Testing on Target: Concepts and Experiences

Prof. Dr. Jan Peleska

Centre for Computing Technologies, University of Bremen, Germany

Dr. M. Oliver Möller

Verified Systems International GmbH, Bremen, Germany

IQNITE2010

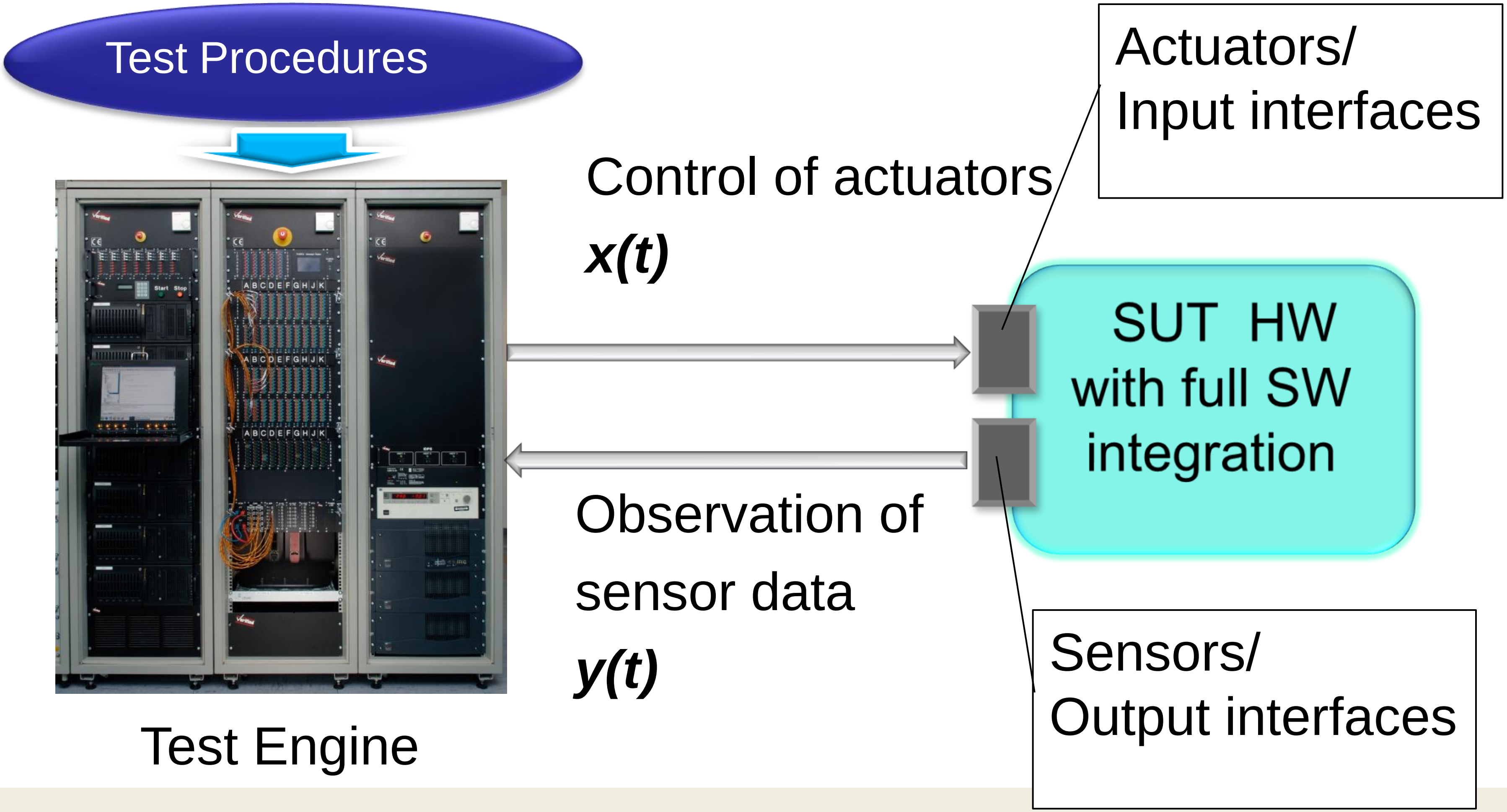
Overview

1. Motivation
2. Framework for testing on target
3. Test system requirements
4. Addressing technical problems
5. Experiences from 3 industrial projects
6. Conclusion

Motivation

- **HW/SW integration testing** with hardware-in-the-loop (HIL) technology:
  - Complete SW system is integrated on target HW
  - **Advantage:** system is tested in the same configuration that will become operational later on
  - **Disadvantage:** some properties are hard/expensive to test in the operational configuration
    - Example: SW reactions on HW faults

HW/SW integration testing



Test Procedures

Control of actuators

x(t)

Actuators/
Input interfaces

SUT HW
with full SW
integration

Observation of
sensor data

y(t)

Sensors/
Output interfaces

Test Engine

Motivation

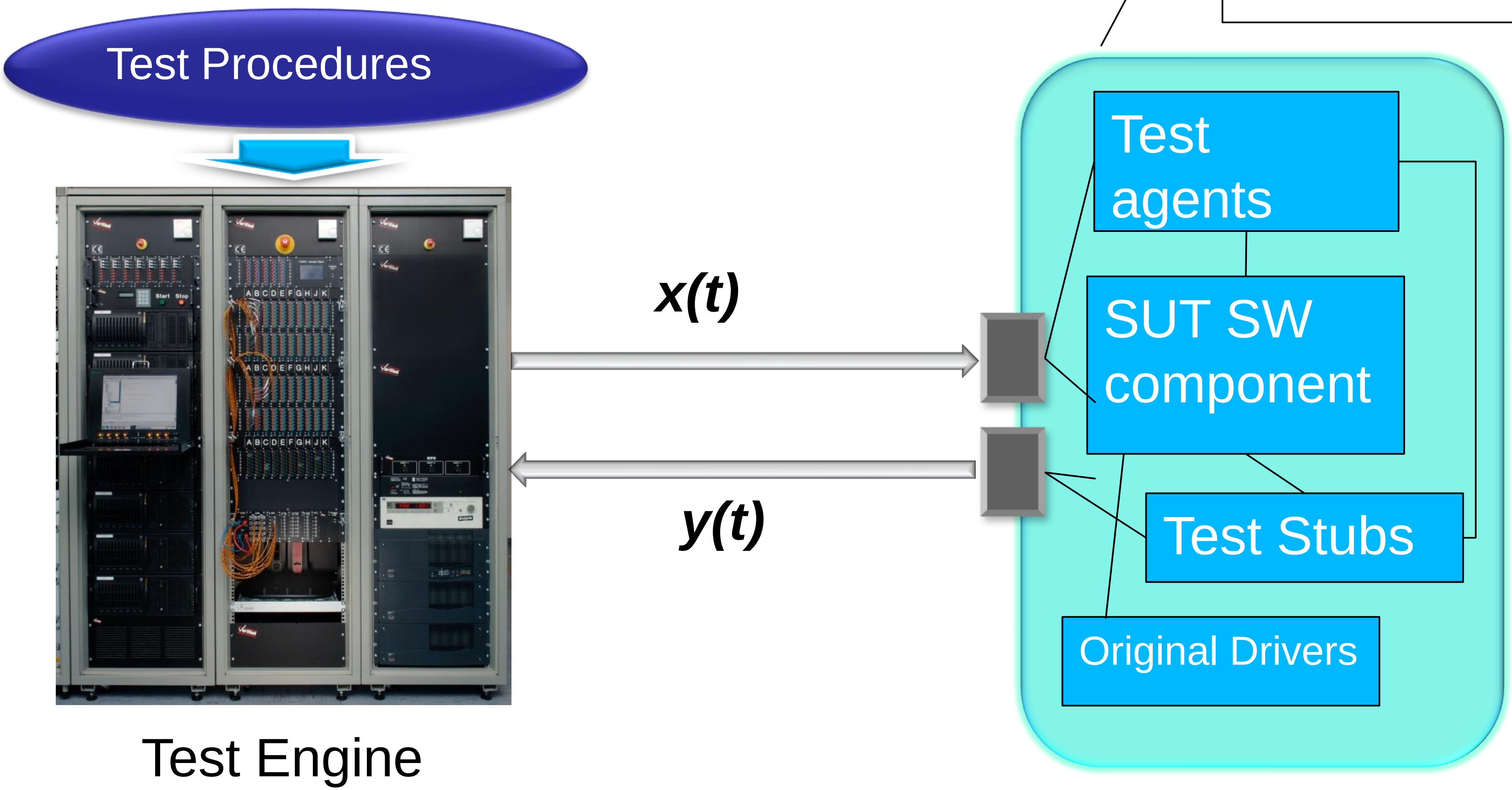
- **SW integration testing** with software-in-the-loop (SIL) technology on host computers:
  - SW components or complete SW system are tested on host computer – testing environment simulates HW behaviour and operational environment
  - **Advantage:** all SW properties can be easily stimulated
  - **Disadvantage:** No proof of proper HW/SW integration on the target HW

Motivation

- These considerations motivate **SW-integration testing on target HW (SWI-on-target testing):**
  - System under test (SUT) components are executed on target HW
  - A portion of the testing environment is deployed on the target HW and may
    - Stimulate SUT components
    - Replace/simulate drivers and HW where specific responses from the environment are required
  - Complex simulations and checks are deployed on host computer (test engine)

SWI-on-target testing

SUT HW
with **partial** SW
integration



Framework for testing on target

Required **capabilities for SWI-on-target testing**:

- Explicit **SUT function calls**
  - Example: test of library or driver functions
- Definition and activation of **complex scenarios** to be executed on the target
  - Example: Simulation of load scenarios on target
- Replace SUT functions by **stubs** in order to simulate different behaviours
  - Example: Stub function simulates driver response in a HW fault situation

Framework for testing on target

- Enable **access to HW interfaces**
    - Example: Test of SUT driver software by stimulating/monitoring SUT HW interfaces
- Enable **glass-box view** on the execution of SUT components on target HW
    - Example: Function calls and actual parameter values
- Enable access to all **test support functions** which are available in a SIL test on host computer
    - Example: code coverage capture, test documentation, test oracle calculation

Building block: remote function calls

- Example: test of function

  ```
  t0 f(t1 x1,...,tn xn)
  ```

- Host side (test engine) runs test procedure where call to `y = f(x1,...,xn)` is performed as if locally available

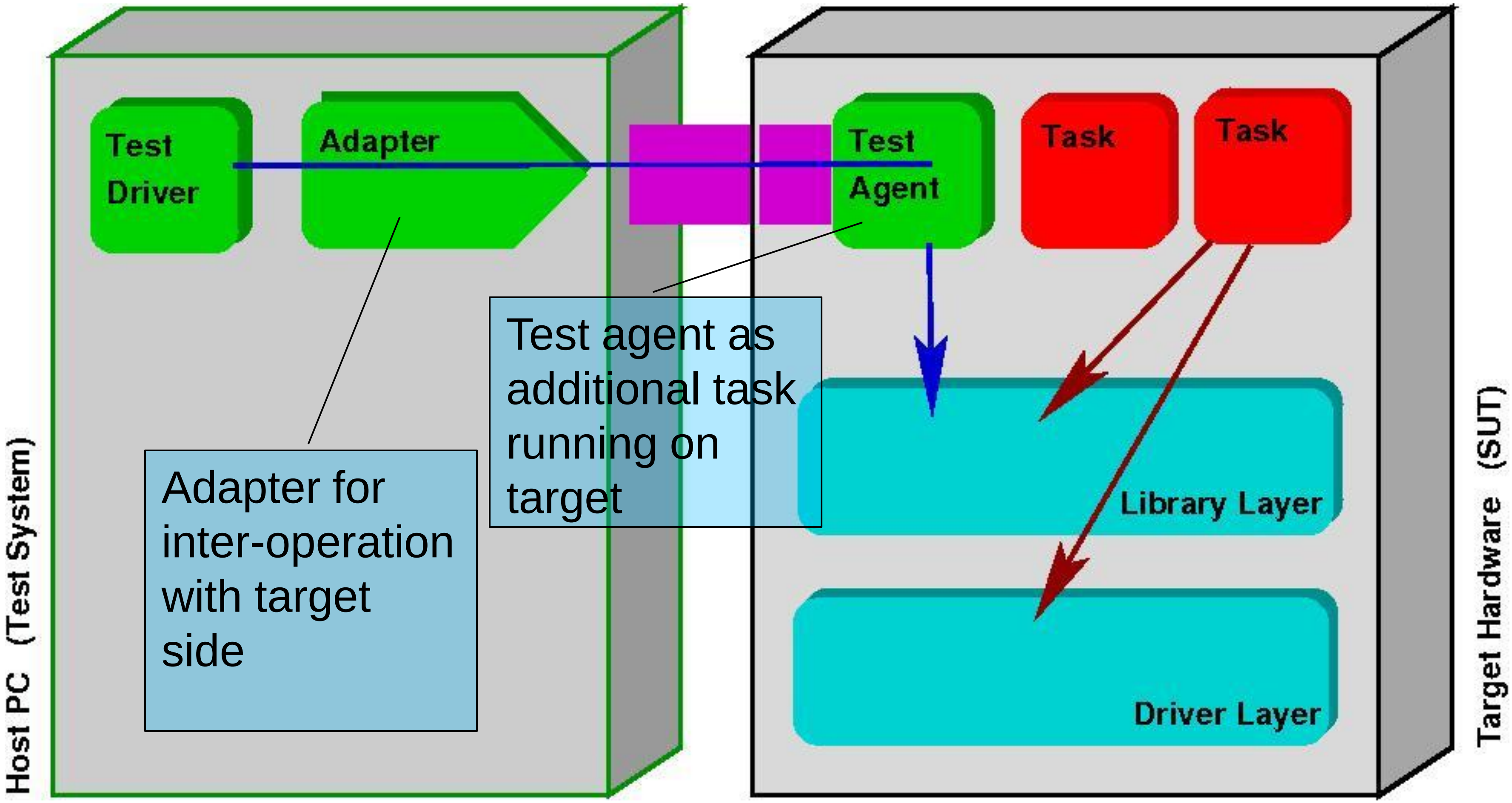
- Host side call sends request

  “Call `y = f(x1,...,xn)`”

  to test agent on target, together with actual parameter values x1,...,xn

- Test agent on target receives request, calls SUT function `f()` and returns return value and out-parameter values to test engine.

Remote function calls



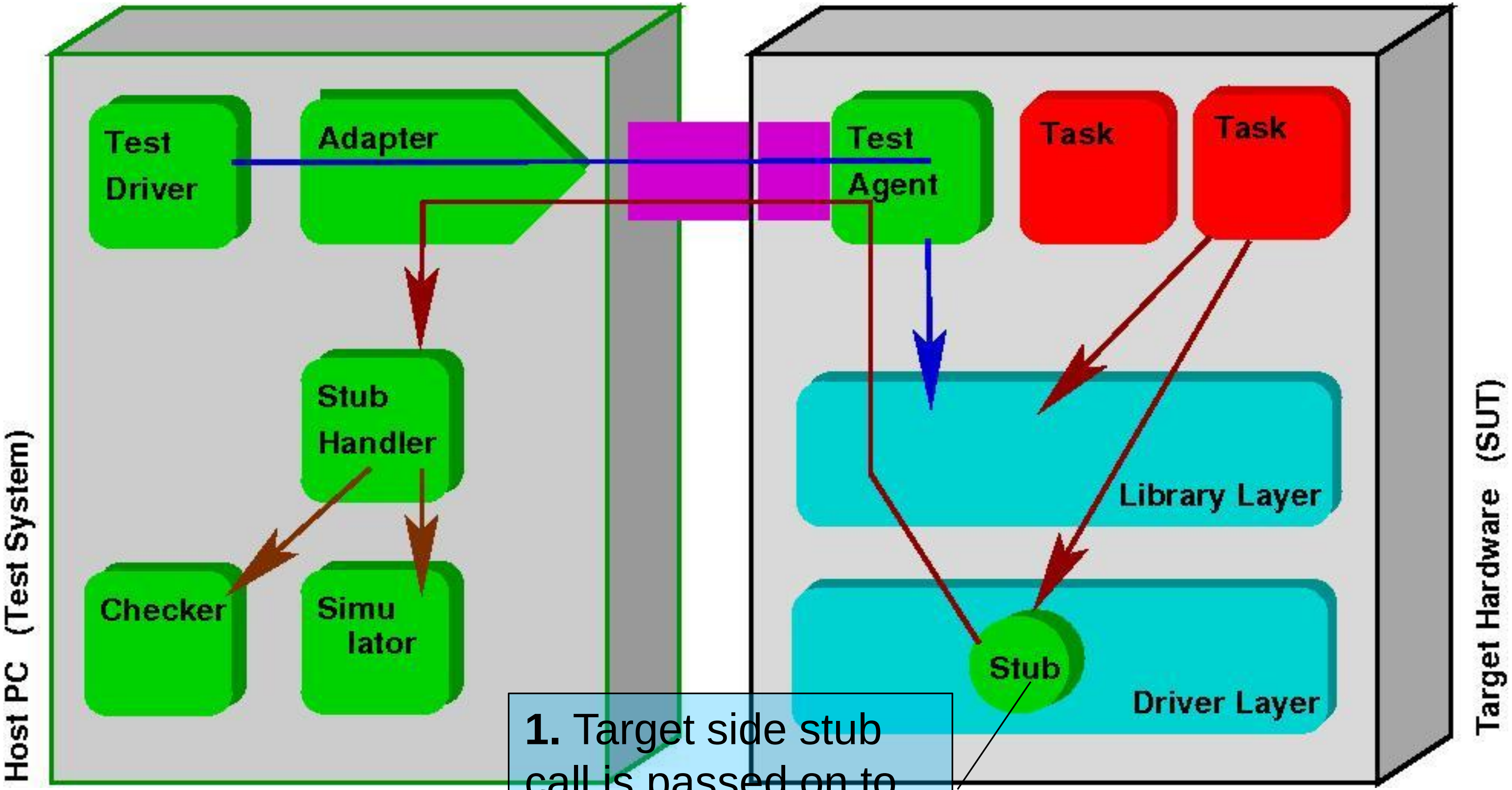
Test agent as additional task running on target

Adapter for inter-operation with target side

Building block: stubbing SUT functions on target

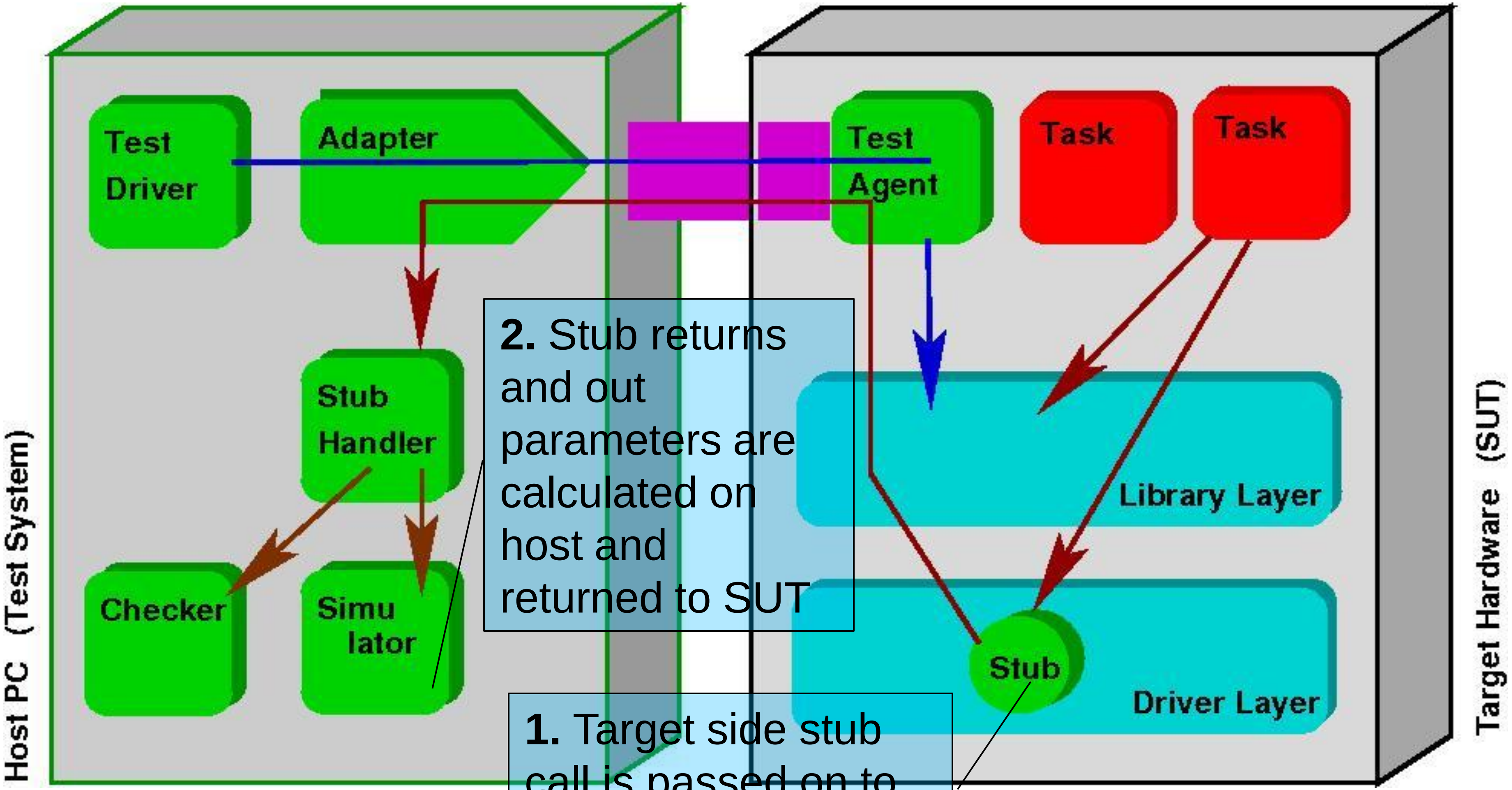
- **Stubbing:**
  - Replacement of SUT function by test environment function with identical interface
  - Test environment controls stub behaviour
- Stubbed function behaviour
  - is **handled on host side** (dynamically) and passes computation results back to target
  - can be used for **fault injection**
  - can be used for **checking** call parameters
  - use (cheap) host side mechanisms for **logging, check, simulation**

Stubbing SUT functions on target



1. Target side stub call is passed on to host (test engine)

Stubbing SUT functions on target



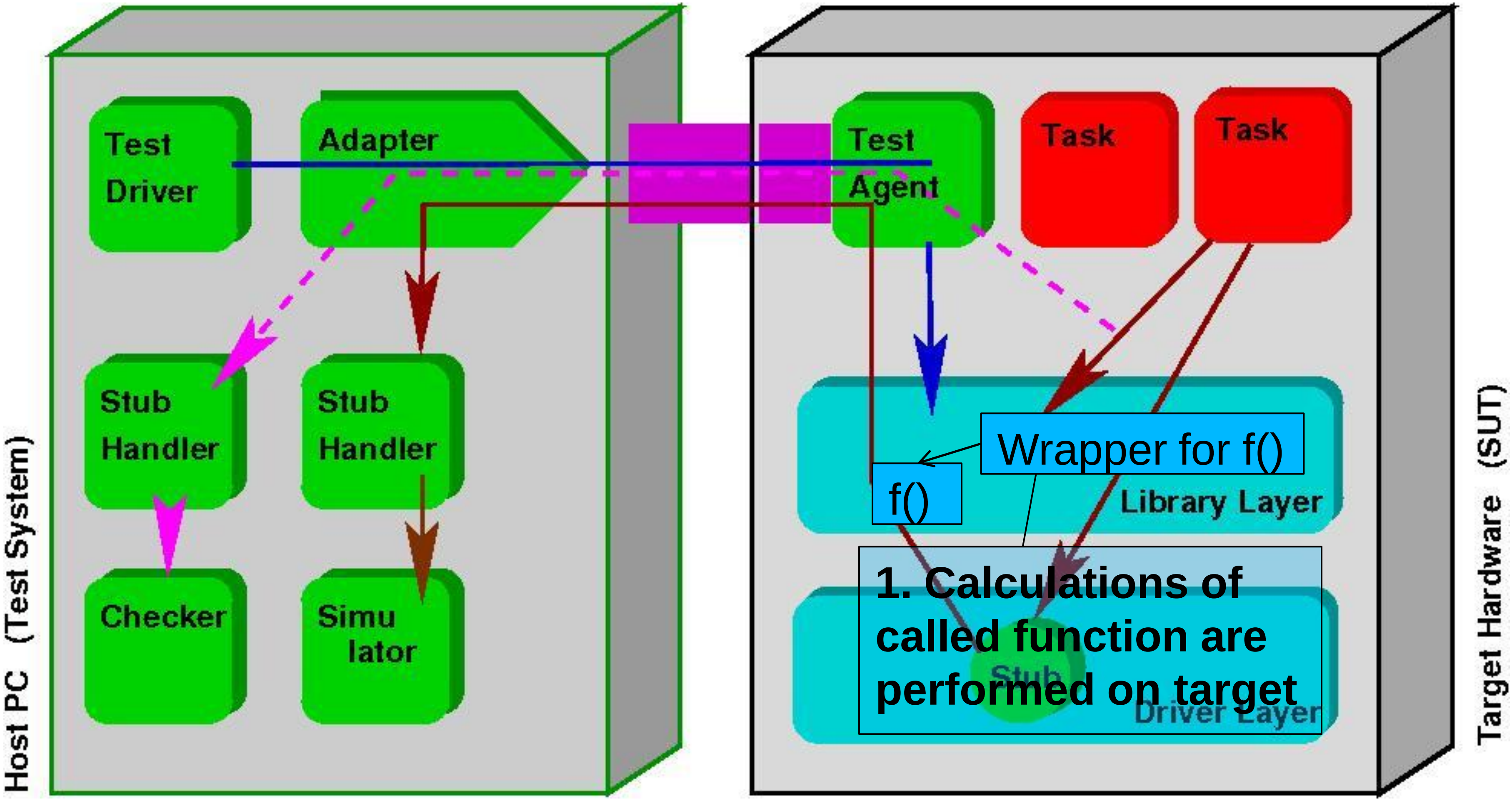
2. Stub returns and out parameters are calculated on host and returned to SUT

1. Target side stub call is passed on to host (test engine)

Building block: observing SUT functions on target

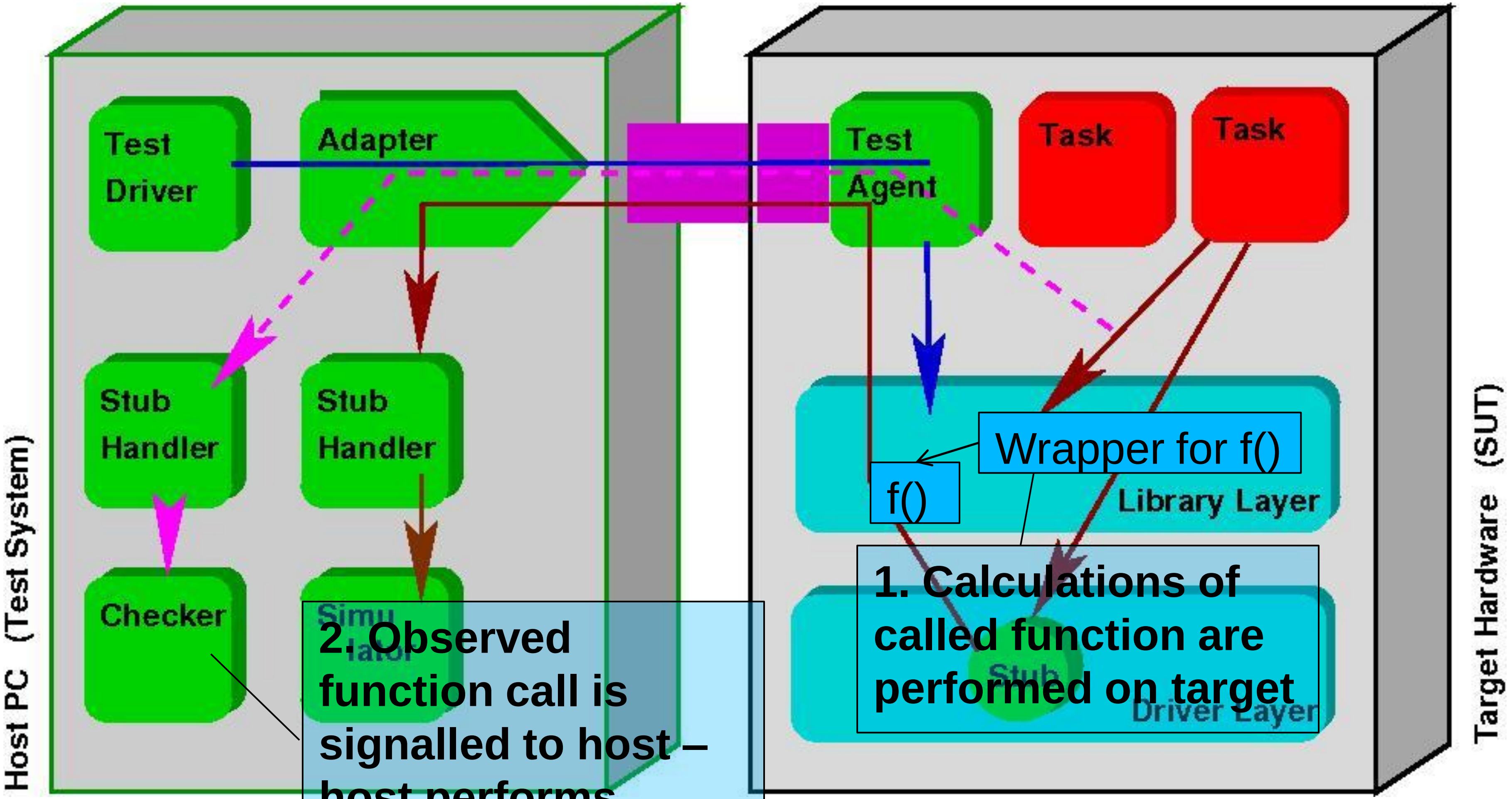
- Similar to stubbing, but without changing original function behaviour:
  - Stub acts as **wrapper** around original function to be called
  - Inputs, return values and out-parameter values are sent by wrapper stub from SUT to host
  - Observed **function calls are captured** by adapter on host-side
  - **Checking** of these data is performed in test procedure running **on the host**

Observing function calls



1. Calculations of called function are performed on target

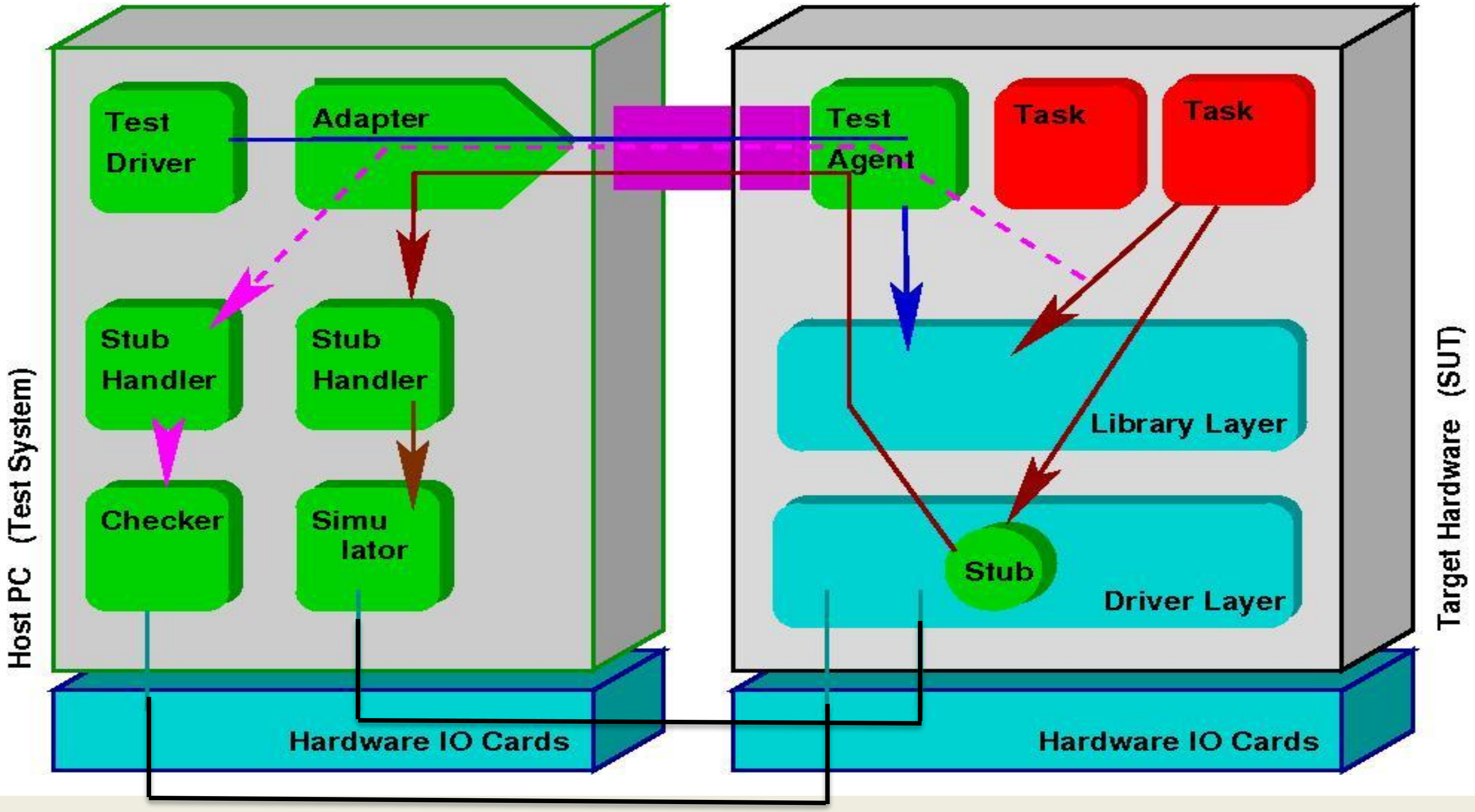
Observing function calls



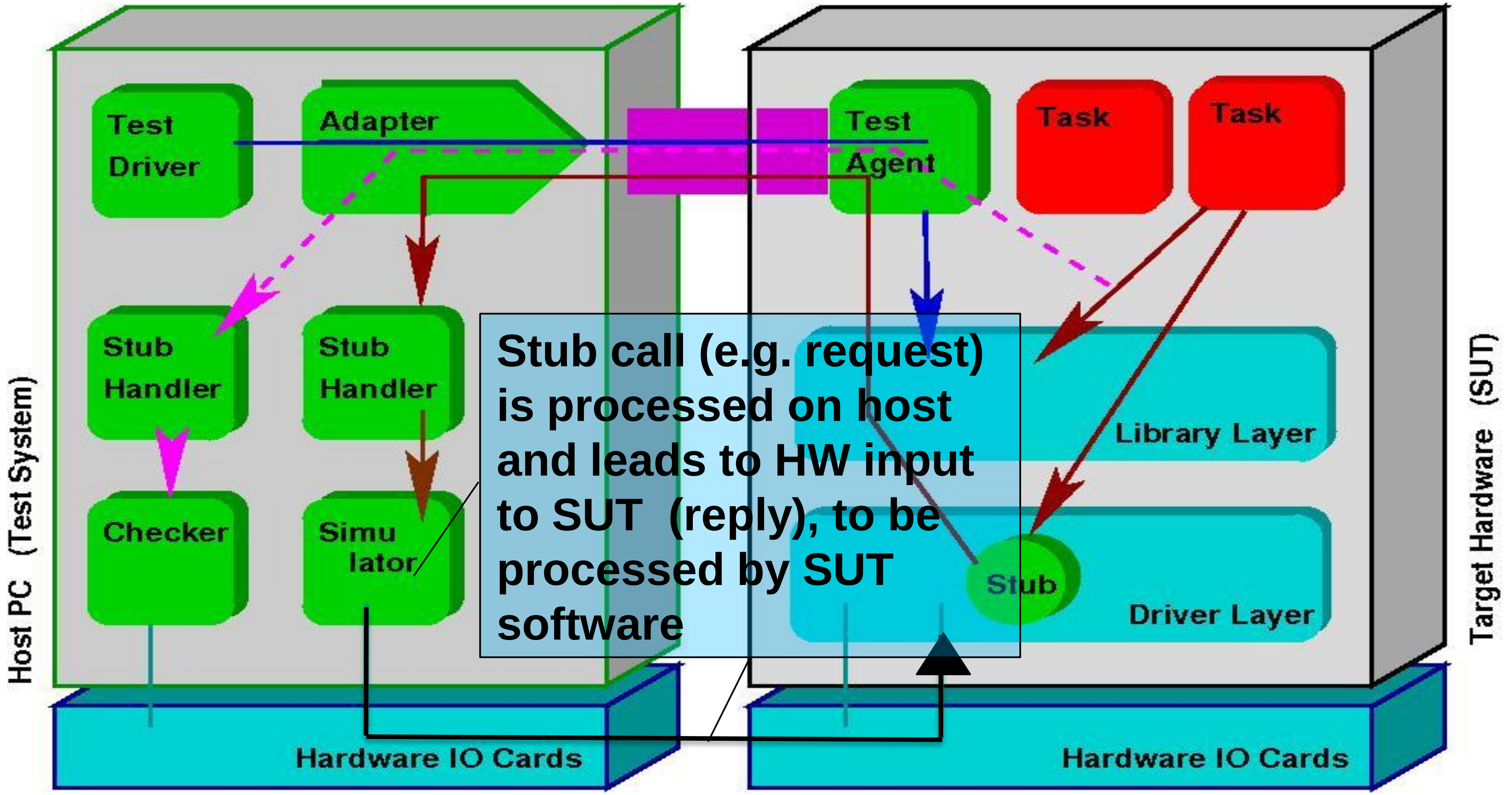
1. Calculations of called function are performed on target

2. Observed function call is signalled to host – host performs parameter checks

Adding Hardware I/O as part of the testing environment

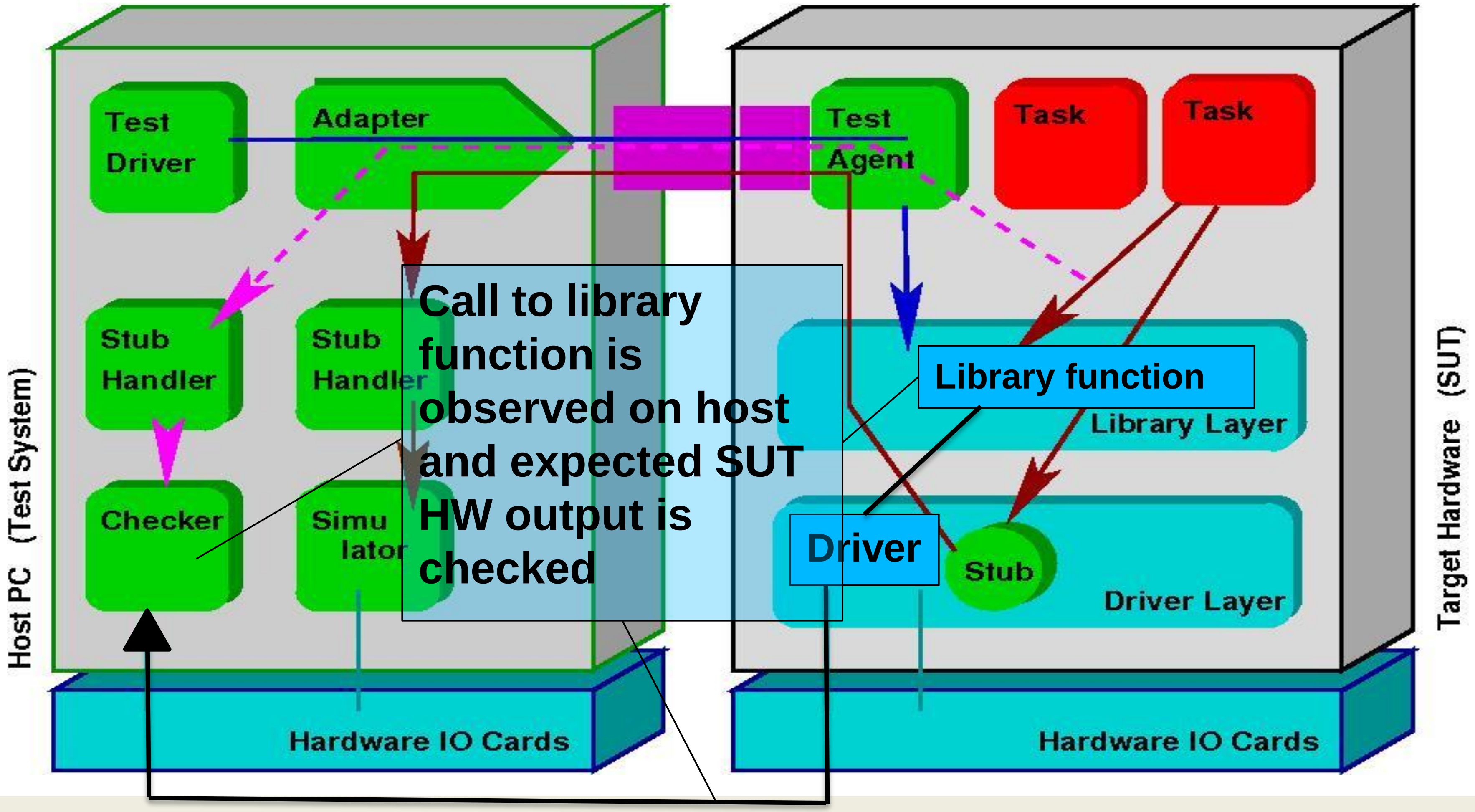


Adding Hardware I/O: stubbing with HW I/O



Stub call (e.g. request) is processed on host and leads to HW input to SUT (reply), to be processed by SUT software

Adding Hardware I/O: Function call observation and SUT HW output checking



Call to library function is observed on host and expected SUT HW output is checked

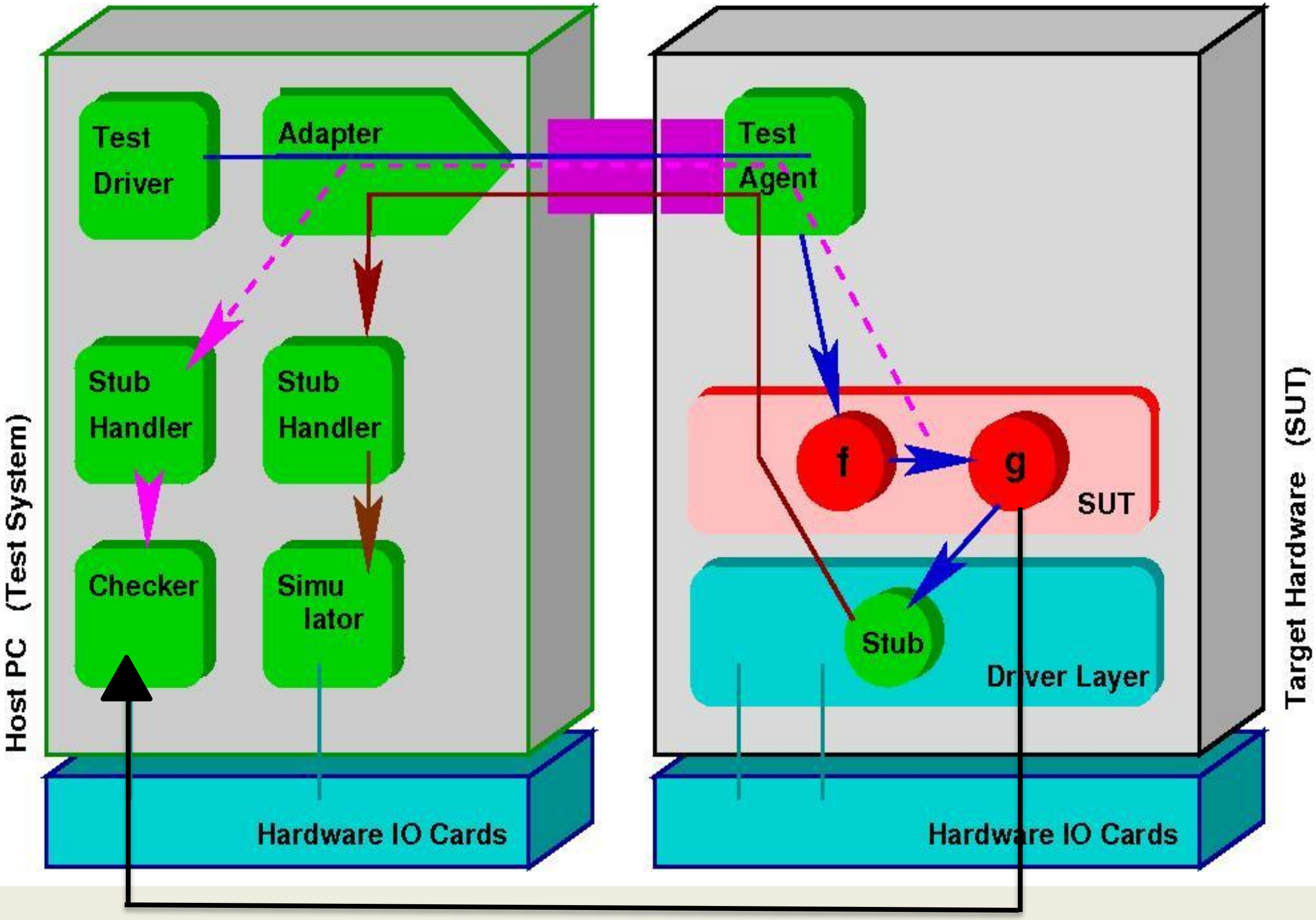
Library function

Driver

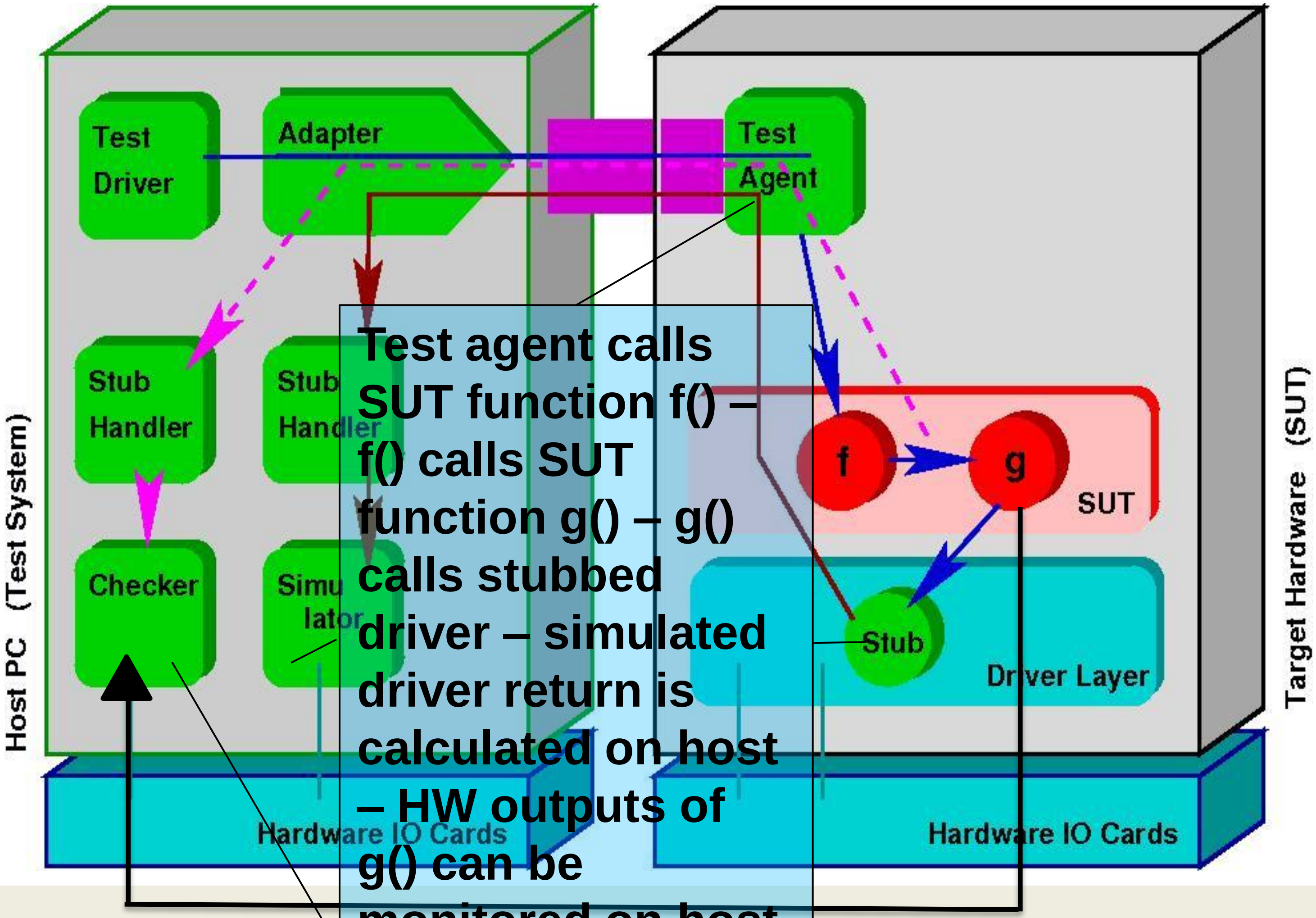
Building block: complex scenarios

- For many situations it does not suffice to call a single function per test step
- Instead, a **sequence of (timed) operations have to be performed without any interruption**
- Introduce **on-target test logic**:
  - **Add new functions** to target object code (written by the test designer)
  - Trigger these functions via remote function calls
  - New functions control **scenarios** with timed sequence of SUT function calls

Specialization: Unit testing on target HW

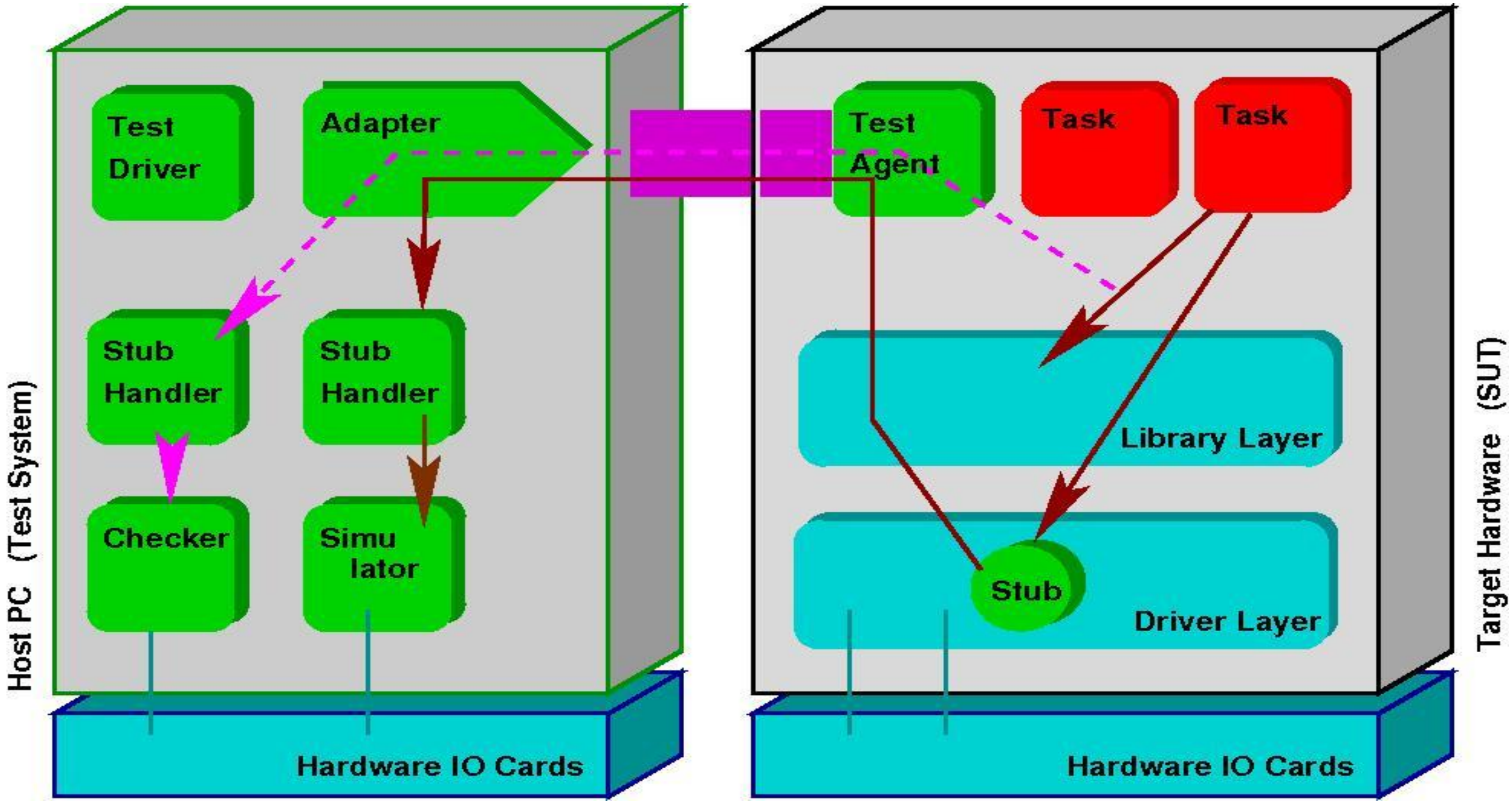


Specialization: Unit testing on target HW

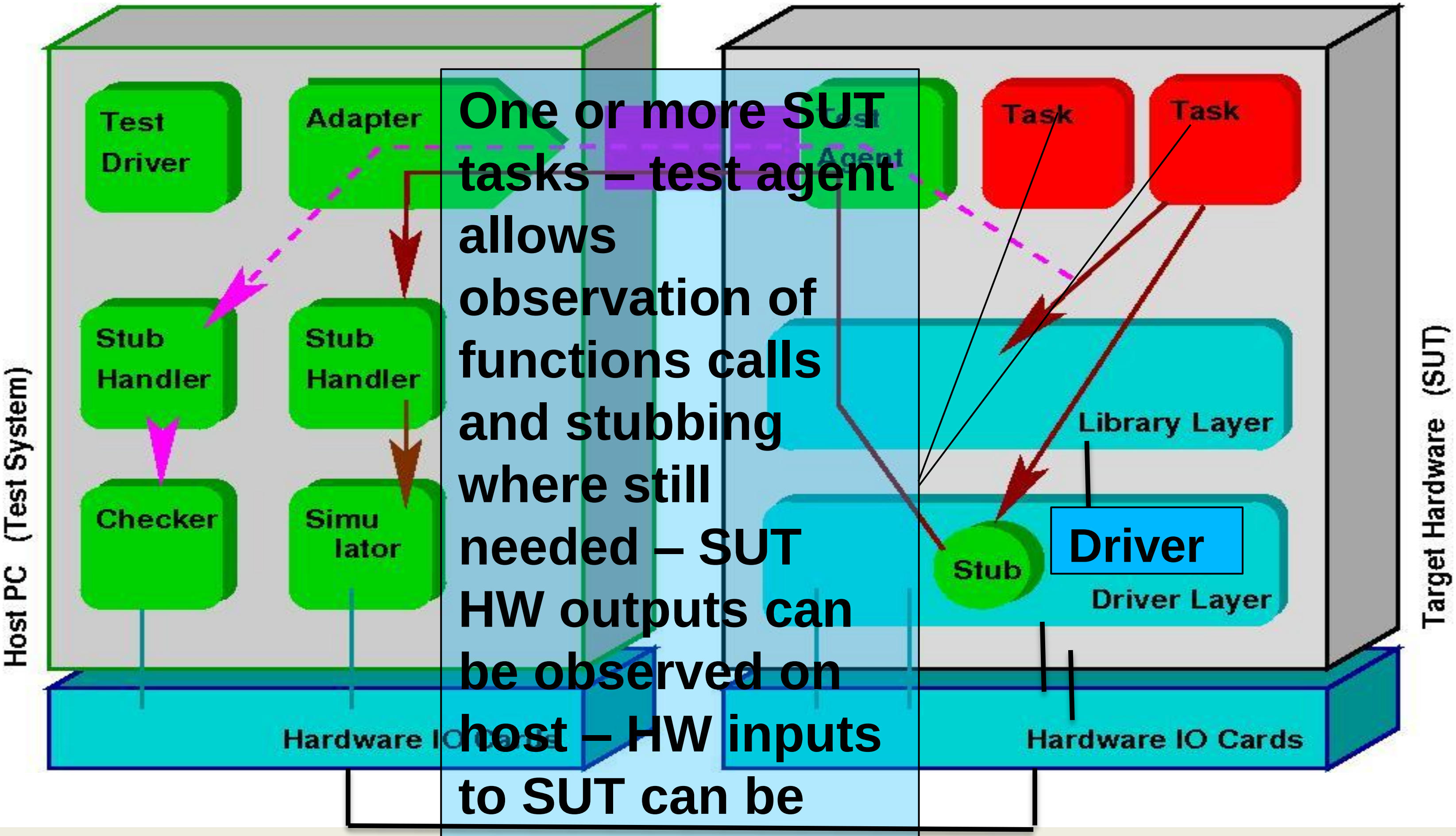


Test agent calls SUT function f() – f() calls SUT function g() – g() calls stubbed driver – simulated driver return is calculated on host – HW outputs of g() can be monitored on host

Specialization: SW integration testing on target HW



Specialization: SW integration testing on target HW



One or more SUT tasks – test agent allows observation of functions calls and stubbing where still needed – SUT HW outputs can be observed on host – HW inputs to SUT can be provided by host

Experiences – Project 1

- Multi-board embedded system (**Airbus aircraft cabin controller**):
- Development of an inter-board communication library layer (multiple CPU boards in one controller)
- 3 test agents (1 for each board) cooperating with host side test procedure
- Approx. 50 requirements
- small team
- custom hardware

Experiences – Project 2

- Test of on-board **Posix library layer for SysGo PikeOS**
- Embedded system is hosting several partitions
- SUT = C-standard library + C-mathematical library + communication layer
- > 2000 requirements
- > 15 team members
- several target hardware platforms
- Emulation environment available (QEmu)

Experiences – Project 3

- Test of **Rail Automation Library Layer for Siemens**
- Embedded system with custom hardware
- Custom observation of Hardware Output (as test environment input)
- Test-Agent replaces Application Logic
- Telegram based communication protocol → Host/Target exchange via Telegrams; no remote function calls/stubbing required
- > 50 requirements
- small teams (2-3 persons, 2 sites)

Conclusion

SWI-on-target testing complements conventional HW/SW integration testing:

- Unit tests and SW-integration tests are already performed on target HW with target machine code and linkage → **HW/SW integration-dependent errors are uncovered** at an early stage
- Major portion of **code coverage can be achieved on target HW**
- **Intrusive HW/SW integration testing can be avoided** since HW errors may be simulated by target-side stubs
- Observation of function call parameters enables **glass-box view on SUT**

Conclusion

- Code-generation for adapters and test-agents can be **automated**:
    - Test designers can concentrate on test logic
    - Successful application in 3 industrial projects – more to come!
- **Tool support** available: Verified's RT-Tester 6.x
- Other **available features** not discussed in this presentation:
    - Automated model-based test generation
    - Automated structural testing