Distributed Memory

Too many people in the same room

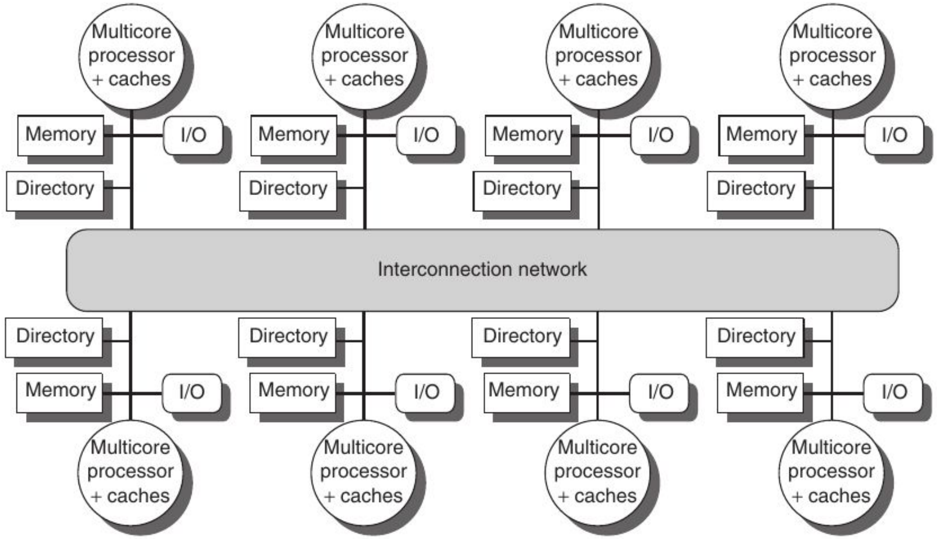
Source material

- J. L. Hennessy, D. A. Patterson, and K. Asanović, *Computer Architecture: A Quantitative Approach*, 5th ed. Waltham, MA: Morgan Kaufmann/Elsevier, 2012.
    - Chapter 6: Warehouse-Scale Computers to Exploit Request-Level and Data-Level Parallelism
    - Appendix F: Interconnection Networks
- D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface: RISC-V Edition*. Cambridge, MA: Morgan Kaufmann, 2018.
    - Chapter 6: Parallel Processors from Client to Cloud

- R. Copetti, *Architecture of Consoles - A practical analysis*

Non-Uniform Memory Access (NUMA)

- Number of processors in a system increase
  - More petitions to/from memory
  - More network contention
  - Less bandwidth per processor
- Break-up memory into slices
  - Each slices is positioned close to a processor
  - Access symmetry is broken
  - Hence, *Non-Uniform Memory Access*
- Threads can still share the memory space

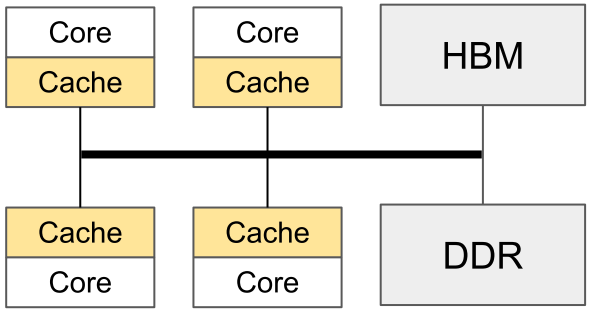


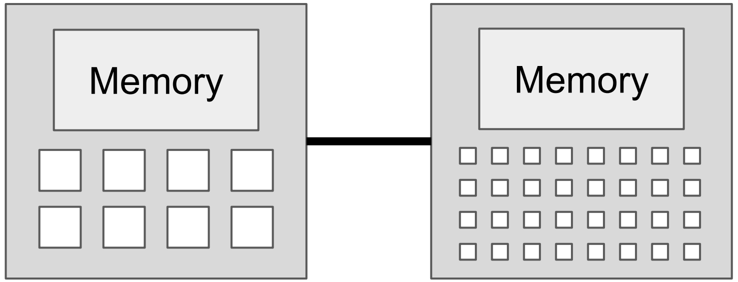
- OS knows NUMA topology and provides mechanisms to work with it
  - When software is capable of distinguishing NUMA regions, we call it *NUMA-aware*

Heterogeneous memories

- Complex systems with different types of memory technologies
  - Eg: DDR + HBM
  - Eg: Host - Device

- OS provides software layer that can be
  - *Implicit* (or transparent): Software is not aware of which physical memory is using
  - *Explicit*: Software must specify which physical memory wants to access





Memories from a distant land

- As systems become bigger and processors have to access farther memories, the cost of maintaining consistency and coherence becomes too high
- Hence, the address space is broken into isolated partitions
  - Threads from the same process still share memory address space
  - Processes, which are managed by the OS, do not share memory address space
- This is what we commonly refer to as ***Distributed Memory***
- New challenges: How do processes share data?
  - A common way to do so is with *Message Passing*
- New possibilities: Programs are not bound by the memory capacity of one node
  - Programs be multi-process and leverage memory of multiple nodes

Hardware support to move data around

- Moving data between locations in a distributed memory system takes time

- Naive approach: have the processor read the data from its memory and send it through the network to the remote memory
  - If the amount of data is very big it will require multiple transactions
  - Even if data is small, the transfers could be very frequent
  - Result: the processor spends most of its cycles moving data from memory locations

- Workaround: implement a dedicated hardware component to move data around
  - This component is called *Direct Memory Access* (DMA) engine
  - Give it a source, the size of the transfer, a destination, and kick it into action
  - When the DMA transfer is done, it will notify the processor with an interruption

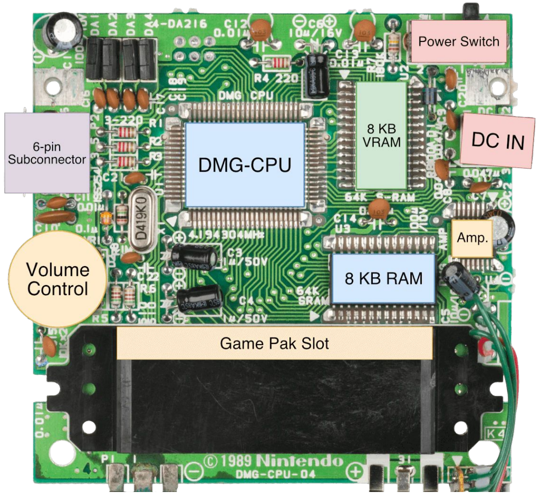
Interfacing with a DMA engine

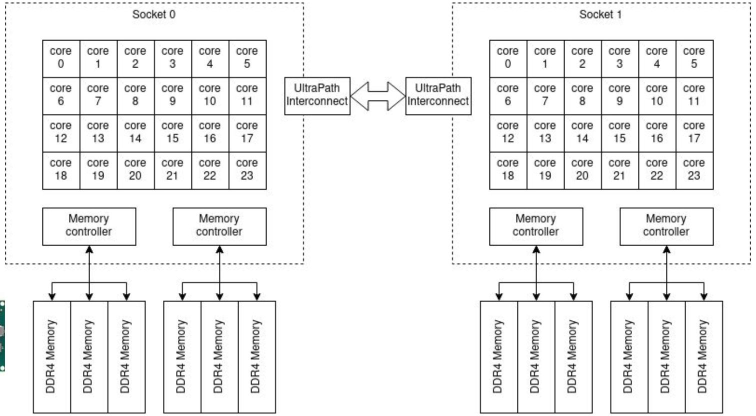
Embedded systems and low-level code

- Write to memory-mapped registers
- Implement response to DMA interrupt

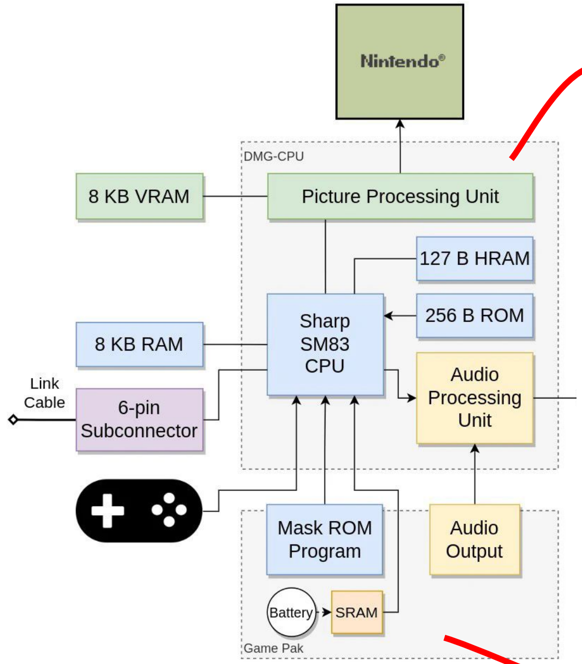
High-level code

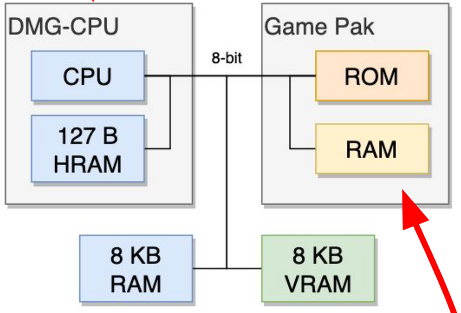
- Completely transparent
  - OS and kernel management (`malloc`)
- Programmer-friendly API
  - `cudamalloc`
  - `MPI_Send, MPI_Broadcast`

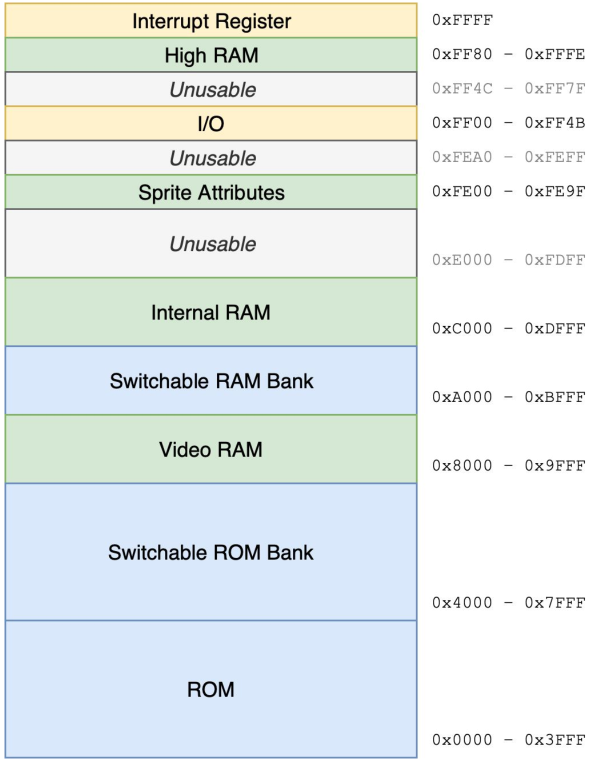




Personal favorite example (not HPC)







Interrupt Register	0xFFFF
High RAM	0xFF80 – 0xFFFE
Unusable	0xFF4C – 0xFF7F
I/O	0xFF00 – 0xFF4B
Unusable	0xFEA0 – 0xFEFF
Sprite Attributes	0xFE00 – 0xFE9F
Unusable	0xE000 – 0xFDFF
Internal RAM	0xC000 – 0xDFFF
Switchable RAM Bank	0xA000 – 0xBFFF
Video RAM	0x8000 – 0x9FFF
Switchable ROM Bank	0x4000 – 0x7FFF
ROM	0x0000 – 0x3FFF

Distributed Memory Programming models

- *Message Passing Interface (MPI)*
  - Very prevalent in CPU-based HPC
  - Explicit message passing with high-level API

- *Compute Unified Device Architecture (CUDA)*
  - Very common in NVIDIA GPUs
  - Explicit communication with high-level API

- *OpenMP Device Offload*
  - Extension of the shared-memory programming model
  - Implicit communication with pragmas

- *Partitioned Global Address Space (PGAS)*
  - Software emulates shared address space across multiple distributed memory nodes
  - Examples: UPC++, Global Arrays

MPI Programming Model

Calling for help

Source material

- MPI: A Message-Passing Interface Standard
- MPI cheatsheet

MPI Programming Model

- Distributed-memory programming model
- Process level parallelism
- Explicit process communication

- Programming via MPI calls

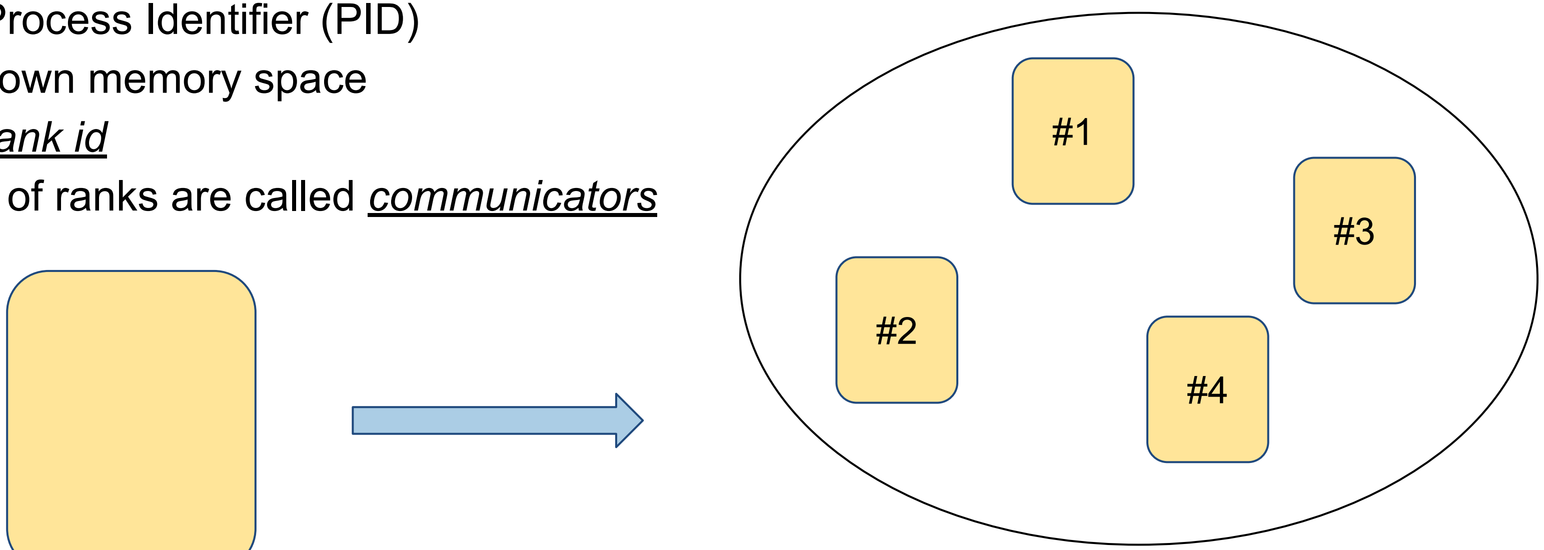
- Code is written with MPI execution in mind from the very start

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
  int nranks, myrank;

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &nranks);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  if (myrank == 0)
  {
    printf("Master rank prints\n");
  }

  MPI_Finalize();
}
```

Vocabulary

Process ≃ Rank

- Program in execution
- Has a Process Identifier (PID)
- Has its own memory space
- Has a *rank id*
- Groups of ranks are called *communicators*



Process

`MPI_COMM_WORLD` communicator

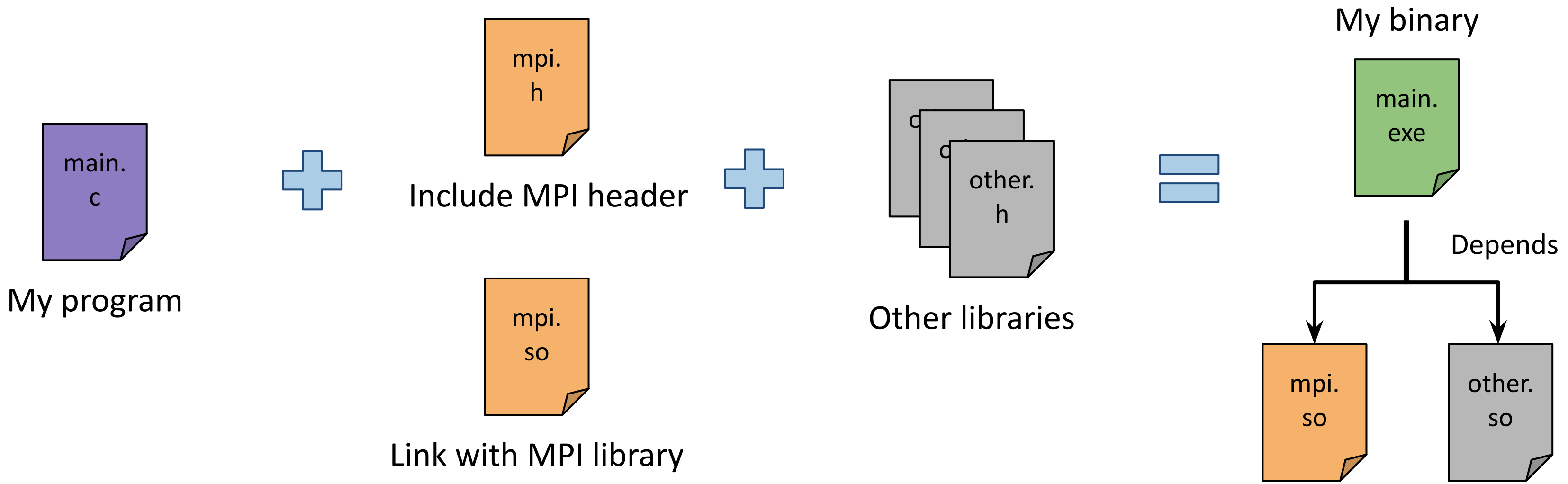
MPI standards and libraries

- MPI is a standard
  - `MPI_Init, MPI_Finalize`
  - `MPI_Barrier, MPI_Allreduce`

- There are different library implementations of the same standard
  - OpenMPI, IntelMPI, MPICH
  - ⚠️ **Disclaimer:** Do not mix the standard OpenMP with the library OpenMPI !!!

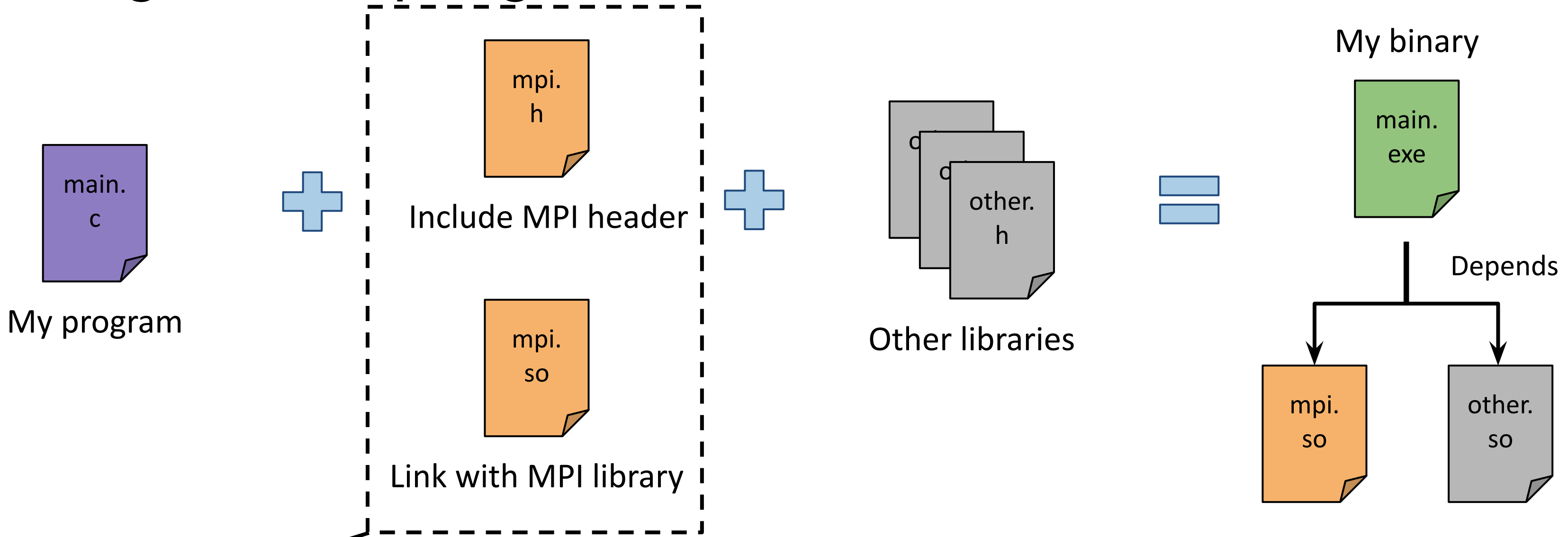
- Different libraries implement the same standard with different strategies

Building an MPI program



```
icc
   main.c -o main
   <other libraries for my program...>
   -I/apps/INTEL/2017.4/impi/2017.3.196/intel64/include
   -L/apps/INTEL/2017.4/impi/2017.3.196/intel64/lib
   -lmpi
```

Building an MPI program



```
mpicc
  main.c -o main
  <other libraries for my program...>
```

MPI parameters and SLURM

- Parameters vary depending on the MPI library
- All MPI libraries have one parameter in common: number of ranks (`-np`)

- Depending on SLURM configuration the number of ranks
    - Will not be set at all (user has to set it)
    - Will be set based on the allocation option `--ntasks`
    - Will be set to a default value (usually the total number of cores)

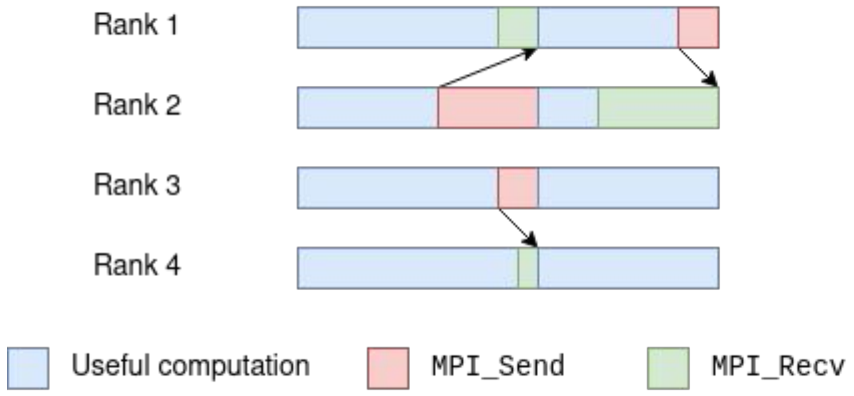
- Best practices
    - If allocating via SLURM, always set `--ntasks`

Who participates in an MPI communication

Point-to-point

- Pair of processes

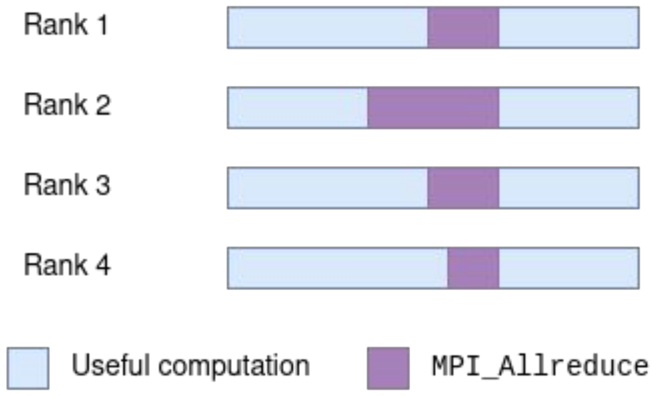
- `MPI_Send(...)`
- `MPI_Recv(...)`



Collective

- All processes in a communicator

- `MPI_Barrier(...)`
- `MPI_Allreduce(...)`

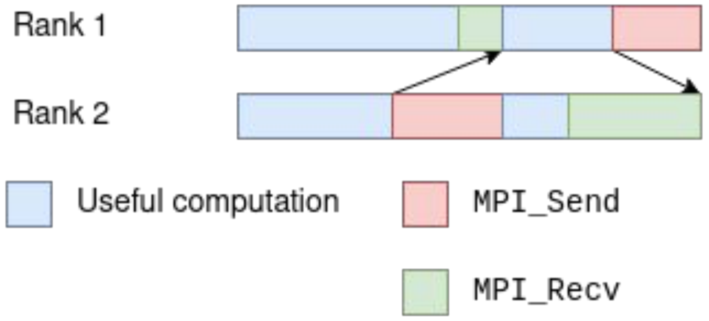


How patient are we during an MPI communication

Blocking

- Wait until communication has finished

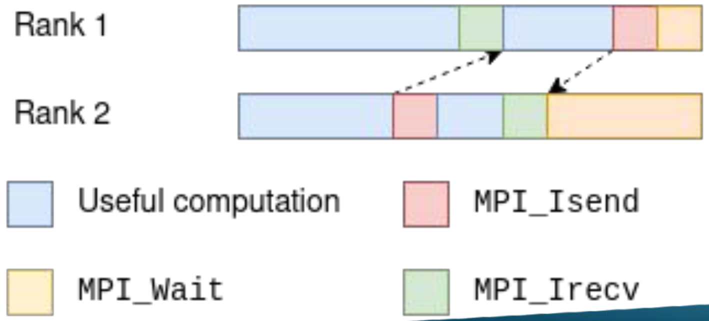
- `MPI_Send(...)`
- `MPI_Recv(...)`



Non-blocking

- Continue until you really need the data

- `MPI_Isend(...)`
- `MPI_IRecv(...)`
- `MPI_Wait(...)`


Rank 1
Rank 2
Useful computation
MPI_Isend
MPI_Wait
MPI_Irecv