



# Sisteme Tolerante la Defecte Introducere MPI

Lect. Dr. Ing. Cristian Chilipirea – [cristian.chilipirea@mta.ro](mailto:cristian.chilipirea@mta.ro)



# MPI

## Framework care facilitează

- Pornirea programelor distribuite (processe pe același sistem sau pe sisteme diferite, dar strâns conectate – ideal aceeași rețea)
- Conectarea proceselor unui program distribuit (accept, bind, connect)
- Simplificarea identificării (identificatori în loc de IP, port)
- Simplificarea comunicării (oferă funcții gen Send/Recv, Broadcast)
- Asigură comunicarea corectă pe sisteme cu arhitecturi de calcul diferite (little/big endian problems)



# MPI memoria

- Nu avem memorie partajată în MPI. Arhitectură NUMA
- Toate variabilele sunt locale proceselor.
- Pentru a muta informație de la un proces la altul vor trebuie folosită comunicație, prin apelul funcțiilor oferite de MPI:
  - ❑ Send/Recv
  - ❑ Broadcast
  - ❑ Scatter
  - ❑ Gather



# Instalare 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`

`./a.out`

Pornește **4 procese**.  
Dacă este setat, va porni  
procese pe mașini  
diferite.

Procese sunt identice  
dar au id-uri diferite.  
Funcționează parțial și  
cu programe care nu  
sunt implementate  
pentru MPI.

Funcționează dar  
pornește **un singur**  
proces.



# 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);
```

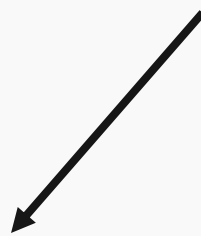
```
    printf("Hello from %i/%i\n", rank, nProcesses);
```

```
    MPI_Finalize();
```

```
    return 0;
```

```
}
```

Pornește procesele MPI







# 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;
```

```
}
```

Întoarce ID-ul  
procesului (rank-ul)







# 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;
```

```
}
```

Întoarce numărul total  
de procese





# 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;
```

```
}
```

←  
Afișează hello (pentru  
fiecare proces pornit).



# 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;
```

```
}
```

← Oprește programul  
MPI.



# 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\_Send/MPI\_Recv

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

v	num_el(v)	[ 0, num_tasks )	MPI_COMM_WORLD
&v[3]	[0,..)		
&a			
v+5			[ 0, .. )
	MPI_INT		
	MPI_CHAR		
	MPI_FLOAT		
	MPI_LONG		



# 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 num\_el(v)  
v+5 [0,..)

[ 0, .. )  
MPI\_ANY\_TAG

[ 0, num\_tasks )  
MPI\_ANY\_SOURCE

MPI\_INT  
MPI\_CHAR  
MPI\_FLOAT  
MPI\_LONG

MPI\_COMM\_WORLD

&Stat  
MPI\_STATUS\_IGNORE  
**Stat.MPI\_SOURCE, Stat.MPI\_TAG**





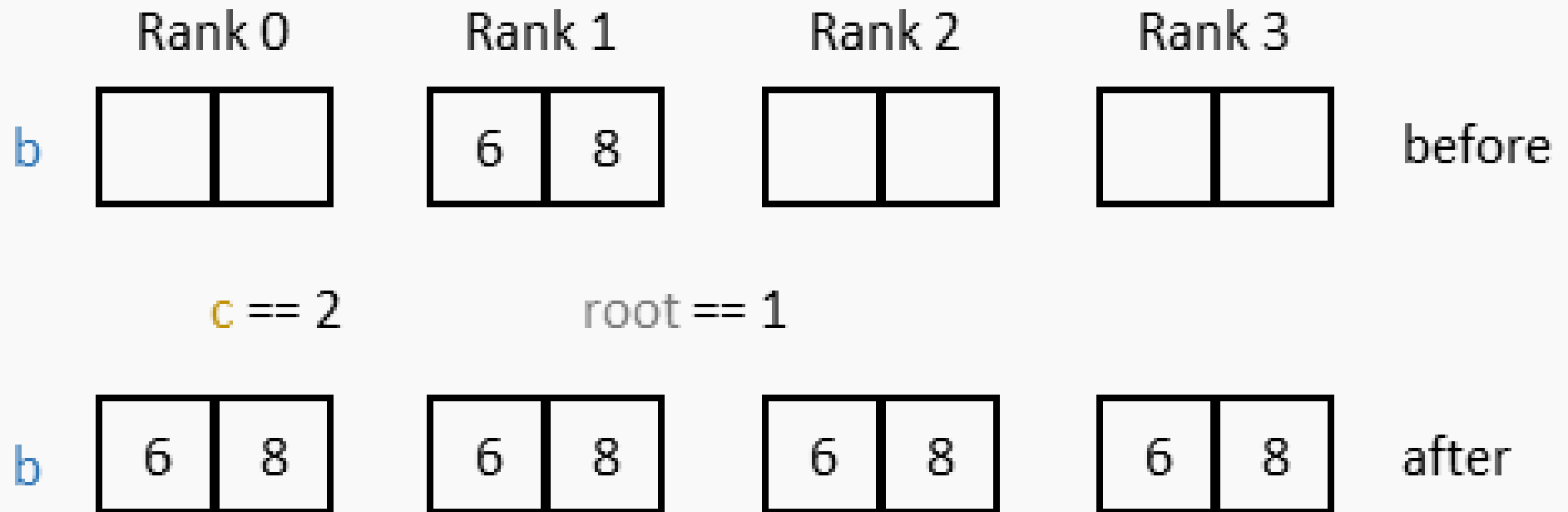
# MPI\_Bcast

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

<code>v</code>	<code>num_el(v)</code>	<code>[ 0, num_tasks )</code>	
<code>&amp;v[3]</code>	<code>[0,..)</code>		
<code>&amp;a</code>		<code>MPI_INT</code>	<code>MPI_COMM_WORLD</code>
<code>v+5</code>		<code>MPI_CHAR</code>	
		<code>MPI_FLOAT</code>	
		<code>MPI_LONG</code>	



# MPI\_Bcast





# MPI\_Scatter

int MPI\_Scatter ( ↓ void \*sb, ↓ int sc, ↓ MPI\_Datatype sd, ↑ void \*rb, ↓ int rc, ↓ MPI\_Datatype rd, ↓ int root, ↓ MPI\_Comm )

[ 0, num\_tasks )

num\_el(v)/num\_tasks  
[0,..)

num\_el(v)/num\_tasks  
[0,..)

v  
&v[3]  
&a  
v+5

MPI\_INT  
MPI\_CHAR  
MPI\_FLOAT  
MPI\_LONG

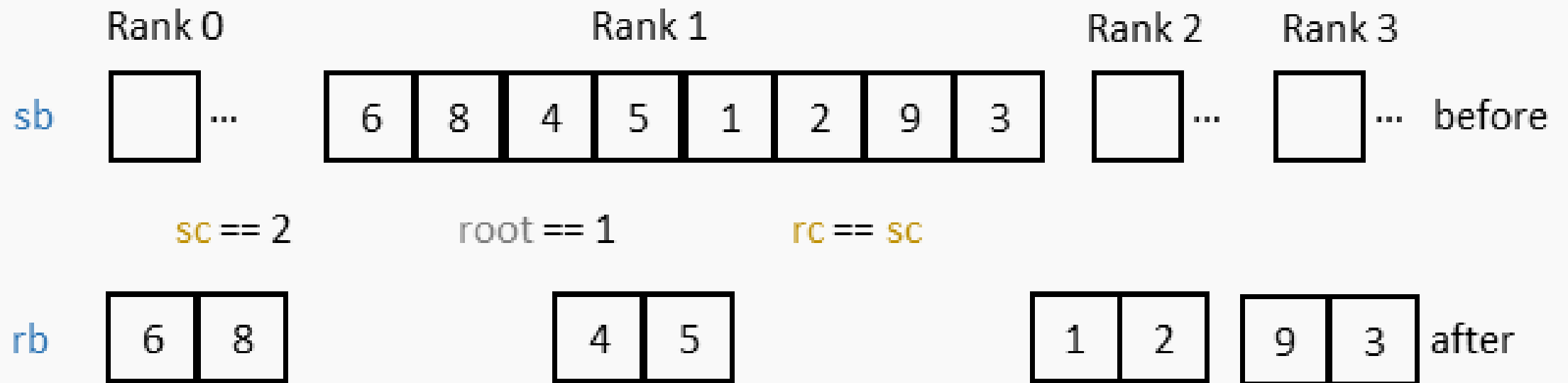
v  
&v[3]  
&a  
v+5

MPI\_INT  
MPI\_CHAR  
MPI\_FLOAT  
MPI\_LONG

MPI\_COMM\_WORLD



# MPI\_Scatter





# MPI\_Gather

int MPI\_Gather ( ↓ void \*sb, ↓ int sc, ↓ MPI\_Datatype sd, ↑ void \*rb, ↓ int rc, ↓ MPI\_Datatype rd, ↓ int root, ↓ MPI\_Comm )

num\_el(v)/num\_tasks  
[0,..)

v  
&v[3]  
&a  
v+5

num\_el(v)/num\_tasks [ 0, num\_tasks )  
[0,..)

v  
&v[3]  
&a  
v+5

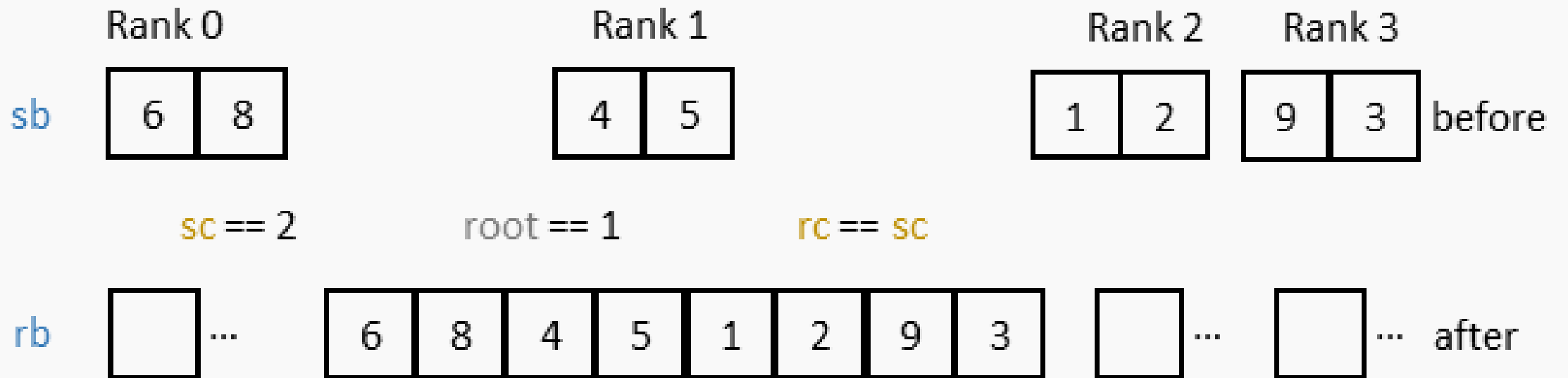
MPI\_INT  
MPI\_CHAR  
MPI\_FLOAT  
MPI\_LONG

MPI\_INT  
MPI\_CHAR  
MPI\_FLOAT  
MPI\_LONG

MPI\_COMM\_WORLD



# MPI\_Gather









# MPI blocking/non-blocking send/recv

Proces 1

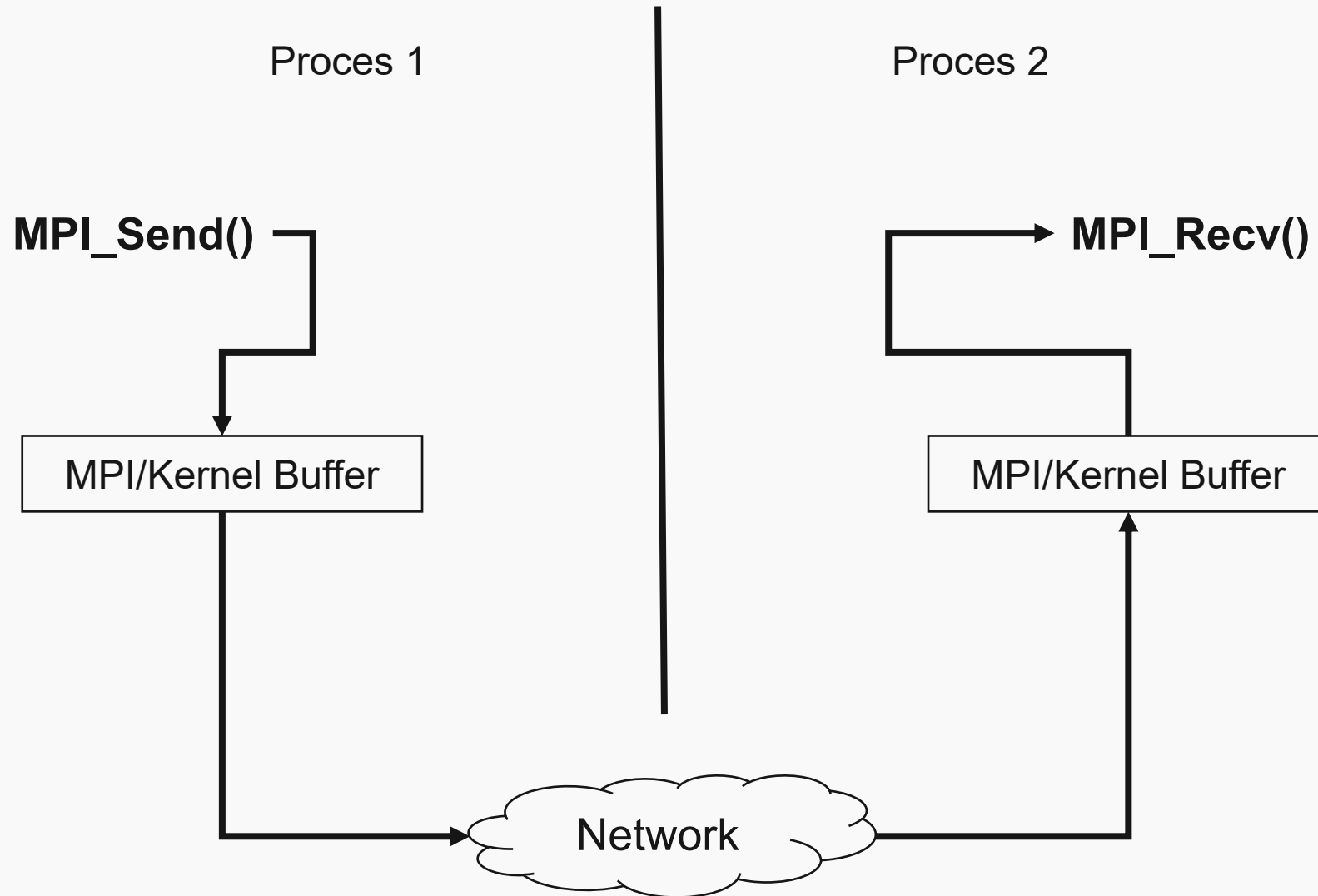
Proces 2

**MPI\_Send()**

**MPI\_Recv()**

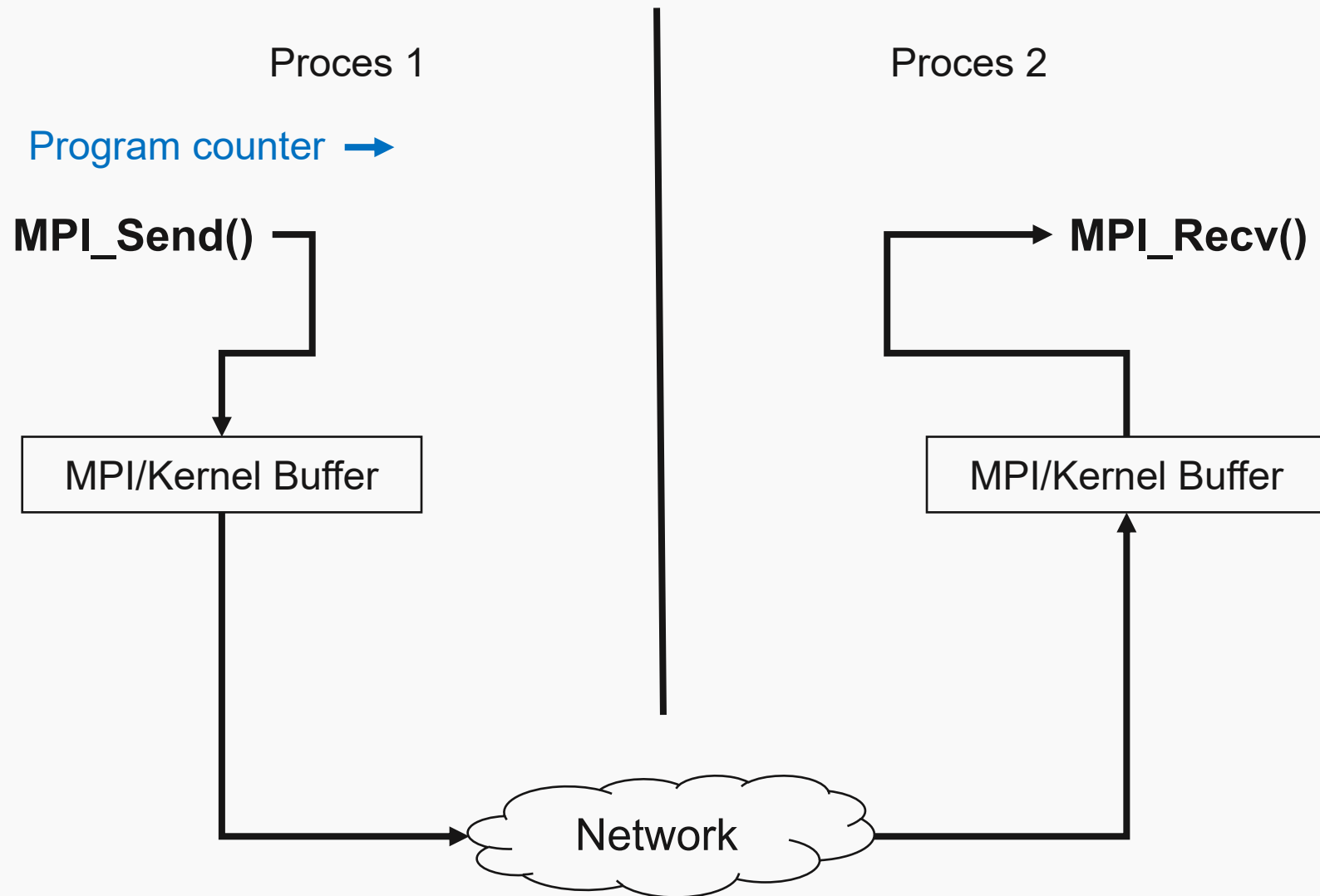


# MPI blocking/non-blocking send/recv



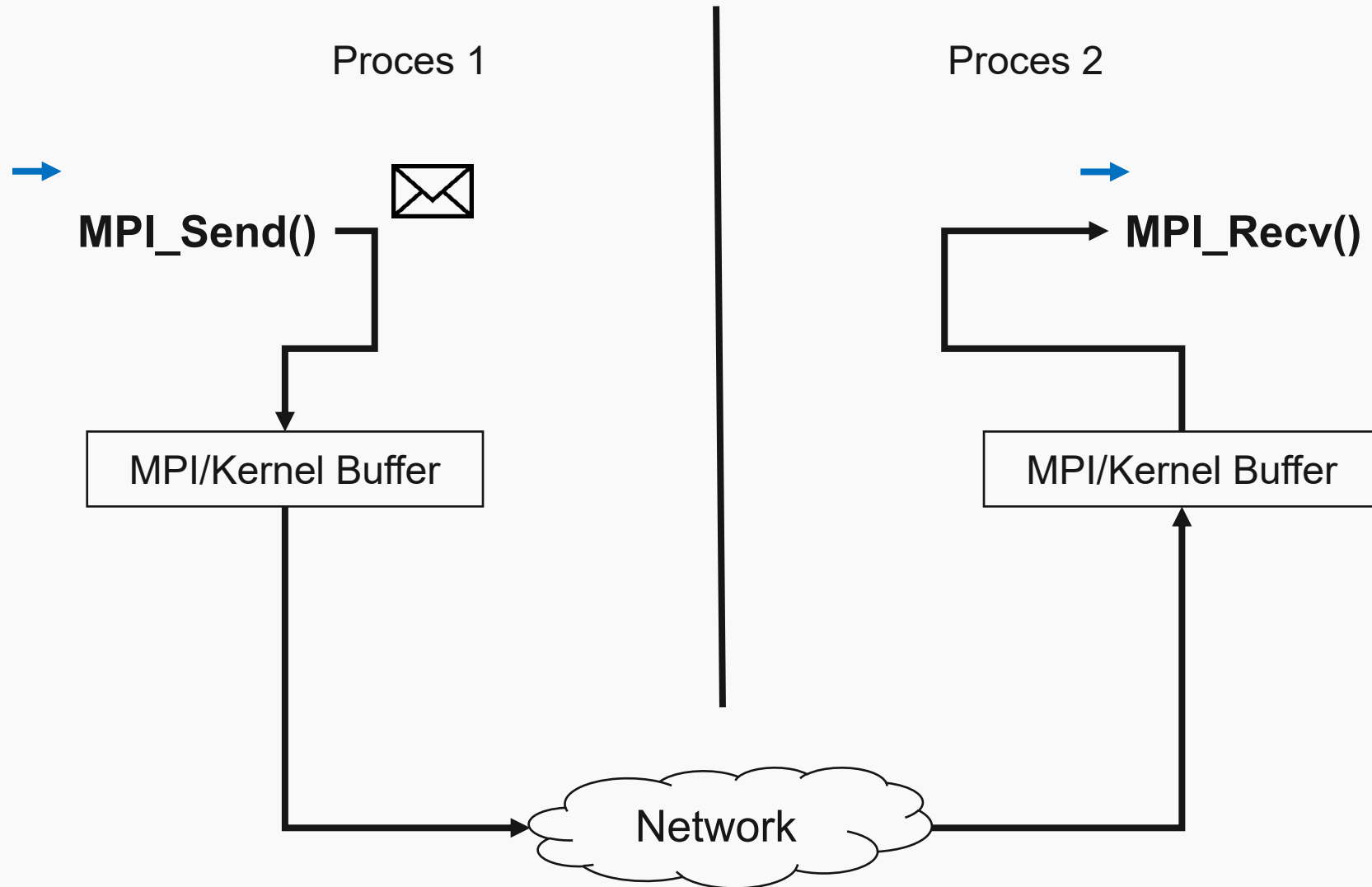


# MPI blocking recv/non-blocking send



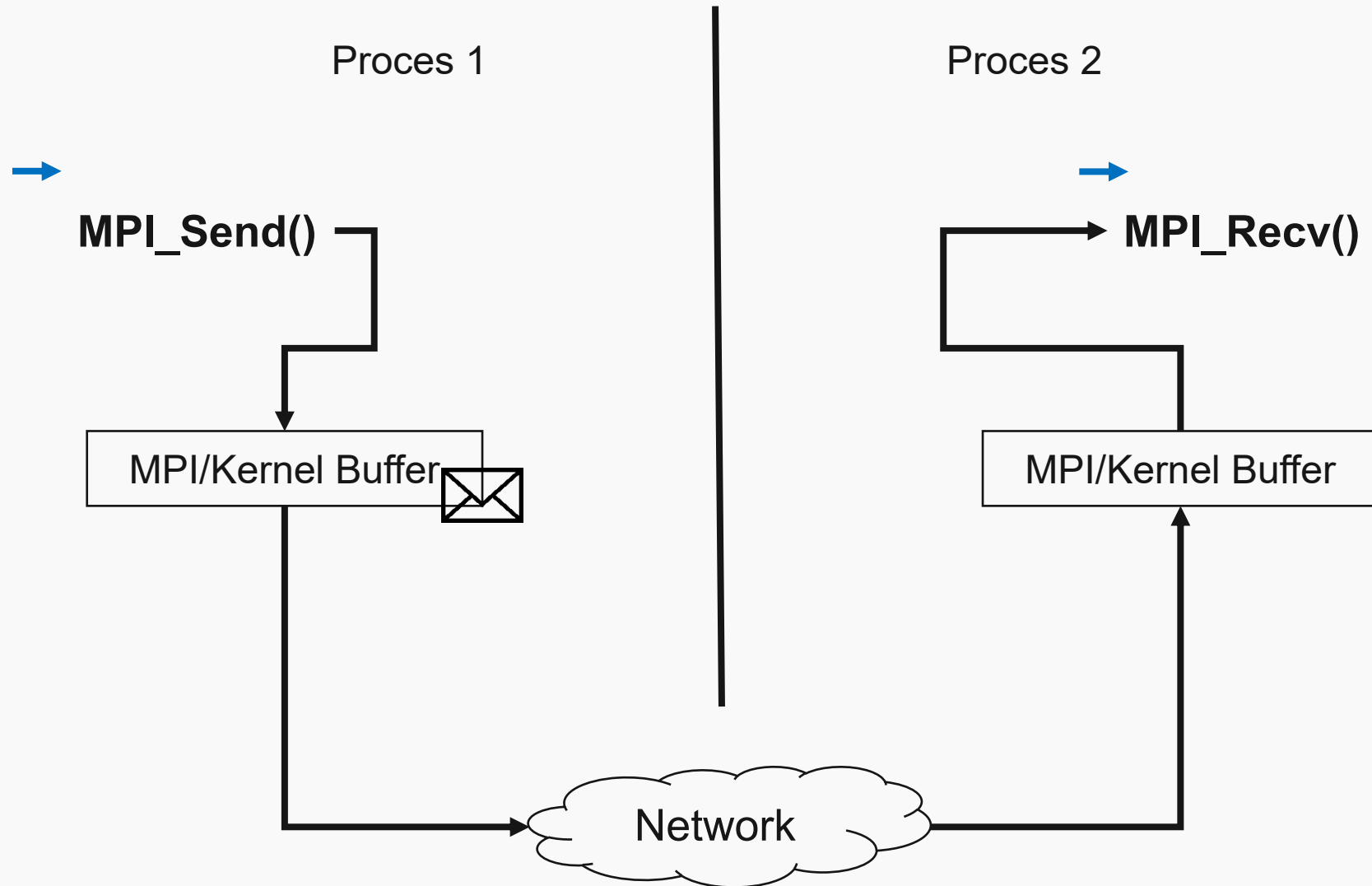


# MPI blocking recv/non-blocking send



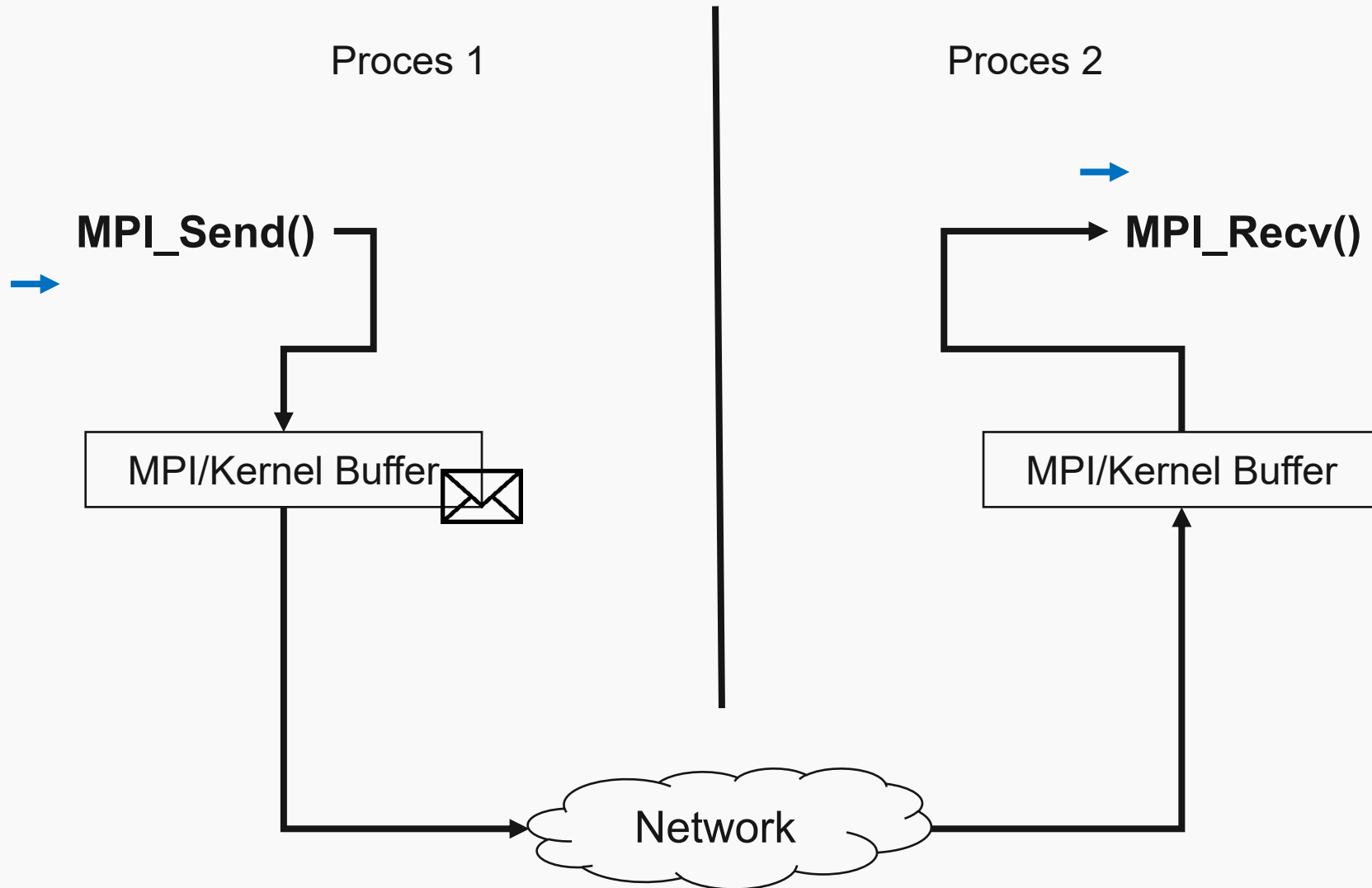


# MPI blocking recv/non-blocking send



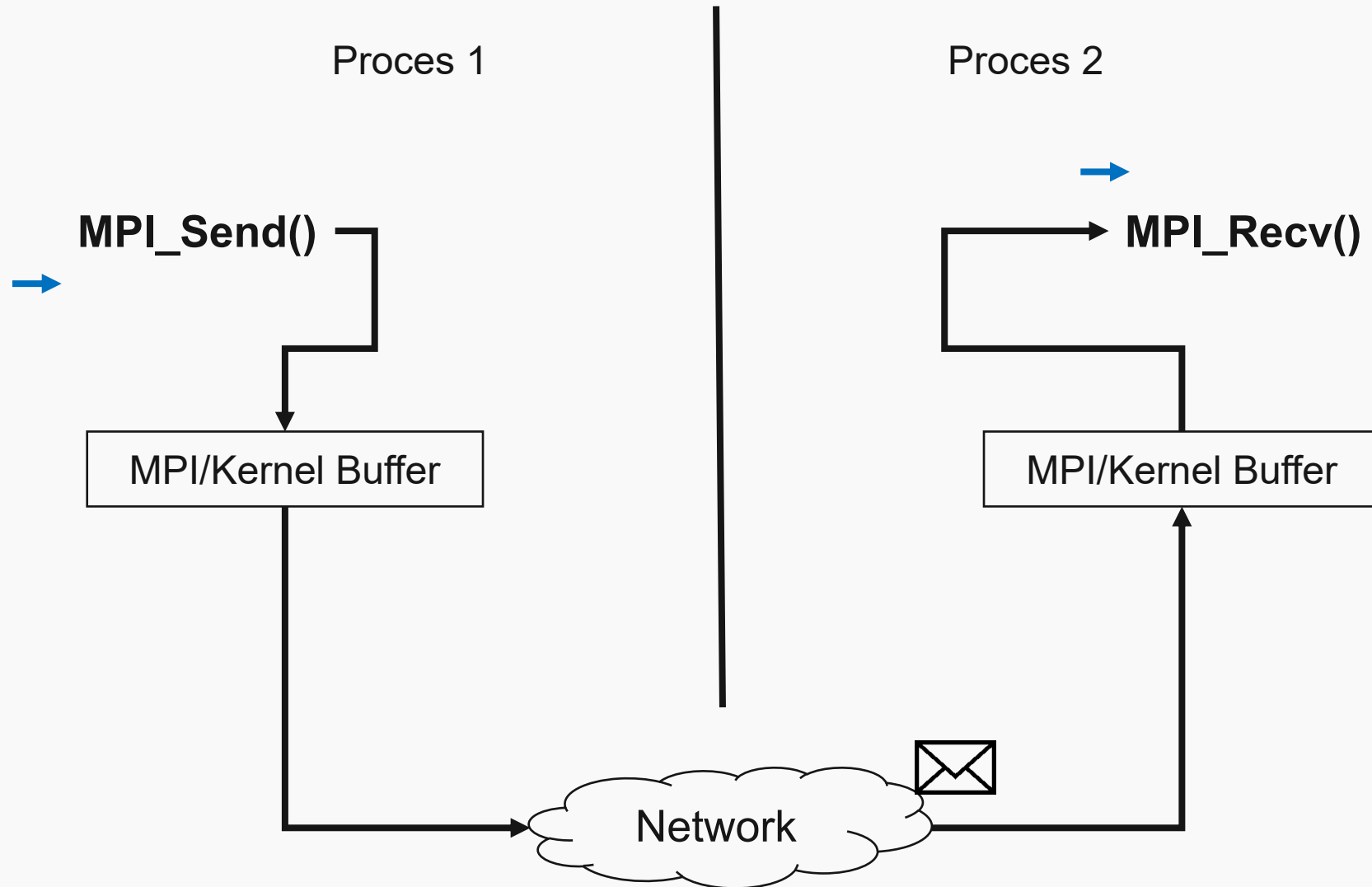


# MPI blocking recv/non-blocking send





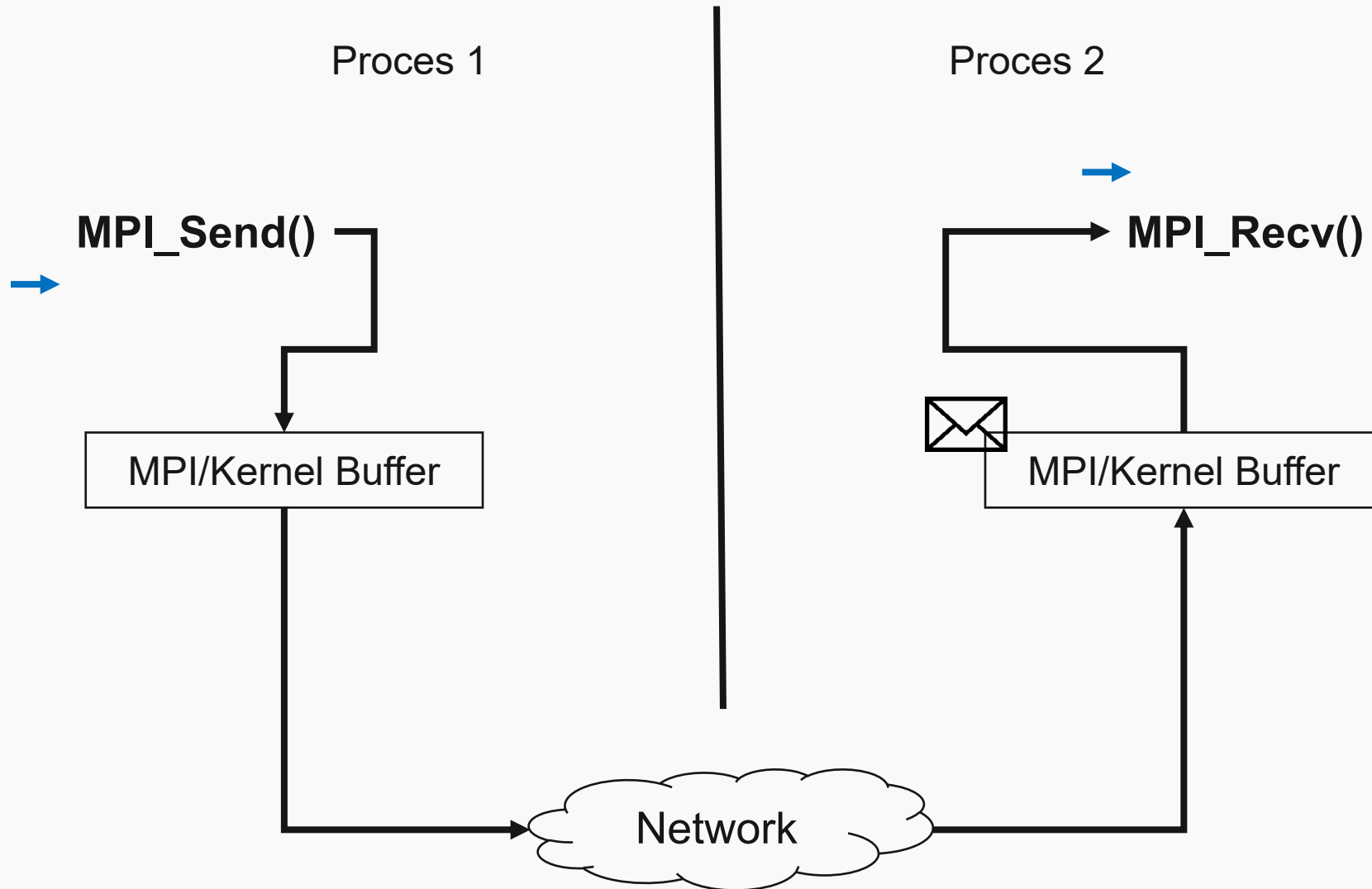
# MPI blocking recv/non-blocking send





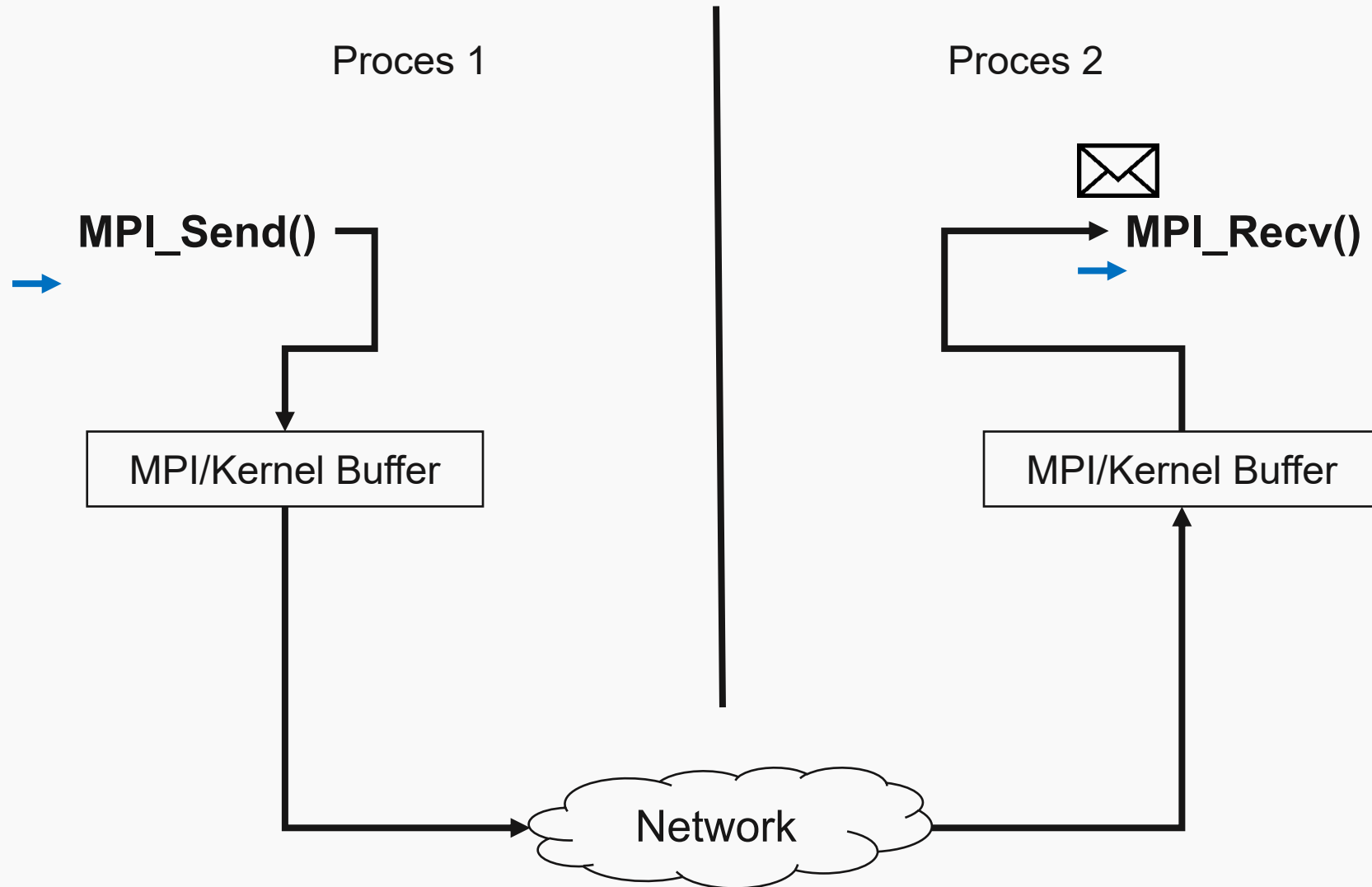


# MPI blocking recv/non-blocking send



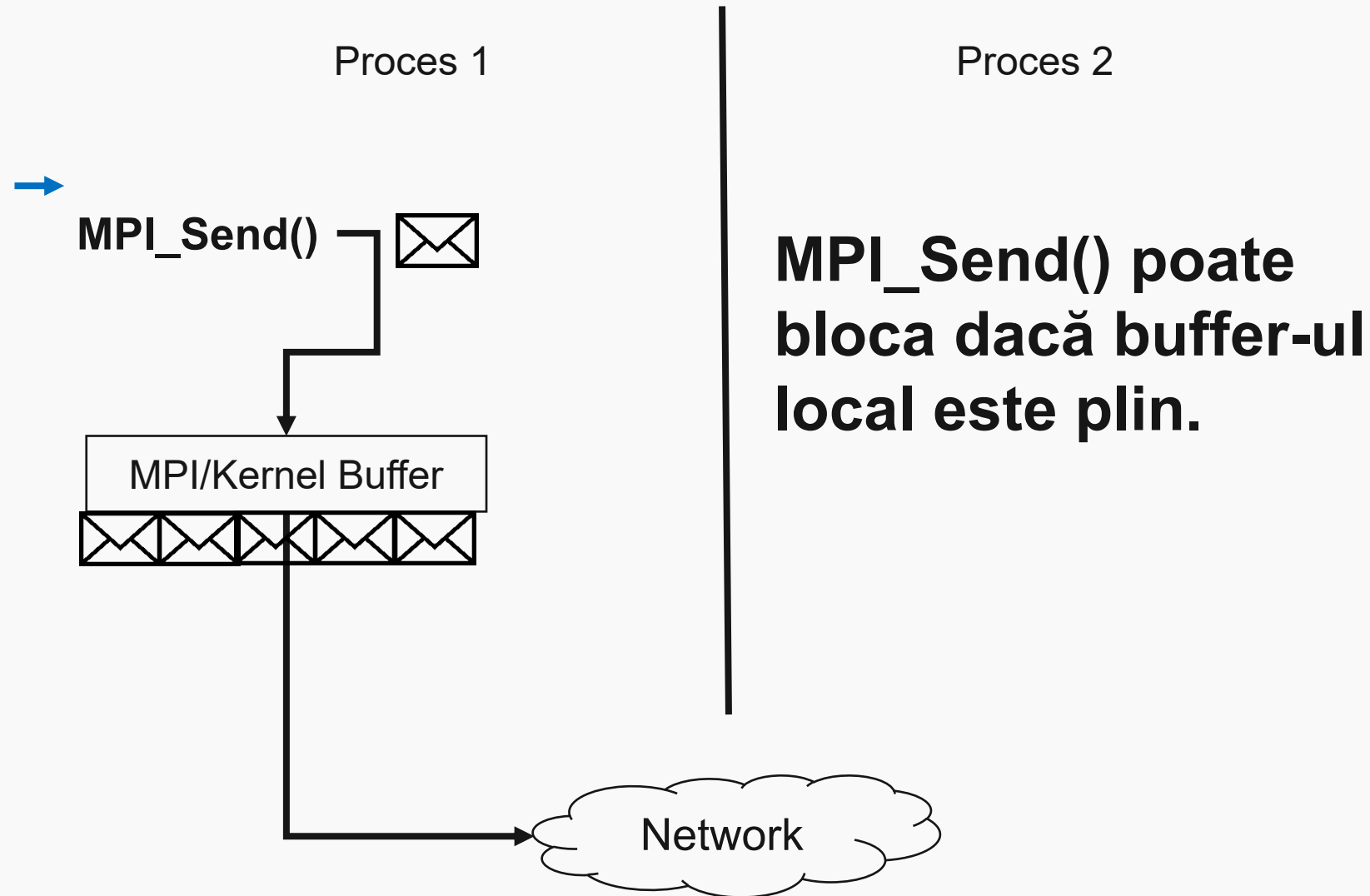


# MPI blocking recv/non-blocking send



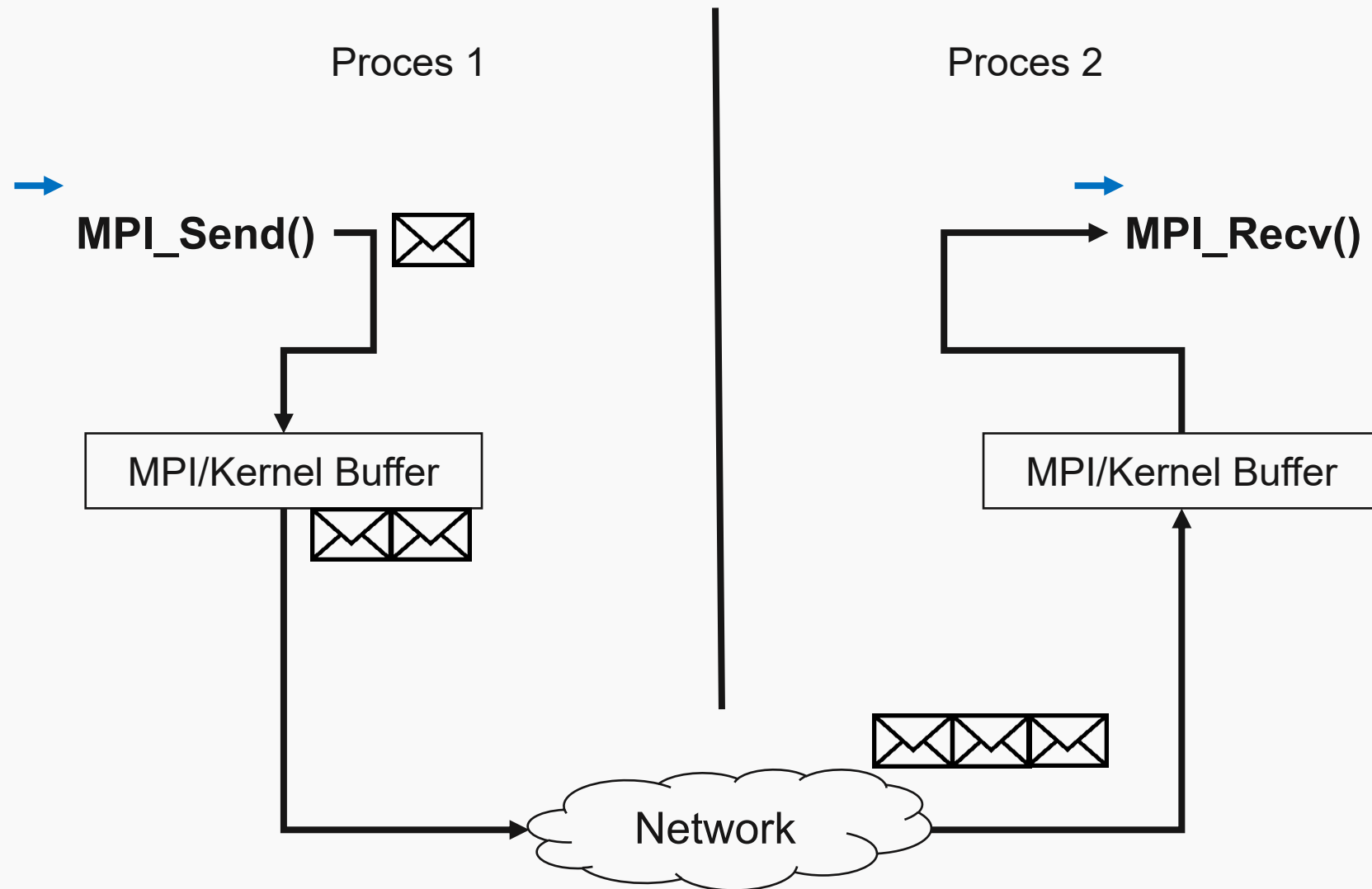


# MPI blocking recv/blocking send



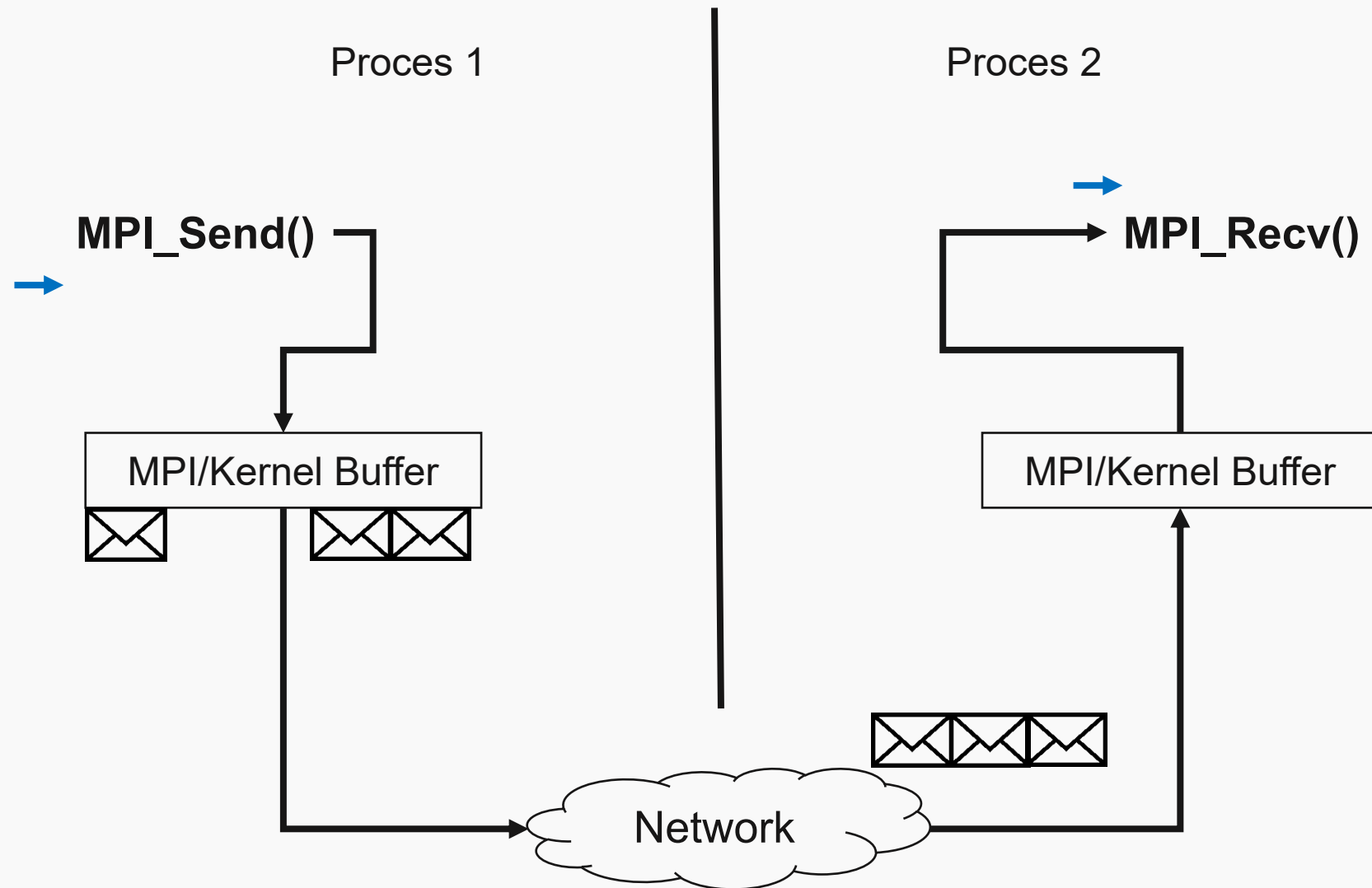


# MPI blocking recv/blocking send





# MPI blocking recv/blocking send

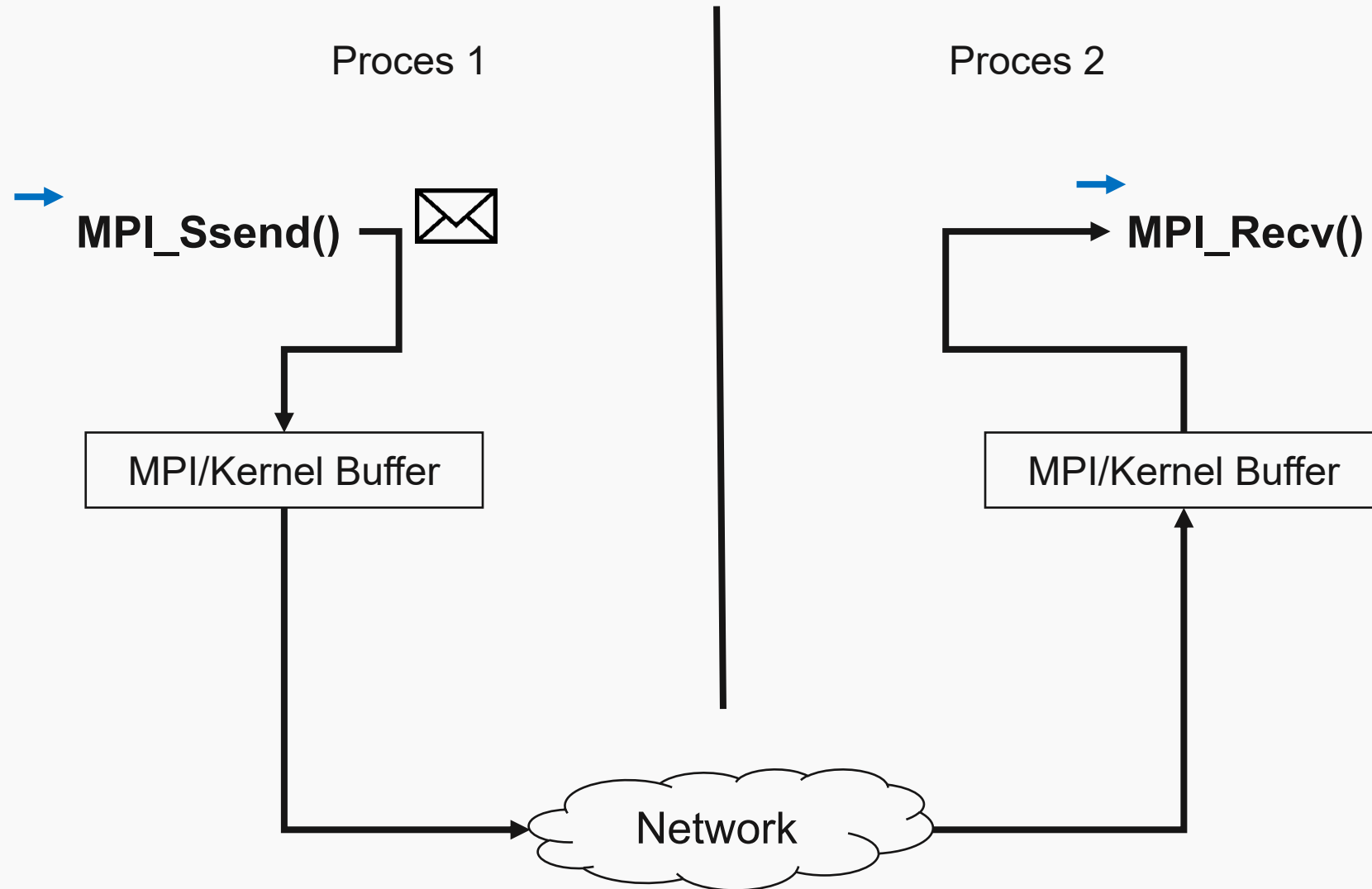




# MPI blocking recv/synchronized send



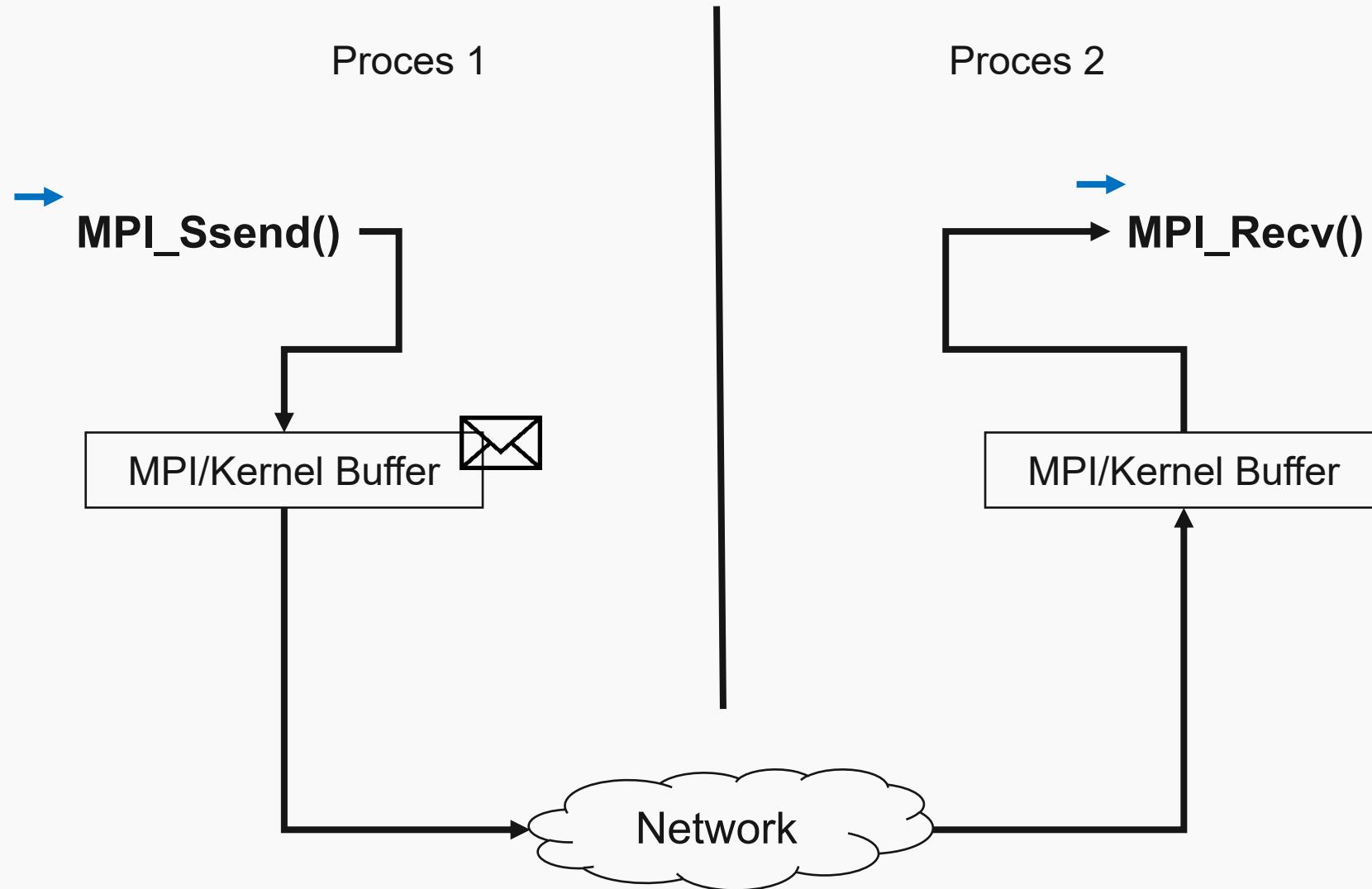
# MPI blocking recv/synchronized send





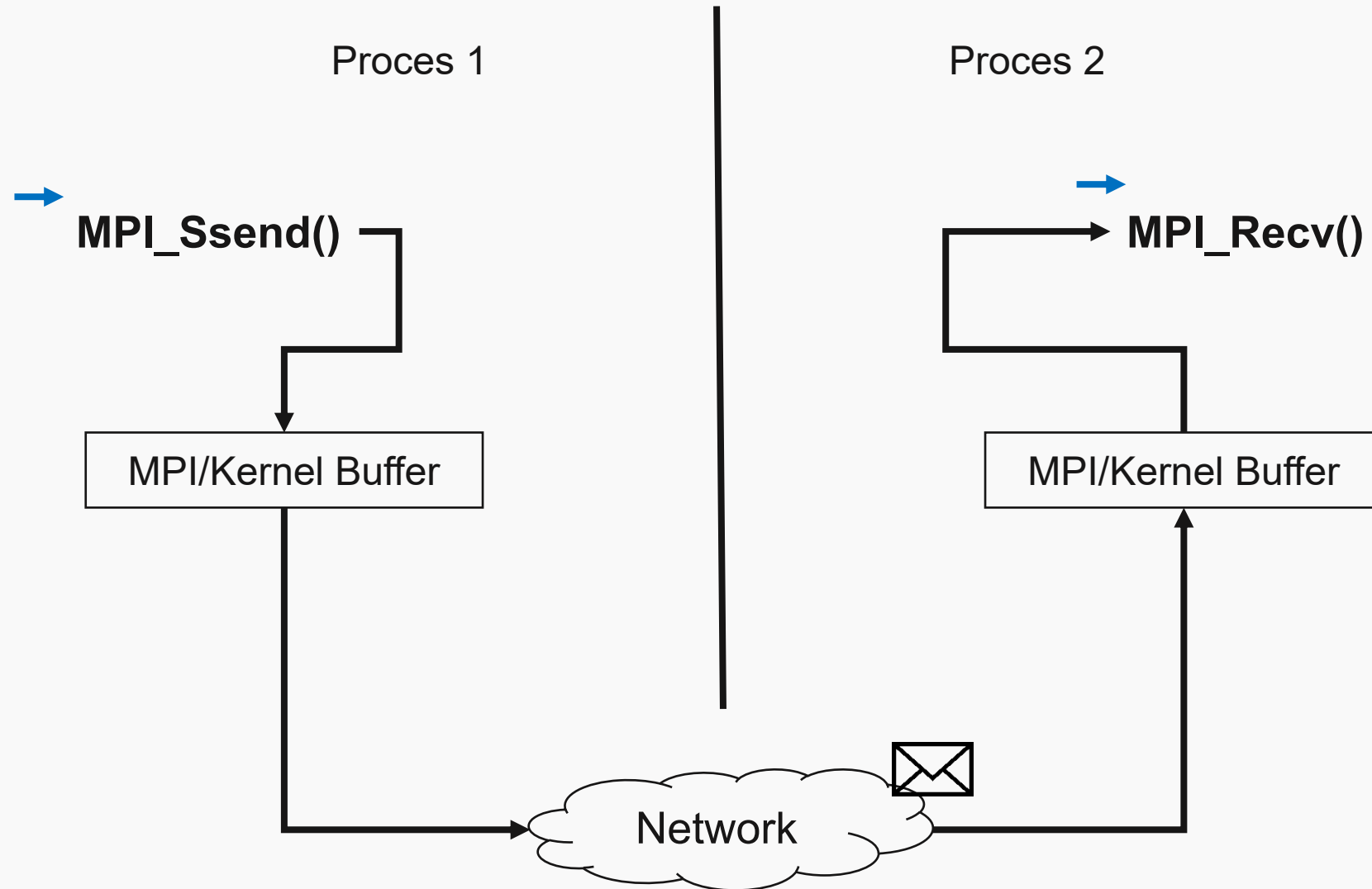


# MPI blocking recv/synchronized send



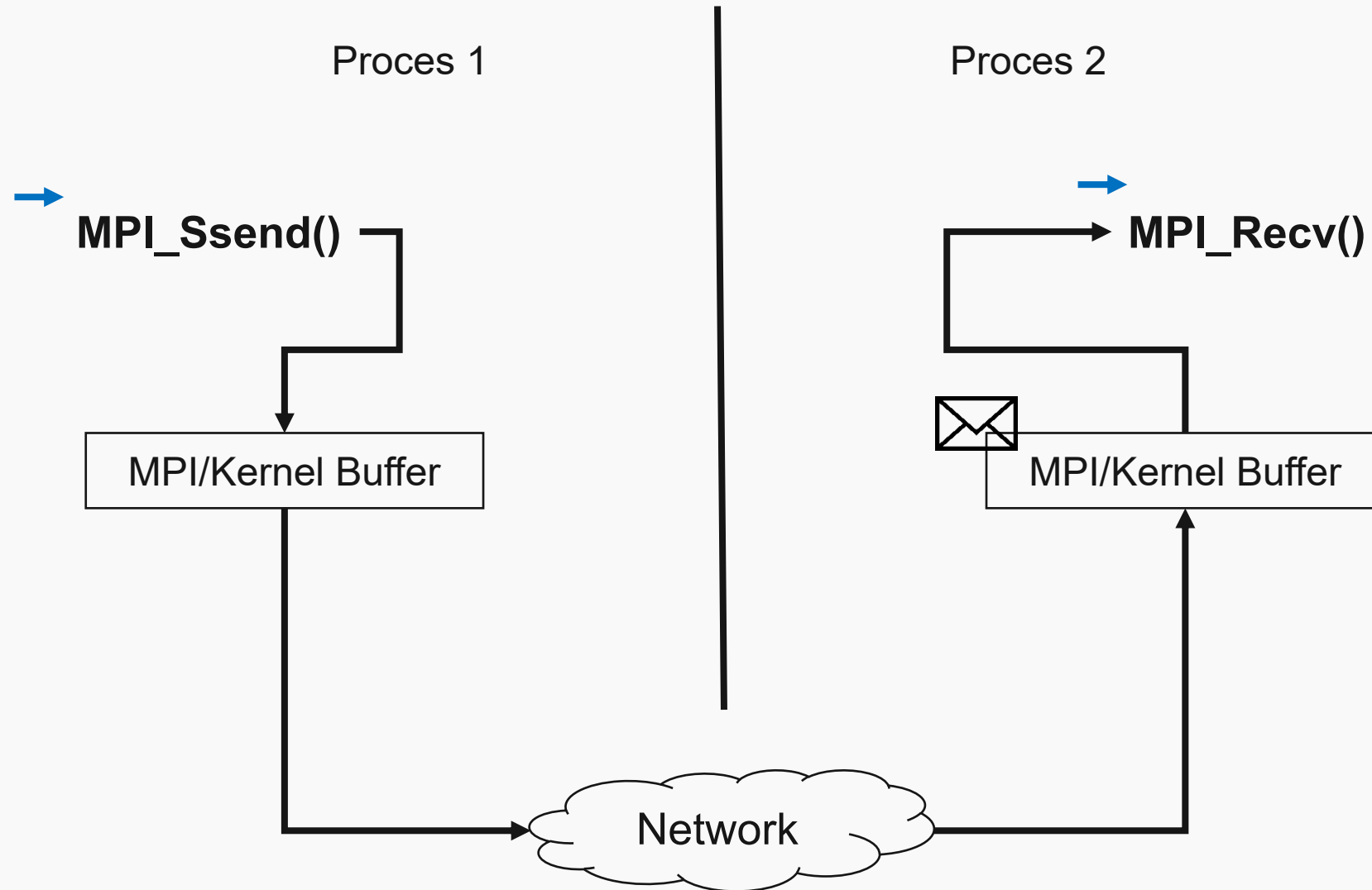


# MPI blocking recv/synchronized send



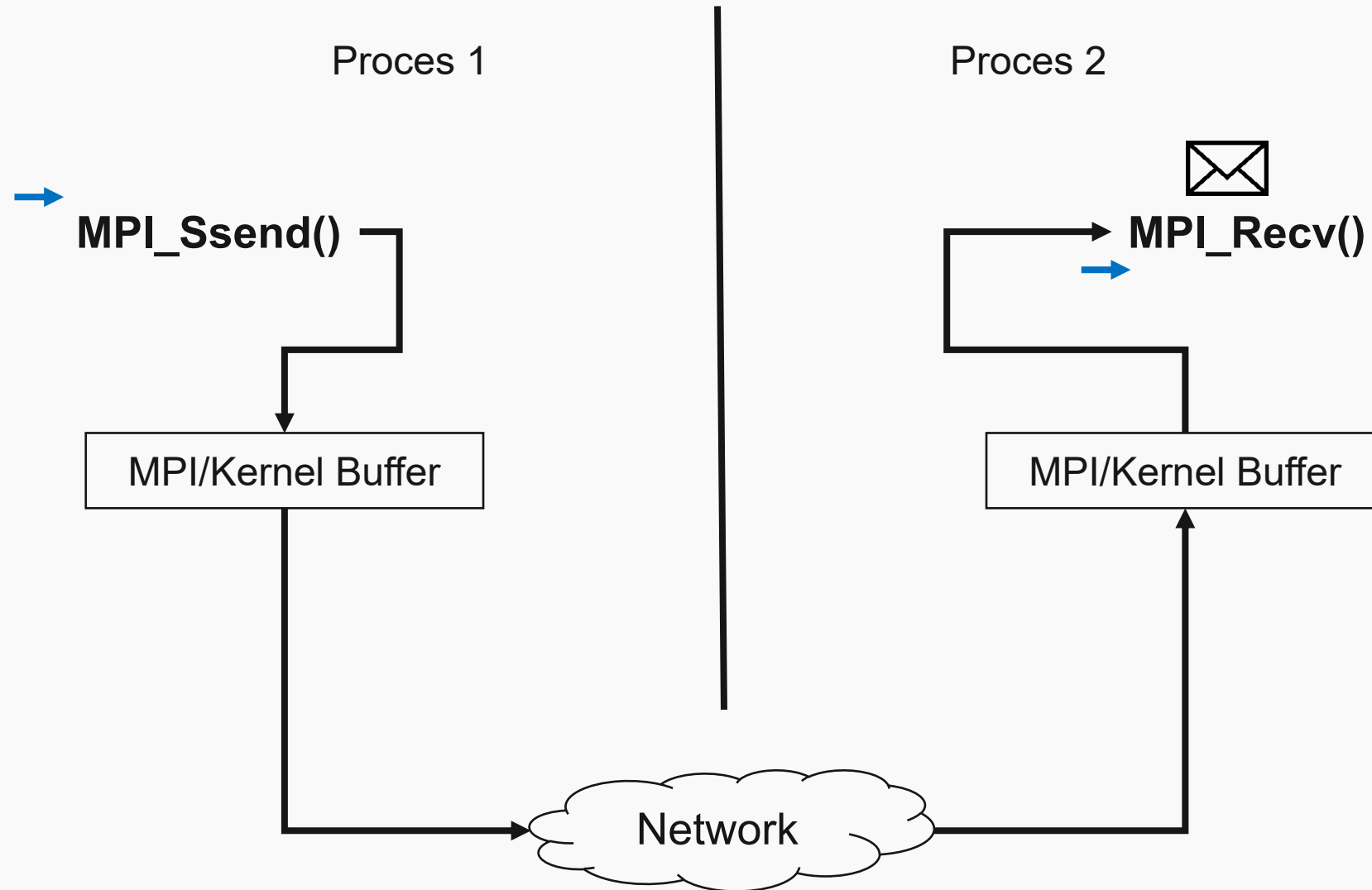


# MPI blocking recv/synchronized send



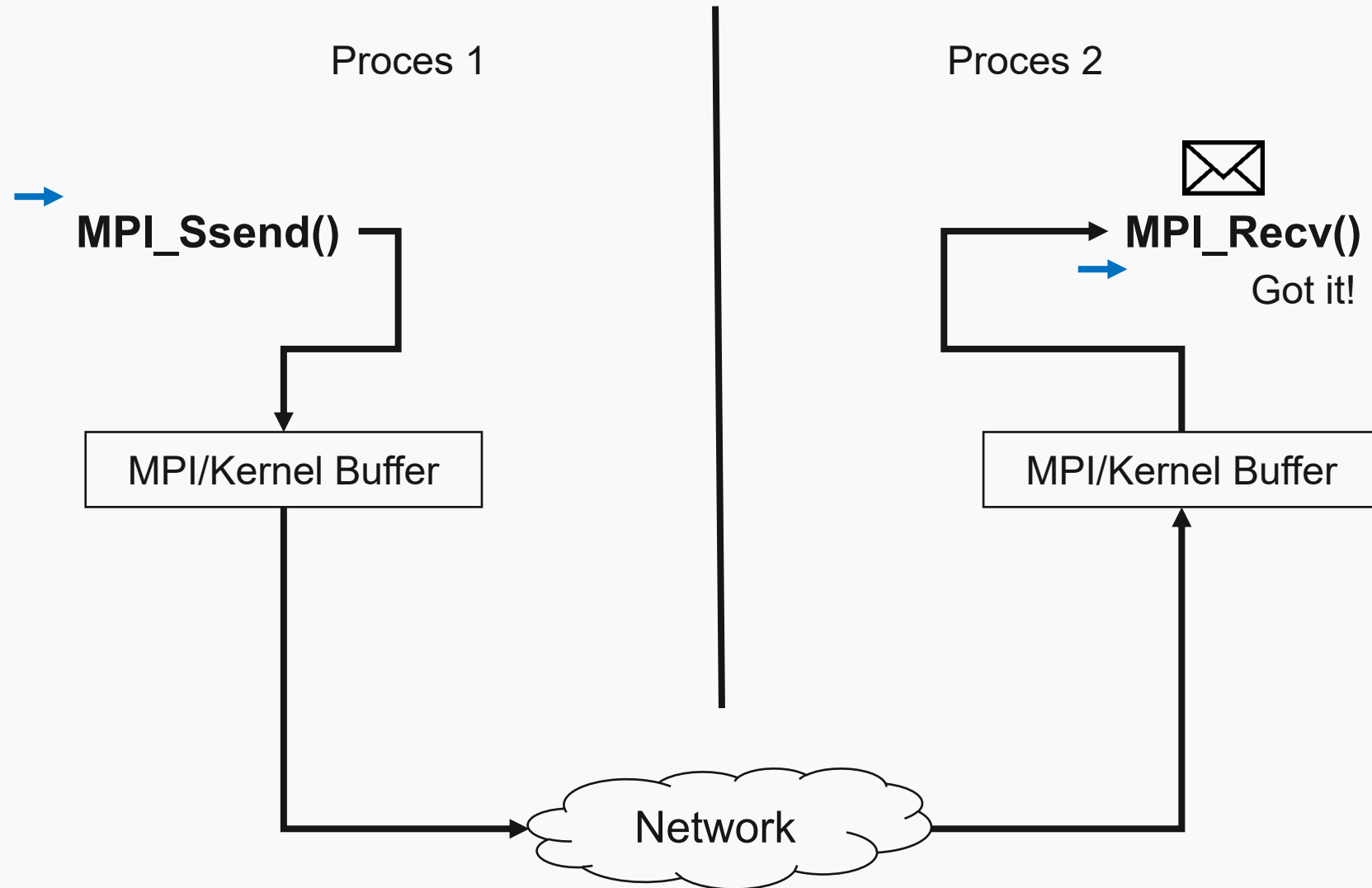


# MPI blocking recv/synchronized send



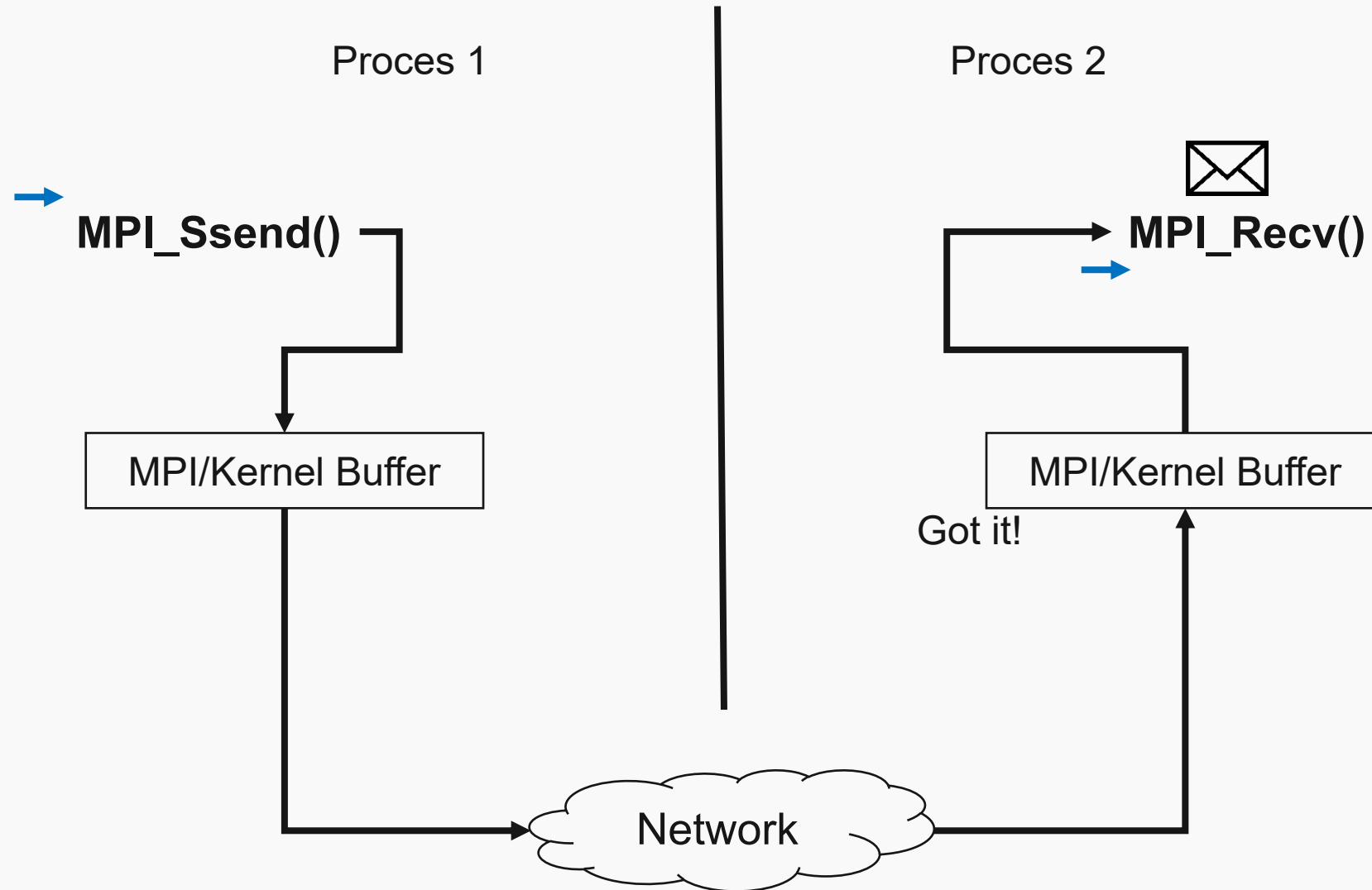


# MPI blocking recv/synchronized send



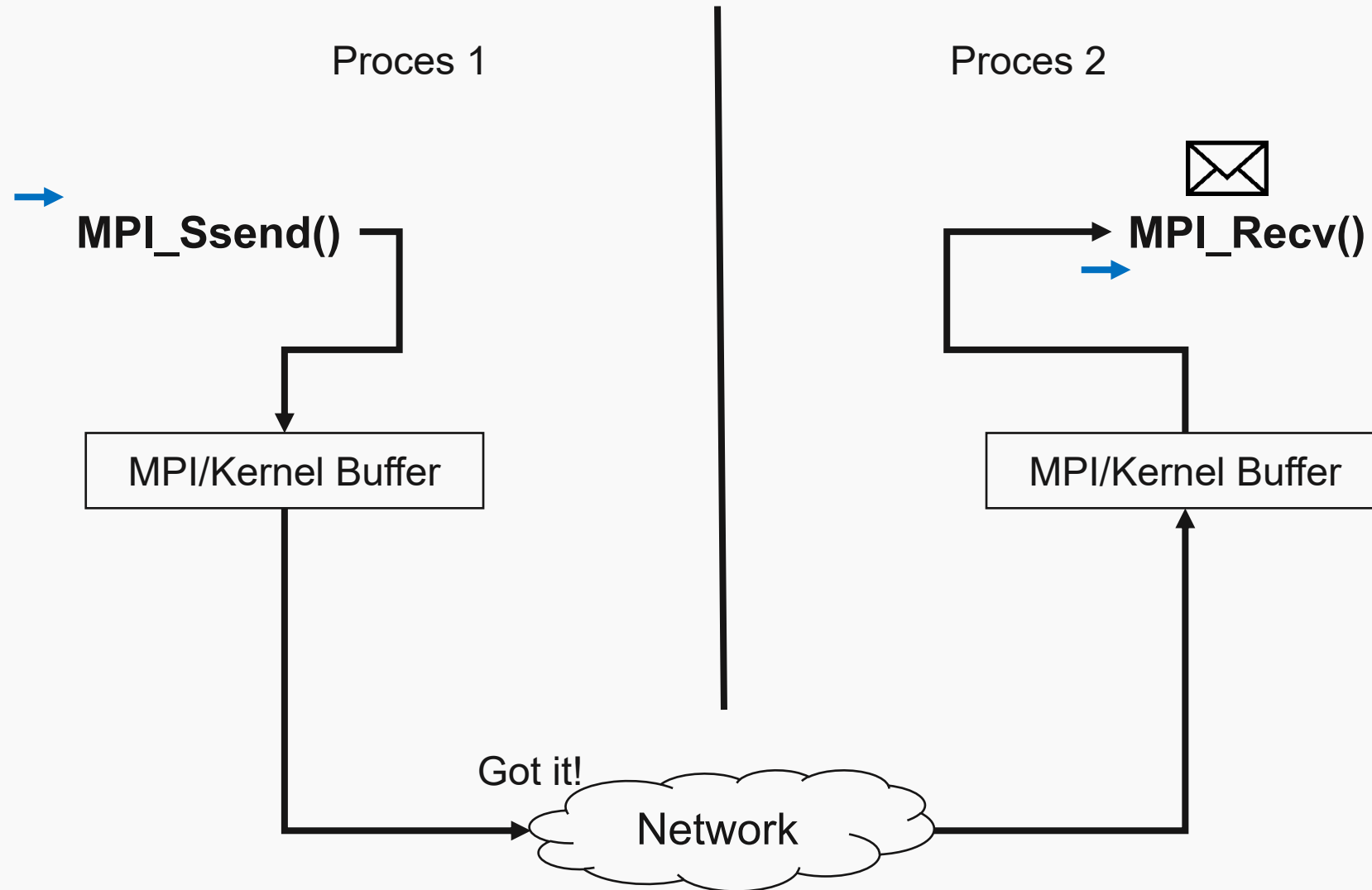


# MPI blocking recv/synchronized send



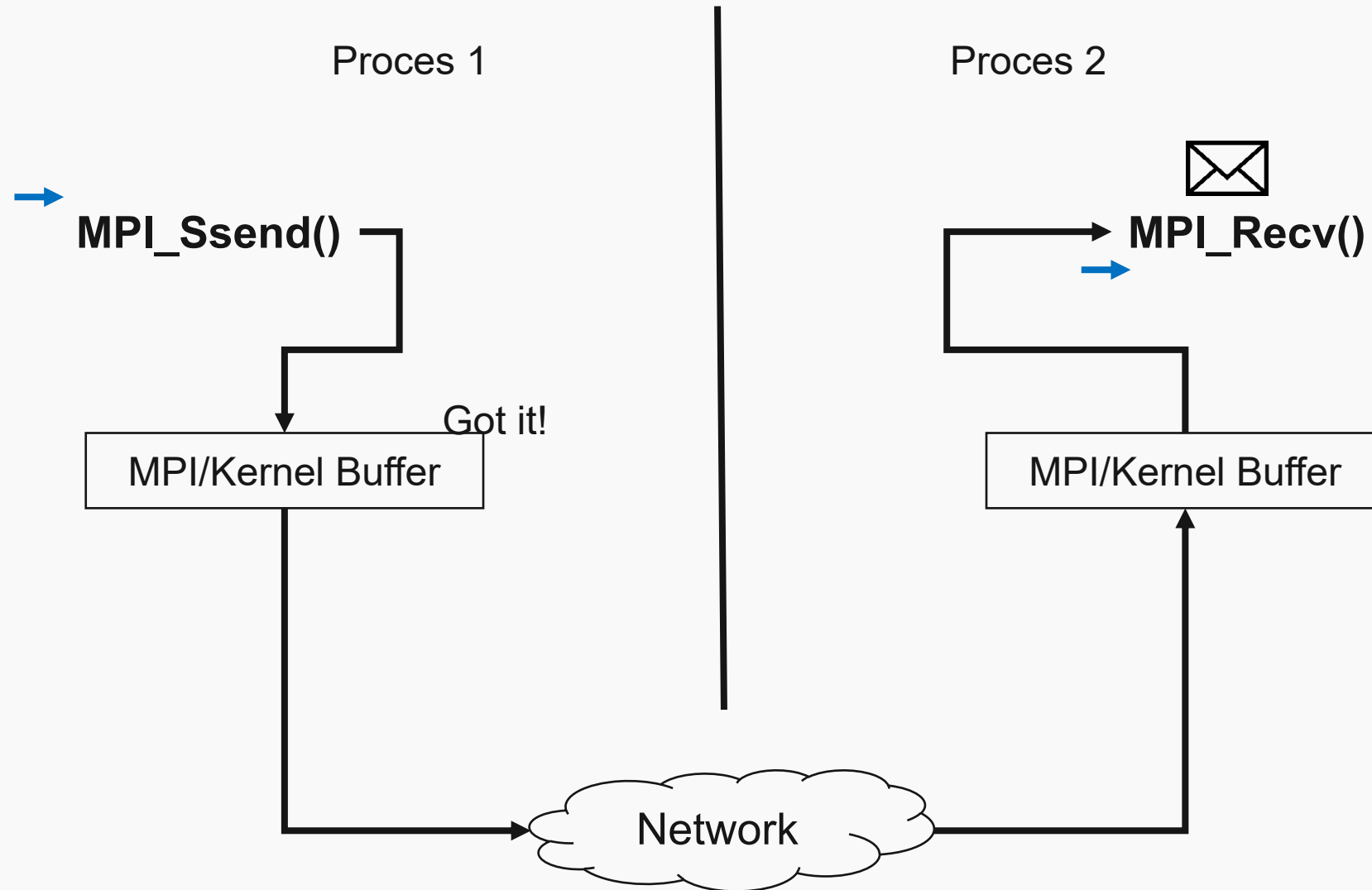


# MPI blocking recv/synchronized send





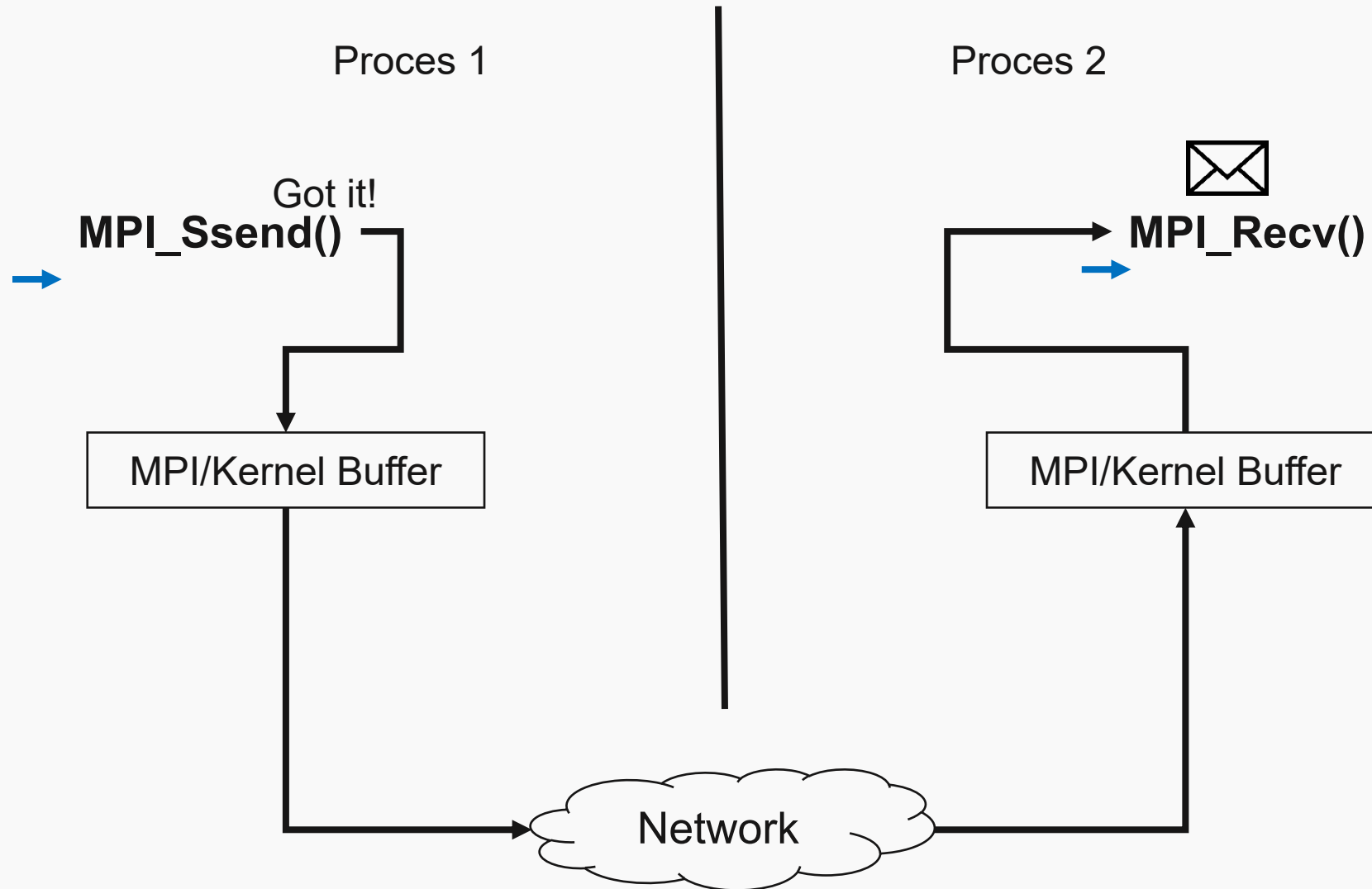
# MPI blocking recv/synchronized send







# MPI blocking recv/synchronized send

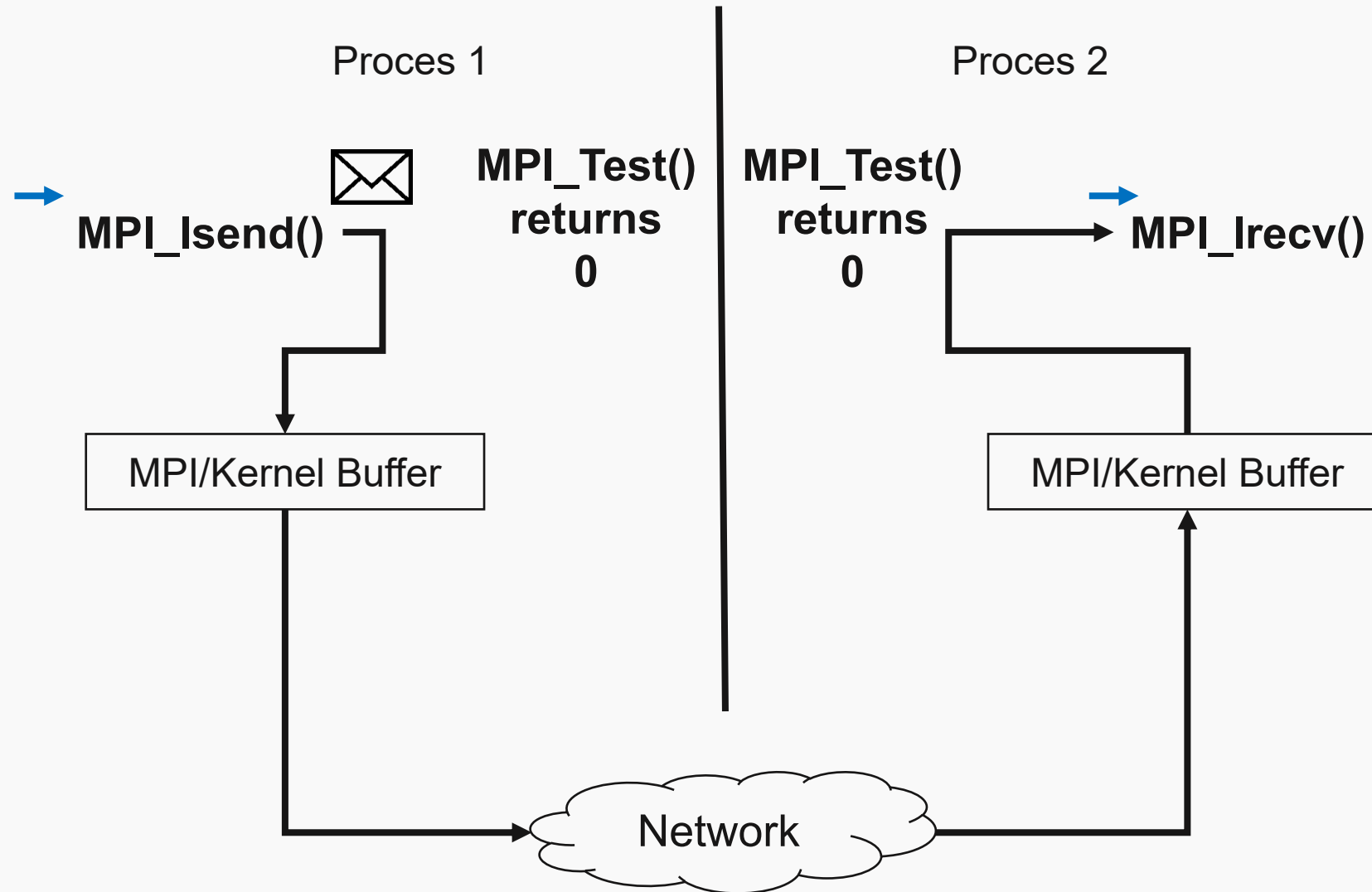




# MPI non-blocking recv/non-blocking send

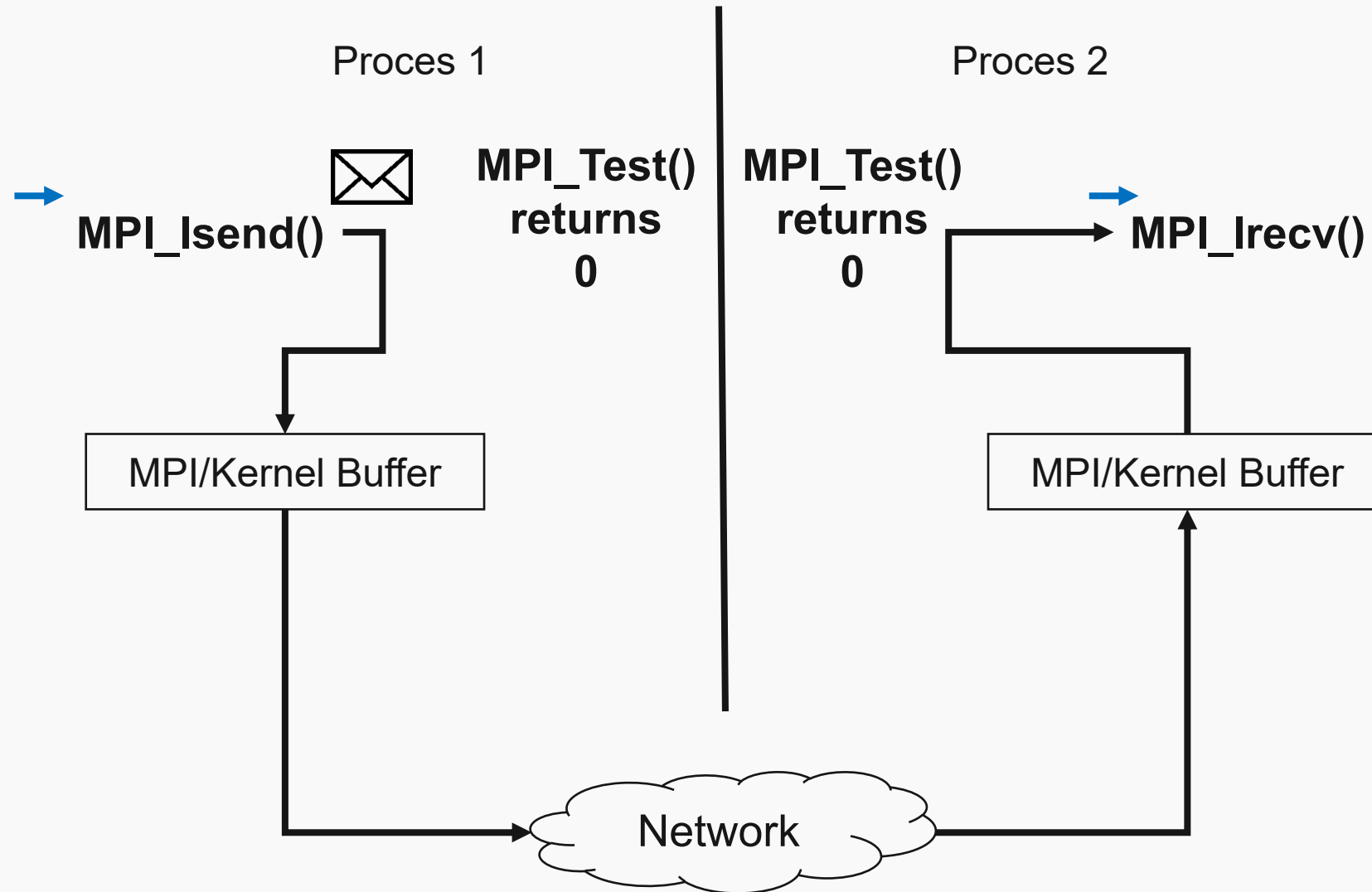


# MPI non-blocking recv/non-blocking send



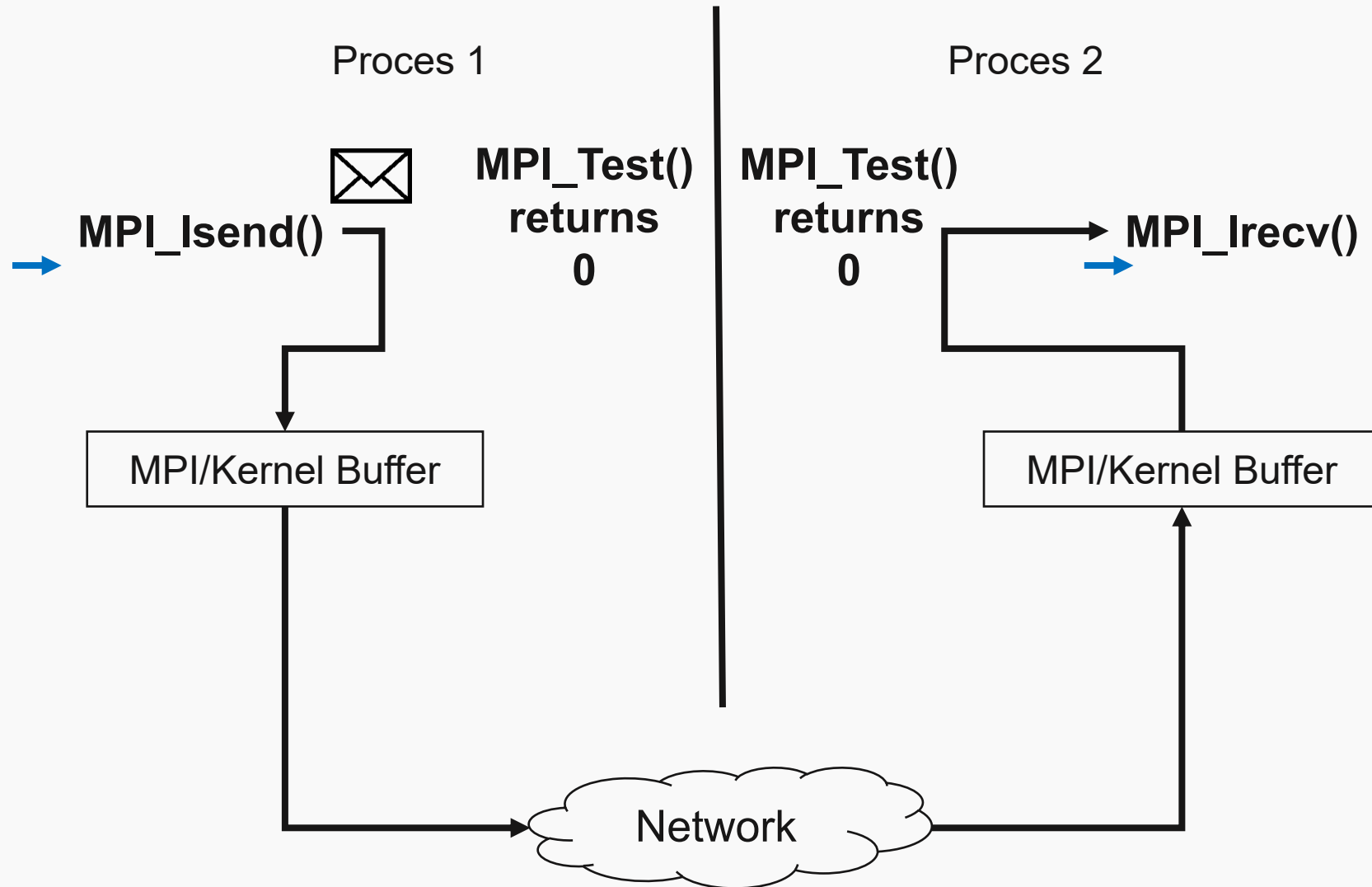


# MPI non-blocking recv/non-blocking send



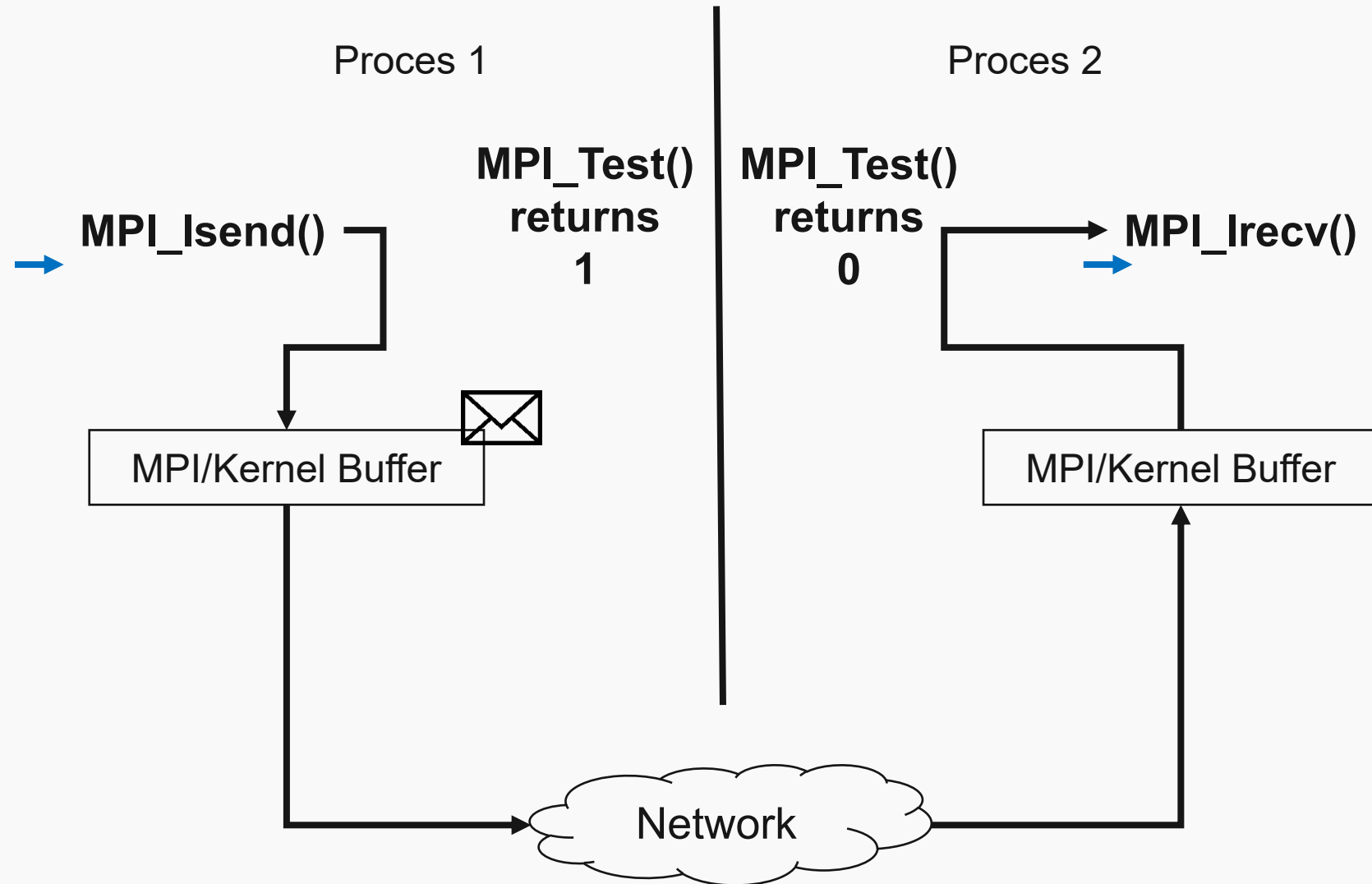


# MPI non-blocking recv/non-blocking send



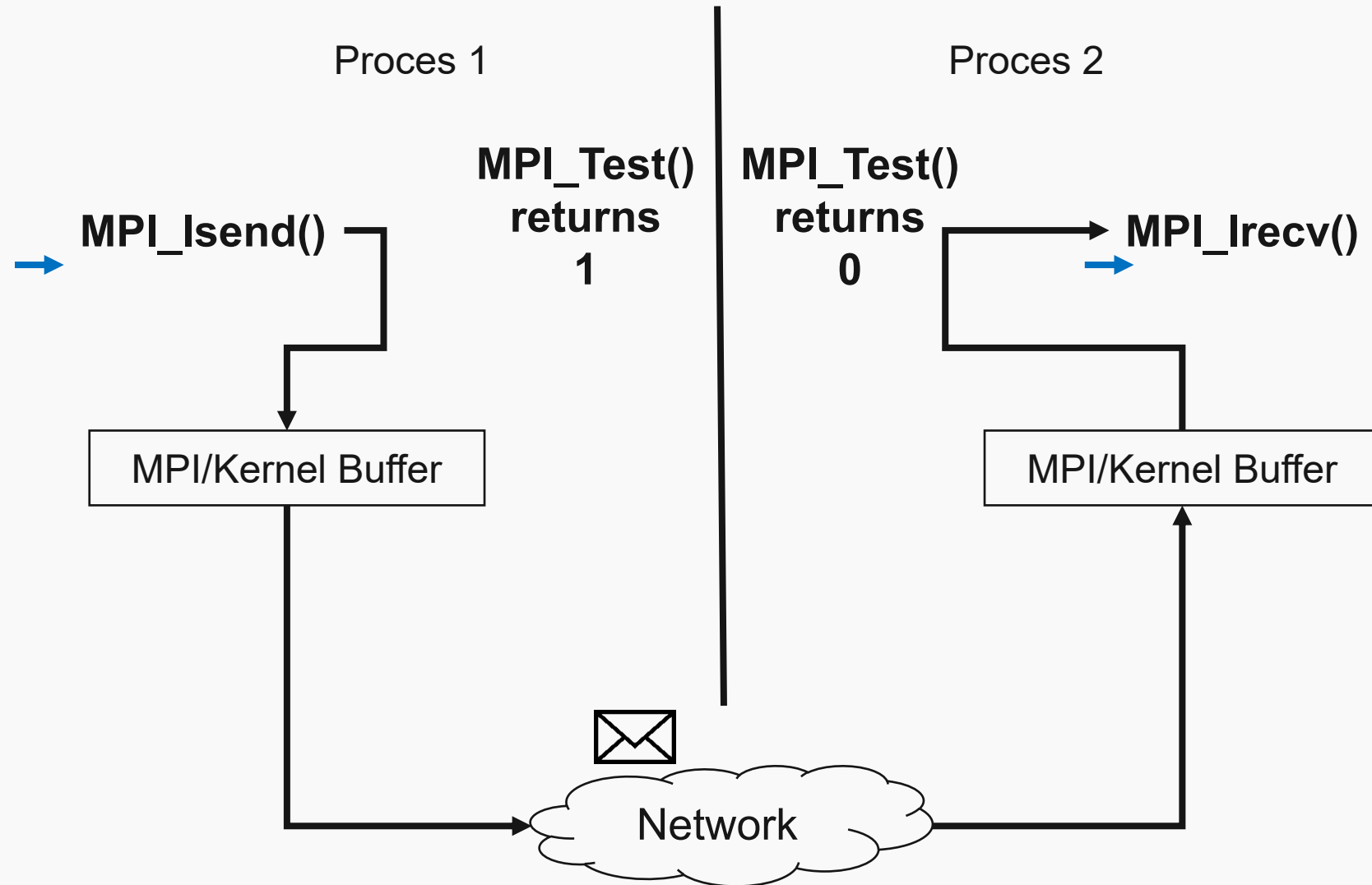


# MPI non-blocking recv/non-blocking send



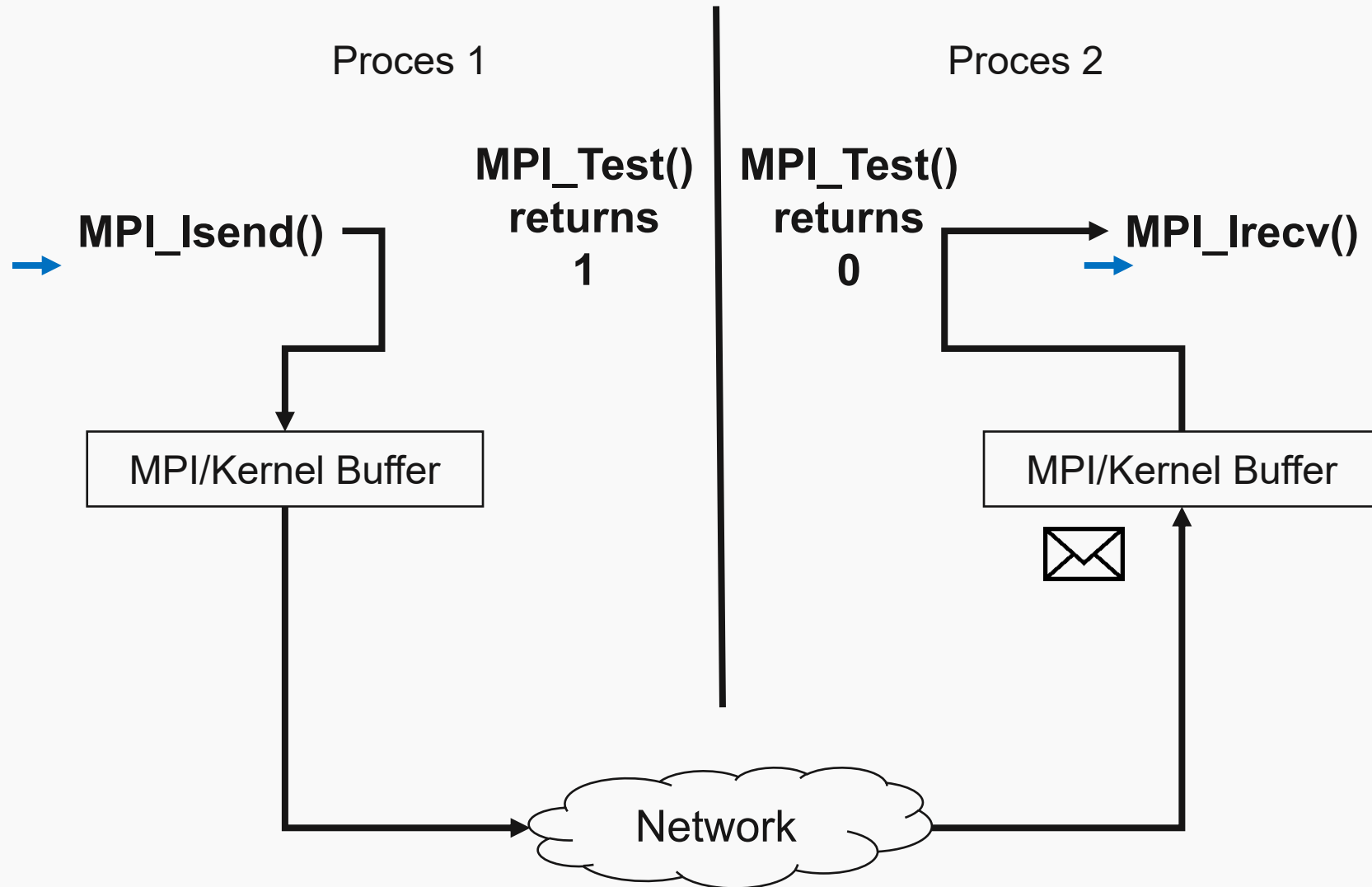


# MPI non-blocking recv/non-blocking send





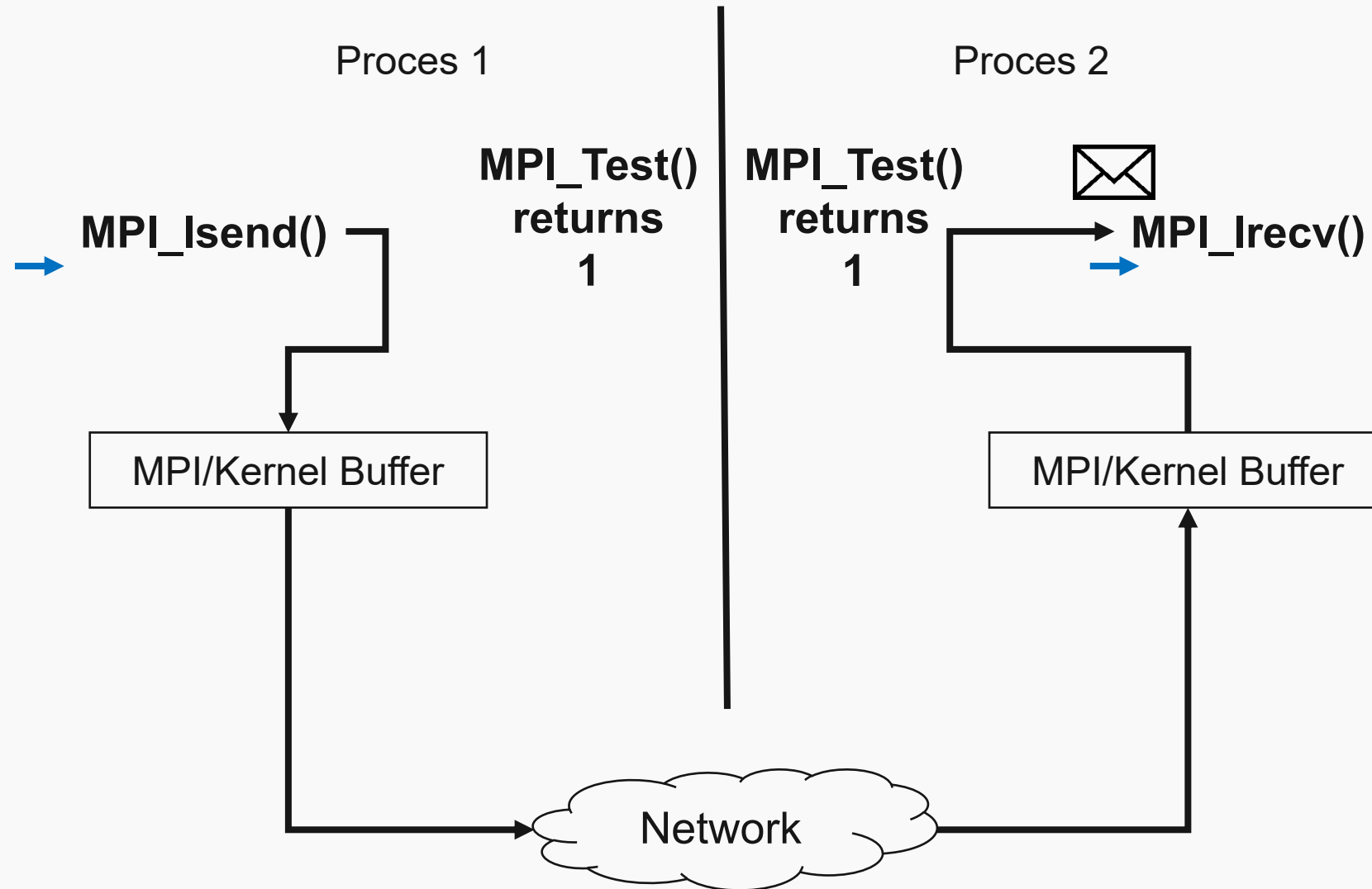
# MPI non-blocking recv/non-blocking send







# MPI non-blocking recv/non-blocking send





# Comunicația non-blocantă

**int MPI\_Isend( ↓ void \*b, ↓ int c, ↓ MPI\_Datatype d, ↓ int recv, ↓ int t, ↓ MPI\_Comm, ↑ MPI\_Request \*)**

**int MPI\_Recv( ↑ void \*b, ↓ int c, ↓ MPI\_Datatype d, ↓ int sender, ↓ int t, ↓ MPI\_Comm, ↑ MPI\_Request \*)**

**MPI\_Test(↓ MPI\_Request \*, ↑ int \* flag, ↑ MPI\_Status \*)**

**MPI\_Testall()**

**MPI\_Testany()**

**MPI\_Testsome()**

**MPI\_Wait(↓ MPI\_Request \*, ↑ MPI\_Status \*)**

**MPI\_Waitall()**

**MPI\_Waitany()**

**MPI\_Waitsome()**





# Calcul Complexitate



# Modelul Foster

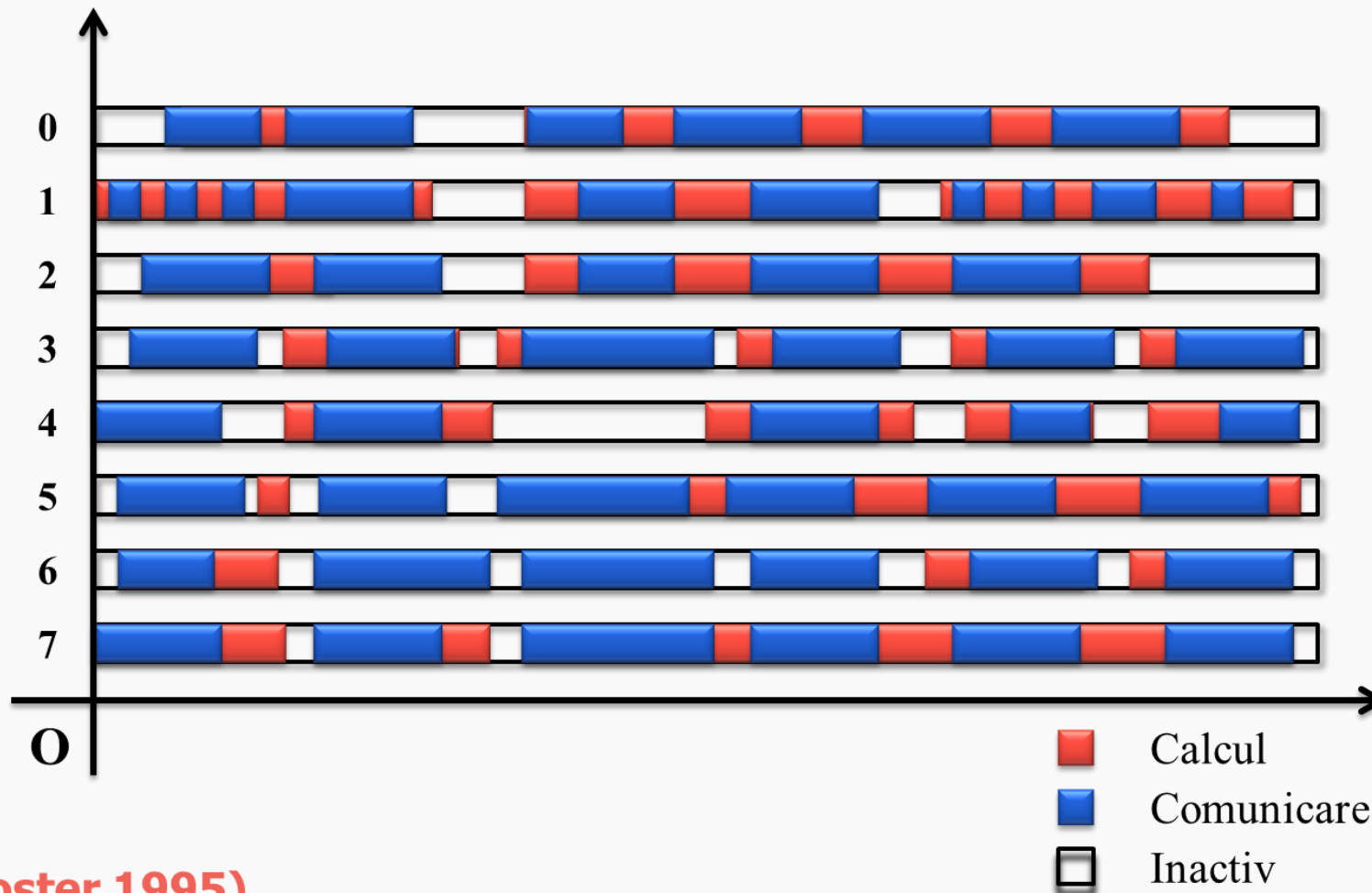
## Designing and Building Parallel Programs

**Ian Foster**





# Modelul Foster



(Foster 1995)



# Modelul Foster

## ■ Definiție

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

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

$$= T_{comp}^j + T_{jcommu_n} + T_{idle}$$

Unde j un proces arbitrar. SAU

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

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



# LogP model

## LogP: Towards a Realistic Model of Parallel Computation<sup>\*</sup>

David Culler, Richard Karp<sup>†</sup>, David Patterson,  
Abhijit Sahay, Klaus Erik Schauser, Eunice Santos,  
Ramesh Subramonian, and Thorsten von Eicken

*Computer Science Division,  
University of California, Berkeley*

### Abstract

A vast body of theoretical research has focused either on overly simplistic models of parallel computation, notably the PRAM, or overly specific models that have few representatives in the real world. Both kinds of models encourage exploitation of formal loopholes, rather than rewarding development of techniques that yield performance across a range of current and future parallel machines. This paper offers a new parallel machine model, called LogP, that reflects the critical technology trends underlying parallel computers. It is intended to serve as a basis for developing fast, portable parallel algorithms and to offer guidelines to machine designers. Such a model must strike a balance between detail and simplicity in order to reveal important bottlenecks without making analysis of interesting problems intractable. The model is based on four parameters that specify abstractly the computing bandwidth, the communication bandwidth, the communication delay, and the efficiency of coupling communication and computation. Portable parallel algorithms typically adapt to the machine configuration, in terms of these parameters. The utility of the model is demonstrated through examples that are implemented on the CM-5.



David Culler



David Patterson





# LogP model

- L – Limita superioară a **latenței (latency)** sau întârzierea de transmitere a unui mesaj de la sursă la destinație
- o - **overhead**, durata de timp în care procesorul execută transmiterea sau recepția fiecărui mesaj; În acest timp procesorul nu poate efectua alte operații
- g - **gap**, intervalul minim de timp între două transmițeri succesive sau două recepții succesive la același procesor. Reciproca lui g este echivalentă cu **lungimea de bandă (bandwidth)**
- P - numărul de module **procesor / memorie**. Presupunem că funcționează la aceeași unitate de timp, numită ciclu.





# Unele probleme nu pot fi paralelizate/distribuite

- Calculating the hash of a hash of a hash ...of a string.
- Deep First Search
- Huffman decoding
- Outer loops of most simulations
- P complete problems



# Paralelizare prin împărțirea problemei

Sunt o serie de probleme care sunt extrem de ușor de paralelizat/distribuit.

**Embarrassingly parallel**



# Embarrassingly parallel problems

Multiplicare unui vector cu un scalar

9	6	9	4	2	7	6	5	6	1
---	---	---	---	---	---	---	---	---	---

\* 3

27	18	27	12	6	21	18	15	18	3
----	----	----	----	---	----	----	----	----	---



# Embarrassingly parallel problems

Toate calculele pot fi efectuate în același timp

9	6	9	4	2	7	6	5	6	1
---	---	---	---	---	---	---	---	---	---

\* 3

27	18	27	12	6	21	18	15	18	3
----	----	----	----	---	----	----	----	----	---



# Embarrassingly parallel problems

Câte elemente sunt?





# Embarrassingly parallel problems

Câte elemente sunt?







# Embarrassingly parallel problems

Câte elemente sunt? **N**





# Embarrassingly parallel problems

Dar câte elemente de procesare?





# Embarrassingly parallel problems

Dar câte elemente de procesare? **P**





# Embarrassingly parallel problems

Dar câte procese?





# Embarrassingly parallel problems

Dar câte procese? **P**





# Embarrassingly parallel problems

Cum este P față de N?





# Embarrassingly parallel problems

$P \ll N$





# Embarrassingly parallel problems

**Caz concret:  $P = 2$**   
**Cum împărțim?**







# Embarrassingly parallel problems

**Caz concret:  $P = 2$**

**Cum împărțim?**



Proces 1

Proces 2



# Embarrassingly parallel problems

**Caz concret:  $P = 2$**   
**Cum împărțim?**



Proces 1

Proces 2



# Embarrassingly parallel problems

**Caz concret:  $P = 2$**   
**Cum împărțim?**



Proces 1

Proces 2



# Embarrassingly parallel problems

**Caz concret:  $P = 2$**

**Cum împărțim? Putem și random**



Proces 1

Proces 2



# Embarrassingly parallel problems

**Caz concret:  $P = 2$**   
**Cum împărțim?**

9	6	9	4	2	7	6	5	6	...	1
---	---	---	---	---	---	---	---	---	-----	---

Proces 1

Proces 2



# Embarrassingly parallel problems

**Caz concret:  $P = 2$**   
**Cum împărțim?**

9	6	9	4	2	7	6	5	6
---	---	---	---	---	---	---	---	---

 ... 

1
---

Proces 1

Proces 2

**Este utilă?**



# Embarrassingly parallel problems

**Caz concret:  $P = 2$**   
**Cum împărțim?**

9	6	9	4	2	7	6	5	6
---	---	---	---	---	---	---	---	---

 ... 

1
---

Proces 1

Proces 2

**Ce ne dorim?**



# Embarrassingly parallel problems

**Caz concret:  $P = 2$**   
**Cum împărțim?**

9	6	9	4	2	7	6	5	6
---	---	---	---	---	---	---	---	---

 ... 

1
---

Proces 1

Proces 2

**Ce ne dorim? Aproximativ același număr elemente**





# Embarrassingly parallel problems

**Aproximativ  $N/P$  elemente pe fiecare proces**

9	6	9	4	2	7	6	5	6	...	1
---	---	---	---	---	---	---	---	---	-----	---

Proces 1

Proces 2



# Embarrassingly parallel problems

**Aproximativ  $N/P$  elemente pe fiecare proces**

9 6 9 4 2 7 6 5 6 ... 1

**Dacă  $N$  nu se divide perfect la  $P$ ?**

Proces 1

Proces 2



# Embarrassingly parallel problems

**Aproximativ** N/P elemente

**Dacă N nu se divide perfect la P?**

1

6		
4	2	7
9	6	9

Proces 1

8		
4	9	2
5	6	3

Proces 2



# Embarrassingly parallel problems

**$\text{floor}(N/P)$  elemente       $\text{floor}(15/2) = 7$**

1

6

4 2 7

9 6 9

Proces 1

8

4 9 2

5 6 3

Proces 2



# Embarrassingly parallel problems

**$\text{ceil}(N/P)$  elements       $\text{ceil}(15/2) = 8$**

6	5	
4	2	7
9	6	9

Proces 1

8	1	
4	9	2
6	3	

Proces 2



# Embarrassingly parallel problems

$$A = \text{floor}(N/P)$$





# Embarrassingly parallel problems

$$A = \text{ceil}(N/P)$$





# Embarrassingly parallel problems

Formule elegante:

**rank** este identificator de proces, are valori de la 0 la **P**

**start** = **rank** \*  $\text{ceil}(N/P)$

**end** =  $\min(N, (\text{rank} + 1) * \text{ceil}(N/P))$







# Embarrassingly parallel problems

Formule elegante:

**rank** este identificator de proces, are valori de la 0 la **P**

**start** = **rank** \*  $\text{ceil}(N / P)$

**end** =  $\min(N, (\text{rank} + 1) * \text{ceil}(N / P))$



Funcționează și:

**start** =  $\text{round}(\text{rank} * N / P)$

**end** =  $\text{round}((\text{rank}+1) * N / P)$  De ce?





# Măsurarea timpului de execuție

`time mpirun -n NUM_PROC ./executabil p a r a m e t r i`



# Măsurare timp – Linia de comandă

***time sleep 5***

real 0m5.001s  
user 0m0.000s  
sys 0m0.001s

***time sleep 5***

sleep 5 0.00s user 0.00s system 0% cpu 5.002 total

***/usr/bin/time sleep 5***

0.00user 0.00system 0:05.00elapsed 0%CPU (0avgtext+0avgdata 2076maxresident)k  
0inputs+0outputs (0major+73minor)pagefaults 0swaps



# Măsurare timp – Linia de comandă

***time sleep 5***

real 0m5.001s  
user 0m0.000s  
sys 0m0.001s

Wall clock time – Timpul trecut de la pornirea programului – Pe acesta îl folosim

***time sleep 5***

sleep 5 0.00s user 0.00s system 0% cpu 5.002 total

***/usr/bin/time sleep 5***

0.00user 0.00system 0:05.00elapsed 0%CPU (0avgtext+0avgdata 2076maxresident)k  
0inputs+0outputs (0major+73minor)pagefaults 0swaps



# Măsurare timp – Linia de comandă

```
time sleep 2
```

```
real 0m2.021s  
user 0m0.000s  
sys 0m0.000s
```

```
time sleep 2
```

```
real 0m2.018s  
user 0m0.000s  
sys 0m0.016s
```

Timpii măsurați nu sunt exacti.  
Pentru a măsura corect trebuie  
să facem medie a timpilor după  
mai multe rulări sau să  
considerăm doar timpi mari –  
peste o secundă.

```
time sleep 2
```

```
real 0m2.016s  
user 0m0.000s  
sys 0m0.000s
```

```
time sleep 2
```

```
real 0m2.015s  
user 0m0.000s  
sys 0m0.000s
```



# Măsurare timp – Linia de comandă

***time sleep 5***

real 0m5.001s  
user 0m0.000s  
sys 0m0.001s

**Suma timpului petrecut  
în user space pe fiecare  
core.**

***time sleep 5***

sleep 5 0.00s user 0.00s system 0% cpu 5.002 total

***/usr/bin/time sleep 5***

0.00user 0.00system 0:05.00elapsed 0%CPU (0avgtext+0avgdata 2076maxresident)k  
0inputs+0outputs (0major+73minor)pagefaults 0swaps



# Măsurare timp – Linia de comandă

***time sleep 5***

real 0m5.001s  
user 0m0.000s  
sys 0m0.001s

**Suma timpului petrecut  
în kernel pe fiecare core.**

***time sleep 5***

sleep 5 0.00s user 0.00s system 0% cpu 5.002 total

***/usr/bin/time sleep 5***

0.00user 0.00system 0:05.00elapsed 0%CPU (0avgtext+0avgdata 2076maxresident)k  
0inputs+0outputs (0major+73minor)pagefaults 0swaps





# Măsurare timp – Linia de comandă

Operațiile de I/O sunt executate de Kernel

```
time dd if=/dev/zero of=file.txt count=1024 bs=1 048576
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 9.4847 s, 113 MB/s

real    0m9.490s
user    0m0.000s
sys    0m0.992s
```



# Măsurare timp cu sau fără I/O?



# Performanța

- Timp de execuție
- Memorie ocupată
- Număr de procese (thread-uri)
- Scalabilitate
- Toleranță la defecte
- Cost



# Măsuri

- T - Timpul total necesar execuției programului paralel
- P - Numărul de procesoare utilizate
- S – Speedup
  - $S = \frac{G}{T}$
- G – Timp execuție cel mai rapid algoritm secvențial



# Workflow - Testarea programelor

