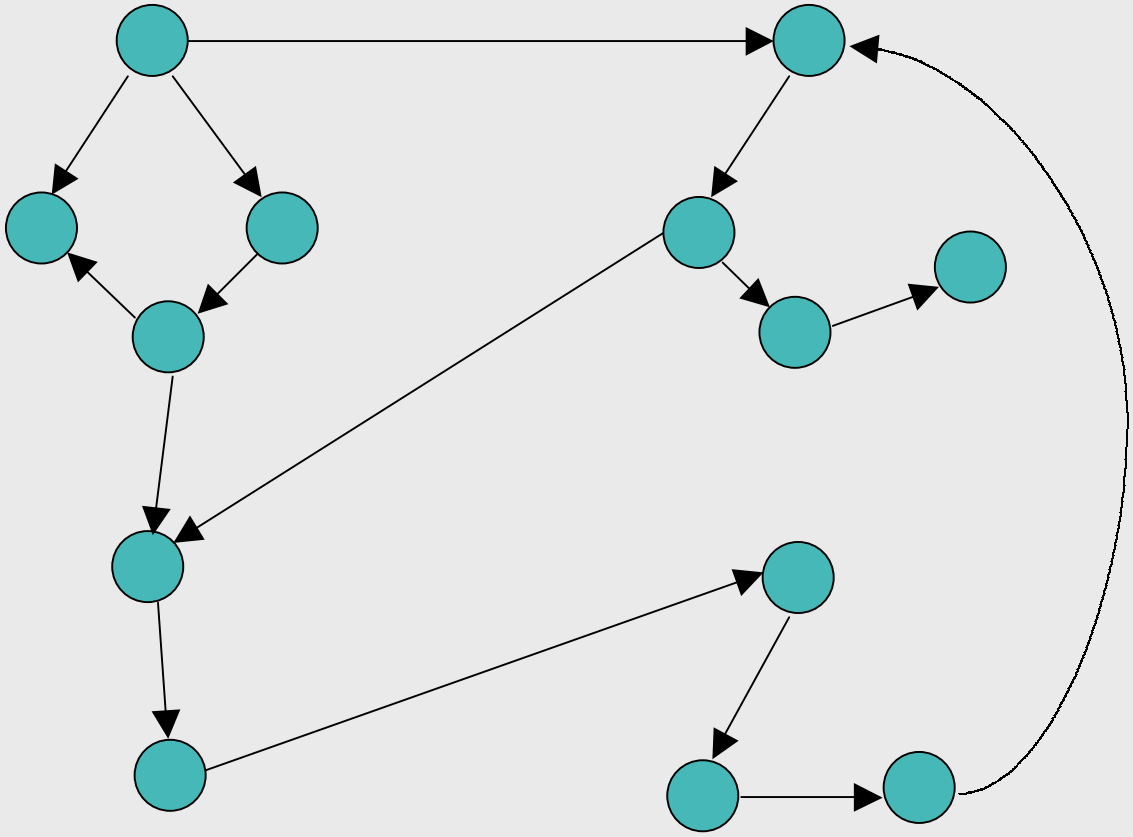
Distribution et parallélisation automatique de programmes objets (Java)

Pascal Grange
Équipe Systèmes et Objets Distribués
LaBRI
grange@labri.u-bordeaux.fr

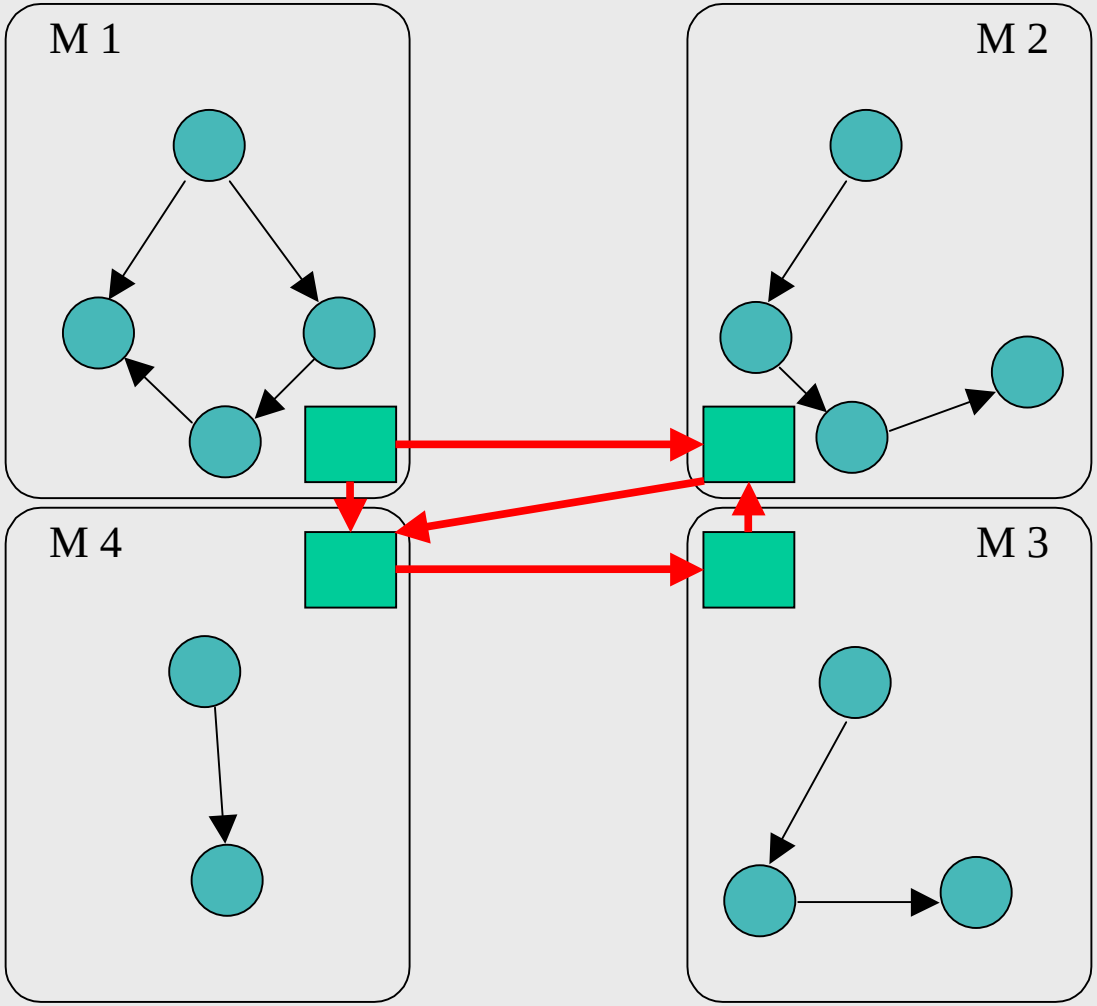
Plan de la présentation

- Application à objets distribuée
- Le passage de paramètres par copie
- Intérêts d'une mémoire virtuellement partagée
- Travaux réalisés
  - Distribution d'application
  - Parallélisation d'application
- Conclusion et perspectives

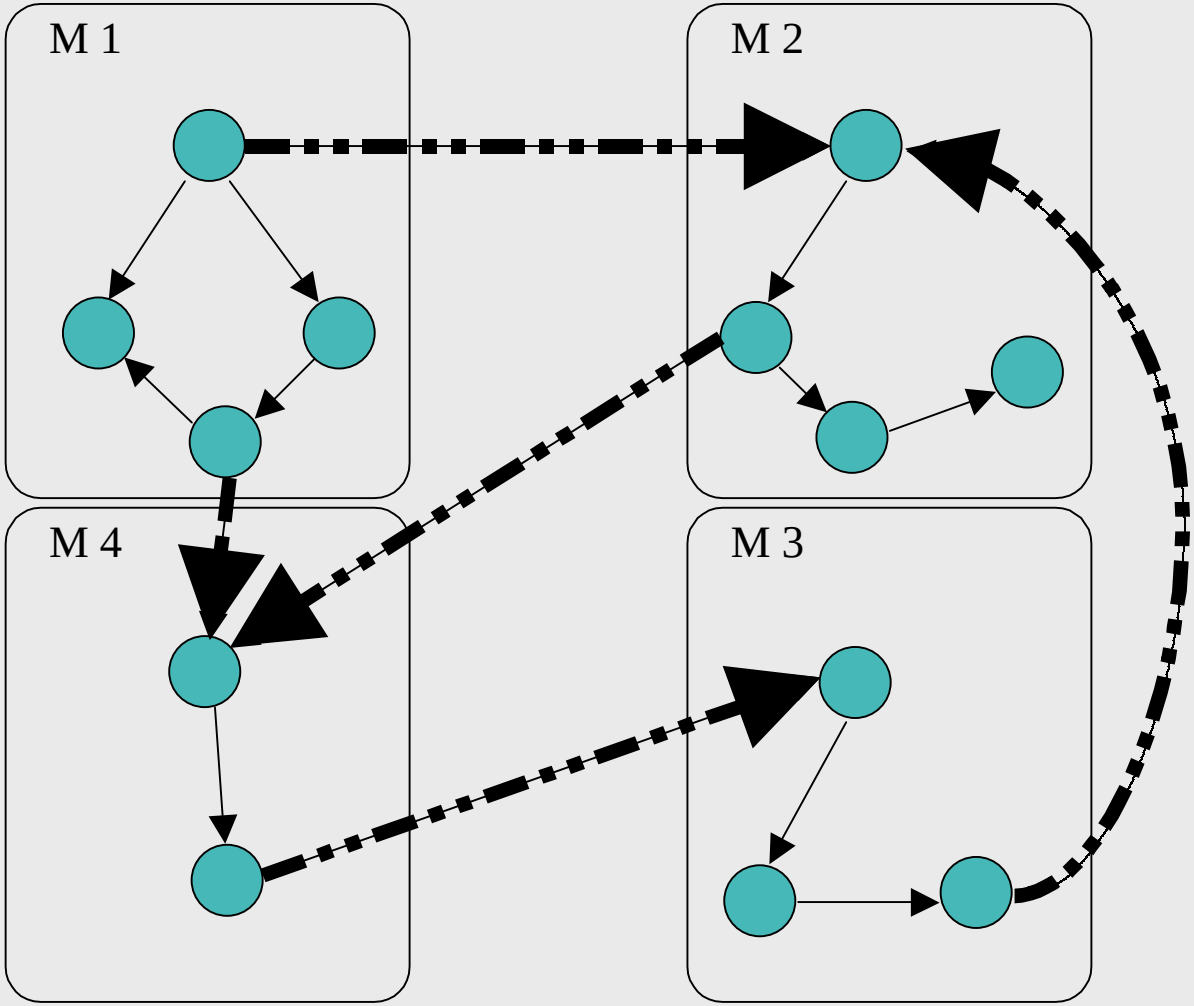
Une application orientée objets



Une application distribuée (processus communicants)



Une application orientée objets distribuée



Écriture d'applications orientées objets distribuées

Bibliothèques

- CORBA
- **RMI (Java)**

Compilateur + bibliothèque:

- JavaParty
  - Un compilateur pour le langage JavaParty (java + mots clés comme « Remote »)
  - Une bibliothèque pour les commandes de distribution pures, i-e création d'objets à distance, migration, etc., …

Les approches « à la RMI »

- Déclaration d'objets (classes) distants
- Invocation de méthodes sur ces objets
  - via le réseau
  - paramètres passés par copie[1]
  - résultat envoyé par copie[1]
- Pendant des RPC pour l'objet
- Problème du passage par copie
  - pas de polymorphisme
  - Pas (ou peu) de réutilisation de code

1 : sauf les paramètres ou résultats distants

Une autre approche : mémoire virtuellement partagée

- Séparation entre
  - Architecture logicielle
  - Placement des données
- Polymorphisme habituel
  - Réutilisation de code complète
- Développement, mise au point, etc. classique
  - Facilite le travail du programmeur
- Sécurité accrue
  - Programme correct → correct en distribué (virtuellement partagé)
- Politiques de placement variées et adaptées
  - Réplication
  - Fractionnement
  - Migration

Mémoire virtuellement partagées : Plusieurs approches

Plate-forme d'exécution

- politiques de cohérence de données

Compilation

- HPF
  - programme + directives de distribution de données
- ADRIAN
  - Écriture d'un programme en mémoire partagée
  - Compilation vers la mémoire distribuée

Plan de la présentation

Application à objets distribuée

Le passage de paramètres par copie

Intérêts d'une mémoire virtuellement partagée

⇒ Travaux réalisés

- Distribution d'application
- Parallélisation d'application

Conclusion et perspectives

Adrian, vers une distribution automatique

Fonctionnement en 4 phases

- Développement et mise au point classique
  - Réutilisation de code
  - Utilisation d'outils rodés pour la mise au point
- Analyse statique du code et détection des objets
  - Basé sur pangaea (André Spiegel, Berlin)
  - Programme objet = graphe d'objets + relations
  - Approximation du graphe par analyse du code
- Placement (statique) de ces objets
  - Détection de choix permettant l'ajout de parallélisme
- Génération du programme distribué

Analyse du code et approximation du graphe

- Calcul de dépendance entre les types du programme
- Analyse des instructions d'instanciation
  - Objets concrets
  - Objets indéfinis
- Calcul des dépendances entre objets
- Vision statique de toute l'exécution

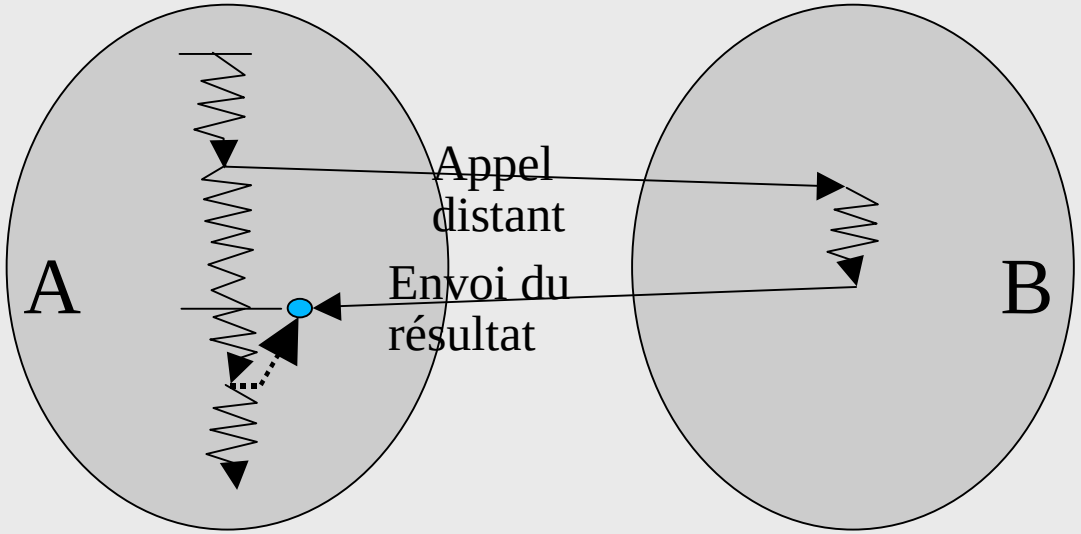
Aide au choix de distribution

Optique visée : exploitation du parallélisme

Étude des tâches du programme

- Quelles tâches partagent quels objets ?

Appels distant asynchrones : objets actifs



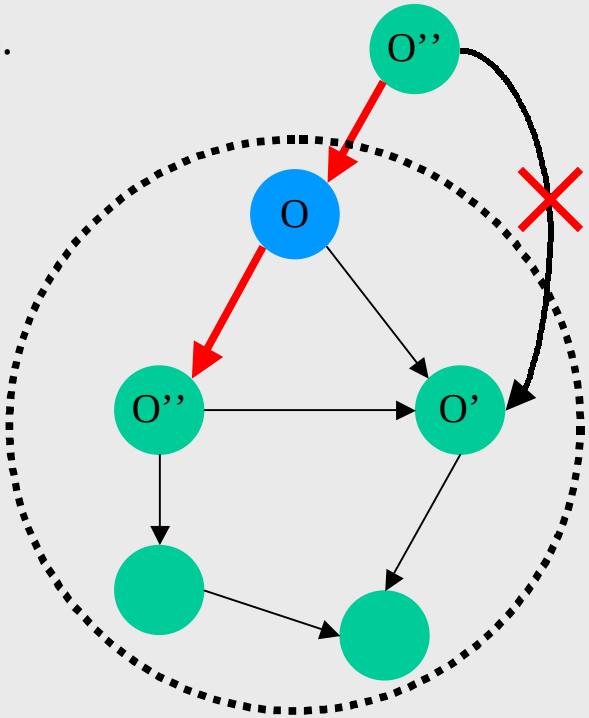
Objets actifs : propriété d'activabilité

Propriété dans le cas séquentiel (basée sur OASIS, Nice)

Un objet **o** dans le graphe d'Adrian est activable si, quelque soit o' « utilisé » par **o**, quelque soit o'', si o'' utilise o' alors,

soit o'' est utilisé par **o**,

soit o'' utilise o' « à travers » **o**.



La propriété d'activabilité (suite)

Propriété d'activabilité dans le cadre multitâche

Un objet est activable dans un programme multitâche si il respecte la propriété précédente ET

Si il n'est utilisé que par **une seule thread**

Limitation

Le programme doit être « localement séquentiel »

MAIS l'objet activable peut lui-même utiliser d'autres tâches

Validation (preuve en PI-calcul)

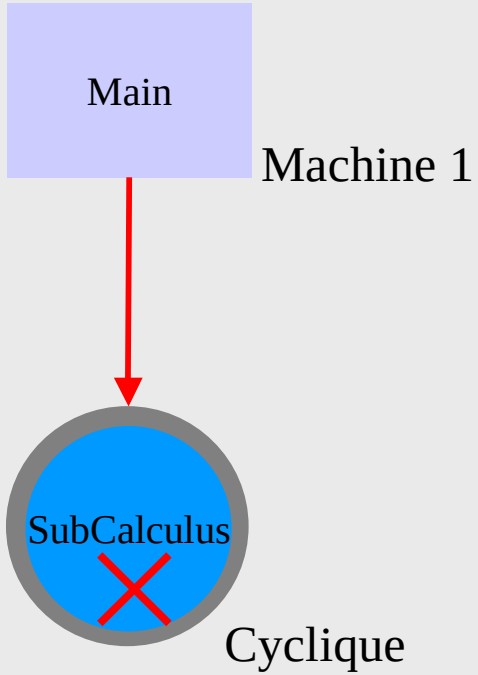
montrer que le processus PI-calcul qui modélise un objet « normal » est en bisimulation (faible) avec le processus qui modélise cet objet activé.

Un exemple : le calcul de PI (2)

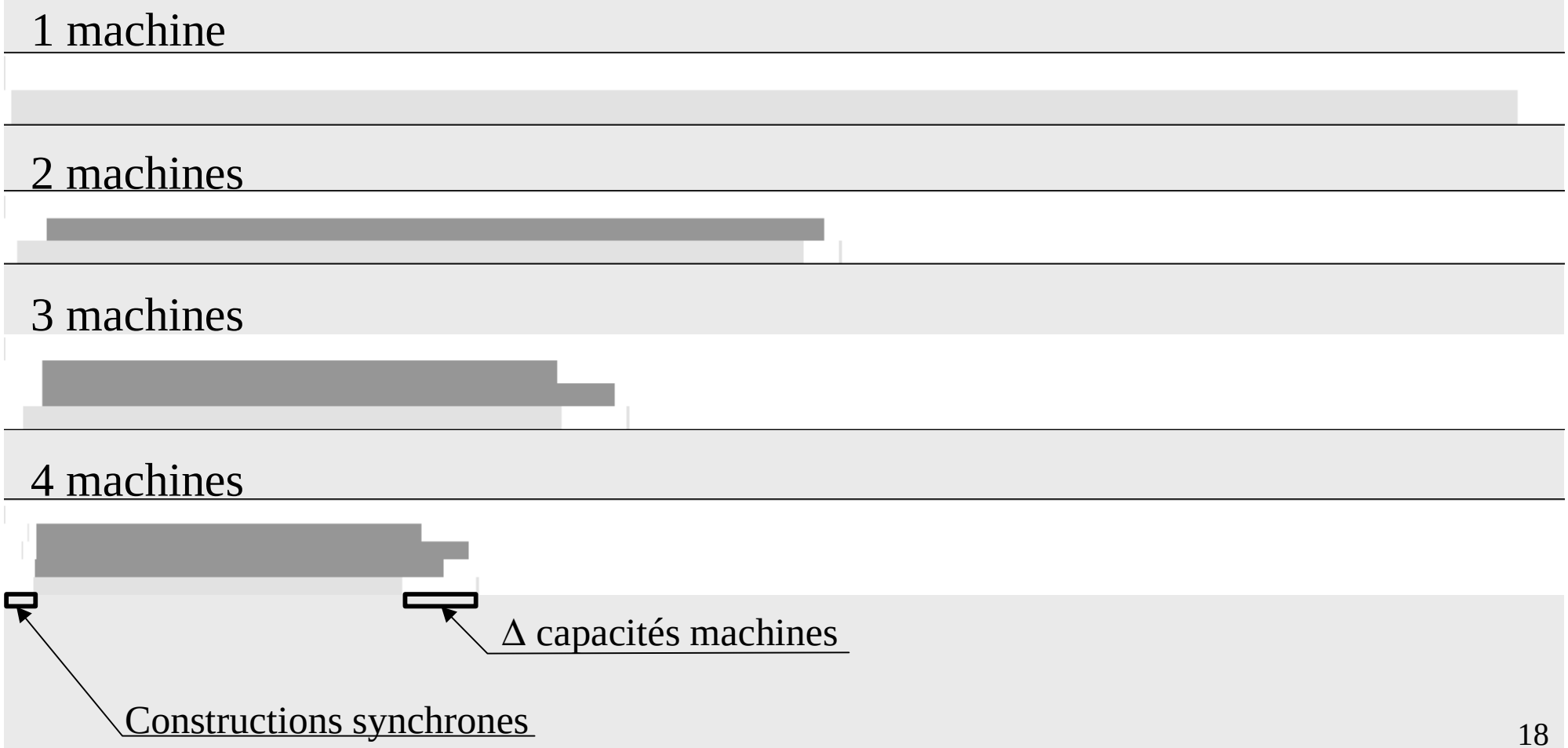
```
public class Main{
   public static void main(String[] args){
      int nbSubCalculus = Integer.parseInt(args[1]);
     SubCalculus subs[] = new SubCalculus[nbSubCalculus];
     Result results[] = new Result[nbSubCalculus];
      for(int i = 0; i < nbSubCalculus; i++){
          subs[i] = new SubCalculus(start, end);
       }
       for(int i = 0; i < nbSubCalculus; i++){
          results[i] = subs[i].compute();
       }
       for(int i = 0; i < nbSubCalculus; i++){
          pi  = pi.add(results[i].getValue());
       }...
```

```
public class SubCalculus{
    public Result compute(){
        ...
    }
}
```

Résultat de l’analyse



Calcul de PI (1000 décimale)



Conclusion

- Nouveaux résultats basés sur deux travaux :
  - Pangaea (université de Berlin)
  - Technique de parallélisation (OASIS, NICE)
- Utilisation de Jacob
- Gestion des ressources (threads) peu efficace
- Certains points ont été mis de coté
  - Exceptions asynchrones
  - Réflexion
- Placement graphique → petits exemples

Perspectives

- Amélioration de la gestion des ressources
- Création asynchrone d'objets actifs
  - Implications théoriques
  - Implémentation pratique
- Recherche de nouvelles techniques d'analyse
  - Activation
    - Plusieurs threads vers un objet activable
    - Invocation de méthodes au plus tôt
  - Graphe d'objets
    - Vision moins statique
- Expression du placement moins statique
  - séparation programme/placement