Formal support for Untimed SystemC Specifications: Application to High-Level Synthesis

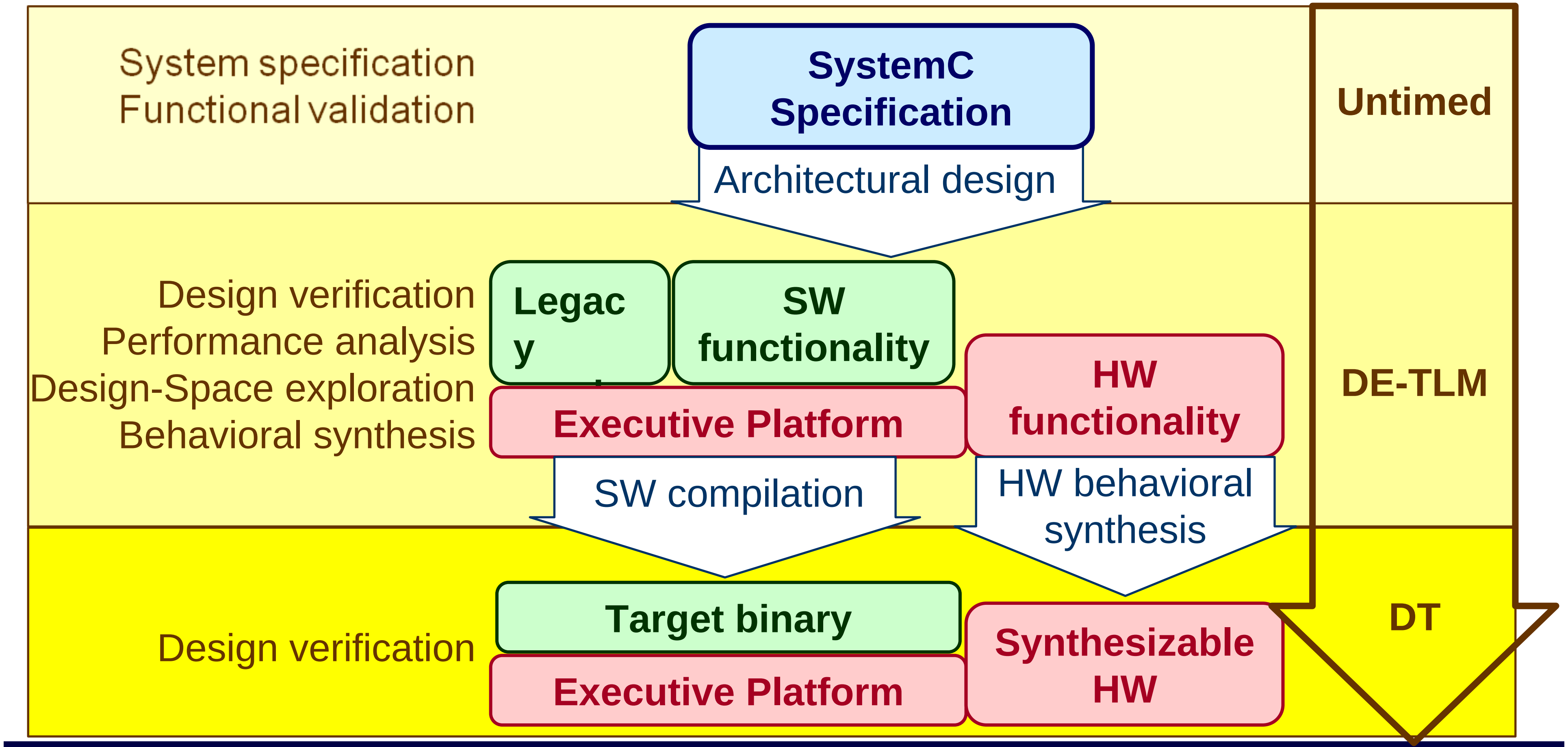
Eugenio Villar
Fernando Herrera
Víctor Fernández

Index

- Motivation
- Introduction
- Formal Framework
  - – Application to High-Level Synthesis Verification
- Conclusions

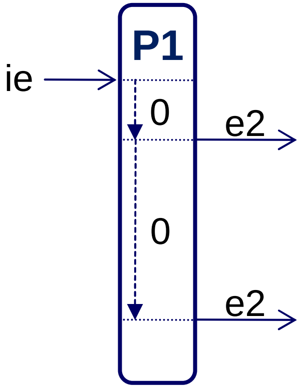
Motivation

- SystemC applications



Motivation

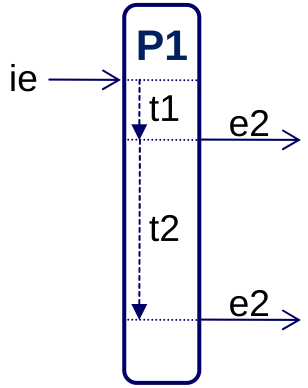
- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design



System simulation

Motivation

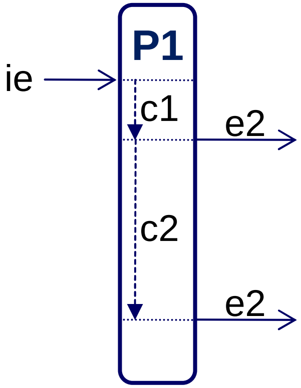
- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design



TLM simulation

Motivation

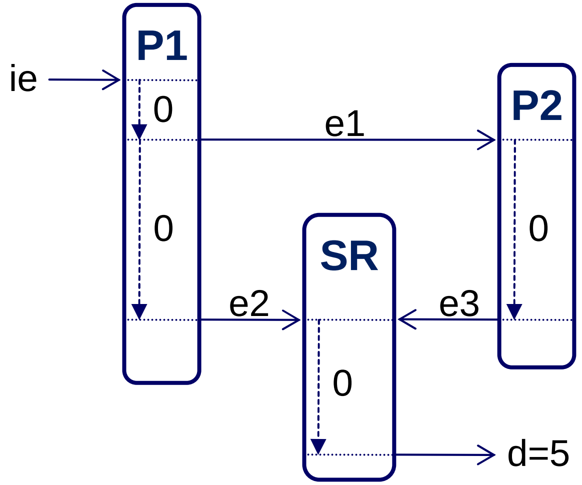
- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design



Cycle simulation

Motivation

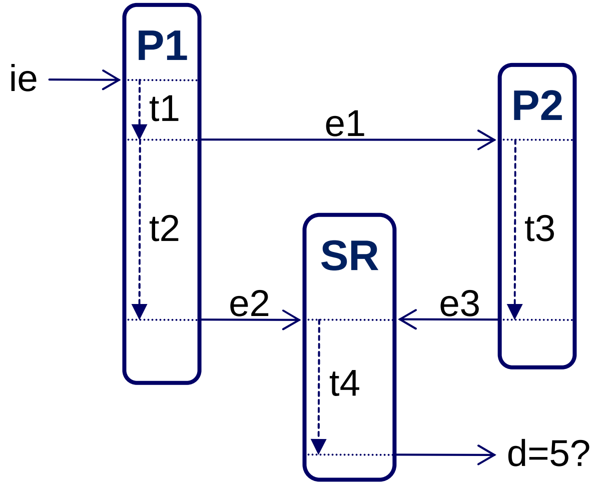
- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design



System simulation

Motivation

- Timing transformations
  - – Source of design errors
  - – Classical problem in concurrent programming
  - – Not enough addressed in SystemC (HW/SW) design

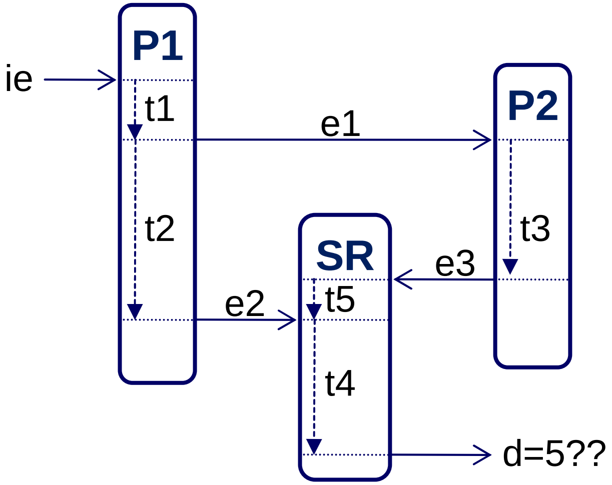


System simulation

TLM simulation

Motivation

- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design

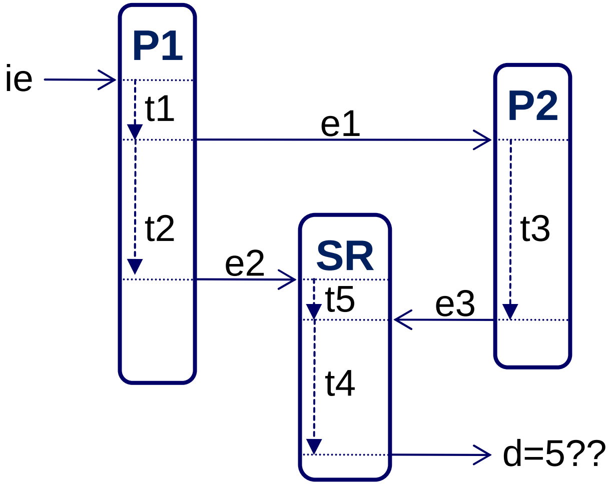


System simulation

TLM simulation

Motivation

- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design

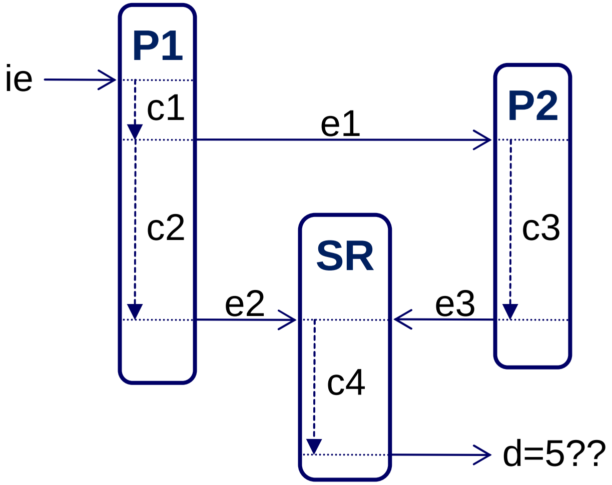


System simulation

TLM simulation

Motivation

- Timing transformations
  - Source of design errors
  - Classical problem in concurrent programming
  - Not enough addressed in system (HW/SW) design



System simulation

TLM simulation

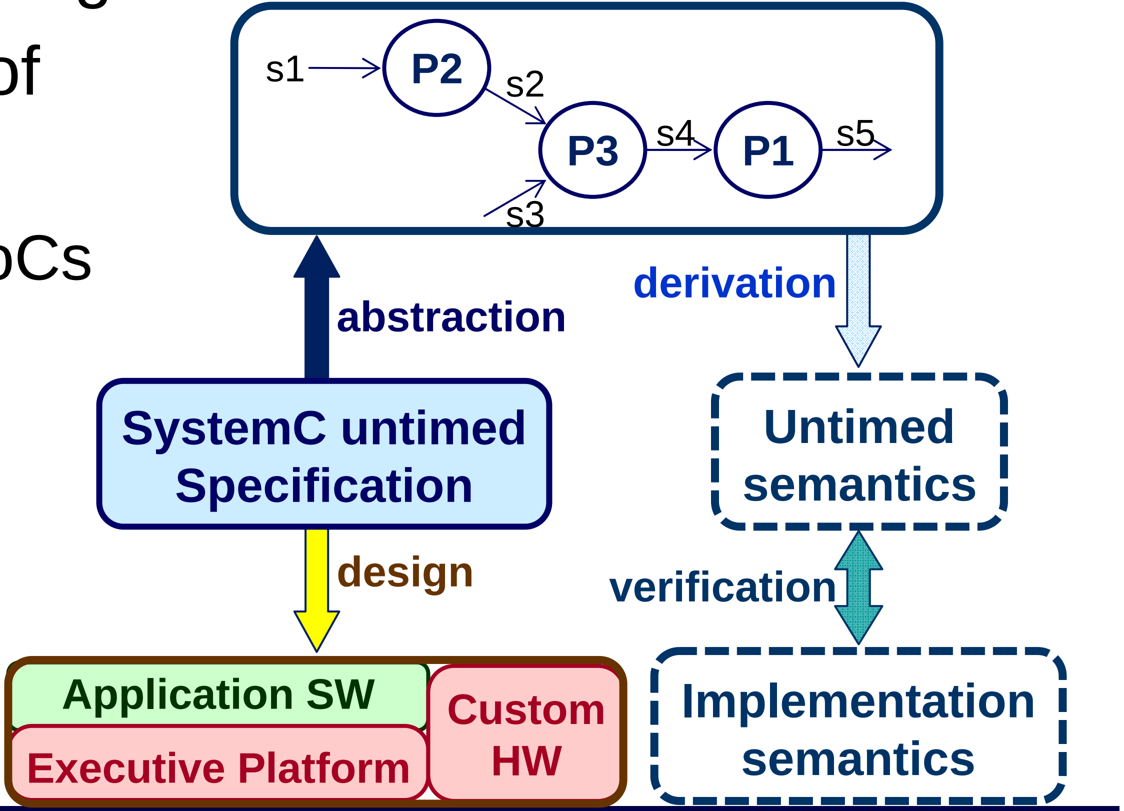
Cycle simulation

Motivation

- Need for formal support
  - – SystemC-based design
  - – Timing transformations
    - From Untimed Specifications
    - Down to TLM and Cycle-accurate implementations

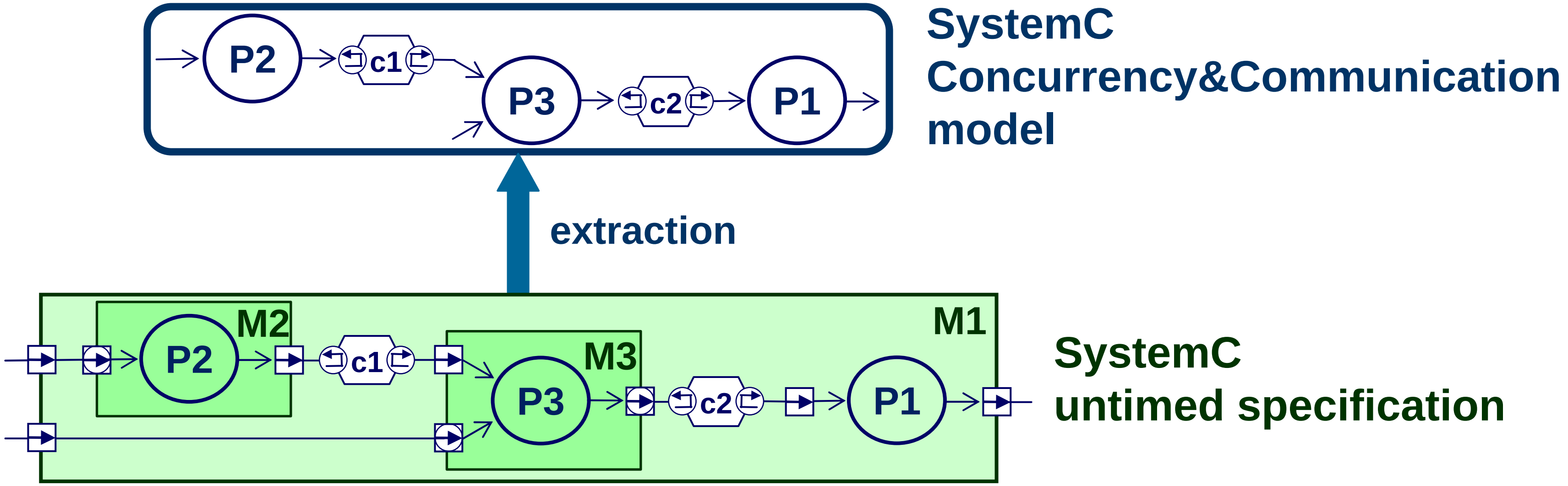
Introduction

- ForSyDe
  - Formal System Design meta-model from KTH
  - Formal definition of
    - Untimed MoCs
    - Synchronous MoCs
    - Timed MoCs



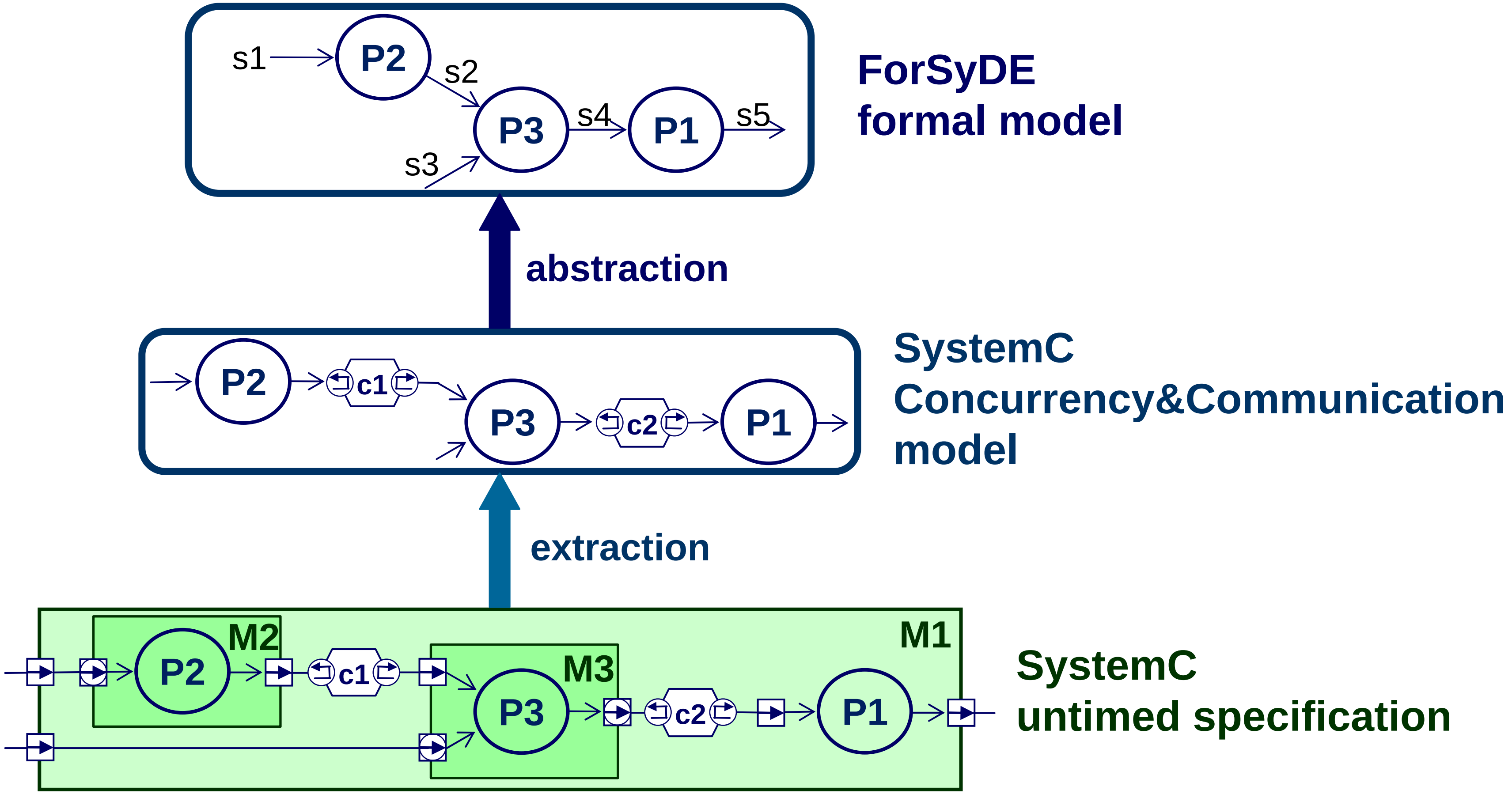
Formal Framework

- Extraction of the SystemC C&C model
  - Removal of hierarchical facilities
  - Flat model of communicating concurrent processes


P2
c1
P3
c2
P1
SystemC Concurrency&Communication model
extraction
M2
P2
c1
M3
P3
c2
P1
M1
SystemC untimed specification

Formal Framework

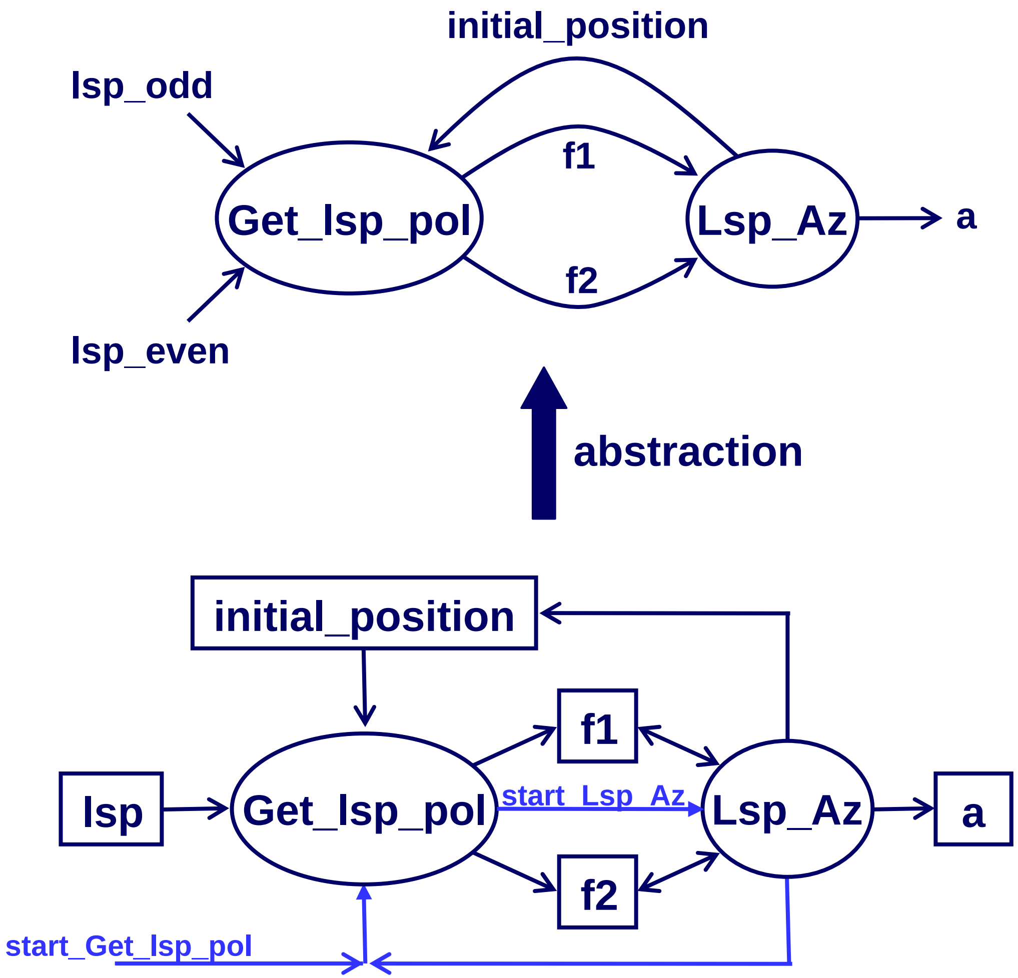
- Abstraction of the formal model



Formal Framework: Simple Example

```
SC_MODULE(Simple_Example) {
  . Port declarations
  . Channel/submodule instances
  Word16 lsp[10], a[11];
  int initial_position=0;
  Word32 f1[6], f2[6];
  sc_event start_Get_lsp_pol; start_Lsp_Az;
  SC_CTOR(Simple_Example) {
    . Connectivity
    SC_THREAD(Get_lsp_pol);
    SC_THREAD(Lsp_Az); }

  void Get_lsp_pol() {
    while (true) {
      wait(start_Get_lsp_pol);
      . Get_lsp_pol computes f1 from the odd positions
      . or f2 from the even positions of 'lsp' depending
      . on the value of 'initial_position'
      notify (start_Lsp_Az); }
  }
  void Lsp_Az() {
    while (true) {
      wait(start_Lsp_Az);
      initial_position = 1;
      notify(start_Get_lsp_pol);
      for (i=5;i>0;i--) f1[i]=L_add(f1[i],f1[i-1]);
      wait(start_Lsp_Az);
      initial_position = 0;
      for (i=5;i>0;i--) f2[i]=L_sub(f2[i],f2[i-1]);
      . a is computed from f1 and f2
}}}
```



Formal Framework: Simple Example

```
SC_MODULE(Simple_Example) {
  . Port declarations
  . Channel/submodule instances
  Word16 lsp[10], a[11];
  int initial_position=0;
  Word32 f1[6], f2[6];
  sc_event start_Get_lsp_pol; start_Lsp_Az;
  SC_CTOR(Simple_Example) {
    . Connectivity
    SC_THREAD(Get_lsp_pol);
    SC_THREAD(Lsp_Az); }

  void Get_lsp_pol() {
    while (true) {
      wait(start_Get_lsp_pol);
      . Get_lsp_pol computes f1 from the odd positions
      . or f2 from the even positions of 'lsp' depending
      . on the value of 'initial_position'
      notify (start_Lsp_Az); }
  }
  void Lsp_Az() {
    while (true) {
      wait(start_Lsp_Az);
      initial_position = 1;
      notify(start_Get_lsp_pol);
      for (i=5;i>0;i--) f1[i]=L_add(f1[i],f1[i-1]);
      wait(start_Lsp_Az);
      initial_position = 0;
      for (i=5;i>0;i--) f2[i]=L_sub(f2[i],f2[i-1]);
      . a is computed from f1 and f2
}}}
```

```
Get_lsp_pol = mapU(1,$\nu_{lsp\_odd}$, $\nu_{lsp\_even}$,f,f)
    Get_lsp_pol(initial_position,lsp_odd,lsp_even) = <<f1><f2>>
    //an 'initial_position' value is always taken
    $\pi(\nu_{initial\_position}$, initial_position) = <initial_position$_i$>
        $\nu_{initial\_position}(i) = 1$
    If (initial_position$_i$ = 0) then
        f1$_i$ = f(lsp_odd$_i$)
        //five 'lsp_odd' data are taken but no 'lsp_even'
        $\pi(\nu_{lsp\_odd}$, lsp_odd) = <lsp_odd$_i$>
        $\nu_{lsp\_odd}(i) = 5$
        $\pi(\nu_{lsp\_even}$, lsp_even) = <lsp_even$_i$>
        $\nu_{lsp\_even}(i) = 0$
        //six 'f1' data are generated but no 'f2'
        $\pi(\nu_{f1}$, f1) = <f1$_i$>
        $\nu_{f1}(i) = 6$
        $\pi(\nu_{f2}$, f2) = <f2$_i$>
        $\nu_{f2}(i) = 0$
    else
        f2$_i$ = f(lsp_even$_i$)
        //five 'lsp_even' data are taken but no 'lsp_odd'
        $\pi(\nu_{lsp\_odd}$, lsp_odd) = <lsp_odd$_i$>
        $\nu_{lsp\_odd}(i) = 0$
        $\pi(\nu_{lsp\_even}$, lsp_even) = <lsp_even$_i$>
        $\nu_{lsp\_even}(i) = 5$
        //six 'f2' data are generated but no 'f1'
        $\pi(\nu_{f1}$, f1) = <f1$_i$>
        $\nu_{f1}(i) = 0$
        $\pi(\nu_{f2}$, f2) = <f2$_i$>
        $\nu_{f2}(i) = 6$
```

Formal Framework: Simple Example

```
SC_MODULE(Simple_Example) {
  . Port declarations
  . Channel/submodule instances
  Word16 lsp[10], a[11];
  int initial_position=0;
  Word32 f1[6], f2[6];
  sc_event start_Get_lsp_pol; start_Lsp_Az;
  SC_CTOR(Simple_Example) {
    . Connectivity
    SC_THREAD(Get_lsp_pol);
    SC_THREAD(Lsp_Az); }

  void Get_lsp_pol() {
    while (true) {
      wait(start_Get_lsp_pol);
      . Get_lsp_pol computes f1 from the odd positions
      . or f2 from the even positions of 'lsp' depending
      . on the value of 'initial_position'
      notify (start_Lsp_Az); }
  }
  void Lsp_Az() {
    while (true) {
      wait(start_Lsp_Az);
      initial_position = 1;
      notify(start_Get_lsp_pol);
      for (i=5;i>0;i--) f1[i]=L_add(f1[i],f1[i-1]);
      wait(start_Lsp_Az);
      initial_position = 0;
      for (i=5;i>0;i--) f2[i]=L_sub(f2[i],f2[i-1]);
      . a is computed from f1 and f2
}}}
```

```
Az_Lsp = mealyU($\nu_{f1}$, $\nu_{f2}$,g,$f_{i\_p}$,$f_a$,$\omega_0$)
  Az_Lsp(f1,f2) = <<initial_position><a>>
  If ($state_i$=$\omega_0$) then
      $initial\_position_i$=$f_{i\_p}$(state)=1
      //six 'f1' data are taken but no 'f2'
      $\pi(\nu_{f1}, f1) = \langle f1_i \rangle$
      $\nu_{f1}(i) = 6$
      $\pi(\nu_{f2}, f2) = \langle f2_i \rangle$
      $\nu_{f2}(i) = 0$

      $state_{i+1}$=$\omega_1$

      //no 'a' data is generated
      $\pi(\nu_a, a) = \langle a_i \rangle$
      $\nu_a(i) = 0$
  else
      $initial\_position_i$=$f_{i\_p}$(state)=0
      //six 'f2' data are taken but no 'f1' although 'f1 is used
      $\pi(\nu_{f1}, f1) = \langle f1_i \rangle$
      $\nu_{f1}(i) = 0$
      $\pi(\nu_{f2}, f2) = \langle f2_i \rangle$
      $\nu_{f2}(i) = 6$

      $state_{i+1}$=$\omega_0$

      $a_i = f_a(f1_i, f2_i)$
      //eleven 'a' data are generated
      $\pi(\nu_a, a) = \langle a_i \rangle$
      $\nu_a(i) = 11$
```

Formal Framework: Simple Example

Untimed semantics

derivation

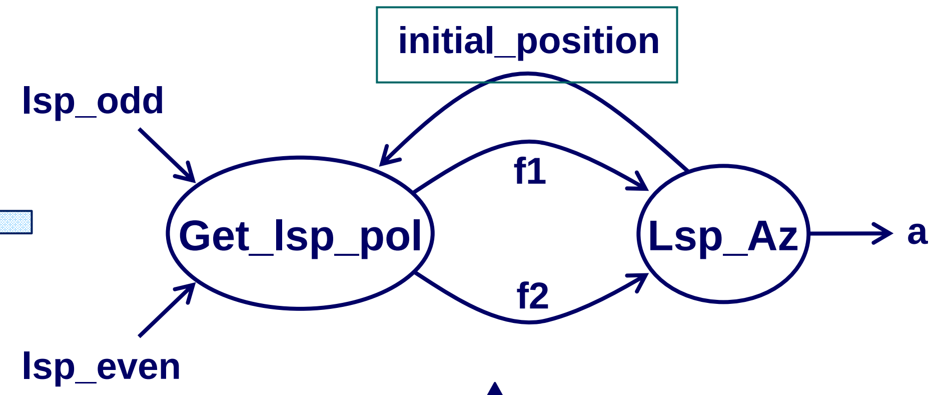
implication

CSV1. *Every data token written by the producer process is read by the consumer process.*

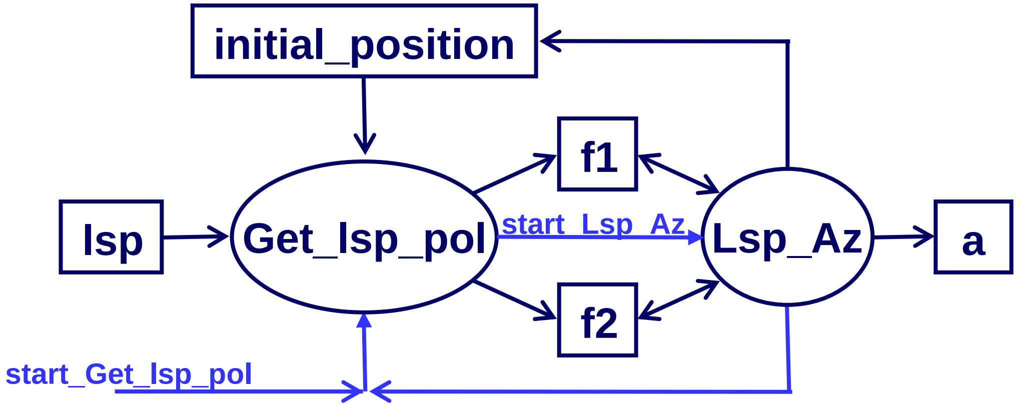
CSV2. *Every data token written by the producer process is read only once by the consumer process.*

design property

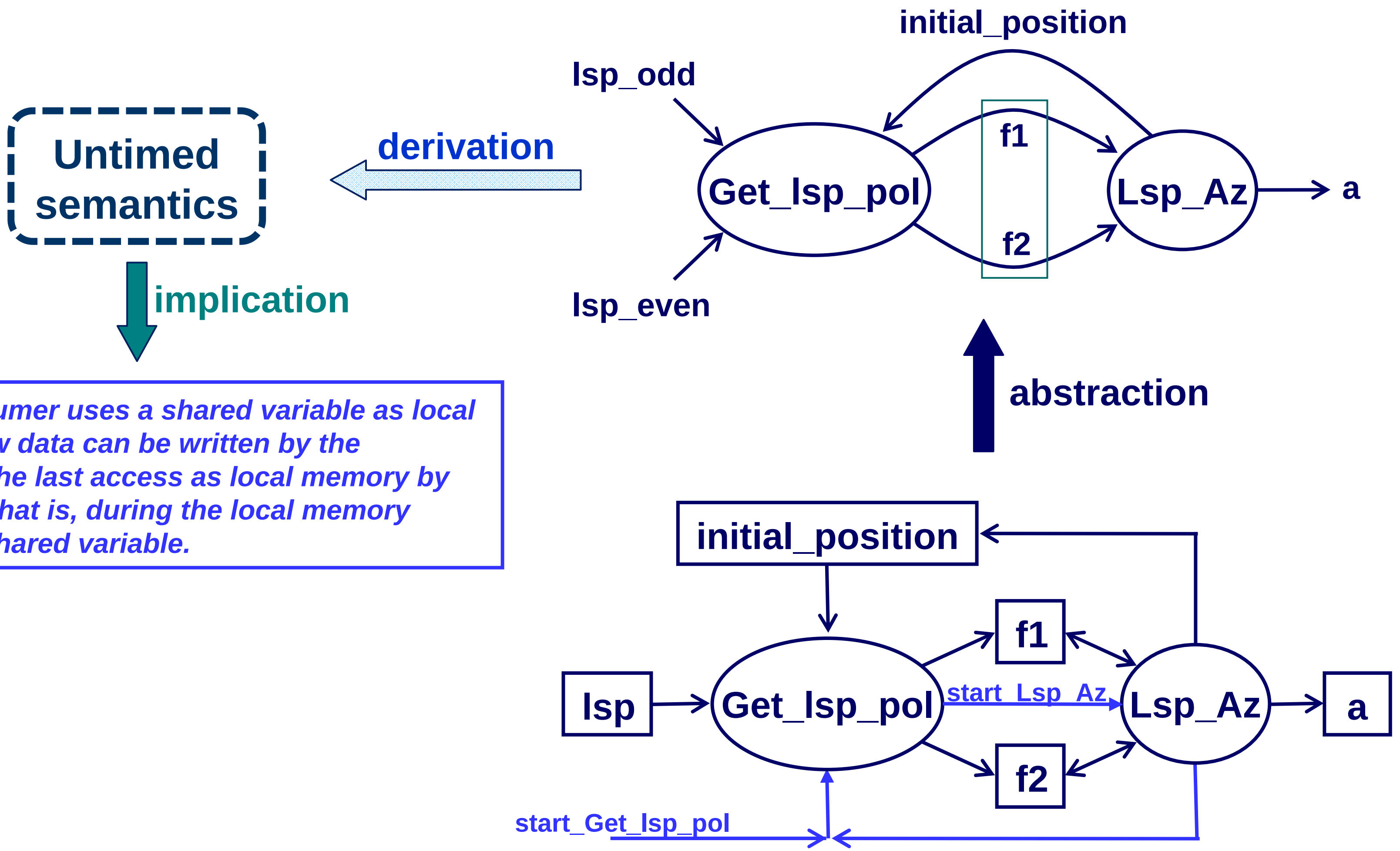
```
-- psl property CSV1_2_initial_position is always
        (initial_position.write_CALL() ->
        next(initial_position.read_CALL()
        before initial_position.write_CALL()));
-- psl assert CSV1_2_initial_position
```



abstraction



Formal Framework: Simple Example



CSV3. *If a consumer uses a shared variable as local memory, no new data can be written by the producer until the last access as local memory by the consumer, that is, during the local memory lifetime of the shared variable.*

Application to High-Level Synthesis

- Synthesis results (after Gaut)

Strategy	Get_lsp_pol	Az_Lsp
Min. Area	1,740	810
Min. Latency	960	370

- Verification results
  - System verification testbench

Design	1	2	3	4
CSV1_2_initial_position	True	True	True	True
CSV1_2_f1	True	True	**False**	**False**
CSV1_2_f2	True	True	True	True
CSV3_f1	True	True	**False**	**False**
CSV3_f2	True	True	True	True
Functional correctness	OK	OK	**Error**	**Error**

Conclusions

- ForSyDe has been shown as a formal meta-model for SystemC
  - Untimed models
  - Specification semantics to be preserved
  - Application to high-level synthesis

Future Work

- Extended SystemC specification
  - More complex communication mechanisms
- Other applications
  - Architectural design
  - HW/SW mappings

Thanks and Questions

- Thank you for your attention

- Funding

SIXTH FRAMEWORK PROGRAMME

ANDRES

COOPERATION

SATURN