Arhitecturi Paralele
MPI

Prof. Florin Pop
As. Drd. Ing. Cristian Chilipirea
cristian.chilipirea@cs.pub.ro

Elemente preluate din cursul Prof. Ciprian Dobre

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Installing OpenMPI

apt-get install libopenmpi-dev openmpi-bin openmpi-doc openmpi-common

Compiling and running MPI programs

mpicc test.c

mpirun –np **4** a.out

mpirun –np **3** date

Starts 4 processes. Possibly on different machines. They are identical but have different ids. Works with non-MPI programs.

./a.out

Works but starts only **one** process.

MPI example

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

int main(int argc, char * argv[])
{
		int rank;
		int nProcesses;
		MPI_Init(&argc, &argv);
		MPI_Comm_rank(MPI_COMM_WORLD, &rank);
		MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
		printf("Hello from %i/%i\n", rank, nProcesses);
		MPI_Finalize();
		return 0;
}
```

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Start MPI Process

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Get the id (rank)

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Get the total number of processed

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Print hello from all processes.

MPI example

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

int main(int argc, char * argv[])
{
        int rank;
        int nProcesses;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
        printf("Hello from %i/%i\n", rank, nProcesses);
        MPI_Finalize();
        return 0;
}
```

Stop the MPI environment.

MPI example executed

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

int main(int argc, char * argv[])
{
	int rank;
	int nProcesses;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
	printf("Hello from %i/%i\n", rank, nProcesses);
	MPI_Finalize();
	return 0;
}
```

Hello from 0/4

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

int main(int argc, char * argv[])
{
	int rank;
	int nProcesses;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
	printf("Hello from %i/%i\n", rank, nProcesses);
	MPI_Finalize();
	return 0;
}
```

Hello from 3/4

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

int main(int argc, char * argv[])
{
	int rank;
	int nProcesses;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
	printf("Hello from %i/%i\n", rank, nProcesses);
	MPI_Finalize();
	return 0;
}
```

Hello from 2/4

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

int main(int argc, char * argv[])
{
	int rank;
	int nProcesses;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &nProcesses);
	printf("Hello from %i/%i\n", rank, nProcesses);
	MPI_Finalize();
	return 0;
}
```

Hello from 1/4

MPI memory

- Nu avem memorie partajată în MPI
- Toate variabilele sunt locale proceselor.
- Pentru a muta informație de la un proces la altul vor trebui folosite funcții explicite.
  - Send/Recv
  - Broadcast
  - Scatter
  - Gather

MPI_Send/MPI_Recv

int MPI_Send(↓ void *b, ↓ int c, ↓ MPI_Datatype d, ↓ int reiceiver, ↓ int t, ↓ MPI_Comm)

void *b	int c	MPI_Datatype d	int reiceiver	int t	MPI_Comm
v &v[3] &a v+5	num_el(v) [0,..)	MPI_INT MPI_CHAR MPI_FLOAT MPI_LONG	[0, **num_tasks**)	[0, ..)	MPI_COMM_WORLD

MPI_Send/MPI_Recv

int MPI_Recv(↑ void *b, ↓ int c, ↓ MPI_Datatype d, ↓ int sender, ↓ int t, ↓ MPI_Comm, ↑ MPI_Status *)

v
&v[3]
&a
v+5

MPI_INT
MPI_CHAR
MPI_FLOAT
MPI_LONG

[0, ..)
MPI_ANY_TAG

[0, **num_tasks**)
MPI_ANY_SOURCE

num_el(v)
[0,..)

MPI_COMM_WORLD

&Stat
MPI_STATUS_IGNORE
Stat.MPI_SOURCE, Stat.MPI_TAG

MPI_Bcast

int MPI_Bcast (↕ void *b, ↓ int c, ↓ MPI_Datatype d, ↓ int root, ↓ MPI_Comm)

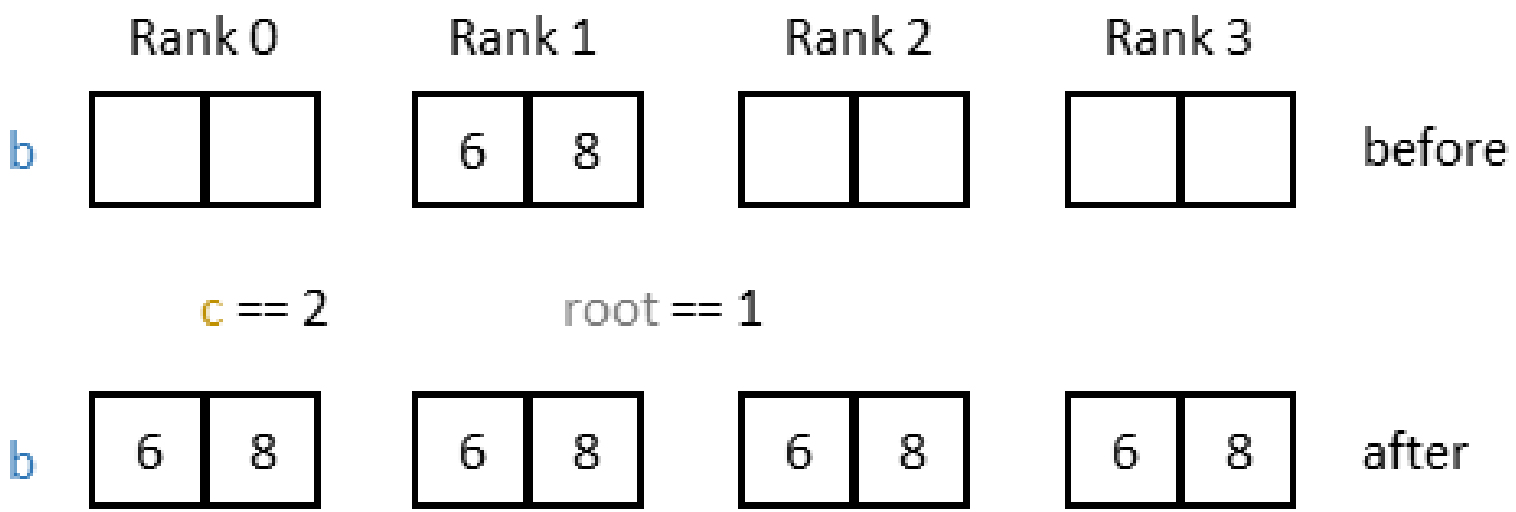
v num_el(v)
&v[3] [0,..)
&a
v+5

[0, **num_tasks**)

MPI_INT
MPI_CHAR
MPI_FLOAT
MPI_LONG

MPI_COMM_WORLD

MPI_Bcast



Rank 0
Rank 1
Rank 2
Rank 3
b
6
8
before
c == 2
root == 1
b
6
8
6
8
6
8
6
8
after

MPI_Scatter

int MPI_Scatter (↓ void *sb, ↓ int sc, ↓ MPI_Datatype sd, ↑ void *rb, ↓ int rc, ↓ MPI_Datatype rd, ↓ int root, ↓ MPI_Comm)

v
&v[3]
&a
v+5

num_el(v)/**num_tasks**
[0,..)

v
&v[3]
&a
v+5

num_el(v)/**num_tasks**
[0,..)

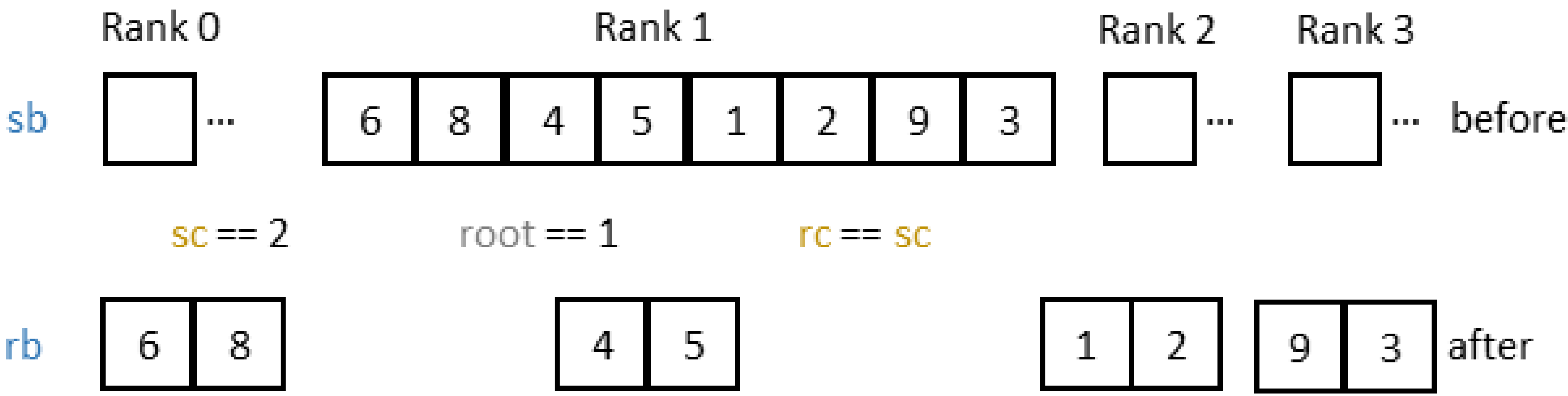
[0, **num_tasks**)

MPI_COMM_WORLD

MPI_INT
MPI_CHAR
MPI_FLOAT
MPI_LONG

MPI_INT
MPI_CHAR
MPI_FLOAT
MPI_LONG

MPI_Scatter


Rank 0
Rank 1
Rank 2
Rank 3
sb
6 8 4 5 1 2 9 3
before
sc == 2
root == 1
rc == sc
rb
6 8
4 5
1 2
9 3
after

MPI_Gather

int MPI_Gather (↓ void *sb, ↓ int sc, ↓ MPI_Datatype sd, ↑ void *rb, ↓ int rc, ↓ MPI_Datatype rd, ↓ int root, ↓ MPI_Comm)

v
&v[3]
&a
v+5

num_el(v)/**num_tasks**
[0,..)

v
&v[3]
&a
v+5

[0, **num_tasks**)

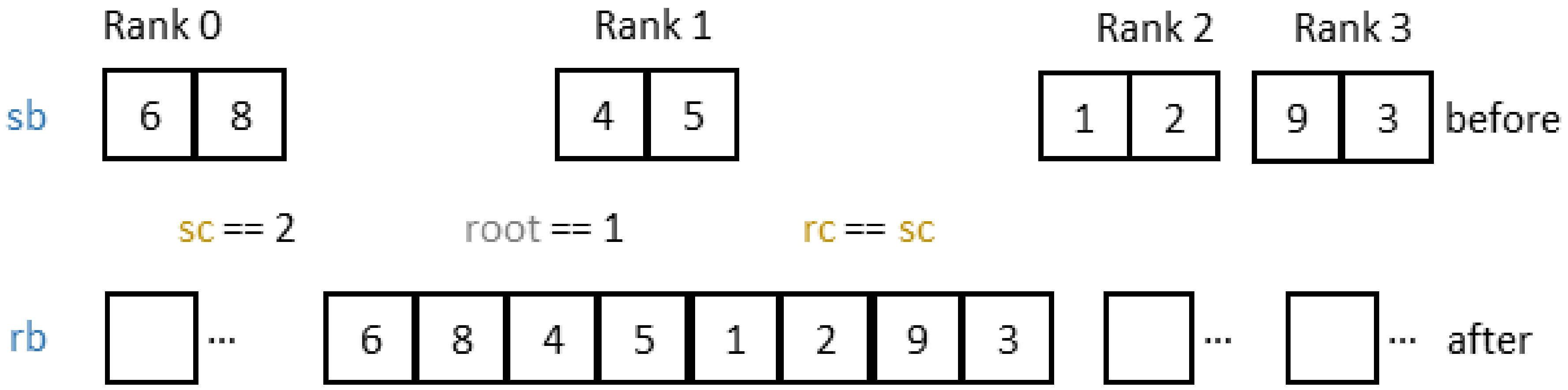
num_el(v)/**num_tasks**
[0,..)

MPI_INT
MPI_CHAR
MPI_FLOAT
MPI_LONG

MPI_INT
MPI_CHAR
MPI_FLOAT
MPI_LONG

MPI_COMM_WORLD

MPI_Gather


Rank 0
Rank 1
Rank 2
Rank 3
sb
6
8
4
5
1
2
9
3
before
sc == 2
root == 1
rc == sc
rb
...
6
8
4
5
1
2
9
3
...
...
after

MPI blocking/non-blocking send/recv

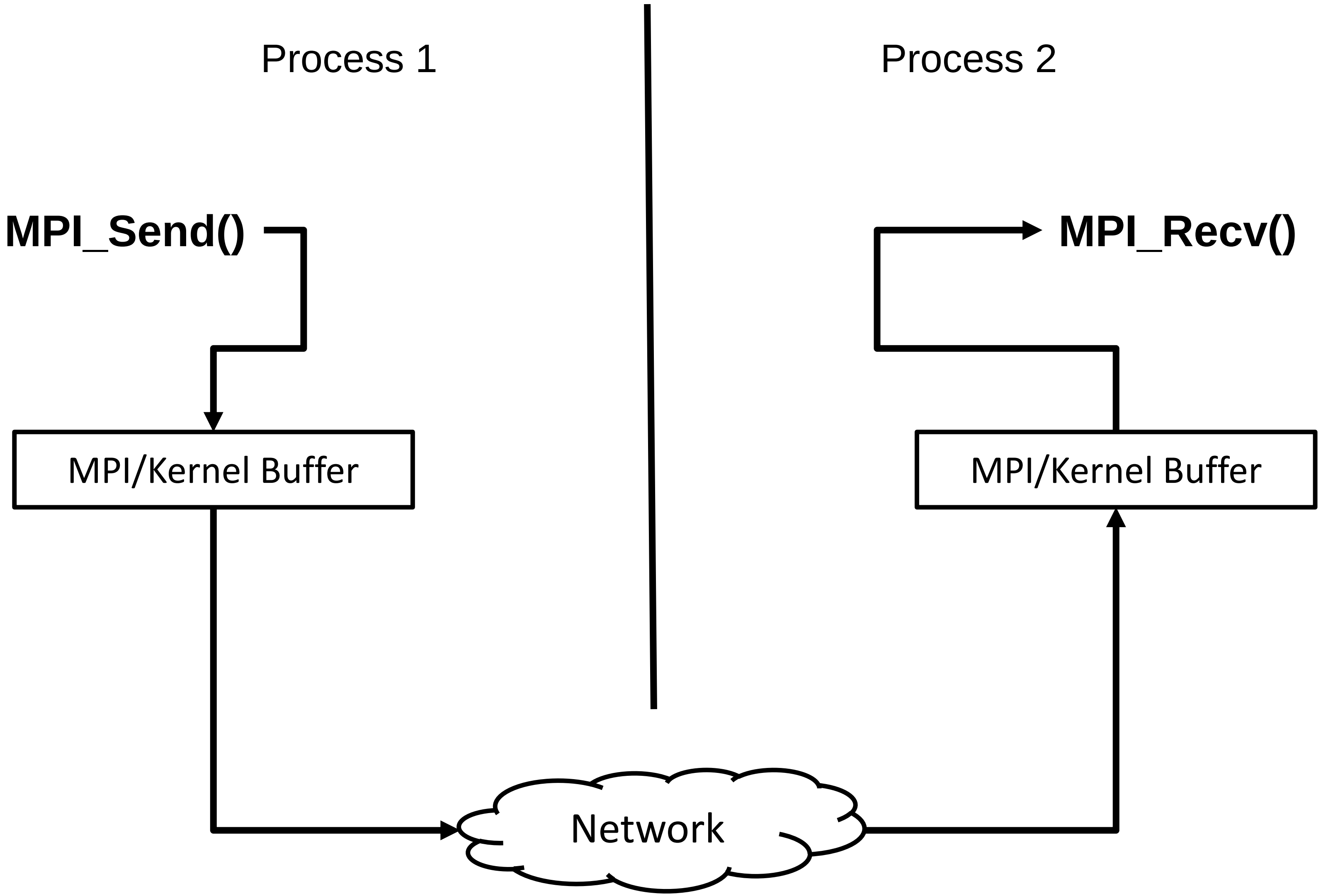
Process 1

Process 2

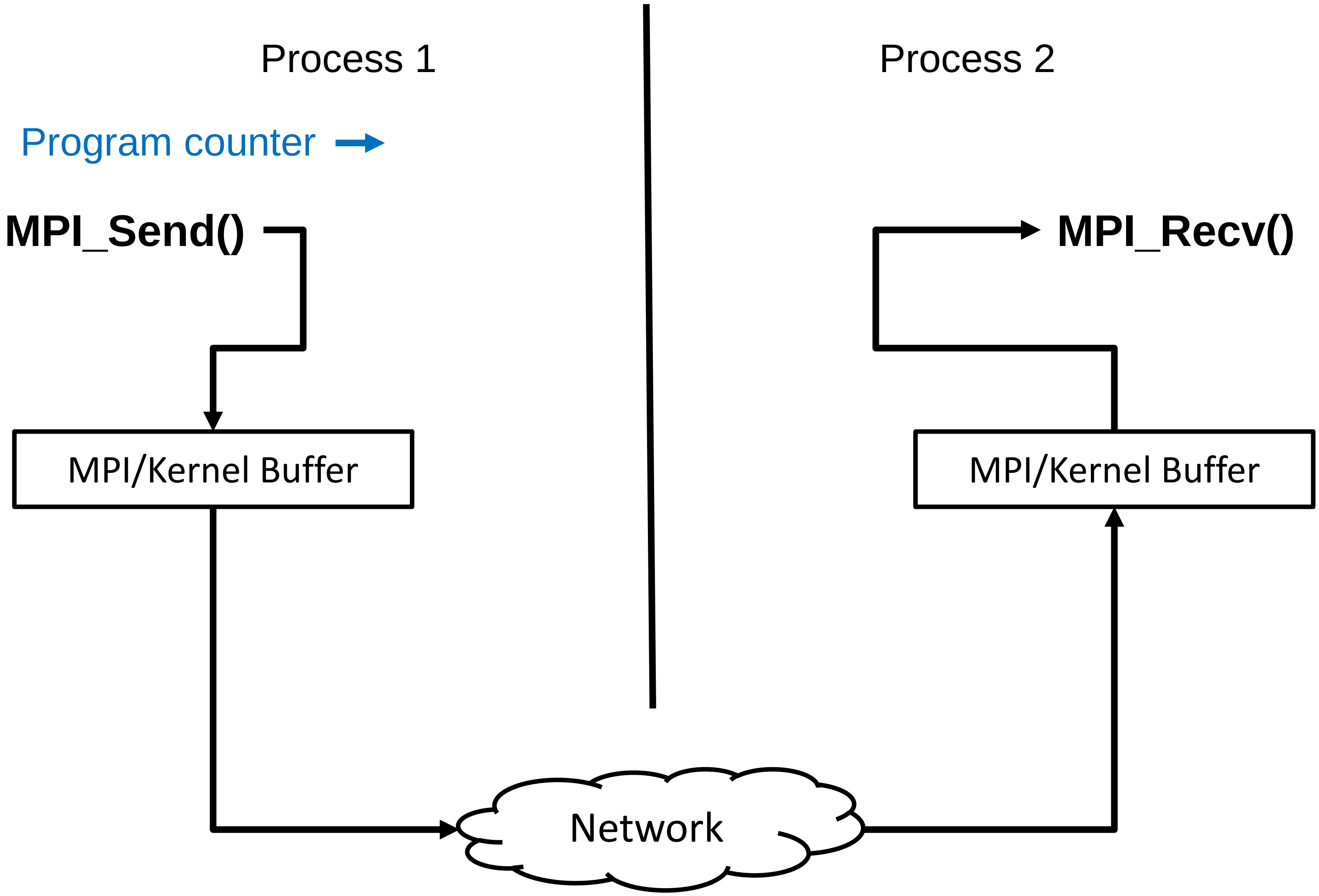
MPI_Send()

MPI_Recv()

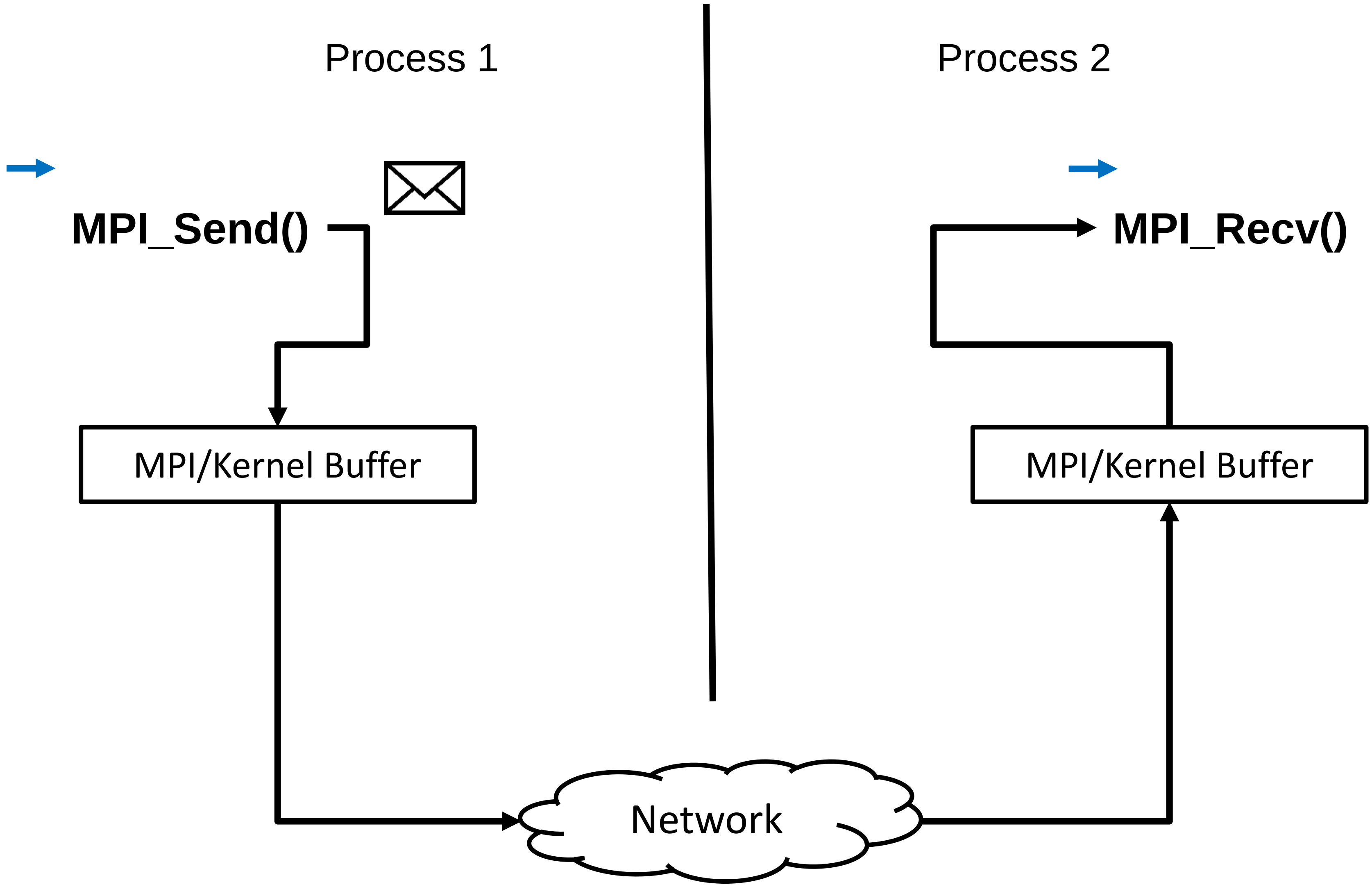
MPI blocking/non-blocking send/recv



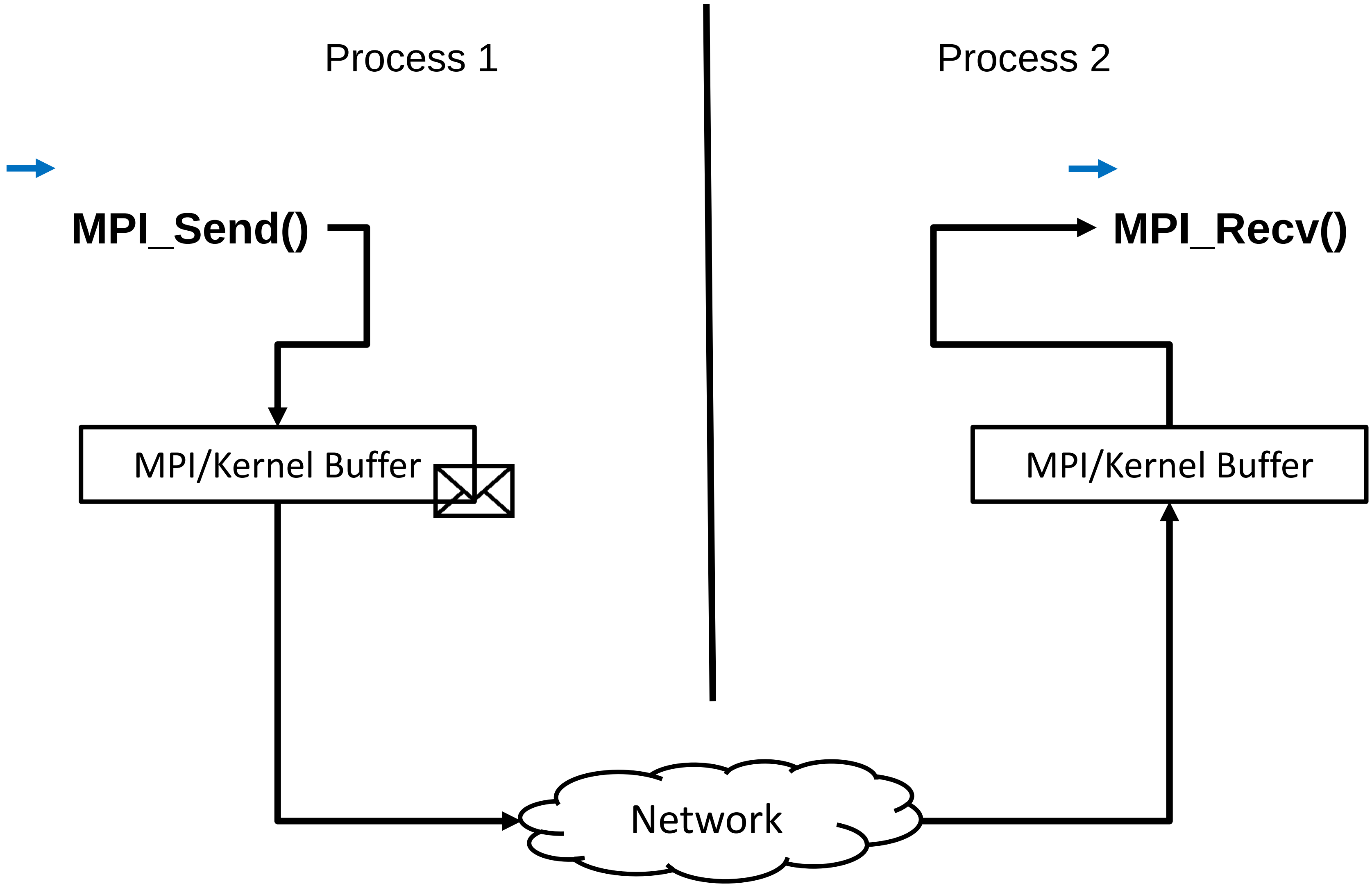
MPI blocking recv/non-blocking send



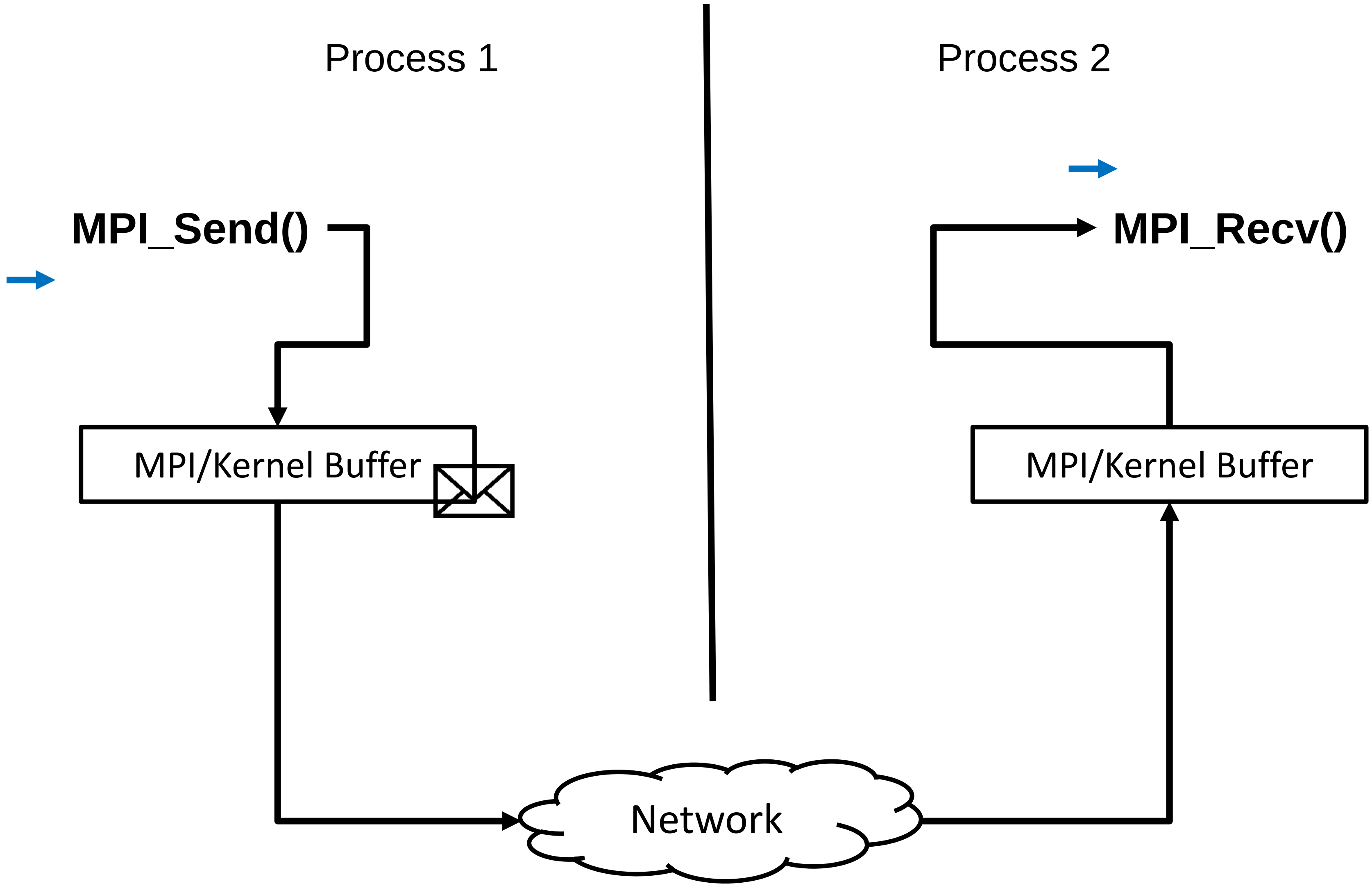
MPI blocking recv/non-blocking send



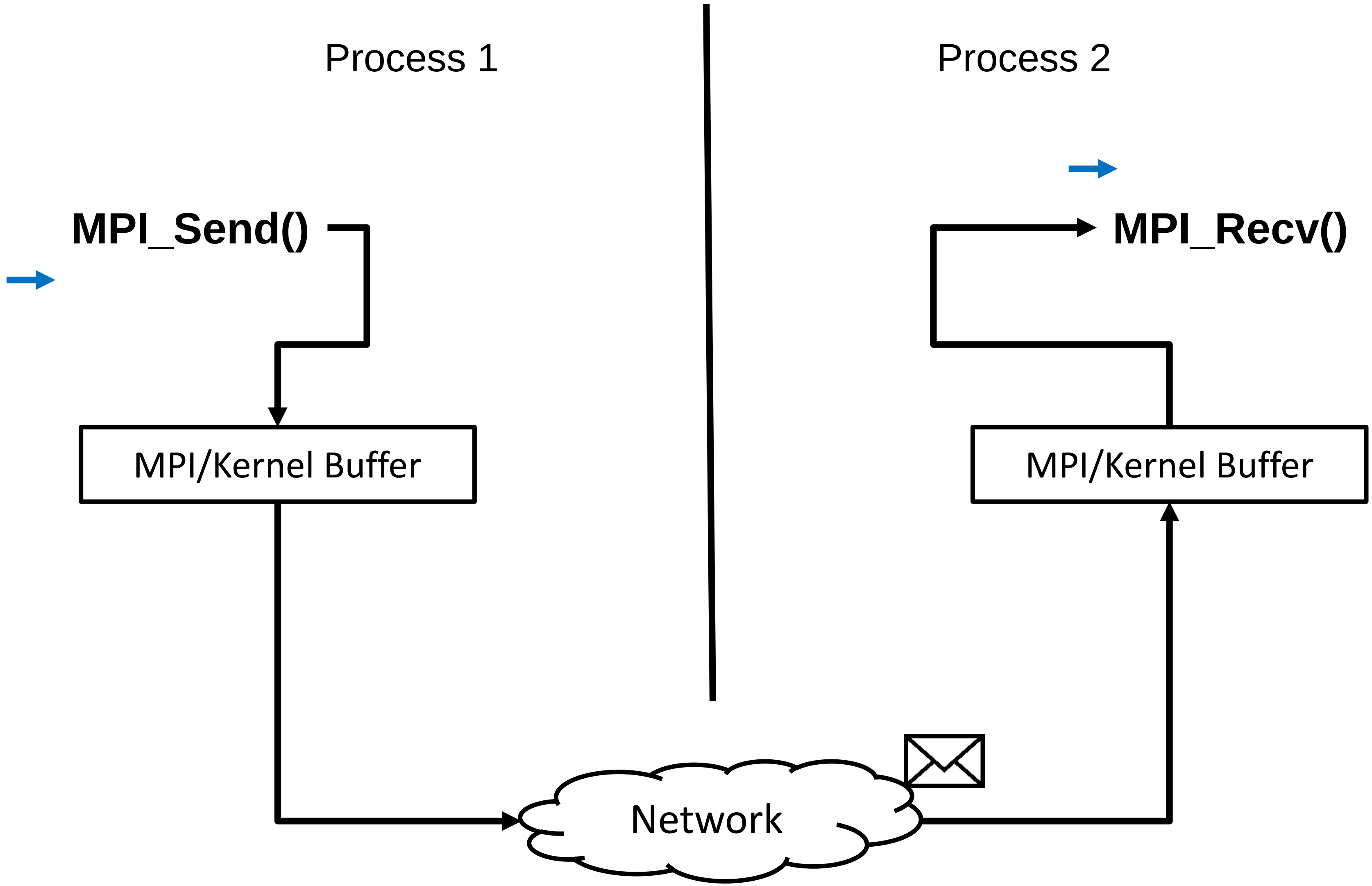
MPI blocking recv/non-blocking send



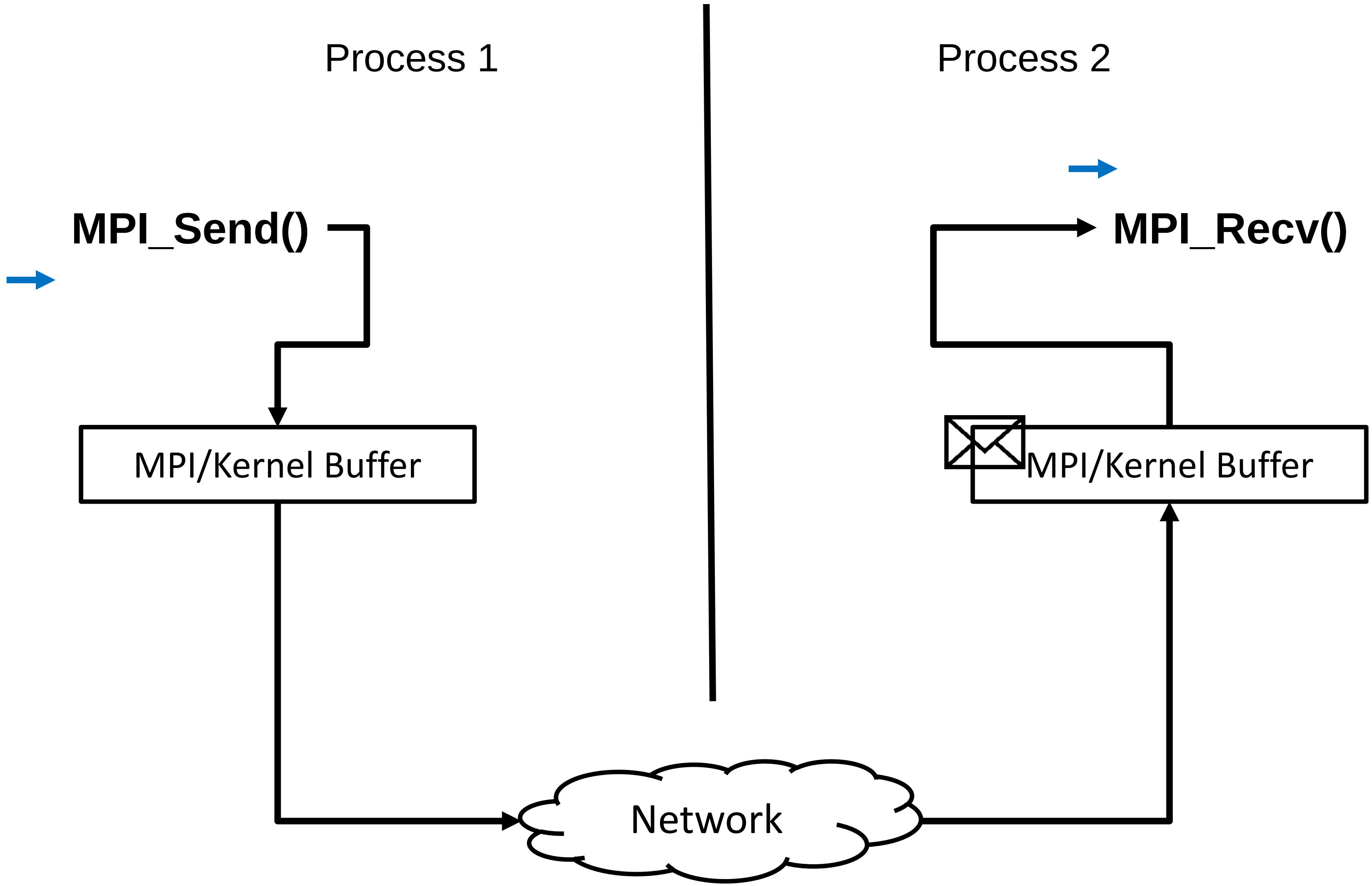
MPI blocking recv/non-blocking send



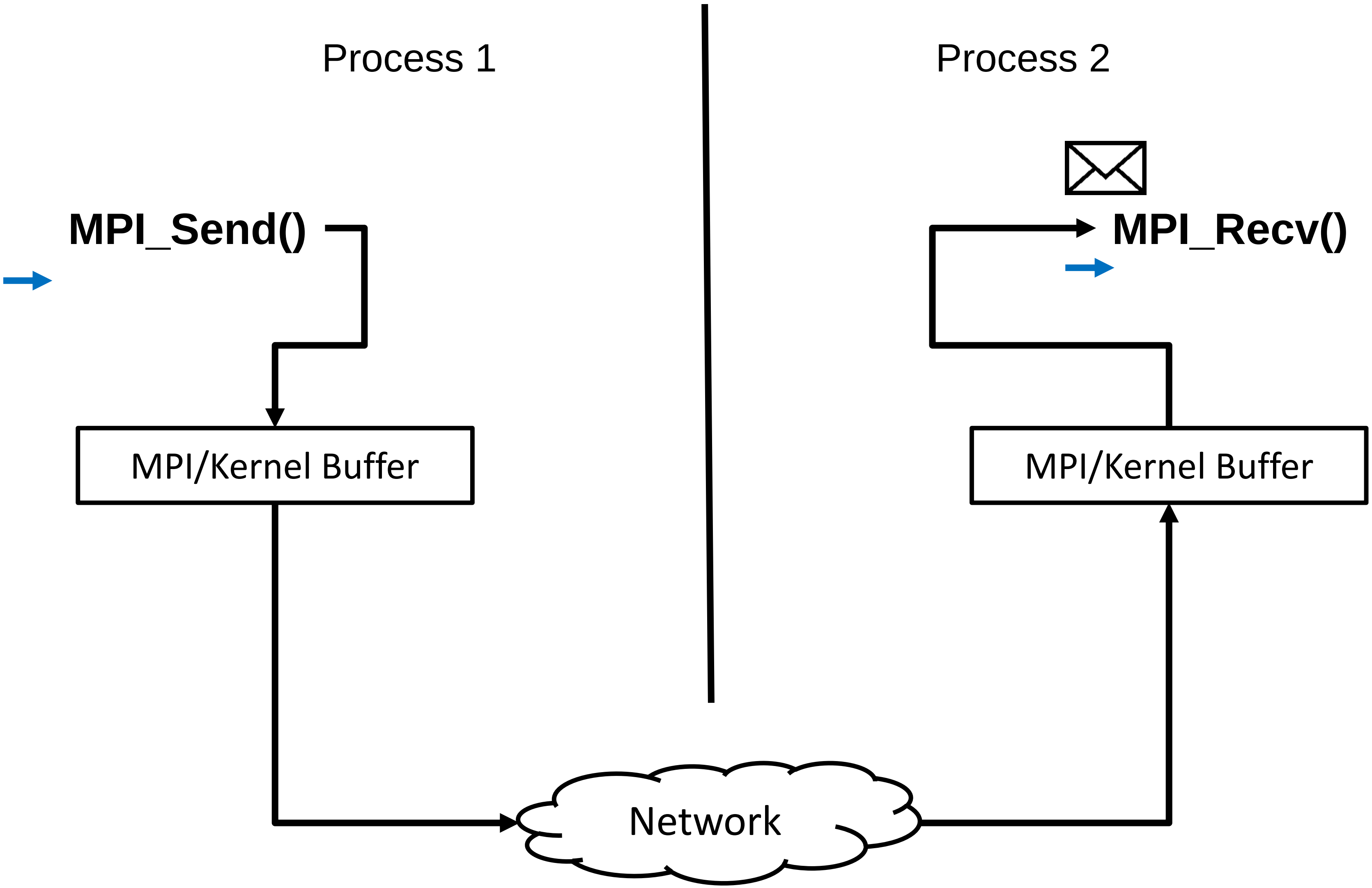
MPI blocking recv/non-blocking send



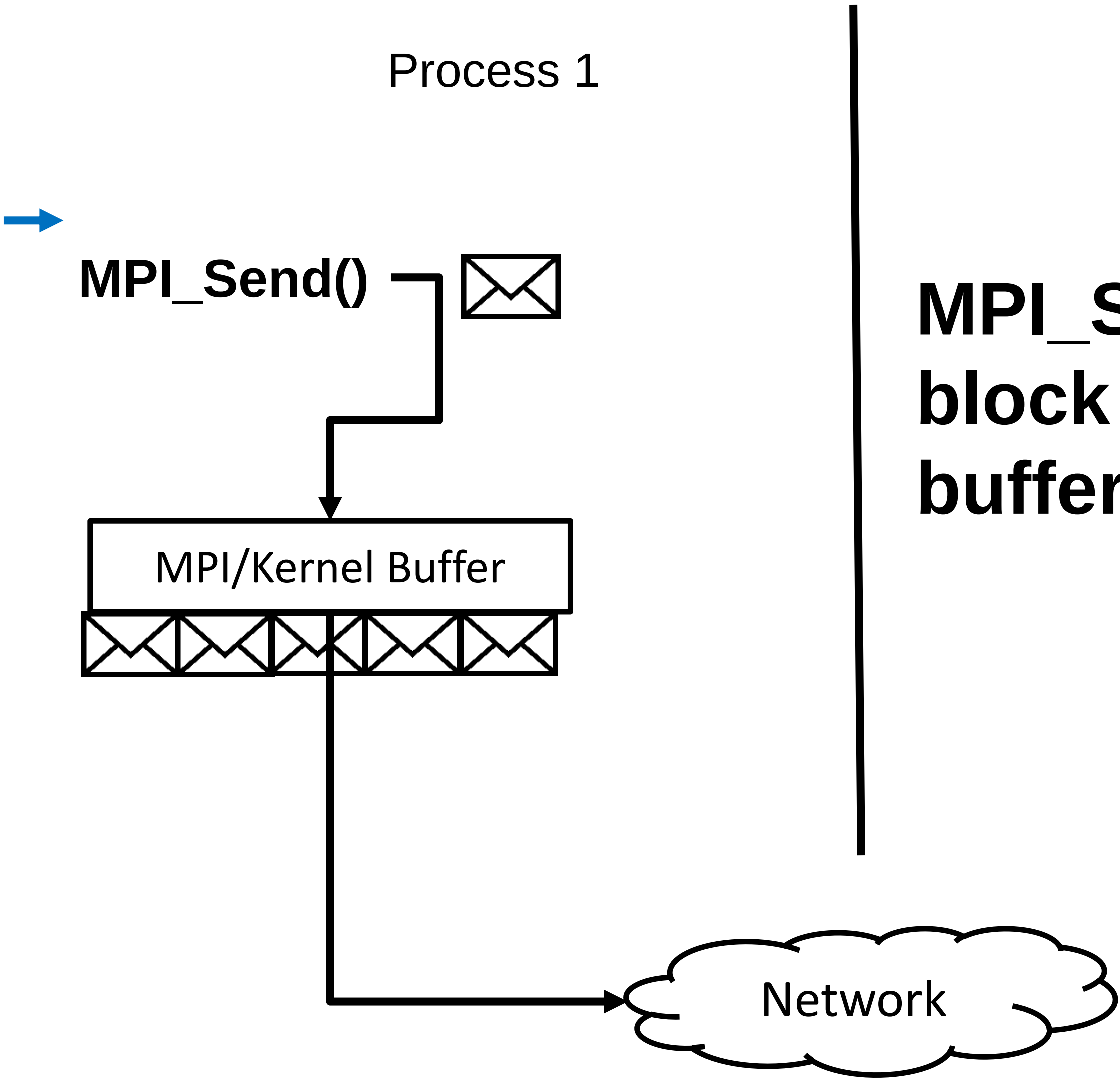
MPI blocking recv/non-blocking send



MPI blocking recv/non-blocking send

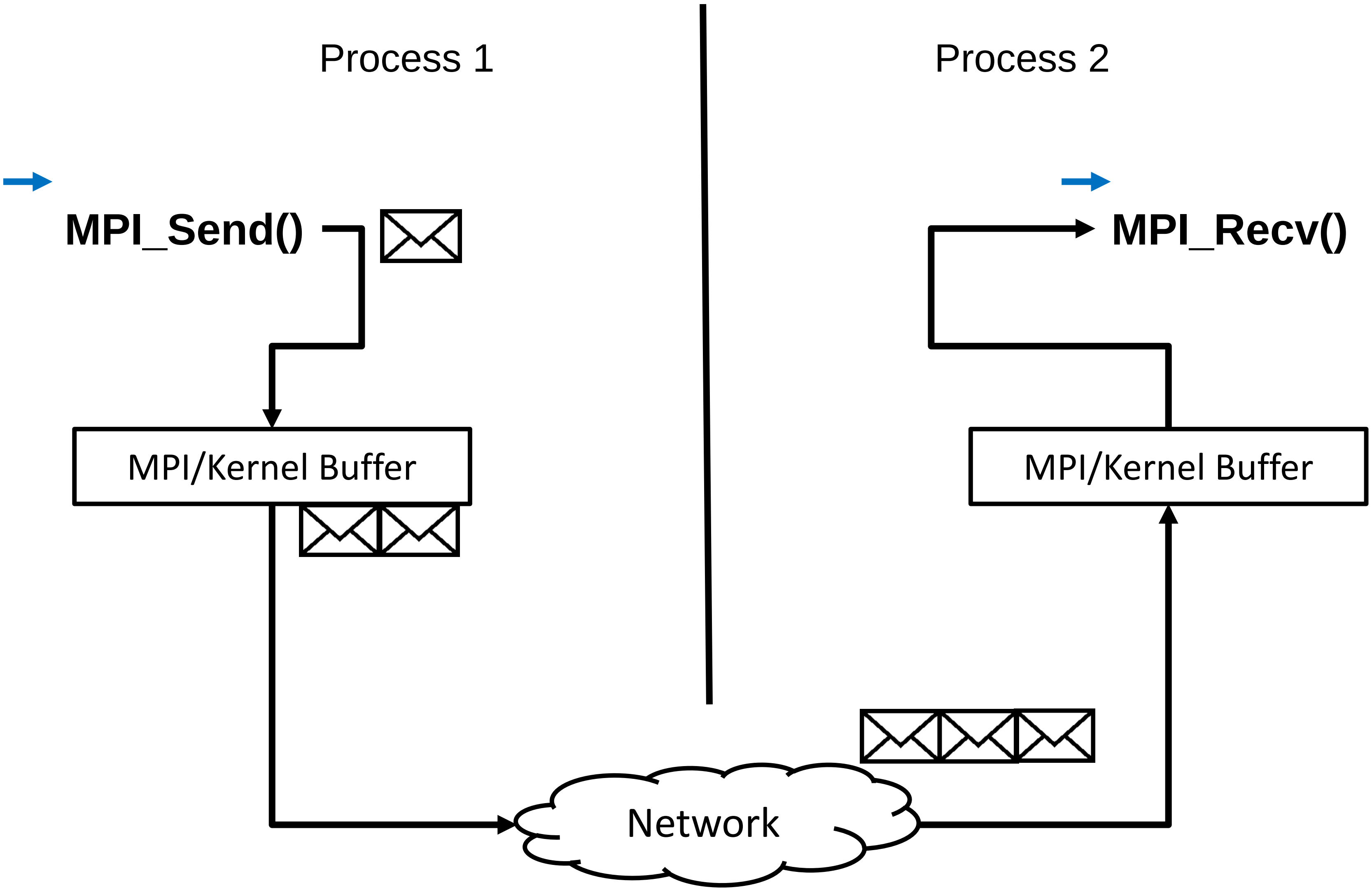


MPI blocking recv/blocking send

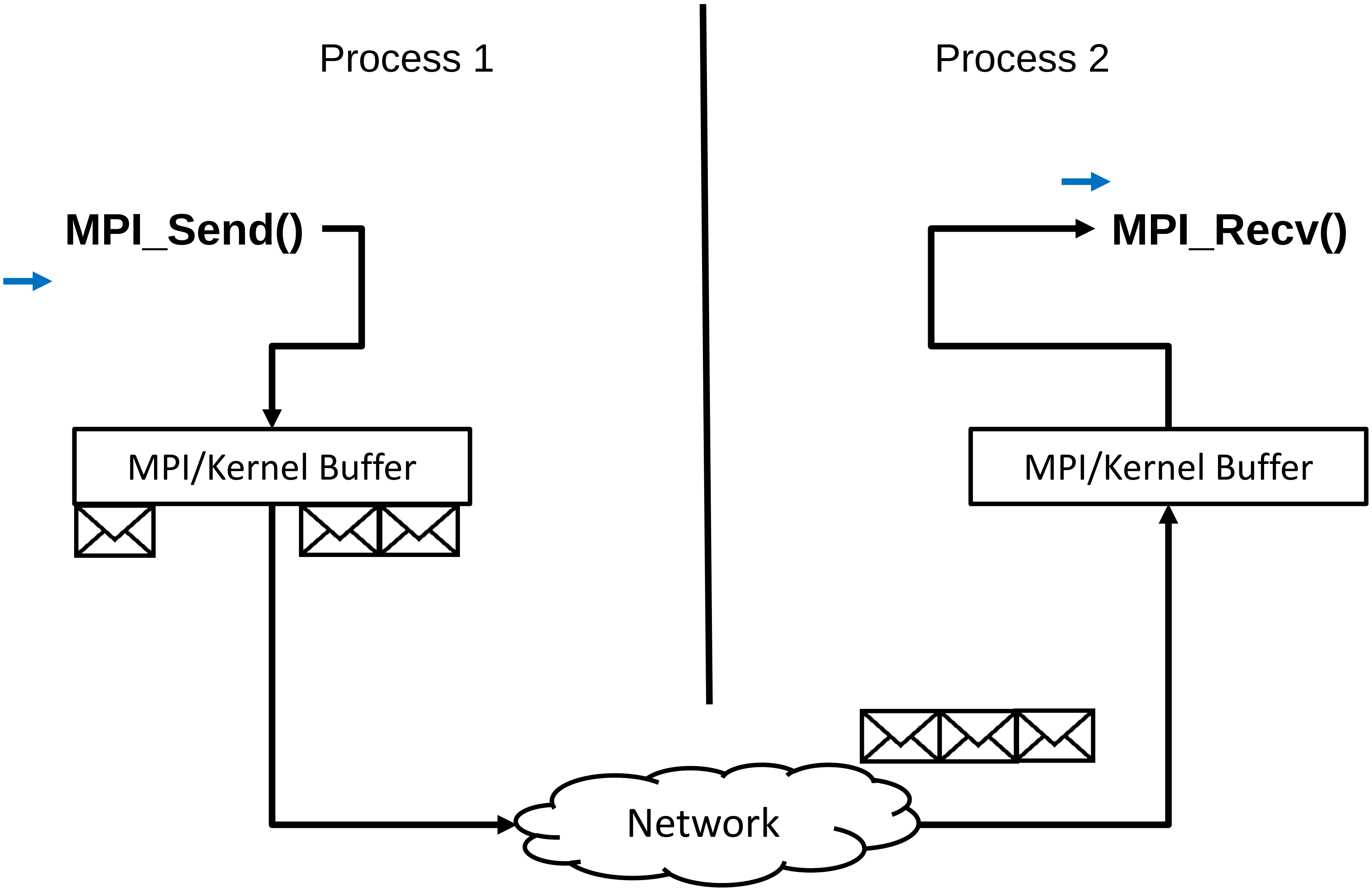


MPI_Send() can block if the local buffer is full

MPI blocking recv/blocking send

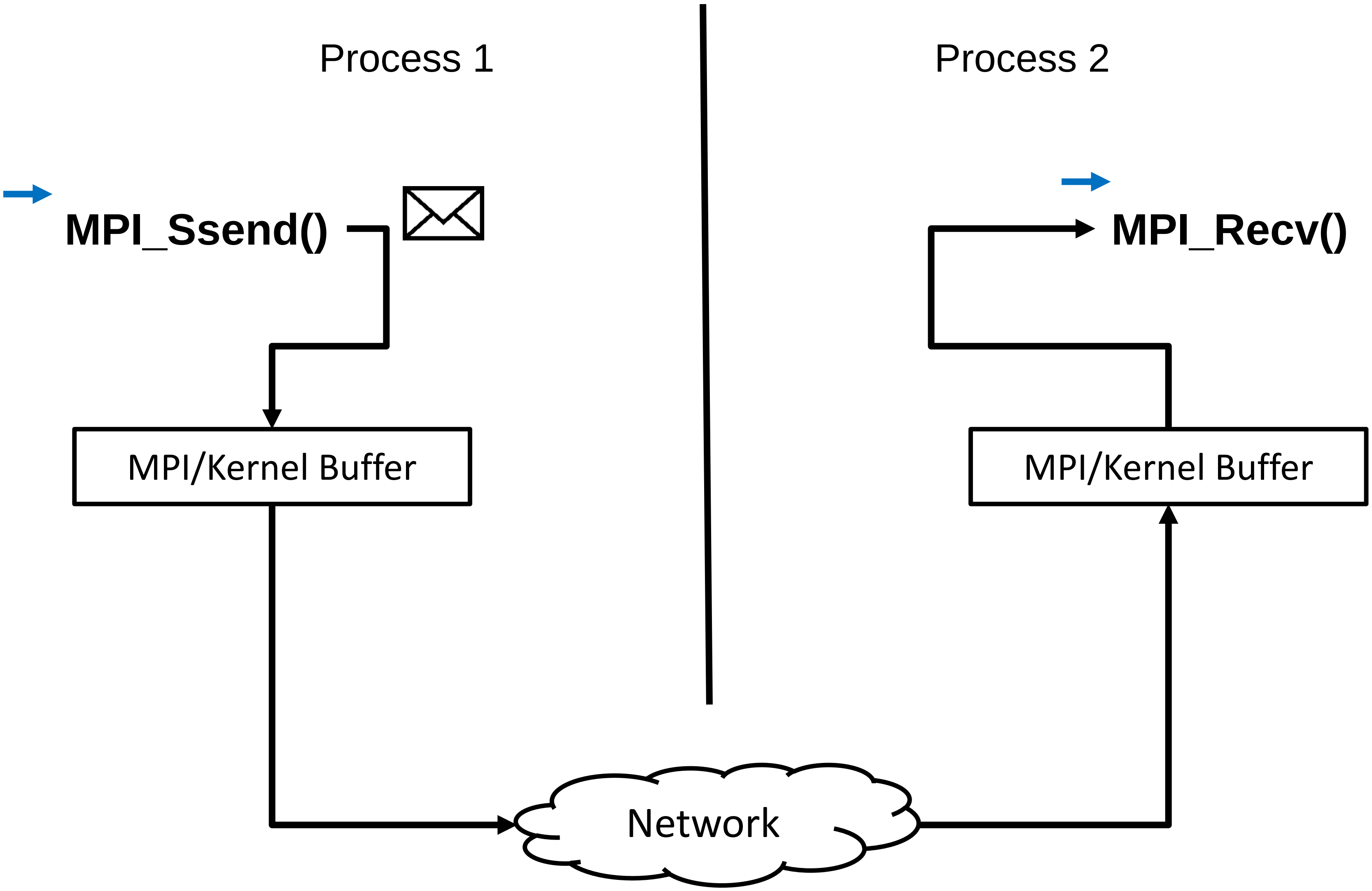


MPI blocking recv/blocking send

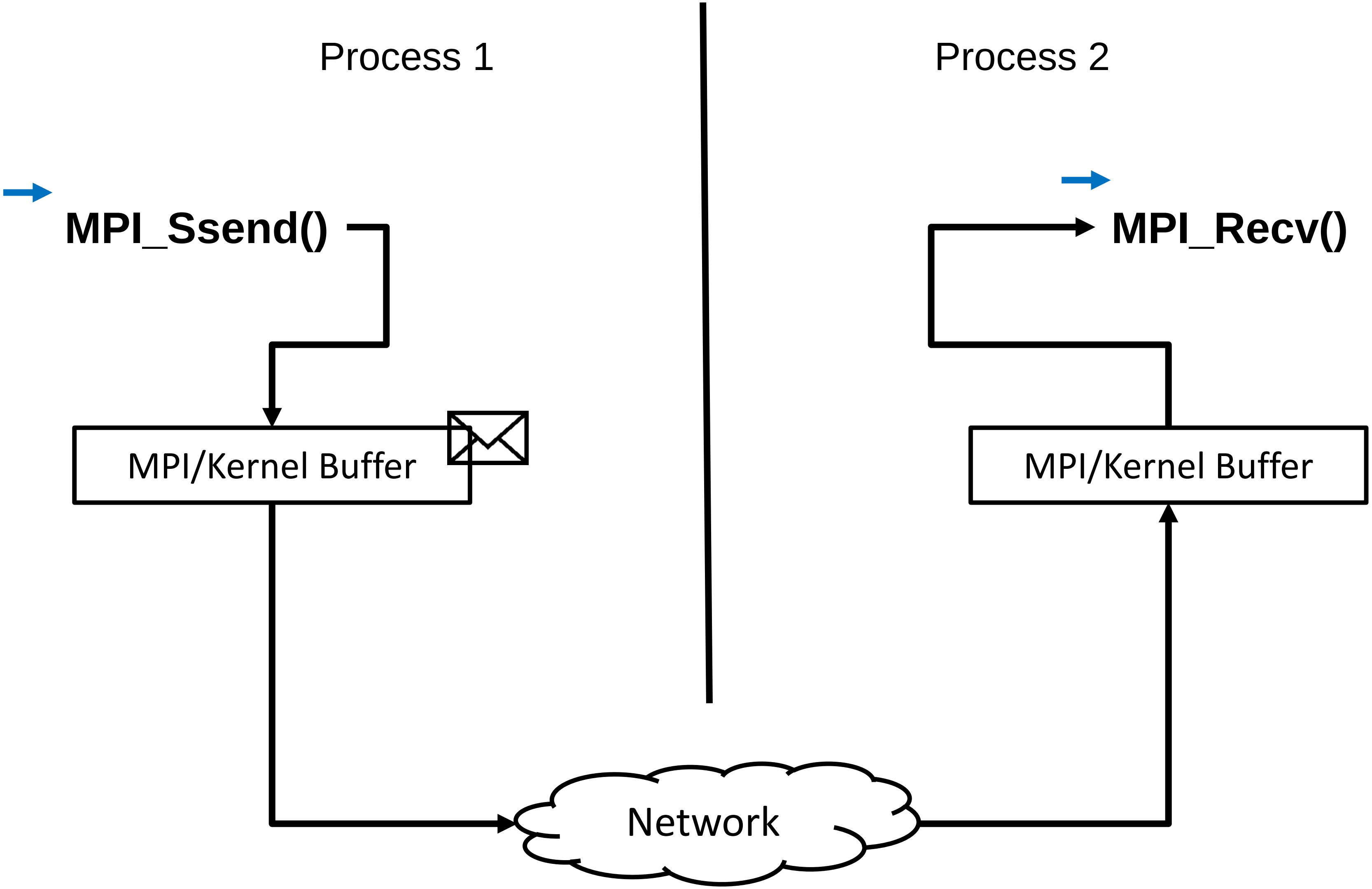


MPI blocking recv/synchronized send

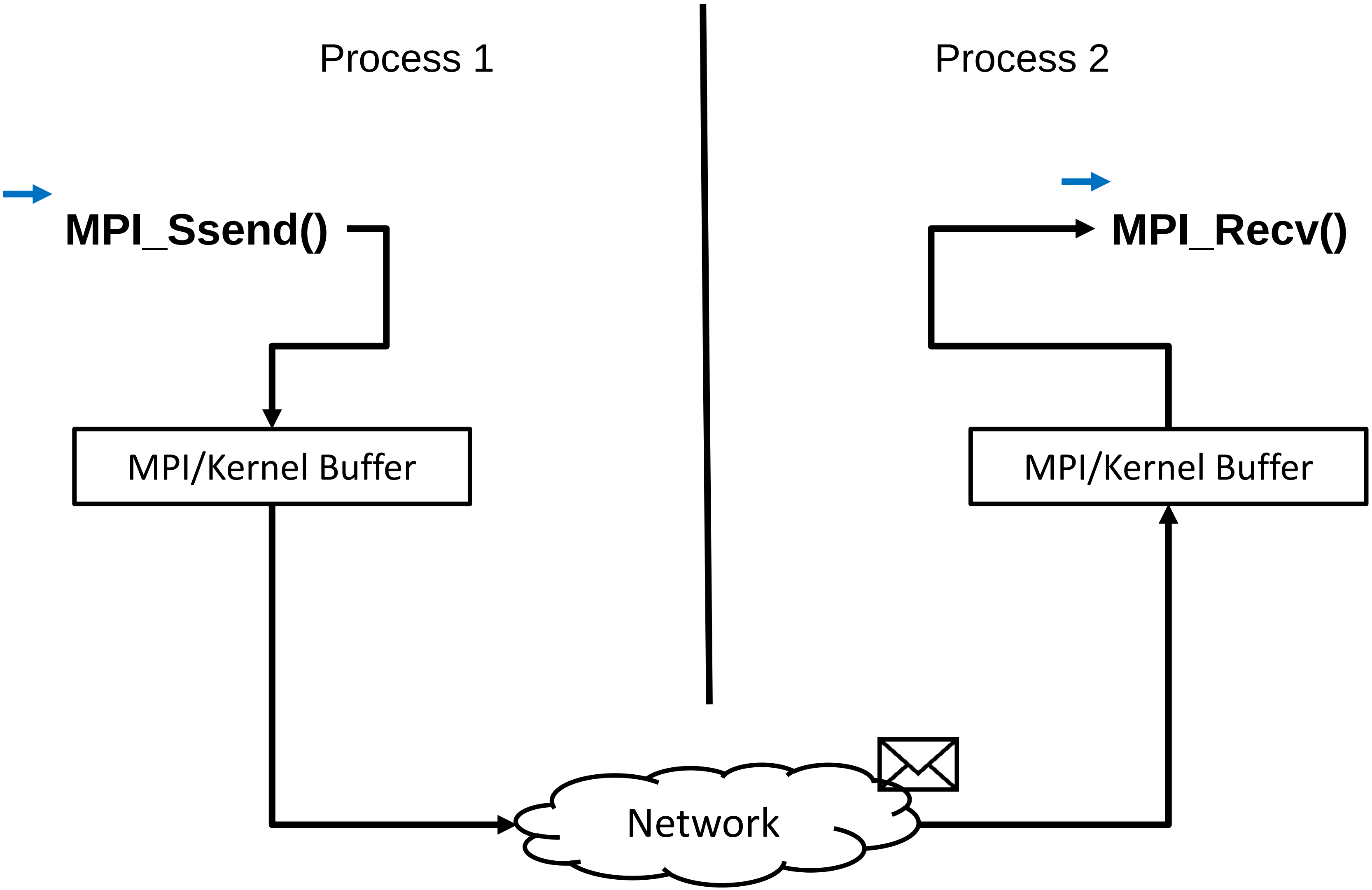
MPI blocking recv/synchronized send



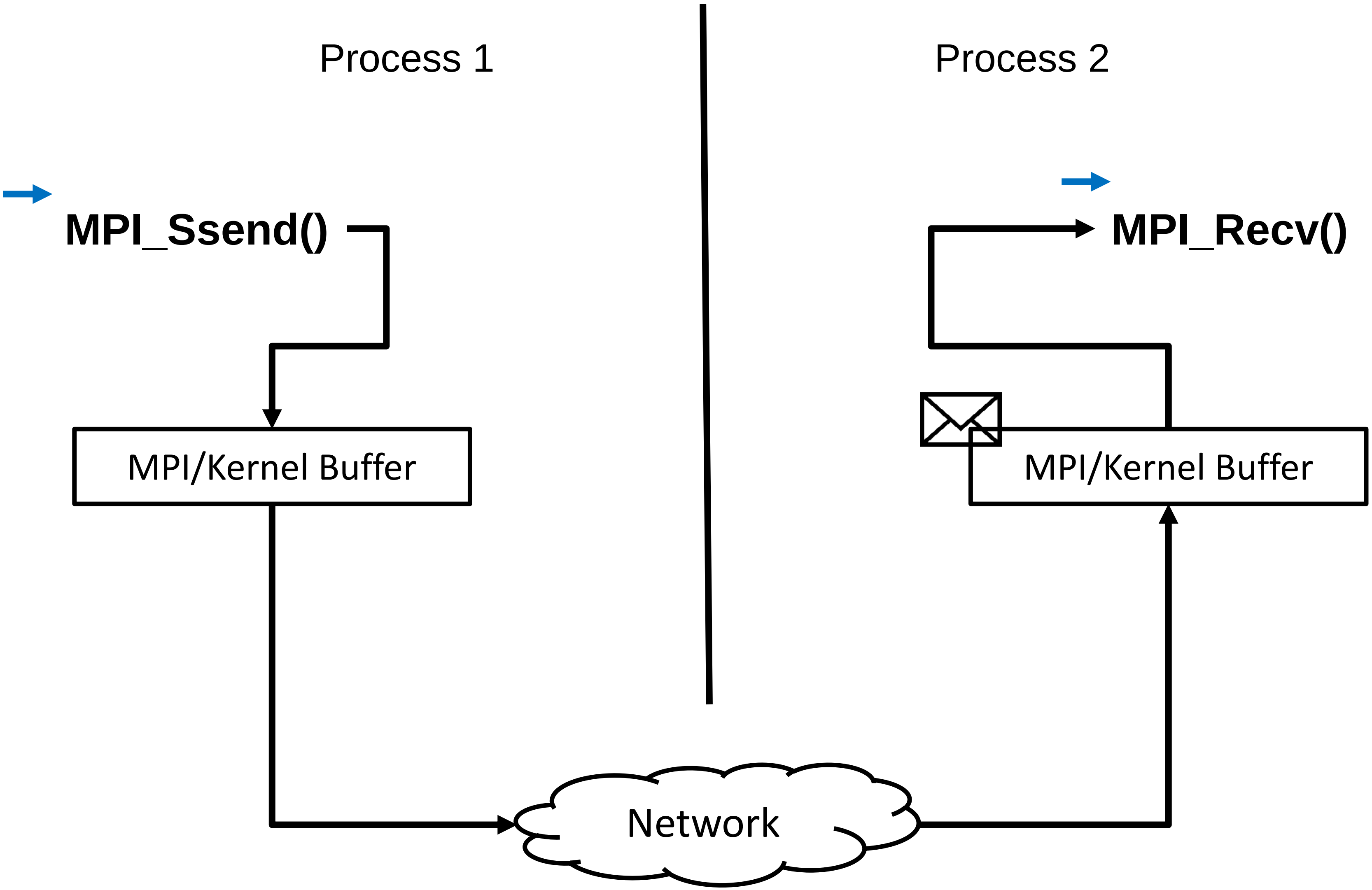
MPI blocking recv/synchronized send



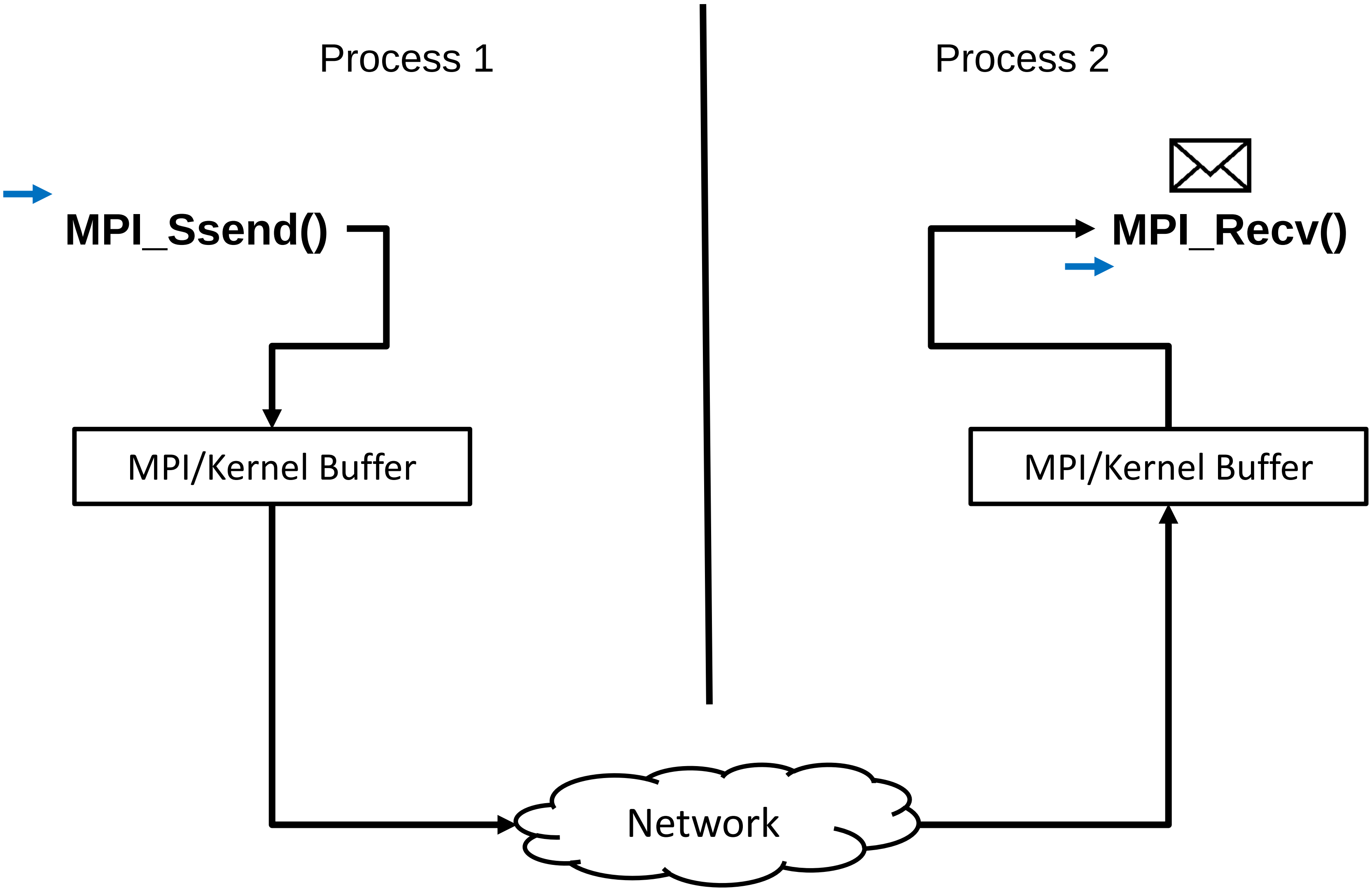
MPI blocking recv/synchronized send



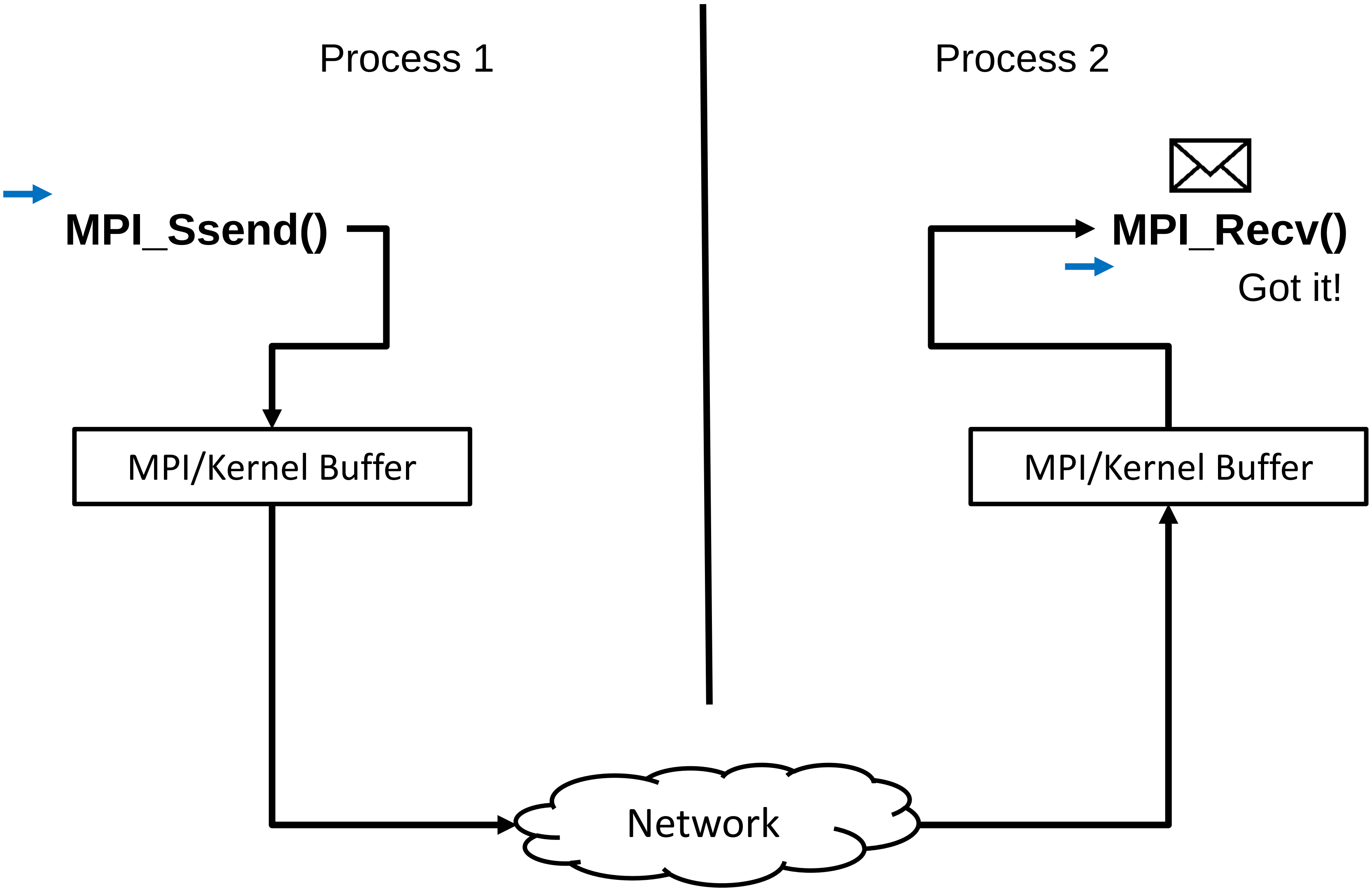
MPI blocking recv/synchronized send



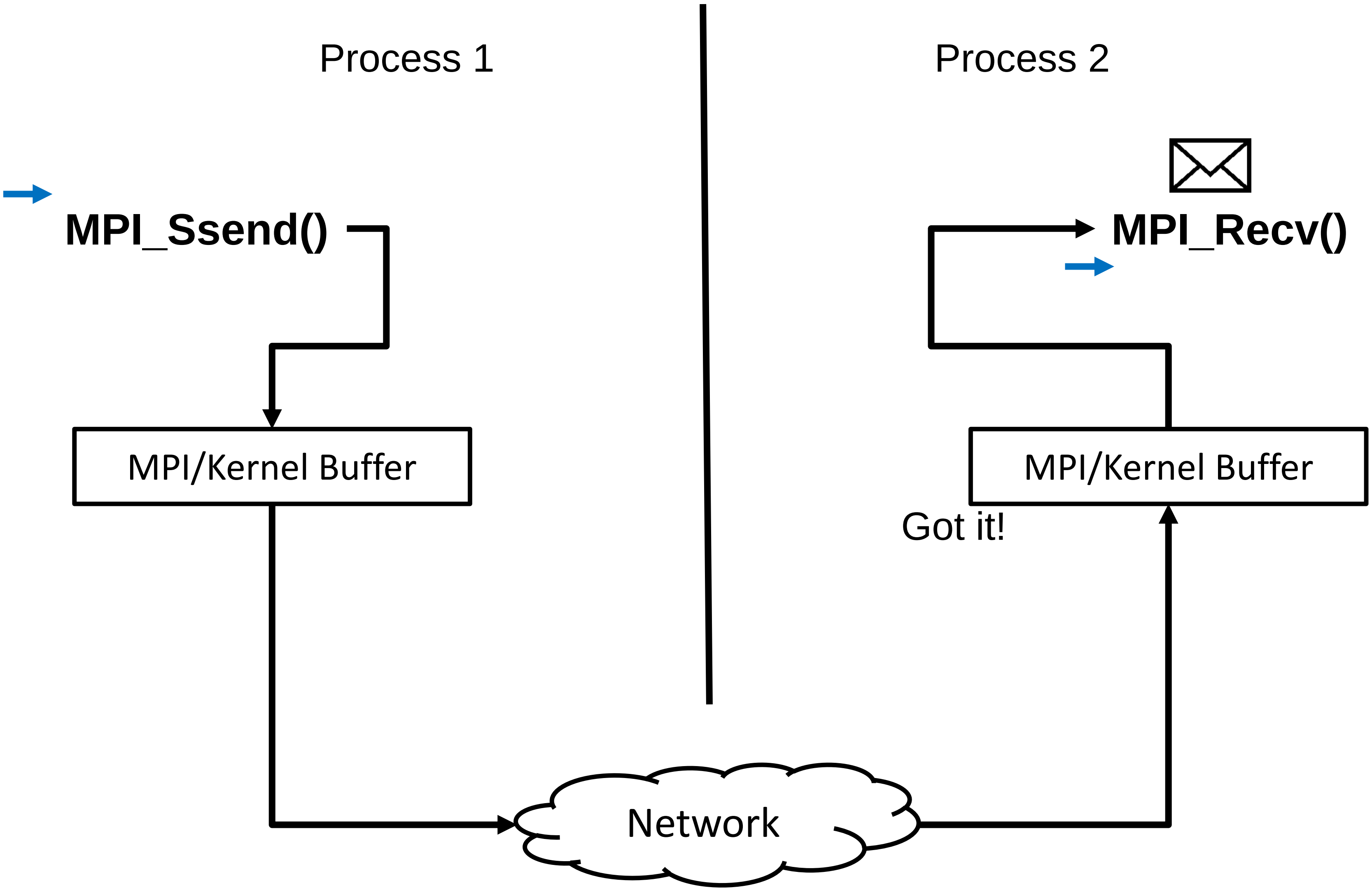
MPI blocking recv/synchronized send



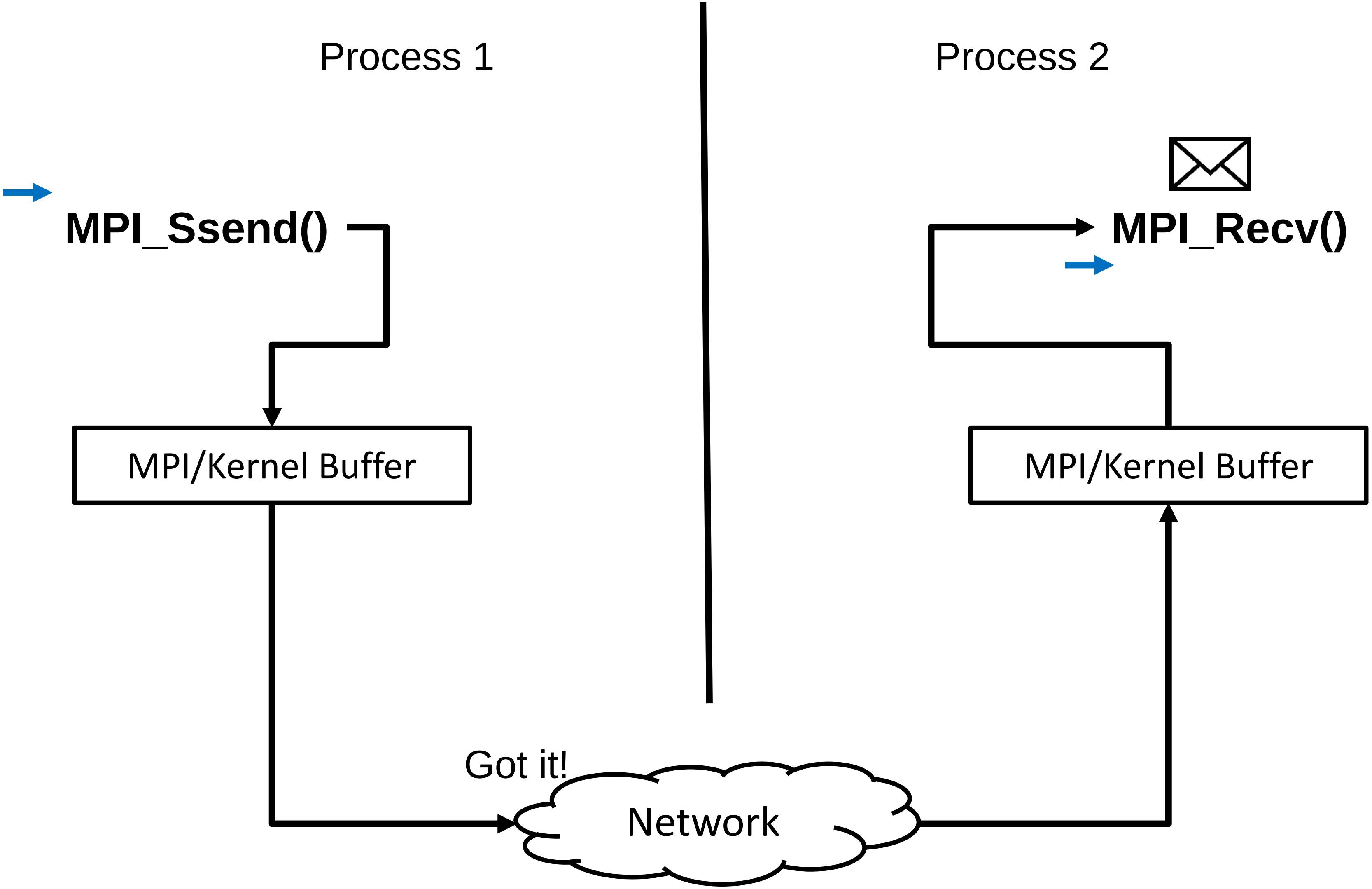
MPI blocking recv/synchronized send



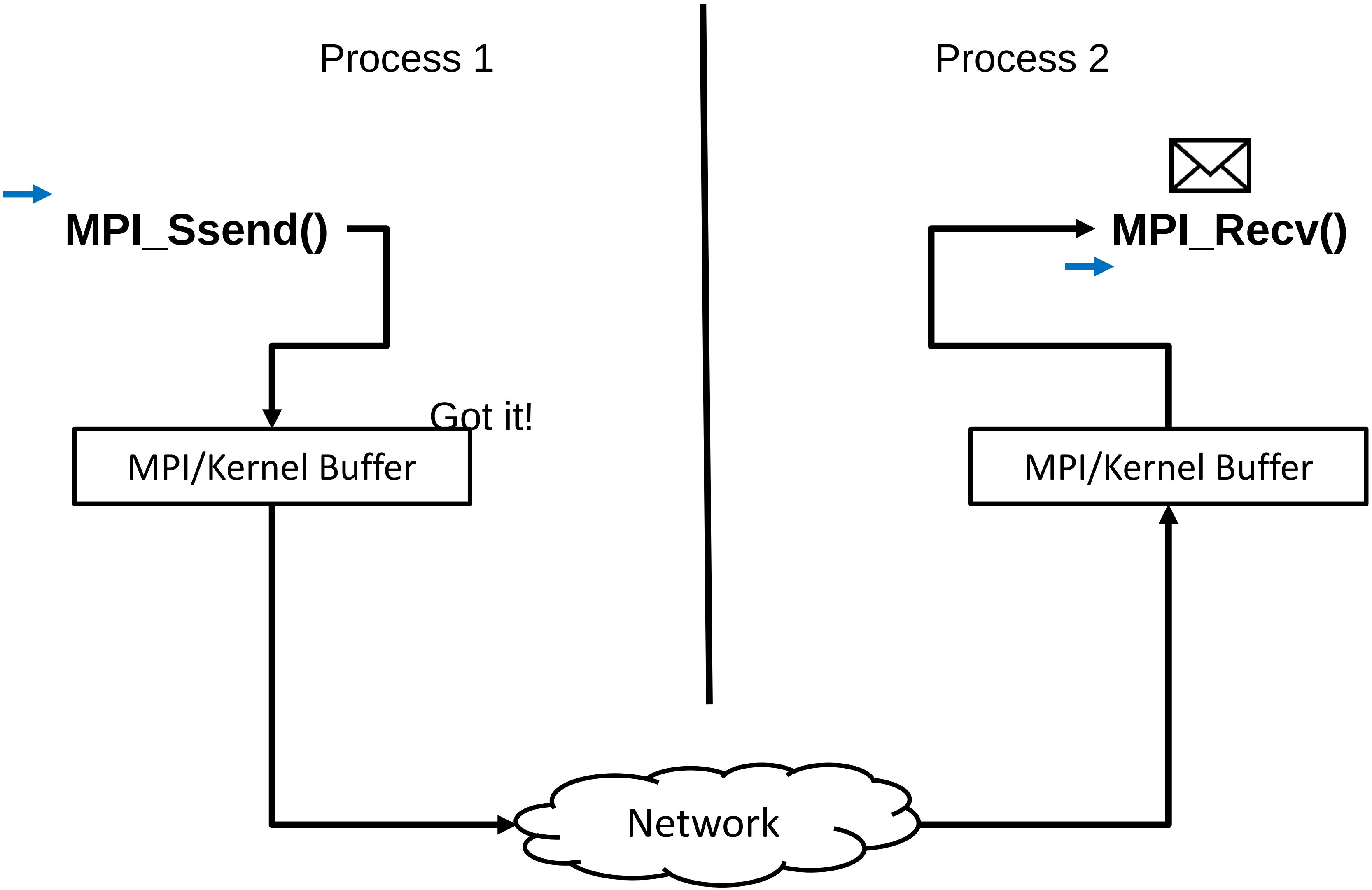
MPI blocking recv/synchronized send



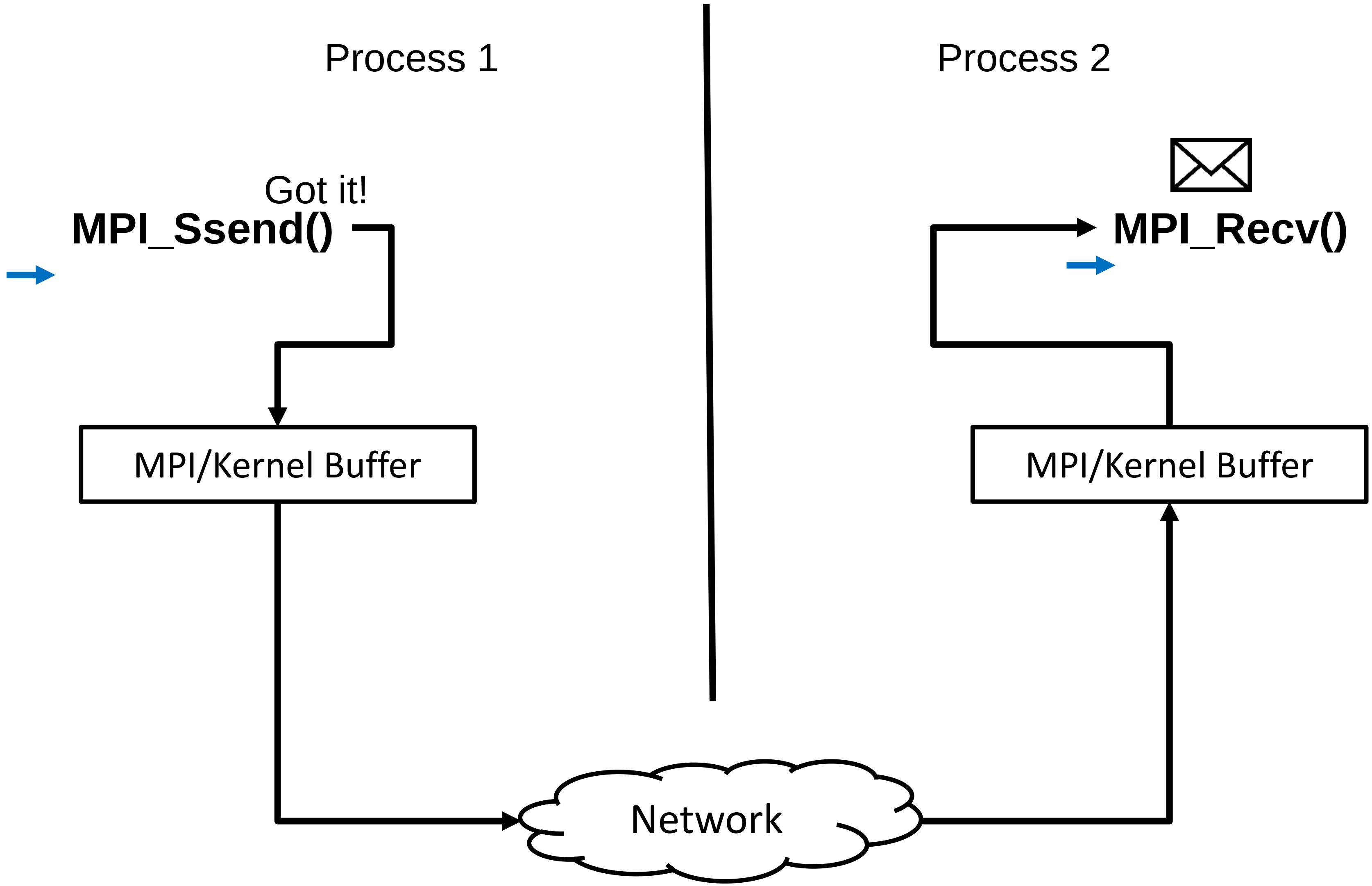
MPI blocking recv/synchronized send



MPI blocking recv/synchronized send

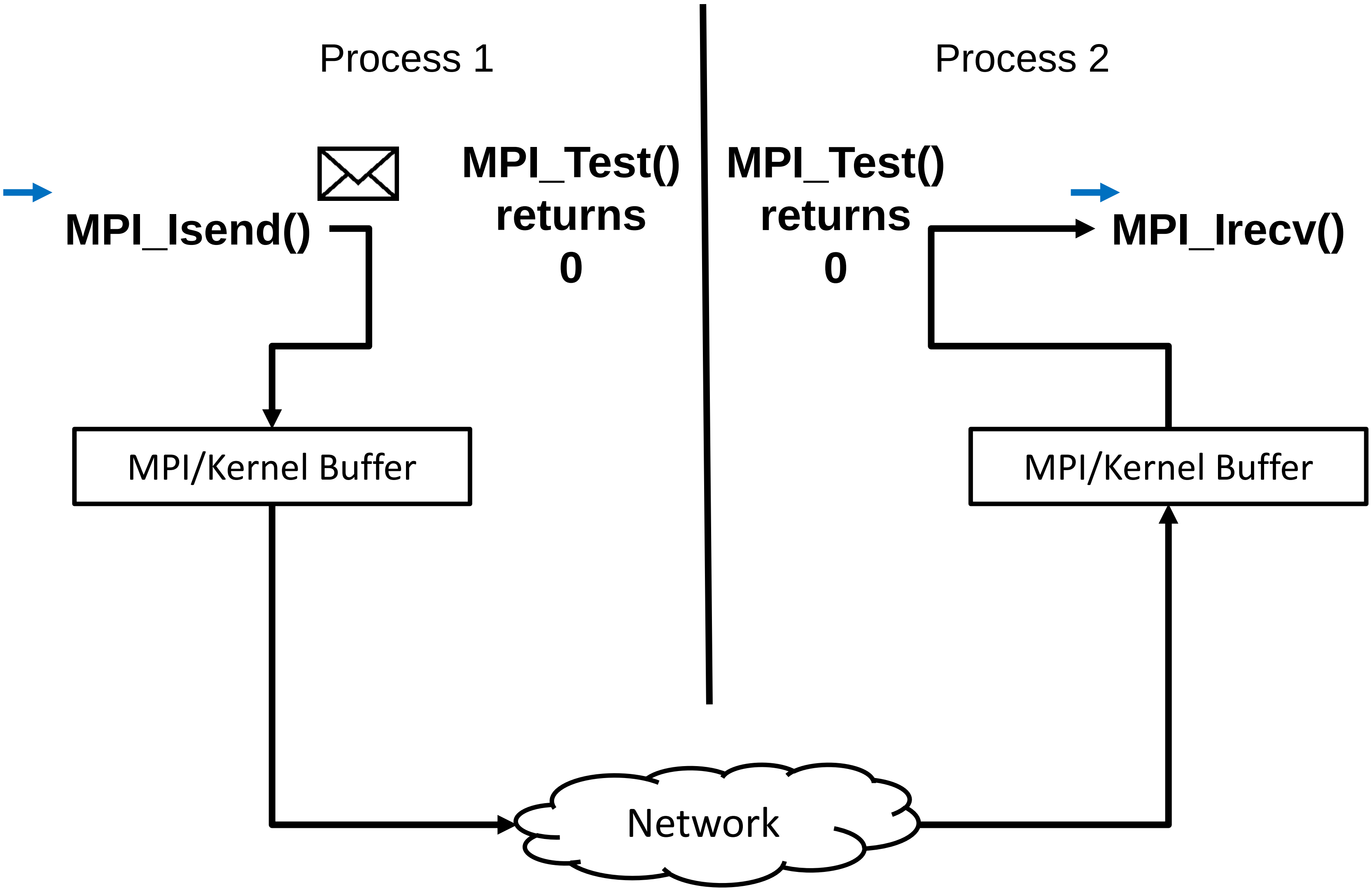


MPI blocking recv/synchronized send

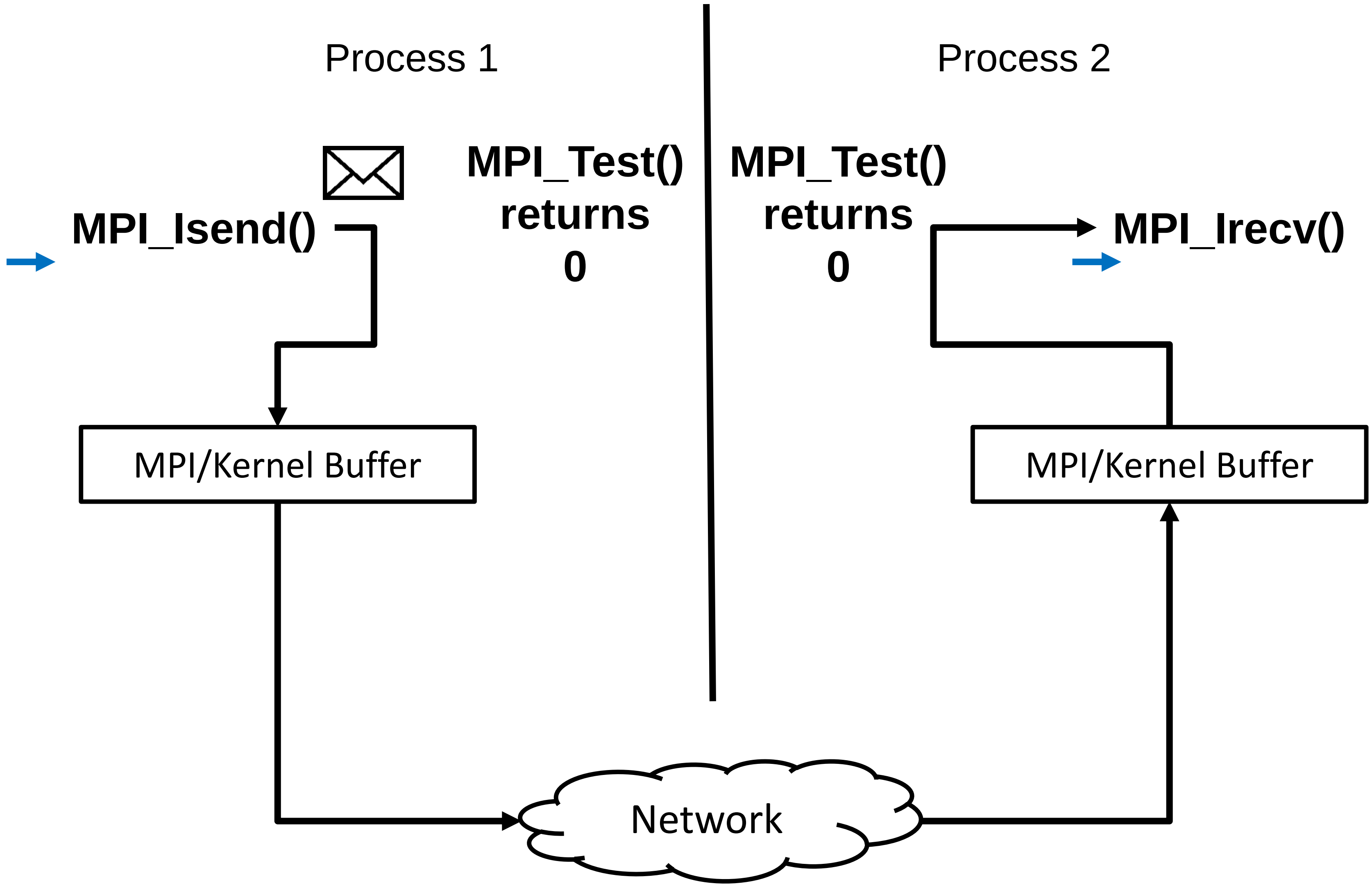


MPI non-blocking recv/non-blocking send

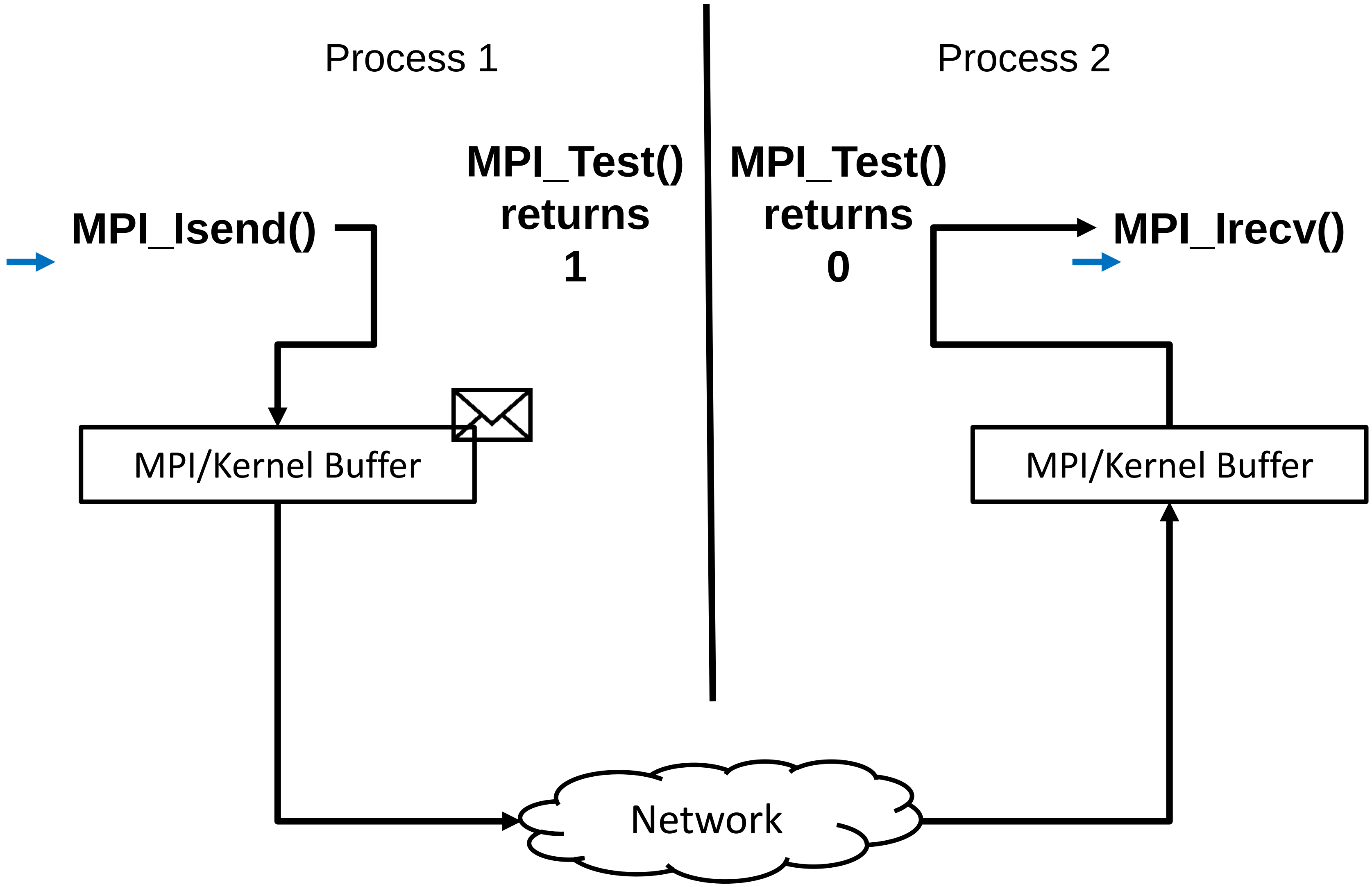
MPI non-blocking recv/non-blocking send



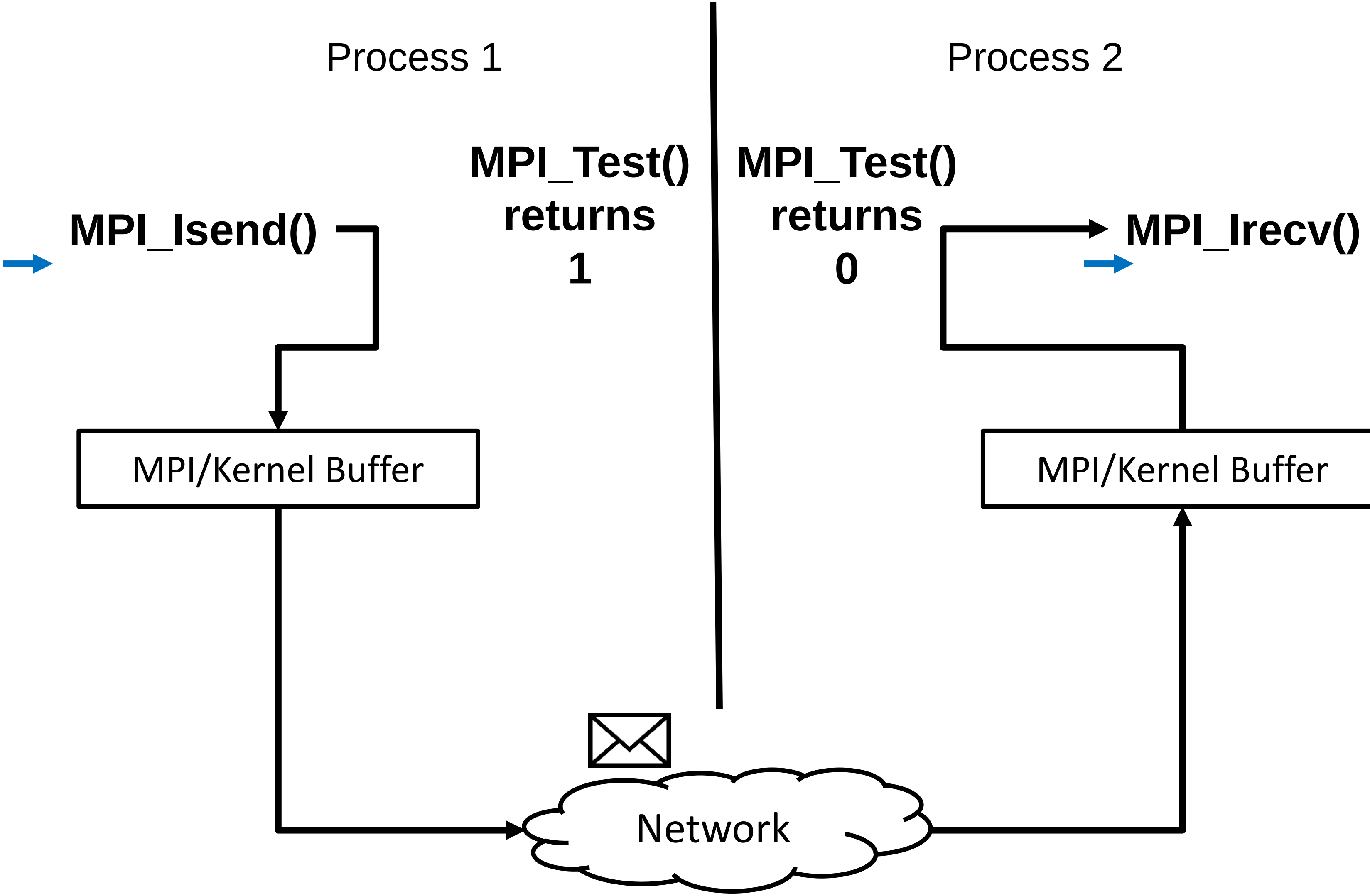
MPI non-blocking recv/non-blocking send



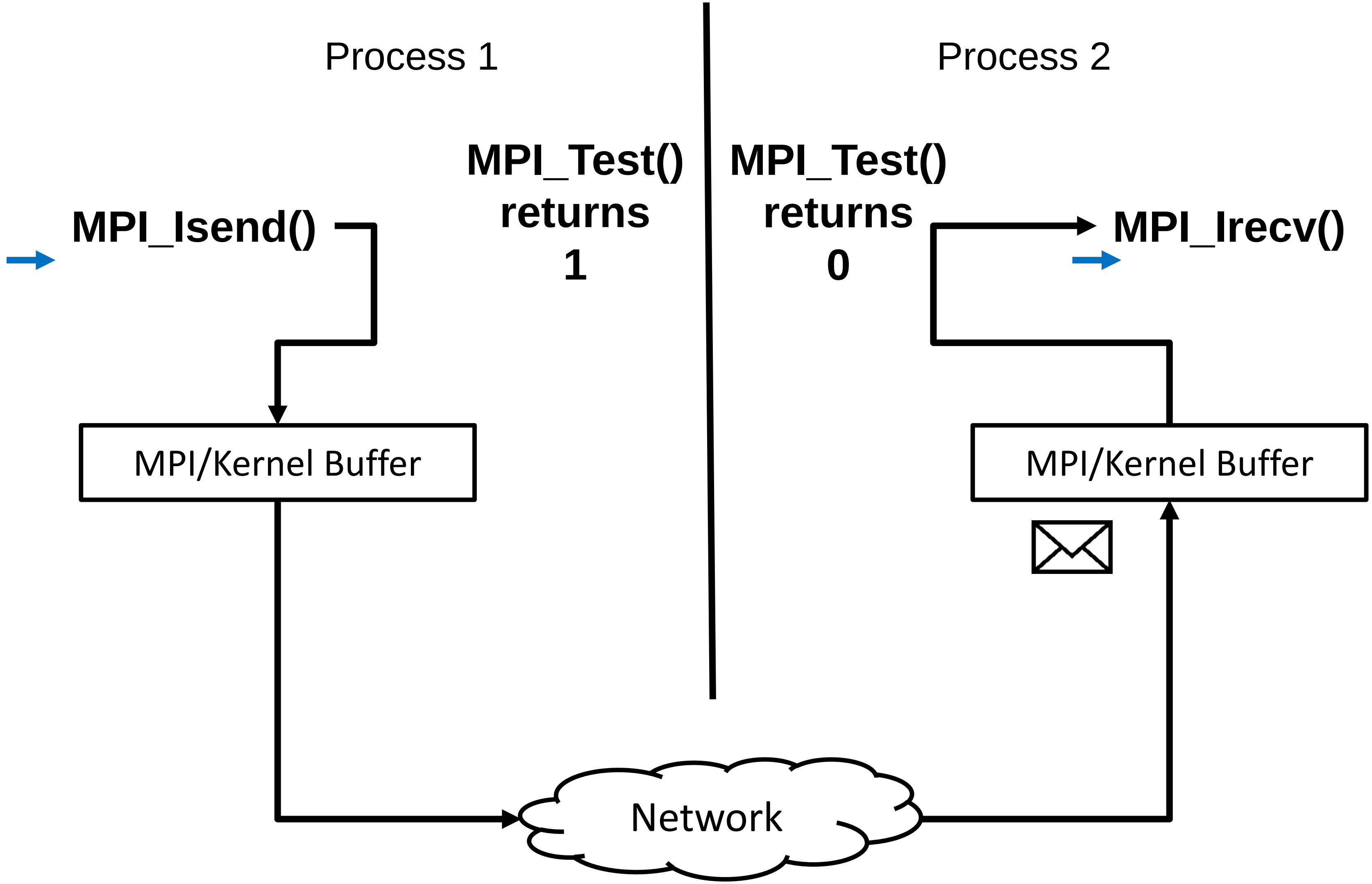
MPI non-blocking recv/non-blocking send



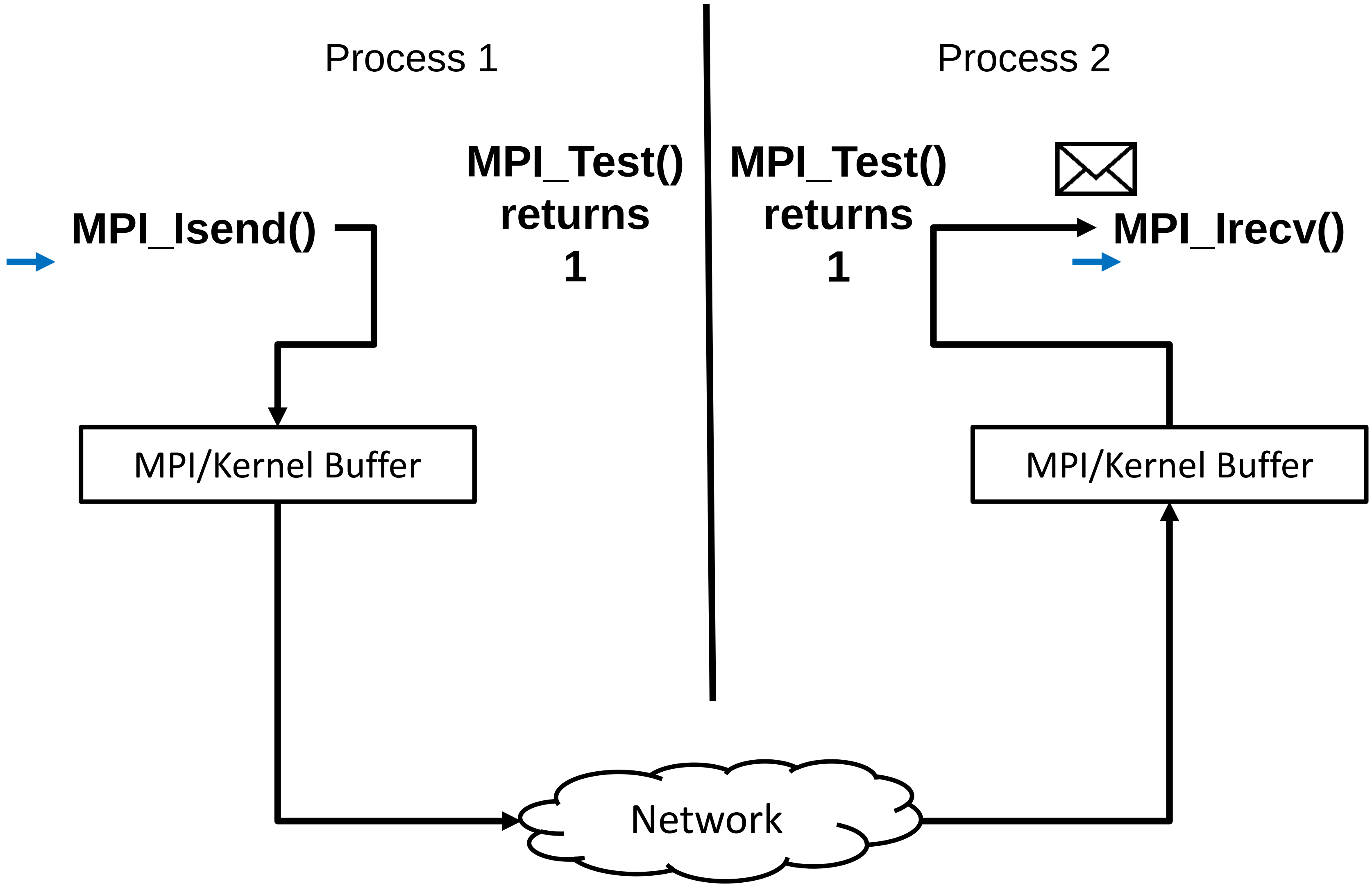
MPI non-blocking recv/non-blocking send



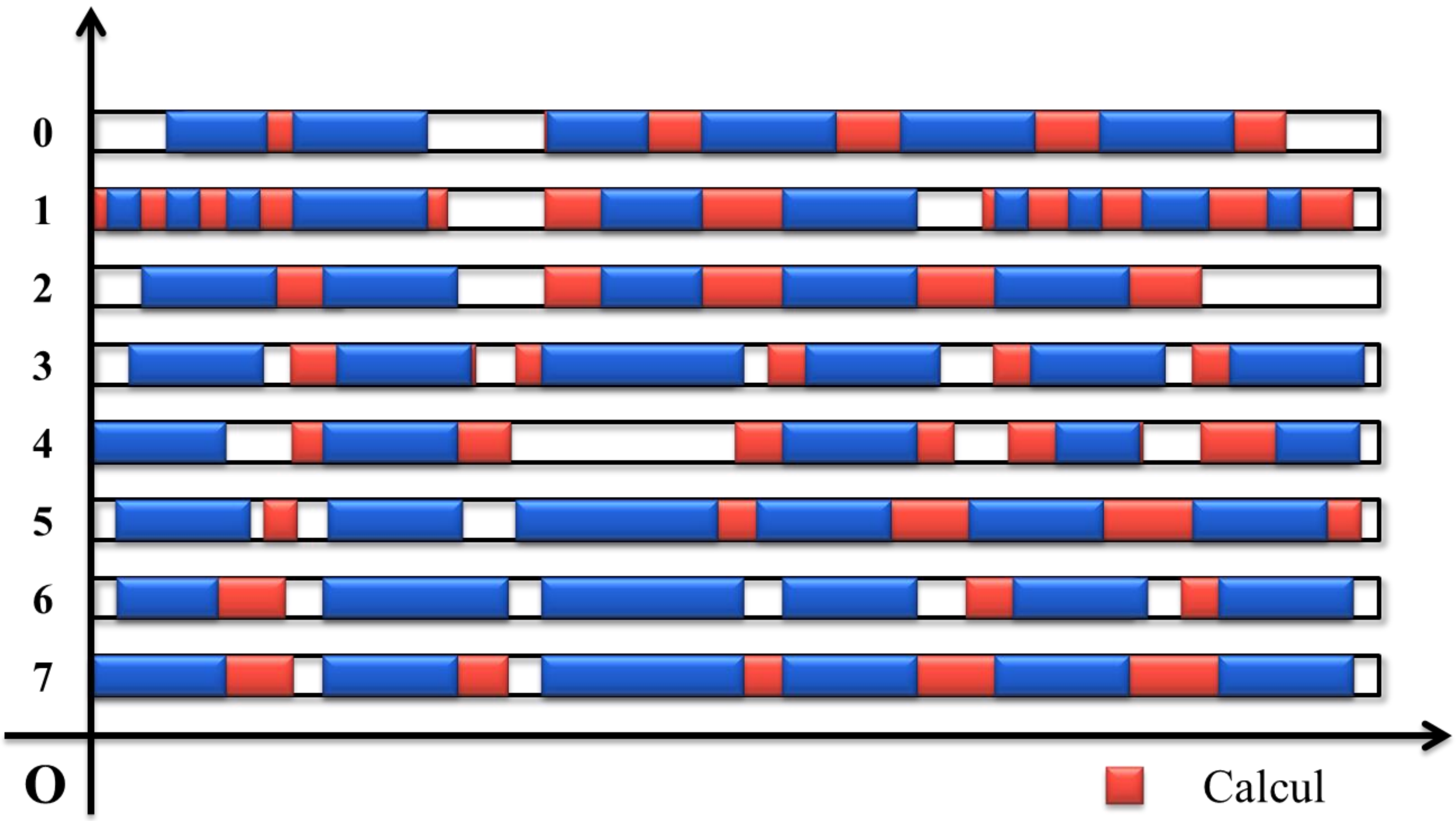
MPI non-blocking recv/non-blocking send



MPI non-blocking recv/non-blocking send



Foster model



(Foster 1995)

Foster model

- Definiţie
  - Timpul scurs de la începerea execuţiei primului proces până la terminarea execuţiei ultimului proces.

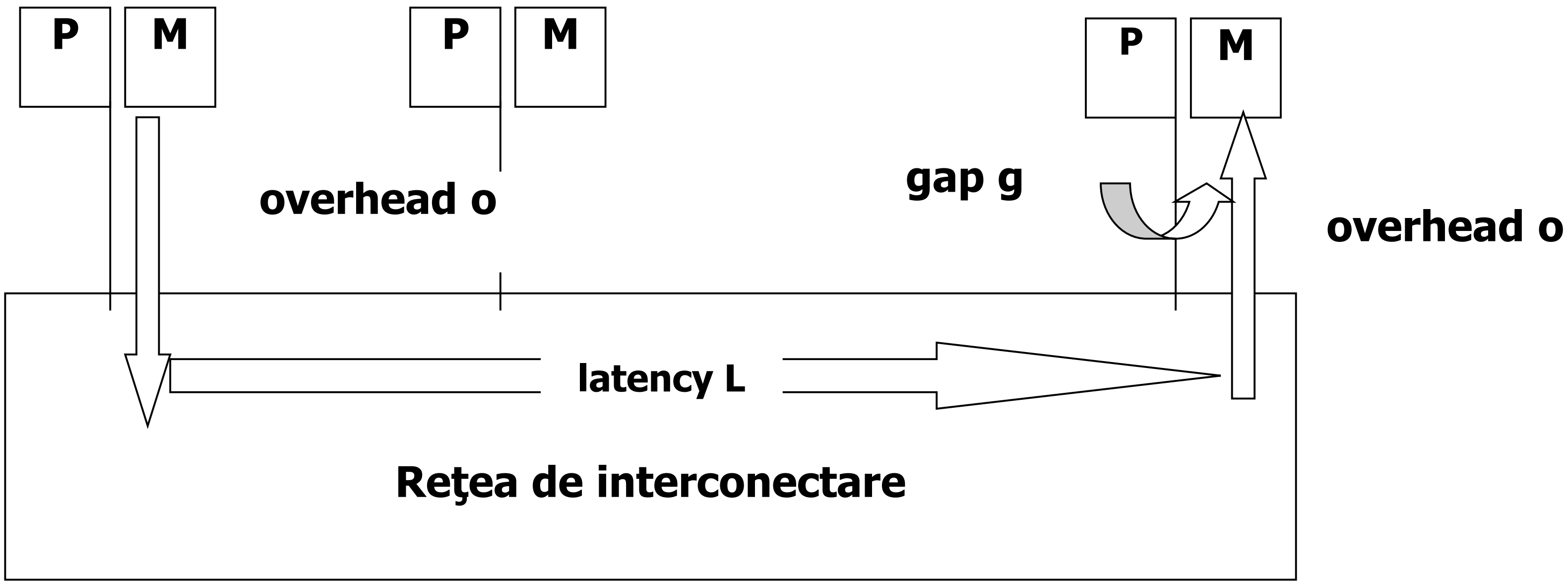
$$T = f\,(\,N,\ P,\ U, \ldots)$$

$$= T^{j}{}_{comp} + T^{j}{}_{commun} + T^{j}{}_{idle}$$

$$= \left(\frac{1}{P}\right) * \left(\sum_{i=0}^{P-1} T^{i}{}_{comp} + \sum_{i=0}^{P-1} T^{i}{}_{commun} + \sum_{i=0}^{P-1} T^{i}{}_{idle}\right)$$

$$= \left(\frac{1}{P}\right) * \left(T_{comp} + T_{commun} + T_{commun}\right)$$

LogP model



L - latency sau întârzierea de transmitere a unui mesaj mic de la sursă la destinatar

o - overhead, durata pentru care procesorul este angajat în transmiterea sau recepţia fiecărui mesaj

g - gap, intervalul minim de timp între două transmiteri succesive sau două recepţii succesive la acelaşi modul

P - numărul de module procesor / memorie.