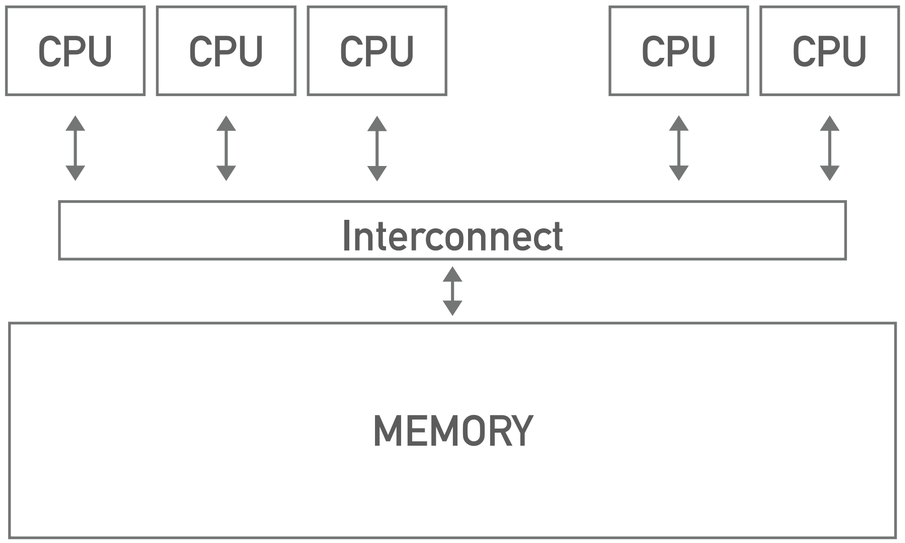
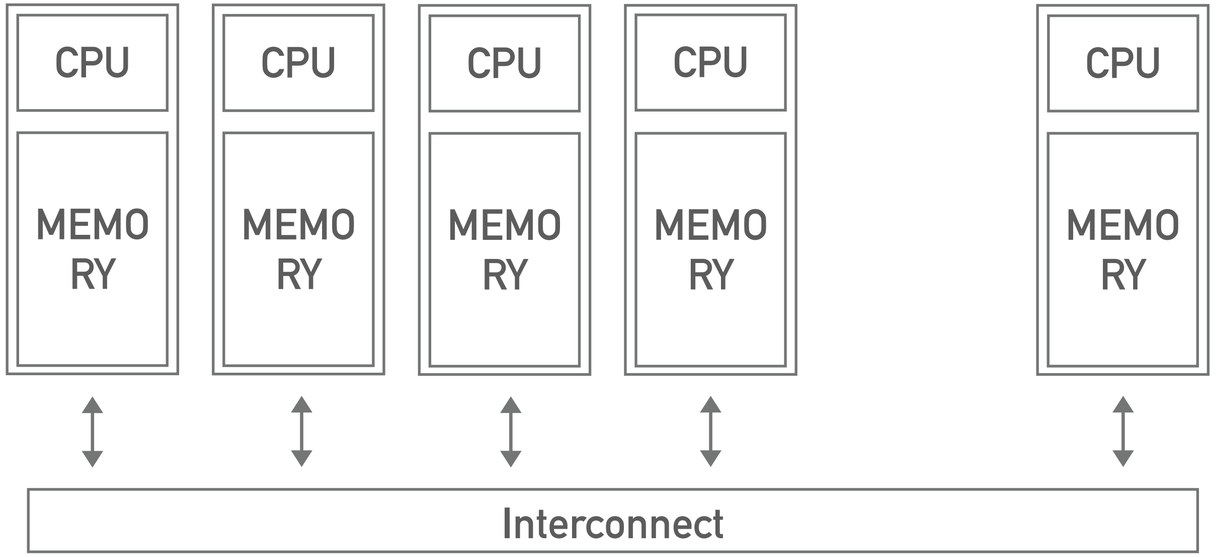
ME 471/571

Distributed Memory and Message Passing Interface

PARALLEL PROGRAMMING MODELS

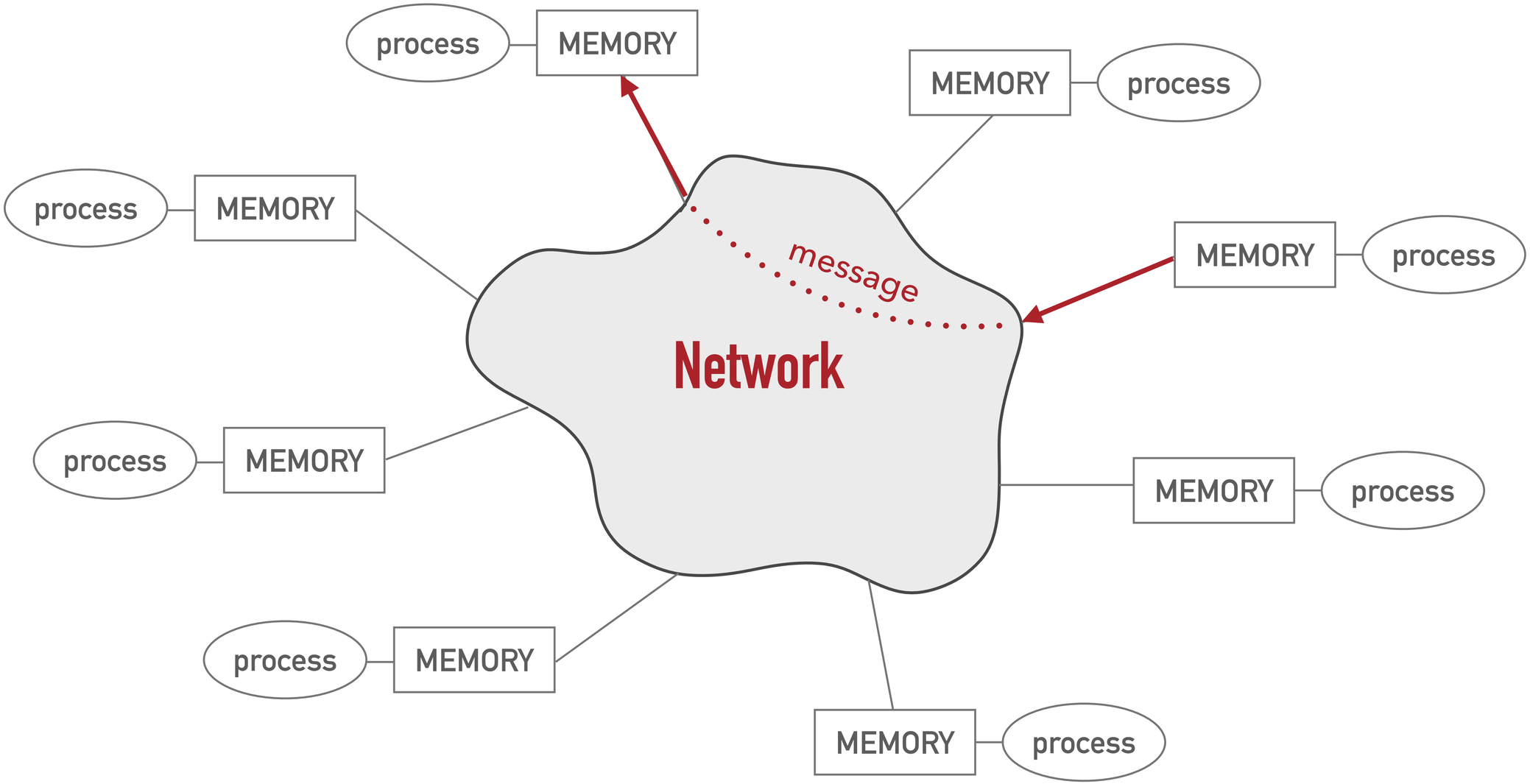


SHARED MEMORY



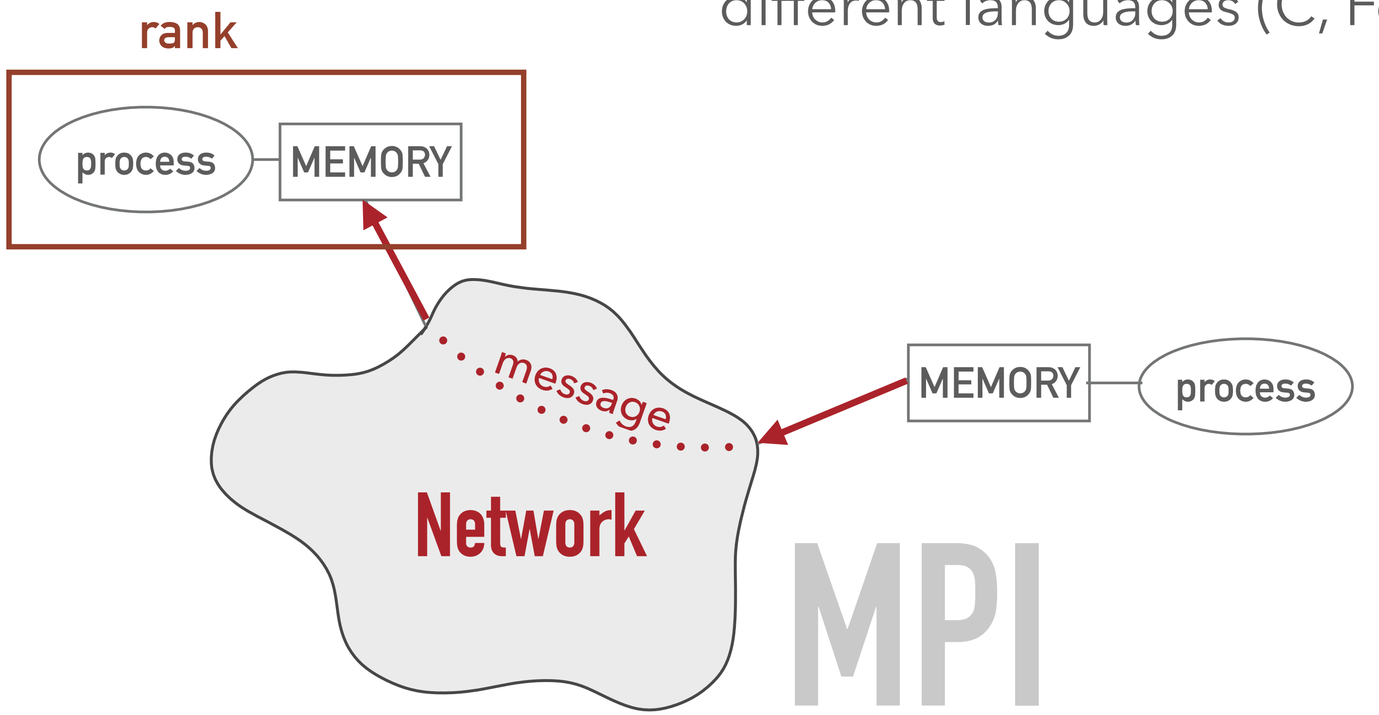
DISTRIBUTED MEMORY

MESSAGE PASSING MODEL



MESSAGE PASSING INTERFACE

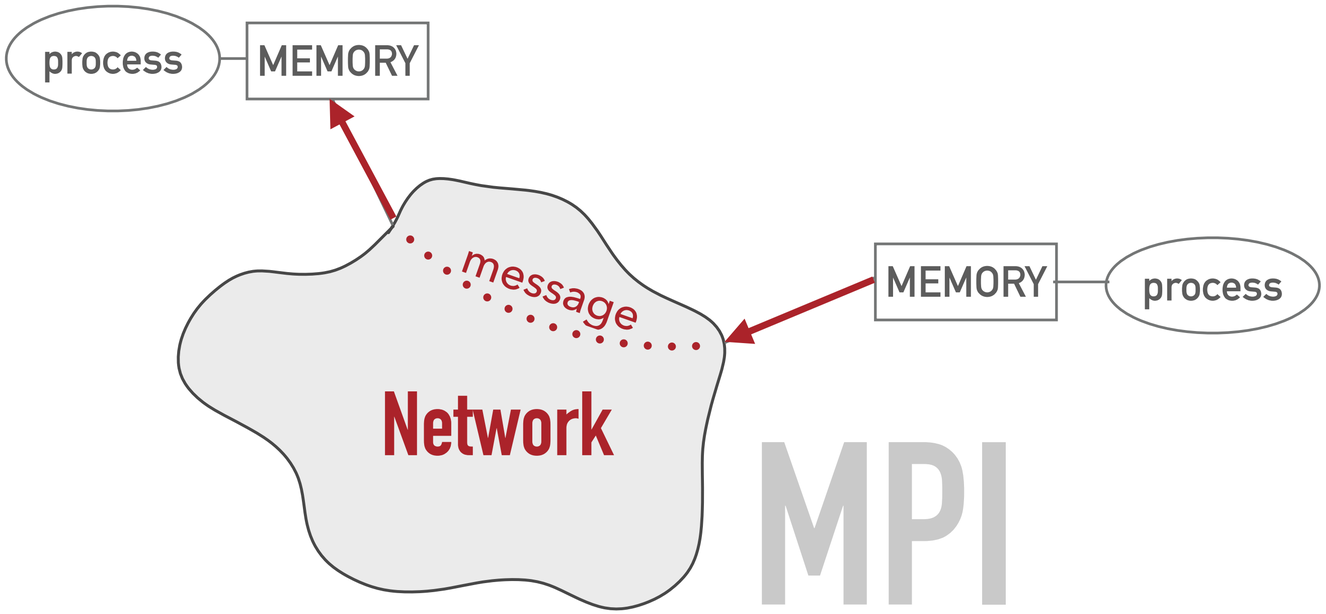
- MPI is a **library** which specifies names and results of subroutines (functions) that allow **passing data** between **processes** (**ranks**) using **messages**.
- MPI **is not** a programming language. It can be implemented in different languages (C, Fortran, Python)



- There are different implementations of MPI (MPICH, OpenMPI), which can perform better or worse on certain systems

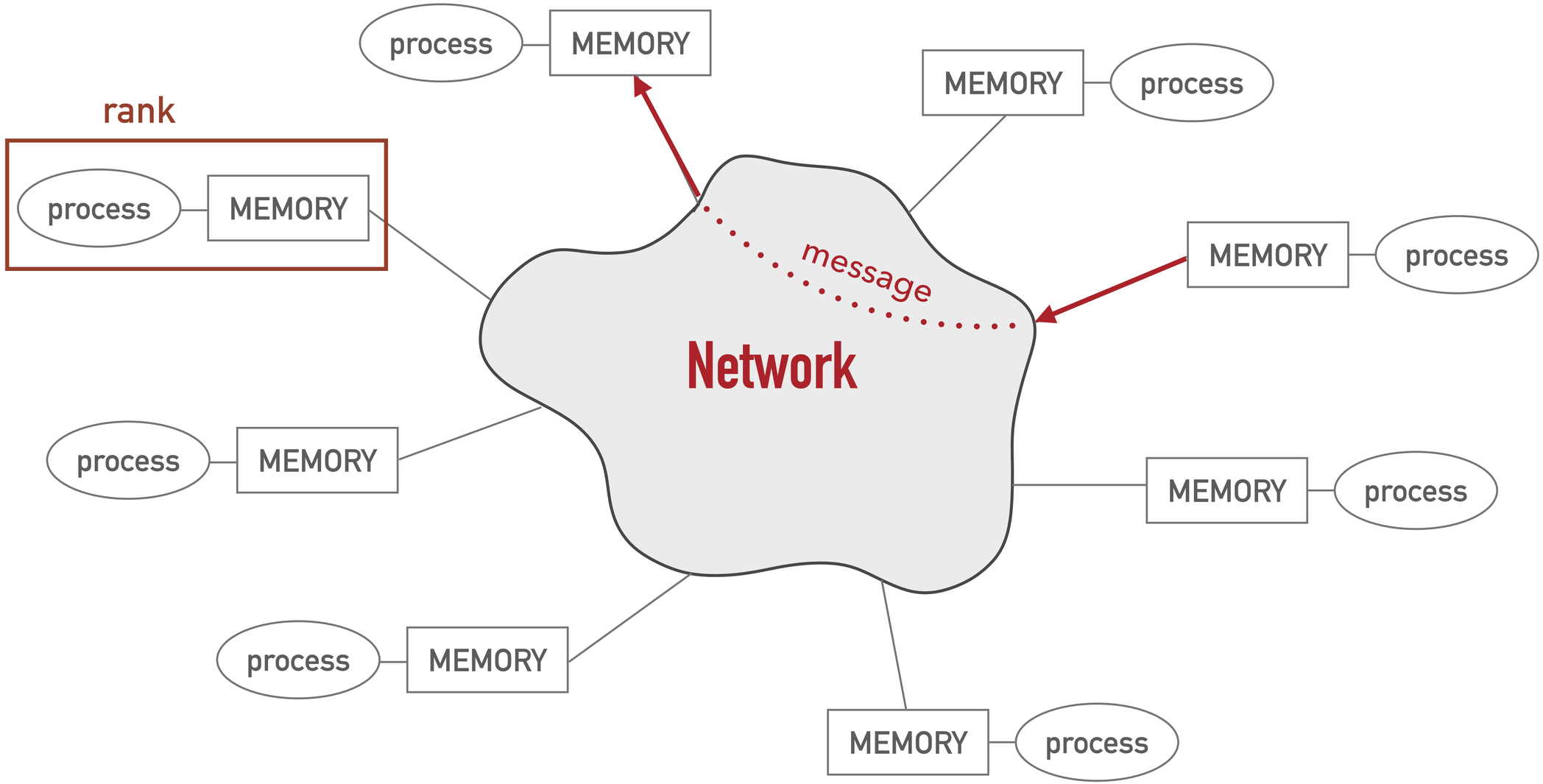
MESSAGE PASSING INTERFACE

- MPI is **universal** - can run on (almost) any system, from supercomputers to clusters of PCs, multicore CPUs, as well as shared memory machines
- MPI is **expressive** - can implement any algorithm invented by a programmer within the message passing model

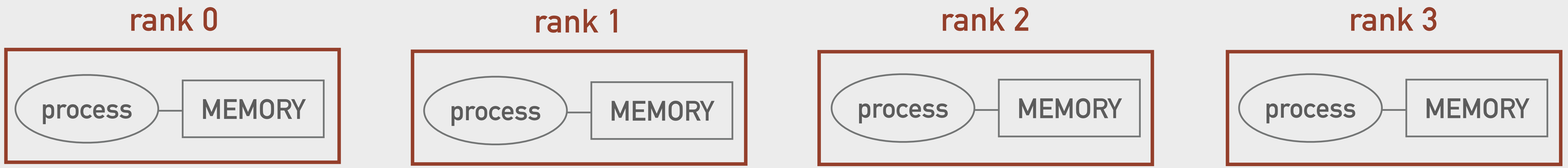


- MPI is (relatively) **easy do debug** due to well defined memory ownership

MESSAGE PASSING MODEL



MESSAGE PASSING MODEL



Communicator

- each process has a unique **rank** (number) within communicator
- communicator **size** is the number of ranks (processes) it contains
- ranks can exchange messages within a communicator

`MPI_COMM_WORLD` - default communicator containing all available ranks

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

FUNCTIONAL PARALLELISM

*Different processes perform **different** tasks.*

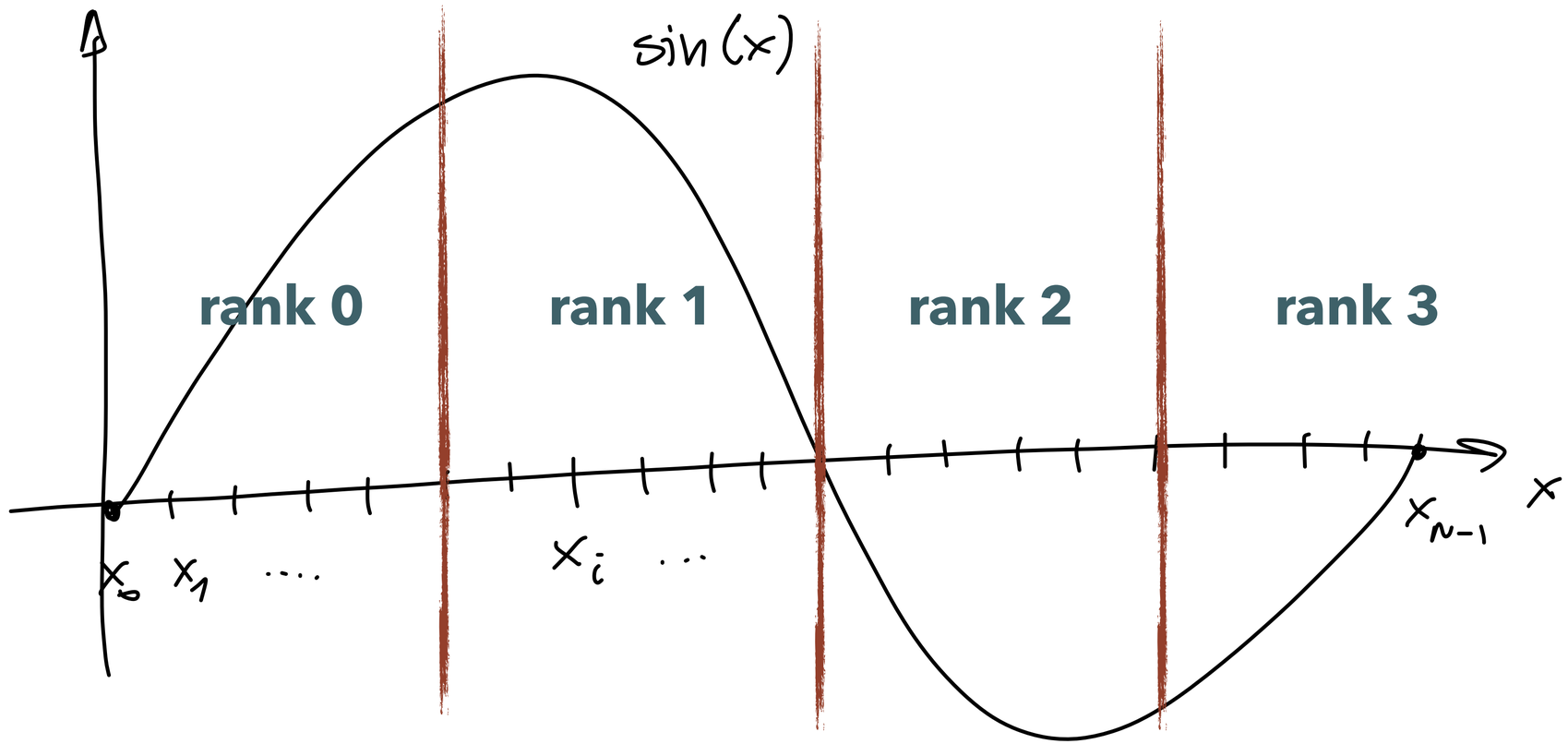
rank 0	*runs ocean model*
rank 1	*runs atmosphere model*
rank 2	*runs ice model*
rank 3	*runs land model*

DATA PARALLELISM

*Different processes perform **the same** tasks on different data.*

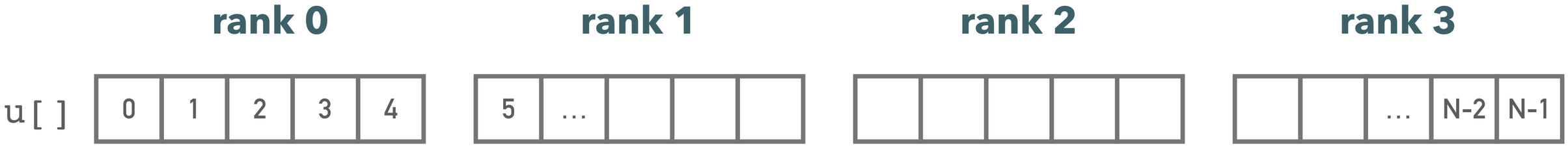
rank 0	*computes derivative on first 1/4 of data*
rank 1	*computes derivative on second 1/4 of data*
rank 2	*computes derivative on third 1/4 of data*
rank 3	*computes derivative on fourth 1/4 of data*

Example of data parallelism - compute the derivative of sin(x)



Divide and Conquer: *We can divide the work among available ranks, where each rank performs a fraction of the work*

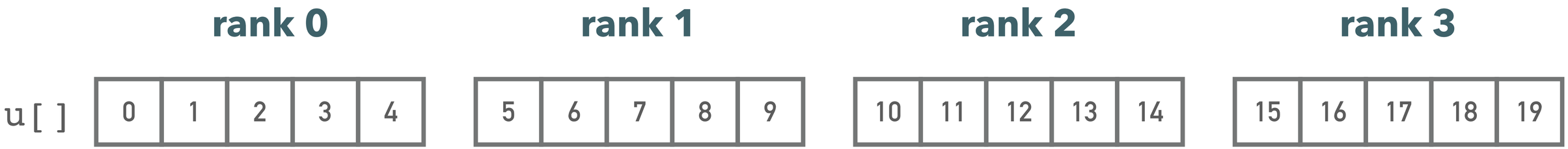

u[ ]
0
1
2
3
4
5
...
...
N-2
N-1



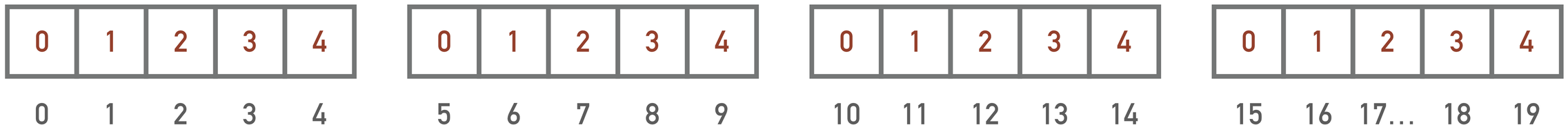

```
N = 20
```
we have $N = 20$ points in this example

```
nproc = 4
```
let's say we have 4 ranks in our communicator

```
N_loc = N/nproc
```
that gives 5 points per rank

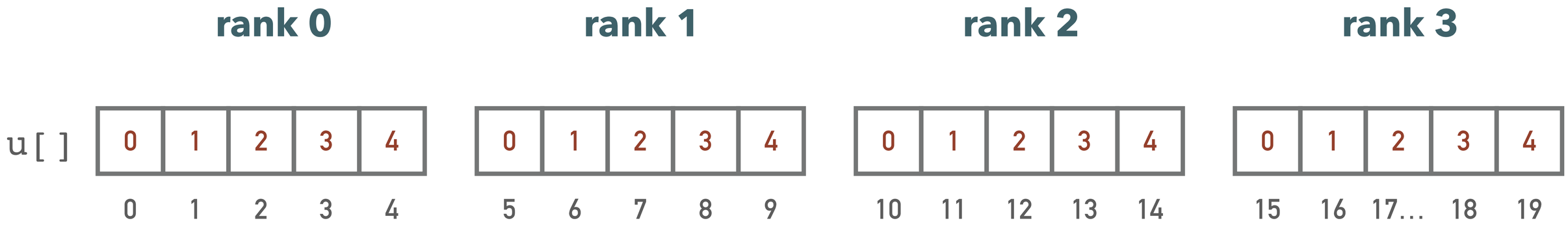


In the serial code points are numbered with a ***global index*** *$i = 0,\ldots N\text{-}1$*



Each rank processes only a subset of those points with different global indices,

but numbers them with their own ***local index****: i_loc=0,…,N_loc-1*




```
N_loc = N/nproc

i = N_loc * irank + i_loc
```