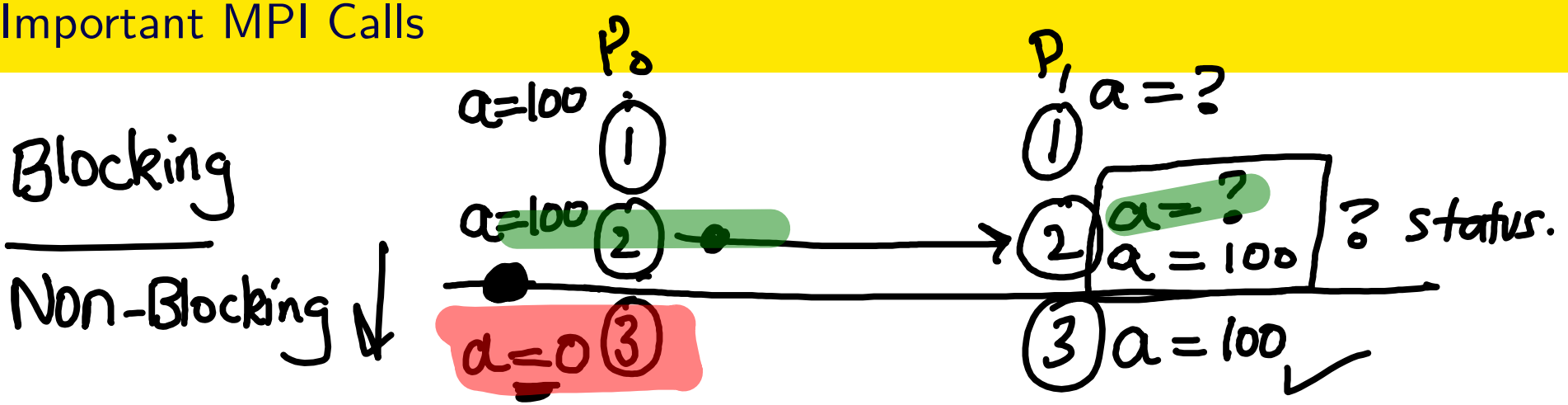
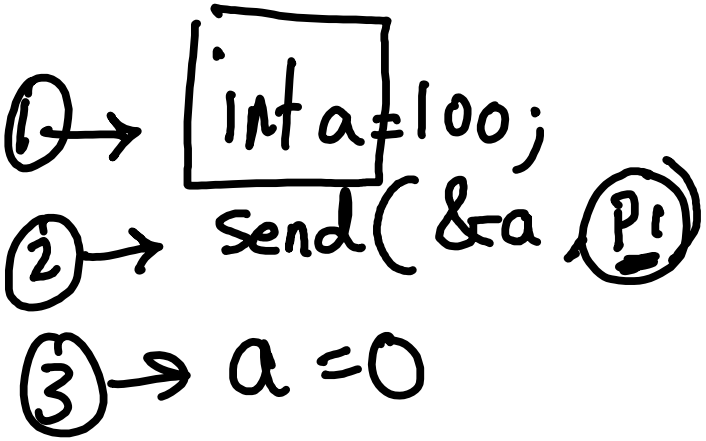
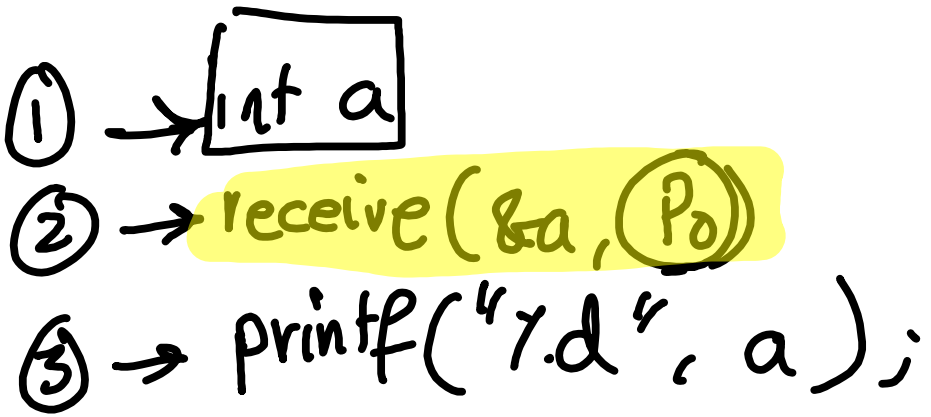
Important MPI Calls



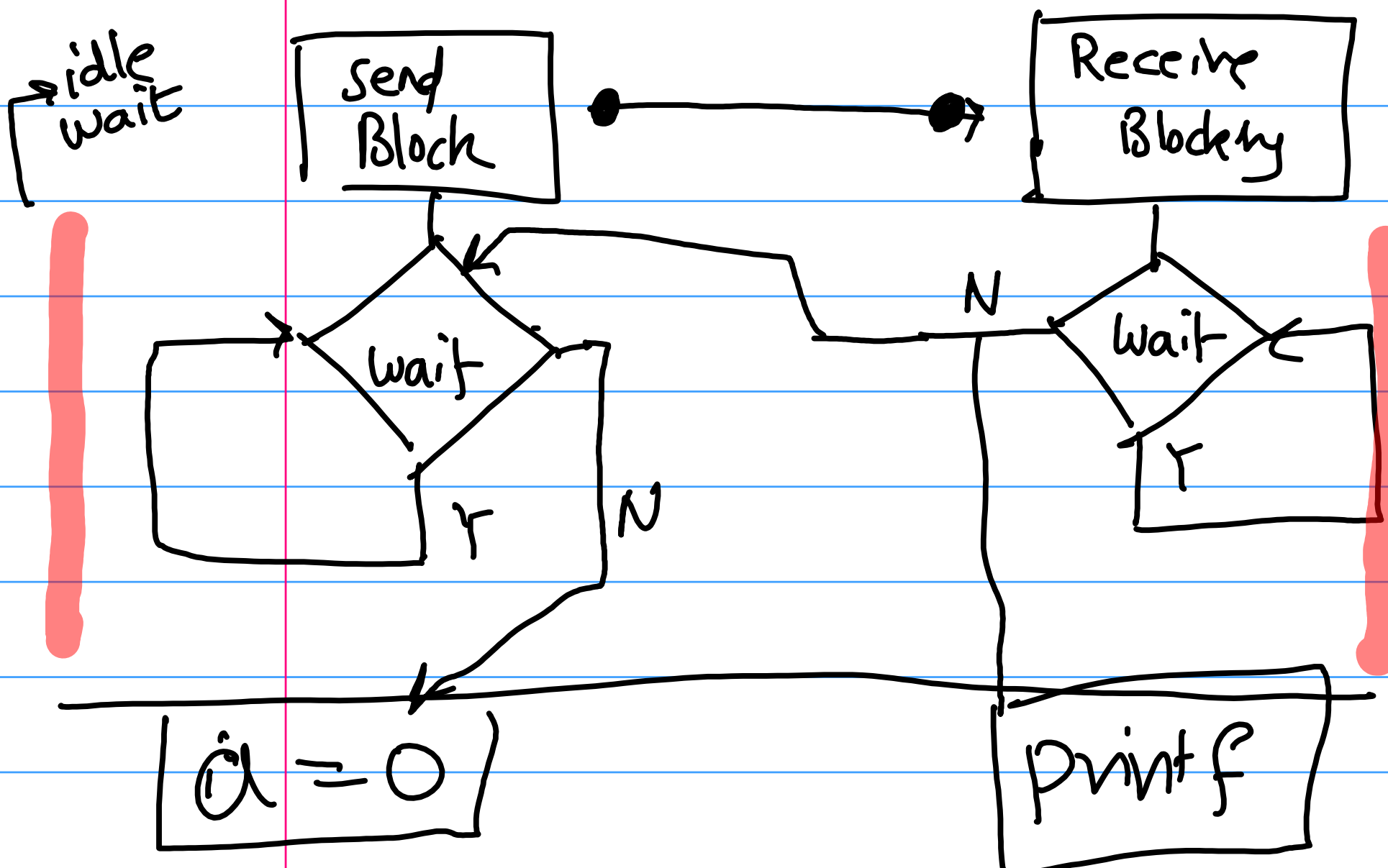

```
MPI_Init(int*, char**);                 // Initiate an MPI Computation
MPI_Finalize(void);                     // Terminate an MPI Computation
MPI_Comm_size(MPI_COMM, int);           // How many processes
MPI_Comm_rank(MPI_COMM, int);           // Who am I?
MPI_Get_processor_name(char*, int);     // What is the hostname?
MPI_Wtime(void);                        // Elapsed time in seconds
MPI_Abort(MPI_COMM);                    // Terminate all processes
```

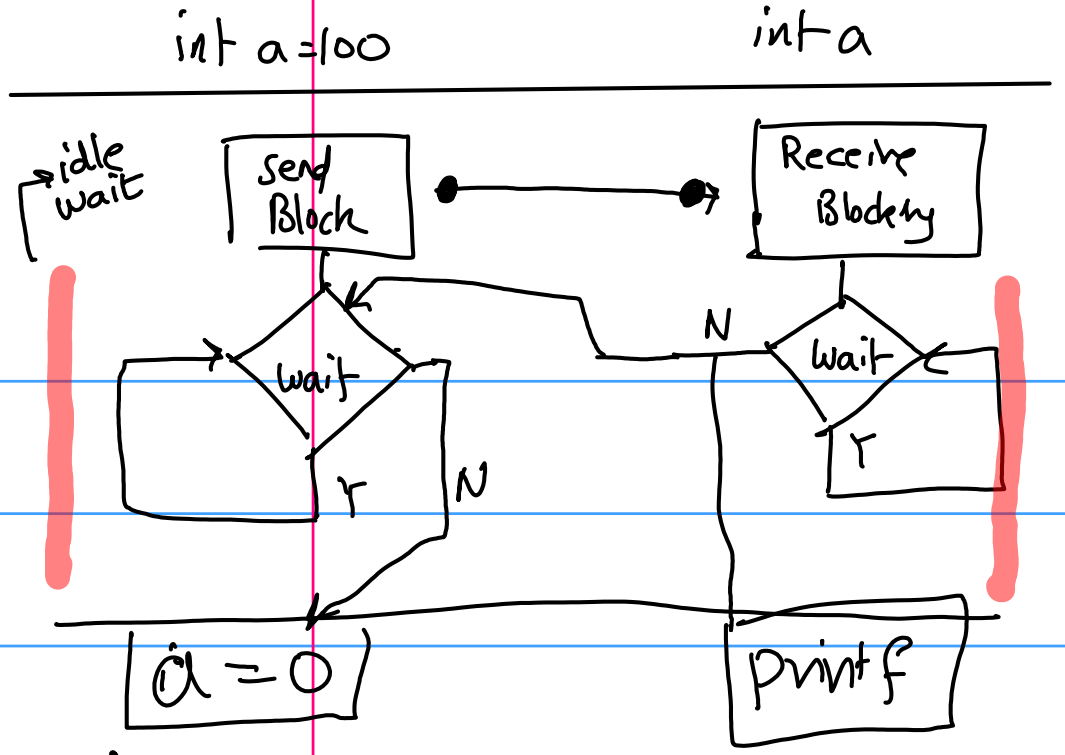




int a = 100

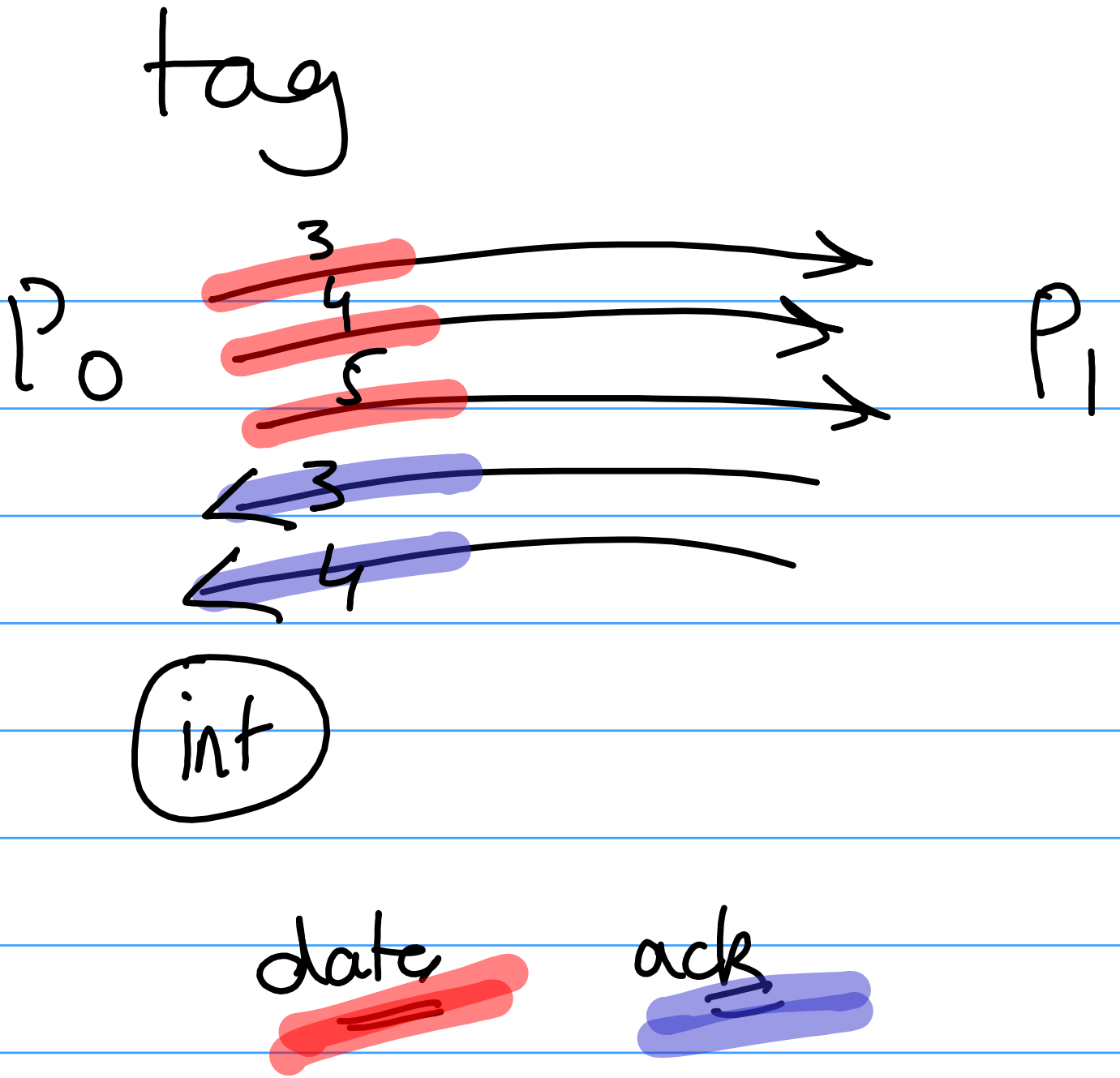
int a






```
int a=100
if(my_rank==0)
{
    send(a, 1)
    ↓ ← Receive
}
a = 0
```

```
int a;
if(my_rank==1)
{
    Send(a,0)
    ↓ ← recv.
}
printf
```


tag
3
4
5
P_0
P_1
3
4
int
date
ack

Approaches to Send/Receive

Blocking (Non-Buffered) Send/Receive

- Follow some form of "handshaking" protocol
  - Request to Send → Clear to Send → Send Data → Acknowledgement
  - Problem 1: Idling Overhead (both sender/receiver side)
  - Problem 2: Deadlock (sending at same time)

Blocking (Buffered) Send/Receive

- Copy send-data to designated buffer, and returns after "copy" operation is completed
- Problem 1: Buffer Size

```
for (i = 0; i < 1000; i++) {
    produce_data(&a);
    send(&a, P1);
}
```

```
for (i = 0; i < 1000; i++) {
    receive(&a, P0);
    consume_data(&a);
}
```

- Problem 2: Deadlock (sending at same time)

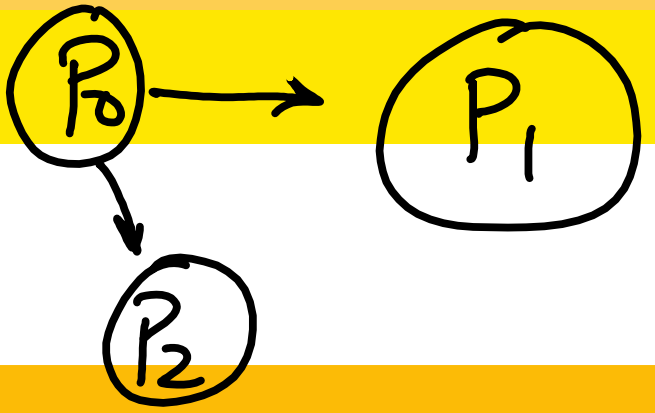
Approaches to Send/Receive (cont.)

Non-Blocking Send/Receive

- Return from Send/Receive operation before it is "safe" to return.
- Programmer responsibility to ensure that "sending data" is not altered immediately

- Blocking Operations: Safe and Easy Programming (at cost of overhead and risk of deadlocks)
- Non-Blocking Operations: Useful for Performance optimization, and breaking deadlocks (but brings in plenty of race-conditions if programmer not careful)

Point to Point Communication



Types of Point-to-Point Send/Receive Calls

- **Synchronous Transfer**: Send/Receive routines return only when the message transfer is completed. Not only does this transfer data, but it also synchronizes processes

```
MPI_Send()      // Blocking Send
MPI_Recv()      // Blocking Receive
```

- **Asynchronous Transfers**: Send/Receive do not wait for transfer data and proceeds with execution next line of instruction. (Precaution: Do not modify the send/receive buffers)

```
MPI_Isend()     // Non-Blocking Send
MPI_Irecv()     // Non-Blocking Receive
```

Point to Point Communication (cont.)

① MPI_Send(&a, 1 - - - ..)

② MPI_Send(a, 100 - - - - - -)

Sending

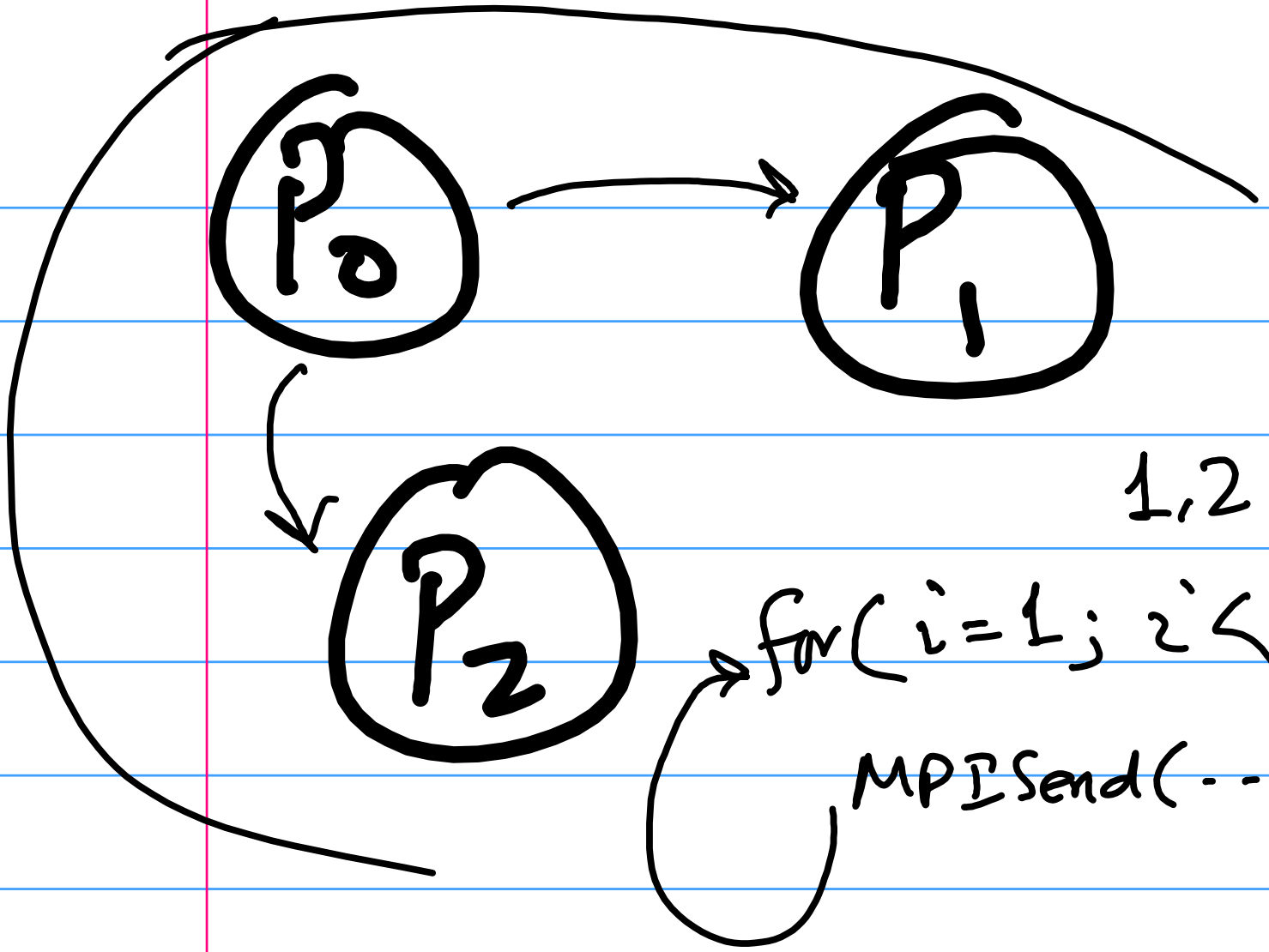
```
int MPI_Send(void *buffer,    int count,    MPI_DATATYPE datatype,
             int destination, int tag,     MPI_Comm comm);
```

① int a

② int a[100]

- Send the data stored in **buffer**
- **Count** is the number of entries in the buffer
- What is the **datatype** of the buffer (MPI_CHAR, MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_CHAR, etc.)
- **Destination** is the rank of process, to whom buffer is to be sent to, residing in communication universe **comm**
- The **tag** of the message (to distinguish between different types of messages)

get_size MPI_Comm_world



1, 2

```
for(i=1; i<3; i++)
    MPISend(.... i ....)
```

Point to Point Communication (cont.)

Receiving

```
int MPI_Recv(void *buffer,    int count,   MPI_DATATYPE datatype,
             int source,      int tag,     MPI_Comm comm,
             MPI_Status *status);
```

- Store the received message in **buffer**
- **Count** is the number of entries to be received in the buffer. If number of entries is larger than the capacity of buffer, an overflow error MPI_ERR_TRUNCATE is returned.
- **Datatype** is the type of data that has been received
- **Source** is the rank of process, residing in communication domain **comm**, from whom buffer is received. Source can be hard-set, or a wild-card MPI_ANY_SOURCE.
- To retrieve message of certain type, set the **tag** argument. If there are many messages of same tag from same process, any one of them may be retrieved. If message of any tag is to be retrieved, use the wild-card MPI_ANY_TAG.
- Store status of received message in **status** (next slide). If not needed, use MPI_STATUS_IGNORE