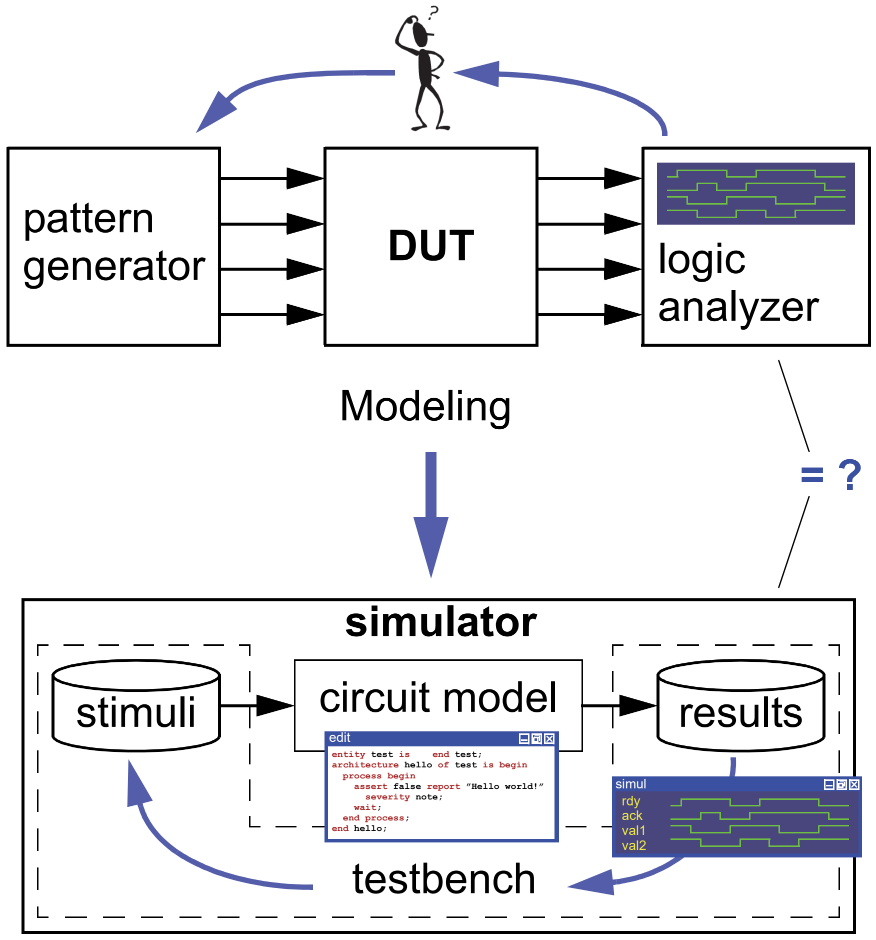
Simulation environment

- **Simulation = modeling + analysis**
- **Environment**
  - **design under test (DUT)**
    - **different abstraction levels**
  - **stimuli generator**
    - **different input-data sequences**
  - **results analyzer**
    - **is DUT responding correctly?**
- **Different combinations exist...**



```
entity test is    end test;
architecture hello of test is begin
  process begin
    assert false report "Hello world!"
      severity note;
    wait;
  end process;
end hello;
```

Use of HDL –> Simulation

- Simulation = modeling + analysis
  - Logic / register-transfer / functional (behavioral) / system level simulation

concurrent / parallel modules

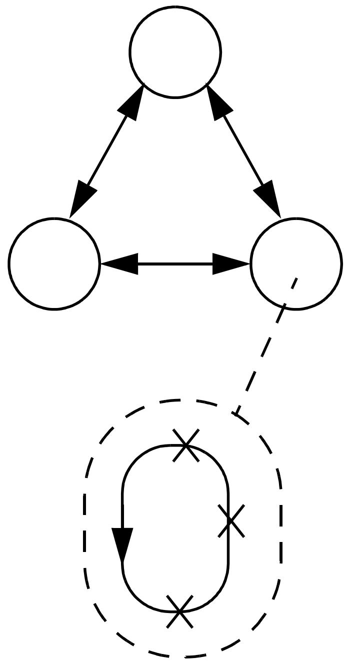
connected via signal / channels

sequential vs. concurrent execution?

execution order?!

current / new values to avoid non-determinism

event queue history+future



module / unit / process

continuous execution is slow

only when needed?

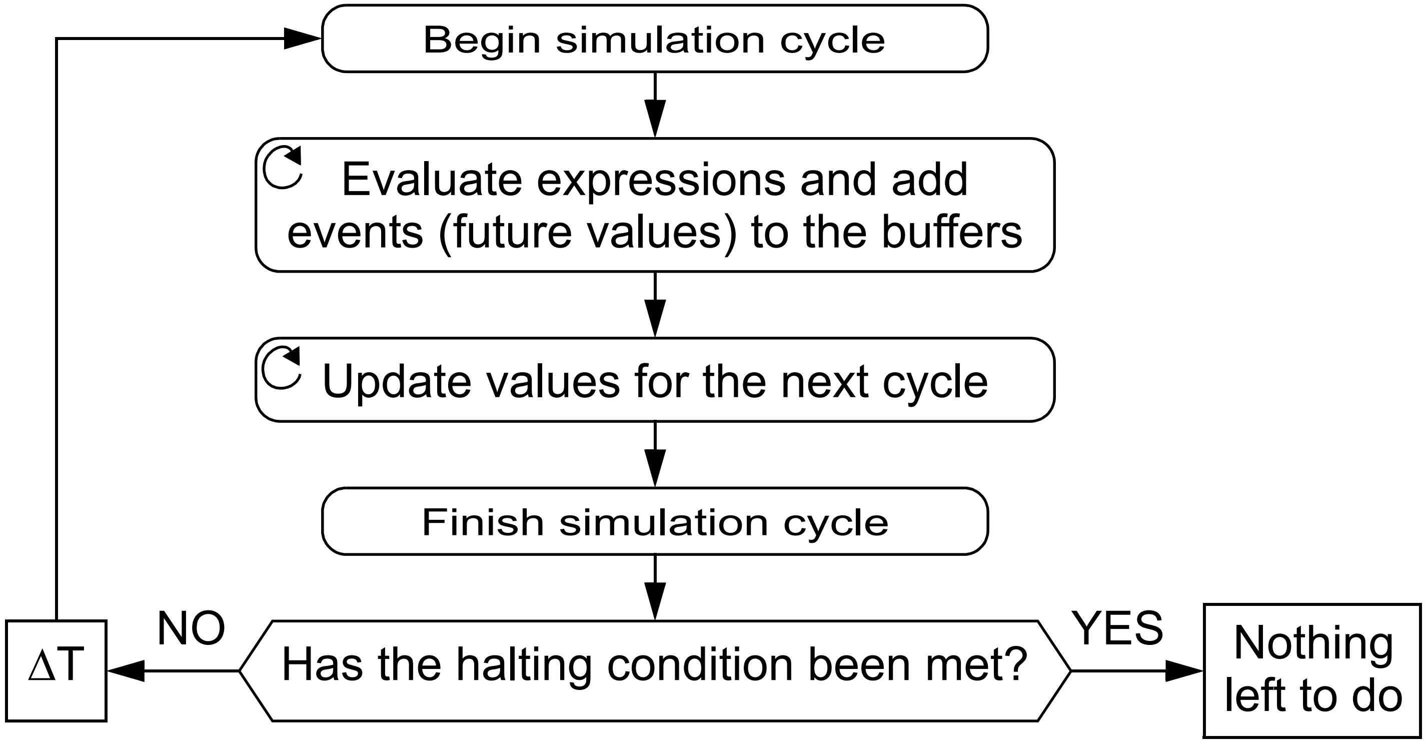
time / event triggered

different simulation engines

Simulators & timing/delay models

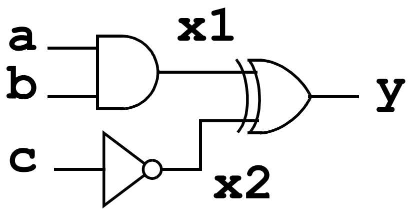
- Time & events
  - *Time-driven*:
    all components of the digital logic system are evaluated at every time step
  - *Event-driven*:
    system input events are kept in an time-ordered event queue
- Delay models
  - unit-delay (RTL simulator)
  - zero-delay (Verilog)
  - *delta*-delay (VHDL) – δ-delay, Δ-delay
- Simulation engines
  - all make use of the three following steps but details differ...
    - (1) calculate (and remember) new values for signals
    - (2) update signal values
    - (3) update time

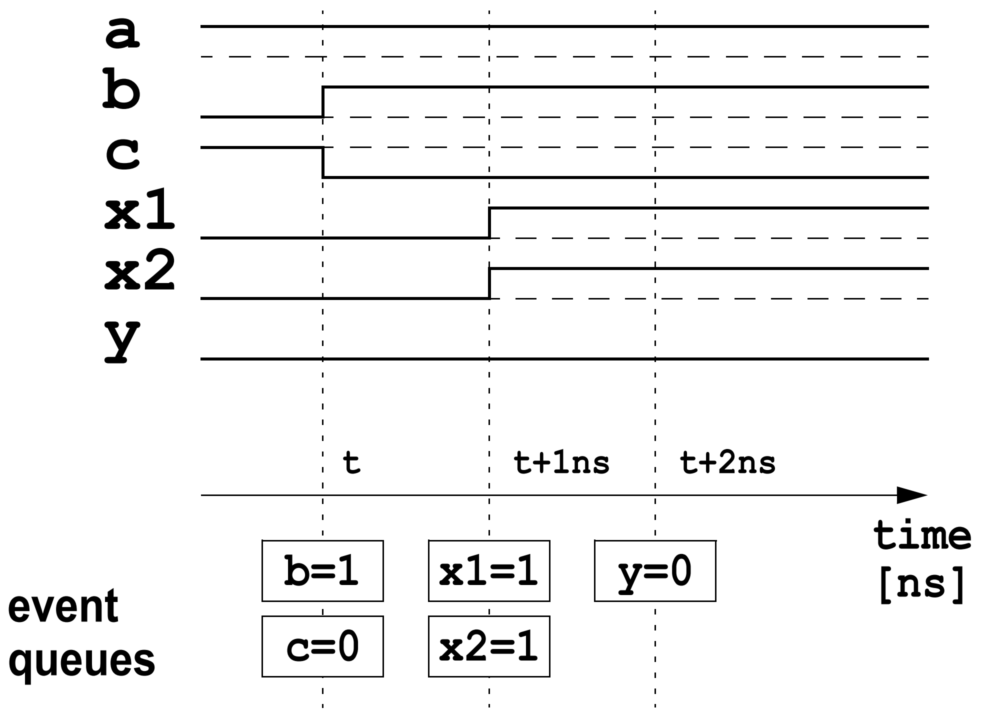
Unit-delay simulation model


Begin simulation cycle
Evaluate expressions and add events (future values) to the buffers
Update values for the next cycle
Finish simulation cycle
Has the halting condition been met?
NO
ΔT
YES
Nothing left to do

Unit-delay simulation model (example)

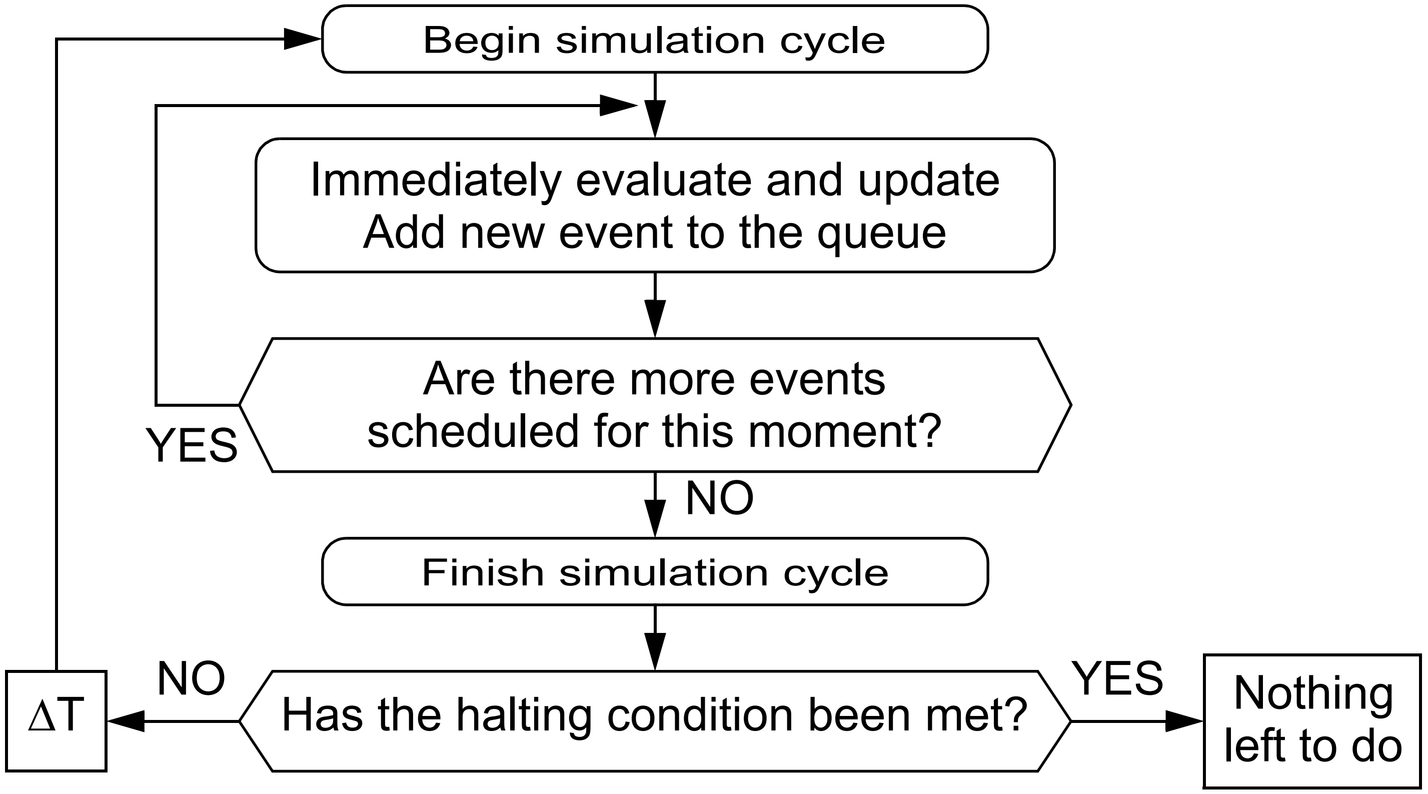
```
x1 <= a and b;
x2 <= not c;
y <= x1 xor x2;
```





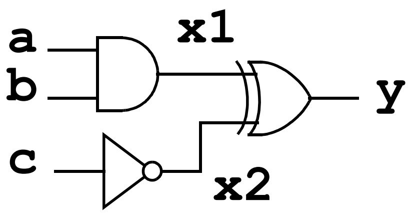
- Very fast but does not allow immediate signal changes

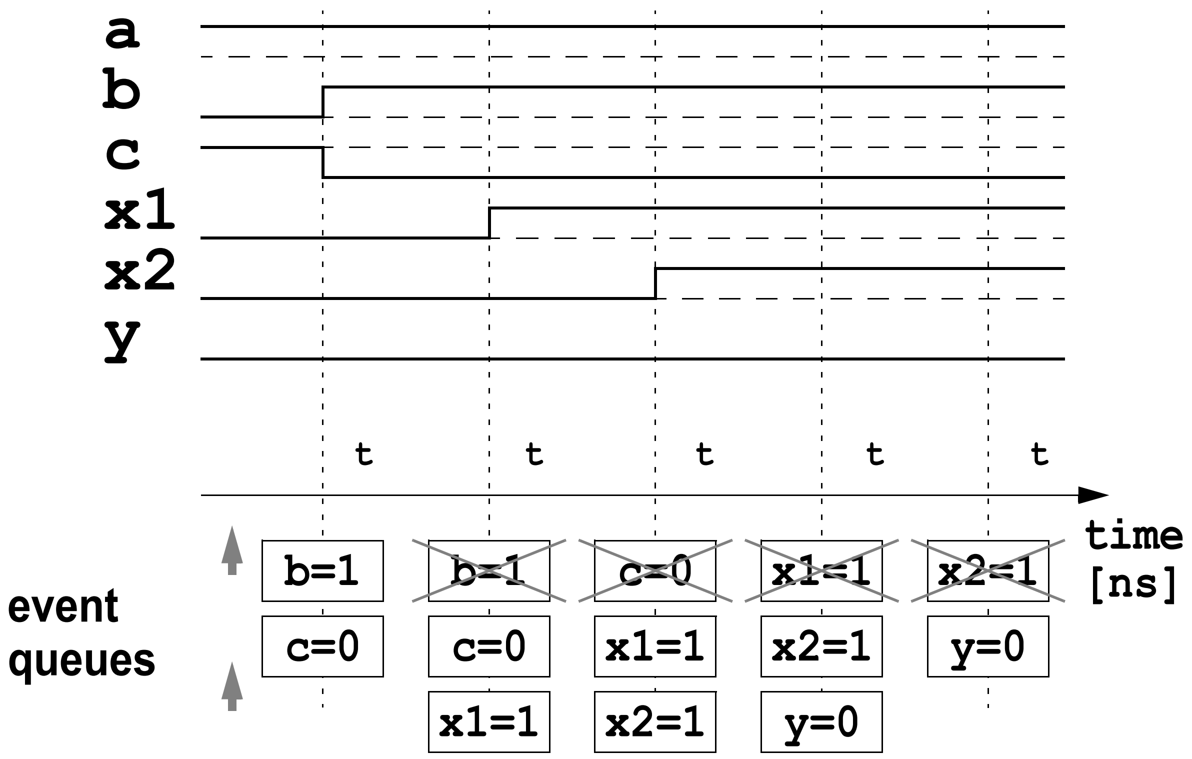
Zero-delay simulation model (Verilog)


Begin simulation cycle
Immediately evaluate and update
Add new event to the queue
Are there more events scheduled for this moment?
YES
NO
Finish simulation cycle
Has the halting condition been met?
NO
YES
ΔT
Nothing left to do

Zero-delay simulation model (example #1)

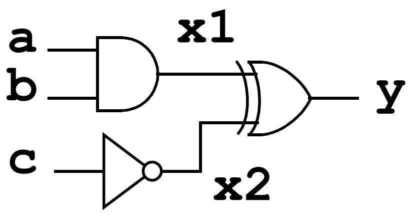
```
x1 <= a and b;
x2 <= not c;
y <= x1 xor x2;
```

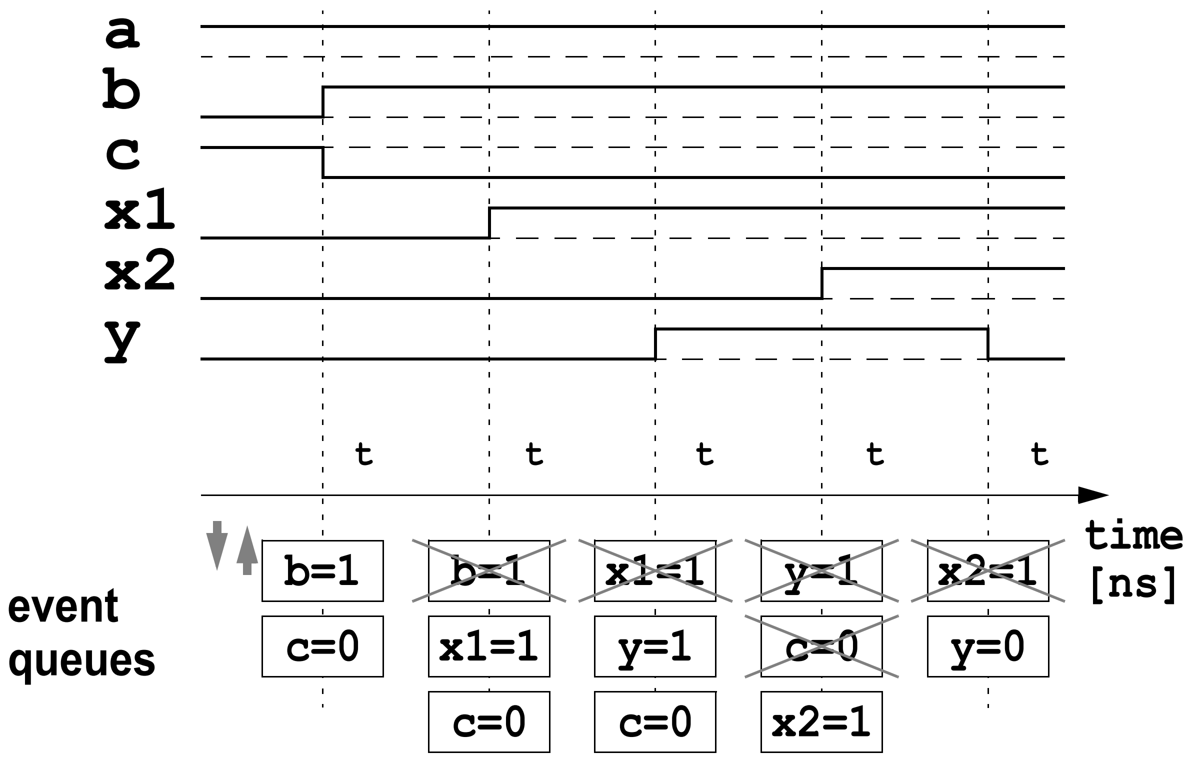




Zero-delay simulation model (example #2)

```
x1 <= a and b;
x2 <= not c;
y <= x1 xor x2;
```



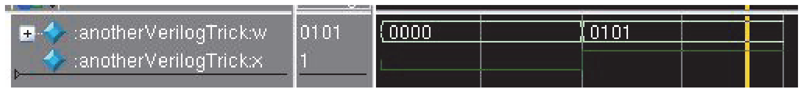


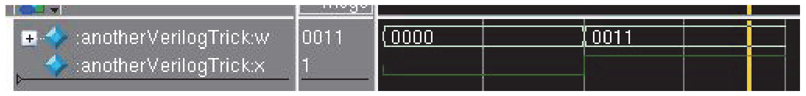
- **Fast and allows immediate signal changes, may suffer from non-determinism**

Non-deterministic behavior

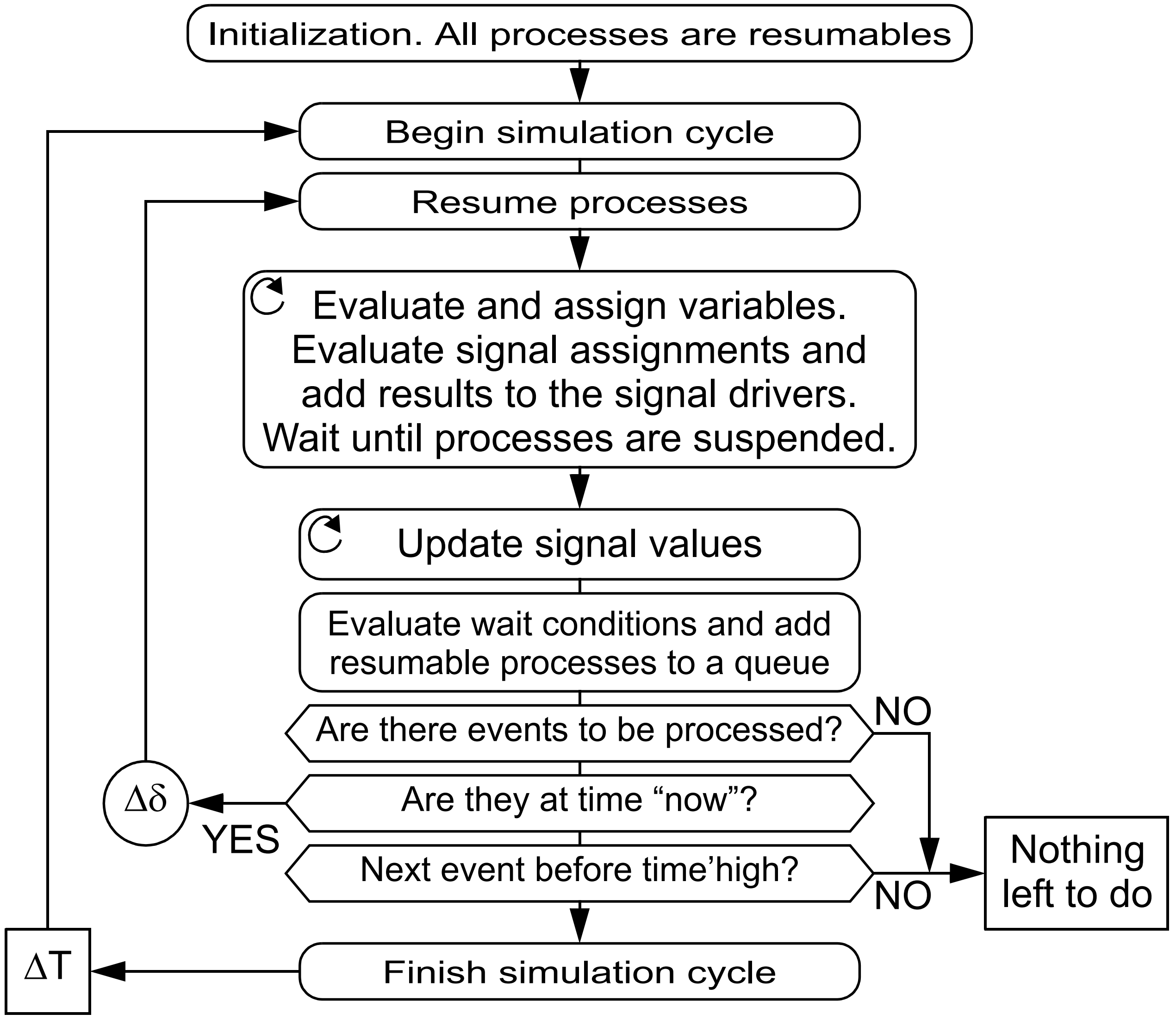
```
module anotherVerilogTrick;
  reg [3:0] w;
  reg x;
  initial begin
    x = 0;  w = 0;
    #100 x = 1;  #100;
  end
  always @(posedge x) w = 3;
  always @(posedge x) w = 5;
endmodule
```

```
module anotherVerilogTrick;
  reg [3:0] w;
  reg x;
  initial begin
    x = 0;  w = 0;
    #100 x = 1;  #100;
  end
  always @(posedge x) w = 5;
  always @(posedge x) w = 3;
endmodule
```


```
:anotherVerilogTrick:w  0101  0000  0101
:anotherVerilogTrick:x  1
```

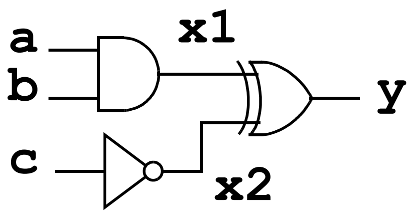

```
:anotherVerilogTrick:w  0011  0000  0011
:anotherVerilogTrick:x  1
```

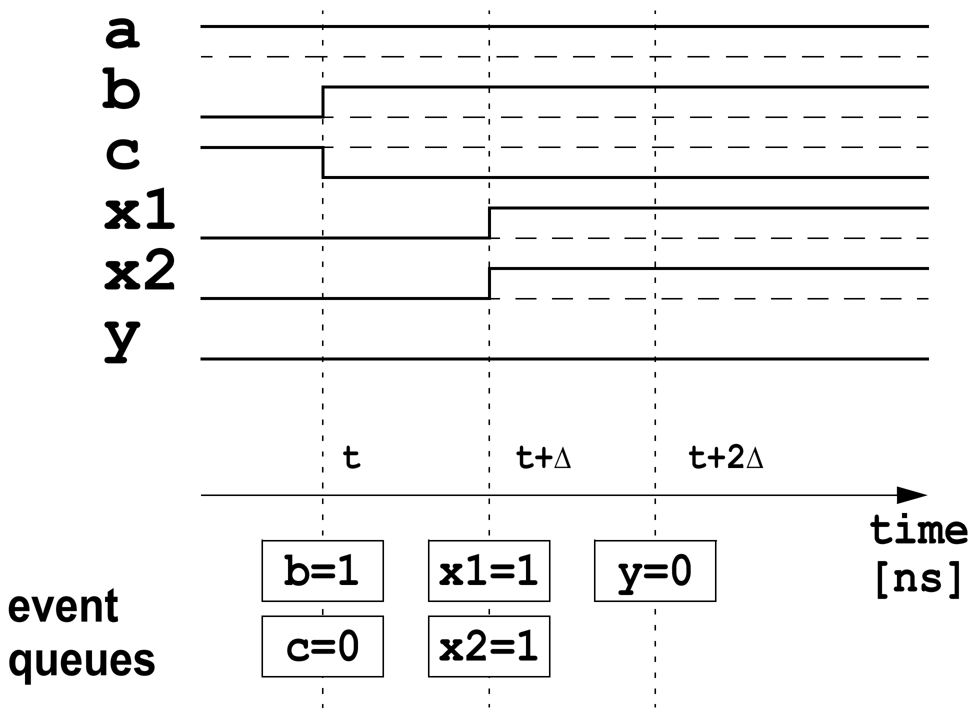
Delta-delay simulation model (VHDL)


Initialization. All processes are resumables
Begin simulation cycle
Resume processes
Evaluate and assign variables. Evaluate signal assignments and add results to the signal drivers. Wait until processes are suspended.
Update signal values
Evaluate wait conditions and add resumable processes to a queue
Are there events to be processed?
NO
Are they at time “now”?
Δδ
YES
Next event before time’high?
NO
Nothing left to do
Finish simulation cycle
ΔT

Delta-delay (VHDL) simulation model (example)

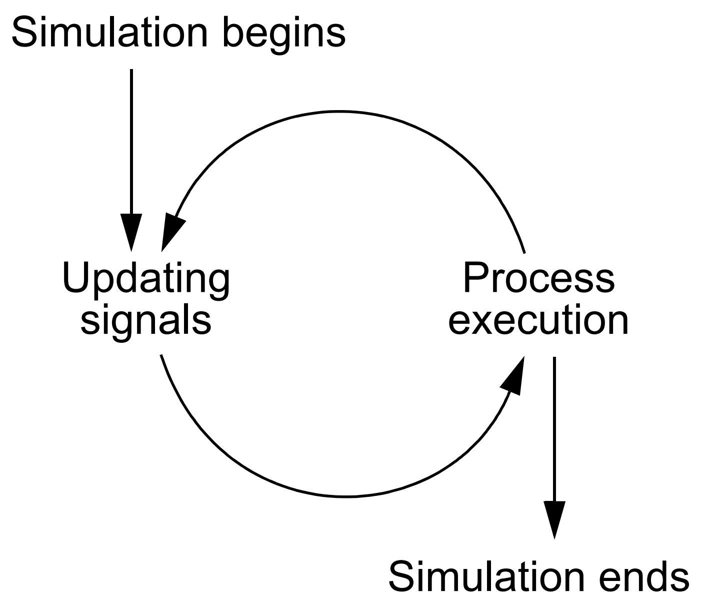
```
x1 <= a and b;
x2 <= not c;
y <= x1 xor x2;
```

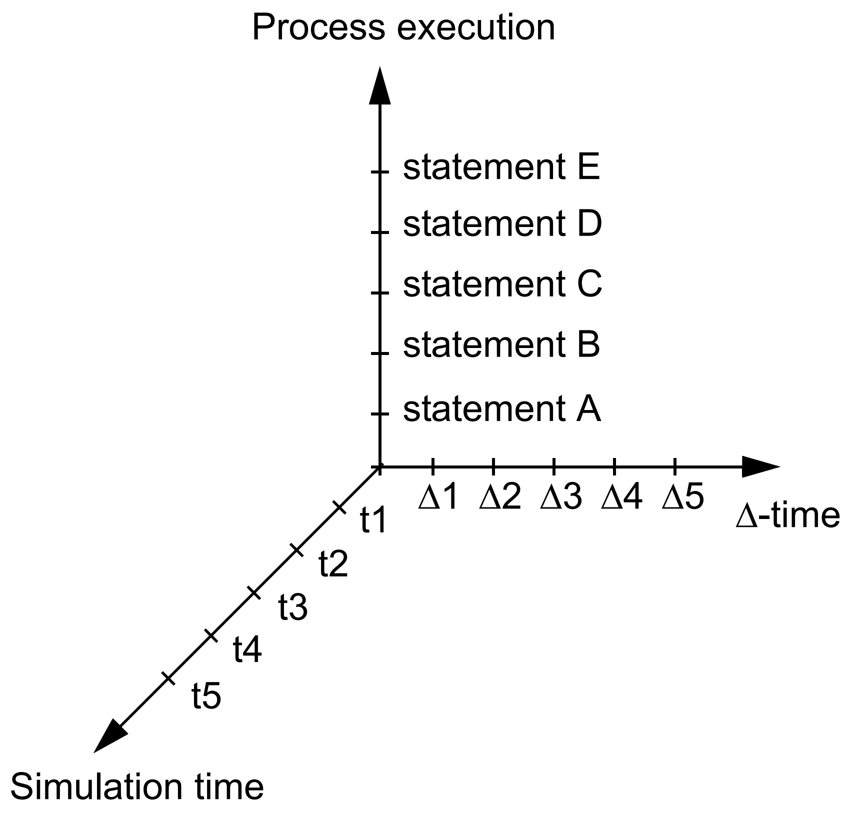




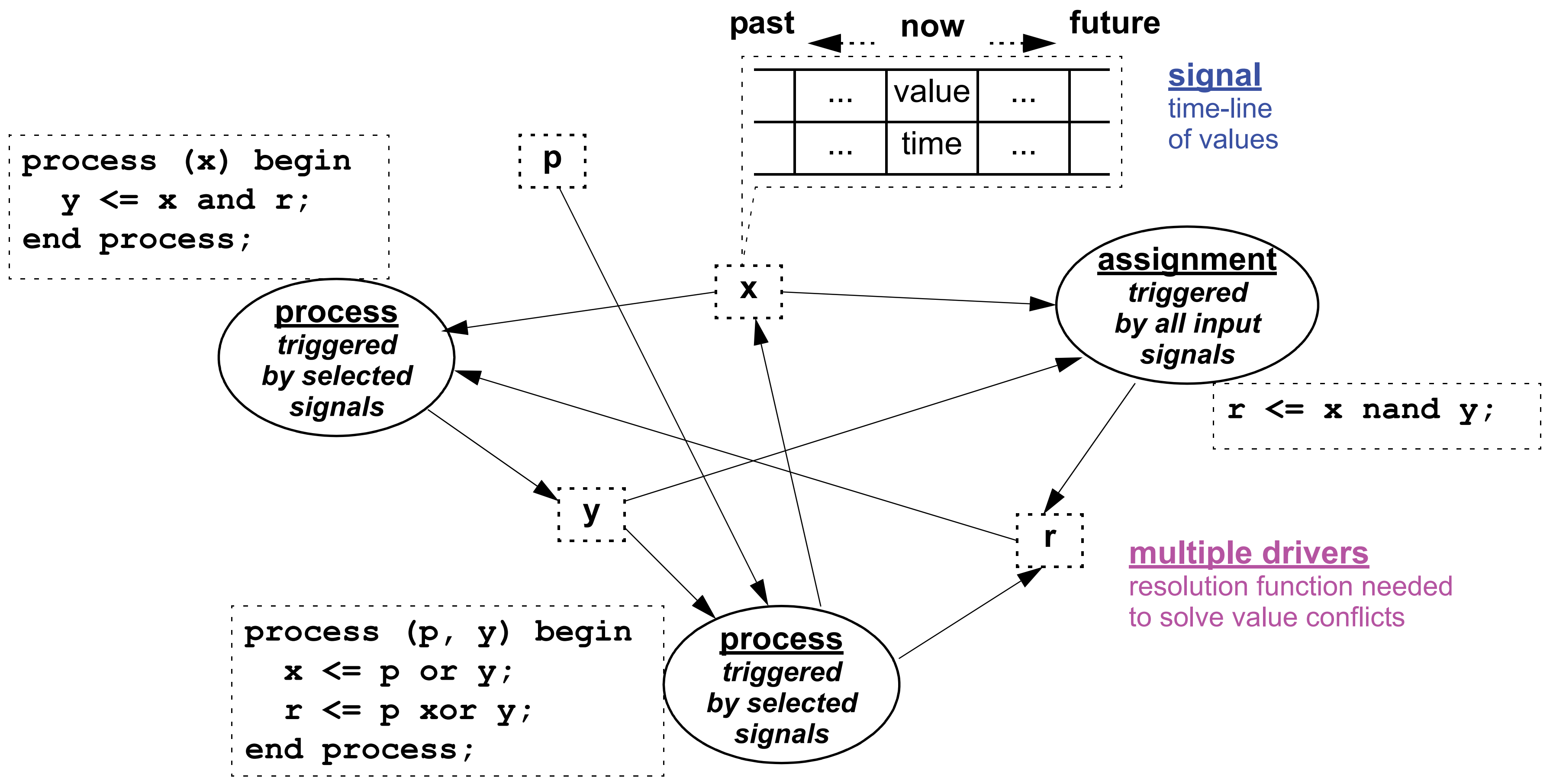
- **Deterministic but slower and may suffer from delta-cycle oscillation**

VHDL simulation scheme





VHDL simulation scheme



```
process (x) begin
  y <= x and r;
end process;
```

```
r <= x nand y;
```

```
process (p, y) begin
  x <= p or y;
  r <= p xor y;
end process;
```

Delta delay example #1

```
-- SR flip-flop
x <= not (y and lset);   -- (1)
y <= not (x and reset);  -- (2)
```

- **Note the different initial values!**

time	lset	x	y	reset	stm.
20 ns	_	0	1	1	(1)
20 ns + 1Δ	0	_/	1	1	(2)
20 ns + 2Δ	0	1	_	1	(1)
20 ns + 3Δ	0	1	0	1	-
30 ns	_/	1	0	1	(1)
30 ns + 1Δ	1	1	0	1	-
40 ns	1	1	0	_	(2)
40 ns + 1Δ	1	1	_/	0	(1)
40 ns + 2Δ	1	_	1	0	(2)
40 ns + 3Δ	1	0	1	0	-

Delta delay example #2

```
-- SR flip-flop + delays
x <= not (y and lset)
      after 2 ns;          -- (1)
y <= not (x and reset)
      after 2 ns;          -- (2)
```

- **Note the different initial values!**

time	lset	x	y	reset	stm.
20 ns	_	0	1	1	(1)
22 ns	0	_/	1	1	(2)
24 ns	0	1	_	1	(1)
24 ns + 1Δ	0	1	0	1	-
30 ns	_/	1	0	1	(1)
30 ns + 1Δ	1	1	0	1	-
40 ns	1	1	0	_	(2)
42 ns	1	1	_/	0	(1)
44 ns	1	_	1	0	(2)
44 ns + 1Δ	1	0	1	0	-

Delta delay example #3

- **Dangers of default initialization**

```
-- SR flip-flop & oscillation
x <= not (y and lset);   -- (1)
y <= not (x and reset); -- (2)
```

- **Equal initial values will result in oscillation even with after clause!**

time	lset	x	y	reset	stm.
0 ns	_/	_	_	_/	(1),(2)
0 ns + 1Δ	1	_/	_/	1	(1),(2)
0 ns + 2Δ	1	_	_	1	(1),(2)
0 ns + 3Δ	1	_/	_/	1	(1),(2)
etc.	1	...	...	1	(1),(2)
0 ns	_/	_/	_/	_/	(1),(2)
0 ns + 1Δ	1	_	_	1	(1),(2)
0 ns + 2Δ	1	_/	_/	1	(1),(2)
0 ns + 3Δ	1	_	_	1	(1),(2)
etc.	1	...	...	1	(1),(2)