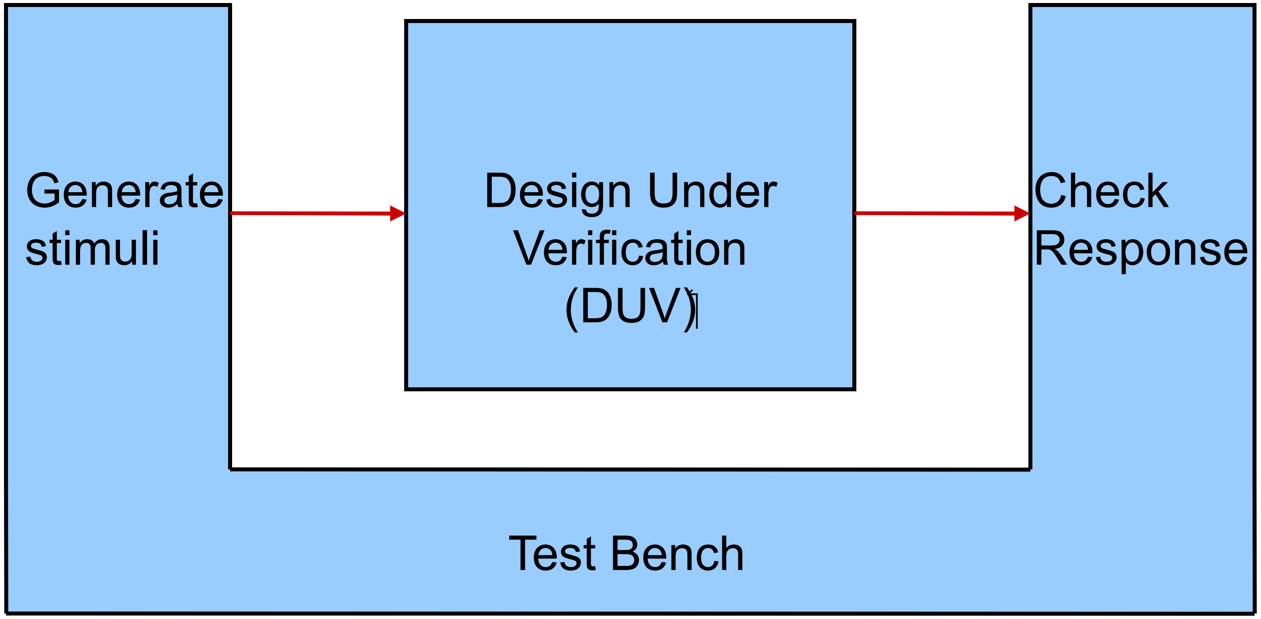
The Basic Testbench



Divide and Conquer

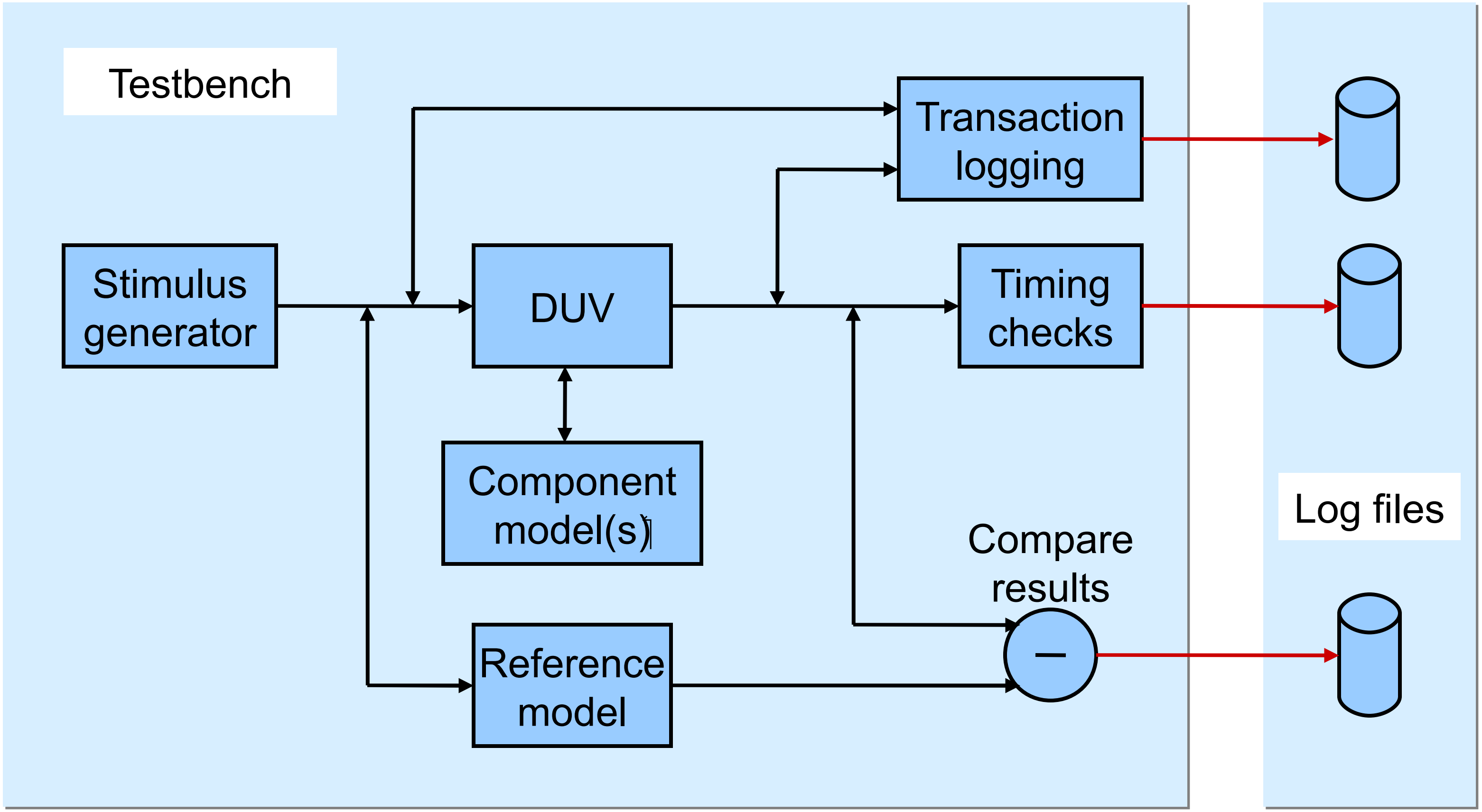
- Your design isn't one huge monolithic block
- Why should the testbench be like that?

- Issues:
  - Verification plan probably requires many different tests
  - Need to test several versions of the design
  - Create environment around the design

- Objectives:
  - Flexibility
  - Low-effort maintenance and enhancement
  - Reliability
  - Re-use

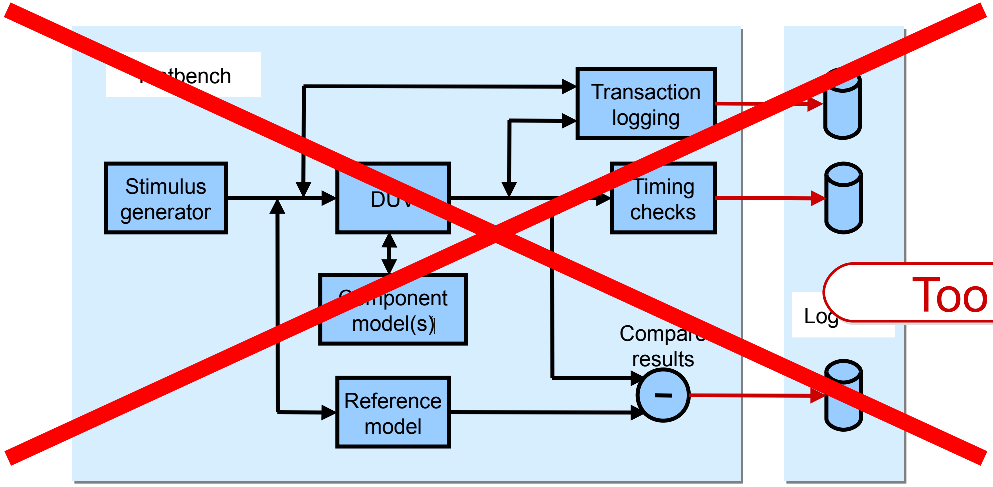
Build the Complete Testbench

- Not ideal - what should be changed to run a different test?



Monolithic Testbenches are Inflexible

- Consider what would need to be changed to run different test cases
  - *test case* = group of tests required by one aspect of the verification plan, run as one simulation
- Stimulus generator and file output writers would need to change
- Top-level testbench then changes to accommodate them



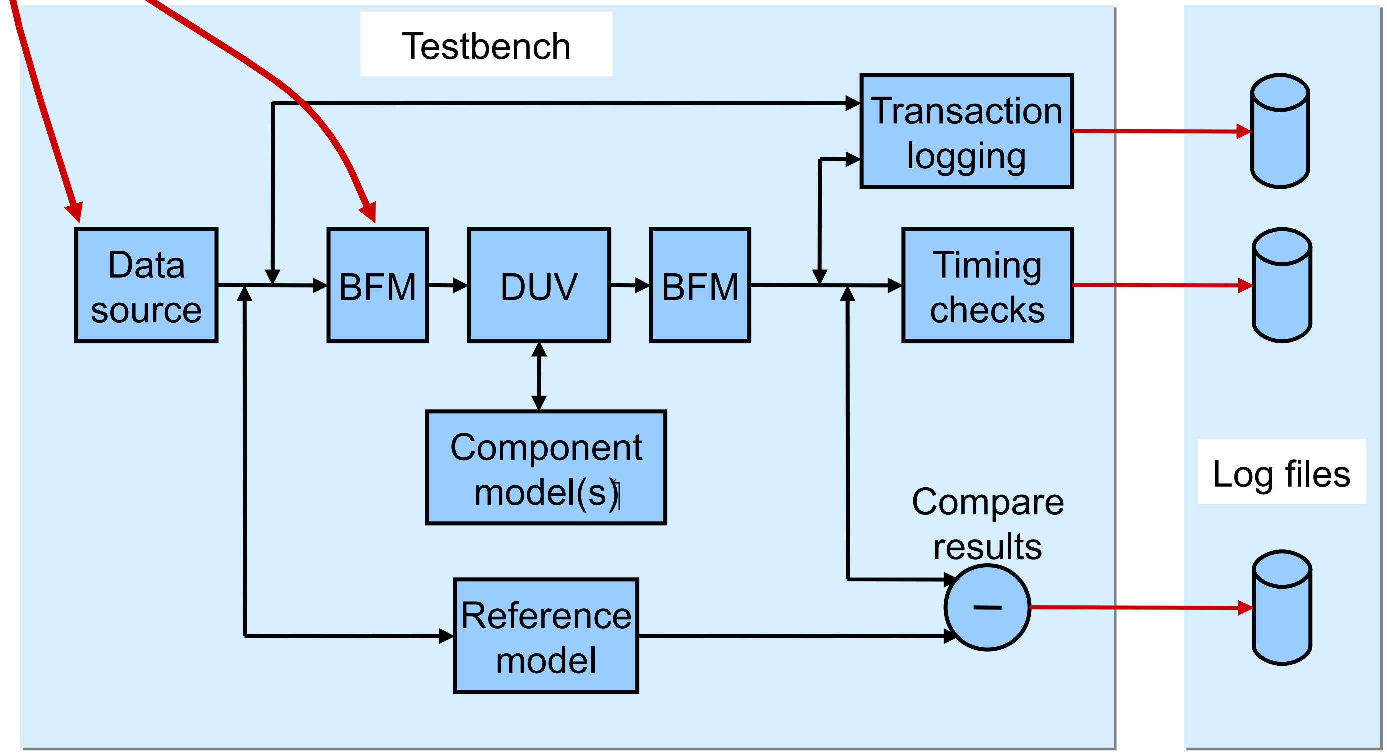
Too much work to change!

Package Useful Functionality

- Component models for other parts of the system
- Stimulus generators
- Output checkers
- Logging and monitoring
- Utility functions (application-specific calculations)

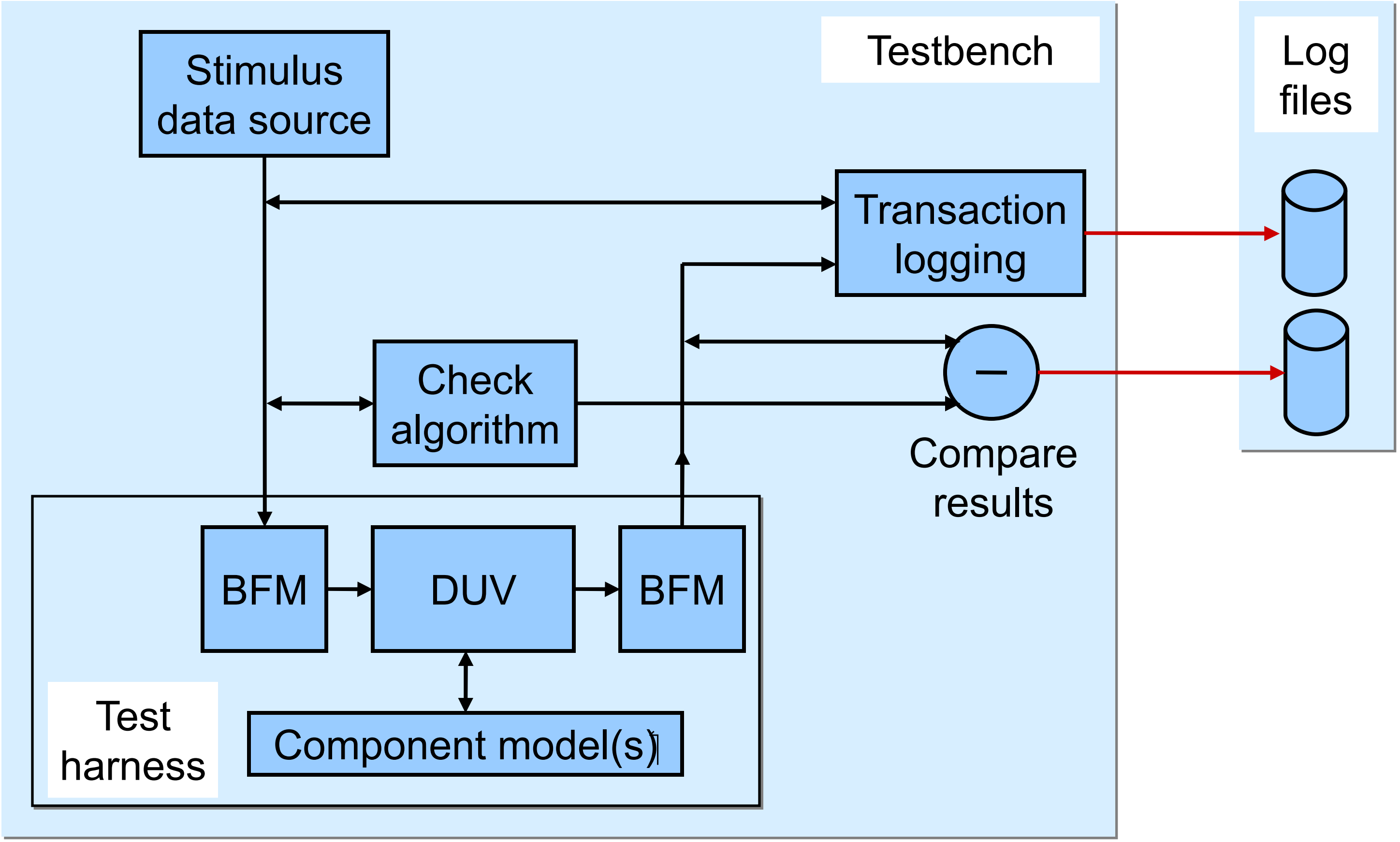
Hide DUV Interface from Testcase

- Split data sources and sinks:
  - Test-dependent section is now independent of DUV
  - DUV-dependent section, typically a Bus Functional Model, determines timing protocols and signal formats at DUV interface



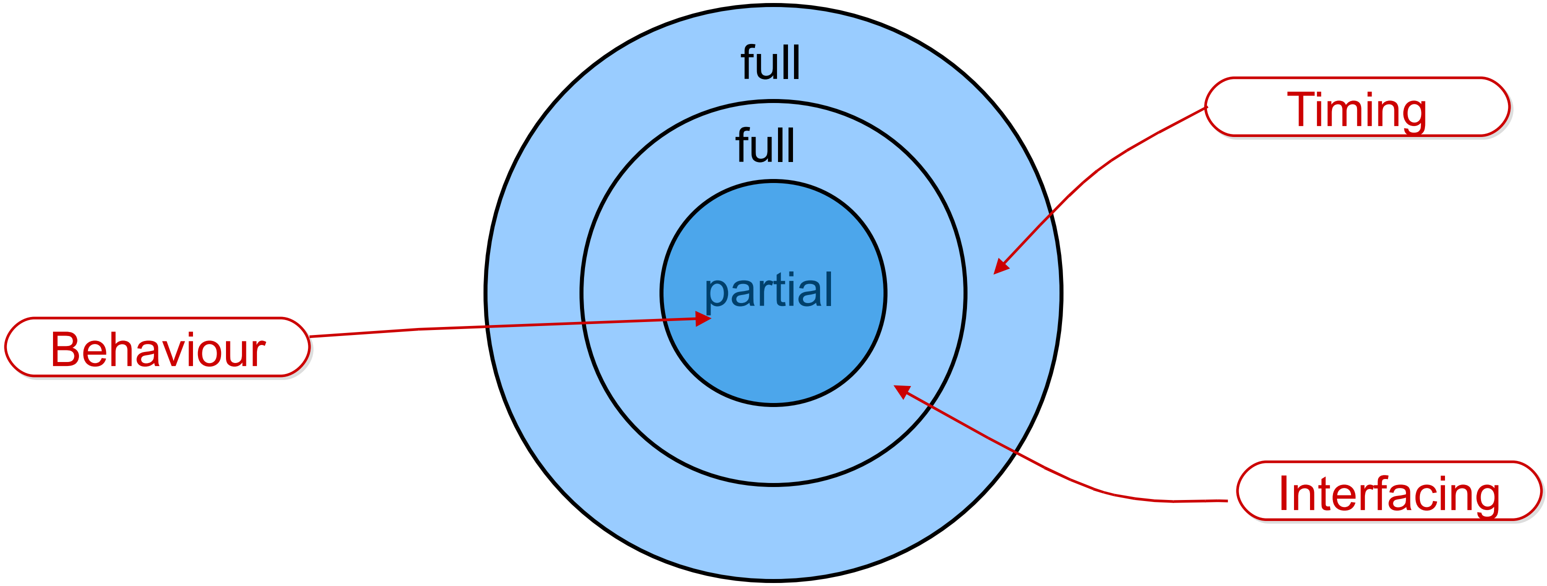
Layered Architecture

- Package DUV and its support environment as a *test harness*
    - Hides DUV interface details from testcase
    - Re-usable without change over various testcases



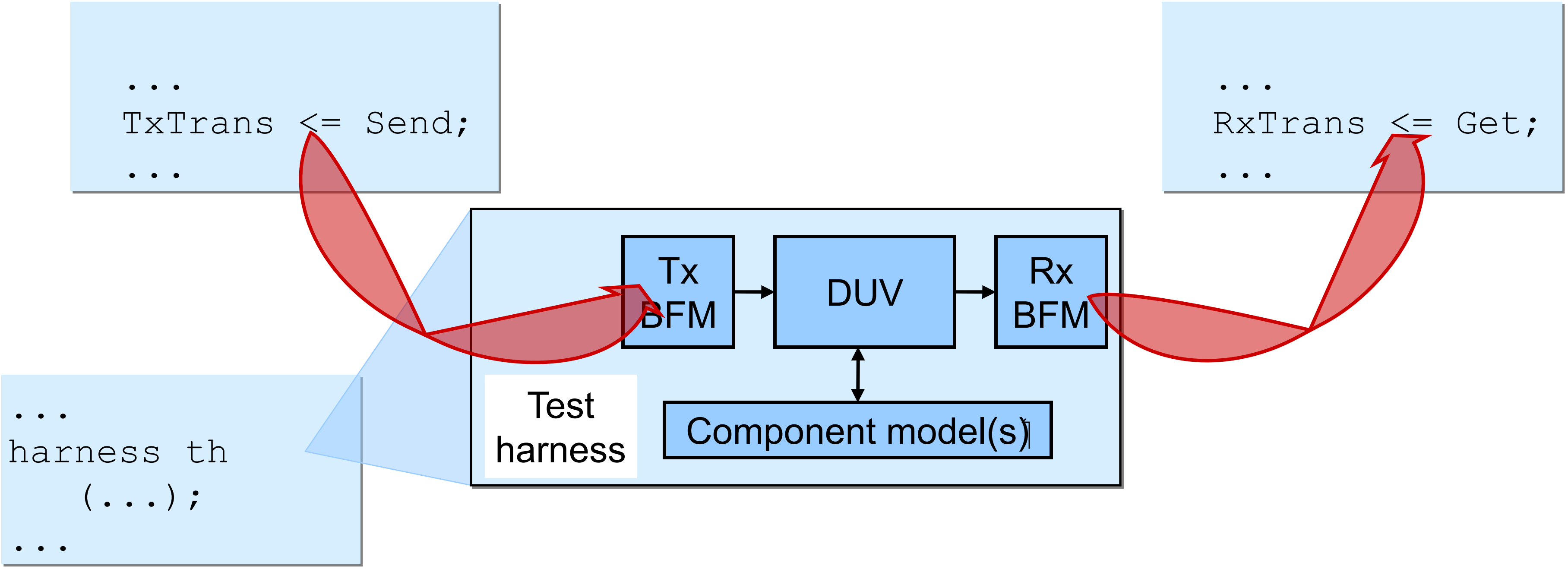
Bus Functional Modelling

- A Bus Functional Model (BFM) models timing and physical implementation of the interface, but simplifies the behaviour
- You may also come across phrases such as
  - transactor
  - transaction verification model (TVM)
  - adapter
- Key point is to abstract the details so the user doesn't need to know them



Interface Between Layers

- Low level signal interface is hard to maintain
  - Adding new interface features will break existing test code
- Hide detail using record types:
  - can be extended without breaking existing code
  - requires handshake



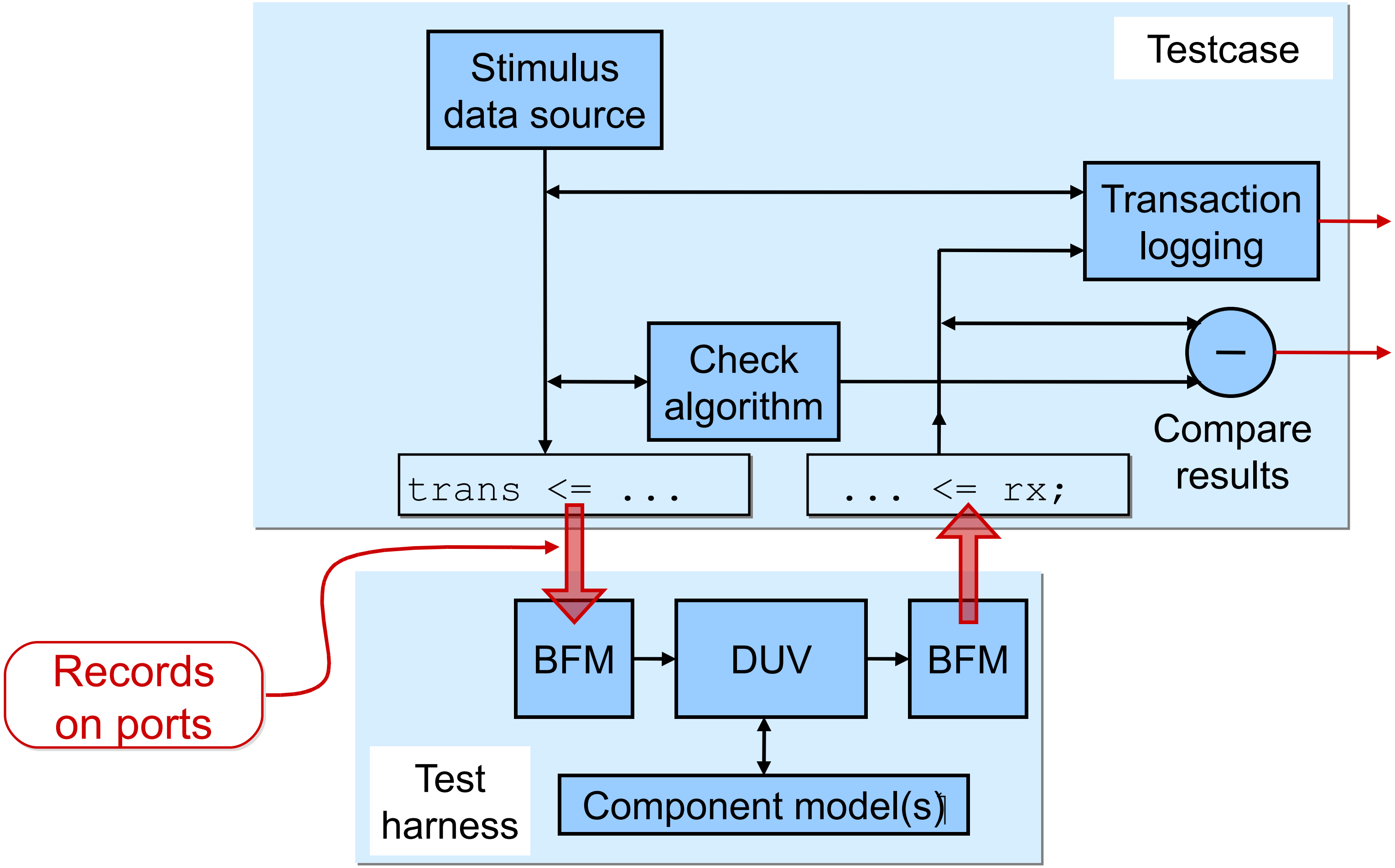
```
...
TxTrans <= Send;
...
```

```
...
RxTrans <= Get;
...
```

```
...
harness th
    (...);
...
```

Separate Top-Level Entities

- Don't instantiate a test harness within a testcase
- Testcase, test harness are parallel entities at the same level



Summary

- Always aim for maximum reusability of verification code
- Structure test benches to hide unnecessary detail
  - Code that doesn't use hidden detail is portable to new situations with the same interface
- Use records and procedures
  - Separates implementation detail from interface
  - Can be extended without breaking existing code (just add some new record fields)
- Split test cases from test harnesses

End ▶