A vector in PETSc is an object of type Vec.

Two basic types: sequential and parallel (MPI based)

In parallel, the vector is distributed over all processes: each process (i.e., MPI rank) stores its part of the vector.
A vector in PETSc is an object of type Vec.

Two basic types: sequential and parallel (MPI based)

In parallel, the vector is distributed over all processes: each process (i.e., MPI rank) stores its part of the vector.

### PETSc Vectors: Types & Create

- **VecCreateSeq**(PETSC_COMM_SELF, int m, Vec* x);

- **VecCreateMPI**(MPI_Comm comm, int m, int M, Vec* x);

where

- m = local size, or PETSC_DECIDE if M given
- M = global size, or PETSC_DETERMINE if m given for all ranks
VecCreateSeq(PETSC_COMM_SELF, int m, Vec* x);

VecCreateMPI(MPI_Comm comm, int m, int M, Vec* x);

Other way:

VecCreate(MPI_COMM comm, Vec* x);
VecSetType(Vec x, VECSEQ/VECMPI);
VecSetSizes(Vec x, int m, int M);
VecCreateSeq(PETSC_COMM_SELF, int m, Vec* x);
VecCreateMPI(MPI_Comm comm, int m, int M, Vec* x);

Other way:
VecCreate(MPI_COMM comm, Vec* x);
VecSetType(Vec x, VECSEQ/VECMPI);
VecSetSizes(Vec x, int m, int M);

Yet another way:
VecCreate(MPI_COMM comm, Vec* x);
VecSetSizes(Vec x, int m, int M);
VecSetFromOptions(Vec x);

and use for instance -vec_type mpi at runtime.
VecCreateSeq(PETSC_COMM_SELF, int m, Vec* x);

VecCreateMPI(MPI_Comm comm, int m, int M, Vec* x);

In fortran:

call VecCreateSeq(PETSC_COMM_SELF,integer m,Vec x,
PetscErrorCode ierr)

call VecCreateMPI(MPI_Comm comm,integer m,integer M,Vec x,
PetscErrorCode ierr)
VecSet(Vec x, PetscScalar value);
PETSc Vectors: Set

VecSet(Vec x, PetscScalar value);

VecSetValue(Vec x, int row, PetscScalar value, INSERT VALUES or ADD VALUES);
PETSc Vectors : Set

VecSet(Vec x, PetscScalar value);

VecSetValue(Vec x, int row, PetscScalar value, INSERT_VALUES or ADD_VALUES);

VecSetValues(Vec x, int n, int* indices, PetscScalar* values, INSERT_VALUES or ADD_VALUES);

In fortran:

call VecSetValues(Vec x, integer n, integer(n) indices, PetscScalar(n) values, INSERT_VALUES or ADD_VALUES, PetscErrorCode ierr)
Notes:

- VecSetValues faster than VecSetValue. VecSetValues fastest if n large.
Notes:

- VecSetValues faster than VecSetValue. VecSetValues fastest if \( n \) large.

- *Global* indices have to be used in VecSetValue and VecSetValues. To use *local* indices:
  
  VecSetValueLocal and VecSetValuesLocal.
PETSc Vectors: Set (cont’d)

Notes:

- VecSetValues faster than VecSetValue. VecSetValues fastest if \( n \) large.

- *Global* indices have to be used in VecSetValue and VecSetValues. To use *local* indices:
  
  VecSetValueLocal and VecSetValuesLocal.

- Always 0-based indices in C and *fortran*. 
After setting values, one must assemble the vector:

```c
VecAssemblyBegin(Vec x);
VecAssemblyEnd(Vec x);
```

Note: allow overlap of communication and calculation.
After setting values, one must assemble the vector:

VecAssemblyBegin(Vec x);
VecAssemblyEnd(Vec x);

Note: allow overlap of communication and calculation.

Caution: INSERT_VALUES and ADD_VALUES cannot be mixed (call assembly routines inbetween).
VecView(Vec x,PETSC_VIEWER_STDOUT_WORLD);

In fortran:
call VecView(Vec x,PETSC_VIEWER_STDOUT_WORLD,
              PetscErrorCode ierr)

PETSC_VIEWER_STDOUT_WORLD ≡ synchronized standard output:
    All processors send their data to the first processor to print.

Other visualization contexts: see on-line documentation.
PETSc Vectors: Operations

VecScale \hspace{1cm} x = a \times x,
VecAXPY \hspace{1cm} y = a \times x + y,
VecDot \hspace{1cm} x \cdot y,
VecPointwiseMult \hspace{1cm} w_i = x_i \times y_i
VecNorm \hspace{1cm} ||A||...
PETSc Vectors: Operations

- VecScale: $x = a \times x$,
- VecAXPY: $y = a \times x + y$,
- VecDot: $x \cdot y$,
- VecPointwiseMult: $w_i = x_i \times y_i$,
- VecNorm: $\|A\|$...
  
  VecDuplicate: $y$ created with same type as $x$; storage allocated for $y$ but values not copied.
  VecCopy: $y \leftarrow x$ (y pre-existing),
VecScale \quad x = a \times x,
VecAXPY \quad y = a \times x + y,
VecDot \quad x \cdot y,
VecPointwiseMult \quad w_i = x_i \times y_i
VecNorm \quad \| A \|...
PETSc Vectors: Get

One can pull only local values from a vector.

- Single Value → VecGetValues (use global numbering)
PETSc Vectors : Get

One can pull only local values from a vector.

- Single Value → VecGetValues (use global numbering)
- All local elements → VecGetArray (no copy made, time-efficient):

```c
VecGetArray(Vec v, PetscScalar **array);
...
VecRestoreArray(Vec v, PetscScalar **array);
```

In fortran:
call VecGetArray(Vec v, PetscScalar vv(1),
PetscOffset offset, PetscErrorCode ierr)
... vv(offset + i) ...
call VecRestoreArray(...)

In fortran90:
call VecGetArrayF90(Vec v, PetscScalar pointer vv,
PetscErrorCode ierr)
... vv(i)
call VecRestoreArrayF90(...)

Serge Van Criekingen
Introduction to PETSc: Vectors
One can pull **only local values** from a vector.

- Single Value → `VecGetValues` (use global numbering)
- All local elements → `VecGetArray` (no copy made, time-efficient):

```c
VecGetArray(Vec v, PetscScalar **array);
...
VecRestoreArray(Vec v, PetscScalar **array);
```

In fortran:

```fortran
call VecGetArray(Vec v, PetscScalar vv(1),
                 PetscOffset offset, PetscErrorCode ierr)
... vv(offset + i) ...
```

```fortran
call VecRestoreArray(...) 
```
PETSc Vectors: Get

One can pull only local values from a vector.

- Single Value → VecGetValues (use global numbering)
- All local elements → VecGetArray (no copy made, time-efficient):

```c
VecGetArray(Vec v, PetscScalar **array);
...
VecRestoreArray(Vec v, PetscScalar **array);
```

In fortran:

```fortran
call VecGetArray(Vec v, PetscScalar vv(1),
                 PetscOffset offset, PetscErrorCode ierr)
... vv(offset + i) ...
call VecRestoreArray(...)
```

In fortran90:

```fortran
call VecGetArrayF90(Vec v, PetscScalar pointer vv,
                     PetscErrorCode ierr)
... call VecRestoreArrayF90(...)
```
• To obtain the range of indices owned by each processor:

```c
VecGetOwnershipRange(Vec x, int* istart, int* iend);
```

In fortran:

```fortran
call VecGetOwnershipRange(Vec x, integer istart,
                           integer iend,
                           PetscErrorCode ierr)
```
Index Set ⊆ generalization of a set of integer indices.
Type: IS
Index Set \equiv generalization of a set of integer indices.

Type : IS

Two ways to create :

\begin{verbatim}
ISCreateGeneral(MPI_Comm comm, int n, int idx[],
                PETSC_COPY_VALUES, IS* is);
\end{verbatim}

\begin{verbatim}
ISCreateStride(MPI_Comm comm, int n, int first, int step,
               IS* is);
\end{verbatim}
PETSc Vectors: Index Set

Index Set ≡ generalization of a set of integer indices.

Type: IS

Two ways to create:

```c
ISCreateGeneral(MPI_Comm comm, int n, int idx[],
                PETSC_COPY_VALUES, IS* is);
```

```c
ISCreateStride(MPI_Comm comm, int n, int first, int step,
               IS* is);
```

To visualize:

```c
ISView(IS is, PETSC_VIEWER_STDOUT_SELF
       or PETSC_VIEWER_STDOUT_WORLD);
```
PETSc Vectors: Scatters and Gathers

The `VecScatter` type describes a context for both scatters and gathers.

To exchange data between the indices `isX` of `vecX` and the indices `isY` of `vecY` (length of `isX` = length of `isY`):

```
VecScatterCreate(Vec vecX, IS isX, Vec vecY, IS isY,
                VecScatter* the_context);
```

Notes:
- Conventional scatter if `isX` = stride with step 1.
- Conventional gather if `isY` = stride with step 1.
- `ADD VALUES`, `SCATTER BACKWARD` also available.
The **VecScatter** type describes a context for both scatters and gathers.

To exchange data between the indices $\text{isX}$ of $\text{vecX}$ and the indices $\text{isY}$ of $\text{vecY}$ (length of $\text{isX} = \text{length of } \text{isY}$):

```c
VecScatterCreate(Vec vecX, IS isX, Vec vecY, IS isY, VecScatter* the_context);
```

⇒ to copy elements from $X$ to $Y$:

```c
VecScatterBegin(the_context, vecX, vecY, INSERT_VALUES, SCATTER_FORWARD);
VecScatterEnd(the_context, vecX, vecY, INSERT_VALUES, SCATTER_FORWARD);
```
PETSc Vectors: Scatters and Gathers

The VecScatter type describes a context for both scatters and gathers.

To exchange data between the indices isX of vecX and the indices isY of vecY (length of isX = length of isY):

```c
VecScatterCreate(Vec vecX, IS isX, Vec vecY, IS isY,
                 VecScatter* the_context);
```

⇒ to copy elements from X to Y:

```c
VecScatterBegin(the_context, vecX, vecY, INSERT_VALUES,
                 SCATTER_FORWARD);
VecScatterEnd(the_context, vecX, vecY, INSERT_VALUES,
             SCATTER_FORWARD);
```

Notes:
- Conventional scatter if isX = stride with step 1.
- Conventional gather if isY = stride with step 1.
The VecScatter type describes a context for both scatters and gathers.

To exchange data between the indices `isX` of `vecX` and the indices `isY` of `vecY` 
(length of `isX` = length of `isY`):

```c
VecScatterCreate(Vec vecX, IS isX, Vec vecY, IS isY,
                 VecScatter* the_context);
```

⇒ to copy elements from `X` to `Y`:

```c
VecScatterBegin(the_context, vecX, vecY, INSERT_VALUES,
                 SCATTER_FORWARD);
VecScatterEnd(the_context, vecX, vecY, INSERT_VALUES,
              SCATTER_FORWARD);
```

Notes:
- Conventional scatter if `isX` = stride with step 1.
  Conventional gather if `isY` = stride with step 1.
- `ADD_VALUES`, `SCATTER_BACKWARD` also available.
PETSc Vectors : Exercise 1

Create a parallel vector with
- each local size equals to one plus the corresponding MPI rank,
- all the vector values set to half the MPI size,
and print the resulting vector on a various number of cores.

Result on 3 cores :

<table>
<thead>
<tr>
<th>Vector Object: 3 MPI processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: mpi</td>
</tr>
<tr>
<td>Process [0]</td>
</tr>
<tr>
<td>1.5</td>
</tr>
<tr>
<td>Process [1]</td>
</tr>
<tr>
<td>1.5</td>
</tr>
<tr>
<td>1.5</td>
</tr>
<tr>
<td>Process [2]</td>
</tr>
<tr>
<td>1.5</td>
</tr>
<tr>
<td>1.5</td>
</tr>
<tr>
<td>1.5</td>
</tr>
</tbody>
</table>
Duplicate the vector resulting from exercise 1 and copy the same values into it.

Use VecDot to compute the dot product of the two vectors and verify that the result equals the square of the 2-norm (computed using VecNorm).
Create a parallel vector of global size 100,000,000 and let PETSc decide the parallel distribution.

Use VecGetOwnershipRange to get the local indices and set each vector value equal to its index using first VecSetValue, then VecSetValues.

Compare the speeds of both options (using the Linux time command or the PetscGetTime function from PETSc).
Use \texttt{VecScatter} to get off-process values:

- Create a parallel vector \texttt{theVecMPI} of global size 100 and set each vector value equal to its index as in exercise 3.
- Create a sequential vector \texttt{theVecSeq} of size 3 (in fact one sequential vector is created on each process).
- Create an index set for both the parallel and sequential vectors, and use it to gather the elements 6, 45 and 97 from \texttt{theVecMPI} on each local copy of the sequential vector \texttt{theVecSeq}.