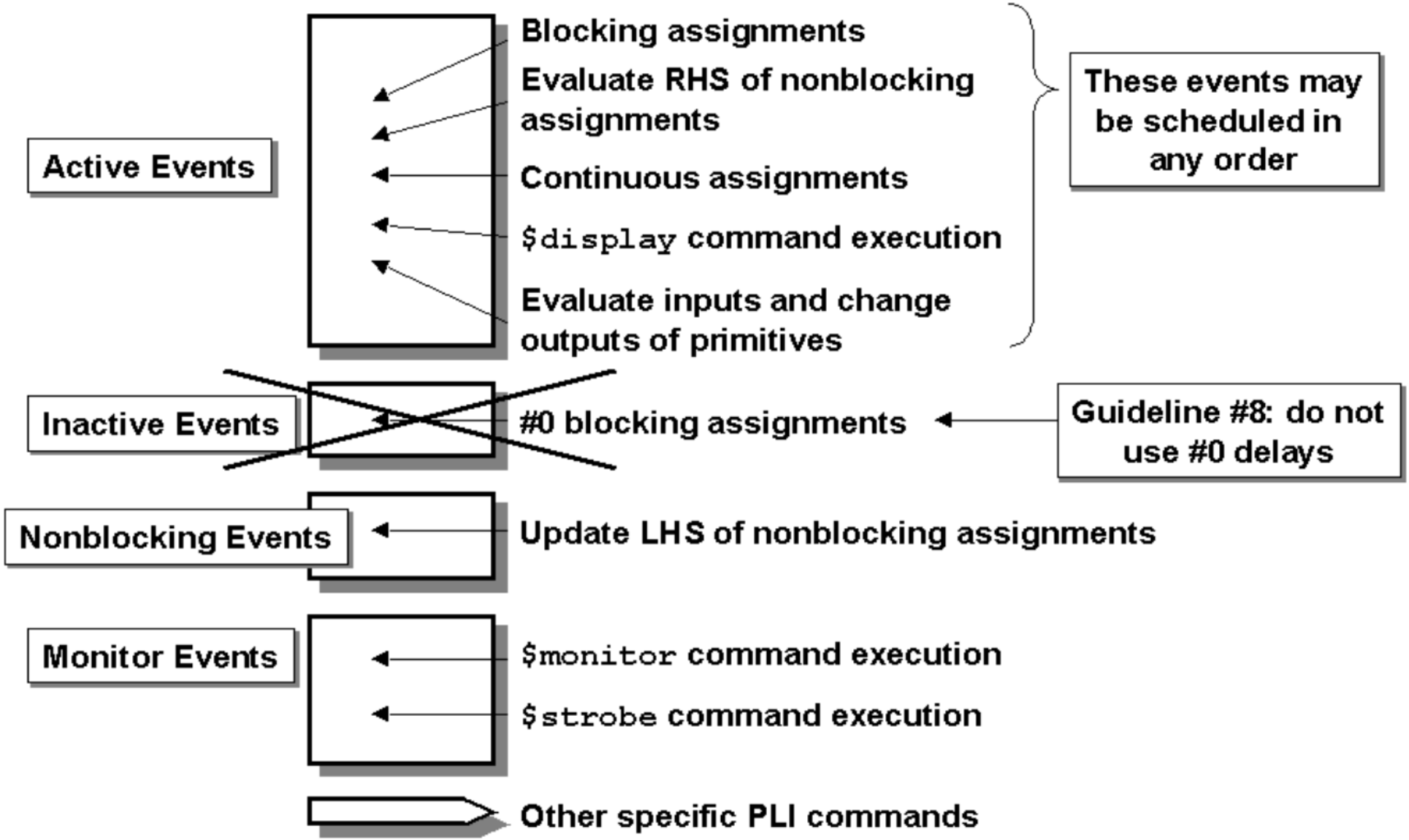
Verilog Simulation Model

Verilog has 5 queue "stratified event model"

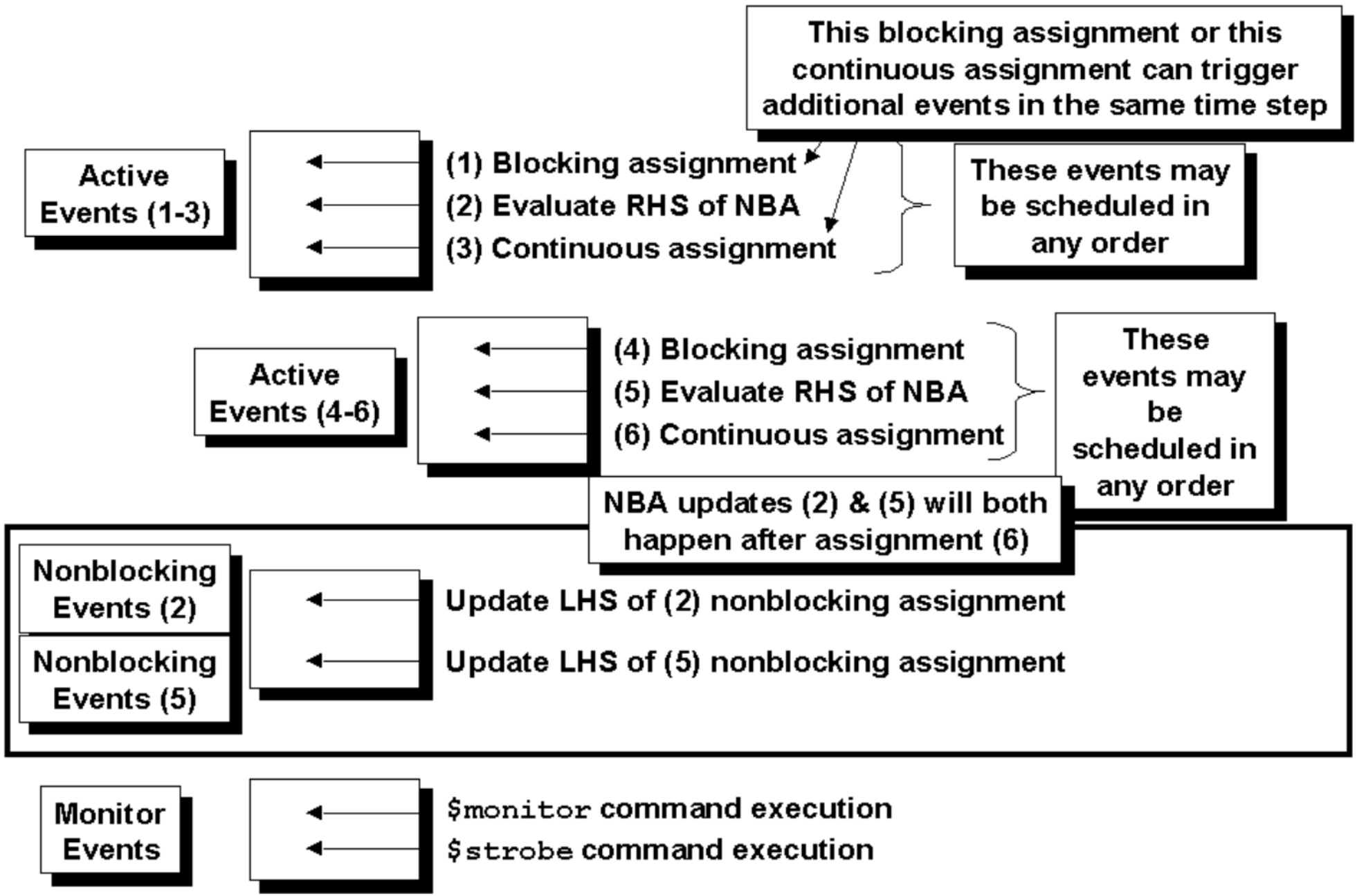


Verilog Simulation Algorithm

```
while (there are events) {
  if (there are active events) {
    E = any active event;
    if (E is an update event) {
      update the modified object;
      add evaluation events for sensitive processes to event queue;
    }
    else { // this is an evaluation event, so ...
      evaluate the process;
      add update events to the event queue;
    }
  }

  else if (there are nonblocking update events) {
    activate all nonblocking update events;
  }

  else {
    advance T to the next event time;
    activate all inactive events for time T;
  }
}
```

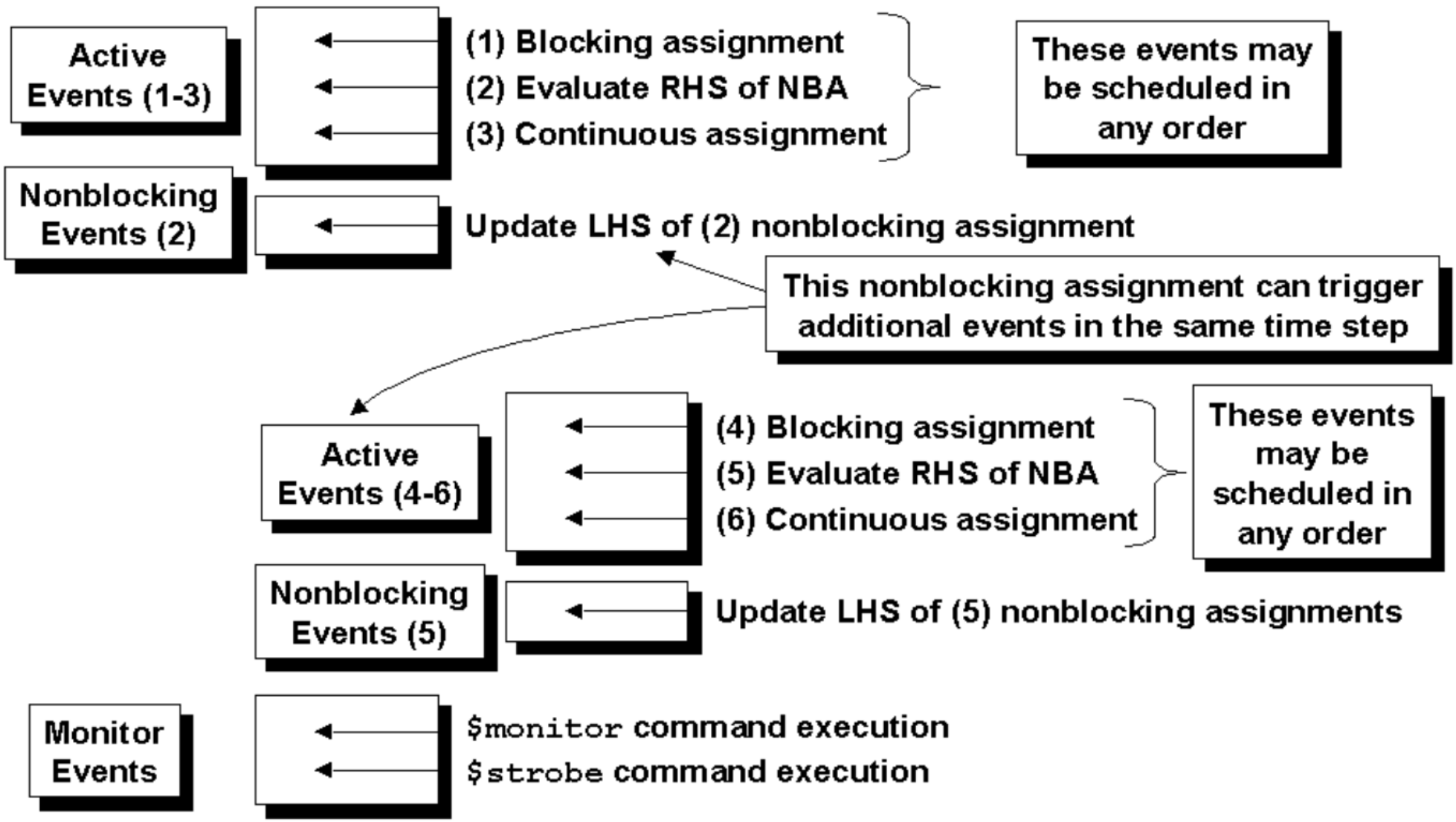
Triggering Events in Same Step 1

BA assignment can generate more events for same step



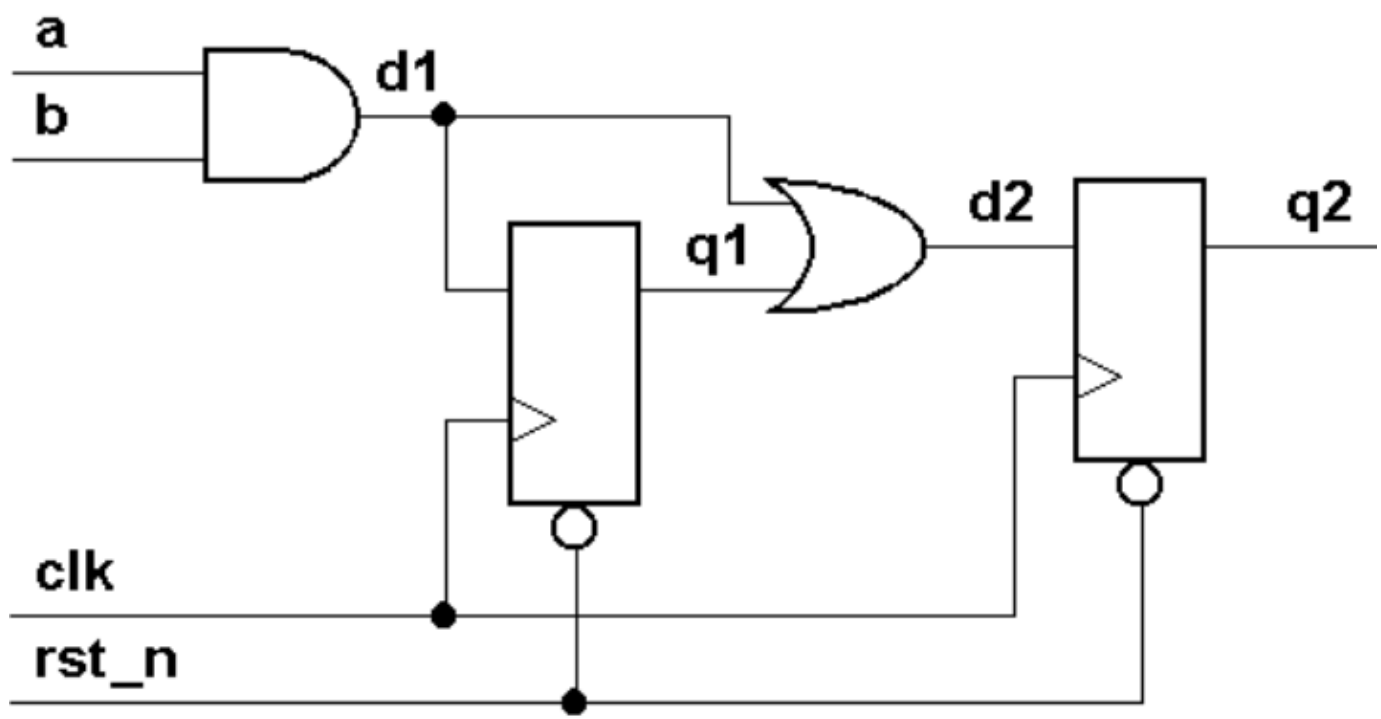
Triggering Events in Same Step 2

A NBA assignment can generate more events for same step



An example

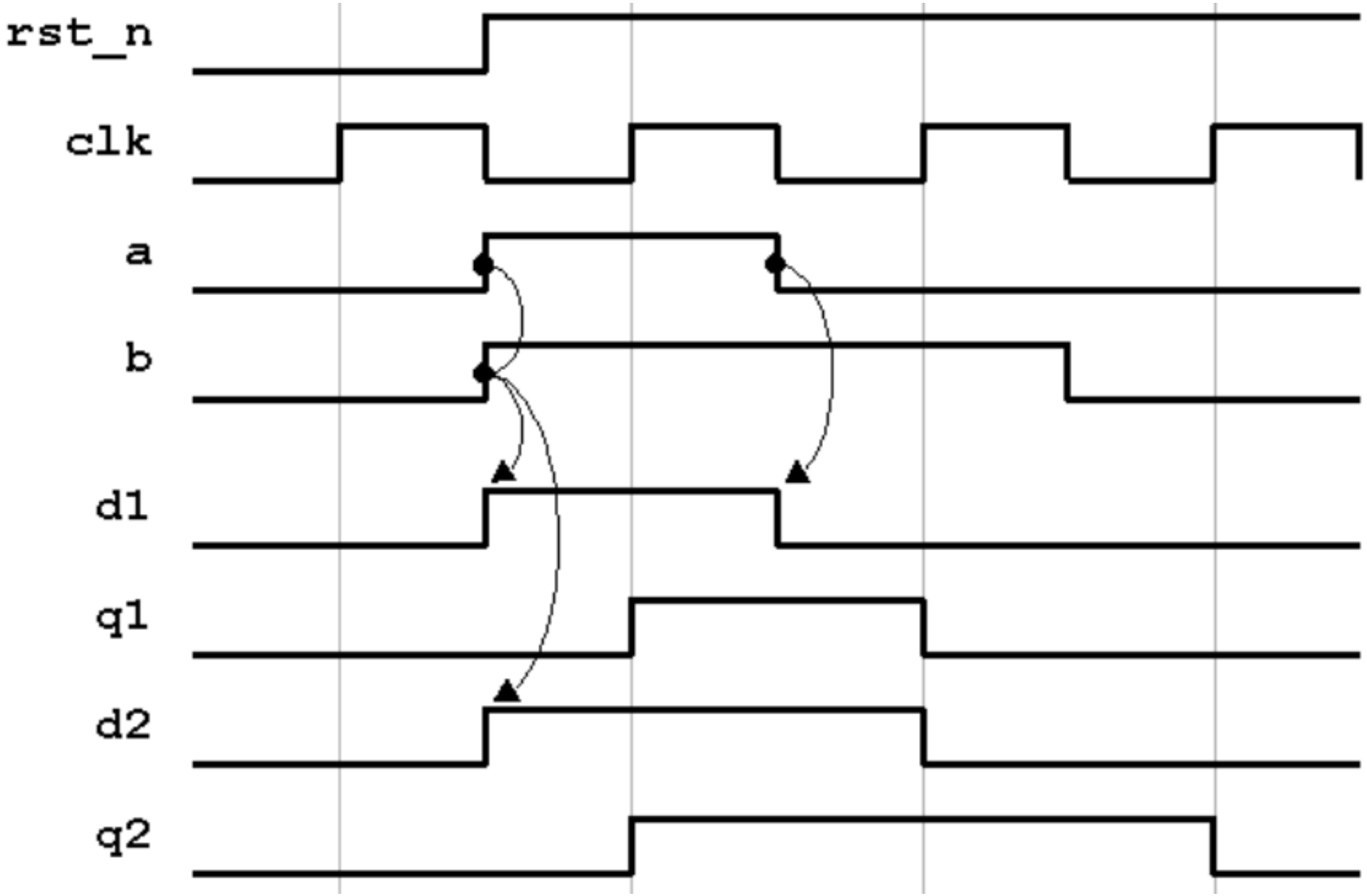
```
module sblk1 (
  output reg q2,
  input      a, b, clk, rst_n);
  reg        q1, d1, d2;

  always @(a or b or q1) begin
    d1 = a & b;
    d2 = d1 | q1;
  end

  always @(posedge clk or negedge rst_n)
    if (!rst_n) begin
      q2 <= 0;
      q1 <= 0;
    end
    else begin
      q2 <= d2;
      q1 <= d1;
    end
endmodule
```



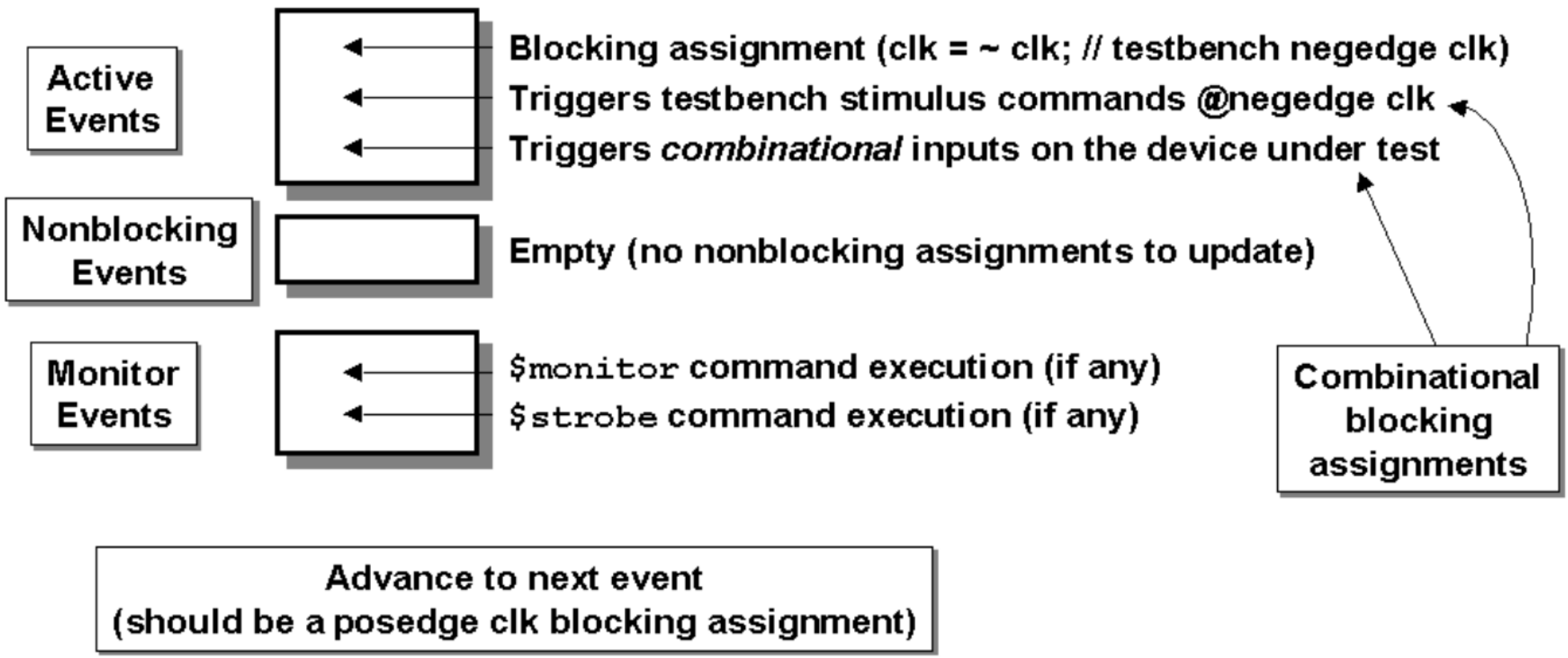
Ex1: testbench

```
module tb;
  reg a, b, clk, rst_n;

  initial begin // clock oscillator
    clk = 0;
    forever #10 clk = ~clk;
  end

  sblk1 u1 (.q2(q2), .a(a), .b(b), .clk(clk), .rst_n(rst_n));

  initial begin // stimulus
    a = 0; b = 0;
    rst_n <= 0;
    @(posedge clk);
    @(negedge clk) rst_n = 1;
    a = 1; b = 1;
    @(negedge clk) a = 0;
    @(negedge clk) b = 0;
    @(negedge clk) $finish;
  end
endmodule
```

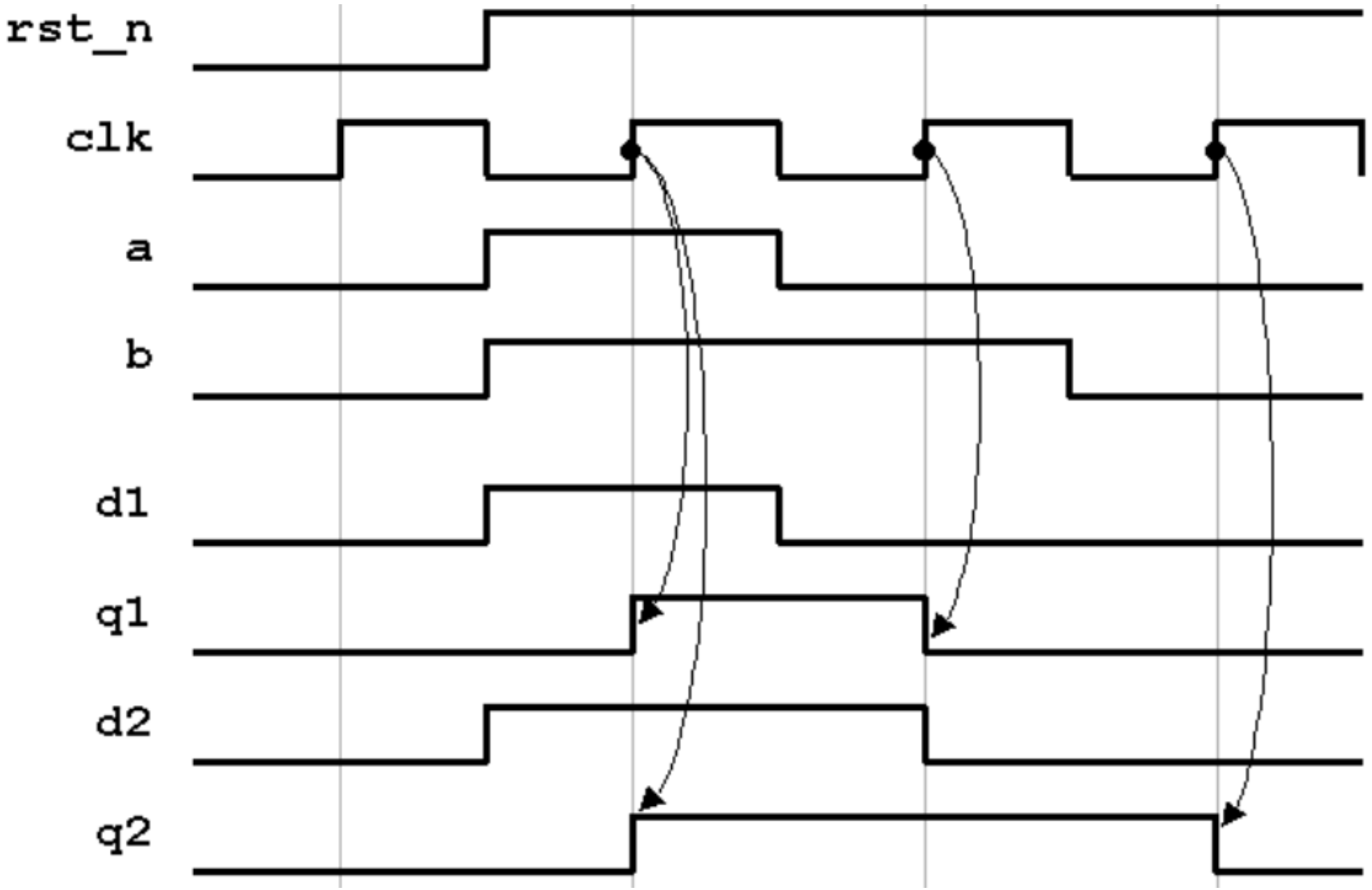
Ex1: First Events



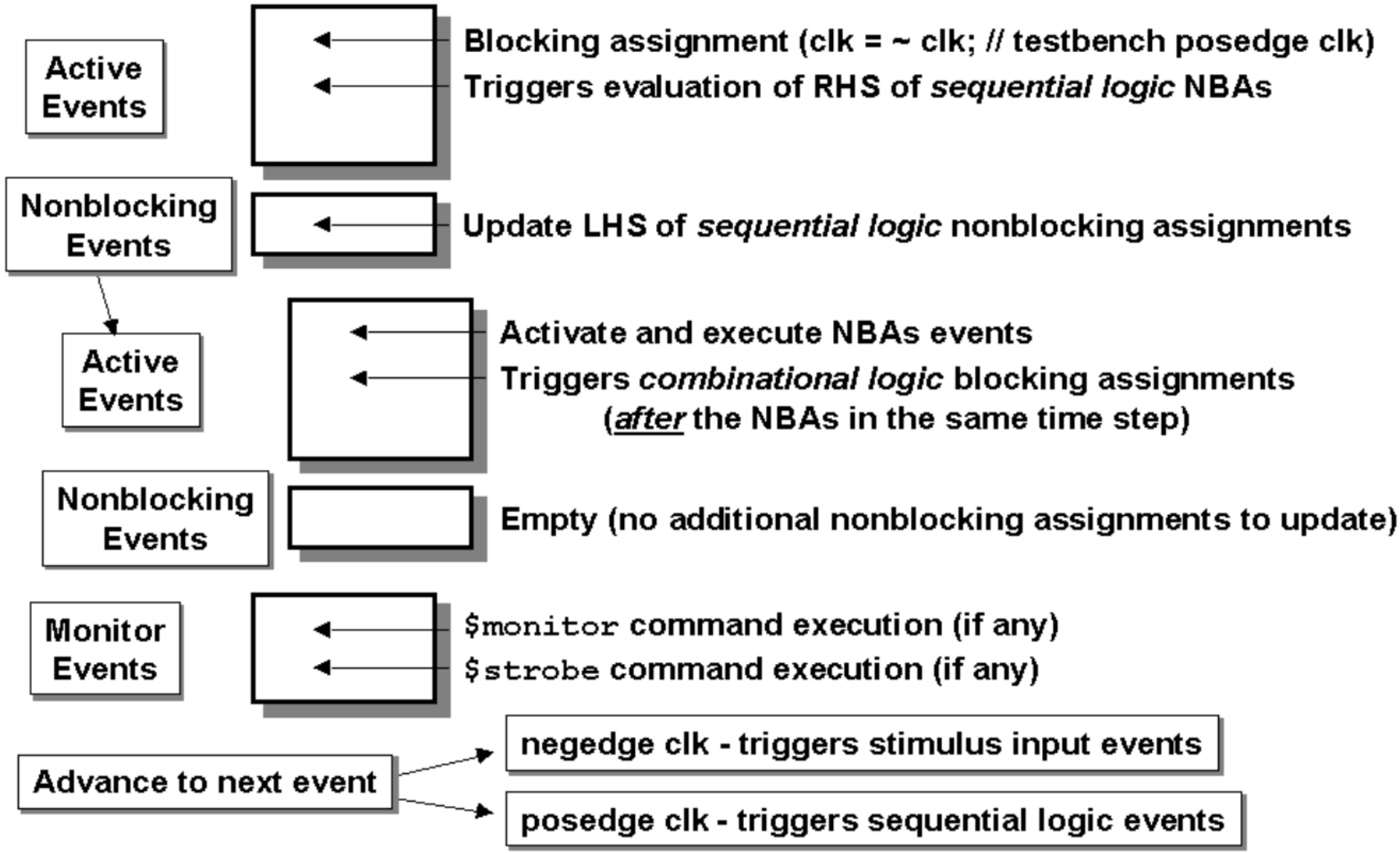
Ex1: Model for these events



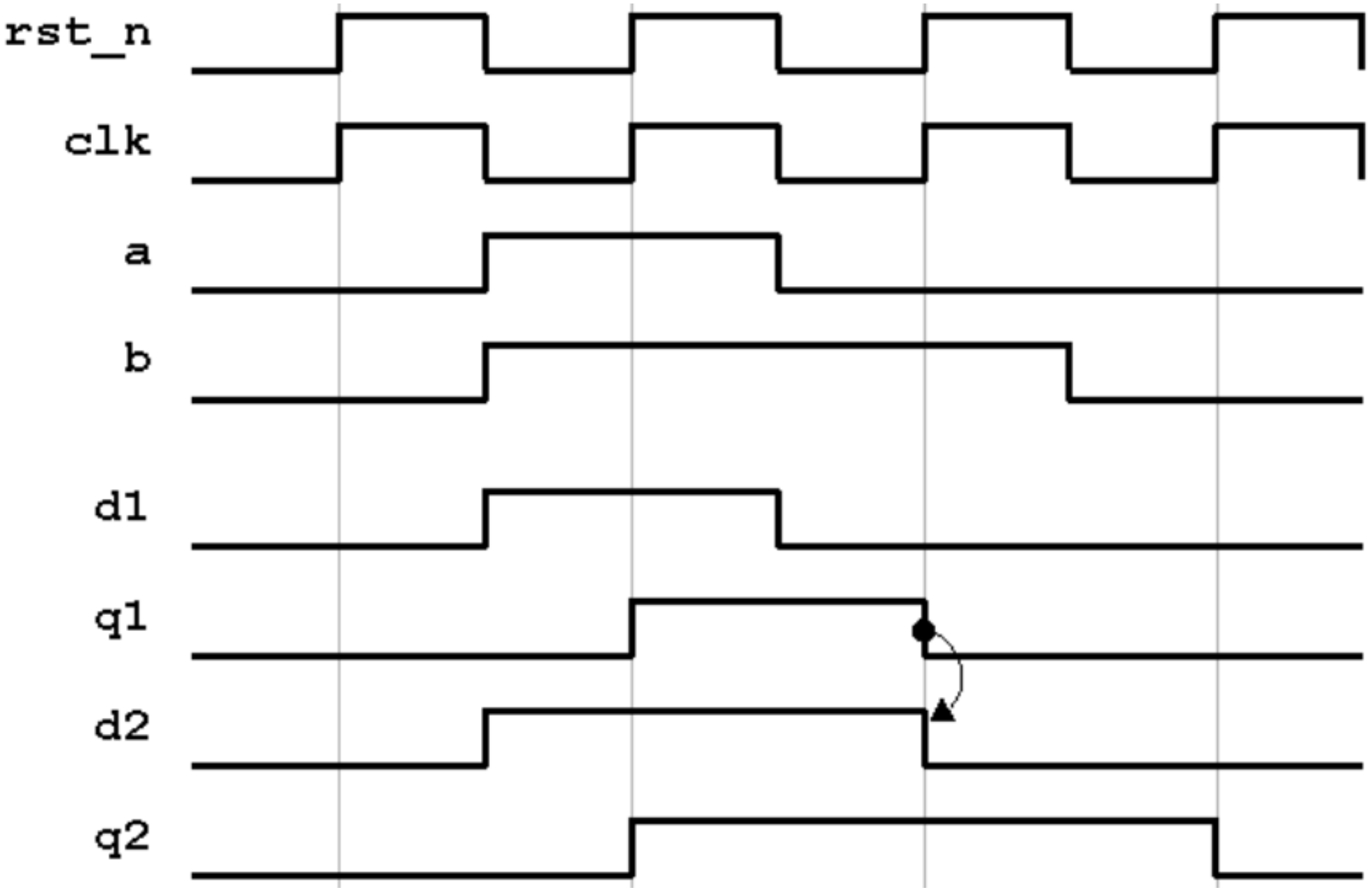
Ex1: Second Events



Ex2: Model for second events



Ex1: Third Events



Race Examples

Races: differing but correct outcomes

```
assign a = b & c;
always @ (b or d) if (a) o = b d;

always @ (posedge clk) b = c;
always @ (posedge clk) a = b;

always @ (b or c)
        if (b != c) errflag = 1;
        else errflag = 0;
always @ (b or d)
        if (b == d) errflag = 1;
        else errflag = 0;
```

```
always @ (posedge clk) q1
    <= t1;
always @ (posedge divclk)
    q2 <= q1;
always @ (posedge clk)
    divclk <= ~divclk;
```

```
initial rst_n = 1'b0;
always @(posedge clk or negedge rst_n)
if (! rst_n) q <= 1'b0;
else q <= d;
```

More about BA and NBA assignment

```
//Consider these:
always @( posedge clk )
begin
  x = z; end
always @( posedge clk )
begin
  x = z;    y = x; end
```

```
//and these:
always @( posedge clk )
begin
  x <= z; end
always @( posedge clk )
begin
  x <= z;    y <= x; end
```

```
//incorrect synth. and sim.
always @(posedge clk) begin q1 = d; q2 = q1; q3 =
    q2; end
```

```
//synthesizes and simulates
always @(posedge clk) begin q3 = q2; q2 = q1; q1 =
    d; end
```

```
//synthesizes, but race in sim.
always @(posedge clk) q1=d;
always @(posedge clk) q2=q1;
always @(posedge clk) q3=q2;
```

Some guidelines

- When modeling sequential logic, use nonblocking assignments.
- When modeling latches, use nonblocking assignments.
- When modeling combinational logic with an always block, use blocking assignments.
- When modeling both sequential and combinational logic within the same always block, use nonblocking assignments.
- Do not mix blocking and nonblocking assignments in the same always block.
- Do not make assignments to the same variable from more than one always block.
- Use $strobe to display values that have been assigned using nonblocking assignments.
- Do not make assignments using #0 delays.

VHDL simulation model

- Concept of delta-delays is used.
- Variables are assigned immediately, Signals are assigned after a delta delay.