Time for specification of embedded systems

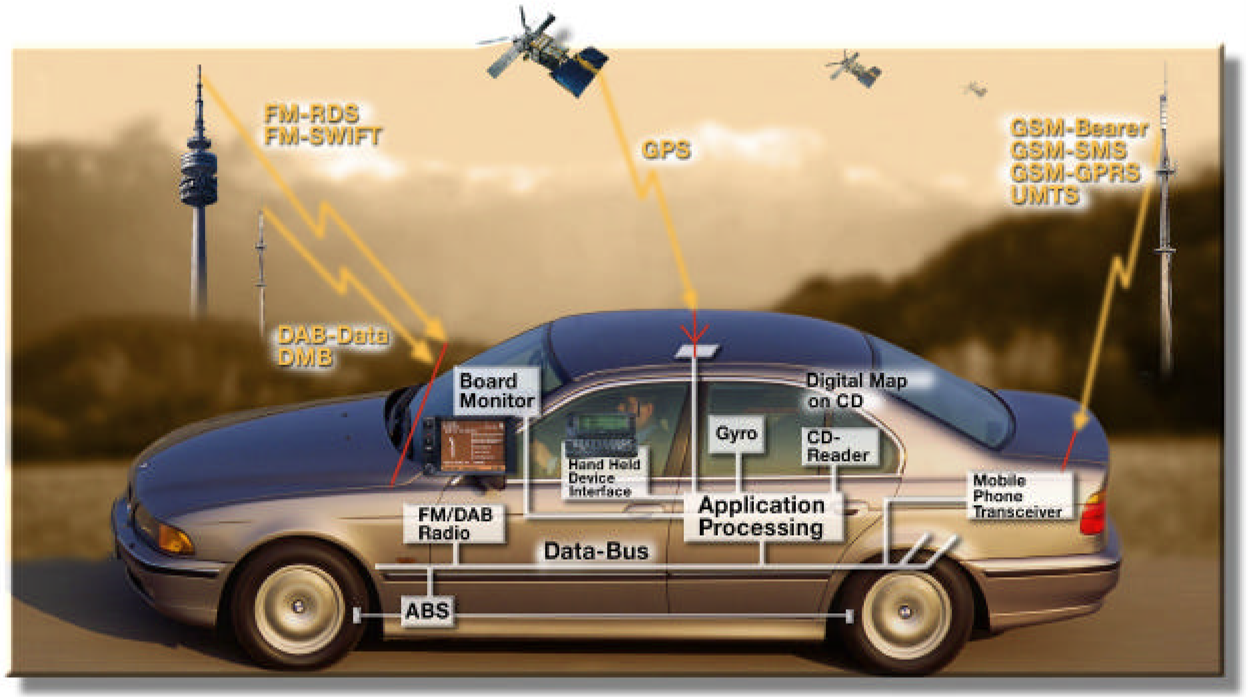
Felice Balarin

Cadence Berkeley Labs

OUTLINE

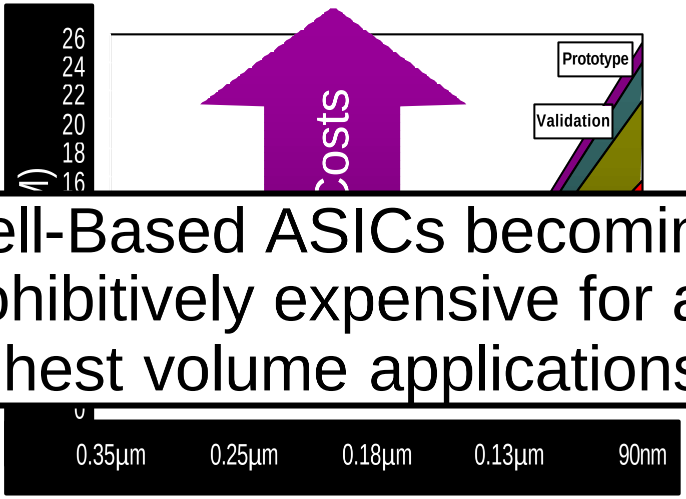
- Embedded systems challenge
- Metropolis project
- Representing time
- Representing timing requirements
- Relation to UML

Automotive Supply Chain:
Car Manufacturers

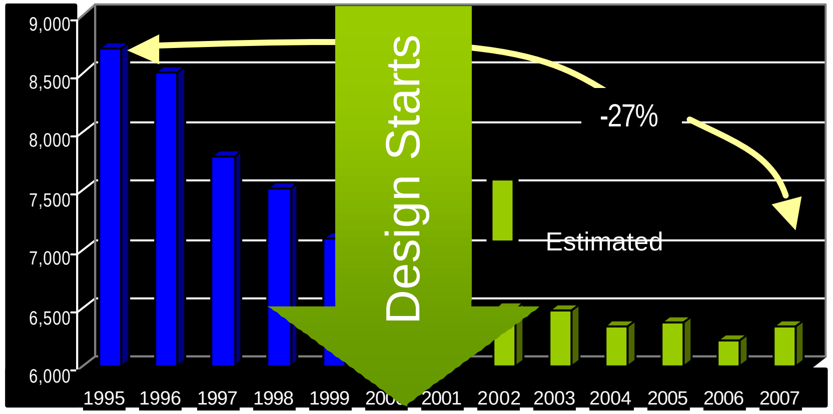


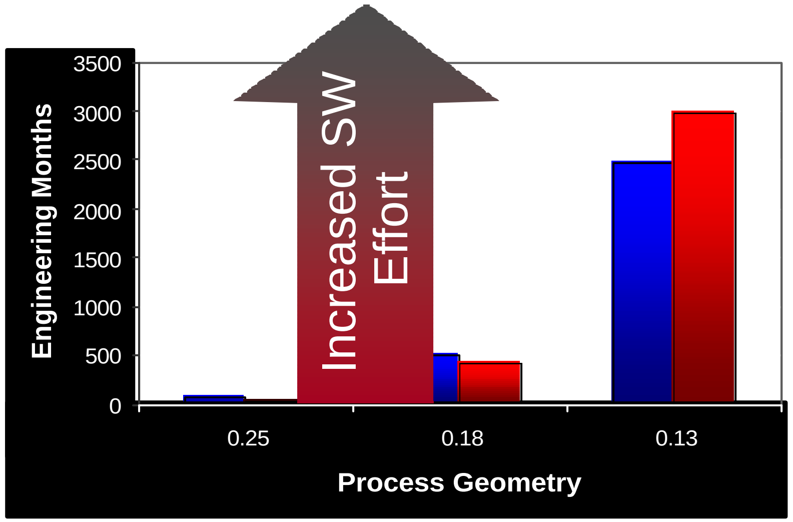
- Product Specification & Architecture Definition
  (e.g., determination of Protocols and Communication standards)
- System Partitioning and Subsystem Specification
- Critical Software Development
- System Integration

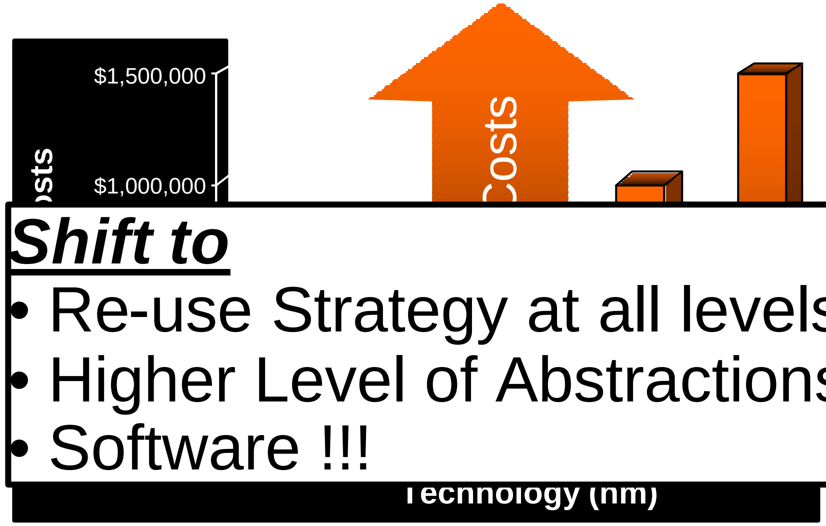
Challenges and Trends



Cell-Based ASICs becoming prohibitively expensive for all but highest volume applications

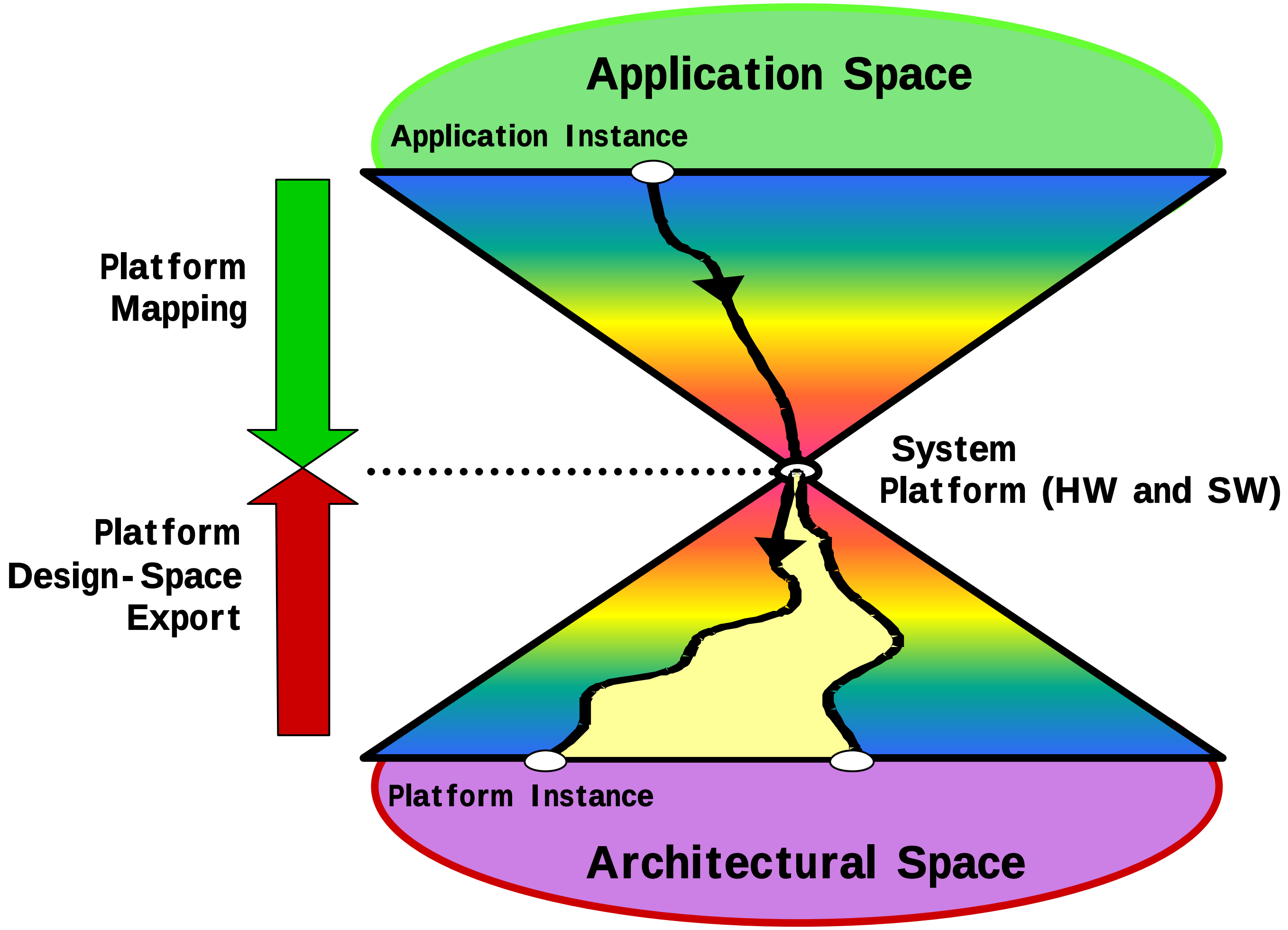




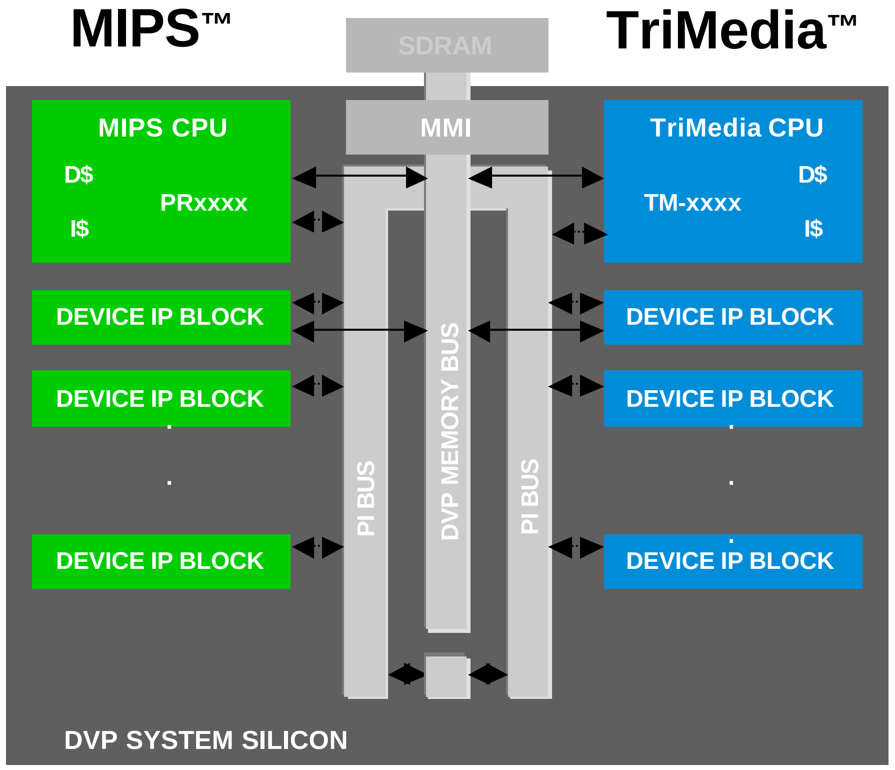


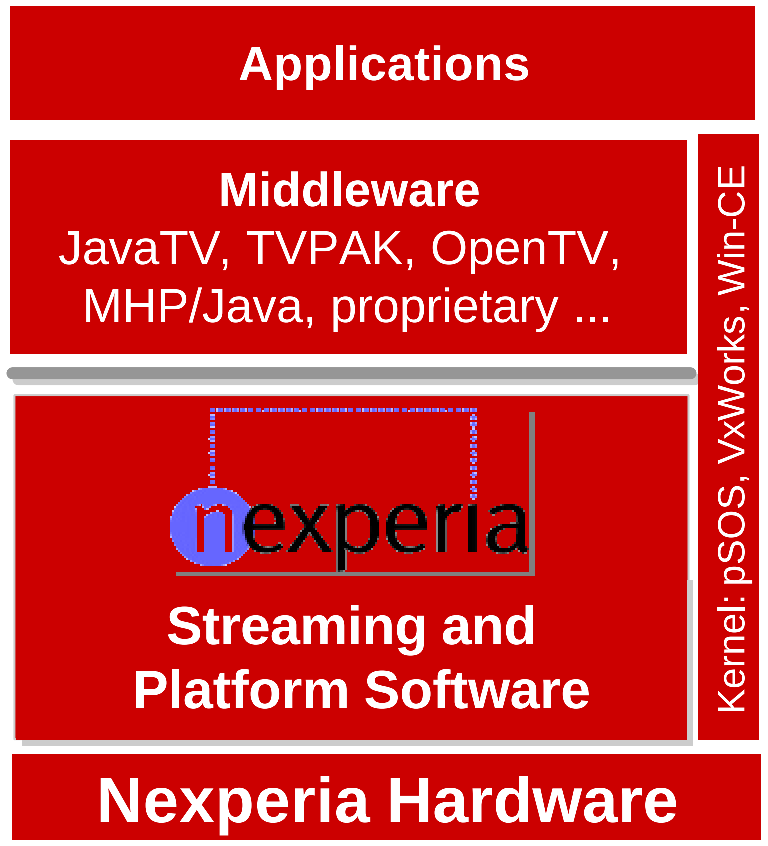
__Shift to__
- Re-use Strategy at all levels
- Higher Level of Abstractions
- Software !!!

Platform-based design



Platform Architectures: Hardware is not enough!



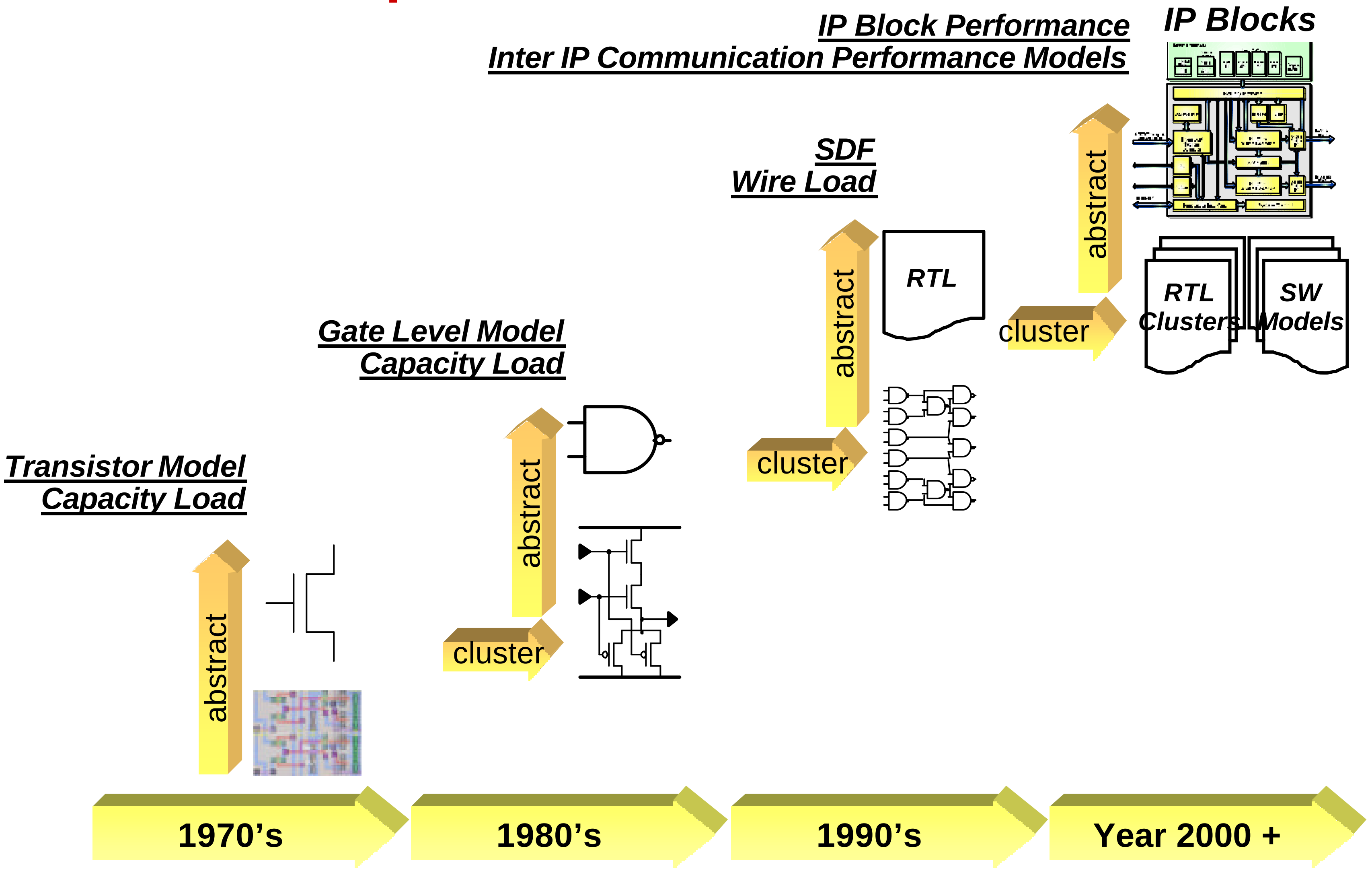


Hardware

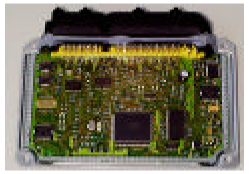
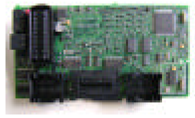
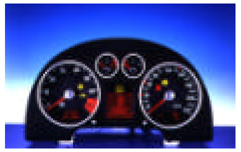
Software

Source: Philips

The Next Level of Abstraction in the Architecture Space



Embedded SW Challenges

	PWT UNIT 	BODY GATEWAY 	INSTRUMENT CLUSTER 	TELEMATIC UNIT
Memory			4 Kb	8 Mb
Lines Of Co			000	300.000
Productivi			es/Day	10 Lines/Day
Residual De Rate @ End C			ppm	1000 ppm
Changing R			ear	< 1 Year
Dev. Effort	40 Man-yr	12 Man-yr	30 Man-yr	200 Man-yr
Validation Time	5 Months	1 Month	2 Months	2 Months
Time To Market	24 Months	18 Months	12 Months	< 12 Months

It's embedded

⇒Need functional model for the rest of the system

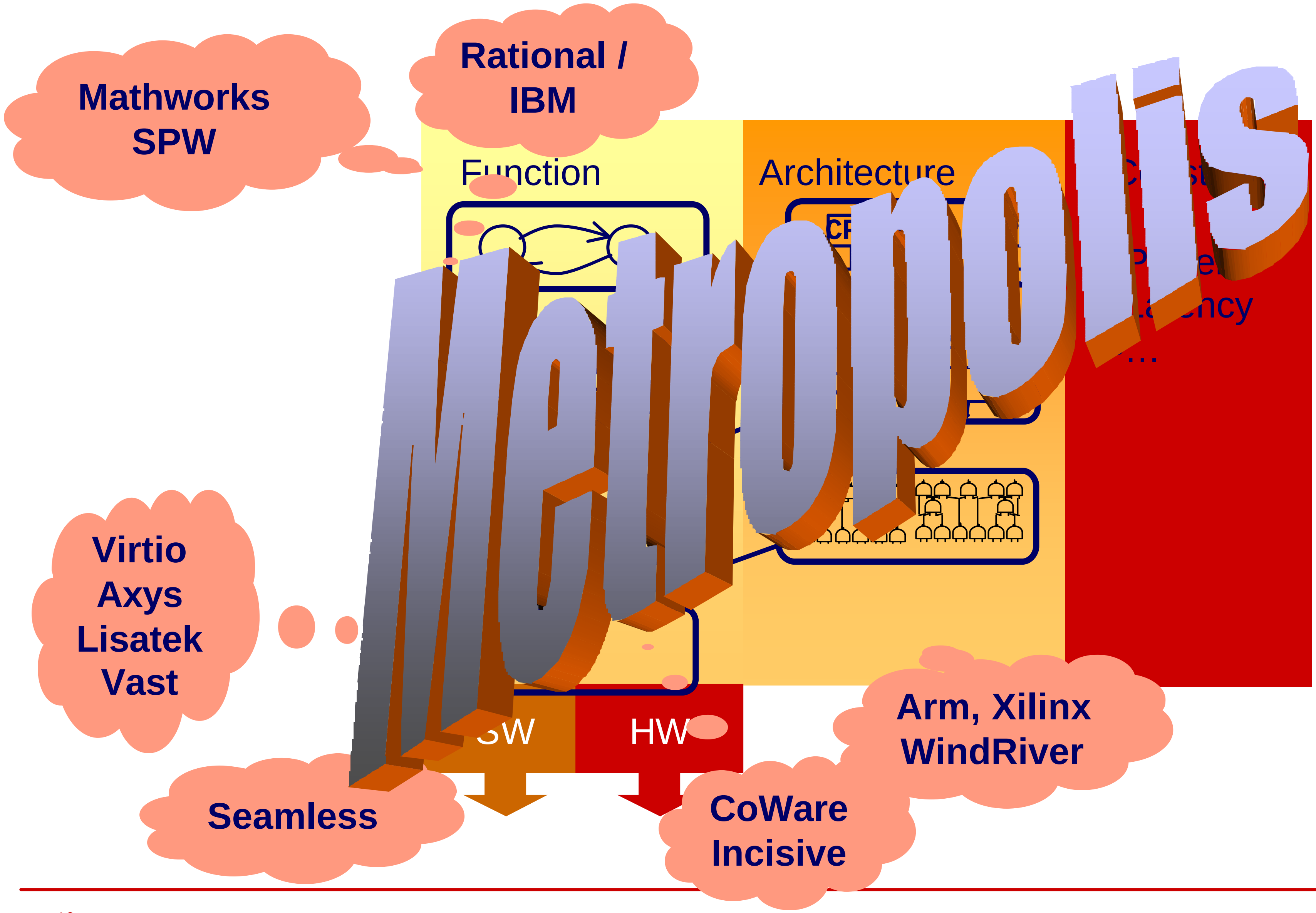
It's real-time

⇒Need performance model for the implementation platform

We need a System Design Platform

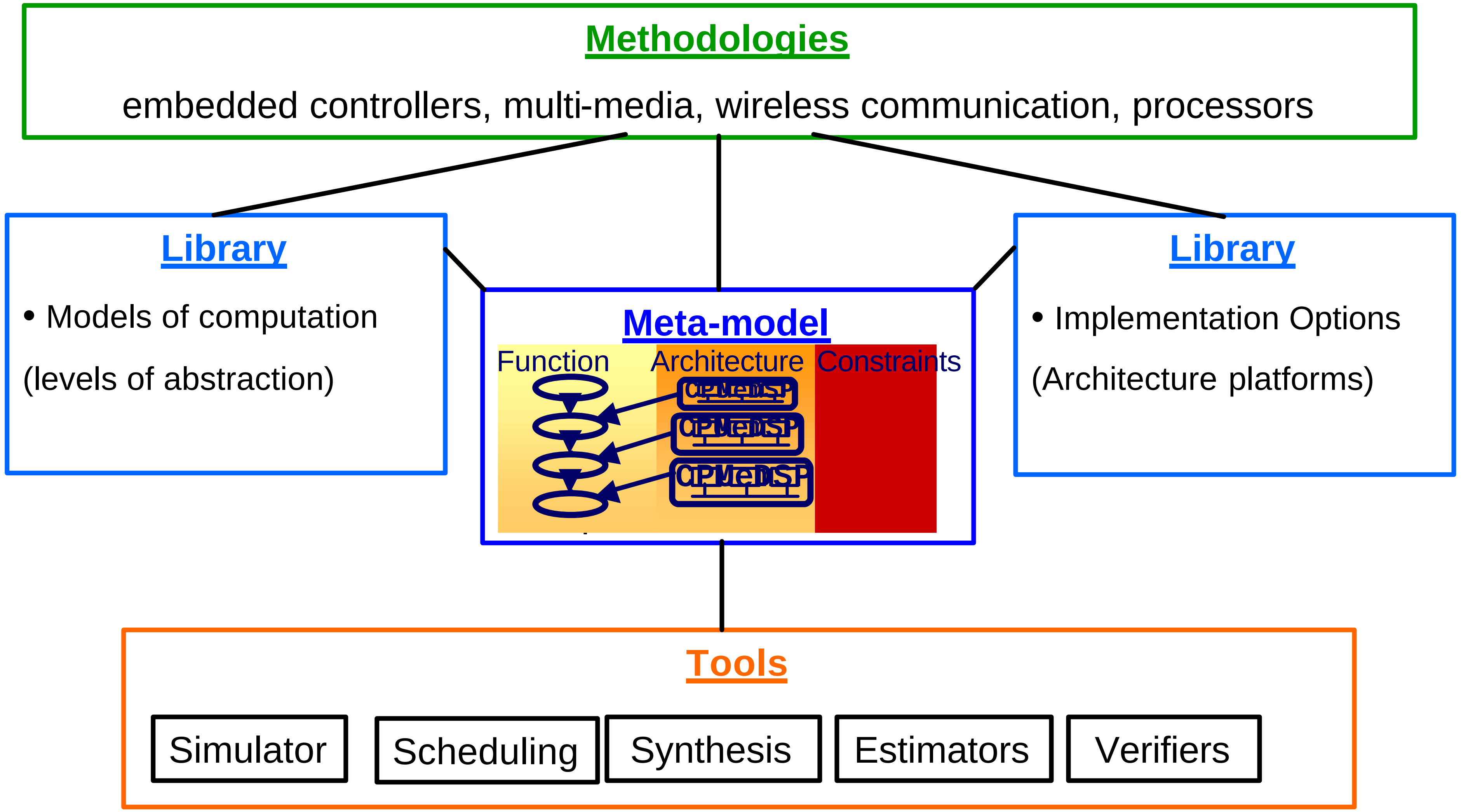
- **To deal with heterogeneity:**
  - Where we can deal with Hardware and Software
  - Where we can mix digital and analog
- **To handle the design chain**
  - Where we can assemble internal and external IPs
  - Where we can integrate tools
- **To explore the design space**
  - Where we can quickly evaluate alternatives
  - Where we can move seamlessly between levels of abstraction all the way to implementation

System Design Platform



OUTLINE

Metropolis Structure

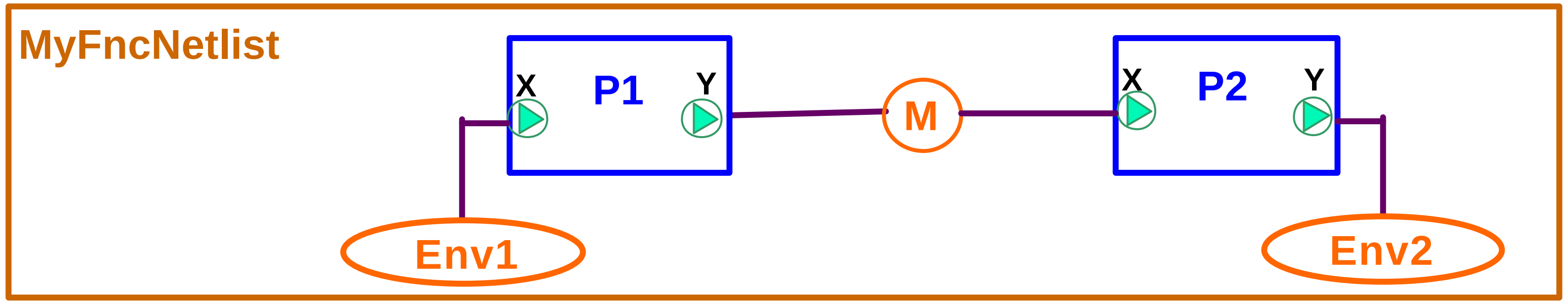


Metropolis meta-model

Concurrent specification with a formal execution semantics:

- **Computation** : $f : X \rightarrow Z$
  - **process** : generates a sequence of *events*
- **Communication** : state evaluation and manipulation
  - **medium** : defines *states* and *methods*
- **Coordination** : constraints over concurrent actions
  - **quantity** : annotated with events
  - **logic** : relates events wrt quantities, defines axioms on quantities
  - **q-manager** : algorithms to realize annotation subject to relations

Meta-model : function netlist



```
process P{
  port reader X;
  port writer Y;
  thread(){
   while(true){
    ...
    z = f(X.read());
    Y.write(z);
   }}}
```

```
interface reader extends Port{
    update int read();
    eval int n();
}
```

```
interface writer extends Port{
    update void write(int i);
    eval int space();
}
```

```
medium M implements reader, writer{
   int storage;
   int n, space;
   void write(int z){
       await(space>0; this.writer ; this.writer)
             n=1; space=0; storage=z;
    }
    word read(){ ... }
}
```

Meta-model: execution semantics

- Processes take *actions*.
  - Calls to port methods:

    port.f()

- 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
  - each process has a component in the network

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

OUTLINE

- Embedded systems challenge
- Metropolis project
- <u>Representing time</u>
- Representing timing requirements
- Relation to UML

Architecture modeling

An architecture is a service provider characterized by:

- what a service *can* do
- how much a service *costs*

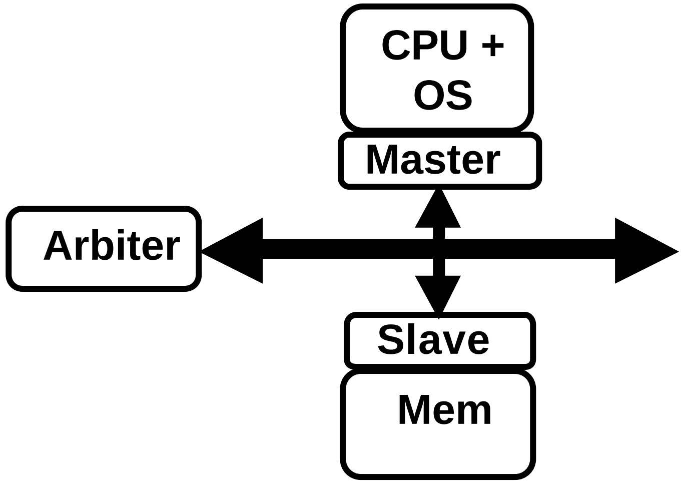
Services are:

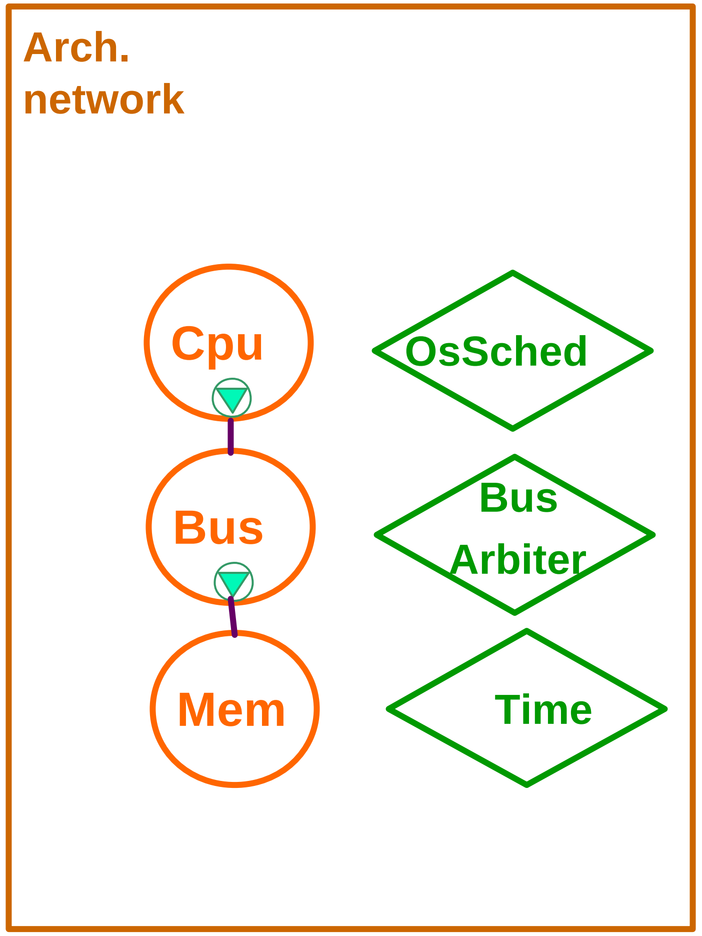
- declared by interfaces
- modeled by media implementing the interfaces
- media are parts of architecture network that may include other media and processes

Costs are modeled as annotations to behaviors

- various types of annotations are specified by quantities
- quantity managers are objects that decide annotations
- time is yet another quantity

Architecture model: example

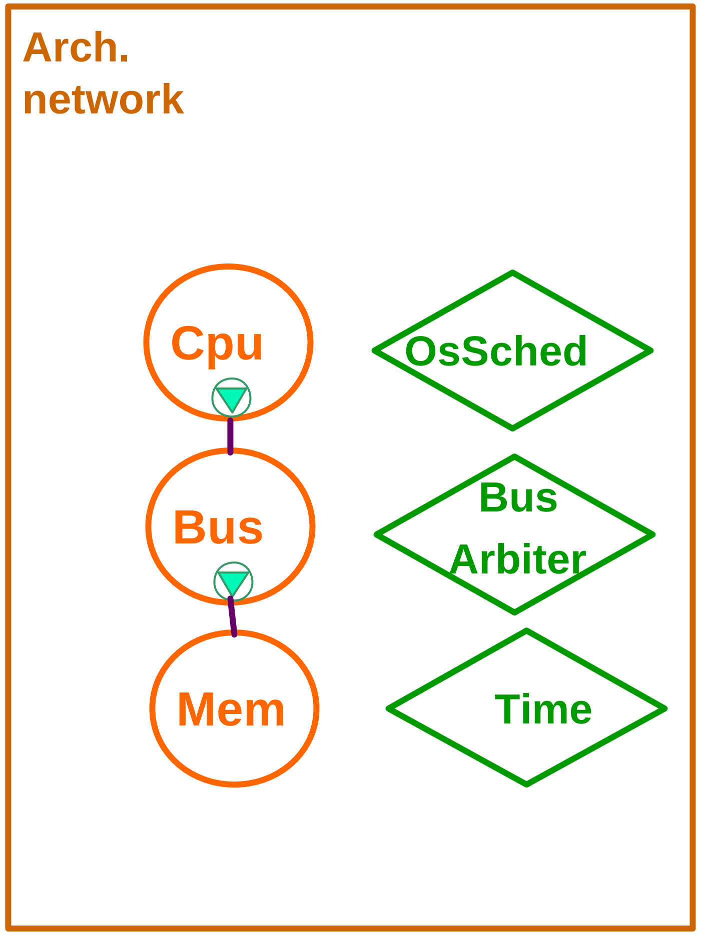




Architecture network specifies configurations of architecture components.

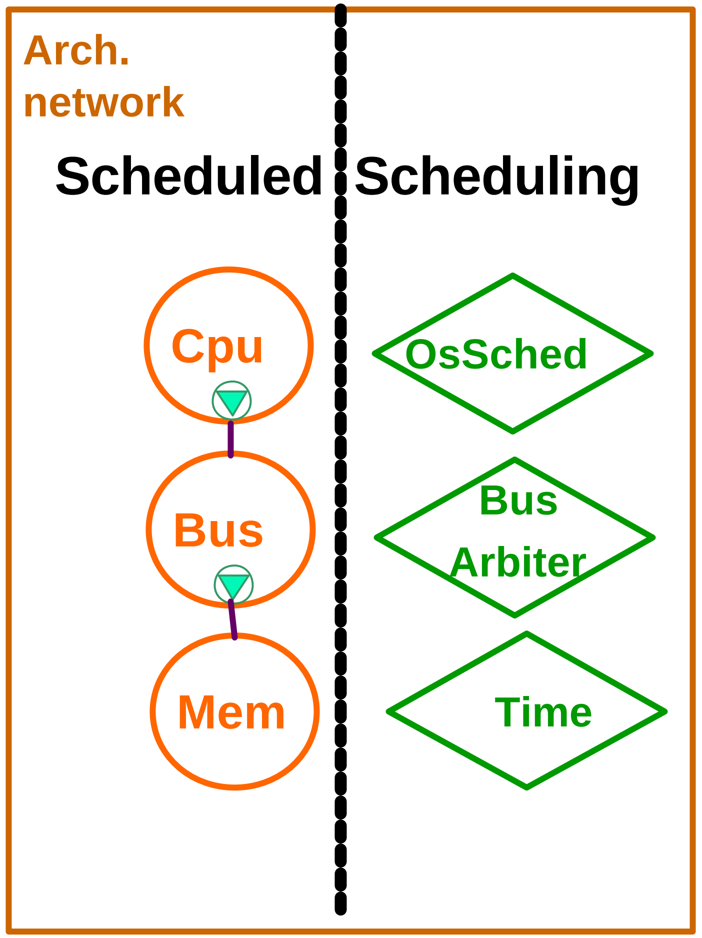
Quantities: annotation and coordination

- If two process attempt to use the CPU, one must be annotated as CPU owner, the other must be disabled
- If two events concurrently require different time stamps, the lower must be granted, and the higher must rejected
- Certain system behaviors are eliminated because they cannot be consistently annotated



Scheduled and scheduling networks

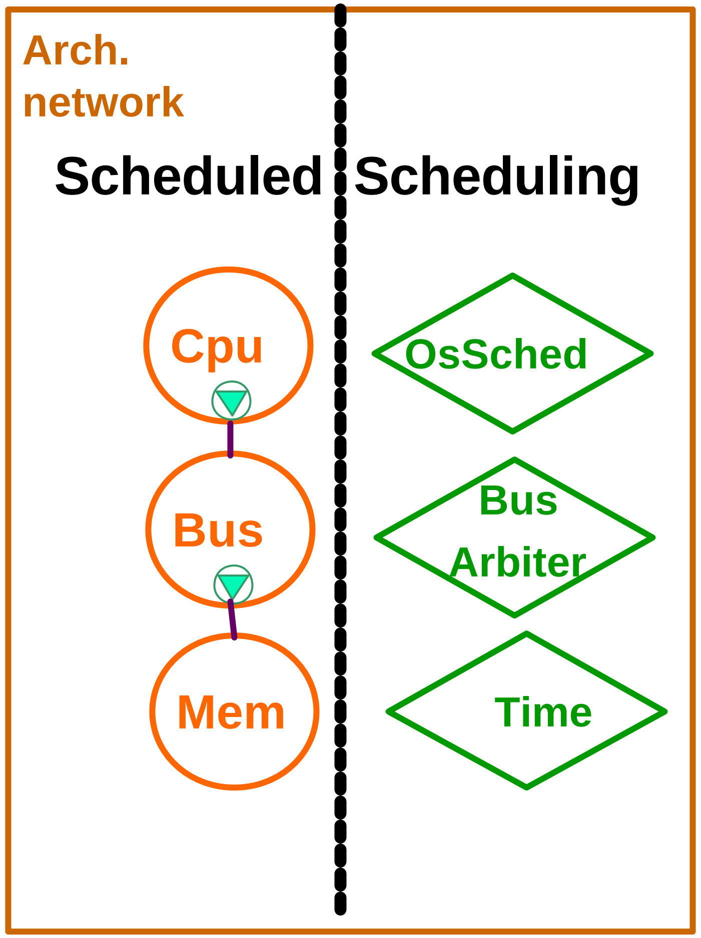
- Architecture components form scheduled network
- Quantity managers form scheduling network
- Scheduling network
  - annotates events in the scheduled network with quantities
  - disables events that cannot be annotated



Interactions between scheduled and scheduling networks

- Scheduled network may ***make requests*** to scheduling network
- When all the scheduled process make their requests, the execution moves into ***resolution*** phase:
  - quantity managers are executed until they agree on set of annotations
  - they may probe the state of the scheduled network
  - They may use services of separate meta-model network

Key for multiple levels of abstraction



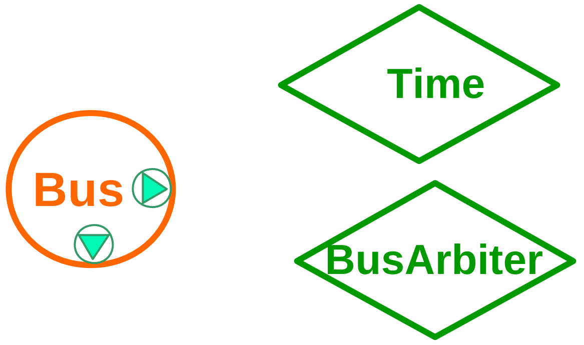
Example

```
interface BusMasterService extends Port {
  update void busRead(String dest, int size);
  update void busWrite(String dest, int size);
}
```

```
interface BusArbiterService extends Port {
  update void request(event e);
  update void resolve();
}
```

```
medium Bus implements BusMasterService …{
   port BusArbiterService Arb;
   port MemService Mem; …
   update void busRead(String dest, int size) {
       if(dest== … ) Mem.memRead(size);
    [[Arb.request(B(thisthread, this.busRead));
      Time.request(B(thisthread, this.memRead),
             BUSCLKCYCLE +
             GTime.A(B(thisthread, this.busRead)));
    ]]
   }
   …
```

```
scheduler BusArbiter extends Quantity
             implements BusArbiterService {
   update void request(event e){ … }
   update void resolve() { //schedule }
 }
```



OUTLINE

- Embedded systems challenge
- Metropolis project
- Representing time
- <u>Representing timing requirements</u>
- Relation to UML

Goals for constraint language

- solid math foundation
- natural to designers
- compatible with functional specification formalism
- expressive
- easy to simulate and verify formally

Logic Of Constraints syntax

Terms are

- **constants** of any sort
- variable *i*
- **e[t]**, **a(e[t])**, where **e** is event, **a** is annotation, **t** is term
- expressions with operators, e.g. y[i+2]-a(x[i])

LOC formulas are

- expressions with relations, e.g x[i]>y[i+2]
- Boolean combinations of formulas

Logic Of Constraints semantics

Interpreted over an annotated behavior:

$(v_{e,1,}\ a_{e,1},\ a'_{e,1} \ldots),\ (v_{e,2,}\ a_{e,2},\ a'_{e,2} \ldots)$

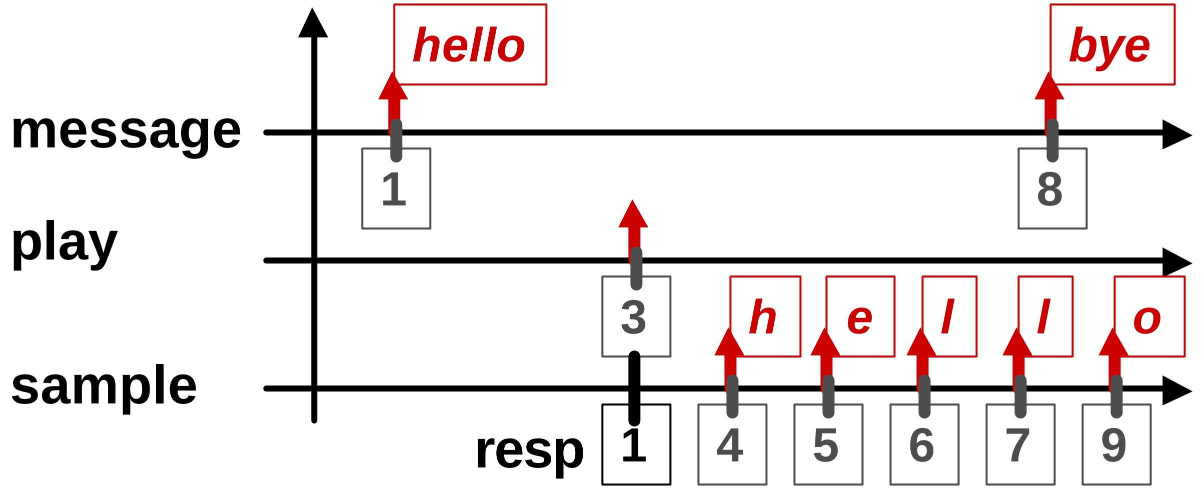
$(v_{e',1,}\ a_{e',1},\ a'_{e',1} \ldots),(v_{e',2,}\ a_{e',2},\ a'_{e',2} \ldots)\ \ldots$

…

- **variable *i* evaluates to any integer**
- **e[t] evaluates to $v_{e,\ eval(t)}$**
- **a(e[t]) evaluates $a_{e,\ eval(t)}$**
- **operators, relations, Boolean connectives as usual**

An annotated behavior satisfies the formula if it does not evaluate to FALSE for any value of *i*

Typical properties



rate

- time(message[i+1]) = time(message[i])+7

latency

- time(play[i])+2 > time(sample[i])
- time(play[i])+2 > time(sample[**resp[play[i]]**])

Verification

by simulation

- not hard to build a simulation monitor from a formula
- cannot prove satisfaction, only disprove it

by formal methods

- undecidable in general
- a subset can be reduced to Presburger arithmetic
- a smaller subset can be reduced to finite state model checking

OUTLINE

- Embedded systems challenge
- Metropolis project
- Representing time
- Representing timing requirements
- <u>Relation to UML</u>

UML Platform Profile

- A profile for specification of embedded system platforms
- Derived from design of wireless protocols
- Supports design specification …
  - Stereotypes like <<*Netlist*>>, <<*Process*>> , <<*Medium*>>, …
- … and methodology specification
  - Stereotype like <<*Implement*>> and <<*Refine*>>

UML Platform Profile

- Semantics is defined by the equivalent Metropolis meta-model network
- Essentially, a translation of the Metropolis meta-model to UML, but not complete
- Remaining challenges:
  - Add to the profile a mechanism to annotate behaviors including time
  - Be precise and complete, while respecting the spirit of UML of being simple and intuitive

Logic of Constraints

vs. UML profile for SPT

- SPT profile use tags to capture a fixed number of complex, parameterized formulas for which analysis has been developed
- LOC can capture many performance requirements, but complete analysis may not be available

vs. OCL

- OCL much better to specify static relations between objects
- LOC much better in reasoning about execution sequences

Thanks to …

- Prof. Sangiovanni-Vincentelli, UC Berkeley
- Yoshi Watanabe, Cadence
- Luciano Lavagno, Cadence
- Guang Yang, UC Berkeley
- Prof. Hsieh, UC Riverside
- Xi Chen, UC Riverside
- Grant Martin, Cadence
- many others
- and last but not least …

THANK YOU!