# *Vers le pilotage des simulations numériques sur la grille*

***Renpar'15 – la Colle sur Loup, octobre 2003.***

M. Dussere & A. Esnard

`http://www.labri.fr/epsn`

Projet EPSN (ACI-GRID PPL02-03)

INRIA Futurs (projet ScAlApplix) & LaBRI (UMR CNRS 5800)

351, cours de la Libération, F-33405 Talence, France.

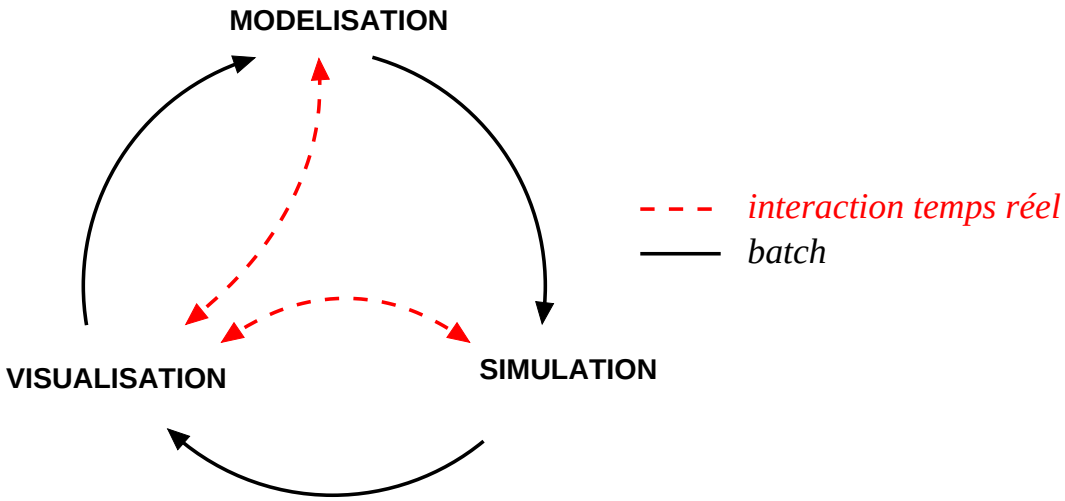
# Plan

1. Introduction
2. EPSN (Environnement pour le Pilotage de Simulations Numériques)
3. Le prototype Epsilon
4. Systèmes de pilotage
5. Extension au parallélisme
6. Conclusion & Perspectives

# Introduction

# *Simulation & Visualisation Scientifique*

Visualisation scientifique $\rightarrow$ analyse des données produites par les simulations numeriques




## Batch

- approche traditionnelle
- une chaîne de traitement séquentiel
- la visualisation comme une étape de post-traitement

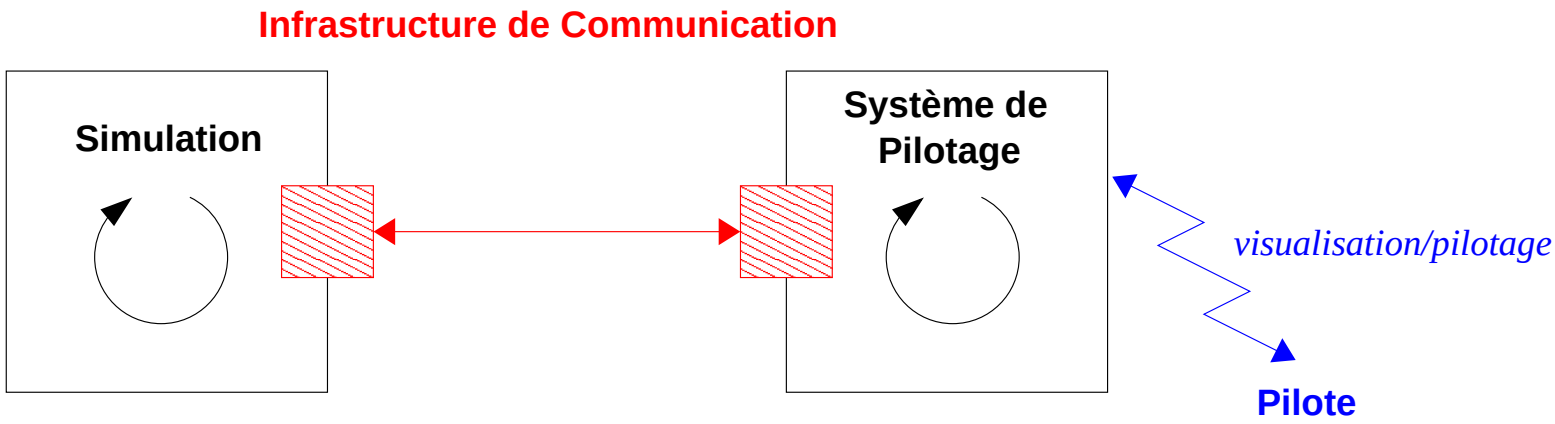
## Simulation interactive (computational steering)

- interaction à la volée avec la simulation
- visualiser les résultats en “temps réel”
- une meilleure compréhension du modèle

# Environnements de pilotage

## Composantes de l'environnement

- simulation numérique
- système de pilotage (visualisation/pilotage)
- infrastructure de communication



# Travaux reliés

## Les modèles de programmation pour la simulation interactive

- débogueur (-g)
  → *dégradation des performances*
- interaction avec les entrées/sorties standards (CaveStudy)
  → *du “batch” évolué*
- “wrapping” d’interface avec des langages scripts (SWIG)
  → *light weight steering*
- Problem solving environment (SCIRUN, COVISE)
  → *construction d’une nouvelle application pilotable, programmation visuel*
- instrumentation (CUMULVS, Falcon, VIPER...)
  → *modification des sources (appels de fonctions d’une librairie)*

# Travaux reliés

## Les infrastructures de communication

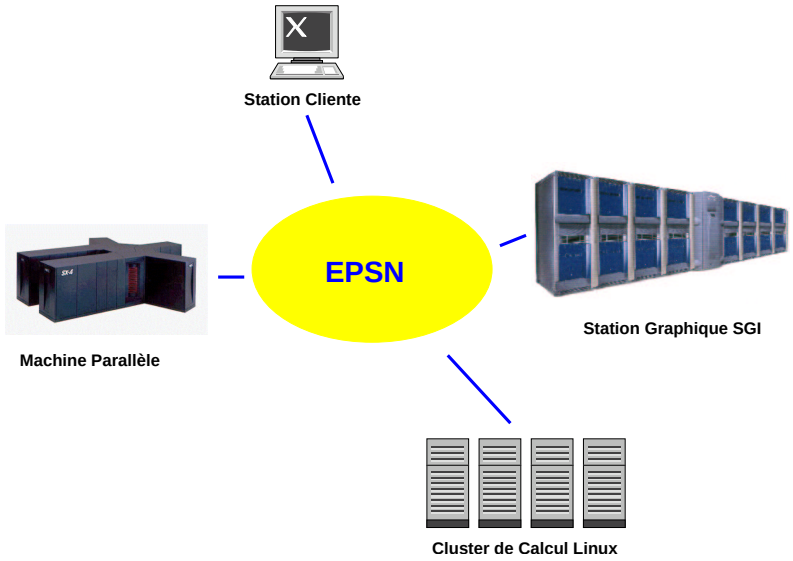
- sockets TCP/IP (POSSE, COVISE, Visit)
- RPC + XDR (VIPER)
- *DataExchange* (Magellan, Progress, Falcon, MOSS)
  $\rightarrow$ *un système d'événements, hétérogène*
- Nexus (CAVEStudy)
  $\rightarrow$ *couche de communication de GLOBUS*
- PVM (CUMULVS)

# EPSN

# Introduction à EPSN

## Objectifs

- développer un environnement logiciel pour le pilotage des simulations
  → *contrôle, exploration du modèle, modification*
- supporter des applications parallèles distribuées
  → *environnements hétérogènes caractéristiques des grilles*
- offrir de bonnes performances (peu de perturbations sur le calcul)



# Introduction à EPSN

## Approche

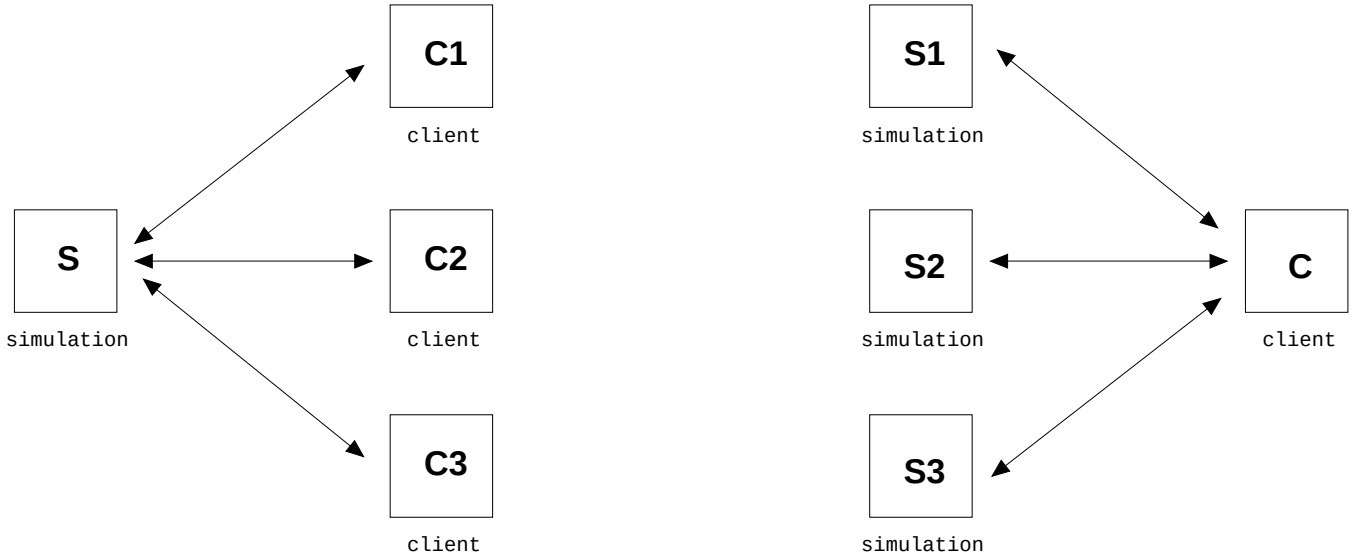
- un couplage générique basée sur **une représentation abstraite** de la simulation
  → *description XML de la simulation interactive*
- **instrumentation** du code peu intrusive
  → *couplages précis et performant*
- infrastructure de communication basée sur **CORBA**
  → *masquer la complexité de CORBA*

## Quelles simulations ?

- codes de simulation C/C++/Fortran
- processus itératifs (boucle de calculs)
- séquentiels → parallèles (SPMD) → code parallèles distribués (MPMD)
- divers paradigmes : MPI, PVM, Posix, OpenMP, CORBA, etc.

# *Infrastructure de communication*

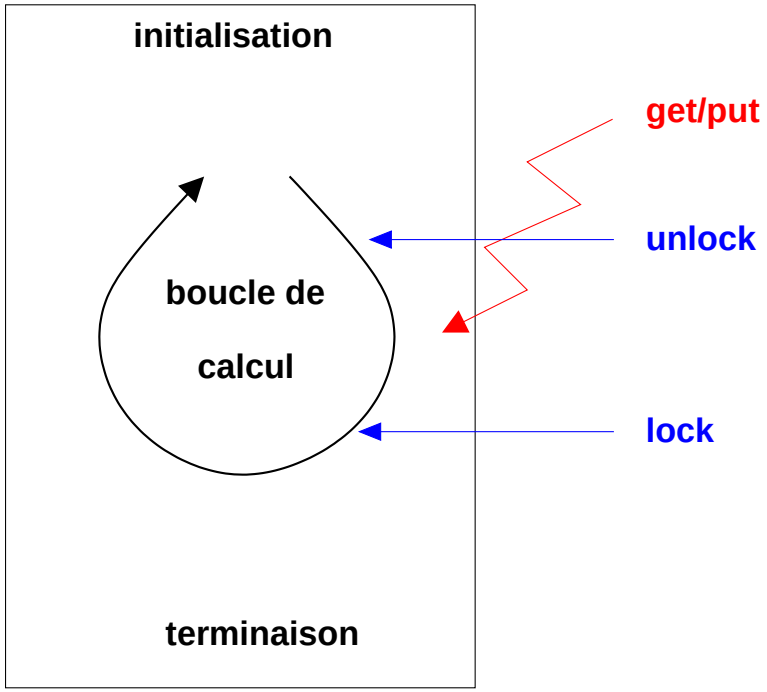
- une approche **client/serveur**
  → *simulation = serveur, visualisation = client*
- une infrastruture de communication **distribuée**
  → *multi-clients & multi-applications*
- une infrastruture de communication **dynamique** et robuste
  → *attachement/détachement des clients en cours d'exécution*



# *Infrastructure de Communication*

## ☐ Modèle de communication

- communication “one-sided” (get/put)
- plage d’accessibilité (lock/unlock)
  → ***accès cohérent*** *aux données*
- communication **asynchrone**
  → *mise en oeuvre du recouvrement*
- un **thread dédié** à la communication
  → *faibles perturbations de la simulation*
- accès aux données via la mémoire partagée
  → *envoi* ***zéro-copie***



# Architecture

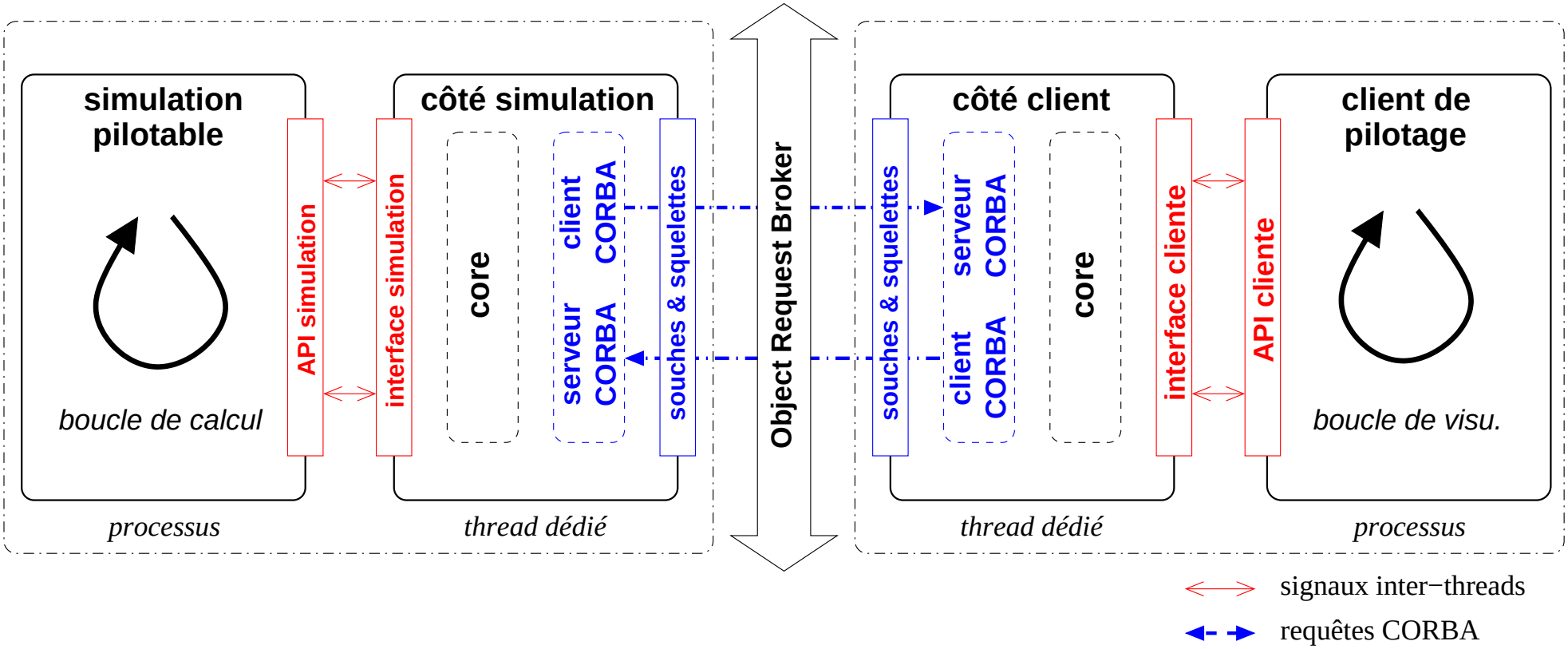
## Le choix de CORBA

- modèle de programmation avançé (client/serveur, RPC, orienté objet)
- transfert de données hétérogènes (CDR)
- transparence réseaux, portabilité, interopérabilité (systèmes/langages)
- utilisation du service de nommage pour le déploiement

## CORBA dans EPSN

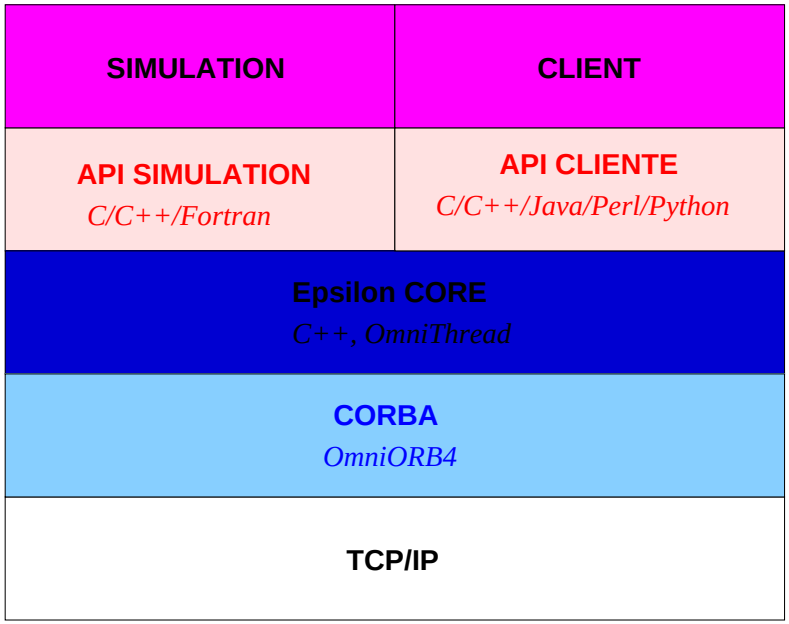
- CORBA masqué aux utilisateurs finaux
- un serveur CORBA embarqué dans le thread EPSN
- un pool de threads CORBA pour la gestion des requêtes (maximiser la concurrence)
- synchronisation inter-threads basé sur des signaux et des sémaphores
- requêtes CORBA asynchrones (*oneway*)

# Architecture



# Le Prototype Epsilon

# Le prototype Epsilon




- 2 bibliothéques de fonctions (API client et API simulation)
- ORB performant (omniORB4)
- Parser XML DOM (libxml2) + DTD

# Description XML d'une simulation

- structure du document XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE simulation SYSTEM "Epsilon.dtd">
< simulation  name="mysimu"  context="sequential">
  < control  running="true"> ... < /control>
  < data> ... < /data>
< /simulation>
```

- les points d'arrêts

```
< breakpoint  id="mybreakpoint"  state="down"/>
```

- les boucles de calcul (associé à un compteur)

```
< loop  id="myloop"  state="up"/>
```

- les données scalaires

```
< scalar  id="myscalar"  access="readonly"  type="double"  label="my label"/>
```

- les tableaux multi-dimensionnels

```
< sequence  id="mysequence"  access="readwrite"  type="long">
  < dimension  size="3"/>
  < dimension  size="variable"/>
< /sequence>
```

- les groupes de données corrélées logiquement (consistance)

```
 < group  id="mygroup"> ...  < /group>
```

# L'exemple de FingerPrint

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE simulation SYSTEM "Epsilon.dtd">

< simulation  name="fingerprint"  context="sequential">
  < control  running="true">
    < breakpoint  id="begin"/>
    < loop  id="deflation"  state="up"/>
    < breakpoint  id="end"  state="down"/>
  < /control>

  < data>
  < group  id="mesh" >
    < scalar  id="nb_nodes"  access="readonly"  type="long"/>
    < scalar  id="nb_cells"  access="readonly"  type="long"/>
    < sequence  id="nodes"  access="readwrite"  type="double">
      < dimension  size="3"/>
      < dimension  size="nb_nodes"/>
    < /sequence>
    < sequence   id="cells"  access="readwrite"  type="long">
      < dimension  size="3"/>
      < dimension  size="nb_cells"/>
    < /sequence>
    < sequence  id="values"  access="readwrite"  type="double">
      < dimension  size="nb_nodes"/>
    < /sequence>
  < /group>
  < /data>
< /simulation>
```

# Les 2 APIs

## API côté simulation

- initialisation/terminaison (`init/exit`)
- publication des données (`publish`)
- placement des points d'arrêts (`barrier`)
- plages d'accessibilité (`lock/unlock`)
- évolution des données (`release`)
- évolution des boucles de calcul (`iterate`)
- forcer les envois de données (`flush`)
- etc.

## API côté client

- initialisation/terminaison (`init/exit`)
- contrôle (`play/step/stop`)
- enregistrement des données (`register/allocate`)
- extraction de données "à la volée" (`get/getnext/wait/test`)
- modification des données (`put`)
- extraction régulière des données (`getp/cancelp`)
- plages d'accessibilité (`lock/unlock/receive`)
- etc.

# Instrumentation de FingerPrint

```
// Initialisation de FingerPrint
int nb_nodes, nb_cells;
int *nodes, *cells; double *values;
initMoleculeAndBuildMesh("crambin.pdb");

// Initialisation d'Epsilon
epsilon_init("fingerprint.xml");

// Publication des données
epsilon_publish("nb_nodes", &nb_nodes);
epsilon_publish("nb_cells", &nb_cells);
epsilon_publish("cells", cells);
epsilon_publish("nodes", nodes);
epsilon_publish("values", values);
epsilon_publishgroup("mesh");

epsilon_unlockall();
epsilon_barrier("begin");
```

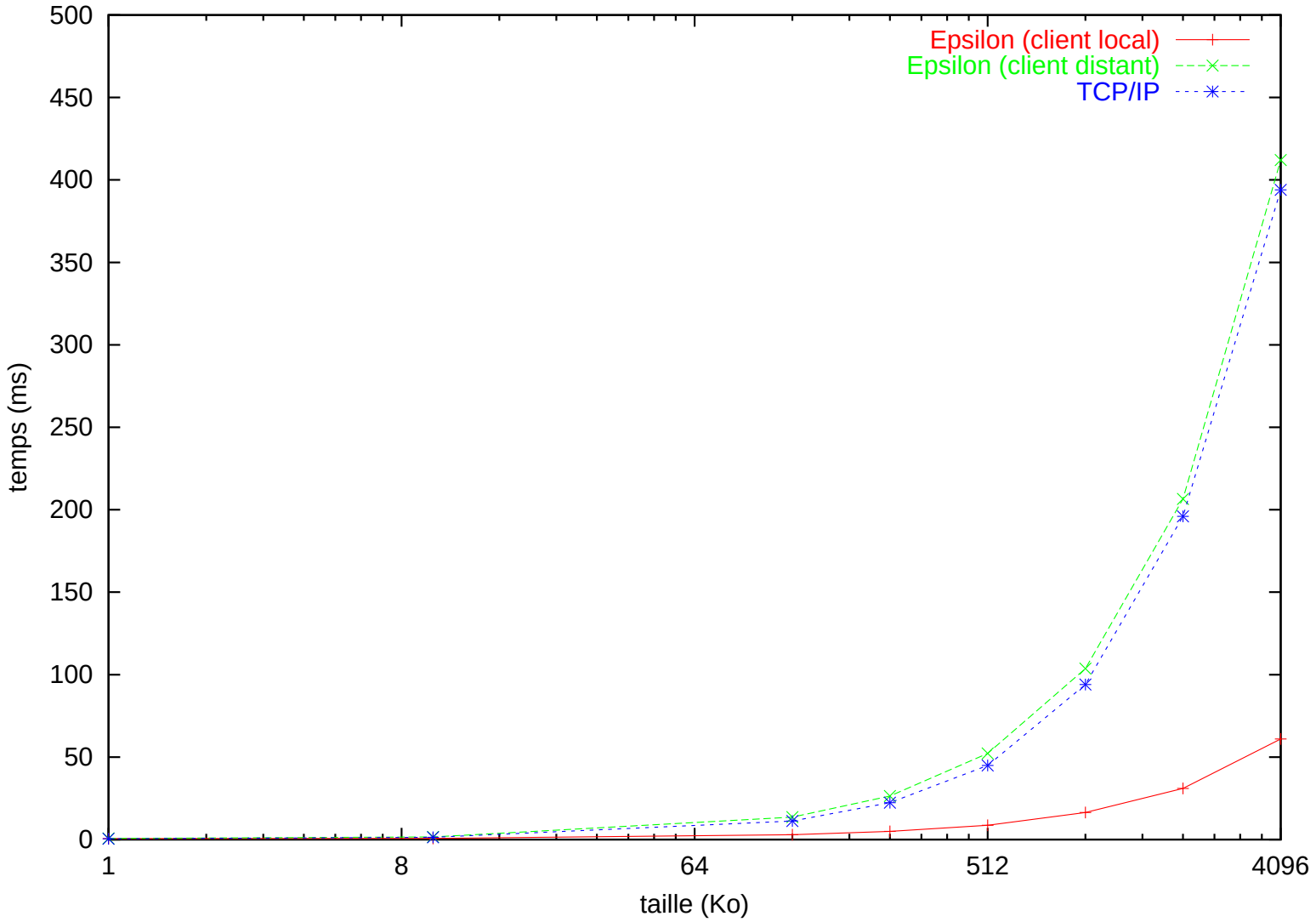
```
// Tant que le maillage ne colle pas à la surface
while(deflatTest()) {
  epsilon_iterate("deflation");
  epsilon_lock("nodes");
  moveNodes();  // déplacement des noeuds
  epsilon_unlock("nodes");
  epsilon_release("nodes");
  evaluateMesh();  // evaluation du maillage
  epsilon_lock("mesh");
  correctMesh();  // correction du maillage
  epsilon_unlock("mesh");
  epsilon_release("mesh");
  epsilon_flush("mesh");
}

// Construction de l'empreinte
makePrint();
epsilon_barrier("end");
epsilon_exit();
```

# Client de visualisation pour FingerPrint

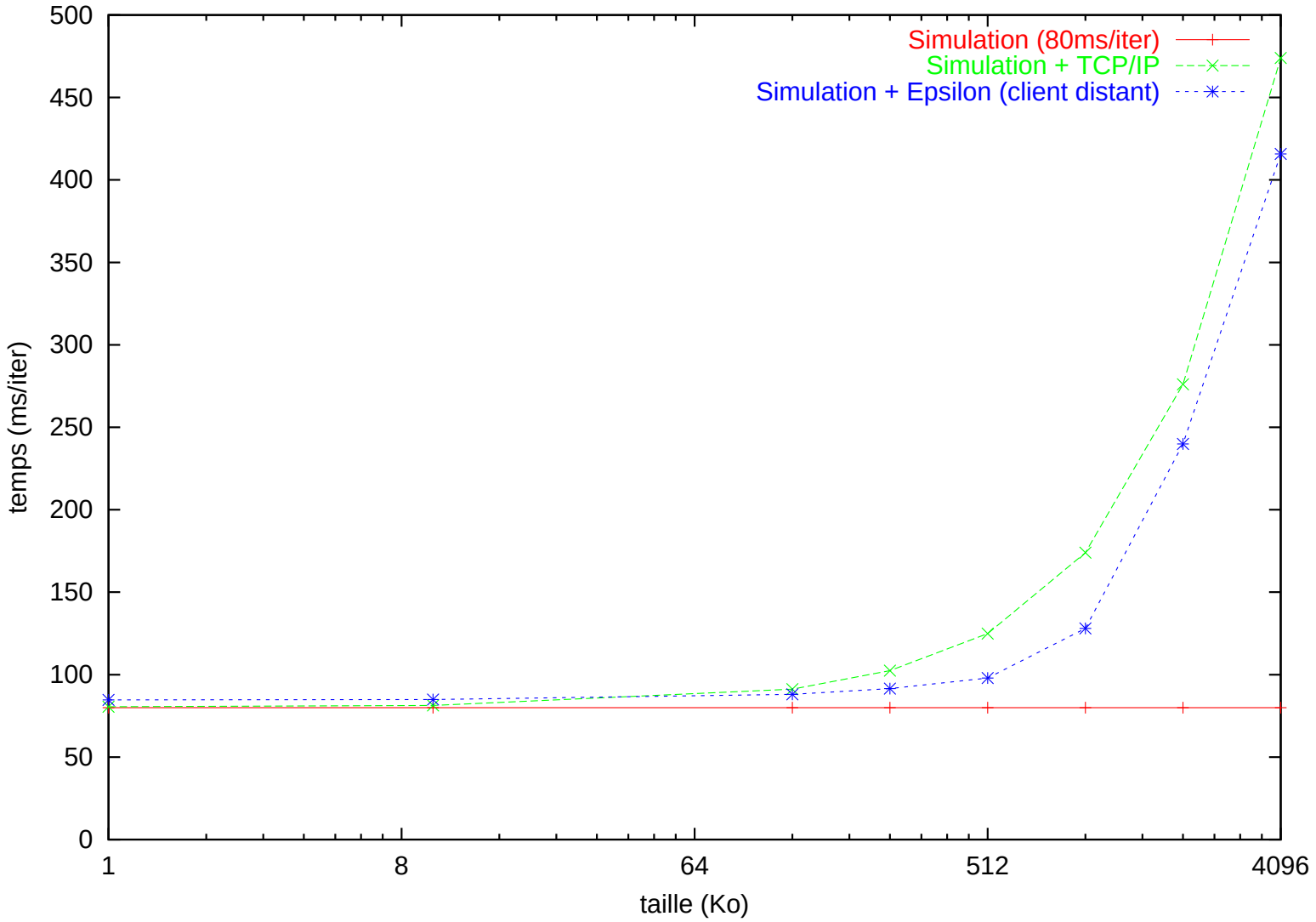
```
int nb_nodes, nb_cells;
int *nodes, *cells; double *values;

// Initialisation de Epsilon
epsilon_init("fingerprint","localhost","2809");

// Enregistrement des données
epsilon_register("nb_nodes",&nb_nodes,NULL);
epsilon_register("nb_cells",&nb_cells,NULL);
values=(double*) epsilon_allocate("values",NULL);
nodes=(int*) epsilon_allocate("nodes",NULL);
cells=(int*) epsilon_allocate("cells",NULL);
epsilon_register_group("mesh");

// Récupération du maillage initial
epsilon_getall();
epsilon_unlockall();
epsilon_waitall();
buildMeshImage();
```

```
// Recupération du maillage à chaque itération
epsilon_lockall();
epsilon_getp("mesh",1,EPSILON_FLUSH_ON);

// Boucle principale de visu.
do {
   // Si une nouvelle version est arrivée
   if( epsilon_receive("mesh"))
     updateMeshImage();
   // ...

} while(!done);


epsilon_exit();
```

# Temps de communication



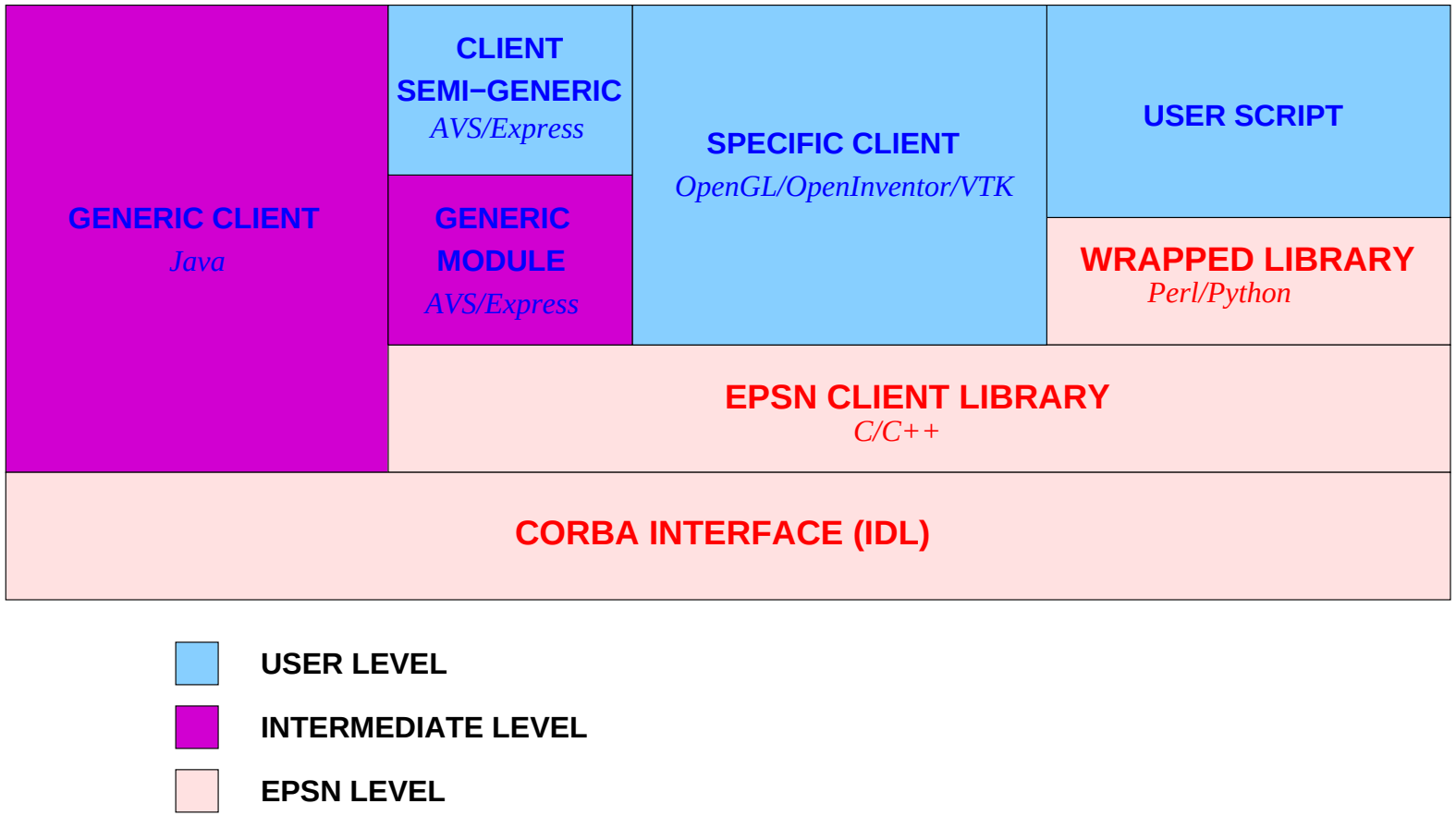
# Extraction des données d'une simulation



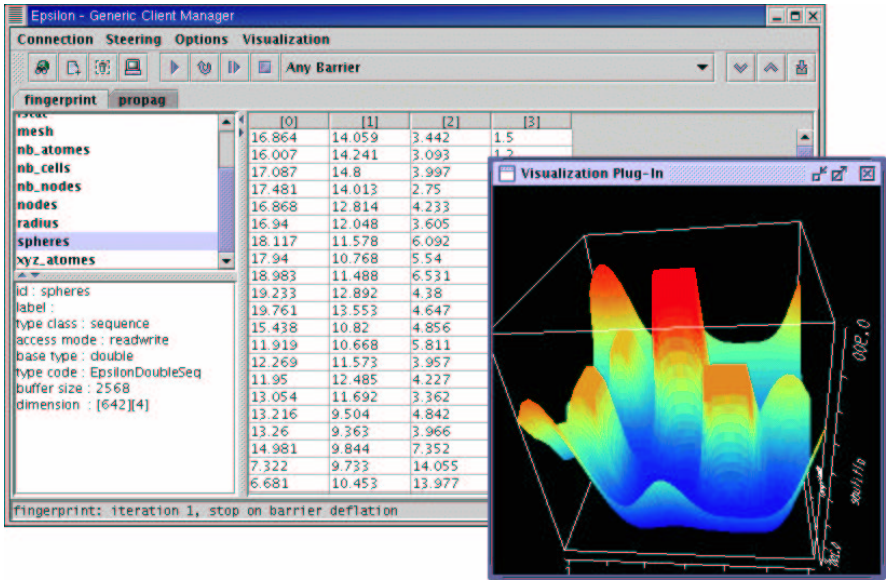
# Systèmes de Pilotage

# Systèmes de pilotage

- Différentes stratégies pour interagir avec une simulation EPSN

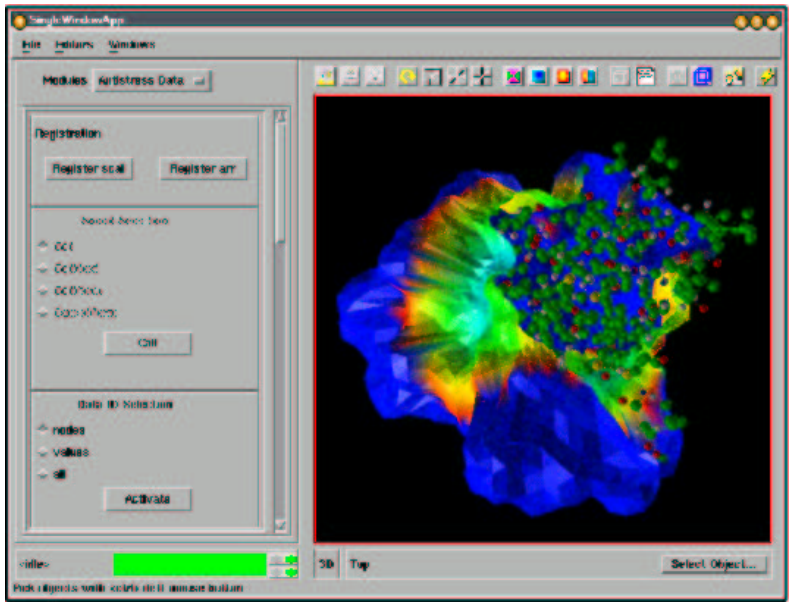


# Client générique EPSN




- implanté en Java/Swing
- contrôle et accès aux données pour les simulations instrumentées avec EPSN
- présentation des données au travers des feuilles numériques
- quelques plug-ins visualisation en Java3D, Visad...

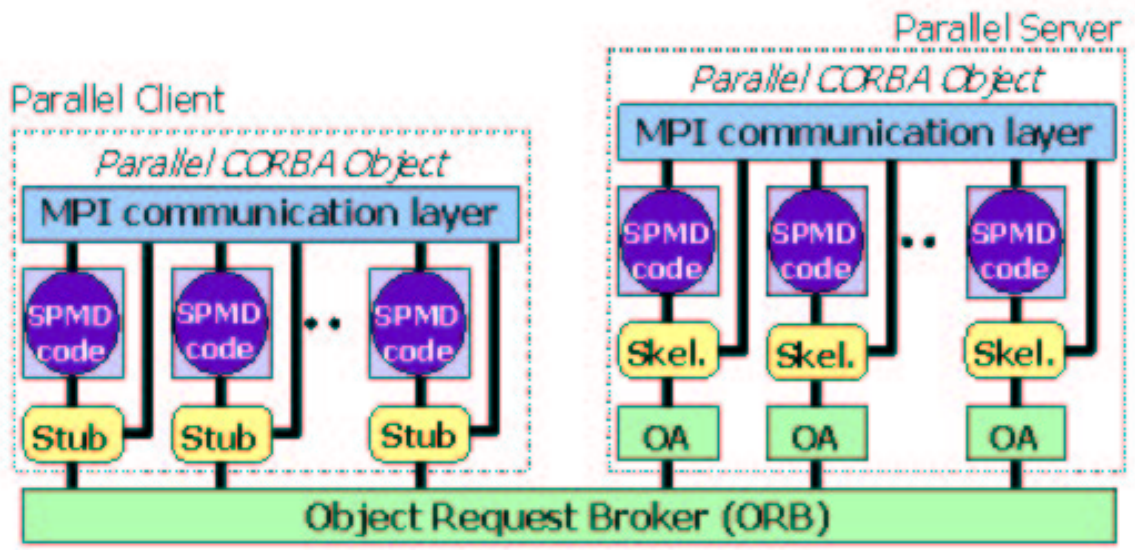
# *Client semi-générique AVS/Express*



- dédiés à la visualisation d'une classe de problème (maillages, molécules, etc.)
- définition de modules génériques EPSN pour le contrôle et l'acquisition de données
- interconnection de ces module selon un modèle *data flow*
- e.g. visualisation de maillage dans AVS/Express

# Extension au Parallélisme

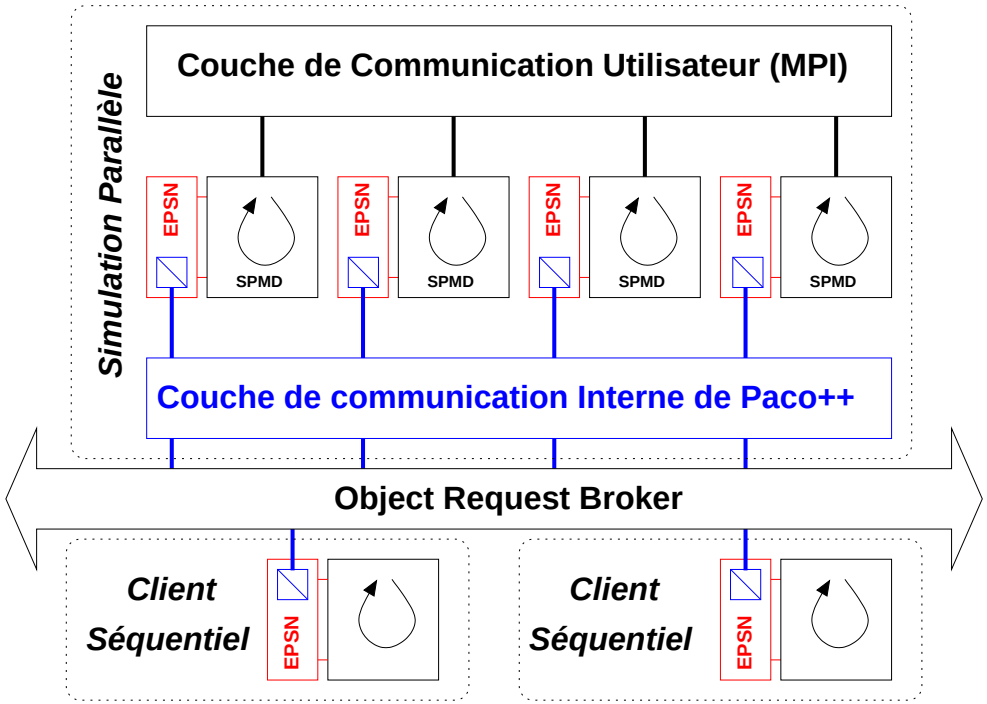
# *Les objets parallèles CORBA (PaCO++)*




*Paris Project (IRISA)*

- collection d'objets CORBA identiques
- pas de modification de l'ORB $\rightarrow$ portabilité
- redistribution transparente des données (description XML + IDL)
- couche de communication interne performante (e.g. MPI)

# Architecture parallèle avec PaCO++




- enrichissement du XML pour prendre en compte la distribution
- extension de la notion de cohérence
- instrumentation de chaque processus de la simulation SPMD
- synchronisation via la couche interne

# *Prototype parallèle 100% CORBA*

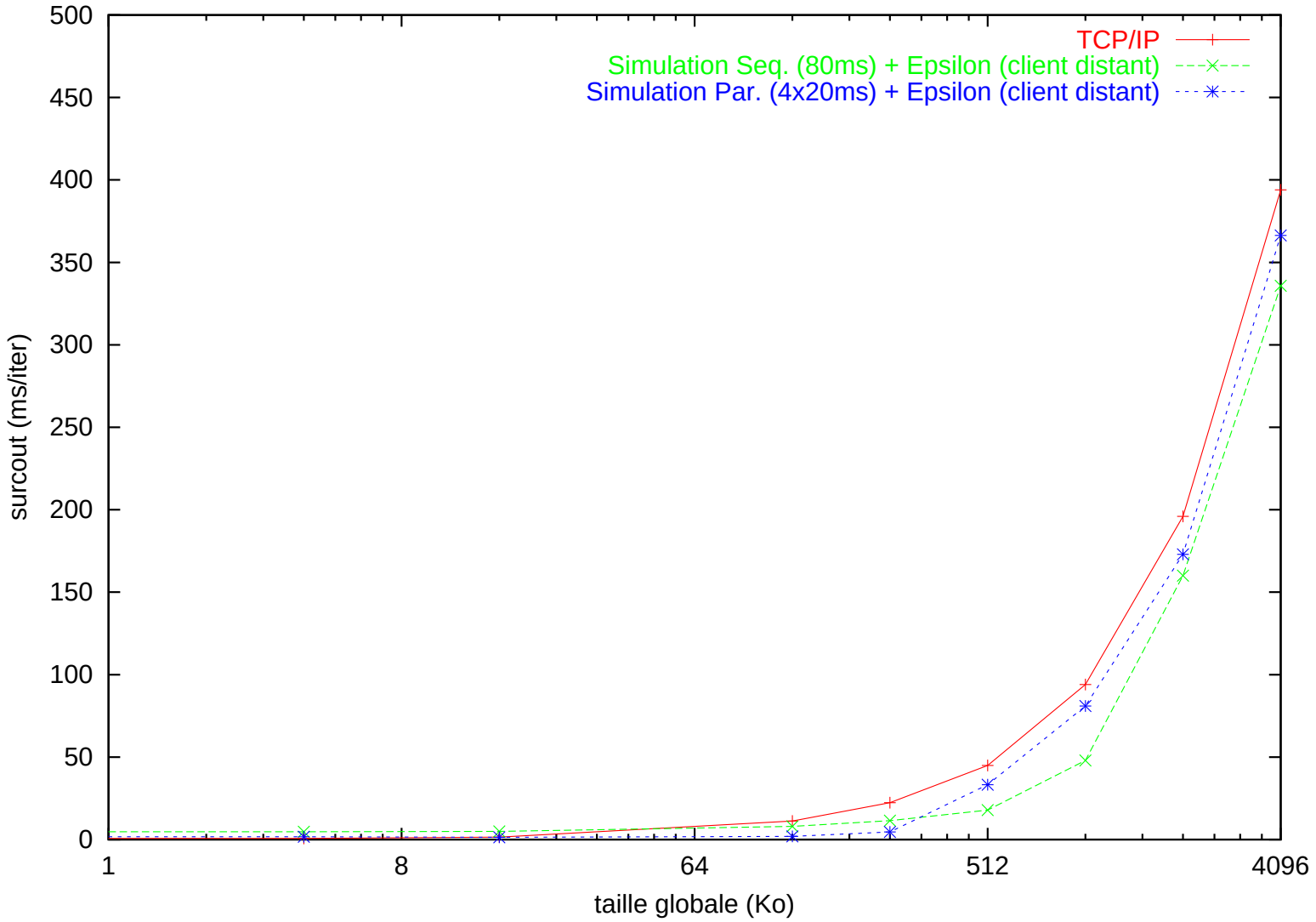
## Solution 100% CORBA (contrôle, accès aux données)

- développement de mécanisme de synchronisation
  → *le contrôle (play/stop en parallèle)*
  → *la consistance “temporelle” des accès (get en parallèle)*
- inconvénients: n’exploite pas efficacement l’infrastructure parallèle (MPI)
- appel à `parallel_init(nbprocs,prank)` côté simulation
- client séquentiel (pour l’instant !)
- transparence du parallélisme pour le client
  → *ni modification du source, ni recompilation*

## Validation

- équation de la chaleur en parallèle (MPI)
- distribution = block 1D “à la main”

# Résultats Préliminaires



# Conclusion & Perspectives

## Conclusion

- un premier pas pour le pilotage des simulations sur la grille
- description XML + instrumentation
- infrastructure de communication basée sur CORBA
- plusieurs stratégies clientes
- performances encourageantes des prototypes

## Perspectives

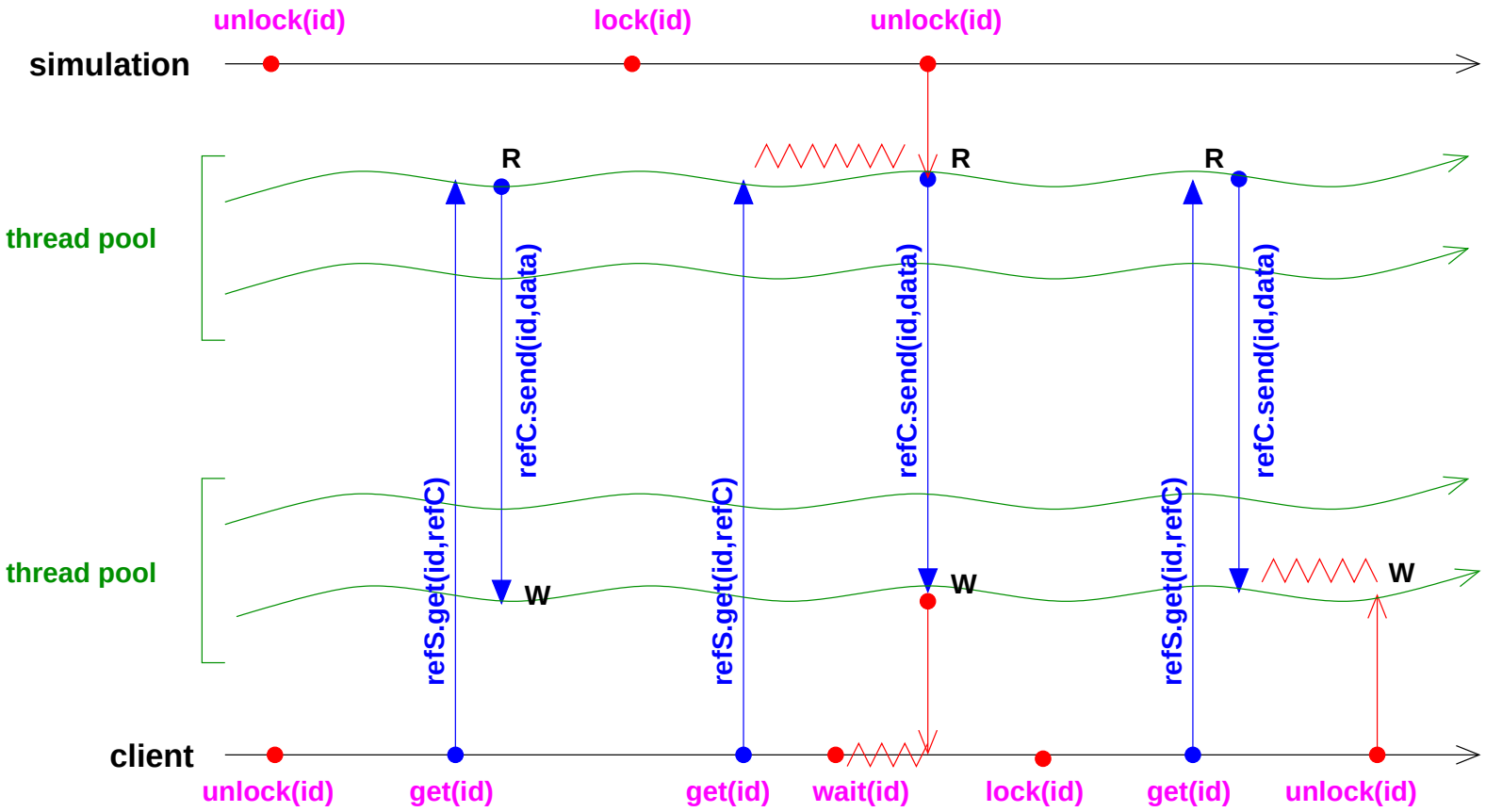
- interaction avec des objets complexes (e.g. maillages, molécules)
- simulations et clients parallèles (couplage M $\leftrightarrow$ N)
  $\rightarrow$ *redistribution de données régulières/irrégulières (RedGRID)*
- support des simulation parallèles distribuées (e.g ocean/atmosphere)

$\Rightarrow$ Epsilon 1.0 bientôt disponible !

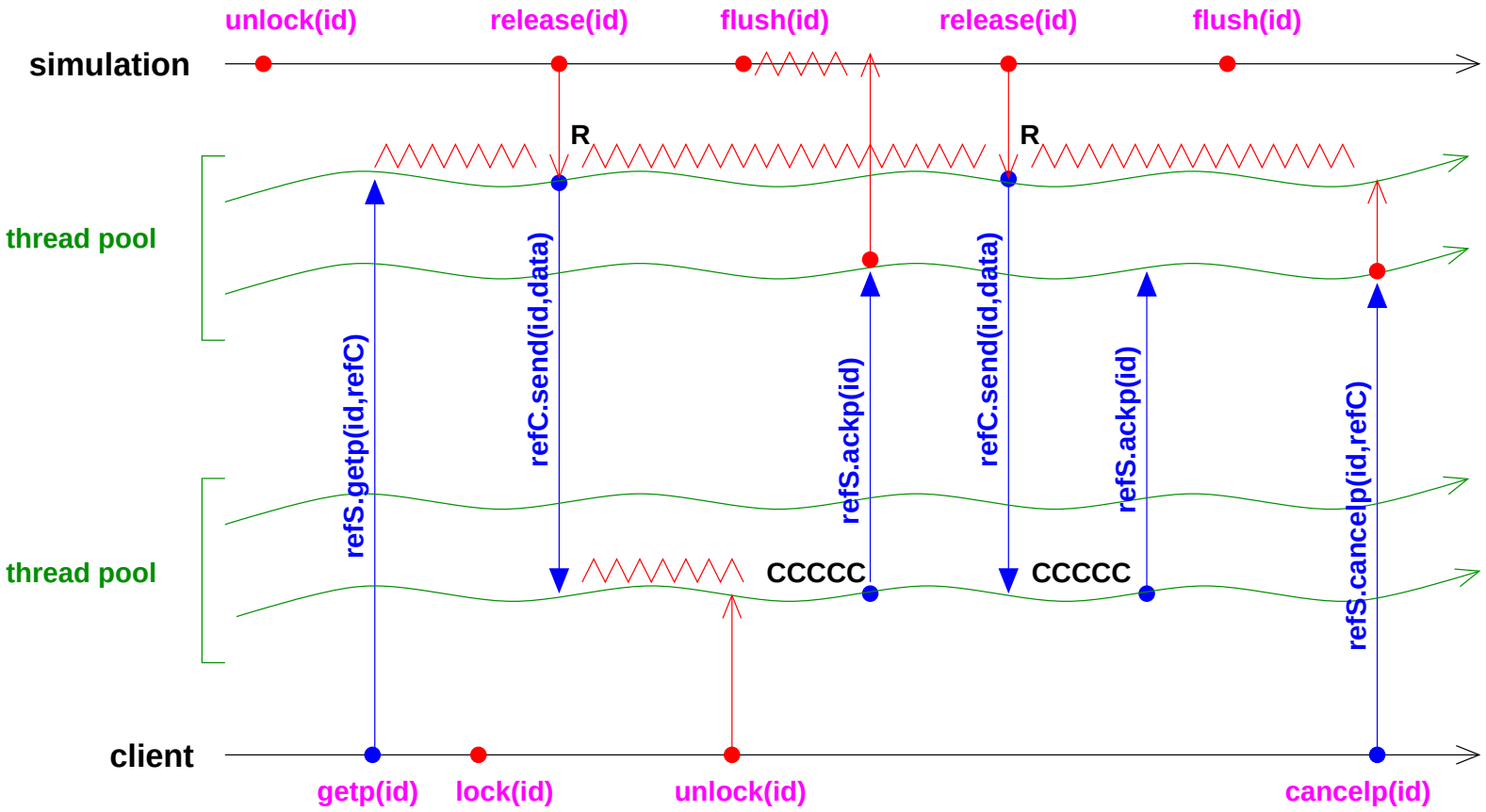
# FIN

# Annexes

# Communication & Synchronisation (get)



# Communication & Synchronisation (getp)



# Comparaison de performances