CADP 2006 from a Model Driven Perspective

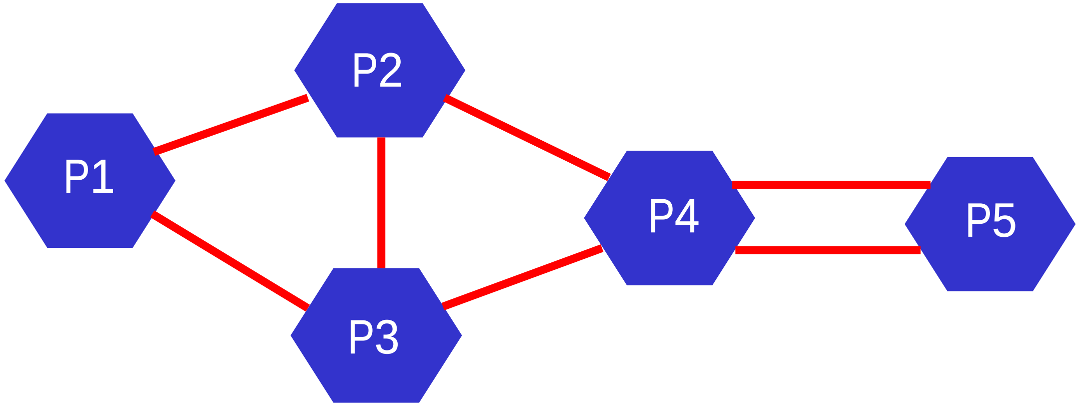
Hubert Garavel

and the VASY team

INRIA Rhône-Alpes

http://www.inrialpes.fr/vasy

Asynchronous systems



- several processes (or tasks, agents, entities)
- that execute concurrently (in parallel)
- at different speeds (no central clock)
- with message-passing communications
- without shared memory (unless explicitly modelled)
- with unspecified communication latencies

The CADP toolbox

- A verification toolbox for asynchronous systems
- A modular, extensible architecture
- Generic software components for verification
- Main functionalities:
  - several input languages
  - step-by-step simulation
  - C code generation - rapid prototyping
  - verification
  - test generation
  - performance evaluation

Some figures about CADP

- A comprehensive toolset

  38 tools, 3 code libraries

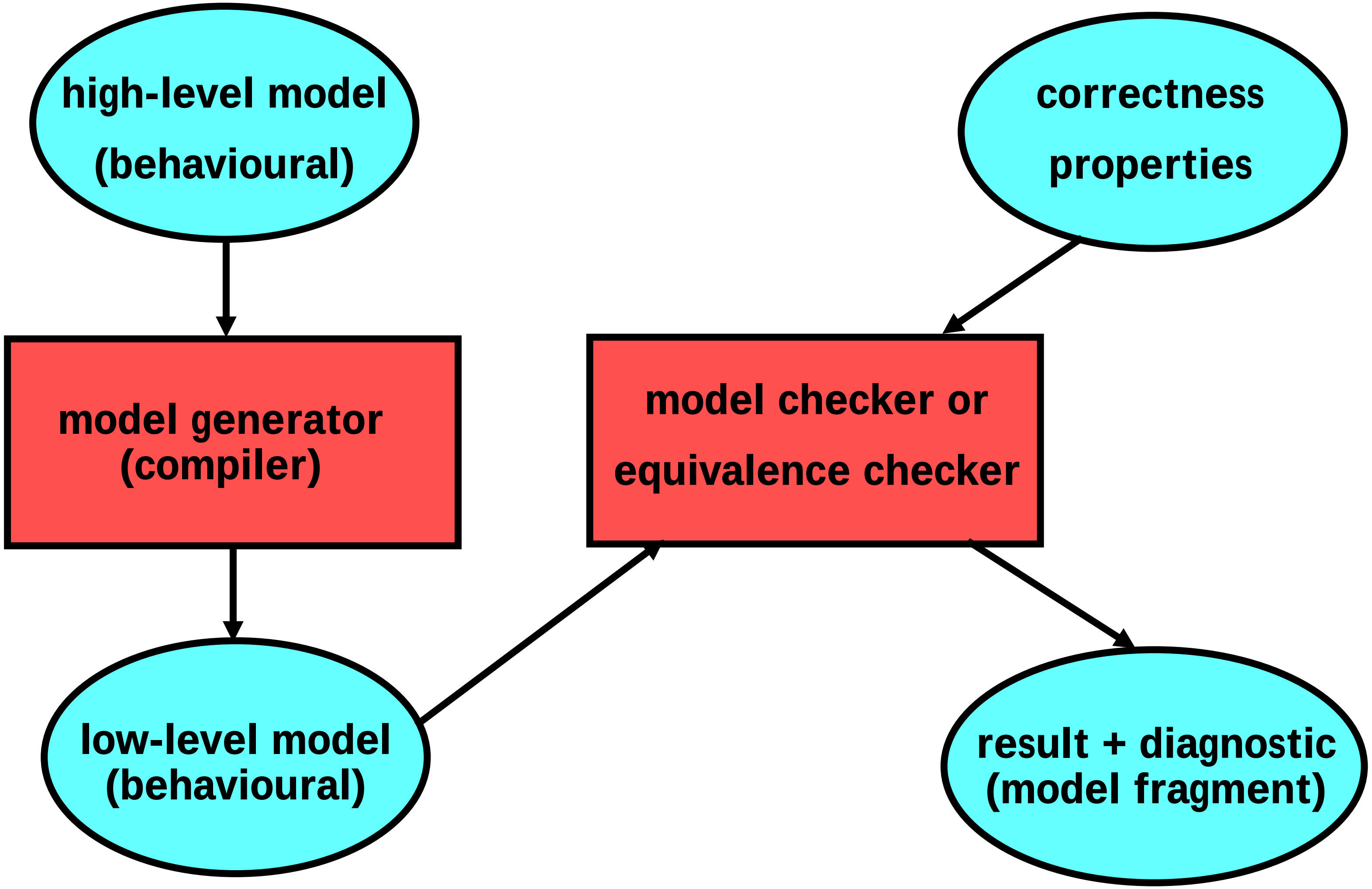
- Four platforms supported

  PC/Linux, Sparc/Solaris, PC/Windows, PowerPC/MacOS X

- International dissemination
  - license agreements signed with 358 organizations
  - since Jan. 1st, 2006: licenses granted to ~800 machines
- Many applications
  - 88 case-studies accomplished using CADP
  - 24 research tools connected to CADP
  - 28 university lectures based on CADP since 2002

Model transformations in CADP

(Standard) model-based verification



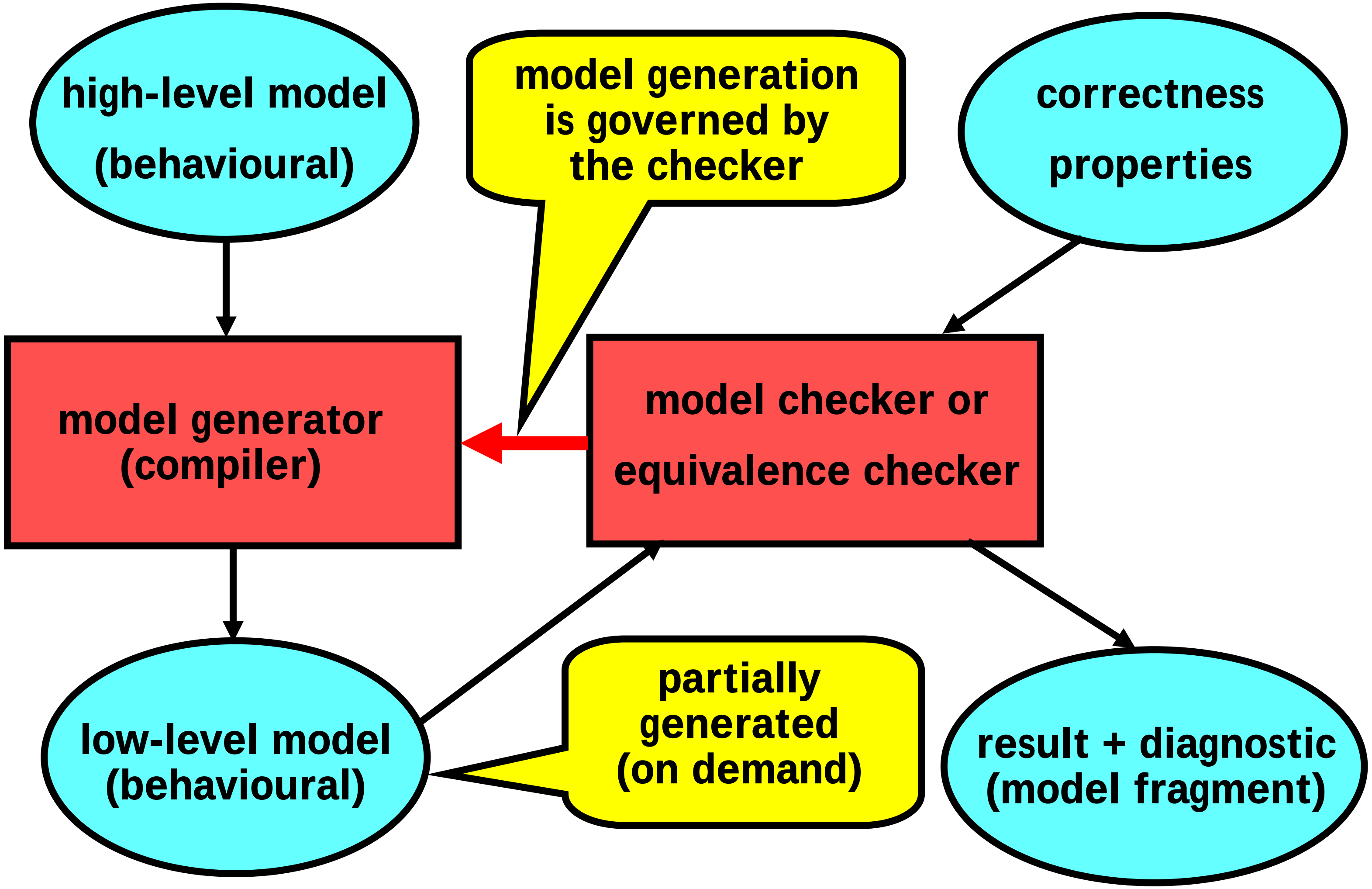
Standard vs. refined verification

- Standard model-based verification fits well within the model-driven framework

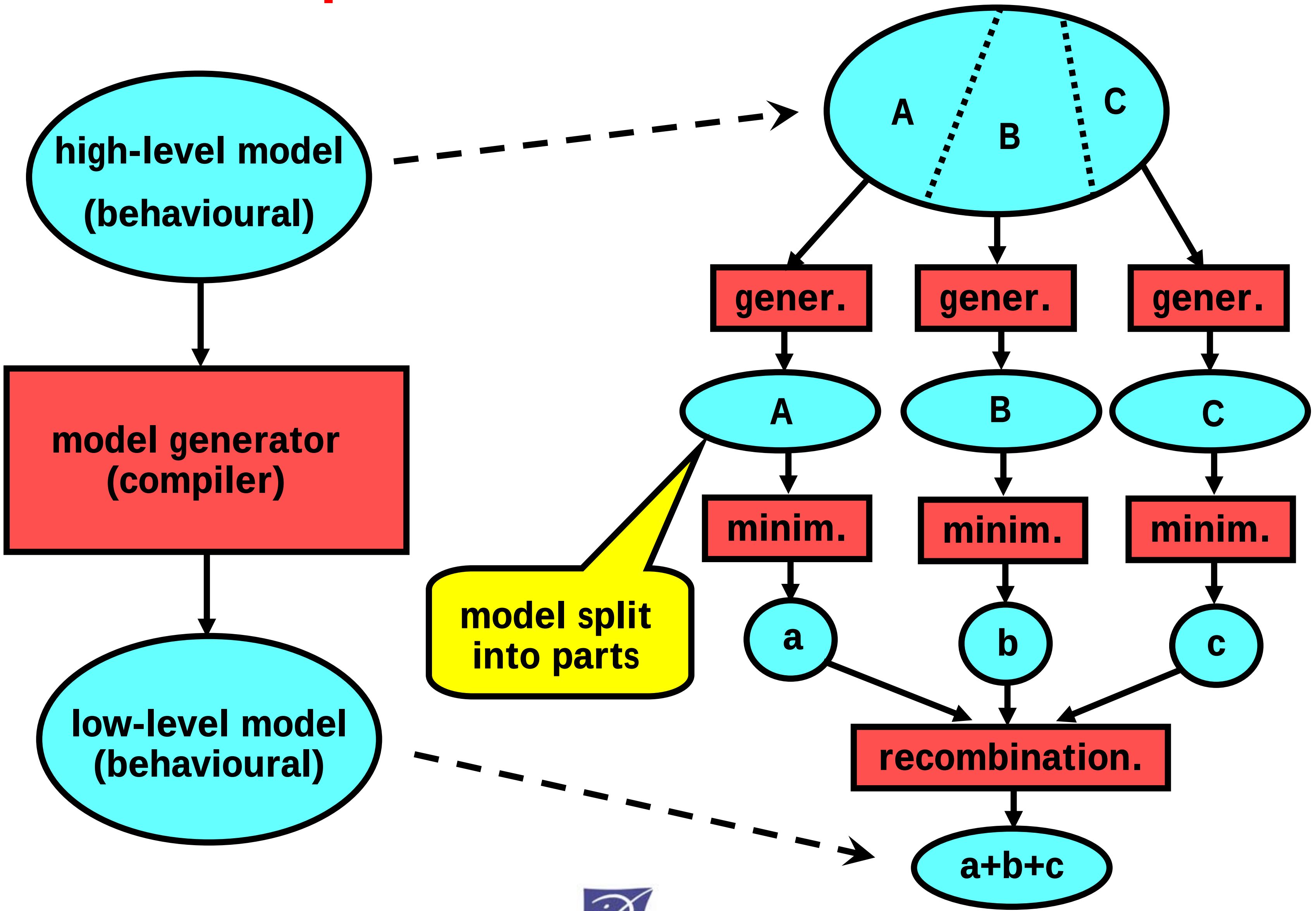
- But there exist refined techniques:
  - Partial / on-the-fly verification
  - Compositional verification
  - Distributed verification

- Are refined techniques compatible with the current model-driven approaches?

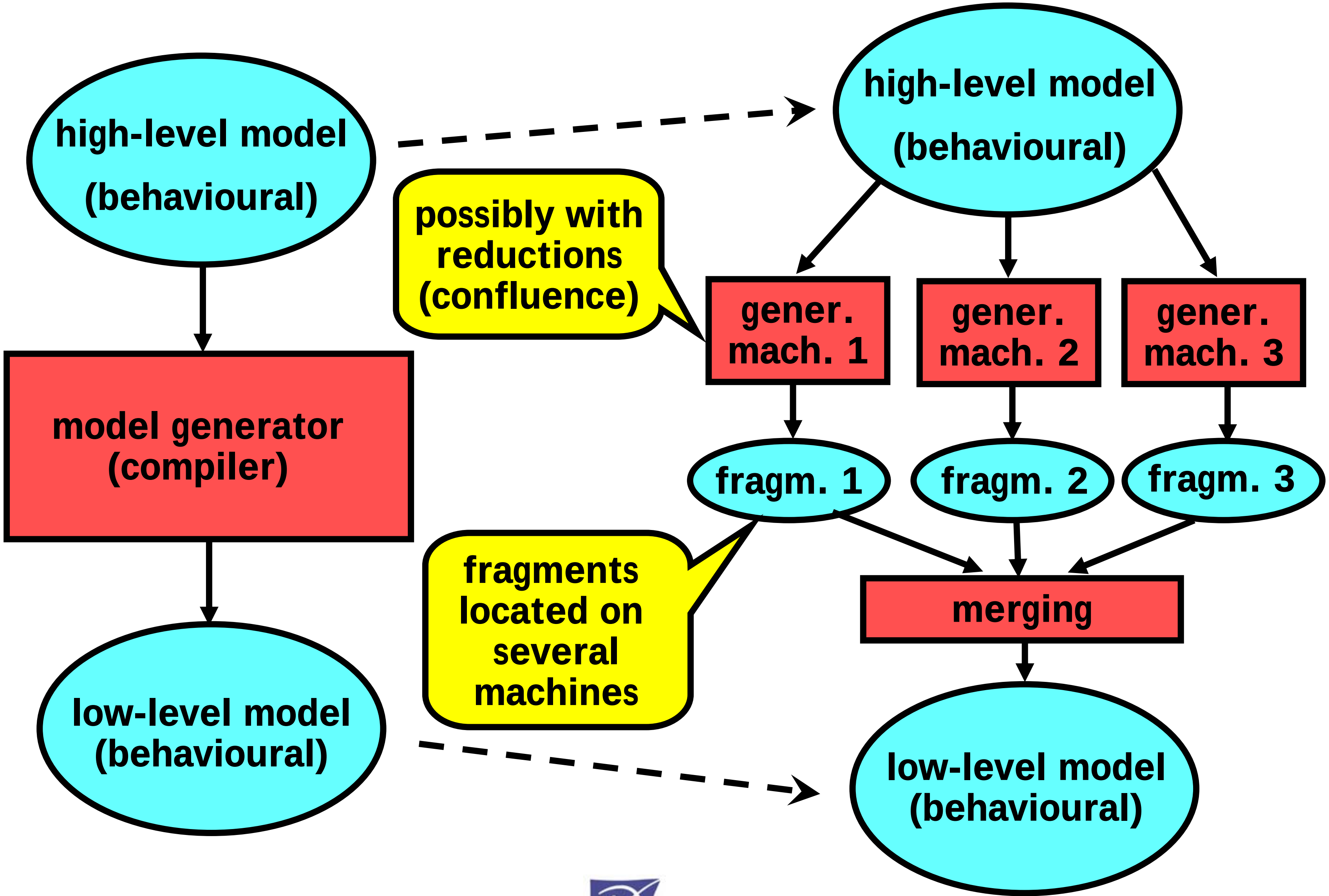
1. Partial / on-the-fly verification



2. Compositional verification



3. Distributed verification



Models supported by CADP

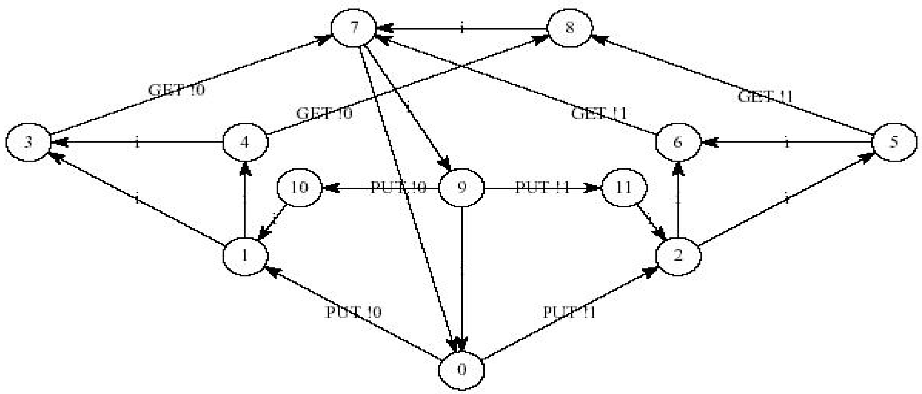
Seven models

1. Labelled Transition Systems
2. Markov models
3. Communicating automata
4. Process calculi - LOTOS
5. Modal mu-calculus formulas
6. Boolean equation systems
7. Verification scenarios

1. Labelled Transition Systems

Labelled Transition Systems

- LTS: a low-level model to describe behaviors (*state spaces)*
- LTS = graph
  - edges labelled by "actions" (or "events", or "labels") containing port names and typed data
  - no information attached to states
  - except the identification of one initial state

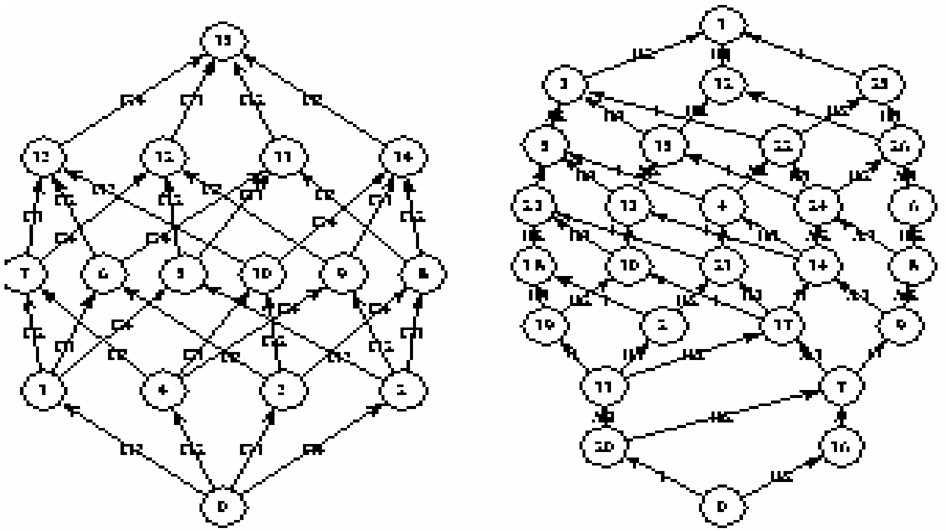


... Labelled Transition Systems

- LTS can be:
  - written by hand (only the small ones)
  - or generated automatically
- LTS can be very large (billions of states and transitions) → state explosion problem
- CADP provides four representations for LTS

1.a. Explicit LTS model

- LTS is given by its list of states and transitions
- CADP provides a compact file format: BCG
- Many CADP libraries and tools for BCG: bcg_draw, bcg_edit, bcg_graph, bcg_io, bcg_info, bcg_labels, bcg_min...



1.b. Traces model

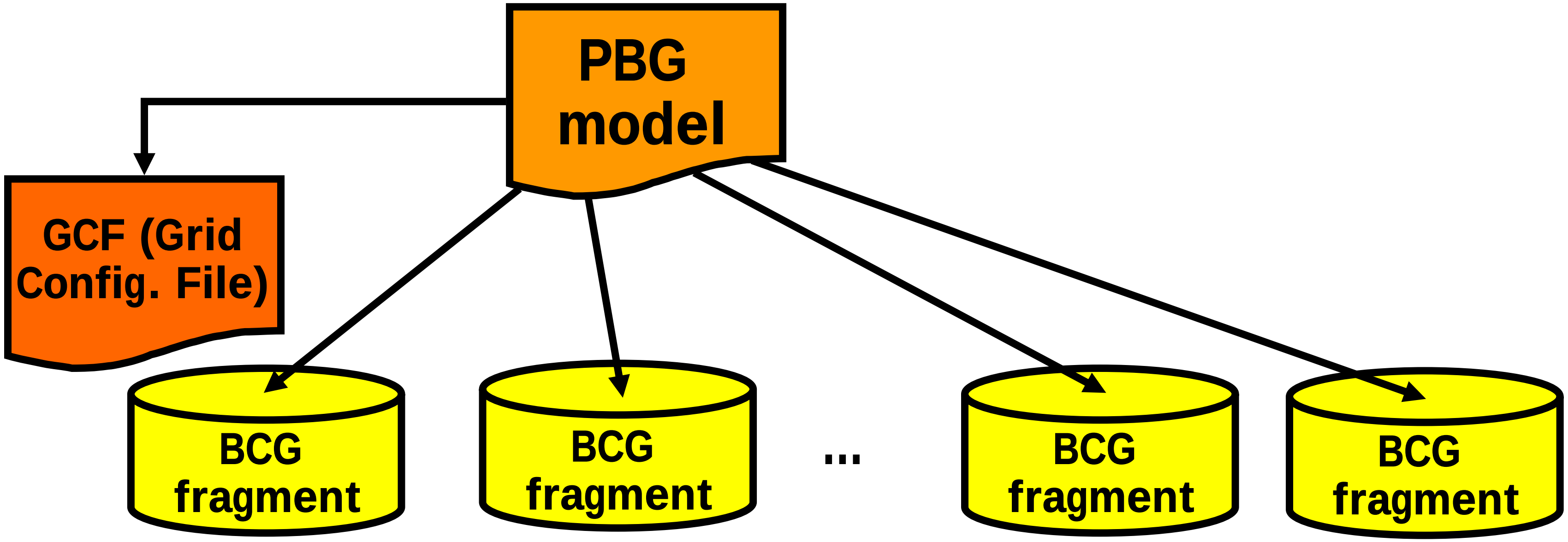
- Event traces (log files) obtained during system execution
- Traces = a particular case of LTS

- CADP provides a dedicated trace format:
  - text files
  - one event per line
  - comments allowed
  - multiple traces allowed in the same file

- CADP tool for handling traces: SEQ.OPEN

1.c. Partitioned LTS model

- A useful model for distributed verification
- PBG *(Partitioned BCG Graph)* format



- LTS fragments are split accross different machines
- CADP tools: DISTRIBUTOR, BCG_MERGE

1.d. Implicit LTS model

- A requirement for on-the-fly verification
  - LTS is only built on demand
  - LTS can be built only partially
  - model is only defined by a programming interface
- CADP tools for handling implicit LTS
  - Open/Caesar programming interface
  - many compilers implementing Open/Caesar: *bcg_open, caesar.open, exp.open, if.open, kronos-open, mcrl.open, seq.open, umlaut 1.0*...
  - many tools built on top of Open/Caesar

2. Markov models

Markov models

Extensions of LTS by adding special transitions:

- ordinary transitions ("SEND !12 !true")
- probabilistic transitions ("prob 0.3")
  → *discrete time Markov chains* (DTMC)
- stochastic transitions ("rate 1.1")
  → *continuous time Markov chains* (CTMC)
- mixed transitions
  - "SEND !12 !true ; prob 0.3"
  - "SEND !12 !true ; rate 1.1"

Markov models are encoded in BCG or Open/Caesar

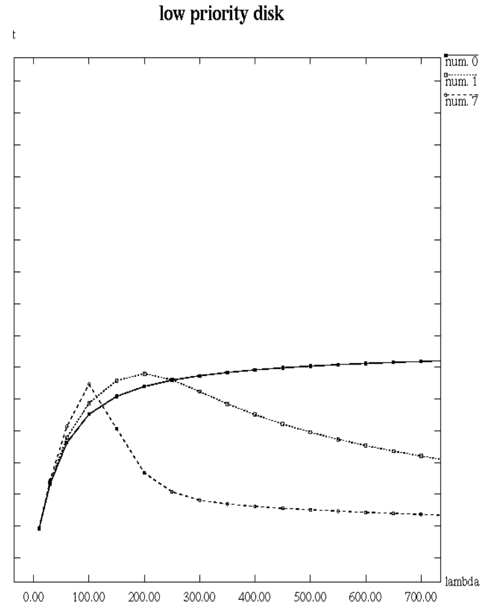
CADP tools for Markov models

BCG_MIN

- minimizes using bisimulation / lumpability

DETERMINATOR

- transforms a well-formed stochastic LTS into a CTMC by removing nondeterminism



BCG_STEADY

- for a CTMC, computes steady-state probabilities and throughputs on the long run
- interfaced with Excel and Gnuplot

BCG_TRANSIENT

- for a CTMC, computes transient probabilities and throughputs at a given list of time instants
- interfaced with Excel and Gnuplot

3. Communicating LTS

Communicating LTS

The EXP 2.0 model of CADP:

- Set of LTS running in parallel asynchronously
- LTS described in various formats (BCG, SEQ...)
- Multiple synchronization primitives:
  - synchronization vectors (MEC, FC2)
  - process algebra parallel operators (CCS, CSP, LOTOS, E-LOTOS)
- Flexible operations on actions:
  - action hiding
  - action renaming
  - action cut

The EXP 2.0 model of CADP

Action lists

$L ::= A_1, \ldots, A_n$

Composition expressions (~ 10 operators)

E ::= "LTS_file"

| **hide** [**all but**] L **in** E

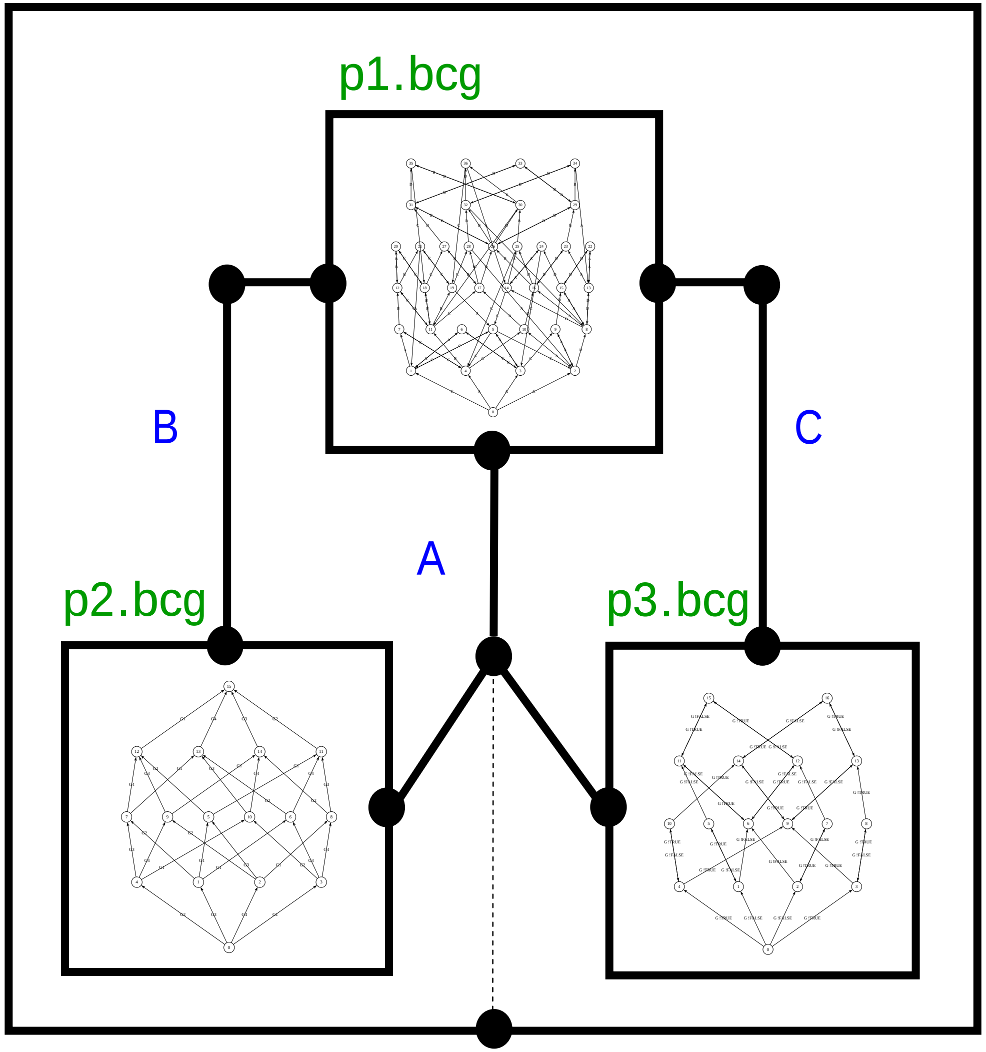
| **rename** $A_1 \rightarrow A'_1, \ldots, A_n \rightarrow A'_n$ **in** E

| **cut** [**all but**] L **in** E

| **par** $[L_1 \rightarrow]\ E_1 \mid\mid \ldots \mid\mid [L_n \rightarrow]\ E_n$

| ...

Example of communicating LTS



hide B, C **in**

par

A, B, C → "p1.bcg"

||

A, B → "p2.bcg"

||

A, C → "p3.bcg"

CADP tools for communicating LTS

- EXP.OPEN 2.0:
  - on-the-fly exploration of communicating LTS
  - partial order and confluence reductions
  - connections to Petri net models: TINA, PEP
  - automatic generation of environment constraints
- PROJECTOR 2.0:
  - LTS generation under environment constraints

4. Process calculi - LOTOS

LOTOS

- A formal model for asynchronous systems (protocols, distributed systems, etc.)
- International standard [ISO-8807:1989]
- Two orthogonal sub-languages:

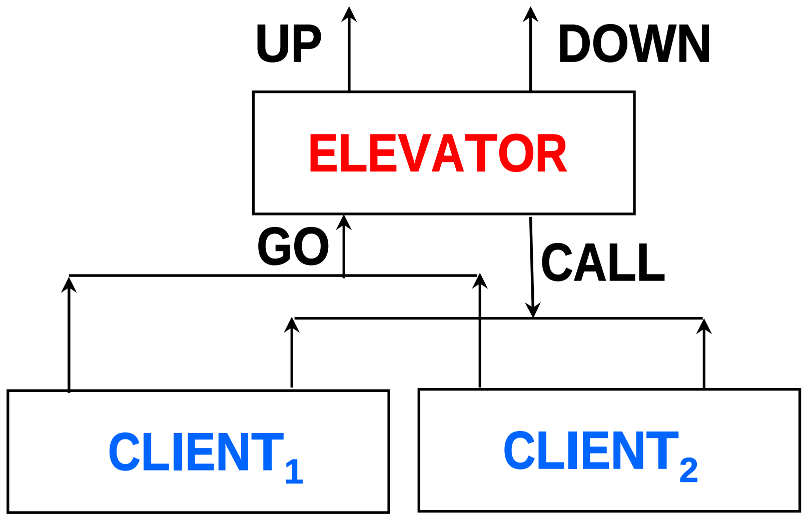
  Data: abstract data types
  - sorts and operations
  - algebraic equations

  Processes: process algebras (CCS, CSP)
  - parallel processes (interleaving semantics)
  - message-passing communication

LOTOS types: Example

```
type FLOOR is  BOOLEAN
  sorts
    FLR

  opns
    LOWER (*! constructor *),
    MIDDLE (*! constructor *),
    UPPER (*! constructor *),
    ERROR (*! constructor *) :-> FLR
    INCR, DECR : FLR -> FLR
    _==_ , _<_ , _>_ : FLR, FLR -> BOOL

  eqns
    forall X, Y:FLR
    ofsort FLR
      INCR (LOWER) = MIDDLE;
      INCR (MIDDLE) = UPPER;
      (* else *) INCR (X) = ERROR;

    ofsort FLR
      DECR (MIDDLE) = LOWER;
      DECR (UPPER) = MIDDLE;
      (* else *) DECR (X) = ERROR;

    ofsort BOOL
      X == X = true;
      (* else *) X == Y = false;

    ofsort BOOL
      LOWER < MIDDLE = true;
      LOWER < UPPER = true;
      MIDDLE < UPPER = true;
      (* else *) X < Y = false;

    ofsort BOOL
      X > Y = Y < X;
endtype
```

LOTOS processes: Example

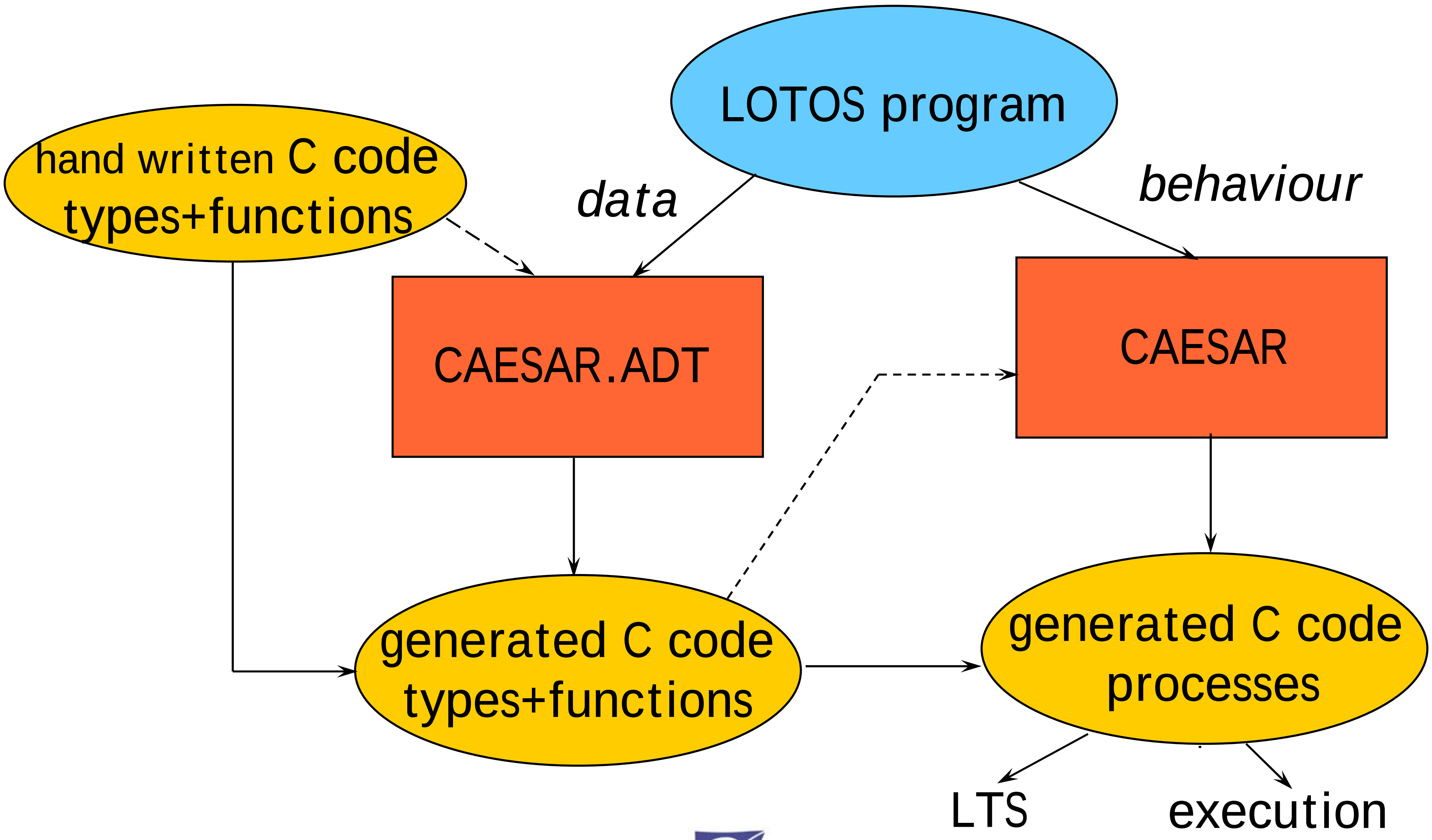


ELEVATOR [CALL, GO, UP, DOWN] (LOWER, LOWER)
|[CALL, GO]|
(
CLIENT [CALL, GO] (LOWER, UPPER)
|||
CLIENT [CALL, GO] (UPPER, MIDDLE)
)

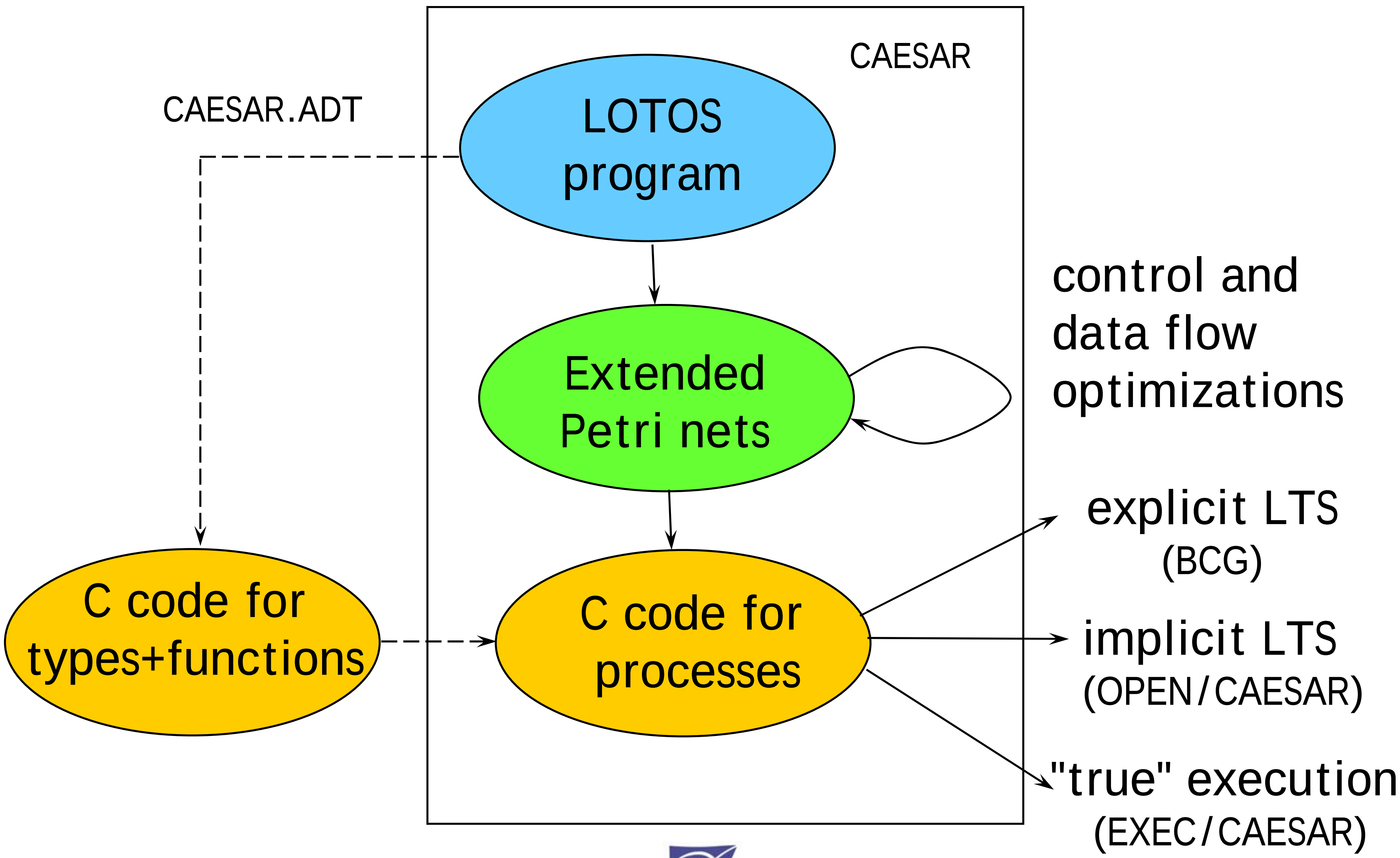
LOTOS processes: Example

```
process ELEVATOR [CALL, GO, UP, DOWN] (CURRENT, TARGET: FLR) : noexit :=
    [TARGET > CURRENT] ->
      UP !INCR (CURRENT);
        ELEVATOR [CALL, GO, UP, DOWN] (INCR (CURRENT), TARGET)
    []
    [TARGET < CURRENT] ->
      DOWN !DECR (CURRENT);
        ELEVATOR [CALL, GO, UP, DOWN] (DECR (CURRENT), TARGET)
    []
    [TARGET == CURRENT] ->
      (
      CALL ?NEW_TARGET:FLR;
        ELEVATOR [CALL, GO, UP, DOWN] (CURRENT, NEW_TARGET)
      []
      GO ?NEW_TARGET:FLR;
        ELEVATOR [CALL, GO, UP, DOWN] (CURRENT, NEW_TARGET)
      )
endproc
```

CADP tools for LOTOS



Intermediate models for LOTOS



5. Modal mu-calculus formulas

Modal mu-calculus

Logic formulas to describe temporal properties of actions *A* present in the LTS

Action formulas

$$\alpha ::= A \mid \neg\alpha \mid \alpha_1 \vee \alpha_2 \mid \alpha_1 \wedge \alpha_2$$

State formulas

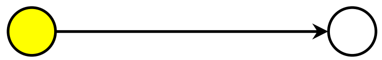
$$\varphi ::= F \mid T \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2$$
$$\mid \langle \alpha \rangle \varphi \mid [\alpha]\varphi \mid X \mid \mu X . \varphi \mid \nu X . \varphi$$

with extensions (regular expressions) for user friendliness

Mu-calculus examples (1)

- Deadlock freedom

  $\langle \mathrm{T} \rangle \mathrm{T}$



- Potential reachability of an action *A*

  $\mu X \,.\, \langle A \rangle \mathrm{T} \vee \langle \mathrm{T} \rangle X$

  or simply

  $\langle \mathrm{T}^*.\, A \rangle \mathrm{T}$

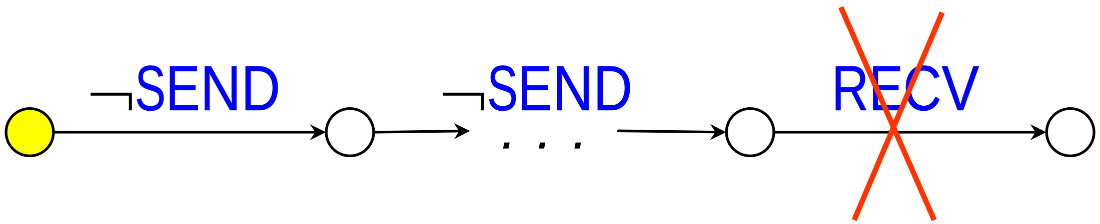
Mu-calculus examples (2)

- No RECV reached before a SEND

  $\nu X \,.\, [\ \text{RECV}\ ]\ \text{F} \wedge [\ \neg\text{SEND}\ ]\ X$

  or simply

  $[(\neg\text{SEND})^*.\text{RECV}\ ]\ \text{F}$



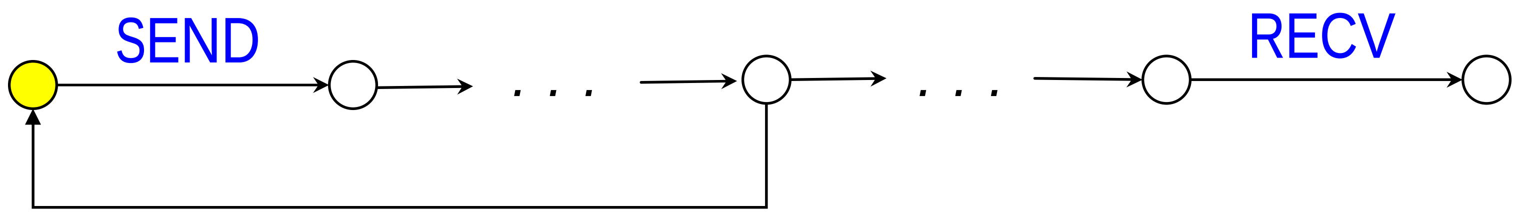
Mu-calculus examples (3)

- A RECV is *fairly* reached after each SEND

  $\nu X . ([\text{SEND}] \mu Y . (\langle \text{RECV} \rangle T \vee \langle \neg \text{RECV} \rangle Y) \wedge [T] X)$

  or simply

  $[T^*.\text{SEND}] \langle (\neg \text{RECV})^*.\text{RECV} \rangle T$



CADP tool for mu-calculus formulas

EVALUATOR 3.5 model checker:

- User-defined libraries of formula templates
- Formulas evaluated on-the-fly while the LTS model is under construction
- Diagnostic (LTS fragment) generated to explain why a formula is true or false
- Linear-time complexity wrt LTS size and formula size

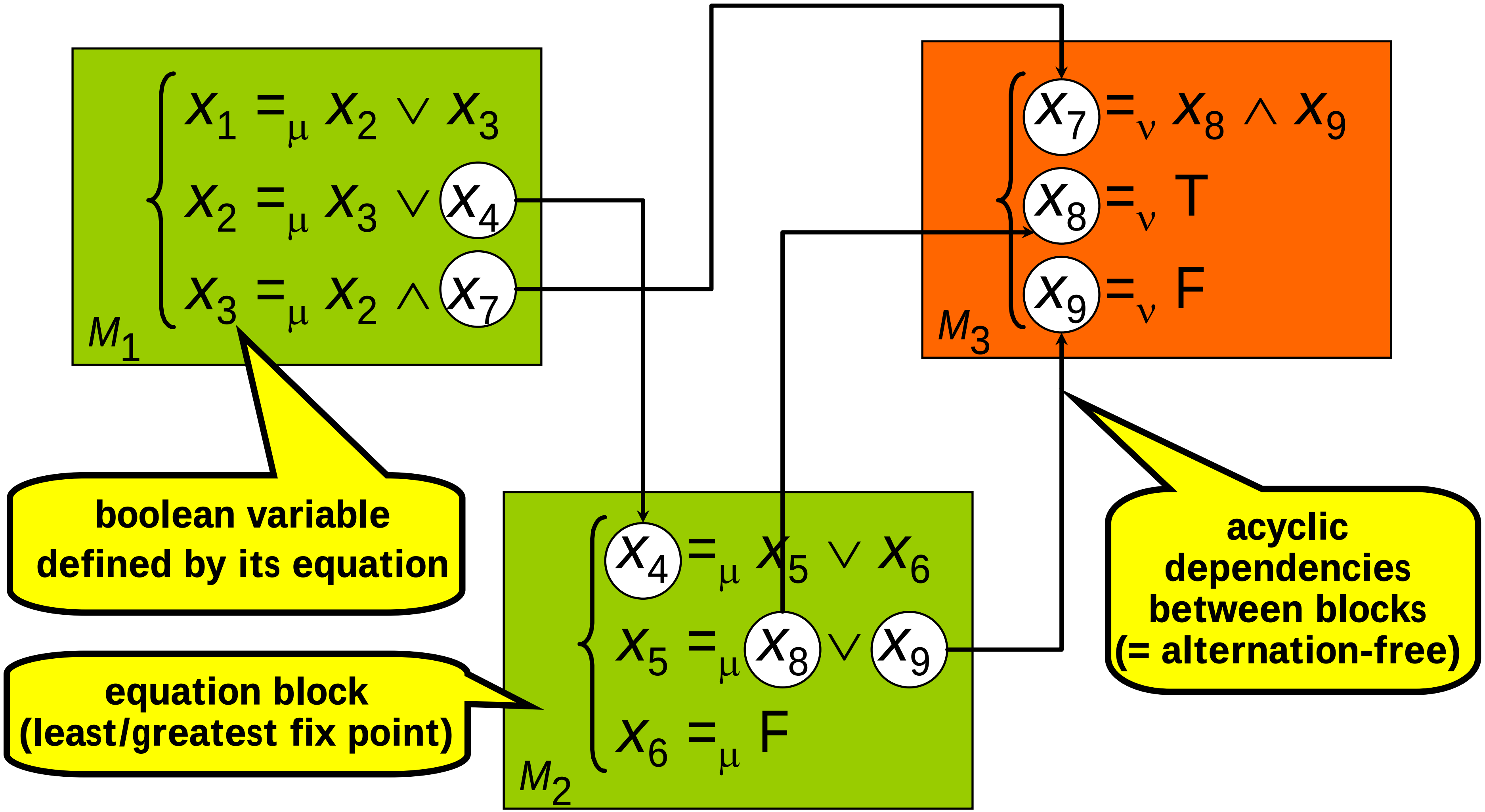
6. Boolean Equation Systems

Boolean Equation Systems

- A key formalism to encode many verification problems
- Examples:
  - Model checking (temporal logics)
  - Equivalence checking (bisimulations)
  - State space reductions (confluence, partial orders, ...)
  - Test case generation

- BES are often represented as *boolean graphs* or *game graphs*

Boolean Equation Systems



Boolean Equation Systems in CADP

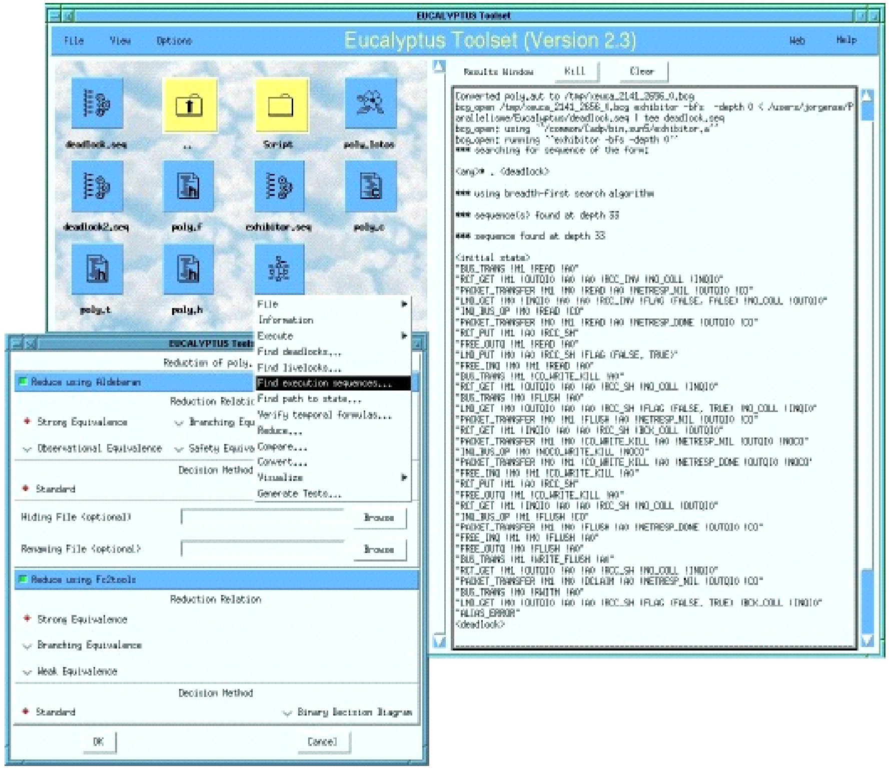
- BES are often large (10^8 variables, 10^9 operators)
- CADP supports two representations:
  - Explicit BES: text file format with gzip compression
  - Implicit BES: "caesar_solve" programming interface
- CADP library for solving BES:
  - CAESAR_SOLVE_1: on the fly solver
  - diagnostic generation (examples or counterexamples)
- Three CADP tools for generating BES:
  - EVALUATOR (model checking)
  - BISIMULATOR (equivalence checking)
  - REDUCTOR (minimization)

7. Verification scenarios

CADP end-user interfaces

2 different interfaces:

- EUCALYPTUS graphical user-interface
  - dialog boxes
  - file types
  - contextual menus
  - online help



- SVL *(Script Verification Language)* a Domain Specific Language for verification

Motivations for SVL

1. Textual interface for all the CADP tools

2. Intuitive specification of verification scenarios

3. Reduced complexity of large compositional verification scenarios

   Example: 70 processes $\Rightarrow$ 500 intermediate files

SVL Script: Example 1

```
% DEFAULT_LOTOS_FILE="bitalt_protocol.lotos"
"bitalt_protocol.exp" =
    leaf strong reduction of
      hide SDT, RDT, RDTe, RACK, SACK, SACKe in
        (
          (BODY_TRANSMITTER ||| BODY_RECEIVER)
          |[SDT, RDT, RDTe, RACK, SACK, SACKe]|
          (MEDIUM1 ||| MEDIUM2)
        );
"bitalt_dead.seq" = deadlock of "bitalt_protocol.exp";
"bitalt_live.seq" = livelock of "bitalt_protocol.exp";
branching comparison "bitalt_protocol.exp" == "bitalt_service.lotos";
```

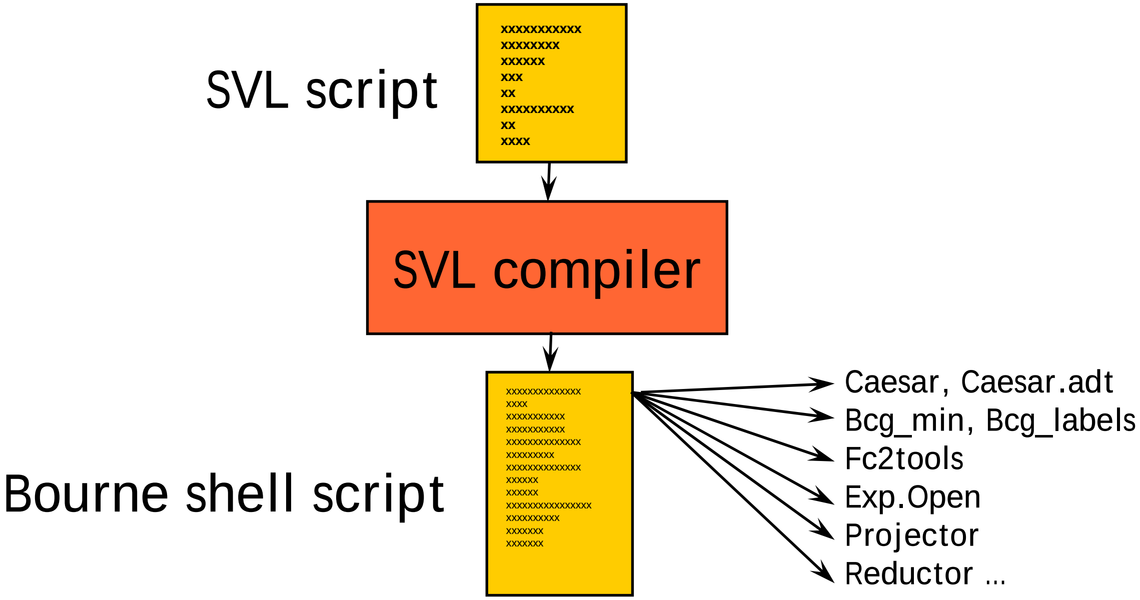
SVL Script: Example 2

```
% DEFAULT_LOTOS_FILE="rel_rel.lotos"
"crash_trans.bcg" = strong reduction of CRASH_TRANSMITTER ;
"rel_rel.bcg" = generation of leaf strong reduction of
   hide R_T1, R_T2, R_T3, R12, R13, R21, R23, R31, R32  in
   (  (  (  (RECEIVER_NODE_1 -||? "r1_interface.lotos")
            |[R12, R21, R13, R31]|
            (   (RECEIVER_NODE_2 -||? "r2_interface.lotos")
                |[R23, R32]|
                 (RECEIVER_NODE_3 -||? "r3_interface.lotos")
            ) -|[R_T2, R_T3]| "crash_trans.bcg"
            ) -|[R_T1, R_T2, R_T3]| "crash_trans.bcg"
        )
    |[R_T1, R_T2, R_T3]|
    "crash_trans.bcg");
```

CADP tool for verification scenarios

SVL compiler:

- translates SVL scripts into Bourne shell scripts
- launches these shells scripts



Conclusion

Models in CADP

Seven models available to CADP users

1. Labelled Transition Systems
2. Markov models
3. Communicating automata
4. Process calculi - LOTOS
5. Modal mu-calculus formulas
6. Boolean equation systems
7. Verification scenarios

More models used internally

- Petri nets extended with data
- Boolean graphs

Some models do not fit in the usual model-driven approach

- compressed binary file formats
- models split into fragments and/or accross different machines
- on-the-fly: models built on-demand by pipelined applications

More information ...

http://www.inrialpes.fr/vasy/cadp