The TLM-2.0 Standard

John Aynsley, Doulos

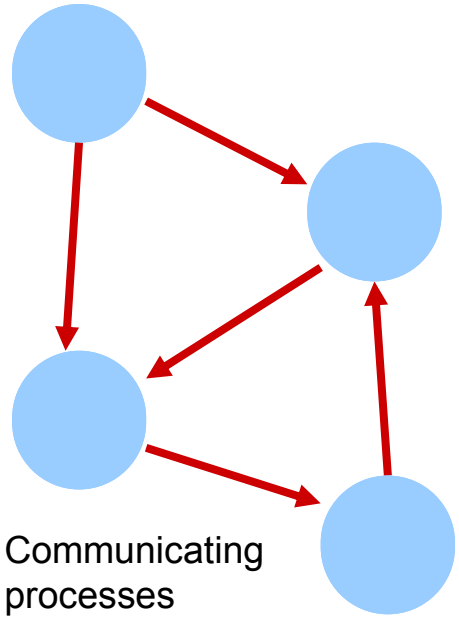
The TLM-2.0 Standard

CONTENTS

- *Review of SystemC and TLM*
- *Review of TLM-2.0*
- *Frequently Asked Questions*

What is SystemC?

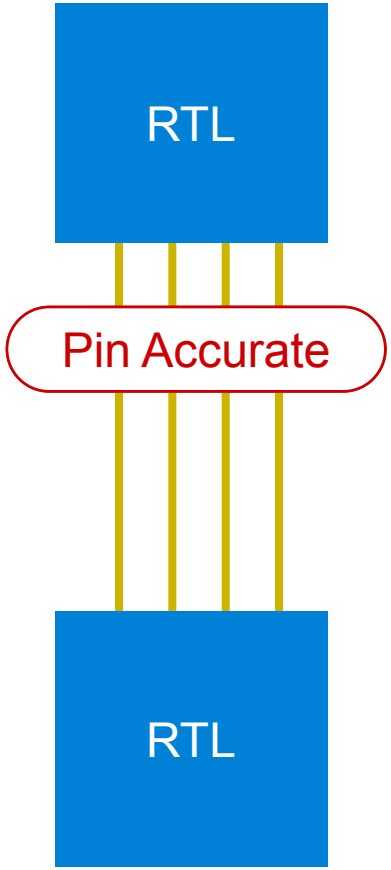
- System-level modeling language
  - Network of communicating processes (c.f. HDL)
  - Modeling hardware and software together
  - New: SystemC-AMS (mixed signal)

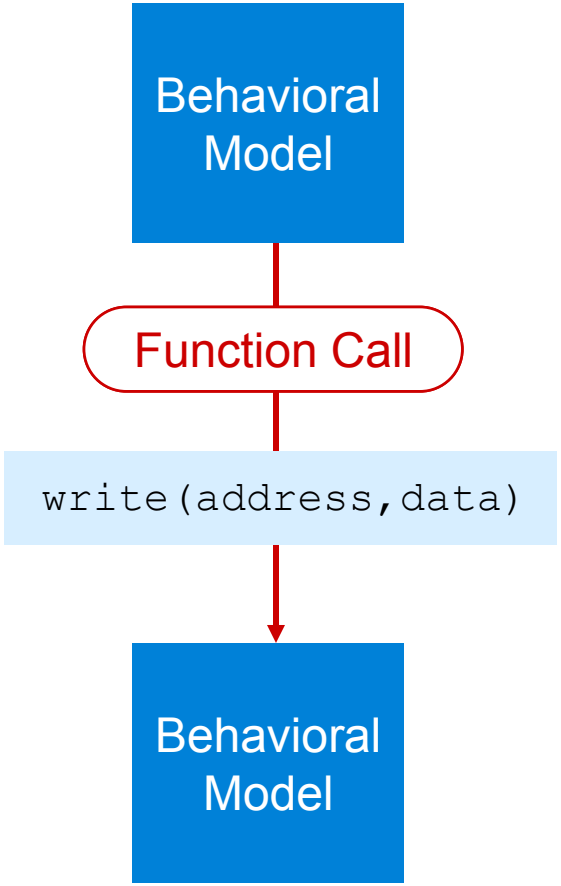


Communicating processes

- C++ class library
  - Industry standard IEEE 1666™
  - Owned and driven by OSCI (Open SystemC Initiative)
  - Open source implementation
  - Mature, robust, easy-to-integrate and “free”

Transaction Level Modeling



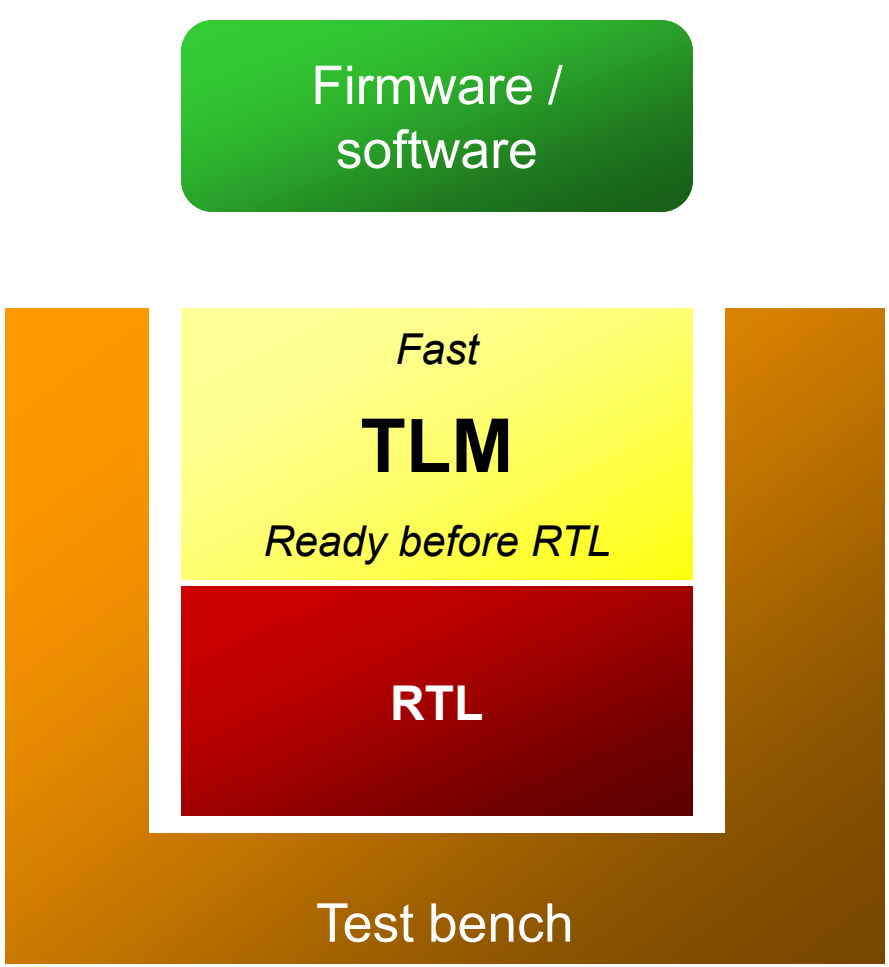


Simulate every event!

100-10,000 X faster simulation!

Reasons for using TLM

Accelerates product release schedule



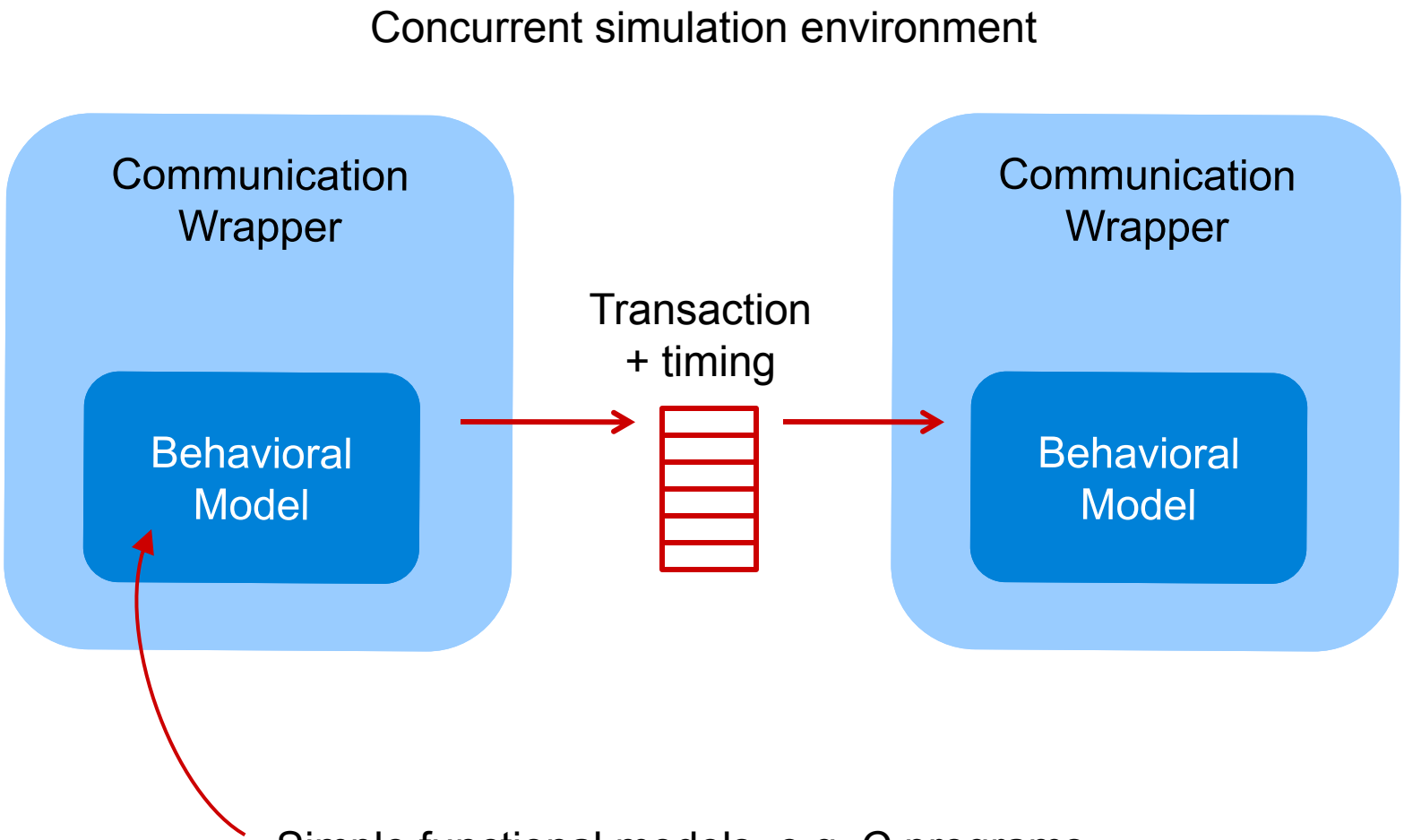
Software development

Architectural exploration

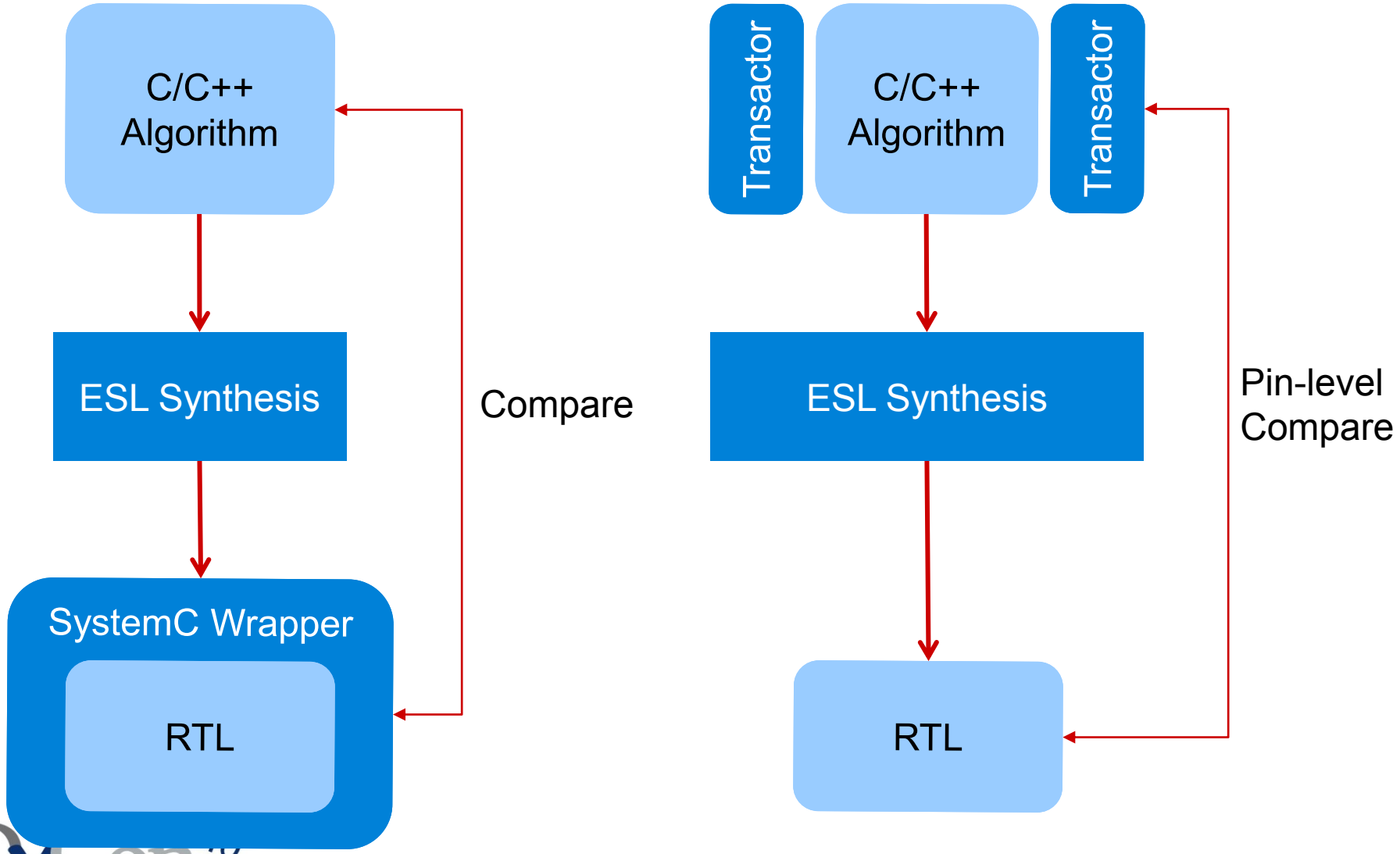
Hardware verification

TLM = golden model

TLM is Communication-Oriented



TLM and Synthesis



The TLM-2.0 Standard

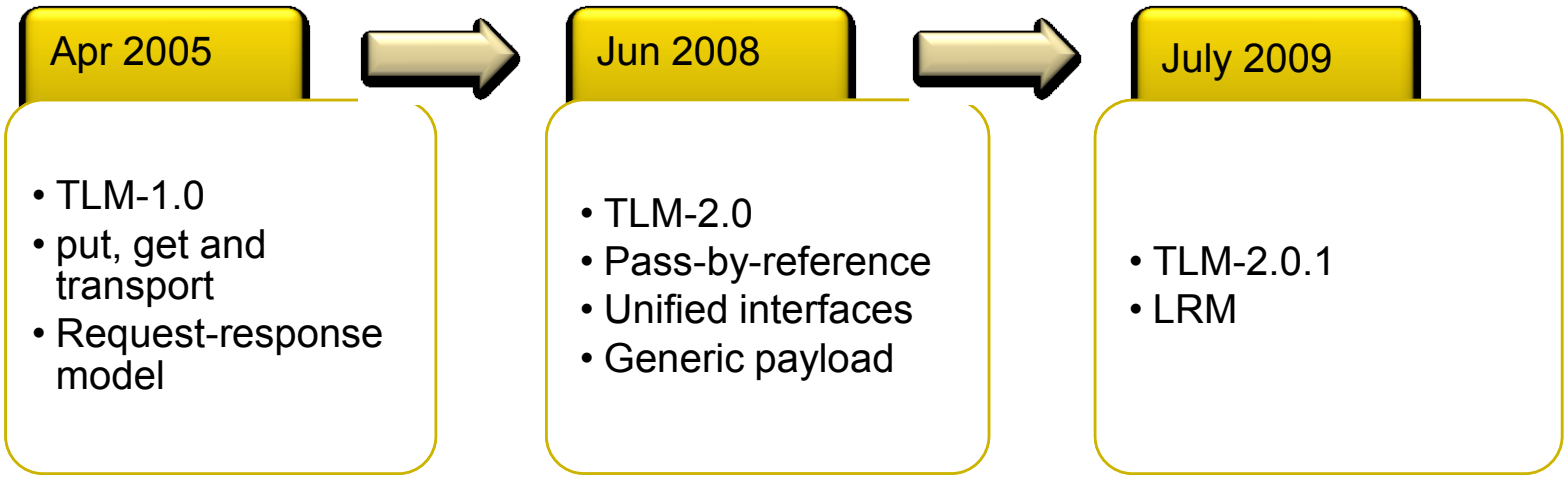
CONTENTS

- *Review of SystemC and TLM*
- ***Review of TLM-2.0***
- *Frequently Asked Questions*

DVCon 2010
Design Verification Conference and Exhibition

DOULOS

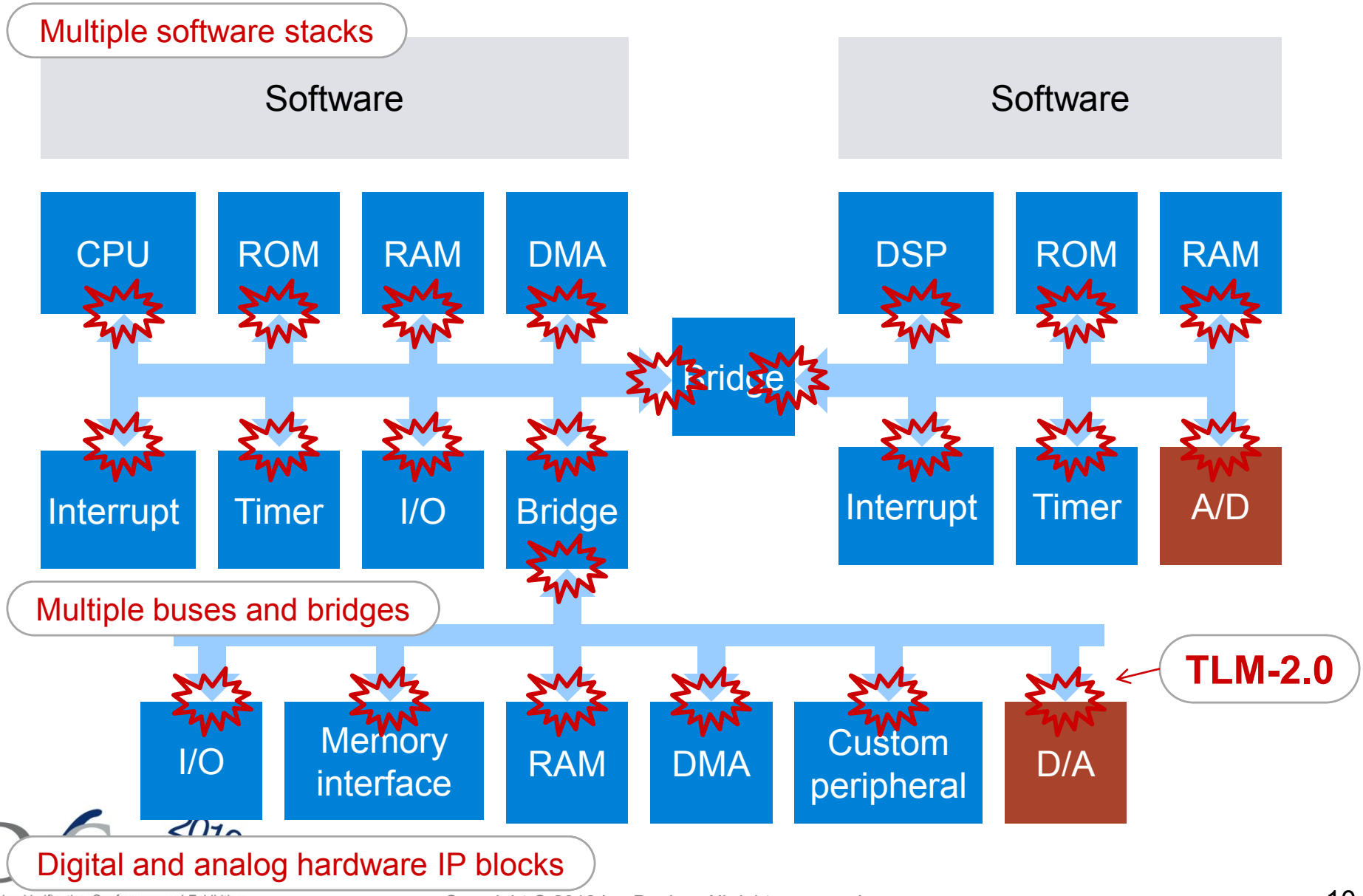
OSCI TLM Timeline



- TLM-2.0 focusses on memory-mapped bus modeling

Speed! **Interoperability!**

Typical Use Case: Virtual Platform

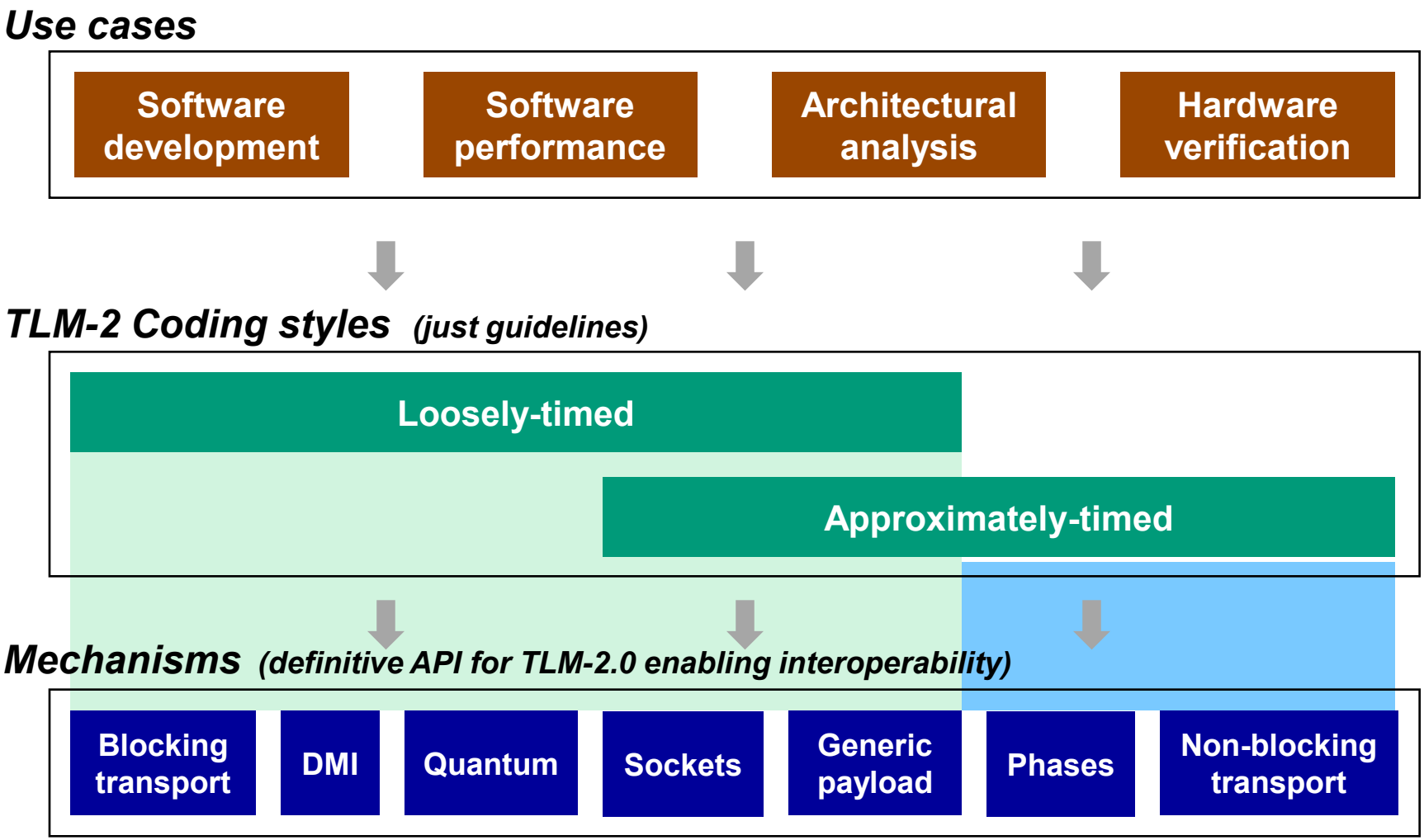


Virtual Platform Characteristics

Instruction Set Simulator or software stubs	Transaction-Level Model	RTL
Available early ✓	Available early ✓	Much later ✗
Fast enough to run applications ✓	Fast enough to run applications ✓	Too slow to run applications ✗
Little or no hardware detail ✗	Register-accurate ✓	Register-accurate and pin-accurate ✓
No timing information ✗	Some timing information ✓	Cycle-accurate timing ✓

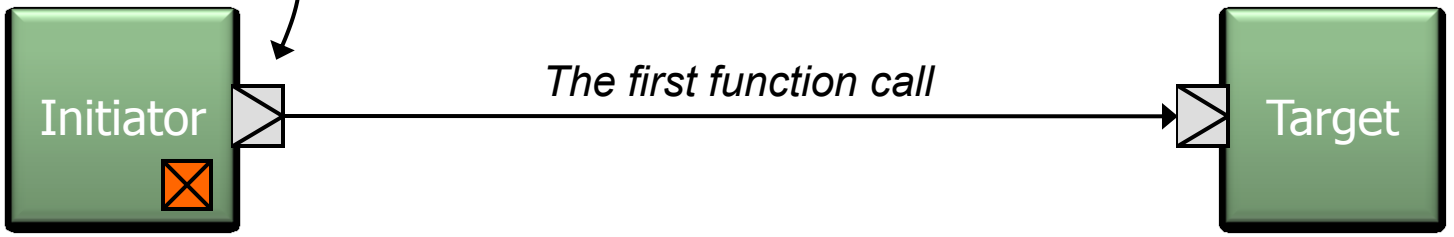
DVCon 2010
Design Verification Conference and Exhibition

Coding Styles and Mechanisms

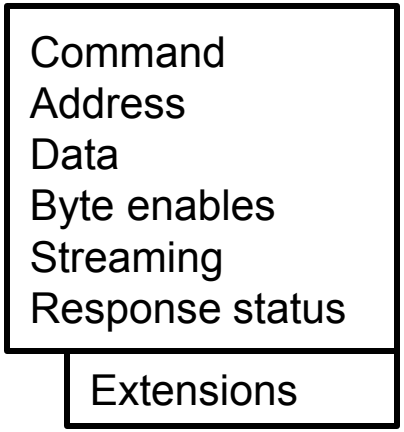


Interoperability Layer

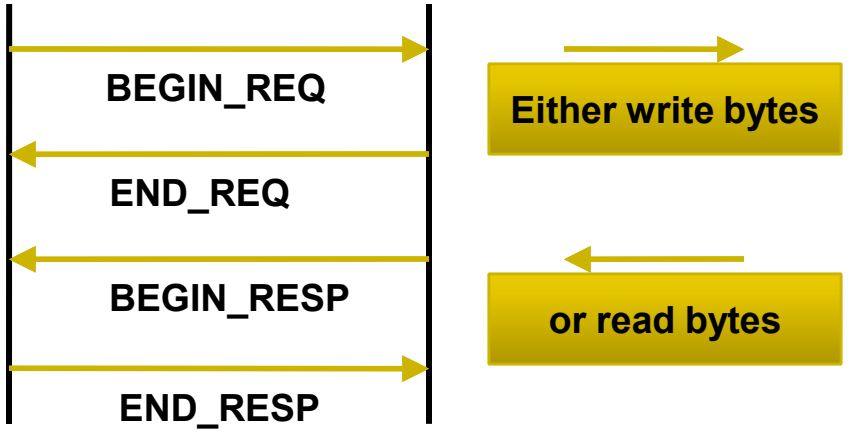
1. Core interfaces and sockets



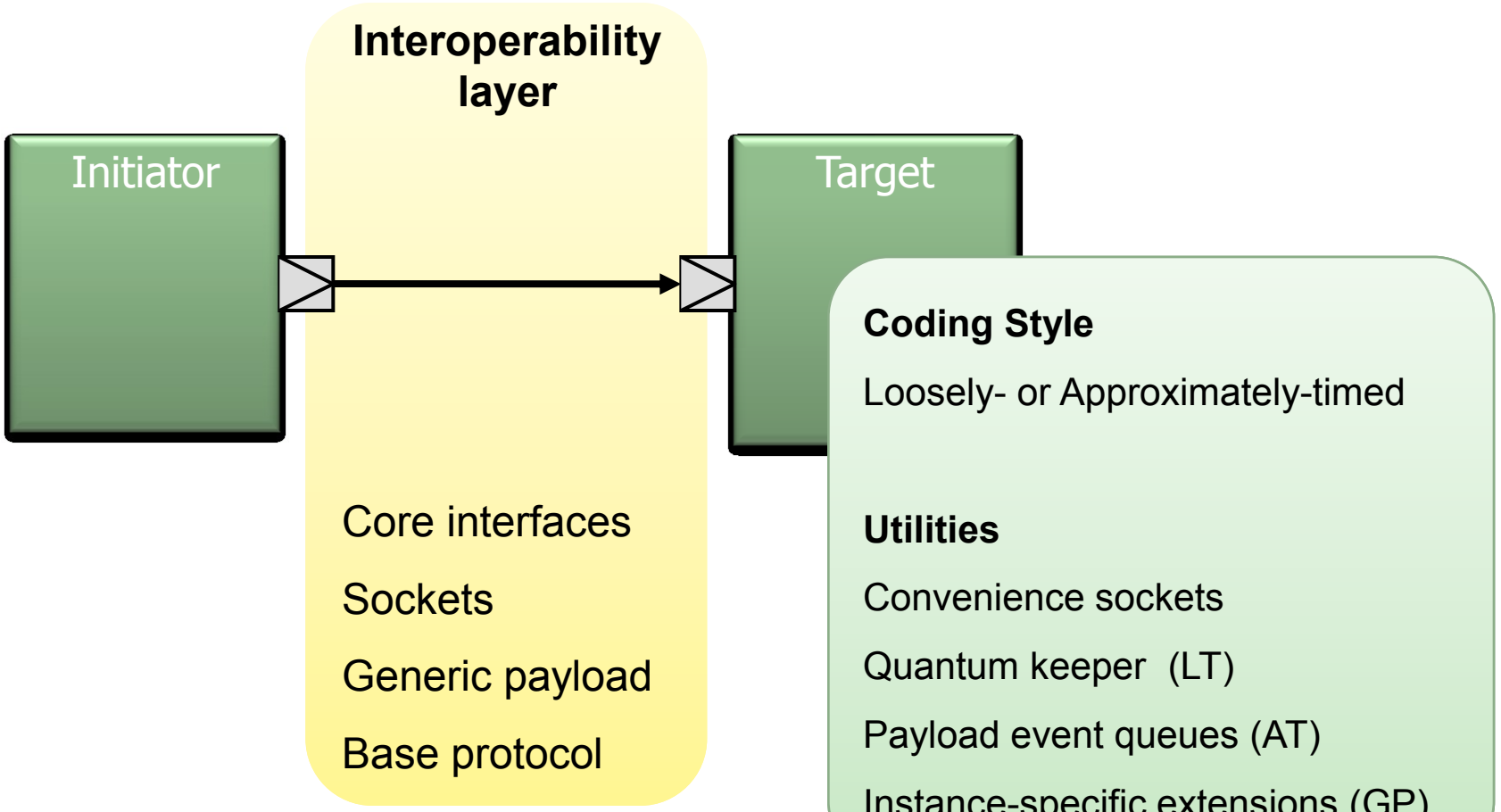
2. Generic payload



3. Base protocol



Utilities



- Productivity
- Shortened learning curve
- Consistent coding style

The TLM-2.0 Standard

CONTENTS

- Review of SystemC and TLM
- Review of TLM-2.0
- **Frequently Asked Questions**

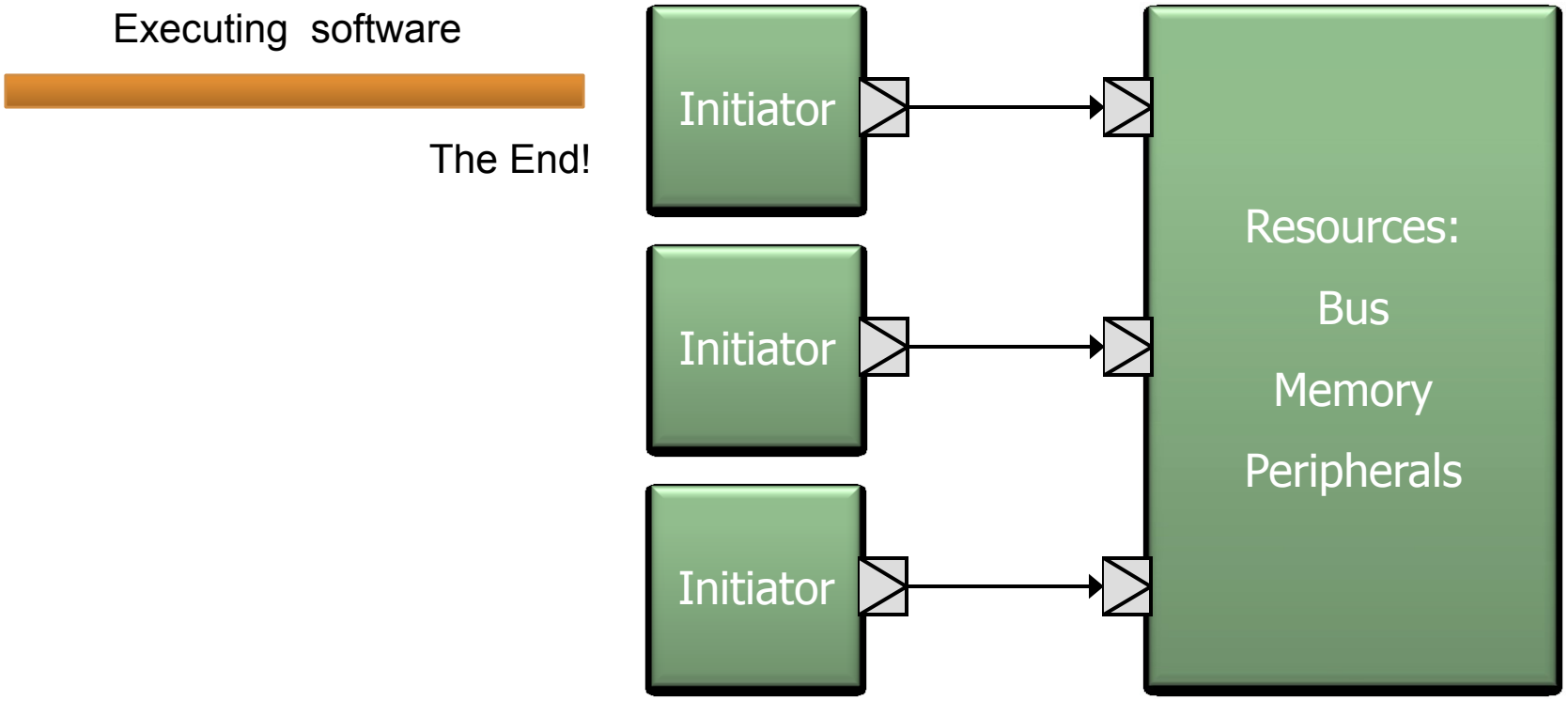
DVCon 2010
Design Verification Conference and Exhibition

DOULOS

FAQ #1

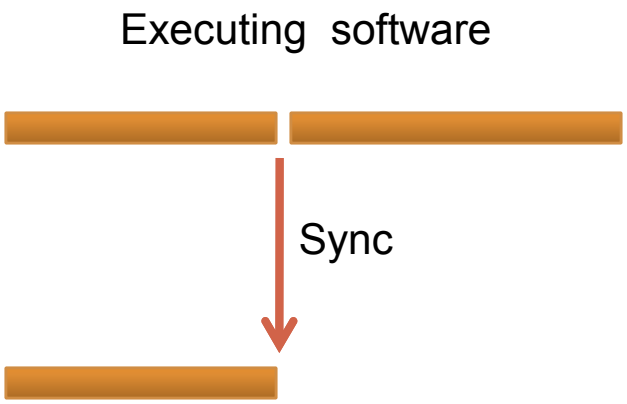
"Do I want LT or AT or CA?"

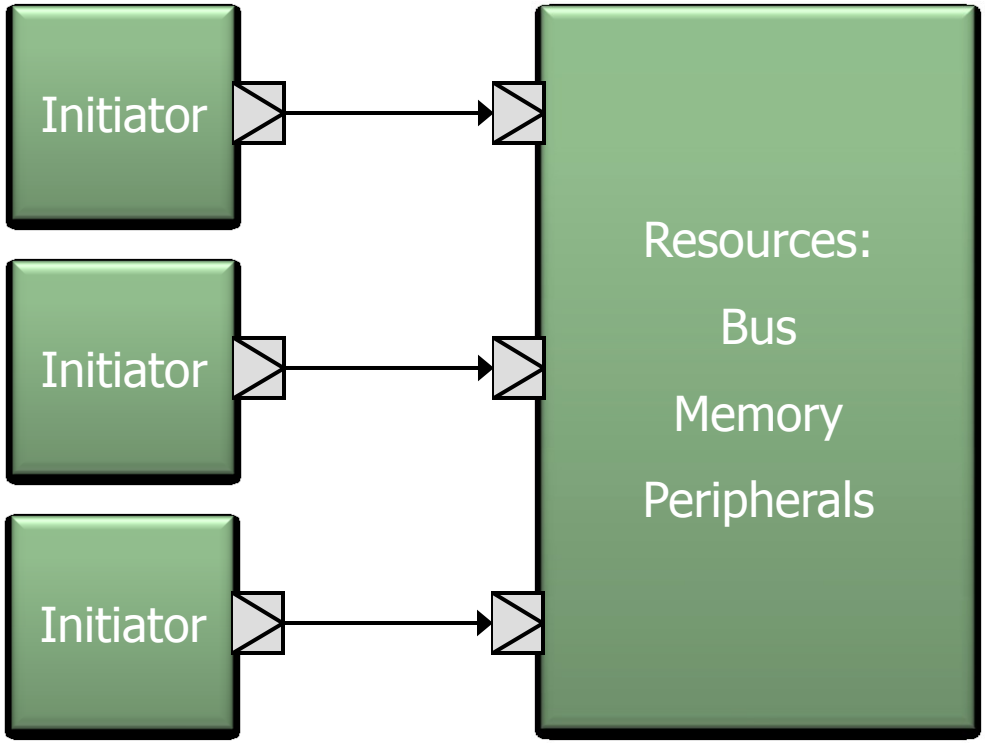
Loosely-Timed



- **Goal: execute software at full speed of host processor**
- Target models do not consume any time, initiators run ahead
- Need the initiators to take turns
- Either have explicit synchronization points (can be untimed)
- or initiators must use a time quantum

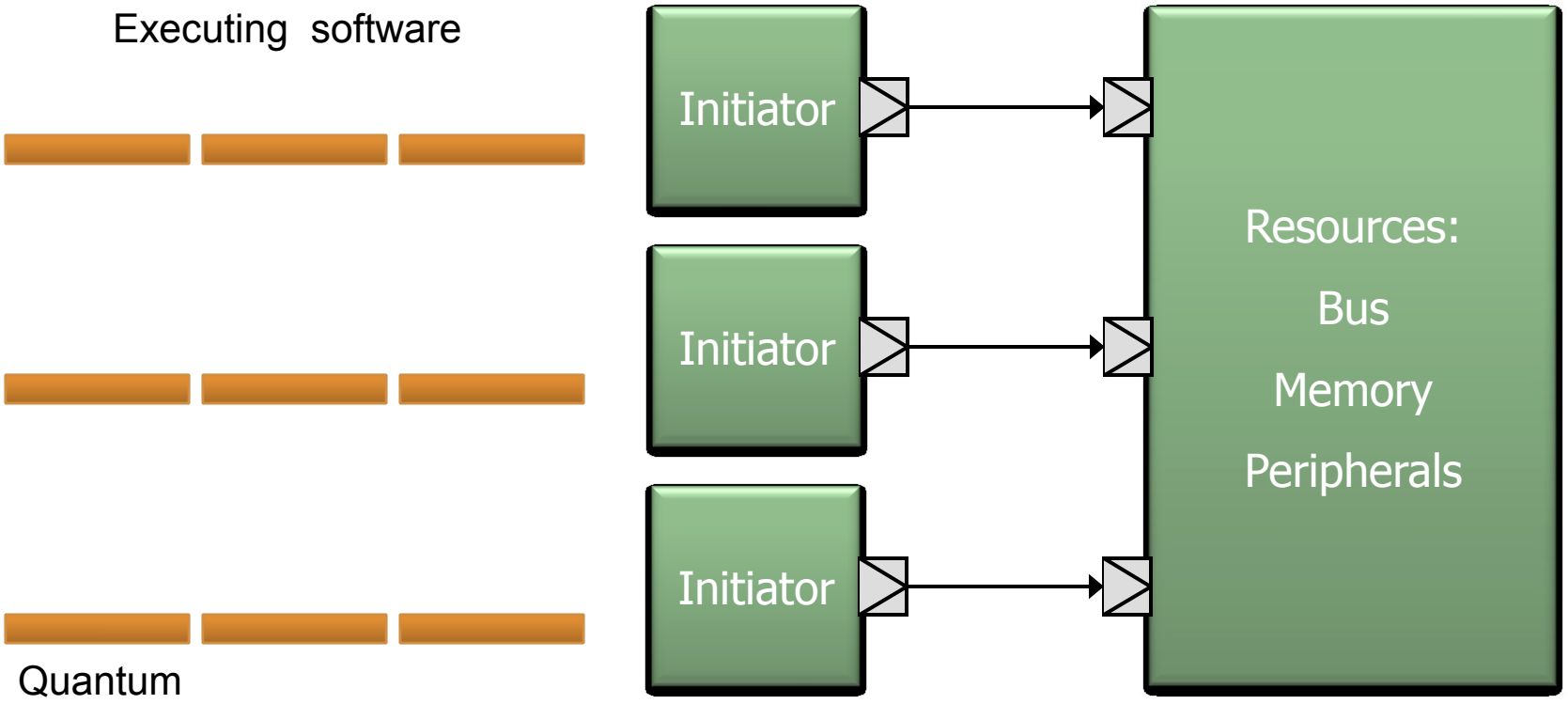
Loosely-Timed





- Goal: execute software at full speed of host processor
- Target models do not consume any time, initiators run ahead
- Need the initiators to take turns
- Either have explicit synchronization points (can be untimed)
- or initiators must use a time quantum

Loosely-Timed

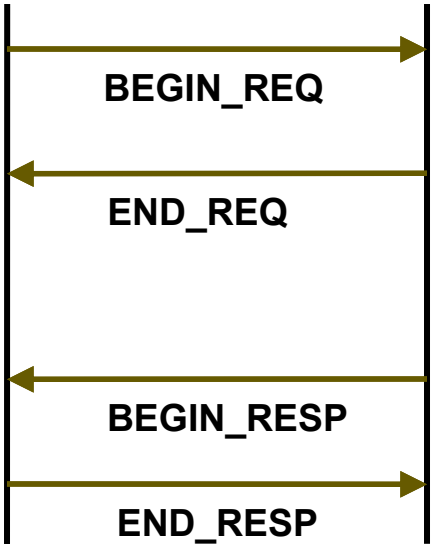


- Goal: execute software at full speed of host processor
- Target models do not consume any time, initiators run ahead
- Need the initiators to take turns
- Either have explicit synchronization points (can be untimed)
- or initiators must use a time quantum

AT and CA

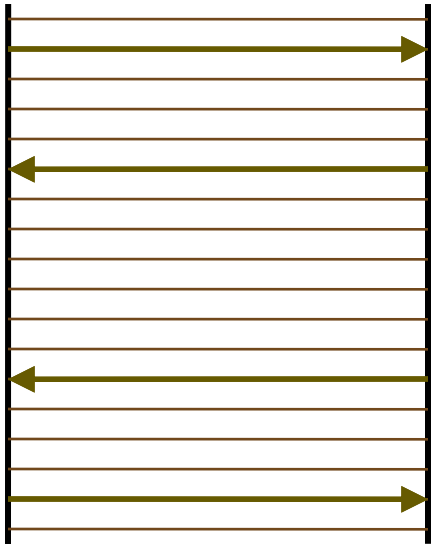
- No running ahead of simulation time; everything stays in sync

AT



Wake up at significant timing points

CA



Wake up every cycle

FAQ #2

"How do I model such-and-such a feature using TLM-2?"

"What does it take to make a model TLM-2-compliant?"

"How much of this 194-page LRM do I really need to understand?"

"How can TLM-2 make models of different protocols interoperable?"

First Kind of Interoperability

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

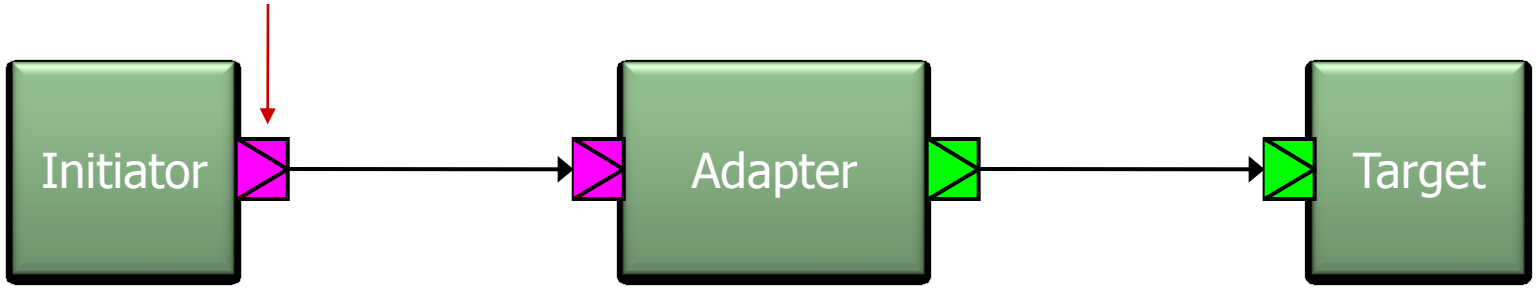
```
tlm_initiator_socket<32, tlm_base_protocol_types>  my_socket;
```

- Functional incompatibilities are still possible (e.g. writing to a ROM)

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

FAQ #2

Q. "How do I model such-and-such a feature using TLM-2?"

A. Either make do with the generic payload, or create extensions

Q. "What does it take to make a model TLM-2-compliant?"

A. Either obey the base protocol or make your own rules

Q. "How much of this 194-page LRM do I really need to understand?"

A. For the base protocol, everything except Utilities and TLM-1

Q. "How can TLM-2 make models of different protocols interoperable?"

A. Either map onto the base protocol, or write adapters

FAQ #3

Q. "How do I model my algorithm using TLM-2.0"

A1. Not applicable. TLM is communication-centric

A2. Use the LT/AT coding styles as guidelines

A3. Use the utilities at least as examples

FAQ #4

Q. "How do I use extensions? It's not really explained anywhere."

Create an Extension Class

```
struct  my_extension:  tlm::tlm_extension<my_extension>
{
  my_extension()  { m_attribute1 = ...; }

  virtual  tlm_extension_base*  clone() const
  {
    my_extension*  ext  =  new my_extension;
    ext->m_attribute1  =  this->m_attribute1;
    ext->m_attribute2  =  this->m_attribute2;
    return ext;
  }

  virtual void copy_from( tlm_extension_base  const&  ext )
  {
    m_attribute1 = static_cast<my_extension const &>(ext).m_attribute1;
    m_attribute2 = static_cast<my_extension const &>(ext).m_attribute2;
  }

  int      m_attribute1;
  string m_attribute2;
};
```

Standard code to clone/copy

Any number of attributes

Setting and Getting Extensions

Initiator

```
tlm_generic_payload*  trans;
...

my_extension* ext = new my_extension;
ext->m_attribute1 = 1;
ext->m_attribute2 = "foo";
trans->set_auto_extension( ext );

socket->b_transport( *trans, delay );
```

Target

```
virtual void b_transport( ... )
{
   my_extension* ext;
   trans.get_extension(ext);
   if (ext) {
      ... = ext->m_attribute1;
      ... = ext->m_attribute2;
  ...
}
```

Have the initiator add the extension

Target can test for an extension

FAQ #5

Q. "Okay, but you've not shown me how to create ignorable extensions.

How do I do that?"

A. Being ignorable is not an explicit property of an extension,

but about how you use it

Ignorable Extensions

An ignorable extension is able to be ignored

- Timestamp when the transaction was created
- An initiator ID or transaction ID

Extensions that might not be ignorable

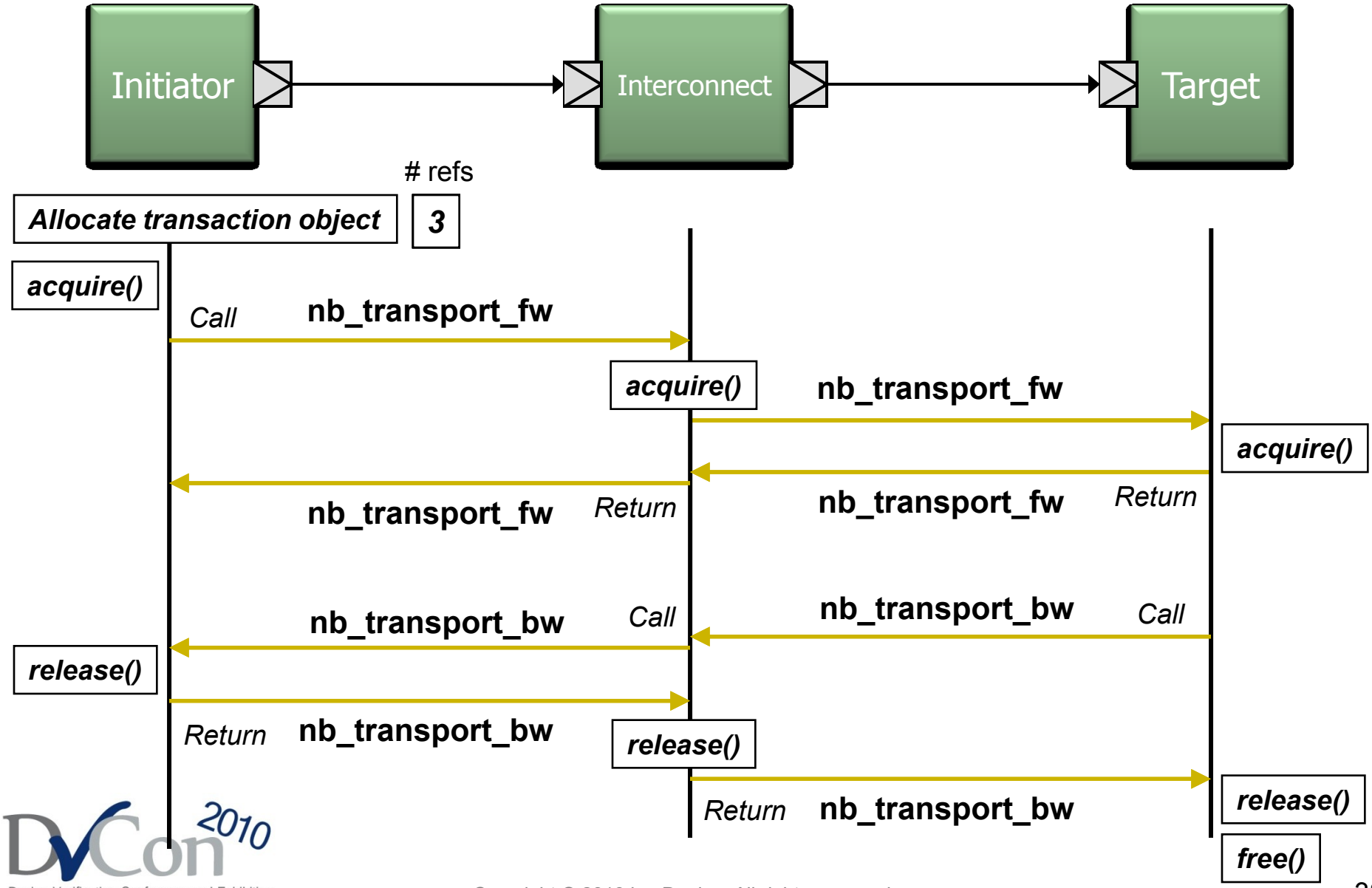
- An extended address (initiator might rely on extended address space)
- A priority (initiator might rely on high priority transaction executing first)
- A flag to lock the interconnect (initiator might rely on have exclusive access)

FAQ #6

Q. "How do I dispose of the extension object? Can I re-use it?"

A. Use a memory manager

Using a Memory Manager



Memory Management Methods

```
class  tlm_generic_payload  {
public:

   tlm_generic_payload () ;
   tlm_generic_payload(tlm_mm_interface* mm) ;
   virtual  ~tlm_generic_payload ();

   void  set_mm(tlm_mm_interface* mm);
   bool  has_mm();
   void  acquire();
   void  release();
   int     get_ref_count();

   void  deep_copy_from( const tlm_generic_payload& ) ;
   void  update_original_from( const tlm_generic_payload& );
   void  free_all_extensions();
   void  reset();
   ...
};
```

Frees all auto-extensions

A User-Defined Memory Manager

```
#include "tlm.h"
class gp_mm: public tlm::tlm_mm_interface
{
  typedef tlm::tlm_generic_payload gp_t;
public:
  gp_mm();
  virtual ~gp_mm();
  gp_t* allocate() {
    ...
    ptr = new gp_t( this );
    ...
  }
  void free(gp_t* trans) {
    trans->reset();
    ...
  }
  ...
```

Pass mm as constructor arg

Called when ref count == 0

Free auto-extensions

Typical Coding Idiom

```
my_module(sc_module_name _n, gp_mm* mm)
: m_mm(mm) {...}
```

Pass mm as constructor arg

```
tlm_generic_payload*  trans;
trans = m_mm.allocate();
trans->acquire();
```

Get transaction object from mm

Increment reference count

```
my_extension*  ext = new my_extension;
ext->m_attribute1 = 1;
ext->m_attribute2 = "foo";
trans->set_auto_extension( ext );
```

Allocate extension on heap

Set for auto-deletion

```
socket->b_transport( *trans, delay );
```

Not just for nb_transport

```
trans->release();
```

Decrement reference count

FAQ #7

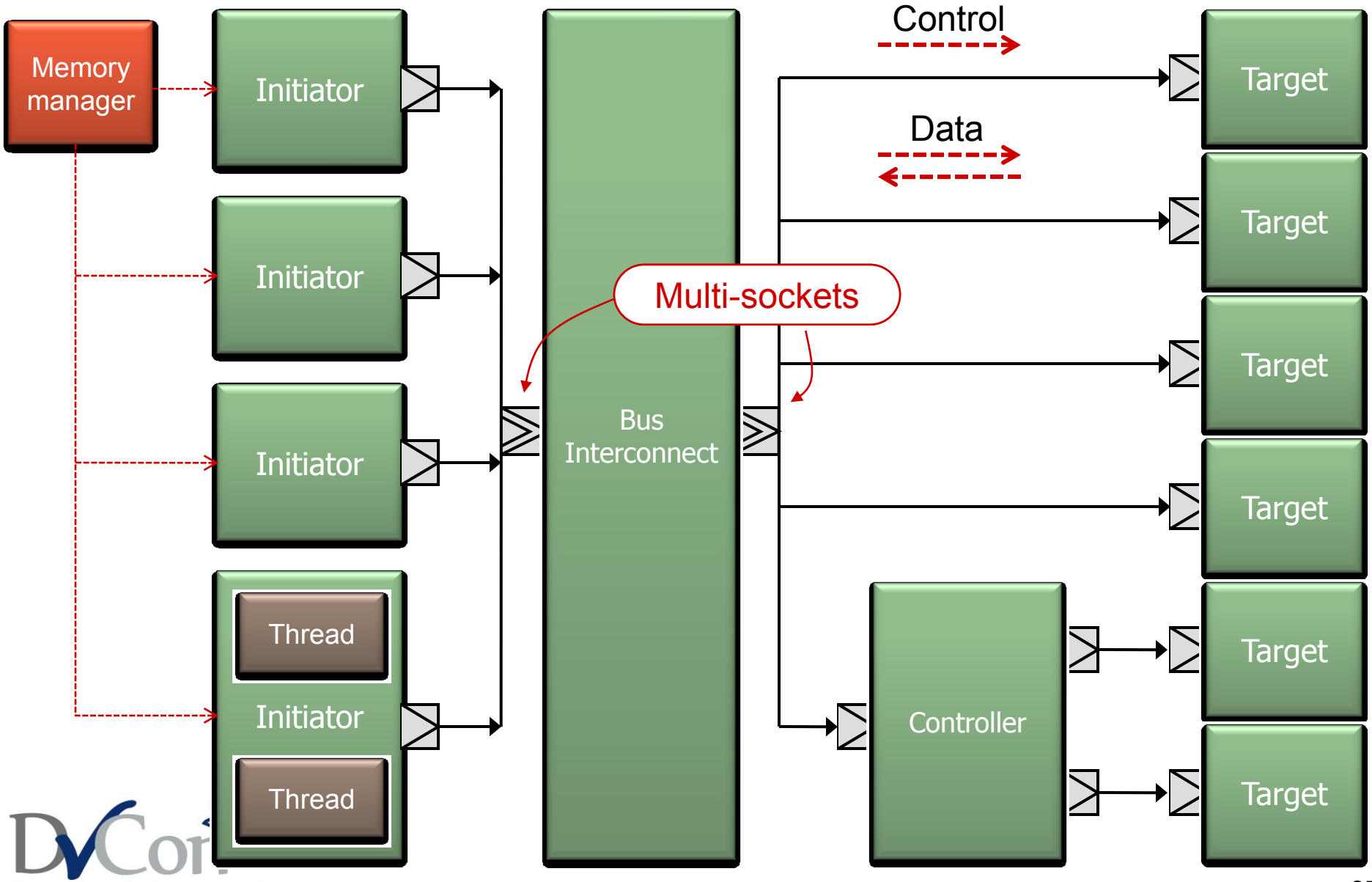
Q. "How do I connect multiple components together?"

Q. "How do I model a bus with multiple masters/slaves?"

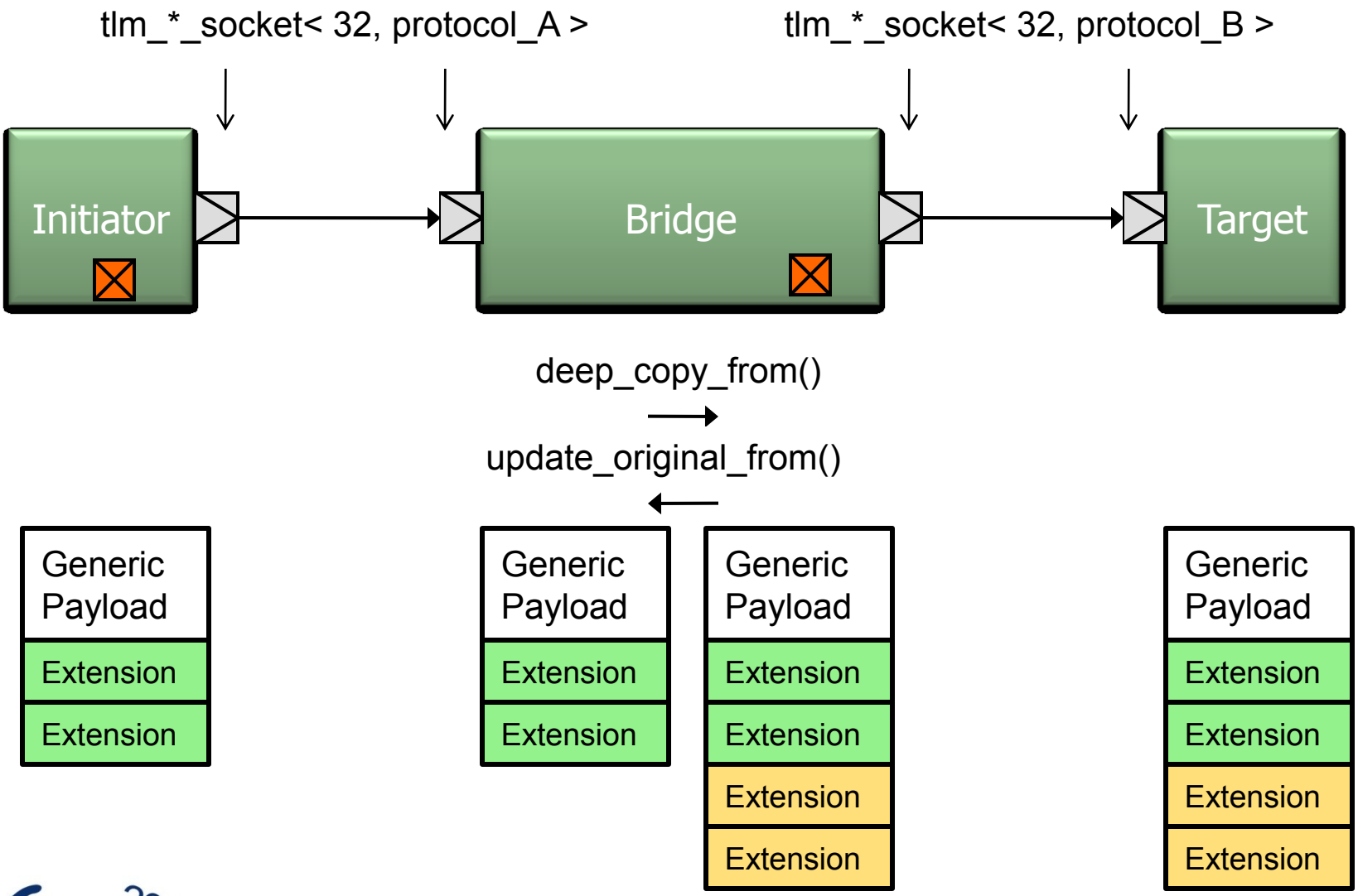
Q. "How many sockets do I need for two-way communication?"

Q. "How many transactions do I use?"

Example Topology



Bridges



Could pass the *same* transaction if their lifetimes allow

Copying the Data Array

- Deep-copy the data array in the bridge

```
new_trans->set_data_ptr( &data_buffer );
new_trans->deep_copy_from ( original_trans );
new_trans->b_transport(...);
original_trans->update_original_from( new_trans );
```

Copies back data array on read

- Shallow-copy the data array in the bridge

```
new_trans->set_data_ptr( 0 );
new_trans->deep_copy_from ( original_trans );
new_trans->set_data_ptr( original_trans->get_data_ptr() );
new_trans->b_transport(...);
original_trans->update_original_from( new_trans );
```

Does not touch data array

FAQ #7

Q. "How do I connect multiple components together?"

A. By having one or more components act as a hub

Q. "How do I model a bus with multiple masters/slaves?"

A. The "hub" can do arbitration and routing. Use multi-sockets for convenience

Q. "How many sockets do I need for two-way communication?"

A. Each initiator needs an initiator socket, each target a target socket

Q. "How many transactions do I use?"

A. As few as possible

FAQ #8

Q. "How do I model something like a SPI port?"

A. If you can abstract the functionality, use the base protocol

Q. "How do I model something like a SPI port with accurate timing?"

A. You can model precise timing with the AT coding style, but why not use RTL?

Q. "How do I model something like AMBA, PCI, or USB?"

A. Buy a model (or get one through a university/research program)

For More FREE Information

- IEEE 1666

  standards.ieee.org/getieee/1666/index.html

- OSCI SystemC 2.2 and TLM-2.0

  www.systemc.org

- Tutorial introduction to TLM-2.0 and *Free TLM-2.0 Protocol Checker*

  www.doulos.com/knowhow/systemc/tlm2

System Design

SystemC
ARM • C++

Verification Methodology

e **• PSL • SCV**
SystemVerilog

Hardware Design

VHDL • Verilog
Altera • Xilinx
Perl • Tcl/Tk