

MPI in Small Bites

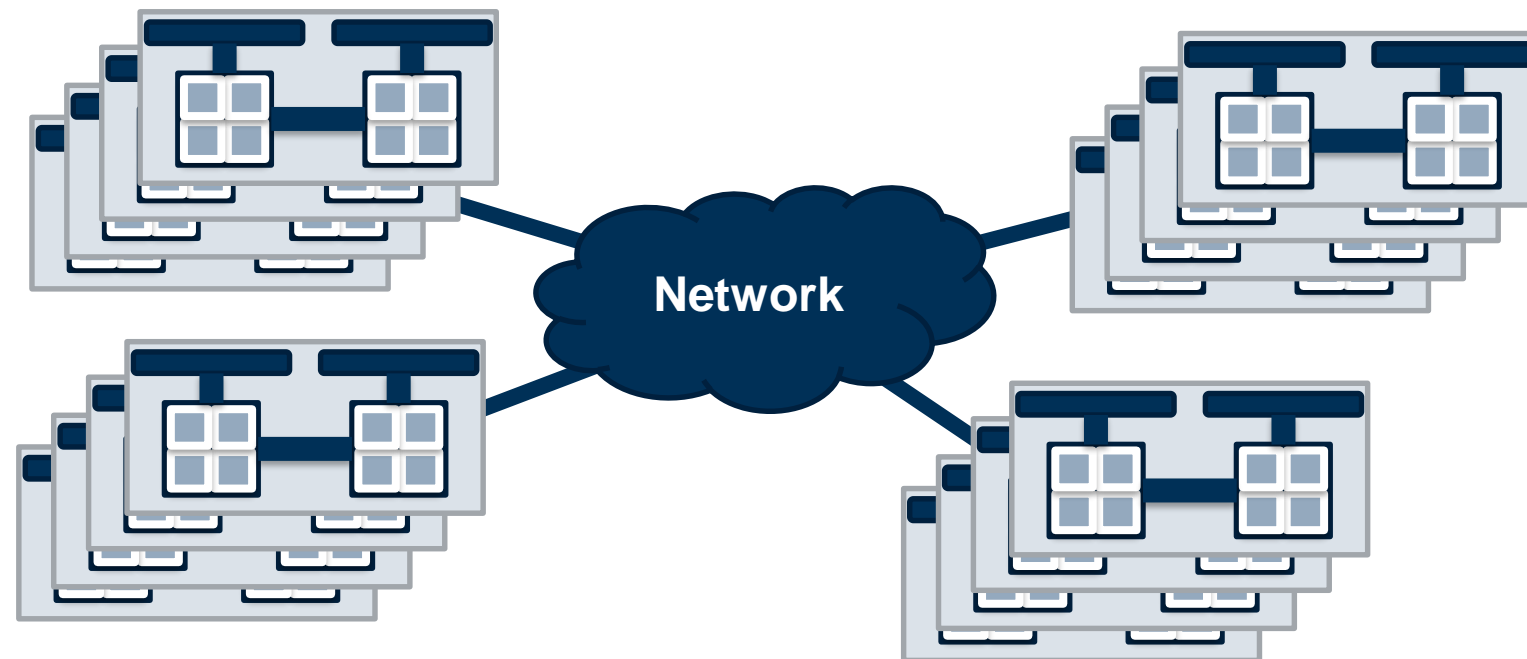
HPC.NRW Competence Network

Message Passing Basics

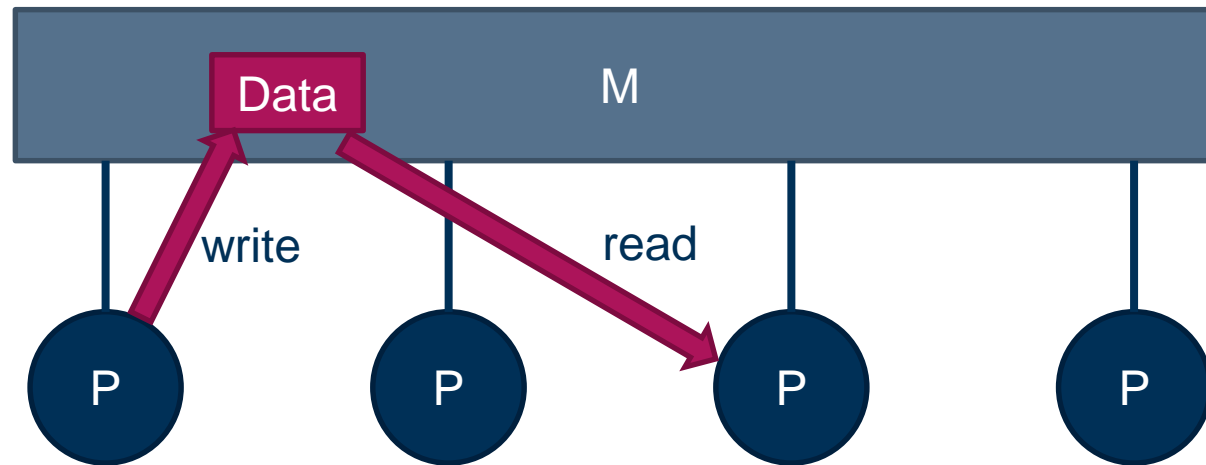
HPC.NRW Competence Network

MPI in Small Bites

- Clusters
 - HPC market dominated by distributed memory *multi-computers*
 - Nodes have no direct access to other nodes' memory
 - Nodes run a separate copy of the (possibly stripped down) OS

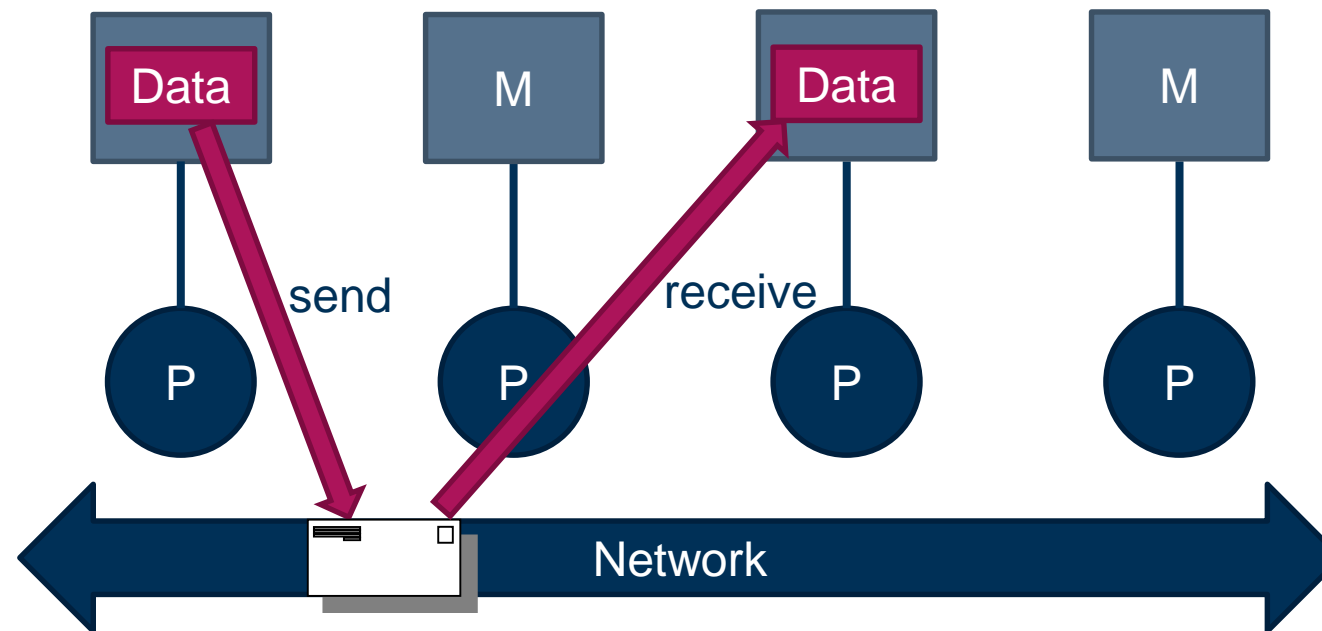


- All processing elements (P) have direct access to the main memory block (M)



- Data exchange is achieved through read/write operations on shared variables located in the global address space

- Each processing element (P) has its separate main memory block (M)



- Data exchange is achieved through message passing over the network
- Two distinct copies of the data at sender and receiver

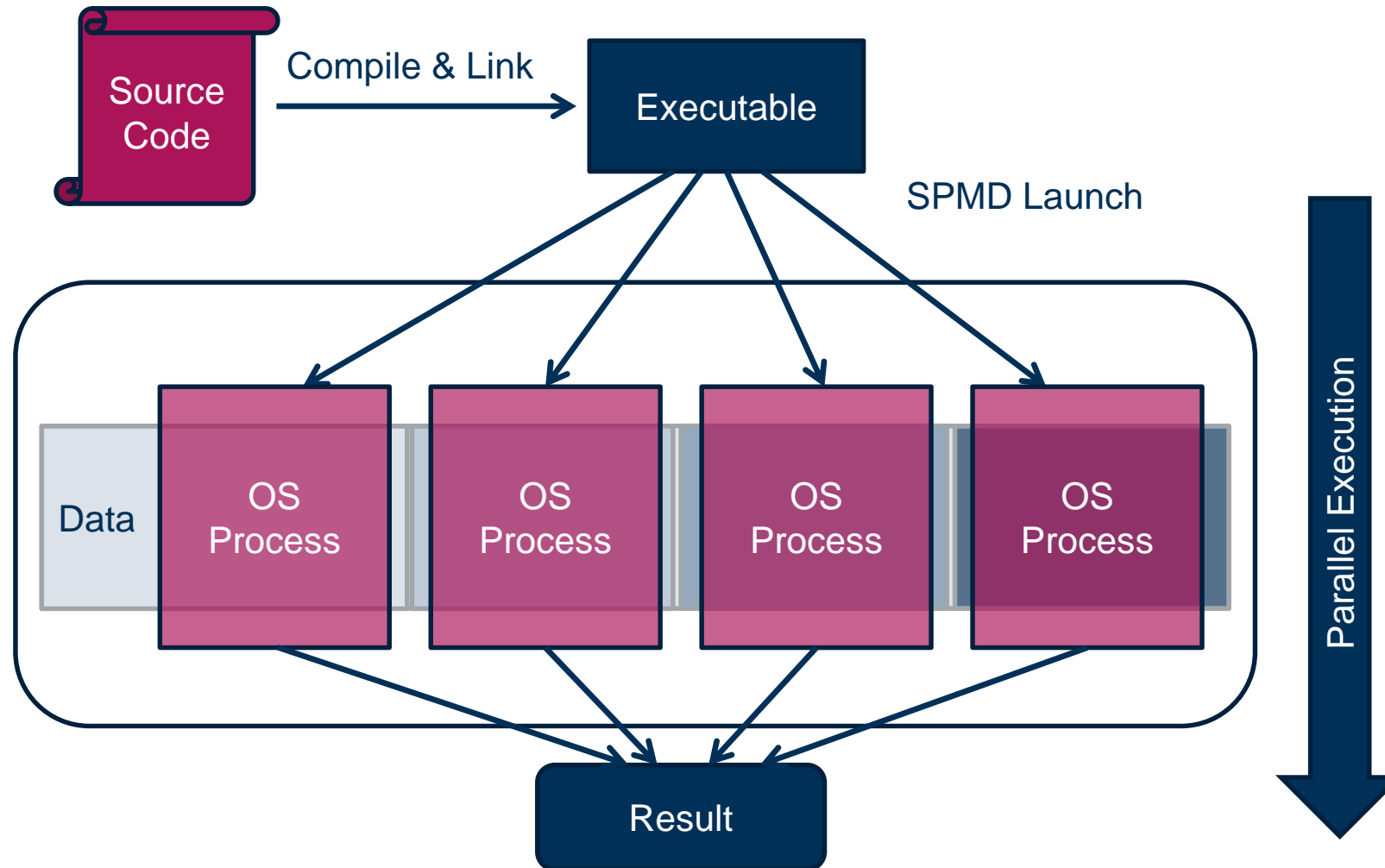
- Each processing element (P) works on a separate main memory block (M)
 - Processes typically have own (virtual) address space
- Data exchange is achieved through message passing (data copy)
- Message passing can be either explicit (MPI) or implicit (PGAS)
 - PGAS provides shared-memory abstractions, but may use message passing at runtime.
- No shared variables
 - No data races
- Implicit synchronisation with message exchange

- A process is a running in-memory instance of an executable file
 - Executable code, e.g., binary machine instructions
 - One or more threads of execution sharing memory address space
 - Memory: data, heap, stack, processor state (CPU registers and flags)
 - Operating system context (e.g., signals, I/O handles, etc.)
 - Process ID (PID)
- Isolation and protection
 - Without OS support, a process cannot ...
 - interoperate with other processes
 - access their context (even on the same node)
 - No direct inter-process data exchange (isolated/virtual address spaces)
 - No direct inter-process synchronisation

- Abstractions ease programming and understanding
- Single Program Multiple Data
 - Single Program (executable)
 - Multiple (different parts of) Data
- Multiple instruction flows (instances)
 - E.g., threads (OpenMP, Pthreads) and/or processes (MPI)
 - Unique instance IDs can be used for flow control

```
if (myID == specificID)
{
    do_something();
}
else
{
    do_something_different();
}
```


SPMD Program Lifecycle – multiple processes (e.g. MPI)



1. Provide dynamic identification of all peers
 - Who else is also working on this problem?
2. Provide robust mechanisms to exchange data
 - Whom to send data to / From whom to receive the data?
 - How much data?
 - What kind of data?
 - Has the data arrived?
3. Provide synchronisation mechanisms
 - Have all processes reached same point in the program execution flow?
4. Provide methods to launch and control a set of processes
 - How do we start multiple processes and get them to work together?
5. Portability

- Message Passing Interface
 - The de-facto standard API for explicit message passing nowadays
 - A moderately large standard (v3.1 has 868 pages)
 - Maintained by the non-profit Message Passing Interface Forum: <https://www.mpi-forum.org/>
- Many specific implementations of the MPI standard
 - Open MPI, MPICH, Intel MPI, MVAPICH, MS-MPI, etc.
- Application Programming Interface (API) to exchange messages in distributed memory
- MPI does not specify ABI compatibility!
 - Applications need to be compiled against a specific implementation

- Language-independent specification (LIS)
 - Standard bindings for C and Fortran
 - Specific function prototypes / interfaces
 - Non-standard bindings for other languages exist:
 - C++ [Boost.MPI](#), [MPL](#)
 - Java [Open MPI](#), [MPJ Express](#)
 - Python [mpi4py](#)
- No extension of the base language
 - Set of libraries, header files, and auxiliary programs
 - Standard compilers can be used to compile and link a program

- The MPI Forum document archive (free standards for everyone!)
 - <http://www.mpi-forum.org/docs/>
- Manual pages
 - `man MPI_Xxx_yyy_zzz` (for all MPI calls)
- Tutorials (like this one)