

Signals & Variables (1A)

Concurrent & Sequential Signal Assignments

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.

This document was produced by using OpenOffice and Octave.

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

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

Concurrent 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

Conditional Signal Assignment (1)

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

← simple concurrent statement

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

← One condition

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

← One condition with 'else'

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

← Two conditions with 'else'

Concurrent Signal Assignment

- Conditional Signal Assignment
- Selected Signal Assignment

Conditional Signal Assignment (2)

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

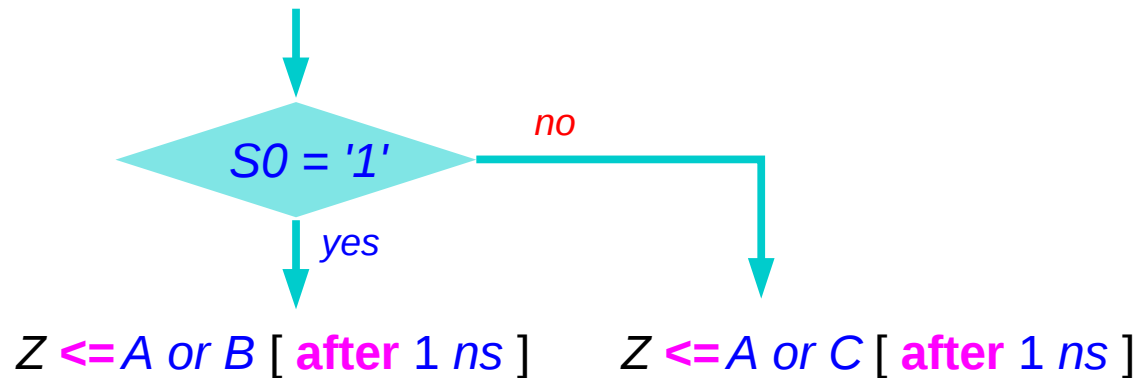
← simple concurrent statement

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

← One condition

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

← One condition with 'else'



Conditional Signal Assignment (3)

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

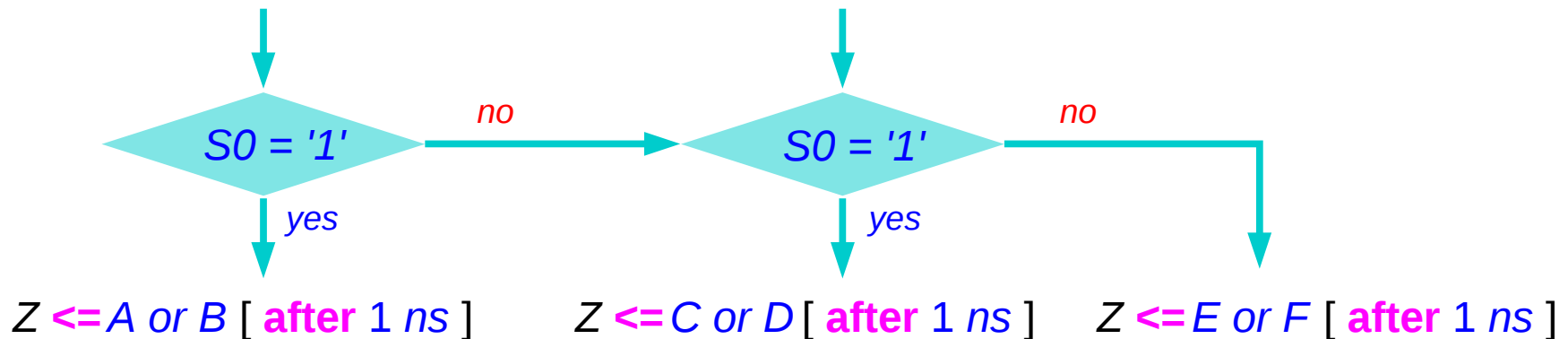
← simple concurrent statement

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

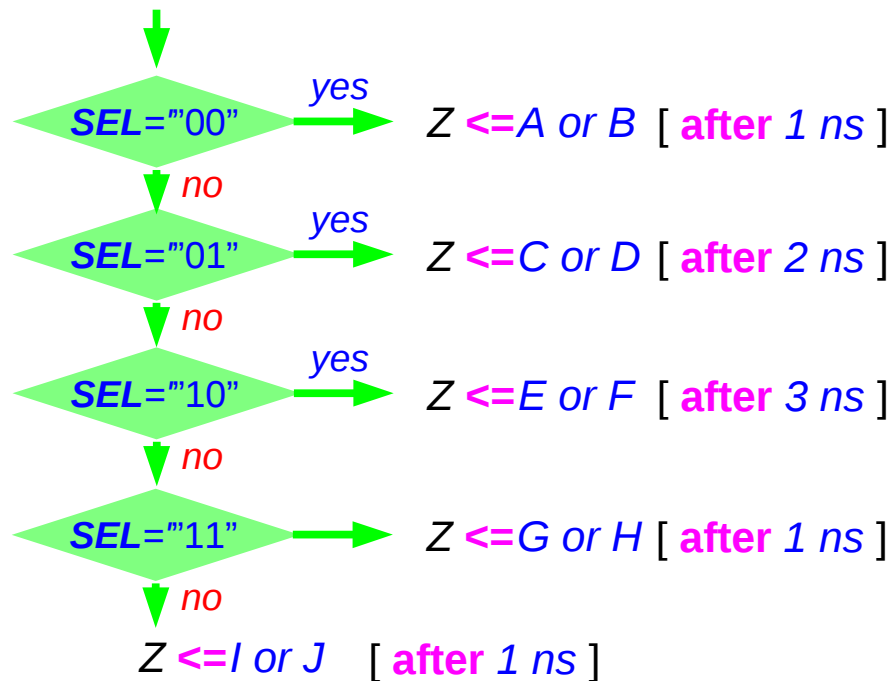
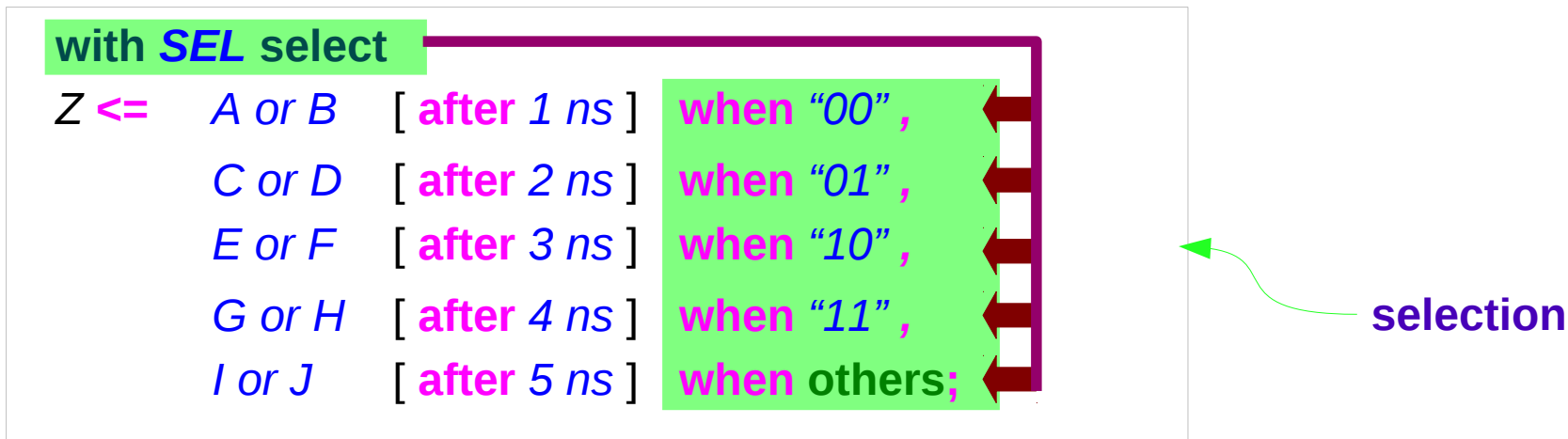
← One condition

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

← Two conditions with 'else'



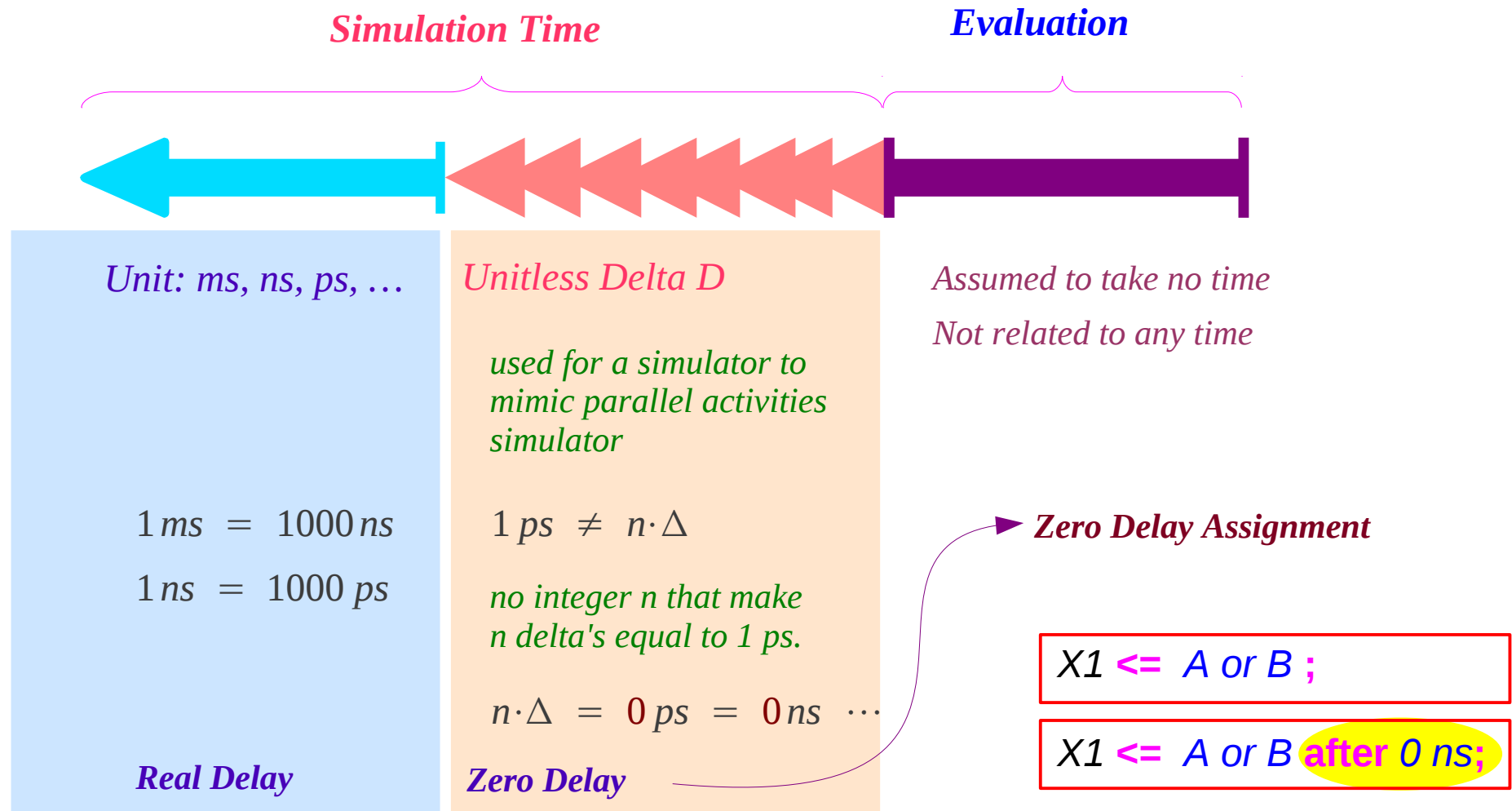
Selected Signal Assignment



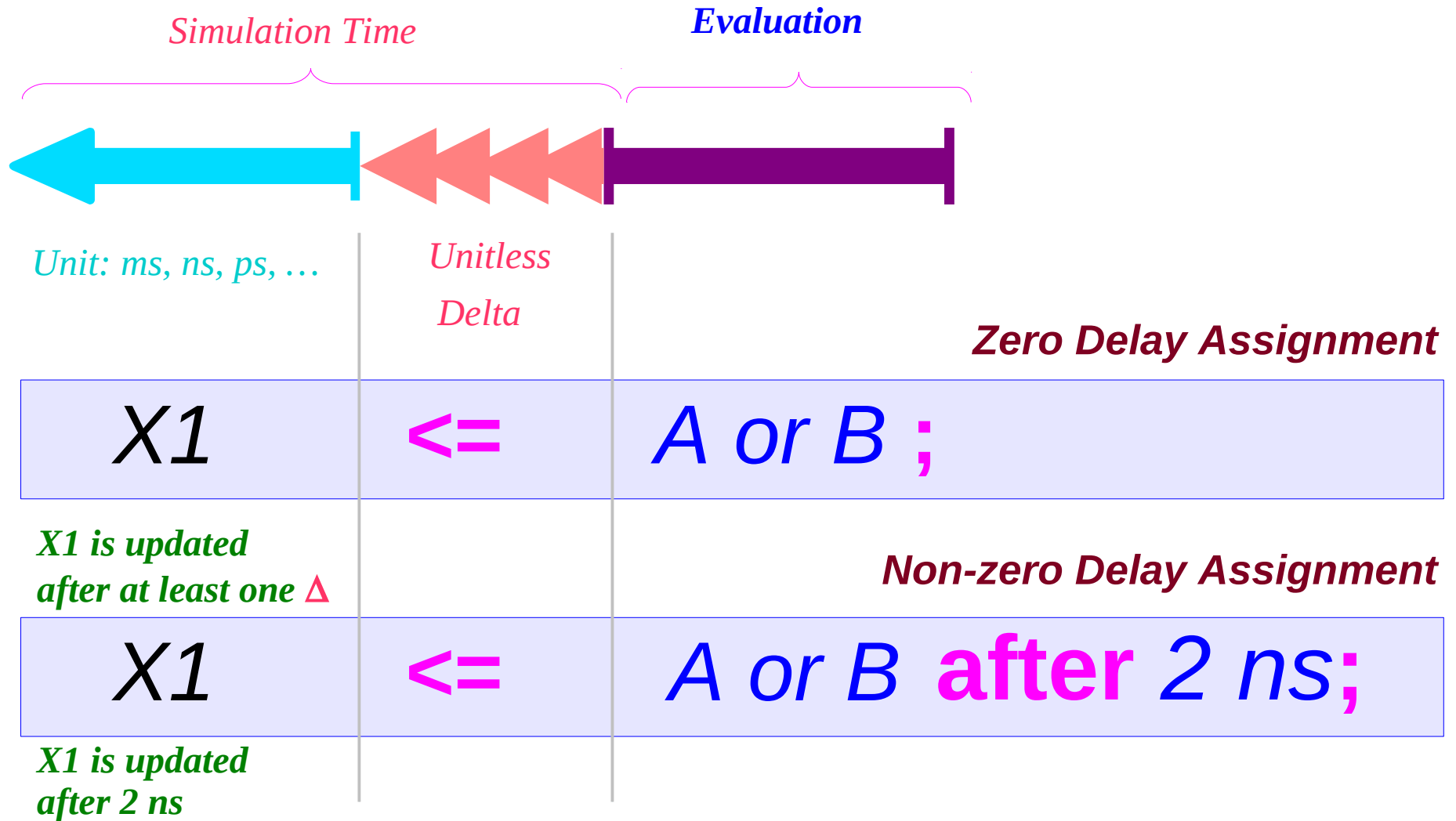
Concurrent Signal Assignment

- Conditional Signal Assignment
- Selected Signal Assignment

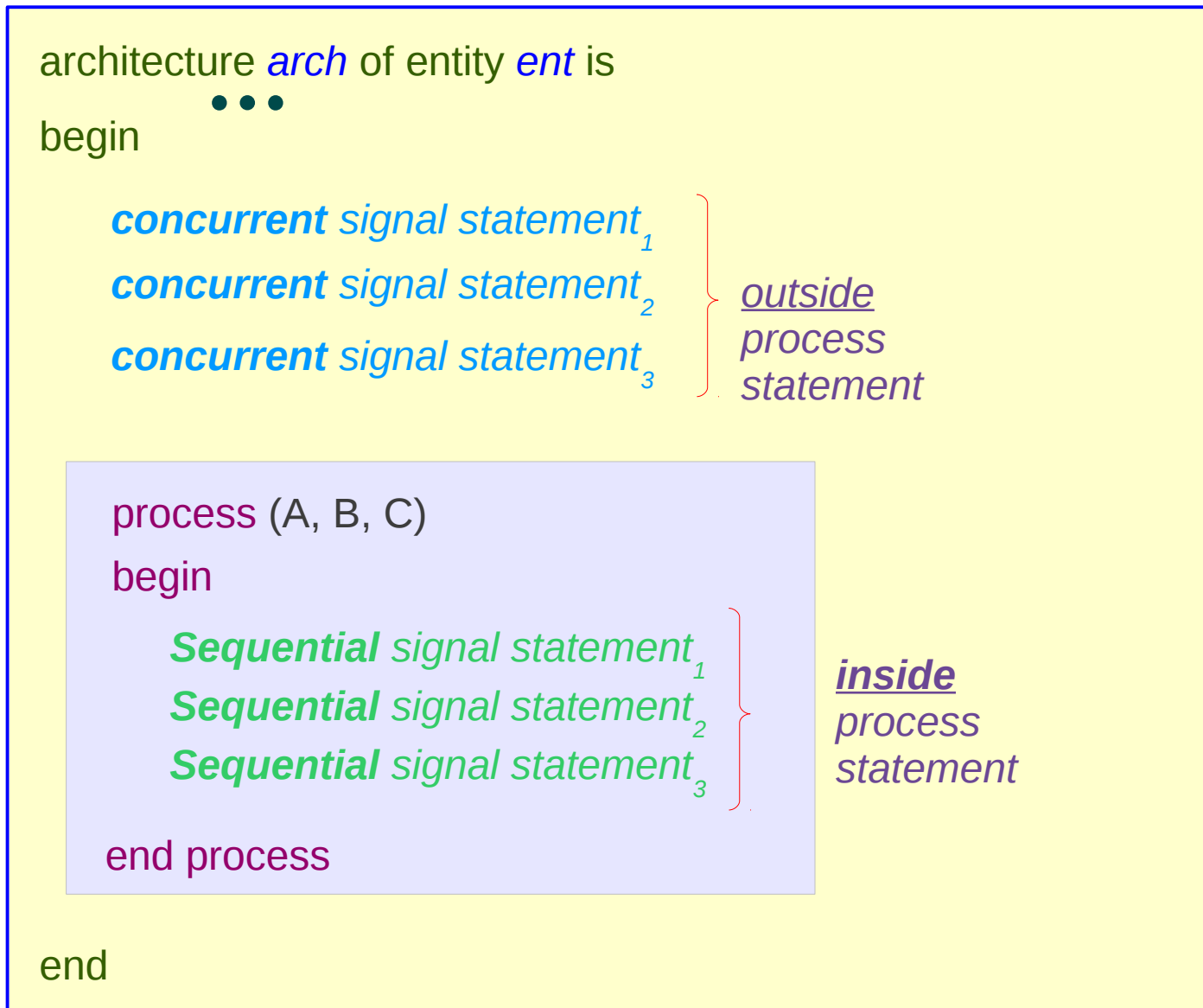
Simulation Time (1)



Simulation Time (2)

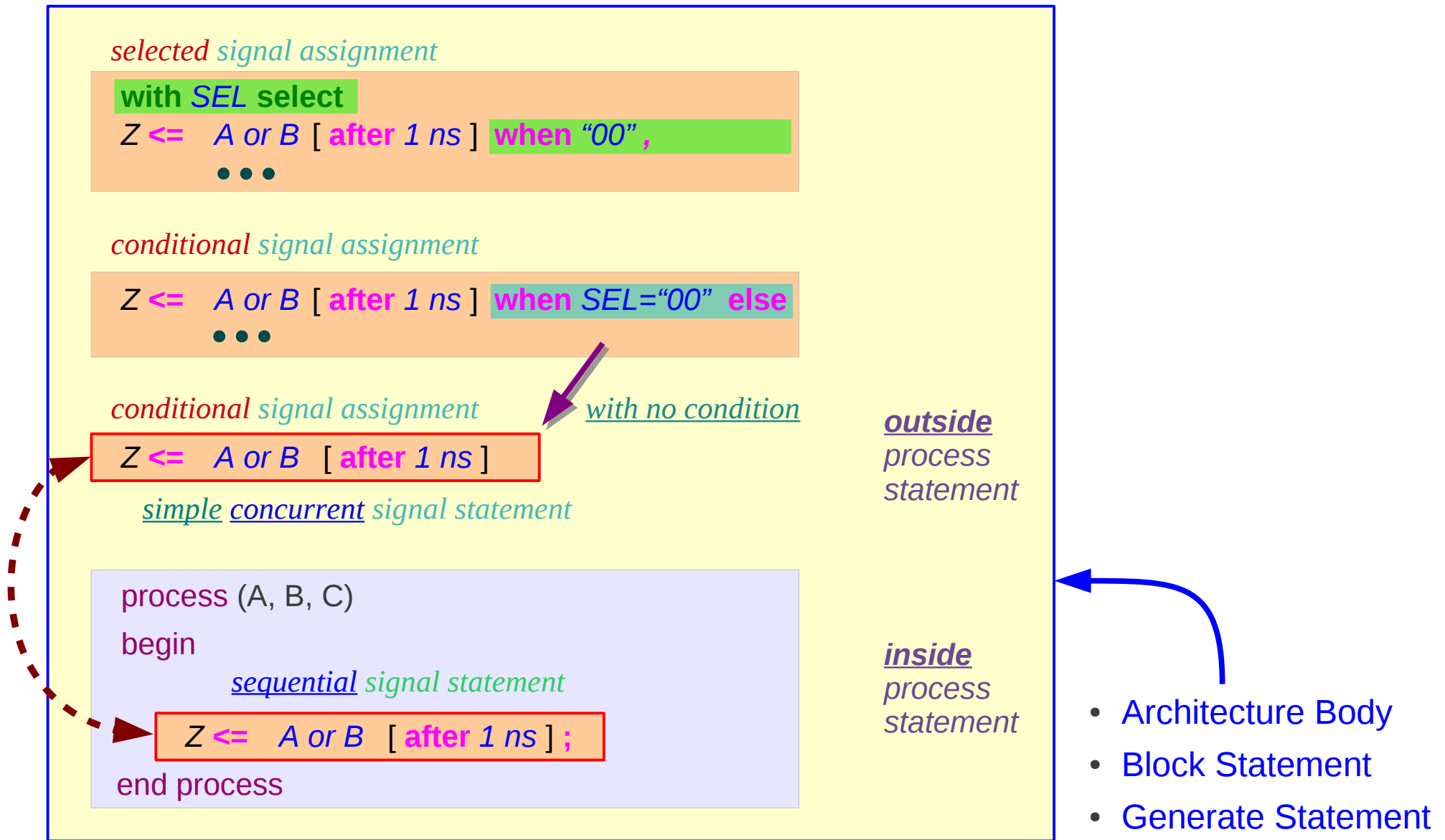


Concurrent vs Sequential (1)

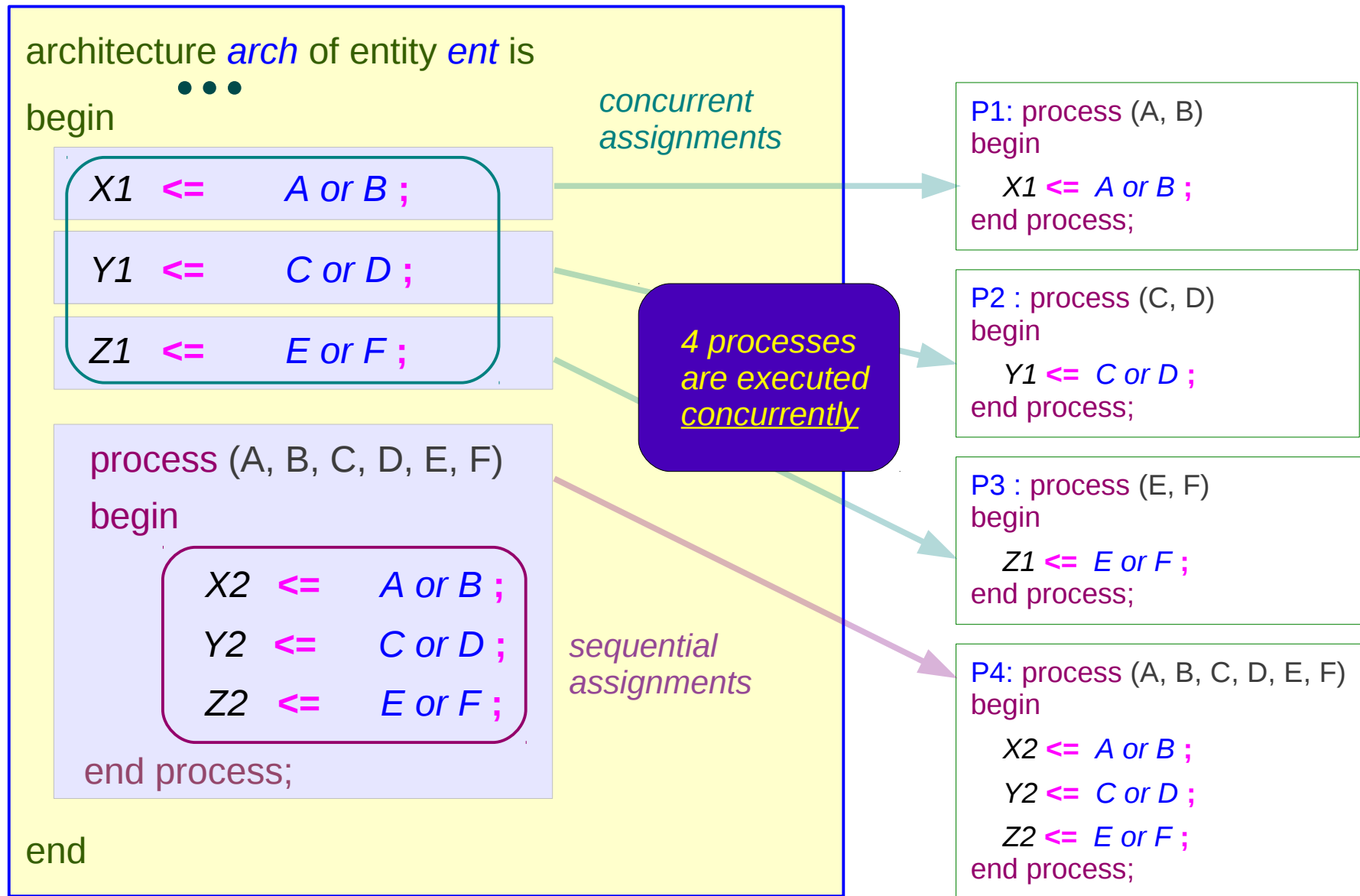


- Architecture Body
- Block Statement
- Generate Statement

Concurrent vs Sequential (2)

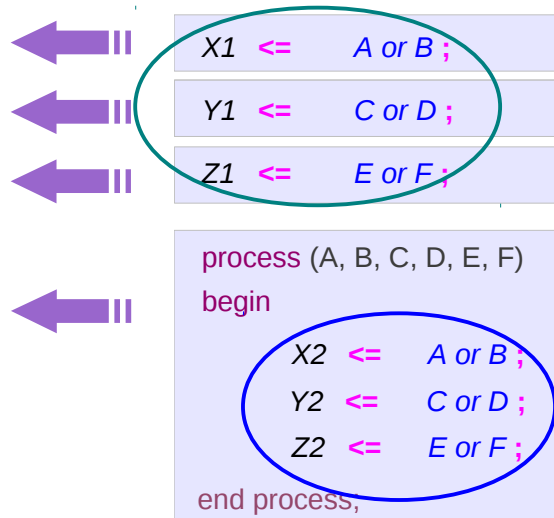


Concurrent vs Sequential (3)



Concurrent vs Sequential (4)

Simulation of parallel activities



Concurrent assignments

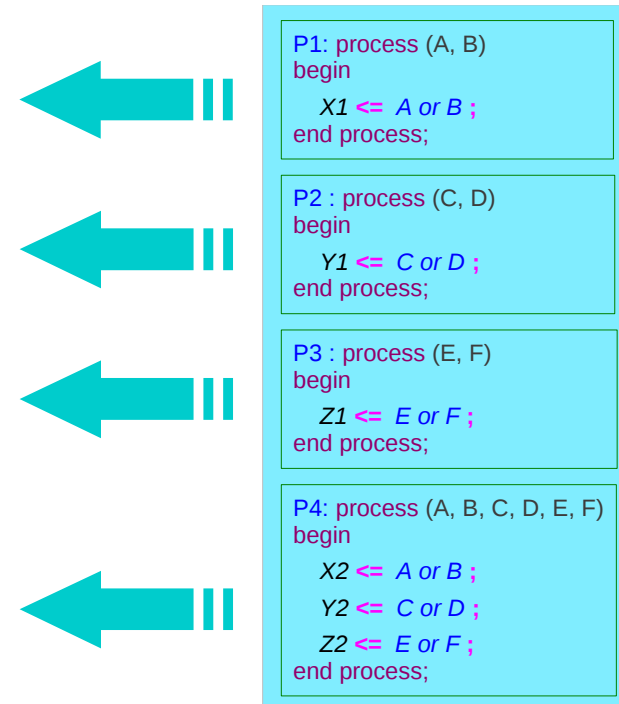
Sequential assignments

The order of statements is important

4 processes are executed concurrently

Non-deterministic Execution Order

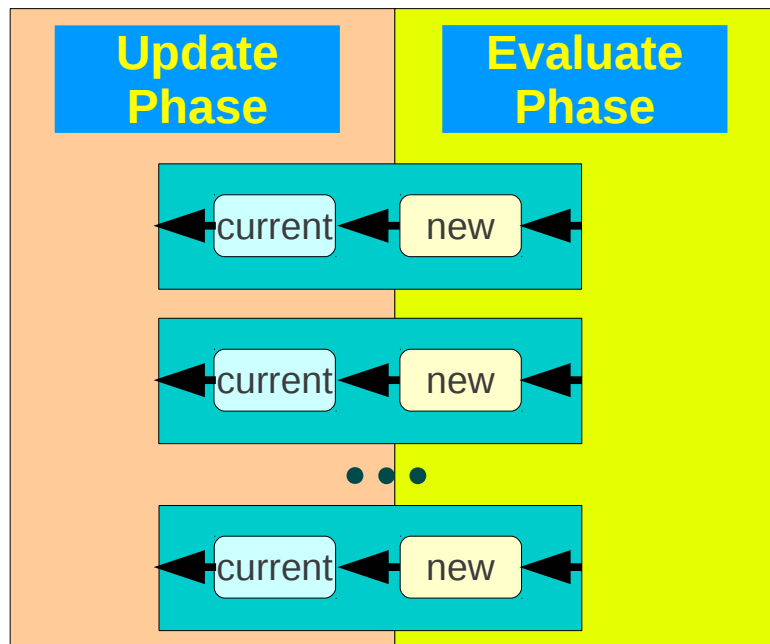
Don't know which process executes first among P1 ~ P4.



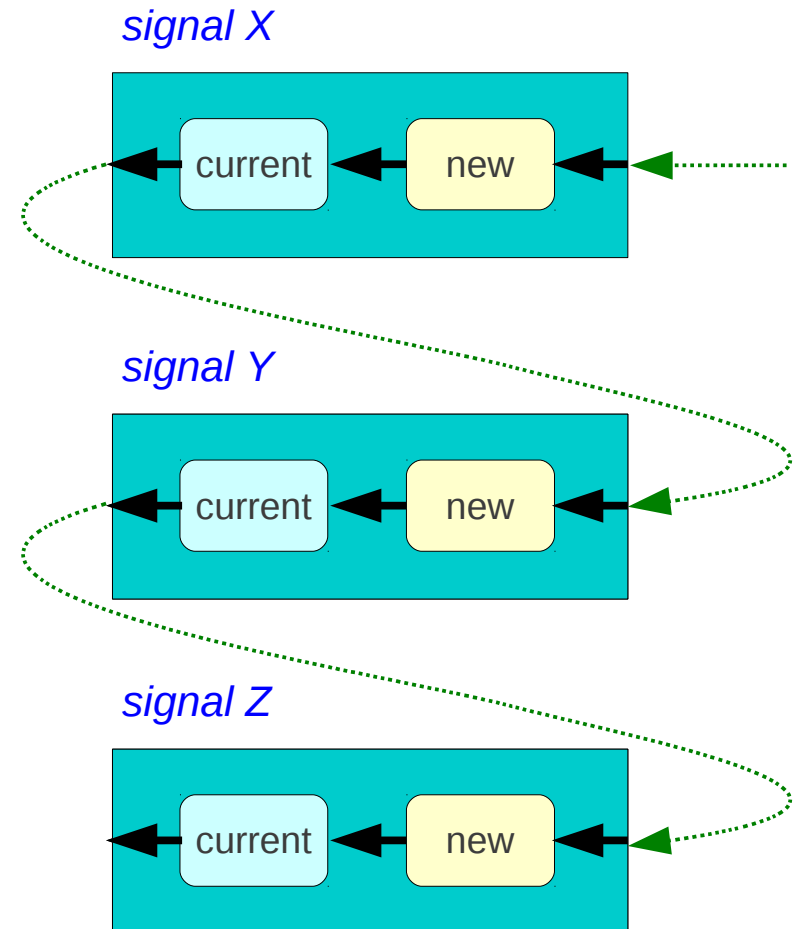
Evaluate - Update

When *X* or *Y* is changed, the assignments are evaluated using the current values, not the updated values of *X* or *Y*

```
process (X, Y)
begin
    Y <= X;
    Z <= Y;
end process;
```



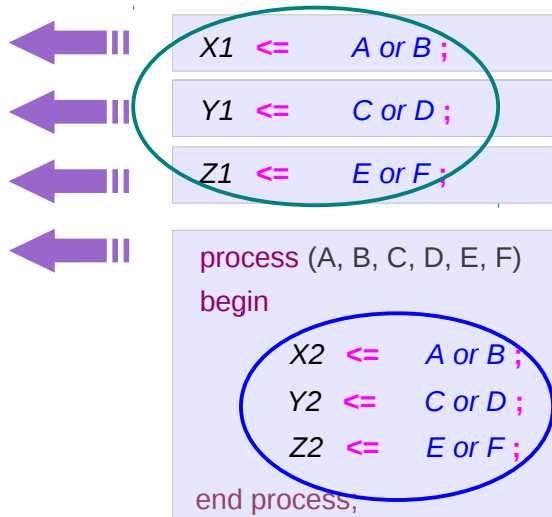
Non-Blocking Assignments



Zero vs Non-zero Delay Assignments (1)

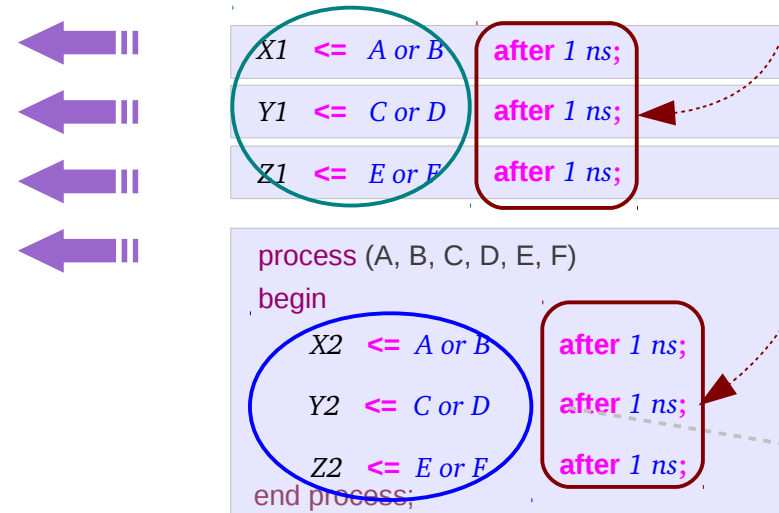
When A, B, C, D, E, or F is changed, the assignments are evaluated using the current values, not the updated values of A, B, C, D, E, F

zero delay assignments



Updated values are observable after at least one delta time

nonzero delay assignments

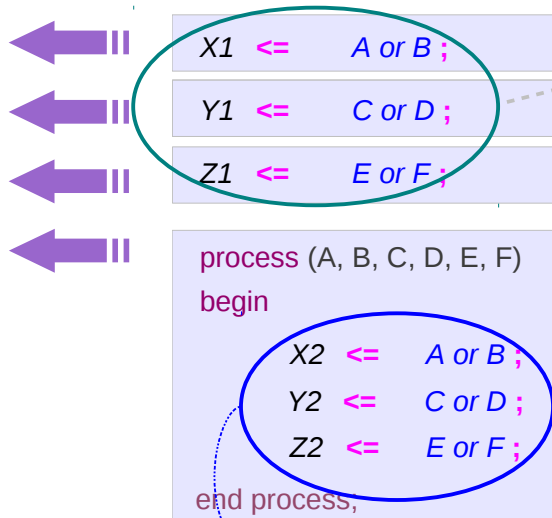


Updated values are observable after 1 ns

Zero vs Non-zero Delay Assignments (2)

When A, B, C, D, E, or F is changed, the assignments are evaluated using the current values, not the updated values of A, B, C, D, E, F

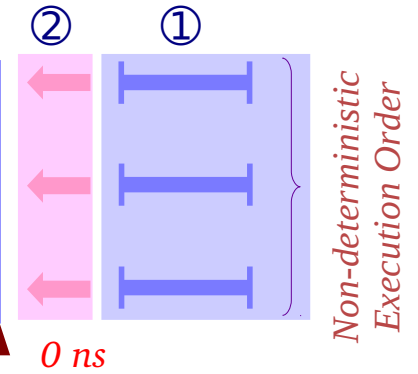
zero delay assignments



- * The order of statements is important
1. scheduled after some delta time
 2. non-blocking assignment

only lumped view
②①

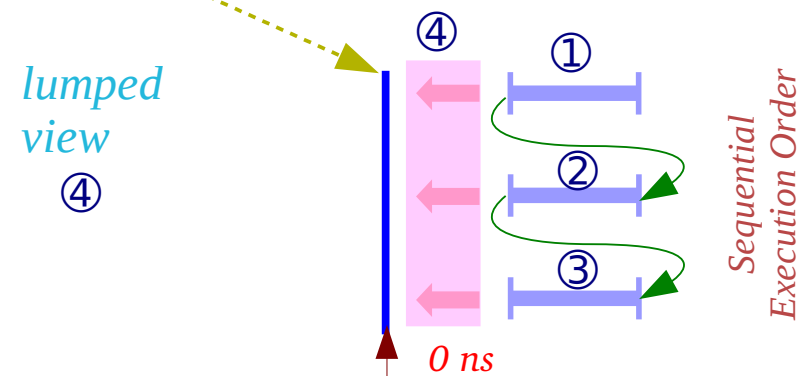
Concurrent assignments



Updated values are observable after at least one delta time



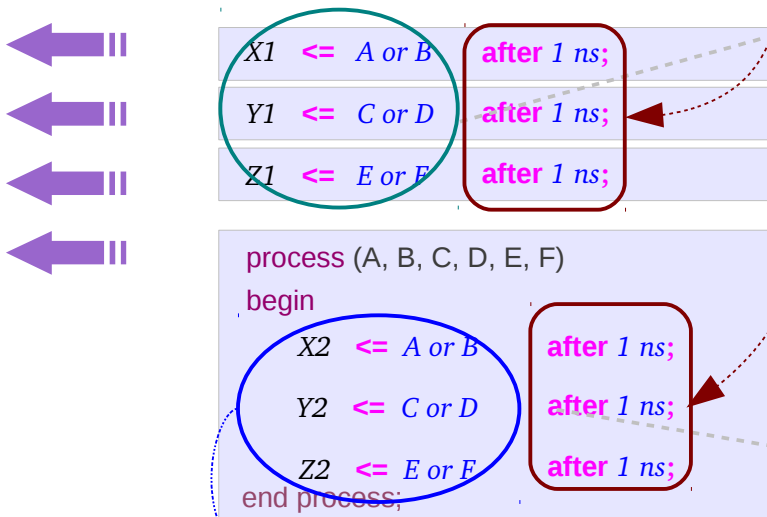
Sequential assignments



Zero vs Non-zero Delay Assignments (3)

When A, B, C, D, E, or F is changed, the assignments are evaluated using the current values, not the updated values of A, B, C, D, E, F

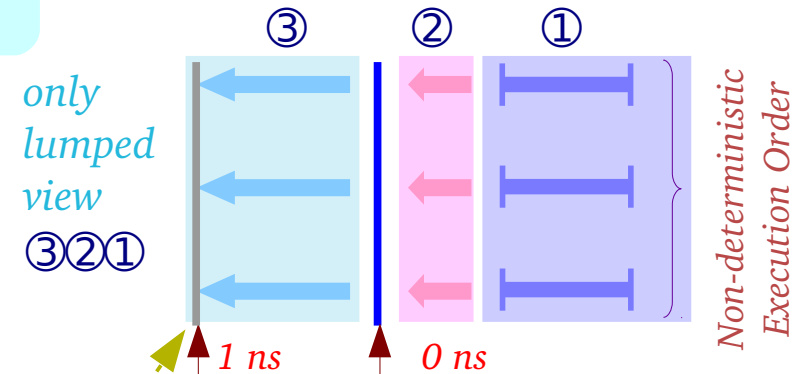
nonzero delay assignments



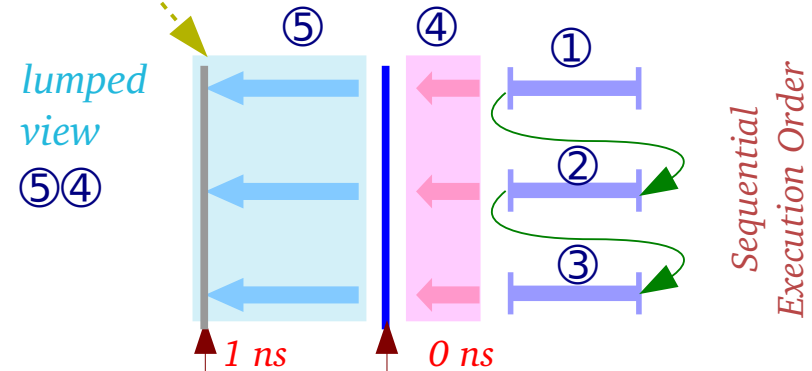
- * The order of statements is important
1. scheduled after some delta time
 2. non-blocking assignment

Updated values are observable after 1 ns

Concurrent assignments



Sequential assignments



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

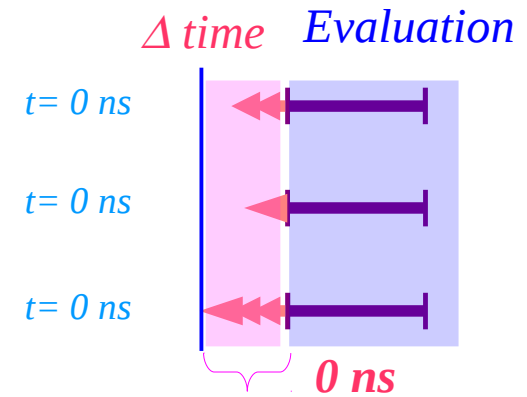
X2 | <= | A or B ;

Y2 | <= | C or D ;

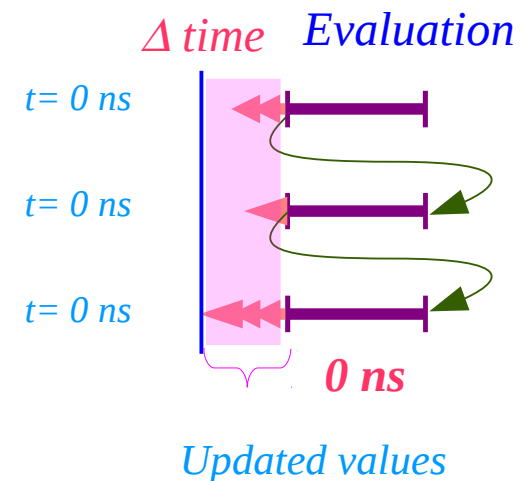
Z2 | <= | E or F ;

end process;

end



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



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;
    
```

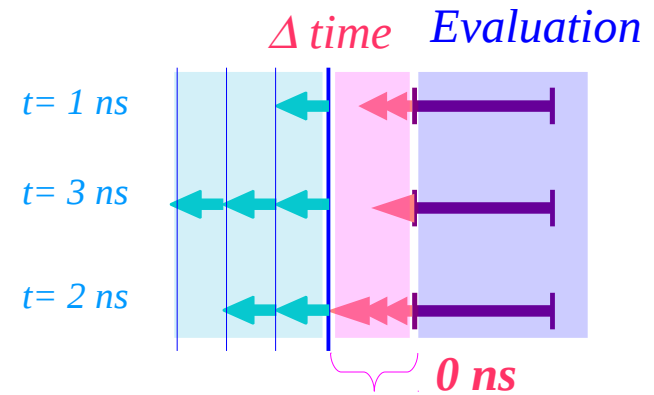
process (A, B, C, D, E, F)

begin

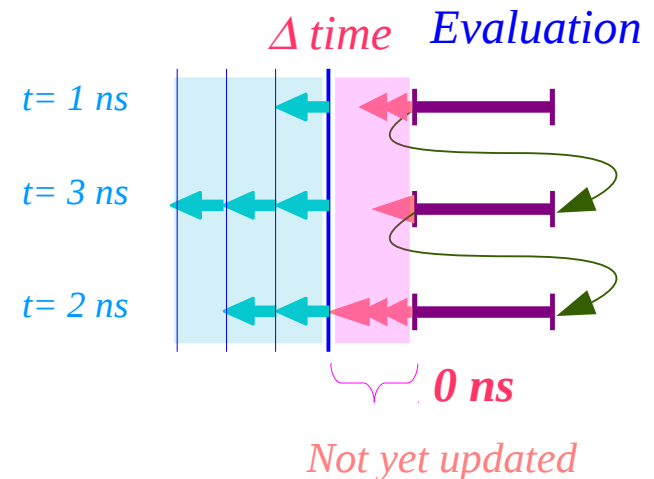
```

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

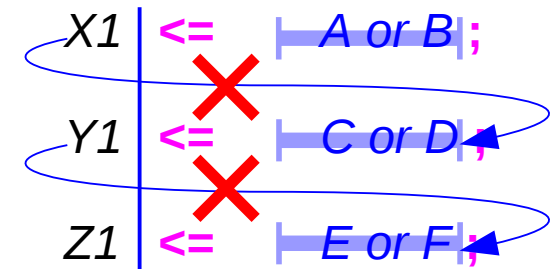
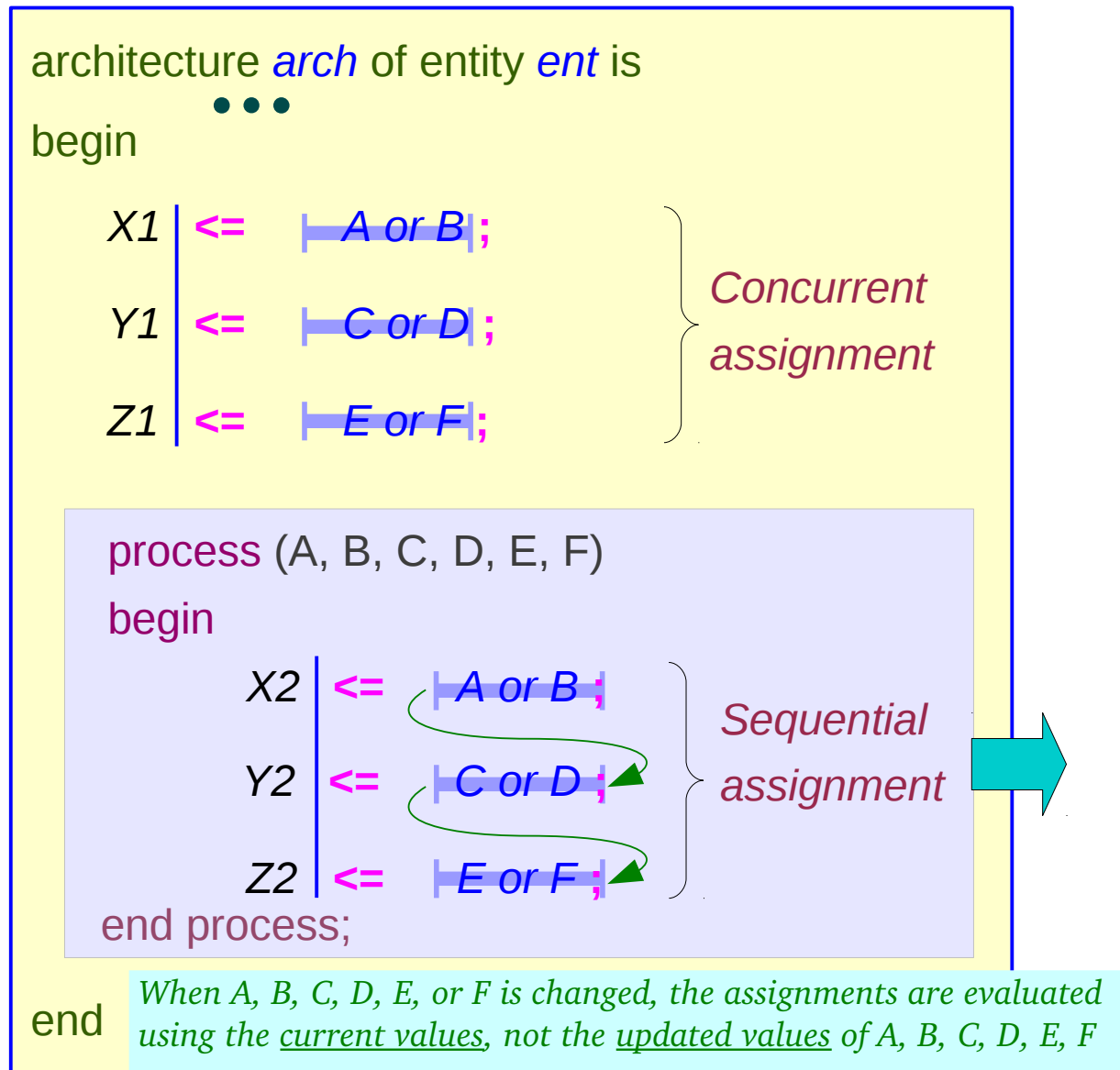
end



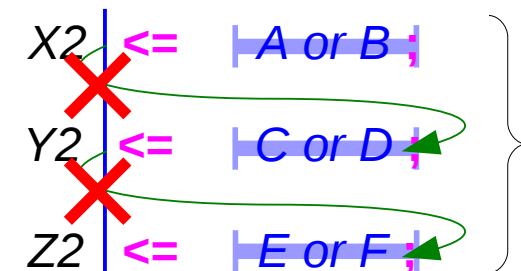
The exact no of delta is determined by the simulator and the context



Non-blocking Assignment (1)



non-blocking assignment

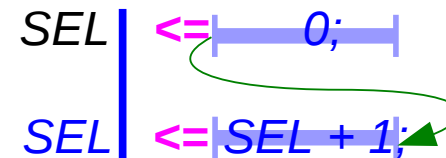


Non-blocking Assignment (2)

```
process (A, I0, I1)
begin
  SEL <= 0;
  if (A='1') then SEL <= SEL + 1; end if;
  case SEL is
    when 0
      Q <= I0;
    when 1
      Q <= I1;
  end case;
end process;
```

Scheduled on the next delta time

➡ *SEL value will not be updated until the next delta time*



Non-blocking Assignment

Without waiting the next delta time, it can continue to process the next sequential statement

(processed with the wrong value of SEL)

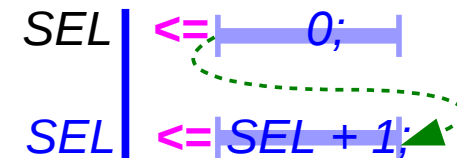
Non-blocking Assignment (3)

```
process
begin
  SEL    <= A or B;
  wait for 0 ns;
  if (A='1') then SEL <= SEL + 1; end if;
  wait for 0 ns;
  case SEL is
    when 0
      Q <= I0;
    when 1
      Q <= I1;
  end case;
  wait on A, I0, I1;
end process;
```

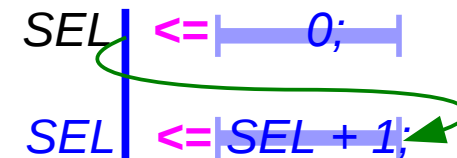
Wait for one delta time

Non-blocking

: next statement **before** update



wait for 0 ns;



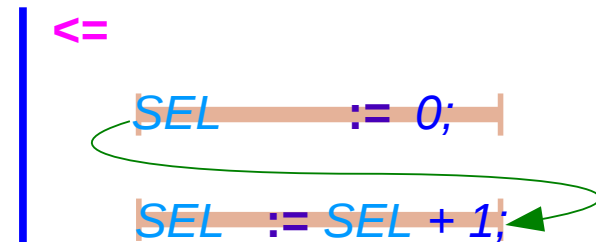
Blocking

: next statement **after** update

Non-blocking Assignment (4)

```
process (A, I0, I1)
  variable SEL : integer range 0 to 1;
begin
  SEL := A or B;
  if (A='1') then SEL := SEL + 1; end if;
  case SEL is
    when 0
      Q <= I0;
    when 1
      Q <= I1;
  end case;
end process;
```

Variable *SEL* changes its value *immediately*.



General MUX model

```
process (A, I0, I1)
begin
  case A is
    when '0'
      Q <= I0;
    when '1'
      Q <= I1;
  end case;
end process;
```

Variable & Signal Assignments

When *A, B, C, D, E, or F* is changed, the assignments are evaluated using the current values, not the updated values of *A, B, C, D, E, F*

Variable assignments

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

```
  variable X2, Y2, Z2 : bit;
```

```
begin
```

```
  X2 := A or B ;
```

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

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

```
end process;
```

Updated values of *X2, Y2, Z2* are observable **immediately**

Signal assignments

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

```
begin
```

```
  X2 <= A or B ;
```

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

```
  Z2 <= E or F ;
```

```
end process;
```

Updated values *X2, Y2, Z2* are observable after at least on delta time.

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 **cannot** have an **after** clause

Variable Assignment (2)

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

```
  variable Z2 : bit;
```

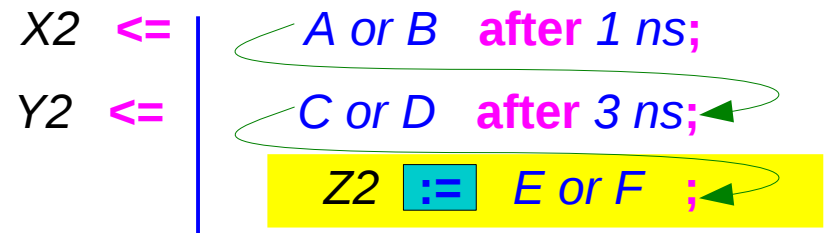
```
begin
```

```
  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;
```

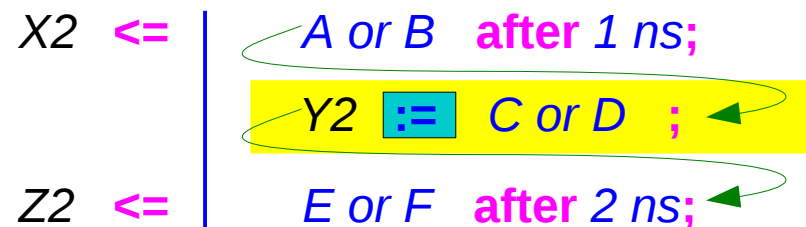
```
begin
```

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

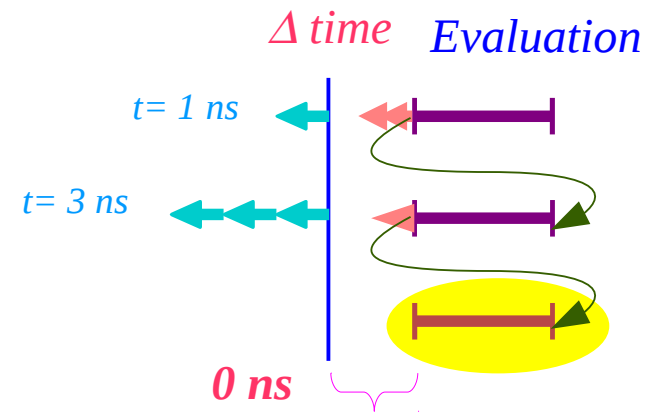
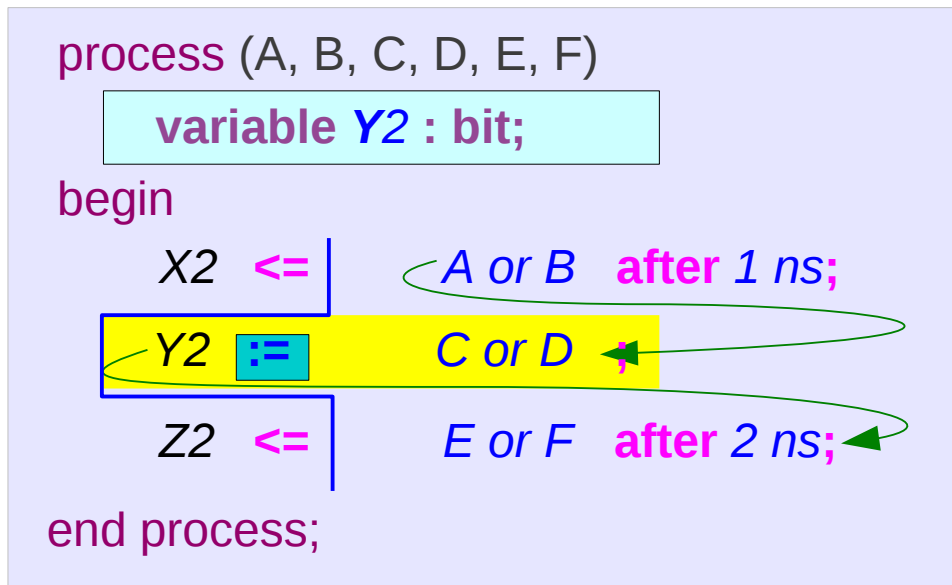
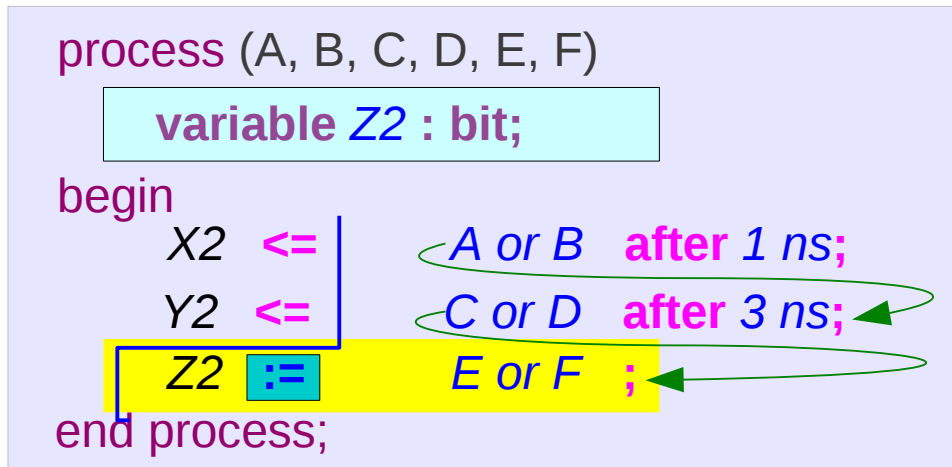
```
  Y2 := C or D ;
```

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

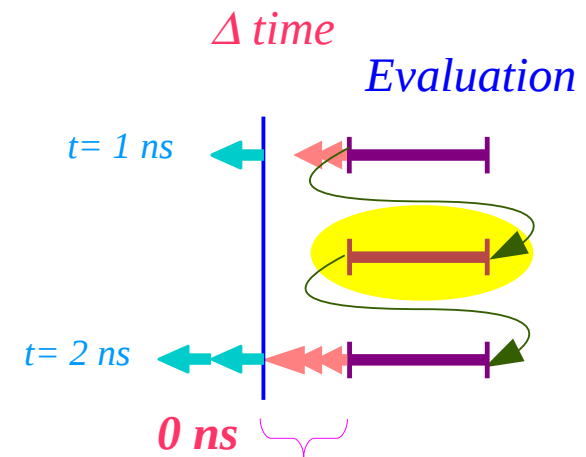
```
end process;
```



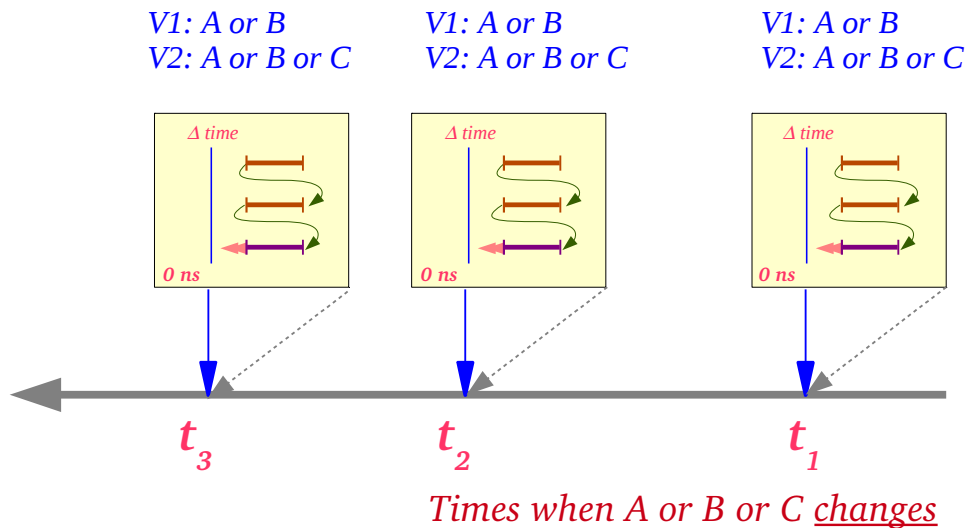
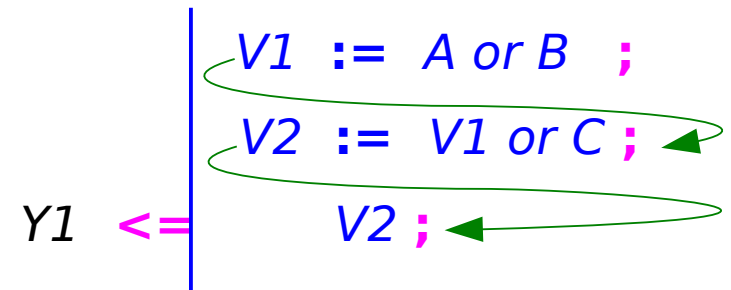
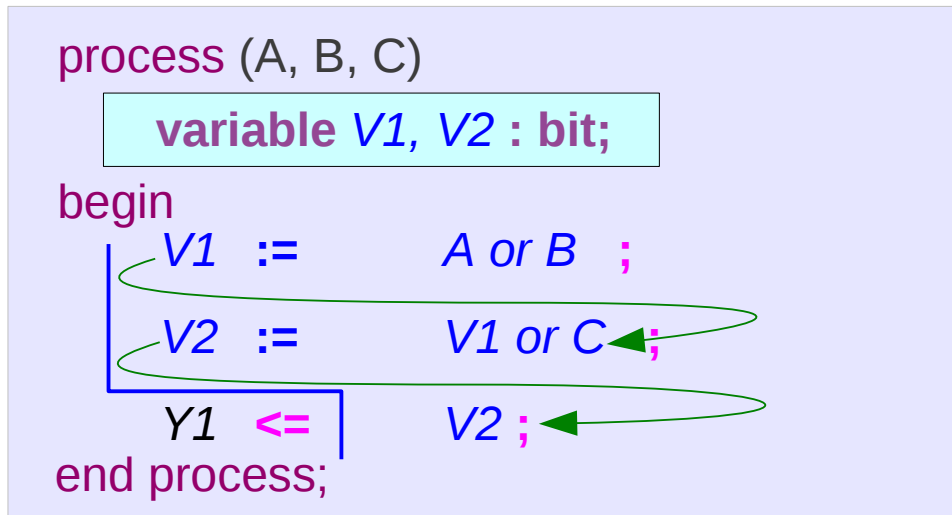
Variable Assignment (3)



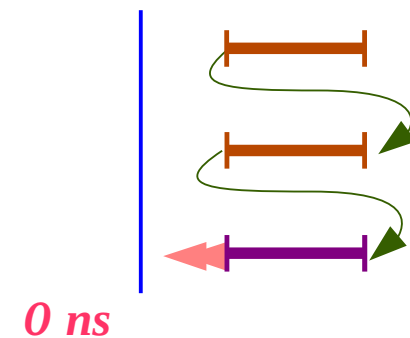
The variable assignment has nothing to do with time. It executes immediately.



Mixed Assignments Example (1)

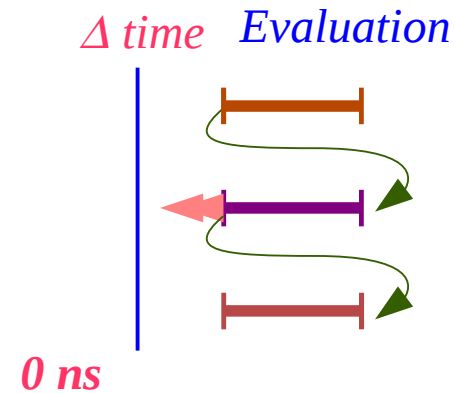
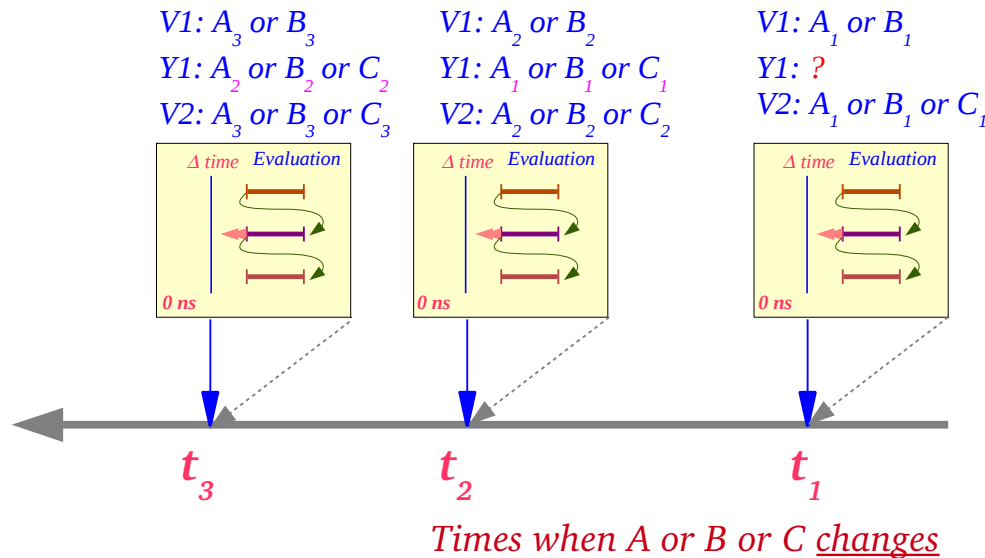
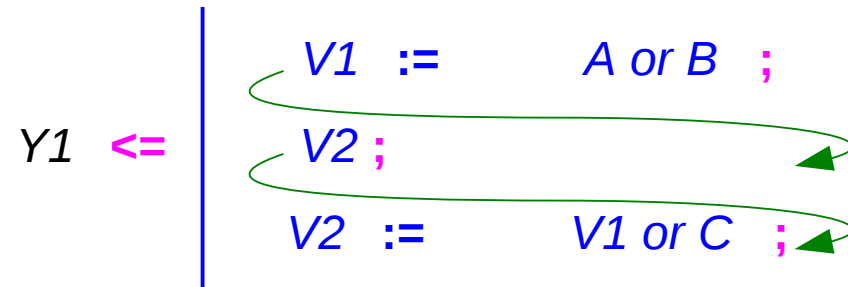
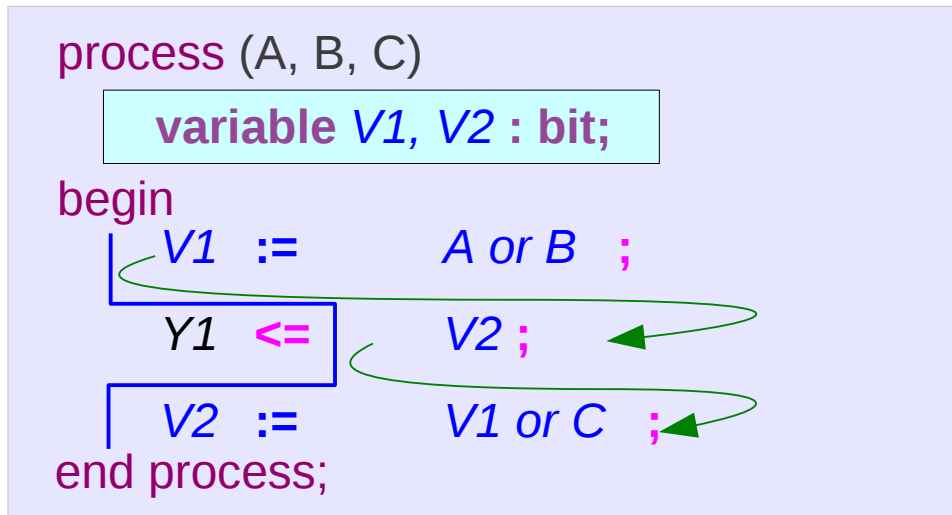


Δ time Evaluation



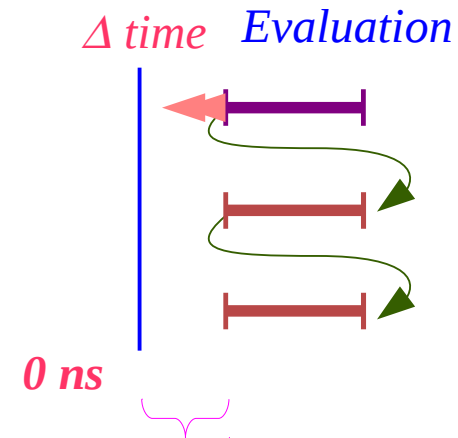
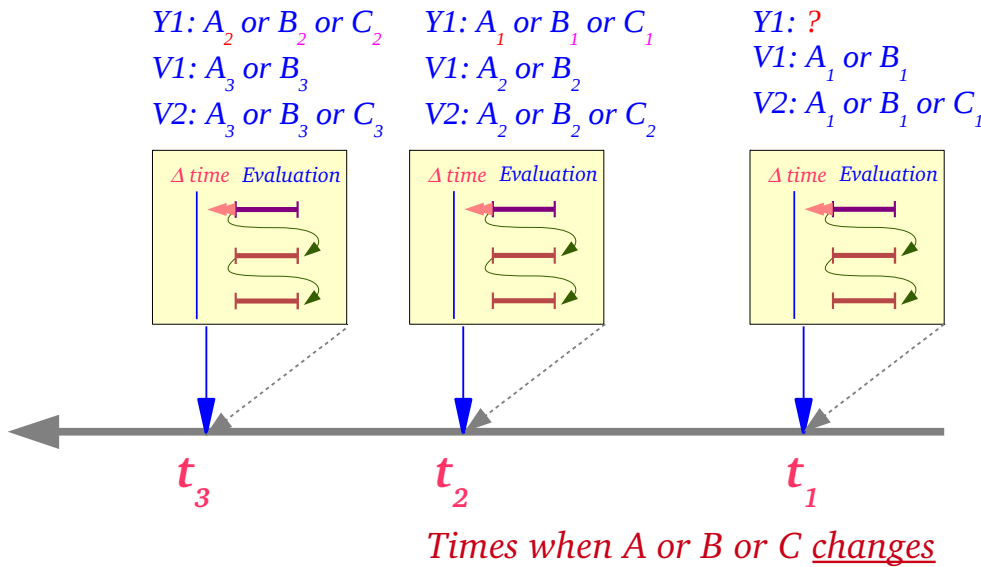
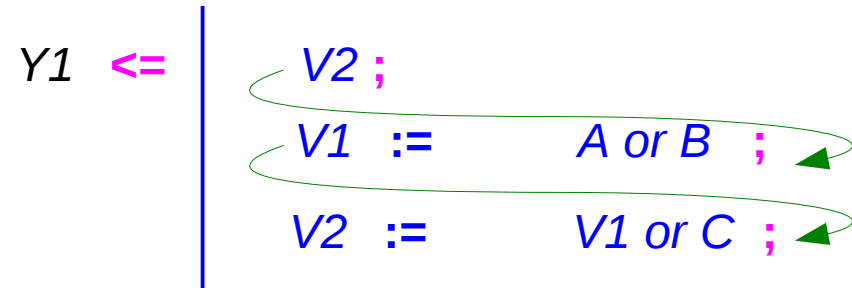
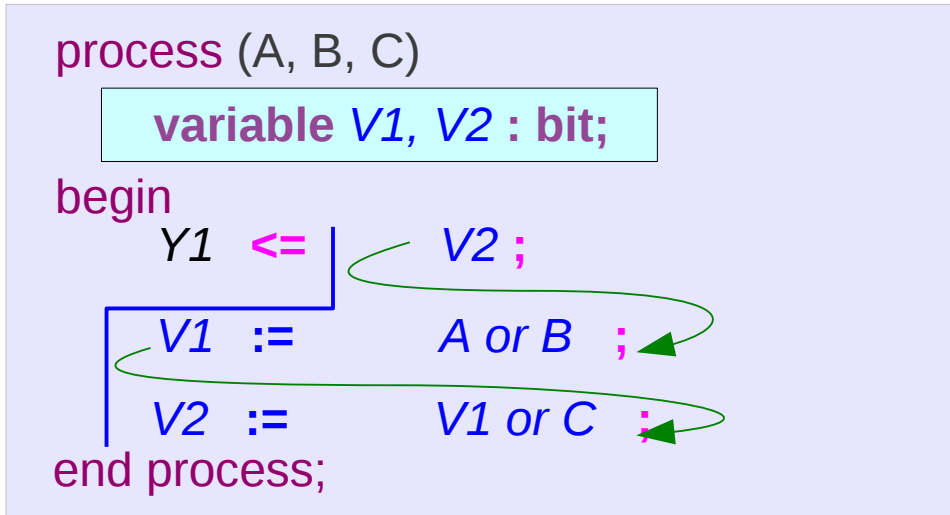
Y1 <= A or B or C ;

Mixed Assignments Example (2)



Y1 <= A or B or C ;

Mixed Assignments Example (3)



$Y1 <= A \text{ or } B \text{ or } C ;$

Mixed Assignments Example (4)

```
process (A, B, C)
```

```
  variable V1, V2 : bit;
```

```
begin
```

```
  V1 := A or B ;
```

```
  V2 := V1 or C ;
```

```
  Y1 <= V2 ;
```

```
end process;
```

```
process (A, B, C)
```

```
  variable V1, V2 : bit;
```

```
begin
```

```
  Y1 <= V2 ;
```

```
  V1 := A or B ;
```

```
  V2 := V1 or C ;
```

```
end process;
```

```
process (A, B, C)
```

```
  variable V1, V2 : bit;
```

```
begin
```

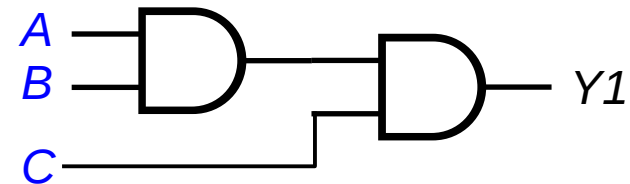
```
  V1 := A or B ;
```

```
  Y1 <= V2 ;
```

```
  V2 := V1 or C ;
```

```
end process;
```

Same Synthesis Result



References

- [1] <http://en.wikipedia.org/>
- [2] J. V. Spiegel, VHDL Tutorial,
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/