
Using Programmer's View Timing Annotation for the Creation of Reusable Transaction Level Models

Tim Kogel
CoWare, Inc.

Agenda

- TLM 2.0 Standard Interfaces
 - Untimed
 - Annotated
- Reusing Programmers View peripherals
 - PV/PVT transactor
- PL172 Multi-Port Memory Controller Case Study

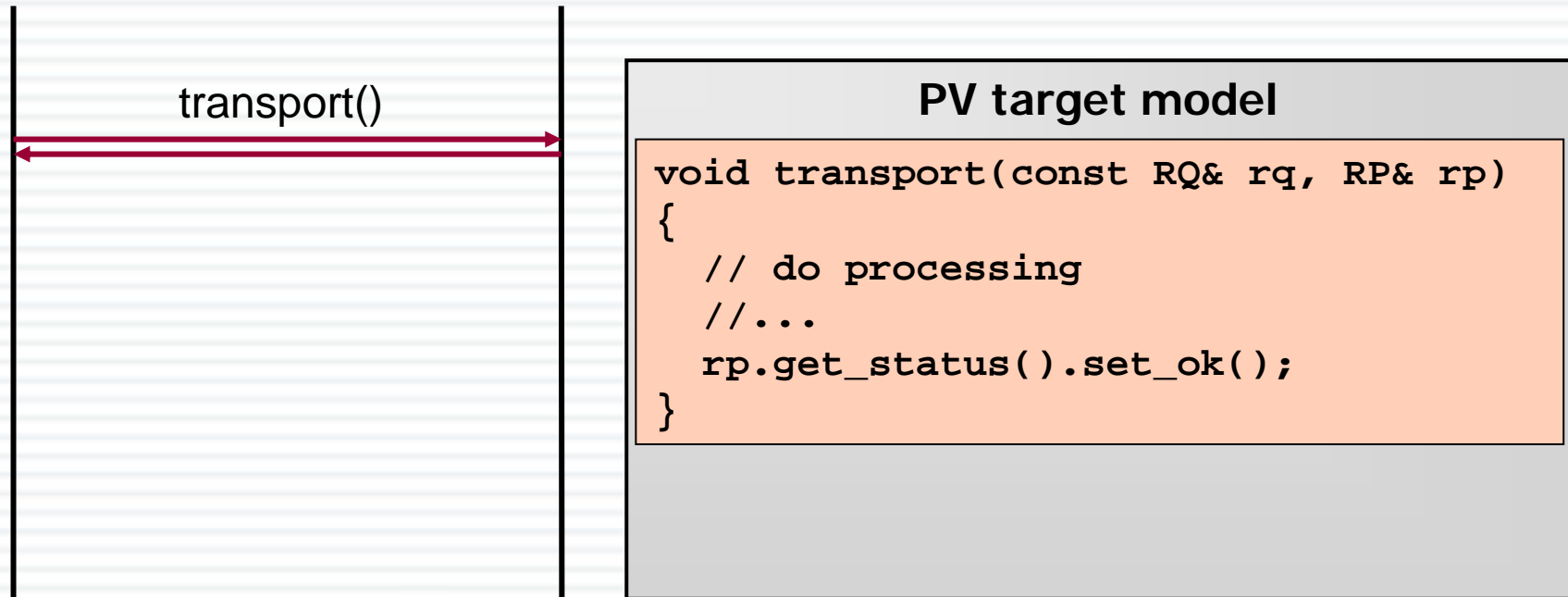
TLM2 Standard Interfaces

	Untimed
Bidirectional Blocking	<code>transport(const REQ&, RSP&)</code>

PV with Untimed model

PV Initiator

PV Target

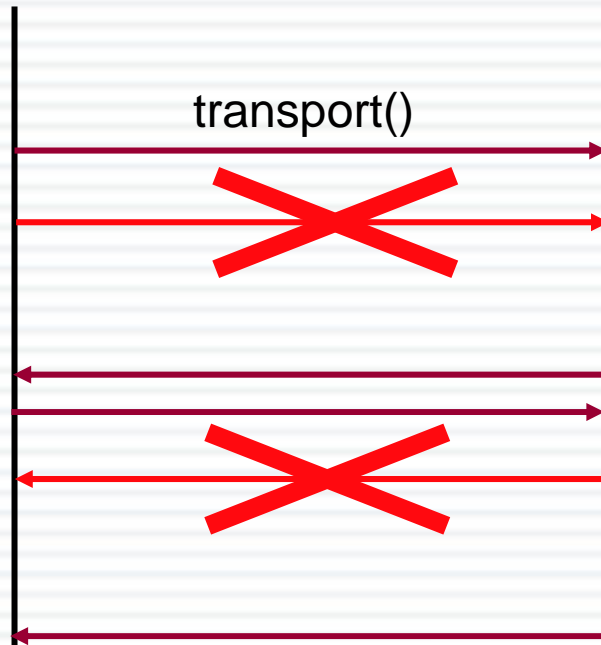


- Drawbacks:
 - No Timing modeled
 - Use-model limited (functional, SW)

PV with SystemC Explicit Timing

PV Initiator

PV Target



PV target model

```
void transport(const REQ& rq, RSP& rp)
{
    // do processing
    // ...
    unsigned int latency = ...;
    wait(latency * clk_period);
    rp.get_status().set_ok();
}
```

- Drawbacks:
 - Low simulation speed as wait() is called with every activation
 - Forces identical accept and response delays
 - Modeling of pipelining not possible

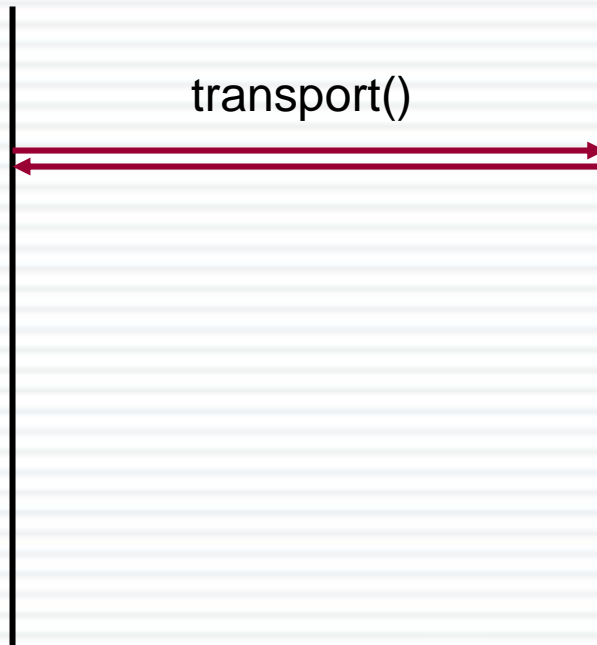
TLM2 Standard Interfaces

	Untimed	Timed
Bidirectional Blocking	<code>transport(const REQ&, RSP&)</code>	<code>transport(const REQ &, RSP &, sc_time &)</code>

PV with Timing Annotation

PV Initiator

PV Target



PV target model

```
void transport(const REQ& rq, RSP& rp
               sc_time& latency)
{
    // do processing ...

    double lat = ...;
    latency = lat * clk_period;

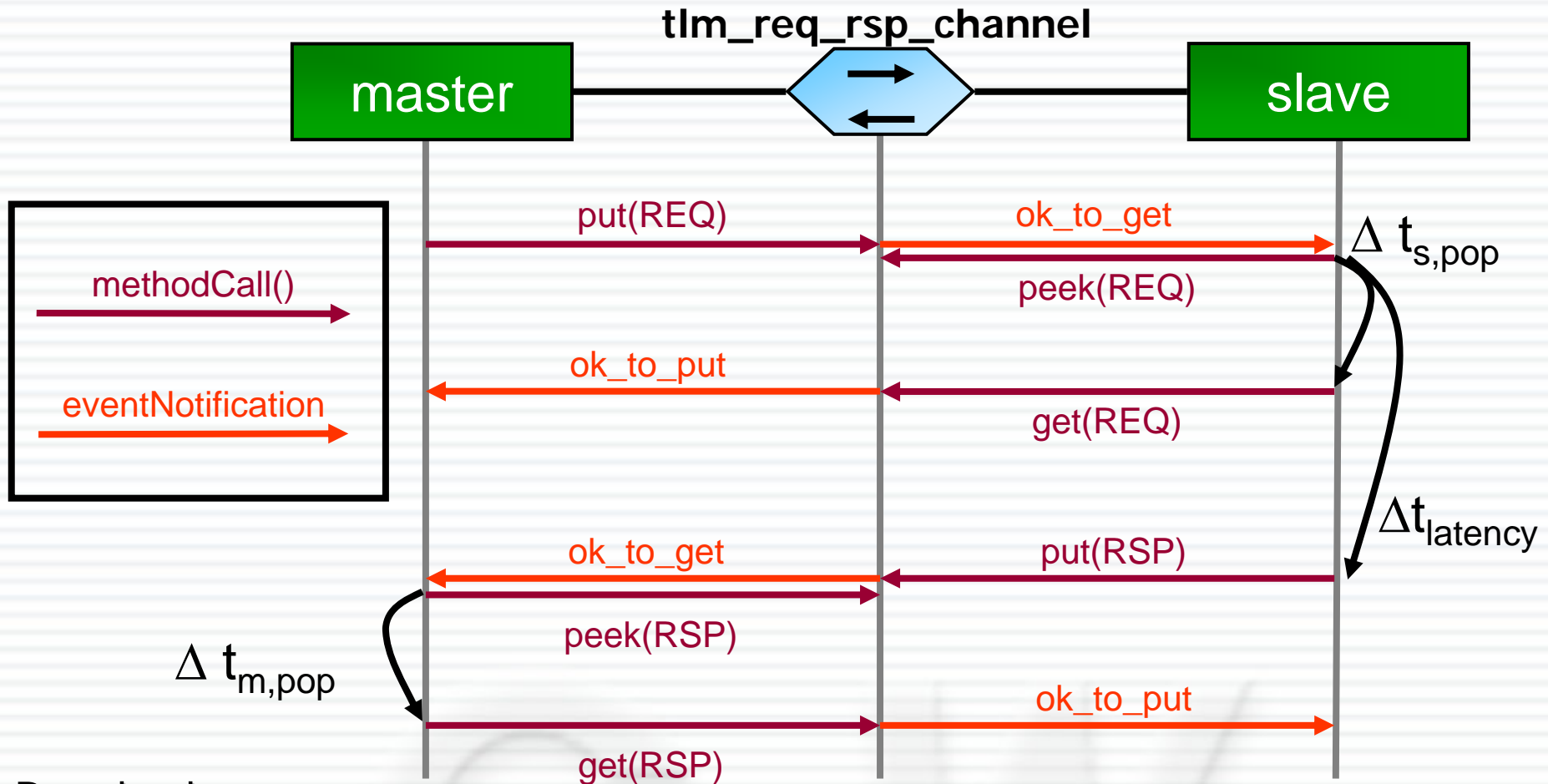
    rp.get_status().set_ok();
}
```

- Benefits:
 - Preserve potential for high simulation speed (not calling wait)
 - Flexibility to model different timing for accept and response delay
 - Defers realization of timing to initiator

TLM2 Standard Interfaces

	Untimed	Timed
Bidirectional Blocking	<code>transport(const REQ&, RSP&)</code>	<code>transport(const REQ &, RSP &, sc_time &)</code>
Unidirectional Blocking	<code>void put(const T&)</code> <code>void get(T&)</code> <code>void peek(T&)</code>	
Unidirectional Non-Blocking	<code>bool nb_put(const T&)</code> <code>bool nb_can_put()</code> <code>sc_event &ok_to_put()</code> <code>bool nb_get(T&)</code> <code>bool nb_can_get()</code> <code>sc_event &ok_to_get()</code> <code>bool nb_peek(T&)</code> <code>...</code>	

Un-timed Unidirectional Interfaces



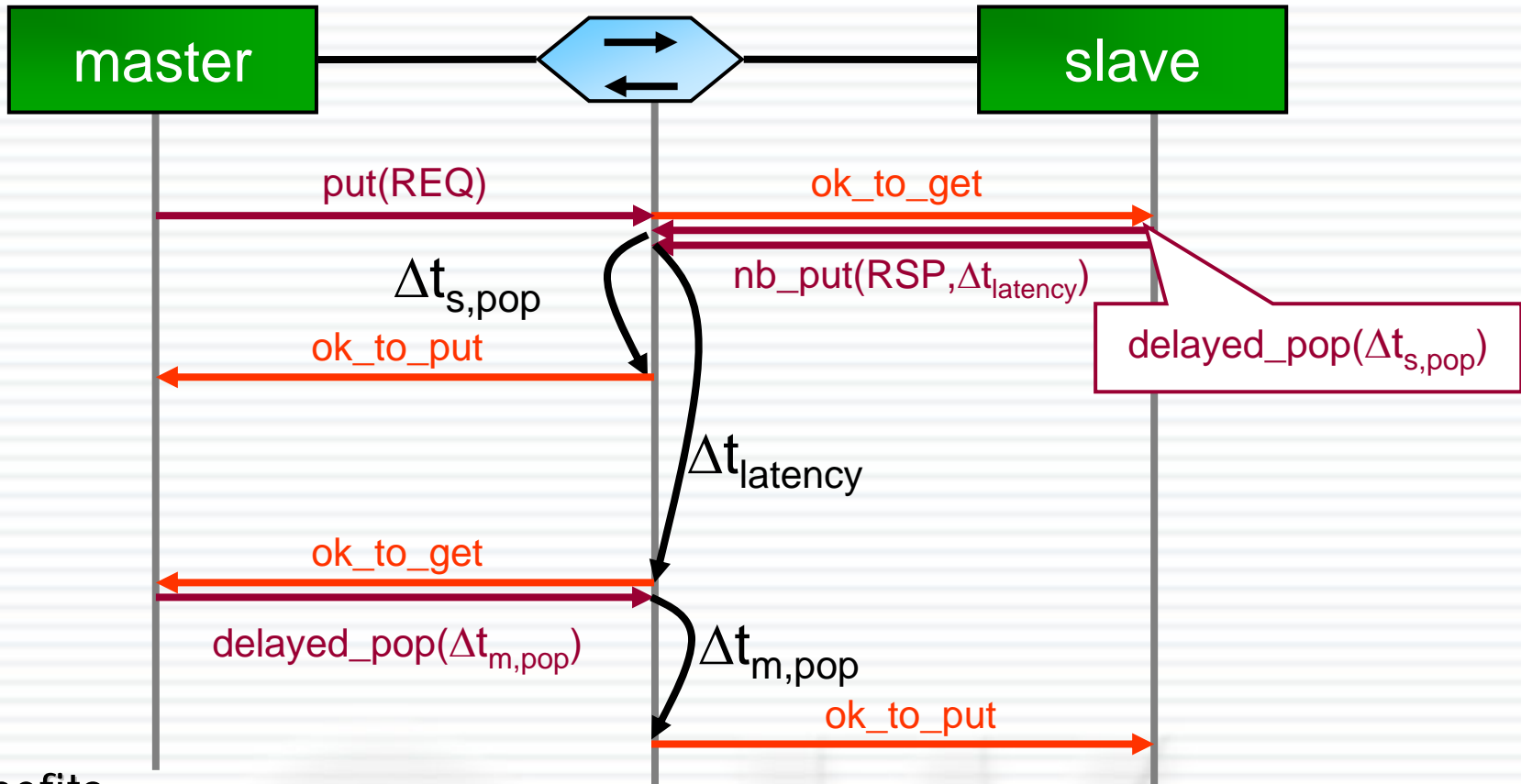
Drawbacks:

- Low modeling efficiency as timing needs to be handled explicitly in the model
- Typically low simulation speed as `wait()` is called with every activation

TLM2 Standard Interfaces

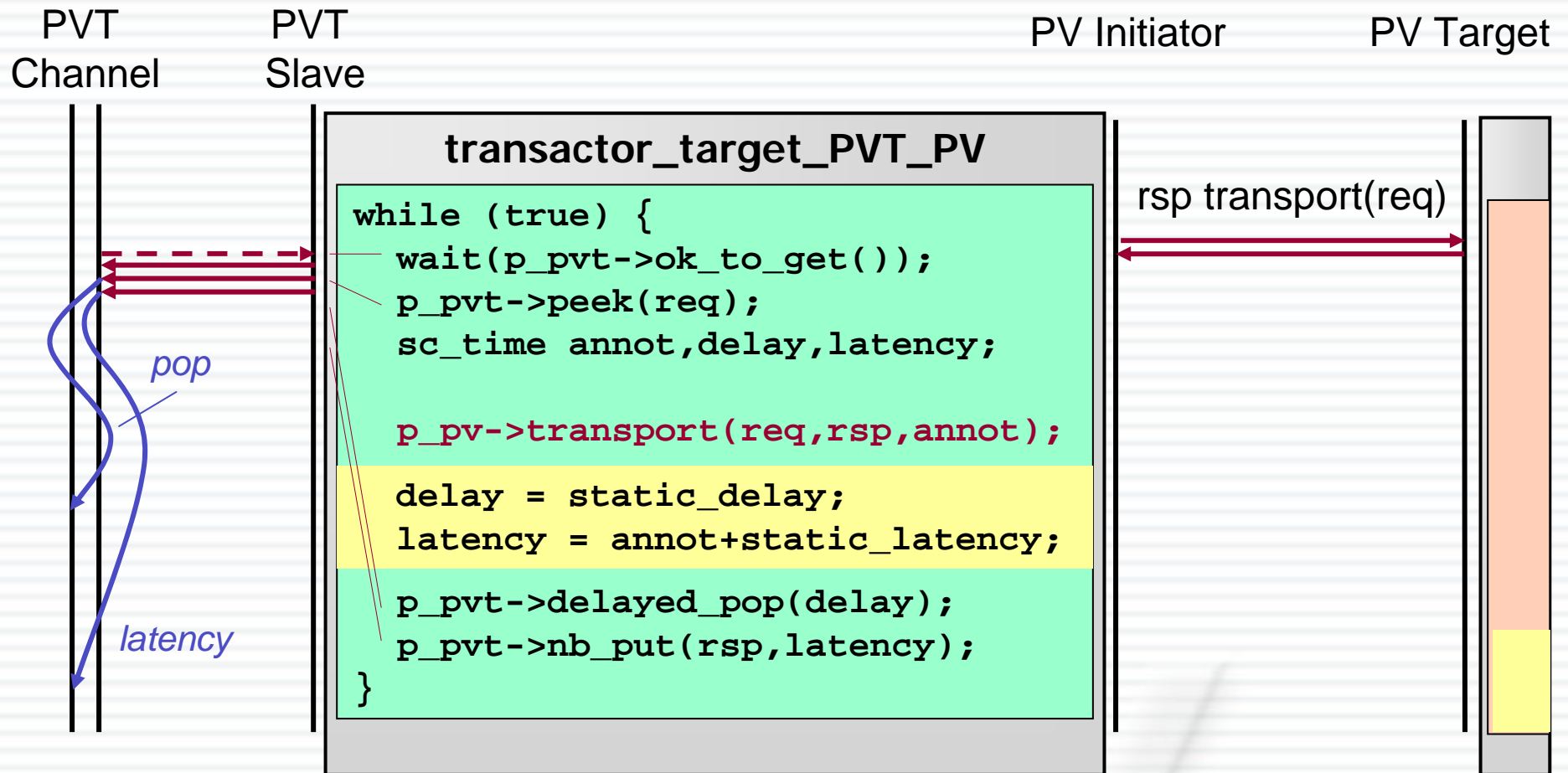
	Untimed	Timed
Bidirectional Blocking	<code>transport(const REQ&, RSP&)</code>	<code>transport(const REQ &, RSP &, sc_time &)</code>
Unidirectional Blocking	<code>void put(const T&) void get(T&) void peek(T&)</code>	
Unidirectional Non-Blocking	<code>bool nb_put(const T&) bool nb_can_put() sc_event &ok_to_put() bool nb_get(T&) bool nb_can_get() sc_event &ok_to_get() bool nb_peek(T&) ...</code>	<code>bool nb_put(const T&, sc_time &) bool nb_can_put(sc_time &) bool delayed_pop(sc_time &)</code>

Timed Unidirectional Interfaces



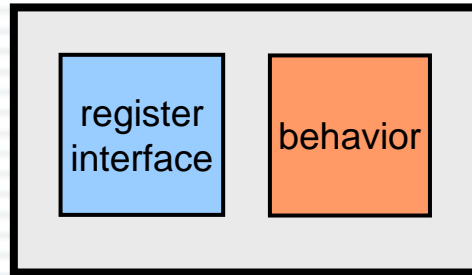
- Benefits:
 - Efficient modeling style:
 - functionality and delay computation handled in single activation
 - Potential for simulation speed optimizations in the tlm channel

PV/PVT Timing Conversion

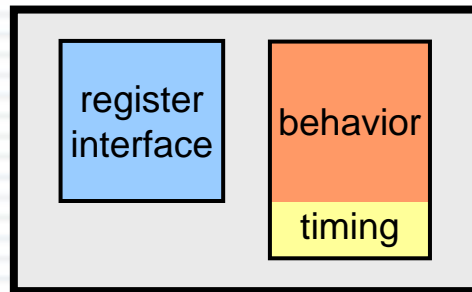
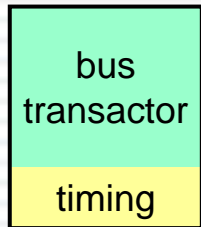


- Re-use PV peripherals for architectural modeling
- Transactor configuration determines selection of timing parameters
- Transactor defers realization of timing to PVT channel

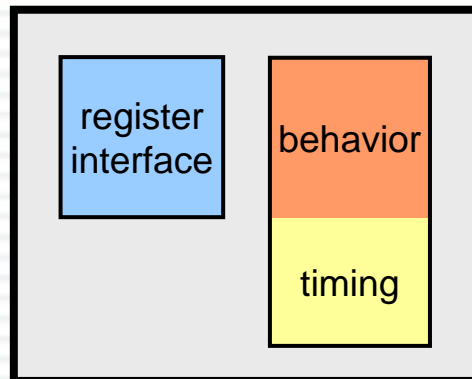
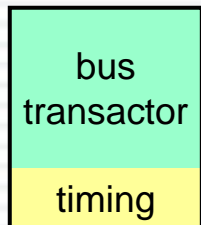
Modeling for Re-use



PV Peripheral = behavior
+ register accurate



PVT Peripheral = PV Peripheral
+ approximate timing
+ PVT transactor

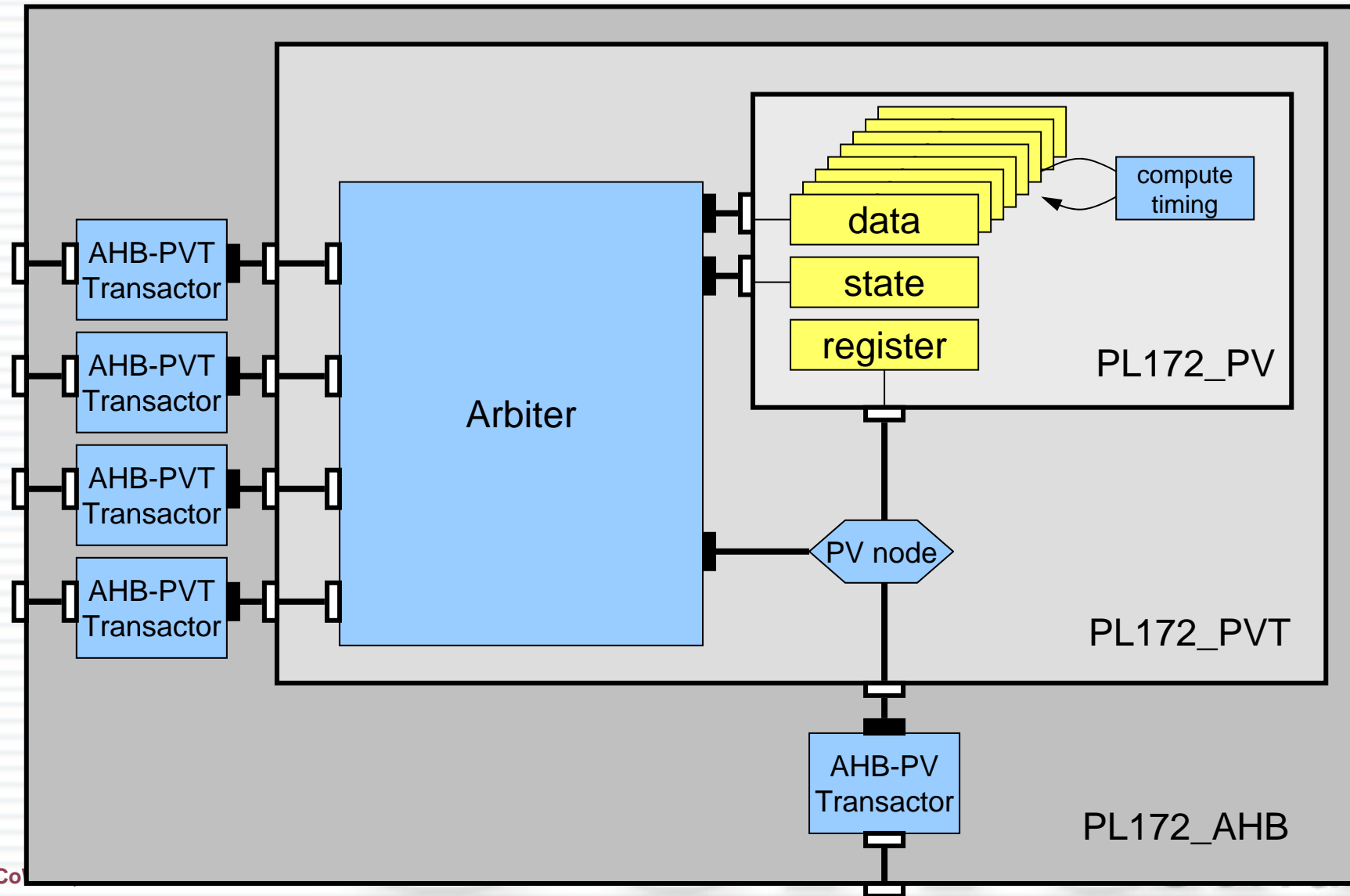


CA Peripheral = PV Peripheral
+ cycle-accurate timing
+ CA transactor

Ways to compute the delay

- Static
 - Constructor argument
 - Low accuracy, only suitable for simplistic timing models
 - Example: register read/write, fixed latency blocks
- Stochastic
 - Configurable distribution and moments
 - Medium accuracy, suitable for high level architecture exploration
 - Example: Cache miss probability
- Dynamic
 - Delay is computed for every individual transaction
 - Highest accuracy
 - Highest modeling effort
 - Example: memory controller, cache controller

Case study: PL172 AHB Memory Controller



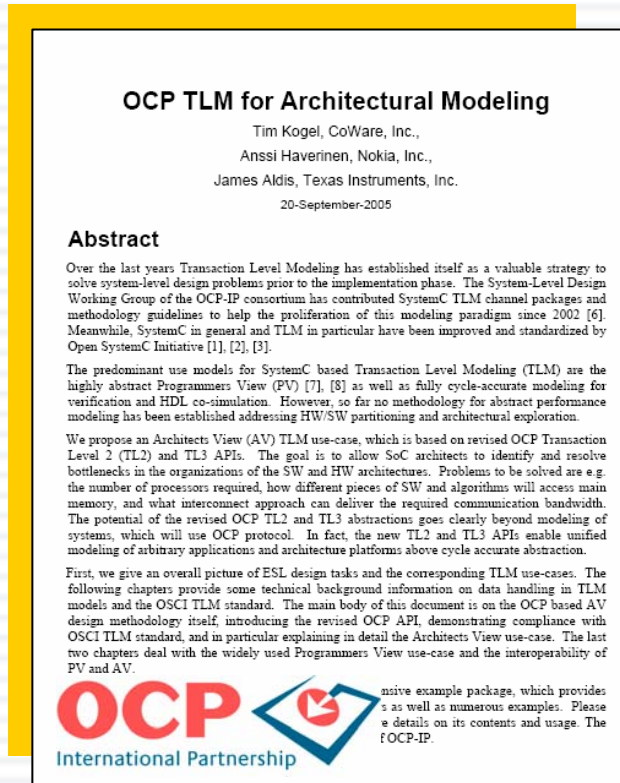
PL172 results

- Effort:
 - PV model: 1500 lines of code, 1 week
 - Timing model: 1500 lines of code, 2 weeks
 - Arbiter: 1000 lines of code, 1.5 weeks
- Accuracy
 - Cycle-by-cycle comparison against cycle-accurate reference
 - Relative average error: 0.53%
 - Positive and relative errors eliminate each other
 - Relative maximum error: 40%

For More Information:



www.coware.com



www.ocpip.org/socket/whitepapers

And of course the upcoming September release of TLM 2.0...