Metropolis

GSRC

Metropolis Project Team

L.Carloni, R.Chen, F.DeBernardinis, D.Densmore, T.Meyerowitz, R.Passerone, C.Pinello, A.Pinto, M.Sgroi, G.Wang G.Yang

in collaboration with

F.Balarin, J.Burch, M.Baleani, J.Cortadella, R.Clariso Vliadrosa, H.Hsieh, E.deKock, T.Kam, M.Kishinevsky, A.Kondratyev, W.Kruijtzer, A.La Rosa, L.Lavagno, J.Moondanos, A.Ong, C.Passerone, S.Reikhi, L.Vanzago, Y.Watanabe

Faculty: Alberto Sangiovanni-Vincentelli

MARCO

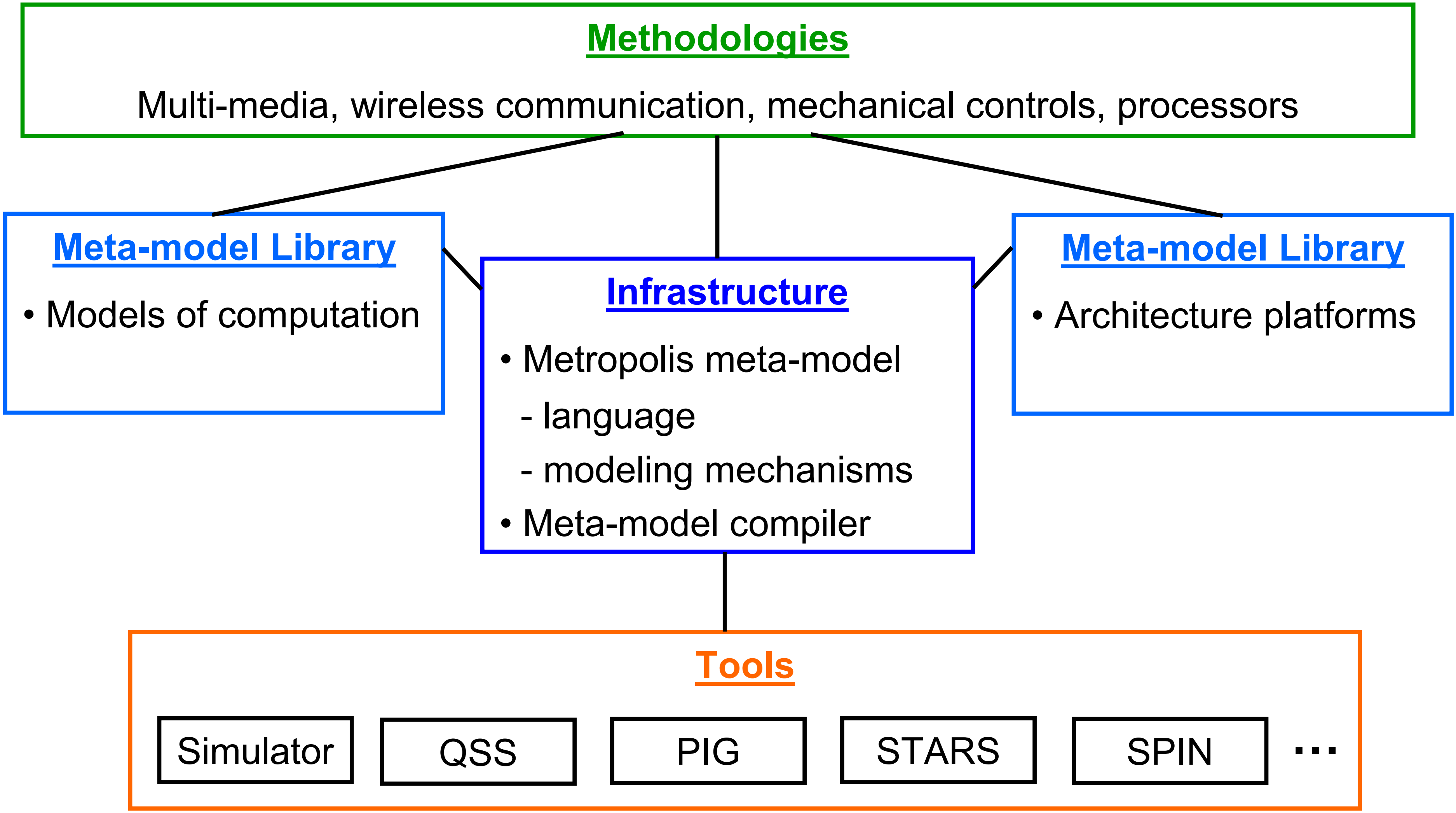
DARPA

SEMICONDUCTOR
INDUSTRY
SUPPLIERS

SIA SEMICONDUCTOR INDUSTRY ASSOCIATION

DUSD(Labs)

Metropolis Framework



Methodologies

Multi-media, wireless communication, mechanical controls, processors

Meta-model Library

- Models of computation

Infrastructure

- Metropolis meta-model
  - language
  - modeling mechanisms
- Meta-model compiler

Meta-model Library

- Architecture platforms

Tools

Simulator

QSS

PIG

STARS

SPIN

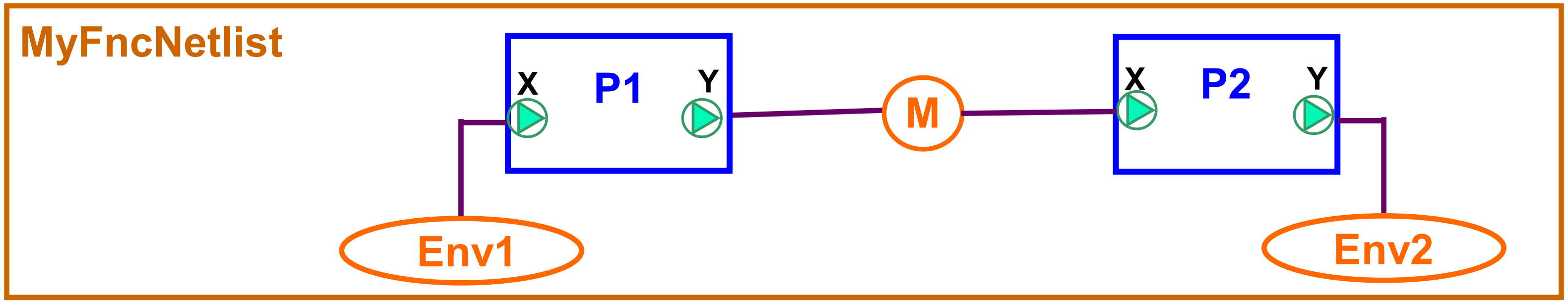
…

Metropolis meta-model

Concurrent specification with a formal execution semantics:

- **Computation** : $f : X \rightarrow Z$
  - **process** : generates a sequence of *events*
- **Communication** : state enumeration 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*.
  - statements and some expressions, e.g.

    y = z+port.f();, z+port.f(), port.f(), i < 10, …

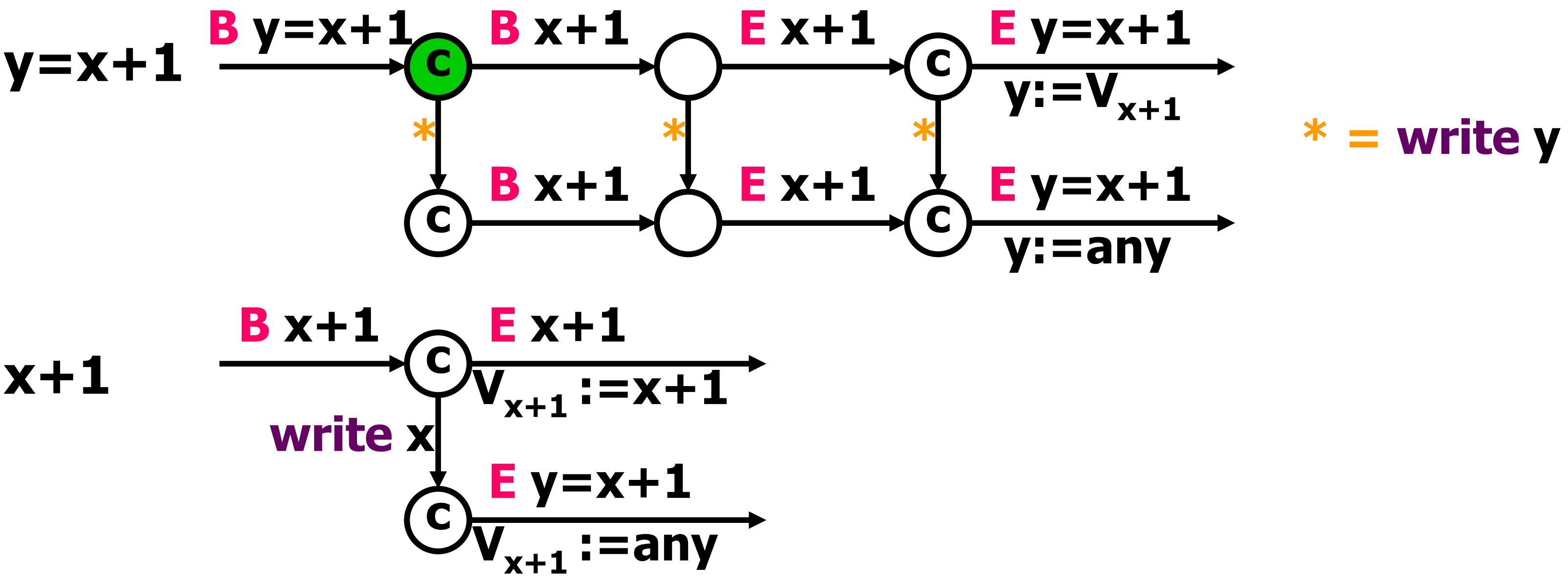
- An *execution* of a given netlist is a sequence of vectors of *events*.
  - *event* : the beginning of an action, e.g. B(port.f()),

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

Meta-model: action automata

- y=x+1;



V_{x+1}	0
y	0
x	0

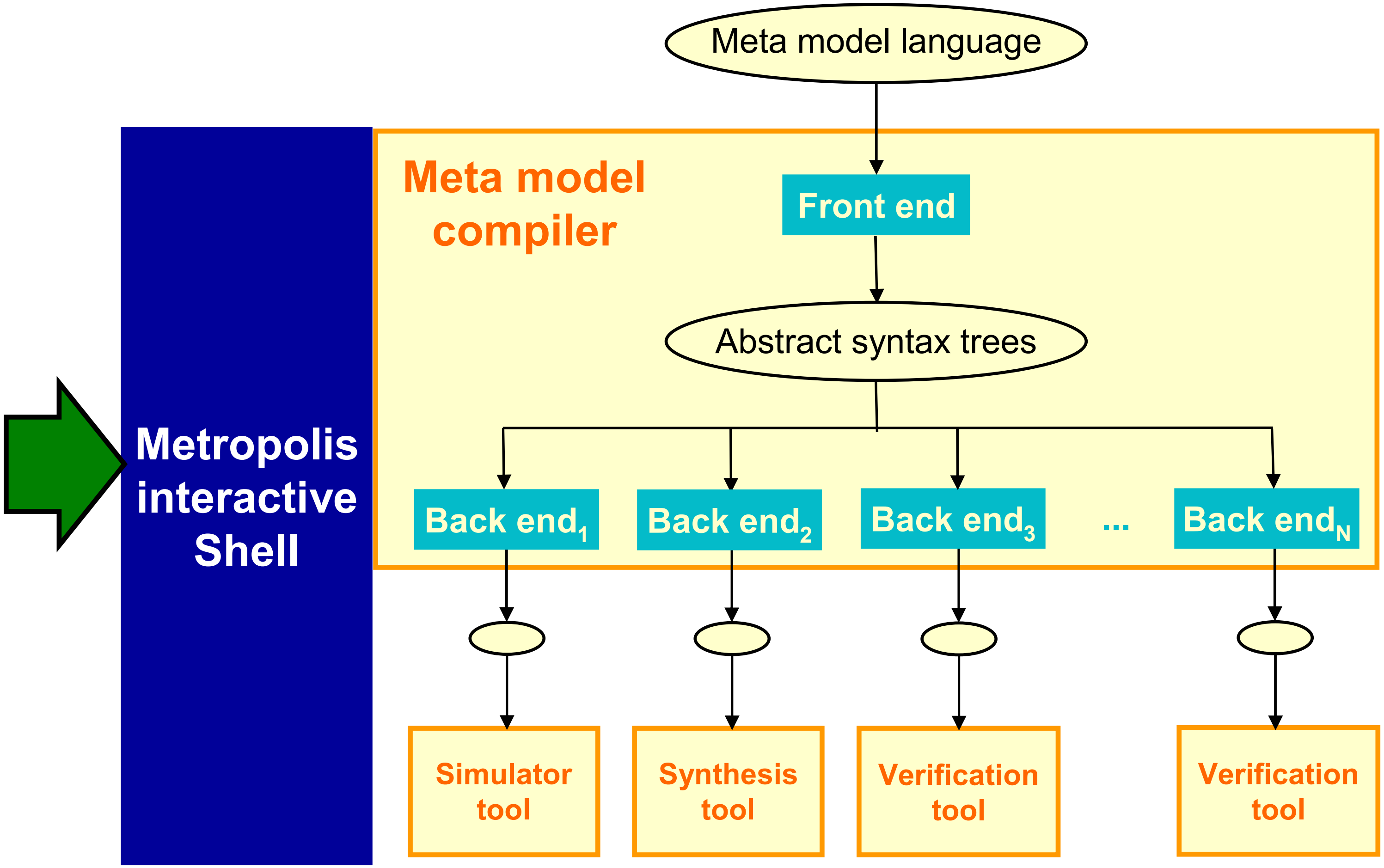
B y=x+1 N

Meta-model: execution semantics

- 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
  - await's
  - function calls and labels to which await-s or LTL constraints refer

Metropolis design environment

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



Meta model language

Metropolis interactive Shell

Meta model compiler

Front end

Abstract syntax trees

Back end$_1$

Back end$_2$

Back end$_3$

…

Back end$_N$

Simulator tool

Synthesis tool

Verification tool

Verification tool

Formal models from the meta-model

Example: Petri nets

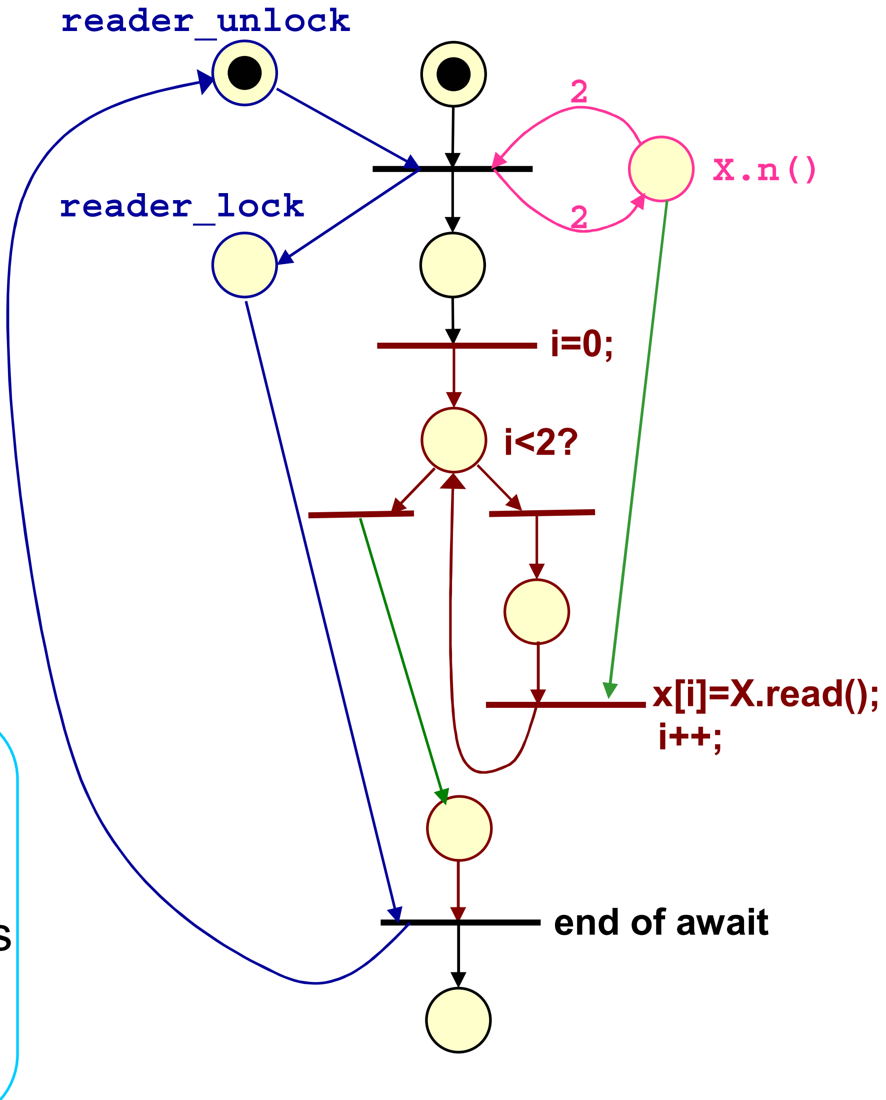
```
await(X.n()>=2; X.reader; X.reader)
  for(i=0; i<2; i++) x[i]=X.read();
```

Restriction:

condition inside await is conjunctive.

Formal methods on Petri nets:

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


reader_unlock
2
X.n()
reader_lock
2
i=0;
i<2?
x[i]=X.read();
i++;
end of await