

# Signals & Variables

---

Copyright (c) 2012 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).

This document was produced by using OpenOffice and Octave.

# Concurrent Statement

- Block Statement
- **Process Statement**
- Component Statement
- Generate Statement
- **Concurrent Signal Assignment**
- Concurrent Assertion
- Concurrent Procedure Call

- **Architecture Body**
- **Block Statement**
- **Generate Statement**

- **Conditional Signal Assignment**
- **Selected Signal Assignemnt**

# Sequential Statement

- Wait Statement
- Assertion Statement
- Report Statement
- Generate Statement
- Signal Assignment
- Variable Assignment
- Procedure Call
- If
- Case
- Loop
- Next
- Exit
- Return
- Null

- **Case Statement**
- **If Statement**
- **Loop Statement**
- **Process Statement**
- **Subprogram Body**

- Sequential Signal Assignment

- Conditional Signal Assignment
- Selected Signal Assignment

X

# Conditional Signal Assignment

```
Z <= A or B [ after 1 ns ] when S0 = '1' else  
      A or C [ after 2 ns ] when S1 = '1' else  
      A or D [ after 3 ns ] ;
```

```
Z <= A or B [ after 1 ns ] when S0 = '1' else  
      A or C [ after 2 ns ] ;
```

```
Z <= A or B [ after 1 ns ] when S0 = '1' ;
```

```
Z <= A or B [ after 1 ns ] ;
```

← *simple concurrent statement*

- Concurrent Signal Assignment

- Conditional Signal Assignment
- Selected Signal Assignment

# Selected Signal Assignment

- Conditional Signal Assignment

```
Z <=  A or B  [ after 1 ns ]  when SEL = "00" else  
      A or C  [ after 2 ns ]  when SEL = "01" else  
      A or D  [ after 2 ns ]  when SEL = "10" else  
      A or E  [ after 3 ns ]  when SEL = "11" else  
      A or F  [ after 4 ns ]  ;
```

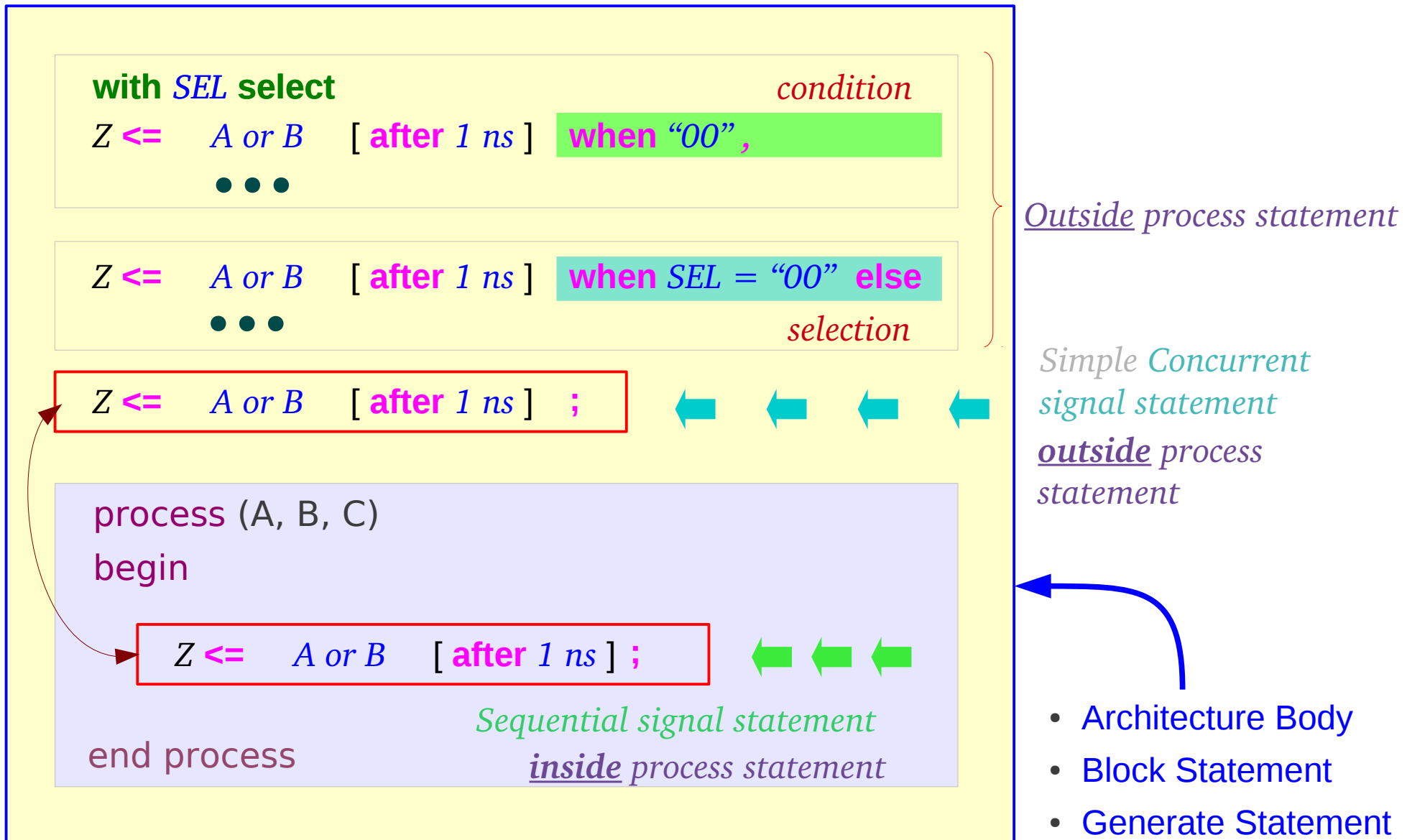
condition

- Selected Signal Assignment

```
with SEL select  
Z <=  A or B  [ after 1 ns ]  when "00",  
      A or C  [ after 2 ns ]  when "01",  
      A or D  [ after 3 ns ]  when "10",  
      A or E  [ after 4 ns ]  when "11",  
      A or F  [ after 5 ns ]  when others;
```

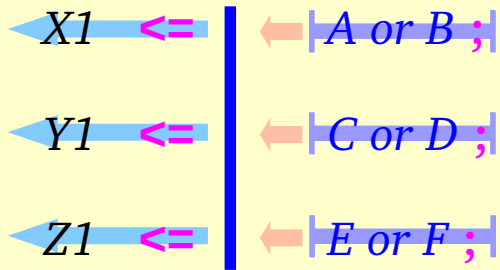
selection

# Concurrent vs Sequential

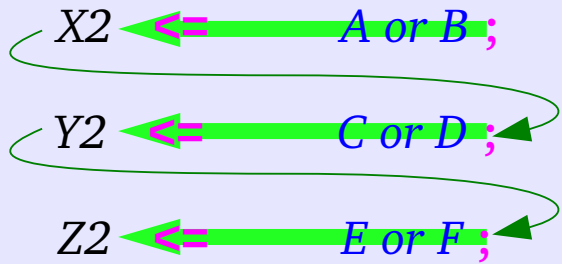


# Order of Statements

```
architecture arch of entity ent is  
    ...  
begin
```



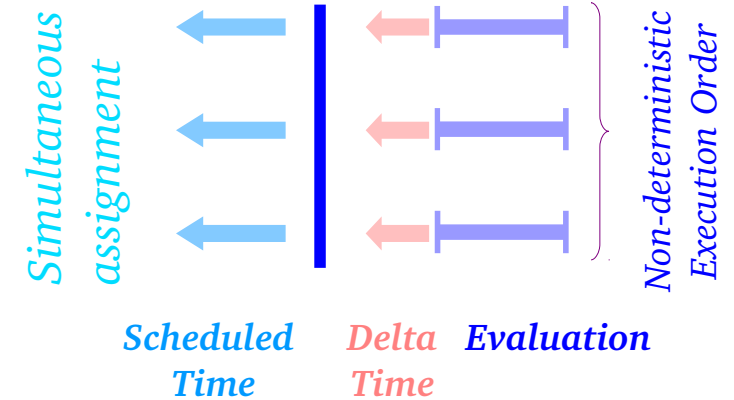
```
process (A, B, C, D, E, F)  
begin
```



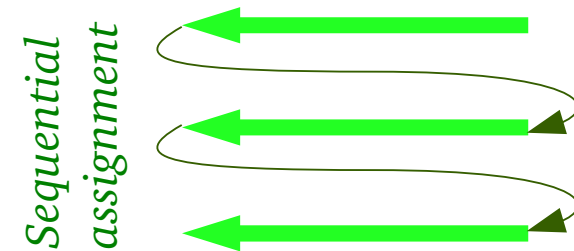
```
end process;
```

```
end
```

*Simulation of parallel activities*

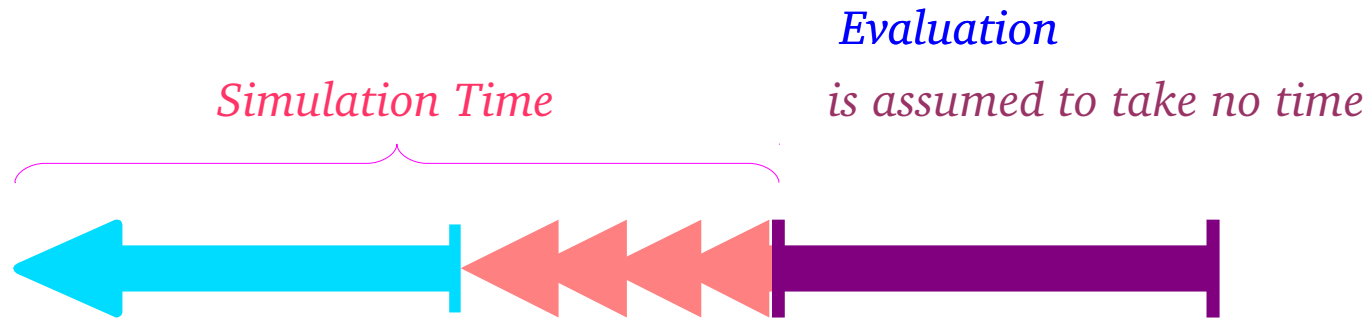


*The order of statements is important*





# Simulation Time



Unit: *ms, ns, ps, ...*

*Unitless Delta  $\Delta$*

***Real Delay***

– used for a simulator to  
mimic parallel activities  
*simulator*

$$1 \text{ ms} = 1000 \text{ ns}$$

$$1 \text{ ns} = 1000 \text{ ps}$$

$$1 \text{ ps} \neq n \cdot \Delta$$

*no integer  $n$  that make  $n$  delta  
equal to 1 ps.*

$$n \cdot \Delta = 0 \text{ ps} = 0 \text{ ns} \dots$$

***Zero Delay***

***Zero Delay Assignment***

```
X1 <= A or B ;
```

```
X1 <= A or B after 0 ns;
```

# Zero Delay Assignment

```
architecture arch of entity ent is
  ...
begin
```

```
X1 <= A or B;
Y1 <= C or D;
Z1 <= E or F;
```

```
process (A, B, C, D, E, F)
```

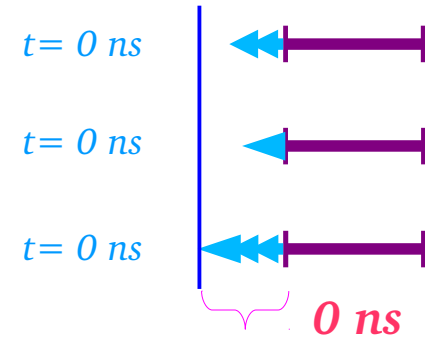
```
begin
```

```
X2 <= A or B;
Y2 <= C or D;
Z2 <= E or F;
```

```
end process;
```

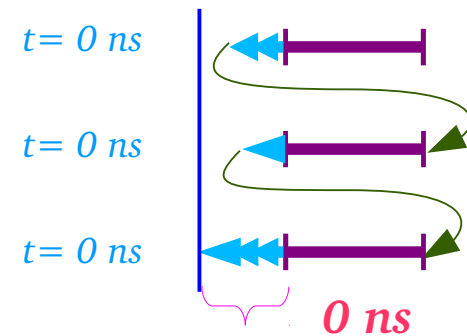
```
end
```

$\Delta$  time Evaluation



The exact no of delta is determined by the simulator and the context

$\Delta$  time Evaluation



Updated values

# Non-Zero Delay Assignment

architecture *arch* of entity *ent* is

begin

```
X1 <= A or B after 1 ns;
Y1 <= C or D after 3 ns;
Z1 <= E or F after 2 ns;
```

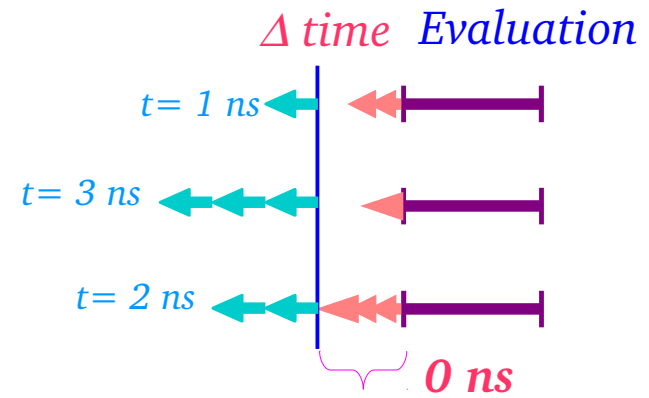
process (A, B, C, D, E, F)

begin

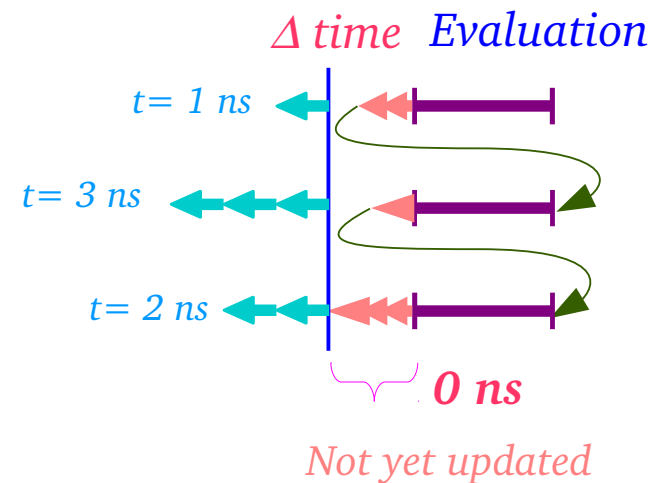
```
X2 <= A or B after 1 ns;
Y2 <= C or D after 3 ns;
Z2 <= E or F after 2 ns;
```

end process;

end

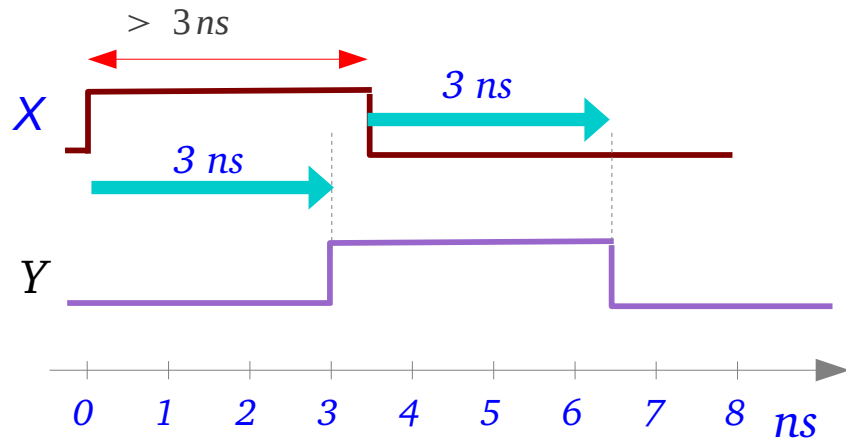


The exact no of delta is determined by the simulator and the context

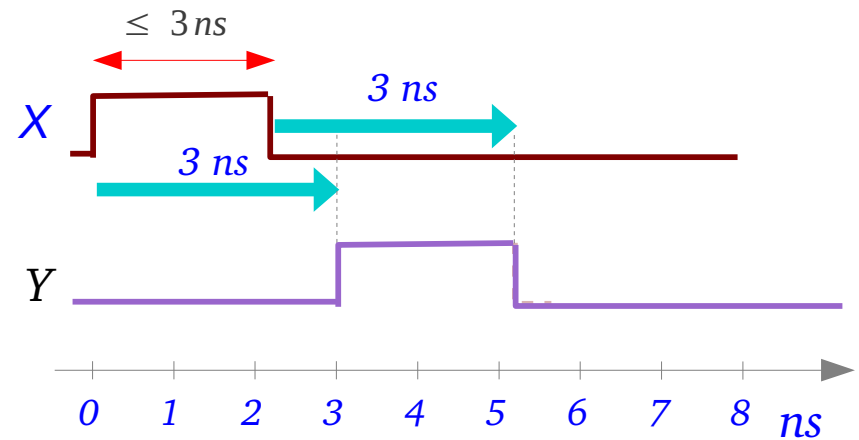
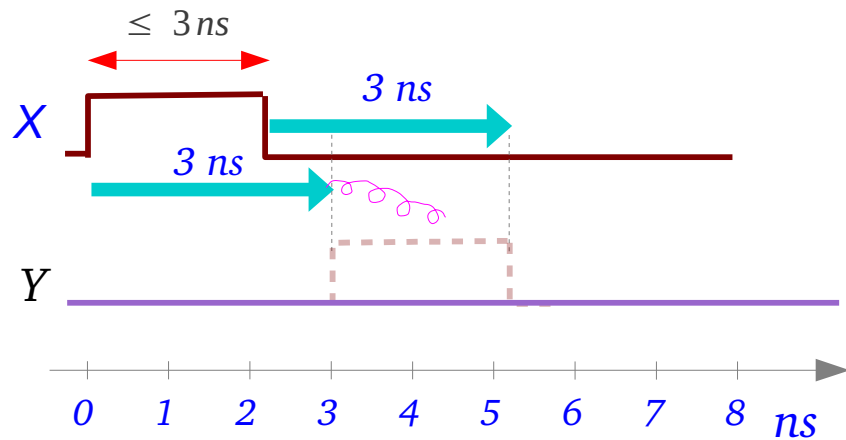
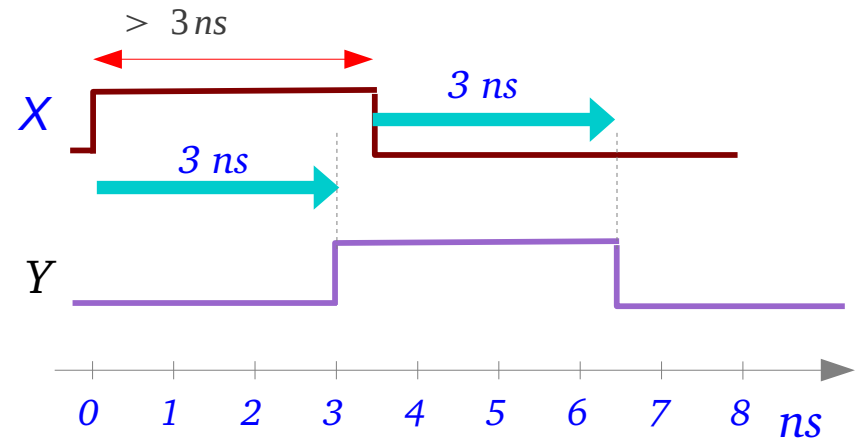


# Inertial Delay & Transport Delay

```
Y <= X after 3 ns;
```



```
Y <= transport X after 3 ns;
```



# Variable Assignment (1)

```
architecture arch of entity ent is
```

•••

```
begin
```

```
  X1 <= A or B after 1 ns;
```

```
  Y1 <= C or D after 3 ns;
```

```
  Z1 := E or F ;
```

```
  process (A, B, C, D, E, F)
```

```
    variable Y2, Z2 : bit;
```

```
  begin
```

```
    X2 <= A or B after 1 ns;
```

```
    Y2 := C or D after 3 ns;
```

```
    Z2 := E or F ;
```

```
  end process;
```

```
end
```

The Variable assignment is a sequential statement and cannot be used outside a process statement.

The variable is **declared** here. They are used for **local storage** in process statement and subprogram body.

The variable assignment has nothing to do with time. It executes immediately. It can not have an **after** clause

# Variable Assignment (2)

```
process (A, B, C, D, E, F)
```

```
variable Z2 : bit;
```

```
begin
```

```
X2 <= A or B after 1 ns;
```

```
Y2 <= C or D after 3 ns;
```

```
Z2 := E or F ;
```

```
end process;
```

```
process (A, B, C, D, E, F)
```

```
variable Y2 : bit;
```

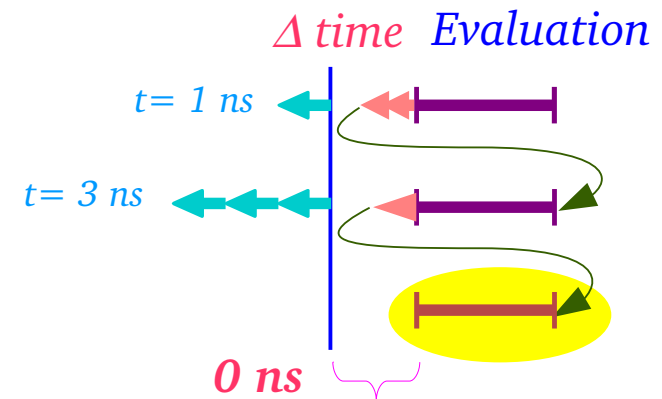
```
begin
```

```
X2 <= A or B after 1 ns;
```

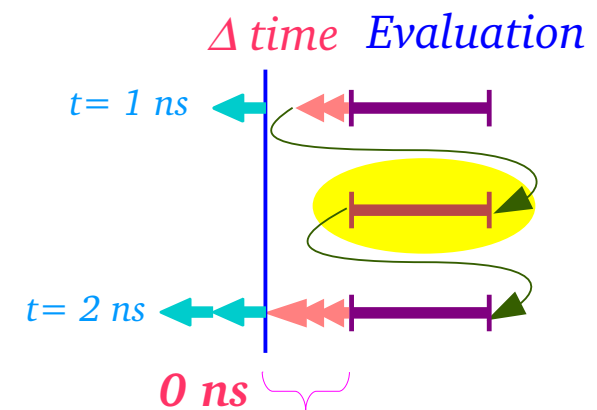
```
Y2 := C or D ;
```

```
Z2 <= E or F after 2 ns;
```

```
end process;
```



The variable assignment has nothing to do with time. It executes immediately.



# Multiple Assignments to the Same Target

```
architecture arch of entity ent is
    ...
begin
    X1 <= A or B ;
    X1 <= C or D ;

    process (A, B, C, D, E, F)
        variable Z2 : bit;
    begin
        Y2 <= A or B ;
        Y2 <= C or D ;

        Z2 := A or B ;
        Z2 := C or D ;
    end process;
end
```

## Multiple Concurrent Assignments

Multiple Concurrent Assignment is legal only when a resolution function is defined.  
(wire-and, wire-or)

## Multiple Sequential Assignments

- Overwrite
- Append
- Keep

## Multiple Variable Assignments

Variable Z2 has the result of the latest assignments (The new assignment overwrites the old one)

# Multiple Sequential Assignments

```
architecture arch of entity ent is
```

```
    ...
```

```
begin
```

```
    X1 <= A or B ;
```

```
    X1 <= C or D ;
```

```
    process (A, B, C, D, E, F)
```

```
        variable Z2 : bit;
```

```
    begin
```

```
        Y2 <= A or B ;
```

```
        Y2 <= C or D ;
```

```
        Z2 := A or B ;
```

```
        Z2 := C or D ;
```

```
    end process;
```

```
end
```

## Multiple Concurrent Assignments

*Multiple Concurrent Assignment is legal only when a resolution function is defined.*

*(wire-and, wire-or)*

## Multiple Sequential Assignments

- *Overwrite*
- *Append*
- *Keep*

## Multiple Variable Assignments

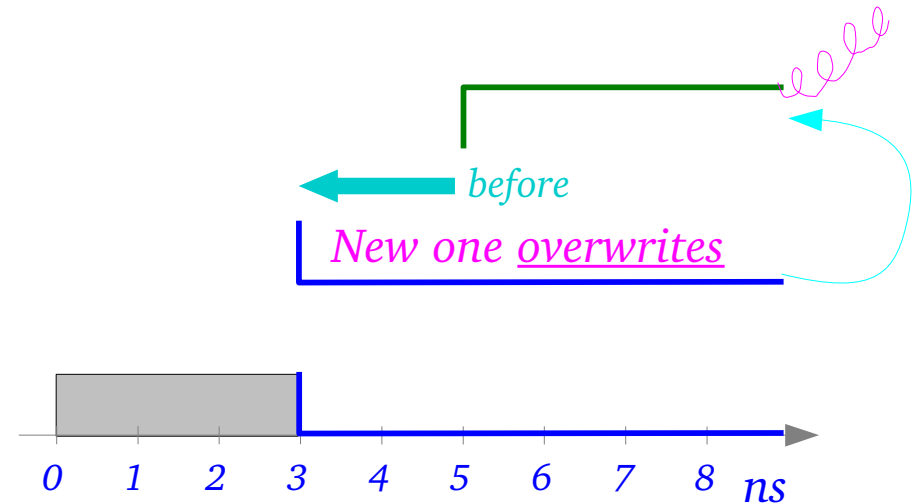
*Variable Z2 has the result of the latest assignments (The new assignment overwrites the old one)*



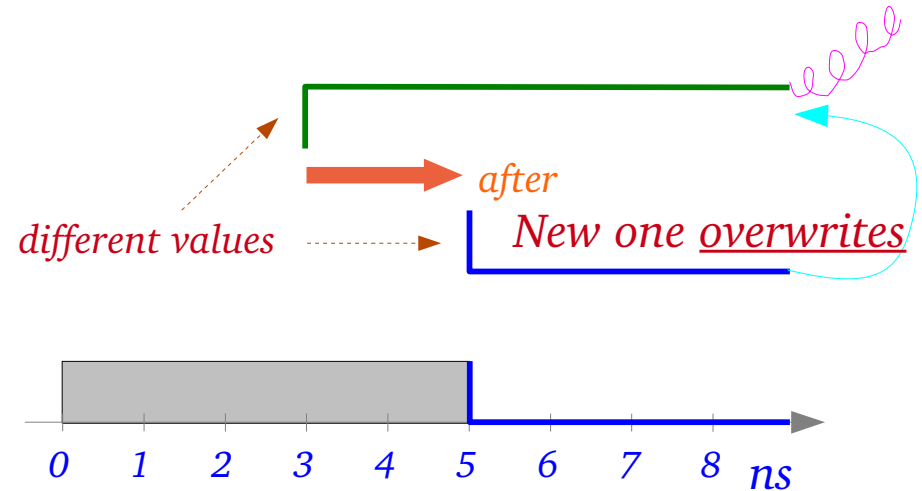
# Inertial Delay (1)

## Multiple Sequential Assignments

```
process (...)  
begin  
  
    X2 <= '1' after 5 ns;  
    X2 <= '0' after 3 ns;  
  
end process;
```



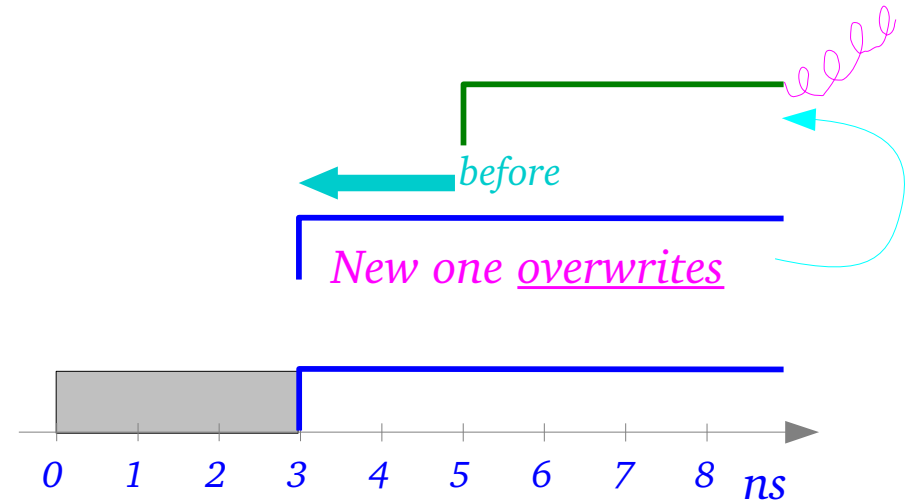
```
process (...)  
begin  
  
    X2 <= '1' after 3 ns;  
    X2 <= '0' after 5 ns;  
  
end process;
```



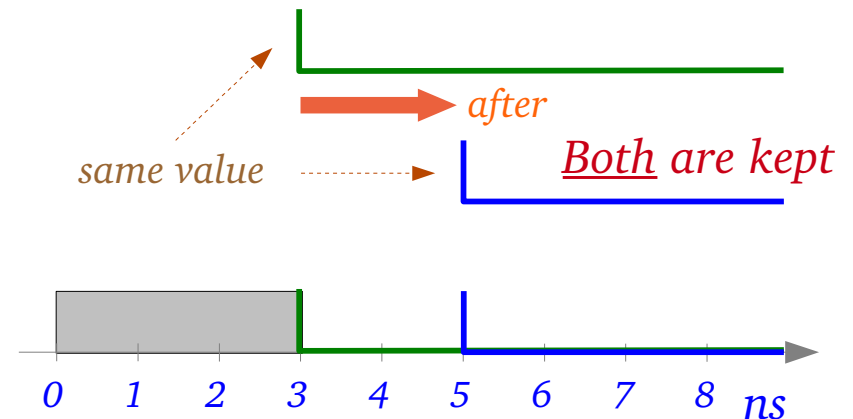
# Inertial Delay (2)

## Multiple Sequential Assignments

```
process (...)  
begin  
  
    X2 <= '1' after 5 ns;  
    X2 <= '1' after 3 ns;  
  
end process;
```



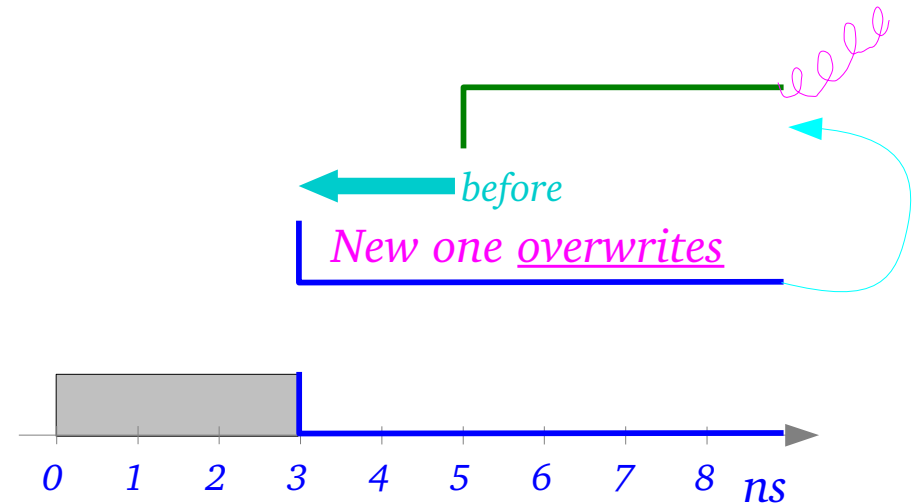
```
process (...)  
begin  
  
    X2 <= '0' after 3 ns;  
    X2 <= '0' after 5 ns;  
  
end process;
```



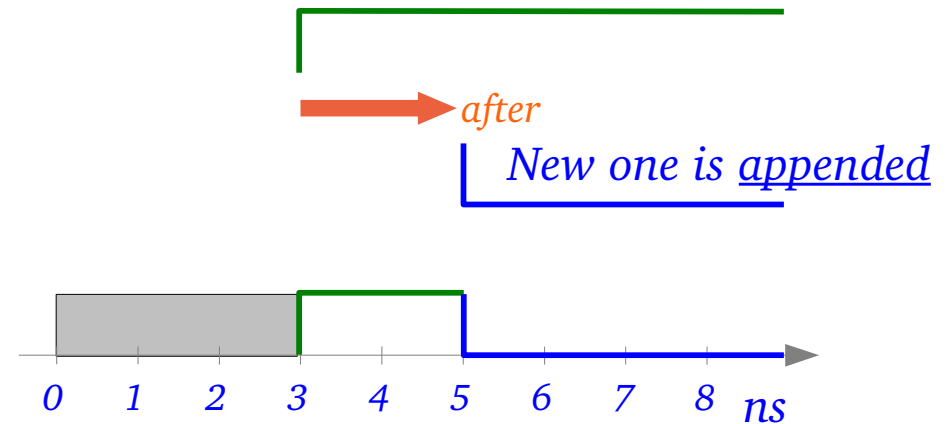
# Transport Delay (1)

## Multiple Sequential Assignments

```
process (...)  
begin  
  
    X2 <= transport '1' after 5 ns;  
    X2 <= transport '0' after 3 ns;  
  
end process;
```



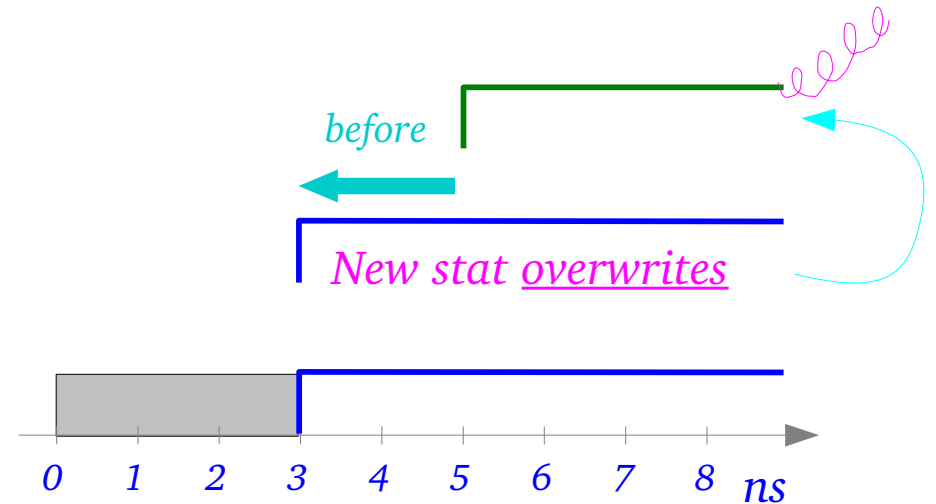
```
process (...)  
begin  
  
    X2 <= transport '1' after 3 ns;  
    X2 <= transport '0' after 5 ns;  
  
end process;
```



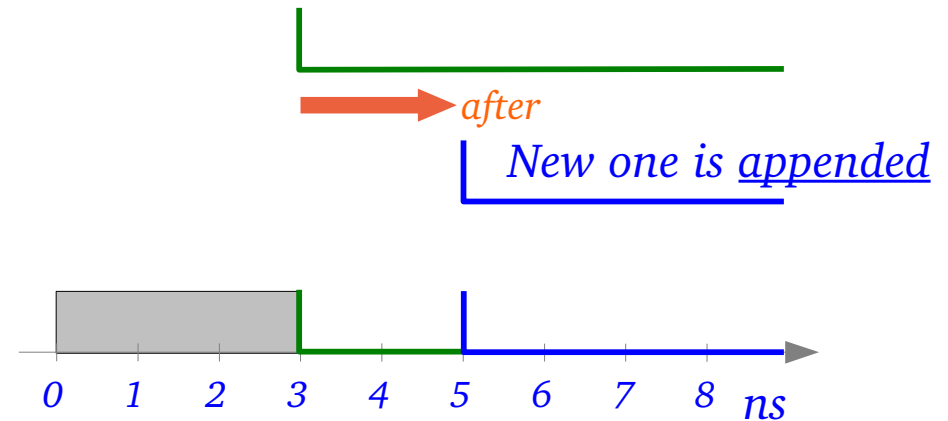
# Transport Delay (2)

## Multiple Sequential Assignments

```
process (...)  
begin  
  
    X2 <= transport '1' after 5 ns;  
    X2 <= transport '1' after 3 ns;  
  
end process;
```



```
process (...)  
begin  
  
    X2 <= transport '0' after 3 ns;  
    X2 <= transport '0' after 5 ns;  
  
end process;
```



# Inertial Delay

## Multiple Sequential Assignments - Inertial Delay

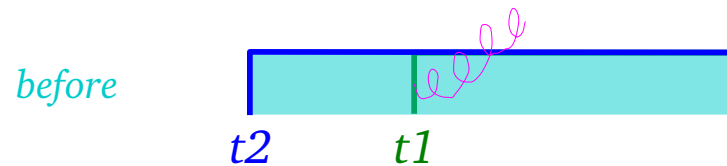
```
process (...)  
begin  
  
    X2 <= v1 after t1 ns;  
    X2 <= v2 after t2 ns;  
  
end process;
```

$t2 < t1$   $v2 = v1$  *New one overwrites*

$v2 \neq v1$  *New one overwrites*

$t1 < t2$   $v1 = v2$  *Both are kept*

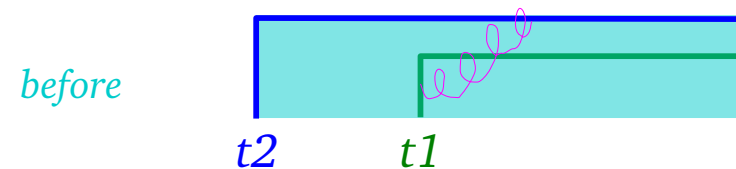
$v1 \neq v2$  *New one overwrites*



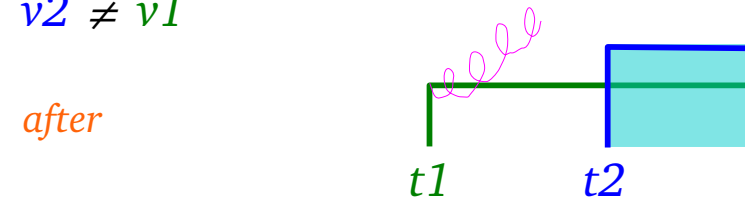
$t2 < t1$  *New one overwrites*  
 $v2 = v1$



$t1 < t2$  *Both are kept*  
 $v1 = v2$



$t2 < t1$  *New one overwrites*  
 $v2 \neq v1$



$t1 < t2$  *New one overwrites*  
 $v1 \neq v2$

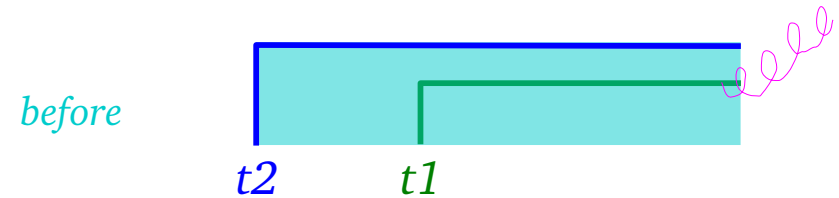
# Transport Delay

## Multiple Sequential Assignments - Transport Delay

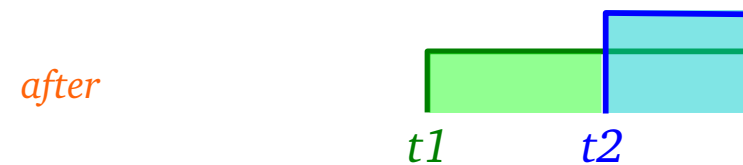
```
process (...)  
begin  
  
    X2 <= transport v1 after t1 ns;  
    X2 <= transport v2 after t2 ns;  
  
end process;
```

$t2 < t1$     *New stat overwrites*

$t1 < t2$     *New stat is appended*



$t2 < t1$     *New one overwrites*



$t1 < t2$     *New one is appended*

# Inertial & Transport Delay Model (1)

## Inertial Delay

The simulation time of a new event	
Before the time of an old one	
New one <u>overwrites</u>	
After the time of an old one	
For the <b>same</b> value	Both are kept
For <b>different</b> values	New one overwrites

$t2 < t1$	$v2 = v1$	New one overwrites
	$v2 \neq v1$	New one overwrites
$t1 < t2$	$v1 = v2$	Both are kept
	$v1 \neq v2$	New one overwrites

## Transport Delay

The simulation time of a new event	
Before the time of an old one	
New one <u>overwrites</u>	
After the time of an old one	
New stat is <u>appended</u>	

$t2 < t1$	New stat <u>overwrites</u>
$t1 < t2$	New stat is <u>appended</u>

# Inertial & Transport Delay Model (2)

## Inertial Delay

The simulation time of a new event

Before the time of an old one

New one overwrites

After the time of an old one

For the **same** value

Both are kept

For **different** values

New one overwrites

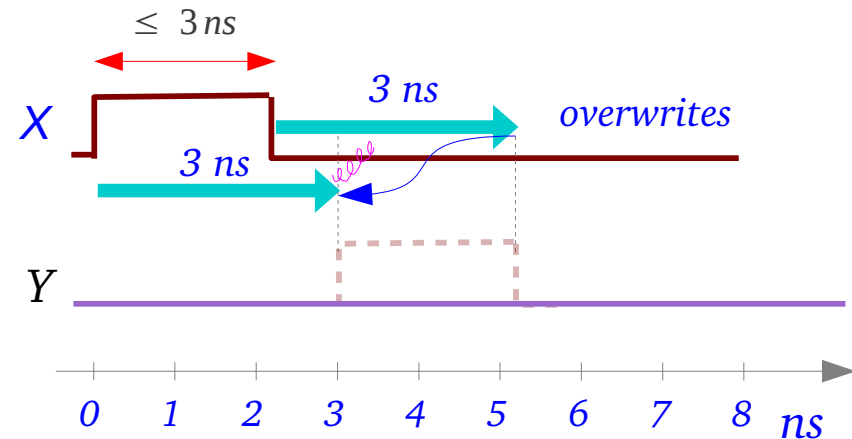
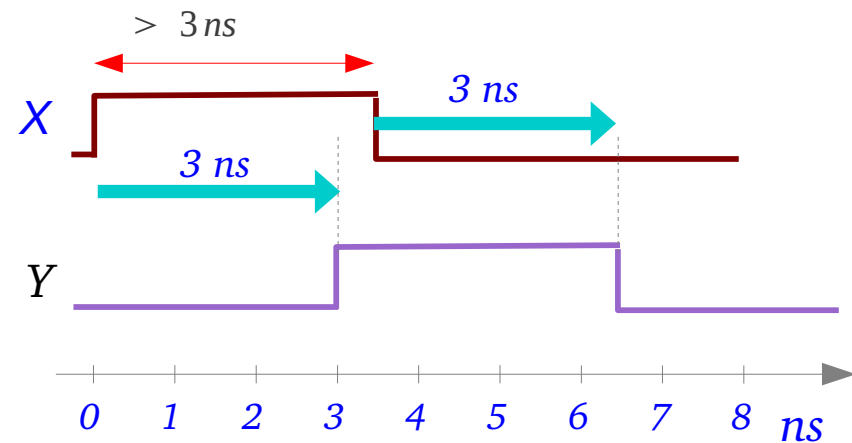
$t2 < t1$   $v2 = v1$  New one overwrites

$v2 \neq v1$  New one overwrites

$t1 < t2$   $v1 = v2$  Both are kept

$v1 \neq v2$  New one overwrites

$Y \leq X$  after 3 ns;





# Inertial & Transport Delay Model (3)

## Transport Delay

The simulation time of a new event

Before the time of an old one

New one overwrites

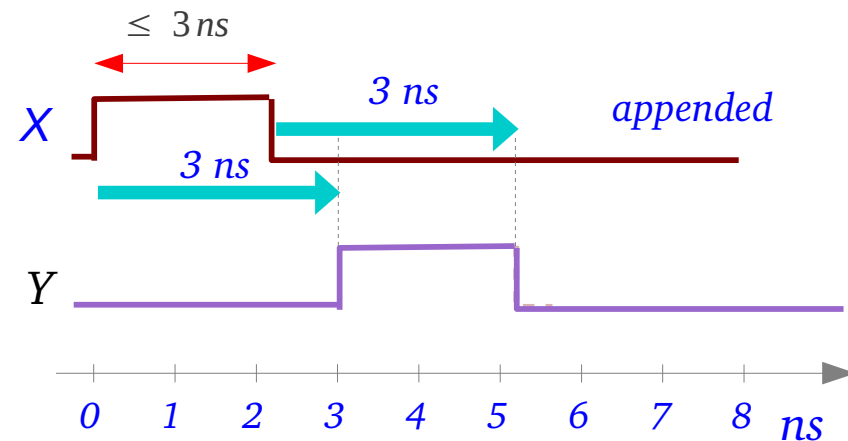
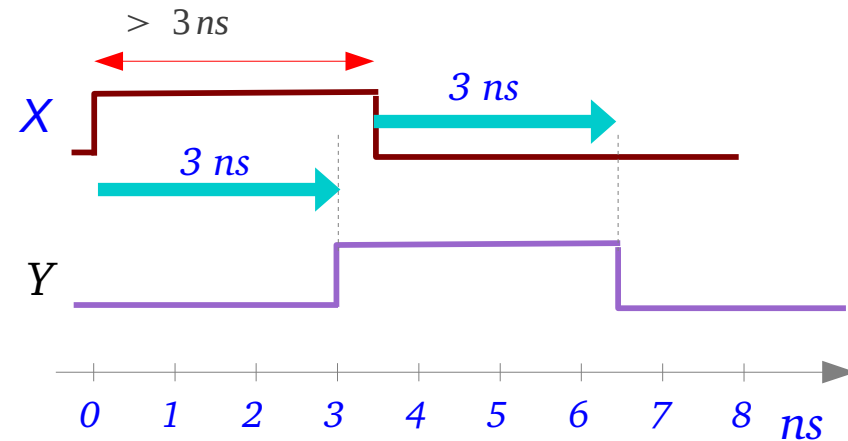
After the time of an old one

New stat is appended

$t_2 < t_1$  New stat overwrites

$t_1 < t_2$  New stat is appended

```
Y <= transport X after 3 ns;
```



# Multiple Concurrent Assignments

```
architecture arch of entity ent is
    ...
begin
    X1 <= A or B ;
    X1 <= C or D ;

    process (A, B, C, D, E, F)
        variable Z2 : bit;
    begin
        Y2 <= A or B ;
        Y2 <= C or D ;

        Z2 := A or B ;
        Z2 := C or D ;
    end process;
end
```

## Multiple Concurrent Assignments

Multiple Concurrent Assignment is legal only when a resolution function is defined.  
(wire-and, wire-or)

## Multiple Sequential Assignments

- Overwrite
- Append
- Keep

## Multiple Variable Assignments

Variable Z2 has the result of the latest assignments (The new assignment overwrites the old one)

# Resolution Function

## architecture *arch* of entity *ent* is

```
FUNCTION w_and (drivers : bit_vector) RETURN bit is  
BEGIN
```

```
    ...
```

```
END w_and;
```

```
SIGNAL X1 : w_and bit;
```

```
    ...
```

## begin

```
X1 <= A or B ;
```

```
X1 <= C or D ;
```

```
process (A, B, C, D, E, F)
```

```
begin
```

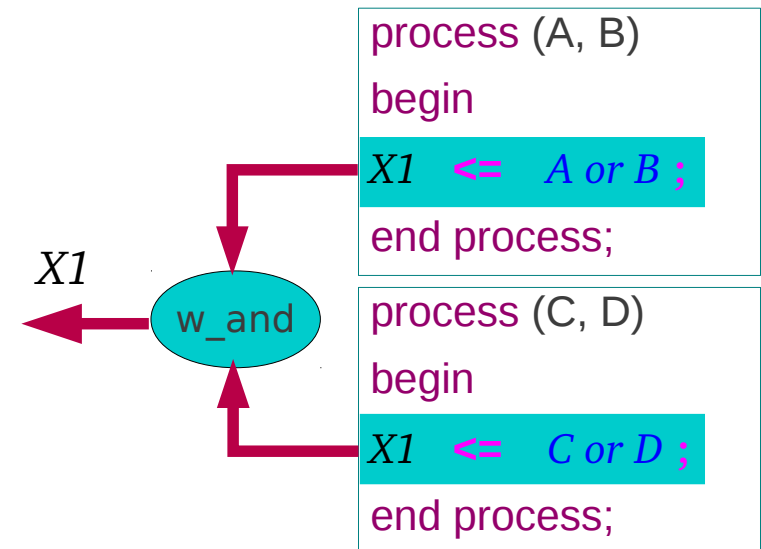
```
    ...
```

```
end process;
```

## end

*Multiple Concurrent Assignment is legal only when a resolution function is defined.*


*(wire-and, wire-or)*



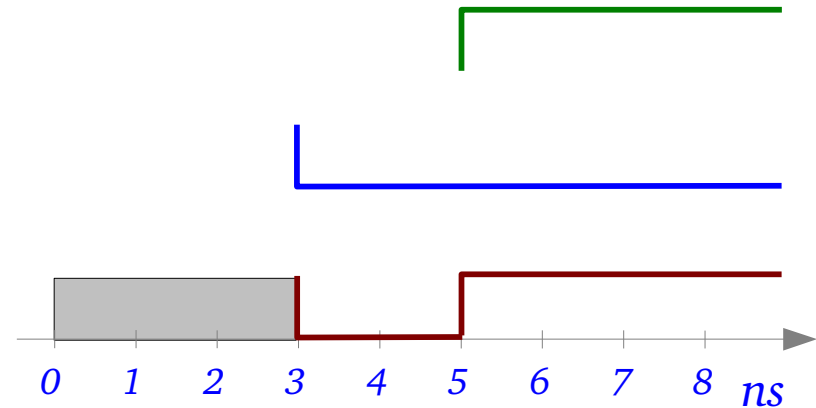
*X1 <= w\_and (A or B, C or D) ;*


# Inertial Delay

## Multiple Concurrent Assignments

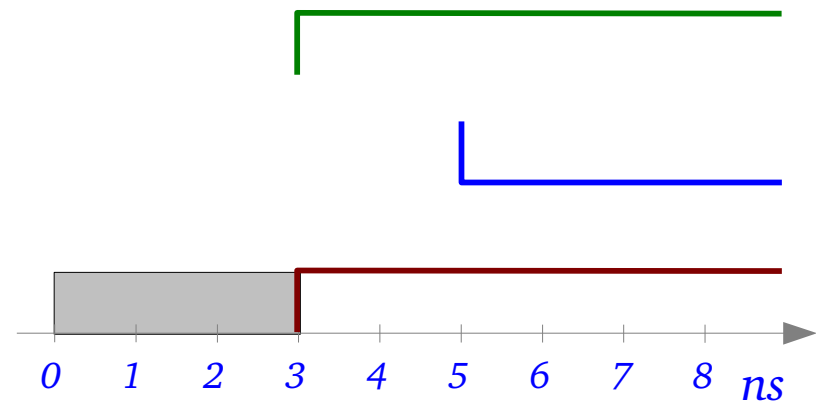
 X2 <= '1' after 5 ns; Wire-or resolution function  
X2 <= '0' after 3 ns; resolution function

```
process (...)  
begin  
    ...  
end process;
```



 X2 <= '1' after 3 ns; Wire-or resolution function  
X2 <= '0' after 5 ns; resolution function

```
process (...)  
begin  
    ...  
end process;
```

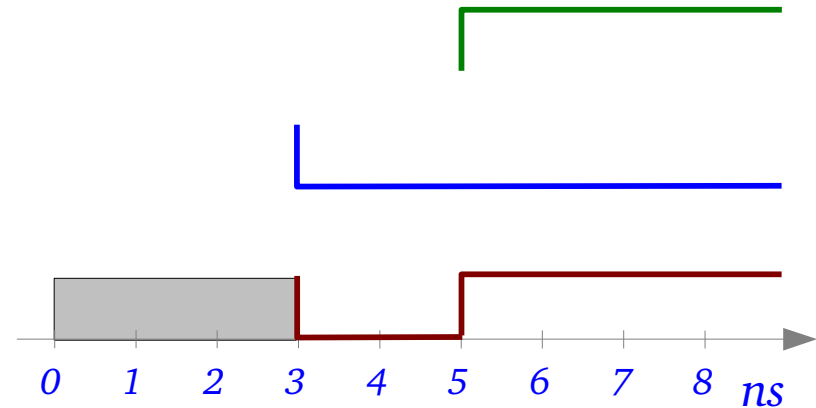


# Transport Delay

## Multiple Concurrent Assignments

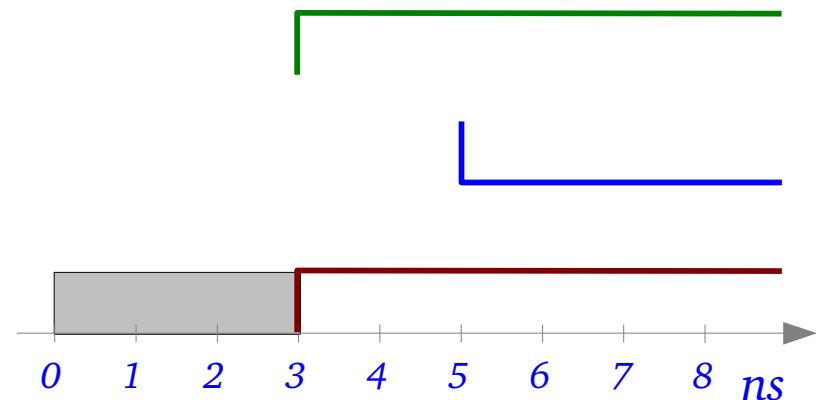
```
X2 <= transport '1' after 5 ns;  
X2 <= transport '0' after 3 ns;  
  
process (...)  
begin  
  ...  
end process;
```

*Wire-or resolution function*




```
X2 <= transport '1' after 3 ns;  
X2 <= transport '0' after 5 ns;  
  
process (...)  
begin  
  ...  
end process;
```

*Wire-or resolution function*

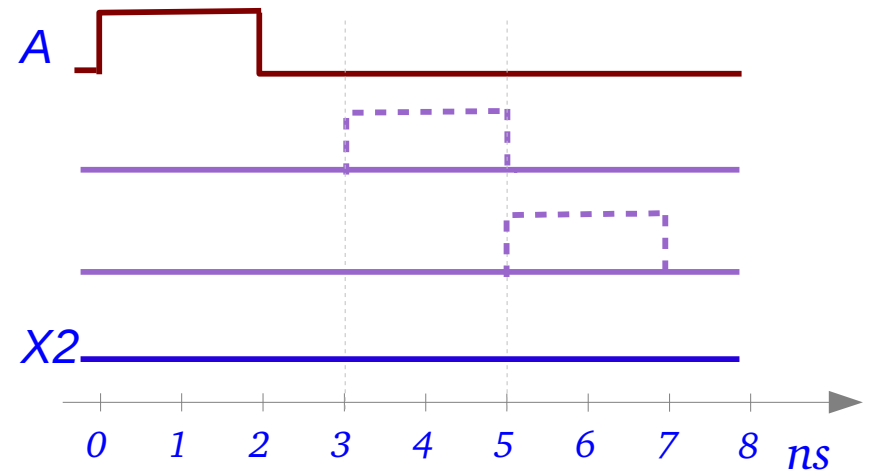



# Inertial Delay

## Multiple Concurrent Assignments

 `X2 <= A after 5 ns;` *Wire-or*  
`X2 <= A after 3 ns;` *resolution*  
*function*

```
process (...)  
begin  
    ...  
end process;
```




 `X2 <= A after 3 ns;` *Wire-or*  
`X2 <= A after 5 ns;` *resolution*  
*function*

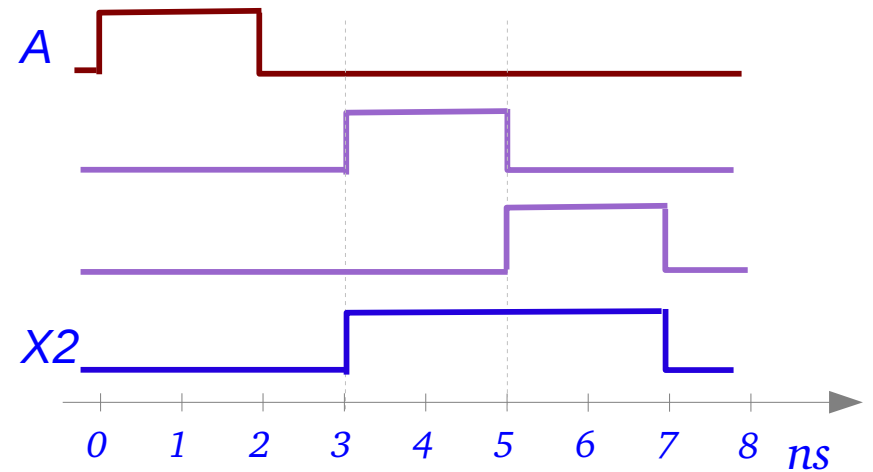
```
process (...)  
begin  
    ...  
end process;
```


# Transport Delay

## Multiple Concurrent Assignments

 `X2 <= A after 5 ns;` *Wire-or resolution function*  
`X2 <= B after 3 ns;` *resolution function*

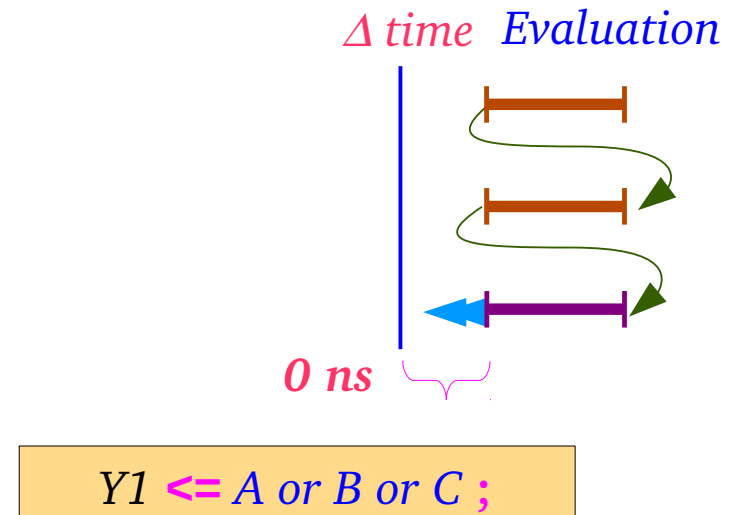
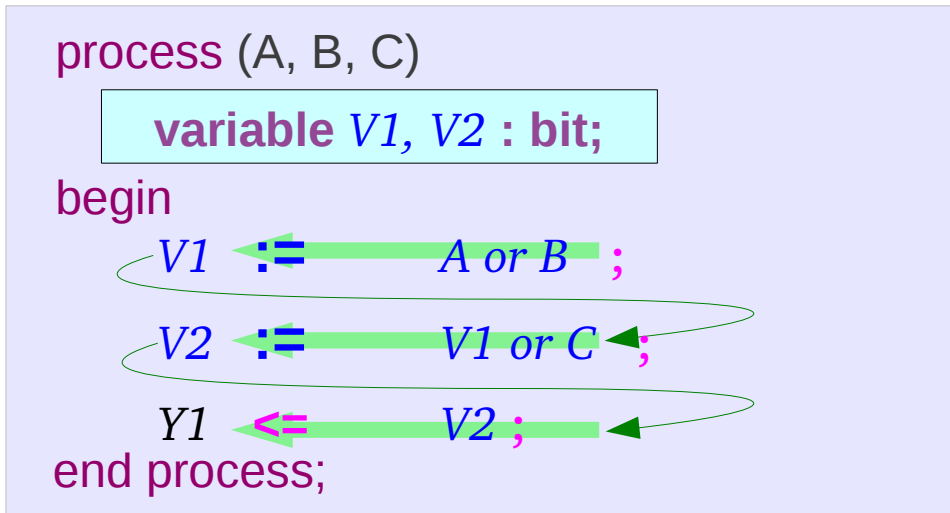
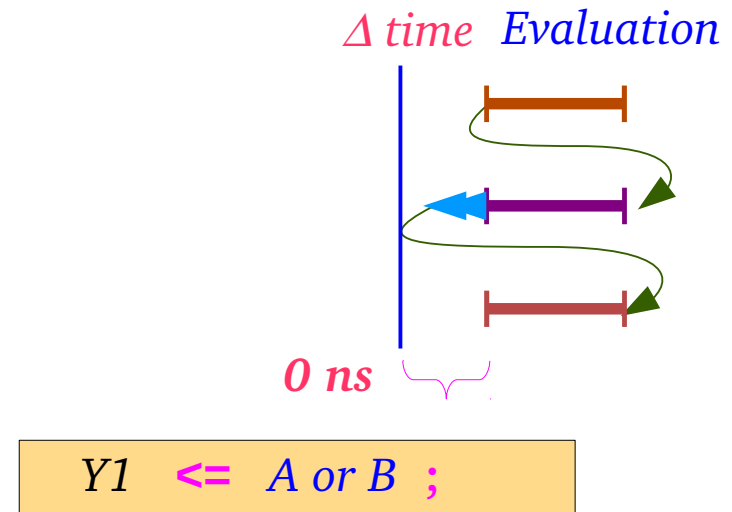
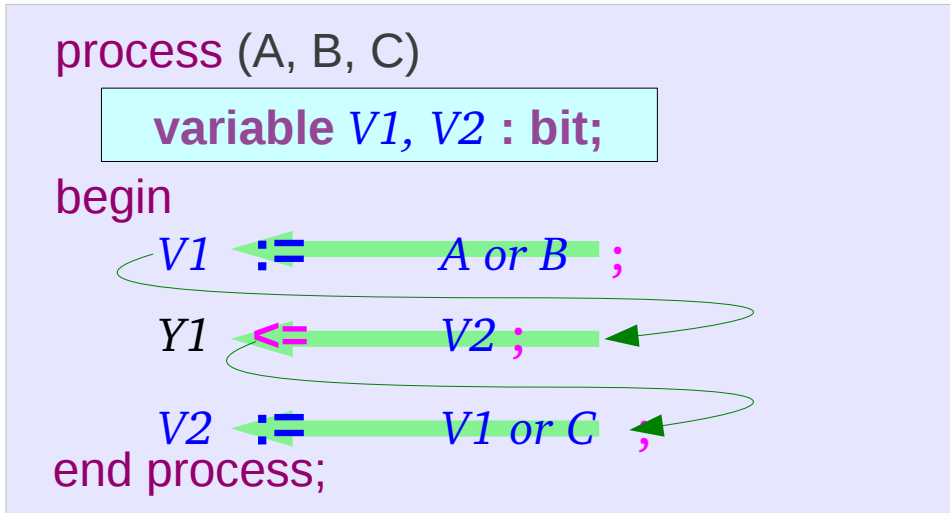
```
process (...)  
begin  
    ...  
end process;
```



 `X2 <= A after 3 ns;` *Wire-or resolution function*  
`X2 <= B after 5 ns;` *resolution function*

```
process (...)  
begin  
    ...  
end process;
```

# Signals & Variable Assignment Example 1





## References

- [1] <http://en.wikipedia.org/>
- [2] J. V. Spiegel, VHDL Tutorial,  
[http://www.seas.upenn.edu/~ese171/vhdl/vhdl\\_primer.html](http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html)
- [3] J. R. Armstrong, F. G. Gray, Structured Logic Design with VHDL
- [4] Z. Navabi, VHDL Analysis and Modeling of Digital Systems
- [5] D. Smith, HDL Chip Design
- [6] <http://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html>
- [7] VHDL Tutorial - VHDL online [www.vhdl-online.de/tutorial/](http://www.vhdl-online.de/tutorial/)