ForSyDe: A Formal Framework for Heterogeneous Models of Computation

Axel Jantsch

Royal Institute of Technology

Sweden

ForSyDe Team:

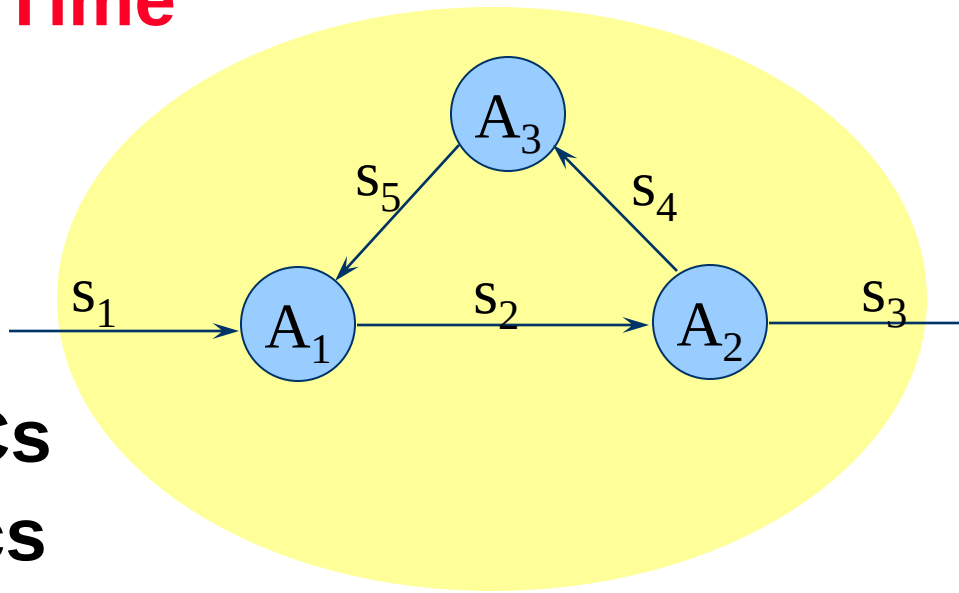
Ingo Sander, Hossein Niaki,

Axel Jantsch, Zhonghai Lu,

Tarvo Raudvere, Jun Zhu, Alfonso Acosta

Models of Computation

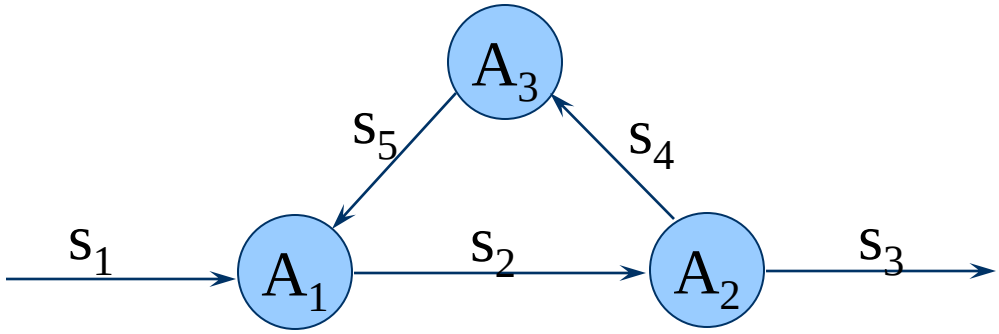
- Concurrent process networks
- Formal definition of Communication
- Formal definition of Synchronization
- Formal definition of Time
  - Untimed
  - Synchronous Time
  - Discrete Time
  - Continuous Time
- Heterogeneous MoCs
- Execution mechanics
- Purpose of a model: simulation, synthesis, verification, performance analysis



ForSyDe Objectives

- **Model time explicitly**
- **Allow for time at all abstraction levels**
- **Allow to mix different time models**
- **Model process invocation explicitly**
- **Abstract execution mechanics**

ForSyDe Features



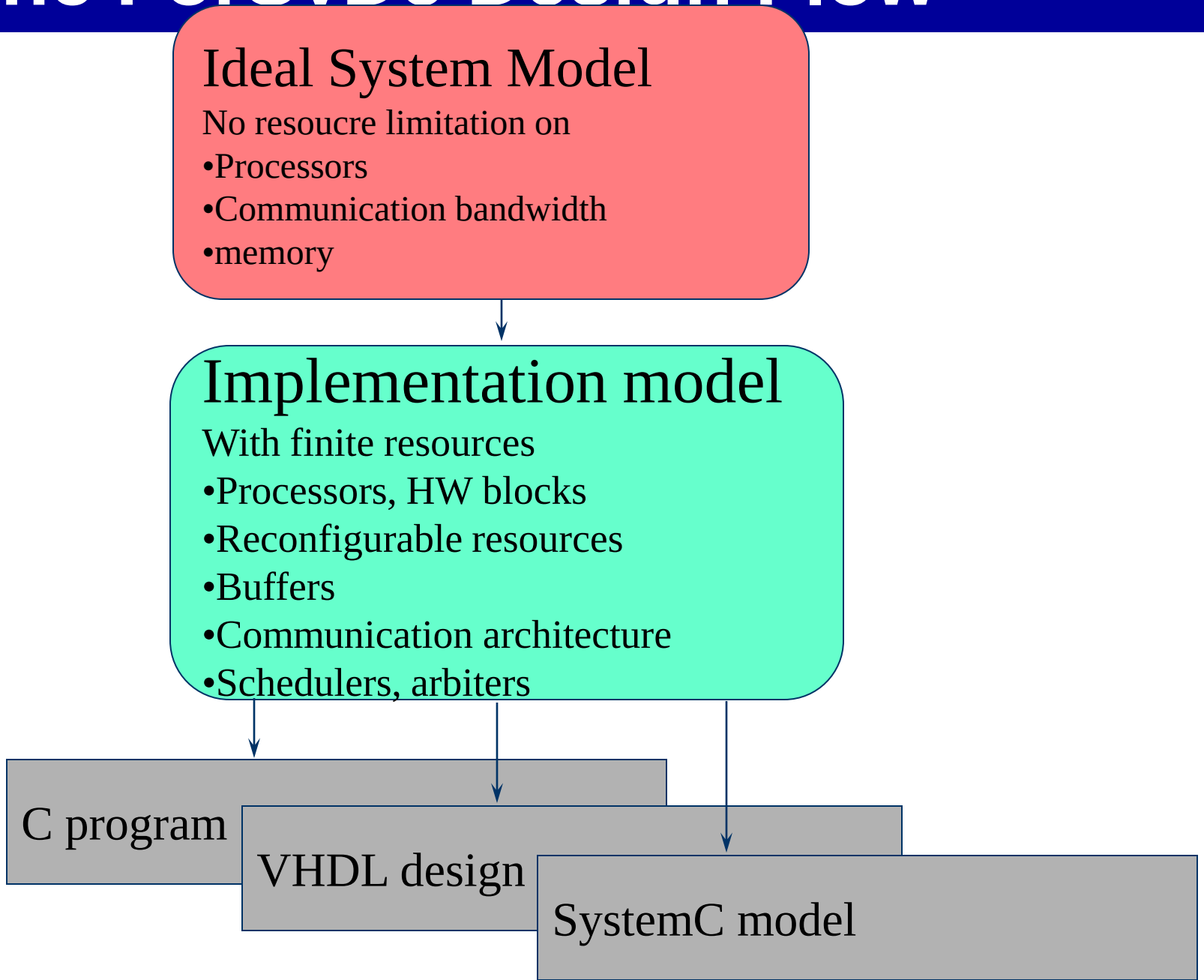
Processes

- Communicate through signals only;
- Functional
- State-full
- Blocking read
- Partition input and output signals
- Evaluate when required input is available

Signals

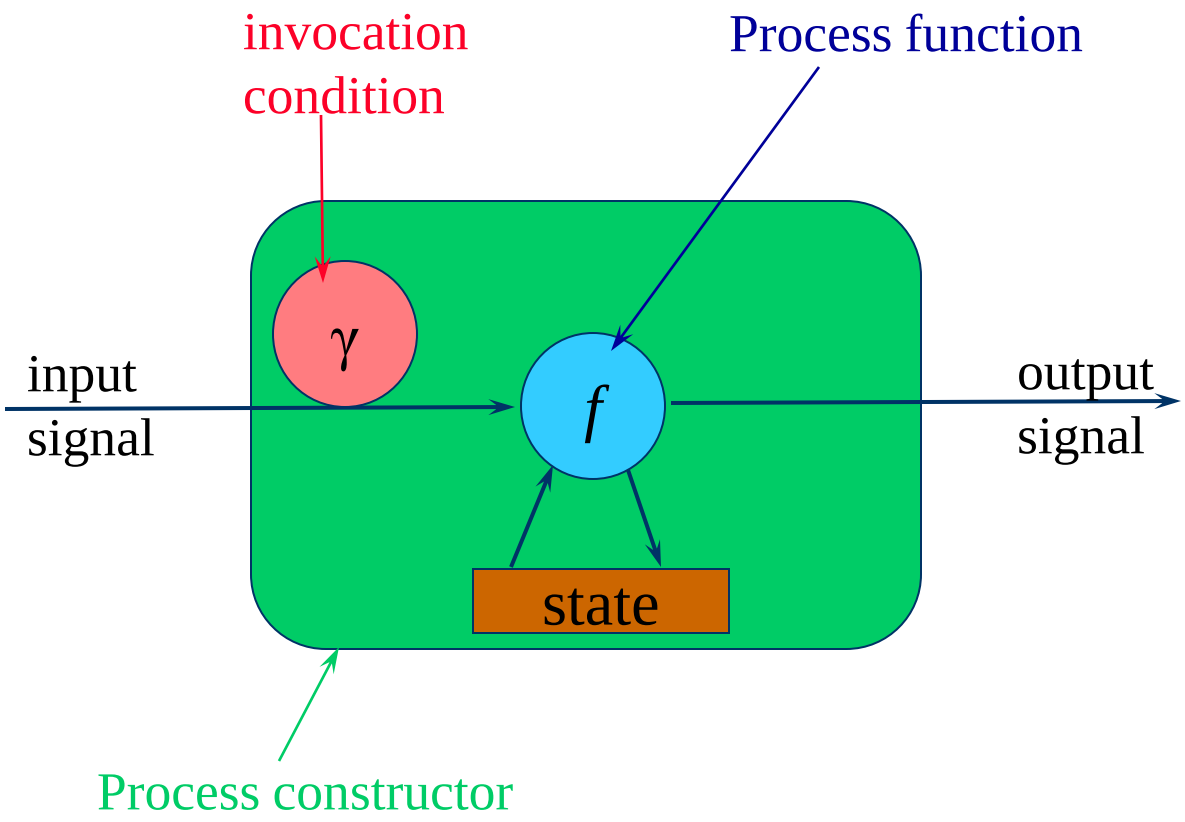
- Sequences of events
- Preserve event order
- Have one writer and multiple readers
- Untimed MoC: Events are partially ordered
- Discrete Time MoCs: Signals carry timing information
- Continuous Time MoC: Signals are functions

The ForSyDe Design Flow



ForSyDe Process Constructors

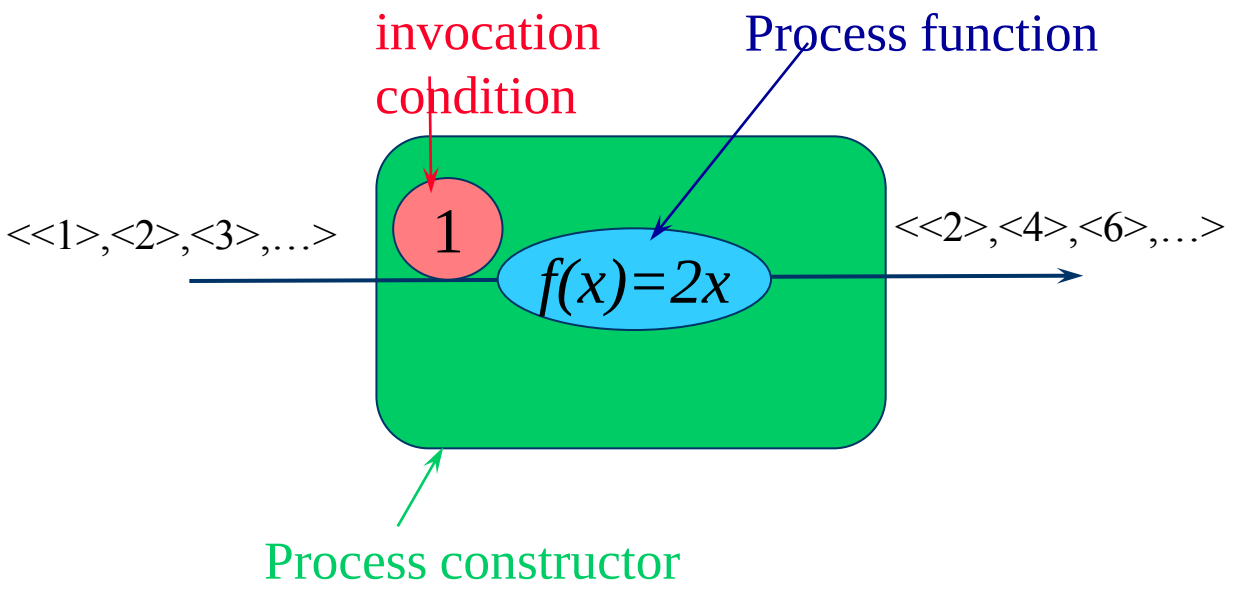
Process = constructor + function + initialState + invocationCondition



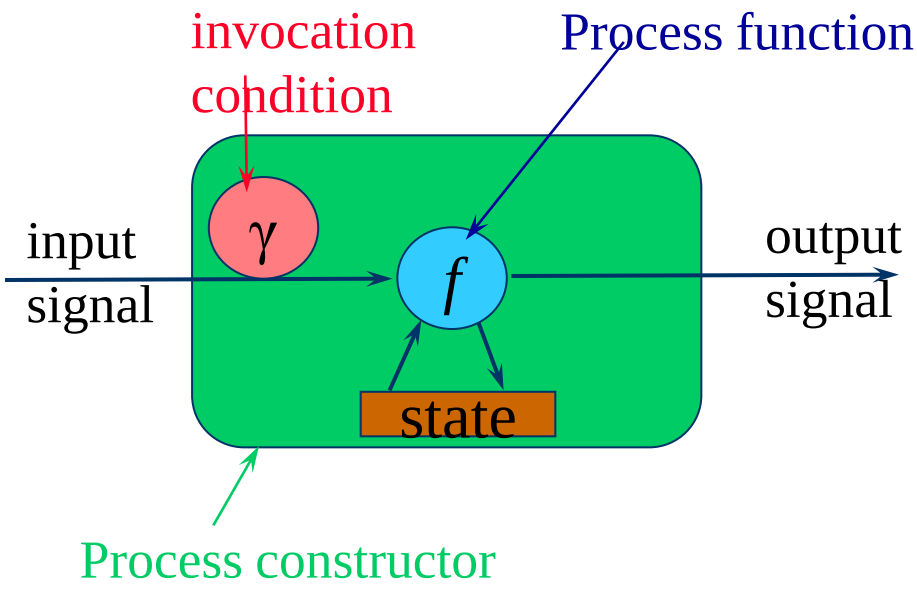
The *combU* Process Constructor

combU (*c*,*f*) = *p*
where *p*(*s*) = *s'*

combU (*c*,*f*) = *A*
where *c*=1
f(*x*) = 2*x*



Models of Computation



Process Constructors
- State-less
- Finite state machines
- Signal merge/split
- Source/sink

Process Combinators
- Sequential composition
- Parallel composition
- Feed-back composition

Models of Computation
- Untimed MoC (Dataflow, SDF)
- Synchronous MoC (Perfectly, Clocked)
- Discrete Time MoC
- Continuous Time MoC

Definition of a Model of Computation

The Untimed Model of Computation is defined as

U-MoC = (*C*,*O*), with

C* = { *combU, scanU, scandU, mealyU, mooreU,
zipU, unzipU, zipWithU, unzipU,
***sourceU, sinkU, initU* }**

***O*= { ||, o, FP_P }**

- **Synchronous MoC**
- **Clocked Synchronous MoC**
- **Discrete Time MoC**
- **Continuous MoC**

Time and Process Invocation In MoCs

- **Untimed MoC:**
  - **No explicit time, ordering of events**
  - **Invocation based on data availability**
- **Synchronous MoC:**
  - **Slot based time abstraction**
  - **Invocation in every slot**
- **Discrete Time MoC:**
  - **Physical, discrete time in seconds**
  - **Invocation based on data availability and progress of time**
- **Continuous Time MoC:**
  - **Physical, continuous time in seconds**
  - **Continuous invocation based on transfer functions**

The Integrated Model of Computation

The Integrated Model of Computation is defined as

HMoC = (*M,C,O*), with

***M* = { U-MoC, S-MoC, CS-MoC, T-MoC, CT-MoC}**

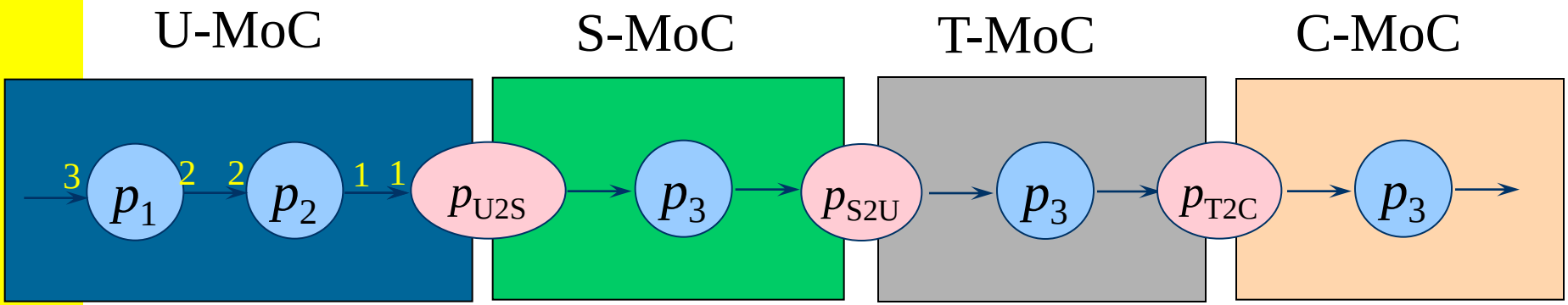
C* = { *intSup, intSdown, intTup, intTdown,
stripT2S, stripT2U, stripS2U,
insertS2T, insertU2T, insertU2S,
***a2dConverter, d2aConverter* }**

***O*= { ||, o, FP_P }**

MoC Interface Processes

Time Refinement

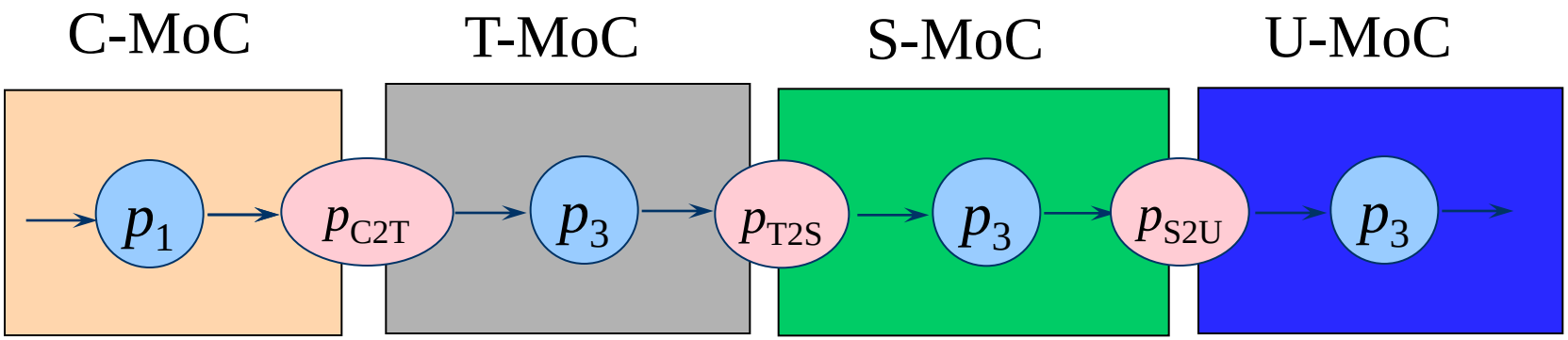
- **U-MoC → S-MoC**
- **U-MoC → T-MoC**
- **U-MoC → C-MoC**
- **S-MoC → T-MoC**
- **S-MoC → C-MoC**
- **T-MoC → C-MoC**



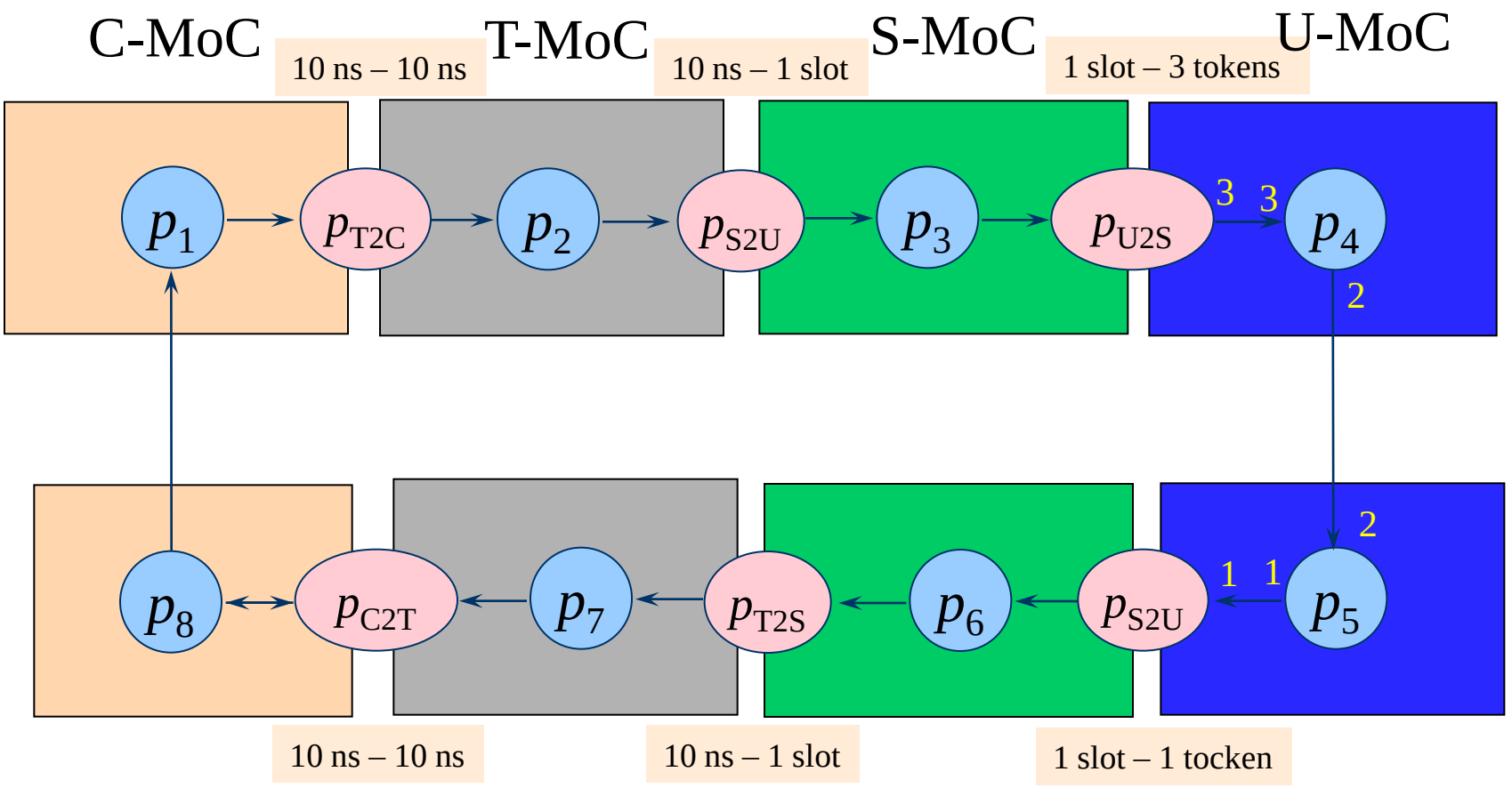
MoC Interface Processes 2

Time Abstraction

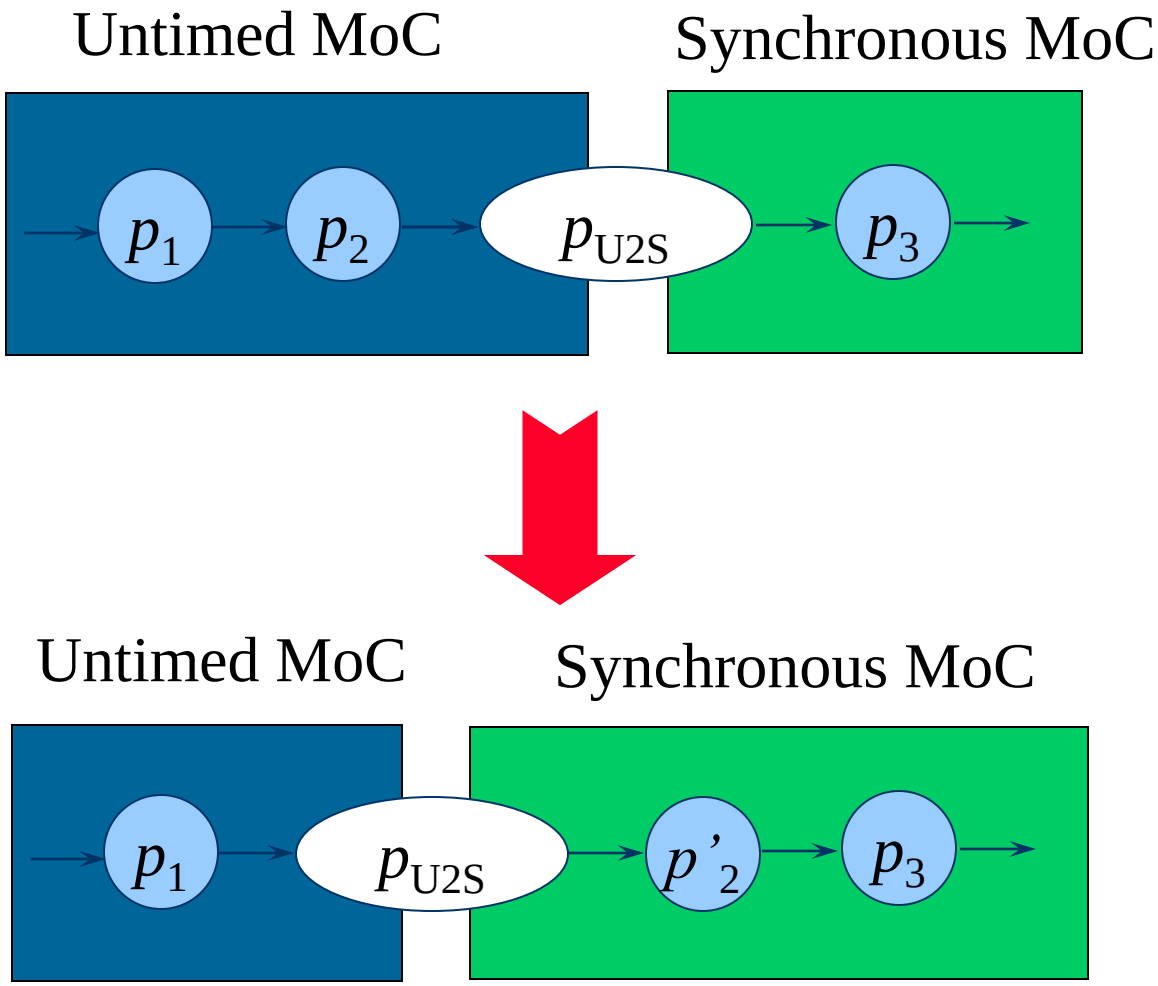
- C-MoC → T-MoC
- C-MoC → S-MoC
- C-MoC → U-MoC
- T-MoC → S-MoC
- T-MoC → U-MoC
- S-MoC → U-MoC



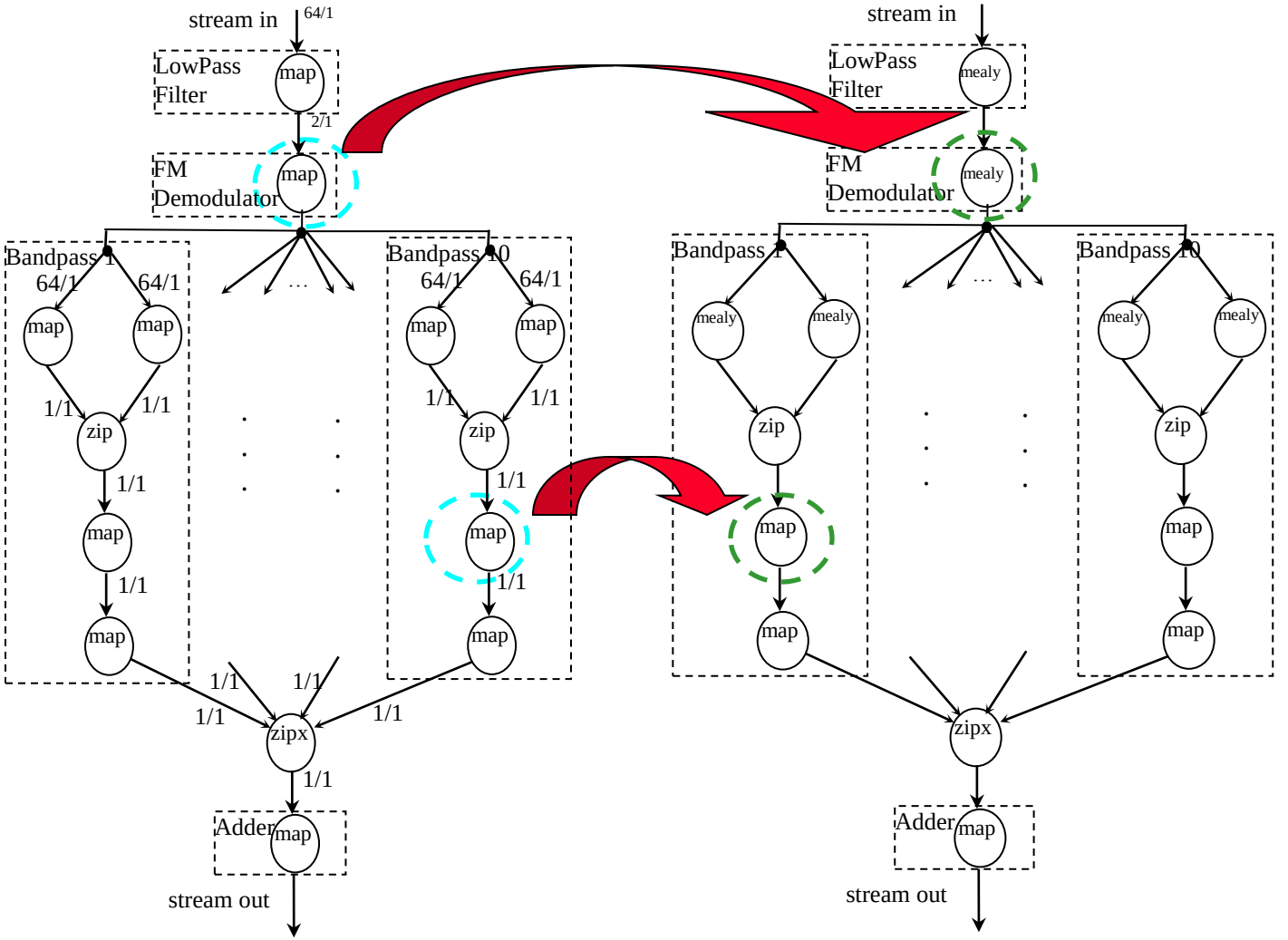
Heterogeneous MoCs



Process Migration



Cross Domain Process Refinement

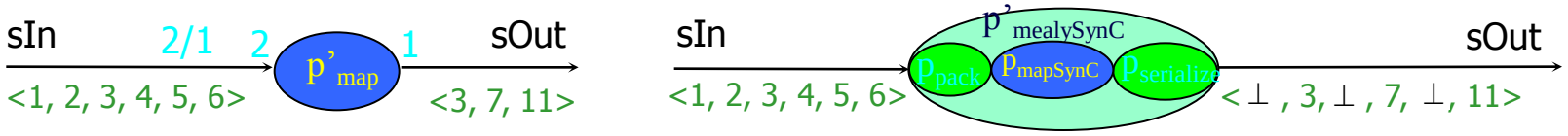


Cross Domain Process Refinement

SDF domain

Synchronous domain

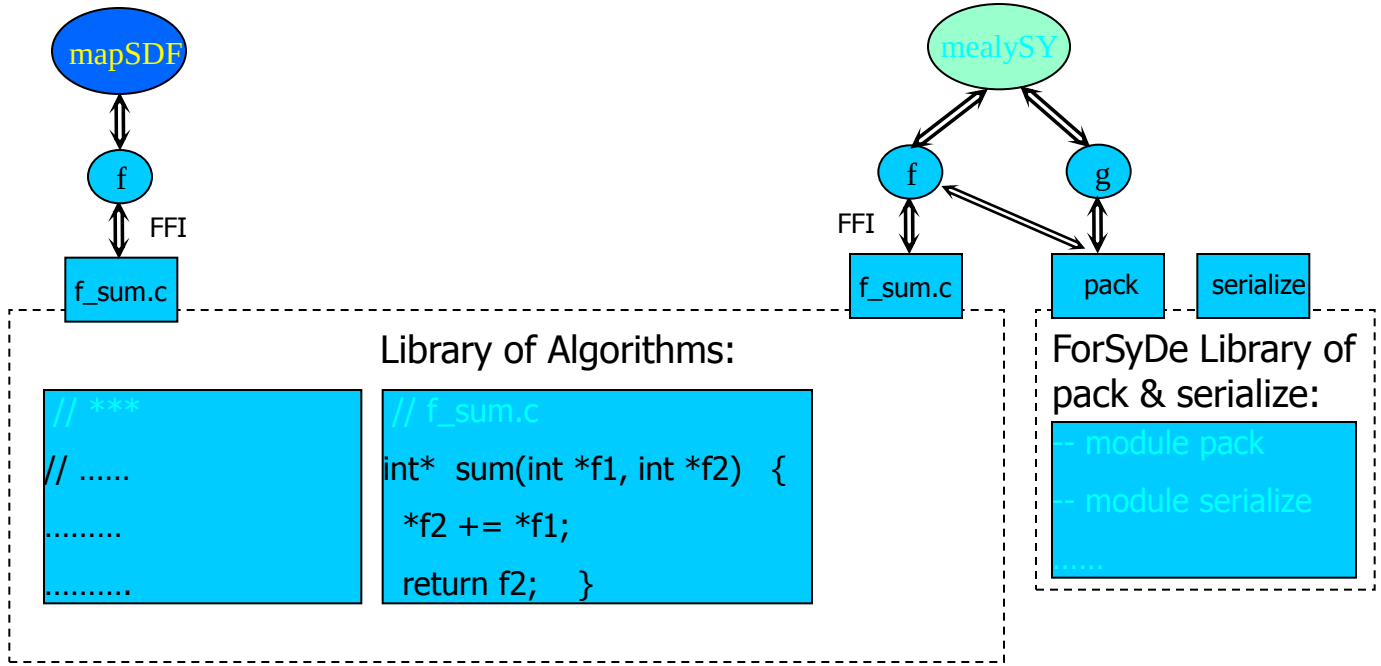
Communication layer:



mapSDF f_sum sIn = sOut

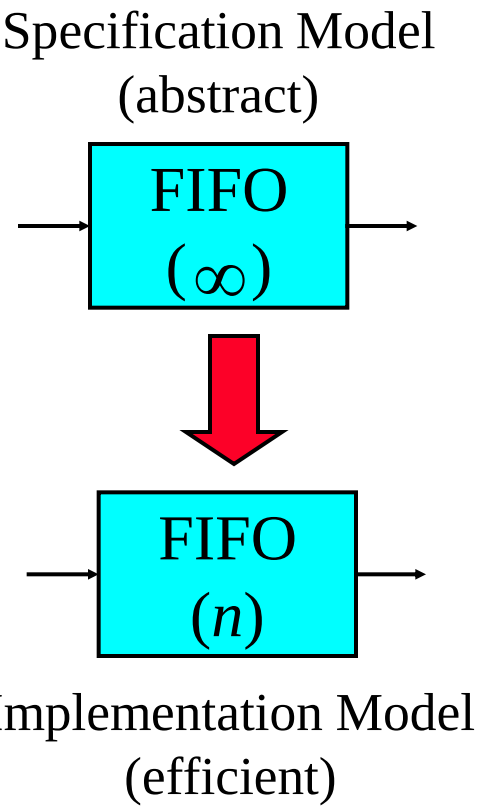
mealySY g f_sum sIn = sOut

Computation layer:



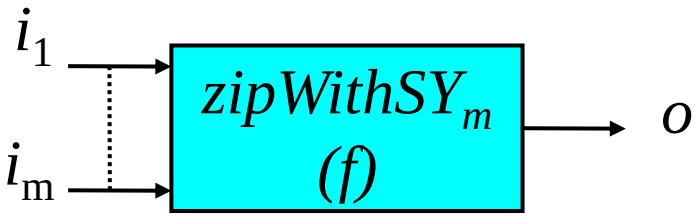
Design Decision Transformations

- **change the meaning of a model**
- **are needed to refine an abstract specification model into an efficient implementation**
- **imply a verification task for the designer**



Both models can have the same behavior, as long as the FIFO buffer will not overflow

Scheduling of Operations

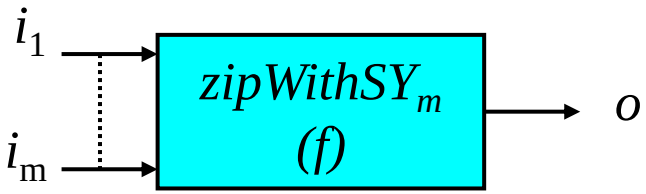


- **A combinational process with *m* input signals is modeled with *zipWithSY*$_m$ (*f*)**
- **In each event cycle the function *f* is applied to the current values of the input signals**
- **A large amount of computational resources may be required for these processes**

Scheduling of operations in time leads to a smaller amount of computational resources (High-Level Synthesis)

Scheduling of Operations

Combinational process

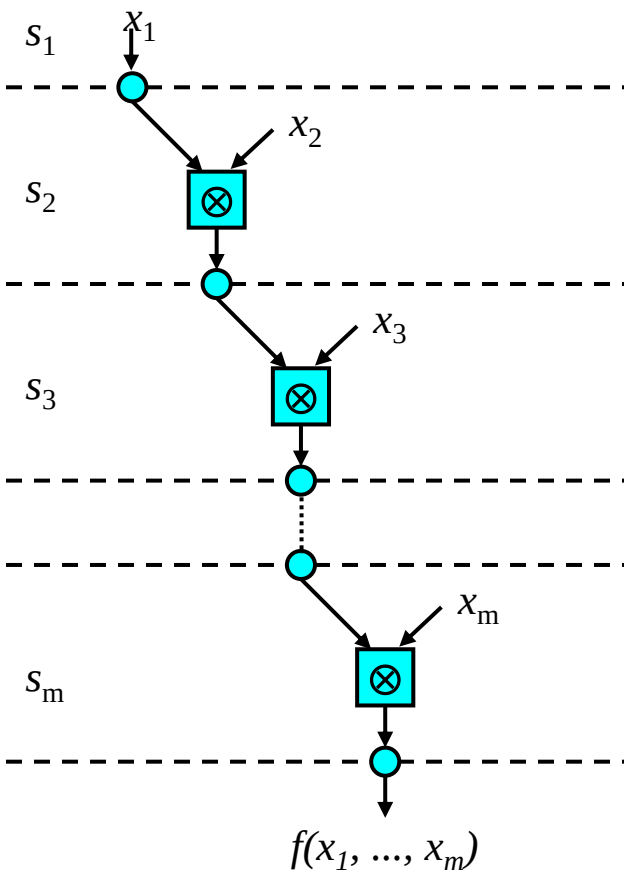


If

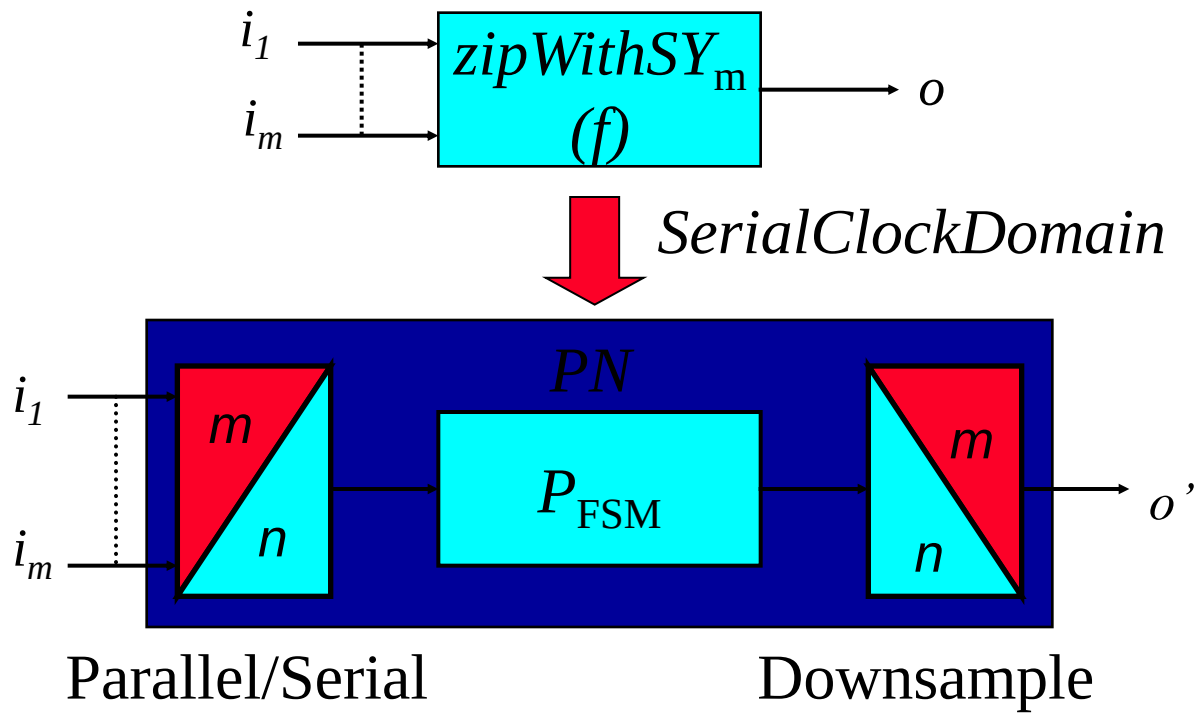
$$f(x_1, ..., x_m) = x_1 \otimes x_2 \otimes ... \otimes x_m$$

the following schedule using only one computational unit can be derived:

Schedule

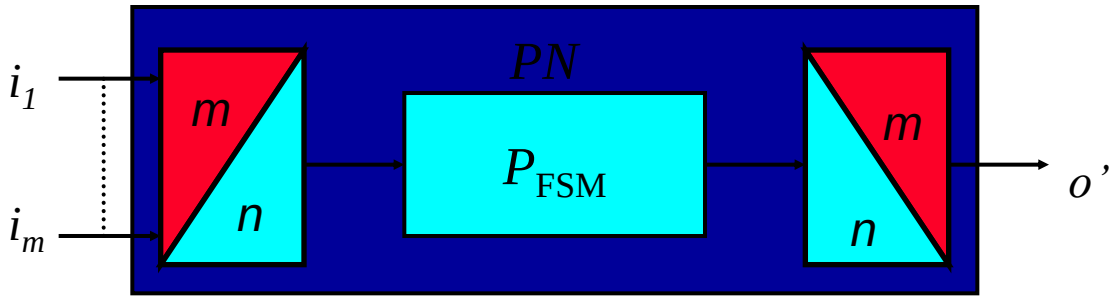


The design decision transformation *SerialClockDomain*

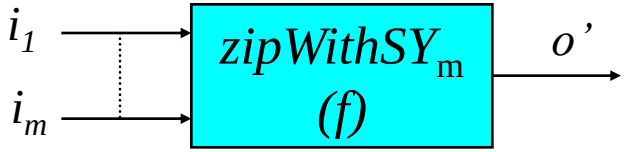


The process P_{FSM} implements the scheduled version of the function *f* and is based on a finite state machine process constructor.

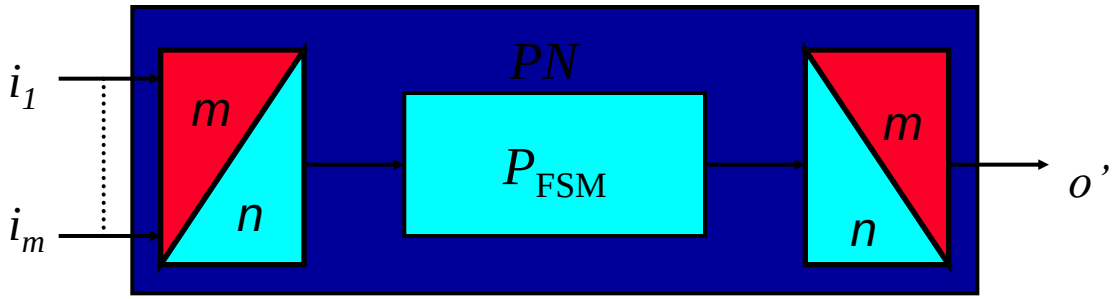
The design decision transformation *SerialClockDomain*



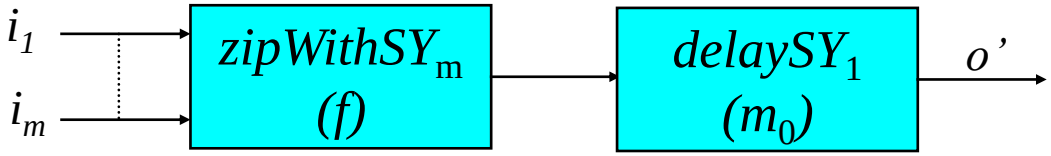
$\neq$



The design decision transformation *SerialClockDomain*

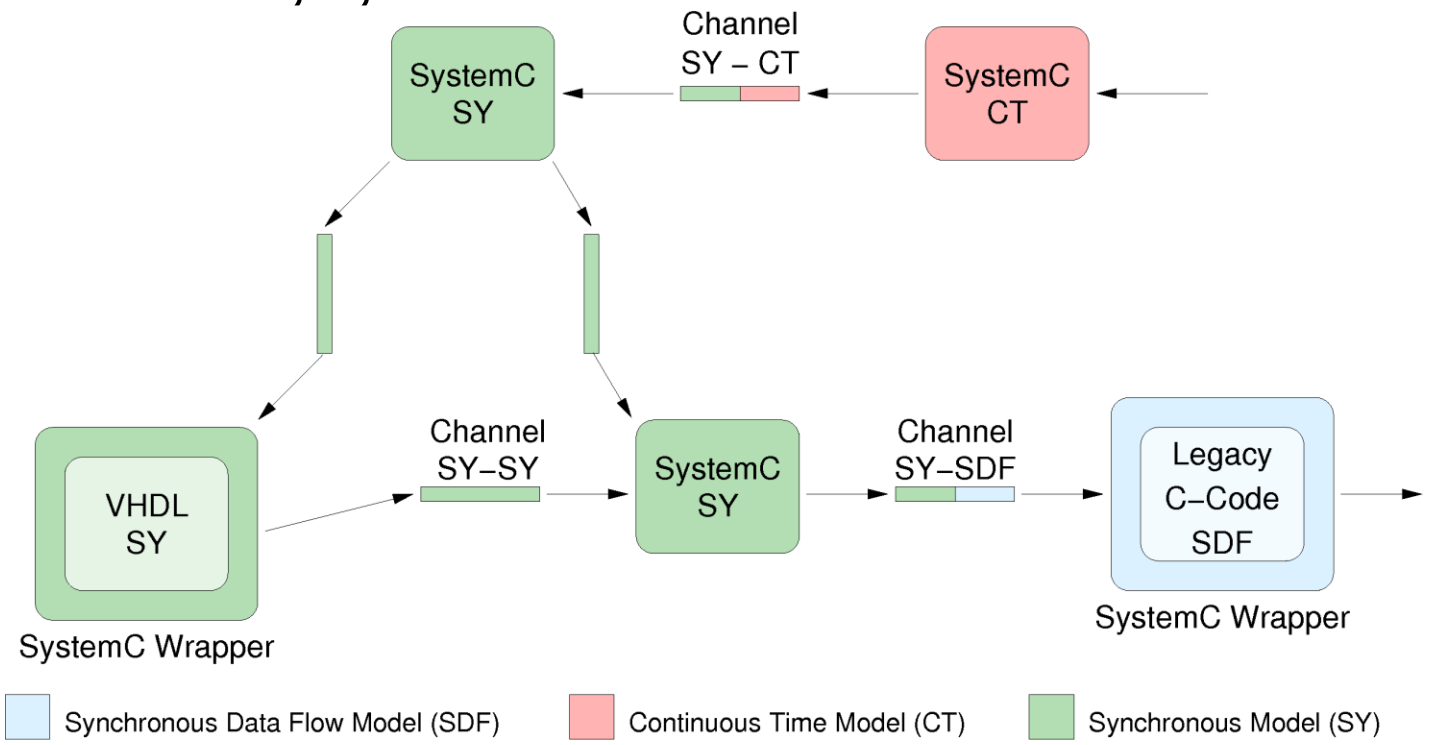


=

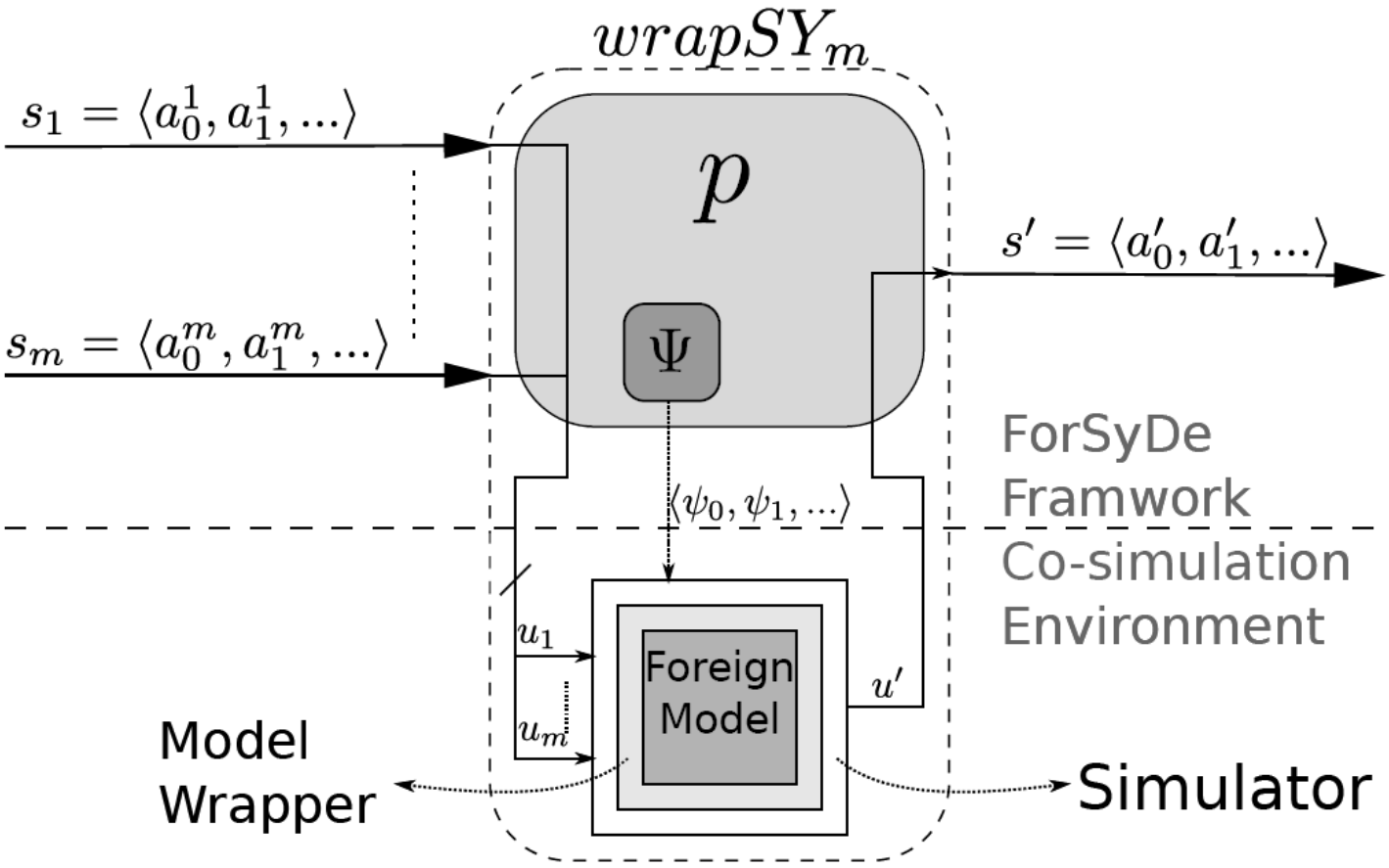


Integration of Existing Models

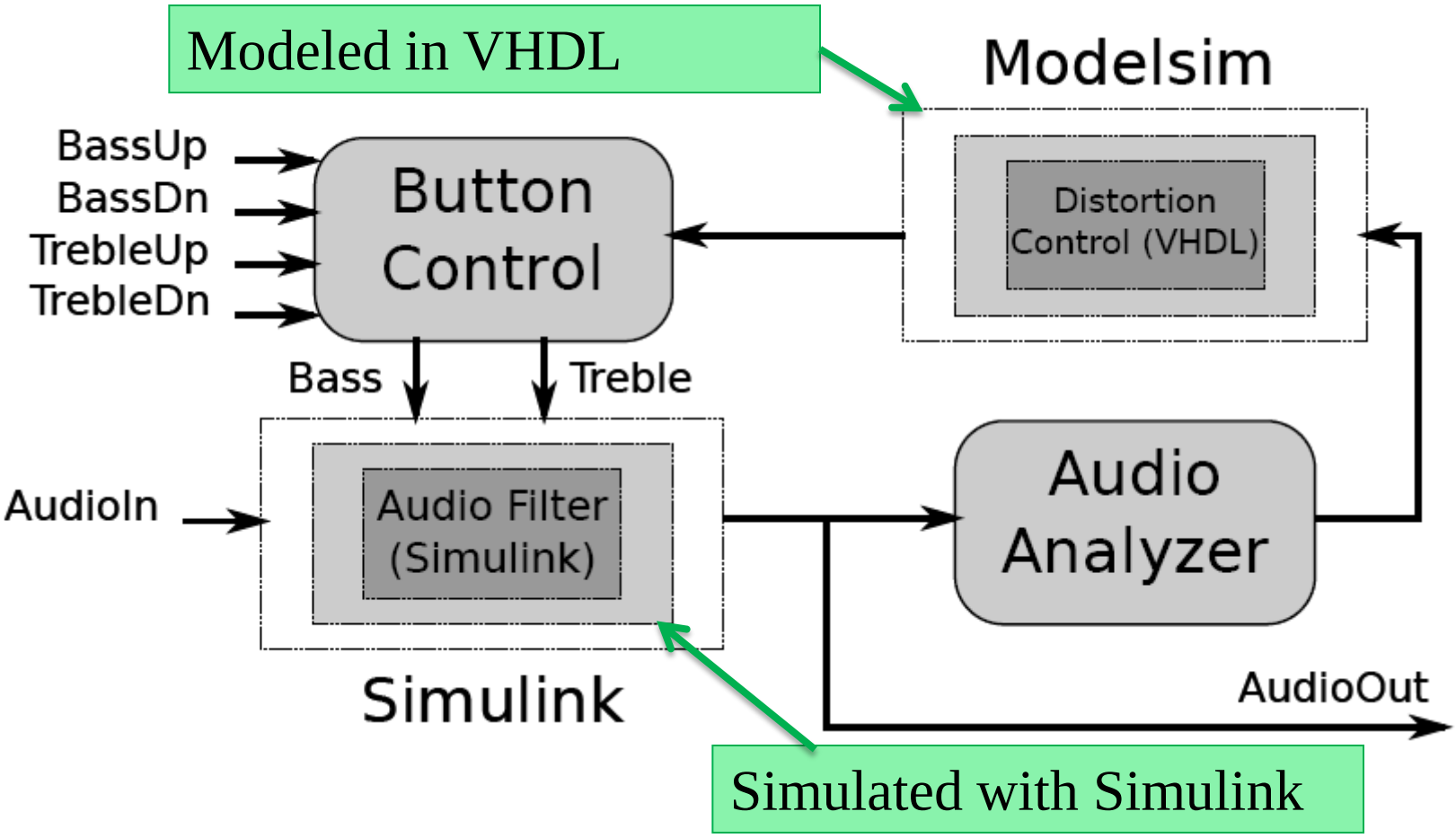
- **Users want to reuse existing models in other design languages**
- **SystemC-wrappers integrate “legacy code”**
  - **Matlab, C, VHDL**



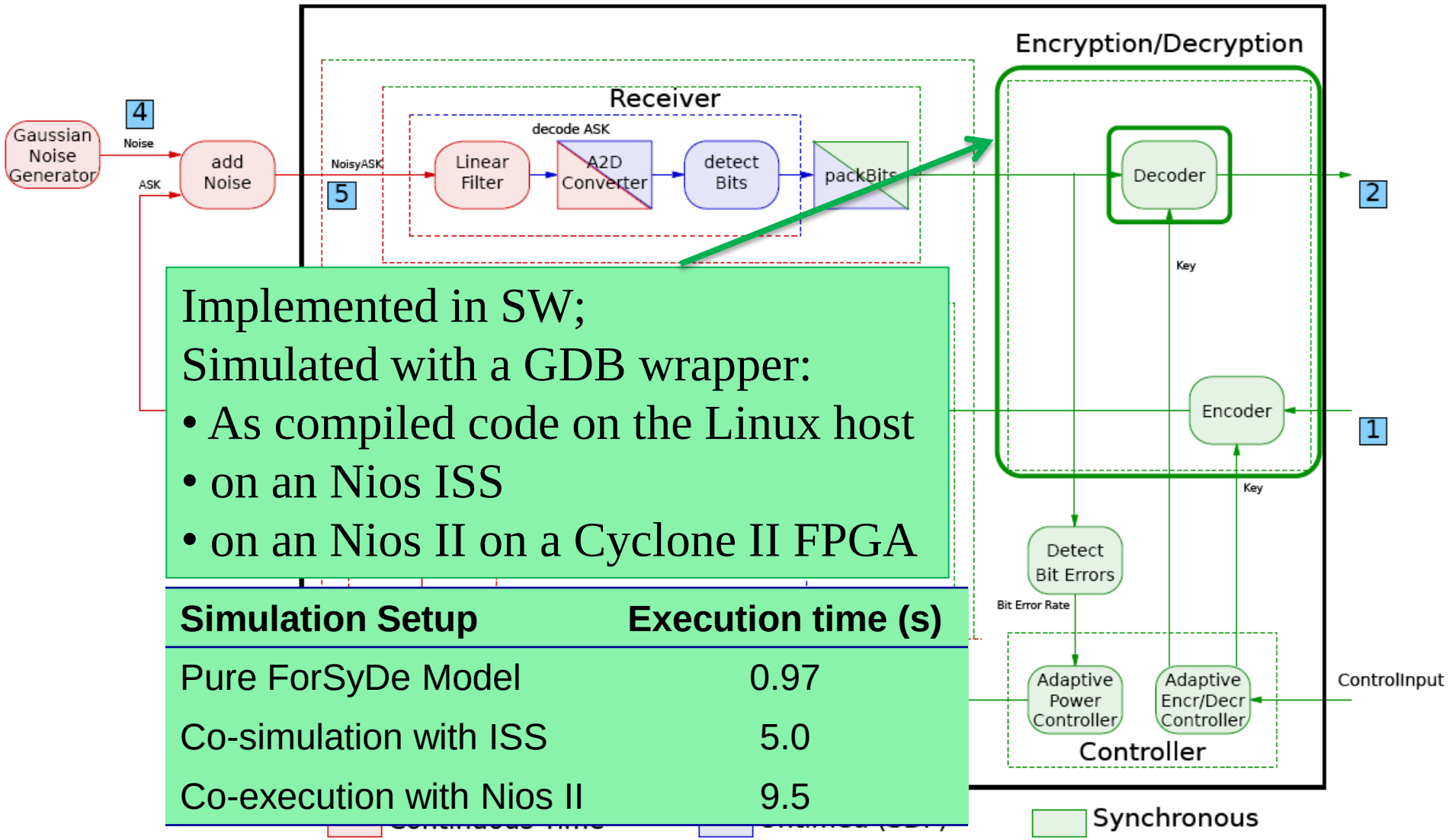
Model Wrapper



Cosimulation in ForSyDe



Cosimulation in ForSyDe



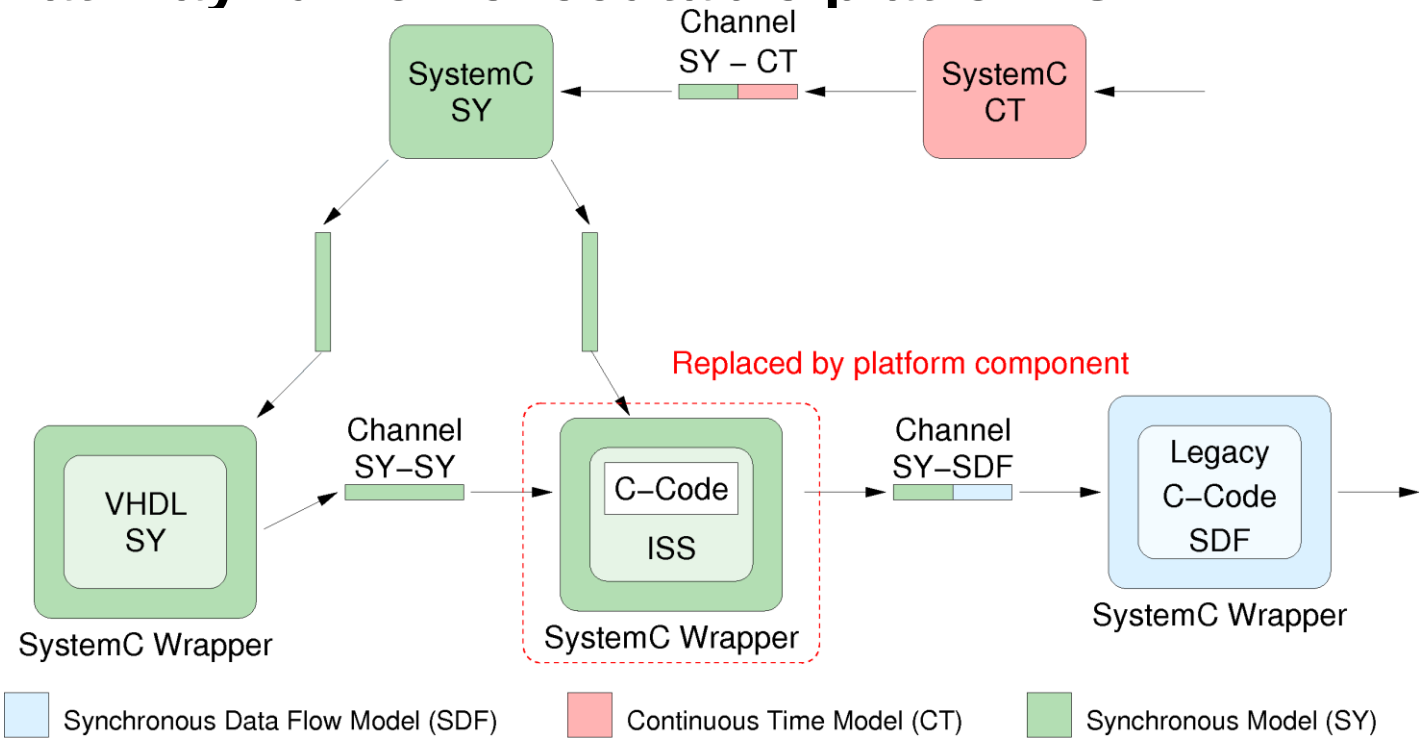
Implemented in SW;
Simulated with a GDB wrapper:

- As compiled code on the Linux host
- on an Nios ISS
- on an Nios II on a Cyclone II FPGA

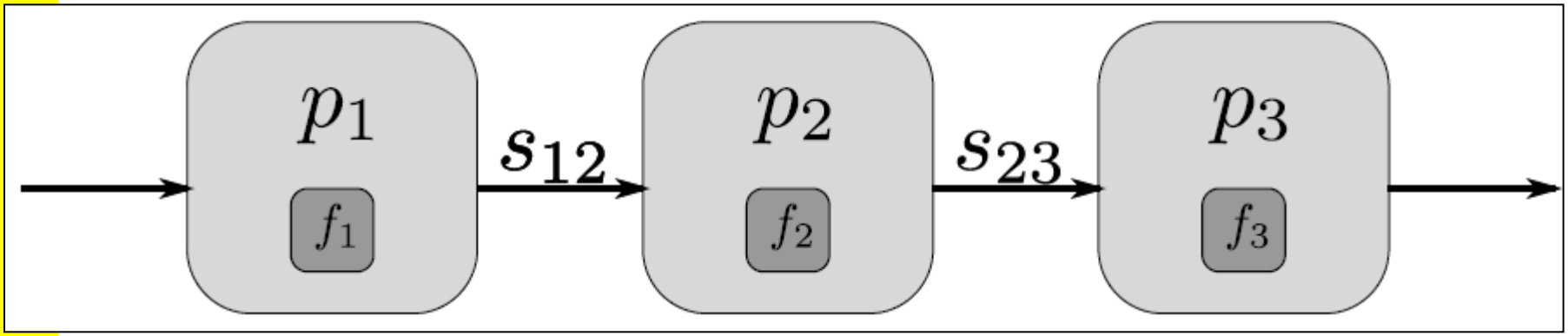
Simulation Setup	Execution time (s)
Pure ForSyDe Model	0.97
Co-simulation with ISS	5.0
Co-execution with Nios II	9.5

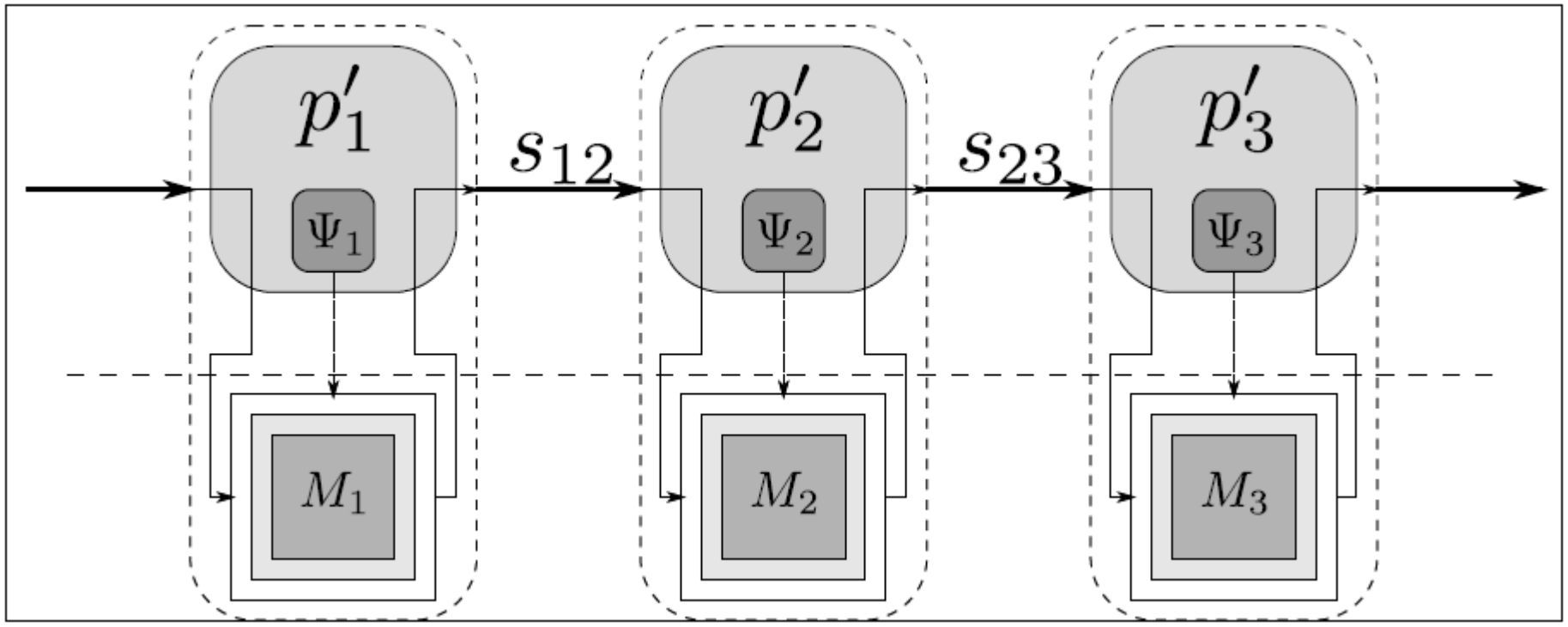
Refinement-by-Replacement

- **SYSMODEL supports refinement-by-replacement approach**
- **High-level models can be replaced by low-level code that may run on executable platforms**

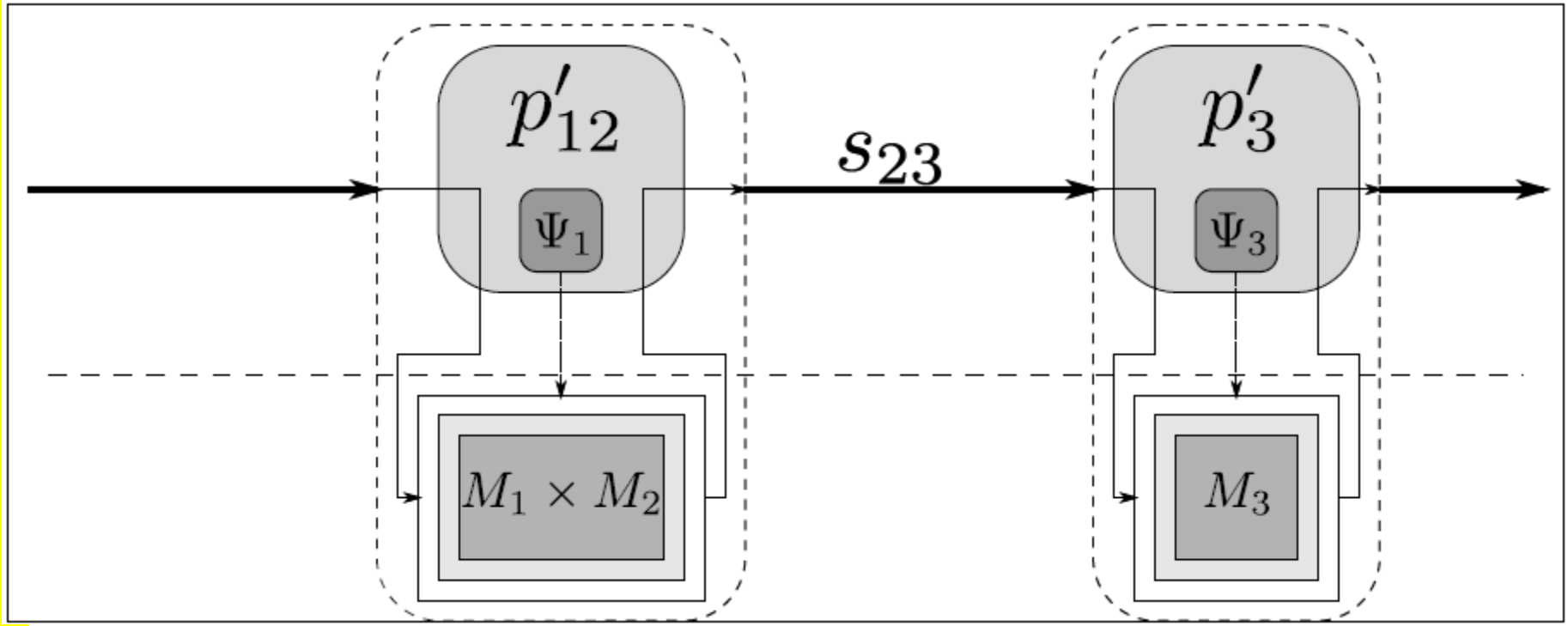


Refinement by Replacement

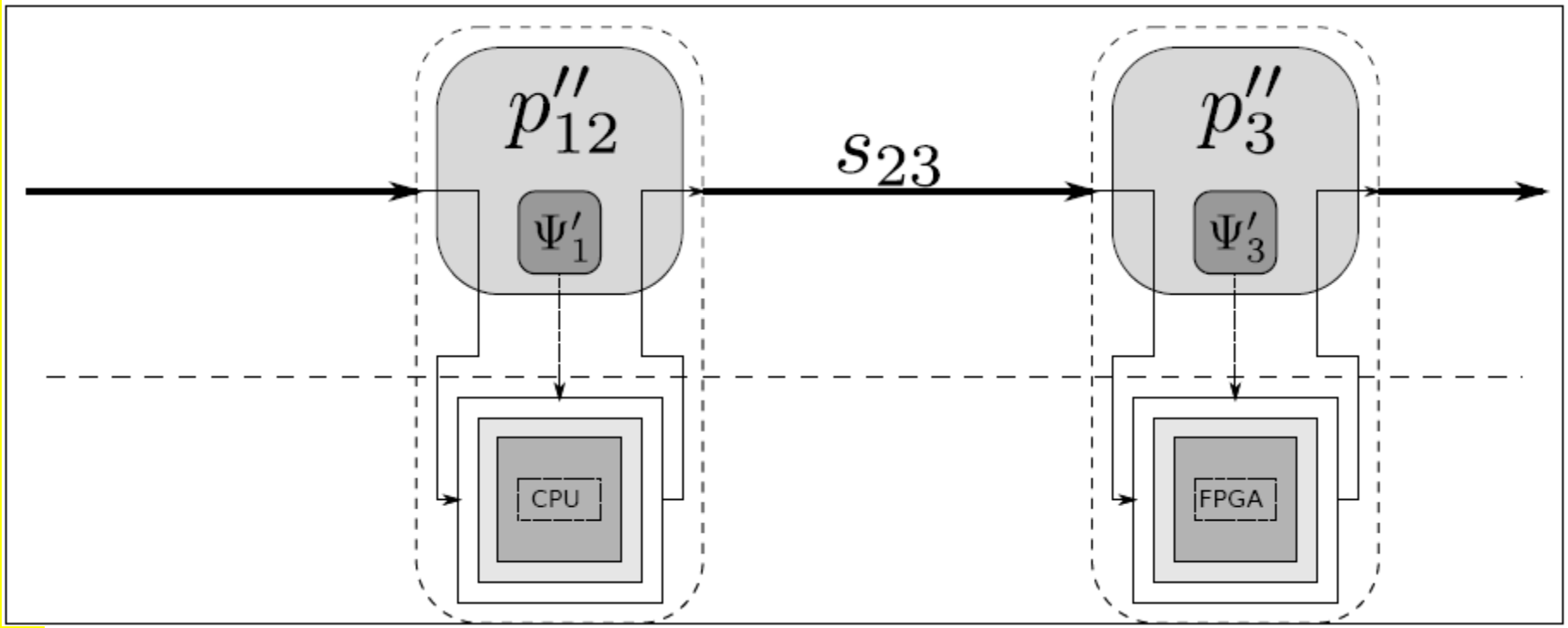




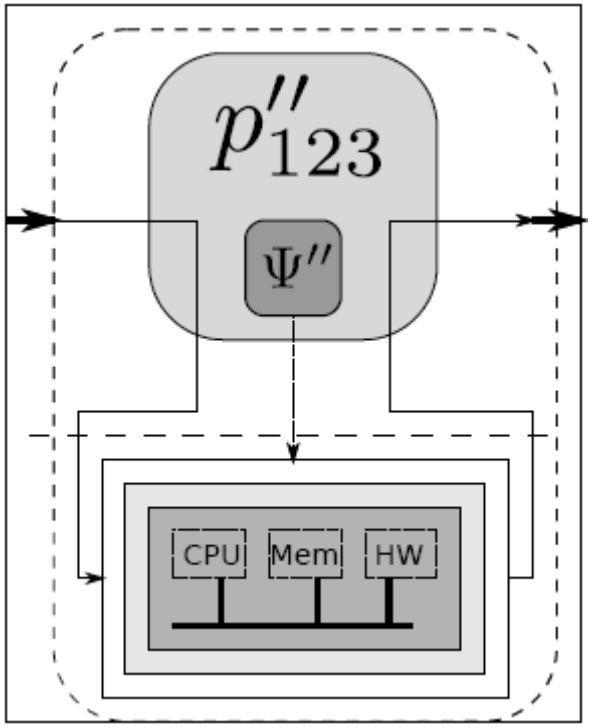
Merging of processes

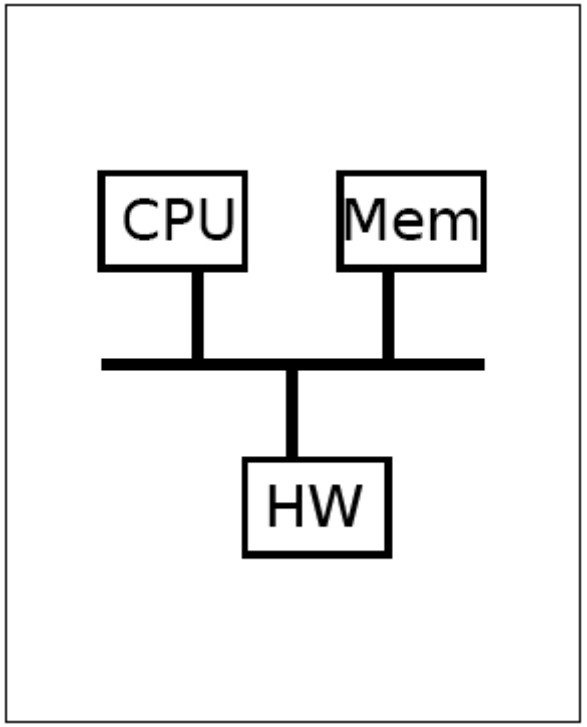


Platform Instantiation

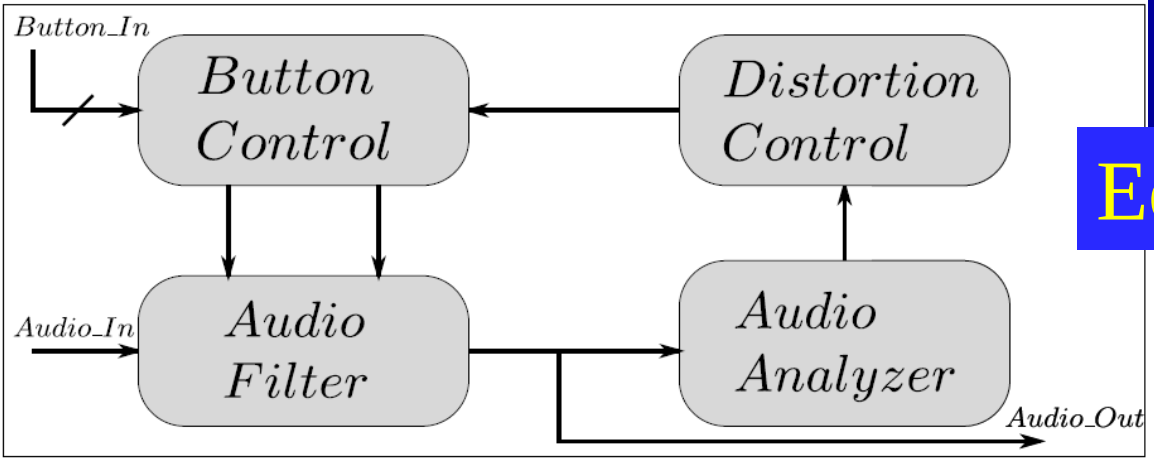


Communication Synthesis and Extraction

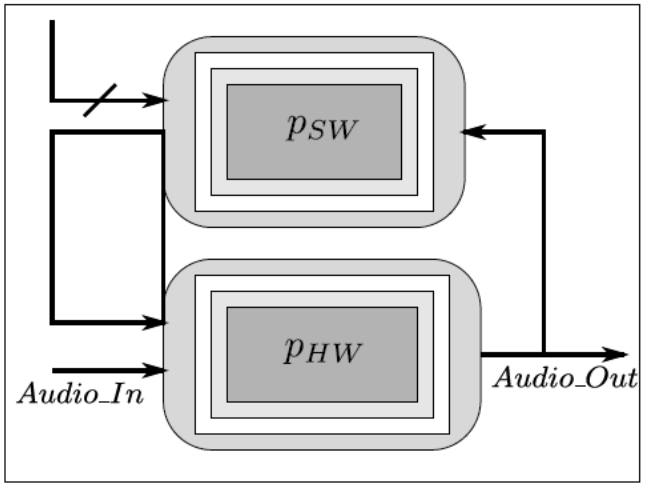




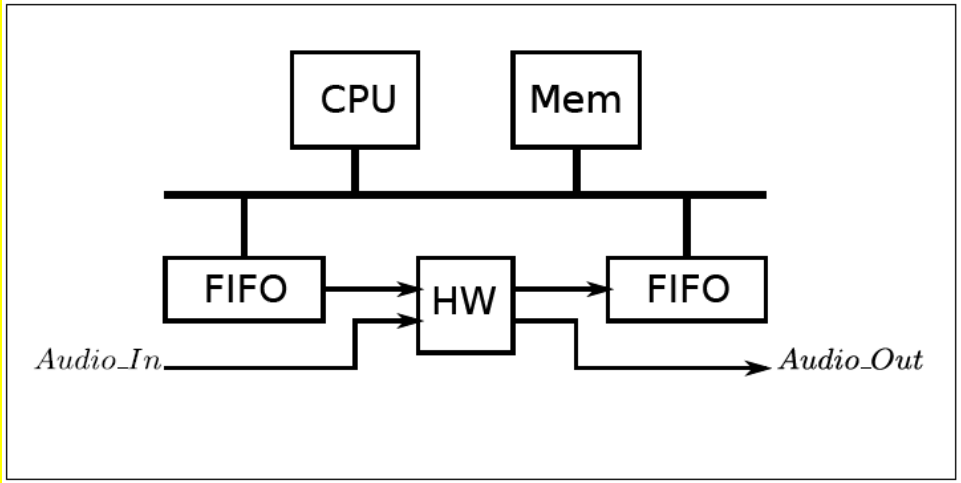
Equalizer Case Study



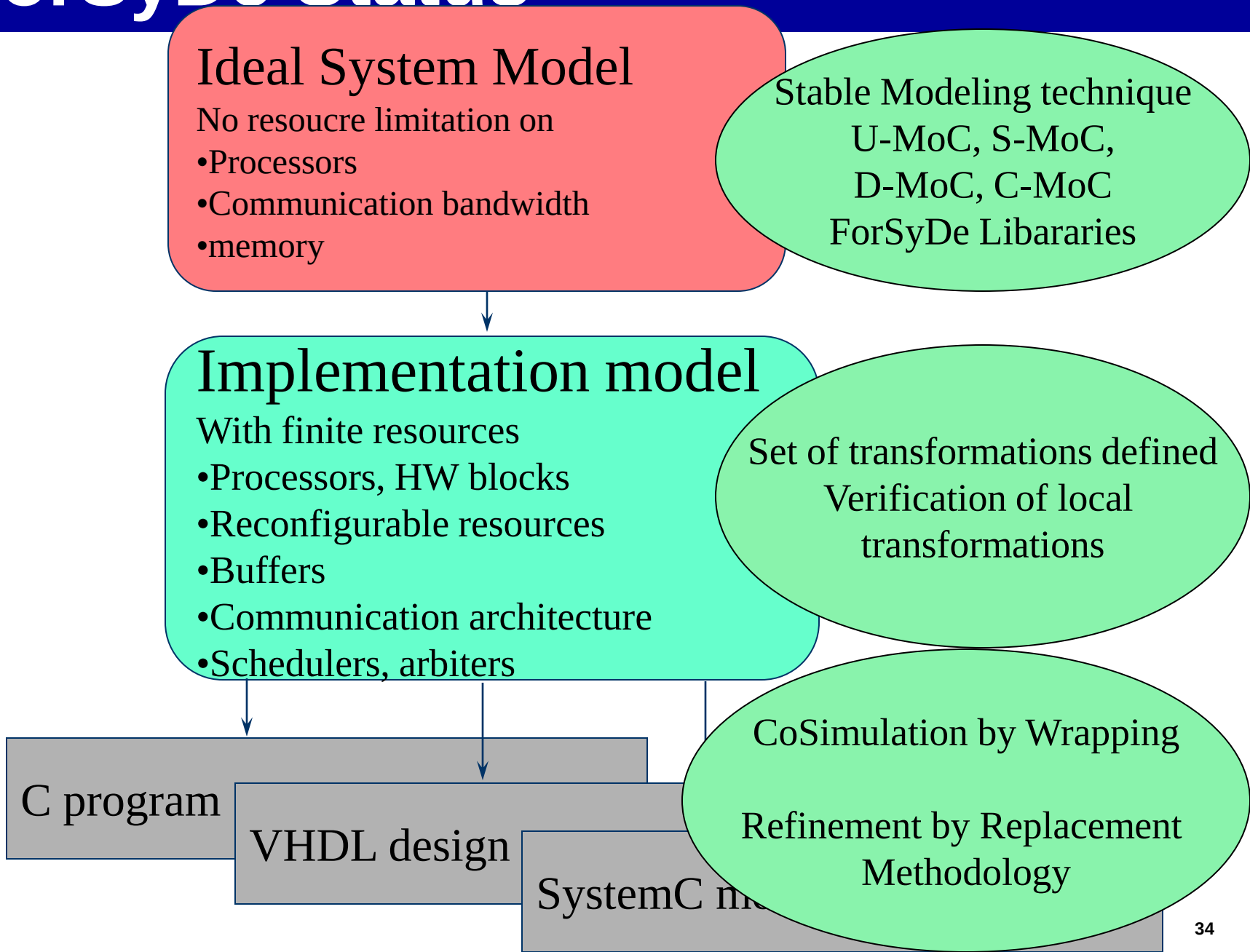
Replacement
Merge



Platform Instatiation
Communication Synthesis
Extraction



ForSyDe Status



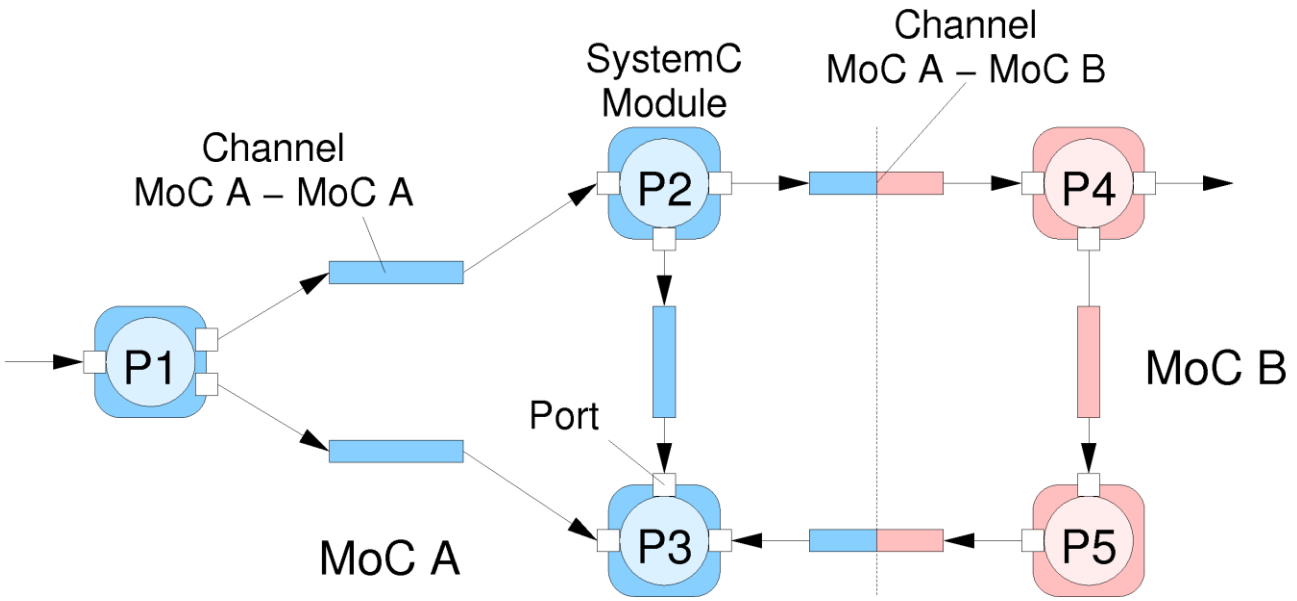
Selected References

- http://www.ict.kth.se/forsyde/
- Seyed Hosein Attarzadeh Niaki, Ingo Sander, "Semi-Formal Refinement of Heterogeneous Embedded Systems by Foreign Model Integration", Proceedings of the Forum on Design Languages (FDL), Oldenbour, Germany, September 2011.
- Seyed Hosein Attarzadeh Niaki and Ingo Sander, "Co-simulation of embedded systems in a heterogeneous MoC-based modeling framework", 6th IEEE International Symposium on Industrial Embedded Systems (SIES), IEEE,, pp. 238--247, June 2011.
- Jun Zhu, Ingo Sander, and Axel Jantsch, "Performance analysis of reconfigurations in adaptive real-time streaming applications", ACM Transactions in Embedded Computing Systems, 2010.
- Tarvo Raudvere, Ingo Sander, and Axel Jantsch. Application and verification of local non-semantic-preserving transformations in system design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1091-1103, June 2008.
- Deepak Mathaikutty, Hiren Patel, Sandeep Shukla, and Axel Jantsch. SML-Sys: A functional framework for multiple models of computation for heterogeneous system design. *Design Automation for Embedded Systems*, 2008.
- Ingo Sander and Axel Jantsch. Modelling adaptive systems in ForSyDe. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 200(2):39-54, 2008.
- Axel Jantsch and Ingo Sander, "Models of Computation in the Design Process", SoC: Next Generation Electronics, IEE, edited by Bashir M Al-Hashimi, Invited contribution, 2005.
- Axel Jantsch and Ingo Sander, "Models of Computation and languages for embedded system design", IEE Proceedings on Computers and Digital Techniques, vol. 152, pp. 114-129, no. 2, Special issue on Embedded Microelectronic Systems; Invited paper, March 2005.
- Ingo Sander, Axel Jantsch, and Zhonghai Lu, "Development and Application of Design Transformations in ForSyDe", IEE Proceedings on Computers and Digital Technique, vol. 150, pp. 313-320, no. 5, September 2003.
- Ingo Sander and Axel Jantsch, "System Modeling and Transformational Design Refinement in ForSyDe", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, pp. 17-32, no. 1, January 2004.

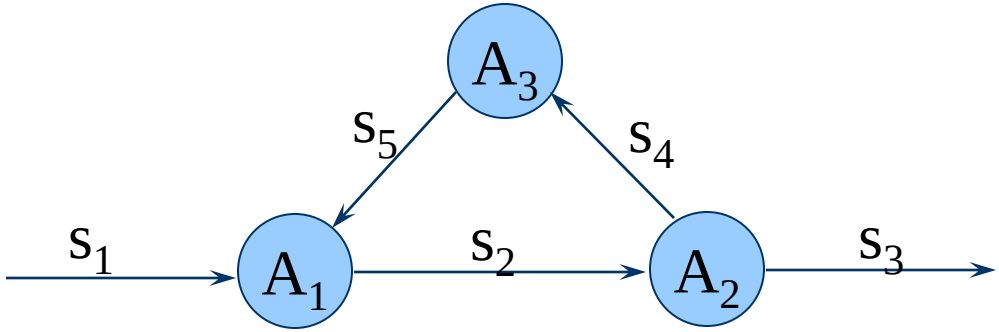
Questions ?

ForSyDe compliant SystemC

- **Project develops SystemC libraries that**
  - **are based on the formal foundations of ForSyDe**
    - **Concept of process constructor**
    - **Well-defined execution semantics**
- **Project develops modeling guidelines**



ForSyDe MoCs and SystemC




```
SCA_SDF_MODULE(A1)
{
  sca_sdf_in<int> s1, s5;
  sca_sdf_out<int> s2;
  void sca_sig_proc();
  SCA_SDF_CTOR();
}
```

```
SCA_SDF_MODULE(A2)
{
  sca_sdf_in<int> s2;
  sca_sdf_out<int> s3, s4;
  void sca_sig_proc();
  SCA_SDF_CTOR();
}
```

```
SCA_SDF_MODULE(A3)
{
  sca_sdf_in<int> s4;
  sca_sdf_out<int> s5;
  void sca_sig_proc();
  SCA_SDF_CTOR();
}
```

The GDB Wrapper

Wraps compiled software

- C projects compiled with GCC

Uses GNU's Debugger to

- Communicate data
- Execute and synchronize (explicitly)

Can wrap

- Software compiled on host
- Software simulated in an ISS
- Software executing on processor

loop

$inps \Leftarrow$ {set by GDB for m inputs}

$out \leftarrow f(inps)$

$out \Rightarrow$ {read by GDB}

end loop

$\Psi_{sw} = \langle \psi_{sw}, \psi_{sw}, \ldots \rangle$

$\psi_{sw} = \langle$set,continue,read,continue$\rangle$

The HDL Wrapper

Wraps synchronous hardware

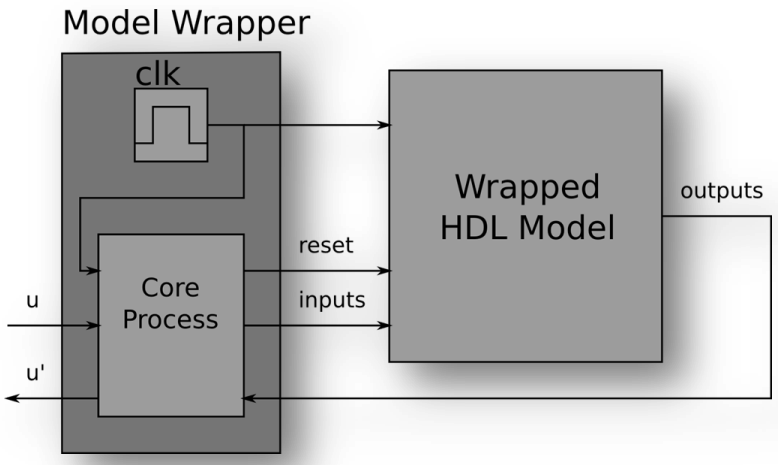
- Synthesizable subset of an HDL

Uses FIFO-like Unix pipes to

- Communicate data
- Implicitly synchronize with blocking semantics

Explicit clock in model wrapper

- Complies to implicit clock of MoC
- Consistent with other wrapper clocks



The HDL Wrapper

```
reset ← active
wait until rising_edge(clk)
reset ← deactive
loop
    inps ← read(ipipe) {for m inputs}
    if inps = Ø then {end of simulation}
        exit loop
    end if
    model inputs ← inps
    wait until rising_edge(clk)
    out ← model outputs {packed to a single output}
    write(opipe, outs)
end loop
```

The Simulink Wrapper

Wraps discrete-time Simulink models

- Input and output must be sampled with the same rate

Uses FIFO-like Unix pipes to

- Communicate data
- Implicitly synchronize with blocking semantics

The model wrapper is

- An input block connected to all system inputs
- An output block connected to all system outputs