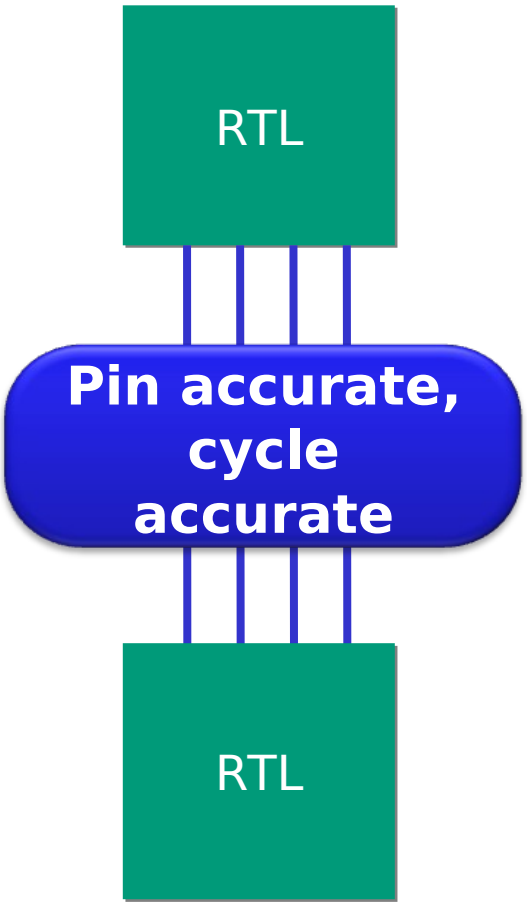
Transaction Level Modeling

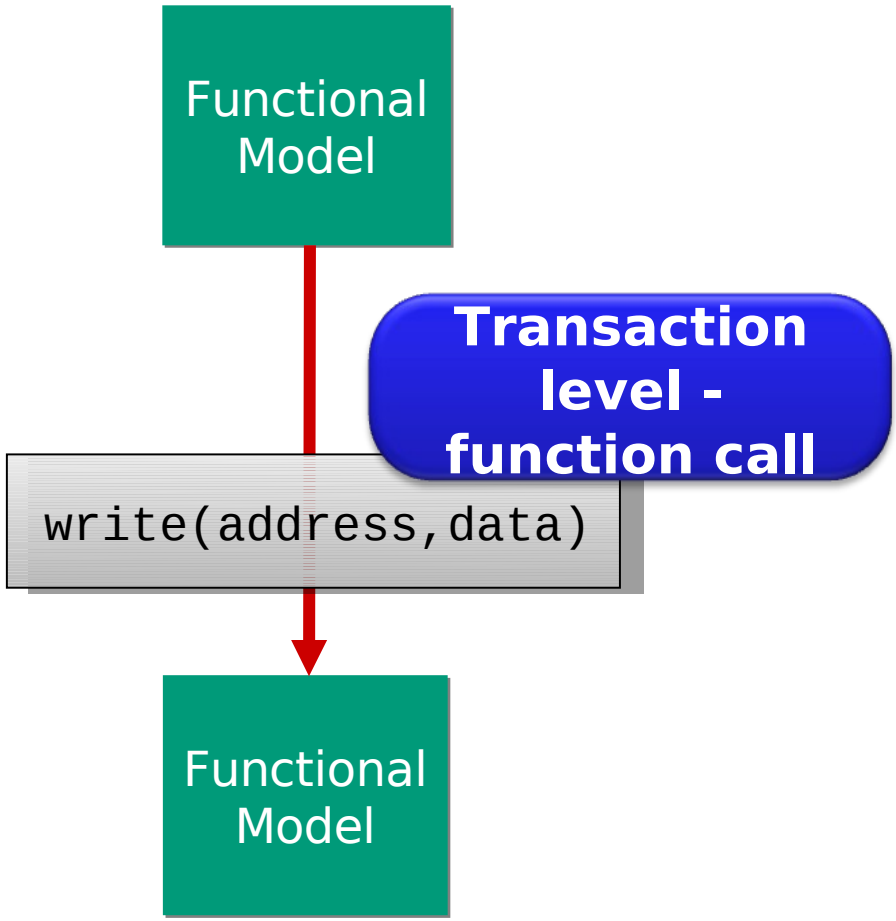
Renesas Design Vietnam Co., Ltd.
Binh Nguyen, SLD Team
Frontend Group

October 10, 2012 Rev. 0.00

Introduction

TLM - Transaction Level Modeling

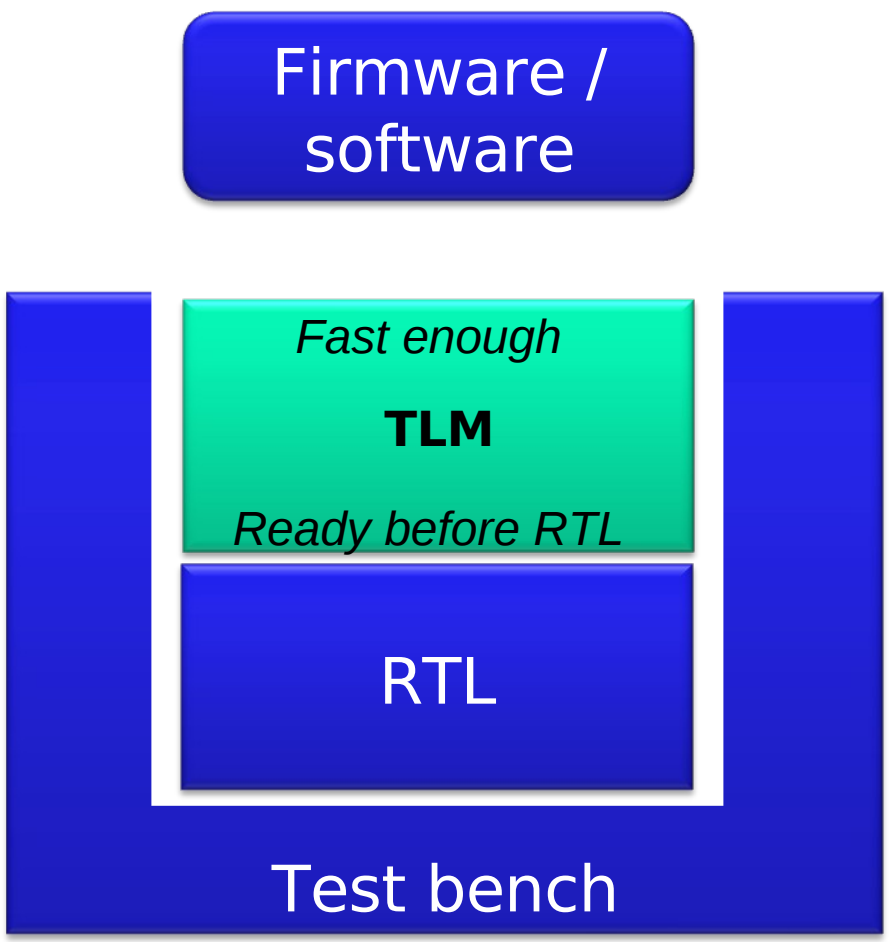




Simulate every event

100-10,000 X faster simulation

Reasons for Using TLM



Accelerates product release schedule

Software development

Architectural modeling

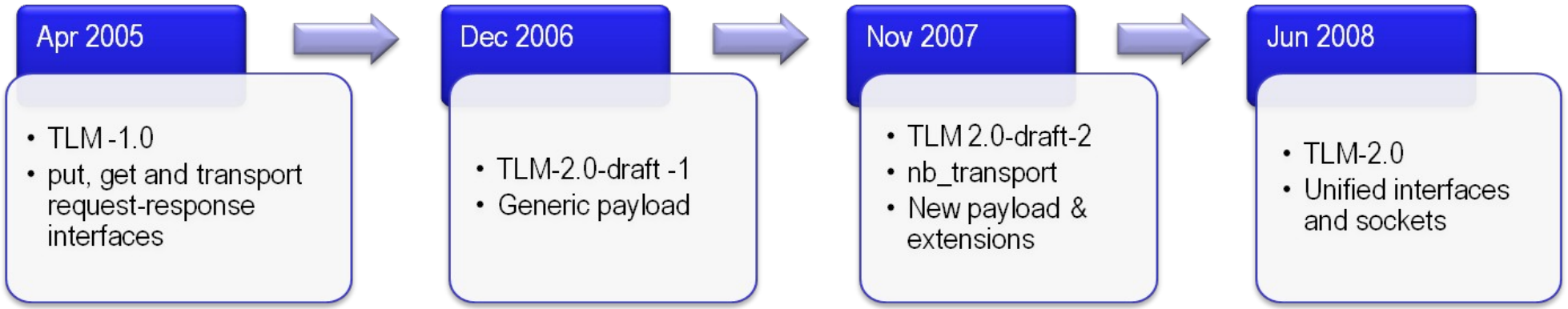
Hardware verification

TLM = golden model

TLM Use Cases

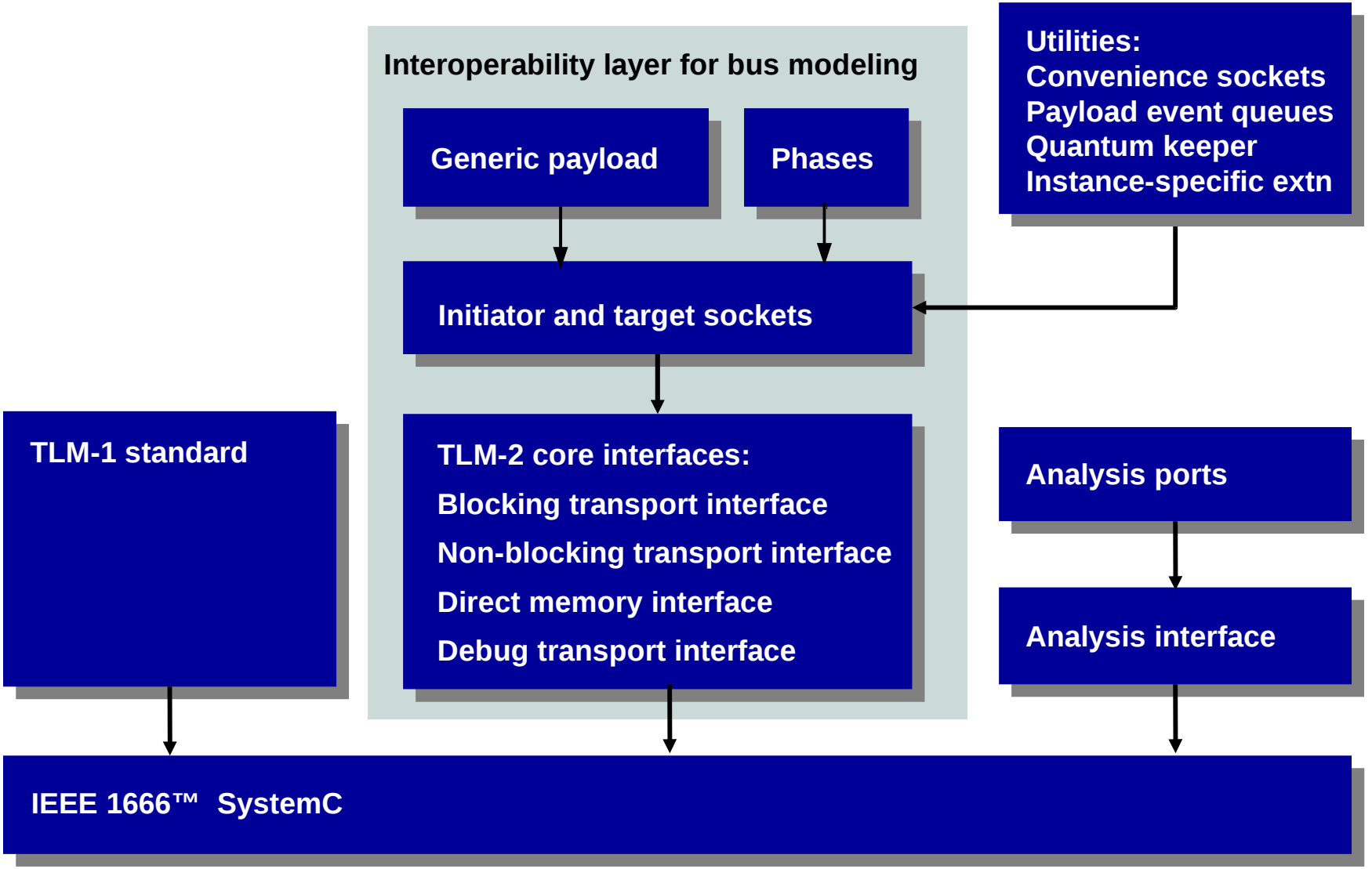
- Represents key architectural components of hardware platform
- Architectural exploration, performance modeling
- Software execution on virtual model of hardware platform
- Hardware functional verification

OSCI TLM Development

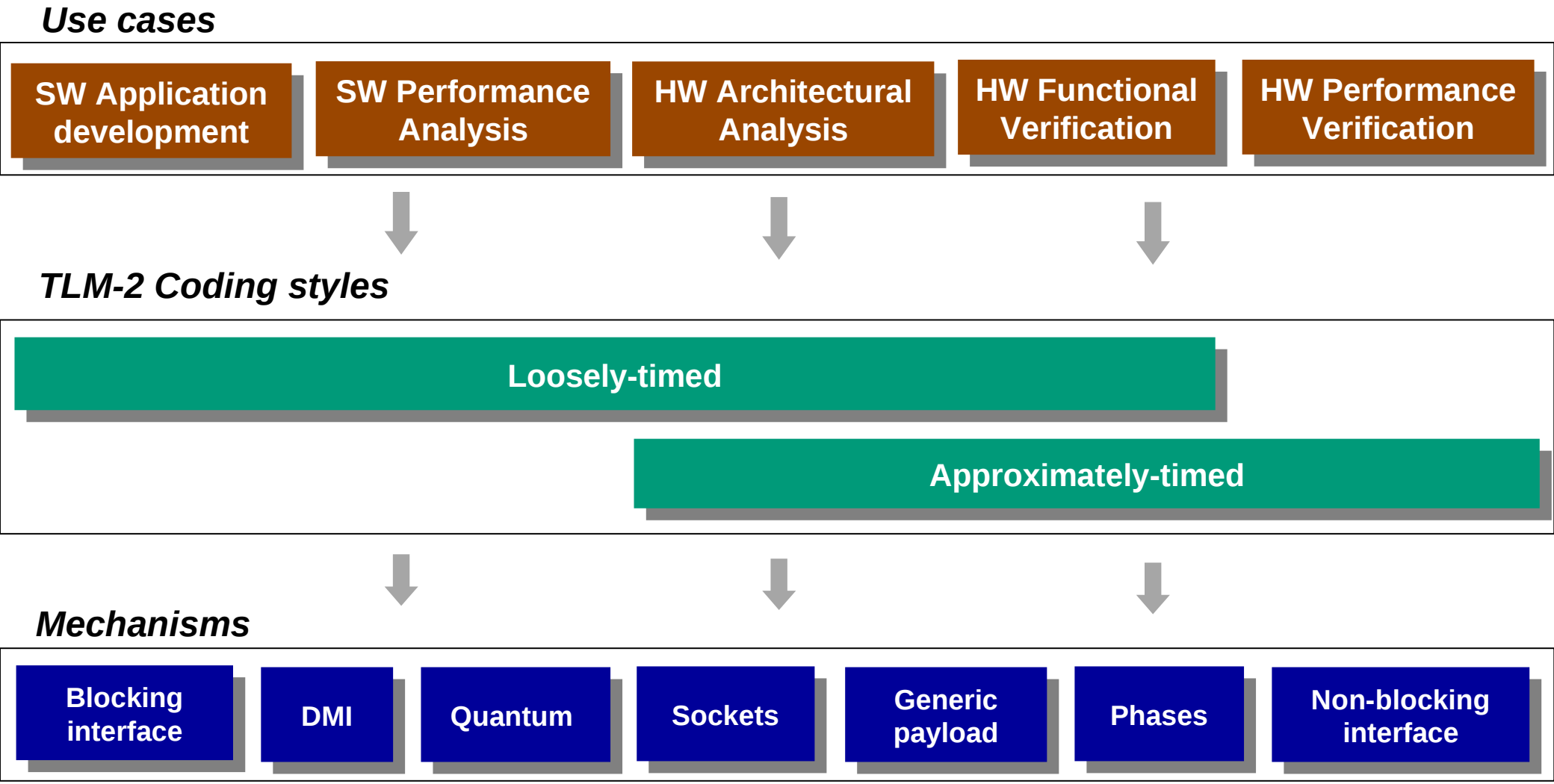


OSCI - Open SystemC Initiative (now is Accellera)

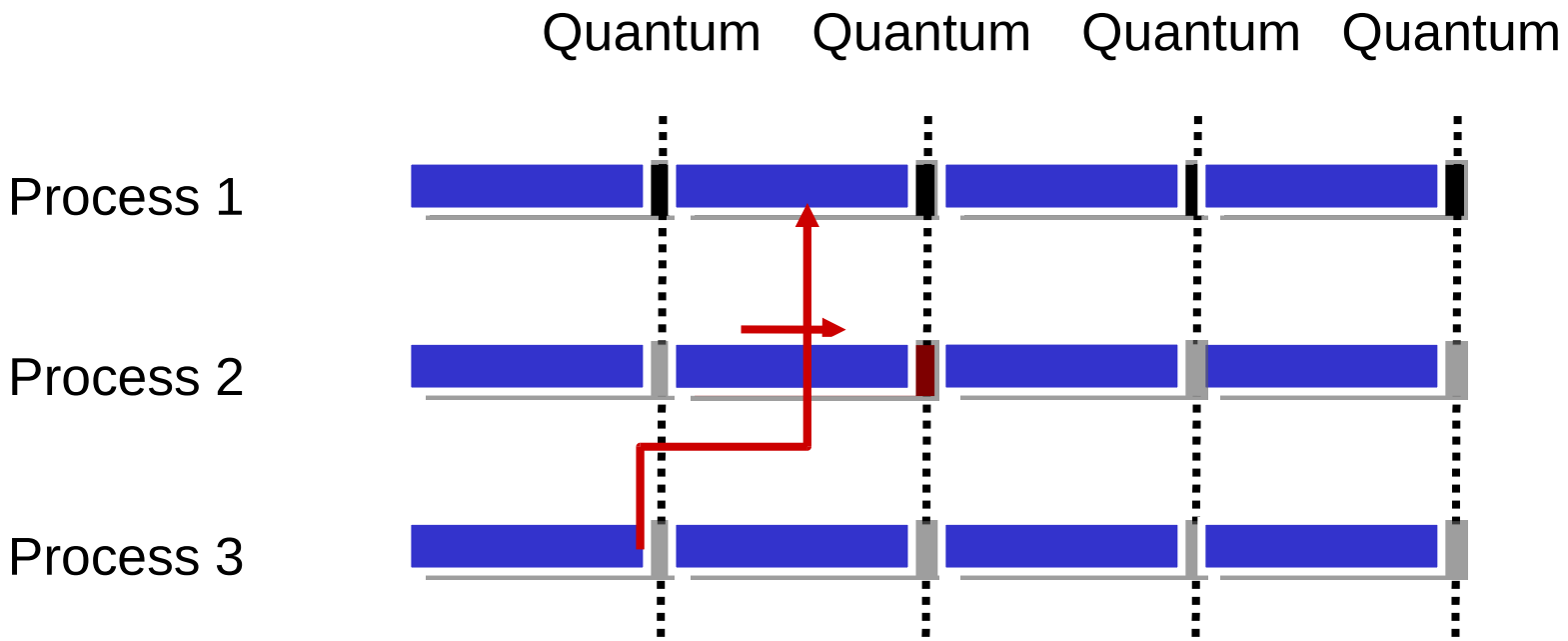
The TLM 2.0 Classes



Use Cases, Coding Styles and Mechanisms

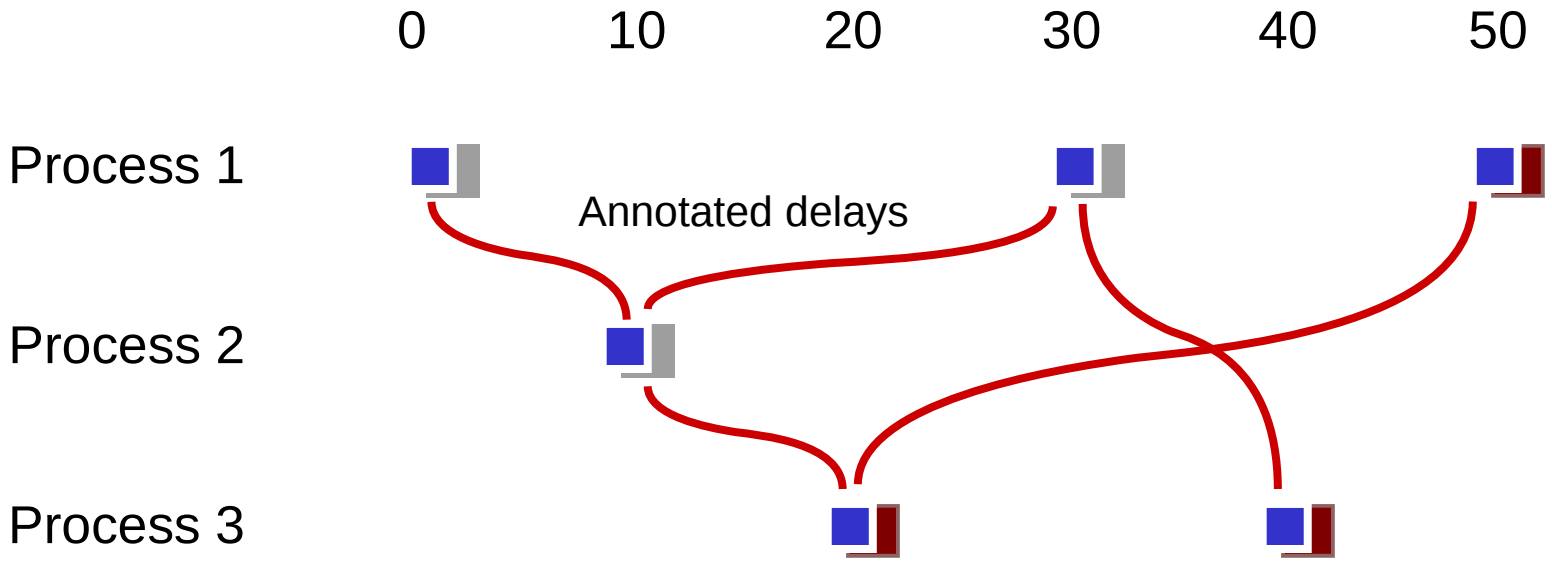


Loosely-timed



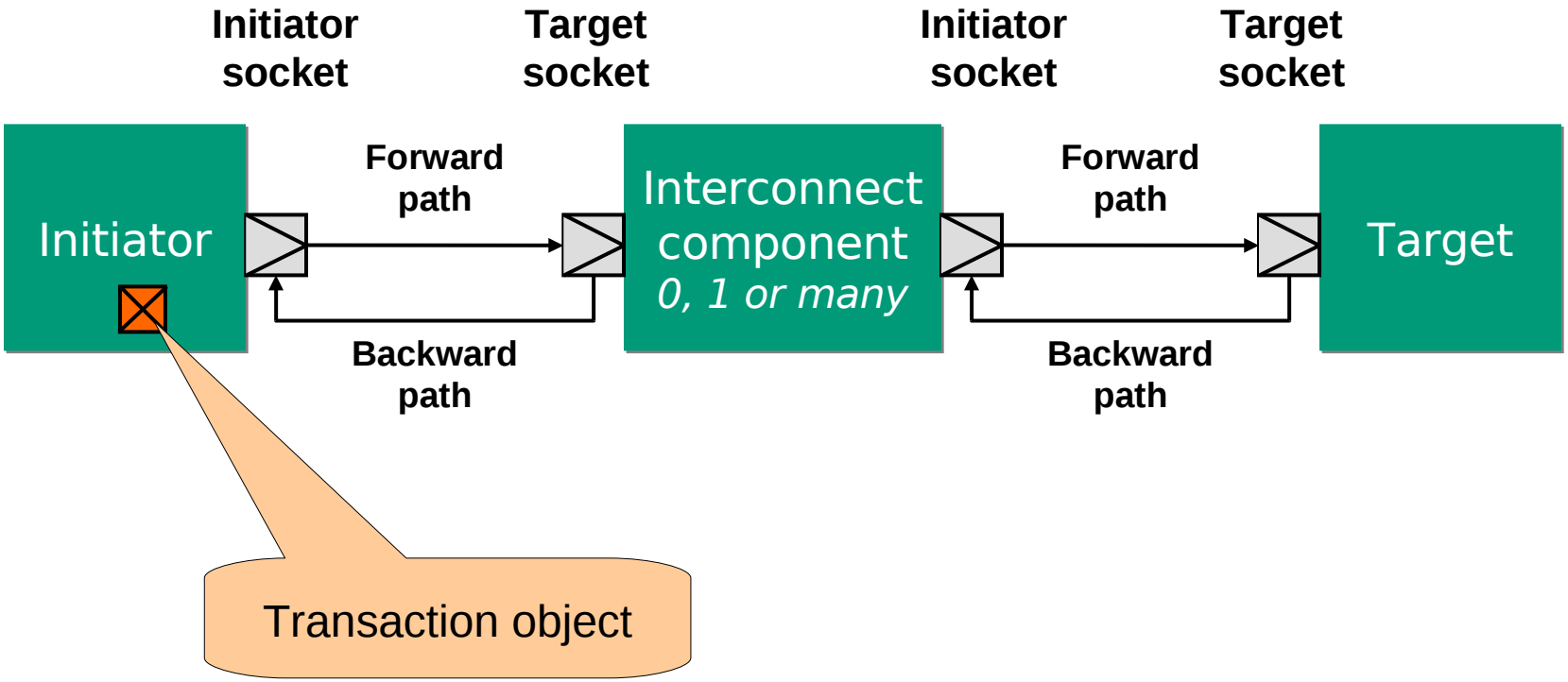
Each process runs ahead up to quantum boundary
sc_time_stamp() advances in multiples of the quantum

Approximately-timed

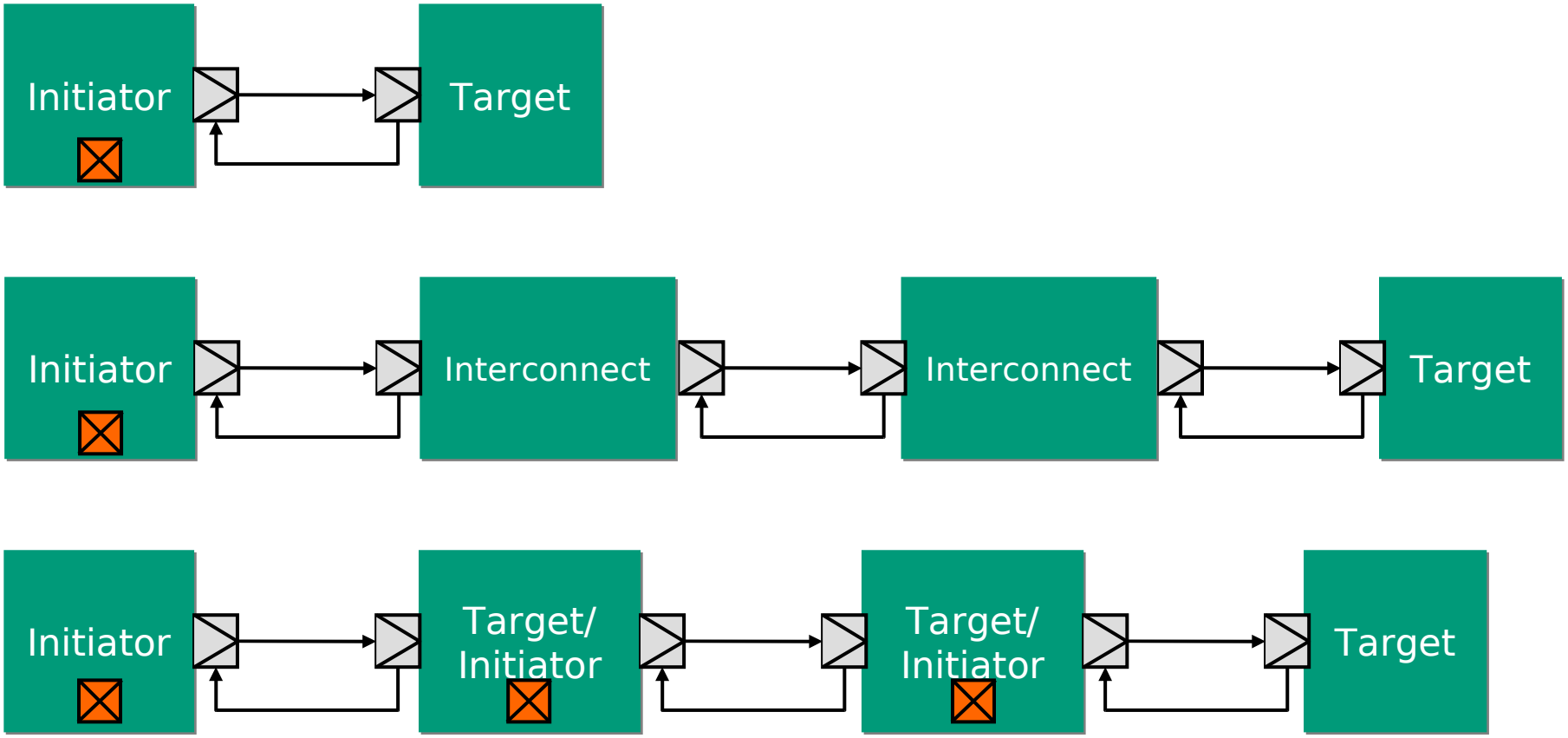


Each process is synchronized with SystemC scheduler
Delays can be accurate or approximate

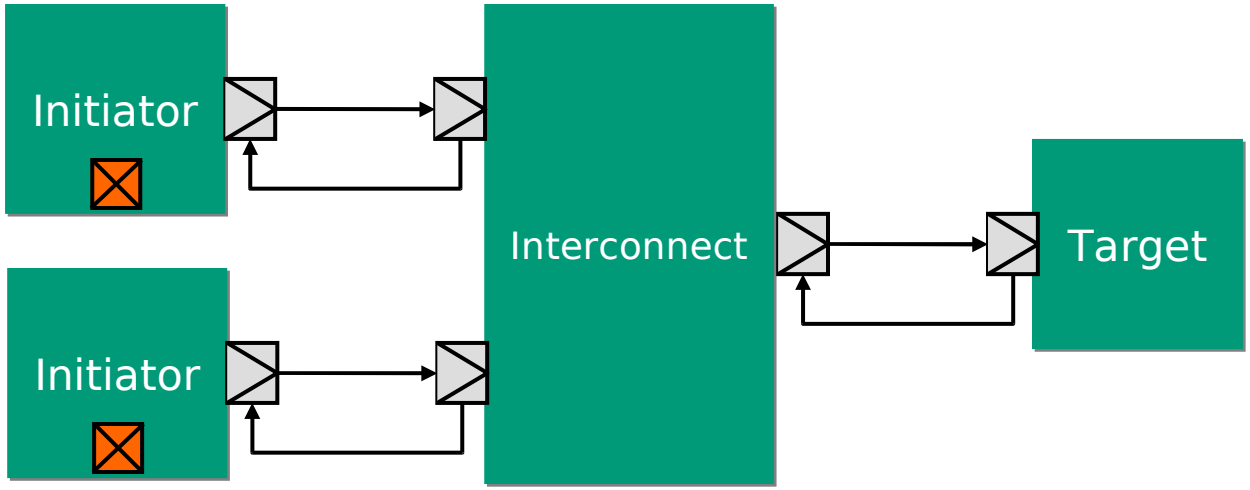
Initiators and Targets



TLM-2 Connectivity



Convergent Paths



TLM-2 core interfaces

Interface

- **Transport interfaces**
  - primary interfaces used to transport transactions between initiators, targets and interconnect components
- **Direct memory interface**
  - Direct access to memory area owned by Target
- **Debug transport interface**
  - Debug access to memory area owned by Target over the same forward path

Transport interfaces

Blocking transport interface

- Support timing annotation, & temporal decoupling
- Loosely-timed coding style
- Forward path only

Non-blocking transport interface

- Support timing annotation, temporal decoupling, & transaction phases
- Approximately-timed coding style
- Called on forward and backward paths

- Both share the same transaction type ---> **Interoperability**
- Unified interface and sockets ---> **can be mixed**

Blocking transport

tlm_blocking_transport_if

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

Transaction type

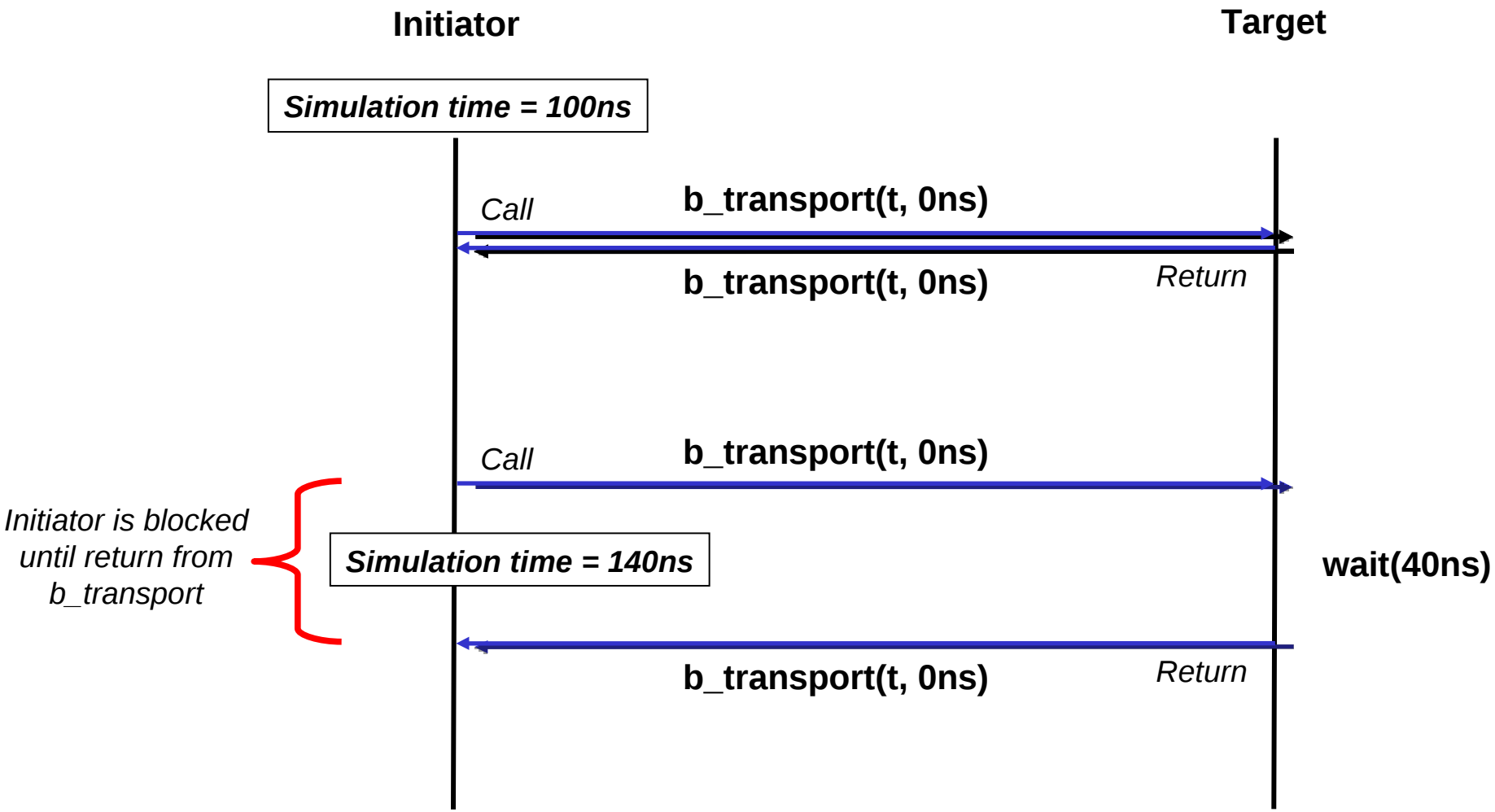
```
template <typename TRANS = tlm_generic_payload >

class tlm_blocking_transport_if : public virtual sc_core::sc_interface
{
public:
   virtual void b_transport (TRANS& trans , sc_core::sc_time& t) = 0;
};
```

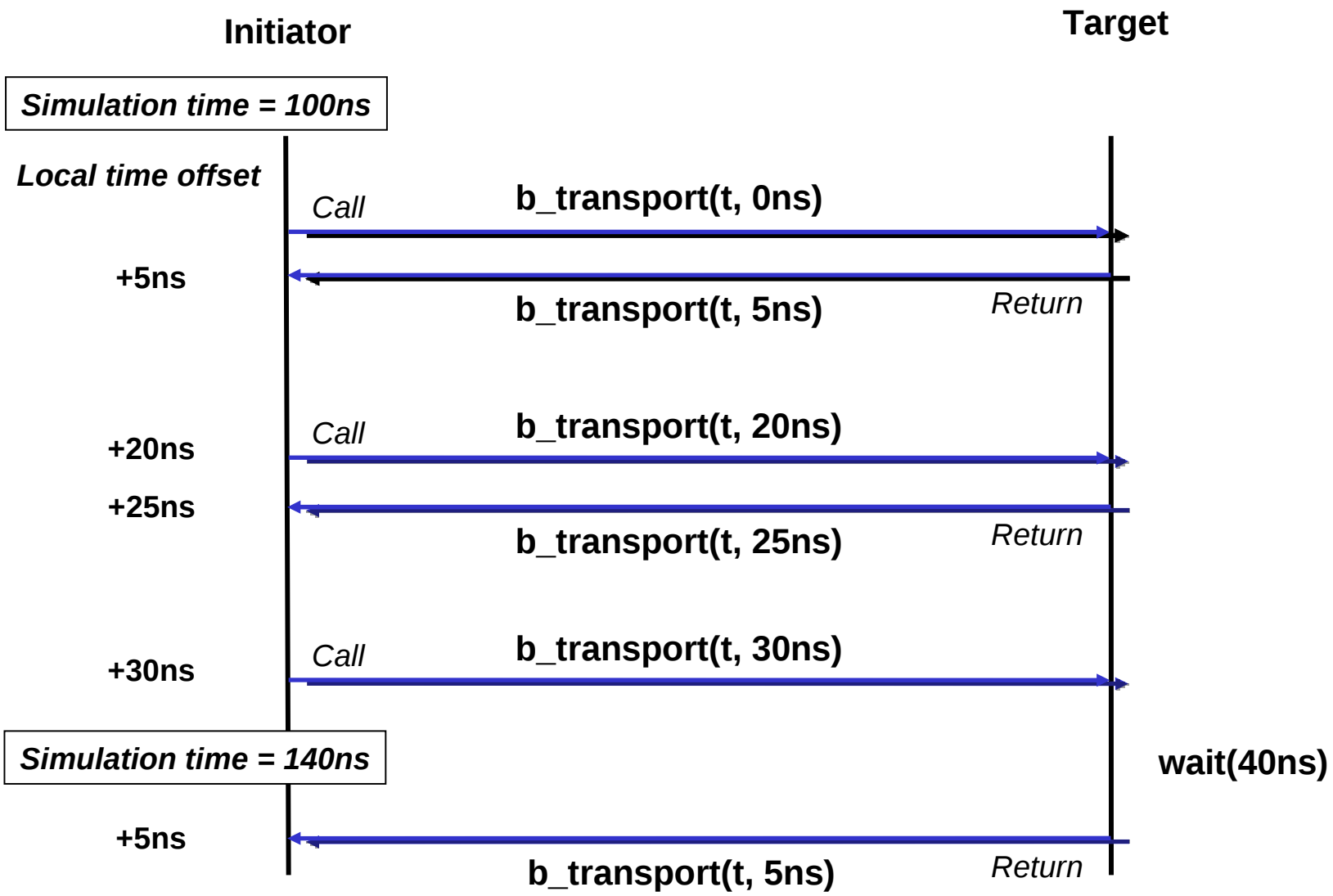
Transaction object

Timing annotation

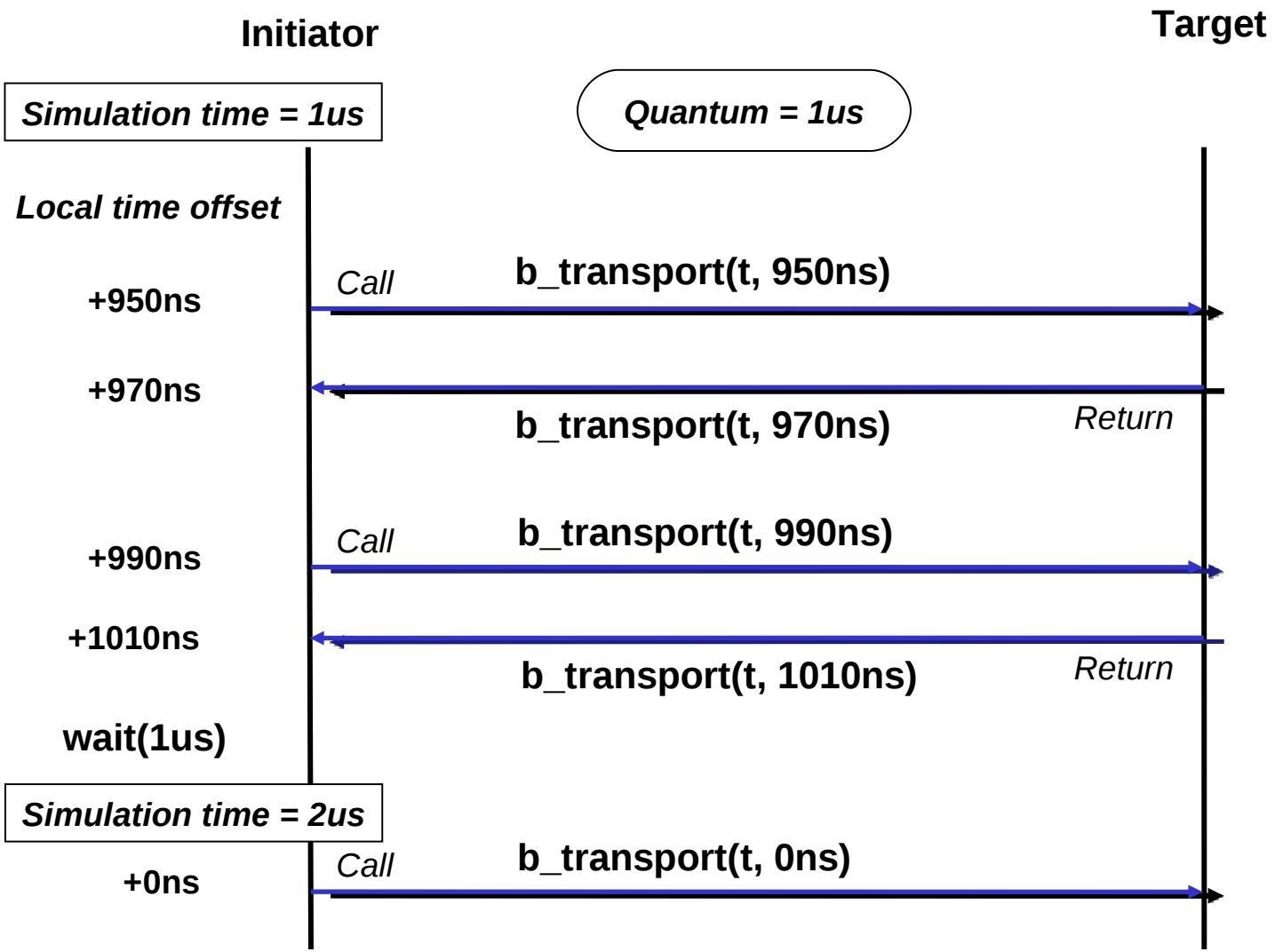
Blocking transport



Blocking transport - Temporal Decoupling



Blocking transport - Time Quantum



Non-blocking transport

tlm_fw_nonblocking_transport_if

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

tlm_bw_nonblocking_transport_if

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

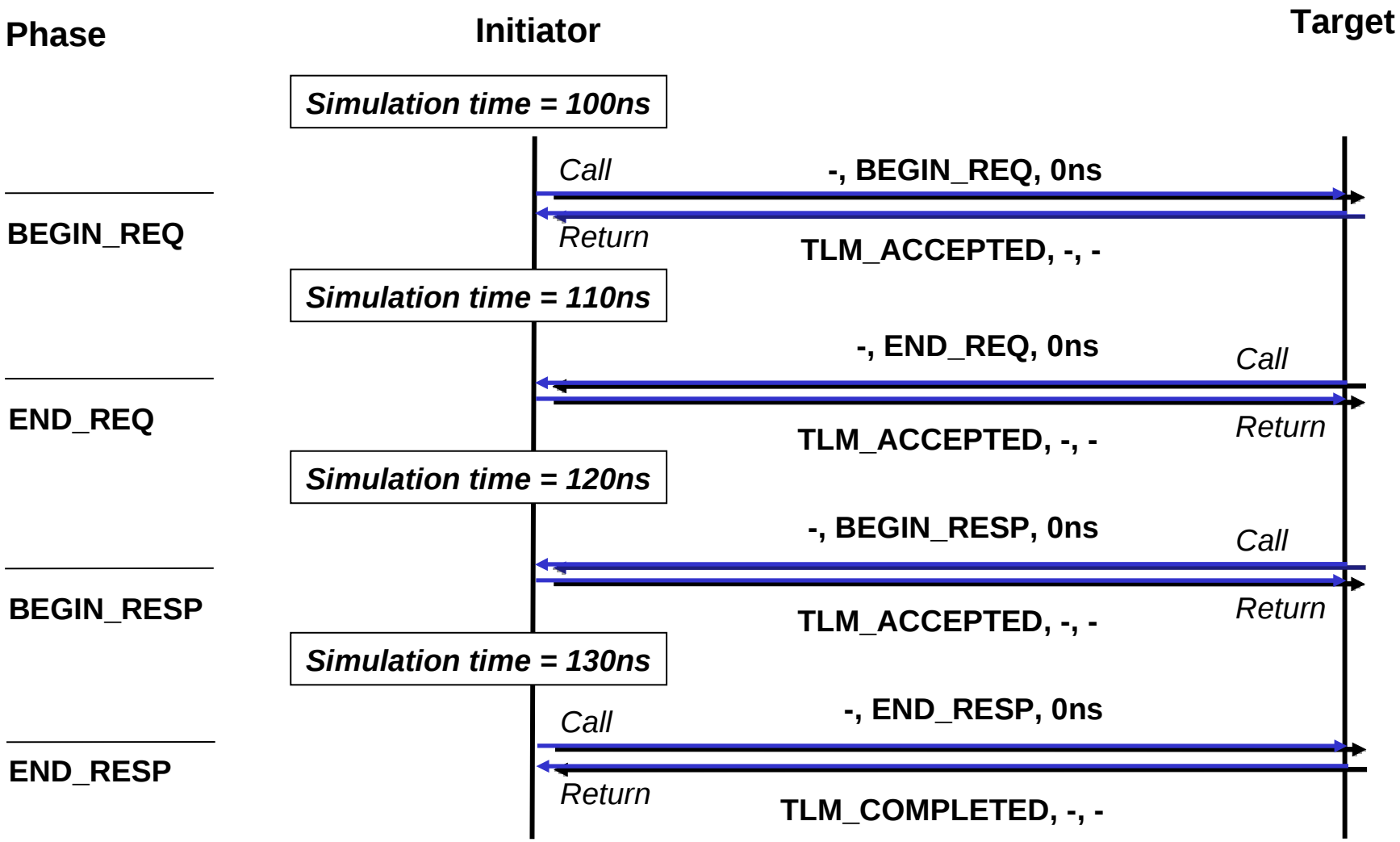
```
enum tlm_sync_enum { TLM_ACCEPTED, TLM_UPDATED, TLM_COMPLETED};
template < typename TRANS = tlm_generic_payload,
           typename PHASE = tlm_phase>

class tlm_fw_nonblocking_transport_if : public virtual sc_core::sc_interface {
public:
    virtual tlm_sync_enum nb_transport(TRANS& trans, PHASE& phase,
                                       sc_core::sc_time& t) = 0;
};
```

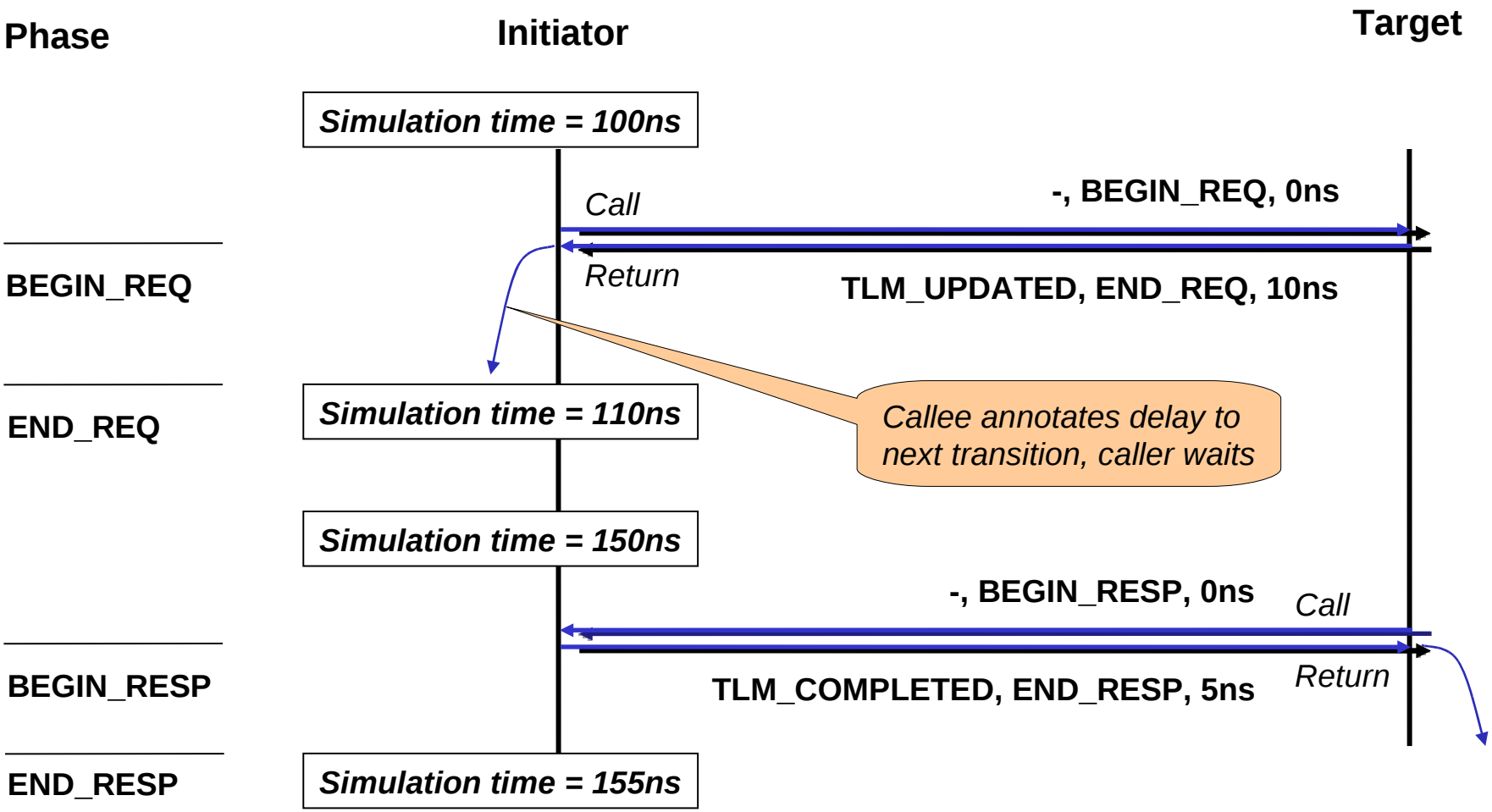
Non-blocking transport

tlm_sync_enum	Transaction object	Phase on return	Timing annotation on return
TLM_ACCEPTED	Unmodified	Unchanged	Unchanged
TLM_UPDATED	Updated	Changed	May be increased
TLM_COMPLETED	Updated	Ignored	May be increased

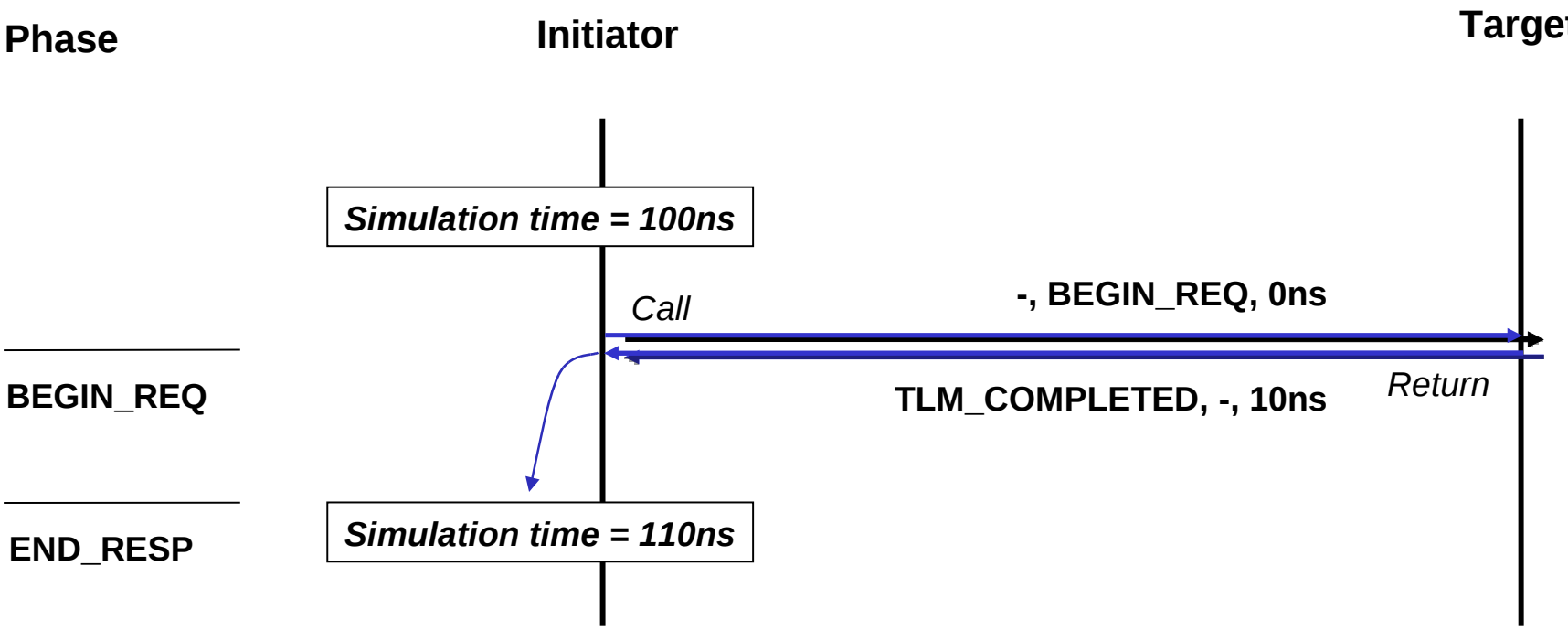
Non-blocking transport - Using Backward Path



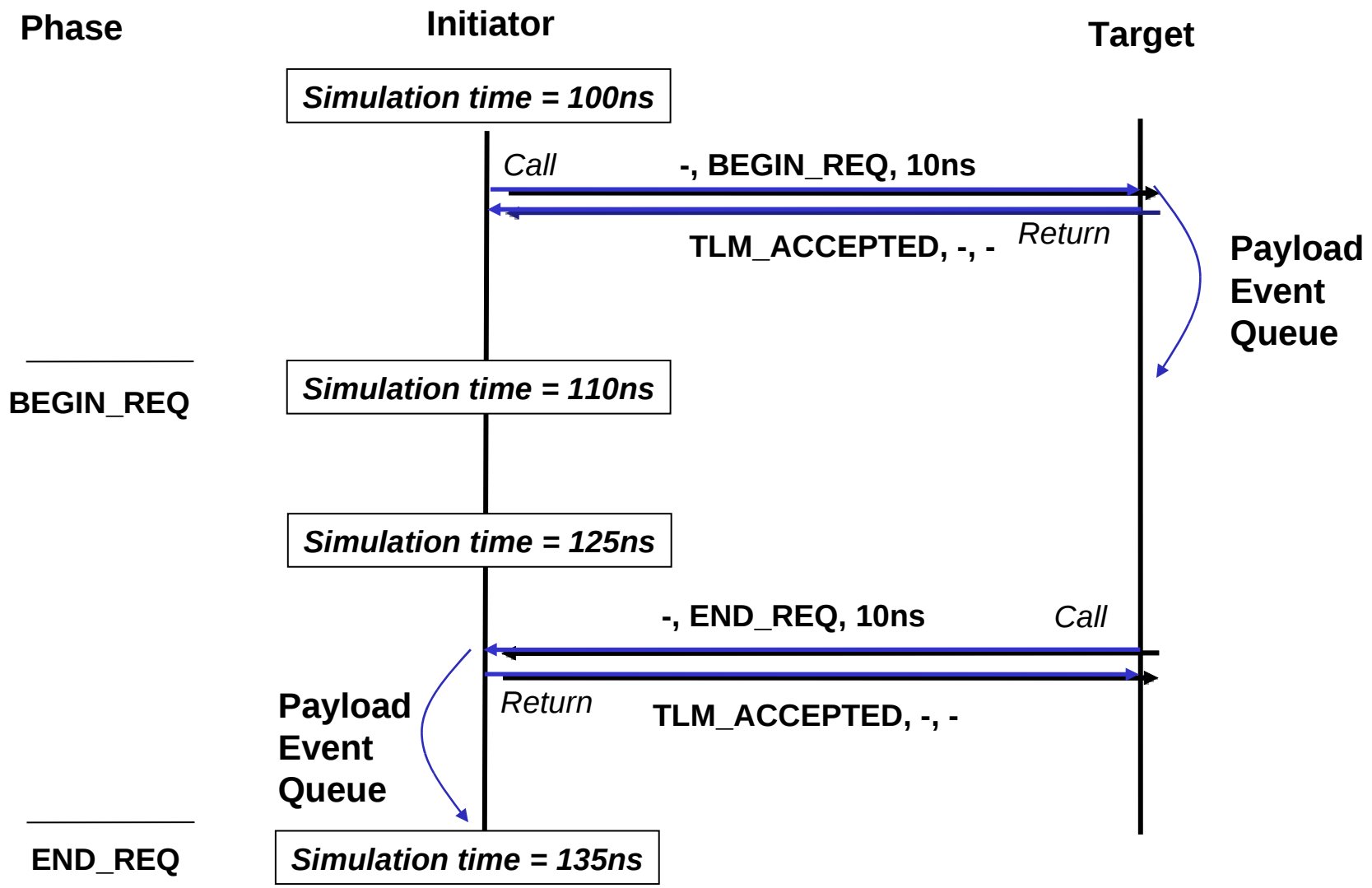
Non-blocking transport - Using Return Path



Non-blocking transport - Early Completion

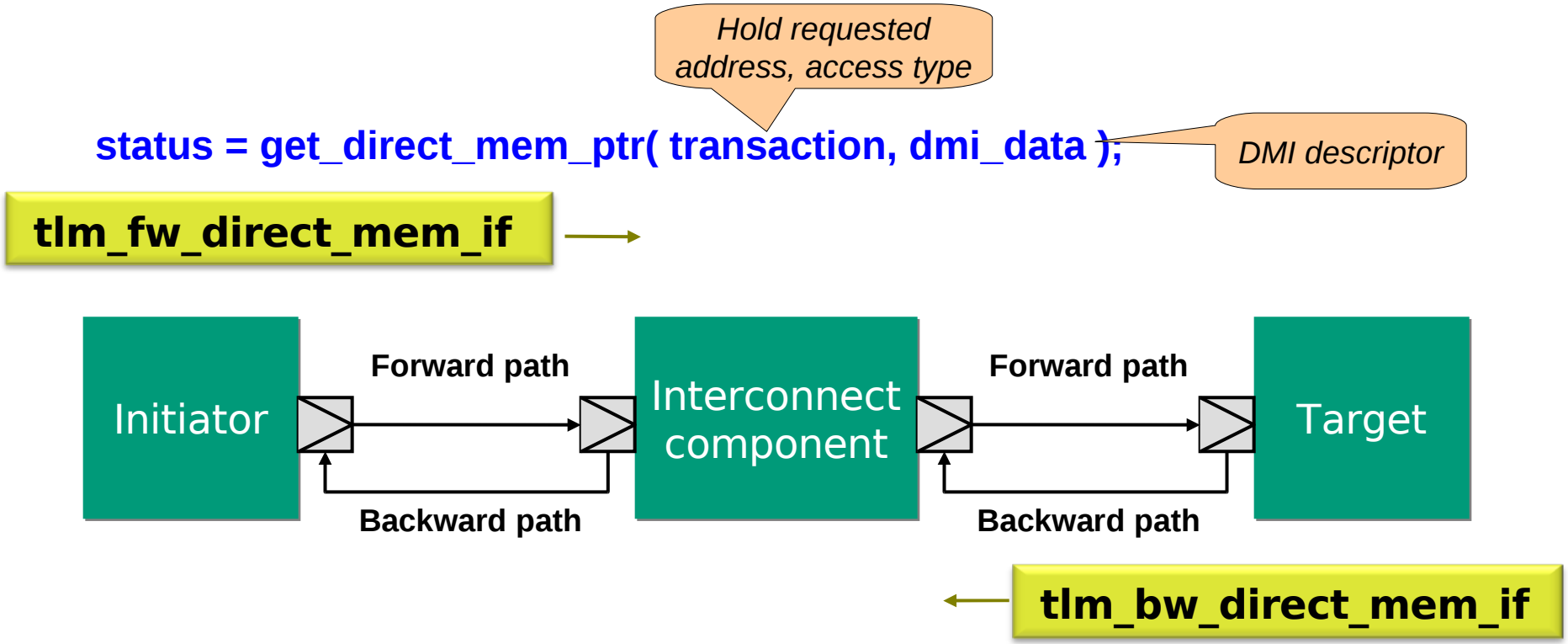


Non-blocking transport - Timing Annotation



Direct memory interface

- Access memory owned by target by using a direct pointer rather than through the transport interface
  - Increase in simulation speed for memory access



Direct memory interface

DMI Transaction

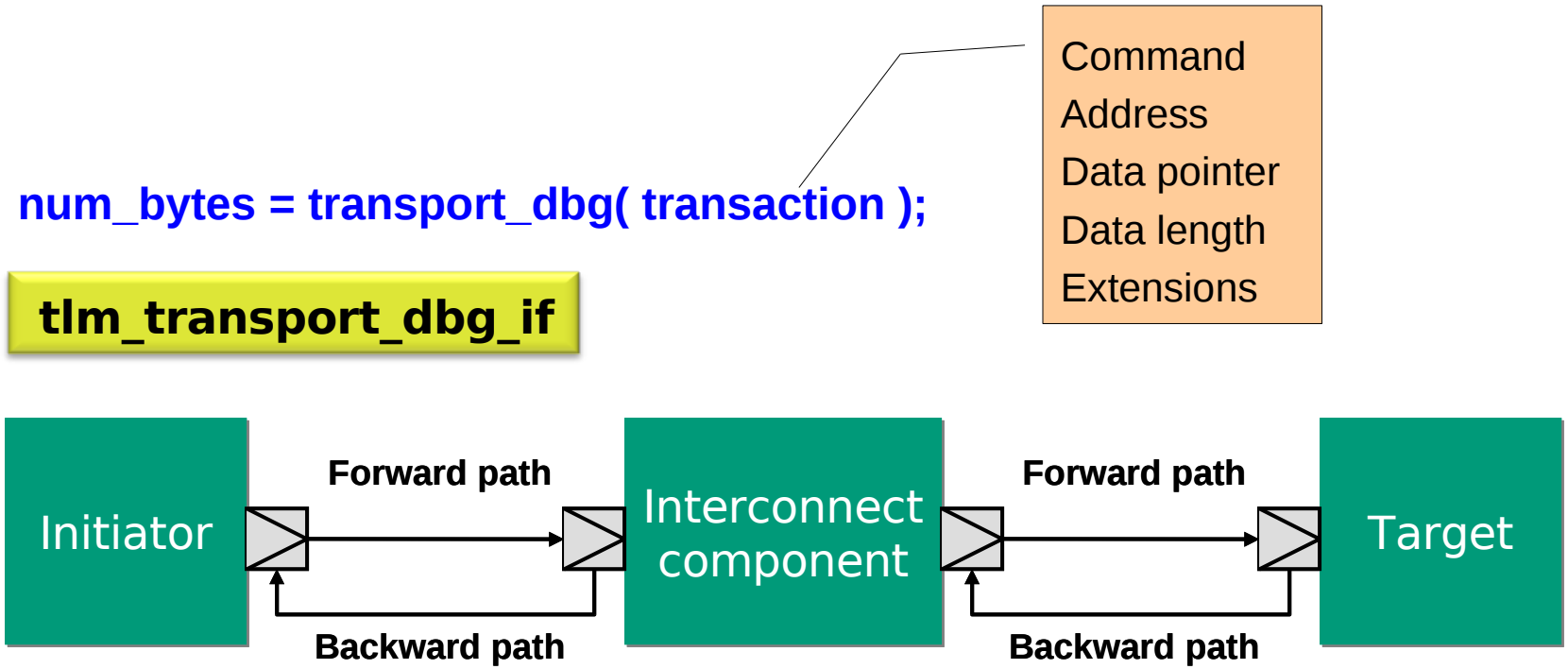
Requests read or write access
For a given address
Permits extensions

class `tlm_dmi`

unsigned char*	dmi_ptr	*Direct memory pointer*
uint64	dmi_start_address	*Region granted for given access type*
uint64	dmi_end_address	
dmi_type_e	dmi_type;	*Read, write or read/write*
sc_time	read_latency	*Latencies to be observed by initiator*
sc_time	write_latency	

Debug transport interface

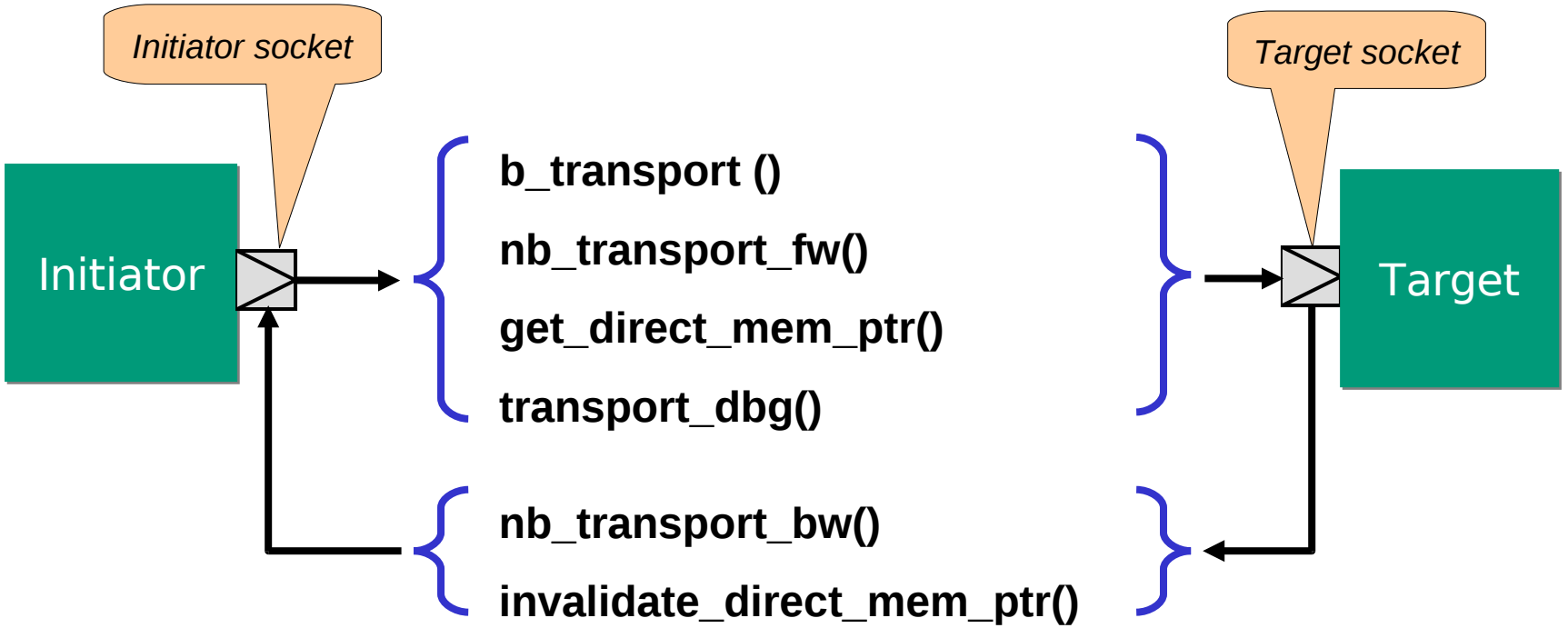
- Provide **non-intrusive access** to storage in Target using Transport interfarce



Sockets

Sockets

- Combines a **port** with an **export**
  - **Initiator socket**: port for the forward path and export for the backward path
  - **Target socket**: export for the forward path and port for the backward path



Convenience sockets

- Derived from the classes **tlm_initiator_socket** and **tlm_target_socket**
- Add functionalities ---> Write component models easier

- **Simple sockets**:
  - Not need to bind sockets to objects
  - Register callback methods with each socket
- **Passthrough sockets**:
  - Not support conversion between blocking and non-blocking calls
- **Tagged simple sockets**:
  - Tag incoming interface method calls with an ID
- **Multi-sockets**
  - Can bind to multiple sockets on other components
  - Use multi-port index number as the tag

Socket Summary

class	Register callbacks?	Multi-ports?	b <-> nb conversion?	Tagged ?
tlm_initiator_socket	no	yes	-	no
tlm_target_socket	no	yes	no	no
simple_initiator_socket	yes	no	-	no
simple_initiator_socket_tagged	yes	no	-	yes
simple_target_socket	yes	no	yes	no
simple_target_socket_tagged	yes	no	yes	yes
passthrough_target_socket	yes	no	no	no
passthrough_target_socket_tagged	yes	no	no	yes
multi_passthrough_initiator_socket	yes	yes	-	yes
multi_passthrough_target_socket	yes	yes	no	yes

Generic payload

Introduction

- Typical attributes for memory-mapped busses
  - Command, address, data, byte enables, single word transfers, burst transfers, streaming, response status
- Off-the-shelf general purpose payload
  - For abstract bus modeling
  - Ignorable extensions allow full interoperability
- Model specific bus protocols
  - Low implementation cost when bridging protocols
- Recommend using with Initial & Target sockets
- Usable with all common interfaces

Generic Payload Attributes

Attribute	Type	Modifiable?	
Command	tlm_command	No	
Address	uint64	Interconnect only	
Data pointer	unsigned char*	Yes (read cmd)	*Array owned by initiator*
Data length	unsigned int	No	
Byte enable pointer	unsigned char*	No	*Array owned by initiator*
Byte enable length	unsigned int	No	
Streaming width	unsigned int	No	
DMI hint	bool	Yes	
Response status	tlm_response_status	Target only	
Extensions	(tlm_extension_base*)[ ]	Yes	*Extension mechanism*

Command, Address and Data

```
enum tlm_command {
    TLM_READ_COMMAND,
    TLM_WRITE_COMMAND,
    TLM_IGNORE_COMMAND
};
```

Copy from target to data array
Copy from data array to target
Neither, but may use extensions

```
tlm_command    get_command() const ;
void           set_command(const tlm_command command);

sc_dt::uint64  get_address()  const;
void           set_address(const sc_dt::uint64 address);

unsigned char* get_data_ptr()  const;
void           set_data_ptr(unsigned char* data);

unsigned int   get_data_length()  const;
void           set_data_length(const unsigned int length);
```

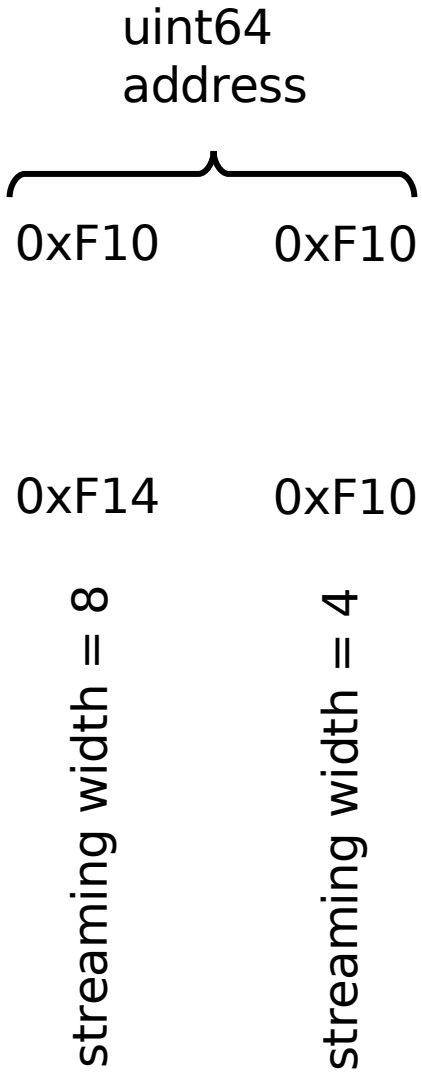
Data array owned by initiator

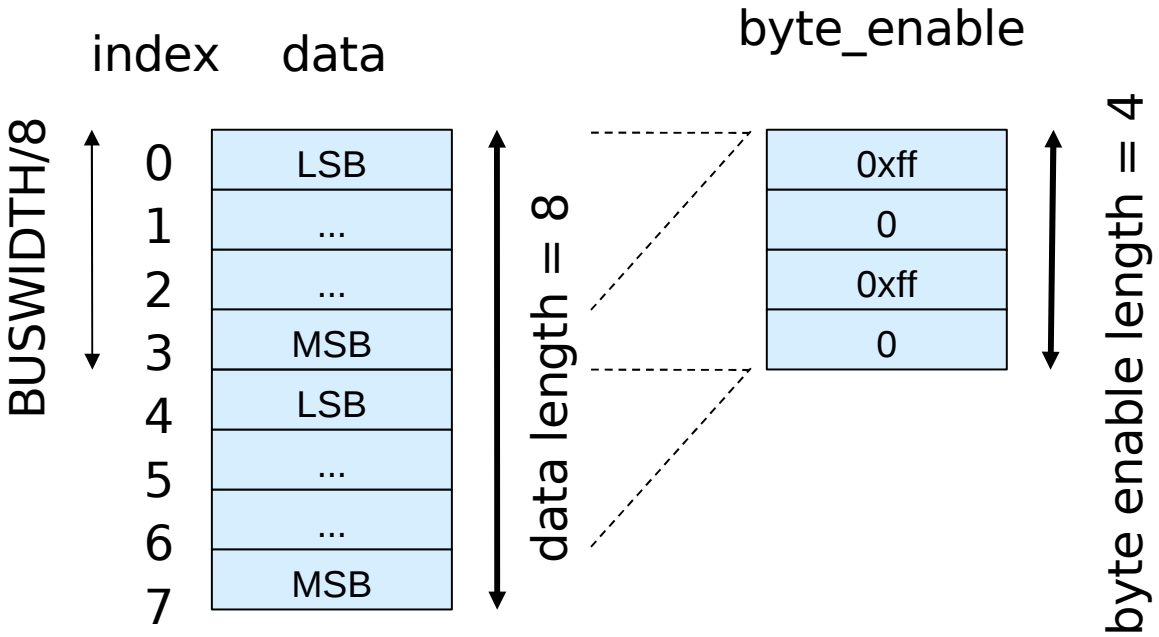
Number of bytes in data array

Response Status

enum tlm_response_status	Meaning
TLM_OK_RESPONSE	Successful
TLM_INCOMPLETE_RESPONSE	Transaction not delivered to target. (Default)
TLM_ADDRESS_ERROR_RESPONSE	Unable to act upon address
TLM_COMMAND_ERROR_RESPONSE	Unable to execute command
TLM_BURST_ERROR_RESPONSE	Unable to act upon data length or streaming width
TLM_BYTE_ENABLE_ERROR_RESPONSE	Unable to act upon byte enable
TLM_GENERIC_ERROR_RESPONSE	Any other error

Byte Enables and Streaming





1-enable-per-byte
Byte enables applied repeatedly
Data interpreted using BUSWIDTH
Streaming width > 0 => wrap address

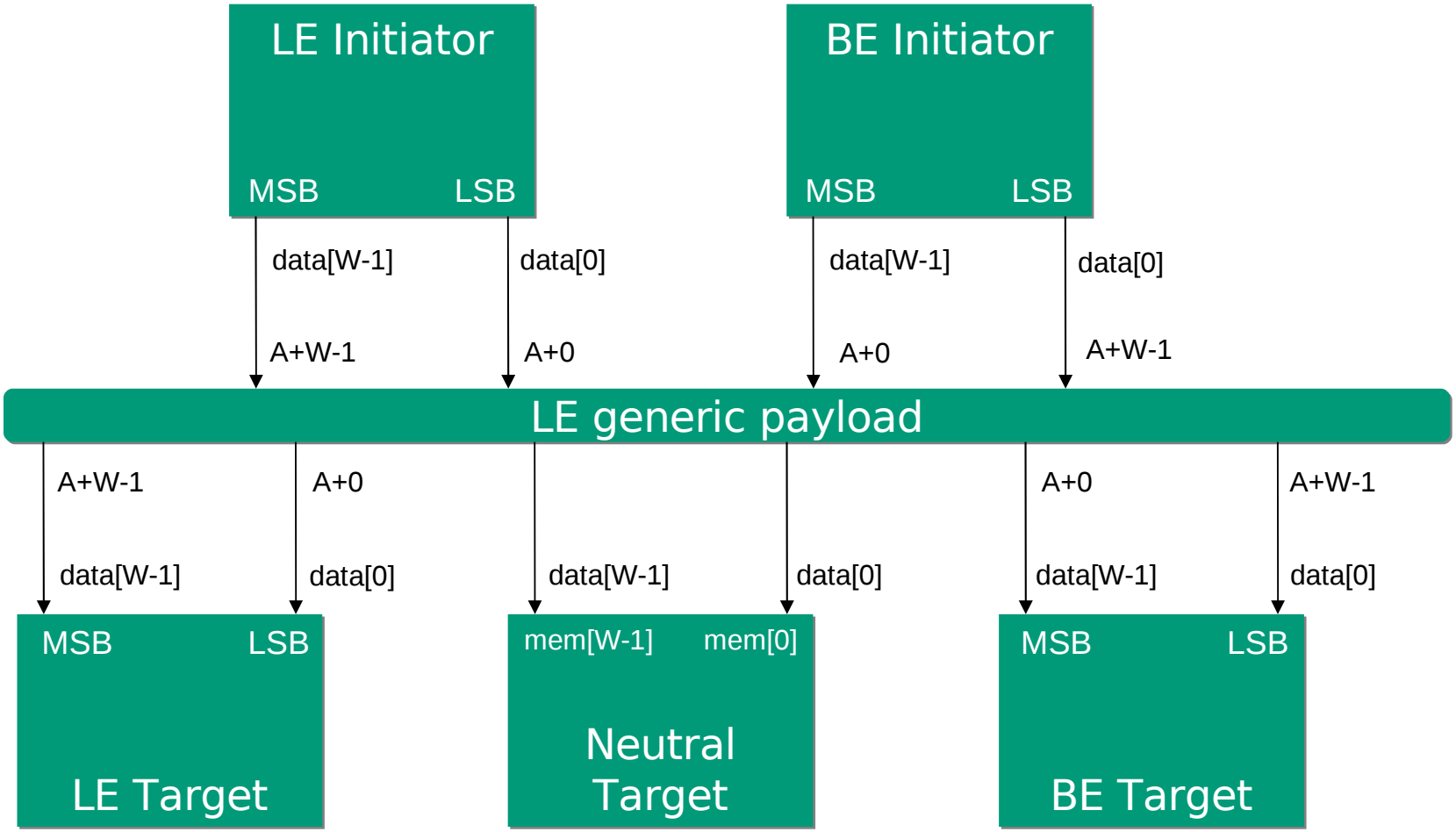
```
#define TLM_BYTE_DISABLED 0x0
#define TLM_BYTE_ENABLED 0xff
```

Endianness

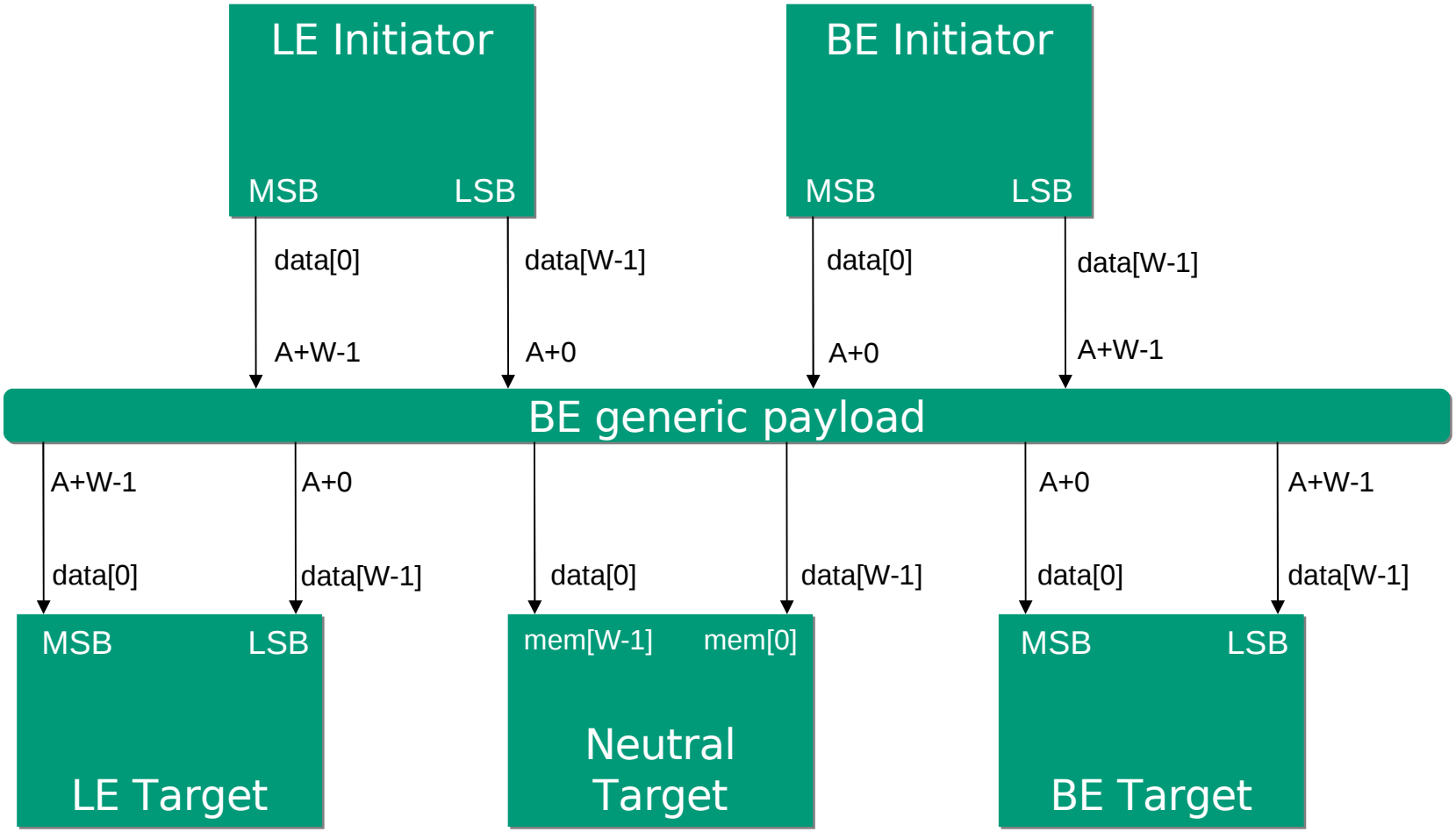
- Endianness of host machine may differ from models

- TLM' rule ensure interoperability between initiators and targets using the generic payload:
  - Words in data array are host-endian
  - Initiators and targets connected LSB-to-LSB, MSB-to-MSB
  - Most efficient when everything is modeled host-endian
  - Width-conversions with same endianness as host are free

Little-endian host



Big-endian host



Phases and base protocol

Phases

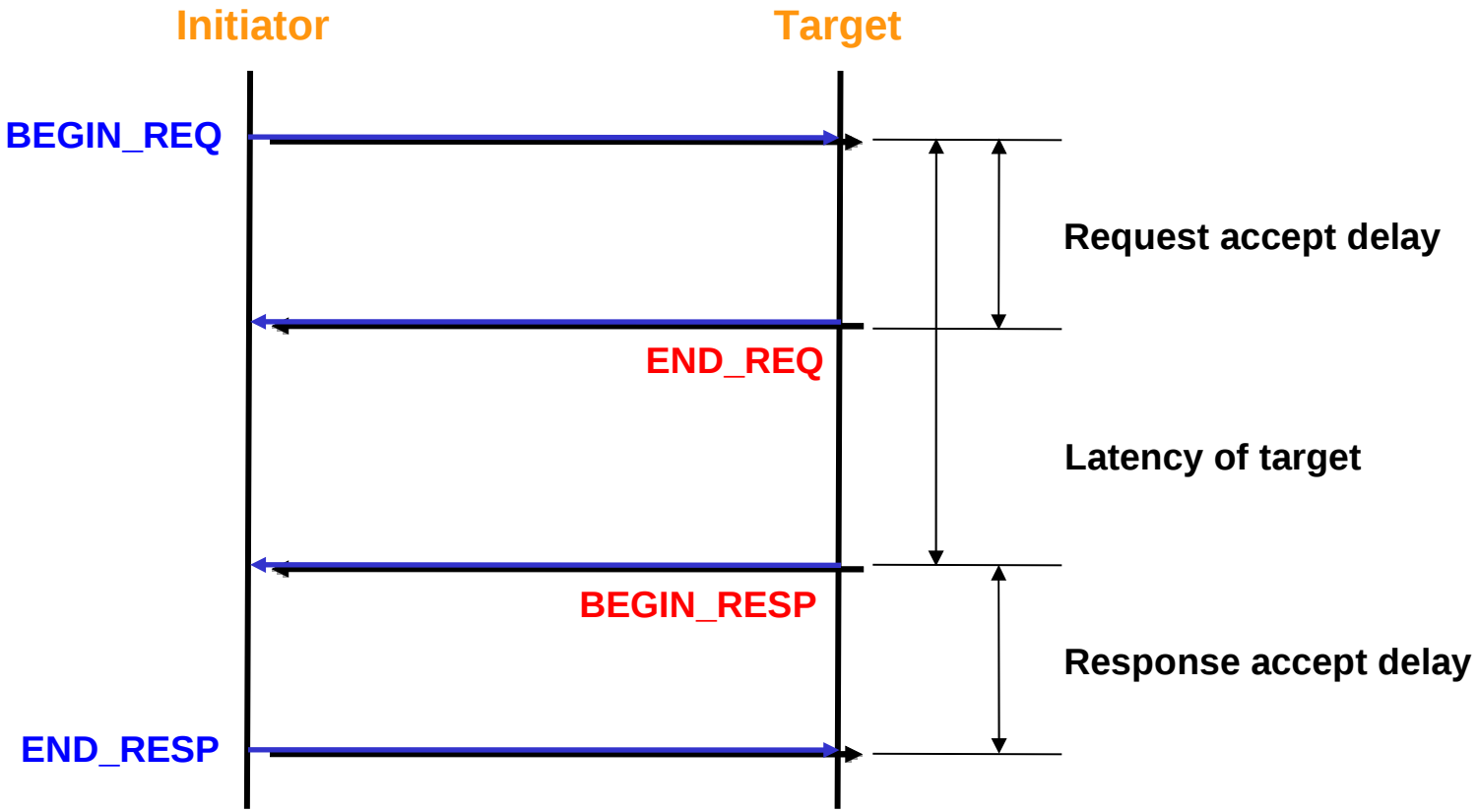
- Blocking Transport Interface:
  - 2 timing point (beginning of request phase, beginning of response phase)
- Non-blocking Transport Interface:
  - Multiple phase (Base Protocol: BEGIN_REQ, END_REQ, BEGIN_RESP, END_RESP)
  - Can be extended to support a specific protocol
  - Ignorable phase & mandatory phase

Base Protocol

- Define rules for phases and call order
- Ensure maximal interoperability between components

- Requirement:
  - TLM-2 core transport, direct memory & debug transport interfaces
  - tlm_initiator_socket & tlm_target_socket
  - tlm_generic_payload
  - tlm_phase

Base protocol phase sequences



- *BEGIN_REQ must wait for previous END_REQ*
- *BEGIN_RESP must wait for previous END_RESP*

Base protocol phase sequences

- Permitted phase transition sequences:
  - **BEGIN_REQ**(fw) (Tgt returns TLM_COMPLETED)
  - **BEGIN_REQ**(fw) → **END_REQ**(bw) (Int returns TLM_COMPLETED)
  - **BEGIN_REQ**(fw) → **BEGIN_RESP**(bw) (Int returns TLM_COMPLETED)
  - **BEGIN_REQ**(fw) → **END_REQ**(rtn/bw) → **BEGIN_RESP**(bw) (Int returns TLM_COMPLETED)
  - **BEGIN_REQ**(fw) → **BEGIN_RESP**(rtn/bw) → **END_RESP**(rtn/fw)
  - **BEGIN_REQ**(fw) → **END_REQ**(rtn/bw) → **BEGIN_RESP**(bw) → **END_RESP**(rtn/fw)

Appendix

Reference

- OSCI TLM-2.0 LANGUAGE REFERENCE MANUAL
  - /shsv/sld/Common/Lib/04_TLM/TLM-2009-07-15/docs/release/TLM_2_0_LRM.pdf

Thanks for your attention!

Renesas Design Vietnam Co., Ltd.