Programming Using the Message-Passing Paradigm (cont.)

Alexandre David

B2-206

Collective Operation – Recall

- One-to-all broadcast – MPI_Bcast.
- All-to-one reduction – MPI_Reduce.
- All-to-all broadcast – MPI_Allgather.
- All-to-all reduction – MPI_Reduce_scatter.
- All-reduce and prefix sum – MPI_Allreduce.
- Scatter – MPI_Scatter.
- Gather – MPI_Gather.
- All-to-all personalized – MPI_Alltoall.

Collective Communication and Computation Operations

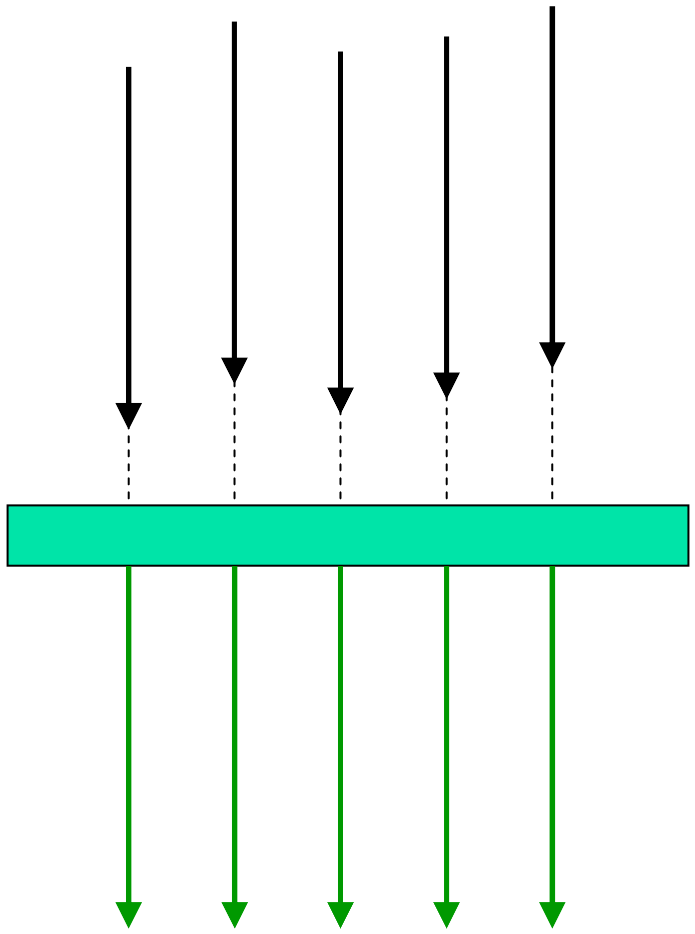
- Common collective operations supported.
  - Over a group or processes corresponding to a communicator.
  - All processes in the communicator must call these functions.
- These operations act like a virtual synchronization step.

Barrier

- Communicator: Group of processes that are synchronized.
- The function returns after all processes in the group have called the function.

```
int MPI_Barrier(MPI_Comm comm)
```

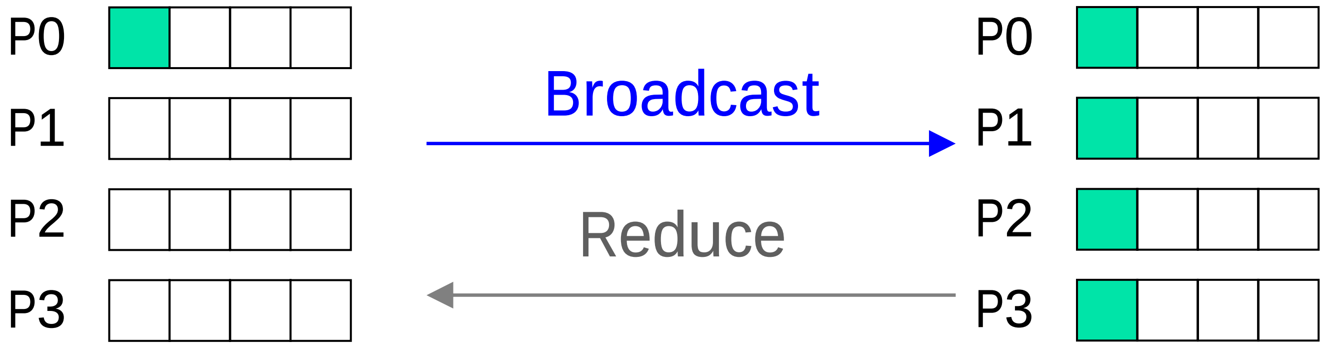
Barrier



One-to-All Broadcast

- All the processes must call this function, even the receivers.

```
int MPI_Bcast(void *buf,
        int count, MPI_Datatype datatype,
        int source, MPI_Comm comm)
```



All-to-One Reduction

- Combine elements in sendbuf (of each process in the group) using the operation op and return in recvbuf of target.
- See table 6.3 for the list of predefined operations that are supported.

```
int MPI_Reduce(void *sendbuf, void *recvbuf,
        int count,MPI_Datatype datatype,
        MPI_Op op, int target,
        MPI_Comm comm)
```

Special Operations

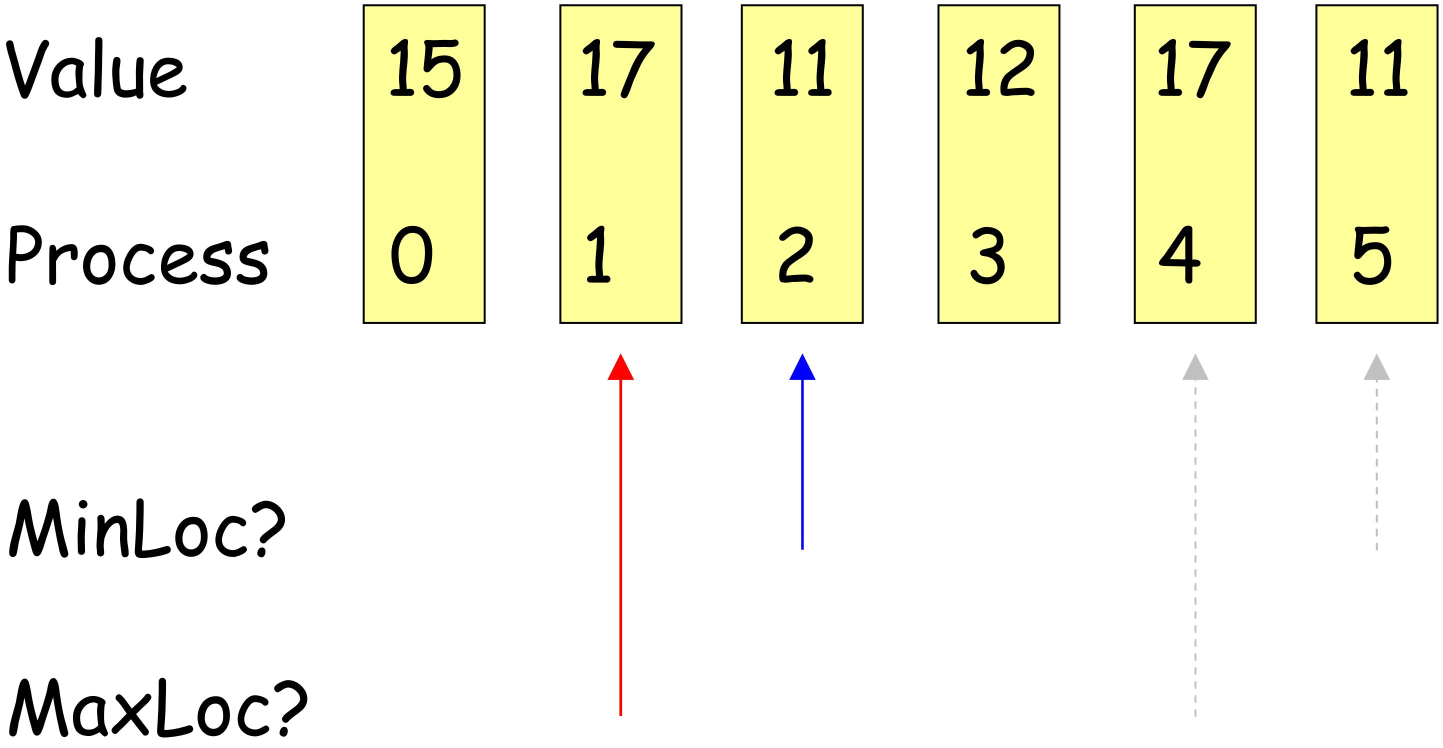
- MPI_MAXLOC and MPI_MINLOC work on pairs (v_i, l_i).

Payload.

Value for comparison.

- Compare with v_i, use l_i to break ties, and return (l,v).
- Additional MPI data-pair types defined.

Example



Value	15	17	11	12	17	11
Process	0	1	2	3	4	5

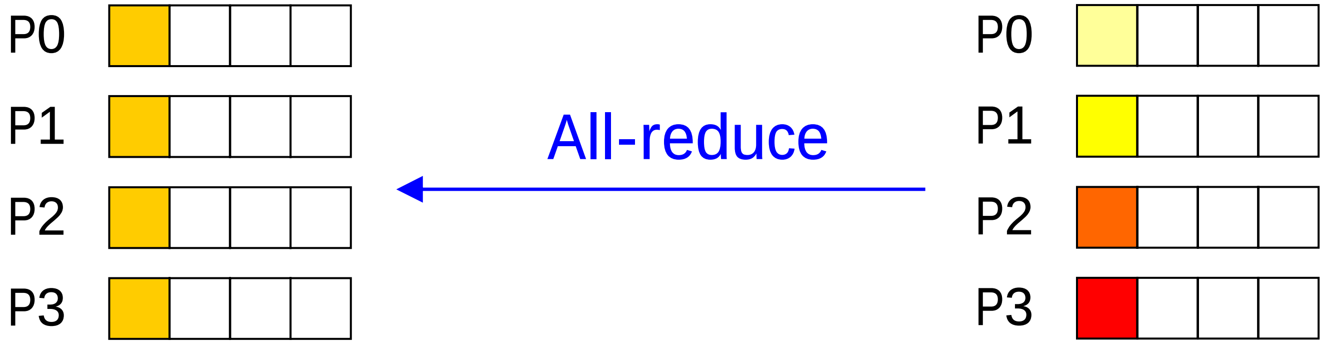
MinLoc?

MaxLoc?

All-Reduce

- No target argument since all processes receive the result.

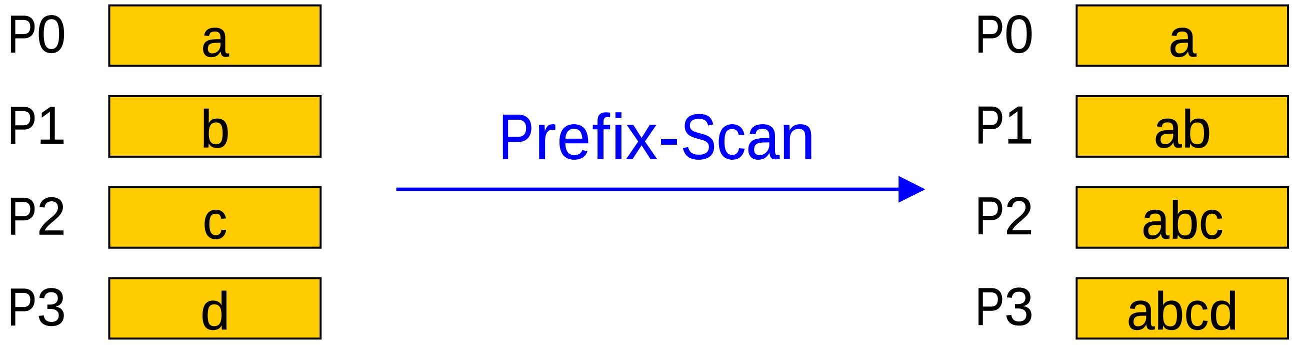
```
int MPI_Allreduce(void *sendbuf, void *recvbuf,
int count, MPI_Datatype datatype,
MPI_Op op, MPI_Comm comm)
```



Prefix-Operations

- Not only sums.
- Process *j* has prefix s_j as expected.

```
int MPI_Scan(void *sendbuf, void *recvbuf,
          int count, MPI_Datatype datatype, MPI_Op op,
          MPI_Comm comm)
```

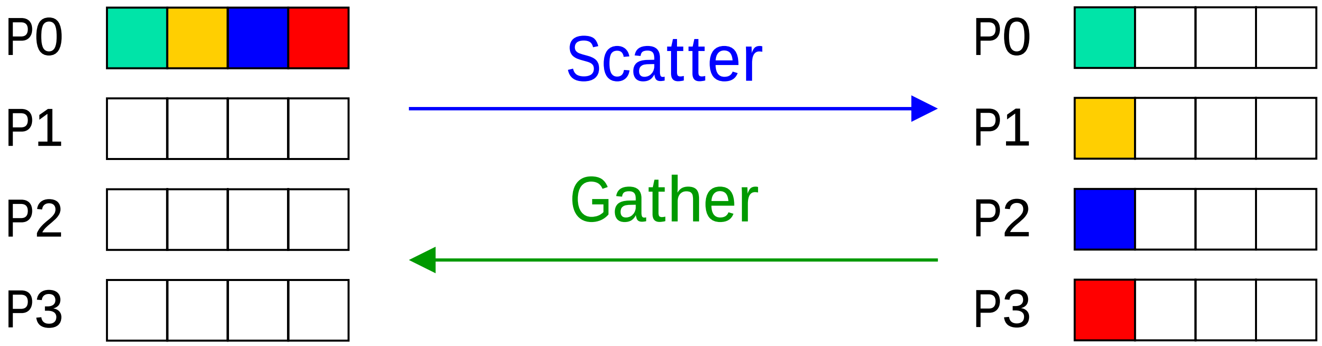


P0 a
P1 b
P2 c
P3 d

Prefix-Scan

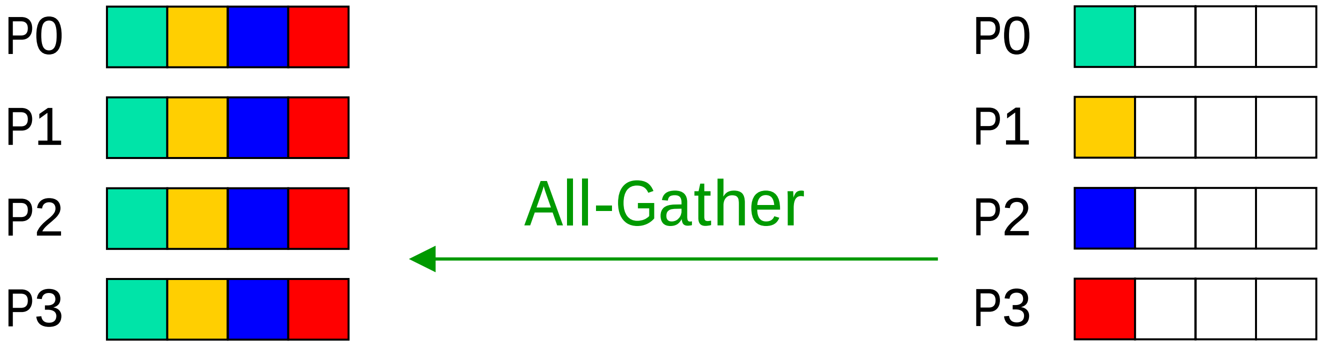
P0 a
P1 ab
P2 abc
P3 abcd

Scatter and Gather



All-Gather

- Variant of gather.



P0

P1

P2

P3

All-Gather

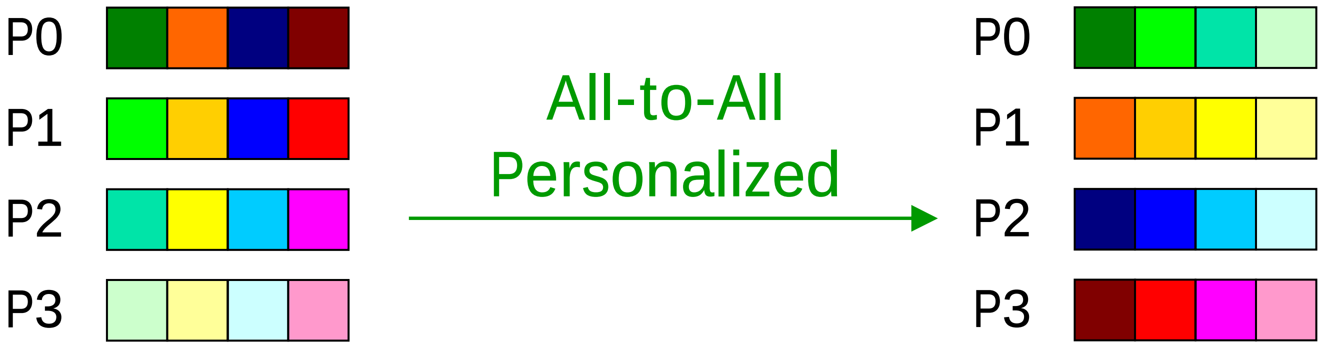
P0

P1

P2

P3

All-to-All Personalized



P0

P1

P2

P3

All-to-All Personalized

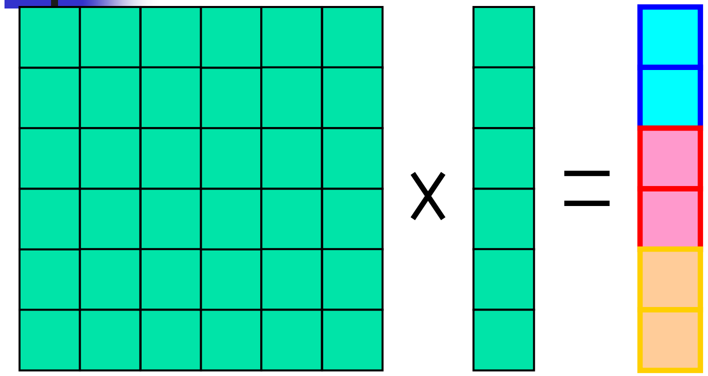
P0

P1

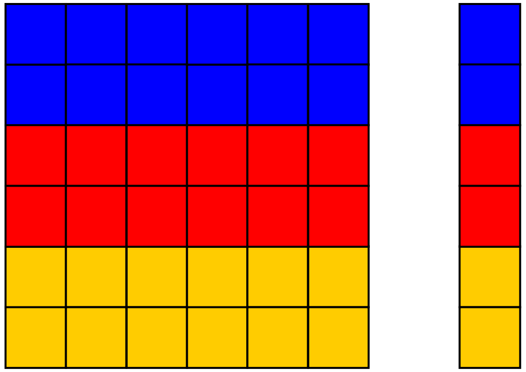
P2

P3

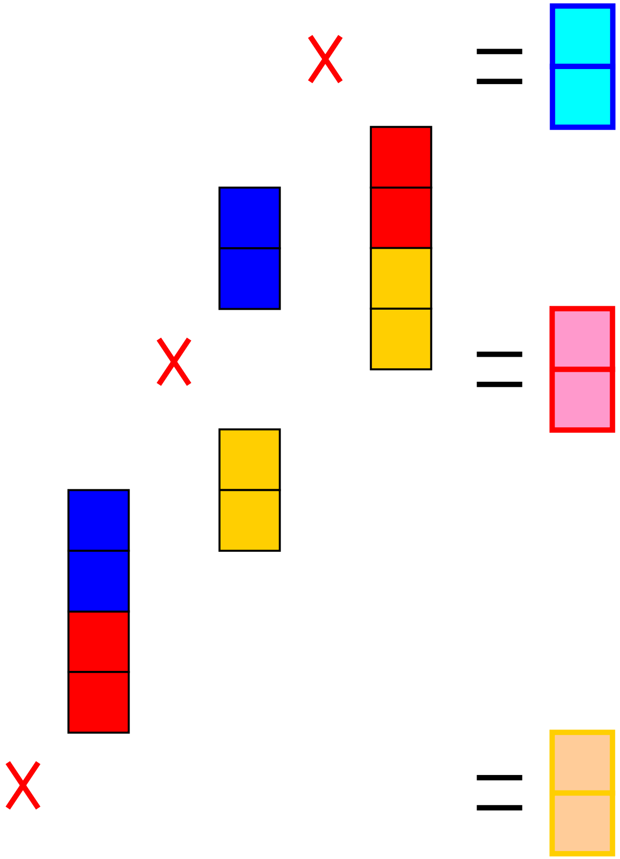
Example Matrix*Vector (Program 6.4)



Partition on rows.



Allgather (All-to-all broadcast)



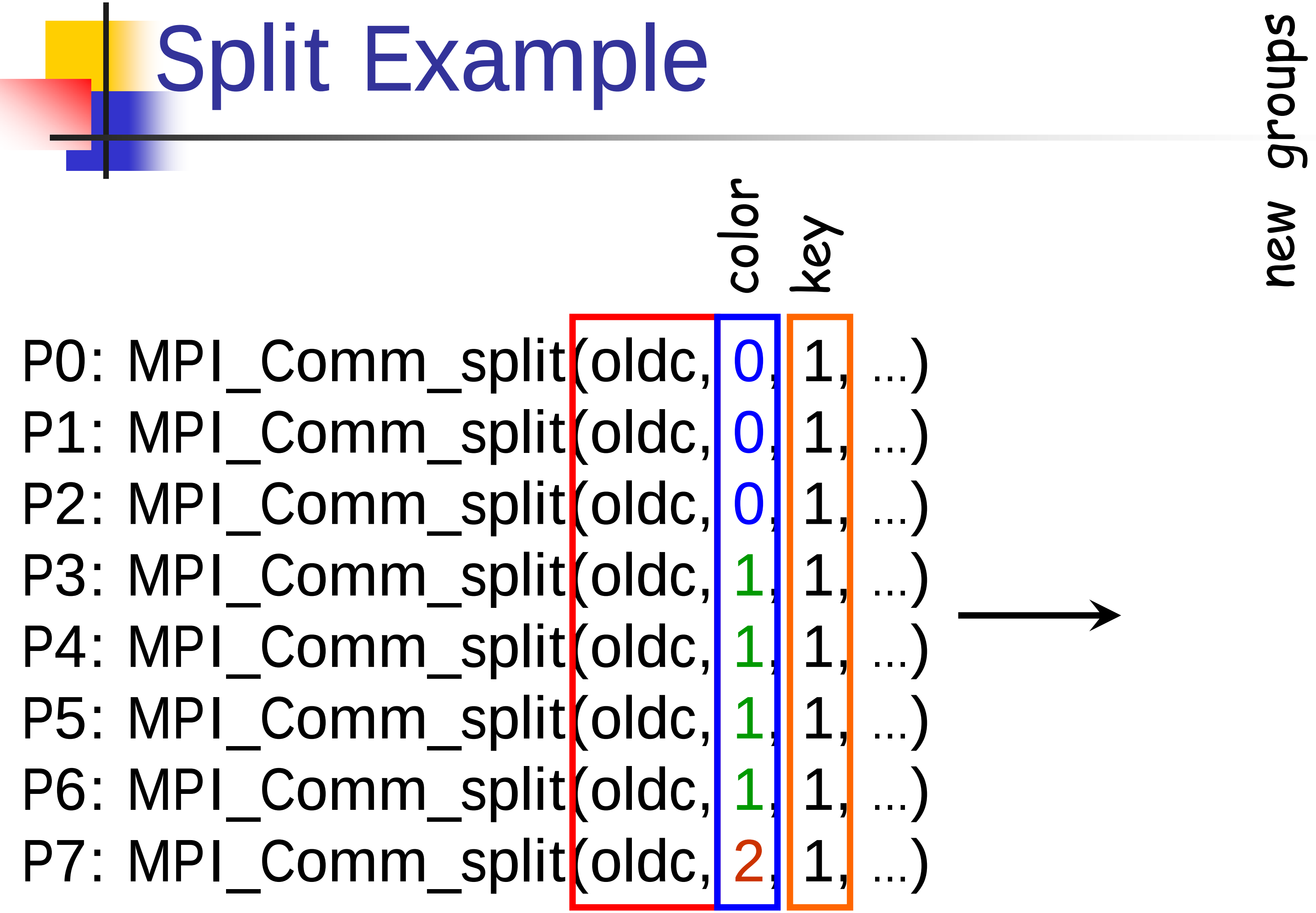
Multiply

Groups and Communicators

- How to partition a group of processes into sub-groups?
- Group by color (different communicators).
- Sort by key (new ranks in the sub-groups).

```
int MPI_Comm_split(MPI_Comm comm,
        int color, int key,
        MPI_Comm *newcomm)
```

Split Example



```
P0: MPI_Comm_split(oldc, 0, 1, ...)
P1: MPI_Comm_split(oldc, 0, 1, ...)
P2: MPI_Comm_split(oldc, 0, 1, ...)
P3: MPI_Comm_split(oldc, 1, 1, ...)
P4: MPI_Comm_split(oldc, 1, 1, ...)
P5: MPI_Comm_split(oldc, 1, 1, ...)
P6: MPI_Comm_split(oldc, 1, 1, ...)
P7: MPI_Comm_split(oldc, 2, 1, ...)
```

Splitting Cartesian Topologies

- Split Cartesian topology into lower dimensional grids.

Original group.

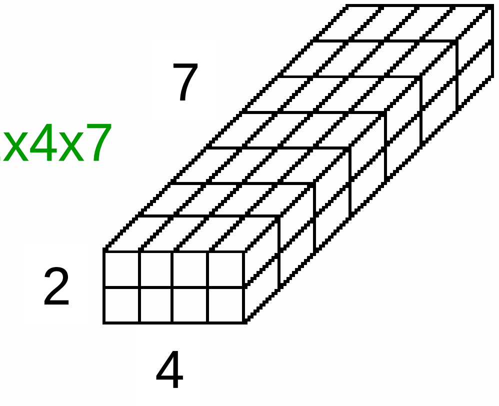
```
int MPI_Cart_sub(MPI_Comm comm_cart,
        int *keep_dims, MPI_Comm *comm_subcart)
```

Tell which dimensions to keep, e.g,
2x4x7 and {1,0,1} → 4* sub (2x7)

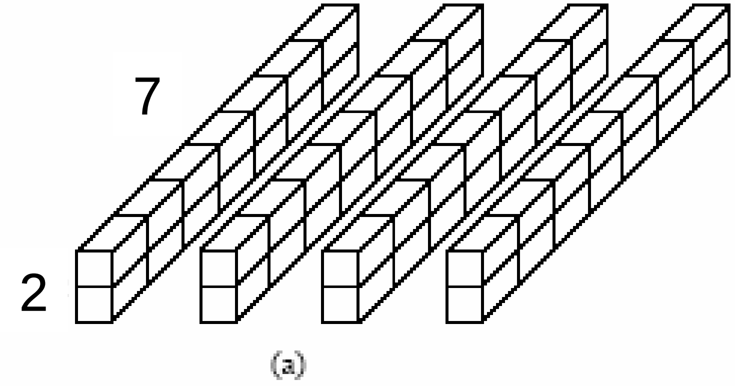
New group.

Example

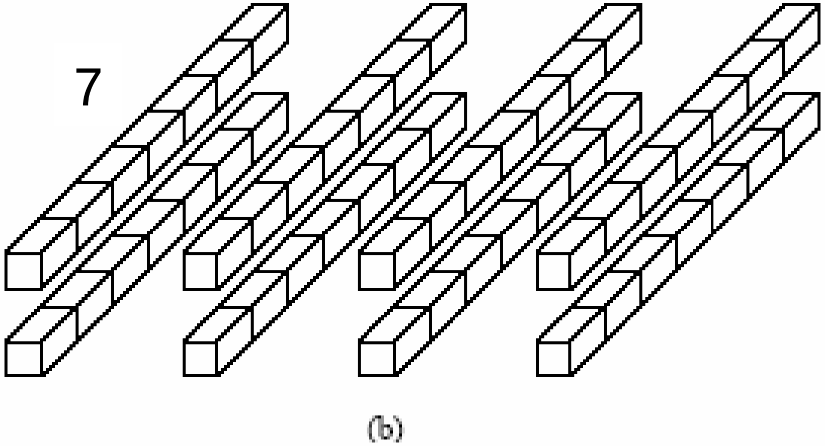
original 2x4x7



(1,0,1) -> 4* (2x7)



(0,0,1) -> 2*4* (7)

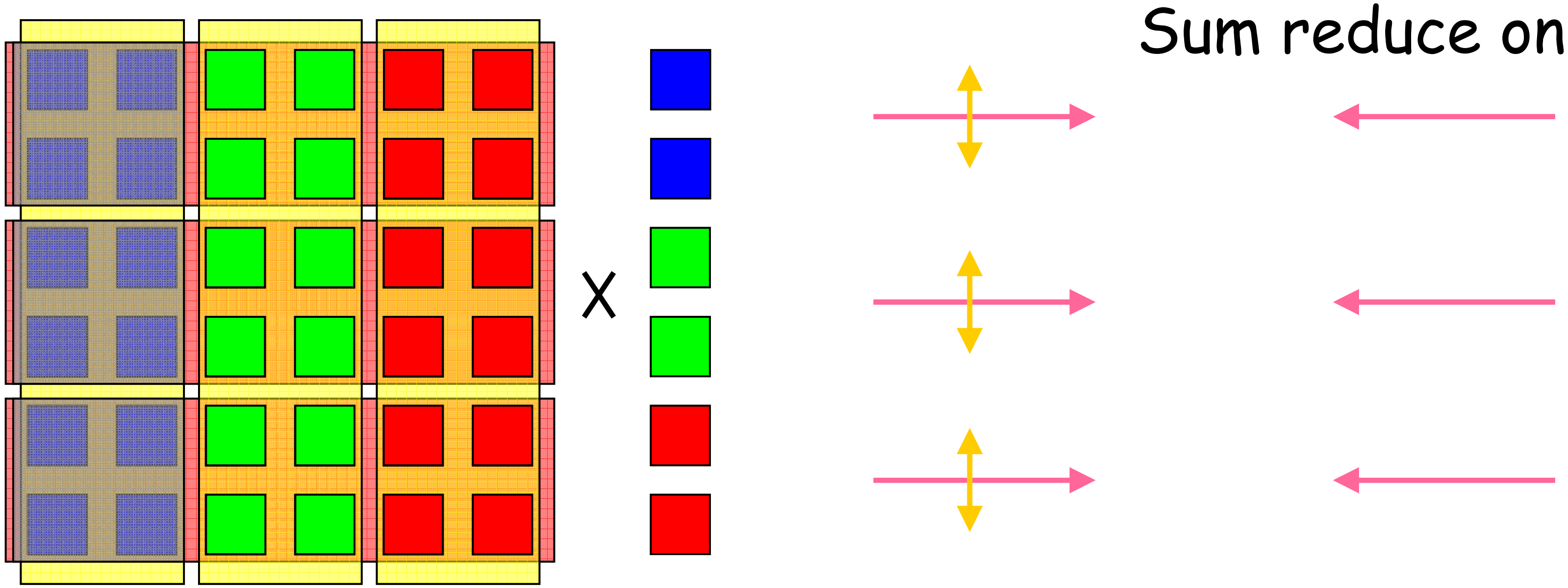


Example Matrix*Vector (Program 6.8)

Partition.

Distribute vector.

Sum reduce on rows.



Row sub-topology.

Local multiplication.

Colum sub-topology.

Performance Evaluation

- Elapsed time.

```
double t1, t2;
t1=MPI_Wtime();
...
t2=MPI_Wtime();
printf("Elapsed time is %f sec\n", t2-t1);
```