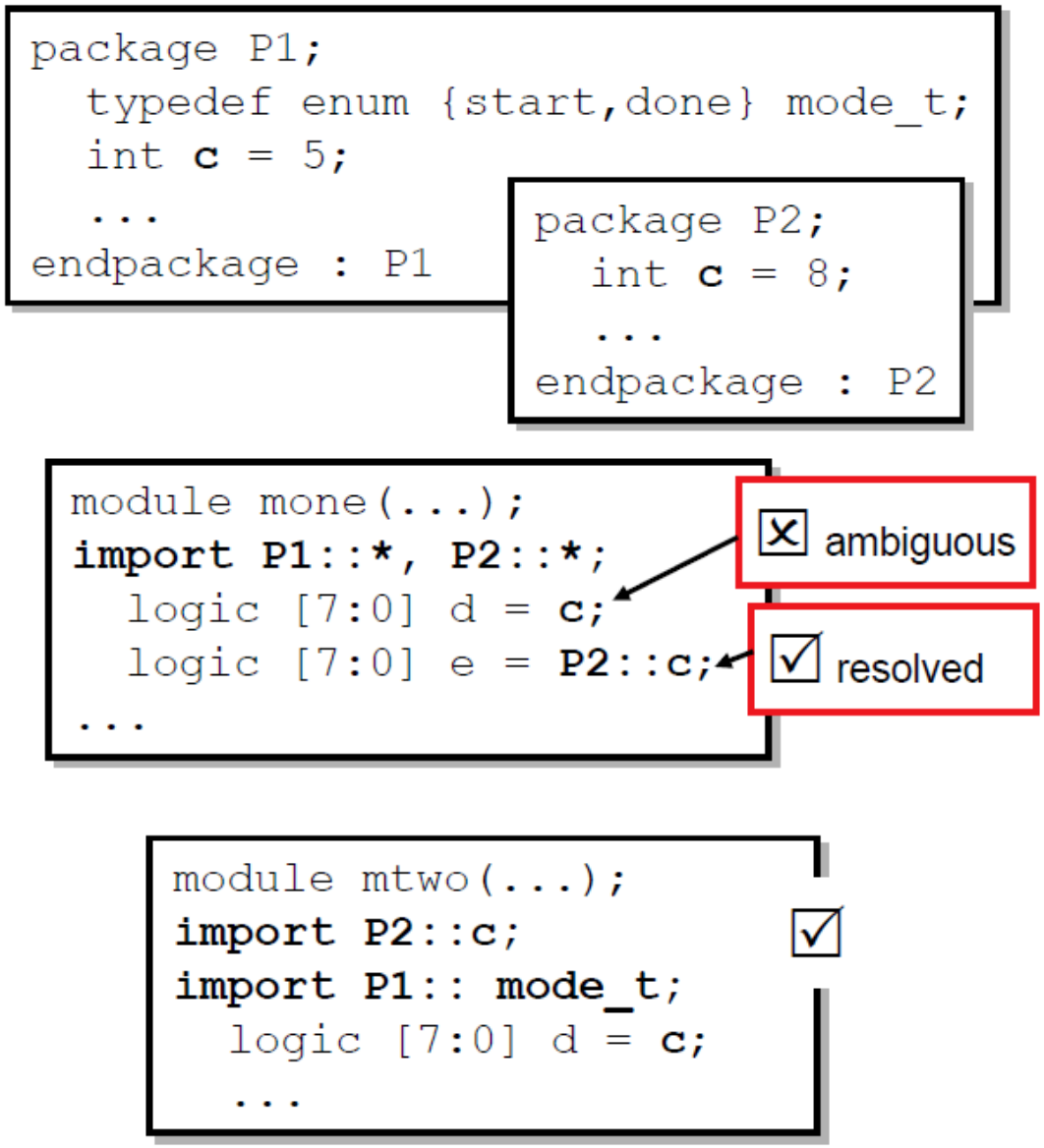
# Packages


package P1;
typedef enum {start,done} mode_t;
int c = 5;
...
endpackage : P1
package P2;
int c = 8;
...
endpackage : P2
module mone(...);
import P1::*, P2::*;
logic [7:0] d = c;
logic [7:0] e = P2::c;
...
ambiguous
resolved
module mtwo(...);
import P2::c;
import P1:: mode_t;
logic [7:0] d = c;
...

# Packages

Find the errors!

```
package P1;
  typedef enum {one, two} mytype1;
  localparam int load = 10;
endpackage : P1
```

```
package P2;
  typedef logic [15:0] mytype2;
endpackage : P2
```

```
import P1::*;
module top;
  import P1::load;
  mytype2 data;
  logic [7:0] count, cnt;
  logic clk, rst, ld, load;
  typedef logic [7:0] mytype1;

  inc U1 (clk, ld, rst, data[7:0], cnt);
  inc U2 (.clk, .ld, .rst, .data, .cnt);
  inc U3 (.*);
  inc U4 (.*, .count);
  inc U5 (.*, .data(data[7:0]);
  ...
endmodule
```

```
module inc (input  logic clk, rst, ld,
            input  logic [7:0] data,
            output logic [7:0] cnt);
...
endmodule
```

# Packages

Answers!

```
package P1;
  typedef enum {one, two} mytype1;
  localparam int load = 10;
endpackage : P1
```

```
package P2;
  typedef logic [15:0] mytype2;
endpackage : P2
```

```
import P1::*;
module top;
  import P1::load;
  mytype2 data;
  logic [7:0] count, cnt;
  logic clk, rst, ld, load;
  typedef logic [7:0] mytype1;

  inc U1 (clk, ld, rst, data[7:0], cnt);
  inc U2 (.clk, .ld, .rst, .data, .cnt);
  inc U3 (.*);
  inc U4 (.*, .count);
  inc U5 (.*, .data(data[7:0]);
  ...
endmodule
```

```
module inc (input  logic clk, rst, ld,
            input  logic [7:0] data,
            output logic [7:0] cnt);
...
endmodule
```

import P2 to define mytype2

load definition already explicitly imported

rst, ld order swapped

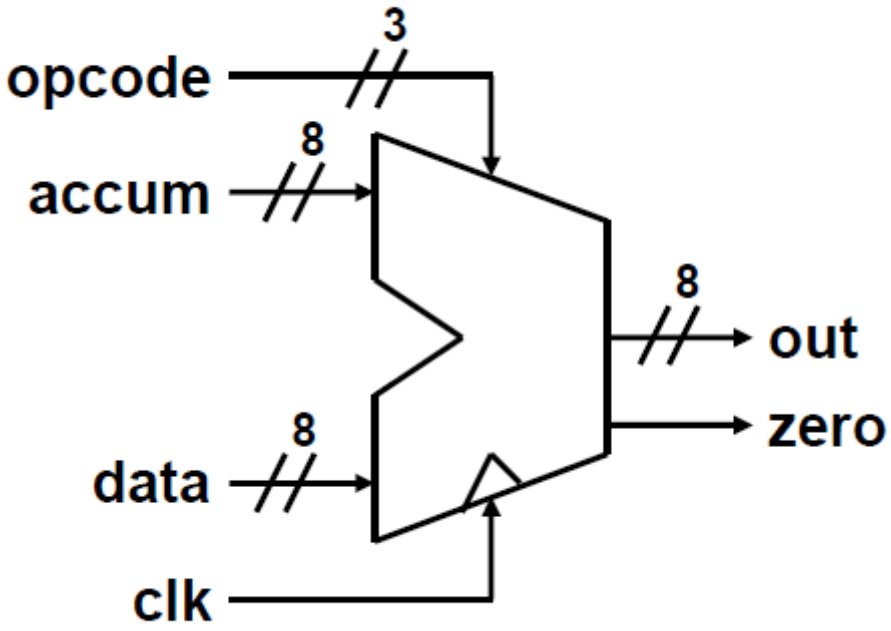
data size mismatch

no port “count”

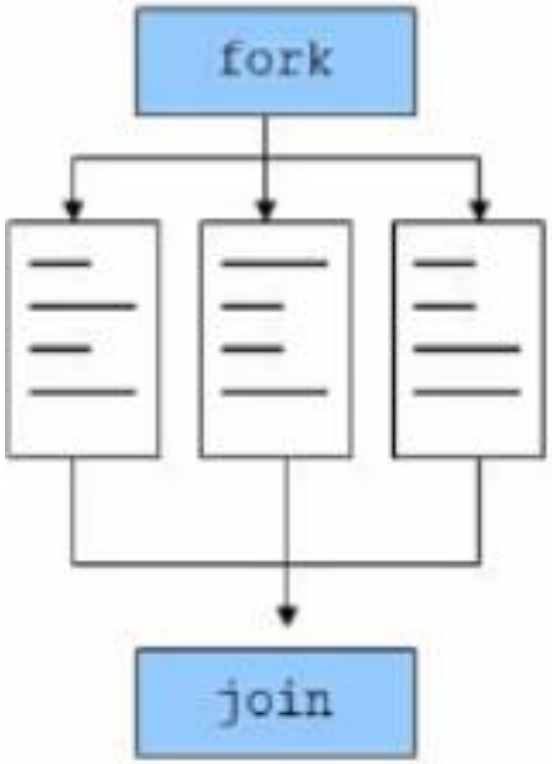
multiple drivers of “cnt” variable

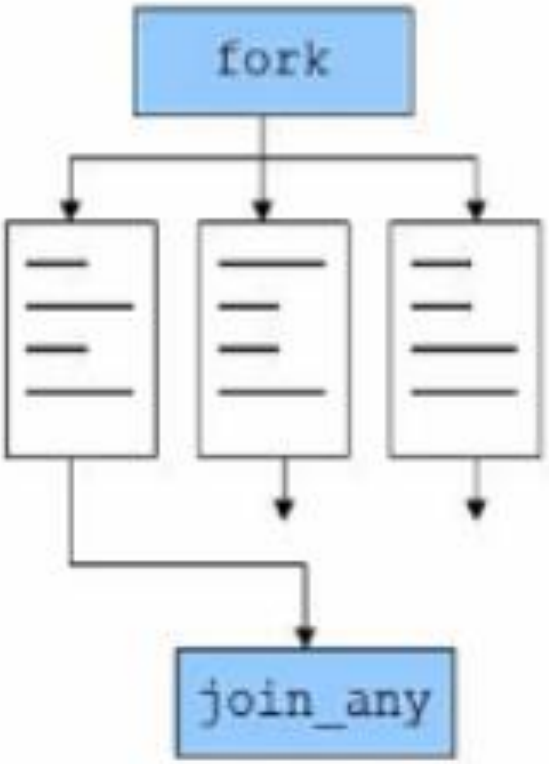
# Packages - Assignment

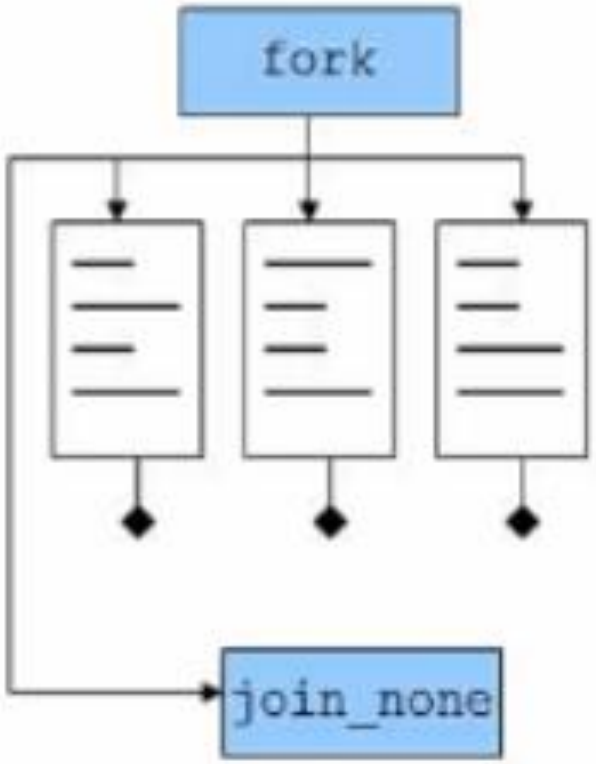
Explore package and import



# Parallel Block Enhancement



- Parallel Block Enhancement join, join_any,join_none

```
module fork_join;

  int a, b, c;

    initial begin
    fork
      begin
        #5 a = 5;
        $display($time, " a=%0d block a", a);
      end
      begin
        #10 b = 10;
        $display($time, " b=%0d block b", b);
      end
      begin
        #20 c = 20;
        $display($time, " c=%0d block c", c);
      end
    //join_any//join //join_none
    join

    $display($time, " Outside fork join");
    end
  endmodule
```

5 a=5 block a
10 b=10 block b
20 c=20 block c
20 Outside fork join

- Parallel Block Enhancement join, join_any,join_none

```
module fork_join;

  int a, b, c;

    initial begin
    fork
      begin
        #5 a = 5;
        $display($time, " a=%0d block a", a);
      end
      begin
        #10 b = 10;
        $display($time, " b=%0d block b", b);
      end
      begin
        #20 c = 20;
        $display($time, " c=%0d block c", c);
      end
    //join_any//join //join_none
    join_any

    $display($time, " Outside fork join");
    end
  endmodule
```

5 a=5 block a
5 Outside fork join
10 b=10 block b
20 c=20 block c

- Parallel Block Enhancement join, join_any,join_none

```
module fork_join;

  int a, b, c;

    initial begin
    fork
      begin
        #5 a = 5;
        $display($time, " a=%0d block a", a);
      end
      begin
        #10 b = 10;
        $display($time, " b=%0d block b", b);
      end
      begin
        #20 c = 20;
        $display($time, " c=%0d block c", c);
      end
    //join_any//join //join_none
    join_none

    $display($time, " Outside fork join");
    end
  endmodule
```

0 Outside fork join
5 a=5 block a
10 b=10 block b
20 c=20 block c

- Parallel Block Enhancement
  join, join_any,join_none
- disable fork

```
module fork_join;

  int a, b, c;

    initial begin
    fork
      begin
        #5 a = 5;
        $display($time, " a=%0d block a", a);
      end
      begin
        #10 b = 10;
        $display($time, " b=%0d block b", b);
      end
      begin
        #20 c = 20;
        $display($time, " c=%0d block c", c);
      end
    //join_any//join //join_none
    join

    $display($time, " Outside fork join");
    disable fork;
    end
  endmodule
```

5 a=5 block a
10 b=10 block b
20 c=20 block c
20 Outside fork join

- Parallel Block Enhancement join, join_any,join_none
- disable fork

```
initial begin

fork
  begin
    #5 a = 5;
    $display($time, " a=%0d - block a", a);
  end
  begin
    #10 b = 10;
    $display($time, " b=%0d - block b", b);
  end
  begin
    #20 c = 20;
    $display($time, " c=%0d - block c", c);
  end
join_any//join //join_none

$display($time, " Outside fork join");
disable fork;

end
```

```
run
           5 a=5 - block a
           5 Outside fork join
```

- Parallel Block Enhancement
  join, join_any,join_none
- disable fork

```
join_none

  $display($time, " Outside fork join");
  disable fork;
```

```
run
           0 Outside fork join
```

- Parallel Block Enhancement
  join, join_any,join_none
- wait fork

```
module fork_join;

  int a, b, c;

    initial begin
    fork
      begin
        #5 a = 5;
        $display($time, " a=%0d block a", a);
      end
      begin
        #10 b = 10;
        $display($time, " b=%0d block b", b);
      end
      begin
        #20 c = 20;
        $display($time, " c=%0d block c", c);
      end
    //join_any//join //join_none
    join

    $display($time, " Outside fork join");
    //disable fork;
    wait fork;
    end
  endmodule
```

5 a=5 block a
10 b=10 block b
20 c=20 block c
20 Outside fork join

- Parallel Block Enhancement join, join_any,join_none
- wait fork

```
module fork_join;

  int a, b, c;

    initial begin
    fork
      begin
        #5 a = 5;
        $display($time, " a=%0d block a", a);
      end
      begin
        #10 b = 10;
        $display($time, " b=%0d block b", b);
      end
      begin
        #20 c = 20;
        $display($time, " c=%0d block c", c);
      end
    //join_any//join //join_none
    join_any

    $display($time, " Outside fork join");
    //disable fork;
    wait fork;
    end
  endmodule
```

5 a=5 block a
5 Outside fork join
10 b=10 block b
20 c=20 block c

- Parallel Block Enhancement join, join_any,join_none
- wait fork

```
module fork_join;

  int a, b, c;

    initial begin
    fork
      begin
        #5 a = 5;
        $display($time, " a=%0d block a", a);
      end
      begin
        #10 b = 10;
        $display($time, " b=%0d block b", b);
      end
      begin
        #20 c = 20;
        $display($time, " c=%0d block c", c);
      end
    //join_any//join //join_none
    join_none

    $display($time, " Outside fork join");
    //disable fork;
    wait fork;
    end
  endmodule
```

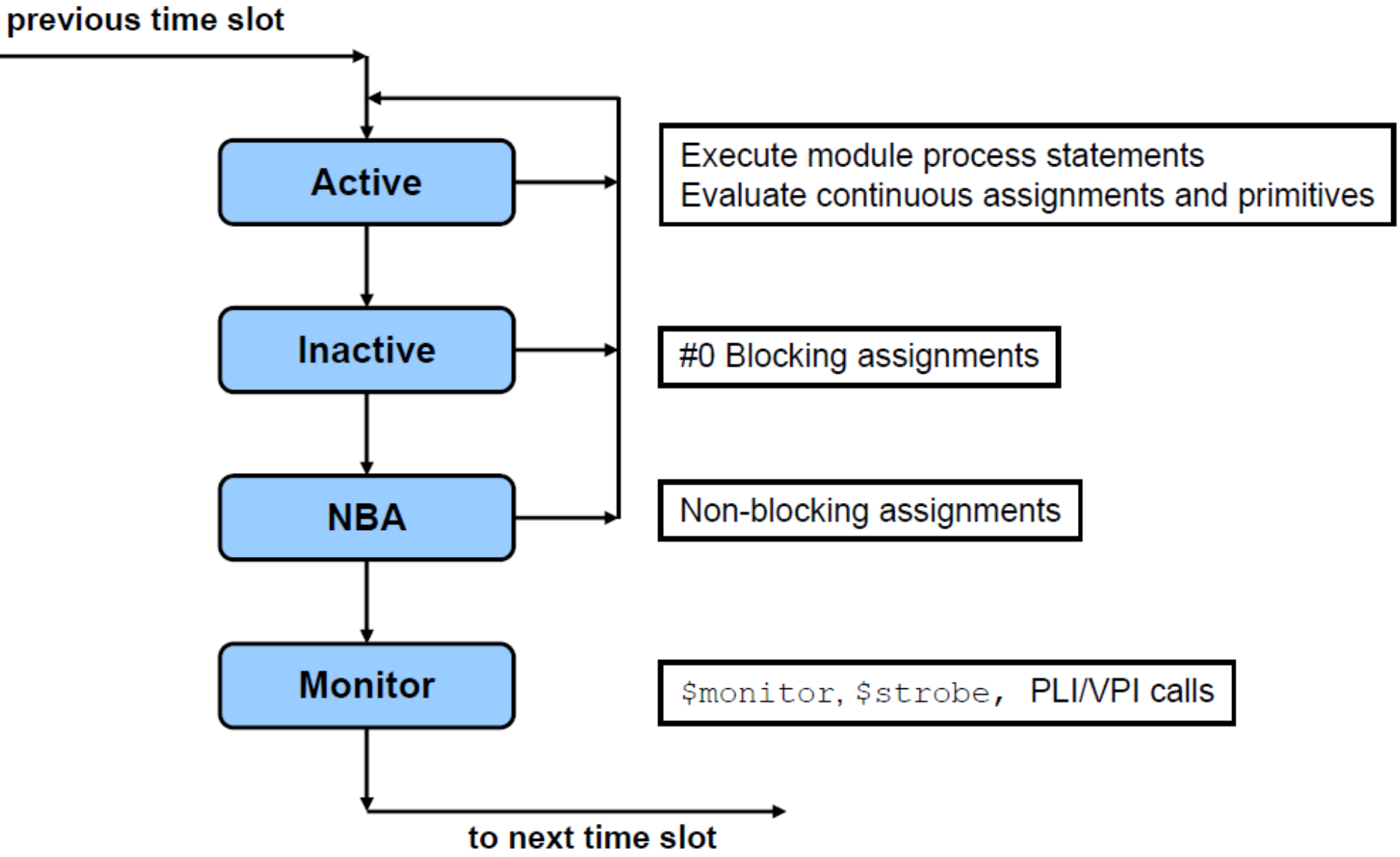
0 Outside fork join
5 a=5 block a
10 b=10 block b
20 c=20 block c

- Parallel Block Enhancement join, join_any,join_none
- wait fork

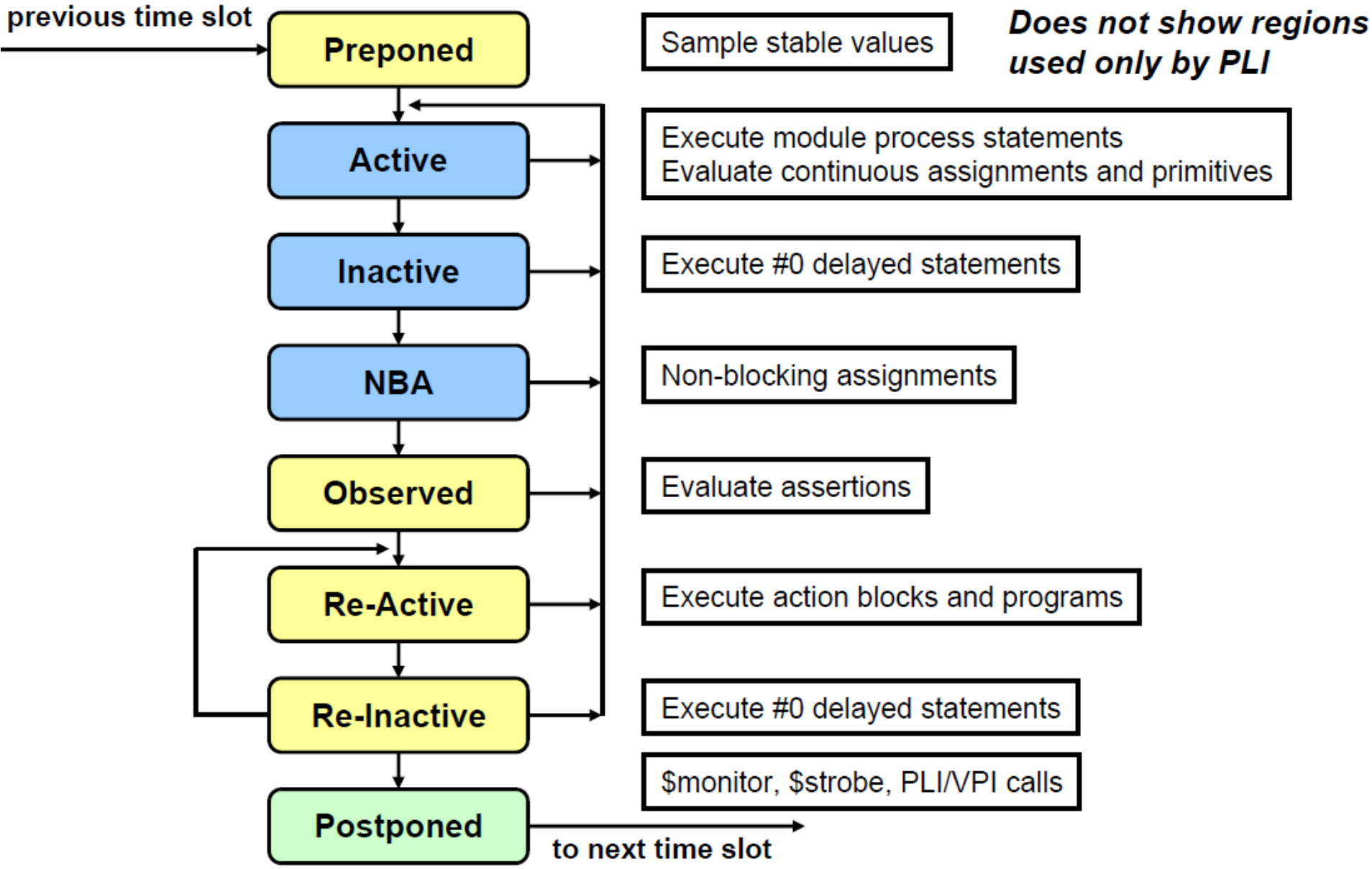
```
run
            5 a=5 - block a
            5 Before wait fork
           10 b=10 - block b
           25 c=20 - block c
           25 After wait fork
```

```
fork
  begin
    #5 a = 5;
    $display($time, " a=%0d - block a", a);
  end
  begin
    #10 b = 10;
    $display($time, " b=%0d - block b", b);
  end
join_any

fork
  begin
    #20 c = 20;
    $display($time, " c=%0d - block c", c);
  end
join_none

$display($time, " Before wait fork");
wait fork;
$display($time, " After wait fork");

end
```

# Verilog Event Scheduler



# SystemVerilog Event Scheduler



# Program block

- Provides race free communication between testbench and module
  - Get executed in reactive region

Ex: Without program block

```
module design_ex(output bit [7:0] a);

  initial begin

    a <= 8'b10101010;

  end

endmodule
```

```
module testbench();

  bit [7:0] a;

  initial begin
    $display("a = %b",a);
  end

endmodule
```

a = 00000000

# Program block

- Provides race free communication between testbench and module
  - Get executed in reactive region

```
module design_ex(output bit [7:0] a);

  initial begin

    a <= 8'b10101010;

  end

endmodule
```

```
program my_pgm(input bit [7:0] a);

  initial begin
    $display("\t a = %0b",a);
  end

endprogram
```

```
`include "top.sv"

module testbench;
  wire [7:0] a;

  design_ex dut(.*);
  my_pgm test(.*);

endmodule
```

a = 10101010

# final block

- Executes at the end of simulation

```
module final_block;
  real frequency = 20.0;
  bit [8:0] s[5];

 initial begin

  for(int i = 0; i <5; i++) begin
    s[i] = i;
    $display($time, " for block s[%0d] = %0d", i, s[i]);
    #10;
  end

   foreach(s[j]) begin
    s[j] = s[j] + 10;
     $display($time, " foreach block s[%0d] = %0d", j, s[j]);
    end
  #10;
  end

final
  $display("number of cycles executed = %0f", $time*frequency);

endmodule
```

```
run
          0 for block s[0] = 0
         10 for block s[1] = 1
         20 for block s[2] = 2
         30 for block s[3] = 3
         40 for block s[4] = 4
         50 foreach block s[0] = 10
         50 foreach block s[1] = 11
         50 foreach block s[2] = 12
         50 foreach block s[3] = 13
         50 foreach block s[4] = 14
         number of cycles executed = 1200.000000
```

# clocking block

- A clocking block specifies timing and synchronization for a group of signals.
- The clocking block specifies,
  - The clock event that provides a synchronization reference for DUT and testbench
  - The set of signals that will be sampled and driven by the testbench
  - The timing, relative to the clock event, that the testbench uses to drive and sample those signals
- Clocking block can be declared in interface, module or program block.

```
module counter(input clk, rst, enb, load, updn,
               input [7:0] data, output reg[7:0] out);
  always @(posedge clk or posedge rst)
    if (rst)
      out <= 0;
    else
      if (enb)
        if (load)
          out <= data;
        else
          if (updn)
            out <= out + 1;
          else
            out <= out - 1;
endmodule
```

- ## - clock cycles
  - ## 5 –> 5 clock cycles
- # - clock skews
- #1step – Sampling on the preponed region of the current time stamp

```
program clk_blk(
  input wire clk,
  input [7:0] out,
  output [7:0] data,
  output logic rst, enb, load, updn
);

    // cloking outputs are DUT inputs and vice versa
    default clocking cb_counter @(posedge clk);
      default input #1step output #4;
      output negedge rst;
      output enb, load, updn, data;
      input out;
    endclocking

    initial begin
      // Set all inputs at the beginning
      enb = 0;
      load = 0;
      updn = 1;
      rst = 1;
      // Will be applied on negedge of clock
      ##1 cb_counter.rst  <= 0;
      // Will be applied 4ns after the clock
      ##1 cb_counter.enb <= 1;
      ##2 cb_counter.updn   <= 0;
      ##4 cb_counter.updn   <= 1;
    end
    endprogram
```

```
`include "clk_blk.sv"
module clk_blk();
  timeunit 1ns;

  reg clk = 0;
  reg rst, enb, load, updn;
  reg [7:0] data;
  wire [7:0] out;

  counter G1(.*);
  clk_blk T1(.*);

  always #5 clk = ~clk;

initial begin
  $monitor($time, "clk=%b, rst=%b, enb=%b, load=%b, updn=%b, data=%h, out=%h",
                clk, rst, enb, load, updn, data, out);
  $dumpfile("test.vcd");
  $dumpvars;
  #500 $finish;

end

endmodule
```

```
// Will be applied on negedge of clock
##1 cb_counter.rst  <= 0;
// Will be applied 4ns after the clock
##1 cb_counter.enb <= 1;
##2 cb_counter.updn   <= 0;
##4 cb_counter.updn   <= 1;
```