Synopsys Users Group
UK 2012

I Spy with My VPI:

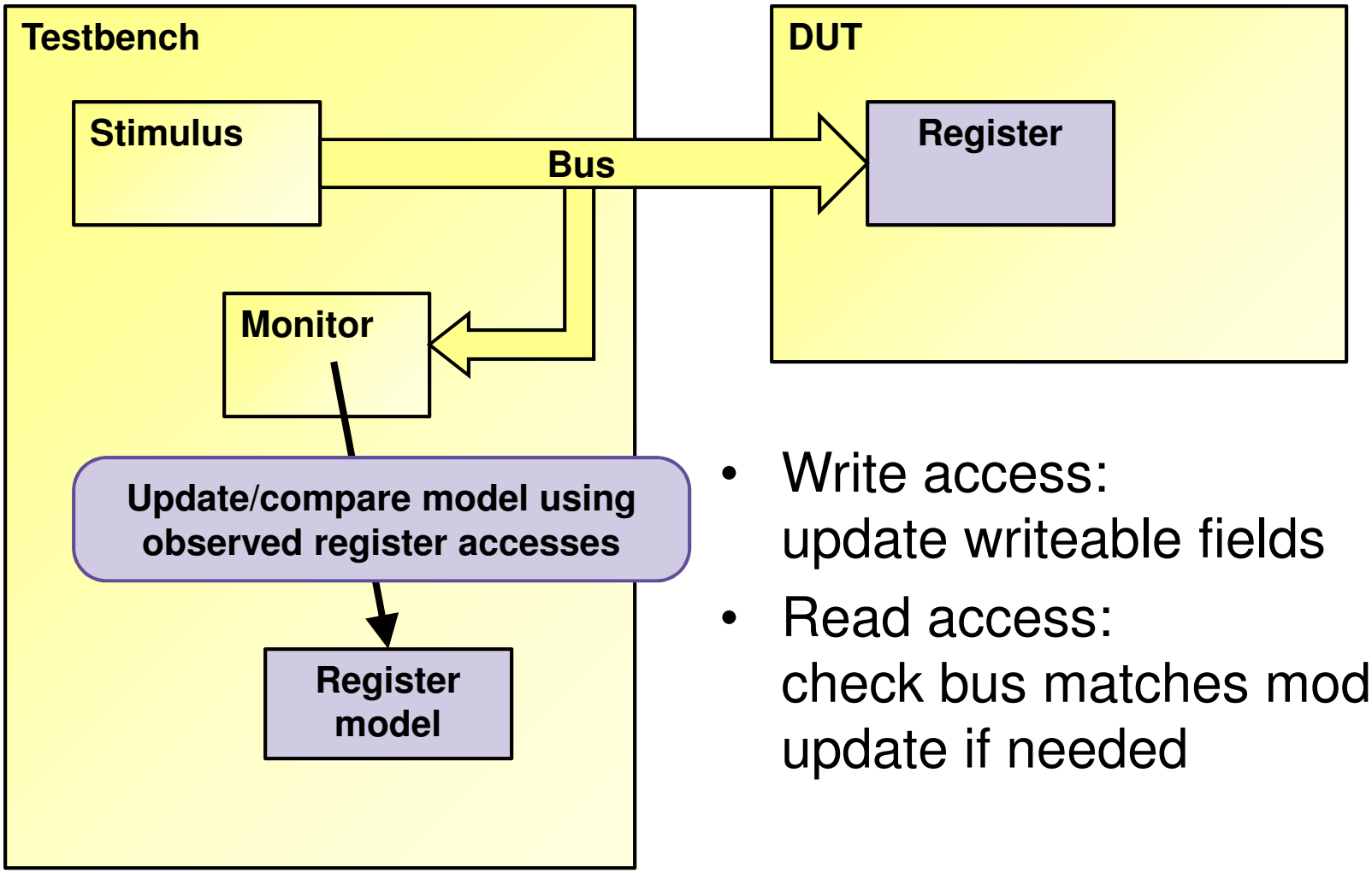
Monitoring signals by name, for the UVM Register Package and more

Jonathan Bromley
Verilab

What do we have here?

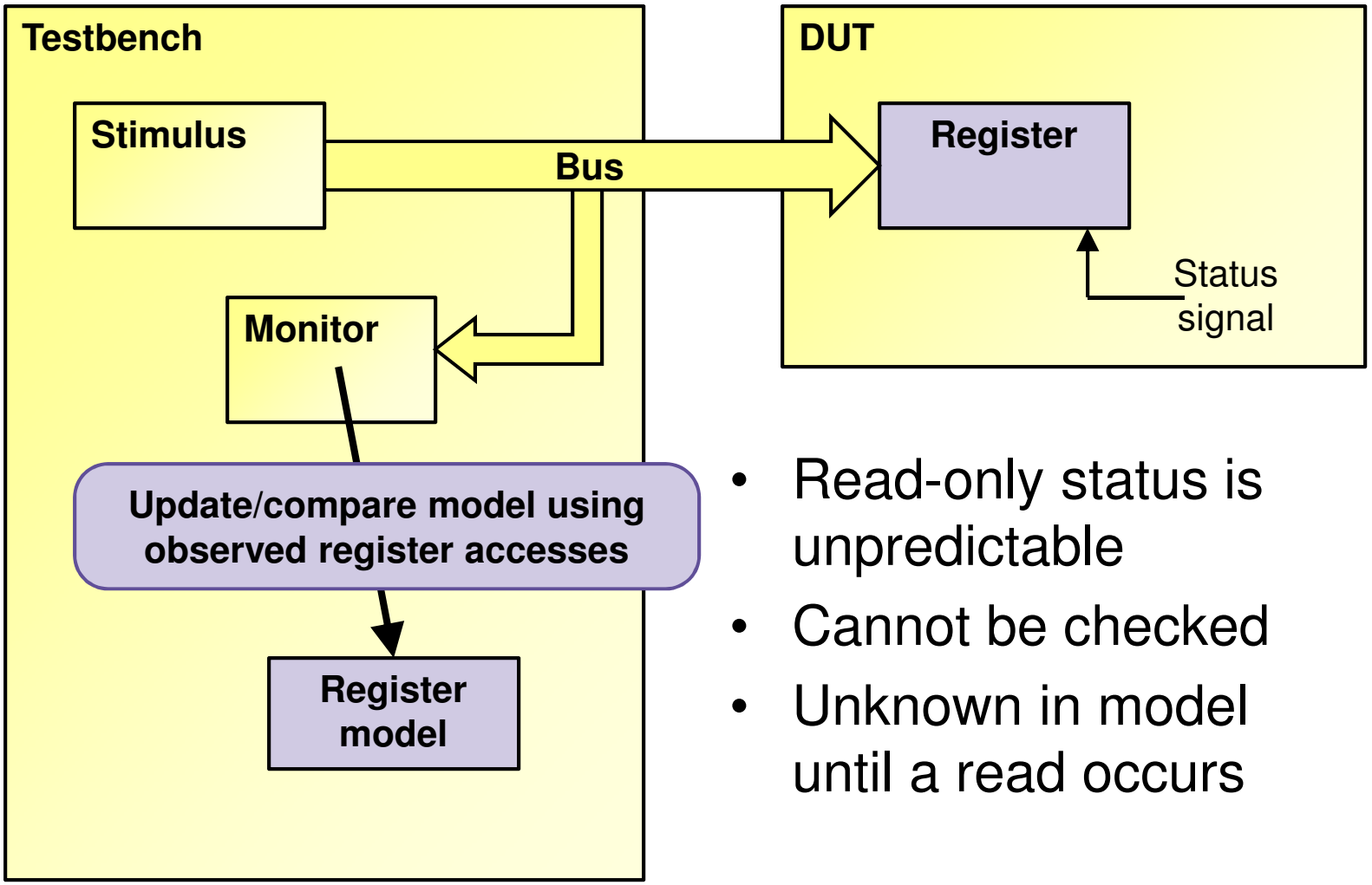
- Principal motivation: register modelling
  - Timely update of status register model values
  - Existing UVM-REG backdoor access mechanism
- New solution for value-change detection
  - Users' view of the package
  - Additional features and other applications
- How it works: internal implementation details
- Concerns: performance, known issues
- Wrap-up

Updating a register model



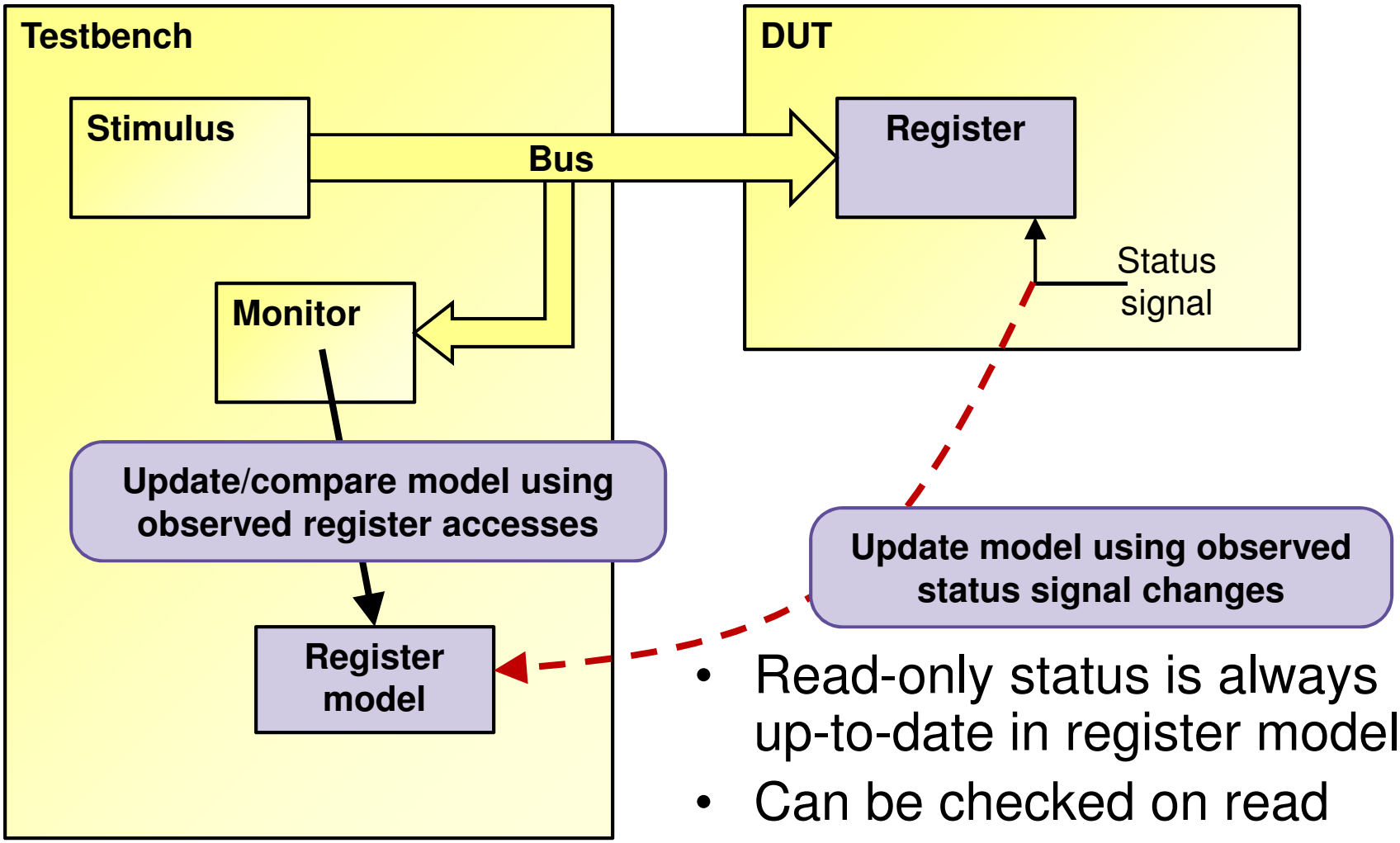
- Write access:
  update writeable fields
- Read access:
  check bus matches model, update if needed

The problem of status registers



- Read-only status is unpredictable
- Cannot be checked
- Unknown in model until a read occurs

Active monitoring



- Read-only status is always up-to-date in register model
- Can be checked on read

Active monitoring in UVM-REG

- Support infrastructure already exists
  - users merely implement `wait_for_change()`

```
task wait_for_change();
  @$root.top.dut.regbank.irq_reg.status;
endtask
```

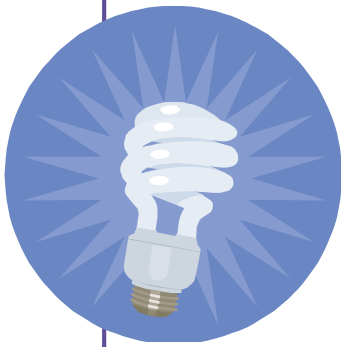
- Infrastructure updates model on every change
  - but this uses string hdl_path – *duplication*
- Can we use hdl_path for both purposes?

> ***"Because there is no standard value-change callback VPI or PLI functionality, the automatic update of a field can only be implemented using a user-defined back-door..."***
>
> **UVM 1.1 User Guide**

VPI value-change callback

- VPI offers a standard value-change callback
- *But it can only call functions in C*
  - There is no context for calling a DPI export
  - No standard way to call back to Verilog!

- **Callback function in C uses VPI to toggle SystemVerilog package variable** `notifier`
- `@pkg::notifier` **triggers Verilog activity**
- **SystemVerilog can then DPI-call back into C to identify the cause (which signal changed)**



User's view of the package

- Import just one class from the package

```
import vlab_probes_pkg::signal_probe;
```

- Create a probe object on your chosen signal

```
signal_probe statusProbe;
string regPath; // configured as "top.dut.regbank.irq_reg"
function void start_of_simulation_phase( uvm_phase phase );
  statusProbe =
       signal_probe::create( {regPath, ".status"} );
endfunction
```

Or borrow the register model's `hdl_path`

- Wait for value changes!

```
  ...
  statusProbe.waitForChange();
  ...
```

Vendor and methodology agnostic

- The package is plain SystemVerilog
  - Class `signal_probe` does not inherit from UVM or any other base class library
- Easy to use within any methodology
  - including "home grown" approaches
- No dependence on vendor features
  - Standard VPI and DPI functionality
  - Tested on all major SystemVerilog simulators
  - Some care needed with compile scripts

Additional features

- Other inspection features were easy to add
- No overhead if you don't use them

```
int numBits = statusProbe.getSize();  // vector width
bit signing = statusProbe.isSigned(); // signed or unsigned
string fullName = statusProbe.getName();
```

- Enable/disable value-change detection on the fly

```
statusProbe.setVcEnable(1);  // or 0 for disable
```

- Mimic a value-change event (debug or cleanup)

```
statusProbe.releaseWaiters();
```

Read the signal's value

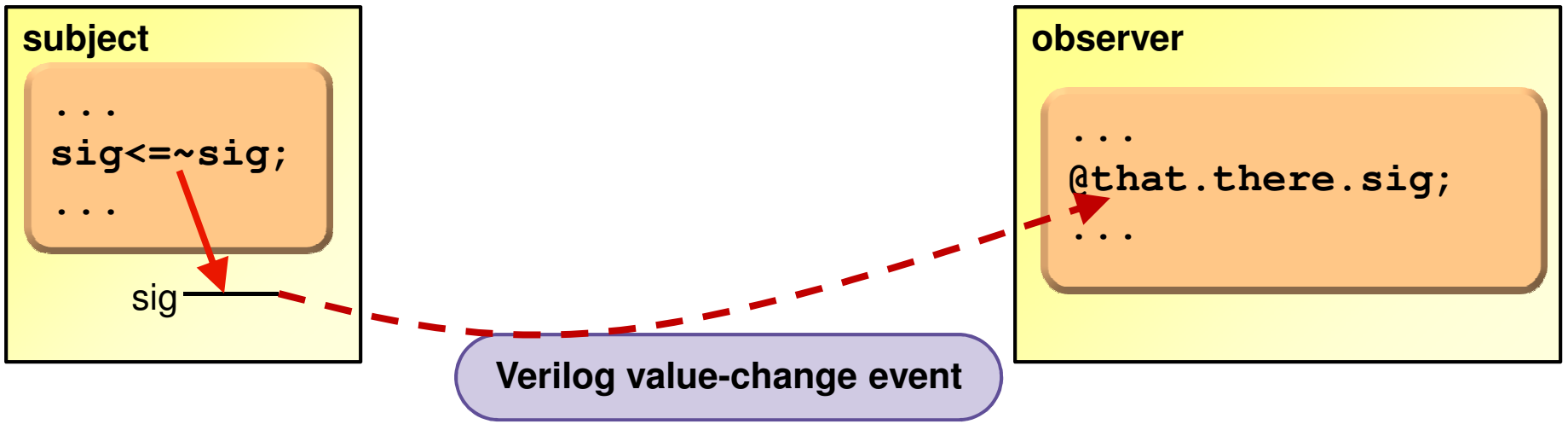
- Returns 32-bit result for any size of signal
- Zero-extended or sign-extended
  - according to signing of the actual signal
- Can be written to narrower vector (Verilog!)

```
logic status = statusProbe.getValue32();
```

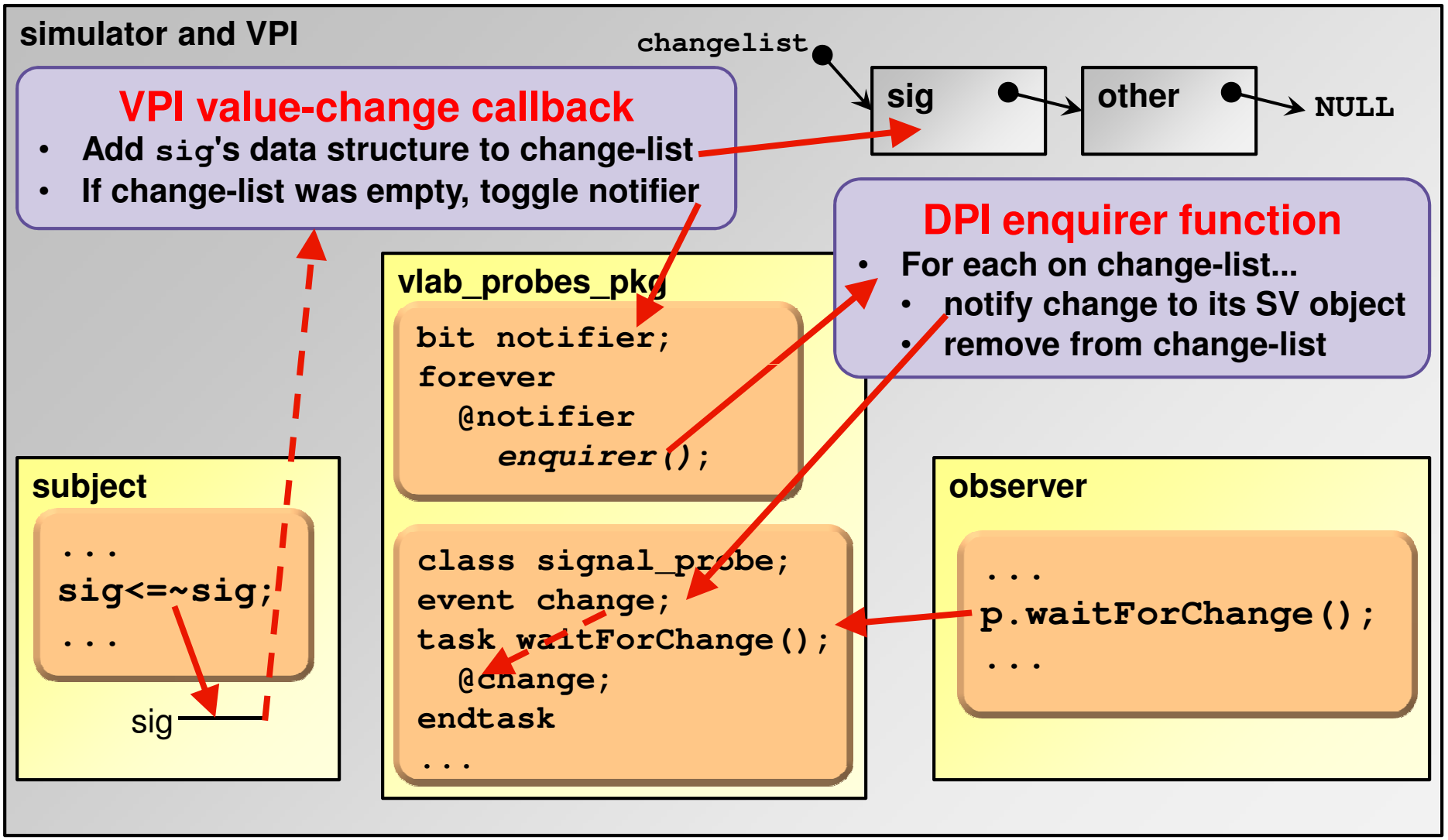
- Any size vector can be read in 32-bit chunks
  - No compiled-in hard limit

```
logic [55:0] value;
value[31:0] = statusProbe.getValue32();
value[55:32] = statusProbe.getValue32(.chunk(1));
```

Value-change detection flow



Value-change detection flow



Performance considerations

- First application: DUT had ~4000 registers
  - performance was a critical concern
- All internal operations are **constant time**
  - scales gracefully to larger problems
- Insignificant setup cost
  - thousands of probes created per second of runtime
- **Runtime cost per value-change event**
  - penalty per *million* events: ~2 seconds
- Benefits outweigh penalty in practice
  - but care is needed: Don't probe a 2GHz clock!

VCS command line options

- VCS fully automates DPI/VPI compilation
  - No `pli.tab` needed for this package

```
$ vcs +vpi +acc+2 -sverilog \
     vlab_probes.c vlab_probes_pkg.sv ...
```

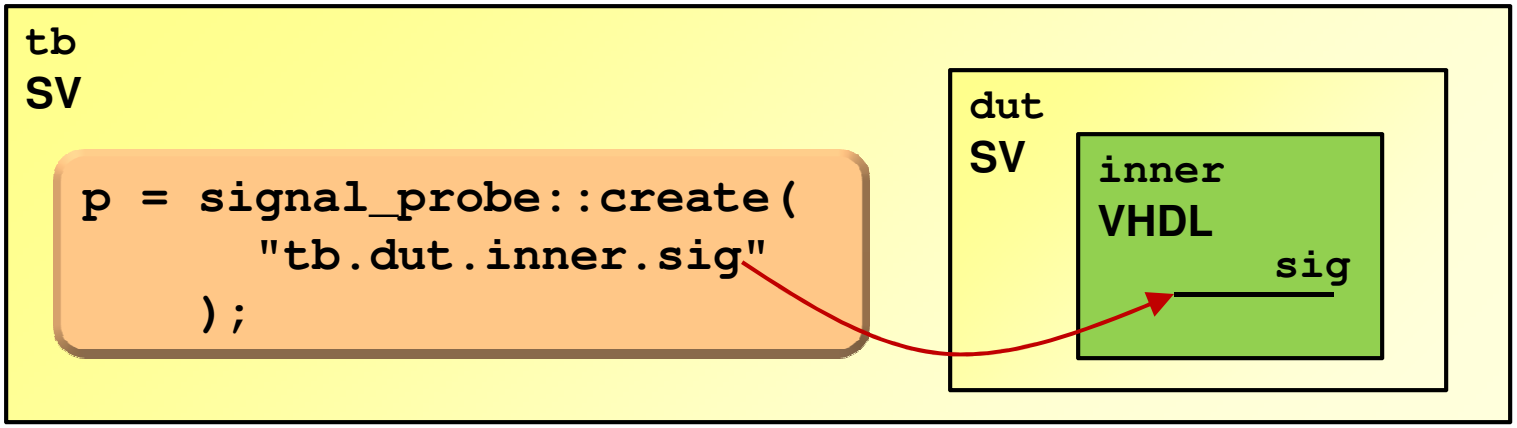
- Global VPI access may degrade performance
- Consider 2-step flow using `+vcs+learn+pli`:

```
$ vcs +vpi +acc+2 -sverilog +vcs+learn+pli ...
$ ./simv
...
$ vcs +apply+learn -sverilog ...
```

Creates `pli_learn.tab`

Uses `pli_learn.tab`

Probing VHDL signals



- Depends on non-standard VPI behaviour
  - available simulators vary
- Continuing work on making the code portable
- Expect some limitations
  - value-change detection is the critical must-have

Known issues

- Message integration with UVM, VMM etc
  - Current package writes messages directly to console
  - Alternative methodology-friendly approaches are under discussion

- Checkpoint save/restore is not supported
  - Challenging for any VPI application that keeps state
  - Silver bullets gratefully received!
- Simulator restart is fully supported
  - no known memory leaks

Wrap-up

- First release of the code is freely available today

www.verilab.com

- Suggestions for improvement are welcome
- Thanks...
  - to you for listening
  - to many others: see *Acknowledgements* in paper

Questions?