# Specification verification and optimization of distributed embedded real-time 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**
- **3500 persons**
- **6 centers**
- **Budget: 135 MEuro**
- **20 Startup companies**

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

# Five research themes

1. **Communication systems**
2. **Cognitive systems**
3. **Symbolic systems**
4. **Numerical systems**
5. **Biological systems**

# INRIA Context

- **AOSTE project-team**
- **Theme Com C: Embedded systems and mobility**
- **Models and methods of Analysis and Optimization for Systems with real-Time and Embedding constraints**
- **Implementation onto embedded platforms:**
  **Algorithm-Architecture Adequation Methodology for the optimized implementation of distributed embedded real-time applications (SynDEx software)**

# 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

# Off-line versus on-line approaches

**AAA Methodology**

Off-line → Off-line

(as much as possible) (when unavoidable)

**Classic design**

On-line → Off-line

**Static :** distrib./sched. off-line without preemtion

**Dynamic :** distrib./ sched. in-line or off-line with preemption

*Advantages*
- Off-line : deterministic, low over-head
- On-line : data dependent durations

*Drawbacks*
- Off-line : not allways possible, knowledge of environment necessary
- On-line : high over-head, soft real-time

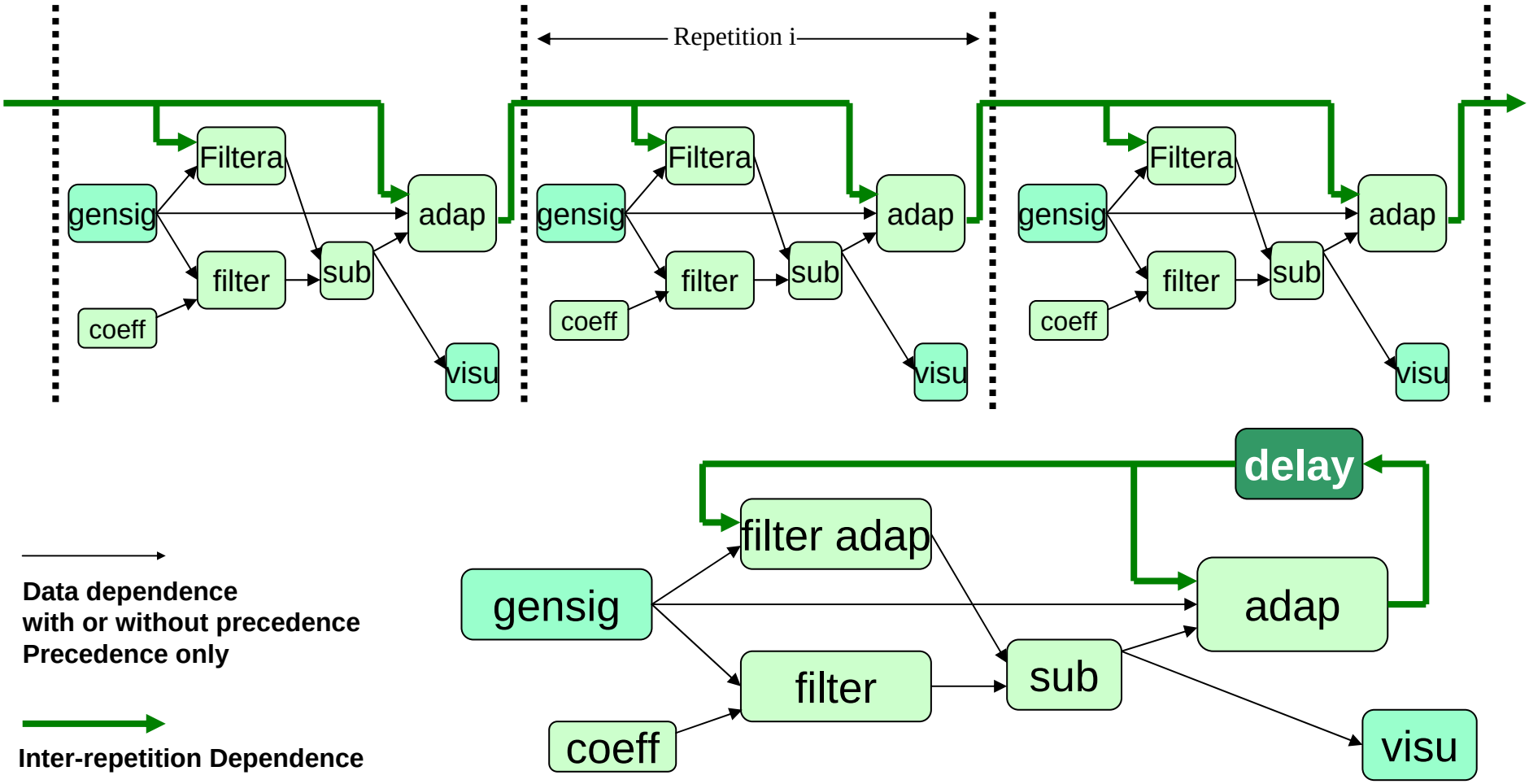
# Algorithm specification: Directed hyper-graph

- **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**

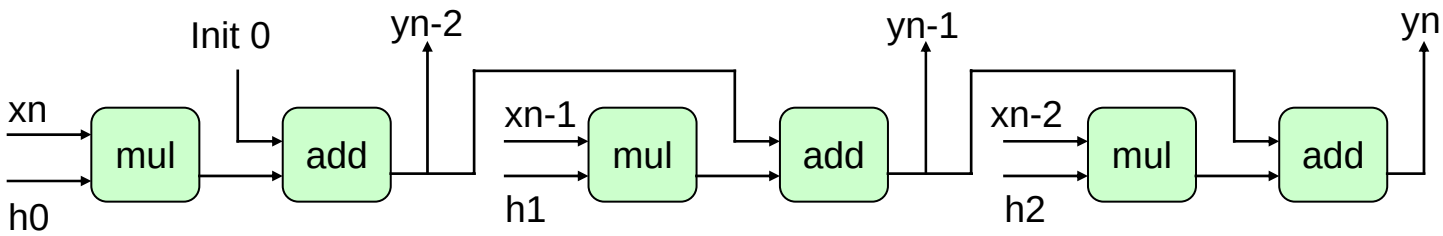
- **May be obtained from compilers of: Synchronous Languages, AIL, Scicos, AVS, CamlFlow,...**

# Algorithm example: Adaptative equalizer



# 3 coefficients digital filter: Finite repetition of 3

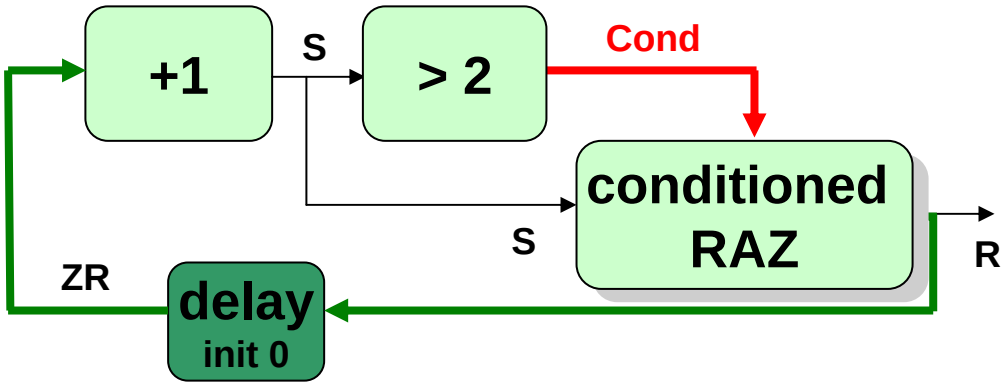
$$Yn = \Sigma(hi*Xn-i) \text{ for } i=0 \text{ to } 2$$

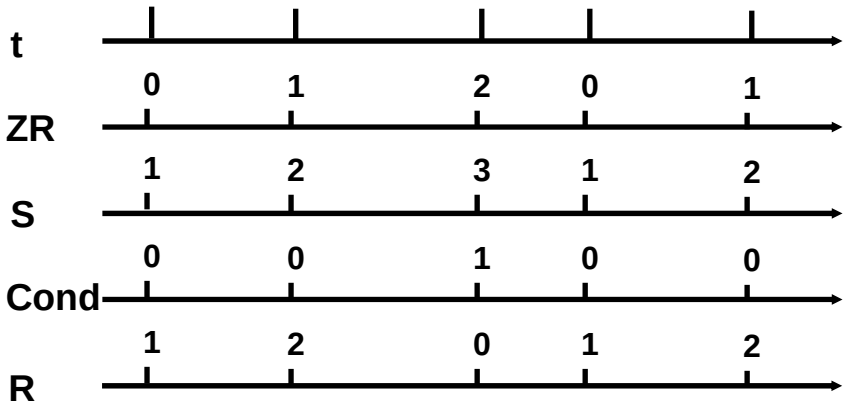


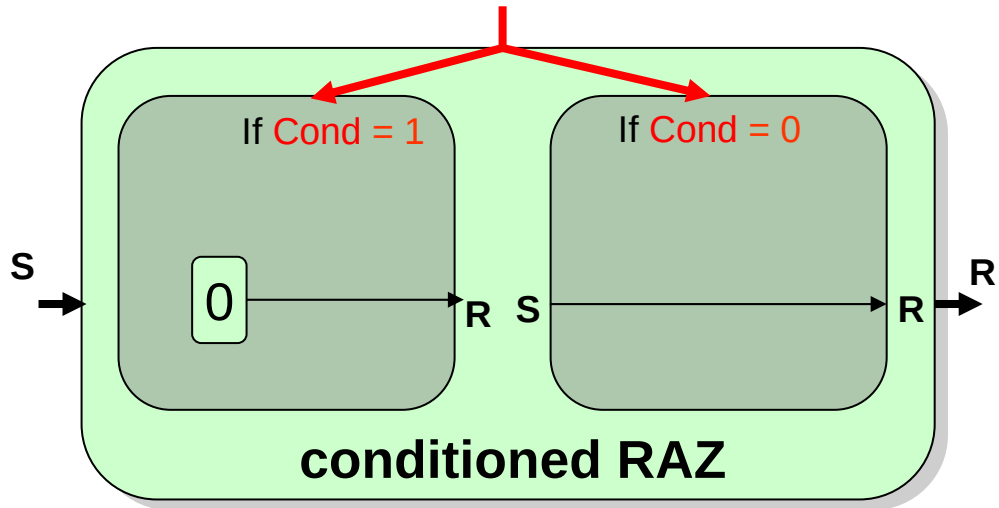

Fork : H(h0, h1, h2) et X(xn, xn-1, xn-2)

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

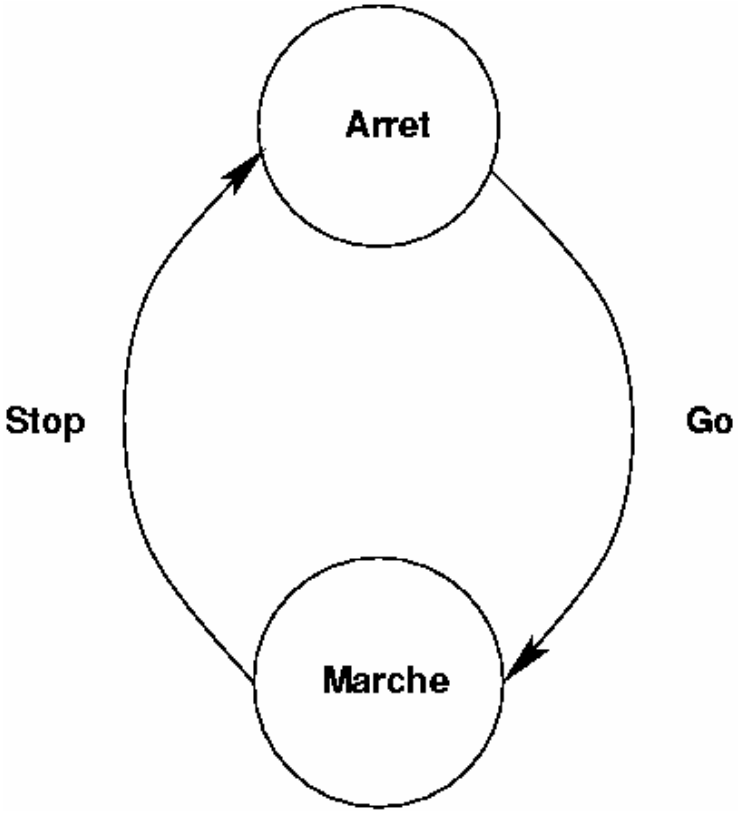
# Conditioned Algorithm: Modulo 3 Counter

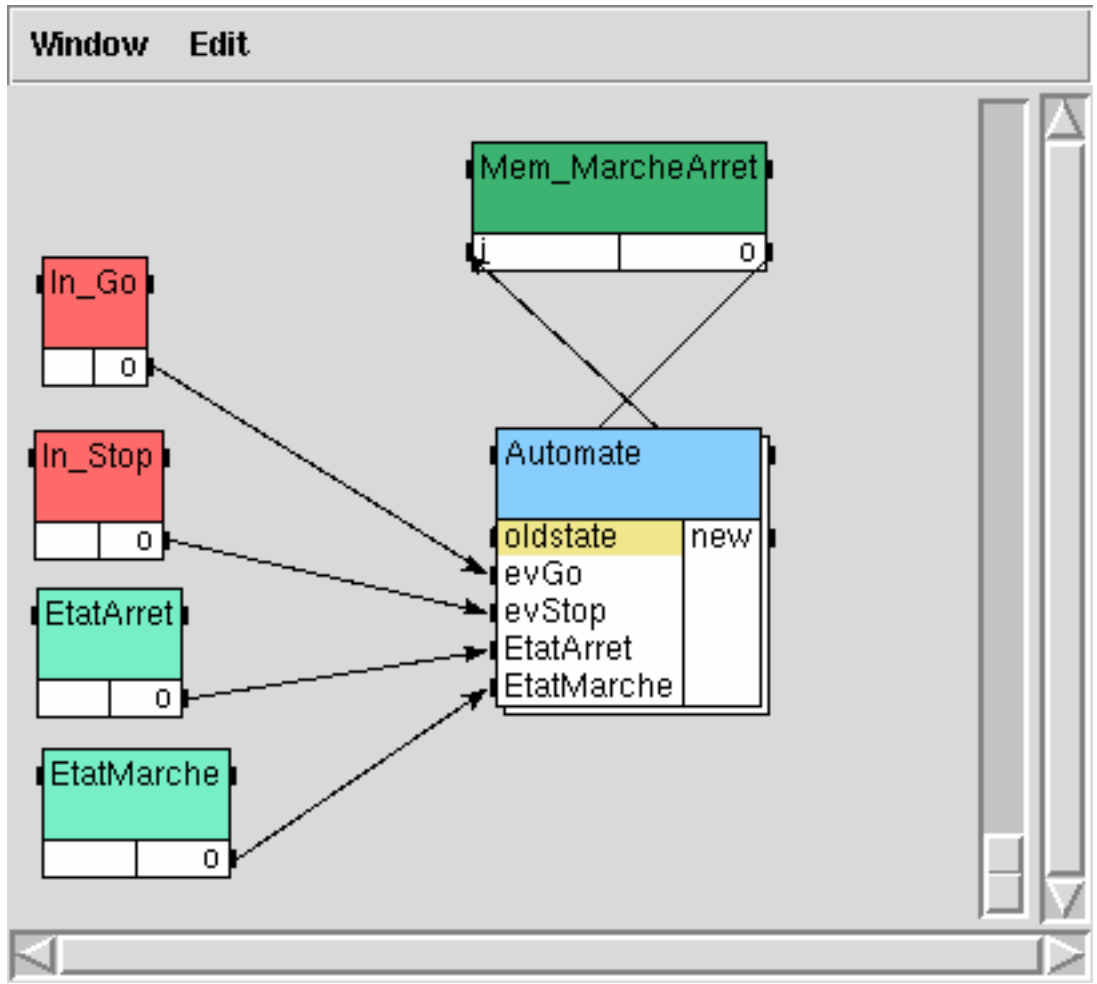


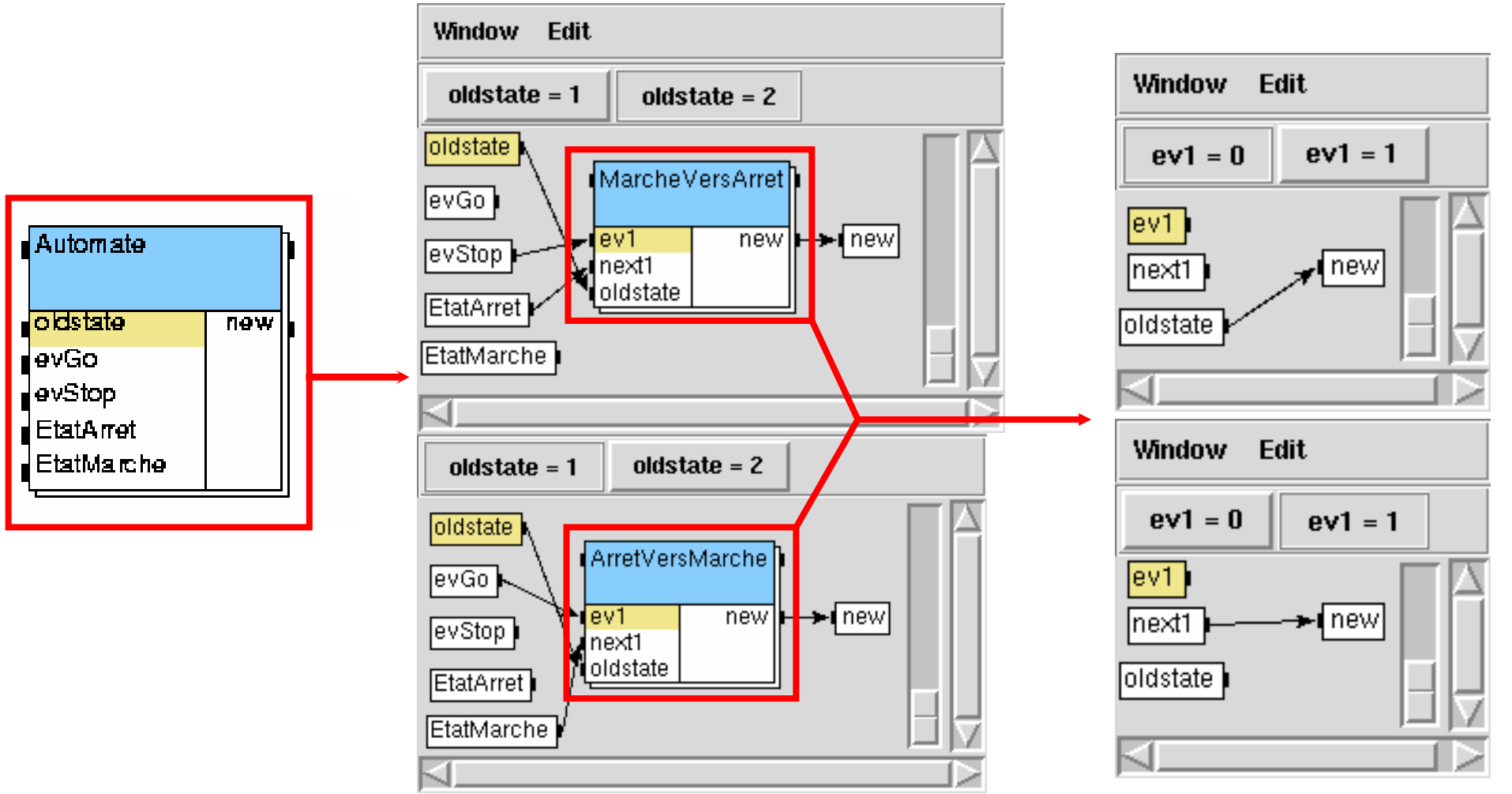



# Finite state machine with AAA/SynDEx (1/2)

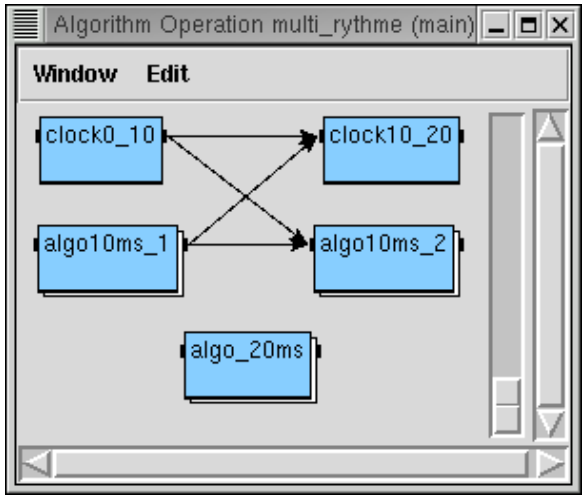


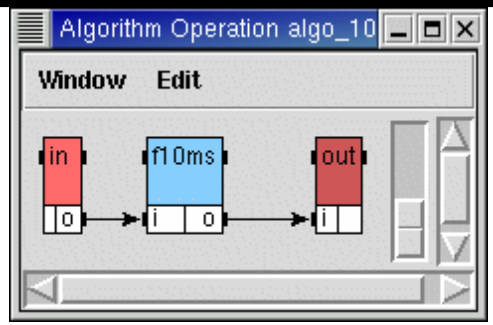



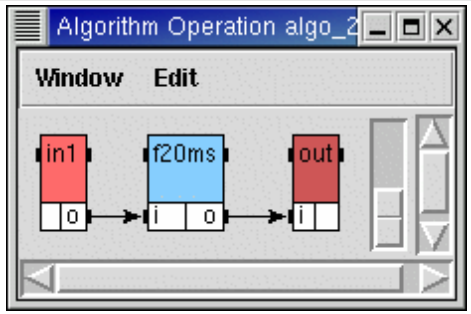
# Finite state machine with AAA/SynDEx (2/2)

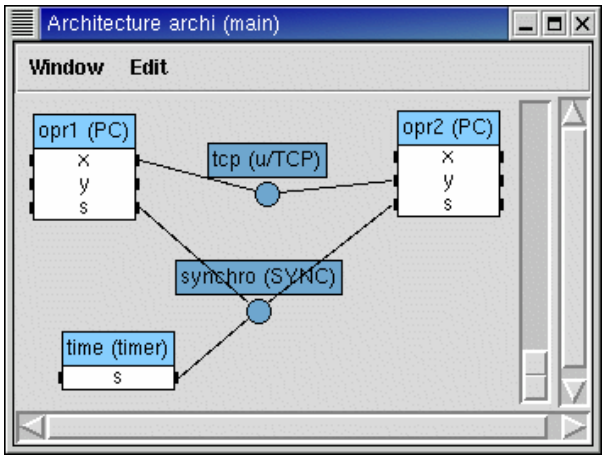


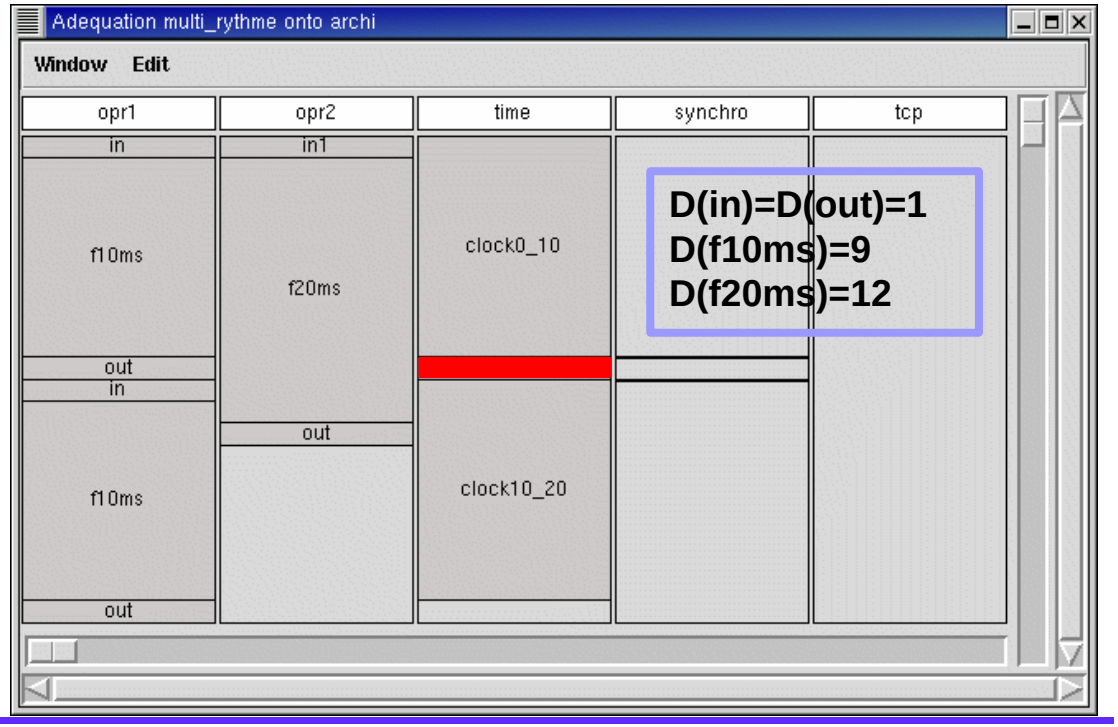
# Multi-period



# 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

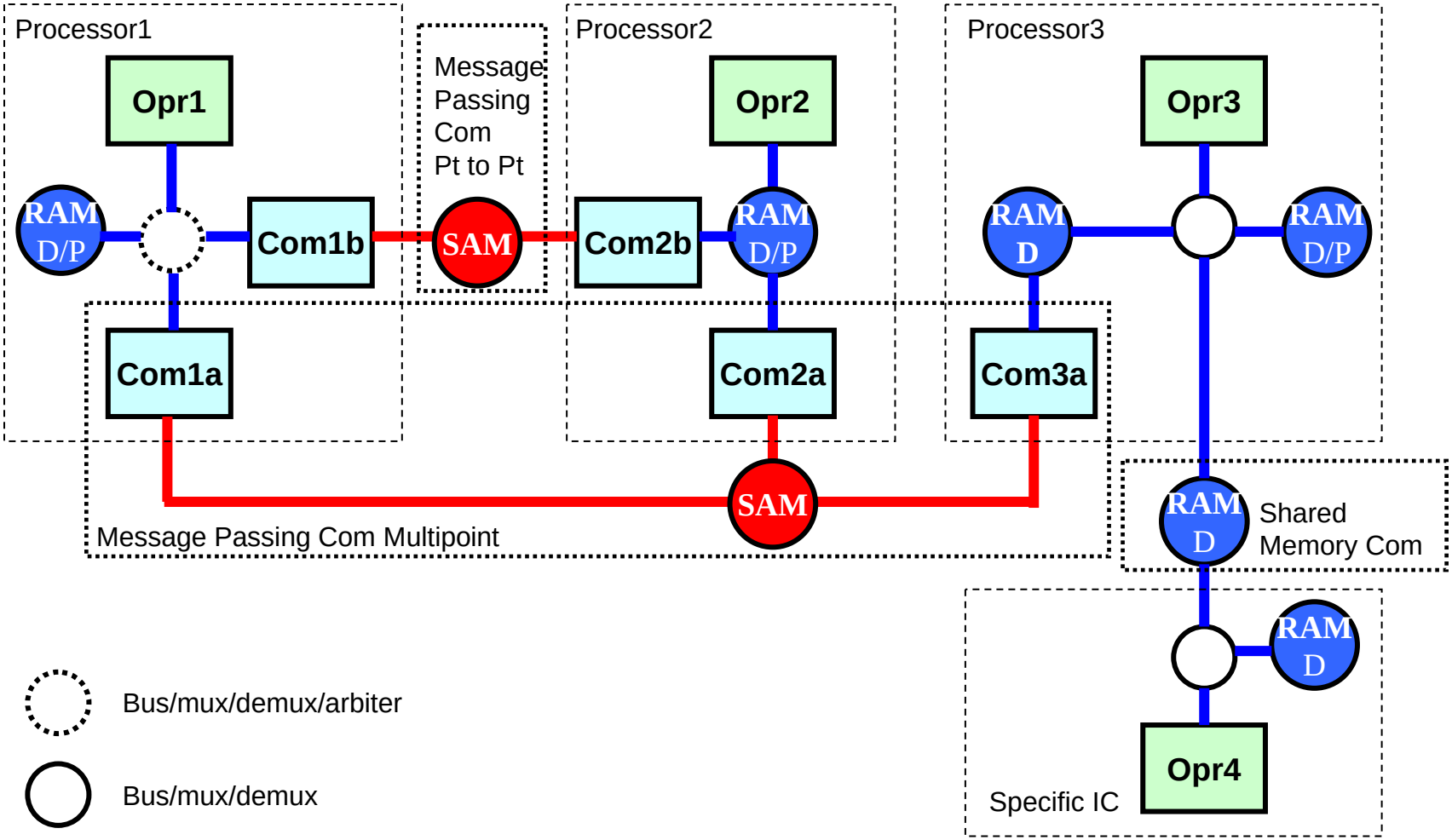
# Architecture specification: Directed graph (1)

- **Vertex (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:**


- **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)

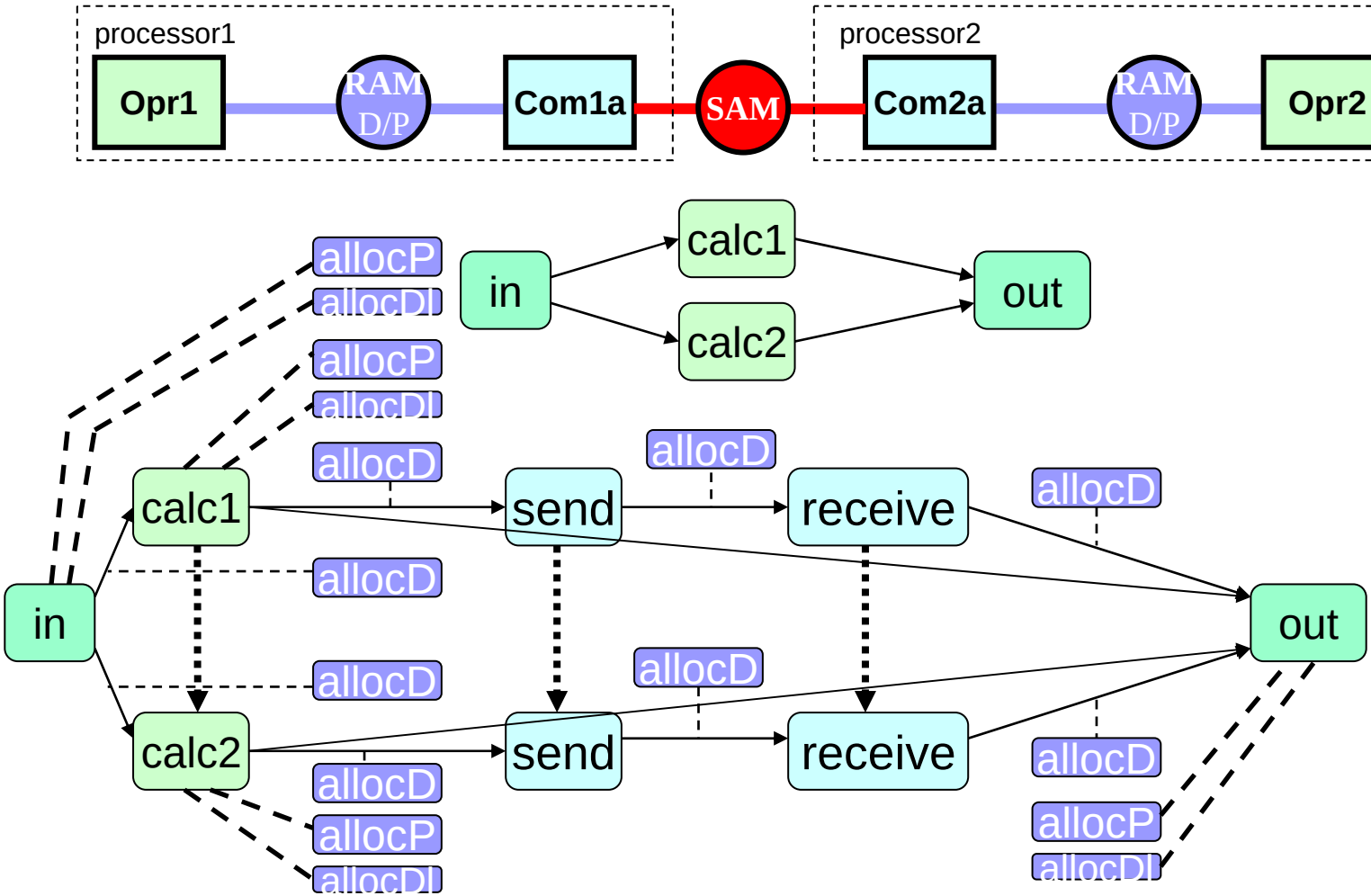
- **Scheduling**: *temporal allocation*
  - Partial Order → Total Order for:
    - Each partition of operations allocated onto an operator
    - Each partition of communication operations allocated onto a communicator

**Routing, distribution and scheduling lead to a partial order consistent with the initial partial order of the algorithm graph**

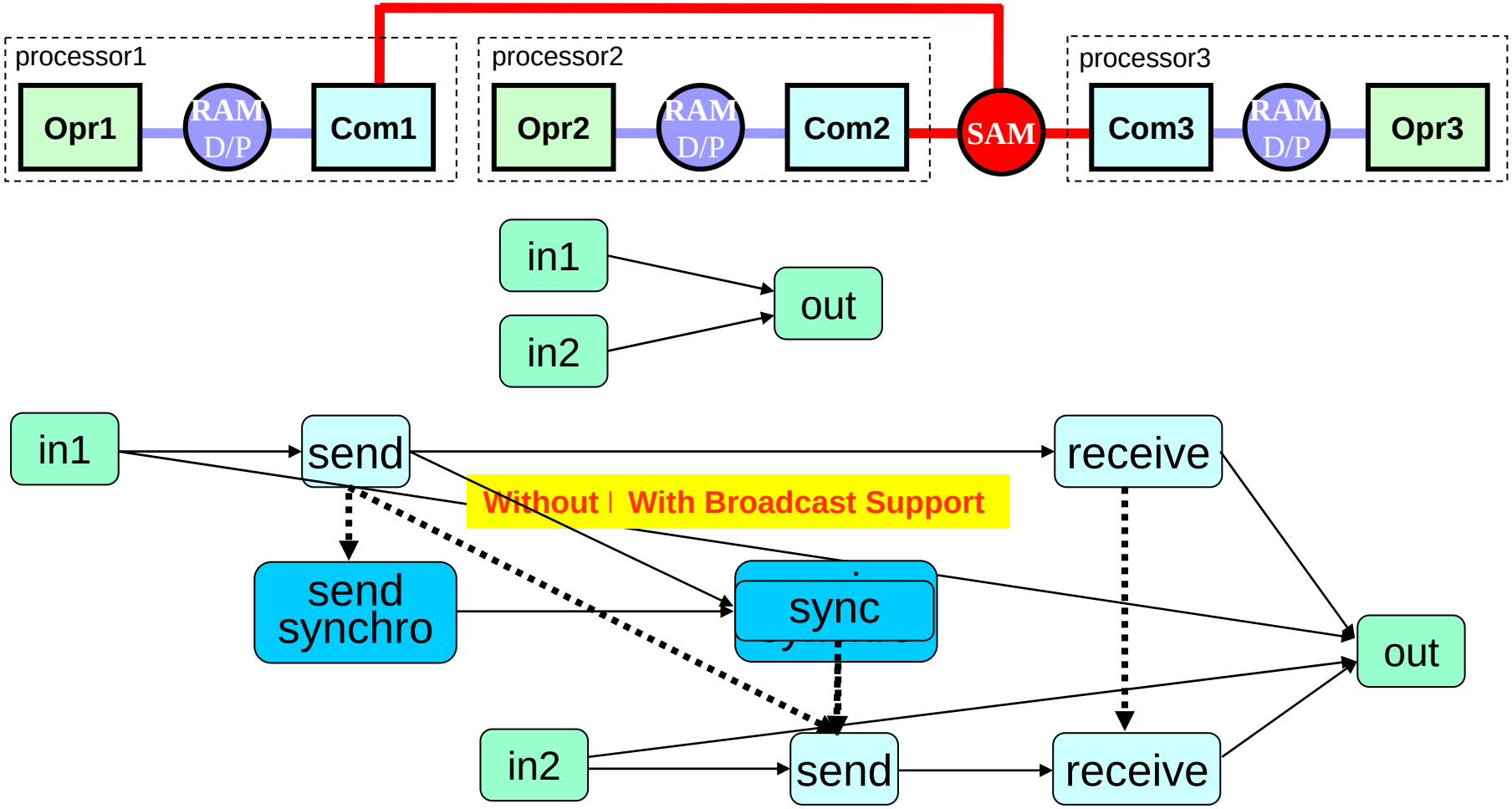
Graph transformation: External Composition Law

Implementation graph Gal' = Gal*Gar

# Implementation example: Point to Point SAM



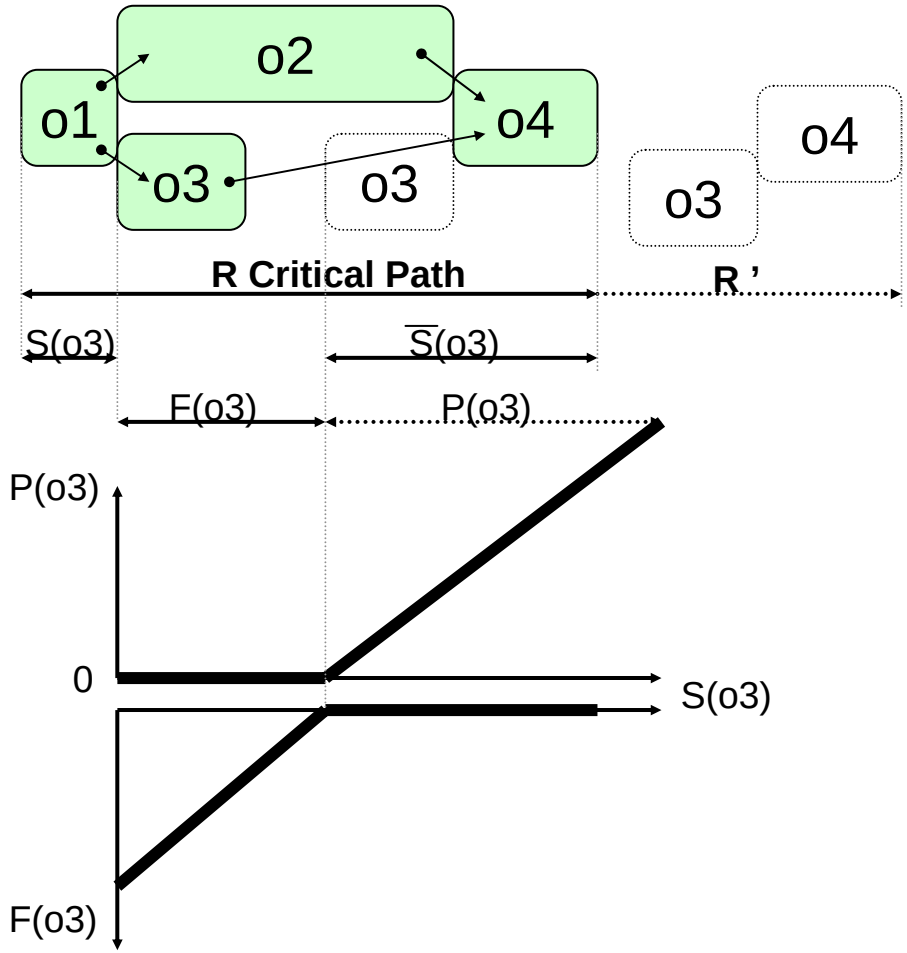
# Implementation example: Multipoint SAM



# 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),

$$opr_{o_i} = \underset{\forall p_i \in Proc}{\mathbf{min}}\ \sigma(o_i, p_i)$$

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

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

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

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

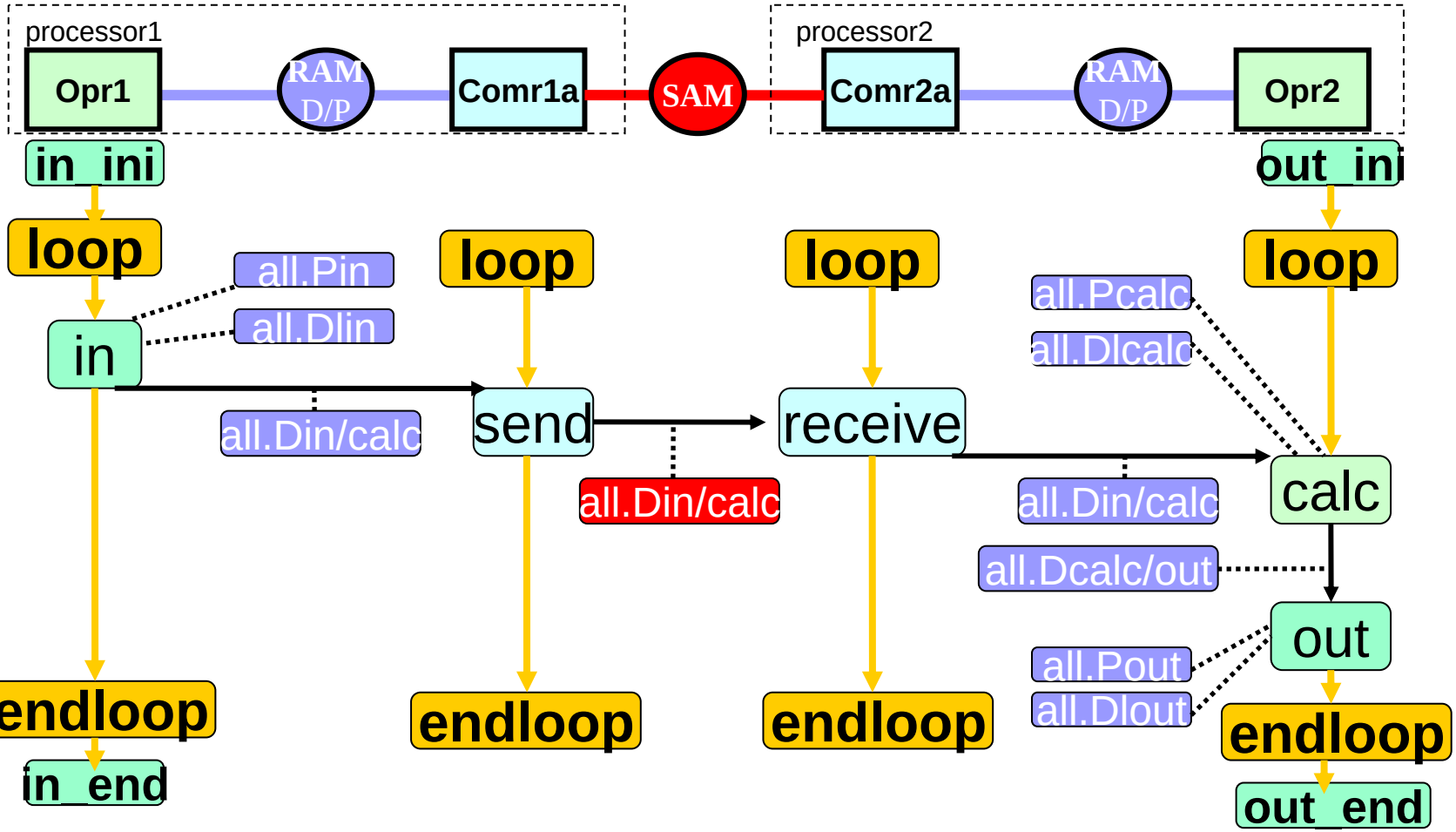
# Real-time executives generation

- **Without deadlock, very low overhead**
- **Processor independent Macro-Code (.m4)**
  - Extensible, easily portable (C, asm, …)
- **Processor dependent executive kernel (.m4x) Macros definitions for:**
  - Microcontrolers: MPC555, MC68332, 80C196
  - DSP: Sharc, TMS320C40
  - Microprocessors: i80x86, PPC G4, C/Unix
  - Communication Media: links, CAN, RS232, TCP/IP

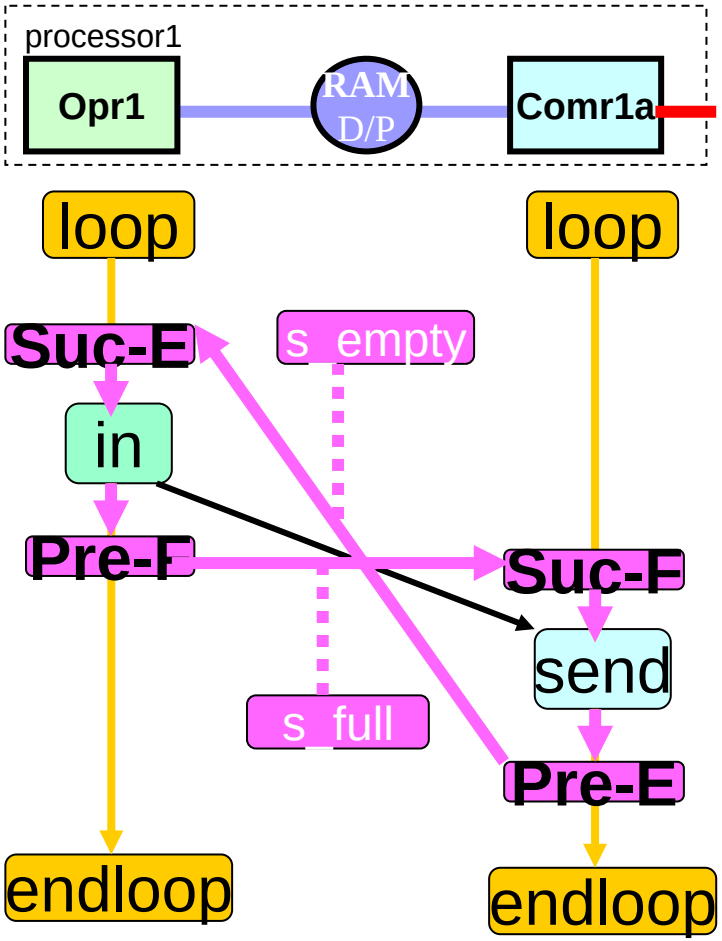
# Graphs transformations

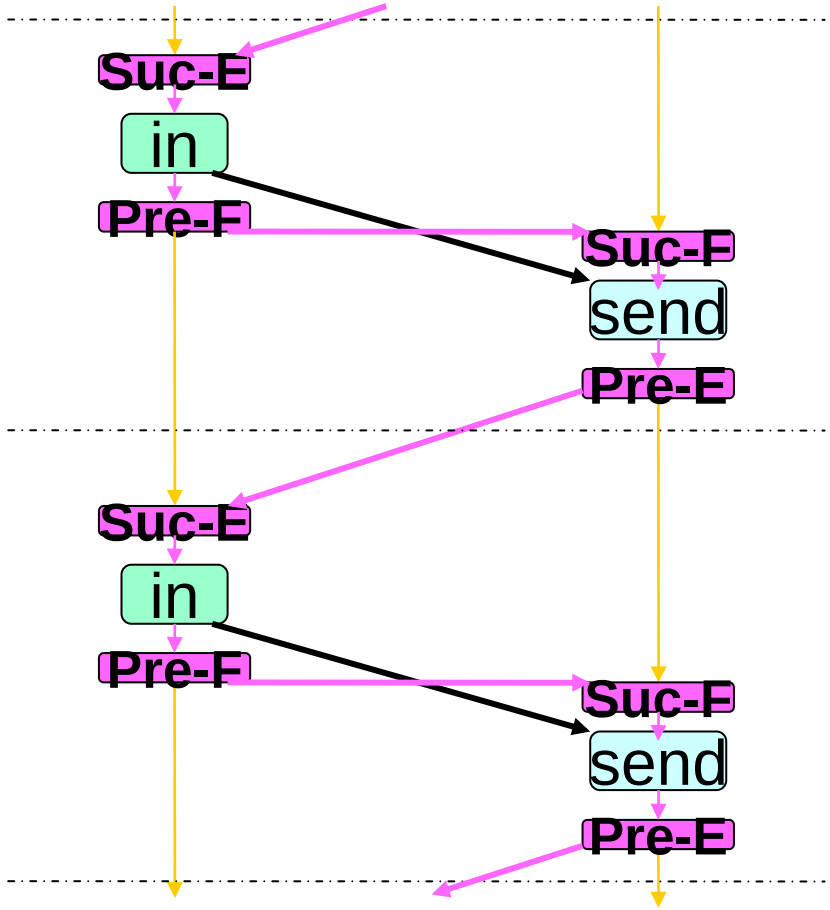
- **Optimized implementation graph *in* execution graph by adding *system vertices* :**
  - Extraction of the infinitely repeated sub-graph
  - Explicit repetition by adding ***loop/endloop vertices***
  - Adding ***init./finalisation vertices*** for inputs and outputs
  - calc/com synchronization by adding ***Pre/Suc vertices*** using semaphores
- **Execution graph *in* macro-code**
- **Macro-code (file.m4) *in* source program:** macro-processor (gm4) + executive libraries (macros definitions file.m4x)

# Execution graph: explicit repetition

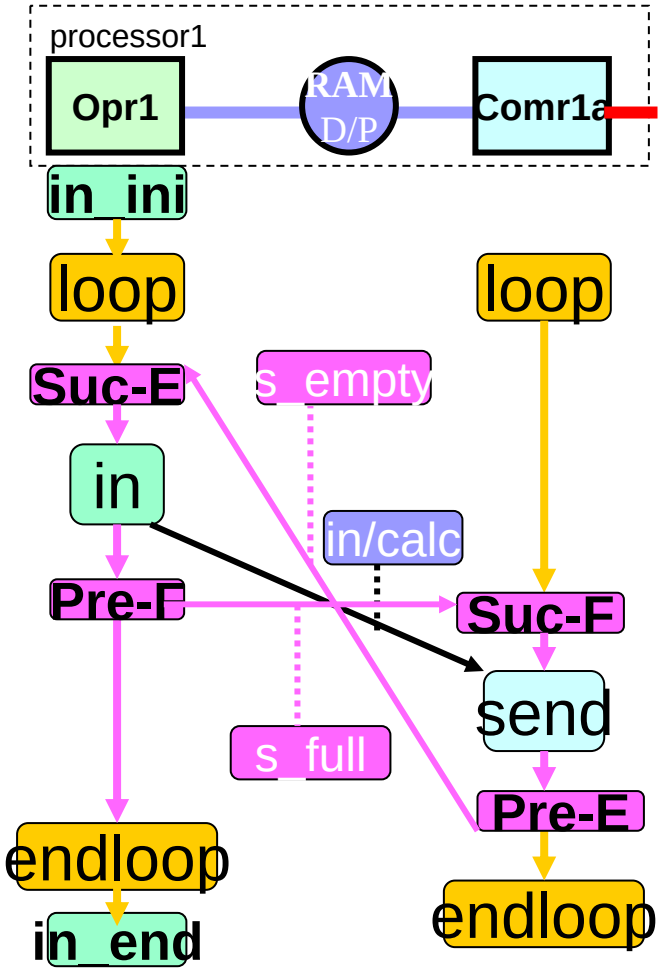


# Execution graph: synchronizations



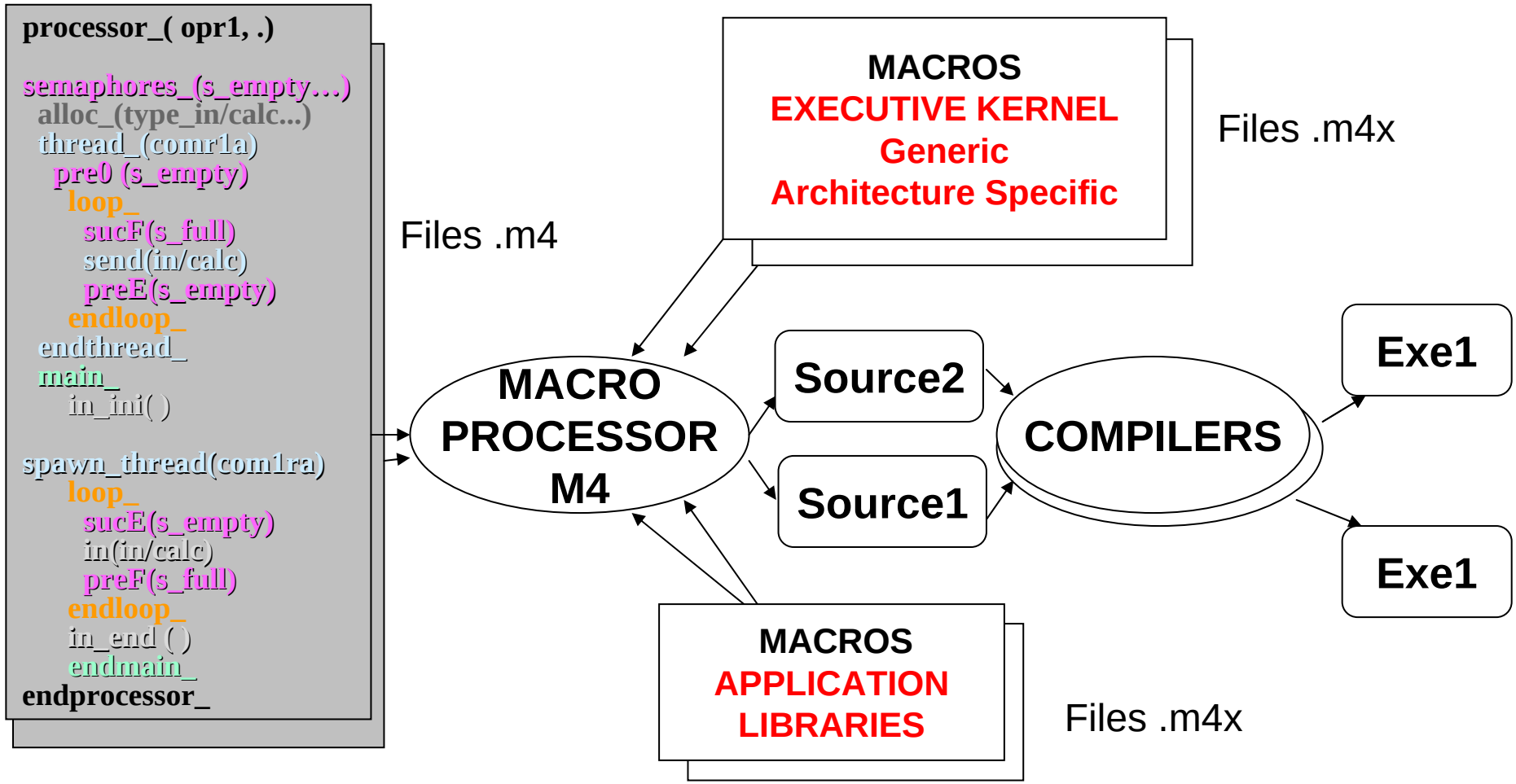



# Macro-code generation

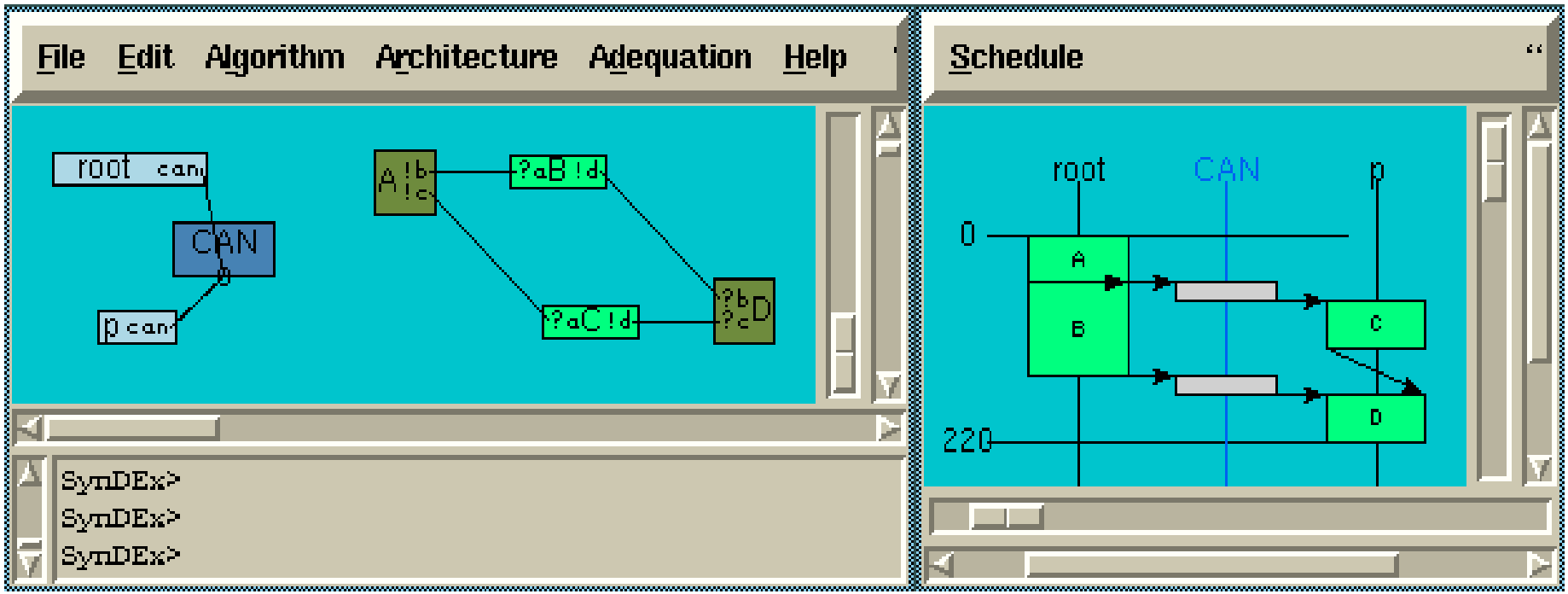



```
processor_( opr1, .)
  semaphores_(s_empty…)
  alloc_(type_in/calc...)
  thread_(comr1a)
    pre0 (s_empty)
      loop_
        sucF(s_full)
        send(in/calc)
        preE(s_empty)
      endloop_
  endthread_
  main_
      in_ini( )
      spawn_thread(com1ra)
      loop_
        sucE(s_empty)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

# Macro-code expansion and compilation



# Distibuted real-time executive generation: simple application

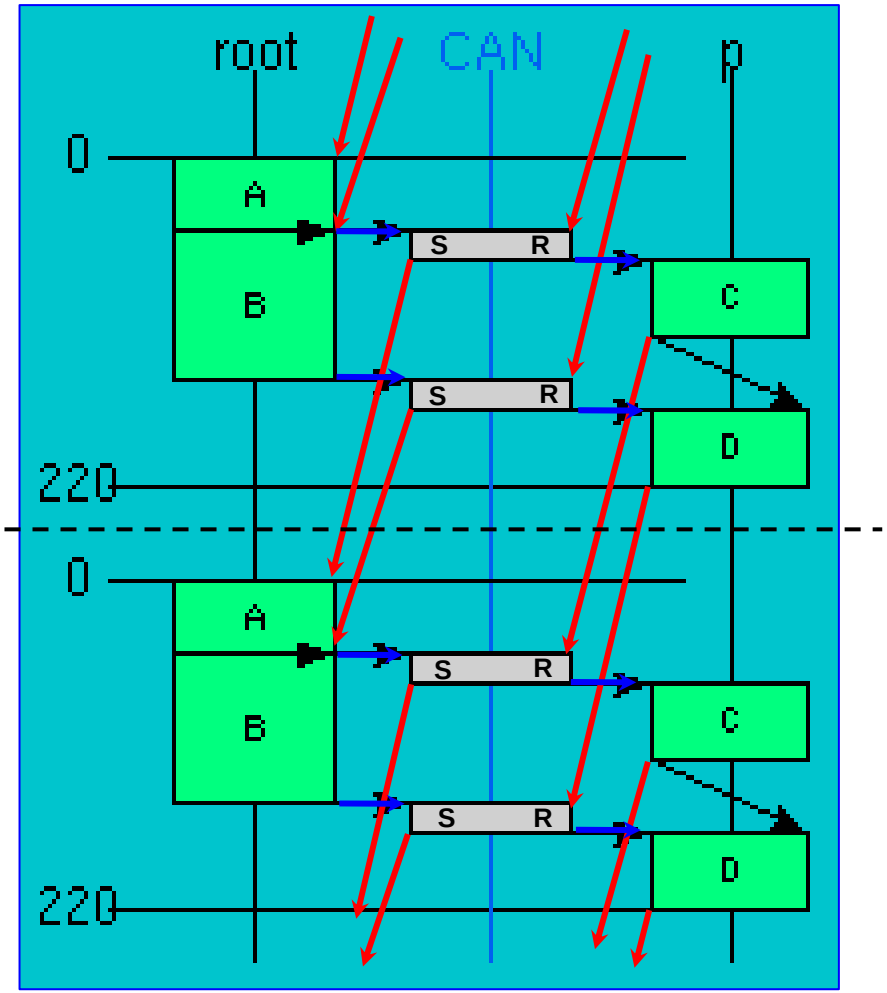


# Executive generation: Synchro calc/com

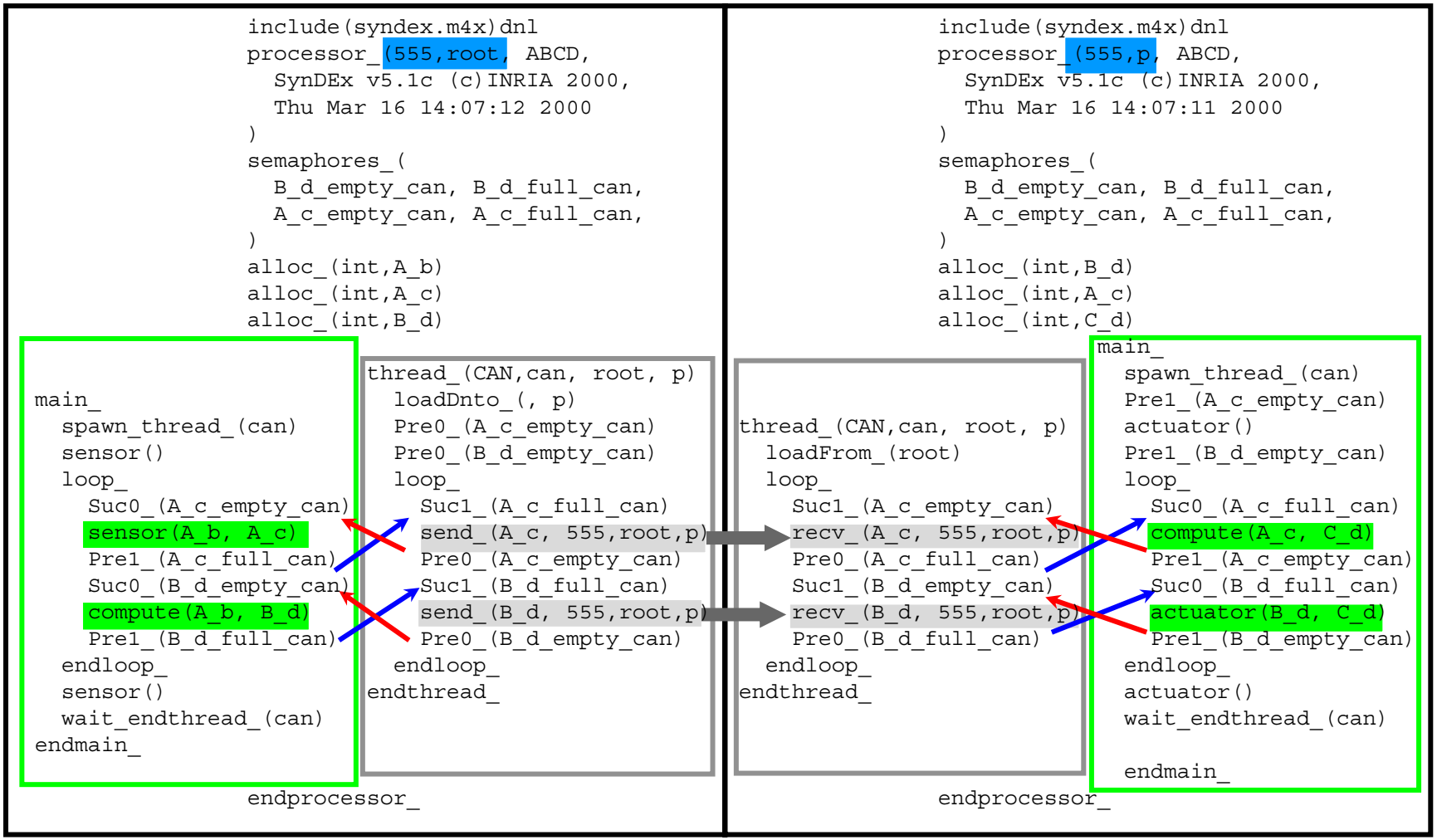
Buffer

Intra-repetition Synchro

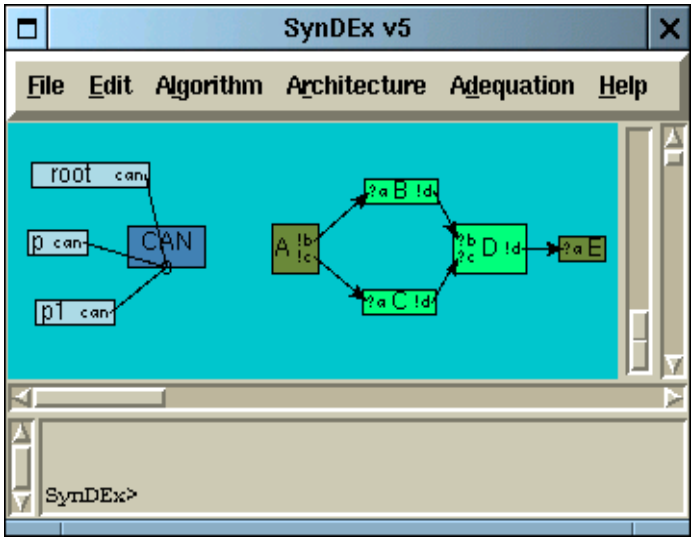
Inter-repetition Synchro

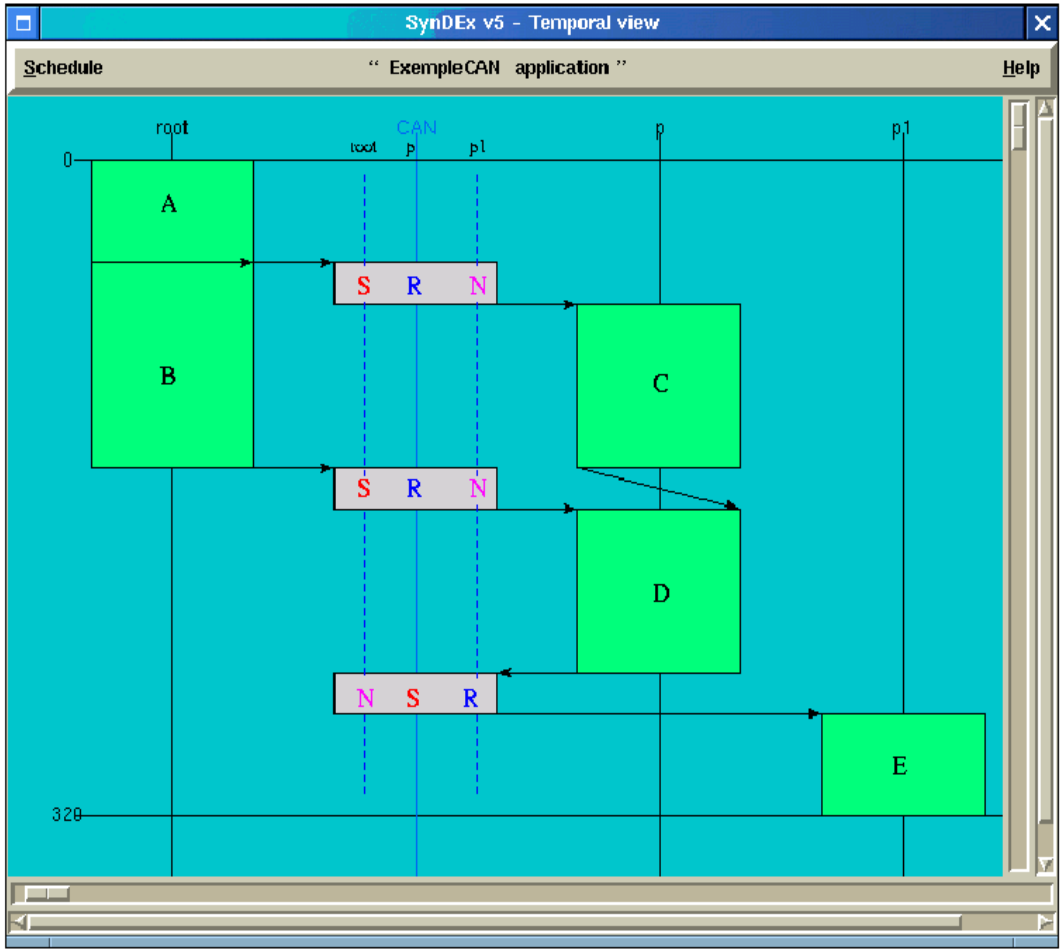


# Executive generation: Code



# Executive generation: Synchro. comm.



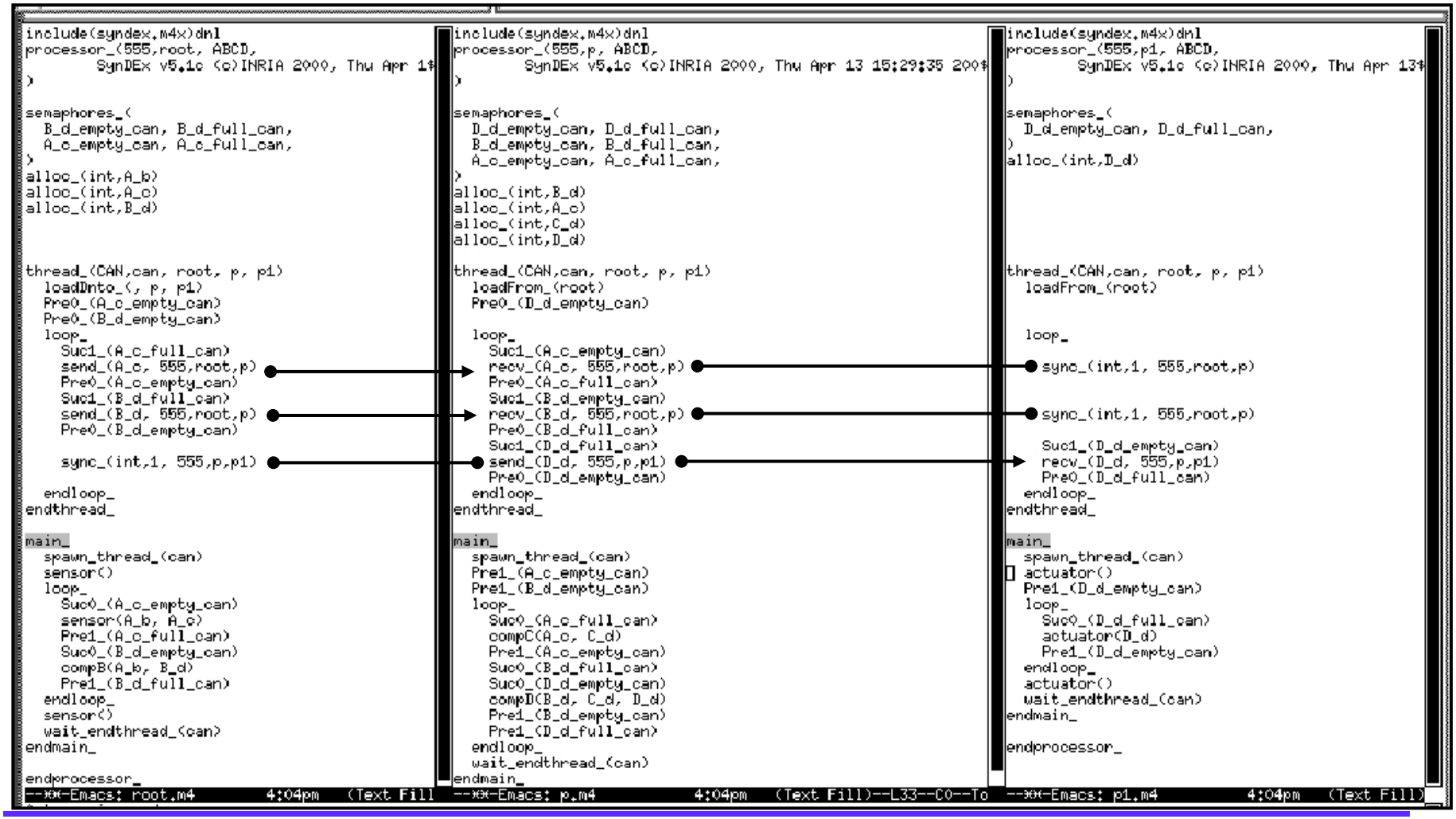




**S : Send**

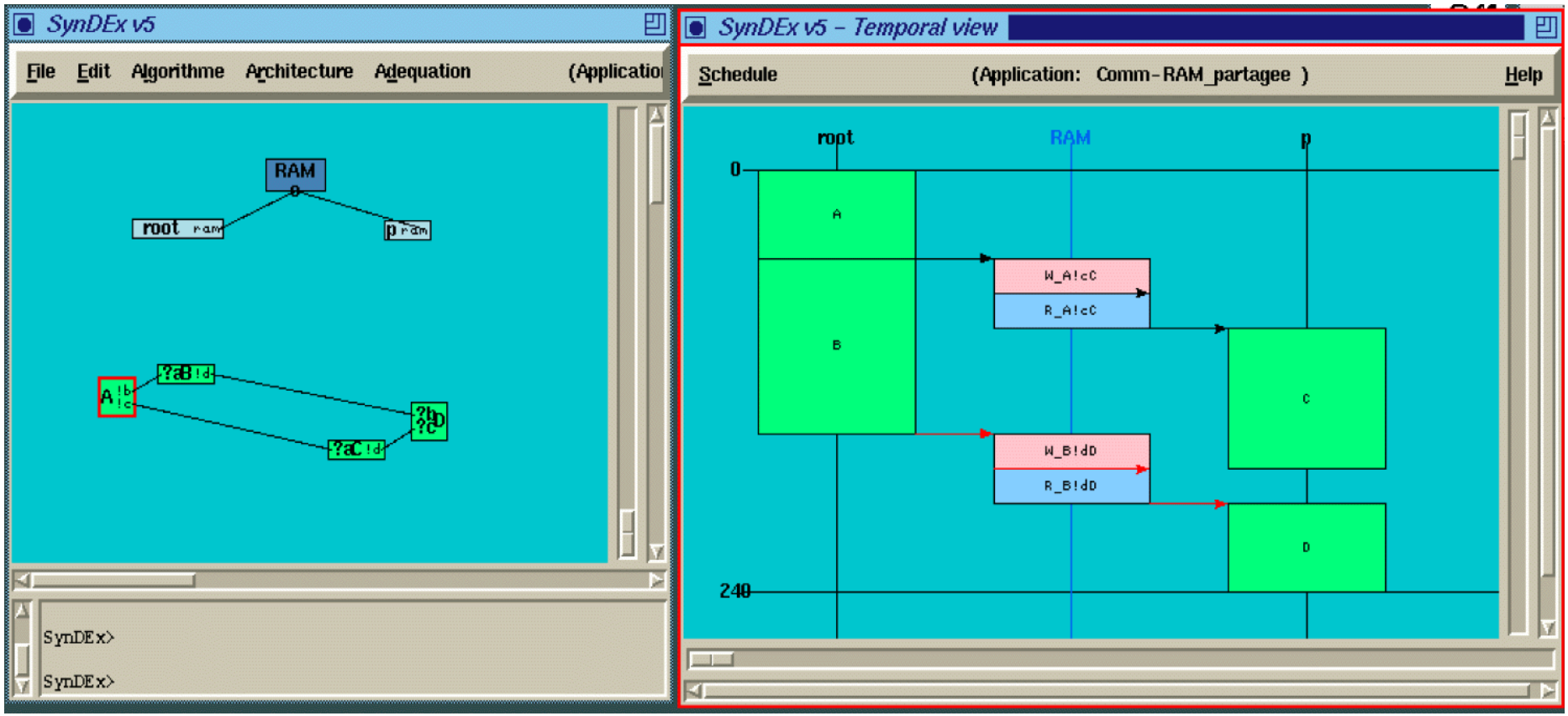
**R :Receive**

**N : Sync**

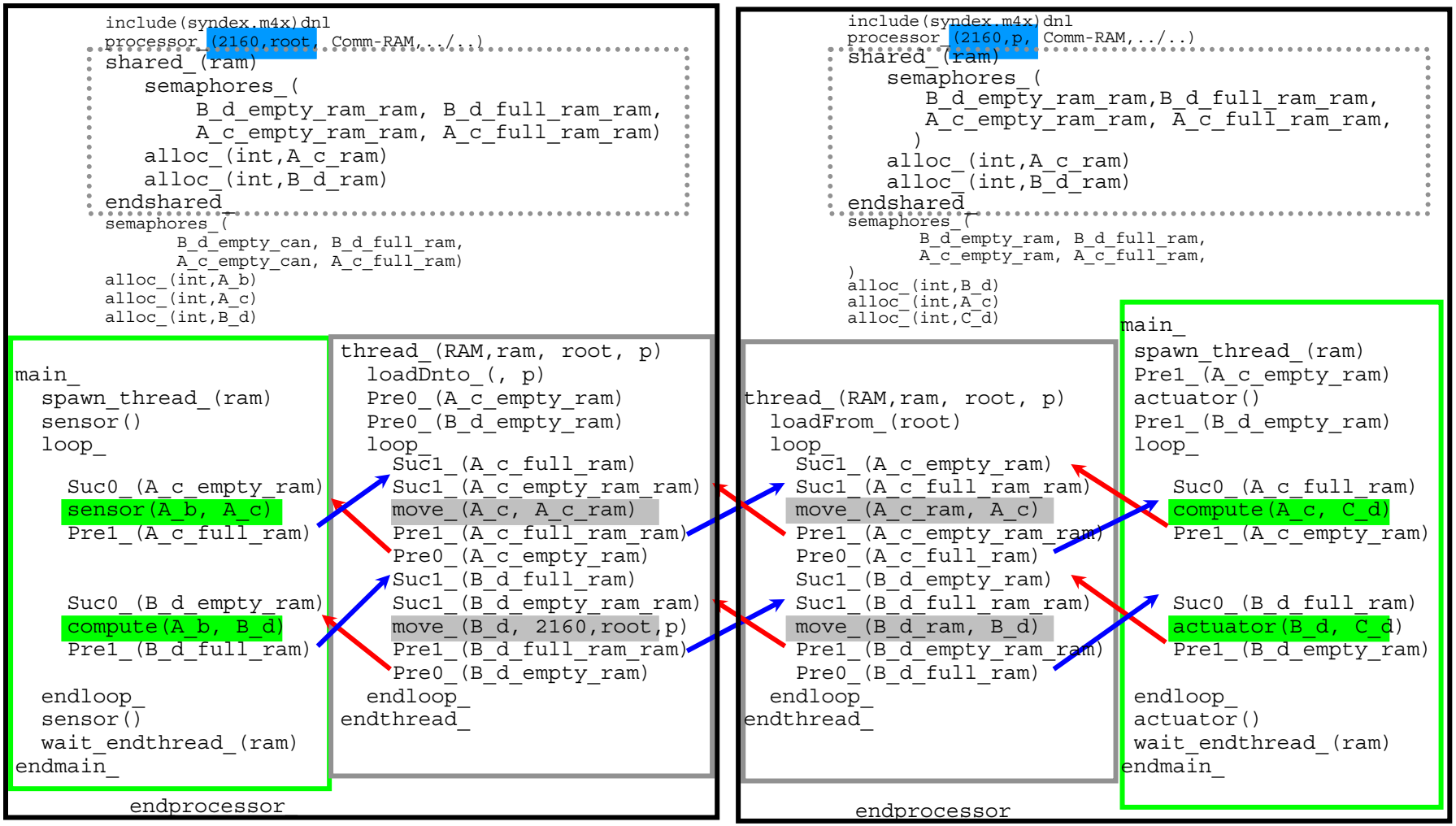
# Executive generation: Code



# Executive generation: Shared mem. comm.



# Executive generation: Shared mem. comm.



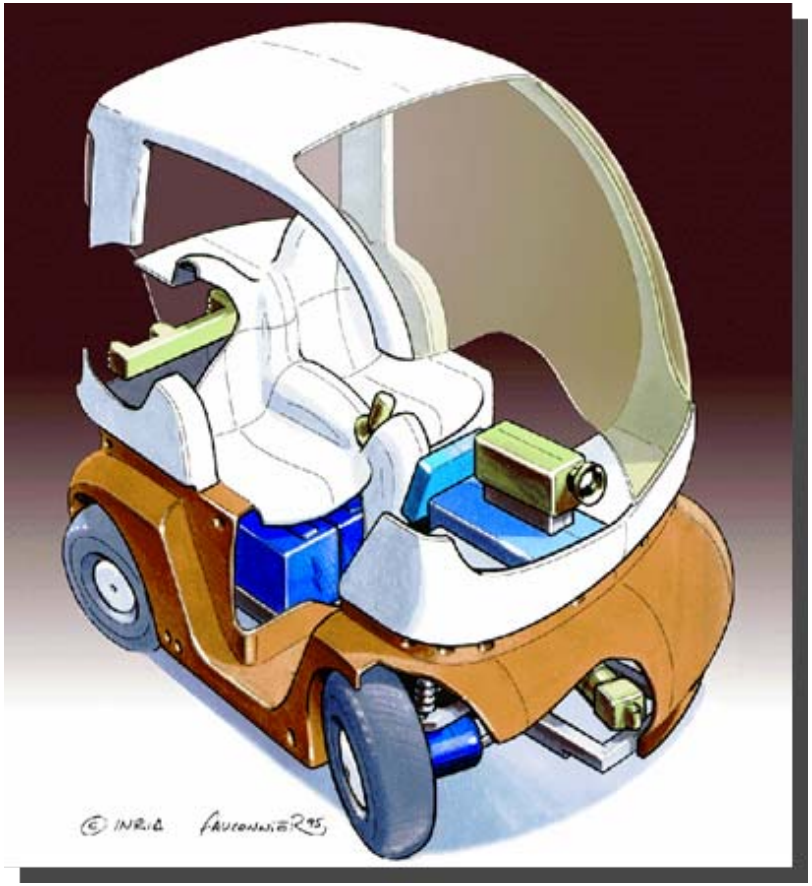
# Configuration of standard executives

- **Without Deadlock**
- **Translation of macro-code in standard executive configuration file**
- **Standard executives**
  - RT-Linux, RT-AI
  - Osek
  - etc

# System level CAD software: SynDEx

- **AAA methodology support**
- **Interface with high-level languages**
- **Graphic interactif 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

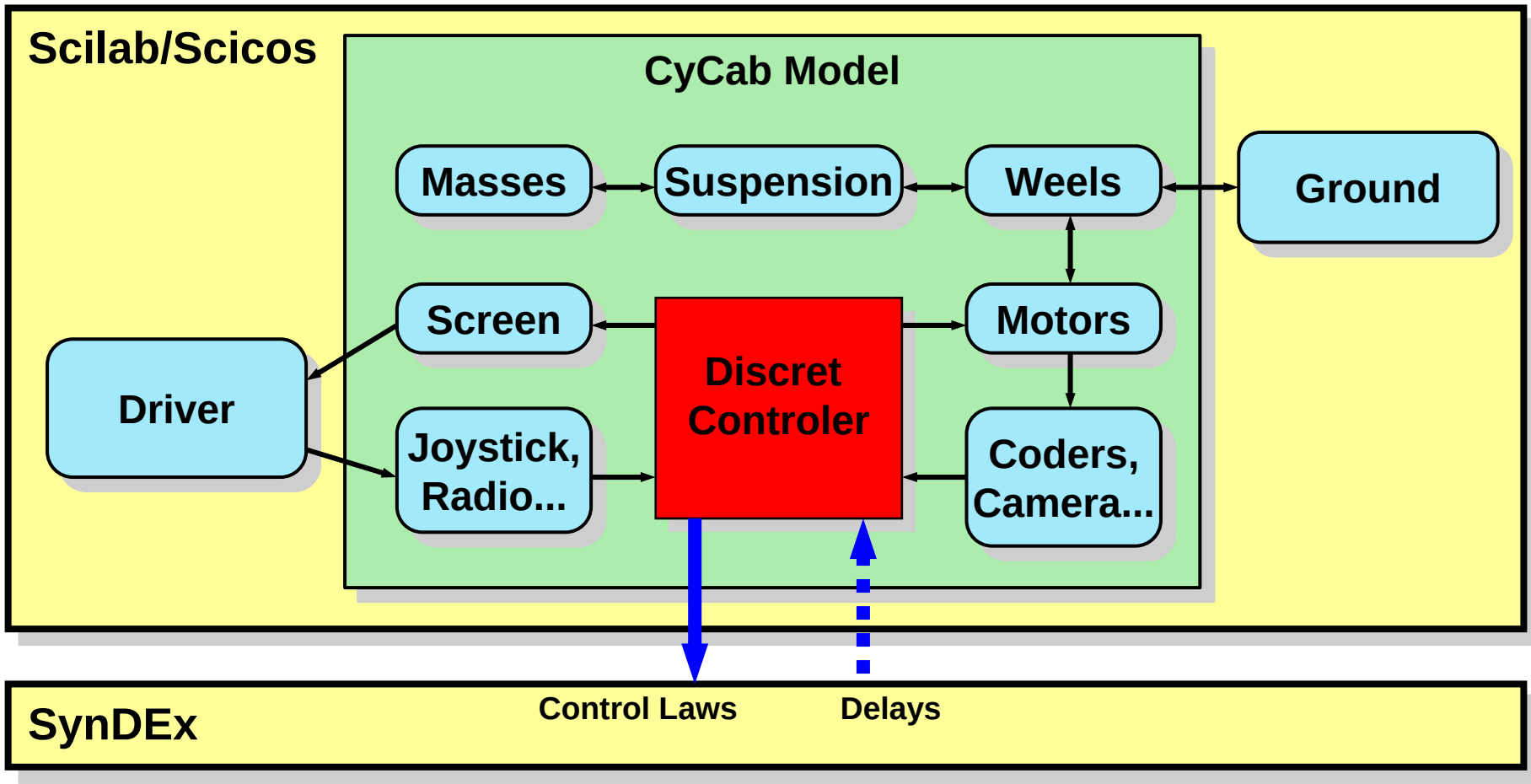



- Speed 30km/h
- Electric motors
- 4 weel drive
- 2 steerings FWD RWD
- 1 to 4 MPC555, 1PC
- CAN Bus

Industrialized by Robosoft

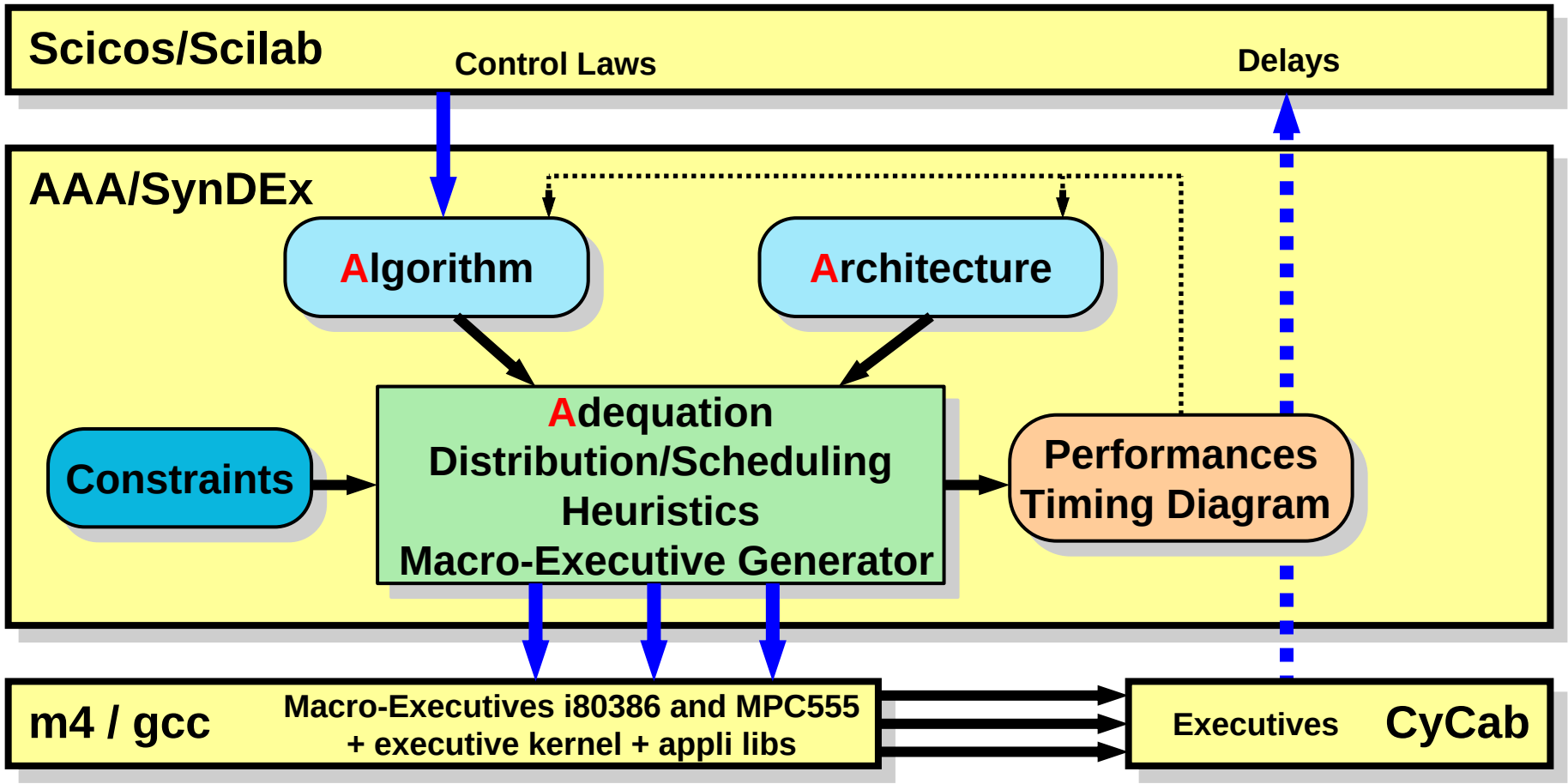
http://www.robosoft.fr

# Modeling/Simulation
# Scilab/Scicos Free at: www.scilab.org

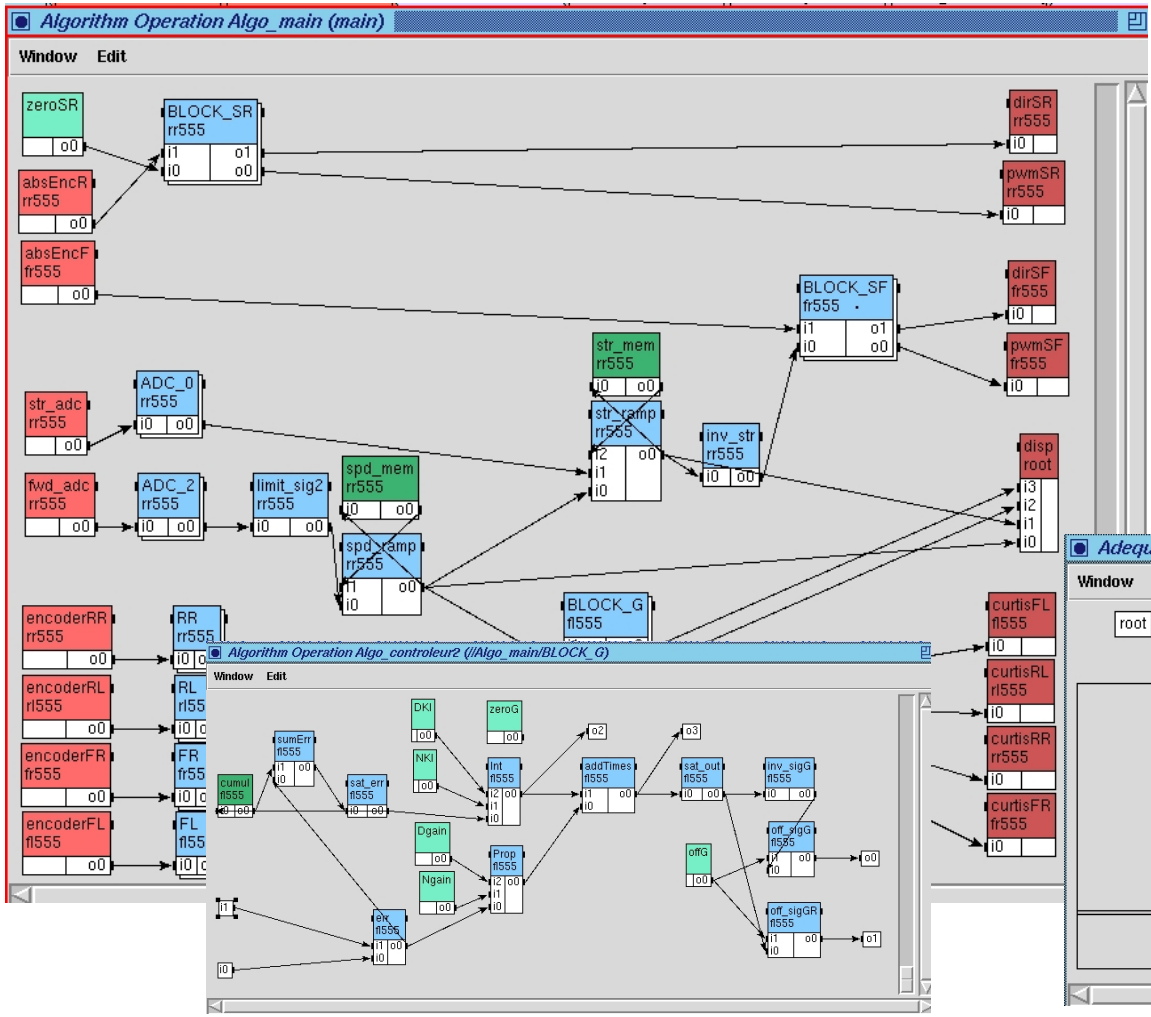


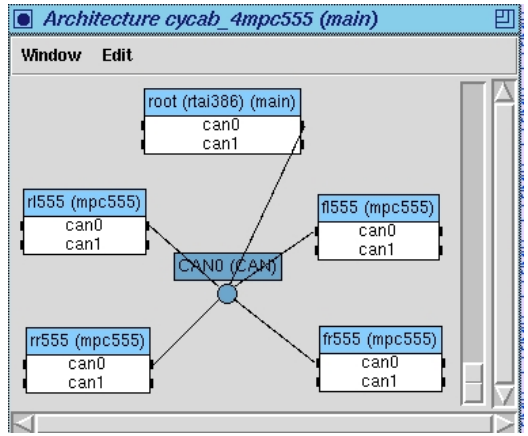
# Implementation AAA/SynDEx

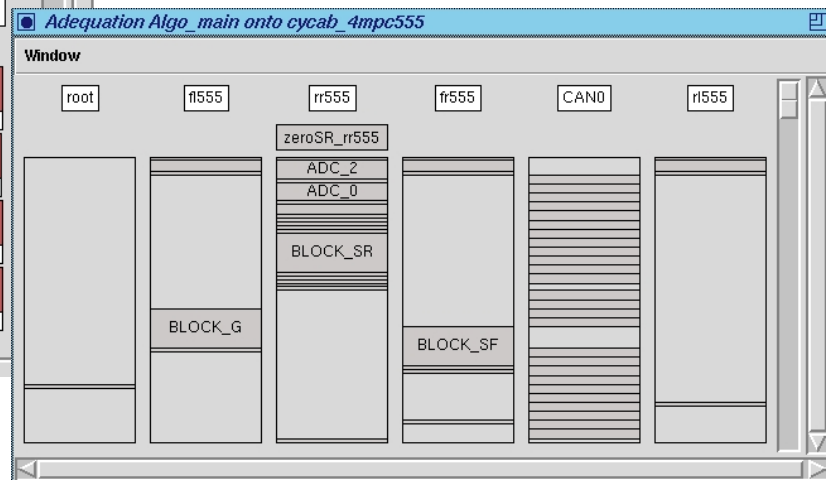
Free: www.syndex.org



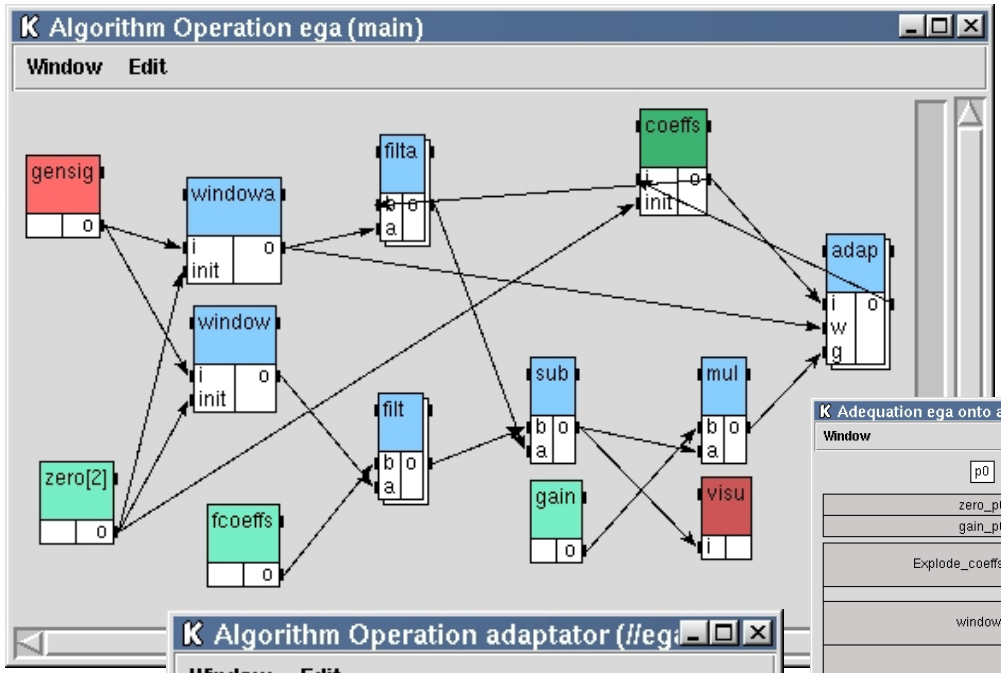
# Manual driving application for the CyCab

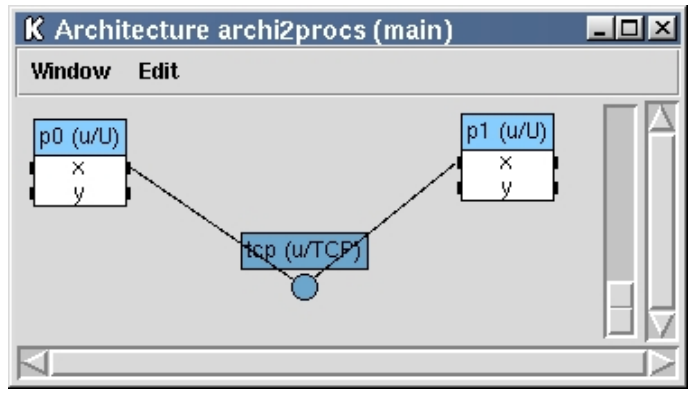


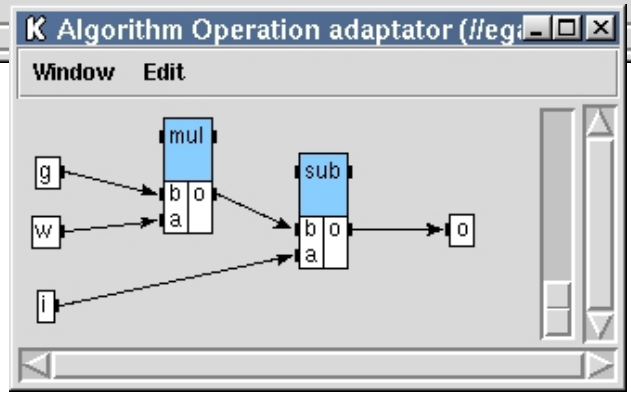



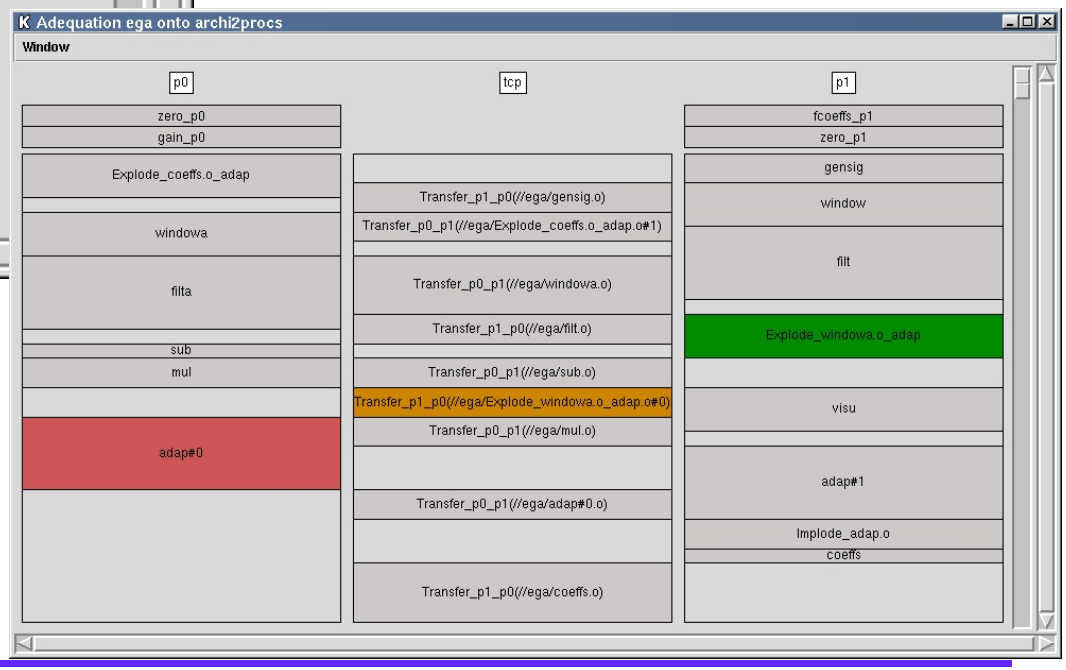



# Adaptive equalizer application

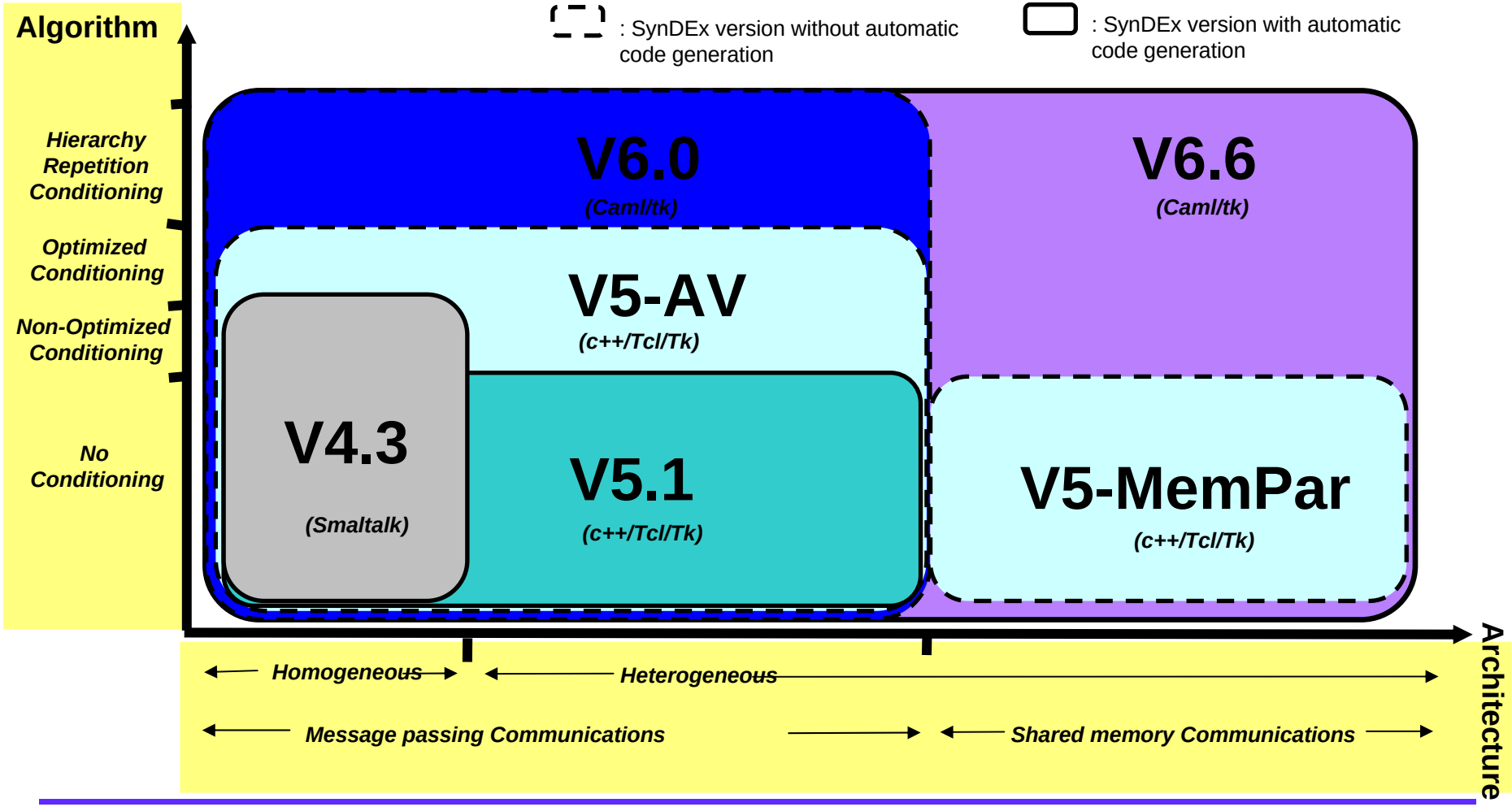


# SynDEx: Version evolution



# 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
- **SynDEx-IC A2SI ESIEE collaboration**

# Conclusion

- **Development cycle is dramatically reduced**
  - Safe design
  - Rapid prototyping  and optimized implementation
  - Deadlock free executives, debug only executive Kernel
- **Work in progress**
  - Real-time multi-constraints: latencies and cadences (period)
    - Off-Line with or without preemption, strict period or jitter
    - In-Line for aperiodic events : slot shifting, adaptive scheduling
  - GALS: globally asynchronous locally synchronous
  - SynDEx/SynDEx-IC coupling: Codesign environment
  - Automatic Software/Hardware partitioning
  - Fault tolerance: collaboration with POP-ART team