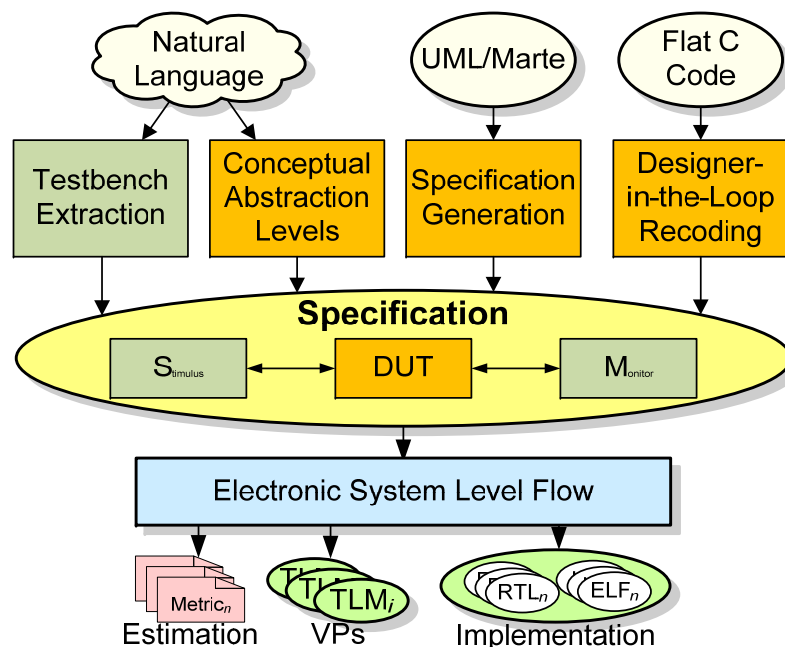


# Modeling and SW Synthesis for Heterogeneous Embedded Systems in UML/MARTE

Tutorial SD1: High-Level Specifications to Cope With Design Complexity

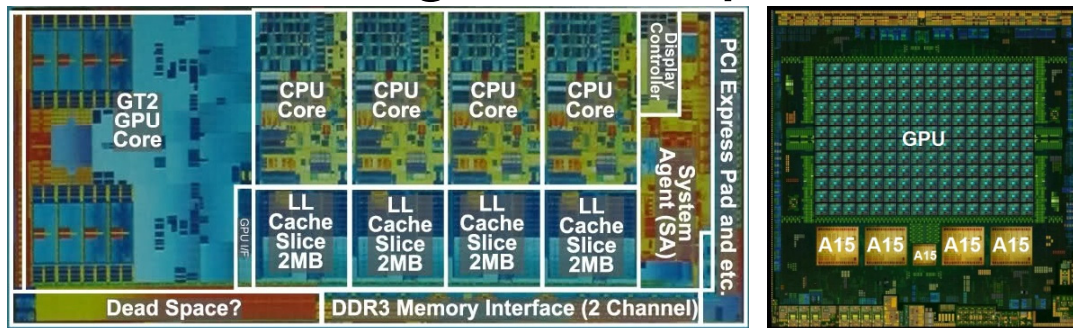


Hector Posadas, Pablo Peñil,  
Alejandro Nicolás, **Eugenio Villar**

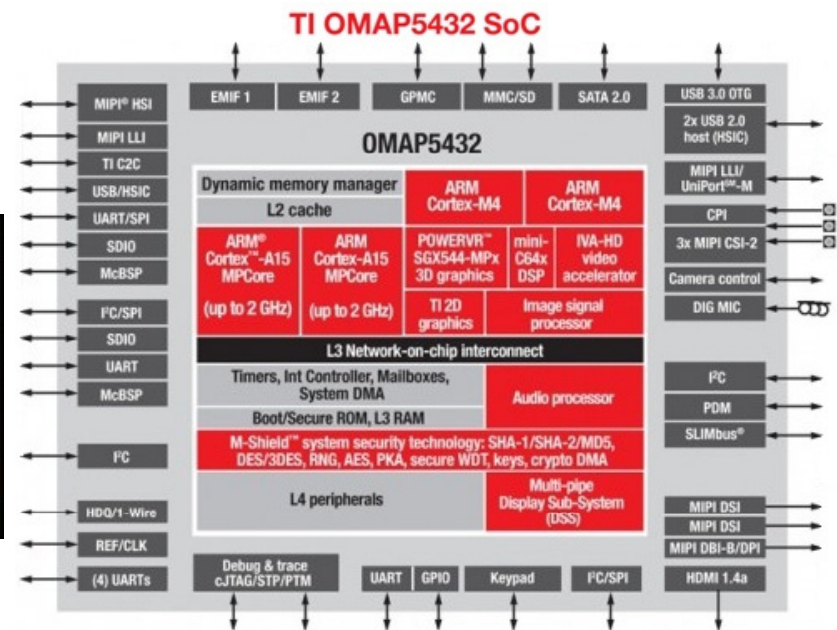
University of Cantabria  
Spain

# Motivation

- Design productivity gap
  - Raising the abstraction level
- Multi-Processing & Heterogeneous platforms

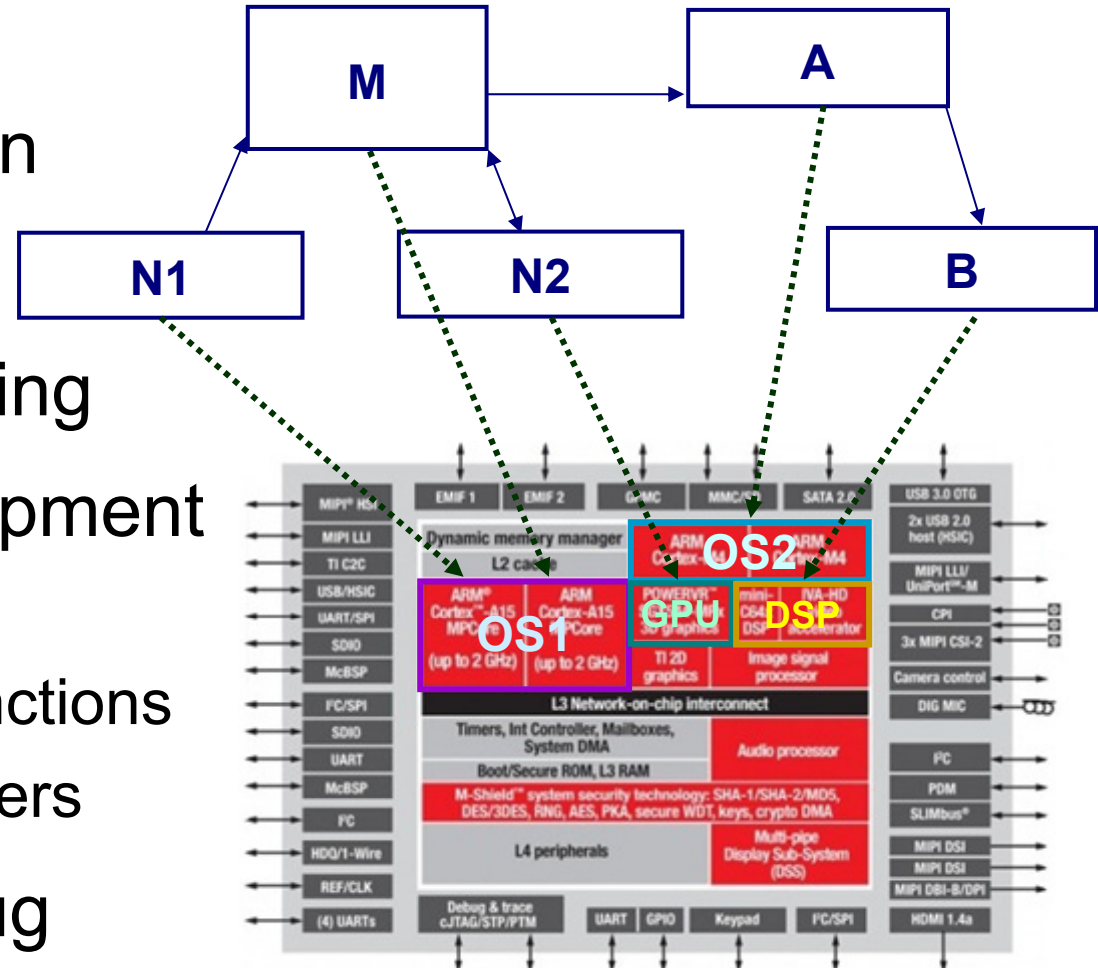


- Increasing SW content
  - SW-centric design methodologies



# Usual SW development flow

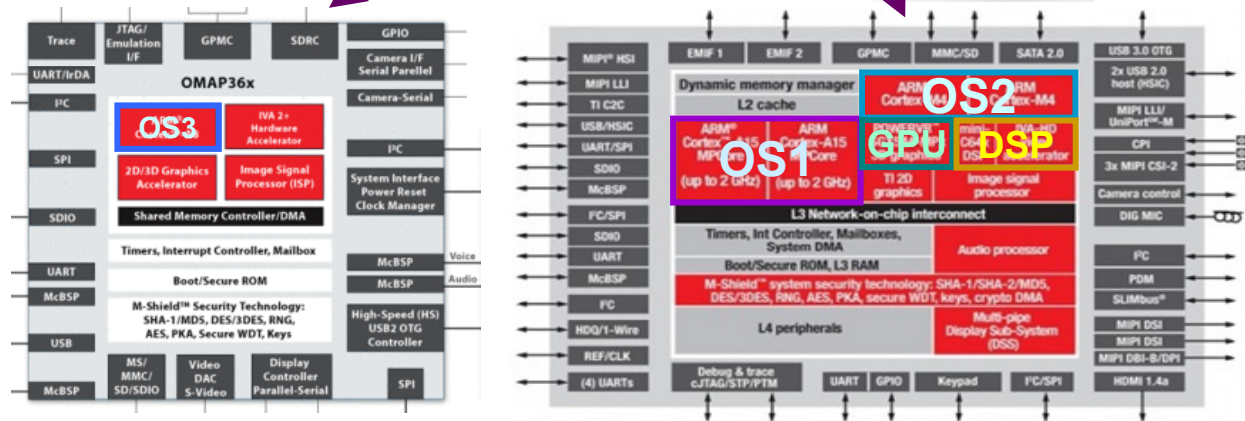
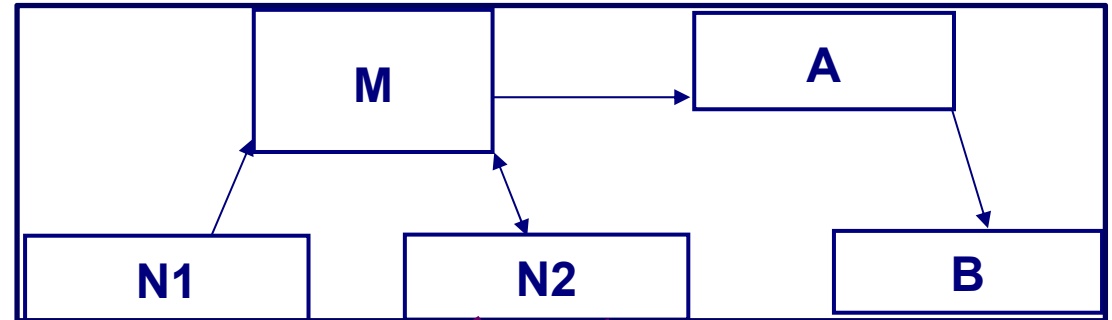
- Architectural Design
- HW/SW platform
- Architectural mapping
- Ad/Hoc SW development
  - System calls
  - Communication functions
  - I/O functions & drivers
- Verification & Debug
- Costly fixing of wrong design decisions



# Usual SW development flow

- Lack of reusability

- Ad-hoc code
- Large re-engineering effort



# Outline

---

- Introduction
- State of the Art
- The PHARAON approach
  - Design Flow
  - Modeling Methodology
  - SW Synthesis
- Conclusions

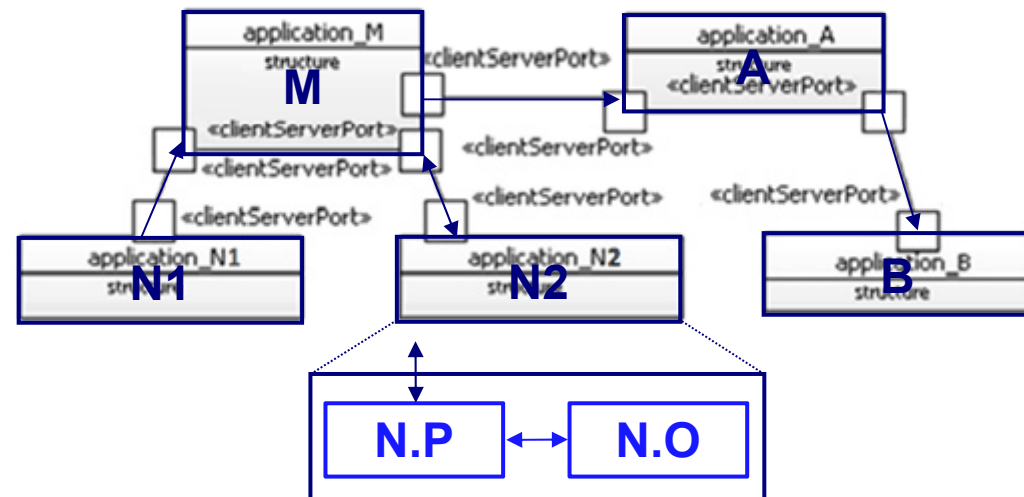
# Introduction

---

- Model-Driven Architecture (MDA)
  - High-abstraction level
  - Mature SW engineering methodology
  
- UML language
  - Application to embedded systems design

# Introduction

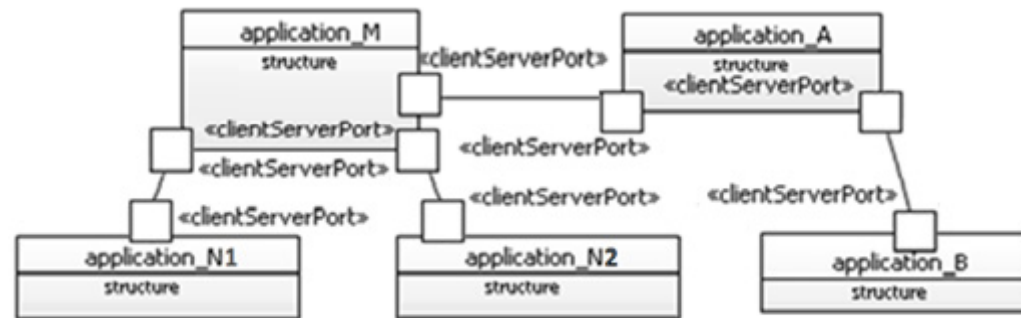
- Why UML?
  - Natural way to capture system architecture
  - Standard way



# Introduction

- Why UML?

- Natural way to capture system architecture
- Standard way




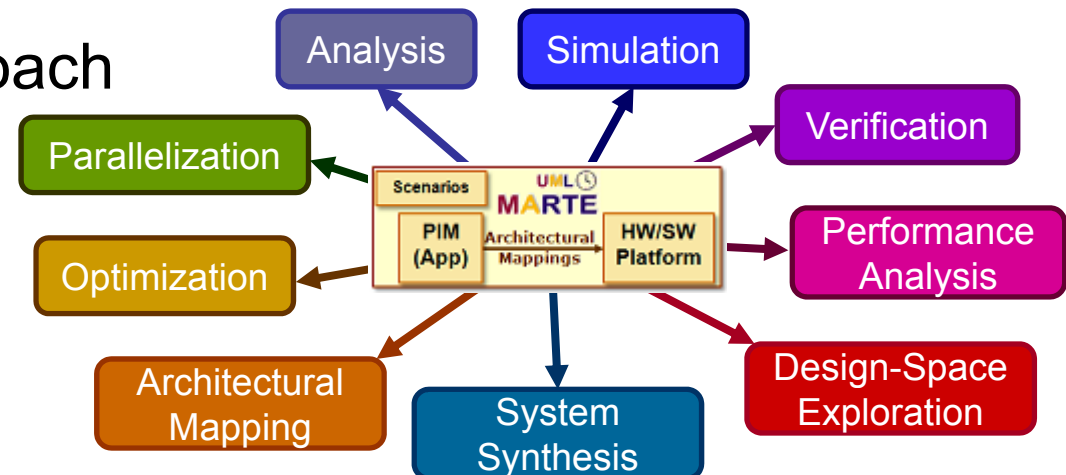
- UML language

- Semantics lacks
  - What is each component?
  - What kind of interaction each link actually means?
- Domain-specific profiles
  - UML/MARTE



# Introduction

- MARTE 
  - Standard UML profile for real-time embedded systems
    - Platform-Independent Model (PIM)
    - Platform Description Model (PDM)
    - Platform-Specific Model (PSM)
  - Rich semantics content
  - Single-source approach



# State-of-the-Art

---

## ■ Discussion

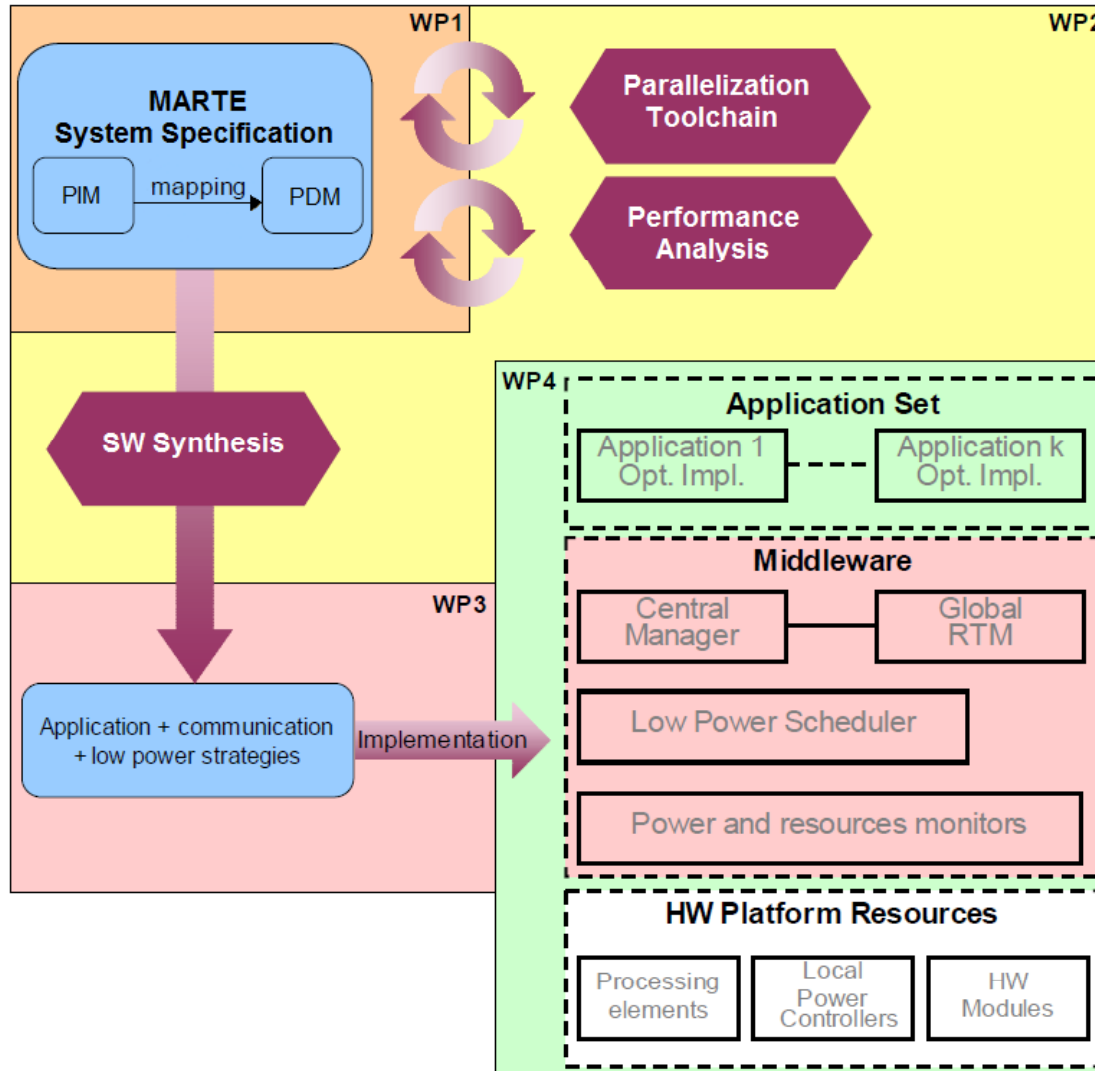
### □ System modeling in MARTE

- Methods based on specific MoCs and/or profiles
  - Requiring additional semantics
  - Non-standard

### □ SW Synthesis

- Commercial code generation available
- Limited support for heterogeneity
- Limited flexibility for different architectural mappings
- Limited support for the MARTE semantics

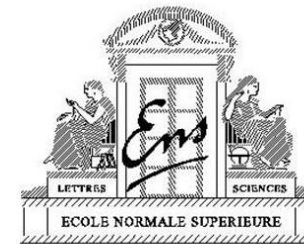
# PHARAON Single-Source Design Flow



THALES

TEDESYS

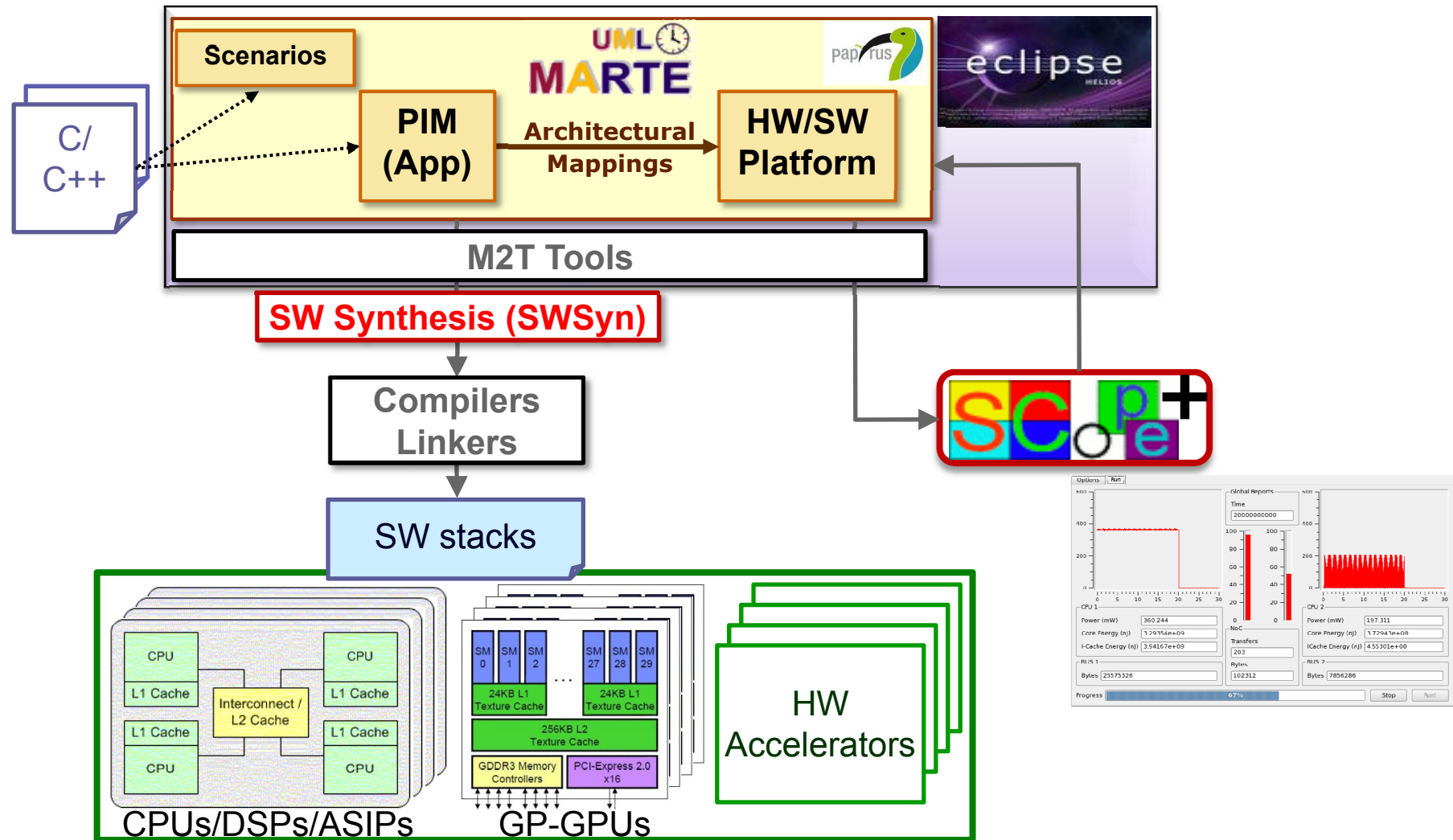
VectorFabrics  
do more with multicore



UC  
UNIVERSIDAD  
DE CANTABRIA

imec

# PHARAON Single-Source Design Flow



# PHARAON Modeling Methodology

---

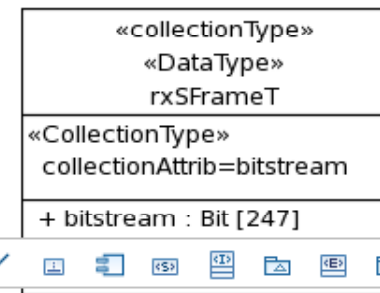
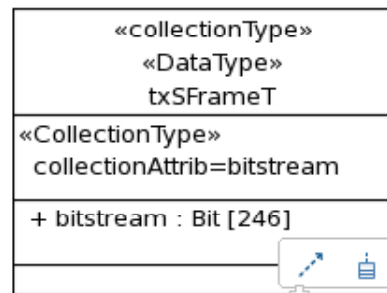
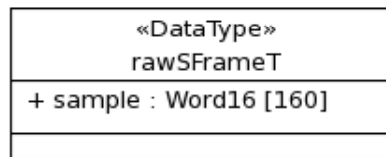
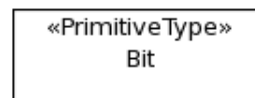
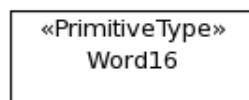
## ■ Main features

- MDD concepts
  - Separation of Concerns
- CBE: Component-Based Engineering approach
- SW centric
- Standard
  - MARTE profile

# PHARAON Modeling Methodology

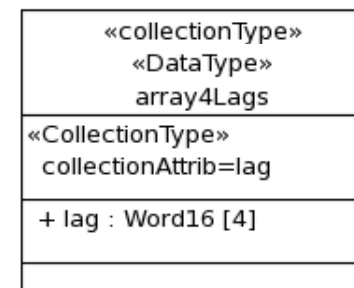
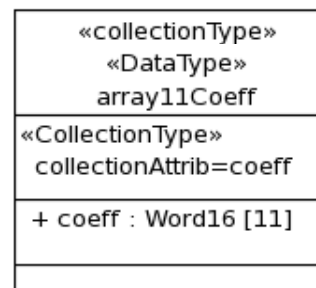
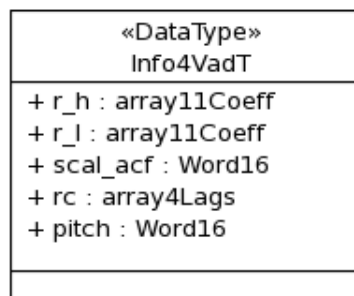
## ■ Data Types for Communication Interfaces

### □ Primitive Types



### □ Bit Arrays

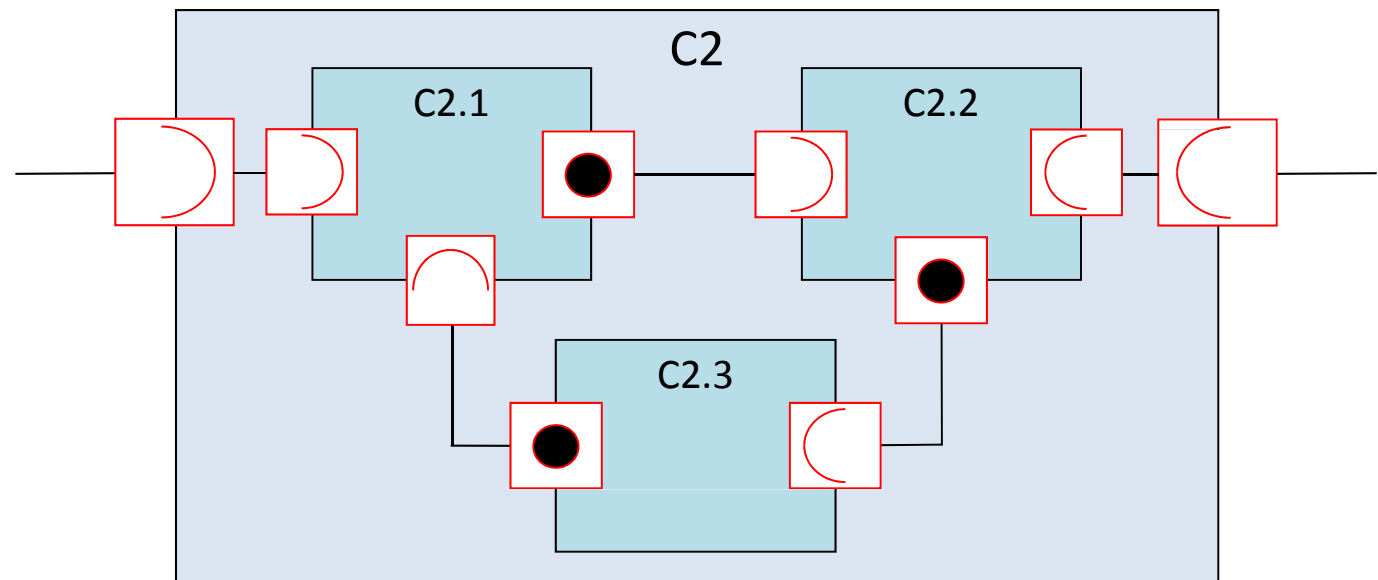
### □ Data Structures



### □ Arrays

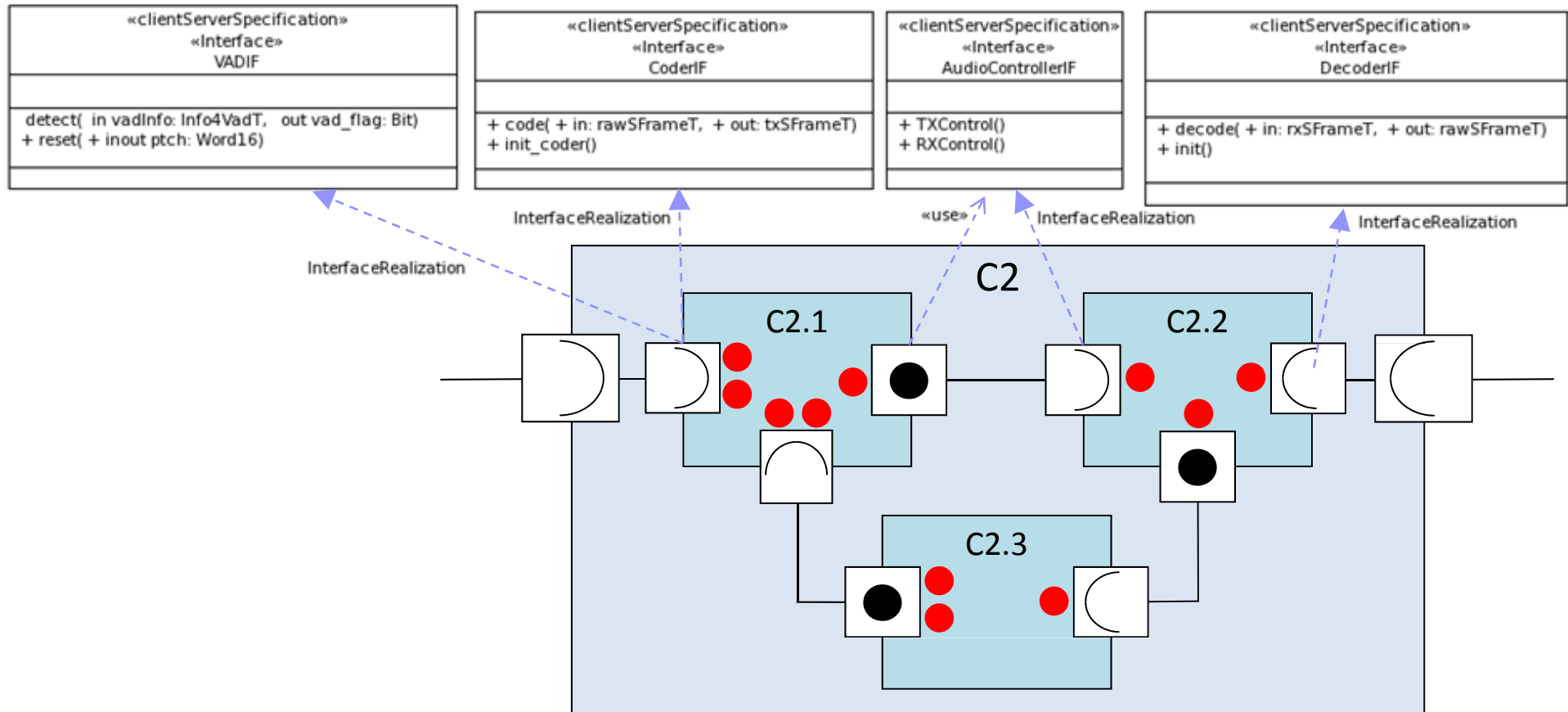
# PHARAON Modeling Methodology

- Component model
  - Hierarchical functional encapsulation
  - Ports
    - provided or required



# PHARAON Modeling Methodology

## ■ Component Interfaces



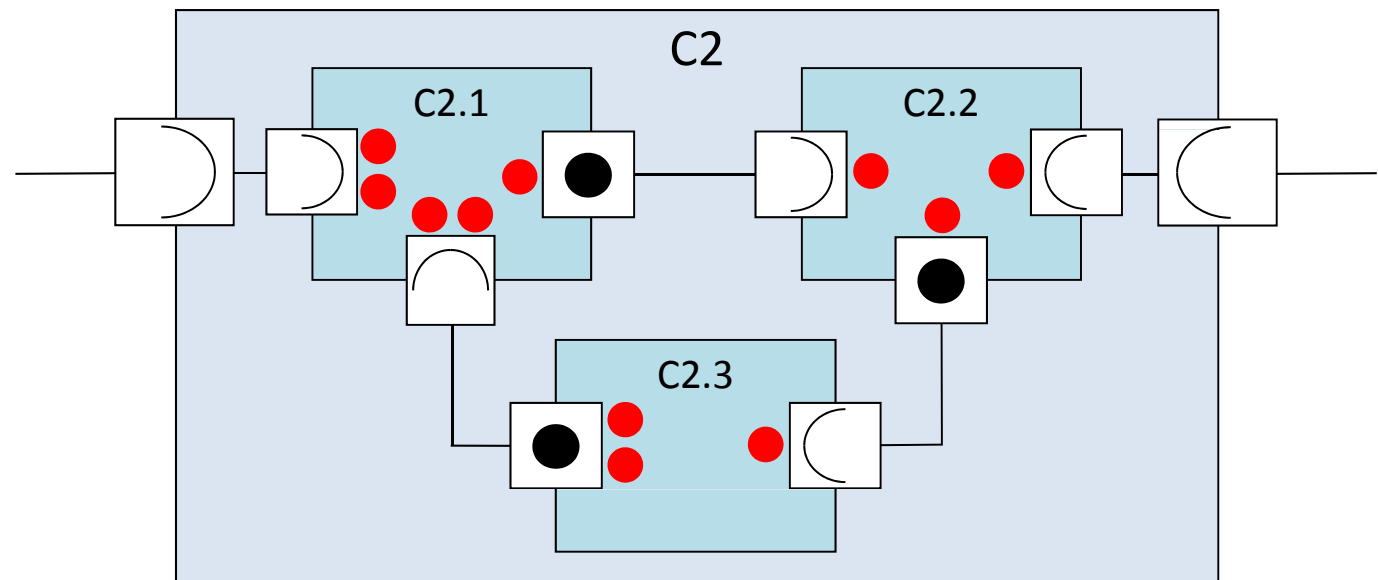


# PHARAON Modeling Methodology

## ■ Component model

### □ Interfaces

- sequential, guarded or concurrent, Max. threads available
- argument sizes (data splitting), Num. of incoming channels



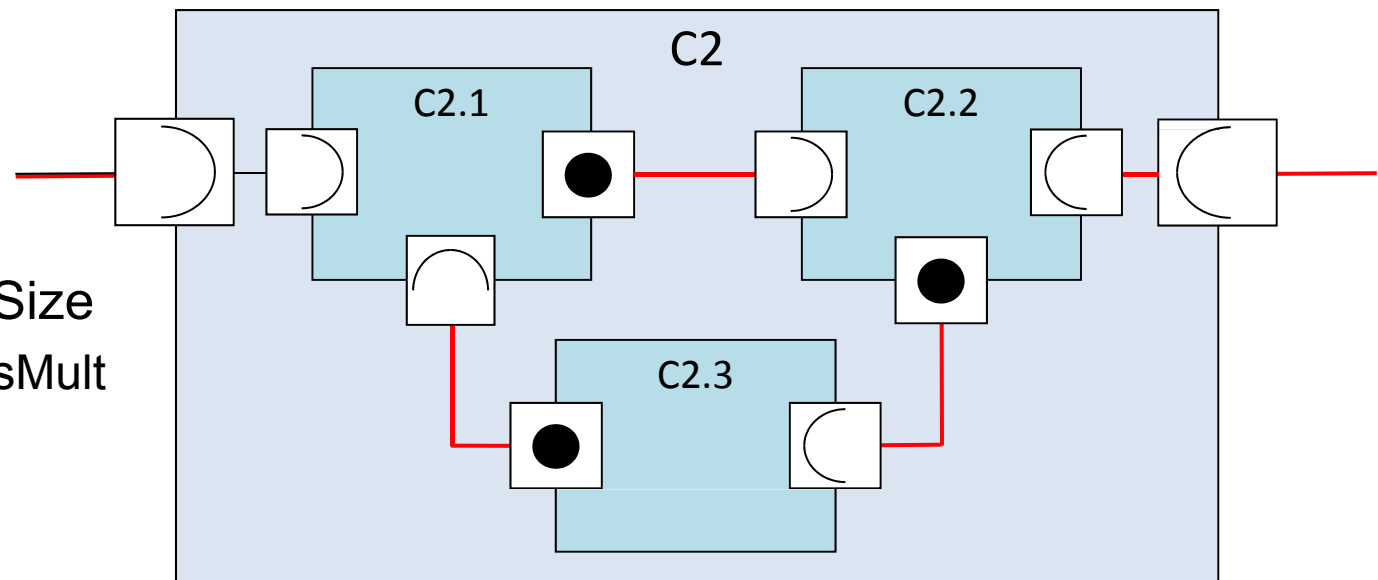
# PHARAON Modeling Methodology

## ■ Component model

### □ Channels manage communications

- BlockingFunctionCall, BlockingFunctionReturn, both or none
- Timeout
- Priority

- Buffer Size
  - ResMult



# PHARAON Modeling Methodology

## ■ The Platform Description Model

### □ HW/SW Components using MARTE stereotypes

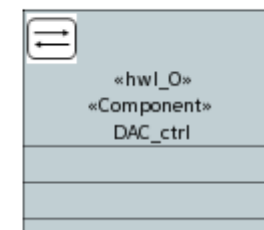
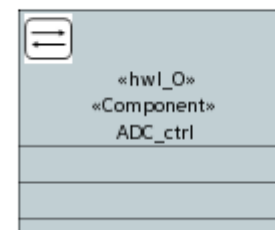
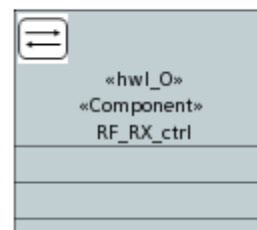
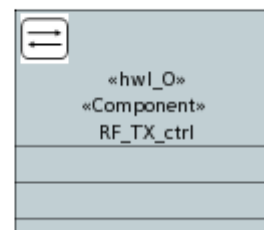
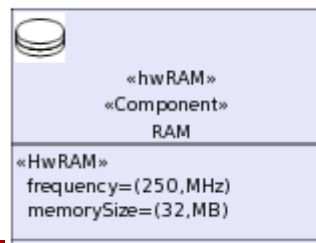
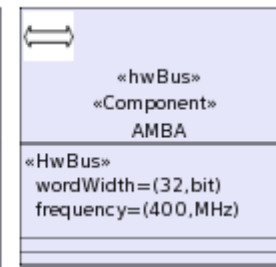
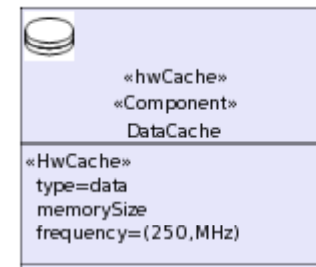
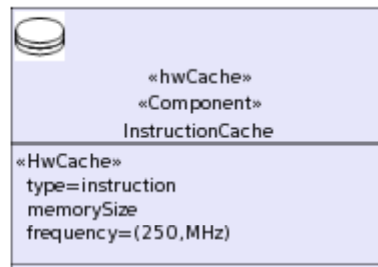
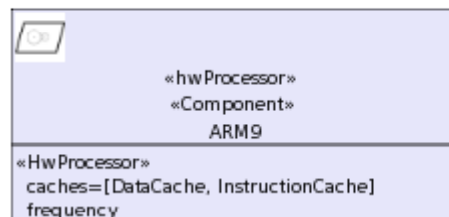
#### ■ Software Components

- OS, HdS, Drivers, ...



#### ■ Hardware Components

- Processors, Memories, Buses, Custom HW, I/O



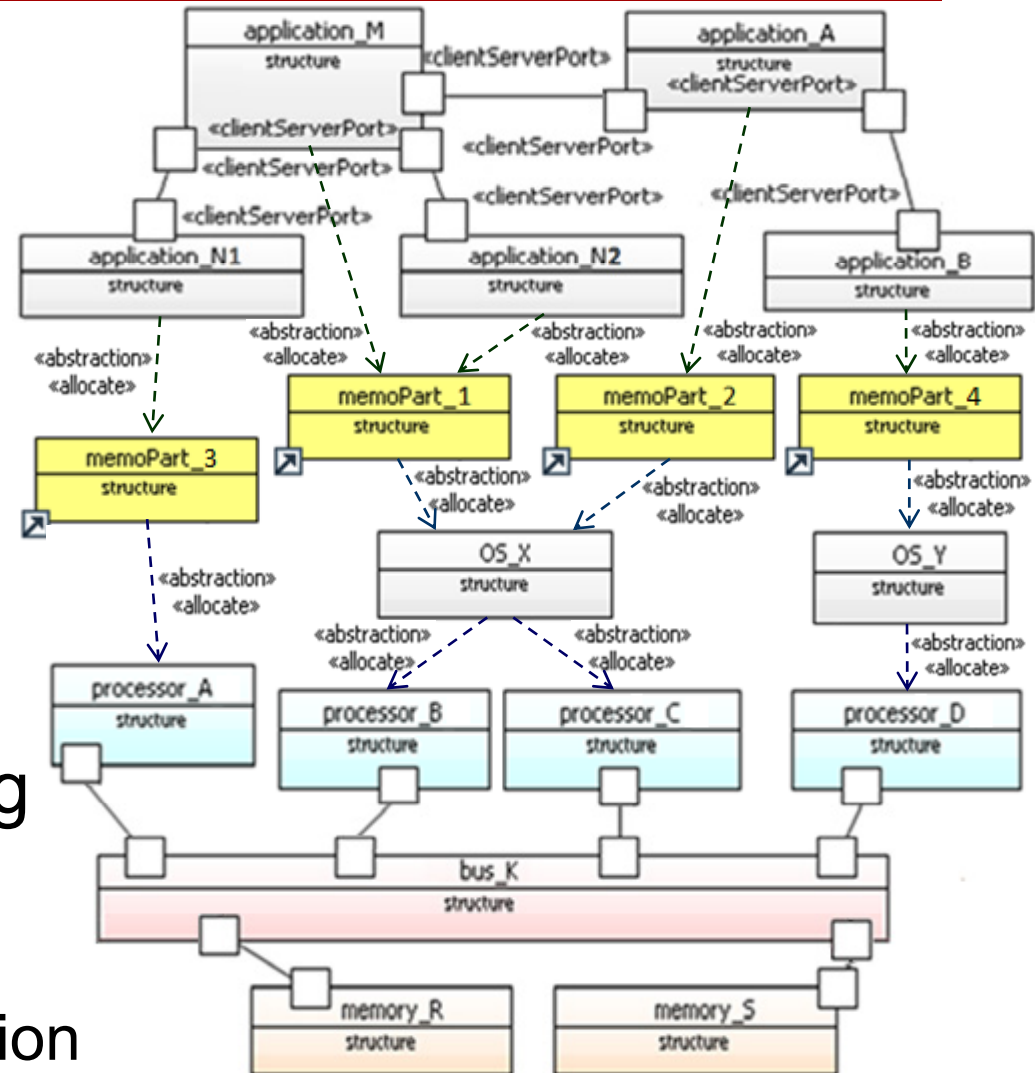
# PHARAON Modeling Methodology

---

- Platform-Specific Model
  - Memory spaces mapped to platform resources
  - Mapping of functional components
    - to memory spaces and/or platform resources

# PHARAON Modeling Methodology

- Architectural Design
- Code reuse and/or development
  - platform independent
- HW/SW platform
- Architectural mapping
- SW Synthesis
  - Fast design optimization



# SW Synthesis

---

- System heterogeneity
- Full support for
  - any architectural mapping decided for each component
  - any specific processing resource selected
  - any processing resource type
  - any memory space
  - any OS used by the processing resource
  - any communication infrastructure

# SW Synthesis

---

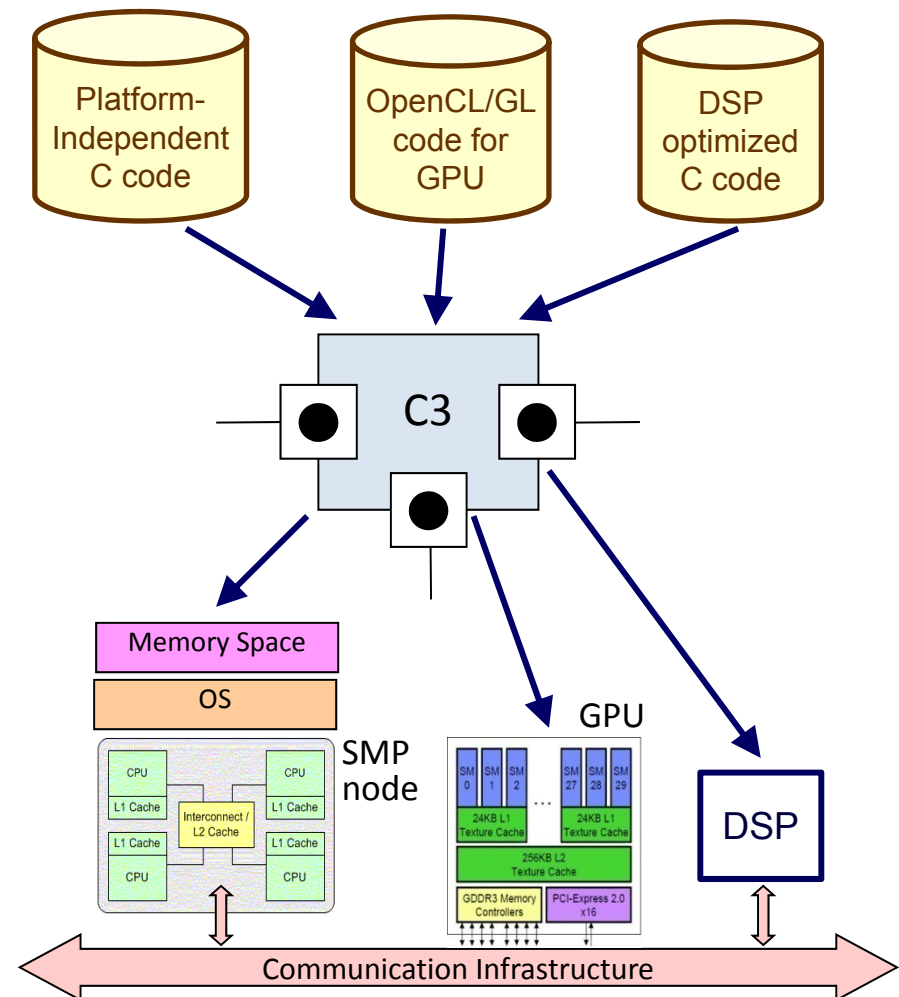
- Functional synthesis

- One executable per memory-space
- Platform-Independent (C/C++) code
  - Highest reusability
  - Non-recommended explicit calls to platform services
    - communication, concurrency, etc.
  - Platform services should derived from the UML/MARTE model
  - POSIX and/or OpenMP as alternatives
  - Static execution flows
    - <<SwSchedulableResource>>

# SW Synthesis

- Functional synthesis
  - Platform-Specific code
    - Optimized C code for DSPs
    - OpenCL/GL for GPUs
    - OpenMP for SMPs...

Configuration	Original	Optimized Code
ARM	908.28 sec	572.92 sec
ARM-NEON	325.81 sec	255.28 sec
ARM+DSP blocking call	206.01 sec	193.45 sec
ARM+DSP non-blocking call	895.98 sec	431.68 sec
ARM-NEON+DSP non-blocking call	247.93 sec	215.96 sec





# SW Synthesis

---

- Communication synthesis
  - Essential activity in heterogeneous SW synthesis
  - Client-Server paradigm
  - Dependent on the architectural mapping

# SW Synthesis

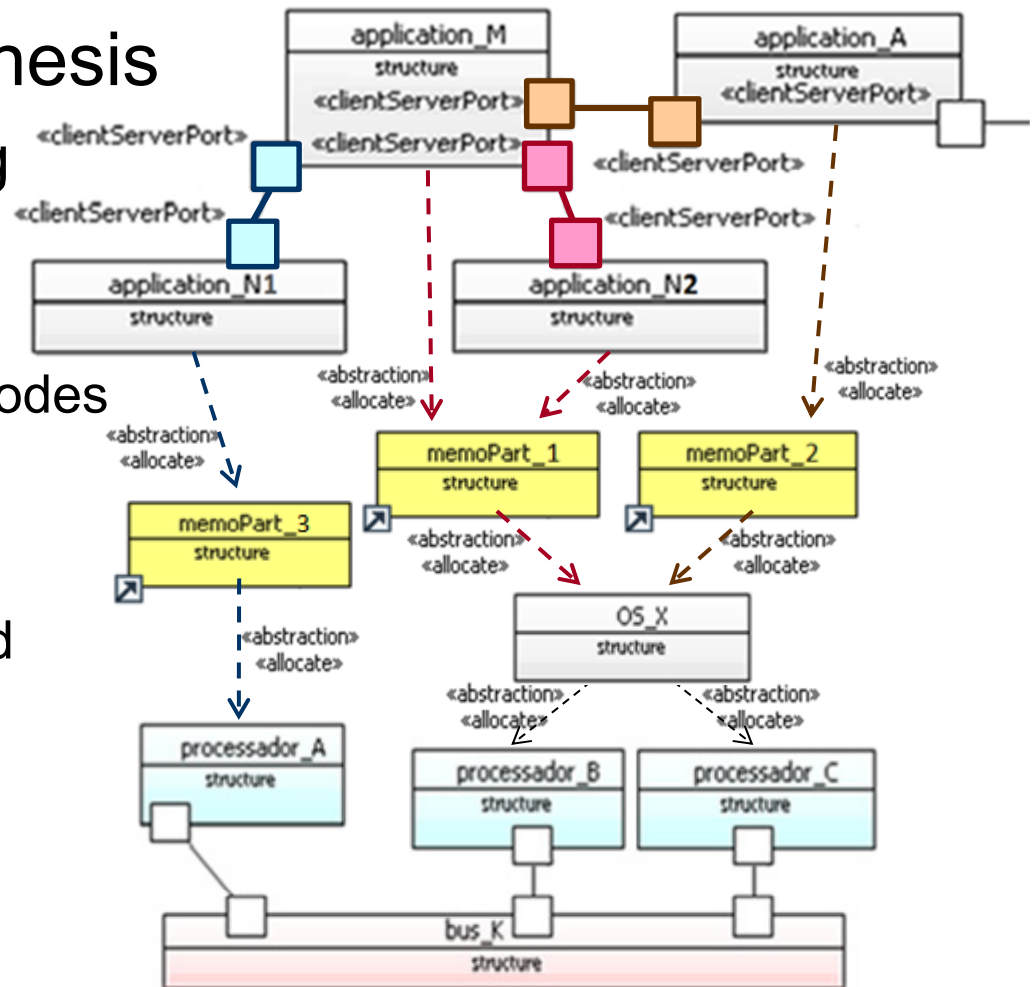
## ■ Communication synthesis

### □ Architectural mapping

- Same memory space
- Same OS
- Different processing nodes

### □ Benefits / Drawbacks

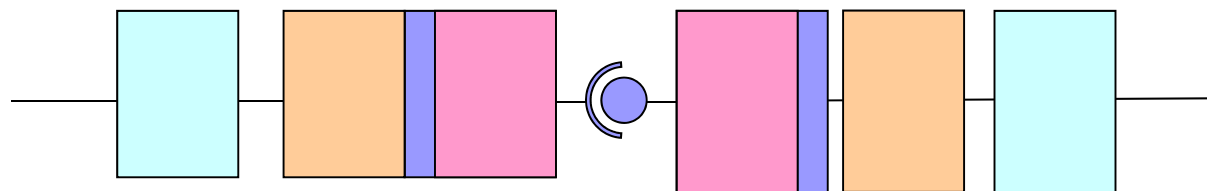
- Communication Speed
- Memory protection
- Memory/cache use
- Scheduling
- Parallelism...



# SW Synthesis

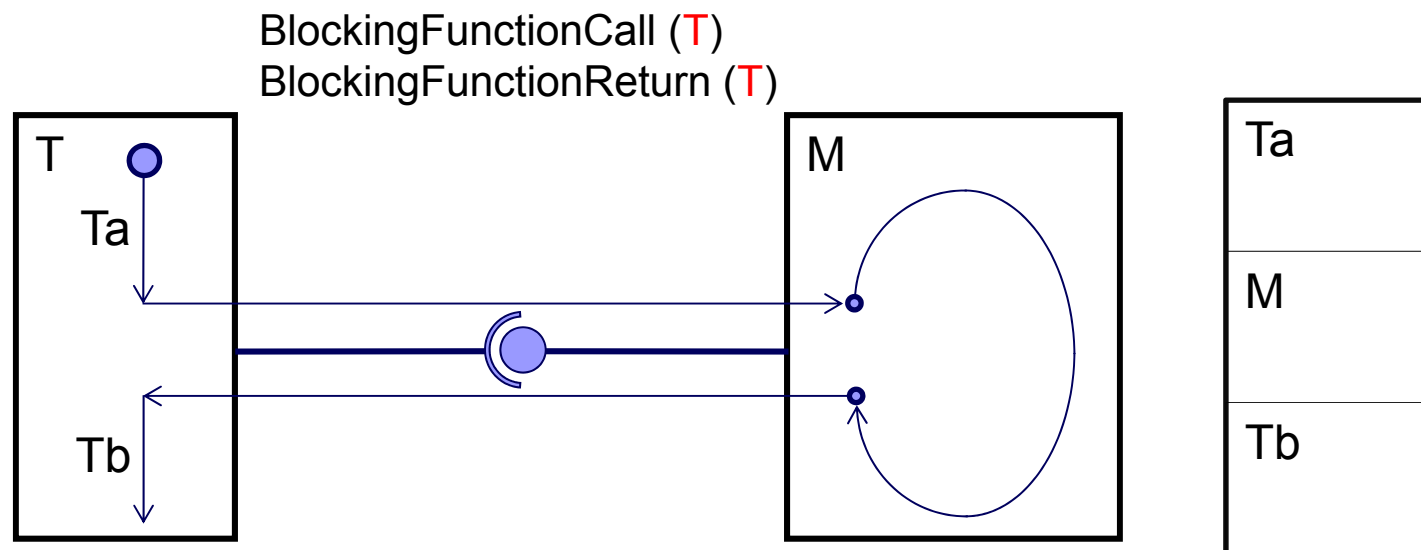
## ■ Communication synthesis

- 3 Layers automatically inserted in service calls
  - Layer 1: communication semantics
    - Allocation independent
  - Layer 2: management for allocation-dependent communications
    - Thread generation, data splitting, synchronization
  - Layer 3: Low-level communications
    - Inter-thread, Inter-process, distributed communication



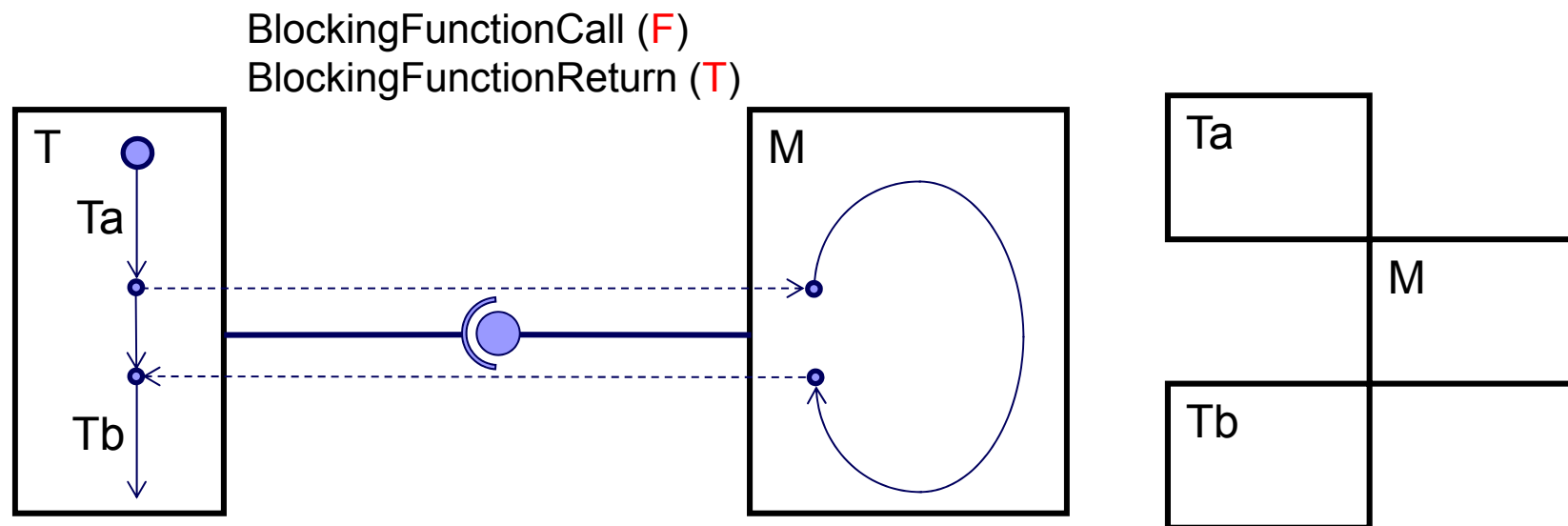
# SW Synthesis

- Communication synthesis
  - Layer1: Independent of architectural mapping
    - Channel properties
      - RPC



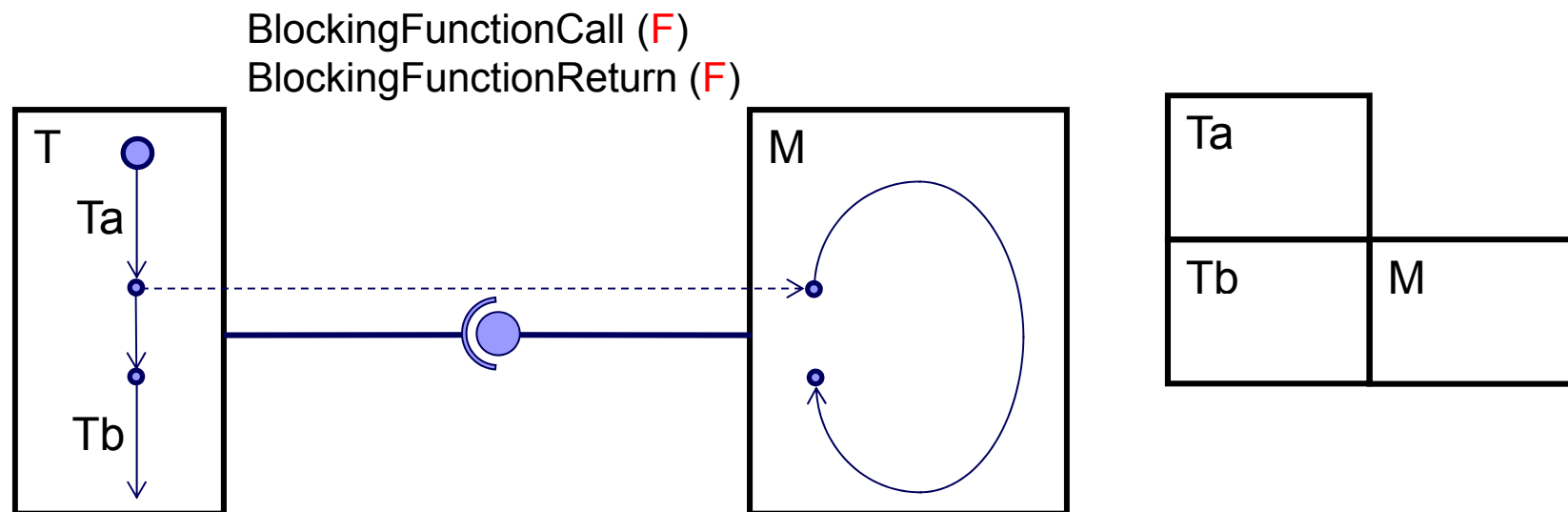
# SW Synthesis

- Communication synthesis
  - Layer1: Independent of architectural mapping
    - Channel properties
      - Pipeline



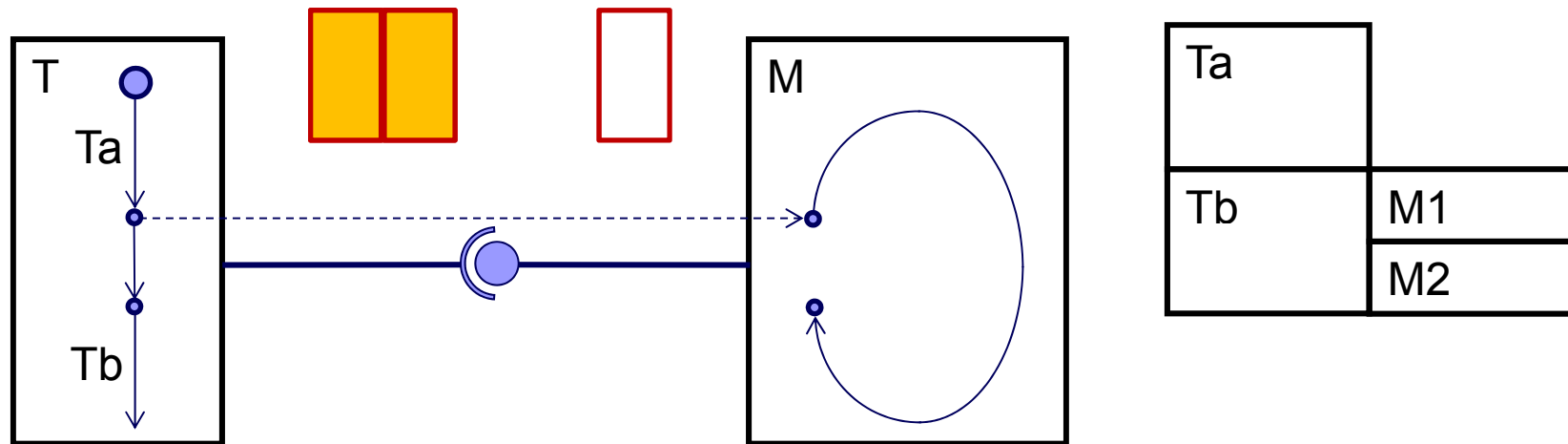
# SW Synthesis

- Communication synthesis
  - Layer1: Independent of architectural mapping
    - Channel properties
      - Pipeline & Parallel



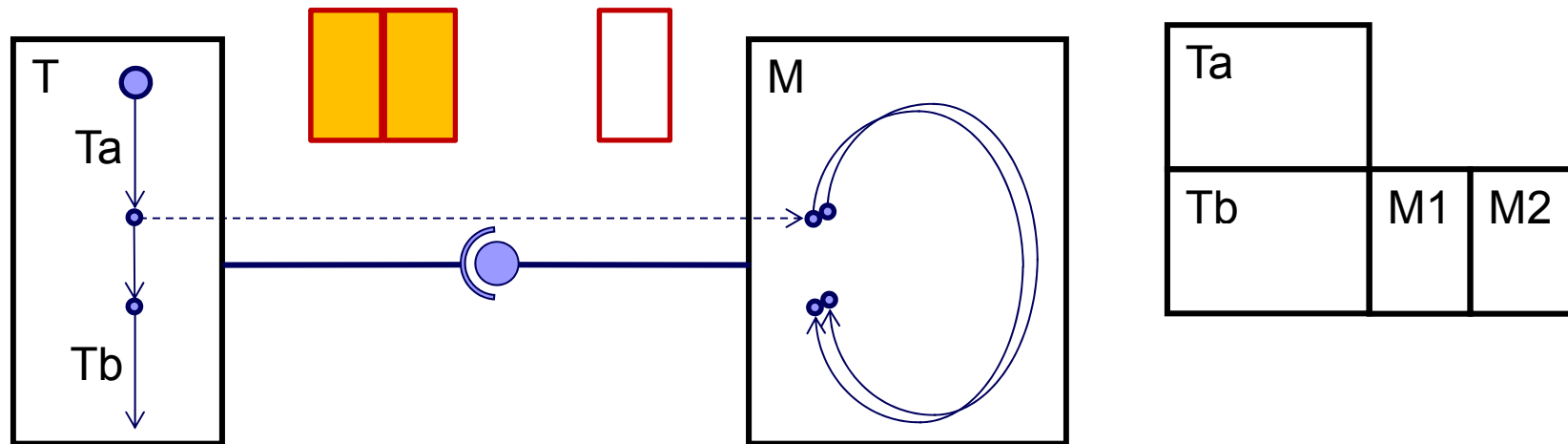
# SW Synthesis

- Communication synthesis
  - Layer1: Independent of architectural mapping
    - Interface properties
      - Data splitting



# SW Synthesis

- Communication synthesis
  - Layer1: Independent of architectural mapping
    - Interface properties
      - Data splitting





# SW Synthesis

---

## ■ Implementation alternatives

- Channel semantics can be implemented in multiple ways
  - Different OS services
    - shared memory
    - message queue
    - socket
    - file...
- Performance is highly dependent on platform and OS
- Synthesis enables fast exploration
  - optimal channel implementation for specific platform and code

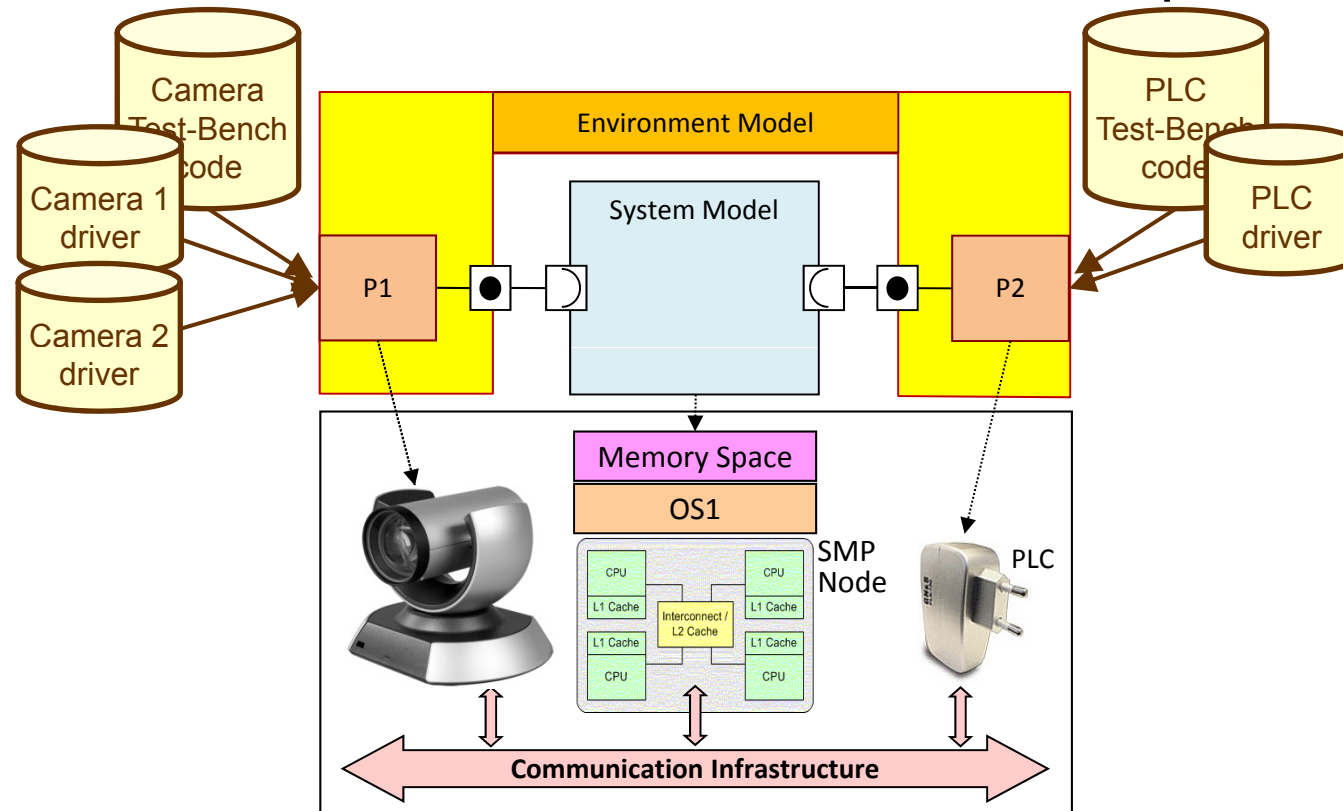
# SW Synthesis

---

- MultiCore Association APIs
  - Standard APIs for communication and synchronization
  - Closely distributed embedded systems
    - MCAPI - communication
    - MRAPI - synchronization
    - MTAPI - task generation
  - Independence from OS
  - OS-agnostic channel implementation
    - Components treated as MCAPI nodes
    - Ports treated as MCAPI endpoints

# SW Synthesis

- Platform Inputs & Outputs
  - Drivers associated to environmental components



# Conclusions

---

## ■ UML/MARTE

- Powerful modeling methodology
- Single-Source approach
- Platform-Independent Modeling
- Maximizing reusability

# Conclusions

---

## ■ SW Synthesis

- Functional modeling
- Functional synthesis
- Communication synthesis
- Platform Inputs & Outputs
  
- Actually enables platform-independent code
- Reduces in-depth knowledge of platforms
- Support shorter design optimization cycles
  - wider design exploration
  - shorter code generation on heterogeneous platforms

# Additional Information

---

- <http://www.teisa.unican.es/gim/en/proyecto?id=95>
- H. Posadas, P. Peñil, A. Nicolás, E. Villar  
 "Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation"  
**Microelectronics Journal**, in press, doi: 10.1016/j.mejo.2013.11.003.
- H. Posadas, E. Villar, et al.  
 "EU FP7-288307 PHARAON project: Parallel and heterogeneous architecture for real-time applications"  
**Euromicro Conference on Digital System Design**, DSD 2013, IEEE, doi: 10.1109/DSD.2013.47.
- H. Posadas, P. Peñil, A. Nicolás, E. Villar  
 "System synthesis from UML/MARTE models: The PHARAON approach",  
**Electronic System Level Synthesis Conference**, ESLsyn, 2013, IEEE.
- P. Peñil, H. Posadas, A. Nicolás, E. Villar  
 "Automatic synthesis from UML/MARTE models using channel semantics"  
**International Workshop on Model-Based Architecting and Construction of Embedded Systems**, ACES-MB 2012, doi: 10.1145/2432631.2432640.