MAE 5032 High Performance Computing: Methods and Practices

Lecture 13: PETSc

Ju Liu
Department of Mechanics and Aerospace Engineering
liuj36@sustech.edu.cn

What is PETSc

A freely available and supported research code for the parallel solution of linear algebraic, nonlinear algebraic, and differential equations.

- Free
  - Download from http://www.mcs.anl.gov/petsc
  - Free to everyone, including comerical uses
- Supported
  - Many tutorial examples
  - Hyperlinked manual, examples, and manual pages for all routines
  - Support email petsc-main@mcs.anl.gov
- Available for C, C++, Fortran, Matla, Python, etc.

What is PETSc

- Portable to any parallel system supporting MPI
- History
  - Begun in 1991
  - Over 60000 downloads since 1995
- Funding and support
  - Department of Energy
  - National Science Foundation
- One of the BIG-3 HPC libraries:
  PETSc, Trilinos, HYPRE

What is PortableETSc

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media

PFLOTRAN

A Massively Parallel Reactive Flow and Transport Model for describing Subsurface Processes

- PETSc has run on over 290000 cores efficiently
  - UNIC on the IBM BG/P Juene at Julich
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL

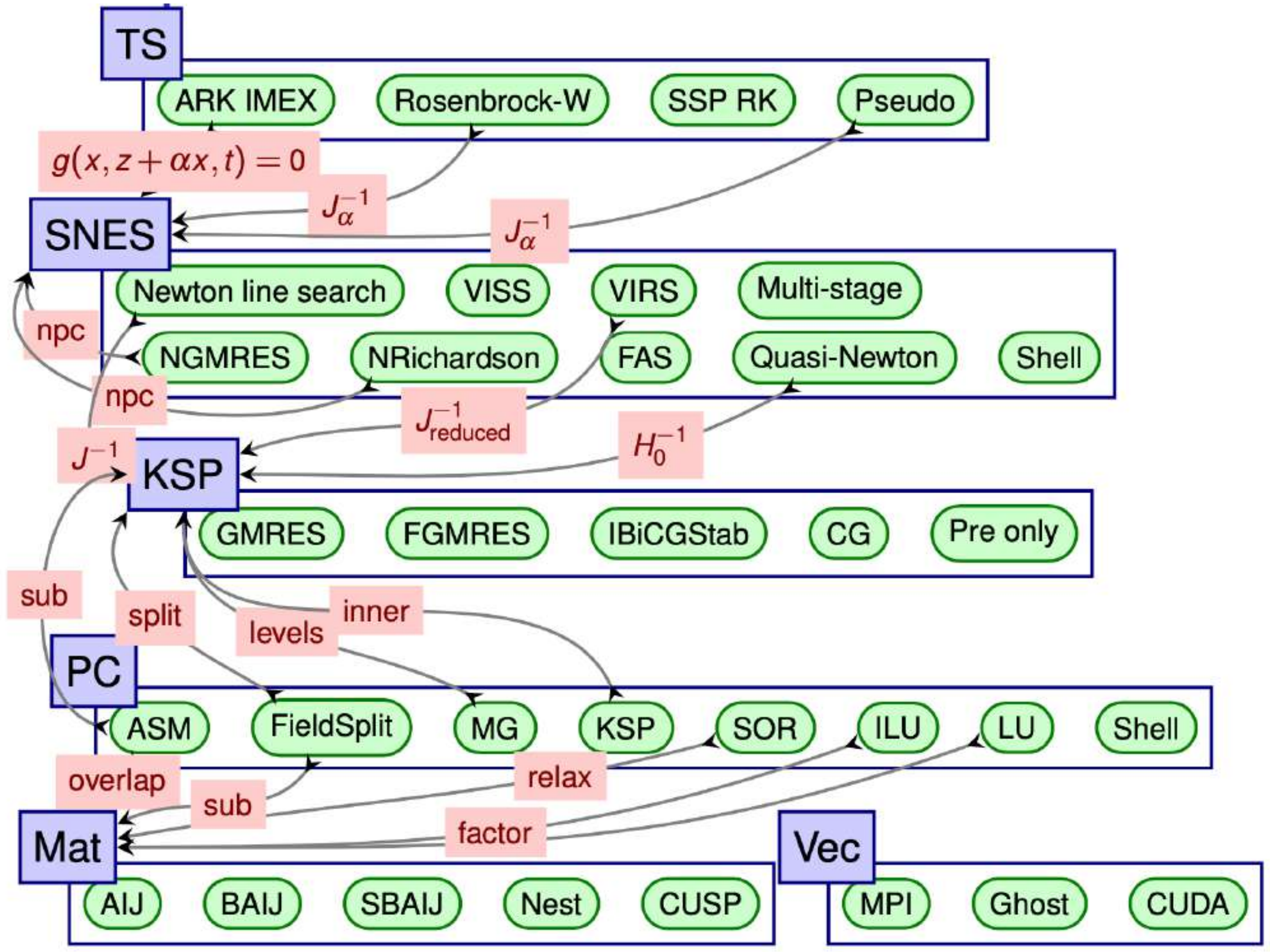
- PETSc applications have run at 23% of peak (600 Teraflops)
  - HPGMG code

PETSc runs not only on clusters, but also on a laptop, iPhone, GPU node

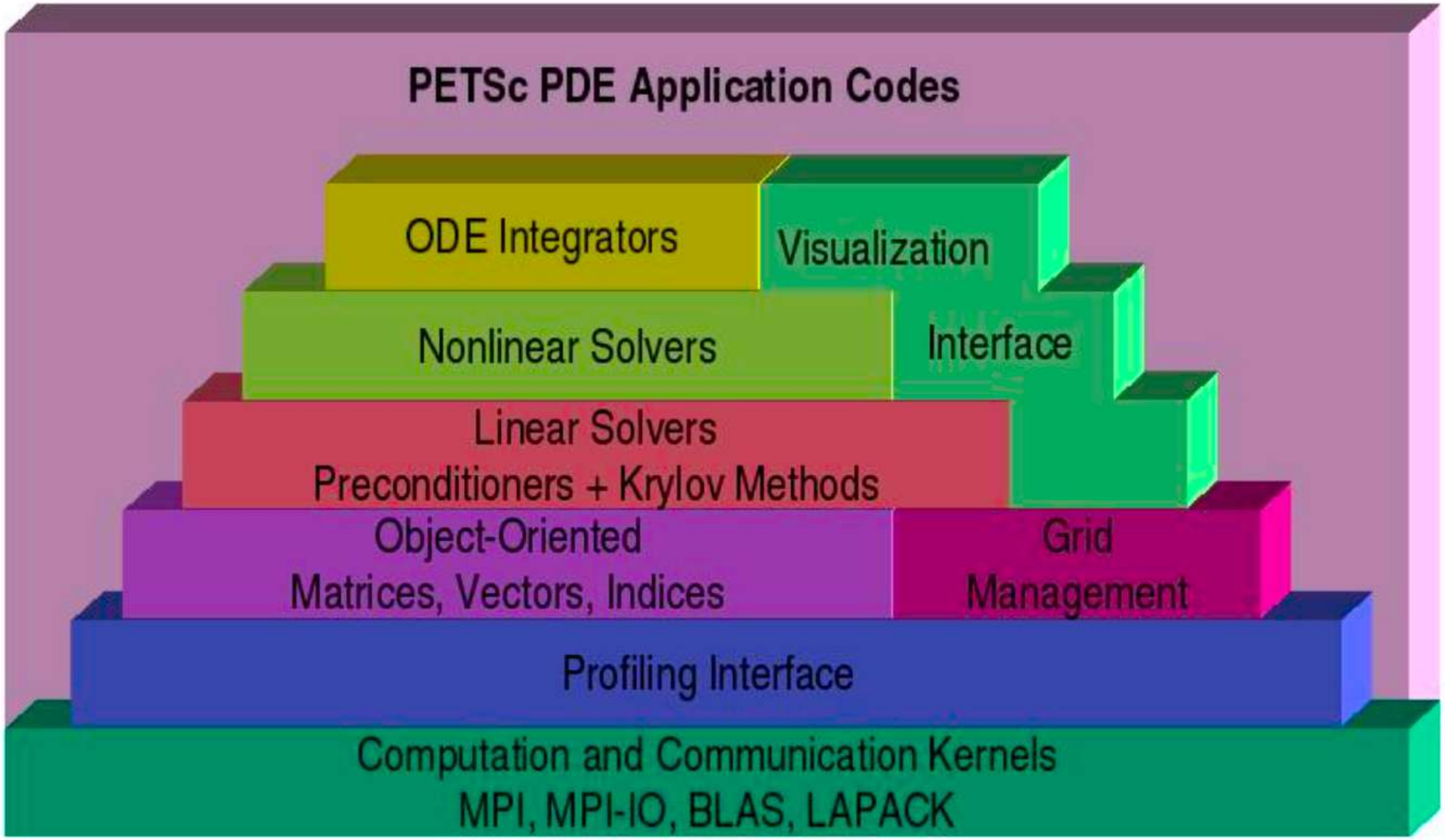
PETSc runs with Linux, Mac, Windows, and works with any compiler, supports real/complex, single/double/quad precision, 32/64-bit integer.

What is PExtensibleToolkitSc

- PETSc is designed in the physiolophy that everything has a plugin architecture
  - Vectors, Matrices, Indices
  - Preconditioners, Krylov methods
  - Nonlinear solvers, Time integrators
- Vendor supplies matrix format and associated preconditioners. Application users only loads plugin at runtime, no source code in sight.
- PETSc provides algorithms, debugging aids, and profiling tools



What is PExtensibleToolkitSc


PETSc PDE Application Codes
ODE Integrators
Visualization
Interface
Nonlinear Solvers
Linear Solvers
Preconditioners + Krylov Methods
Object-Oriented
Matrices, Vectors, Indices
Grid
Management
Profiling Interface
Computation and Communication Kernels
MPI, MPI-IO, BLAS, LAPACK

What is PETScientific computing

- Earth science: Underworld (Monash), Magma Dynamics (Columbia & Oxford)
- Subsurface flow and porous media: STOMP (DOE), PFLOTRAN (DOE)
- CFD: Firedrake, OpenFOAM, freeCFD, OpenFVM, Fluidity, PyClaw
- Micro-magnetics: MagPar
- Fusion: XGC, BOUT++, NIMROD
- Biomechanics: Chaste (Oxford)
- FEM: FEniCS, DEAL.ii (TAMU), PetIGA, MOOSE (INL)
- FSI: preCICE, PERIGEE (SUSTech)

Download an install PETSc

The latest tarball is on the PETSc website

https://petsc.org/release/download/

There is a Git development repository open to public

https://bitbucket.org/petsc/petsc

All releases are tags

Basic install steps

```
./configure –download-mpich –download-fblaslapack
make
make install
```
(if you specified prefix)

Download an install PETSc

- Common configure options
  - --prefix (for out-of-source install)
  - --with-scalar-type=<real or complex>
  - --with-precision=<single,double,__float128>
  - --with-64-bit-indices
  - --with-debugging=1/0 (default 1)
  - --download-{metis,mumps,scalapack,……}
    - BLAS, LAPACK, MPICH, Open MPI
    - ScaLAPACK, Elemental, ParMetis, Metis, Chaco, Zoltan
    - MUMPS, SuperLU, SuperLU_Dist
    - HYPRE, ML
    - HDF5, NetCDF

  Can also use –with-xxx-dir=/path-to-your-install-of-3rd-party-libs

  Watch my video on BB for a slightly advanced install of PETSc on TaiYi.

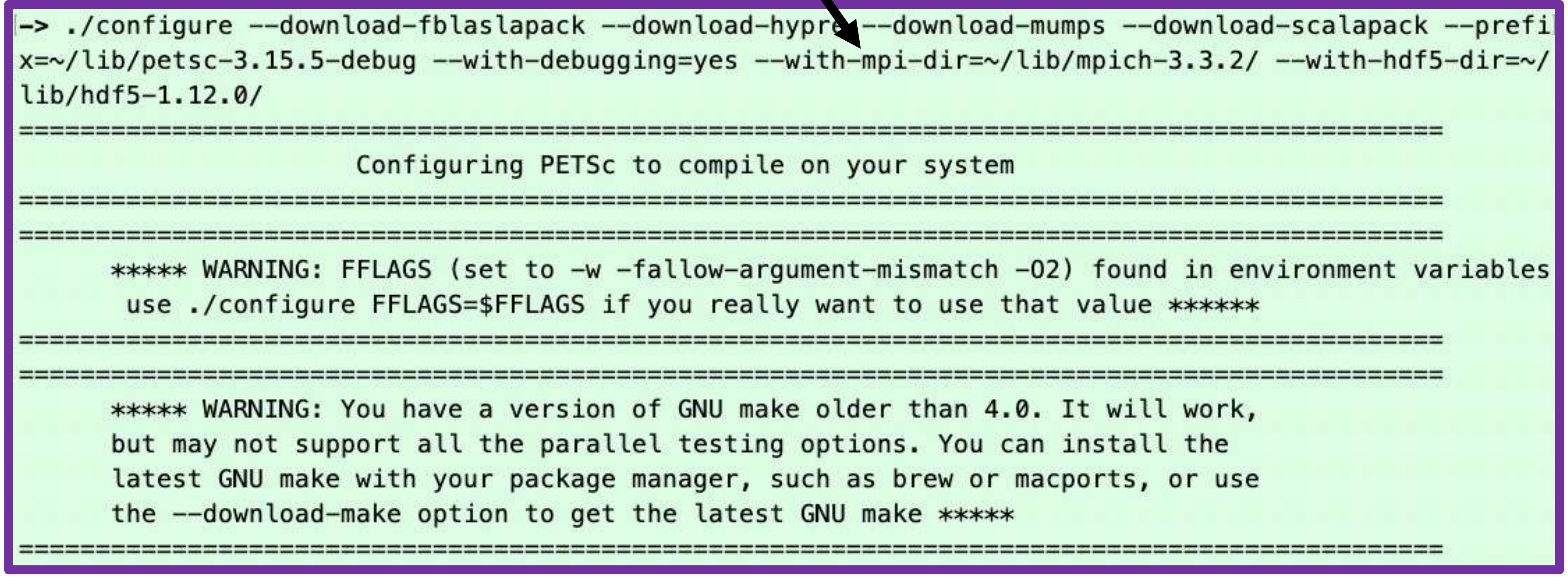
Download an install PETSc

- My configure command:

```
./configure
--with-mpi-
dir=/share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/mpi/intel64
/ --with-blaslapack-
dir=/share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/mkl --with-
debugging=no --prefix=/work/mae-liuj/lib/petsc-3.16.6-opt --download-
hypre=/work/mae-liuj/petsc-3.16.6-extlibs/hypre-2.23.0.tar.gz --download-
mumps=/work/mae-liuj/petsc-3.16.6-extlibs/petsc-pkg-mumps-
6d1470374d32.tar.gz --download-metis=/work/mae-liuj/petsc-3.16.6-
extlibs/petsc-pkg-metis-c8d2dc1e751e.tar.gz COPTFLAGS="-O3 -xHOST"
CXXOPTFLAGS="-O3 -xHOST" FOPTFLAGS="-O3 -xHOST" --with-scalapack-
include=/share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/mkl/inc
lude --with-scalapack-lib="-
L/share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/mkl/lib/intel6
4/ -lmkl_blacs_intelmpi_lp64 -lmkl_scalapack_lp64"
```

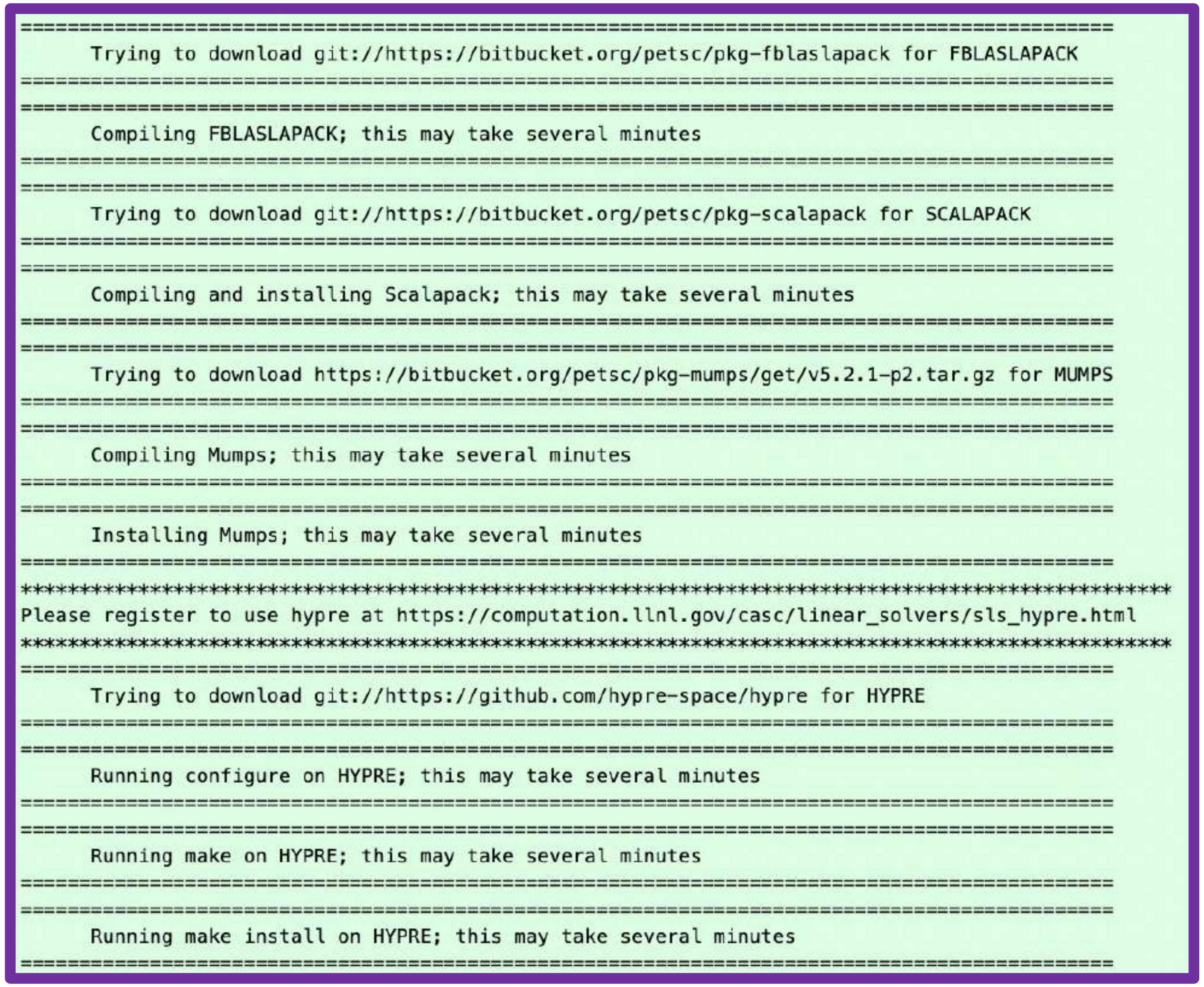
Download an install PETSc

configure with external libraries you need.
If you have already installed some of them, specify the install location.



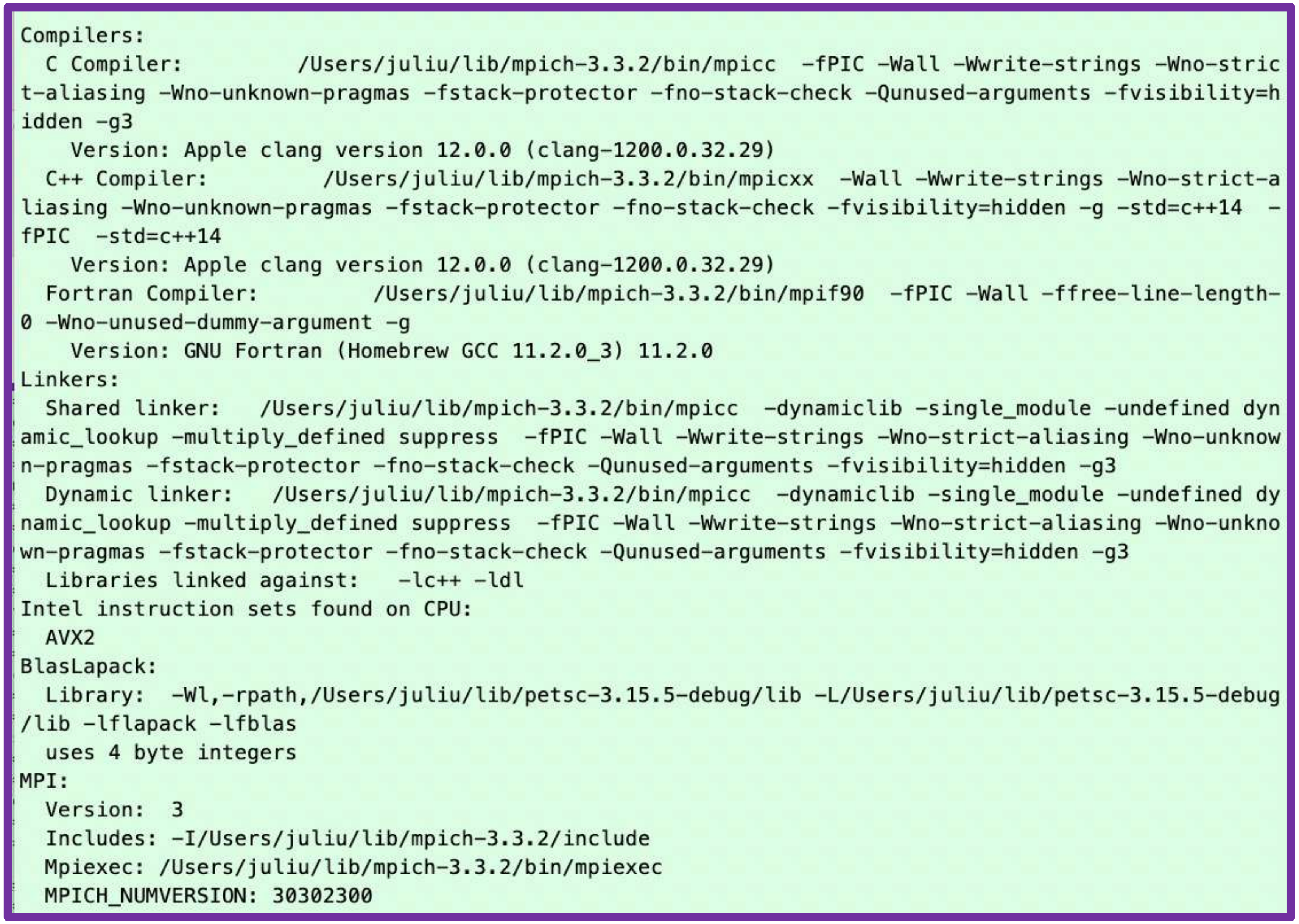
```
-> ./configure --download-fblaslapack --download-hypre --download-mumps --download-scalapack --prefi
x=~/lib/petsc-3.15.5-debug --with-debugging=yes --with-mpi-dir=~/lib/mpich-3.3.2/ --with-hdf5-dir=~/
lib/hdf5-1.12.0/
=============================================================================================
                      Configuring PETSc to compile on your system
=============================================================================================
=============================================================================================
      ***** WARNING: FFLAGS (set to -w -fallow-argument-mismatch -O2) found in environment variables
       use ./configure FFLAGS=$FFLAGS if you really want to use that value ******
=============================================================================================
=============================================================================================
      ***** WARNING: You have a version of GNU make older than 4.0. It will work,
      but may not support all the parallel testing options. You can install the
      latest GNU make with your package manager, such as brew or macports, or use
      the --download-make option to get the latest GNU make *****
=============================================================================================
```

Download an install PETSc

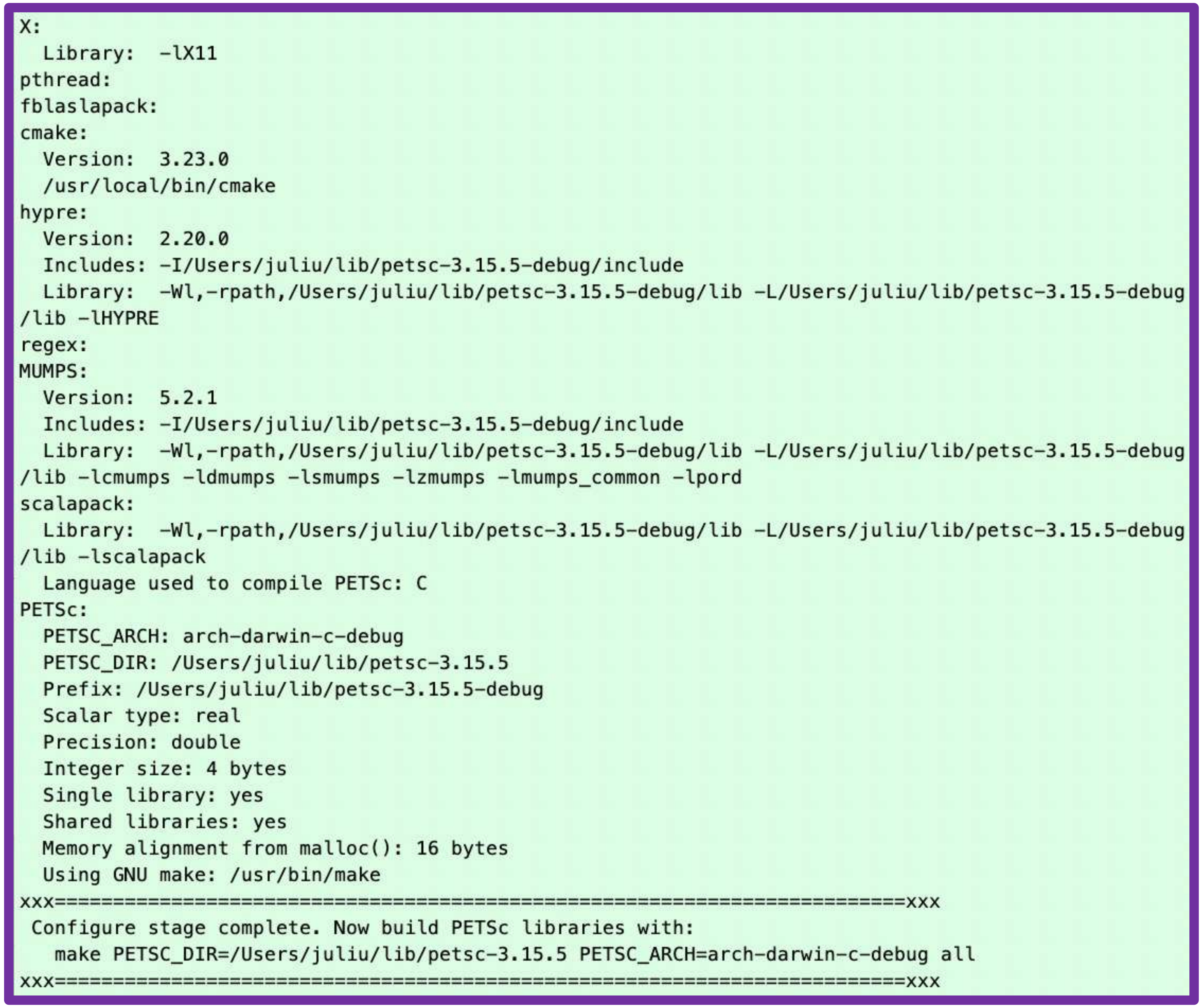


```
=============================================================================================
      Trying to download git://https://bitbucket.org/petsc/pkg-fblaslapack for FBLASLAPACK
=============================================================================================
=============================================================================================
      Compiling FBLASLAPACK; this may take several minutes
=============================================================================================
=============================================================================================
      Trying to download git://https://bitbucket.org/petsc/pkg-scalapack for SCALAPACK
=============================================================================================
=============================================================================================
      Compiling and installing Scalapack; this may take several minutes
=============================================================================================
=============================================================================================
      Trying to download https://bitbucket.org/petsc/pkg-mumps/get/v5.2.1-p2.tar.gz for MUMPS
=============================================================================================
=============================================================================================
      Compiling Mumps; this may take several minutes
=============================================================================================
=============================================================================================
      Installing Mumps; this may take several minutes
=============================================================================================
**************************************************************************************************
Please register to use hypre at https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html
**************************************************************************************************
=============================================================================================
      Trying to download git://https://github.com/hypre-space/hypre for HYPRE
=============================================================================================
=============================================================================================
      Running configure on HYPRE; this may take several minutes
=============================================================================================
=============================================================================================
      Running make on HYPRE; this may take several minutes
=============================================================================================
=============================================================================================
      Running make install on HYPRE; this may take several minutes
=============================================================================================
```

MUMPS is a Multifrontal Massively Parallel sparse direct Solver



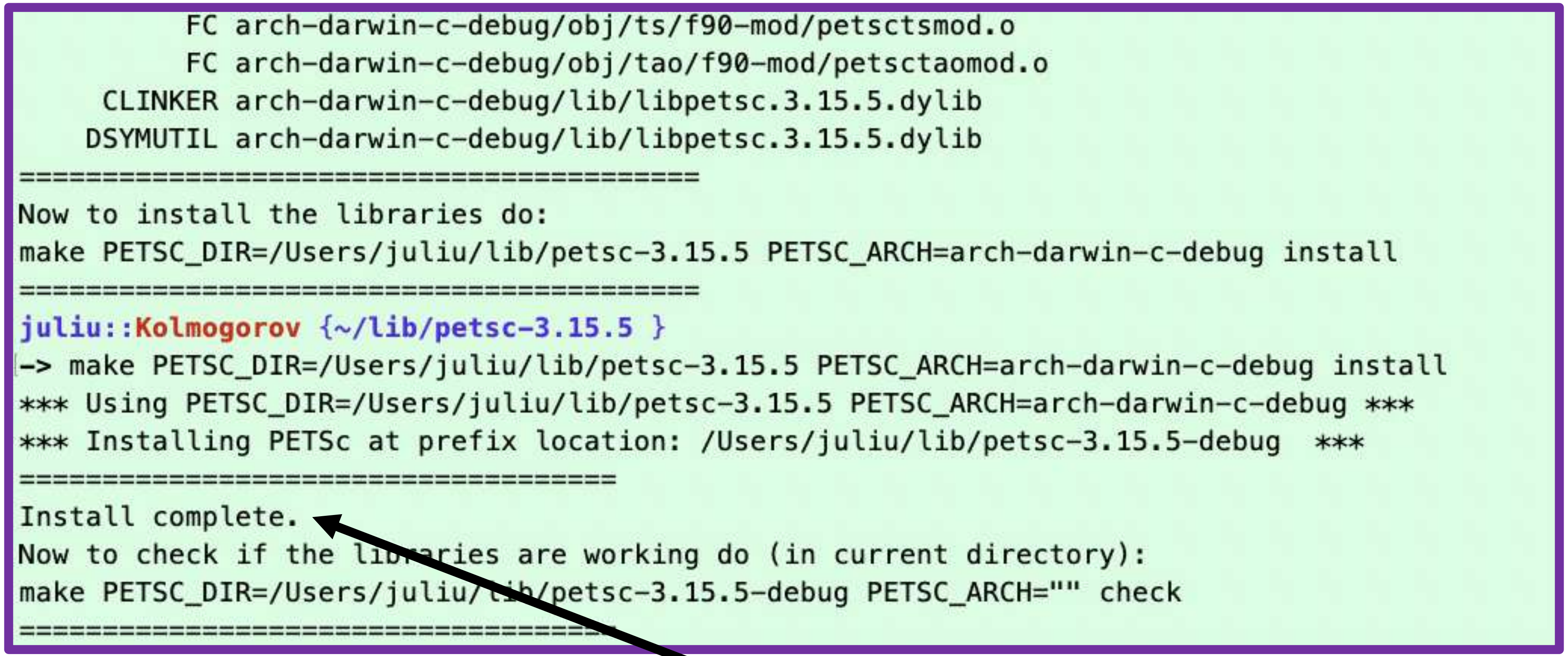
```
Compilers:
  C Compiler:         /Users/juliu/lib/mpich-3.3.2/bin/mpicc  -fPIC -Wall -Wwrite-strings -Wno-stric
t-aliasing -Wno-unknown-pragmas -fstack-protector -fno-stack-check -Qunused-arguments -fvisibility=h
idden -g3
    Version: Apple clang version 12.0.0 (clang-1200.0.32.29)
  C++ Compiler:         /Users/juliu/lib/mpich-3.3.2/bin/mpicxx  -Wall -Wwrite-strings -Wno-strict-a
liasing -Wno-unknown-pragmas -fstack-protector -fno-stack-check -fvisibility=hidden -g -std=c++14  -
fPIC  -std=c++14
    Version: Apple clang version 12.0.0 (clang-1200.0.32.29)
  Fortran Compiler:         /Users/juliu/lib/mpich-3.3.2/bin/mpif90  -fPIC -Wall -ffree-line-length-
0 -Wno-unused-dummy-argument -g
    Version: GNU Fortran (Homebrew GCC 11.2.0_3) 11.2.0
Linkers:
  Shared linker:   /Users/juliu/lib/mpich-3.3.2/bin/mpicc  -dynamiclib -single_module -undefined dyn
amic_lookup -multiply_defined suppress  -fPIC -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknow
n-pragmas -fstack-protector -fno-stack-check -Qunused-arguments -fvisibility=hidden -g3
  Dynamic linker:   /Users/juliu/lib/mpich-3.3.2/bin/mpicc  -dynamiclib -single_module -undefined dy
namic_lookup -multiply_defined suppress  -fPIC -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unkno
wn-pragmas -fstack-protector -fno-stack-check -Qunused-arguments -fvisibility=hidden -g3
  Libraries linked against:   -lc++ -ldl
Intel instruction sets found on CPU:
  AVX2
BlasLapack:
  Library:  -Wl,-rpath,/Users/juliu/lib/petsc-3.15.5-debug/lib -L/Users/juliu/lib/petsc-3.15.5-debug
/lib -lflapack -lfblas
  uses 4 byte integers
MPI:
  Version:  3
  Includes: -I/Users/juliu/lib/mpich-3.3.2/include
  Mpiexec: /Users/juliu/lib/mpich-3.3.2/bin/mpiexec
  MPICH_NUMVERSION: 30302300
```



```
X:
  Library:  -lX11
pthread:
fblaslapack:
cmake:
  Version:  3.23.0
  /usr/local/bin/cmake
hypre:
  Version:  2.20.0
  Includes: -I/Users/juliu/lib/petsc-3.15.5-debug/include
  Library:  -Wl,-rpath,/Users/juliu/lib/petsc-3.15.5-debug/lib -L/Users/juliu/lib/petsc-3.15.5-debug
/lib -lHYPRE
regex:
MUMPS:
  Version:  5.2.1
  Includes: -I/Users/juliu/lib/petsc-3.15.5-debug/include
  Library:  -Wl,-rpath,/Users/juliu/lib/petsc-3.15.5-debug/lib -L/Users/juliu/lib/petsc-3.15.5-debug
/lib -lcmumps -ldmumps -lsmumps -lzmumps -lmumps_common -lpord
scalapack:
  Library:  -Wl,-rpath,/Users/juliu/lib/petsc-3.15.5-debug/lib -L/Users/juliu/lib/petsc-3.15.5-debug
/lib -lscalapack
  Language used to compile PETSc: C
PETSc:
  PETSC_ARCH: arch-darwin-c-debug
  PETSC_DIR: /Users/juliu/lib/petsc-3.15.5
  Prefix: /Users/juliu/lib/petsc-3.15.5-debug
  Scalar type: real
  Precision: double
  Integer size: 4 bytes
  Single library: yes
  Shared libraries: yes
  Memory alignment from malloc(): 16 bytes
  Using GNU make: /usr/bin/make
xxx=========================================================================xxx
 Configure stage complete. Now build PETSc libraries with:
   make PETSC_DIR=/Users/juliu/lib/petsc-3.15.5 PETSC_ARCH=arch-darwin-c-debug all
xxx=========================================================================xxx
```

- Complete means configuration is successful.
- Follow the instructions to run make and make install, and perhaps other make commands.

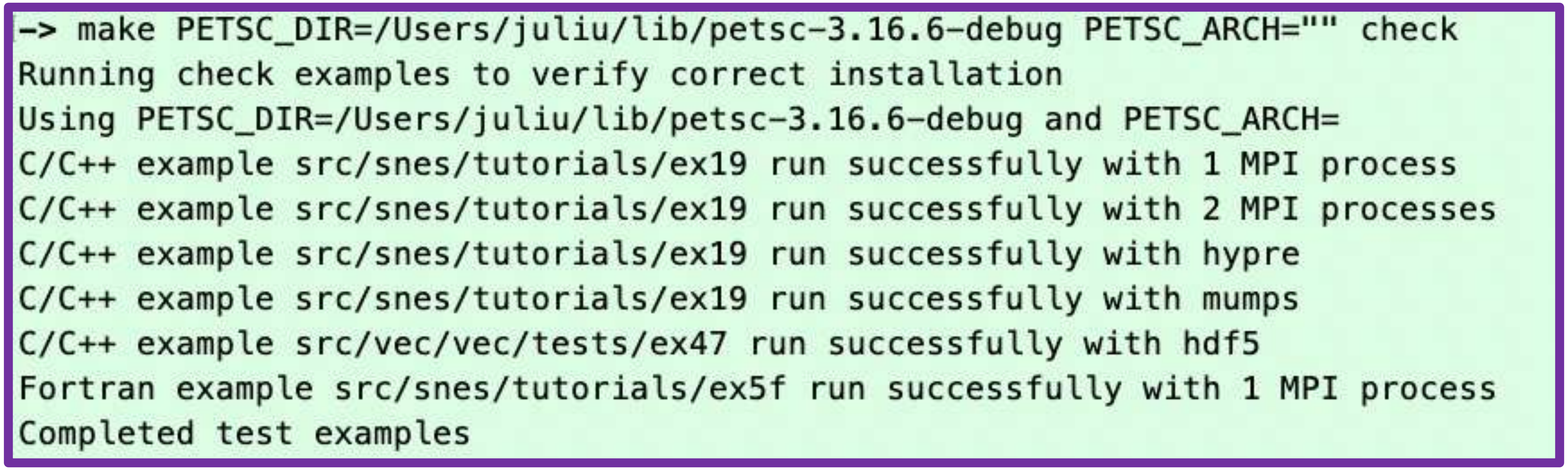
Download an install PETSc



```
          FC arch-darwin-c-debug/obj/ts/f90-mod/petsctsmod.o
          FC arch-darwin-c-debug/obj/tao/f90-mod/petsctaomod.o
     CLINKER arch-darwin-c-debug/lib/libpetsc.3.15.5.dylib
    DSYMUTIL arch-darwin-c-debug/lib/libpetsc.3.15.5.dylib
=========================================
Now to install the libraries do:
make PETSC_DIR=/Users/juliu/lib/petsc-3.15.5 PETSC_ARCH=arch-darwin-c-debug install
=========================================
juliu::Kolmogorov {~/lib/petsc-3.15.5 }
-> make PETSC_DIR=/Users/juliu/lib/petsc-3.15.5 PETSC_ARCH=arch-darwin-c-debug install
*** Using PETSC_DIR=/Users/juliu/lib/petsc-3.15.5 PETSC_ARCH=arch-darwin-c-debug ***
*** Installing PETSc at prefix location: /Users/juliu/lib/petsc-3.15.5-debug  ***
====================================
Install complete.
Now to check if the libraries are working do (in current directory):
make PETSC_DIR=/Users/juliu/lib/petsc-3.15.5-debug PETSC_ARCH="" check
====================================
```

Follow the instructions, untill you see “Install complete”.

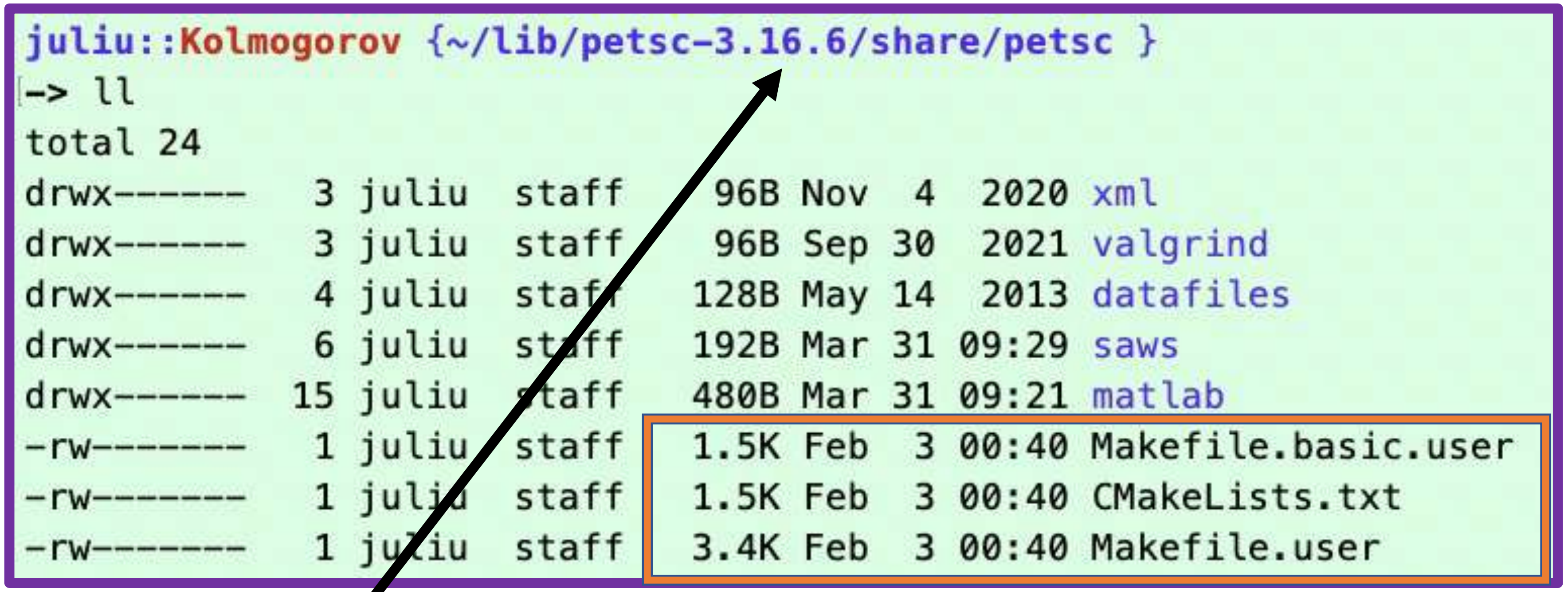
Download an install PETSc



```
-> make PETSC_DIR=/Users/juliu/lib/petsc-3.16.6-debug PETSC_ARCH="" check
Running check examples to verify correct installation
Using PETSC_DIR=/Users/juliu/lib/petsc-3.16.6-debug and PETSC_ARCH=
C/C++ example src/snes/tutorials/ex19 run successfully with 1 MPI process
C/C++ example src/snes/tutorials/ex19 run successfully with 2 MPI processes
C/C++ example src/snes/tutorials/ex19 run successfully with hypre
C/C++ example src/snes/tutorials/ex19 run successfully with mumps
C/C++ example src/vec/vec/tests/ex47 run successfully with hdf5
Fortran example src/snes/tutorials/ex5f run successfully with 1 MPI process
Completed test examples
```

You may want to run make check to make sure things are all correctly installed.

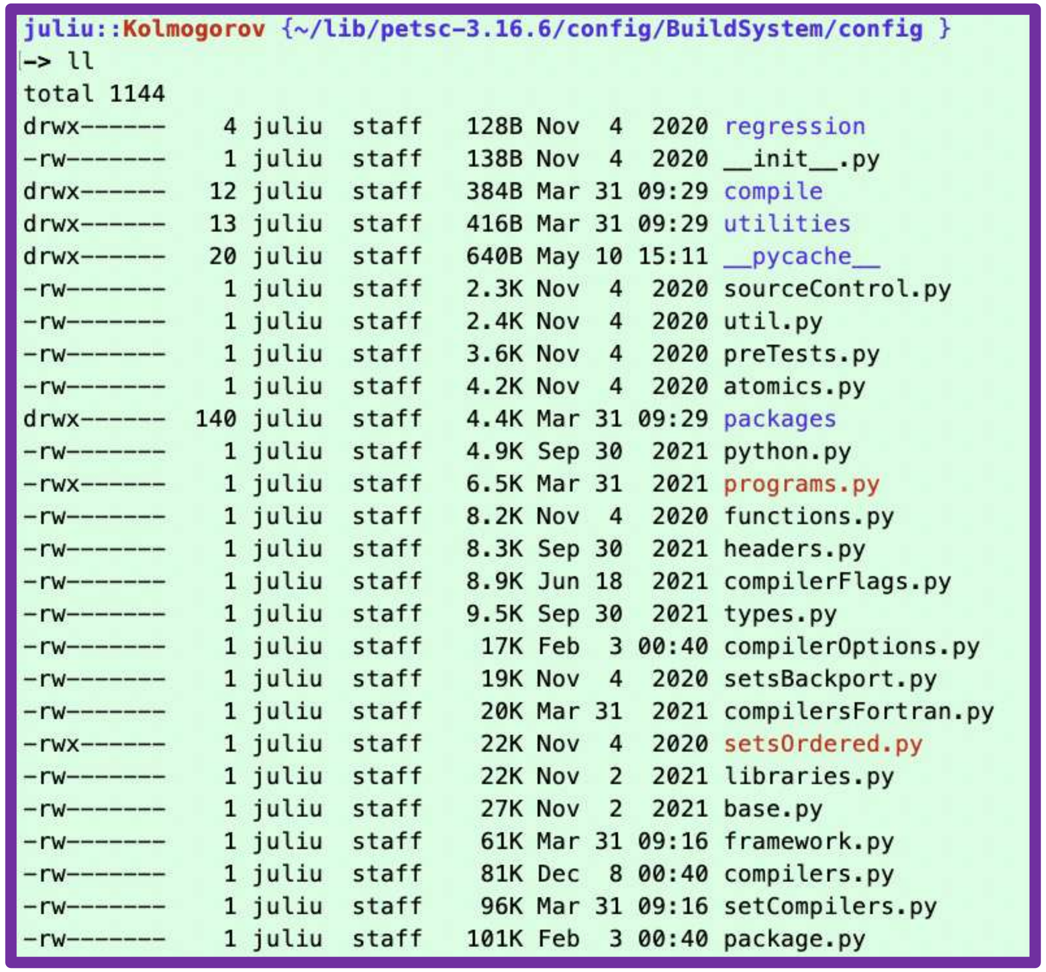
Download an install PETSc



```
juliu::Kolmogorov {~/lib/petsc-3.16.6/share/petsc }
-> ll
total 24
drwx------   3 juliu  staff    96B Nov  4  2020 xml
drwx------   3 juliu  staff    96B Sep 30  2021 valgrind
drwx------   4 juliu  staff   128B May 14  2013 datafiles
drwx------   6 juliu  staff   192B Mar 31 09:29 saws
drwx------  15 juliu  staff   480B Mar 31 09:21 matlab
-rw-------   1 juliu  staff   1.5K Feb  3 00:40 Makefile.basic.user
-rw-------   1 juliu  staff   1.5K Feb  3 00:40 CMakeLists.txt
-rw-------   1 juliu  staff   3.4K Feb  3 00:40 Makefile.user
```

Compiling with PETSc is not trivial because it is too good (portable & extensible)
Check the /share/petsc folder for sample Makefile and CMakeLists.

Download an install PETSc



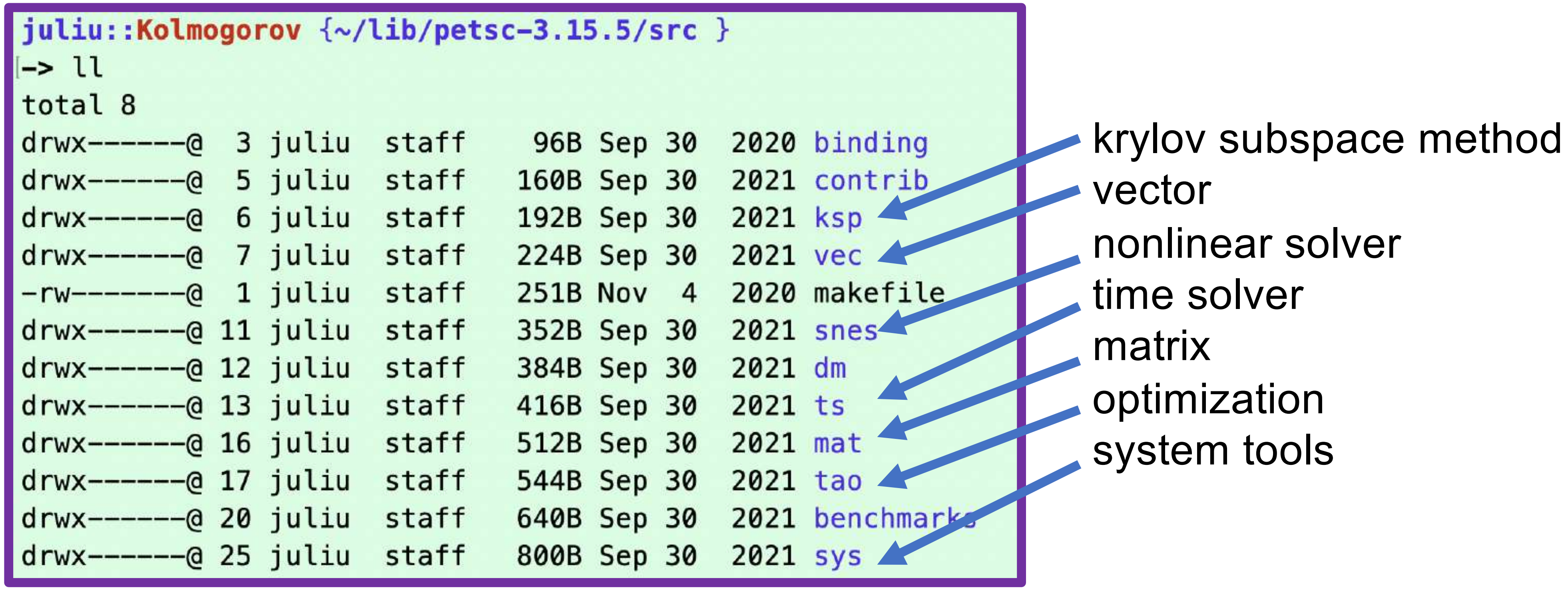
```
juliu::Kolmogorov {~/lib/petsc-3.16.6/config/BuildSystem/config }
|-> ll
total 1144
drwx------    4 juliu  staff   128B Nov  4  2020 regression
-rw-------    1 juliu  staff   138B Nov  4  2020 __init__.py
drwx------   12 juliu  staff   384B Mar 31 09:29 compile
drwx------   13 juliu  staff   416B Mar 31 09:29 utilities
drwx------   20 juliu  staff   640B May 10 15:11 __pycache__
-rw-------    1 juliu  staff   2.3K Nov  4  2020 sourceControl.py
-rw-------    1 juliu  staff   2.4K Nov  4  2020 util.py
-rw-------    1 juliu  staff   3.6K Nov  4  2020 preTests.py
-rw-------    1 juliu  staff   4.2K Nov  4  2020 atomics.py
drwx------  140 juliu  staff   4.4K Mar 31 09:29 packages
-rw-------    1 juliu  staff   4.9K Sep 30  2021 python.py
-rwx------    1 juliu  staff   6.5K Mar 31  2021 programs.py
-rw-------    1 juliu  staff   8.2K Nov  4  2020 functions.py
-rw-------    1 juliu  staff   8.3K Sep 30  2021 headers.py
-rw-------    1 juliu  staff   8.9K Jun 18  2021 compilerFlags.py
-rw-------    1 juliu  staff   9.5K Sep 30  2021 types.py
-rw-------    1 juliu  staff    17K Feb  3 00:40 compilerOptions.py
-rw-------    1 juliu  staff    19K Nov  4  2020 setsBackport.py
-rw-------    1 juliu  staff    20K Mar 31  2021 compilersFortran.py
-rwx------    1 juliu  staff    22K Nov  4  2020 setsOrdered.py
-rw-------    1 juliu  staff    22K Nov  2  2021 libraries.py
-rw-------    1 juliu  staff    27K Nov  2  2021 base.py
-rw-------    1 juliu  staff    61K Mar 31 09:16 framework.py
-rw-------    1 juliu  staff    81K Dec  8 00:40 compilers.py
-rw-------    1 juliu  staff    96K Mar 31 09:16 setCompilers.py
-rw-------    1 juliu  staff   101K Feb  3 00:40 package.py
```

In the config subfolder, there are Python scripts that determines the actual configuration of PETSc.

You may locate the external package dependencies for your own PETSc version in /config/BuildSystem/config/packages

Learn from the tutorials

- Use makefile to compile a single file can be convenient.
- See a makefile example in $PETSC_DIR/share/petsc/Makefile.user, read its comments, and make modifications.

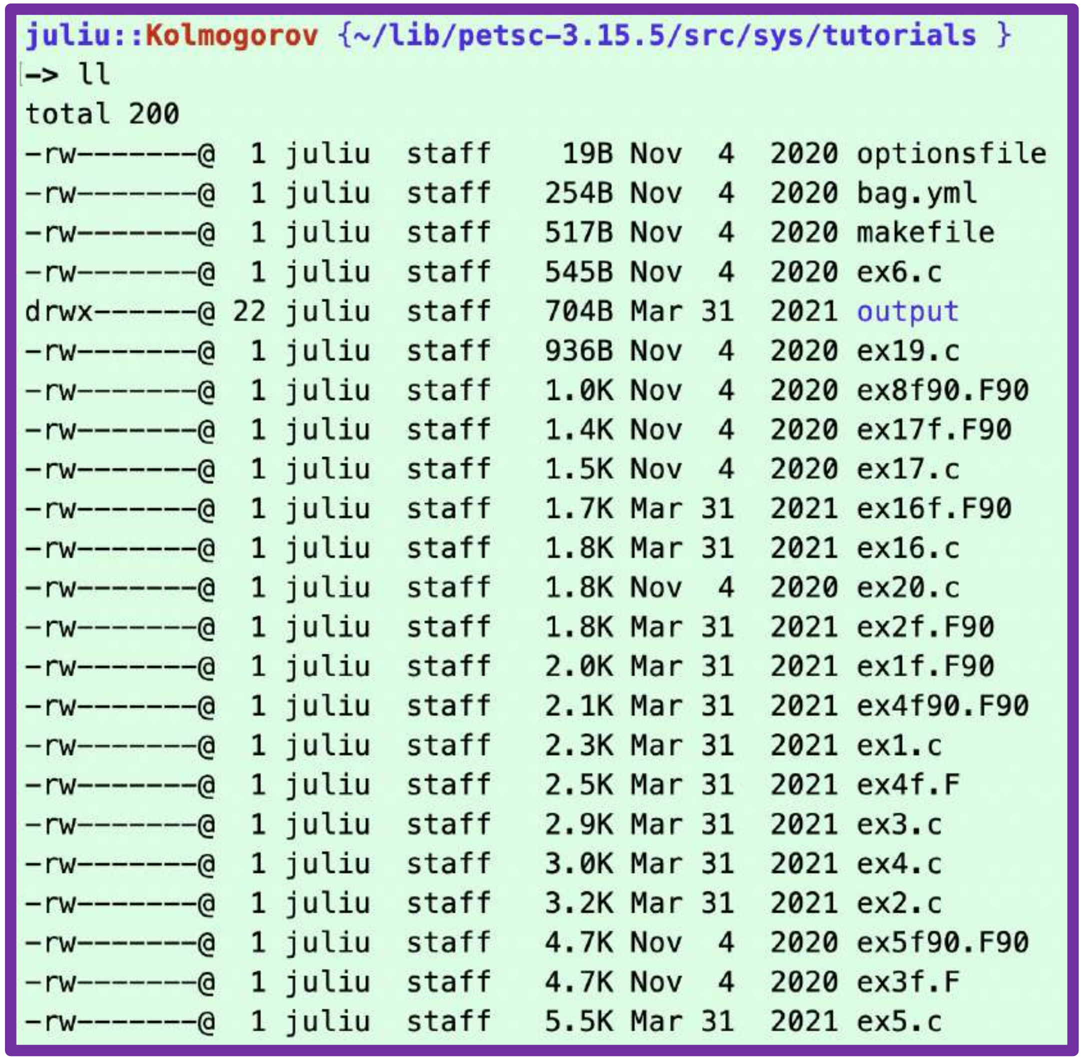


```
juliu::Kolmogorov {~/lib/petsc-3.15.5/src }
-> ll
total 8
drwx------@  3 juliu  staff   96B Sep 30  2020 binding
drwx------@  5 juliu  staff  160B Sep 30  2021 contrib
drwx------@  6 juliu  staff  192B Sep 30  2021 ksp
drwx------@  7 juliu  staff  224B Sep 30  2021 vec
-rw-------@  1 juliu  staff  251B Nov  4  2020 makefile
drwx------@ 11 juliu  staff  352B Sep 30  2021 snes
drwx------@ 12 juliu  staff  384B Sep 30  2021 dm
drwx------@ 13 juliu  staff  416B Sep 30  2021 ts
drwx------@ 16 juliu  staff  512B Sep 30  2021 mat
drwx------@ 17 juliu  staff  544B Sep 30  2021 tao
drwx------@ 20 juliu  staff  640B Sep 30  2021 benchmarks
drwx------@ 25 juliu  staff  800B Sep 30  2021 sys
```

Learn from the tutorials

There is a tutorial folder in each src subfolder, containing examples for different tools (sys, vec, mat, ksp, snes, etc.)

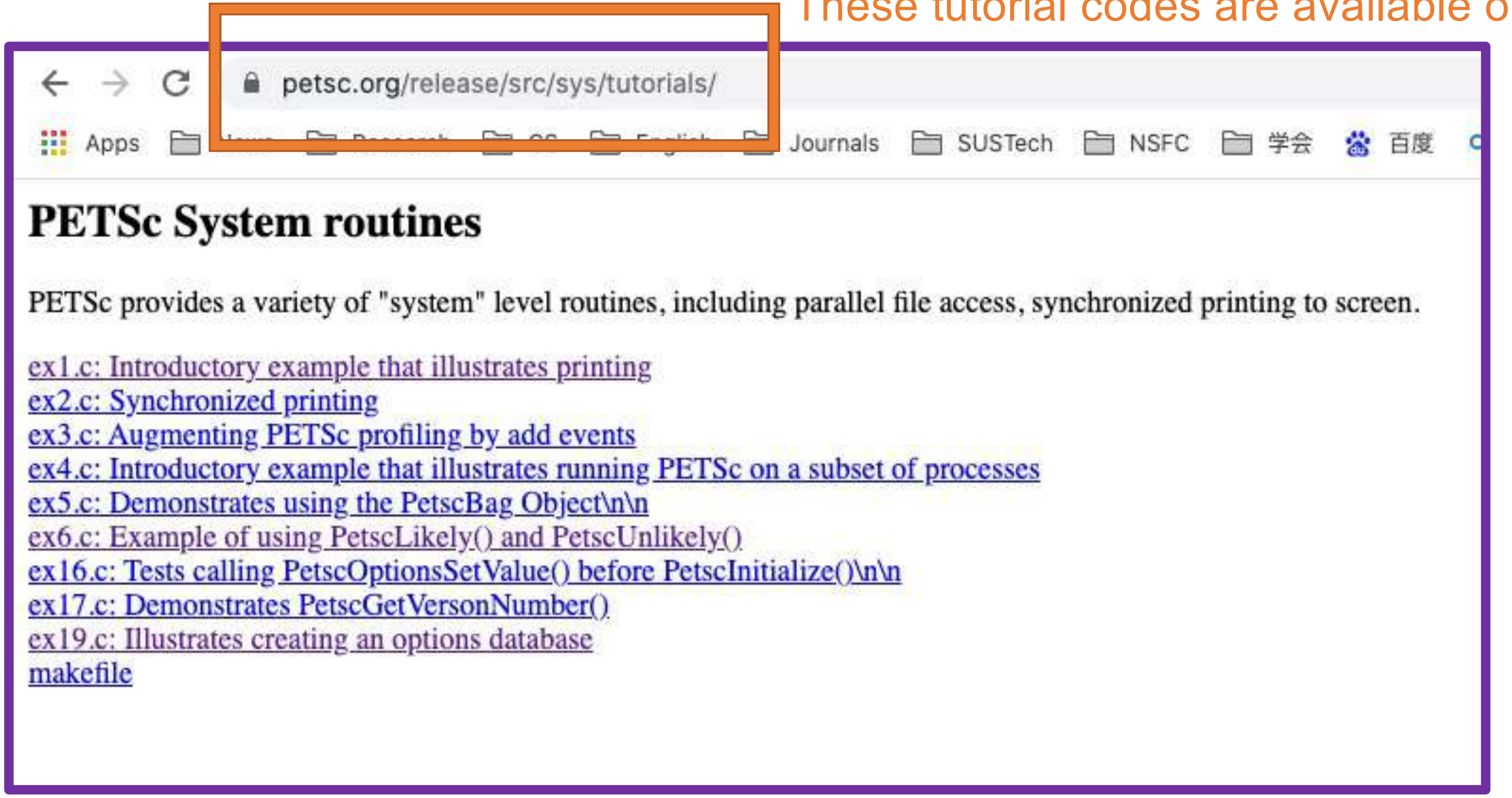
These tutorial codes are available on PETSc website as well.



```
juliu::Kolmogorov {~/lib/petsc-3.15.5/src/sys/tutorials }
-> ll
total 200
-rw-------@  1 juliu  staff   19B Nov  4  2020 optionsfile
-rw-------@  1 juliu  staff  254B Nov  4  2020 bag.yml
-rw-------@  1 juliu  staff  517B Nov  4  2020 makefile
-rw-------@  1 juliu  staff  545B Nov  4  2020 ex6.c
drwx------@ 22 juliu  staff  704B Mar 31  2021 output
-rw-------@  1 juliu  staff  936B Nov  4  2020 ex19.c
-rw-------@  1 juliu  staff  1.0K Nov  4  2020 ex8f90.F90
-rw-------@  1 juliu  staff  1.4K Nov  4  2020 ex17f.F90
-rw-------@  1 juliu  staff  1.5K Nov  4  2020 ex17.c
-rw-------@  1 juliu  staff  1.7K Mar 31  2021 ex16f.F90
-rw-------@  1 juliu  staff  1.8K Mar 31  2021 ex16.c
-rw-------@  1 juliu  staff  1.8K Nov  4  2020 ex20.c
-rw-------@  1 juliu  staff  1.8K Mar 31  2021 ex2f.F90
-rw-------@  1 juliu  staff  2.0K Mar 31  2021 ex1f.F90
-rw-------@  1 juliu  staff  2.1K Mar 31  2021 ex4f90.F90
-rw-------@  1 juliu  staff  2.3K Mar 31  2021 ex1.c
-rw-------@  1 juliu  staff  2.5K Mar 31  2021 ex4f.F
-rw-------@  1 juliu  staff  2.9K Mar 31  2021 ex3.c
-rw-------@  1 juliu  staff  3.0K Mar 31  2021 ex4.c
-rw-------@  1 juliu  staff  3.2K Mar 31  2021 ex2.c
-rw-------@  1 juliu  staff  4.7K Nov  4  2020 ex5f90.F90
-rw-------@  1 juliu  staff  4.7K Nov  4  2020 ex3f.F
-rw-------@  1 juliu  staff  5.5K Mar 31  2021 ex5.c
```

Learn from the tutorials

These tutorial codes are available online.

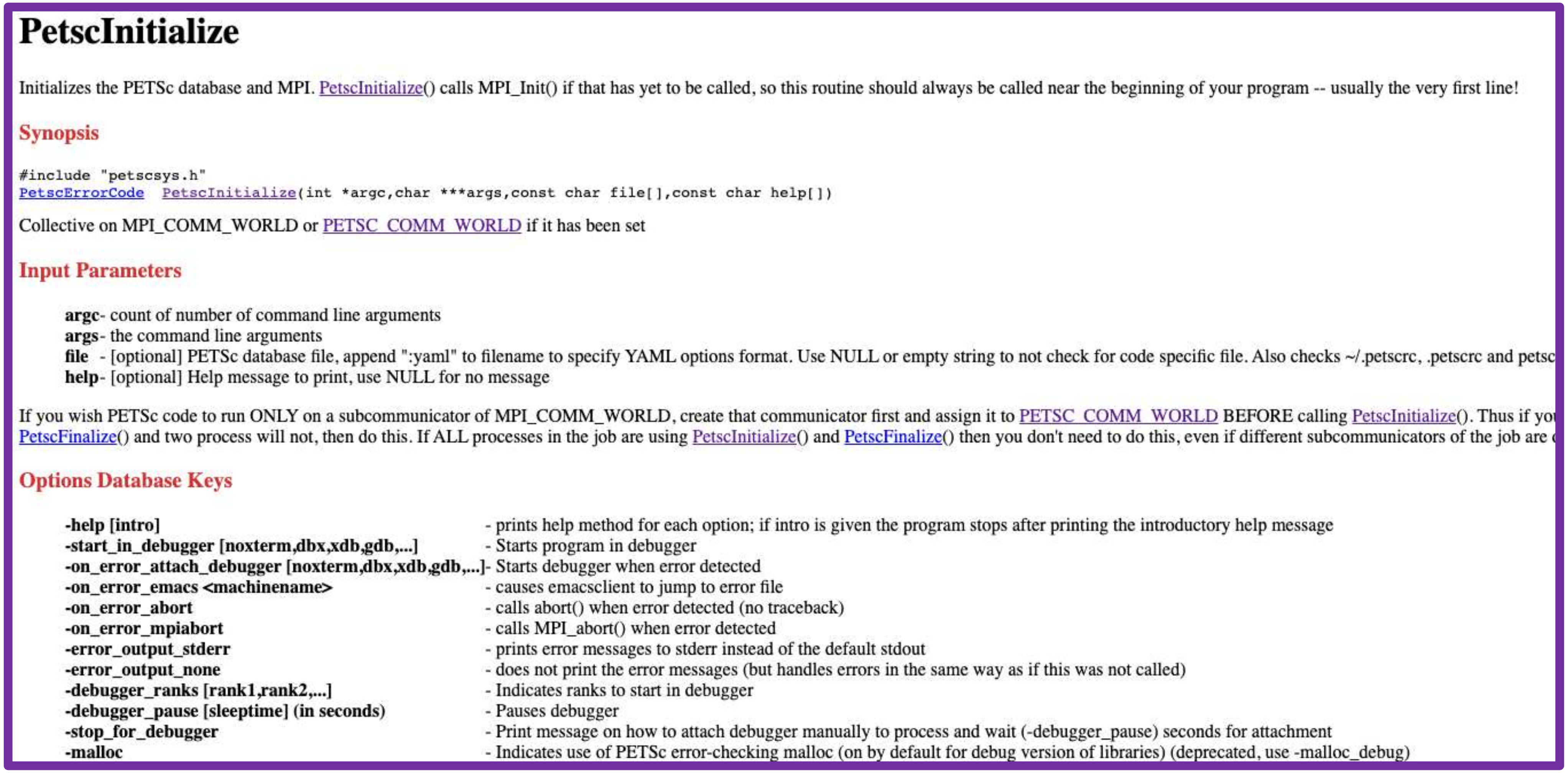


PETSc System routines

PETSc provides a variety of "system" level routines, including parallel file access, synchronized printing to screen.

ex1.c: Introductory example that illustrates printing
ex2.c: Synchronized printing
ex3.c: Augmenting PETSc profiling by add events
ex4.c: Introductory example that illustrates running PETSc on a subset of processes
ex5.c: Demonstrates using the PetscBag Object\n\n
ex6.c: Example of using PetscLikely() and PetscUnlikely()
ex16.c: Tests calling PetscOptionsSetValue() before PetscInitialize()\n\n
ex17.c: Demonstrates PetscGetVersonNumber()
ex19.c: Illustrates creating an options database
makefile

Learn from the tutorials



PetscInitialize

Initializes the PETSc database and MPI. PetscInitialize() calls MPI_Init() if that has yet to be called, so this routine should always be called near the beginning of your program -- usually the very first line!

Synopsis

```
#include "petscsys.h"
PetscErrorCode  PetscInitialize(int *argc,char ***args,const char file[],const char help[])
```

Collective on MPI_COMM_WORLD or PETSC_COMM_WORLD if it has been set

Input Parameters

argc- count of number of command line arguments
args- the command line arguments
file - [optional] PETSc database file, append ":yaml" to filename to specify YAML options format. Use NULL or empty string to not check for code specific file. Also checks ~/.petscrc, .petscrc and petsc
help- [optional] Help message to print, use NULL for no message

If you wish PETSc code to run ONLY on a subcommunicator of MPI_COMM_WORLD, create that communicator first and assign it to PETSC_COMM_WORLD BEFORE calling PetscInitialize(). Thus if yo
PetscFinalize() and two process will not, then do this. If ALL processes in the job are using PetscInitialize() and PetscFinalize() then you don't need to do this, even if different subcommunicators of the job are

Options Database Keys

-help [intro]	- prints help method for each option; if intro is given the program stops after printing the introductory help message
-start_in_debugger [noxterm,dbx,xdb,gdb,...]	- Starts program in debugger
-on_error_attach_debugger [noxterm,dbx,xdb,gdb,...]	- Starts debugger when error detected
-on_error_emacs <machinename>	- causes emacsclient to jump to error file
-on_error_abort	- calls abort() when error detected (no traceback)
-on_error_mpiabort	- calls MPI_abort() when error detected
-error_output_stderr	- prints error messages to stderr instead of the default stdout
-error_output_none	- does not print the error messages (but handles errors in the same way as if this was not called)
-debugger_ranks [rank1,rank2,...]	- Indicates ranks to start in debugger
-debugger_pause [sleeptime] (in seconds)	- Pauses debugger
-stop_for_debugger	- Print message on how to attach debugger manually to process and wait (-debugger_pause) seconds for attachment
-malloc	- Indicates use of PETSc error-checking malloc (on by default for debug version of libraries) (deprecated, use -malloc_debug)

Use command line arguments

- Views objects: -vec_view, -snes_view, –mat_view
- Display the residual: -ksp_monitor or graphically –ksp_monitor_draw
  -snes_monitor
- Display the true residual: -ksp_monitor_true_residual
- Display the spectrum: -ksp_monitor_singular_value

Learn from the tutorials

We copied snes/tutorial/ex5.c to MAE5032-2022-spring/petsc-tutorial-code/ex1, and rename it as ex1.c

```
make ex1.out

mpirun ./ex1.out -snes_monitor -snes_view

mpirun ./ex1.out -snes_type newtontr -snes_monitor -snes_view

mpirun ./ex1.out -ksp_monitor -snes_monitor -snes_view

mpirun ./ex1.out -pc_type jacobi -ksp_monitor -snes_monitor -snes_view

mpirun ./ex1.out -ksp_type bicg -ksp_monitor -snes_monitor -snes_view
```

One Simple Example

Using MPI

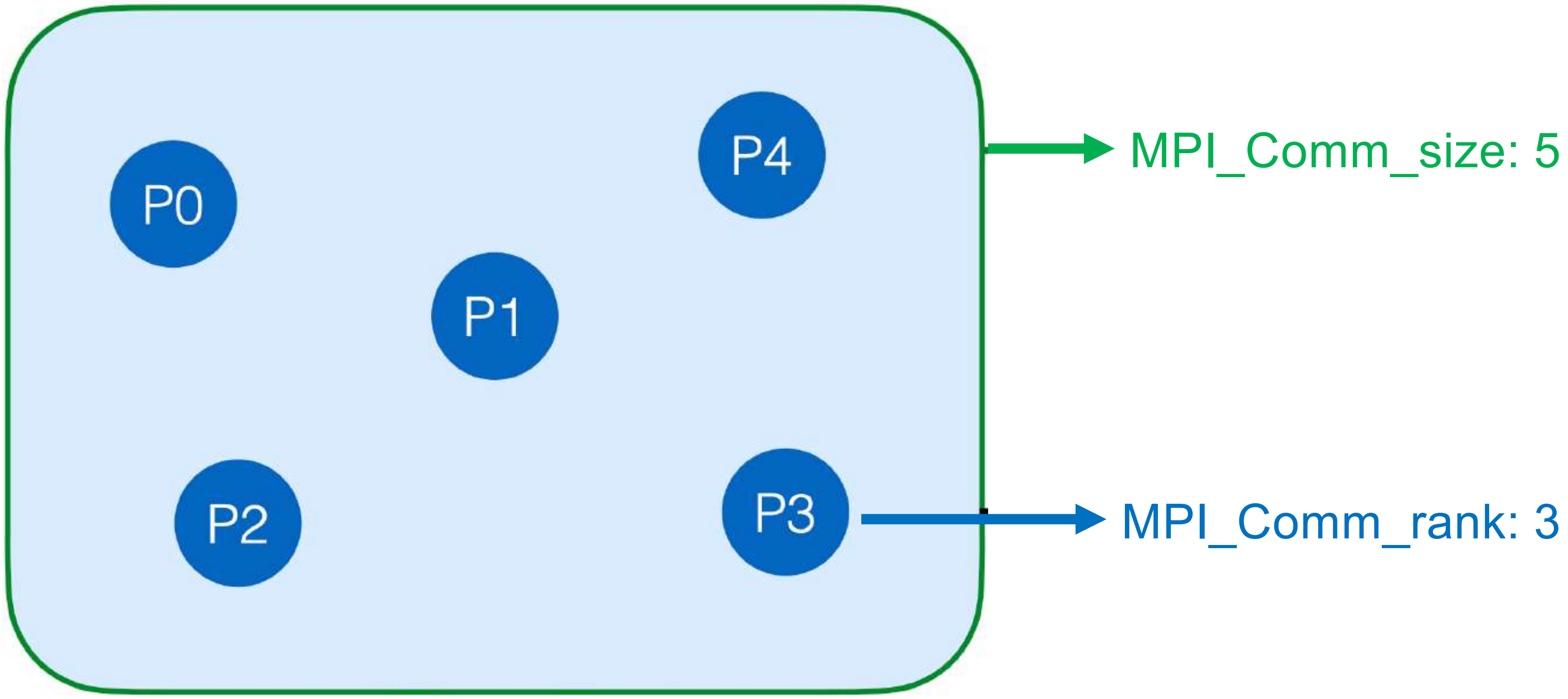
- The Message Passing Interface is:
  - ➢ a library for parallel communication
  - ➢ a system for launching parallel jobs (mpirun/mpiexec)
    - o `mpiexec -n 4 ./a.out`
    - o `$PETSC_DIR/lib/petsc/bin/petscmpiexec -n 4 ./a.out`
  - ➢ a community standard (MPICH, OpenMPI, ……)

- Yet, you rarely need to make explicit MPI calls with PETSc in HPC.

- Communicator: A context (or scope) for parallel communication
  - There are two defaults: yourself (PETSC_COMM_SELF), everyone launched (PETSC_COMM_WORLD)
  - Can create new communicators by splitting existing ones.
  - Point-to-point communication: happens between two processes (like in MatMult())
  - Reduction operations: happens among all processes (like VecNorm()).

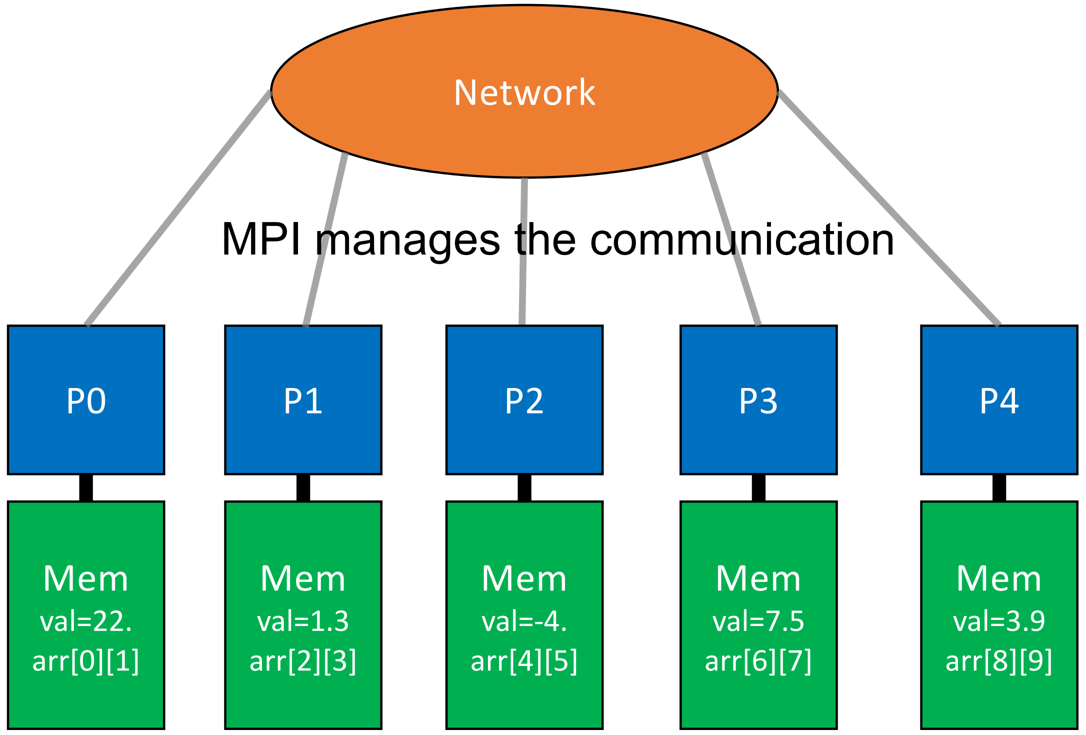
Writing a basic PETSc program

- A communicator owns a group of processes that can communicate information among them. Each process has a unique rank within the communicator.
- PETSc uses MPI and thus one may call MPI routines directly.



Writing a basic PETSc program

- A communicator owns a group of processes that can communicate information among them. Each process has a unique rank within the communicator.
- PETSc uses MPI and thus one may call MPI routines directly.



Writing a basic PETSc program

- PetscInitialize(int *argc, char ***argv, char *file, char *help) initialize PETSc and MPI. argc and argv are the command line arguments delivered in C and C++ programs; the argument file indicates an alternative name for the PETSc option file; the argument help is a string that will be printed if the code is run with –help.

- PetscFinalize() needs to be called at the end of the program (for collecting log information).

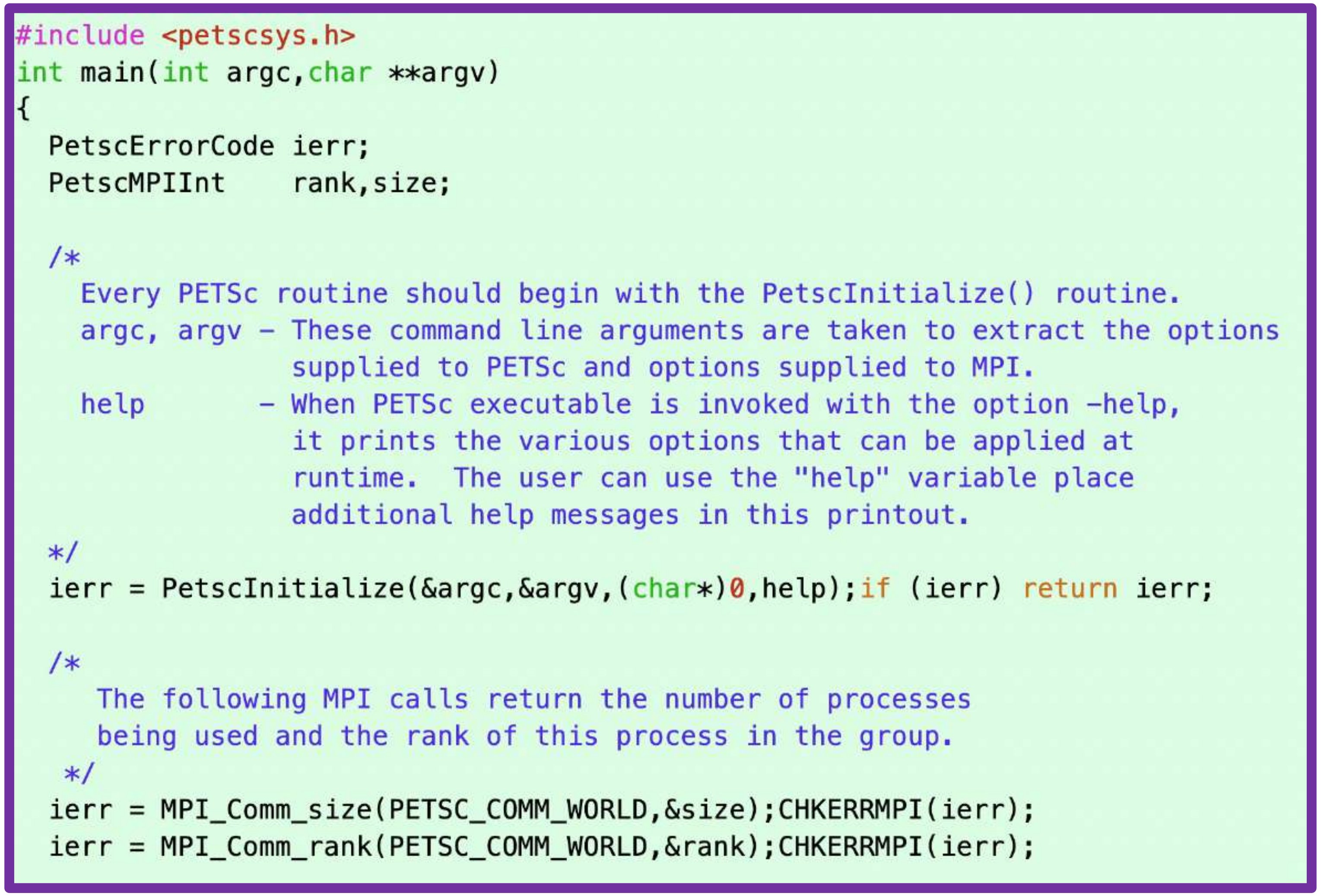
- All petsc function return an integer (ierr) known as error code, indicating if the function execution is successful.

- PetscPrintf(MPI_Comm, char format[], …) prints to standard output from the first processor in the communicator.

Runtime options

- There are options supported by all PETSc programs
  - -help: view a complete list of options available
  - -log_view: summarize the program's performance
  - -fp_trap: stop on floating-point exceptions
  - -malloc_dump: enable memory tracing
  - -malloc_debug: enable memory debugging
  - -start_in_debugger [noxterm,gdb,lldb] start all processes in a debugger
  - -on_error_attach_debugger [noxterm,gdb,lldb] start debugger only on encountering an error
  - -info: print a list of information about the program.

A very simple PETSc code

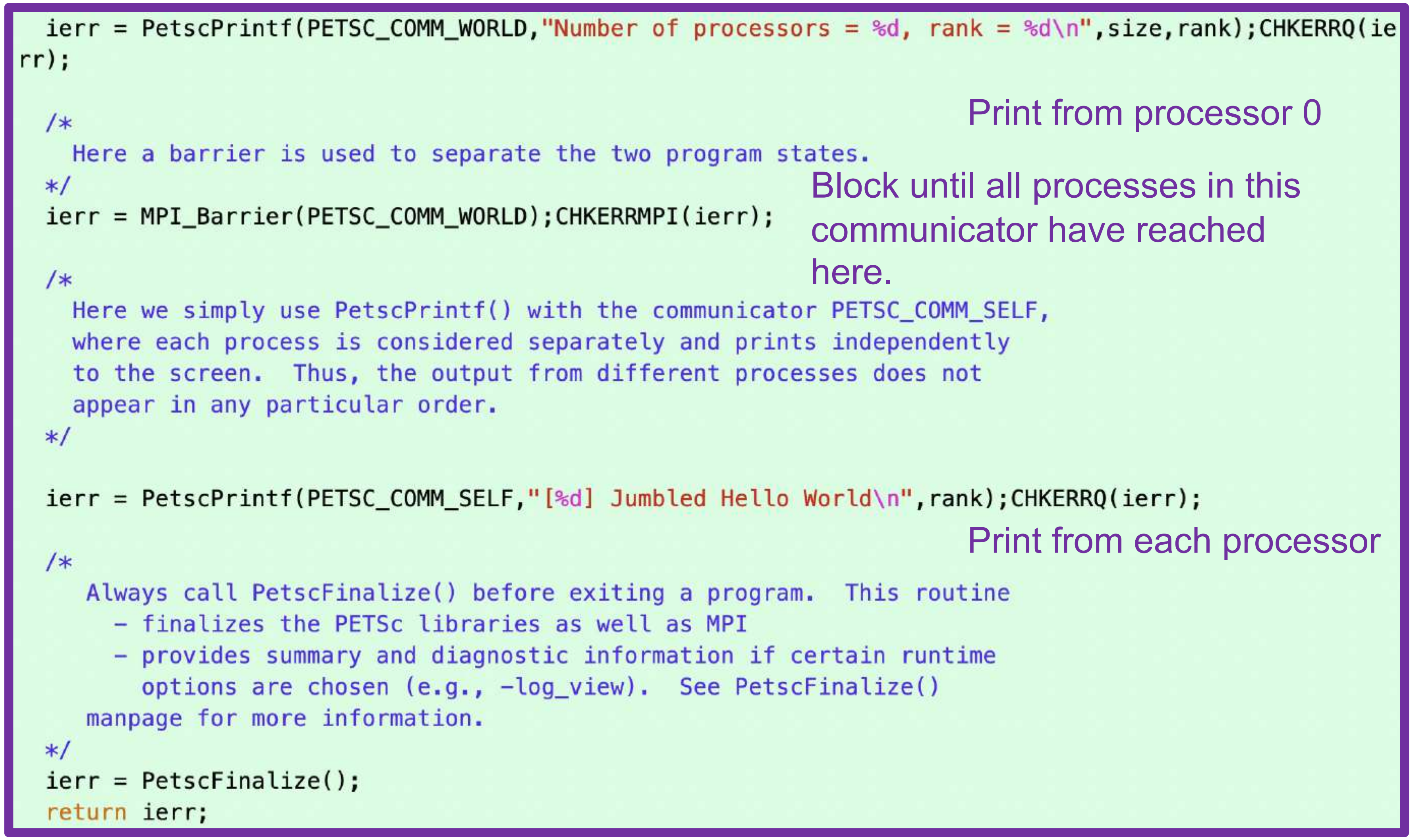


```c
#include <petscsys.h>
int main(int argc,char **argv)
{
  PetscErrorCode ierr;
  PetscMPIInt    rank,size;

  /*
    Every PETSc routine should begin with the PetscInitialize() routine.
    argc, argv - These command line arguments are taken to extract the options
                 supplied to PETSc and options supplied to MPI.
    help       - When PETSc executable is invoked with the option -help,
                 it prints the various options that can be applied at
                 runtime.  The user can use the "help" variable place
                 additional help messages in this printout.
  */
  ierr = PetscInitialize(&argc,&argv,(char*)0,help);if (ierr) return ierr;

  /*
     The following MPI calls return the number of processes
     being used and the rank of this process in the group.
   */
  ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size);CHKERRMPI(ierr);
  ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRMPI(ierr);
```

Initialize PETSc, implicitly initialize MPI

Obtain size and rank from the communicator PETSC_COMM_WORLD

A very simple PETSc code



```
  ierr = PetscPrintf(PETSC_COMM_WORLD,"Number of processors = %d, rank = %d\n",size,rank);CHKERRQ(ie
rr);

  /*
    Here a barrier is used to separate the two program states.
  */
  ierr = MPI_Barrier(PETSC_COMM_WORLD);CHKERRMPI(ierr);

  /*
    Here we simply use PetscPrintf() with the communicator PETSC_COMM_SELF,
    where each process is considered separately and prints independently
    to the screen.  Thus, the output from different processes does not
    appear in any particular order.
  */

  ierr = PetscPrintf(PETSC_COMM_SELF,"[%d] Jumbled Hello World\n",rank);CHKERRQ(ierr);

  /*
     Always call PetscFinalize() before exiting a program.  This routine
       - finalizes the PETSc libraries as well as MPI
       - provides summary and diagnostic information if certain runtime
         options are chosen (e.g., -log_view).  See PetscFinalize()
     manpage for more information.
  */
  ierr = PetscFinalize();
  return ierr;
```

Print from processor 0

Block until all processes in this communicator have reached here.

Print from each processor

Example 2

```
make ex2.out

mpirun -np 2 ./ex2.out

mpirun -np 10 ./ex2.out

mpirun -np 10 ./ex2.out -log_view

mpirun -np 2 ./ex2.out -help
```

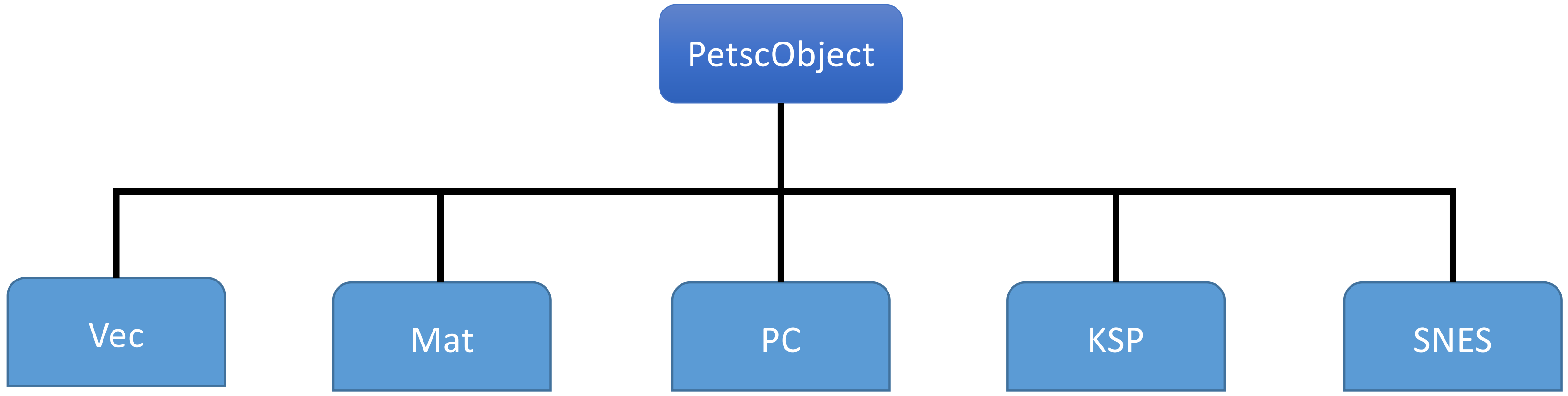
Getting more from online resources

- www.mcs.anl.gov/petsc

- Hyperlinked documentation
  - Manual
  - HTML of all example code with link to manual pages
- FAQ
- Email: petsc-maint@mcs.anl.gov
  petsc-users@mcs.anl.gov

Objects

PETSc object



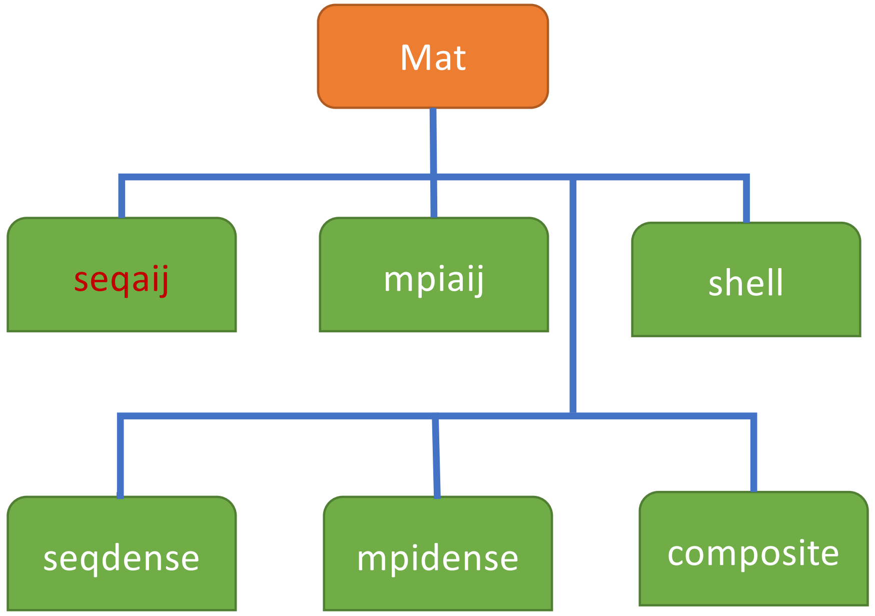
- PETSc is designed with a 3-level inheritance structure
- Every object in PETSc is an instance of a “class”: `Vec, Mat, PC, KSP, SNES,` …
- All classes inherit from PetscObject
- function called on objects (methods) are prefixed with the class name:

  `MatMult` (Mat-prefixed)
- A new object is created with a class-specific Create function (constructor):

  ```
  Mat A; MatCreate(comm, &A);
  ```
- Every class is further refined to types specified with SetType:

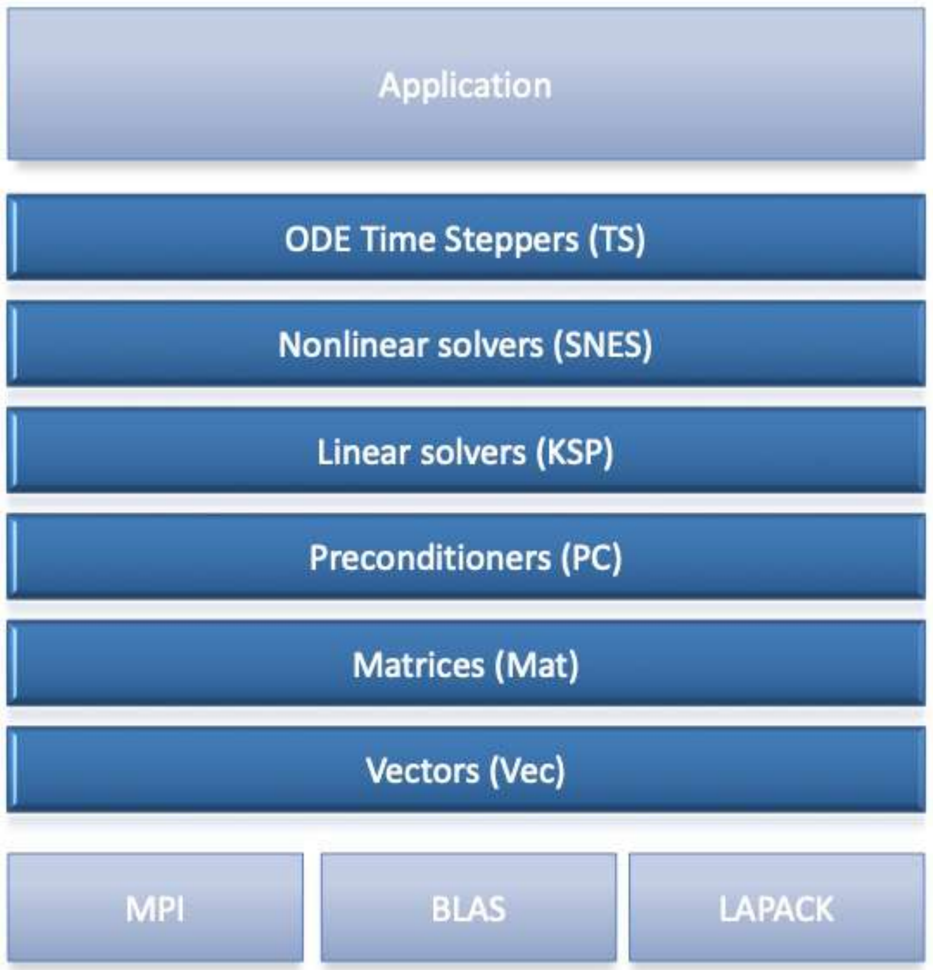
  ```
  MatSetType(A, MATSEQAIJ);
  ```

PETSc inheritance



- Every class is further refined to types specified with SetType:

```
    MatSetType(A, MATSEQAIJ);
```



PETSc inheritance

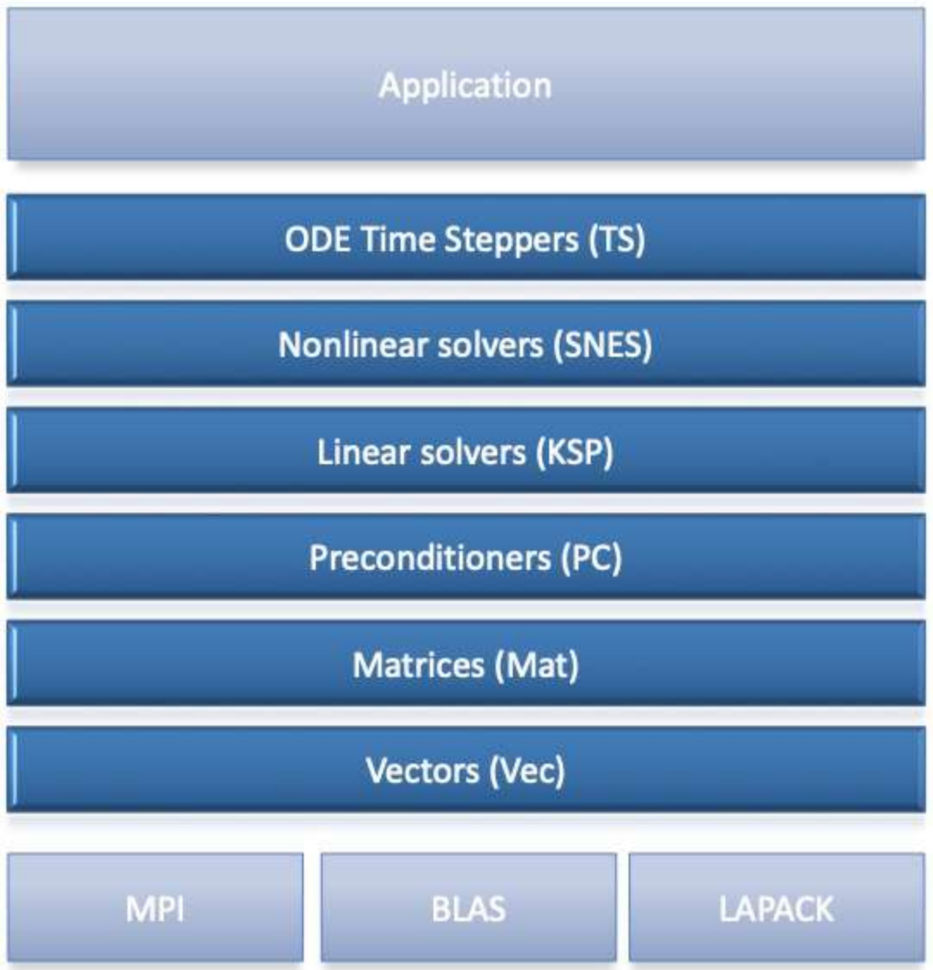
- “Upper” class instance contains “lower” class instance
```
    KSPGetPC(KSP, PC *);
    SNESGetKSP(SNES, KSP *);
```

- Objects are opaque, meaning you do NOT access their data directly. If you really need to touch the internal data, there are functions allowing you to do so.

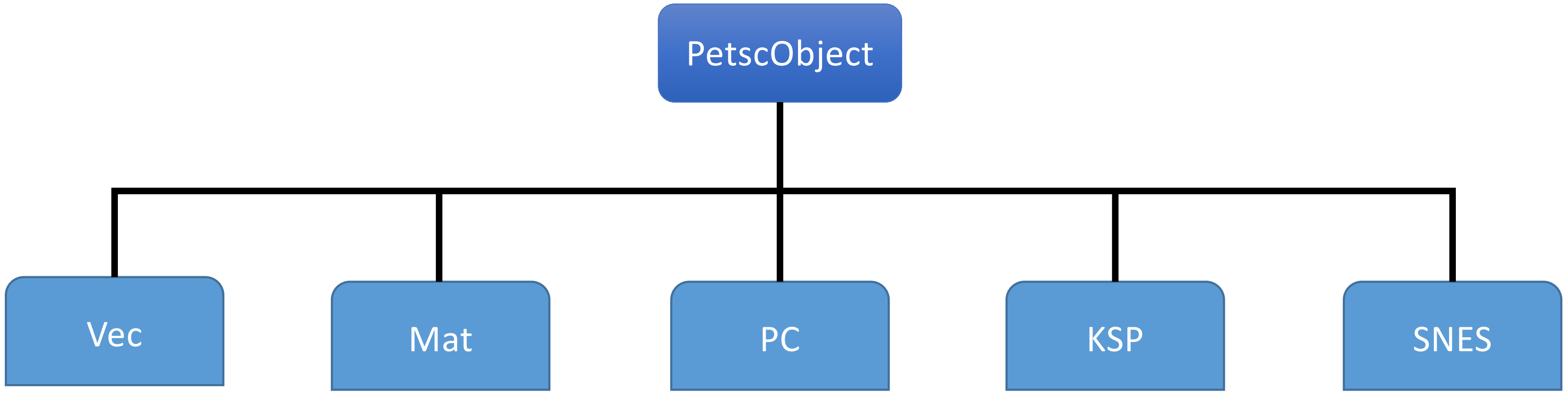
- You may find their definition in (e.g. Mat) include/petscmat.h

- Polymorphism: public interface for all types of, say, matrices: sequential, parallel, dense, sparse, blocked, symmetric, … …

```
MatMult(Mat A, Vec x, Vec y);
```
$\Uparrow$
```
MatMult_SeqDense(Mat A, Vec x, Vec y);
```



PETSc object

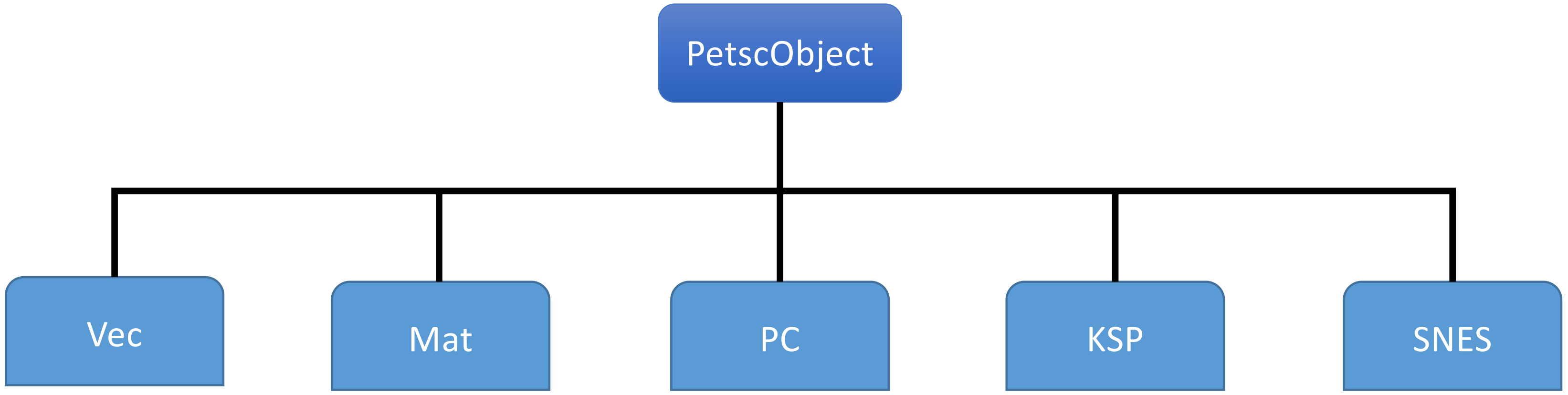


- Every PETSc object can be cast to PetscObject

```
Mat A;
PetscObject obj;
MatCreate(PETSC_COMM_WORLD, &A);
obj = (PetscObject) A;
```

- PetscObject provides general method such as
  - `Get/SetName`(): name the object (used for printing profiling info, command line arguments, etc.)
  - `GetType`: the type of the object
  - `GetComm`: the communicator the object belongs to

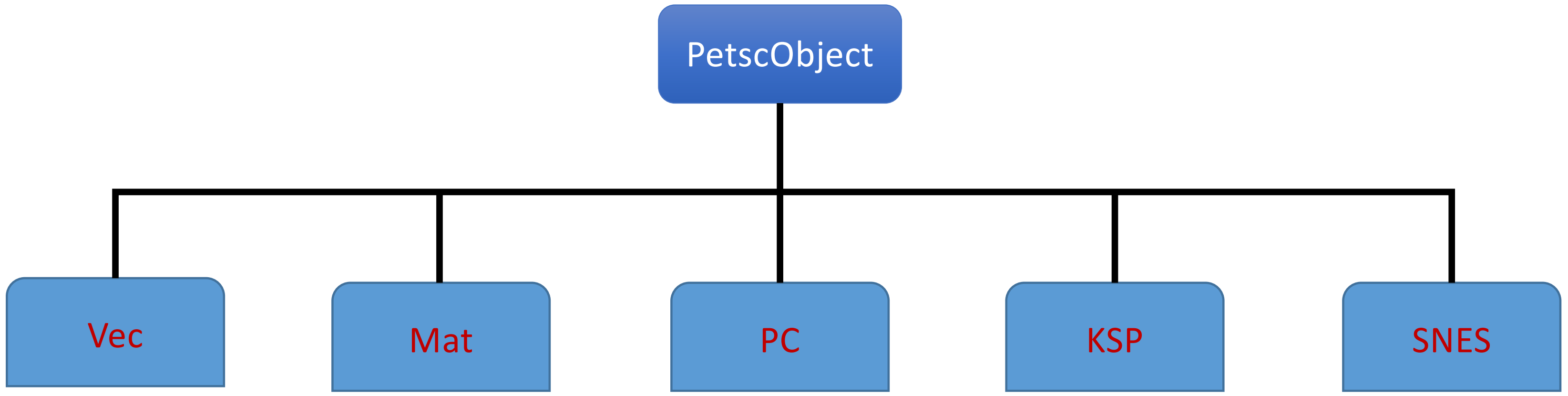
PETSc object



- Every PETSc object can be cast to PetscObject

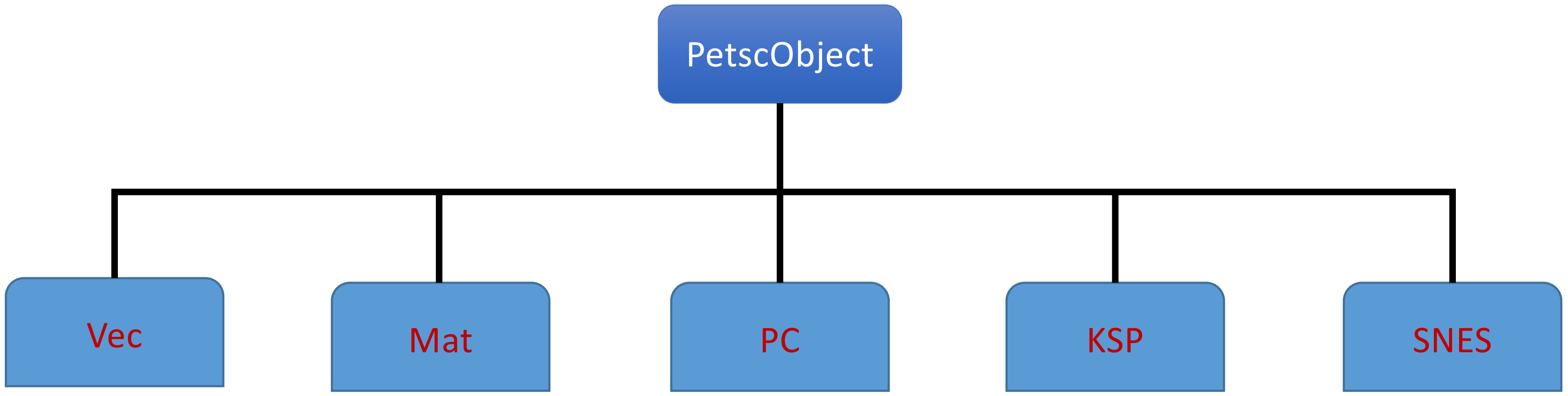
```
Mat A;
const char * type, * name;
MPI_Comm comm;
PetscObjectGetComm((PetscObject)A, &comm);
PetscObjectGetName((PetscObject)A, &name);
PetscObjectGetType((PetscObject)A, &type);
```

PETSc common methods



- Method names are prefixed by the class name: Vec, Mat, KSP, …
- All PETSc built-in classes define the following class-specific methods:
  - Create() : create the object (constructor)
  - Get/SetType() : set the implementation type
  - SetUp() : prepare the object inner state for computation
  - Destroy() : deallocate memory (destructor)
  - View() : print/save object to specified output
  - Load() : load object from specified input

PETSc common methods



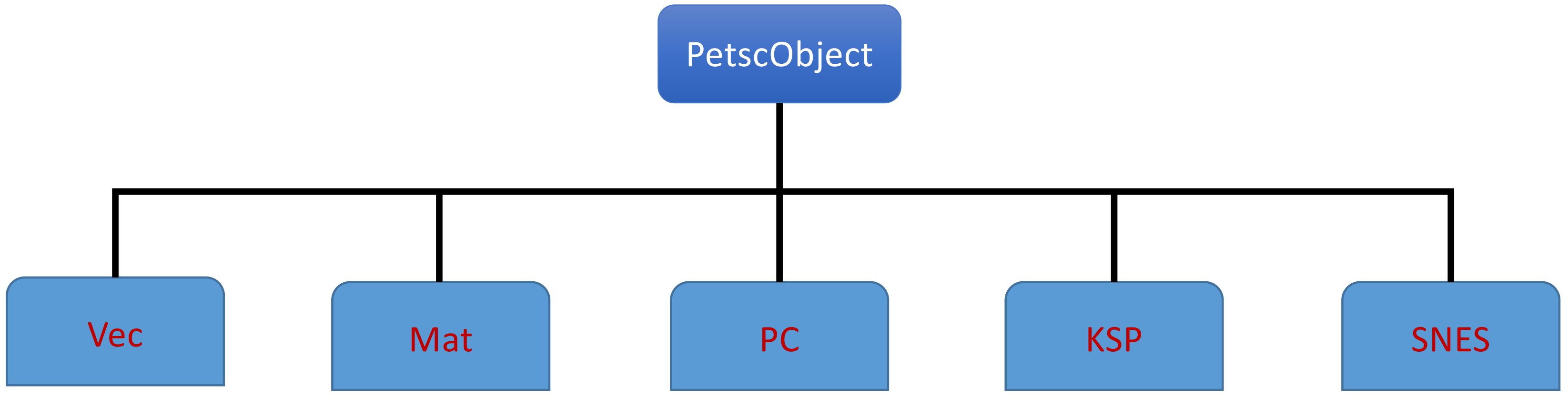
- Method names are prefixed by the class name: Vec, Mat, KSP, …
- All PETSc built-in classes can be controlled through the command line
  - SetFromOptions() : set object properties from the options database (command line or input file)

e.g. `-ksp_type cg` → `KSPSetType(ksp, KSPCG);`

  - Get/SetOptionsPrefix() : set a specific option prefix for the given object.

```
e.g. KSPSetOptionsPrefix(ksp, "stiffness_");
        -stiffness_ksp_type cg        KSPSetType(ksp, KSPCG);
```

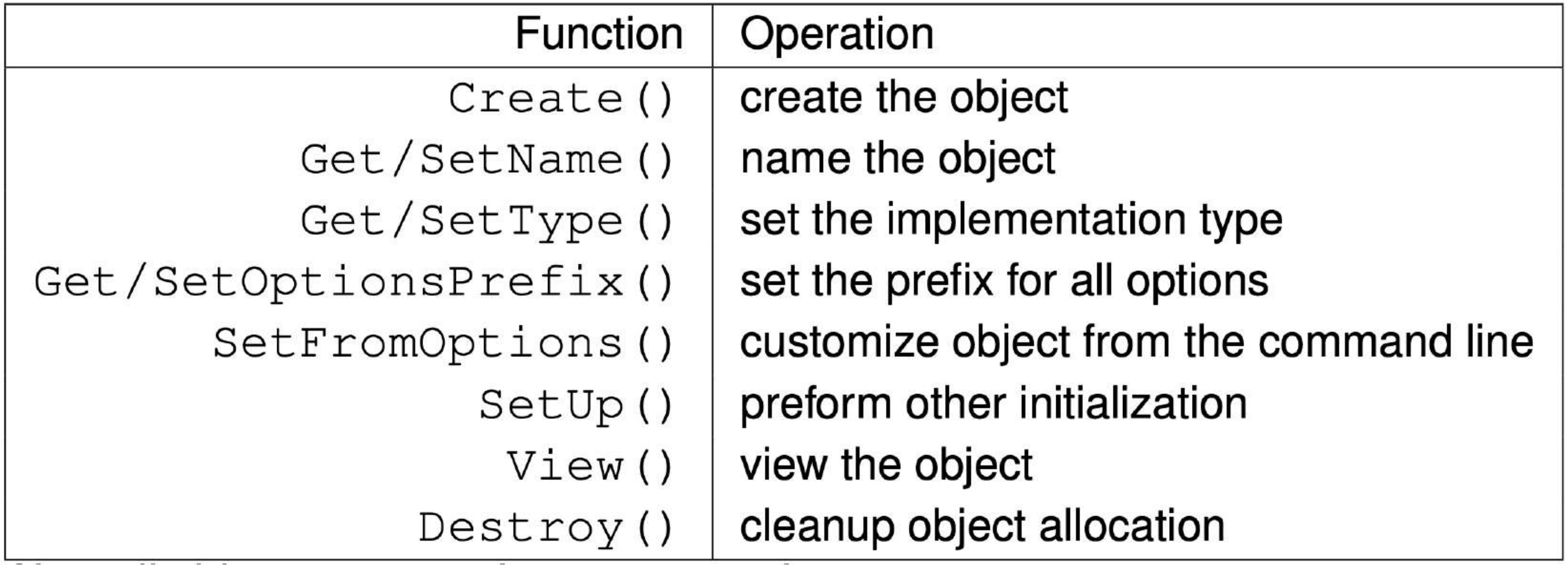
Objects and communicators



- Every object in PETSc belongs to some communicator
- MPI_Comm is the first argument of every object’s constructor
- Two objects can interact only if they belong to the same communicator, e.g. Mat and Vec in matrix-vector product.

Basic PETSc object usage summary

- Every object (Vec, Mat, KPS, PC, SNES, TS, etc) supports a uniform interface.



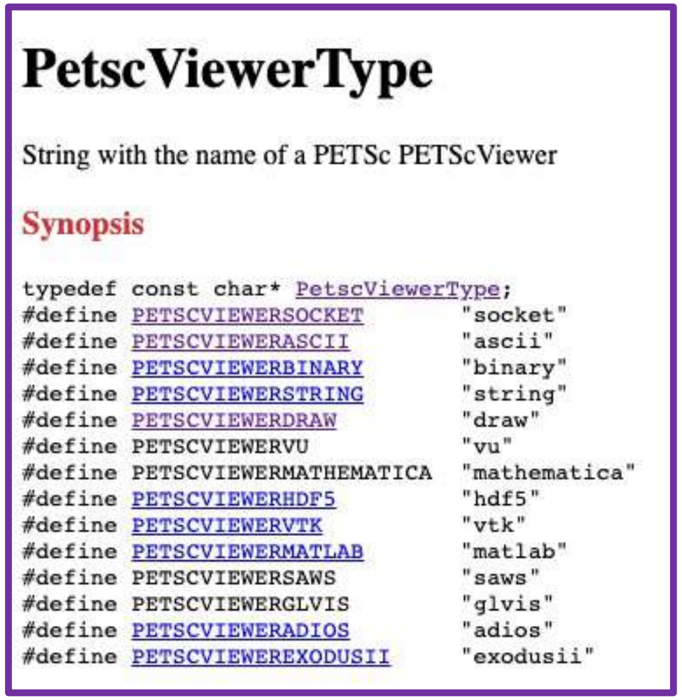
Function	Operation
`Create()`	create the object
`Get/SetName()`	name the object
`Get/SetType()`	set the implementation type
`Get/SetOptionsPrefix()`	set the prefix for all options
`SetFromOptions()`	customize object from the command line
`SetUp()`	preform other initialization
`View()`	view the object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

Viewers

- PetscViewer class is used for printing to stdout.
- Basic usage:

```
PetscViewer viewer;
PetscViewerCreate(comm, &viewer);
PetscViewerSetType(viewer, PETSCVIEWERASCII);
MatView(A, viewer);
VecView(x, viewer);
PetscViewerDestroy(&viewer);
```

- Always destroy objects like in C/C++
  - ➢ no overhead
  - ➢ programmer is responsible for calling destroy functions once object is no longer needed
  - ➢ slightly harder than smart pointers in new C++ standard,Trilinos, Boost, etc.



PetscViewerType

String with the name of a PETSc PETScViewer

Synopsis

```
typedef const char* PetscViewerType;
#define PETSCVIEWERSOCKET        "socket"
#define PETSCVIEWERASCII         "ascii"
#define PETSCVIEWERBINARY        "binary"
#define PETSCVIEWERSTRING        "string"
#define PETSCVIEWERDRAW          "draw"
#define PETSCVIEWERVU            "vu"
#define PETSCVIEWERMATHEMATICA   "mathematica"
#define PETSCVIEWERHDF5          "hdf5"
#define PETSCVIEWERVTK           "vtk"
#define PETSCVIEWERMATLAB        "matlab"
#define PETSCVIEWERSAWS          "saws"
#define PETSCVIEWERGLVIS         "glvis"
#define PETSCVIEWERADIOS         "adios"
#define PETSCVIEWEREXODUSII      "exodusii"
```

Vectors

Vector (Vec) basics

```
Vec v;
PetscInt m=2, M=8;
VecType type=VECMPI;
MPI_Comm comm=PETSC_COMM_WORLD;
```

Sequential alternative:

```
VECSEQ;
PETSC_COMM_SELF
```

`VecCreate(comm, &v);`	Creation
`VecSetSizes(v,m,M);`	Layout
`VecSetType(v,type);`	Type
`VecSetFromOptions(v);`	Enable options
`VecDestroy(v);`	Dealloc

VecSetSizes(v, M, M);

Vector (Vec) basics

```
Vec v;
PetscInt m=2, M=8;
VecType type=VECMPI;
MPI_Comm comm=PETSC_COMM_WORLD;
```

```
VecCreate(comm, &v);      Creation
VecSetSizes(v,m,M);       Layout
VecSetType(v,type);       Type
```

```
VecCreateMPI(comm, m, M, &v);
```

```
VecSetFromOptions(v);     Enable options
VecDestroy(&v);           Dealloc
```

Sequential alternative:

```
VECSEQ;
PETSC_COMM_SELF
```

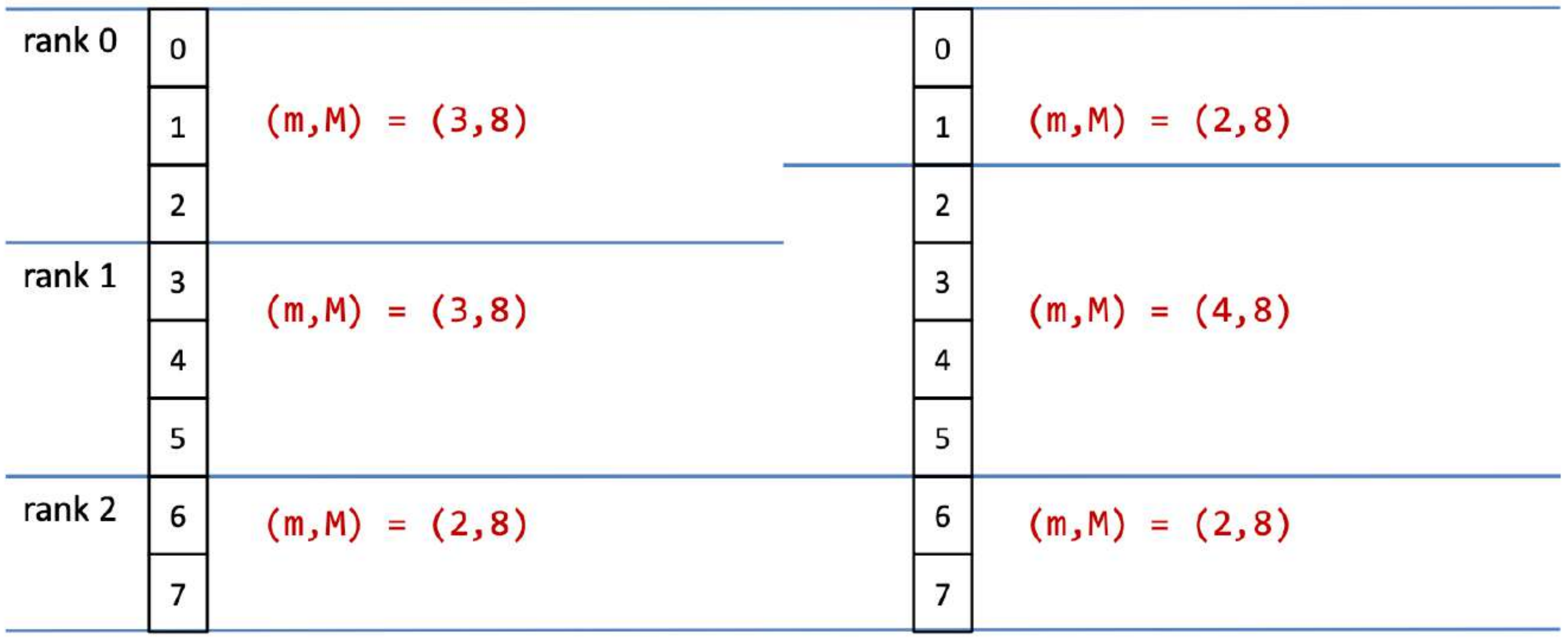
```
VecSetSizes(v, M, M);
```

```
VecCreateSeq(comm,M,&v);
```

Vector (Vec) parallel layout

Consider the vector with local size m, global size M, distributed across 3 processes

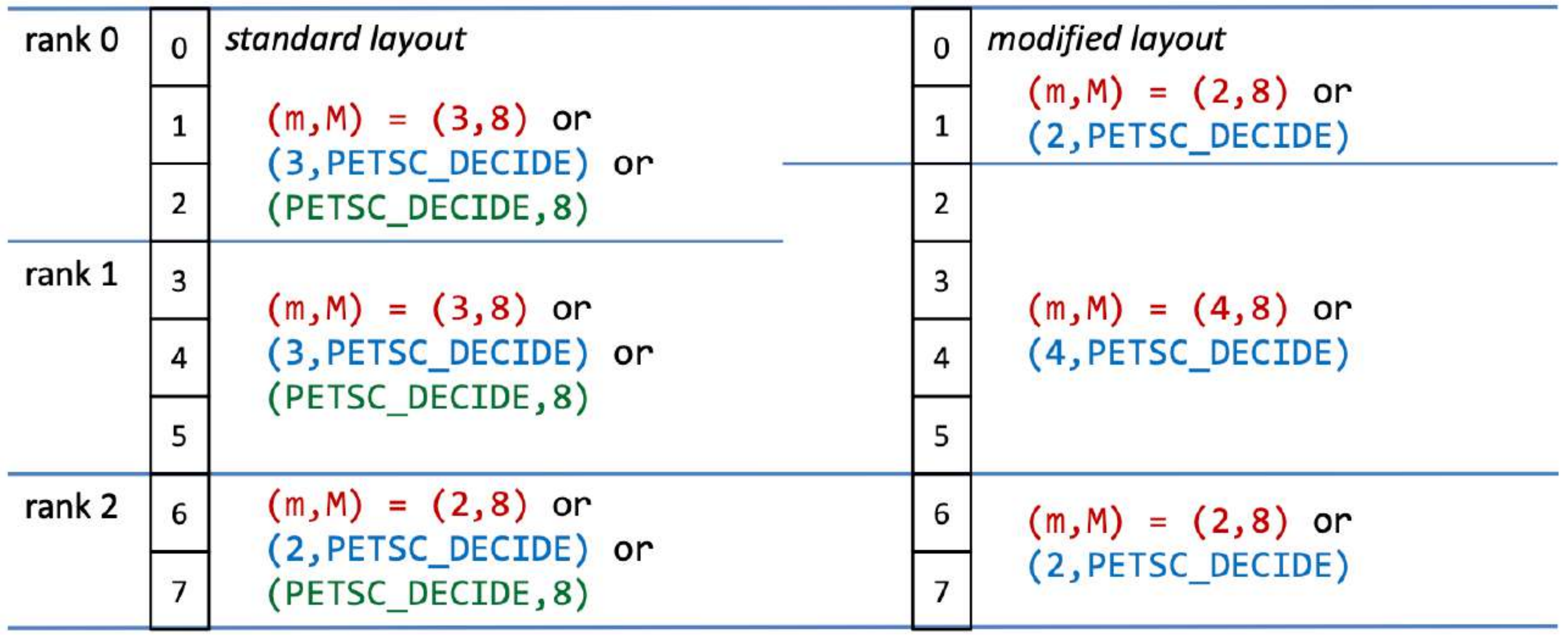
Call VecSetSizes(v,m,M) to set the layout of the vector



In MPI, each process owns its own memory, meaning the same variable name in different processes have different values.

Vector (Vec) parallel layout

Set either m or M to PETSC_DECIDE to enforce standard layout.



- Query the layout: `VecGetLocalSize(v, &m)` and `VecGetSize(v, &M)`.
- **Global indices** of the first and last local entries: `VecGetOwnershipRange(v.&low,&high);`
- Create another vector with the **same type and layout**: `VecDuplicate(v, &w);`

Vector (Vec) set all values at once

- Copy the entries from v to w

```
Vec v, w;
VecDuplicate(v, &w);         // same laylout, undefined values
VecCopy(v,w);                // v = w, w already defined
```

- Initialize values of Vec

```
VecSet(v, 1.0);              // set all entries to the same value 1.0
VecSetRandom(v, NULL);       // set pseudo-random values

PetscRandom r;
PetscRandomCreate(comm, &r);
VecSetRandom(x,r);           // using a specific seed, e.g. random123
PetscRandomDestroy(&r);
```

D E Shaw Research

Random123

Vector (Vec) set all values at once

- Set an individual element (global indexing):

```
Vec x; PetscInt i = 10; PetscReal v = 3.14;
VecSetValue(x, i, v, INSERT_VALUES);
```

OR

```
VecSetValues(x, 1, &i, &v, INSERT_VALUES);
```

- Set multiple entries at once

```
PetscInt ii[] = {1,2}; PetscReal vv[] = {2.7, 3.1};
VecSetValues(x, 2, ii, vv, INSERT_VALUES);
```

- The last argument can be

`INSERT_VALUES`	replace original value **(=)**
`ADD_VALUES`	add the new values to the original ones **(+=)**

Assembly a vector

- VecSetValues is purely local with **no** inter-process **communication**.

- Entries need NOT be generated locally (local means the process on which they are stored.) Instead, their values are **cached** locally.

- PETSc automatically moves data if necessary, which happens during the **assembly** stage.

- To set values of a vector, there are three steps:
    - Each processes sets or addes values:mode = INSERT_VALUES/ADD_VALUES

```
VecSetValues(Vec v, PetscInt n, PetscInt rows[],
                    PetscScalar values[], InsertMode mode)
```

    - Begin communication to send values to the correct process
    - Complete the communication.

```
VecAssemblyBegin(Vecv)
VecAssemblyEnd(Vecv)
```

One way to set the components of a vector

```
VecGetSize(x, &N);
MPI_Comm_rank(PETSC_COMM_WORLD, &rank);
if (rank == 0) {
  val = 0.0;
  for(i = 0; i < N; ++i) {
    VecSetValues(x, 1, &i, &val, INSERT_VALUES);
    val += 10.0;
  }
}
/* These routines ensure that the data is
   distributed to the other processes */
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

A better way to set the components of a vector

```
VecGetOwnershipRange(x, &low, &high);
val = low*10.0;
for(i = low; i < high; ++i) {
  VecSetValues(x, 1, &i, &val, INSERT_VALUES);
  val += 10.0;
}
/* No data will be communicated here */
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

Getting values

Vec x;

- Get a copy of 2 local entries of x with global indices ix to array v

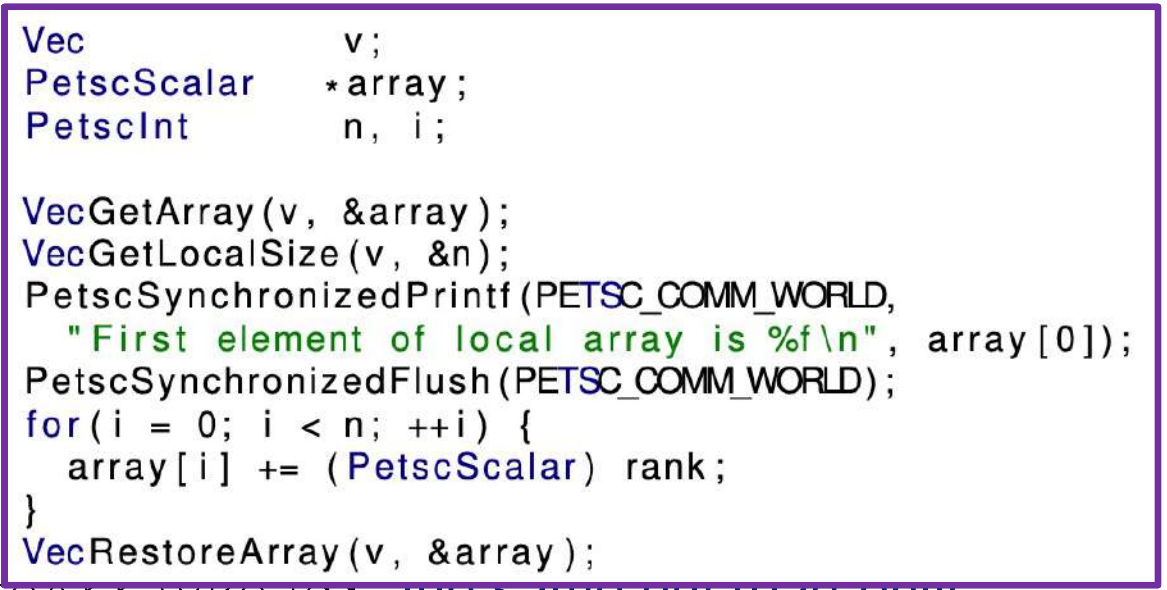
```
    PetscInt ix[] = {10, 20};
    PetscScalar v[2];
    VecGetValues(x, 2, ix,
```

- Get the pointer to the whole local

```
    PetscScalar *a;
    VecGetArray(x, &a);
```

    // Work on the array a

```
    VecRestoreArray(x, &a)
```



```
Vec              v;
PetscScalar     *array;
PetscInt         n, i;

VecGetArray(v, &array);
VecGetLocalSize(v, &n);
PetscSynchronizedPrintf(PETSC_COMM_WORLD,
  "First element of local array is %f\n", array[0]);
PetscSynchronizedFlush(PETSC_COMM_WORLD);
for(i = 0; i < n; ++i) {
  array[i] += (PetscScalar) rank;
}
VecRestoreArray(v, &array);
```

- `VecGetArrayRead/VecRestoreArrayRead:` the same but read-only.

- GetArray functions are fast. You do not need to worry about the overhead.

Selected vector operations

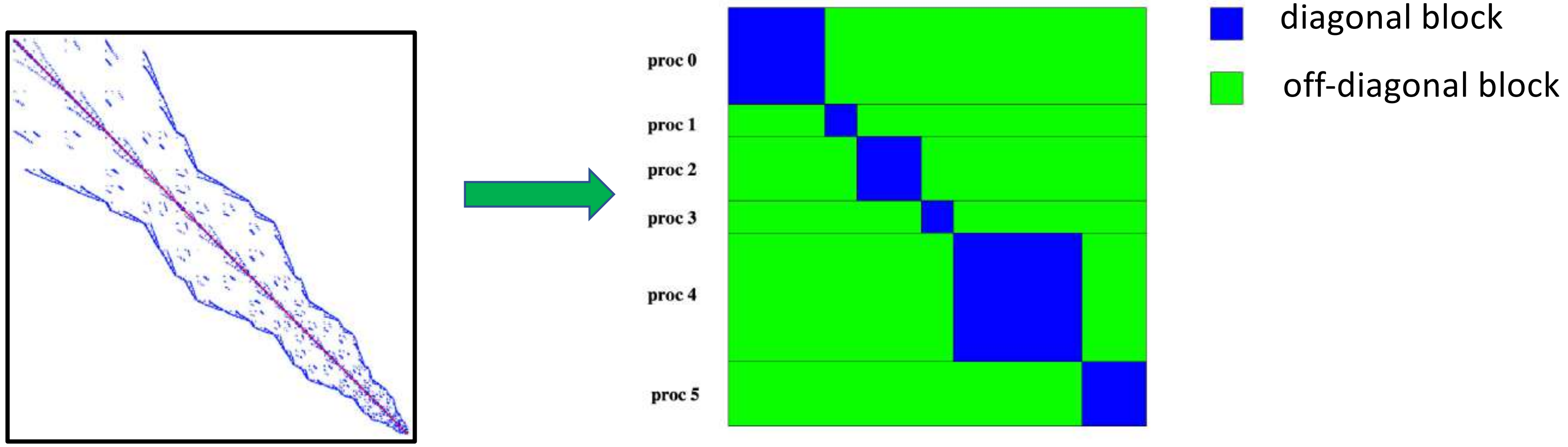
Function Name	Operation
VecAXPY(Vec y, PetscScalar a, Vec x)	$y = y + a * x$
VecAYPX(Vec y, PetscScalar a, Vec x)	$y = x + a * y$
VecWAYPX(Vec w, PetscScalar a, Vec x, Vec y)	$w = y + a * x$
VecScale(Vec x, PetscScalar a)	$x = a * x$
VecCopy(Vec y, Vec x)	$y = x$
VecPointwiseMult(Vec w, Vec x, Vec y)	$w_i = x_i * y_i$
VecMax(Vec x, PetscInt *idx, PetscScalar *r)	$r = \max r_i$
VecShift(Vec x, PetscScalar r)	$x_i = x_i + r$
VecAbs(Vec x)	$x_i = \|x_i\|$
VecNorm(Vec x, NormType type, PetscReal *r)	$r = \|\|x\|\|$

Matrices

Mat basics

- PETSc matrices are
  - fundamental objects for storing stiffness matrices, Jacobians, etc.
  - Each process locally owns a contiguous set of rows
  - Supports many data types: AIJ, Block AIJ, Symmetric AIJ, etc.
  - Supports structures for many packages: MUMPS, SuperLU, UMFPack, etc.

- Parallel sparse matrix:
  - each process owns a submatrix of contiguous global rows;
  - each process consists of diagonal and off-diagonal parts



Matrix (Mat) basics

```
Mat A;
PetscInt m=2, n=3, M=8, N=12;
MatType type=MATMPIAIJ;
MPI_Comm comm=PETSC_COMM_WORLD;

MatCreate(comm, &A);     // Creation
MatSetSizes(A,m,n,M,N);  // Layout
MatSetType(A,type);      // Type
MatMPIAIJSetPreallocation(A, 5, PETSC_NULL, 5, PETSC_NULL); // prealloc
MatSetUp(A);             // Setup
MatSetFromOptions(v);    // Enable options
MatDestroy(v);           // Dealloc
```

Sequential alternative:

```
MATSEQAIJ;
PETSC_COMM_SELF
MatSetSizes(A, M, N, M, N);
MatSeqAIJSetPreallocation(A,5,PETSC_NULL);
```

Matrix (Mat) basics

```
Mat A;
PetscInt m=2, n=3, M=8, N=12;
MatType type=MATMPIAIJ;
MPI_Comm comm=PETSC_COMM_WORLD;
```

Sequential alternative:

```
MATSEQAIJ;
PETSC_COMM_SELF
```

```
MatCreate(comm, &A);      // Creation
MatSetSizes(A,m,n,M,N);   // Layout
MatSetType(A,type);       // Type
MatMPIAIJSetPreallocation(A, 5, PETSC_NULL, 5, PETSC_NULL); // prealloc
```

MatSetSizes(A, M, N, M, N);

```
MatCreateMPIAIJ(comm,m,n,M,N,5,PETSC_NULL,5,PETSC_NULL, &A); // All-in-one
```

```
MatSetUp(A);              // Setup
MatSetFromOptions(v);     // Enable options
MatDestroy(v);            // Dealloc
```

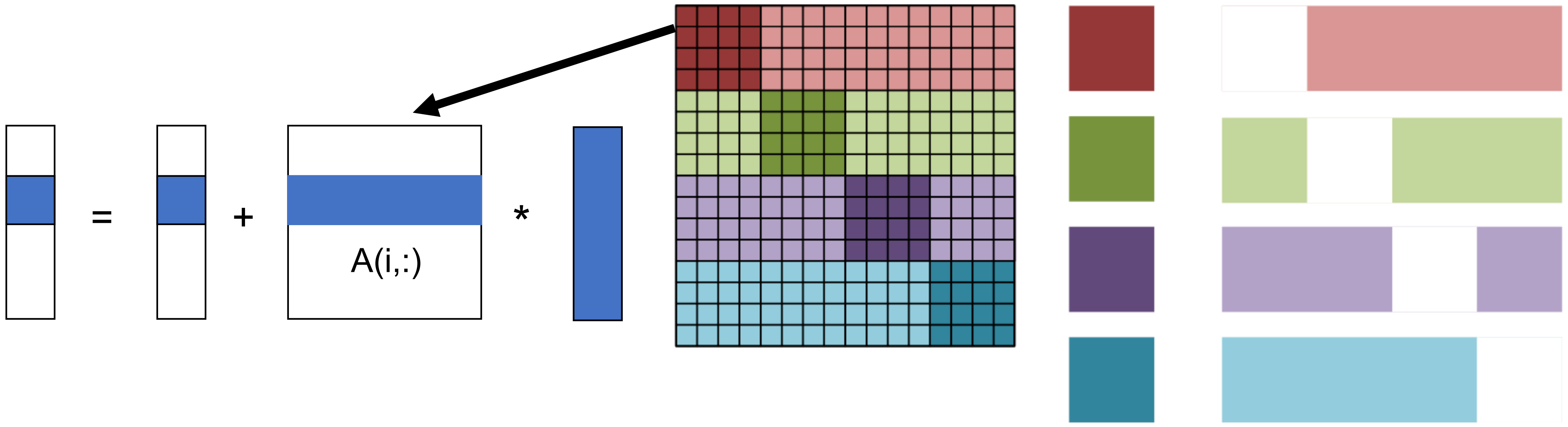
MatSeqAIJSetPreallocation(A,5,PETSC_NULL);

Matrix types

- MATAIJ, MATSEQAIJ, MATMPIAIJ
  - basic sparse format, known as compressed row format, CRS, Yale
  - MATAIJ = MATSEQAIJ if the communicator contains only one process, otherwise, MATAIJ=MATMPIAIJ
- MATBAIJ, MATSEQBAIJ, MATMPIBAIJ
  - extension of the AIJ format by storing matrix in terms of small fixed-size dense blocks (that fit into cache)
  - intended for use with PDEs with multiple DOFs per mesh node
- MATDENSE, MATSEQDENSE, MATMPIDENSE
  - plain dense matrix stored column-wise (like Fortran).

Mat basics

- MatXAIJPreallocation(Mat, int dnz, int dnnz[], int onz, int onnz[]);
  - X=Seq, MPI, etc.
  - dnz: expected number of nonzeros in any row in the diagonal block
  - dnnz[i]: expected number of nonzeros in rwo i in the diagonal block
  - onz: expected number of nonzeros in any row in the off-diagonal portion
  - onnz[i]: expected number of nonzeros in row i in the off-diagonal portion



Mat basics

- MatXAIJPreallocation(Mat, int dnz, int dnnz[], int onz, int onnz[]);
    - X=Seq, MPI, etc.
    - dnz: expected number of nonzeros in any row in the diagonal block
    - dnnz[i]: expected number of nonzeros in rwo i in the diagonal block
    - onz: expected number of nonzeros in any row in the off-diagonal portion
    - onnz[i]: expected number of nonzeros in row i in the off-diagonal portion

$$\begin{array}{l} \\ \text{CPU 0} \\ \\ \\ \text{CPU 1} \\ \\ \\ \text{CPU 2} \\ \\ \end{array} \left(\begin{array}{ccc|ccc|cc} 1 & 2 & 0 & 0 & 3 & 0 & 0 & 4 \\ 0 & 5 & 6 & 7 & 0 & 0 & 8 & 0 \\ 9 & 0 & 10 & 11 & 0 & 0 & 12 & 0 \\ \hline 13 & 0 & 14 & 15 & 16 & 17 & 0 & 0 \\ 0 & 18 & 0 & 19 & 20 & 21 & 0 & 0 \\ 0 & 0 & 0 & 22 & 23 & 0 & 24 & 0 \\ \hline 25 & 26 & 27 & 0 & 0 & 28 & 29 & 0 \\ 30 & 0 & 0 & 31 & 32 & 33 & 0 & 34 \end{array}\right)$$

dnz = 2 or dnnz = {2,2,2}
onz = 2 or onnz = {2,2,2}

dnz = 3 or dnnz = {3,3,2}
onz = 2 or onnz = {2,1,1}

dnz = 1 or dnnz = {1,1}
onz = 4 or onnz = {4,4}

Assembly a matrix

- Set an individual element (global indexing):

```
Mat A; PetscInt i=1, j=2; PetscReal v = 3.14;
MatSetValue(A, i, j, v, INSERT_VALUES);  // one value
```

OR

```
MatSetValues(A, 1, &i, 1, &j, &v, INSERT_VALUES);  // array
```

- Set multiple entries at once

```
PetscInt ii[] = {1,2}, jj[]={11,12};
PetscReal vv[] = {2.7, 3.1, 4.5, -1.2};
MatSetValues(A, 2, ii, 2, jj, vv, INSERT_VALUES);
```

- In MatSetValues, vv is row-oriented, and index is 0-based.

- The last argument can be
    - `INSERT_VALUES` replace original value **(=)**
    - `ADD_VALUES` add the new values to the original ones **(+=)**

Assembly a matrix

- MatSetValues is purely **local** with no inter-process communication
- Values are locally cached
- Call the **assembly** function pair to exchange **values** among processors.

```
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

Assembly a matrix

- MatSetValues is purely **local** with no inter-process communication
- Values are locally cached
- Call the **assembly** function pair to exchange **values** among processors.

```
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
// You can do something here
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

- Communication can take place simultaneously with computation (rarely used)

- Cannot mix **inserting** and **adding** values!

```
MatSetValues(A, …, INSERT_VALUES);
MatAssemblyBegin(A, MAT_FLUSH_ASSEMBLY);
MatAssemblyEnd(A, MAT_FLUSH_ASSEMBLY);
MatSetValues(A, …, ADD_VALUES);
```

- MAT_FINAL_ASSEMBLY: final assembly to make A ready to use
- MAT_FLUSH_ASSEMBLY: cheaper, sufficient for switching between insert/add.

Assembly Matrices

- You need MatSetValues with MathAssemblyBegin/End to assemble a matrix

```
v[0] = -1.0; v[1] = 2.0; v[2] = -1.0;
if (rank == 0) {
  for(row = 0;  row < N; row++) {
    cols[0] = row-1; cols[1] = row; cols[2] = row+1;
    if (row == 0) {
      MatSetValues(A,1,&row,2,&cols[1],&v[1],
                          INSERT_VALUES);
    } else if (row == N-1) {
      MatSetValues(A,1,&row,2,cols,v,INSERT_VALUES);
    } else {
      MatSetValues(A,1,&row,3,cols,v,INSERT_VALUES);
    }
  }
}
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

Assembly Matrices: a better one

```
v[0] = -1.0; v[1] = 2.0; v[2] = -1.0;
for(row = start;  row < end; row++) {
  cols[0] = row-1; cols[1] = row; cols[2] = row+1;
  if (row == 0) {
    MatSetValues(A,1,&row,2,&cols[1],&v[1],
                             INSERT_VALUES);
  } else if (row == N-1) {
    MatSetValues(A,1,&row,2,cols,v,INSERT_VALUES);
  } else {
    MatSetValues(A,1,&row,3,cols,v,INSERT_VALUES);
  }
}
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

Keep all processors busy;   Less communications in the assembly functions.

Getting values

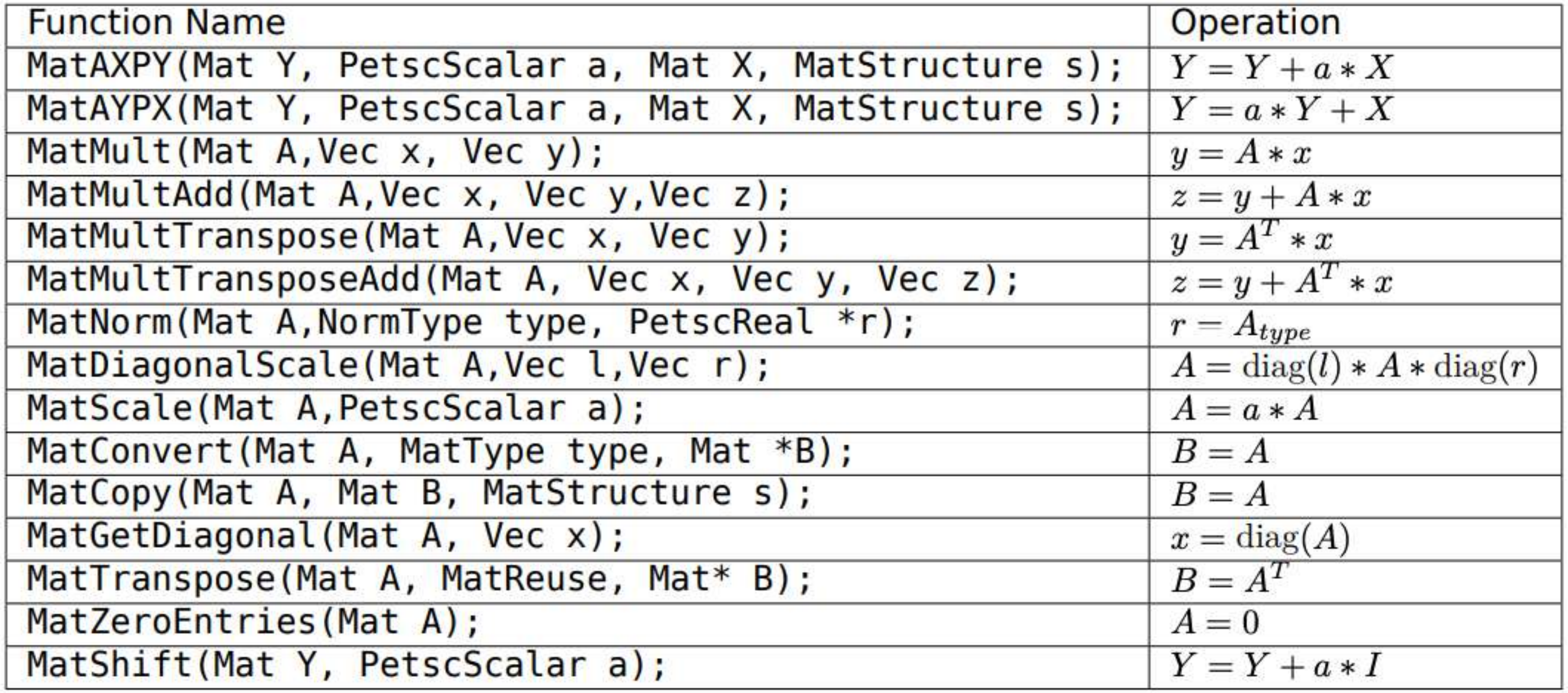
- Get a copy of 3x2 local block of A with global row indices ii and global column indices jj to an array vv:

```
PetscInt ii[ ] = {11,22,33}; PetscInt jj[ ] = {12, 24};
PetscScalar vv[6];
MatGetValues(A, 3, ii, 2, jj, vv);
```

- Get the row of the matrix A

```
PetscInt ncols;
const PetscInt *cols;
const PetscScalar *vals;
MatGetRow(A, i, &ncols, &cols, &vals);
// Access to arrays cols and vals
MatRestoreRow(A, i, &ncols, &cols, &vals);
```

More matrix operations



Function Name	Operation
`MatAXPY(Mat Y, PetscScalar a, Mat X, MatStructure s);`	$Y = Y + a * X$
`MatAYPX(Mat Y, PetscScalar a, Mat X, MatStructure s);`	$Y = a * Y + X$
`MatMult(Mat A,Vec x, Vec y);`	$y = A * x$
`MatMultAdd(Mat A,Vec x, Vec y,Vec z);`	$z = y + A * x$
`MatMultTranspose(Mat A,Vec x, Vec y);`	$y = A^T * x$
`MatMultTransposeAdd(Mat A, Vec x, Vec y, Vec z);`	$z = y + A^T * x$
`MatNorm(Mat A,NormType type, PetscReal *r);`	$r = A_{type}$
`MatDiagonalScale(Mat A,Vec l,Vec r);`	$A = \mathrm{diag}(l) * A * \mathrm{diag}(r)$
`MatScale(Mat A,PetscScalar a);`	$A = a * A$
`MatConvert(Mat A, MatType type, Mat *B);`	$B = A$
`MatCopy(Mat A, Mat B, MatStructure s);`	$B = A$
`MatGetDiagonal(Mat A, Vec x);`	$x = \mathrm{diag}(A)$
`MatTranspose(Mat A, MatReuse, Mat* B);`	$B = A^T$
`MatZeroEntries(Mat A);`	$A = 0$
`MatShift(Mat Y, PetscScalar a);`	$Y = Y + a * I$

Please refer to the manual for more info.

Some important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
4. Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

The red ones are non-sparse and we prefer not assemble them. There are matrices that implement via algorithms in PETSc.

Implicit matrices

- Some of the matrix types are not stored by elements but they **behave** like matrices in some operations.
- nomenclature:
  { matrix-free | implicit | unassembled } { matrices | linear operators }

- The most important operation is a matrix-vector product (**MatMult**), which can be considered as an application of a linear operator.

- In Krylov methods, this operation is sufficient to solve a linear system.

- Example 1: MatComposite

```
Mat F,G;
Mat arr[3] = {C, B, A}; /* reverse order! */

/* F = A*B*C (implicitly) */
MatCreateComposite(comm, 3, arr, &F);
MatCompositeSetType(F, MAT_COMPOSITE_MULTIPLICATIVE);
```

Implicit matrices

- Example 2: MatShell – User implement the operation and pass the function pointer to MatShellSetOperation.

```
Mat A;
PetscInt m,n,M,N;
MyType Adata;
...
MatCreate(comm,&A);
MatSetSizes(A,m,n,M,N);
MatSetType(A,MATSHELL);
MatShellSetContext(A,Adata);
MatShellSetOperation(A,MATOP_MULT,(void(*)(void))mymatmult);
...
```

```
/* user-defined matrix-vector multiply */
PetscErrorCode mymatmult(Mat mat,Vec in,Vec out) {
  MyType *matData;

  PetscFunctionBegin;
  MatShellGetContext(mat, &matData);
  /* compute out from in, using matData */
  PetscFunctionReturn(0);
}
```

Linear system solvers

Krylov methods

Question: What can we do with a matrix that may not have entries?

Krylov method for Ax=b

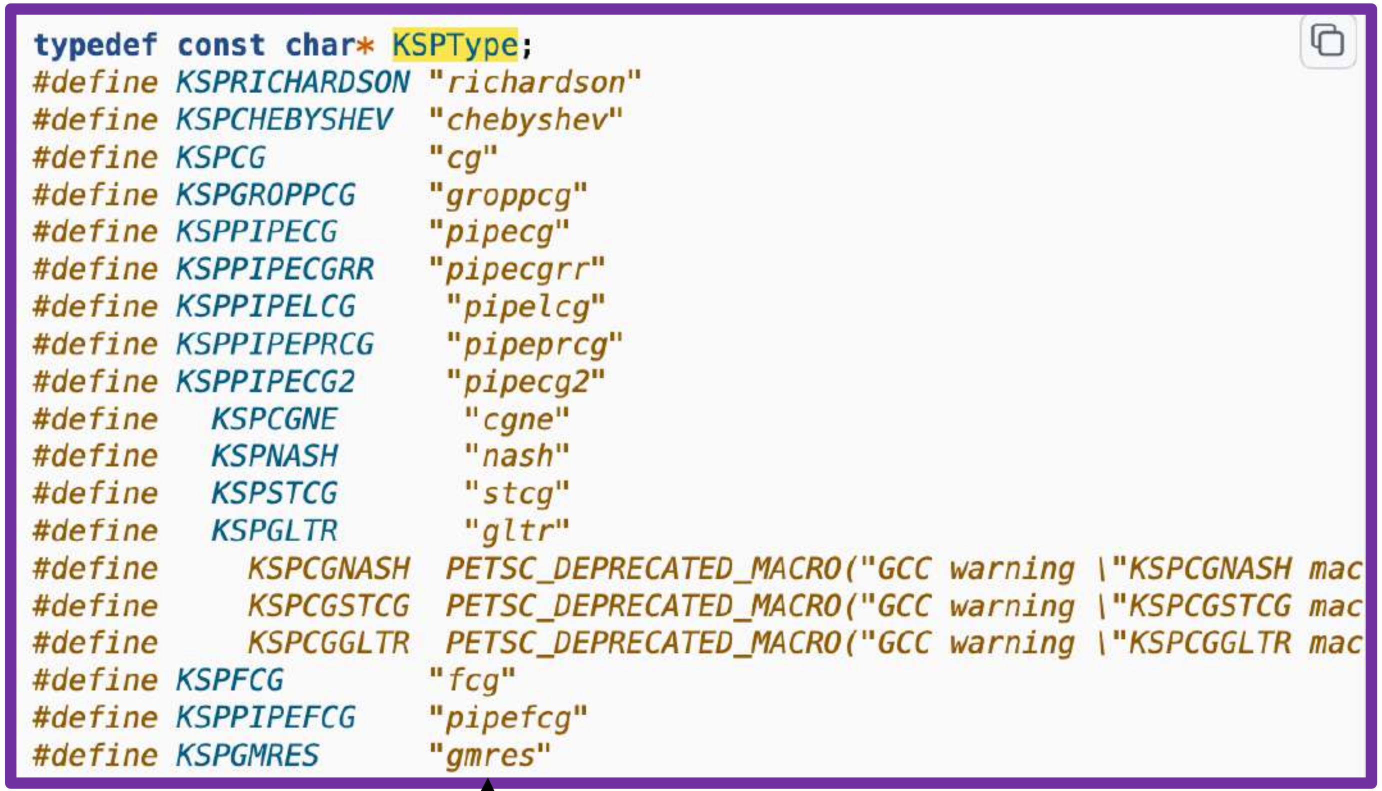
- Krylov subspace $\{b, Ab, A^2b, A^3b, \ldots\}$
- Search the one in the subspace that minimizes the residual
- Matrix-vector multiplication needs O(n) operations
- Residual is monotonically decreasing
- Restarted variants are used in practice to bound memory requirements

KSP (Krylov Space solvers) basics

```
KSP ksp;
KSPType type = KSPCG;
MPI_Comm comm = PETSC_COMM_WORLD;

KSPCreate(comm, &ksp);    // Creation
KSPSetType(ksp, type);    // Type, default is KSPGMRES
KSPSetFromOptions(v);     // Enable options

Vec b, x;
Mat A, B;
// Assembly the objects A, B, and b
KSPSetOperators(ksp, A, B);
KSPSetTolerances(ksp, rtol, atol, dtol, maxit);

KSPSolve(ksp, b, x);      // Ax = b

KSPDestroy(v);            //Dealloc
```

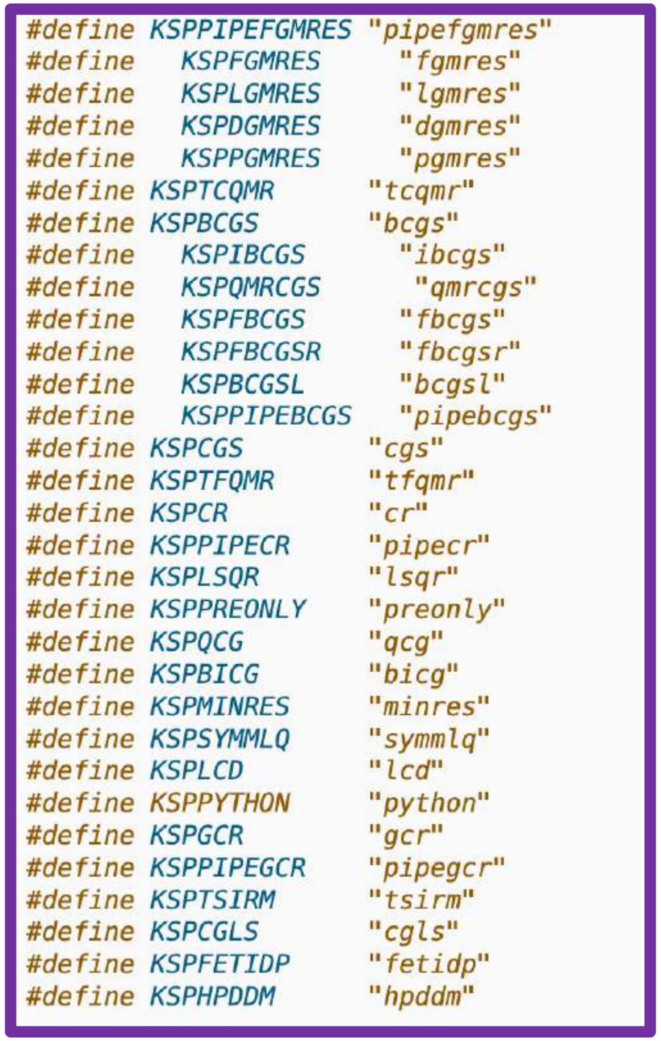
- A defines the linear system.
- A Can be an implicit matrix.
- B is used for constructing preconditioner.
- For beginners, A=B

Linear solvers in PETSc



```
typedef const char* KSPType;
#define KSPRICHARDSON "richardson"
#define KSPCHEBYSHEV  "chebyshev"
#define KSPCG         "cg"
#define KSPGROPPCG    "groppcg"
#define KSPPIPECG     "pipecg"
#define KSPPIPECGRR   "pipecgrr"
#define KSPPIPELCG     "pipelcg"
#define KSPPIPEPRCG    "pipeprcg"
#define KSPPIPECG2     "pipecg2"
#define   KSPCGNE       "cgne"
#define   KSPNASH       "nash"
#define   KSPSTCG       "stcg"
#define   KSPGLTR       "gltr"
#define     KSPCGNASH  PETSC_DEPRECATED_MACRO("GCC warning \"KSPCGNASH mac
#define     KSPCGSTCG  PETSC_DEPRECATED_MACRO("GCC warning \"KSPCGSTCG mac
#define     KSPCGGLTR  PETSC_DEPRECATED_MACRO("GCC warning \"KSPCGGLTR mac
#define KSPFCG        "fcg"
#define KSPPIPEFCG    "pipefcg"
#define KSPGMRES      "gmres"
```

Default: GMRES



```
#define KSPPIPEFGMRES "pipefgmres"
#define   KSPFGMRES     "fgmres"
#define   KSPLGMRES     "lgmres"
#define   KSPDGMRES     "dgmres"
#define   KSPPGMRES     "pgmres"
#define KSPTCQMR      "tcqmr"
#define KSPBCGS       "bcgs"
#define   KSPIBCGS      "ibcgs"
#define   KSPQMRCGS      "qmrcgs"
#define   KSPFBCGS      "fbcgs"
#define   KSPFBCGSR     "fbcgsr"
#define   KSPBCGSL      "bcgsl"
#define   KSPPIPEBCGS   "pipebcgs"
#define KSPCGS        "cgs"
#define KSPTFQMR      "tfqmr"
#define KSPCR         "cr"
#define KSPPIPECR     "pipecr"
#define KSPLSQR       "lsqr"
#define KSPPREONLY    "preonly"
#define KSPQCG        "qcg"
#define KSPBICG       "bicg"
#define KSPMINRES     "minres"
#define KSPSYMMLQ     "symmlq"
#define KSPLCD        "lcd"
#define KSPPYTHON     "python"
#define KSPGCR        "gcr"
#define KSPPIPEGCR    "pipegcr"
#define KSPTSIRM      "tsirm"
#define KSPCGLS       "cgls"
#define KSPFETIDP     "fetidp"
#define KSPHPDDM      "hpddm"
```

KSP (Krylov Space solvers) basics

```
KSP ksp;
KSPType type = KSPCG;
MPI_Comm comm = PETSC_COMM_WORLD;

KSPCreate(comm, &ksp);    // Creation
KSPSetType(ksp, type);    // Type, default is KSPGMRES
KSPSetFromOptions(ksp);   // Enable options

Vec b, x;
Mat A, B;
// Assembly the objects A, B, and b
KSPSetOperators(ksp, A, B);
KSPSetTolerances(ksp, rtol, atol, dtol, maxit);

KSPSolve(ksp, b, x);      // Ax = b

KSPDestroy(v);            //Dealloc
```

- rtol : relative tolerance
  stop if residual < rtol * norm(b)
- atol : absolute tolerance
  stop if residual < atol
- dtol : divergence tolerance
  stop if residual > dtol * norm(b)
- maxit : maximum number of iterations

control them at run time:
-ksp_rtol 1.0e-6 –ksp_atol 1.0e-50
-ksp_dtol 1.0e5 –ksp_max_it 2000

Preconditioners

Idea: improve the conditioning of the Krylov operator

Left preconditioning

$$(P^{-1}A)x = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}A)P^{-1}b, (P^{-1}A)^2P^{-1}b, \dots\}$$

Right preconditioning

$$(AP^{-1})Px = b$$

$$\{b, (P^{-1}A)b, (P^{-1}A)^2b, \dots\}$$

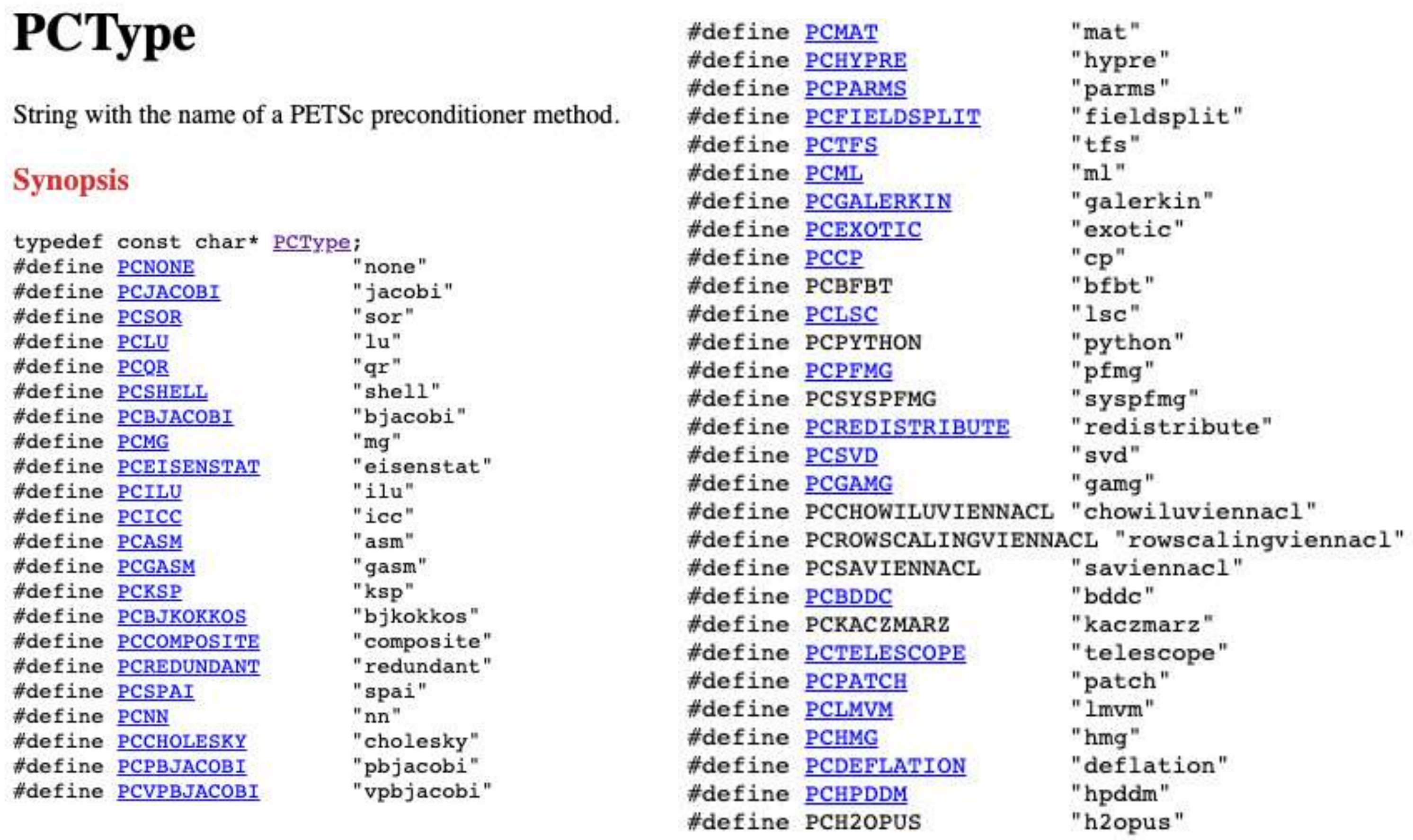
- The matrix B is utilized to construct P^{-1}
- There are many built-in and interfaced preconditioners: ILU, Jacobi, ASM, AMG, … …
- Can be composed in additive or multiplicative way (PCCOMPOSITE)

Preconditioners

```
KSP ksp;
PC pc;
PCType pctype=PCILU;
…
KSPGetPC(ksp, &pc);
PCSetType(pc, pctype);
```

Check the manual page for all PC types and their usage.

Try them from command line directly.



PCType

String with the name of a PETSc preconditioner method.

Synopsis

```
typedef const char* PCType;
#define PCNONE            "none"
#define PCJACOBI          "jacobi"
#define PCSOR             "sor"
#define PCLU              "lu"
#define PCQR              "qr"
#define PCSHELL           "shell"
#define PCBJACOBI         "bjacobi"
#define PCMG              "mg"
#define PCEISENSTAT       "eisenstat"
#define PCILU             "ilu"
#define PCICC             "icc"
#define PCASM             "asm"
#define PCGASM            "gasm"
#define PCKSP             "ksp"
#define PCBJKOKKOS        "bjkokkos"
#define PCCOMPOSITE       "composite"
#define PCREDUNDANT       "redundant"
#define PCSPAI            "spai"
#define PCNN              "nn"
#define PCCHOLESKY        "cholesky"
#define PCPBJACOBI        "pbjacobi"
#define PCVPBJACOBI       "vpbjacobi"
#define PCMAT               "mat"
#define PCHYPRE             "hypre"
#define PCPARMS             "parms"
#define PCFIELDSPLIT        "fieldsplit"
#define PCTFS               "tfs"
#define PCML                "ml"
#define PCGALERKIN          "galerkin"
#define PCEXOTIC            "exotic"
#define PCCP                "cp"
#define PCBFBT              "bfbt"
#define PCLSC               "lsc"
#define PCPYTHON            "python"
#define PCPFMG              "pfmg"
#define PCSYSPFMG           "syspfmg"
#define PCREDISTRIBUTE      "redistribute"
#define PCSVD               "svd"
#define PCGAMG              "gamg"
#define PCCHOWILUVIENNACL "chowiluviennacl"
#define PCROWSCALINGVIENNACL "rowscalingviennacl"
#define PCSAVIENNACL        "saviennacl"
#define PCBDDC              "bddc"
#define PCKACZMARZ          "kaczmarz"
#define PCTELESCOPE         "telescope"
#define PCPATCH             "patch"
#define PCLMVM              "lmvm"
#define PCHMG               "hmg"
#define PCDEFLATION         "deflation"
#define PCHPDDM             "hpddm"
#define PCH2OPUS            "h2opus"
```

Options prefixes

- Sometimes, there can be multiple instances of the same class that we want to control separately.

  ```
  KSP ksp1, ksp2, ksp3;
  …
  KSPSetOptionsPrefix(ksp2, “ksp2_”);
  ```

- Command line arguments:

  ```
  -ksp_rtol 1.0e-8 # sets relative tolerance for all unprefixed (ksp1 & ksp3)
  -ksp2_ksp_rtol 1.0e-4 # sets relative tolerance for ksp2
  ```

- It is easy to try with many different solver types:

  ```
  -ksp2_ksp_type fgmres –ksp2_pc_type bjacobi –ksp2_ksp_rtol 1.0e-3
  -ksp2_sub_ksp_type richardson –ksp2_sub_pc_type icc
  ```

Options prefixes

- It is easy to try with many different solver types:

```
-ksp2_ksp_type fgmres –ksp2_pc_type bjacobi –ksp2_ksp_rtol 1.0e-3
-ksp2_sub_ksp_type richardson –ksp2_sub_pc_type icc
```

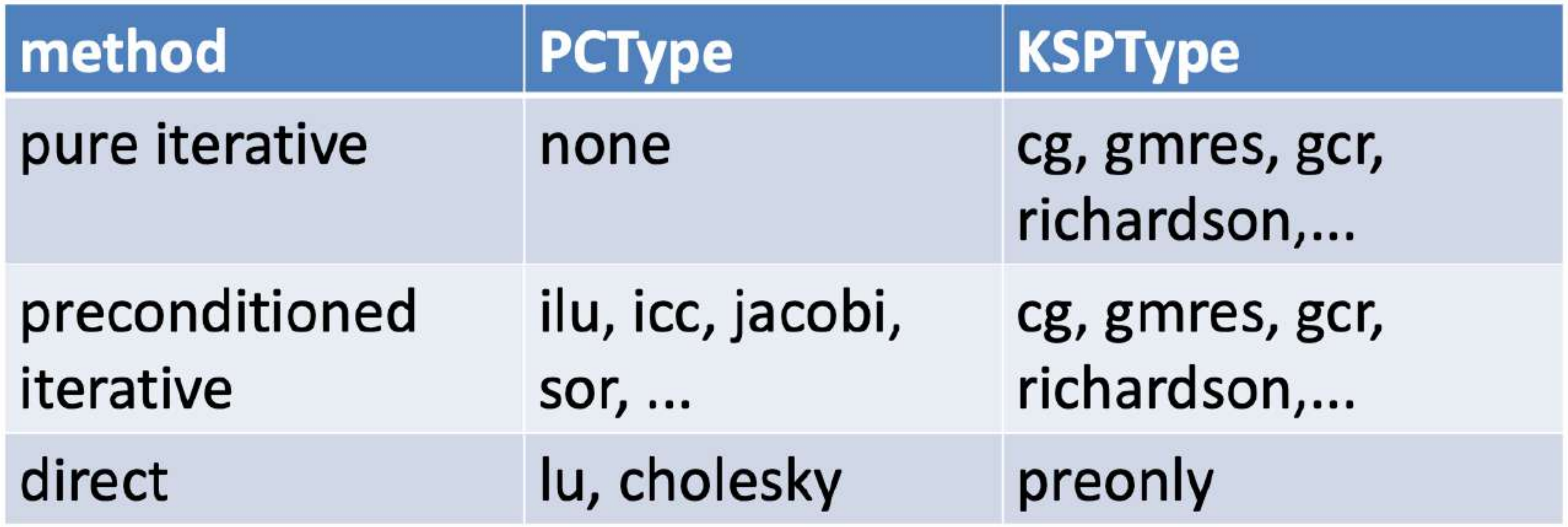
- There can be built-in prefixes related composed solver/preconditioners (refer to the manual)

- sub_ above points to PCBJACOBI blockwise KSP/PC

Direct solvers

- Direct solvers = special case of preconditioned iterative solvers
  - just one iteration with application of “greatest preconditioner”, i.e. full LU factorization.

```
KSPSetType(ksp, KSPPREONLY); // preonly means apply preconditioner only
PCSetType(pc, PCLU);         // Or PCCHOLESKY if symmetric
```



method	PCType	KSPType
pure iterative	none	cg, gmres, gcr, richardson,...
preconditioned iterative	ilu, icc, jacobi, sor, ...	cg, gmres, gcr, richardson,...
direct	lu, cholesky	preonly

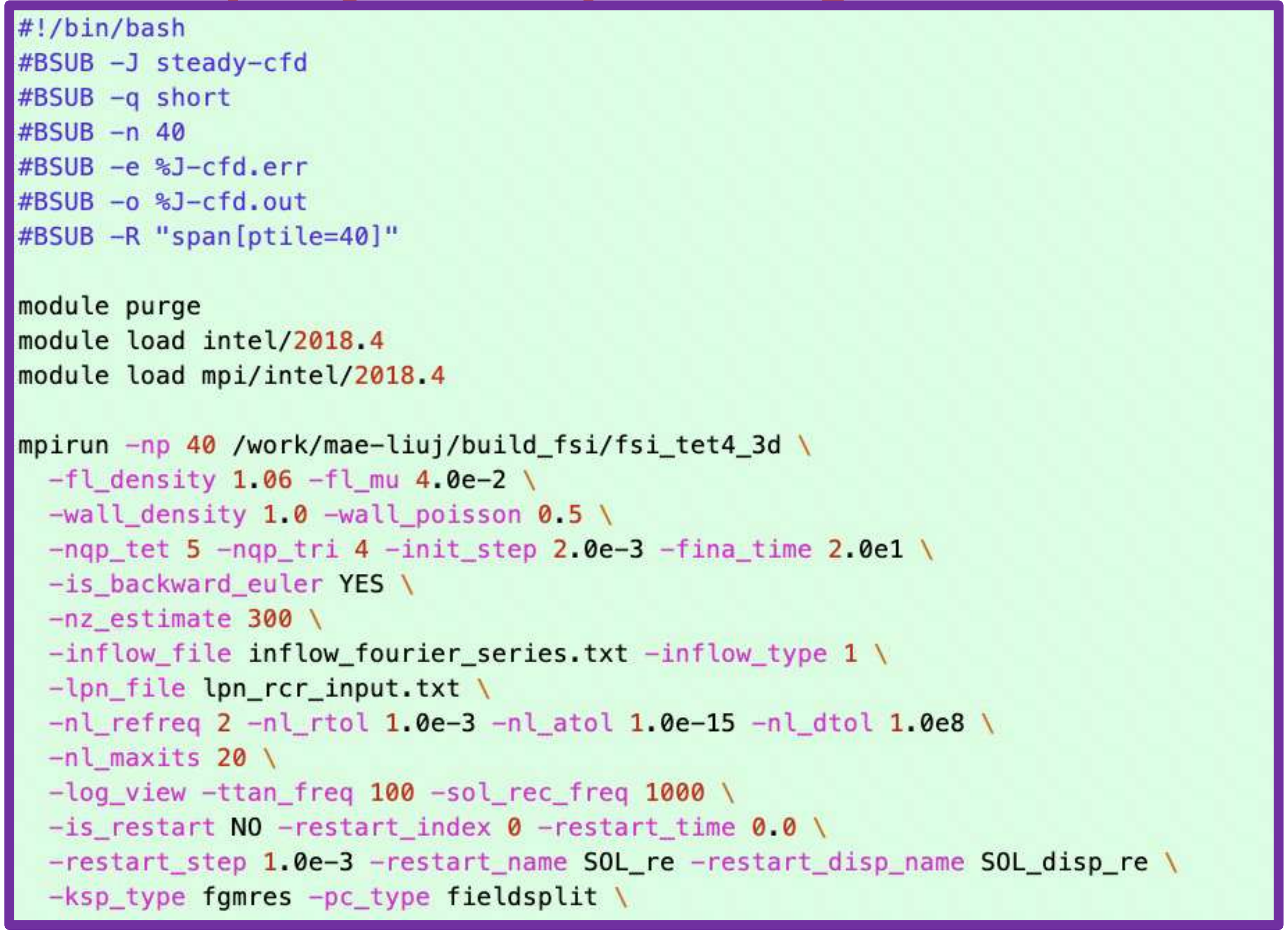
Direct solvers

- Direct solvers = special case of preconditioned iterative solvers
  - just one iteration with application of “greatest preconditioner”, i.e. full LU factorization.

```
KSPSetType(ksp, KSPPREONLY); // preonly means apply preconditioner only
PCSetType(pc, PCLU);         // Or PCCHOLESKY if symmetric
```

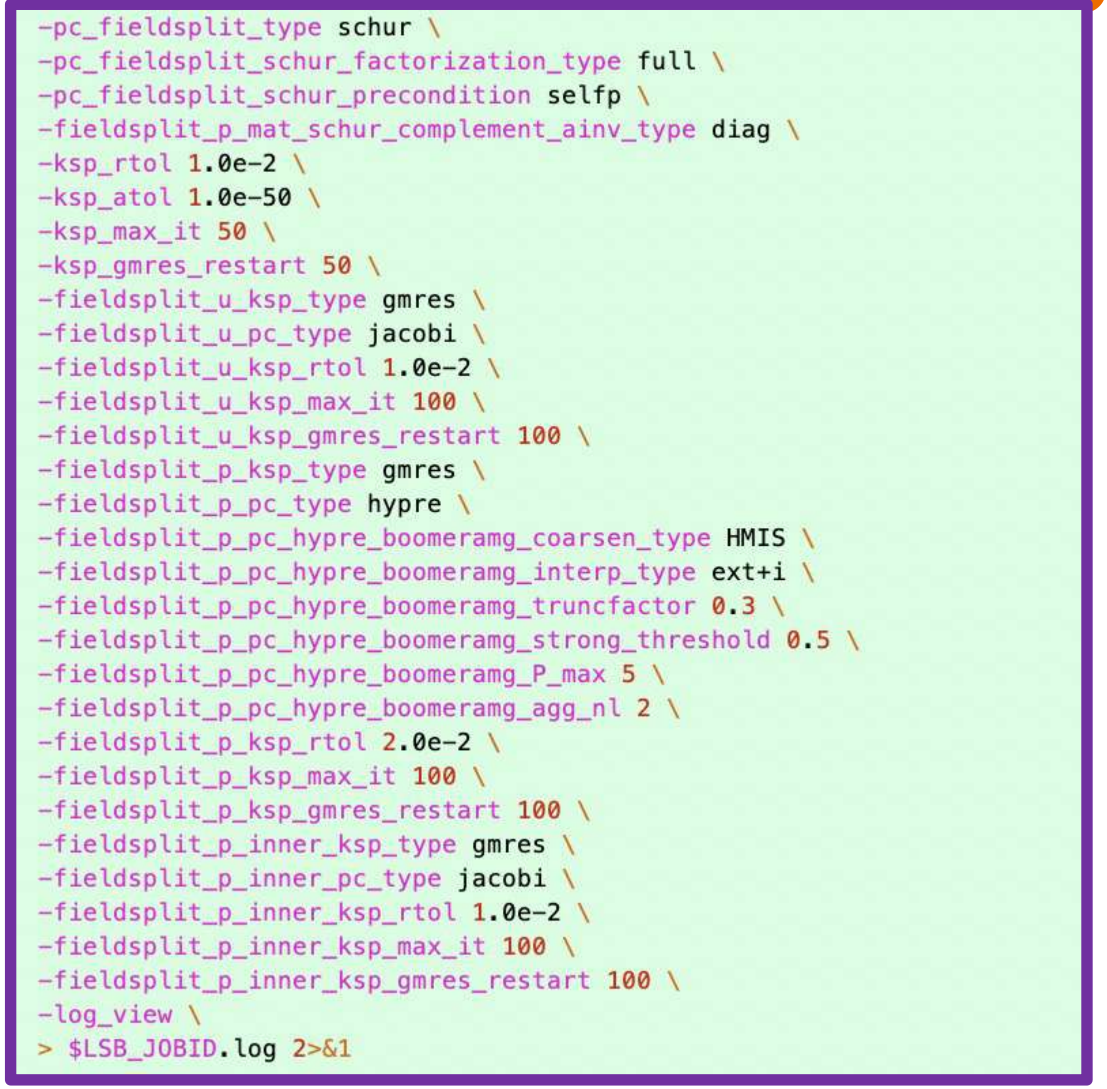
- PETSc built-in factorization routines are not very efficient, there are interfaces to several external implementations of parallel LU (MUMPS, SuperLU, PaStiX, …)
- For the current list, search MATSOLVER* at https://petsc.org/release/docs/manualpages/Mat/index.html
- You can specify the external solver in code by PCFactorSetMatSolverPackage(pc, MATSOLVERMUMPS) or directly from command line: -pc_factor_mat_solver_package mumps

Sample jobscript using PETSc



```
#!/bin/bash
#BSUB -J steady-cfd
#BSUB -q short
#BSUB -n 40
#BSUB -e %J-cfd.err
#BSUB -o %J-cfd.out
#BSUB -R "span[ptile=40]"

module purge
module load intel/2018.4
module load mpi/intel/2018.4

mpirun -np 40 /work/mae-liuj/build_fsi/fsi_tet4_3d \
  -fl_density 1.06 -fl_mu 4.0e-2 \
  -wall_density 1.0 -wall_poisson 0.5 \
  -nqp_tet 5 -nqp_tri 4 -init_step 2.0e-3 -fina_time 2.0e1 \
  -is_backward_euler YES \
  -nz_estimate 300 \
  -inflow_file inflow_fourier_series.txt -inflow_type 1 \
  -lpn_file lpn_rcr_input.txt \
  -nl_refreq 2 -nl_rtol 1.0e-3 -nl_atol 1.0e-15 -nl_dtol 1.0e8 \
  -nl_maxits 20 \
  -log_view -ttan_freq 100 -sol_rec_freq 1000 \
  -is_restart NO -restart_index 0 -restart_time 0.0 \
  -restart_step 1.0e-3 -restart_name SOL_re -restart_disp_name SOL_disp_re \
  -ksp_type fgmres -pc_type fieldsplit \
```

Sample jobscript using PETSc (cont.)



```
-pc_fieldsplit_type schur \
-pc_fieldsplit_schur_factorization_type full \
-pc_fieldsplit_schur_precondition selfp \
-fieldsplit_p_mat_schur_complement_ainv_type diag \
-ksp_rtol 1.0e-2 \
-ksp_atol 1.0e-50 \
-ksp_max_it 50 \
-ksp_gmres_restart 50 \
-fieldsplit_u_ksp_type gmres \
-fieldsplit_u_pc_type jacobi \
-fieldsplit_u_ksp_rtol 1.0e-2 \
-fieldsplit_u_ksp_max_it 100 \
-fieldsplit_u_ksp_gmres_restart 100 \
-fieldsplit_p_ksp_type gmres \
-fieldsplit_p_pc_type hypre \
-fieldsplit_p_pc_hypre_boomeramg_coarsen_type HMIS \
-fieldsplit_p_pc_hypre_boomeramg_interp_type ext+i \
-fieldsplit_p_pc_hypre_boomeramg_truncfactor 0.3 \
-fieldsplit_p_pc_hypre_boomeramg_strong_threshold 0.5 \
-fieldsplit_p_pc_hypre_boomeramg_P_max 5 \
-fieldsplit_p_pc_hypre_boomeramg_agg_nl 2 \
-fieldsplit_p_ksp_rtol 2.0e-2 \
-fieldsplit_p_ksp_max_it 100 \
-fieldsplit_p_ksp_gmres_restart 100 \
-fieldsplit_p_inner_ksp_type gmres \
-fieldsplit_p_inner_pc_type jacobi \
-fieldsplit_p_inner_ksp_rtol 1.0e-2 \
-fieldsplit_p_inner_ksp_max_it 100 \
-fieldsplit_p_inner_ksp_gmres_restart 100 \
-log_view \
> $LSB_JOBID.log 2>&1
```

Debugging and profiling

Interaction with debugger

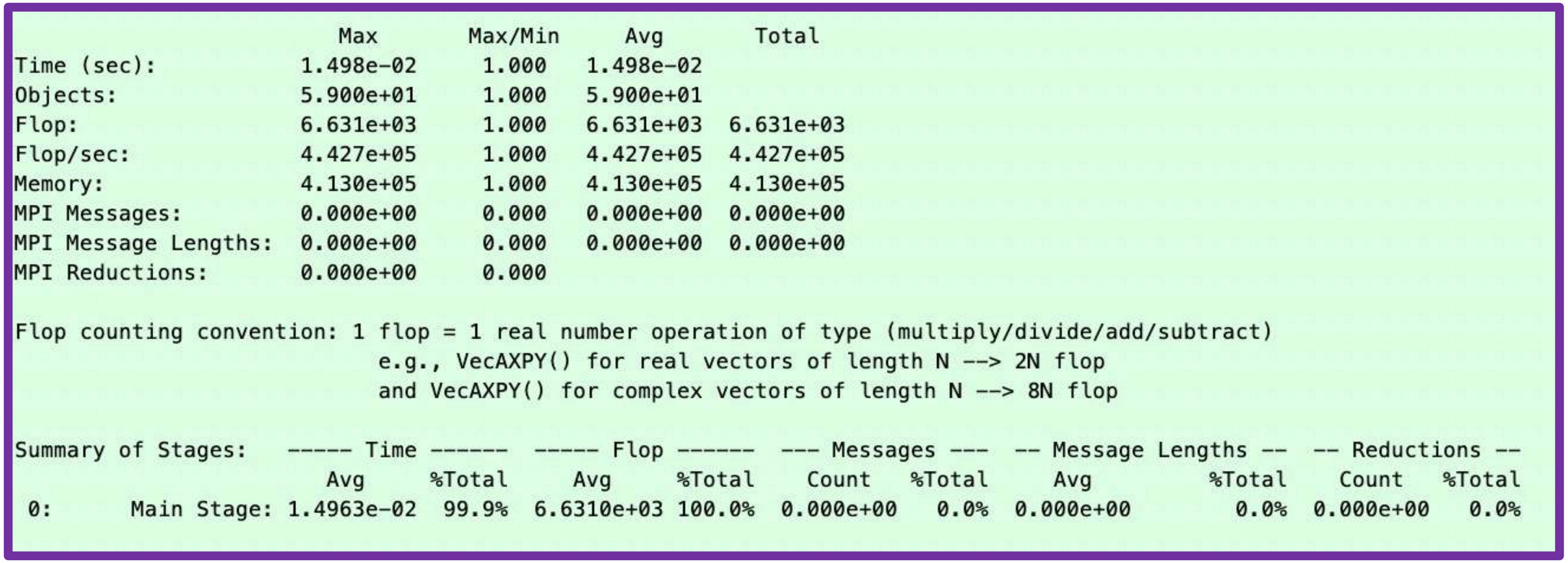
- Launch the debugger
  - ➢ -start_in_debugger [gdb, dbx, lldb, …]
  - ➢ -on_error_attach_debugger [gdb, dbx, lldb, …]

- Attach the debugger only to certain parallel processes
  - ➢ -debugger_nodes 0, 1

- Do not forget valgrind, which is the “best tool” (PETSc developer said that).
  - need –trace-children=yes when running MPI

Code profiling

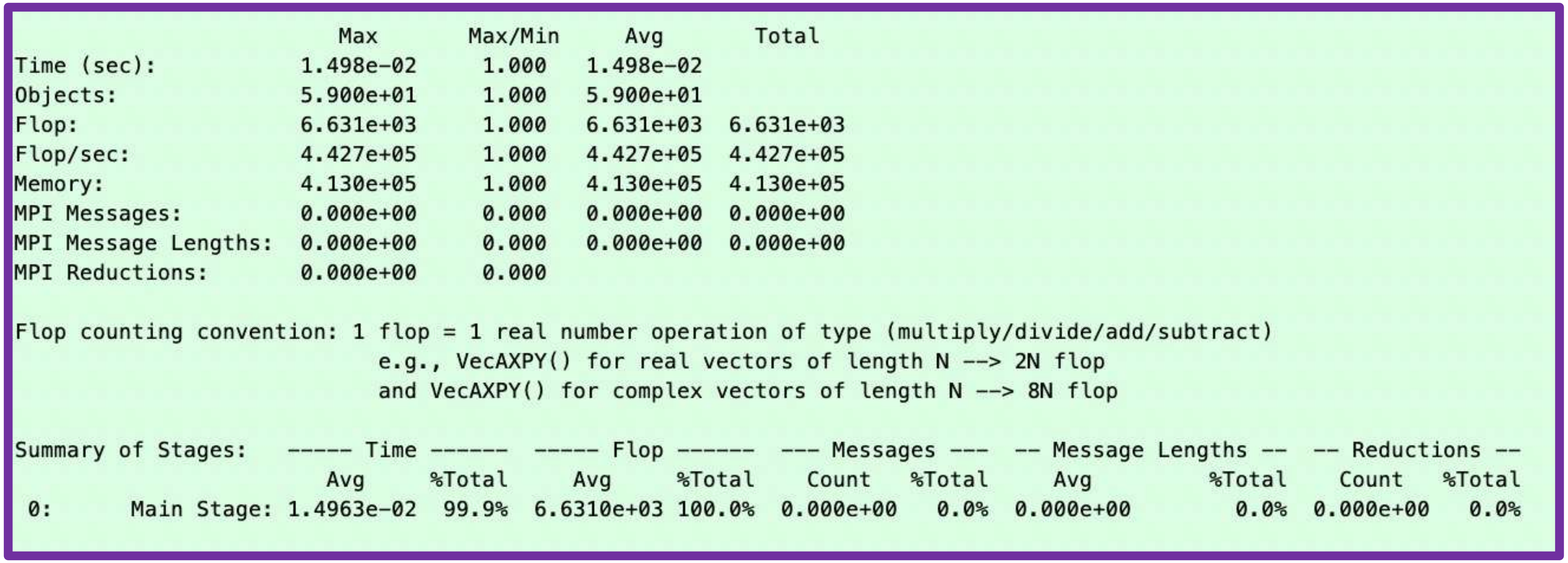
- Use –log_view to get a performance profile
  - Event timing
  - Event flops
  - Memory usage
  - MPI messages



```
                         Max       Max/Min     Avg       Total
Time (sec):           1.498e-02     1.000   1.498e-02
Objects:              5.900e+01     1.000   5.900e+01
Flop:                 6.631e+03     1.000   6.631e+03  6.631e+03
Flop/sec:             4.427e+05     1.000   4.427e+05  4.427e+05
Memory:               4.130e+05     1.000   4.130e+05  4.130e+05
MPI Messages:         0.000e+00     0.000   0.000e+00  0.000e+00
MPI Message Lengths:  0.000e+00     0.000   0.000e+00  0.000e+00
MPI Reductions:       0.000e+00     0.000

Flop counting convention: 1 flop = 1 real number operation of type (multiply/divide/add/subtract)
                            e.g., VecAXPY() for real vectors of length N --> 2N flop
                            and VecAXPY() for complex vectors of length N --> 8N flop

Summary of Stages:   ----- Time ------  ----- Flop ------  --- Messages ---  -- Message Lengths --  -- Reductions --
                        Avg     %Total     Avg     %Total    Count   %Total     Avg         %Total    Count   %Total
 0:      Main Stage: 1.4963e-02  99.9%  6.6310e+03 100.0%  0.000e+00   0.0%  0.000e+00        0.0%  0.000e+00   0.0%
```

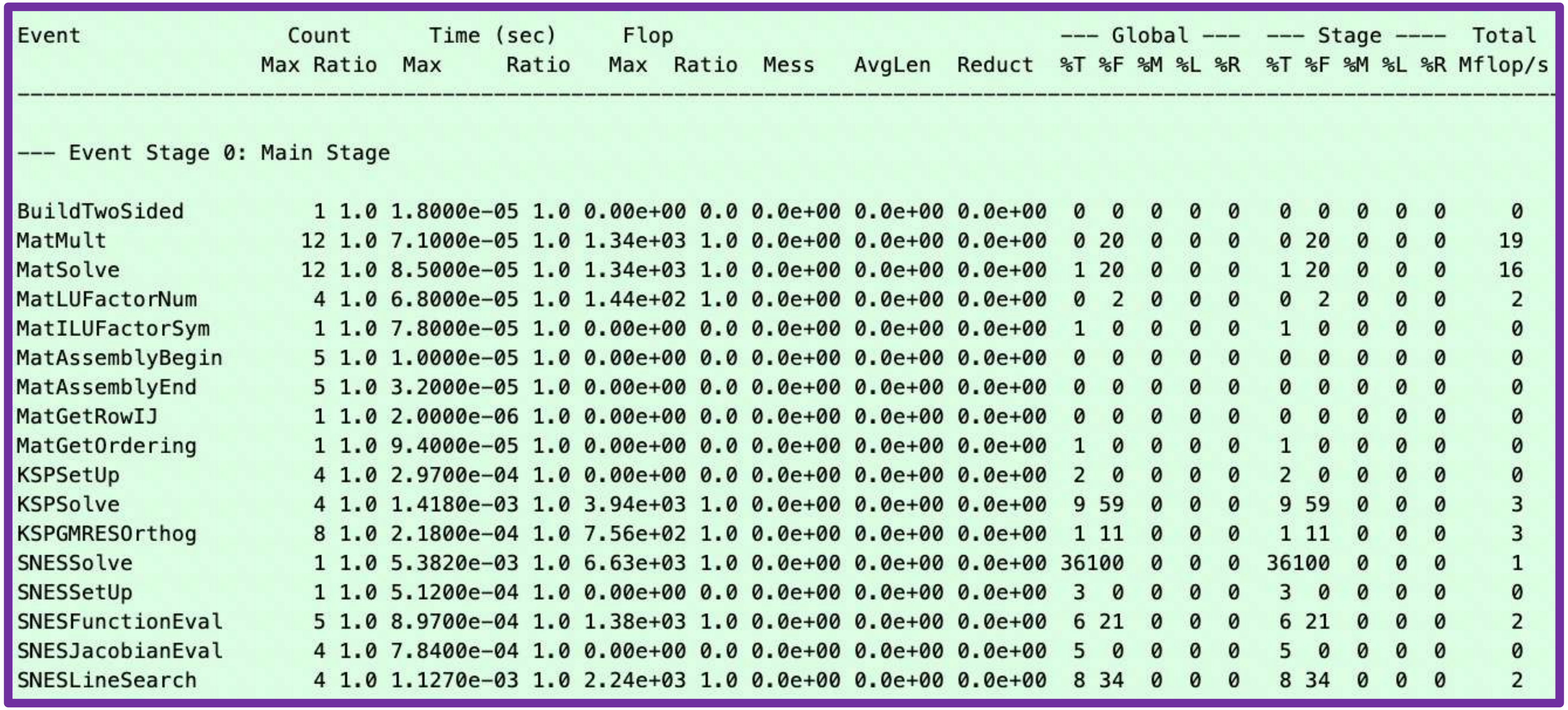
Code profiling

- Use –log_view to get a performance profile
- Call PetscLogStagePush() and PetscLogStagePop() to add new stages
- Call PetscLogEventBegin() and PetscLogEventEnd() to add new events



```
                         Max       Max/Min     Avg       Total
Time (sec):           1.498e-02     1.000   1.498e-02
Objects:              5.900e+01     1.000   5.900e+01
Flop:                 6.631e+03     1.000   6.631e+03  6.631e+03
Flop/sec:             4.427e+05     1.000   4.427e+05  4.427e+05
Memory:               4.130e+05     1.000   4.130e+05  4.130e+05
MPI Messages:         0.000e+00     0.000   0.000e+00  0.000e+00
MPI Message Lengths:  0.000e+00     0.000   0.000e+00  0.000e+00
MPI Reductions:       0.000e+00     0.000

Flop counting convention: 1 flop = 1 real number operation of type (multiply/divide/add/subtract)
                            e.g., VecAXPY() for real vectors of length N --> 2N flop
                            and VecAXPY() for complex vectors of length N --> 8N flop

Summary of Stages:   ----- Time ------  ----- Flop ------  --- Messages ---  -- Message Lengths --  -- Reductions --
                        Avg     %Total     Avg     %Total    Count   %Total     Avg         %Total    Count   %Total
 0:      Main Stage: 1.4963e-02  99.9%  6.6310e+03 100.0%  0.000e+00   0.0%  0.000e+00        0.0%  0.000e+00   0.0%
```

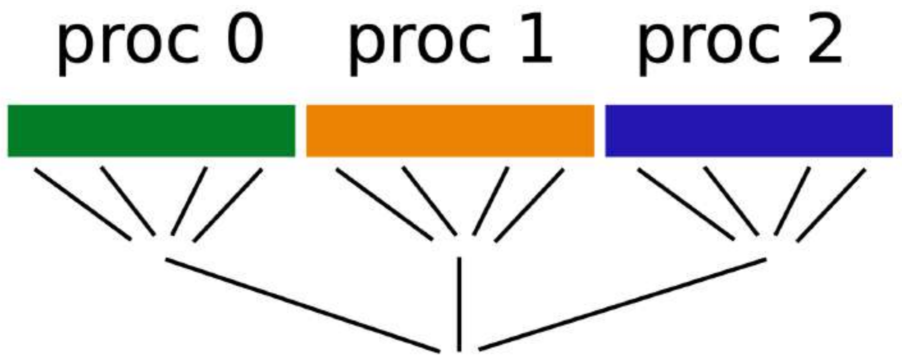
Code profiling



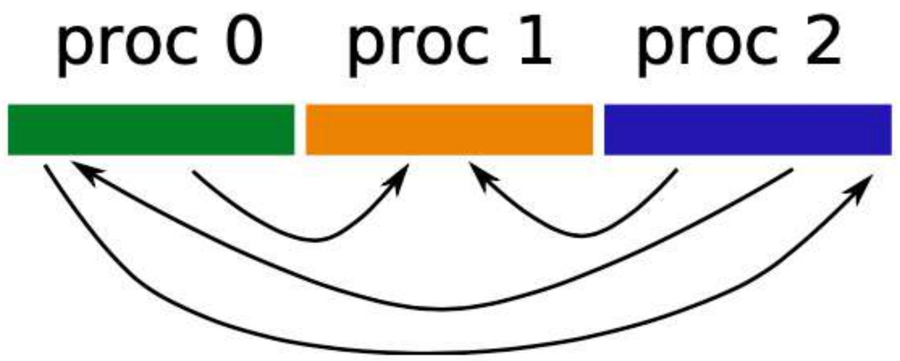
```
Event                Count      Time (sec)     Flop                              --- Global ---  --- Stage ----  Total
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   AvgLen  Reduct  %T %F %M %L %R  %T %F %M %L %R Mflop/s
------------------------------------------------------------------------------------------------------------------------------


--- Event Stage 0: Main Stage

BuildTwoSided          1 1.0 1.8000e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatMult               12 1.0 7.1000e-05 1.0 1.34e+03 1.0 0.0e+00 0.0e+00 0.0e+00  0 20  0  0  0   0 20  0  0  0    19
MatSolve              12 1.0 8.5000e-05 1.0 1.34e+03 1.0 0.0e+00 0.0e+00 0.0e+00  1 20  0  0  0   1 20  0  0  0    16
MatLUFactorNum         4 1.0 6.8000e-05 1.0 1.44e+02 1.0 0.0e+00 0.0e+00 0.0e+00  0  2  0  0  0   0  2  0  0  0     2
MatILUFactorSym        1 1.0 7.8000e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  1  0  0  0  0   1  0  0  0  0     0
MatAssemblyBegin       5 1.0 1.0000e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatAssemblyEnd         5 1.0 3.2000e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatGetRowIJ            1 1.0 2.0000e-06 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatGetOrdering         1 1.0 9.4000e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  1  0  0  0  0   1  0  0  0  0     0
KSPSetUp               4 1.0 2.9700e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  2  0  0  0  0   2  0  0  0  0     0
KSPSolve               4 1.0 1.4180e-03 1.0 3.94e+03 1.0 0.0e+00 0.0e+00 0.0e+00  9 59  0  0  0   9 59  0  0  0     3
KSPGMRESOrthog         8 1.0 2.1800e-04 1.0 7.56e+02 1.0 0.0e+00 0.0e+00 0.0e+00  1 11  0  0  0   1 11  0  0  0     3
SNESSolve              1 1.0 5.3820e-03 1.0 6.63e+03 1.0 0.0e+00 0.0e+00 0.0e+00 36100  0  0  0  36100  0  0  0     1
SNESSetUp              1 1.0 5.1200e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  3  0  0  0  0   3  0  0  0  0     0
SNESFunctionEval       5 1.0 8.9700e-04 1.0 1.38e+03 1.0 0.0e+00 0.0e+00 0.0e+00  6 21  0  0  0   6 21  0  0  0     2
SNESJacobianEval       4 1.0 7.8400e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  5  0  0  0  0   5  0  0  0  0     0
SNESLineSearch         4 1.0 1.1270e-03 1.0 2.24e+03 1.0 0.0e+00 0.0e+00 0.0e+00  8 34  0  0  0   8 34  0  0  0     2
```

Communication costs

- Reductions: usually part of Krylov method
  - VecDot(), VecNorm(),
  - MatAssemblyBegin/End()



- Point-to-point communication
  - MatMult
  - PCApply
  - VecScatter



Summary

- PETSc can help you
  - easily construct a code to test your ideas without worrying about how to parallelize the code
  - scale your code to large distributed machines
  - test your code with different algorithms
  - tune your code easily

- Documentation are available online http://www.mcs.anl.gov/petsc/docs
  - PETSc users manual
  - Hyperlinked examples
  - FAQ

- Furture features:
  - Support of 1D/2D/3D structured and unstructured grids
  - Nonlinear solver, time solver, eigenvalue solver (HW 6)
  - Load balancing and grid partitioning, GPU support

Summary

- PETSc can help you
  - easily construct a code to test your ideas without worrying about how to parallelize the code
  - scale your code to large distributed machines
  - test your code with different algorithms
  - tune your code easily

- Documentation are available online http://www.mcs.anl.gov/petsc/docs
  - PETSc users manual
  - Hyperlinked examples
  - FAQ

- References:

  Using MPI, by Gropp, Lusk, and Skjellum
  Domain Decomposition, by Smith, Bjorstad and Gropp
  PETSc for Partial Differential Equations, by E. Bueler