Peter Lüthi, Ulrich Hensel
AMD Dresden Design Center, Germany

Verification Glue: How to compose system-level environments

March 03, 2003
3rd European Specman User Group Conference

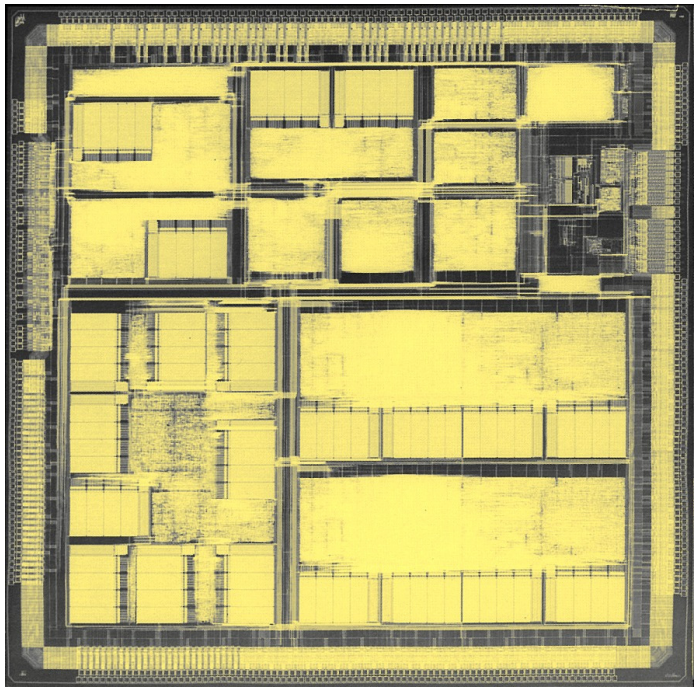
Presentation Outline

- Motivation
- Basic Ingredients
- Example: Checker composition
- Example: USB block-level setup
- Example: PCI system-level setup
- Example: Superposition of PCI and USB
- Summary

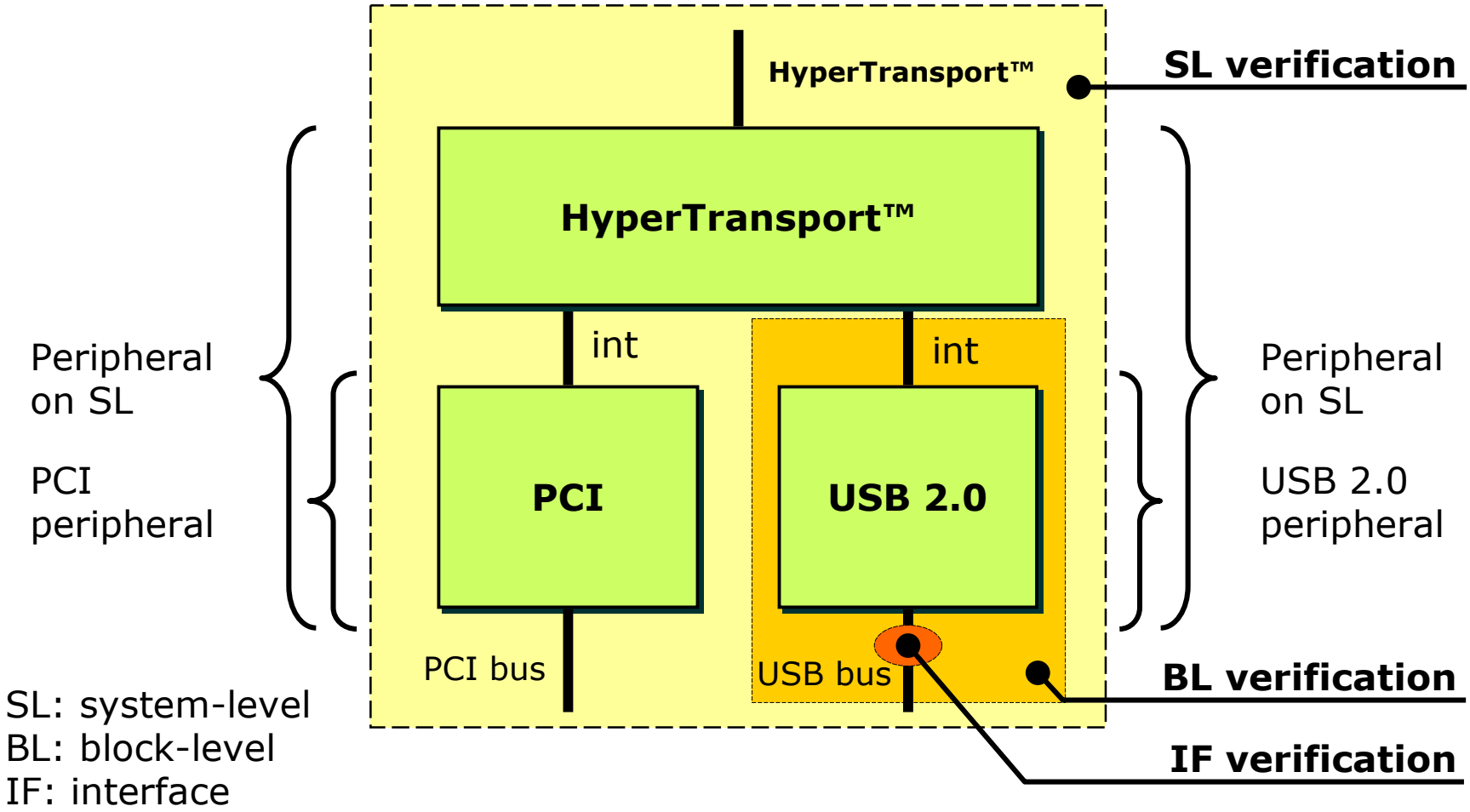
Motivation

Verification Dilemma

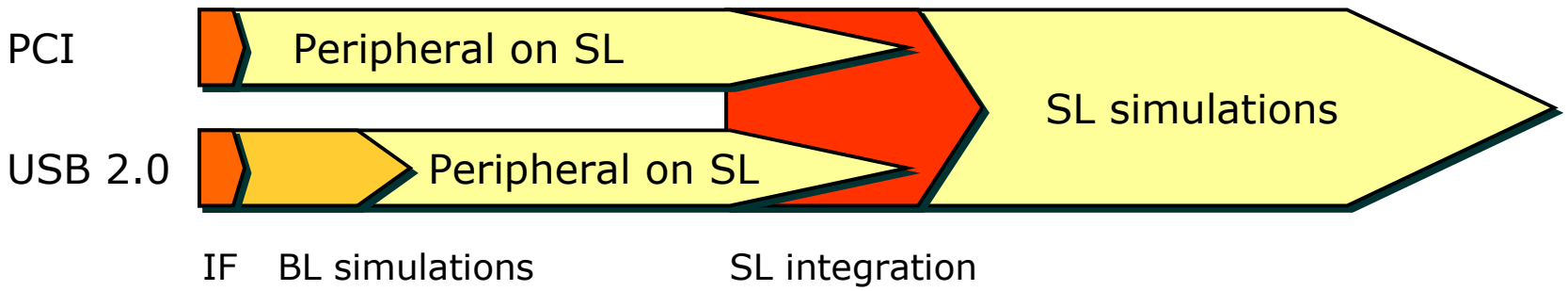
- Southbridge consists of various peripheral interfaces and bridges
- Block-level verification
  - High controllability and observability
  - Fast simulations and debugging
- System-level verification
  - Ensures correct integration
  - Complex system-level tests designed to ensure interoperability between different blocks
- Maximum reuse of verification components necessary to cope with widening design gap



Motivation cont.



Motivation cont.



Verification challenges

- Leverage of block level components to system level
- Seamless integration to overall chip environment including concurrent traffic to multiple devices
- Check system level access to shared resources (host memory, IRQs)

Reuse

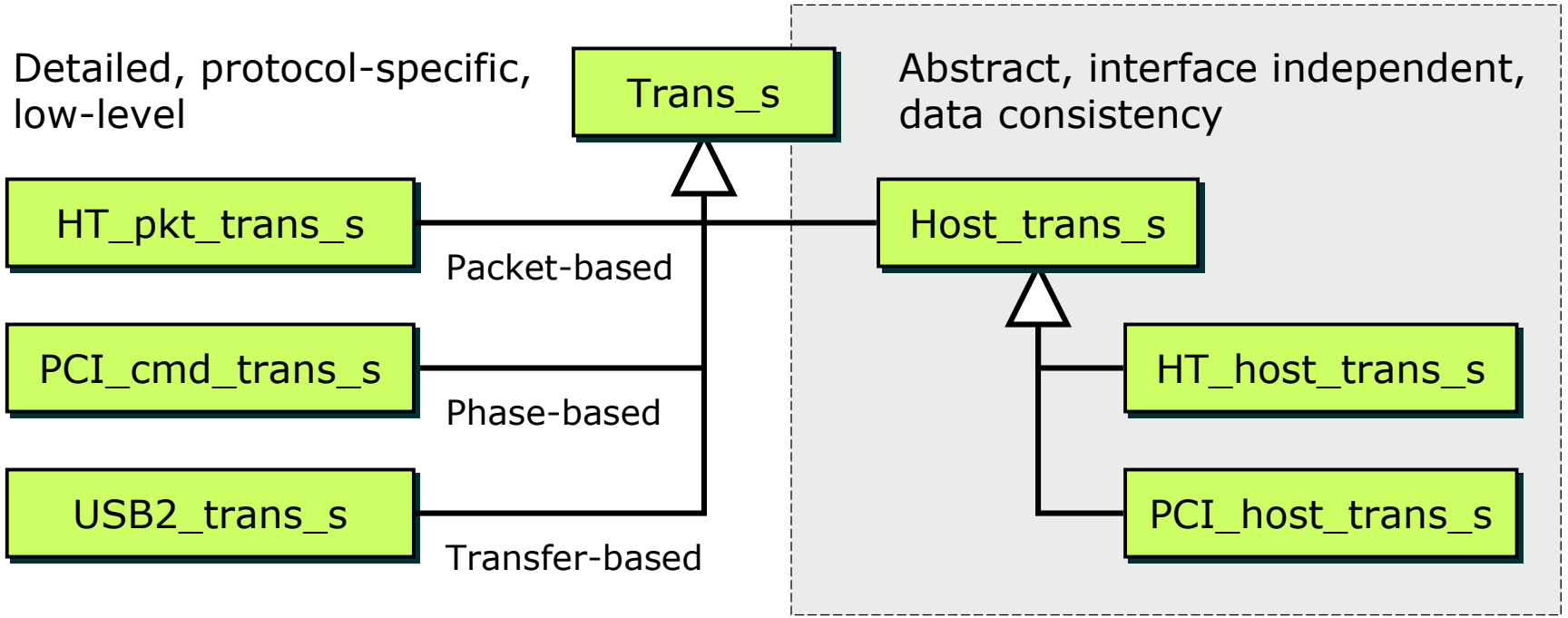
- BL generation, bus functional models (BFM), reference models
- Transaction and protocol checkers, coverage

SL: system-level
BL: block-level
IF: interface

Basic Ingredients: Transactions

AMD

- Common prototype *trans_s*, implemented to support split transactions (as superset of split and non-split transactions)
- Split transactions simplify transaction-based checking



Basic Ingredients: Transactions cont.

AMD

Code snippets

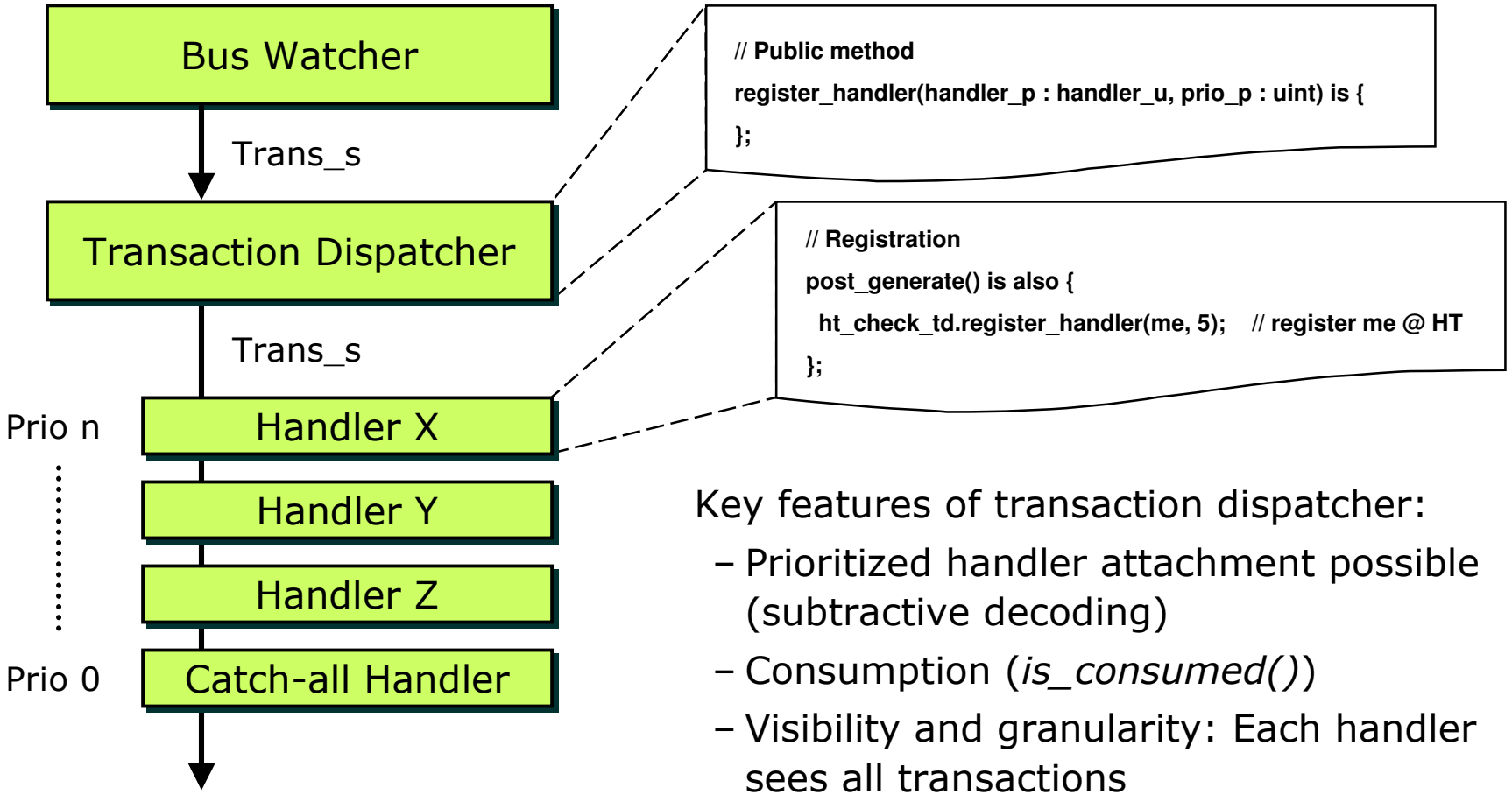
Trans_s

```
dir_m : dir_t;          // either tx = transmit or rx = receive
kind_m : kind_t;        // either req = request or resp = response


consume(trans_p : trans_s, handler_p : handler_u) is undefined;


is_consumed() : trans_s is undefined;     // returns NULL if
                                          // unconsumed
display() : string is {
  result = append(get_kind(), " ", get_dir(), " trans_s");
};
```

Host_trans_s

```
// public methods
get_typ() : cycle_t is {};  // transaction type: mem, io, conf
get_cmd() : cmd_t is {};  // transaction command: read, write
get_addr() : dword is {};  // 32 bit address
get_hi_addr() : dword is {};  // upper 32 bit (64 bit addressing)
get_be() : list of nibble is {};  // high active byte enables
get_term() : term_t is {};  // termination - only for responses
get_data() : list of dword is {};  // data
get_consumer() : handler_u is {};  // consumer
```

Basic Ingredients: Transaction Dispatcher



Key features of transaction dispatcher:

- Prioritized handler attachment possible (subtractive decoding)
- Consumption (*is_consumed()*)
- Visibility and granularity: Each handler sees all transactions

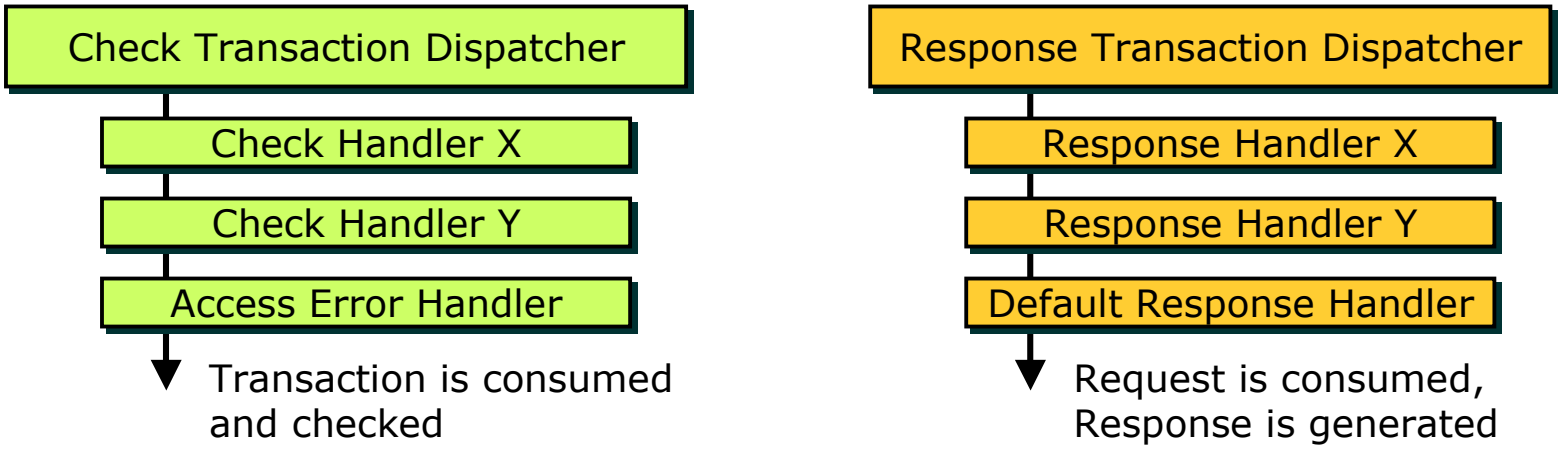
Basic Ingredients: Transaction Dispatcher cont.

```
unit check_trans_disp_u like trans_disp_u {
  exec_check() @ trigger_e is only {
    check_handler.run_check(trans_m);
  };
};
```

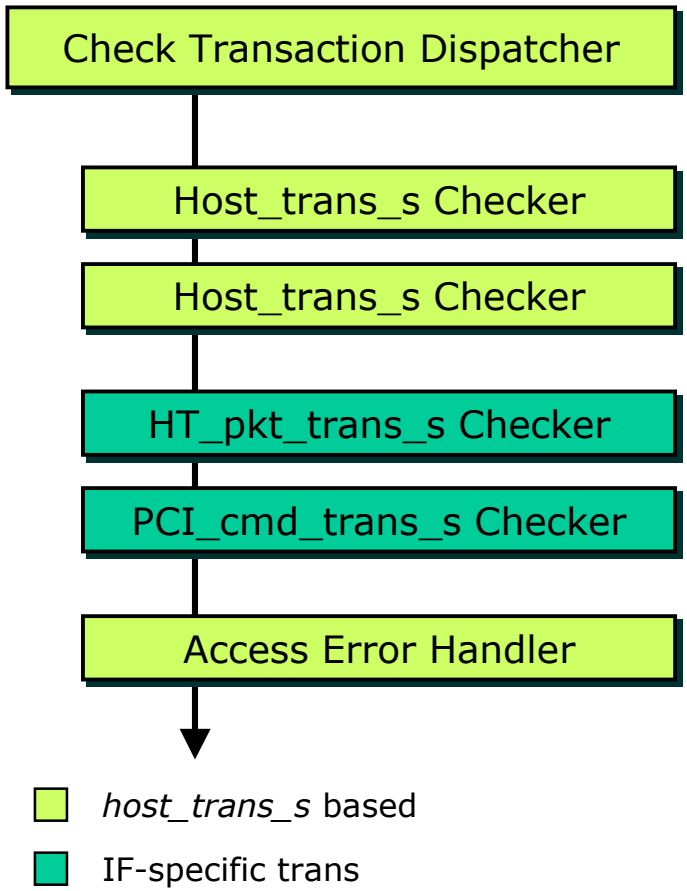
```
unit resp_trans_disp_u like trans_disp_u {
  exec_resp() @ trigger_e is only {
    resp_trans_m = respond_handler.respond(req_trans_m, resp_trans_m);
  };
};
```

```
unit my_check_handler_u like check_handler_u {
  run_check(trans_p : trans_s) @ trigger_e is
    only {
  };
};
```

```
unit my_respond_handler_u like respond_handler_u {
  respond(req_trans_p : trans_s, resp_trans_p : trans_s) : trans_s @ trigger_e
    is only {
      result = resp_trans_p;     // by default, pass incoming response further
  }; };
```



Example:
Composition of different checkers



Checker using *host_trans_s*:

- No interface-specific checks
- Reuse possible in both block- and system-level (e.g. USB2HT checker)

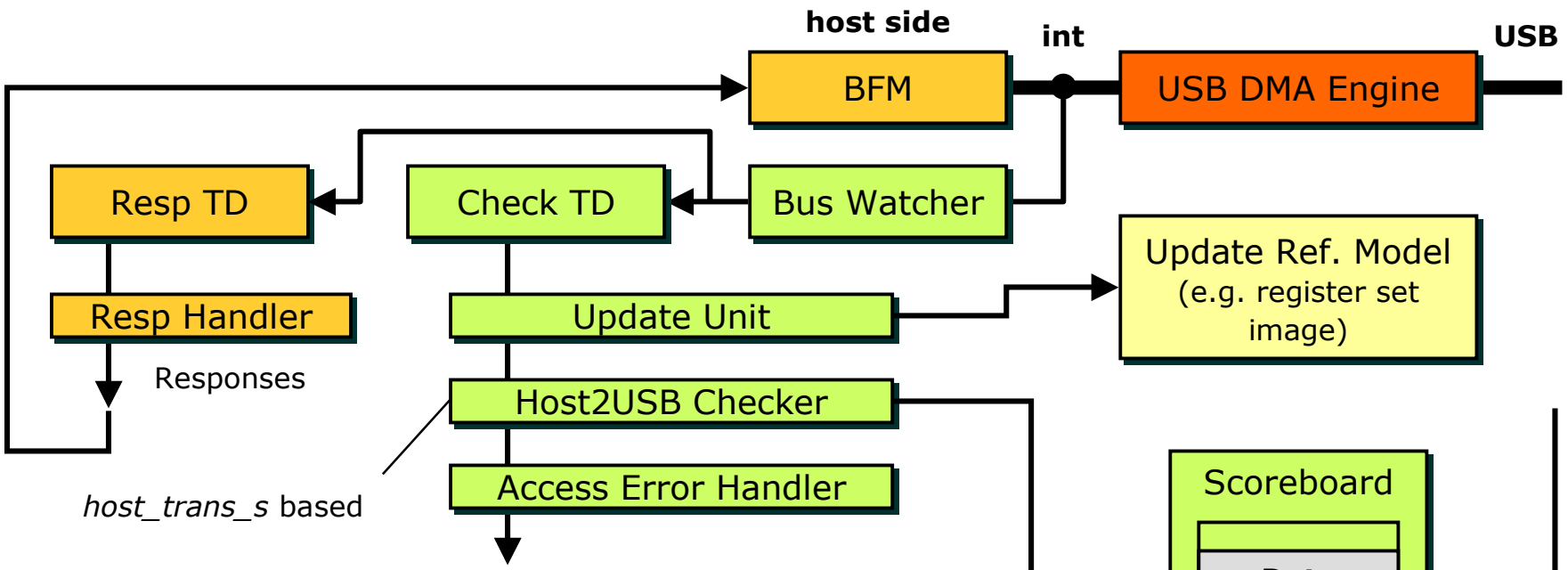
Checker using IF-specific trans:

- Interface-proprietary, allows more profound checks
- Checker using single point attachment can be reused in both block- and system-level (e.g. HT2HT checker)
- Checker using dual point attachment cannot be reused (e.g. HT2PCI checker)

Access error handler:

- Complains as soon as an unconsumed transaction arrives

Example:
USB block-level setup

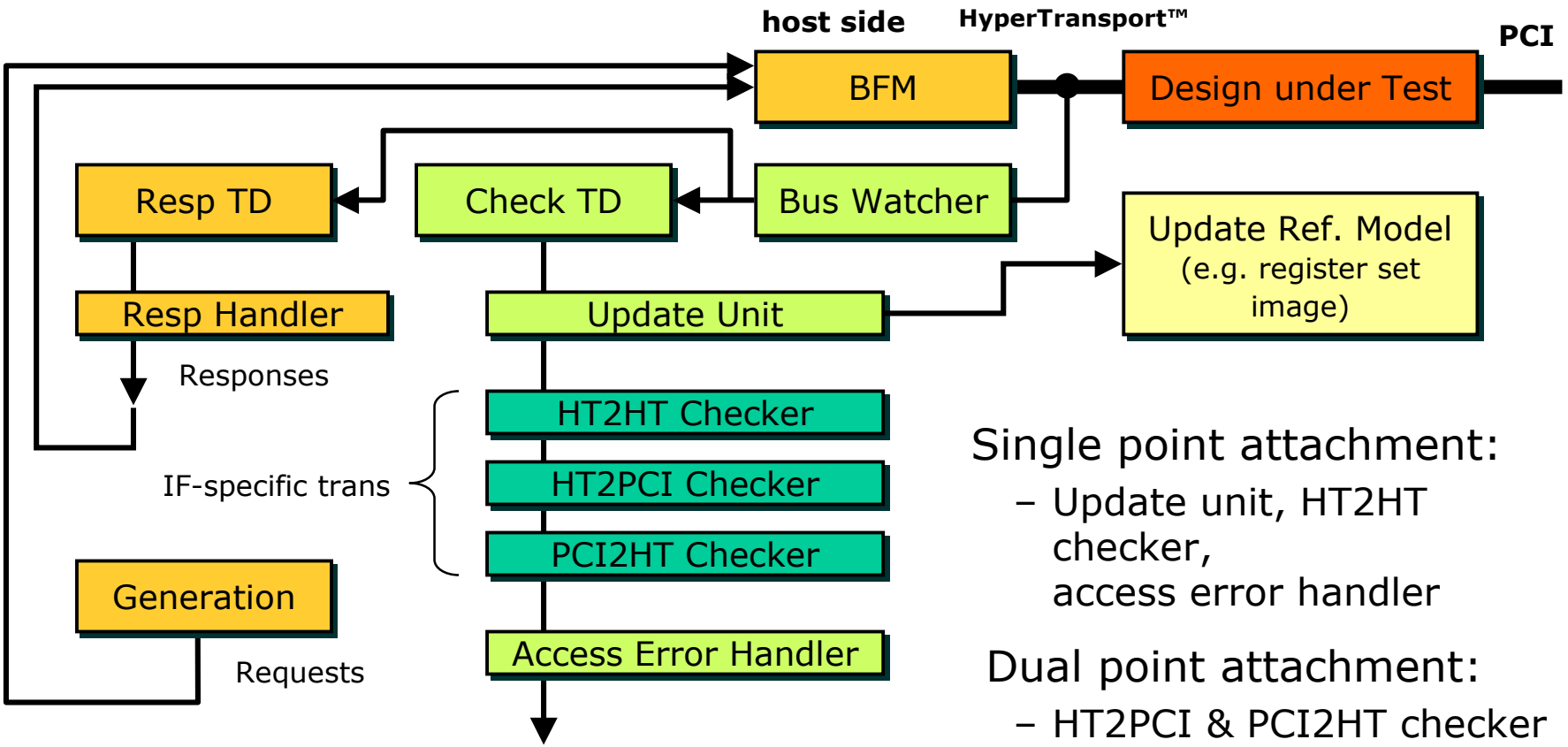


Update Reference Model:

- represents DUT-internal state for stimulation and check

Entire checking is based on producer-consumer concept and data consistency using scoreboard techniques.

Example:
PCI system-level setup



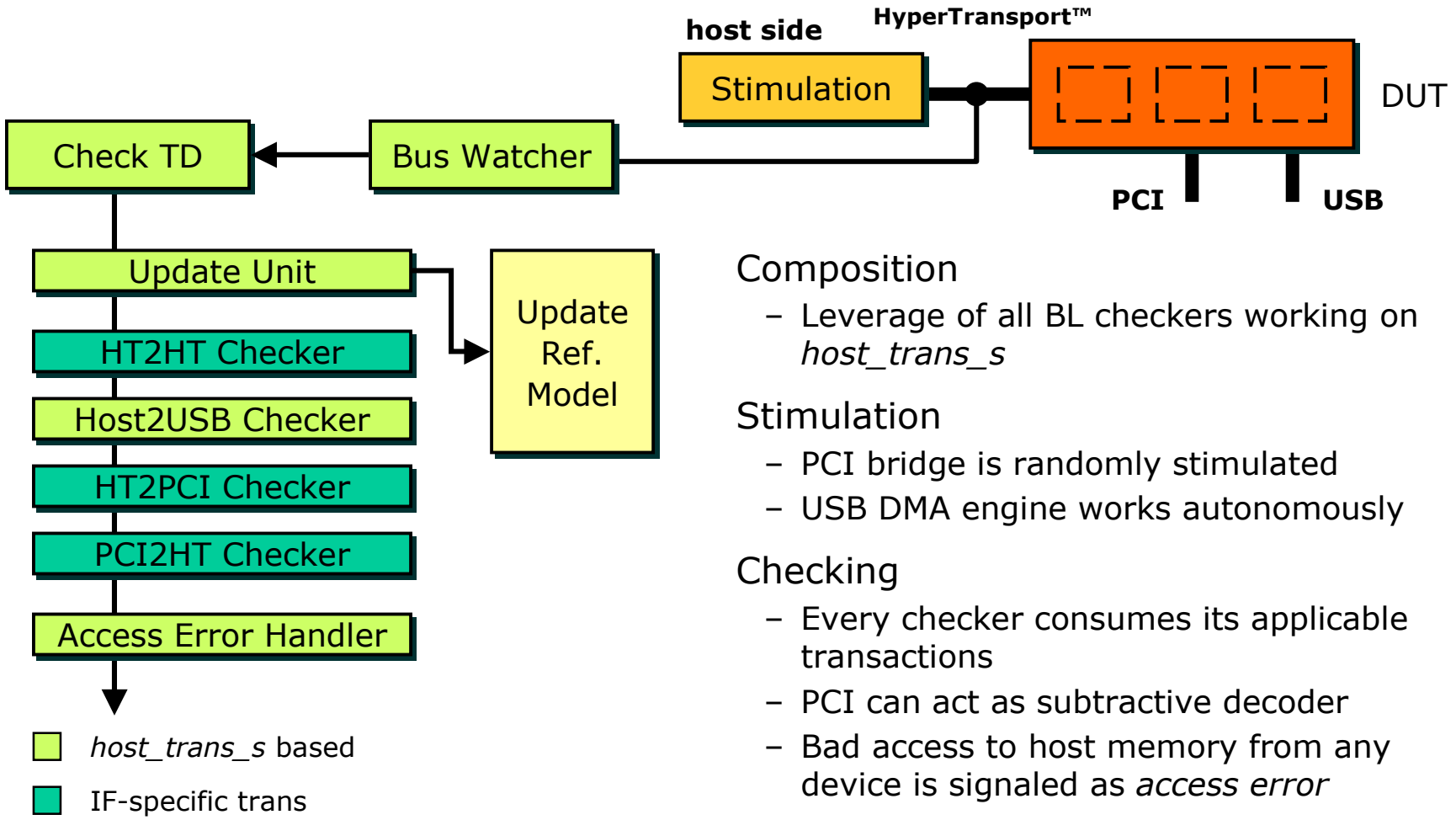
Single point attachment:

- Update unit, HT2HT checker,
  access error handler

Dual point attachment:

- HT2PCI & PCI2HT checker are connected to HT and PCI CTD

Superposition: PCI and USB on system-level



Composition

- Leverage of all BL checkers working on *host_trans_s*

Stimulation

- PCI bridge is randomly stimulated
- USB DMA engine works autonomously

Checking

- Every checker consumes its applicable transactions
- PCI can act as subtractive decoder
- Bad access to host memory from any device is signaled as *access error*

Summary

- Stimulation and checker components are independent of host bus specifics.
- Checking is completely independent from stimulus generation.
- Data flow and consistency check is distinguished from resource access check, i.e. the legality of memory access.
- Both generation and checking units can be composed uniformly to build a system level environment.