# Introduction to MPI

2020/10/19

之江实验室  ZHEJIANG LAB

# Outline

I. **Parallel computing**

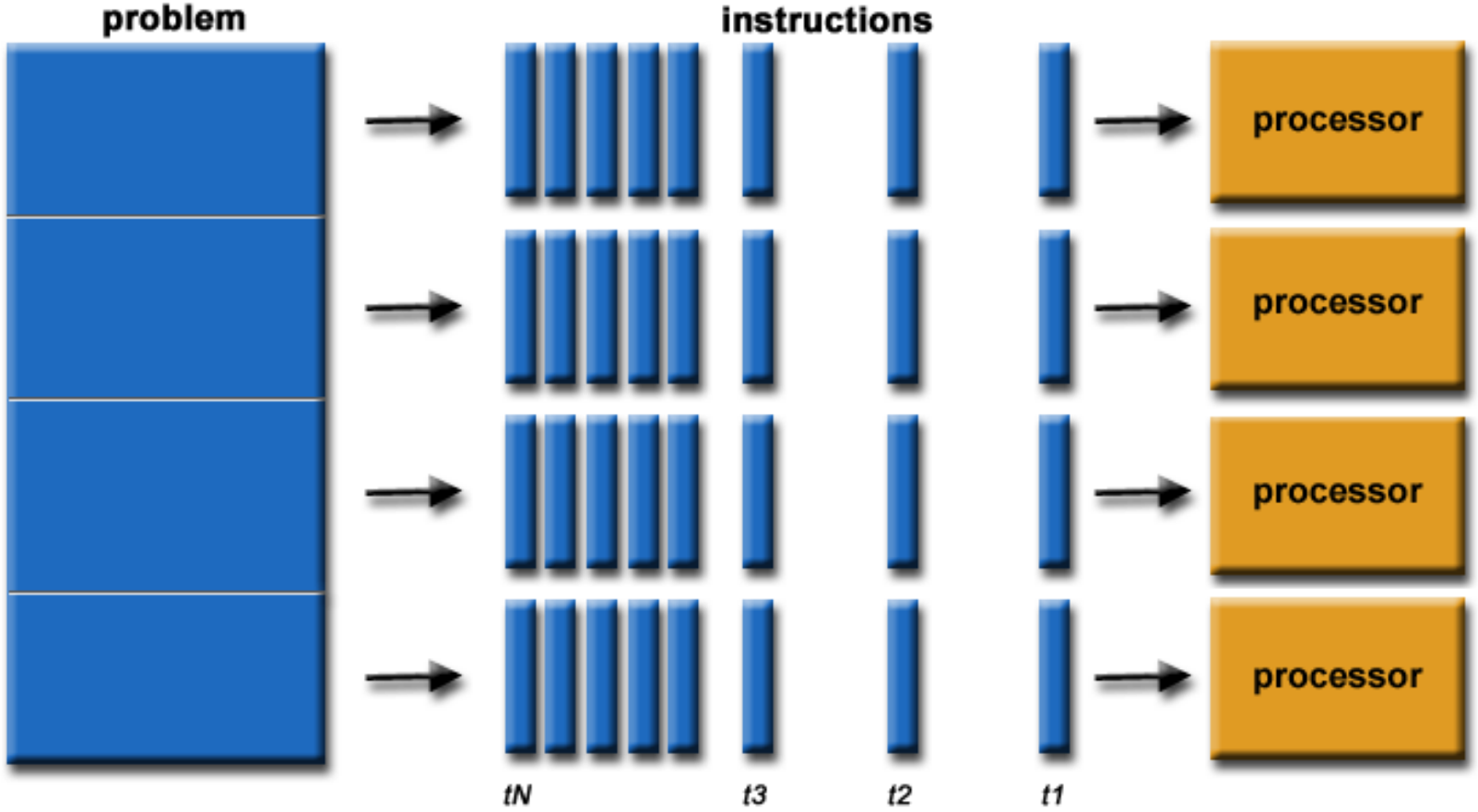
II. Basic Concepts of MPI

III. Point-to-point communication

IV. Collective communication

# What is parallel computing

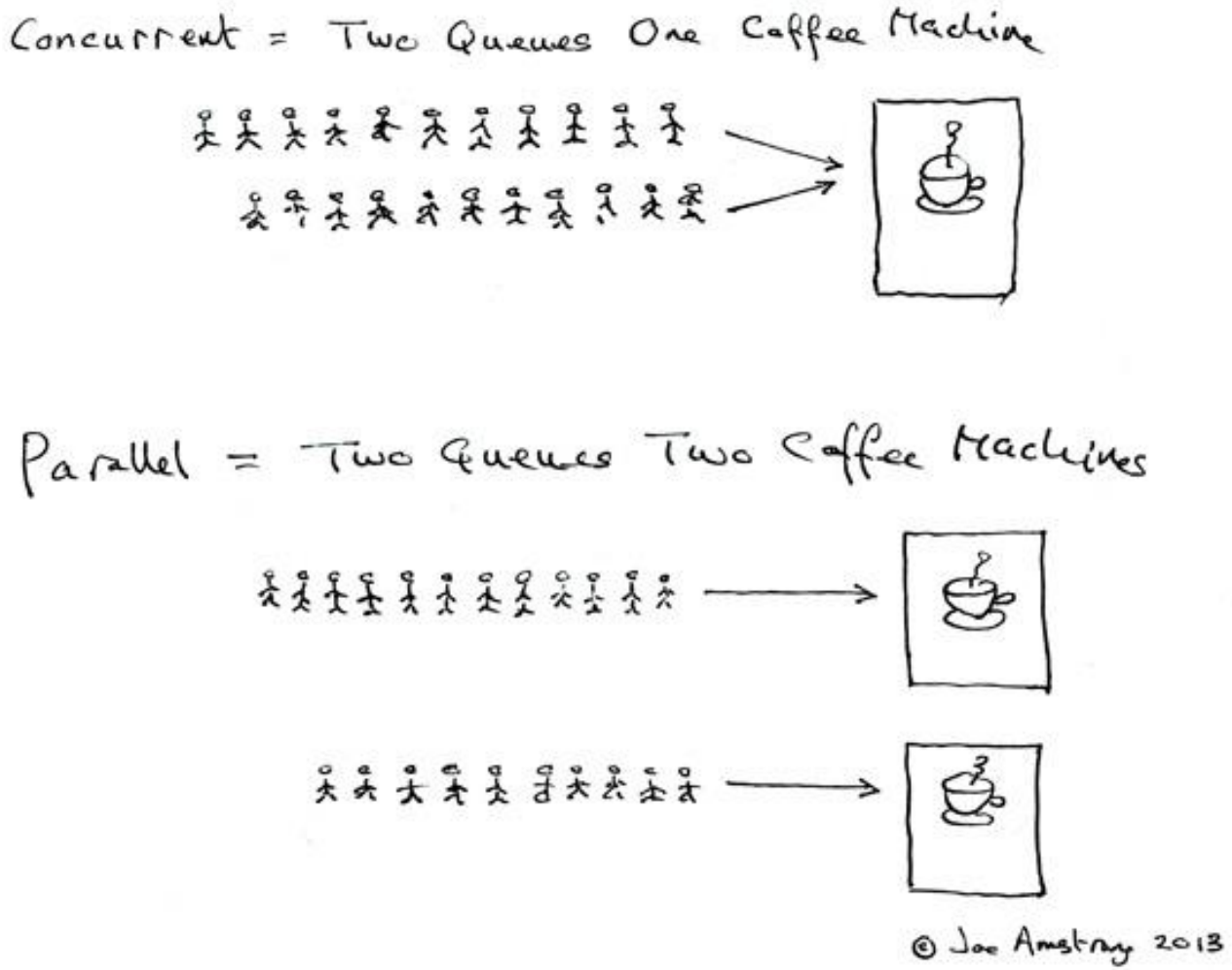
Parallel computing is a type of computing architecture in which several processors execute or process an application or computation simultaneously.



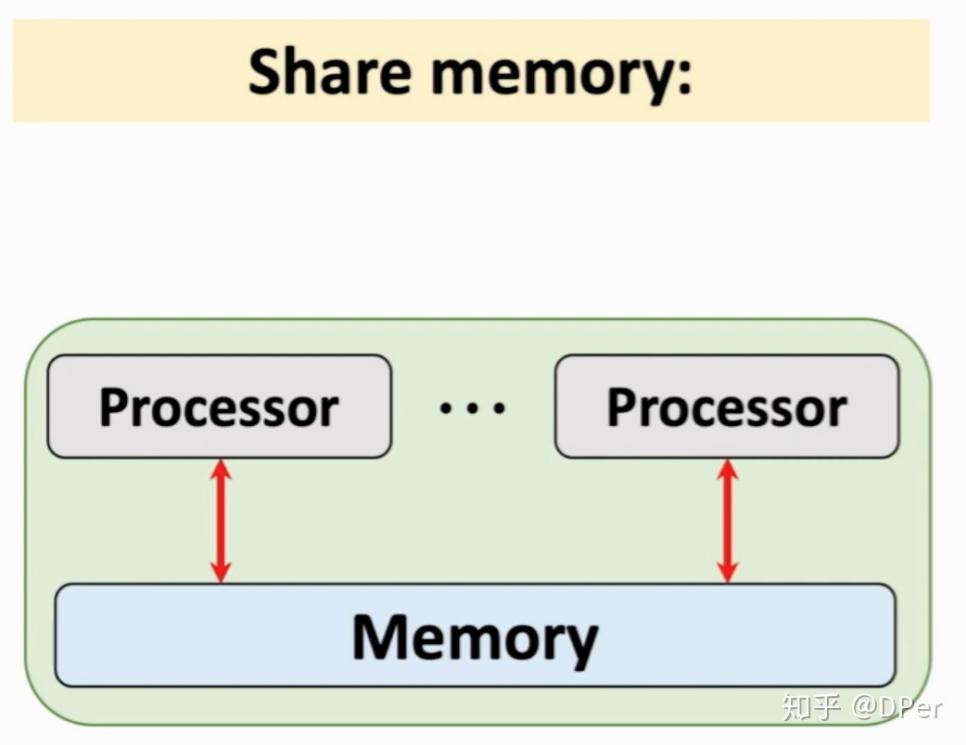
# Why parallel computing for DL?

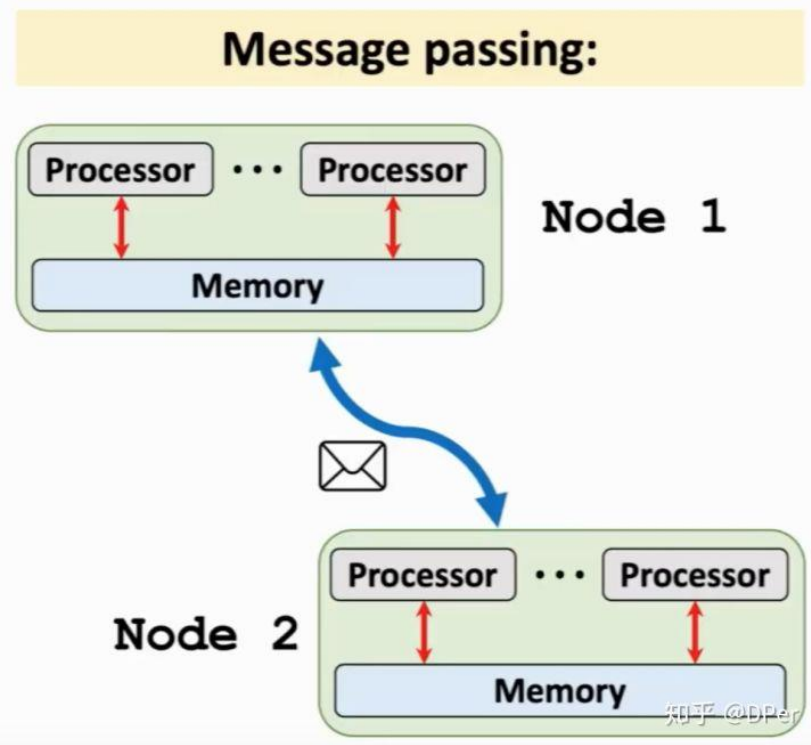
- Deep learning models are big: ResNet-50 has 25M parameters.
- Big models are trained on big data, e.g., ImageNet has 14M images.
- Big model + big data ➔ Big computation cost.
- Example: Training ResNet-50 on ImageNet (run 90-epochs) ImageNet using a single NVIDIA M40 GPU takes 14 days.
- Parallel computing: using multiple processors to make the computation faster (in terms of wall-clock time.)

# Concurrent vs. Parallel



# Parallel communication model




OpenMP：Open Multi-Processing

MPI：Message Passing Interface

# Outline

I. Parallel computing

**II. Basic Concepts of MPI**

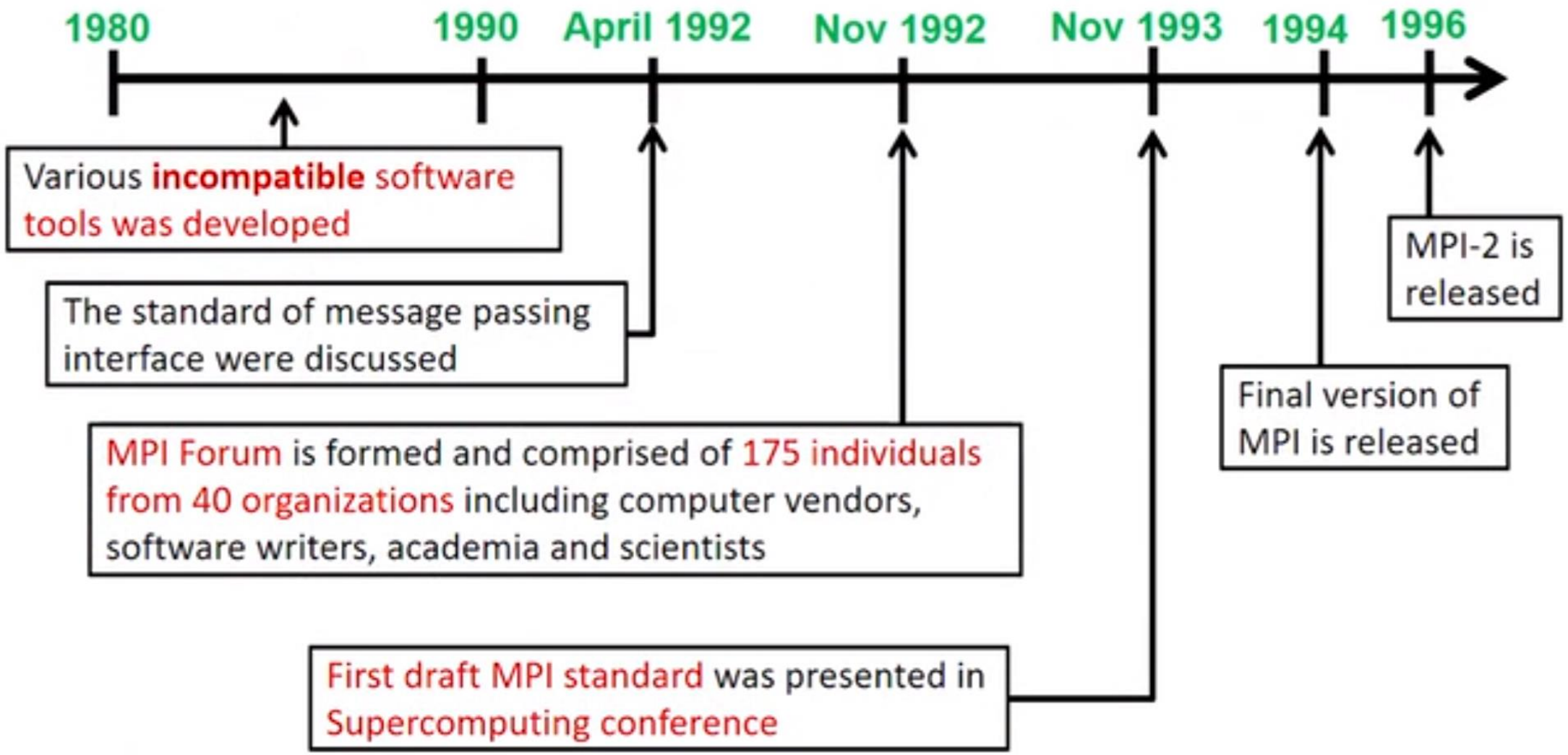
III. Point-to-point communication

IV. Collective communication

# What is MPI

- MPI: Message Passing Interface
  - MPI is a message-passing library interface specification
  - MPI addresses primarily the message-passing parallel programming model, in which data is moved from the address space of one process to that of another process through cooperative operations on each process.
  - All MPI operations are expressed as functions, subroutines, or methods, according to the appropriate language bindings.
  - Commonly used for distributed system & high-performance computing
- MPI is not…
  - a language or compiler specification
  - a specific implementation or product

# History of MPI



# Reasons for Using MPI

- Standardization: MPI is the only message passing library which can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries
- Portability: There is no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard
- Performance: Vendor implementations should be able to exploit native hardware features to optimize performance
- Functionality: Rich set of features
- Availability: A variety of implementations are available, both vendor and public domain
  - MPICH is a popular open-source and free implementation of MPI
  - Vendors and other collaborators take MPICH and add support for their systems: Intel MPI, IBM Blue Gene MPI, Cray MPI, Microsoft MPI, MVAPICH, MPICH-MX

# What is MPICH

- MPICH is a high-performance and widely portable implementation of MPI
- It provides all features of MPI that have been defined so far
- Active development lead by Argonne National Laboratory and University of Illinois at Urbana-Champaign
- Current release is MPICH-3.3.2

# Basic API

```
// Initialize the MPI environment
int MPI_Init(int *argc, char ***argv);
// Get the number of processes
int MPI_Comm_size(
    MPI_Comm comm, /* communicator,
    A predefined communicator MPI_COMM_WORLD is provided by MPI */
    int *size      // number of processes in the group of comm
);
// Get the rank of the process
int MPI_Comm_rank(
    MPI_Comm comm, // communicator
    int *rank      // rank of the calling process in group of comm
);
// Finalize the MPI environment.
int MPI_Finalize();
```

# Hello World

```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv)
{
    MPI_Init(NULL, NULL);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    // Get the name of the processor
    MPI_Get_processor_name(processor_name, &name_len);
    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
            processor_name, world_rank, world_size);
    MPI_Finalize();
}
```

# Outline

I. Parallel computing

II. Basic Concepts of MPI

**III. Point-to-point communication**

IV. Collective communication

# MPI_Send And MPI_Recv

```
int MPI_Send(
    const void *buf,        // initial address of send buffer
    int count,              // number of elements in send buffer
    MPI_Datatype datatype, // datatype of each send buffer element
    int dest,               // rank of destination
    int tag,                // message tag
    MPI_Comm comm           // communicator
);
int MPI_Recv(
    void *buf,              // initial address of receive buffer
    int count,              // number of elements in receive buffer
    MPI_Datatype datatype, // datatype of each receive buffer element
    int source,             // rank of source
    int tag,                // message tag
    MPI_Comm comm,          // communicator
    MPI_Status *status      // status object
)
```

# MPI ping pong program

```
int ping_pong_count = 0;
int partner_rank = (world_rank + 1) % 2;
while (ping_pong_count < PING_PONG_LIMIT) {
  if (world_rank == ping_pong_count % 2) {
    ping_pong_count++; // Increment the ping pong count before you send it
    MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
    fprintf(fp, "[%d -> %d]: sent ping_pong_count %d\n",
            world_rank, partner_rank, ping_pong_count);
  } else {
    MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    fprintf(fp, "[%d <- %d]: received ping_pong_count %d\n",
            world_rank, partner_rank, ping_pong_count);
  }
}
```

# MPI ping pong program

cat oneflow-12__0.txt
[0 -> 1]: sent ping_pong_count 1
[0 <- 1]: recv ping_pong_count 2
[0 -> 1]: sent ping_pong_count 3
[0 <- 1]: recv ping_pong_count 4
[0 -> 1]: sent ping_pong_count 5
[0 <- 1]: recv ping_pong_count 6
[0 -> 1]: sent ping_pong_count 7
[0 <- 1]: recv ping_pong_count 8
[0 -> 1]: sent ping_pong_count 9
[0 <- 1]: recv ping_pong_count 10

cat oneflow-13__1.txt
[1 <- 0]: recv ping_pong_count 1
[1 -> 0]: sent ping_pong_count 2
[1 <- 0]: recv ping_pong_count 3
[1 -> 0]: sent ping_pong_count 4
[1 <- 0]: recv ping_pong_count 5
[1 -> 0]: sent ping_pong_count 6
[1 <- 0]: recv ping_pong_count 7
[1 -> 0]: sent ping_pong_count 8
[1 <- 0]: recv ping_pong_count 9
[1 -> 0]: sent ping_pong_count 10

# Outline

I. Parallel computing

II. Basic Concepts of MPI

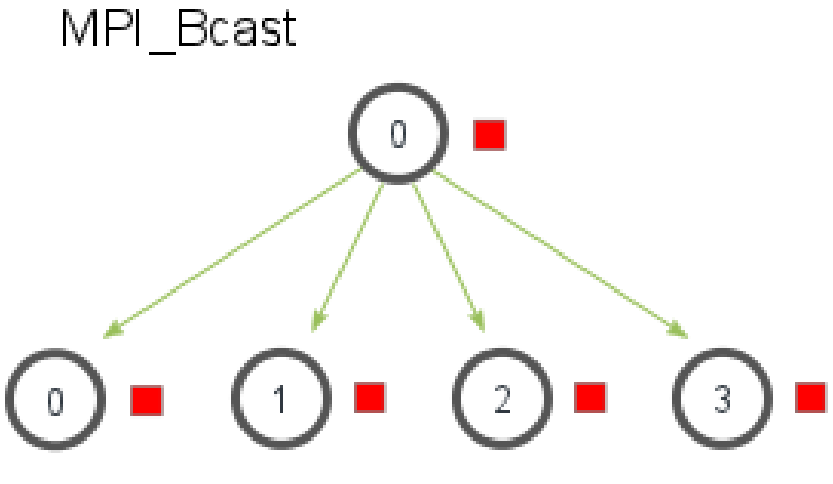
III. Point-to-point communication

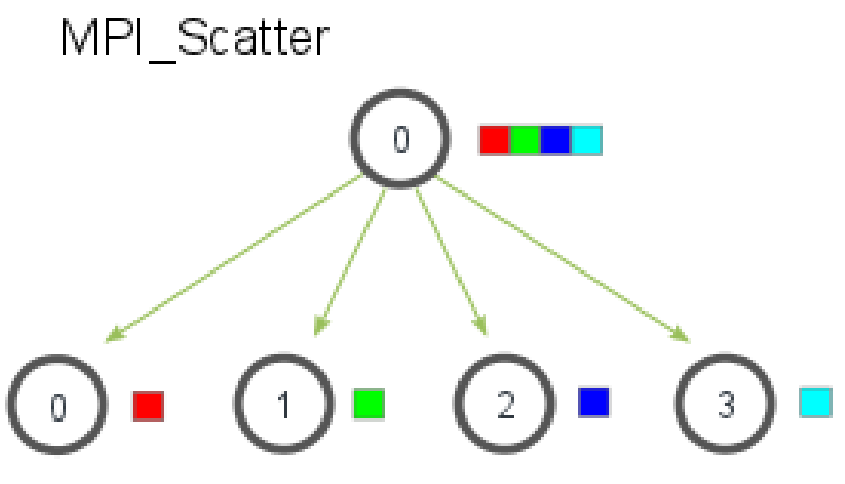
**IV. Collective communication**

# MPI_Barrier and MPI_Bcast

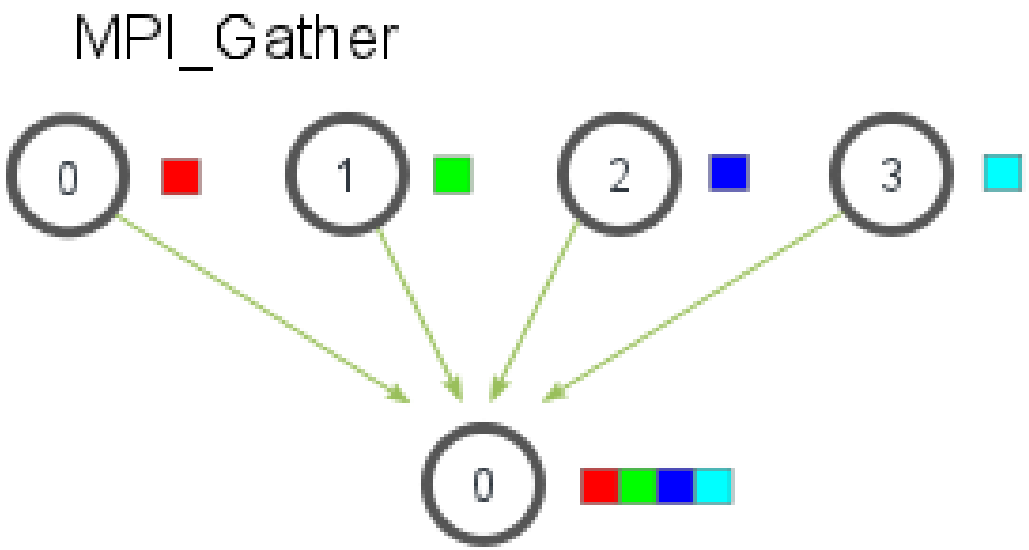
```
// Barrier synchronization across all members of a group
int MPI_Barrier(
    MPI_Comm comm // communicator
);
// Broadcast from one member to all members of a group
int MPI_Bcast(
    void *buffer,          // starting address of buffer
    int count,             // number of entries in buffer
    MPI_Datatype datatype, // data type of buffer
    int root,              // rank of broadcast root
    MPI_Comm comm          // communicator
);
```

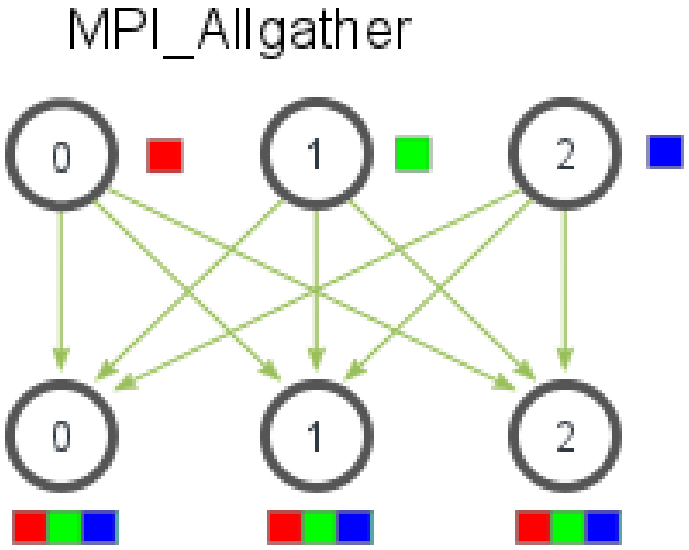
# MPI_Bcast and MPI_Scatter



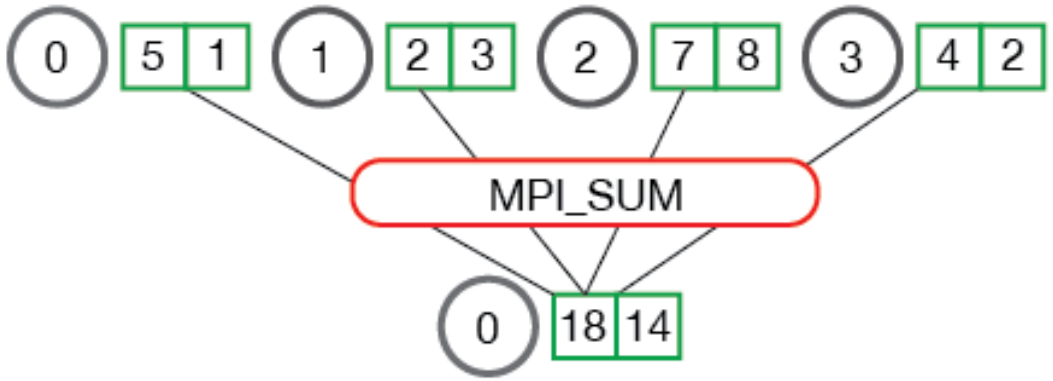
# MPI_Gather and MPI_Allgather



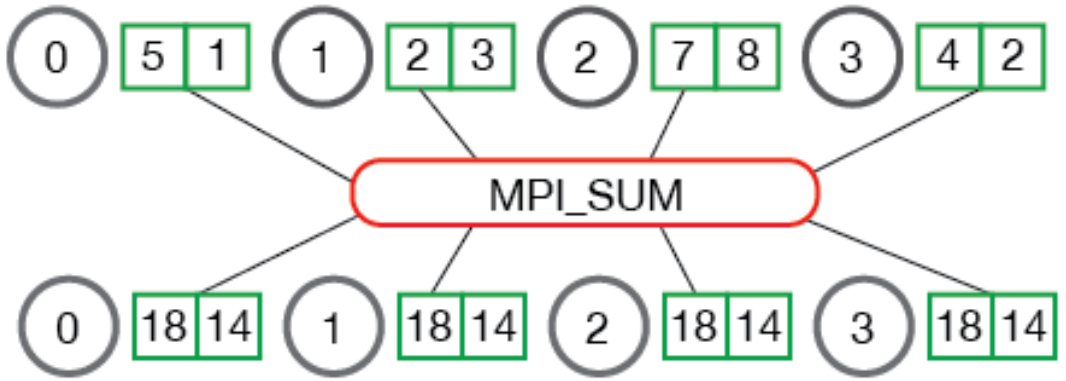



# MPI_Reduce and MPI_Allreduce

MPI_Reduce




MPI_Allreduce



# More Concepts

- Blocking vs. Non-blocking Communication
- MPI Datatypes
- MPI Group
- Parallel I/O
- Remote Memory Access
- …

# References


- MPI-3.1: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
- MPICH: http://www.mpich.org
- MPI Tutorial: https://mpitutorial.com
- MPI for Dummies： https://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-basic.pdf
- 周志遠《平行程式》： http://ocw.nthu.edu.tw/ocw/index.php?page=course&cid=231

之江实验室 ZHEJIANG LAB

# Thanks