# Runtime Configurability in PETSc

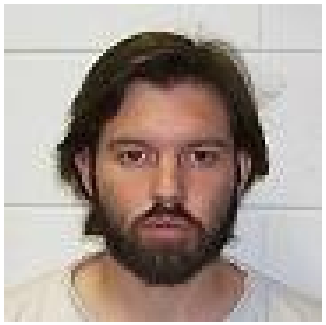
Matthew Knepley $\in$ PETSc Team

Computation Institute
University of Chicago

Department of Molecular Biology and Physiology
Rush University Medical Center

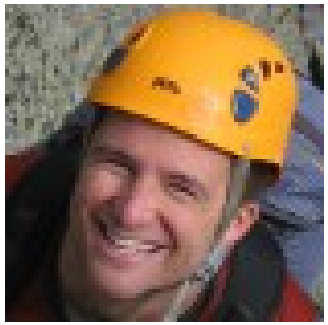
SIAM Conference on Parallel Processing and Scientific Computing
Portland, OR February 20, 2014


# The PETSc Team


Matt Knepley


Barry Smith


Satish Balay


Jed Brown

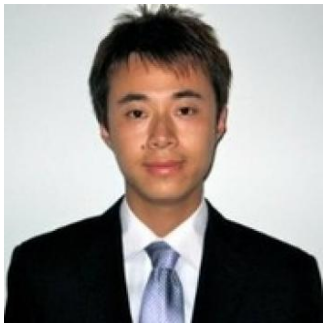

Hong Zhang

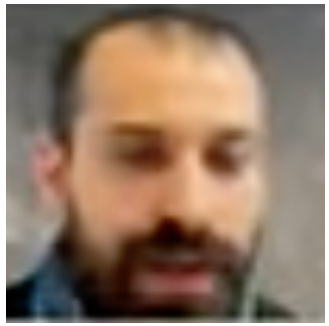

Lisandro Dalcin


Stefano Zampini


Mark Adams


Toby Issac


Hong Zhang


Pierre Jolivet


Junchao Zhang

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ, KSPGMRES, SNESFAS, DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# What can be Configured in PETSc?

- Object behavior
  - Tolerances, subspace sizes, preallocation, . . .
  - Eisenstat-Walker tolerances for **SNES**
- Concrete object types
  - **MATAIJ**, **KSPGMRES**, **SNESFAS**, **DMPLEX**
  - User-defined types
- Object organization
  - Linear & Nonlinear preconditioners
  - Number of splits in block methods
  - Explicit/Semi-implicit/Implicit divison for **TS**

# Why Configure at Runtime?

## Vectors and Matrices

- architecture
- problem/discretization

# Why Configure at Runtime?

Solvers

- the equation and boundary conditions
- domain
- discretization
- solution evolution
  - nonlinear feedback
  - dynamic instabilities
  - strong or emergent anisotropy
  - resonance

# Why Configure at Runtime?

Solvers

- the equation and boundary conditions
- domain
- discretization
- solution evolution
  - nonlinear feedback
  - dynamic instabilities
  - strong or emergent anisotropy
  - resonance

# Why Configure at Runtime?

Arguments against concrete types in applications:

**Programming Languages for Scientific Computing**

Encyclopedia of Applied and Computational Mathematics, Springer, 2012.

http://arxiv.org/abs/1209.1711

# Outline

1 Configuring PETSc

2 Extending PETSc

# User Solve

```
MPI_Comm comm;
SNES snes;
DM dm;
Vec u;

SNESCreate(comm, &snes);
SNESSetDM(snes, dm);
SNESSetFromOptions(snes);
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, NULL, u);
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e2
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view
```

$$
\begin{aligned}
-\Delta U - \partial_y \Omega &= 0 \\
-\Delta V + \partial_x \Omega &= 0 \\
-\Delta \Omega + \nabla \cdot ([U\Omega, V\Omega]) - \mathrm{Gr}\, \partial_x T &= 0 \\
-\Delta T + \mathrm{Pr}\, \nabla \cdot ([UT, VT]) &= 0
\end{aligned}
$$

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e2
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view

lid velocity = 100, prandtl # = 1, grashof # = 100
  0 SNES Function norm 768.116
  1 SNES Function norm 658.288
  2 SNES Function norm 529.404
  3 SNES Function norm 377.51
  4 SNES Function norm 304.723
  5 SNES Function norm 2.59998
  6 SNES Function norm 0.00942733
  7 SNES Function norm 5.20667e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 7
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e4
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e4
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view

lid velocity = 100, prandtl # = 1, grashof # = 10000
  0 SNES Function norm 785.404
  1 SNES Function norm 663.055
  2 SNES Function norm 519.583
  3 SNES Function norm 360.87
  4 SNES Function norm 245.893
  5 SNES Function norm 1.8117
  6 SNES Function norm 0.00468828
  7 SNES Function norm 4.417e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 7
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view

lid velocity = 100, prandtl # = 1, grashof # = 100000
  0 SNES Function norm 1809.96
Nonlinear solve did not converge due to DIVERGED_LINEAR_SOLVE iterations 0
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5
  -da_grid_x 16 -da_grid_y 16 -da_refine 2 -pc_type lu
  -snes_monitor_short -snes_converged_reason -snes_view

lid velocity = 100, prandtl # = 1, grashof # = 100000
  0 SNES Function norm 1809.96
  1 SNES Function norm 1678.37
  2 SNES Function norm 1643.76
  3 SNES Function norm 1559.34
  4 SNES Function norm 1557.6
  5 SNES Function norm 1510.71
  6 SNES Function norm 1500.47
  7 SNES Function norm 1498.93
  8 SNES Function norm 1498.44
  9 SNES Function norm 1498.27
 10 SNES Function norm 1498.18
 11 SNES Function norm 1498.12
 12 SNES Function norm 1498.11
 13 SNES Function norm 1498.11
 14 SNES Function norm 1498.11
 ...
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type newtonls -snes_converged_reason
 -pc_type lu
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
  1 SNES Function norm 1132.29
  2 SNES Function norm 1026.17
  3 SNES Function norm 925.717
  4 SNES Function norm 924.778
  5 SNES Function norm 836.867
  :
 21 SNES Function norm 585.143
 22 SNES Function norm 585.142
 23 SNES Function norm 585.142
 24 SNES Function norm 585.142
  :
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type fas -snes_converged_reason
 -fas_levels_snes_type gs -fas_levels_snes_max_it 6
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
  1 SNES Function norm 574.793
  2 SNES Function norm 513.02
  3 SNES Function norm 216.721
  4 SNES Function norm 85.949
Nonlinear solve did not converge due to DIVERGED_INNER iterations 4
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type fas -snes_converged_reason
 -fas_levels_snes_type gs -fas_levels_snes_max_it 6
  -fas_coarse_snes_converged_reason
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 12
  1 SNES Function norm 574.793
    Nonlinear solve did not converge due to DIVERGED_MAX_IT its 50
  2 SNES Function norm 513.02
    Nonlinear solve did not converge due to DIVERGED_MAX_IT its 50
  3 SNES Function norm 216.721
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 22
  4 SNES Function norm 85.949
    Nonlinear solve did not converge due to DIVERGED_LINE_SEARCH its 42
Nonlinear solve did not converge due to DIVERGED_INNER iterations 4
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type fas -snes_converged_reason
 -fas_levels_snes_type gs -fas_levels_snes_max_it 6
  -fas_coarse_snes_linesearch_type basic
  -fas_coarse_snes_converged_reason


lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  ⋮
 47 SNES Function norm 78.8401
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 5
 48 SNES Function norm 73.1185
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
 49 SNES Function norm 78.834
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 5
 50 SNES Function norm 73.1176
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  ⋮
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type nrichardson -npc_snes_max_it 1 -snes_converged_reason
 -npc_snes_type fas -npc_fas_coarse_snes_converged_reason
  -npc_fas_levels_snes_type gs -npc_fas_levels_snes_max_it 6
  -npc_fas_coarse_snes_linesearch_type basic
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  1 SNES Function norm 552.271
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 27
  2 SNES Function norm 173.45
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 45
  :
 43 SNES Function norm 3.45407e-05
    Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
 44 SNES Function norm 1.6141e-05
    Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
 45 SNES Function norm 9.13386e-06
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 45
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type ngmres -npc_snes_max_it 1 -snes_converged_reason
 -npc_snes_type fas -npc_fas_coarse_snes_converged_reason
  -npc_fas_levels_snes_type gs -npc_fas_levels_snes_max_it 6
  -npc_fas_coarse_snes_linesearch_type basic
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  1 SNES Function norm 538.605
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 13
  2 SNES Function norm 178.005
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 24
  :
 27 SNES Function norm 0.000102487
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 2
 28 SNES Function norm 4.2744e-05
    Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
 29 SNES Function norm 1.01621e-05
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 29
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type ngmres -npc_snes_max_it 1 -snes_converged_reason
 -npc_snes_type fas -npc_fas_coarse_snes_converged_reason
 -npc_fas_levels_snes_type newtonls -npc_fas_levels_snes_max_it 6
  -npc_fas_levels_snes_linesearch_type basic
  -npc_fas_levels_snes_max_linear_solve_fail 30
  -npc_fas_levels_ksp_max_it 20 -npc_fas_levels_snes_converged_reason
  -npc_fas_coarse_snes_linesearch_type basic
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve did not converge due to DIVERGED_MAX_IT its 6
    :
    :
        Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 1
    :
    :
  1 SNES Function norm 0.1935
  2 SNES Function norm 0.0179938
  3 SNES Function norm 0.00223698
  4 SNES Function norm 0.000190461
  5 SNES Function norm 1.6946e-06
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 5
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type composite -snes_composite_type additiveoptimal
 -snes_composite_sneses fas,newtonls -snes_converged_reason
 -sub_0_fas_levels_snes_type gs -sub_0_fas_levels_snes_max_it 6
   -sub_0_fas_coarse_snes_linesearch_type basic
 -sub_1_snes_linesearch_type basic -sub_1_pc_type mg


lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
  1 SNES Function norm 541.462
  2 SNES Function norm 162.92
  3 SNES Function norm 48.8138
  4 SNES Function norm 11.1822
  5 SNES Function norm 0.181469
  6 SNES Function norm 0.00170909
  7 SNES Function norm 3.24991e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 7
```

# Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type composite -snes_composite_type multiplicative
 -snes_composite_sneses fas,newtonls -snes_converged_reason
 -sub_0_fas_levels_snes_type gs -sub_0_fas_levels_snes_max_it 6
   -sub_0_fas_coarse_snes_linesearch_type basic
 -sub_1_snes_linesearch_type basic -sub_1_pc_type mg


lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
  1 SNES Function norm 544.404
  2 SNES Function norm 18.2513
  3 SNES Function norm 0.488689
  4 SNES Function norm 0.000108712
  5 SNES Function norm 5.68497e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 5
```

# Nonlinear Preconditioning

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|---|---|---|---|---|---|---|---|
| $(\mathcal{N}\backslash\mathrm{K}-\mathrm{MG})$ | 9.83 | 17 | 352 | 34 | 85 | 370 | – |
| NGMRES $-_R$ $(\mathcal{N}\backslash\mathrm{K}-\mathrm{MG})$ | 7.48 | 10 | 220 | 21 | 50 | 231 | 10 |
| FAS | 6.23 | 162 | 0 | 2382 | 377 | 754 | – |
| FAS $+$ $(\mathcal{N}\backslash\mathrm{K}-\mathrm{MG})$ | 8.07 | 10 | 197 | 232 | 90 | 288 | – |
| FAS $*$ $(\mathcal{N}\backslash\mathrm{K}-\mathrm{MG})$ | 4.01 | 5 | 80 | 103 | 45 | 125 | – |
| NRICH $-_L$ FAS | 3.20 | 50 | 0 | 1180 | 192 | 384 | 50 |
| NGMRES $-_R$ FAS | 1.91 | 24 | 0 | 447 | 83 | 166 | 24 |

# Nonlinear Preconditioning

## See discussion in:

Composing Scalable Nonlinear Algebraic Solvers,
Peter Brune, Matthew Knepley, Barry Smith, and Xuemin Tu,

SIAM Review, **57**(4), 535–565, 2015.

http://www.mcs.anl.gov/uploads/cels/papers/P2010-0112.pdf

# Outline

1 Configuring PETSc

2 Extending PETSc
- Creating a new Class Implementation
- Distributing your new Implementation

# Outline

2 Extending PETSc
- Creating a new Class Implementation
- Distributing your new Implementation

# Creating a Preconditioner (**PC**)

Include the private header for access to the **PC** struct:

```
#include<petsc-private/pcimpl.h>
```

# Creating a Preconditioner (**PC**)

## Define a struct for the concrete type:

```
typedef struct {
  /* The maximum and actual half-bandwidth */
  PetscInt  kmax, k;
  /* The limit and actual norm fraction */
  PetscReal frac, f;
  /* The banded approximation */
  Mat B;
  /* The embedded PC */
  PC pc;
} PC_Banded;
```

# Creating a Preconditioner (**PC**)

Register a constructor for the concrete type:

```
PCRegister("banded", PCCreate_Banded)
```

# Creating a Preconditioner (**PC**)

## Define the constructor:

```
PetscErrorCode PCCreate_Banded(PC pc)
{
  PC_Banded      *b;
  PetscErrorCode ierr;

  PetscFunctionBegin;
  /* Create concrete type struct */
  /* Setup function table for class */
  /* Setup concrete type-specific functions */
  /* Setup subobjects */
  PetscFunctionReturn(0);
}
```

# Creating a Preconditioner (**PC**)

## Define the constructor:

```
PetscErrorCode PCCreate_Banded(PC pc)
{
  PetscFunctionBegin;
  ierr = PetscNewLog(pc, &b);CHKERRQ(ierr);
  pc->data = (void *) b;
  b->kmax = 50;
  b->frac = 0.95;
  /* Setup function table for class */
  /* Setup concrete type-specific functions */
  /* Setup subobjects */
  PetscFunctionReturn(0);
}
```

# Creating a Preconditioner (**PC**)

## Define the constructor:

```
PetscErrorCode PCCreate_Banded(PC pc)
{
  PetscFunctionBegin;
  /* Create concrete type struct */
  pc->ops->apply               = PCApply_Banded;
  pc->ops->applytranspose      = NULL;
  pc->ops->setup               = PCSetUp_Banded;
  pc->ops->reset               = PCReset_Banded;
  pc->ops->destroy             = PCDestroy_Banded;
  pc->ops->setfromoptions      = PCSetFromOptions_Banded;
  pc->ops->view                = PCView_Banded;
  pc->ops->applyrichardson     = NULL;
  pc->ops->applysymmetricleft  = NULL;
  pc->ops->applysymmetricright = NULL;
  /* Setup concrete type-specific functions */
  /* Setup subobjects */
  PetscFunctionReturn(0);
}
```

# Creating a Preconditioner (**PC**)

## Define the constructor:

```
PetscErrorCode PCCreate_Banded(PC pc)
{
  PetscFunctionBegin;
  /* Create concrete type struct */
  /* Setup function table for class */
  PetscObjectComposeFunction((PetscObject) pc,
    "PCBandedSetMaxHalfBandwidth_C", PCBandedSetMaxHalfBandwidth_Banded);
  PetscObjectComposeFunction((PetscObject) pc,
    "PCBandedSetNormFraction_C", PCBandedSetNormFraction_Banded);
  /* Setup subobjects */
  PetscFunctionReturn(0);
}
```

# Creating a Preconditioner (**PC**)

## Define the constructor:

```
PetscErrorCode PCCreate_Banded(PC pc)
{
  PetscFunctionBegin;
  /* Create concrete type struct */
  /* Setup function table for class */
  /* Setup concrete type-specific functions */
  {
    const char *prefix;

    PCCreate(PetscObjectComm((PetscObject) pc), &b->pc);
    PetscObjectGetOptionsPrefix((PetscObject) pc, &prefix);
    PetscObjectSetOptionsPrefix((PetscObject) b->pc, prefix);
    PetscObjectAppendOptionsPrefix((PetscObject) b->pc, "banded_");
  }
  PetscFunctionReturn(0);
}
```

# Creating a Preconditioner (**PC**)

## Define concrete type-specific functions:

```
PetscErrorCode PCBandedSetMaxHalfBandwith(PC pc, PetscInt kmax)
{
  PetscErrorCode ierr;

  PetscFunctionBegin;
  PetscValidHeaderSpecific(pc, PC_CLASSID, 1);
  PetscTryMethod(pc, "PCBandedSetMaxHalfBandwith_C",
    (PC, PetscInt), (pc, kmax));
  PetscFunctionReturn(0);
}
```

# Outline

2 Extending PETSc
- Creating a new Class Implementation
- Distributing your new Implementation

# Distributing a Shared Library

## Package Initialization

- Register classes,
- concrete type constructors,
- logging events,
- and finalizer with `PetscRegisterFinalize()`

# Distributing a Shared Library

## Package Finalization

- Destroy list of constructors,
- and class memory allocations

# Distributing a Shared Library

## Package Loading

- Library located using

  `-dll_append/prepend <libname>`

- PETSc calls

  `PetscDLLibraryRegister_<libname>()`

- No recompiling or relinking of PETSc libraries or user code