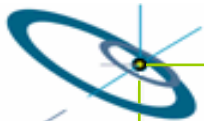


MPI

collective operations

Edgar Gabriel

High Performance Computing Center Stuttgart (HLRS)
gabriel@hls.de



Edgar Gabriel

High Performance Computing Center Stuttgart



Collective operation

- all process of a process group have to participate in the same operation
 - process group is defined by a communicator
 - all processes have to provide the same arguments
 - for each communicator, you can have one collective operation ongoing at a time
- collective operations are abstractions for often occurring communication patterns
 - eases programming
 - enables low-level optimizations and adaptations to the hardware infrastructure



MPI collective operations

MPI_Barrier

MPI_Bcast

MPI_Scatter

MPI_Scatterv

MPI_Gather

MPI_Gatherv

MPI_Allgather

MPI_Allgatherv

MPI_Alltoall

MPI_Alltoallv

MPI_Reduce

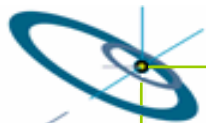
MPI_Allreduce

MPI_Reduce_scatter

MPI_Scan

MPI_Exscan

MPI_Alltoallw



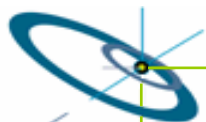
Edgar Gabriel

High Performance Computing Center Stuttgart



More MPI collective operations

- Creating and freeing a communicator is considered a collective operation
 - e.g. `MPI_Comm_create`
 - e.g. `MPI_Comm_spawn`
- Collective I/O operations
 - e.g. `MPI_Write_all`
- Window synchronization calls are collective operations
 - e.g. `MPI_Win_fence`



MPI_Barrier

```
MPI_Barrier (MPI_Comm comm);
```

- Synchronizes all processes of the communicator
 - no process can continue with the execution of the application until all process of the communicator have reached this function
 - often used before timing certain sections of the application
- MPI makes no statement about the quality of the synchronization
- Advice: no scenario is known to me, which requires a barrier for correctness. Usage of `MPI_Barrier` strongly disadvised.



MPI_Bcast

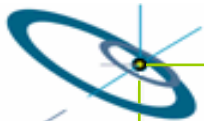
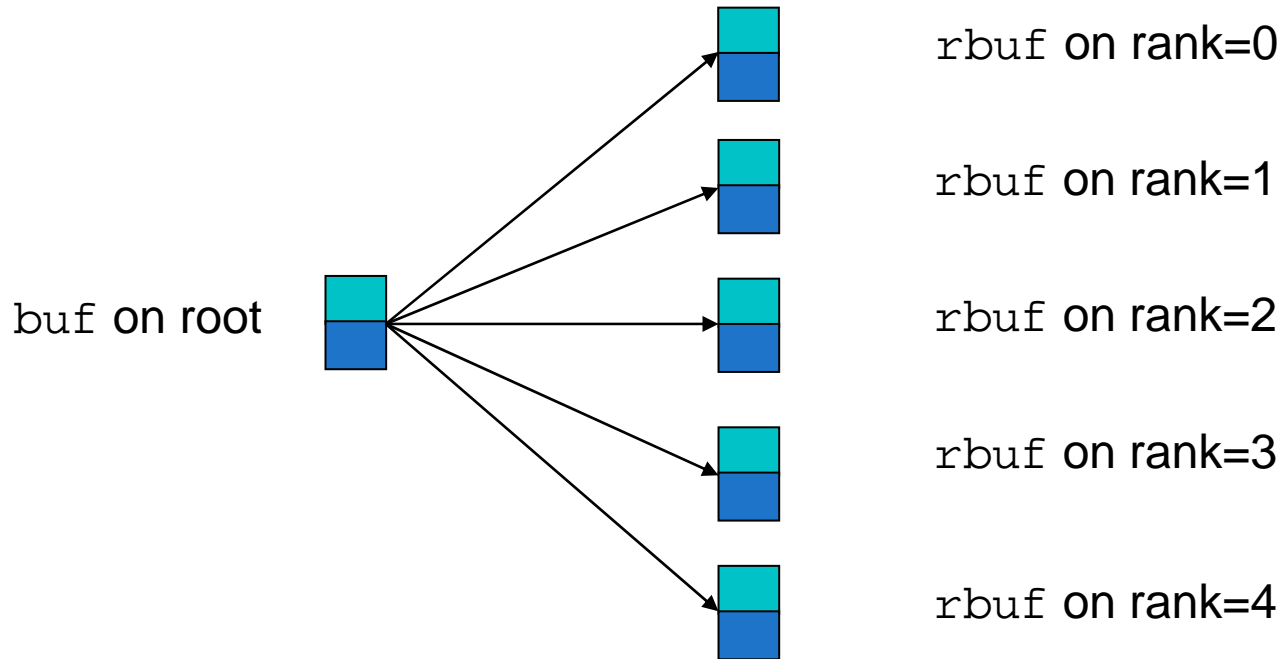
```
MPI_Bcast (void *buf, int cnt, MPI_Datatype dat,  
          int root, MPI_Comm comm);
```

- The process with the rank `root` distributes the data stored in `buf` to all other processes in the communicator `comm`.
- Data in `buf` is identical on all processes after the bcast
- Compared to point-to-point operations no `tag`, since you cannot have several ongoing collective operations



MPI_Bcast (II)

```
MPI_Bcast (buf, 2, MPI_INT, 0, comm);
```



Example: distributing global parameters

```
int rank, problemsize;
float precision;
MPI_Comm comm=MPI_COMM_WORLD;

MPI_Comm_rank ( comm, &rank );
if (rank == 0 ) {
    FILE *myfile;
    myfile = fopen("testfile.txt", "r");
    fscanf (myfile, "%d", &problemsize);
    fscanf (myfile, "%f", &precision);
    fclose (myfile);
}

MPI_Bcast (&problemsize, 1, MPI_INT, 0, comm);
MPI_Bcast (&precision, 1, MPI_FLOAT, 0, comm);
```



MPI_Scatter

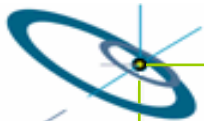
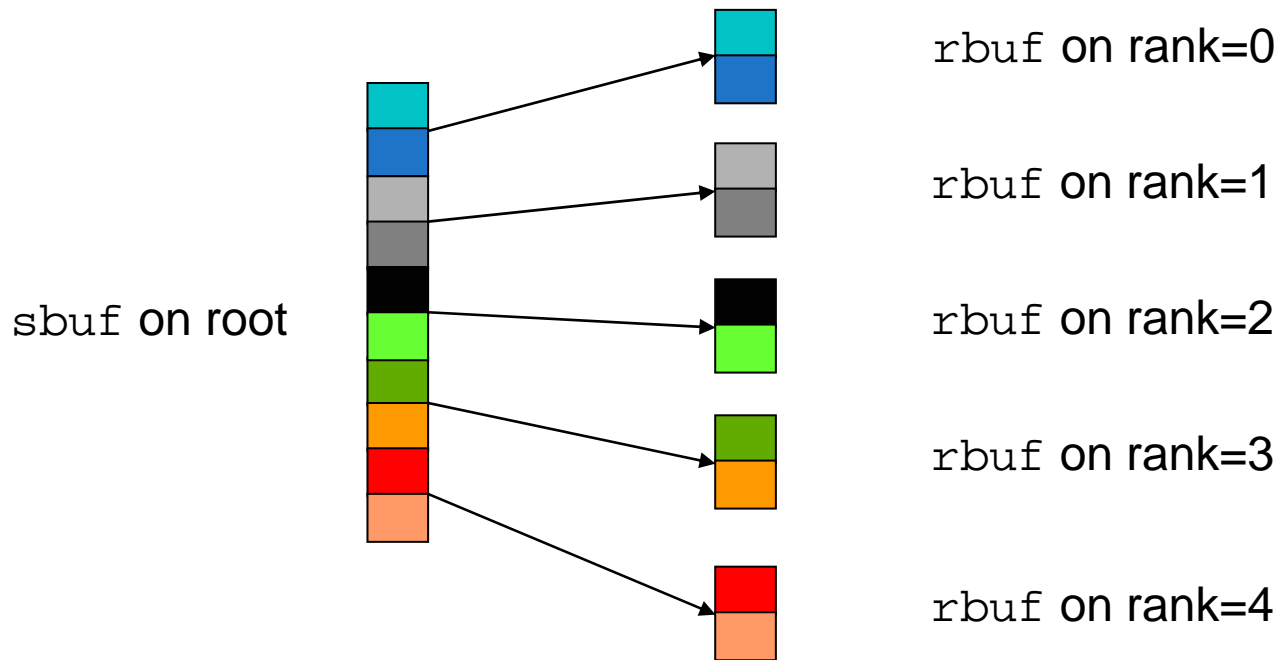
```
MPI_Scatter (void *sbuf, int scnt, MPI_Datatype sdat,  
            void *rbuf, int rcnt, MPI_Datatype rdat,  
            int root, MPI_Comm comm);
```

- The process with the rank `root` distributes the data stored in `sbuf` to all other processes in the communicator `comm`
- Difference to Broadcast: every process gets different segment of the original data at the root process
- Arguments `sbuf`, `scnt`, `sdat` only relevant and have to be set at the root-process



MPI_Scatter (II)

```
MPI_Scatter (sbuf, 2, MPI_INT, rbuf, 2, MPI_INT, 0, comm);
```



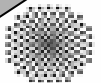
Example: partition a vector among processes

```
int rank, size;
float *sbuf, rbuf[3] ;
MPI_Comm comm=MPI_COMM_WORLD;

MPI_Comm_rank ( comm, &rank );
MPI_Comm_size ( comm, &size );

if (rank == root ) {
    sbuf = malloc (3*size*sizeof(float);
    /* set sbuf to required values etc. */
}

/* distribute the vector, 3 Elements for each process */
MPI_Scatter (sbuf, 3, MPI_FLOAT, rbuf, 3, MPI_FLOAT,
            root, comm);
if ( rank == root ) {
    free (sbuf);
}
```



MPI_Gather

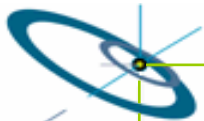
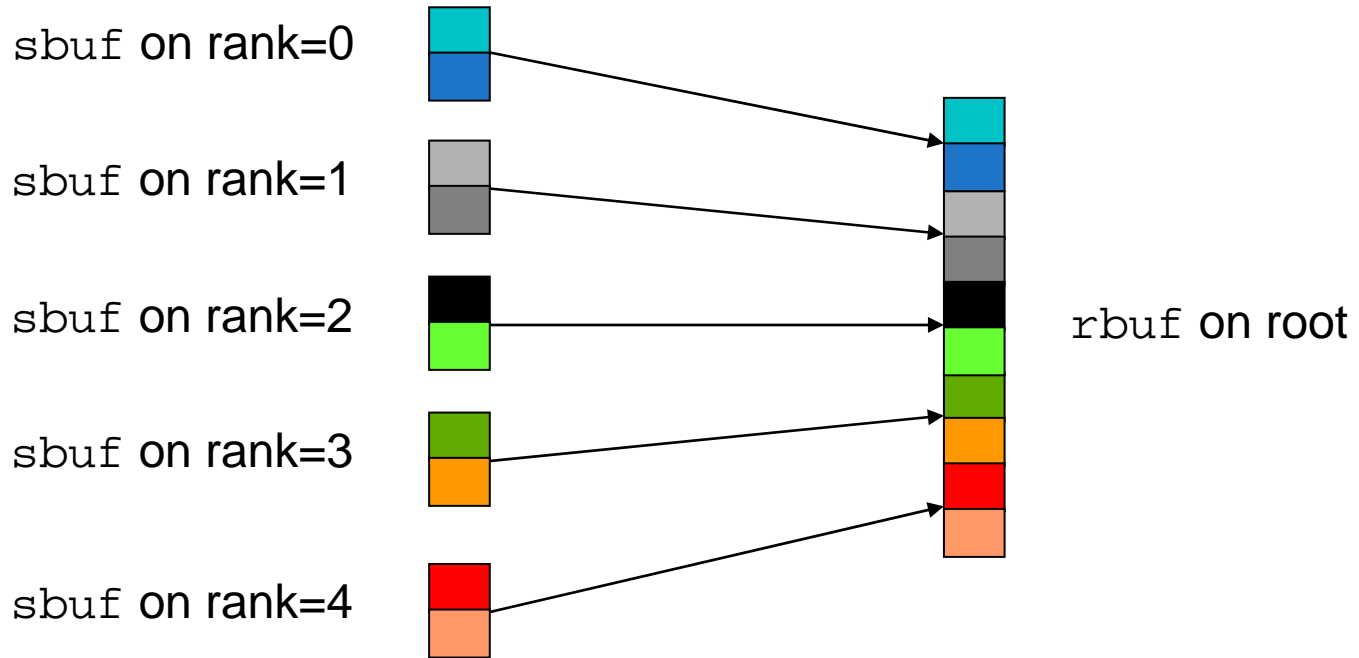
```
MPI_Gather (void *sbuf, int scnt, MPI_Datatype sdat,  
           void *rbuf, int rcnt, MPI_Datatype rdat,  
           int root, MPI_Comm comm);
```

- Reverse operation of `MPI_Scatter`
- The process with the rank `root` receives the data stored in `sbuf` on all other processes in the communicator `comm` into the `rbuf`
- Arguments `rbuf`, `rcnt`, `rdat` only relevant and have to be set at the root-process



MPI_Gather (II)

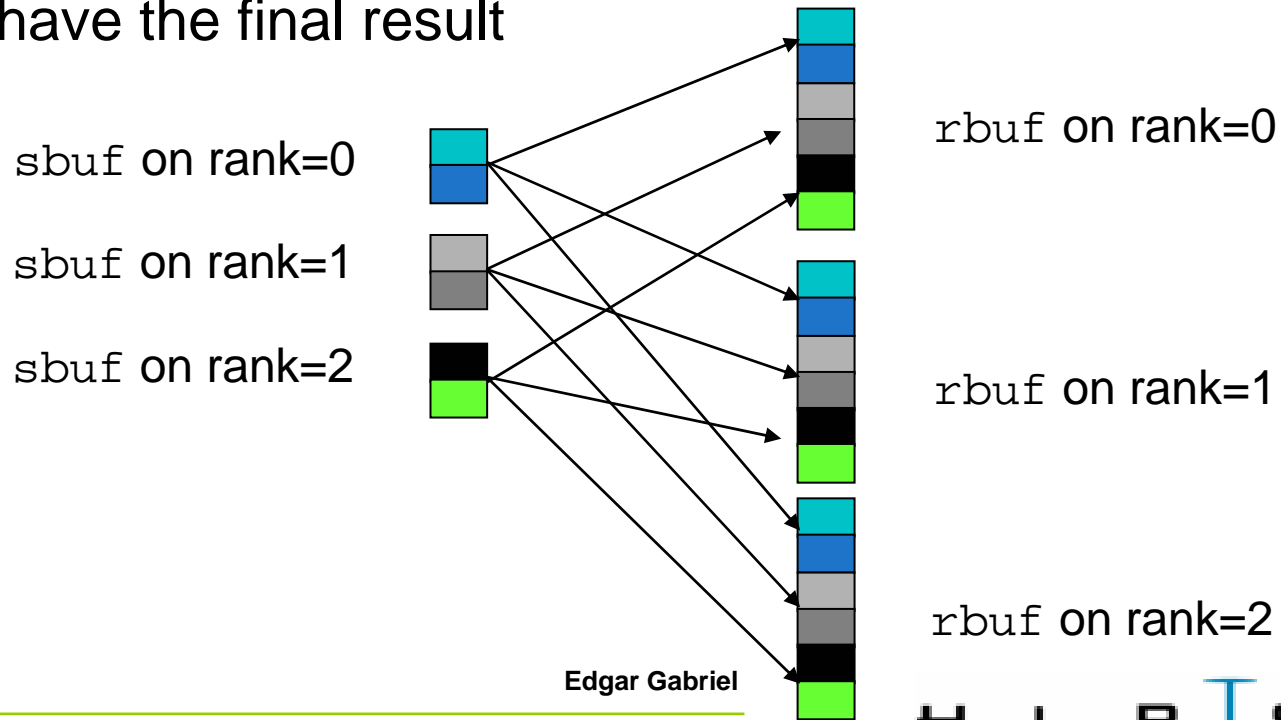
```
MPI_Gather (sbuf, 2, MPI_INT, rbuf, 2, MPI_INT, 0, comm);
```



MPI_Allgather

```
MPI_Allgather (void *sbuf, int scnt, MPI_Datatype sdat,  
              void *rbuf, int rcnt, MPI_Datatype rdat,  
              MPI_Comm comm);
```

- Identical to MPI_Gather, except that all processes have the final result



Edgar Gabriel

High Performance Computing Center Stuttgart

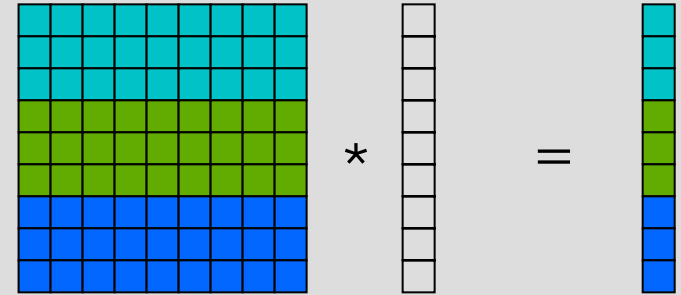
Example: matrix-vector multiplication with row-wise block distribution

```
int main( int argc, char **argv)
{
  double A[nlocal][n], b[n];
  double c[nlocal], cglobal[n];
  int i,j;

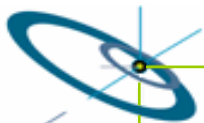
  ...

  for (i=0; i<n; i++) {
    for ( j=0;j<nlocal;j++ ) {
      c[i] = c[i] + A(i,j)*b(j);
    }
  }
```

```
MPI_Allgather( c, nlocal, MPI_DOUBLE, cglobal, nlocal,
              MPI_DOUBLE, MPI_COMM_WORLD );
```



Each process holds the final result for its part of c



Reduction operations

```
MPI_Reduce (void *inbuf, void *outbuf, int cnt,  
            MPI_Datatype dat, MPI_Op op, int root,  
            MPI_Comm comm);  
MPI_Allreduce (void *inbuf, void *outbuf, int cnt,  
               MPI_Datatype dat, MPI_Op op,  
               MPI_Comm comm);
```

- Perform simple calculations (e.g. calculate the sum or the product) over all processes in the communicator
- MPI_Reduce
 - `outbuf` has to be provided by all processes
 - result is only available at `root`
- MPI_Allreduce
 - result available on all processes



Predefined reduction operations

- MPI_SUM sum
- MPI_PROD product
- MPI_MIN minimum
- MPI_MAX maximum
- MPI_LAND logical and
- MPI_LOR logical or
- MPI_LXOR logical exclusive or
- MPI_BAND binary and
- MPI_BOR binary or
- MPI_BXOR binary exclusive or
- MPI_MAXLOC maximum value and location
- MPI_MINLOC minimum value and location



Simple example

```
int rank, size, root=0;
int globalmax, globalmin;

MPI_Comm_rank ( MPI_COMM_WORLD, &rank);
MPI_Comm_size ( MPI_COMM_WORLD, &size);

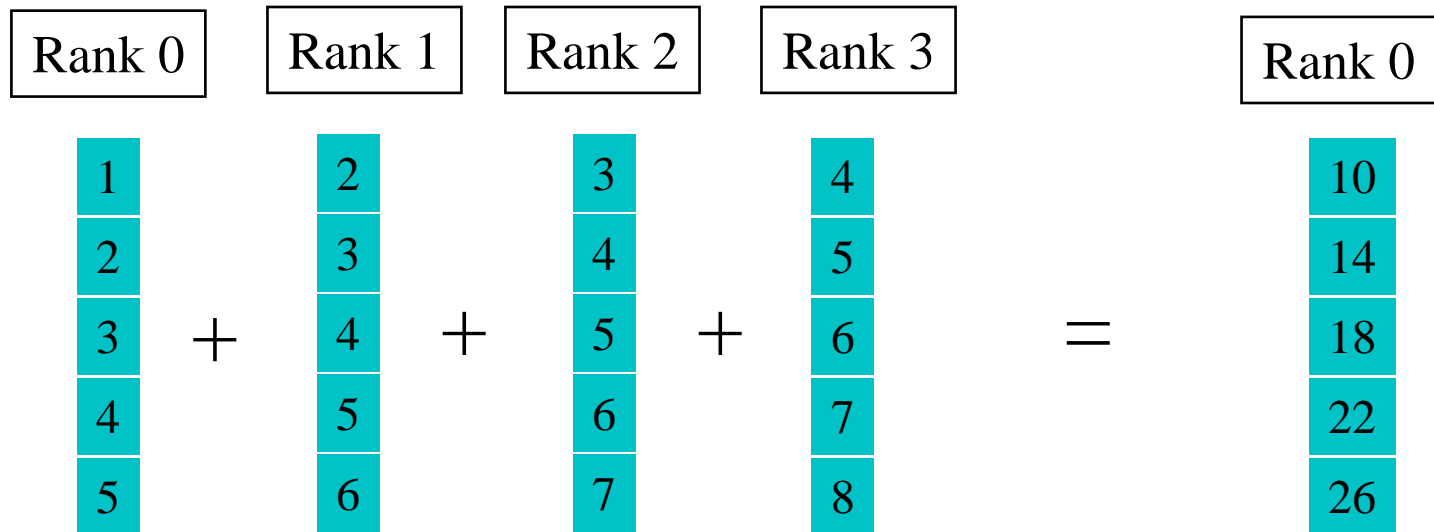
MPI_Reduce ( &rank, &globalmax, 1, MPI_INT, MPI_MAX, 0,
            MPI_COMM_WORLD);
if ( globalmax != (size -1 ) && ( rank == root ) ) {
    printf("You claim that one of the basic principles of MPI"
           " did not work!");
}

MPI_Reduce ( &rank, &globalmin, 1, MPI_INT, MPI_MIN, 0,
            MPI_COMM_WORLD);
if ( globalmin != 0 && rank == root ) {
    printf("You claim that one of the basic principles of MPI"
           " did not work!");
}
```



Reduction operations on vectors

- Reduce operation is executed element wise on each entry of the array



- Reduction of 5 elements with root = 0



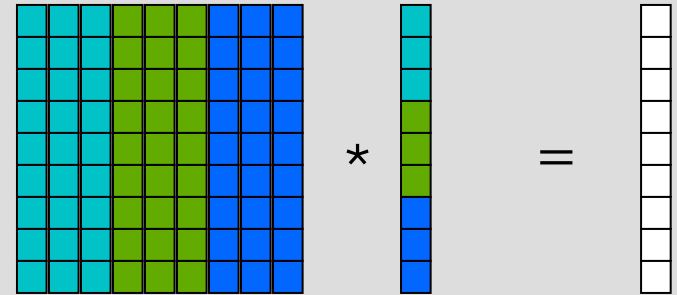
Example: matrix-vector multiplication with column-wise block distribution

```
int main( int argc, char **argv)
{
  double A[n][nlocal], b[nlocal];
  double c[n], ct[n];
  int i,j;
```

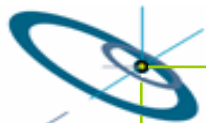
...

```
for (i=0; i<n; i++) {
  for ( j=0; j<nlocal; j++ ) {
    ct[i] = ct[i] + A(i,j)*b(j);
  }
}
```

```
MPI_Allreduce ( ct, c, n, MPI_DOUBLE, MPI_SUM,
               MPI_COMM_WORLD );
```



Result of local computation in temporary buffer



MPI_IN_PLACE

- In many cases the input and output buffer should be identical
 - e.g. the result of the reduction operation should replace the input value
 - pass `MPI_IN_PLACE` instead of `sendbuf` for reduction operations
 - saves memory (unless the MPI library simple allocates the missing memory ☹)

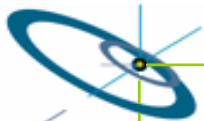
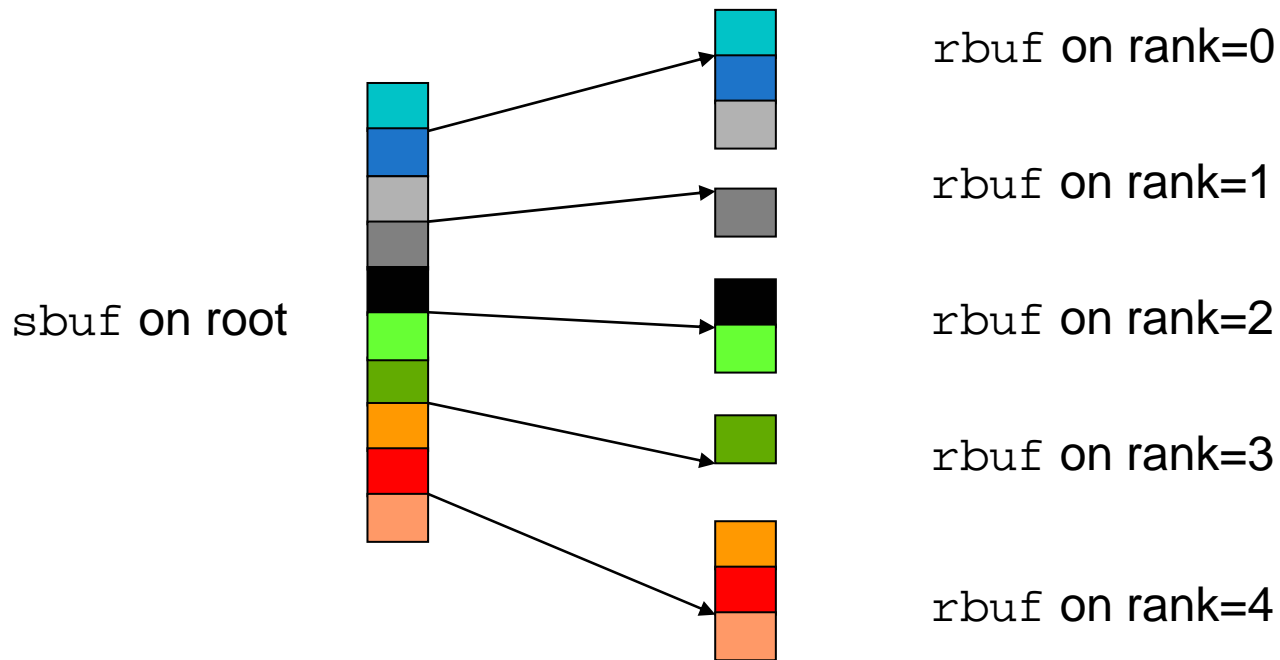


The v-operations

- Some collective operations have a vector version define:
 - MPI_Scatterv
 - MPI_Gatherv
 - MPI_Allgatherv
 - MPI_Alltoallv
- The difference between the vector version and the regular version is, that the input/output value of the processes does not have to be identical
- In case of rooted operations, the root has to know the data-length of every process. All other processes just need to know their own data-length
- In case of non-rooted operations, everybody has to know the data-length of every other process in the communicator



Example: MPI_Scatterv



What else is there

- A lot more functions, which we did not discuss here
 - `MPI_Alltoall(v, w)`
 - `MPI_(Ex)Scan`
 - `MPI_Reduce_scatter`
- User defined operations for reductions
 - e.g. if you are passing a user-defined (derived) datatype
- MPI-2 extended collective operation
 - definition of collective operations for inter-communicators

