Specification, Verification and Optimization of Real-Time Distributed Embedded Systems

Algorithm Architecture Adequation Methodology

Yves SOREL
Yves.Sorel@inria.fr

http://www.syndex.org

INRIA

French National Institute for Research in Computer Science and Control

http://www.inria.fr

- **Since 1968**
- **3000 People**
- **5 centers**
- **Budget: 90 MEuro**
- **20 Startup Companies**

- **Research**
- **Knowledge Transfer**
- **Training**
- **W3C**
- **International collab.**

4 Research Themes

1. **Network and System**
2. **Software Engineering and Symbolic Computation**
3. **Human-Computer interaction, Image Processing, Robotics, Data and Knowledge Management**
4. **Simulation and Optimisation of Complex Systems**

INRIA Context

- **Network and System (Theme 1C)**
- **Modeling Analysis Optimization of Distributed Real-Time Embedded Systems (AOSTE Team)**
- **Algorithm-Architecture Adequation Methodology (SynDEx software) for Optimized Rapid Prototyping and Implementation of Distributed Real-Time Embedded applications**

Applications

Automobile (AEE, EAST, ECLIPSE)

Mobile Robotic (CyCab)

Signal Processing: Software radio (Mitsubishi ITE)

Telecommunication: SoC UMTS (PROMPT)

Image Processing: Automatic Guidance (MBDA), MPEG4 (INSA)

…

Goals

- **Safe Design**
- **Rapid Prototyping**
- **Optimization**
- **Automatic Code Generation**
- ***Reduced Development Cycle***

Characteristics

- **Algorithms**: Automatic Control, Signal & Image Processing
- **Reactive**: *Stimulus event - Operations – Reaction event*
- **Real-Time**: Constraints: Latency = bounded Reaction Time
  Cadence = bounded Input Rate
- **Distributed**: Power, Modularity, Wires minimization
  - **Heterogeneous Multicomponent Architecture**
    - Network of Processors and Specific Integrated Circuits
    - Specific Integrated Circuits = ASIC, ASIP, FPGA, IP
- **Embedded**: Resources minimization

Algorithm Architecture Adequation Method.

- **Global approach** based on the Synchronous Languages Semantics and the hardware RTL models
- **Unified Model:** Directed graphs
  - **A**lgorithm: Operation / Data-Conditioning Dependence
  - **A**rchitecture: FSM / Connection
  - Implementation: Graphs Transformations
- **Adequation:** Optimized Implementation (best matching)
- **Macro-Generation:**
  - Real-Time Executives for Multicomponent
  - Structural VHDL for Integrated Circuit Synthesis

Static versus Dynamic Approaches

AAA Methodology		**Classic Design**	
Static →	Dynamic	Dynamic →	Static
as much as possible)	(when unavoidable)		

tatic : distrib./sched. off-line without preemtion
ynamic : distrib./ sched. in-line or off-line with preemption

Advantages
- Static : deterministic, low over-head
- Dynamic : data dependent durations

Drawbacks
- Static : not allways possible, knowledge of environment necessary
- Dynamic : high over-head, soft real-time

Algorithm Specification: Directed Hyper-Graph

Vertex: Conditioned Operation: inputs-computations-outputs

Directed Hyper-Edge: Dependence (Diffusion):
Data with or without Precedence, Precedence only, or Conditioning (Exec or not)

=> ***Partial Order = Potential Parallelism***

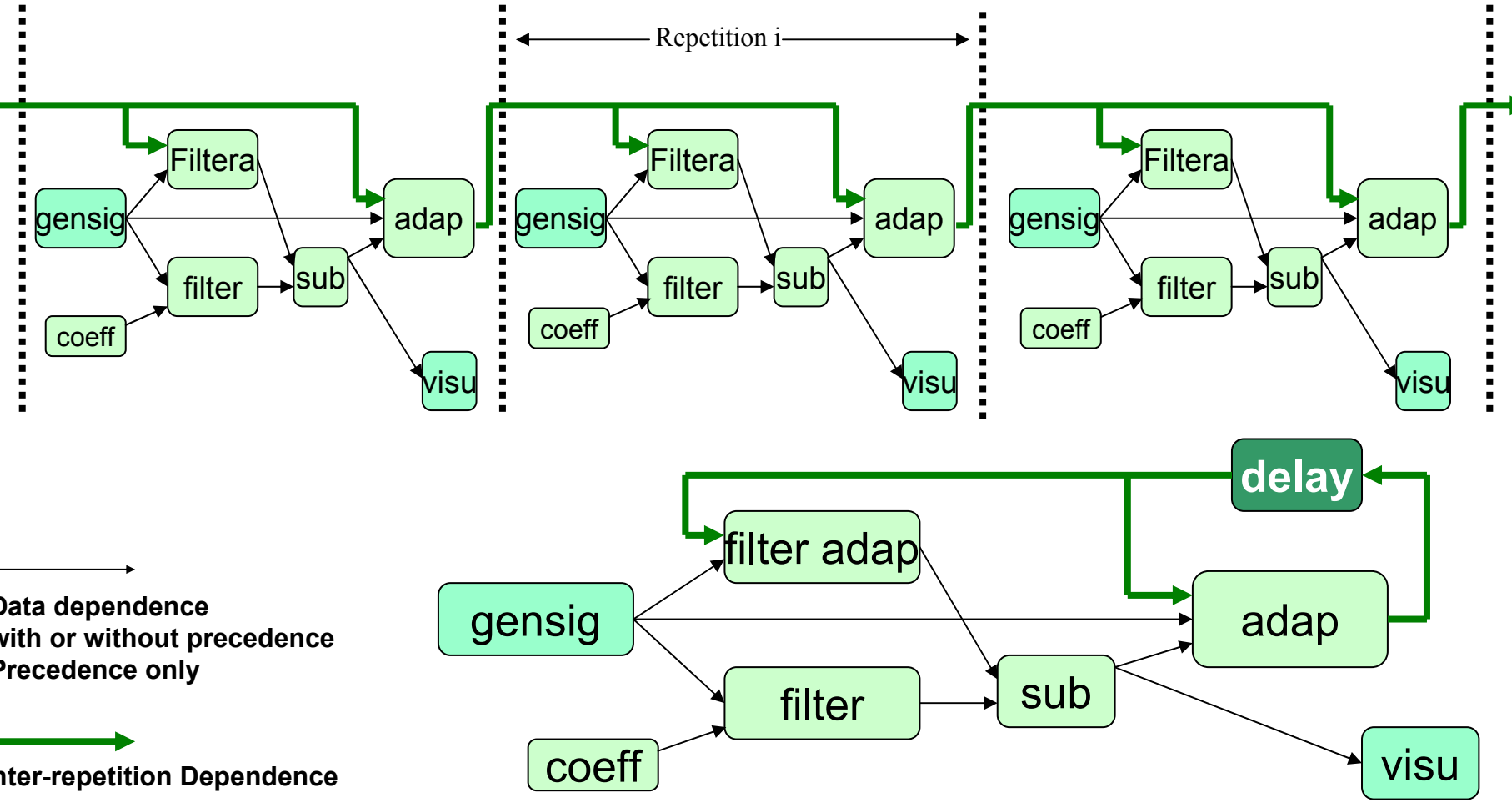
Infinite Repetition of sub-graph: Reactive Infinite Loop

Finite Repetition of sub-graph: Finite Loop

→ **Factorized Conditioned Data-Dependence Graph**

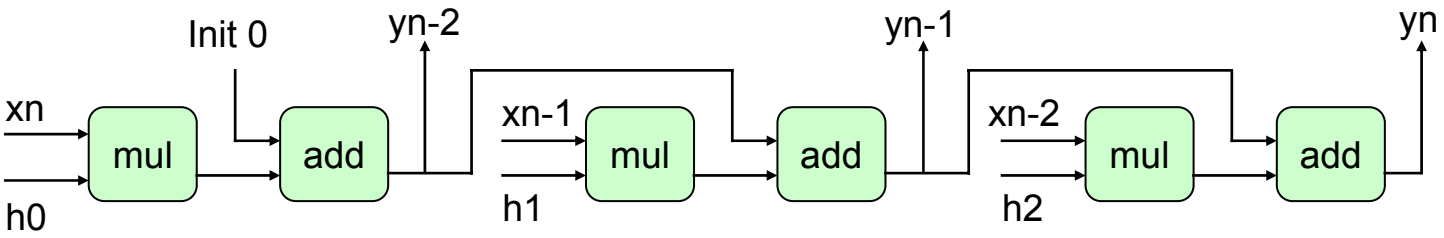
May be obtained from compilers of: Synchronous Languages, AIL, Scicos, AVS, CamlFlow,...

Algorithm example: Adaptative Equalizer



3 coefficients digital filter: finite repetition of 3

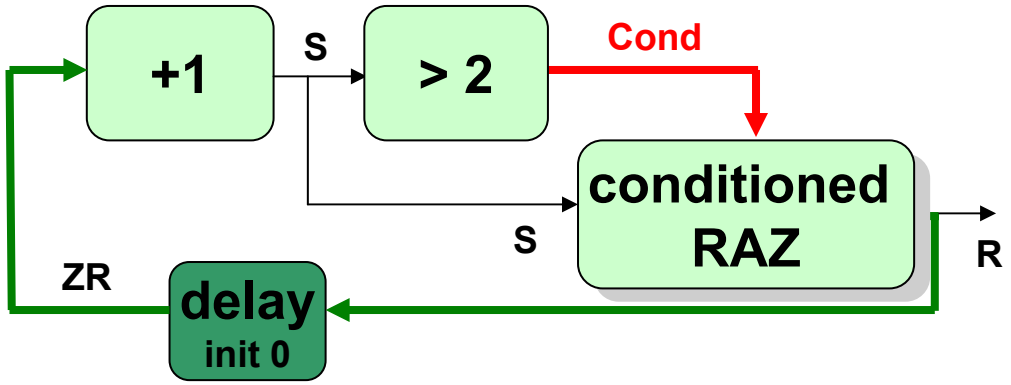
$$Yn = \Sigma(hi*Xn-i) \text{ for } i=0 \text{ to } 2$$

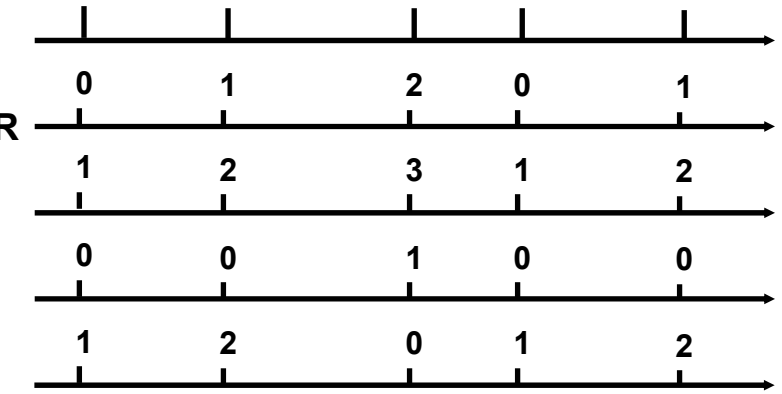


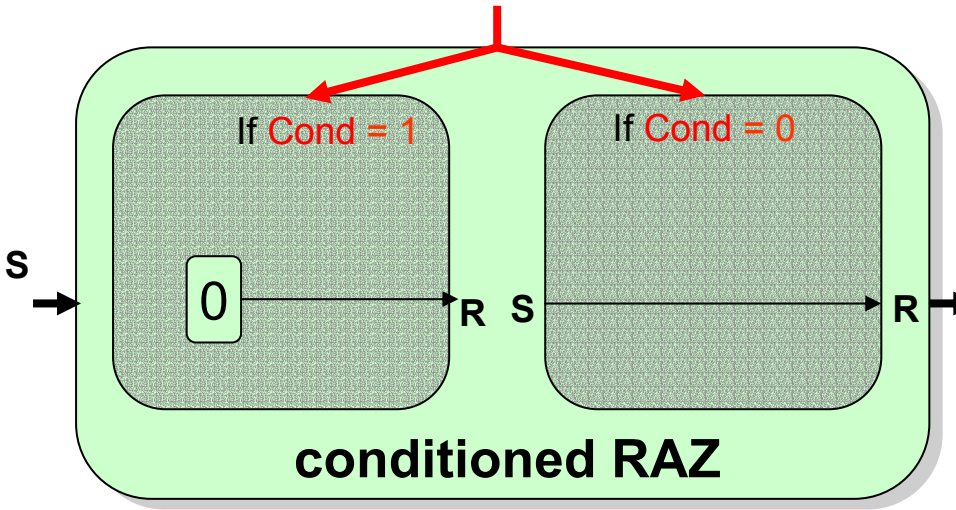
Fork : H(h0, h1, h2) et X(xn, xn-1, xn-2)

Join : Y(yn, yn-1, yn-2)

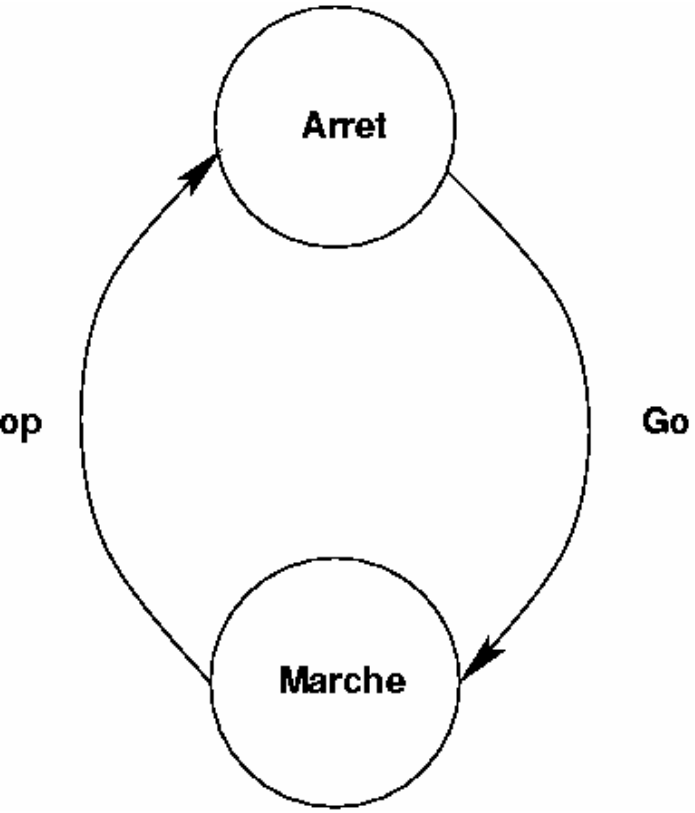
Conditioned Algorithm: Modulo 3 Counter

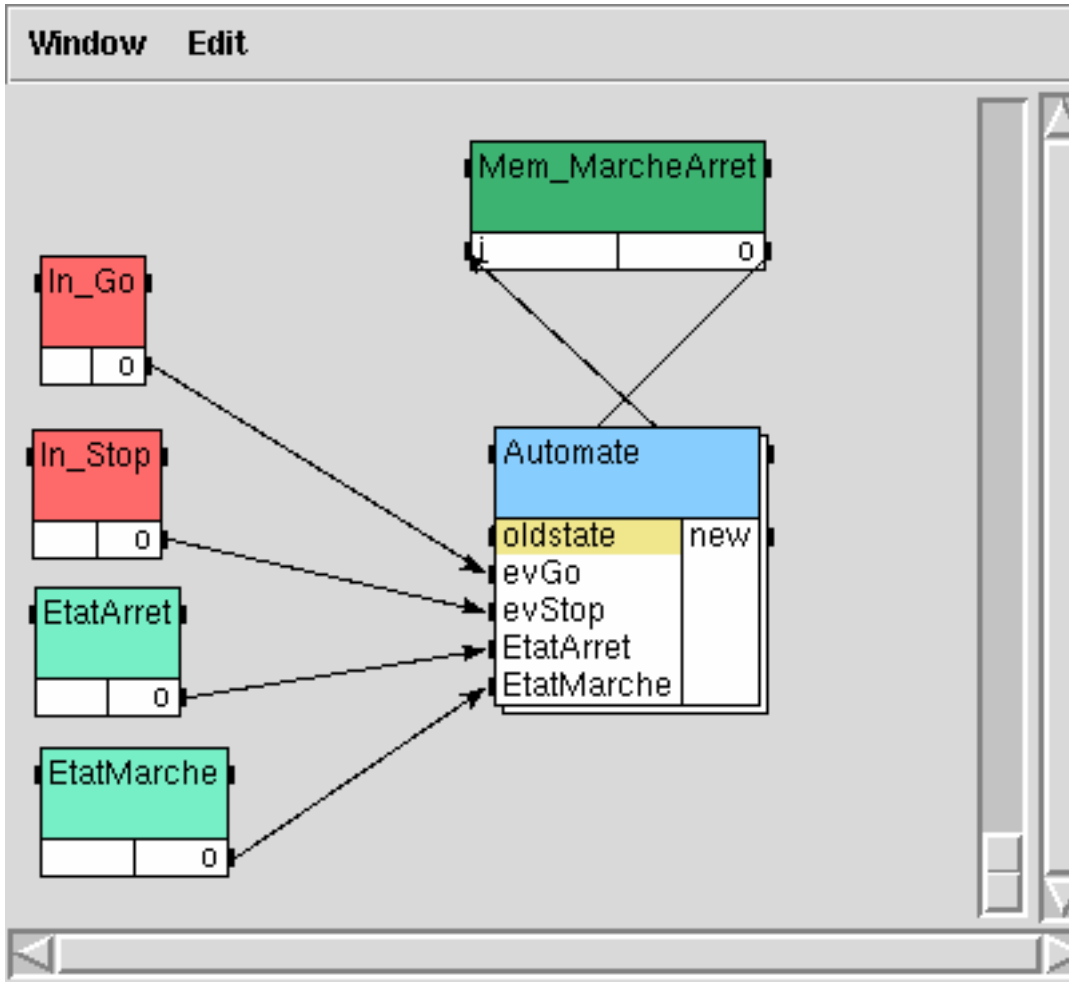




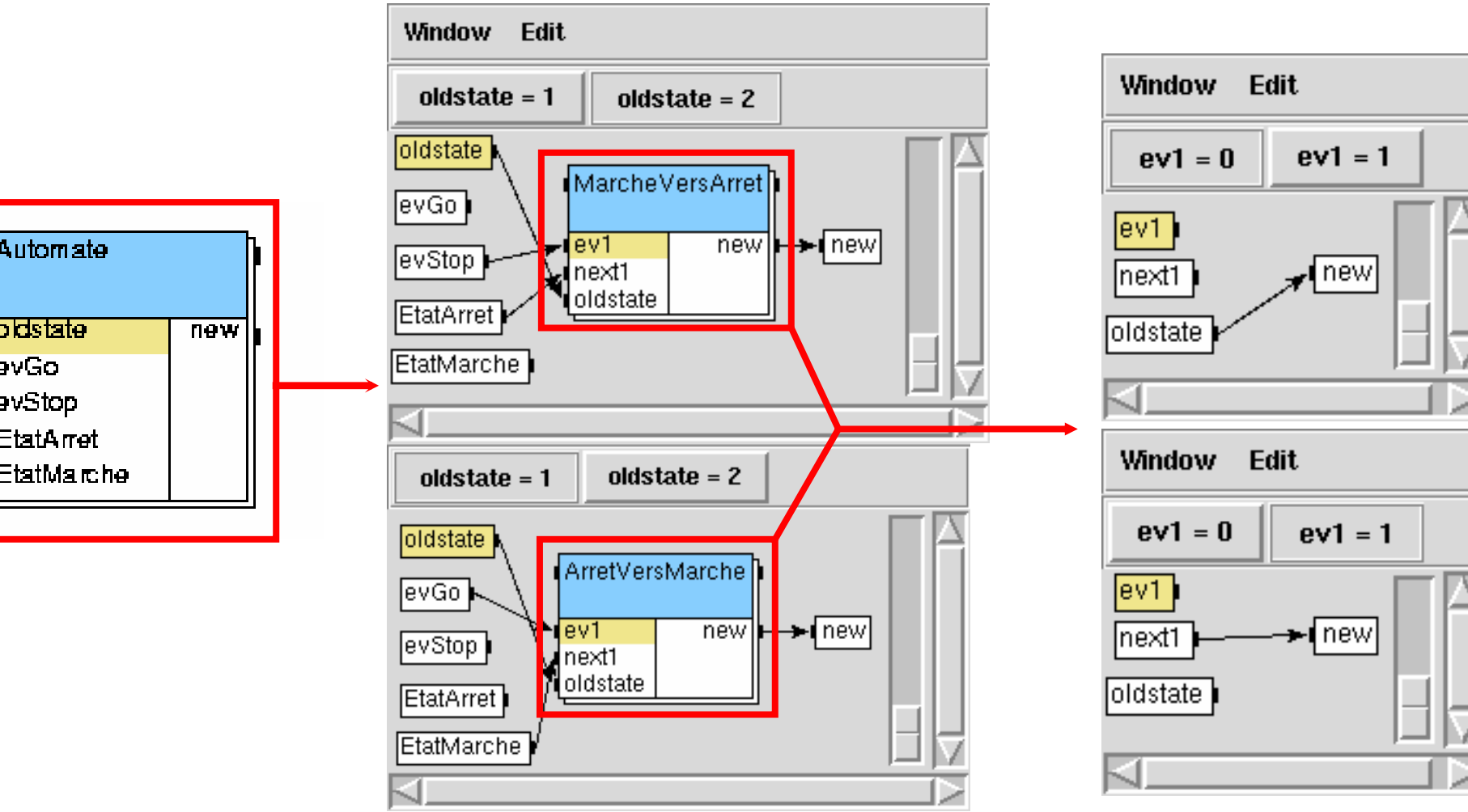


Finite State Machine with AAA/SynDEx (1/2)

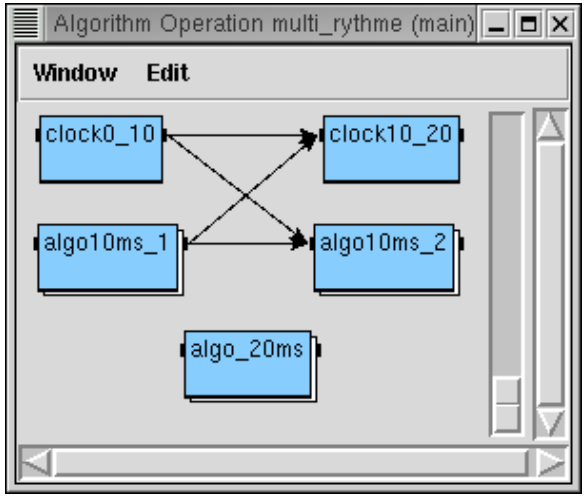


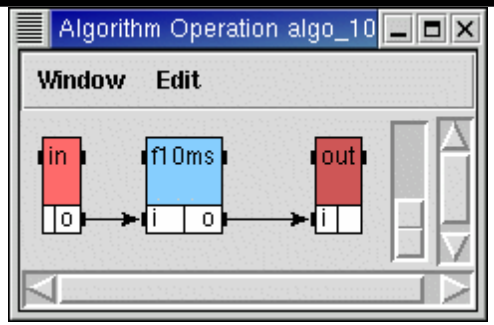


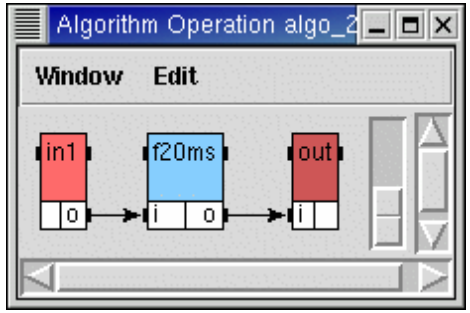
Finite State Machine with AAA/SynDEx (2/2)

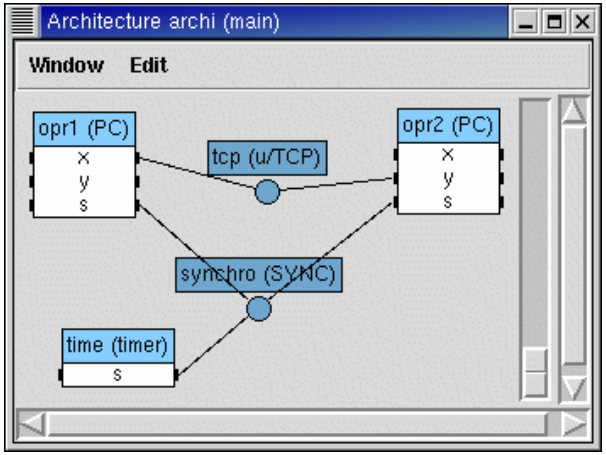


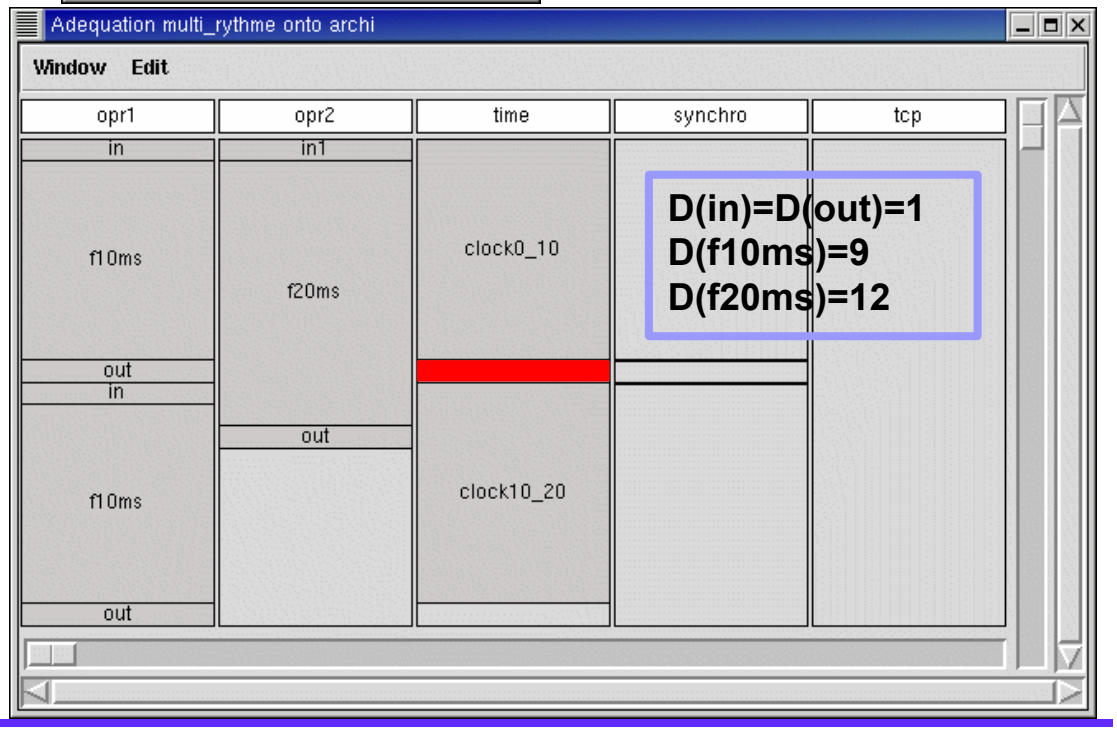
Multi-period











Algorithm Verification

- **Synchronous Languages**
  - Esterel, Lustre, Signal, SyncCharts ...
  - Modular Specification
  - No Hardware constraint, independent from Physical Time => Logical Time, events ordering
  - Reaction ***simultaneous*** with Stimulus
  - Verifications:
    - Dependence Cycle ***only with*** Delay
    - Reactions order ***consistent*** with Stimuli order
    - Logical Temporal Properties: ex. event always occurs

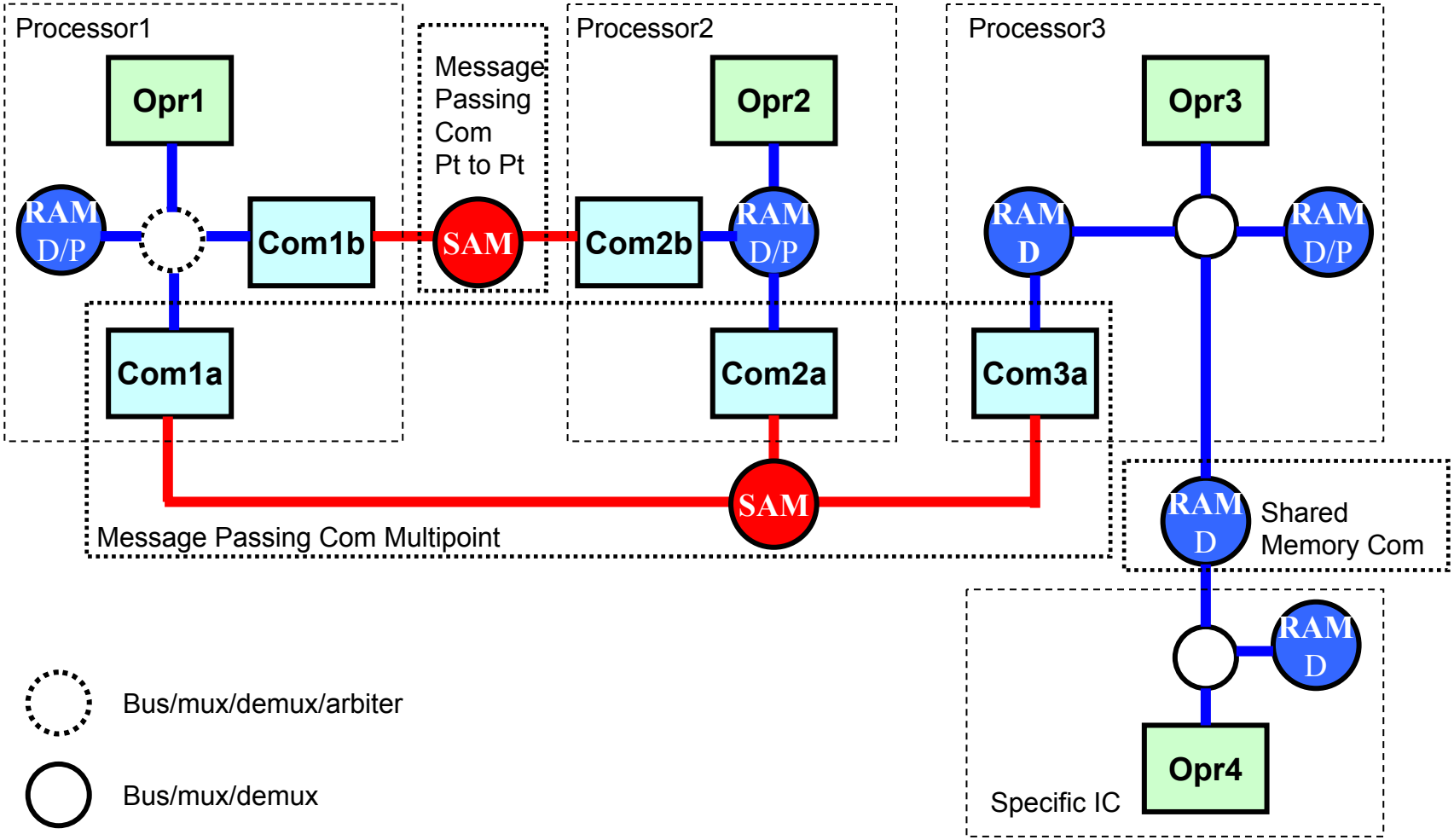
Architecture Specification: Directed Graph (1)

- **Vertice (FSM sequential machine)**
  - Operator: executes sequentially operations
  - Communicator: executes sequentially communications
  - Memory with or without Arbiter
    - Random Access (RAM): non synchronized comm.
      - Stores Program and Data, Shared memory Comm.
    - Sequential Access (SAM): synchronized comm., R/W order
      - Stores Data
      - Message passing Comm., Point-to-point, multi-point with or without hardware diffusion
  - Bus/mux/demux with ou without arbiter
    - Bus/mux/demux: selects a memory among several
    - Bus/mux/demux/arbiter: arbitrates shared memory

Architecture Specification: Directed Graph (2)

- **Directed Edge**
  - Connection between two vertices: models data transfers
  - Connections must follow a set of rules:
    - Operators must not be connected together
    - Communicators must not be connected together
    - Memories must not be connected together
    - Bus/mux/demux may be connected together
    - ...
- **Macro-RTL Model:** operation, macro-register

Architecture Example



Multicomponent Implementation (1)

The set of all possible implementations is described as the composition of three binary relations:

(Gal, Gar) $\xrightarrow{\textit{rout o distrib o sched}}$ (Gal', Gar')

Routing: creation of all the inter-operator routes

Distribution: *spatial allocation*

- Partitioning and allocation: operations onto operator
- Partitioning of inter-partition edges according to routes
- Creation and allocation:
  - Communication vertices **onto** communicators of the route
  - Allocation vertices **onto** memories
  - Identity vertices **onto** bus/mux/demux/ with or without arbiter

Multicomponent Implementation (2)

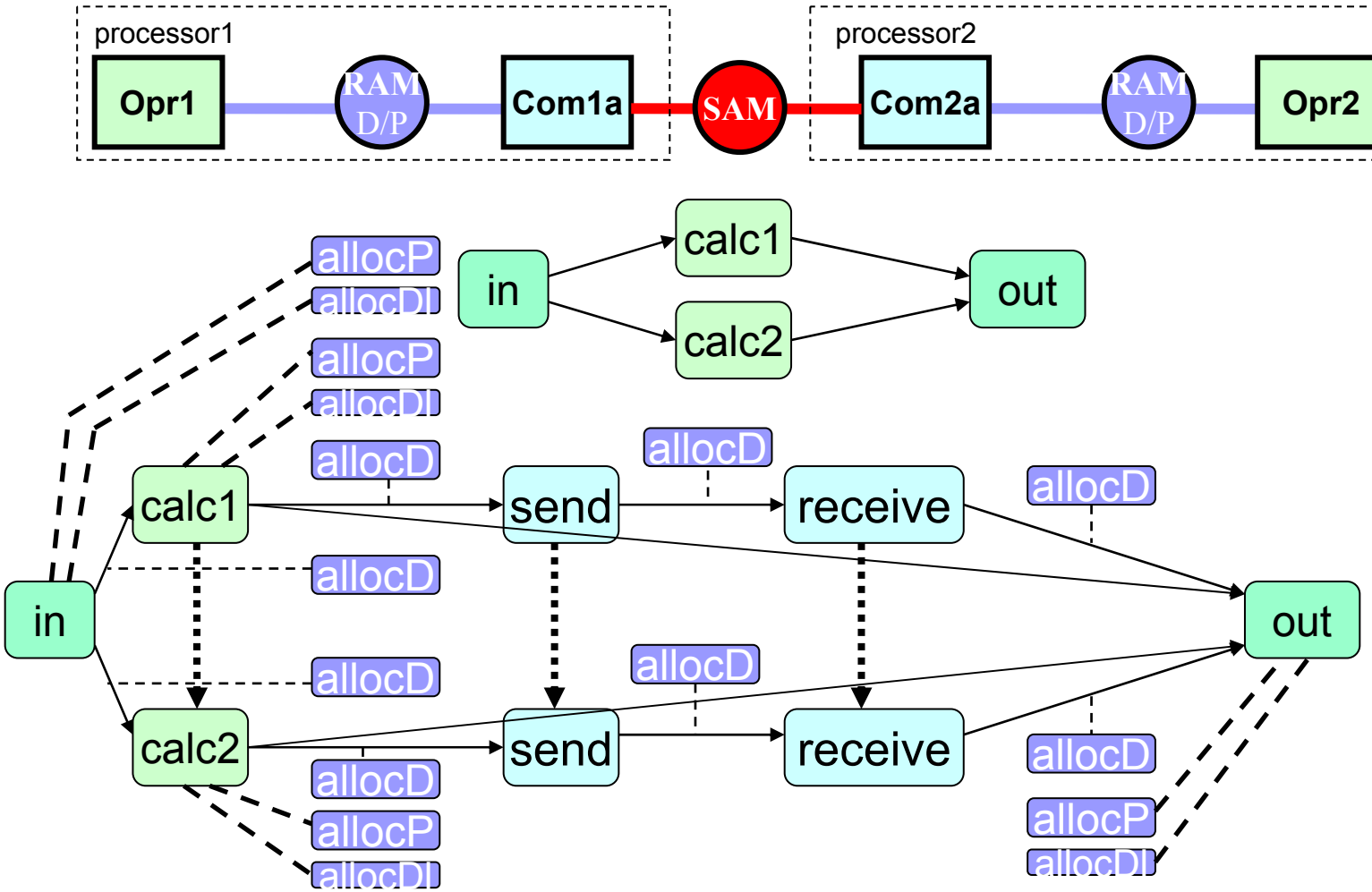
- **Scheduling**: *temporal allocation*
  - Partial Order → Total Order for:
    - Each partition of operations allocated onto an operator
    - Each partition of communication operations allocated onto a communicator

Routing, Distribution and Scheduling lead to a Partial Order consistent with the initial Partial Order of the Algorithm Graph

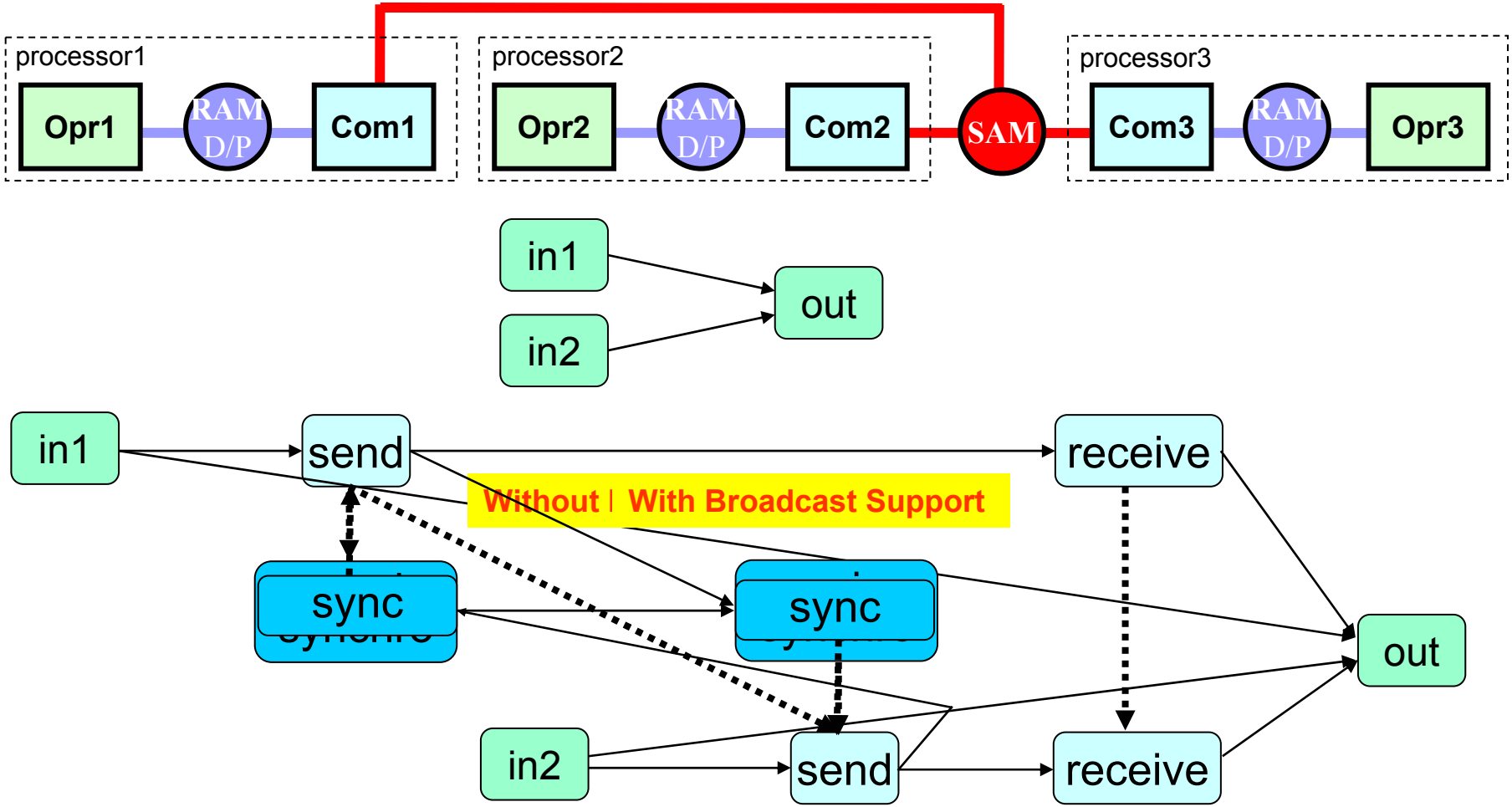
Graph transformation: External Composition Law

Implementation graph Gal' = Gal*Gar

Implementation example: Point to Point SAM



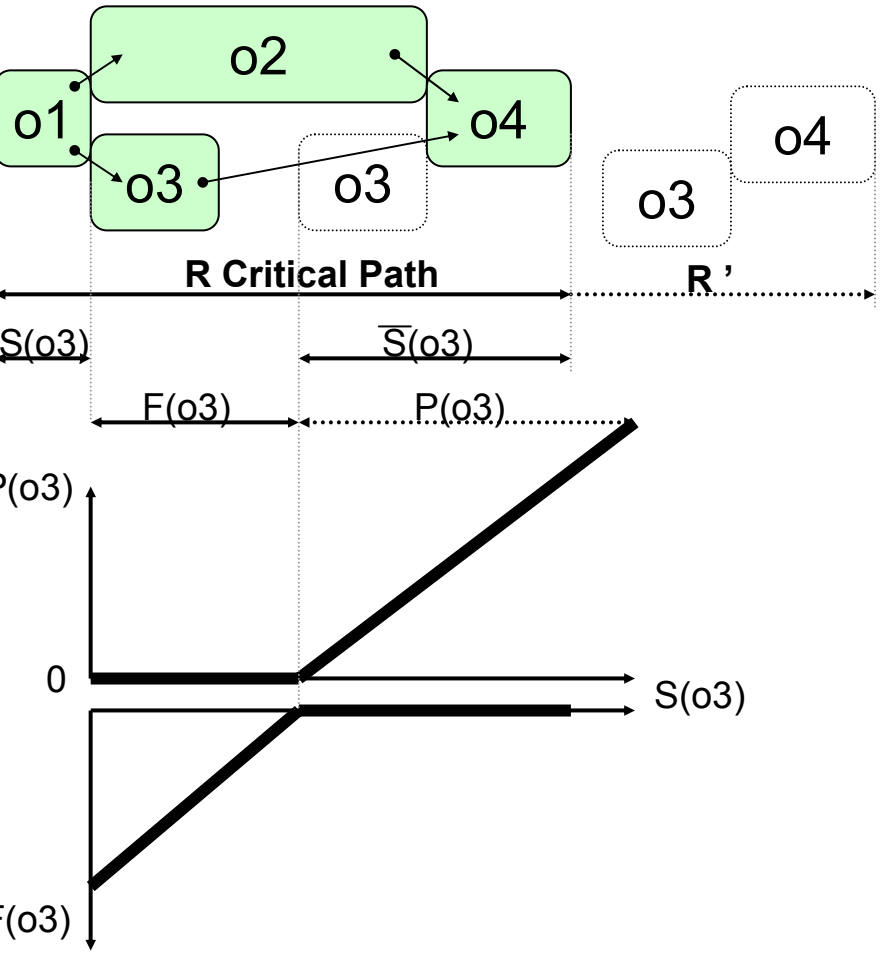
Implementation example: Multipoint SAM



Optimization Principles

- **Operations/Operators Characterization**
  - Measures: duration, memory, comp./comm. interferences
- **Choice of one implementation among all**
  - Satisfying Real-Time (latency=cadence) and Distribution Constraints
  - Minimizing resources
- **Distribution/Scheduling:** Off-Line, with or without preemption
- **NP-hard problems:** Heuristics
  - Fast: Greedy: list-scheduling, etc for Rapid Prototyping
  - Slow: Neighboring: Tabou, Simulated Anealing, etc

Optimization Example
Latency=Cadence, Off-Line without preemption



- **Earliest start from start** date:

$$S(o_i) = \max_{\forall x_j \in pred(o)} E(x_j) \text{ (ou 0 si } pred(o_i) = \varnothing \text{)}$$

- **Earliest end from start** date:

$$E(o_i) = S(o_i) + \Delta(o_i)$$

- **Latest end from end** date:

$$\overline{E}(o_i) = \max_{\forall x_j \in succ(o)} \overline{S}(x_j) \text{ (ou 0 si } succ(o_i) = \varnothing \text{)}$$

- **Latest start from end** date:

$$\overline{S}(o_i) = \overline{E}(o_i) + \Delta(o_i)$$

- **Scheduling Flexibility**:

$$F(o_i) = R - S(o_i) - \overline{S}(o_i)$$

- **Scheduling Penalty**:

$$P(o_i) = R - R'$$

- **Scheduling Pressure**:

$$\sigma(o_i) = P(o_i) - F(o_i)$$

List-Scheduling Greedy Heuristic

- Initialize the list of candidates with operations without predecessor:

$$Cand = \{o_i / pred(o_i) = \varnothing\}$$

- While the list is not empty:

  ❶ For each operation o_i of the list search the best operator (taking into account communications costs),

$$opr_{o_i} = \min_{\forall p_i \in Proc} \sigma(o_i, p_i)$$

  ❷ Select from the list the most urgent operation to schedule,

$$o_{urgent} = \max_{\forall o_i \in Cand} \sigma(o_i, opr_{o_i})$$

  ❸ Remove the operation from the list and add all its successors, which are now schedulable,

$$Cand = Cand - o_{urgent} + Succ\_ordo(o_{urgent})$$

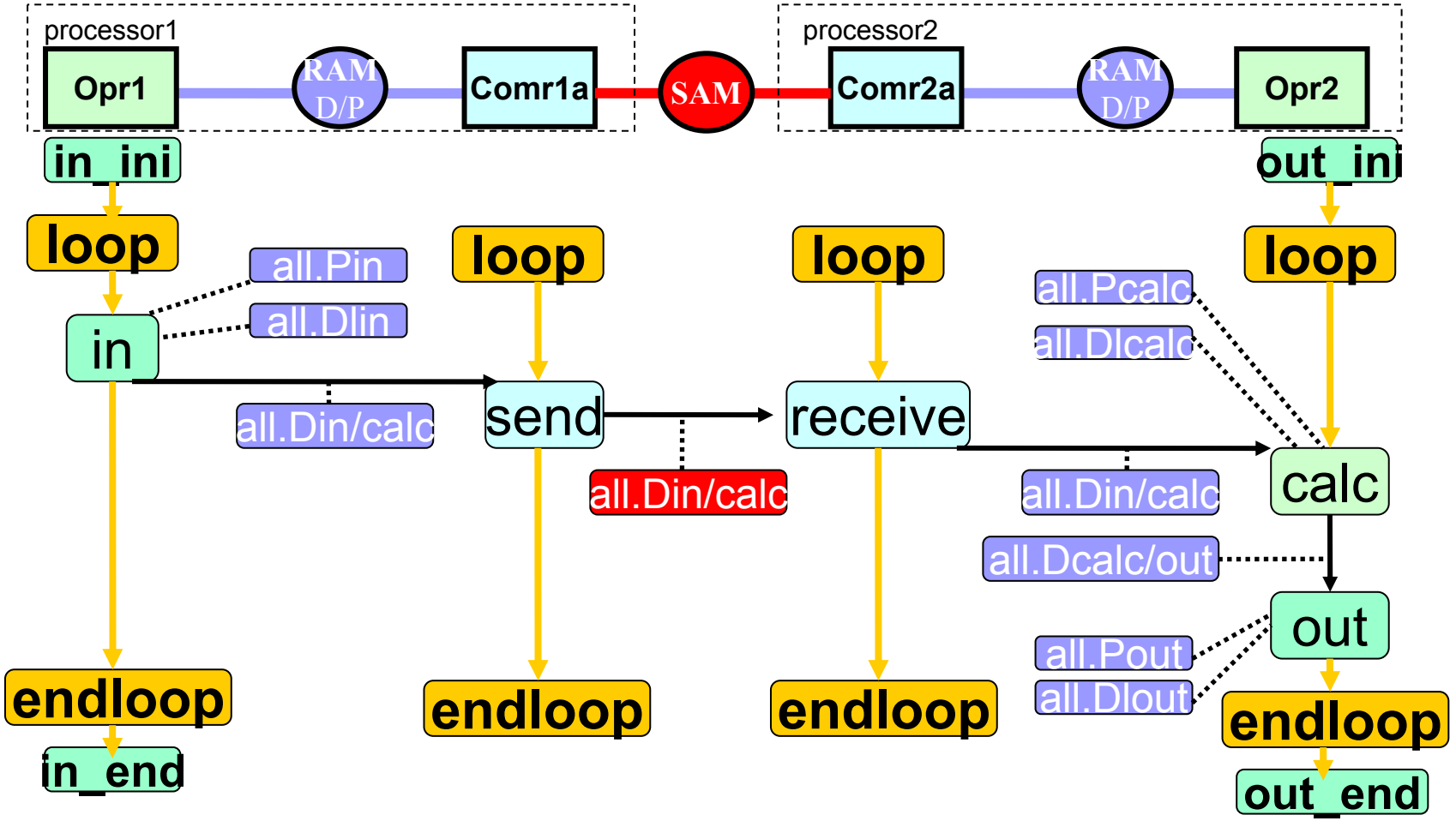
Real-time Executives Generation

- **Without Deadlock, very low Overhead**
- **Processor independent Macro-Code (m4)**
  - Extensible, easily portable (C ... asm)
- **Executive Kernel: Processor dependent Macros Definitions for:**
  - Microcontrolers: MPC555, MC68332, 80C196
  - DSP: Sharc, TMS320C40
  - Microprocessors: i80x86, PPC G4, C/Unix
  - Communication Media: links, CAN, RS232, TCP/IP

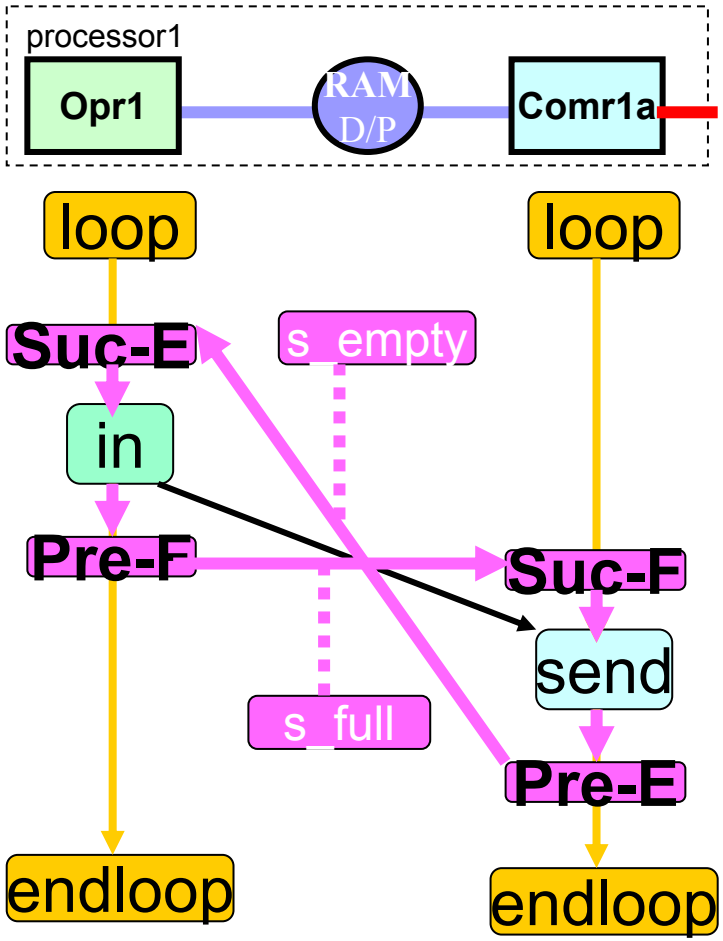
Graphs Transformations

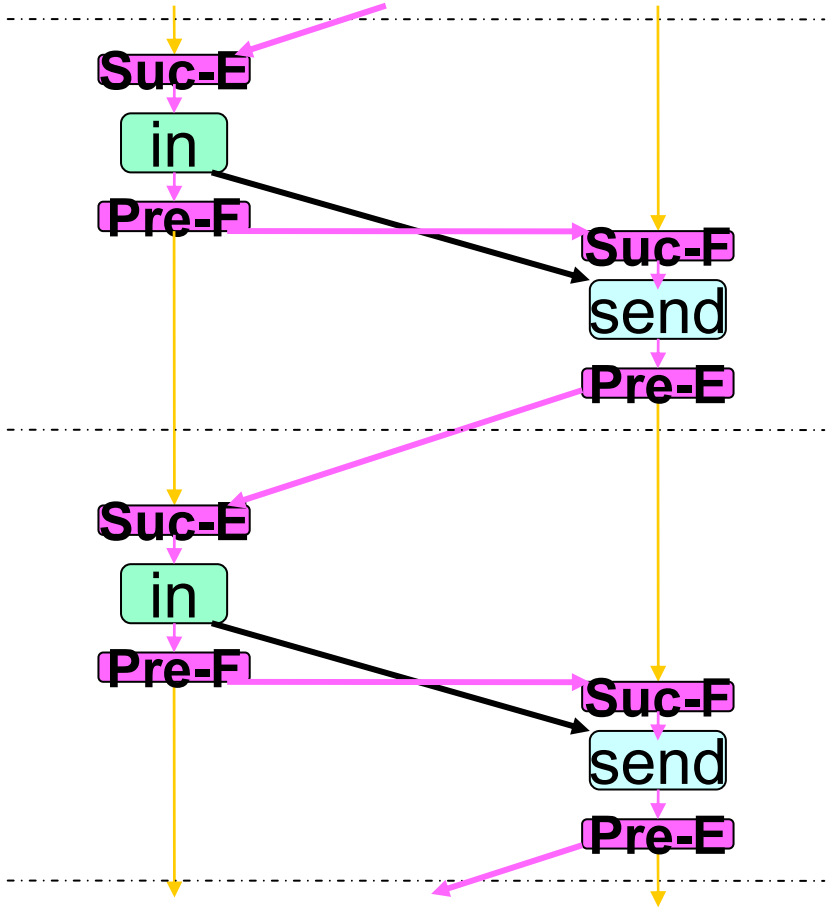
- **Optimized Implementation Graph *in* Execution Graph by adding *system vertices* :**
  - Extraction of the infinitely repeated sub-graph
  - Explicit repetition by adding ***loop/endloop vertices***
  - Adding ***init./finalisation vertices*** for inputs and outputs
  - calc/com synchronization by adding ***Pre/Suc vertices*** using semaphores
- **Execution Graph *in* macro-code**
- **Macro-code *in* source program:** macro-processor (m4) + executive libraries (macros definitions)

Execution Graph: explicit repetition

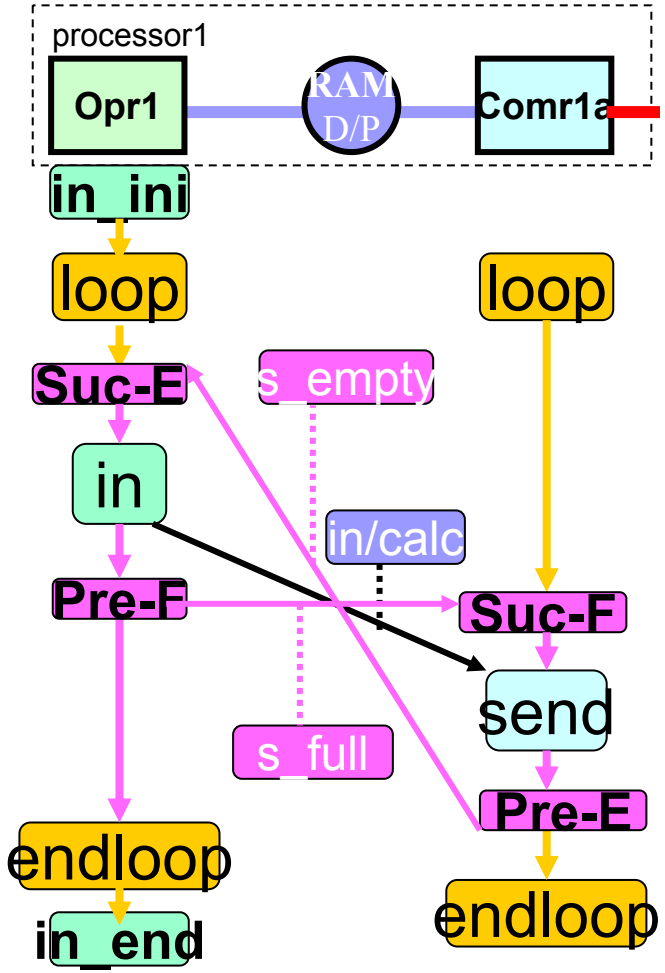


Execution Graph: synchronizations





Macro-code Generation



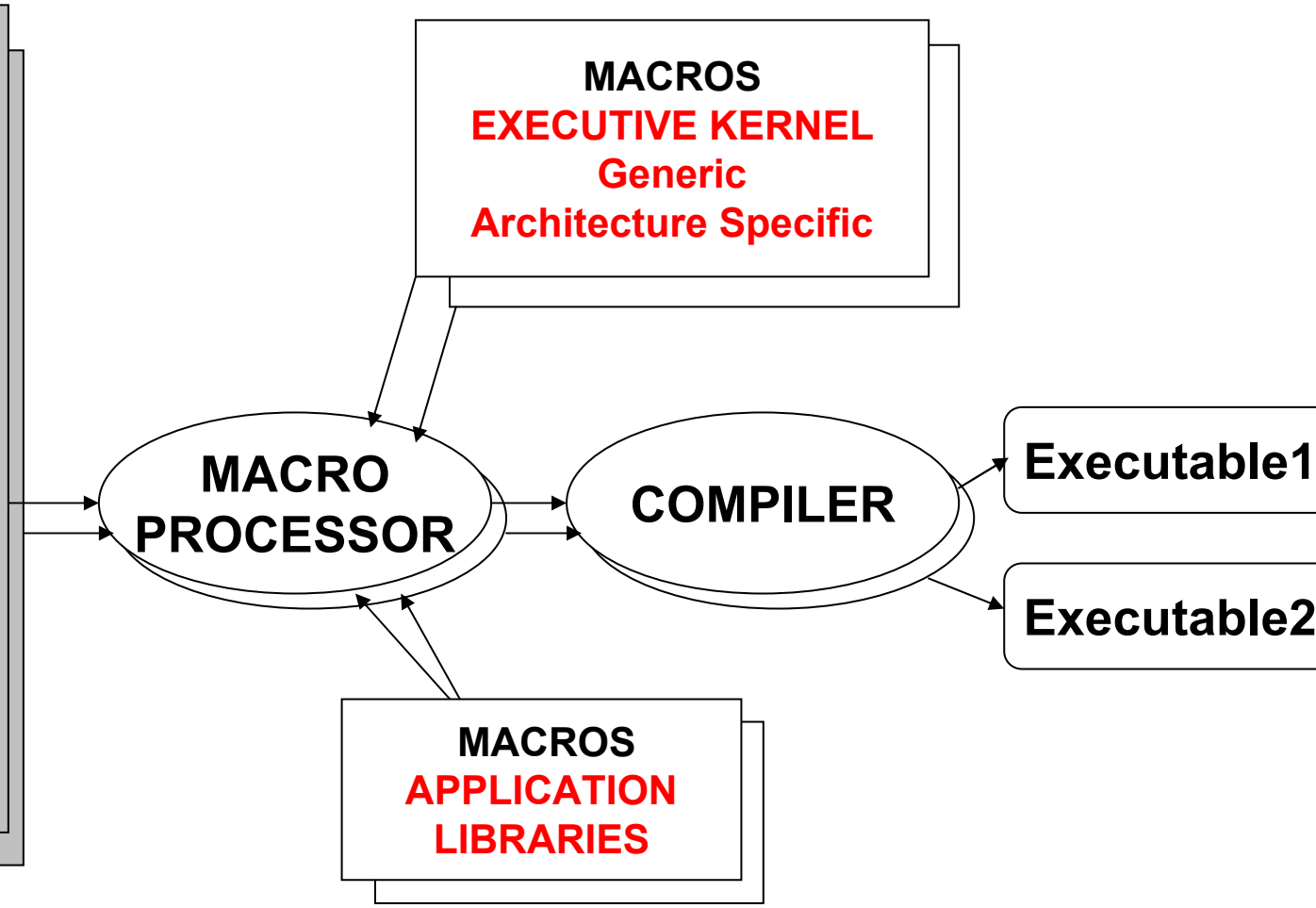

```
processor_( opr1, .)
  semaphores_(s_empty…)
  alloc_(type_in/calc...)
  thread_(comr1a)
    pre0 (s_empty)
      loop_
        sucF(s_full)
        send(in/calc)
        preE(s_empty)
      endloop_
  endthread_
  main_
      in_ini( )
      spawn_thread(com1ra)
      loop_
        sucE(s_empty)
        in(in/calc)
        preF(s_full)
      endloop_
      in_end ( )
      endmain_
endprocessor_
```

Macro-code Compilation

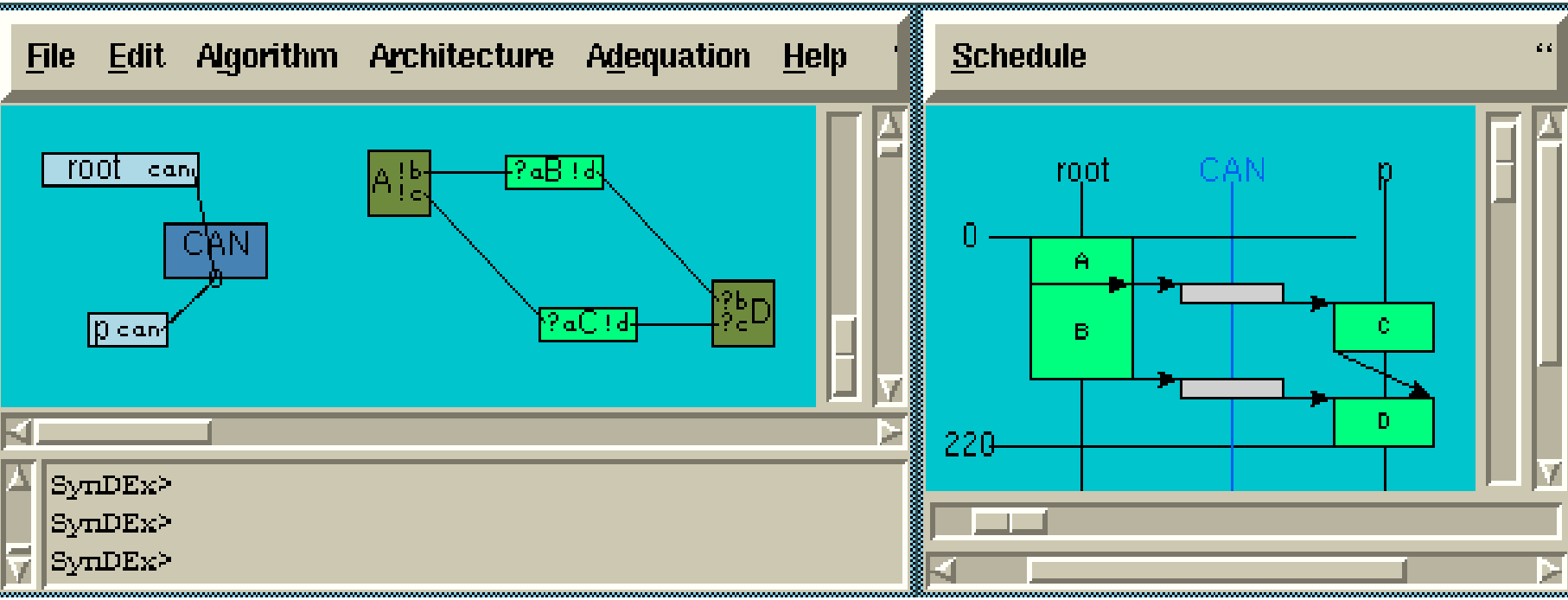
```
processor_( opr1, .)

semaphores_(s_empty…)
  alloc_(type_in/calc...)
  thread_(comr1a)
    pre0 (s_empty)
      loop_
        sucF(s_full)
        send(in/calc)
        preE(s_empty)
      endloop_
  endthread_
  main_
     in_ini( )

spawn_thread(com1ra)
      loop_
        sucE(s_empty)
        in(in/calc)
        preF(s_full)
      endloop_
      in_end ( )
      endmain_
endprocessor_
```



Distibuted Real-Time Executive Generation: simple application

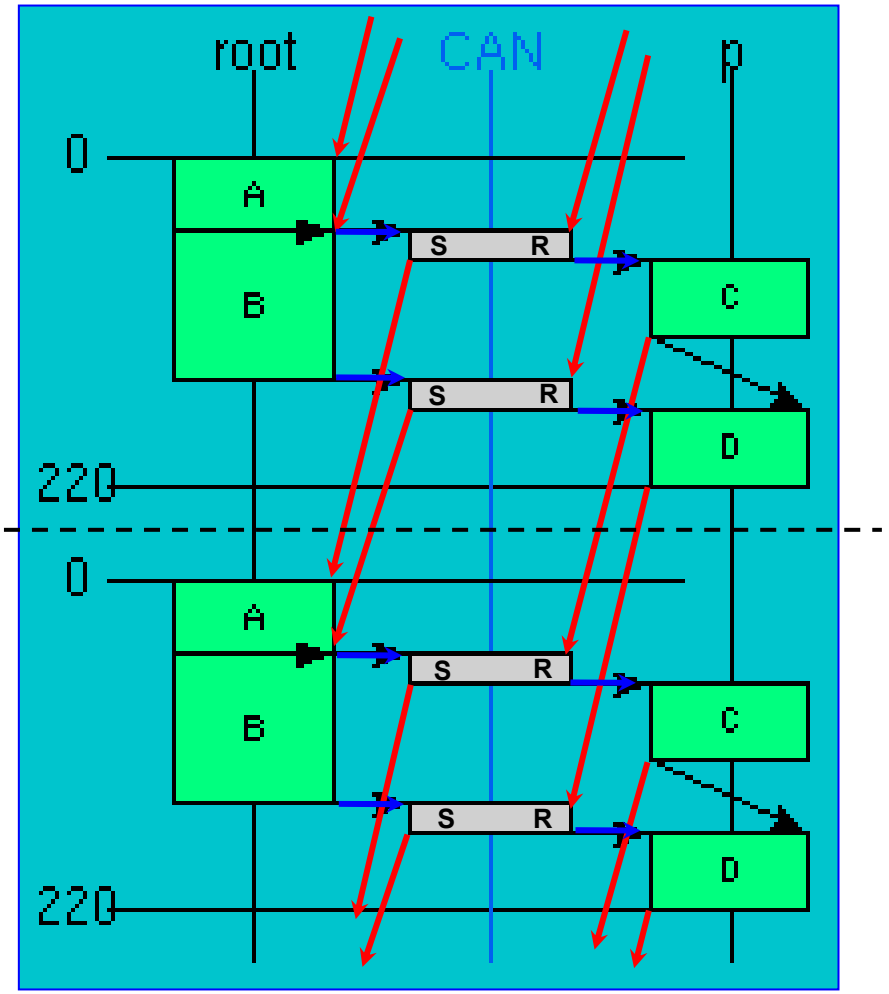


Executive Generation : Synchro calc/com

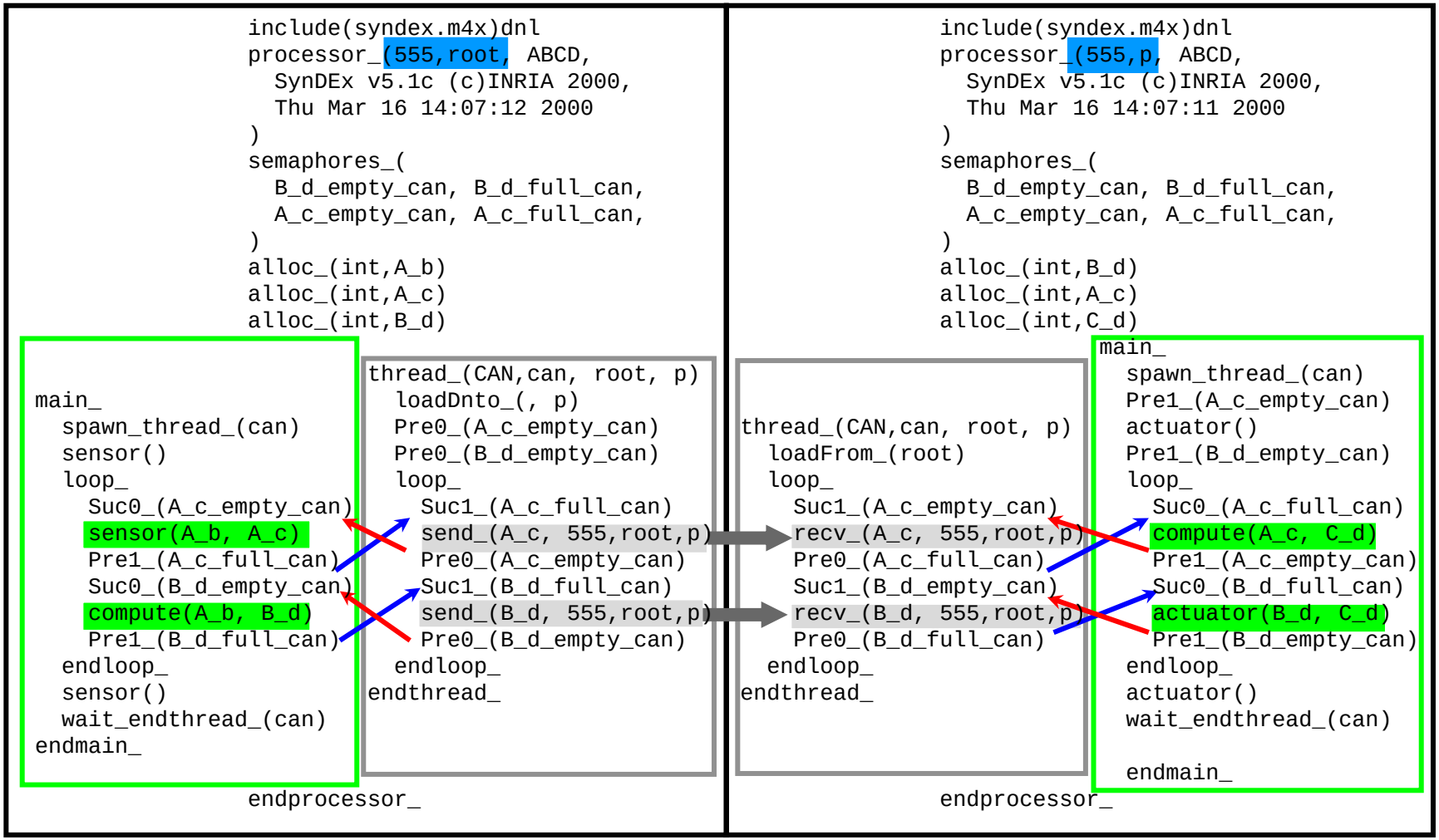
Buffer

Intra-repetition Synchro

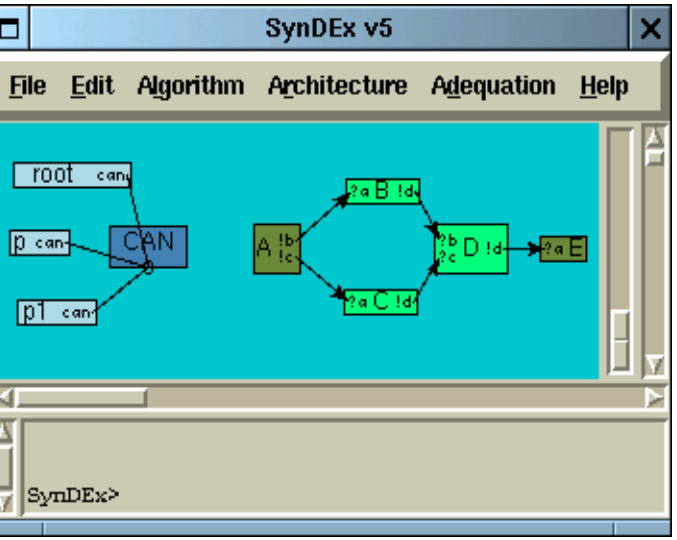
Inter-repetition Synchro

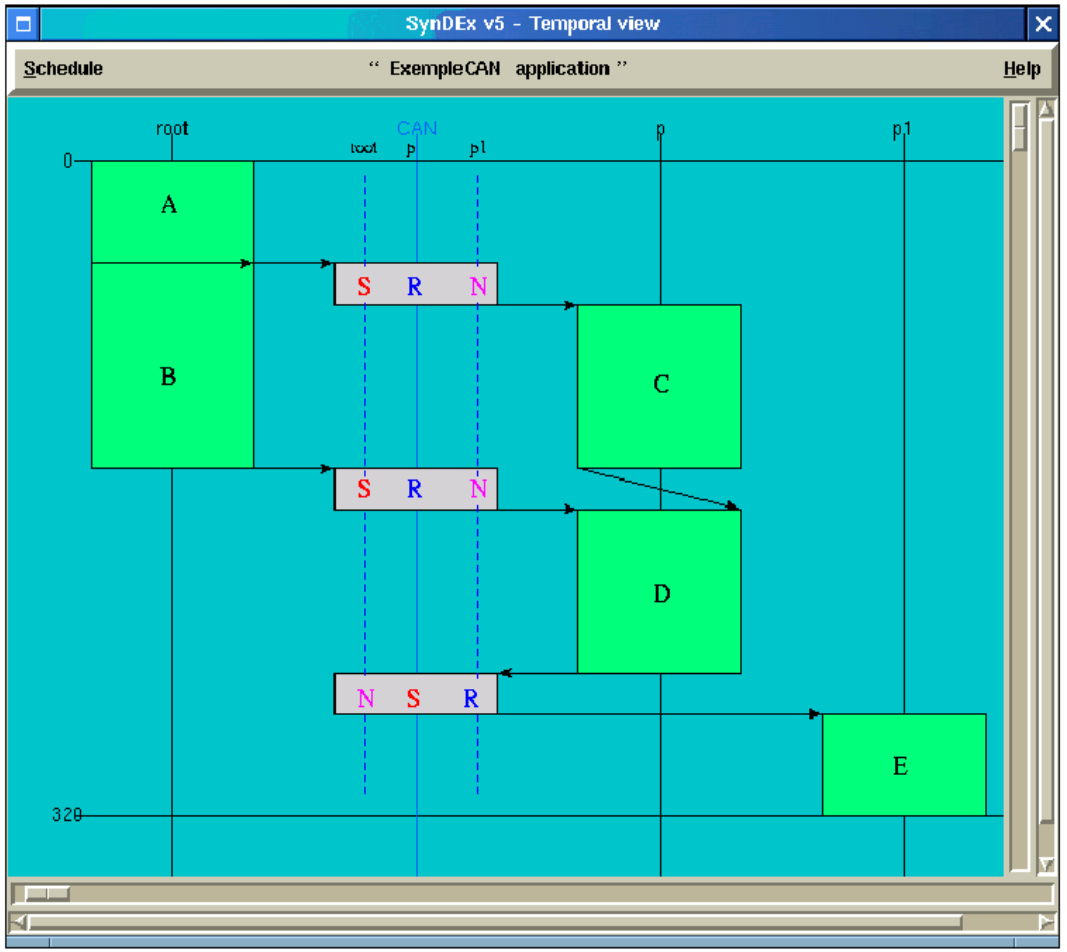


Executive Generation: Code



Executive Generation: Synchro. comm.



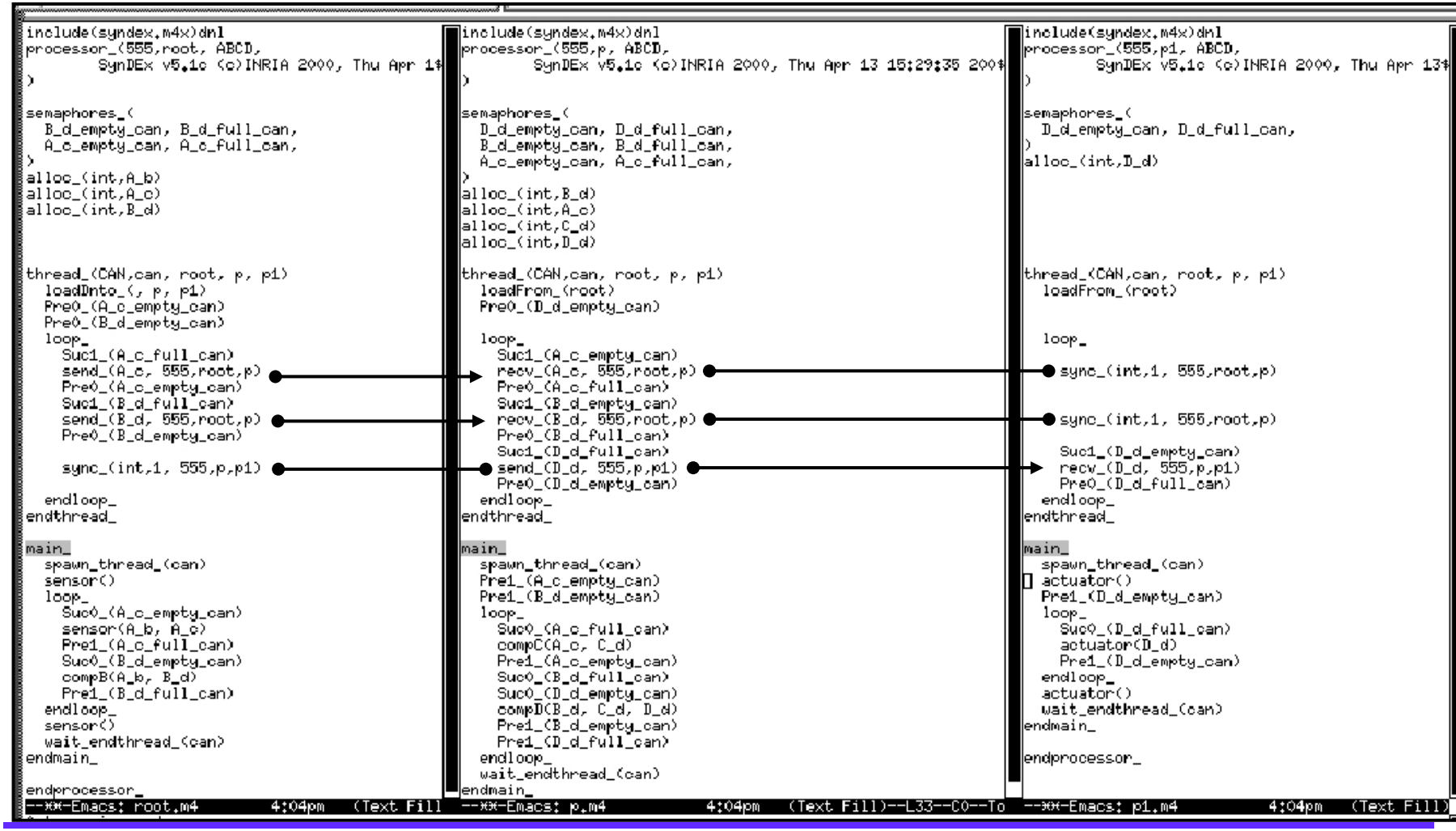


S : Send

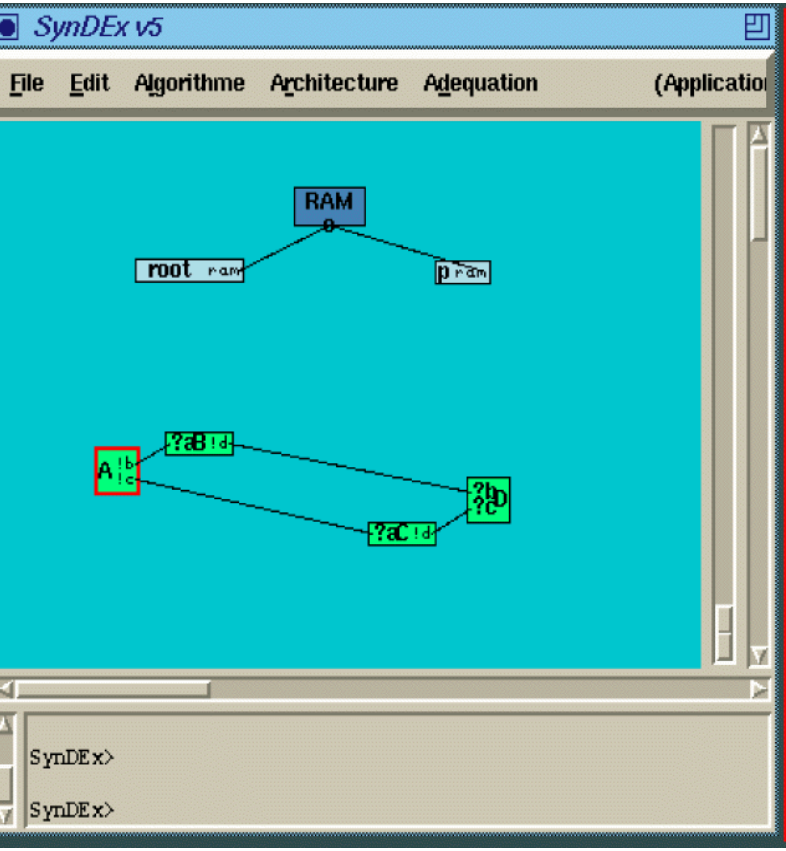
R :Receive

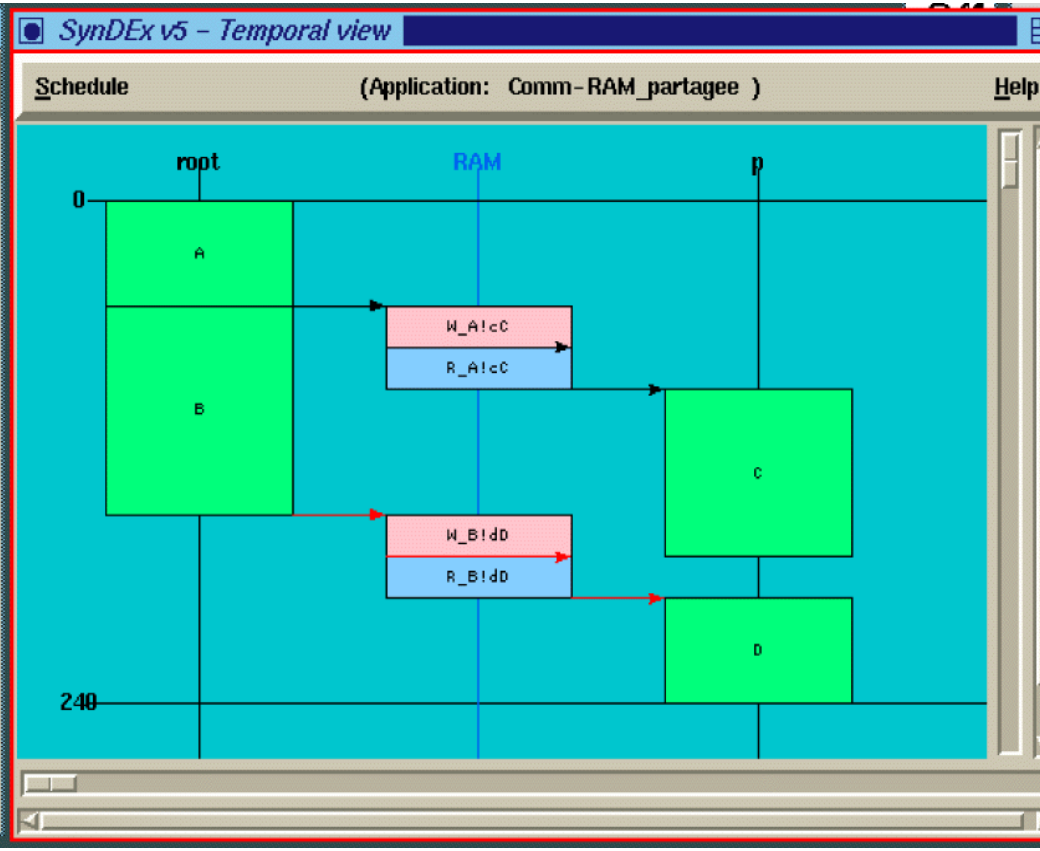
N : Sync

Executive Generation: Code

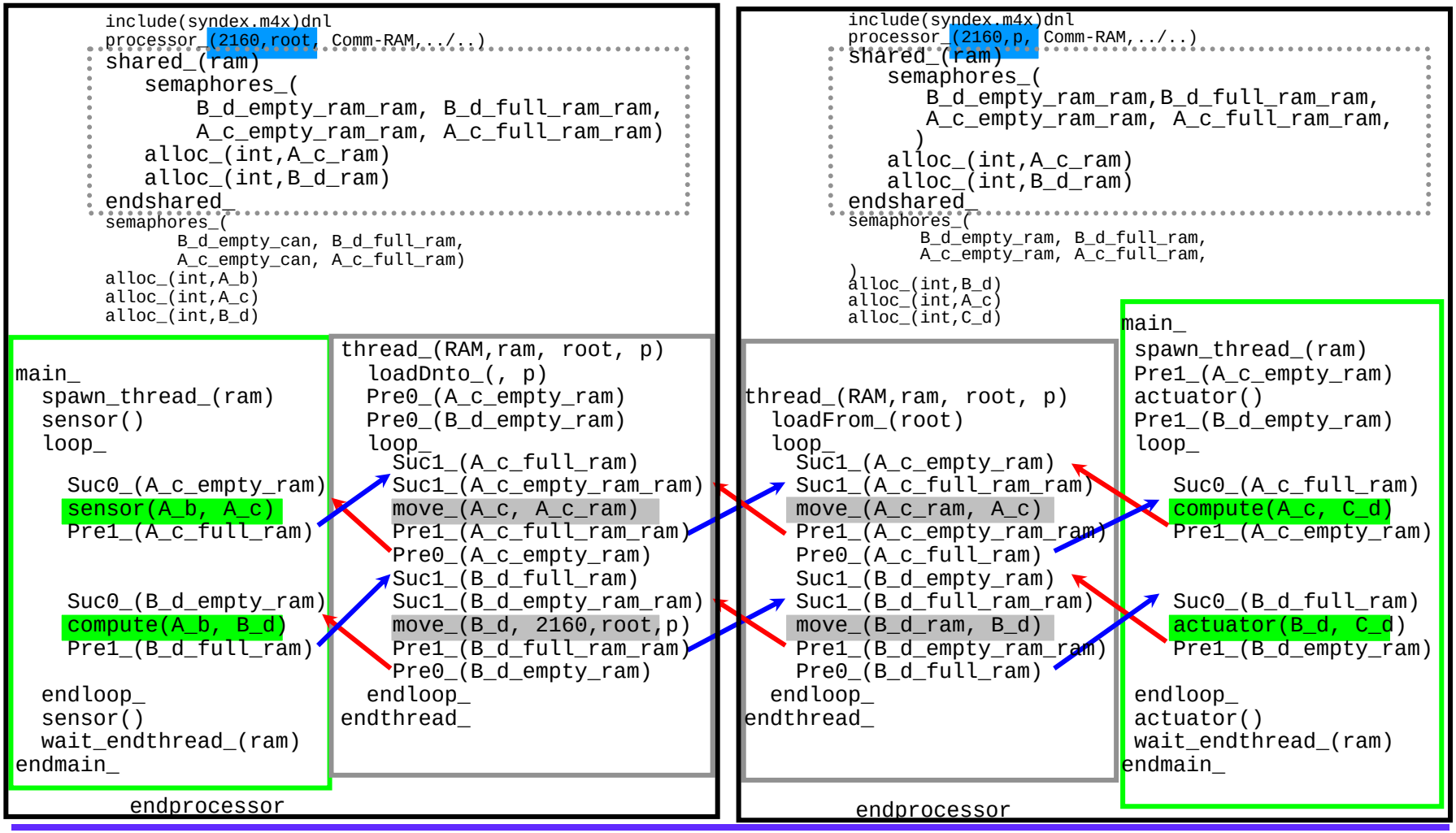


Executive Generation: Shared Mem. Comm.





Executive Generation: Shared Mem. Comm.



System Level CAD Software: SynDEx

- **AAA Methodology Support**
- **Interface with High-Level Languages**
- **Graphic Interactive Unix X11 or Windows**
- **Algorithm and Architecture Graphs Edition**
- **Off-Line Distribution and Scheduling**
- **Real-Time Predicted Diagrams Visualization**
- **Real-Time Performances Measure**
- **Real-Time Distributed Executives Generation**

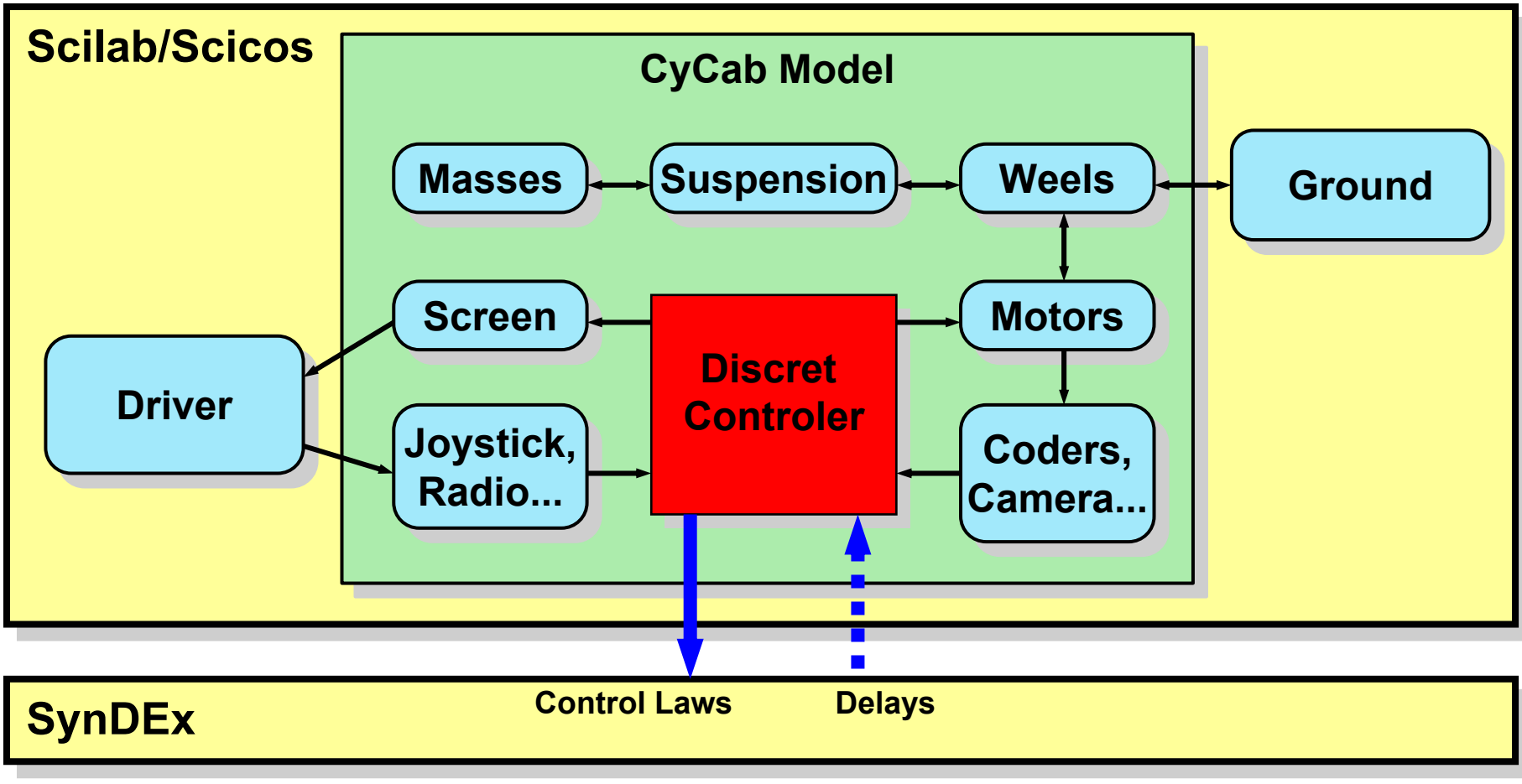
CyCab Example

- Length 1,9m
- Witdh 1,2m
- Weight 350kg
- Speed 30km/h
- Electric Motors 4x900w
- Autonomy 2h
- Reload by Induction

Industrialized by Robosoft

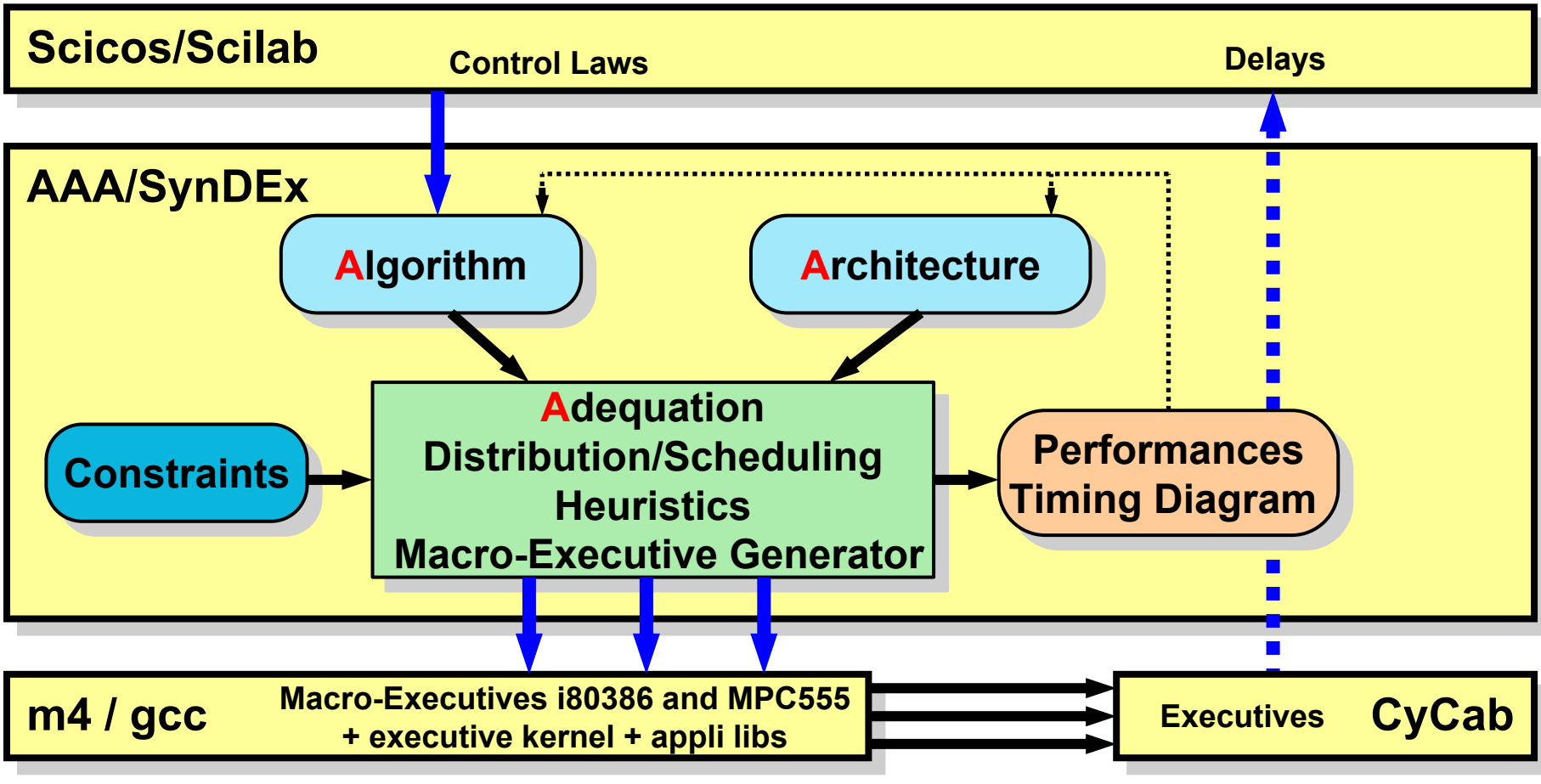
http://www.robosoft.fr

Modelling/Simulation
Scilab/Scicos Free at: www.scilab.org

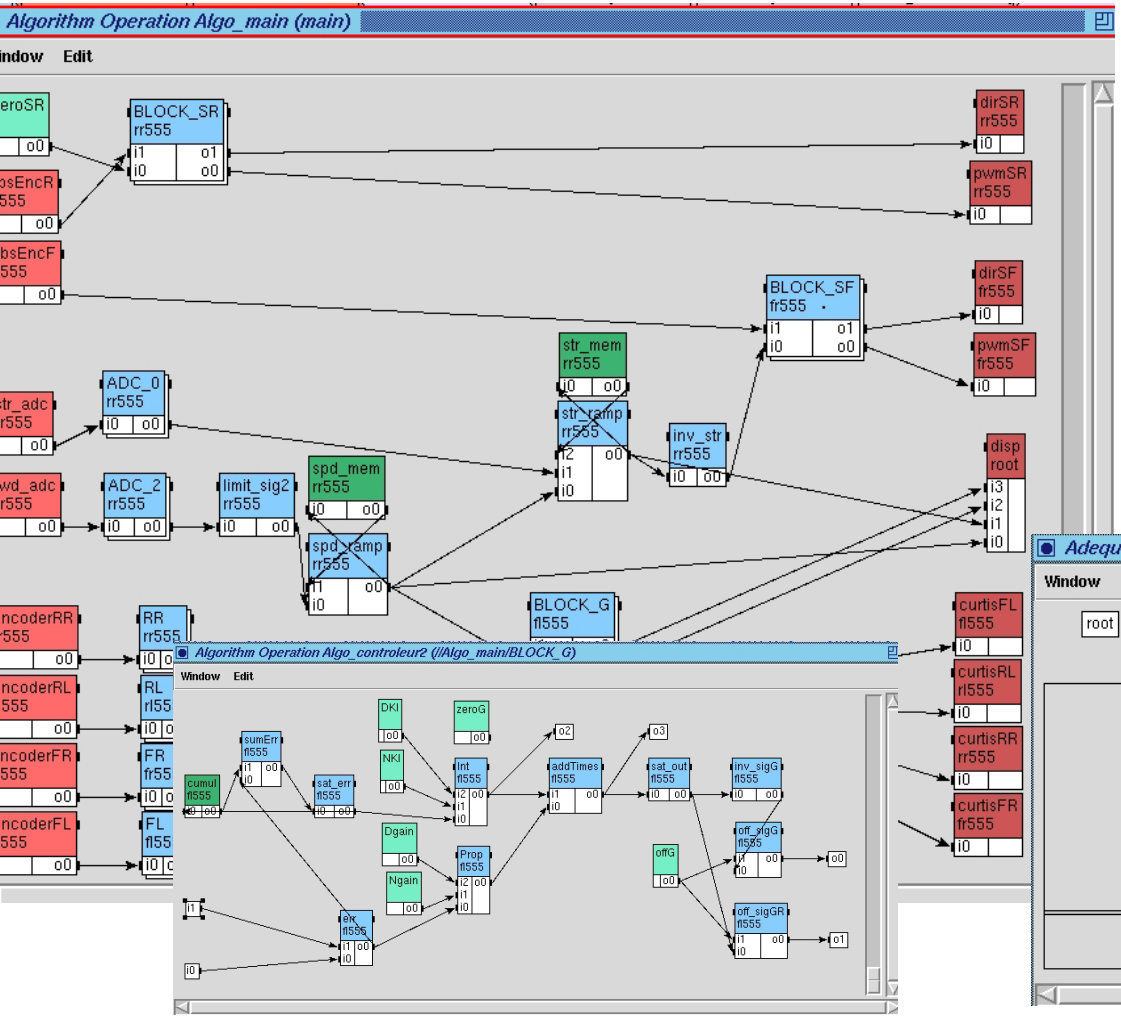


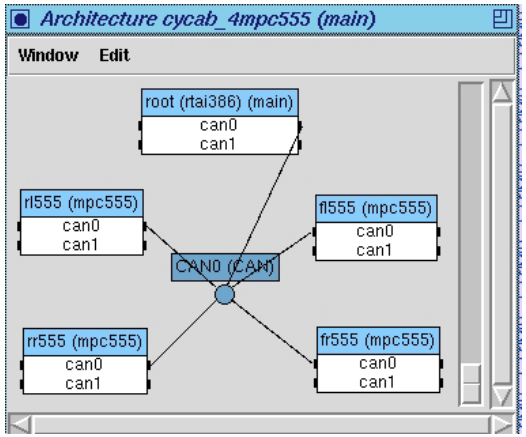
Implementation AAA/SynDEx

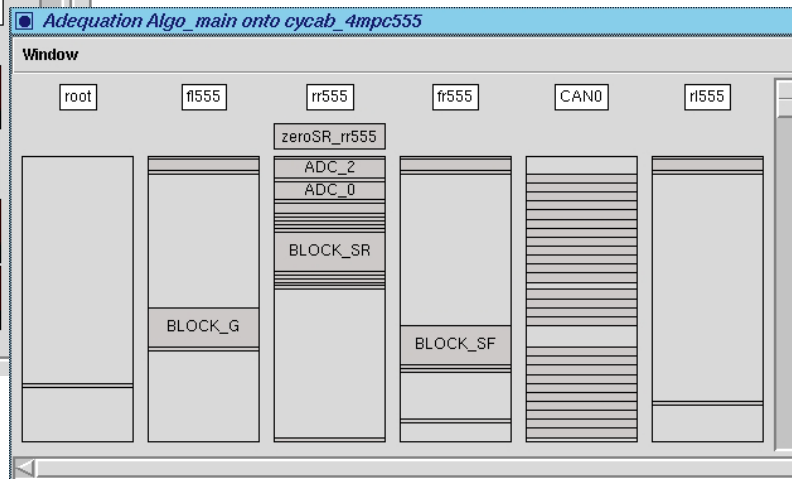
Free: www.syndex.org



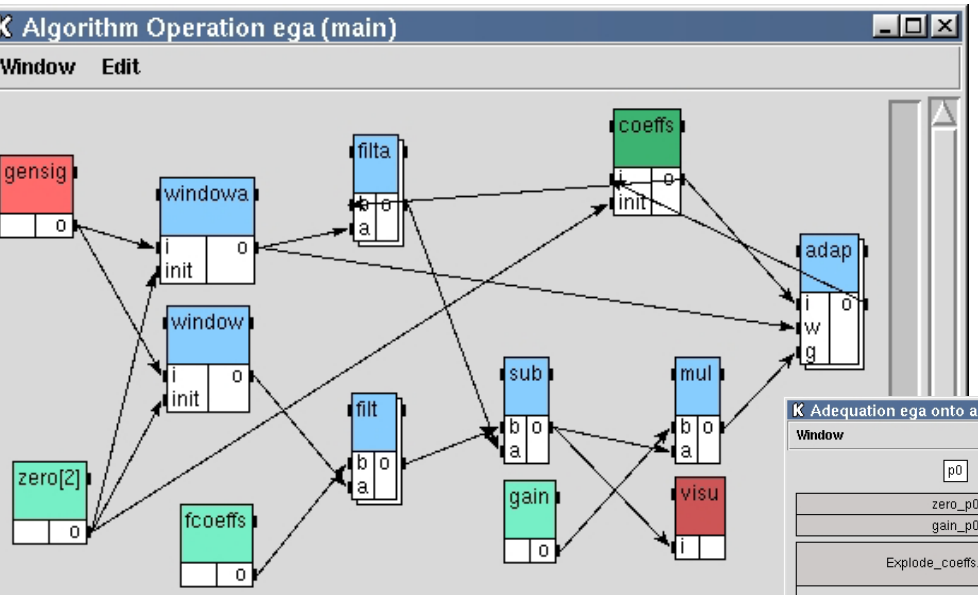
Manual Driving Application for the CyCab

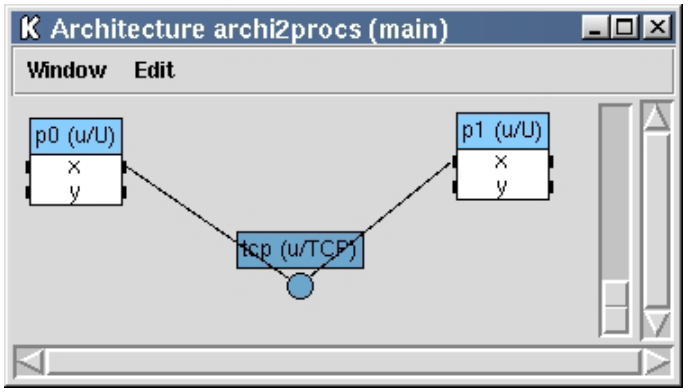


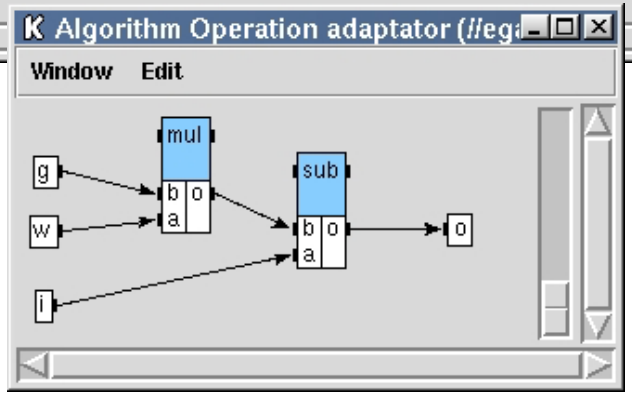


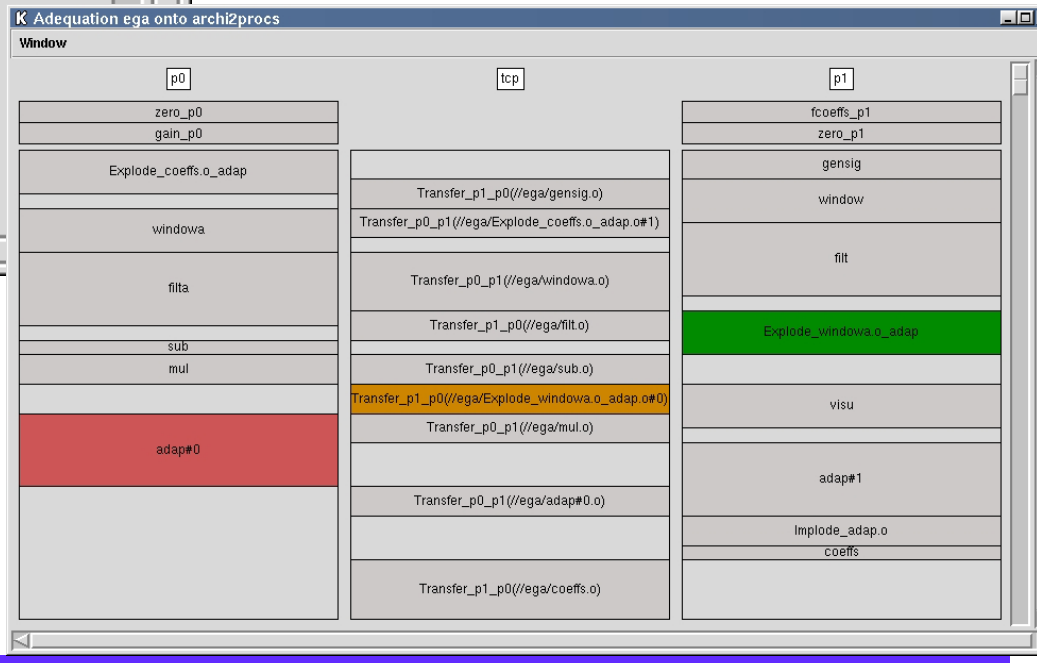


Adaptive Equalizer Application

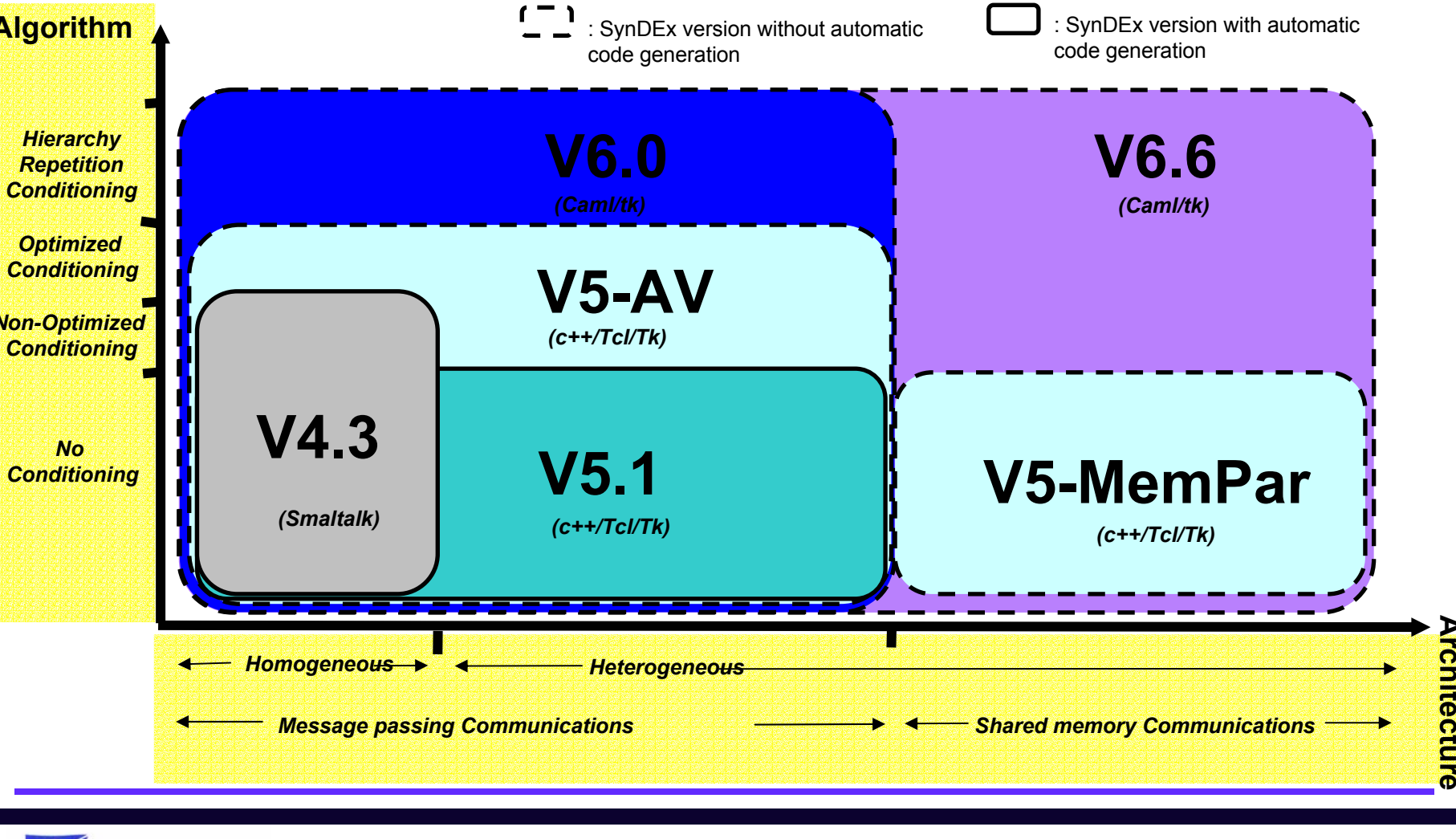








SynDEx: Version Evolution



Integrated Circuit Implementation

- **Algorithm Graph transformed in Implementation Graph**
  - Factorized Vertex = VHDL component repeatedly executed on different data
  - Factorized Hyperedge = VHDL Signal
  - Component/signals controled by counters/mux/registers
  - Automatic Synthesis of Data and Control Paths
- **Optimization**
  - Surface/(latency=cadence) compromise
  - Defactorization: data parallelism, pipe-line, retiming
  - Until Real-Time constraints are satisfied

Conclusion

- **Development Cycle is Dramatically Reduced**
  - Safe Design
  - Rapid Prototyping and Optimised Implementation
  - Dead-Lock free Executives, Debug only Executive Kernel
- **Future works**
  - Control-Flow/Data-Flow integration
  - Multi-Real-Time Constraints: EAST, P2I
    - Off-Line with or without preemption
    - In-Line for aperiodic events
  - GALS: Glob. Async. Loc. Sync. systems
  - Co-Design: Processor/IC Unification
  - Fault Tolerance with BIP team