Praktikum: SystemC

SystemC-TLM Tutorial

Joachim Falk (`falk@cs.fau.de`)

Agenda

- SystemC and TLM
- Transaction
- TLM 2.0
  - Overview
  - Interfaces
  - Examples
- Virtual Prototype

SystemC/TLM

- C++ library that allows for high-level system modeling
- Provides
  - Concepts
    - Modules
    - Ports
    - Channels
    - Processes
    - Events
  - Data types
  - Simulation kernel
- With it, modeling of communicating concurrently executed modules is easy

- SystemC TLM 2.0: Transaction level modeling
  - Even more abstract modeling possible

Agenda

- SystemC and TLM
- Transaction
- TLM 2.0
  - Overview
  - Interfaces
  - Examples
- Virtual Prototype

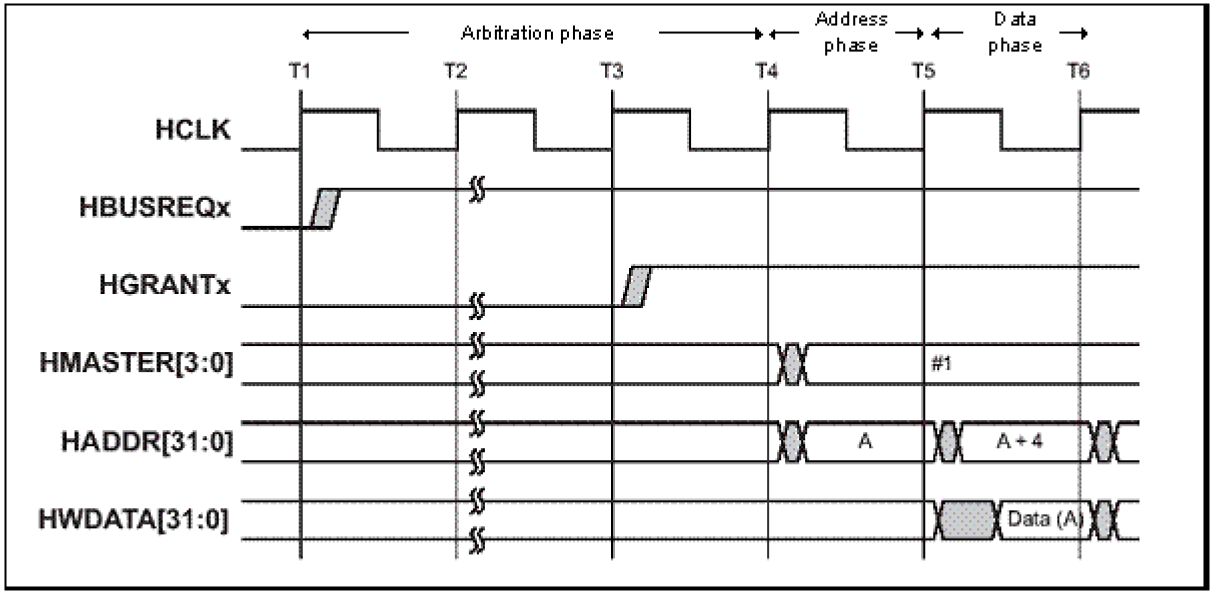
Transaction?

- Transaction: Abstraction of communication
- *Not* transaction level: C function or single process
  - Algorithmic model
- TL requires multiple processes to simulate concurrent execution and communication
- Note: Some slides and phrases are taken from OSCI documentation (available at www.systemc.org)
  - OSCI TLM2 User Manual
  - Slides from the TLM 2.0 kit

Transaction on AHB Bus

- Opt1: Encapsulate protocol in a SystemC channel

- Opt2: Abstract timing to entire transactions: TLM 2.0



[http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0119e/index.html]

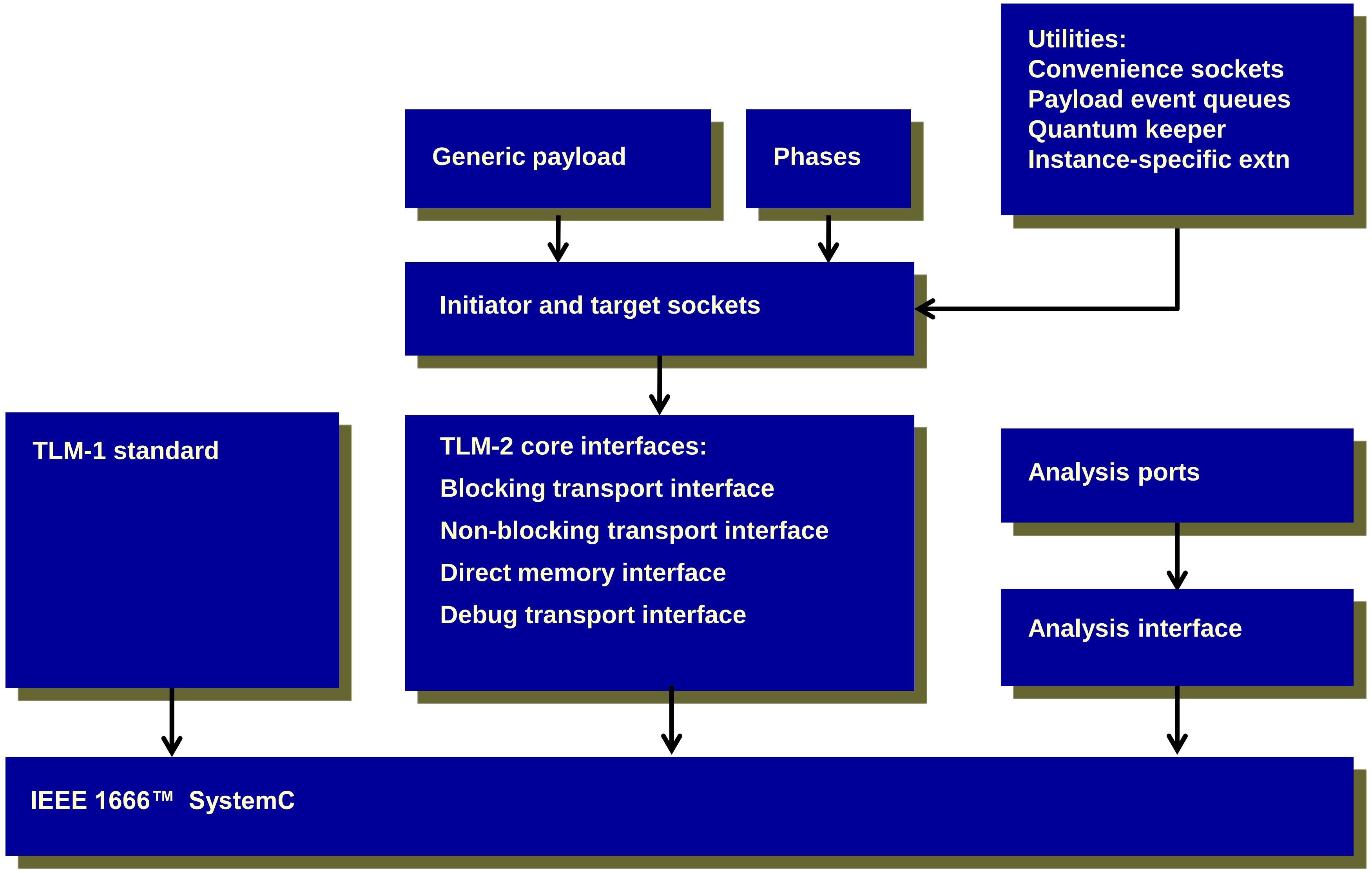
Agenda

- SystemC and TLM
- Transaction
- TLM 2.0
  - Overview
  - Interfaces
  - Examples
- Virtual Prototype

TLM 2.0

- Open SystemC Initiative (OSCI) standard (June 2008)
- Mission: Standardize the way models communicate
- Why use TLM 2.0 in system level modeling?
  - Standard for interoperability
  - Early available
  - High simulation speed
- Use cases for TLM
  - Represents key architectural components of hardware platform
  - Architectural exploration, performance modeling
  - Software execution on virtual model of hardware platform
  - Golden model for hardware functional verification

The TLM 2.0 Classes

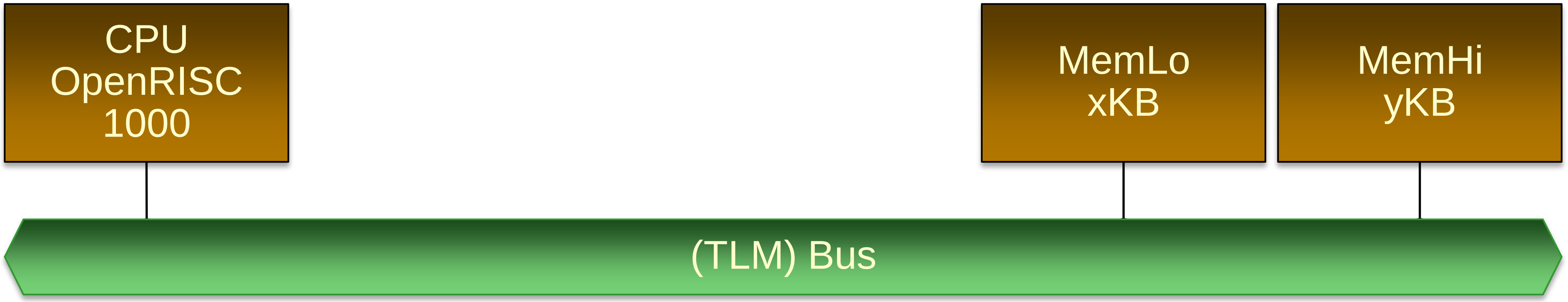


Coding Styles

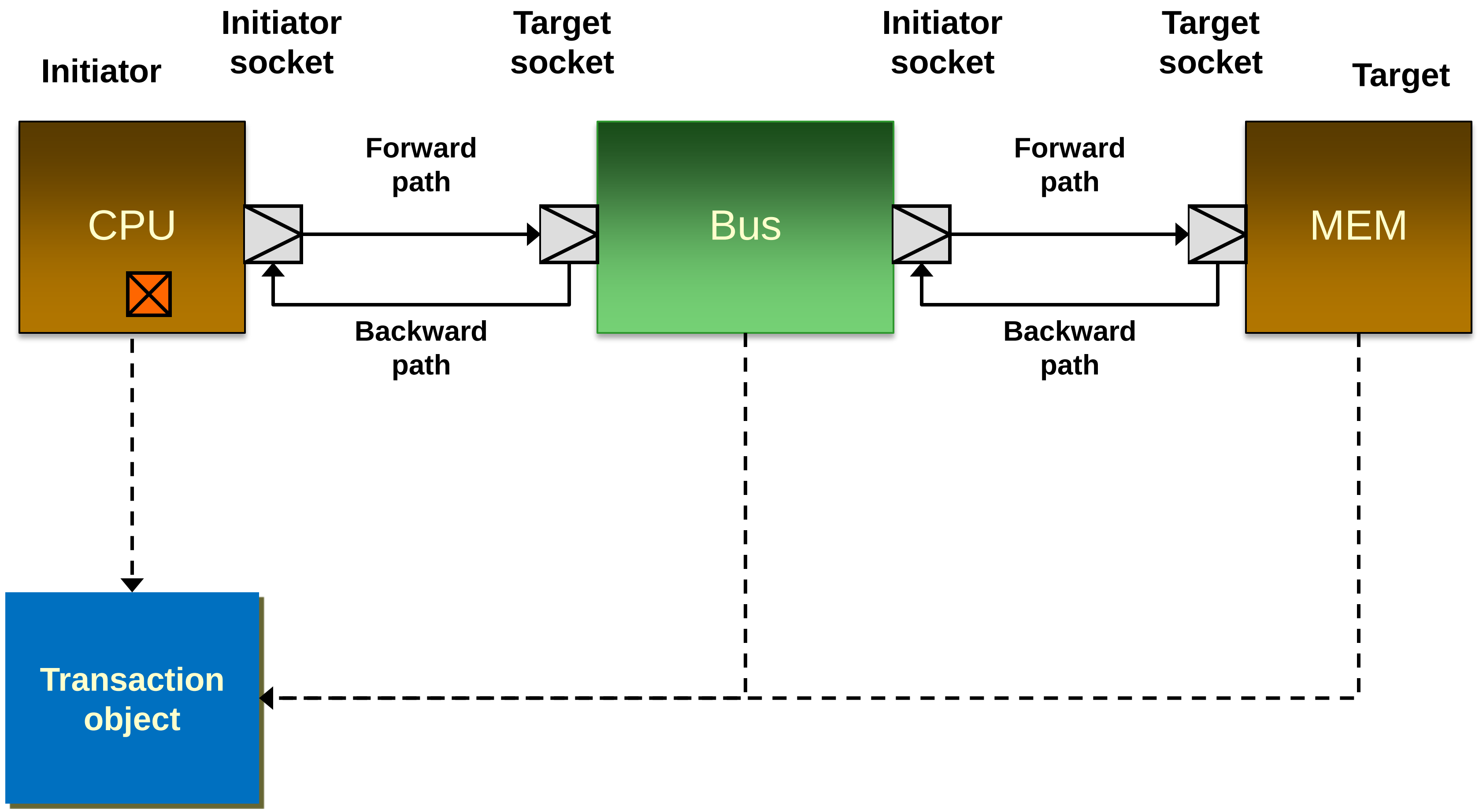
- *Guides* to model writing
- Chosen style depends on use case
- Each style can support a range of abstraction across functionality, timing, and communication

- Loosely-timed
  - Processes are temporally decoupled from simulation time (may run ahead)
  - Each transaction has two timing points (begin and end)
- Approximately-timed
  - Processes run in lock-step with simulation time: Delays annotated onto transactions cause waits or timed notifications
  - Each transaction has four timing points (phases)

Example



Initiators and Targets



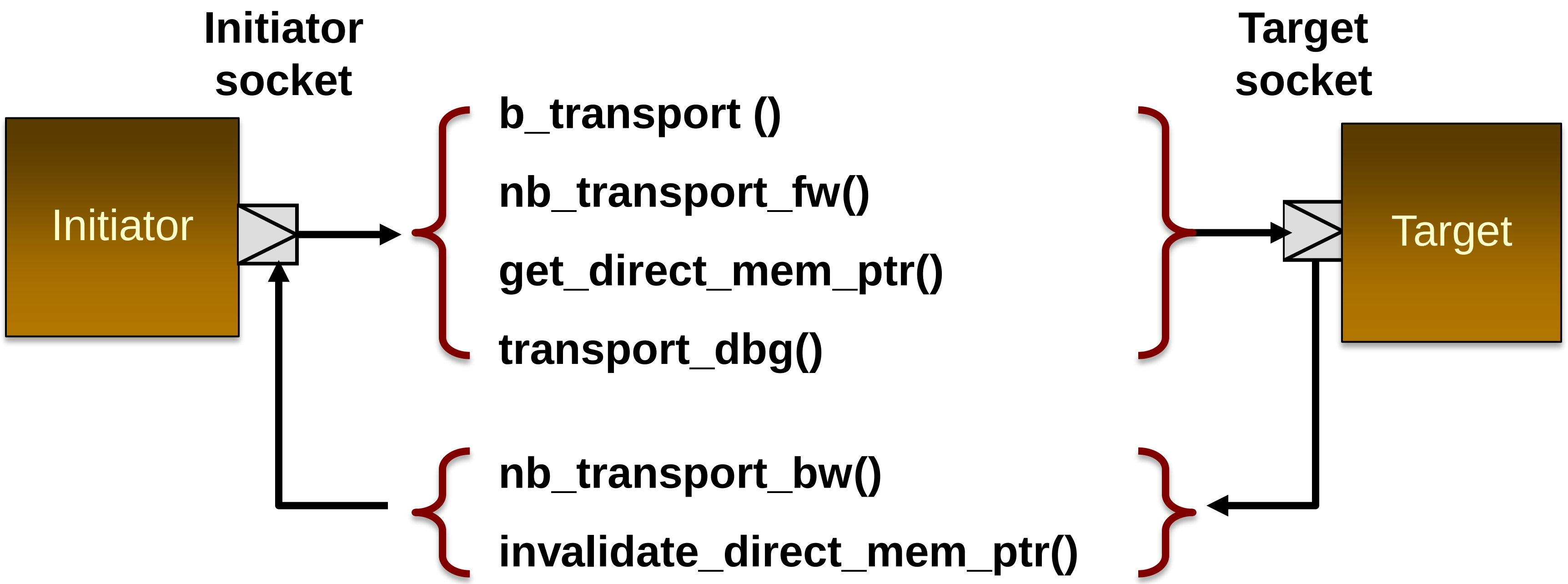
References to a single transaction object are passed along the forward and backward paths

Agenda

- SystemC and TLM
- Transaction
- TLM 2.0
  - Overview
  - Interfaces
  - Examples
- Virtual Prototype

Initiator and Target Sockets

- Initiator and target are connected via sockets
- Sockets
  - group transport, DMI, and debug transaction interfaces
  - bind forward and backward path with a single call
  - Convenience: “simple” sockets



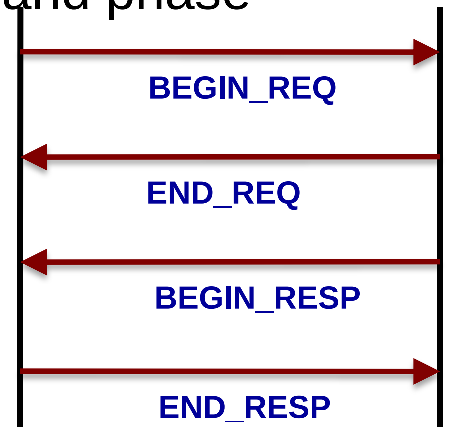
Blocking/Non-blocking Transport

- Blocking transport
  - Typical usage: Loosely-timed coding style
  - Parameter: Transaction and timing annotation
  - Uses forward path only

```
void b_transport(TRANS &, sc_time &);
```

- Non-blocking transport
  - Typical usage: Approximately-timed coding style
  - Parameter: Transaction, timing annotation, and phase
  - Calls on forward- and backward-paths

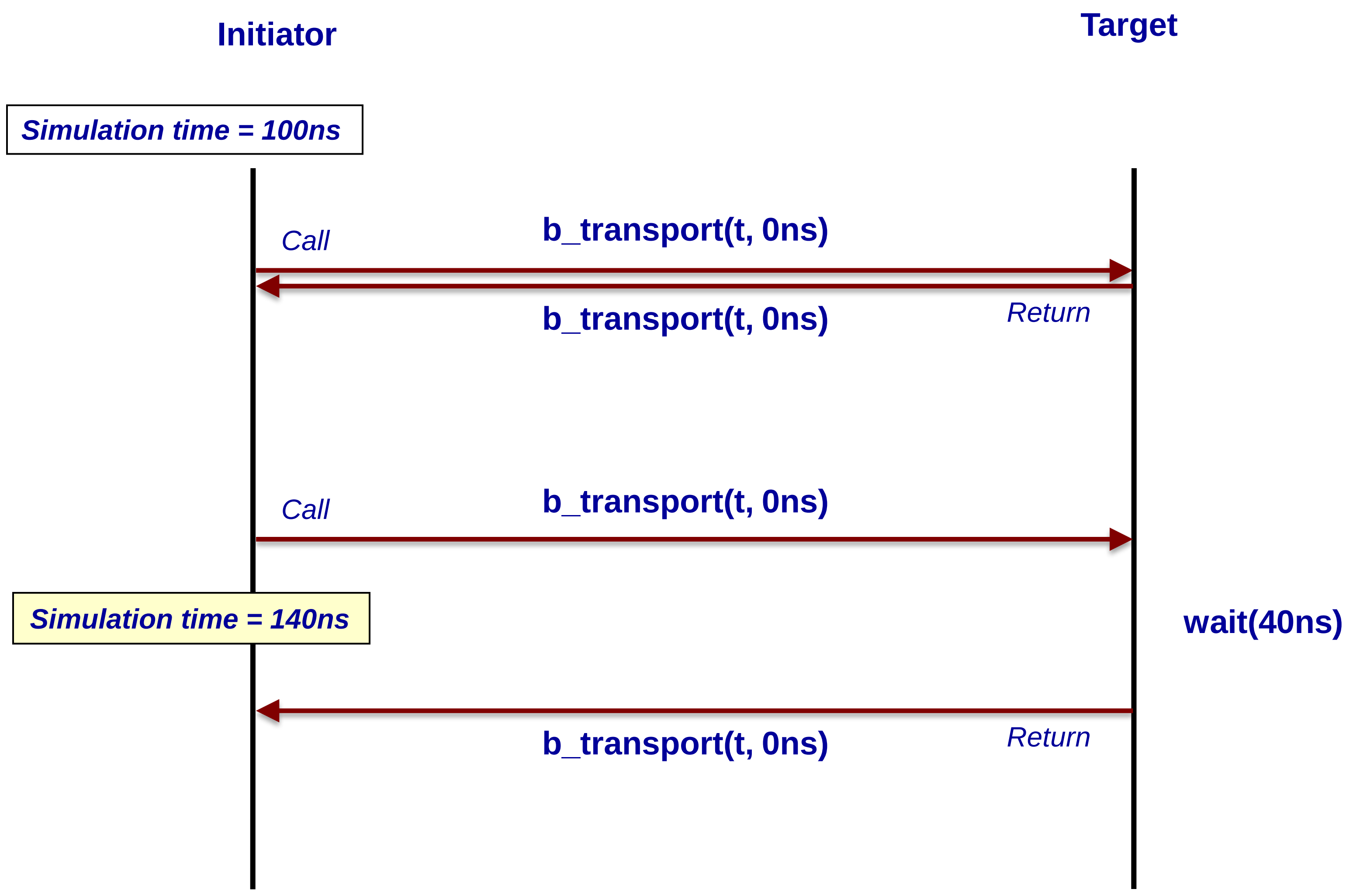
```
tlm_sync_enum
nb_transport_fw(TRANS &, PHASE &, sc_time &);

tlm_sync_enum
nb_transport_bw(TRANS &, PHASE &, sc_time &);
```



Transport Interfaces

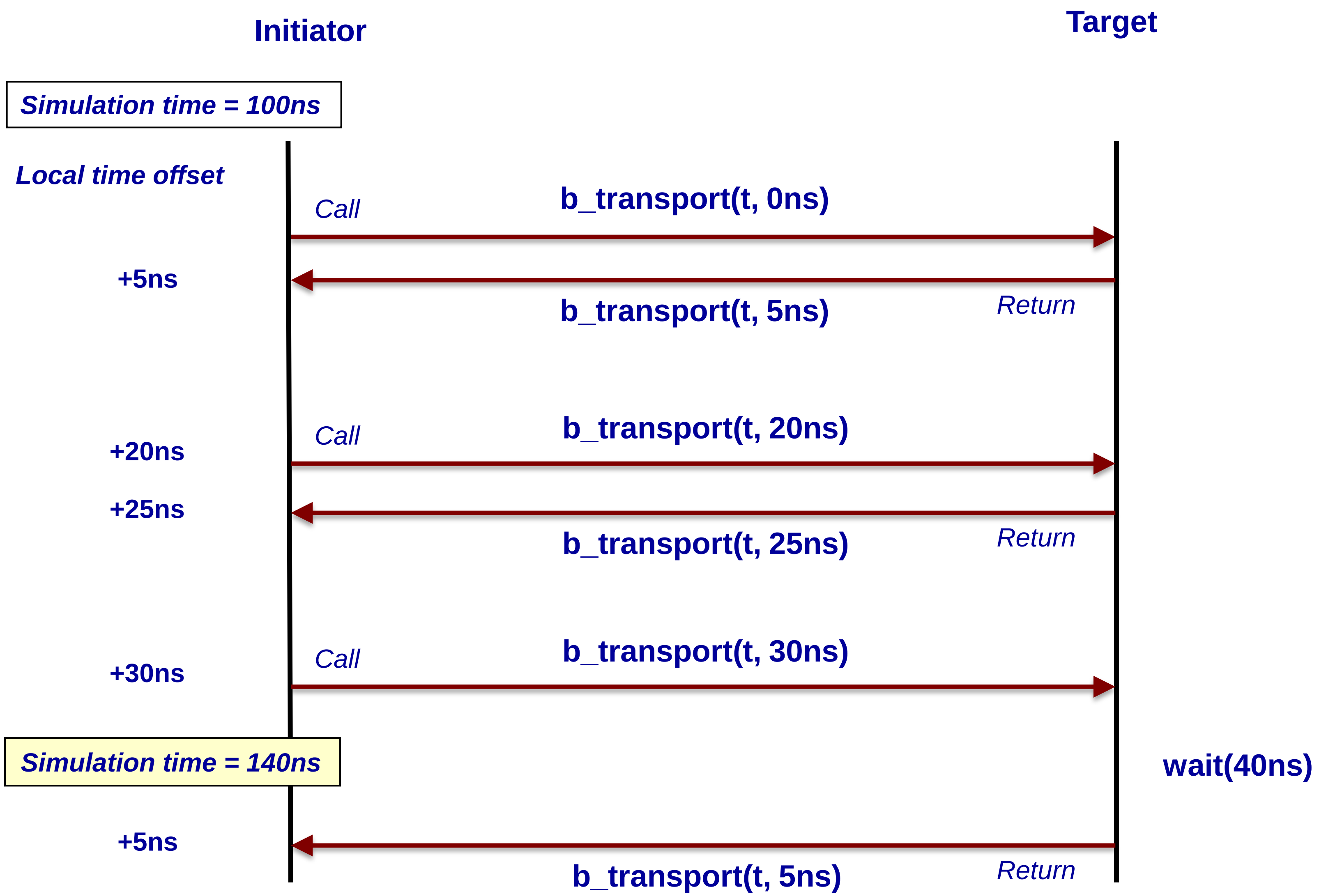
- TLM2 transport interfaces pass transactions from initiators to targets
- Forward path
  - Transaction is transported by `b_transport()`/ `nb_transport_fw()` calls from the initiator to the target
  - Traveling through interconnection network or fabric possible
- Transaction is executed in the target
- Backward path
  - Blocking: Transaction "returns" to initiator carried with the return from the `b_transport()` calls as they unwind
  - Non-blocking: Passed back by making explicit `nb_transport_bw()` calls in the opposite direction

Blocking Transport



Initiator is blocked until return from b_transport

Blocking - Temporal Decoupling



Generic Payload

- Possible type of transaction
- Appropriate for memory mapped buses
- Attributes
  - Typical attributes of MMBs (command, address, pointer, …)
  - TLM specific (return status, DMI hint)
  - Extensions (ignorable for interoperability)
- Most attributes are set by initiator and shall not be modified by interconnect components or the target; exceptions are
  - Address (only interconnect, for example, a router)
  - Response status (only target)
  - If read command: data array by target

Generic Payload

- Payload (transaction object) is “transported” by reference
- Memory management for the transaction object
  - Ad hoc by the initiator (static, dynamic, automatic, pool strategy, etc.)
  - Provide memory manager
- Initiators must not delete transactions until their lifetime ends
- Initiators are responsible for valid data pointers and corresponding memory management

Agenda

- SystemC and TLM
- Transaction
- TLM 2.0
  - Overview
  - Interfaces
  - Examples
- Virtual Prototype

Example (Loosely-timed Coding Style)

```
struct Initiator: sc_module {
  tlm_utils::simple_initiator_socket<Initiator> socket;
  …
};

struct Memory: sc_module {
  tlm_utils::simple_target_socket<Memory> socket;
  …
};

SC_MODULE(Top) {
  Initiator initiator;
  Memory    memory;

  Top(sc_module_name name)
    : sc_module(name),
      initiator(“initiator”),
      target(“target”)
  {
    initiator->socket.bind(memory->socket);
  }
};
```

Example - Initiator

```
int data;

tlm::tlm_generic_payload trans;
sc_time delay = sc_time(10, SC_NS);

trans.set_write();
trans.set_address(0x100);
trans.set_data_ptr(reinterpret_cast<unsigned char*>(&data));
trans.set_data_length(sizeof(data));
trans.set_streaming_width(sizeof(data));
trans.set_byte_enable_ptr(0);
trans.set_dmi_allowed(false);
trans.set_response_status(tlm::TLM_INCOMPLETE_RESPONSE);

initiatorSocket->b_transport(trans, delay);

// check response status, perform delay
```

Example - Target

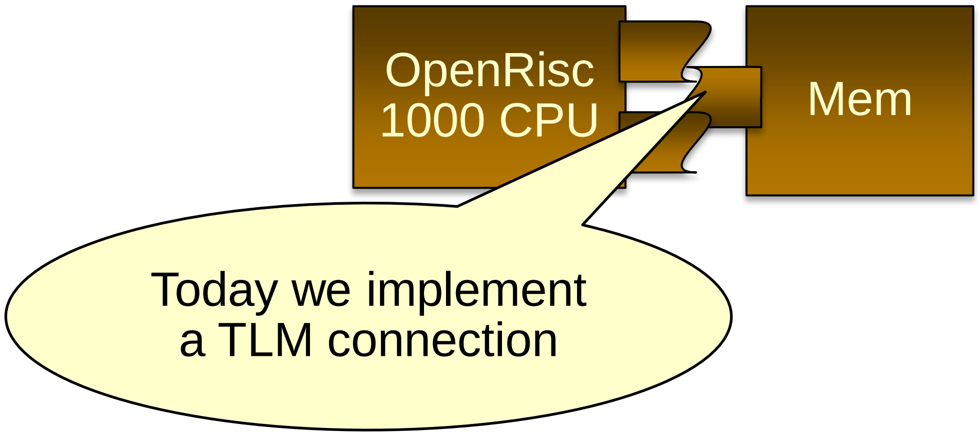
```
void Target::b_transport(
    tlm::tlm_generic_payload &trans,
   sc_time &delay)
{
  addr_type addr = trans.get_address();
  if (addr > mMaxAddr) {
    // set error response status and return
  }

  tlm::tlm_command cmd = trans.get_command();
  unsigned char* ptr = trans.get_data_ptr();
  unsigned int   len = trans.get_data_length();
  if (cmd == tlm::TLM_READ_COMMAND)
    memcpy(ptr, &mMemory[addr], len);
  else if (cmd == tlm::TLM_WRITE_COMMAND)
    memcpy(&mMemory[addr], ptr, len);
  else {
    // set error response and return
  }
  // add some delay
  delay = delay + sc_time(5, SC_NS);
  trans.set_response_status(tlm::TLM_OK_RESPONSE);
}
```

Agenda

- SystemC and TLM
- Transaction
- TLM 2.0
  - Overview
  - Interfaces
  - Examples
- Virtual Prototype

Virtual Prototype

- What do we have?
  - OpenRisc 1000 CPU with Harvard architecture
    (That is, two TLM connections one for data and one for code)
  - Memory
    (With only one TLM connection)


OpenRisc 1000 CPU
Mem
Today we implement a TLM connection