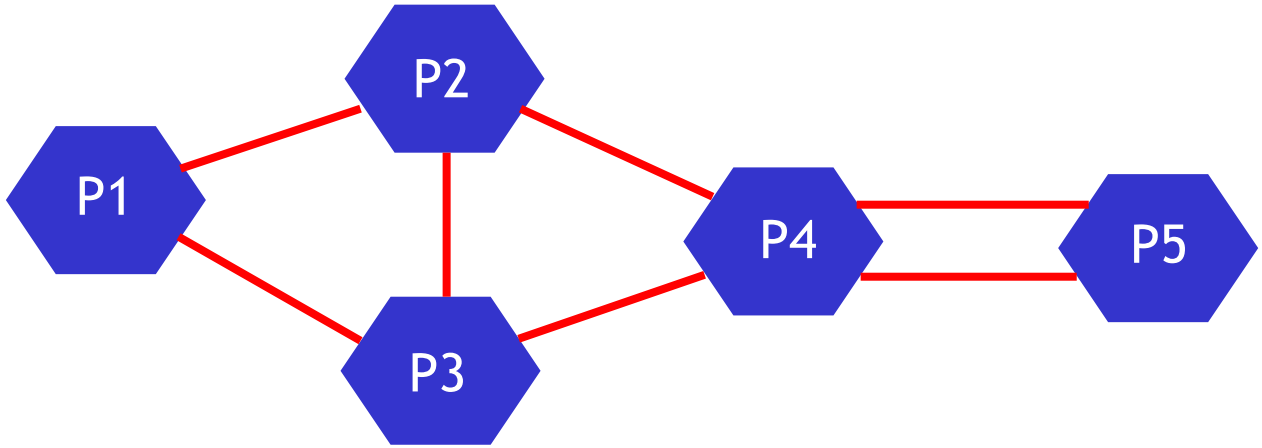
Analyse et vérification automatique de systèmes asynchrones

Hubert Garavel
INRIA / VASY
http://www.inrialpes.fr/vasy

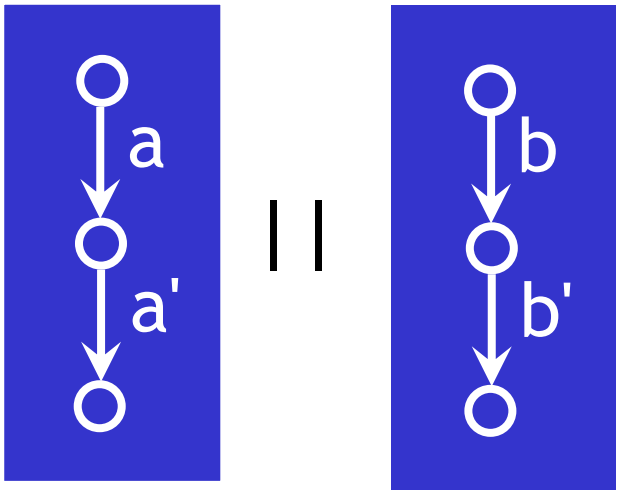
Systèmes asynchrones



- Caractéristiques :
  - différents processus / tâches / activités / agents
  - qui s'exécutent en parallèle
  - à des vitesses différentes (pas d'horloge centrale)
  - n'ayant pas forcément de mémoire commune
  - avec des délais de communication pouvant varier

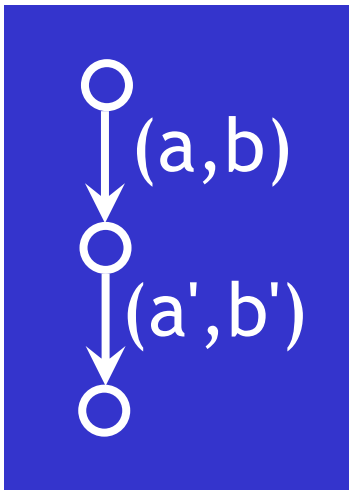
Synchrone / Asynchrone

- Quelle différence ?
  - Synchrone : a || b = (a,b)
  - Asynchrone : a || b = a.b + b.a



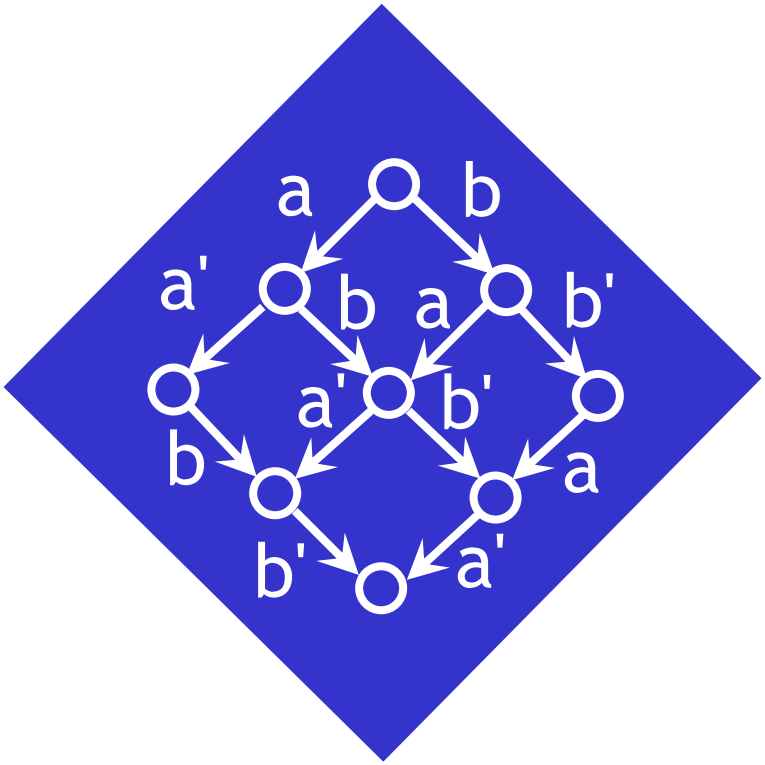
||

=



synchrone

ou



asynchrone
(entrelacements)

Multiples domaines d'application

- Télécommunications
  - protocoles et services
- Logiciel
  - applications réparties
  - agents mobiles
  - systèmes embarqués
  - transactions en ligne, réseaux P2P…
  - sécurité, distribution de clés cryptographiques
- Matériel
  - architectures multiprocesseurs
  - arbitres de bus
  - cohérence de caches
  - circuits asynchrones

Plusieurs défis à relever

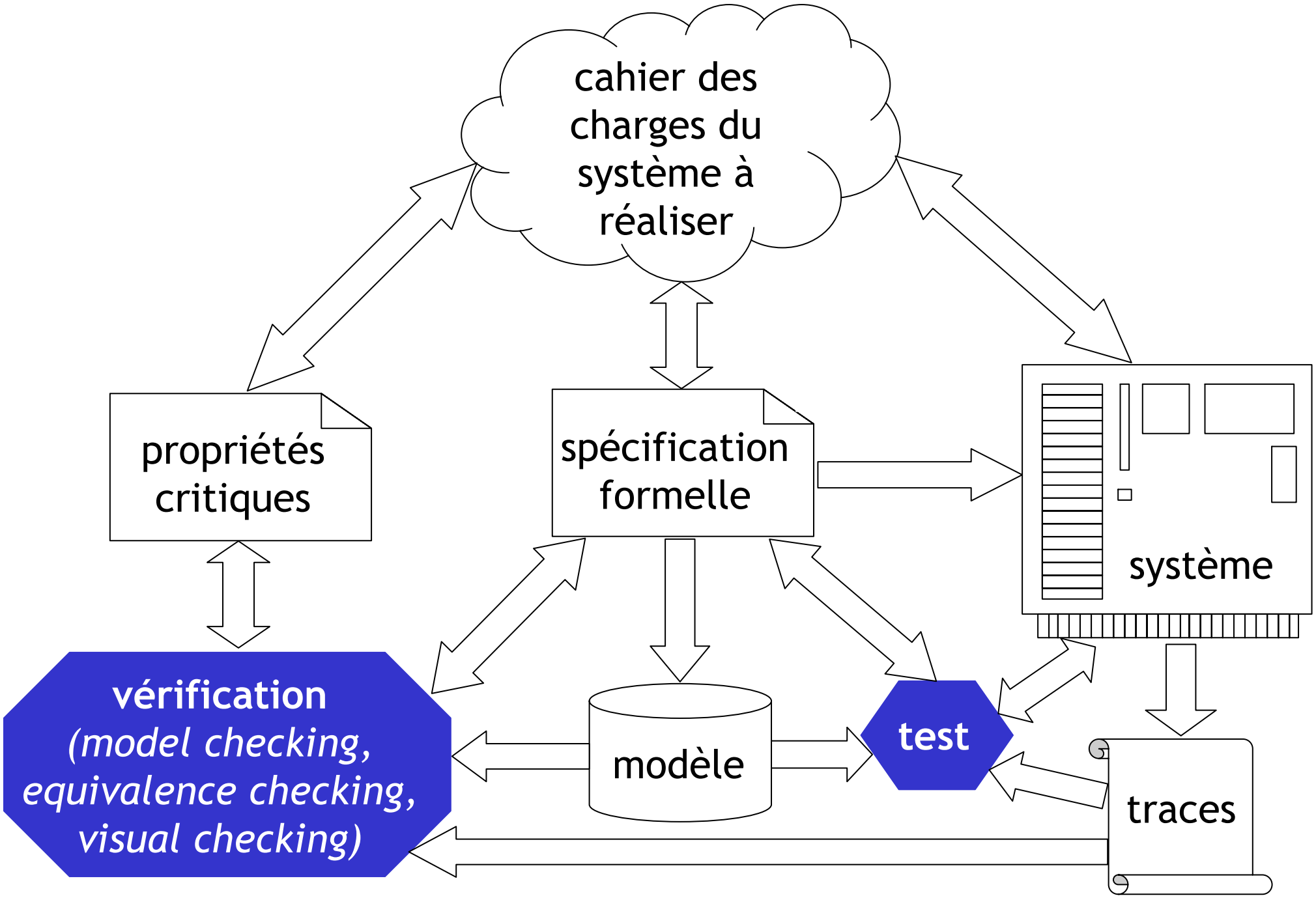
1. Complexité croissante des applications
2. Besoins de sécurité élevés
3. Difficultés de conception propres aux systèmes asynchrones:
   - problèmes d'algorithmique distribuée
   - présence d'indéterminisme
     test difficile, risque important d'erreurs résiduelles
   - complexité exponentielle des espaces d'états
     contrairement aux systèmes synchrones
   - absence de compositionnalité des "bonnes" propriétés
     contrairement aux programmes séquentiels (modules, objets)
     "*Correct by design*" reste hors d'atteinte

Erreurs et pièges à éviter…

- Ne pas reconnaître le caractère asynchrone d'un problème et tenter de le traiter avec les méthodes ordinaires
- Se reposer uniquement sur le test pour trouver les erreurs d'un système asynchrone (le coût peut être exponentiel)
- Se lancer tête baissée dans la programmation :
  Les langages séquentiels (sauf Ada) sont inadaptés pour penser le parallélisme
  - C/C++ : rien de prévu
  - Java/C# : modèle de mémoire centralisée
  - L'approche objet n'est pas forcément optimale

Méthodologie "standard"

- Avant de programmer : modéliser formellement
- Utiliser des outils de vérification/preuve
- Produire automatiquement certains tests
- Bénéficier du prototypage rapide
- Avantages escomptés :
  - délais de développement raccourcis
  - détection au plus tôt des erreurs
  - qualité accrue
  - meilleure documentation
  - maintenance plus aisée


cahier des charges du système à réaliser
propriétés critiques
spécification formelle
système
vérification (model checking, equivalence checking, visual checking)
modèle
test
traces

Difficultés de mise en oeuvre

A. Réticences fréquentes des industriels face aux méthodes formelles

B. Manque d'outils commerciaux couvrant tout le cycle "modélisation-vérification-test"

C. Multiplicité des langages de modélisation :
   - réseaux de Petri étendus
   - automates communicants (Estelle, SDL, Promela…)
   - algèbres de processus (LOTOS, CSP, muCRL, CHP, Tangram…)
   - UML (informel)

4 critères pour un langage optimal…

1. **Expressivité**
   Pouvoir modéliser simplement les caractéristiques essentielles des systèmes asynchrones :
   - données (types, variables, expressions, fonctions…)
   - contrôle (processus, communications, synchronisations…)
   - temps réel (délais d'attente, urgence…)
   - structure (modules, classes, bibliothèques…)

…4 critères pour un langage optimal

2. **Universalité**
   Convenir à tous les domaines d'activité où le parallélisme asynchrone intervient, sans être trop spécifique d'un secteur particulier
3. **Exécutabilité**
   Permettre la génération automatique de code efficace (pour simulation, prototypage rapide, test…)
4. **Vérifiabilité**
   Etre compatible avec les techniques de preuve et de vérification : posséder une sémantique formelle et de "bonnes propriétés" (abstraction, compositionnalité)

Les démarches qui réussissent

- Ne pas chercher à inventer une méthodologie générale pour les systèmes asynchrones
- Se concentrer sur un système concret et ses aspects critiques pour la fiabilité
- Planifier des tâches de modélisation et vérification
- Prévoir un temps de formation
- Prendre l'avis d'experts
  INRIA: algorithmique distribuée, modélisation formelle, vérification, preuve
- Choisir des outils robustes adaptés au problème

Travaux du projet VASY

- Conception de systèmes asynchrones fiables
- Axes de recherche :
  - Langages et compilation
  - Modèles et vérification
  - Développement d'outils
    CADP, TRAIAN
  - Etudes de cas industrielles
    Bull, Engineering, MGE-UPS, Scalagent…

Boîte à outils CADP

Construction and Analysis of Distributed Processes

- Modélisation formelle
  - algèbres de processus
- Mise au point
  - génération de code C
  - simulation interactive
  - simulation aléatoire
  - prototypage rapide
- Vérification
  - *visual checking*
  - *equivalence checking* (bisimulations)
  - *model checking* (mu-calcul)
  - vérification à la volée
  - vérification compositionnelle
- Génération de tests
  - outil TGV (projet VERTECS)

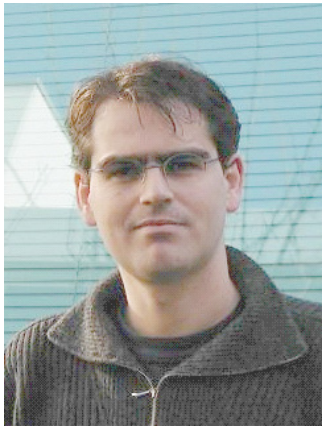
La boîte à outils CADP

- Architecture modulaire et extensible (APIs)
- Plusieurs plateformes supportées
  - Sun/Solaris, PC/Linux, PC/Windows
- Diffusion internationale
  - contrats de licence avec 310 organisations
  - en 2003: installation sur 840 machines
- Nombreuses applications
  - 72 études de cas effectuées avec CADP
  - 16 outils de recherche connectés à CADP
  - 17 cours universitaires utilisant CADP

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

Pour plus d'information…

Radu Mateescu



Frédéric Lang

démos CADP

VASY

http://www.inrialpes.fr/vasy