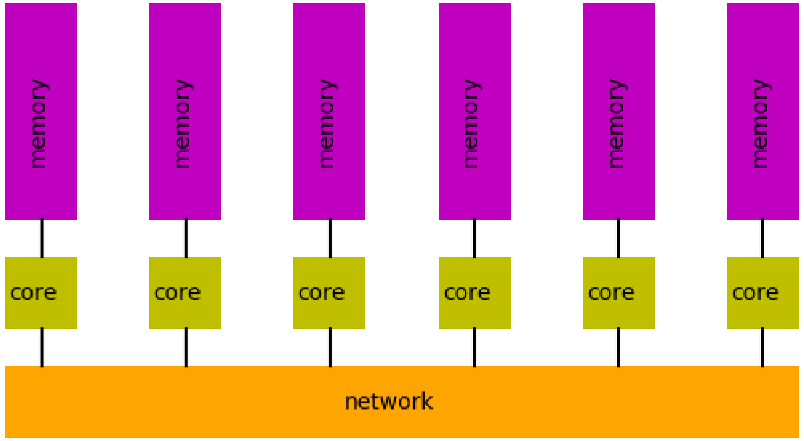
MPI introduction: Part 1

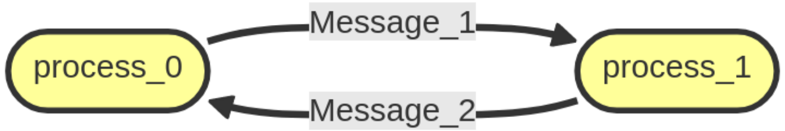
The target

- MPI is intended for distributed memory computers (clusters).

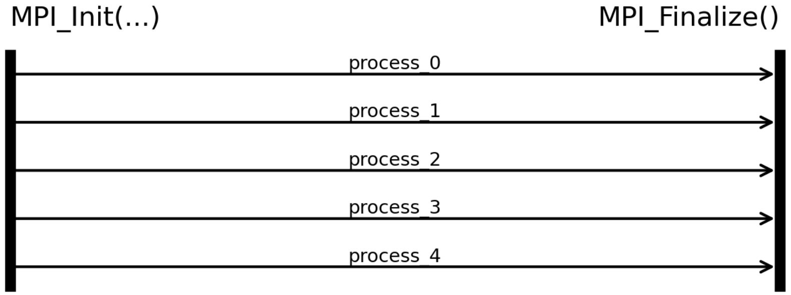


The Concept

- Relies on message exchanges between processes



- Requires full code parallelization (incremental parallelization is difficult and not recommended)



- Should be introduced during the application's design

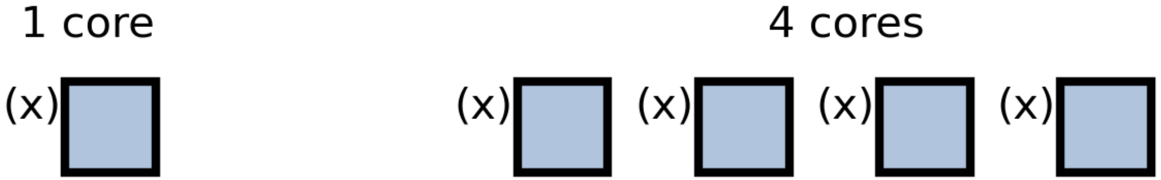
An Introduction to MPI in Two Parts

Part 1 : sharing work through data partitioning and distribution

Part 2 : different types of communications and the concept of ghost points

Variable Status

- there are only private variables
- they exist in each process/core.



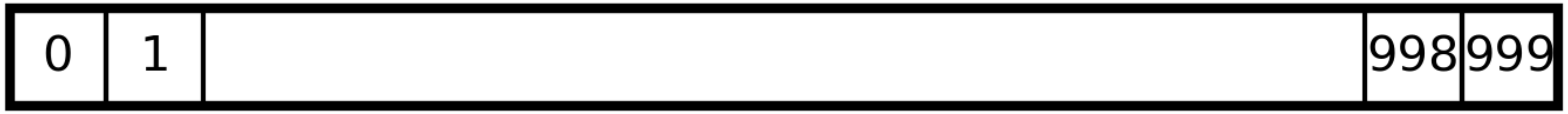
Study Case : the sequential algorithm

Lets consider the following problem:

- x is a vector of integers with a size of 1000.
- it is initialized with $x[i] = i$

```
vector<int> x(1000);  // define a vector of size 1000
for (int i = 0; i < 1000; ++i)
  x[i] = i;
```

original vector



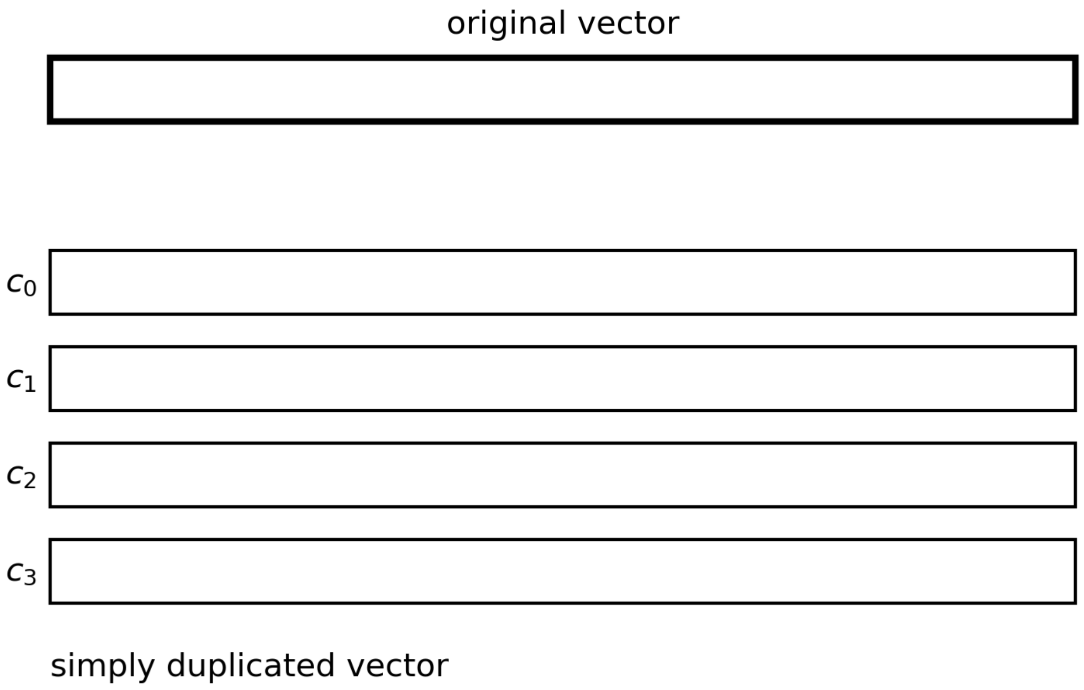
Study Case : parallel algorithm (OpenMP)

```
vector<int> x(1000);  // define a vector of size 1000
#pragma omp parallel for // parallelize the initialization
for(int i = 0; i < 1000; ++i)
  x[i] = i;
```

Study Case : parallel algorithm (MPI)

Creation of the distributed vector

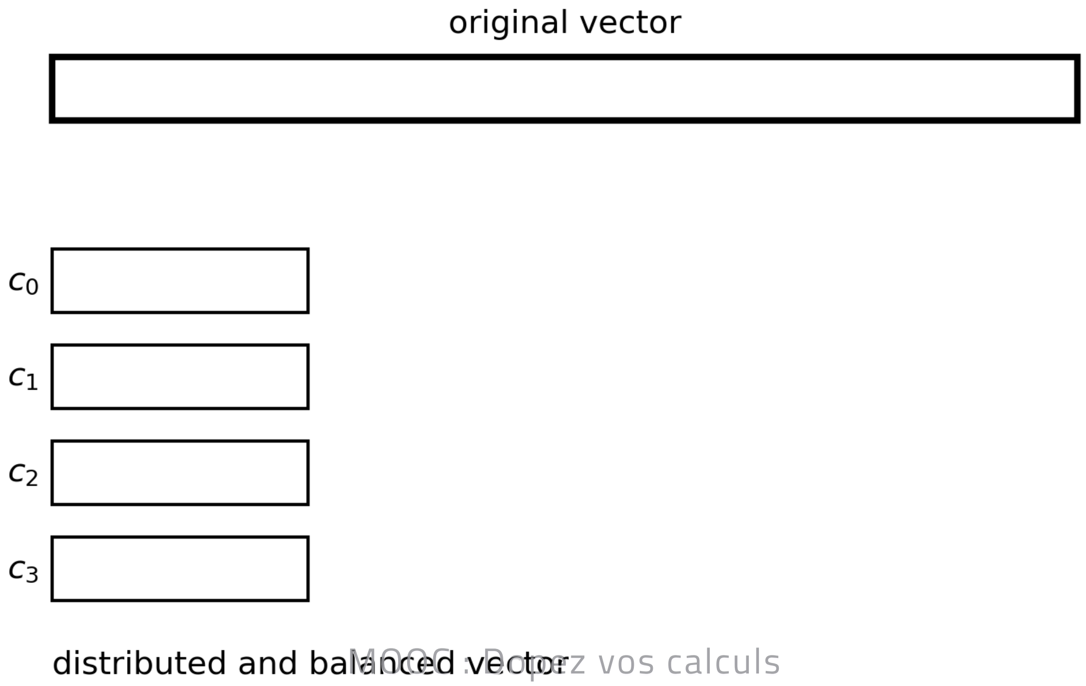
```cpp
vector<int> x(1000); // duplicate the vector across all cores.
```



Study Case : parallel algorithm (MPI)

Creation of the distributed vector

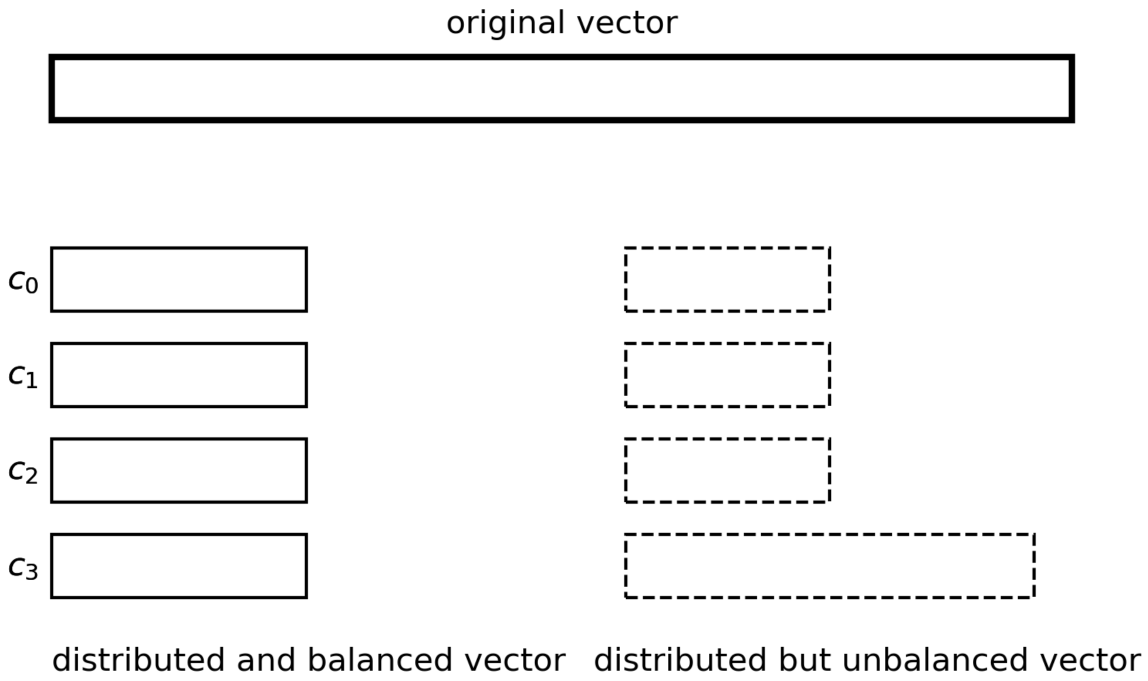
```
int size;
MPI_Comm_size(MPI_COMM_WORLD,&size);
vector<int> x(1000 / size); // distribute the vector across all cores
```



Study Case : parallel algorithm (MPI)

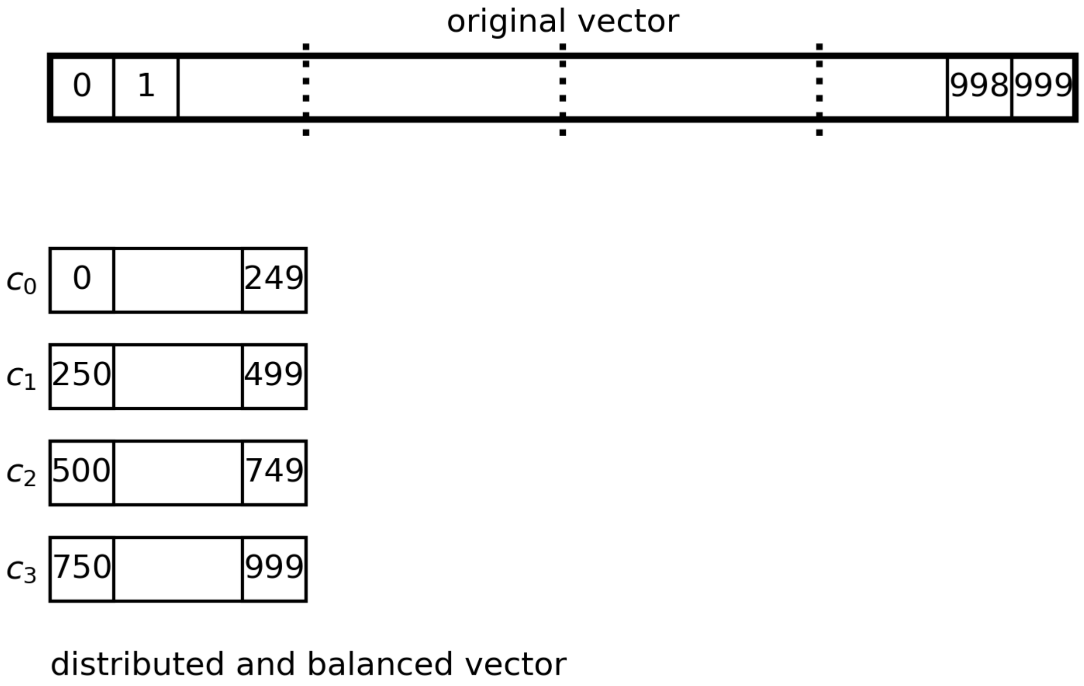
Creation of the distributed vector

Another possible partitioning, but unbalanced.


original vector
c_0
c_1
c_2
c_3
distributed and balanced vector
distributed but unbalanced vector

Study Case : parallel algorithm (MPI)

Initialization of the distributed vector: Subvectors should be organized in a way that allows representing the original vector.



Study Case : parallel algorithm (MPI)

Initialization of the distributed vector: Subvectors should be organized in a way that allows representing the original vector.

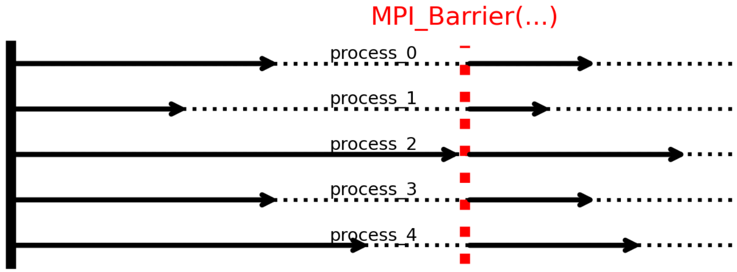
```
int rank; // current process
int size; // number of processes
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);
int xlocalsize = 1000 / size; // size of subvectors
vector<int> x(xlocalsize); // define subvectors

for (int i = 0; i < xlocalsize; ++i)  // xlocalsize elements
  x[i] = (rank * xlocalsize) + i; // offset by rank*xlocalsize
```

Synchronizations

- explicit synchronization of all processes

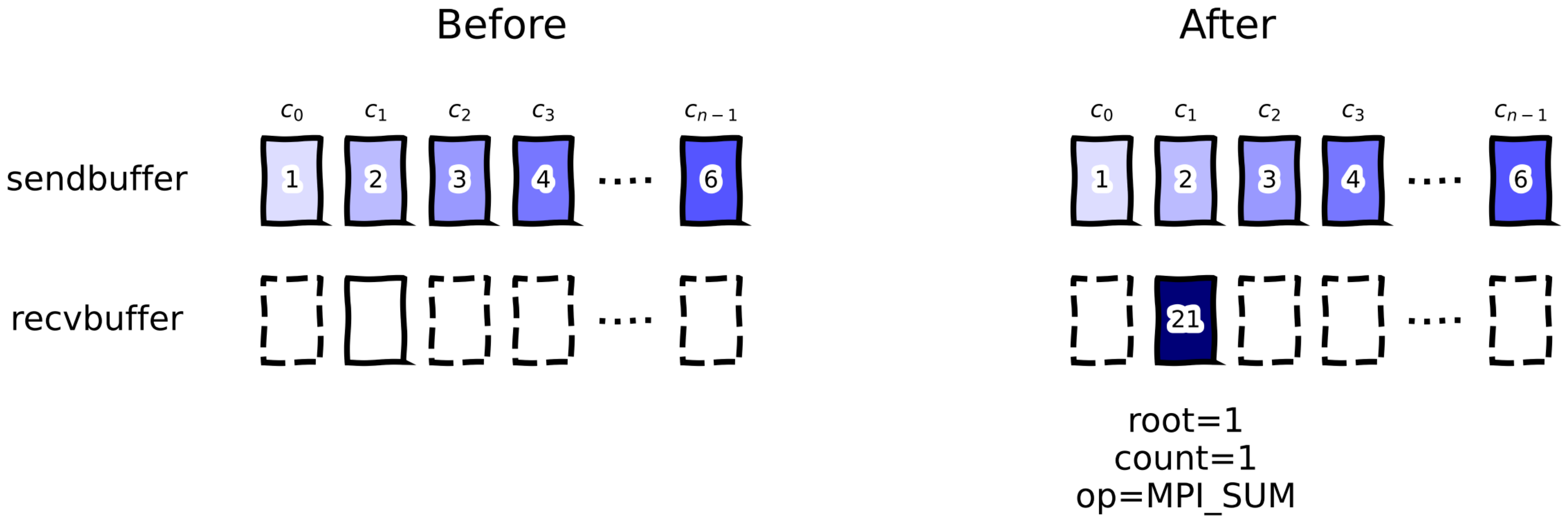
```
MPI_Barrier(MPI_Comm comm);
```



- during collective communications such as reductions

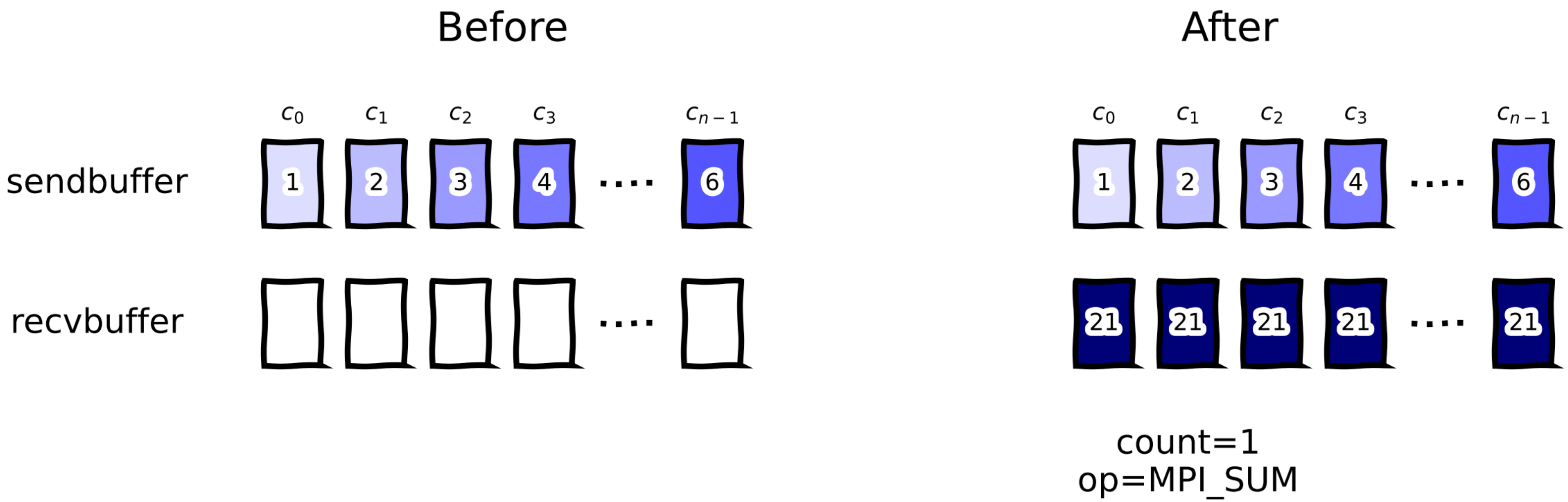
Synchronizations : MPI_Reduce()

```
int MPI_Reduce(void* sendbuffer, void * recvbuffer, int count,
               MPI_Datatype datatype, MPI_Op op, int root,
               MPI_Comm comm);
```



Synchronizations : MPI_Allreduce()

```
int MPI_Allreduce(void* sendbuffer, void * recvbuffer, int count,
                  MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);
```



Synchronizations : Returning to the Case study case

To sum all the values in the distributed vector, we will use reduction.

```
long localsum = 0.0
long globalsum ;
for (int i = 0; i < xlocsize; ++i)
  localsum += x[i] ;

MPI_Allreduce(&locsum,&globsum,1,MPI_LONG,MPI_SUM,MPI_COMM_WORLD);
```

Execution on a Single Core

To execute a portion of code on a single core, you can simply use an `if` statement.

In our **study case**, if we want to display the global sum, we would write:

```
...
if (rank == 0)  // rank 0 is always present
{
  cout << "The sum is: " << globalsum << endl;
}
```

MPI Parallelization

- The **main task** is **data partitioning** (seen here in the simple case of a vector)
- **Exercise:** Parallelize the dot product $dp = \sum_i x_i y_i$
  1. with a vector size that is a multiple of the number of cores used
  2. with an arbitrary vector size