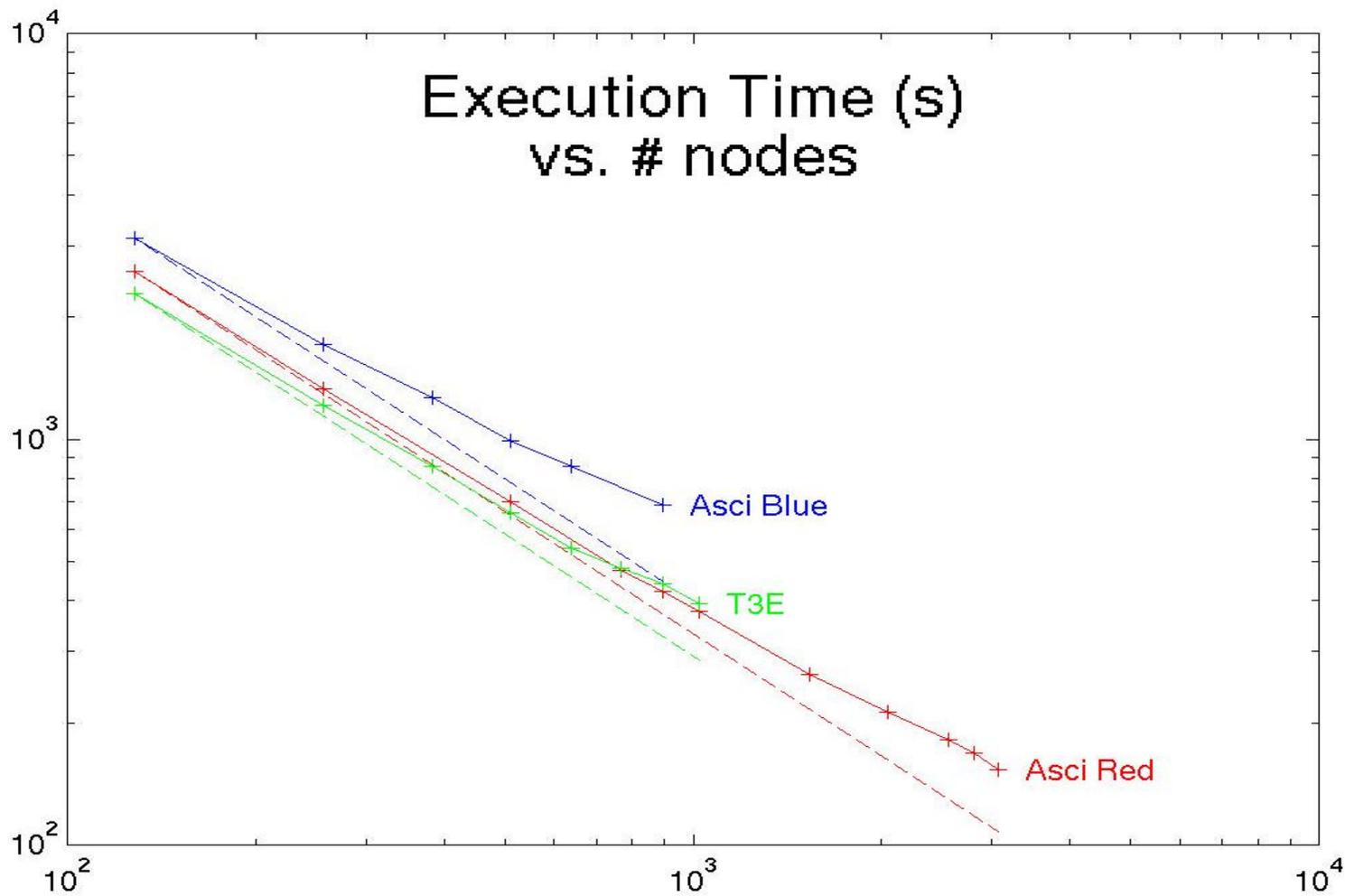
Fixed-size Parallel Scaling Results (GFlop/s)

Dimension=11,047,096



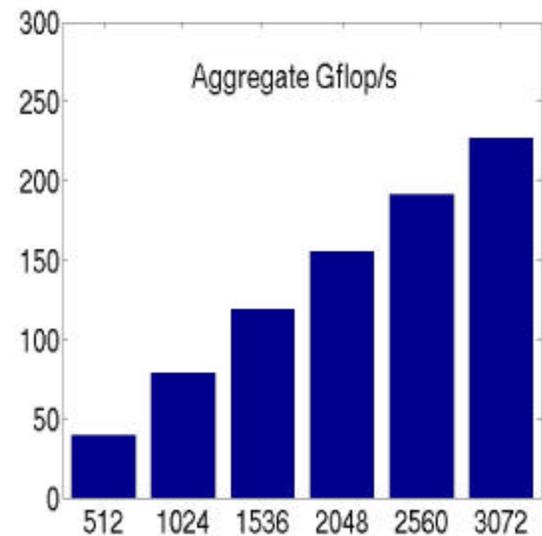
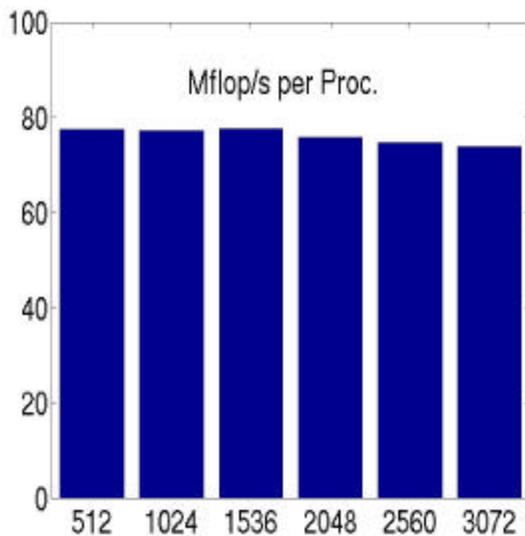
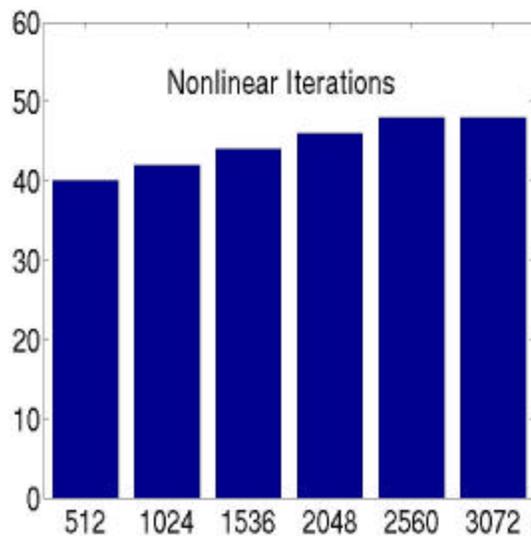
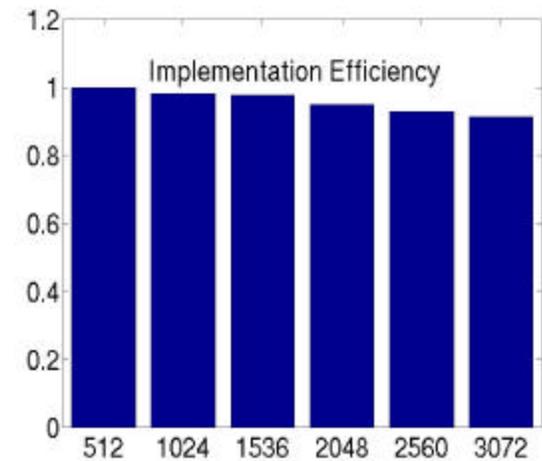
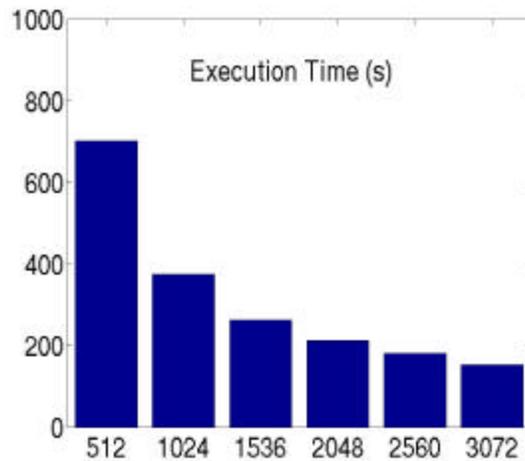
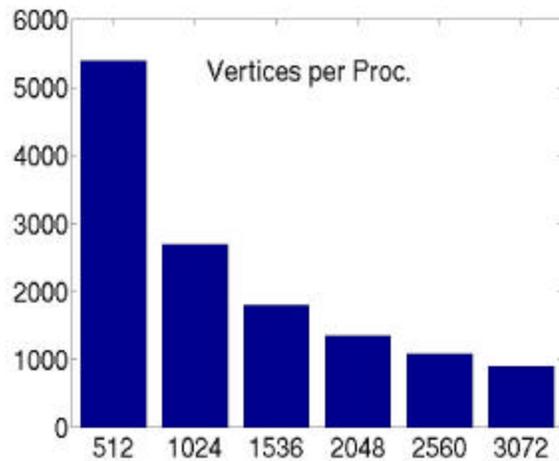
Fixed-size Parallel Scaling Results

(Time in seconds)



Inside the Parallel Scaling Results on ASCI Red

ONERA M6 wing test case, tetrahedral grid of 2.8 million vertices (about 11 million unknowns)
 on up to 3072 ASCI Red nodes (each with dual Pentium Pro 333 MHz processors)



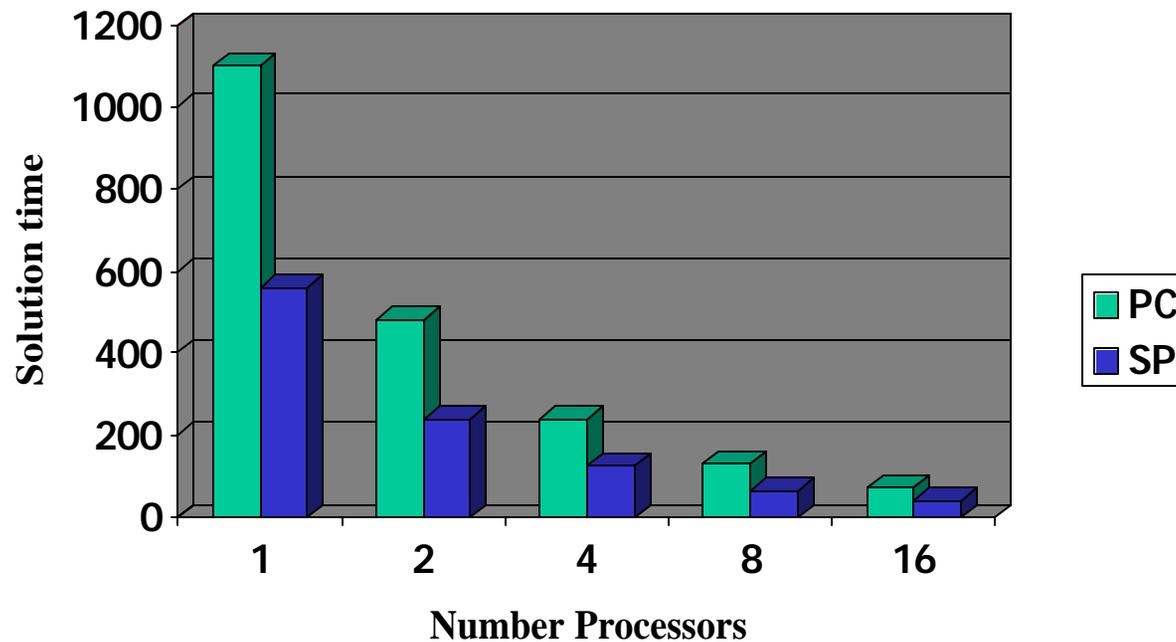
Multiphase Flow

- Oil reservoir simulation: fully implicit, time-dependent
- First fully implicit, parallel compositional simulator
- 3D EOS model (8 DoF per cell)
- Structured Cartesian mesh
- Over 4 million cell blocks, 32 million DoF
- Primary PETSc tools: linear solvers (SLES)
 - restarted GMRES with Block Jacobi preconditioning
 - Point-block ILU(0) on each process
- Over 10.6 gigaflops sustained performance on 128 nodes of an IBM SP. 90+ percent parallel efficiency

Results courtesy of collaborators Peng Wang and Jason Abate, Univ. of Texas at Austin, partially funded by DOE ER FE/MICS

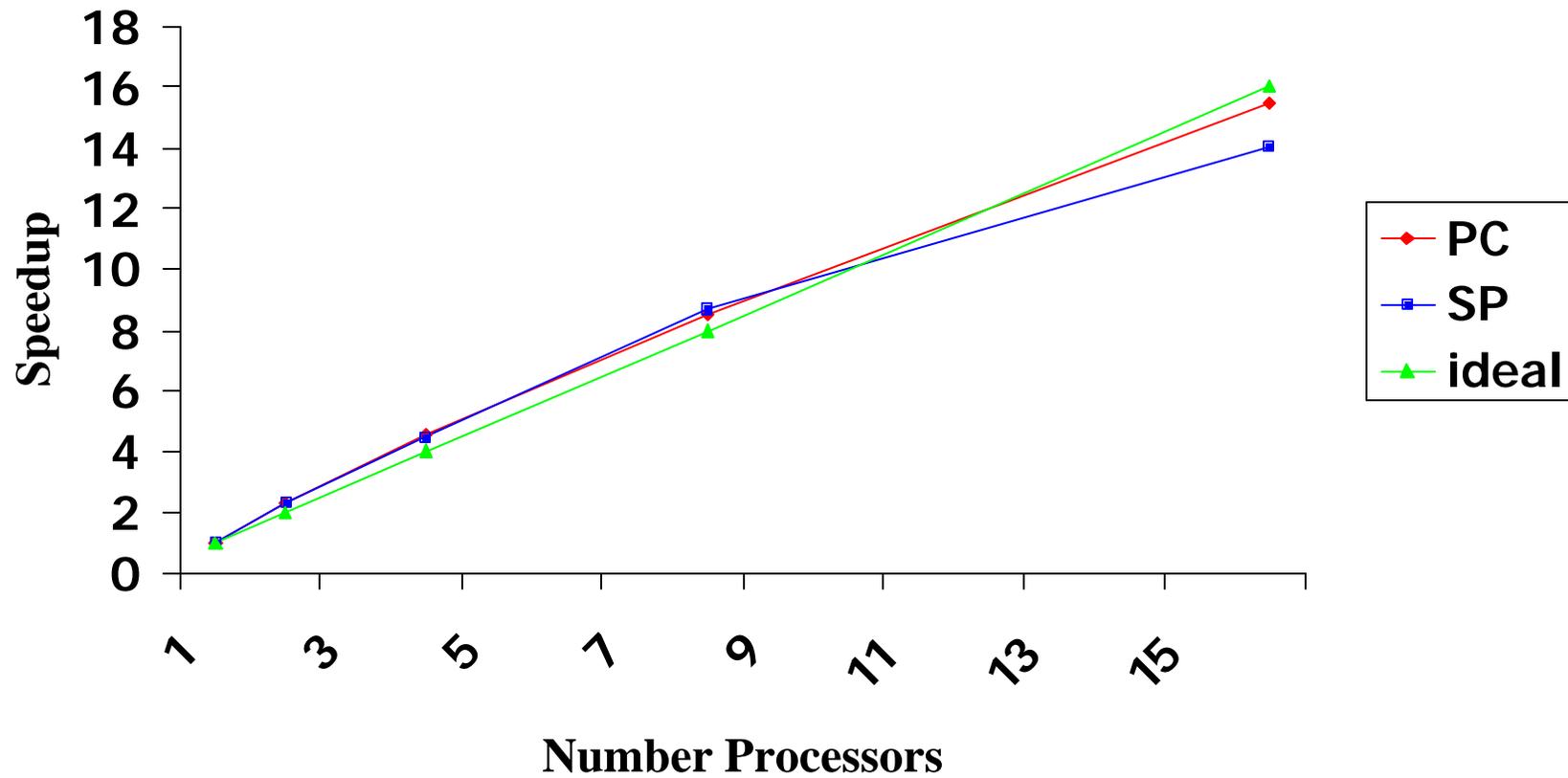
PC and SP Comparison

179,000 unknowns (22,375 cell blocks)



- **PC**: Fast ethernet (100 Megabits/second) network of 300 Mhz Pentium PCs with 66 Mhz bus
- **SP**: 128 node IBM SP with 160 MHz Power2super processors and 2 memory cards

Speedup Comparison

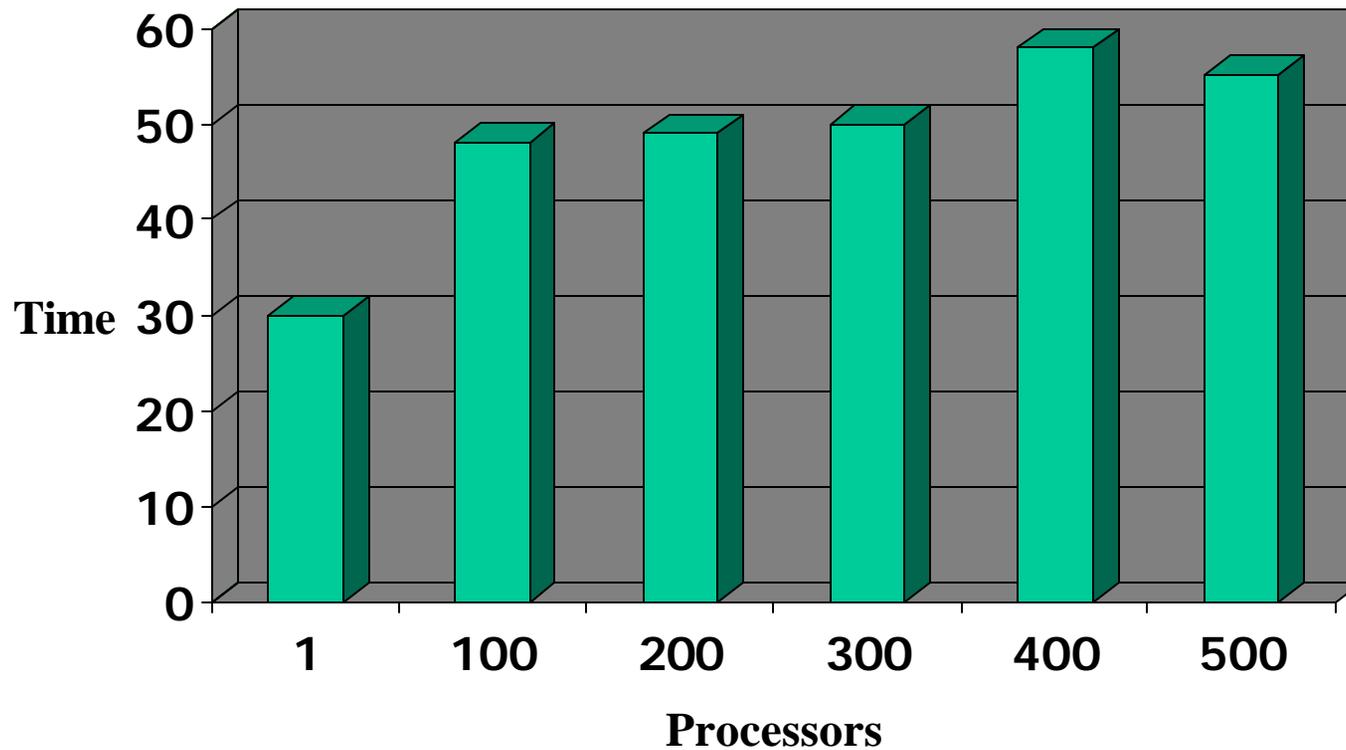


Structures Simulations

- ALE3D (LLNL structures code) test problems
- Simulation with over 16 million degrees of freedom
- Run on NERSC 512 processor T3E and LLNL ASCI Blue Pacific
- Primary PETSc tools: multigrid linear solvers (SLES)

Results courtesy of Mark Adams (Univ. of California, Berkeley)

ALE3D Test Problem Performance



NERSC Cray T3E Scaled Performance
 15,000 DoF per processor

Tutorial Approach

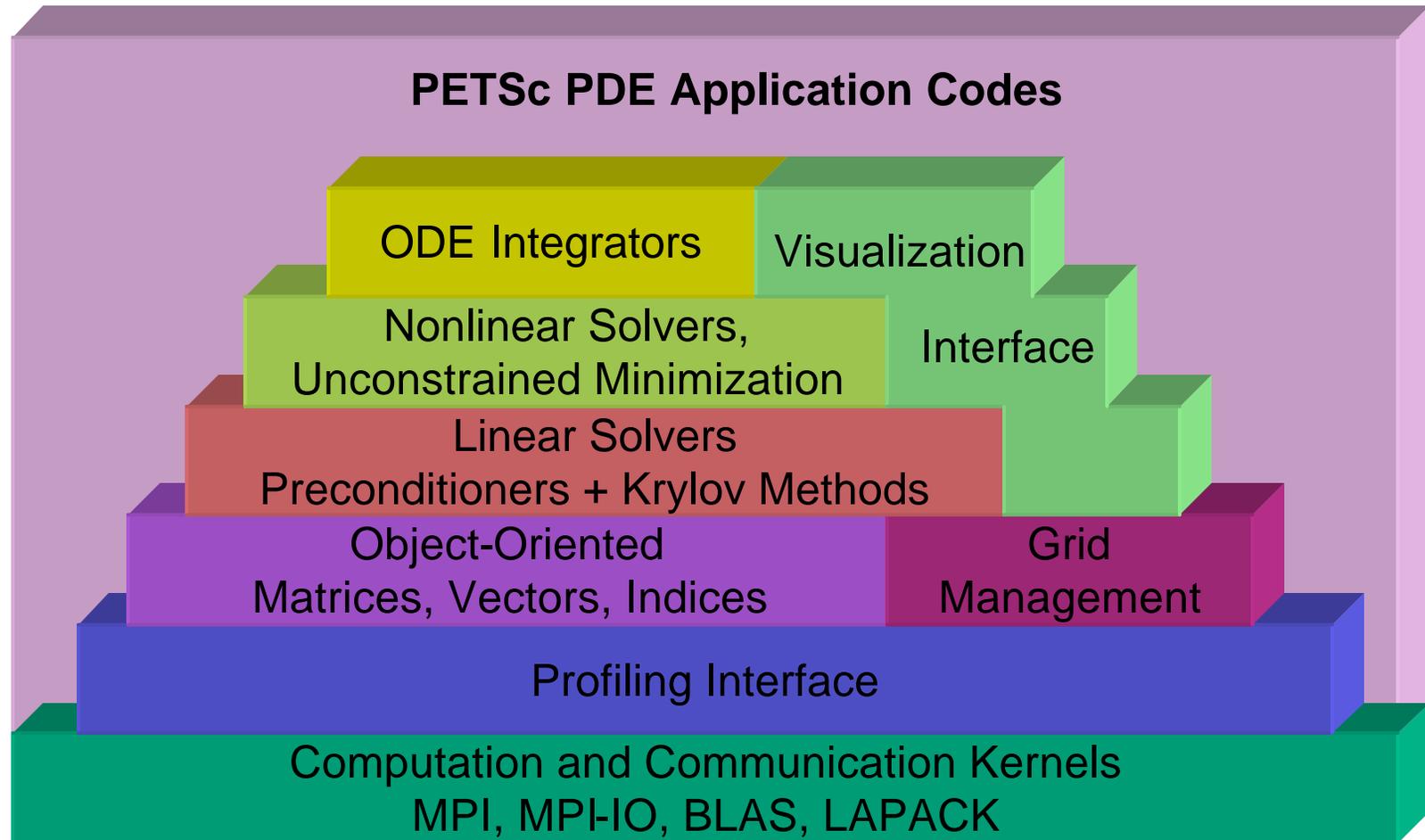
From the perspective of an application programmer:

<ul style="list-style-type: none"> • Beginner <ul style="list-style-type: none"> – basic functionality, intended for use by most programmers <div style="text-align: center;">  <div style="border: 1px solid black; padding: 2px; display: inline-block;">beginner</div> </div>	<ul style="list-style-type: none"> • Advanced <ul style="list-style-type: none"> – user-defined customization of algorithms and data structures <div style="text-align: center;">  <div style="border: 1px solid black; padding: 2px; display: inline-block;">advanced</div> </div>
<ul style="list-style-type: none"> • Intermediate <ul style="list-style-type: none"> – selecting options, performance evaluation and tuning <div style="text-align: center;">  <div style="border: 1px solid black; padding: 2px; display: inline-block;">intermediate</div> </div>	<ul style="list-style-type: none"> • Developer <ul style="list-style-type: none"> – advanced customizations, intended primarily for use by library developers <div style="text-align: center;">  <div style="border: 1px solid black; padding: 2px; display: inline-block;">developer</div> </div>

Incremental Application Improvement

- Beginner
 - Get the application “up and walking”
- Intermediate
 - Experiment with options
 - Determine opportunities for improvement
- Advanced
 - Extend algorithms and/or data structures *as needed*
- Developer
 - Consider interface and efficiency issues for integration and interoperability of multiple toolkits
- Full tutorials available at
<http://www.mcs.anl.gov/petsc/docs/tutorials>

Structure of PETSc



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

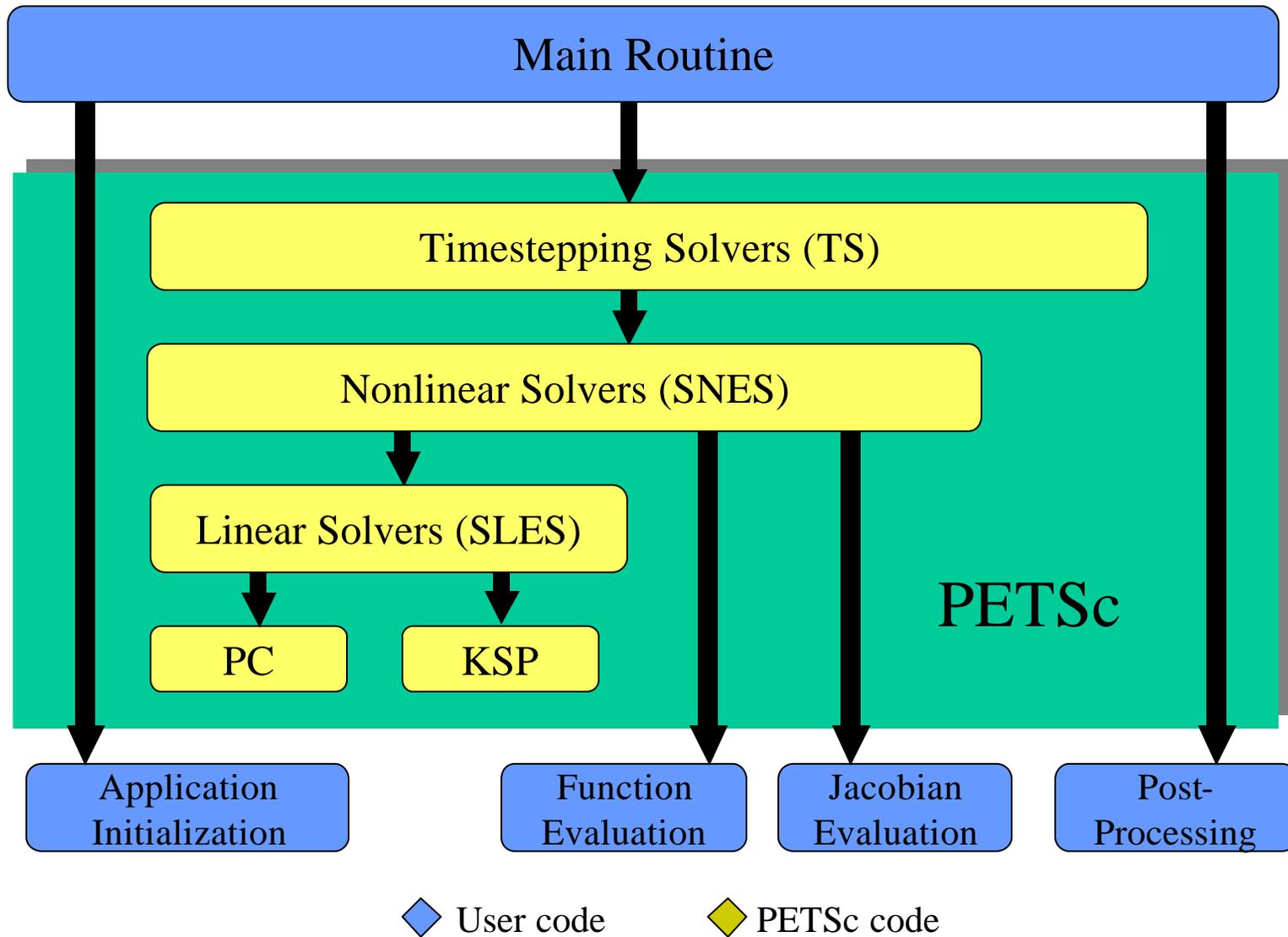
Index Sets			
Indices	Block Indices	Stride	Other

What is not in PETSc?

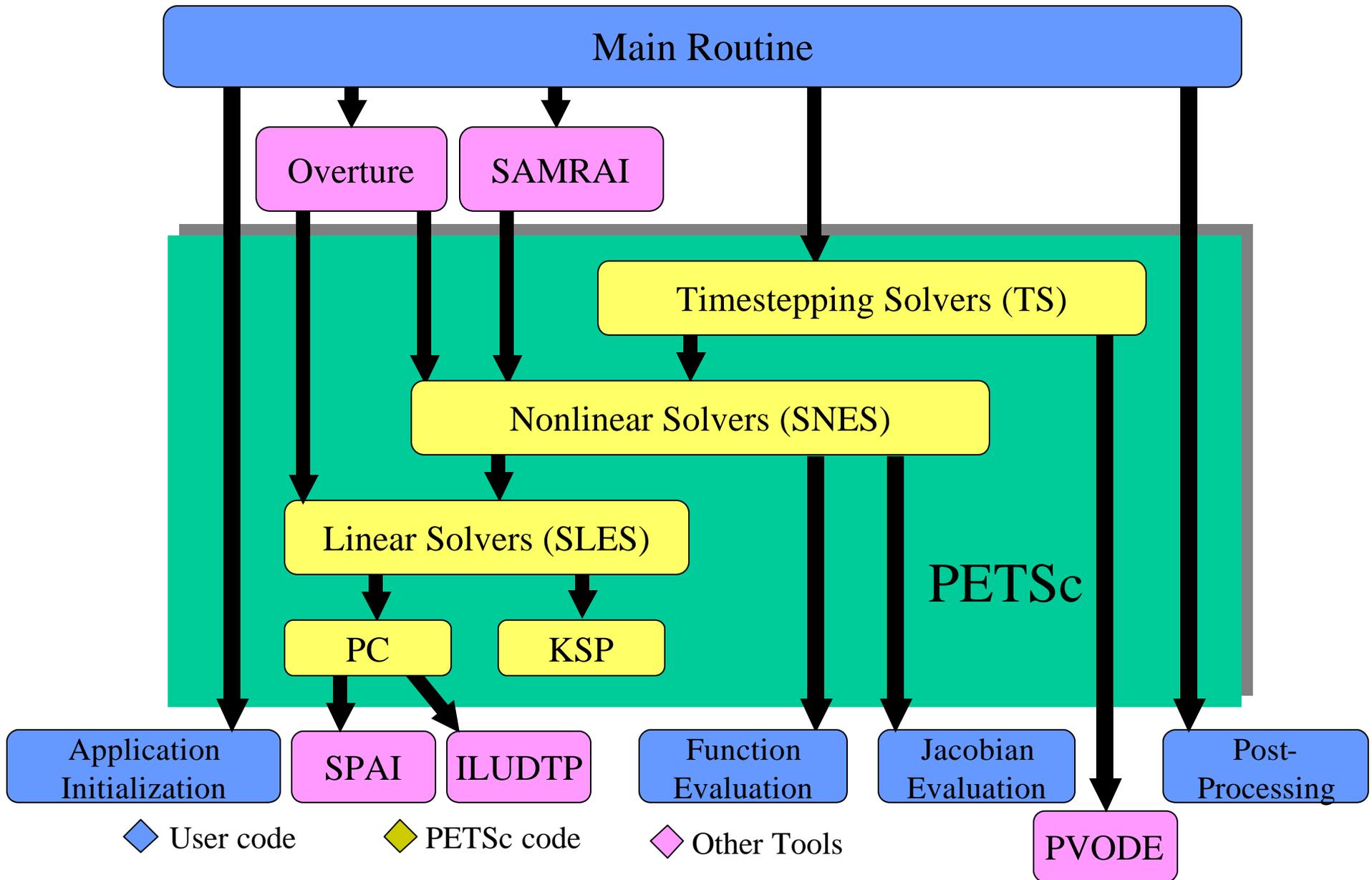
- Discretizations
- Unstructured mesh generation and refinement tools
- Load balancing tools
- Sophisticated visualization capabilities

But PETSc does interface to external software that provides some of this functionality.

Flow of Control for PDE Solution



Flow of Control for PDE Solution



Levels of Abstraction in Mathematical Software

- Application-specific interface
 - Programmer manipulates objects associated with the application
- High-level mathematics interface
 - Programmer manipulates mathematical objects, such as PDEs and boundary conditions
- Algorithmic and discrete mathematics interface
 - Programmer manipulates mathematical objects (sparse matrices, nonlinear equations), algorithmic objects (solvers) and discrete geometry (meshes)
- Low-level computational kernels
 - e.g., BLAS-type operations



*PETSc
emphasis*

Solver Definitions: For Our Purposes

- **Explicit:** Field variables are updated using neighbor information (no global linear or nonlinear solves)
- **Semi-implicit:** Some subsets of variables (e.g., pressure) are updated with global solves
- **Implicit:** Most or all variables are updated in a single global linear or nonlinear solve

Focus On Implicit Methods

- Explicit and semi-explicit are easier cases
- No direct PETSc support for
 - ADI-type schemes
 - spectral methods
 - particle-type methods

Numerical Methods Paradigm

- Encapsulate the latest numerical algorithms in a consistent, application-friendly manner
- Use mathematical and algorithmic objects, not low-level programming language objects
- Application code focuses on mathematics of the global problem, not parallel programming details

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

The PETSc Programming Model

- **Goals**
 - Portable, runs everywhere
 - 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 exploit “compiler discovered” parallelism on each node (e.g., SMP)
 - Hide within parallel objects the details of the communication
 - User orchestrates communication at a higher abstract level than message passing

Collectivity

- MPI communicators (MPI_Comm) specify collectivity (processes involved in a computation)
- All PETSc routines for creating solver and data objects are collective with respect to a communicator, e.g.,
 - `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, e.g.,
 - collective: `VecNorm()`
 - not collective: `VecGetLocalSize()`
- If a sequence of collective routines is used, they **must** be called in the same order by each process.

Hello World

```
#include "petsc.h"
int main( int argc, char *argv[] )
{
    PetscInitialize(&argc,&argv,PETSC_NULL,PETSC_NULL);
    PetscPrintf(PETSC_COMM_WORLD,"Hello World\n");
    PetscFinalize();
    return 0;
}
```

Hello World (Fortran)

```
program main
integer ierr, rank
#include "include/finclude/petsc.h"
call PetscInitialize( PETSC_NULL_CHARACTER, ierr )
call MPI_Comm_rank( PETSC_COMM_WORLD, rank, ierr )
if (rank .eq. 0) then
    print *, 'Hello World'
endif
call PetscFinalize(ierr)
end
```

Fancier Hello World

```
#include "petsc.h"
int main( int argc, char *argv[] )
{
    int rank;
    PetscInitialize(&argc,&argv,PETSC_NULL,PETSC_NULL);
    MPI_Comm_rank(PETSC_COMM_WORLD,&rank );
    PetscSynchronizedPrintf(PETSC_COMM_WORLD,
                           "Hello World from %d\n",rank);
    PetscSynchronizedFlush(PETSC_COMM_WORLD);
    PetscFinalize();
    return 0;
}
```

Data Objects

- Vectors (Vec)
 - focus: field data arising in nonlinear PDEs
- Matrices (Mat)
 - focus: linear operators arising in nonlinear PDEs (i.e., Jacobians)

beginner

beginner

intermediate

intermediate

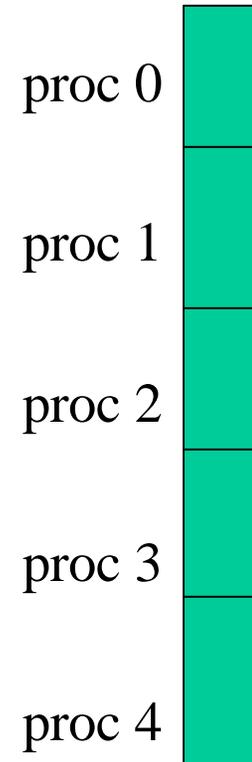
advanced

- Object creation
- Object assembly
- Setting options
- Viewing
- User-defined customizations

tutorial outline:
data objects

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*



beginner

data objects:
vectors

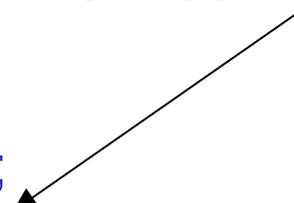
Creating a vector

```

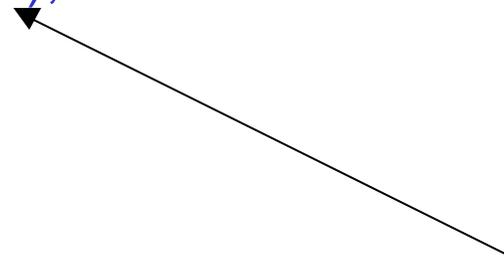
Vec x;
int n;
...
PetscInitialize(&argc,&argv,(char*)0,help);
PetscOptionsGetInt(PETSC_NULL,"-n",&n,PETSC_NULL);
...
VecCreate(PETSC_COMM_WORLD,&x);
VecSetSizes(x,PETSC_DECIDE,n);
VecSetType(x,VEC_MPI);
VecSetFromOptions(x);

```

Use PETSc to get value from command line



Global size



PETSc determines local size

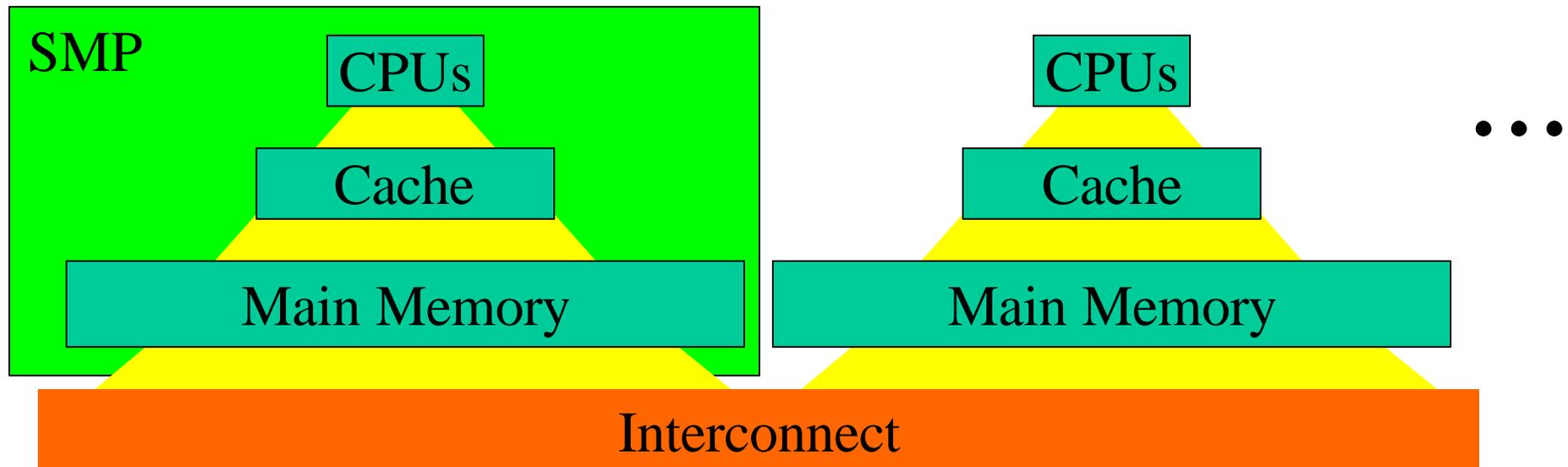


How Can We Use a PETSc Vector

- In order to support the distributed memory “shared nothing” model, as well as single processors and shared memory systems, a 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 want to *require* that the programmer work only with the “local” part of the vector; we want to permit operations, such as setting an element of a vector, to be performed by any process.
- Recall how data is stored in the distributed memory programming model...

Sample Parallel System Architecture

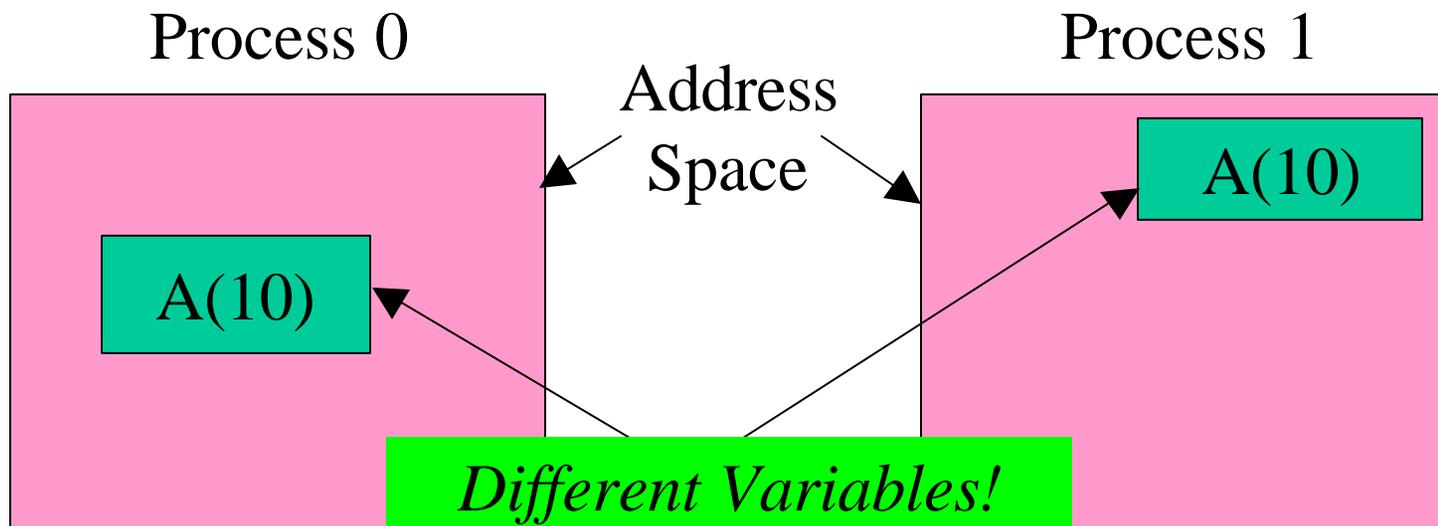
- Systems have an increasingly deep memory hierarchy (1, 2, 3, and more levels of cache)
- Time to reference main memory 100's of cycles
- Access to shared data requires synchronization
 - Better to ensure data is local and unshared when possible



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
 - 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*

Distributed Memory Model



- Integer A(10)

...

print *, A



- Integer A(10)

```
do i=1,10
  A(i) = i
enddo
```

...



This A is completely different from this one

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)

beginner

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**

beginner

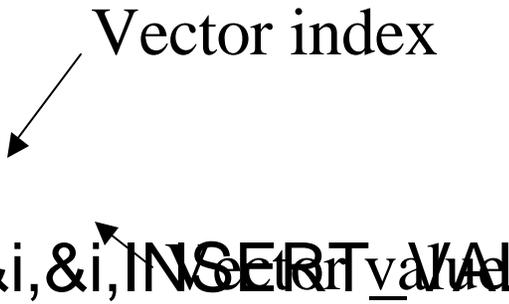
data objects:
vectors and
matrices

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);
}
/* These two routines ensure that the data is
distributed to the other processes */
VecAssemblyBegin(x);
VecAssemblyEnd(x);

```



# 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);

# 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\ $       |

# Simple Example Programs

Location: `petsc/src/sys/examples/tutorials/`

**E** `ex2.c` - synchronized printing 

Location: `petsc/src/vec/examples/tutorials/`

**E** `ex1.c, ex1f.F, ex1f90.F` - basic vector routines   
**E** `ex3.c, ex3f.F` - parallel vector layout



beginner

*And many more examples ...*

**E** - on-line exercise

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;
}
```

# 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 are done with it
  - `VecRestoreArray( Vec, double *[] )`
- This allows PETSc to handle and data structure conversions
  - For most common uses, these routines are inexpensive and do *not* involve a copy of the vector.

# 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);
```

# 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*.

# 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 *properties*, the operations that you can perform with it.
  - Not by its data structures

beginner

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)**

# Matrix Assembly Example

simple 3-point stencil for 1D discretization

```

Mat A;
int column[3], i;
double value[3];
...
MatCreate(PETSC_COMM_WORLD,
 PETSC_DECIDE,PETSC_DECIDE,n,n,&A);
MatSetFromOptions(A);
/* mesh interior */
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
if (rank == 0) { /* Only one process creates matrix */
 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

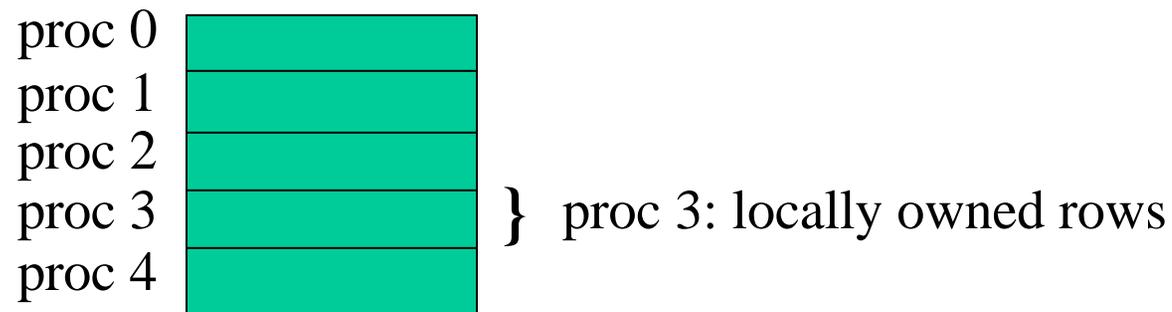
Let PETSc decide how  
to allocate matrix  
across processes

beginner

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

# 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,
 PETSC_DECIDE, PETSC_DECIDE, n, n, &A);
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++) {
 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

beginner

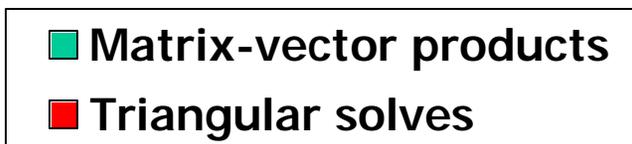
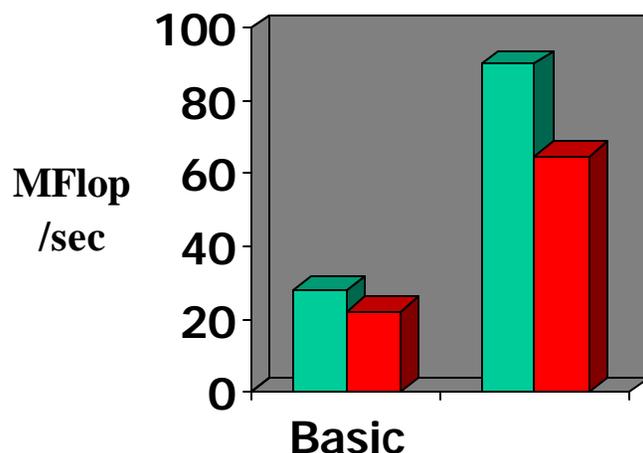
data objects:  
matrices

# 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 is simple and makes it easier to work with other codes.
  - For applications with other ordering needs, PETSc provides “Application Orderings” (AO), described later.

# Blocking: Performance Benefits

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



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

beginner

data objects:  
matrices

# Viewers

beginner

- Printing information about solver and data objects

beginner

- Visualization of field and matrix data

intermediate

- Binary output of vector and matrix data

tutorial outline:  
viewers

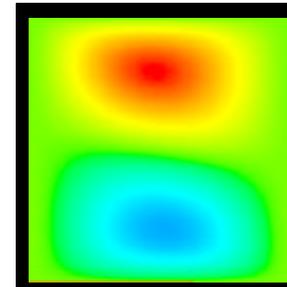
# Viewer Concepts

- Information about PETSc objects
  - runtime choices for solvers, nonzero info for matrices, etc.
- Data for later use in restarts or external tools
  - vector fields, matrix contents
  - various formats (ASCII, binary)
- Visualization
  - *simple* x-window graphics
    - vector fields
    - matrix sparsity structure

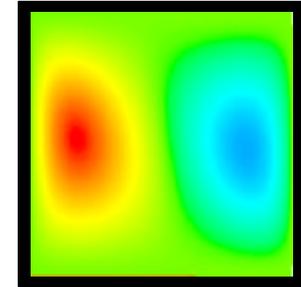
# Viewing Vector Fields

- `VecView(Vec x, PetscViewer v);`
- Default viewers
  - ASCII (sequential):  
`PETSC_VIEWER_STDOUT_SELF`
  - ASCII (parallel):  
`PETSC_VIEWER_STDOUT_WORLD`
  - X-windows:  
`PETSC_VIEWER_DRAW_WORLD`
- Default ASCII formats
  - `PETSC_VIEWER_ASCII_DEFAULT`
  - `PETSC_VIEWER_ASCII_MATLAB`
  - `PETSC_VIEWER_ASCII_COMMON`
  - `PETSC_VIEWER_ASCII_INFO`
  - etc.

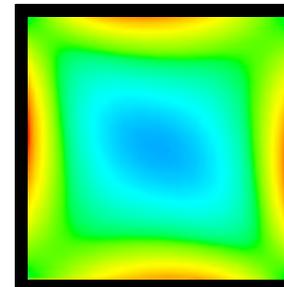
Solution components,  
using runtime option  
`-snes_vecmonitor`



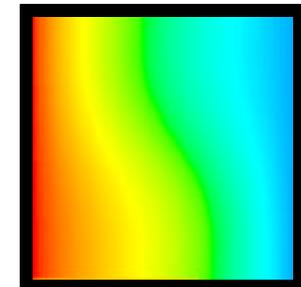
velocity:  $u$



velocity:  $v$



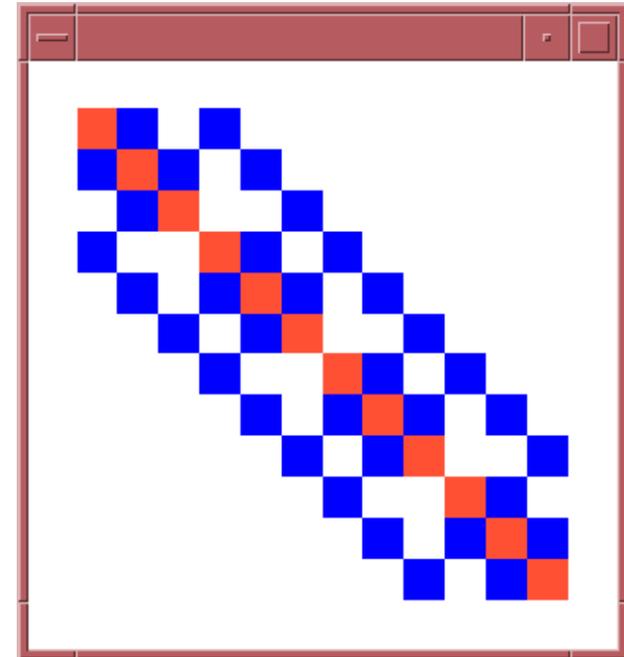
vorticity: ?



temperature:  $T$

# Viewing Matrix Data

- `MatView(Mat A, PetscViewer v);`
- Runtime options available after matrix assembly
  - `-mat_view_info`
    - info about matrix assembly
  - `-mat_view_draw`
    - sparsity structure
  - `-mat_view`
    - data in ASCII
  - etc.



# More Viewer Info

- Viewer creation
  - `ViewerASCIIOpen()`
  - `ViewerDrawOpen()`
  - `ViewerSocketOpen()`
- Binary I/O of vectors and matrices
  - output: `ViewerBinaryOpen()`
  - input: `VecLoad()`, `MatLoad()`

# Running PETSc Programs

- The easiest approach is to use the PETSc Makefile provided with the examples
  - Ensures that all of the correct libraries are used and that the correct compilation options are used.
- 1. Make sure that you have your PETSc environment setup correctly:
  - `setenv PETSC_DIR <directory with PETSc installed>`
  - `setenv PETSC_ARCH <name of PETSc architecture>`
- 2. Copy an example file and the makefile from the directory to your local directory:
  - `cp ${PETSC_DIR}/src/vec/examples/tutorials/ex1.c .`
  - `cp ${PETSC_DIR}/src/vec/examples/tutorials/makefile .`
- 3. Make and run the program:
  - `make BOPT=g ex1`
  - `mpirun -np 2 ex1`
- 4. Edit the example to explore PETSc

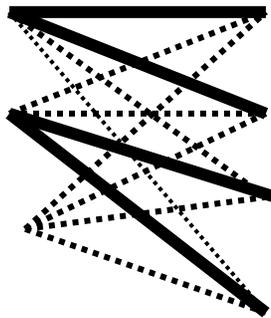


# End of Day 1

# Solvers: Usage Concepts

## Solver Classes

- Linear (SLES)
- Nonlinear (SNES)
- Timestepping (TS)



## Usage Concepts

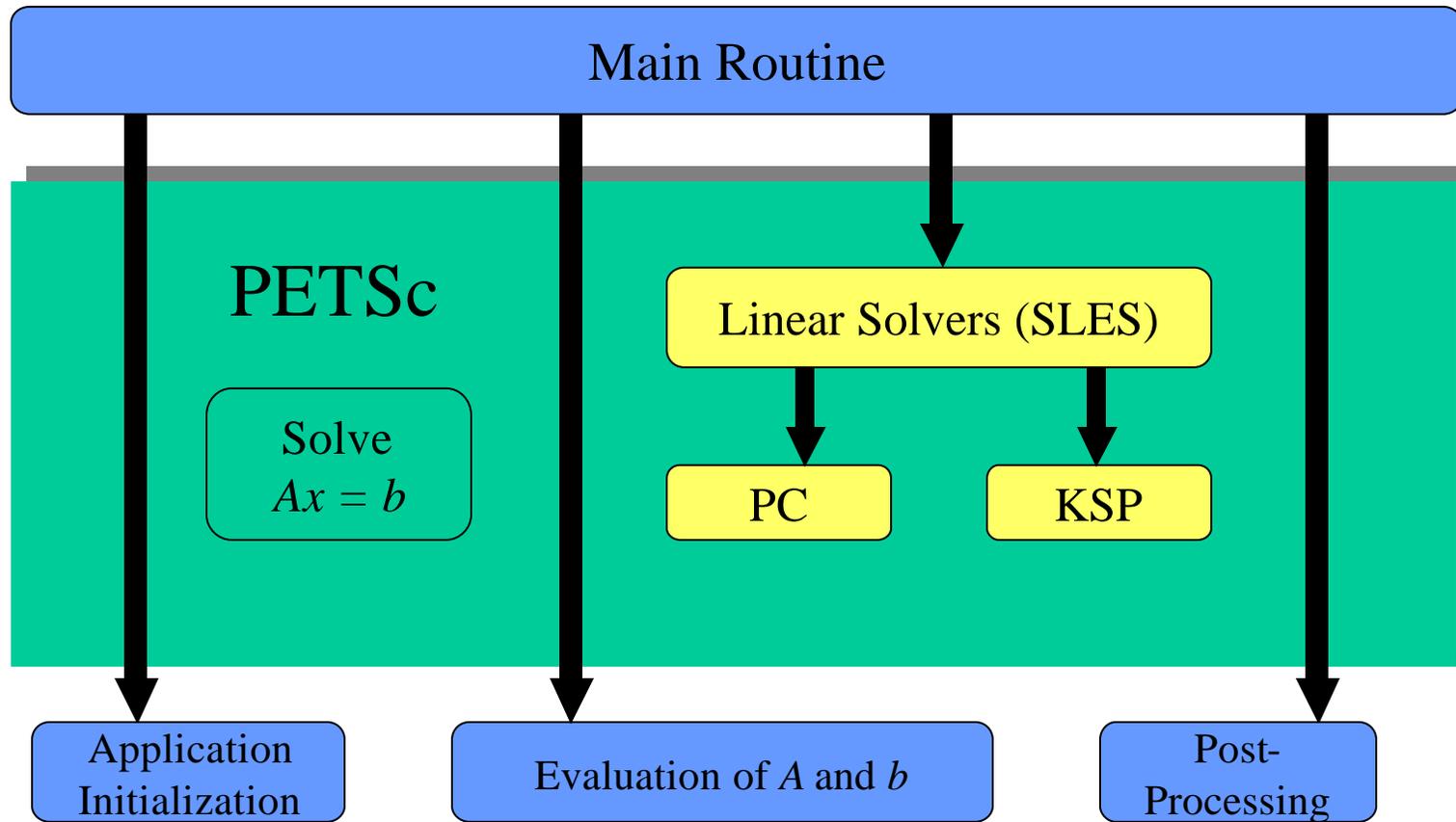
- Context variables
- Solver options
- Callback routines
- Customization



important concepts

tutorial outline:  
solvers

# Linear PDE Solution



◆ User code

◆ PETSc code

beginner

solvers:  
linear

# Linear Solvers

**Goal:** Support the solution of linear systems,

$$Ax=b,$$

particularly for sparse, parallel problems arising within PDE-based models

User provides:

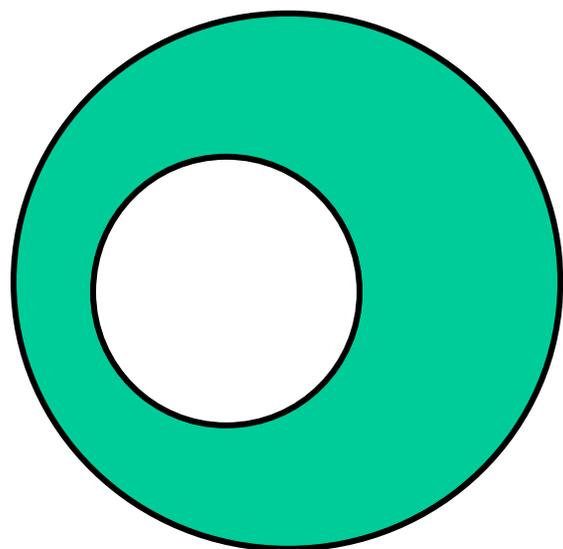
- Code to evaluate  $A$ ,  $b$

beginner

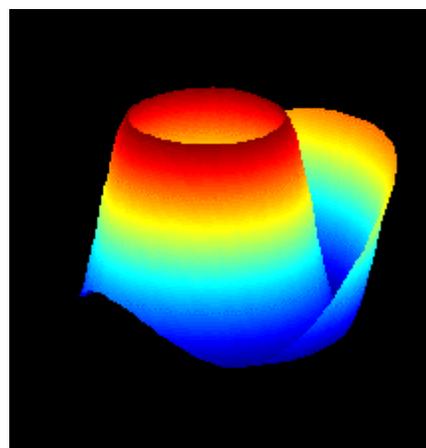
solvers:  
linear

# Sample Linear Application:

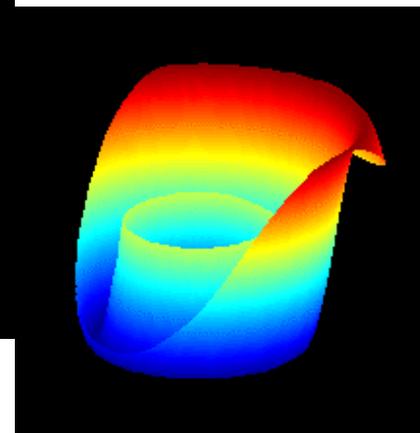
## Exterior Helmholtz Problem



### Solution Components



Real



Imaginary

$$\Delta u - k^2 u = 0$$

$$\lim_{r \rightarrow \infty} r^{1/2} \left( \frac{\partial u}{\partial r} - iku \right) = 0$$

Collaborators: *H. M. Atassi, D. E. Keyes,*

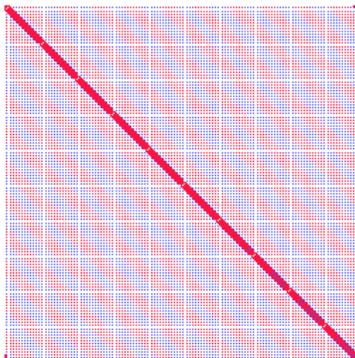
*L. C. McInnes, R. Susan-Resiga*

beginner

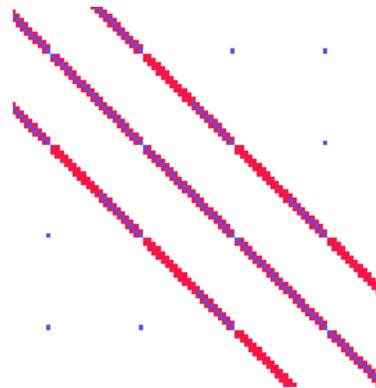
solvers:  
linear

# Helmholtz: The Linear System

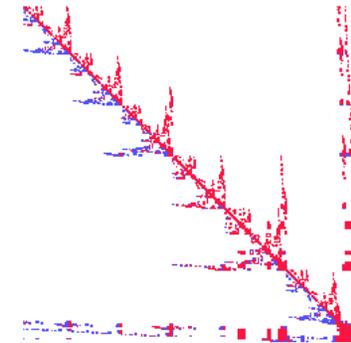
- Logically regular grid, parallelized with DAs
- Finite element discretization (bilinear quads)
- Nonreflecting exterior BC (via DtN map)
- Matrix sparsity structure (option: `-mat_view_draw`)



Natural ordering



Close-up



Nested dissection ordering

beginner

solvers:  
linear

# Linear Solvers (SLES)

*SLES: Scalable Linear Equations Solvers*

beginner

beginner

beginner

beginner

intermediate

intermediate

advanced

advanced

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

tutorial outline:  
solvers:  
linear

# Objects In PETSc

- How should a matrix be described in a program?
  - Old way:
    - Dense matrix:
      - double precision  $A(10,10)$
    - Sparse matrix:
      - integer  $ia(11)$ ,  $ja(max\_nz)$
      - double precision  $a(max\_nx)$
  - New way:
    - Mat M
- Hides the choice of data structure
  - Of course, the library still needs to represent the matrix with some choice of data structure, but this is an *implementation detail*
- Benefit
  - Programs become independent of particular choices of data structure, making it easier to modify and adapt

# Operations In PETSc

- How should operations like “solve linear system” be described in a program?
  - Old way:
    - `mpiaijgmres( ia, ja, a, comm, x, b, nlocal, nglobal, ndir, orthmethod, convtol, &its )`
  - New way:
    - `SLESSolve( sles, b, x, &its )`
- Hides the choice of algorithm
  - Algorithms are to operations as data structures are to objects
- Benefit
  - Programs become independent of particular choices of algorithm, making it easier to explore algorithmic choices and to adapt to new methods.
- In PETSc, operations have their own handle, called a “*context variable*”

# 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)

# Creating the SLES Context

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

# SLES Structure

- Each SLES object actually contains two other objects:
  - KSP — 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 2.0

## 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++)

```
SLES sles; /* 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)*/
```

```
SLESCreate(PETSC_COMM_WORLD,&sles);
```

```
SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN);
```

```
SLESSetFromOptions(sles);
```

```
SLESSolve(sles,b,x,&its);
```

```
SLESDestroy(sles);
```

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)

```
SLES sles
Mat A
Vec x, b
integer n, its, ierr
```

```
call MatCreate(PETSC_COMM_WORLD,PETSC_DECIDE,n,n,A,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)
```

then assemble matrix and right-hand-side vector

```
call SLESCreate(PETSC_COMM_WORLD,sles,ierr)
call SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN,ierr)
call SLESSetFromOptions(sles,ierr)
call SLESSolve(sles,b,x,its,ierr)
call SLESDestroy(sles,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

# 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

beginner

2

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

- `SLESGetKSP(SLES sles,KSP *ksp)`
  - `KSPSetType(KSP ksp,KSPType type)`
  - `KSPSetTolerances(KSP ksp,PetscReal rtol, PetscReal atol,PetscReal dtol, int maxits)`
  - etc....
- `SLESGetPC(SLES sles,PC *pc)`
  - `PCSetType(PC pc,PCType)`
  - `PCASMSetOverlap(PC pc,int overlap)`
  - etc....

# Recursion: Specifying Solvers for Schwarz Preconditioner Blocks

- Specify SLES 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>

# Helmholtz: Scalability

128x512 grid, wave number = 13, IBM SP

GMRES(30)/Restricted Additive Schwarz

1 block per proc, 1-cell overlap, ILU(1) subdomain solver

| Procs | Iterations | Time (Sec) | Speedup |
|-------|------------|------------|---------|
| 1     | 221        | 163.01     | -       |
| 2     | 222        | 81.06      | 2.0     |
| 4     | 224        | 37.36      | 4.4     |
| 8     | 228        | 19.49      | 8.4     |
| 16    | 229        | 10.85      | 15.0    |
| 32    | 230        | 6.37       | 25.6    |

# SLES: Review of Basic Usage

- SLESCreate( ) - Create SLES context
- SLESSetOperators( ) - Set linear operators
- SLESSetFromOptions( ) - Set runtime solver options for [SLES, KSP, PC]
- SLESSolve( ) - Run linear solver
- SLESView( ) - View solver options actually used at runtime (alternative: `-sles_view`)
- SLESDestroy( ) - Destroy solver

# SLES: Review of Selected Preconditioner Options

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

 1

beginner

 2

intermediate

*And many more options...*

solvers: linear:  
preconditioners

# SLES: 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 | KSPGMRESSetOrthogonalization( ) | -ksp_unmodifiedgramschmidt -ksp_irorthog                                                                                             |

*And many more options...*



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:...

# SLES: 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 SLES 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)
- SLESSetOperators(SLES sles,
  - 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

# User-defined Customizations

- Restricting the available solvers
  - Customize `PCRegisterAll( )`, `KSPRegisterAll( )`
- Adding user-defined preconditioners via
  - `PCShell` preconditioner type

3

- Adding preconditioner and Krylov methods in library style
  - Method registration via `PCRegister( )`, `KSPRegister( )`
- Heavily commented example implementations
  - Jacobi preconditioner: `petsc/src/sles/pc/impls/jacobi.c`
  - Conjugate gradient: `petsc/src/sles/ksp/impls/cg/cg.c`

4

3

4

|          |           |
|----------|-----------|
| advanced | developer |
|----------|-----------|

|                    |
|--------------------|
| solvers:<br>linear |
|--------------------|

# SLES: Example Programs

Location: `petsc/src/sles/examples/tutorials/`

|                                                                                                                                                                                                                                                                                       |                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• <code>ex1.c</code>, <code>ex1f.F</code> - basic uniprocess codes</li> <li>• <b>E</b> <code>ex23.c</code> - basic parallel code</li> <li>• <code>ex11.c</code> - using complex numbers</li> </ul>                                             |    |
| <ul style="list-style-type: none"> <li>• <code>ex4.c</code> - using different linear system and preconditioner matrices</li> <li>• <code>ex9.c</code> - repeatedly solving different linear systems</li> <li>• <b>E</b> <code>ex22.c</code> - 3D Laplacian using multigrid</li> </ul> |    |
| <ul style="list-style-type: none"> <li>• <code>ex15.c</code> - setting a user-defined preconditioner</li> </ul>                                                                                                                                                                       |  |

*And many more examples ...*



beginner



intermediate



advanced

**E** - on-line exercise

solvers:  
linear

# Now What?

- If you have a running program, are you done?
  - Yes, if your program answers your question.
  - No, if you now need to run much larger problems or many more problems.
- How can you tune an application for performance?
  - PETSc approach:
    - Debug the application by focusing on the mathematically operations. The parallel matrix assembly operations are an example.
    - Once the code *works*,
      - Understand the performance
      - Identify performance problems
      - Use special PETSc routines and other techniques to optimize only the code that is underperforming

# Profiling and Performance Tuning

## Profiling:

beginner

- Integrated profiling using `-log_summary`

intermediate

- User-defined events

intermediate

- Profiling by stages of an application

## Performance Tuning:

intermediate

- Matrix optimizations

intermediate

- Application optimizations

advanced

- Algorithmic tuning

tutorial outline:  
profiling and  
performance tuning

# Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- All PETSc events are logged 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`

# User-defined Events

```
int USER_EVENT;
int user_event_flops
PetscLogEventRegister(&USER_EVENT,"User event
 name", "eventColor");
PetscLogEventBegin(USER_EVENT,0,0,0,0);
[code to monitor]
PetscLogFlops(user_evnet_flops);
PetscLogEvent End(USER_EVENT,0,0,0,0);
```

intermediate

profiling and  
performance tuning

# Nonlinear Solvers (SNES)

*SNES: Scalable Nonlinear Equations Solvers*

beginner

- Application code interface

beginner

- Choosing the solver

beginner

- Setting algorithmic options

beginner

- Viewing the solver

intermediate

- Determining and monitoring convergence

advanced

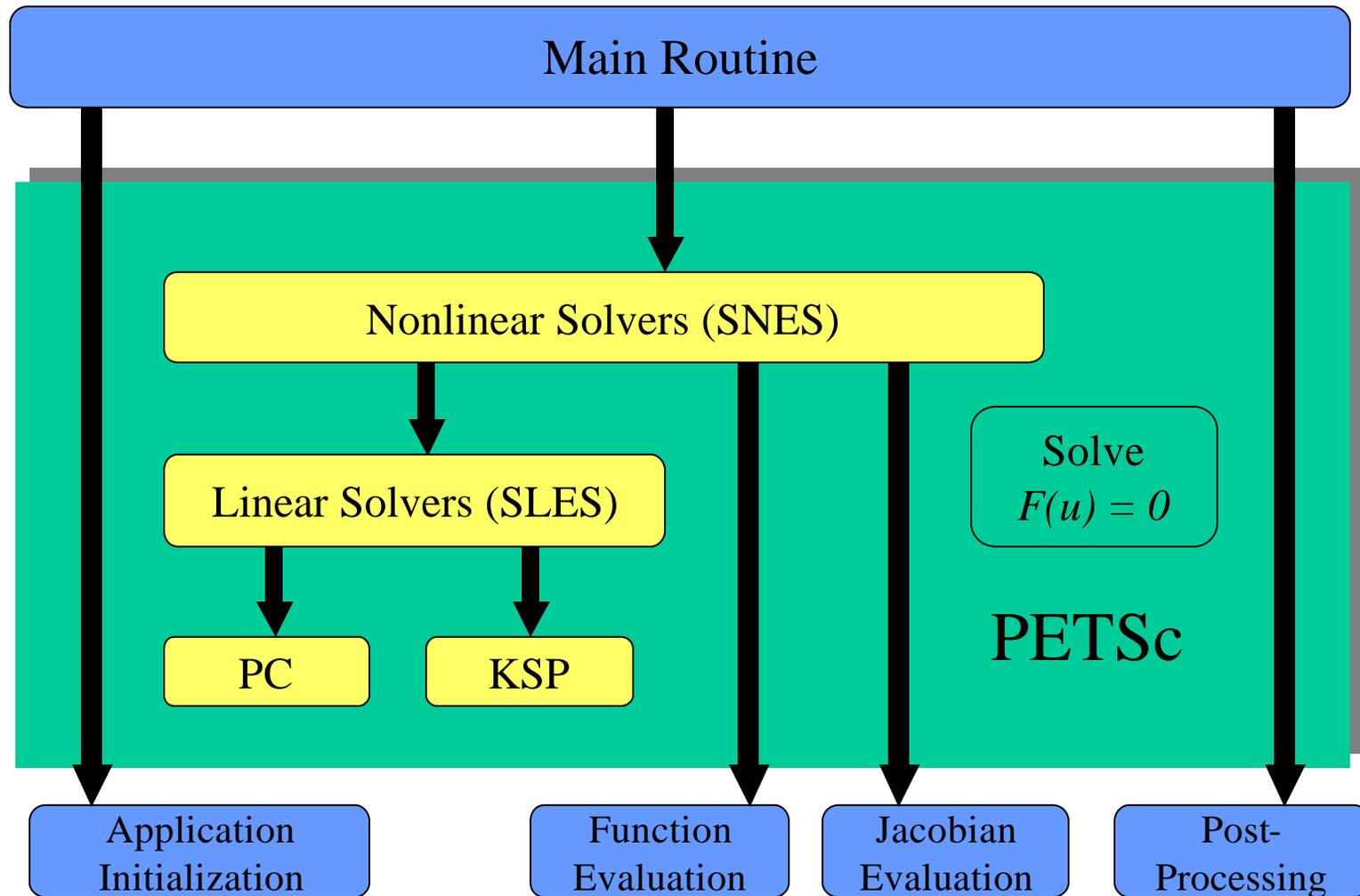
- Matrix-free solvers

advanced

- User-defined customizations

tutorial outline:  
solvers:  
nonlinear

# Nonlinear PDE Solution



◆ User code

◆ PETSc code

beginner

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>)

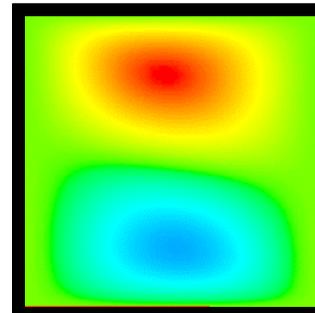
# Nonlinear Solvers (SNES)

- Newton-based methods, including
  - Line search strategies
  - Trust region approaches
  - Pseudo-transient continuation
  - Matrix-free variants
- User can customize all phases of the solution process

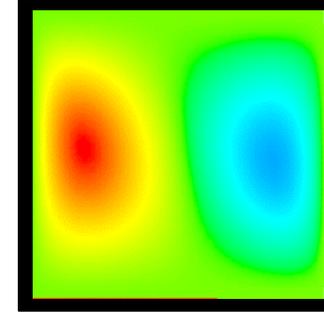
# Sample Nonlinear Application: Driven Cavity Problem

- Velocity-vorticity formulation
- Flow driven by lid and/or buoyancy
- Logically regular grid, parallelized with DAs
- Finite difference discretization
- source code:  
`petsc/src/snes/examples/tutorials/ex19.c`

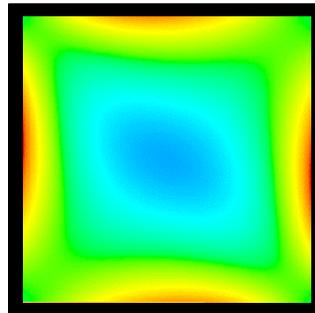
## Solution Components



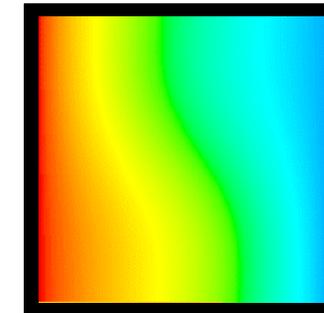
velocity:  $u$



velocity:  $v$



vorticity: ?



temperature:  $T$

beginner

*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,PETSC_DECIDE,PETSC_DECIDE,n,n,&J);
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);
```

beginner

solvers:  
nonlinear

# Basic Nonlinear Solver Code (Fortran)

```
SNES snes
Mat J
Vec x, F
int n, its
```

```
...
```

```
call MatCreate(PETSC_COMM_WORLD,PETSC_DECIDE, PETSC_DECIDE,n,n,J,ierr)
call MatSetFromOptions(J, ierr)
call VecCreate(PETSC_COMM_WORLD,x,ierr)
call VecSetSizes(x,PETSC_DECIDE,n,ierr)
call VecSetFromOptions(x,ierr)

call VecDuplicate(x,F,ierr)

call SNESCreate(PETSC_COMM_WORLD, snes,ierr)
call SNESSetFunction(snes,F,EvaluateFunction,PETSC_NULL,ierr)
call SNESSetJacobian(snes,J,J,EvaluateJacobian,PETSC_NULL,ierr)
call SNESSetFromOptions(snes,ierr)
call SNESolve(snes,x,its,ierr)
call SNESDestroy(snes,ierr)
```

beginner

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

beginner

solvers:  
nonlinear

# Sample Application Context:

## Driven Cavity Problem

```

typedef struct {
 /* ----- basic application data ----- */
 double lid_velocity, prandtl, 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;

```

beginner

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;
}
```

beginner

solvers:  
nonlinear

# Sample Local Computational Loops: Driven Cavity Problem

```

#define U(i) 4*(i)
#define V(i) 4*(i)+1
#define Omega(i) 4*(i)+2
#define Temp(i) 4*(i)+3
....
for (j = jstart; j<jend; j++) {
 row = (j - gys) * gxm + istart - gxs - 1;
 for (i = istart; i<iend; i++) {
 row++; u = x[U(row)];
 uxx = (two * u - x[U (row-1)] - x [U (row+1)]) * hydhx;
 uyy = (two * u - x[U (row-gxm)] - x [U (row+gxm)]) * hxdhy;
 f [U(row)] = uxx + uyy -
 p5 * (x [(Omega (row+gxm))] - x [Omega (row-gxm)]) * hx;
 }
}
....

```

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

beginner

solvers:  
nonlinear

# Finite Difference Jacobian Computations

- 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`

# Uniform access to all linear and nonlinear solvers

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



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




beginner



intermediate

 solvers:  
nonlinear

# Customization via Callbacks:

## Setting a user-defined line search routine

`SNESSetLineSearch(SNES snes,int(*ls)(...),void *lsctx)`  
where available line search routines `ls( )` include:

- `SNESCubicLineSearch( )` - cubic line search 2
- `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( )` 3

2

3

intermediate

advanced

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

# SNES: Review of Selected Options

| Functionality                                                                                             | Procedural Interface                                                                                           | Runtime Option                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Set nonlinear solver<br>Set monitoring routine                                                            | SNESsetType( )<br>SNESSetMonitor( )                                                                            | -snes_type [ls,tr,umls,umtr,...]<br>-snes_monitor<br>-snes_xmonitor, ...                                                                                                        |
| Set convergence tolerances<br>Set line search routine<br>View solver options<br>Set linear solver options | SNESSetTolerances( )<br>SNESSetLineSearch( )<br>SNESView( )<br>SNESGetSLES( )<br>SLESGetKSP( )<br>SLESGetPC( ) | -snes_rtol <rt> -snes_atol <at><br>-snes_max_its <its><br>-snes_eq_ls [cubic,quadratic,...]<br>-snes_view<br>-ksp_type <ksptype><br>-ksp_rtol <krt><br>-pc_type <pctype> ...  |



beginner



intermediate

*And many more options...*

 solvers:  
nonlinear

# SNES: Example Programs

Location: `petsc/src/snes/examples/tutorials/`

|                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                  |                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• <code>ex1.c</code>, <code>ex1f.F</code></li> <li>• <code>ex2.c</code></li> </ul>                                                                                                                                                                                                                                                             | <ul style="list-style-type: none"> <li>- basic uniprocess codes</li> <li>- uniprocess nonlinear PDE (1 DoF per node)</li> </ul>                                  |    |
| <ul style="list-style-type: none"> <li>  <code>ex5.c</code>, <code>ex5f.F</code>, <code>ex5f90.F</code><br/>per node)                 </li> </ul>                                                                                                                                                    | <ul style="list-style-type: none"> <li>- parallel nonlinear PDE (1 DoF per node)</li> </ul>                                                                      |                                                                                       |
| <ul style="list-style-type: none"> <li>  <ul style="list-style-type: none"> <li>• <code>ex18.c</code></li> </ul> </li> <li>  <ul style="list-style-type: none"> <li>• <code>ex19.c</code></li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>- parallel radiative transport problem with multigrid</li> <li>- parallel driven cavity problem with multigrid</li> </ul> |  |

*And many more examples ...*



beginner



intermediate

 - on-line exercise

solvers:  
nonlinear

# Timestepping Solvers (TS)

(and ODE Integrators)

beginner

- Application code interface

beginner

- Choosing the solver

beginner

- Setting algorithmic options

beginner

- Viewing the solver

intermediate

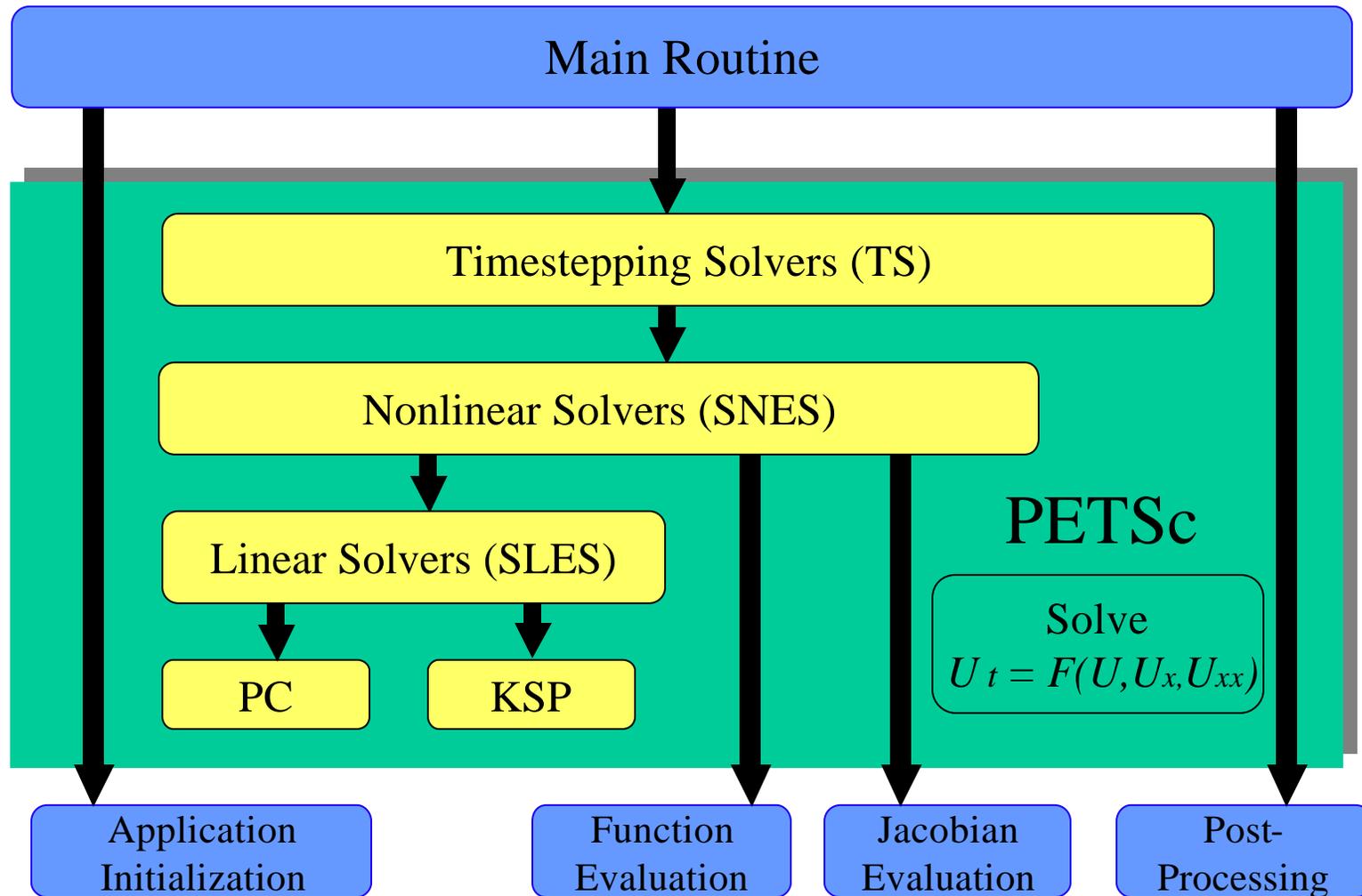
- Determining and monitoring convergence

advanced

- User-defined customizations

tutorial outline:  
solvers:  
timestepping

# Time-Dependent PDE Solution



◆ User code

◆ PETSc code

beginner

solvers:  
timestepping

# Timestepping Solvers

**Goal:** Support the (real and pseudo) time evolution of PDE systems

$$U_t = F(U, U_x, U_{xx}, t)$$

User provides:

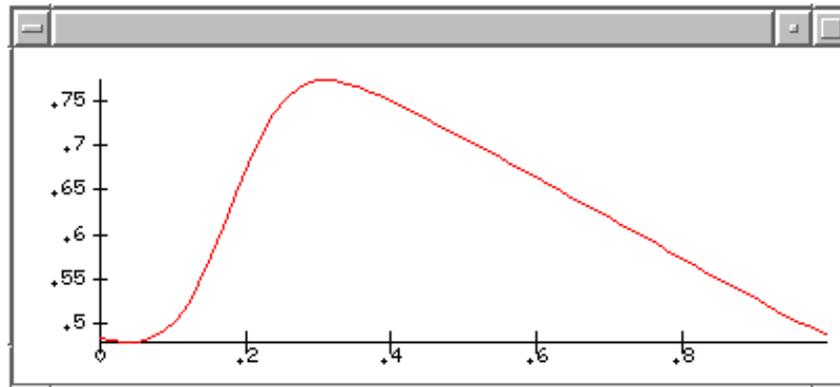
- Code to evaluate  $F(U, U_x, U_{xx}, t)$
- Code to evaluate Jacobian of  $F(U, U_x, U_{xx}, t)$ 
  - or use sparse finite difference approximation
  - or use automatic differentiation (coming soon!)

# Sample Timestepping Application: Burger's Equation

$$U_t = U U_x + \nu U_{xx}$$

$$U(0, x) = \sin(2\pi x)$$

$$U(t, 0) = U(t, 1)$$



beginner

solvers:  
timestepping

# Actual Local Function Code

$$U_t = F(t, U) = U_i (U_{i+1} - U_{i-1}) / (2h) +$$
$$?? (U_{i+1} - 2U_i + U_{i-1}) / (h * h)$$

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

# Timestepping Solvers

- Euler
- Backward Euler
- Pseudo-transient continuation
- Interface to **PVODE**, a sophisticated parallel ODE solver package by Hindmarsh et al. of LLNL
  - Adams
  - BDF

# Timestepping Solvers

- Allow full access to all of the PETSc
  - nonlinear solvers
  - linear solvers
  - distributed arrays, matrix assembly tools, etc.
- User can customize all phases of the solution process

# TS: Review of Basic Usage

- `TSCreate( )` - Create TS context
- `TSSetRHSFunction( )` - Set function eval. routine
- `TSSetRHSJacobian( )` - Set Jacobian eval. routine
- `TSSetFromOptions( )` - Set runtime solver options for [TS,SNES,SLES,KSP,PC]
- `TSSolve( )` - Run timestepping solver
- `TSView( )` - View solver options actually used at runtime (alternative: `-ts_view`)
- `TSDestroy( )` - Destroy solver

# TS: Review of Selected Options

| Functionality                                                                   | Procedural Interface                                                                              | Runtime Option                                                                                                                                                                                                                                          |
|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Set timestepping solver<br>Set monitoring routine                               | TSSetType( )<br>TSSetMonitor()                                                                    | -ts_type [euler,beuler,pseudo,...]<br>-ts_monitor <br>-ts_xmonitor, ...                                                                                              |
| Set timestep duration<br>View solver options<br>Set timestepping solver options | TSSetDuration ( )<br>TSView( )<br>TSGetSNES( )<br>SNESGetSLES( )<br>SLESGetKSP( )<br>SLESGetPC( ) | -ts_max_steps <maxsteps><br>-ts_max_time <maxtime><br>-ts_view<br>-snes_monitor -snes_rtol <rt><br>-ksp_type <ksptype> <br>-ksp_rtol <rt><br>-pc_type <pctype> ... |



beginner



intermediate

*And many more options...*

solvers:  
timestepping

# TS: Example Programs

Location: `petsc/src/ts/examples/tutorials/`

- |                                                                                                                            |                                                         |                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|-------------------------------------------------------------------------------------|
| • <code>ex1.c</code> , <code>ex1f.F</code>                                                                                 | - basic uniprocess codes (time-dependent nonlinear PDE) |  |
|  <code>ex2.c</code> , <code>ex2f.F</code> | - basic parallel codes (time-dependent nonlinear PDE)   |                                                                                     |

- |                      |                            |                                                                                      |
|----------------------|----------------------------|--------------------------------------------------------------------------------------|
| • <code>ex3.c</code> | - uniprocess heat equation |  |
| • <code>ex4.c</code> | - parallel heat equation   |                                                                                      |

*And many more examples ...*



beginner



intermediate

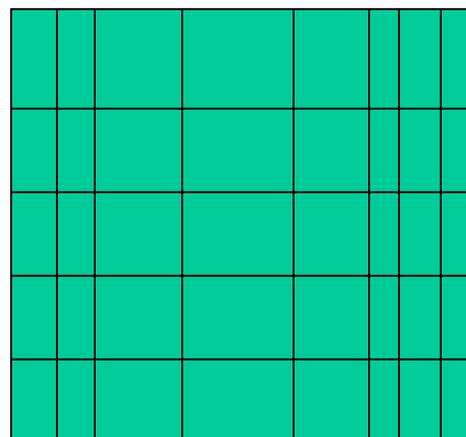
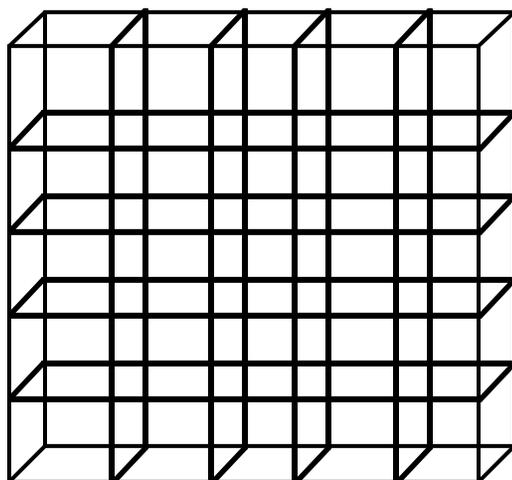
 - on-line exercise

solvers:  
timestepping

## Mesh Definitions: For Our Purposes

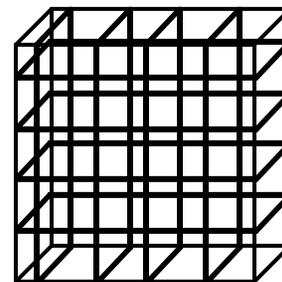
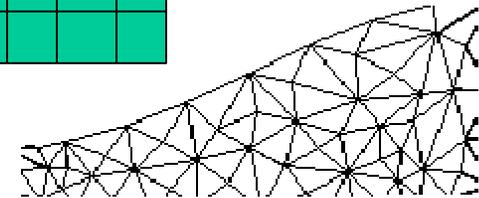
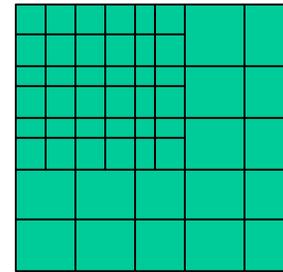
- **Structured:** Determine neighbor relationships purely from logical I, J, K coordinates
- **Semi-Structured:** In well-defined regions, determine neighbor relationships purely from logical I, J, K coordinates
- **Unstructured:** Do not explicitly use logical I, J, K coordinates

# Structured Meshes

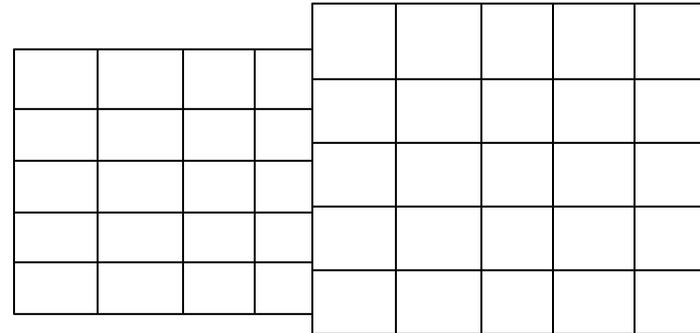
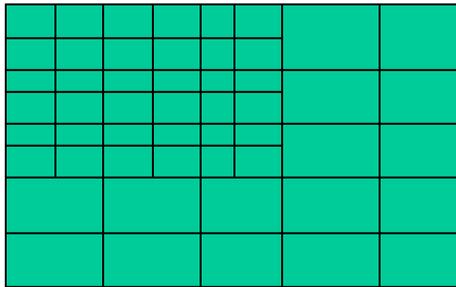


- PETSc support provided via DA objects

# Unstructured Meshes



# Semi-Structured Meshes



- No explicit PETSc support
  - **OVERTURE-PETSc** for composite meshes
  - **SAMRAI-PETSc** for AMR

# 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

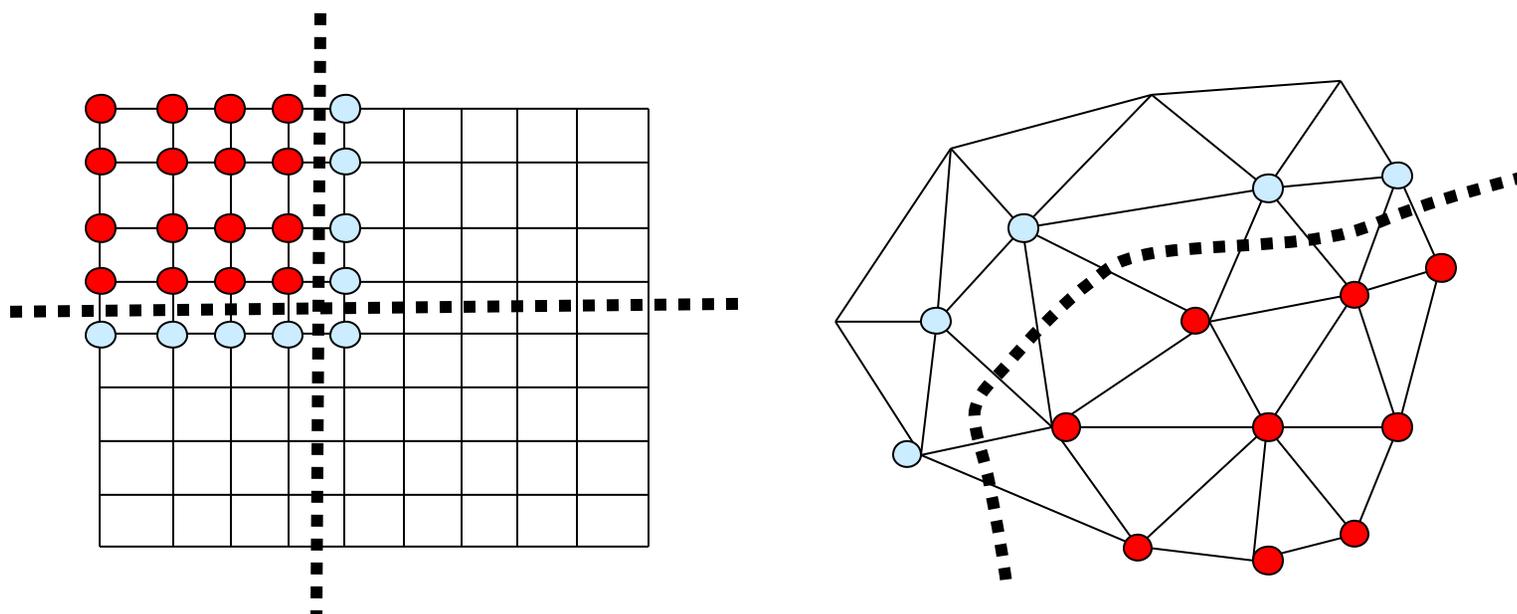


important concepts

tutorial outline:  
data layout

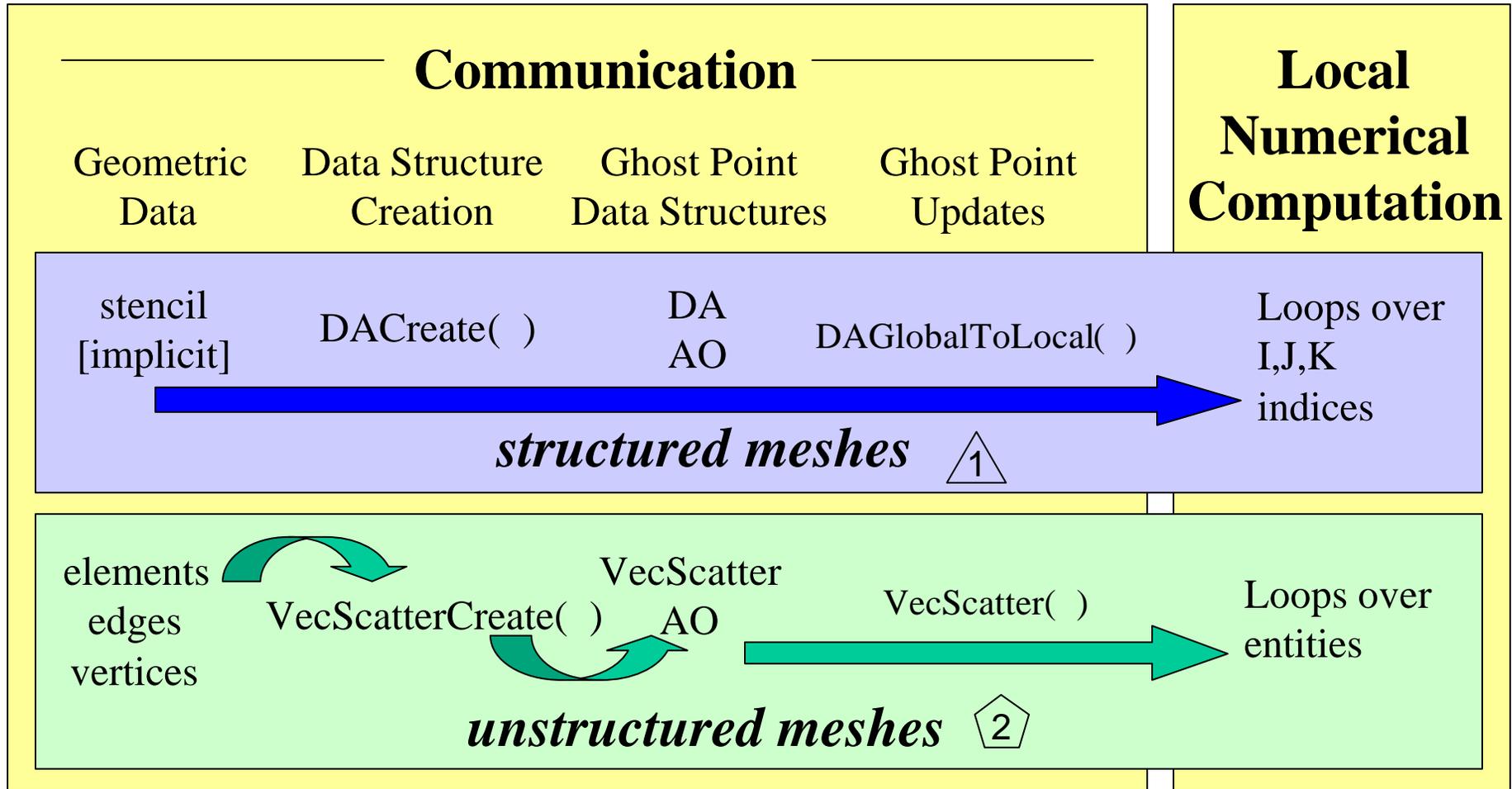
# Ghost Values

● 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.

# Communication and Physical Discretization



1

2

beginner    intermediate

data layout

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

beginner

beginner

beginner

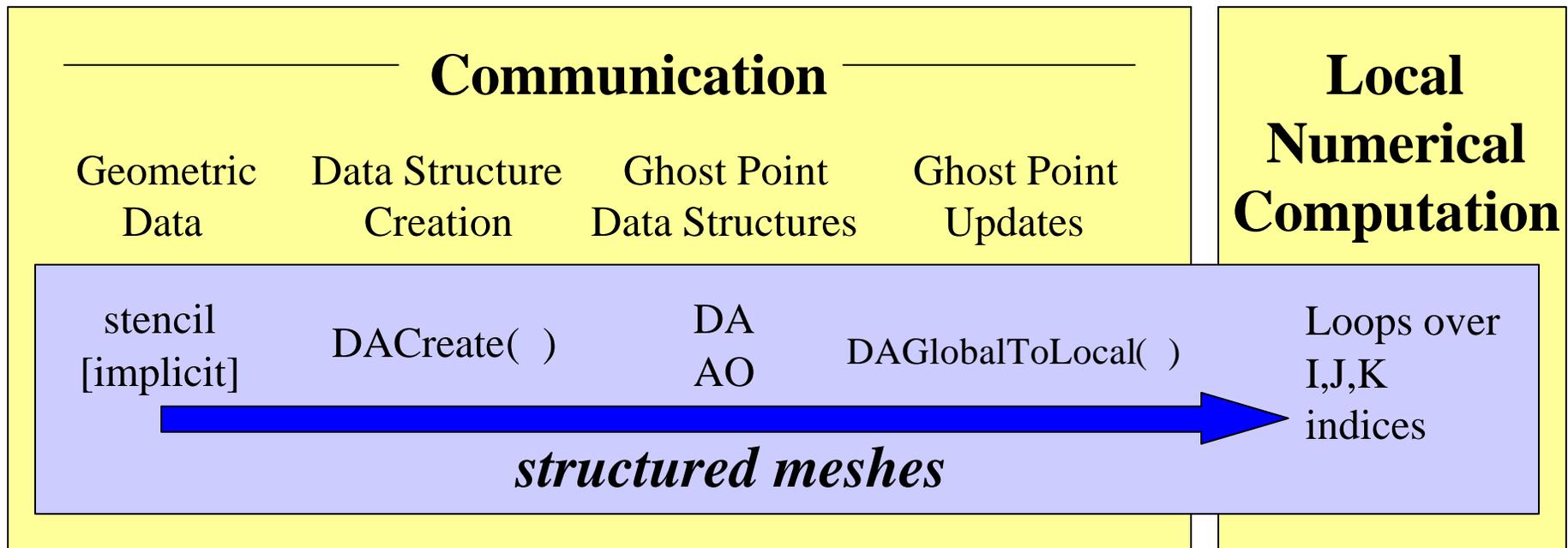
intermediate

intermediate

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

tutorial outline:  
data layout:  
distributed arrays

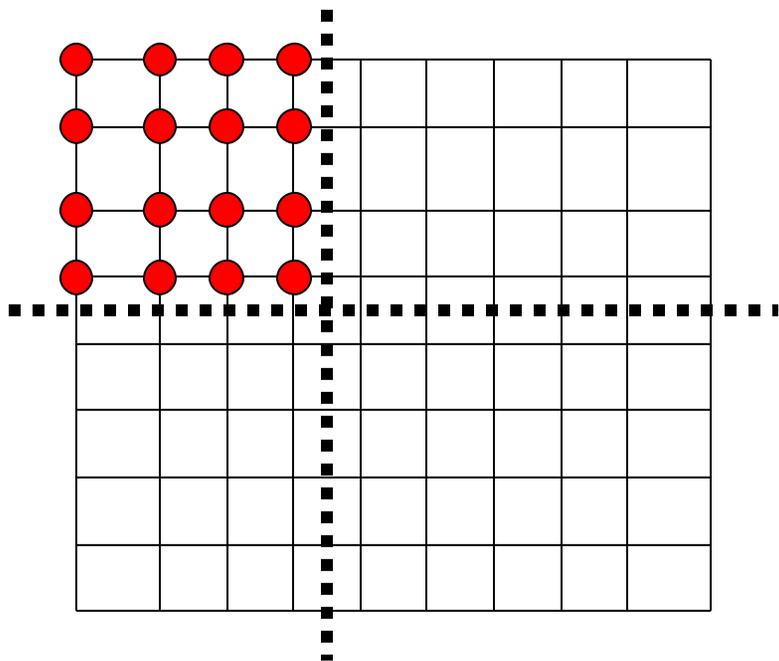
# Communication and Physical Discretization: Structured Meshes



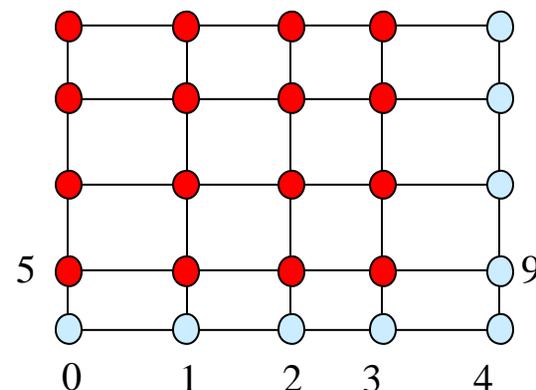
beginner

data layout:  
distributed arrays

# Global and Local Representations



- Local node
- Ghost node



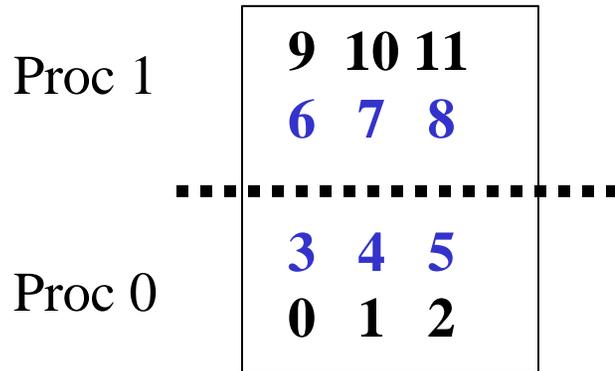
**Global:** each process stores a unique local set of vertices (and each vertex is owned by exactly one process)

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

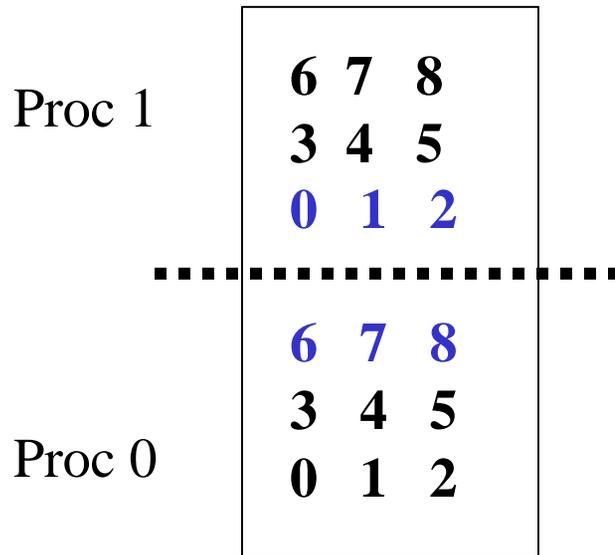
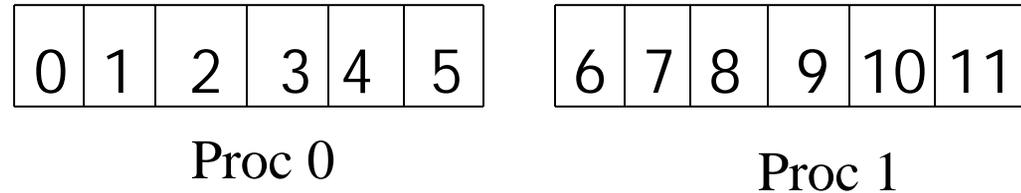
beginner

data layout:  
distributed arrays

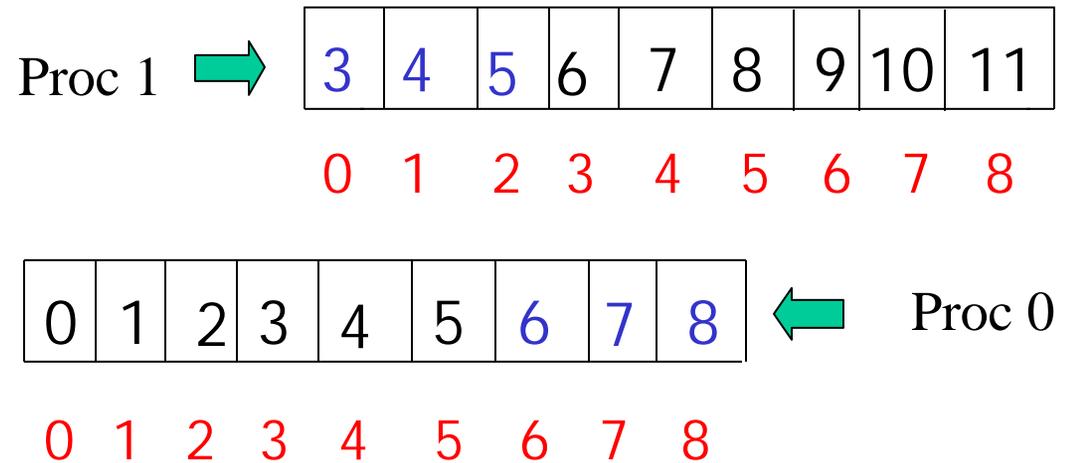
# Global and Local Representations (cont.)



## Global Representation:



## Local Representations:



beginner

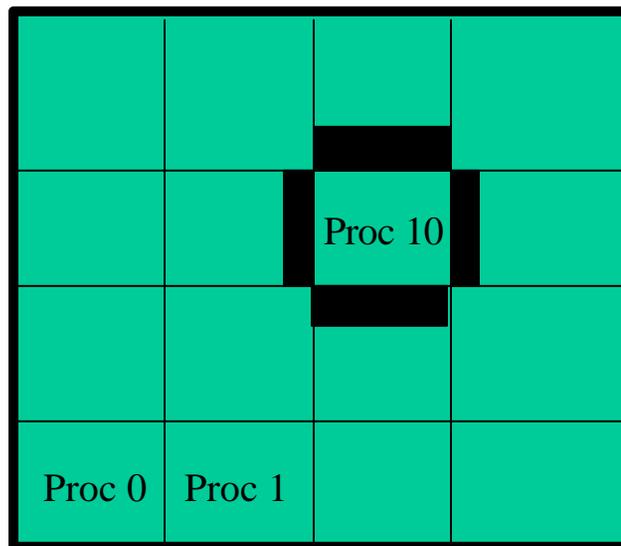
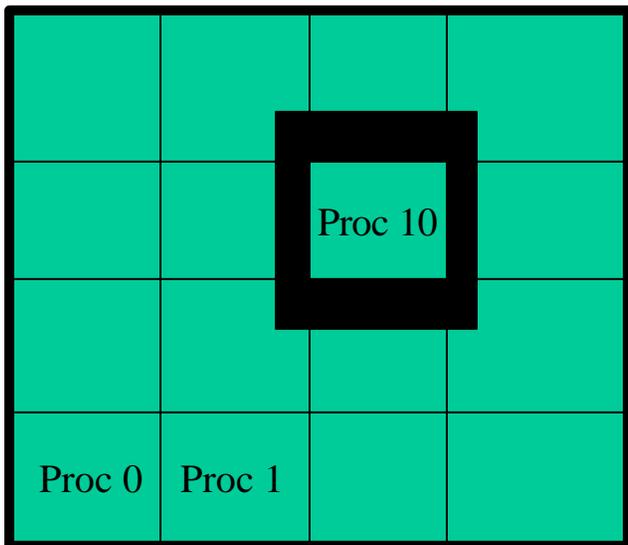
data layout:  
distributed arrays

# Logically Regular Meshes

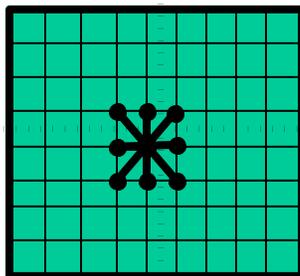
- DA - Distributed Array: object containing information about vector layout across the processes and communication of ghost values
- Form a DA
  - `DACreate1d(...,DA *)`
  - `DACreate2d(...,DA *)`
  - `DACreate3d(...,DA *)`
- Create the corresponding PETSc vectors
  - `DACreateGlobalVector( DA, Vec *)`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

# Distributed Arrays

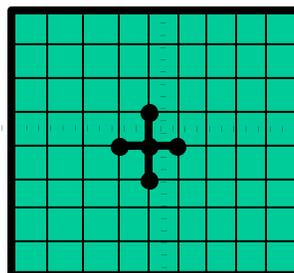
## Data layout and ghost values



*Box-type stencil*



*Star-type stencil*



beginner

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  $(0, 1, \dots, n_{local}-1)$

## DACreate1d(..., \*DA)

- DACreate1d(MPI\_Comm comm, DAPeriodicType wrap, int M, int dof, int s, int \*lc, DA \*inra)
  - MPI\_Comm — processes containing array
  - DA\_[NONPERIODIC, XPERIODIC]
  - number of grid points in x-direction
  - degrees of freedom per node
  - stencil width
  - Number of nodes for each cell (use PETSC\_NULL for the default)

## DACreate2d(..., \*DA)

- DACreate2d(MPI\_Comm comm, DAPeriodicType wrap, DAStencilType stencil\_type, int M, int N, int m, int n, int dof, int s, int \*lx, int \*ly, DA \*inra)
  - DA\_[NON,X,Y,XY]PERIODIC
  - DA\_STENCIL\_[STAR,BOX]
  - 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 cell (use PETSC\_NULL for the default) as tensor-product

And similarly for DACreate3d()

beginner

data layout:  
distributed arrays

# Updating the Local Representation

Two-step process that enables overlapping computation and communication

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

beginner

data layout:  
distributed arrays

# Ghost Point Scatters: Burger's Equation Example

```

call
DAGlobalToLocalBegin(da,u_global,INSERT_VALUES,u_local,ierr)
call
DAGlobalToLocalEnd(da,u_global,INSERT_VALUES,u_local,ierr)

```

```

call VecGetArray(u_local, 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

```

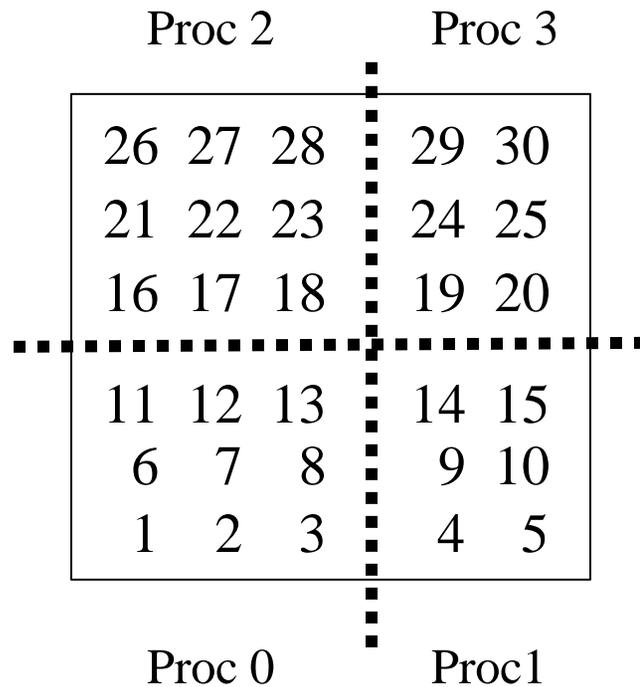
```
call VecRestoreArray(u_local, uv, ui, ierr)
```

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

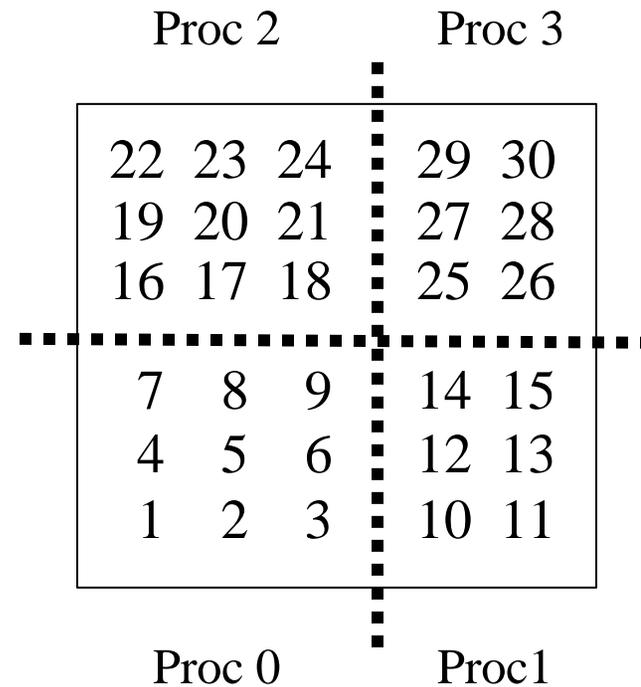
beginner

data layout:  
distributed arrays

# Global Numbering used by DAs



Natural numbering, corresponding to the entire problem domain



PETSc numbering used by DAs

intermediate

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

intermediate

data layout:  
distributed arrays

# Distributed Array Example

- `src/snes/examples/tutorial/ex5.c, ex5f.F`
  - Uses functions that help construct vectors and matrices naturally associated a DA
    - `DAGetMatrix`
    - `DASetLocalFunction`
    - `DASetLocalJacobian`



# End of Day 2

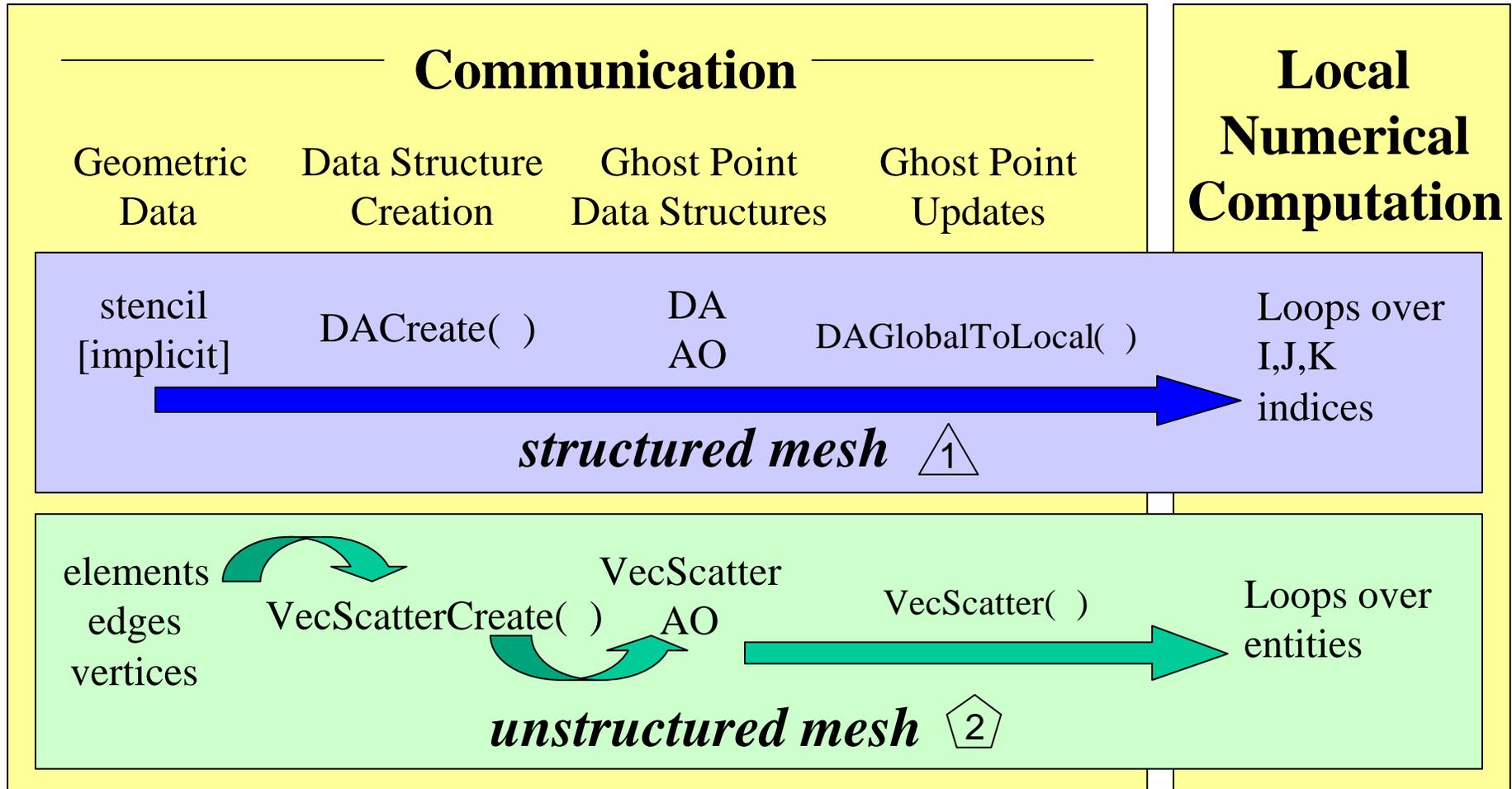
# Unstructured Meshes

- Setting up communication patterns is much more complicated than the structured case due to
  - mesh dependence
  - discretization dependence
    - cell-centered
    - vertex-centered
    - cell and vertex centered (e.g., staggered grids)
    - mixed triangles and quadrilaterals
- Can use VecScatter
  - Introduction in this tutorial
  - See additional tutorial material available via [PETSc web site](#)

beginner

data layout:  
vector scatters

# Communication and Physical Discretization



1

2

beginner    intermediate

data layout

# 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 (calling process) numbering of indices to a global (across-processes) numbering

intermediate

data layout:  
vector scatters

# Setting Up the Communication Pattern

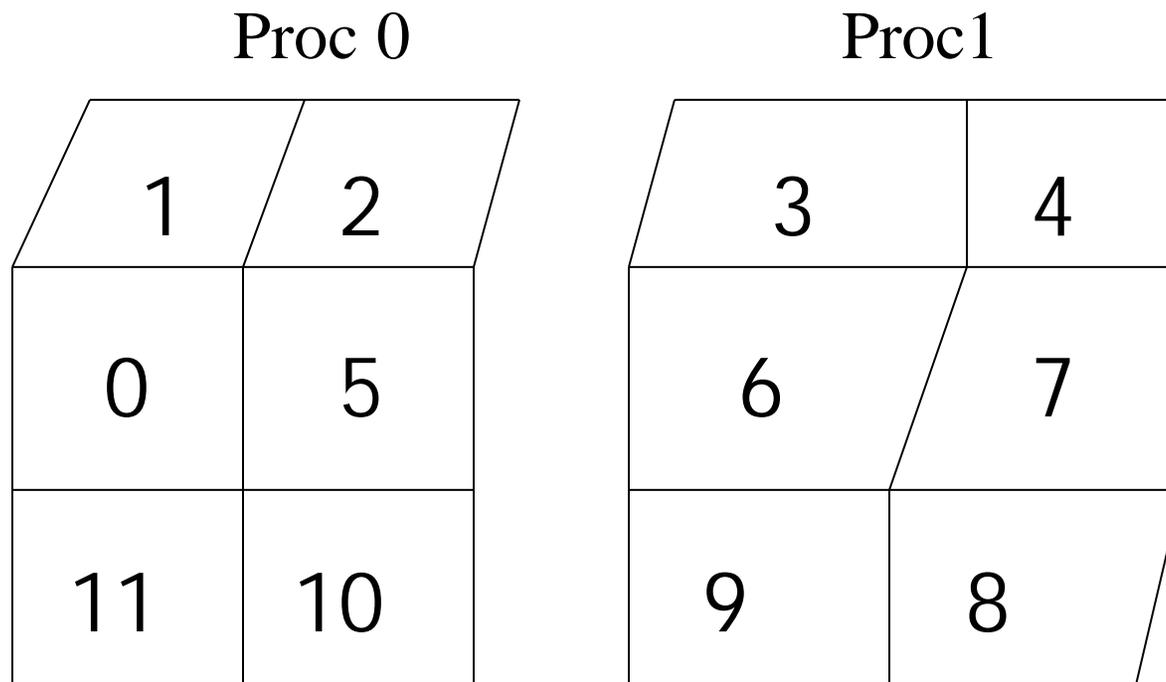
(the steps in creating VecScatter)

- Renumber objects so that contiguous entries are adjacent
  - Use AO to map from application to PETSc ordering
- 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

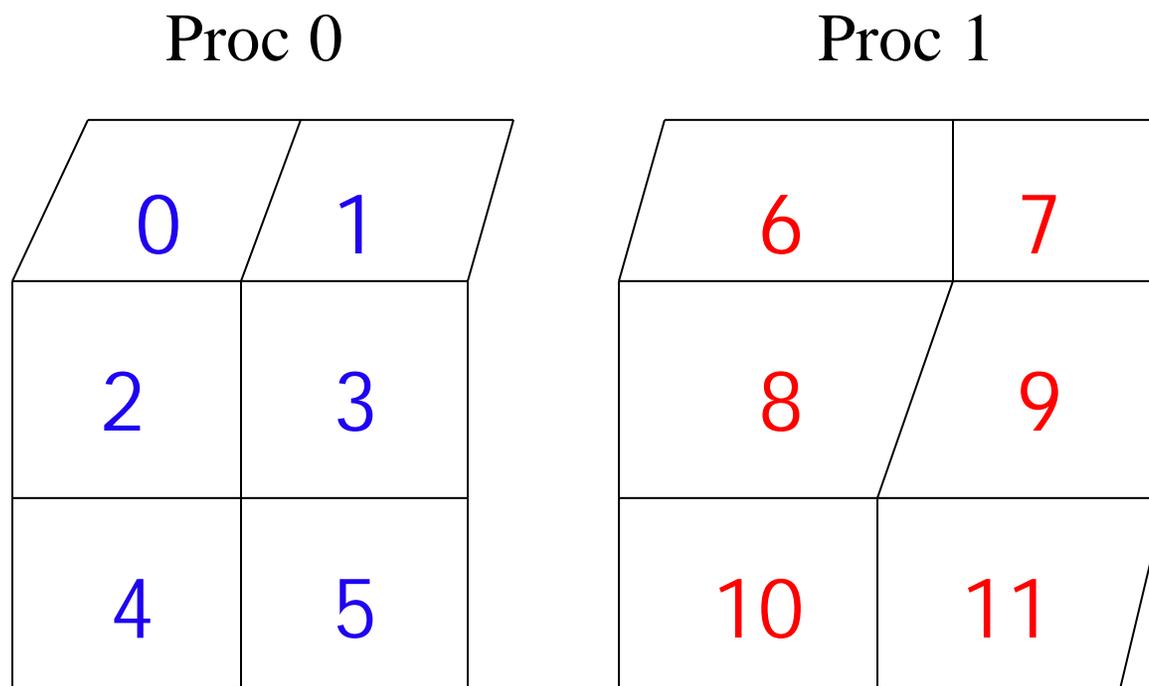


global indices defined by application

intermediate

data layout:  
vector scatters

# PETSc Parallel Numbering



global indices numbered contiguously on each process

intermediate

data layout:  
vector scatters

# Remapping Global Numbers: An Example

- Process 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 |
| 2 | 3 |
| 4 | 5 |

|    |    |
|----|----|
| 6  | 7  |
| 8  | 9  |
| 10 | 11 |

- Process 1

- n\_local: 6

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

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

application  
numbers

|    |    |
|----|----|
| 1  | 2  |
| 0  | 5  |
| 11 | 10 |

|   |   |
|---|---|
| 3 | 4 |
| 6 | 7 |
| 9 | 8 |

intermediate

data layout:  
vector scatters

# Remapping Numbers (1)

- Generate a parallel object (AO) to use in mapping between numbering schemes
- AOCreateBasic( 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

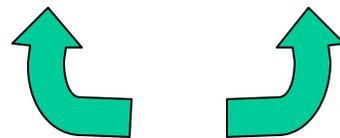
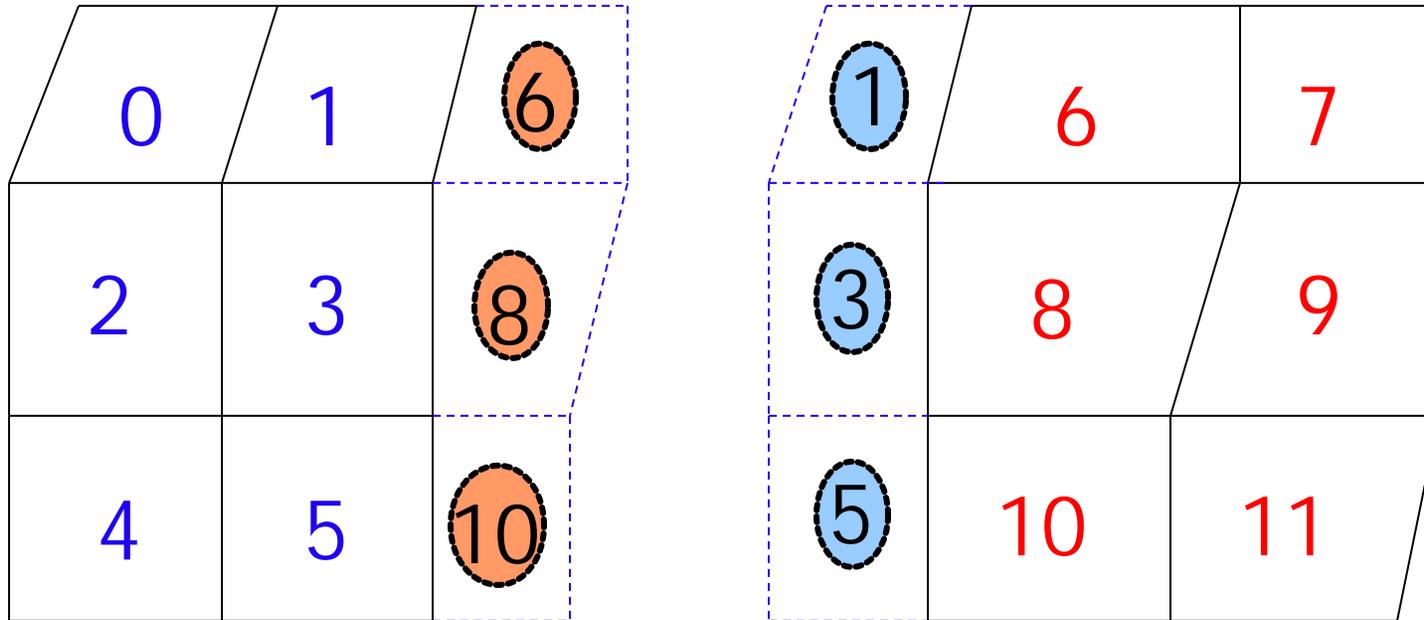
data layout:  
vector scatters

# Neighbors

using global PETSc numbers (ordering of the global PETSc vector)

Process 0

Process 1

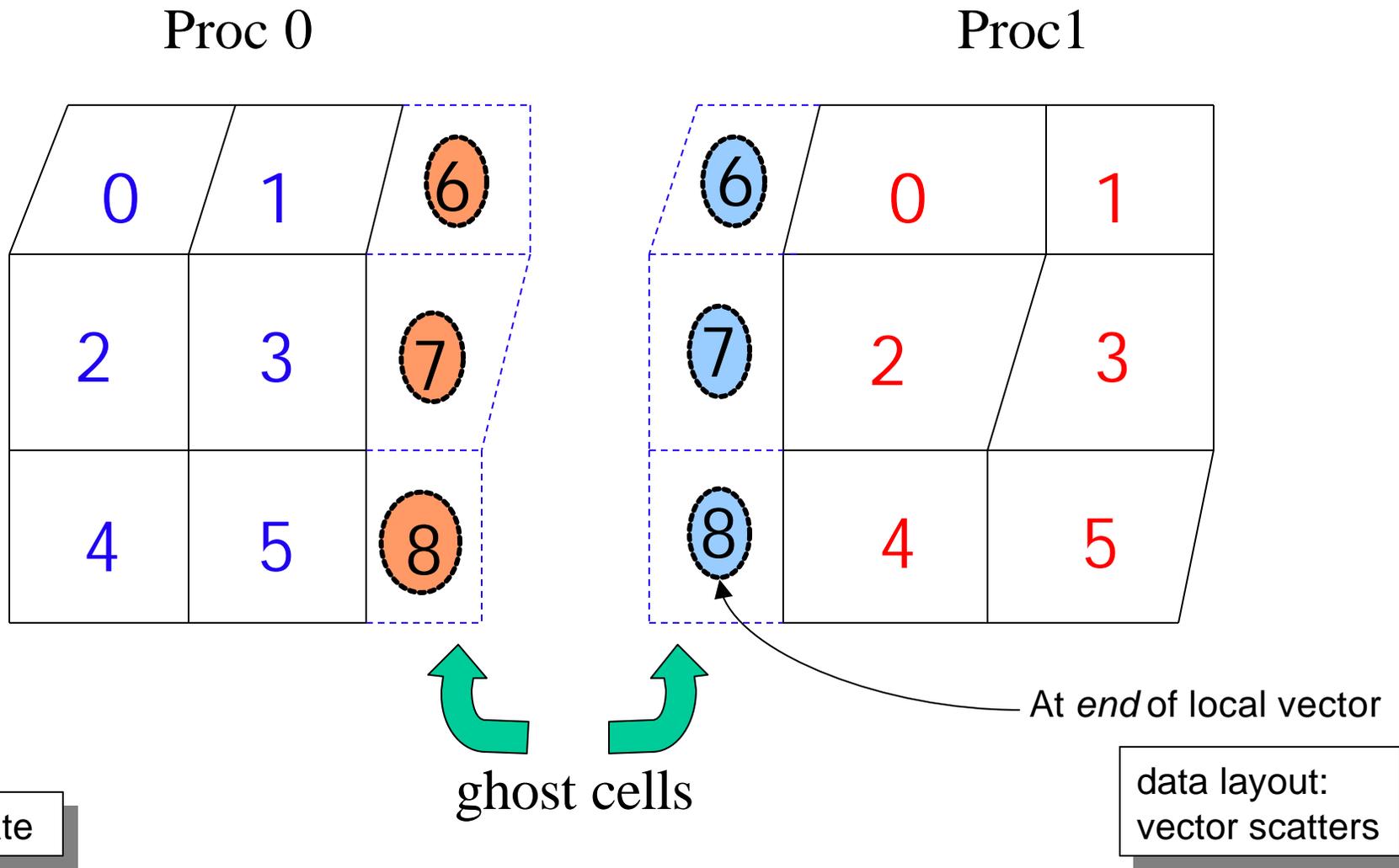


ghost cells

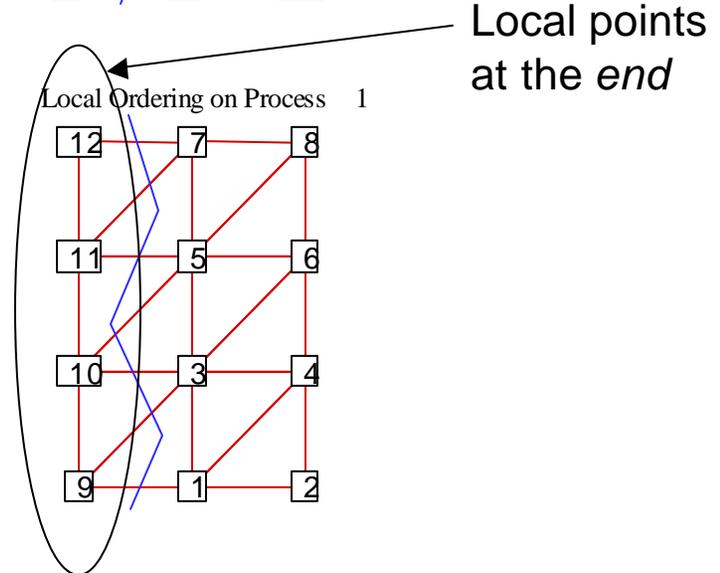
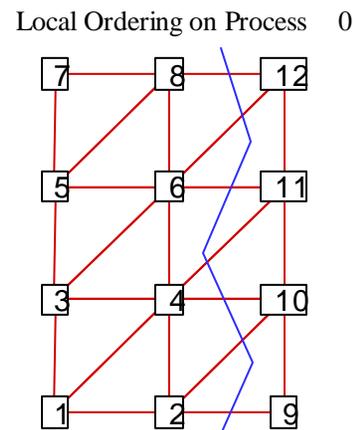
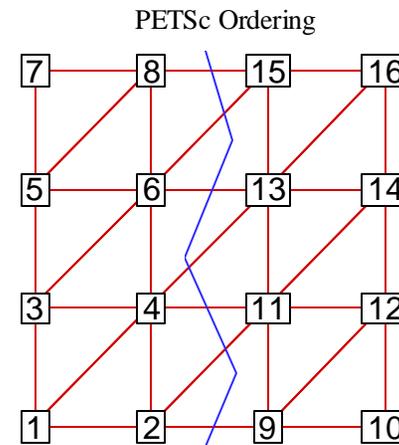
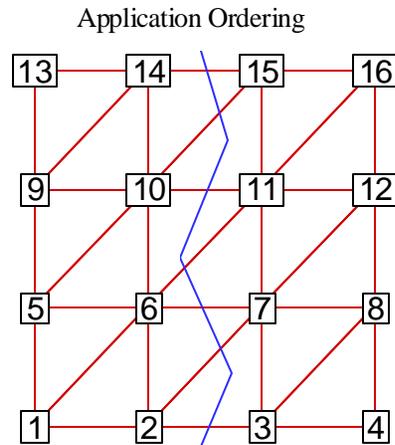
intermediate

data layout:  
vector scatters

# Local Numbering



# Summary of the Different Orderings



# Global and Local Representations

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

intermediate

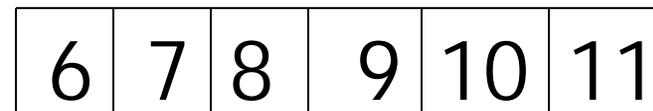
data layout:  
vector scatters

# Global and Local Representations

Global representation:



Proc 0



Proc1

Local Representations:



← Proc 0

0 1 2 3 4 5 6 7 8

Proc1 →



0 1 2 3 4 5 6 7 8

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

- Process 0

- int indices[] = {0,1,2,3,4,5,6,8,10};

- ISCreateGeneral(PETSC\_COMM\_WORLD,  
9, indices, IS \*isg);

ISCreateGeneral()

- Specify *global* numbers of locally owned cells, including ghost cells

- ISCreateStride(PETSC\_COMM\_SELF, 9,0,1,IS \*isl);

- Process 1

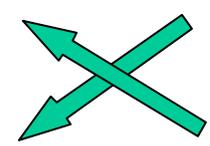
- int indices = {6,7,8,9,10,11,1,3,5};

- ISCreateGeneral(PETSC\_COMM\_WORLD,  
9, indices, 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

## 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, user->Xlocal, SCATTER_FORWARD,
 INSERT_VALUES);
 VecScatterEnd(user->scatter, Xglobal, user->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, user->Flocal, Fglobal, SCATTER_REVERSE,
 INSERT_VALUES);
 VecScatterEnd(user->scatter, user->Flocal, Fglobal, SCATTER_REVERSE,
 INSERT_VALUES);
 return 0;
}

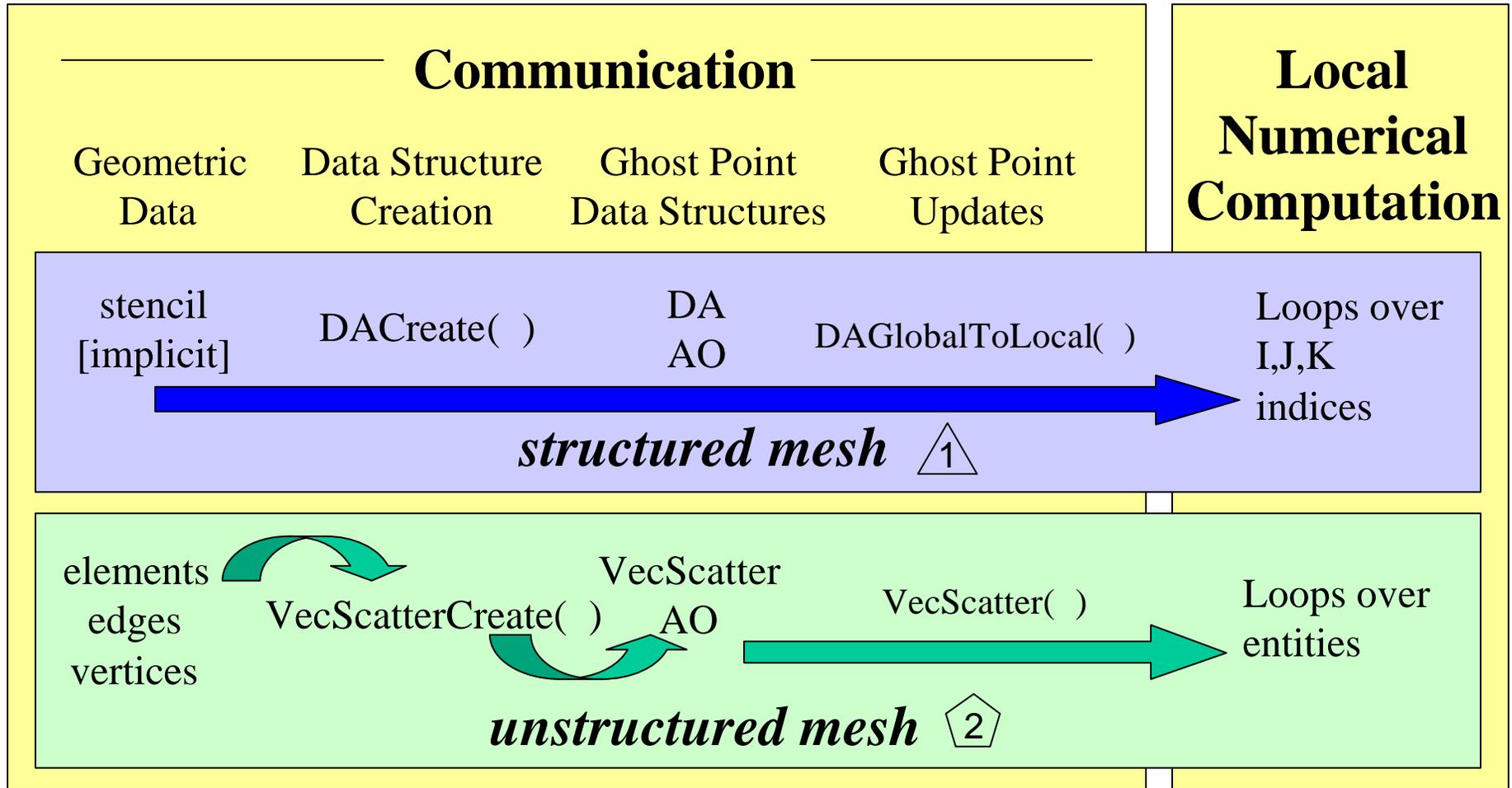
```

Predefined Scatter Context

intermediate

data layout:  
vector scatters

# Communication and Physical Discretization



1

2

beginner    intermediate

data layout

# Unstructured Meshes: Example Programs

- Location:
  - [petsc/src/snes/examples/tutorials/ex10d/ex10.c](#)
  - 2 process example (PETSc requires that a separate tool be used to divide the unstructured mesh among the processes).

# Matrix Partitioning and Coloring

Intermediate

intermediate

- Partitioner object creation and use
- Sparse finite differences for Jacobian computation (using colorings)

tutorial outline:  
data objects:  
matrix partitioning  
and coloring

# Partitioning for Load Balancing

- **MatPartitioning** - object for managing the partitioning of meshes, vectors, matrices, etc.
- Interface to parallel partitioners, i.e., the adjacency matrix, is stored in parallel
- **MatPartitioningCreate**(
  - `MPI_Comm comm`
  - `MatPartitioning *matpart`);

intermediate

data objects:  
matrices:  
partitioning

# Setting Partitioning Software

- `MatPartitioningSetType(`
  - `MatPartitioning matpart,`
  - `MatPartitioningType`  
`MATPARTITIONING_PARMETIS);`
- PETSc interface could support other packages (e.g., Pjostle, just no one has written wrapper code yet)

intermediate

data objects:  
matrices:  
partitioning

# Setting Partitioning Information

- `MatPartitioningSetAdjacency(  
– MatPartitioning matpart,  
– Mat adjacency_matrix);`
- `MatPartitioningSetVertexWeights(  
– MatPartitioning matpart,  
– int *weights);`
- `MatPartitioningSetFromOptions(  
– MatPartitioning matpart);`

intermediate

data objects:  
matrices:  
partitioning

# Partitioning

- MatPartitioningApply(
  - MatPartitioning matpart,
  - IS\* processforeachlocalnode);
- ...
- MatPartitioningDestroy(
  - MatPartitioning matpart);

intermediate

data objects:  
matrices:  
partitioning

# Constructing Adjacency Matrix

- `MatCreateMPIAdj()` (Uses CSR input format)
  - `MPI_Comm comm`,
  - `int numberlocalrows`,
  - `int numbercolumns`,
  - `int rowindices[ ]`,
  - `int columnindices[ ]`,
  - `int values[ ]`,
  - `Mat *adj`)
- Other input formats possible, e.g., via `MatSetValues()`

intermediate

data objects:  
matrices:  
partitioning

# Finite Difference Approximation of Sparse Jacobians

Two-stage process:

- 1 Compute a coloring of the Jacobian (e.g., determine columns of the Jacobian that may be computed using a single function evaluation)
  - 2 Construct a `MatFDColoring` object that uses the coloring information to compute the Jacobian
- example: driven cavity model:  
`petsc/src/snes/examples/tutorials/ex8.c`

intermediate

data objects:  
matrices:  
coloring

# Coloring for Structured Meshes

- DAGetColoring (
  - DA da,
  - ISColorType ctype,
  - ISColoring \*coloringinfo );
- DAGetMatrix(
  - DA da,
  - MatType mtype,
  - Mat \*sparsematrix );
- For structured meshes (using DAs) we provide parallel colorings

intermediate

data objects:  
matrices:  
coloring

# Coloring for Unstructured Meshes

- MatGetColoring (
  - Mat A,
  - MatColoringType
    - MATCOLORING\_NATURAL
    - MATCOLORING\_SL
    - MATCOLORING\_LF
    - MATCOLORING\_ID
  - ISColoring \*coloringinfo)
- Automatic coloring of sparse matrices currently implemented only for sequential matrices
- If using a “local” function evaluation, the sequential coloring is enough

intermediate

data objects:  
matrices:  
coloring

# Actual Computation of Jacobians

- MatFDColoringCreate (
  - Mat J,
  - ISColoring coloringinfo,
  - MatFDColoring \*matfd)
- MatFDColoringSetFunction(
  - MatFDColoring matfd,
  - int (\*f)(void),void \*fctx)
- MatFDColoringSetFromOptions(
  - MatFDColoring)

intermediate

data objects:  
matrices:  
coloring

# Computation of Jacobians within SNES

- User calls: `SNESSetJacobian()`
  - `SNES snes,`
  - `Mat A ,`
  - `Mat J,`
  - `SNESDefaultComputeJacobianColor( ),`
  - `fdcoloring);`
- where `SNESSetJacobian()` actually uses .... `MatFDColoringApply()`
  - `Mat J,`
  - `MatFDColoring coloring,`
  - `Vec x1,`
  - `MatStructure *flag,`
  - `void *sctx)`
- to form the Jacobian

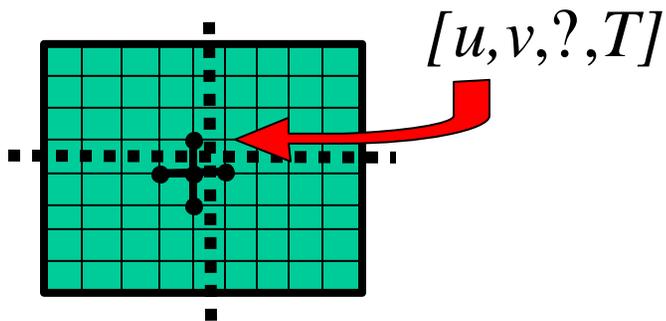
intermediate

data objects:  
matrices:  
coloring

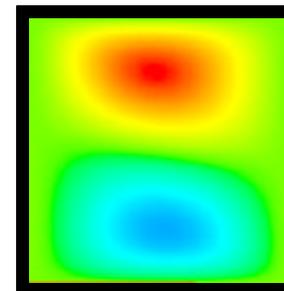
# Driven Cavity Model

Example code: `petsc/src/snes/examples/tutorials/ex19.c`

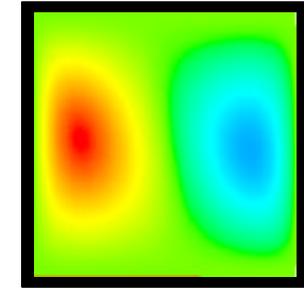
- Velocity-vorticity formulation, with flow driven by lid and/or bouyancy
- Finite difference discretization with 4 DoF per mesh point



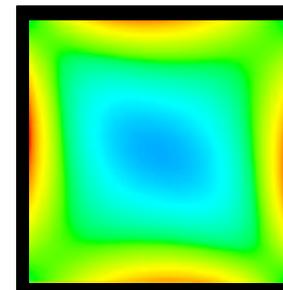
## Solution Components



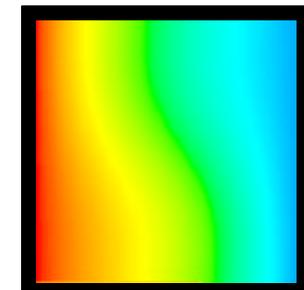
velocity:  $u$



velocity:  $v$



vorticity: ?



temperature:  $T$

1

beginner

2

intermediate

solvers:  
nonlinear

# Driven Cavity Program

- **Part A:** Parallel data layout
- **Part B:** Nonlinear solver creation, setup, and usage
- **Part C:** Nonlinear function evaluation
  - ghost point updates
  - local function computation
- **Part D:** Jacobian evaluation
  - default colored finite differencing approximation
- Experimentation

1

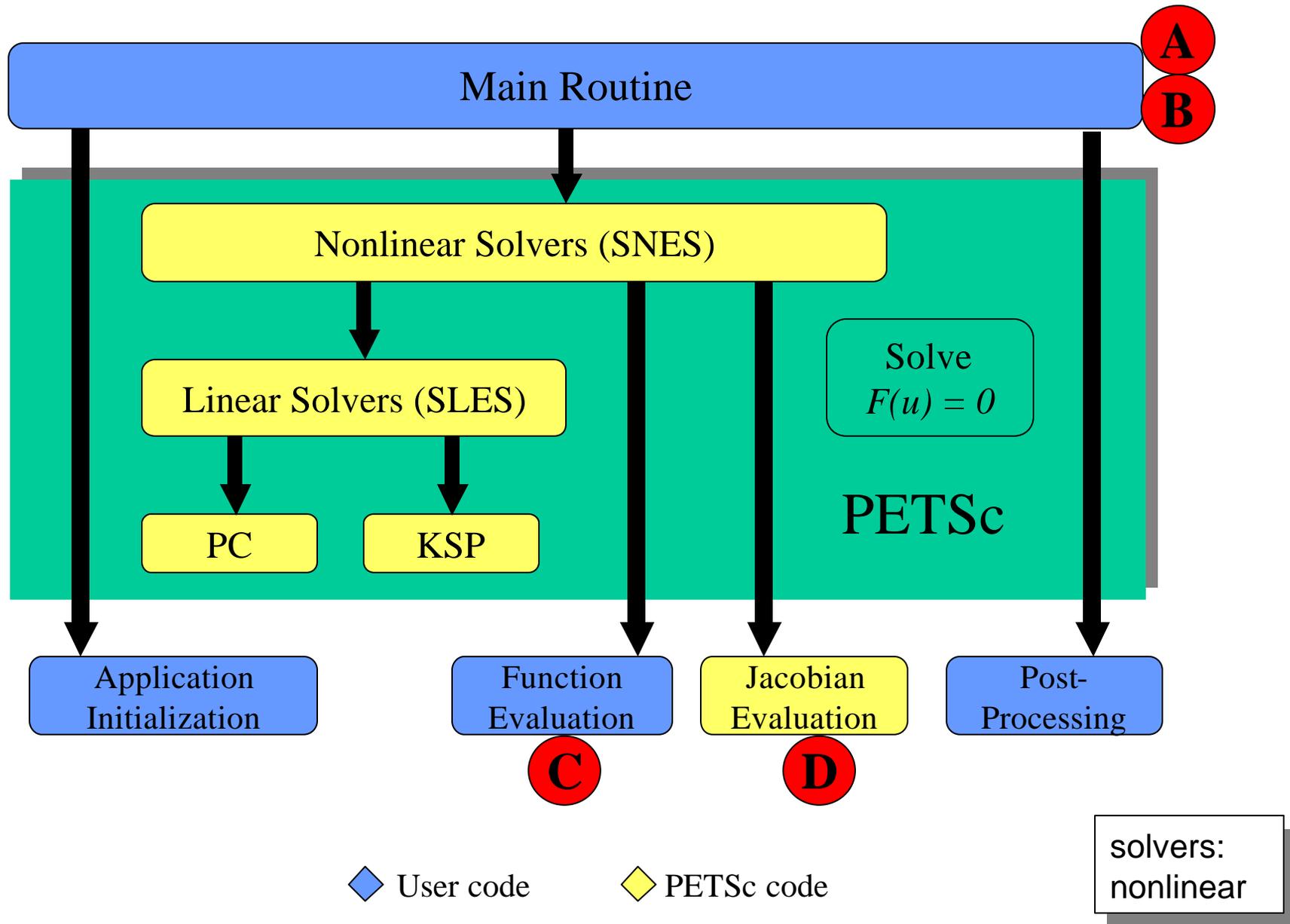
beginner

2

intermediate

solvers:  
nonlinear

# Driven Cavity Solution Approach



# Driven Cavity:

## Running the program (1)

Matrix-free Jacobian approximation with no preconditioning (via `-snes_mf`) ... does not use explicit Jacobian evaluation

- 1 process: (thermally-driven flow)
  - `mpirun -np 1 ex19 -snes_mf -snes_monitor -grashof 1000.0 -lidvelocity 0.0`
- 2 processes, view DA (and pausing for mouse input):
  - `mpirun -np 2 ex19 -snes_mf -snes_monitor -da_view_draw -draw_pause -1`
- View contour plots of converging iterates
  - `mpirun ex19 -snes_mf -snes_monitor -snes_vecmonitor`

# Driven Cavity:

## Running the program (2)

- Use MatFDColoring for sparse finite difference Jacobian approximation; view SNES options used at runtime
  - `mpirun ex8 -snes_view -mat_view_info`
- Set trust region Newton method instead of default line search
  - `mpirun ex8 -snes_type tr -snes_view -snes_monitor`
- Set transpose-free QMR as Krylov method; set relative KSP convergence tolerance to be .01
  - `mpirun ex8 -ksp_type tfqmr -ksp_rtol .01 -snes_monitor`

# Debugging and Error Handling

beginner

beginner

developer

- Automatic generation of tracebacks
- Detecting memory corruption and leaks
- Optional user-defined error handlers

tutorial outline:  
debugging and errors

# Debugging

## Support for parallel debugging

- `-start_in_debugger [gdb,dbx,noxterm]`
- `-on_error_attach_debugger [gb,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%-----
```

beginner

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

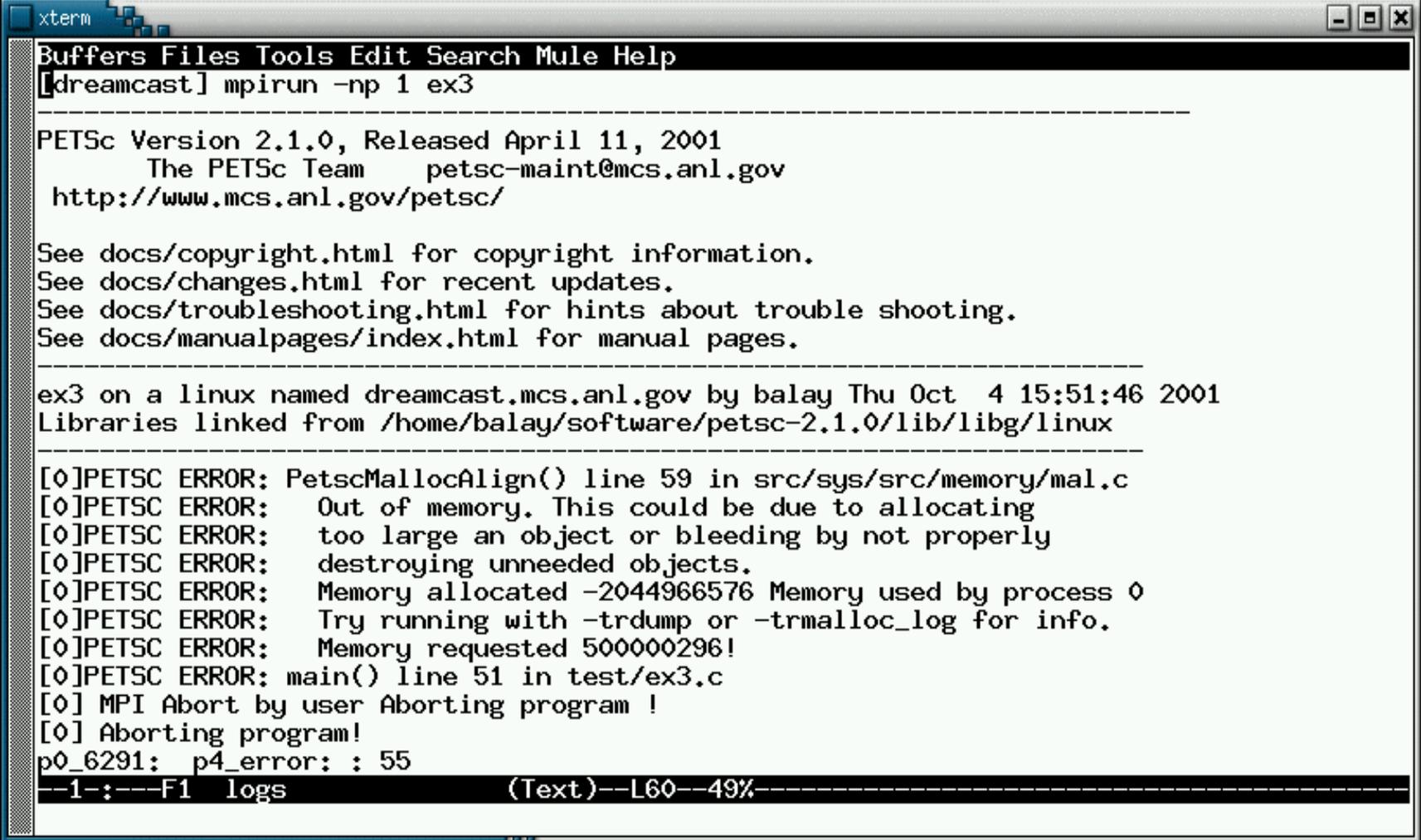
PETSc Version 2.1.0, Released April 11, 2001
 The PETSc Team petsc-maint@mcs.anl.gov
 http://www.mcs.anl.gov/petsc/

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%
```

# 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/

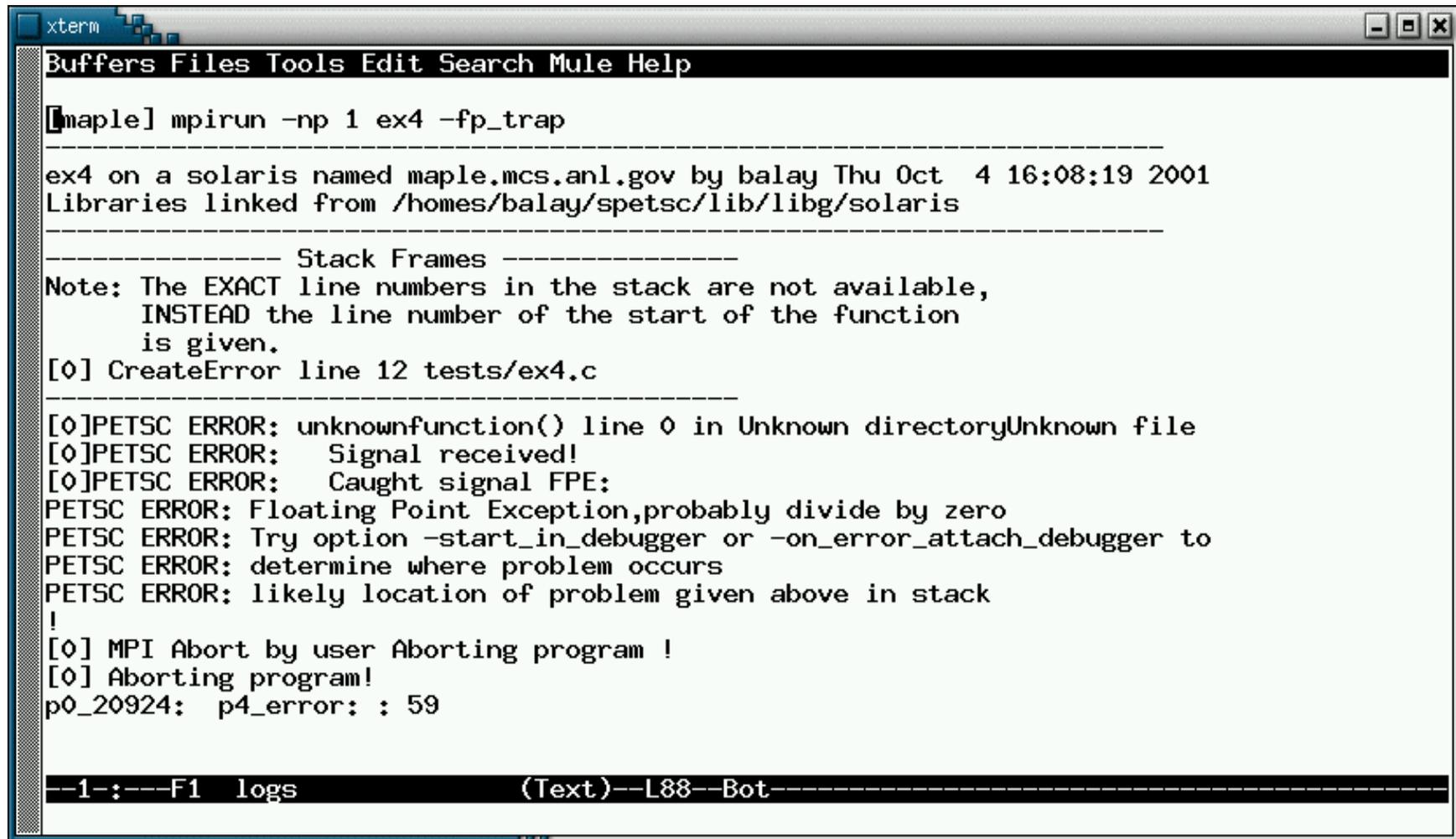
See docs/copyright.html for copyright information.
See docs/changes.html for recent updates.
See docs/troubleshooting.html for hints about trouble shooting.
See docs/manualpages/index.html for manual pages.

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%
```

# 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-----
```

# 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

intermediate

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  $a[nnz]$ ,  $j[nnz]$ ,  $i[n]$ ,  $x[n]$ ,  $y[n]$

# Simple Performance Analysis

- Memory motion:
  - $nnz (\text{sizeof}(\text{double}) + \text{sizeof}(\text{int})) + n (2 * \text{sizeof}(\text{double}) + \text{sizeof}(\text{int}))$
  - 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:
  - $nz (2 * \text{load-double} + \text{load-int} + \text{mult-add}) + n (\text{load-int} + \text{store-double})$
- 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

intermediate

profiling and  
performance tuning

# Alternative Building Blocks

- Performance of sparse matrix - multi-vector multiply:

| <i>Format</i> | <i>Number<br/>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

intermediate

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

intermediate

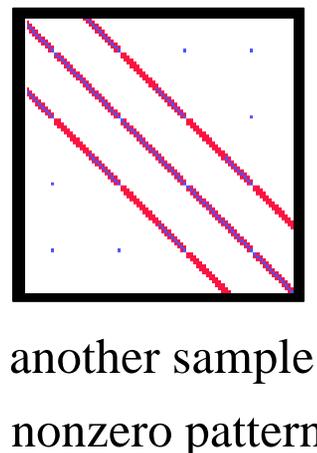
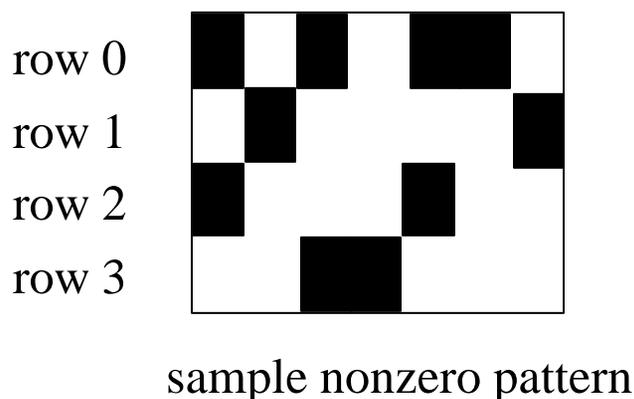
profiling and  
performance tuning

# 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



intermediate

profiling and  
performance tuning

# Symbolic Computation of Matrix Nonzero Structure

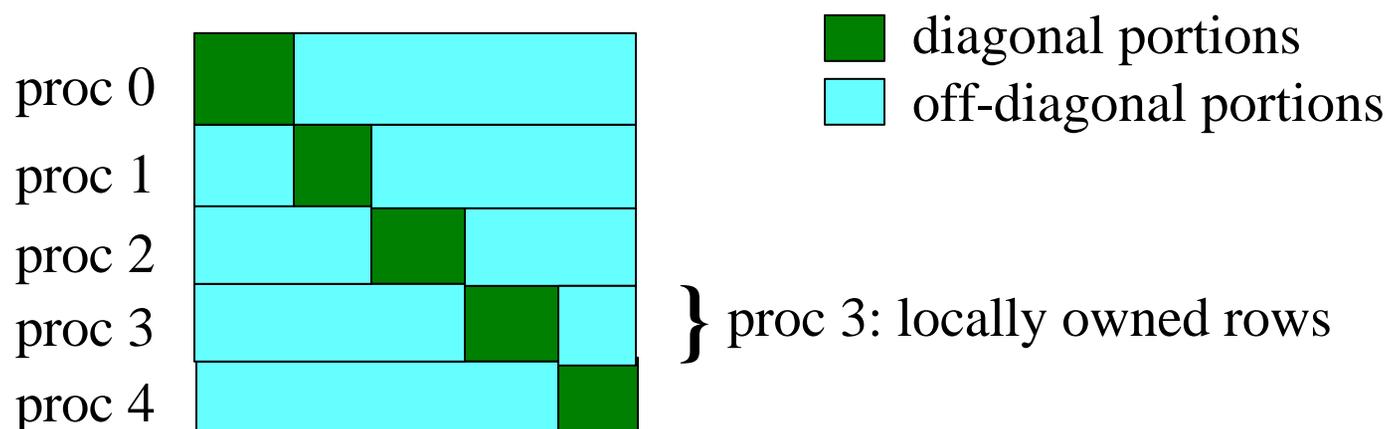
- Write code that forms the nonzero structure of the matrix
  - loop over the grid for finite differences
  - loop over the elements for finite elements
  - etc.
- Then create matrix via `MatCreateSeqAIJ()`, `MatCreateMPIAIJ()`, ...

intermediate

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.



intermediate

profiling and  
performance tuning

# Indicating Expected Nonzeros Parallel Sparse Matrices

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

- `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

# Verifying Predictions

- Use runtime option: `-log_info`
- Output:
  - [proc #] Matrix size: % d X % d; storage space: % d unneeded, % 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 unneeded, 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!
```

intermediate

profiling and  
performance tuning

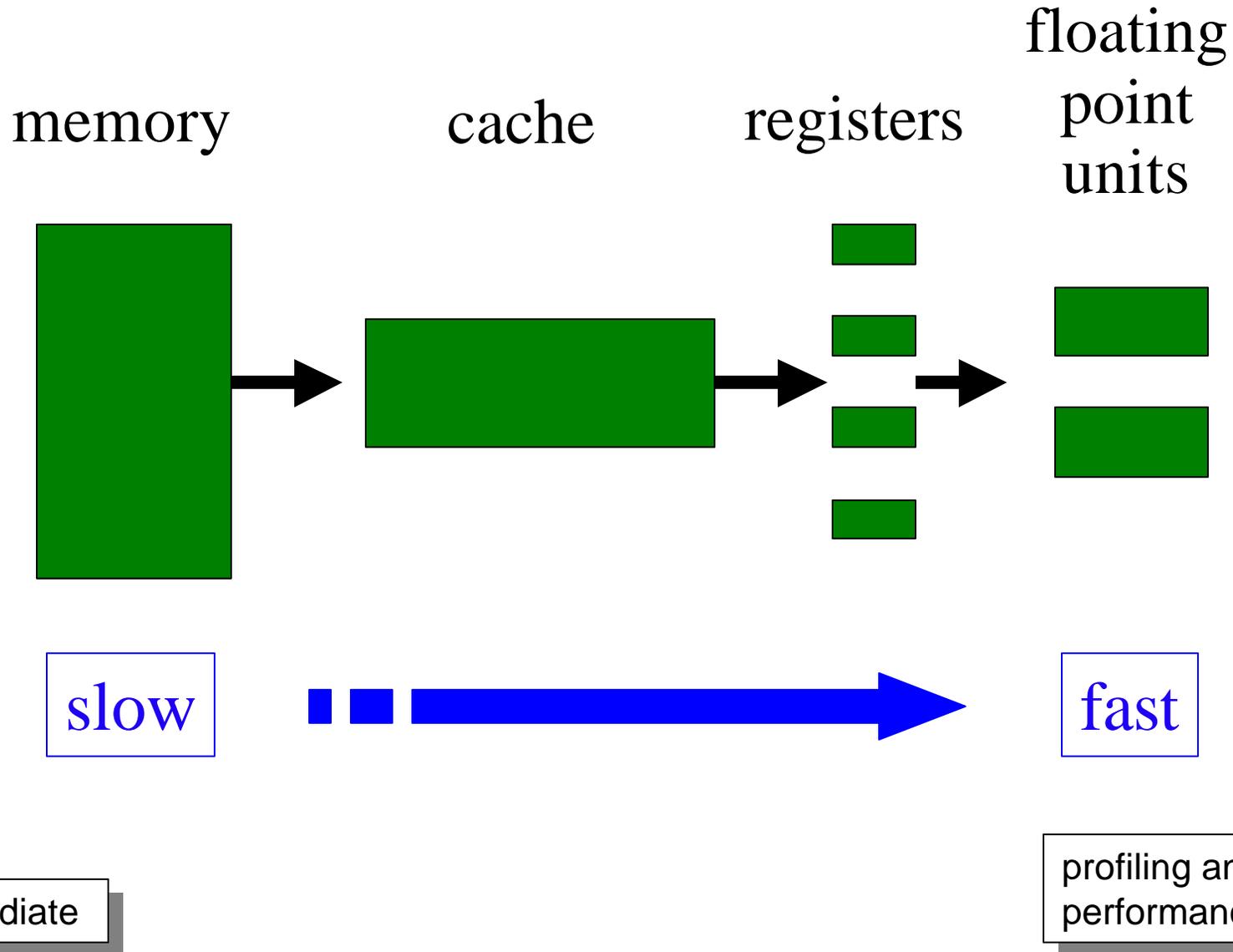
# Application Code Optimizations

Coding techniques to improve performance of user code on cache-based RISC architectures

intermediate

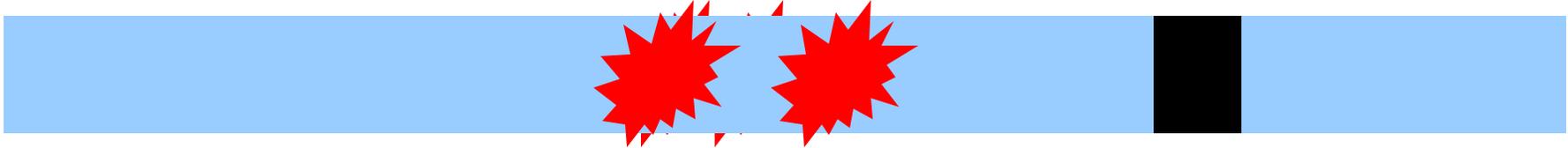
profiling and performance tuning

# Cache-based CPUs



# Variable Interleaving

- real  $u(\max n)$ ,  $v(\max n)$   
...  
 $u(i) = u(i) + a * u(i-n)$   
 $v(i) = v(i) + b * v(i-n)$
- Consider direct-mapped cache:



*Doubles* number of cache misses

Halves performance

*n*-way associative caches defeated by *n*+1 components

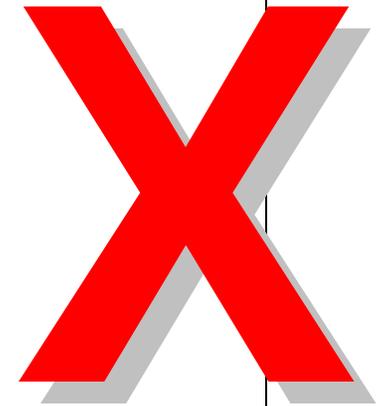
Many problems have at least four to five components per point

intermediate

profiling and  
performance tuning

# Techniques: Interleaving Field Variables

- Not
  - `double u[ ], double v[ ], double p[ ], ...`
- Not
  - `double precision fields(largesize,nc)`



- Instead
  - `double precision fields(nc,largesize)`

intermediate

profiling and  
performance tuning

# Techniques: Reordering to Reuse Cache Data

- Sort objects so that repeated use occurs together
  - e.g., sort edges by the first vertex they contain
- Reorder to reduce matrix bandwidth
  - e.g., RCM ordering

intermediate

profiling and  
performance tuning

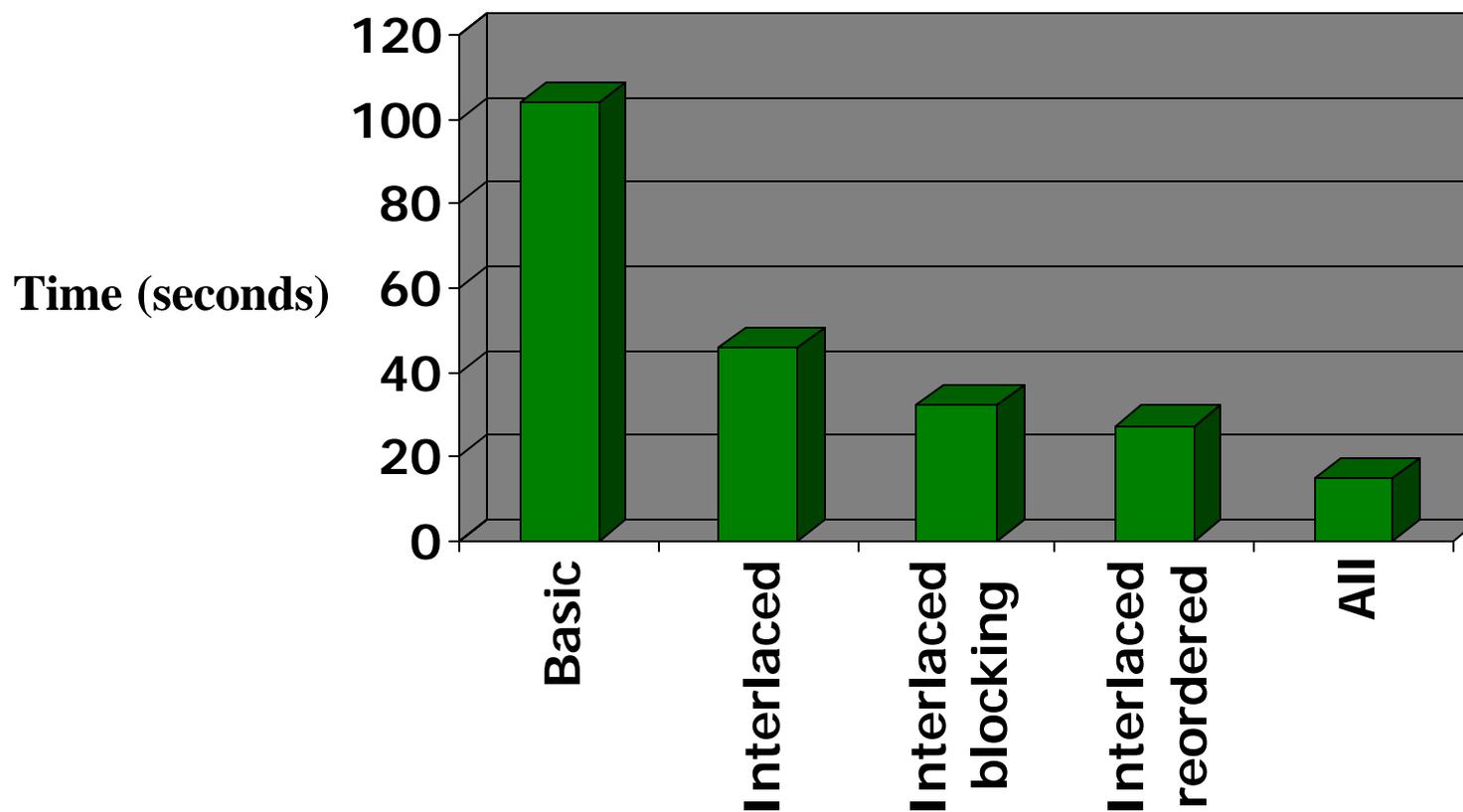
# Effects on Performance: Sample

- Incompressible Euler code
- Unstructured grid
- Edge-based discretization (i.e., computational loops are over edges)
- Block size 4
- Results collated by Dinesh Kaushik (ODU)

intermediate

profiling and  
performance tuning

# Effects on Performance: Sample Results on IBM SP



intermediate

profiling and performance tuning

# Other Features of PETSc

- Summary
- New features
- Interfacing with other packages
- Extensibility issues
- References

tutorial outline:  
conclusion

# Summary

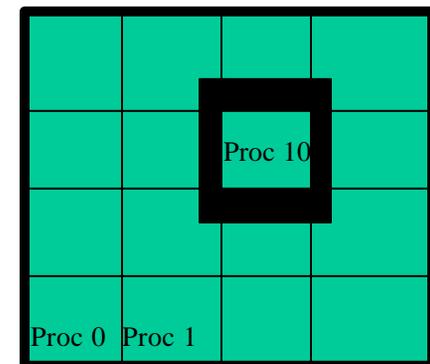
- Creating data objects
- Setting algorithmic options for linear, nonlinear and ODE solvers
- Using callbacks to set up the problems for nonlinear and ODE solvers
- Managing data layout and ghost point communication
- Evaluating parallel functions and Jacobians
- Consistent profiling and error handling

# New Features

- Version 2.1.x
  - Simple interface for multigrid on structured meshes
  - VecPack – manages treating several distinct vectors as one
    - useful for design optimization problems written as a nonlinear system
  - Parallel interface to SuperLU
- Next release
  - Automatically generated Jacobians via ADIC and ADIFOR
    - Fully automated for structured mesh parallel programs using DAs
    - General parallel case under development
- Under development
  - Interface to SLEPc eigenvalue software under development by V. Hernandez and J. Roman
  - Support for ESI interfaces (see <http://z.ca.sandia.gov/esi>)
  - Support for CCA-compliant components (see <http://www.cca-forum.org>)

## Multigrid Structured Mesh Support: DMMG: New Simple Interface

- General multigrid support
  - PC framework wraps MG for use as preconditioner
    - See MGSetXXX(), MGGetXXX()
    - Can access via -pc\_type mg
  - User provides coarse grid solver, smoothers, and interpolation/restriction operators
- DMMG - simple MG interface for structured meshes
  - User provides
    - “Local” function evaluation
    - [Optional] local Jacobian evaluation



## Multigrid Structured Mesh Support: Sample Function Computation

```
int Function(DALocalInfo *info,double **u,double **f,AppCtx *user)
...
lambda = user->param;
hx = 1.0/(info->mx-1);
hy = 1.0/(info->my-1);
for (j=info->ys; j<info->ys+info->ym; j++) {
 for (i=info->xs; i<info->xs+info->xm; i++) {
 f[j][i] = ... u[j][i] ... u[j-1][i]
 }
}
```

## Multigrid Structured Mesh Support: Sample Jacobian Computation

```
int Jacobian (DALocalInfo *info,double **u,Mat J,AppCtx *user)
 MatStencil mrow,mcols[5];
 double v[5];
 ...
 for (j=info->ys; j<info->ys+info->ym; j++) {
 row.j = j;
 for (i=info->xs; i<info->xs+info->xm; i++) {
 v[0] = ...; col[0].j = j - 1; col[0].i = i;
 v[1] = ...; col[1].j = j; col[1].i = i-1;
 v[2] = ...; col[2].j = j; col[2].i = i;
 v[3] = ...; col[3].j = j; col[3].i = i+1;
 v[4] = ...; col[4].j = j + 1; col[4].i = i;
 MatSetValuesStencil(jac,1,&row,5,col,v,INSERT_VALUES);
 }
 }
```

## Multigrid Structured Mesh Support: Nonlinear Example

- 2-dim **nonlinear** problem with 4 degrees of freedom per mesh point
- **Function()** and **Jacobian()** are user-provided functions

```
DMMG dmmg;
DMMGCreate(comm,nlevels,user,&dmmg)
DACreate2d(comm,DA_NONPERIODIC,DA_STENCIL_STAR,4,
 4,PETSC_DECIDE,PETSC_DECIDE,4,1,0,0,&da)
DMMGSetDM(dmmg,da)
DMMGSetSNESLocal(dmmg,Function,Jacobian,0,0)
DMMGSolve(dmmg)
solution = DMMGGetx(damg)
```

All standard **SNES**, **SLES**, **PC** and **MG** options apply.

# Multigrid Structured Mesh Support: Jacobian via Automatic Differentiation

- Collaboration with P. Hovland and B. Norris (see <http://www.mcs.anl.gov/autodiff>)
- Additional alternatives
  - Compute sparse Jacobian explicitly using AD  
`DMMGSetSNESLocal(dmmg,Function,0,ad_Function,0)`
    - PETSc + ADIC automatically generate `ad_Function`
  - Provide a “matrix-free” application of the Jacobian using AD  
`DMMGSetSNESLocal(dmmg,Function, 0,0,admf_Function)`
    - PETSc + ADIC automatically generate `admf_Function`
- Similar situation for Fortran and ADIFOR

# Using PETSc with Other Packages

- Linear algebra solvers
  - AMG
  - BlockSolve95
  - DSCPACK
  - hypre
  - ILUTP
  - LUSOL
  - SPAI
  - SPOOLES
  - SuperLU, SuperLU\_Dist
- Optimization software
  - TAO
  - Veltisto
- Mesh and discretization tools
  - Overture
  - SAMRAI
  - SUMAA3d
- ODE solvers
  - PVODE
- Others
  - Matlab
  - ParMETIS

# Using PETSc with Other Packages: Linear Solvers

- **Interface Approach**

- Based on interfacing at the matrix level, where external linear solvers typically use a variant of compressed sparse row matrix storage

- **Usage**

- Install PETSc indicating presence of any optional external packages in the file `petsc/bmake/$PETSC_ARCH/base.site`, e.g.,
  - `PETSC_HAVE_SPAI = -DPETSC_HAVE_SPAI`
  - `SPAI_INCLUDE = -I/home/username/software/spai_3.0/include`
  - `SPAI_LIB = /home/username/software/spai_3.0/lib/${PETSC_ARCH}/libspai.a`
- Set preconditioners via the usual approach
  - Procedural interface: `PCSetType(pc,"spai")`
  - Runtime option: `-pc_type spai`
- Set preconditioner-specific options via the usual approach, e.g.,
  - `PCSPAISetEpsilon()`, `PCSPAISetVerbose()`, etc.
  - `-pc_spai_epsilon <eps> -pc_spai_verbose etc.`

# Using PETSc with Other Packages: Linear Solvers

- **AMG**
  - Algebraic multigrid code by J. Ruge, K. Steuben, and R. Hempel (GMD)
  - <http://www.mgnet.org/mgnet-codes-gmd.html>
  - PETSc interface by D. Lahaye (K.U.Leuven), uses MatSeqAIJ
- **BlockSolve95**
  - Parallel, sparse ILU(0) for symmetric nonzero structure and ICC(0)
  - M. Jones (Virginia Tech.) and P. Plassmann (Penn State Univ.)
  - <http://www.mcs.anl.gov/BlockSolve95>
  - PETSc interface uses MatMPIRowbs
- **ILUTP**
  - Drop tolerance ILU by Y. Saad (Univ. of Minnesota), in SPARSKIT
  - <http://www.cs.umn.edu/~saad/>
  - PETSc interface uses MatSeqAIJ

# Using PETSc with Other Packages: Linear Solvers (cont.)

- LUSOL

- Sparse LU, part of MINOS
- M. Saunders (Stanford Univ)
- <http://www.sbsi-sol-optimize.com>
- PETSc interface by T. Munson (ANL), uses MatSeqAIJ

- SPAI

- Sparse approximate inverse code by S. Barnhard (NASA Ames) and M. Grote (ETH Zurich)
- <http://www.sam.math.ethz.ch/~grote/spai>
- PETSc interface converts from any matrix format to SPAI matrix

- SuperLU

- Parallel, sparse LU
- J. Demmel, J. Gilbert, (U.C. Berkeley) and X. Li (NERSC)
- <http://www.nersc.gov/~xiaoye/SuperLU>
- PETSc interface uses MatSeqAIJ
- Currently only sequential interface supported; parallel interface under development

# 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>)
  - Support for ESI interfaces to various linear algebra libraries (see <http://z.ca.sandia.gov/esi>)

# 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: PVIDE – ODE Integrators

- PVIDE
  - Parallel, robust, variable-order stiff and non-stiff ODE integrators
  - A. Hindmarsh et al. (LLNL)
  - <http://www.llnl.gov/CASC/PVIDE>
  - L. Xu developed PVIDE/PETSc interface
- Interface Approach
 

|                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>– PVIDE               <ul style="list-style-type: none"> <li>• ODE integrator – evolves field variables in time</li> <li>• vector – holds field variables</li> <li>• preconditioner placeholder</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>– PETSc               <ul style="list-style-type: none"> <li>• ODE integrator placeholder</li> <li>• vector</li> <li>• sparse matrix and preconditioner</li> </ul> </li> </ul> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
- Usage
  - TSCreate(MPI\_Comm, TS\_NONLINEAR, &ts)
  - TSSetType(ts, TS\_PVIDE)
  - ..... regular TS functions
  - TSPVIDESetType(ts, PVIDE\_ADAMS)
  - .... other PVIDE options
  - TSSetFromOptions(ts) – accepts PVIDE options

# Using PETSc with Other Packages: Mesh Management and Discretization

- **SUMAA3d**
  - Scalable Unstructured Mesh Algorithms and Applications
  - L. Freitag (ANL), M. Jones (VA Tech), P. Plassmann (Penn State)
  - <http://www.mcs.anl.gov/sumaa3d>
  - L. Freitag and M. Jones developed SUMAA3d/PETSc interface
- **SAMRAI**
  - Structured adaptive mesh refinement
  - R. Hornung, S. Kohn (LLNL)
  - <http://www.llnl.gov/CASC/SAMRAI>
  - SAMRAI team developed SAMRAI/PETSc interface
- **Overture**
  - Structured composite meshes and discretizations
  - D. Brown, W. Henshaw, D. Quinlan (LLNL)
  - <http://www.llnl.gov/CASC/Overture>
  - K. Buschelman and Overture team developed Overture/PETSc interfaces

# 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 that allows C/Fortran programmers to use Matlab functions in programs
    - PetscMatlabEngine – unwraps PETSc vectors and matrices so that MatlabEngine can understand them
- 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)

# Recent Additions

- Hypre ([www.llnl.gov/casc/hypre](http://www.llnl.gov/casc/hypre)) via PCHYPRE, includes
  - EUCLID (parallel ILU(k) by David Hysom)
  - BoomerAMG
  - PILUT
- DSCPACk, a Domain-Separator Cholesky Package for solving sparse spd problems, by Padma Raghavan
- SPOOLES (SParse Object Oriented Linear Equations Solver), by Cleve Ashcraft
- Both SuperLU and SuperLU\_Dist sequential and parallel direct sparse solvers, by Xiaoye Li et al.

# Extensibility Issues

- Most PETSc objects are designed to allow one to “drop in” a new implementation with a new set of data structures (similar to implementing a new class in C++).
- Heavily commented example codes include
  - Krylov methods: [petsc/src/sles/ksp/impls/cg](#)
  - preconditioners: [petsc/src/sles/pc/impls/jacobi](#)
- Feel free to discuss more details with us in person.

## Caveats Revisited

- Developing parallel, non-trivial 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.
- Users are invited to interact directly with us regarding correctness and performance issues by writing to [petsc-maint@mcs.anl.gov](mailto:petsc-maint@mcs.anl.gov).

# 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