A Model Based Approach to Design Applications for Network Processor

S. Afsharian[3], A.Bertolino[1], G.De Angelis[1], P.Iovanna[3], R.Mirandola[2]

[1]Istituto di Scienza e Tecnologie
dell'Informazione "Alessandro Faedo" CNR
Via G. Moruzzi 1, I--56124 Pisa, Italy
{antonia.bertolino,guglielmo.deangelis}@isti.cnr.it

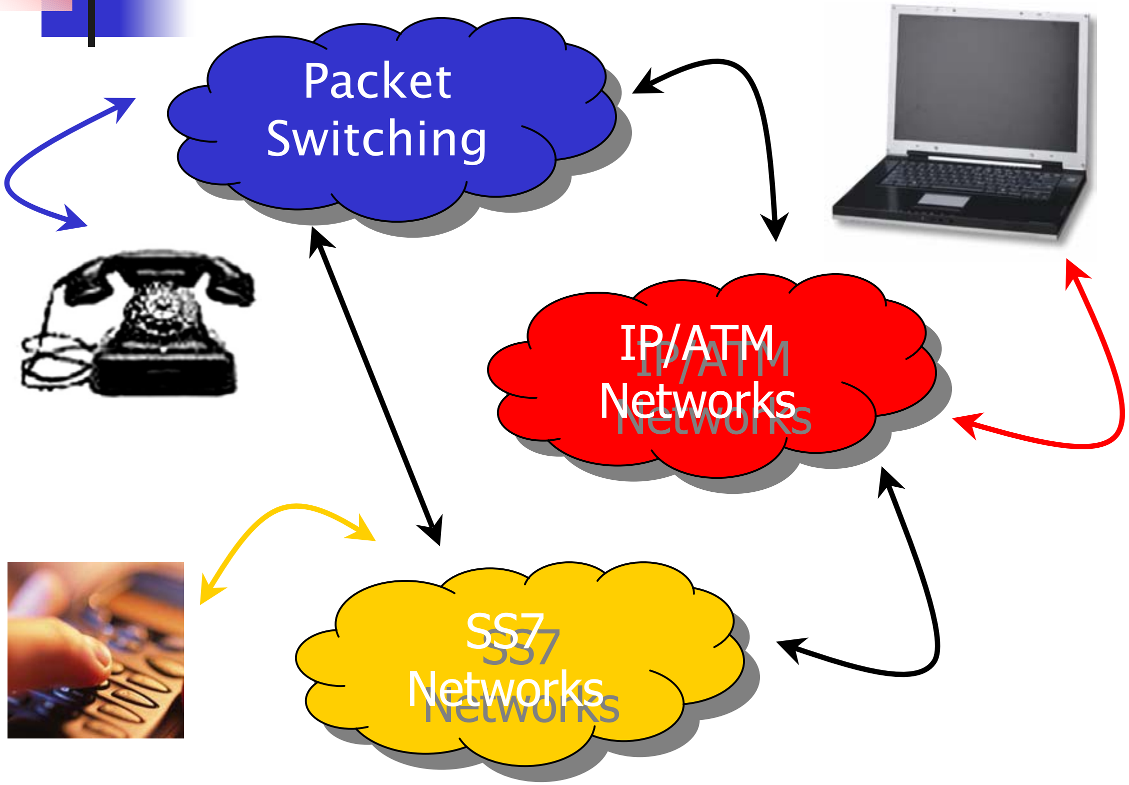
[2]Dip. di Informatica,Sistemi e Produzione
Università di Roma "Tor Vergata" 00113 Roma, Italy
raffaela@info.uniroma2.it

[3]Ericsson Lab Italy
Via Anagnina 203, 00040 Roma - Italy
{sharareh.afsharian,paola.iovanna}@ericsson.com

Roadmap

- Scenario
- Network Processor Overview
- Model Based Development
- MBD Applied to NPs
- Future Works
- Conclusions

Scenario



Packet Switching

IP/ATM Networks

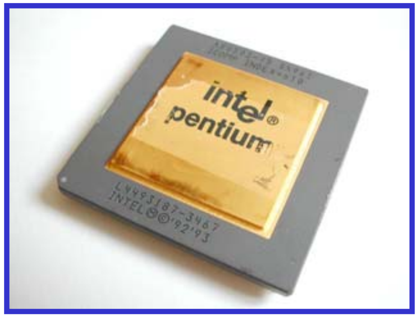
SS7 Networks

- Network Trends
  - Increased network traffic
  - Voice/data convergence
  - Rapid introduction of new technologies/standards

- Network devices are growing as a class of embedded systems

Different Solutions

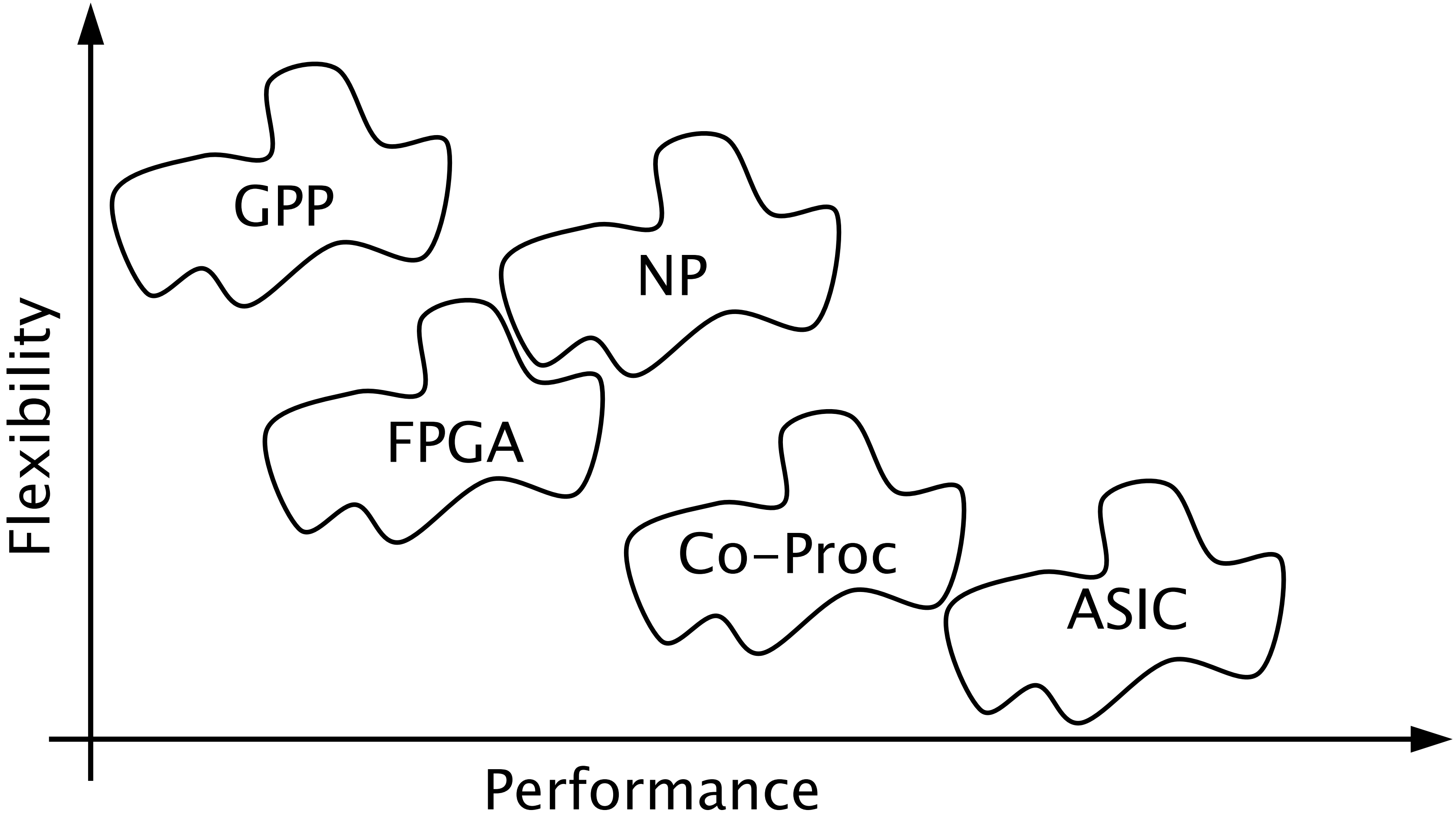
- ASIC
- ASIP
- Co-Processor
- FPGA
- GPP



What is a Network Processor?

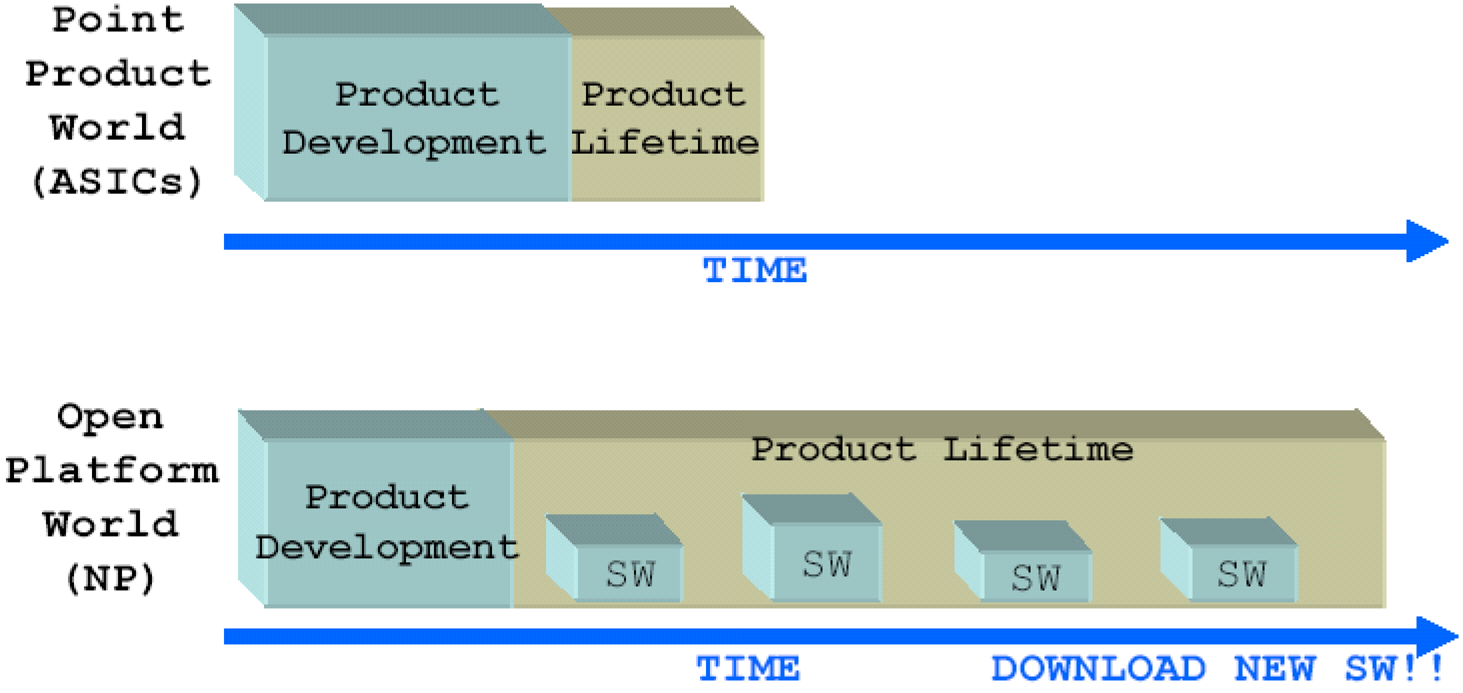
- It is an instruction set processor for network applications.
- It enables software implementations of key communications functions at hardware speeds.
- The main NP functions are:
  - Header classification
  - Deep packet analysis
  - Packet Processing
  - Policing and statistics
  - Traffic management

Relations Among Solutions

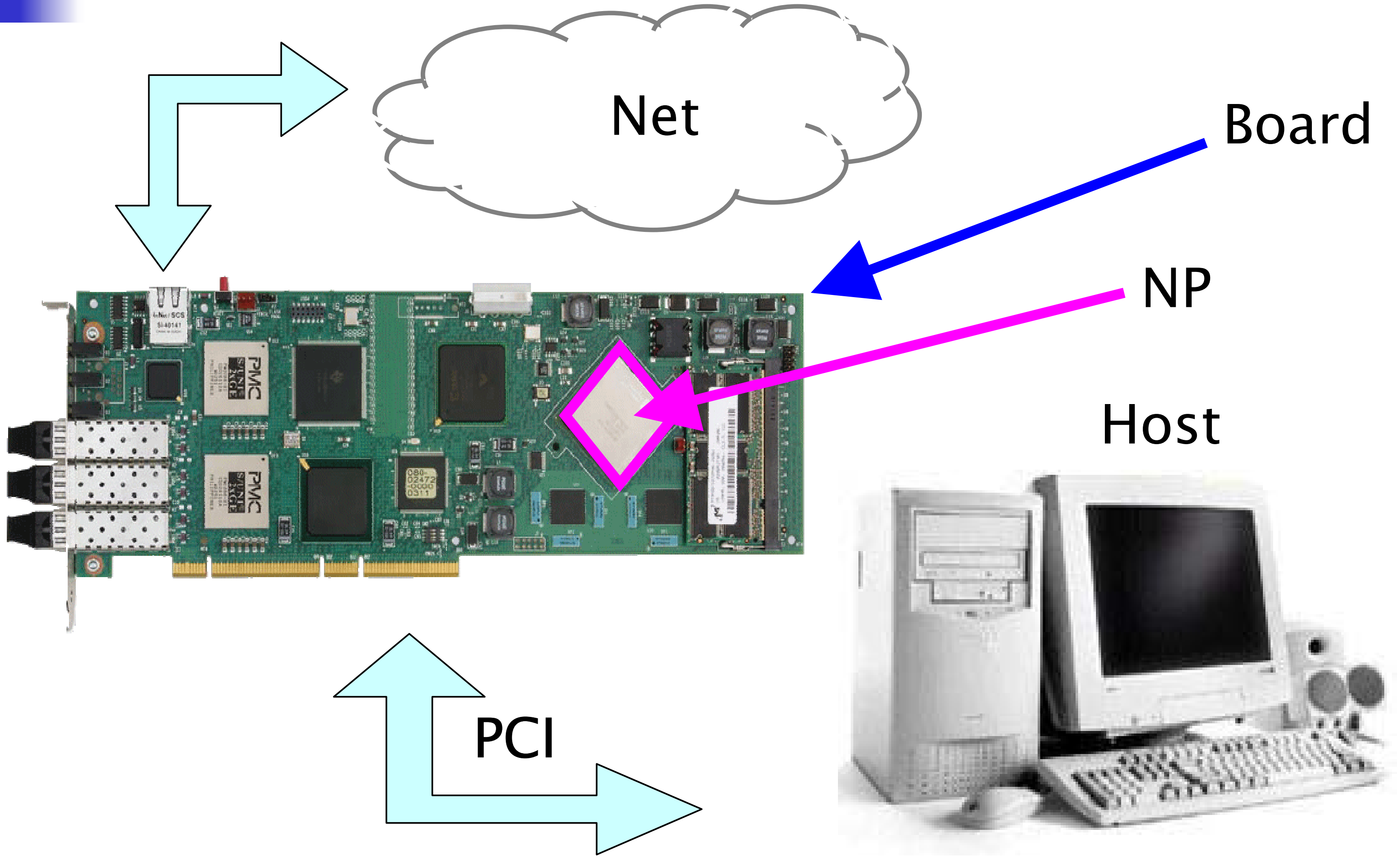


ASIC vs. NP

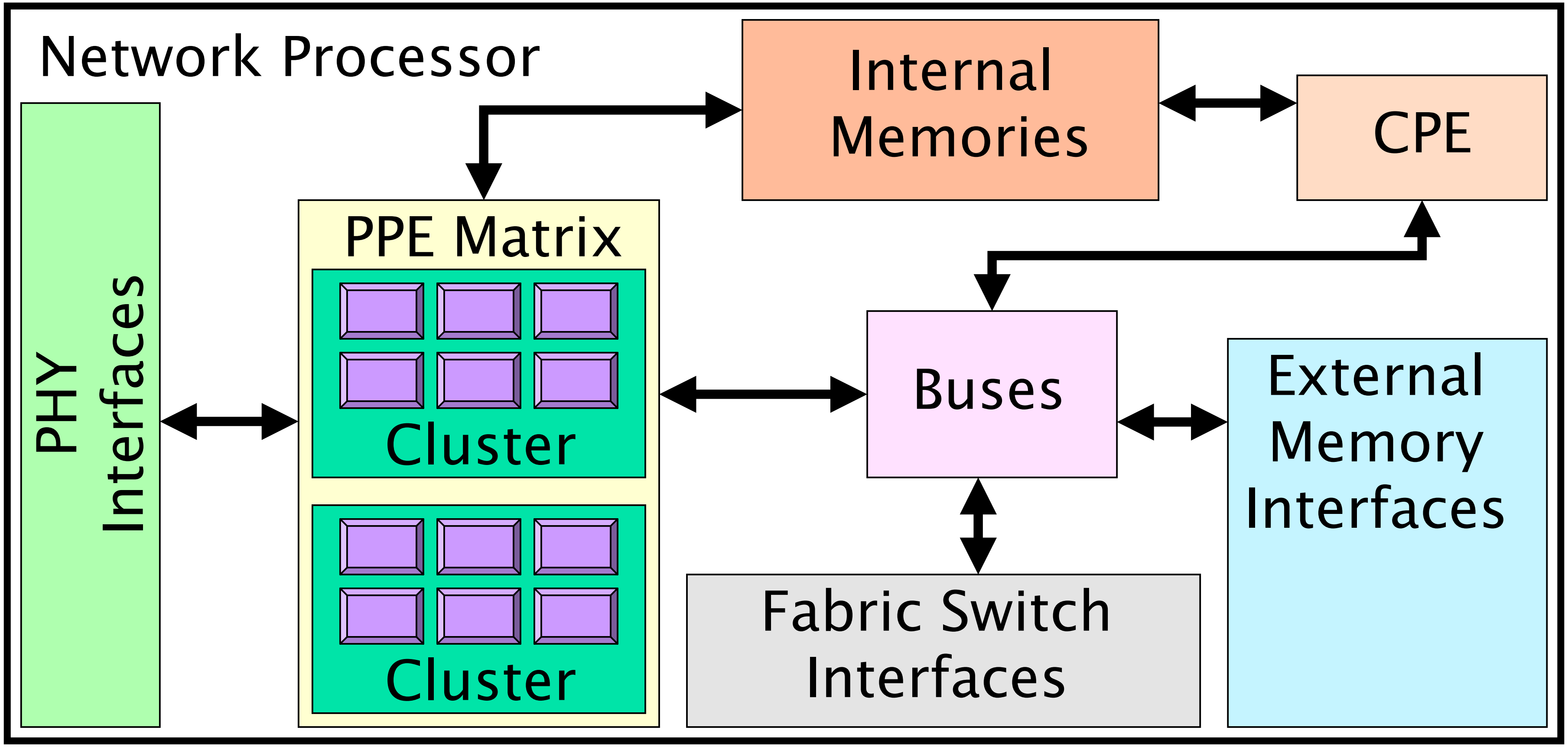
Time to Market and Time in Market



NP vs. Outdoor World



A Generic Network Processor



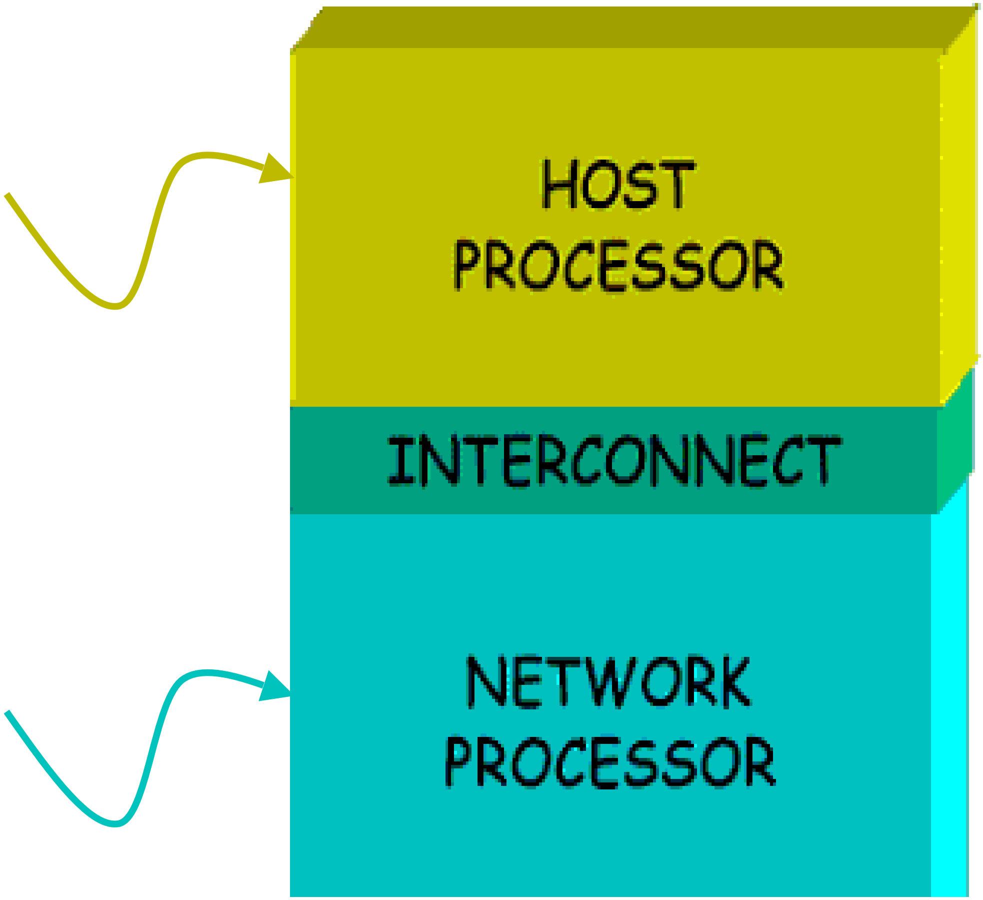
NP Philosophy

Control Plane :

- complex algorithms
- unusual functions
- control tasks

Data Plane :

- simple algorithms
- usual functions
- data manage tasks



Programming an NP

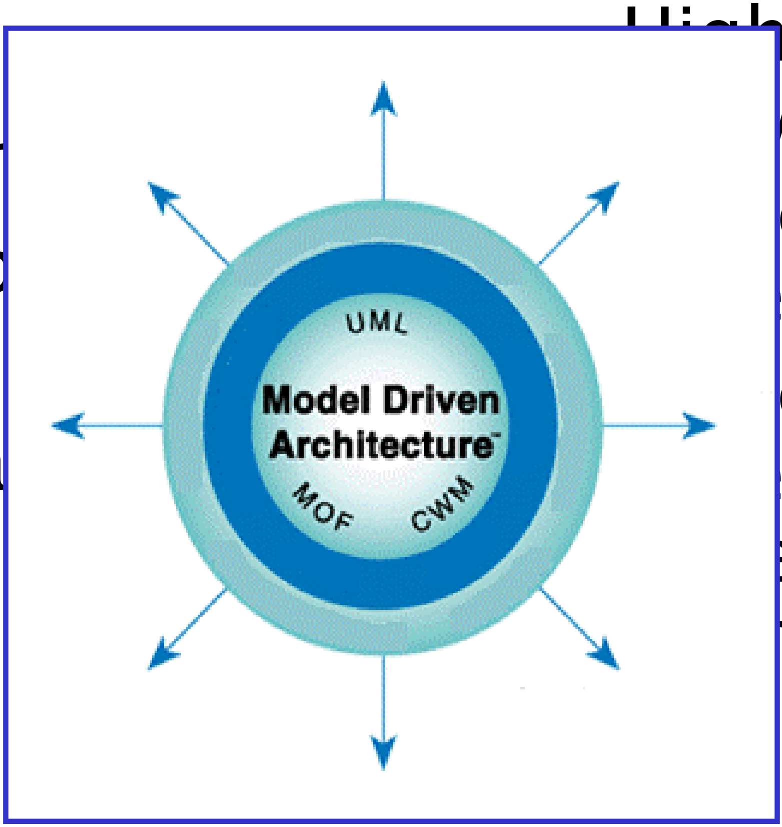
- Typical languages approaches are used for programming network processors.
- Imperative Paradigm :
    - The C language or an its variant :
        - CPE
        - PPE (some cases)
    - Assembly approach :
        - PPE
- Functional or 4*th* generation programming languages.

Summarizing

GAINS

… BUT

- Faster
- Cheap ASIC.
- Increa



Highly difficult ware design.

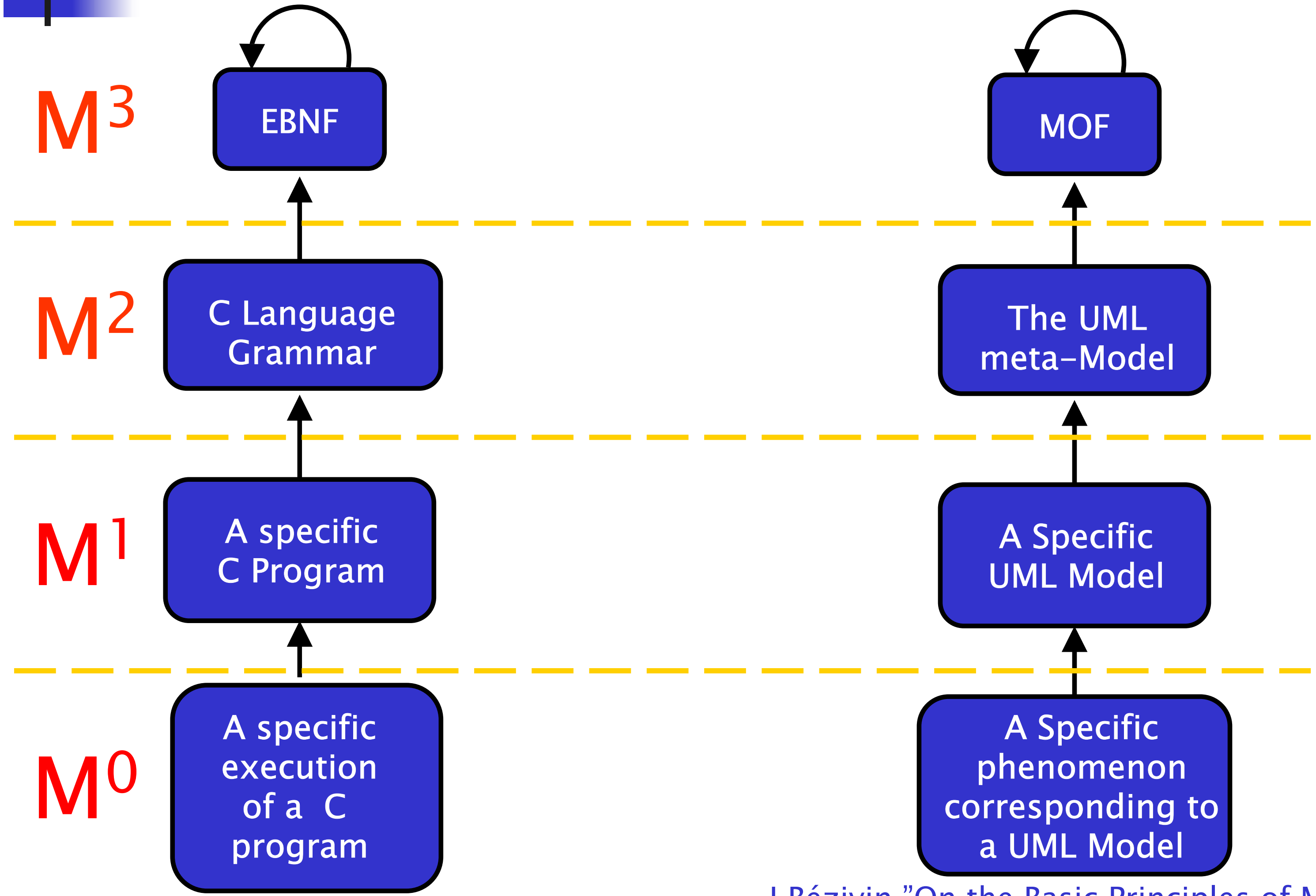
euse among rent solution.

euse among rent erations netimes).

Model Based Development

- MBD is an approach to software development in which the primary artefacts of development are models instead of software.
- MBD does not see everything at once.
- MBD uses representation that can be useful for the objective of the study at the given stage.

Abstract Syntax Systems Compared

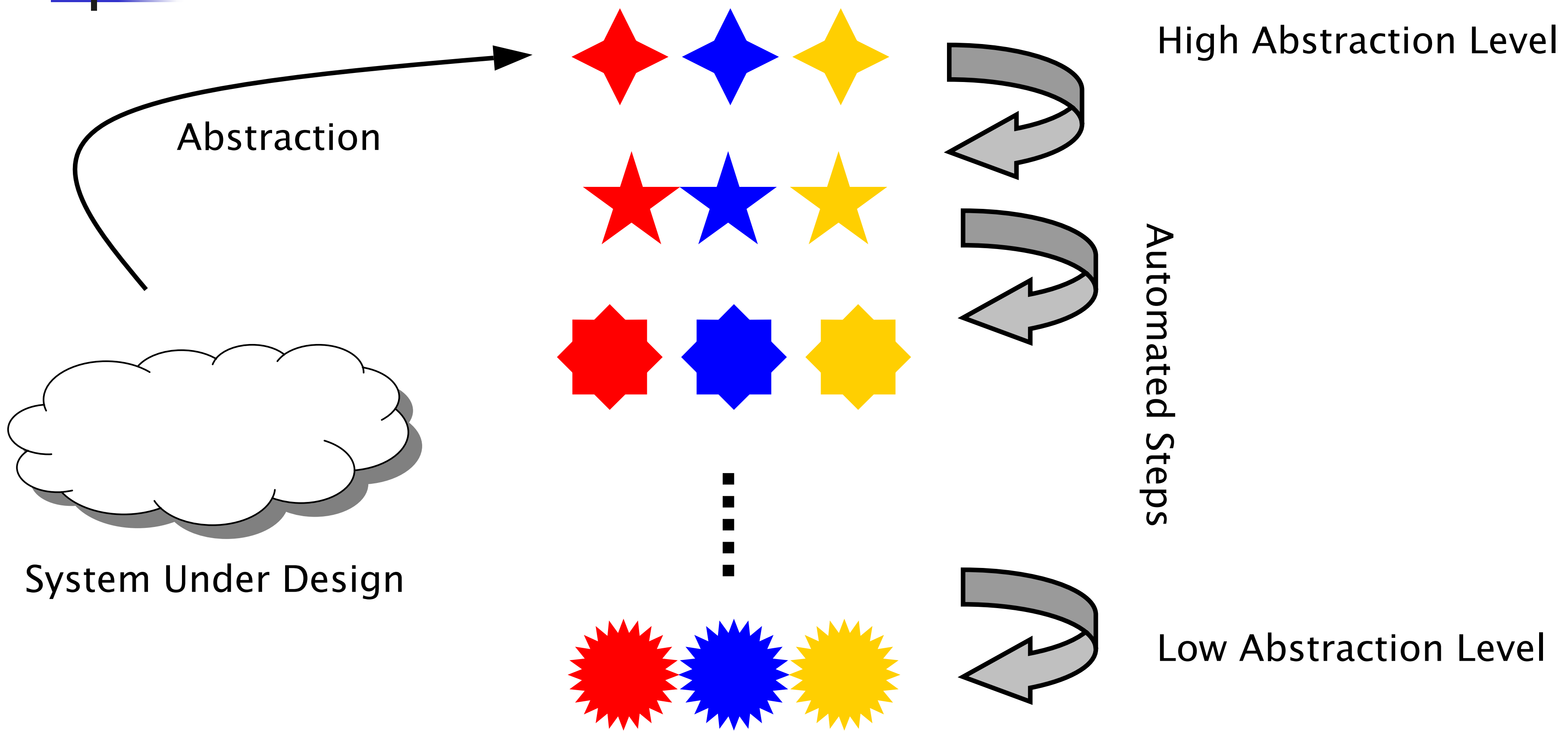


J.Bézivin,"On the Basic Principles of Model Engineering"

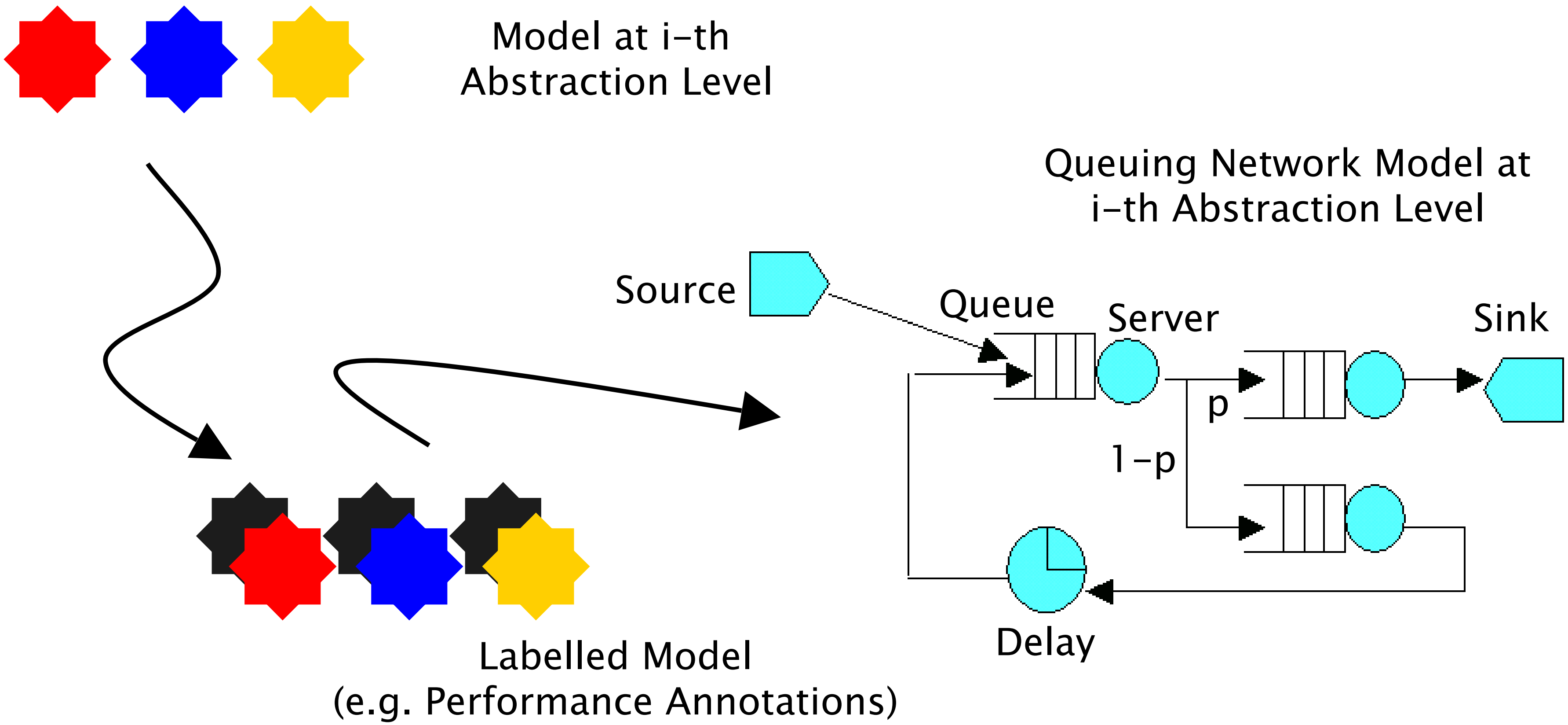
Model Transformations

- Classification of model transformations:
  - Model to Text
  - Model to Model
- Automation of Model transformation is key to MBD.
- Different approaches :
  - General purpose language approach (Java, C++, …)
  - XML based (XMI, XSLT)
  - Dedicated Transformation Language (QVT)

MBD: Top Down Approach



MBD: An Horizontal Refinement



Model at i-th
Abstraction Level

Labelled Model
(e.g. Performance Annotations)

Queuing Network Model at
i-th Abstraction Level

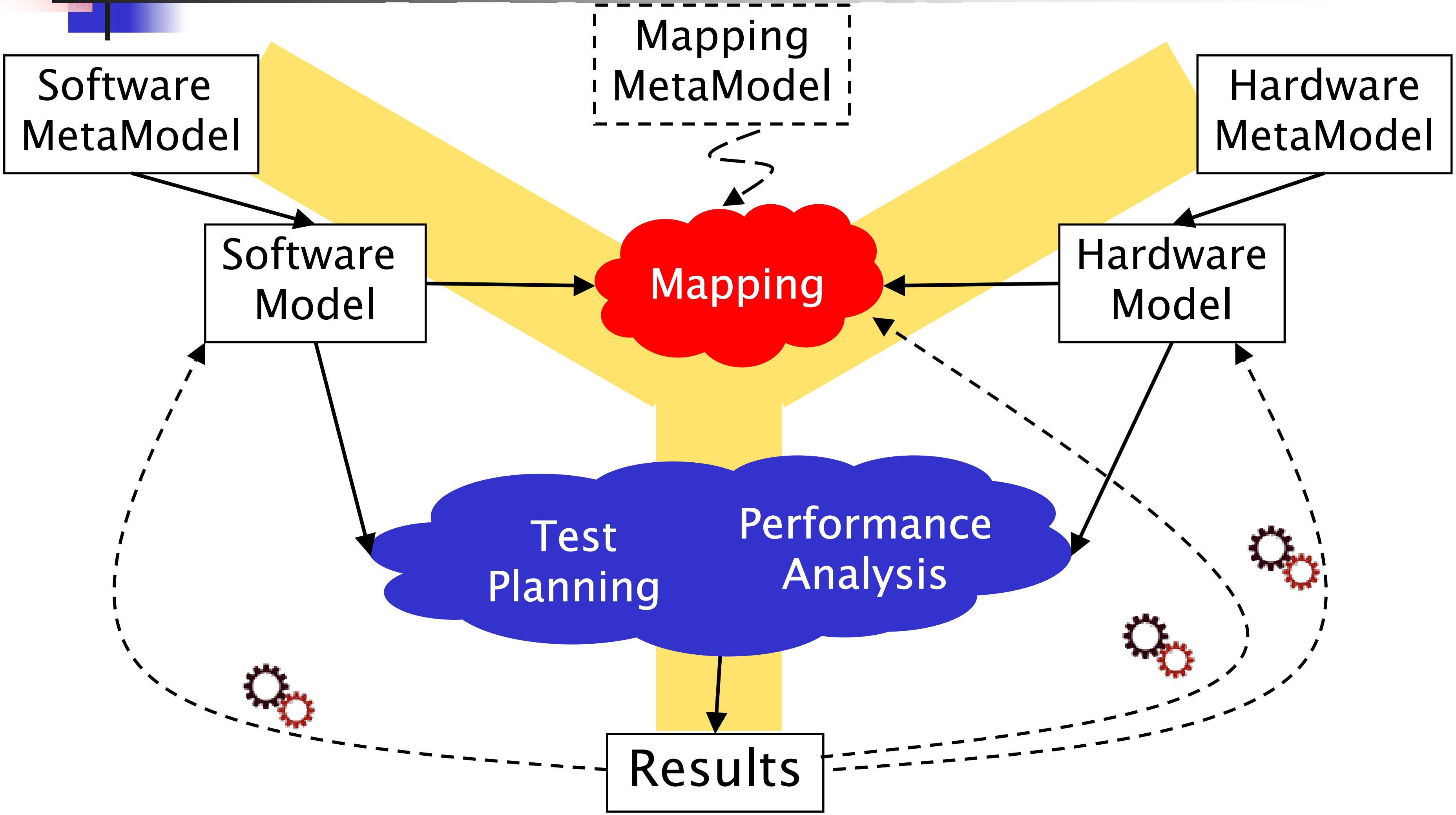
What Do We Propose ?!?

- Design an application for NP:
    - Decide which software architecture is best suited for the goal.
    - Represent the hardware architecture of the chosen NP.
    - Map each software unit on a specific hardware element.
    - Work according to the OMG Architecture.

How Can You Do It ?!?

- Software Model:
  - Software Entities
  - Relations
  - Performance Annotations: Number of code lines, Memory Allocation Space, etc …
- Hardware Model:
  - Elements : PPEs, Memories, etc …
  - Resources : Memory Size, Latency Access Time, etc …
- Mapping: Does a software element performance annotation meet resources limitations ?!?

Y–Model for NP Applications



Dynamic Aspects

- The hardware and software models represent a static description of the whole system.
- For a complete application design also dynamic aspects are required.
- The software model should describe both the dynamic of a single software unit and the data–flow among the different units:
    - Sequence diagrams
    - Queuing networks

Future Works

- Refine the definition of the methodology (this is on-going work).
  - Defining a Meta-Model for the software applications.
  - Specifying mapping aspects.
- A methodology application to case studies coming from the industrial world.

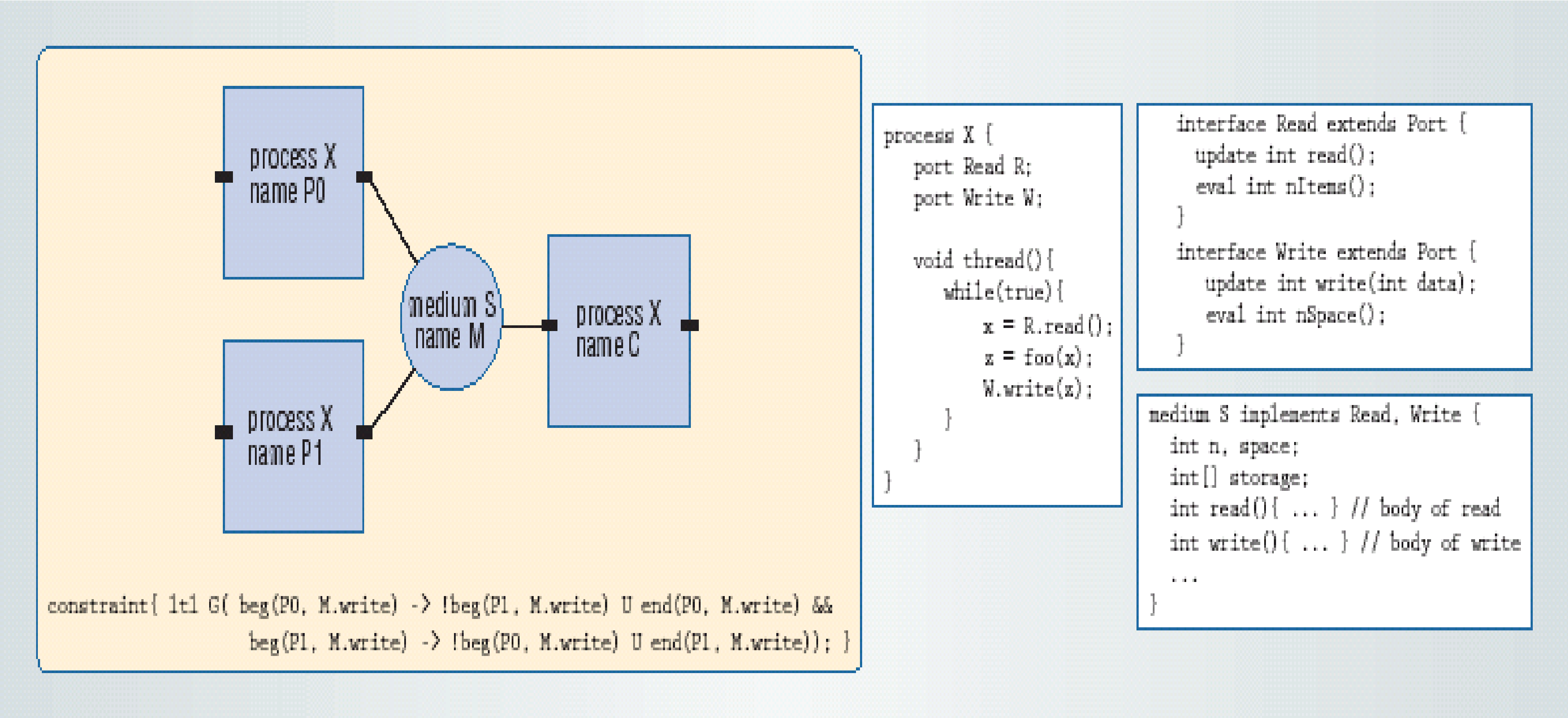
Conclusions

- We have presented an on-going work whose goal is the definition of a MBD approach for the design of software applications for network processors.
- The combination of MBD and NPs opens a new promising research field in software system engineering.

References

1. Agere. *The Challenge for Next Generation Network Processors*. White Paper.
2. A.Heppel. An introduction to network processors, January 2003.
3. B.Kienhuis, E.Deprettere, K.Vissers, and P.Van Der Wolf. An approach for quantitative analysis of application-specific dataflow architectures, August 04 1997.
4. B.Selic. The pragmatic of model-driven development. *IEEE Software*.
5. B.Selic. Model-driven development, uml 2.0, and performance engineering. In *Proceedings of the Fourth Int. Workshop on Software and Performance*. ACM, 2004. Invited talk WOSP2004.
6. C.U.Smith and L.Williams. *Performance Solutions: A practical Guide To Creating Responsive, Scalable Software*. Addison-Wesley, 2001.
7. D.Gajski and R.Kuhn. Guest Editors' introduction: New VLSI tools. *Computer*, 16(12):11-14, December 1983.
8. D.Hamlet, D.Mason, and D.Woit. Properties of software systems synthesized from components, June 2003. To appear as a book chapter. http://www.cs.pdx.edu/˜hamlet/lau.pdf.
9. D.Husak. *Network Processors: A Definition and Comparison*. C-Port. White Paper.
10. E.D.Lazowska, J.Zahorjan, G.S.Graham, and K.C.Sevcik. *Quantitative System Performance. Computer Systems Analysis Using Queueing Network Models*. Prentice Hall, Inc., 1984.
11. Intel. *Intel IXP2400 Network Processor: Flexible, High-Performance Solution for Access and Edge Applications*. White Paper.
12. P.Boulet, J.Dekeyser, C.Dumoulin, and P.Marquet. Mda for soc design, intensive signal processing experiment. *FDL'03, Frankfurt am Main.ECSI.*
13. S.A.Hissam, G.A.Moreno, J.A.Stafford, and K.C.Wallnau. Packaging predictable assembly. *Lecture Notes in Computer Science*, 2370:108-124, 2002.
14. Niraj Shah. Understanding network processors. Master's thesis, University of California, Berkeley, September 2001.
15. The Aspect Oriented Software Development Web Site. http://aosd.net.
16. The MDA Web Site. http://www.omg.org/mda/.
17. The Model-Driven Software Development Web Site. http://www.mdsd.info.
18. The MOF Web Site. http://www.omg.org/mof/.
19. T.Stefanov, P.Lieverse, E.Deprettere, and P.Van Der Wolf. Y-chart based system level performance analysis: An M-JPEG case study, October 16 2000.
20. V.Cortellessa and R.Mirandola. Prima-uml: A performance validation incremental methodology on early uml diagrams. *Science of Computer Programming*, 44(1):101-129, jul 2002.

Metropolis: Functionalities



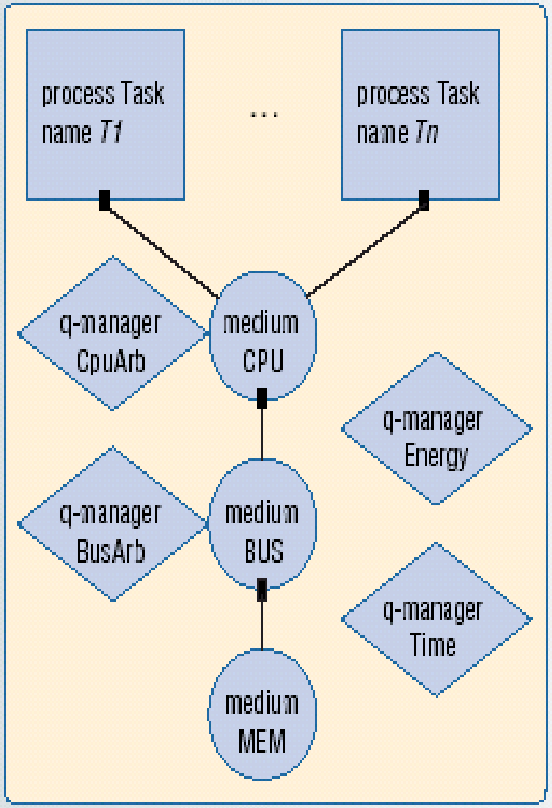
```
constraint{ ltl G( beg(P0, M.write) -> !beg(P1, M.write) U end(P0, M.write) &&
                   beg(P1, M.write) -> !beg(P0, M.write) U end(P1, M.write)); }
```

```
process X {
   port Read R;
   port Write W;

   void thread(){
      while(true){
          x = R.read();
          z = foo(x);
          W.write(z);
      }
   }
}
```

```
interface Read extends Port {
   update int read();
   eval int nItems();
}
interface Write extends Port {
    update int write(int data);
    eval int nSpace();
}
```

```
medium S implements Read, Write {
  int n, space;
  int[] storage;
  int read(){ ... } // body of read
  int write(){ ... } // body of write
  ...
}
```

Metropolis: Platform



```
process Task {
  port CpuService cpu;

  void execute(int n) {
    {$
      ... // make request to CpuArb
      ... // to become CPU owner
    $}
    cpu.execute(n);
  }
  void read() {
    ...
    cpu.read();
  }
  void write() { ... }
  void thread() { ... }
}
```

```
medium MEM implements SlaveService {
  void read {
    {$
      ... // make request to Time
      ... //  for a memory read delay
    $}
  }
} void write { ... }
```

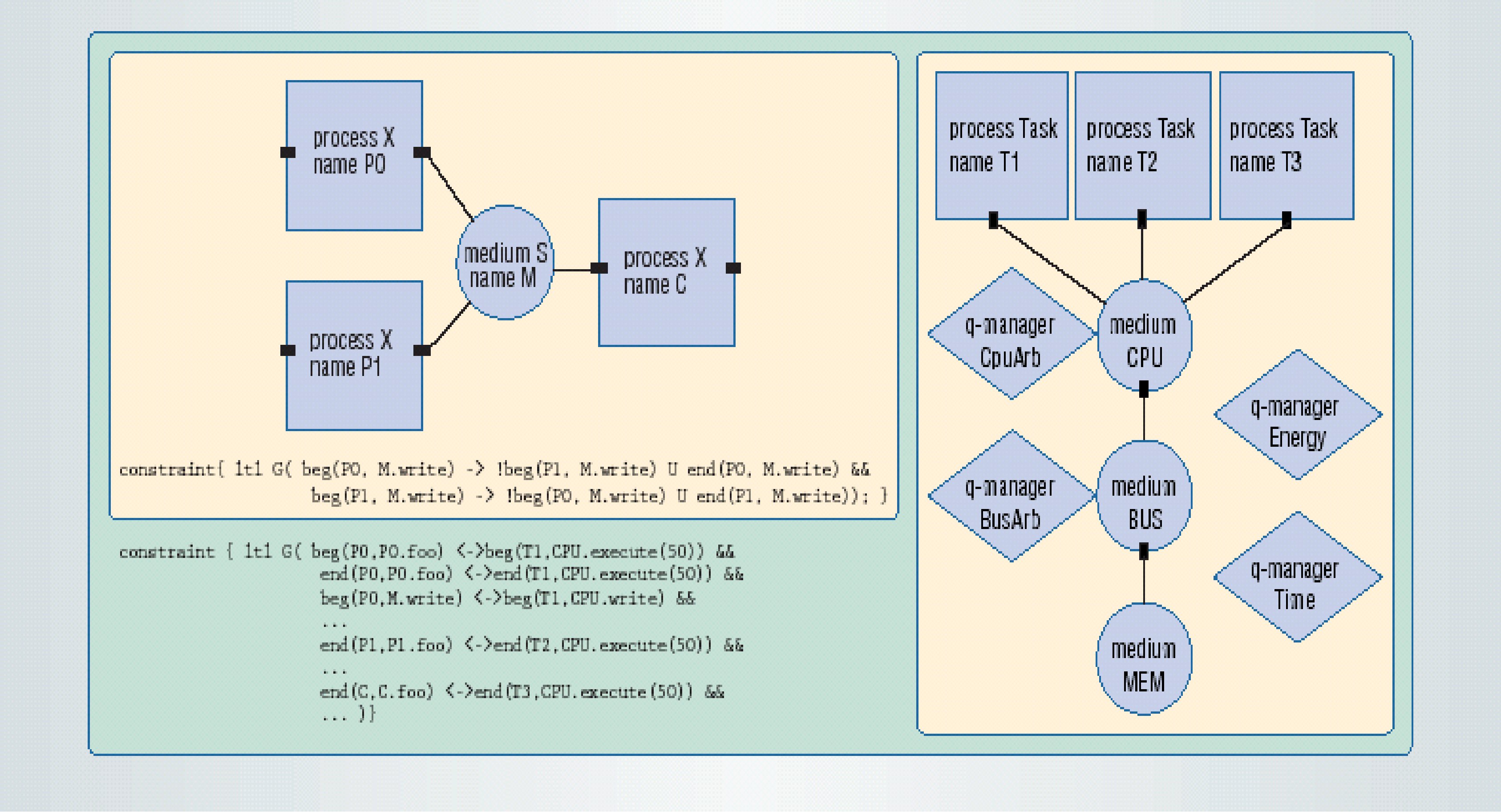
```
medium CPU implements CpuService {
  prt BusService bus;

  void execute(int n) {
    {$
      ... // make request to Time
      ... //  for a delay of n clock cycles
    $}
  }
  void read()  { ...   bus.read(); }
  void write() { ...   bus.write(); }
  }
```

```
medium BUS implements BusService {
  port SlaveService mem;

  void read()  {

    {$
      ... // make request to BusArb
      ... // to become bus master
    $}
    mem.read();
  }
  void write() { ...   mem.write(); }
  }
```

Metropolis: Mapping

- Mapping is defined by a new network to encapsulate the functional and architectural networks and relating the two by synchronizing events between them.

Metropolis: Mapping



```
constraint{ ltl G( beg(P0, M.write) -> !beg(P1, M.write) U end(P0, M.write) &&
                   beg(P1, M.write) -> !beg(P0, M.write) U end(P1, M.write)); }
```

```
constraint { ltl G( beg(P0,P0.foo) <->beg(T1,CPU.execute(50)) &&
                    end(P0,P0.foo) <->end(T1,CPU.execute(50)) &&
                    beg(P0,M.write) <->beg(T1,CPU.write) &&
                    ...
                    end(P1,P1.foo) <->end(T2,CPU.execute(50)) &&
                    ...
                    end(C,C.foo) <->end(T3,CPU.execute(50)) &&
                    ... )}
```