Modélisation de Soc avec SystemC

francois.pecheux@lip6.fr

Presentation overview

- Context
- Levels of abstraction
  - CABA
  - TLM
- CABA
  - overview
  - SocLib user's view, a sample platform
  - SocLib programmer's view, initiator and ISS
- TLM
  - TLM 2.0, TLM-AT
  - SocLib TLM-DT

Context: LIP6/SoC/ALSoC

- ALSoC : Architecture et logiciel pour SoC

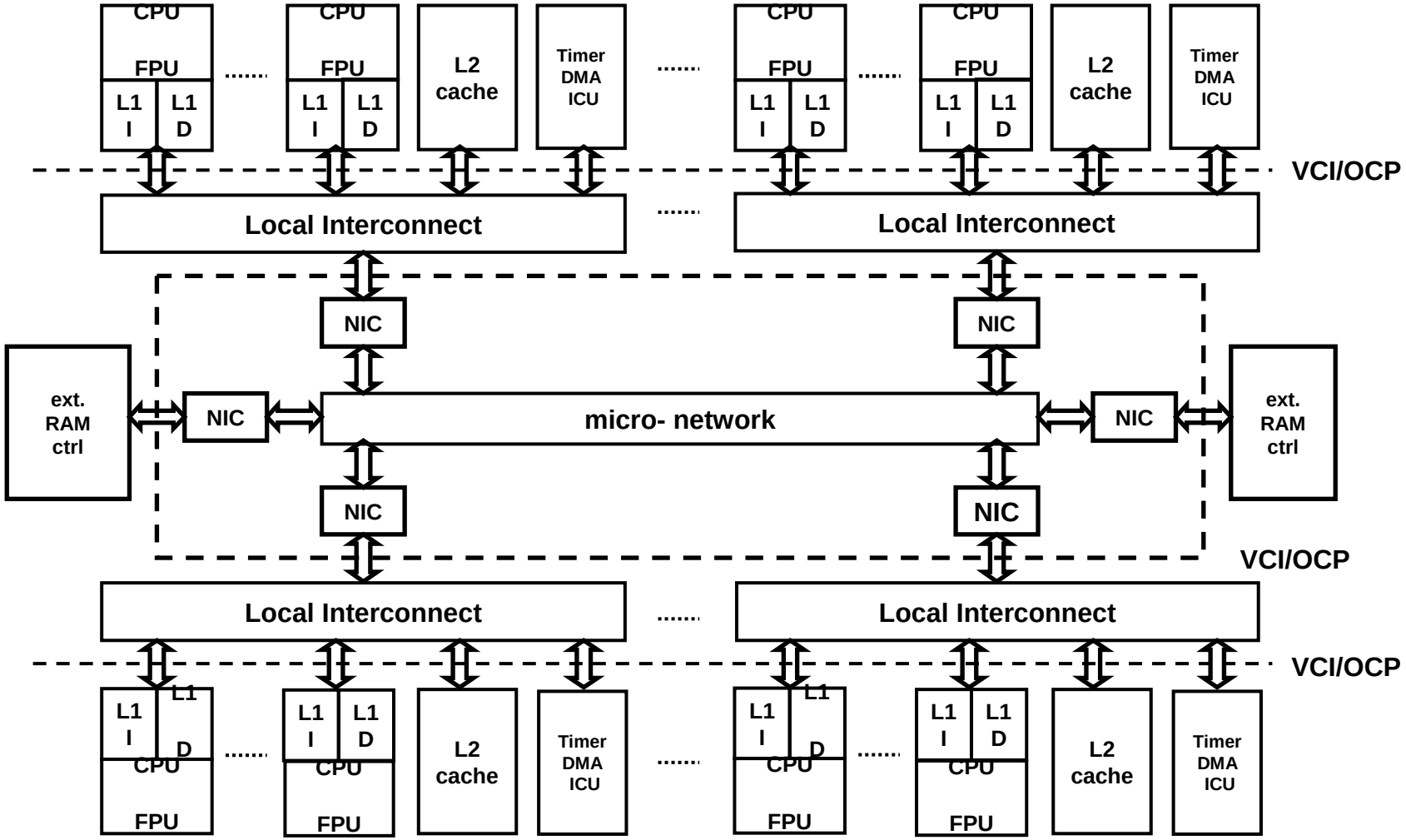
- Développement de modèles de composants des systèmes intégrés, plate-forme de modélisation et de simulation. **Plateforme SoCLib.**
- Développement de systèmes d'exploitation embarqués temps réel (multiprocesseurs hétérogènes).
- Sur ces systèmes/modèles, développement de méthodes/techniques pour :
  - Conception conjointe logiciel-matériel, exploration architecturale.
  - Développement de méthodes de conception et de vérification.
  - Optimisation de codes et parallélisation

MP²SoCs

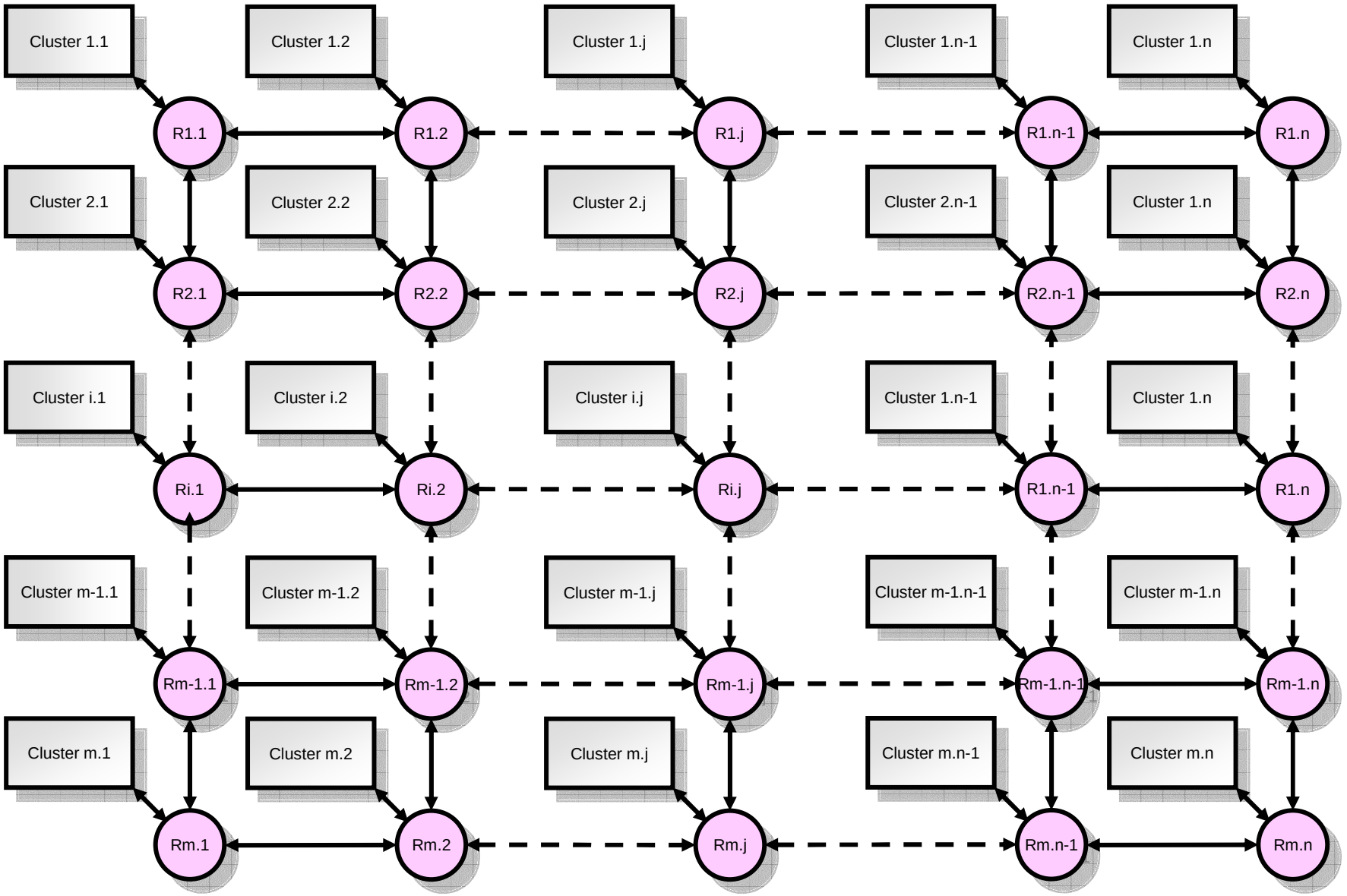
- « Massively Parallel Multi-Processor System on Chip »
- Processeurs scalaires simples mais en grande quantité (> 100 processeurs)
- Mémoire partagée, NUMA, en grande quantité (> 100 bancs mémoire)
- Réseau d'interconnexion (NoC) à deux niveaux (global mesh et local bus/crossbar)

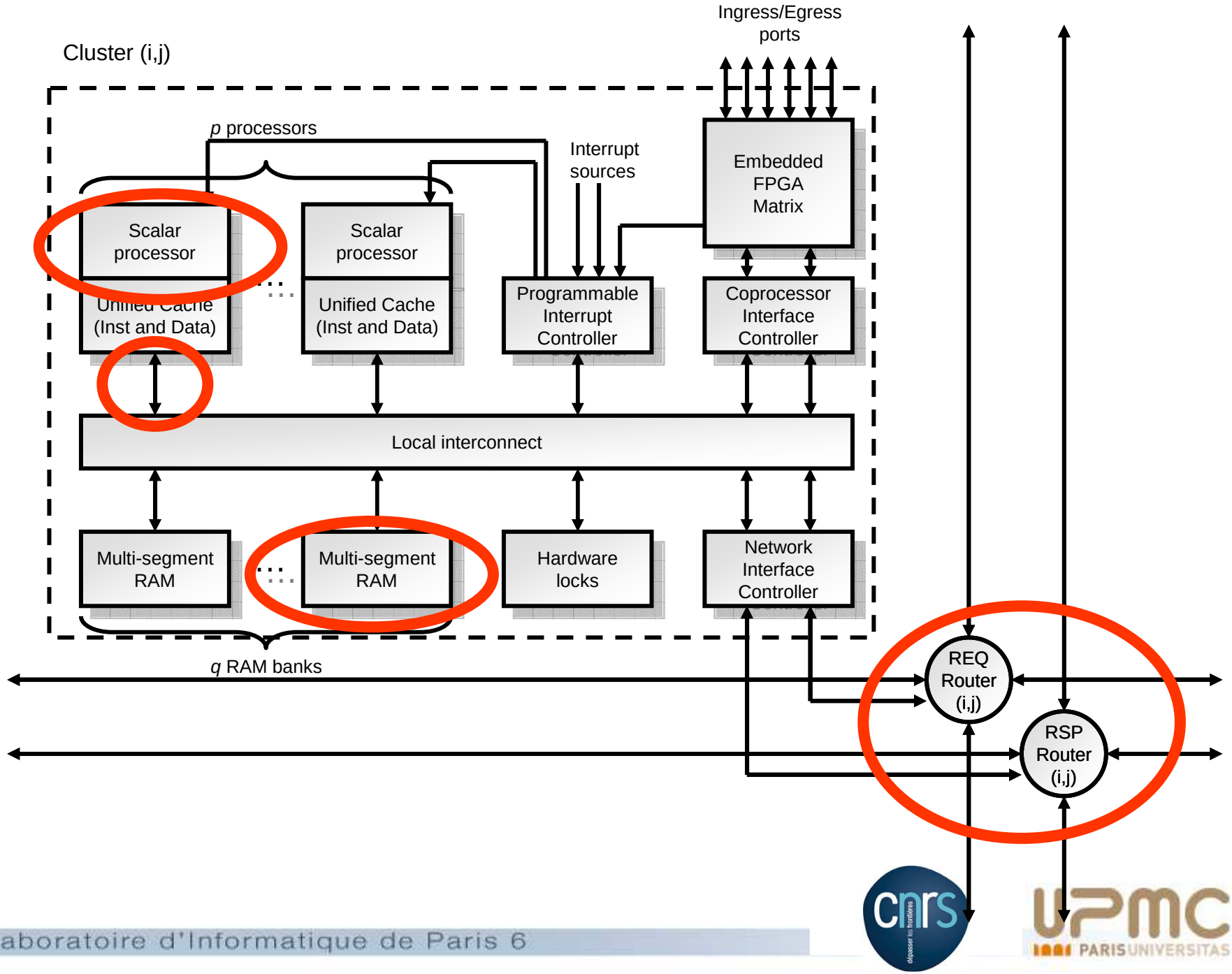
Architecture matérielle visée:

ADAM (MV) TSAR (CC)



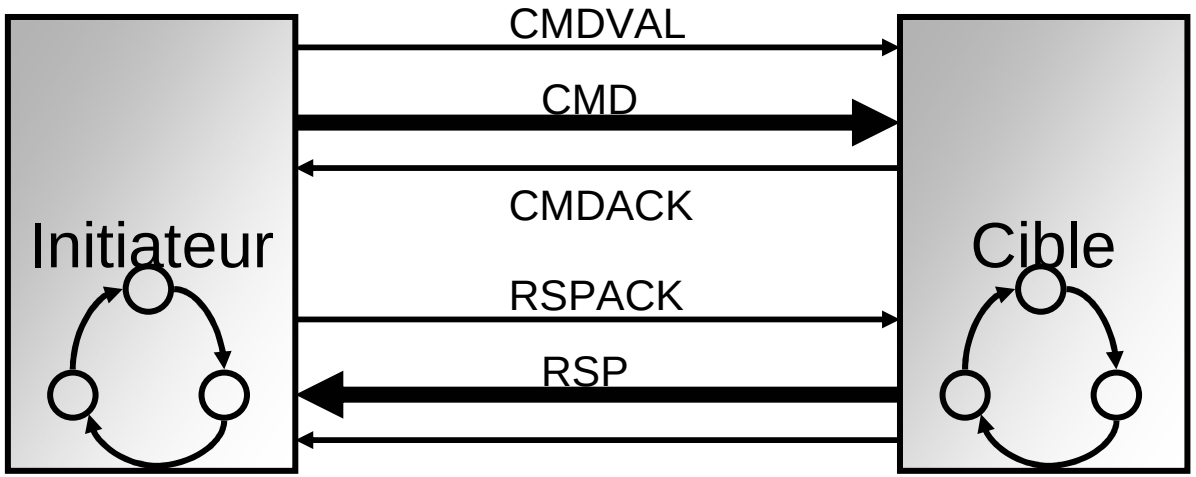
Mesh 2D de clusters




Cluster (i,j)
Ingress/Egress ports
p processors
Interrupt sources
Scalar processor
Scalar processor
Unified Cache (Inst and Data)
Unified Cache (Inst and Data)
Embedded FPGA Matrix
Programmable Interrupt Controller
Coprocessor Interface Controller
Local interconnect
Multi-segment RAM
Multi-segment RAM
Hardware locks
Network Interface Controller
q RAM banks
REQ Router (i,j)
RSP Router (i,j)

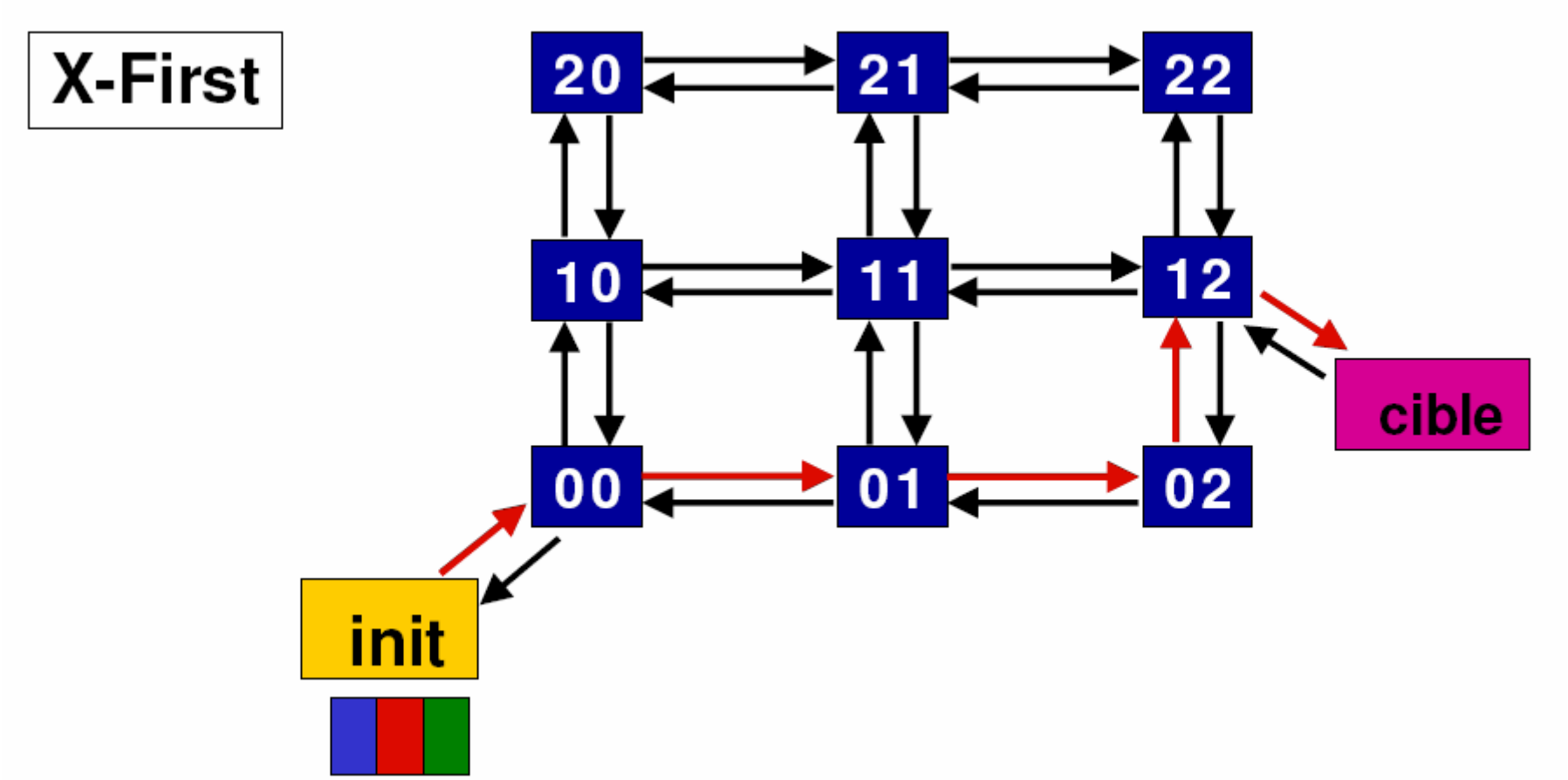
VCI/OCP

- Composants écrits en SystemC OSCI
- Interopérables (standard VCI/OCP)
- 3 niveaux de modélisation dont 2 disponibles
  - TLM avec temps, phénomènes dynamiques (comportement fin des caches, gestion de la contention)
  - Cycle Accurate Bit Accurate
  - RTL synthétisable

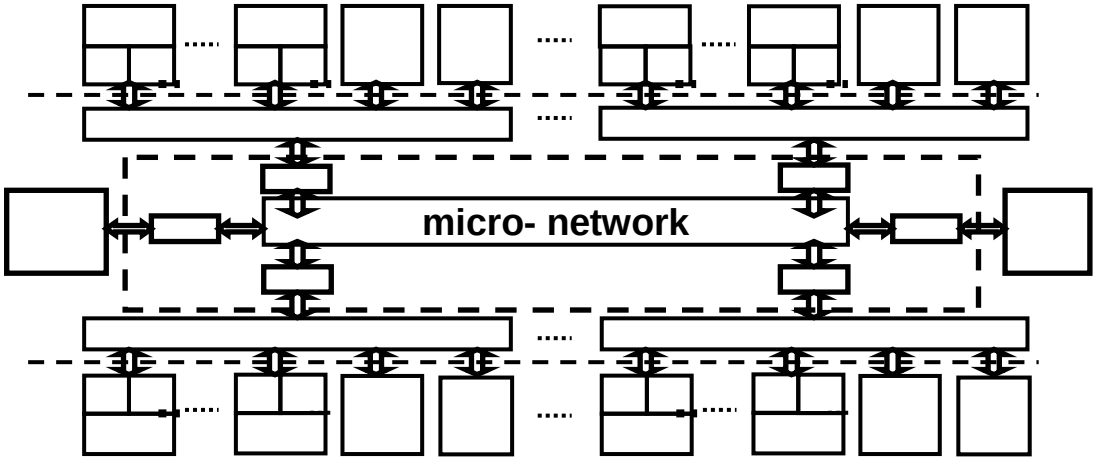


Network on chip

GALS et VFI

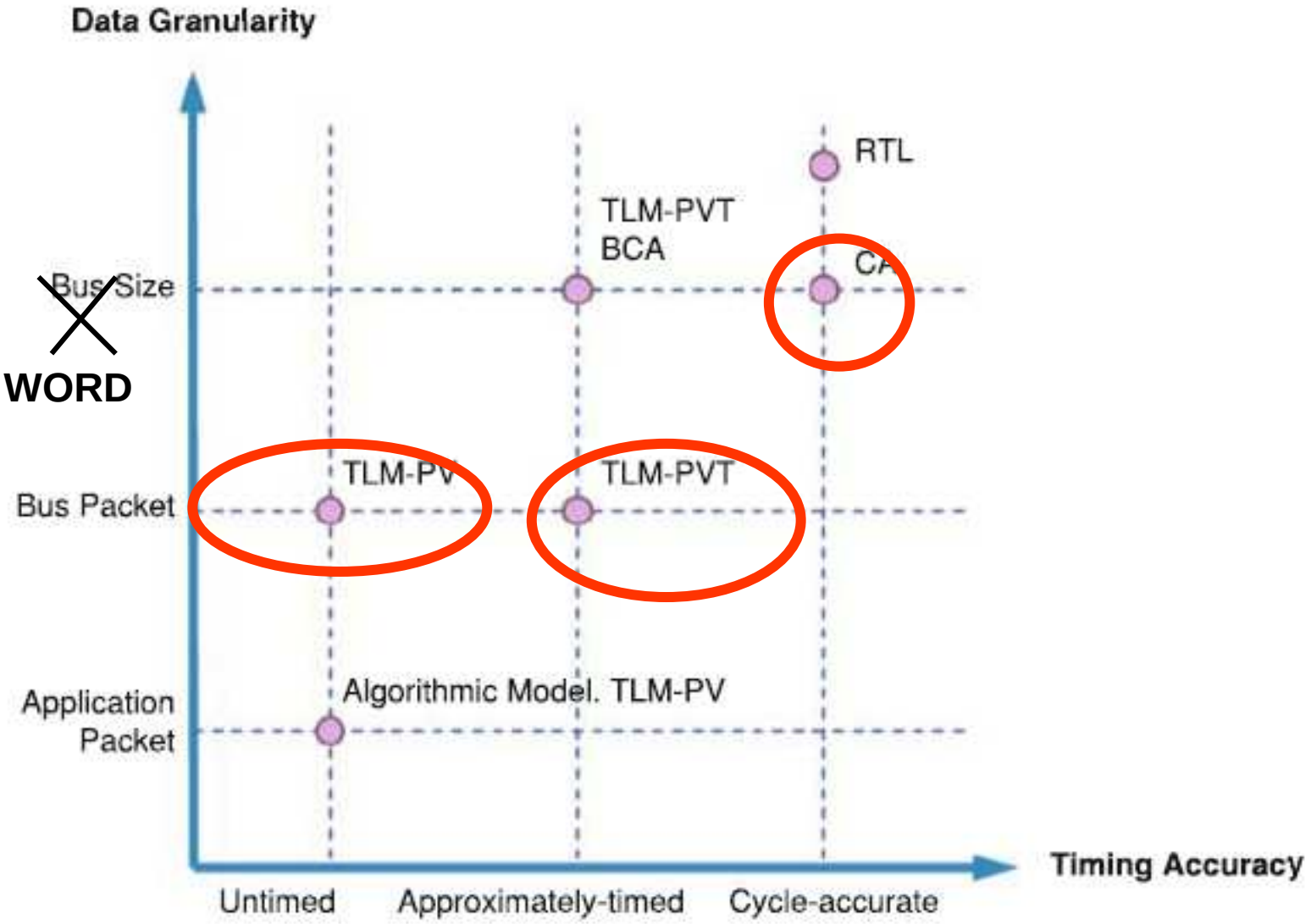


Plateforme Technologique : SoCLib

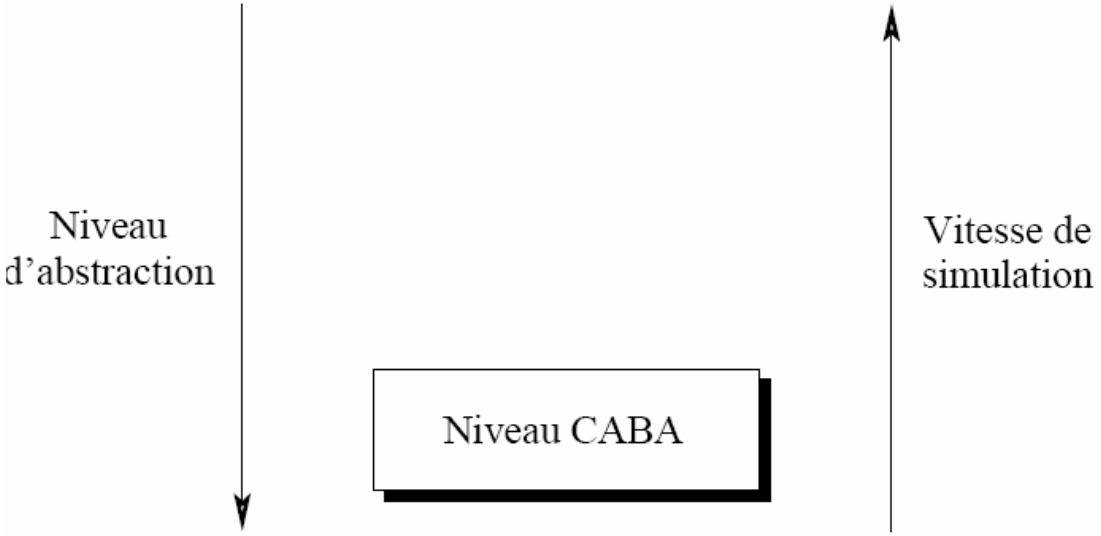


• Le projet ANR SOCLIB vise la constitution d'une plate-forme de prototypage virtuel de systèmes multi-processeurs intégrés sur puce.

• 11 laboratoires et 6 industriels (dont ST Microelectronics, Thales, et Thomson) participent au projet, qui est soutenu par trois pôles de compétitivité (System@TIC, Minalogic, Images/réseaux)

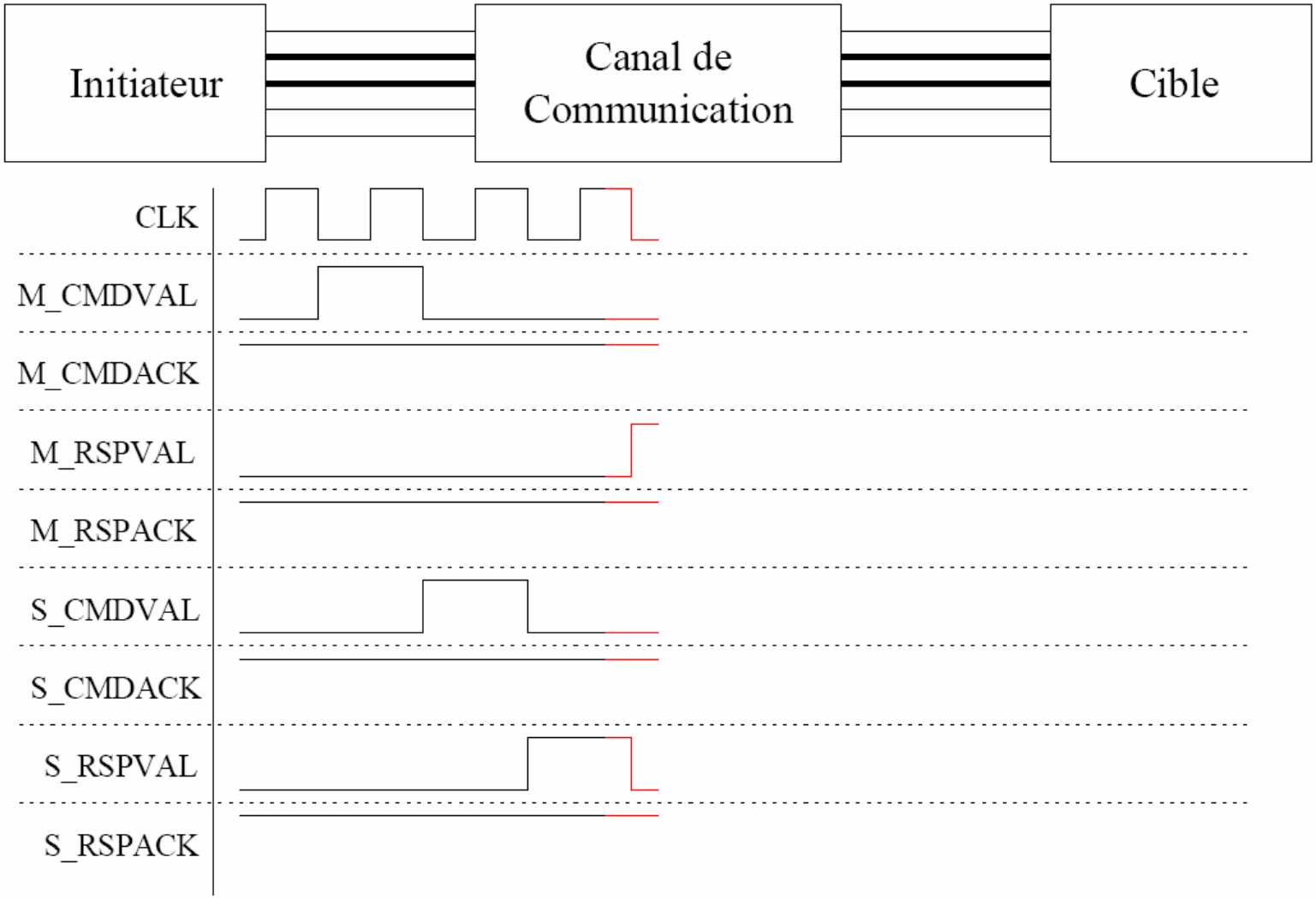

Data Granularity
RTL
TLM-PVT
BCA
CA
Bus Size
WORD
TLM-PV
TLM-PVT
Bus Packet
Algorithmic Model. TLM-PV
Application
Packet
Timing Accuracy
Untimed
Approximately-timed
Cycle-accurate

Cycle-Accurate Bit-Accurate

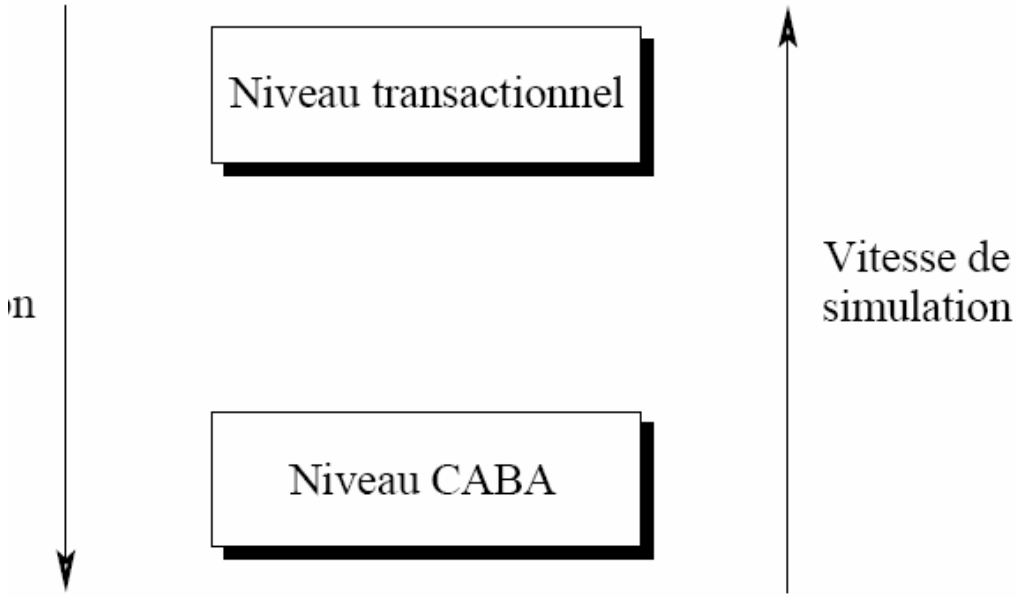


- simulation fine et fidèle par rapport au matériel ;
- mais simulation lente.

CABA waveform

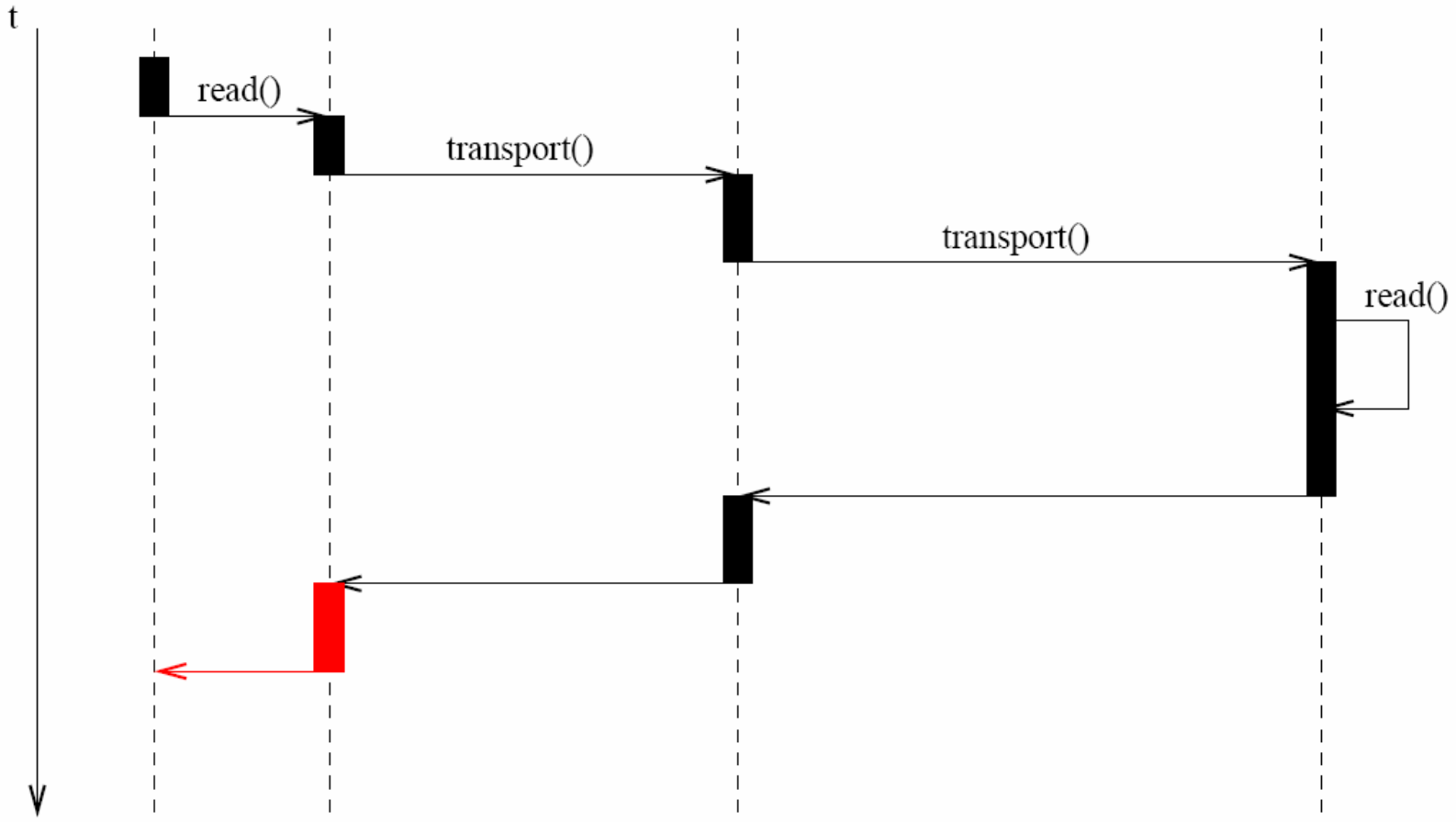


Transaction Level Modeling (TLM)



- validation fonctionnelle de l'application ;
- simulation très rapide mais pas de validation des performances et de l'adéquation de l'architecture matérielle.

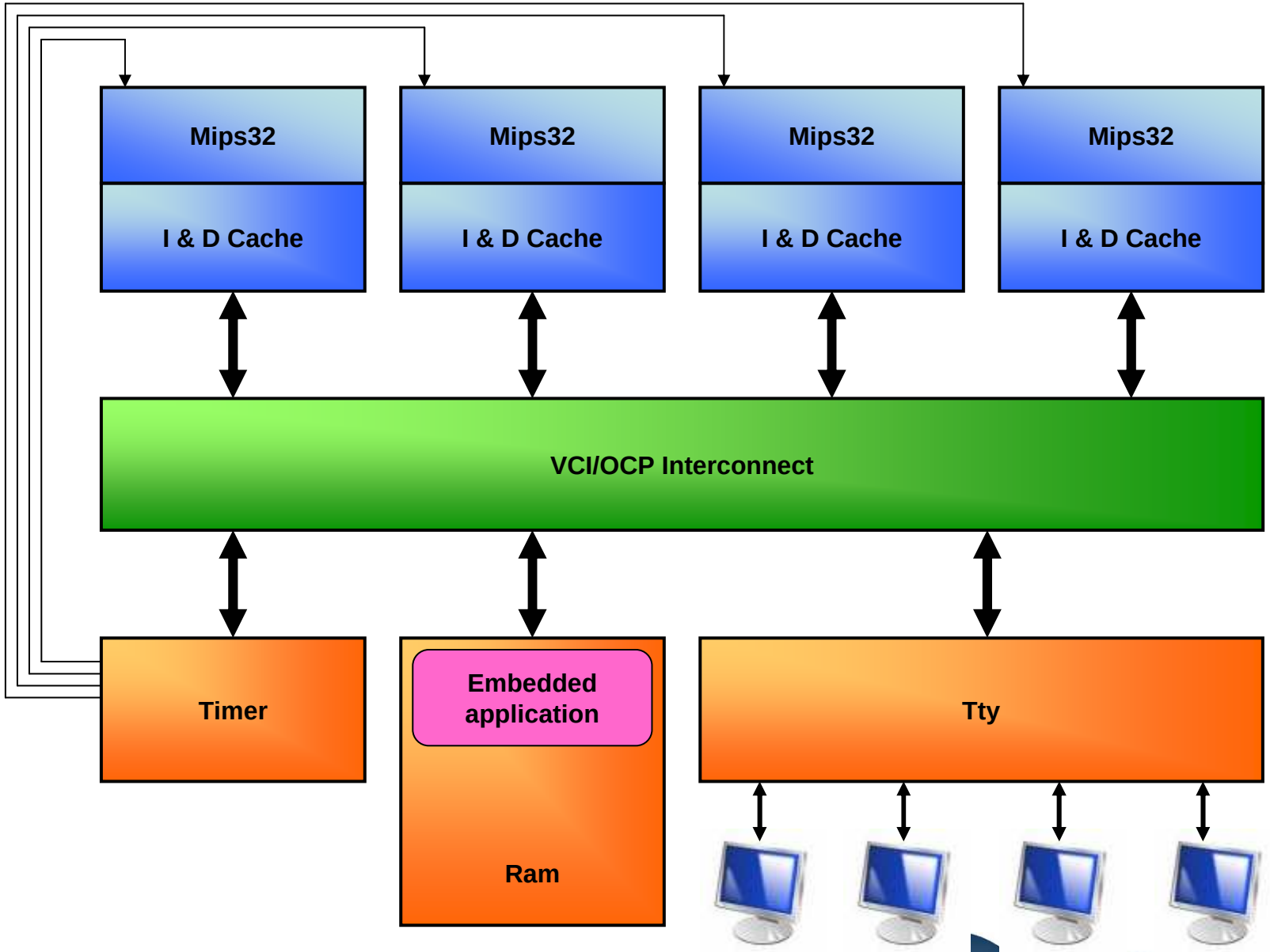
TLM Interface Method Calls

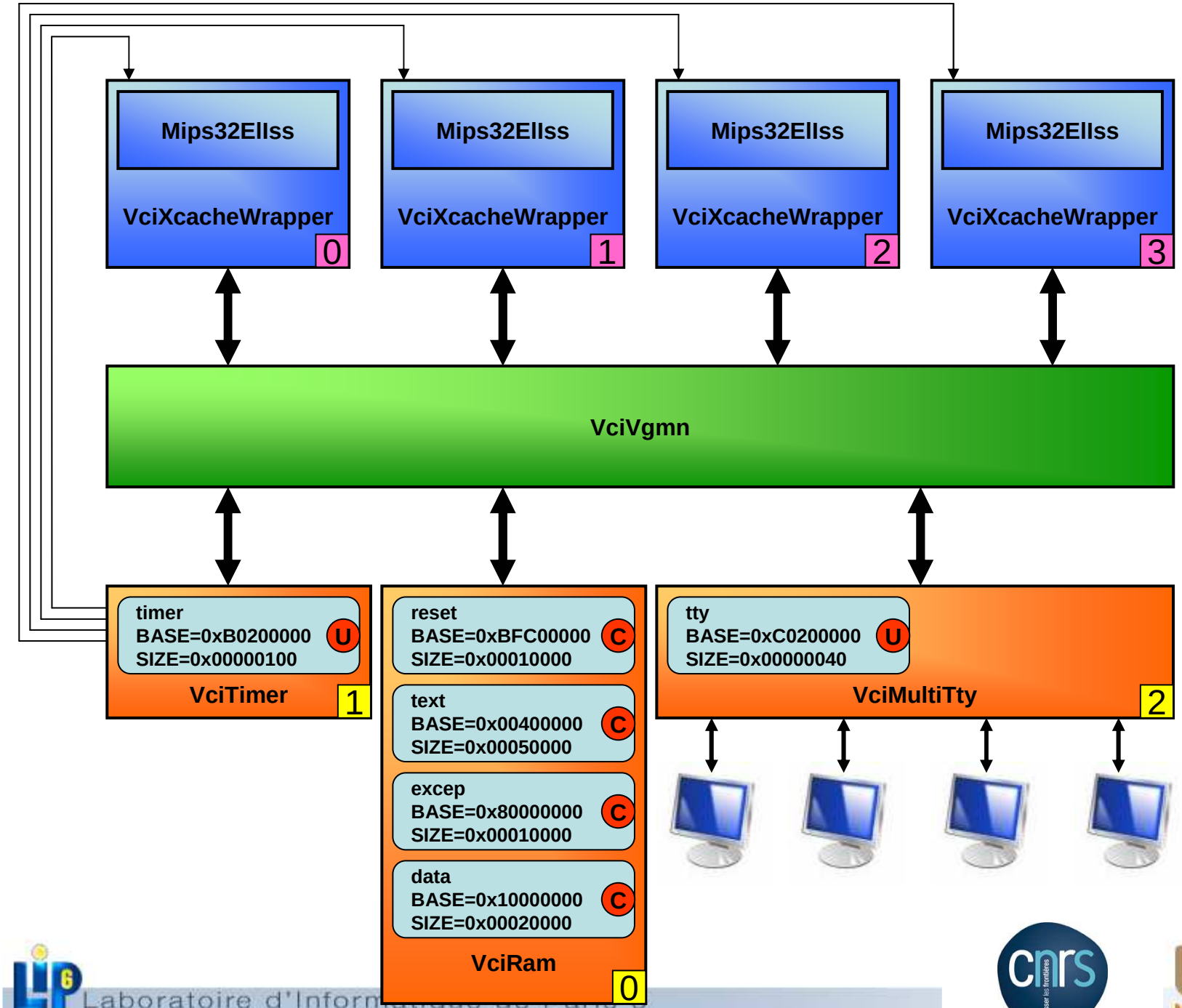


SocLib Tutorial #1

caba-vgmn-multi_timer-mips32

- How to design a « cycle-accurate bit-accurate » four processor platform
- The VCI/OCP protocol
- How to instanciate SocLib initiator components
- How to instanciate SocLib target components
- How to write and to run an embedded application on this platform


Mips32
Mips32
Mips32
Mips32
I & D Cache
I & D Cache
I & D Cache
I & D Cache
VCI/OCP Interconnect
Timer
Embedded application
Ram
Tty


Mips32Ellss
VciXcacheWrapper
0
Mips32Ellss
VciXcacheWrapper
1
Mips32Ellss
VciXcacheWrapper
2
Mips32Ellss
VciXcacheWrapper
3
VciVgmn
timer
BASE=0xB0200000
SIZE=0x00000100
U
VciTimer
1
reset
BASE=0xBFC00000
SIZE=0x00010000
C
text
BASE=0x00400000
SIZE=0x00050000
C
excep
BASE=0x80000000
SIZE=0x00010000
C
data
BASE=0x10000000
SIZE=0x00020000
C
VciRam
0
tty
BASE=0xC0200000
SIZE=0x00000040
U
VciMultiTty
2

Scalar processor+cache

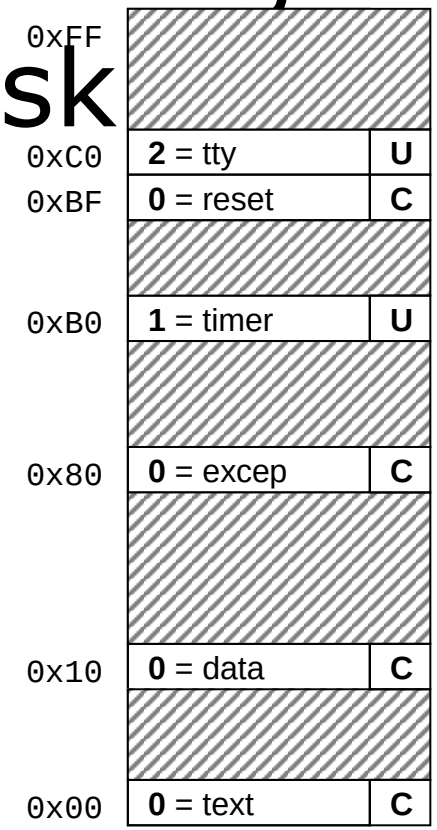
- RISC, 5-stage pipeline ARM7, PowerPC 405, MIPSR32
- Direct Mapped Write-Through
- GNU GCC toolchain
- Simulated via an ISS (Instruction Set Simulator), cache effects, delayed slots

Segments, address decoding and cacheability mask

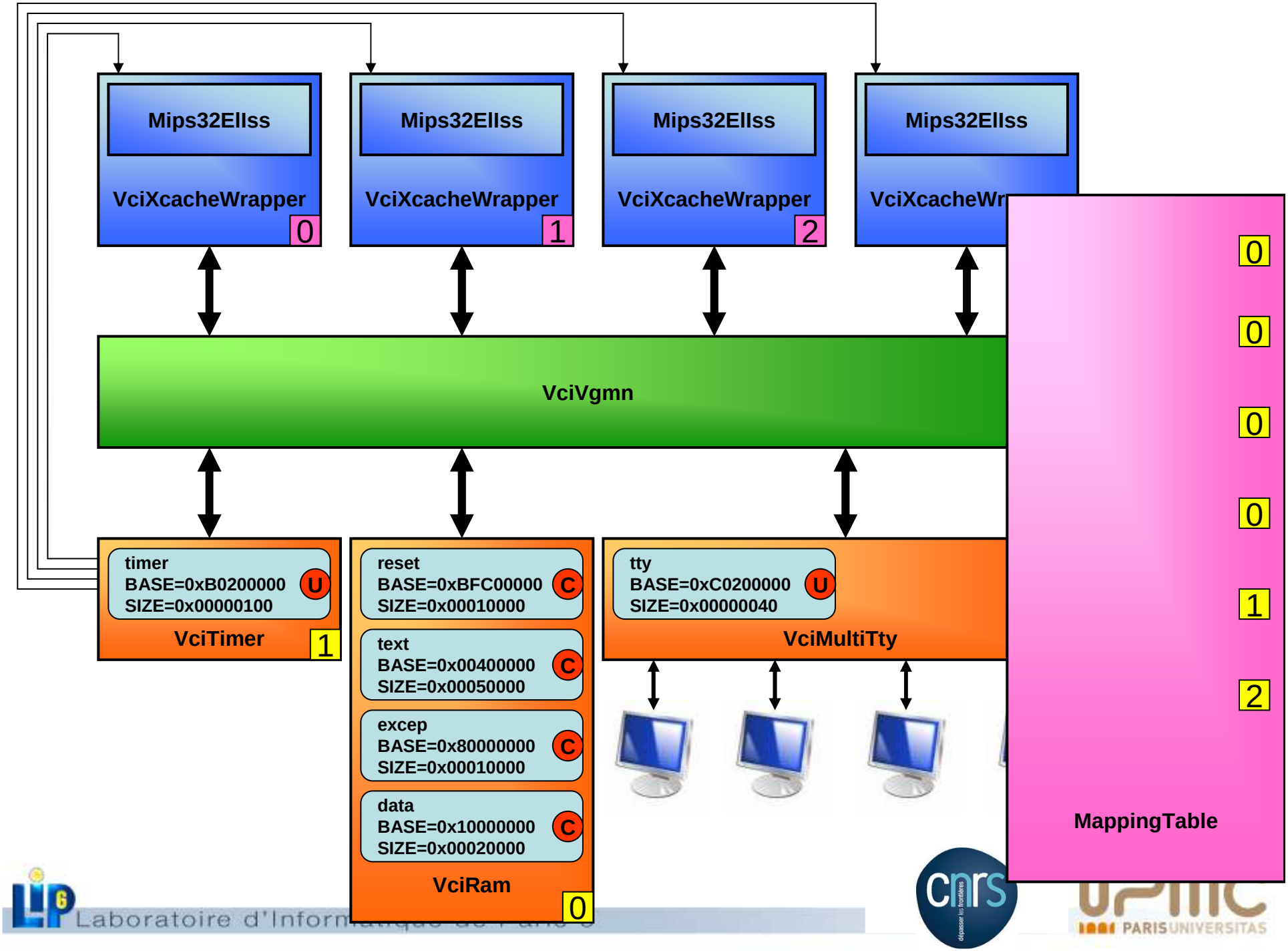
```
reset 0xBFC00000  = 1011 1111 1100 0000 0000 0000 0000 0000
text  0x00400000  = 0000 0000 0100 0000 0000 0000 0000 0000
excep 0x80000000  = 1000 0000 0000 0000 0000 0000 0000 0000
data  0x10000000  = 0001 0000 0000 0000 0000 0000 0000 0000
timer 0xB0200000  = 1011 0000 0010 0000 0000 0000 0000 0000
tty   0xC0200000  = 1100 0000 0010 0000 0000 0000 0000 0000
mask  0x00300000  = 0000 0000 0011 0000 0000 0000 0000 0000
```

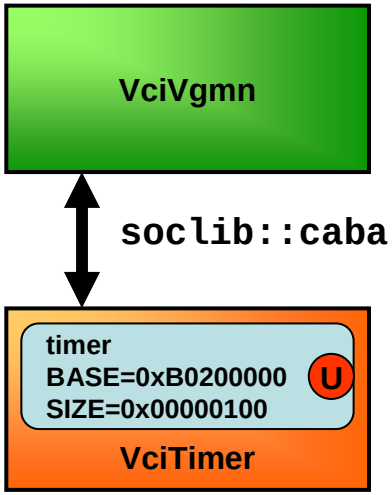
8 bits for target decoding

2 bits for cacheability



Platform address space = **Mapping table**


Mips32Ellss
VciXcacheWrapper
0
Mips32Ellss
VciXcacheWrapper
1
Mips32Ellss
VciXcacheWrapper
2
Mips32Ellss
VciXcacheWr
VciVgmn
timer
BASE=0xB0200000
SIZE=0x00000100
U
VciTimer
1
reset
BASE=0xBFC00000
SIZE=0x00010000
C
text
BASE=0x00400000
SIZE=0x00050000
C
excep
BASE=0x80000000
SIZE=0x00010000
C
data
BASE=0x10000000
SIZE=0x00020000
C
VciRam
0
tty
BASE=0xC0200000
SIZE=0x00000040
U
VciMultiTty
0
0
0
0
1
2
MappingTable




```
soclib::caba::VciSignals<vci_param> signal_vci_vcitimer("signal_vci_vcitimer");
```

```
typedef soclib::caba::VciParams<4,6,32,1,1,1,8,1,1,1> vci_param;
```

cell_size = 4 * 8 = 32 bits

plen_size = 64 words

addr_size = 32 bits

rerror_size = 1 bit

clen_size = 1 bit

rflag_size = 1 bit

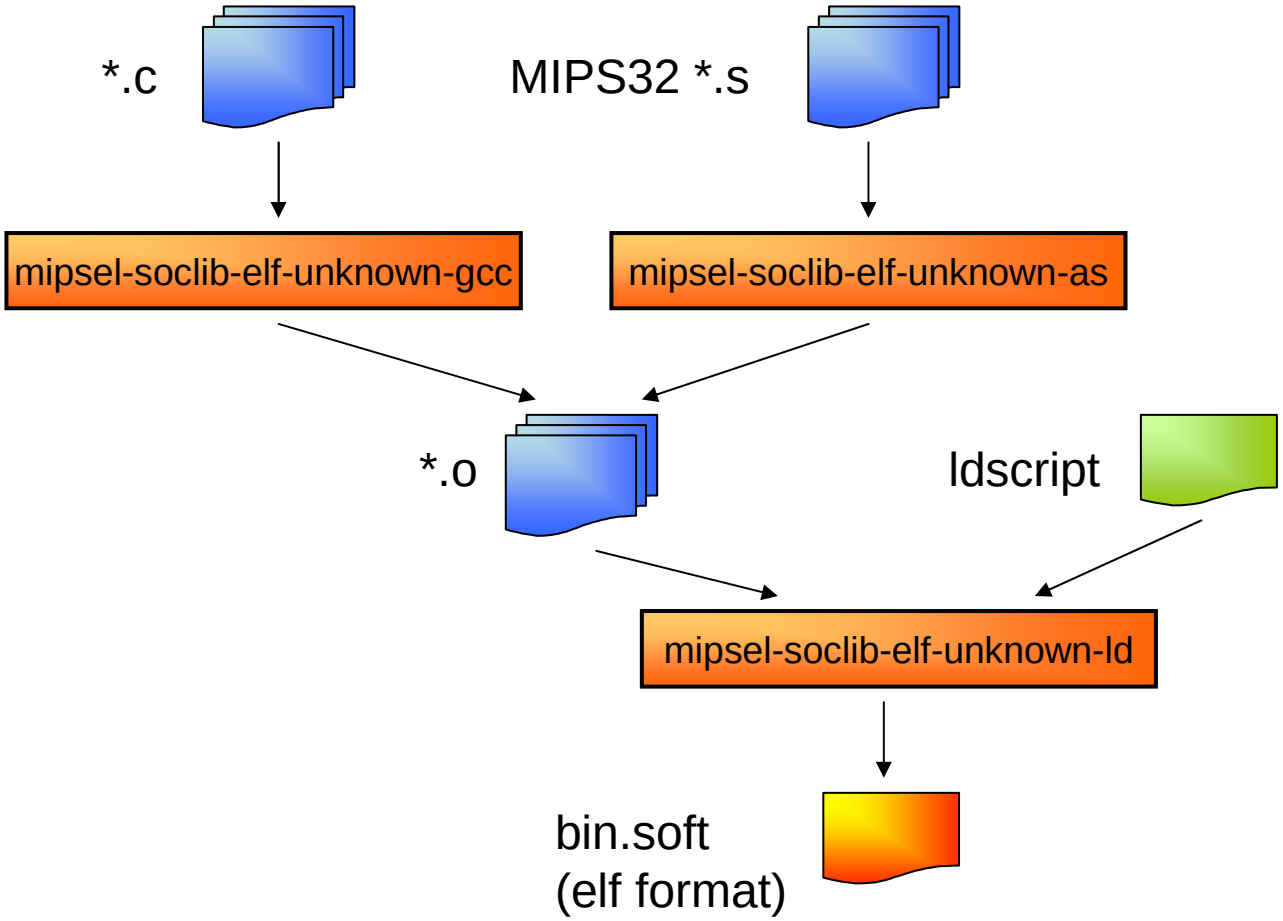
srcid_size = 8 bits

pktid_size = 1 bit

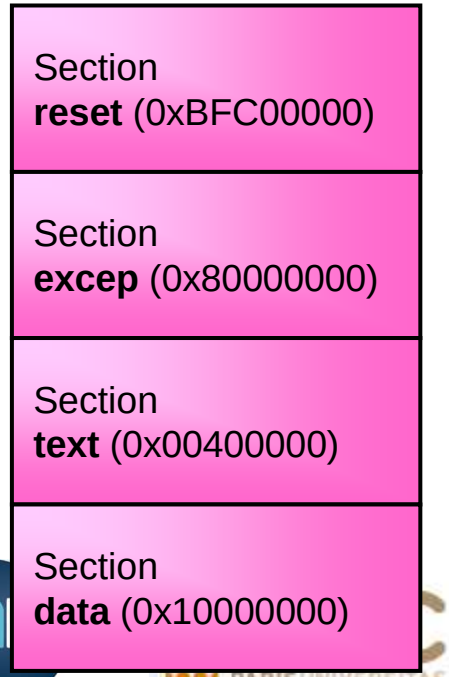
trdid_size = 1 bit

wrplen_size = 1 bit

Building the embedded application

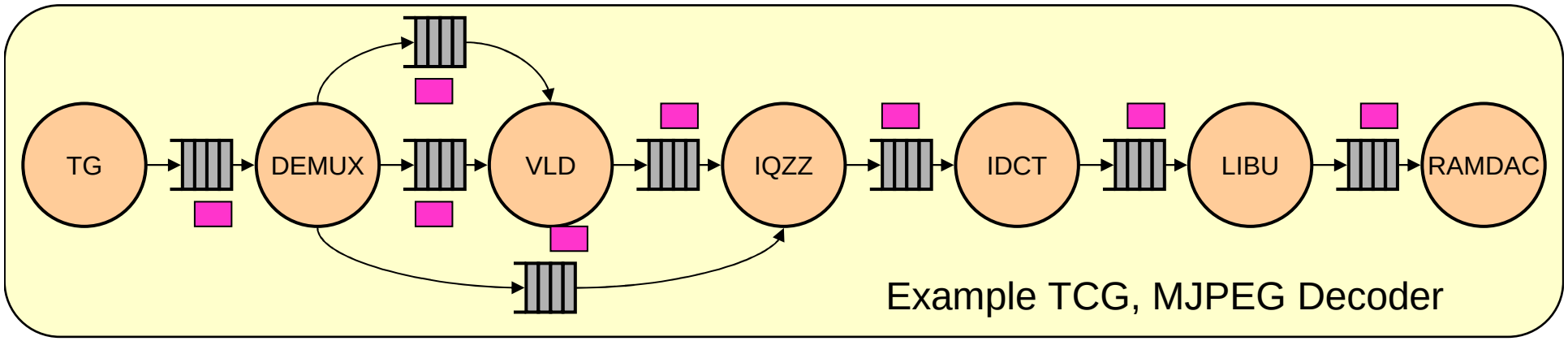


Application binary composed of **sections**



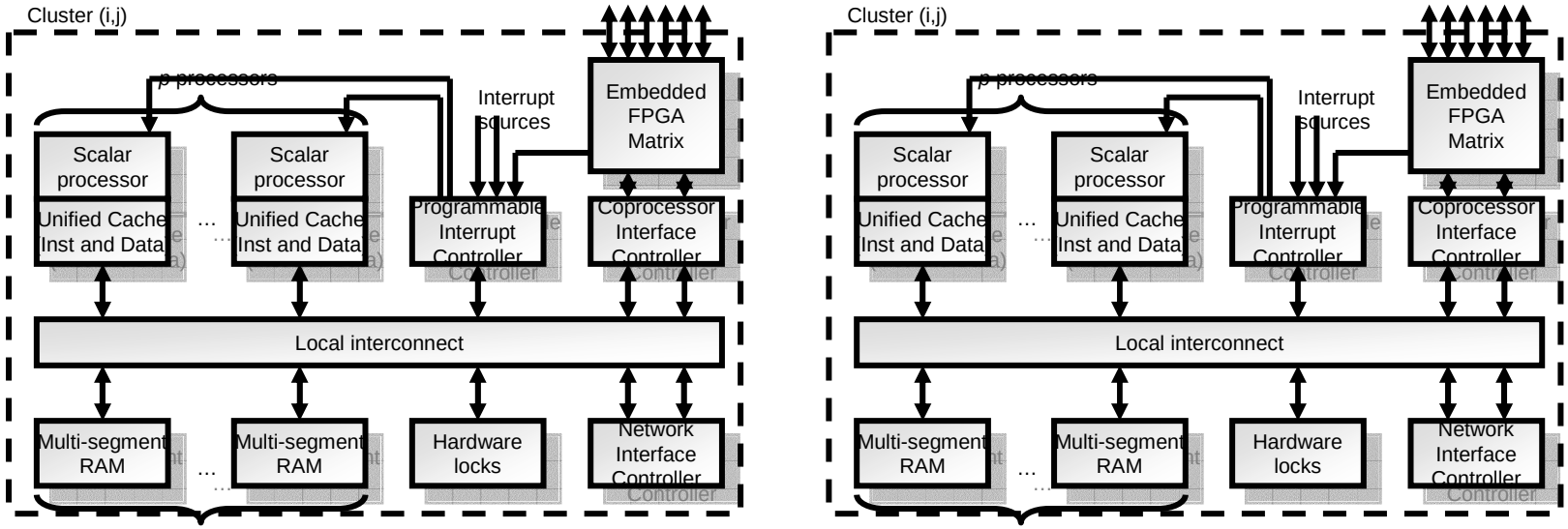
TCG Task and communication graph

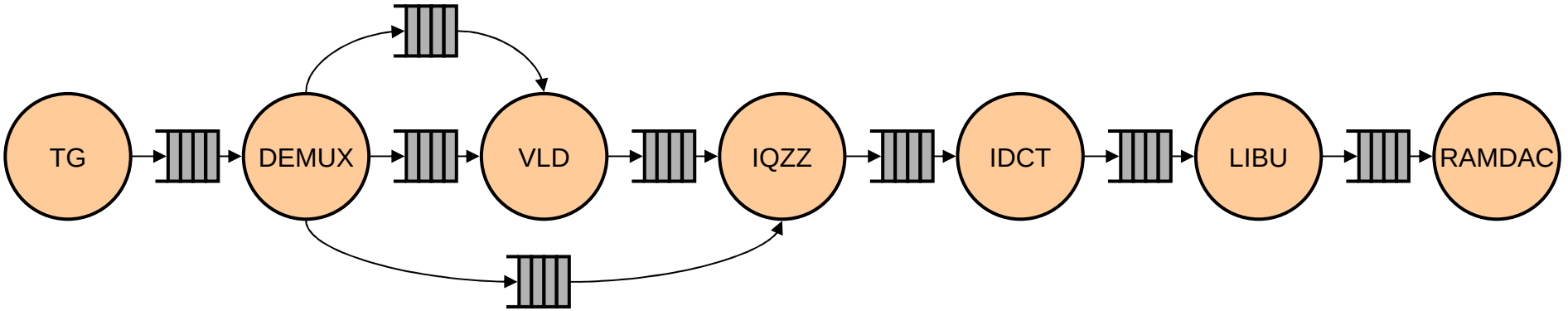
- Hardware or software tasks
- Communicating through software fifos



Example TCG, MJPEG Decoder

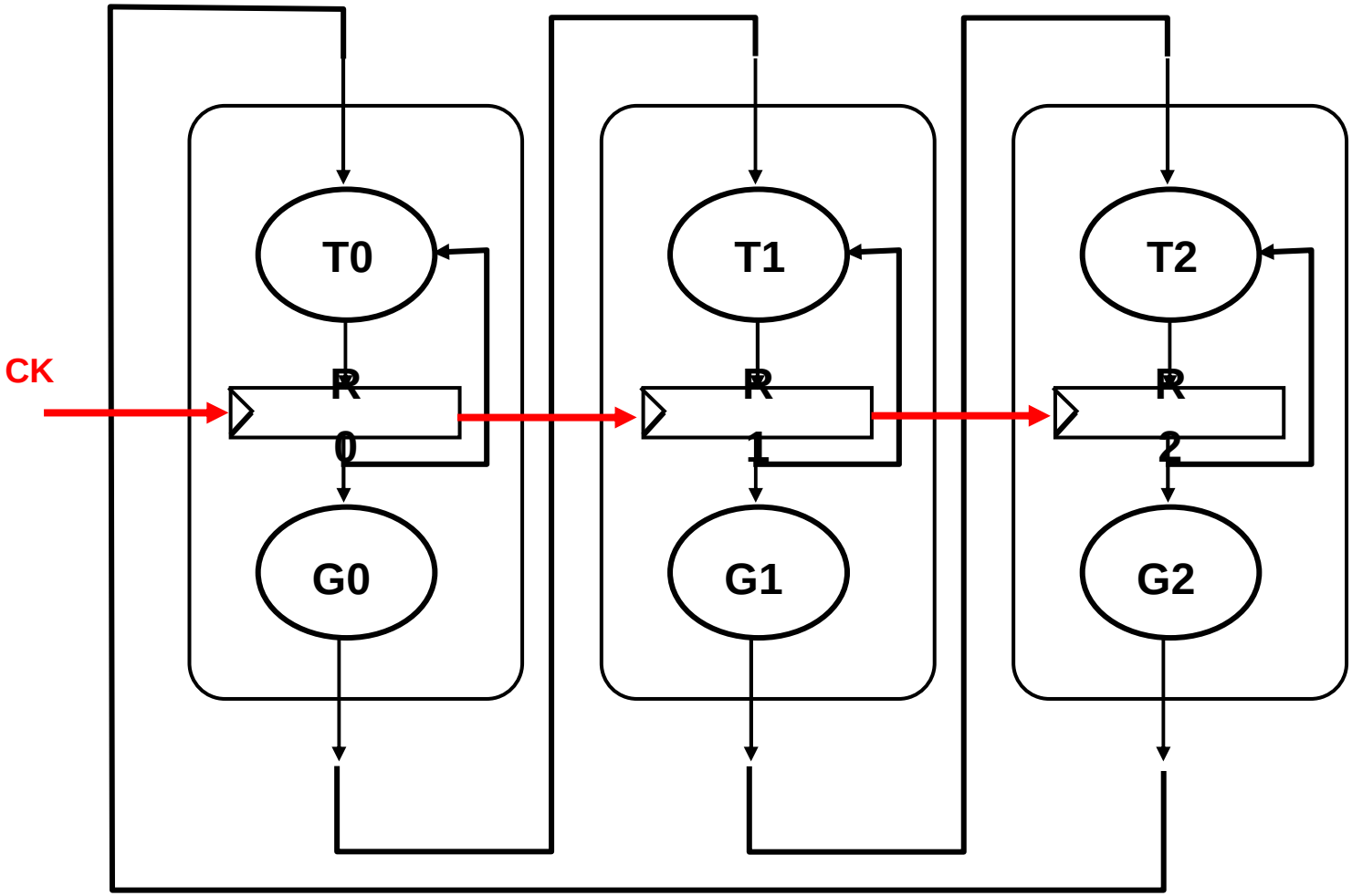
TCG Mapping


Cluster (i,j)
p processors
Interrupt sources
Embedded FPGA Matrix
Scalar processor
Unified Cache Inst and Data
Programmable Interrupt Controller
Coprocessor Interface Controller
Local interconnect
Multi-segment RAM
Hardware locks
Network Interface Controller
Cluster (i,j)
p processors
Interrupt sources
Embedded FPGA Matrix
Scalar processor
Unified Cache Inst and Data
Programmable Interrupt Controller
Coprocessor Interface Controller
Local interconnect
Multi-segment RAM
Hardware locks
Network Interface Controller


TG
DEMUX
VLD
IQZZ
IDCT
LIBU
RAMDAC

CABA modeling

Communicating Finite State Machines Model



Cycle-based Simulation Principle

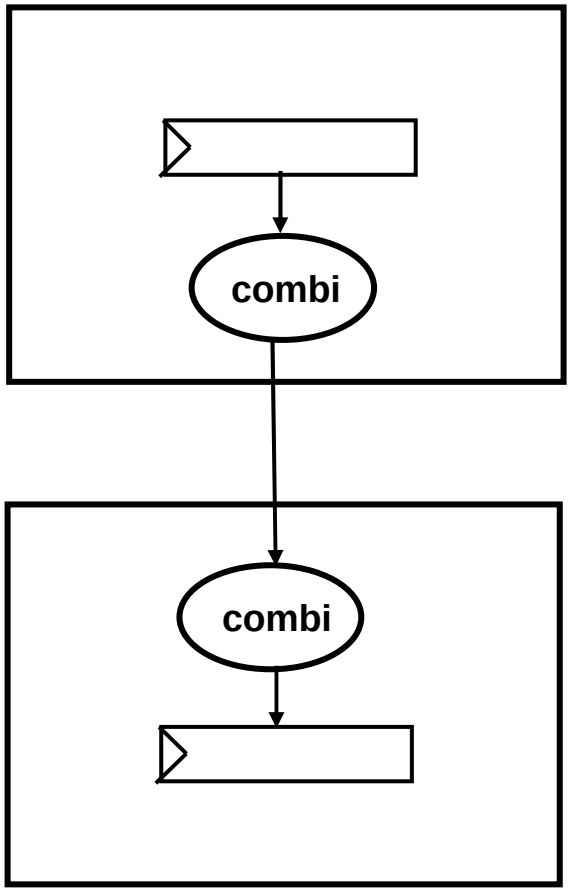
No scheduler ! The simulation engine is just an execution loop :

```
For (cycle = 0 ; cycle < MAX ; cycle++)
        {
        Transition(MODULE_0) ;
        …
        Transition(MODULE_N) ;
        Generation(MODULE_0)
        …
        Generation(MODULE_N) ;
        }
```

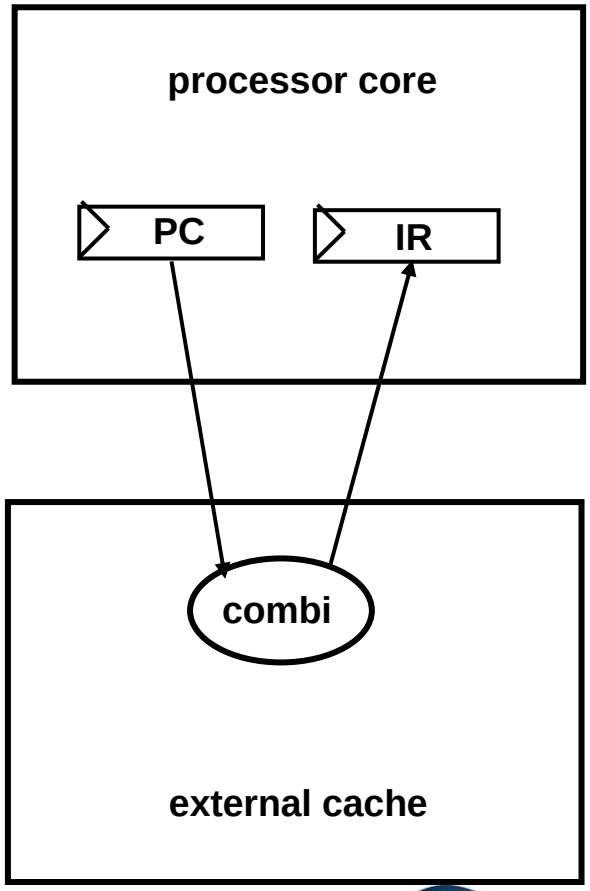
If all components behave as Moore FSMs, the evaluation order is not significant.

The Mealy signals issue

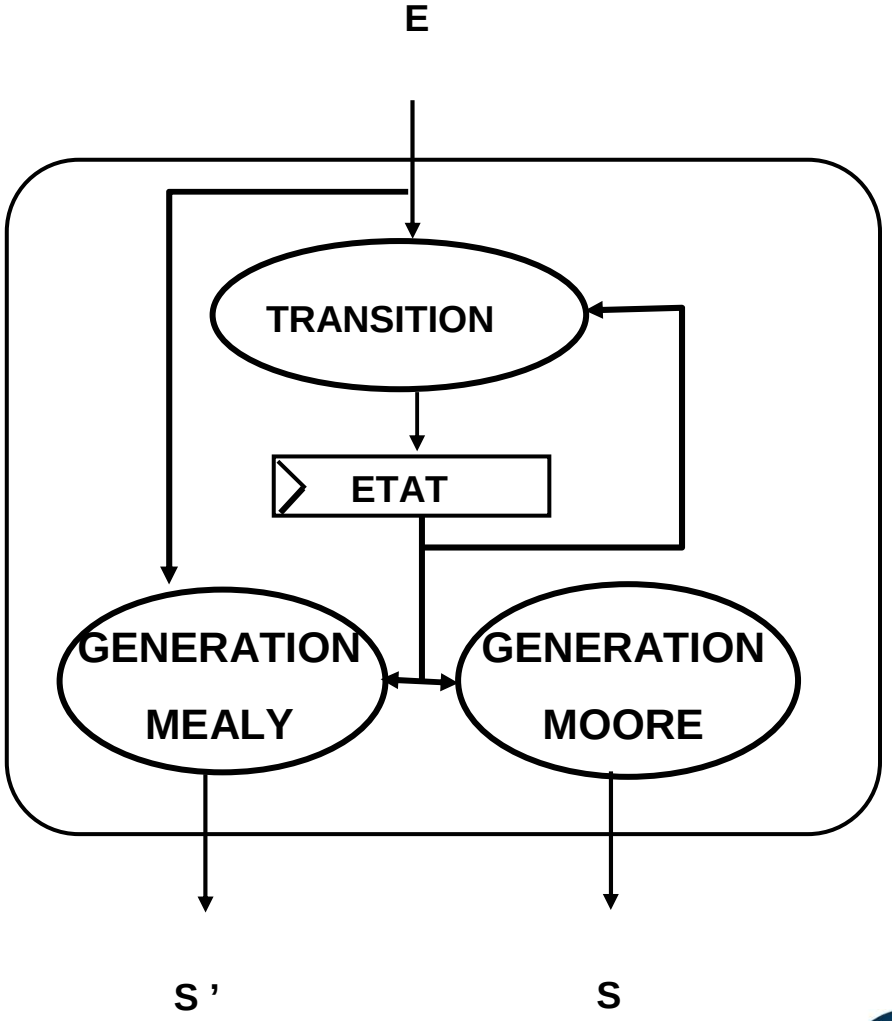
The most frequent case



Combinational dependencies...



CABA model general structure



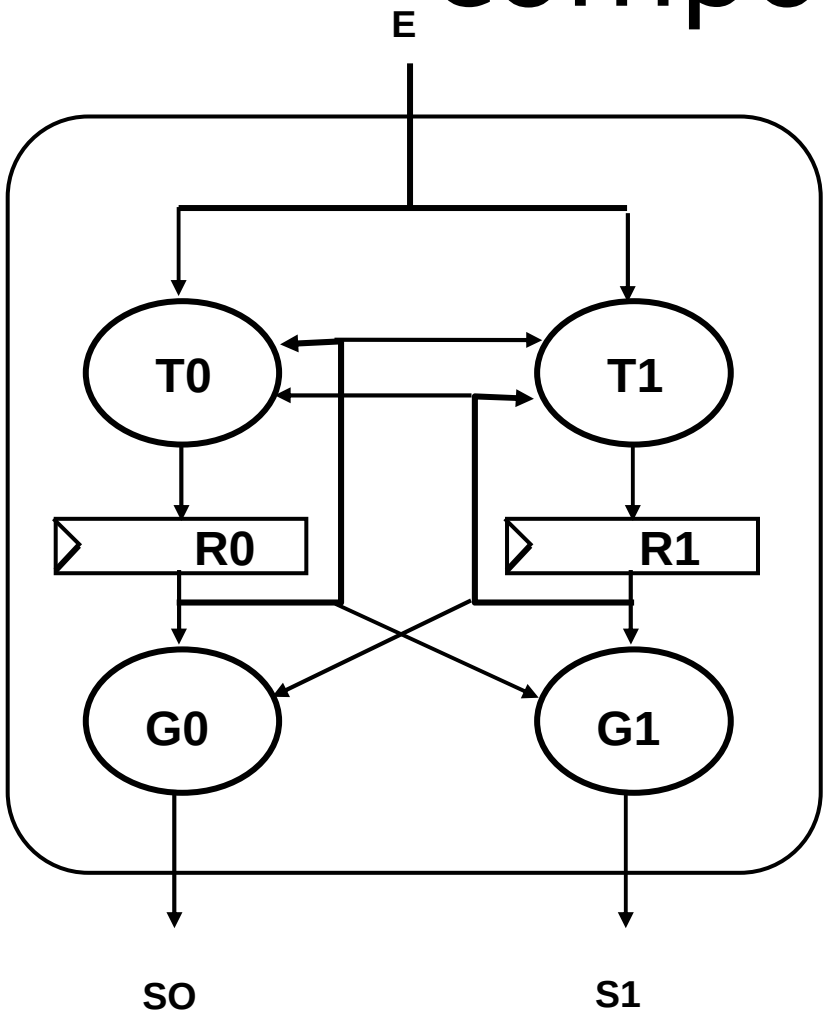
Cycle-based simulation with Mealy signals

```
For (cycle = 0 ; cycle < MAX ; cycle++)
        {
        Transition(MODULE_0) ;
        Transition(MODULE_N) ;
        GenMoore(Module_0)
        GenMoore(Module_N)
        while(instable)
                {
                GenMealy(MODULE_0) ;
                GenMealy(MODULE_N) ;
                }
        }
```

The internal evaluation loop for GenMealy() functions can be optimised

by a proper ordering respecting the Mealy dependencies.

Multi-FSMs component




```
Transition()
{
R0 = T0(E,R0,R1);
R1 = T1(E,R0,R1);
}
```

Delayed assignement provided by SystemC

Cycle-Based simulation with OSCI SystemC

We use the sensitivity lists to force the cycle based scheduling :

Transition() methods contain only the CK rising edge in the sensitivity list.

GenMoore() methods contain only the CK falling edge in the sensitivity list.

GenMealy() methods contain the CK falling edge, plus a selected set of input signals in the sensitivity list.

Cycle-based simulation with SystemCASS

SystemCASS is a static-scheduling simulation engine, optimized to take advantage of the communicating FSMs modeling approach (CFSM) :

a static scheduling, able to handle the Mealy dependencies is computed at elaboration time.

Unlike the general purpose OSCI SystemC simulation kernel, SystemCass only accept models respecting the CFSM approach, but SystemCass is about 10 times faster than OSCI systemC ...

SystemCASS is distributed by UPMC/LIP6 as free software (GPL license), and can be downloaded on the SoCLib WEB server : www.soclib.fr

Some results

Simple architecture containing one MIPS32 (with instruction & data caches), four memory banks, one TTY controler, & one system bus controller.

with System C : 140 000 cycles / s
with SystemCASS : 1 090 000 cycles / s

The simulation time is proportional to the number of Simulated hardware components.

TLM-T over TLM-2.0

alain.greiner@lip6.fr,
aline.vieira-de-mello@lip6.fr,
francois.pecheux@lip6.fr
Université Pierre et Marie Curie,
Laboratoire LIP6/SoC/ALSoC

Special thanks to Daniel Gracia Pérez, Gilles Mouchard from CEA-LIST

Presentation overview

- PDES principles, TLM-T
- TLM-2.0 basics, TLM-AT
- VCI Interconnect modeling
- VCI initiator modeling
- VCI Target modeling
- Conclusion

3 keys for simulation speedup

- Granularity
  In CABA, the transfer unit is the « word ». In TLM-T, the transfer unit is the « packet ».

- Communication by means of IMC
  In CABA, data transfers are performed through signals. In TLM-T, data transfers are performed thanks to Interface Method Calls.

- Parallelizing the simulation
  CABA simulation relies on the SystemC global scheduler that is hard to be parallelized on a SMP workstation. TLM-T simulation can naturally be parallelized using PDES paradigm… if and only we accept to shift from the global simulation time paradigm to the more promising **distributed simulation time** paradigm.

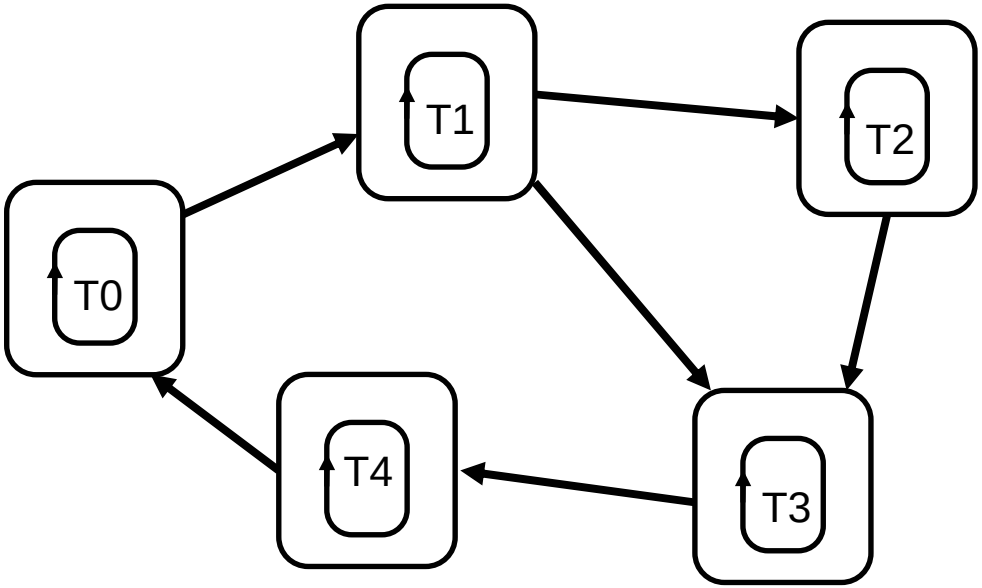
PDES principles

- The complete simulated system is described as a set of **logical processes** that execute in parallel and communicate via point-to-point channels.

- There is neither **global scheduler** nor **global clock**.
- To each process is associated a **local clock** with a value, the **local time**.

- The processes synchronize themselves by **embedding timing information** in the packets carried along the channels.

Optimistic PDES / Pessimistic PDES



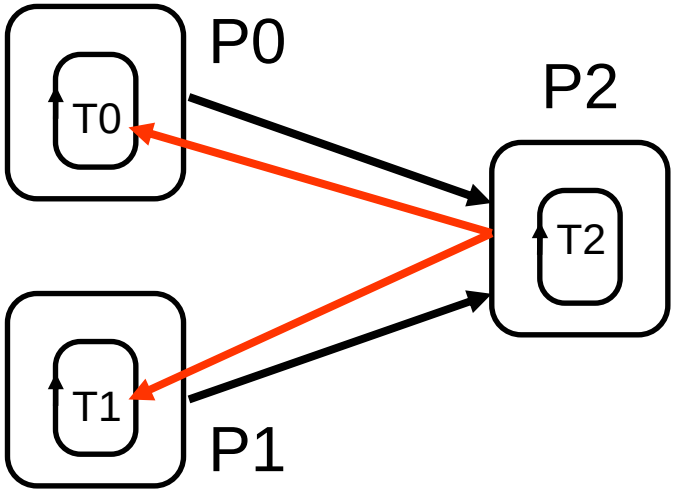
- In pessimistic PDES, a process is allowed to increase its local time if and only if it has been proved **it can not receive on any of its input channels a message with a timestamp smaller than its local time**.

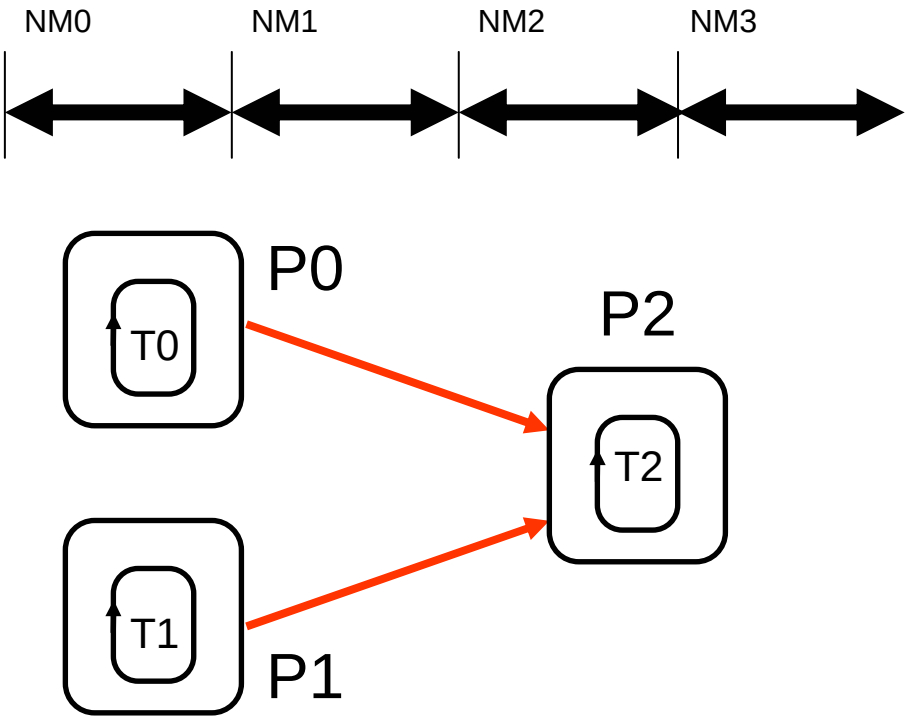
- This constraint is not relevant in optimistic PDES, but the roll-back mechanism needed to put a process into a previous state is very expensive and can not reasonably be used with MPSoC components.

Null-messages

- The pessimistic PDES algorithm relies on temporal filtering of the incoming messages.
- A PDES process that has N input channels is only allowed to process when it has timing information on all its N input ports.
  - For example, an interconnect is allowed to let a command packet reach a given target if and only if all the initiators that can theorically address this target have sent at least one timed message.
- To solve this issue, the traditional PDES algorithm uses ***null message***. A null message contains no data, but only a time information. Moreover, all processes can be in two modes : active & non-active. Only processes that are active participate to the temporal filtering.

Null-messages policies





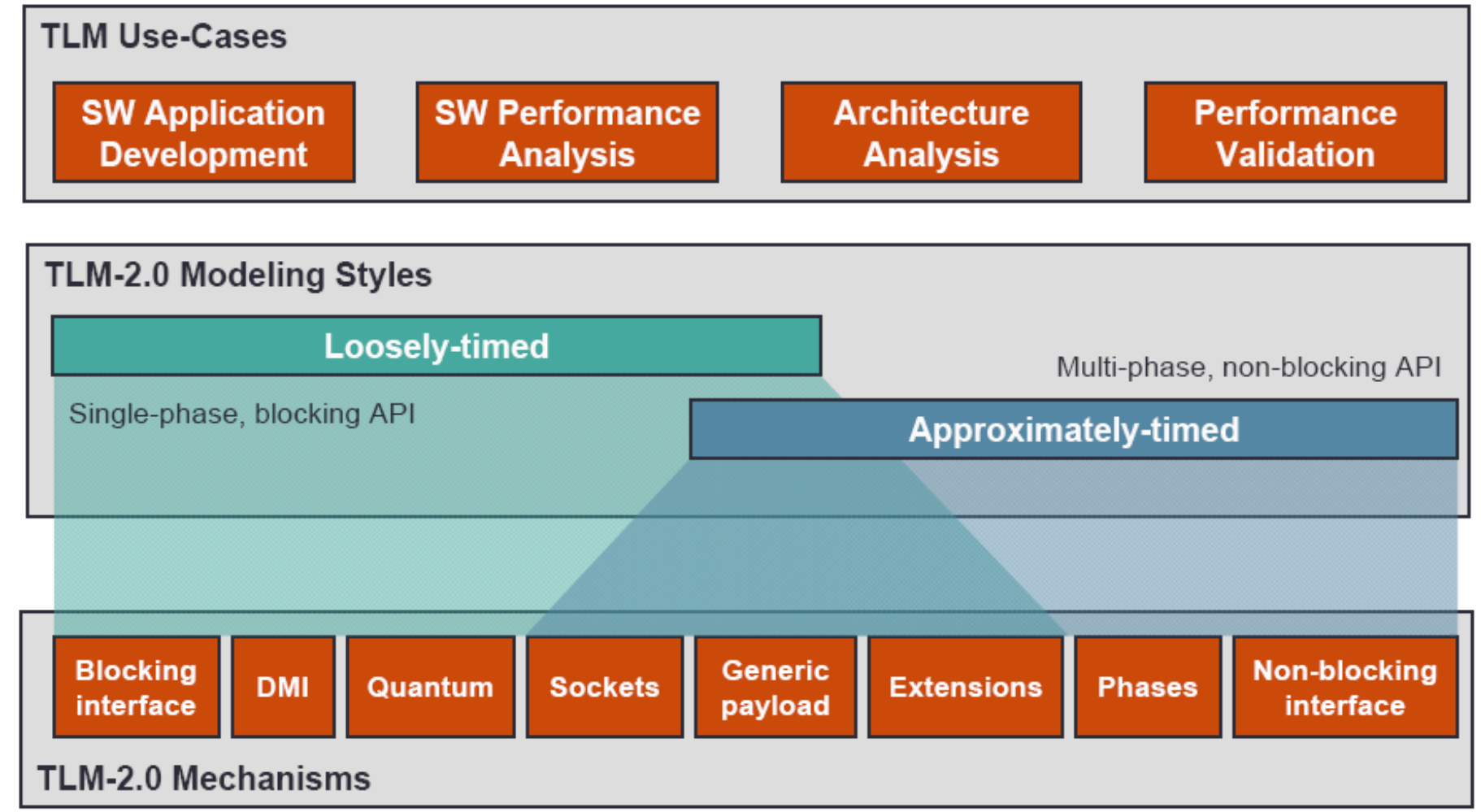
- P2 requests null-messages (soliciting null-messages)
- P0 & P1 are null-message agnostic

- P0 & P1 send their null-messages (**period issue**)
- P2 is simpler

Presentation overview

- PDES principles, TLM-T
- TLM-2.0 basics, TLM-AT
- VCI Interconnect modeling
- VCI initiator modeling
- VCI Target modeling
- Conclusion

TLM 2.0 overview



© T. Kogel, MPSoC'08

Generic payload

- Typical set of memory mapped bus attributes

command	: enum,	READ, WRITE, IGNORE
address	: uint64,	byte address
data	: unsigned char*,	pointer to storage
length	: unsigned int,	number of bytes in the data array
byte_enable	: unsigned char*,	species sub-word accesses
byte_enable_length	: unsigned int,	number of elements in byte_enable
streaming_width	: unsigned int,	defines a streaming burst
response_status	: enum,	INCOMPLETE, OK, ERROR-code

- Extension mechanism
  - Array of pointers to user defined payload extensions
  - Defines rules for ignorable and mandatory extensions

Used in SocLib

- Memory Management
  - Reference counting mechanism
  - Mandatory for AT, optional for LT

© T. Kogel, MPSoC'08

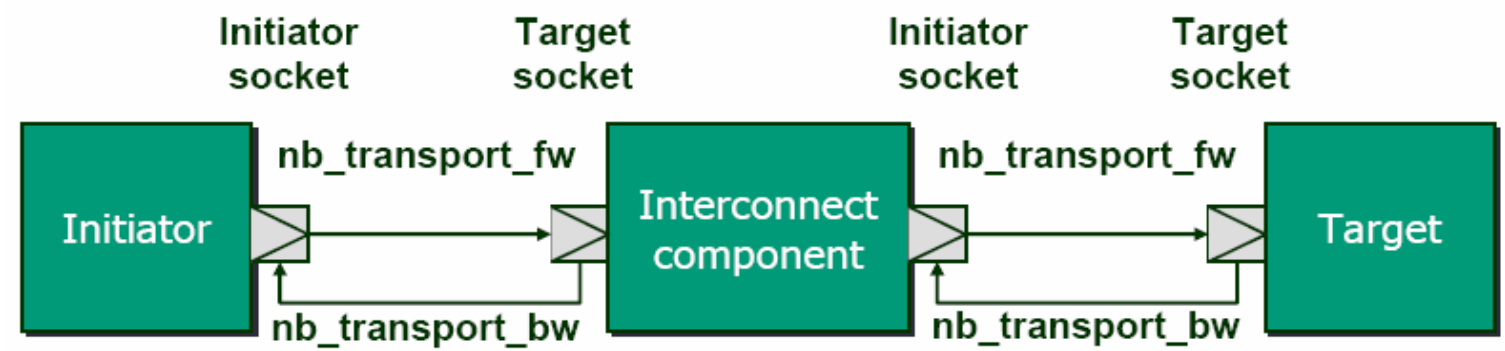
- Helper functions for endianness conversion

soclib_payload_extension

```
enum command m_soclib_command;
unsigned int m_src_id;
unsigned int m_trd_id;
unsigned int m_pkt_id;
```

VCI_READ_COMMAND
VCI_WRITE_COMMAND
VCI_LINKED_READ_COMMAND
VCI_STORE_CONDITIONAL_COMMAND
PDES_NULL_MESSAGE
PDES_ACTIVE
PDES_INACTIVE

Non-blocking transport



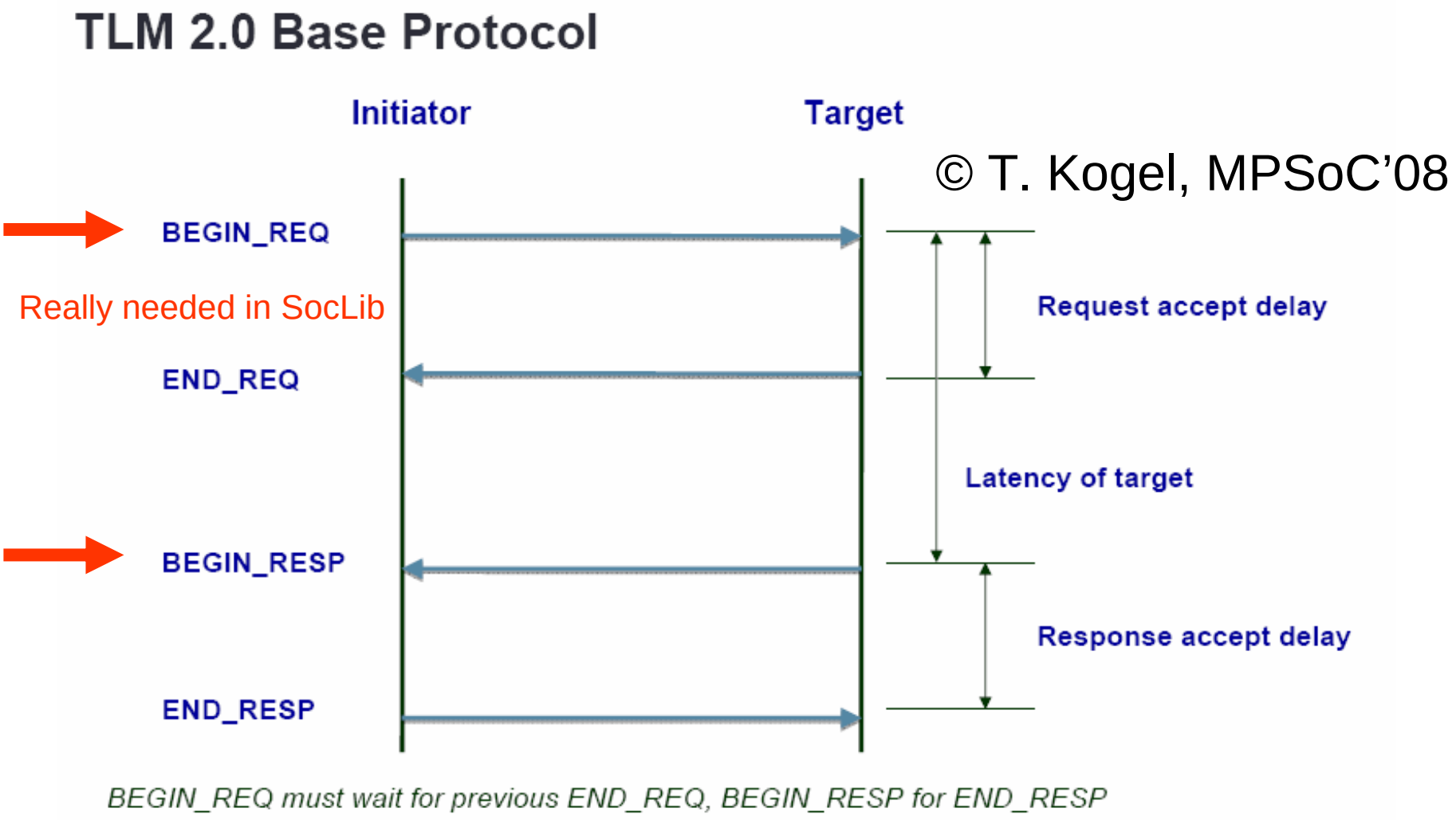

```
template <  typename  TRANS  =  tlm_generic_payload,
            typename  PHASE  =  tlm_phase>

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

payload

Positive **Offset** relative to Global Simulation time

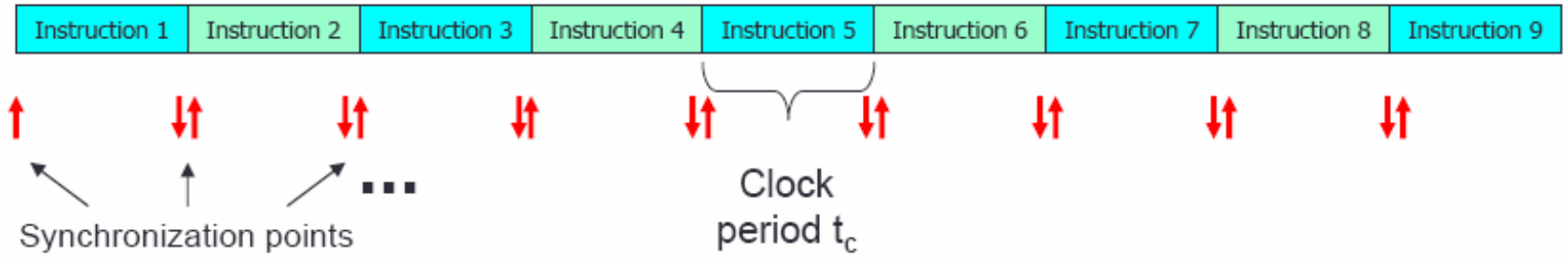
© T. Kogel, MPSoC'08

Transaction phases

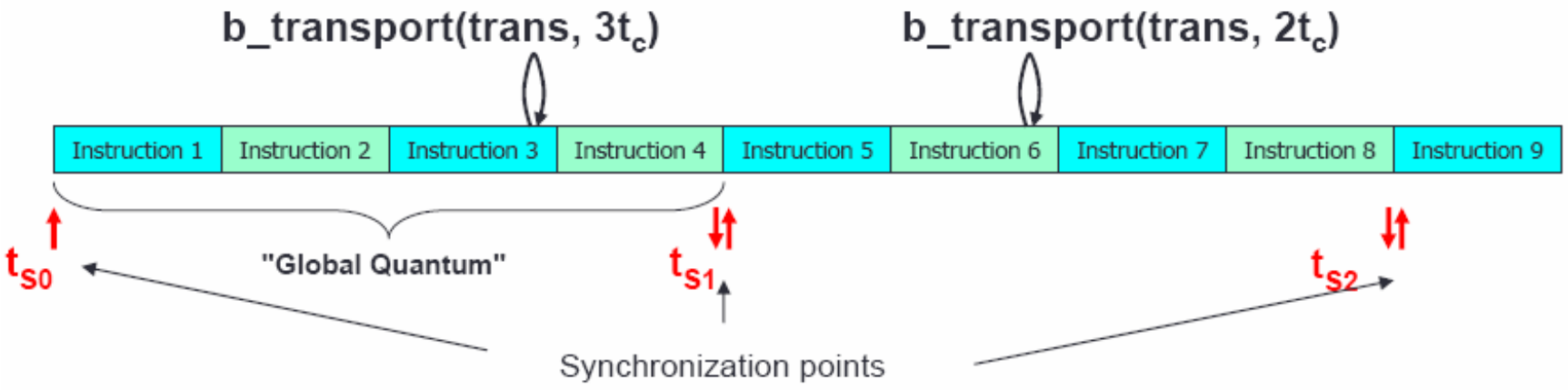


Temporal decoupling

Clock-driven Modeling Style

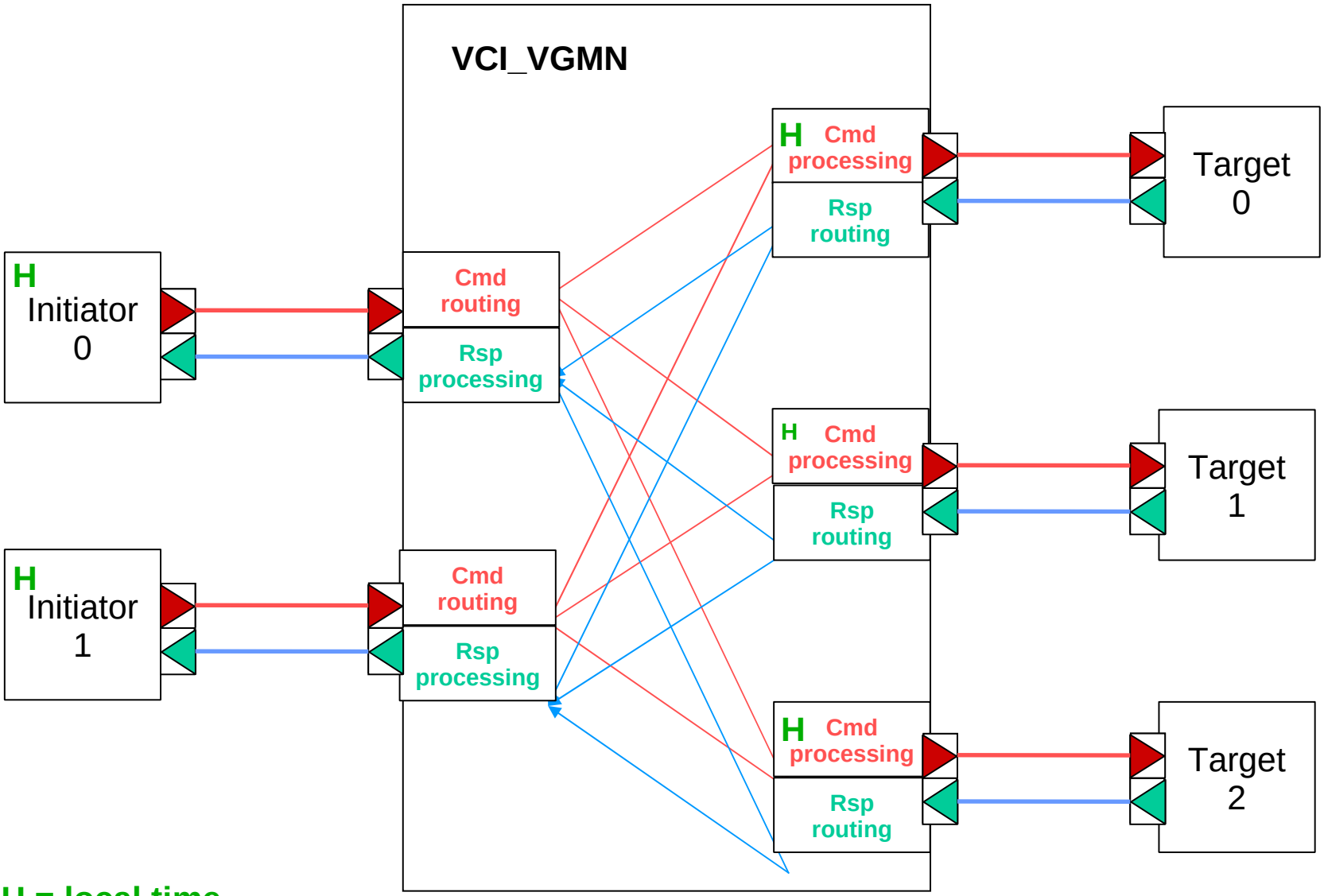


Loosely Timed Modeling Style



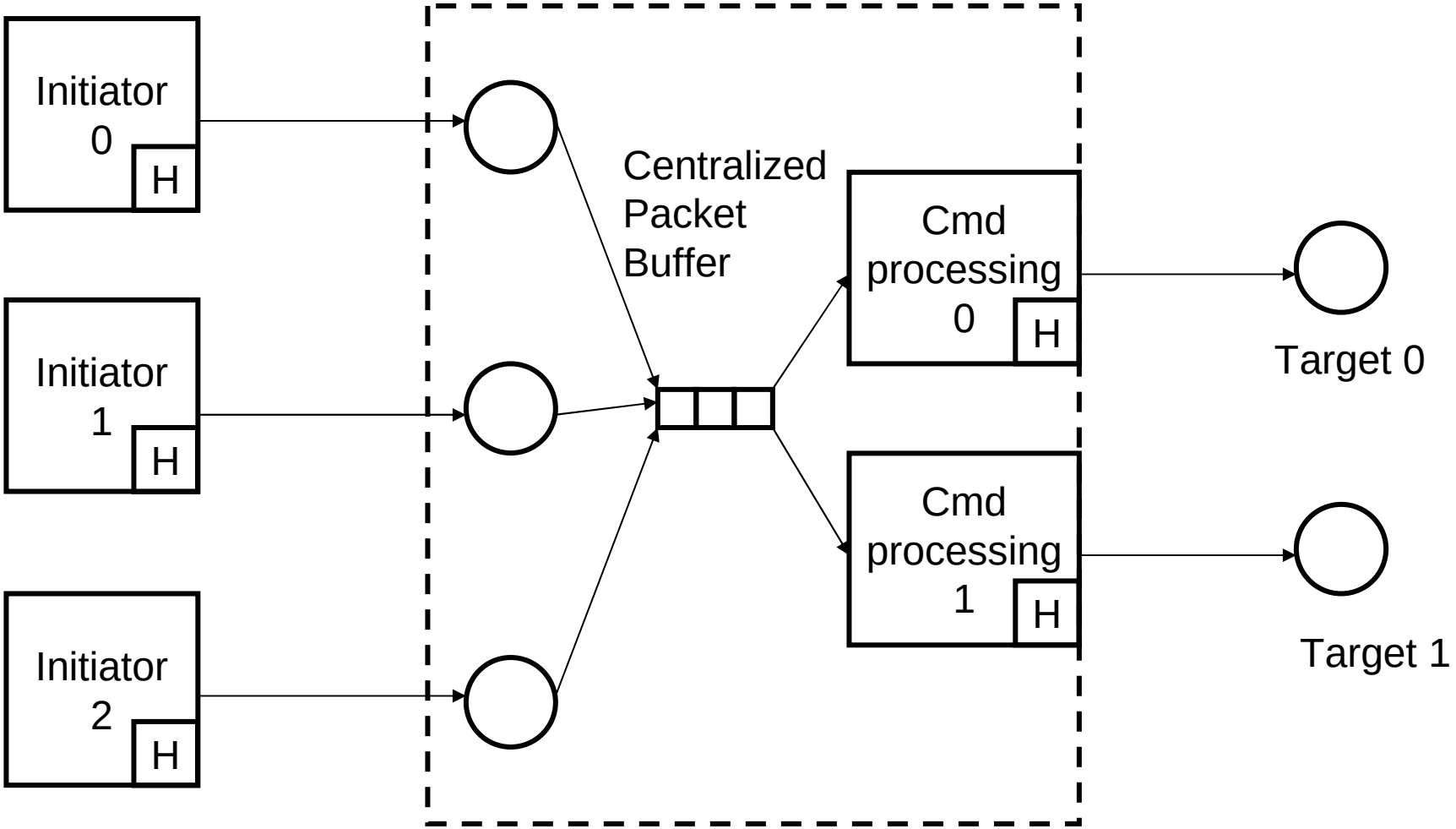
Presentation overview

- PDES principles, TLM-T
- TLM-2.0 basics, TLM-AT
- VCI Interconnect modeling
- VCI initiator modeling
- VCI Target modeling
- Conclusion


VCI_VGMN
H
Initiator
0
H
Initiator
1
Cmd
routing
Rsp
processing
Cmd
routing
Rsp
processing
H Cmd
processing
Rsp
routing
H Cmd
processing
Rsp
routing
H Cmd
processing
Rsp
routing
Target
0
Target
1
Target
2

H = local time

CMB-compliant interconnect



Operates with an interconnect hierarchy

Interconnect behavior

- **Temporal filtering operation** on incoming messages can be factorized for all targets
- These messages are stored in a **centralized data structure**. This structure stores tree information: the packet, the timestamps and the current initiator activity.
- After elaboration of the simulator, the activity information for each initiator is set to true. Only the active initiators are taken into account during filtering. A coprocessor initiator not yet programmed will send a message with **m_soclib_command** set to **PDES_INACTIVE**.
- Therefore, when all slots of this centralized structure are filled with real or null messages with their associated timestamps, a temporal filtering iteration can occur.

Presentation overview

- PDES principles, TLM-T
- TLM-2.0 basics, TLM-AT
- VCI Interconnect modeling
- VCI initiator modeling
- VCI Target modeling
- Conclusion

VCI initiator modeling

- A VCI initiator = {
  - A **TLM initiator port** and the associated transport function to send VCI packets
  - A **behavior function** in the infinite loop of a SC_THREAD
  - A **local time structure** that is continuously updated during component operation
  - A **null-message sending policy**

  }

Initiator constructor

```
my_initiator::my_initiator
          ( sc_core::sc_module_name name,          // module name
            const soclib::common::IntTab &index,    // index of mapping table
            const soclib::common::MappingTable &mt, // mapping table
            sc_core::sc_time time_quantum,         // time quantum
            sc_core::sc_time simulation_time)      // simulation time
          : sc_module(name),                    // init module name
          m_mt(mt),                           // mapping table
          p_vci_initiator("socket")              // vci initiator socket name
{
 //register callback function VCI INITIATOR SOCKET
 p_vci_initiator.register_nb_transport_bw(this, &my_initiator::my_nb_transport_bw);

 //initiator identification
 m_srcid = mt.indexForId(index);

 //PDES local time
 m_pdes_local_time = new pdes_local_time(time_quantum);

 //PDES activity status
 m_pdes_activity_status = new pdes_activity_status(true);

 //determine the simulation time
 m_simulation_time = simulation_time;

 // register thread process
 SC_THREAD(behavior);
}
```

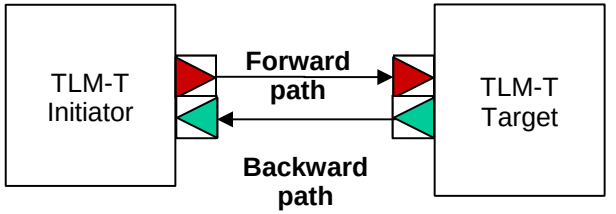
Local time structure
pdes_local_time

- Constructor (sc_core::sc_time TimeQuantum)

- void **add**(sc_core::sc_time time)
- void **set**(sc_core::sc_time time)
- sc_core::sc_time **get**()
- bool **need_sync**()

Sending a real write packet

```
tlm::tlm_generic_payload *payload_ptr = new tlm::tlm_generic_payload();
tlm::tlm_phase           phase;
sc_core::sc_time         time;
soclib_payload_extension *extension_ptr = new soclib_payload_extension();

  payload_ptr->set_command(tlm::TLM_IGNORE_COMMAND);
  payload_ptr->set_address(0x10000000);
  payload_ptr->set_byte_enable_ptr(byte_enable);
  payload_ptr->set_byte_enable_length(nbytes);
  payload_ptr->set_data_ptr(data);
  payload_ptr->set_data_length(nbytes);
  // set the values in payload extension
  extension_ptr->set_write();
  extension_ptr->set_src_id(m_srcid);
  extension_ptr->set_trd_id(0);
  extension_ptr->set_pkt_id(0);
  // set the extension to tlm payload
  payload_ptr->set_extension (extension_ptr );
  // set the tlm phase
  phase = tlm::BEGIN_REQ;
  // set the local time to transaction time
  time = m_pdes_local_time->get();

  // send the transaction
  p_vci_initiator->nb_transport_fw(*payload_ptr, phase, time);
  wait(m_rspEvent);
```



Generic part

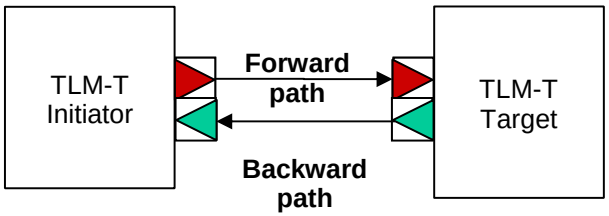
Extension part

payload

Time is **ABSOLUTE** !

Receiving the response packet

```
tlm::tlm_sync_enum my_initiator::my_nb_transport_bw       // inbound
    nb_transport_bw
( tlm::tlm_base_protocol_types::tlm_payload_type &payload, // payload
  tlm::tlm_base_protocol_types::tlm_phase_type   &phase,   // phase
  sc_core::sc_time                               &time)    // time
{
  m_pdes_local_time->set(time);
  m_rspEvent.notify(sc_core::SC_ZERO_TIME);
  return tlm::TLM_COMPLETED;
} // end backward nb transport
```



Null-message policy implementation

```
m_pdes_local_time->add(1000 * UNIT_TIME);

// if initiator needs to synchronize then it sends a null message
if (m_pdes_local_time->need_sync()) {
        // set the null message command
        extension_ptr->set_null_message();
        extension_ptr->set_src_id(m_srcid);
         // set the extension to tlm payload
        payload_ptr->set_extension(extension_ptr);
        //set the tlm phase
        phase = tlm::BEGIN_REQ;
        //set the local time to transaction time
        time = m_pdes_local_time->get();

        //send a null message
        p_vci_initiator->nb_transport_fw(*payload_ptr, phase, time);
        //deschedule the initiator thread
        wait(sc_core::SC_ZERO_TIME);
}
```

Presentation overview

- PDES principles, TLM-T
- TLM-2.0 basics, TLM-AT
- VCI Interconnect modeling
- VCI initiator modeling
- VCI Target modeling
- Conclusion

VCI target modeling (1)

```
class my_target                          // vci ram
  : public sc_core::sc_module              // inherit from SC module base clase
{
 private:
  typedef soclib::tlmt::VciParams<uint32_t,uint32_t,4> vci_param;

tlm::tlm_sync_enum my_nb_transport_fw    /// receive request from an
      initiator
  ( tlm::tlm_generic_payload &payload,          /// payload
    tlm::tlm_phase &phase,                    /// phase
    sc_core::sc_time &time);                  /// time

 protected:
  SC_HAS_PROCESS(my_target);
 public:

tlm_utils::simple_target_socket<my_target,
     32,tlm::tlm_base_protocol_types> p_vci_target;      // VCI TARGET socket

};
```

VCI target modeling (2)

```
tlm::tlm_sync_enum my_target::my_nb_transport_fw (
      tlm::tlm_generic_payload &payload,
      tlm::tlm_phase          &phase, sc_core::sc_time        &time) {
  soclib_payload_extension *extension_pointer;
  payload.get_extension(extension_pointer);

switch(extension_pointer->get_command()){
  case VCI_READ_COMMAND: … break;
  case VCI_WRITE_COMMAND: … break;
  case VCI_LINKED_READ_COMMAND:
  case VCI_STORE_COND_COMMAND:
  default:
    //send error message
    payload.set_response_status(tlm::TLM_COMMAND_ERROR_RESPONSE);
    break;
  }
  //modify the phase
  phase = tlm::BEGIN_RESP;
  //increment the target processing time
  time = time + nwords * UNIT_TIME;
  //send the response
  p_vci_target->nb_transport_bw(payload, phase, time);
  return tlm::TLM_COMPLETED;
}
```

Presentation overview

- PDES principles, TLM-T
- TLM-2.0 basics, TLM-AT
- VCI Interconnect modeling
- VCI initiator modeling
- VCI Target modeling
- Conclusion