Panorama des outils CADP

Hubert Garavel

projet VASY

INRIA

655, avenue de l'Europe

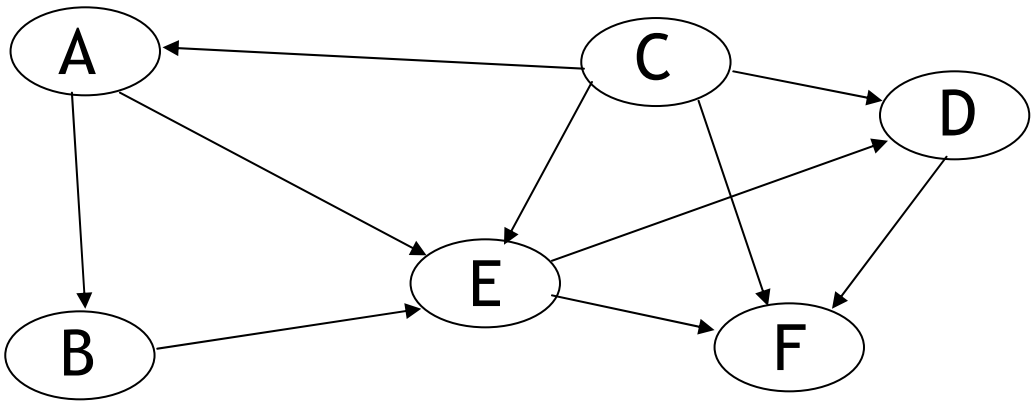
38330 Montbonnot Saint Martin

Le projet Vasy de l'INRIA

- Fondé en 1997, projet INRIA depuis 2000
- 3 permanents (INRIA)
  - Hubert Garavel
  - Radu Mateescu
  - Frédéric Lang
- 1 chef de projet (Bull)
  - Solofo Ramangalahy
- 1 post-doc : Gordon Pace
- 2 ingénieurs experts
  - Bruno Ondet
  - Frédéric Tronel
- 1 étudiant DEA : Christophe Joubert

Problématique scientifique

- Systèmes d'agents concurrents
- Parallélisme asynchrone (≠ synchrone)
- Communication par messages (≠ mémoire partagée)



Modèle sémantique très général

Domaines d'application

- Télécommunications
  - protocoles
- Logiciel
  - algorithmique distribuée
  - systèmes répartis
- Matériel
  - circuits asynchrones
  - architectures multiprocesseurs
    - arbitrage de bus
    - cohérence de caches

Deux axes scientifiques

- Modéliser les systèmes asynchrones
  - Définir des langages de haut niveau offrant
    - sémantique formelle
    - expressivité
    - simplicité, confort
  - Développer des techniques de compilation
- Valider les modélisations
  - Combiner simulation, vérification, test
  - Lutter contre l'explosion d'états
  - Concevoir des modèles intermédiaires
  - Développer des composants logiciels réutilisables

Langages de modélisation

- Beaucoup de formalismes existants
- Algèbres de processus
  - Formalismes textuels (≠ graphiques)
  - Sémantique formelle
  - Expressivité
  - Théorie : bisimulation, compositionalité
- Langages étudiés par VASY
  - LOTOS
  - E-LOTOS

LOTOS

Language Of Temporal Ordering Specification [ISO-8807:1989]

- Une technique de description formelle pour la description des systèmes asynchrones
- Deux sous-langages « orthogonaux » :
  - **Données:** types de données abstraits (ActOne)
    - Sortes
    - Operations (constructeurs et non-constructeurs)
    - Equations et filtrage (*pattern-matching*)
  - **Comportements:** algèbre de processus (CCS et CSP)
    - Parallélisme asynchrone (sémantique d'entrelacement)
    - Communication par messages

E-LOTOS

Enhanced LOTOS [ISO-15437:2001]

- Une méthode formelle « de 2ème génération »
- Objectifs :
  - Avoir la sémantique formelle (algèbres de processus)
  - Faciliter la diffusion en milieu industriel
- Principaux choix de conception :
  - Types de données et fonctions : fonctionnel/impératif
  - Symétrie entre fonctions et processus
  - Traitement d'exceptions
  - Temps quantitatif

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

Modèles et vérification

- Vérification énumérative
  - Exploration des états accessibles
  - Construction de systèmes de transitions étiquetées (STE)
  - Relations d'équivalence (bisimulation)
  - Model-checking (μ-calcul modal)
- Composants logiciels
  - BCG (Binary-Code Graphs) : stockage de STE de très grande taille
  - Open/Caesar : support pour l'exploration de STE à la volée
  - NTIF : description de processus avec données symboliques
- Environnements multi-fonctions
  - Exécution aléatoire
  - Simulation
  - Vérification
  - Test

CADP

- *CAESAR/ALDEBARAN Development Package*
- Une boîte à outils pour l'ingénierie des protocoles
- Caractéristiques essentielles :
  - algèbres de processus (LOTOS)
  - vérification par bisimulations
  - vérification par model-checking
  - simulation
  - génération de tests
  - prototypage rapide

Origines de CADP

- Développement: 1986-2002…
- Diffusé sous licence INRIA/CNRS
- Travail conjoint entre
  - **le projet VASY de l'INRIA**
  - **le laboratoire Verimag**

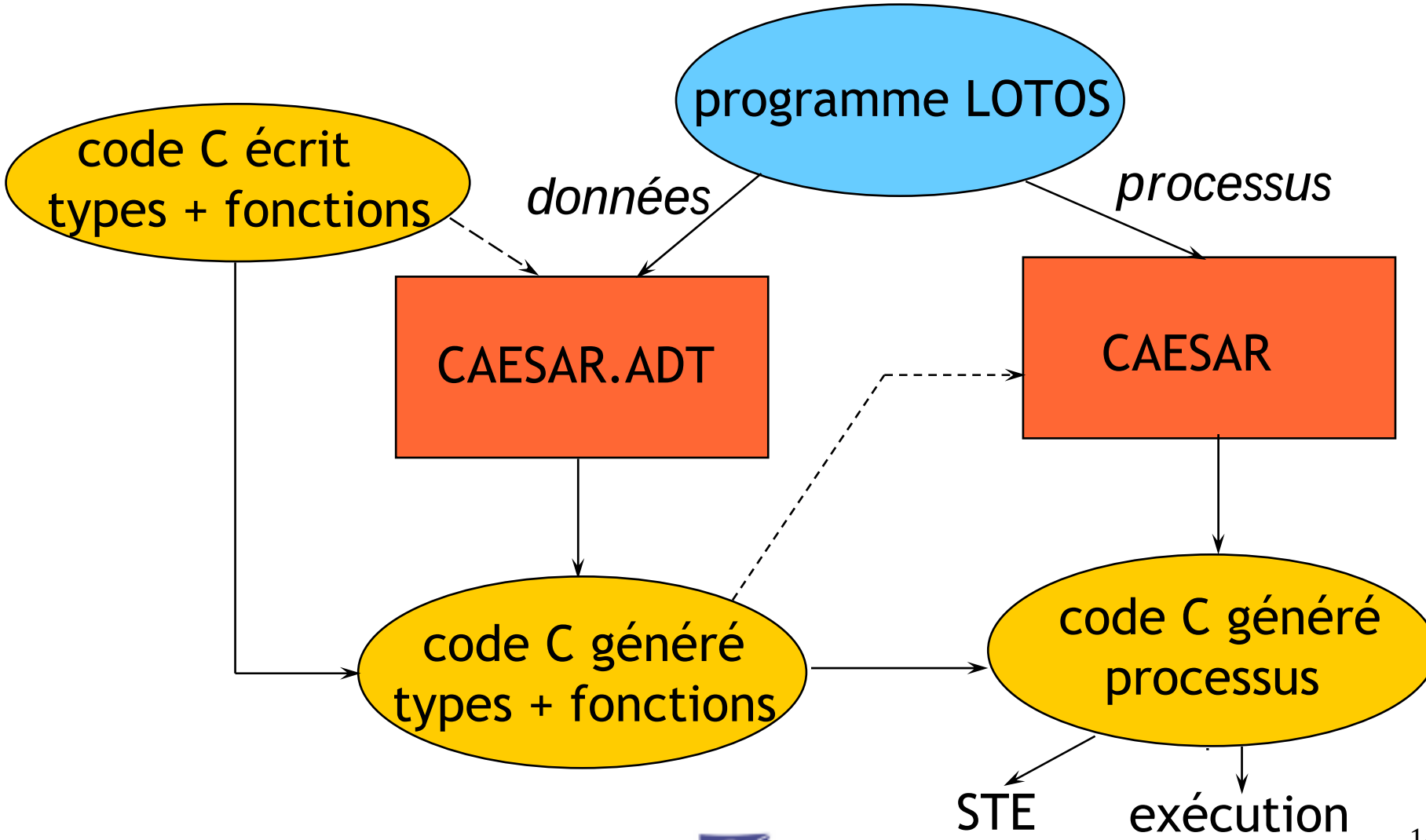
  avec des contributions
  - **de l'ex-projet PAMPA de l'IRISA**
  - **du groupe FMT de l'Université de Twente**

Plan

- Outils pour LOTOS
- Outils pour les STE explicites
- Outils pour les STE implicites
- Architecture de CADP

Outils pour LOTOS

CAESAR.ADT et CAESAR



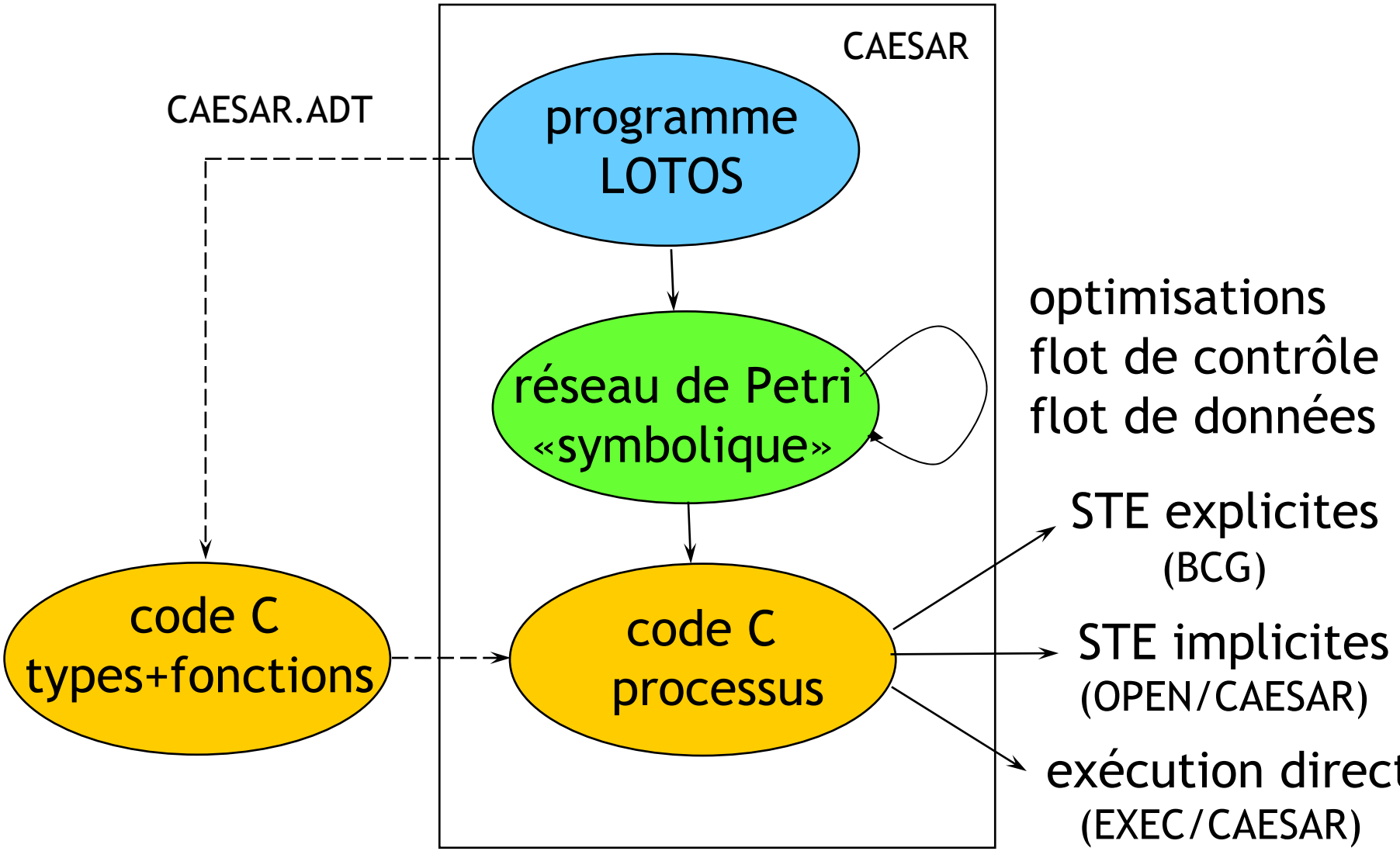
CAESAR.ADT (1989-1993)

- **Traduction : types abstraits LOTOS → C**
- **Hypothèses ajoutées à la norme LOTOS**
  - constructeurs / non-constructeurs
  - constructeurs
  - equations vues comme des règles de réécriture avec priorité
- **Génération de code C spécialisé**
  - Ciblé vers les besoins du model-checking
  - *Optimiser d'abord la mémoire, puis la vitesse*

CAESAR.ADT (suite)

- **Compilation des structures de données**
  - structures de données dynamiques (listes, arbres...) permises
  - implantation mémoire optimisée
    - nombre minimal de bits
    - permutation des champs de « records »
    - mise en facteur des sous-termes communs
- **Compilation des fonctions**
  - algorithme de compilation du filtrage
  - optimisations ad hoc

CAESAR (1986-2000)



Outils pour les STE ***explicites***

Motivations

- Comment stocker des STE **très grands** dans des fichiers ?
- Les formats textuels ne conviennent pas
  - Format Aldébaran
    - coûteux en place disque (centaines de Mbytes, voire Gbytes)
    - lent (lecture/écriture coûteuses en temps)
  - Format Fc2
    - difficile à « parser »
    - ambigu

BCG (1992-1997)

BCG (*Binary-Coded Graphs*)

- **un format de fichier compact** pour les STE
- **un ensemble d'APIs**
- **un ensemble de bibliothèques logicielles**
- **un ensemble d'outils**

Implémentation : 30,000 lignes de code C

BCG est fourni comme composant de CADP

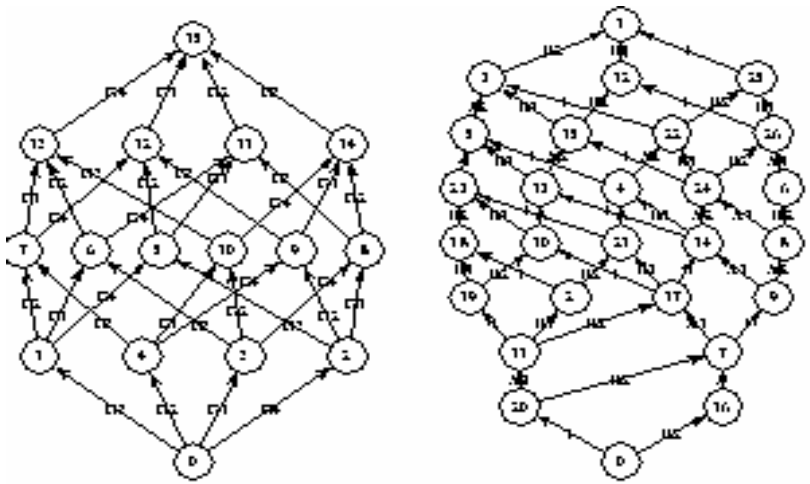
Tous les outils de CADP utilisent BCG uniformément

Bibliothèques et API BCG

- **bcg_write:** API pour créer un fichier BCG
- **bcg_read:** API pour lire un fichier BCG
- **bcg_transition:** API pour stocker une relation de transition en mémoire :
  - fonction «successeur»
  - fonction «prédécesseur»
  - fonctions «successeur» + «prédécesseur»

Les outils BCG de base

- **bcg_info:** extrait les infos d'un fichier BCG
- **bcg_io:** conversion BCG $\leftrightarrow$ autres formats
- **bcg_labels:** masque/renomme les étiquettes
- **bcg_draw, bcg_edit:** visualisation de STE

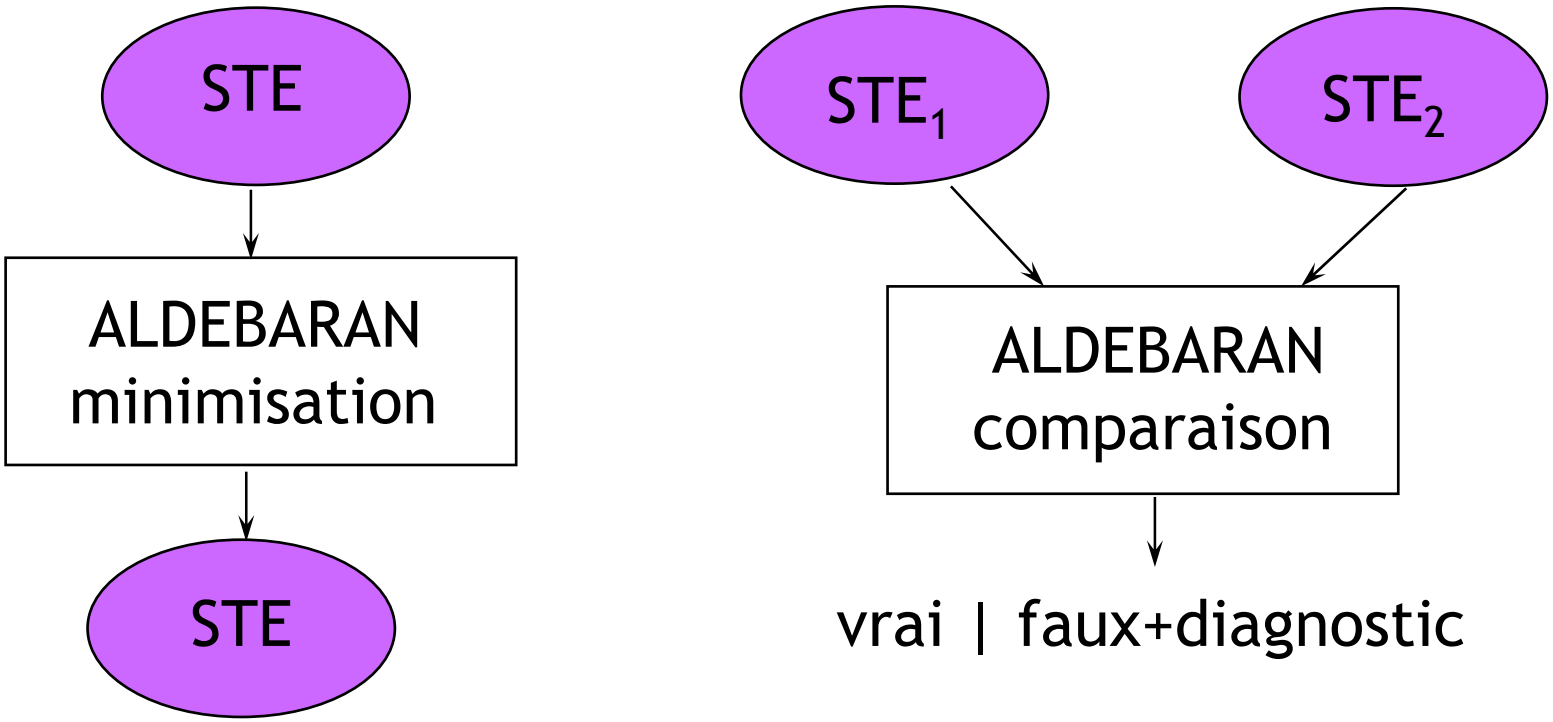


XTL (1994-1999)

- Un langage d'interrogation pour le format BCG
- Caractéristiques :
  - Langage fonctionnel
  - Types prédéfinis : ***states, state sets, transitions, transition sets, labels…***
  - Accès aux types, variables… du programme source
- Applications
  - bibliothèques: HML, CTL, ACTL, μ-calcul
  - prototypage rapide de logiques temporelles
  - logiques temporelles étendues avec valeurs

ALDEBARAN (1985-1999)

- Outil pour calculer des relations de bisimulation [Fernandez, Mounier, Kerbrat]



Relations implémentées : équivalence forte, observationnelle, de branchement, de sûreté...

BCG_MIN (2000-2002)

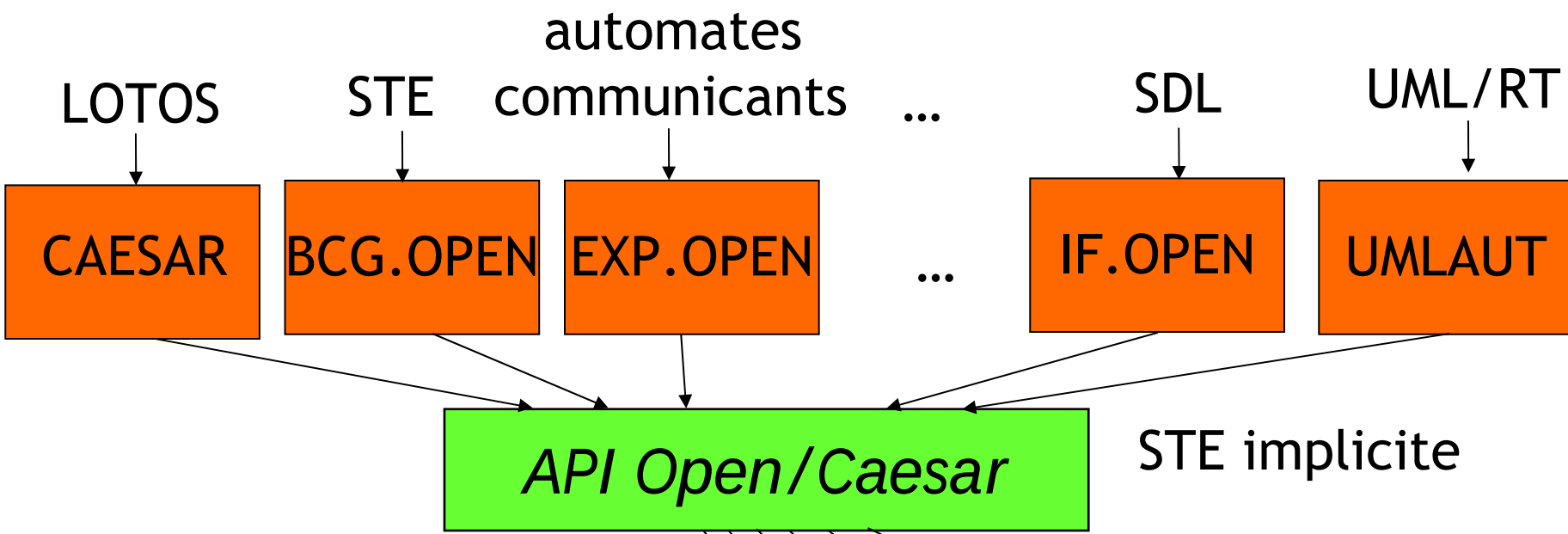
- Minimisation efficace de « grands » STE
- Plusieurs types de STE
  - **STE « ordinaires »**
    - ✓ bisimulation forte [Kanellakis-Smolka]
    - ✓ bisimulation de branchement [Groote-Vaandrager]
    - ✓ performances supérieures à Aldebaran et Fc2min
    - ✓ meilleur affichage des classes d'équivalence
  - **STE probabilistes** : "**prob** *p*" transitions
  - **STE stochastiques** : "**rate** λ" transitions
  - **modèles mixtes** : "*label* ; **prob** *p*"
- *Travail fait avec Holger Hermanns (Univ. of Twente)*

Outils pour les STE ***implicites***

Motivations

- La plupart des *model checkers* sont dédiés à un langage d'entrée particulier: Spin, SMV…
- Difficiles à réutiliser pour d'autres langages
- Idée: introduire de la modularité en séparant
  - **les aspects dépendants du langage source :** compilation du langage source en un STE
  - **les aspects indépendants du langage source :** algorithmes pour l'exploration des STE

OPEN/CAESAR (1992-1998)



API Open/Caesar

STE implicite

bibliothèques Open/Caesar

- génération de STE
- simulation interactive
- exécution aléatoire
- vérification à la volée
- vérification partielle
- génération de tests

Bibliothèques OPEN/CAESAR

Des structures de données prédéfinies

- EDGE : listes de transitions (ex.: listes de successeurs)
- HASH : catalogue de fonctions de hash-code
- STACK_1 : piles d'états et/ou d'étiquettes
- DIAGNOSTIC_1 : ensemble de traces d'exécution
- TABLE_1 : tables d'états
- BITMAP : tables "*bit state*" de Holzmann

Des primitives dédiées à la vérification à la volée

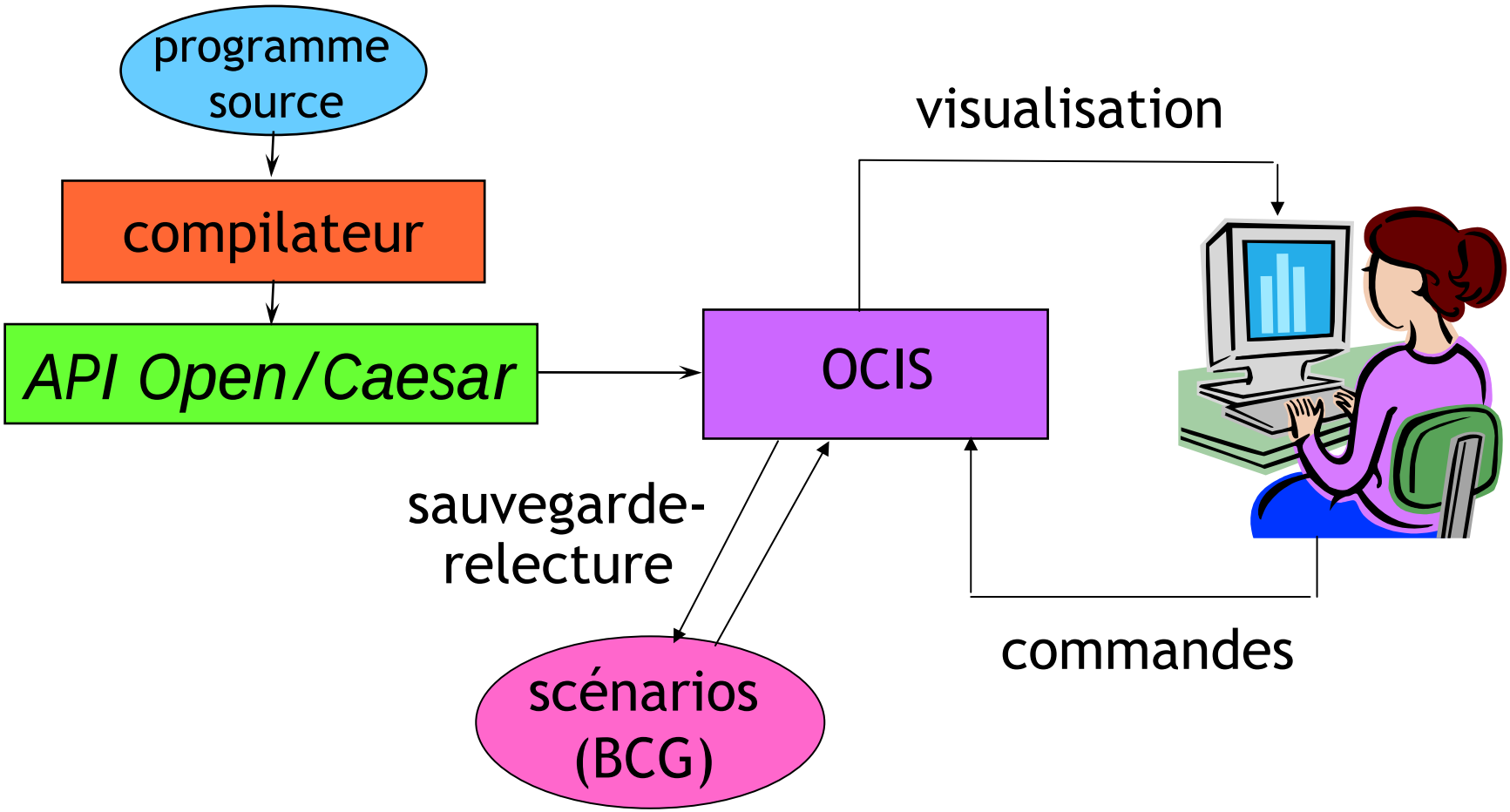
- possibilité d'attacher des informations aux états
- débordement de pile ou de table $\Rightarrow$ *backtracking*
- etc.

Outils OPEN/CAESAR

- EXECUTOR : exécution aléatoire
- SIMULATOR : simulation interactive (tty)
- XSIMULATOR : simulation interactive (x-windows)
- GENERATOR : génération exhaustive de STE
- REDUCTOR : génération de STE avec réduction
- PROJECTOR : génération de STE sous contrainte
- TERMINATOR : algorithme *bit space* de Holzmann
- EXHIBITOR : recherche de chemins selon exp. rég.
- EVALUATOR : évaluation de formules de μ-calcul
- TGV : génération de séquences de tests

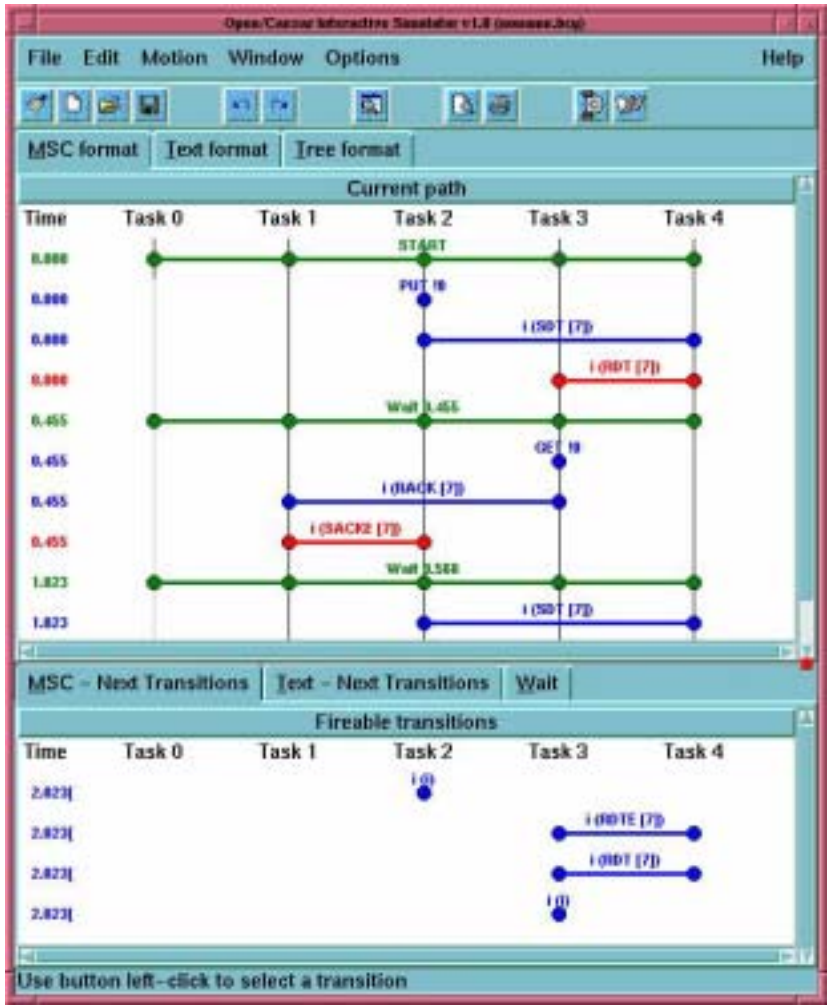
```
#include "caesar_graph.h"
#include "caesar_edge.h"
#include "caesar_table_1.h"

TYPE_TABLE_1 t;     TYPE_STATE s1, s2;                TYPE_EDGE e1_en, e;
TYPE_LABEL l;       TYPE_INDEX_TABLE_1 n1, n2     TYPE_POINTER dummy;

INIT_GRAPH ();
INIT_EDGE (FALSE, TRUE, TRUE, 0, 0);
CREATE_TABLE_1 (&t, 0, 0, 0, 0, TRUE, NULL, NULL, NULL, NULL);
if (t == NULL) ERROR ("not enough memory for table");

START_STATE ((TYPE_STATE) PUT_BASE_TABLE_1 (t));
PUT_TABLE_1 (t);
while (!EXPLORED_TABLE_1 (t)) {
    s1 = (TYPE_STATE) GET_BASE_TABLE_1 (t);
    n1 = GET_INDEX_TABLE_1 (t);
    GET_TABLE_1 (t);

    CREATE_EDGE_LIST (s1, &e1_en, 1);
    if (TRUNCATION_EDGE_LIST () != 0) ERROR ("not enough memory for edge lists");

    ITERATE_LN_EDGE_LIST (e1_en, e, l, s2) {
        COPY_STATE ((TYPE_STATE) PUT_BASE_TABLE_1 (t), s2);
        (void) SEARCH_AND_PUT_TABLE_1 (t, &n2, &dummy);
        print_edge (n1, STRING_LABEL (l), n2);
    }
    DELETE_EDGE_LIST (&e1_en);
}
```

OCIS *(Open/Caesar Interactive Simulator)*



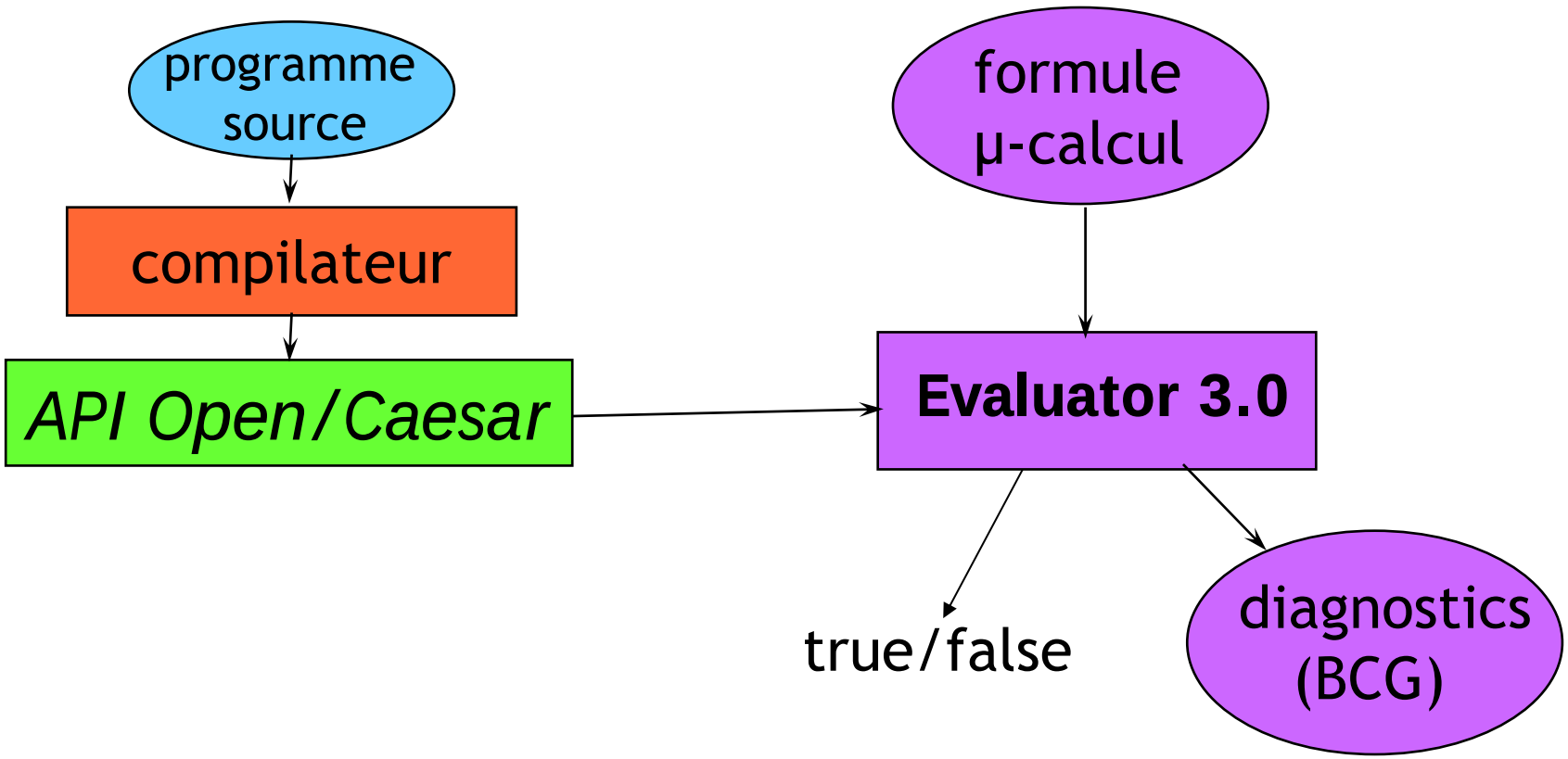
OCIS *(Open/Caesar Interactive Simulator)*

- indépendant du langage
- scénarios arborescents
- sauvegarde/relecture de scénarios
- remontée au code source
- recompilation dynamique
- support pour tâches/MSC



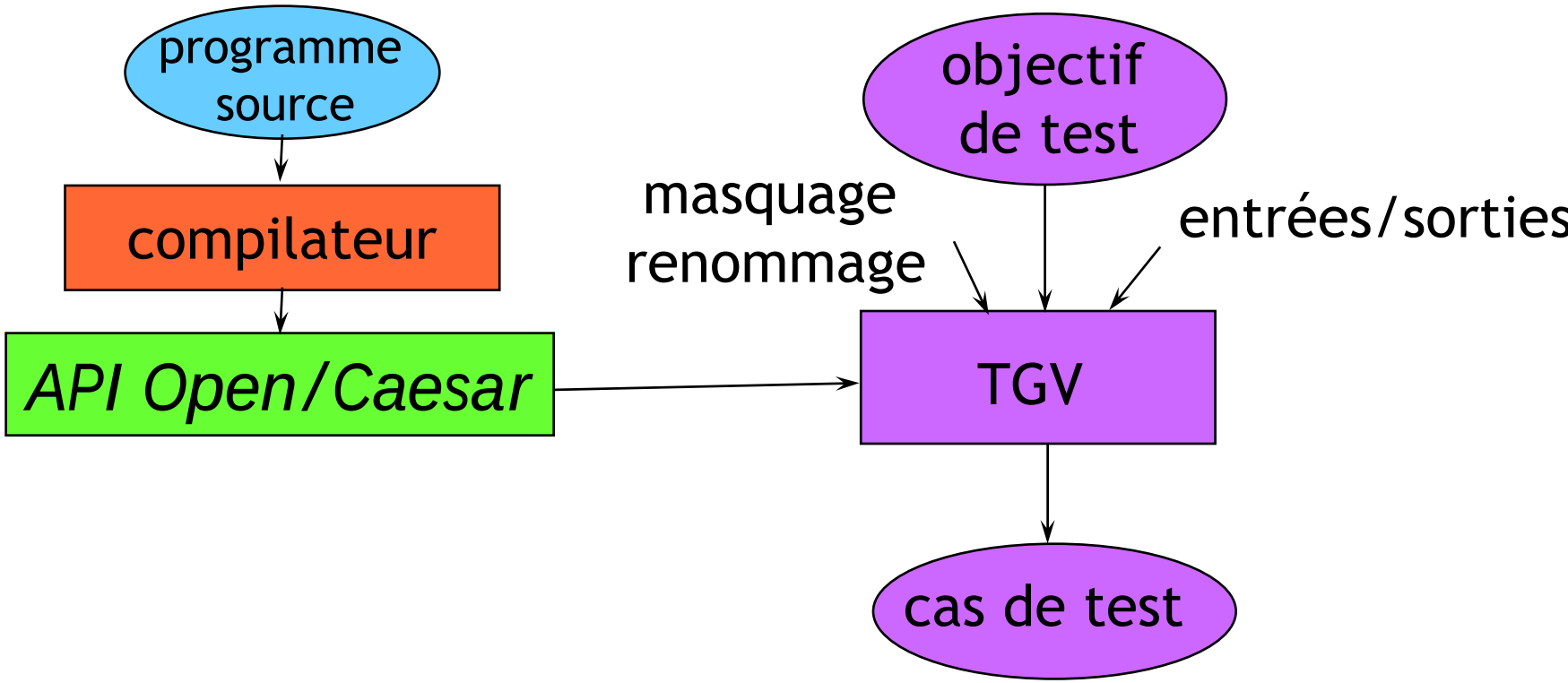
Evaluator

Model checking à la volée de formules
de μ-calcul régulier sans alternance [R. Mateescu]



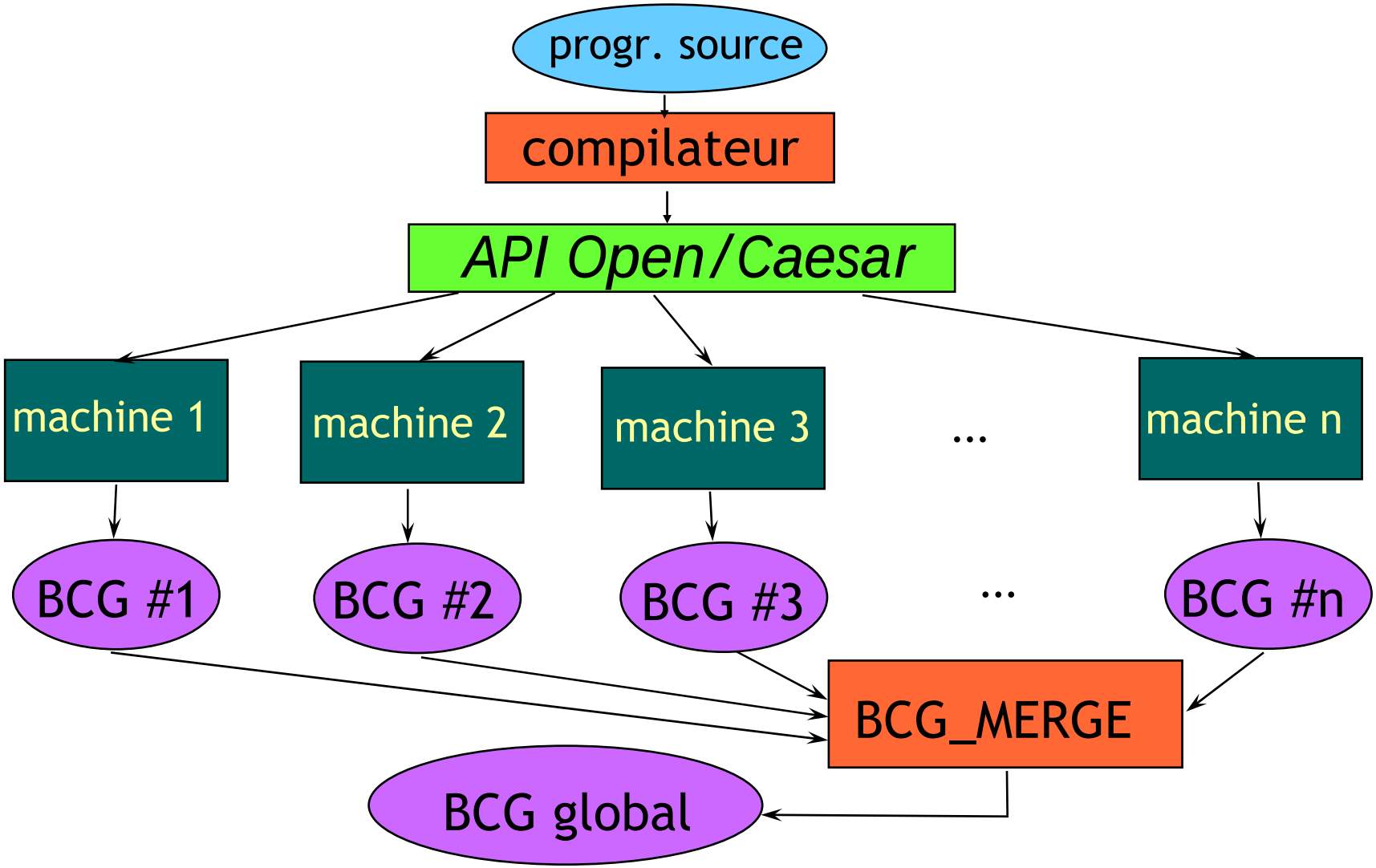
TGV

Génération à la volée de cas de test définis par des objectifs de tests écrits manuellement



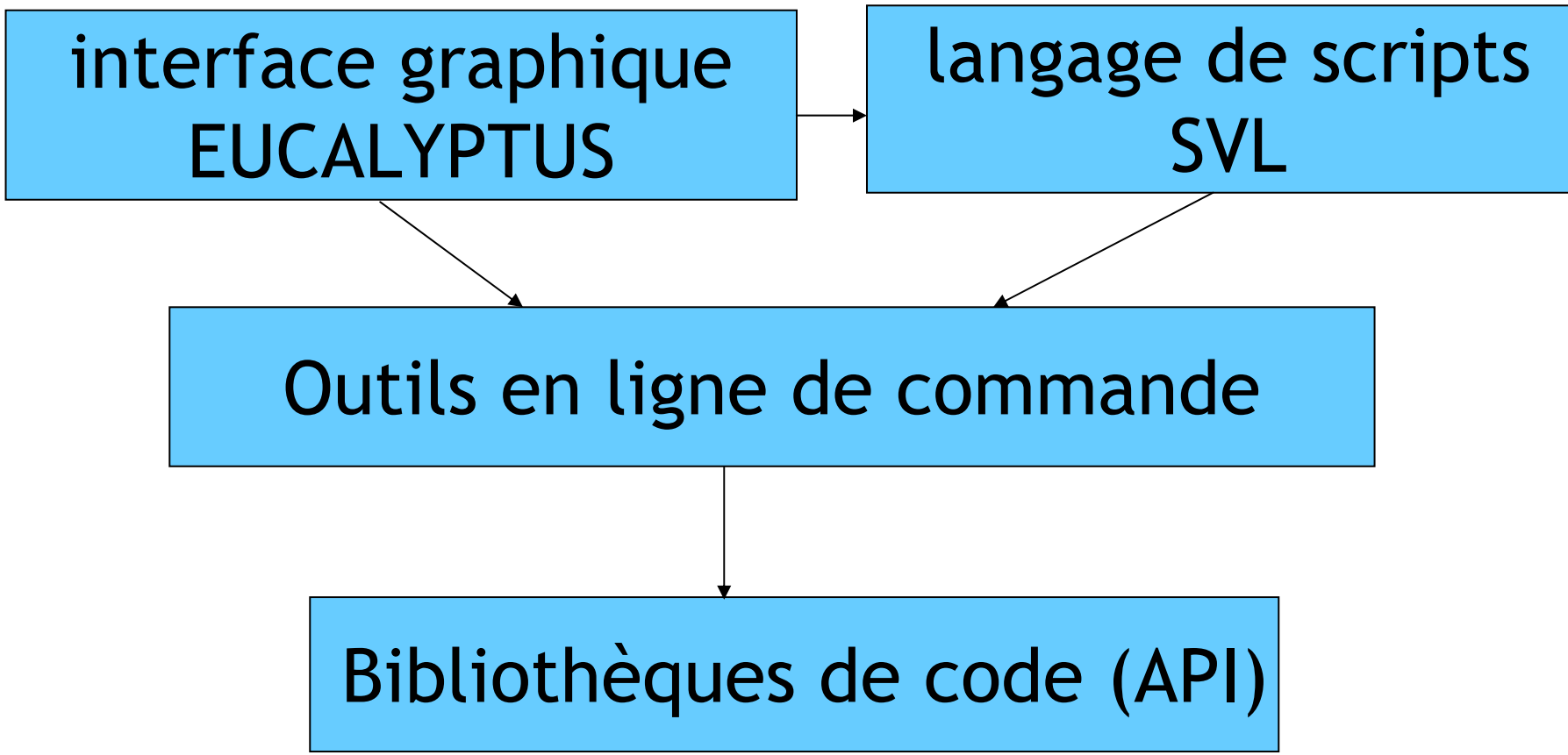
travail joint entre IRISA et Verimag (+ VASY)

DISTRIBUTOR : parallélisme massif



Architecture de CADP

Une architecture stratifiée



L’interface EUCALYPTUS (1993-1997)

- Typage des fichiers
- Menus contextuels
- Boîtes de dialogue
- Multi-outils : CADP, FC2
- Aide en ligne

SVL (1998-2001)

- Double motivation
- Langage de scripts (interface textuelle)
  - De nombreux outils dans CADP (+ Fc2Tools)
  - Plusieurs langages supportés (Open/Caesar)
  - Utilisateurs novices (industriels et étudiants)
- Scénarios de vérification compositionnelle
  - Génération de processus séparés
  - Minimisation des processus
  - Recomposition des processus

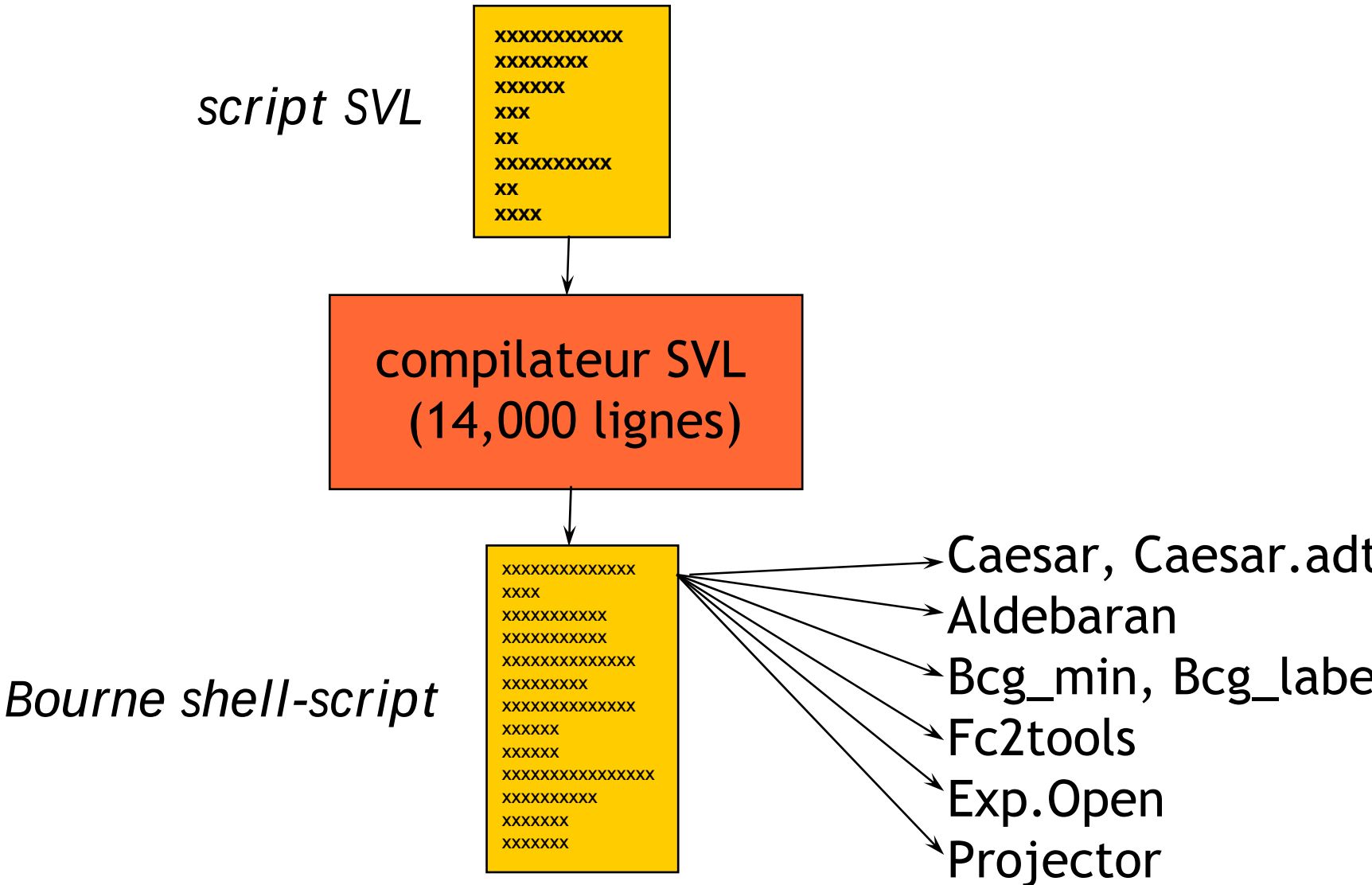
Exemple de script SVL

```
% 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 using fly with aldebaran
          "bitalt_protocol.exp" == "bitalt_service.lotos";
```

Exemple de script SVL

```
% 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");
```

Le compilateur SVL



Conclusion

Conclusion

- CADP: une plate-forme pour l'ingénierie des protocoles
- Architectures supportées
  - Sun sous SunOS ou Solaris
  - PC sous Linux
  - PC sous Windows
- Large diffusion
  - **250 sites** (dont Alcatel, **Bull**, Ericsson, Gemplus, Lucent, Nokia, Oblog, Philips, etc.)
  - En 2000: licences accordées pour **770 machines**
  - En 2001: licences accordées pour **954 machines**
  - **58 études de cas** effectuées avec CADP
  - **11 outils de recherche** construits avec CADP

Trois classes d'utilisation

- **Etudes de cas industrielles**
  - matériel, logiciel, télécoms…
  - spécification formelle de systèmes/protocoles critiques
  - simulation, vérification, test
- **Recherche**
  - développement d'outils de vérification/test avec CADP
  - implémentation de nouveaux langages avec CADP
- **Enseignement**
  - parallélisme, algèbres de processus, bisimulations, *model checking*
  - outils robustes pour TD, TP et projets d'étudiants

Plus d’information ?

http://www.inrialpes.fr/vasy