Verification of UML models with timing constraints using IF

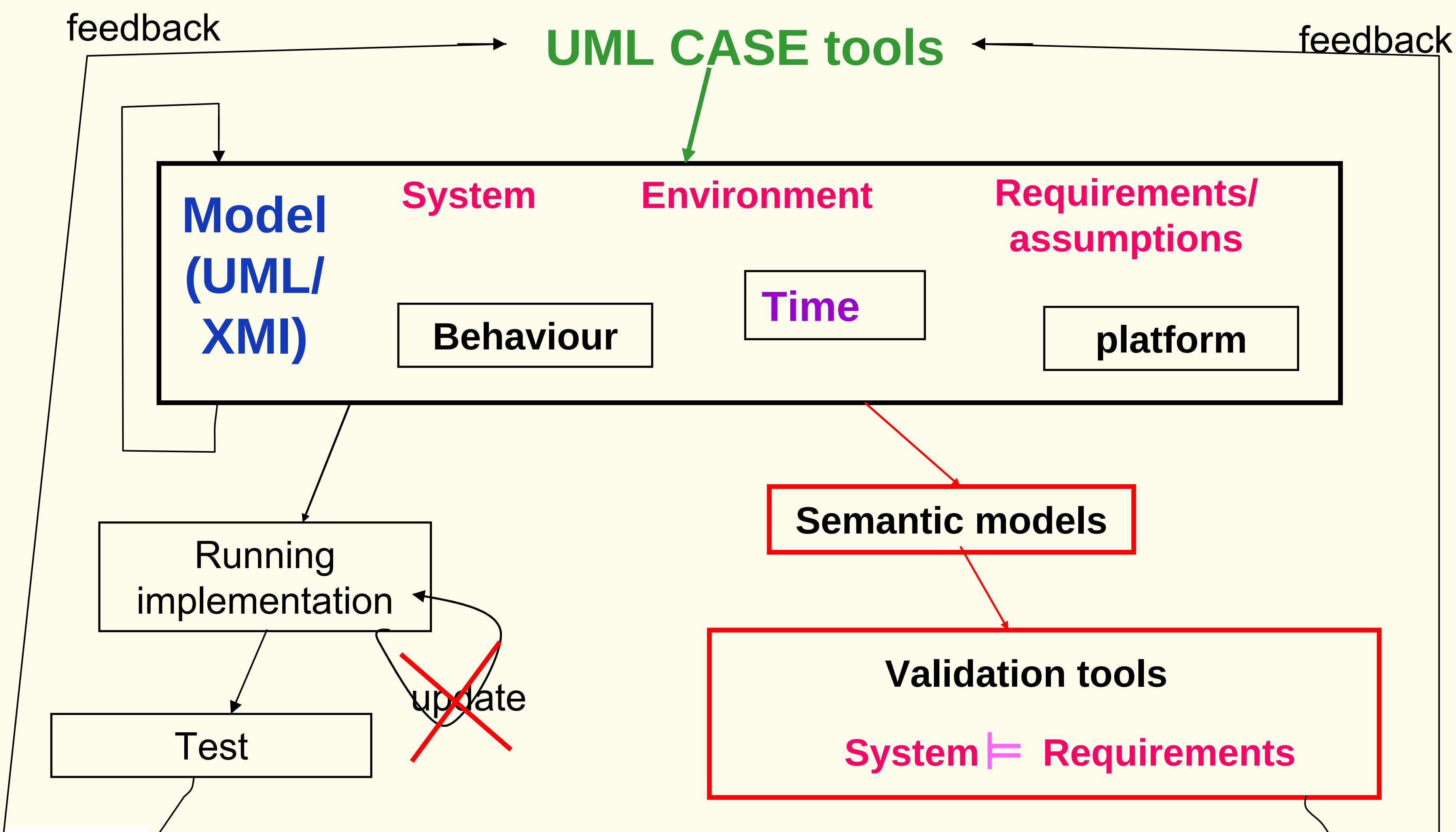
Susanne Graf
Verimag

http://www-if.imag.fr/

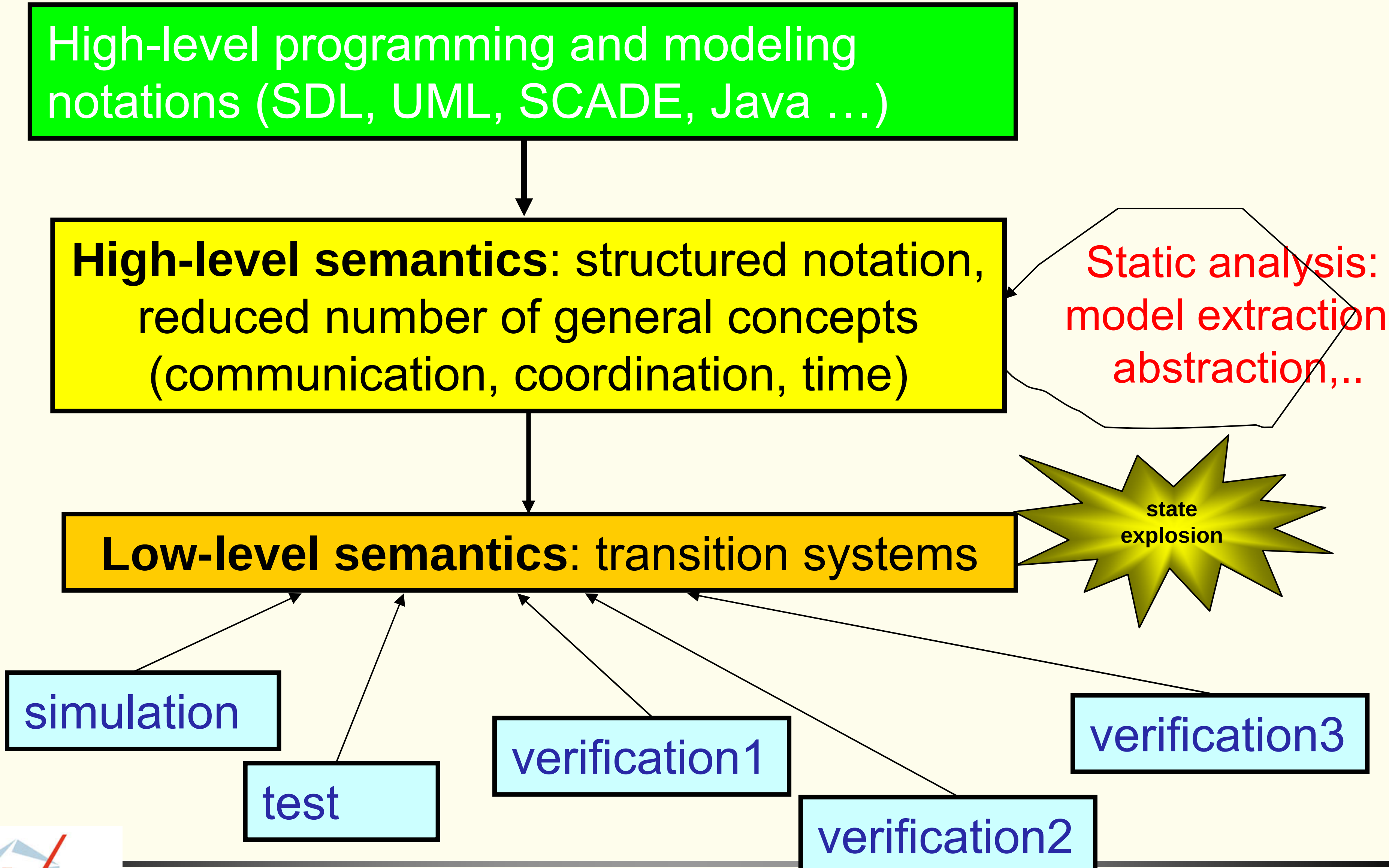
http://www-omega.imag.fr/

http://www-verimag.imag.fr/~graf/Artist-summerschool/

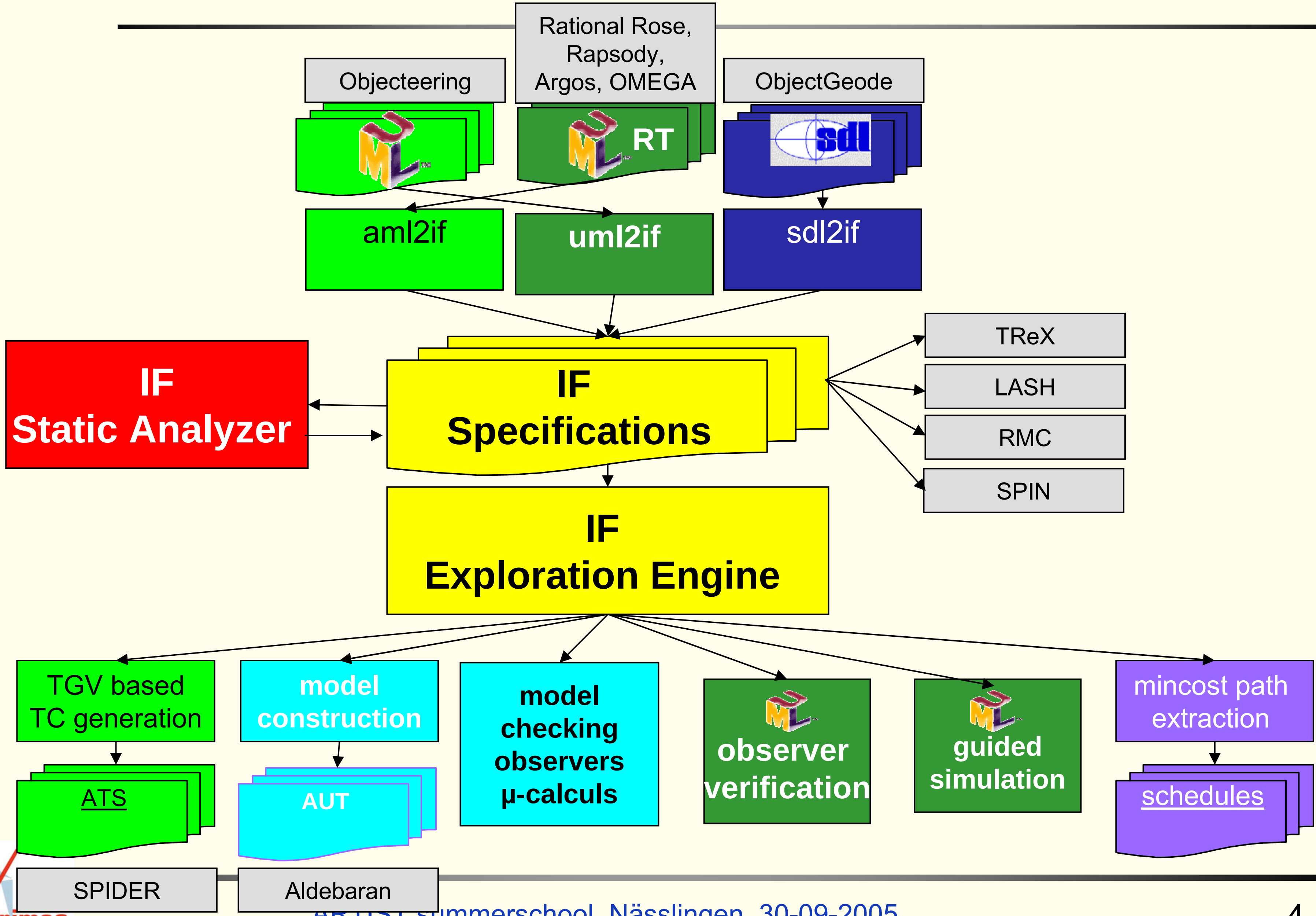
IST OMEGA: validation in the context of model-based development of real-time systems



The IF toolbox: approach



IF tool-set: overview



Outline

IF language

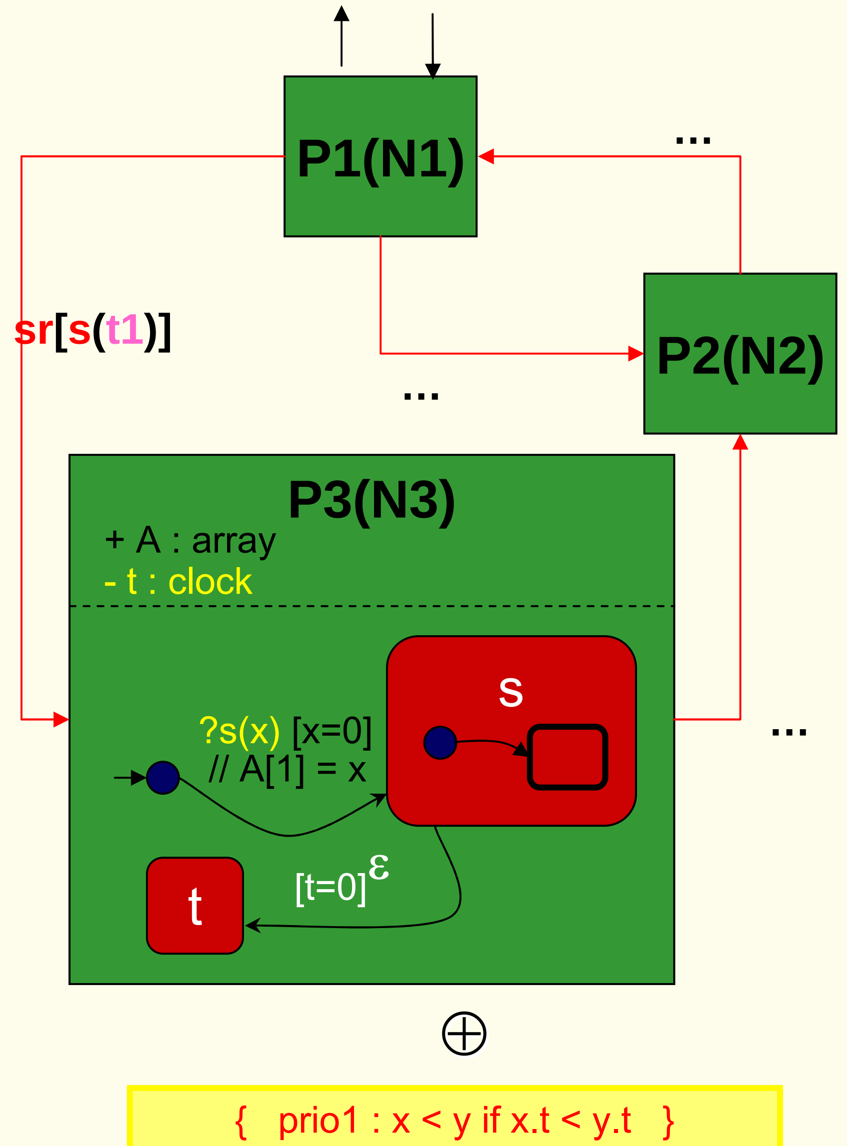
System =

Set of **concurrent processes**
- **timed automata with urgency**
- hierarchical automata
- complex + abstract data types
- dynamic creation
- non-determinism

Communication
- asynchronous channels
- various routing / delay / loss models
- shared variables

Execution control
- dynamic priorities

Assumptions and Requirements
- observers (weak synchronization)



Observers

- assumptions
- requirements

Processes (components)

Extended *hierarchical timed automata*

(non-determinism, dynamic creation)

Data

- predefined data types (basic types, arrays, records)
- abstract data types

Interactions

- asynchronous channels
- shared variables

Execution control

- priority rules

IF: system description

// processes

process P1(N1)

...

endprocess;

...

process P3(N3)

...

endprocess;

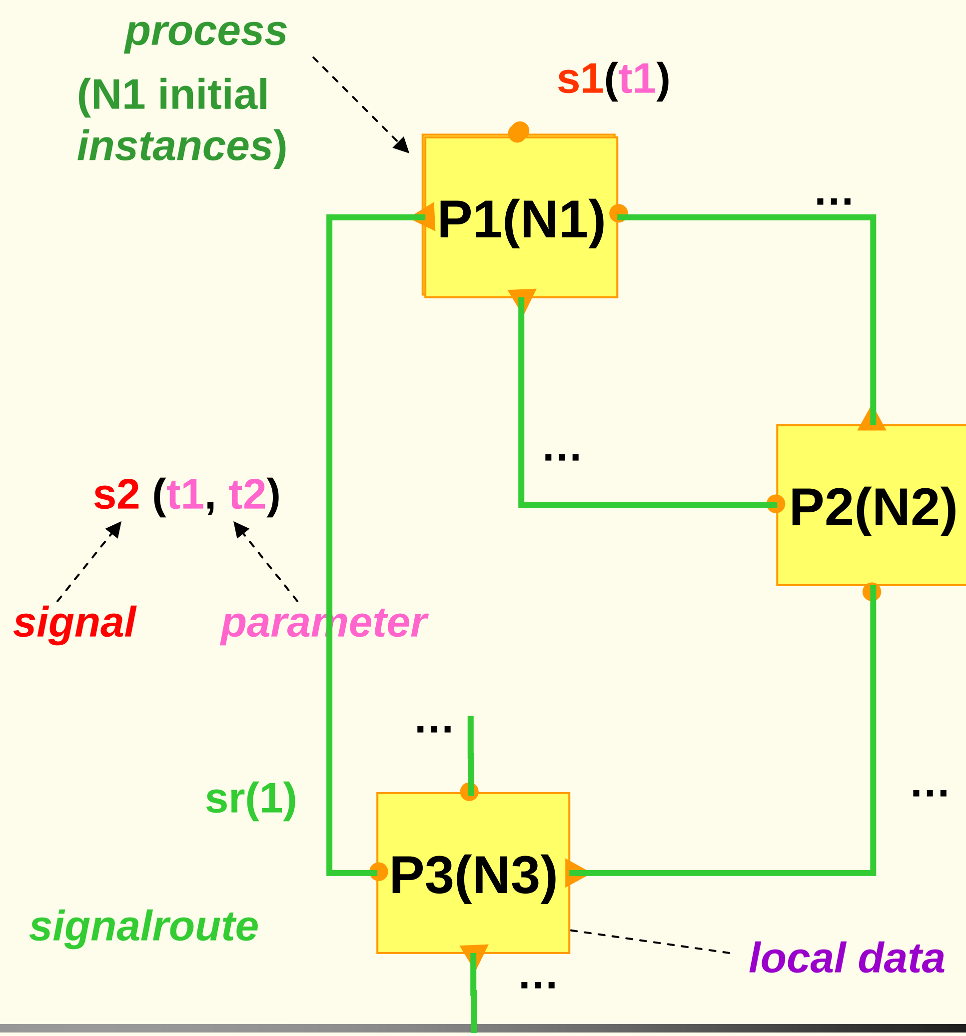
// signalroutes

signalroute sr1(1) …
from P1 to P3 ;

// signals

signal s1(t1)

signal s2(t1, t2),



IF: process description

Process = hierarchical timed automaton

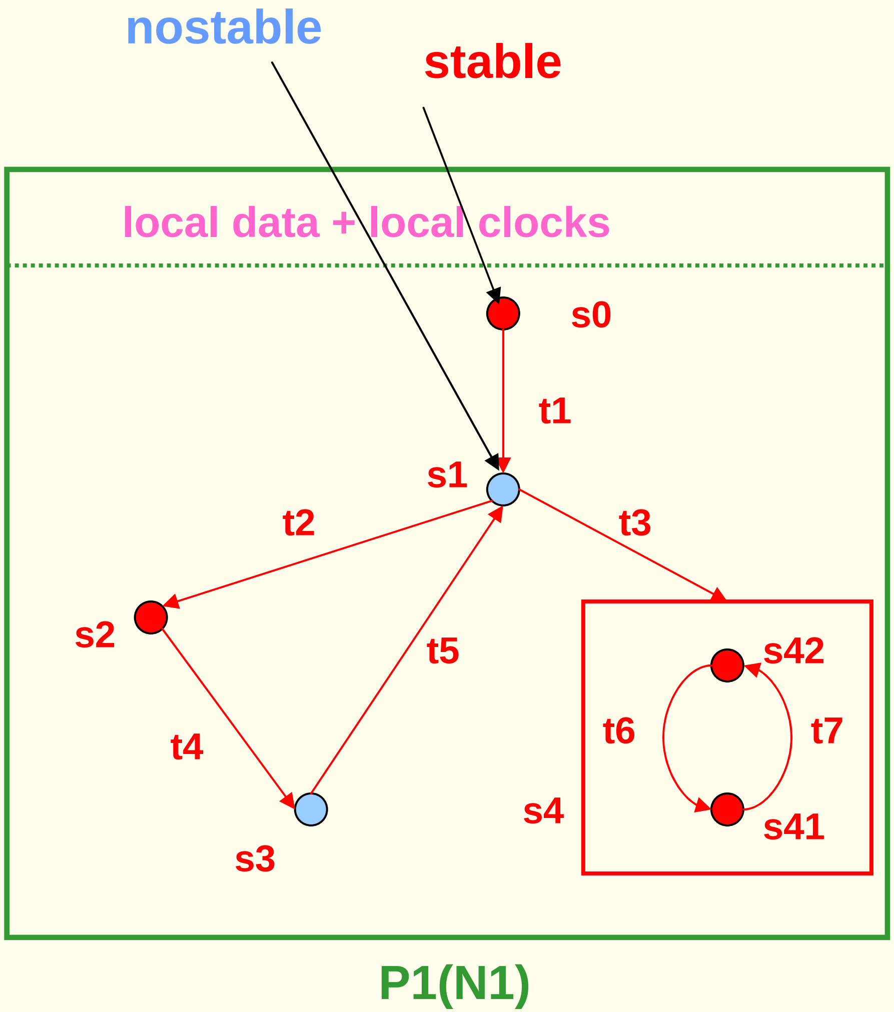
```
process P1(N1);
fpar ... ;

// types, variables, constants,
procedures

state s0 ...;
      ...  // transition t1
endstate;

state s1 #unstable...;
      ... // transitions t2, t3
endstate;

  ...   // states s2, s3, s4
endprocess;
```

parameters

local data

state

outgoing transitions



transition = *urgency* + trigger + body

```
state s0
…
t1  urgency eager
    provided x!=10;
    when c2 >= 4;
    input update(m);
       body    ….
    nextstate s1;
…
endstate;
```

urgency

untimed guard

timed guard

signal consumption from the process buffer

= **trigger**

statement list, ext. proc.

sequential. conditional, or iterative composition

statement = data assignment
message sending,
process or signalroute creation or destruction, …

IF: signal routes

signal route = connector = process to process communication channel with ***attributes,*** can be ***dynamically*** created

route name

initial instance number

attributes

signal set

signalroute s1(1) **#unicast #lossy #fifo**
from server **to** client **with** grant, fail;

endpoints

attributes:

- queuing policy: **fifo | multiset**
- reliability: **reliable | lossy**
- delivery policy: **peer | unicast | multicast**
- *delay policy: urgent | delay[l,u] | rate[l,u]*

IF: dynamic priorities

- priority order between process instances p1, p2

  (free variables ranging over the active process set)

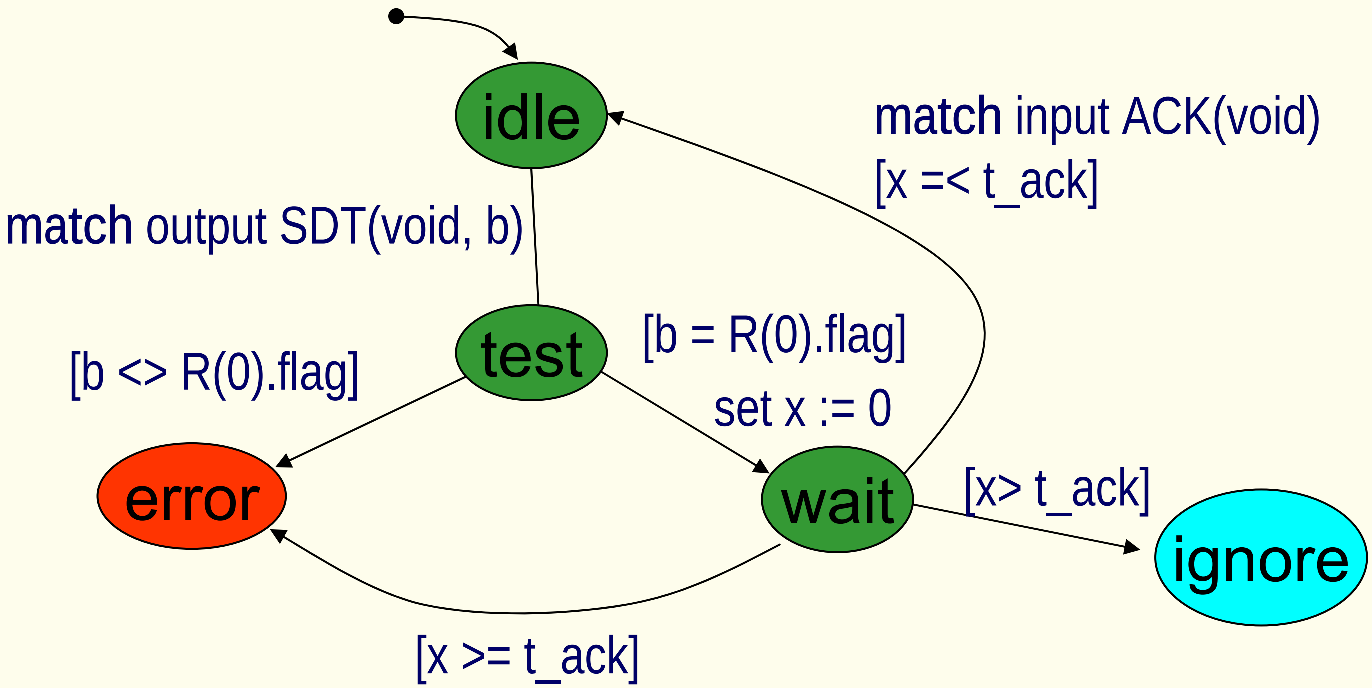
priority_rule_name: p1 < p2 **if** *condition*(p1,p2)

- semantics: *only maximal enabled processes can execute*

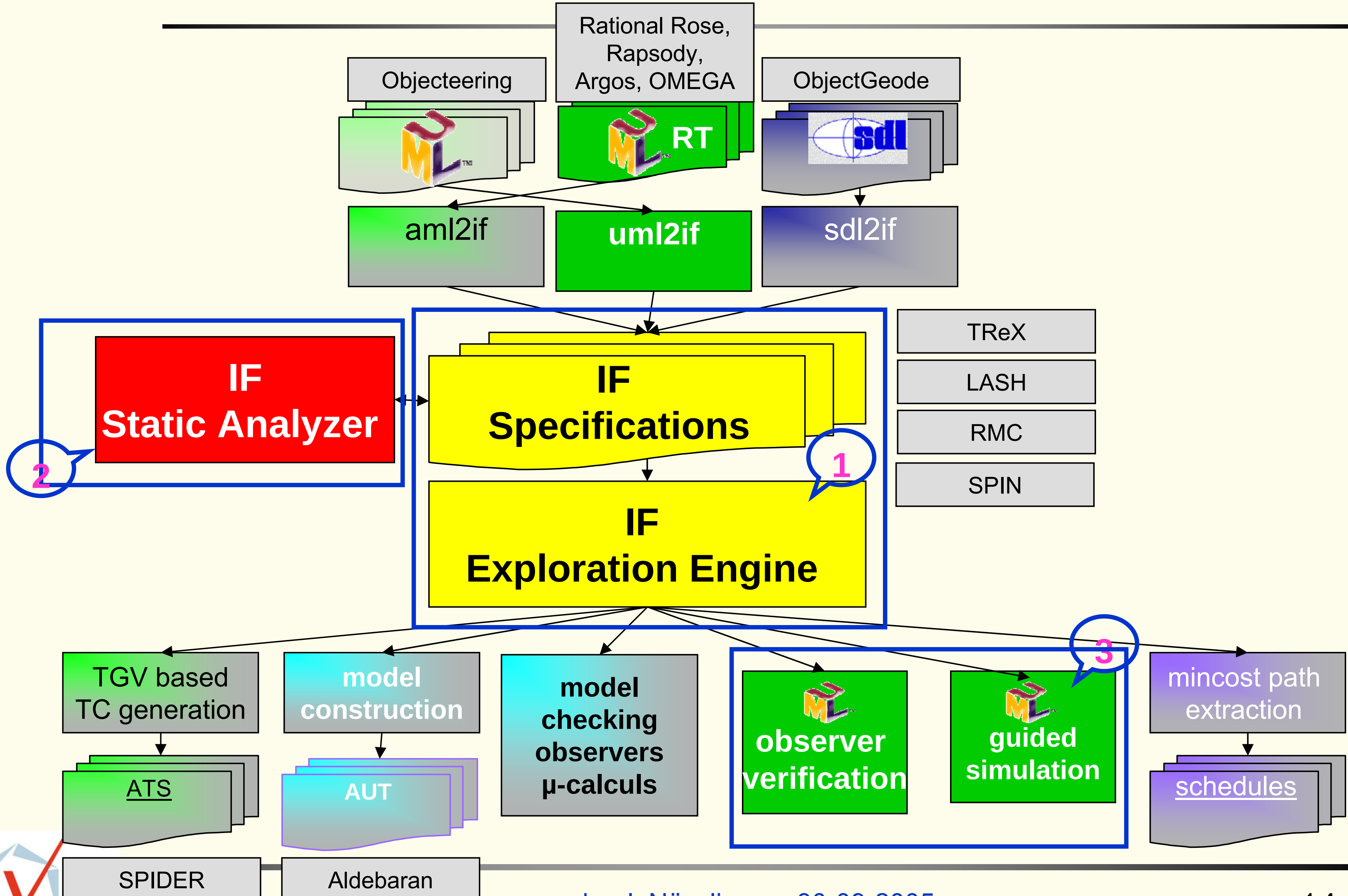
- examples of scheduling policies
  - **fixed priority**: p1 < p2 if p1 instanceof T and p2 instanceof R
  - **EDF**: p1 < p2 if Task(p2).timer < Task(p1).timer
  - **run-to-completion**: p1 < p2 if p2 = manager(0).running

IF: observer for the expression of properties

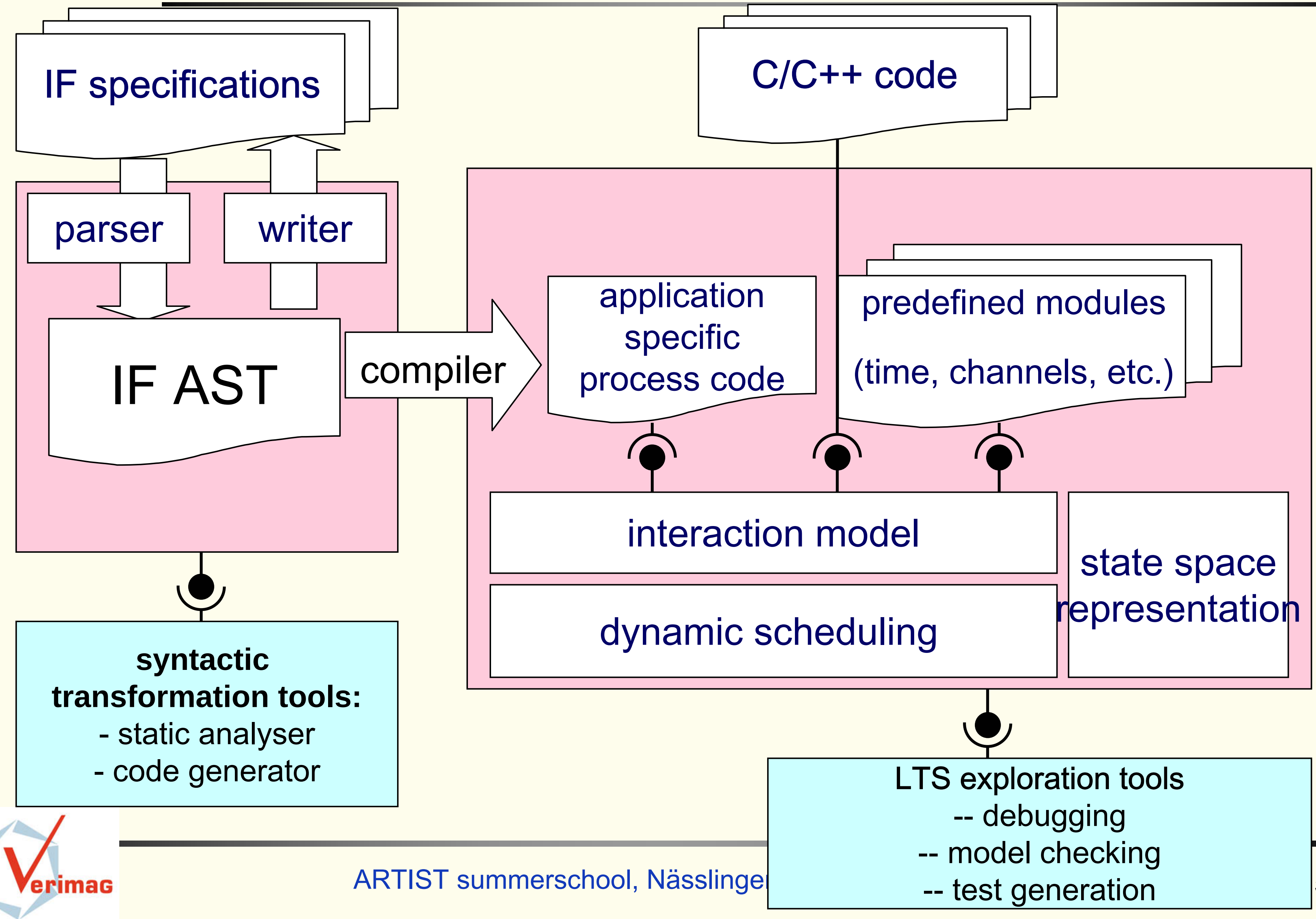
- *Observers* specify safety properties (assumptions and requirements)
- Event language acceptors: processes with specific triggers for monitoring events, system state, elapsed time
- 3 types of states : normal / error / ignore
- *Semantics*:
  - transitions triggered by monitored events are executed with highest priority
  - Reaching an ignore state = reaching un uninteresting part (assumption)



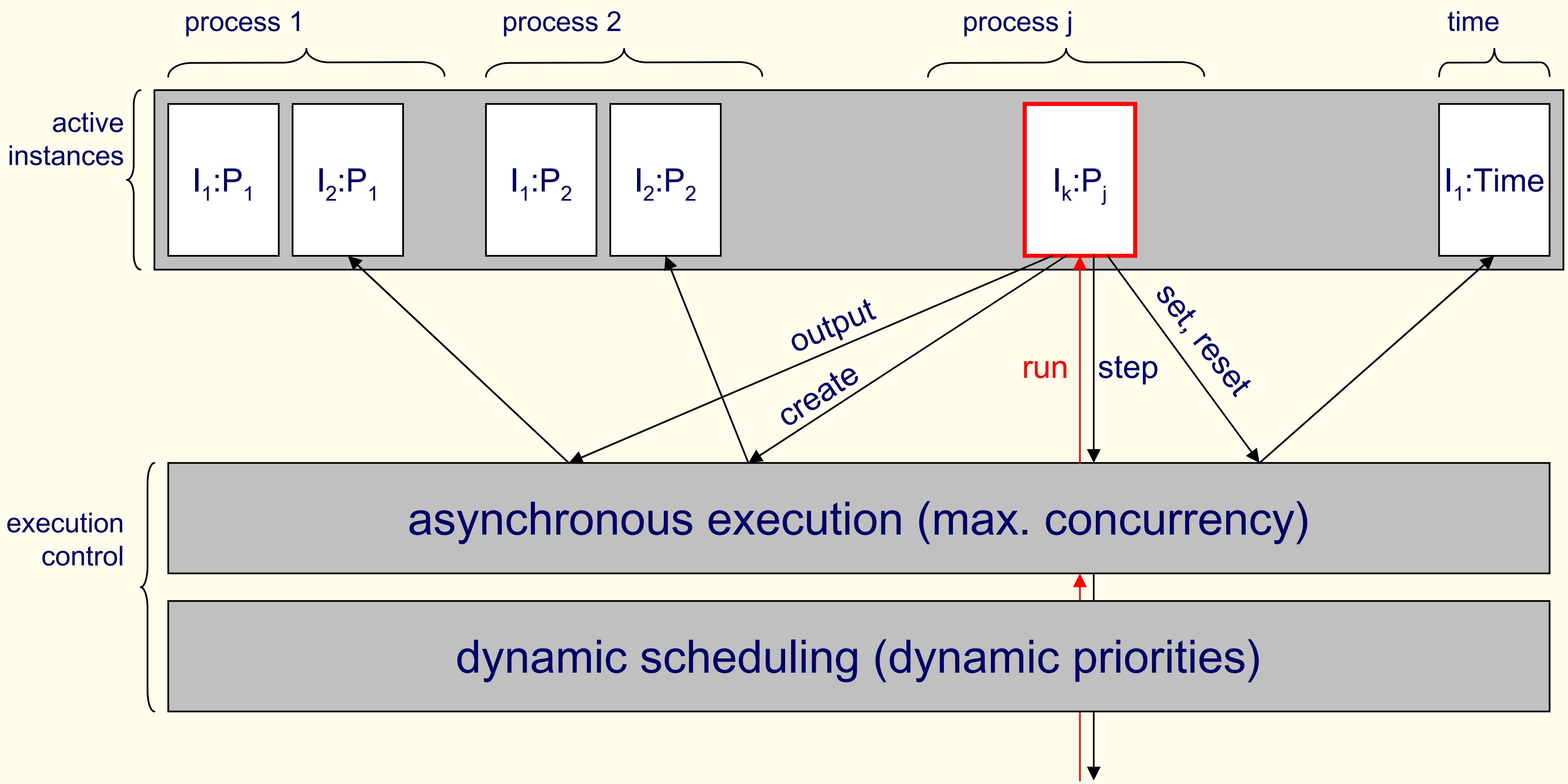
IF tool-set: overview



IF: core components



IF: exploration engine

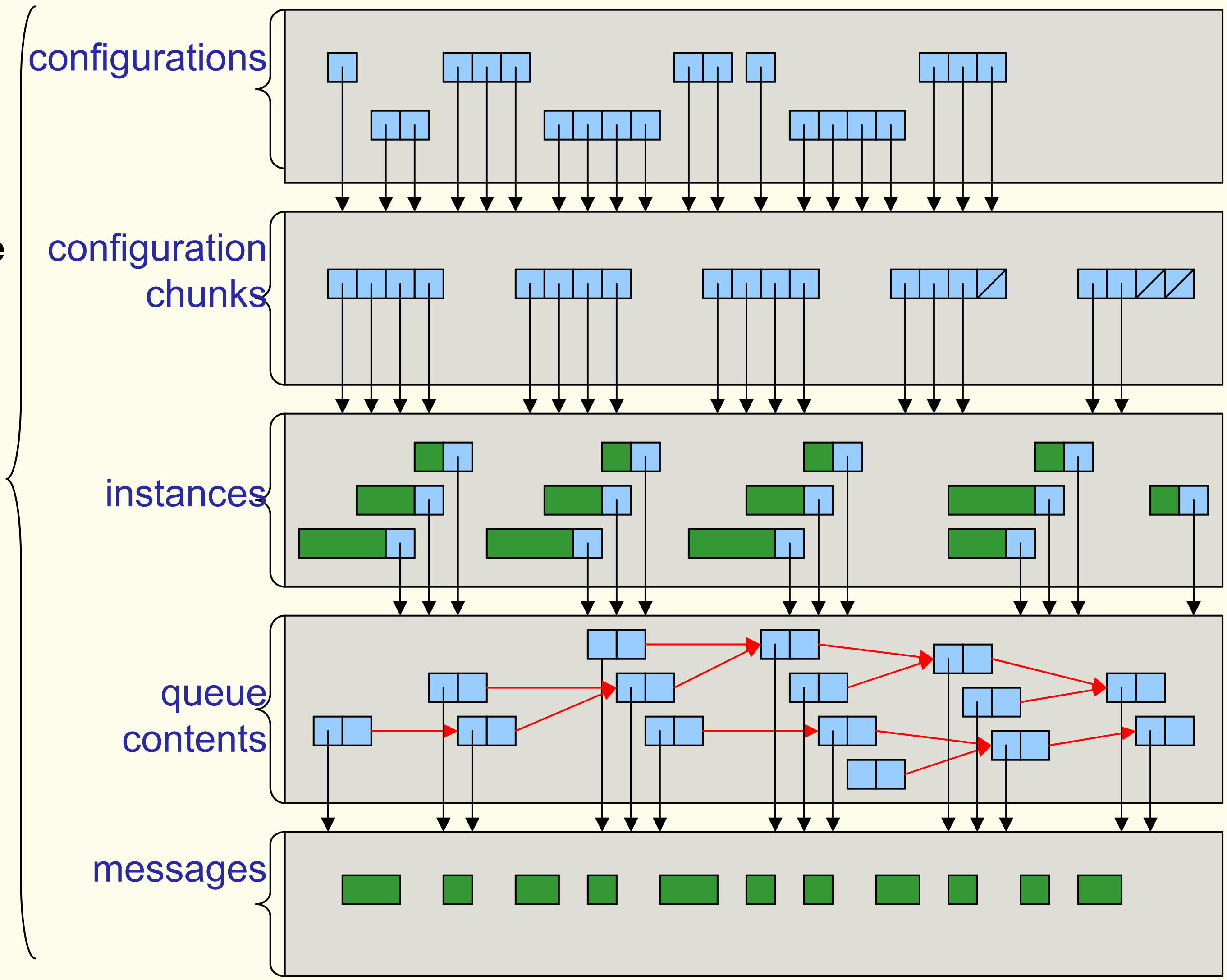


IF: state space representation

state storage is completely done by the simulator

structural representation of configurations offering maximal sharing

unique tables implemented as hash tables with collision or search trees (splay trees or 2-3 trees)



IF: representation of time

Time represented by a dedicated process instance handling:

- dynamic clock allocation (set, reset)
- representation of clock valuations
- checking time constraints (time guards)
- computation of time progress conditions w.r.t. actual deadlines
- firing time progress transitions, if enabled

Two concrete implementations are available (others can be easily added)

i) *discrete time*

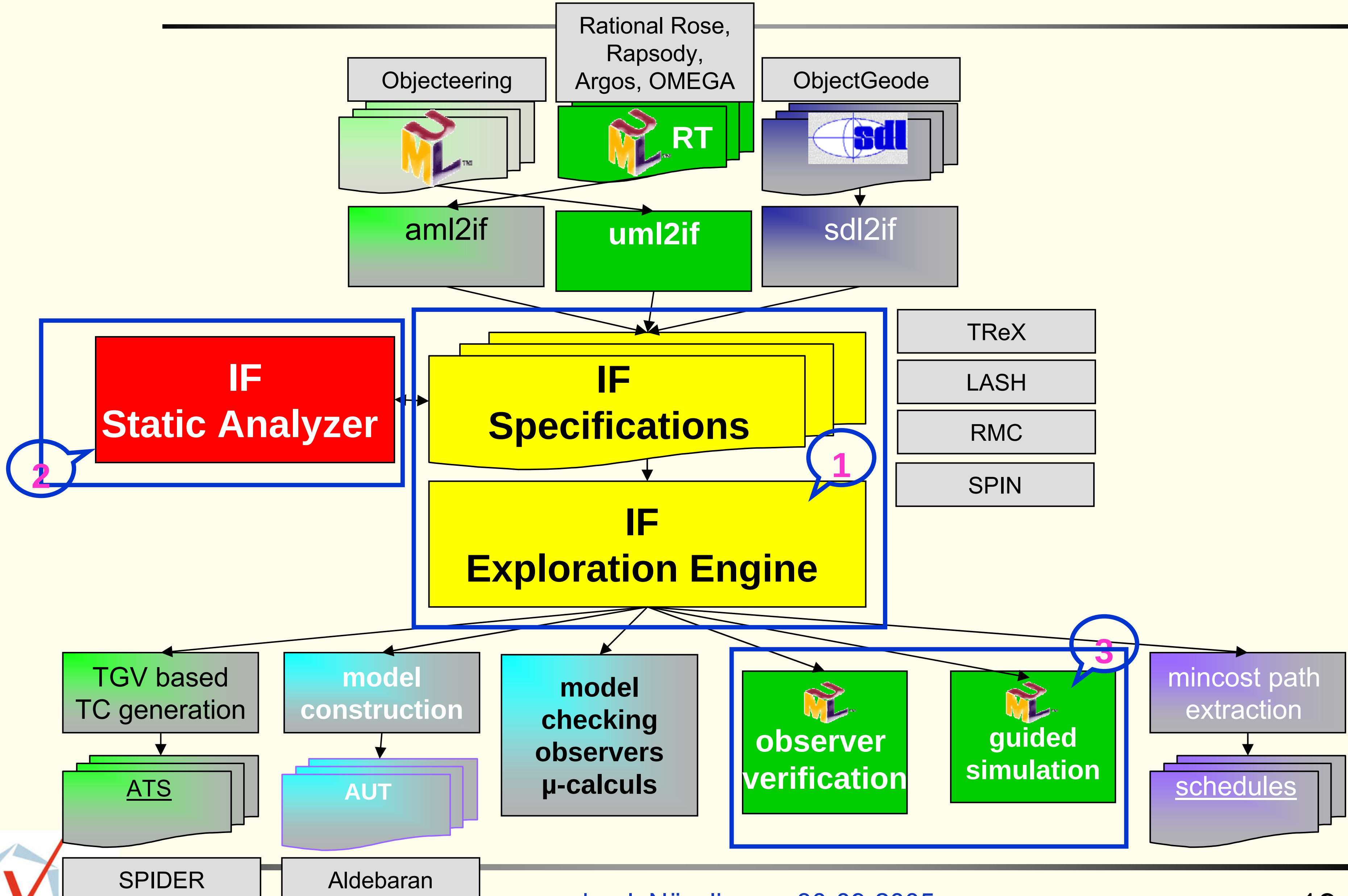
- clock valuations represented as integer values
- time progress by an explicit *tick transition* to the next deadline

ii) *symbolic time*

- clock valuations represented by (varying size) difference bound matrices (DBMs)
- time progress is implicit: State = state + time constraint
- non convex time zones may arise due to urgency: represented implicitly by unions of DBMs

KRONOS, UPPAAL

IF tool-set: overview



- **Approach**
  - source code transformations for model reduction
  - code optimization methods

- **Particular techniques implemented** so far
  - live variable analysis: remove dead variables and/or reset variables when useless in a control state
  - slicing: remove unreachable code, model elements w.r.t. a property, e.g. assumptions about the environment
  - variable abstraction: extract the relevant part after removing some variables
  - queue reduction: static analysis of queues

- Result: usually, ***impressive state space reduction***

Outline

Omega UML profile: general features

Structure

- class diagrams distinguishing active and passive classes
- structuring concepts : inheritance, associations, compositions
- architecture and components (UML 2.0-like, not available in UML 1.4)

Behavior

- state machines with action language (compatible to UML1.4 A.S.)
- operations defined by methods (action body) → polymorphic
- ***concurrency : active/passive objects*** ➔ ***activity groups***
- interactions: primitive/triggered operations, asynchronous signals

Requirements and assumptions

- ***operational : observers, Live Sequence Charts***
- declarative : *OCL* constraints on event histories

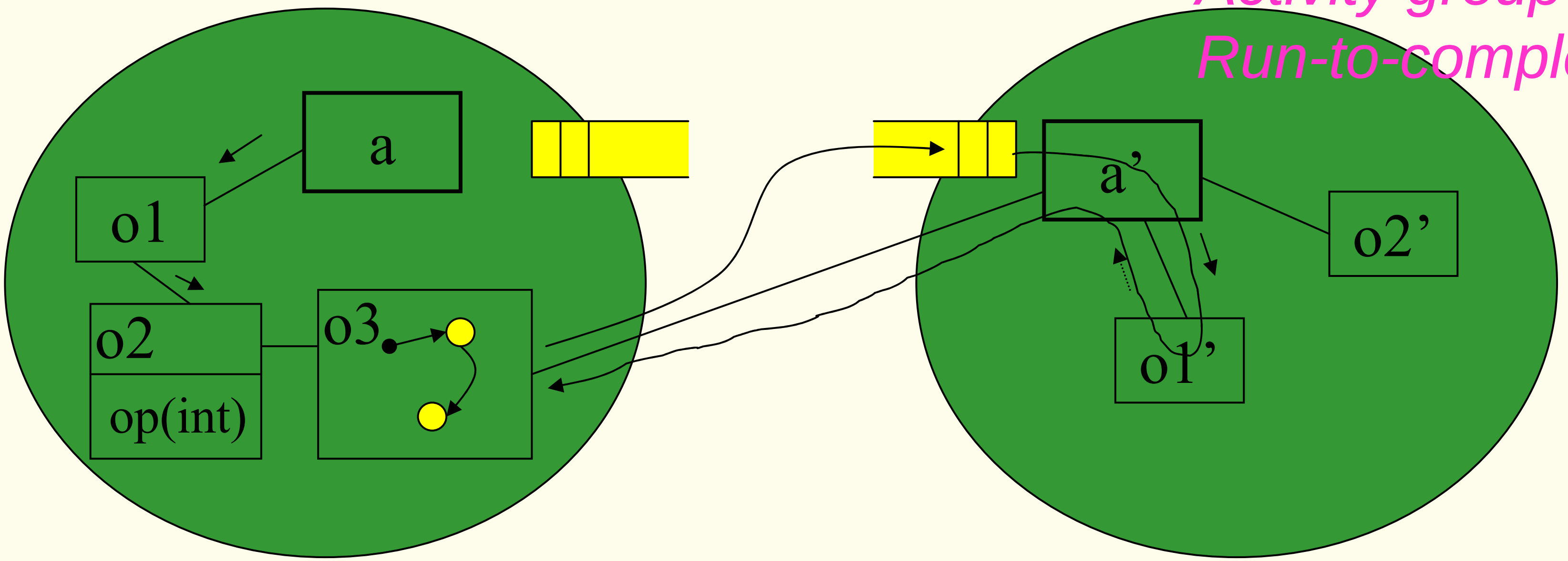
Timing constraints (in requirements, structure and design)

- ***declarative : timed events, linear (duration) constraints***
- ***imperative : timers, clocks***
- ***Deployment related***

Omega UML profile: interaction model & semantics

- active/passive objects define ***activity groups***
- interactions: primitive/triggered operations, asynchronous signals

Activity group
Run-to-completion



- [Damm, Josko, Pnueli, Votintseva 2002 & Hooman, Zwaag 2003] – based on the Rhapsody tool semantics

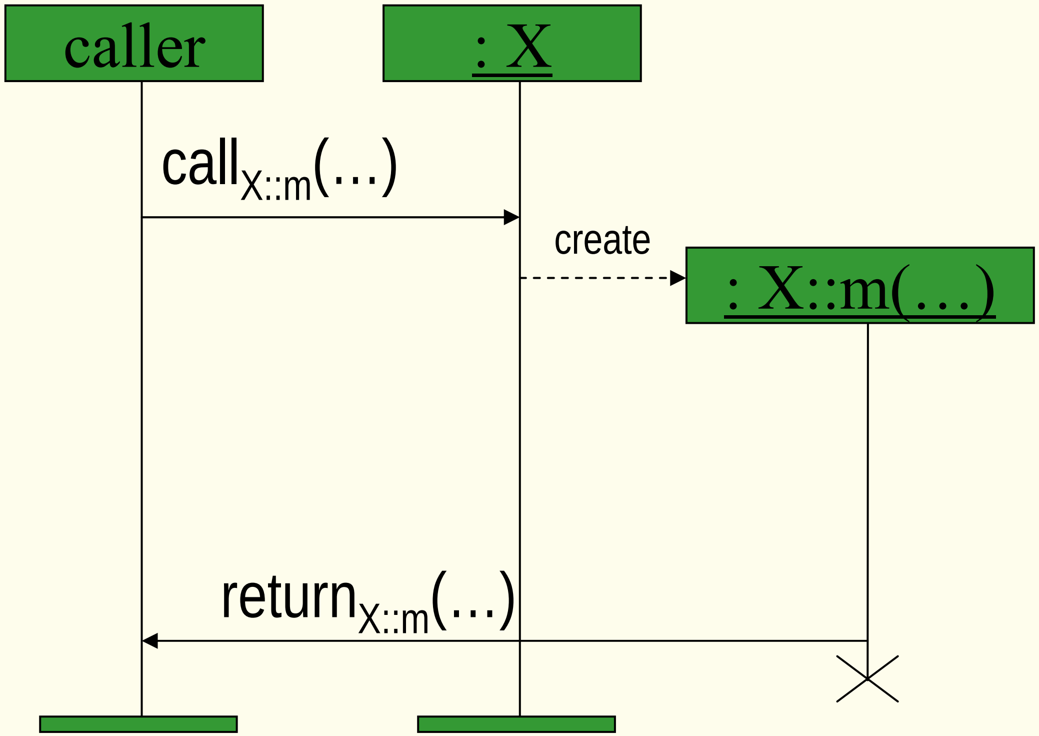
Omega UML profile: Time extensions

Compatible SPT profile and UML 2.0

- **Basics**
  - A notion of global time, *time progress non-deterministic, but controllable* by the model
  - Time primitive types: *Time*, *Duration* with operations
  - *Timed Events*: instants of occurrences of identified state changes in executions

- **Operational time access** (UML 2.0)
  - *time dependent behavior*
  - Mechanisms for measuring durations: *timers, clocks*
  - Corresponding actions: *set, reset*,…

compilation of method calls



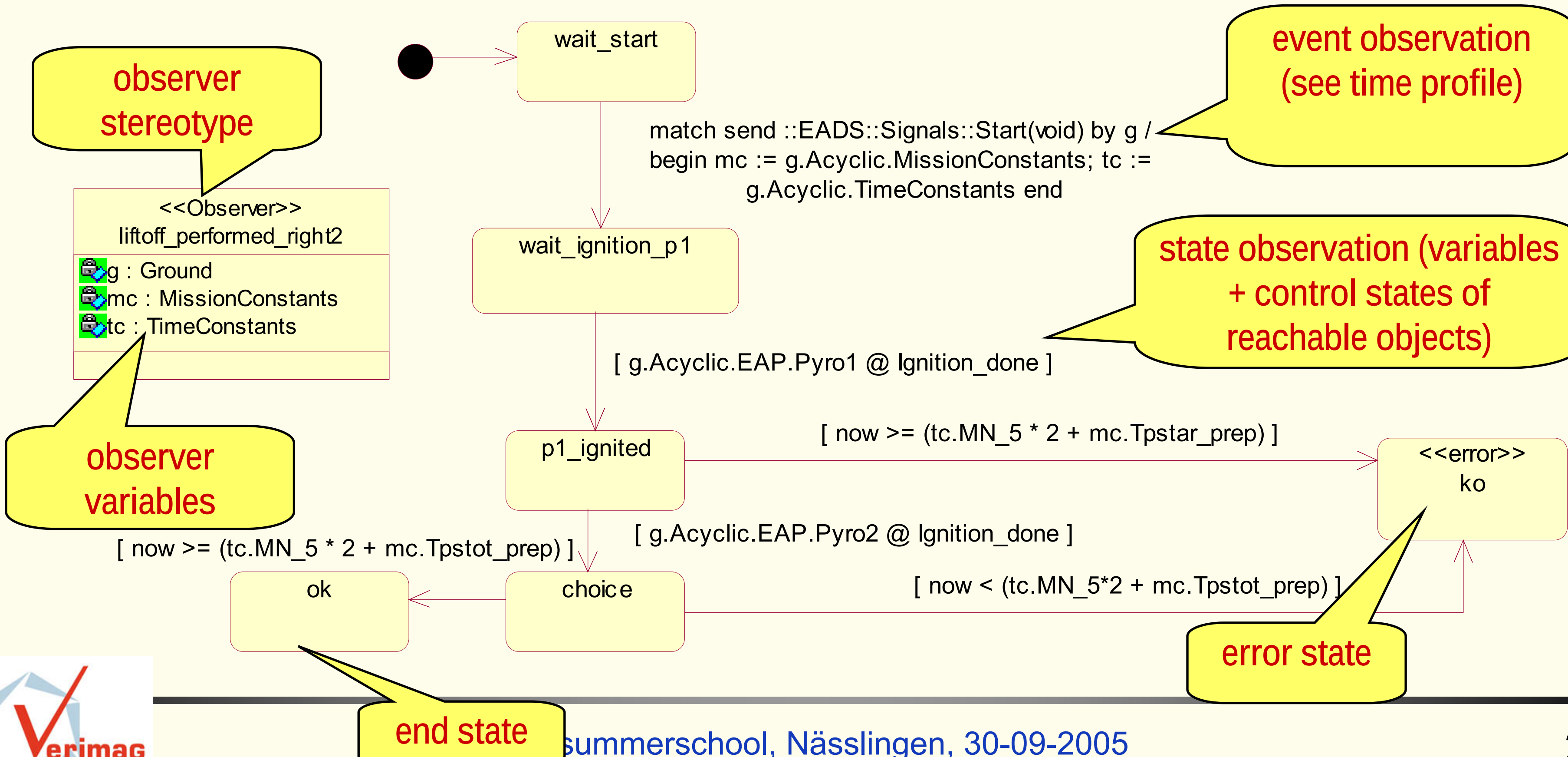
Omega UML profile: Time extensions

- **Time constraints**
  - **Constraints on durations** between *occurrences of events*
    - OCL based patterns for constraining durations between occurrences of 2 events
    - SPT like derived patterns associated with syntactic entities
      - – response time, duration of actions → deadline constraints,
      - – duration in state, delay of channel, ...
  - **Observers** with time guards

- **Scheduling**
  - Notion of resource, explicit model of architecture elements (can be reused)
  - *Execution time* of actions
  - *Priorities* for expressing scheduling policies

Omega UML profile : requirements as *observers*

- special objects monitoring the system state / events
- example (Ariane-5) : ***If the Pyro1 object enters state "Ignition_done", then the Pyro2 object shall enter the state "Ignition_done" in not less than TimeConstants.MN_5*2 + Tpstot and not more than TimeConstants.MN_5*2 + Tpstar time units.***



Omega UML profile : *observables*

- observable events
  - for signals : send, receive, accept
  - for operations : invoke, receive, accept, invokereturn, …
  - for states : entry, exit
  - for actions : start, end, start-end (for instantaneous actions)
- observable state
  - all entities reachable by navigation from already known entities (e.g. obtained from events)
  - can be stored in the observer
- observing time
  - use clocks local to an observer
  - read clocks of visible part of the model

Omega UML profile : requirements as constraints

- Define explicit events and constraints
- example (Ariane-5) : ***If the Pyro1 object enters state "Ignition_done", then the Pyro2 object shall enter the state "Ignition_done" in not less than TimeConstants.MN_5*2 + Tpstot and not more than TimeConstants.MN_5*2 + Tpstar time units.***

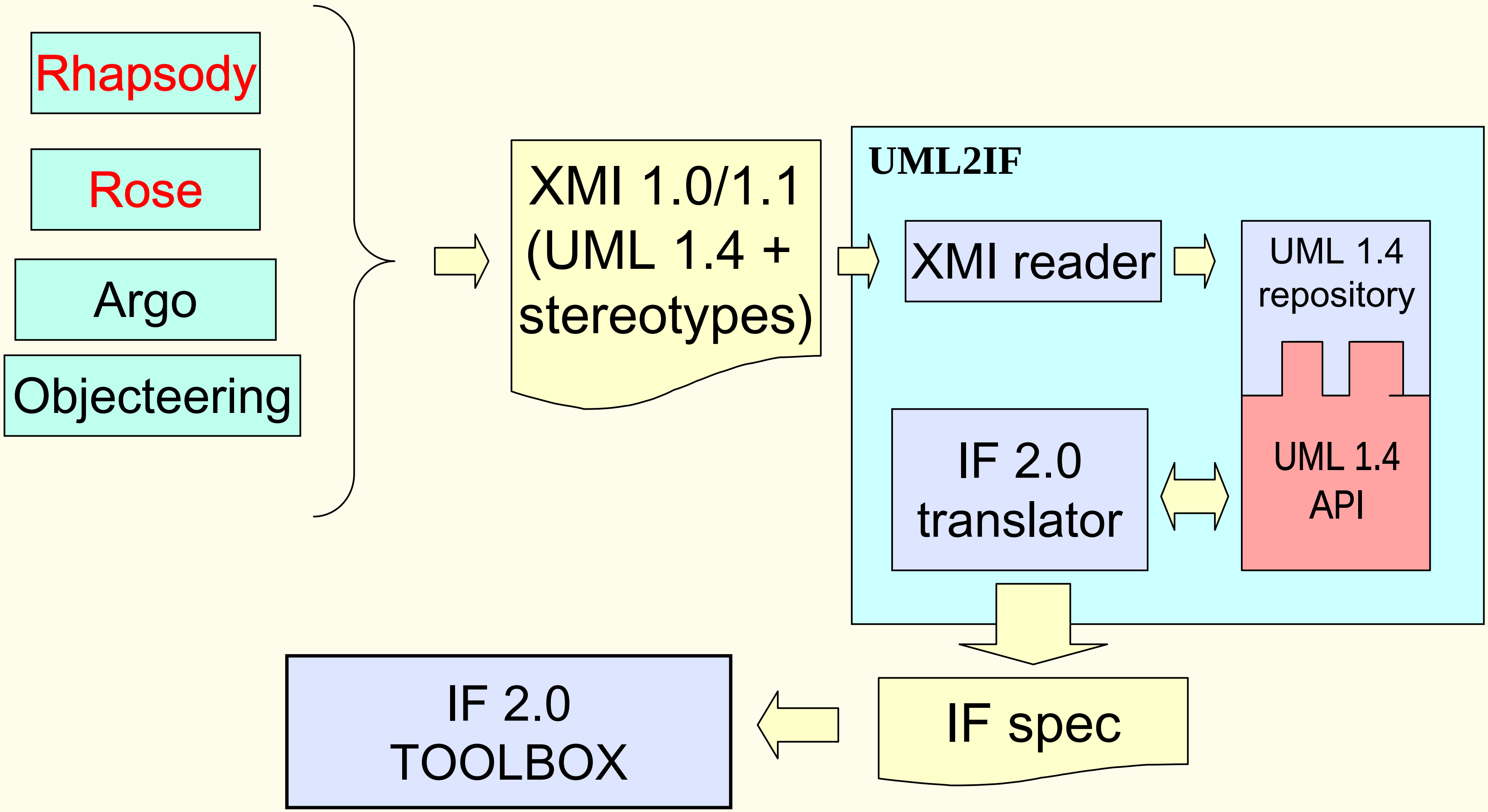
<<TimedEvent>>
IgnPyro1
p : Pyro
{ match enter Pyro @ Ignition_done by p when p = p.EAP.Pyro1 }

<<TimedEvent>>
IgnPyro2
p : Pyro
{ match enter Pyro @ Ignition_done by pwhen p = p.EAP.Pyro2 }

<<TimedAssert>>
liftoff_performed_right
i1 : IgnPyro1
i2 : IgnPyro2
{ duration(i1,i2) >= TimeConstants.MN_5*2 + Tpstot
duration(i1,i2) <= TimeConstants.MN_5*2 + Tpstar
}

Outline

IFx: overview

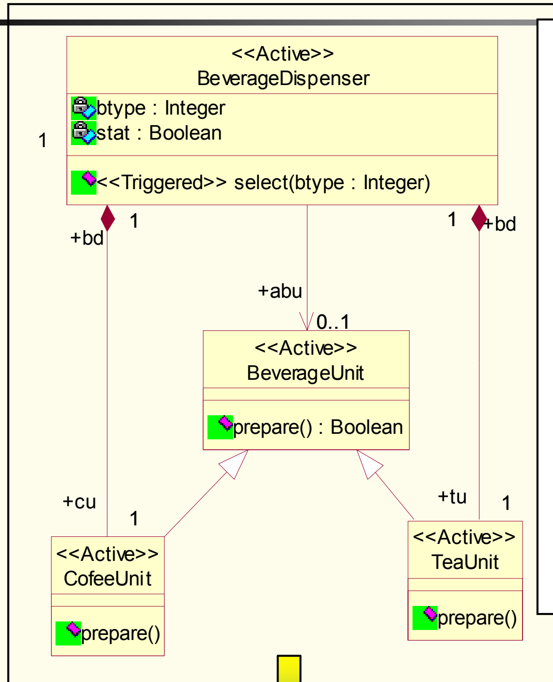


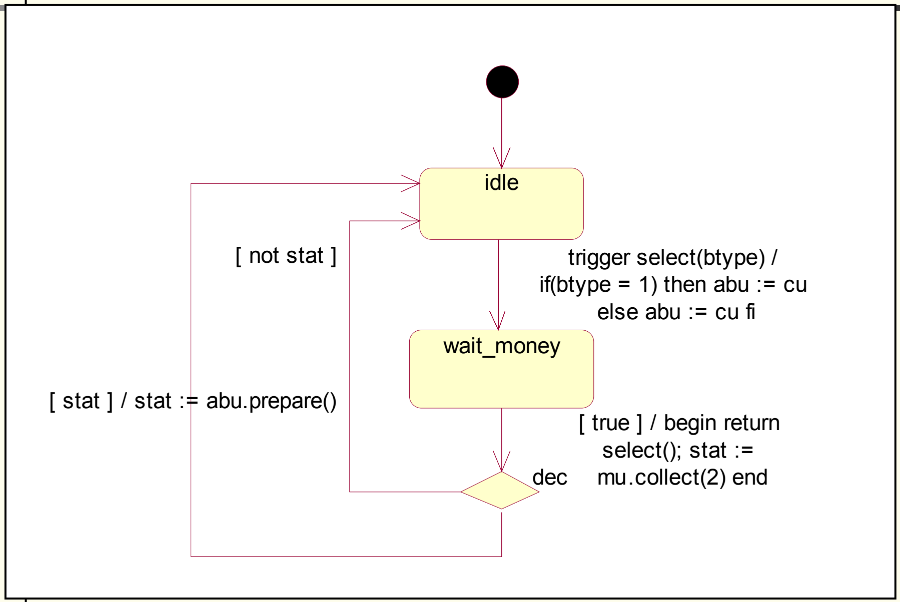
IFx: mapping UML to IF

Mapping OO concepts to (extended) communicating automata

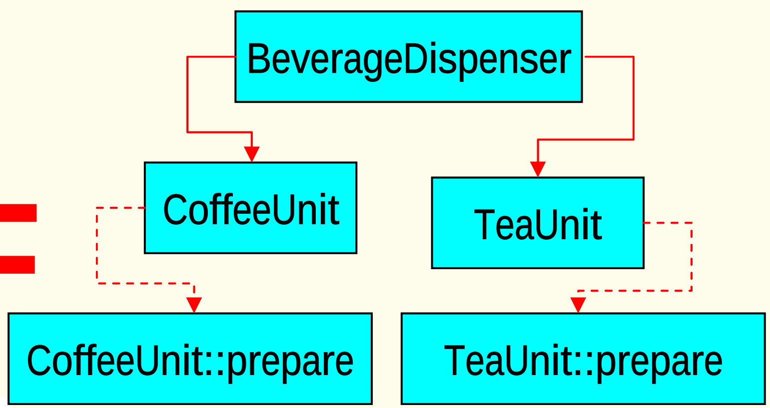
- Structure
  - class → process type
  - attributes & associations → variables
  - inheritance → replication of features
  - signals, basic data types → direct mapping
- Behavior
  - state machines (with restrictions) → IF hierarchical automata
  - action language → IF actions, automaton encoding
  - operations:
    - operation call/return → signal exchange
    - procedure activations → process creation
    - polymorphism → untyped PIDs
    - dynamic binding → destination object automaton determines the executed procedure
- Observers and events: direct mapping

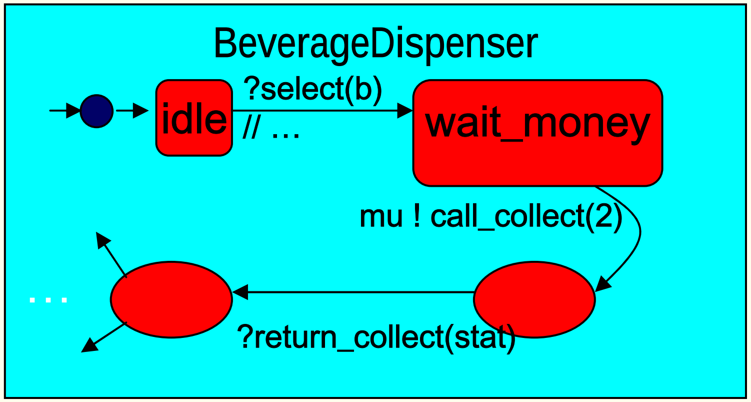
IFx: example of mapping



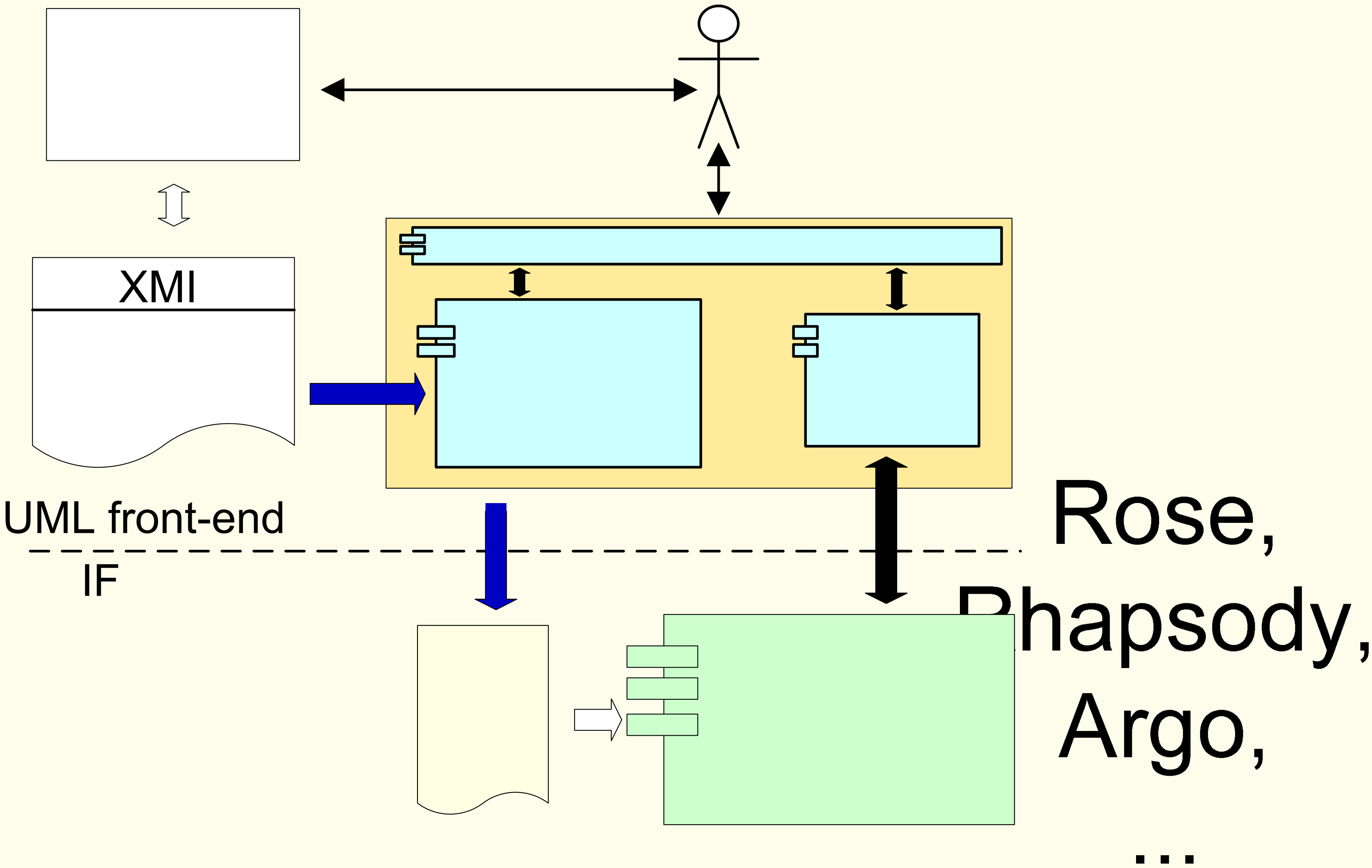


IF



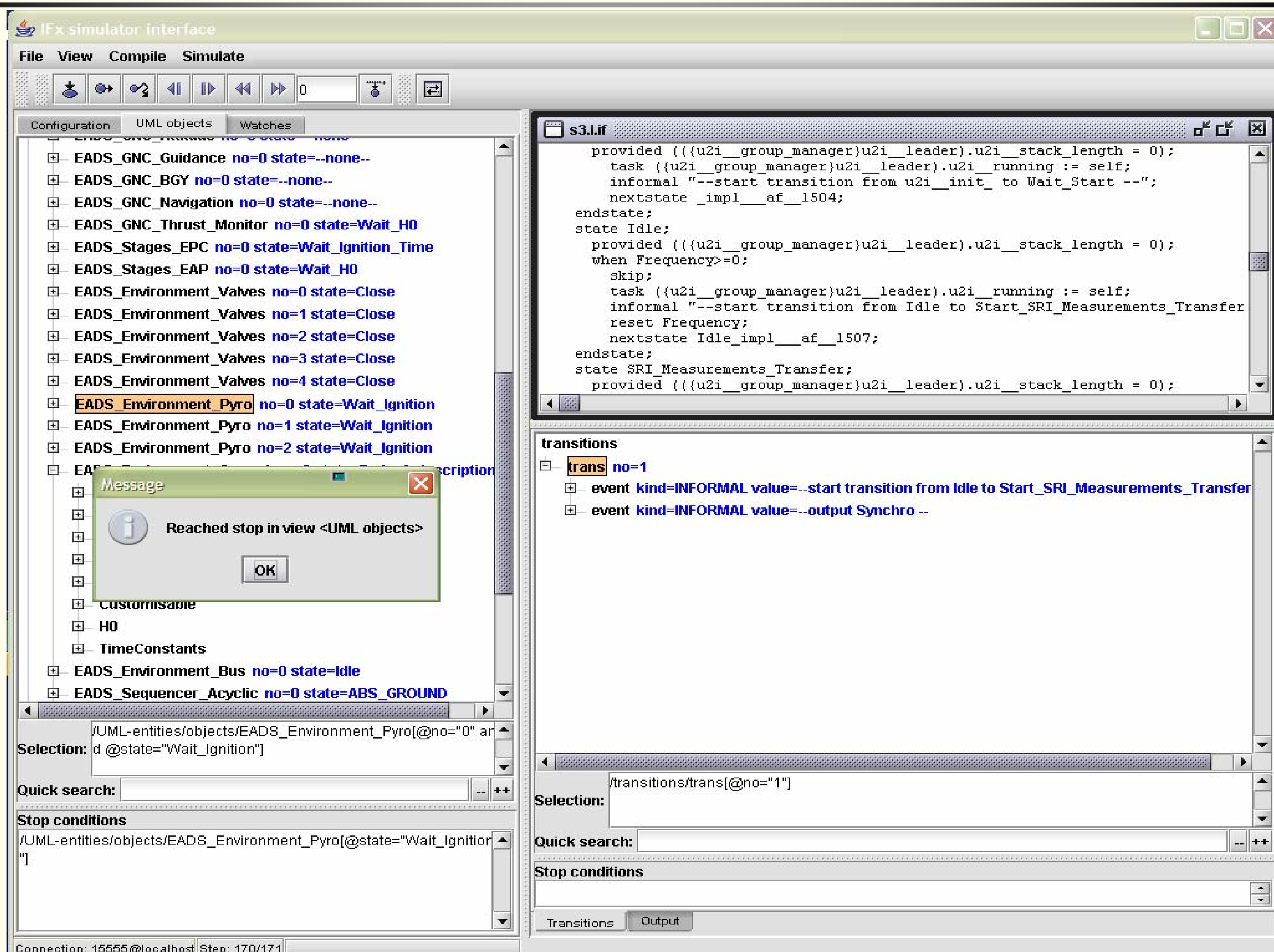


IFx: global architecture



IFx: simulation/verification interface

- user friendly simulation
  - rewind/reply
  - conditional breakpoints
  - …
- customizable presentation of results for UML users



Outline

IFx: case studies

Ariane-5 flight program (together with EADS) – Rational Rose

- statically validate the well formedness of the model wrt the Omega profile,
- 9 safety properties of the flight regulation and configuration components,
- analyzed the schedulability of the cyclic / acyclic components under the assumption of fixed priority preemptive scheduling policy,
- safety properties concerning bus read/write access under this policy

MARS bus monitor (together with NLR) – I-Logix Rhapsody

- static validation
- proved 4 safety properties concerning the correctness of the MessageReceiver,
- discover reactivity limits of the MessageReceiver and to fine-tune its behavior in order to improve reactivity.

Sensor Voting (together with IAI) – Rational Rose

- static validation
- proved 4 safety properties concerning the timing of data acquiring by the three Sensors: end-to-end duration, duration between consecutive reads, etc.

A depannage service specification (done FT) – Rational Rose and IF

- showed service level timing properties

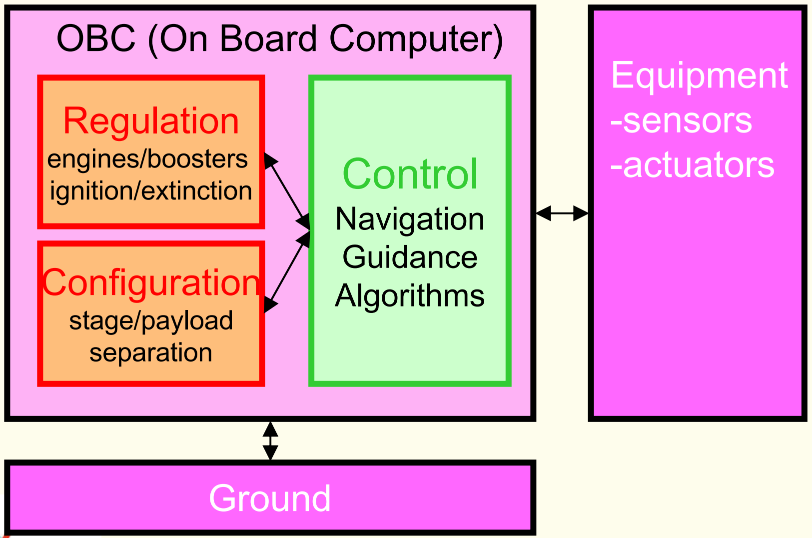
Ariane 5 flight program

Joint work with EADS SPACE Transportation

flight program specification

built by reverse engineering by EADS
high level, non-deterministic, abstracts the whole program as a OMEGA UML model

23 classes, 27 runtime objects
~7000 lines of IF code



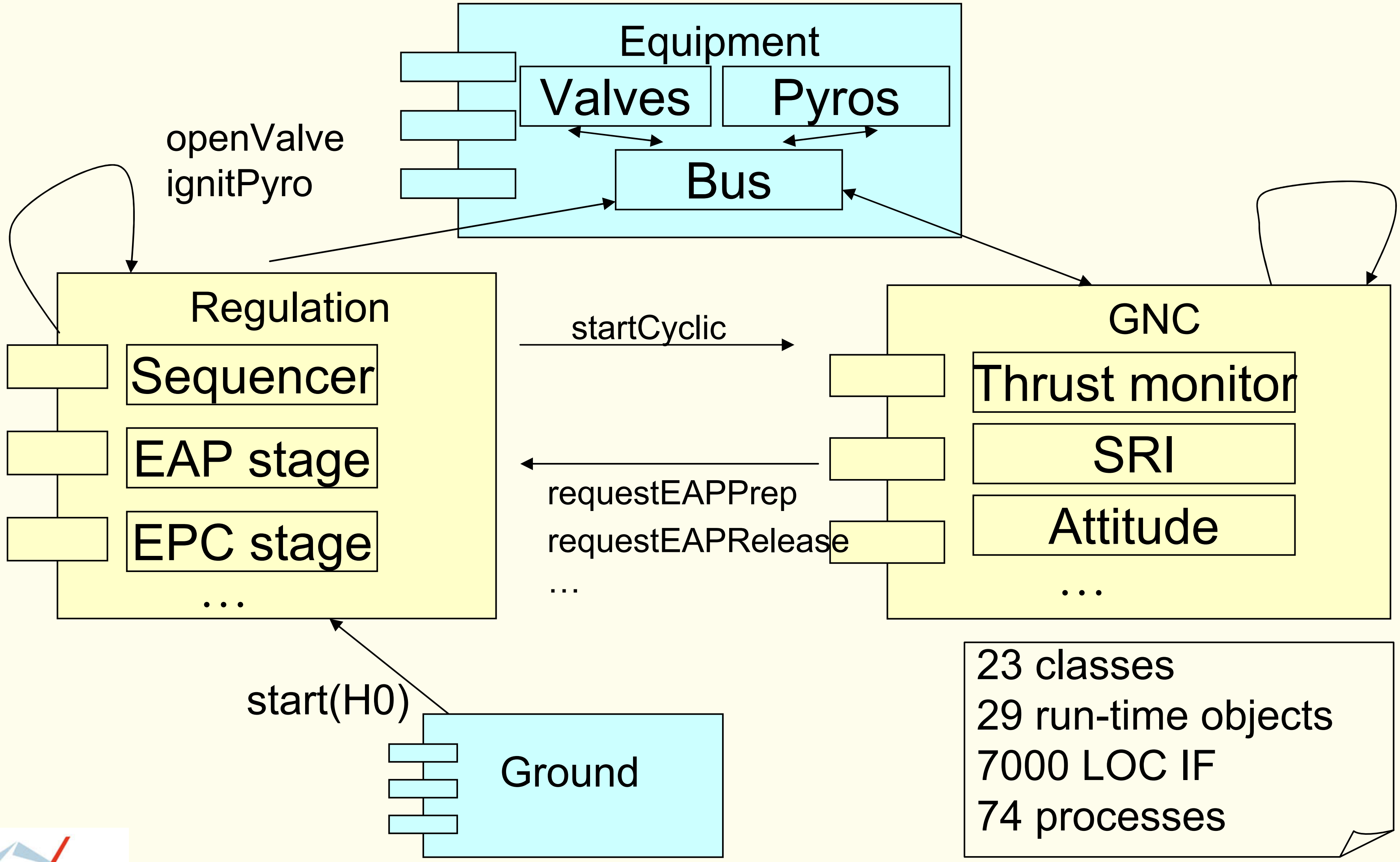
flight program requirements

General requirements
- no deadlock, no timelock
- no implicit signal consumption

Overall system requirements
- flight phase order
- stop sequence order

Local requirements of components
- activation signals arrive in some predefined time interval

Ariane 5: detailed architecture



Ariane 5: techniques applied

translation

- Mapping of complete UML specification into IF with uml2if
- fixed static errors (typing, naming)

model exploration

random or guided simulation

several inconsistencies found

static analysis

live variable analysis

20% of all variables are dead in each state

model generation

partial order reduction needed

Full state space cannot be constructed

use some conservative abstractions

model checking

9 safety properties about correct sequencing of sub-phases

- concern only the acyclic part
- abstraction of GNC part

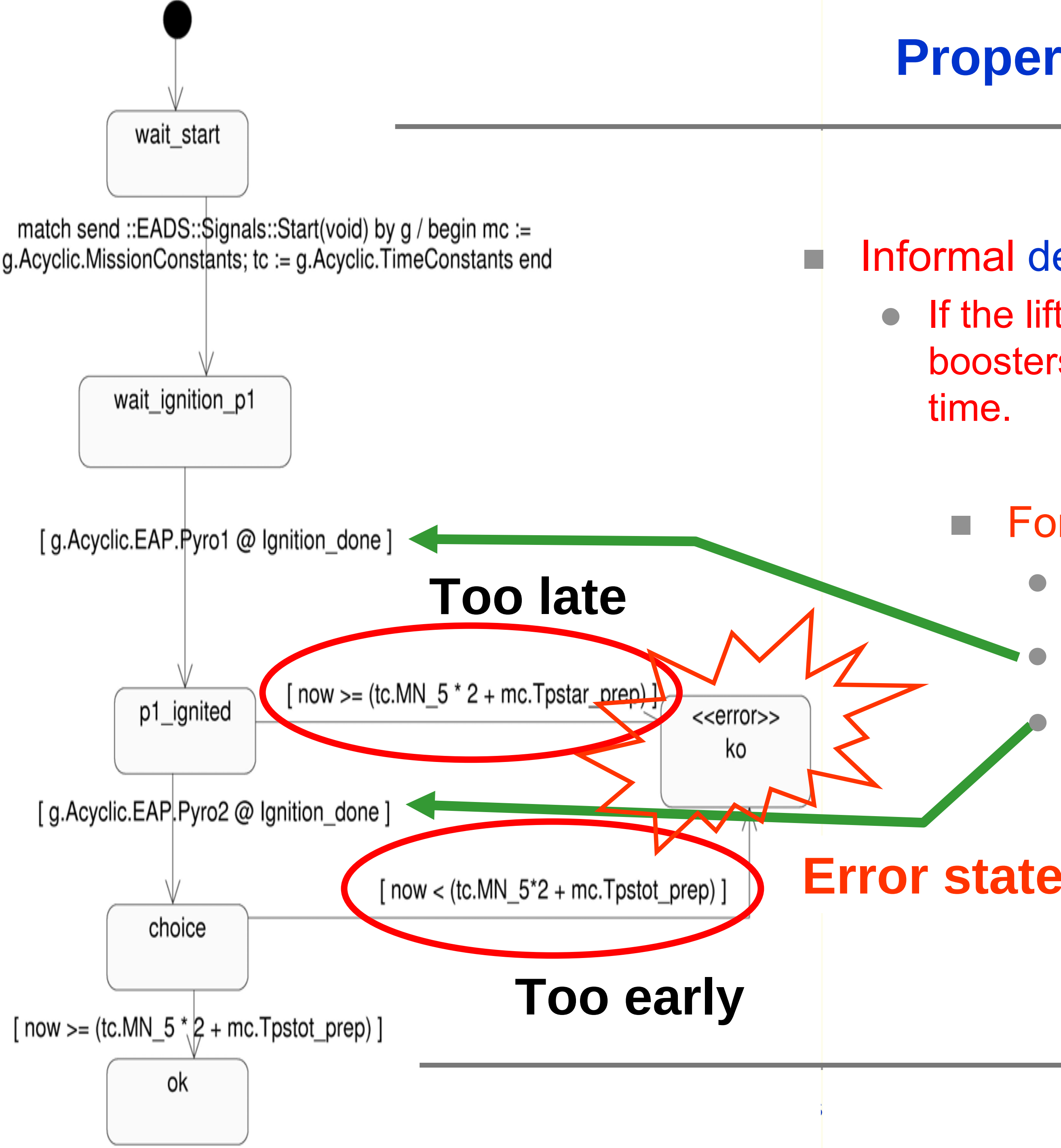
schedulability analysis

- concerns the entire system
- abstraction of mission duration

Ariane 5: safety properties

- *9 safety properties about the correct sequencing of sub-phases:*
  - *between any two commands sent by the flight program to the valves there should elapse at least 50ms*
  - *a valve should not receive signal Open while in state Open, nor signal Close while in state Closed.*
  - *if some instance of class Valve fails to open (i.e. enters the state Failed Open) then*
    - No instance of the Pyro class reaches the state Ignition done.
    - All instances of class Valve shall reach one of the states Failed Close or Close after at most 2 seconds since the initial valve failure.
    - The events EAP Preparation and EAP Release are never emitted.
  - *…*

Property example (timed)



- Informal description
  - If the liftoff is performed, the boosters shall be released at due time.

- Formal description
  - Using an observer
  - Liftoff = pyro1.ignition
  - Boosters release = pyro2.ignition

A typical scenario for this category of systems

- pre-emptive fixed priority scheduling
  - one processor
  - three tasks :

Regulation	NC	Guidance
- sporadic	- periodic 72ms	- periodic 576ms
- E = 2-5ms (func)	- E = 36-64ms (f)	- E = ? ms
- priority : 0	- priority : 1	- priority : 2

Standard scheduling analaysis:
? < 64ms

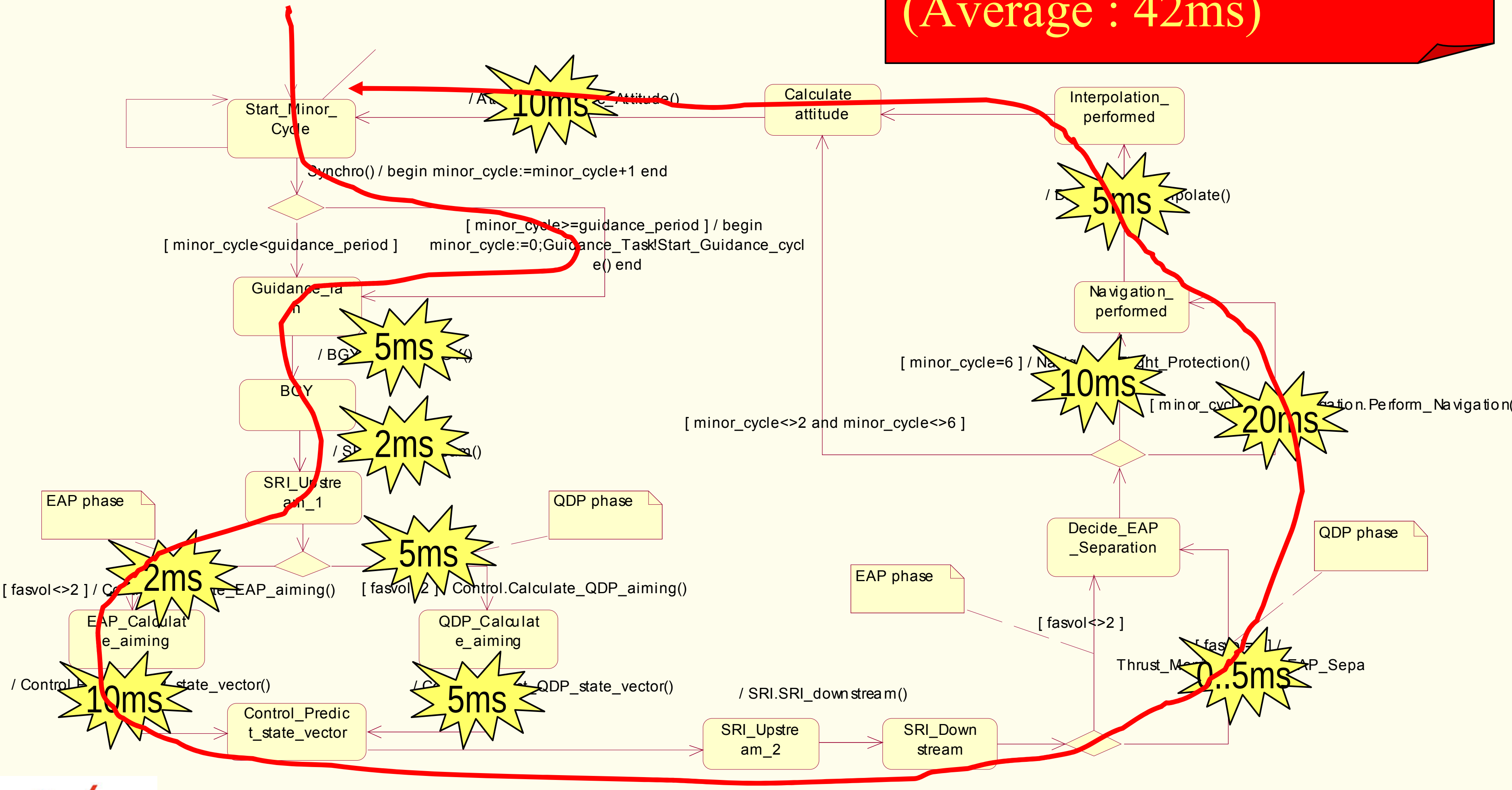
Not enough

- scheduling goals :
  - NC and Guidance cycles are respected
  - Bus read / writes do not conflict

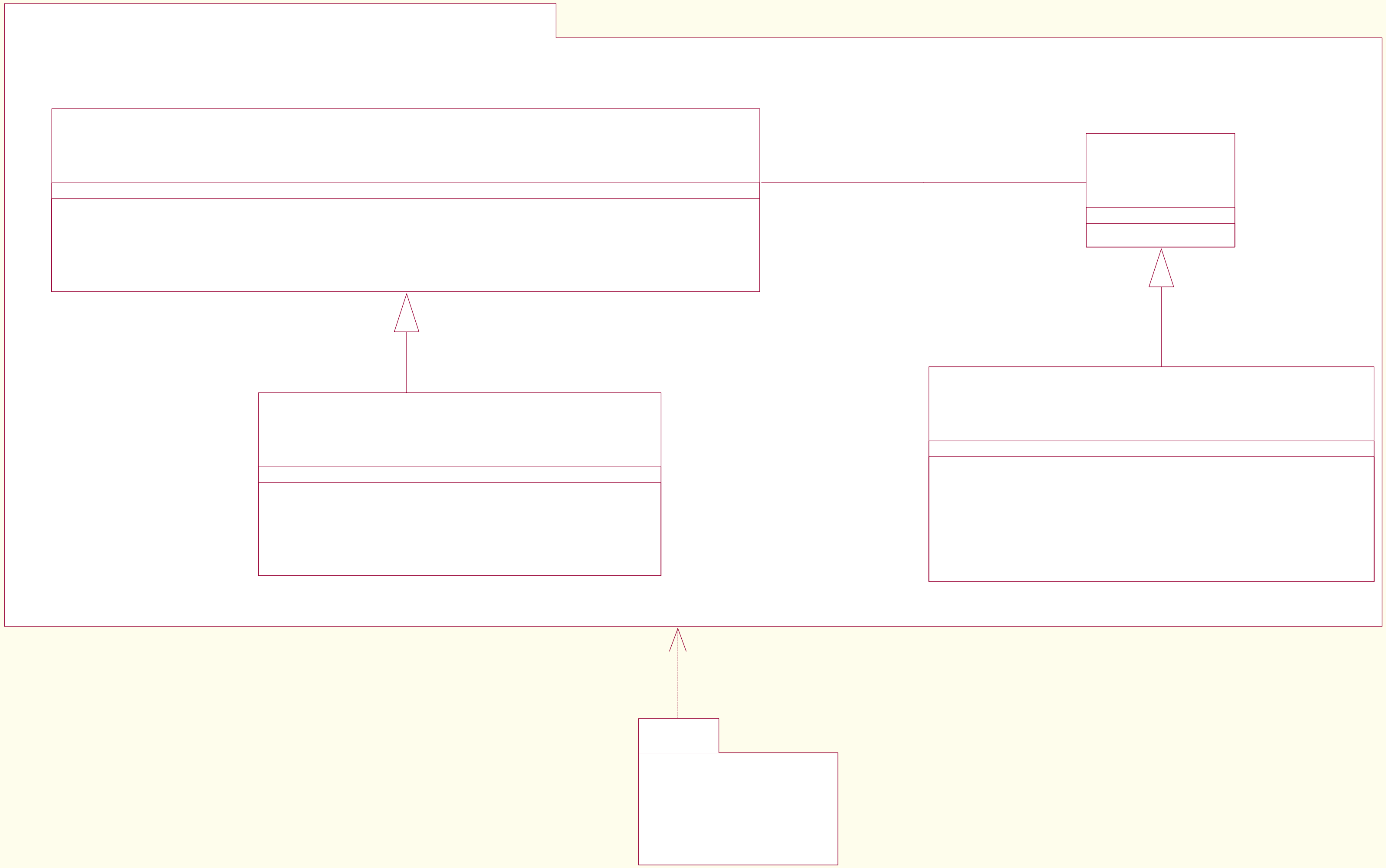
=> modeled by (simple) safety observers

Ariane 5: why we cannot abstract functionality

Worst case : 64ms (/72 !)
(Average : 42ms)



Model of multitasking in OMEGA UML: an explicit model



- Definition of task priority

```
begin
    theTask := new::CPU::FPPSTask::FPPSTask(1, Acyclic.Ground.CPU)
end
```

This task has the first priority

- Definition of CPU consumption for each function

```
begin
    Cyclics.theTask.exec(5)
end
```

This action consumes 5 units of time

Ariane 5: results of scheduling analysis

=> scheduling analysis reduced to verification

Regulation
- sporadic
- E = 2-5ms (func)
- priority : 0

NC
- periodic 72ms
- E = 37-64ms (f)
- priority : 1

Guidance
- periodic 576ms
- E = 230 ms
- priority : 2

- **Difficulty:**

  combination of long / short cycles and mission phases

  => state explosion

- **2 Solutions:**

  → over-approximate Regulation part (non-deterministic) leads to a more pessimistic resource estimation for Guidance

  → drastic shortening of mission duration is an exact under-approximation and gives more precise results

Conclusions: Omega UML profile

- Modeling reactive systems
  - Good expressivity (similar to Rhapsody, Room, SDL,…)
  - Well defined *semantics* with some non determinism with a sufficient granularity for adding timing
- Timing and deployment
  - Direct expression of time constraints as in existing frameworks
  - General naming scheme for *events* for the expression of semantic level granularity (SPT defines names for pairs of events)
  - A framework for defining semantics of any SPT like RT profile
  - Architecture (processors, buses,…) defined by special components and mappings with a few specific concepts
- To be done
  - *Explicit component models*, where interfaces (required, provided or mixed) can be modeled by observers
  - Architecture and deployment related concepts not settled, need more experimentation

Conclusions: IF language and tools

IF language:

- Few concepts, but rich enough for efficient state space reduction and abstraction in the context of state space exploration based validation
- Multiple observers reacting to the same event pose semantic problems ➔ need for a composition framework

Mapping UML to IF: designed for flexibility (anticipate semantic variations):

- Can hardwire stricter or less strict concurrency constraints, can handle asynchronous calls,

IF validation tools:

- Very flexible and positive results on all case studies
- more specialized abstractions would be useful
- Need for verification of observers ➔ need for a framework for composition

IF UML user interface:

- Very helpful, exploits information in XMI
- more powerful interfaces need information of case tool internal APIs

Questions ?

Advertisements:

- MARTES workshop with Models 2005, October 4, 2005 (http://www.martes.org)

- IF webpage: http://www-if.imag.fr/ (there are more tutorials)