ARTIST2 - MOTIVES
Trento - Italy, February 19-23, 2007

Modeling and Design of Heterogeneous Systems

Modeling of Heterogeneous Systems in Metropolis

Alberto Ferrari

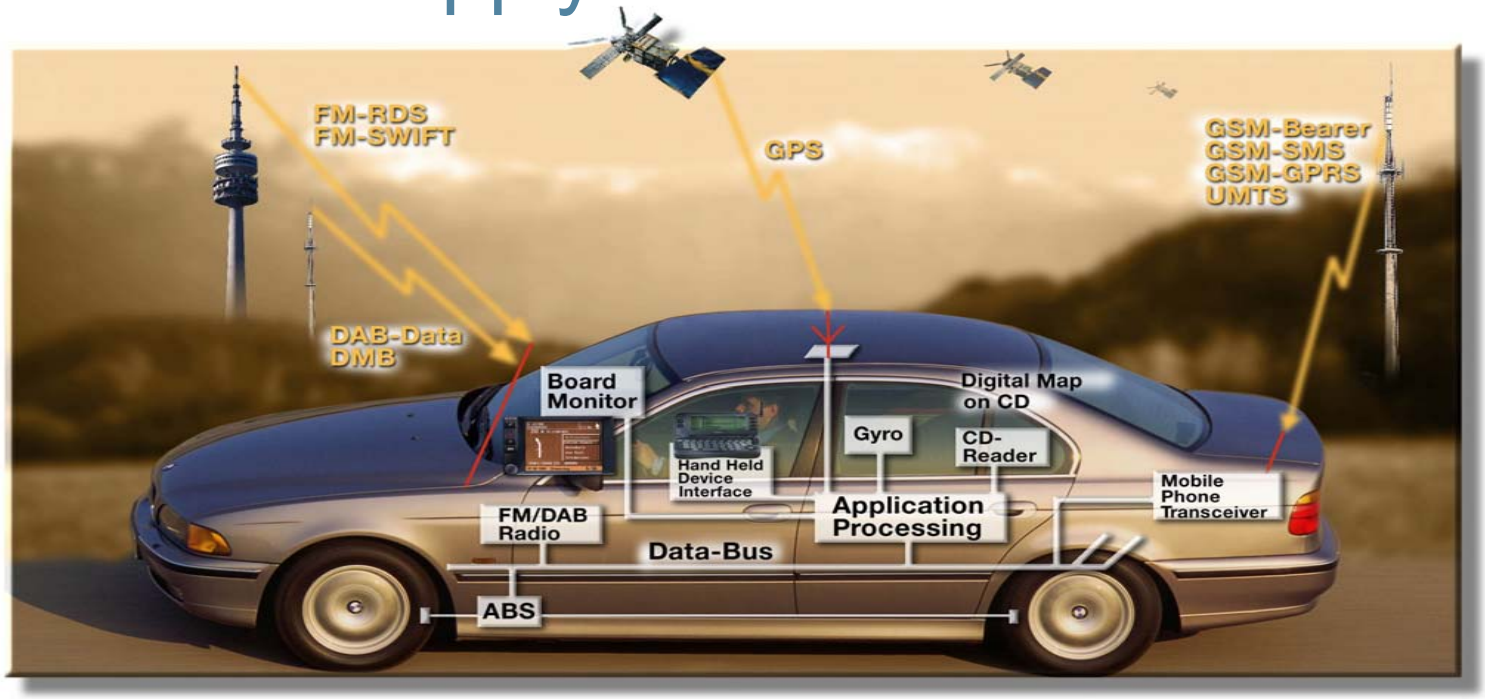
Alberto.Ferrari@parades.rm.cnr.it

PARADES GEIE, Roma, IT

Outline

- Motivation and Industrial Landscape
- Platform Based Design Methodology
- The Metropolis Framework
- The Semantics of the Metropolis Metamodel
- Architecture and Mapping Modeling in Metropolis
- Conclusions

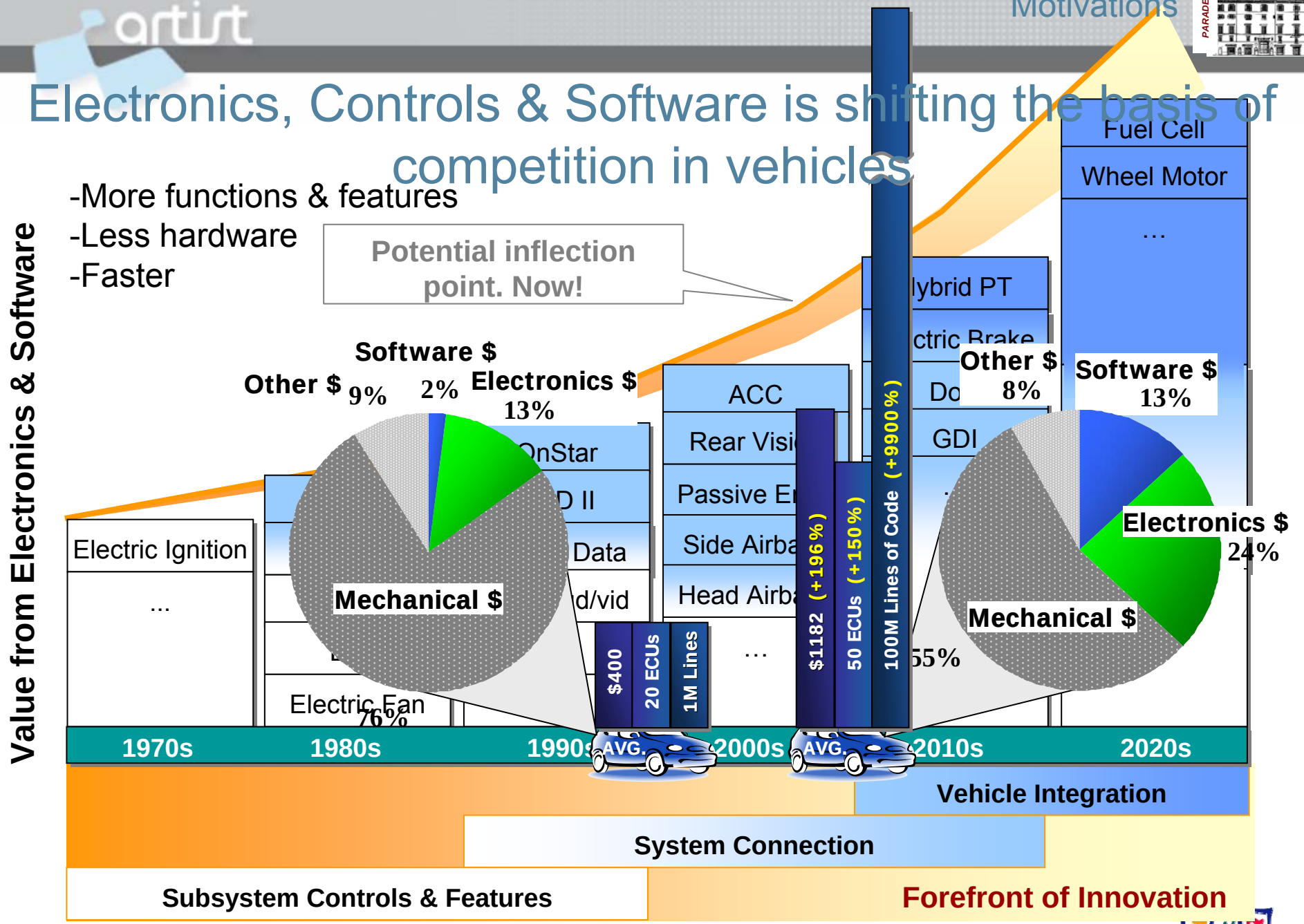
Automotive Supply Chain: Car Manufacturers



- Product Specification & Architecture Definition
  (e.g., determination of Protocols and Communication)
- System Partitioning and Subsystem Specification
- Critical Software Development
- System Integration

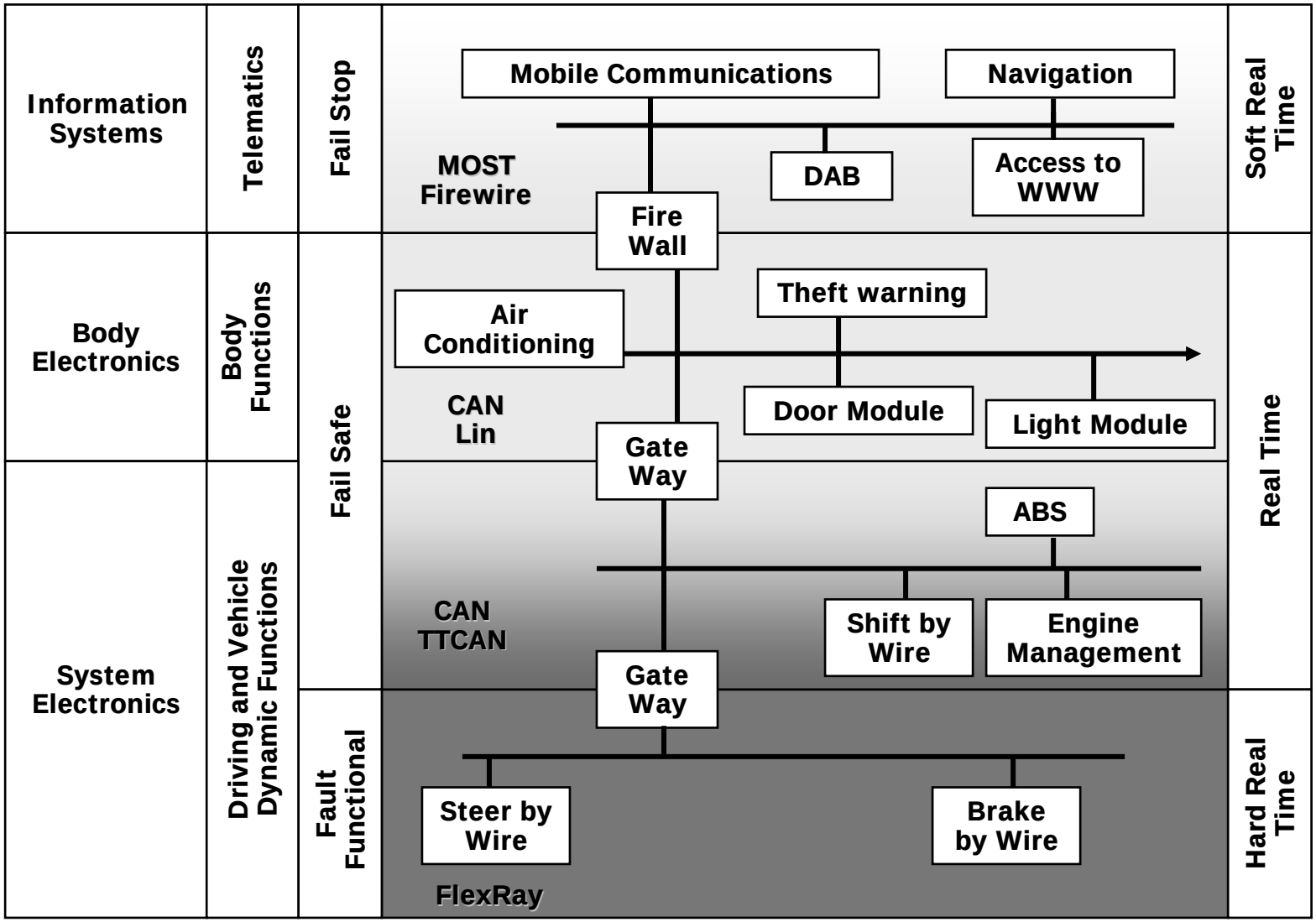
Electronics, Controls & Software is shifting the basis of competition in vehicles

-More functions & features
-Less hardware
-Faster



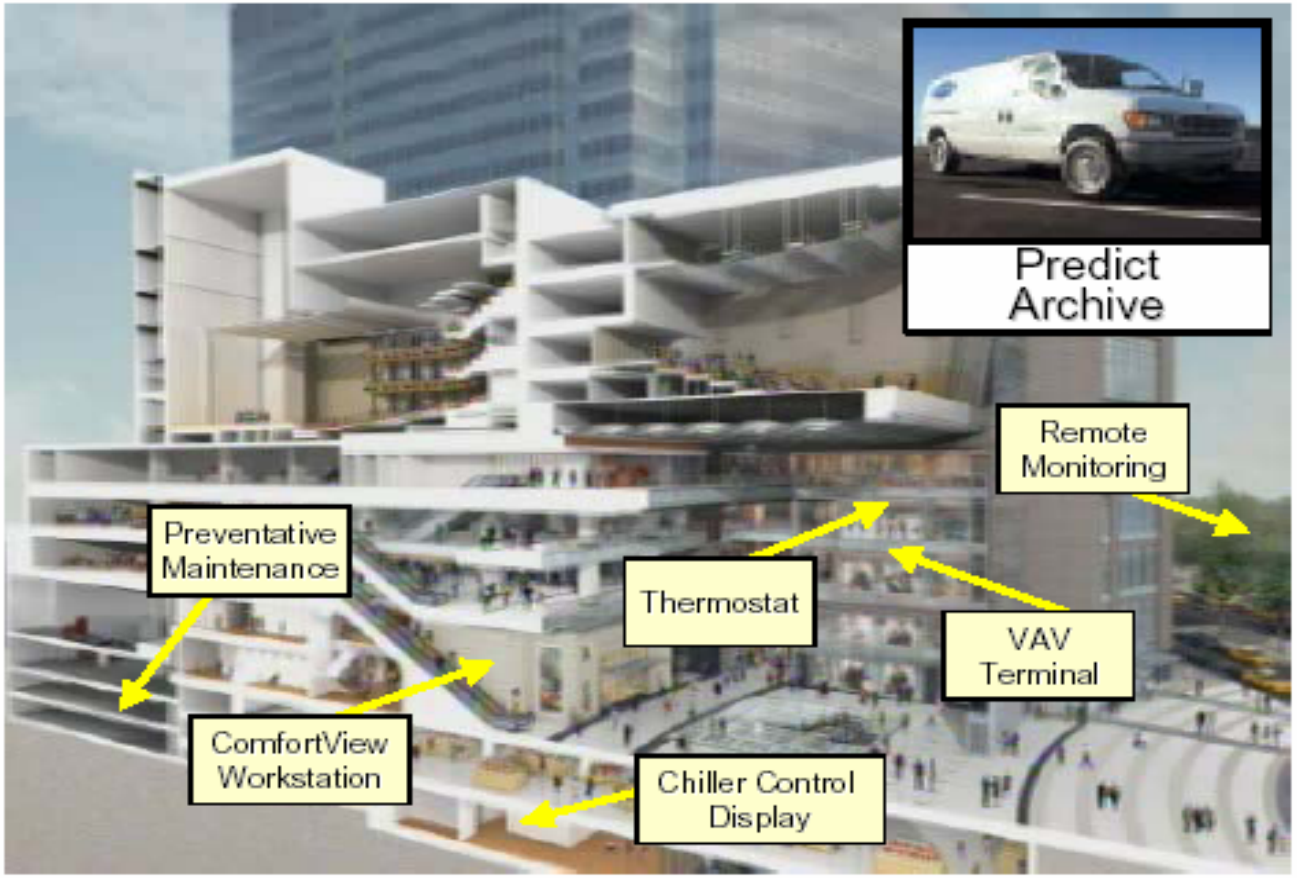
Source: Matt Tsien, GM

Real-Time and Distributed



FUNCTION OF CONTROLS

Typical commercial HVAC application



Configure

Sense

Actuate

Regulate

Display

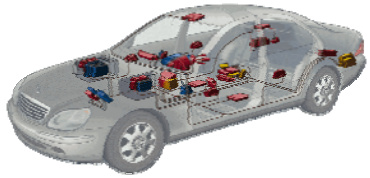
Trend

Diagnose

Predict

Archive

Platform Models for Model Based Development



Development of Distributed System

Distributed System Requirements

Distributed System Partitioning

Sub-Systems *Model Based* Development

Sub-Systems (s) Requirements

Network Protocol Requirements

Sub-System(s) Sign-Off!

Platform Abstraction

Distributed System Sign-Off!

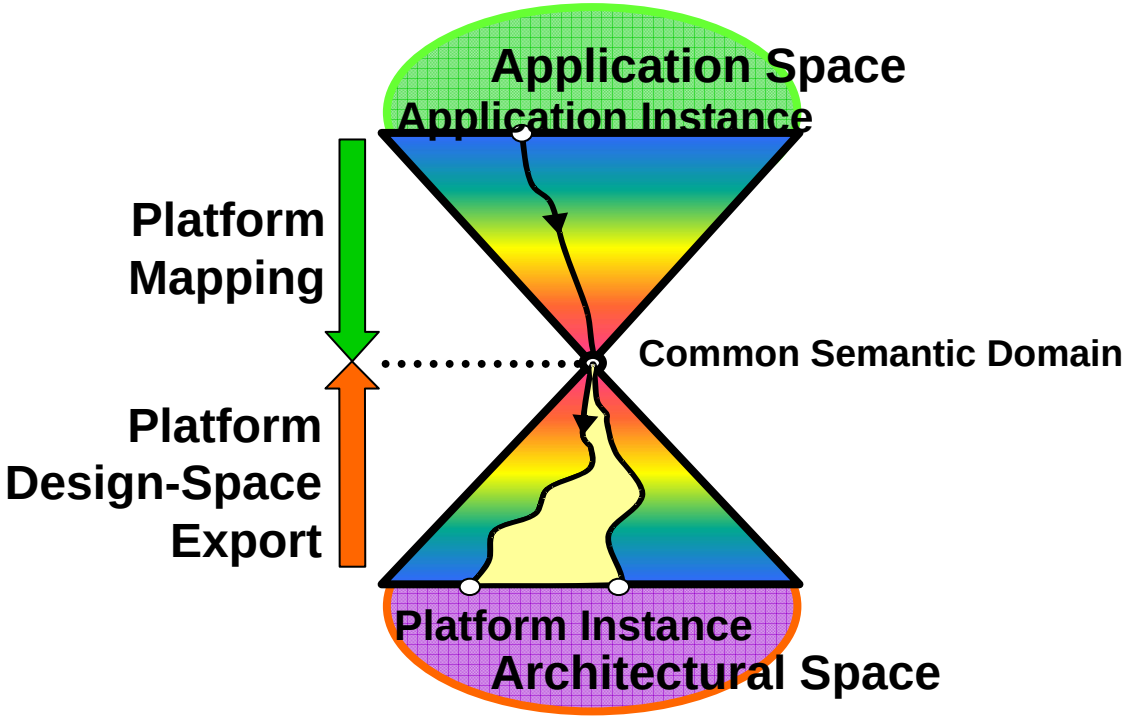
Sub-System(s) Integration, Test, and Validation

Virtual Integration of Sub-System(s) w/ Network Protocol, Test, and Validation

Sub-System(s) Implementation Models Sign-Off!

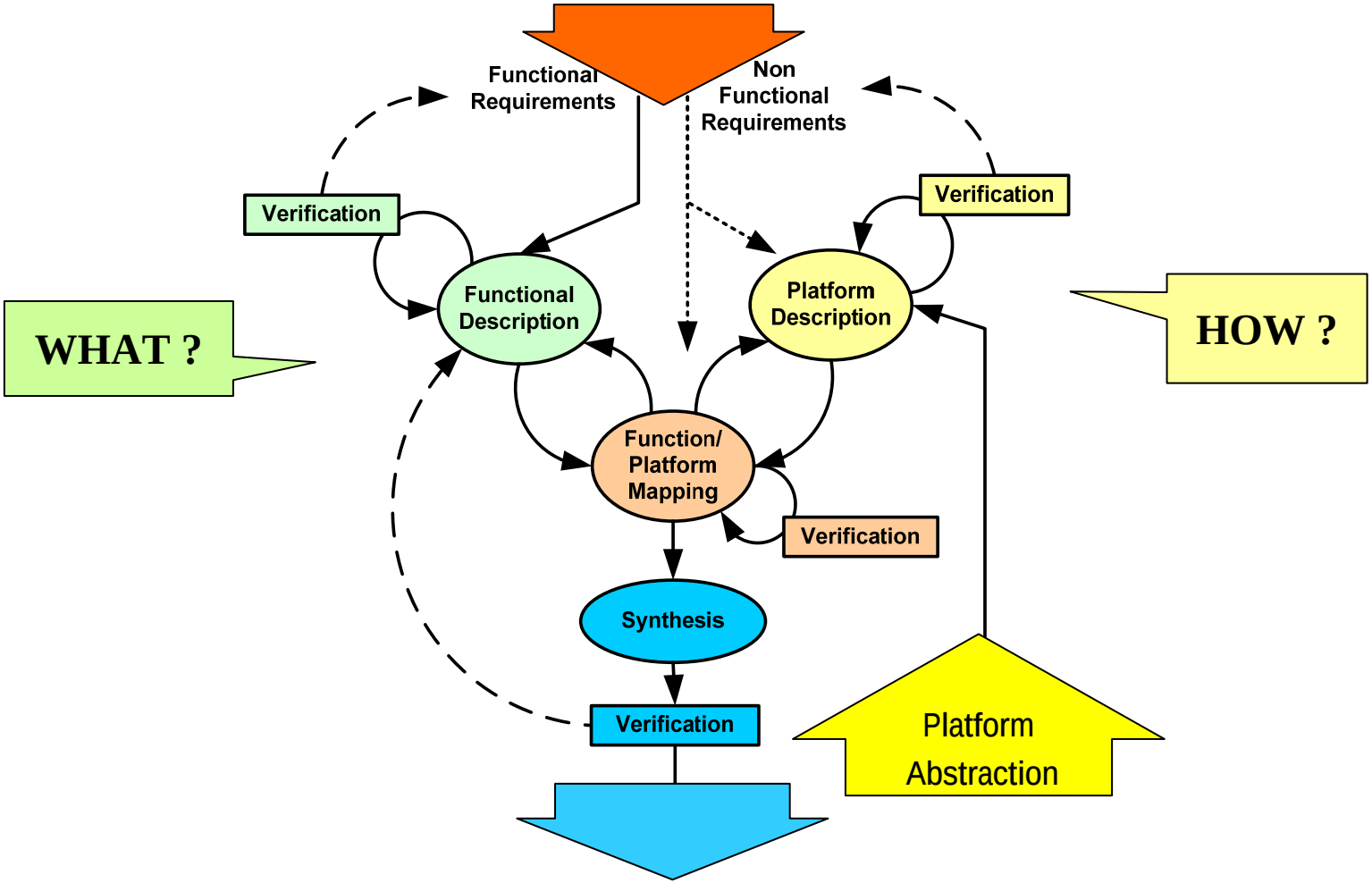
Network Communication Protocol Sign-Off!

Platform-based Design



- **Platform: library of resources defining an abstraction layer**
  - **hide unnecessary details**
  - **expose only relevant parameters for the next step**

Meet-in-the-middle



Putting it all together….CHALLENGE and Opportunity!

- We need an integration design platform
  - To deal with heterogeneity:
    - Where we can deal with Hardware and Software (verification and synthesis)
    - Where we can mix digital and analog
    - Where we can assemble internal and external IPs
    - Where we can work at different levels of abstraction
  - To handle the design chain
  - To support integration
    - e.g. tool integration
    - e.g. IP integration
- The integration platform must subsume the traditional design flow, rather than displacing it

Metropolis: an Environment for System-Level Design

- Motivation
  - Design complexity and the need for verification and time-to-market constraints are increasing
  - Semantic link between specification and implementation is necessary
- Platform-Based Design
  - Meet-in-the-middle approach
  - Separation of concerns
    - Function vs. architecture
    - Capability vs. performance
    - Computation vs. communication
- Metropolis Framework
  - Extensible framework providing simulation, verification, and synthesis capabilities
  - Easily extract relevant design information and interface to external tools
- Released Sept. 15th, 2004

Metropolis Project Big Picture: Target and Goals

Target: Embedded System Design

- **Set-top boxes, cellular phones, automotive controllers, …**
- Heterogeneity:
  - **computation: Analog, ASICs, programmable logic, DSPs, ASIPs, processors**
  - **communication: Buses, cross-bars, cache, DMAs, SDRAM, …**
  - **coordination: Synchronous, Asynchronous (event driven, time driven)**

Goals:

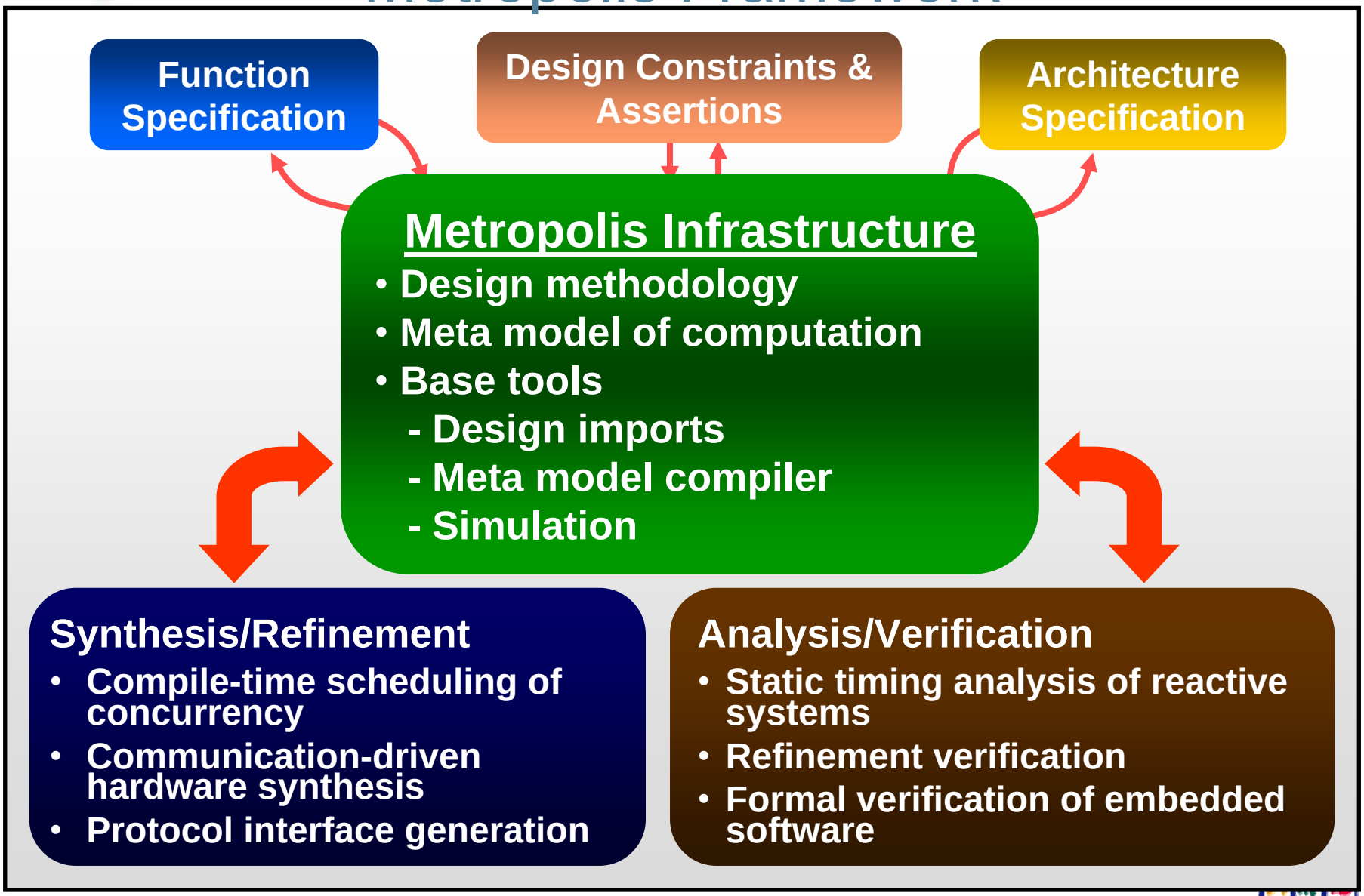
- Design methodologies:
  - **abstraction levels: design capture, mathematics for the semantics**
  - **design tasks: cache size, address map, SW code generation, RTL generation, …**
- Tool set:
  - **synthesis: data transfer scheduling, memory sizing, interface logic, SW/HW generation, …**
  - **verification: property checking, static analysis of performance, equivalence checking, …**

Metropolis Project

Participants:

- UC Berkeley **(USA): methodologies, modeling, formal methods**
- CMU **(USA): formal methods**
- Politecnico di Torino **(Italy): modeling, formal methods**
- Universita Politecnica de Catalunya **(Spain): modeling, formal methods**
- Cadence Berkeley Labs **(USA): methodologies, modeling, formal methods**
- PARADES **(Italy): methodologies, modeling, formal methods**
- ST **(France-Italy): methodologies, modeling**
- Philips **(Netherlands): methodologies (multi-media)**
- Nokia **(USA, Finland): methodologies (wireless communication)**
- BWRC **(USA): methodologies (wireless communication)**
- Magneti-Marelli **(Italy): methodologies (power train control)**
- BMW **(USA): methodologies (fault-tolerant automotive controls)**
- Intel **(USA): methodologies (microprocessors)**
- Cypress **(USA): methodologies (network processors, USB platforms)**
- Honeywell **(USA): methodologies (FADEC)**

Metropolis Framework

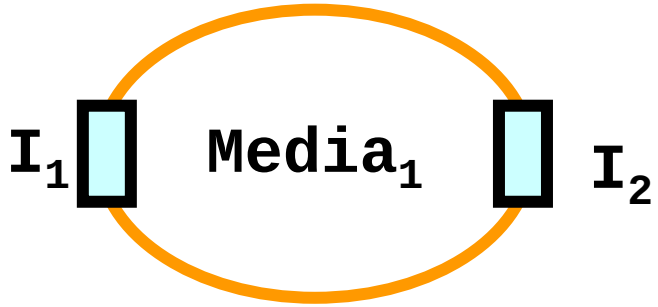


Metropolis Objects

- **Processes (Computation)**

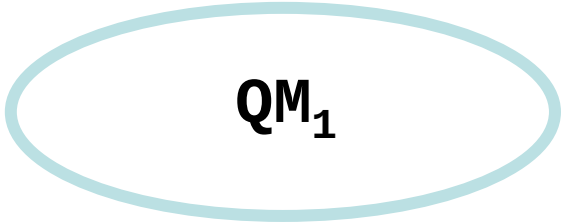
Active Objects
Sequential Executing Thread

- **Media (Communication)**



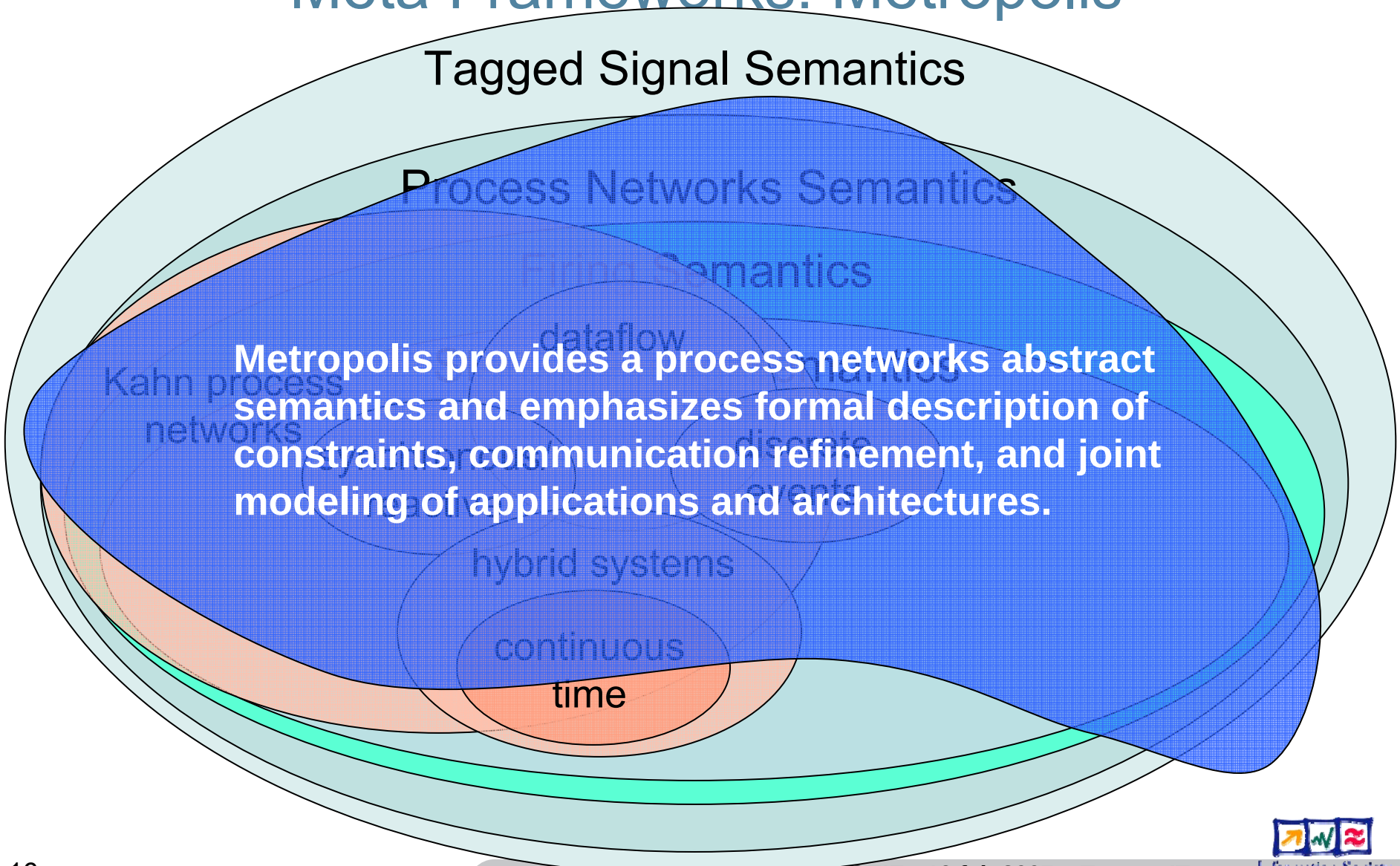
Passive Objects
Implement Interface Services

- **Quantity Managers (Coordination)**



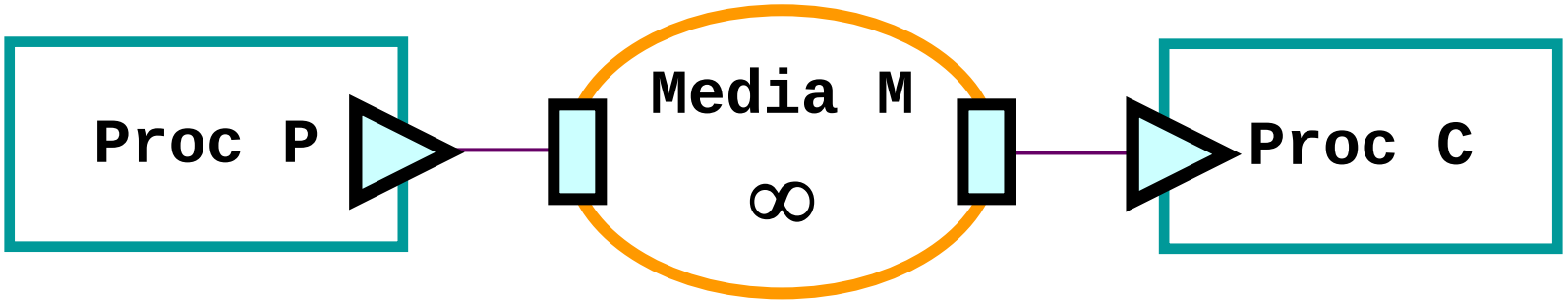
Schedule access to resources and quantities

Meta Frameworks: Metropolis



A Producer–Consumer Example

- A process P producing integers
- A process C consuming integers
- A media M implementing the communication services

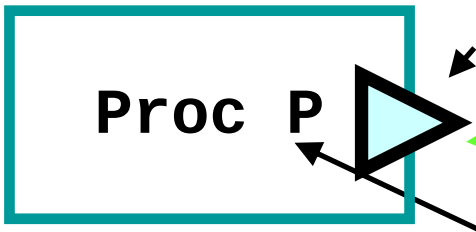


Writer: Process P (Producer)

- Writer.mmm:
  - Port (interface) definition

```
package producers_consumer;
interface IntWriter extends Port{
    update void writeInt(int i);
    eval int nspace();
}
```

- P.mmm:
  - Process behavior definition



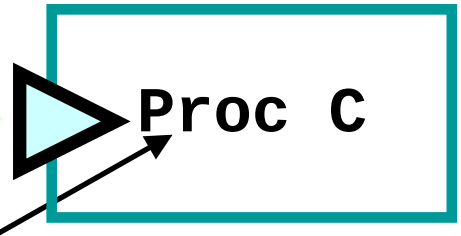

```
package producers_consumer;
process P {
    port IntWriter port_wr;
    public P(String name) {}
    void thread() {
        int w = 0;
        while (w < 30) {
            port_wr.writeInt(w);
            w = w + 1;
        }
}}
```

Reader: Process C (Consumer)

```
package producers_consumer;
interface IntReader extends Port{
    update int readInt();
    eval int num();
}
```

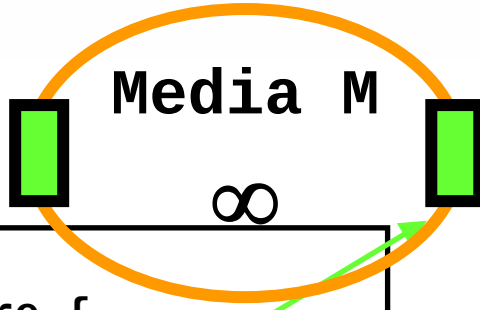
- Reader.mmm:
  - Port (Interface) definition

```
package producers_consumer;
process C {
    port IntReader port_rd;
    public C(String name) { };
    void thread() {
        int r = 0;
        while (r < 30) {
                port_rd.readInt();
        }
}}
```



- C.mmm:
  - Process behavior definition

Media M: Writer services



- Communication definition

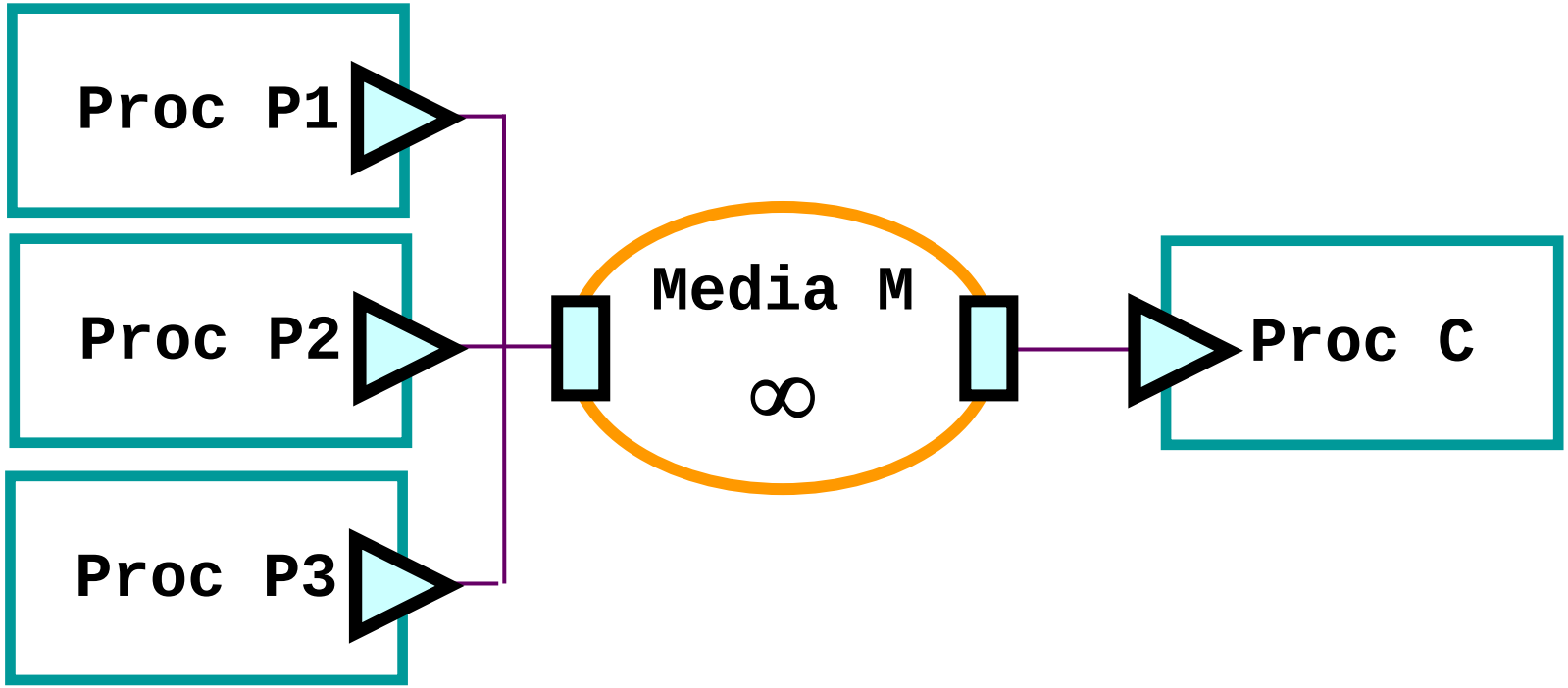
```
package  producers_consumer;
medium IntM implements IntReader, IntWriter, Semaphore {
    int_infinite_queue storage;

    public IntM(String name) { }

    public update void writeInt(int w) {
        await( true; this.Semaphore; this.Semaphore) {
               storage.write(w);
         }
    }
    public update int readInt() {
        int _retval = 0;
        await( storage.getN()>0; this.Semaphore; this.Semaphore){
               _retval = storage.read();
        }
        return _retval;
    }
}
```

Example: producers-consumers

- A set of processes “Writer” write data to a FIFO
- A set of processes “Reader” read data from a FIFO
- An unbounded FIFO is used to store written data and to retrieve read data



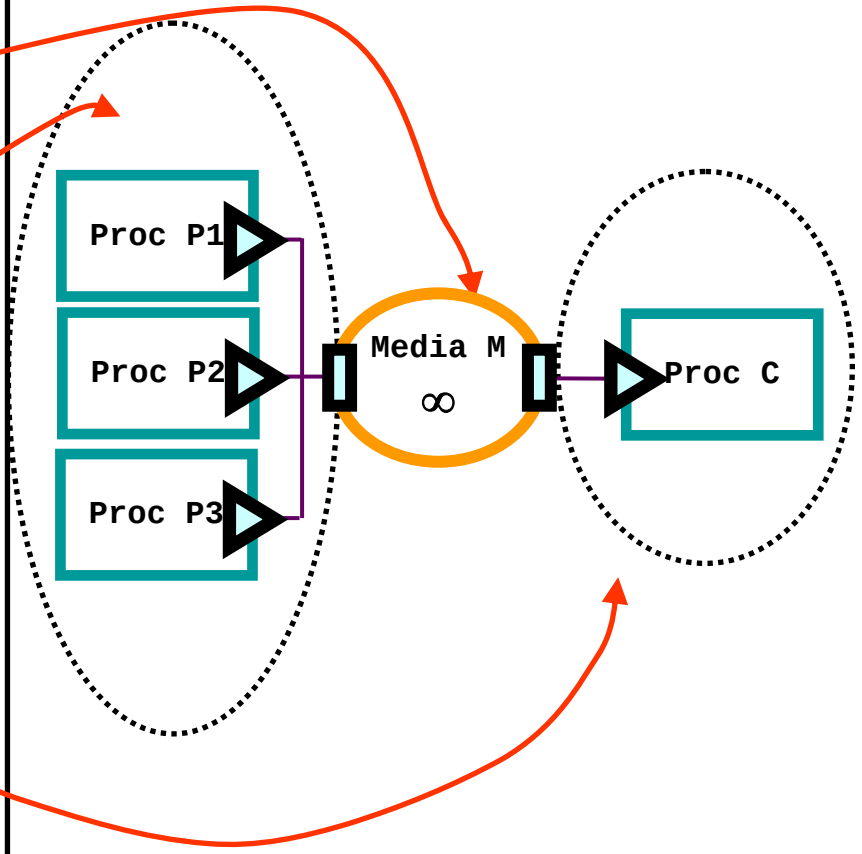
Top level netlist

```
package producers_consumer;

public netlist IwIr {
  IntM m;
  public IwIr(String name) {
    m = new IntM("InstIntM");
    addcomponent(m, this, "MEDIUM");

    for (int i = 0; i < 3; i++) {
        P p = new P("Producer"+i);
        addcomponent(p, this, "Producer"+i);
        connect(p, port_wr, m);
    }

    for (int i = 0; i < 1; i++) {
        C c = new C("Consumer"+i);
        addcomponent(c, this, "Consumer"+i);
        connect(c, port_rd, m);
    }
  }
}
```



Protecting against concurrency

- If more than one process is calling the services, the services can be executed concurrently, hence the state of the media must be protected by concurrent access

- The protection is guaranteed by the await statement

Media M: await

- Communication definition

```
public update void writeInt(int w) {
        await { (true; this.Semaphore; this.Semaphore){
                storage.write(w);
        }
    }

public update int readInt() {
        int _retval = 0;
        await( storage.getN()>0; this.Semaphore; this.Semaphore){
                _retval = storage.read();
        }
        return _retval;
    }

}
```

AWAIT

- The await statement is used to describe a situation that a process object waits for a given condition to become true, and once the condition holds, it continues its execution with the guarantee that the condition still holds

```
await  {
        ( guard, testlist, setlist) statements;
}
```

Logic of Constraints (LOC)

- A transaction-level quantitative constraint language
- Works on a sequence of events from a particular execution trace

Throughput: "at least 3 *Display* events will be produced in any period of 30 time units".

$$t(Display[i+3]) - t(Display[i]) <= 30$$

Other LOC constraints

Performance: rate, latency, jitter, burstiness

Functional: data consistency

Key Modeling Concepts

- An event is the fundamental concept in the framework
  - Represents a transition in the action automata of an object
  - An event is owned by the object that exports it
  - During simulation, generated events are termed as *event instances*
  - Events can be annotated with any number of quantities
  - Events can partially expose the state around them, constraints can then reference or influence this state
- A service corresponds to a set of sequences of events
  - All elements in the set have a common begin event and a common end event
  - A service may be parameterized with arguments

1. E. Lee and A. Sangiovanni-Vincentelli, *A Unified Framework for Comparing Models of Computation*, IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, Vol. 17, N. 12, pg. 1217-1229, December 1998

Action Automata

- **Processes take *actions*.**
  - **statements and some expressions, e.g.**

    y = z+port.f();, z+port.f(), port.f(), i < 10, …
  - **only calls to media functions are *observable actions***

- **An *execution* of a given netlist is a sequence of vectors of *events*.**
  - ***event* : the beginning of an action, e.g. B(port.f()),**

    the end of an action, e.g. E(port.f()), or null N
  - **the *i*-th component of a vector is an event of the *i*-th process**

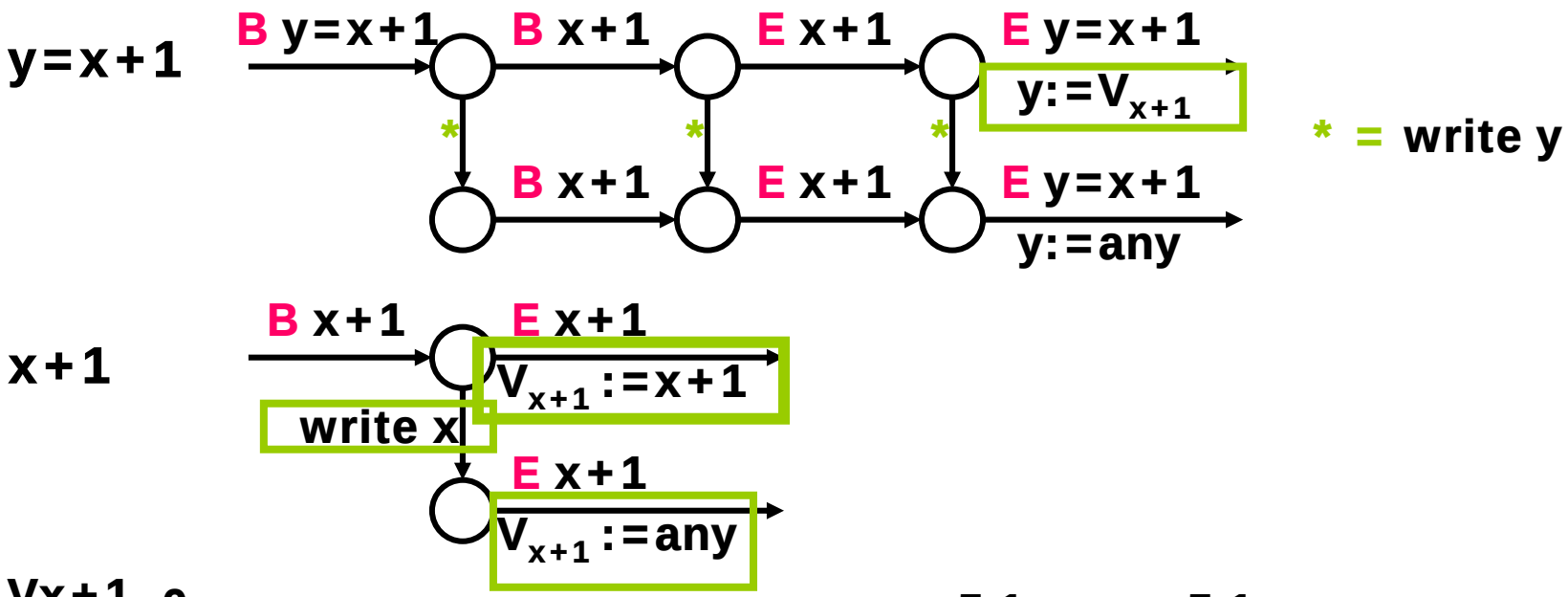
- **An execution is *legal* if**
  - **it satisfies all coordination constraints, and**
  - **it is accepted by all action automata.**

Execution semantics

Action automaton:

- one for each action of each process
  - **defines the set of sequences of events that can happen in executing the action**
- a transition corresponds to an event:
  - **it may update shared memory variables:**
    - process and media member variables
    - values of actions-expressions
  - **it may have guards that depend on states of other action automata and memory variables**
- each state has a self-loop transition with the null N event.
- all the automata have their alphabets in common:
  - **transitions must be taken together in different automata, if they correspond to the same event.**

Action Automata for “y=x+1”



Return

Semantics summary

- Processes run sequential code concurrently, each at its own arbitrary pace.
- Read-Write and Write-Write hazards may cause unpredictable results
  - **atomicity has to be explicitly specified.**
- Progress may block at synchronization points
  - **awaits**
  - **function calls and labels to which awaits or constraints refer.**
- The legal behavior of a netlist is given by a set of sequences of event vectors.
  - **multiple sequences reflect the non-determinism of the semantics: concurrency, synchronization (awaits and constraints)**

Metropolis Meta Model

- Do not commit to the semantics of a particular Model of Computation (MoC)
- Define a set of "building blocks":
  - specifications with many useful MoCs can be described using the building blocks.
  - unambiguous semantics for each building block.
  - syntax for each building block ⇨ a language of the meta model.
- Represent behavior at all design phases; mapped or unmapped

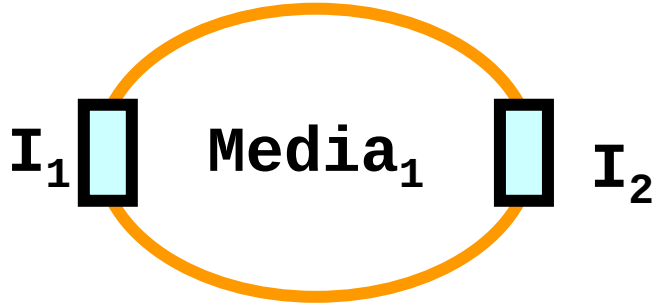
Metropolis Objects: adding quantity managers

- Metropolis elements adhere to a “separation of concerns” point of view.

• **Processes (Computation)**

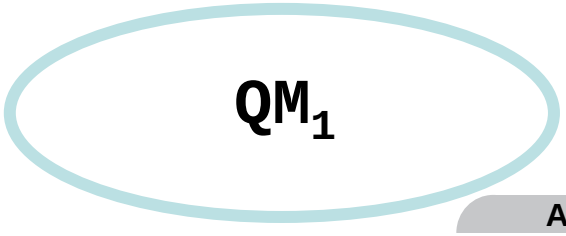
Active Objects
Sequential Executing Thread

• **Media (Communication)**



Passive Objects
Implement Interface Services

• **Quantity Managers (Coordination)**

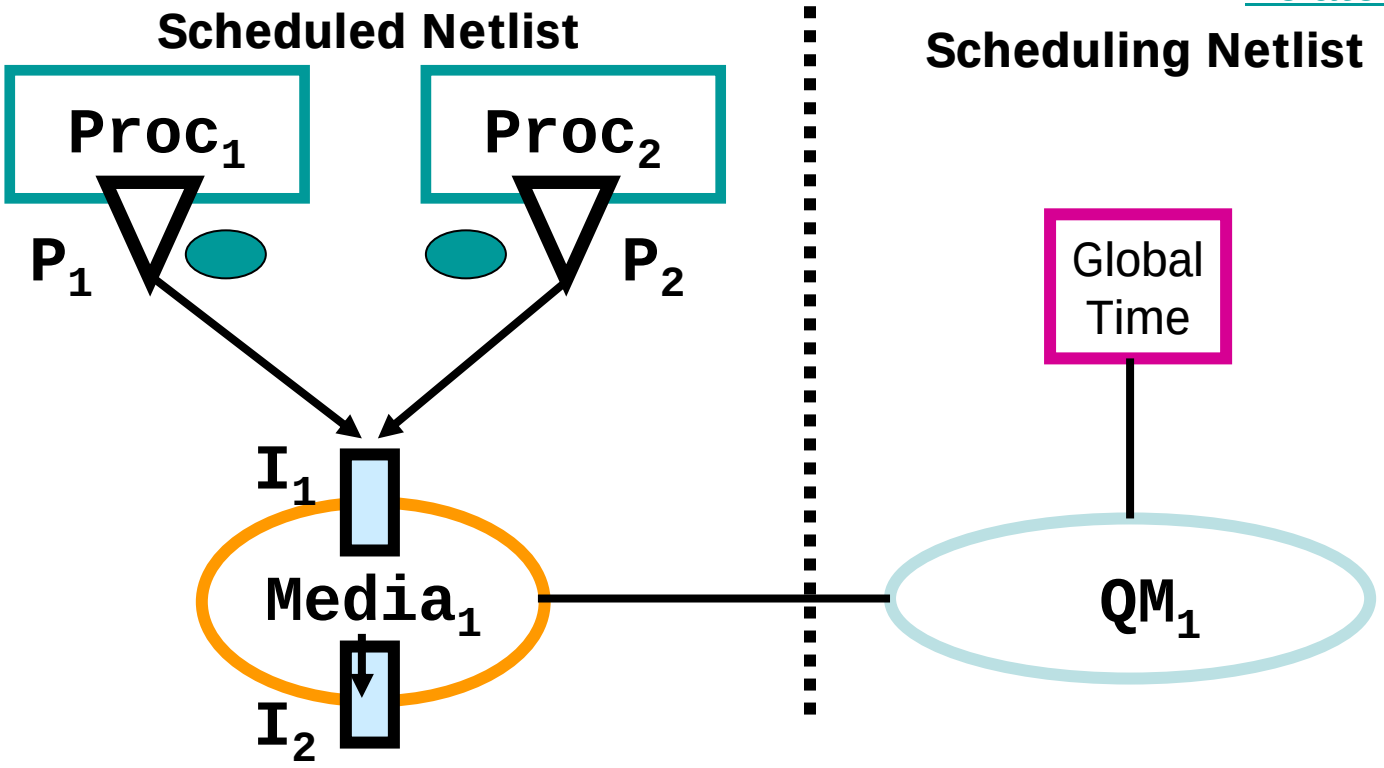


Schedule access to
resources and quantities

Metro. Netlists and Events

Metropolis Architectures are created via two netlists:

- Scheduled – generate **events**[1] for services in the scheduled netlist.
- Scheduling – allow these **events** access to the services and **annotate events** with **quantities**.



Related Work

Event[1] – represents a transition in the action automata of an object. Can be annotated with any number of quantities. This allows performance estimation.

Meta-model: architecture components

An architecture component specifies *services*, i.e.

- **what it *can* do:**

  interfaces, methods, coordination (awaits, constraints), netlists

- **how much it *costs:***

  quantities, annotated with events, related over a set of events

- Which levels of abstraction, what kind of quantities, what kind of cost constraints should be used to capture architecture components?
  - depends on applications: *on-going research*

Programmable Architecture Modeling

Computation Services

PPC405 | MicroBlaze | SynthMaster | SynthSlave

Read (addr, offset, cnt, size), Write(addr, offset, cnt, size),
Execute (operation, complexity)

Communication Services

Processor Local Bus (PLB)

On-Chip Peripheral Bus (OPB)

BRAM

addrTransfer(target, master)
addrReq(base, offset, transType, device)
addrAck(device)

dataTransfer(device, readSeq, writeSeq)
dataAck(device)

Coordination Services

PPC Sched | MicroBlaze Sched | PLB Sched | OPB Sched

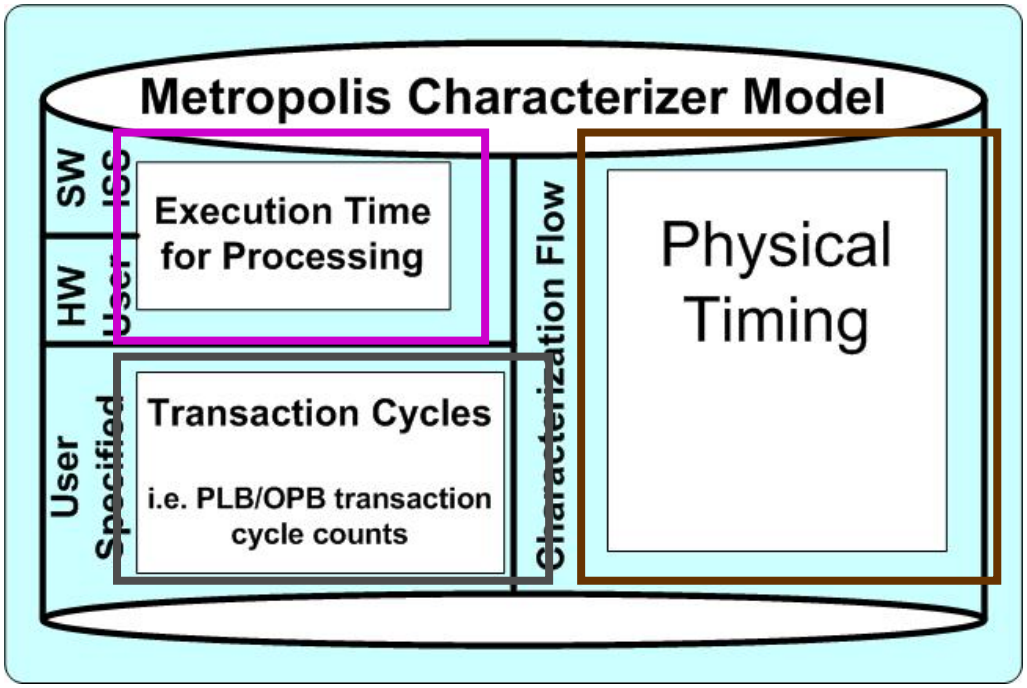
Prog. Platform Characterization

Create database **ONCE** prior to simulation and populate with independent (modular) information.

1. Data detailing performance based on physical implementation.

2. Data detailing the composition of communication transactions.

3. Data detailing the processing elements computation.



From Char Flow Shown

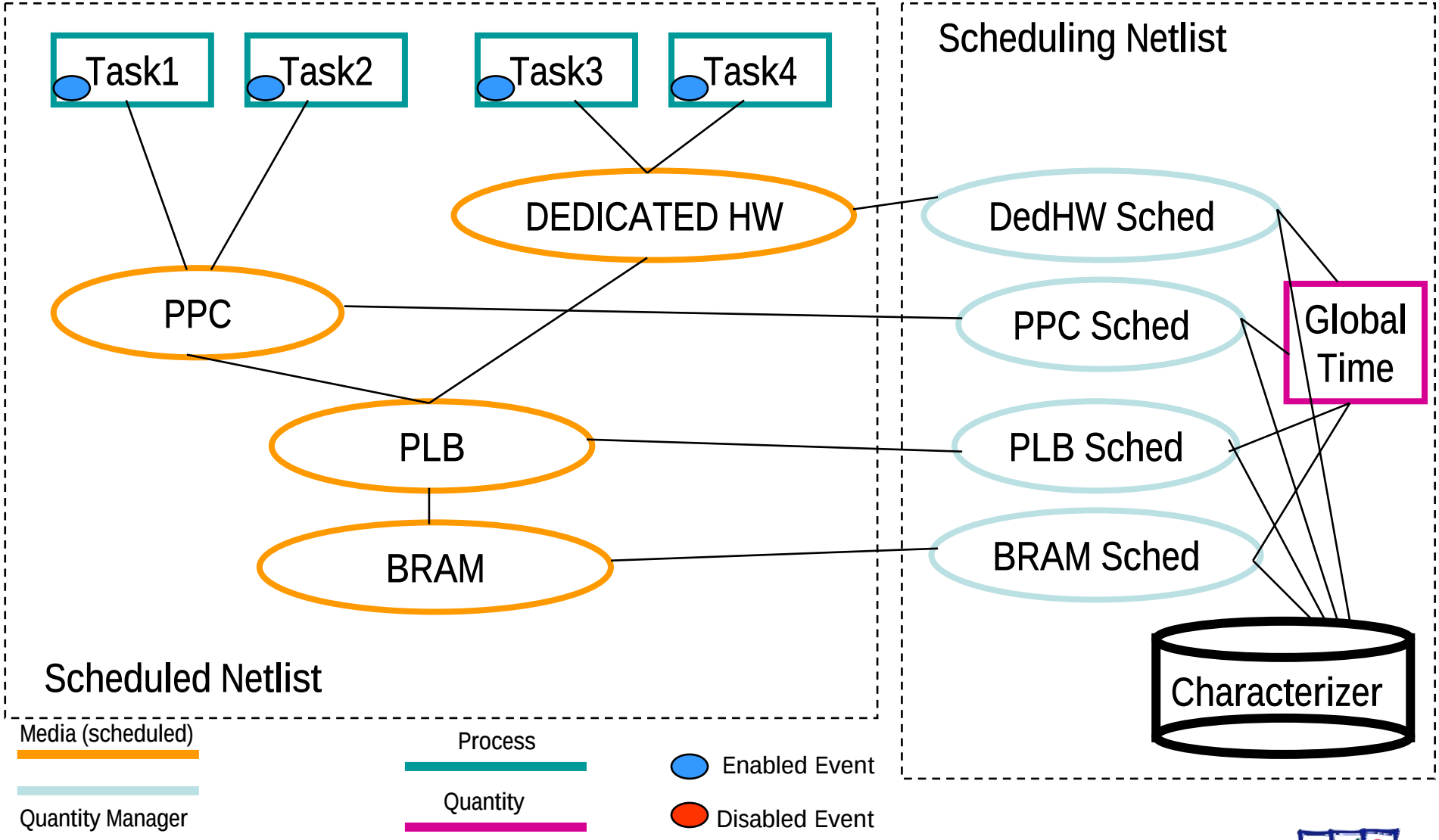
From Metro Model Design

From ISS for PPC

Work with Xilinx Research Labs

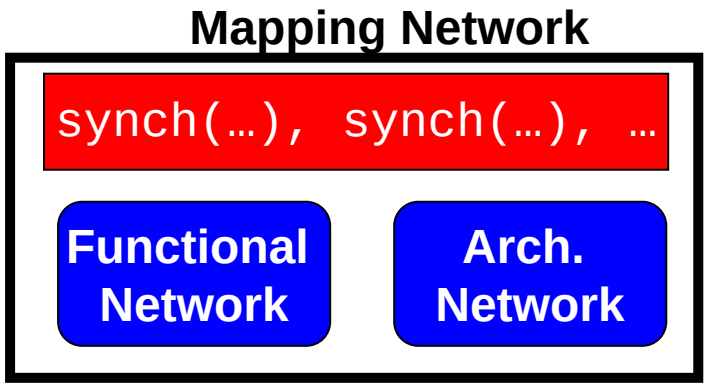
1. Douglas Densmore, Adam Donlin, A.Sangiovanni-Vincentelli, *FPGA Architecture Characterization in System Level Design*, Submitted to CODES 2005.
2. Adam Donlin and Douglas Densmore, *Method and Apparatus for Precharacterizing Systems for Use in System Level Design of Integrated Circuits*, Patent Pending.

Modeling and Characterization Review



Mapping in Metropolis

- Objectives:
  - Map a functional network with an architectural network without changing either of the two
    - Support design reuse
  - Specify the mapping between the two in a formal way
    - Support analysis techniques
    - Make future automation easier
- Mechanism:
  - Use declarative synchronization constraints between events
  - One of the unique aspects of Metropolis



Synchronization constraints

- Synchronization constraint between two events e1 and e2:
  - ltl synch(e1, e2)
  - e1 and e2 occur simultaneously or not at all
- Optional variable equality portion:
  - ltl synch(e1, e2: var1@e1 == var2@e2)
  - The value of *var1* in the scope of e1 is equal to the value of *var2* when e1 and e2 occur
  - Can be useful for “passing” values between functional and architectural models

MyMapNetlist

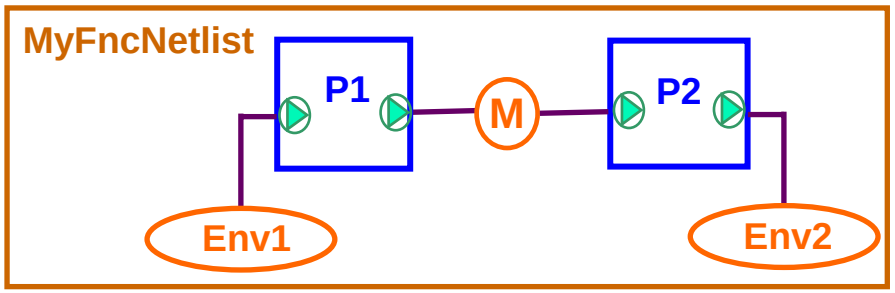
Meta-model: mapping netlist

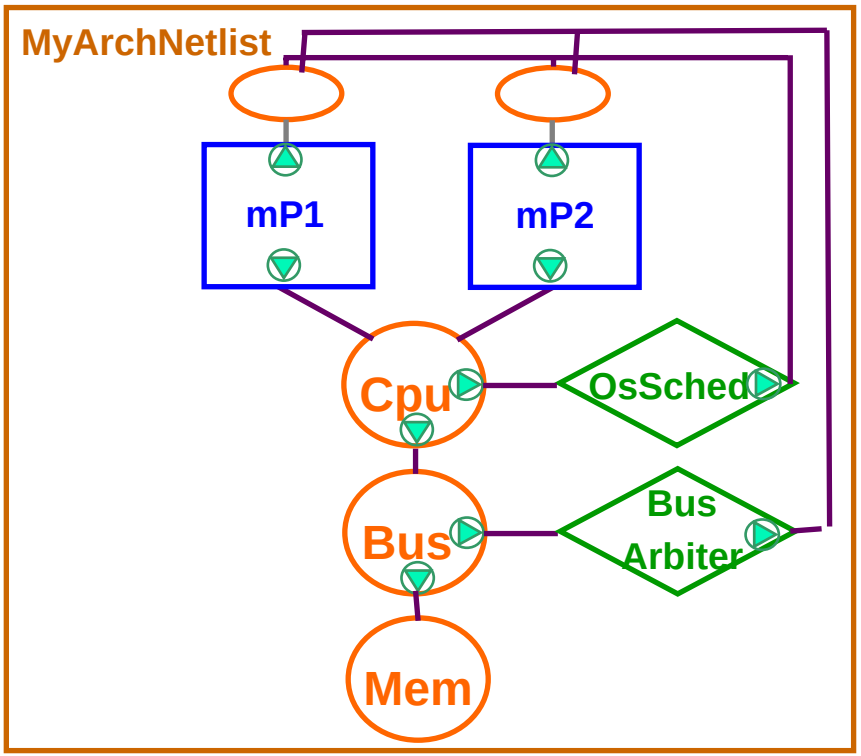
B(P1, M.write) <=> B(mP1, mP1.writeCpu); E(P1, M.write) <=> E(mP1, mP1.writeCpu);

B(P1, P1.f) <=> B(mP1, mP1.mapf); E(P1, P1.f) <=> E(mP1, mP1.mapf);

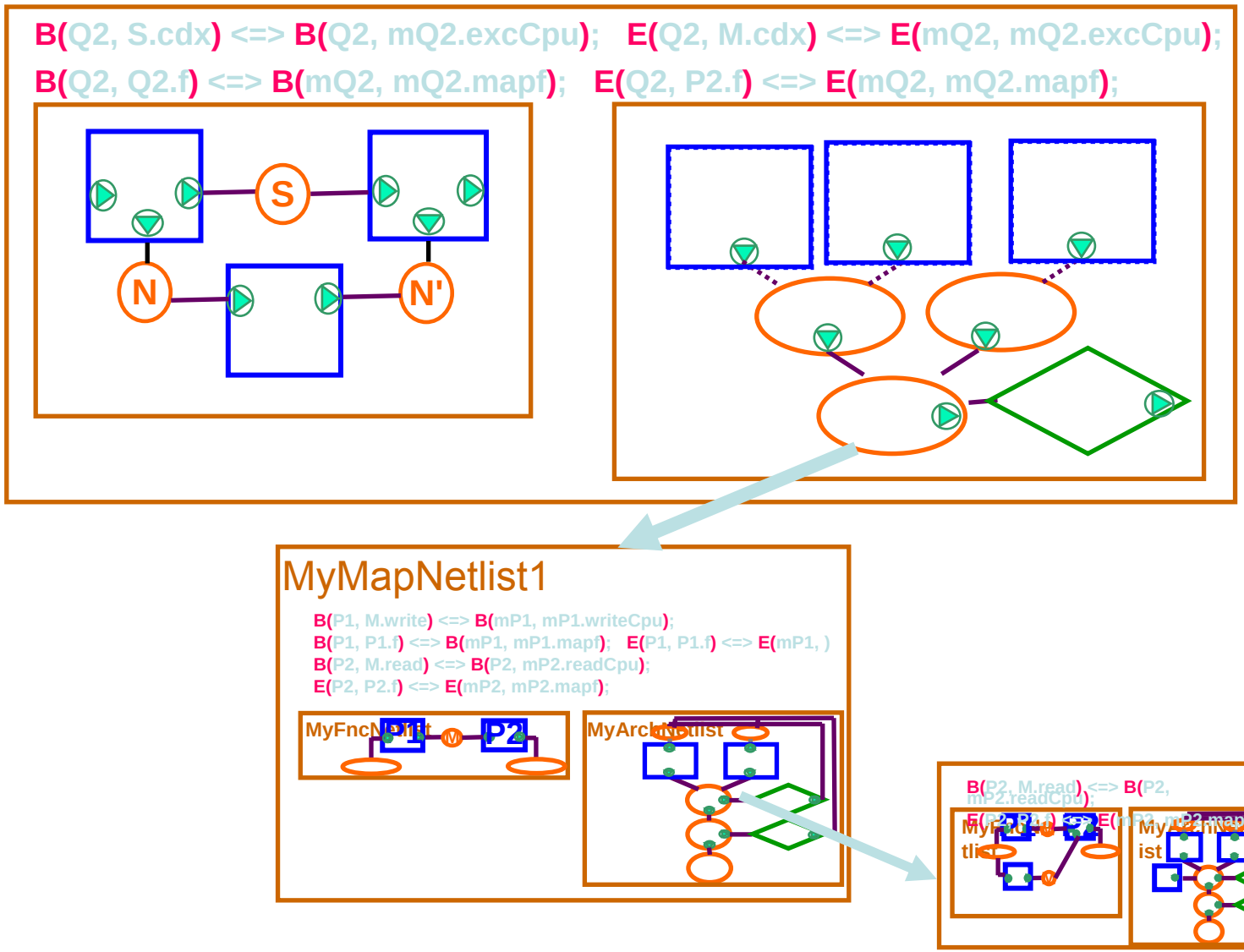
B(P2, M.read) <=> B(P2, mP2.readCpu); E(P2, M.read) <=> E(mP2, mP2.readCpu);

B(P2, P2.f) <=> B(mP2, mP2.mapf); E(P2, P2.f) <=> E(mP2, mP2.mapf);





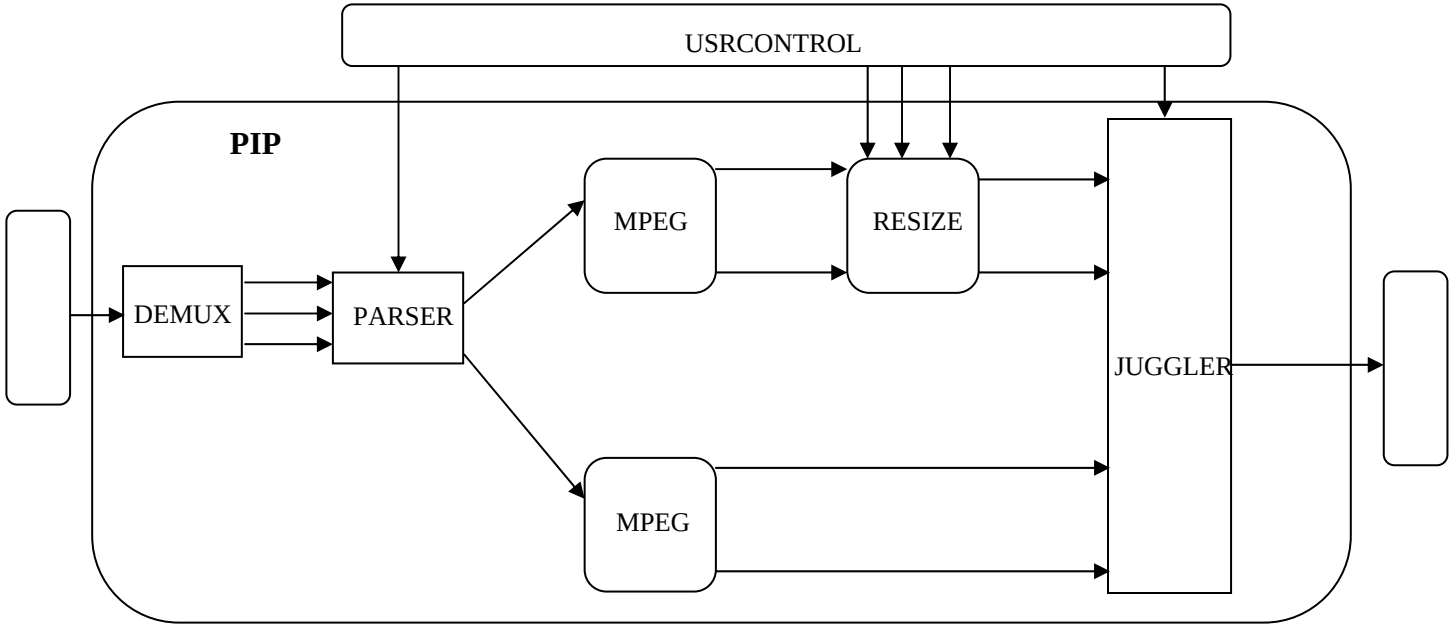
Meta-model: recursive paradigm of platforms



Metropolis Driver: Picture-in-Picture Design Exercise

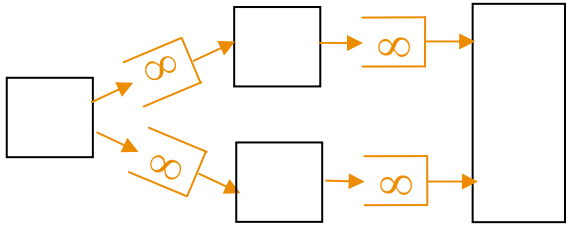
Evaluate the methodology with formal techniques applied.

- Function
  - Input: a transport stream for multi-channel video images
  - Output: a PiP video stream
    - the inner window size and frame color dynamically changeable

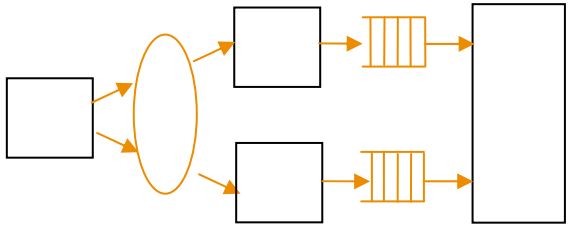


60 processes with 200 channels

Multi-Media System: Abstraction Levels

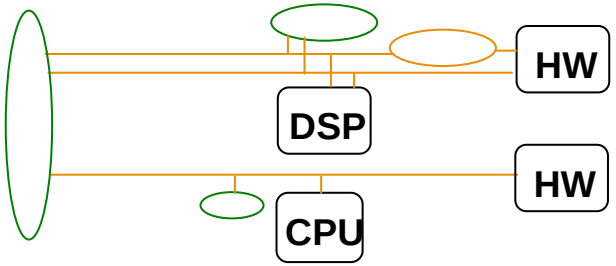


- Network of processes with sequential program for each
- Unbounded FIFOs with multi-rate read and write

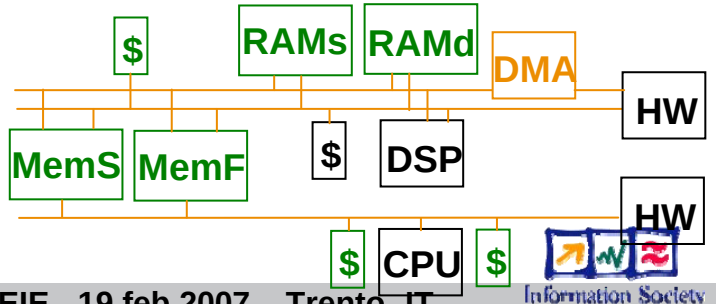


- Communication refined to bounded FIFOs and shared memories with finer primitives (called TTL API):
  allocate/release space, move data, probe space/data

- Mapped to resources with coarse service APIs
- Services annotated with performance models
- Interfaces to match the TTL API

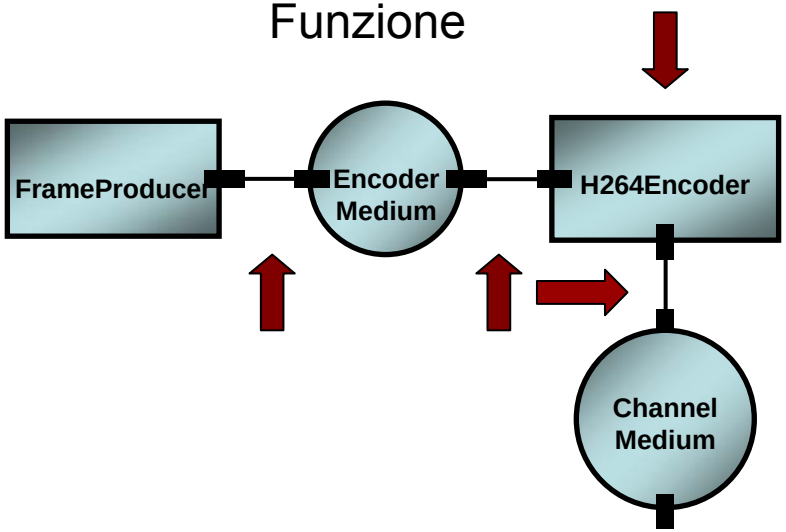


- Cycle-accurate services and performance models



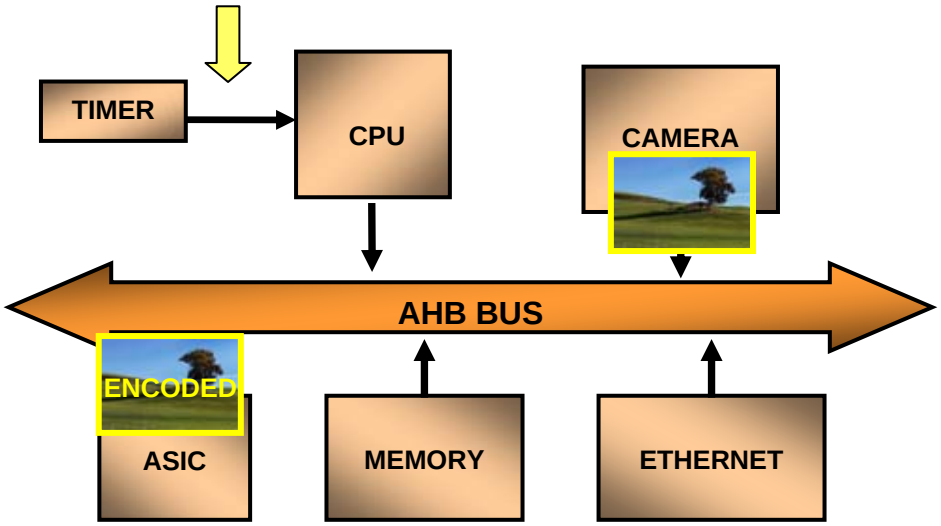
H264 - Mapping

Funzione

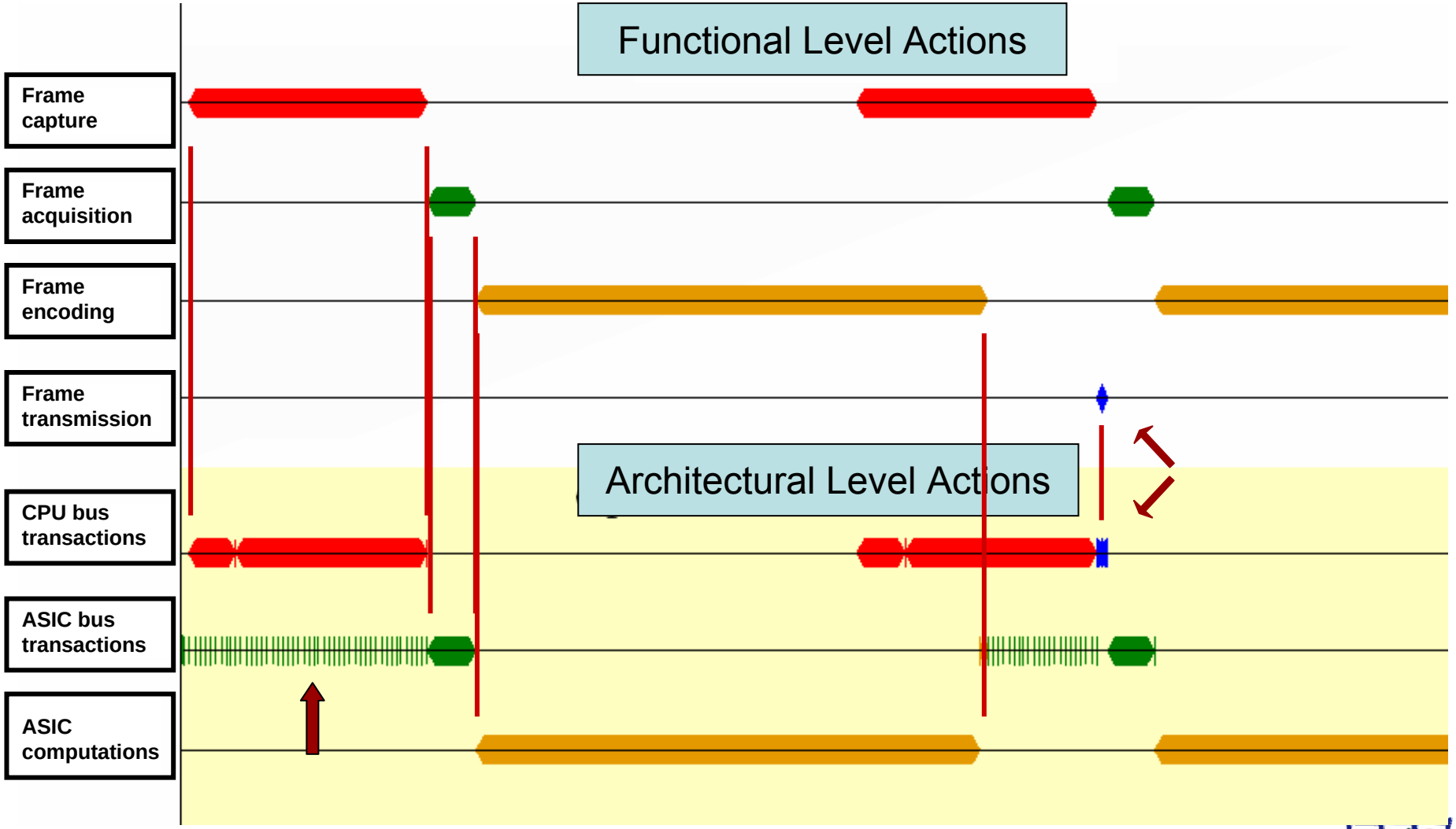


- Frame capture
- Frame acquisition
- Frame coding
- Frame transmission

Architettura

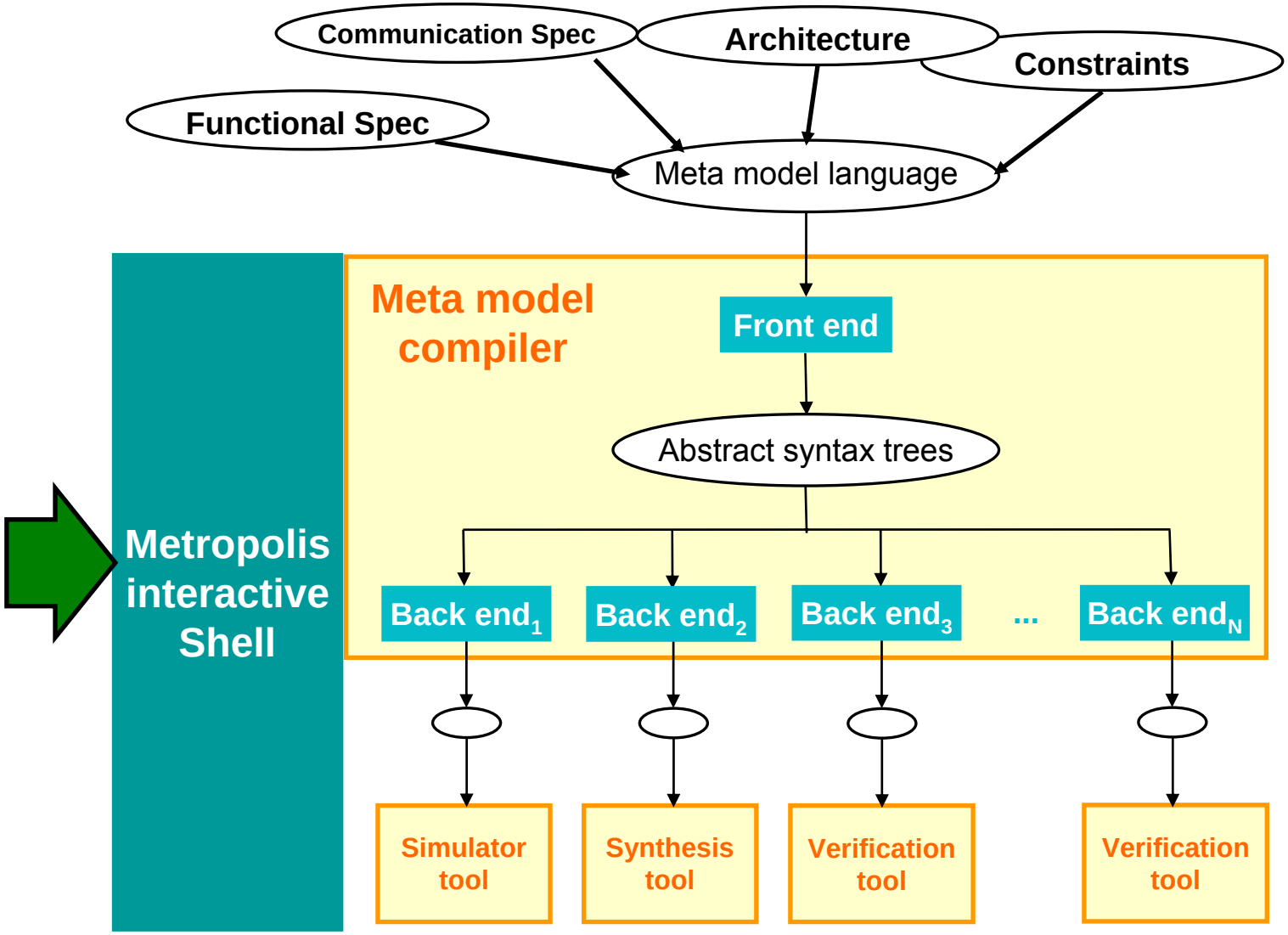


Netlist Synchronization



Metropolis design environment

- Load designs
- Browse designs
- Relate designs
  refine, map etc
- Invoke tools
- Analyze results



Backend Point Tools

- **Synthesis/refinement:**
  - **Quasi-static scheduling**
  - **Scheduler synthesis from constraint formulas**
  - **Interface synthesis**
  - **Refinement (mapping) synthesis**
  - **Architecture-specific synthesis from concurrent processes for:**
    - Hardware (with known architecture)
    - Dynamically reconfigurable logic
- **Verification/analysis:**
  - **Static timing analysis for reactive processes**
  - **Invariant analysis of sequential programs**
  - **Refinement verification**
  - **Formal verification for software**

Conclusions: a Metropolis Summary

- Concurrent specification with a formal execution semantics
- Feasible executions of a netlist: sequences of event vectors
- Quantities can be defined and annotated with events, e.g.
  - Time, power, global indices, composite quantities.
- Concurrent events can be coordinated in terms of quantities:
  - logic can be used to define the coordination,
  - algorithms can be used to implement the coordination.
- The mechanism of event coordination wrt quantities plays a key role:
  - architecture modeling as service with cost,
  - a mapping coordinates executions of function and architecture netlists,
  - a refinement through event coordination provides a platform.
- Metropolis design environment:
  - meta-model compiler to provide an API to browse designs,
  - backend tools to analyze designs and produce appropriate models.

Acknowledgments

- Prof. A.Sangiovanni-Vincentelli
- PARADES EEIG:
  - L.Mangeruca, M.Baleani, F.Carnevale
- UCB: the entire Metropolis Team
  - http://www.gigascale.org/metropolis/index.html