Metropolis

Design Environment for Heterogeneous Systems

Luciano Lavagno

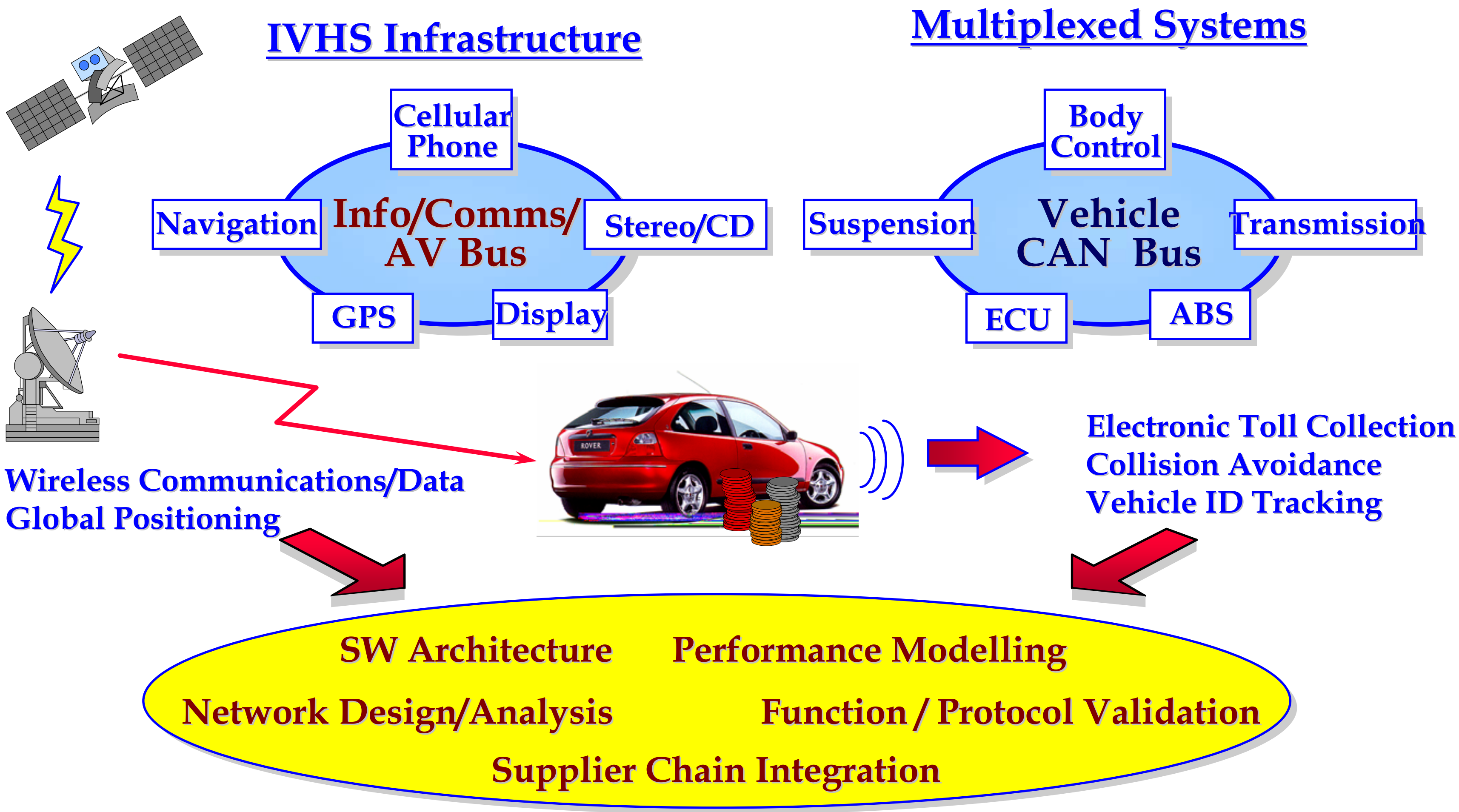
Cadence Berkeley Labs & Politecnico di Torino

Metropolis Project Team

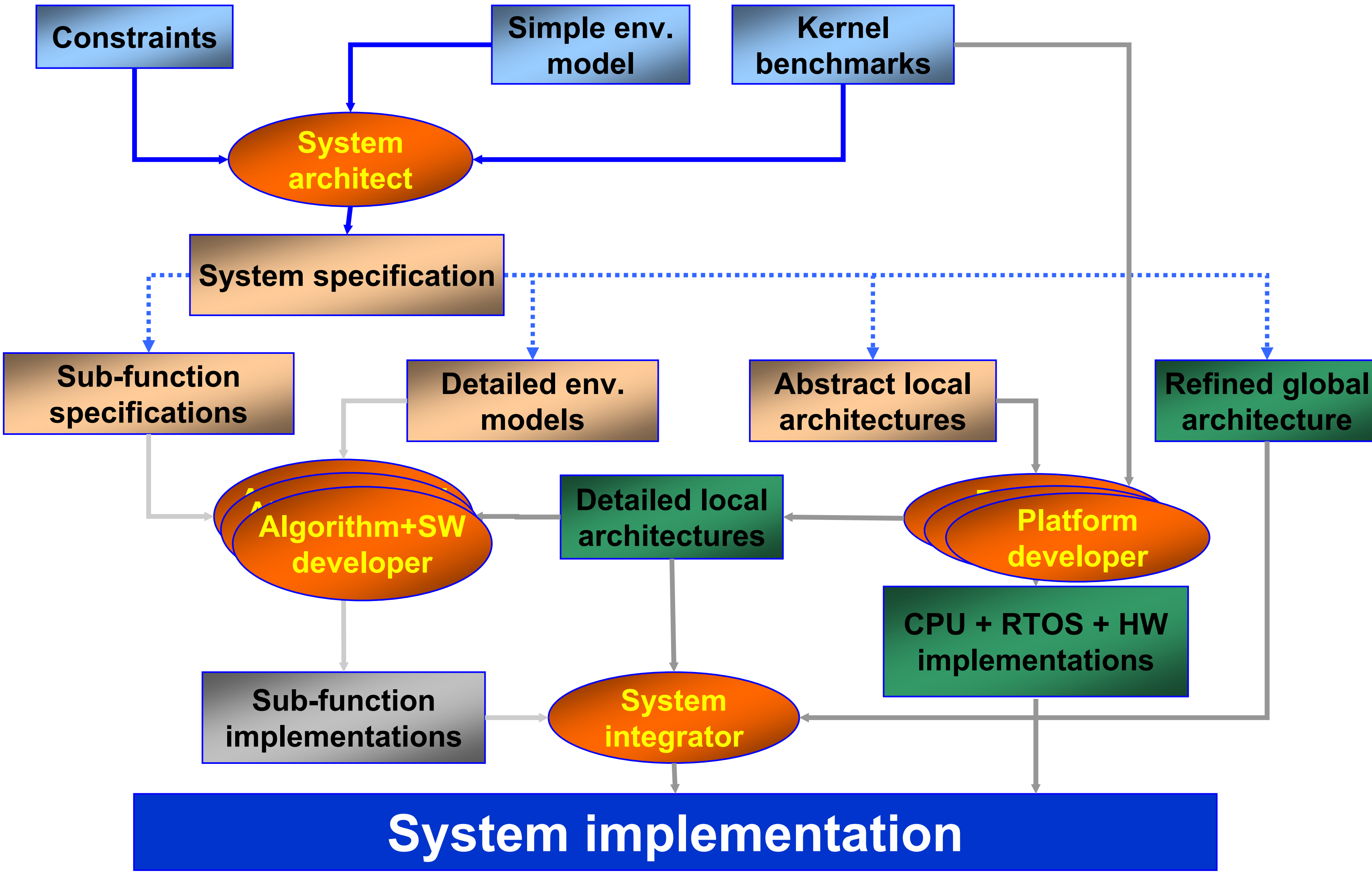
Outline

- **System-level design scenario**
- **Metropolis design flow**
- **Meta-model syntax**
  - **processes and media**
  - **constraints and schedulers**
- **Meta-model semantics**
- **Conclusions**

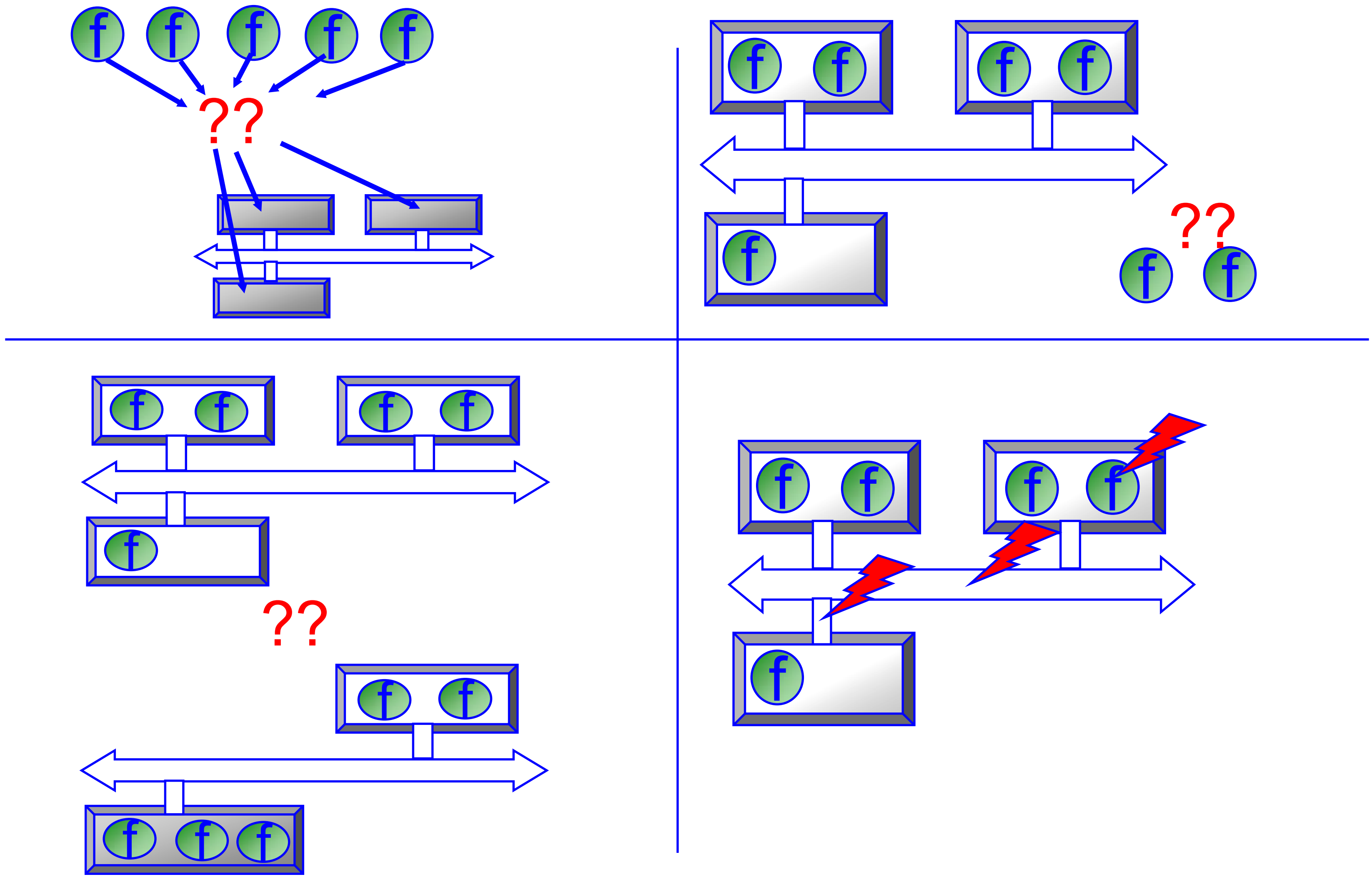
A Modern Car, an Electronic System



Design Roles and Interactions



Design Scenarios



Metropolis Project

etropolis

- Goal: develop a formal design environment
  - Design methodologies: abstraction levels, design problem formulations
  - Design automation tools:
    A modeling mechanism: heterogeneous semantics, concurrency
    Formal methods for automatic synthesis and verification

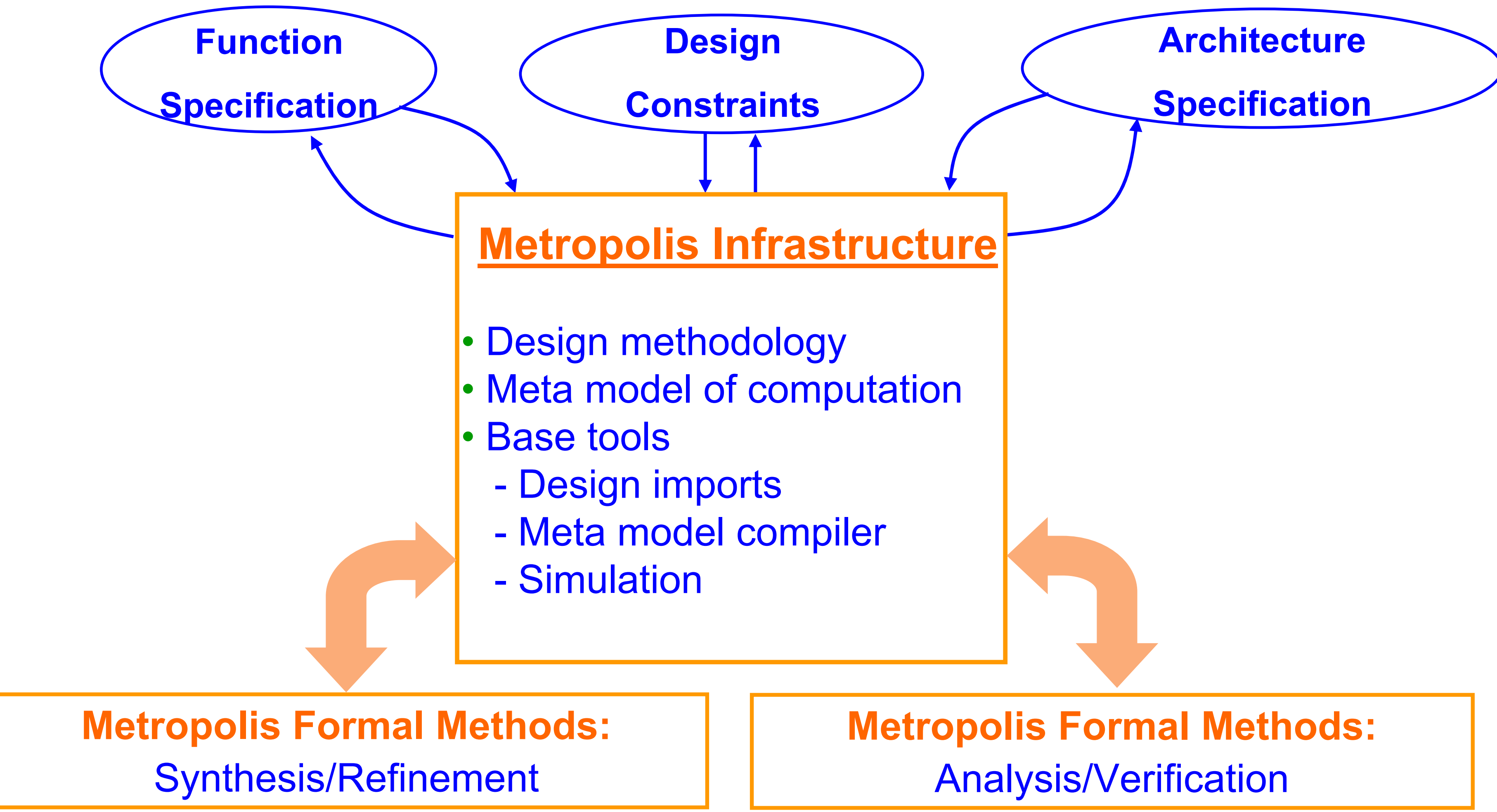
- Participants:
  - **Cadence Berkeley Labs** (USA): methodologies, modeling, formal methods
  - **UC Berkeley** (USA): methodologies, modeling, formal methods
  - **Politecnico di Torino** (Italy): modeling, formal methods
  - **Universitat Politecnica de Catalunya** (Spain): modeling, formal methods
  - **CMU** (USA): formal methods
  - **Philips** (Netherlands): methodologies (multi-media)
  - **Nokia** (USA, Finland): methodologies (wireless communication)
  - **BWRC** (USA): methodologies (wireless communication)
  - **BMW** (USA, Germany): methodologies (fault-tolerant automotive controls)
  - **Intel** (USA): methodologies (microprocessors)

Orthogonalization of concerns

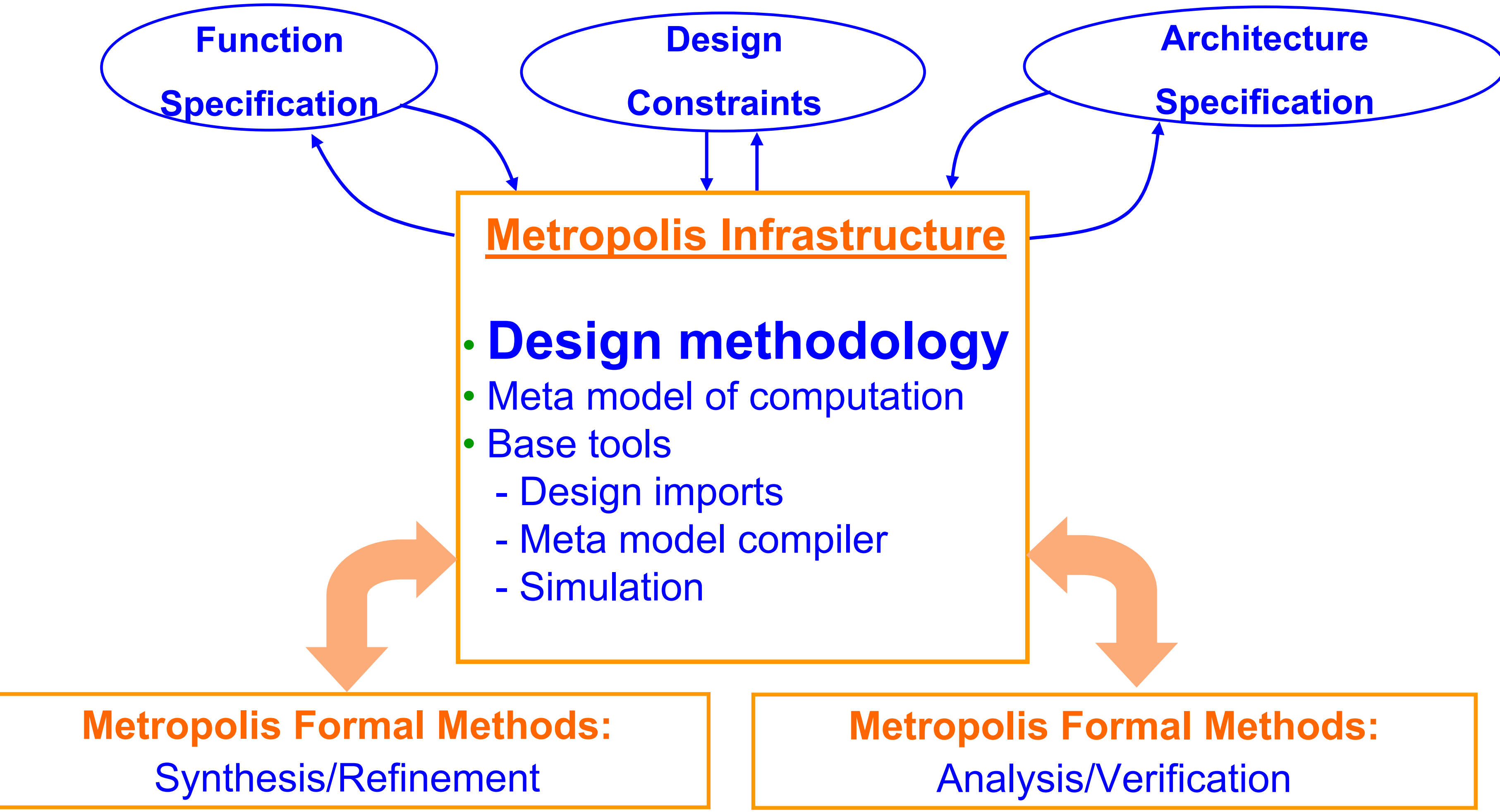
Separate:

- **functionality from architectural platform (function requires services offered by architecture)**
  - **increased re-use**
  - **use same level of abstraction for HW and SW**
  - **design space exploration**
  - **drive synthesis algorithms (compiler, scheduler, …)**
  - **separates behavior from performance (time, power, …)**
  - **performance derives from mapping**
- **computation from communication**
  - **computation (functionality) is scheduled and compiled**
  - **communication (interfacing) is refined via patterns based on mapping**

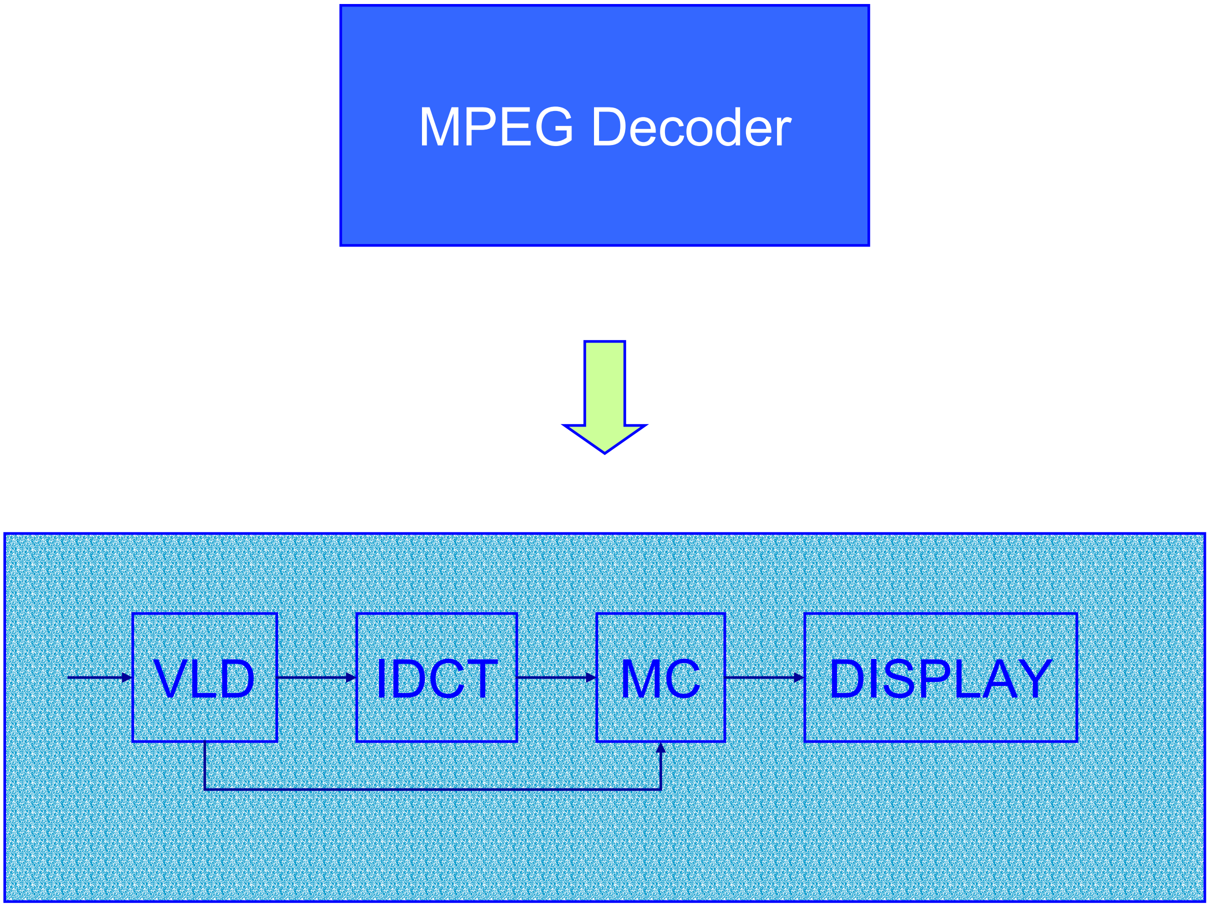
Metropolis Framework



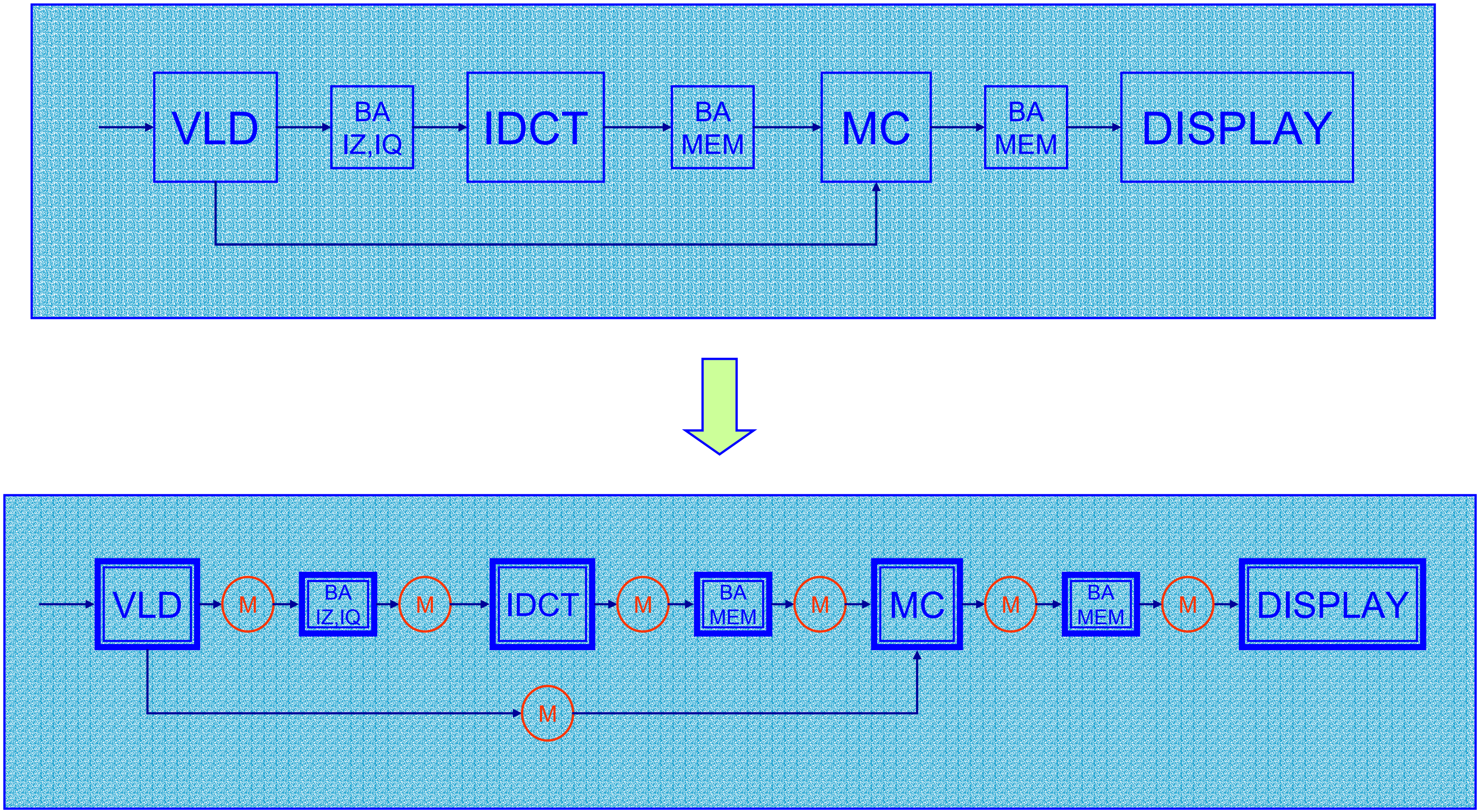
Metropolis Framework: methodology



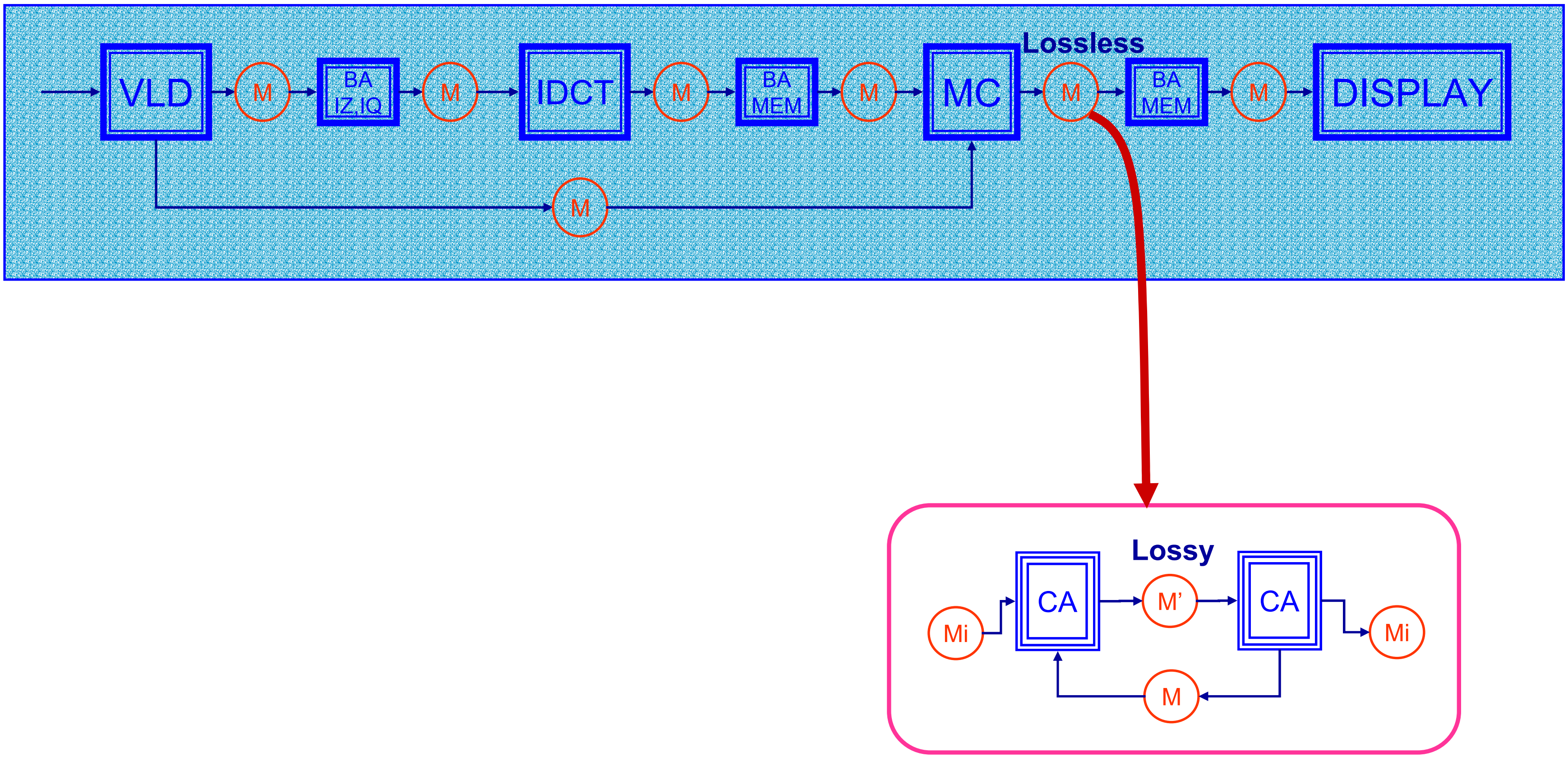
Functional Decomposition



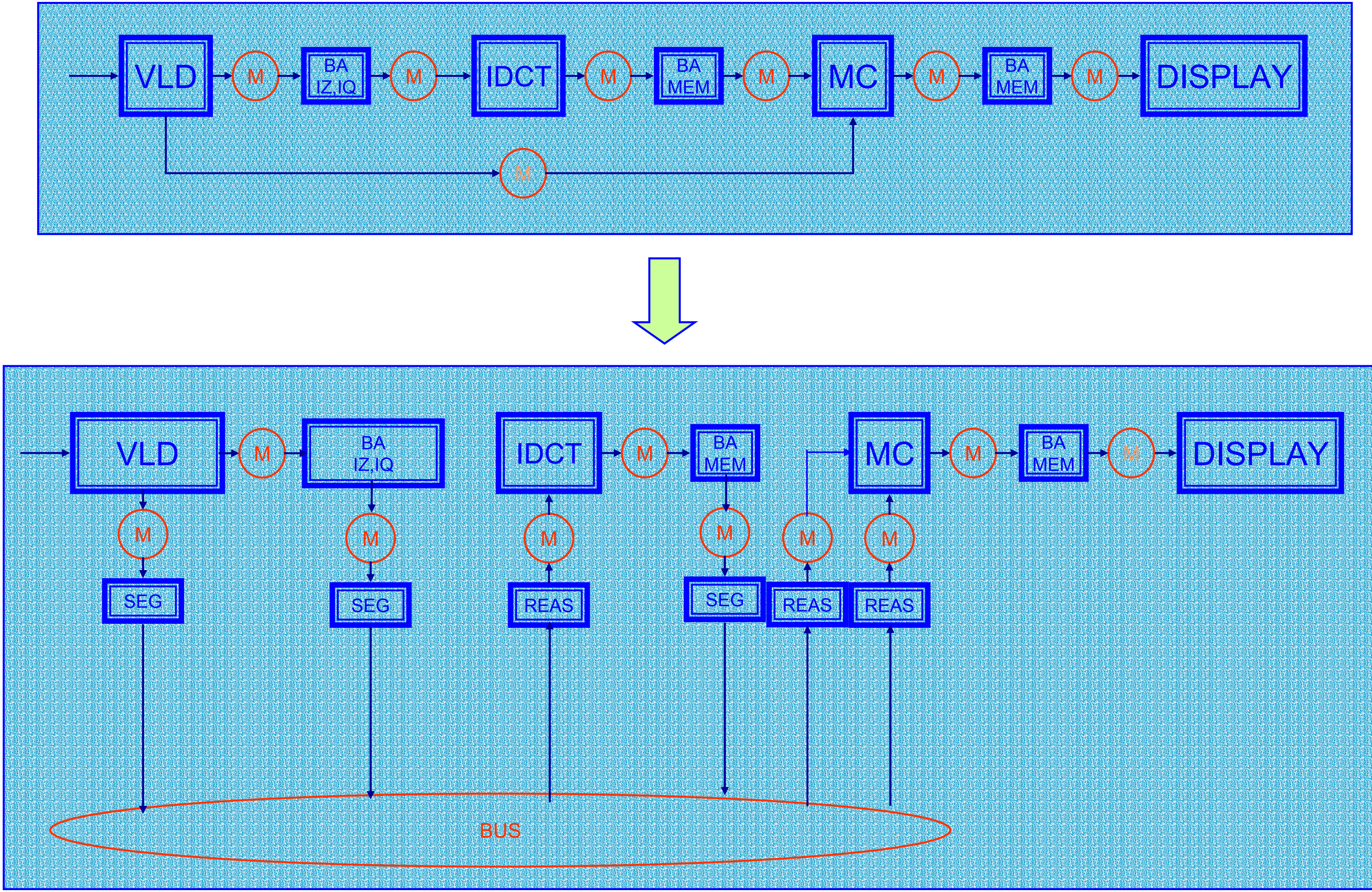
Communication



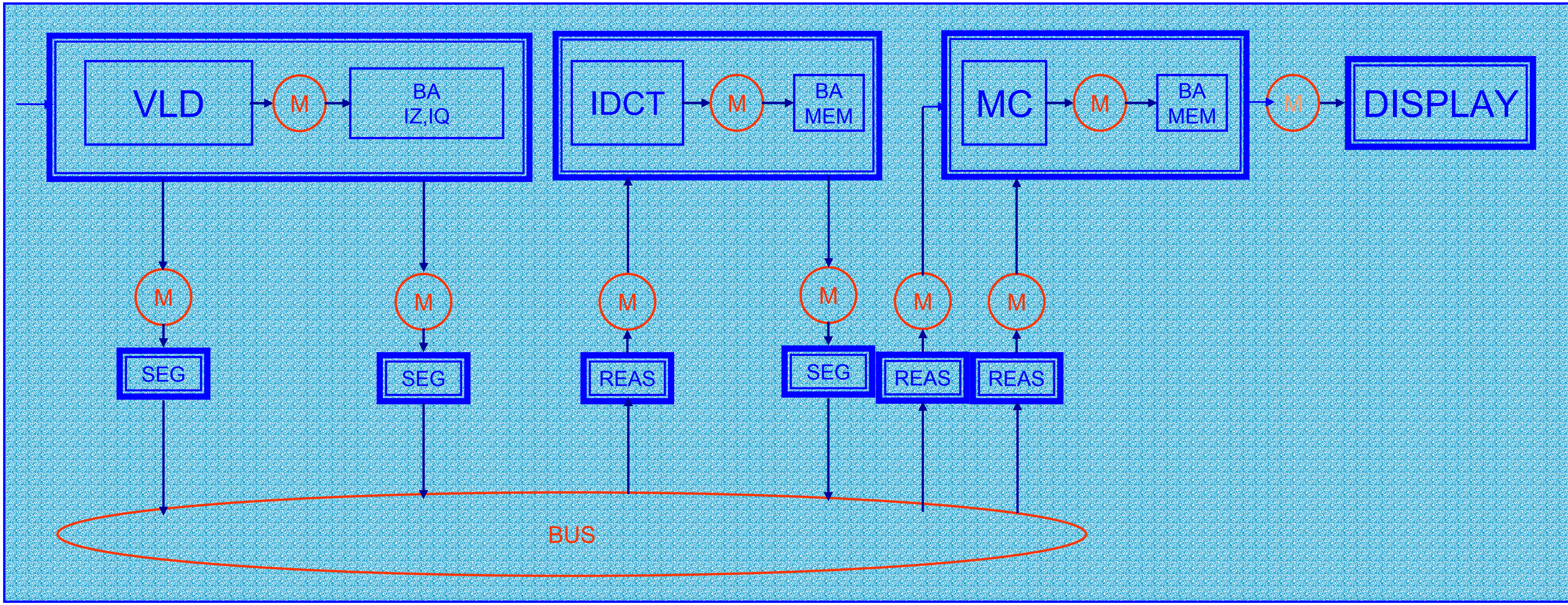
Refinement


VLD
M
BA
IZ,IQ
M
IDCT
M
BA
MEM
M
MC
Lossless
M
BA
MEM
M
DISPLAY
M
Lossy
Mi
CA
M'
CA
Mi
M

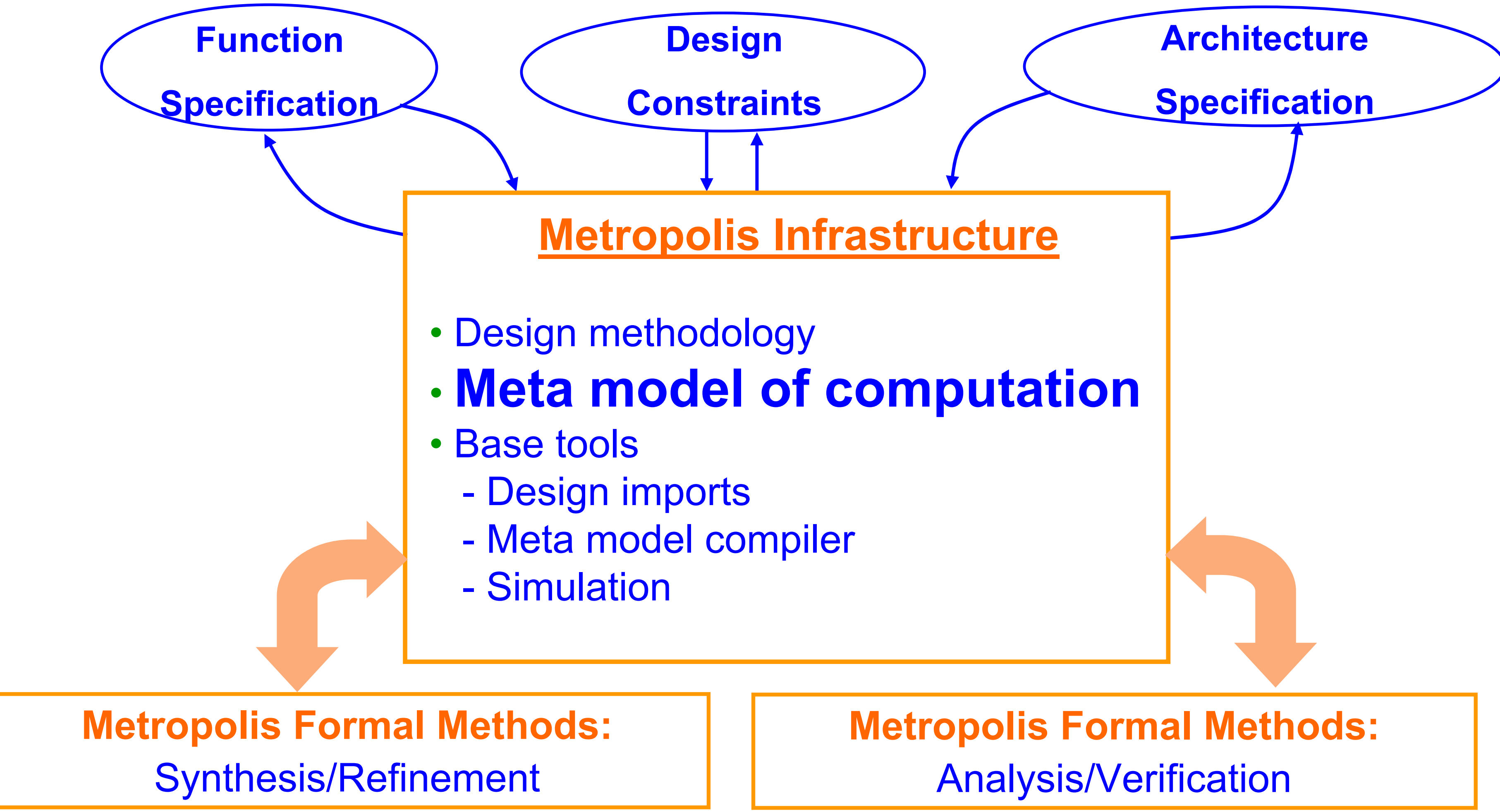
Refinement


VLD
M
BA
IZ,IQ
M
IDCT
M
BA
MEM
M
MC
M
BA
MEM
M
DISPLAY
M
VLD
M
BA
IZ,IQ
IDCT
M
BA
MEM
MC
M
BA
MEM
M
DISPLAY
M
M
M
M
M
M
SEG
SEG
REAS
SEG
REAS
REAS
BUS

Optimization



Metropolis Framework: meta-model



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**

Question: What is a good set of building blocks?

Metropolis Meta Model

The behavior of a concurrent system:

computation

- f: X ⟶ Z
- firing rule

processes

communication

- state
- methods to
  - store data
  - retrieve data

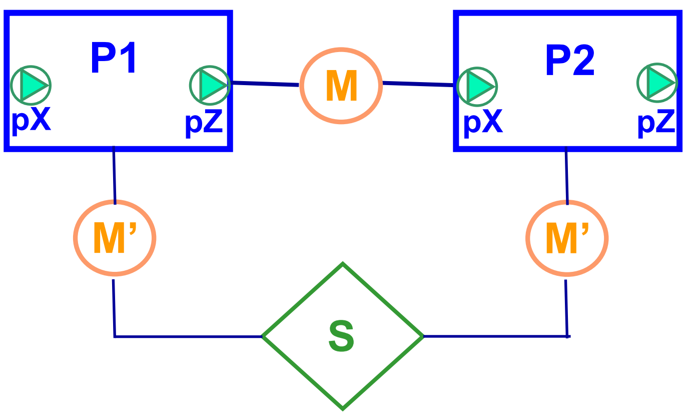
media

coordination

- constraints on concurrent actions
- action annotation with quantity requests (time, energy, memory)
- algorithms to enforce the constraints

constraints and quantity managers

```
process P1{
port pX, pZ;
thread(){
  // condition to read X
  // an algorithm for f(X)
  // condition to write Z
}
}
```



P1.pZ.write() ◆ P2.pX.read()

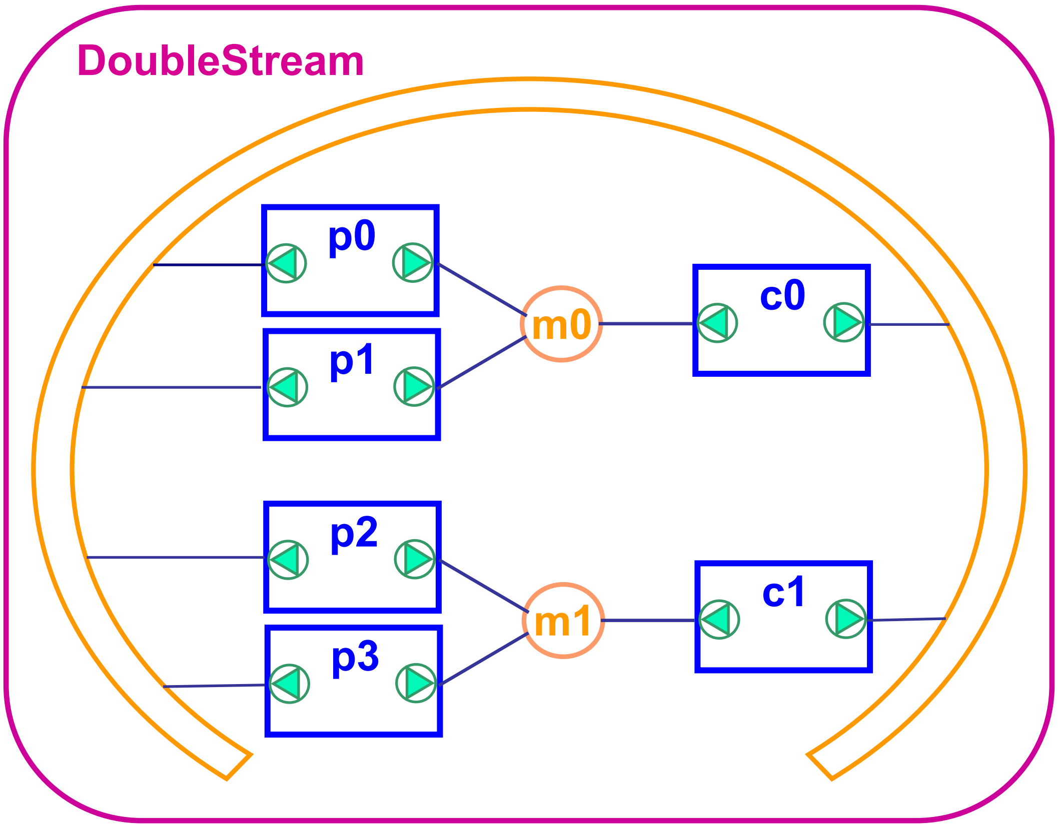
```
medium M{
   int[] storage;
   int space;

   void write(int[] z){ ... }

   int[] read(){ ... }

}
```

Netlist

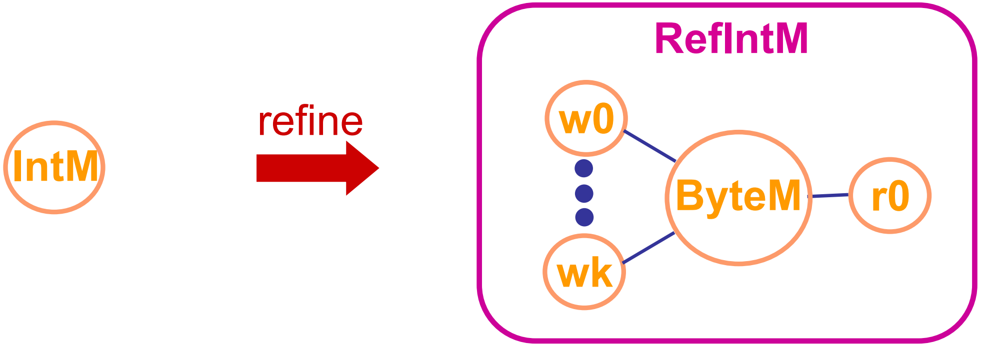
Define

- processes, media, schedulers, netlists
- connections among the objects
- constraints

used also for specifying refinements



Communication and computation refinement



Define a refinement "pattern":

1. Define objects that constitute the refinement.

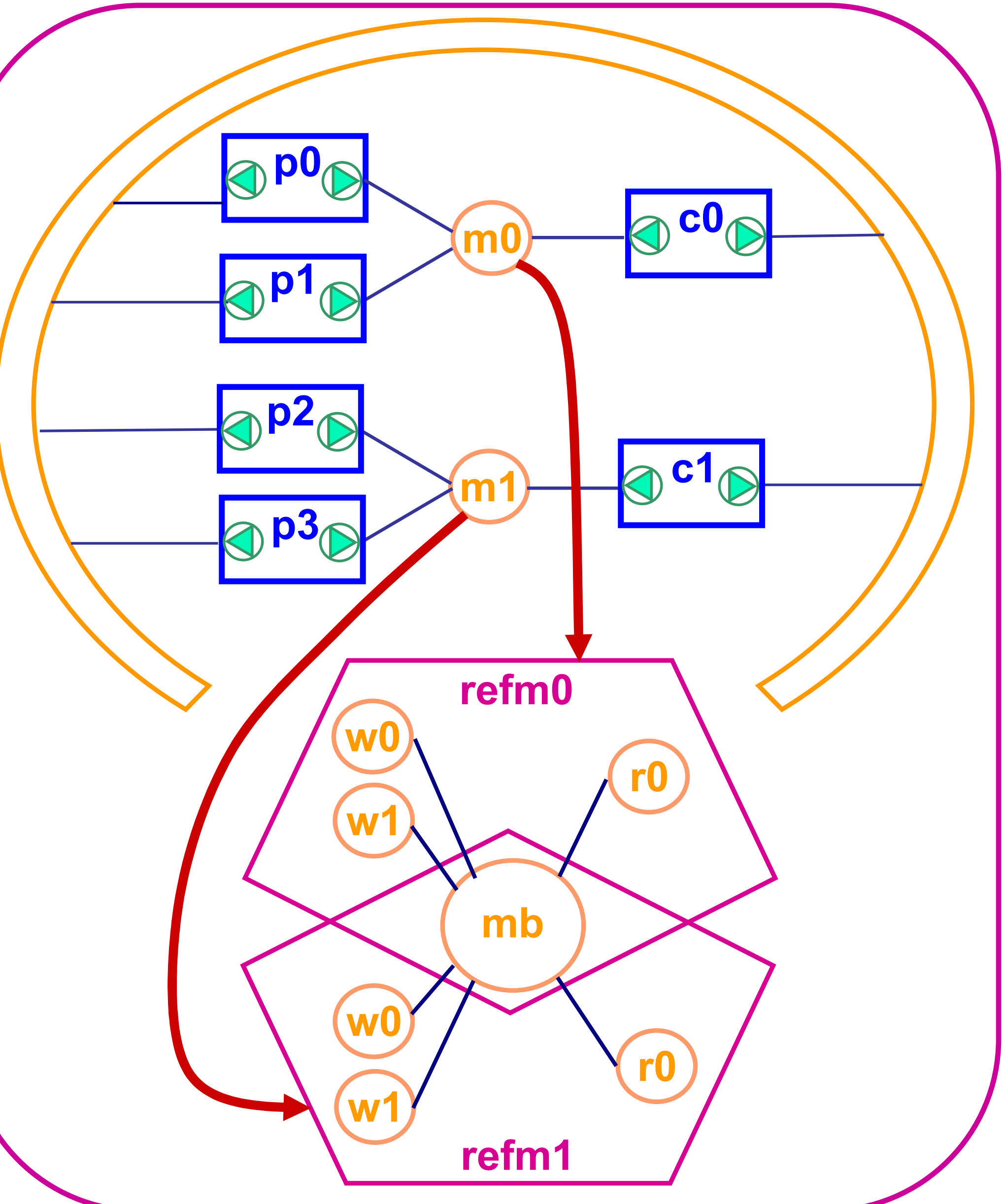
2. Define connections among the refinement objects.

3. Specify connections with objects outside the refinement netlist:

Some objects in the refinement may be internally created; others may be given externall

⇨ **write a constructor of the refinement netlist for each refinement scenario.**

Netlist after Refinement



// create mb, and then refine m0 and m

ByteM mb = new ByteM();

RefIntM refm0 = new RefIntM(m0, mb);

RefIntM refm1 = new RefIntM(m1, mb);

But, we need coordination:

- if p0 has written to mb, c0 must read
- if p2 has written to mb, c1 must reac
- ...

Constraints

Two mechanisms are supported to specify constraints:

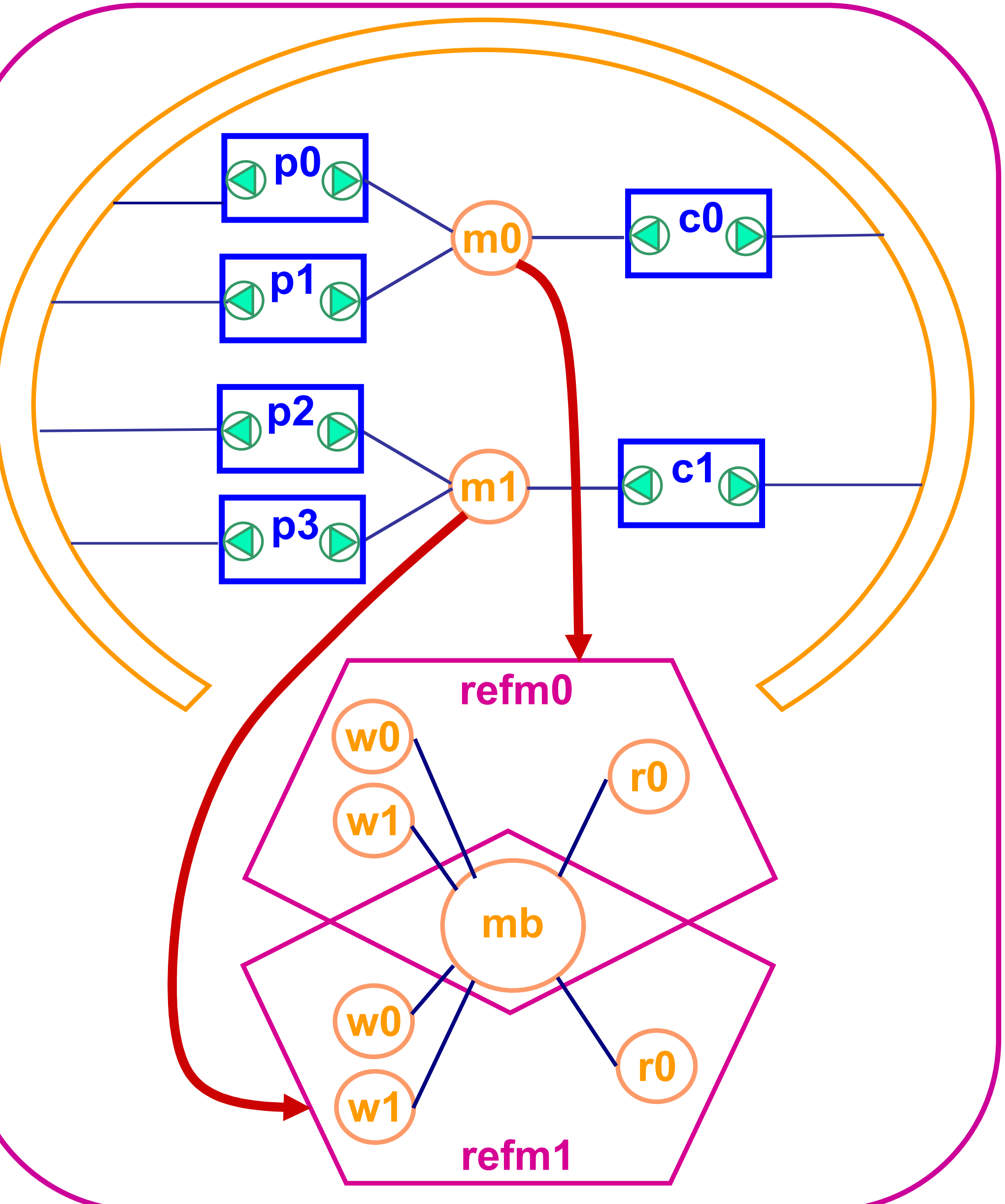
1. Propositions over temporal orders of states
   - execution is a sequence of states
   - specify constraints using linear temporal logic
   - good for **functional** constraints, e.g.

     "if process P starts to execute a statement s1, no other process can start the statement until P reaches a statement s2."

2. Propositions over instances of transitions between states
   - particular transitions in the current execution: called "*actions*"
   - annotate actions with quantity, such as time, power.
   - specify constraints over actions with respect to the quantities
   - good for **performance** constraints, e.g.

     "any successive actions of starting a statement s1 by process P must take place with at most 10ms interval."

Netlist after Refinement



// create mb, and then refine m0 and m

ByteM mb = new ByteM();

RefIntM refm0 = new RefIntM(m0, mb);

RefIntM refm1 = new RefIntM(m1, mb);

But, we need coordination:

- if p0 has written to mb, c0 must read
- if p2 has written to mb, c1 must reac
- ...

Can be specified using the linear temporal logic.

Constraints

1. Propositions over temporal orders of states

State variables

• <u>process</u>:

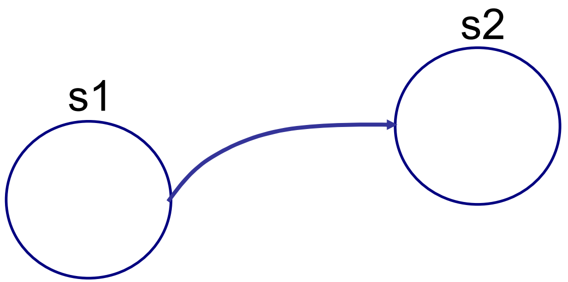
- instances of local variables of called functions

- program counter:

{**beg(s)**, **end(s)**} for each statement **s**

• <u>medium</u>

field instances



• execution (s1, s2, ...) : a linear (possibly infinite) order of states such that

- it starts from the initial state,

- each adjacent pair is a transition

Propositions on Temporal Order of States

- Linear Temporal Logic (LTL):

  propositions over state variables

  - temporal operators: **X, U, F, G**

  - logical operators: &&, !, ||, **->**, **<->**

  - ltl() method to specify constraints

- Built-in constructs on the LTL:

  excl, mutex, simul

constraints{...} can appear anywhere in the meta-model programs.

```
medium M{
  word storage;
  int n, space;
  void write(word z){
  wr: {
    await(space>0)[this]
  l1:       n=1; space=0; storage=z;
    }
  }
  word read(){
  rd: {
    await(n>0)[this]
  l2:      n=0; space=1; return storage;
    }
  }
  constraints{
      process p, q;
      ltl(G(pc(p)==beg(wr) ->
          F(pc(q)==end(rd))));
  }
}
```

Constraints

2. Propositions over instances of transitions between states

- Action: instantiation of a transition in an execution (s1, s2, ...)

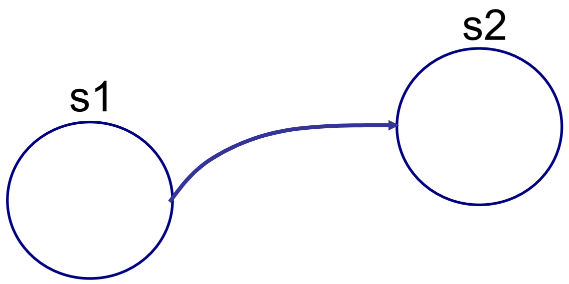
  action $a = (p, s_c, s_n, o)$

  p : process object

  s_c : current value of the program counter of p

  s_n : next value of the program counter of p

  o : occurrences of the transition $s_c \rightarrow s_n$ by p in the execution



- Quantity: annotated with the set A of actions of the current execution
  - The domain D of the quantity, e.g. *real* for the global time
  - The operations and relations on D, e.g. subtraction, <, =
  - The relation between D and A, e.g. *gt(a)* denotes the global time of the action *a*
  - Constraints on the quantity and actions, e.g.

    for all actions a1, a2, if a2 follows a1 in the execution, $gt(a1) < gt(a2)$

Constraints using Actions

- ```
  public final class Action {
     process p;
     pcval sc, sn;
     int o;
  }
  ```

- ```
  public class Gtime extends Quantity {
     static double t;
     double sub(double t2, double t1){...}
     boolean equal(double t1, double t2){ ... }
     boolean less(double t1, double t2){ ... }
     double gtime(Action a){ ... }
     constraints{ ... }
  }
  ```

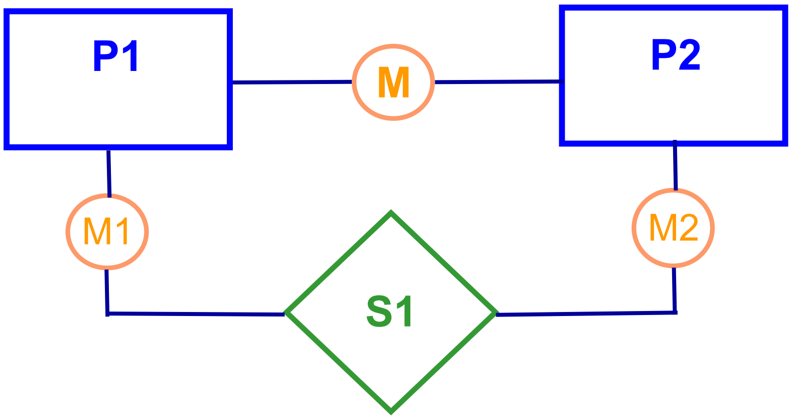
```
process P1{
  port reader pX, pY;
  port writer pZ;
  thread(){
   while(true){
   ...
     await(pX.n()>0 && pY.n()>0)
        [pX.reader,pY.reader]
l1:          z = f(pX.read(), pY.read());
l2:  pZ.write(z);
     ...
   }
 }
}
```

```
constraints{
    Action a1, a2;
    Gtime gt;
    lfo(a1.p()==a2.p() && a1.sn()==beg(l1)
        && a2.sn==end(l2) && a1.o()== a2.o()
        -> gt.gtime(a2) - gt.gtime(a1) < 5);

    lfo(a1.p()==a2.p() && a1.sn()==beg(l1)
        && a2.sn==beg(l1)
        && a2.o()==a1.o()+1
        -> gt.gtime(a2) - gt.gtime(a1) < 10);
```

```
constraints{
    maxLatency(l1, l2, 5);
    maxRate(l1, 10);
}
```

Schedulers

- Scheduler specifies an algorithm for *some* constraints.




```
scheduler S1{
  port SMsched port0, port1;
  ...
  void doScheduling(void){
      // priority scheduling
  }
}
```

- The algorithms are used during simulation.
- Typically, later in the design phase, thread() is added to a scheduler,
  - to specify protocols to communicate with the controlled processes,
  - to call doScheduling() as a sub-routine.

  At that point, the scheduler becomes a process.
- Schedulers may be hierarchical.

Execution semantics

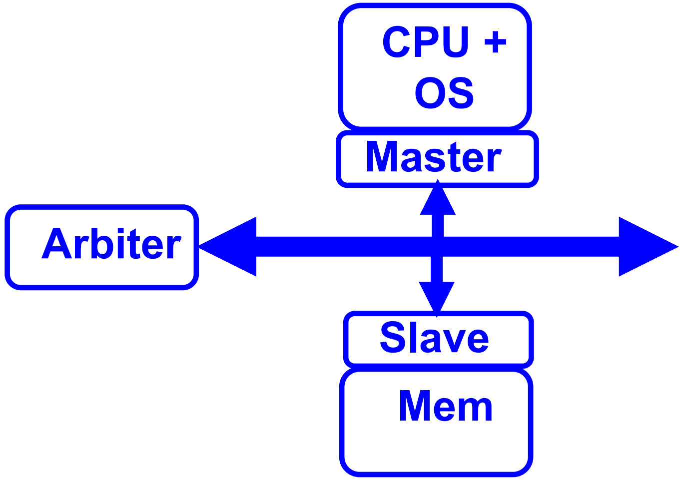
- **Normal approach (VHDL, SystemC, …):**
  1. **define simulation algorithm**
  2. **define suitable language and semantics**
  3. **try to synthesize, verify, refine**
  4. **oops… semantics gap!**
- **Our approach:**
  1. **define semantics for synthesis, refinement**
  2. **figure out how to simulate it**
  3. **oops… inefficient simulation?**

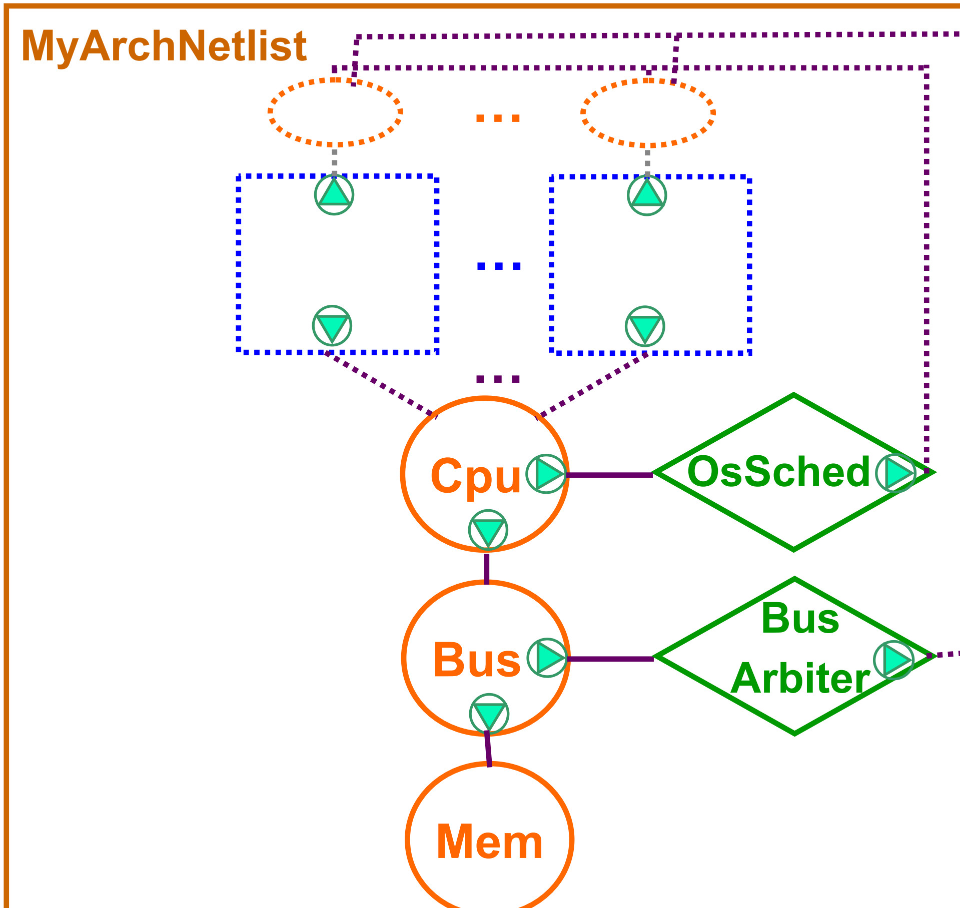
Meta-model: architecture netlist

Architecture netlist specifies configurations of architecture components.

Each netlist constructor

- instantiates architectural components,
- connects them,
- takes as input *mapping processes*.





Meta-model: mapping processes

Function process

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

Mapping process

```
process MapP{
  port CpuService Cpu;
  void readCpu(){
     Cpu.exec();   Cpu.cpuRead();
  }
  void mapf(){ …}
   …
  thread(){
     while(true){
      await {
        (true; ; ;) readCpu();
        (true; ; ;) mapf();
        (true; ; ;) readWrite();
} }}}
```

B(P, X.read) <=> B(MapP, readCpu);   E(P, X.read) <=> E(MapP, readCpu

B(P, f) <=> B(MapP, mapf);   E(P, f) <=> E(MapP, mapf);

…

Meta-model: mapping netlist

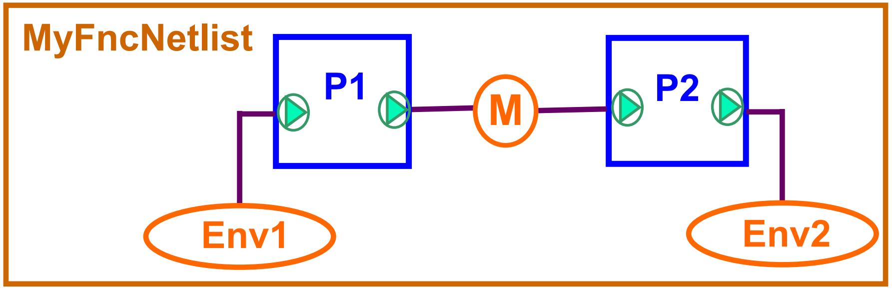
MyMapNetlist

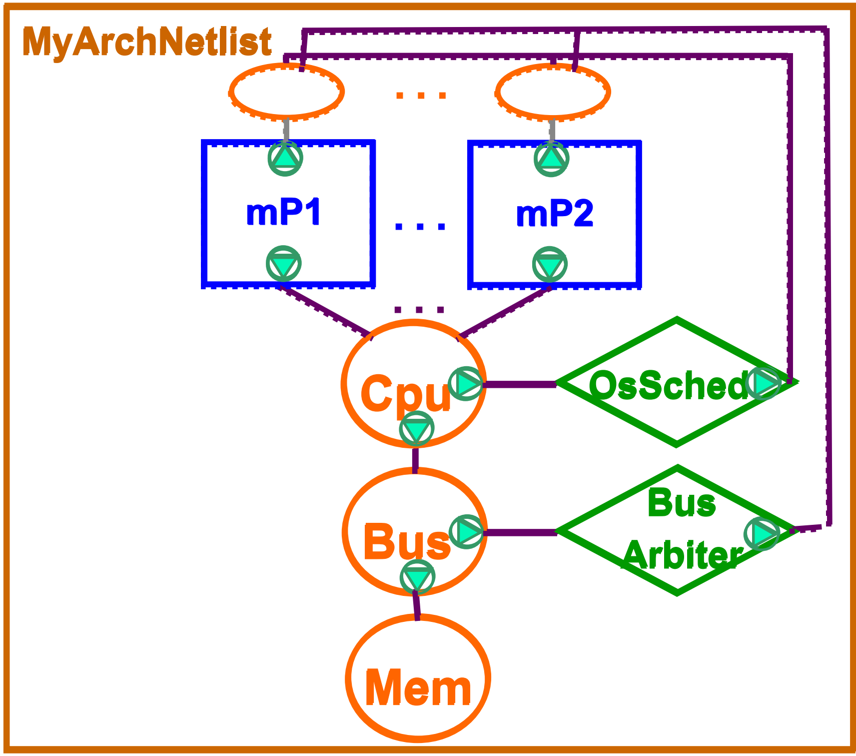
B(P1, M.write) <=> B(mP1, mP1.writeCpu); E(P1, M.write) <=> E(mP1, mP1.writeCpu);

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);

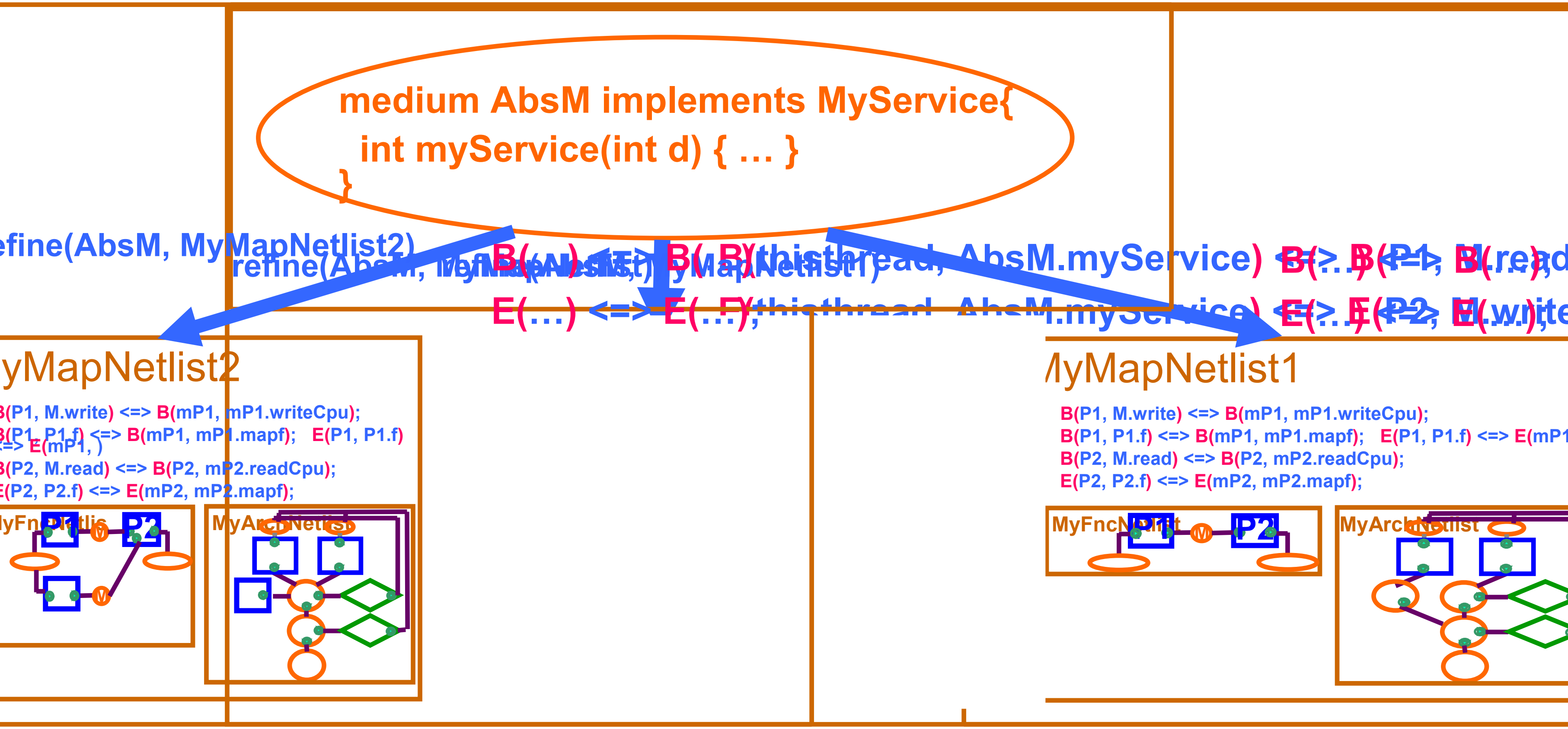
B(P2, P2.f) <=> B(mP2, mP2.mapf); E(P2, P2.f) <=> E(mP2, mP2.mapf);





Meta-model: platforms

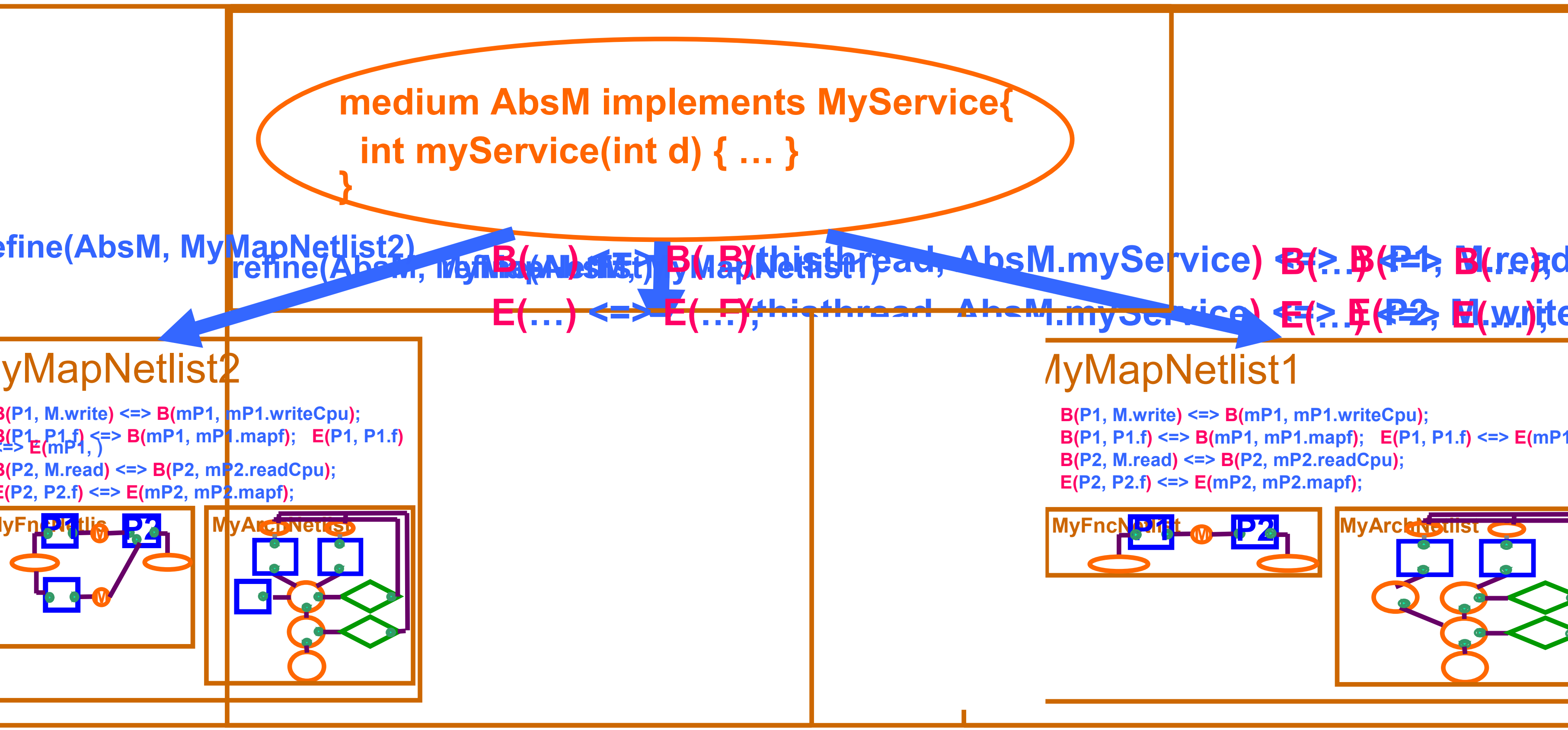
interface MyService extends Port { int myService(int d); }



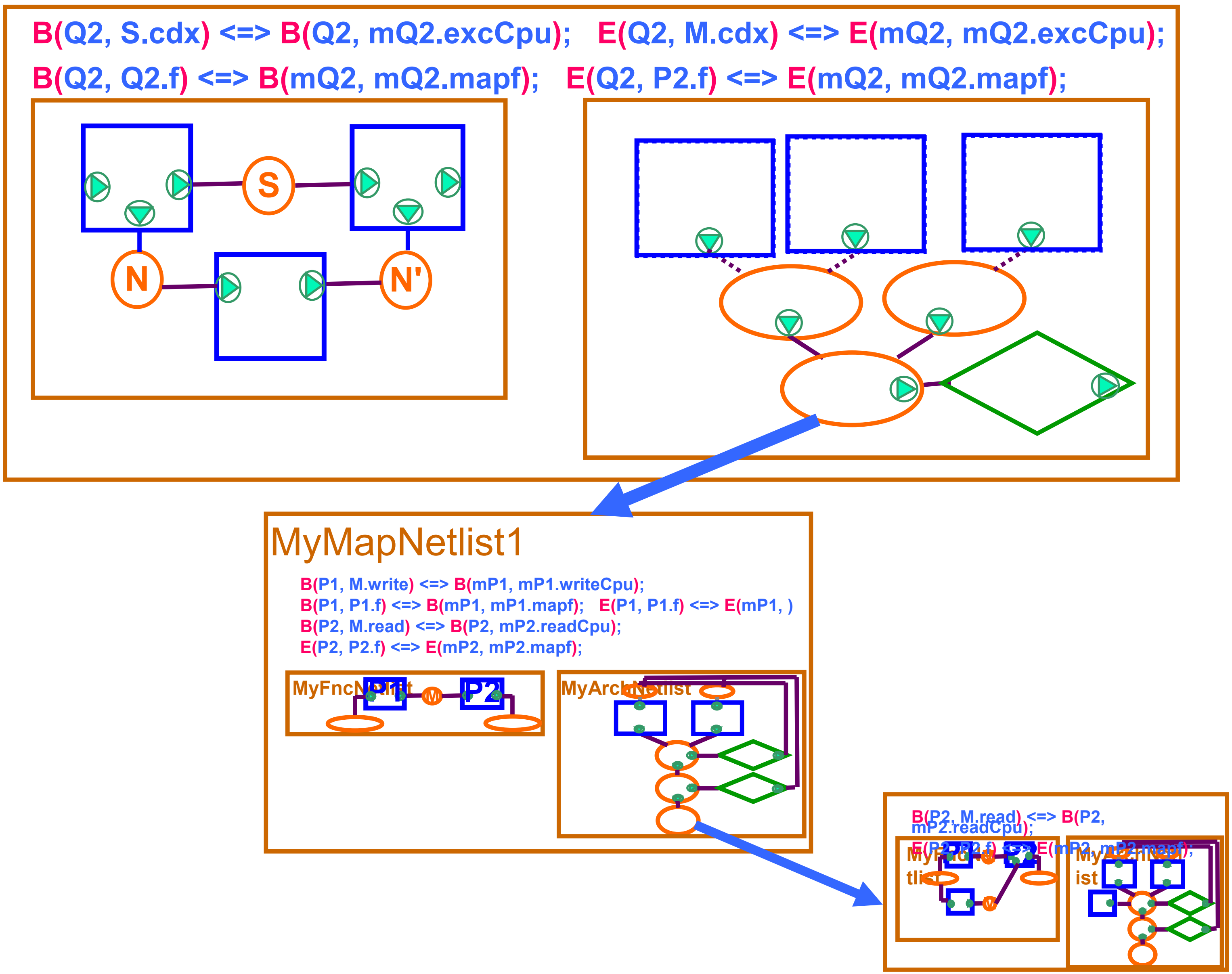
Meta-model: platforms

A set of mapping netlists, together with constraints on event relations to a given interface implementation, constitutes a ***platform*** of the interface.

interface MyService extends Port { int myService(int d); }



Meta-model: recursive platforms

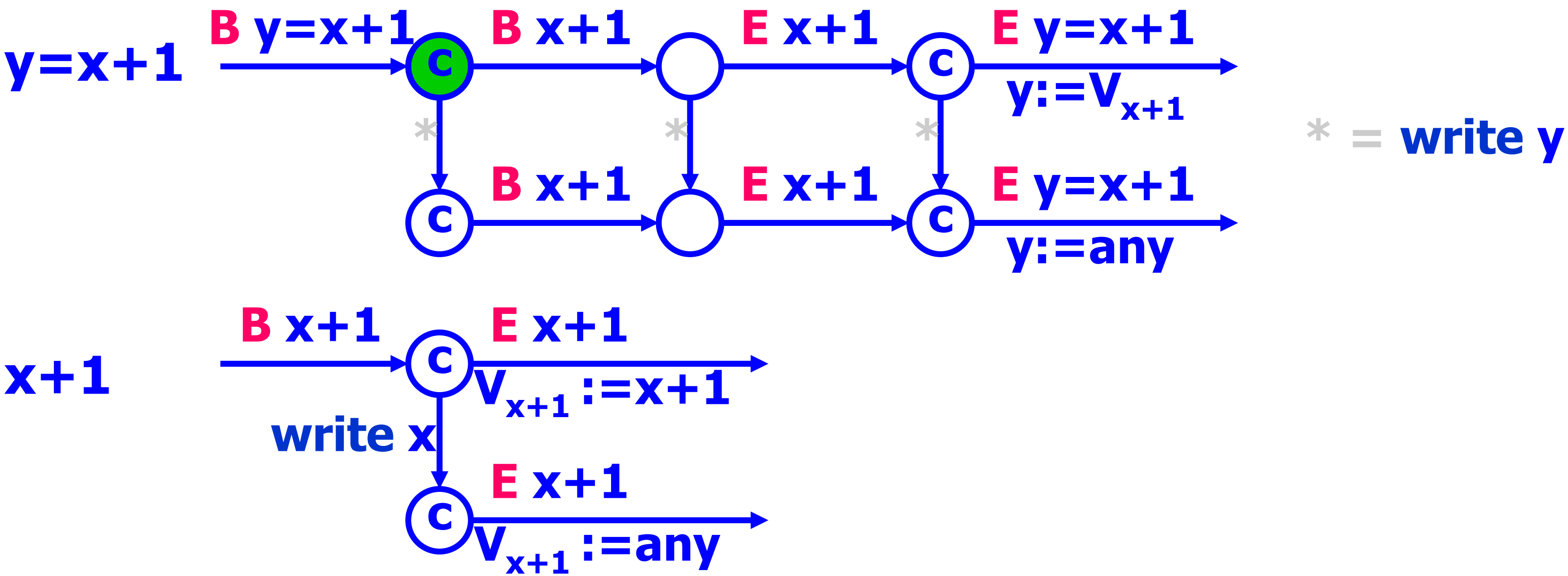


Execution semantics

- **Processes take *actions***
  - **statements and function calls are actions**
    - **e.g.** y=x+port.f();, x+port.f(), port.f()
  - **only calls to media functions are *observable actions***
- **Behaviors are sequences of vectors of *events***
  - **events are beginning of an action (**B port.f()**), end of an action (**E port.f()**), no-op (**N**),**
  - **one event per (sequential) process in a vector**
- **A sequence of vectors of events is a legal behavior if it**
  - **satisfies all constraints**
  - **is accepted by all *action automata* (one for each action of each process)**

Action automata

- y=x+1;

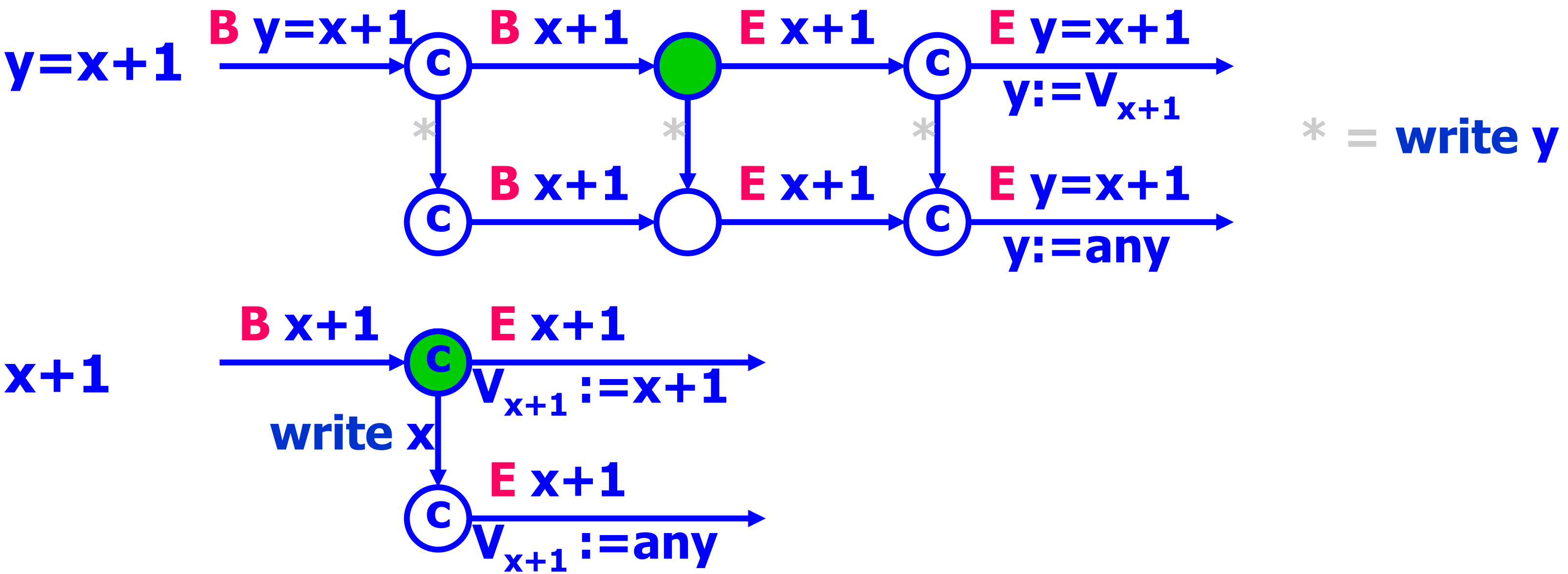


V_{x+1} 0
y 0
x 0

B y=x+1 N

Action automata

- y=x+1;

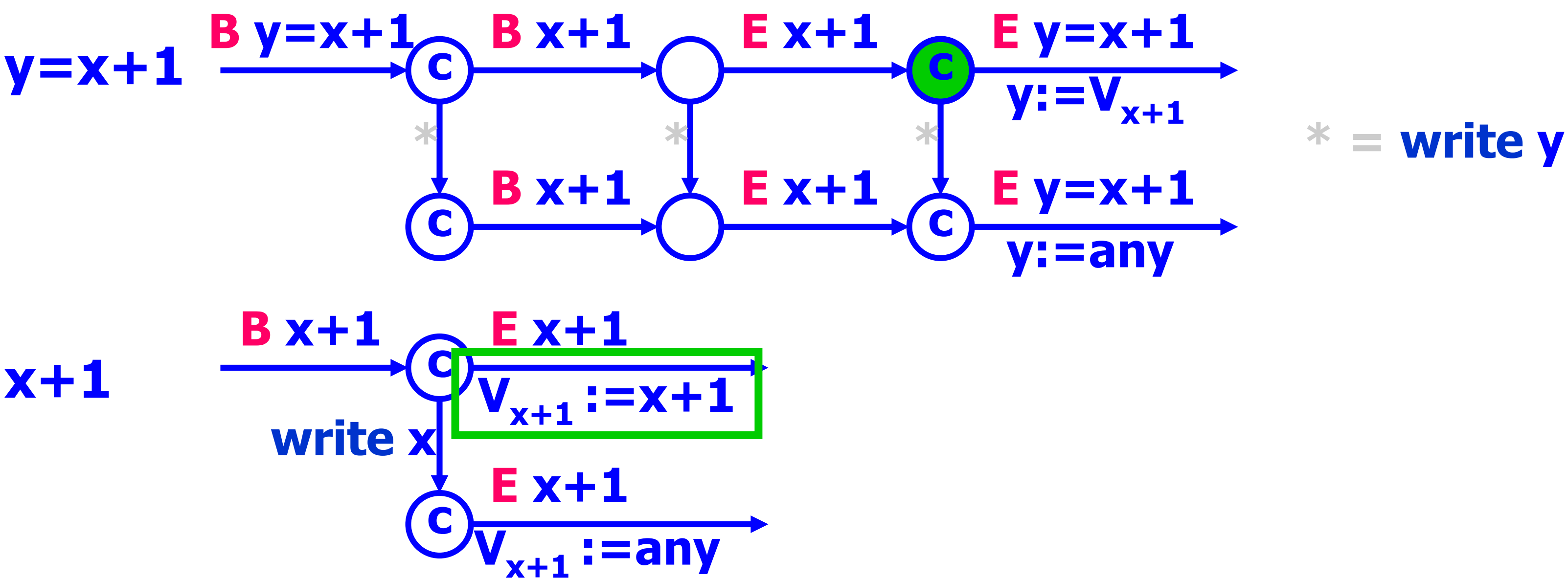


V_{x+1} 0
y 0
x 0

B y=x+1 N B x+1 N N

Action automata

- y=x+1;

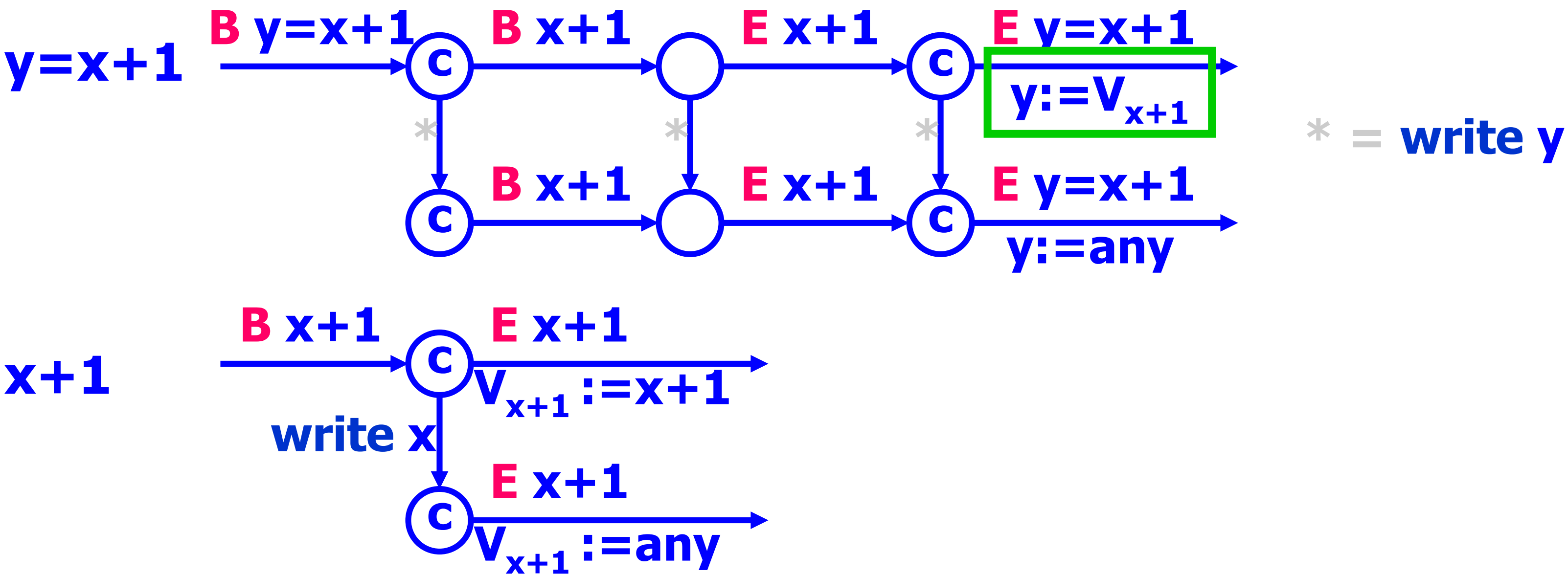


V_{x+1}	0	1
y	0	0
x	0	0

B y=x+1 N B x+1 N N E x+1

Action automata

- y=x+1;

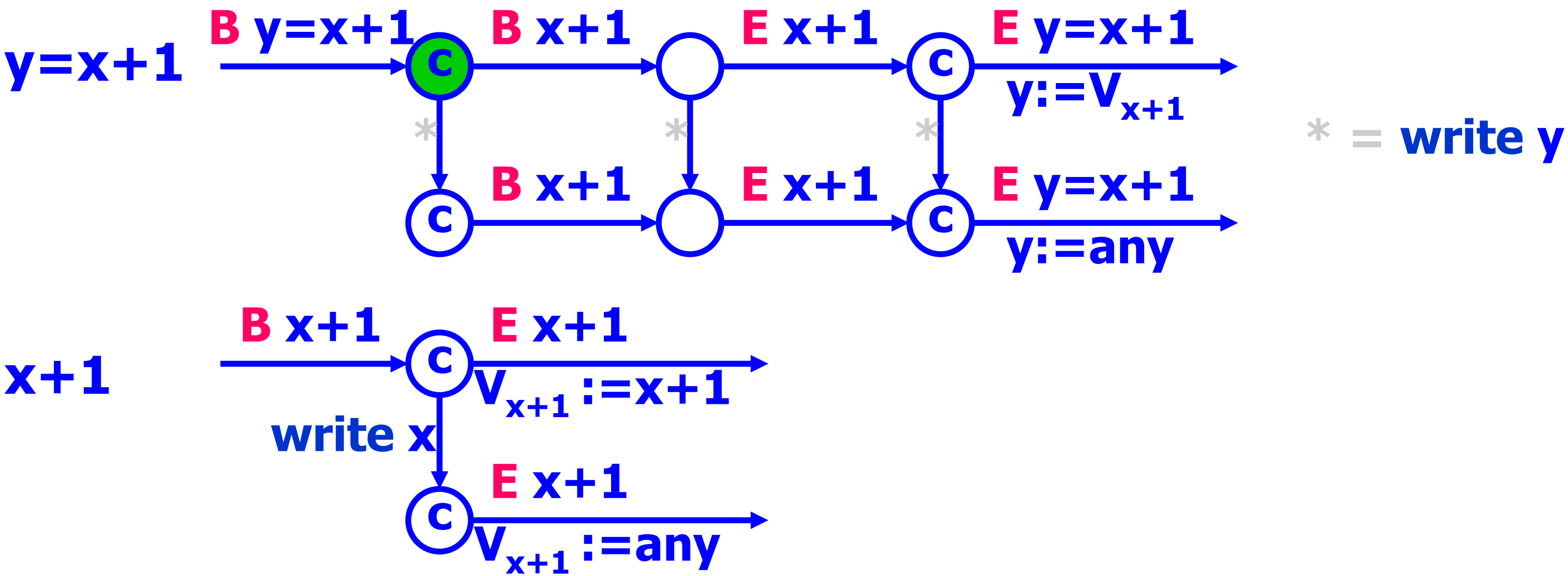


V_{x+1}	0	1	1
y	0	0	1
x	0	0	0

B y=x+1 N B x+1 N N E x+1 E y=x+1

Action automata

- y=x+1;

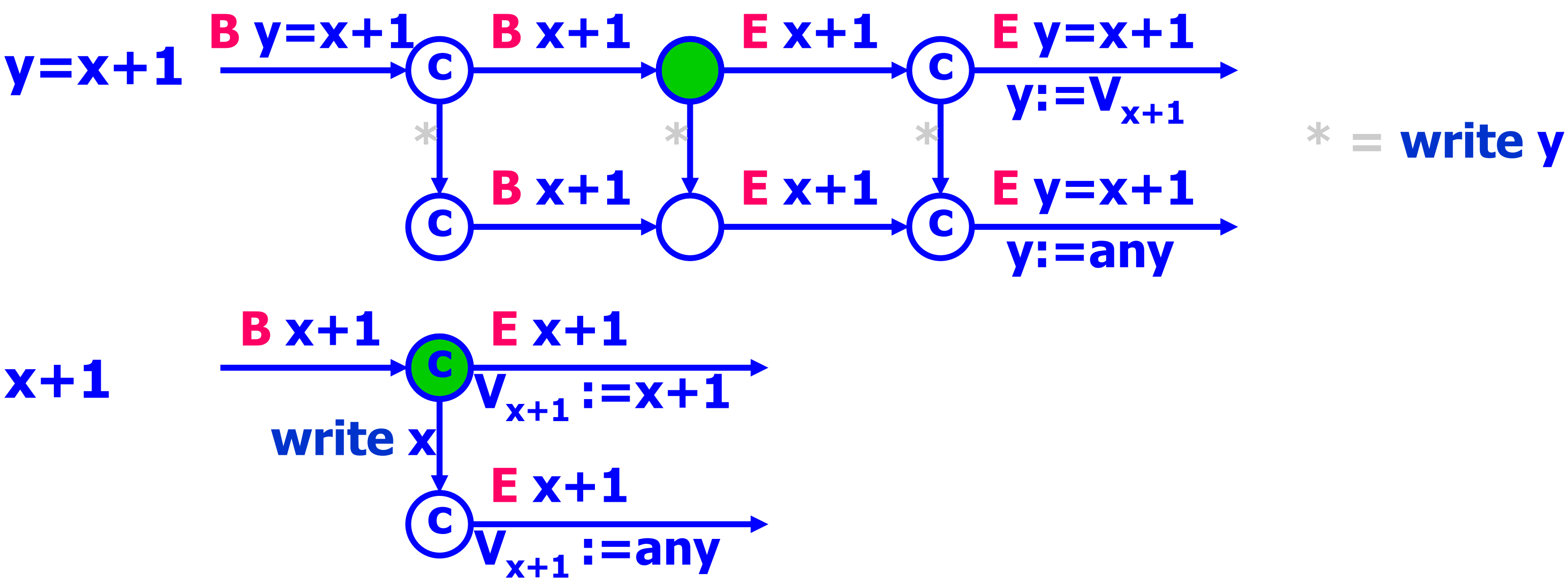


V_{x+1}	0
y	0
x	0

B y=x+1 N

Action automata

- y=x+1;

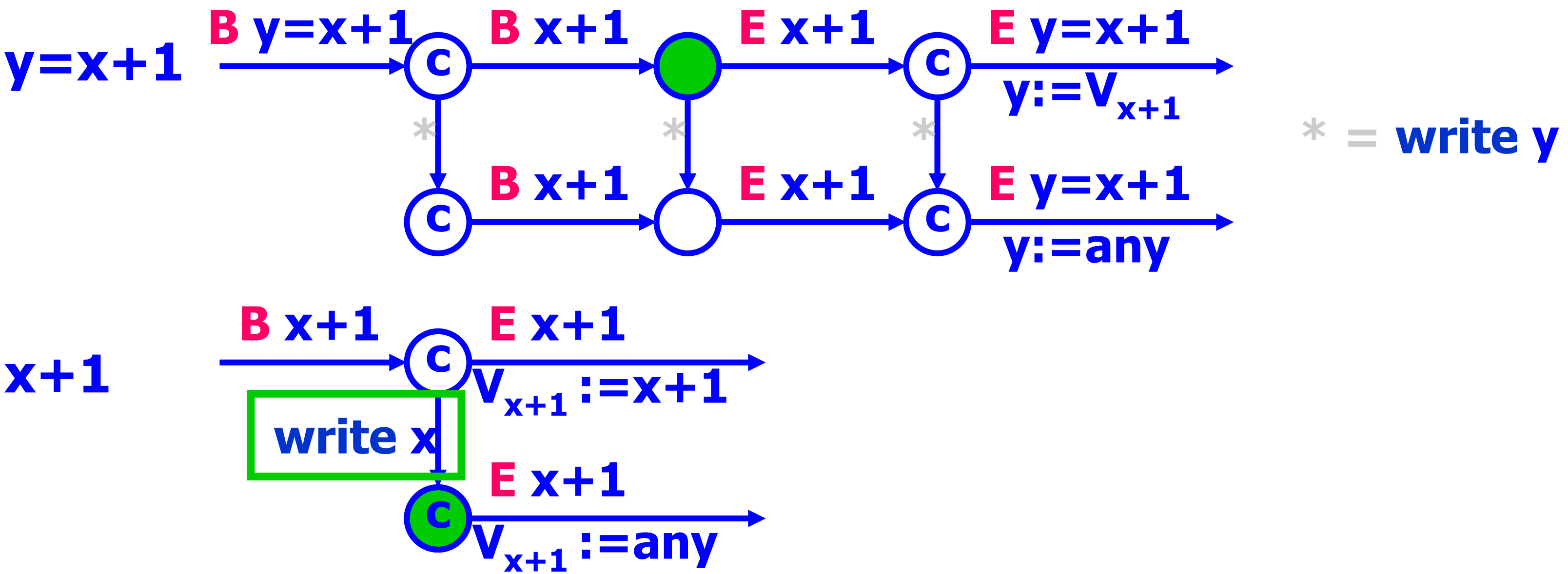


V_{x+1}	0
y	0
x	0

B y=x+1 N B x+1 N

Action automata

- y=x+1;

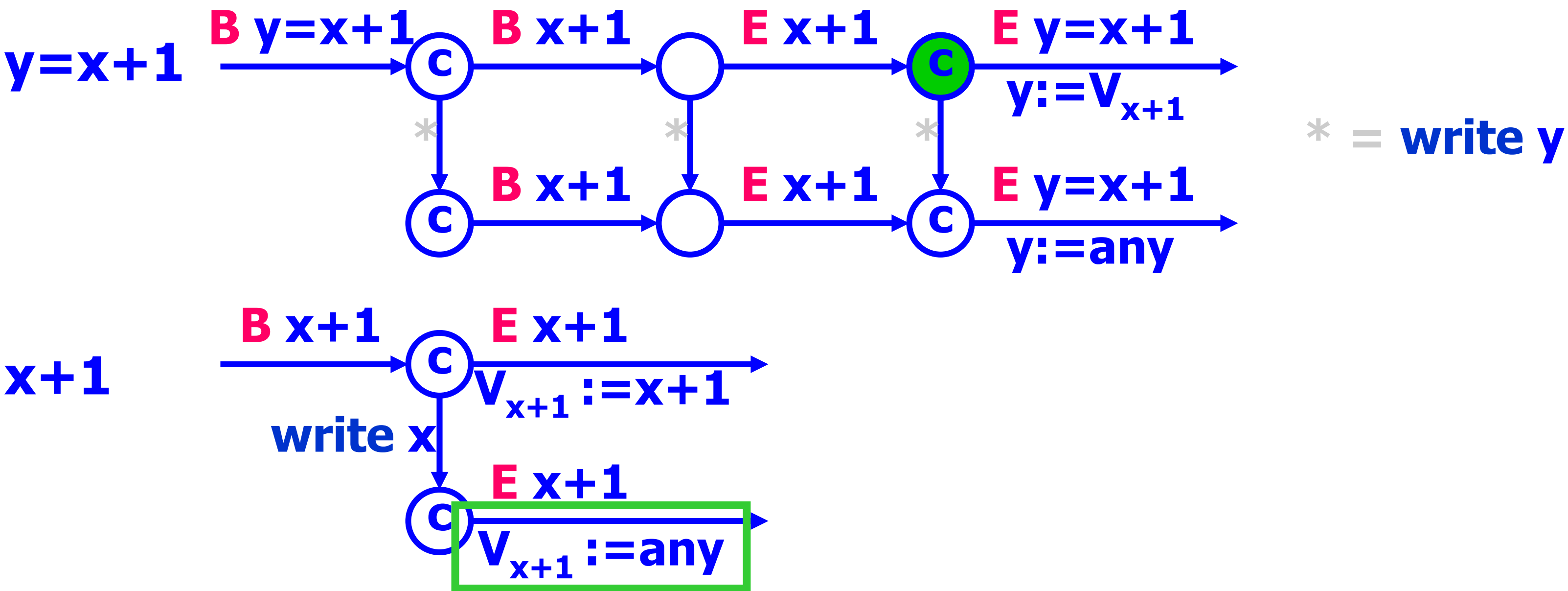


V_{x+1}	0
y	0
x	0

B y=x+1 N B x+1 N N

Action automata

- y=x+1;

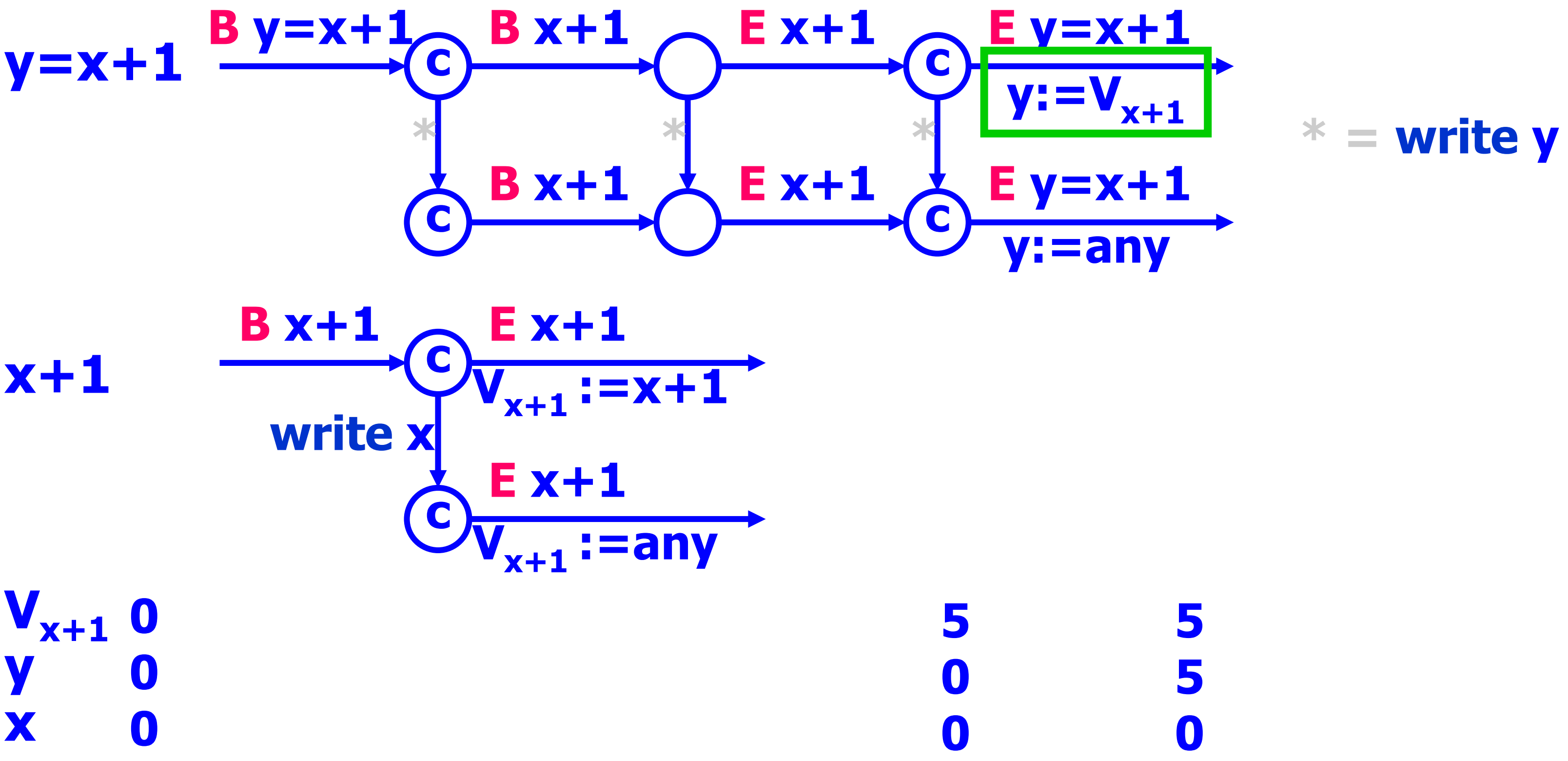


V_{x+1}	0	5
y	0	0
x	0	0

B y=x+1 N B x+1 N N E x+1

Action automata

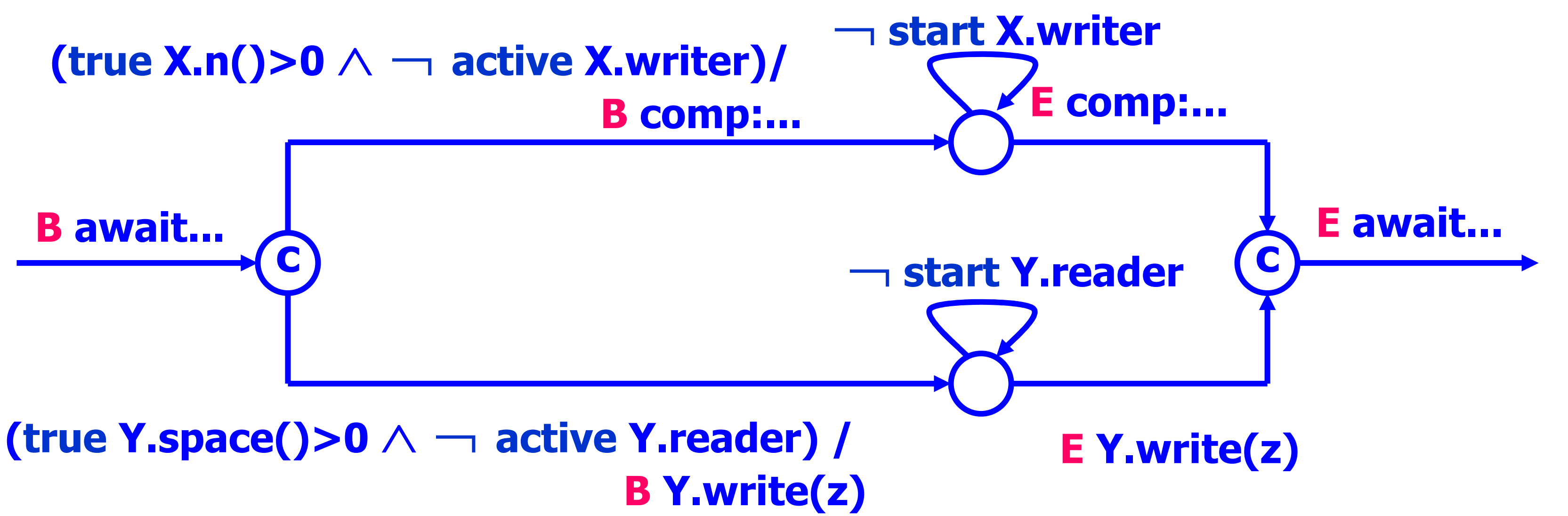
- y=x+1;



B y=x+1 N B x+1 N N E x+1 E y=x+1

Semantics of await

```
await {
    (X.n()>0;X.writer; X.writer)          comp:  z = f(X.read());
    (Y.space()>0;Y.reader;Y.reader)  Y.write(z);
}
```



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 LTL constraints refer

Why ...

- ... bother about concurrency and hazards?
  - they are expensive and dangerous in reality
- ... consider non-determinism and constraints?
  - want to express design freedom simply and precisely
- ... adopt a new synchronization primitive (await)?
  - don’t want to bias towards a particular implementation
  - avoid synchronization objects, talk about actions of processes

Why ...

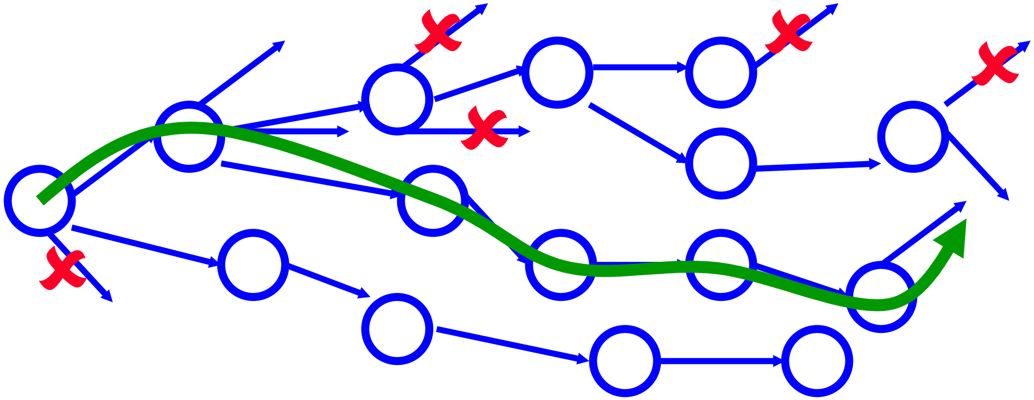
... because we want a framework:

- **that enables synthesis and refinement by allowing precise expression of design space to be explored, in an unbiased way**
- **that enables platform-based design by allowing accurate representation of platform capabilities and limitations**

Cost

- **C, SystemC, HDL's all have semantics that reflect their execution engines (CPU, co-routines, event queue)**
  - **not suitable for us,**
  - **does it improve simulation performance?**
  - **NO, simulating the meta-model can be as efficient as any multi-threaded execution**

Simulation Task

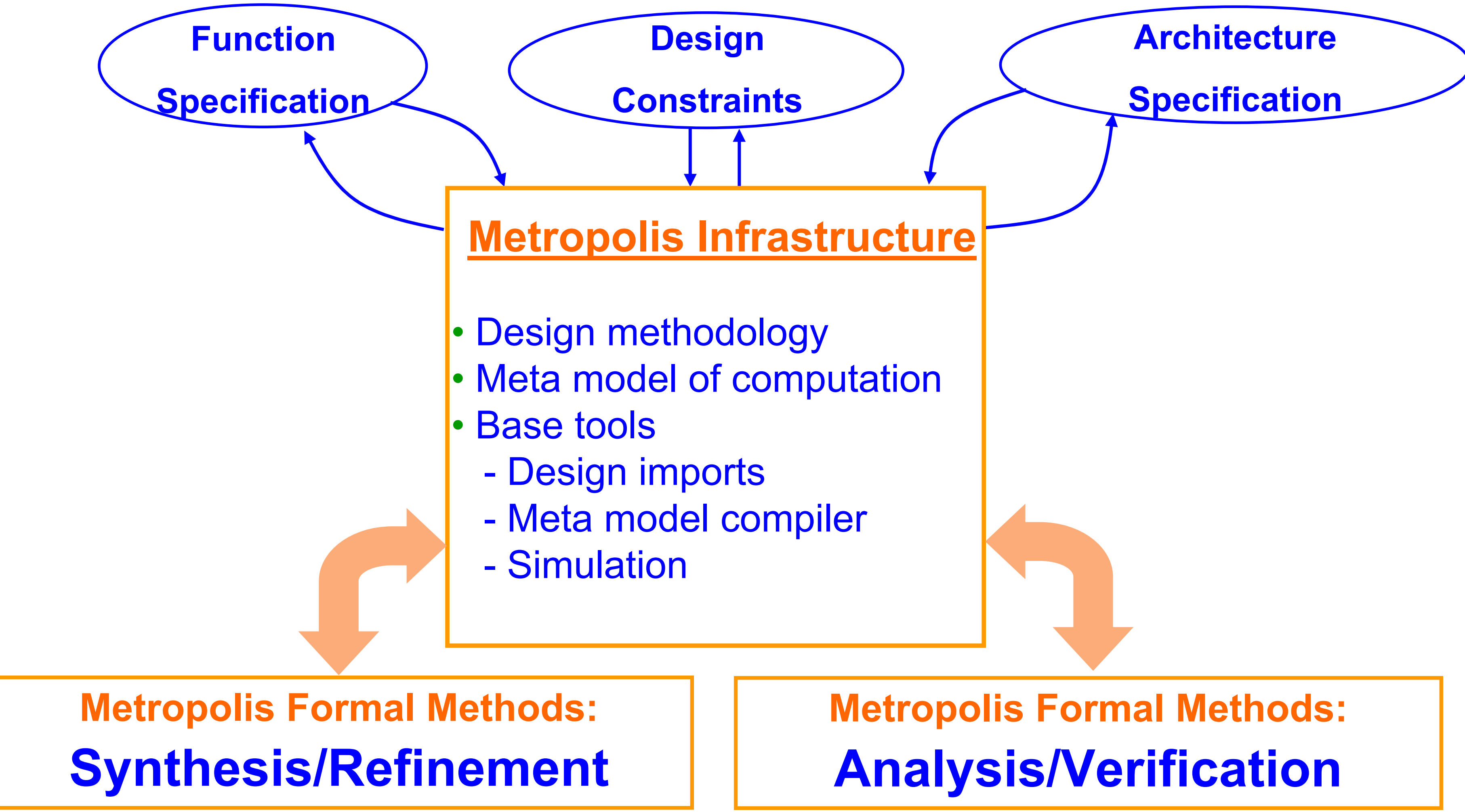


- Choose one execution satisfying awaits and constraints
- Choice may be biased:
  - to minimize context switching
  - to discover corner cases
  - ...

Sequential Simulation Algorithm

	C++	JAVA, SystemC
repeat {		
pick a process	pick one enabled process	pick several enabled processes, and let JAVA or SystemC decide in which order to execute them
run it for a while	until it is blocked	until the next synchronization point
} until done		
	minimize context switches	explore corner cases, parallelize

Metropolis Framework: tools



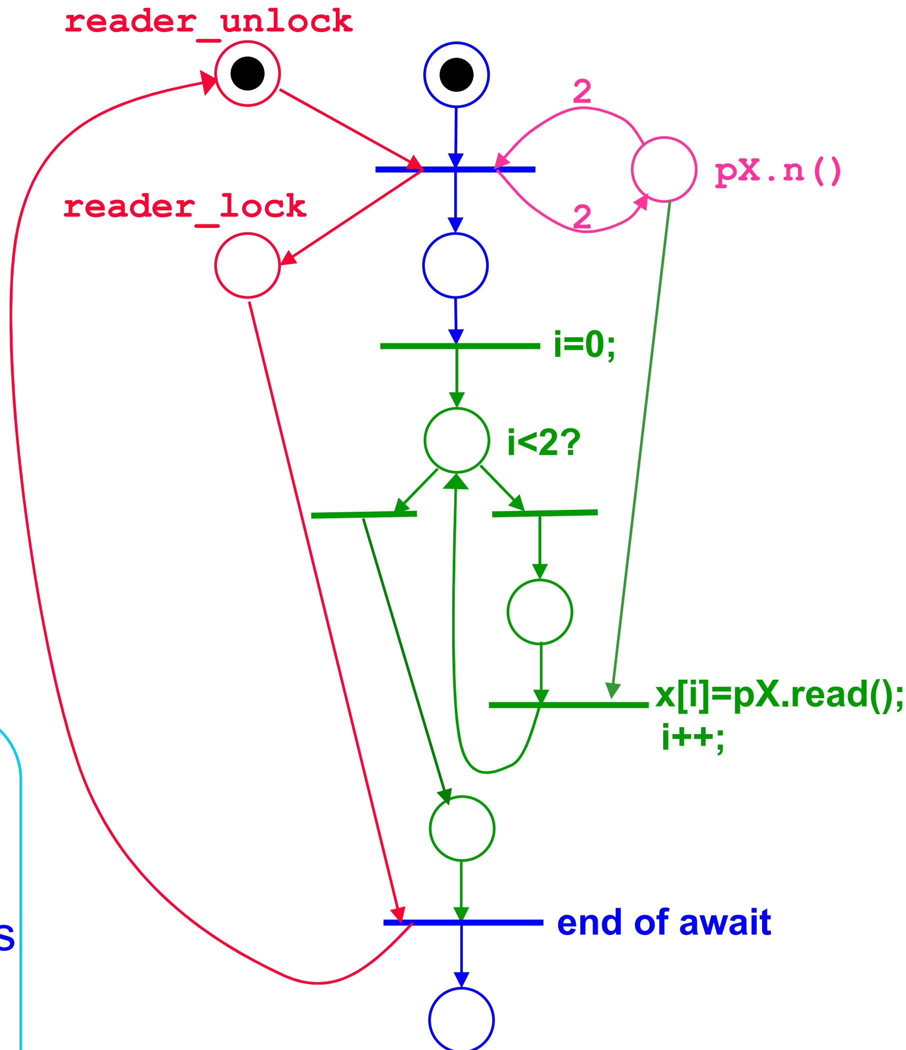
Formal Models for analysis and synthesis

Formal model: derived from the meta-model for applying formal methods

- **Mathematical formulations** of the semantics of the meta model:
  - each construct (‘if’, ‘for’, ‘await’, ...)
  - sequence of statements
  - composition of connected objects
  - → the semantics may be abstracted
- **Restrictions** on the meta model

Formal methods (verification and synthesis) applicable on given models

Example of formal model: Petri nets




```
await(pX.n()>=2)[pX.reader]
  for(i=0; i<2; i++) x[i]=pX.read();
```

Restriction:

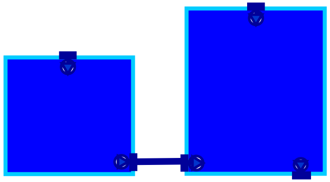
condition inside await is conjunctive.

Formal Methods on Petri nets:

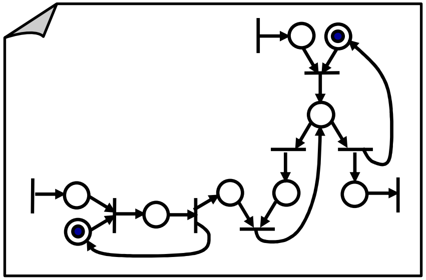
- analyze the schedulability
- analyze upper bounds of storage sizes
- synthesize schedules

Example: quasi-static scheduling

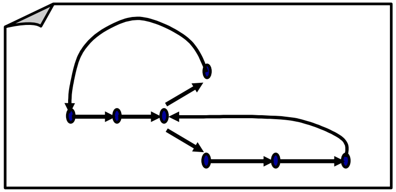
1. **Specify a network of processes**



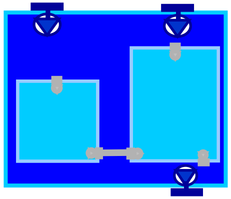
2. **Translate to the computational model**
   - **Petri net**



3. **Find a "schedule" on the Petri net**



4. **Translate the schedule to a new set of processes**

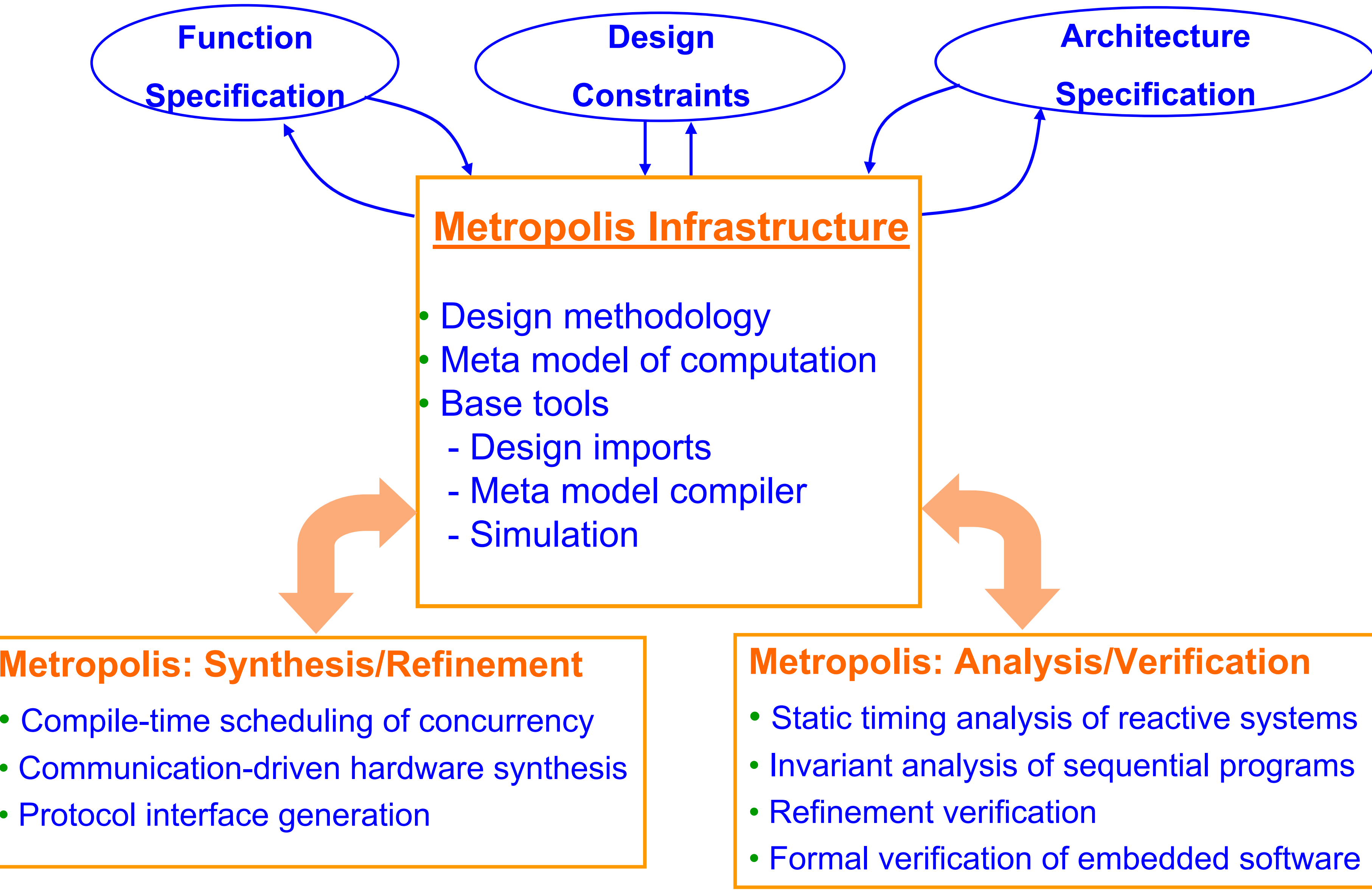


Design automation tools

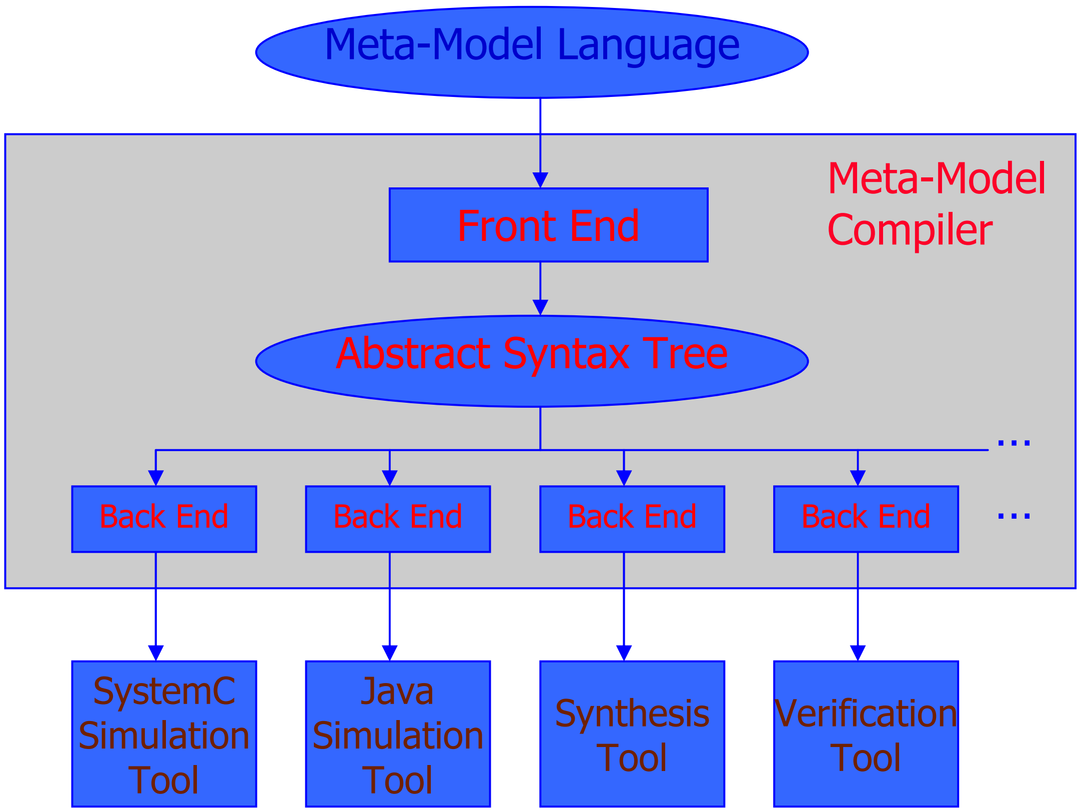
Work in progress:

- **Quasi-static scheduling for multiple processors**
- **Hardware synthesis from concurrent processes**
- **Processor micro-architecture exploration**
- **Communication architecture design (on-chip and off-chip)**
- **Fault-tolerant design for safety-critical applications:functionality and architecture definition and mapping**
- **Communication buffer memory sizing and allocation**

Metropolis Framework



Metropolis Infrastructure



Summary

Metropolis: Metropolis

- Interdisciplinary, intercontinental project (10 institutions in 5 countries)
- Goal:
  - Design methodologies: abstraction levels, design problem formulations
  - Design automation tools:
    formal methods for automatic synthesis and verification,
    a modeling mechanism: heterogeneous semantics, concurrency
- Primary thrusts:
  - Metropolis Meta Model:
    - Building blocks for modular descriptions of heterogeneous semantics
    - Modeling mechanism for function, architecture, and constraints
  - Design Methodology:
    - Multi-media digital systems
    - Wireless communication
    - Fault-tolerant automotive systems
    - Microprocessors
  - Formal Methods and design tools

For more information…

- **Metropolis home page: http://www.gigascale.org/metropolis/**
- **Updated version of the slides: http://polimage.polito.it/~lavagno/metro_mpsoc_03.ppt**
- **Additional (free ☺) advertising: open-source asynchronous implementation of DLX processor, ready for technology map, place and route: http://www.ics.forth.gr/carv/aspida**