OSCI TLM-2.0

The Transaction Level Modeling standard of the Open SystemC Initiative (OSCI)

OSCI TLM-2.0

Software version: TLM-2.0.1
Document version: ja8
This presentation authored by: John Aynsley, Doulos

SWGMXPQ1624

CONTENTS

- Introduction
- Transport Interfaces
- DMI and Debug Interfaces
- Sockets
- The Generic Payload
- The Base Protocol
- Analysis Ports

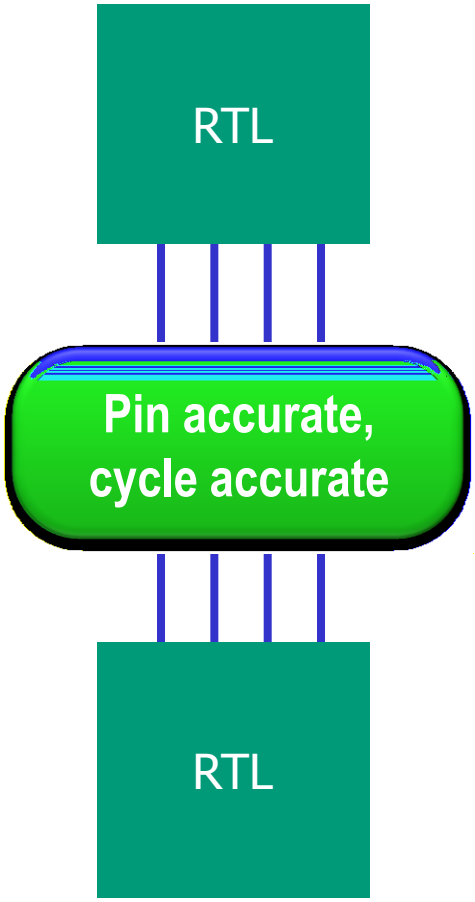
SYSTEMC™

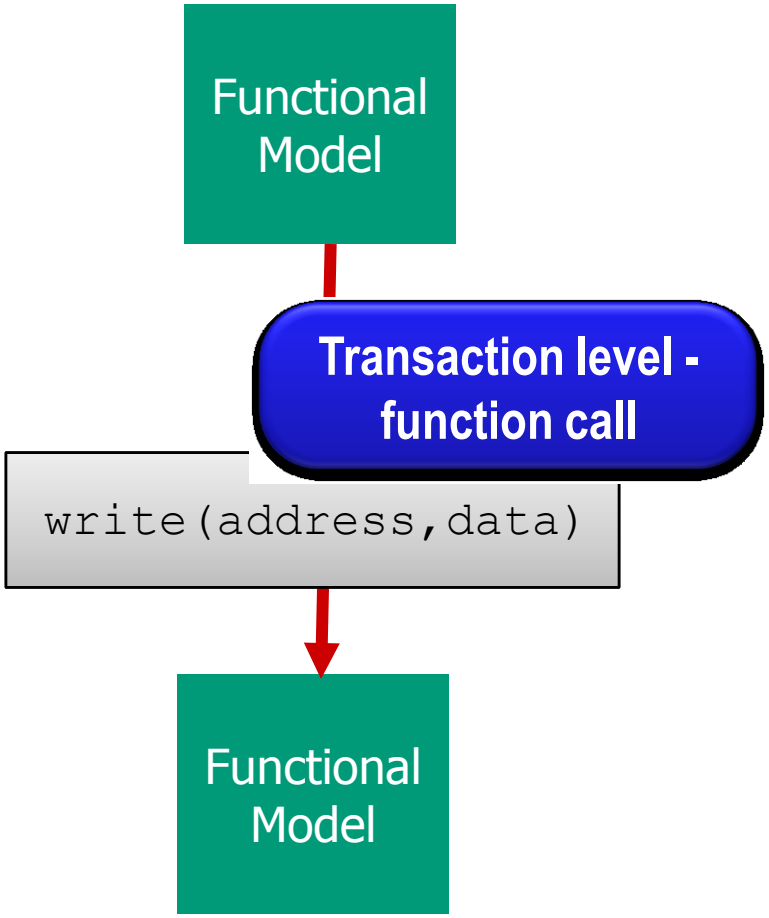
SWGMXPQ1621

INTRODUCTION

- Transaction Level Modeling 101
- OSCI TLM-1 and TLM-2
- Coding Styles
- Structure of the TLM-2.0.1 Kit

Transaction Level Modeling 101

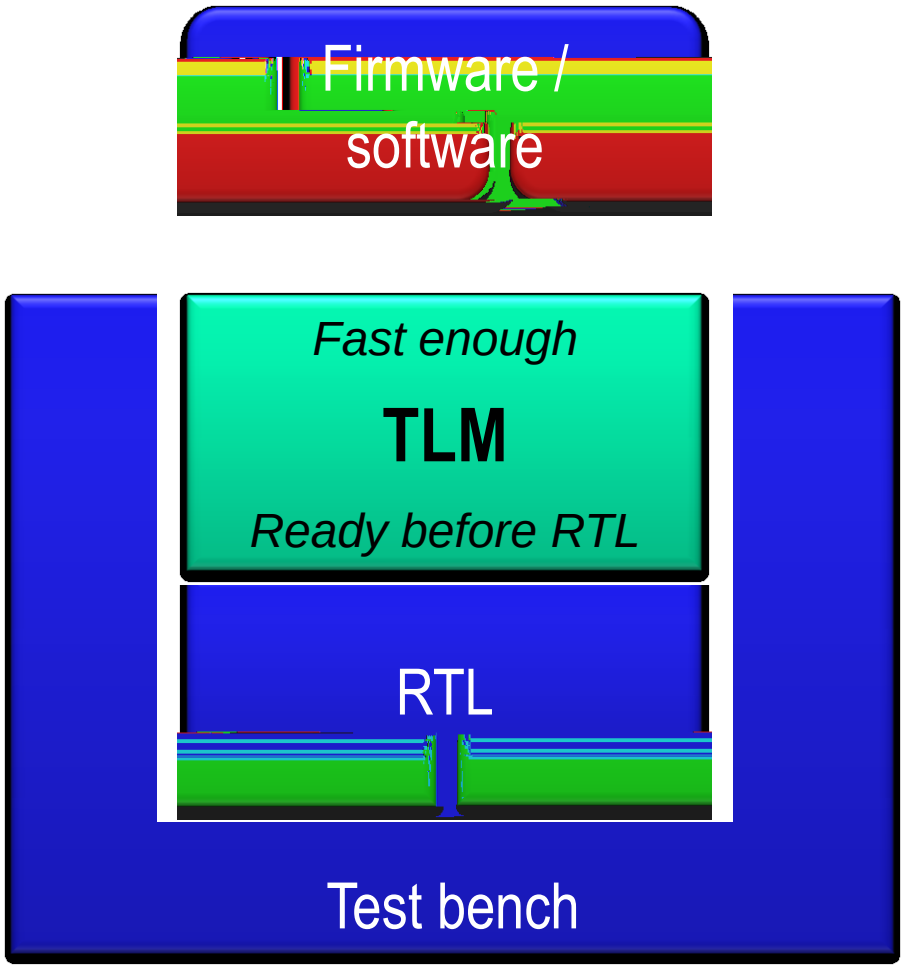




Wrq pe i i i v i i r

5441540444\ je i v rq pe nsr

Reasons for Using TLM



Accelerates product release schedule

Software development

Architectural modeling

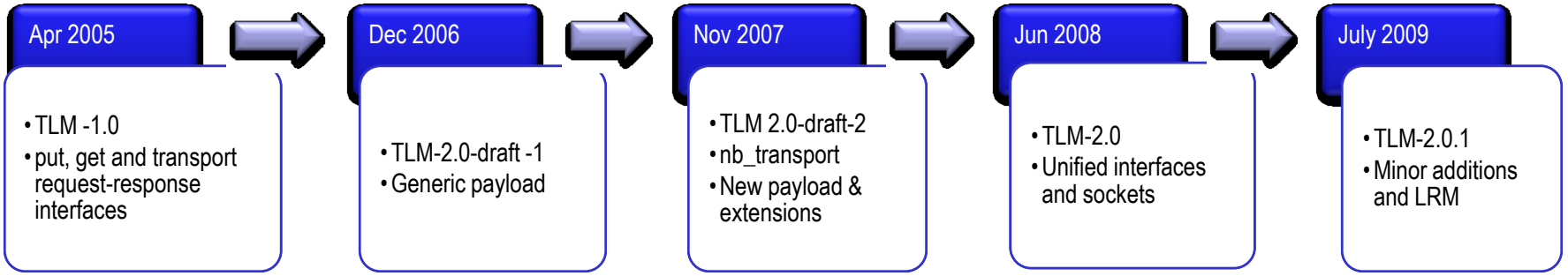
Hardware verification

TLM = golden model

Typical 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
- Available before RTL — Early!
- Simulates much faster than RTL — Fast!

OSCI TLM Development



SYSTEMC™

TLM-1.0 → TLM-2.0

- TLM-2.0 is the new standard for interoperability between memory mapped bus models
  - Incompatible with TLM-2.0-draft1 and TLM-2.0-draft2
- TLM-1.0 is not deprecated (put, get, nb_put, nb_get, transport)
- TLM-1.0 is included within TLM-2.0
  - Migration path from TLM-1.0 to TLM-2.0 (see examples)

TLM-2 Requirements

- Transaction-level memory-mapped bus modeling
- Register accurate, functionally complete
- Fast enough to boot software O/S in seconds — Speed
- Loosely-timed and approximately-timed modeling
- Interoperable API for memory-mapped bus modeling
- Generic payload and extension mechanism
- Avoid adapters where possible — Interoperability

See TLM_2_0_requirements.pdf

Use Cases, Coding Styles and Mechanisms

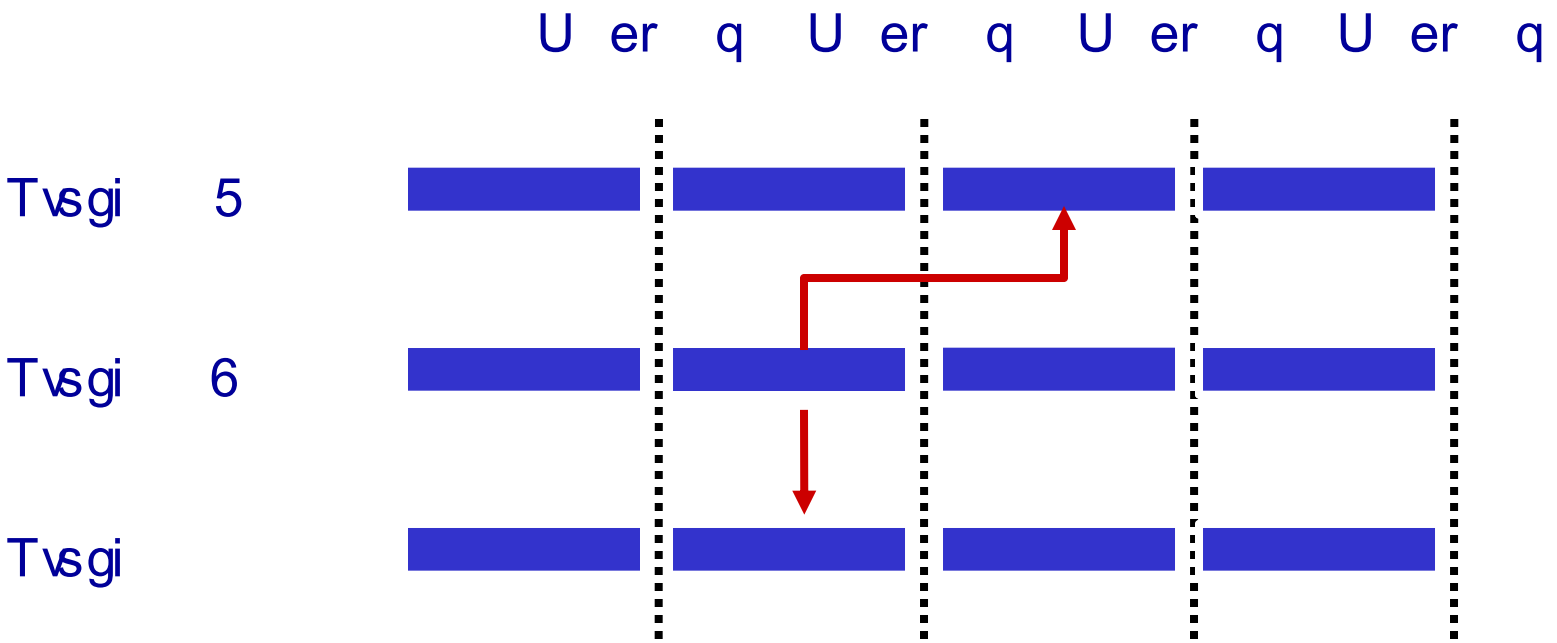
Blocking interface

SYSTEMC™

Coding Styles

- Loosely-timed
  - Only sufficient timing detail to boot O/S and run multi-core systems
  - Processes can run ahead of simulation time (temporal decoupling)
  - Each transaction has 2 timing points: *begin* and *end*
  - Uses direct memory interface (DMI)
- Approximately-timed
  - *aka* cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - Processes run in lock-step with simulation time
  - Each transaction has 4 timing points (extensible)
- Guidelines only – not definitive

Loosely-timed

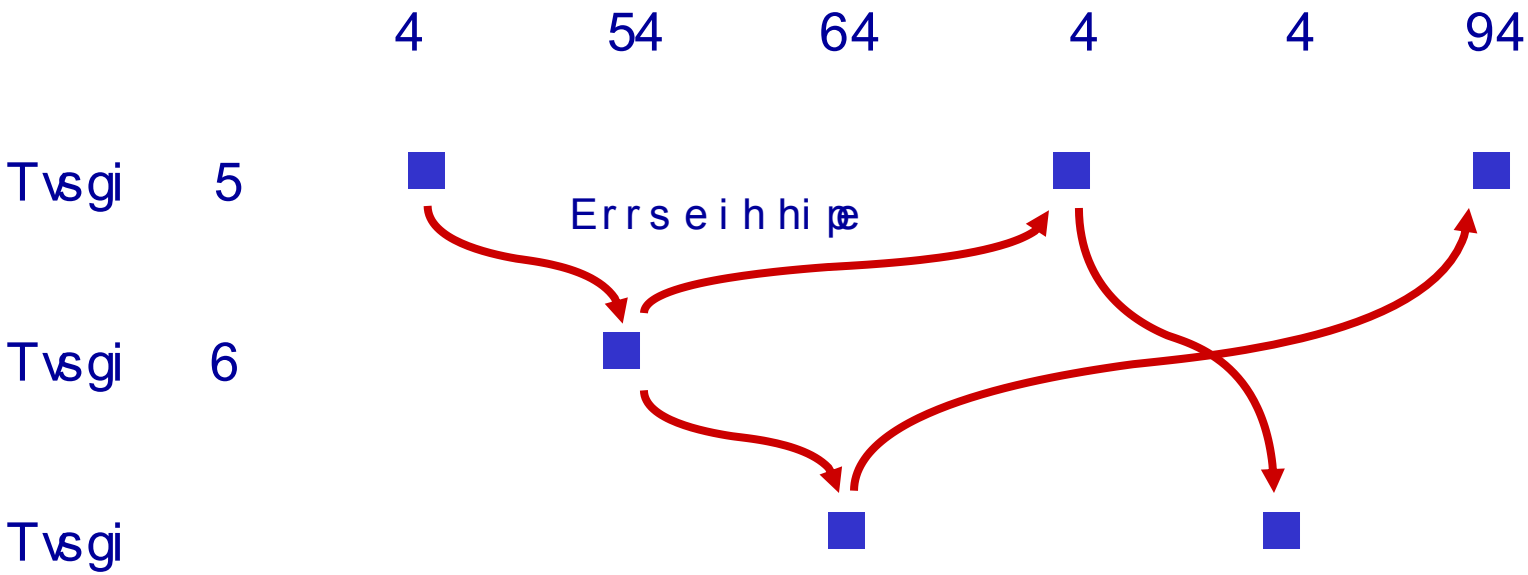


Each process runs ahead up to quantum boundary

sc_time_stamp() advances in multiples of the quantum

Deterministic communication requires explicit synchronization

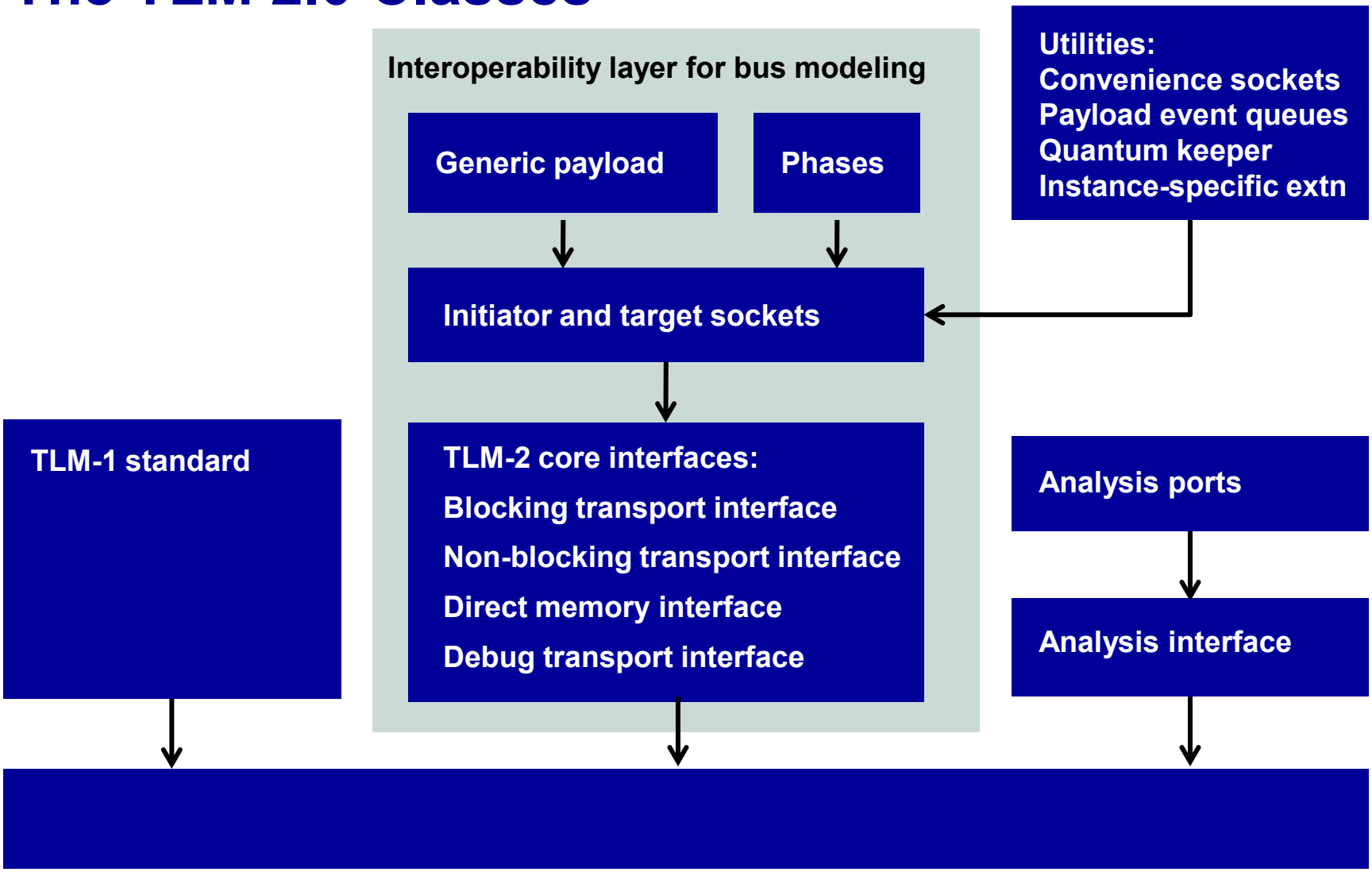
Approximately-timed



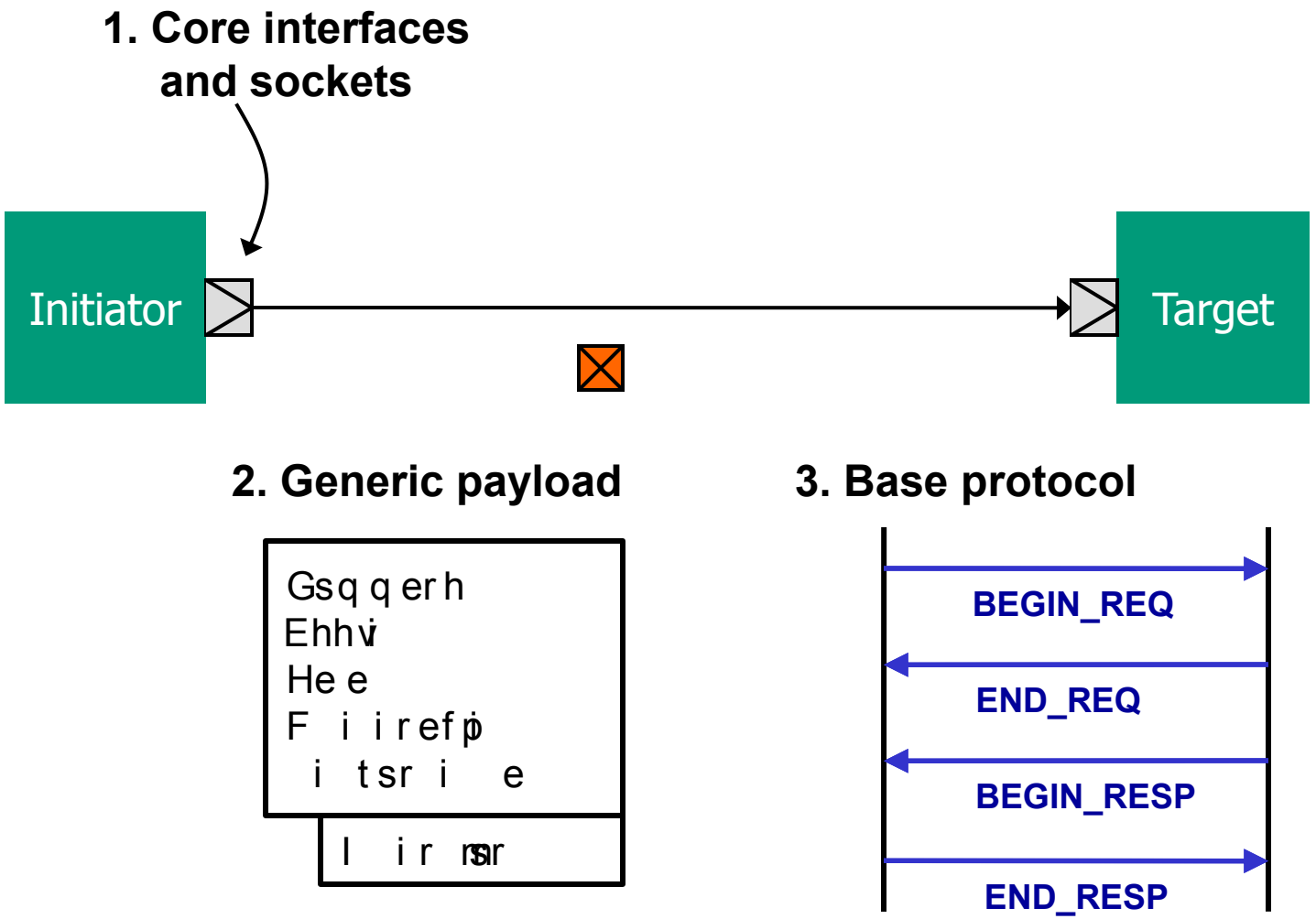
Each process is synchronized with SystemC scheduler

Delays can be accurate or approximate

The TLM 2.0 Classes



Interoperability Layer



Maximal interoperability for memory-mapped bus models

Utilities

- tlm_utils
  - Convenience sockets
  - Payload event queues
  - Quantum keeper
  - Instance-specific extensions

- Productivity
- Shortened learning curve
- Consistent coding style

- *Not* part of the interoperability layer – write your own?

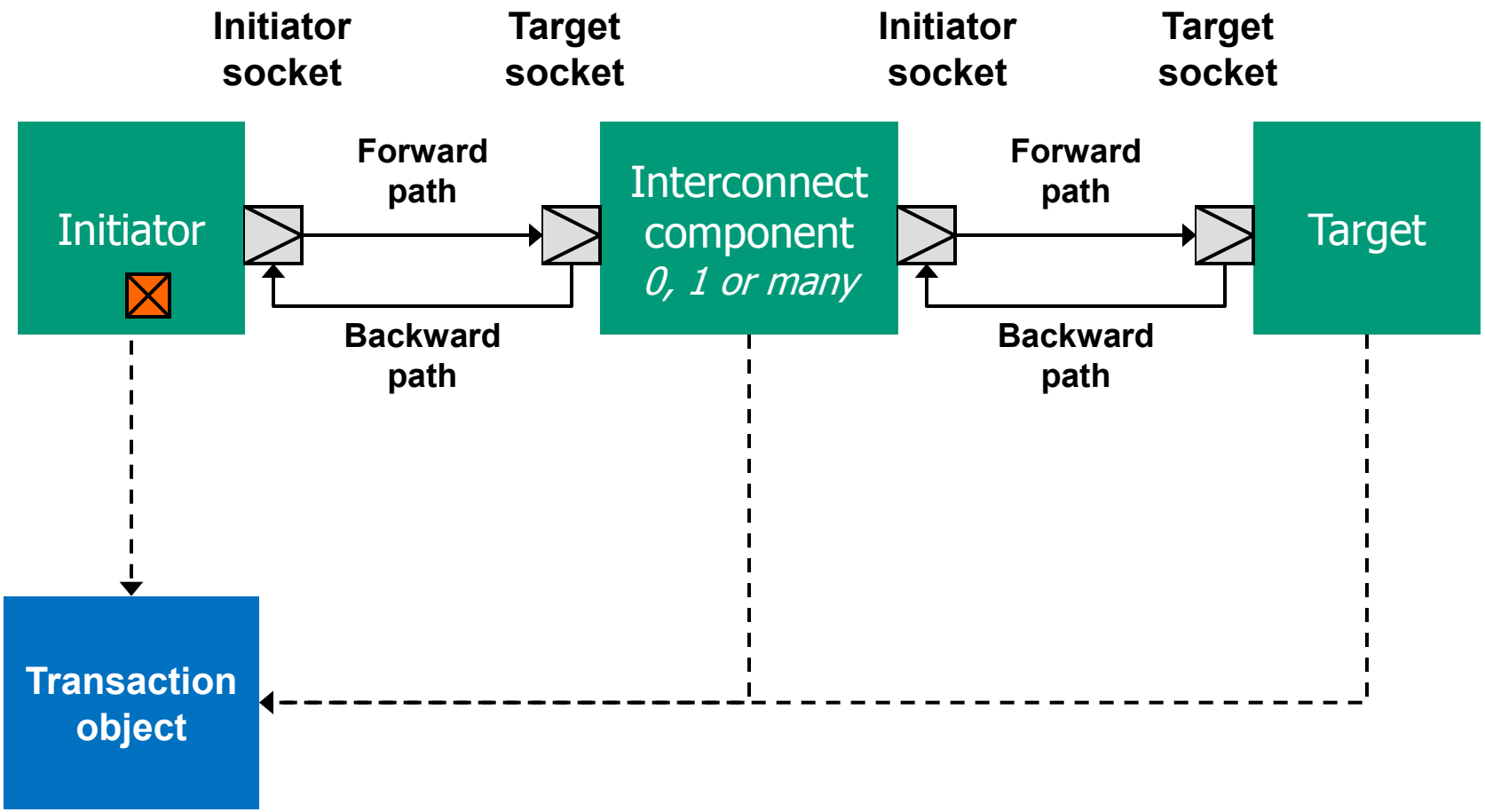
Directory Structure

```
mngphi3pq
    pqcl                          TLM-2 interoperability classes
        pqc6cmnivjegi             TLM-2 core interfaces
        pqckirivmgcte pseh        TLM-2 generic payload
        pqc sgoi                  TLM-2 initiator and target sockets
        pqcu er  q                TLM-2 global quantum
    pqc5
        pqcviucvt                 TLM-1.0 legacy
        pqcerep m                 Analysis interface, port, fifo
    pqc  mp                       TLM-2 utilities

hsg
   hs  kir
i  eqtpo
 rmci
```


TRANSPORT INTERFACES

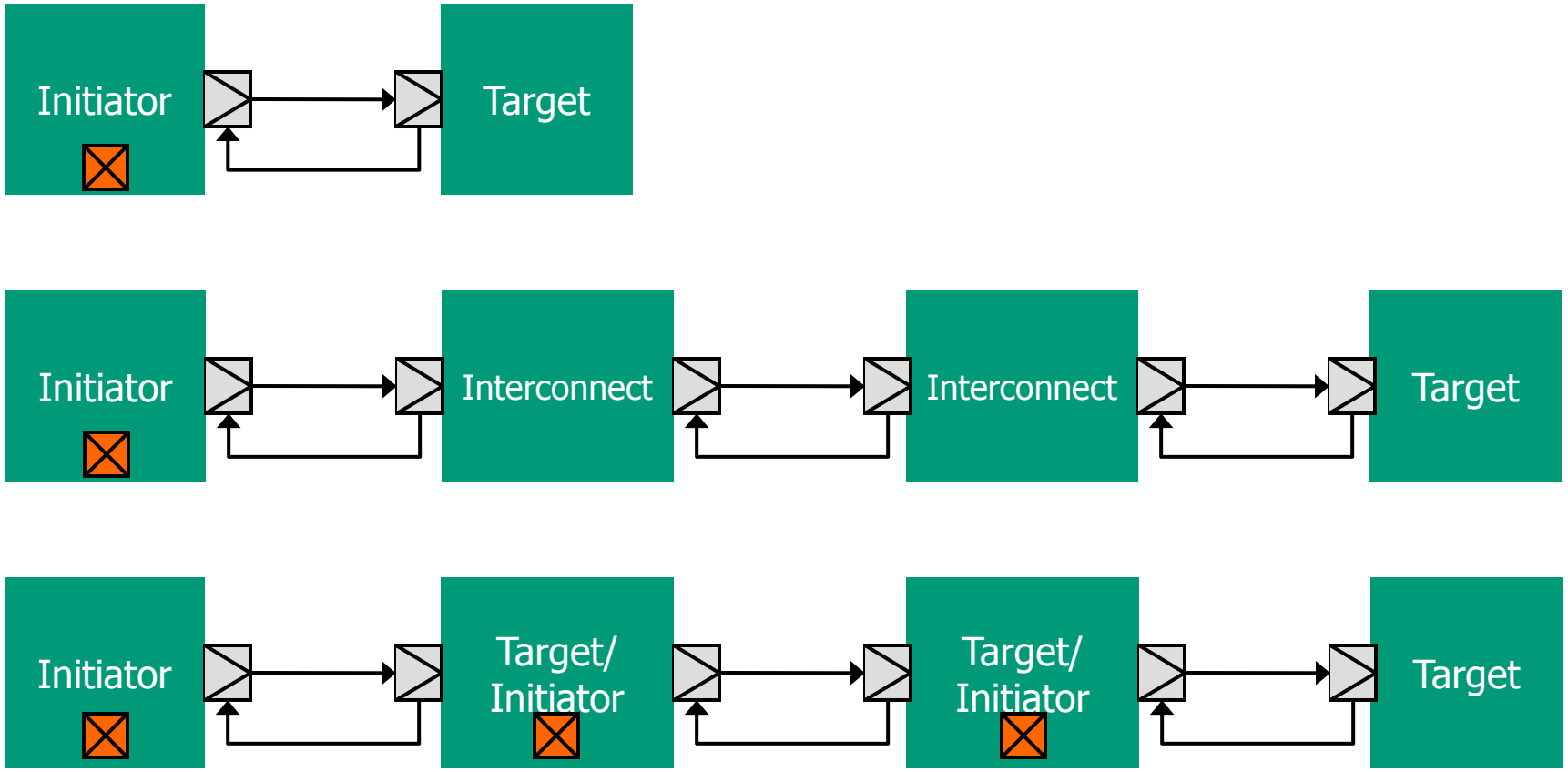
- Initiators and Targets
- Blocking Transport Interface
- Timing Annotation and the Quantum Keeper
- Non-blocking Transport Interface
- Timing Annotation and the Payload Event Queue

Initiators and Targets



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

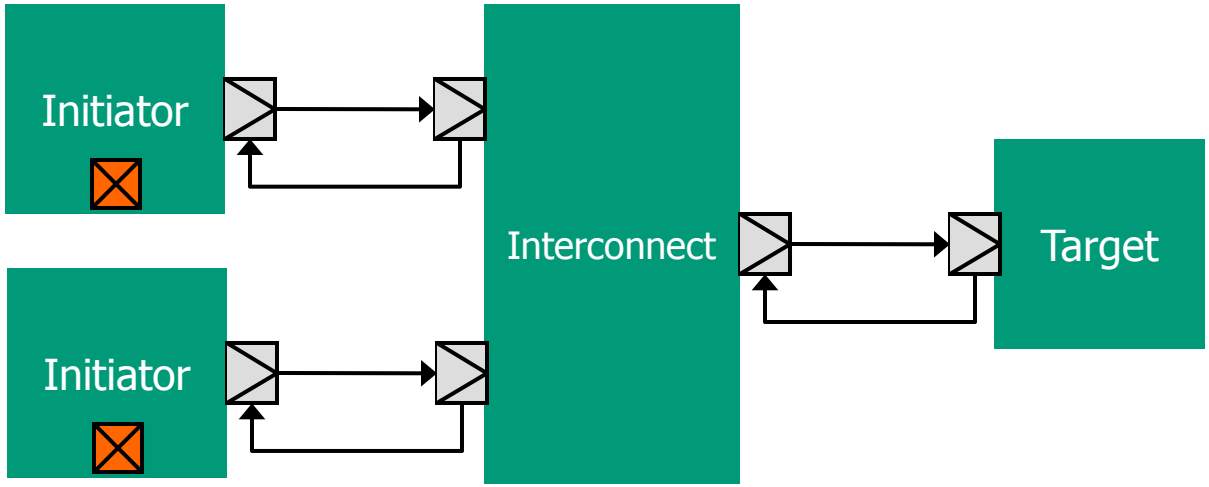
TLM-2 Connectivity



Transaction memory management needed

SYSTEMC™

Convergent Paths



Blocking versus Non-blocking Transport

- Blocking transport interface
  - Includes timing annotation
  - Typically used with loosely-timed coding style
  - Forward path only
- Non-blocking transport interface
  - Includes timing annotation and transaction phases
  - Typically used with approximately-timed coding style
  - Called on forward and backward paths
- Share the same transaction type for interoperability
- Unified interface and sockets – can be mixed

TLM-2 Core Interfaces - Transport

tlm_blocking_transport_if

```
sfm b_transport, X ERW* 0 gc mqi* -
```

tlm_fw_nonblocking_transport_if

```
pqc rgcir q nb_transport_fw, X ERW* 0TLEW* 0 gc mqi* -
```

tlm_bw_nonblocking_transport_if

```
pqc rgcir q nb_transport_bw, X ERW* 0TLEW* 0 gc mqi* -
```

TLM-2 Core Interfaces - DMI and Debug

tlm_fw_direct_mem_if

fssp **get_direct_mem_ptr**, X ERW* ver 0 pq chq m hq mchee -

tlm_bw_direct_mem_if

smm **invalidate_direct_mem_ptr**, gch ≫ m: evc verki 0
gch ≫ m: i r hc verki -

tlm_transport_dbg_if

r mkr i h m **transport_dbg**, X ERW* ver -

May all use the generic payload transaction type

Blocking Transport

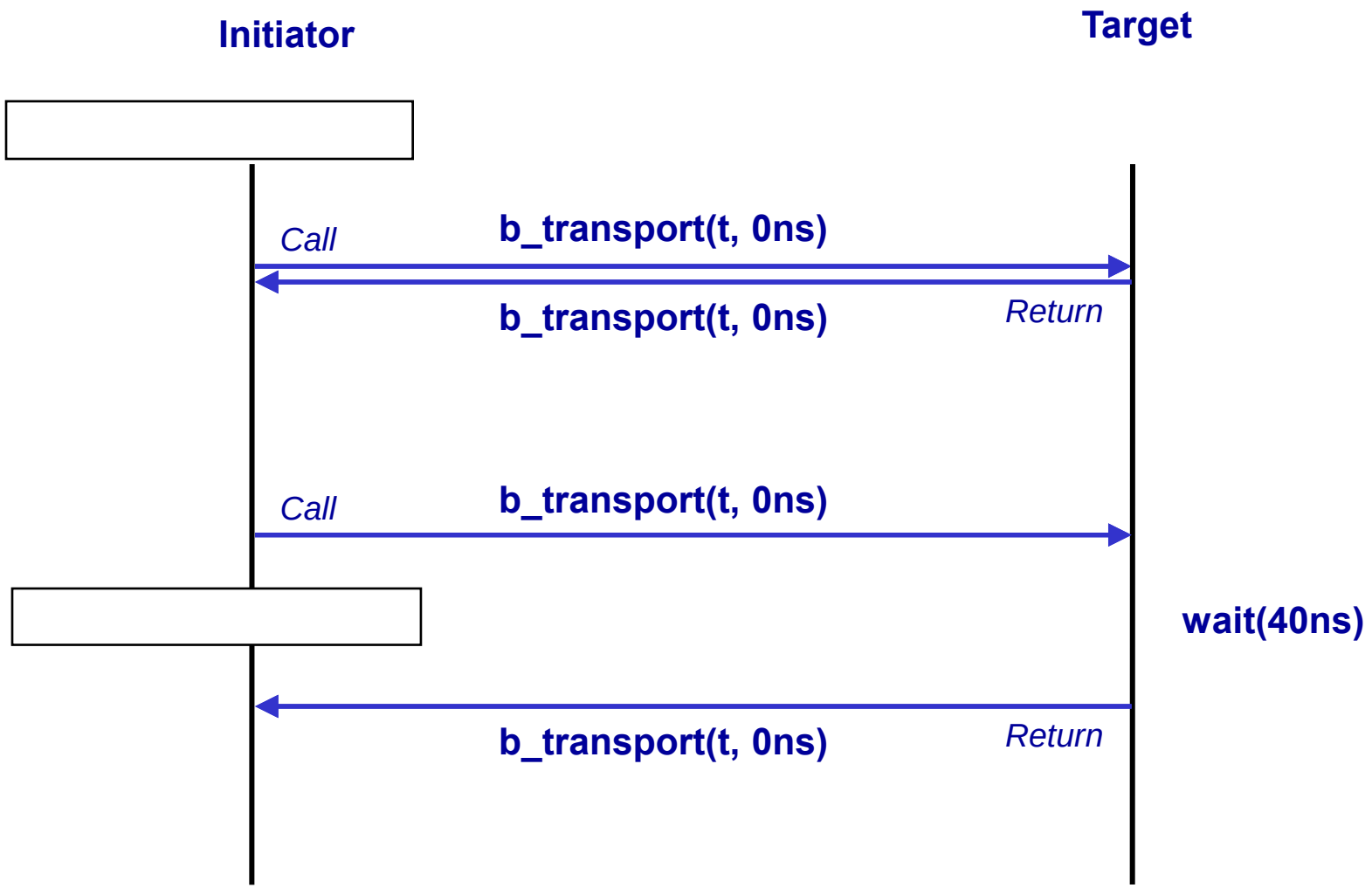
Transaction type

```
iqtpei    tireqi X ERW A pqckirivgcte pseh B

gpe   pqcfpsgomkc ver tsvcrjn>t fpqg mv ep gcgsvi ≫gcmnivjegi
t fpqg>
    mv ep  smn b_transport, X ERW*  ver  0 gcgsvi ≫gc mqi *  - A 4
```

Transaction object

Timing annotation

Blocking Transport



Initiator is blocked until return from b_transport

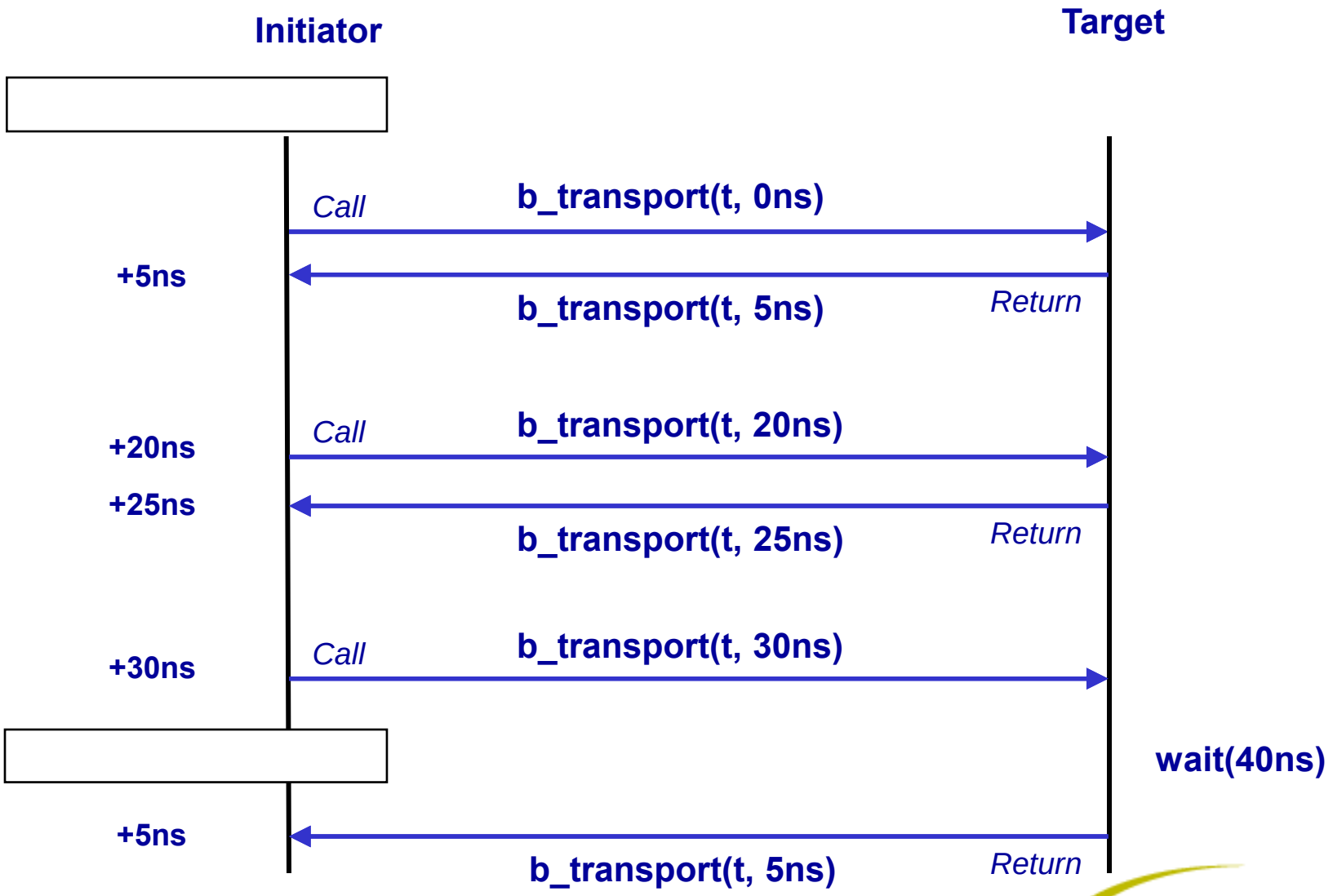
Timing Annotation

```
virtual  void  b_transport ( TRANS& trans , sc_core::sc_time& delay )
{
  //
  ...
  delay = delay + latency;
}
```

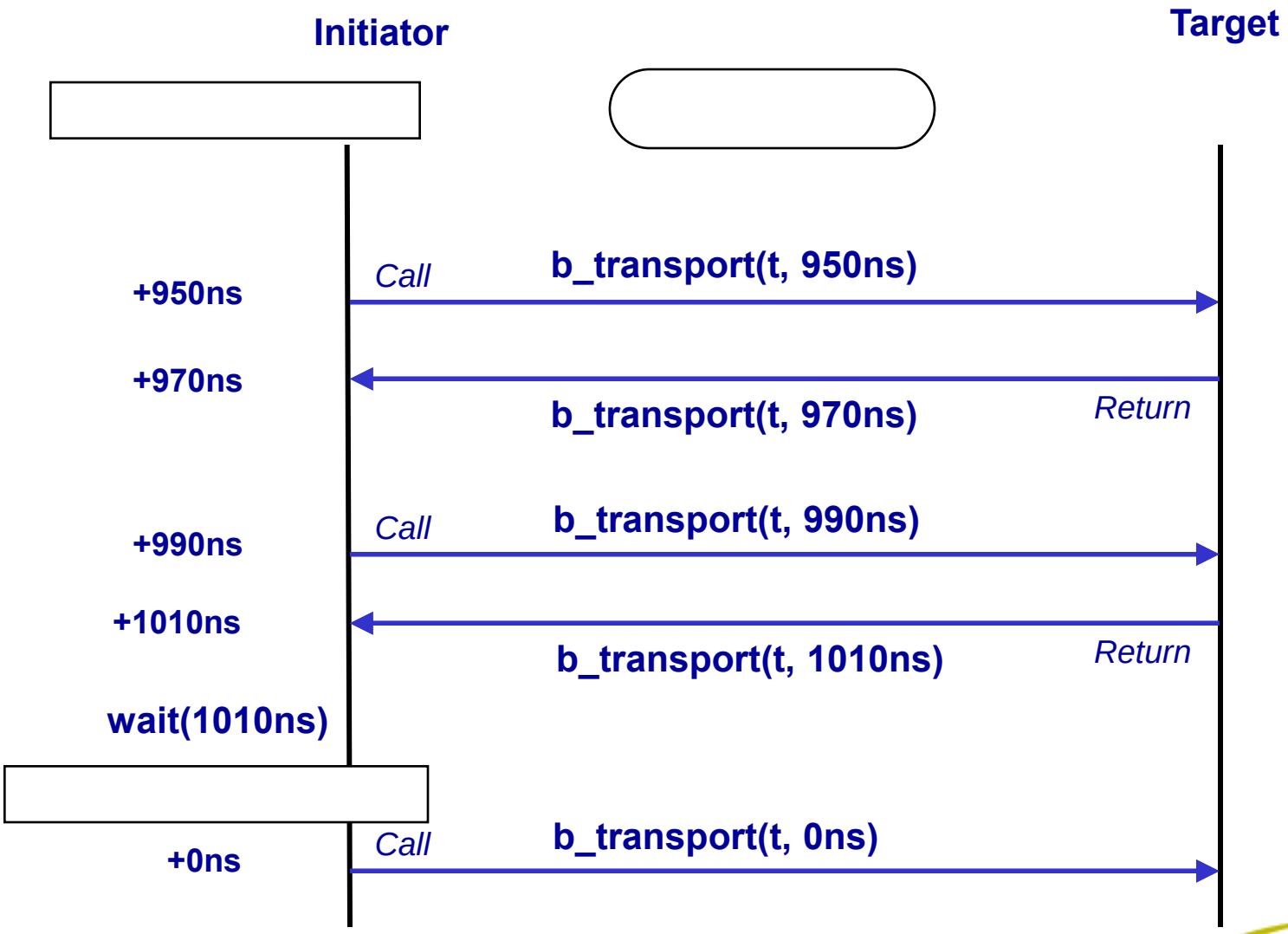
```
b_transport( transaction, delay );

 //
```

- Recipient may
  - Execute transactions immediately, out-of-order – Loosely-timed
  - Schedule transactions to execution at proper time – Approx-timed
  - Pass on the transaction with timing annotation

Temporal Decoupling

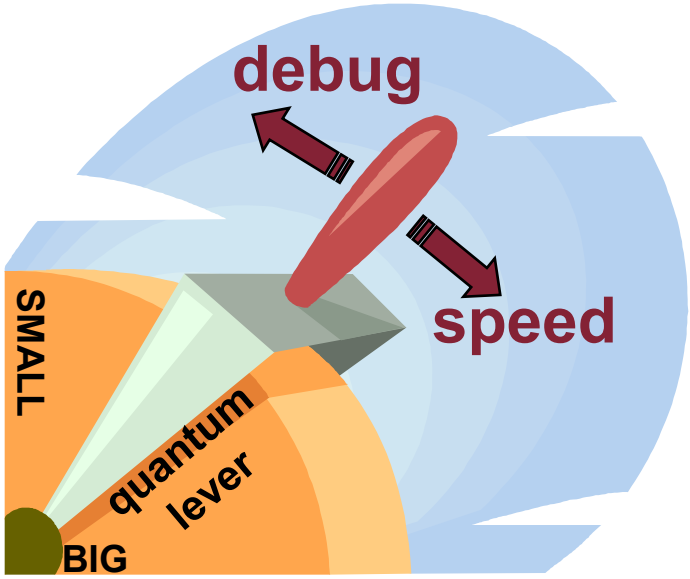


The Time Quantum



The Quantum Keeper (tlm_quantumkeeper)

- Quantum is user-configurable



- Processes can check local time against quantum

Quantum Keeper Example

```
v g MVmmeasv> gcqsh pi

 poqc  mp>>mqtpicmnmeasvc sgoi  MVmmeasvB mnnc sgoi
tlm_utils::tlm_quantumkeeper qcuo

WGcGXS ,MVmmeasv->mnnc sgoi ,&mnnc sgoi &
   222
   qcuo2set_global_quantum, gc mqi ,50WGcYW-
   qcuo2reset,-

 smn l vieh,- 222
  jsv,mn mA4 m  YRcPl RKXL m/A -
     222
     hi pe Aqcuo2get_local_time,-
     mnnc sgoi 1Bfc ver tsv, ver 0hi pe -
     qcuo2set, hi pe -
     qcuo2inc, gc mqi ,5440WGcRW-
     mn, qcuo2need_sync,--
        qcuo2sync,-
```

The quantum keeper

Replace the global quantum

Recalculate the local quantum

Time consumed by transport

Further time consumed by initiator

Check local time against quantum and sync if necessary

Non-blocking Transport

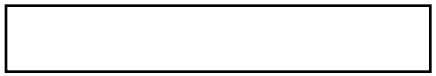
```
ir q  tlm_sync_enum    XPQcEGGI TXI H0 XPQcYTHEXI H0 XPQcGSQTPI XI H


iqtpei      tireqi X ERW A pqckirivgcte pseh0
            tireqi TLEW A pqctleiB

gpe   tlm_fw_nonblocking_transport_if >t fpg  m ep gcgsvi ≫gcmnivjegi
t fpg>
   m ep pqc  rgcir q  nb_transport,        X ERW*  ver 0
                                           TLEW* tlei0
                                            gcgsvi ≫gc mqi*   - A 4
```

Trans, phase and time arguments set by caller and modified by callee

tlm_sync_enum

- TLM_ACCEPTED
  - Transaction, phase and timing arguments unmodified (ignored) on return
  - Target may respond later (depending on protocol)
- TLM_UPDATED
  - Transaction, phase and timing arguments updated (used) on return
  - Target has advanced the protocol state machine to the next state
- TLM_COMPLETED
  - Transaction, phase and timing arguments updated (used) on return
  - Target has advanced the protocol state machine straight to the final phase

Notation for Message Sequence Charts

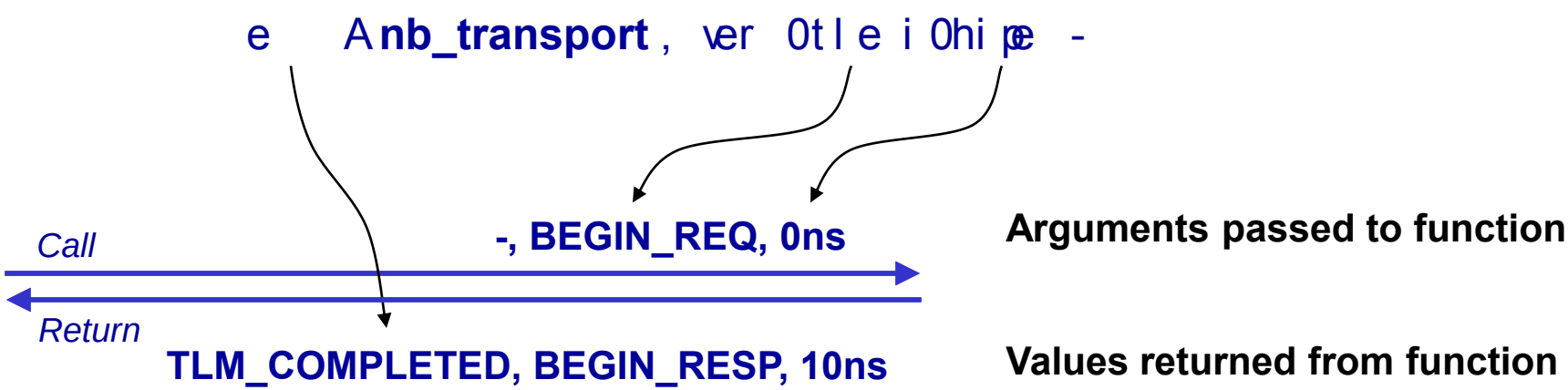


+10ns

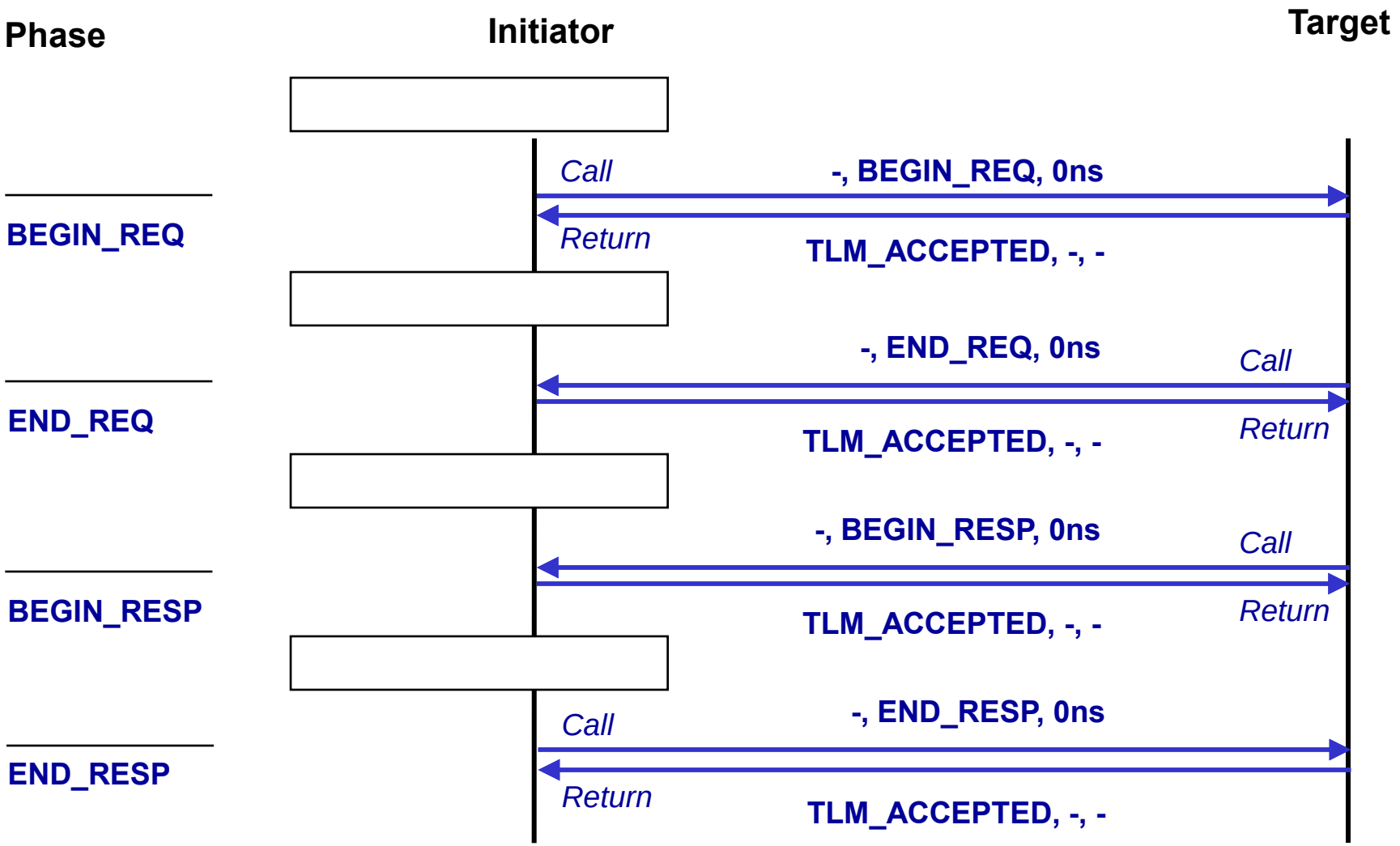
+20ns

= sc_time_stamp()

For temporal decoupling, local time is added to simulation time (explained on slides)

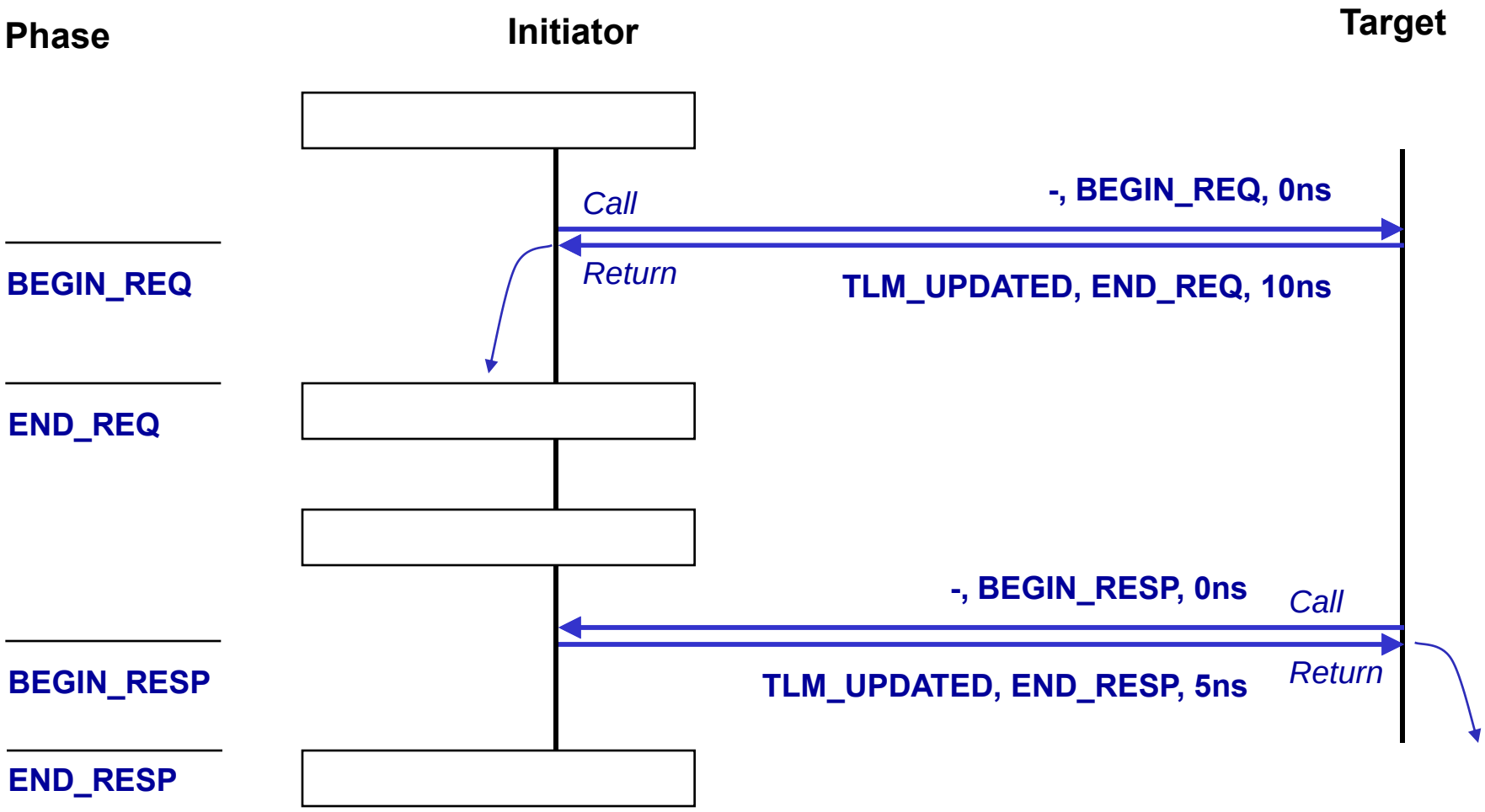


Using the Backward Path



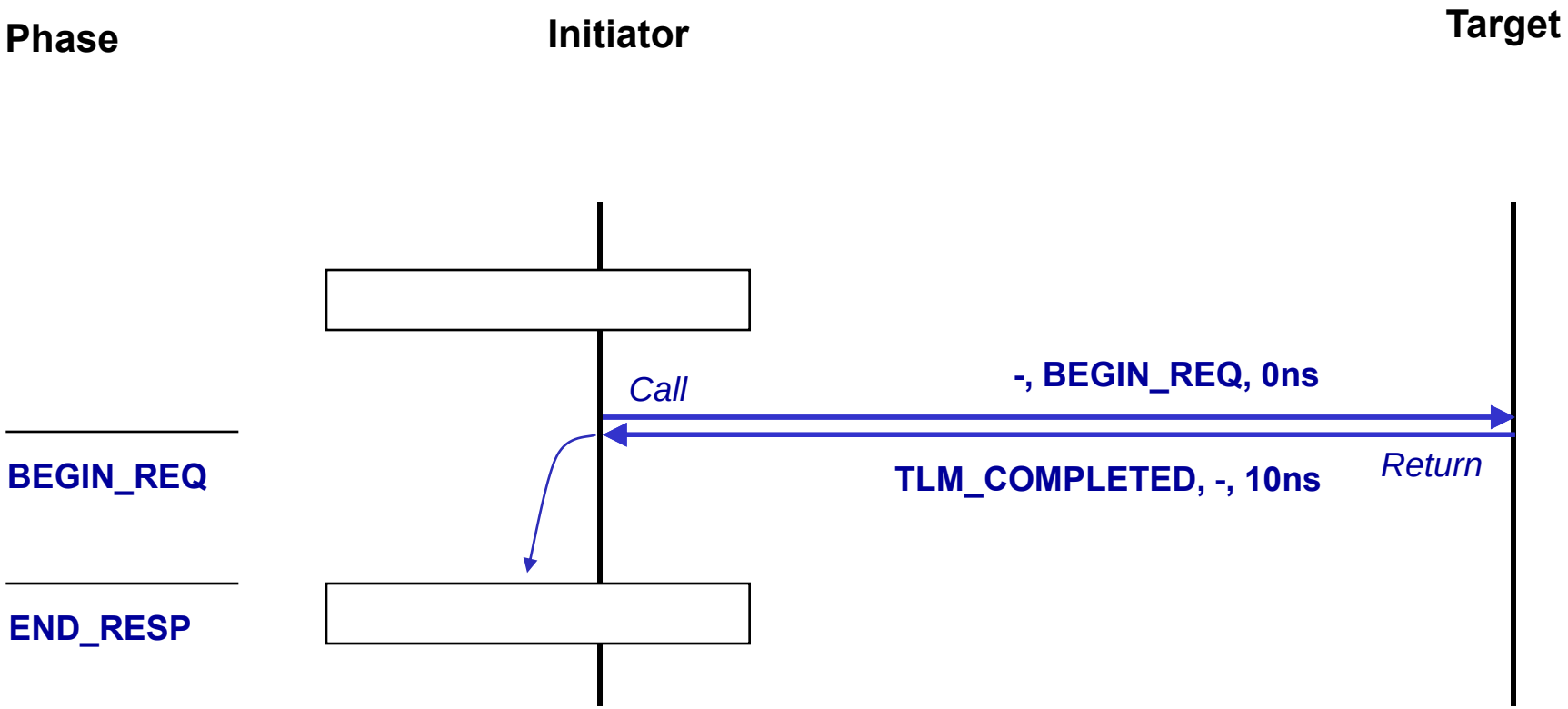
Transaction accepted now, caller asked to wait

Using the Return Path



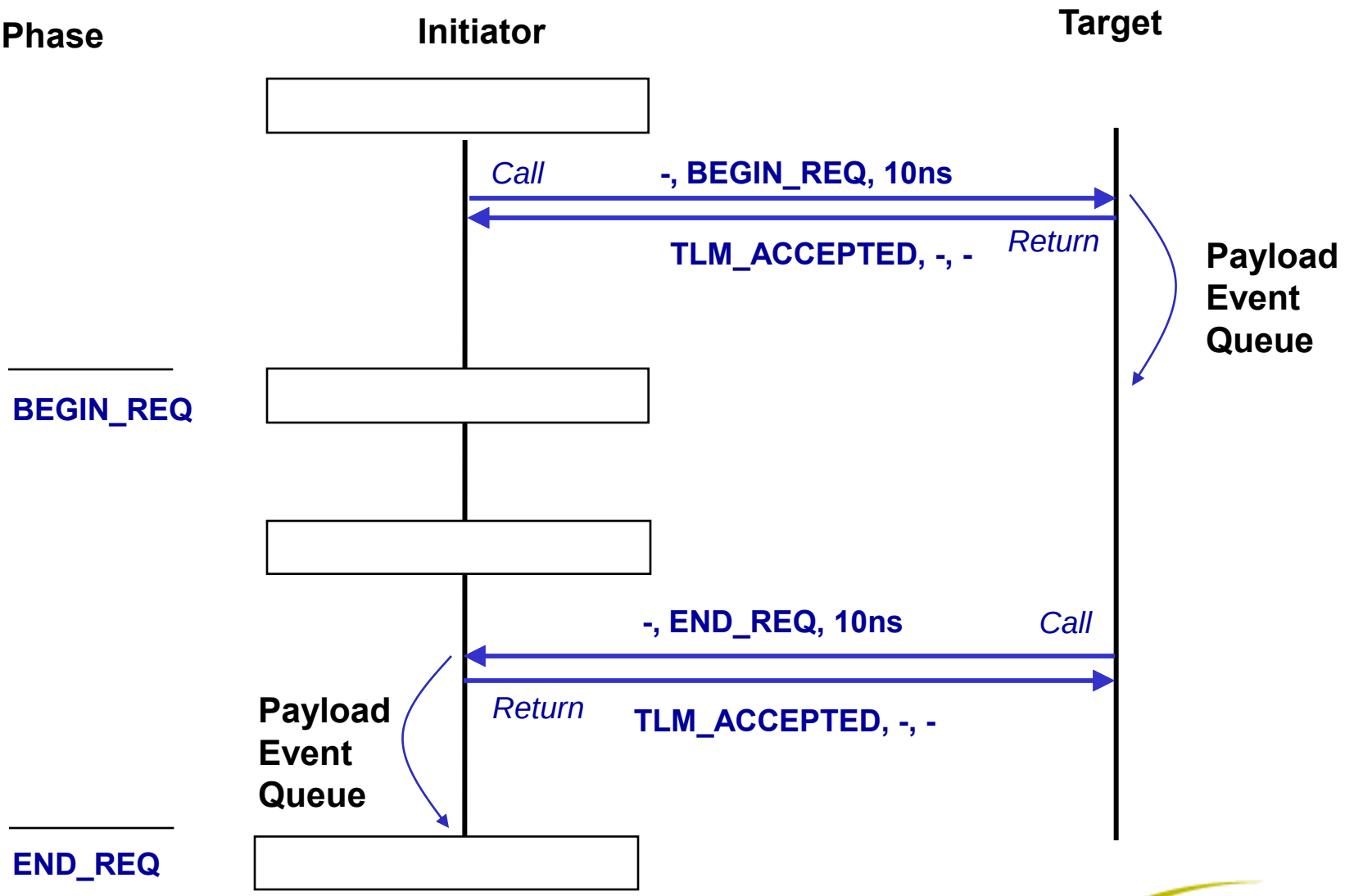
Callee annotates delay to next transition, caller waits

Early Completion



Callee annotates delay to next transition, caller waits

Timing Annotation



Payload Event Queue

```
iqtpei  gpe  TE]PSEHB
gpe  peq_with_get>t fpog gcgsvi≫gcsfnig

t fpog>
 tiuc mlcki,gsr  glev.reqi-

  smn notify, TE]PSEH*  ver 0 gcgsvi≫gcmqi*  -
  smn notify, TE]PSEH*  ver  -

  ver egmsrc ti. get_next_transaction,-
  gcgsvi≫gci ir * get_event(-
```

```
lmn,vi-
   em,qctiu2get_event,--
   lmn,,ver Aqctiu2get_next_transaction,--%A4-
    222
```


SYSTEMC™

SWGMXPQ1624

DMI AND DEBUG INTERFACES

- Direct Memory Interface
- Debug Transport Interface

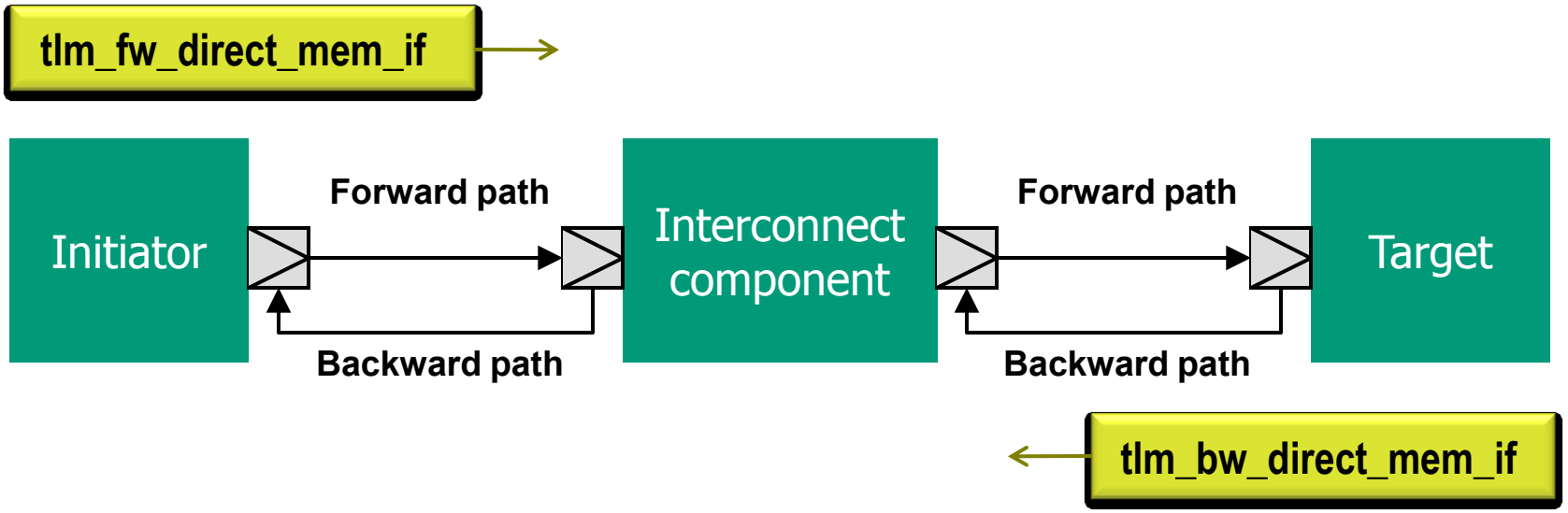
DMI and Debug Transport

- Direct Memory Interface
  - Gives an initiator a direct pointer to memory in a target, e.g an ISS
  - By-passes the sockets and transport calls
  - Read or write access by default
  - Extensions may permit other kinds of access, e.g. security mode
  - Target responsible for invalidating pointer
- Debug Transport Interface
  - Gives an initiator debug access to memory in a target
  - Delay-free
  - Side-effect-free
- May share transactions with transport interface

Direct Memory Interface

Access requested *Access granted*

status = get_direct_mem_ptr(transaction, dmi_data);



invalidate_direct_mem_ptr(start_range, end_range);

Transport, DMI and debug may all use the generic payload

Interconnect may modify address and invalidated range

DMI Transaction and DMI Data

HQMXver eg rsr

```
 i u i    vieh sv  vmi eggi
 sve kmi r ehhvi
Ti vq m i  i r rsr
```

gpe pq chq m

```
 r mkrih gl ev.   hq rcnt v
 mn:              hq rcn evcehhvi
 mn:              hq rcni r hcehhvi
hq rcn ti ci      hq rcn ti
 gc rq i          vi ehcpe i r g
 gc rq i            vmi cpe i r g
```

Direct memory pointer

Region granted for given access type

Read, write or read/write

Latencies to be observed by initiator

DMI Rules 1

- Initiator requests DMI from target at a given address
- DMI granted for a particular access type and a particular region
  - Target can only grant a single contiguous memory region containing given address
  - Target may grant an expanded memory region
  - Target may promote READ or WRITE request to READ_WRITE
- Initiator may assume DMI pointer is valid until invalidated by target
- Initiator may keep a table of DMI pointers

DMI Rules 2

- DMI request and invalidation use same routing as regular transactions
- The invalidated address range may get expanded by the interconnect
- Target may grant DMI to multiple initiators (given multiple requests)
  - and a single invalidate may knock out multiple pointers in multiple initiators
- Use the Generic Payload DMI hint (described later)
- Only makes sense with loosely-timed models

Debug Transport Interface

Gsq q er h
Ehhvi
He e t sm i v
He e p r k l
l i r rsr

num_bytes = transport_dbg(transaction);

tlm_transport_dbg_if

Uses forward path only

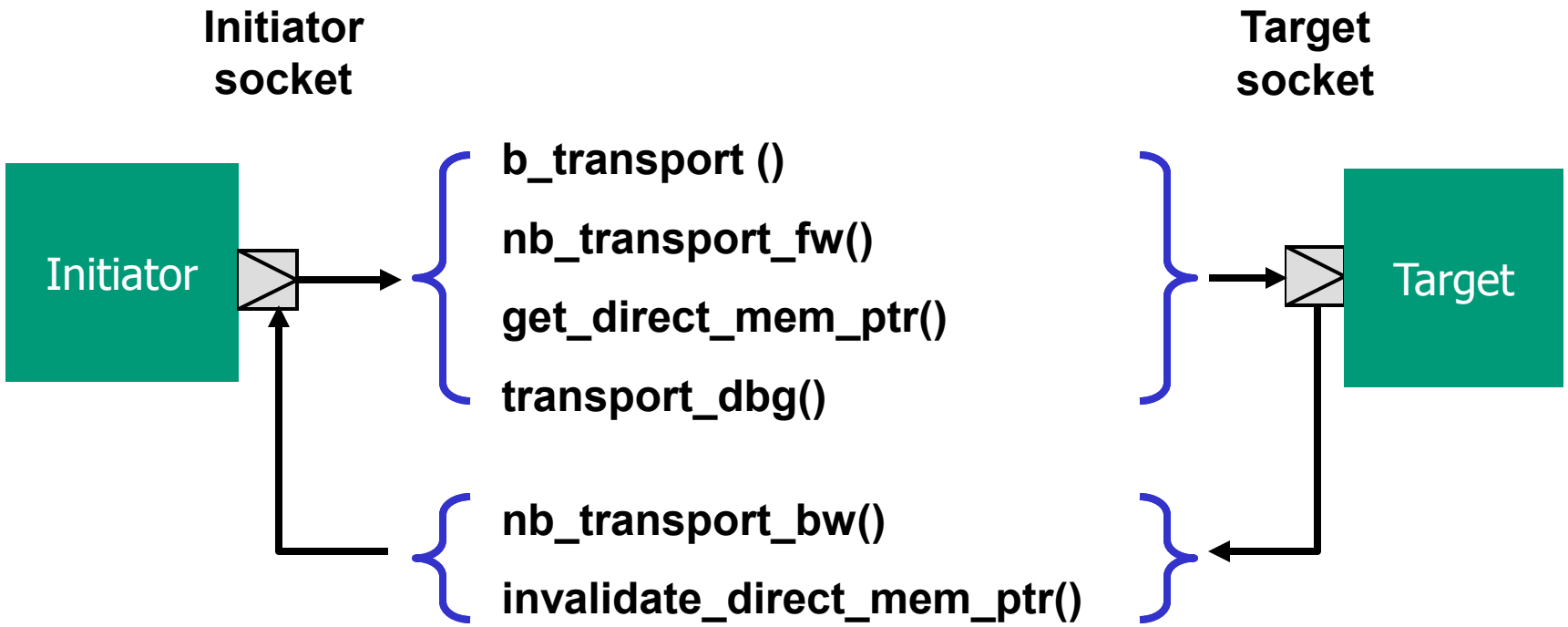
Interconnect may modify address, target reads or writes data

SWGMXKPQ1621

SOCKETS

- Initiator and target sockets
- Simple sockets
- Tagged sockets
- Multi-port sockets

Initiator and Target Sockets



Sockets provide fw and bw paths and group interfaces

Benefit of Sockets

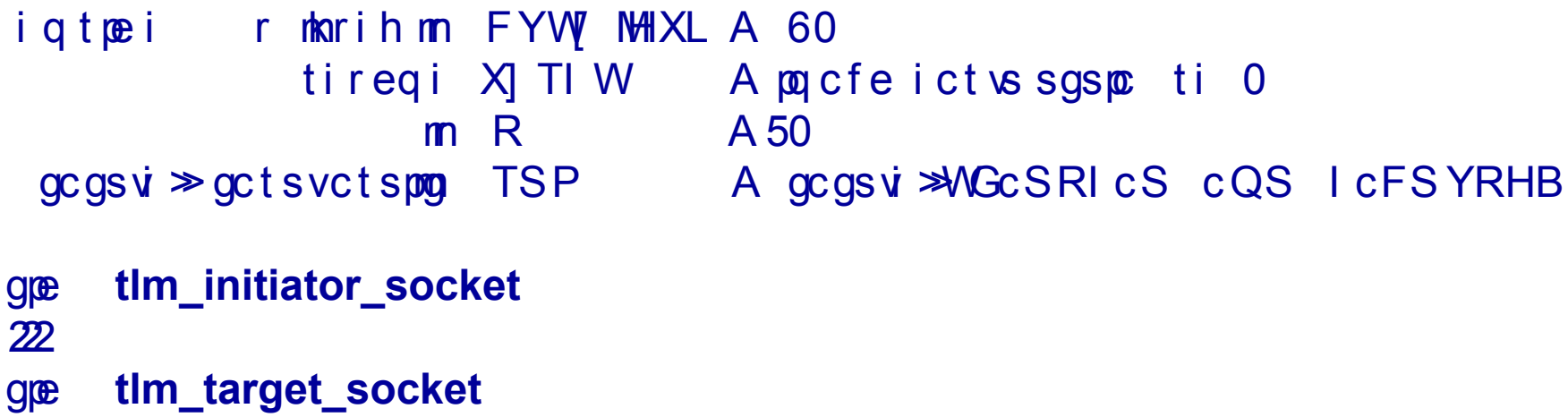
- Group the transport, DMI and debug transport interfaces
- Bind forward and backward paths with a single call
- Strong connection checking
- Have a bus width parameter

- Using core interfaces without sockets is not recommended

Sockets and Transaction Types

- All interfaces templated on transaction type
- Use the generic payload and base protocol for interoperability
  - Use with transport, DMI and debug transport
  - Supports extensions
  - Even supports extended commands and phases
  - Ignorable extensions allow interoperability
  - Mechanism to disallow socket binding for non-ignorable extensions
  - Described later

Standard Socket Classes



- Part of the interoperability layer
- Initiator socket must be bound to an object that implements entire backward interface
- Target socket must be bound to an object that implements entire forward interface
- Can mix blocking and non-blocking calls – target must support both together
- Allow hierarchical binding

Socket Binding Example 1

```
v g  MMmmeasv> gcq sh pi 0 pq >>tlm_bw_transport_if B

 pq >>tlm_initiator_socket<> mnnc sgoi

WGcGXS  ,MMmmeasv->mnnc sgoi ,&mnnc sgoi &
  WGcXL  I EH,Ivieh-
  mnnc sgoi 2bind,.Im-

 smn Ivieh,-  222
  mnnc sgoi 1Bfc ver tsv, ver 0hipe -
  mnnc sgoi 1Brfc ver tsvcj , ver 0tIe i0hipe -
  mnnc sgoi 1Bki chmigcqiqct v, ver 0hqmchee-
  mnnc sgoi 1B ver tsvchfk, ver -

 m ep pq >>pqc  rgcir q rfc ver tsvcf , 222-  222
 m ep smn mn epmheichmigcqiqct v, 222-  222
```

Combined interface required by socket

Protocol type defaults to base protocol

Initiator socket bound to initiator itself

Calls on forward path

Methods for backward path

Socket Binding Example 2

```
v g  Xevki  > gcq sh  pi 0  pq >tlm_fw_transport_if  B
```

Combined interface required by socket

```
 pq >tlm_target_socket<>  evkc sgoi
```

Protocol type default to base protocol

```
 WGcGXS  ,Xevki  - > evkc sgoi  ,&evkc sgoi  &
    evkc sgoi  2bind, . l m-
```

Target socket bound to target itself

```
 m ep smfc ver  t sv, 222-  222
 m ep pq >pqc  rgci r  q  r fc ver  t svcj  , 222-  222
 m epfsspki chmigcqiqct v, 222-  222
 m ep r mkrih m  ver  t svchfk, 222-  222
```

Methods for forward path

```
WGcQSHYPI ,Xst-
   MImmesv . mnm
   Xevki    . evk
   WGcGXS  ,Xst-
     mm Ari   MImmesv,&nm&
      evk Ari   Xevki  ,&evk&
     mm1Bmmc  sgoi  2bind, evk1B evkc sgoi  -
```

Bind initiator socket to target socket

SYSTEMC™

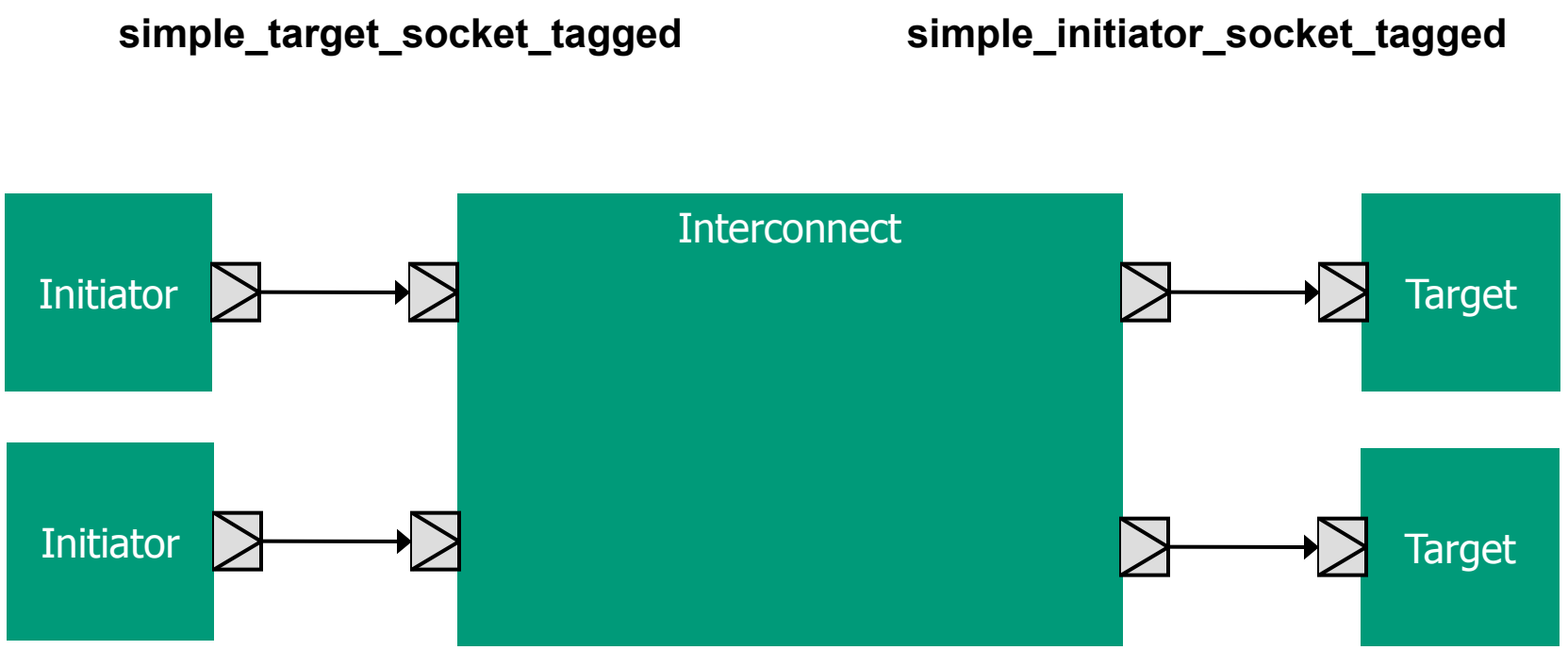
Convenience Sockets

- The “simple” sockets
  - simple_initiator_socket and simple_target_socket
    - In namespace tlm_utils
    - Derived from tlm_initiator_socket and tlm_target_socket
  - “simple” because they are simple to use
    - Do not bind sockets to objects (implementations)
    - Instead, register methods with each socket
    - Do not allow hierarchical binding
  - Not obliged to register both b_transport and nb_transport
    - Automatic conversion (assumes base protocol)
      - Variant with no conversion – passthrough_target_socket

Simple Socket Example

```
v g  MMi vgsrri g > gcqsh p

pqc  mp>>simple_target_socket MMi vgsrri gB  evkc sgoi
pqc  mp>>simple_initiator_socket MMi vgsrri gB mnnc sgoi

WGcGXS ,MMi vgsrri g- > evkc sgoi ,&evkc sgoi &0mnnc sgoi ,&mnnc sgoi &

 evkc sgoi 2register_nb_transport_fw,   lm0* MMi vgsrri g>>rfc ver tsvcj -
 evkc sgoi 2register_b_transport,       lm0* MMi vgsrri g>>fc ver tsv-
 evkc sgoi 2register_get_direct_mem_ptr, lm0* MMi vgsrri g>>ki chmi gcqiqct v-
 evkc sgoi 2register_transport_dbg,     lm0* MMi vgsrri g>>ver tsvchfk-
mnnc sgoi 2register_nb_transport_bw,   lm0* MMi vgsrri g>>rfc ver tsvcf  -
mnnc sgoi 2register_invalidate_direct_mem_ptr,
                                       lm0* MMi vgsrri g>>mepmeichmi gcqiqct v-

mi ep smmfc ver tsv, 222-
mi ep pq>>pqc  rgcir q rfc ver tsvcj , 222-
mi epfsspki chmi gcqiqct v, 222-
mi ep r mkrihmn ver tsvchfk, 222-
mi ep pq>>pqc  rgcir q rfc ver tsvcf , 222-
mi ep smmmn epmeichmi gcqiqct v, 222
```

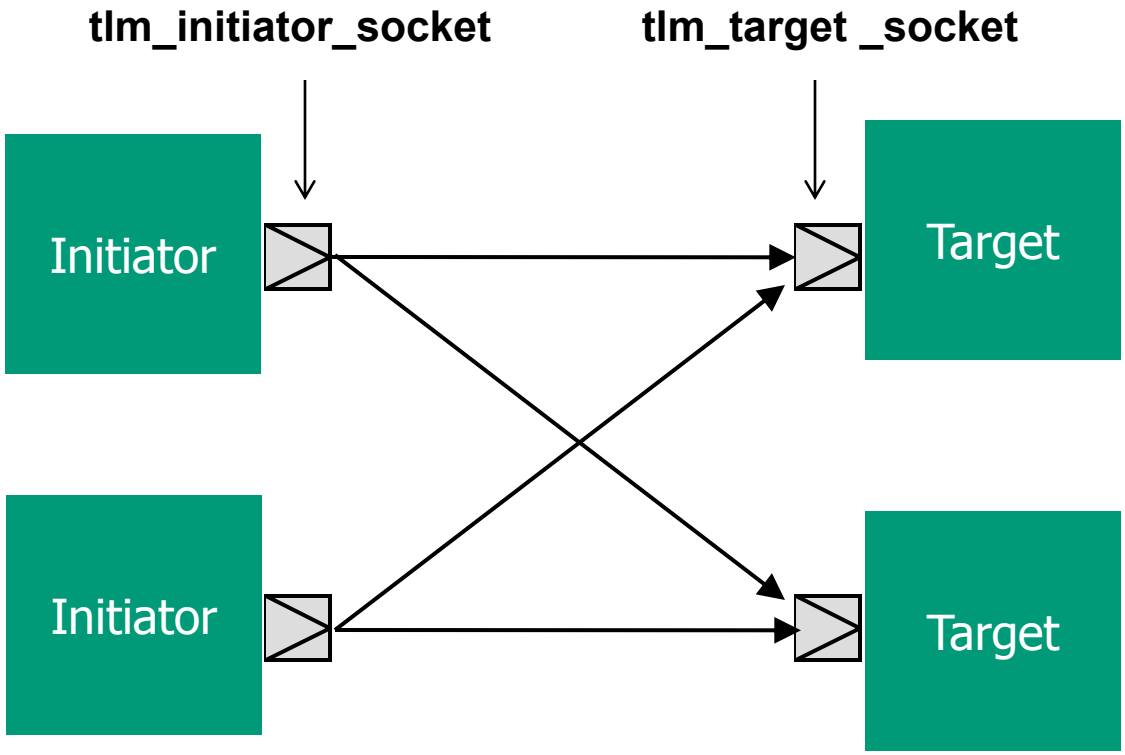
Tagged Simple Sockets



Tagged Simple Socket Example

```
mngphi &pqc mp3mqtpicmnmeasvc sgoi 2&
mngphi &pqc mp3mqtpic evki c sgoi 2&

iqtpei r mkrihmn RcMRNXMEXS W0 r mkrihmn RcXE KIXWB
 vg F > gcqsh pi

 pqc mp>>simple_target_socket_tagged F B. evkc sgoi _RcMRNXMEXS Wa
 pqc mp>>simple_initiator_socket_tagged F B. mnc sgoi _RcXE KIXWa

WGcGXS ,F -
 jsv, r mkrihmn mnA4 m RcMRNXMEXS Wm/-
   evkc sgoi _mnaAri pqc mp>>mqtpic evki c sgoi cekkih F B, -

   evkc sgoi _mn1Bregister_b_transport Im0* F >>fc ver tsv0id -
   222

 m ep smnfc ver tsv, int id0 pq>>pqckiri vgcte pseh* ver 0 gc mqi * hipe -
 222
```

Many-to-many Binding




```
mnmc sgoi _4a1Bfc ver tsv,222
mnmc sgoi _5a1Bfc ver tsv,222
```

```
evki c sgoi _4a1Br fc ver tsvcf ,222
evki c sgoi _5a1Br fc ver tsvcf ,222
```

- **Multi-ports – can bind many-to-many, but incoming calls are anonymous**

Multi-port Convenience Sockets

- multi_passthrough_initiator_socket
- multi_passthrough_target_socket

- Many-to-many socket bindings
- Method calls tagged with multi-port index value

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

SWGMXPQ1621

THE GENERIC PAYLOAD

- Attributes
- Memory management
- Response status
- Endianness
- Extensions

The Generic Payload

- Typical attributes of 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
- Used to model specific bus protocols
  - mandatory static extensions
  - compile-time type checking to avoid incompatibility
  - low implementation cost when bridging protocols

Specific protocols can use the same generic payload machinery

Generic Payload Attributes

Attribute	Type	Modifiable?	
Command	tlm_command	No	
Address	uint64	Interconnect only	
Data pointer	unsigned char*	No (array – yes)	*Array owned by initiator*
Data length	unsigned int	No	
Byte enable pointer	unsigned char*	No (array – yes)	*Array owned by initiator*
Byte enable length	unsigned int	No	
Streaming width	unsigned int	No	
DMI hint	bool	Yes	*Try DMI !*
Response status	tlm_response_status	Target only	
Extensions	(tlm_extension_base*)[ ]	Yes	*Consider memory management*

class tlm_generic_payload

```
gpe    pqckirivgcte pseh
t fpqg>
```

Not a template

```
   33Constructors, memory management
    pqckirivgcte pseh,-
    pqckirivgcte pseh, pqcqqcmnivjegi * qq-
     mv ep  pqckirivgcte pseh,-
     smnvi i,-

     smn i cqq, pqcqqcmnivjegi. qq-
   fsspl e cqq,-
     smnegu mi,-
     smnvipe i,-
   mn ki cvijcgs r,-

     smnhiitcgst cjvsq,gsr    pqckirivgcte pseh* sliv-

   222
```

Construct & set mm

Frees all extensions

mm is optional

Incr reference count

Decr reference count, 0 => free trans

Memory Management Rules

- b_transport – memory managed by initiator, or reference counting (set_mm)
- nb_transport – reference counting only
  - Reference counting requires heap allocation
  - Transaction automatically freed when reference count == 0
  - free() can be overridden in memory manager for transactions
  - free() can be overridden for extensions
- When b_transport calls nb_transport, must add reference counting
  - Can only return when reference count == 0
- b_transport can check for reference counting, or assume it could be present

Command, Address and Data

```
ir q pqcgsqqerh
  XPQc lEHcGSQQERH0
  XPQc[ MXlcGSQQERH0
  XPQcMKRS lcGSQQERH
```

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

```
pqcgsqqerh     ki cgsqqerh,- gsr
 srn            i cgsqqerh, gsr   pqcgsqqerh gsqqerh -

 gch≫rn:       ki cehhvi  ,- gsr
 srn            i cehhvi  , gsr   gch≫rn:  ehhvi   -

 r rkrihglev   ki cheect v,- gsr
 srn            i cheect v, r rkrihglev hee -

 r rkrihrn     ki cheecpirkl,- gsr
 srn            i cheecpirkl, gsr   r rkrihrn pirkl -
```

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 on address
TLM_COMMAND_ERROR_RESPONSE	Unable to execute command
TLM_BURST_ERROR_RESPONSE	Unable to act on data length or streaming width
TLM_BYTE_ENABLE_ERROR_RESPONSE	Unable to act on byte enable
TLM_GENERIC_ERROR_RESPONSE	Any other error

The Standard Error Response

- A target shall either
  - Execute the command and set TLM_OK_RESPONSE
  - Set the response status attribute to an error response
  - Call the SystemC report handler and set TLM_OK_RESPONSE
- Many corner cases
  - e.g. a target that ignores the data when executing a write – OK
  - e.g. a simulation monitor that logs out-of-range addresses – OK
  - e.g. a target that cannot support byte enables - ERROR

Generic Payload Example 1

```
smn l vi ehct vsgi   ,-  33XI i mnmme sv
  tlm::tlm_generic_payload  ver
   gc mqi  hi pe  AWGc  l  ScXMQI

   ver 2set_command, pq >>XPQc[   MXI cGSQQERH -
   ver 2set_data_length,   -
   ver 2set_byte_enable_ptr, 4 -
   ver 2set_streaming_width,   -

  jsv, mn mA4 m   YRcPI RKXL  m/A  -
     mn   svh Am
      ver 2set_address, m-
      ver 2set_data_ptr, , r  mkri h gl ev.-, *  svh - -
      ver 2set_response_status, pq >>XPQcMRGSQTPI XI c  I WTSRW -

     mmc sgoi 1Bb_transport, ver 0hi pe  -

     rjn, ver 2get_response_status,-  A4 -
        WGc  l TS  XcI    S  ,&XPQ680 ver 2get_response_string,-2gc  v,--
     222
```

Would usually pool transactions

Generic Payload Example 2

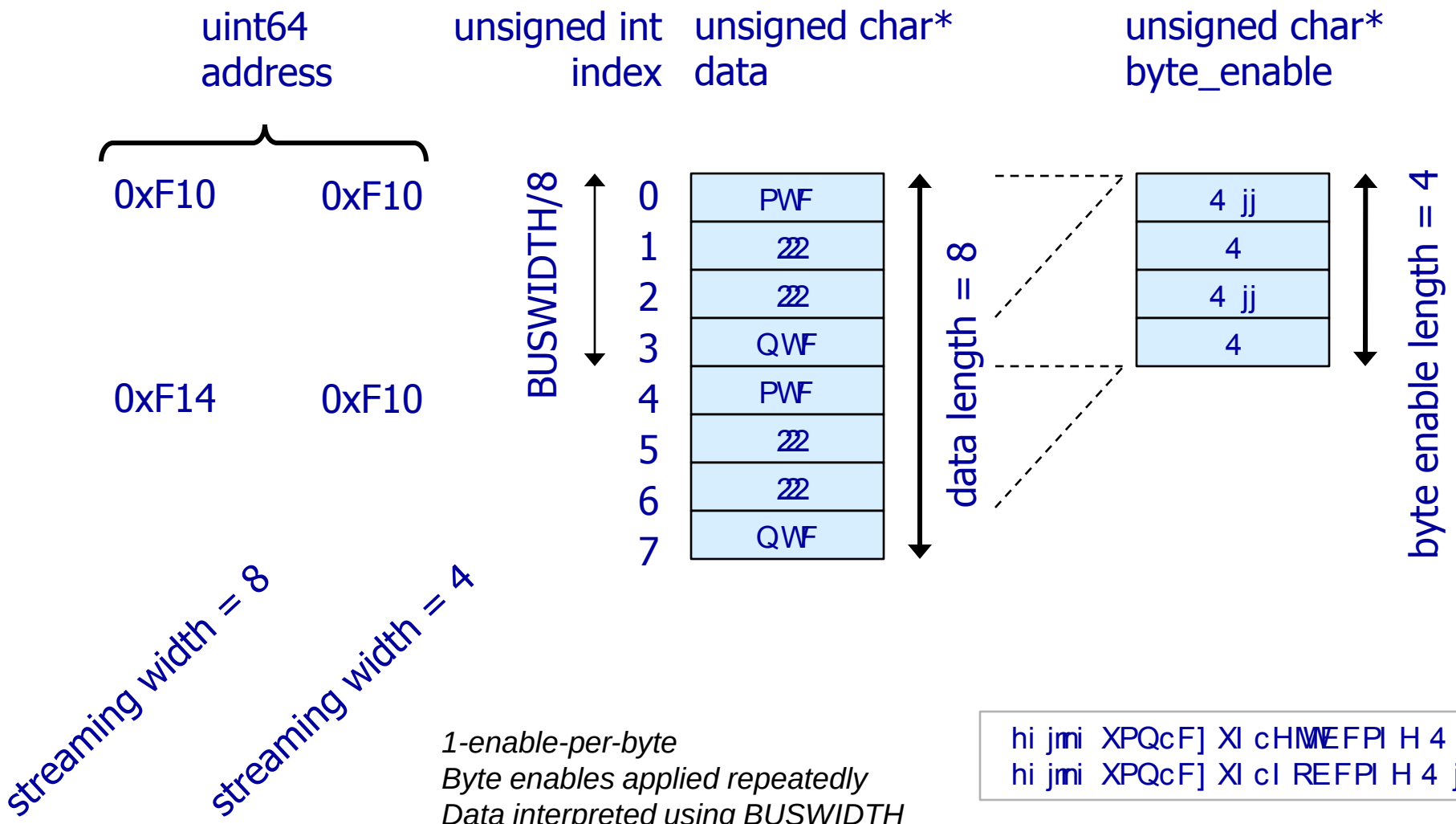
```
mv ep smn b_transport, 33Xli evki
  pq>>pqckirivrgcte pseh* ver 0 gcgsvi>>gc mqi* -

  pq>>pqcgsqqerh  gqh  A ver 2get_command,-
  gch>>mn:        ehv  A ver 2get_address,-
  r mkrihglev.    t v  A ver 2get_data_ptr,-
  r mkrihmn       pir  A ver 2get_data_length,-
  r mkrihglev.    f    A ver 2get_byte_enable_ptr,-
  r mkrihmn       mn   A ver 2get_streaming_width,-

  mn,ehv/pir Bqcpirkl-                                   Check for storage overflow
     ver 2set_response_status, pq>>XPQcEHH IWWcI  S c IWTSRW -
    vi vr

  mn,f -                                                 Unable to support byte enable
     ver 2set_response_status, pq>>XPQcF] XIcI REFPIcI  S c IWTSRW -
    vi vr

  mn, mn%A4** mn pir-                                    Unable to support streaming
     ver 2set_response_status, pq>>XPQcFY WXcI  S c IWTSRW -
    vi vr
```


SYSTEMC™

Generic Payload Example 3

```
mv ep smn b_transport, 33Xl i evki
  pqi >>pqi ckiri vngcte pseh*  ver  0 gcgsvi >>gc mqi *  -
  222
  222
  rjn, gqh AA pqi >>XPQc[    MXl cGSQQERH -                    Execute command

    qiqgt , * qc  sveki _ehva0t v0pir -

  ipi rjn, gqh AA pqi >>XPQc  l EHcGSQQERH -

    qiqgt , t v0* qc  sveki _ehva0pir -

   ver  2set_response_status, pqi >>XPQcSOc  l WTSRW -      Successful completion
```

Byte Enables and Streaming



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

hi jmi XPQcF] XI cHMWEFPI H 4 4
hi jmi XPQcF] XI cI REFPI H 4 jj

Byte Enable Example 1

```
33XIi mnmmeasv
 smn lviehctvsgi   ,-
   pq>>pqckirivrgcte pseh  ver
    gc rqi hipe

     erg  svhc  f  icirefpicqe oA4 4444jjjj p

    ver 2set_byte_enable_ptr,
      vimnivtvi cge    r mkrihglevB,*f  icirefpicqe o- -

    ver 2set_byte_enable_length,  -

    ver 2i cgsqqerh, pq>>XPQc[   MXIcGSQQERH-
    ver 2i cheecpirkl,  -
   222
   jsv,mn mA4 m   YRcPlRKXL  m/A -
      ver 2i cehhvi  ,m-
      ver 2i cheect v,, r mkrihglev-,*  svh--

     mnmc sgoi 1Bfc ver tsv, ver 0hipe -
   222
```

Uses host-endianness MSB..LSB

Byte Enable Example 2

```
mv ep smn fc ver tsv, pq >>pqckirivrgcte pseh* ver 0 gcgsvi >>gc mqi* -33XIi evki
  222
   r mkrihglev.      f      A ver 2get_byte_enable_ptr,-
   r mkrihmn         fip    A ver 2get_byte_enable_length,-
  222
  rjn,gqh AA pq >>XPQc[    MXIcGSQQERH-
    rjn,f  -
      jsv, r mkrihmn mA4 m pir rh/-
        rjn,f  _m) fipaAAXPQcF]XIclREFPIH-
          qc sveki_ehv/maAt v_ma
     ipi
      qiqgt ,* qc sveki_ehva0t v0pir-

   ipi rjn,gqh AA pq >>XPQc IEHcGSQQERH-
    rjn,f  -
       ver 2i cvi tsr ic e  , pq >>XPQcF]XIclREFPIcl   S c IWTSRW -
      vi  vr pq >>XPQcGSQTPIXIH
     ipi
      qiqgt ,t v0* qc sveki_ehva0pir-
```

Byte enable applied repeatedly

byt[i] corresponds to ptr[i]

No byte enables

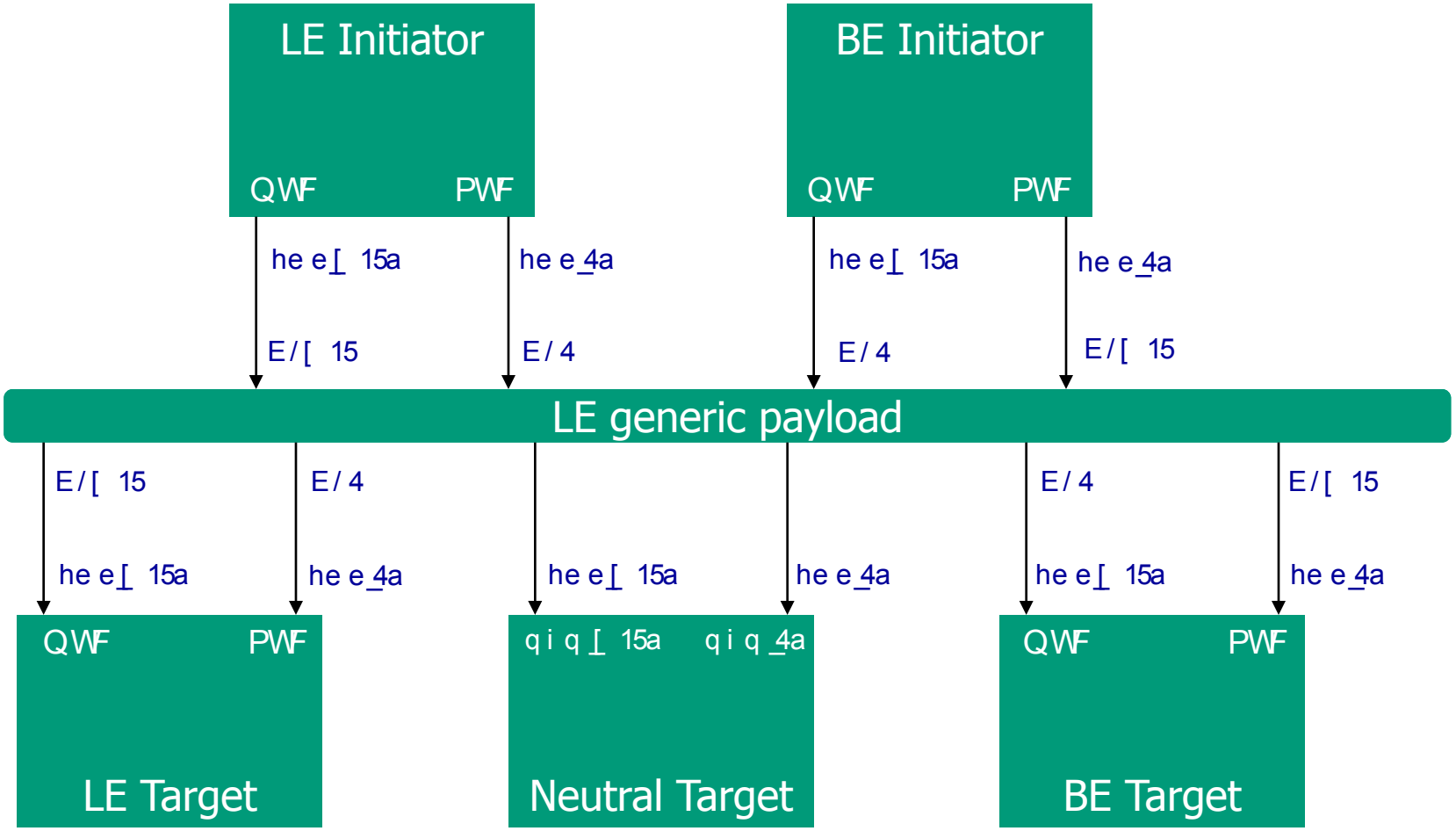
Target does not support read with byte enables

Endianness

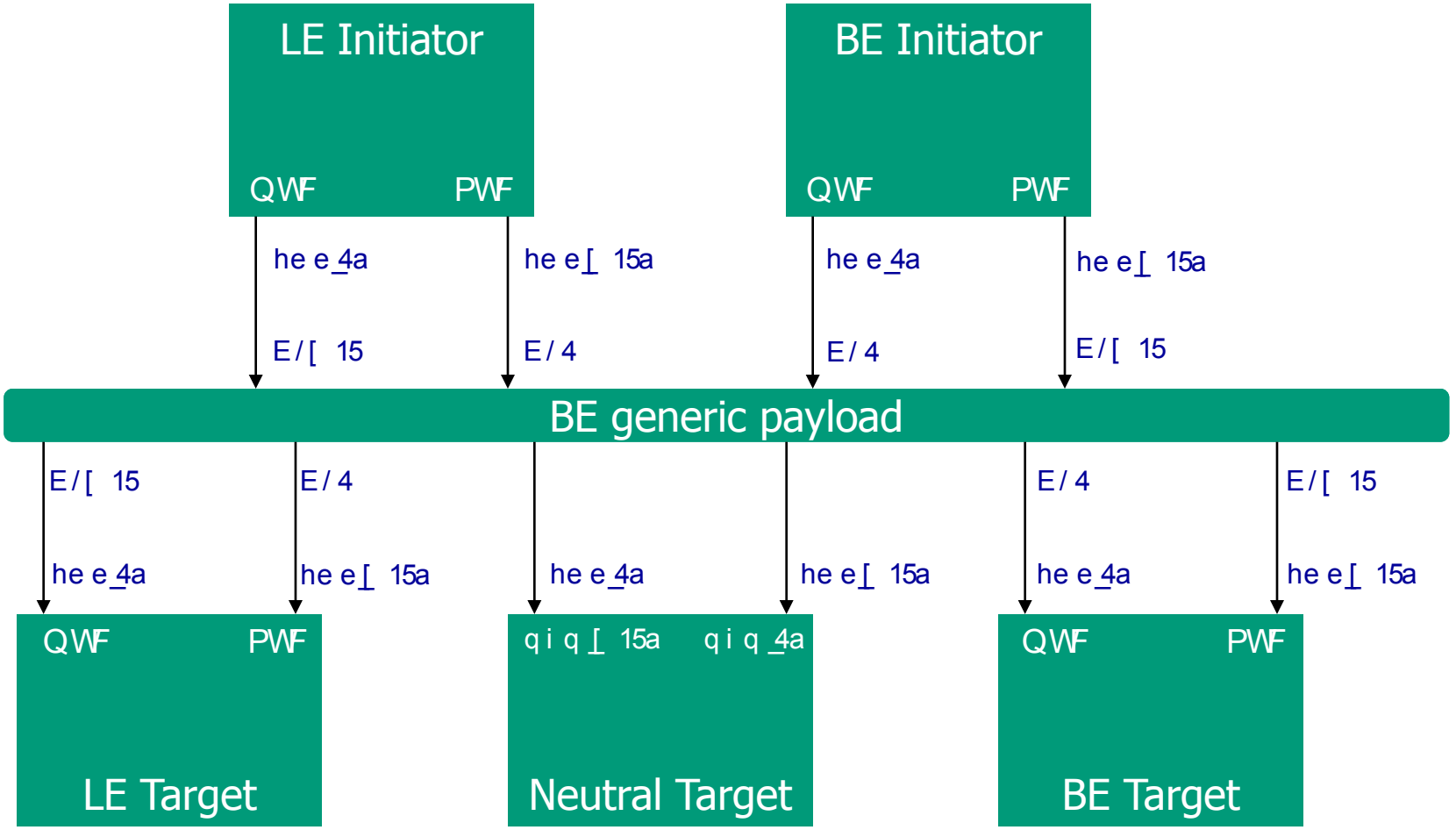
- Designed to maximize simulation speed
- Words in data array are host-endian
- Effective word length W = (BUSWIDTH + 7) / 8
- Initiators and targets connected LSB-to-LSB, MSB-to-MSB
- Most efficient when *everything* is modeled host-endian
- Width-conversions with same endianness as host are *free*

Common transfers can use memcpy, width conversions don't modify transaction

Little-endian host



Big-endian host



Part-word Transfers

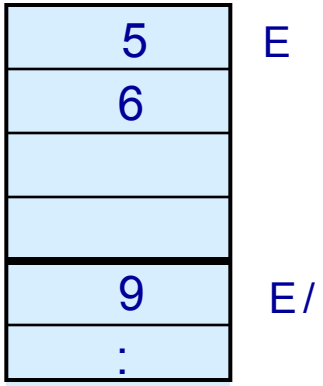
Pmpi1irhrerls

[A

pirkl A:

ehhvi AE

hee A



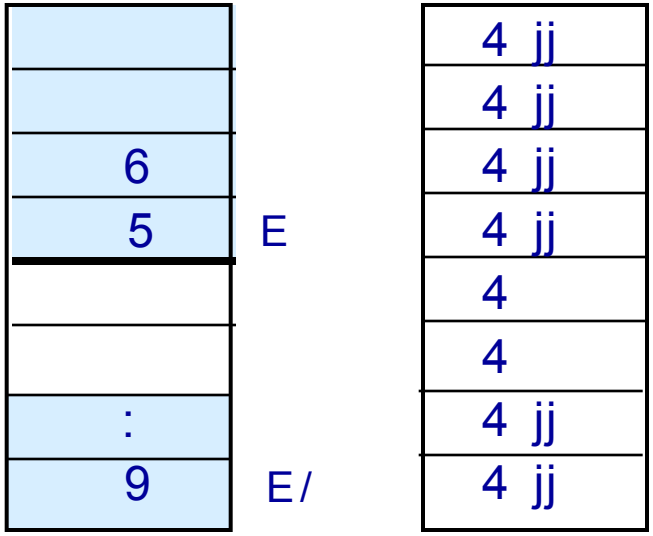
Fmk1irhrerls

[A

pirkl A:

ehhvi AE

hee A f i irefpi A



Generic Payload Extension Methods

- Generic payload has an array-of-pointers to extensions
- One pointer per extension type
- Every transaction can potentially carry every extension type
- Flexible mechanism

```
iqtpei   tireqi XB  X. set_extension , X. i  -
iqtpei   tireqi XB  X. set_auto_extension , X. i  -
iqtpei   tireqi XB  X. get_extension,- gsr
iqtpei   tireqi XB  smn clear_extension ,-
iqtpei   tireqi XB  smn release_extension ,-
```

Sticky extn

Freed by ref counting

Clears pointer, not extn object

mm => convert to auto
no mm => free extn object

Extension Example

```
vg q ci ir rnsr > tlm_extension q ci ir rnsrB

q ci ir rnsr,- >rn,4-
 pqci ir rnsrcfe i. clone,- gsr  222
  rn ep snn copy_from, pqci ir rnsrcfe i gsr  *i  -  222
 rn rn

222

pqckiri vgcte pseh.  ver  Aqiqcqkv1Bepgsgei,-
ver 1Bacquire,-

q ci ir rnsr. i  Ari  q ci ir rnsr
i 1Brn A5
ver 2set_extension, i  -

sgoi 1Brfc ver tsvcj ,. ver 0tl e i 0hi pe -
ver 2release_extension q ci ir rnsrB,-

ver 1Brelease,-
```

User-defined extension

Pure virtual methods

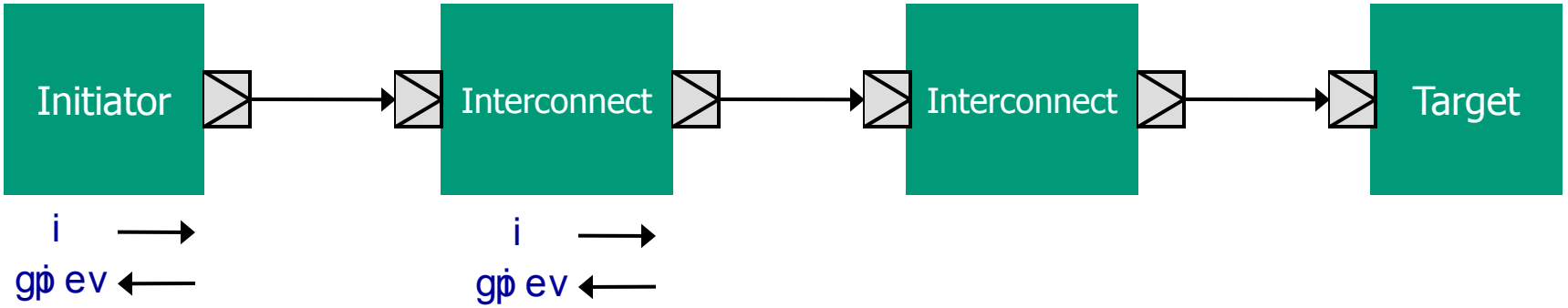
Heap allocation

Reference counting

Freed when ref count = 0

Trans and extn freed

Extension Rules



- Extensions should only be used downstream of the setter
- Whoever sets the extension should clear the extension
- If not reference counting, use set_extension / clear_extension
- If reference counting, use set_auto_extension
- For sticky extensions, use set_extension
- Within b_transport, either check or use set_extension / release_extension

Instance-Specific Extensions

```
mgphi &pqc_mp3mn_ergic_tigrjmgci_ir_msr_2&
```

```
vg_q_ci_r>pqc_mp>>instance_specific_extension_q_ci_rB
mn_r_q
```

```
gpe_Mivgsrrig>gcqsh_pi
  pqc_mp>>instance_specific_extension_accessor_eggi_sv

  mn_ep_pq>>pqc_rgcir_q_rfc_ver_tsvcj,_222-

    q_ci_r.i_r
    eggi_sv,_ver_-2get_extension,i_r-
    rjn,i_r-
      gs_i_r1Br_q_irhp
      eggi_sv,_ver_-2clear_extension,i_r-
     ipi
      i_r_Ari_q_ci_r
      i_r1Br_q_Ags_r//
      eggi_sv,_ver_-2set_extension,i_r-
      2222
```

Gives unique extensions per module instance

SWGMXPQ1621

THE BASE PROTOCOL

- tlm_phase
- Base protocol rules
- Base protocol phases
- Defining new protocol types

Base Protocol - Coding Styles

- Loosely-timed is typically
  - Blocking transport interface, forward and return path
  - 2 timing points
  - Temporal decoupling and the quantum keeper
  - Direct memory interface

- Approximately-timed is typically
  - Non-blocking transport interface, forward and backward paths
  - 4 phases
  - Payload event queues

- Loosely-timed and approximately-timed are only coding styles
- The base protocol defines rules for phases and call order

Base Protocol and tlm_phase

- The base protocol = tlm_generic_payload + tlm_phase
- tlm_phase has 4 phases, but can be extended to add new phases

```
ir q tlm_phase_enum  YRNRXMPMIHcTLEW A40
          FIKNRc IUA50IRHc IU0FIKNRc IWT0IRHc IWT

gpe  tlm_phase
t fpg>
   pqctlei,-
   pqctlei, r mkrih mn mn-
   pqctlei,gsr  pqctleicir q*  erhevh-
   pqctlei* stivesvA,gsr  pqctleicir q*  erhevh-
  stivesv r mkrihmn,-gsr


 hijmni DECLARE_EXTENDED_PHASE,reqicevk-
gpe  pqctleic reqicevk>t fpg pq>>pqctlei
222
```


Base Protocol Rules 1

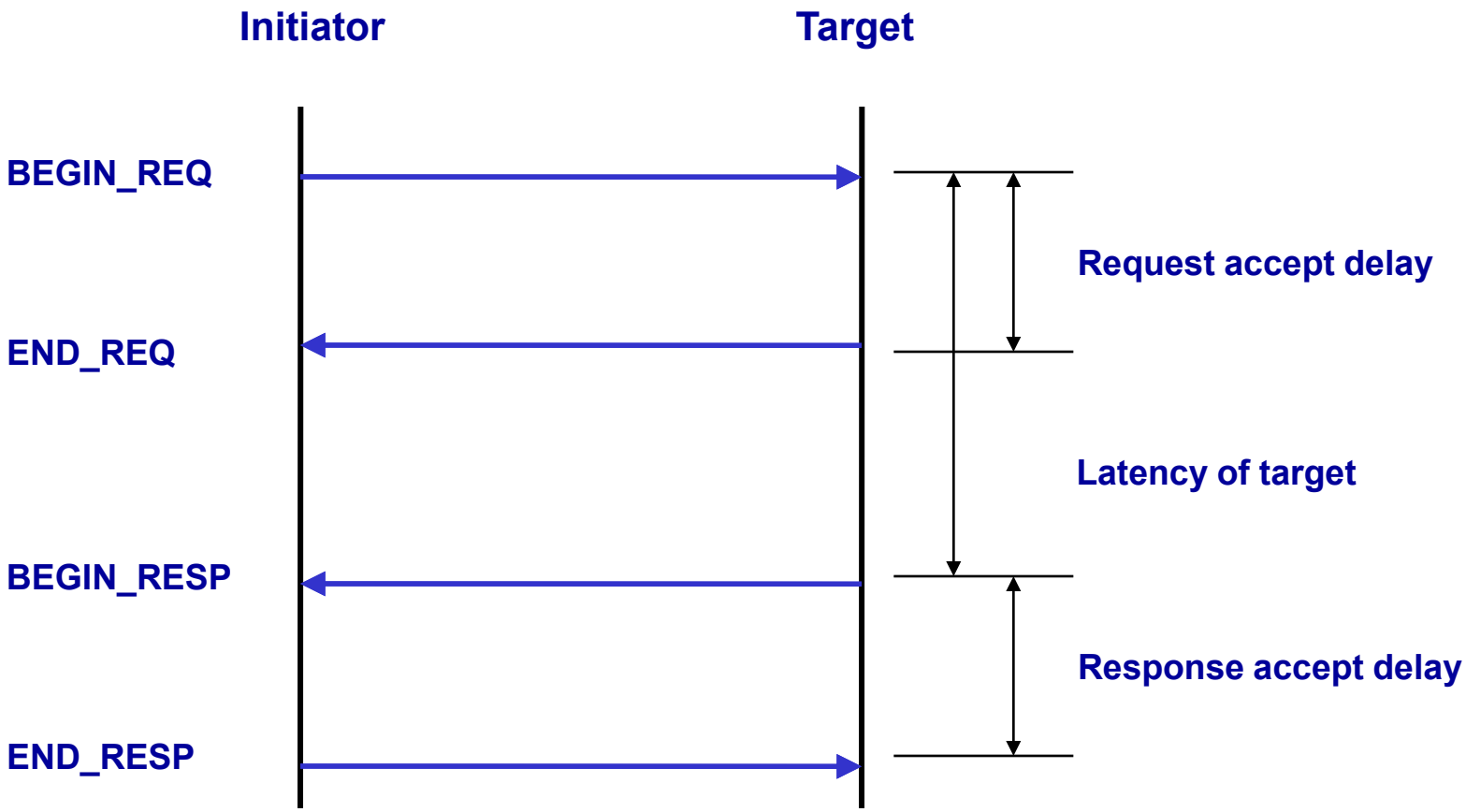
- Base protocol phases
  - BEGIN_REQ → END_REQ → BEGIN_RESP → END_RESP
  - Must occur in non-decreasing simulation time order
  - Only permitted one outstanding request or response per socket
  - Phase must change with each call (other than ignorable phases)
  - May complete early

- Generic payload memory management rules
- Extensions must be ignorable
- Target is obliged to handle mixed b_transport / nb_transport
- Write response must come from target

Base Protocol Rules 2

- Timing annotation on successive calls to nb_transport
  - for a given transaction, must be non-decreasing
  - for different transactions, mutual order is unconstrained
- Timing annotation on successive calls to b_transport
  - order is unconstrained (loosely-timed)
- b_transport does not interact with phases
- b_transport is re-entrant
- For a given transaction, b_transport / nb_transport must not overlap

Approximately-timed Timing Parameters



BEGIN_REQ must wait for previous END_REQ, BEGIN_RESP for END_RESP

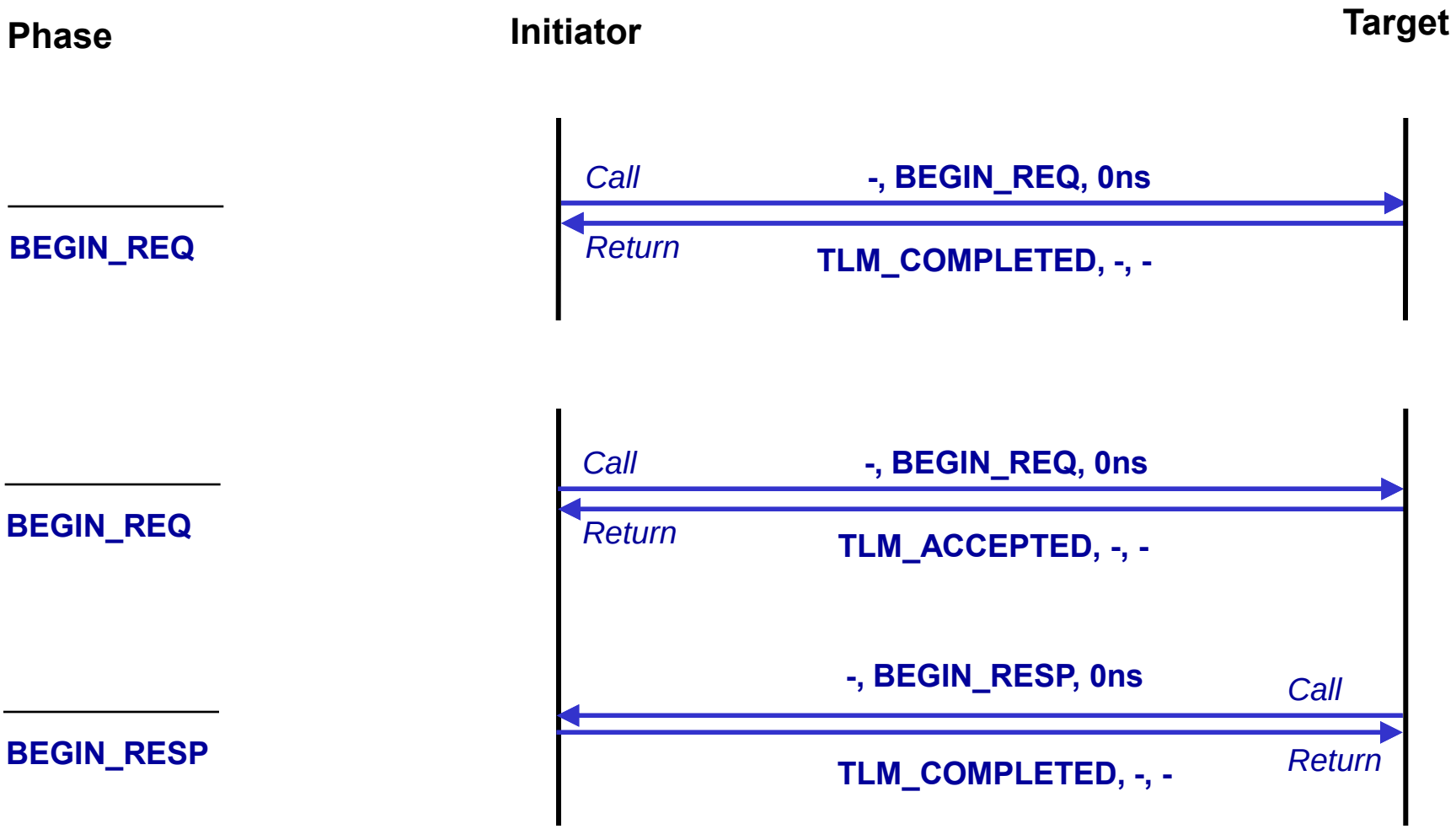
Pre-emption and Early Completion

- Permitted phase transition sequences
  - BEGIN_REQ
  - BEGIN_REQ (→ END_REQ) → BEGIN_RESP
  - BEGIN_REQ → END_REQ → BEGIN_RESP
  - BEGIN_REQ (→ END_REQ) → BEGIN_RESP → END_RESP
  - BEGIN_REQ → END_REQ → BEGIN_RESP → END_RESP

- Initiator sends BEGIN_REQ and END_RESP
- Target sends END_REQ and BEGIN_RESP

Transaction completes early if nb_transport returns TLM_COMPLETED

Examples of Early Completion



Transaction Types

- **Only three recommended alternatives**
  - **Use the base protocol directly (with ignorable extensions)**

    Excellent interoperability

  - **Define a new protocol type class with a typedef for tlm_generic_payload**

    Do whatever you like with extensions

  - **Define a new transaction type unrelated to the generic payload**

    Sacrifice interoperability; you are on your own

Protocol Types Class

v g **tlm_base_protocol_types**

tihij pqckiri vrgcte pseh **tlm_payload_type**
tihij pqctle i **tlm_phase_type**

iqtpei tireqi X] TI WA pqcfe ictvs sgspc ti B
gpe **tlm_fw_transport_if**
>t fpvg nv ep pqcj crsr fpsgomkc ver tsvcrjn tireqi X] TI W>**tlm_payload_type**0
tireqi X] TI W>**tlm_phase_type**B
0t fpvg nv ep pqcfpsgomkc ver tsvcrjn tireqi X] TI W>**tlm_payload_type**B
0t fpvg nv ep pqcj chnigcqiqcrjn tireqi X] TI W>**tlm_payload_type**B
0t fpvg nv ep pqc ver tsvchfkcrjn tireqi X] TI W>**tlm_payload_type**B

iqtpei tireqi X] TI WA pqcfe ictvs sgspc ti B
gpe **tlm_bw_transport_if**
222

Defining a New Protocol Types Class

```
pqcmnmmasvc sgoi  B sgoi 5
```

1. Use tlm_base_protocol_types

```
vg  my_protocol_types
```

2. Use new protocol based on generic payload

```
  tihij pqckirivgcte pseh   pqcte psehc ti
  tihij pqctlei             pqctleic ti

pqcmnmmasvc sgoi    60my_protocol_types B sgoi 6
```

```
vg  custom_protocol_types
```

3. Use new protocol unrelated to generic payload

```
  tihijq cte pseh           pqcte psehc ti
  tihijq ctlei              pqctleic ti

pqcmnmmasvc sgoi    60custom_protocol_types B sgoi
```

Extended Protocol Example 1

```
33Y ivlhijmnihi ir rsar gpe
  v g MVgvcgqhci ir rsar: tlm::tlm_extension MVgvcgqhci ir rsarB

    m ep pqci ir rsarcfe i. clone,-gsr
      MVgvcgqhci ir rsar. Ari MVgvcgqhci ir rsar
       1Bmngvcgqh A l m1Bmngvcgqh
      vi vr

    m ep sm copy_from, pqci ir rsarcfe i gsr * jvsq -
      mngvcgqh A e mgcge MVgvcgqhci ir rsar gsr *B,jvsq-2mngvcgqh

  MVgvcgqhci ir rsar,->mngvcgqh,jepi-
  fsspmngvcgqh


  v g incr_payload_types

    tihij pq>>pqckirivmgcte pseh pqcte psehc ti
    tihij pq>>pqctlei pqctleic ti
```

User-defined protocol types class using the generic payload

Extended Protocol Example 2

```
v g  Mmme s v> gc q sh  pi

 pq c   mp ≫ mq t pi c mnme s vc  s goi    Mmme s v0  60 incr_payload_types B  init_socket;
22
 s mn   l vi ehc t vs gi     ,-

  tlm::tlm_generic_payload   ver
  22
  Mgvc gq hci   i r  msr.  mgvc gq hci   i r  msr  A r i    Mgvc gq hci   i r  msr
   ver  2set_extension, mgvc gq hci   i r  msr -
  22
   ver  2i  c gsq q er h,  pq ≫XPQc[    MXI c GS QQERH -
  mnmc  s goi  1Bf c  ver  t s v,  ver  0hi pe   -
  22
   ver  2i  c gsq q er h,  pq ≫XPQc MKRS   I c GS QQERH -
  mgvc gq hci   i r  msr 1Bmgvc gq h  A  v i
  mnmc  s goi  1Bf c  ver  t s v,  ver  0hi pe   -
```

Extended Protocol Example 3

```
33Xli evki

pqc mp>>mqtpic evki c sgoi  Qi q sv0 60 incr_payload_types B targ_socket

 mi ep smn fc ver tsv, pq>>pqckiri vmgcte pseh*  ver 0 gcgsvi>>gc mqi*  -

  pq>>pqcgsqqerh gqh A ver 2ki cgsqqerh,-
  222
  Mgvcgqhci ir msr. mgvcgqhci ir msr
  ver 2get_extension( incr_cmd_extension );

  rjn, mgvcgqhci ir msr1Bmgvcgqh -
    rjn, gqh %A pq>>XPQcMKRS IcGSQQERH -
       ver 2i cvi tsr ic e , pq>>XPQcKIRI MGcI S c IWTSRW -
       vi vr

    // qc sveki_ehva

  222
```

Assume the extension exists

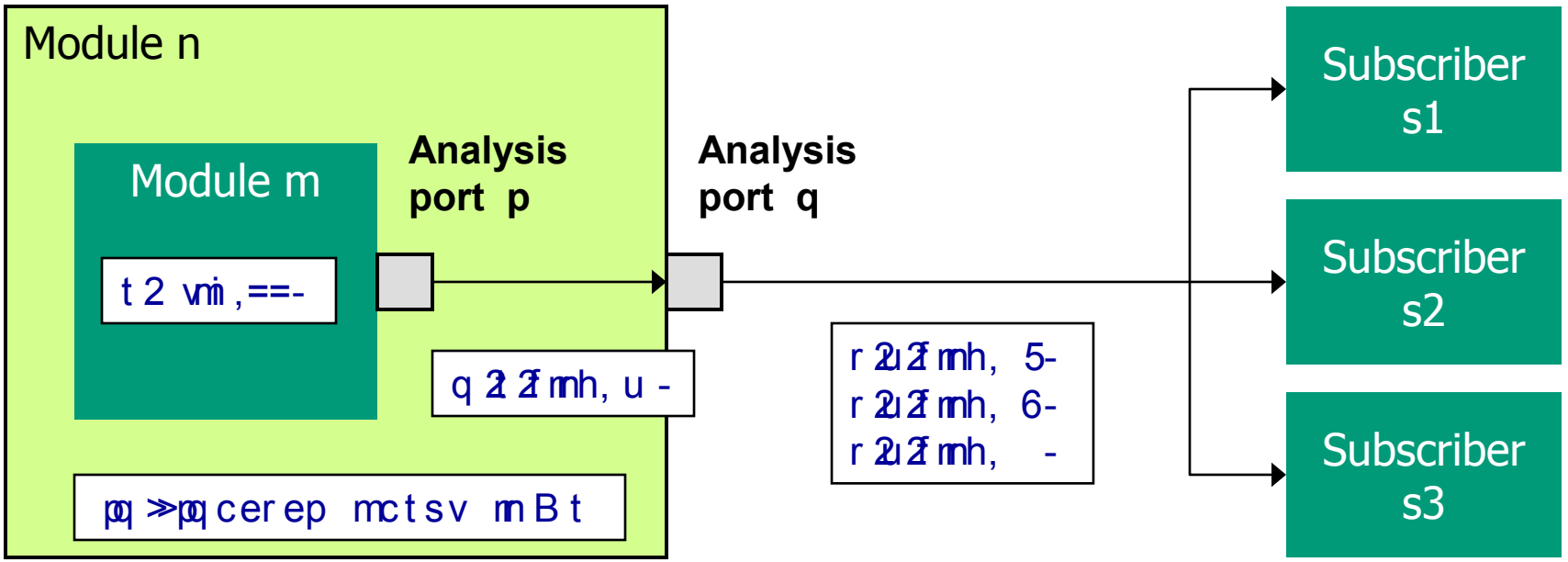
Detect clash with read or write

SWGMXPQ1621

ANALYSIS PORTS

- Analysis Interface and Ports

Analysis Ports




```
v g Wf gvrhi v> gcsfri g 0pq >>pq cer ep mcrjn mn B

Wf gvrhi v,gl ev. r - > gcsfri g,r -
 m ep smn vmi,gsr mn* - 222
```

Analysis port may be bound to 0, 1 or more subscribers

Analysis Interface

```
iqtpei   tireqi XB
gpe  tlm_write_if >t fpog mn ep gcgsvi ≫gcmnivjegi
t fpog>
   mn ep smn write,gsr   X*  -A4                                    "Non-negotiated"


iqtpei   tireqi XB
gpe  tlm_analysis_if >t fpog mn ep pqc  vmicrjn XB

gpe  tlm_analysis_port >t fpog gcgsvi ≫gcsfrig 0t fpog mn ep pqcerep mcrjn XB
t fpog>
   smn bind, pqcerep mcrjn XB *crjn-
   smn operator() , pqcerep mcrjn XB *crjn-
  fsspunbind, pqcerep mcrjn XB *crjn-

   smn write, gsr   X*  -
    jsv, mAqcmnivjegi 2fikmn,-  m?Aqcmnivjegi 2rh,-   mn/ -
      ,.m1B vmi,  -
```

write() sends transaction to every subscriber

Analysis Port Example

```
v g  W f  gvrhi v>  gcsfri g 0 tlm::tlm_analysis_if  Xver  B
W f  gvrhi v, gsr   gl ev r - > gcsfri g ,r -
 m  ep  sm n write, gsr   Xver  *   -
 gs      &L i ps0ks &    2m   &r &
```

```
WGcQSHYPI  ,Q-
  tlm::tlm_analysis_port  Xver  B et

  WGcGXS   ,Q- >et ,&et &
    WGcXL  I EH ,X-

   sm n X ,-
    Xver    A  ===
    et 2write,   -
```

```
WGcQSHYPI  ,Xst -
  Q. q
  W f  gvrhi v.    f  gvrhi v5
  W f  gvrhi v.    f  gvrhi v6
  WGcGXS   ,Xst -
    q  A r i   Q, &q &
      f  gvrhi v5  A r i   W f  gvrhi v, &  f  gvrhi v5&
      f  gvrhi v6  A r i   W f  gvrhi v, &  f  gvrhi v6&
    q 1Bet 2bind, .   f  gvrhi v5 -
    q 1Bet 2bind, .   f  gvrhi v6 -
```

Subscriber implements analysis interface, analysis port bound to subscriber

Summary: Key Features of TLM-2

- Transport interfaces with timing annotation and phases
- DMI and debug interfaces
- Loosely-timed coding style and temporal decoupling for simulation speed
- Approximately-timed coding style for timing accuracy
- Sockets for convenience and strong connection checking
- Generic payload for memory-mapped bus modeling
- Base protocol for interoperability between TL- models
- Extensions for flexibility of modeling

For further information visit

www.systemc.org