• Introduction to the VHDL Hardware description language

1. Introduction
2. Basic elements
3. Scalar data types
4. Composed data types
5. Basic constructs (system definition)
6. Data flow description level
7. Structural description level
8. Behavioral description level
9. Design organisation

1. Introduction

- VHSIC Project (DoD, USA)
- Initial goal: facilitate documentation
- Developed by TI, Intermetrics, IBM (1985)
- IEEE Standard (1076) since 1987 (revisions 1993 and 2000)
- Strongly linked to data types (ADA)
- Basic description levels:
  - data flow, structural, behavioral
- Basic objects:
  - constants, variables, signals and files

2. Basic elements

- **Comments:** --
  -- That's a comment
- **Identifiers (labels):**
  - alphabetic characters + digits + _
  - start: **always** with alphabetic character
  - **can not** end with _
  - **can not** contain two successive _
- **Characters:** 'C'
- **Strings:** "OnlY a lINe"
- **Numbers (integer or real):** 23 45.2E-4 16#A4#E-7

- **Bit strings:** [B|O|X]”101”
- **Constants:**

  constant n_bits_per_word: integer:=12;
  constant pi: real:=3.1415926535
- **Variables:**

  variable counter: integer:=0;
- **End of sentence:** ;
- **Assignments:** <= (signals) := (variables and constants)

- **Assert primitive:** Condition monitoring

  assert vdd/=5 **report** “VDD not connected”
  severity warning;

3. Scalar data types

- **Type declaration:**

  type mean_weight **is** 65 **to** 80;

  type logic_level **is** (‘0’, ‘1’, ‘Z’, ‘X’);

- **Integer type:**

  type word_type **is** (4, 8, 16, 32, 64);

  variable opcode: word_type:=4;

- **Floating point type:**

  type probability **is** 0.0 **to** 1.0;

- **Physical type:**

  type resistance **is range** 0 **to** 1E9
  units
  ohm;
  kohm=1000 ohm;
  Mohm = 1000 kohm
  end units resistance;

- **Time type:** Default time resolution: *fs*
- **Character type:** 8-bits ISO character set
- **Boolean type:**

  type boolean **is** (false, true);
- **Bit type:**

  type bit **is** (‘0’, ‘1’);
- **Subtype declaration:**

  subtype pointer **is** interger **range** 15 **downto** 0;
- **Conversion between types:** real(84) integer(36.9);
- **Operators:**

Precedence ↑

** abs not
* / mod rem
+ - &
sll srl sla sra rol ror
= /= < <= > >=
and or nan nor xor **xnor**

4. Composed data types

- **Array:**

```
type  word is (0 to 15) of bit;
variable register_A: word:=X"A42E";
carry := register_A(4);

type curve_points is (0 to 3) of integer;
variable curve_1: curve_points:=
        (1=>0, 2=>5, others =>0);

type RAM is (natural range <>) of bit_vector(7 downto 0);
variable control_ram: ram(0 to 1024);

type bidim_array is (0 to 3, 0 to 7) of bit;
```

- **Record:** Fields with **different types**

```
type instruction is record
        opcode: bit_vector(7 downto 0);
        source_reg: bit_vector(3 downto 0);
        destination_reg: bit_vector(3 downto 0);
end record instruction;
variable instruction_reg: instruction;


instruction_reg.opcode:=data_cpu(7 downto 0);
instruction_reg.source_reg:=data_cpu(11 downto 8);
instruction_reg.destination_reg:=data_cpu(15 downto 12);
```

- **Attributes:** Specify properties of an element

Scalar attributes

Attribute	Type of T	Type of result	Result
T'left	any scalar type or subtype	same as T	leftmost value in T
T'right	"	"	rightmost value in T
T'low	"	"	lowest value in T
T'high	"	"	highest value in T
T'ascending	"	boolean	true if T is an ascending range, false otherwise
T'image(x)	"	string	textual representation of the value x from type T
T'value(s)	"	base type of T	value in T represented by the string s
T'pos(x)	any discrete type or subtype	universal integer	position of x in T
T'val(x)	"	base type of T	value at position x in T

Scalar attributes (cont.)

Attribute	Type of T	Type of result	Result
T'succ(x)	any discrete type or subtype	base type of T	value in T at position one greater than that of x
T'pred(x)	"	"	value in T at position one less than that of x
T'leftof(x)	"	"	value in T at position one to the left of x
T'rightof(x)	any discrete or physical type or subtype	"	value in T at position one to the right of x
T'base	any type or subtype	"	base type of type T, only allowed as a prefix of another attribute

Array attributes

Attribute	Result
A'left(n)	left bound of index range of dimension n of A
A'right(n)	right bound of index range of dimension n of A
A'low(n)	lower bound of index range of dimension n of A
A'high(n)	upper bound of index range of dimension n of A
A'range(n)	index range of dimension n of A
A'reverse_range(n)	reverse of index range of dimension n of A
A'length(n)	length of index range of dimension n of A
A'ascending(n)	true if index range of dimension n of A is ascending, false otherwise

Signal attributes

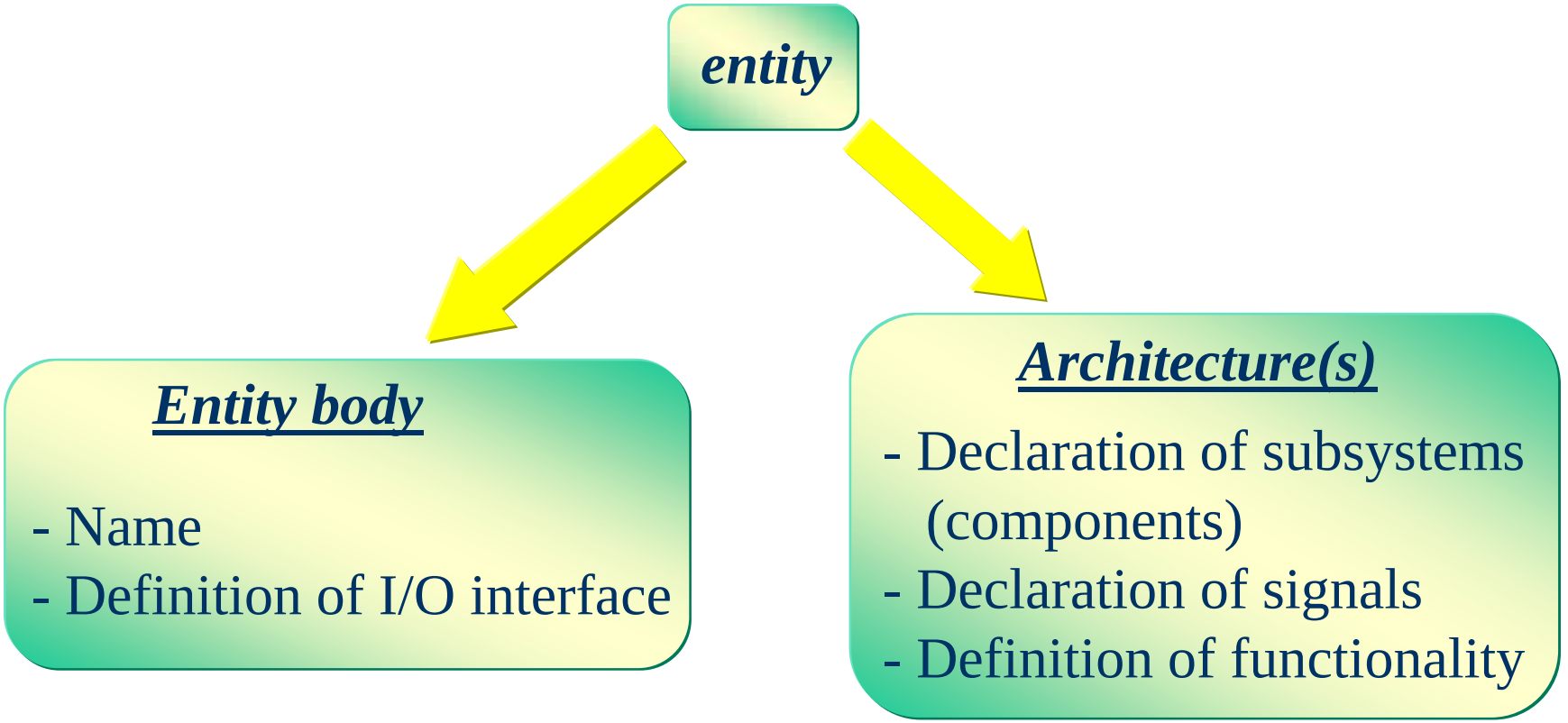
Attribute	Type of result	Result
S'delayed(t)	base type of S	a signal that takes on the same value as S but is delayed by time T
S'stable(t)	boolean	a boolean signal that is true if there has been no event on S in the time interval t up to the current time, otherwise false
S'quiet(t)	boolean	a boolean signal that is true if there has been no transaction on S in the time interval t up to the current time, otherwise false
S'transaction	bit	implicit bit signal, which changes its value each time there is a transaction on S
S'event	boolean	true if there is a transaction on S in the current simulation cycle, otherwise false
S'active	boolean	true if there is a transaction on S in the current simulation cycle, otherwise false
S'last_event	time	time interval since the last event on S
S'last_active	time	time interval since the last transaction on S
S'last_value	base type of S	value of S before the last event
S'driving	boolean	true if the current process is producing a transaction on S
S'driving value	base type of S	value assigned to S in the current process

➢ Example:

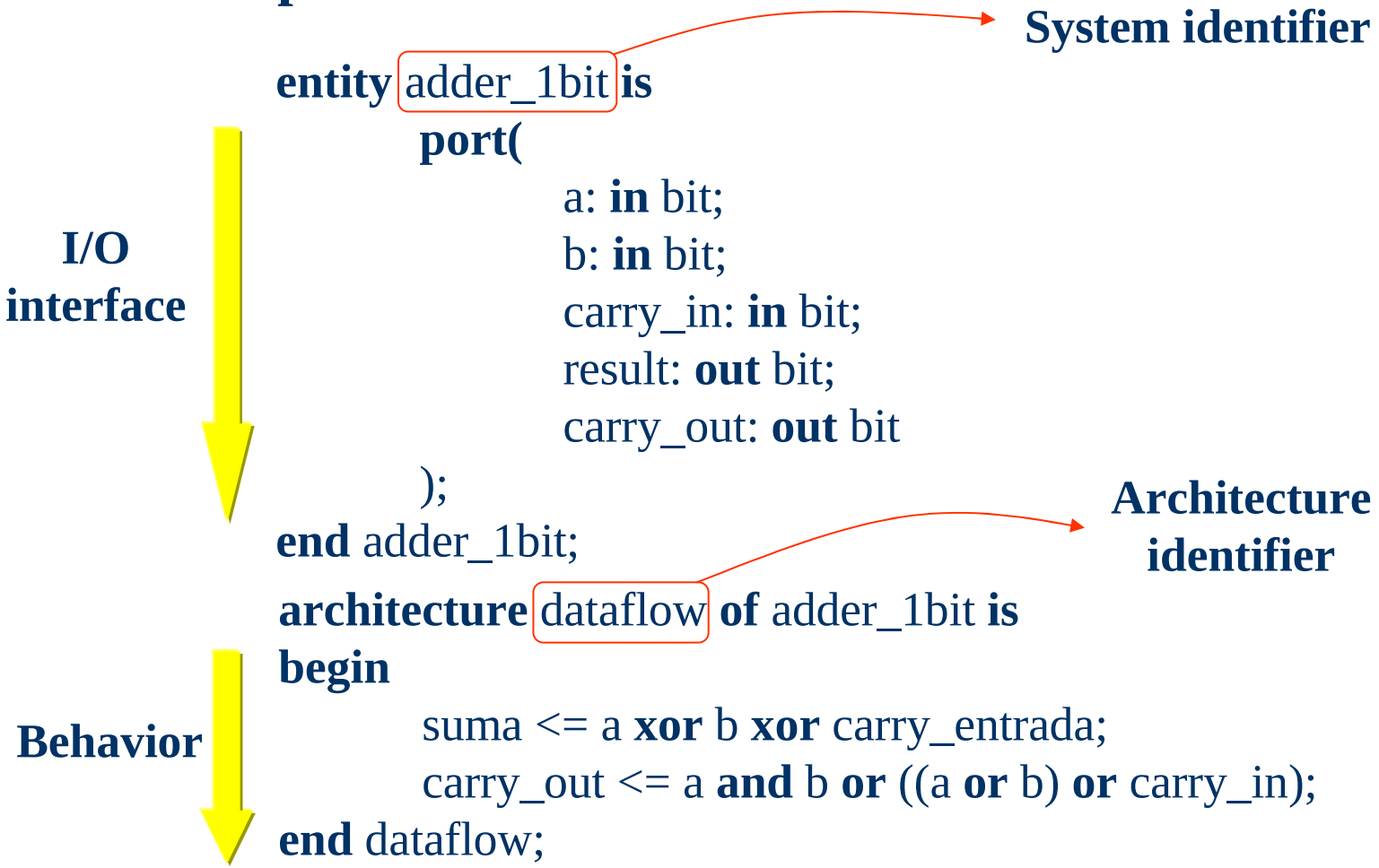
```
function increment (val:bit_vector) return bit_vector is
variable result: bit_vector(val'range);
variable carry: bit;
begin
        result(0):= not val(0);
        carry:= val(0);
        for i in 1 to val'high loop
                carry:= carry and val(i-1);
                result(i):= val(i) xor carry;
        end loop;
        return result;
end increment;
```

5. Basic constructs

- **System = entity**



➢ Example:


System identifier
entity adder_1bit is
port(
a: in bit;
b: in bit;
carry_in: in bit;
result: out bit;
carry_out: out bit
);
I/O interface
end adder_1bit;
Architecture identifier
architecture dataflow of adder_1bit is
begin
suma <= a xor b xor carry_entrada;
carry_out <= a and b or ((a or b) or carry_in);
Behavior
end dataflow;

- **Architecture description levels:**

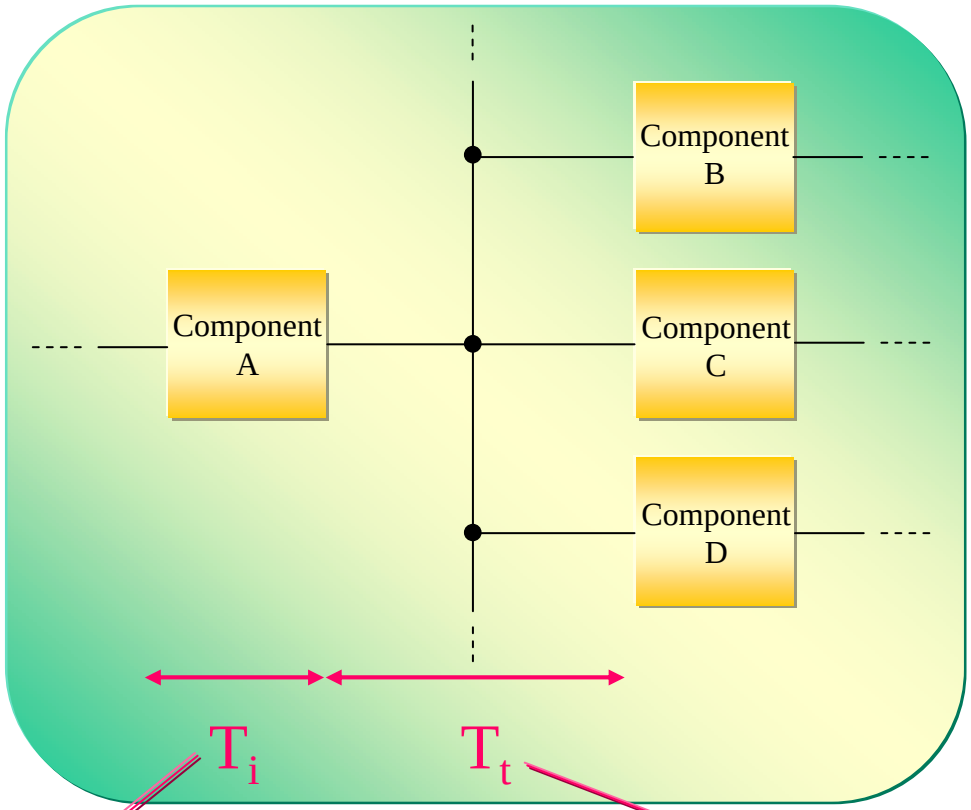
➢ **Data flow (declarative):**
- Assignments
- Blocks

➢ **Structural (applicative semantics):**
- Components
- Generate primitive

➢ **Behavioral (procedural):**
- Processes
- Procedures and functions

6. Data flow description level

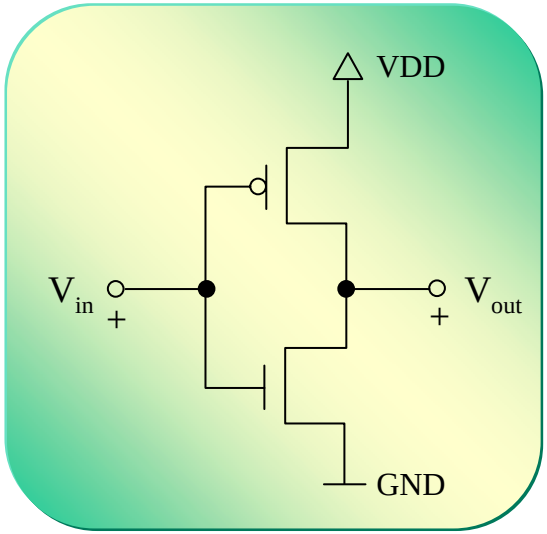
- **Timing:**

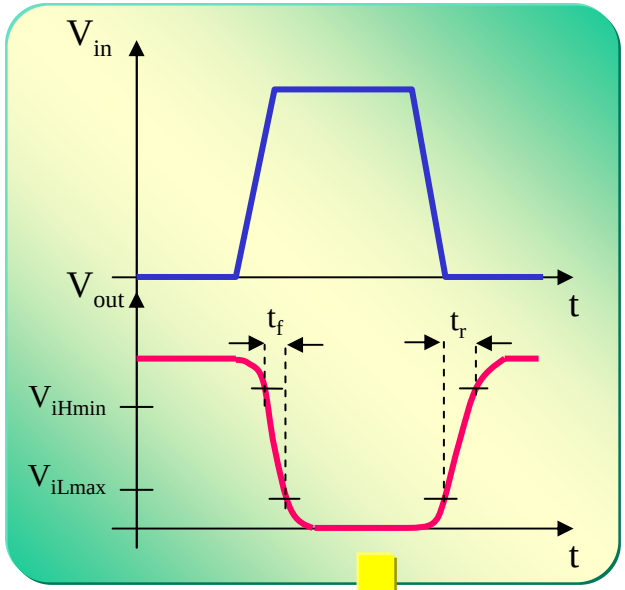


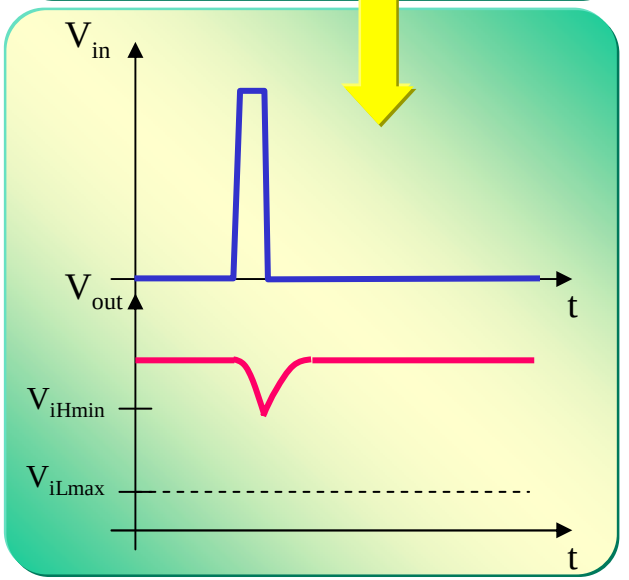
Response time (inertial delay)

Propagation time (transport delay)

- **Inertial delay:**

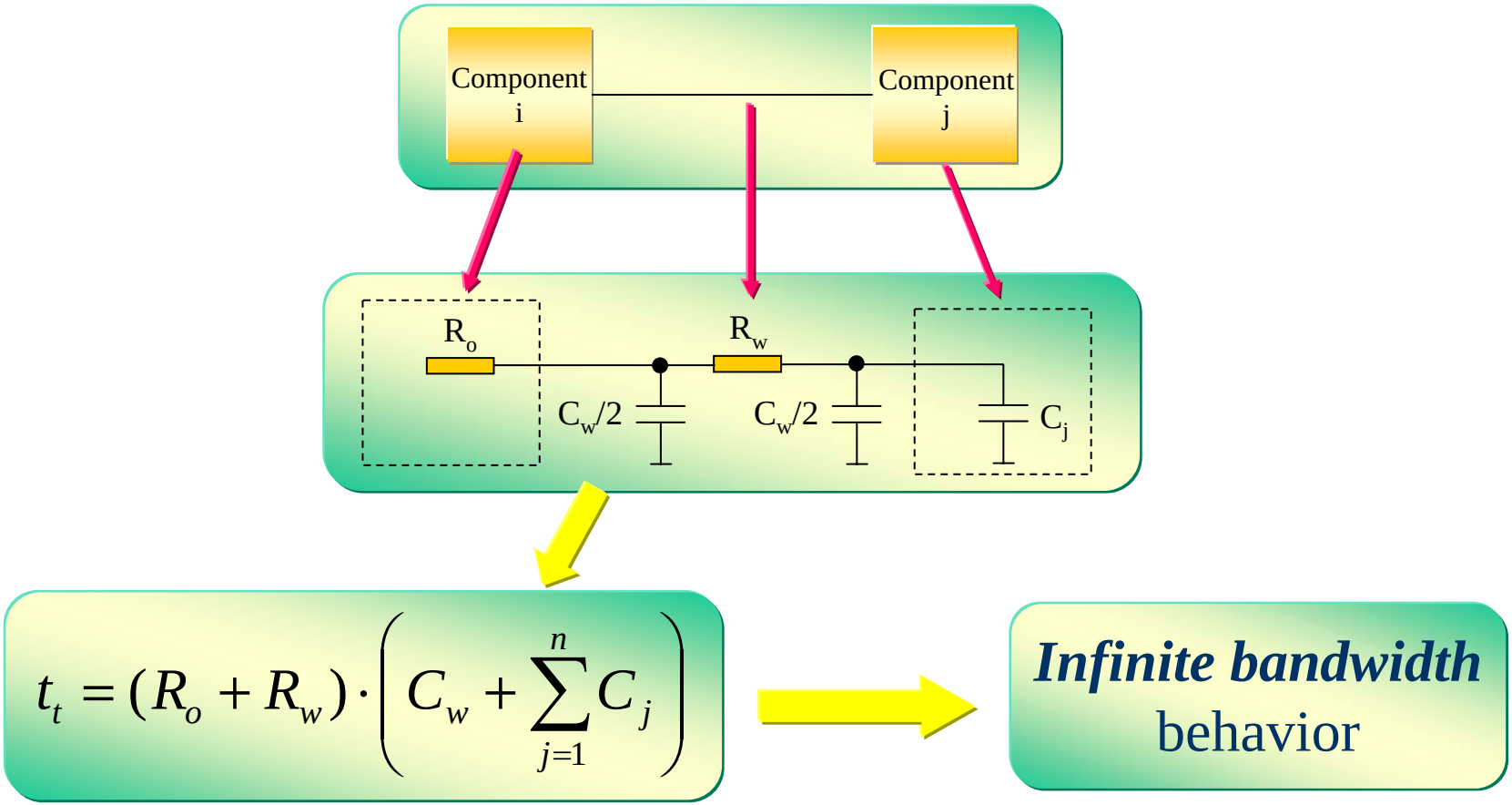






Low-pass filter behavior

- **Transport delay:**



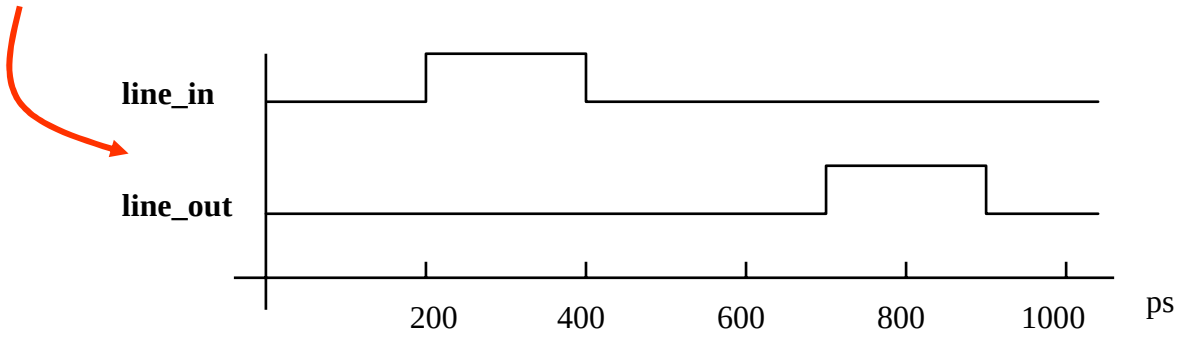
- **Unconditional assignment:**

A_signal <= value1 **after** delay1, value2 **after** delay2, ..., valuen **after** delayn;

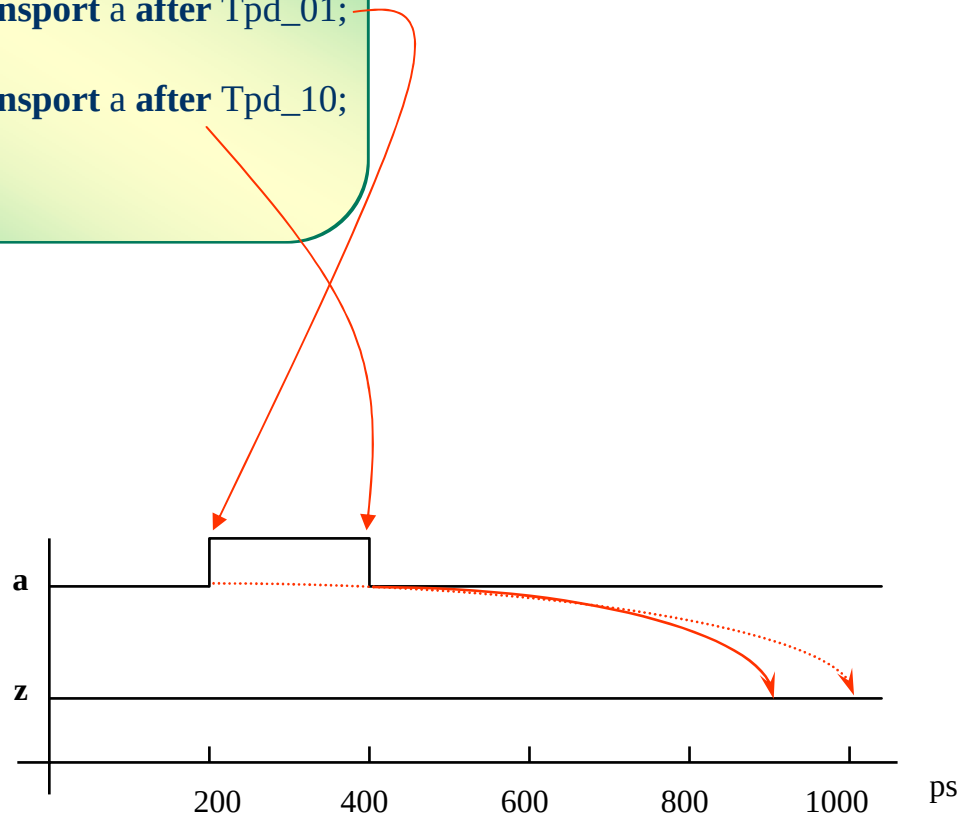
- **Delay types: inertial and transport**

- transport: infinite frequency (propagation of current)

```
t_line: process (line_in)
begin
        line_out <= transport line_in after 500 ps;
end process t_line;
```

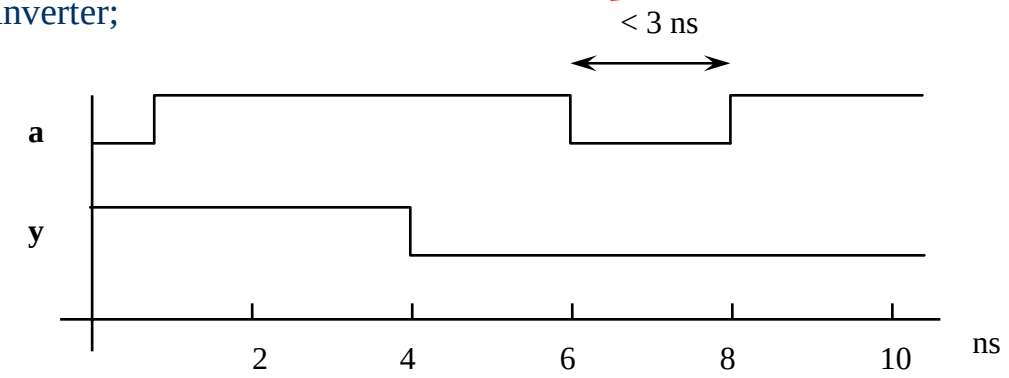


```
asym_delay: process (a)
constant Tpd_01: time:=800 ps;
constant Tpd_10: time:=500 ps;
begin
        if a='1' then
                z<= transport a after Tpd_01;
        else
                z<= transport a after Tpd_10;
        end if;
end process asym_delay;
```



- inertial: default delay type

```
inverter: process (a)
begin
        y<= (inertial) not a after 3 ns;
end process inverter;
```




```
S<= 1 after 5 ns, 5 after 10 ns, 10 after 15 ns;
                    |||
S<= 1 after 5 ns;
S<= transport 5 after 10 ns;
S<= transport 10 after 15 ns;
```

➢ Conditional assignment:

```
A_signal <=   value1 when condition1
              else value2 when condition2
              ...
              else default_value;
```

➢ Selective assignment:

```
with expression select
        A_signal <=   value1 when selection1,
                      value2 when selection2,
                      ...
                      default_value when others;
```

- **Signal resolution:**

  ➢ Definition of ***resolution functions*** which determine the value of a net when it is connected to different drivers (active or not)

  - **Guarded signals:** Definition of a value when drivers are disconnected (**null** assignment)

    - **Bus:** Empty array (*pull-up*)
    - **Register:** Last value (*dynamic memory*)

    signal overflow: pullup_type **bus**;
    signal load_1: bit **register**;

- **IEEE Standard 1164:** 9-value logic

'U': Not initialised	'W': Unknown (weak driver)
'X': Unknown (strong driver)	'L': Low level (weak driver)
'0': Low level (strong driver)	'H': High level (weak driver)
'1': High level (strong driver)	**'-': Don't care**
'Z': High impedance	

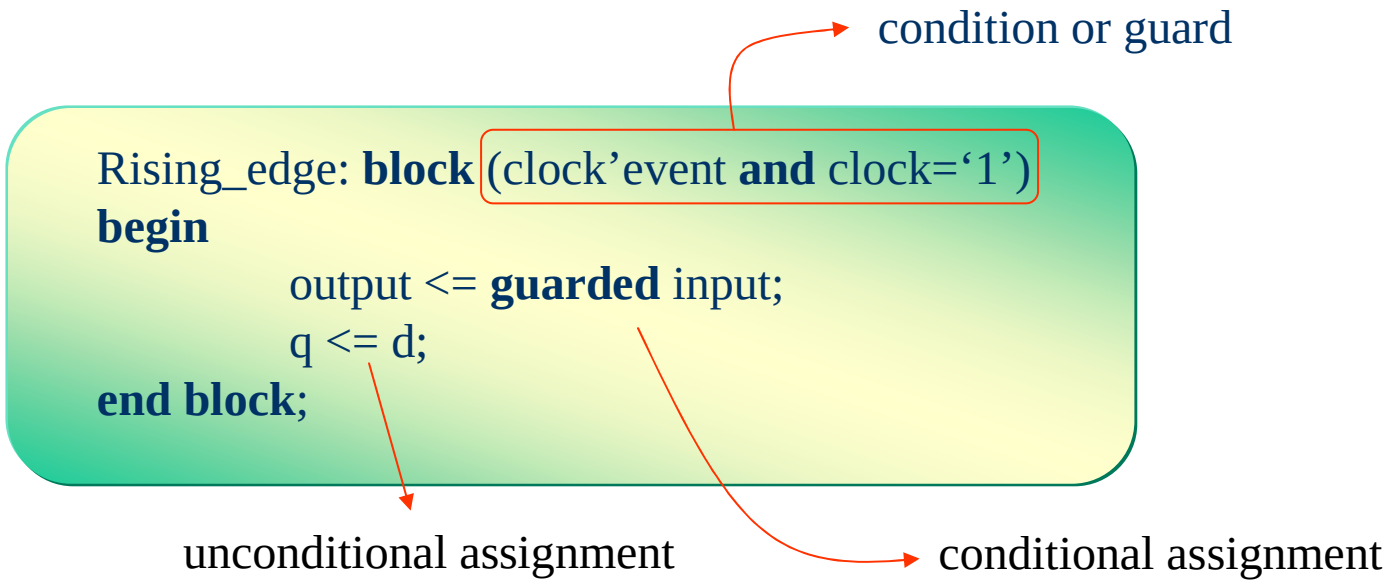
- **Use of the 1164 data types:**

```
library ieee;
use ieee.std_logic_1164.all;
```

- **Basic types:**

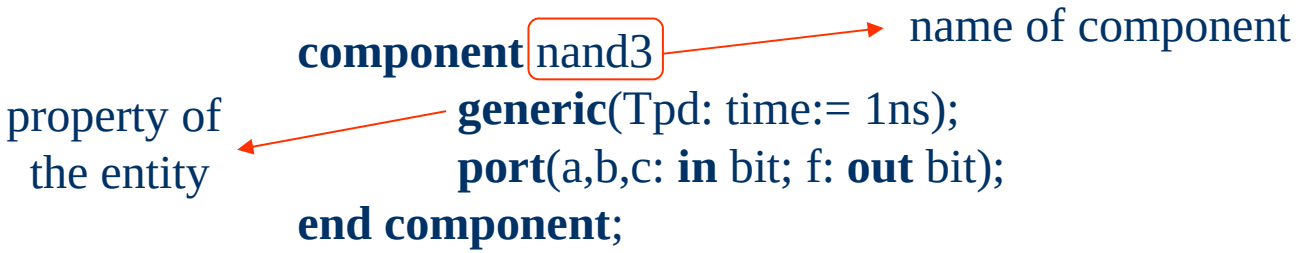
- std_logic (= bit)
- std_logic_vector (= bit_vector)

- **Blocks:**
  - Functional subsystem
  - **Concurrent** execution
  - Interconnection through ports
  - Definition of guards



7. Structural description level

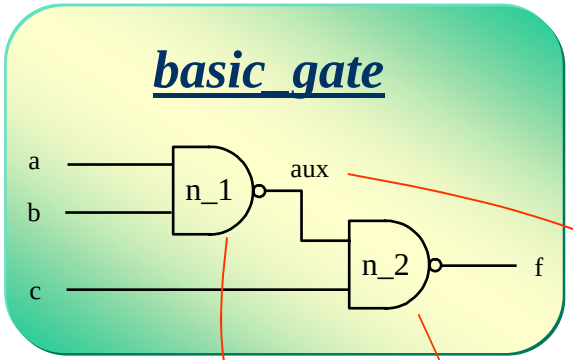
- **Components:**
  - Subsystem defined in a library
  - Component declaration:



  - Component instantiation:

```
enable: nand3
        port map(en1, en2, int_req, interrupt);
```

➢ Example:




```
entity basic_gate is
port(a,b,c: in std_logic; f: out std_logic);
end basic_gate;

architecture structural of basic_gate is
component nand2
port(a,b: in std_logic; f: out std_logic);
end component;
signal aux: std_logic;
begin
        n_1: nand2
        port map(a=>a,b=>b,f=>aux);
        n_2: nand2
        port map(a=>aux,b=>c,f=>f);
end structural;
```

• Definition of regular structures:

• Generate primitive:

```
adder: for i in 0 to k-1
        ls_bit: if i=0 generate
                ls_cell: half_adder port map(a(0), b(0), s(0), c_in(1));
        end generate ls_bit;
        middle_bit: if i > 0 and i < k-1 generate
                mid_cell: full_adder port map(a(i), b(i), c_in(i), s(i), c_in(i+1));
        end generate middle_bit;
        ms_bit: if i=k-1 generate
                ms_cell: full_adder port map(a(i), b(i), c_in(i), s(i), carry);
        end generate ms_bit;
end generate adder;
```

8. Behavioral description level

- **Processes:**
  - Sequential execution body
  - Activated by signal transitions
  - **Concurrent** execution

- **Activation control:**

Sensitivity list

```
process(reset, clock)
        variable state: bit:= false;
begin
        if reset then
                state:=false;
        elsif clock=true then
                state:= not state;
        end if;
        q<= state after delay;
end process;
```

Wait primitive

```
process
        variable state: bit:=false;
begin
        wait until (clock'event or reset'event);
        if reset then
                state:= false;
        elsif clock=true then
                state:= not state;
        end if;
        q<= state after delay;
end process;
```

- **Primitives for sequential control:**
  - **Wait:**

```
wait    [until condition;]
        [on signal1, signal2, ...;]
        [for temporal_expression;]
```

  - **If:**

```
if [condition] then [actions] elsif [actions] ... else [actions] ... end if;
```

  - **Case:**

```
case [expression] is
        when [selection] => [actions];
        ...
end case;
```

- **Indefinite loop:**

```
loop [actions] end loop;
```

- **While loop:**

```
while [condition] loop
    [actions];
end loop;
```

- **For loop:**

```
for [identifier] in [range] loop
    [actions];
end loop;
```

- **Loop control: next, exit [when condition]**

➢ Example:

```
architecture behavior of counter is
begin
        increment: process
        variable value: integer:=0;
        begin
                output <= value;
                loop
                        loop
                                wait until clk='1' or reset='1';
                                exit when reset='1';
                                value:= (value+1) mod 16;
                                output <= value;
                        end loop;
                        value:=0;
                        output <= value;
                        wait until reset='0';
                end loop;
        end process increment;
end behavior;
```

- **Procedures:** May return a value

```
procedure mean is
variable partial: real:=0.0;
begin
        for index in matrix'range loop
                partial:= partial+matrix(index);
        end loop;
        mean_value:= partial / real(matrix'length);
end procedure mean;
```

- **Functions:** Always return a value

9. Design organisation

- **Library:** Object (directory, file, ...) which contains already designed modules

- **Definition of a reference library:**

```
library library_name;
```

- **Use of elements included in a library:**

```
use libray_name.defined_element;
```

- **Configuration of objects (binding):**

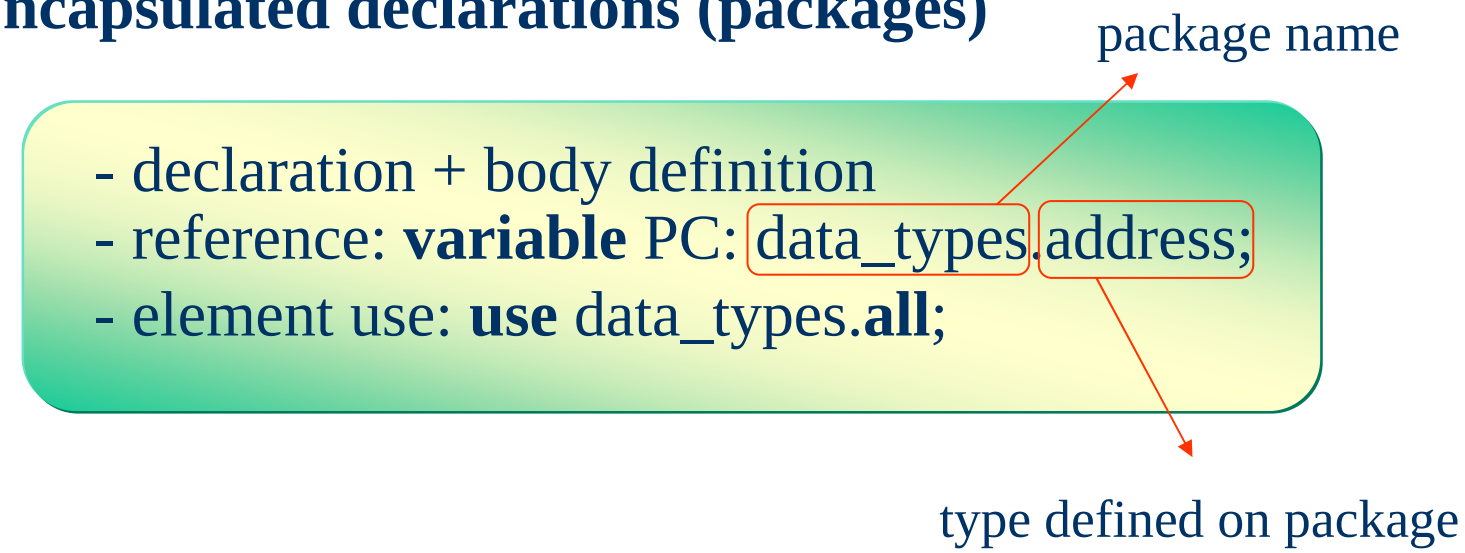
configuration name

```
library basic_flip_flop;
use basic_flip_flop.ff_d_rising_edge;

configuration reg4_structural of reg4 is
        for structural
                for bit0: flipflop
                        use entity ff_d_rising_edge(drive4);
                end for;
                for others: flipflop
                        use entity ff_d_rising_edge(drive1);
                end for;
        end for;
end configuration reg4_structural;
```

architecture of reg4

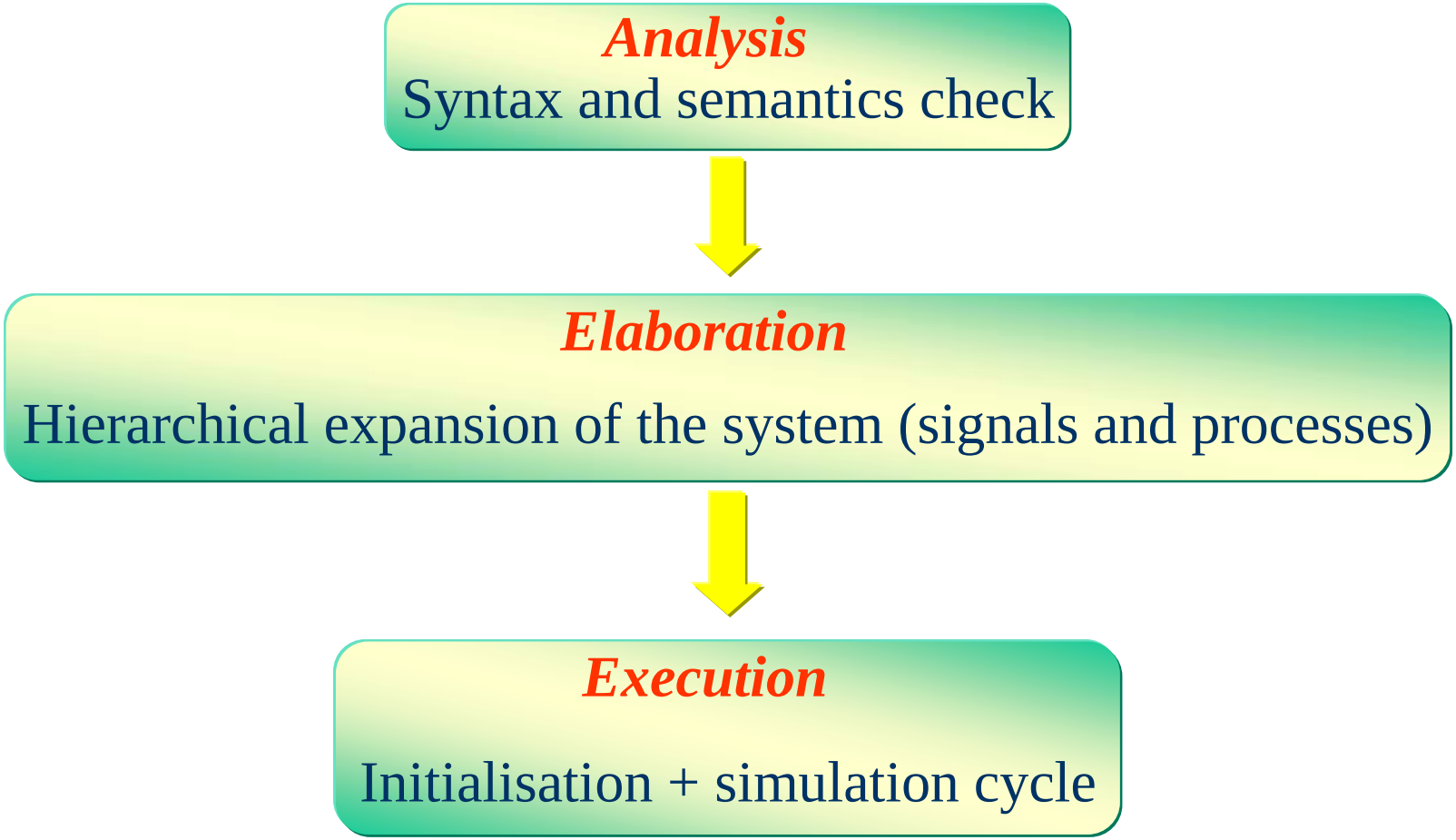
- **Encapsulated declarations (packages)**



- **Simulation:**

➢ **Test Bench:** VHDL model where stimuli for a given system are declared (and/or its outputs are verified)

- **Organisation of the simulation process**



- **Simulation cycle**

1) Time is advanced until the next time point where transactions on signals are to be performed

2) Transactions on signals are performed

3) The processes which are sensistive to the events produced are activated and executed

```
process …
begin
        …
        s <= ‘1’;
        …
        if s=‘1’ then …;
        …
end process;
```