OCCN: A Network-On-Chip Modeling and Simulation Framework

M.Coppola, S.Curaba, M.Grammatikakis,
R.Locatelli, G.Maruccia, F.Papariello, L.Pieralisi

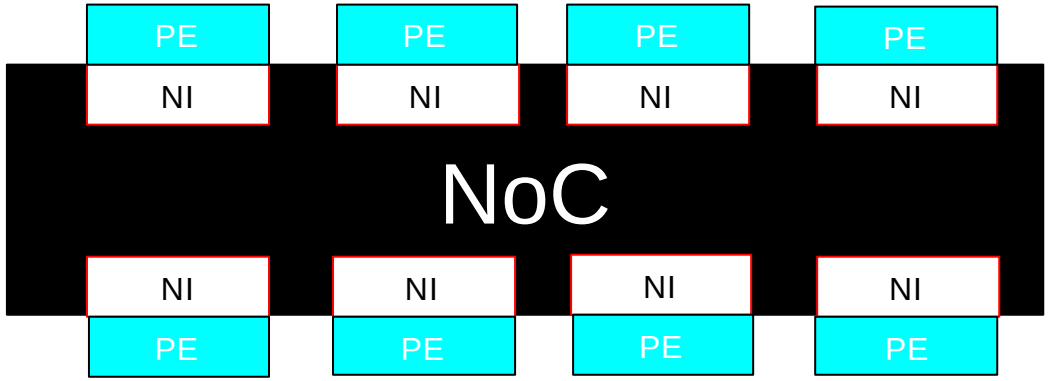
Outline

- Introduction
- SoC trends
  - SoC: Towards a NoC centric design
  - NoC centric design flow
- OCCN: Methodology for communication modeling
  - What is OCCN ?
  - OCCN conceptual model
- OCCN core
  - Generic representation of a connection
  - Framework key points
- Performance measurement with Grace
- Evolution for NoC
- Conclusion

Introduction

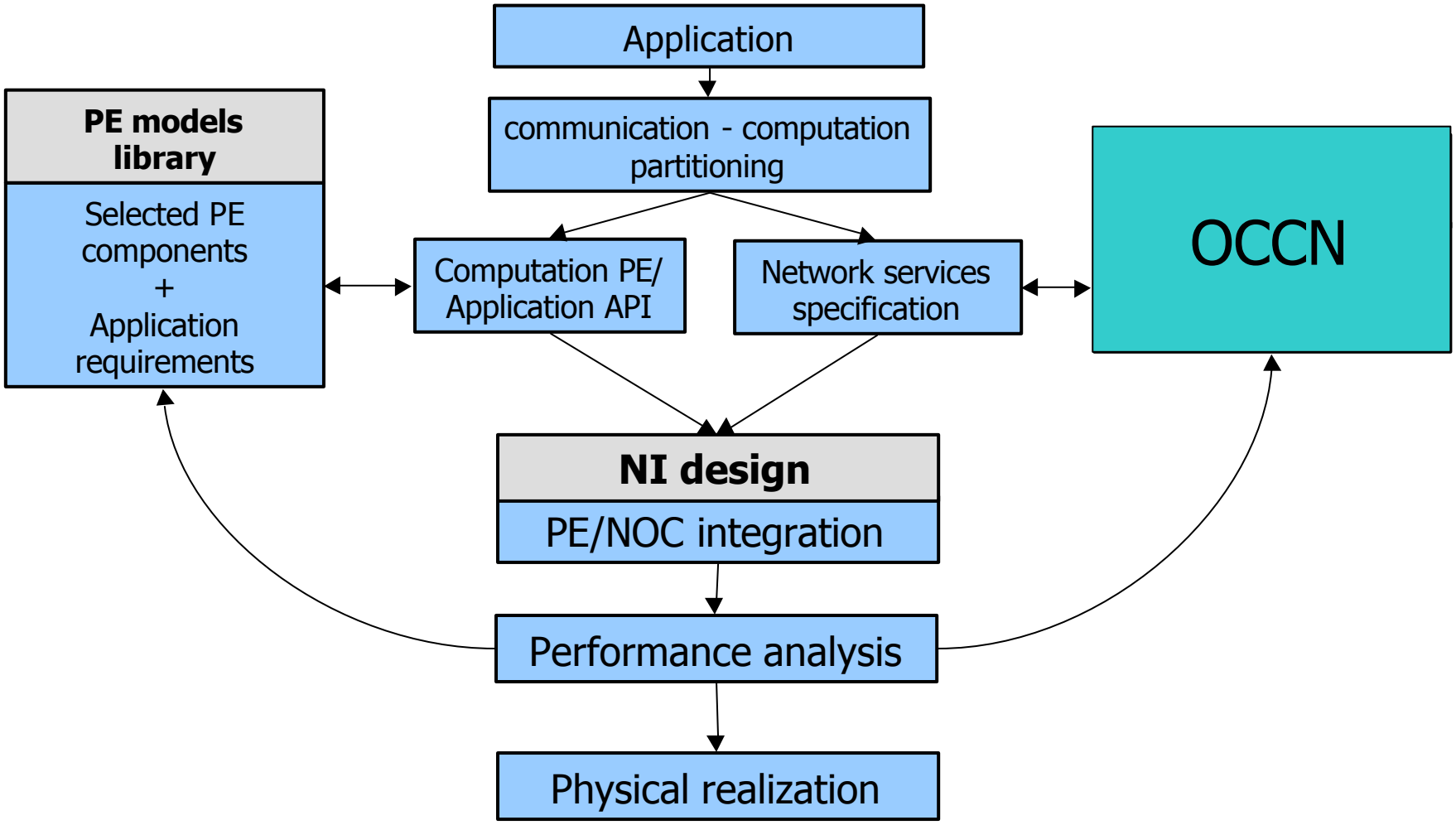
- SoC interconnection backbones are moving towards NoC paradigm in order to solve DSM issues and to master on-chip complexity
- Future SoC simulation platforms will require the development of NoC models and modeling of NoC stack protocol in order to speed up the exploration of different solutions and to allow flexible architecture specification
- Increasing interest in NoC simulation environments
  - Co-simulation between the network and the rest of the chip
  - Modeling of NoC protocol stack
  - Well defined APIs in order to simplify models exchange and re-use
- OCCN Methodology

SoC – Towards a NoC centric design



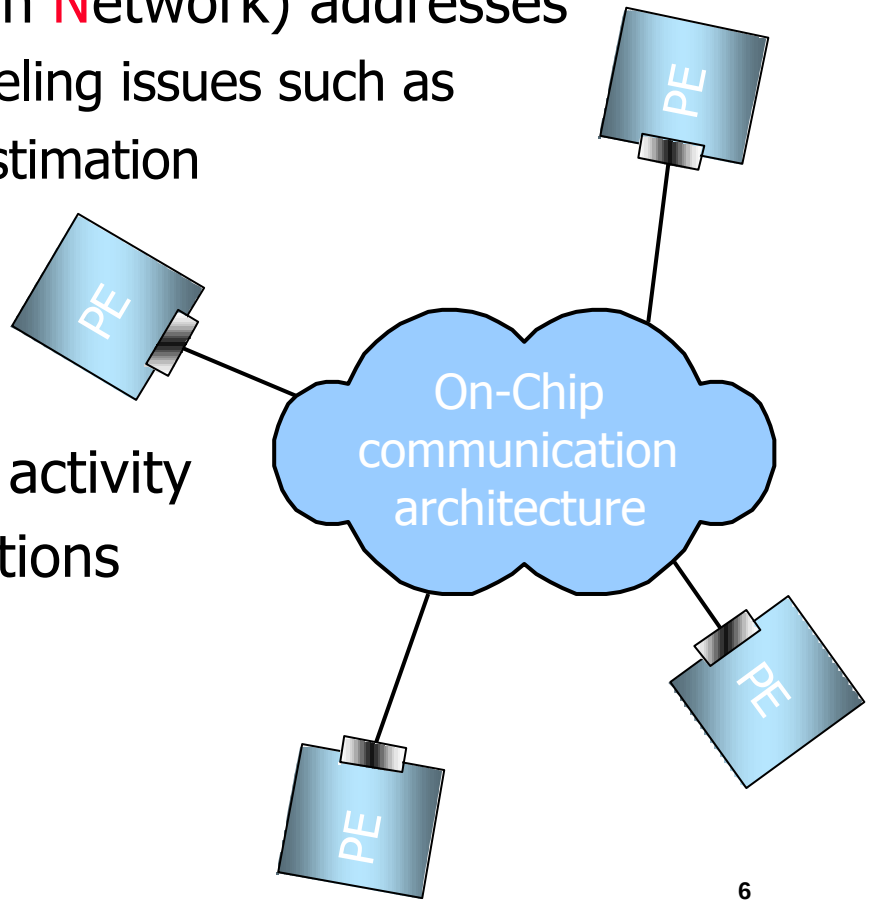
- To empower designers with techniques and tools to map the system's communication requirements onto a well optimized communication architecture
- Key issues
  - NOC design
  - NI design
- Massive interest in NoC and NI simulation environments

NoC centric design flow



OCCN : Methodology for communication modeling

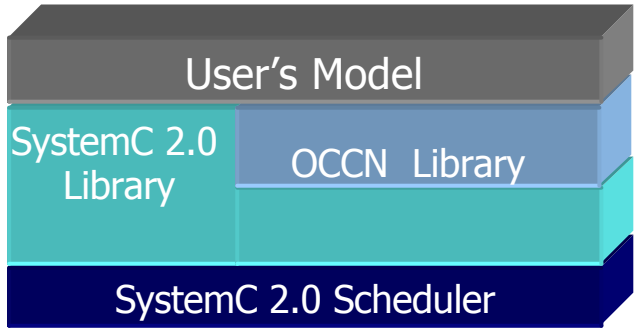
- Generic communication-centric design methodology based on C++ and SystemC
- OCCN (On-Chip Communication Network) addresses
  - high level performance modeling issues such as speed, latency and power estimation
  - modeling productivity
  - model portability
  - simulation speed-up
- OCCN is an on-going research activity between several R&D organizations

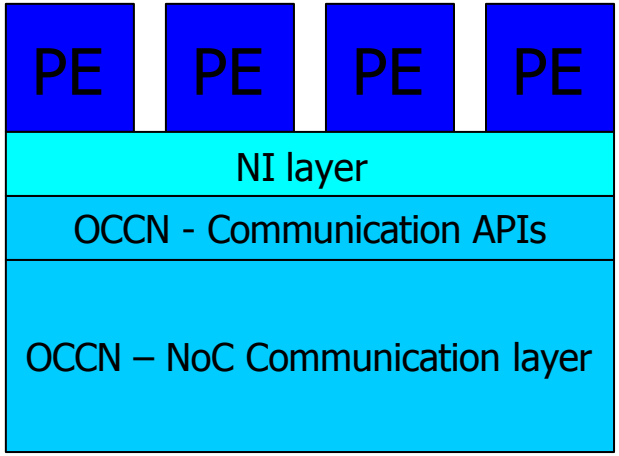


What is OCCN ?

OCCN: A framework for NoC modelling

- open source C++ code built on top of SystemC
- Generic message passing APIs simplify the task of implementing communication drivers at different levels of abstraction
- Blocking send/receive primitives





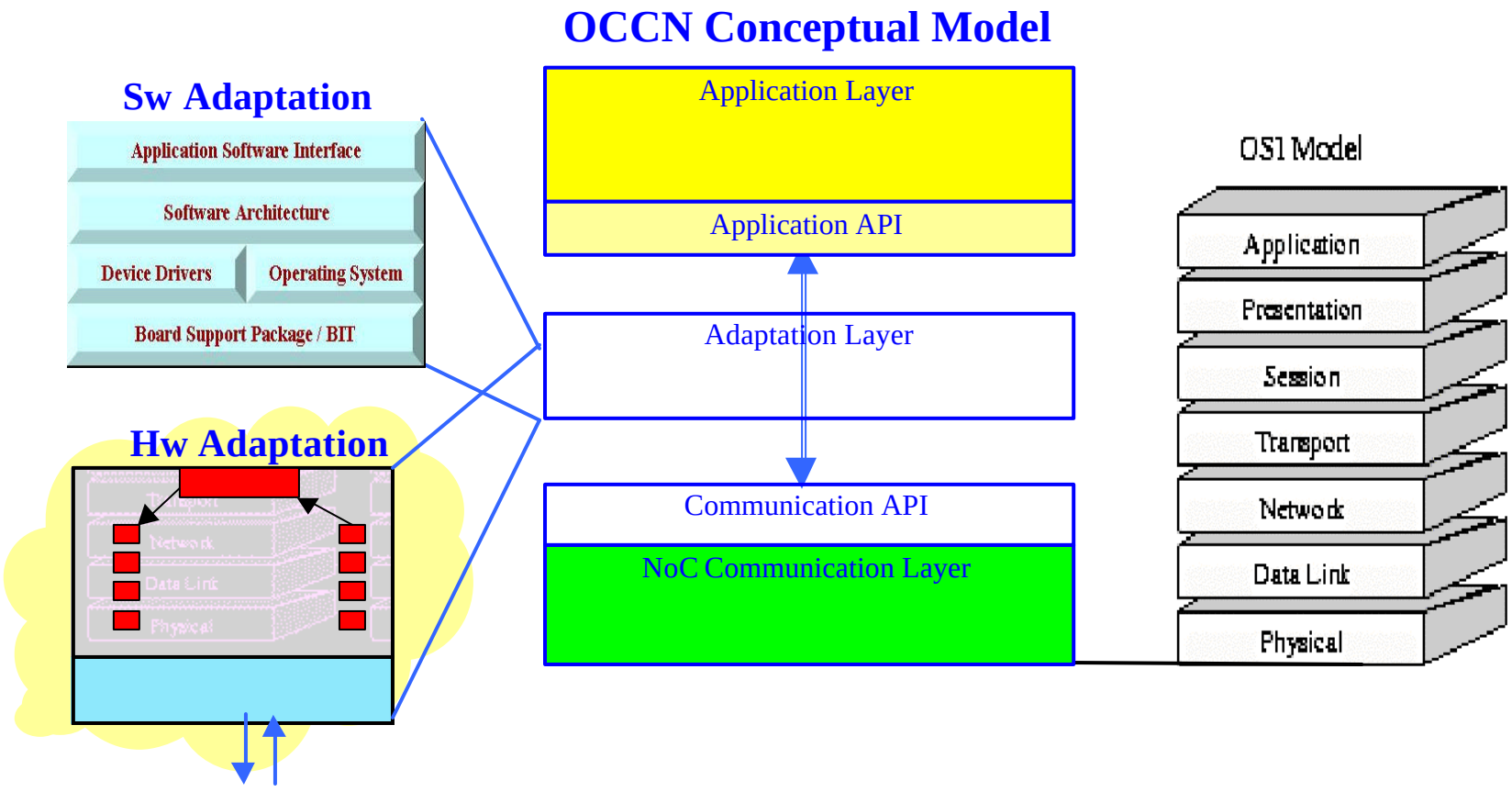
NoC communication layer

- set of C++ classes derived from **sc_channel**
- channel establishes transfer of messages among different ports according to the protocol stack supported by a specific NoC

Communication APIs

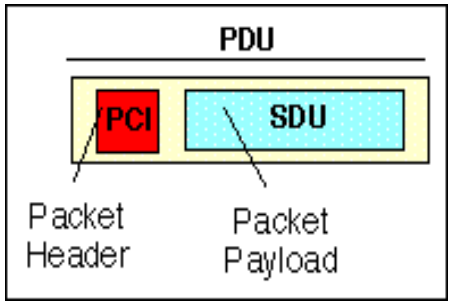
- specialization of **sc_port** SystemC object
- message passing paradigm for inter-module communication

OCCN Conceptual Model



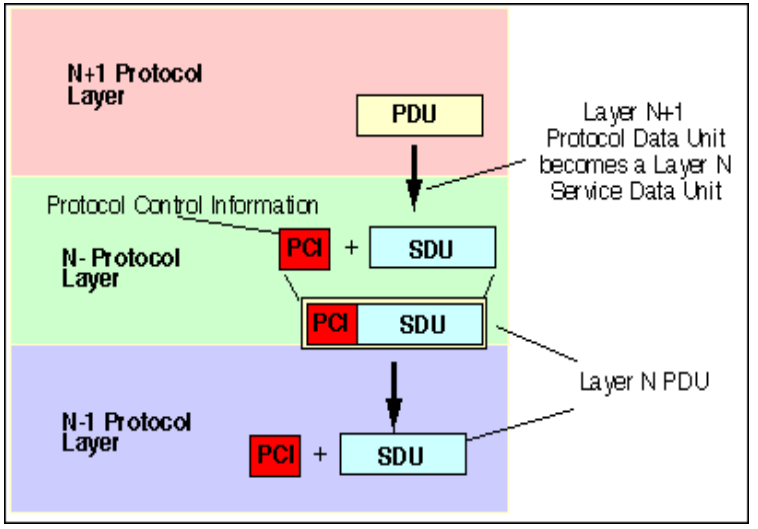
OCCN core: the PDU

- Protocol = syntax + semantics
  - syntax = PDU
  - semantics = how the PDUs are exchanged



- The PDUs exchanged have two parts:
  - a header also known as the Protocol Control Information (PCI)
  - a payload also known as a Service Data Unit (SDU)
- Several operators are defined for handling protocol operations (segmentation/reassembling)
- Syntax example

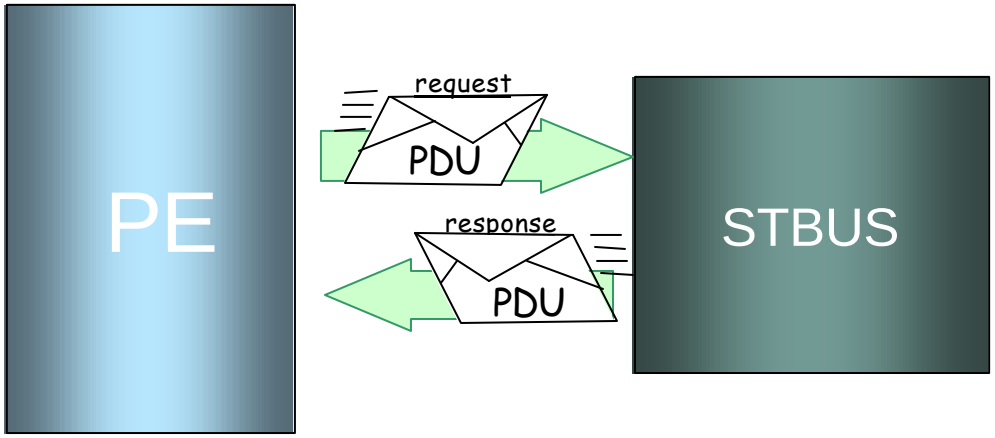
```
struct MyHeader {int P; char T;};
Pdu<MyHeader,char,4> my_pdu;
```



Generic representation of a connection

- Any connection of a module to the communication node (network) is based on 2 sets of PDU
  - **Pdu< PCIRequest, uint32>**
  - **Pdu< PCIResponse, uint32>**
- The PCI and SDU sets are defined according to the bus specification and thus are specific to a model.

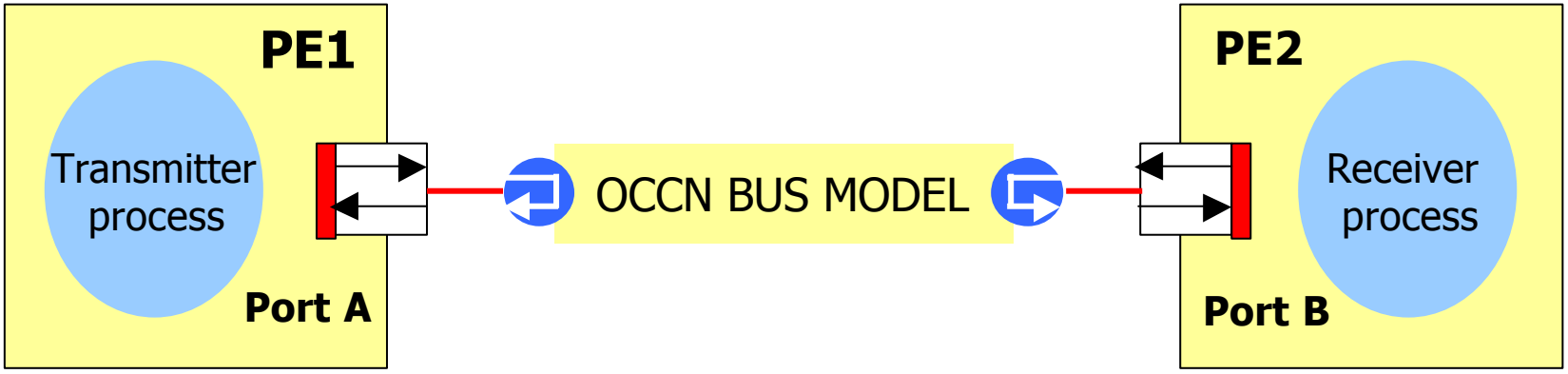
  For instance they will be different for an AHB model and an STBUS model




```
struct PCIRequest
{
  bool Request;
  unsigned int address;
  unsigned char Opcode;
  bool Lock;
  unsigned char SrcId;
   …
};
```

```
struct PCIResponse
{
  bool ReturnRequest;
  unsigned char ReturnOpcode;
  unsigned char SrcId;
  …
};
```

OCCN core : API syntax
simple message passing API



void asend(Pdu<>* p)

void send(Pdu<>* p)

void send(Pdu<>* p, sc_time& time_out, bool& sent)

void asend(Pdu<>* p, sc_time& time_out, bool& sent)

Pdu<>* receive()

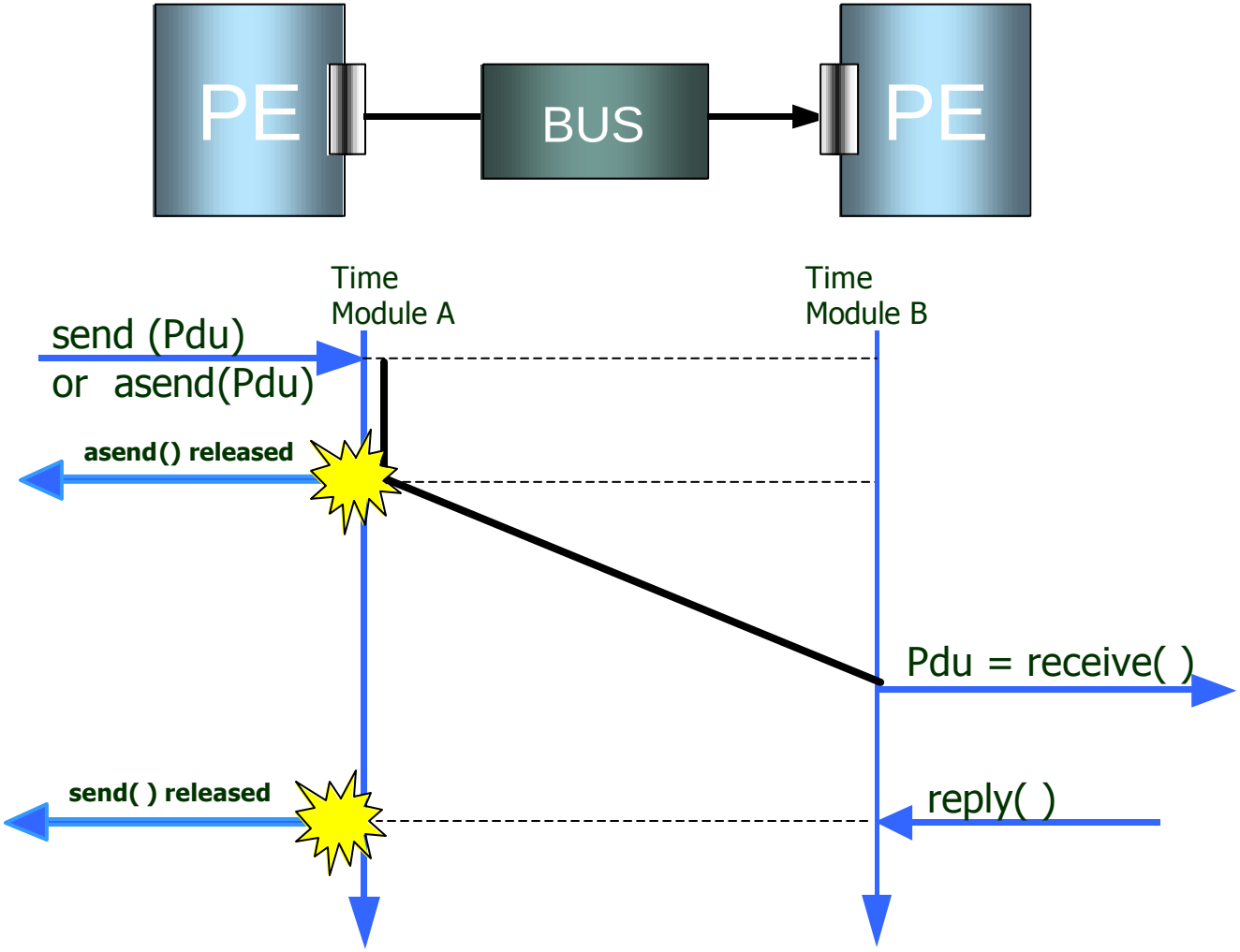
void reply()

void reply(uint nb_cycles)

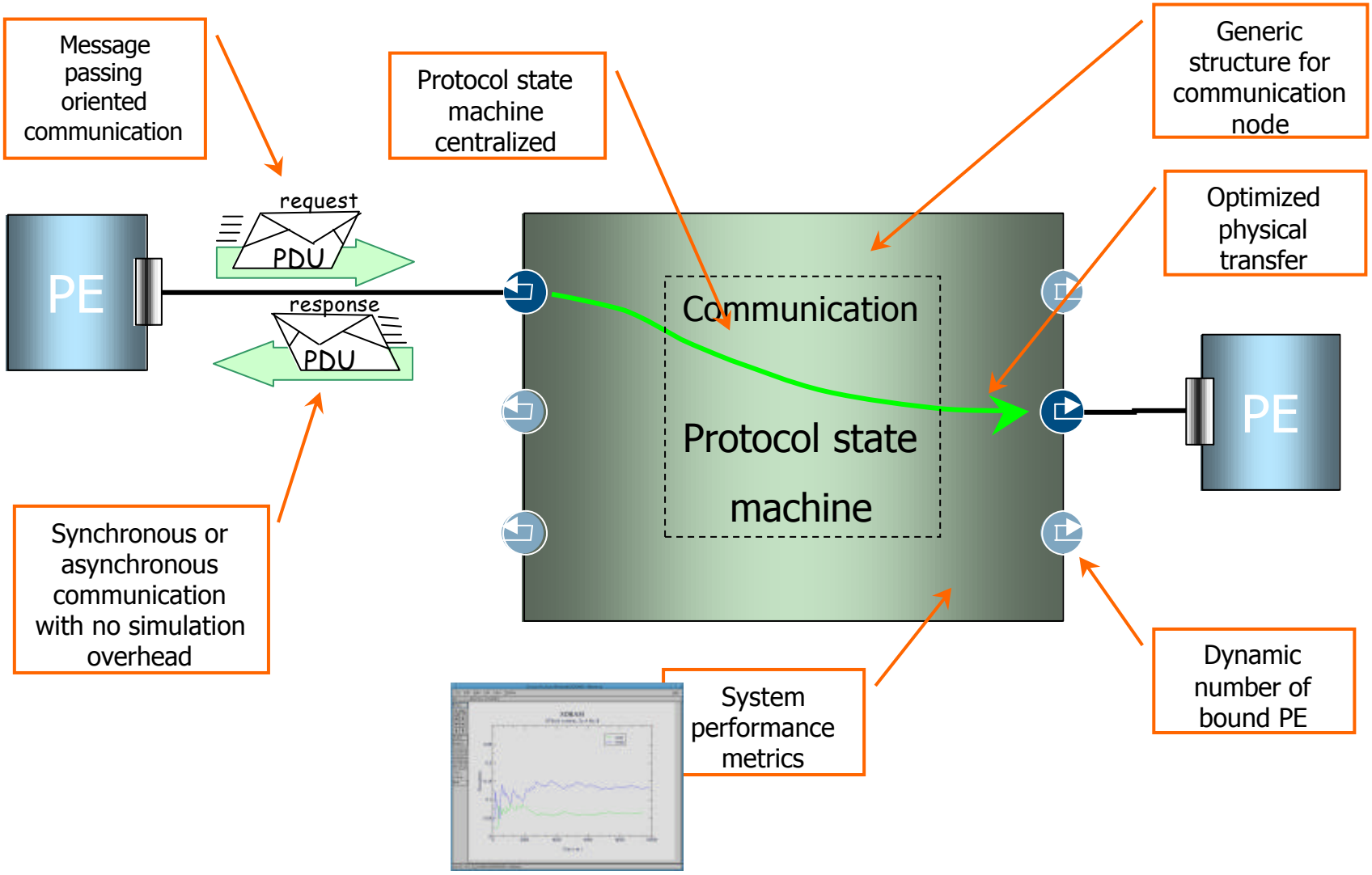
void reply(sc_time& delay)

Pdu<>* receive(sc_time& time_out, bool& received)

OCCN core : API semantics with or without acknowledge



OCCN framework key points



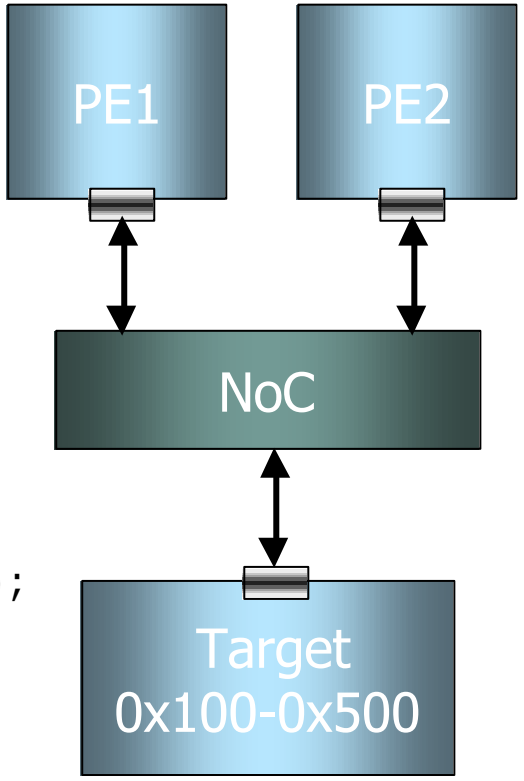
OCCN: PE code example

```
#include “producer.h”
producer::producer(sc_module_name name) : sc_module(name)
{SC_THREAD(read);}


void producer::read() {
   char c;
   Pdu<char>* msg;
   while (cin.get(c)) {
         msg = new Pdu<char>;
         // producer sends c
         *msg = c;
         out.send(msg);
    }// after the send the msg is not usable

}
```

Protocol inlining:
protocol is automatically generated

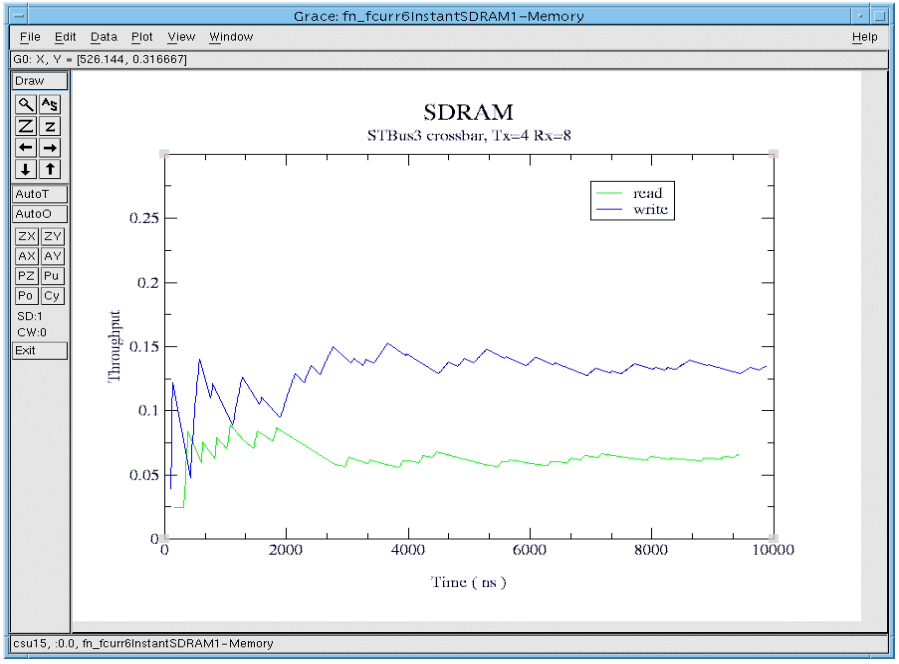
MPSoC architecture in OCCN

```
main()
{
        sc_clock my_clock(10, SC_NS);
        PE pe1, pe2;
        SE se1;
        NoC occa();

        occa.clk(my_clock);
        pe1.port(occa);
        pe2.port(occa);
        se1.port(occa);

        occa.set_address_range(&se1.port,0x100,0x500);
        occa.set_priority(&pe1.port, 2);
        occa.set_priority(&pe2.port, 5);
        sc_start(-1);
}
```



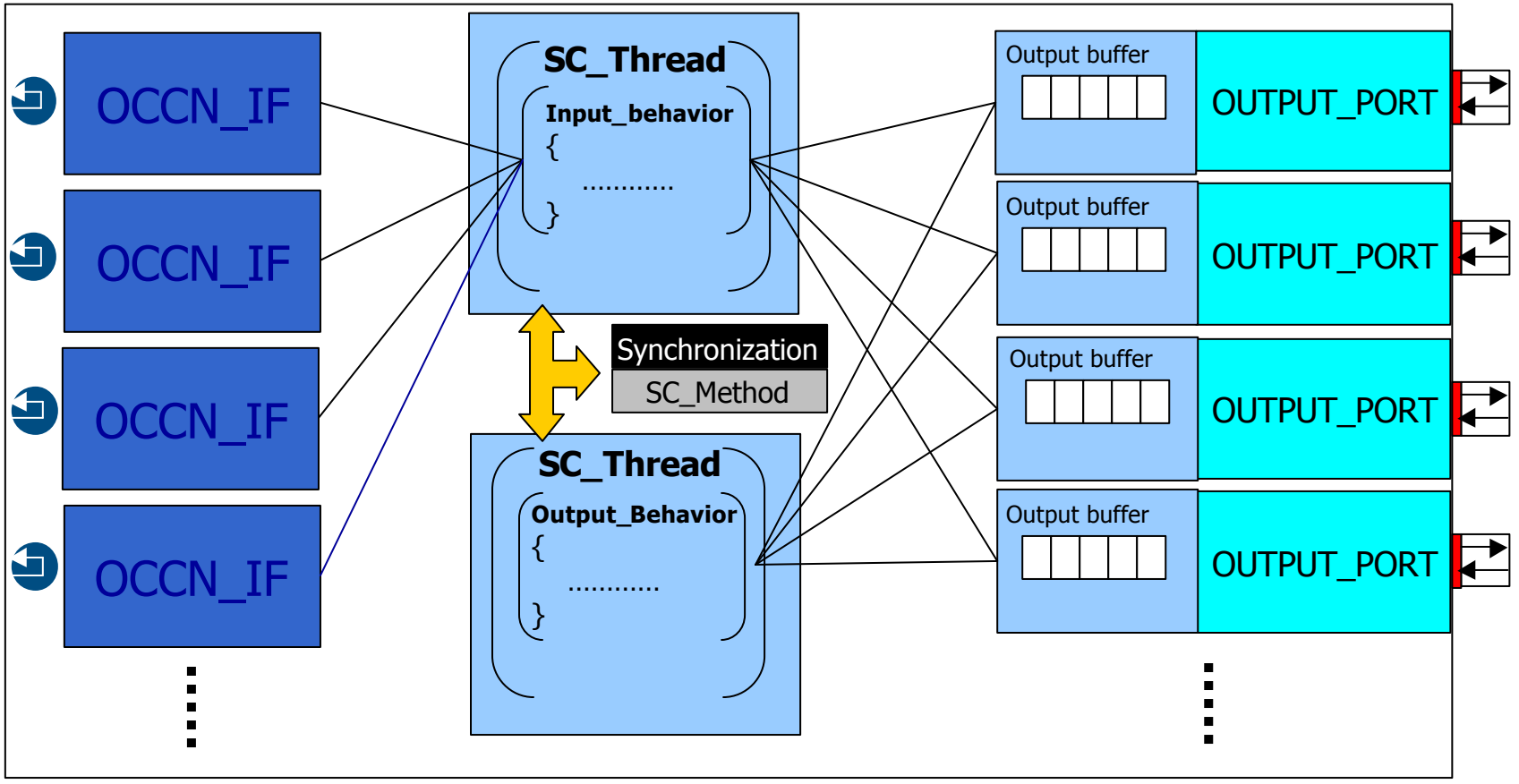
Performance measurement with Grace

- XY graph, XY charts, pie charts, polar, and fixed graphs.
- User-defined scaling, ticks, labels, symbols, line styles, fonts, colors.
- Merging, validation, cumulative average, curve fitting, regression, filtering, DFT/FFT, cross/auto-correlation, sorting, interpolation, integration, differentiation...
- Internal language, and dynamic module loading (C, Fortran, etc).
- Hardcopy support with PS, PDF, GIF and PNM formats.



Evolution for NoC

- OCCN: A NoC Router



Conclusion

- OCCN
  - based on SystemC methodology
  - open & flexible API
  - simulation speed-up
  - reusability
  - productivity
  - communication architecture exploration
- Public part -> http://occn.sourceforge.net