Virtual Prototyping using SystemC and TLM-2.0

John Aynsley, Doulos

accellera SYSTEMS INITIATIVE

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

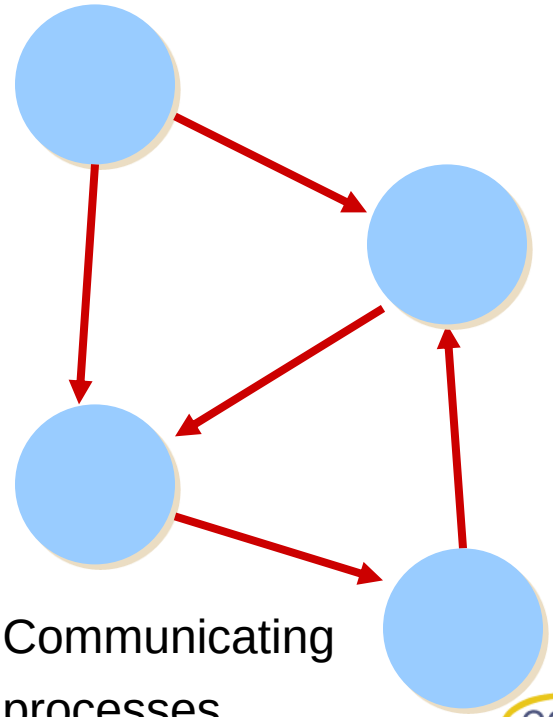
Virtual Prototyping using SystemC and TLM-2.0

- Introduction
- Development of TLM-2.0
- TLM-2.0 Sockets and Interfaces
- LT, AT, and CA
- Generic Payload and Extensions
- Interoperability

What is SystemC?

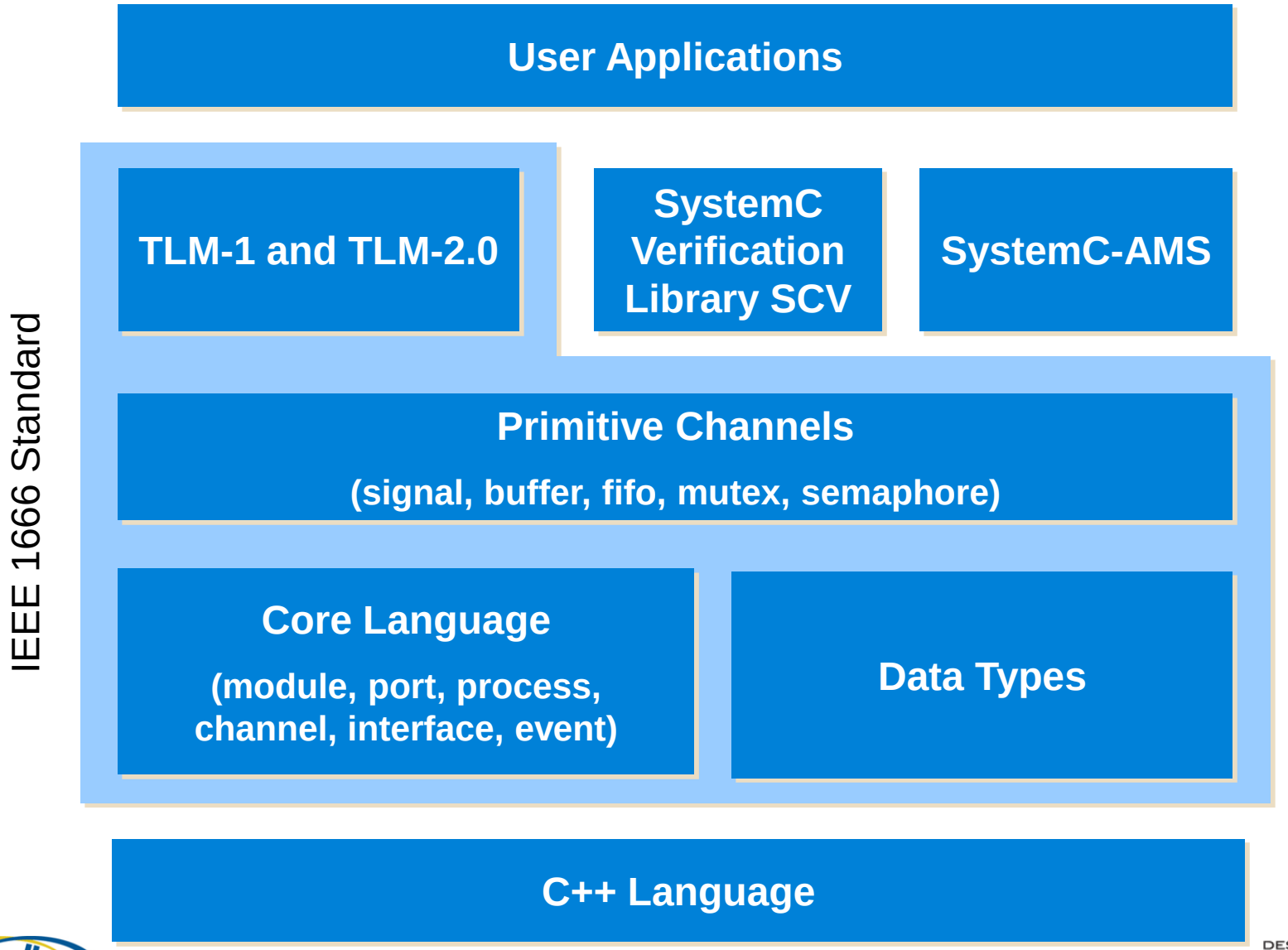
- System-level modeling language
  - Network of communicating processes (c.f. HDL)
  - Supports heterogeneous models-of-computation
  - Models hardware and software, digital and analog

- C++ class library
  - Open source proof-of-concept simulator
  - Owned by Accellera (previously OSCI)



Communicating processes

Architecture of SystemC



TLM Standardization

Apr 2005

- TLM-1.0
- Unidirectional message-passing interfaces

Jun 2008

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

July 2009

- LRM
- TLM-2.0.1

2011

- TLM-1 and TLM-2.0 part of IEEE 1666
- TLM-2.0.2

- Beyond SystemC ...

Jan 2008

- SystemVerilog OVM - UVM

What can you do with SystemC?

- Discrete Event Simulation (events, time)
  - Register Transfer Level (delta delays, bus resolution)
  - Behavioral Modeling (functions, processes, parallelism)
  - Transaction Level (communication using function calls)
  - Kahn Process Networks (infinite fifos, reads block when empty)
  - Dataflow (input-execution-output stages)
  - CSP (rendezvous, blocking reads & writes)

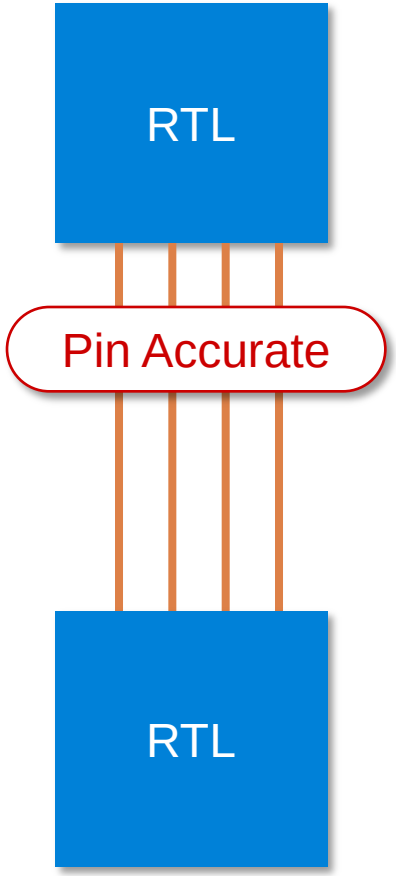
- Analog at ESL (SystemC AMS)
- High-Level Synthesis (synthesis subset)

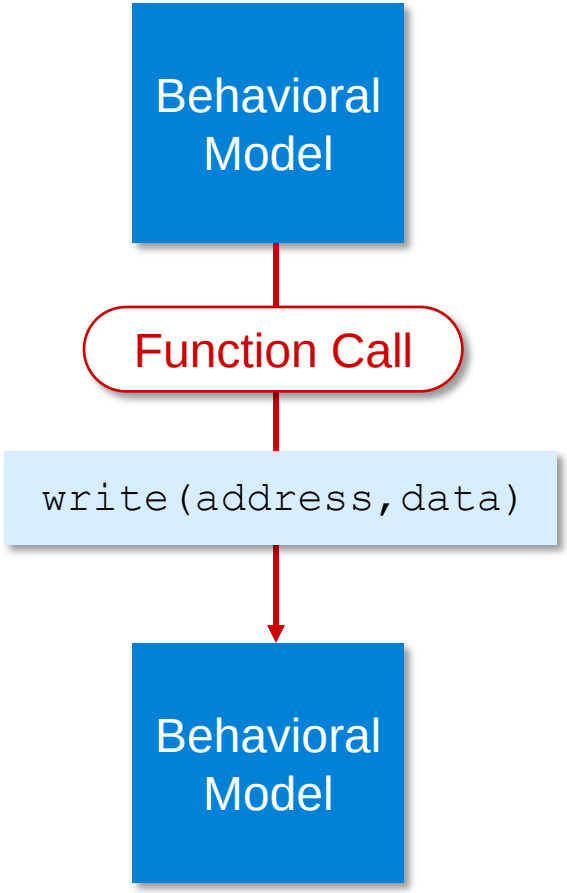
- NOT gate level
- NOT Spice level
- NOT software / RTOS modeling (process scheduling, priorities, preemption)

What is SystemC Used For?

- Behavioral Modeling and Reference Models
- Virtual Platforms (aka Software Virtual Prototypes)
  - Architectural exploration, performance modeling
  - Software development
  - Reference model for functional verification
- High-Level Synthesis (C/C++)

Transaction-Level Modeling

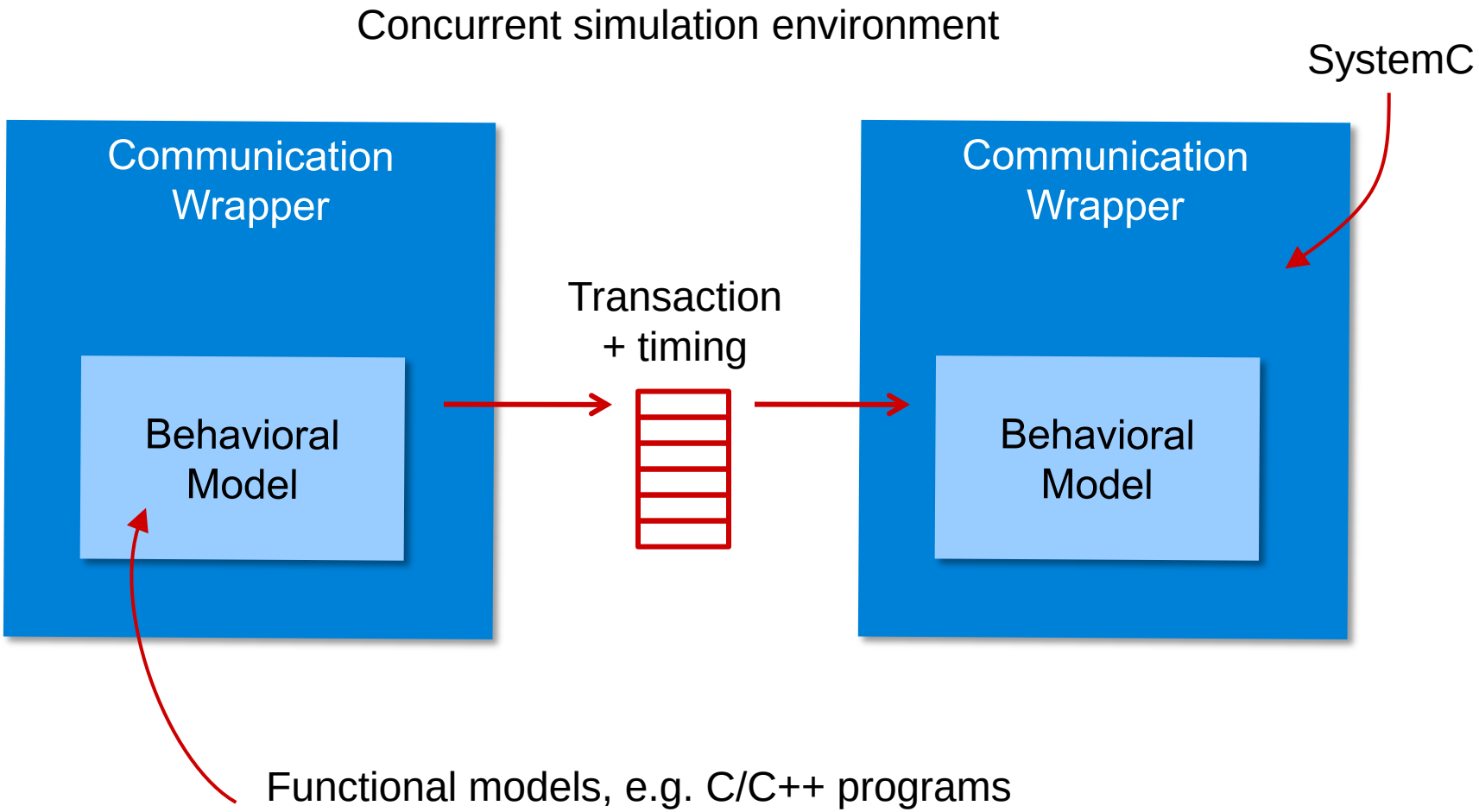




Simulate every event!

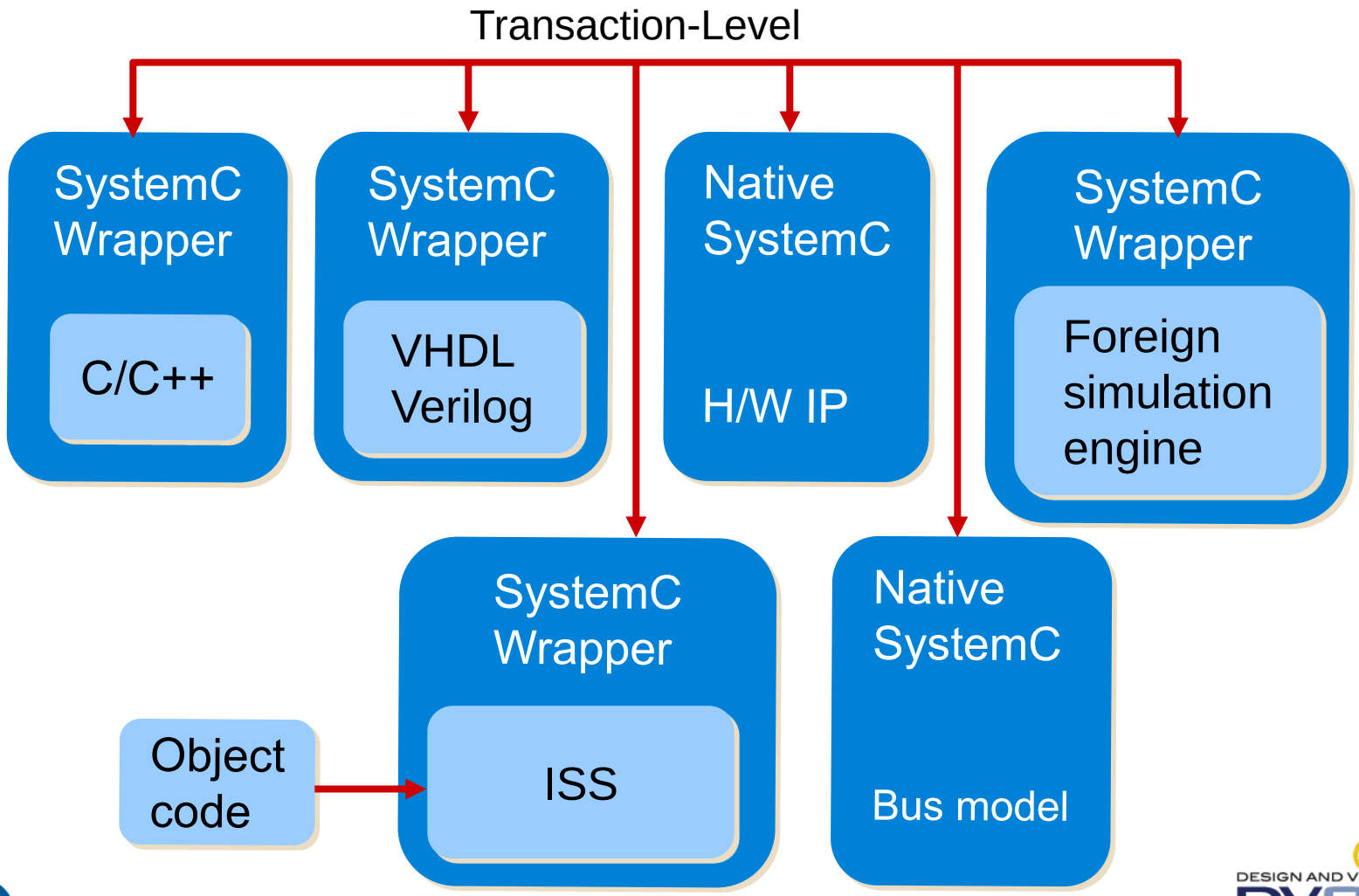
100-10,000 X faster simulation!

Transaction-Level Modeling

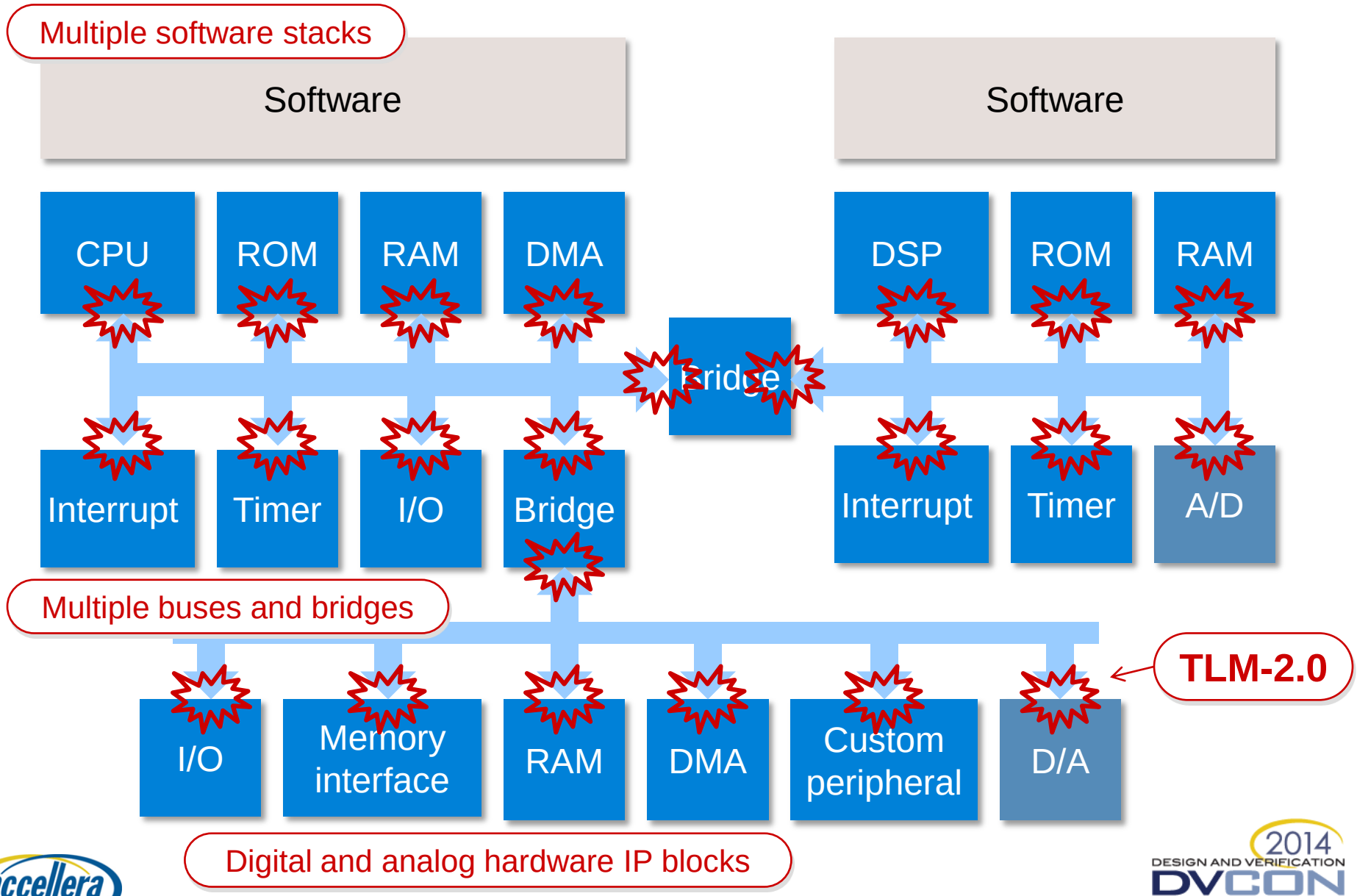


Use Model: SystemC as Glue!

- Transaction-level modeling is communication-centric



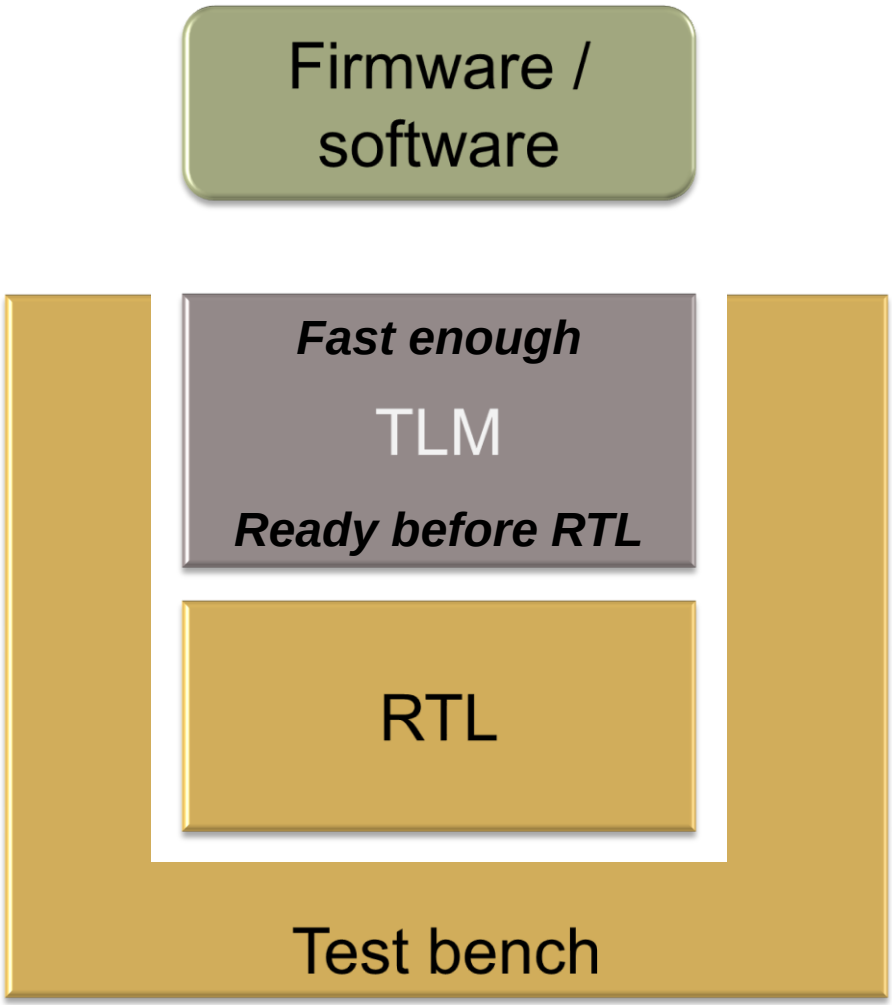
Typical Use Case: Virtual Platform



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 ✓

What is TLM Used For?



Accelerates product release schedule

Software development

Architectural exploration

Hardware (co)verification

TLM = golden model

Virtual Prototyping using SystemC and TLM-2.0

- Introduction
- Development of TLM-2.0
- TLM-2.0 Sockets and Interfaces
- LT, AT, and CA
- Generic Payload and Extensions
- Interoperability

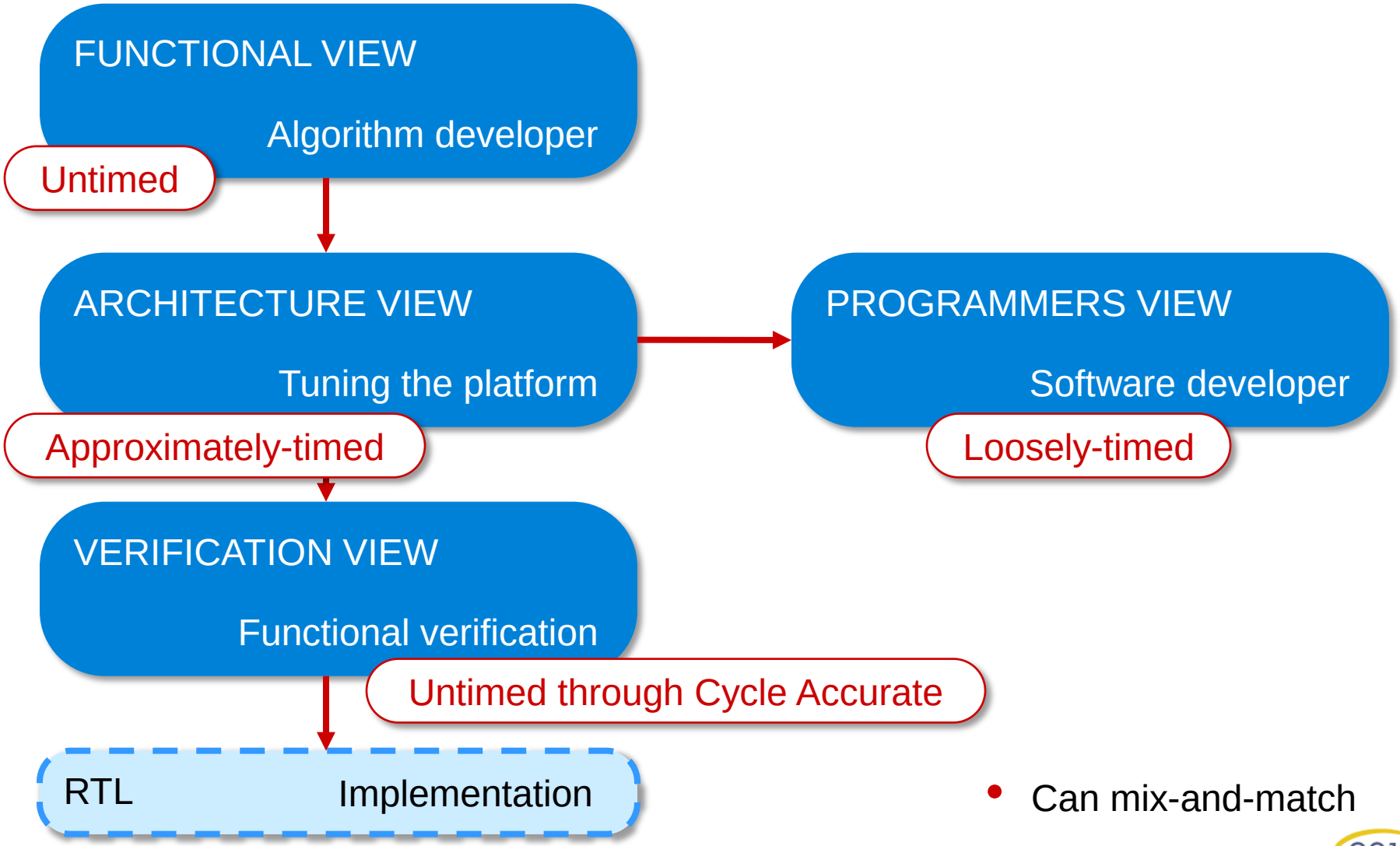
TLM-2 Requirements

- Focus on memory-mapped bus modeling
- Not meant to exclude non-bus protocols

Speed!

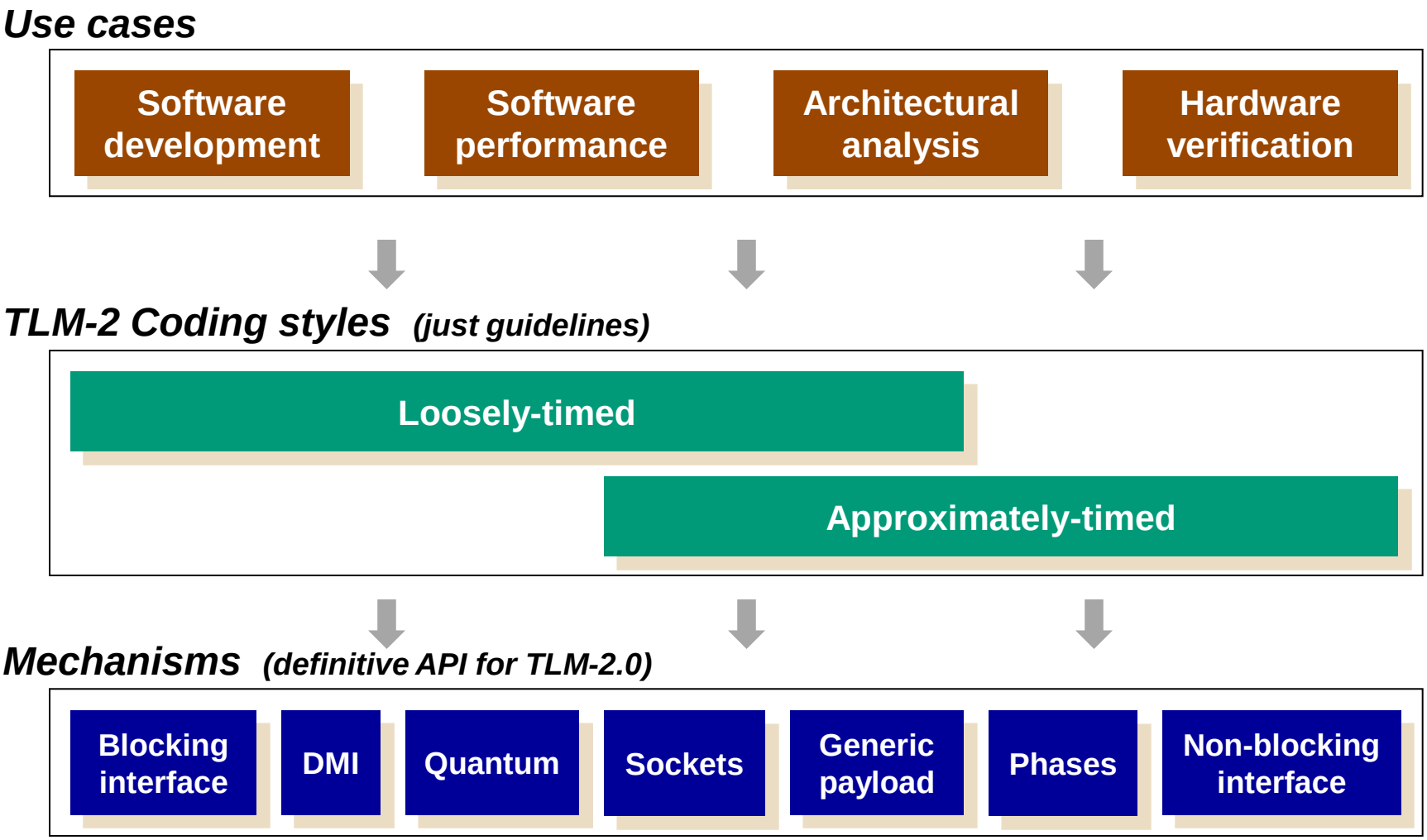
Interoperability!

Multiple Abstraction Levels



- Can mix-and-match

Use Cases, Coding Styles and Mechanisms



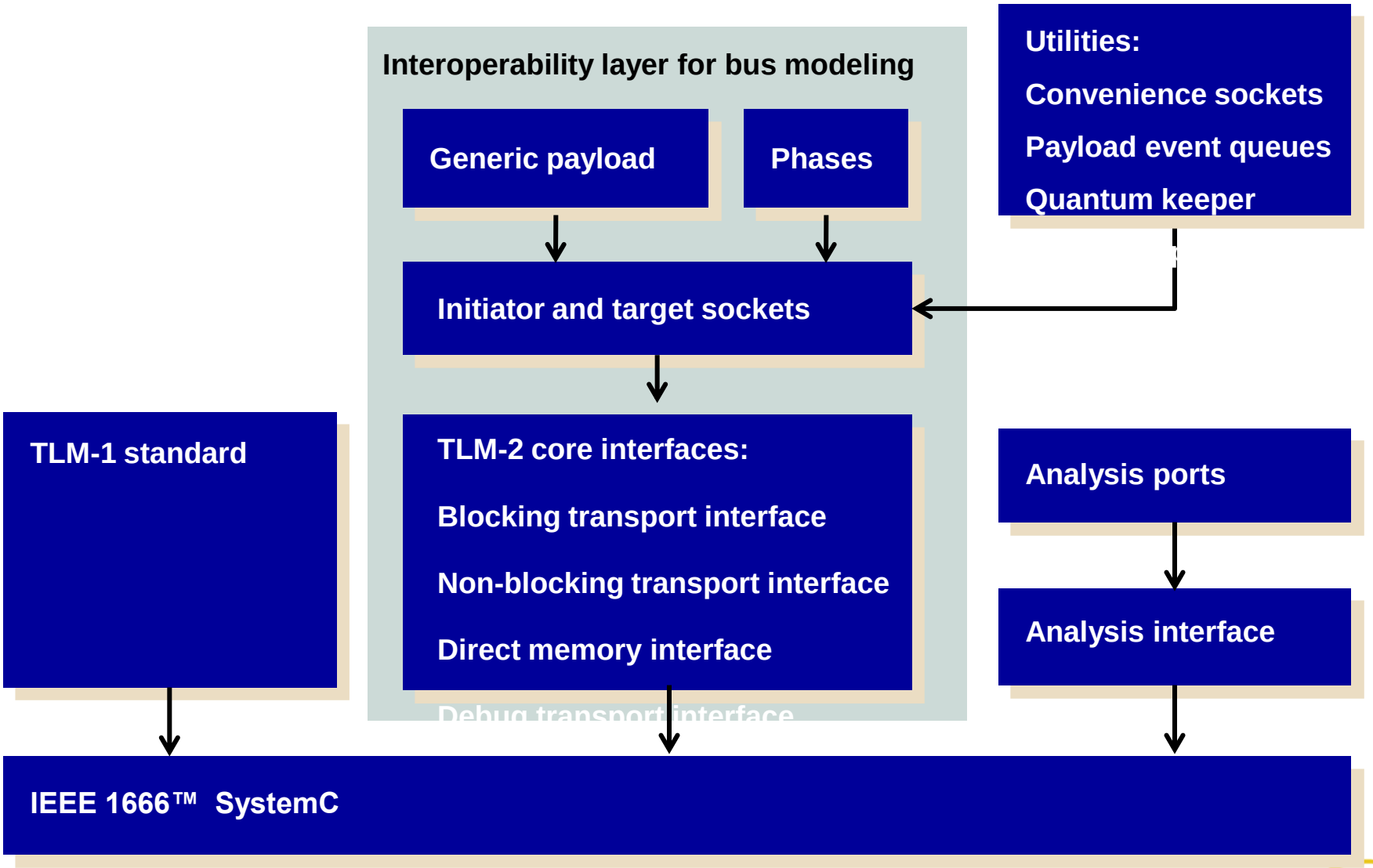
Coding Styles

- Loosely-timed = as fast as possible
  - Only sufficient timing detail to boot O/S and run multi-core systems
  - Processes can run ahead of simulation time (temporal decoupling)
  - Each transaction completes in one function call
  - Uses direct memory interface (DMI)

- Approximately-timed = just accurate enough for performance modeling
  - *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)

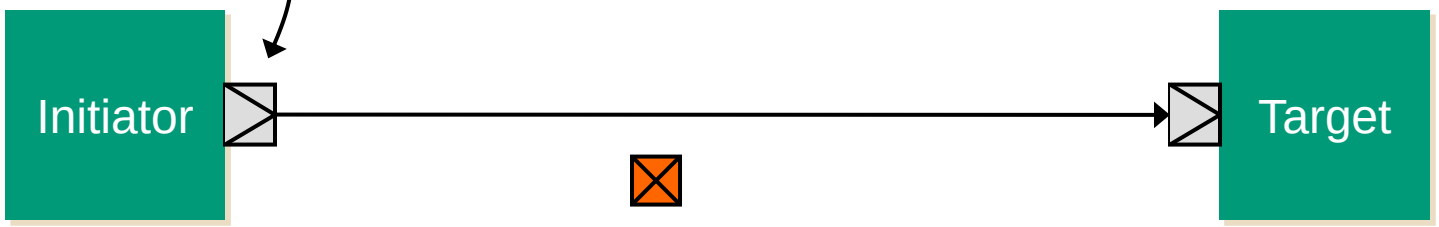
- Guidelines only – not definitive

The TLM 2.0 Classes

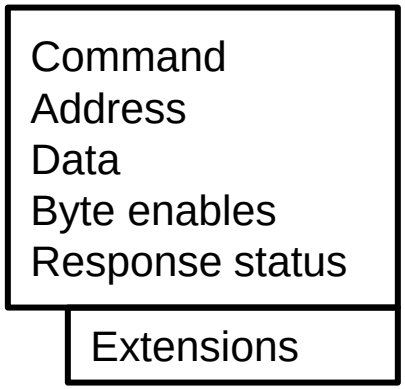


Interoperability Layer

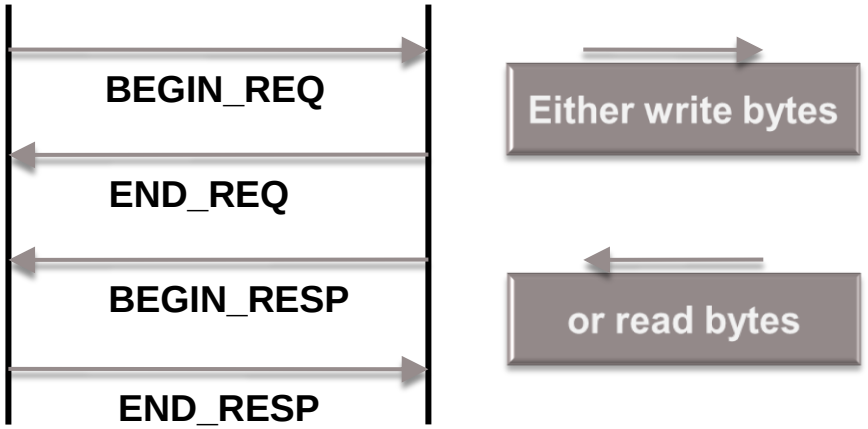
1. Core interfaces and sockets



2. Generic payload

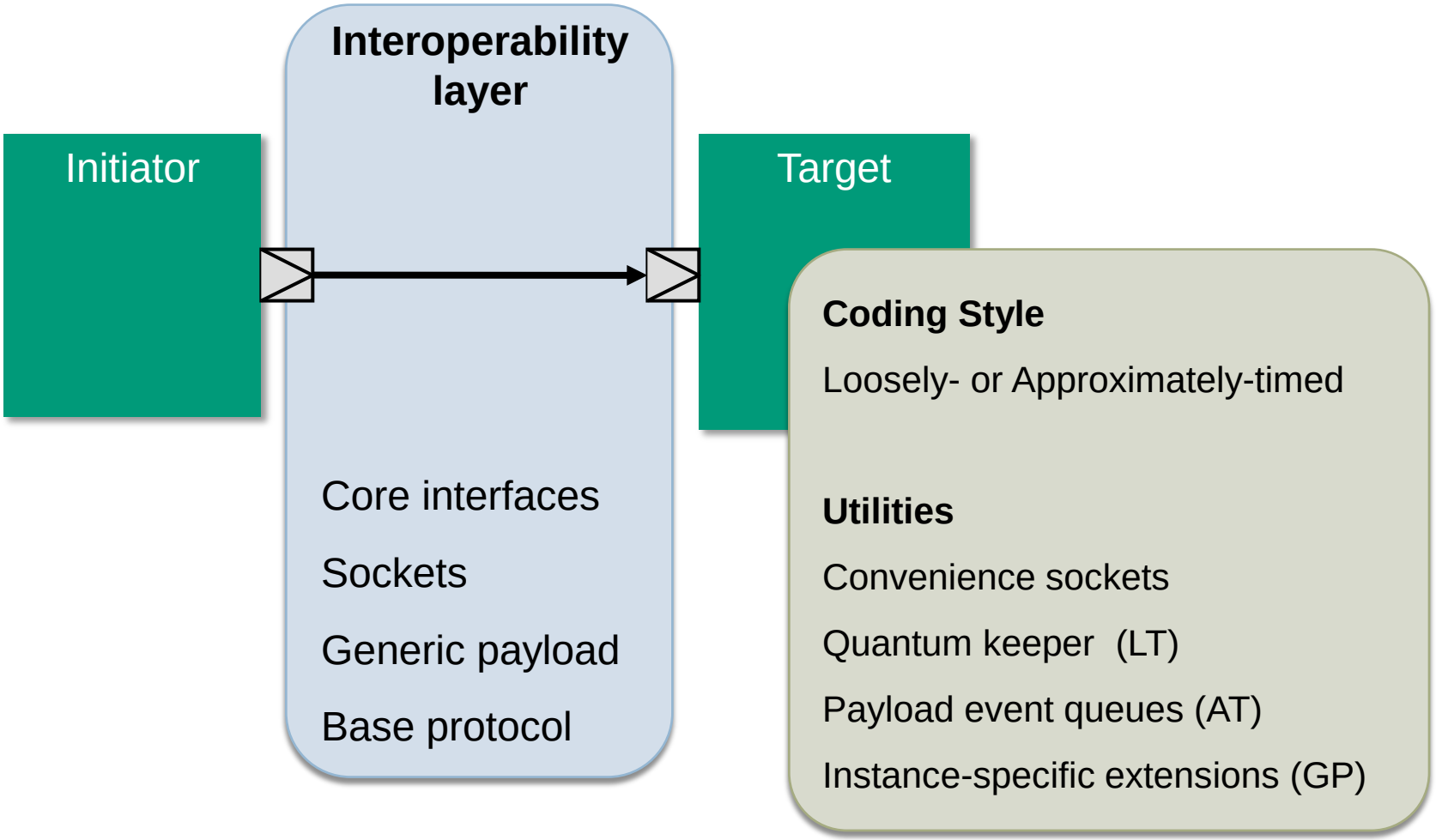


3. Base protocol



- Maximal interoperability for memory-mapped bus models

Utilities

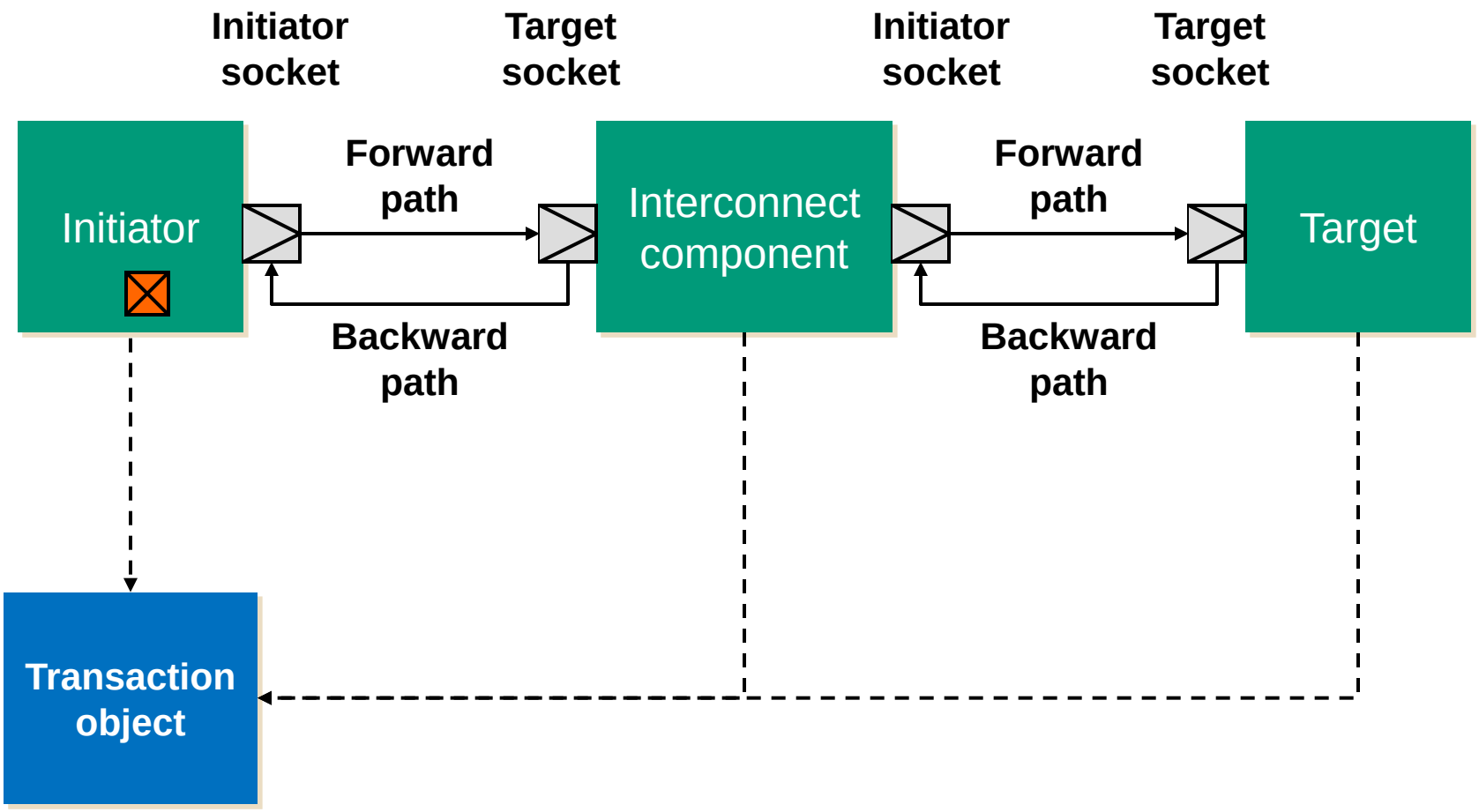


- Productivity
- Shortened learning curve
- Consistent coding style

Virtual Prototyping using SystemC and TLM-2.0

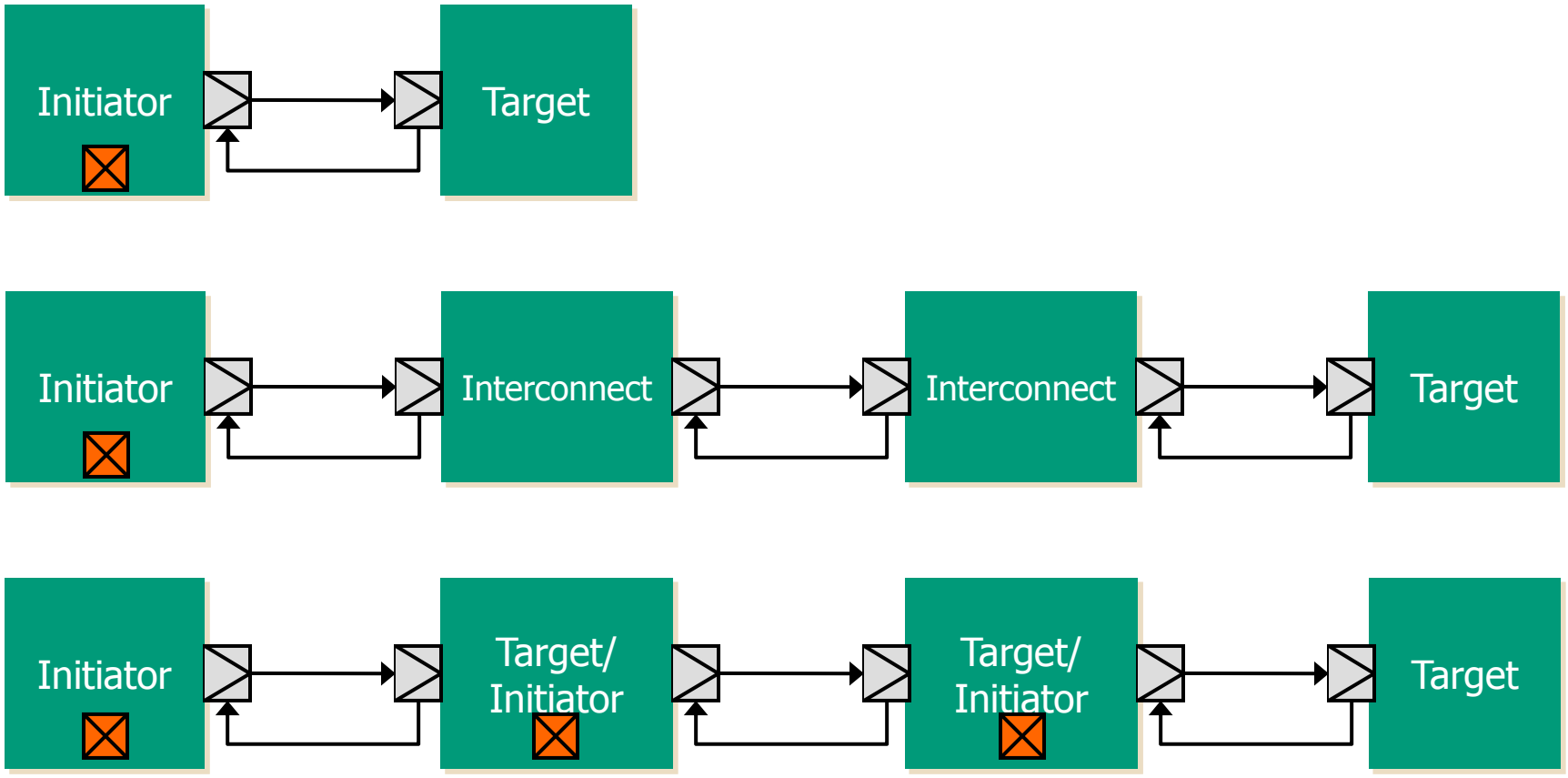
- Introduction
- Development of TLM-2.0
- TLM-2.0 Sockets and Interfaces
- LT, AT, and CA
- Generic Payload and Extensions
- Interoperability

Initiators and Targets



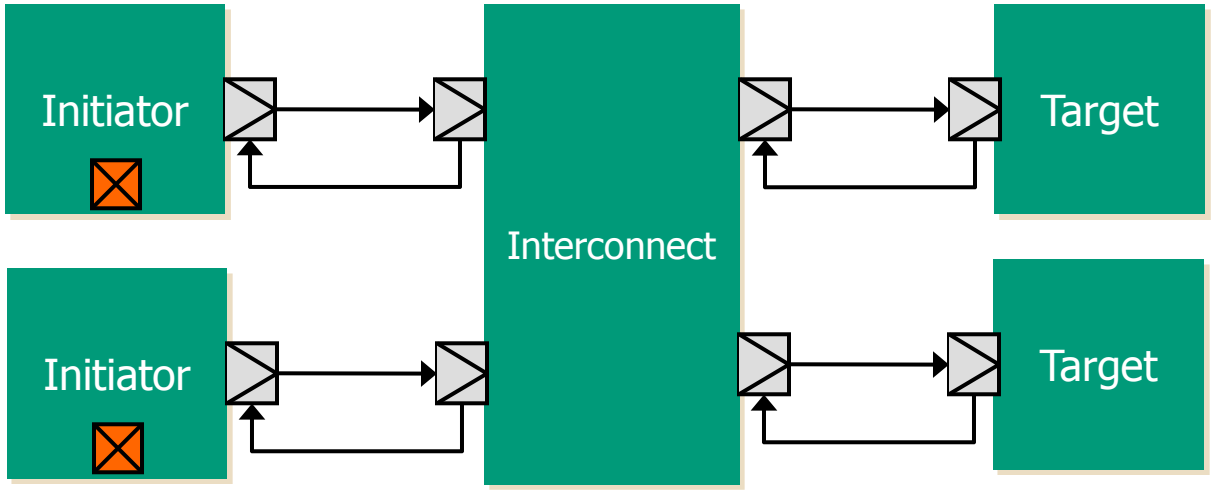
References to a single transaction object are passed along the forward and backward paths

TLM-2 Connectivity



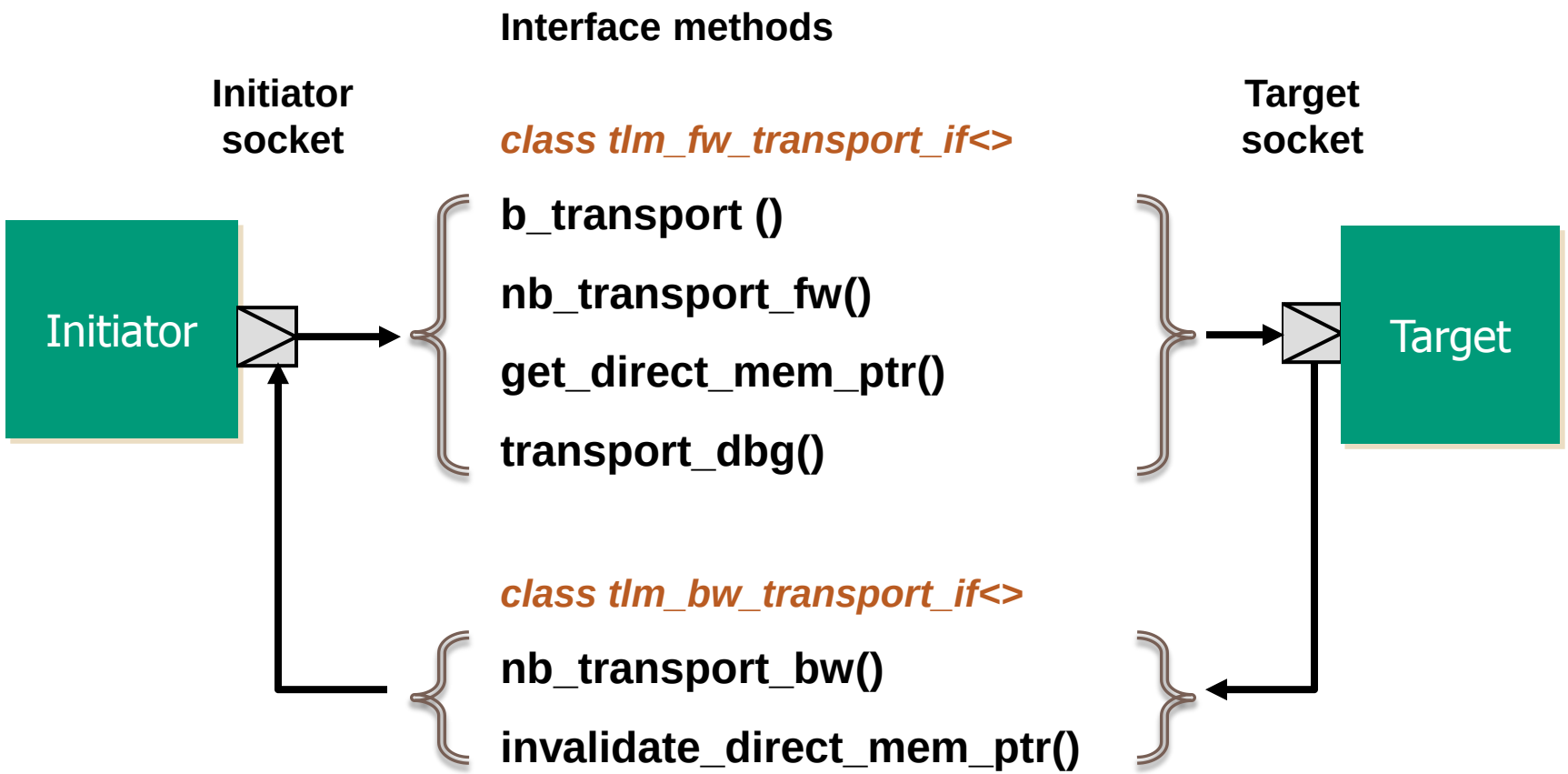
- Roles are dynamic; a component can choose whether to act as interconnect or target
- Transaction memory management needed

Convergent Paths



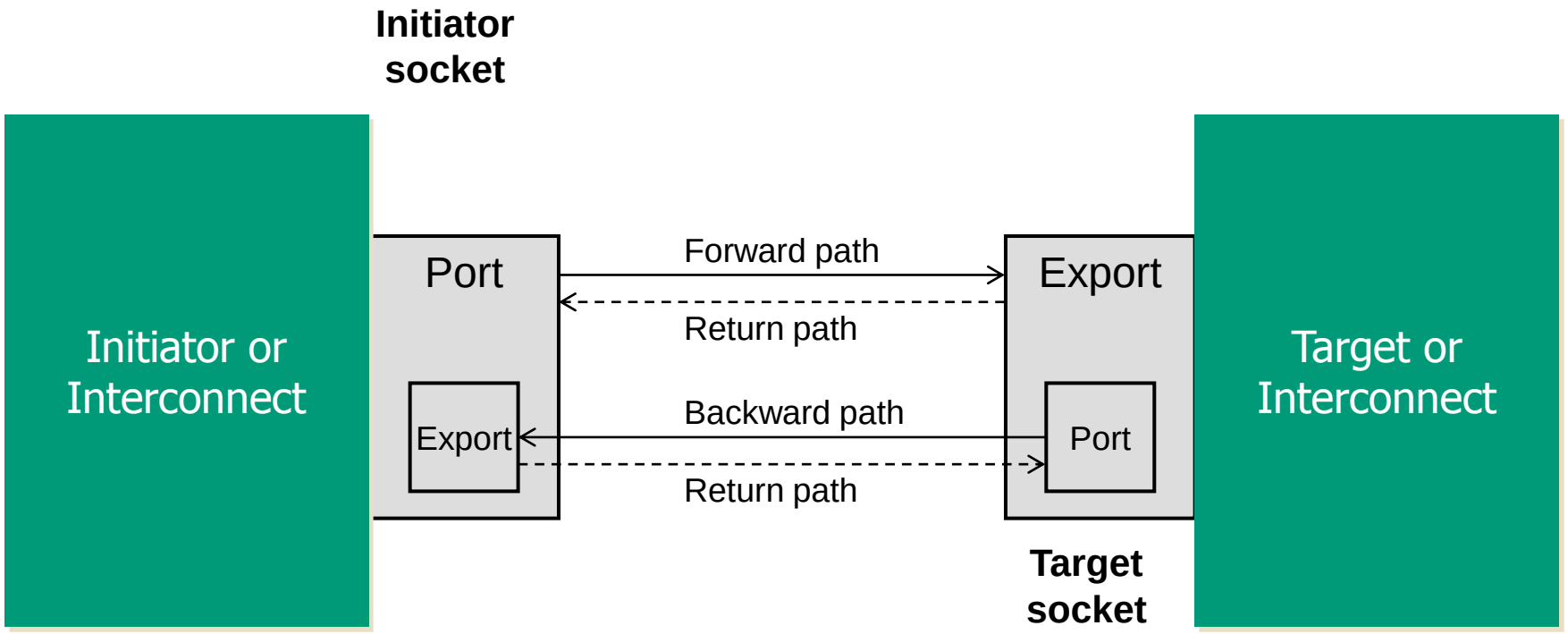
- Paths not predefined; routing may depend on transaction attributes (e.g. address)
- Whether arbitration is needed depends on the coding style

Initiator and Target Sockets

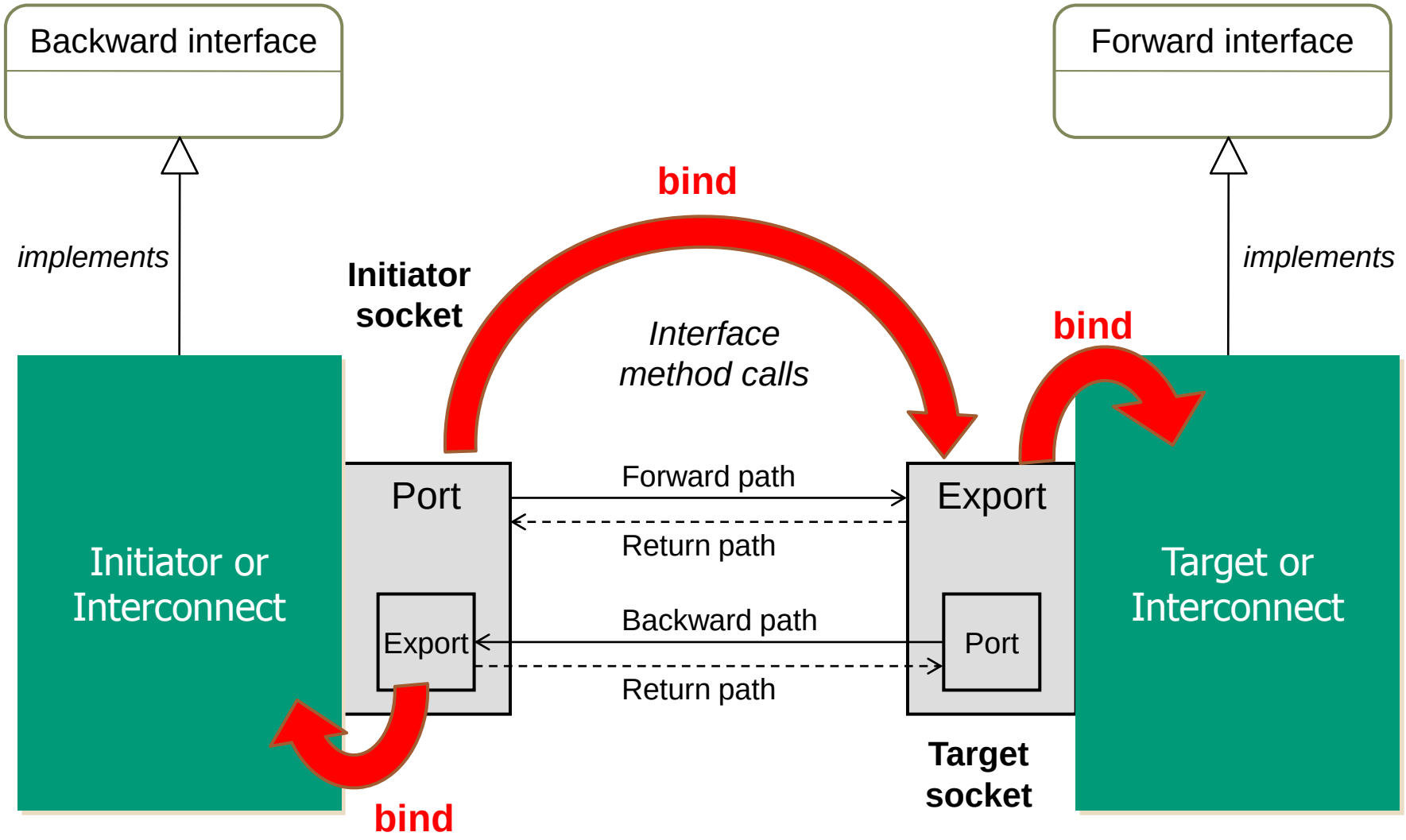


- Sockets provide fw and bw paths, and group interfaces

Sockets, Ports and Exports



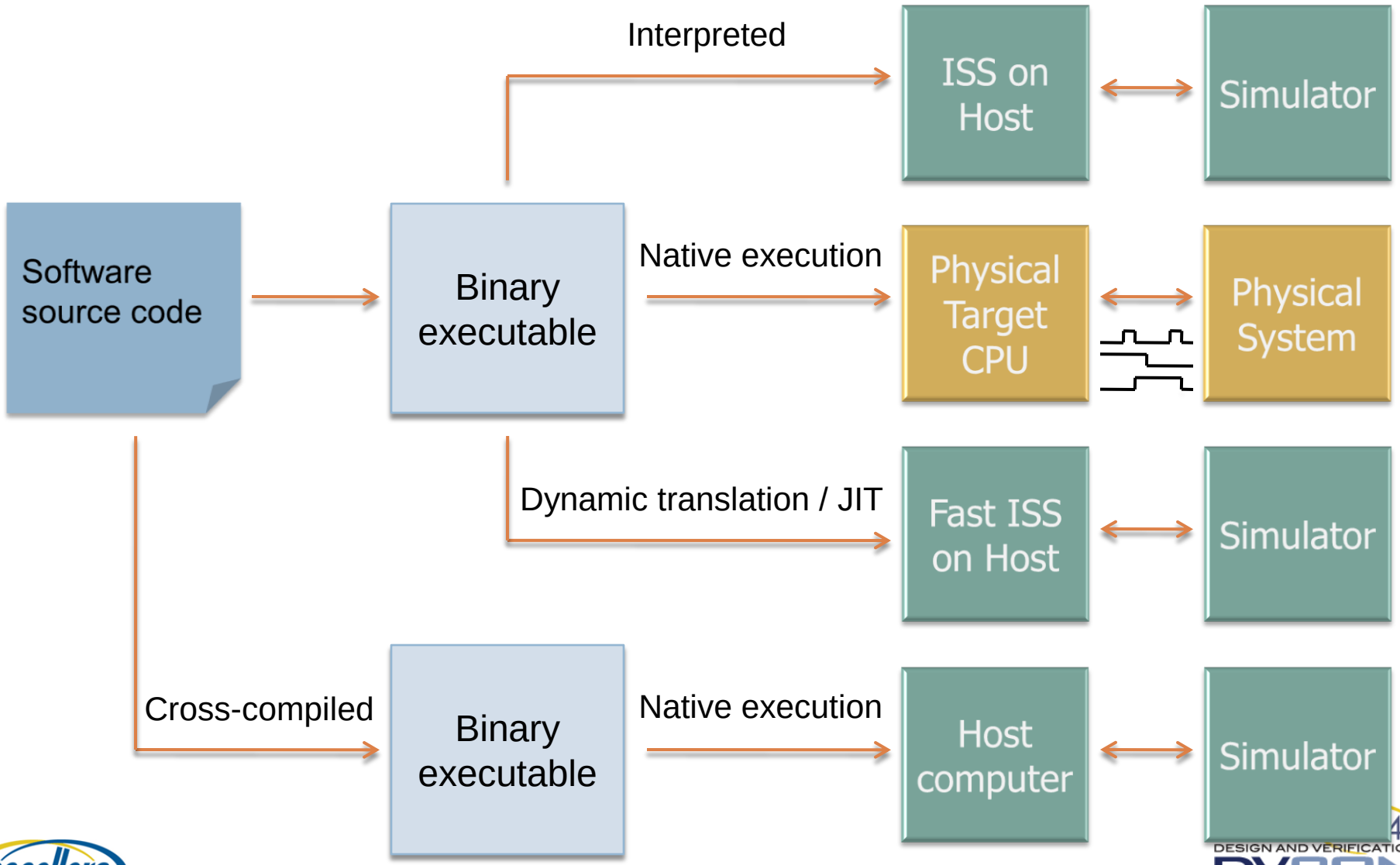
Socket Binding and Interfaces



Virtual Prototyping using SystemC and TLM-2.0

- Introduction
- Development of TLM-2.0
- TLM-2.0 Sockets and Interfaces
- LT, AT, and CA
- Generic Payload and Extensions
- Interoperability

Software Execution and Simulation

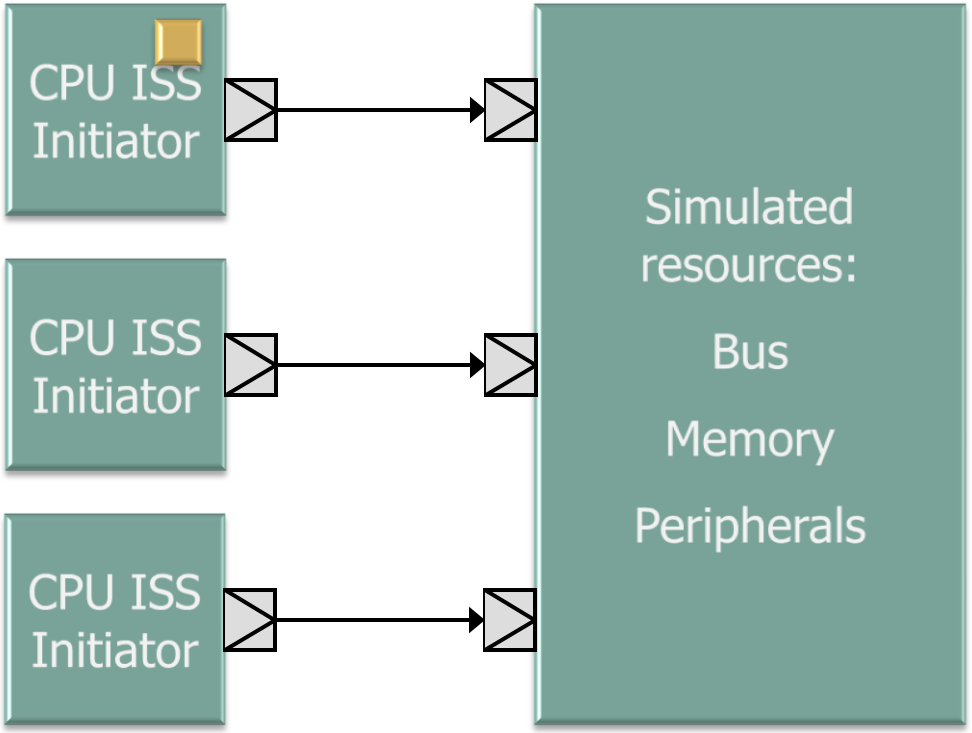


Software Execution without Sync

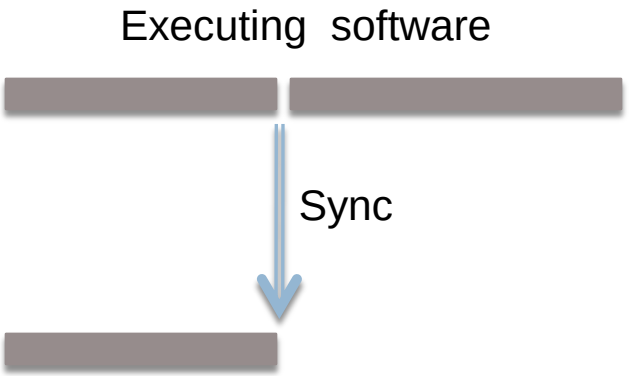
Executing software

Initiator model does not yield

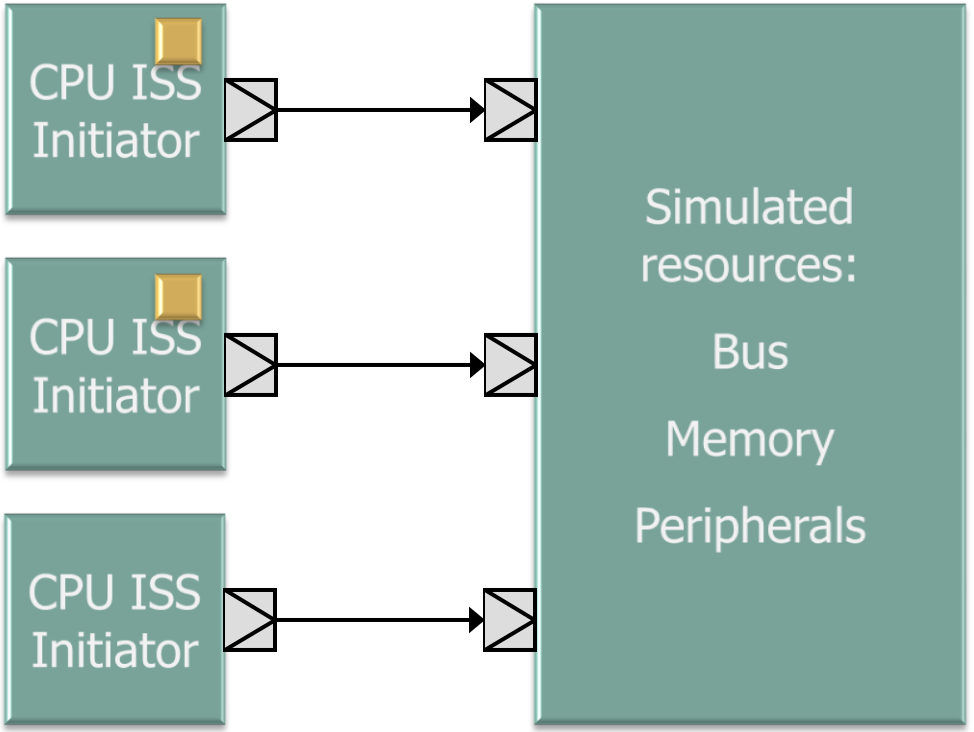
Other initiators do not get to run



Software Execution with Sync

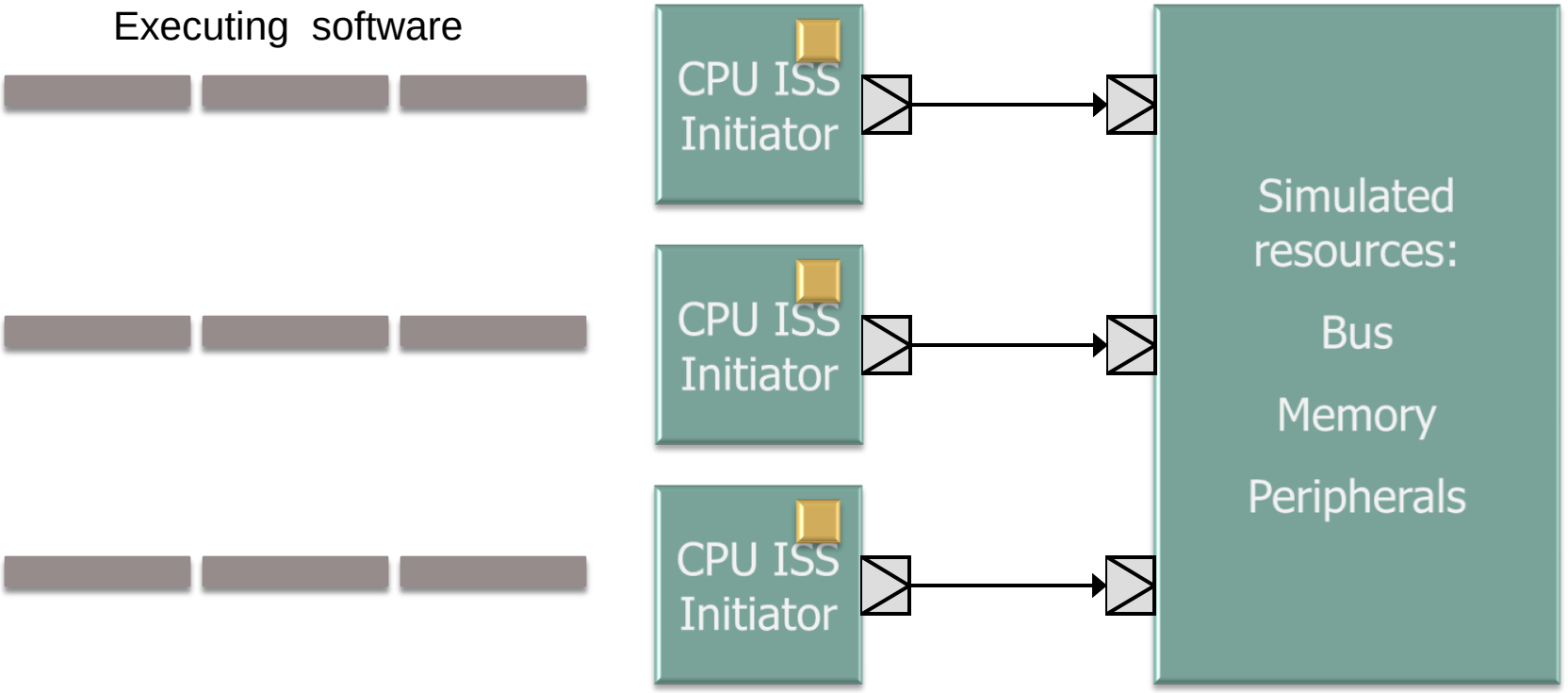


Explicit synchronization between software threads



Synchronization is independent of platform timing

Software Execution with a Quantum

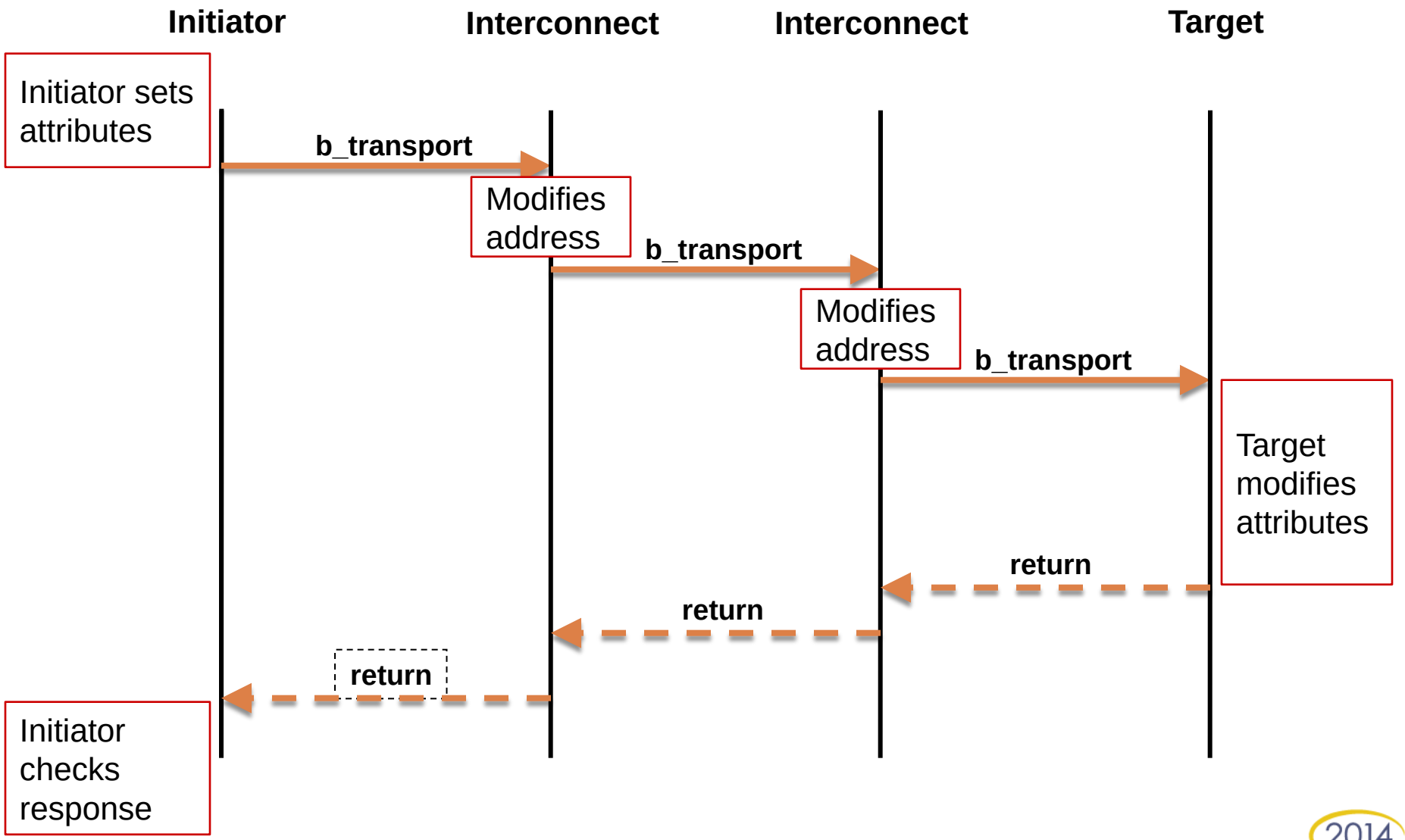


Initiators use a time quantum

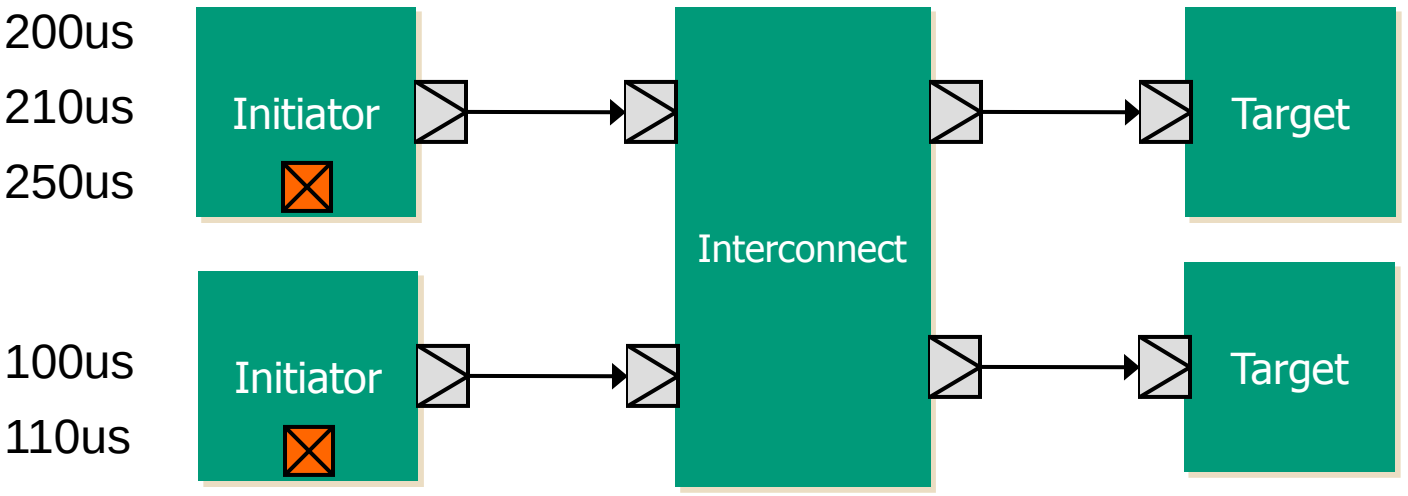
Each initiator gets a time slice

Resources consume simulation time

Causality with b_transport

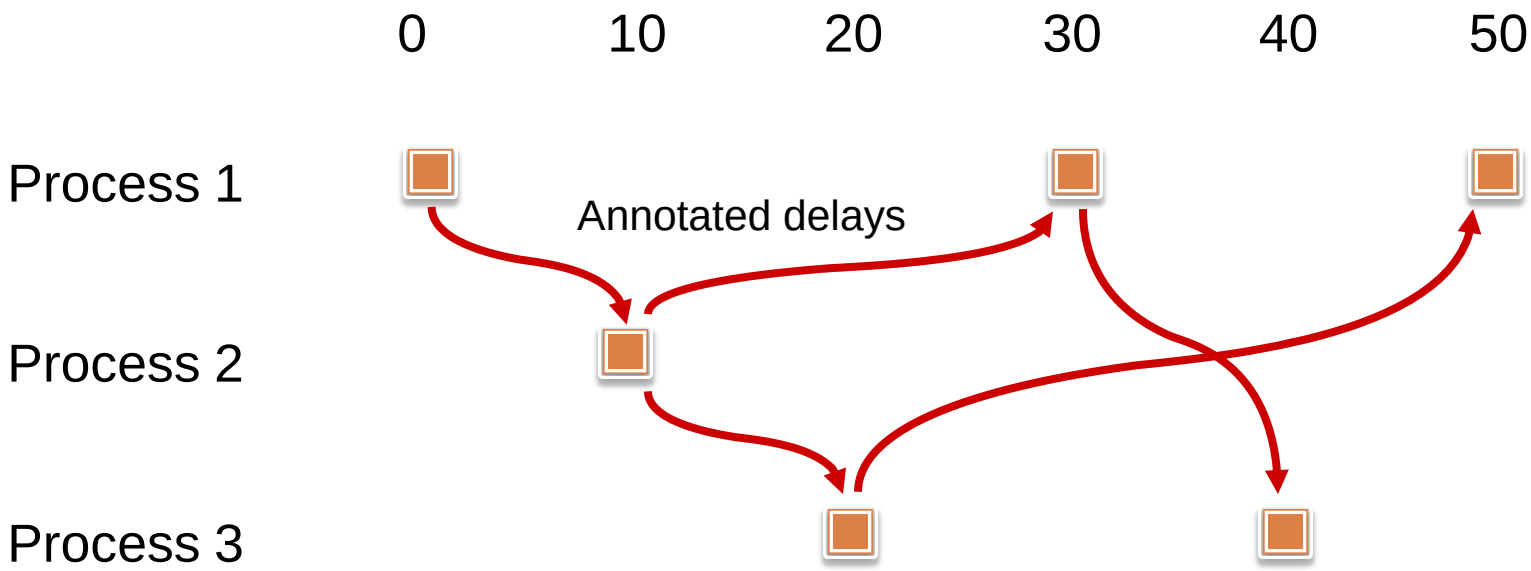


Loosely-Timed Components



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

Approximately-Timed

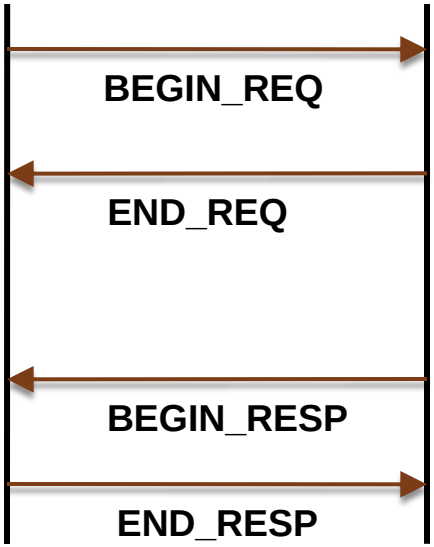


- Inter-process communication is annotated with delays
- Each process is synchronized using the SystemC scheduler
- Delays can be accurate or approximate

AT and CA

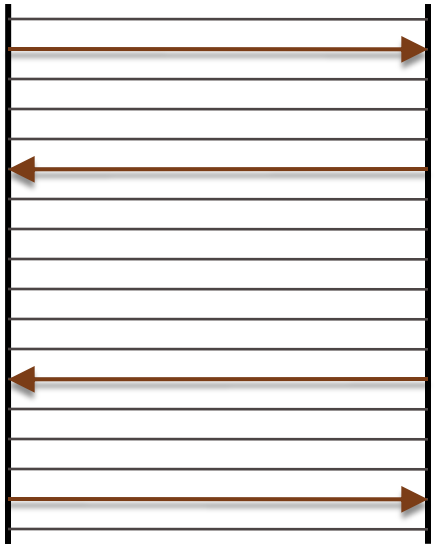
- No running ahead of simulation time; everything stays in sync

AT



Wake up at significant timing points

CA



Wake up every cycle

Virtual Prototyping using SystemC and TLM-2.0

- Introduction
- Development of TLM-2.0
- TLM-2.0 Sockets and Interfaces
- LT, AT, and CA
- Generic Payload and Extensions
- Interoperability

The Generic Payload

- Typical attributes of memory-mapped busses
  - reads, writes, byte enables, single word transfers, burst transfers, streaming
- 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

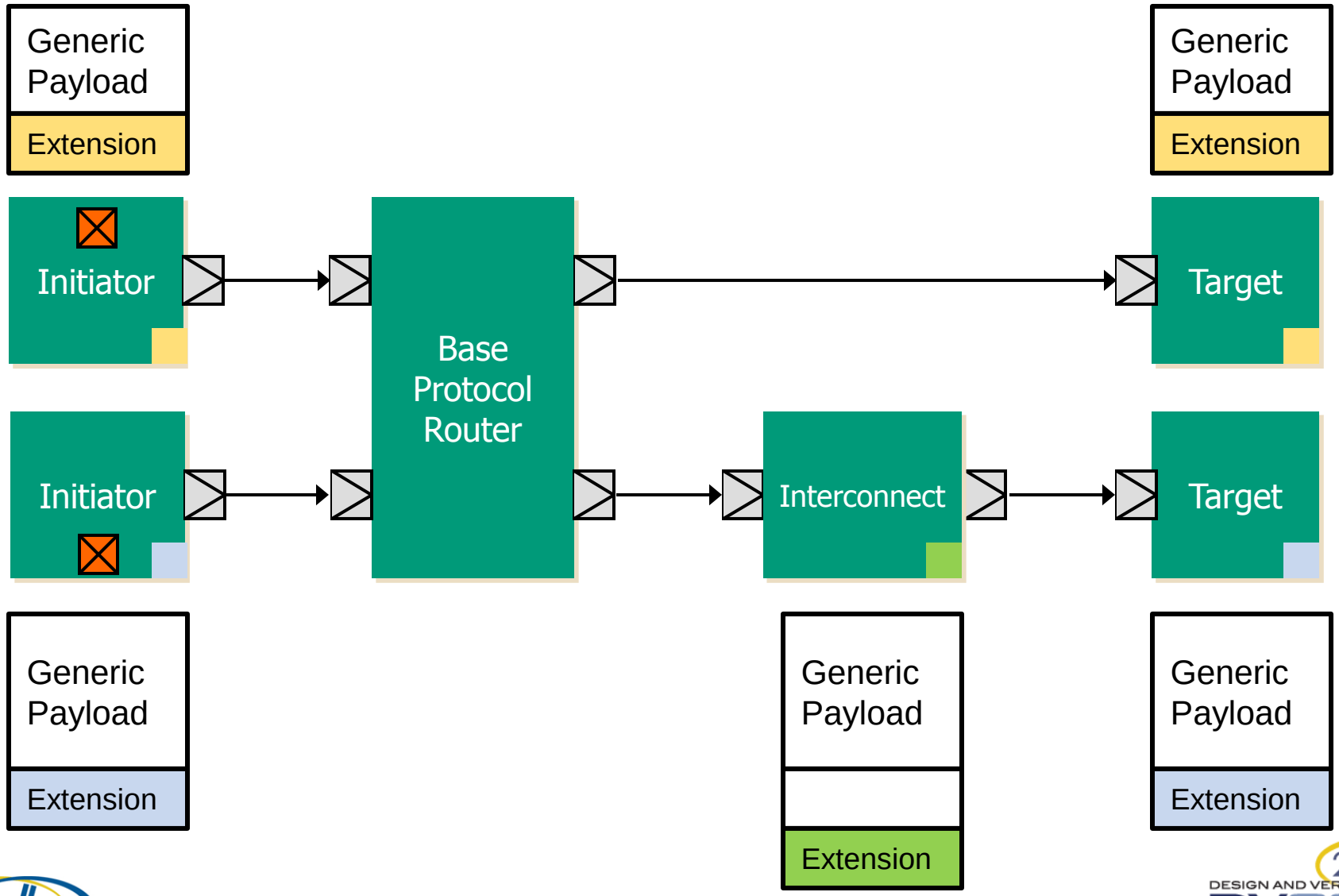
Specific protocols can use the same generic payload machinery

Generic Payload Attributes

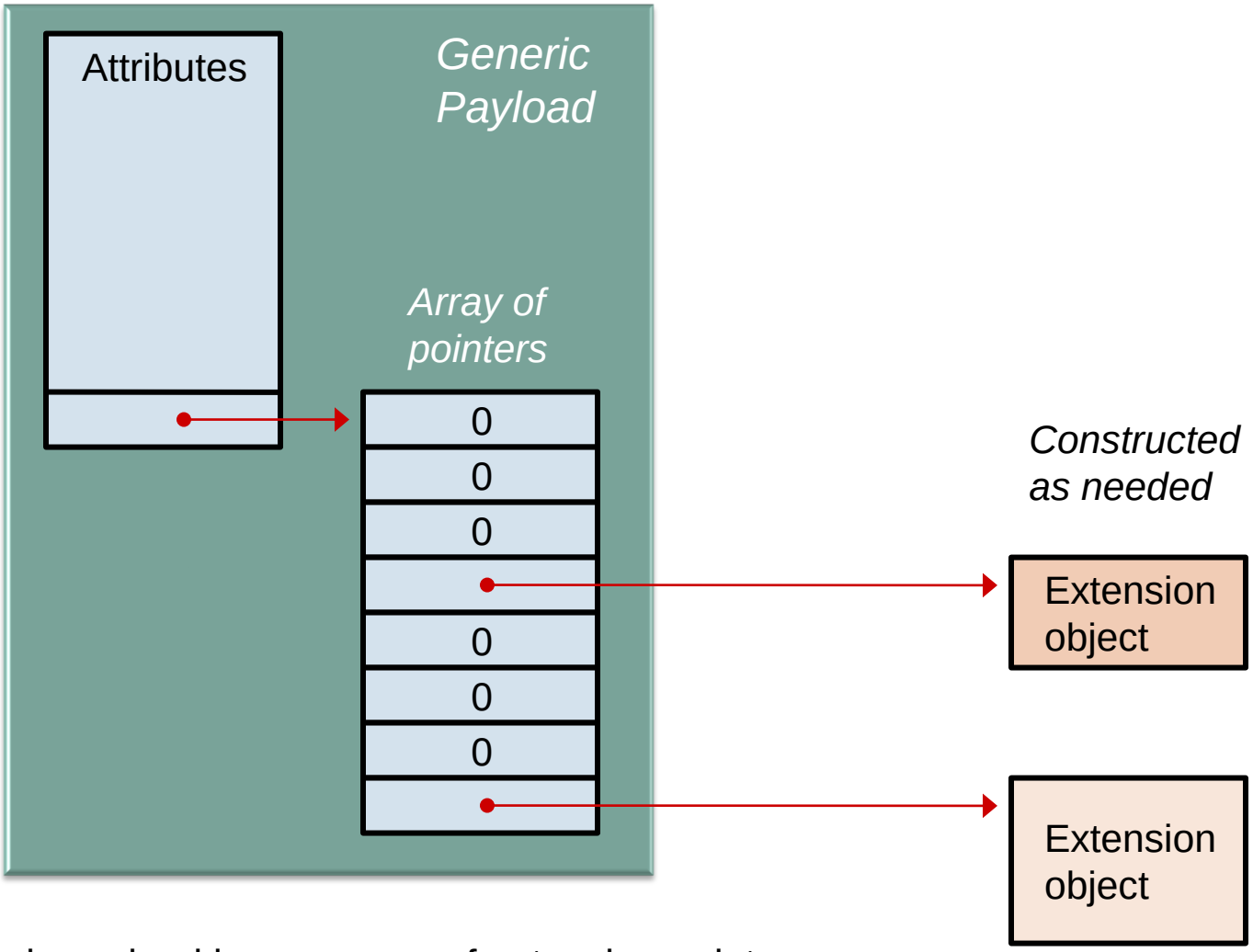
Attribute	Type	Modifiable?	
Command	tlm_command	No	
Address	uint64	Interconnect only	
Data pointer	unsigned char*	No (array – yes)	*Array owned by initiator*
Data length	unsigned int	No	
Byte enable pointer	unsigned char*	No	*Array owned by initiator*
Byte enable length	unsigned int	No	*Ignored if ptr = 0*
Streaming width	unsigned int	No	*Must be > 0*
DMI hint	bool	Yes	*Try DMI !*
Response status	tlm_response_status	Target only	
Extensions	(tlm_extension_base*)[ ]	Yes	

- There are defaults, but transaction objects are typically pooled
- Initiator must set all attributes except byte enable length and extensions

Extensions

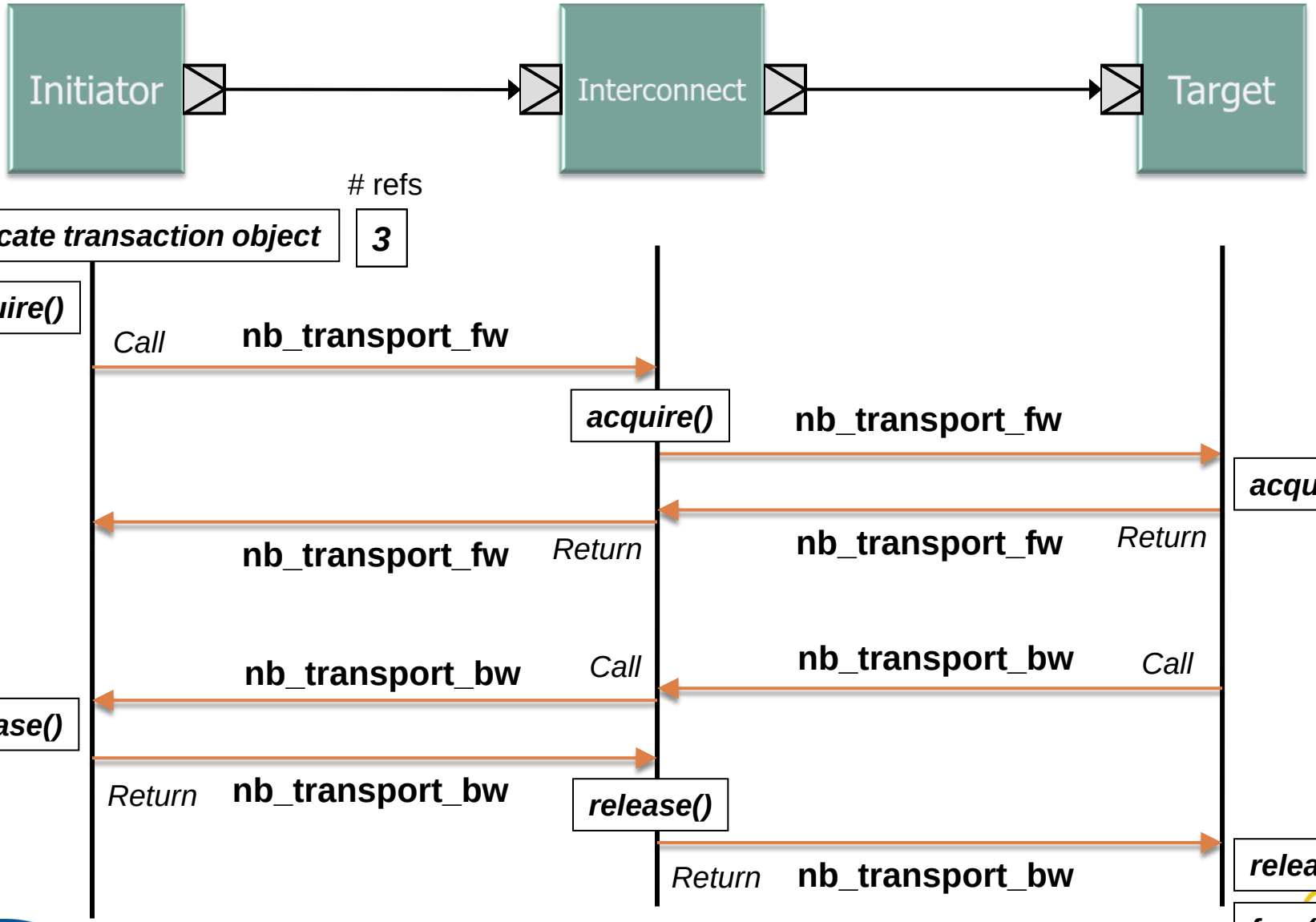


The Extension Mechanism

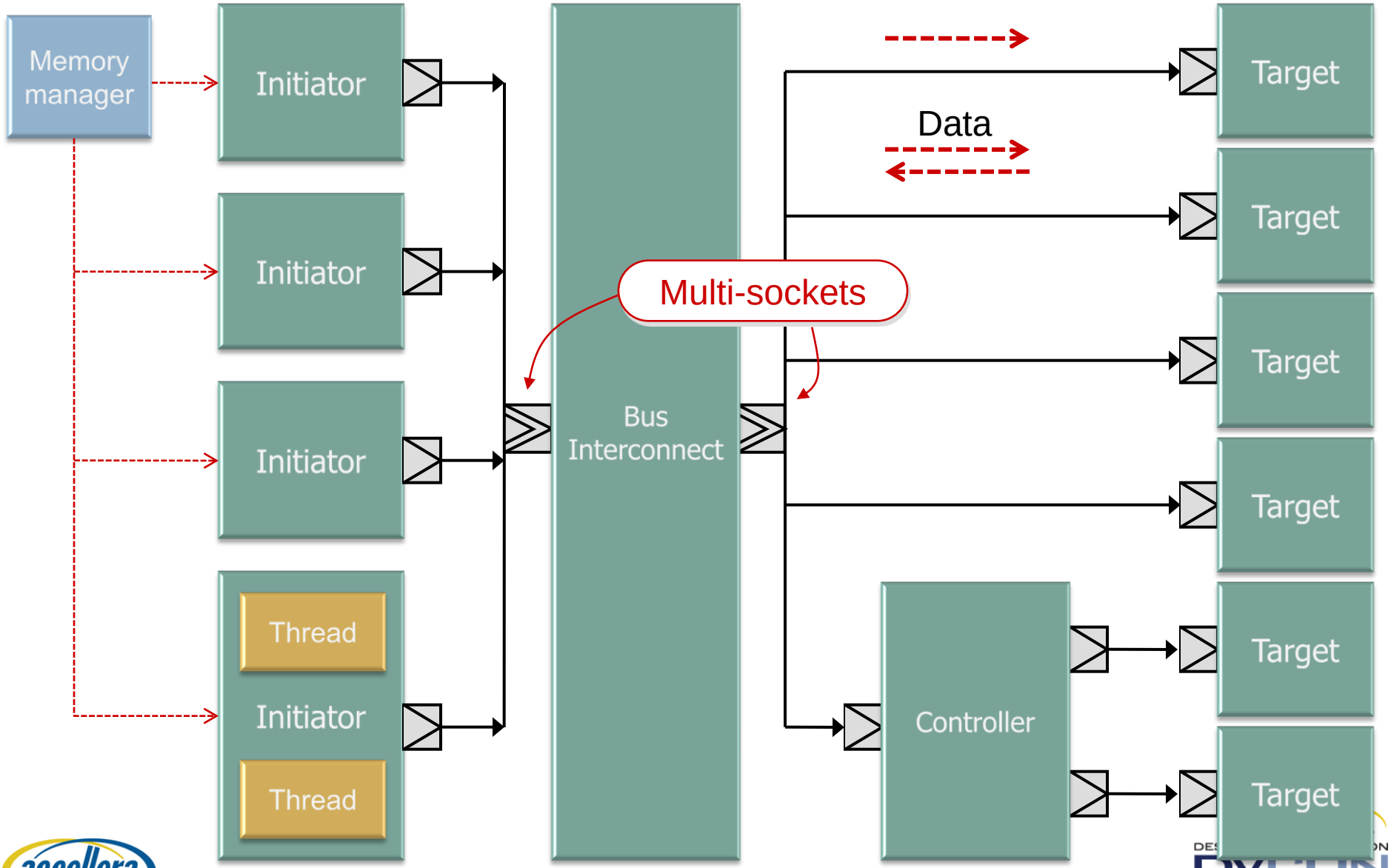


- Every generic payload has an array-of-extension-pointers
- One pointer per extension type, initialized by the constructor

Using a Memory Manager

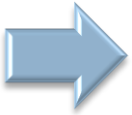


Example Topology



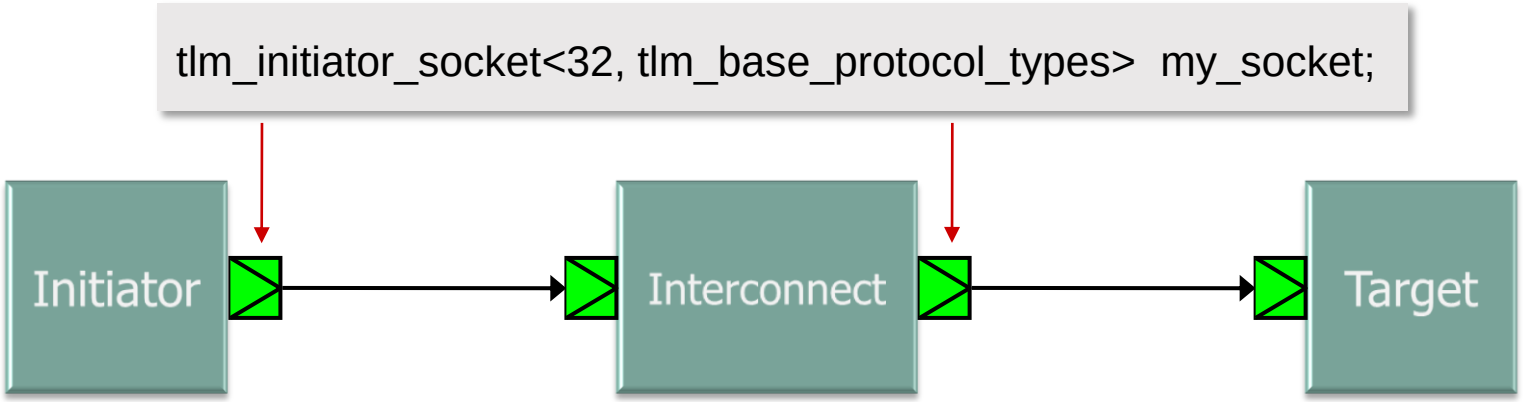
Virtual Prototyping using SystemC and TLM-2.0

- Introduction
- Development of TLM-2.0
- TLM-2.0 Sockets and Interfaces
- LT, AT, and CA
- Generic Payload and Extensions
- Interoperability



First Kind of Interoperability

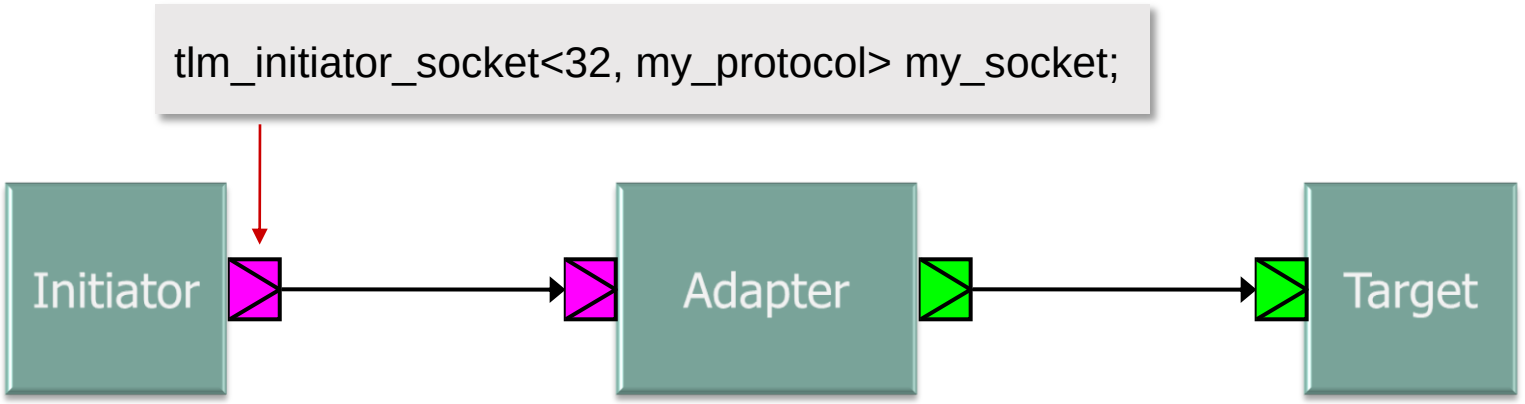
- Use the full interoperability layer
- Use the generic payload + ignorable extensions
- Obey all the rules of the base protocol. The LRM is your rule book



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

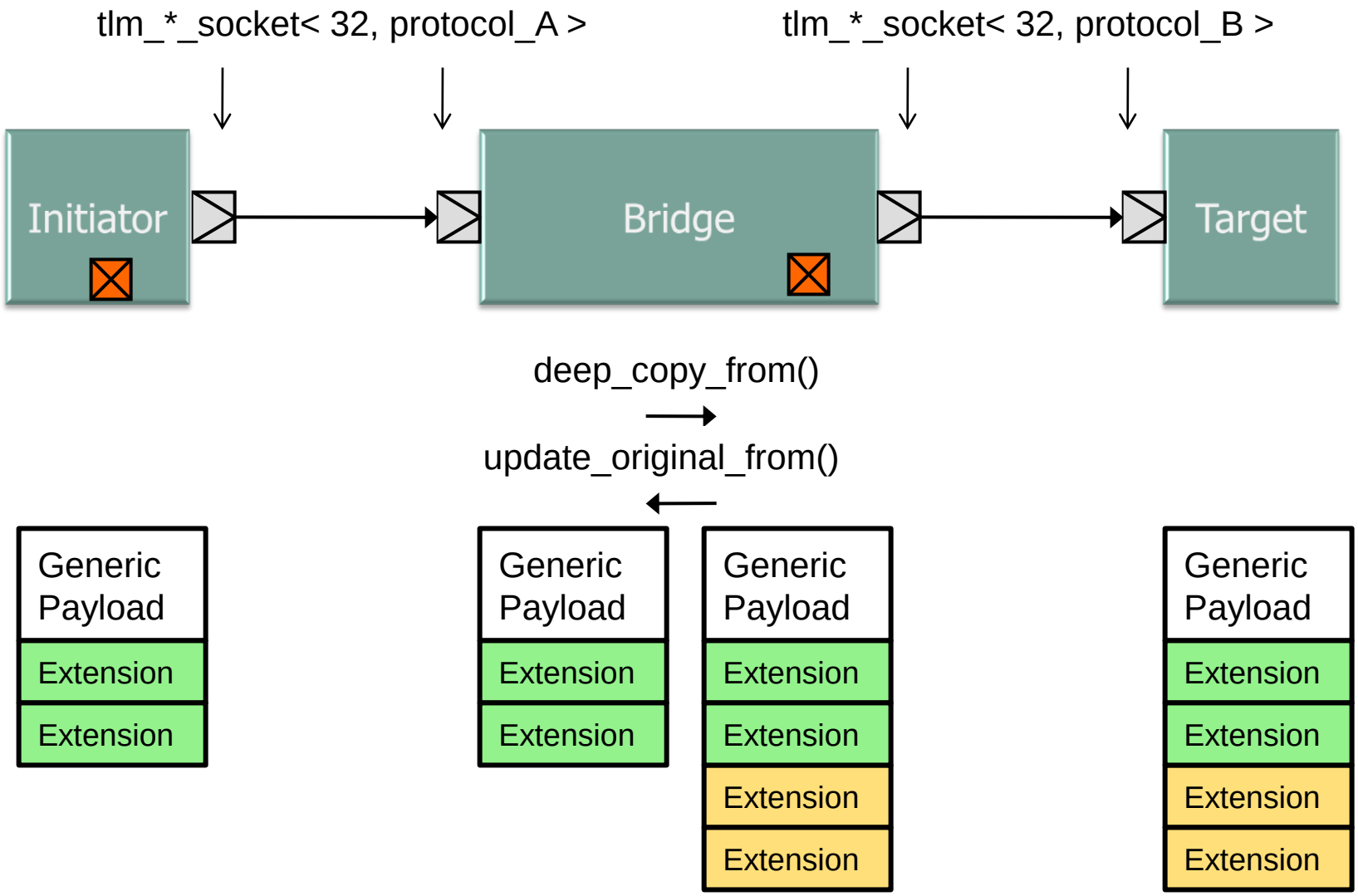
Second Kind of Interoperability

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



- 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

Bridges



Could pass the *same* transaction if their lifetimes allow

Levels of Use

1. Buy models with the TLM-2.0 sticker
2. Write LT components

   Beware: requires more expertise...
3. Write AT components
4. Support LT/AT switching

Missing from TLM-2.0?

- Interrupts
- Other wire-level interfaces
- Register address maps
- Parameterization of TLM IP
- TL power models

Associated Standards and Activities

- Accellera Configuration, Control and Inspection (CCI) Working Group
- Synopsys Virtualizer & SCML, Mentor Vista, Cadence VSP

  ARM FastModels, Carbon, Sonics, Arteris, OVP, ...
- TLM Central www.dr-embedded.com/tlmcentral

Questions?

For Further Information

http://www.doulos.com/knowhow/systemc/tlm2