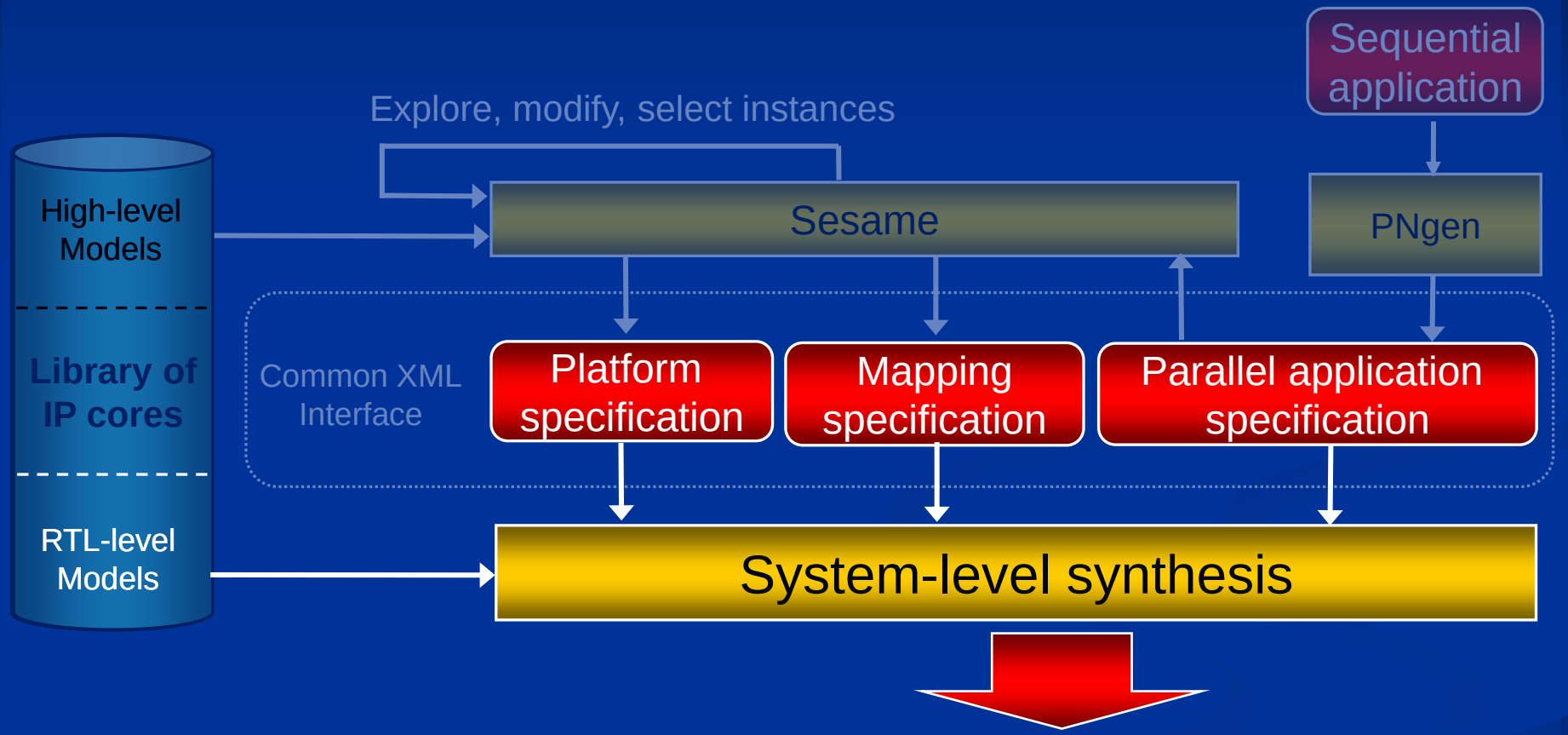
System-level Synthesis

Todor Stefanov

Leiden Embedded Research Center,
Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands

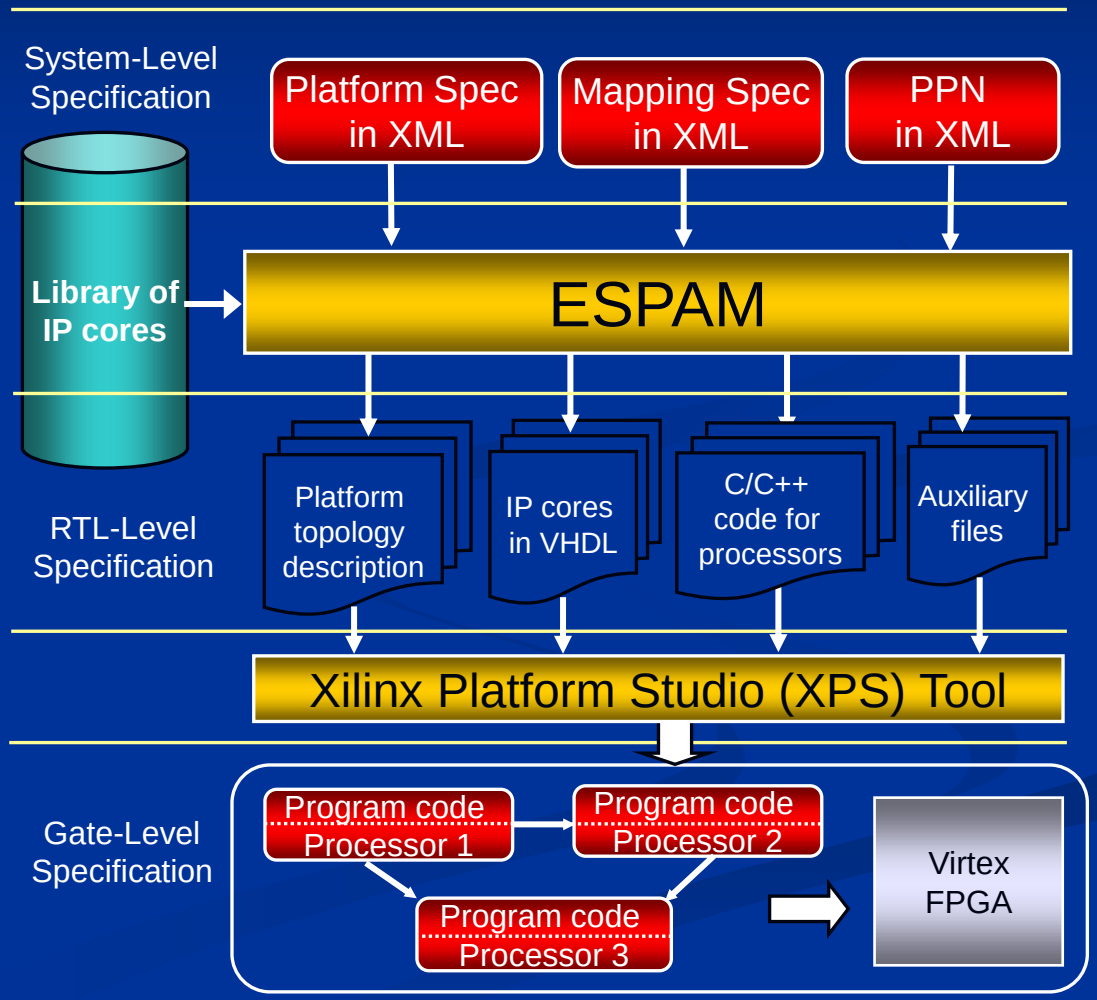
System-level Synthesis: ESPAM tool



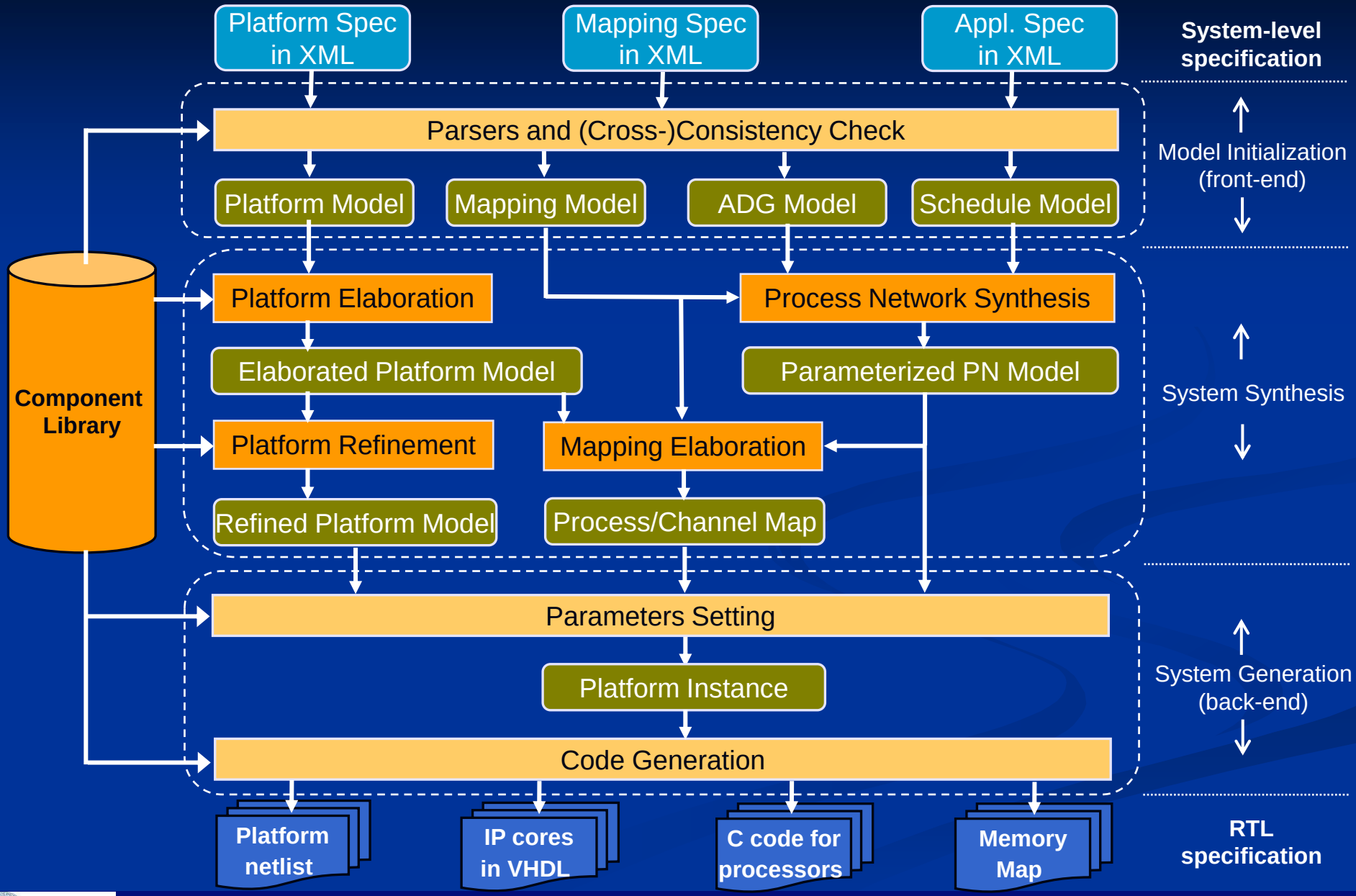
ESPAM

Embedded System-level Platform Synthesis and Application Mapping

- Simple descriptions in XML format
- Automated System-to-RTL Level conversion and software code generation
- Ready for direct implementation
- FPGA-based prototyping with Xilinx boards
- All of this in a matter of hours



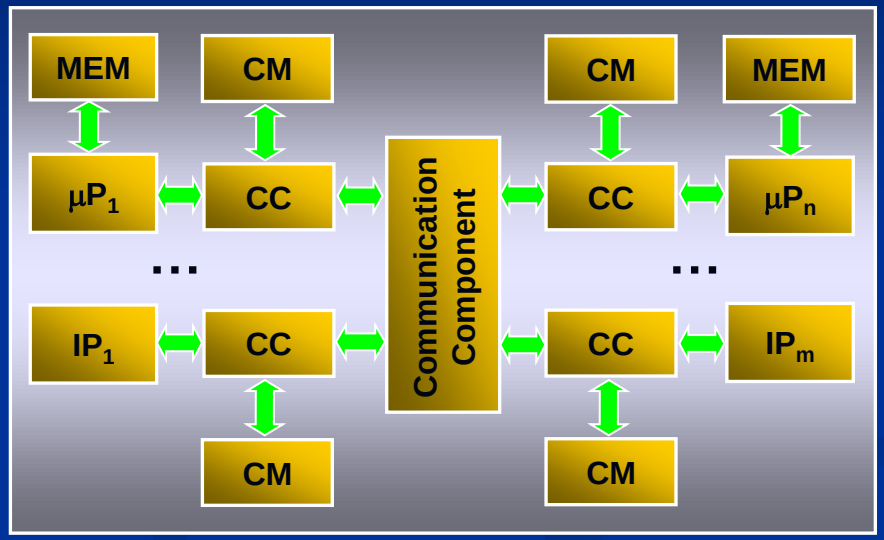
ESPAM: Internal Structure



MP-SoCs currently considered

Library of parameterized components:

- Processing Components:
  - Programmable processors
  - Hardware IP Cores
- Memory Components:
  - Program, Data (on-chip and external) Memory (MEM)
  - Communication Memory (CM)
- Communication Components:
  - Point-to-point network
  - Crossbar switch
  - Shared bus with Round-Robin, Fixed Priority, or TDMA arbitration

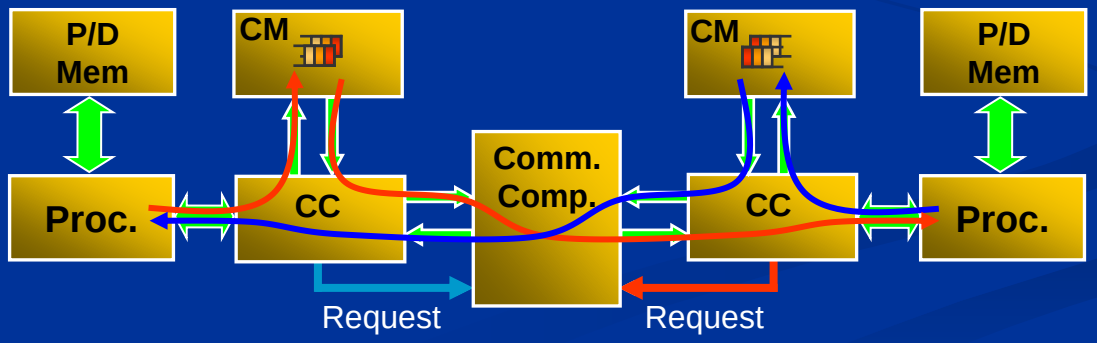


Many alternative platforms can be constructed fast and easily by instantiating different type/number of components and setting their parameters.

- Communication Controller (CC) – interface between processing, memory, and communication components

Communication and Synchronization

- Data communication and synchronization between processors only through ***FIFOs*** mapped in the Communication Memories (CM)
- A processor can write only to its ***local*** CM
- A processor can access other CMs only for ***read*** operations through the communication component using ***requests***
- The synchronization mechanism is implemented by ***Read*** and ***Write SW*** primitives that interact directly with the ***Communication Controllers***



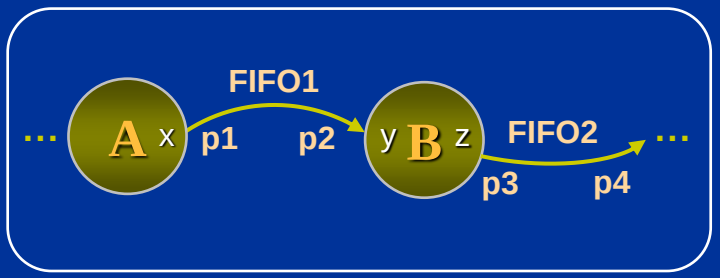
Platform Synthesis and Programming

- Platform Elaboration and Refinement
- Mapping of processes
- FIFOs to CMs mapping
- Memory map of the system
- Program code for each processor
- Read and write synchronization primitives

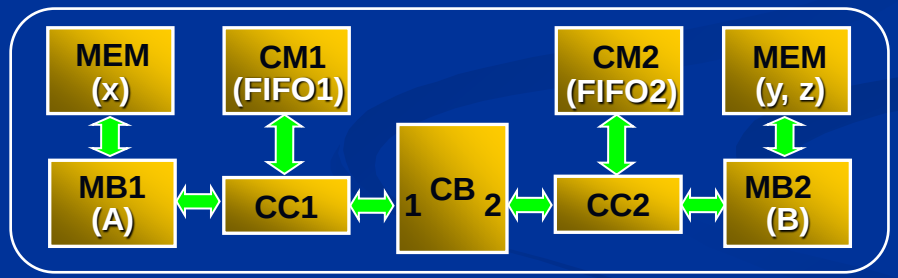
Program Fragment

```
int q=0;

. . .

for (int i=0; i<=N; i++) {
    A( &q ) ;
    B( &q ) ;
}
. . .
```

Polyhedral Process Network



Platform



Mapping

```
. . .

A -> MB1
B -> MB2
. . .
```

Code of MB1

```
int x=0;
for (int i=0; i<=N; i++) {

    execute_A( &x ) ;
    write( p1, x, 1 );
}
```

Code of MB2

```
int y=0, z=0;
for (int i=0; i<=N; i++) {
    read( p2, y, 1 );
    execute_B( y, &z ) ;
    write( p3, z, 1 );
}
```

Read/Write addresses of the FIFOs

```
#define p1 0xe0000008 // write addr. FIFO1
#define p2 0x00010000 // read addr. FIFO1

#define p3 0xe0000008 // write addr. FIFO2
#define p4 0x00020000 // read addr. FIFO2
```

Write and Read Primitives

Write Synchronization Primitive

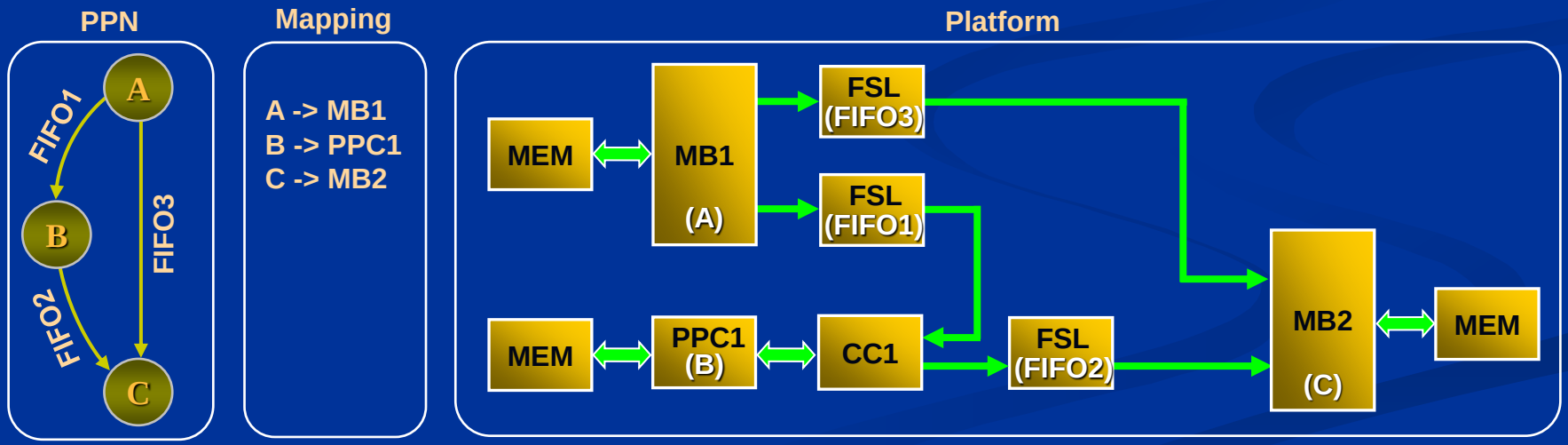
```
#define writeCM( pos, value, n ) \

    int i;\
    volatile int *isFull;\
    volatile int *outPort = (volatile int *) pos;\

    isFull = outPort + 1;\

    for (i = 0; i < n; i++) {\
        while ( *isFull )  {  };\
        *outPort = ((volatile int *) value)[ i ];\
    }\
```

Read Synchronization Primitive

```
#define readCM( pos, value, n ) \

    int i;\
    volatile int *isEmpty;\
    int inPort = (int) pos;\
    (volatile int *) dataReqReg = (volatile int *) 0xE0000000;\

    isEmpty = dataReqReg + 1;\
    *dataReqReg = 0x80000000 | inPort;\

     for (i = 0; i < n; i++) {\
        while ( *isEmpty )  {  };\
        ((volatile int *) value)[ i ] = * dataReqReg;\
     }\

     *dataReqReg = 0x7FFFFFFF & inPort;\
```

Platform Synthesis: Point-to-Point

- No Communication Component, no communication overhead!
  - Number of processes in PPN is equal to number of processors in the platform
  - Only one process mapped onto one processor



Write and Read Primitives

Write FIFO Synchronization Primitive

```
#define write( pos, value, n ) \

    int i;\
    volatile int *isFull;\
    volatile int *outPort = (volatile int *) pos;\

    isFull = outPort + 1;\

    for (i = 0; i < n; i++) {\
        while ( *isFull )  {  };\
        *outPort = ((volatile int *) value)[ i ];\
    }\
```

Read FIFO Synchronization Primitive

```
#define read( pos, value, n ) \

    int i;\
    volatile int *isEmpty;\
    volatile int *inPort = (volatile int *) pos;\

    isEmpty = inPort + 1;\

    for (i = 0; i < n; i++) {\
        while ( *isEmpty )  {  };\
        ((volatile int *) value)[ i ] = *inPort;\
    }\
```

Write FSL Synchronization Primitive

```
#define writeFSL( pos, value, n ) \

    int i;\

    for (i = 0; i < n; i++) {\
        microblaze_bwrite_datafsl(
                    ((volatile int *) value)[ i ], pos);\
    }\
```

Read FSL Synchronization Primitive

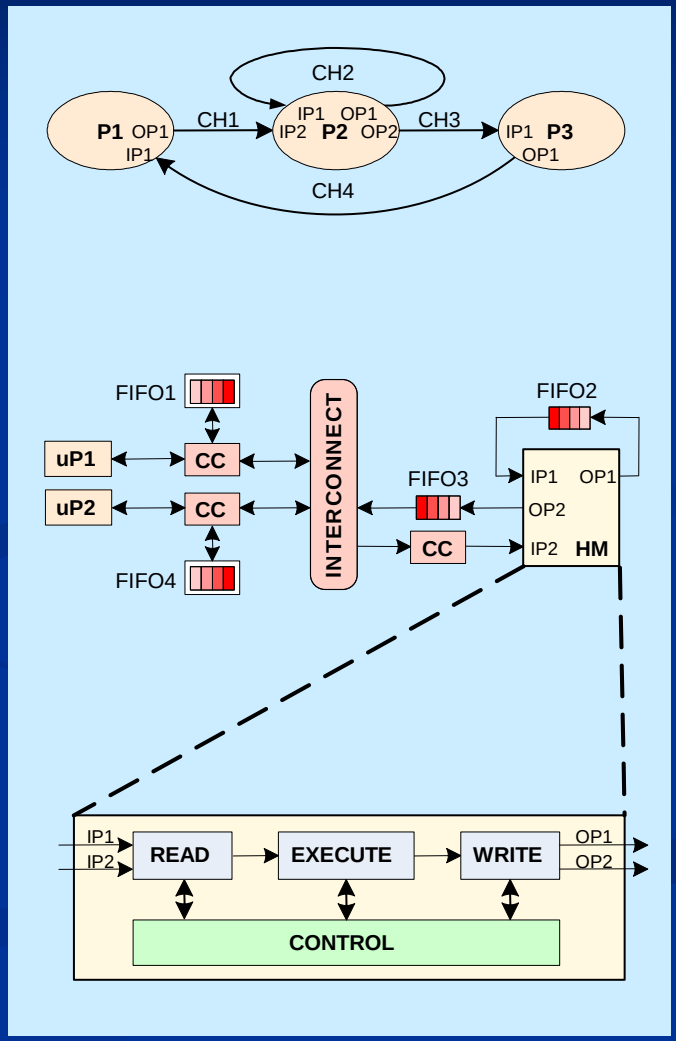
```
#define writeFSL( pos, value, n ) \

    int i;\

    for (i = 0; i < n; i++) {\
        microblaze_bread_datafsl(
                    ((volatile int *) value)[ i ], pos);\
    }
```

Dedicated Hardware IPs Integration with ESPAM

- To meet higher application requirements,
  - integrate hardware IP cores into ESPAM generated systems
- Done by automated HW Module (wrapper) generation
  - including/wrapping third-party IP core
- Features of a generated HW module
  - modularity, i.e., HW Module consisting of well defined parameterized components
  - clearly defined interfaces between components of a HW module

PPN to Heterogeneous System

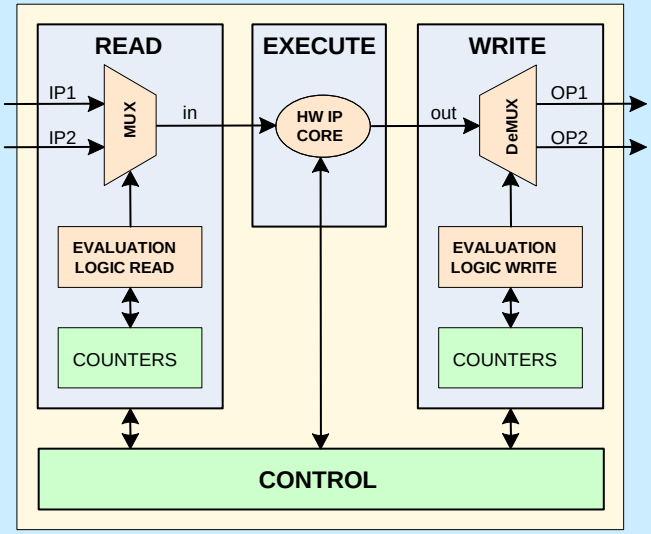
- Processes mapped to programmable processors and/or dedicated HW IPs
- Hardware Modules – wrappers around predefined IP cores
  - **Read Block**
  - **Execute Block, i.e., the IP core**
  - **Write Block**
  - **Control Block**
- HW IPs must provide:
  - **Function call behavior**
  - **Unidirectional I/O data interfaces**
  - **Enable/Valid control interface**



Structure of a HW Module

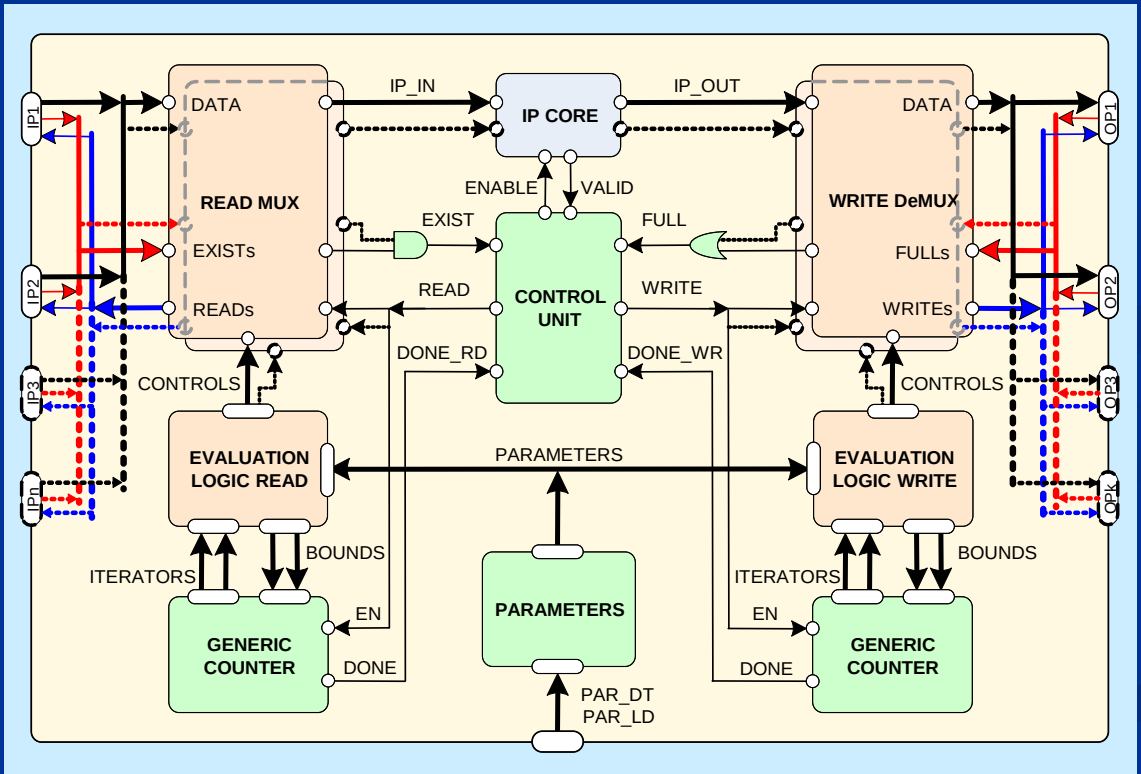
- Read Block
  - Fetch data from communication channels
  - For each input argument select from which port to fetch the data using the control information derived from the PPN
- Execute Block
  - A functional sub-wrapper for the IP core
- Write Block
  - Write back the results from the execution to the communication channels
  - For each output argument select which port to receive the corresponding data
- Control Block
  - Control and synchronize reading, writing and execution

```
1  // process P2
2  void main( ) {
3    for( int  i=2; i<=N; i++ )                          CONTROL
4      for( int  j=1; j<=M+i; j++ )  {

5          if( i-2 == 0 )
6            read( IP1, in_0, size  );                  READ
7          if( i-3 >= 0 )
8            read( IP2, in_0, size );

9        execute( in_0, out_0);                         EXECUTE

10         if( -i+N-1 >= 0 )
11           write( OP1, out_0, size );
12         if( i-N == 0 ) {                             WRITE
13           write( OP2, out_0, size );
14     } // for j
15 } // main
```



HW Module Structure in Detail

- Composed of well defined components
- Clearly defined component interfaces



ESPAM Summary

- **Reduced design time:**
  - Implementations correct by construction
    - no simulations are needed
  - Complete implementation and programming
    - about 2 hours for systems with 5 processors
  - Design space exploration is feasible at implementation level
    - 100% accuracy

Making system-level design take off

http://daedalus.liacs.nl