POSIX TRAINING

On the Kalray MPPA Developer

With the AB01 and the TC2

Outline

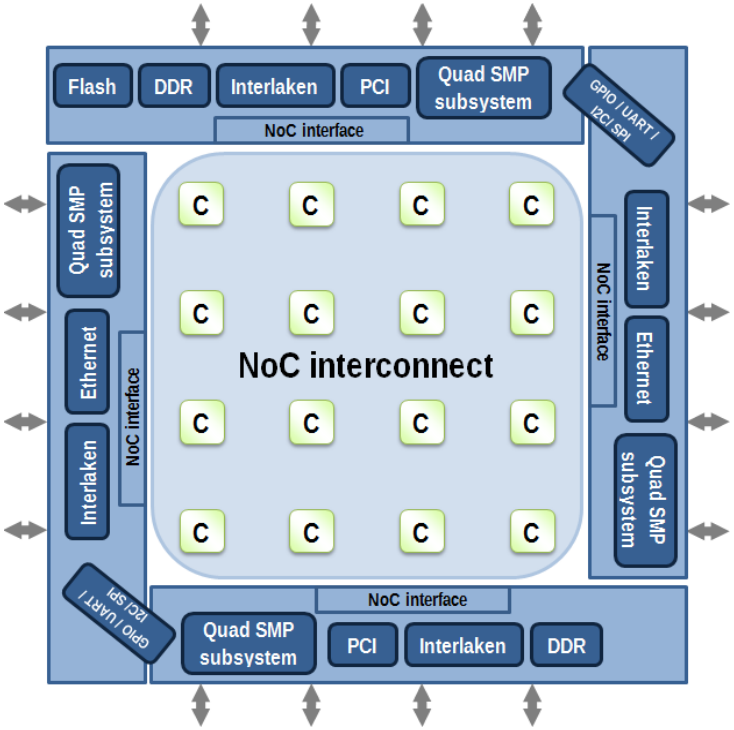
- MPPA Architecture:
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
  - ACCESSCORE-TOOLS
- Getting started with MPPA programming :
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

Outline

- MPPA Architecture:
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
  - ACCESSCORE-TOOLS
- Getting started with MPPA programming :
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

MPPA Architecture

- MPPA-256 Configuration

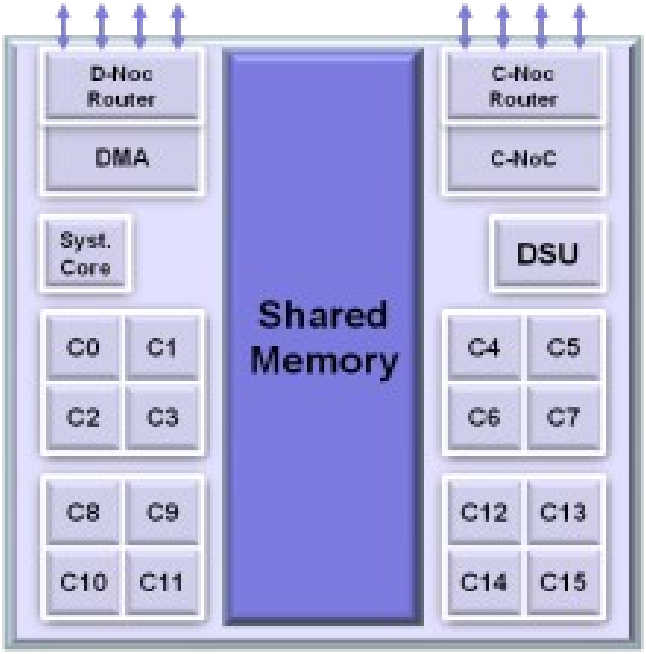


MPPA-256 chip I/O capabilities

- DDR3 Memory interfaces
- PCIe Gen3 interface
- 1G/10G/40G Ethernet interfaces
- SPI/I2C/UART interfaces
- Flash controller
- GPIOs
- Interlaken interface

MPPA Architecture

- Compute Cluster Configuration



The compute cluster

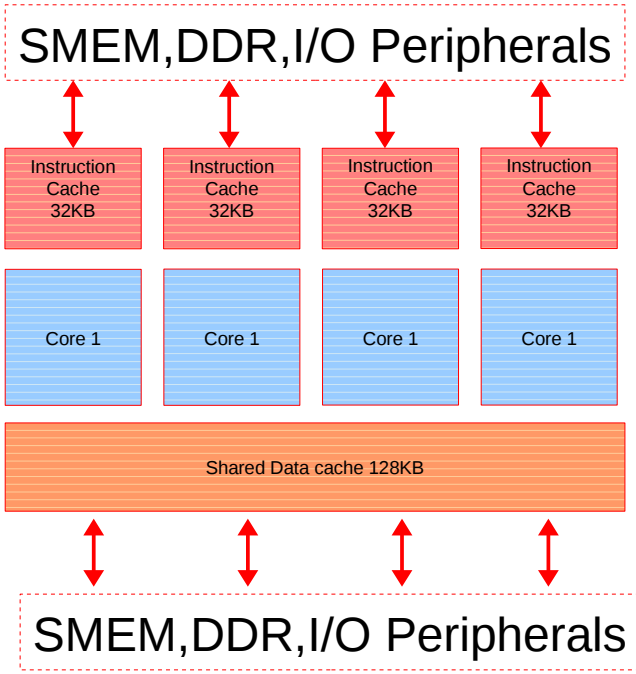
- 2MB of shared memory
- Processor :

  - 32 bit 5-way VLIW processor
  - Instruction cache :
    - 8KB
    - 2-way set associative
  - Data cache :
    - 8KB
    - 2-way set associative
- System core :

  - Events/interrupts connected to the DMA and control NoC
  - Events/interrupts connected to the 16 cores

MPPA Architecture

- I/O Cluster Configuration



The I/O cluster

- More than 4 GB of memory -DDR
- Core :

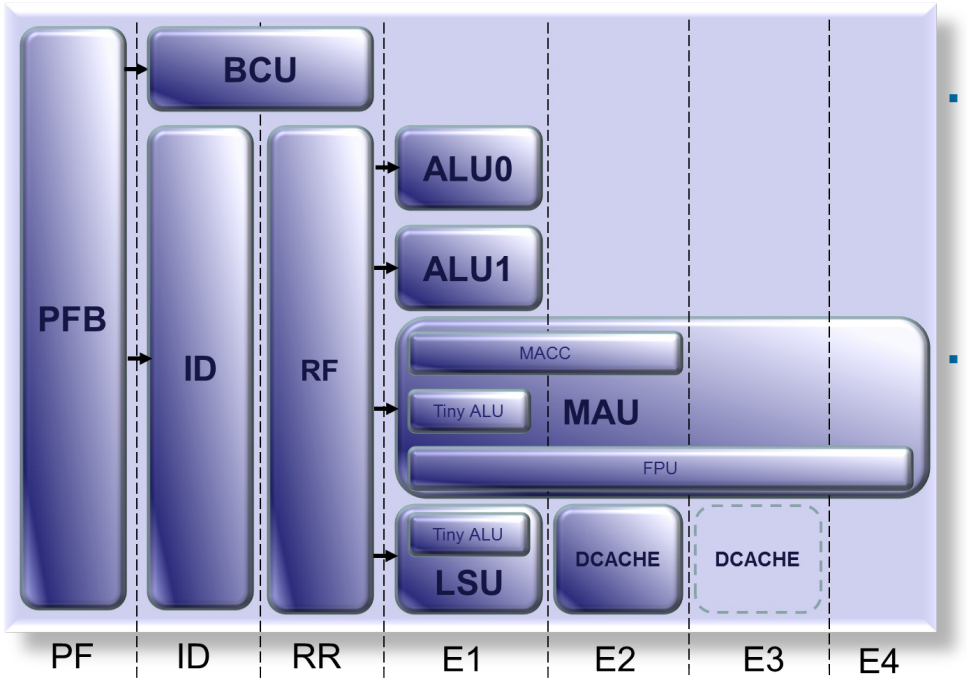
- 32 bit 5-way VLIW processor
- Instruction cache :
  - 32KB
  - 8-way set associative
- Data cache :
  - 128KB
  - 4 banks of 8-way set associative

SMEM 512Ko | DDR :4 GB | I/O Peripherals

Only for Os system

MPPA Architecture

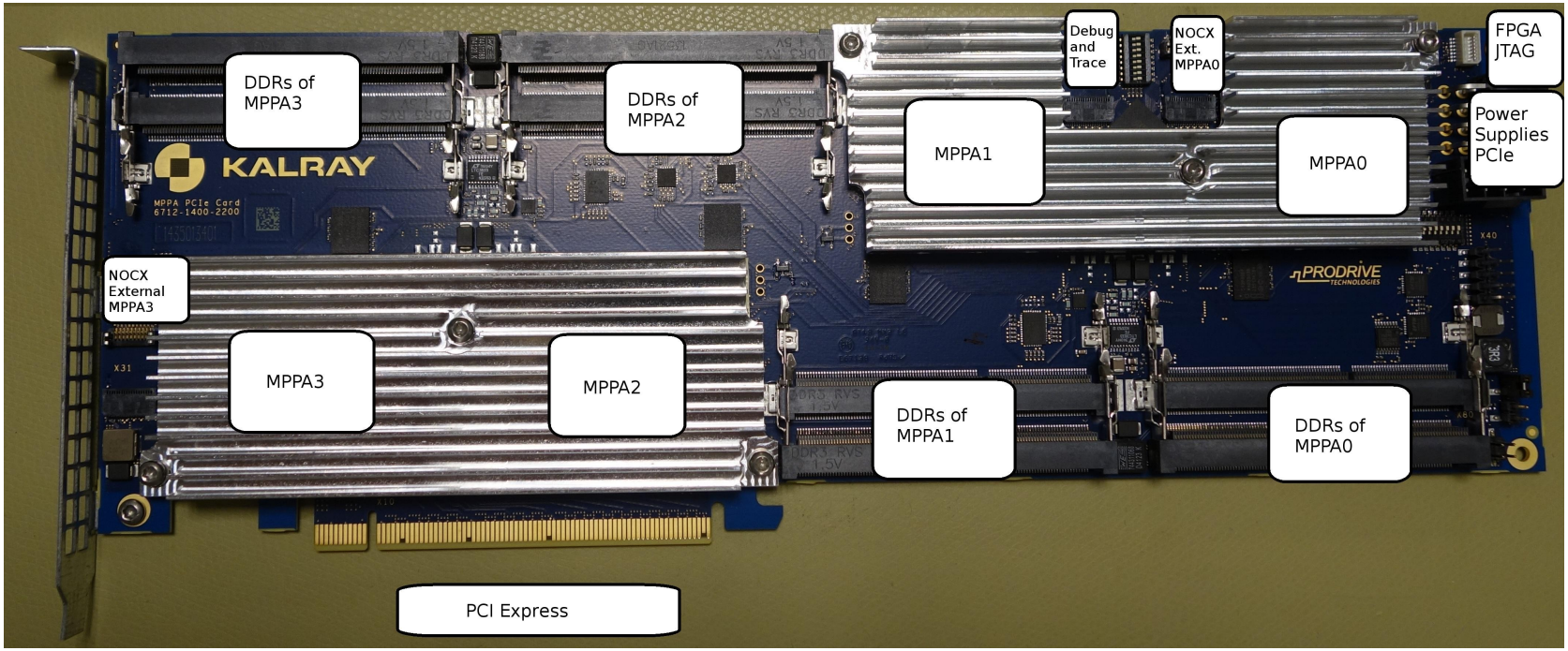
- K1 Processor



K1 processor

- 32 bit 5-way VLIW processor :
  - SIMD 2x32b & 4x16b capabilities
  - « DSP » &« MMX » operating modes
  - 7 pipeline stages for high power efficiency
  - IEEE 754 Floating Point Extended Fused Multiply – Add (FMA)
- Processing power (@400MHz)
  - 230 GFLOPS
  - 0,7 TOPS

TC2 Architecture

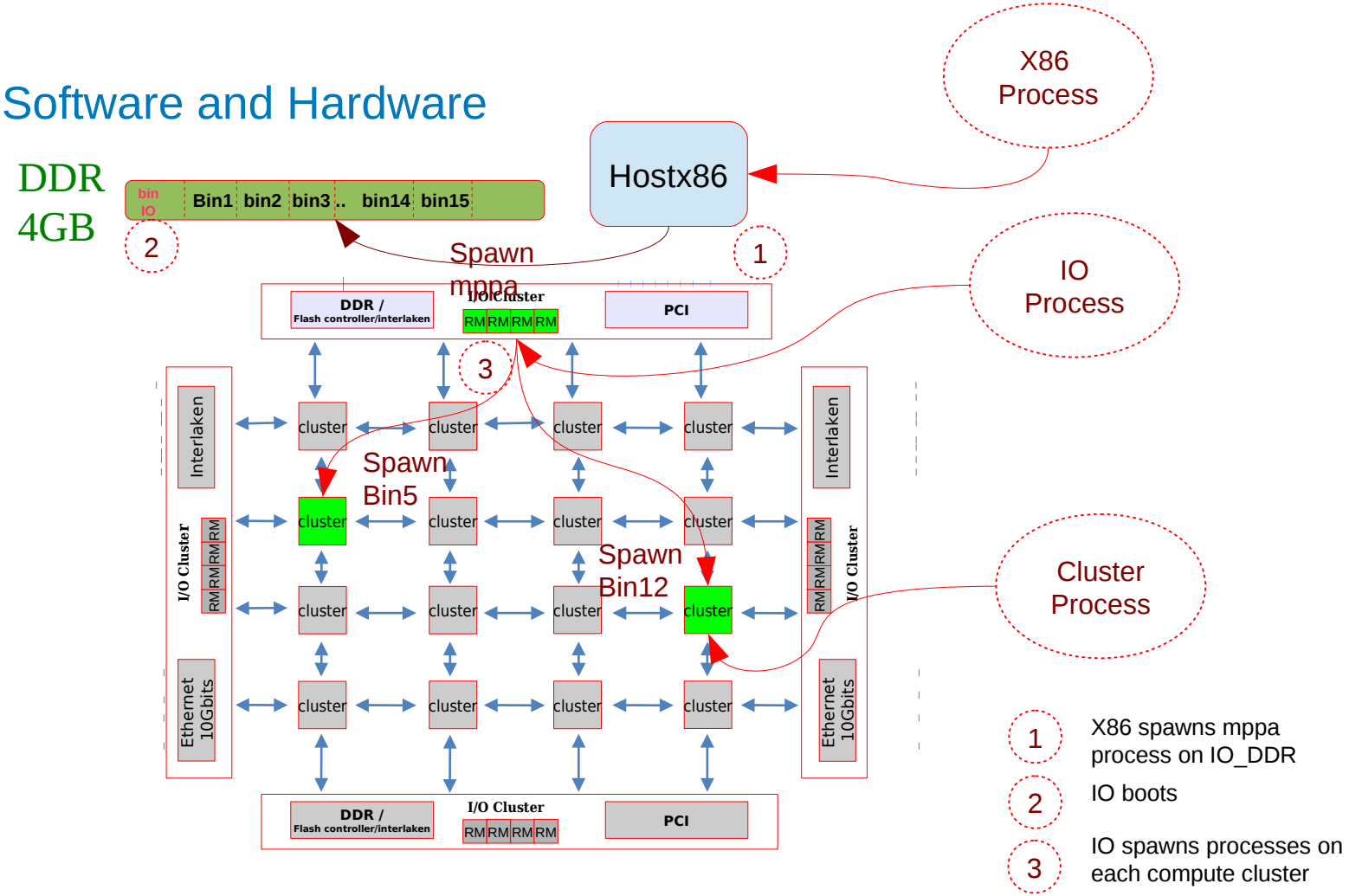


POSIX LEVEL

- MPPA Architecture
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
  - ACCESSCORE-TOOLS
- Getting started with MPPA programming:
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

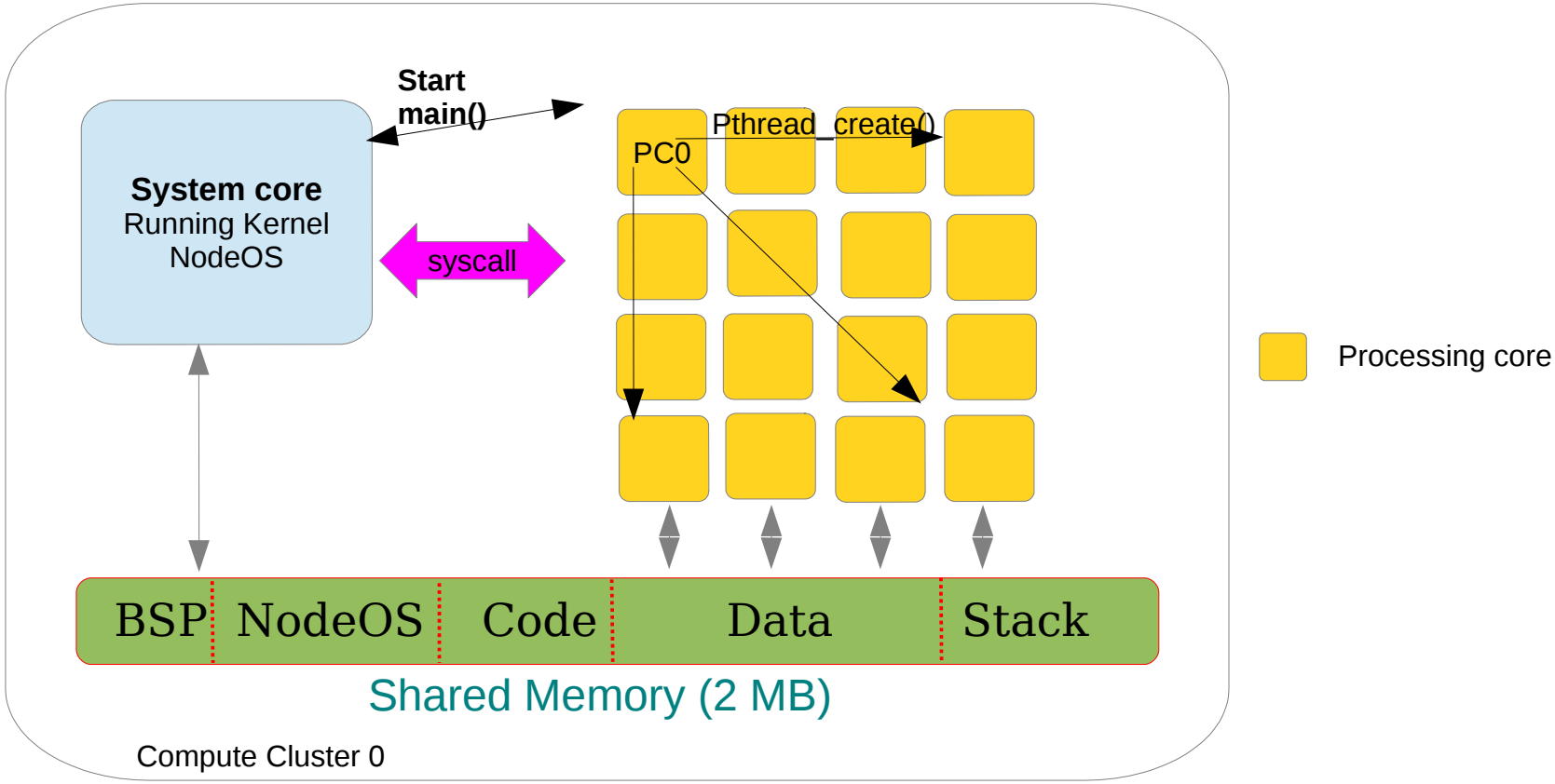
MPPA Architecture

- Software and Hardware



MPPA Architecture

- Software and Hardware

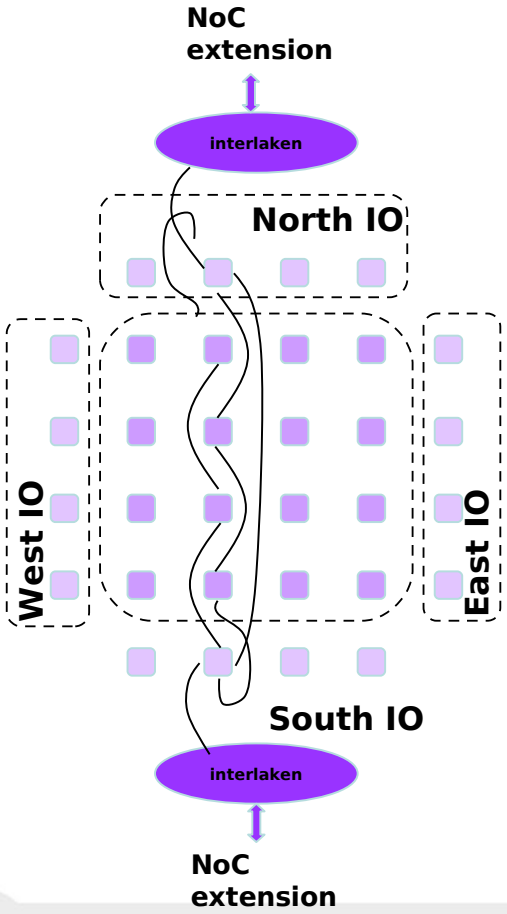


POSIX LEVEL

- MPPA Architecture
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
  - ACCESSCORE-TOOLS
- Getting started with MPPA programming:
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

MPPA Architecture

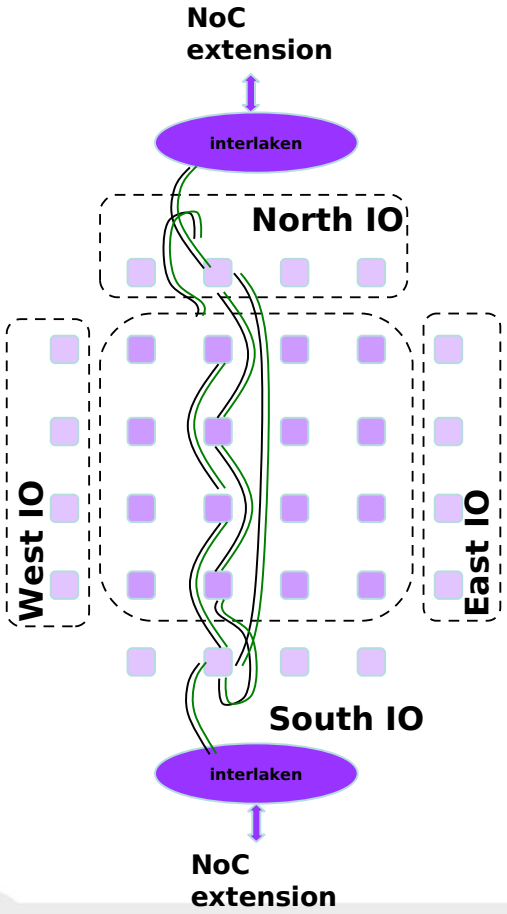
- Network-on-chip topology



– D-NoC : Data Network-on-chip (high bandwidth)

MPPA Architecture

- Network-on-chip topology



– D-NoC : Data Network-on-chip (high bandwidth)
– C-NoC : Control Network-on-chip (low bandwidth)

Outline

- MPPA Architecture:
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
  - ACCESSCORE-TOOLS
- Getting started with MPPA programming :
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

ACCESSCORE-TOOLS

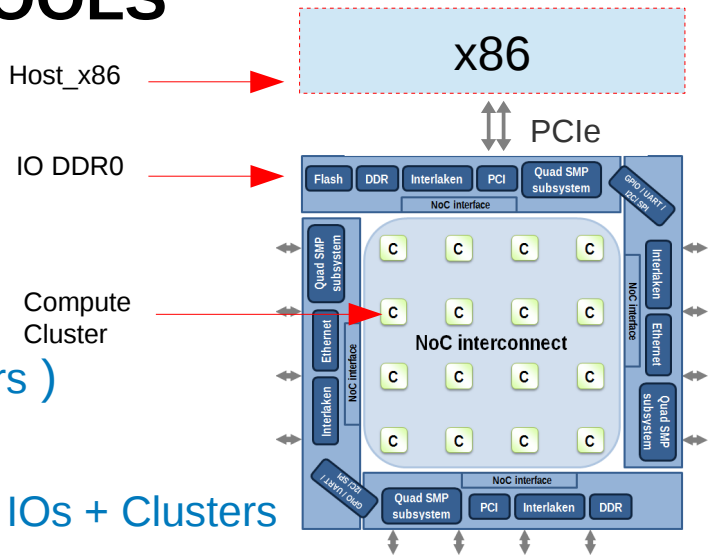
- Target Simulators & Hardware
- Profilers
- System Tracing
- Debugger
- k1-power
- Eclipse compatibility

ACCESSCORE-TOOLS

- Target Simulators & Hardware
- Profilers
- System Tracing
- Debugger
- k1-power
- Eclipse compatibility

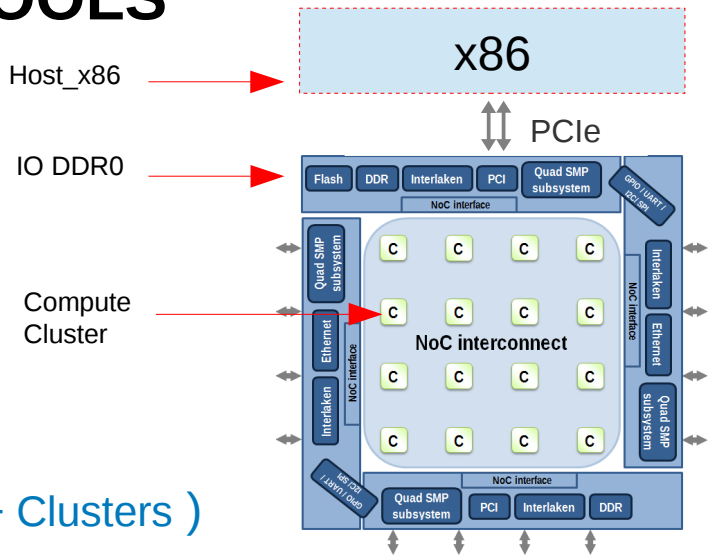
ACCESSCORE-TOOLS

- Target Simulators :
  - + Simulator for cluster :
    - $k1-cluster
  - + Simulator for MPPA (IOs + Clusters)
    - $k1-mppa
  - + Simulator for Host_x86 + MPPA (IOs + Clusters
    - $k1-pciesim-runner
  - (functional or cycle-based precision)



ACCESSCORE-TOOLS

- Target Hardware :
  - + JTAG : Cluster :
    - $k1-jtag-runner
  - + JTAG : MPPA (IOs + Clusters)
    - $k1-jtag-runner
  - + (PCIe) : Host_x86 + MPPA (IOs + Clusters)
    - $./host_x86.bin mppamultibin.bin

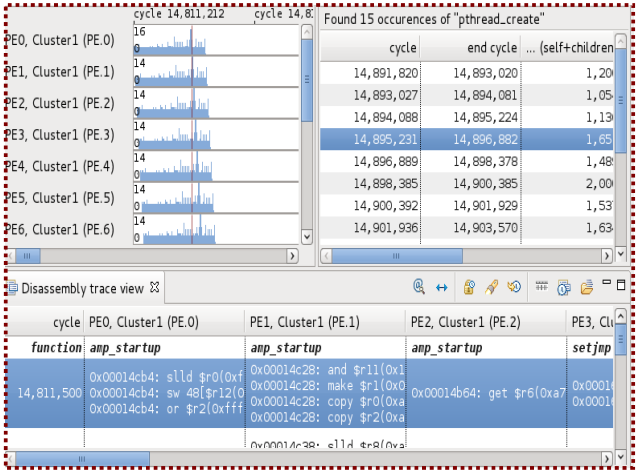


ACCESSCORE-TOOLS

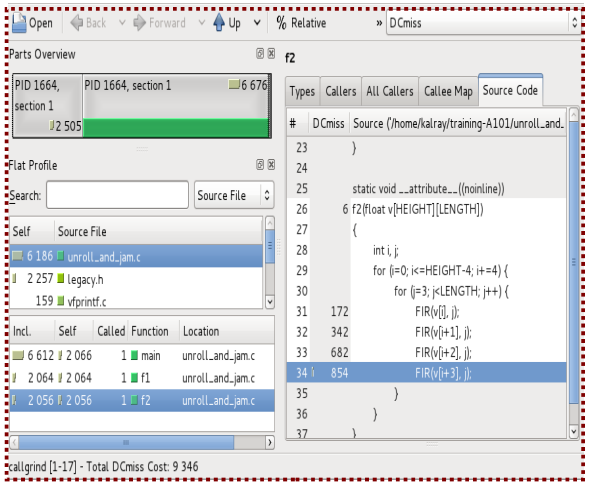
- Target Simulators & Hardware
- Profilers
- System Tracing
- Debugger
- Eclipse compatibility

ACCESSCORE-TOOLS

- Profiler :
  - On Simulator :
    - K1-DTV : The K1 Disassembly Trace
    - Callgrind Profiling
  - On Hardware :
    - Performance Monitor Counter



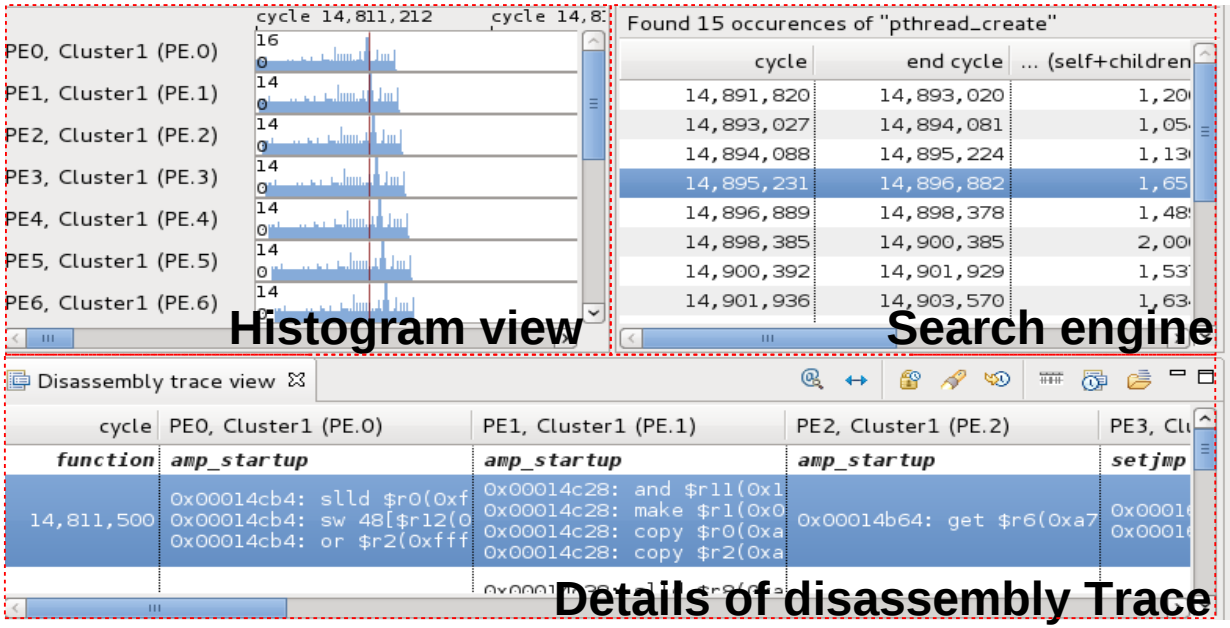
k1-dtv : Disassembly trace



Kcachegrind

ACCESSCORE-TOOLS

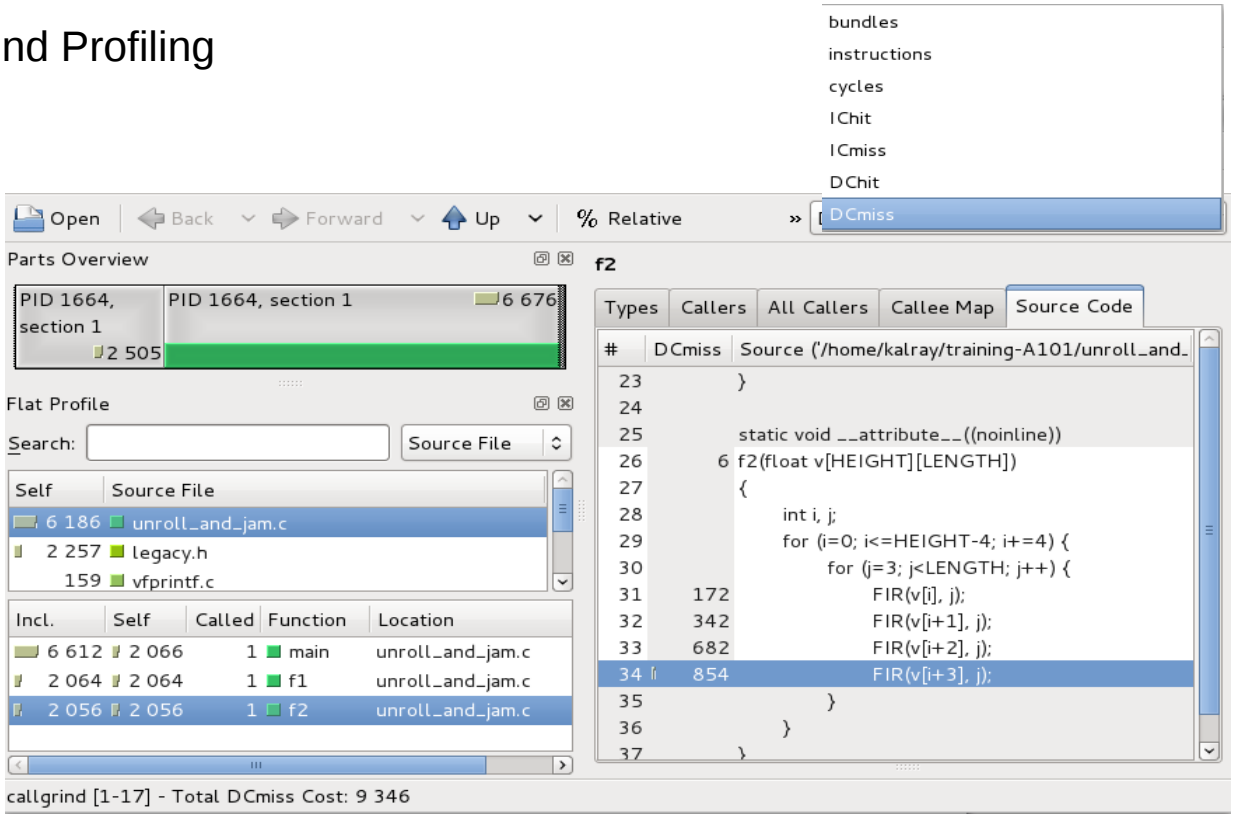
- Profiler :
  - On Simulator :
    - K1-DTV : The K1 Disassembly Trace



ACCESSCORE-TOOLS

- Profiler :
  - On Simulator :
    - Callgrind Profiling

Profiling information :
– Executed bundles
– Executed instruction (Equivalent to executed bundles in functional simulation)
– Executed cycles
– Instruction cache hits count
– Instruction cache misses count
– Data cache hits count
– Data cache misses count



ACCESSCORE-TOOLS

- Profiler :
  - On Hardware :
    - Performance Monitor Counter

DSU-CLUSTERS/MPPA
Global counter 64 b (cycle)

Type **:**

__k1_read_dsu_timestamp()

=> Very quick to profile an application !

K1-CORES Counters
4 counter 32 b / core

Type :

__k1_counter_*

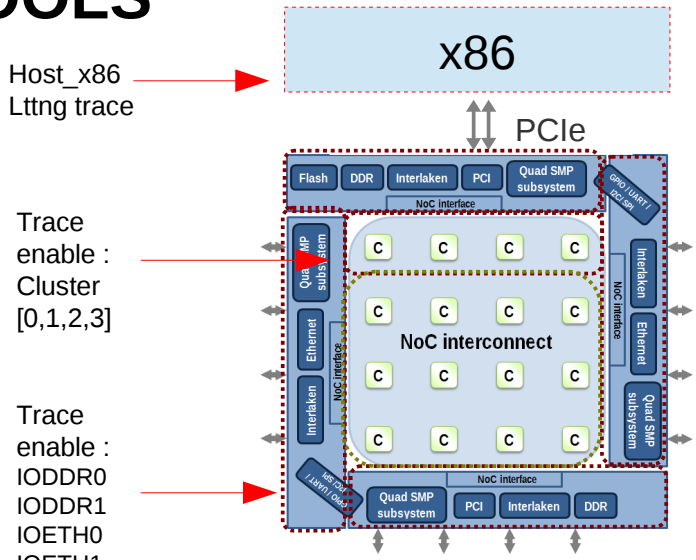
Value	Condition
0x0	Clocks cycles (default)
0x1	Instruction cache hits number
0x2	Instruction cache misses number
0x3	Instruction cache misses stalls
0x4	Data cache hits number
0x5	Data cache misses number
0x6	Data cache misses stalls
0x7	Bundles executed number
0x8	Branches and re-fetches number
0x9	Branches and re-fetches stalls
0xA	Register dependence stalls
0xB	Instruction µTLB refill stalls
0xC	Data µTLB refill stalls
0xD	Streaming load buffer stalls
0xE	Stop counter
0xF	Reset counter (to zero)

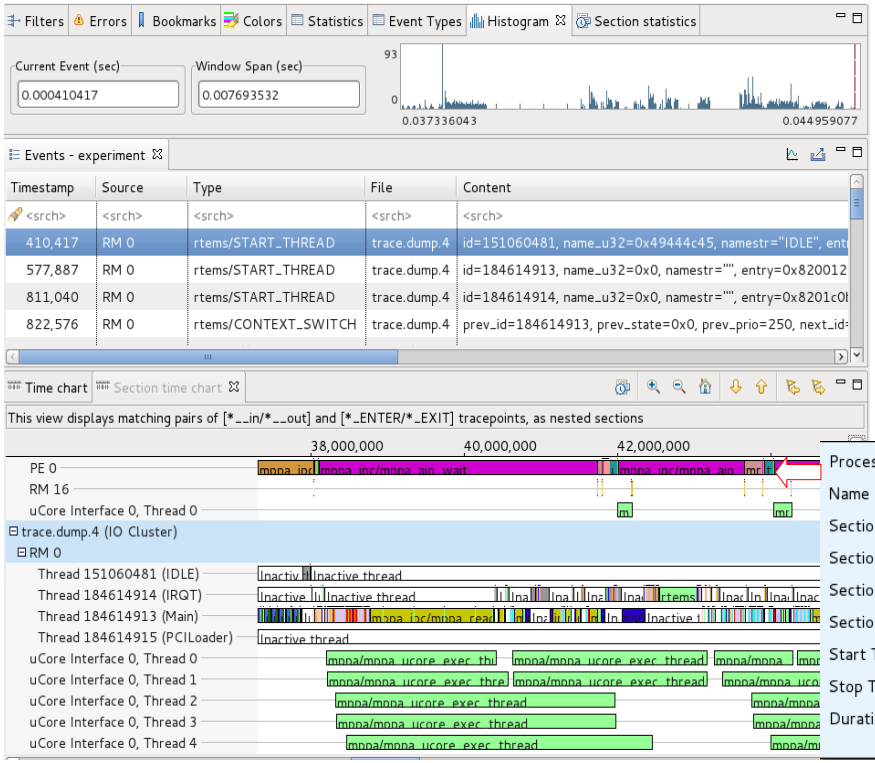
ACCESSCORE-TOOLS

- Target Simulators & Hardware
- Profilers
- System Tracing
- Debugger
- k1-power
- Eclipse compatibility

ACCESSCORE-TOOLS

- System Tracing :





Processor/Thread	uCore Interface 0, Thread 0
Name	mppa/mppa_ucore_exec_thread
Section start	44,339,871
Section end	48,144,219
Section duration (self)	3,804,348
Section duration (self+children)	3,804,348
Start Time	44,339,871
Stop Time	48,144,219
Duration	3,804,348

ACCESSCORE-TOOLS

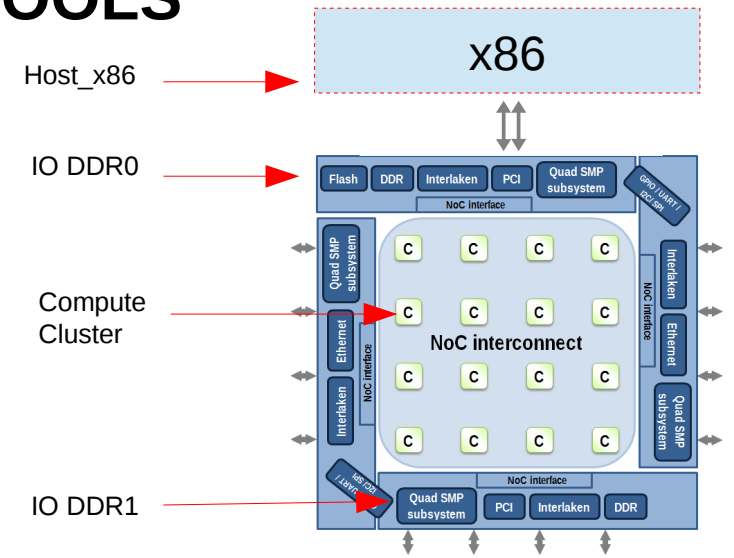
- Target Simulators & Hardware
- Profilers
- System Tracing
- Debugger
- k1-power
- Eclipse compatibility

ACCESSCORE-TOOLS

- Debugger :
  - K1 GDB on Harware&Simulator

Using GDB to debug a MPPA application
+ K1 processors as threads
+ Hot-Attach (with PCI application)

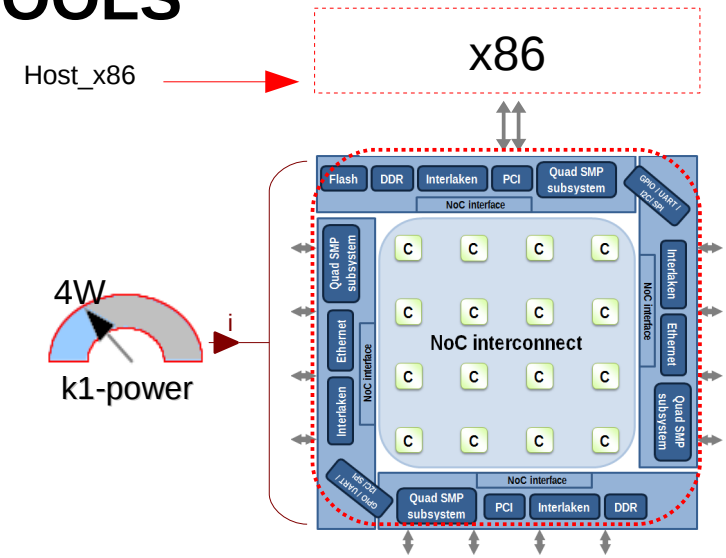
```
 19   PE 13 of Cluster 0 (running)
 18   PE 12 of Cluster 0 (running)
 17   PE 11 of Cluster 0 (running)
 16   PE 10 of Cluster 0 (running)
 15   PE 9 of Cluster 0 (running)
 14   PE 8 of Cluster 0 (running)
 13   PE 7 of Cluster 0 (running)
 12   PE 6 of Cluster 0 (running)
 11   PE 5 of Cluster 0 (running)
 10   PE 4 of Cluster 0 (running)
 9    PE 3 of Cluster 0 (running)
 8    PE 2 of Cluster 0 (running)
 7    PE 1 of Cluster 0 (running)
 6    PE 0 of Cluster 0 (running)
 5    RM of Cluster 0   (running)
 4    RM 3 of Cluster 128 (running)
 3    RM 2 of Cluster 128 (running)
 2    RM 1 of Cluster 128 (running)
* 1   RM 0 of Cluster 128 0x00000000 in _start ()
(gdb) list
24
25              ////////////////////////////////
26              // Global variables declaration //
27              ////////////////////////////////
28
29              int status;
30              mppa_pid_t pids[CLUSTER_COUNT];
31              int rank;
32              int ret;
33              int i;
(gdb)
```



ACCESSCORE-TOOLS

- Target Simulators & Hardware
- Profilers
- System Tracing
- Debugger
- k1-power
- Eclipse compatibility

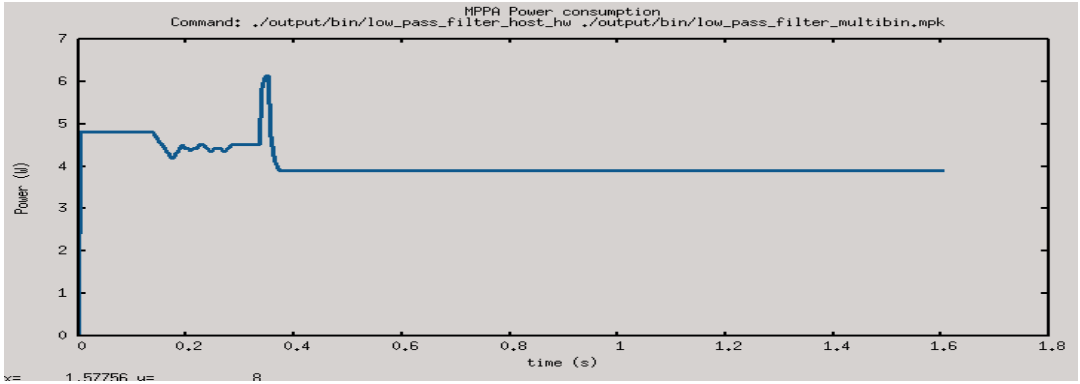
ACCESSCORE-TOOLS



- k1-power
  - Average :
    - $k1-power – k1-jtag-runner ****

Time : 1.61 s
Power (avg): 4.07 W
Energy : 6.54 J

  - In real time
    - $k1-power -oplot -gx11 – k1-jtag-runner ****

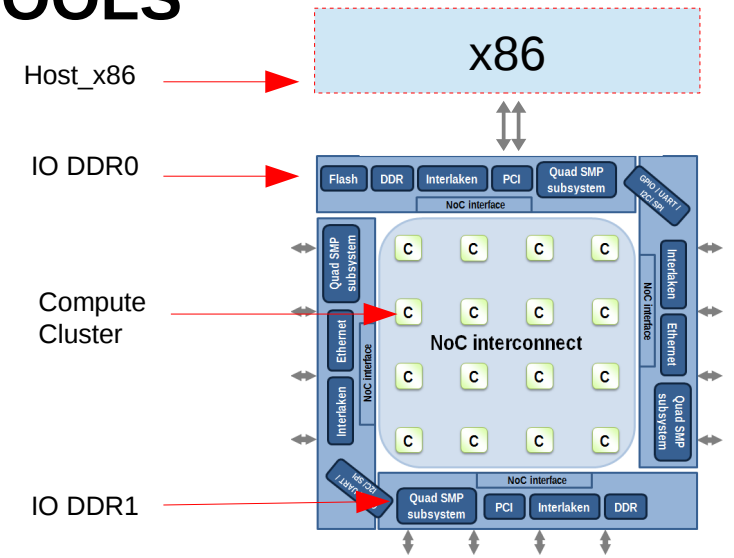


ACCESSCORE-TOOLS

- Target Simulators & Hardware
- Profilers
- System Tracing
- Debugger
- k1-power
- Eclipse compatibility

ACCESSCORE-TOOLS

- Eclipse compatibility :
  - Easy to start
  - Kalray Makefile support
  - Hardware & Simulator Debugger



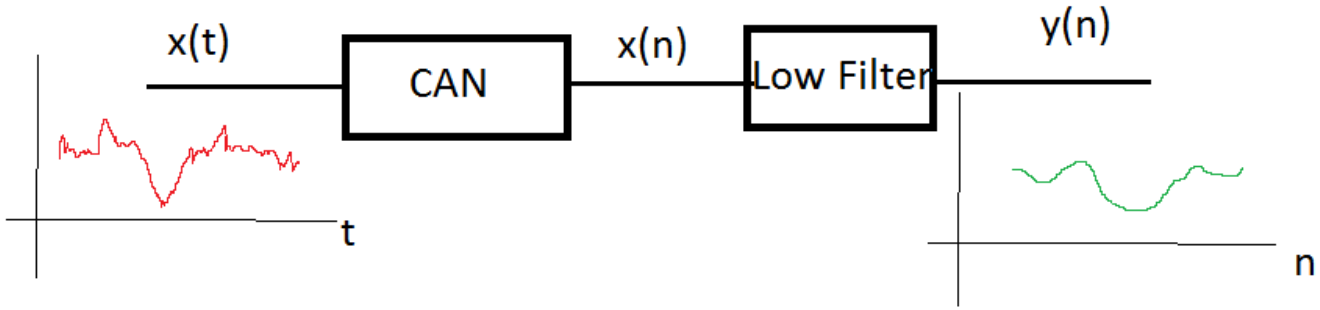
POSIX LEVEL

- MPPA Architecture
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
- Getting started with MPPA programming:
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

Getting started with MPPA programming:

- How to program a low-pass filter application

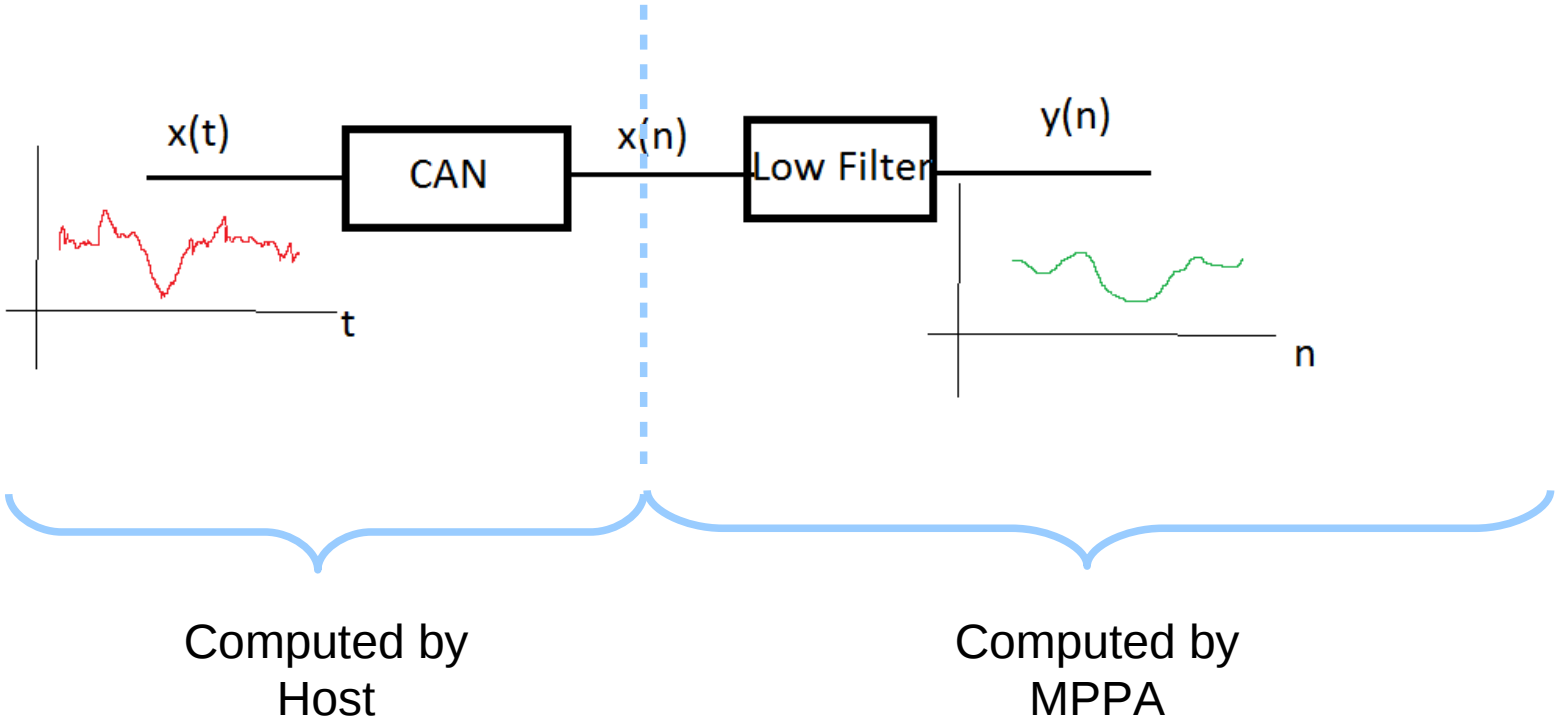
$y(n) = x * h = \Sigma(x(n-k) * coef(k))$



Getting started with MPPA programming:

- How to program a low-pass filter application

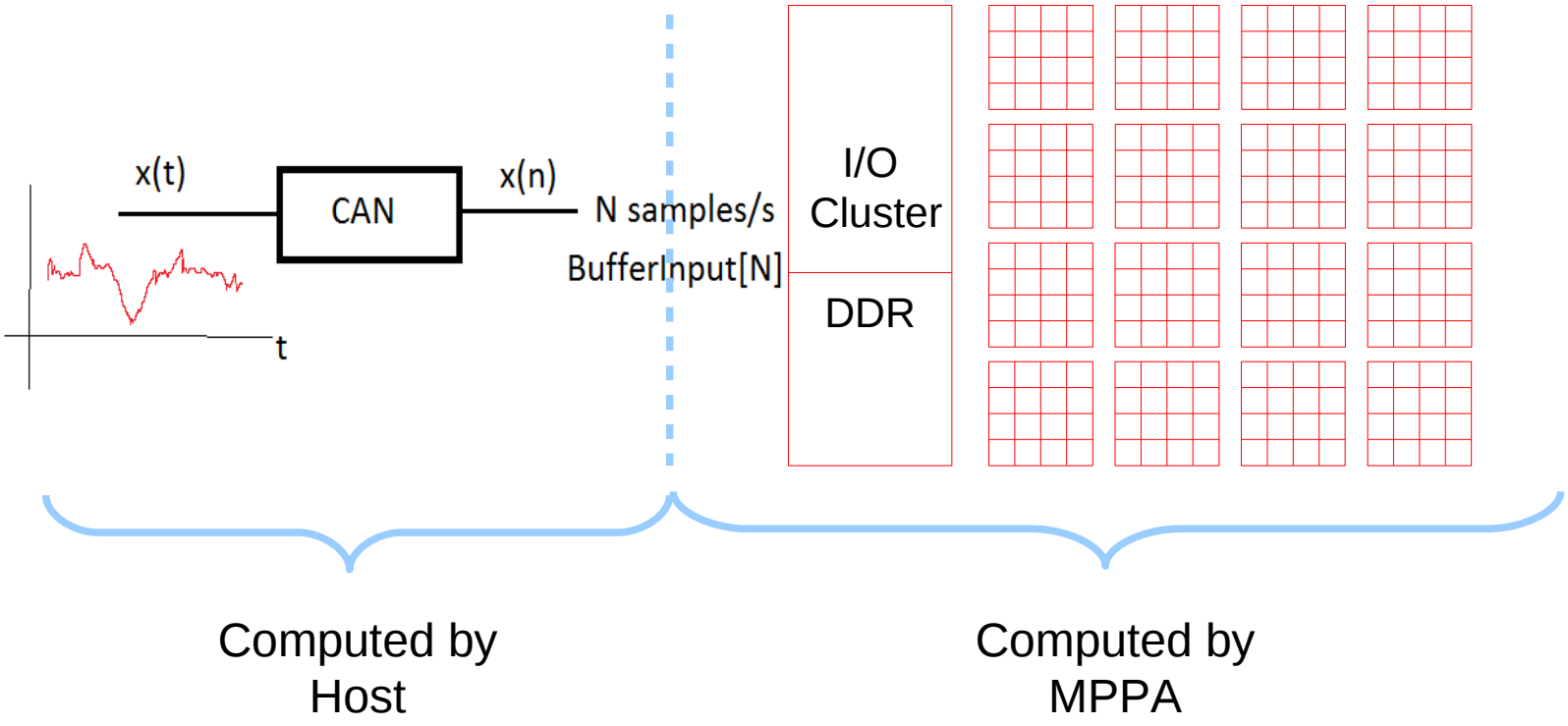
$y(n) = x * h = \Sigma(x(n-k) * coef(k))$



Getting started with MPPA programming:

- How to program a low-pass filter application

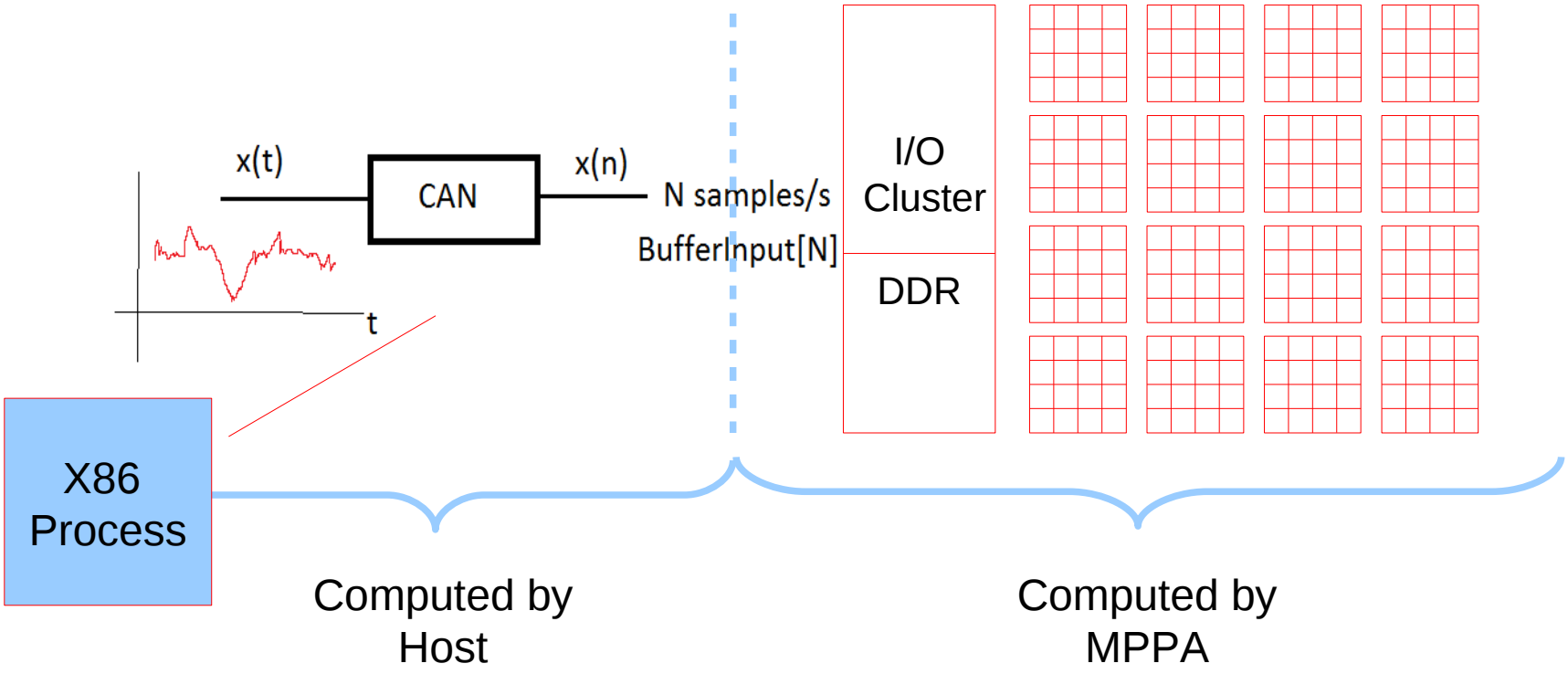
$y(n) = x * h = \Sigma(x(n-k) * coef(k))$



Getting started with MPPA programming:

- How to program a low-pass filter application

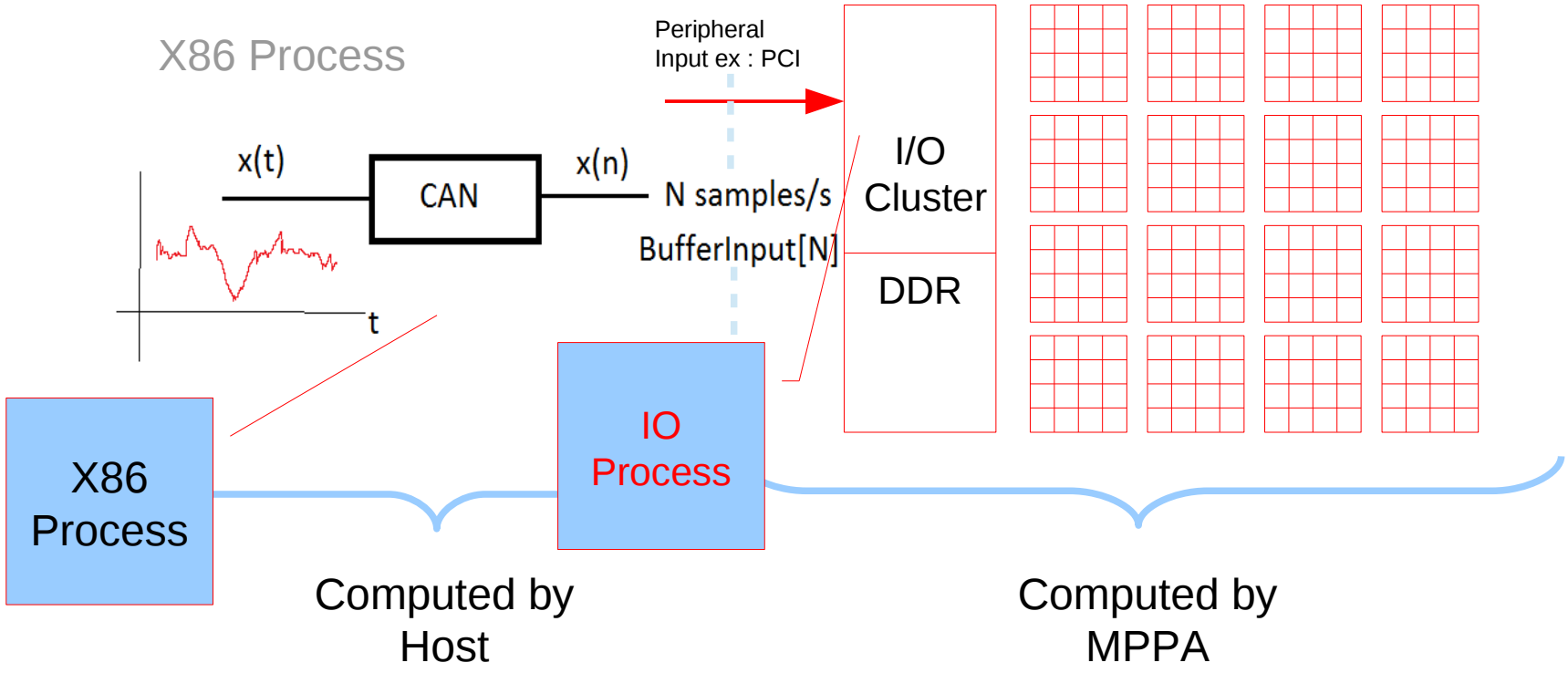
$y(n) = x * h = \Sigma(x(n-k) * coef(k))$



Getting started with MPPA programming:

- How to program a low-pass filter application

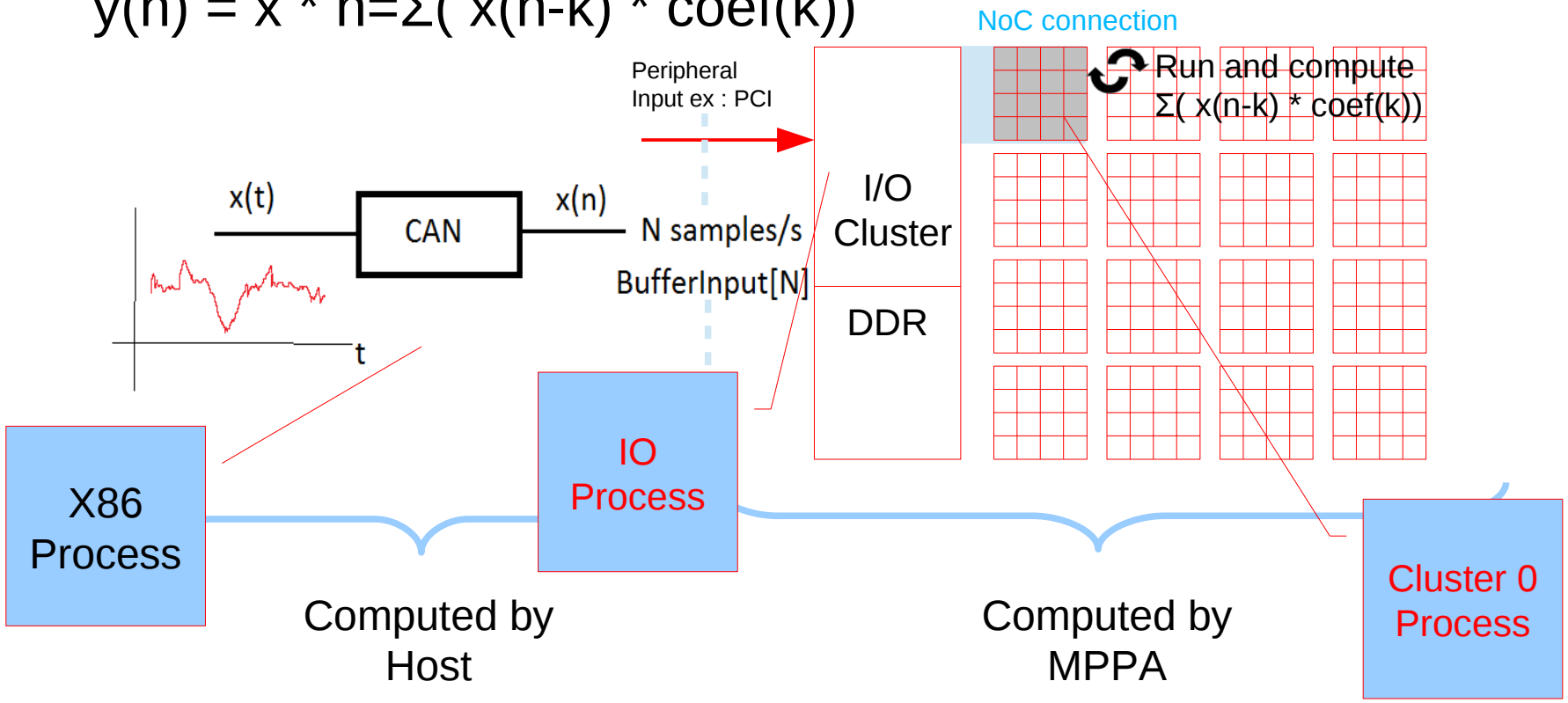
$y(n) = x * h = \Sigma(x(n-k) * coef(k))$



Getting started with MPPA programming:

- How to program a low-pass filter application

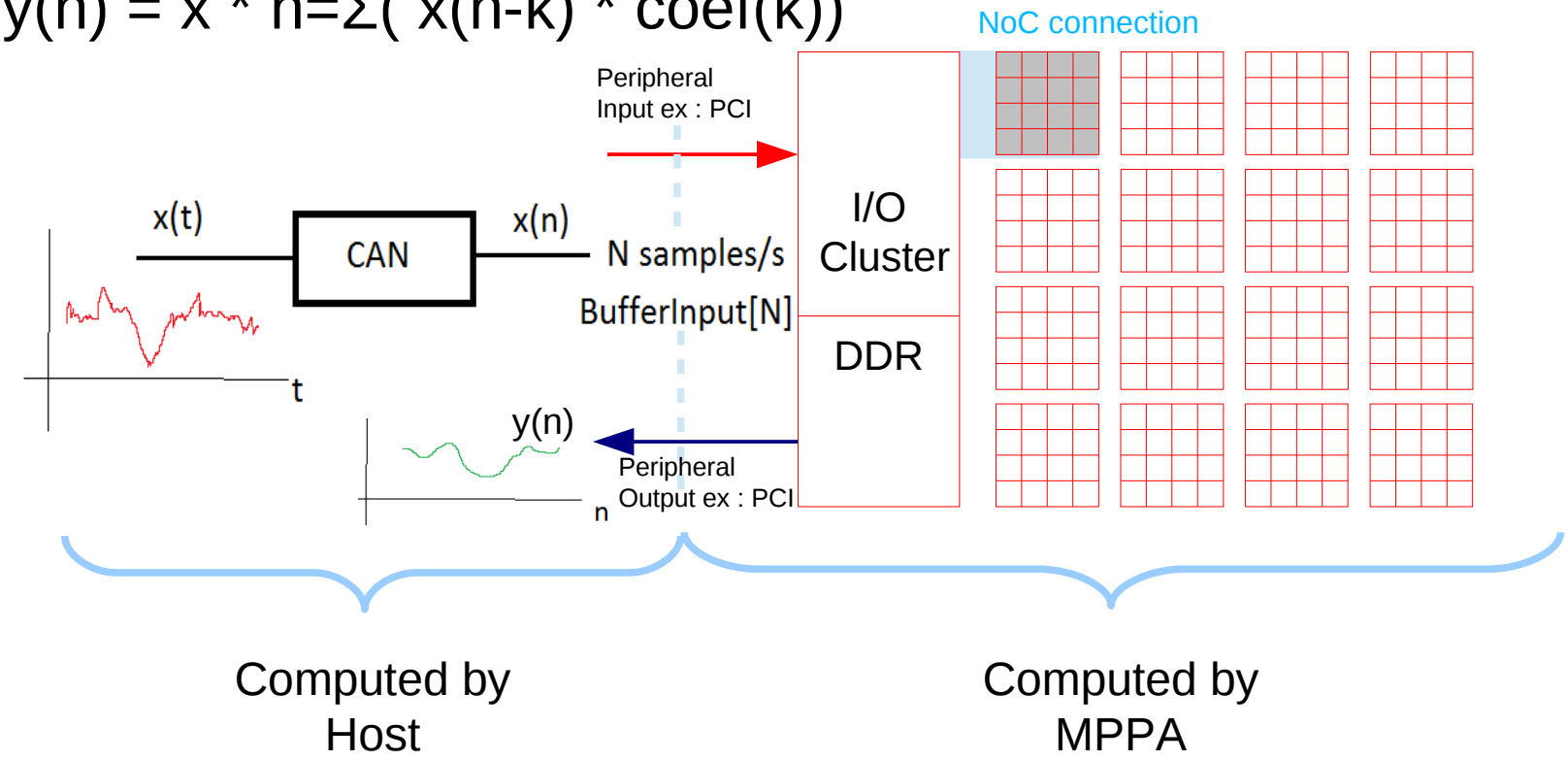
$y(n) = x * h = \Sigma(x(n-k) * coef(k))$



Getting started with MPPA programming:

- How to program a low-pass filter application

$y(n) = x * h = \Sigma(x(n-k) * coef(k))$



How to program a low-pass filter application

- Methodology to implement a low-pass filter example :

I- Implement example on Cluster_1.

(already done)

II- Add NoC communication : Cluster_IODDR with Cluster_1.

(exercise 1&2)

III- Parallelize computation on Cluster_1.

(exercise 3)

IV- Add PCI Communication : Host_x86 with Cluster_IODDR

(exercise 4)

POSIX LEVEL

- MPPA Architecture
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
- Getting started with MPPA programming:
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Channel Transfers
  - Queue Transfers
  - Portal Transfers
- MPPA IPC synchronisation type :
  - Sync

Inter-Cluster Communication presentation

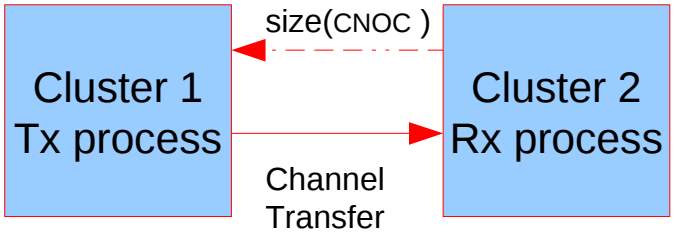
- MPPA IPC Transfer types :
  - Channel Transfers

Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Channel Transfers

The Channel connector operations carry out a rendez-vous between a Tx process and a Rx process, paired with zero-copy data transfer.

The channel is used for example to transfer data from Cluster_1 to Cluster_2.

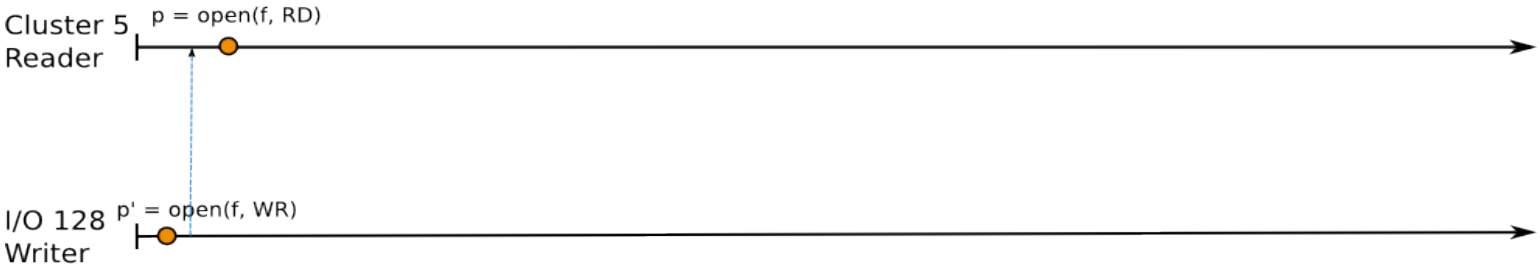


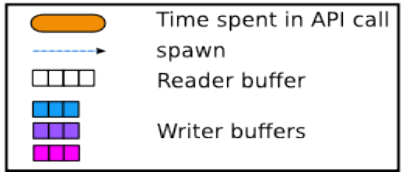
Inter-Cluster Communication presentation

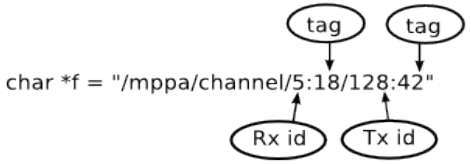
- MPPA IPC Transfer types :
  - Channel Transfers

step 1 : The IO_DDR opens a channel connector for the Tx Process and spawns the « channel-slave.bin » process to cluster_5

step 2 : The Cluster_5 opens a channel connector for the Rx Process.







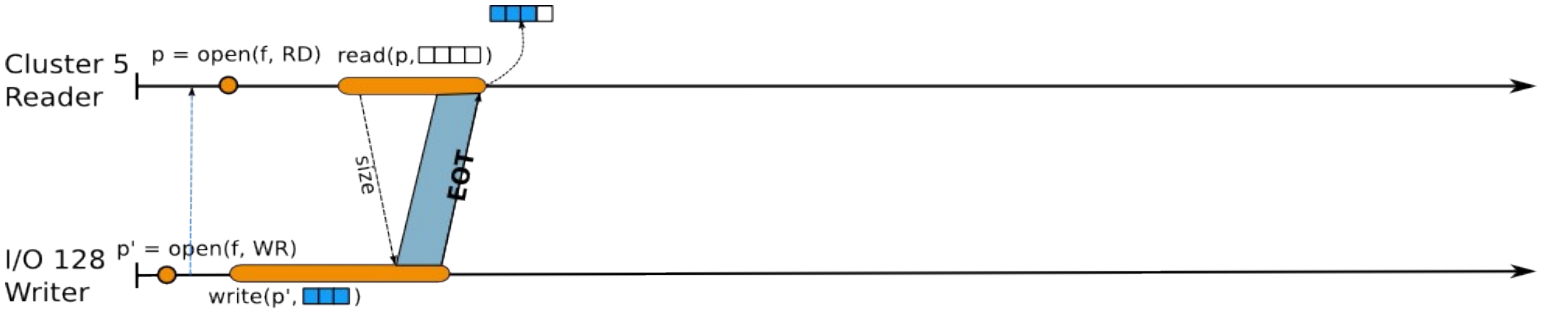
Inter-Cluster Communication presentation

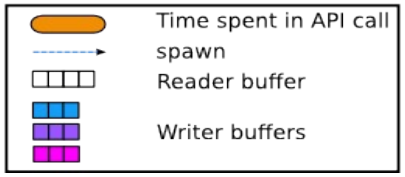
- MPPA IPC Transfer types :
  - Channel Transfers

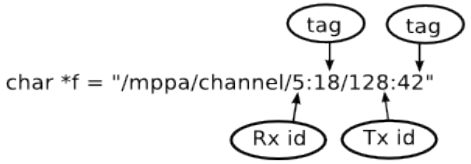
step 3 : The Tx process starts (write()) and is blocked until the buffer is sent.

step 4 : The Rx process starts, the maximum buffer size is sent by the CNOC to the Tx process.

step 5 : At the End Of Transmission, the Tx process sends an EOT notification to the Rx process.



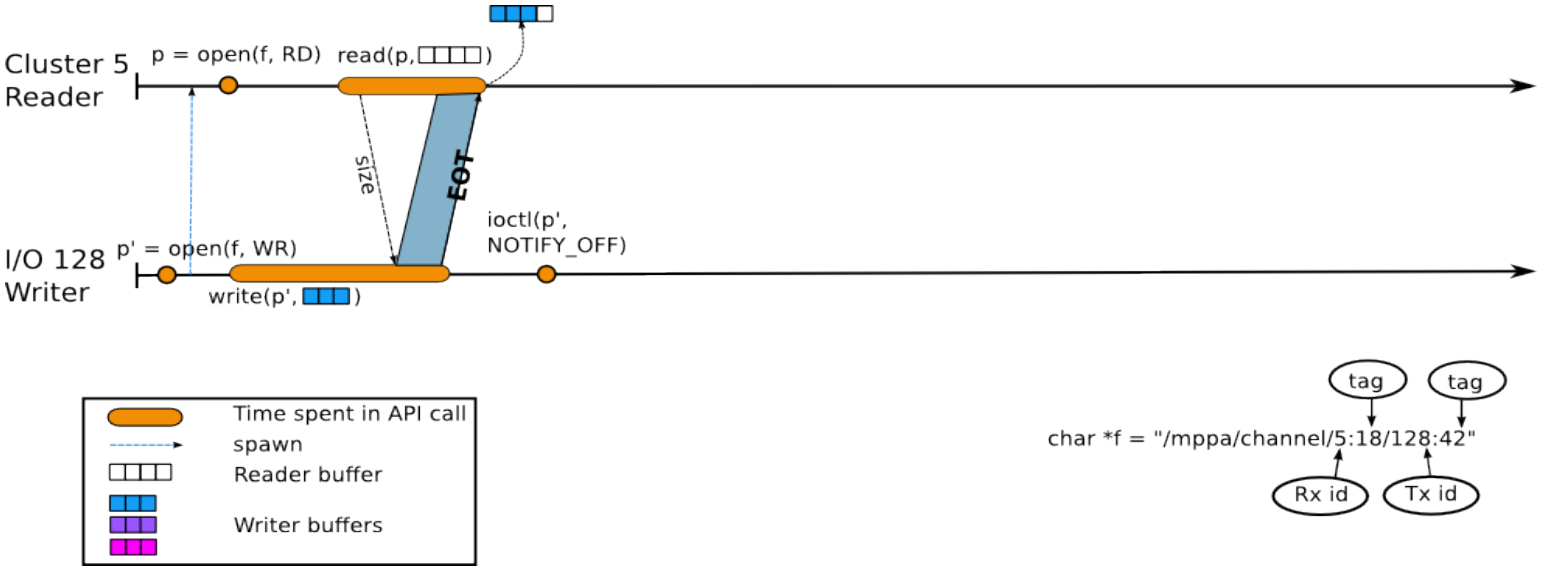




Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Channel Transfers

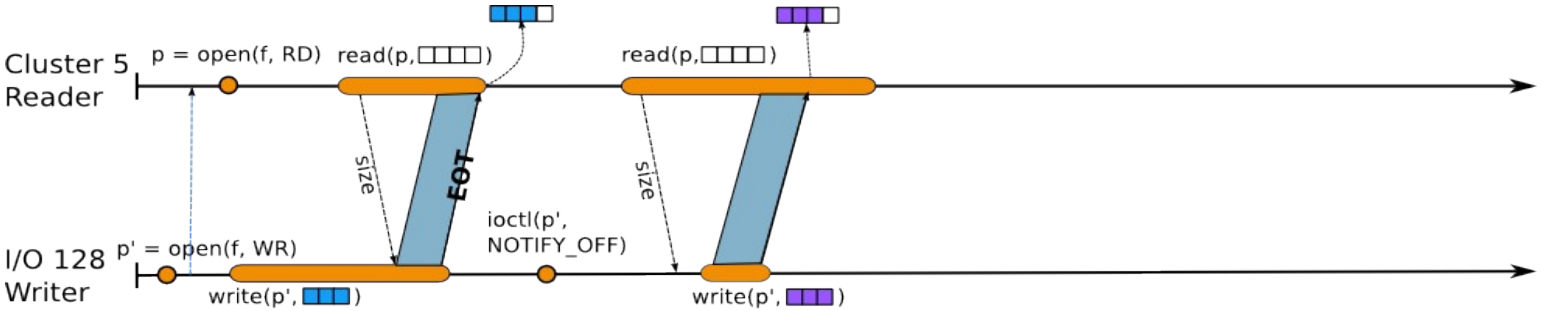
step 1b : The Tx disables the EOT notification of its process « ioctl (NOTIFICATION_OFF) »

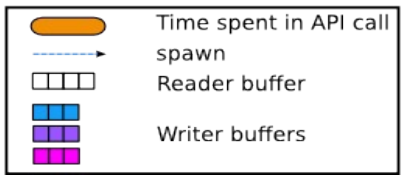


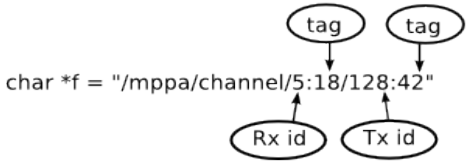
Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Channel Transfers

step 2b : The Tx process sends all of its data, but the Rx process waits for the EOT notification from the Rx process





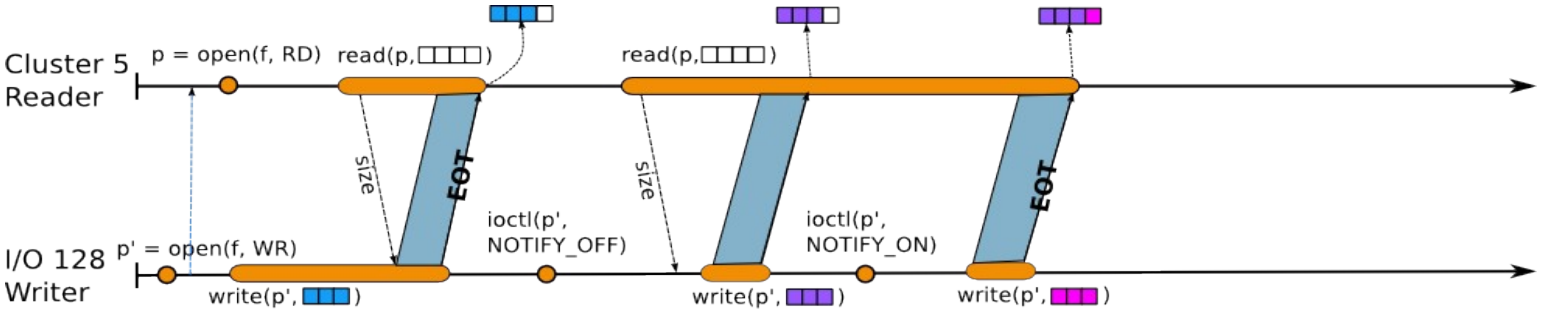


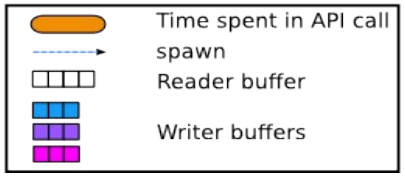
Inter-Cluster Communication presentation

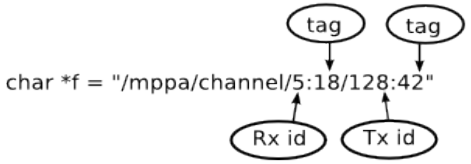
- MPPA IPC Transfer types :
  - Channel Transfers

step 3b : The Tx enables the EOT notification of its process « ioctl (NOTIFICATION_ON) »

step 4b : The Tx resends a new buffer and fills only the part of the Rx buffer that is empty, and the Tx finishes its transmission with a EOT notification.





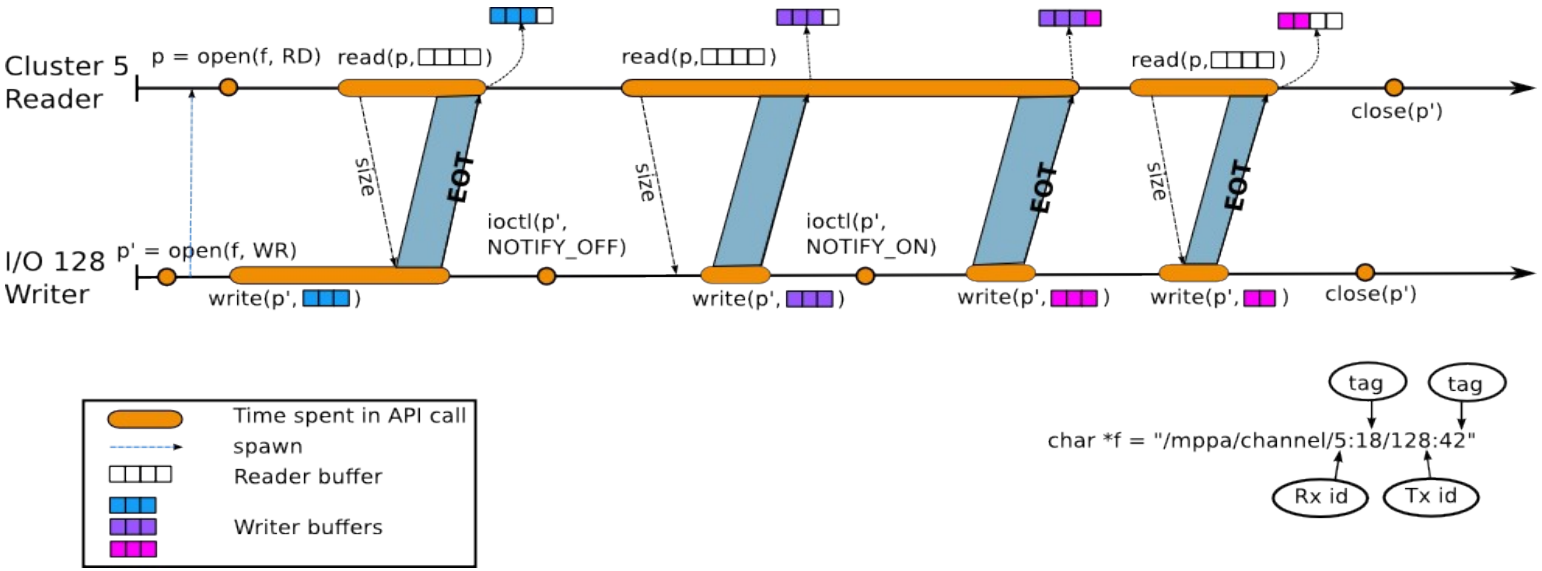


Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Channel Transfers

Notification is by default enabled,

last step: close and free the communication configuration.

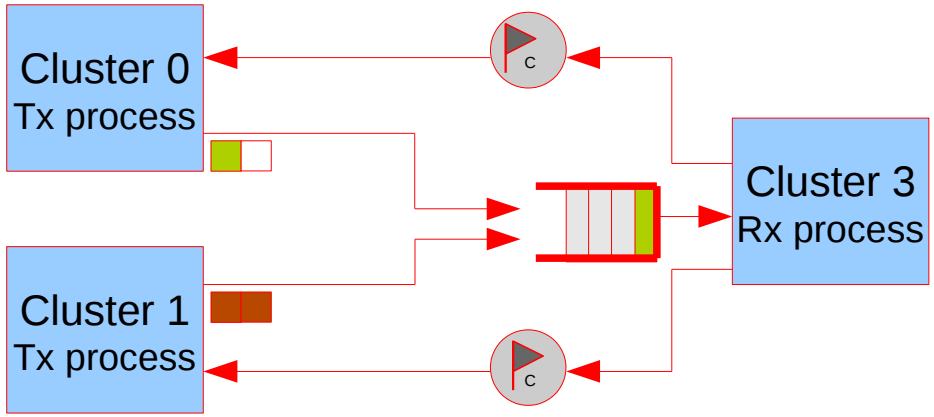


Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

Credit based fixed sized messaging: messages stored in the memory area of the Rx process, where N Tx processes can en-queue messages of msize bytes. The maximum msize value supported is 120.

The Rqueue Transfers are used for example to send requests, or for exemple to send small packets containing a matrix descriptor (case of matrice multiply application).

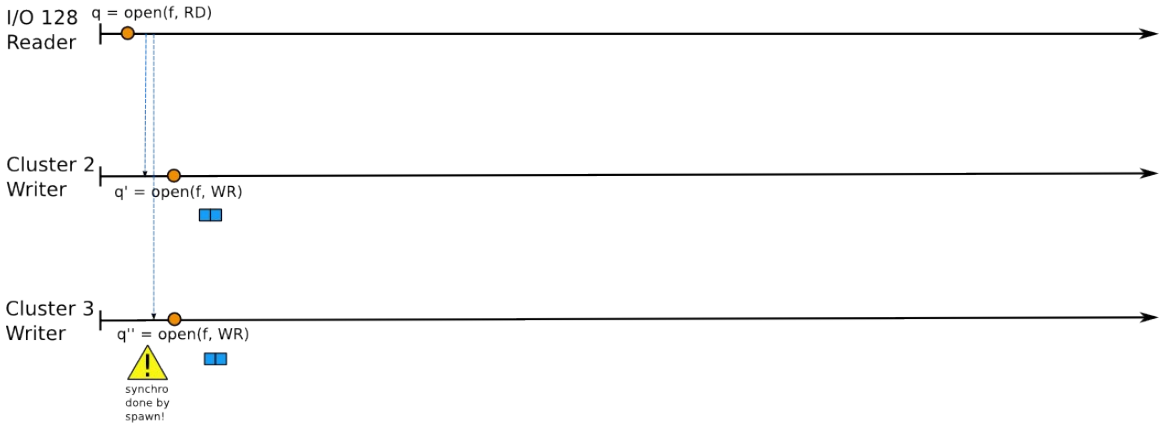


Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

step 1 : The IO_DDR opens Rqueue connectors for the Rx Process and spawns the «Rqueue-slave1&2.bin » process to cluster_2 and cluster_3

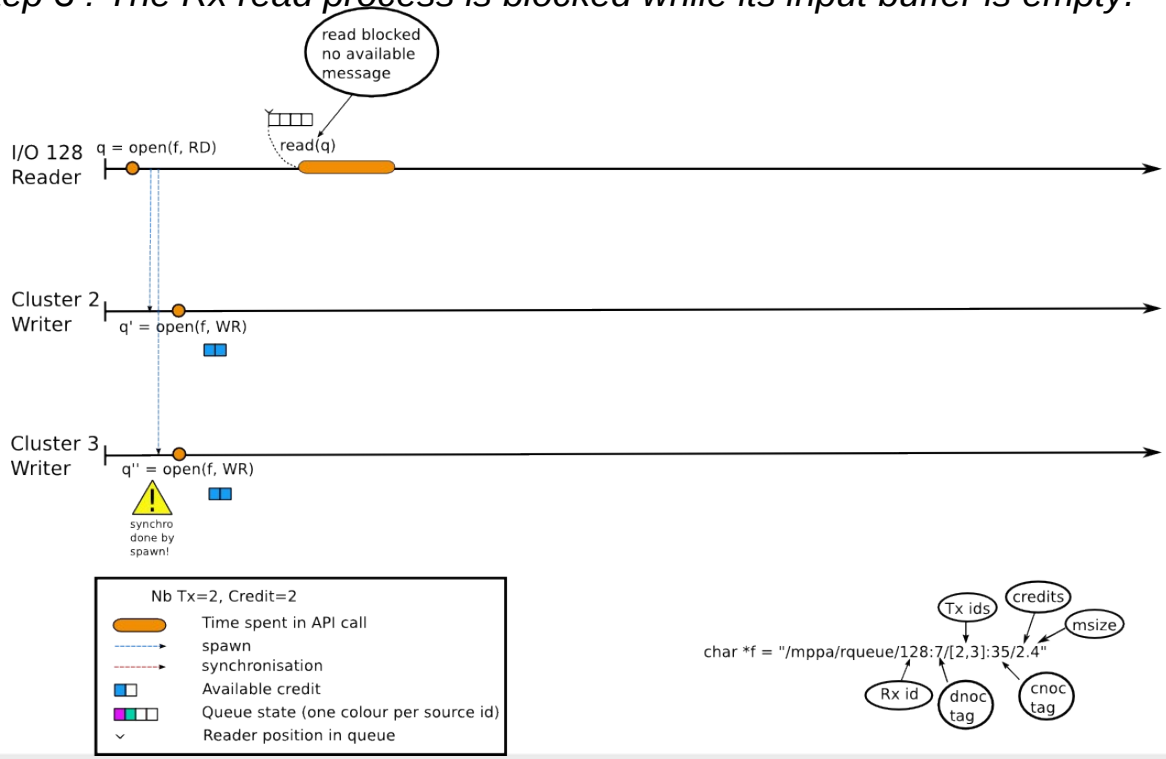
step 2 : The Cluster_2&3 start and open the Rqueue connector.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

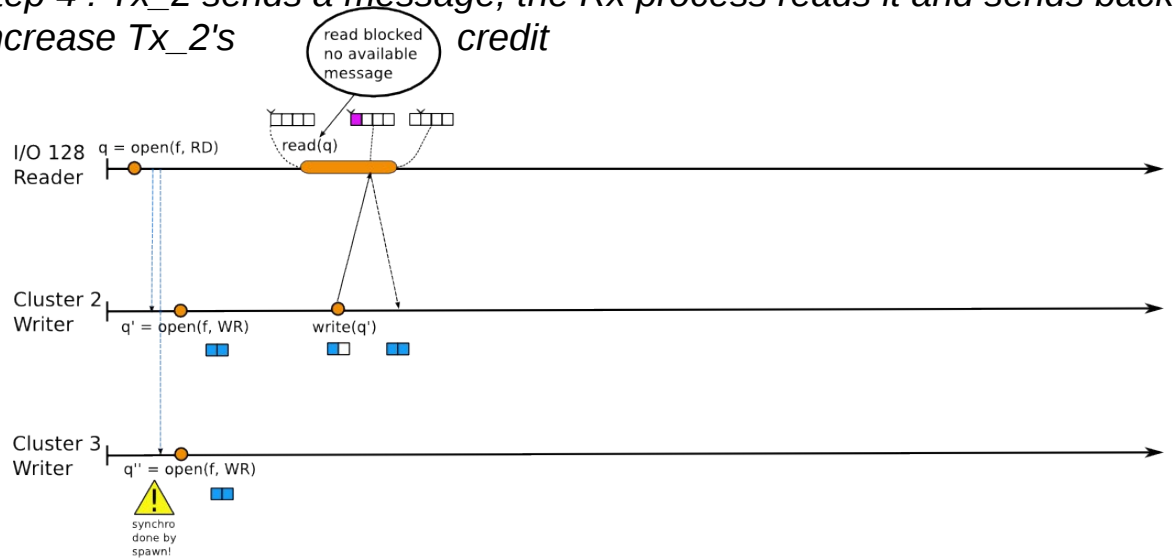
step 3 : The Rx read process is blocked while its input buffer is empty.

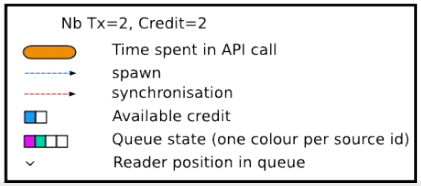


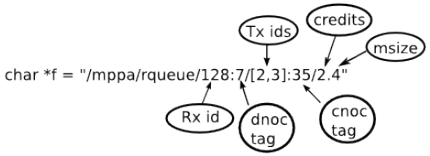
Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

step 4 : Tx_2 sends a message, the Rx process reads it and sends back a notification to increase Tx_2's credit



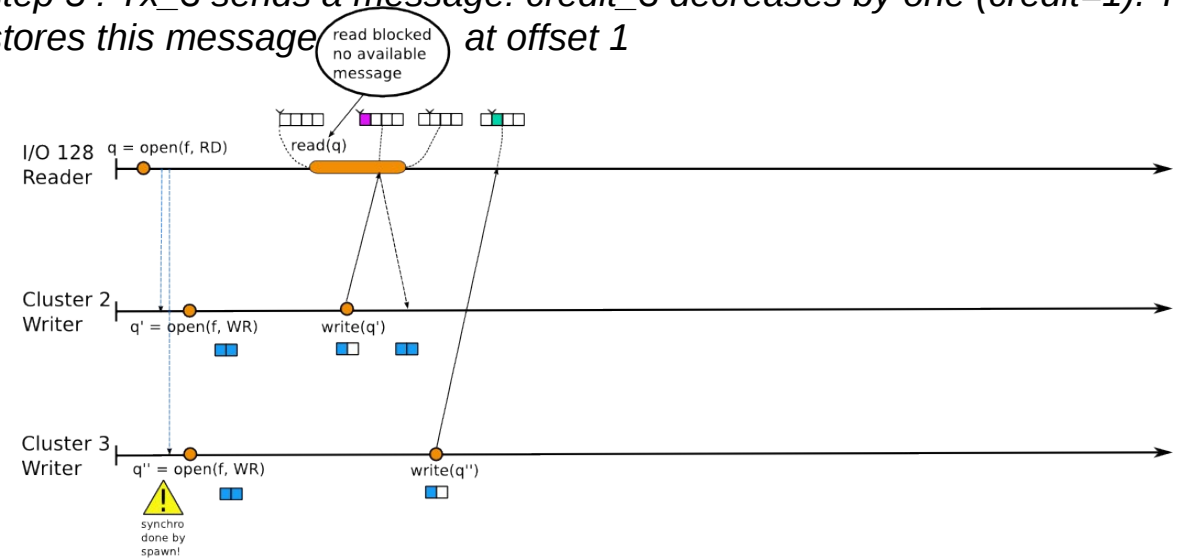




Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

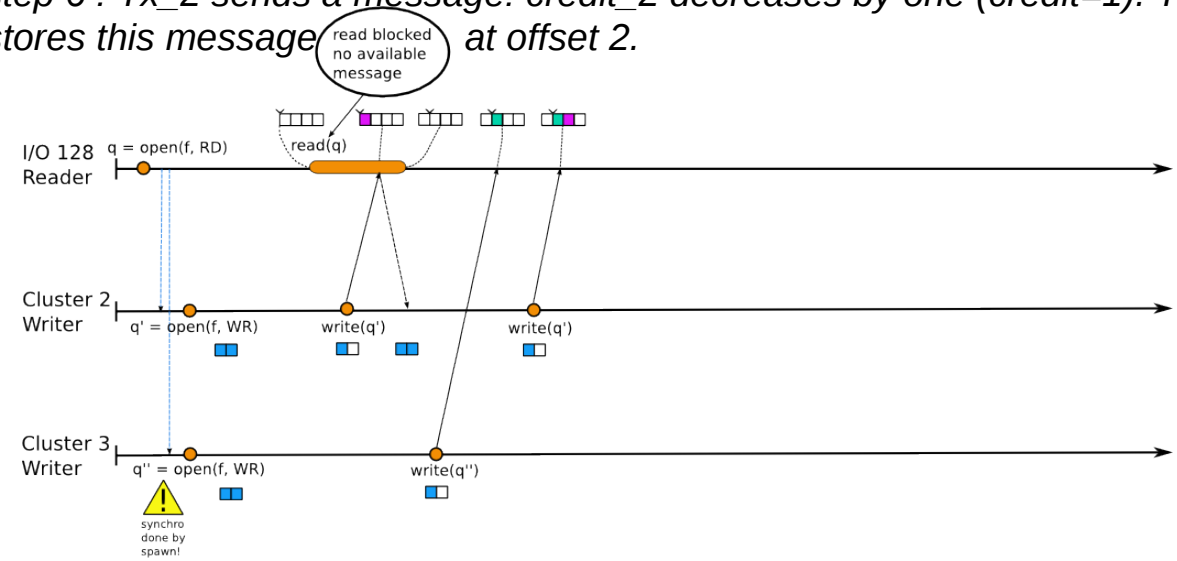
step 5 : Tx_3 sends a message: credit_3 decreases by one (credit=1). The Rx_input buffer stores this message at offset 1



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

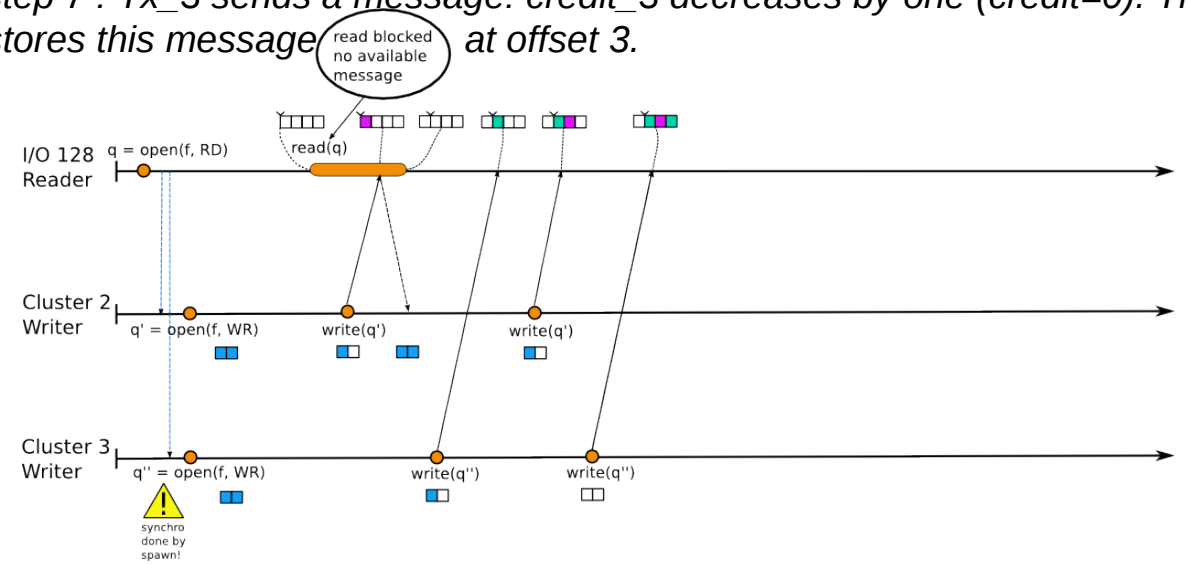
step 6 : Tx_2 sends a message: credit_2 decreases by one (credit=1). The Rx_input buffer stores this message at offset 2.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

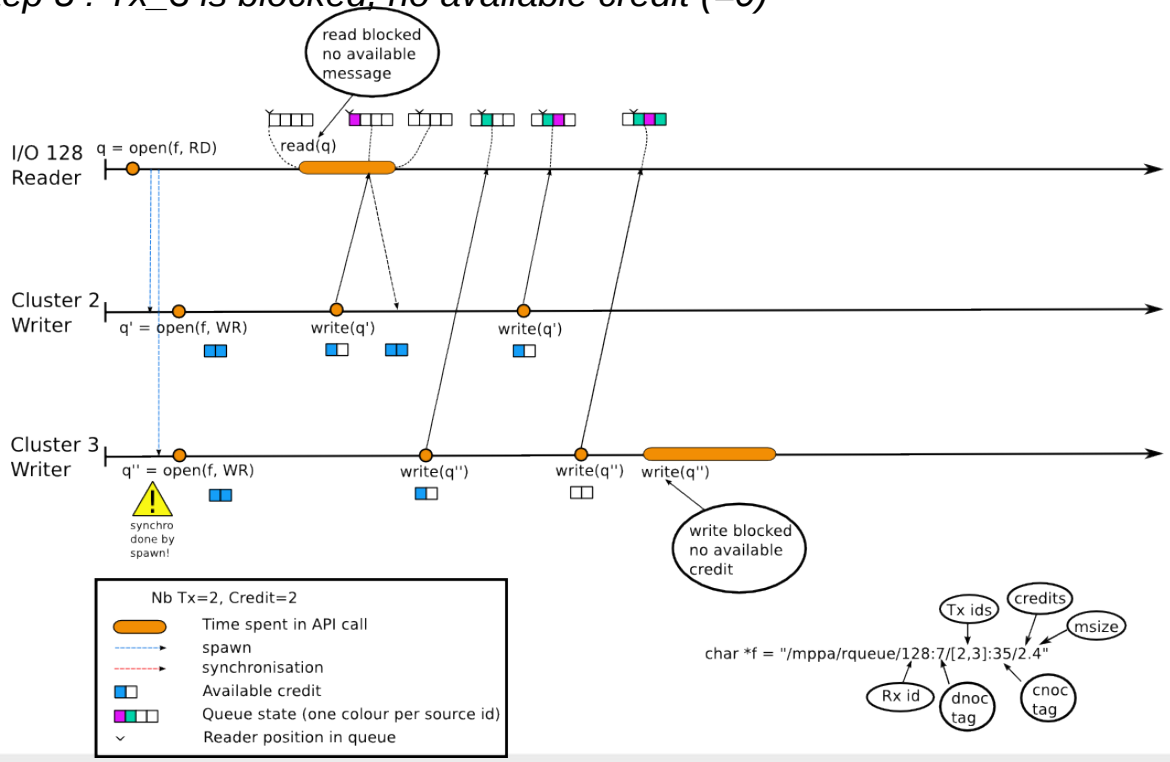
step 7 : Tx_3 sends a message: credit_3 decreases by one (credit=0). The Rx_input buffer stores this message at offset 3.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

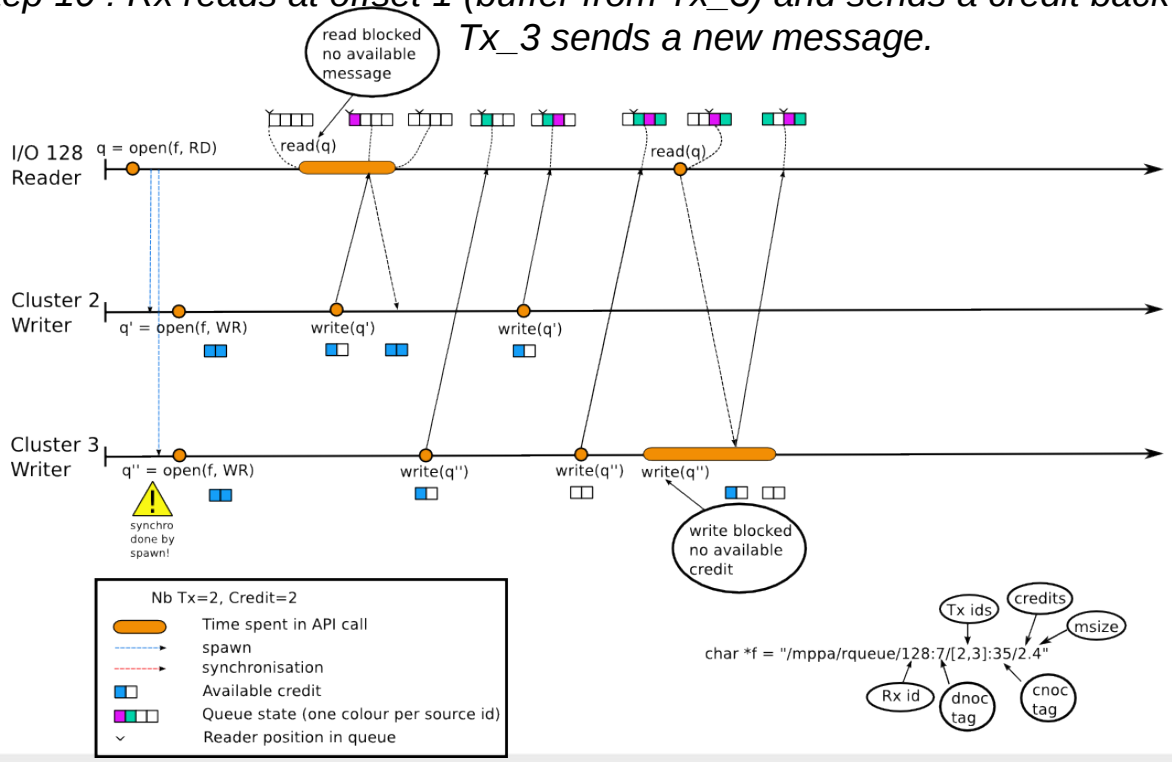
step 8 : Tx_3 is blocked, no available credit (=0)



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

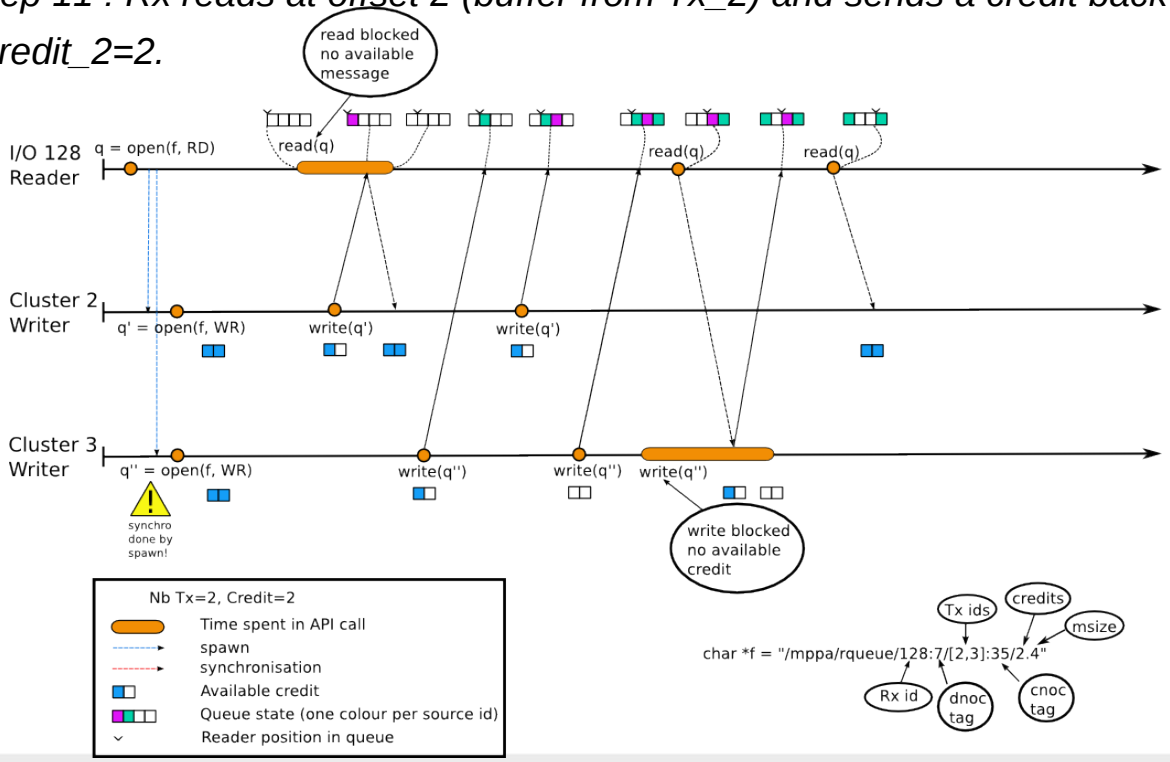
step 10 : Rx reads at offset 1 (buffer from Tx_3) and sends a credit back to Tx_3. Tx_3 sends a new message.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

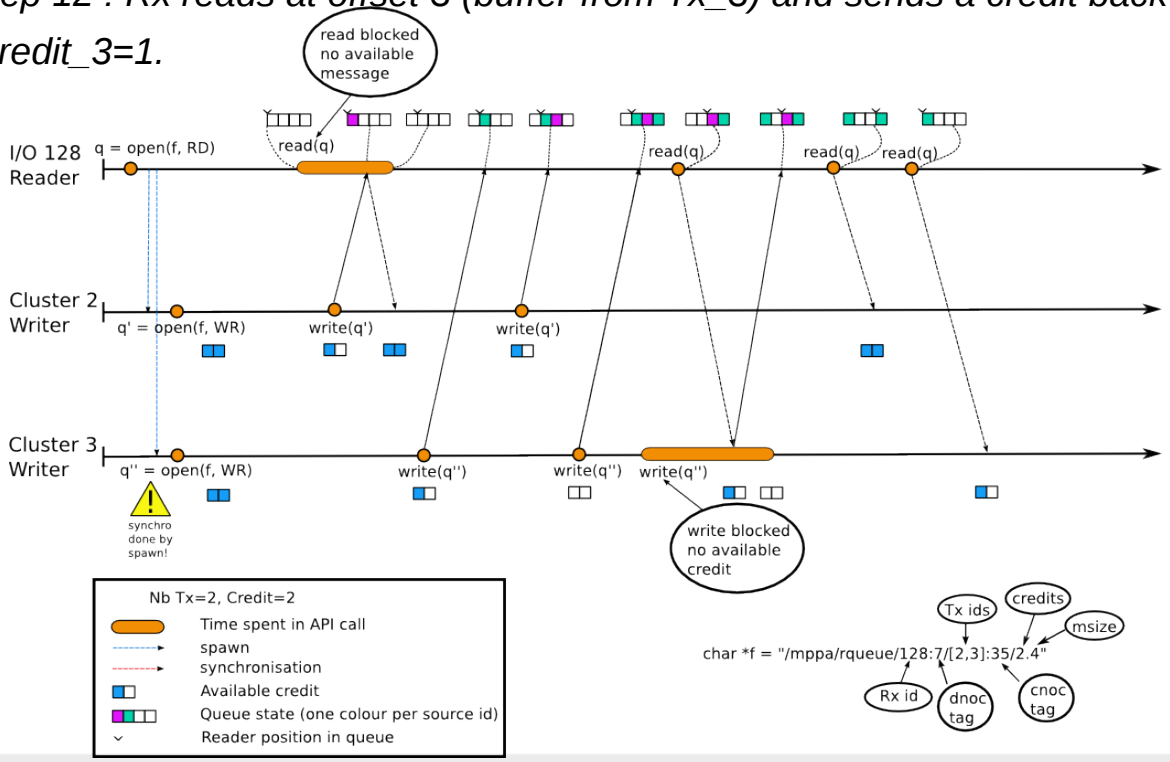
step 11 : Rx reads at offset 2 (buffer from Tx_2) and sends a credit back to Tx_2. Credit_2=2.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

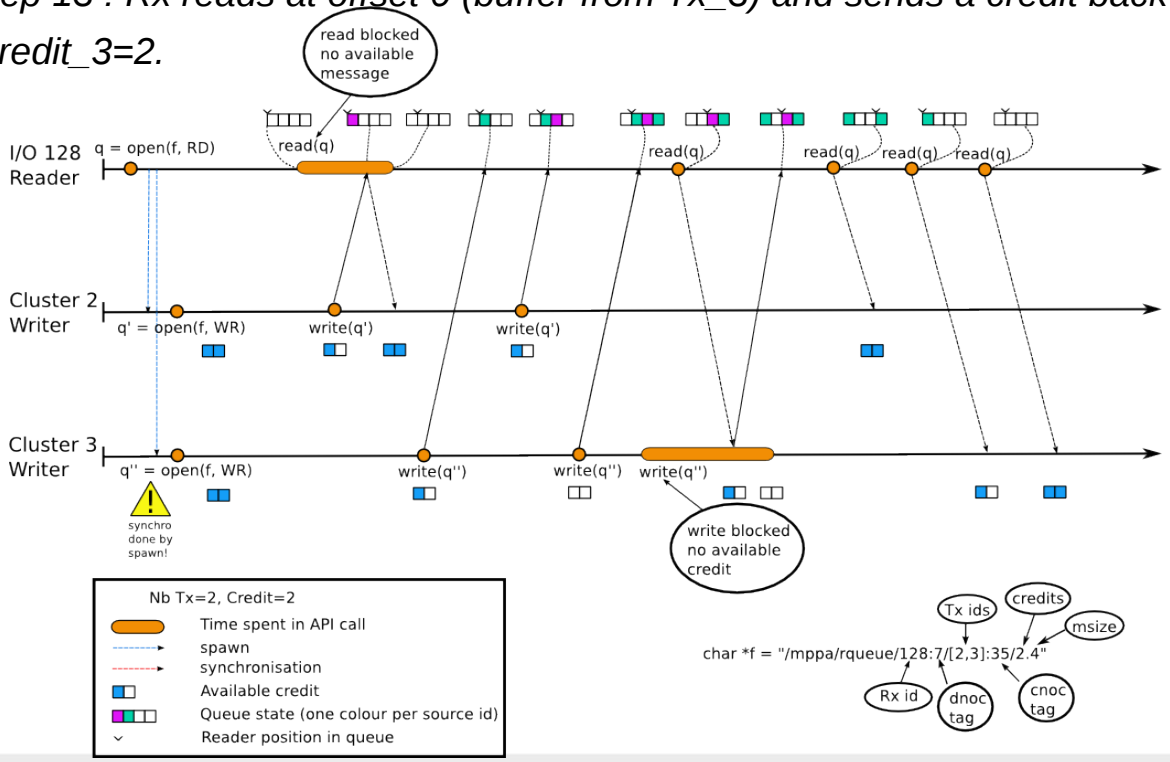
step 12 : Rx reads at offset 3 (buffer from Tx_3) and sends a credit back to Tx_3. Credit_3=1.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Rqueue Transfers

step 13 : Rx reads at offset 0 (buffer from Tx_3) and sends a credit back to Tx_3. Credit_3=2.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Portal Transfers

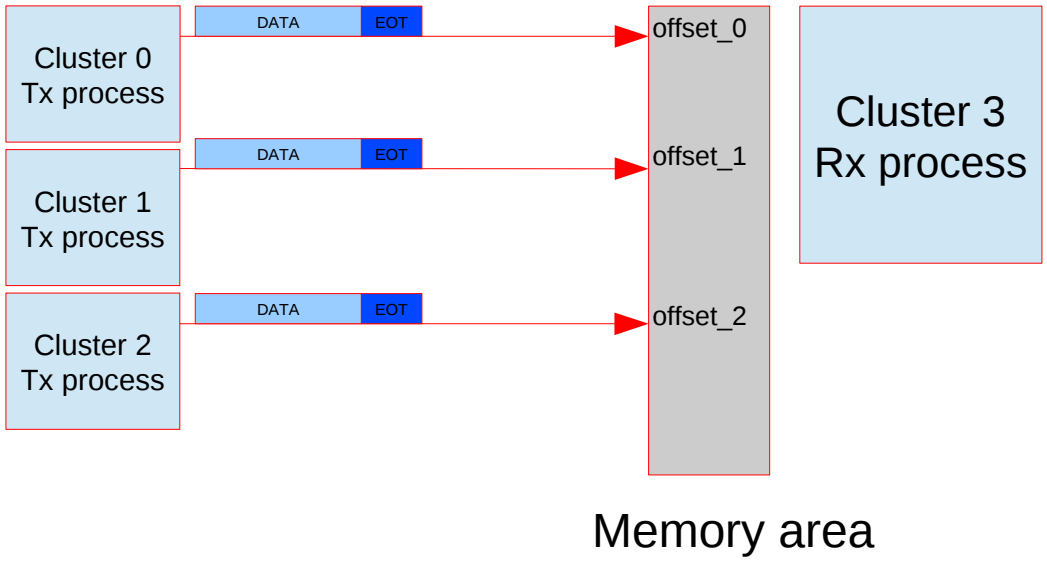
Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Portal Transfers

Portals occupy a memory area of the Rx process into which N Tx processes can write data at an arbitrary offset.

The Rx process must use asynchronous I/O operations.

The Portal Transfers are used for example to send result data that have been computed by N clusters.

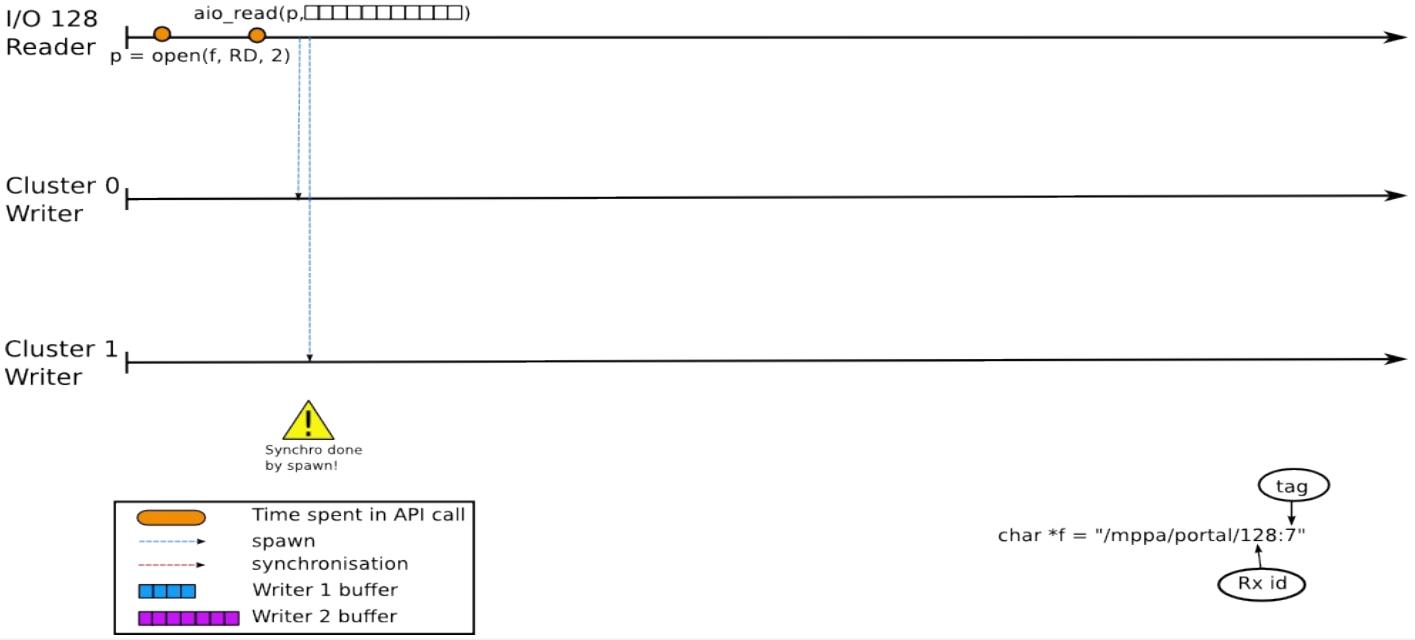


Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Portal Transfers

    step 1 : The IO_DDR opens a Portal connector for the Rx Process, and spawns the «Portal-slave1&2.bin » process to cluster_0 and cluster_1.

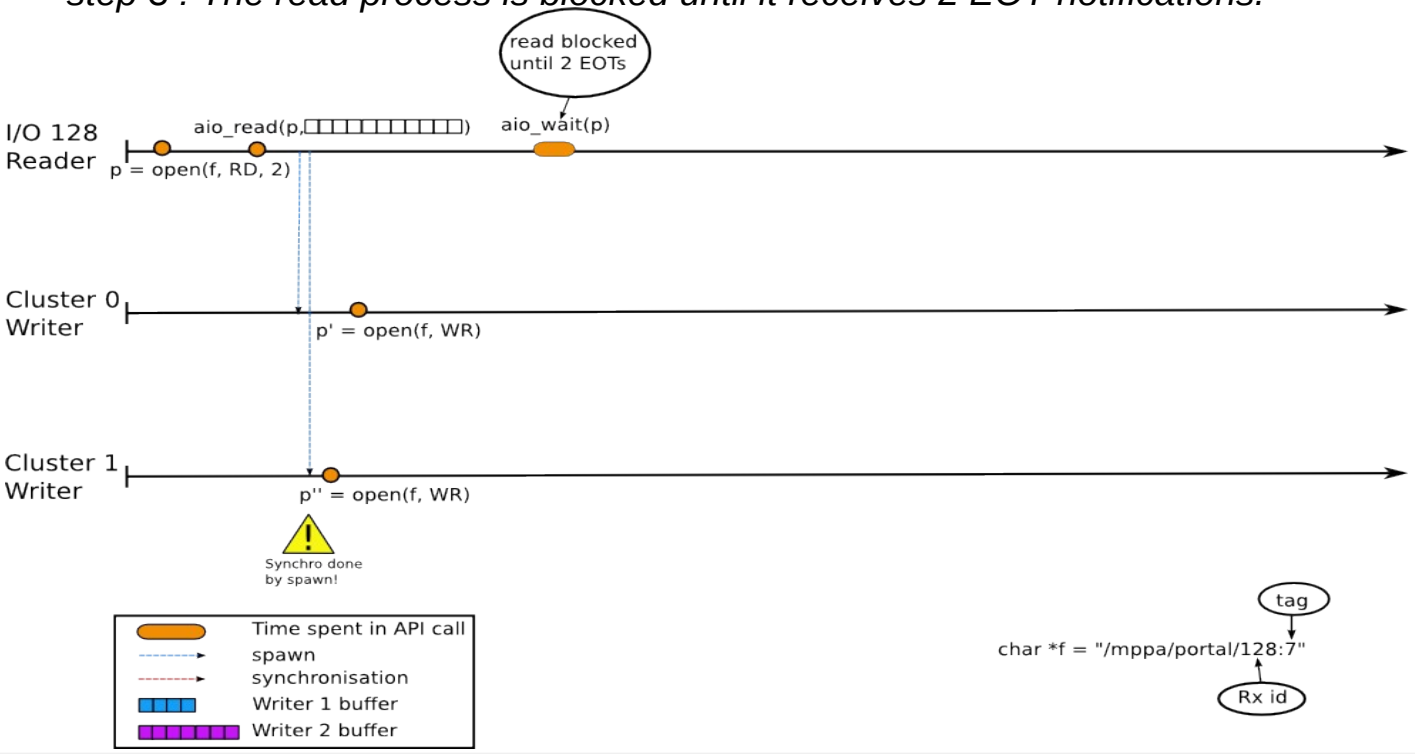
    step 2 : The Cluster_2&3 start and open their Portal connectors.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Portal Transfers

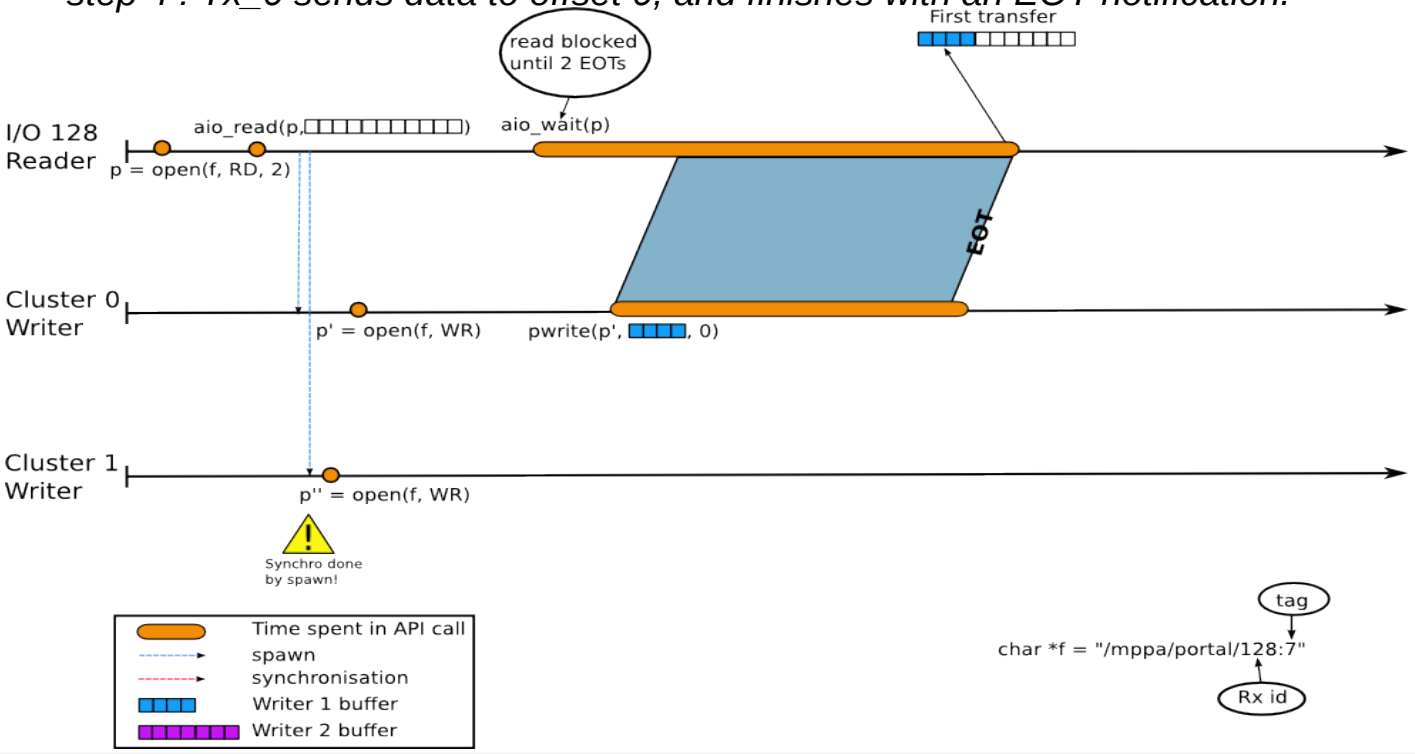
step 3 : The read process is blocked until it receives 2 EOT notifications.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Portal Transfers

step 4 : Tx_0 sends data to offset 0, and finishes with an EOT notification.

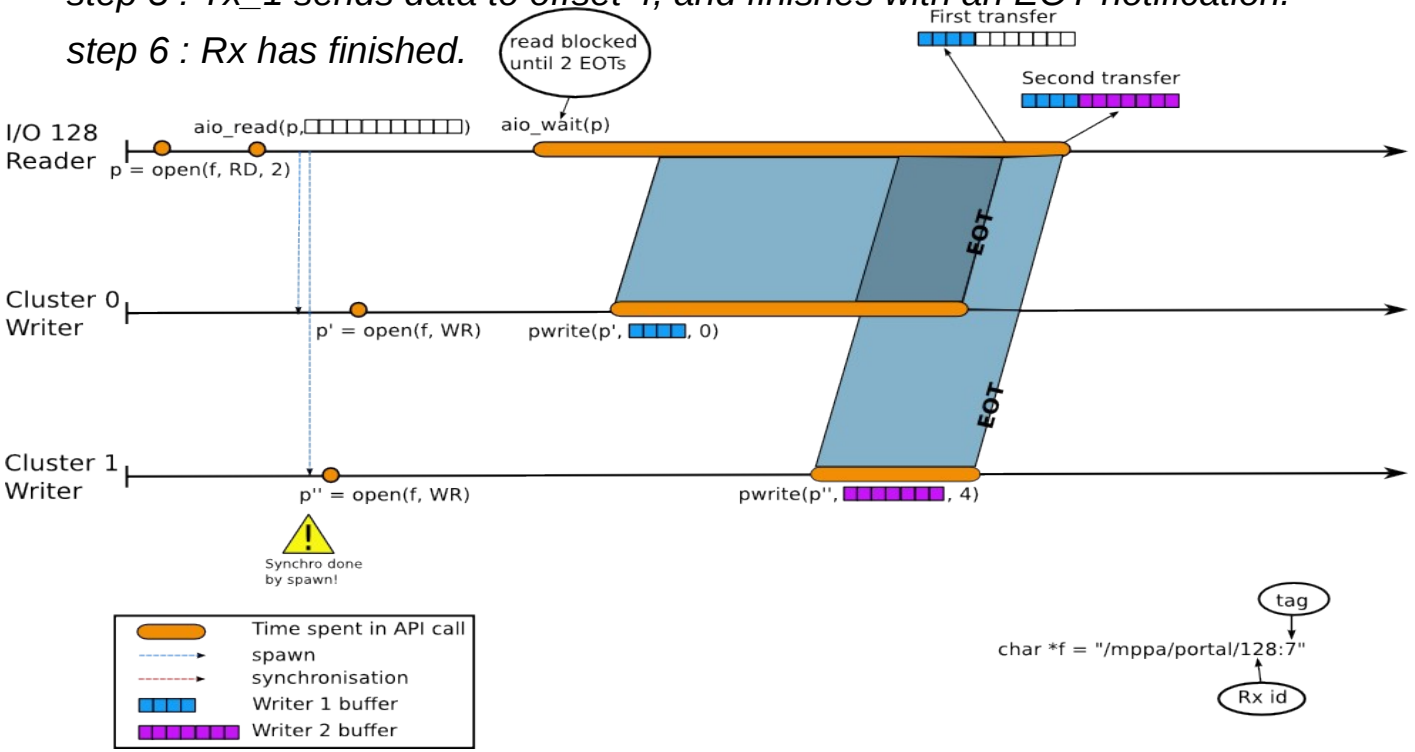


Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Portal Transfers

step 5 : Tx_1 sends data to offset 4, and finishes with an EOT notification.

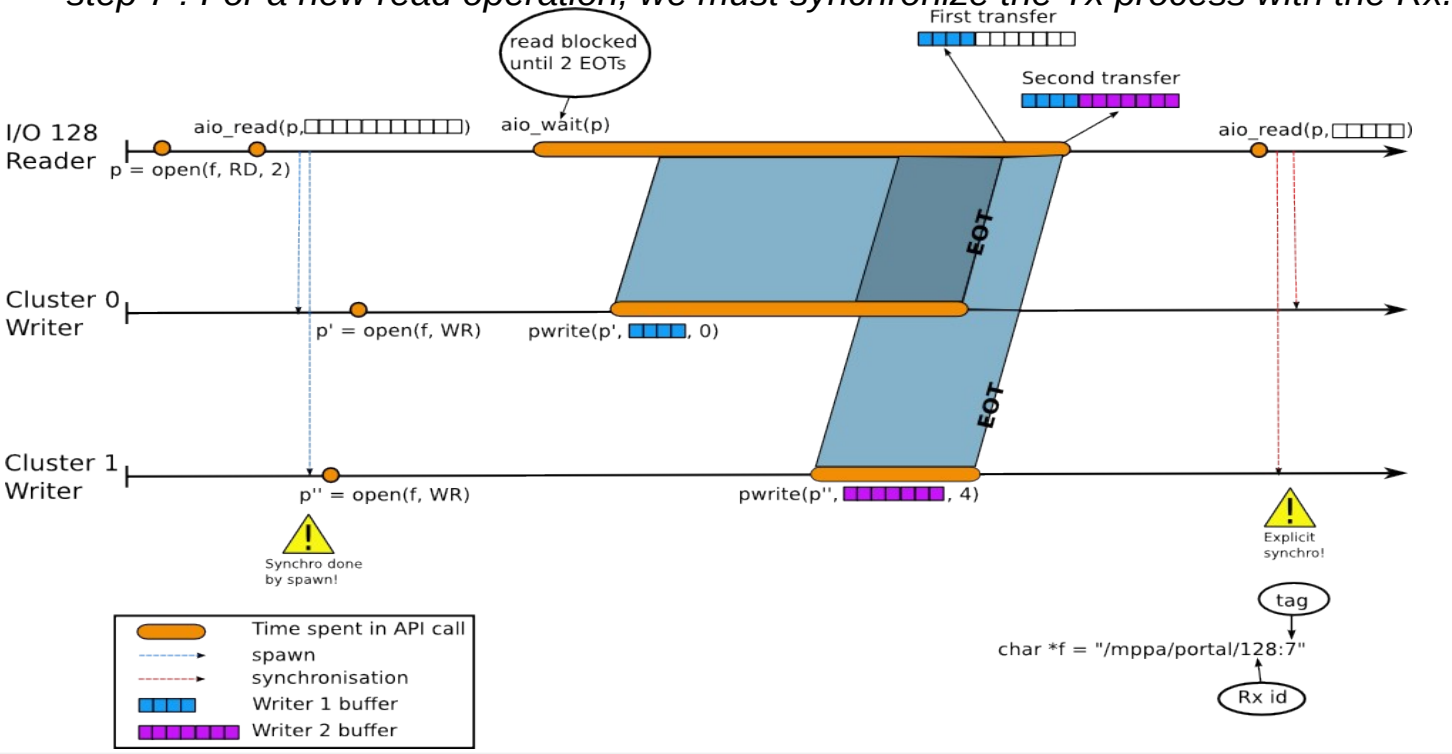
step 6 : Rx has finished.



Inter-Cluster Communication presentation

- MPPA IPC Transfer types :
  - Portal Transfers

step 7 : For a new read operation, we must synchronize the Tx process with the Rx.



Inter-Cluster Communication presentation

- MPPA IPC synchronisation type :
  - Sync

Inter-Cluster Communication presentation

- MPPA IPC synchronisation type :
  - Sync

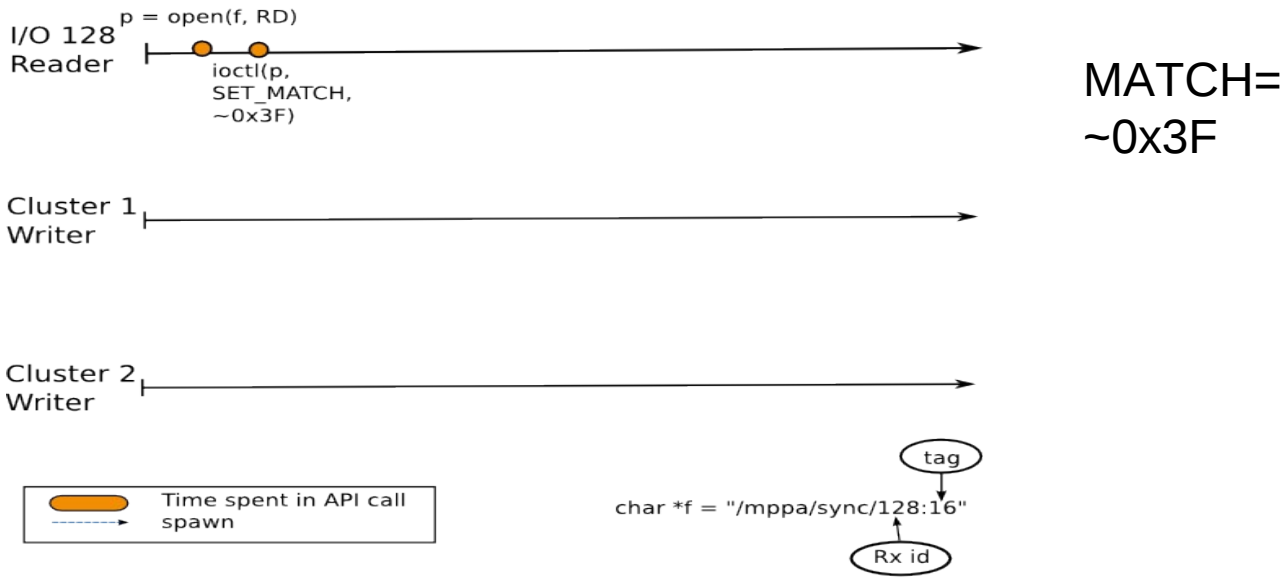
    The Sync connector is used for example to synchronize processes

Inter-Cluster Communication presentation

- MPPA IPC synchronisation type :
  - Sync

step 1 : The IO_DDR opens a sync connector for the Rx Process.

step 2 : The IO_DDR sets the 'match' value for the Rx Process.



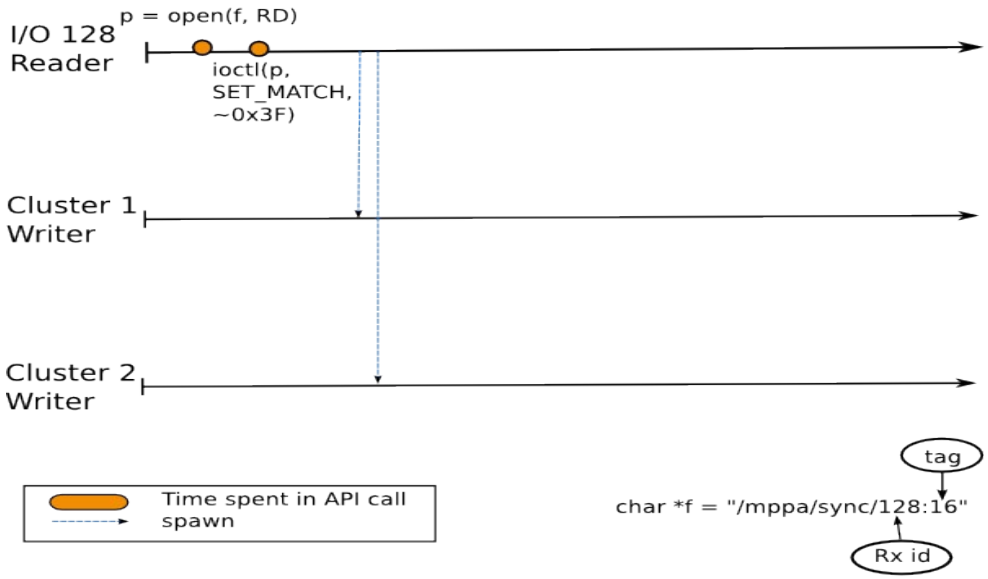
MATCH=
~0x3F

Inter-Cluster Communication presentation

- MPPA IPC synchronisation type :
  - Sync

step 3 : The IO_DDR spawns the «Portal-slave1&2.bin » process to cluster_1 and cluster_2.

Step 4 : The Cluster_1&2 start and open their sync connectors.



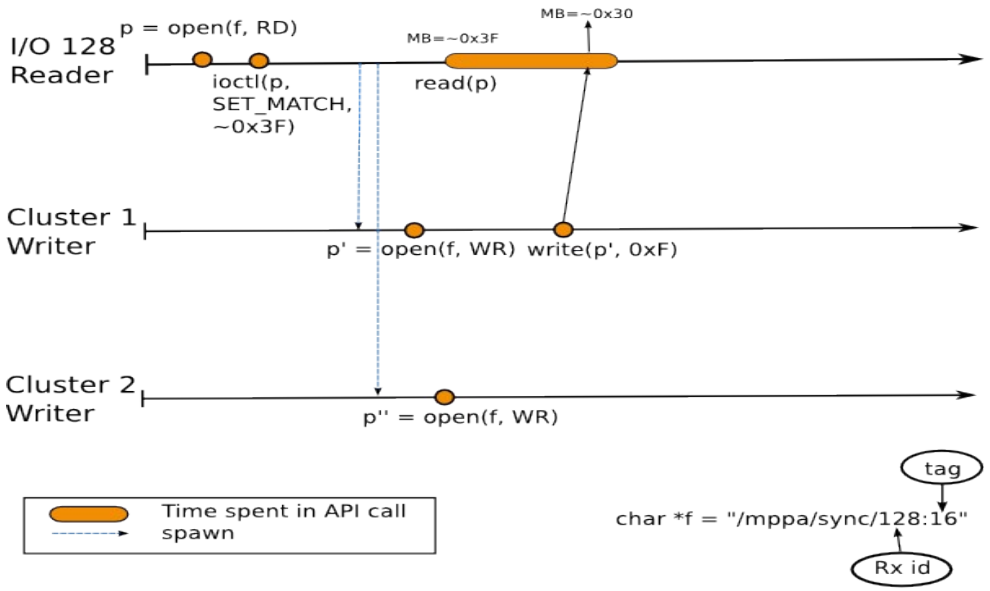
MATCH= ~0x3F

Inter-Cluster Communication presentation

- MPPA IPC synchronisation type :
  - Sync

step 5 : The Rx process is blocked until the result of the 'match' value equals ~0x00.

step 6 : Tx_1 sends '0x0F' : the match value is now equal to ~0x30



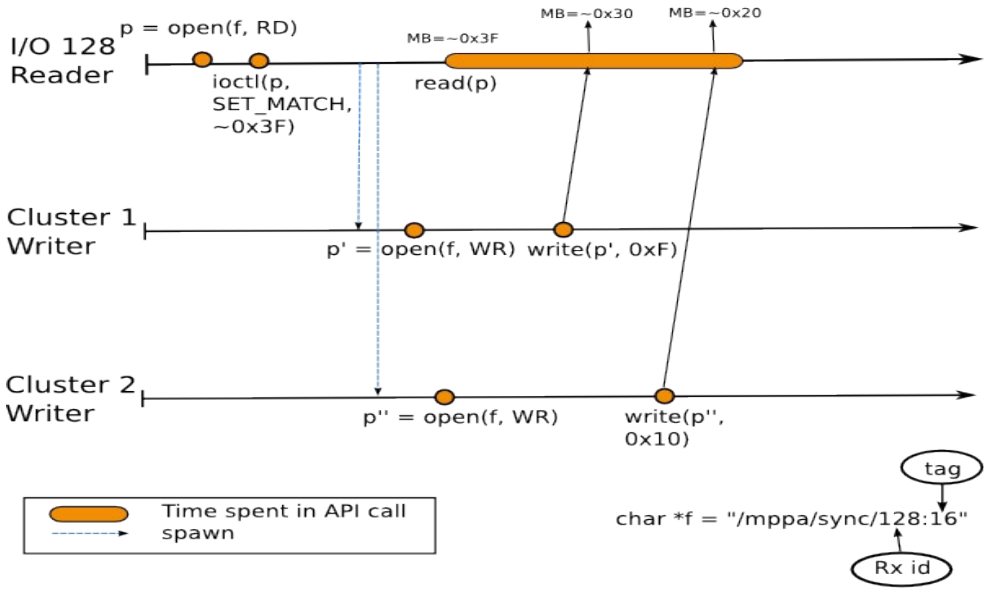
MATCH=

~0x3F	11000000
+0x0F	00001111
~0x30	11001111

Inter-Cluster Communication presentation

- MPPA IPC synchronisation type :
  - Sync

step 7 : Tx_2 sends '0x10' , the match value is now equal to ~0x20



MATCH=

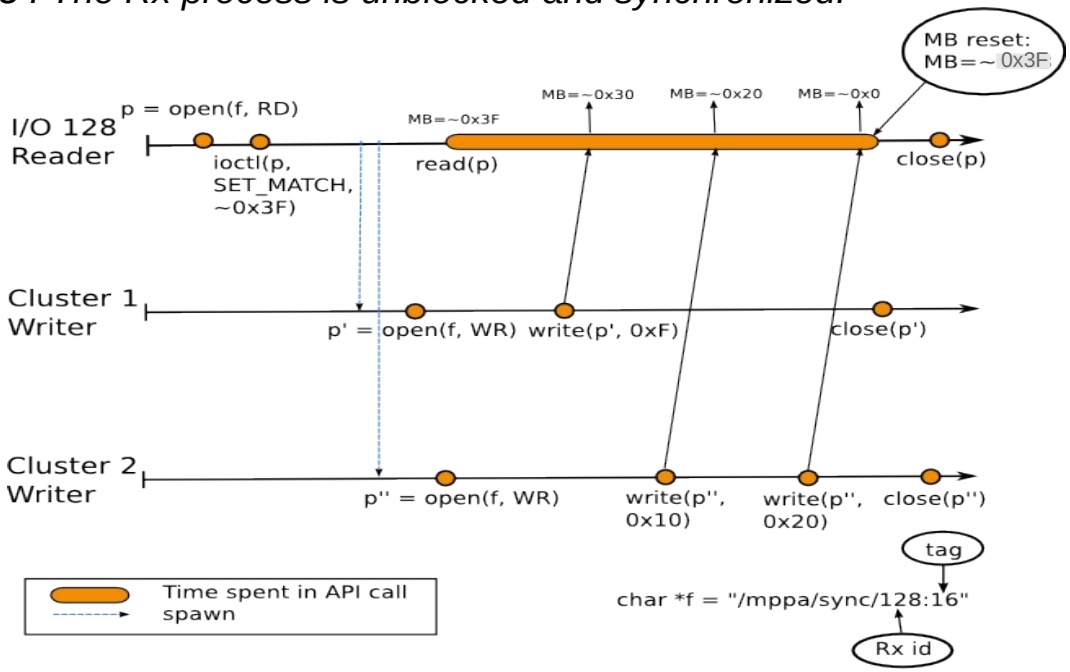
~0x30	11000000
+0x10	00010000
~0x20	11011111

Inter-Cluster Communication presentation

- MPPA IPC synchronisation type :
  - Sync

step 7 : Tx_2 sends '0x20' , the match value is now equal to ~0x00.

step 8 : The Rx process is unblocked and synchronized.



MATCH=

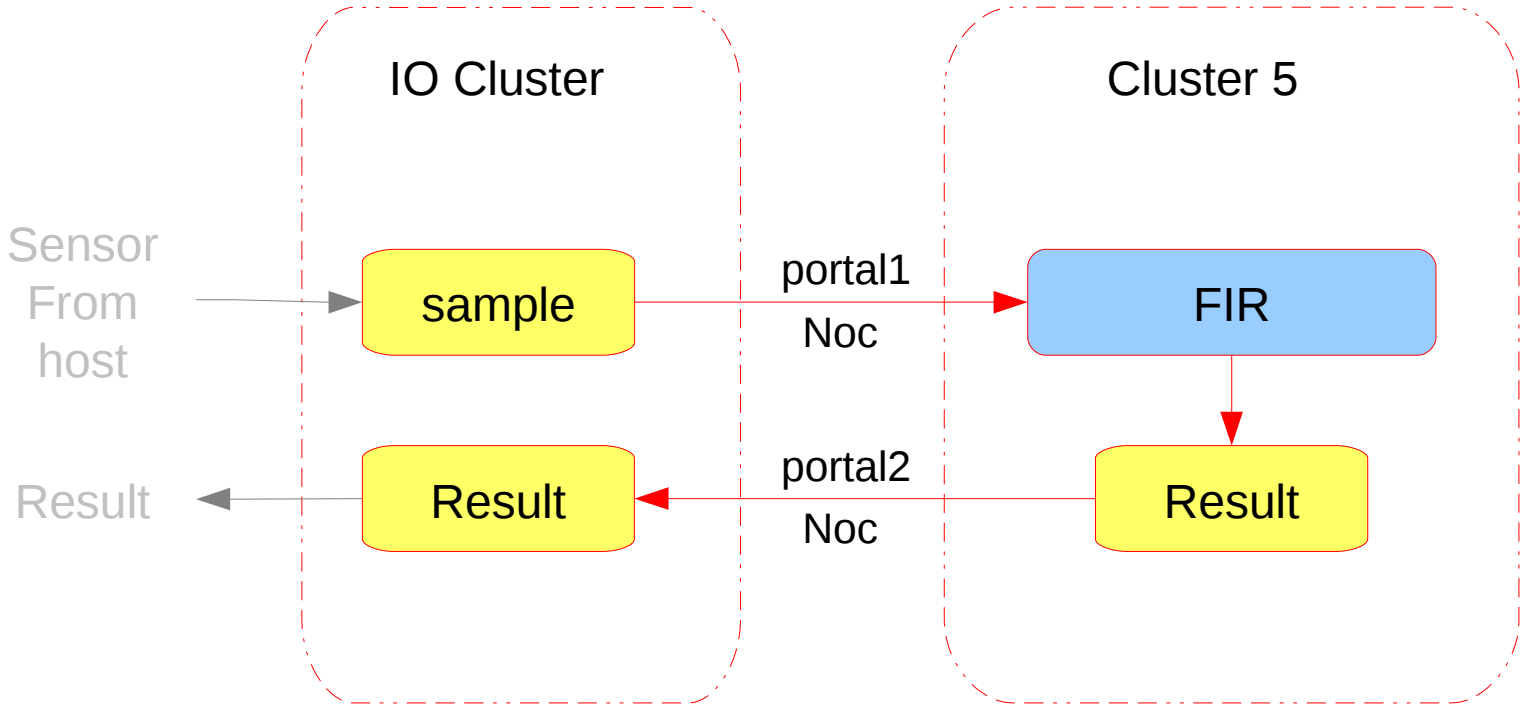
~0x20	00100000
+0x20	00100000
~0x00	11111111

Inter-Cluster Communication presentation

- More MPPA IPC details :
  - /usr/local/k1tools/doc/Specification/Process/**Process.pdf**
- 4 examples :
  - /Use_Cases/Portal
  - /Use_Cases/Channel
  - /Use_Cases/Rqueue
  - /Use_Cases/Sync

Inter-Cluster Communication presentation

- Portal streaming example for a low-pass filter application



Inter-Cluster Communication presentation

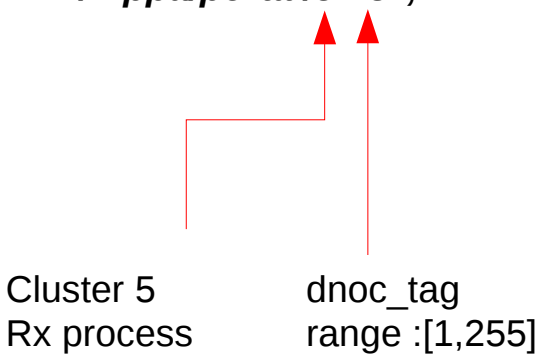
- Portal streaming example for a low-pass filter application
  - NoC Descriptor
  - Rx File Operations
  - Tx File Operations

Inter-Cluster Communication presentation

- Portal streaming example for a low-pass filter application
  - NoC Descriptor

Inter-Cluster Communication presentation

- Portal streaming example for a low-pass filter application
  - NoC Descriptor
  - Channel Pathname :
    - ***/mppa/portal/rx_node:dnoc tag***
    - ***Example :***
      - ***char *portal1= "/mppa/portal/5:18";***



Inter-Cluster Communication presentation

- Portal streaming example for a low-pass filter application

  - Rx File Operations
    - **mppa open(pathname, O RDONLY)** Open the Portal connector for the Rx process.
    - **mppa aio read(portal_aiocb)** Bind the D-NoC buffer on the Rx process to the data specified by the struct mppa_aiocb *portal_aiocb.

      Note: The macro MPPA_AIOCB_INITIALIZER and the function mppa_aiocb_ctor() both set the trigger value to 1.
    - **mppa aio return(portal_aiocb)** Complete the asynchronous read, and return the size initially set in portal_aiocb.

      Note: when a notify function is attached to the portal_aiocb, a call to this function is not allowed.
    - **mppa aio wait(portal_aiocb**) Block until the notification counter is greater than or equal to the trigger, then complete the asynchronous read like mppa_aio_return().
    - **mppa aio rearm(portal_aiocb)** Wait for the end of the associated asynchronous request to the mppa_aiocb and re-configure the same asynchronous request. Its behavior is the same as calling mppa_aio_wait() followed by calling mppa_aio_read().
    - **mppa close(portal fd)** Release the Rx process NoC resources.

See documentation : /usr/local/k1tools/doc/Specifications/Process/Process.pdf

Inter-Cluster Communication presentation

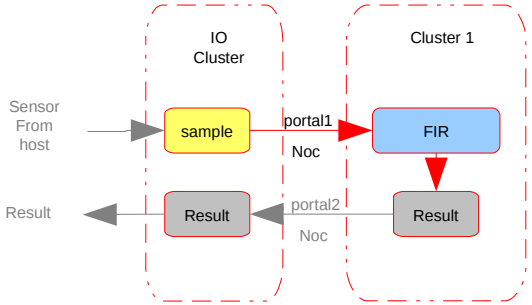
- Portal streaming example for a low-pass filter application

  - Tx File Operations
    - **mppa open(pathname, O WRONLY)** Open the Portal connector for a Tx process.
    - **mppa pwrite(portal fd, base, size, offset)** Remote write of mppa_size_t size bytes from the void *base address in the Tx process, to the Rx process address space at an offset of mppa_off_t (in bytes). This operation carries a notification on the last packet, unless notifications are disabled by mppa_ioctl() with request MPPA_TX_NOTIFY_OFF. It is an unchecked error to write past the end of the address space opened by the Rx process.
    - **mppa aio write(portal aiocb)** Bind a D-NoC micro-core engine to asynchronously transfer data specified within the aiocb. At most one call to mppa_aio_write() on a given file descriptor may be pending at any time
    - **mppa aio return(portal aiocb)** Complete the asynchronous request, and return the count of bytes transferred for an asynchronous write. In case of error, return -1.
    - **mppa aio wait(portal aiocb)** Executed on an asynchronous write request, block until the micro-core engine notifies that remote write has been sent. This function must not be called in a SIGEV_CALLBACK notify function.
    - **mppa close(portal fd)** Release the Tx process NoC resources.

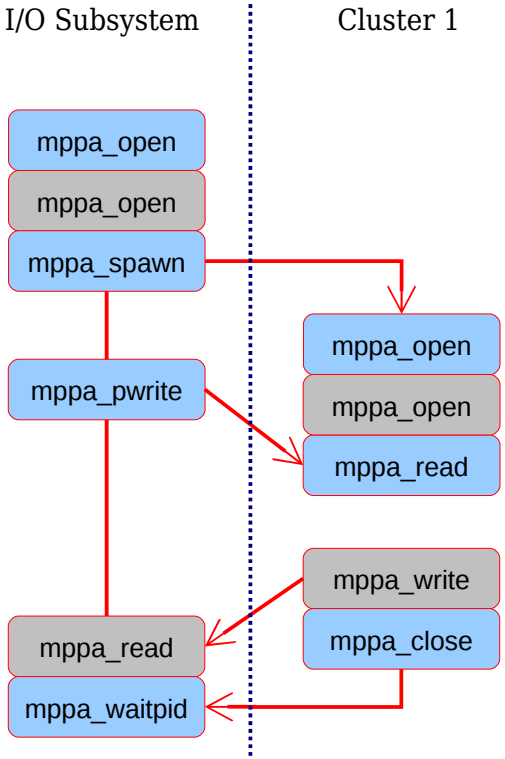
See documentation : /usr/local/k1tools/doc/Specifications/Process/Process.pdf

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- **STEP 1 :** Send 'Sample' from IO DDR to Cluster 1



I/O Subsystem :

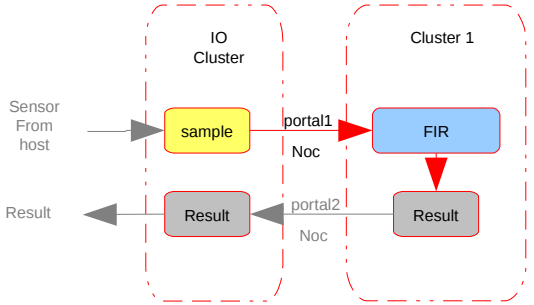
mppa_open : configure Portal 1 connector (WR)
mppa_spawn : transfer cluster 1 binary
mppa_pwrite : send 'Sample' to cluster 1 via the Portal 1 connector
mppa_waitpid : wait for the end of the cluster 1 process.

Cluster 1 :

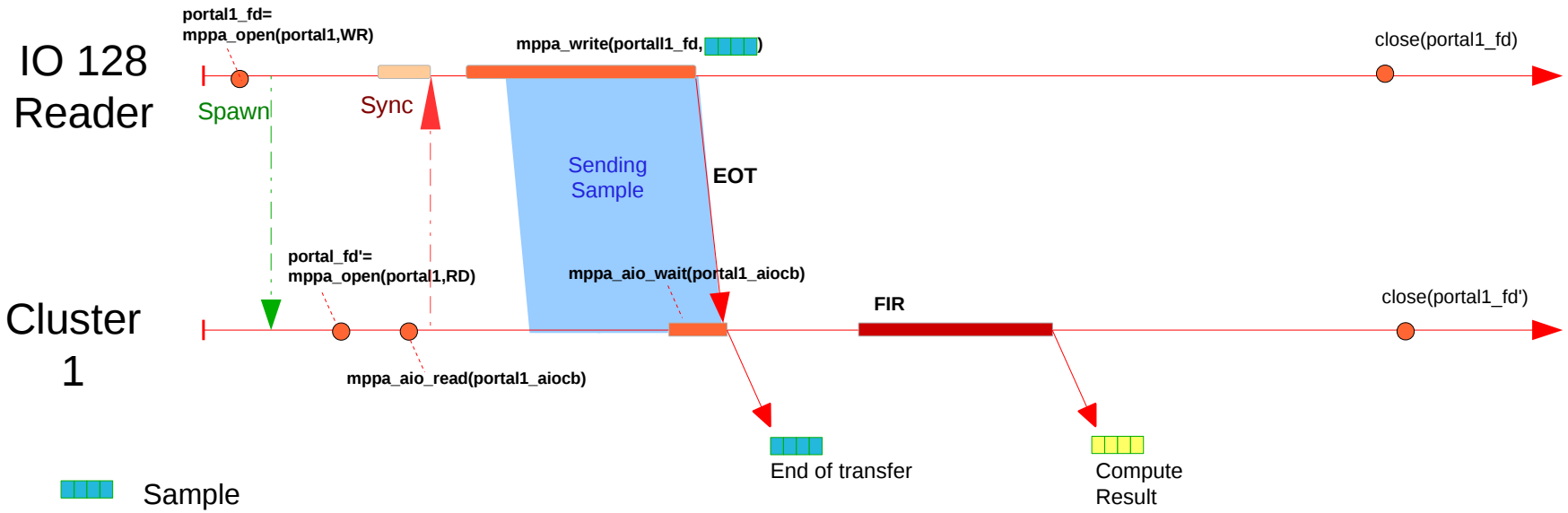
mppa_open : configure Portal 1 connector (RX)
mppa_read :receive 'Sample' from IO DDR via the Portal 1 connector (in fact aio_read + ai_wait)
mppa_close : end cluster 1 process.

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- STEP 1 : Send 'Sample' from IO DDR to Cluster 1

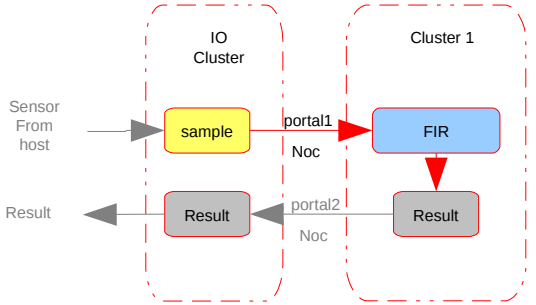


portal1= "/mppa/portal/128:2";

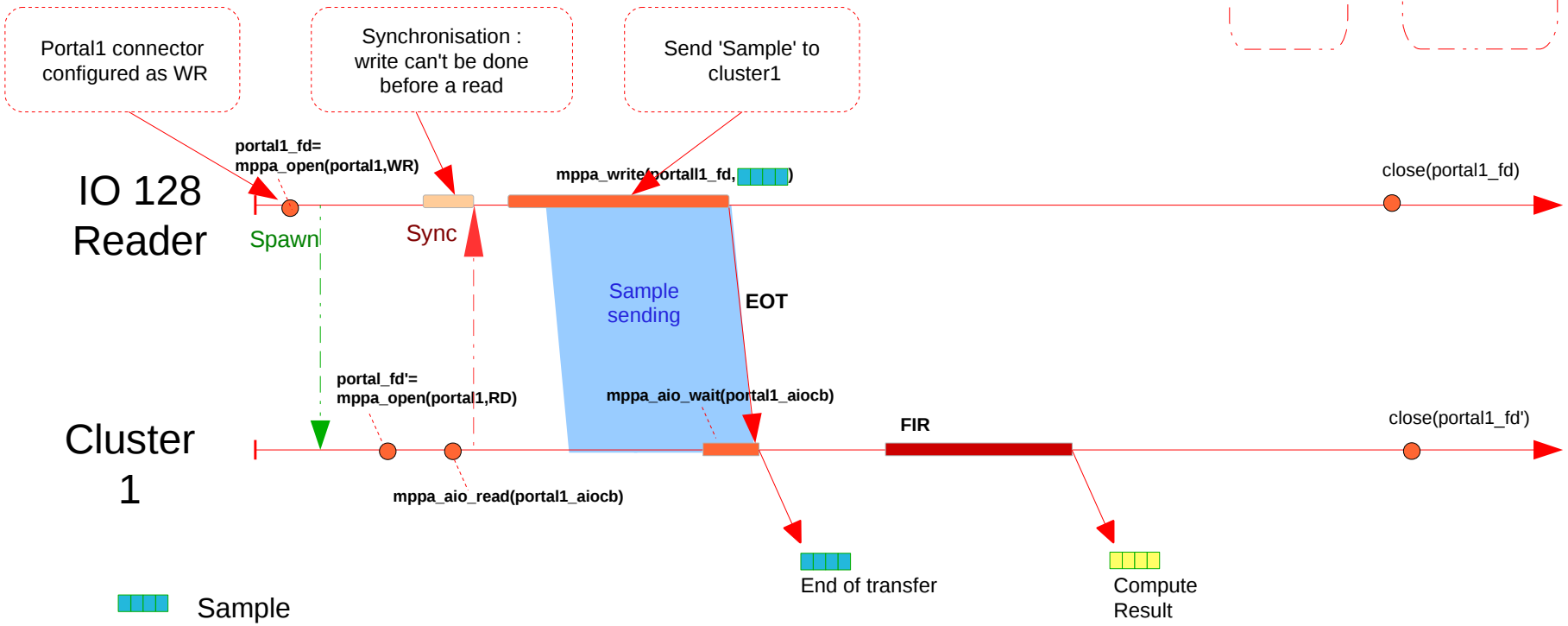
mppa_aiocb_t portal1_aiocb // srtuct with portal_fd, sizeof data ; and address of data.

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- STEP 1 : Send 'Sample' from IO DDR to Cluster 1

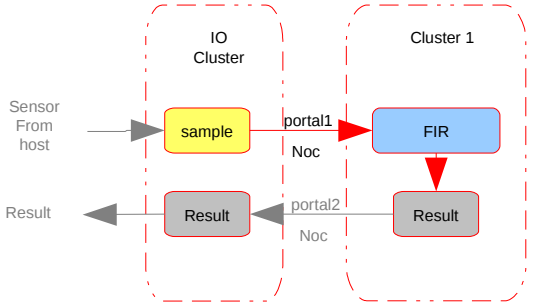


portal1= "/mppa/portal/128:2";

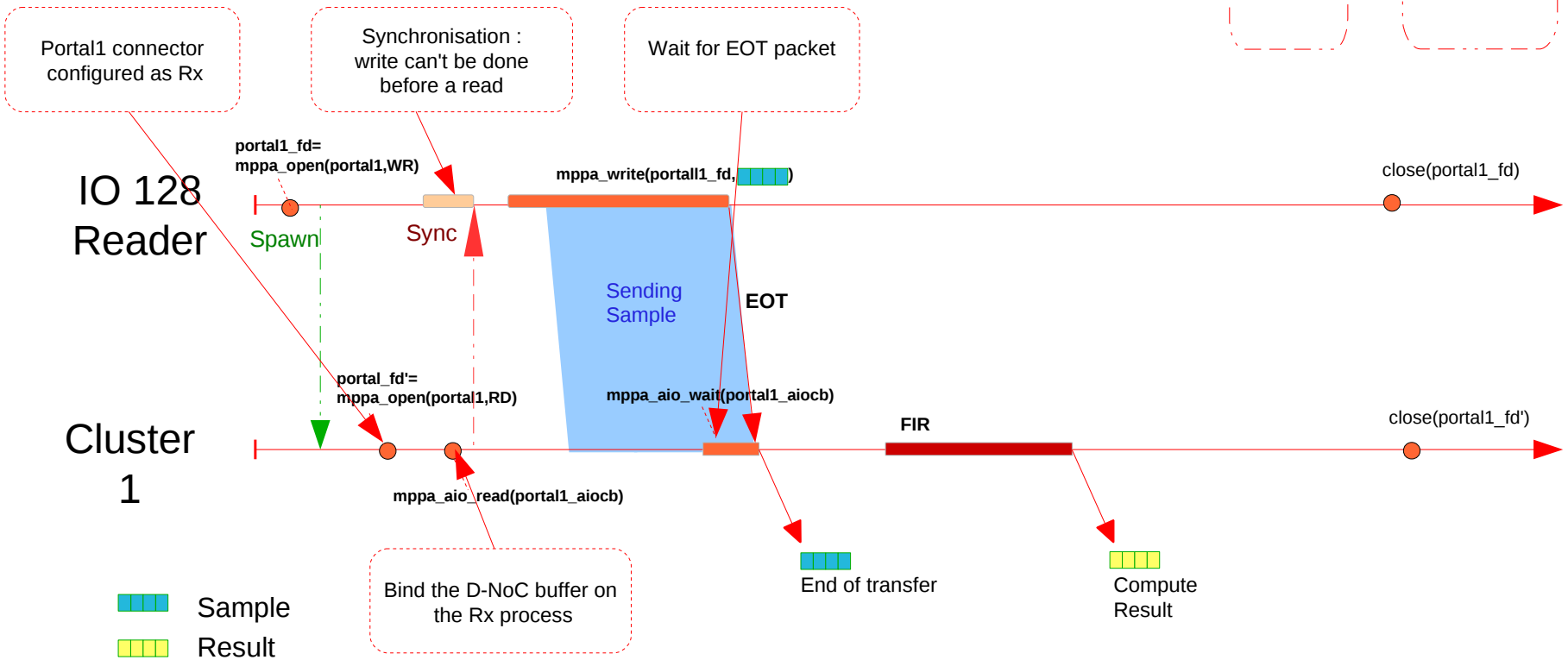
mppa_aiocb_t portal1_aiocb // srtuct with portal_fd, sizeof data ; and address of data.

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- **STEP 1 :** Send 'Sample' from IO DDR to Cluster 1

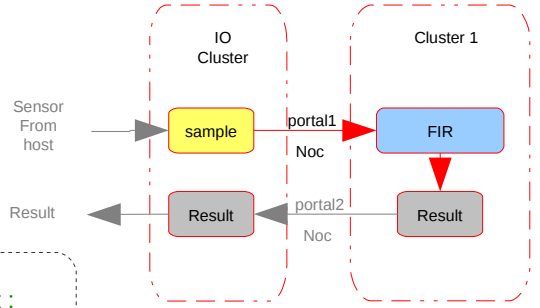


portal1= "/mppa/portal/128:2";

mppa_aiocb_t portal1_aiocb // srtuct with portal_fd, sizeof data ; and address of data.

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- **STEP 1 :** Send 'Sample' from IO DDR to Cluster 1

Add this part :
cluster_main.c:136

Add this part :
io_main.c:154

Add this part :
cluster_main.c:195

IO 128
Reader

portal1_fd=
mppa_open(portal1,WR)

mppa_write(portall1_fd,)

close(portal1_fd)

Spawn

Sync

Sending
Sample

EOT

Cluster
1

portal_fd'=
mppa_open(portal1,RD)

mppa_aio_wait(portal1_aiocb)

FIR

close(portal1_fd')

mppa_aio_read(portal1_aiocb)

Add this part :
cluster_main.c:166

End of transfer

Compute
Result

Sample

Result

Synchronisation

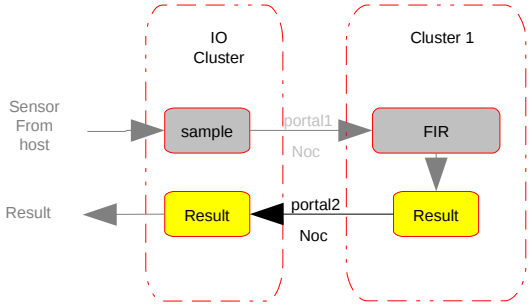
Spawn

portal1= "/mppa/portal/128:2";

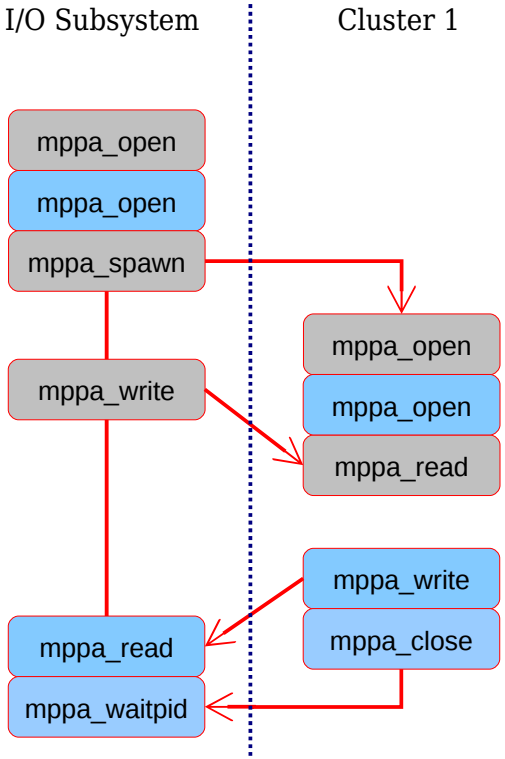
mppa_aiocb_t portal1_aiocb // srtuct with portal_fd, sizeof data ; and address of data.

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- STEP 2 : Send 'Result' from Cluster 1 to IO DDR



I/O Subsystem :

mppa_open : configure Portal 2 connector (RX)
mppa_read :receive 'Result' from cluster 1 via the Portal1 connector (in fact aio_read + ai_wait)
mppa_waitpid : wait for the end of the cluster 1 process.

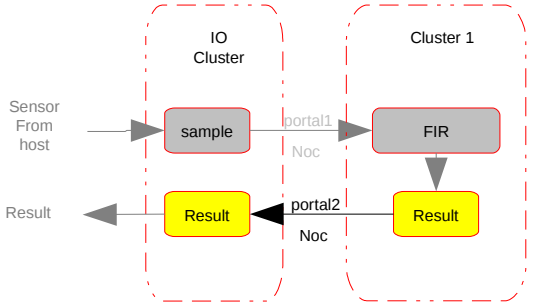
Cluster 1 :

mppa_open : configure Portal 2 connector (WR)
mppa_write : send 'Result' to IO DDR via the Portal2 connector

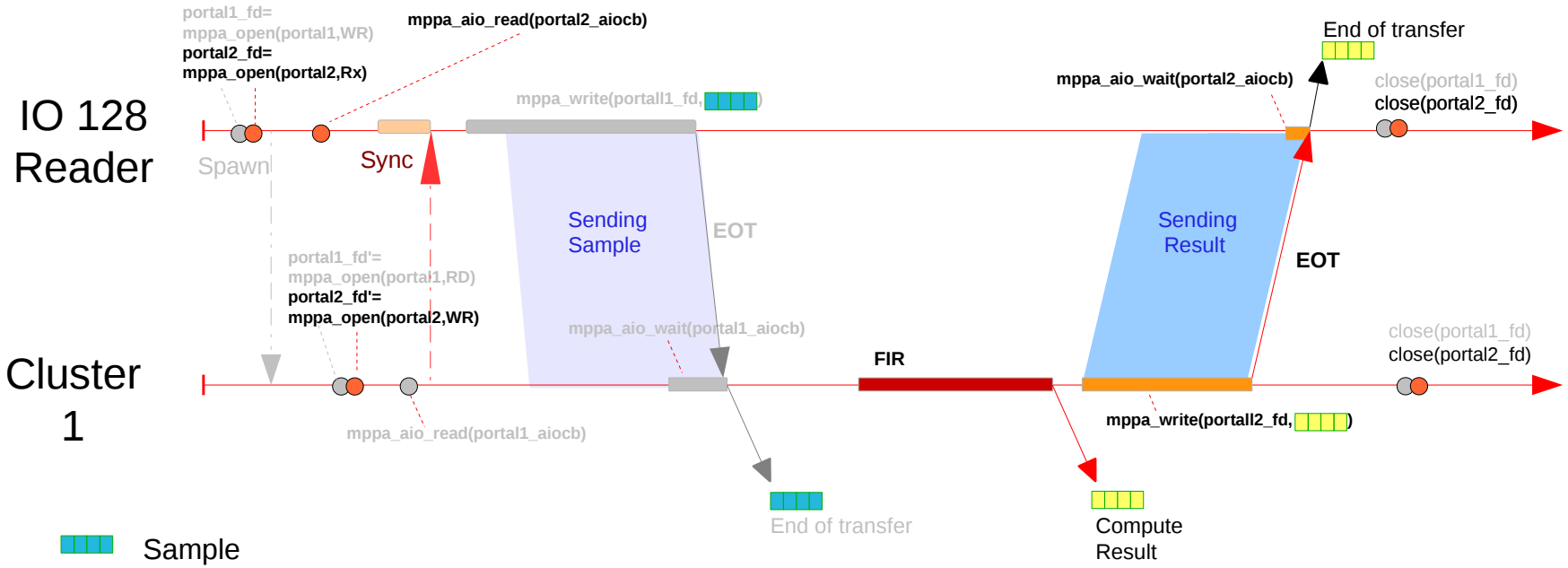
mppa_close : end the cluster 1 process.

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- STEP 2 : Send 'Result' from Cluster 1 to IO DDR



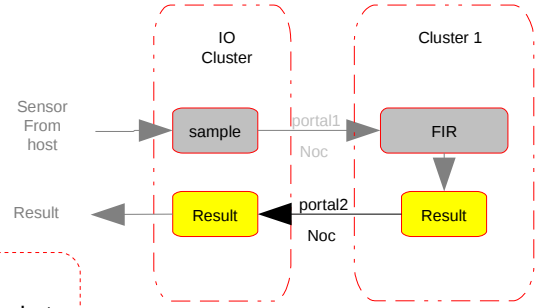
portal1= "/mppa/portal/128:2";

portal2= "/mppa/portal/1:3";

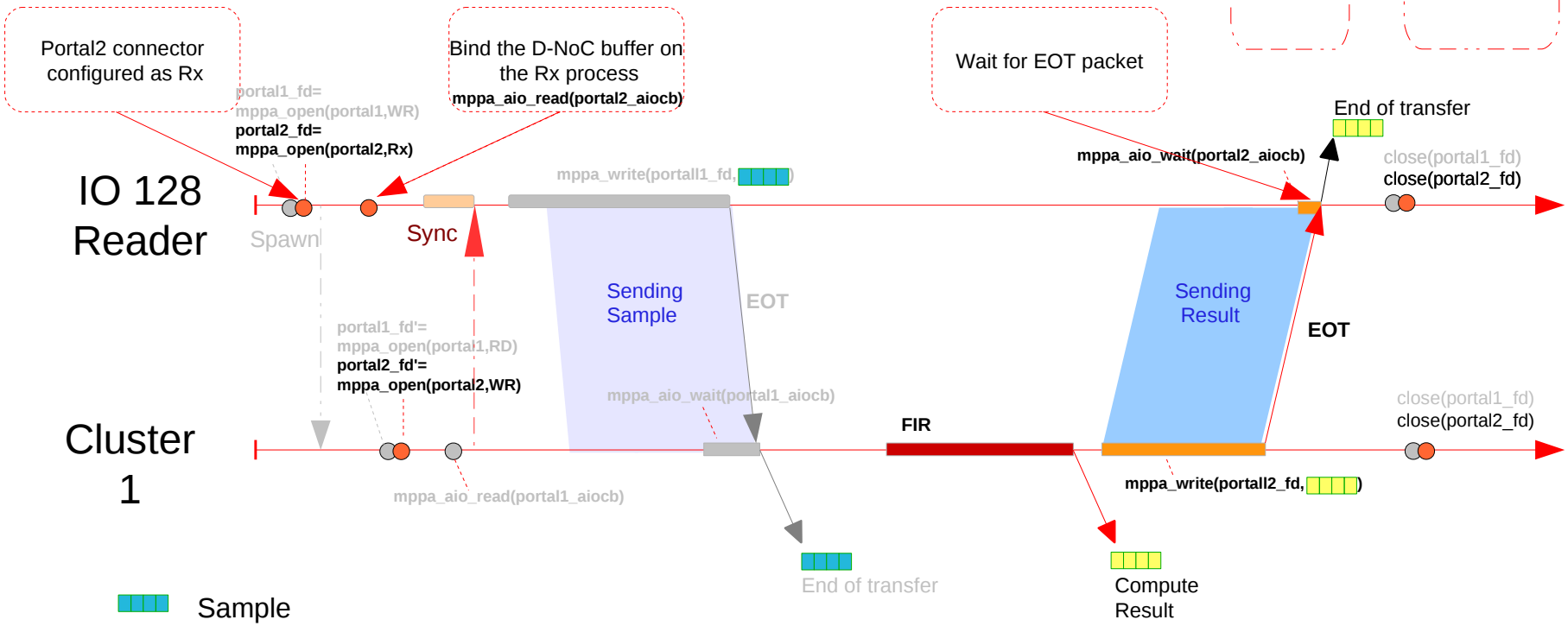
mppa_aiocb_t ***portal_aiocb*** ***// srtuct with portal_fd, sizeof data ; and address of data.***

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- **STEP 2 :** Send 'Result' from Cluster 1 to IO DDR



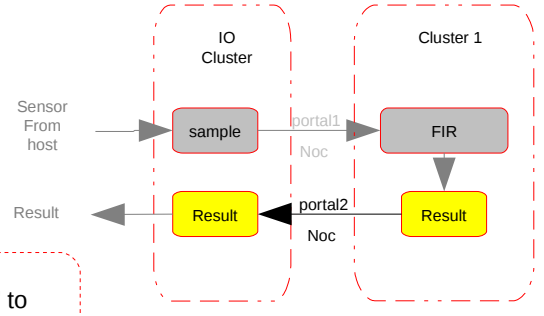
portal1= "/mppa/portal/128:2";

portal2= "/mppa/portal/1:3";

mppa_aiocb_t ***portal_aiocb*** ***// srtuct with portal_fd, sizeof data ; and address of data.***

Exercise1&2

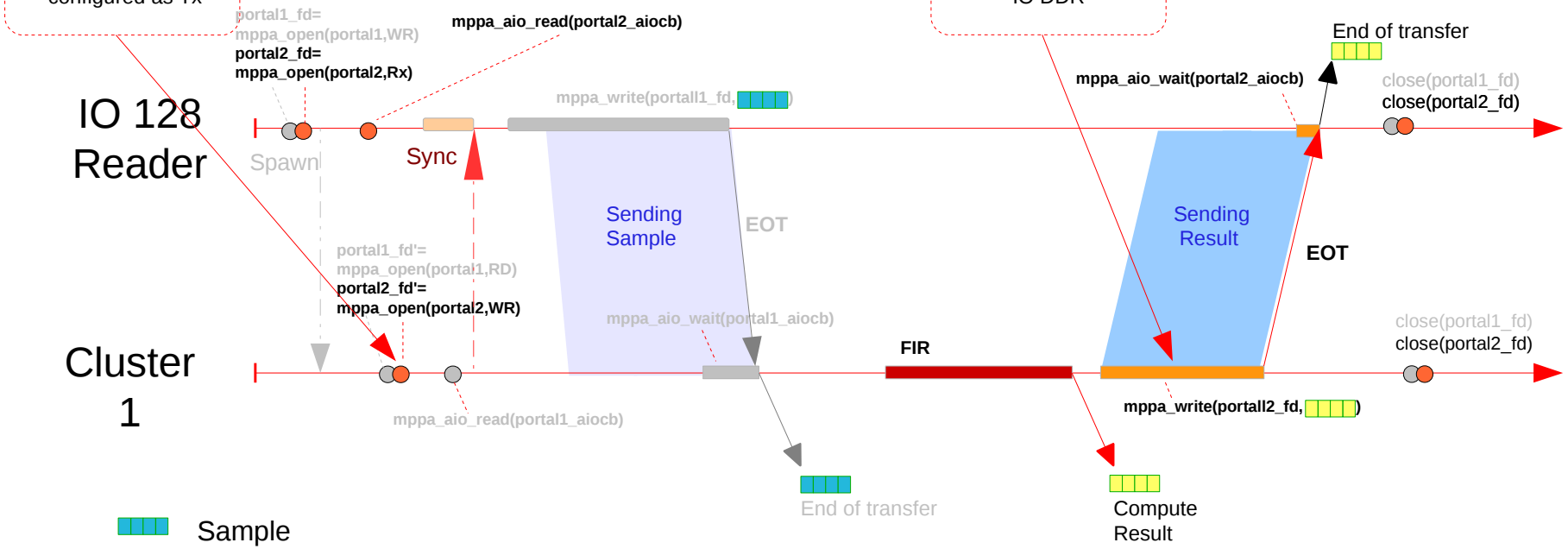
/posix_exercise/exercise1&2/low-pass-filter



- **STEP 2 :** Send 'Result' from Cluster 1 to IO DDR

Portal2 connector configured as Tx

Send 'Result' to IO DDR



Sample

Result

Synchronisation

Spawn

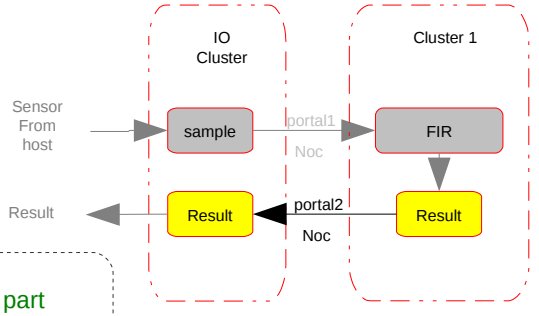
portal1= "/mppa/portal/128:2";

portal2= "/mppa/portal/1:3";

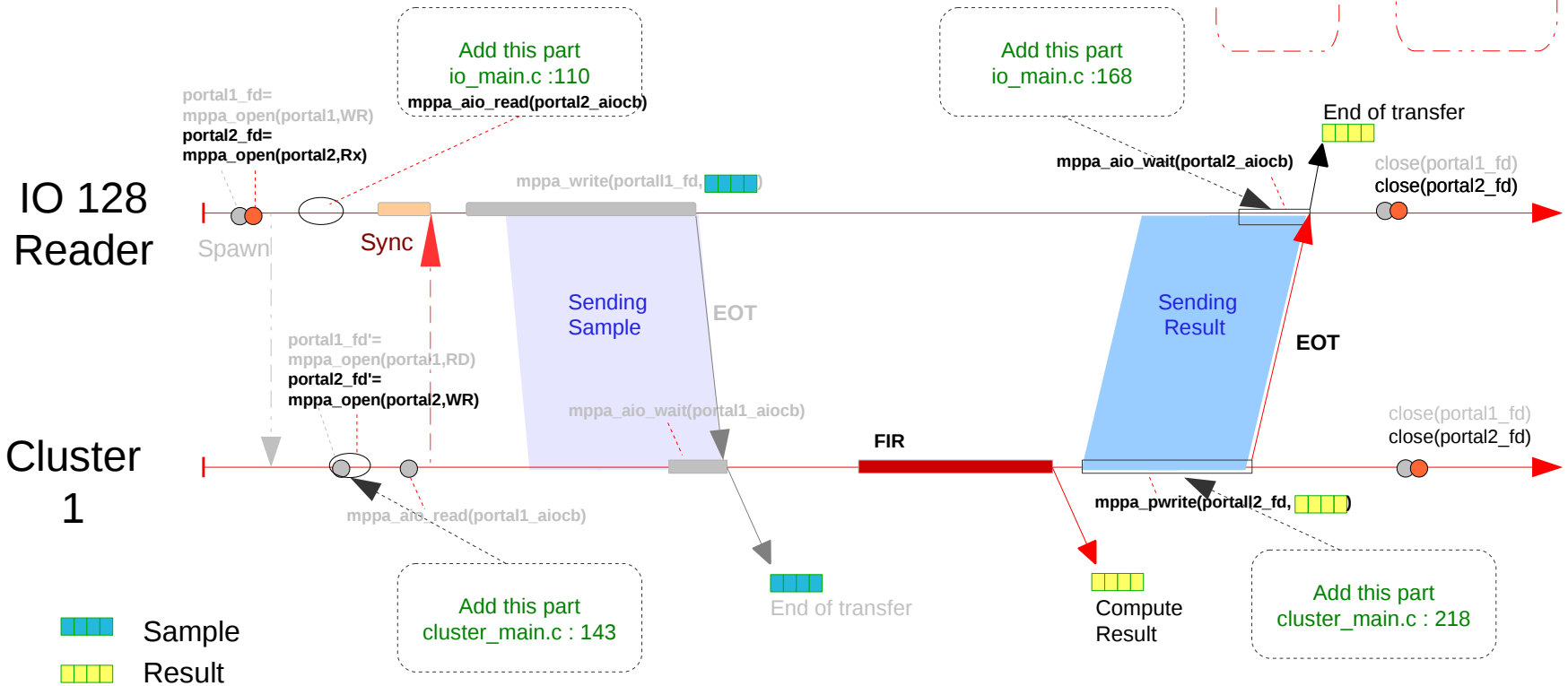
mppa_aiocb_t ***portal_aiocb*** ***// srtuct with portal_fd, sizeof data ; and address of data.***

Exercise1&2

/posix_exercise/exercise1&2/low-pass-filter



- STEP 2 : Send 'Result' from Cluster 1 to IO DDR



portal1= "/mppa/portal/128:2";

portal2= "/mppa/portal/1:3";

mppa_aiocb_t ***portal_aiocb*** ***// srtuct with portal_fd, sizeof data ; and address of data.***

POSIX LEVEL

- MPPA Architecture
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
- Getting started MPPA programming:
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

NodeOS , RTEMS Presentation

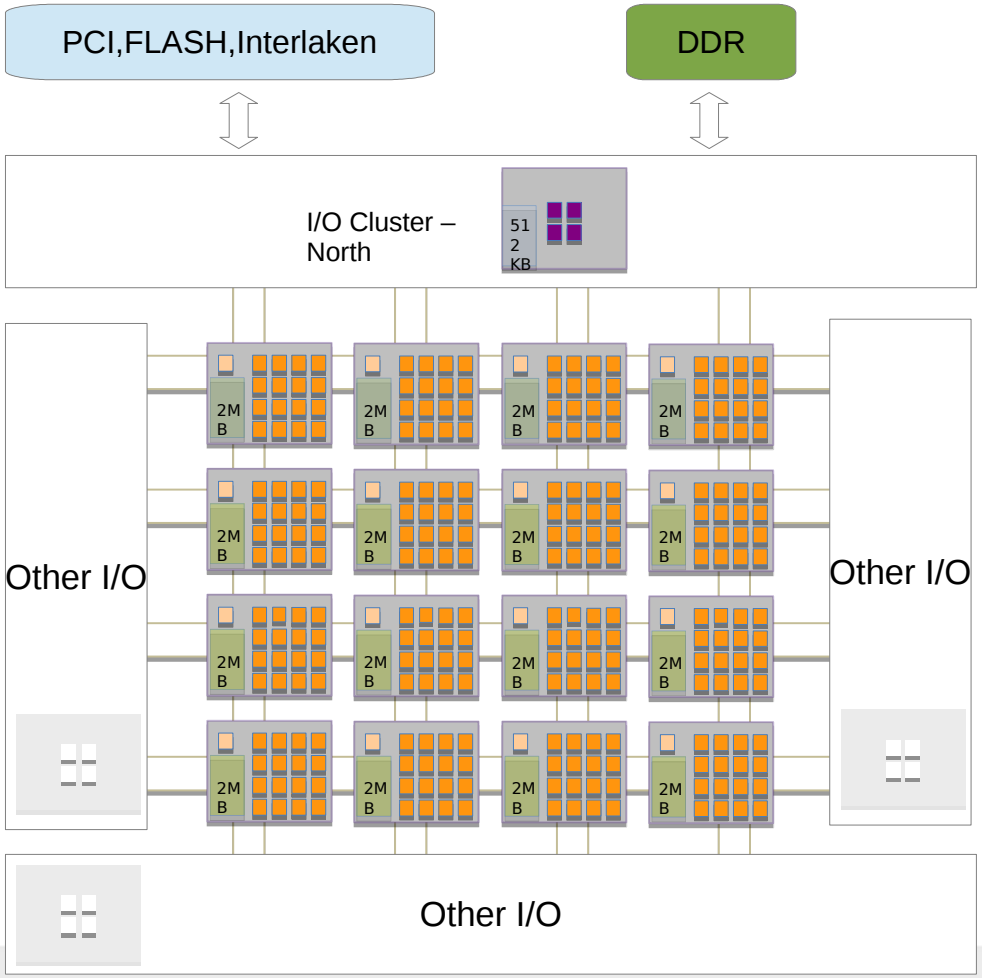
- Implementation on the MPPA
- Parallelism using Posix Threads
- Parallelism using OpenMP

NodeOS , RTEMS Presentation

- Implementation on the MPPA

NodeOS , RTEMS Presentation

- Implementation on the MPPA



Compute Cluster -NodeOS

I/0 cluster -RTEMS

NodeOS , RTEMS Presentation

- Implementation on the MPPA

Target :	I/O Quadcore : ETH IODDR	Compute cluster : 16 cores (2MB SMEM)
Running on :	RTEMS (Real-Time Executive for Multiprocessor Systems)	NodeOS
Goal :	Interface management	Computation
Compatible :	POSIX C POSIX Threads	POSIX C POSIX Threads
Limitations :	Default number of threads limited to 20 To modify : CONFIGURE_MAXIMUM_POSIX_THREADS N in the main project	16 working threads dispatched on the 16 cores

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Posix-Threads Subset
  - Compilation and Simulation:ISS
  - Posix Thread example : « helloworld »
  - Posix Thread exercise : « low-pass filter »

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Posix Threads Subset

pthread_attr_init	***pthread_spin_trylock***
pthread_attr_setstack	***pthread_spin_unlock***
pthread_create	***pthread_mutex_destroy***
pthread_join	***pthread_mutex_init***
pthread_exit	***pthread_mutex_lock***
pthread_spin_destroy	***pthread_mutex_trylock***
pthread_spin_init	***pthread_mutex_unlock***
pthread_spin_lock	***....***
pthread_spin_trylock	

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Compilation and Simulation:ISS

NodeOS

mode	Natif : x86	k1
compilation	gcc	k1-gcc with flag -mos=nodeos
Lib :	-I/.../k1tools/include -L/.../k1tolls/lib64 -Wl,-rpath=/usr/local/k1tools/lib64 -lmppaipc_native	-lmppaipc
simulation	./executable	k1-cluster mcluster=node ./executable

Include <mppa/osconfig.h> in the file containing the main() function.

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Compilation and Simulation:ISS

RTEMS

mode	Natif : x86	k1
compilation	gcc	k1-gcc with flag -mos=rtems
Lib :	-I/.../k1tools/include -L/.../k1tolls/lib64 -Wl,-rpath=/usr/local/k1tools/lib64 -lmppaipc_native	-lmppaipc
simulation	./executable	k1-cluster mcluster=ioddr ./executable

Include <mppa/osconfig.h> in the file containing the main() function.

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Posix Thread example : « helloworld »

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Posix Thread example : « helloworld »

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include <mppa.h>
#include <mppa/osconfig.h>

#define THREAD_NO     15

int param[THREAD_NO];
```

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Posix Thread example : « helloworld »

```
void *task(void *arg)
{
        int id = ((int *)arg)[0];
        printf("Hello from %d processor %d\n", id, __k1_get_cpu_id());
        param[id] = id + 42;
        pthread_exit((void *)&param[id]);
        return NULL;
}
```

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Posix Thread example : « helloworld »

```
int main(int argc, char** argv)
{
  pthread_t t[THREAD_NO];
  int i, res;
  void *ret;
  int ret_code;
  printf("*** pthread16 TEST with %d threads ***\n", THREAD_NO);
  for (i = 0; i < THREAD_NO; i++) {
    param[i] = i;
    while(1) {
      res = pthread_create(&t[i], NULL, task, &param[i]);
      if( res == 0) {
        break;
      }
      if( res == EAGAIN ){
        usleep(100000);
      } else {
        printf("pthread_create failed i %d, res = %d\n", i, res);
        abort();
      }
    }
```

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Posix Thread example : « helloworld »

```
    printf ("pthread create launched thread %d\n", (int) t[i]);
  }
  for (i = 0; i < THREAD_NO; i++) {
    if (pthread_join(t[i], &ret) != 0) {
      printf("pthread_join failed\n");
      abort();
    }
    ret_code = ((int *)ret)[0];
    if(ret_code != (i+42)){
      printf("pthread return code for %d failed\n", i);
      abort();
    }
  }
  printf("join OK\n");
  printf("*** END OF TEST ***\n");
  exit(0);
  return 0;
}
```

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Exercise3 : « low_pass_filter_thread »

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Exercise3 : « low_pass_filter_thread »
    - **Step 1** : create thread task :

      convert old function 'firFixed to firFixed_thread' :

      - the goal is to parallelise for (k = 0; k <SAMPLE; k++) on 16 PE.
    - **Step 2** : schedule on 16 PE.

      Use Thread Management Posix function :

      - pthread_create / pthread_join

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Exercise3 : « low_pass_filter_thread »

    Use function Thread Management :
    - pthread_create / pthread_join

NodeOS , RTEMS Presentation

- Parallelism using Posix Threads
  - Exercise3 : « low_pass_filter_thread »

    Use function Thread Management :
    - pthread_create / pthread_join

```
int pthread_create( pthread_t *pthread,
const pthread_attr_t *attr, void *(*start)(void *),
 void *arg  );

/*
* pthread Pointer for where to store the thread ID.
* attr If not NULL, determine the new thread attributes.
* start Pointer to the routine where thread execution
starts.
* arg Argument passed to the start routine.
* return 0 on success, else non-zero error number.
*/
```

```
int pthread_join(pthread_t pthread_id, void **retval);

/*
* pthread_id ID of the target thread to wait for.
* retval If not NULL, the exit status is stored into **retval.
* return 0 on success, else non-zero error number.
*/
```

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - Fork – join Model
  - Compilation and Simulation:ISS
  - OpenMP example : « helloworld »
  - OpenMP exercice : « low-pass-filter »

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - Fork – join Model

NodeOS , RTEMS Presentation

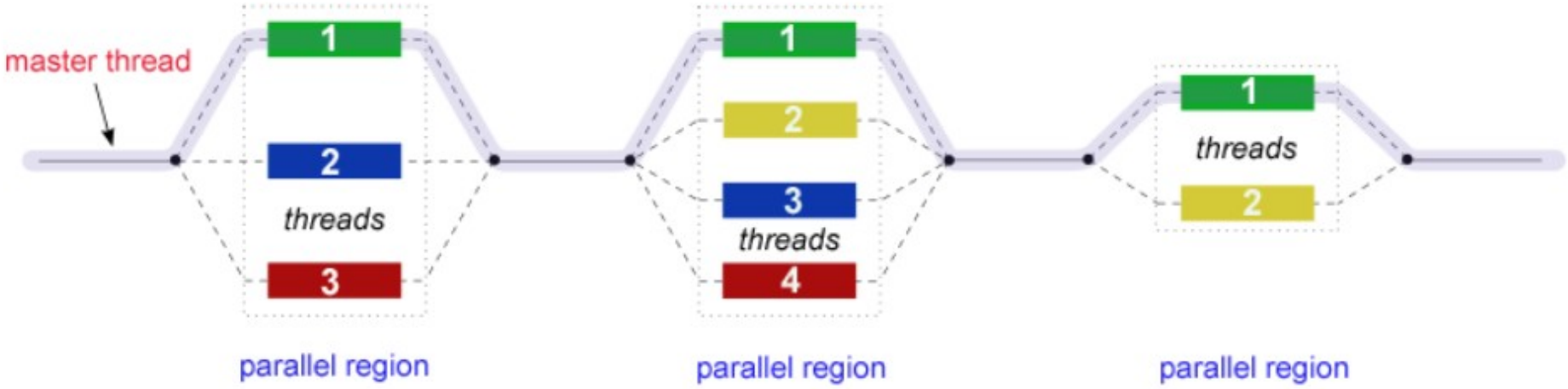
- Parallelism using OpenMP
  - Fork – join Model

FORK : the master thread creates a group of parallel threads.

JOIN : When the threads complete the statements in the parallel region construct, they synchronise and terminate, leaving only the master thread.

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - Fork – join Model



NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - Compilation and Simulation:ISS

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - Compilation and Simulation:ISS

NodeOS

mode	Natif : x86	k1
compilation	gcc	k1-gcc with flag -mos=nodeos
Lib :	-I/.../k1tools/include -L/.../k1tolls/lib64 -lmppaipc_native -lrt **-fopenmp**	-lmppaipc **-fopenmp**
simulation	./executable	k1-cluster mcluster=node ./executable

Include <mppa/osconfig.h>, <omp.h> in the file containing the main() function.

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - Compilation and Simulation:ISS

RTEMS

mode	Natif : x86	k1
compilation	gcc	k1-gcc with flag -mos=rtems
Lib :	-I/.../k1tools/include -L/.../k1tolls/lib64 -lmppaipc_native -lrt **-fopenmp**	-lmppaipc **-fopenmp**
simulation	./executable	k1-cluster mcluster=node ./executable

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - OpenMP example : « helloworld »

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - OpenMP example : « helloworld »

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include <mppa.h>
#include <mppa/osconfig.h>

#include <omp.h>
```

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - OpenMP example : « helloworld »

```
int main (int argc, char *argv[]) {
int th_id, nthreads;
/* Fork a team of threads with each thread having a private th_id variable */
#pragma omp parallel private(th_id)
{
          /* Obtain and print thread id */
          th_id = omp_get_thread_num();
          printf("Hello World from thread %d\n", th_id);

          /* Only master thread does this */
          if ( th_id == 0 ) {
          nthreads = omp_get_num_threads();
          printf("There are %d threads\n",nthreads);
          }
}/* All threads join master thread and terminate */
}
```

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - OpenMP exercise : « low-pass-filter »

NodeOS , RTEMS Presentation

- Parallelism using OpenMP
  - OpenMP exercise : « low-pass-filter »

used DO / For Directive :

example :

```
#pragma omp parallel for
 for (i = 0; i < slice; i++) {
bufferOut[i]=bufferIn[i]*1/N
}
```

POSIX LEVEL

- MPPA Architecture
  - Hardware Configuration
  - Software and Hardware
  - Network-On-Chip
- Getting started with MPPA programming:
  - How to program a low-pass filter application
  - Inter-Cluster Communication presentation
  - NodeOS, RTEMS presentation
    - parallelism using Posix Threads
    - parallelism using Open MP
  - PCI Communication

PCI Communication

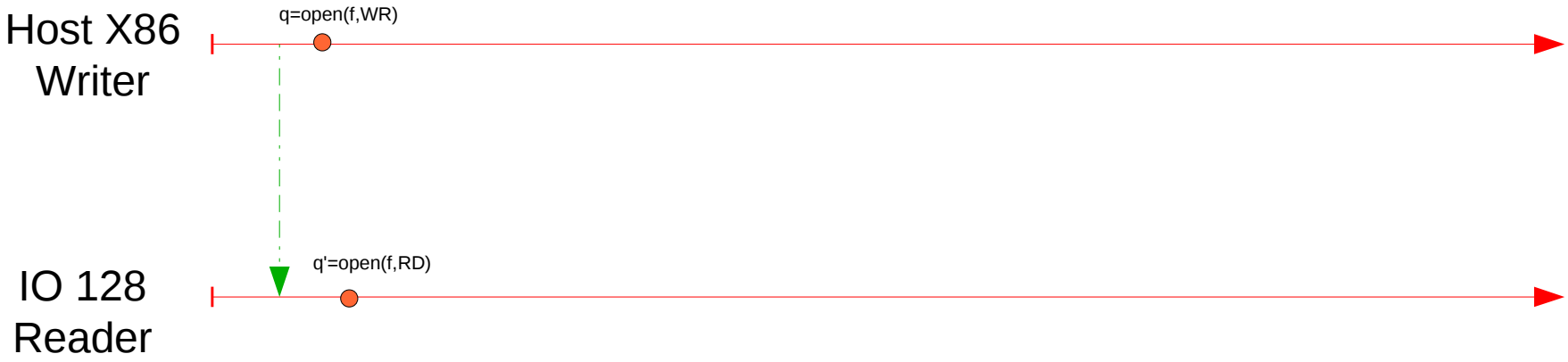
- PCI-IPC Transfer types :
  - Buffer Connector
  - MQueue Connector

PCI Communication

- PCI-IPC Transfer types :
  - Buffer Connector :

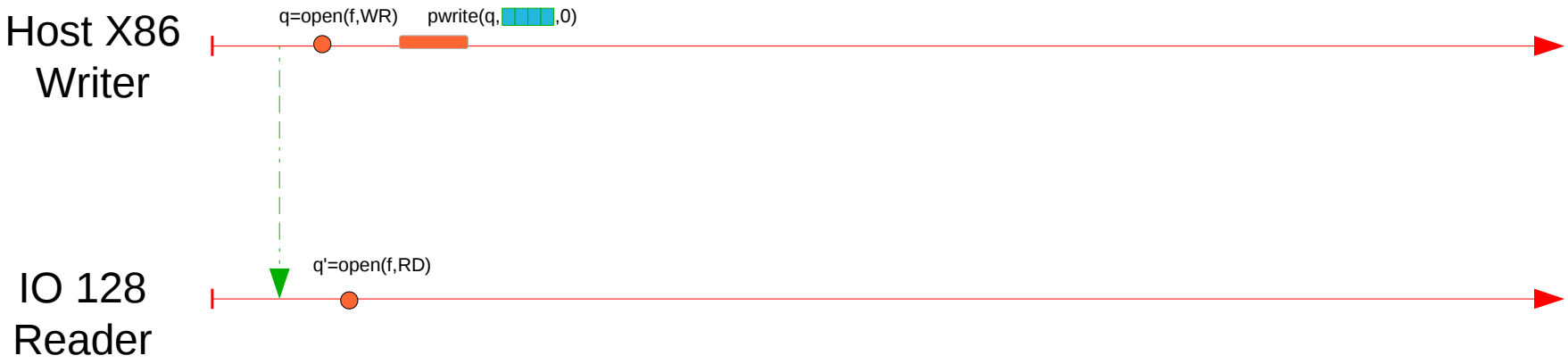
PCI Communication

- PCI-IPC Transfer types :
  - Buffer Connector :



PCI Communication

- PCI-IPC Transfer types :
  - Buffer Connector :

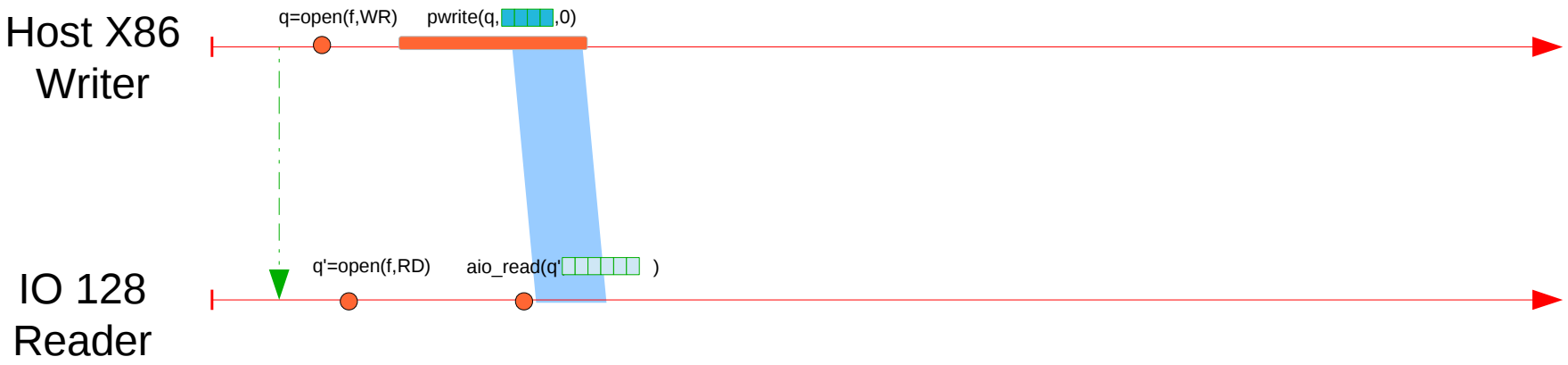


Spawn

***f= "/mppa/buffer/pcie0#2/host#2";**

PCI Communication

- PCI-IPC Transfer types :
  - Buffer Connector :

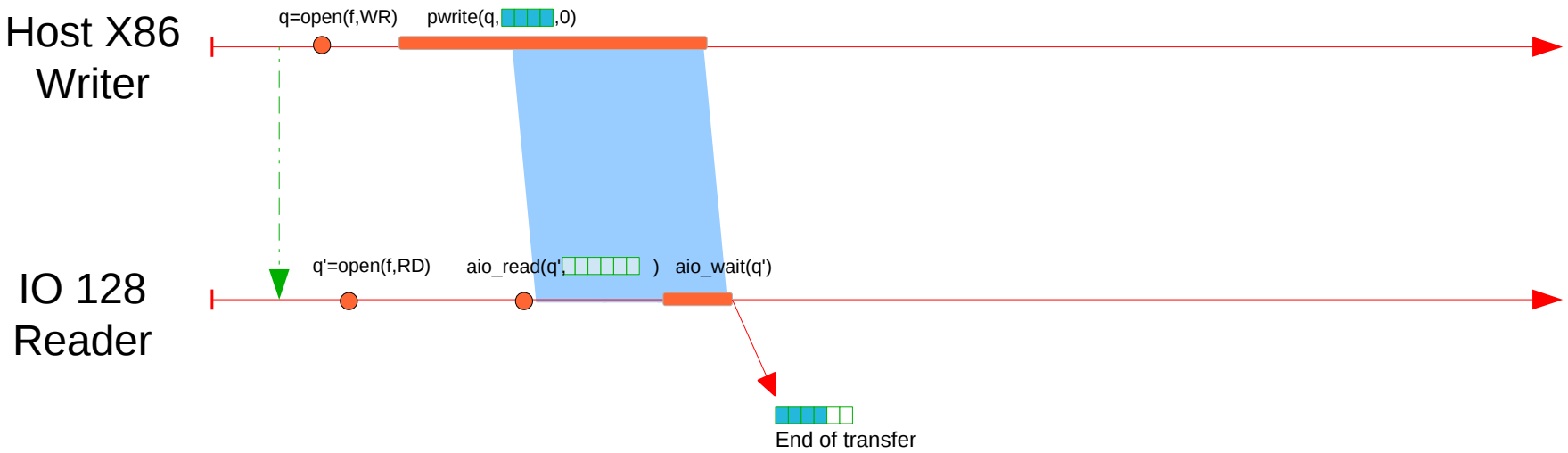


Spawn

f= "/mppa/buffer/pcie0#2/host#2";

PCI Communication

- PCI-IPC Transfer types :
  - Buffer Connector :

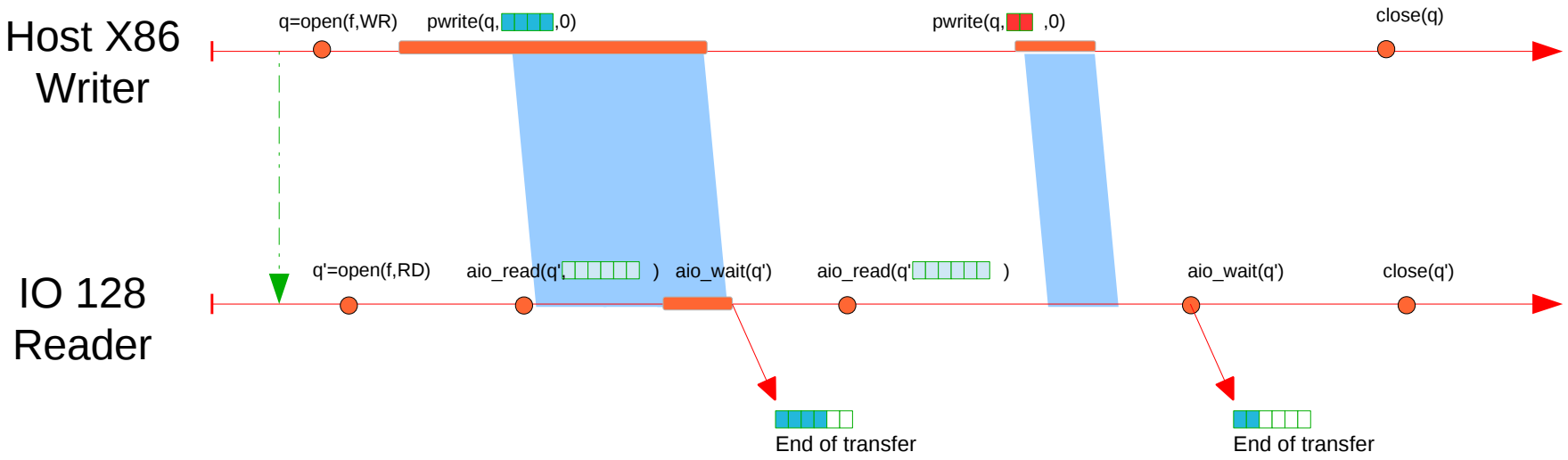


Spawn

***f= "/mppa/buffer/pcie0#2/host#2";**

PCI Communication

- PCI-IPC Transfer types :
  - Buffer Connector :



Spawn

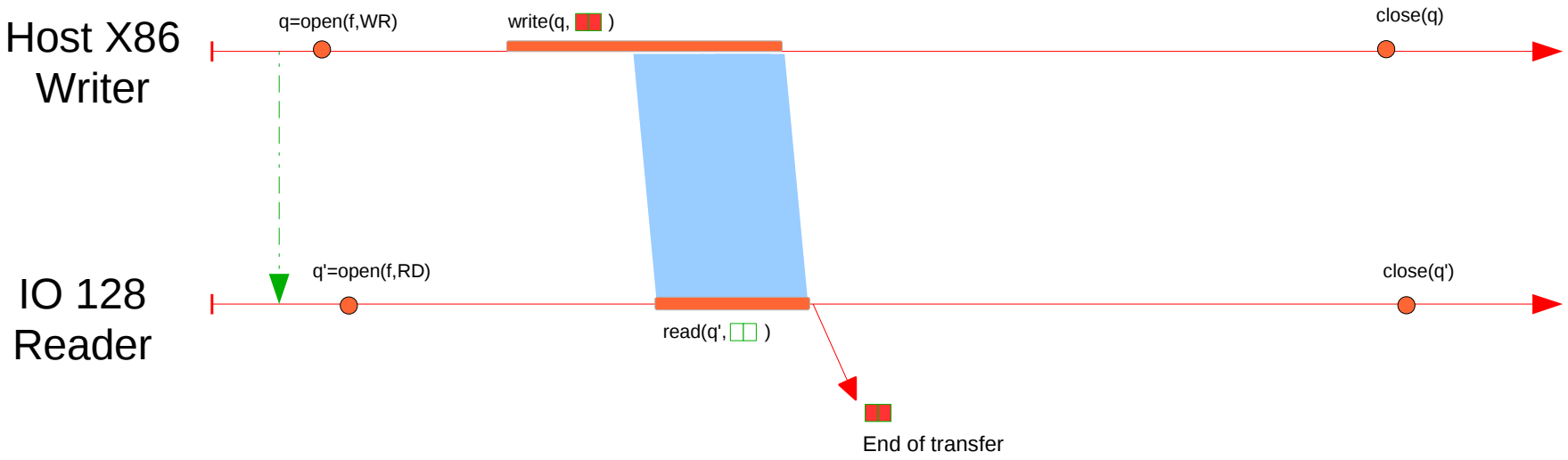
***f= "/mppa/buffer/pcie0#2/host#2";**

PCI Communication

- PCI-IPC Transfer types :
  - MQueue Connector

PCI Communication

- PCI-IPC Transfer types :
  - MQueue Connector



PCI Communication

- PCI-Buffer example for a low-pass filter application
  - PCI Buffer Descriptor
  - Rx File Operation
  - Tx File Operation

PCI Communication

- PCI-Buffer example for a low-pass filter application
  - PCI Buffer Descriptor

PCI Communication

- PCI-Buffer example for a low-pass filter application
  - PCI Buffer Descriptor
  - Pathname for PCI connector Buffer :
    - ***/mppa/buffer/rx node#number/tx_node#number***
    - ***Example :***
      - ****channel = "/mppa/buffer/pcie0#2/host#2";***

PCI Communication

- PCI-Buffer example for a low-pass filter application
  - Rx File Operation
    - ***mppa_open(pathname, O_RDONLY)*** Open the Channel connector for the Rx process.
    - ***mppa_read(buffer fd, data, size)*** Read up to *size* bytes to the address *data*
    - ***mppa_pread(buffer fd, base, size, offset)*** Read up to *size* bytes to the address *data* from remote offset *offset*
    - ***mppa_aio_read(buffer aiocb)*** Initiate asynchronous read of data specified by the struct mppa_aiocb *buffer_aiocb. At most one call to mppa_aio_read() on a given file descriptor may be pending at any time.
    - ***mppa aio return(buffer aiocb)*** Complete the asynchronous read, and return the size of the buffer allocated by mppa_aio_read(). In case of error, return -1.
    - ***mppa aio wait(buffer aiocb)*** Block until a previous call to mppa aio read(buffer aiocb) is completed. Return value is the same as for mppa_aio_return().
    - ***mppa close(buffer fd)*** Release the Rx process PCI resources.

PCI Communication

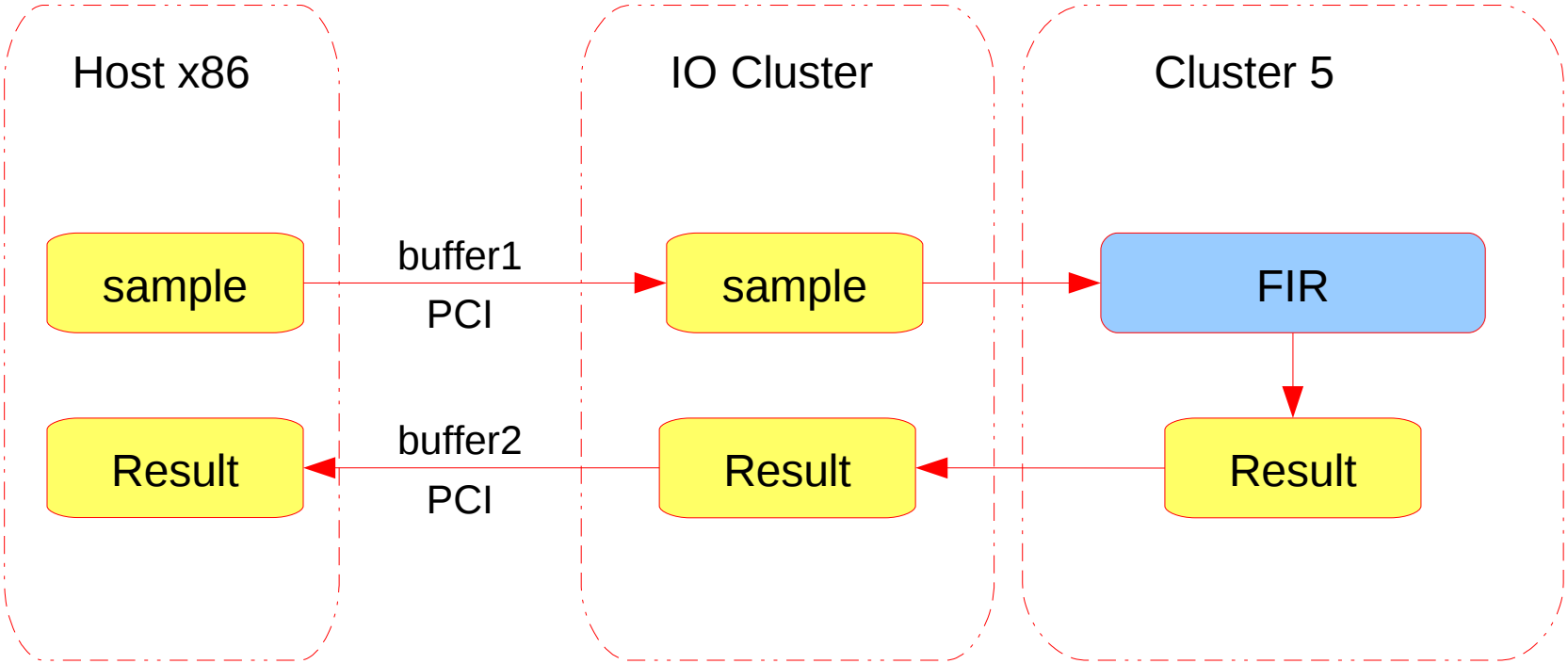
- PCI-Buffer example for a low-pass filter application

  - Tx File Operation

    - ***mppa_open(pathname, O WRONLY)*** Open the Buffer connector for a Tx process
    - ***mppa_write(buffer fd, base, size)*** Remote write of *size* bytes from the *base* address in the Tx process, to the Rx process address space at offset 0
    - ***mppa_pwrite(buffer fd, base, size, offset)*** Remote write of mppa_size_t size bytes starting at the mppa_off_t offset in bytes from the void *base address in the Tx process, to the Rx process address space opening at mppa_off_t offset in bytes
    - ***mppa_pwrites(buffer fd, base, esize, ecount, sstride, tstride, offset)*** Strided remote write of *esize* x *ecount* bytes from the *base* address in the Tx process, to the Rx process address space opening at *offset* in bytes
    - ***mppa_aio_write(buffer aiocb)*** Use the PCI DMA engine to asynchronously transfer data specified within the aiocb. At most one call to mppa_aio_write() on a given file descriptor may be pending at any time
    - ***mppa_aio_return(buffer aiocb)*** Complete the asynchronous request, and return the count of bytes transferred for an asynchronous write. In case of error, return -1.
    - ***mppa aio wait(buffer aiocb)*** Block until a previous call to mppa_aio_write(buffer aiocb) is complete. Return value is the same as for mppa_aio_return()
    - ***mppa_close(buffer fd)*** Release the Tx process PCI resources

PCI-Buffer example for a low-pass filter application

- Exercise4 : PCI_Low-pass-filter :



PCI Communication

- PCI-Buffer example for a low-pass filter application
  - Exercise4 : PCI_Low-pass-filter :

PCI-Buffer example for a low-pass filter application

- Exercise4 : PCI_Low-pass-filter
  - Types and functions used :

```
int mppa_open(const char *path, int flags);
/*
* path Path to the file.
* flags O_RDONLY or O_WRONLY.
* mode Depends on the type of connector and flags value.
* return A file descriptor on success, -1 on error.
* In case of error, the errno variable is set.
*/
```

PCI-Buffer example for a low-pass filter application

- Exercise4 : PCI_Low-pass-filter
  - Types and functions used :

Tx Process

```
mppa_ssize_t mppa_pwrite(int fd, const void *base, mppa_size_t size, mppa_off_t offset);
/*
* fd file descriptor.
* base Base of data to write.
* size Size of data to write.
* offset Offset from the base of the special file.
* return Count of bytes written on success, -1 on error.
* In case of error, the errno variable is set.
*/
```

PCI-Buffer example for a low-pass filter application

- Exercise4 : PCI_Low-pass-filter
  - Types and functions used :

```
int mppa_read(int fd, void *data, mppa_size_t size);
/*
* aiocb Pointer to a mppa_aiocb_t object.
* return 0 on success, -1 on error.
* In case of error, the errno variable is set.
*/
```

Rx Process

Developing An Application on the TC2 Board

- APIs include specific details about chips used in an application
  - Eg : mppa_spawn has parameters to define the board number and the mppa for each executable
- Compilation :
  - -mboard=tc2
- Same multibinary process as for the AB01
- PCIe redirection is automatic
- Traces only available on Cluster 0 and Cluster 1

QUESTIONS ?