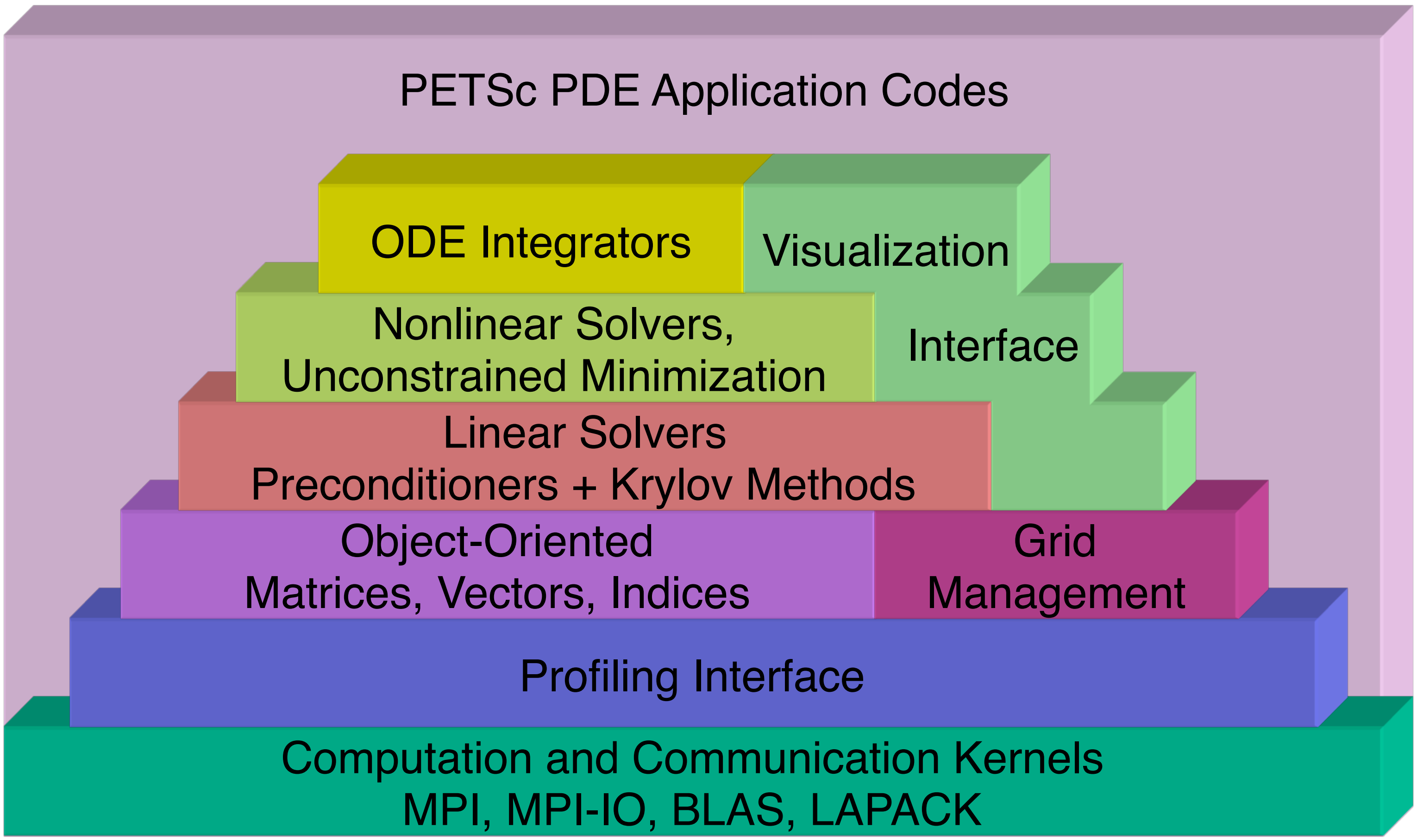
PETSc

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

Satish Balay,
Kris Buschelman, Bill Gropp,
Dinesh Kaushik, Lois McInnes,
Barry Smith

PDE Application Codes



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	Chebychev	Other

Preconditioners

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

Matrices

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

Structured Meshes

Unstructured Meshes

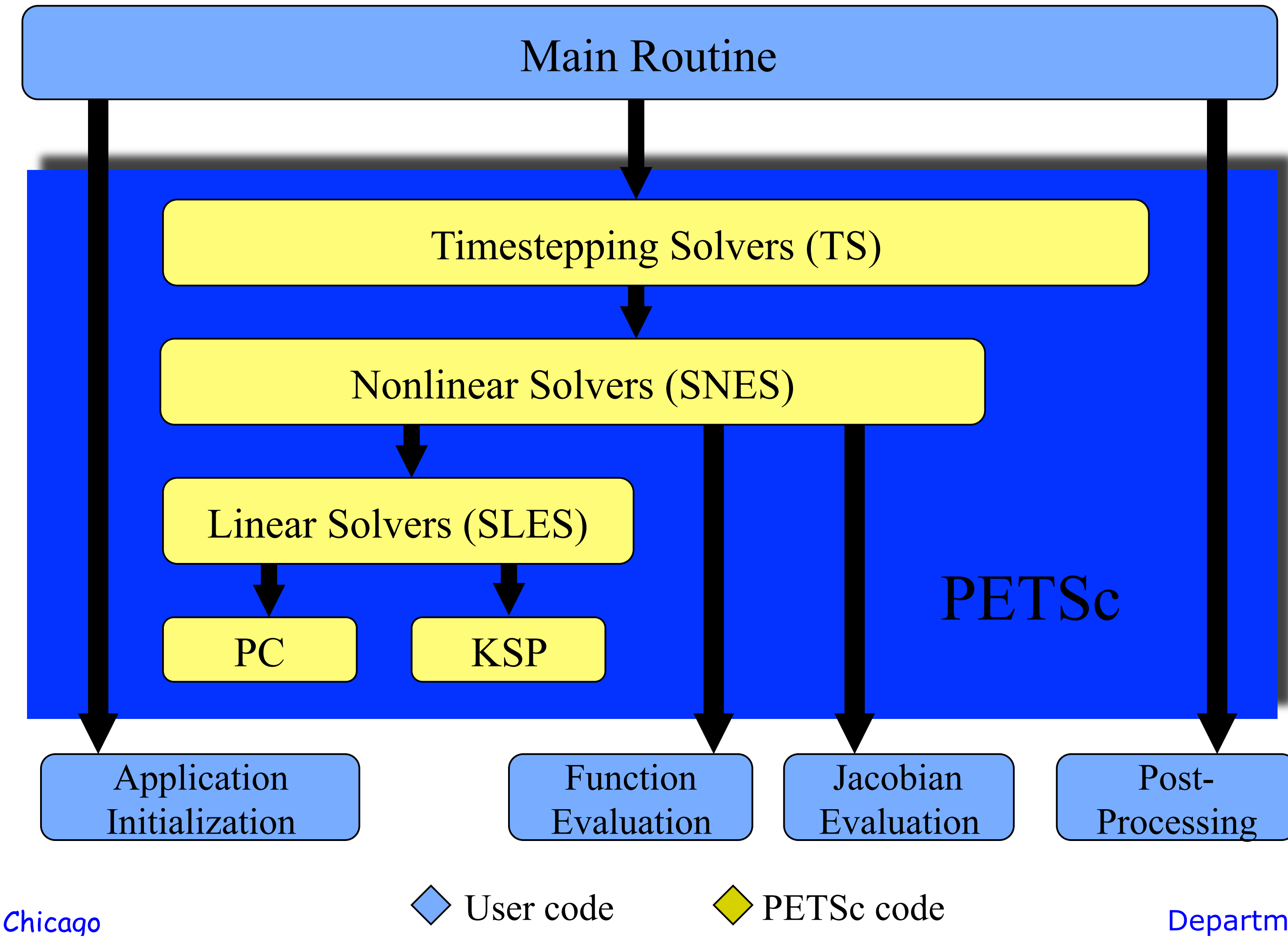
Distributed Arrays

Index Sets

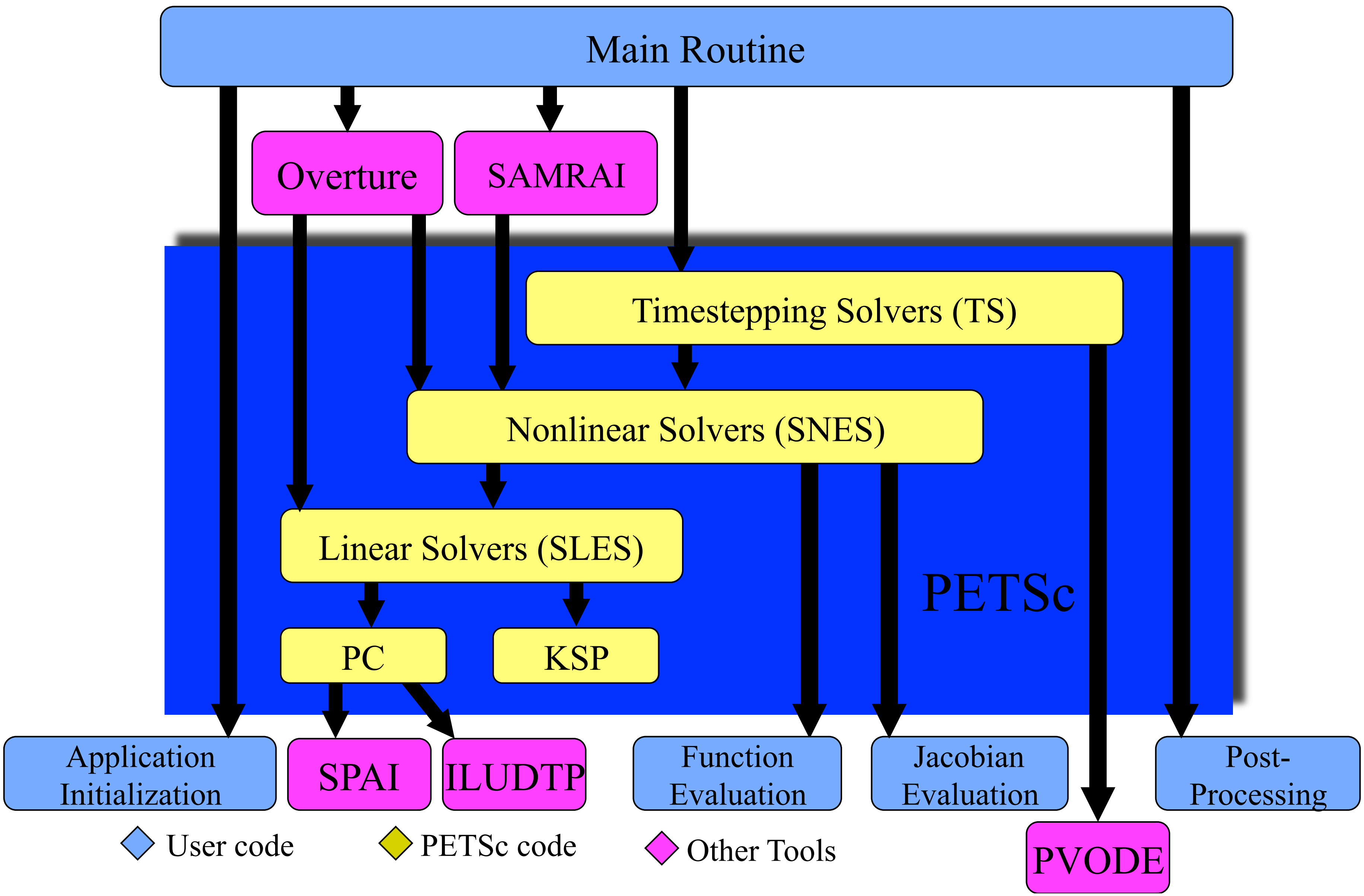
Indices	Block Indices	Stride	Other

Vectors

Flow of Control for PDE Solution



Flow of Control for PDE Solution



Ease of Integration With Existing Programs in Fortran, C, C++

- Library with conventional procedural interface
- Can use user-structures, preconditioners, matrix-vector multiplication routines
- Used as solver in Whitfield, Fun3D code (legacy Fortran apps)

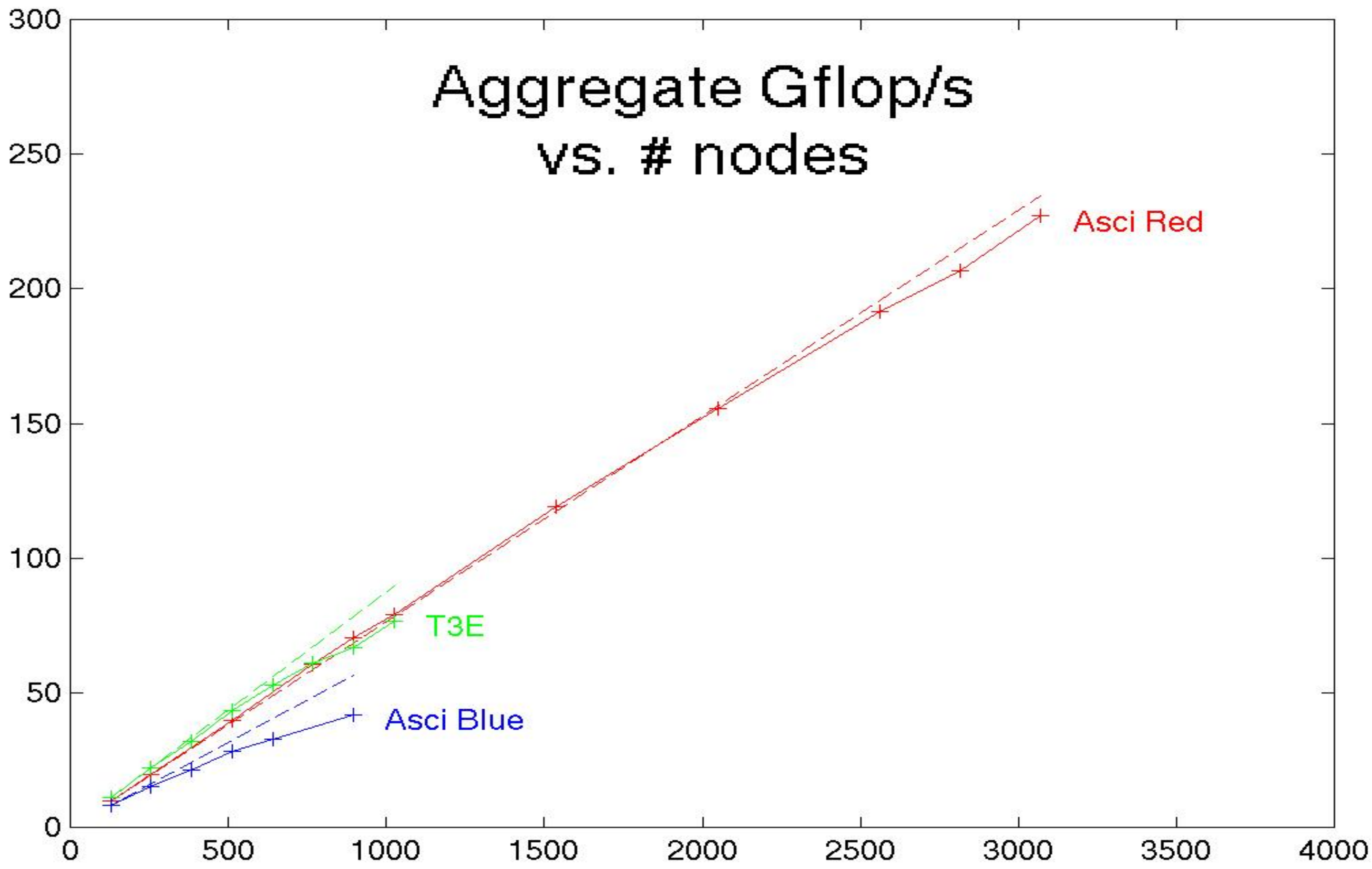
PETSc Philosophy

- Writing hand-parallelized application codes from scratch is extremely difficult and time consuming.
- Scalable parallelizing compilers for real application codes are very far in the future.
- We can ease the development of parallel application codes by developing general-purpose, parallel numerical PDE libraries.
- Caveats
  - Developing parallel, non-trivial PDE solvers that deliver high performance is still difficult, and requires months of concentrated effort.
  - PETSc is a toolkit that can reduce the development time, but it is not a black-box PDE solver nor a silver bullet.

Performance

- Optimized for multicomponent, not scalar problems
- Provides tools for measuring and improving performance
- A PETSc application won a Gordon Bell prize in 1999, achieving >220 GF on an unstructured mesh application

Fixed-size Parallel Scaling Results for Unstructured Mesh CFD Application



Scalability and Parallelism

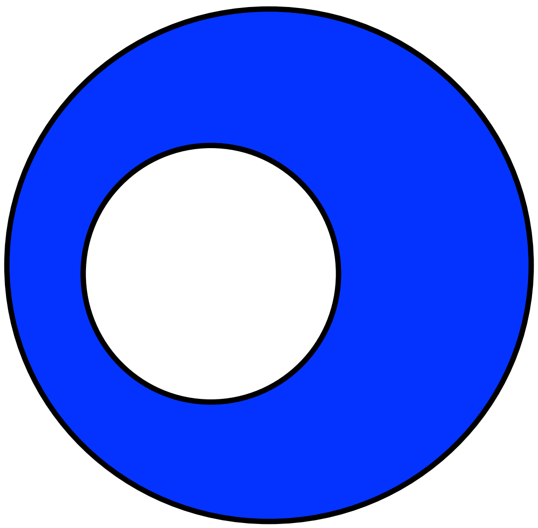
- Scalable to 1000's of processors
- Typical use on dozens to hundreds
- Single processor support
  - MPI not required
  - Develop on your workstation (even Windows Laptop!)
  - Run on a local cluster
  - Run on an MPP at a national lab

Documentation and Example Programs

- Extensive man pages
- Petsc examples by concept

Sample Linear Application:

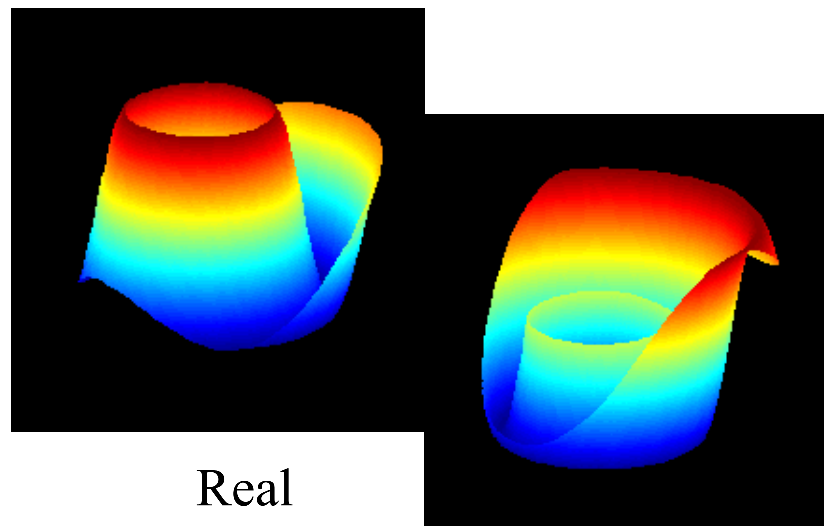
Exterior Helmholtz Problem



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

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

Solution Components



Real

Imaginary

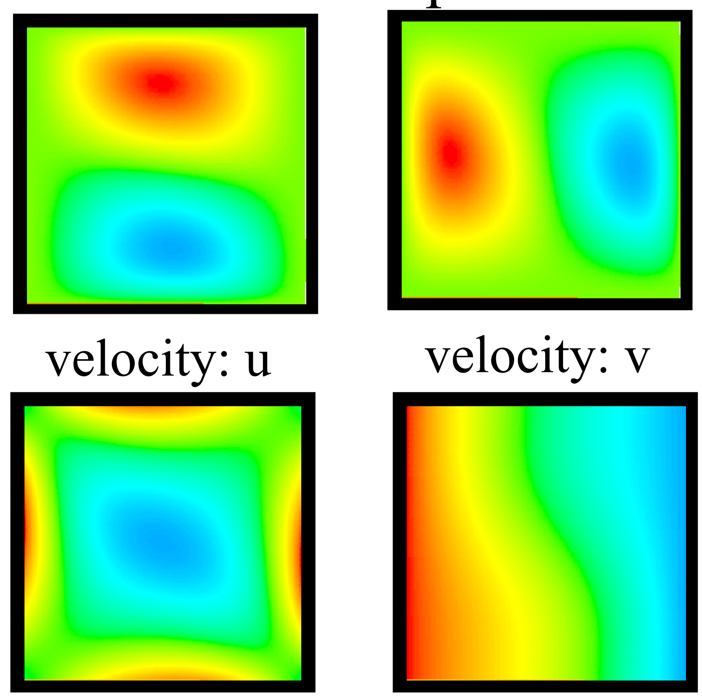
Collaborators: H. M. Atassi, D. E. Keyes,
L. C. McInnes, R. Susan-Resiga

Sample Nonlinear Application:
Driven Cavity Problem

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

petsc/src/snes/examples/tutorials/ex8.c

Solution Components



temperature: T

Application code author: D. E. Keyes

Caveats

- 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 or performance issues by writing to petsc-maint@mcs.anl.gov.

Using Petsc With Other Packages

- **PVODE** – ODE integrator
  - A. Hindmarsh et al. - http://www.llnl.gov/CASC/PVODE
- **ILUDTP** – drop tolerance ILU
  - Y. Saad - http://www.cs.umn.edu/~saad
- **ParMETIS** – parallel partitioner
  - G. Karypis - http://www.cs.umn.edu/~karypis
- **Overture** – composite mesh PDE package
  - D. Brown, W. Henshaw, and D. Quinlan - http://www.llnl.gov/CASC/Overture
- **SAMRAI** – AMR package
  - S. Kohn, X. Garaiza, R. Hornung, and S. Smith - http://www.llnl.gov/CASC/SAMRAI
- **SPAI** – sparse approximate inverse preconditioner
  - S. Bernhard and M. Grote - http://www.sam.math.ethz.ch/~grote/spai
- **Matlab**
  - http://www.mathworks.com
- **TAO** – optimization software
  - S. Benson, L.C. McInnes, and J. Moré - http://www.mcs.anl.gov/tao

- Backup slides

A Freely Available and Supported Research Code

- Available via http://www.mcs.anl.gov/petsc
- Usable in C, C++, and Fortran77/90 (with minor limitations in Fortran 77/90 due to their syntax)
- Users manual
- Hyperlinked manual pages for all routines
- Many tutorial-style examples
- Support via email: petsc-maint@mcs.anl.gov

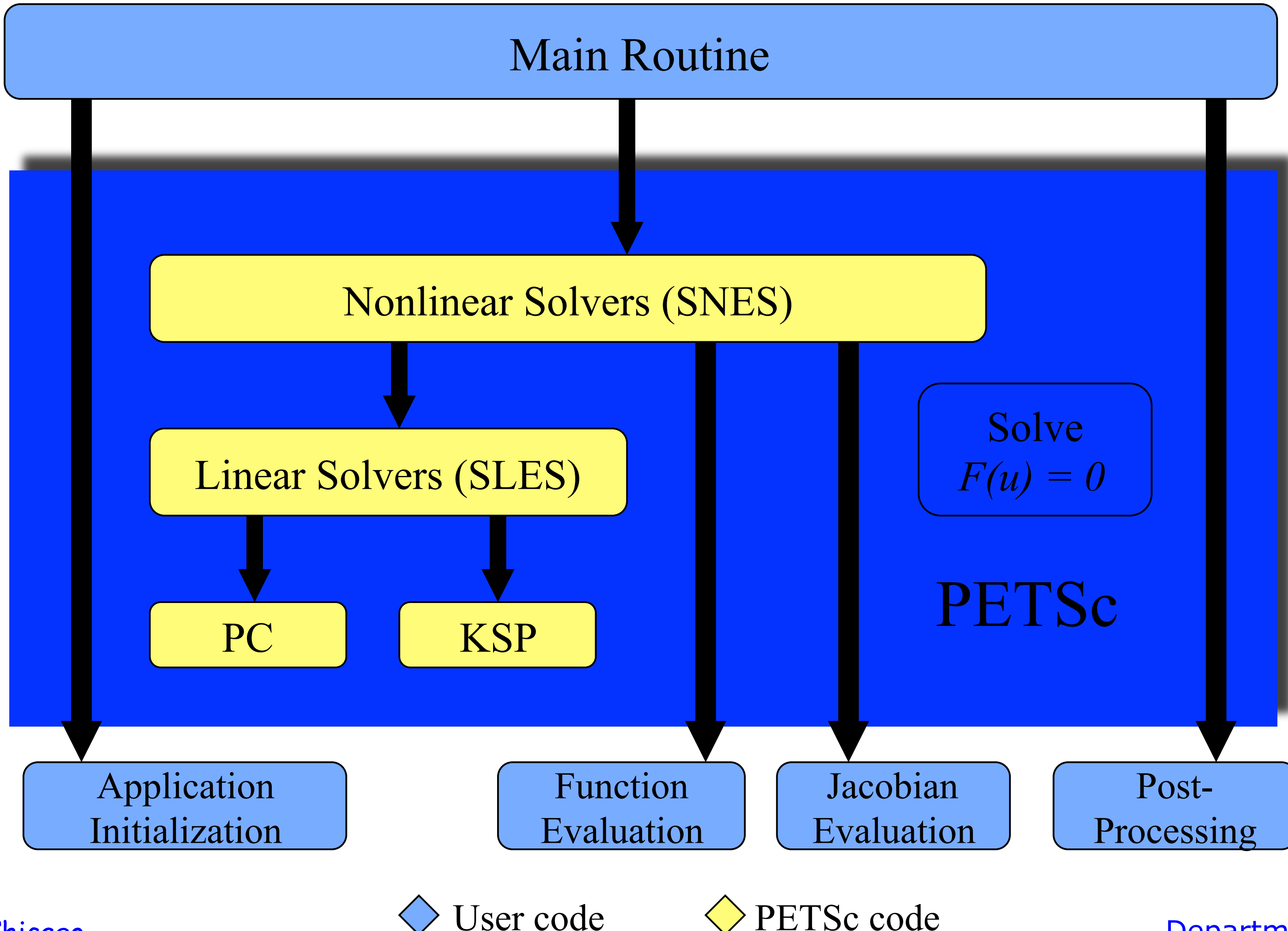
True Portability

- Tightly coupled systems
  - Cray T3D/T3E
  - SGI/Origin
  - IBM SP
  - Convex Exemplar
- Loosely coupled systems, e.g., networks of workstations
  - Sun OS, Solaris
  - IBM AIX
  - DEC Alpha
  - HP
  - Linux
  - Freebsd
  - Windows 98/2000
  - Mac OS X
  - BeOS

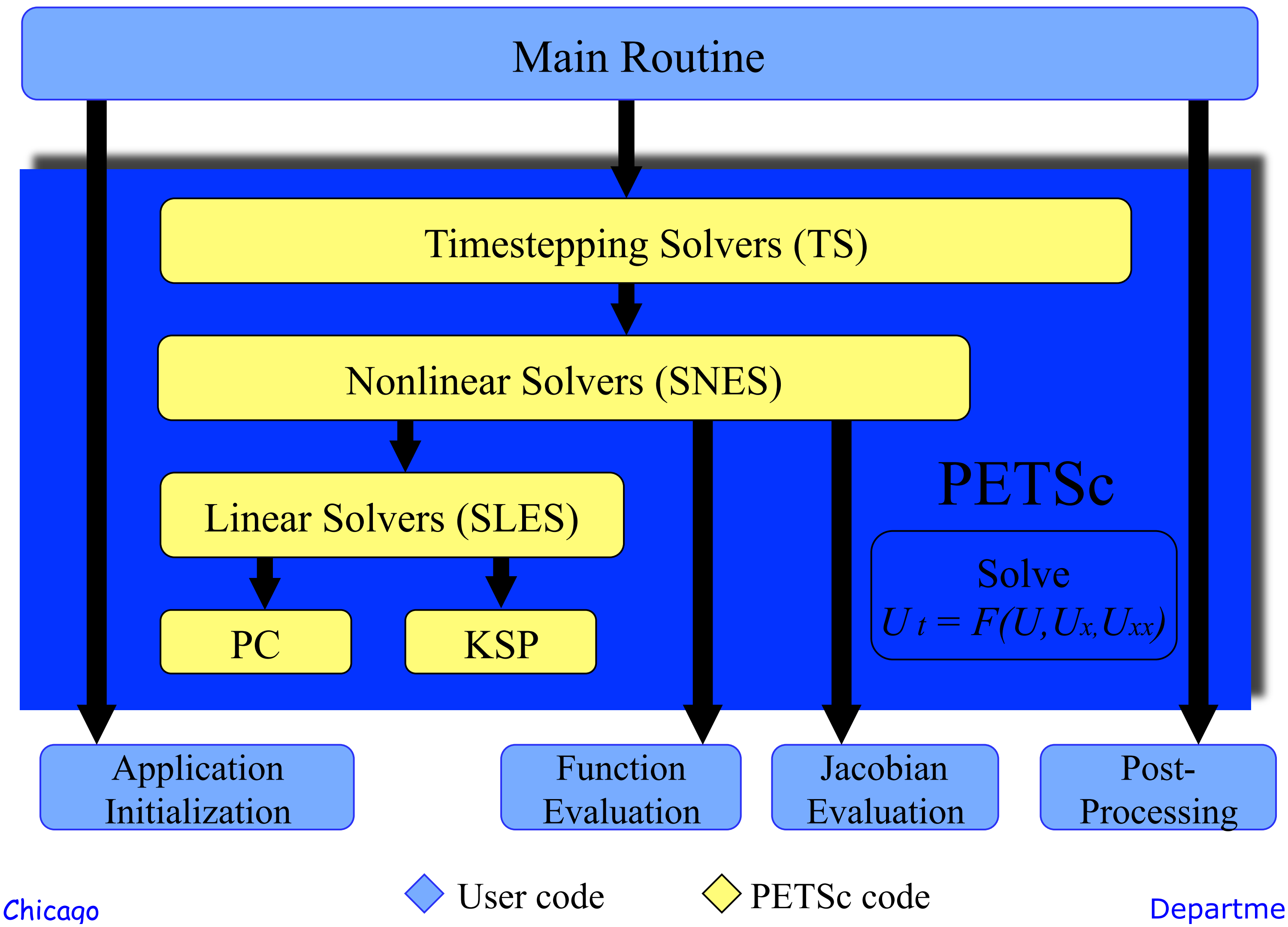
Using PETSc

- 2 Lifecycle example:
- First: new code
  - Express problem
    - Get discretization from elsewhere
    - Use PETSc TS/SNES/SLES to solve as appropriate
    - Use DA, VecScatter tools for parallelism (Distributed Memory parallelism)
    - Use PETSc profiling/logging to improve efficiency
    - Use PETSc algorithm-independent formulation to explore alternatives
- Second: Updating Legacy Code
  - Find TS/SNES/SLES step
    - (look for *outer*, not inner)
    - Replace with appropriate PETSc call
    - Use PETSc tools to match legacy data structures with PETSc
      - May need changes for performance

Nonlinear PDE Solution



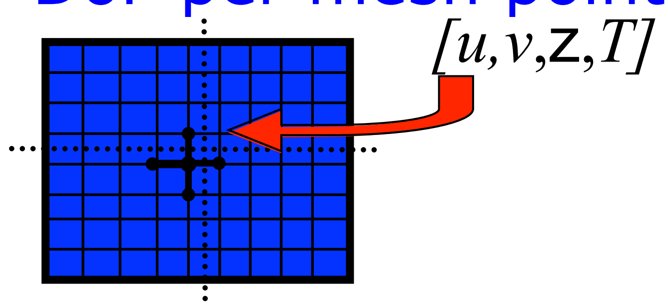
Time-Dependent PDE Solution



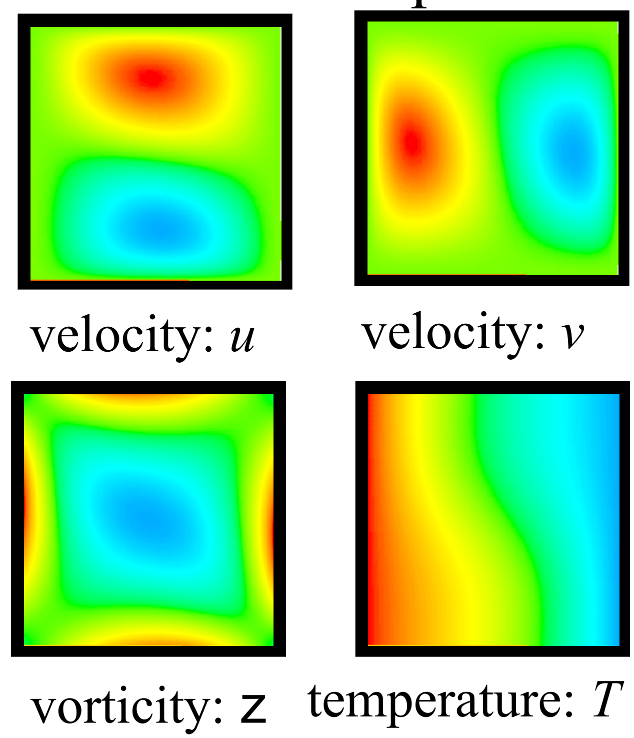
Driven Cavity Model

Example code: petsc/src/snes/examples/tutorials/ex8.c

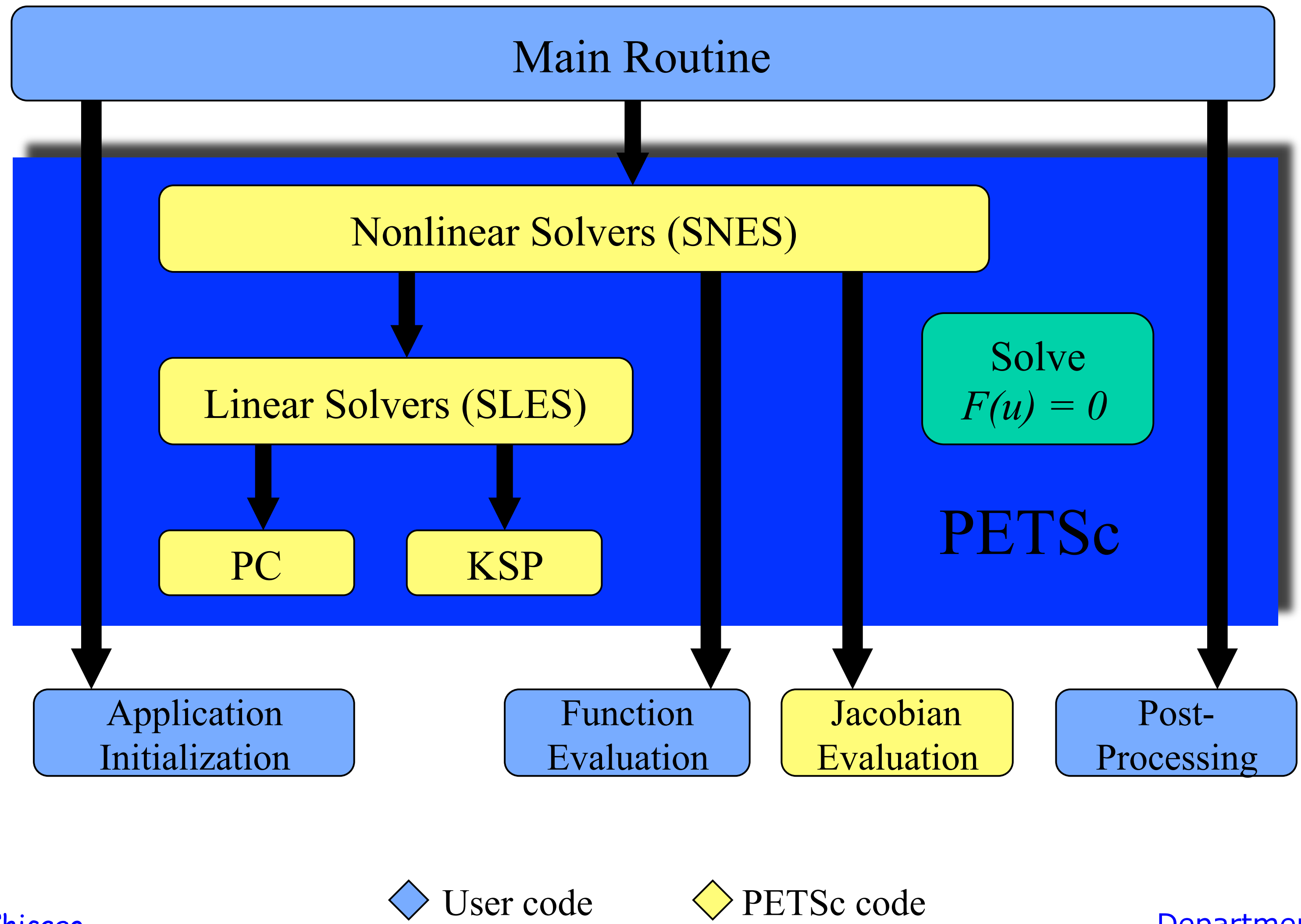
- Velocity-vorticity formulation, with flow driven by lid and/or bouyancy
- Finite difference discretization with 4 DoF per mesh point



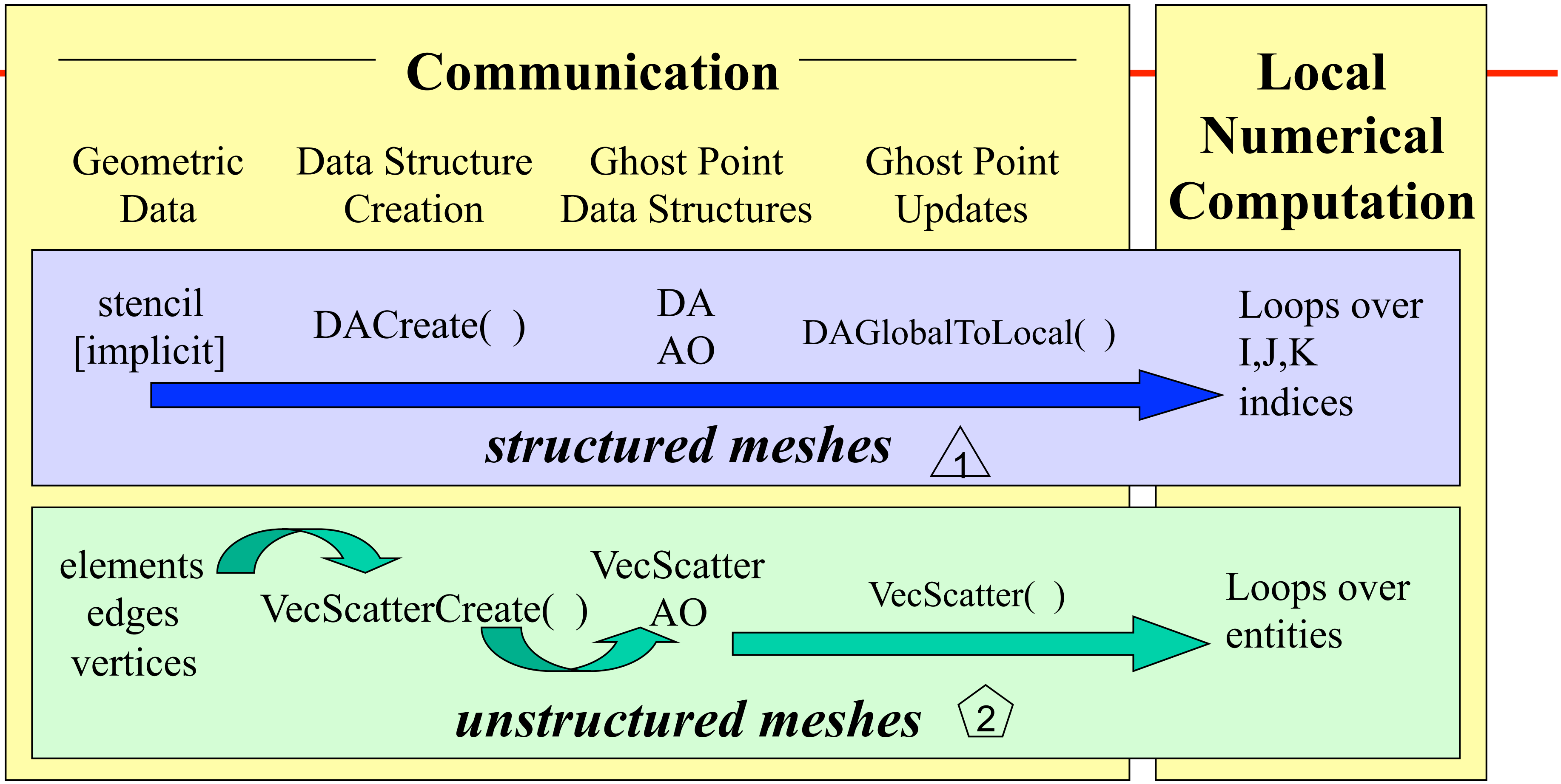
Solution Components



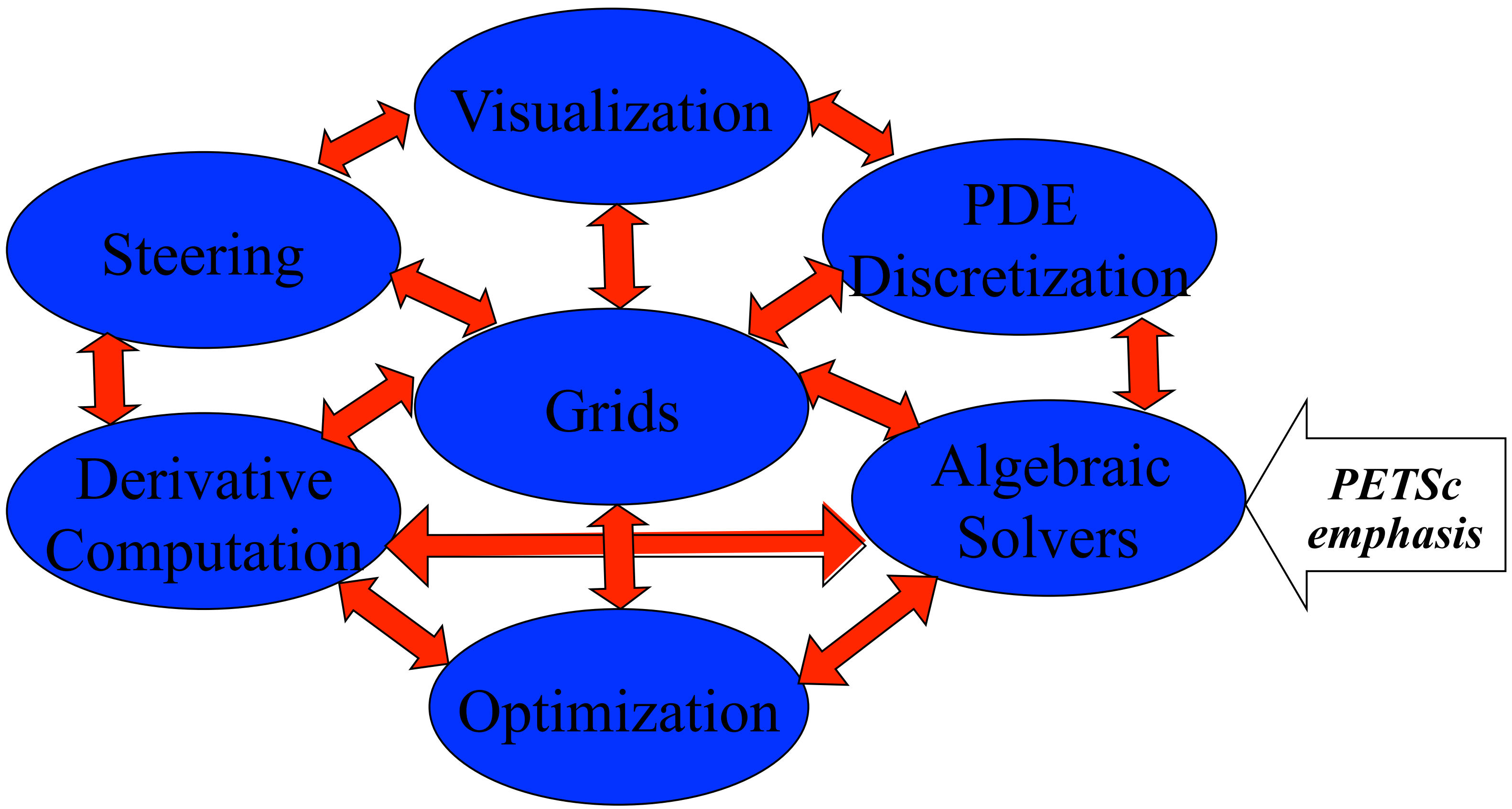
Driven Cavity Solution Approach



Communication and Physical Discretization



Component Interactions for Numerical PDEs



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., OpenMP)
  - Hide within parallel objects the details of the communication
  - User orchestrates communication at a higher abstract level than message passing

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

Performance Issues

- Flexible design to allow experimentation.
- Do certain optimizations after analyzing performance.
- Use –log_summary as a tool, but always use API, tuned for high performance.
- Modular design enables multiple implementations of the same component (AIJ,BAIJ etc..)
- Machine specific optimizations possible (using fortran kernels, for loops etc..)
- Create once and reuse – Scatters, factorizations etc..
- Pay attention to data layout/cache issues.

Other Issues

- Object header, creation, composition, dynamic methods etc.
- Extensive and consistent error handling.
- Profiling interface – application information, performance.
- Fortran interface/Fortran 90 support.
- Viewers – to debug/visualize PETSc objects.
- Interoperability with BlockSolve, PVode, Overture.
- Alice memory snooper(AMS), Toolkit for Advanced Optimization(TAO).