
# OccN

## *A Methodology for NoC*

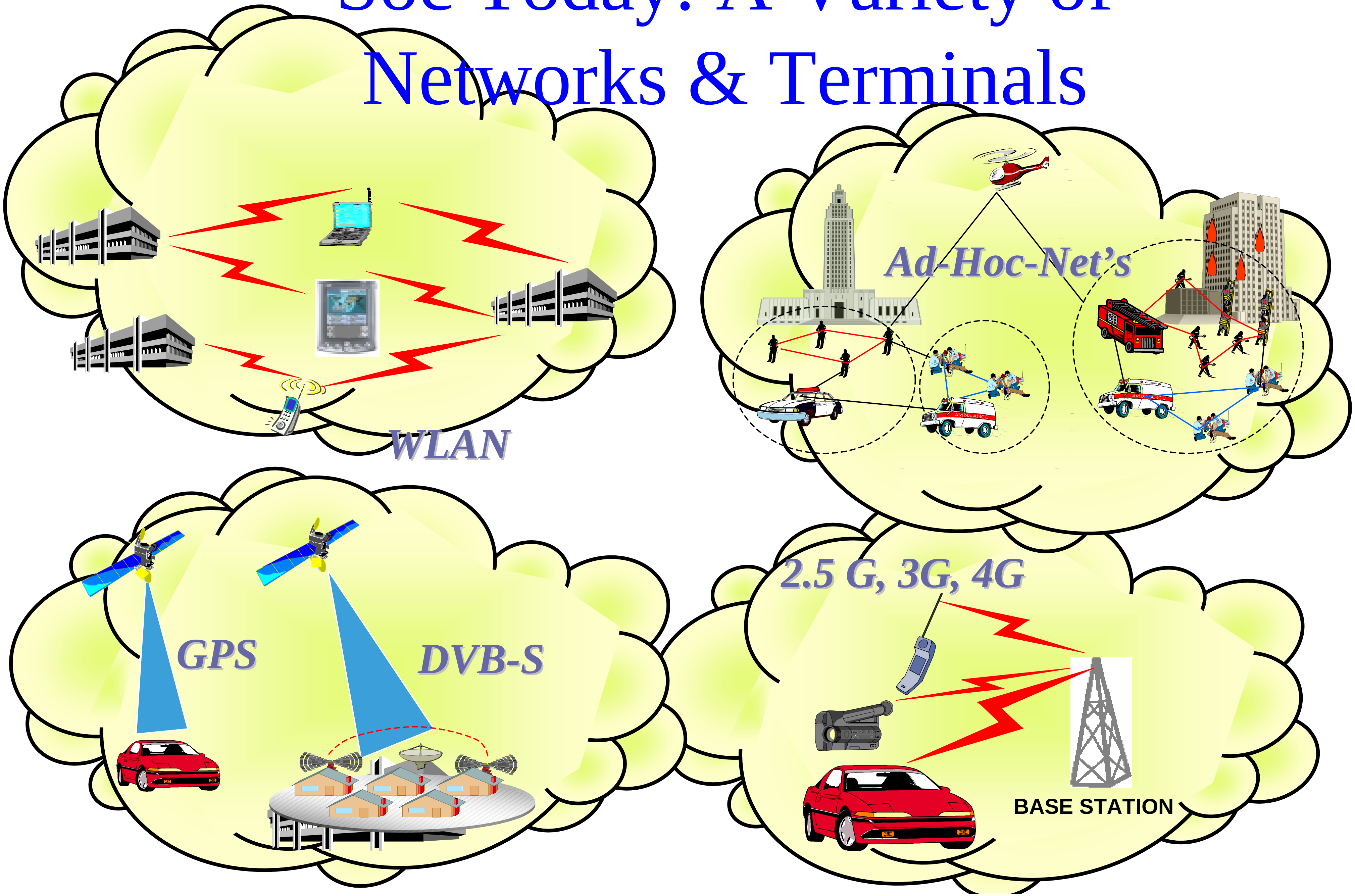
AST Grenoble

Marcello Coppola

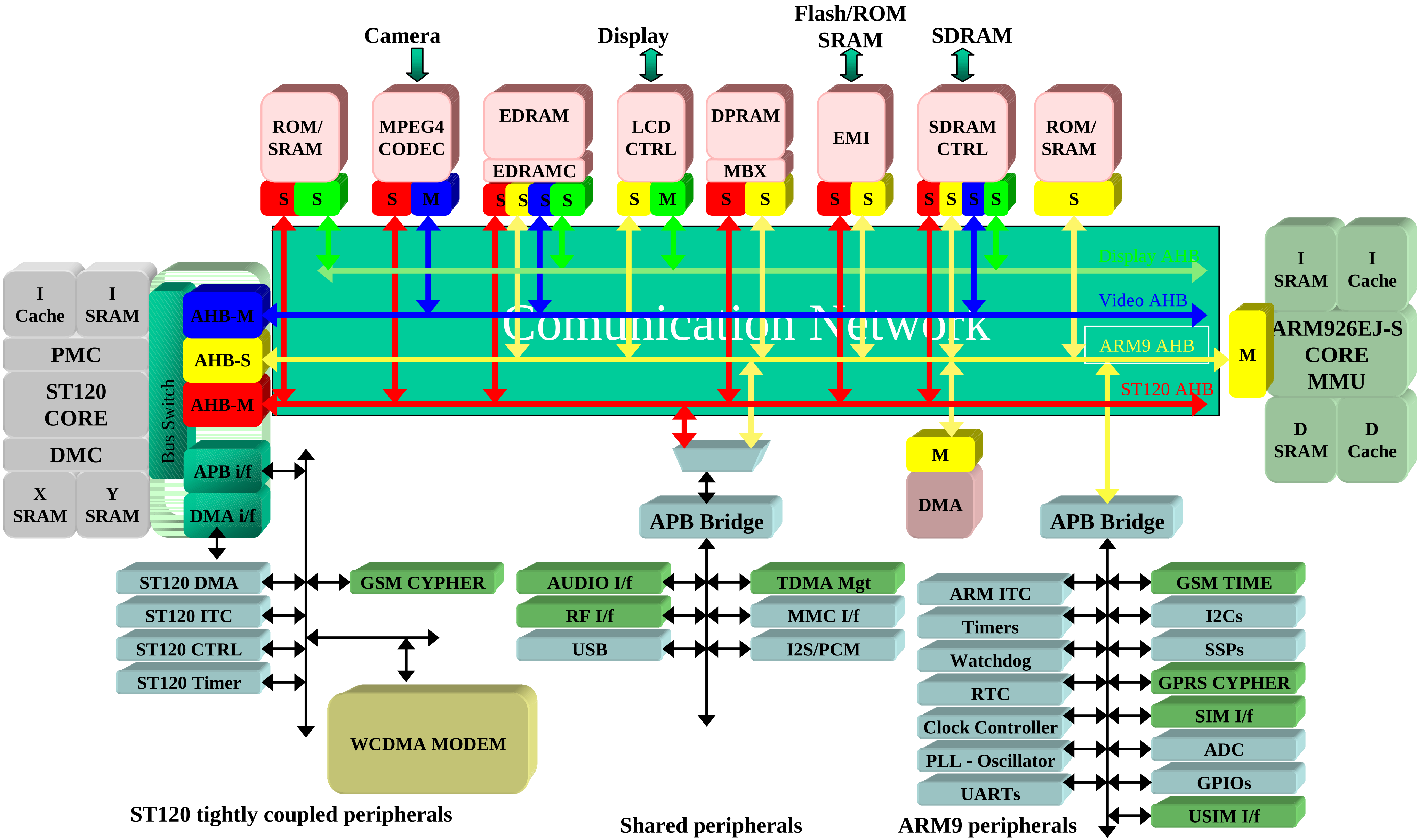
# Outline

- SoC today
- NoC
- OCCN
- Case study
- Conclusion

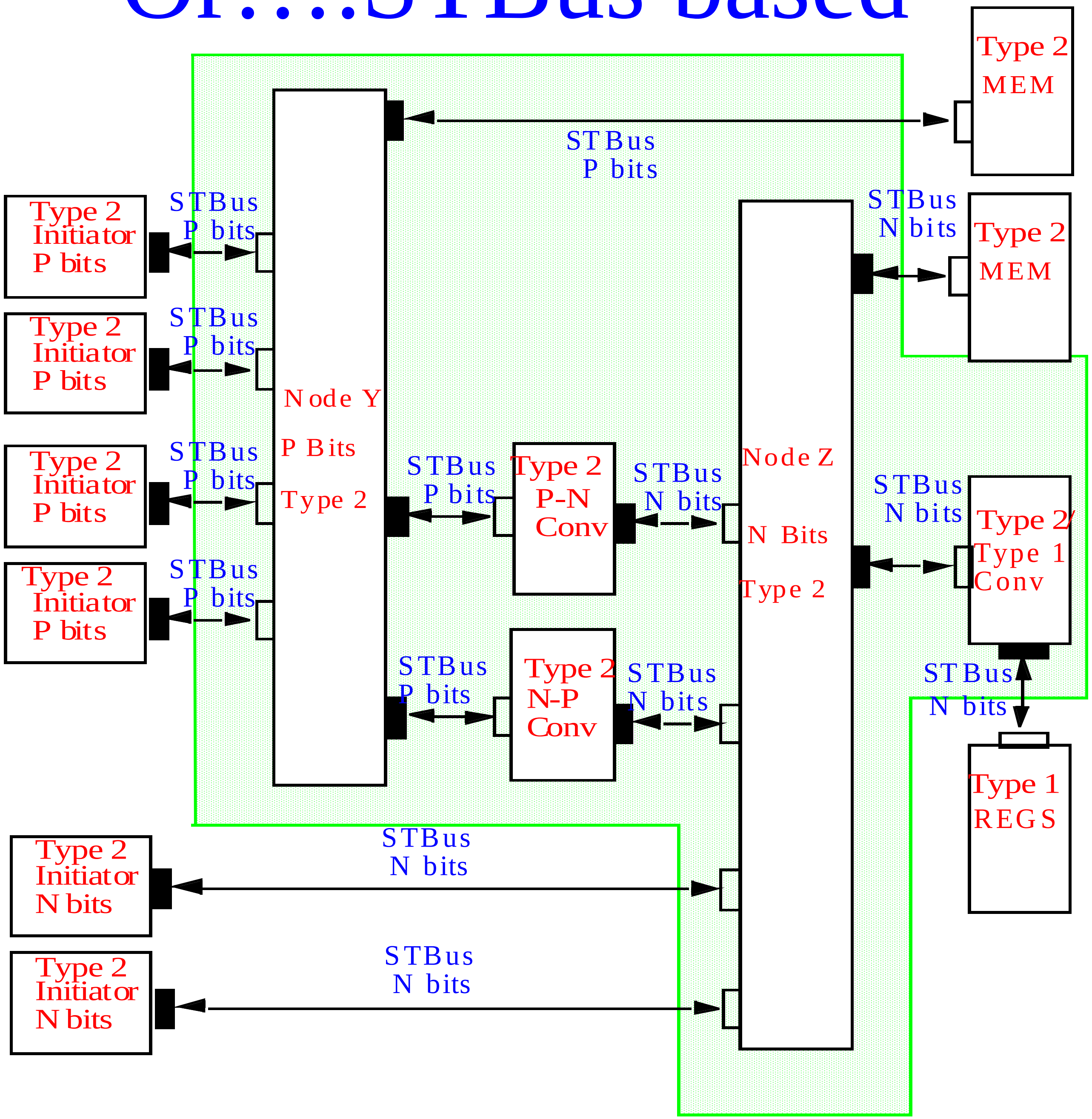
# Soc Today: A Variety of Networks & Terminals



# MP-SoC architecture: AMBA based



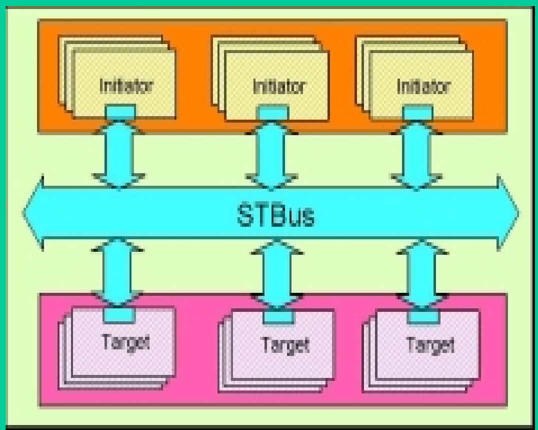
# Or….STBus based



# Architecture evolution: from SoC to NoC

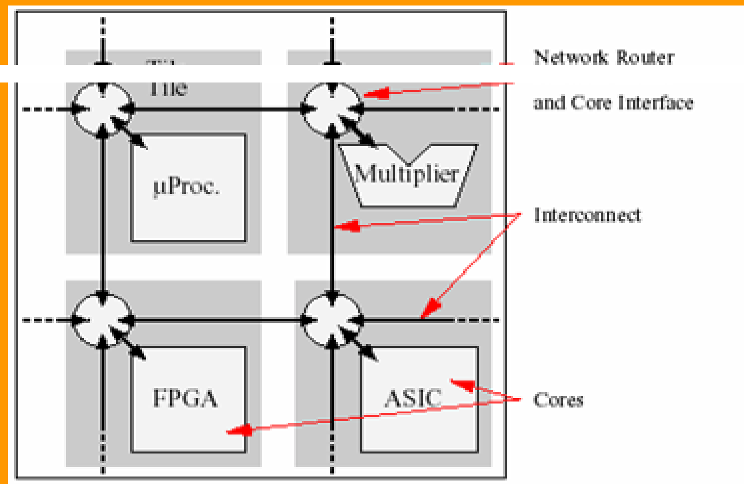
## System on a chip

Programmable computation
Hardwired interconnectivity
Partially distributed storage

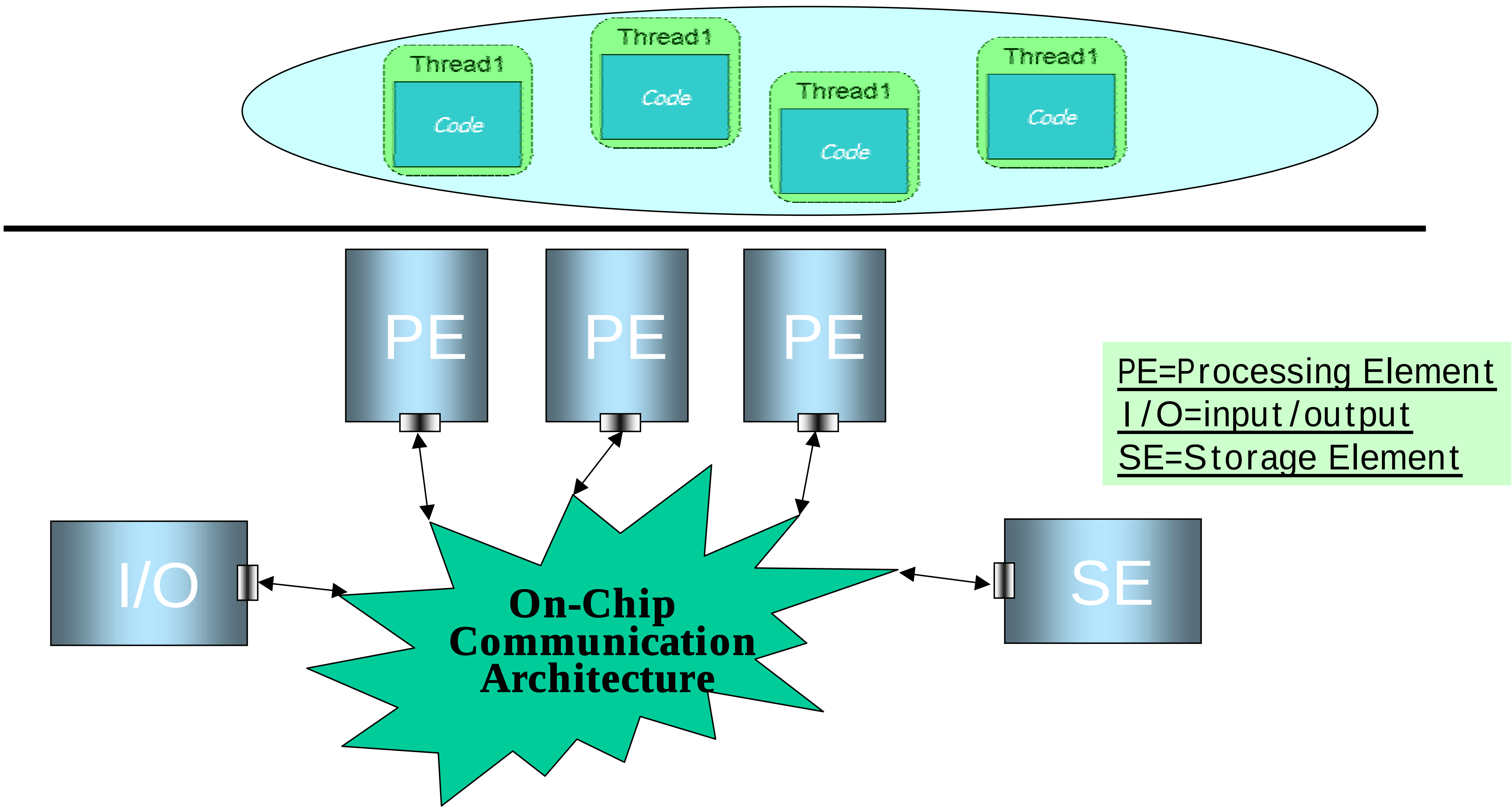



## Network on a chip

Programmable computation
Programmable interconnectivity
Fully distributed storage

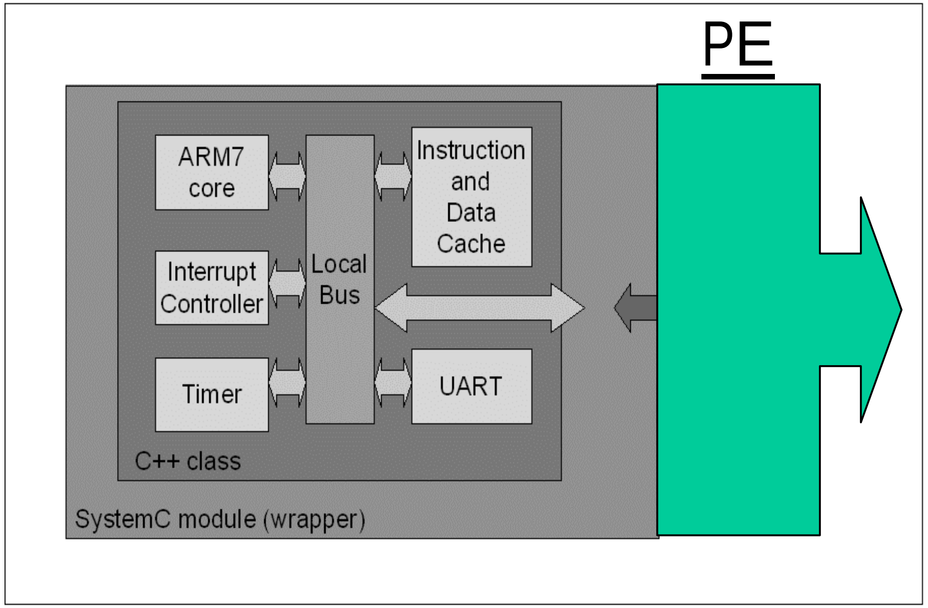


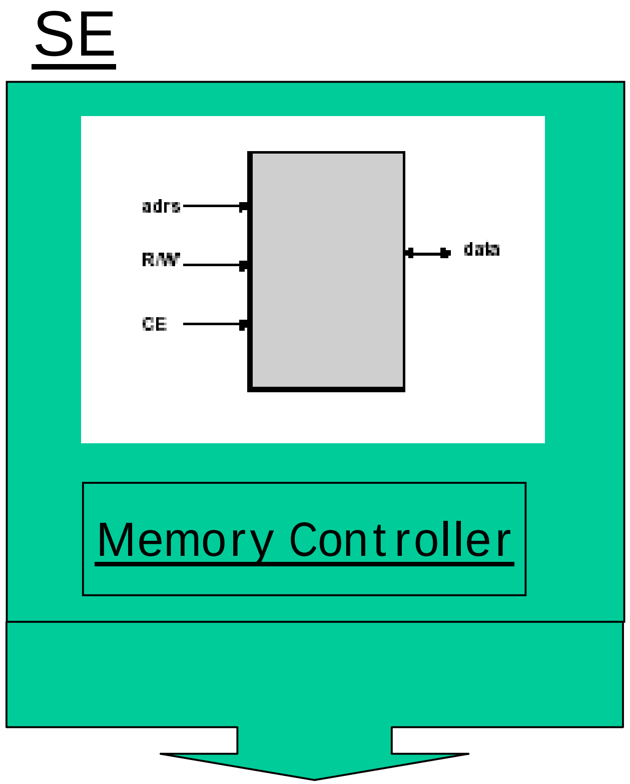
# NoC

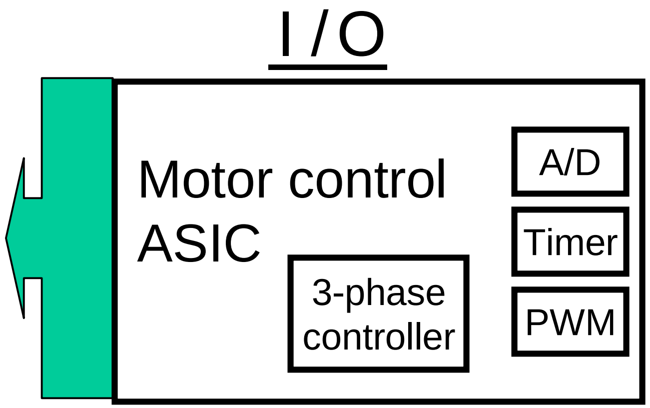


Thread1
Code
Thread1
Code
Thread1
Code
Thread1
Code
PE
PE
PE
I/O
On-Chip Communication Architecture
SE
PE=Processing Element
I/O=input/output
SE=Storage Element

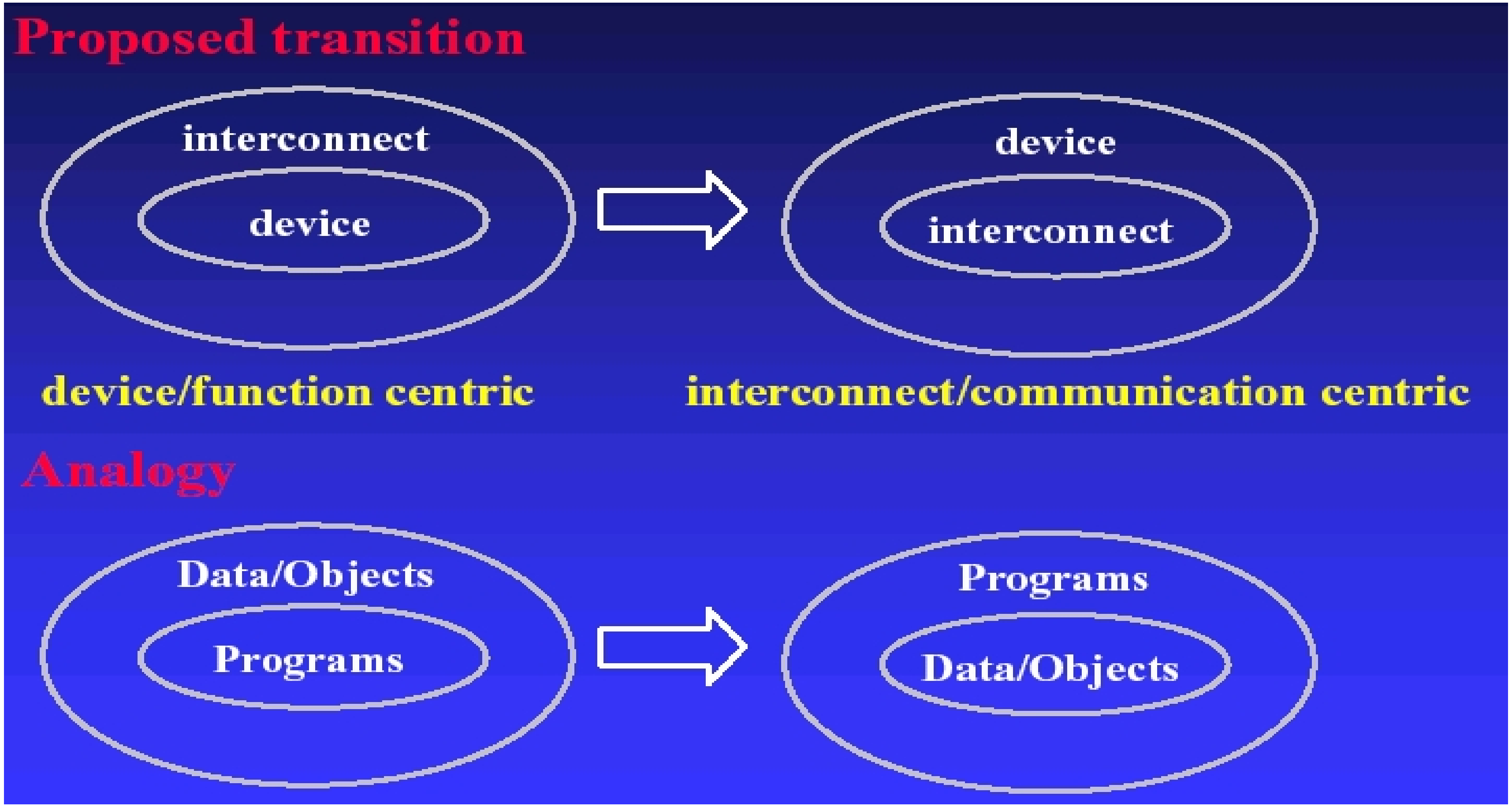
# PE,I/O,SE


PE
ARM7 core
Interrupt Controller
Timer
Local Bus
Instruction and Data Cache
UART
C++ class
SystemC module (wrapper)


SE
adrs
R/W
CE
data
Memory Controller

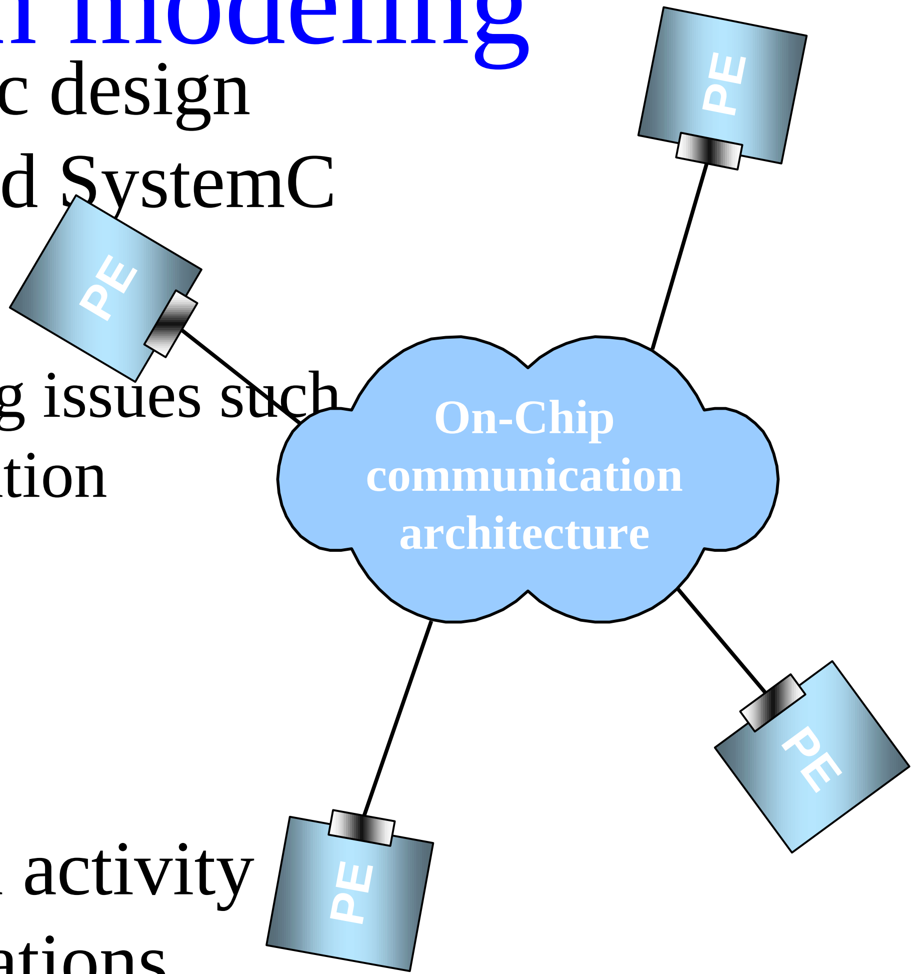

I/O
Motor control ASIC
A/D
Timer
PWM
3-phase controller

# Communication Centric Methodology

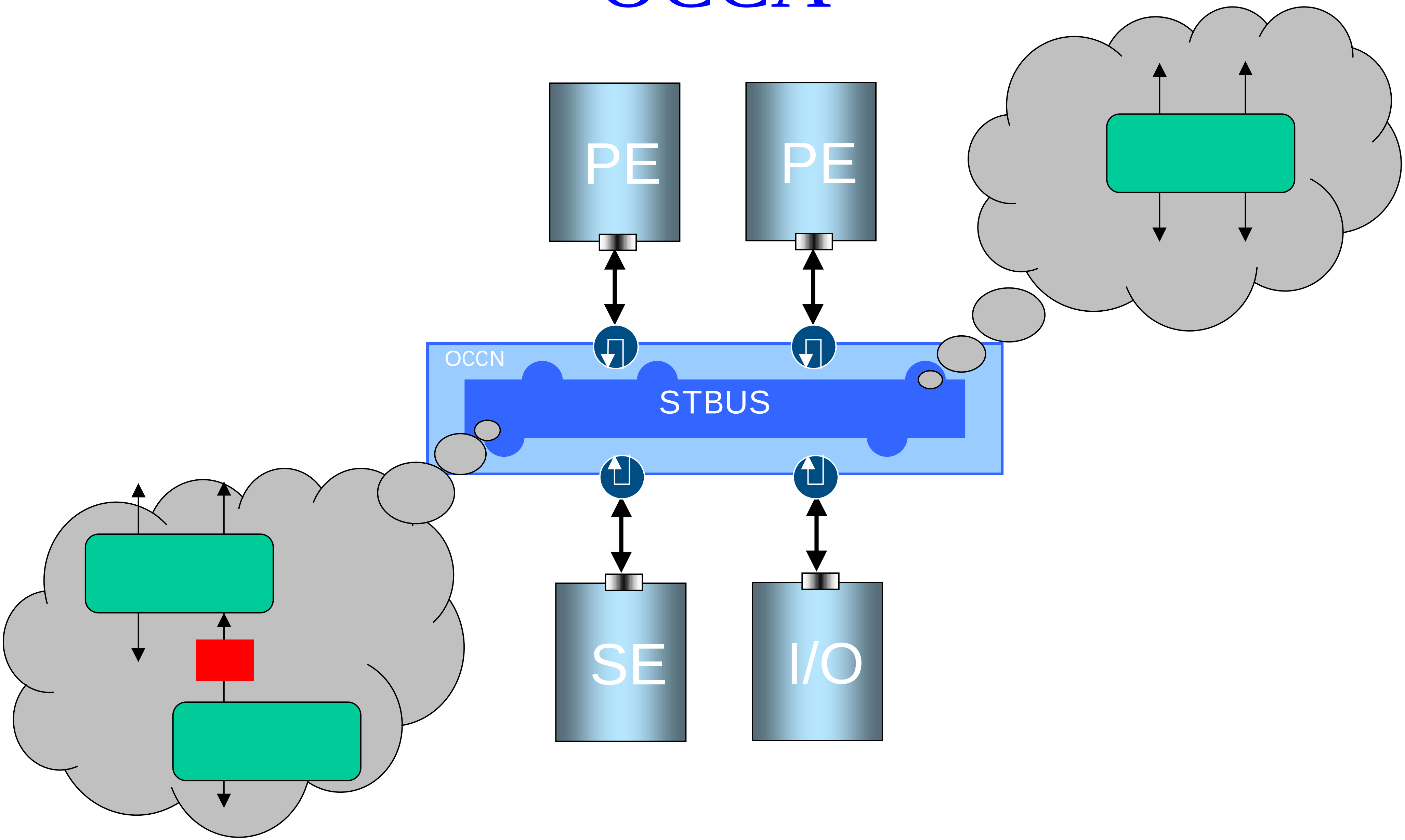


# OCCN : methodology for communication modeling

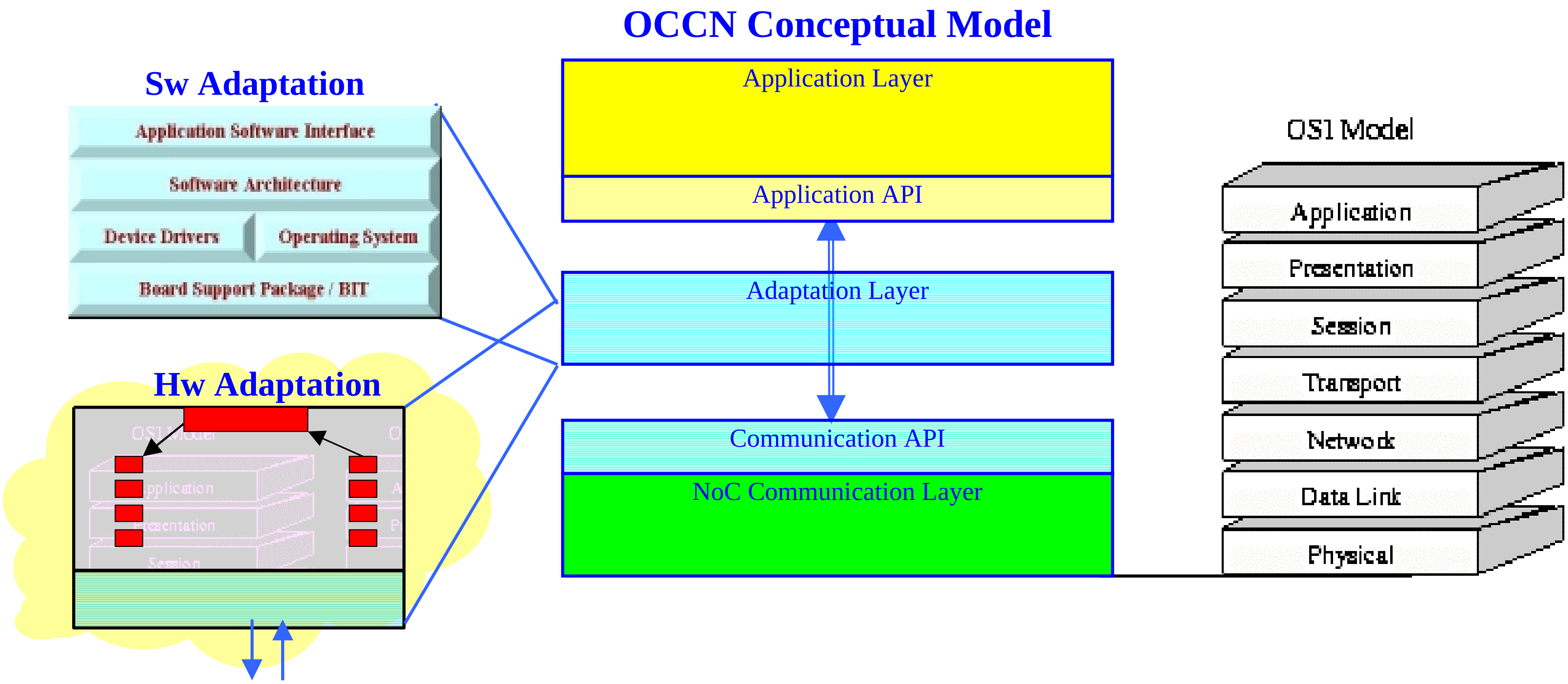
- Generic communication-centric design methodology based on C++ and SystemC
- OCCN addresses
  - high level performance modeling issues such speed, latency and power estimation
  - modeling productivity
  - model portability
  - simulation speed-up
- OCCN is an on-going research activity between several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



# OCCA

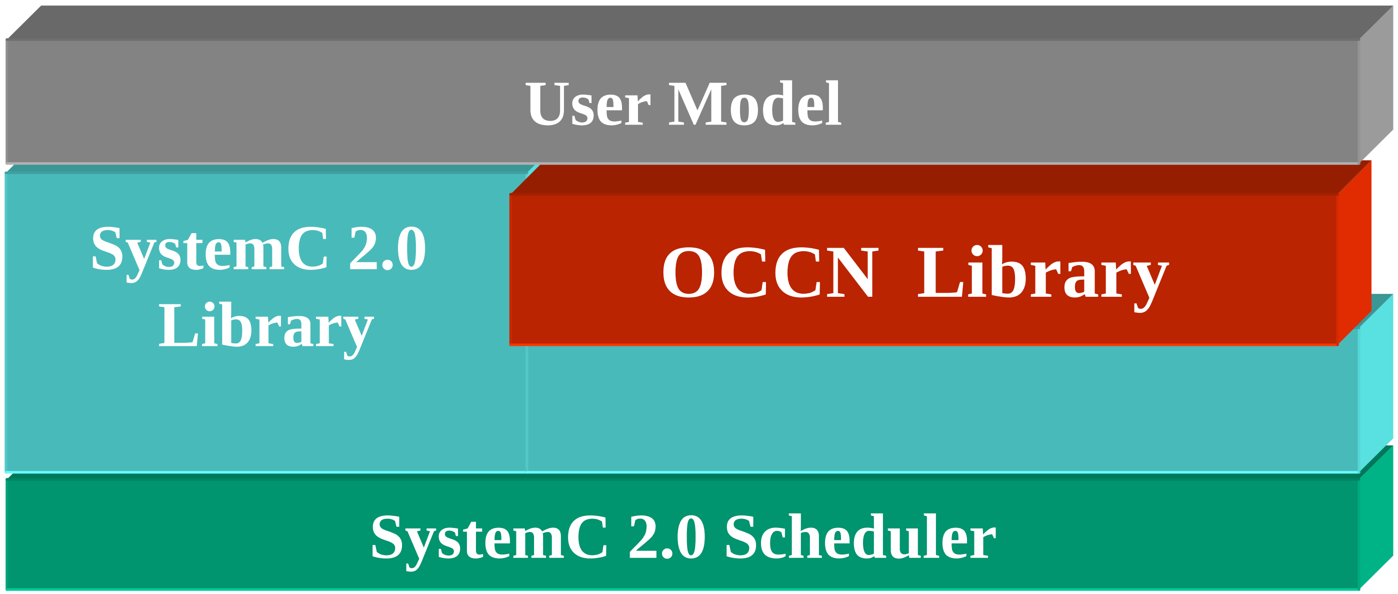


# OCCN Conceptual Model



# What is OCCN ?

OCCN aims at IC modeling, providing a real **object-oriented methodology** based on a **C++ library** and a fully documented design flow **based on SystemC 2.0**



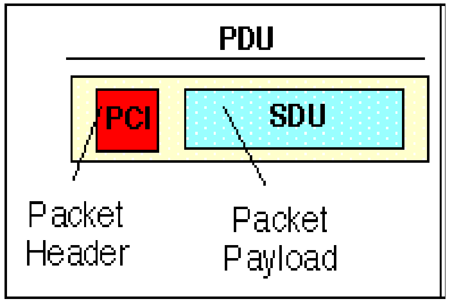
# OCCN Focus

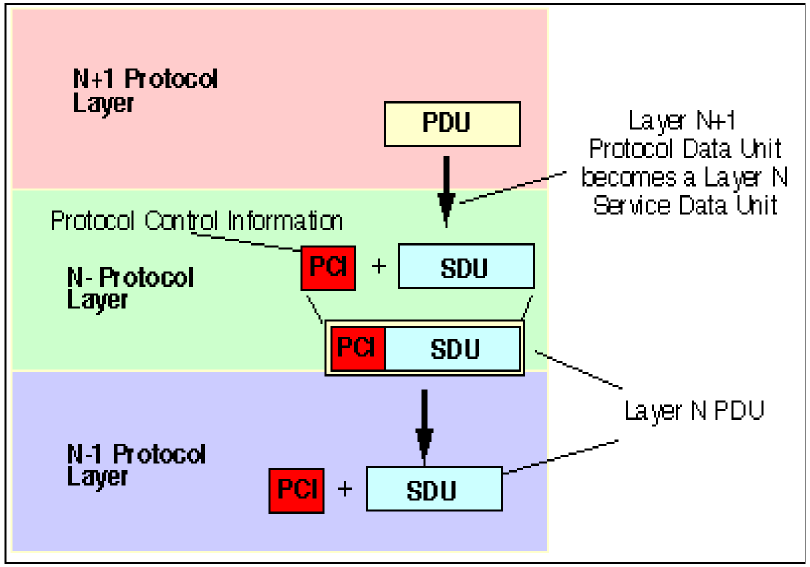
| | | |
|---|---|---|
| UML, SDL, MatLab ... | Algorithmic Level (AL)<br>Foundation: Function | Function-calls<br>Functional |
| SystemC | Programmer's View (PV)<br>Foundation: Memory Map | Bus generic<br>Architectural |
| | Programmer's View + Timing (PVT)<br>Foundation: Timed Protocol | Bus architecture<br>Timing approx. |
| | Cycle Callable (CC)<br>Foundation: Clock Edge | Word transfers<br>Cycle-accurate |
| VHDL, Verilog | RT Level (RT)<br>Foundation: Implementation | Signal/Bit<br>Cycle-accurate |

Source: ARM

# OCCN core: the PDU

- Protocol = syntax + semantics
  - syntax = PDU
  - semantics = how the PDU 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



# PDU Examples

| 8 bits |
|---|

```
Pdu<char> p1;
```

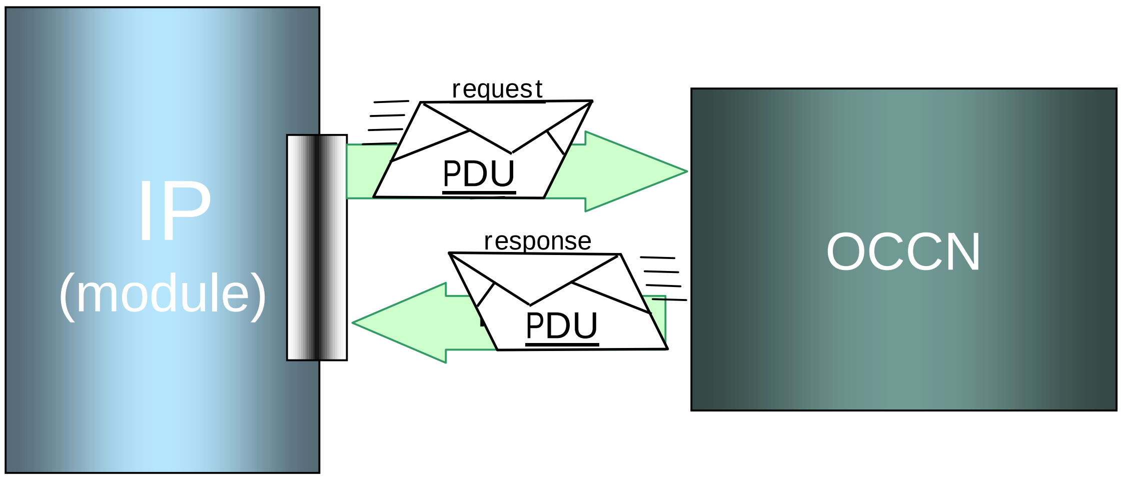
| P | T | Data |
|---|---|---|

```
Struct DSLINK_token {unsigned int P:1; unsigned int T:1};

Pdu<DSLINK_token,char> p2;


occn_hdr(pk1,P)=1;

pk1=‘a’;
```

# Generic representation of a connection

- Any connection of a module to the communication node (network) is based on 2 sets of PDU
  - Pdu<uint32,PCIRequest>
  - Pdu<uint32,PCIResponse>

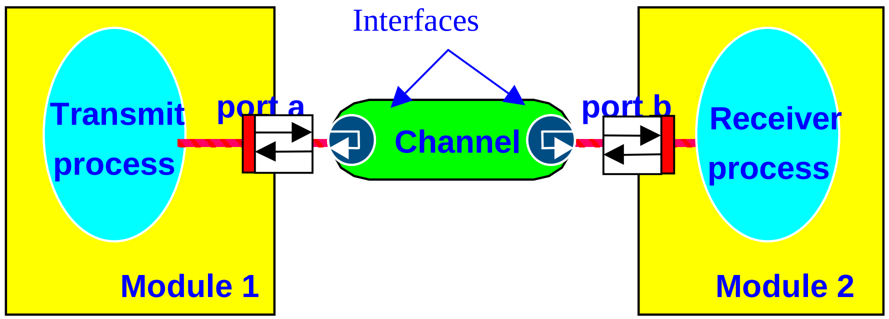
- The PCI sets are described thanks to C/C++ structures. They are defined according to the bus specification and thus are specific to a model. For instance it will be different for an AHB model and an STBUS model




```
Struct
{
  bool Request;
  unsigned char Opcode;
  bool Lock;
}
PCIControl
```

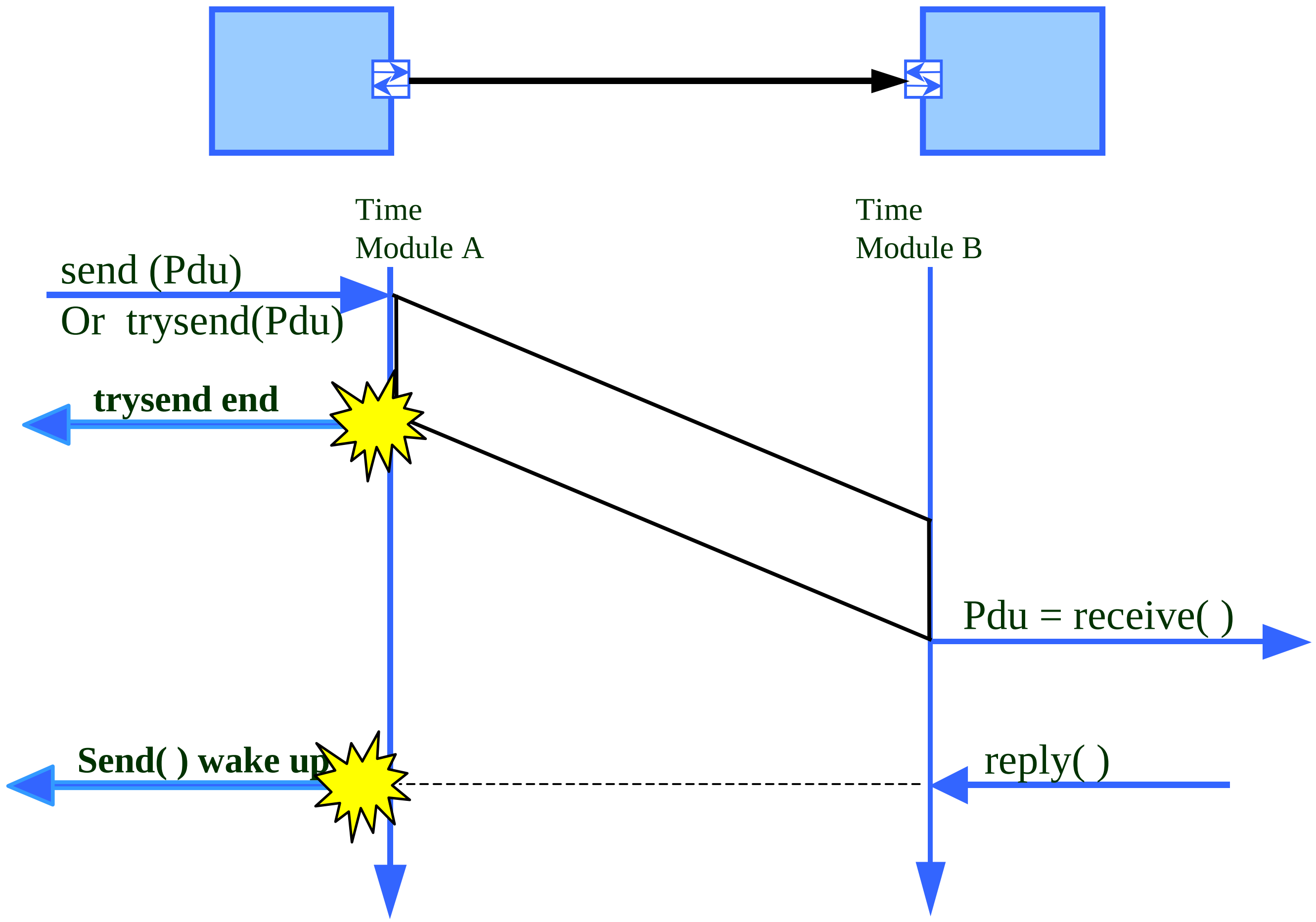
# OCCN: communication

- SystemC based
- Simple Message Passing API

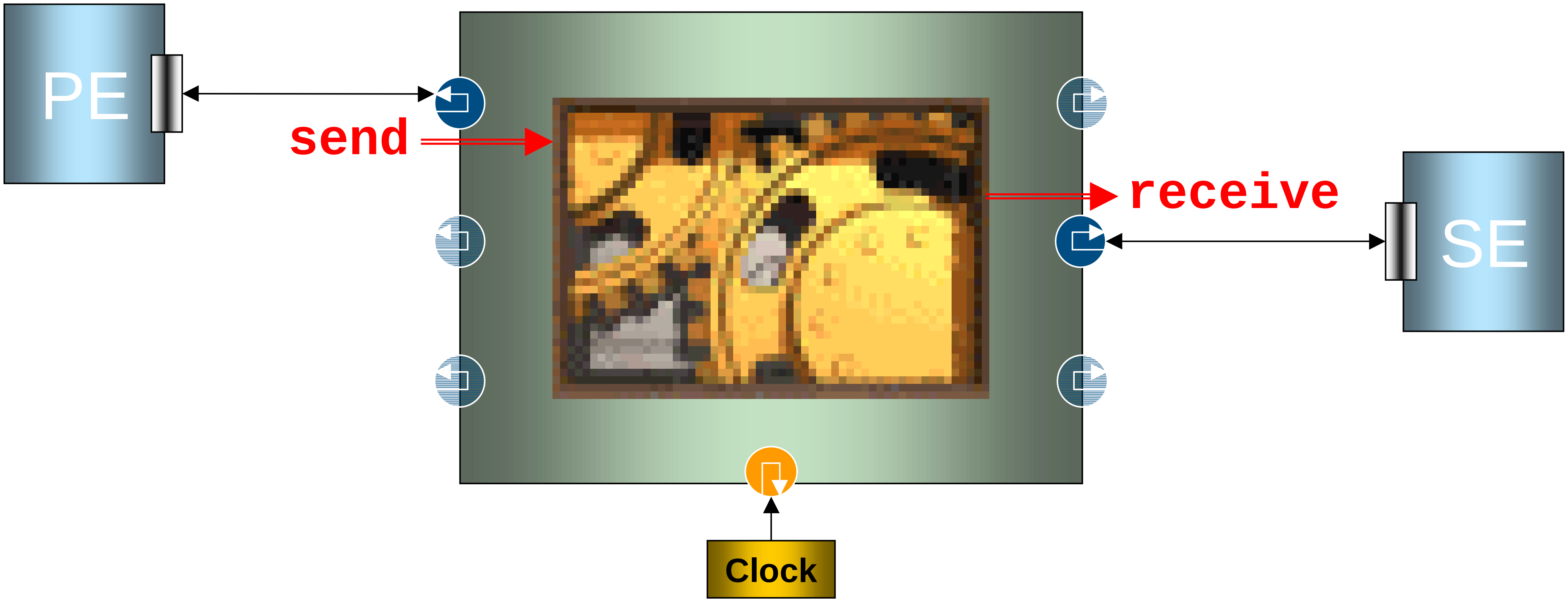



- `Pdu<…>* send(Pdu<…>* p, sc_time& time_out=-1);`
- `int trysend(Pdu<…>* p);`
- `Pdu<…>* receive(int ack_time, sc_time& time_out=-1);`
- `Pdu<…>* receive(sc_time& ack_time, sc_time& time_out=-1);`
- `Pdu<…>* receive(sc_time& time_out=-1);`
- `void reply(Pdu<…>* p=0);`

# OCCN core : API semantic



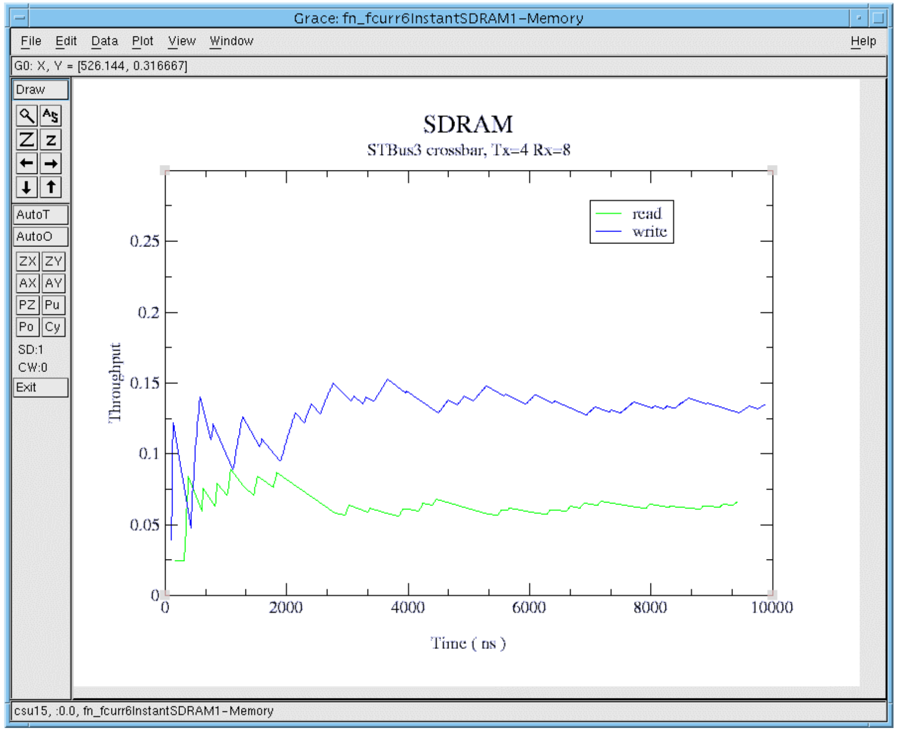
# OCCN core : protocol state machine centralized



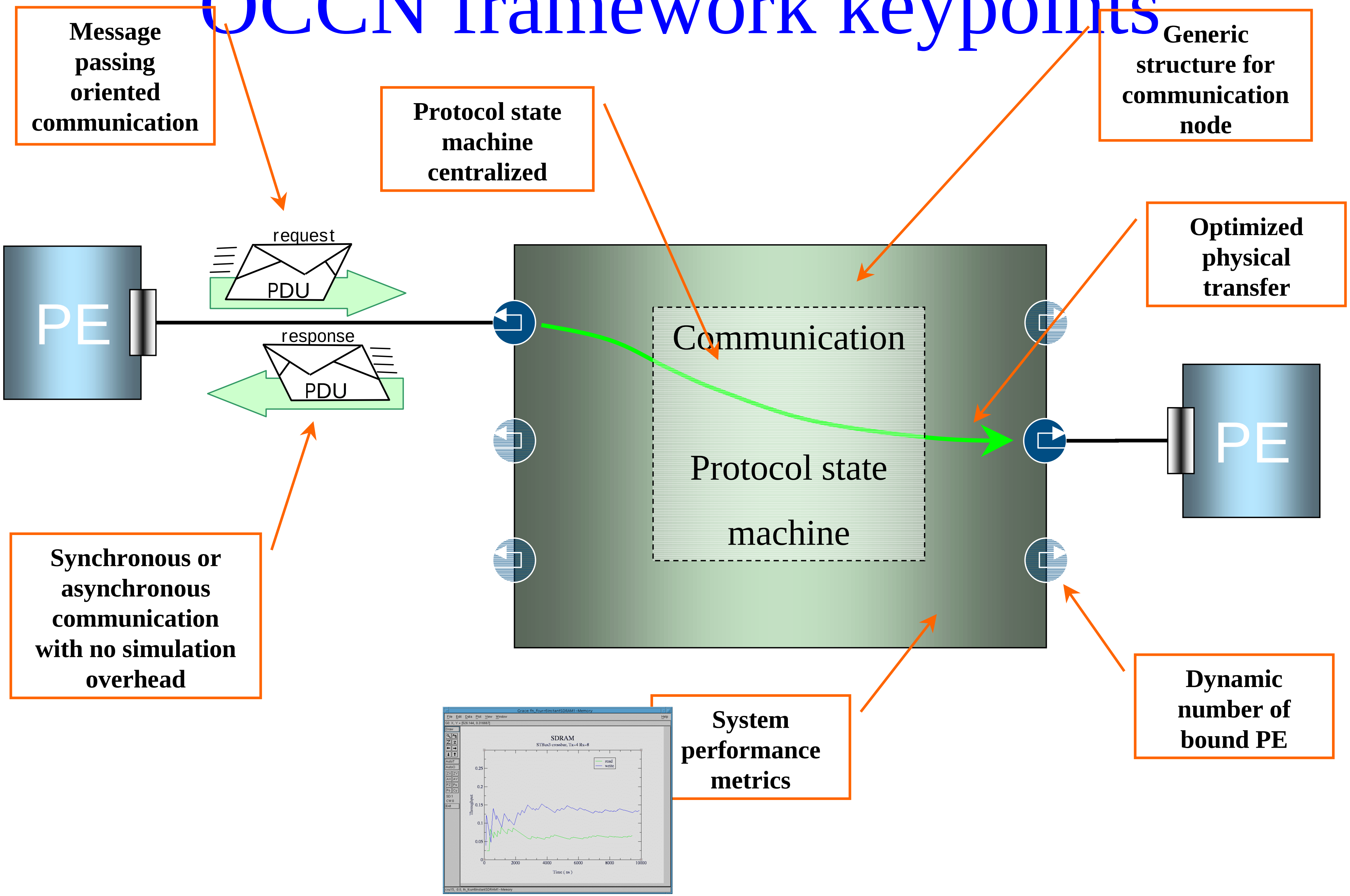

- allows synchronous and asynchronous communication modeling
- For synchronous com, PEs don’t need to be connected to the clock signal

# 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.

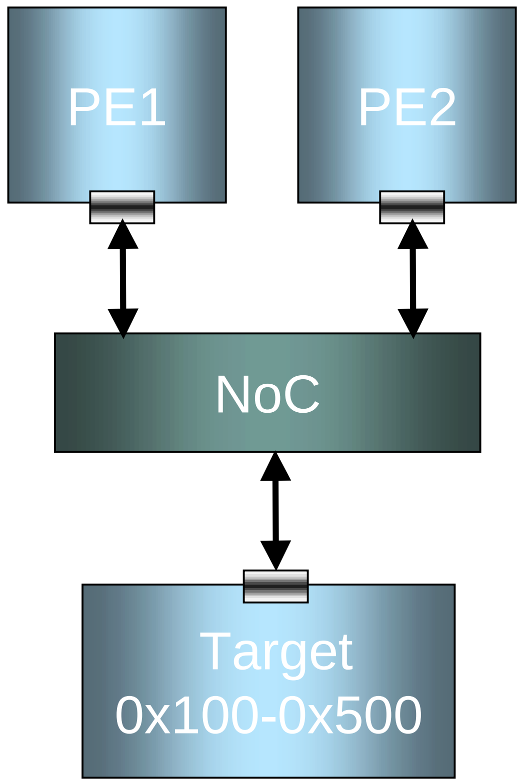


# OCCN framework keypoints



# MP SoC architecure

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


        occa.clk(my_clock);
        pe1.port(occa);
        pe2.port(occa);
        se.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);
}
```



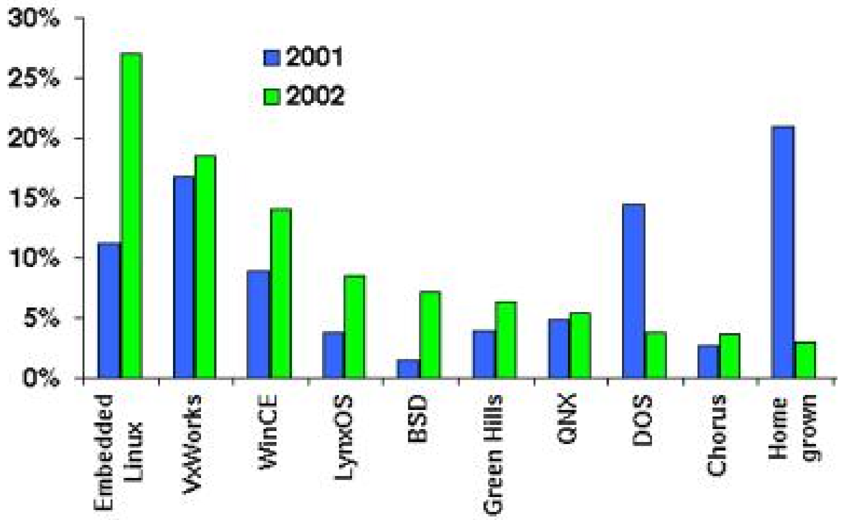
# OCCN: PE code example

```
#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 automatic generated

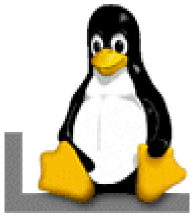
# A Wide Variety of Linux Devices

Embedded OS trends 2001–2002, sorted by 2002 expectation
(multiple selections permitted; top 10 for 2002 shown)



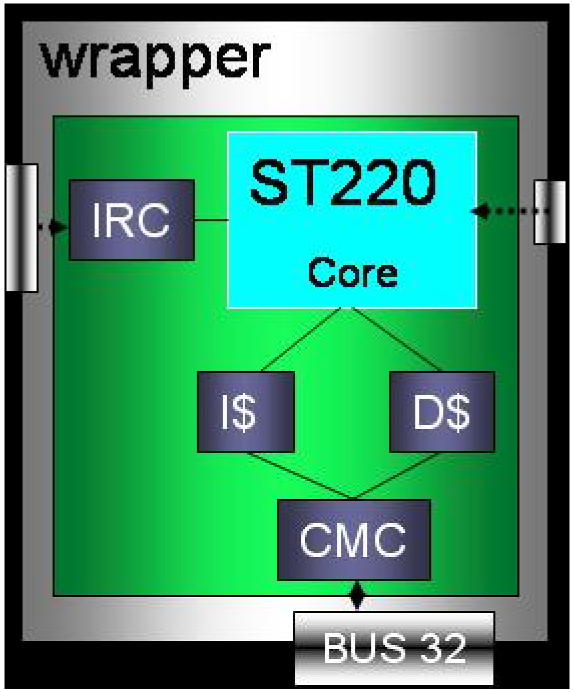

Source: Evans Data Corporation 2001 Embedded Systems Developer Survey

# Case study: Embedded Linux BSP for ST220

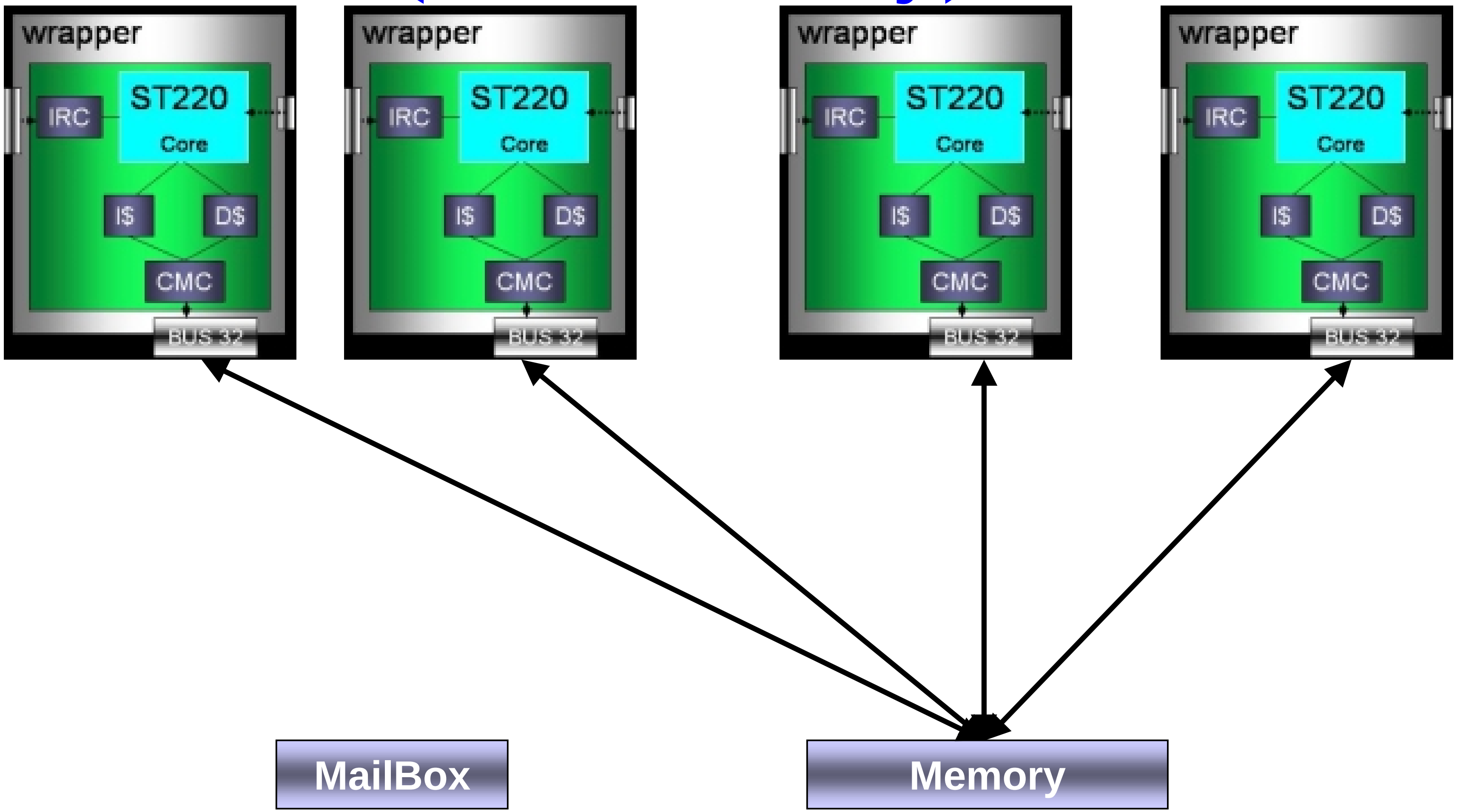


- Linux BSP
  - Kernel+drivers:
  - ~10.000.000 bundles (>15.000.000 instructions)
  - Elf Linux Boot load in ext memory (3.5Mb)
- PE architecture
  - D$/I$ Cache, Timer, Intrp Ctrl, Reg., etc.
- Simulation environment
  - SystemC + OCCN + Models: compilation -O3
  - Linux version 2.4.7-10 (Red Hat 7.1 2.96-98) with Pentium 4 a 2.4 GHz, 512 Kb cache, 1 Gb RAM

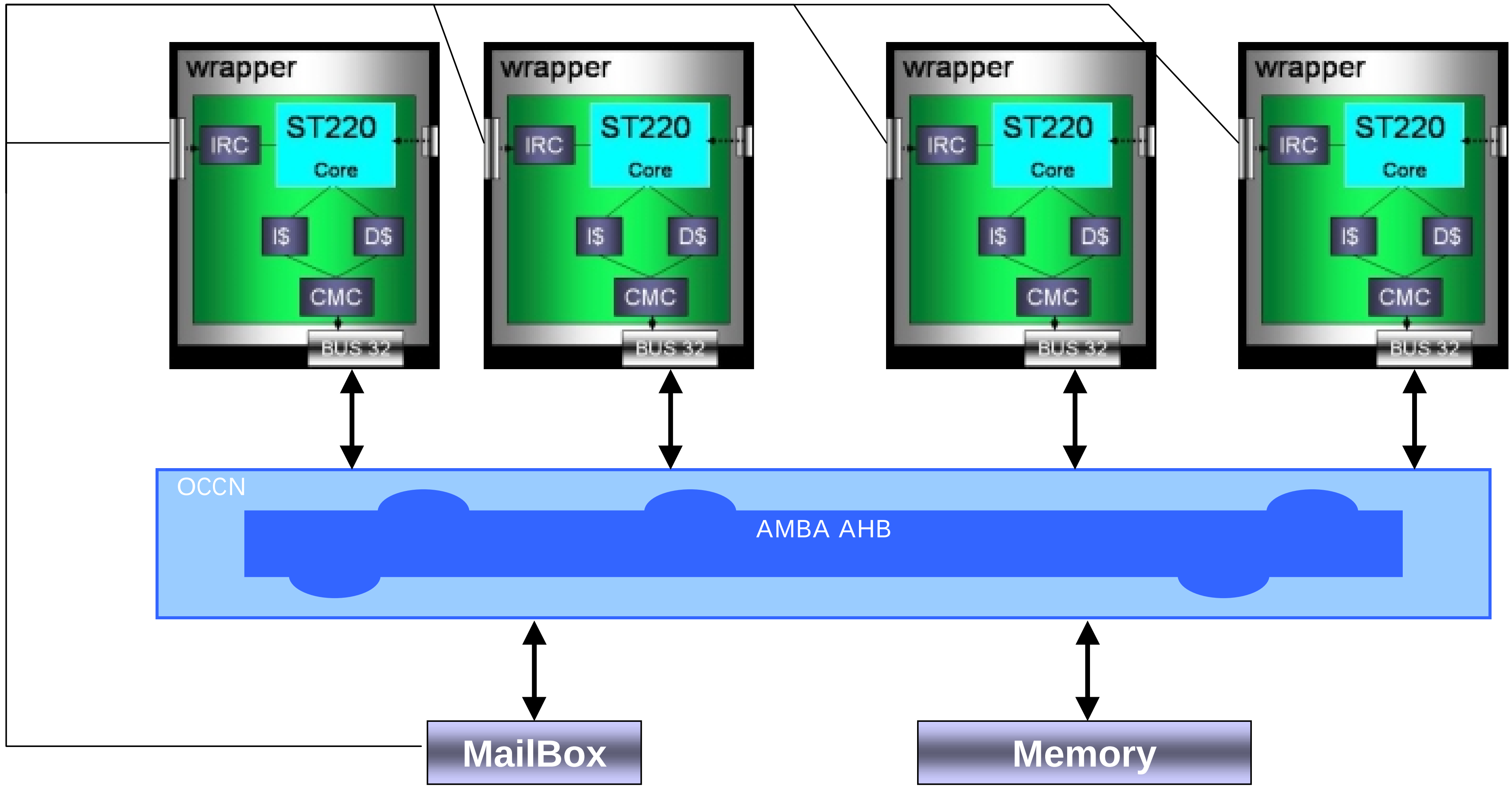
# Configuration 1: ISS Standalone/SystemC



# Configuration 2: No bus-ExtMemory (no wait/delay)



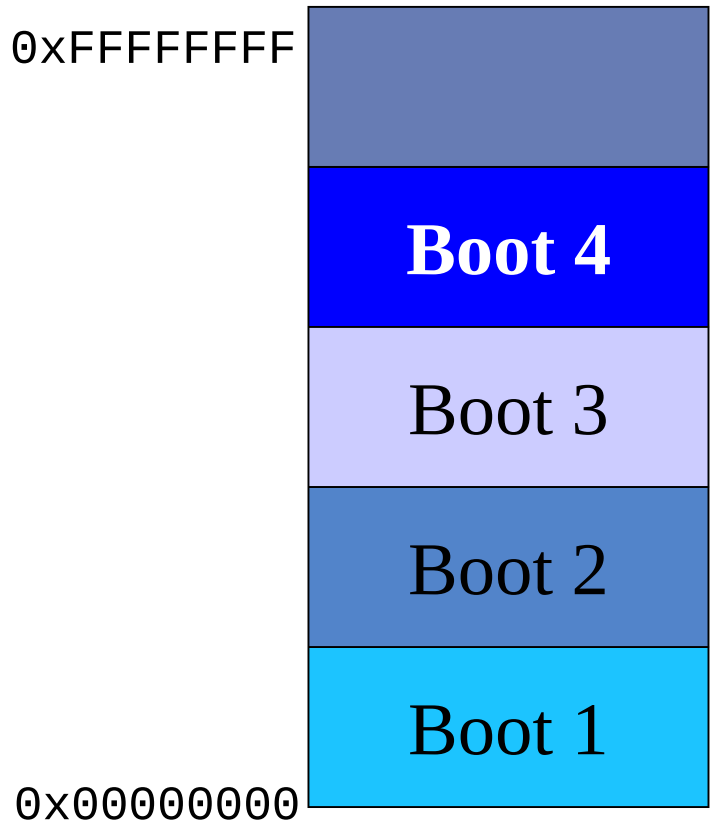
# Configuration 3: NoC Platform



# Linux Boot: One ST220

| | IPS | SEC |
|---|---|---|
| ISS (StandAlone) | 1304000 | 8.2 |
| ISS (systemC) | 633000 | 17.1 |
| No bus EXTMEM (no wait) | 610000 | 17.8 |
| No bus EXTMEM (delay 10ns) | 577000 | 18.8 |
| STANDARD BUS | 640000 | 17.0 |
| AHB BUS | 344000 | 31.5 |

# Memory Map for 4 ST220



# Linux Boot: for ST220

| | IPS | SEC |
| --- | --- | --- |
| ISS (systemC) | 137000 | 78.8 |
| No bus EXTMEM (no wait) | 138000 | 78.4 |
| No bus EXTMEM (delay 10ns) | 130000 | 83.3 |
| STANDARD BUS | 135000 | 79.9 |
| AHB BUS | 98000 | 110.0 |

# Conclusion 1/2

- OCCN
  - based on SystemC methodology
  - open & flexible API
  - simulation speed-up
  - reusability
  - productivity
  - communication architecure exploration
- Similar work: Gigascale Silicon Research Center (GSRC) effort Princeton University: MESCAL Project

  Modern Embedded Systems Compilers Architectures and Languages

  Princeton and UC Berkeley

# Conclusion 2/2

- Research Activity funded in Medea
- Public part **http://occn.sourceforge.net**

Univ. of Bologna

Univ. of Roma

Univ. of Ancona

# Thank You and …