Track 3: ESL and SystemC

The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard

David C Black, Doulos

52 DESIGN AUTOMATION CONFERENCE

DOULOS THURSDAY IS TRAINING DAY

TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

IEEE Std 1666-2011

- Approved Sept 2011
- Available Jan 2012
- Combines SystemC and TLM-2.0
- Adds a formal definition for TLM-1
- New features

TLM 2.0 and the IEEE Std 1666-2011

Free LRM, courtesy of Accellera Systems Initiative

- http://standards.ieee.org/getieee/1666/download/1666-2011.pdf

Proof-of-concept implementation

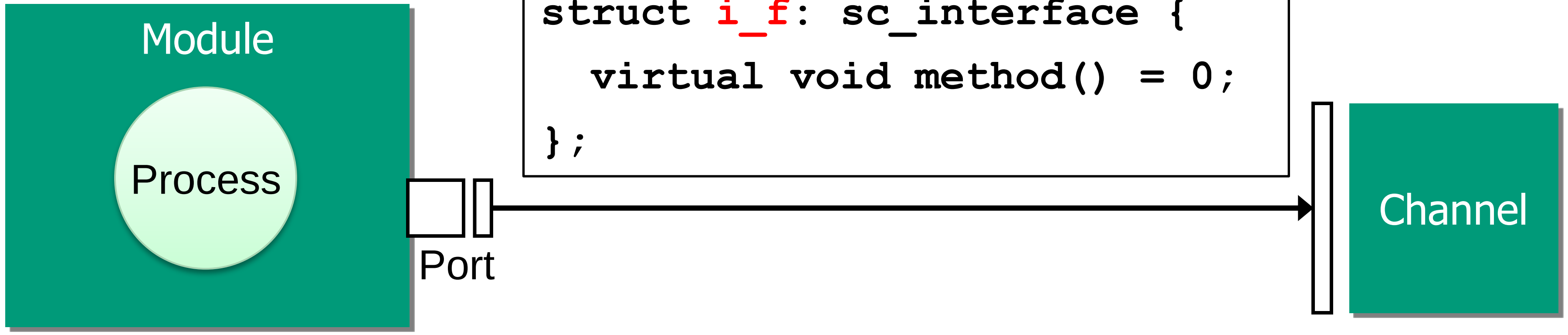
- http://www.accellera.org/downloads/standards/systemc

SystemC is

- Modules
- Ports
- Processes
- Channels
- Interfaces
- Events

- and the hardware data types

SystemC in Five Slides - 1

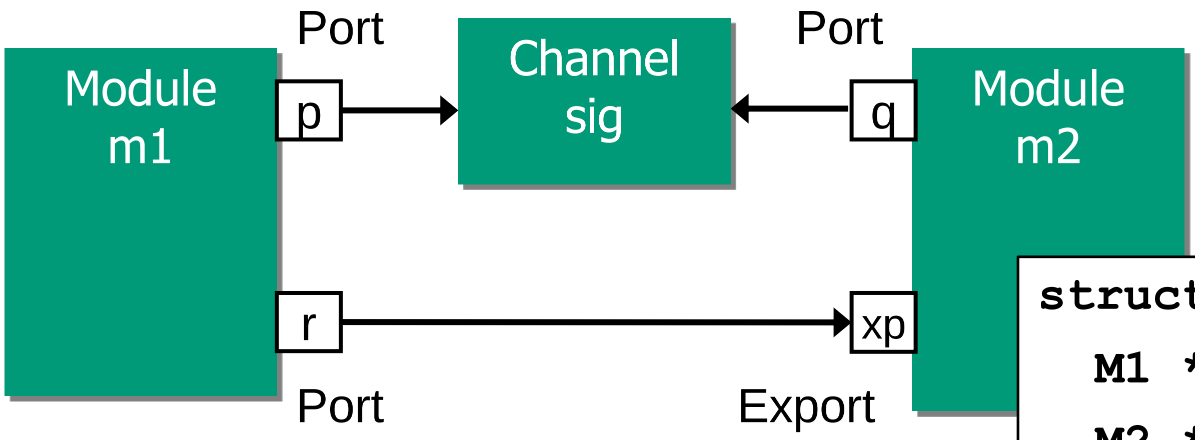


```
struct i_f: sc_interface {
  virtual void method() = 0;
};
```

```
struct M: sc_module {
  sc_port<i_f> p;

  M(sc_module_name n) {
    SC_HAS_PROCESS(M);
    SC_THREAD(process);
  }

  void process() {
    p->method();
  }
};
```

```
struct C: i_f, sc_module {

  C(sc_module_name n) {}

  void method() {...}
};
```

SystemC in Five Slides - 2



```
struct M1: sc_module {
  sc_out<bool> p;
  sc_port<i_f> r;
  ...
};
```

```
struct Top: sc_module {
  M1 *m1;
  M2 *m2;
  sc_signal<bool> sig;
  Top(sc_module_name n) {
    m1 = new M1("m1");
    m2 = new M2("m2");
    m1->p.bind(sig);
    m2->q.bind(sig);
    m1->r.bind(m2.xp);
  }
};
```

SystemC in Five Slides - 3

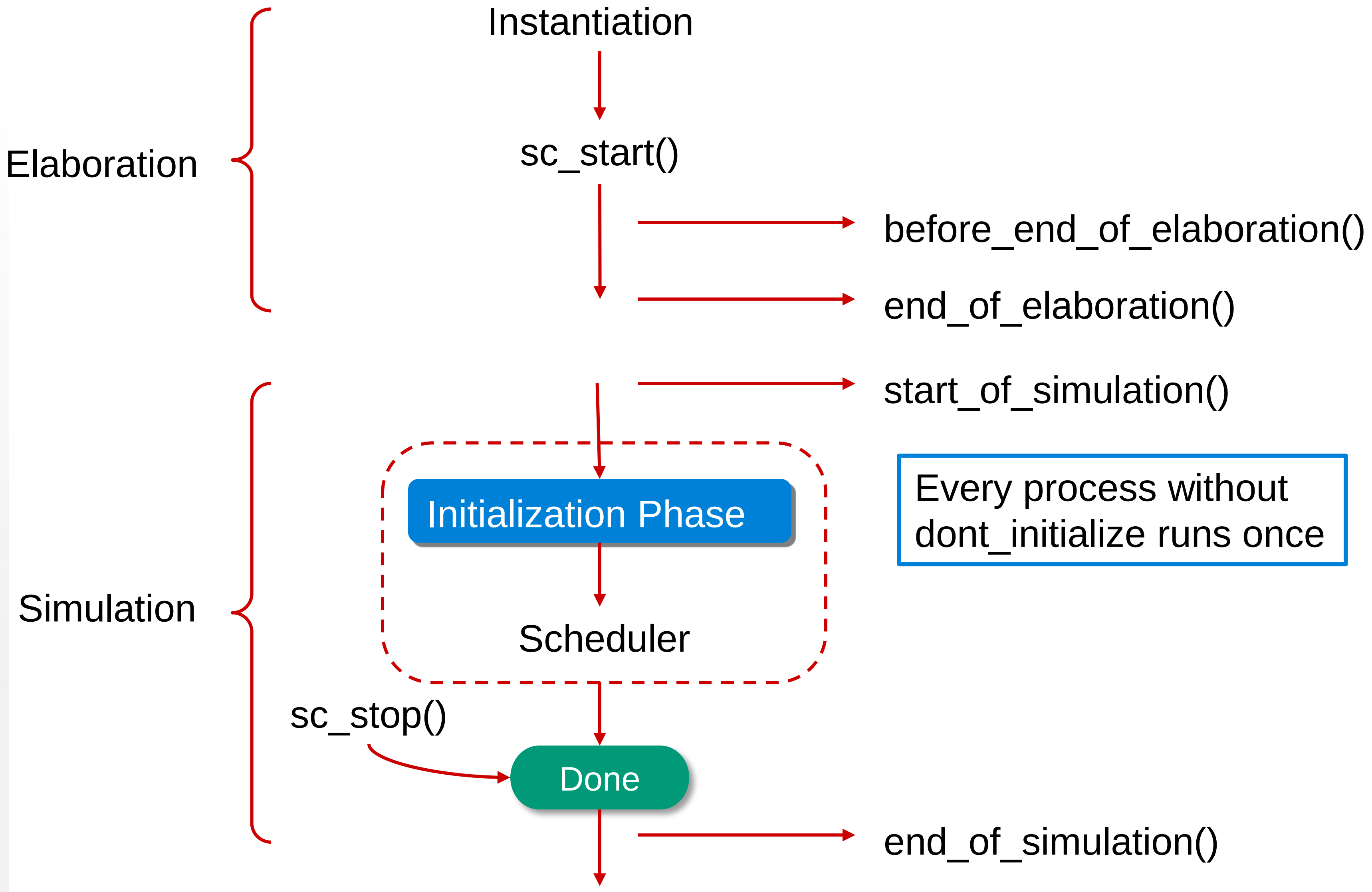
```
sc_in<bool> clk, reset, a, b;

M(sc_module_name n)
{
  SC_METHOD(method_proc);
    sensitive << reset;
    sensitive << clk.pos();
    dont_initialize();

  SC_THREAD(thread_proc);
    sensitive << a << b;
}
```

```
void method_proc() {
  if (reset)
    q = 0;
  else if (clk.posedge())
    q = d;
}
```

```
sc_event ev;
```

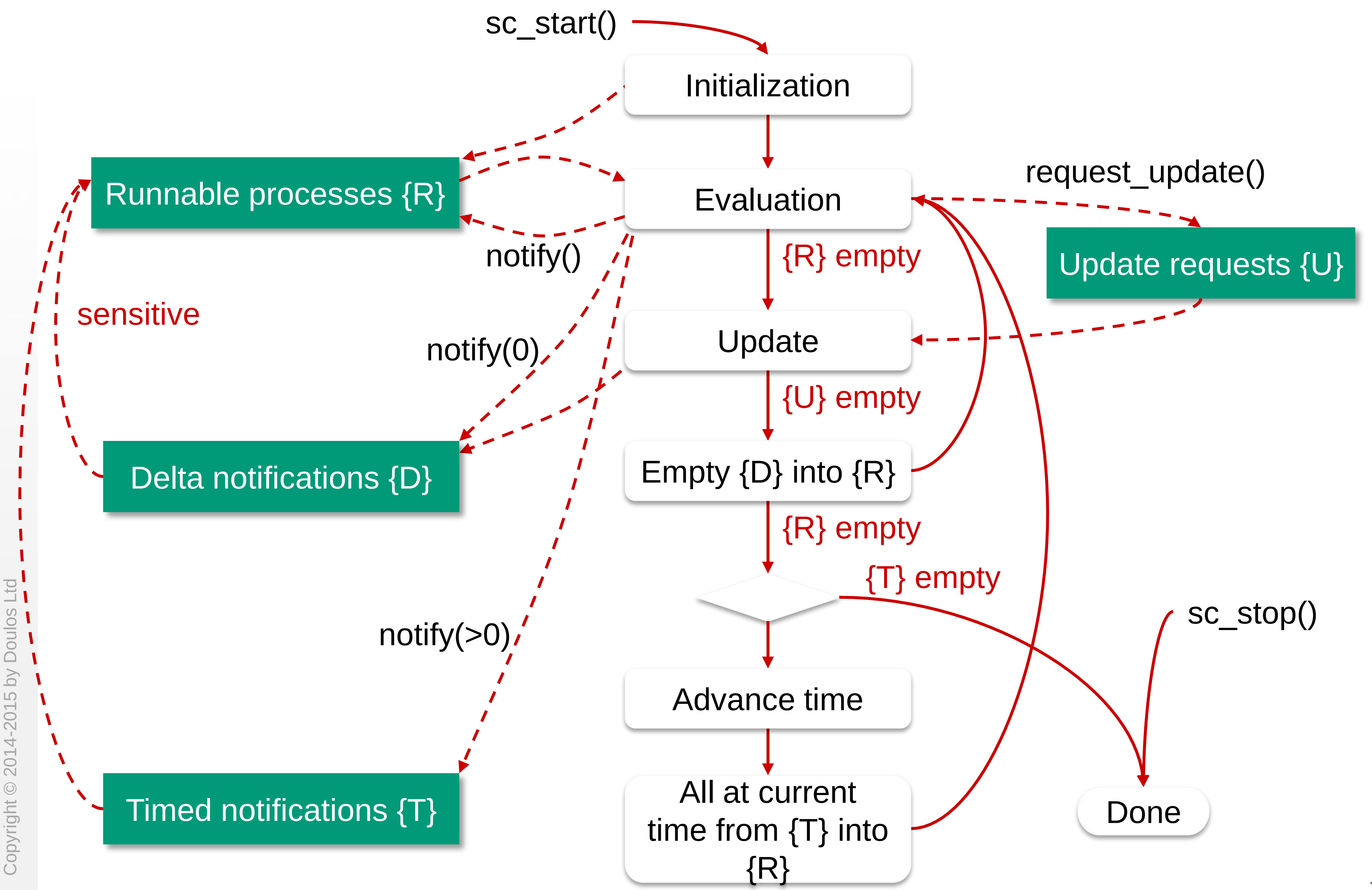
```
void thread_proc()
{
  while (1)
  {
    wait();
    ...
    wait(10, SC_NS);
    ev.notify(5, SC_NS);
    wait(ev);
    ...
  }
}
```

SystemC in Five Slides - 4



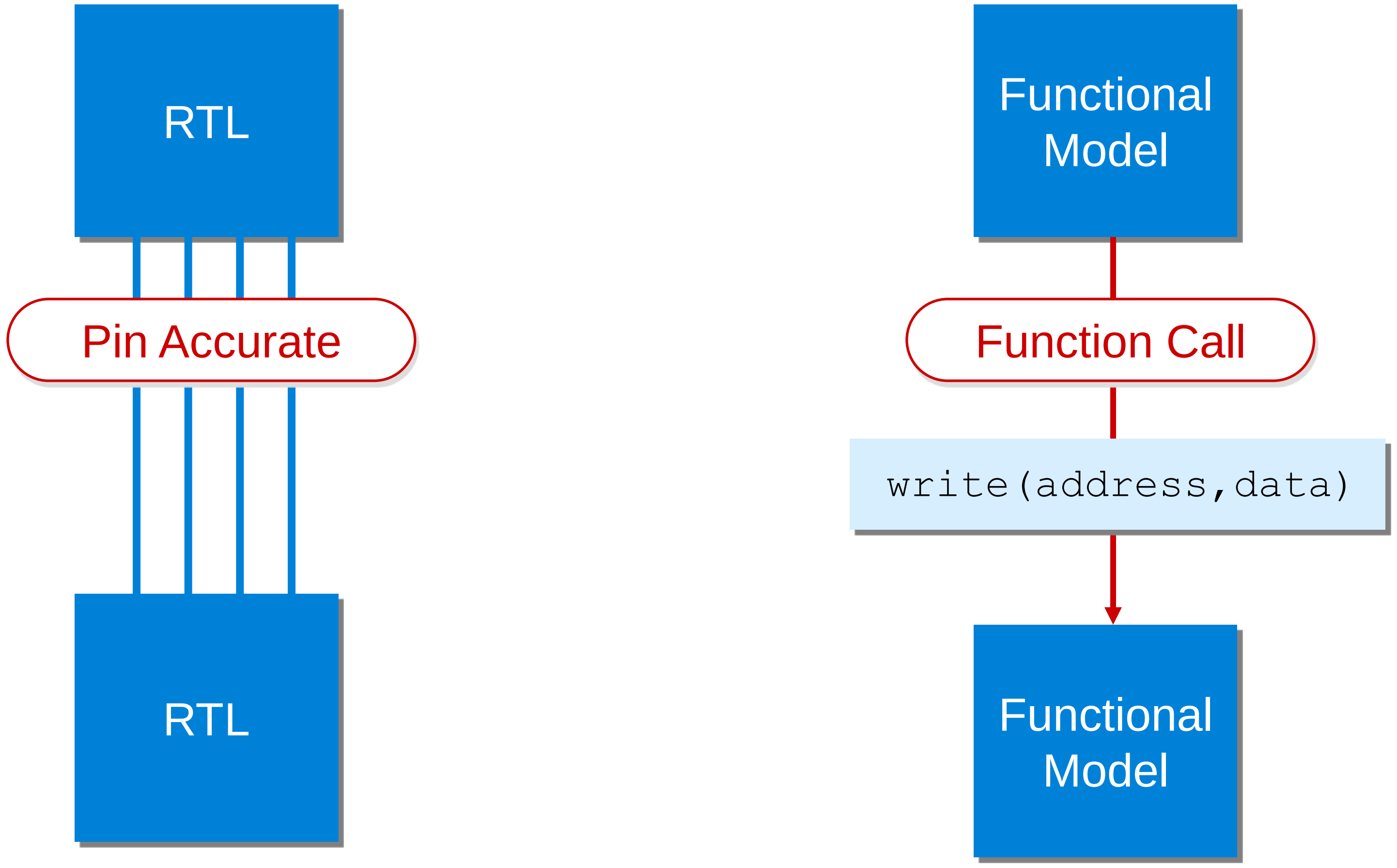
Every process without dont_initialize runs once

SystemC in Five Slides - 5



TLM-2.0 and IEEE 1666-2011

Transaction Level Modeling

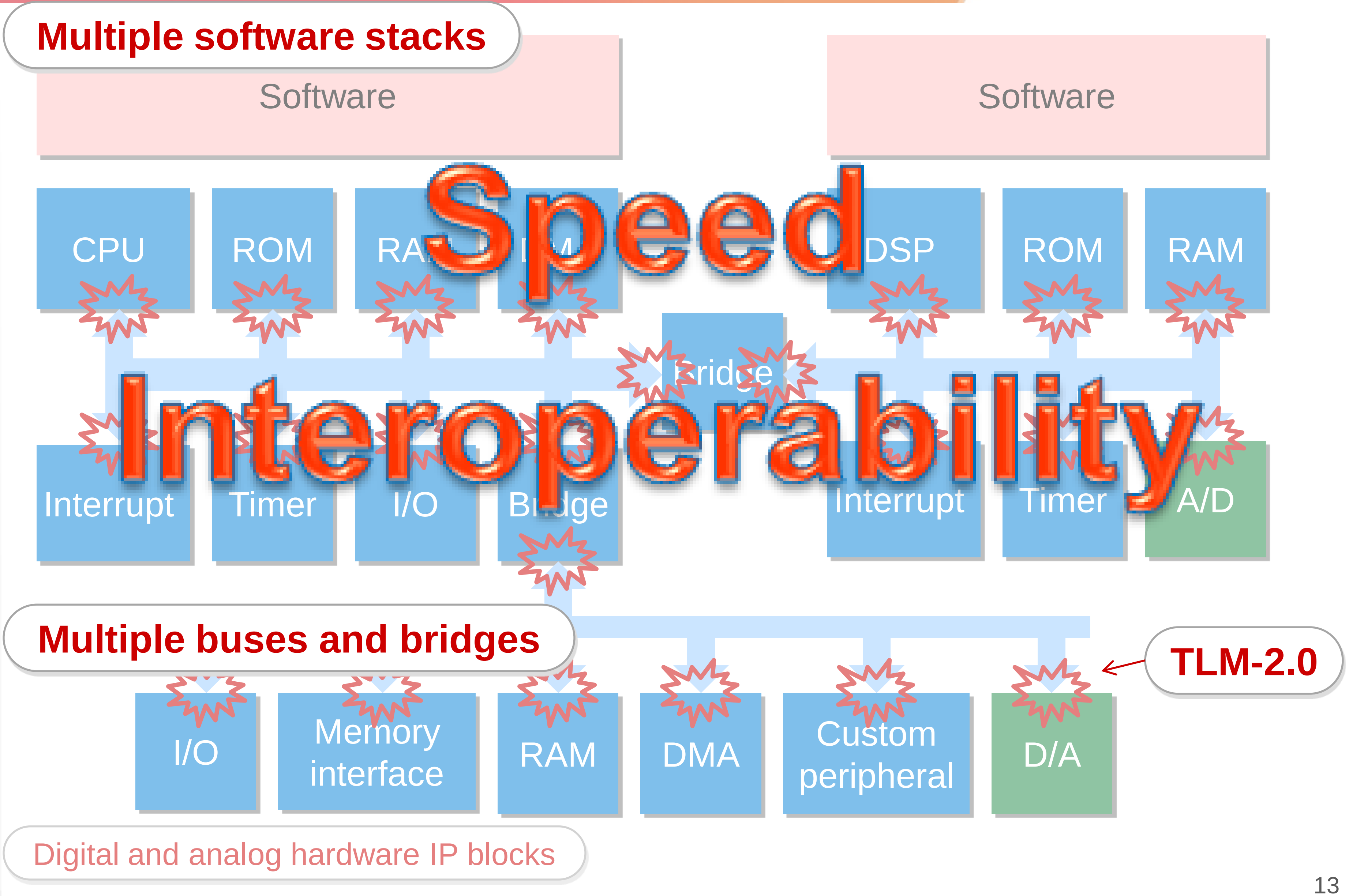


```
write(address,data)
```

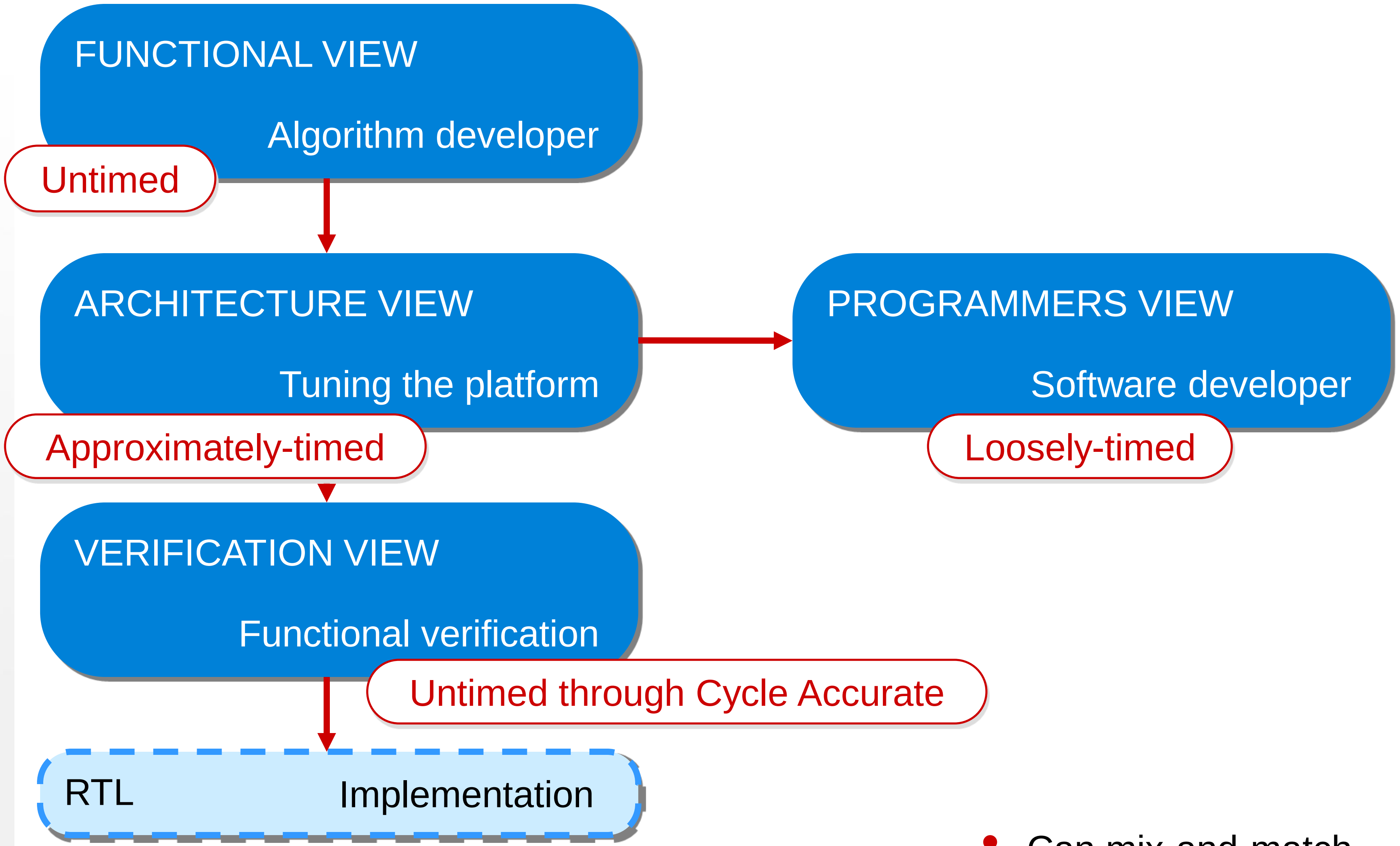
Simulate every event!

100-10,000 X faster simulation!

Virtual Platforms and TLM-2.0



Multiple Abstraction Levels



- Can mix-and-match

SystemC TLM Development

Apr 2005
- TLM-1.0
- put, get and transport
- Request-response model

Jun 2008
- TLM-2.0
- Pass-by-reference
- Unified interfaces
- Generic payload

July 2009
- TLM-2.0.1
- LRM

2011
- TLM-1 and TLM-2.0 part of IEEE 1666

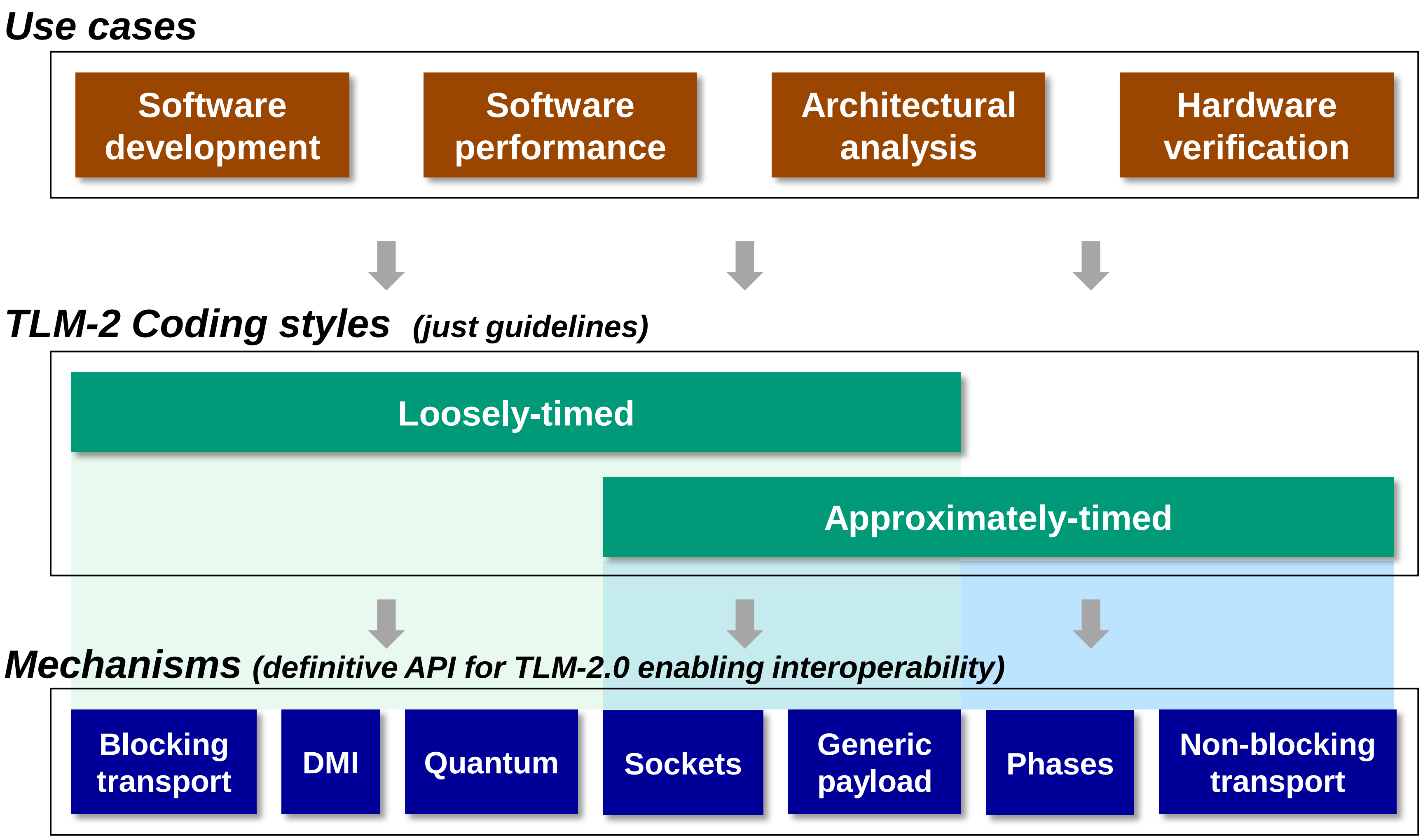
- As an aside, beyond SystemC ...

Jan 2008
- SystemVerilog OVM

Feb 2011
- SystemVerilog UVM

TLM-2.0 and IEEE 1666-2011

Use Cases, Coding Styles and Mechanisms



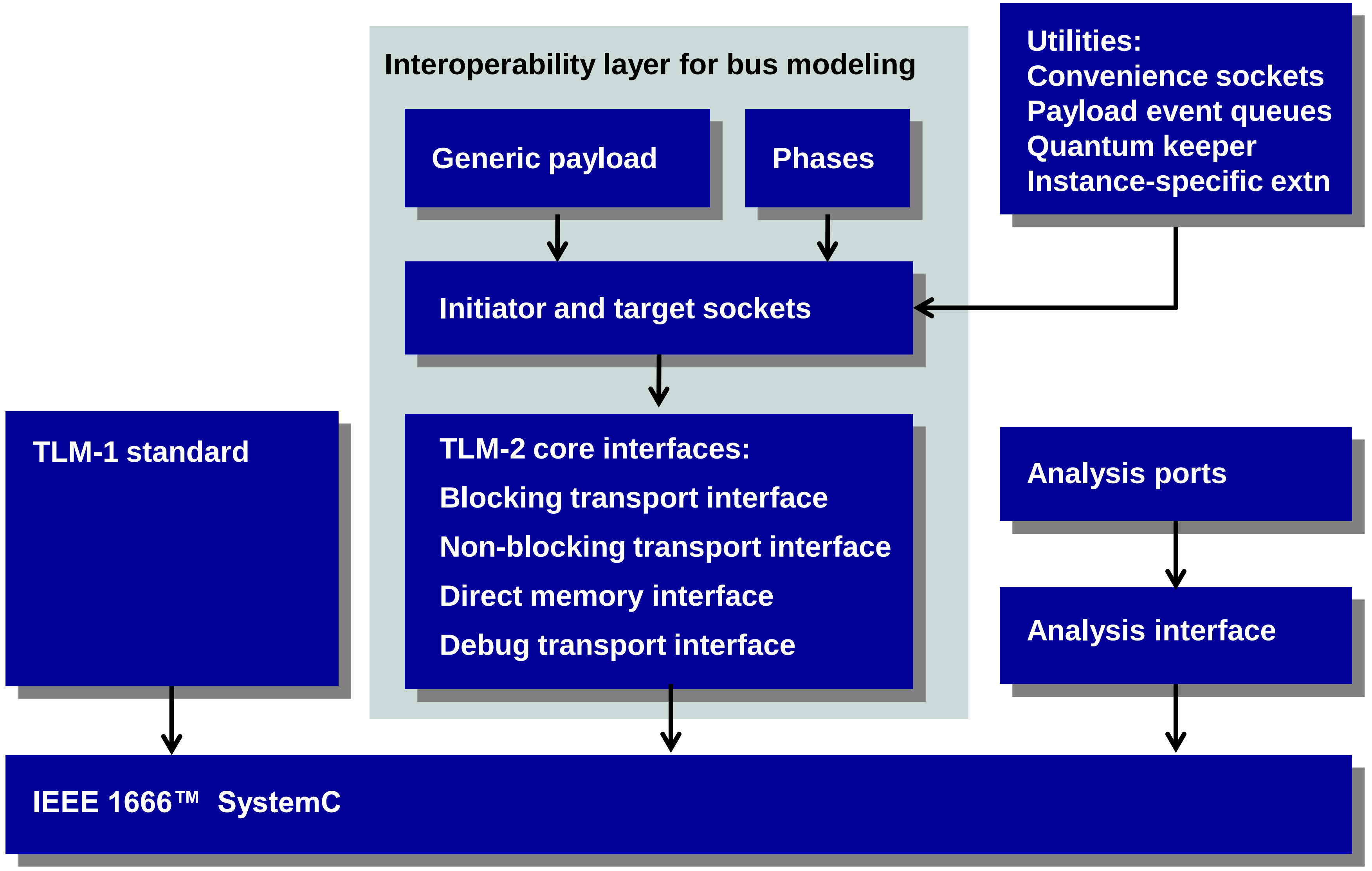
Coding Styles

- Loosely-timed = as fast as possible
  - Register-accurate
  - Only sufficient timing detail to boot O/S and run multi-core systems
  - b_transport – each transaction completes in one function call
  - Temporal decoupling
  - Direct memory interface (DMI)

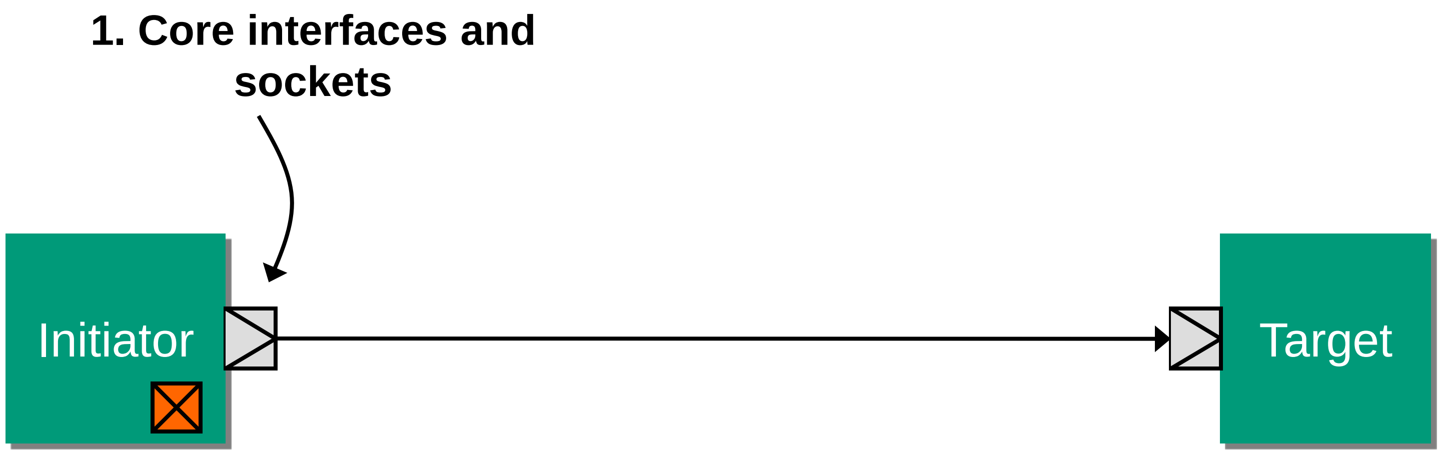
- Approximately-timed = just accurate enough for performance modeling
  - *aka* cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - nb_transport – each transaction has 4 timing points (extensible)

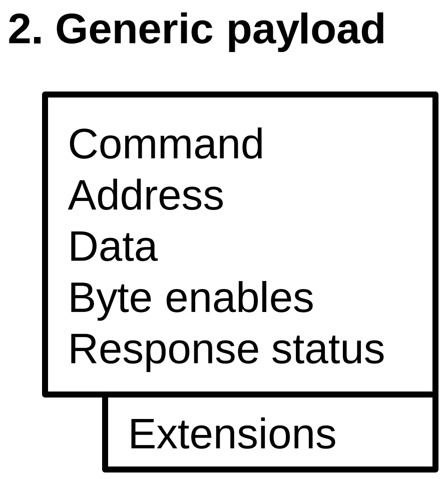
- Guidelines only – not definitive

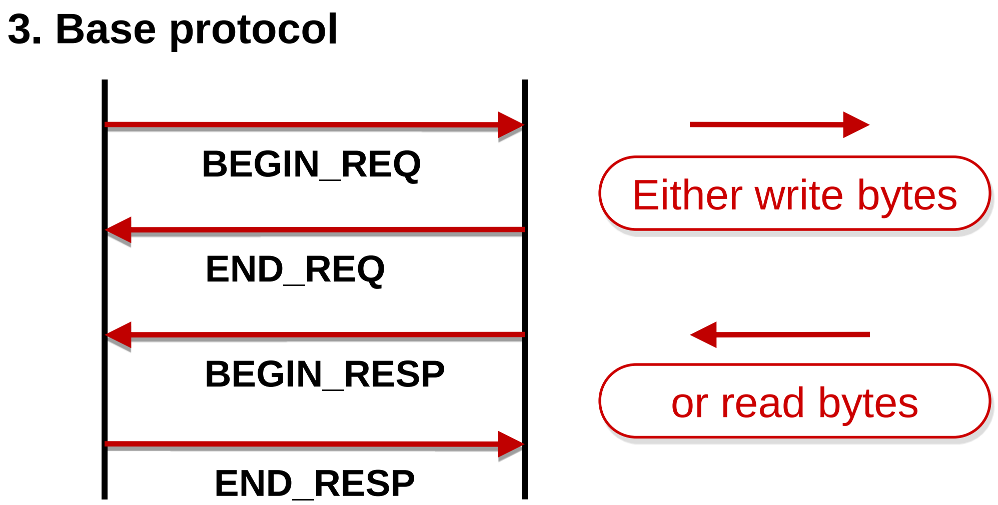
The TLM 2.0 Classes



Interoperability Layer

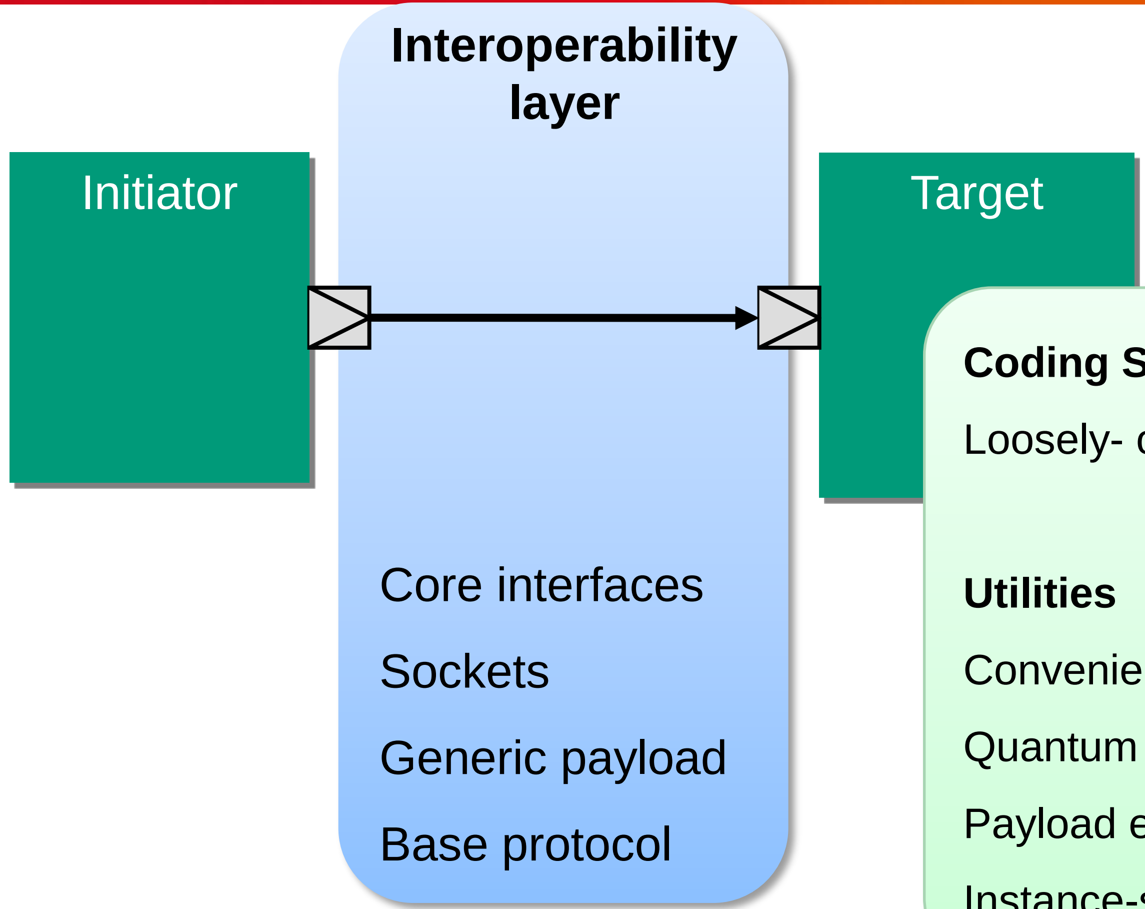






- Maximal interoperability for memory-mapped bus models

Utilities



Core interfaces

Sockets

Generic payload

Base protocol

Coding Style

Loosely- or Approximately-timed

Utilities

Convenience sockets

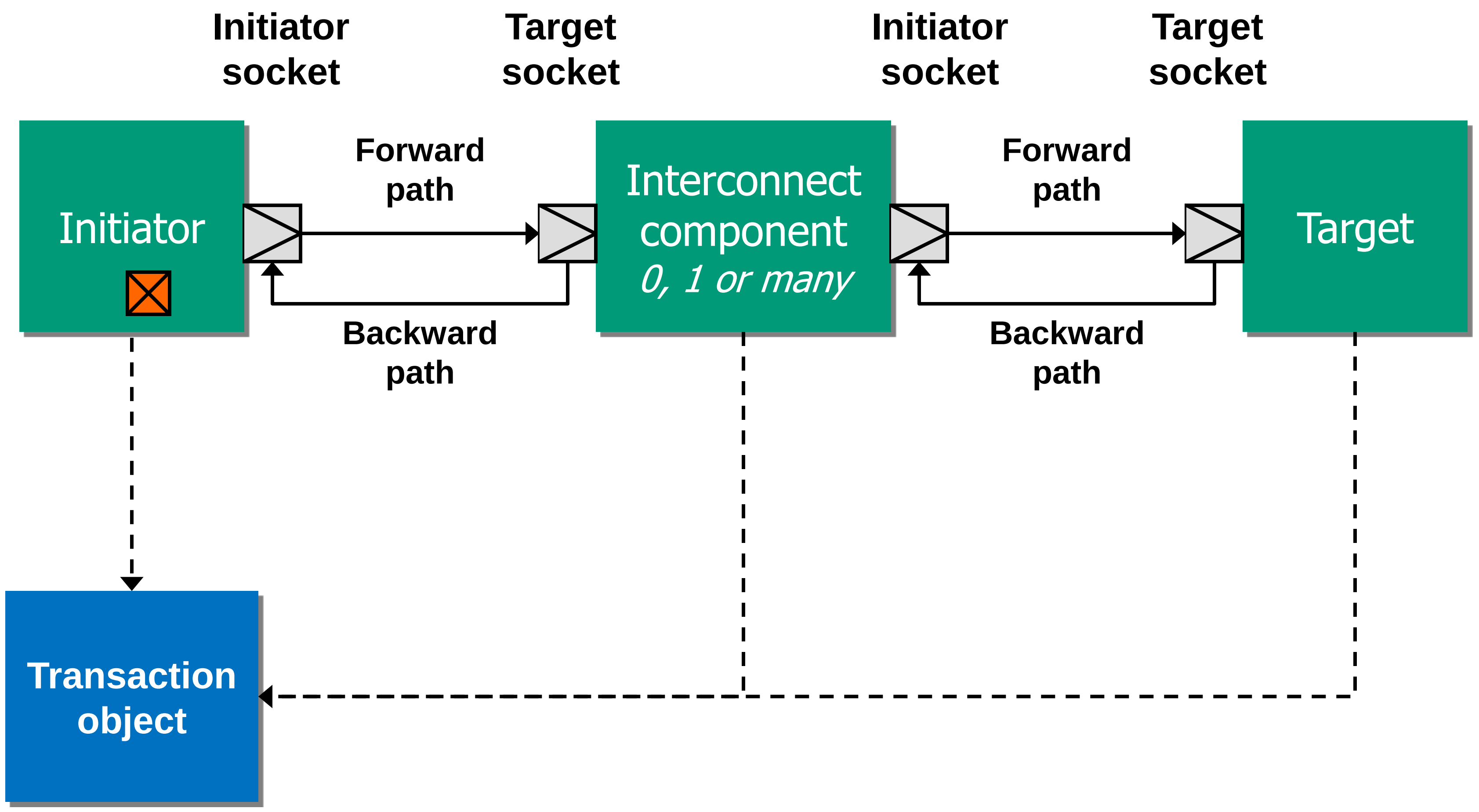
Quantum keeper (LT)

Payload event queues (AT)

Instance-specific extensions (GP)

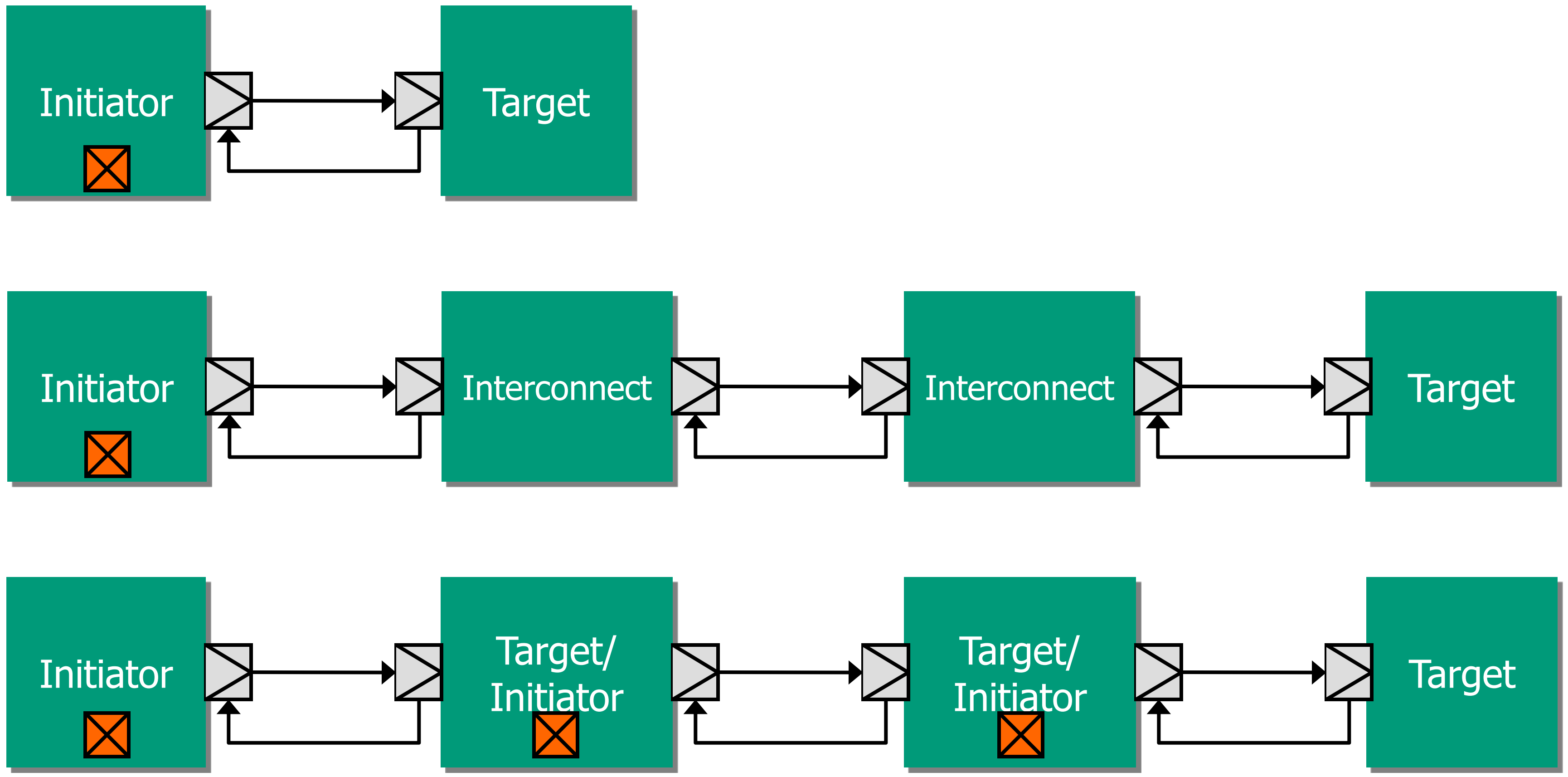
- Productivity
- Shortened learning curve
- Consistent coding style

Initiators, Targets and Interconnect



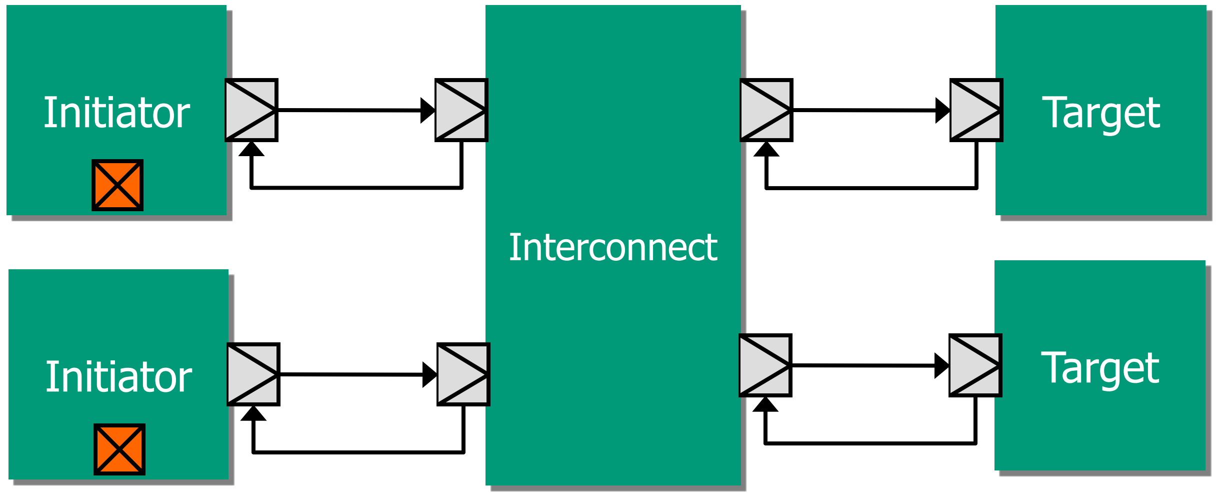
- Single transaction object for request and response
- References to the object are passed along the forward and backward paths

TLM-2 Connectivity



- Roles are dynamic; a component can choose whether to act as interconnect or target
- Transaction memory management needed

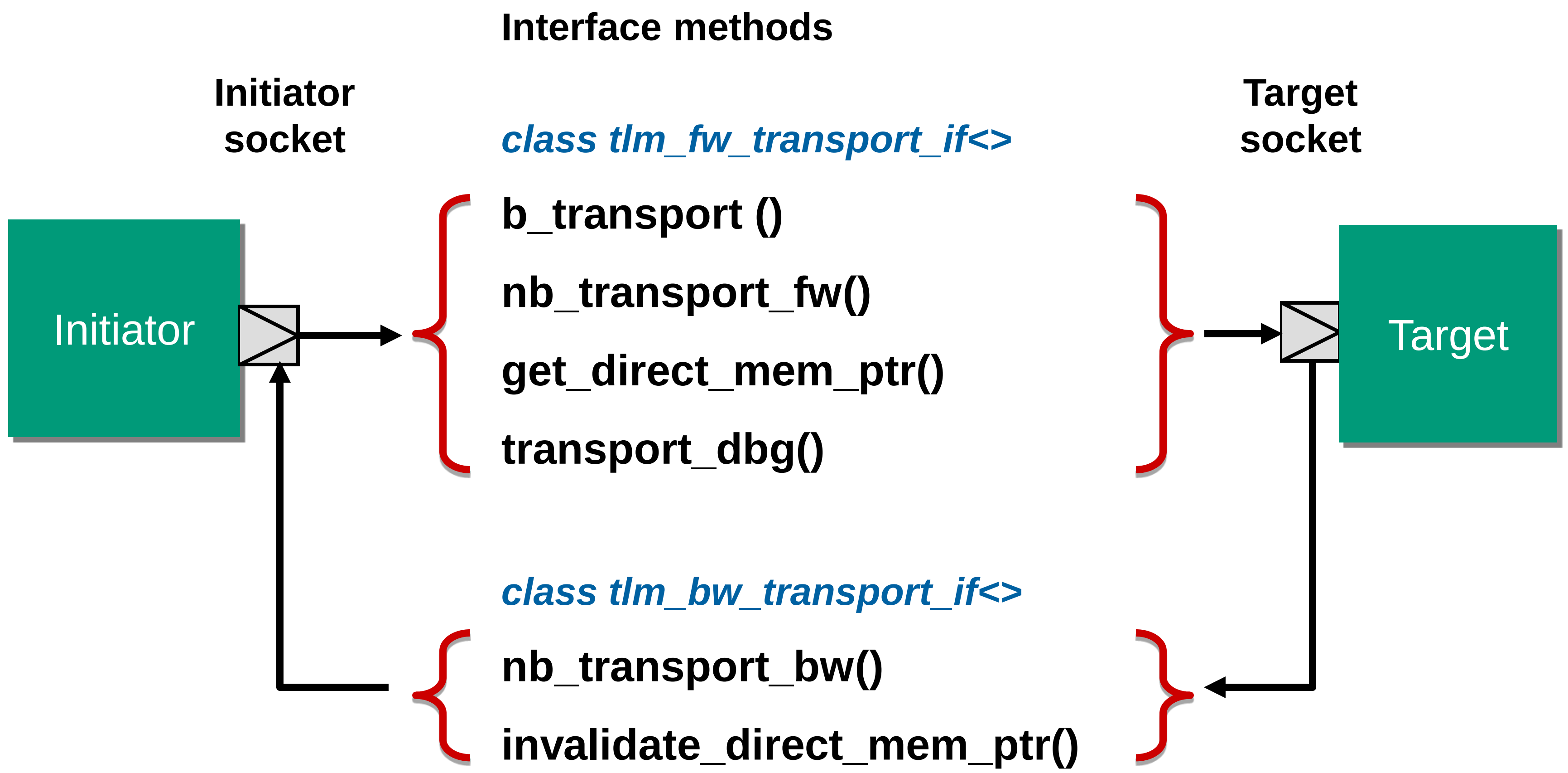
Convergent Paths



- Paths not predefined; routing may depend on transaction attributes (e.g. address)
- Whether arbitration is needed depends on the coding style

TLM-2.0 and IEEE 1666-2011

Initiator and Target Sockets



- Sockets provide fw and bw paths, and group interfaces

Initiator Socket

```
#include "tlm.h"
struct  Initiator: sc_module,  tlm::tlm_bw_transport_if<>
{
    tlm::tlm_initiator_socket<>  init_socket;

    SC_CTOR(Initiator) : init_socket("init_socket") {
        SC_THREAD(thread);
        init_socket.bind( *this );
    }

    void thread() { ...
        init_socket->b_transport( trans, delay );
        init_socket->nb_transport_fw( trans, phase, delay );
        init_socket->get_direct_mem_ptr( trans, dmi_data );
        init_socket->transport_dbg( trans );
    }

    virtual  tlm::tlm_sync_enum  nb_transport_bw( ... ) { ... }
    virtual void  invalidate_direct_mem_ptr( ... ) { ... }
};
```

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

tlm_initiator_socket must be bound to object that implements entire bw interface

Target Socket

```
struct  Target: sc_module,  tlm::tlm_fw_transport_if<>
{
    tlm::tlm_target_socket<>  targ_socket;

    SC_CTOR(Target) : targ_socket("targ_socket") {
        targ_socket.bind( *this );
    }
    virual void b_transport( ... ) { ... }
    virtual tlm::tlm_sync_enum  nb_transport_fw( ... ) { ... }
    virtual bool get_direct_mem_ptr( ... ) { ... }
    virtual unsigned int transport_dbg( ... ) { ... }
};
```

Combined interface required by socket

Protocol type default to base protocol

Target socket bound to target itself

Methods for forward path

tlm_target_socket must be bound to object that implements entire fw interface

Socket Binding

```
SC_MODULE(Top)  {
  Initiator  *init;
  Target    *targ;


  SC_CTOR(Top)  {
    init  = new Initiator("init");
    targ = new Target("targ");
    init->init_socket.bind( targ->targ_socket );
  }
  ...
```

Bind initiator socket to target socket

TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

The Generic Payload

- Has typical attributes of a memory-mapped bus

Attribute	Type	Modifiable?
Command	tlm_command	No
Address	uint64	Interconnect only
Data pointer	unsigned char*	No (array – yes)
Data length	unsigned int	No
Byte enable pointer	unsigned char*	No
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

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/ streaming width
TLM_BYTE_ENABLE_ERROR_RESPONSE	Unable to act on byte enable
TLM_GENERIC_ERROR_RESPONSE	Any other error

- Set to TLM_INCOMPLETE_RESPONSE by the initiator
- May be modified by the target
- Checked by the initiator when transaction is complete

Generic Payload - Initiator

```
void thread_process() {  // The initiator
    tlm::tlm_generic_payload  trans;
    sc_time  delay = SC_ZERO_TIME;

    trans.set_command( tlm::TLM_WRITE_COMMAND );
    trans.set_data_length( 4 );
    trans.set_streaming_width( 4 );
    trans.set_byte_enable_ptr( 0 );

    for ( int i = 0; i < RUN_LENGTH; i += 4 ) {
        int  word = i;
        trans.set_address( i );
        trans.set_data_ptr( (unsigned char*)( &word ) );
        trans.set_dmi_allowed( false );
        trans.set_response_status( tlm::TLM_INCOMPLETE_RESPONSE );

        init_socket->b_transport( trans, delay );

        if ( trans.is_response_error() )
            SC_REPORT_ERROR("TLM-2", trans.get_response_string().c_str());
        ...
}
```

Would usually pool transactions

8 attributes you must set

Generic Payload - Target

```
virtual  void  b_transport(  // The target
    tlm::tlm_generic_payload& trans,  sc_core::sc_time& t )
{
    tlm::tlm_command    cmd   = trans.get_command();
    sc_dt::uint64       adr   = trans.get_address();
    unsigned char*      ptr   = trans.get_data_ptr();
    unsigned int        len   = trans.get_data_length();
    unsigned char*      byt   = trans.get_byte_enable_ptr();
    unsigned int        wid   = trans.get_streaming_width();

    if ( byt != 0 || len > 4 || wid < len || adr+len > memsize ) {
        trans.set_response_status( tlm::TLM_GENERIC_ERROR_RESPONSE );
        return;
    }

    if ( cmd == tlm::TLM_WRITE_COMMAND )
        memcpy( &m_storage[adr], ptr, len );
    else if ( cmd == tlm::TLM_READ_COMMAND )
        memcpy( ptr, &m_storage[adr], len );

    trans.set_response_status( tlm::TLM_OK_RESPONSE );
}
```

6 attributes you must check

Target supports 1-word transfers

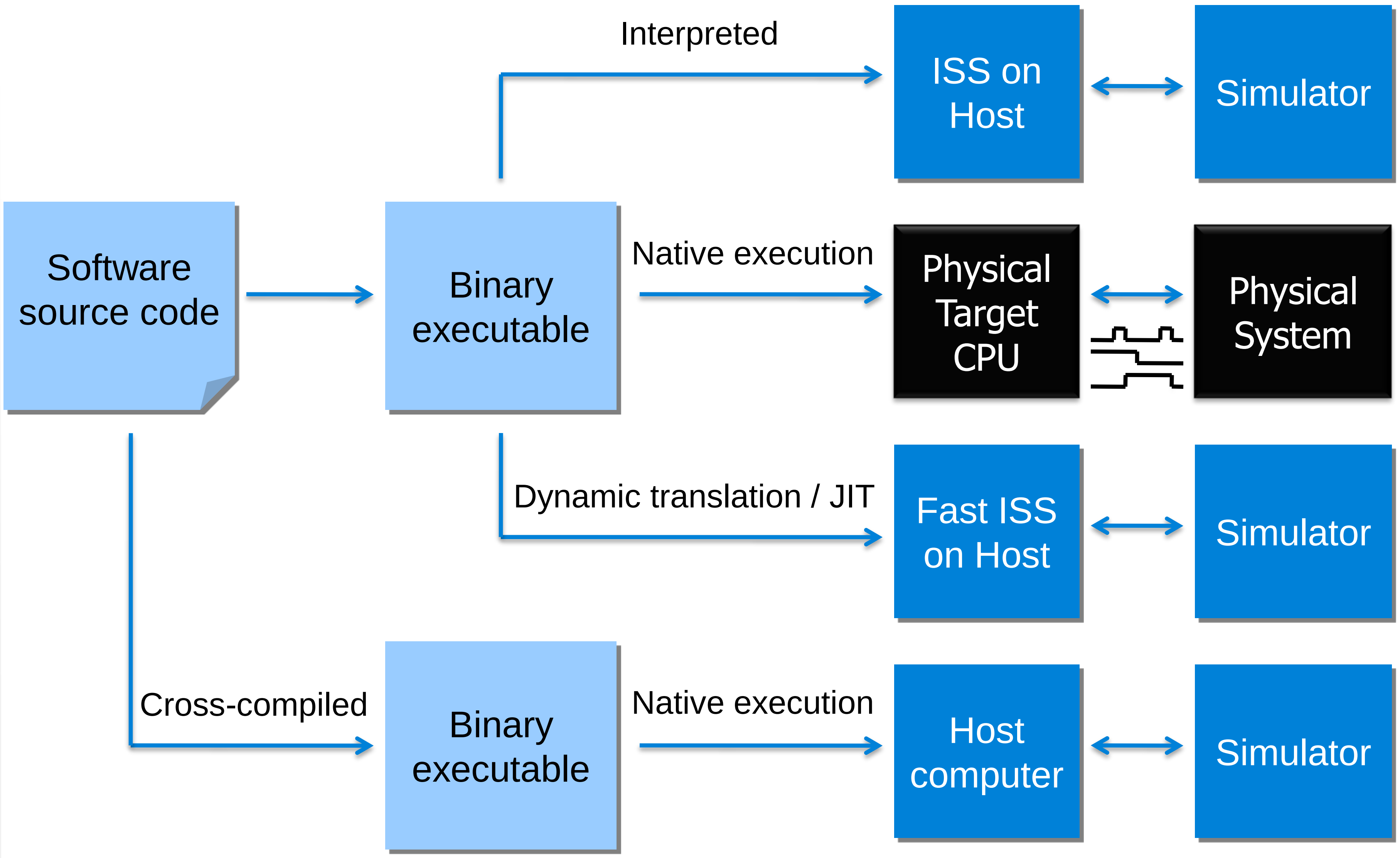
Execute command

Successful completion

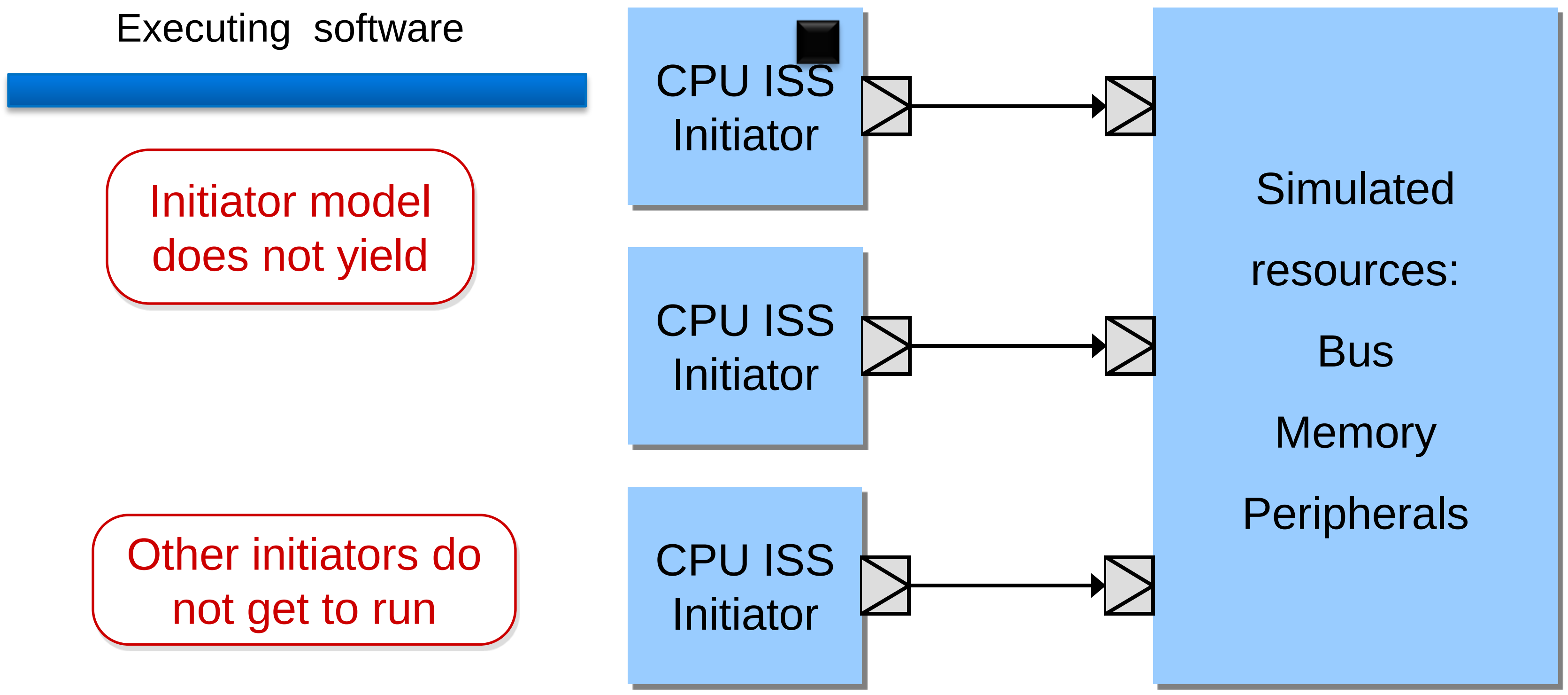
TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

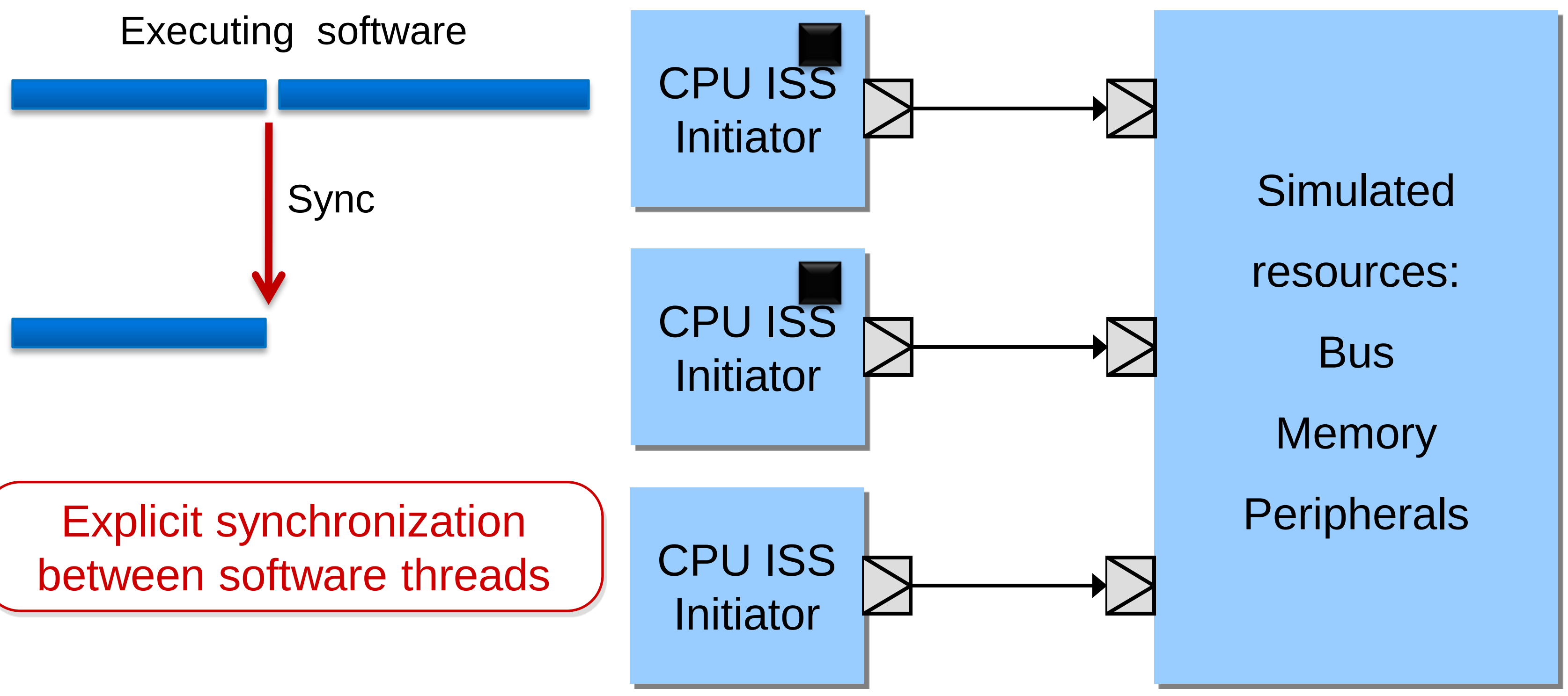
Software Execution and Simulation



Software execution without Sync



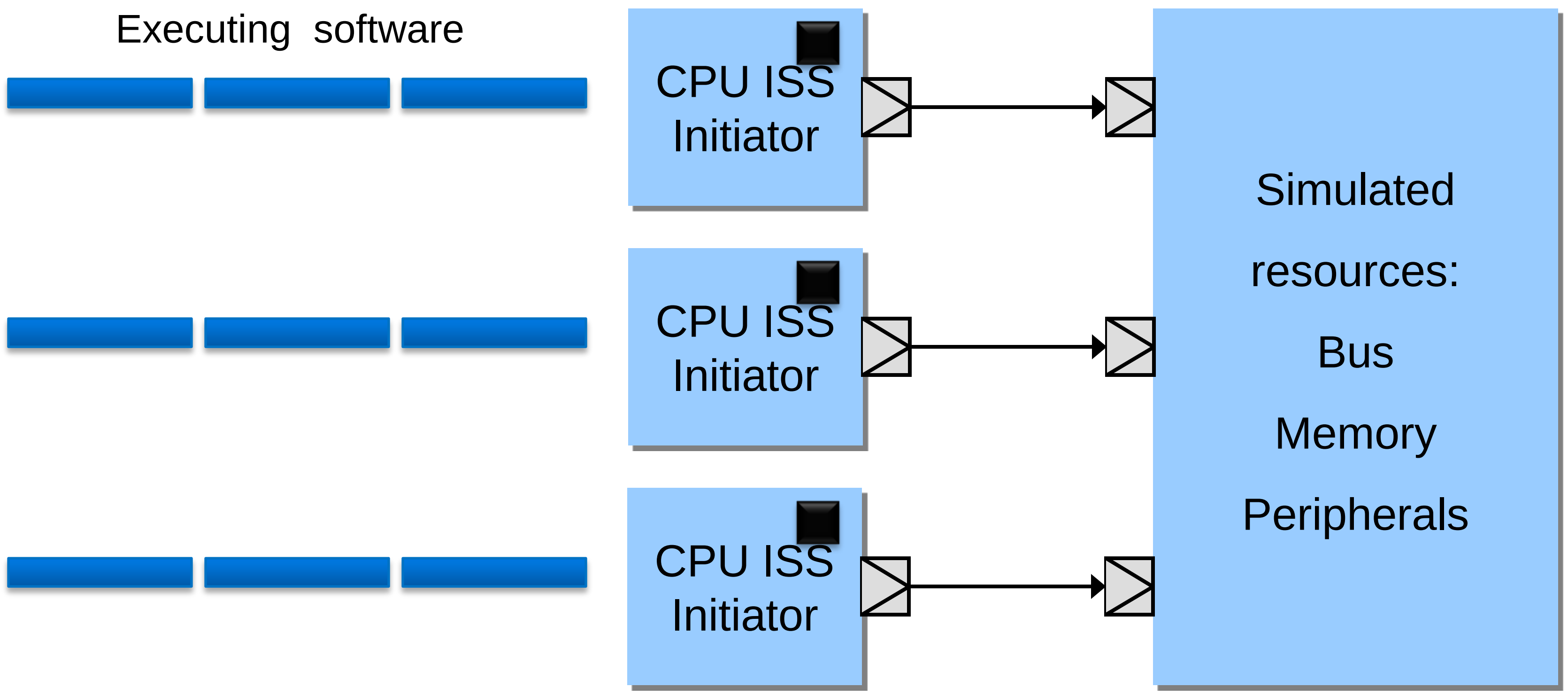
Software execution with Sync



Explicit synchronization between software threads

Synchronization is independent of platform timing

Software execution with a Quantum



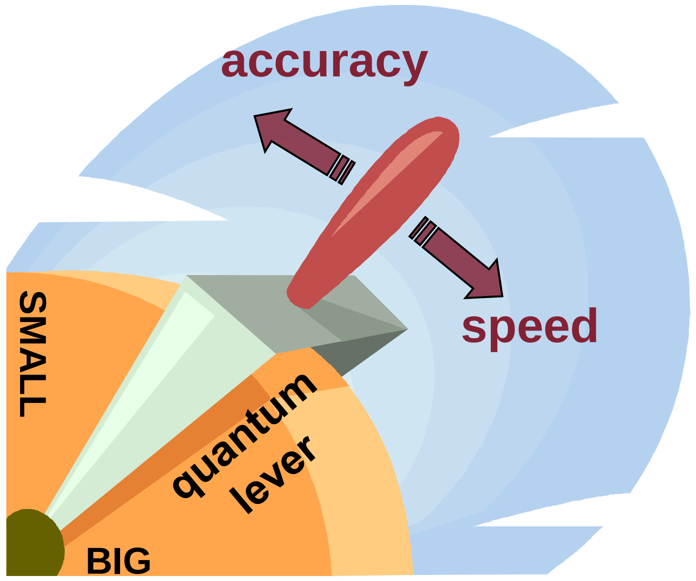
Initiators use a time quantum

Each initiator gets a time slice

Resources consume simulation time

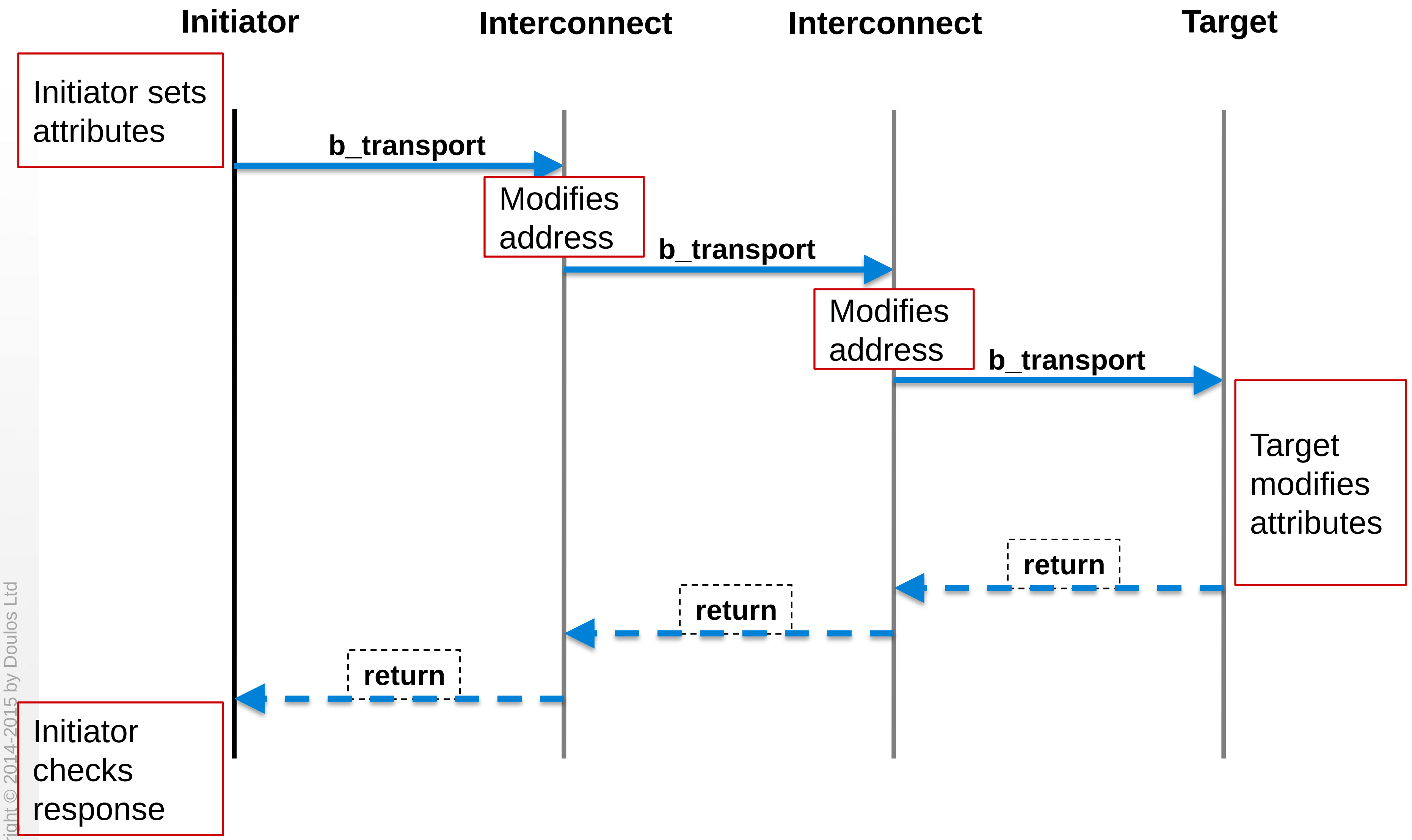
The Quantum

- Quantum can be set by user



- Typically, all initiators use the same global quantum
- Individual initiators can be permitted to sync more frequently
- Not obliged to use the quantum at all if there are explicit synchronization points
- Quantum could be changed dynamically to “zoom in” (but no explicit support)

Causality with b_transport



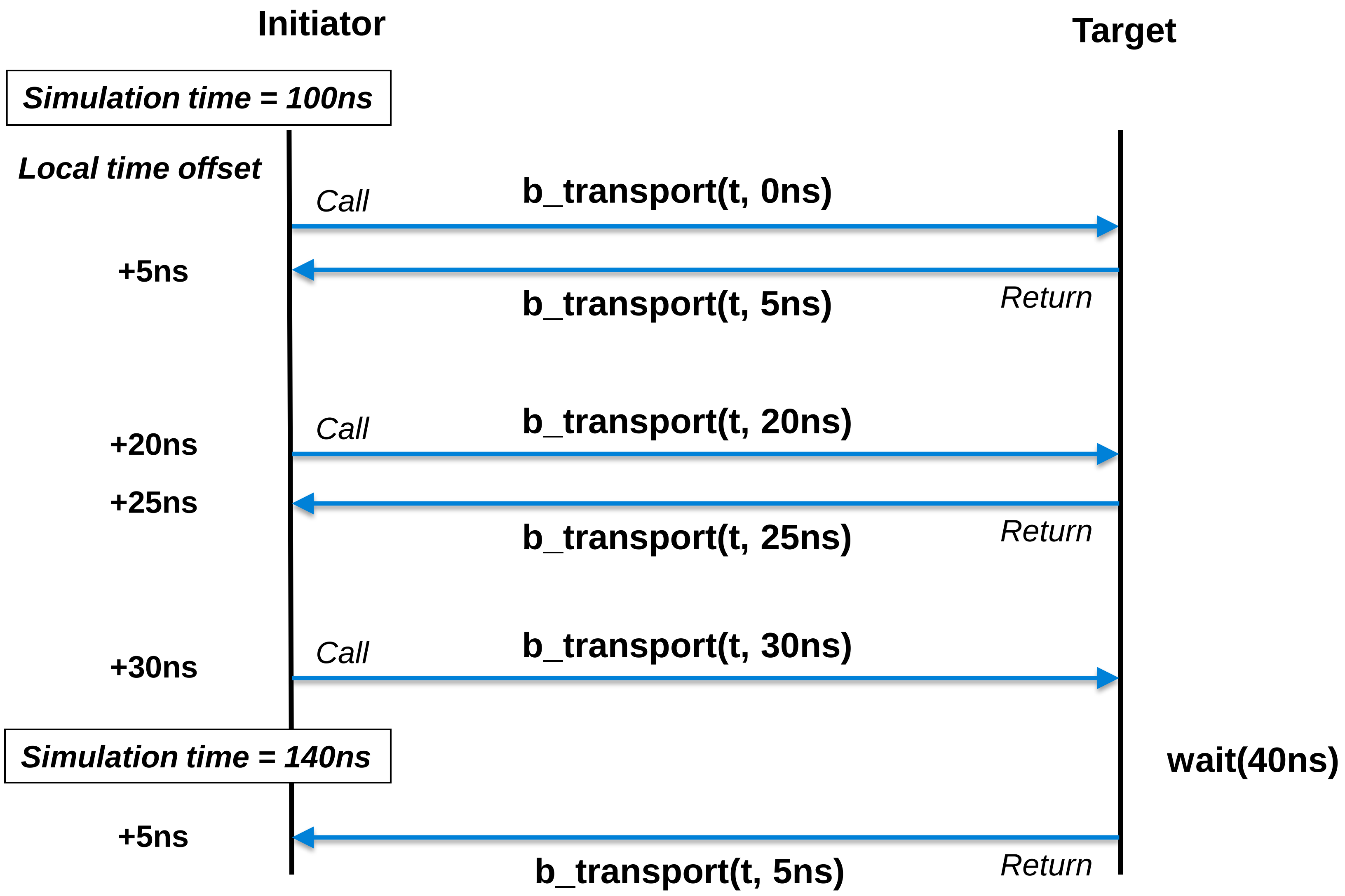
Timing Annotation and b_transport

```
virtual  void  b_transport ( TRANS& trans , sc_core::sc_time& delay )
{
  Behave as if method were called at sc_time_stamp() + delay
  ...
  delay = delay + latency;
}
```

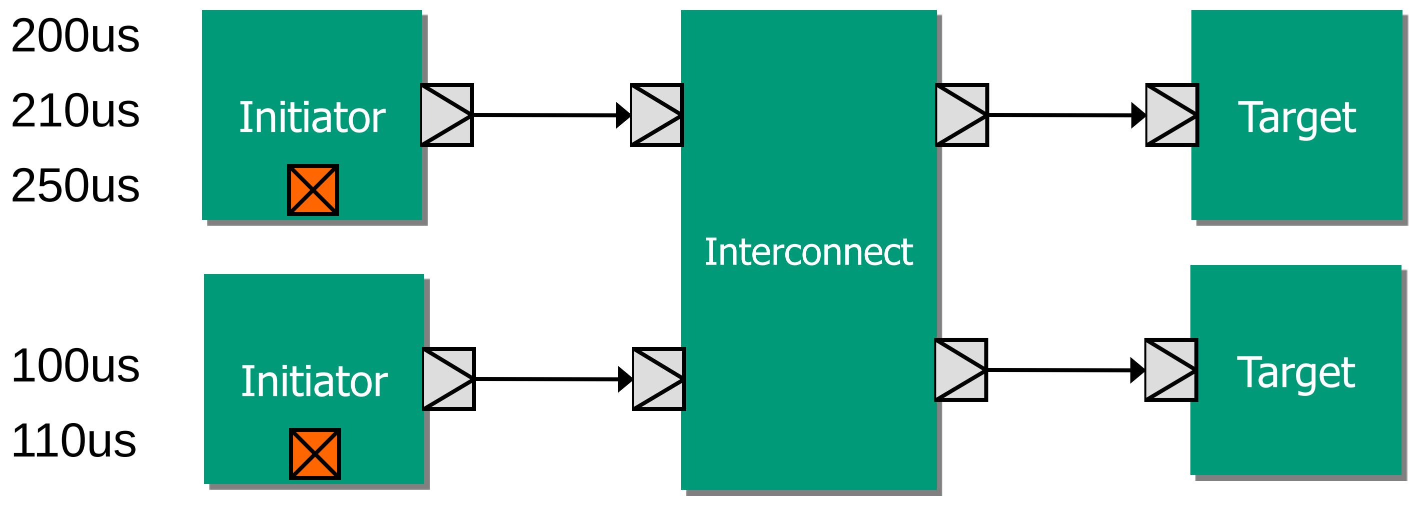
```
socket->b_transport( transaction, delay );

Behave as if method returned at sc_time_stamp() + delay
```

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

Temporal Decoupling



Base Protocol Rules



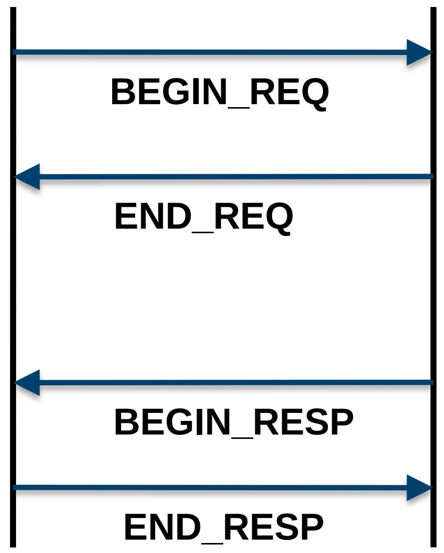
- Each initiator should generate transactions in non-decreasing time order
- Targets usually return immediately (don't want b_transport to block)
- b_transport is re-entrant anyway
- Incoming transactions through different sockets may be out-of-order
- Out-or-order transactions can be executed in any order
- Arbitration is typically inappropriate (and too slow)

Custom protocols make their own rules

AT and CA

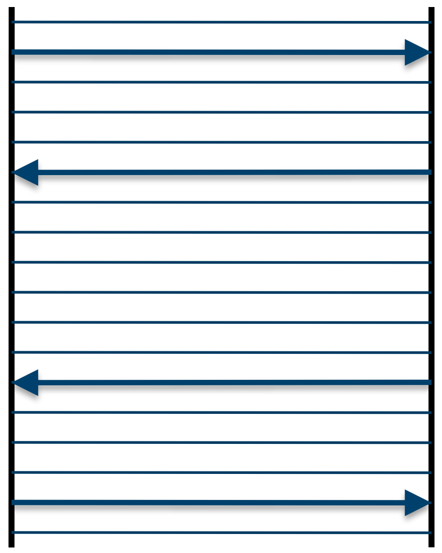
- No running ahead of simulation time; everything stays in sync

AT / nb_transport



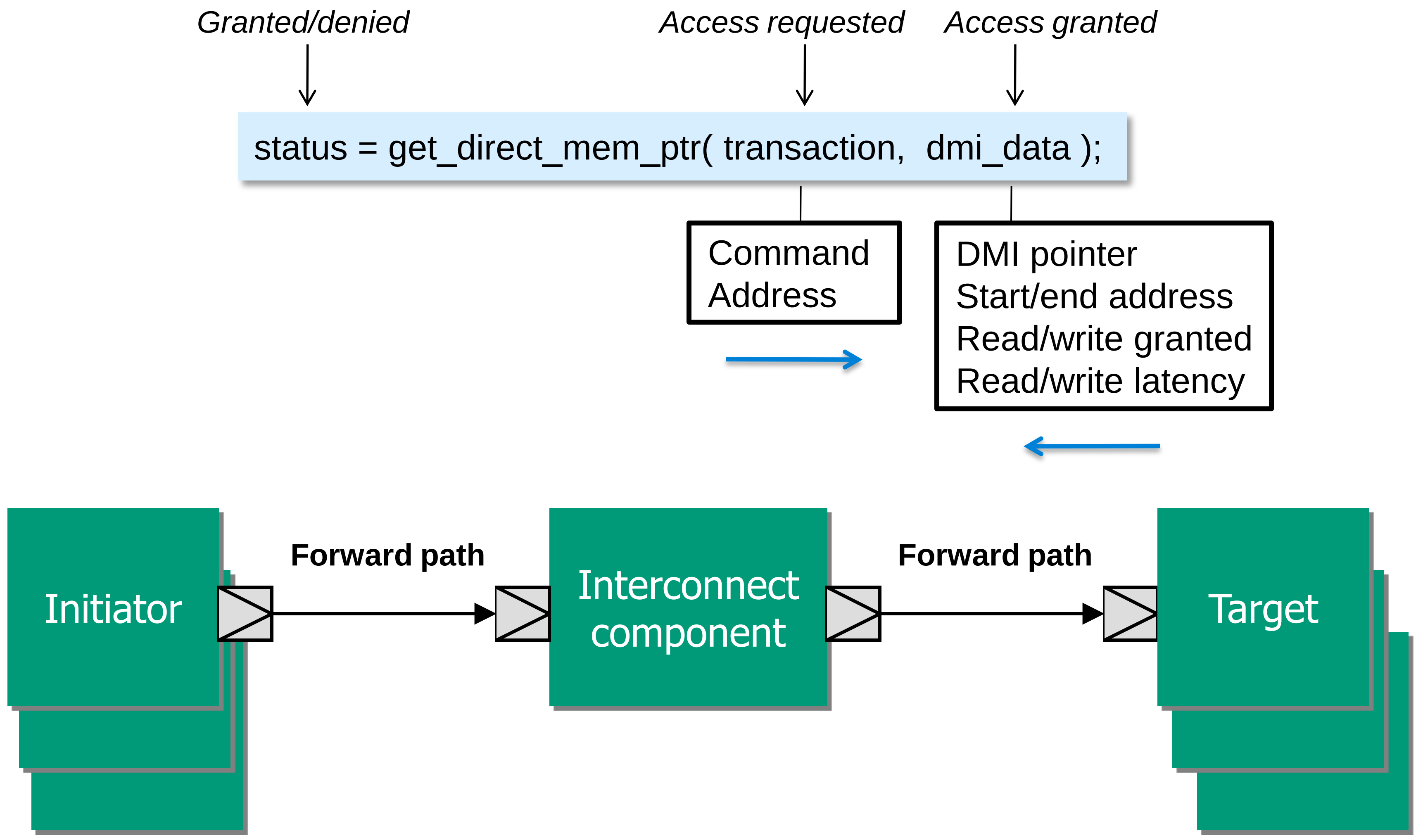
Wake up at significant timing points

CA



Wake up every cycle

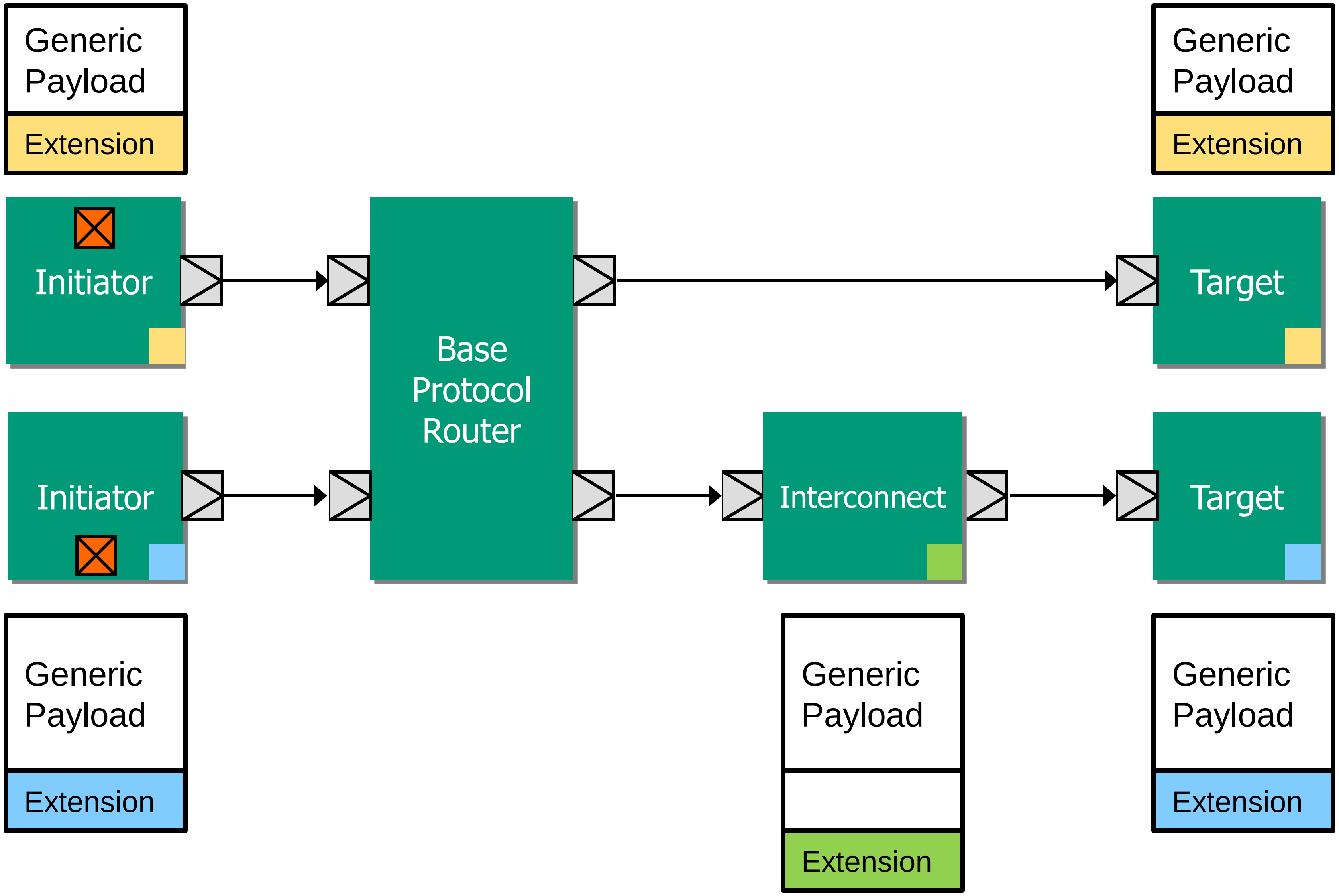
Direct Memory Interface



DMI pointer bypasses interconnect

TLM-2.0 and IEEE 1666-2011

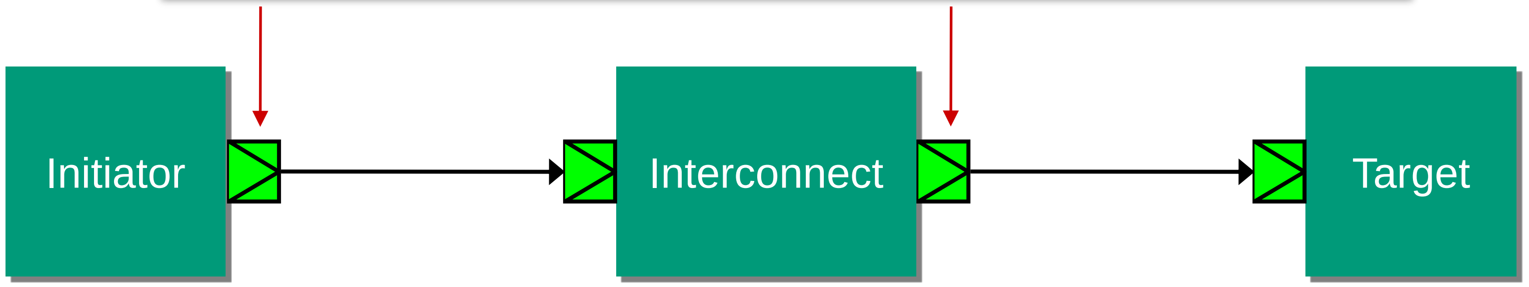
Extensions


Generic Payload
Extension
Generic Payload
Extension
Initiator
Base Protocol Router
Target
Initiator
Interconnect
Target
Generic Payload
Extension
Generic Payload
Extension
Generic Payload
Extension

First Kind of Interoperability

- Use the full interoperability layer
- Use the generic payload + ignorable extensions as an abstract bus model
- Obey all the rules of the base protocol. The LRM is your rule book

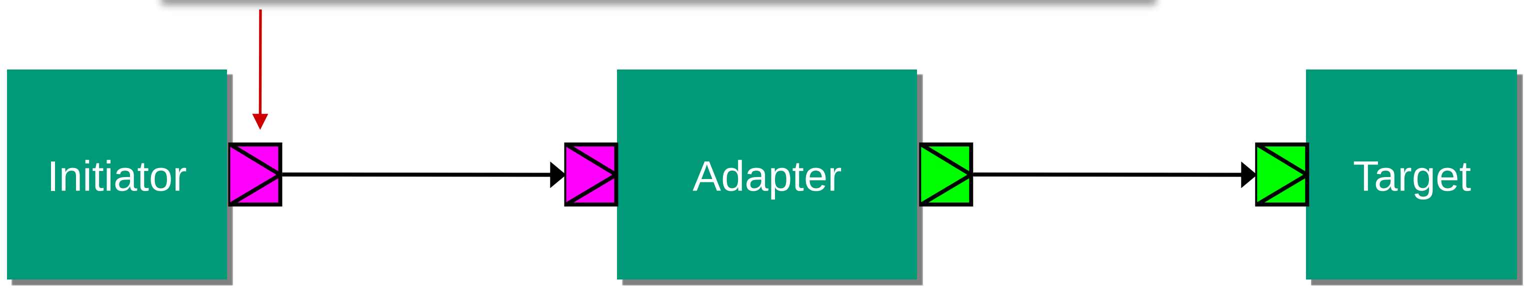
```
tlm_initiator_socket<32, tlm_base_protocol_types>  my_socket;
```



Second Kind of Interoperability

- Create a new protocol traits class
- Create user-defined generic payload extensions and phases as needed
- Make your own rules!

```
tlm_initiator_socket<32, my_protocol> my_socket;
```



- One rule enforced: cannot bind sockets of differing protocol types
- Recommendation: keep close to the base protocol. The LRM is your guidebook
- The clever stuff in TLM-2.0 makes the adapter fast

TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

Process Control

- suspend
- resume
- disable
- enable
- sync_reset_on
- sync_reset_off
- reset
- kill
- throw_it

- reset_event
- sc_unwind_exception
- sc_is_unwinding

- reset_signal_is
- async_reset_signal_is

Framework for Examples

```
struct M: sc_module
{
  M(sc_module_name n)
  {
    SC_THREAD(calling);
    SC_THREAD(target);
  }

  void calling()
  {
    ...
  }

  void target()
  {
    ...
  }

  SC_HAS_PROCESS(M);
};
```

```
int sc_main(int argc, char* argv[])
{
  M m("m");
  sc_start(500, SC_NS);
  return 0;
}
```

Events

```
M(sc_module_name n)
{
   SC_THREAD(calling);
   SC_THREAD(target);
}
```

```
sc_event ev;
```

```
void calling()
{
   ev.notify(5, SC_NS);
}
```

```
void target()
{
   while (1)
   {
      wait(ev);
      cout << sc_time_stamp();
   }
}
```

5

Process Handles

```
M(sc_module_name n)
{
  SC_THREAD(calling);
  SC_THREAD(target);
    t = sc_get_current_process_handle();
}
```

```
sc_process_handle t;
```

```
void calling()
{
  assert( t.valid() );
  cout << t.name();
  cout << t.proc_kind();
}
```

```
m.target  2
```

```
void target()
{
  while (1)
  {
    wait(100, SC_NS);
    cout << sc_time_stamp();
  }
}
```

```
100 200 300 400
```

suspend & resume

```
void calling()
{
  wait(20, SC_NS);
  t.suspend();        at 20
  wait(20, SC_NS);
  t.resume();         at 40

  wait(110, SC_NS);
  t.suspend();        at 150
  wait(200, SC_NS);
  t.resume();         at 350
}
```

```
void target()
{
  while (1)
  {
    wait(100, SC_NS);
    cout << sc_time_stamp();
  }
}
```

```
100 350 450
```

suspend & resume

```
void calling()
{
  wait(20, SC_NS);
  t.suspend();            at 20
  wait(20, SC_NS);
  t.resume();             at 40

  wait(110, SC_NS);
  t.suspend();            at 150
  wait(200, SC_NS);
  t.resume();             at 350
}
```

```
void tick() {
  while (1) {
    wait(100, SC_NS);
    ev.notify();
  }
}
```

```
void target()
{
  while (1)
  {
    wait(ev);
    cout << sc_time_stamp();
  }
}
```

```
100 350 450
```

disable & enable

```
void calling()
{
  wait(20, SC_NS);
  t.disable();          at 20
  wait(20, SC_NS);
  t.enable();           at 40

  wait(110, SC_NS);
  t.disable();          at 150
  wait(200, SC_NS);
  t.enable();           at 350
}
```

```
SC_THREAD(target);
  sensitive << clock.pos();
```

```
void target()
{
  while (1)
  {
    wait();
    cout << sc_time_stamp();
  }
}
```

100 400

suspend versus disable

```
void calling()
{
   ...
   t.suspend();
   ...
   t.resume();
   ...
}
```

- Clamps down process until resumed
- Still sees incoming events & time-outs
- Unsuitable for clocked target processes
- Building abstract schedulers

```
void calling()
{
   ...
   t.disable();
   ...
   t.enable();
   ...
}
```

- Disconnects sensitivity
- Runnable process remains runnable
- Suitable for clocked targets
- Abstract clock gating

Scheduling

```
void calling1()
{
   t.suspend();
}
```

Target suspended immediately

```
void calling2()
{
   t.resume();
}
```

Target runnable immediately,
may run in current eval phase

```
void target()
{
   while (1)
   {
      wait(ev);
      ...
   }
}
```

```
void calling3()
{
   t.disable();
}
```

Sensitivity disconnected immediately,
target may run in current eval phase

```
void calling4()
{
   t.enable();
}
```

Sensitivity reconnected immediately,
never itself causes target to run

Self-control

THURSDAY IS TRAINING DAY

```
M(sc_module_name n)
{
  SC_THREAD(thread_proc);
    t = sc_get_current_process_handle();
  SC_METHOD(method_proc);
    m = sc_get_current_process_handle();
}
```

```
void thread_proc()
{
  ...
  t.suspend();      Blocking
  ...
  t.disable();      Non-blocking
  wait(...);
  ...
}
```

```
void method_proc()
{
  ...
  m.suspend();      Non-blocking
  ...
  m.disable();      Non-blocking
  ...
}
```

sync_reset_on/off

```
SC_THREAD(calling);
SC_THREAD(target);
  t = sc_get_current_process_handle();
```

```
void calling() {
  wait(10, SC_NS);
  ev.notify();                ++q

  wait(10, SC_NS);
  t.sync_reset_on();

  wait(10, SC_NS);
  ev.notify();                q = 0

  wait(10, SC_NS);
  t.sync_reset_off();

  wait(10, SC_NS);
  ev.notify();                ++q
}
```

```
void target()
{
  q = 0;
  while (1)
  {
    wait(ev);
    ++q;
  }
}
```

Wakes at 10 30 50

Interactions

3 independent flags

- ○ disable / enable (highest priority)
- ● suspend / resume
- ● sync_reset_on / off (lowest priority)

```
void calling()
{
  t.suspend();
  ...
  t.sync_reset_on();
  ...
  t.suspend();
  ...
  t.disable();
  ...
  t.sync_reset_off();
  ...
  t.resume();
  ...
  t.enable();
  ...
  t.resume();
}
```

```
void target()
{
  q = 0;
  while (1)
  {
    wait(ev);
    ++q;
  }
}
```

Process Control

- suspend
- resume
- disable
- enable
- sync_reset_on
- sync_reset_off
- reset
- kill
- throw_it

- reset_event
- sc_unwind_exception
- sc_is_unwinding

- reset_signal_is
- async_reset_signal_is

reset and kill

THURSDAY IS TRAINING DAY

```
SC_THREAD(calling);
SC_THREAD(target);
  t = sc_get_current_process_handle();
```

```
void calling()
{
  wait(10, SC_NS);
  ev.notify();

  wait(10, SC_NS);
  t.reset();

  wait(10, SC_NS);
  ev.notify();

  wait(10, SC_NS);
  t.kill();
}
```

++q

q = 0

++q

```
void target()
{
  q = 0;
  while (1)
  {
    wait(ev);
    ++q;
  }
}
```

Wakes at 10 20 30

Terminated at 40

reset and kill are Immediate

```
void calling()
{
  wait(10, SC_NS);
  ev.notify();
  assert( q == 0 );

  wait(10, SC_NS);
  assert( q == 1 );

  t.reset();
  assert( q == 0 );

  wait(10, SC_NS);
  t.kill();
  assert( t.terminated() );
}
```

++q

q = 0

Forever

```
int q;
```

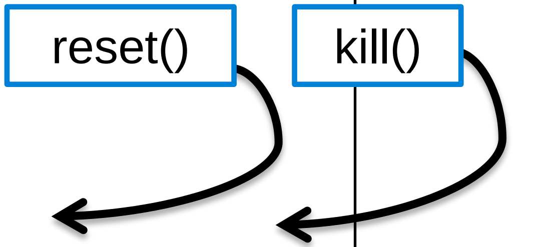
```
void target()
{
  q = 0;
  while (1)
  {
    wait(ev);
    ++q;
  }
}
```

Cut through suspend, disable

Disallowed during elaboration

Unwinding the Call Stack

```
void target()
{
  q = 0;
  while (1)
  {
    try {
      wait(ev);
      ++q;
    }
    catch (const sc_unwind_exception& e)
    {




    }
    ...
```



Unwinding the Call Stack

```
void target()
{
  q = 0;
  while (1)
  {
    try {
      wait(ev);
      ++q;
    }
    catch (const sc_unwind_exception& e)
    {
      sc_assert( sc_is_unwinding() );
      if (e.is_reset()) cout << "target was reset";
      else              cout << "target was killed";



    }
    ...
```

reset()

kill()

Unwinding the Call Stack

```
void target()
{
  q = 0;
  while (1)
  {
    try {
      wait(ev);
      ++q;
    }
    catch (const sc_unwind_exception& e)
    {
      sc_assert( sc_is_unwinding() );
      if (e.is_reset()) cout << "target was reset";
      else              cout << "target was killed";
      proc_handle.reset();

      throw e;
    }
    ...
```

reset()

kill()

Resets some other process

Must be re-thrown

reset_event

```
SC_THREAD(calling);
SC_THREAD(target);
  t = sc_get_current_process_handle();

SC_METHOD(reset_handler);
  dont_initialize();
  sensitive << t.reset_event();

SC_METHOD(kill_handler);
  dont_initialize();
  sensitive << t.terminated_event();
```

```
void calling()
{
  wait(10, SC_NS);
  t.reset();
  wait(10, SC_NS);
  t.kill();
  ...
```

```
void target()
{
  ...
  while (1)
  {
    wait(ev);
    ...
  }
}
```

throw_it

std::exception recommended

```
std::exception ex;
```

```
void calling()
{
   ...
   t.throw_it(ex);
   ...
}
```

Immediate - 2 context switches

```
void target()
{
   q = 0;
   while (1) {
      try {
         wait(ev);
         ++q;
      }
      catch (const std::exception& e)
      {
         if (...)
            ; // wait(ev);
         else
            return;
      }
   ...
```

Must catch exception

May continue or terminate

Process Control

- suspend
- resume
- disable
- enable
- sync_reset_on
- sync_reset_off
- reset
- kill
- throw_it

- reset_event
- sc_unwind_exception
- sc_is_unwinding

- reset_signal_is
- async_reset_signal_is

Styles of Reset

	asynchronous	sync reset state
`handle.reset();`	●	

`handle.sync_reset_on();`
`...`
`handle.sync_reset_off();`
		●

`SC_THREAD(target);`		
`reset_signal_is(reset, active_level);`		●
`async_reset_signal_is(reset, active_level);`	●	●
`sc_spawn_options opt;`		
`opt.reset_signal_is(reset, active_level);`		●
`opt.async_reset_signal_is(reset, true);`	●	●

Styles of Reset

```
SC_THREAD(target);
  sensitive << ev;
  reset_signal_is(sync_reset, true);
  async_reset_signal_is(async_reset, true);
```

Effectively

```
t.reset();                          t.reset();
t.sync_reset_on();
...
ev.notify();                        t.reset();
...
t.sync_reset_off();
sync_reset = true;
...
ev.notify();                        t.reset();
sync_reset = false;
...
async_reset = true;                 t.reset();
..
ev.notify();                        t.reset();
```

Processes Unified!

```
SC_METHOD(M);
  sensitive << clk.pos();
  reset_signal_is(r, true);
  async_reset_signal_is(ar, true);
```

```
SC_THREAD(T);
  sensitive << clk.pos();
  reset_signal_is(r, true);
  async_reset_signal_is(ar, true);
```

```
SC_CTHREAD(T, clk.pos());
  reset_signal_is(r, true);
  async_reset_signal_is(ar, true);
```

```
void M() {
  if (r|ar)
    q = 0;
  else
    ++q
}
```

```
void T() {
  if (r|ar)
    q = 0;
  while (1)
  {
    wait();
    ++q;
  }
}
```

TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

Array of Ports or Signals

```
struct Child: sc_module
{
  sc_in<int> p[4];
  ...
```

Ports cannot be named

```
struct Top: sc_module
{
  sc_signal<int> sig[4];
  Child* c;

  Top(sc_module_name n)
  {
    c = new Child("c");
    c->p[0].bind(sig[0]);
    c->p[1].bind(sig[1]);
    c->p[2].bind(sig[2]);
    c->p[3].bind(sig[3]);
  }
  ...
```

Signals cannot be named

Array or Vector of Modules

THURSDAY IS TRAINING DAY

```
struct Child: sc_module
{
  sc_in<int> p;
  ...
```

```
struct Top: sc_module
{
  sc_signal<int> sig[4];
  std::vector<Child*> vec;

  Top(sc_module_name n) {
    vec.resize(4);
    for (int i = 0; i < 4; i++)
    {
      std::stringstream n;
      n << "vec_" << i;
      vec[i] = new Child(n.str().c_str(), i);
      vec[i]->p.bind(sig[i]);
    }
  }
  ...
```

Modules not default constructible

sc_vector of Ports or Signals

```
struct Child: sc_module
{
  sc_vector< sc_in<int> > port_vec;

  Child(sc_module_name n)
  : port_vec("port_vec", 4)
  { ...
```

Elements are named

```
struct Top: sc_module
{
  sc_vector< sc_signal<int> > sig_vec;
  Child* c;

  Top(sc_module_name n)
  : sig_vec("sig_vec", 4)
  {
    c = new Child("c");
    c->port_vec.bind(sig_vec);
  }
  ...
```

Size passed to ctor

Vector-to-vector bind

sc_vector of Modules

```
struct Child: sc_module
{
  sc_in<int> p;
  ...
```

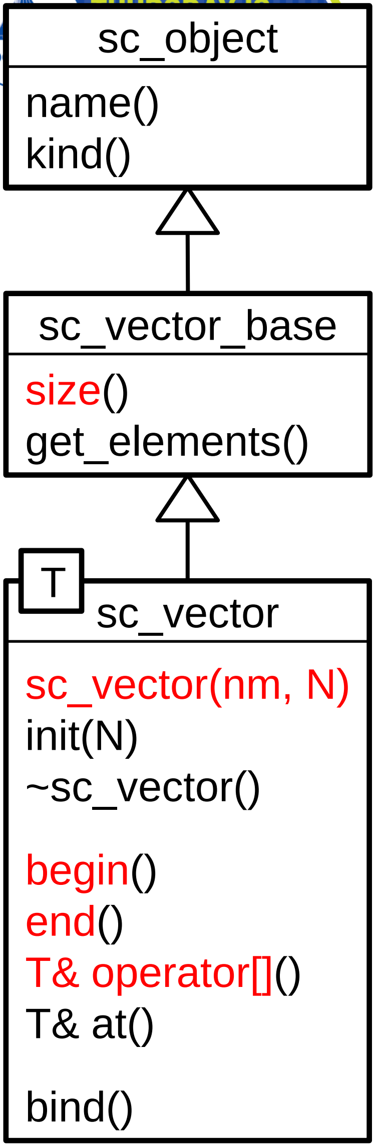
```
struct Top: sc_module
{
  sc_vector< sc_signal<int> > sig_vec;
  sc_vector< Child > mod_vec;

  Top(sc_module_name n)
  : sig_vec("sig_vec")
  , mod_vec("mod_vec")
  {
    sig_vec.init(4);
    mod_vec.init(4);
    for (int i = 0; i < 4; i++)
      mod_vec[i]->p.bind(sig_vec[i]);
  }
  ...
```

Elements are named

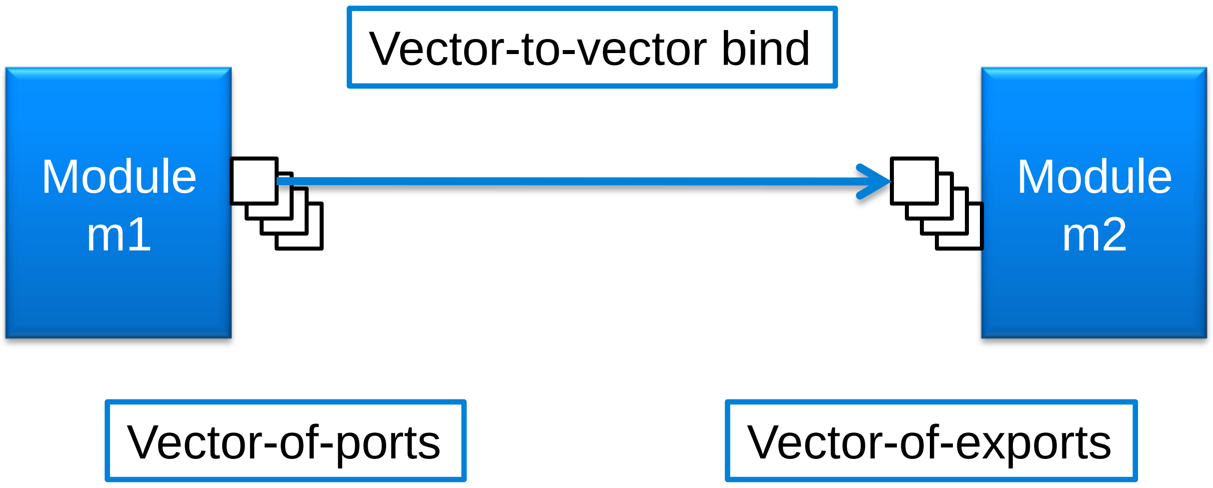
Size deferred

sc_vector methods

```
struct M: sc_module
{
  sc_vector< sc_signal<int> > vec;

  M(sc_module_name n)
  : vec("vec", 4) {
    SC_THREAD(proc)
  }
  void proc() {
    for (unsigned int i = 0; i < vec.size(); i++)
      vec[i].write(i);

    wait(SC_ZERO_TIME);

    sc_vector< sc_signal<int> >::iterator it;
    for (it = vec.begin(); it != vec.end(); it++)
      cout << it->read() << endl;
    ...
```

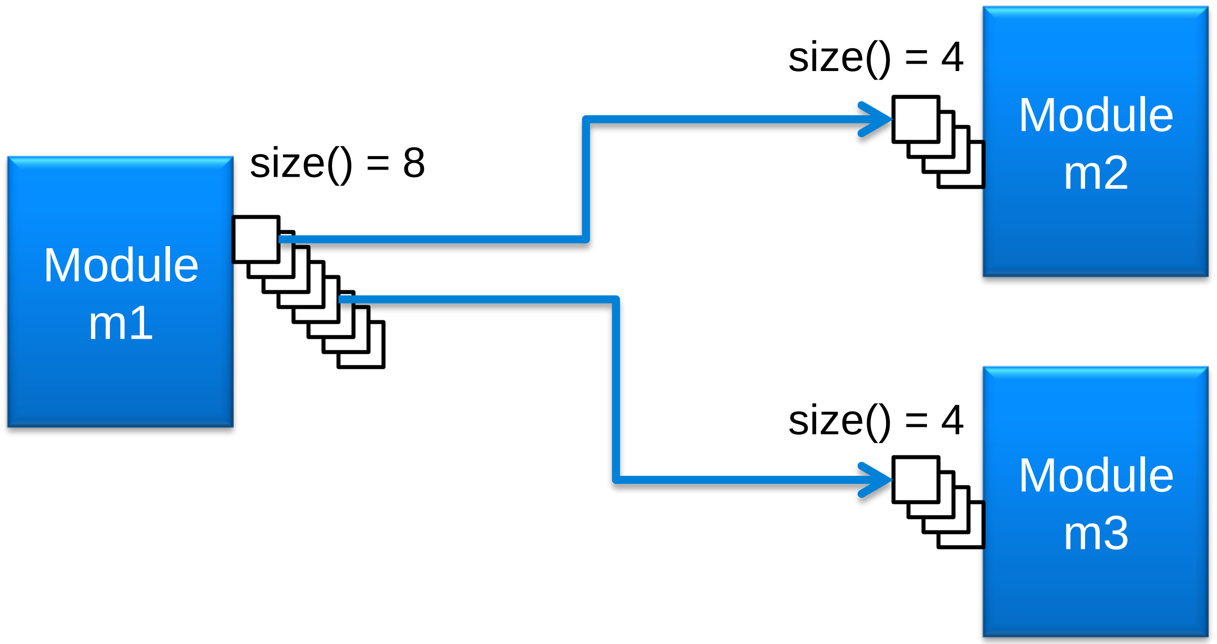


Binding Vectors



```
m1->port_vec.bind( m2->export_vec );
```

Partial Binding

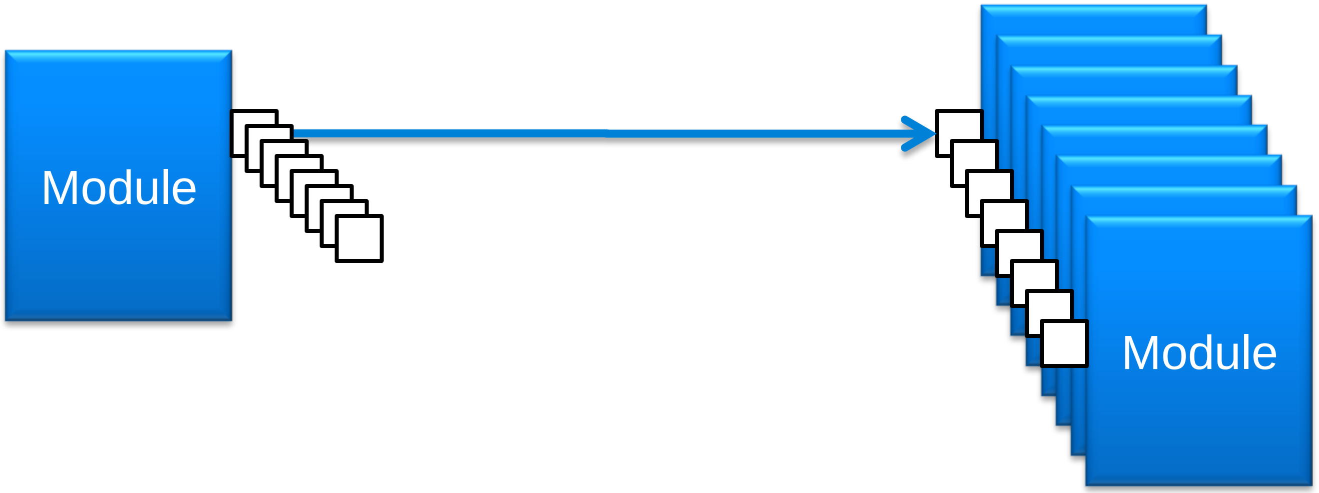


```
sc_vector<sc_port<i_f> >::iterator it;
it = m1->port_vec.bind( m2->export_vec );

it = m1->port_vec.bind( m3->export_vec.begin(),
                        m3->export_vec.end(),
                        it );
```

1st unbound element

Start binding here

sc_assemble_vector



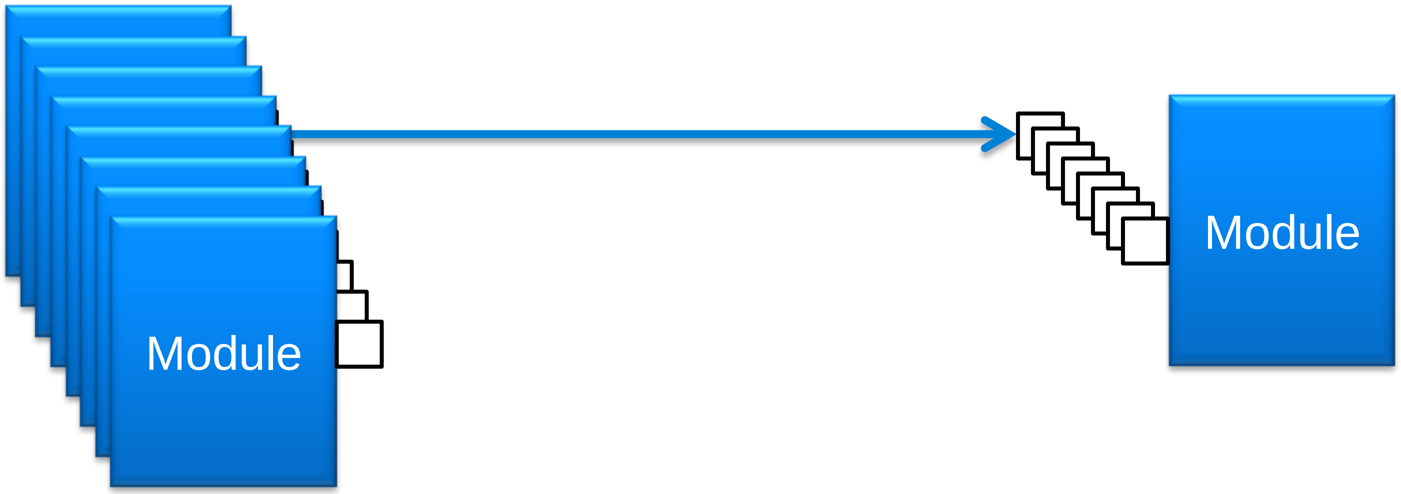
Vector-of-ports

Vector-of-modules, each with one export

```
init->port_vec.bind(
    sc_assemble_vector(targ_vec, &Target::export) );
```

Substitute for a regular vector

sc_assemble_vector



Vector-of-modules,
each with one port

Vector-of-exports

```
sc_assemble_vector(init_vec, &Init::port).bind(
                                         targ->export_vec);
```

For More FREE Information

- IEEE 1666

standards.ieee.org/getieee/1666/download/1666-2011.pdf

- Accellera Systems Initiative Proof-of-Concept Simulator: SystemC 2.3.1

www.accellera.org

- On-line tutorials

www.doulos.com/knowhow/systemc

DOULOS

Delivering Know-How www.doulos.com

Hardware Design

» VHDL » Verilog » SystemVerilog

Embedded Systems and ARM

» C » C++ » UML » RTOS » Linux » Yocto
» ARM Cortex » Python » Android » OpenCL » OpenGL

ESL & Verification

» SystemC » TLM-2.0 » SystemVerilog
» OVM/VMM/UVM » Perl » Tcl/Tk