Appendix D: Linked List

Appendix D: Linked List

- Linked List is a parameterized class
- Two types:

  DList#(parameter type T); // element

  DListIter#(parameter type T); // iterator
- Built-in Class
- No need for macros or header files
- All other methods stay the same!
- Edit: Purge Vera references

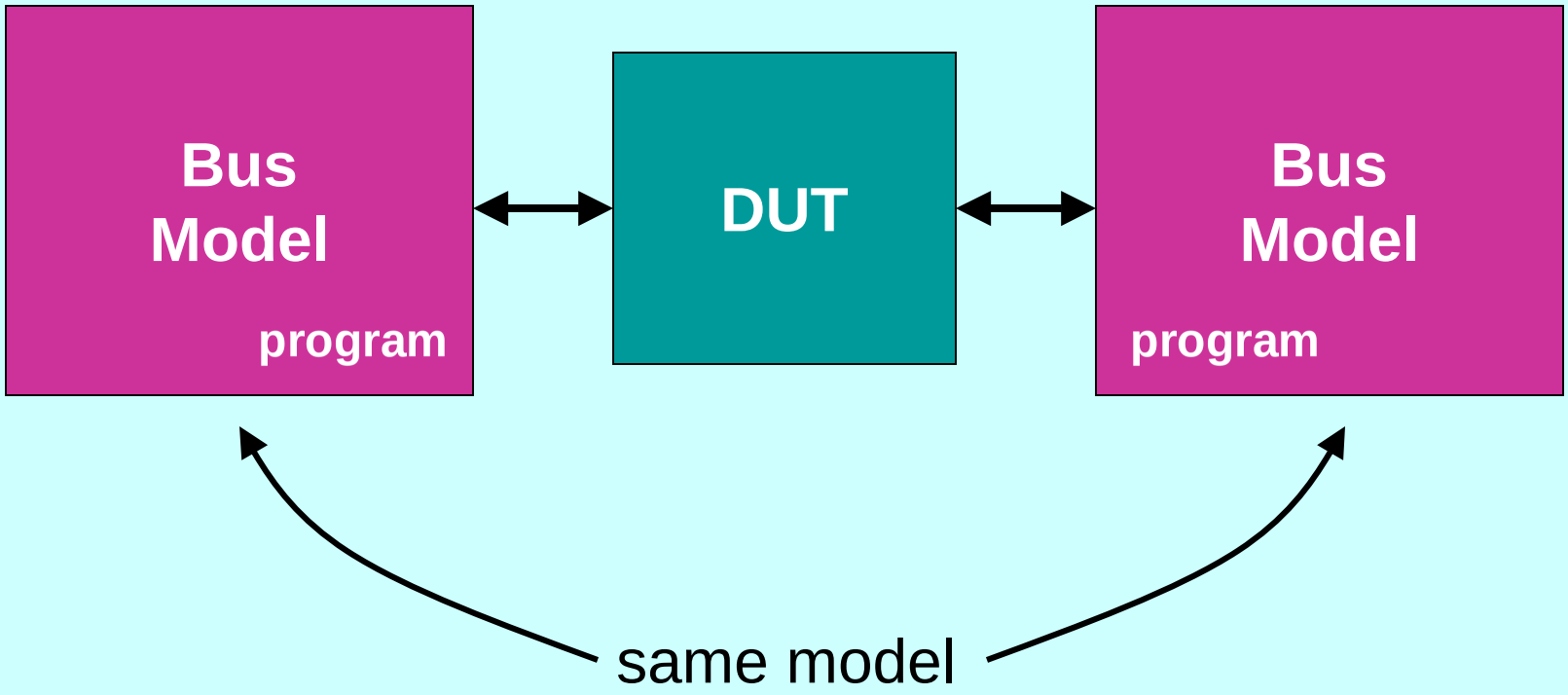
Program Block

Program

Motivation

- Provide a syntactical construct that allows users to specify **reactive region** execution
- Allow encapsulation of model routines & data
  - Scope
  - Verification Variables vs. Design Variables
    - Reactive-region Variables
    - Reactive-region tasks and methods
- Functional model re-use: Multiple instances

Program Re-Use



Program as Hierarchical Context

(Jay Lawrence's Proposal)

- Program is similar to an **initial** block
- Benefits
  - No need to instantiate program
  - Uses Verilog's natural hierarchy and interconnect mechanisms
  - Allows multiple cooperating programs in a module, sharing module data

Program as Hierarchical Context

- Problem
  - Inability to declare tasks, functions, methods (classes) with **Reactive** execution semantics

    Blocking tasks

```
task cover_nth( int cycles, ref int j );
  ## [cycles];
  if( property_status( p1 ) )   // wrong property status!
        j++;
endtask
```

Program as Hierarchical Context

- Problem
  - Inability to distinguish design and verification tasks
  - Easy for users to get into trouble
    - Create race conditions unwittingly
    - Create false-positives
      - Design appears to work due to interference of testbench/functional-model
  - Compiler can’t help: All tasks look the same

Program

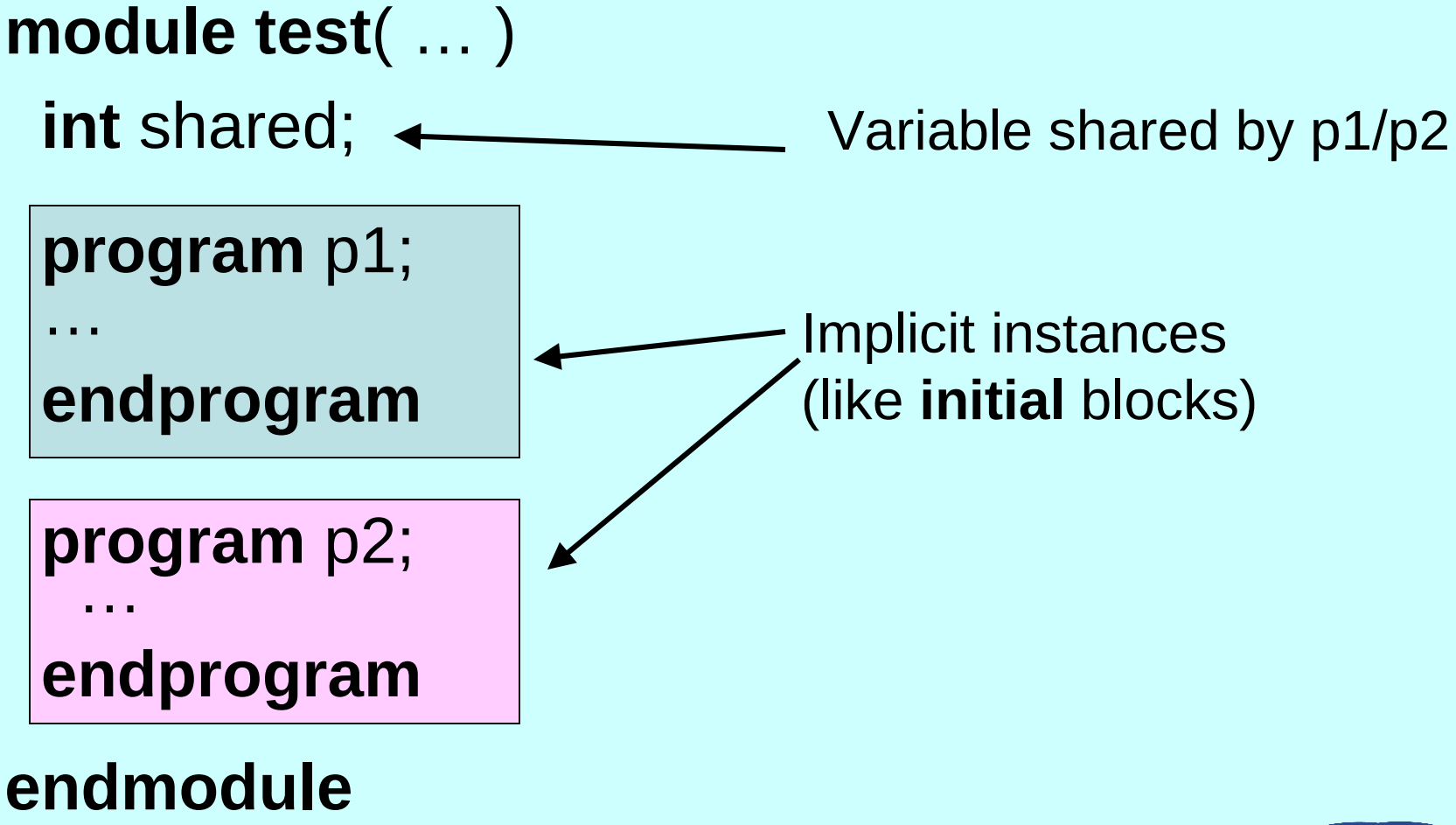
Solution

- Implicit Program instantiation
- Allows both models to mix seamlessly

How does this work?

- Nested programs (within a module) that have no ports and are not instantiated explicitly, are instantiated implicitly once

Implicit Program Example



Changes

CH-60

- Section 13.13 Input sampling
- Replace

  If the input skew is zero then the value sampled corresponds to the signal value at the start of the verification phase
- With

  If the input skew is zero then the value sampled corresponds to the signal value during the **observe** region

CH-84

2 for, 5 against

- Section 13.3
- Input signals with zero skew are sampled at the same time as their corresponding clock edge clocking event, but to avoid races, the sampling is done at the observe region. Output signals with zero output skew are driven at the same time as their specified clock edge clocking event, as non-blocking assignments (in the NBA region).

CH-96

- Section 13.14
- Clocking domain drives are always assigned as NBA's.
- Propose changing the syntax to use the non-blocking assignment:
  - **##[3] a <= b;**
  - **a <= ##[3] b;**

CH-97

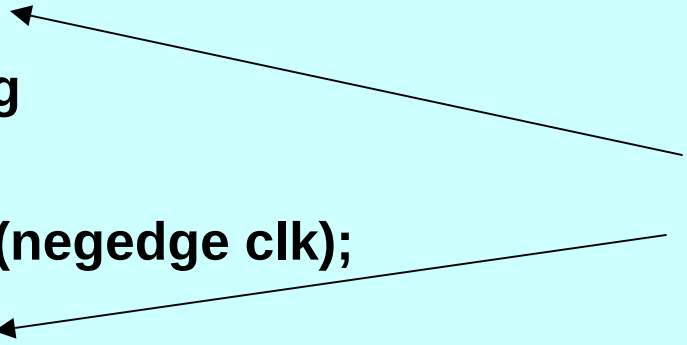
- Section 13.14.2

When the same variable is an output from multiple clocking domains, the last drive determines the value of the variable. This allows a single module to model multi-rate devices, such as a DDR memory, using a different clocking domain to model each active edge. Naturally, clock-domain outputs driving a net (i.e., through different ports) cause the net to be driven to its resolved signal value.

```
reg j;

clocking @(posedge clk);
    output j;
endclocking

clocking @(negedge clk);
    output j;
endclocking
```



CH-98

- Section 13.14.1
- Synchronous signal drives are queued and processed at the end of the verification phase, like nonblocking assignments, that is, they are propagated in one fell swoop without process execution in between drives.

- Synchronous signal drives are processed as non-blocking assignments.