Toward a Distributed Computational Steering Environment based on CORBA

O. Coulaud, M. Dussere and A. Esnard

ParCO 2003 – Desden (Germany), september 2003.

EPSN project, french ACI-GRID initiative (grant number PPL02-03)

INRIA Futurs (ScAlApplix project) and LaBRI (UMR CNRS 5800)

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

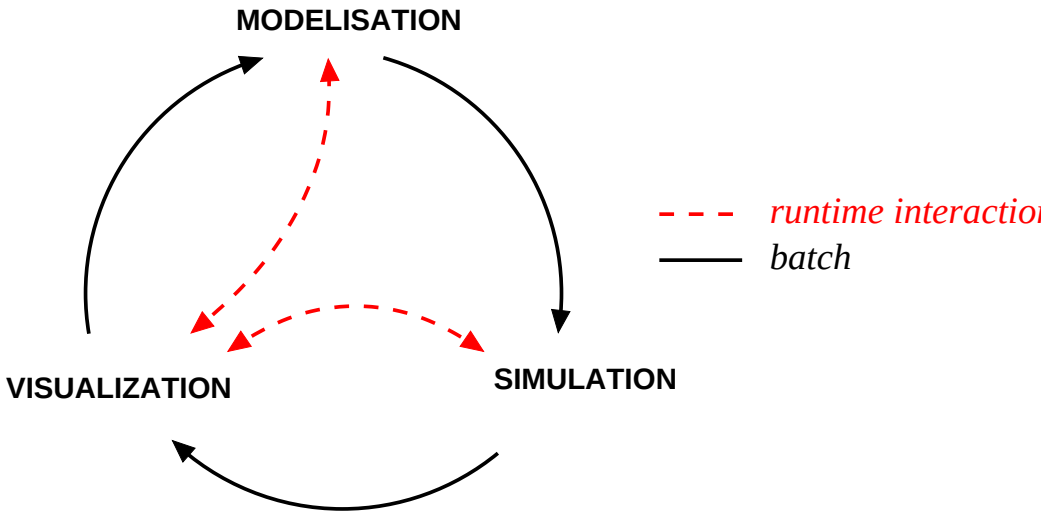
Outline

1. Introduction to Computational Steering
2. Overview of the EPSN Environment
3. Sequential Architecture and Parallel Extension
4. Steering System Design
5. Performance Evaluation
6. Conclusion and Prospects

Introduction to Computational Steering

Scientific Simulation

Scientific visualization plays a central role in the analysis of data generated by scientific simulations.



Batch

- a sequential work-flow
- visualization as a post-processing step

Computational steering

- “to close the loop”
- a more interactive approach

What is Computational Steering?

1. **J. Mulder, J. Wijk, R. Liere – 1999.**
   "Computational steering can be defined as the interactive control over a computational process."

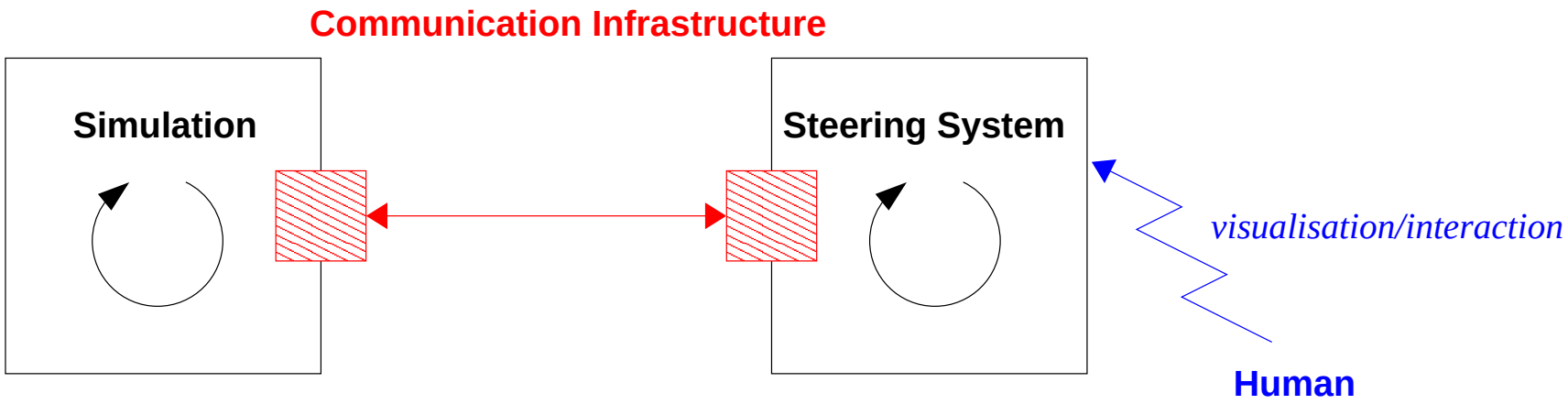
2. **J. Vetter, K. Schwan – 1996.**
   "Computational steering is the run-time control of an application and of the resources it uses for purposes of experimenting with application parameters or improving application performance."

3. **S. Parker, D. Beazley, C. Johnson – 1996.**
   "Computational steering (...) allows the efficient extraction of scientific information and permits changes to simulation parameters and data in a meaningful way."

Software Environnement

Three main components:

- a numerical simulation
- a steering system (remote user interface for monitoring & steering)
- a communication infrastructure



- Monitoring: observation of the program's behavior during execution
- Steering: modification of the program's behavior during execution

Steering Programming Model

How to integrate steering functionnalities in an existing simulation?

- Debugger approach
- Interaction through standard I/O (CaveStudy)
- Scripting languages (SWIG)
  $\rightarrow$ *light weight steering*
- Problem solving environment (SCIRUN, COVISE)
  $\rightarrow$ *construct new steering applications, visual programming*
- Program instrumentation (CUMULVS, Falcon, VIPER...)
  $\rightarrow$ *small modification in the source (subroutine calls)*

Communication Infrastructure

How to exchange data over heterogeneous distributed environment?

Low-level communication infrastructure:

- TCP/IP sockets (POSSE, COVISE, Visit)
- RPC (VIPER)
- XDR for heteregeneous data transfer (VIPER)

High-level communication infrastructure:

- *DataExchange* (Magellan, Progress, Falcon, MOSS)
  $\rightarrow$ *an event system which allows heterogeneous data transfer*
- Nexus (CAVEStudy)
  $\rightarrow$ *the GLOBUS communication layer*
- PVM (CUMULVS)

The EPSN Environment

Overview

Our goals:

- to develop a software environment for computational steering
  $\rightarrow$ *control, data exploration and modification*
- to support heterogeneous distributed system
- to achieve good performances (few perturbations on computation)

Our approach:

- a programming model based on source code instrumentation
- a communication infrastructure based on CORBA
- a generic coupling based on an abstract representation of the simulation

Overview

Features:

- a client/server approach
  $\rightarrow$ *simulation = server, steering system = client*
- a distributed and dynamic infrastructure
  $\rightarrow$ *multi-clients & multi-applications*
- a client request approach (different from the event system approach)

Target Simulations:

- existing C/C++/Fortran simulation code
- iterative processes
- sequential $\rightarrow$ parallel simulation (SPMD) $\rightarrow$ coupled simulation $\rightarrow$ distributed simulation (MPMD)

Instrumentation Principles

XML Description

- simulation scheme as a hierarchy of computational loops
- set of breakpoints $\rightarrow$ *id, state, location...*
- data description (scalar, sequence) and access permissions $\rightarrow$ *id, type, location, distribution, ...*
- group of logically correlated data

Source code annotation

- data publication, data memory address (`publish`)
- locate breakpoints at stable points (`barrier`)
- restricted access area (`lock/unlock`) $\rightarrow$ *data coherency*
- data release management (`release`)
- loop timestep evolution (`iterate`)

Steering Principles

Control

- `play/step/stop`

On-the-fly data access

- data extraction (`get/wait`)
- data modification (`put`)
- *callback* function at reception

“Systematic” data extraction

- automatically sending new data releases (simulation $\rightarrow$ client)
- registration request (`getp/cancelp`)
- acknowledgment system for frequency regulation
- “on-line” visualization

A Simple Example

XML Description

```
<simulation  name="spray"  context="parallel">
  <control  running="true">
    <breakpoint  id="begin"  state="down"  location="specific"/>
    <loop  id="loop"  state="up"/>
    <breakpoint  id="end"  location="distributed"/>
  </control>

  <data>
    <group  id="mesh">
      <scalar  id="nbNodes"  type="long"  access="readonly"  location="replicated"/>
      <scalar  id="nbCells"  type="long"  access="readonly"  location="replicated"/>
      <sequence  id="nodes"  type="double"  location="replicated">
        <dimension  size="nbNodes"/>
        <dimension  size="2"/>
      </sequence>
      <sequence  id="cells"  type="long"  location="replicated">
        <dimension  size="nbCells"/>
        <dimension  offset="1"  size="3"  alloc="5"/>
      </sequence>
      <sequence  id="energy"  type="double"  location="distributed">
        <dimension  size="nbNodes"  decomposition="block"/>
      </sequence>
    </group>
  </data>
</simulation>
```

Instrumentation (1/2)

```
// Mesh Initialization
struct Mesh { int nbNodes; int nbCells; int ** nodes;
              int ** cells;  double * energy; } * mesh;
InitMeshFromFile(mesh,"file");

// MPI Initialization
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&nbProcs);
MPI_Comm_rank(MPI_COMM_WORLD,&numProc);

// Epsilon Initialization
epsilon_init("spray.xml",numProc,nbProcs);
epsilon_publish("nbNodes",&mesh->nbNodes);
epsilon_publish("nbCells",&mesh->nbCells);
epsilon_publish("nodes",mesh->nodes);
epsilon_publish("cells",mesh->cells);
epsilon_publish("energy",mesh->energy);
epsilon_publishgroup("mesh");
epsilon_unlockall();

// Barrier on master process
if(numproc == 0) { epsilon_barrier("begin");}
```

Instrumentation (2/2)

```
MPI_Barrier(MPI_COMM_WORLD);

// Computation Loop
for(int kt = 0; kt < ktmax; kt++){
   epsilon_flush("energy");

   // fluid injection
   inject(mesh,energy_buffer);
   epsilon_lock("energy");
   copy(mesh->energy,energy_buffer);
   epsilon_unlock("energy");
   epsilon_release("energy");
   // Post Processing & MPI communication
   postprocessing(mesh);
   epsilon_iterate("loop");
}

epsilon_barrier("end");
// Simulation Ending
WriteResult(mesh);
// Epsilon Finalization
epsilon_exit();
```

Architecture

The Choice of CORBA

CORBA features:

- advanced programming model
- design for distributed communication
- CDR for heterogeneous data transfer
- portability, network transparency, interoperability

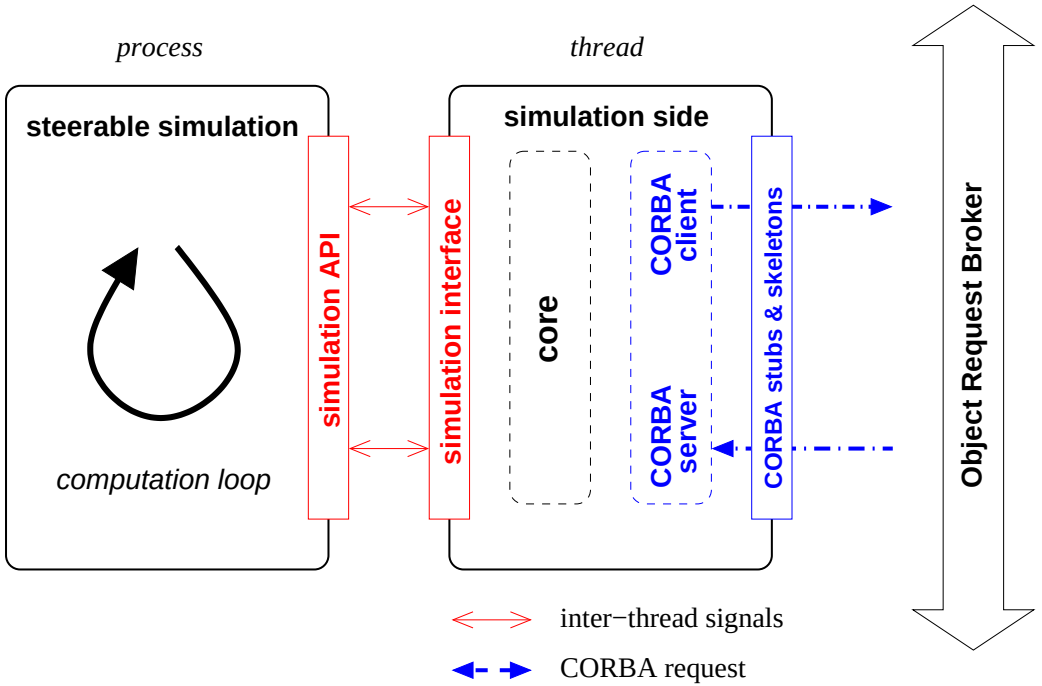
Requirements for HPC:

- efficiency on high-speed networks $\rightarrow$ OmniORB, TAO, PADICO
- parallelism support $\rightarrow$ OMG, PARDIS, PACO++

Requirements for EPSN:

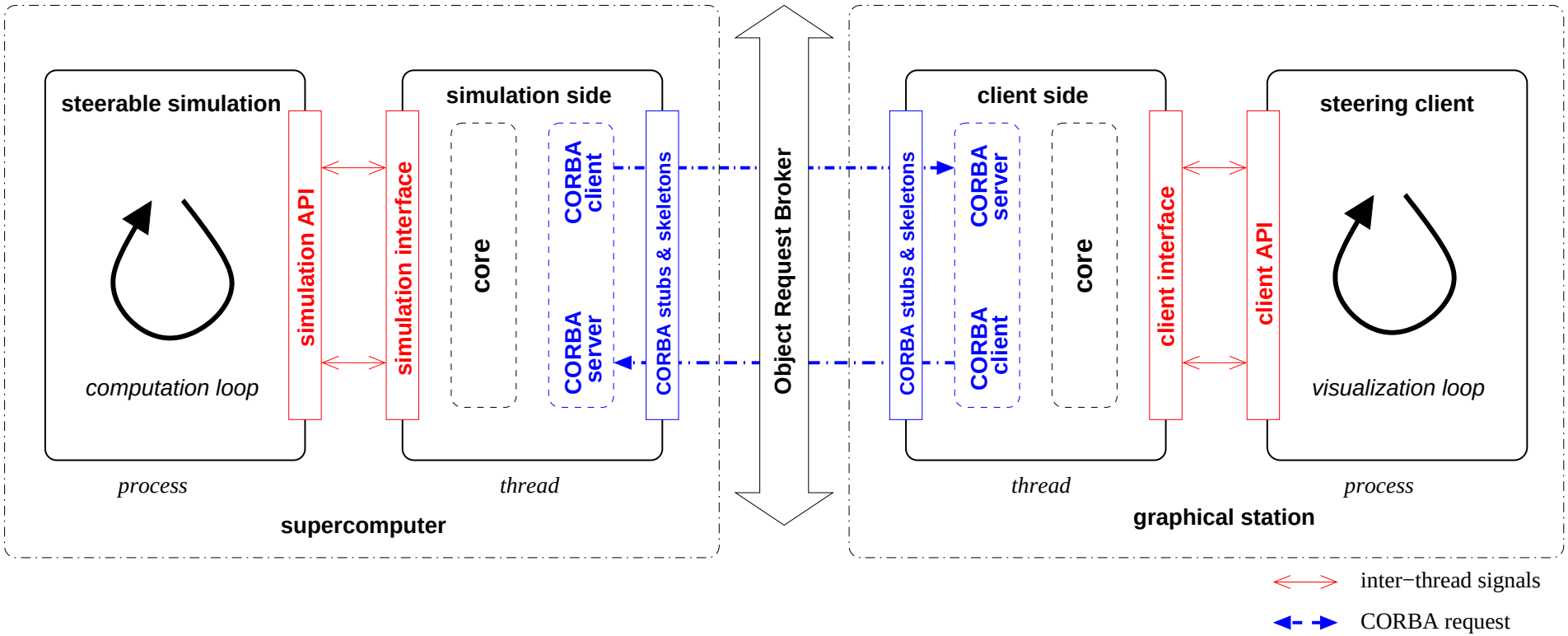
- hiding the inherent complexity of CORBA to the end-user
- imperative languages (Fortran, C)

Communication Infratructure



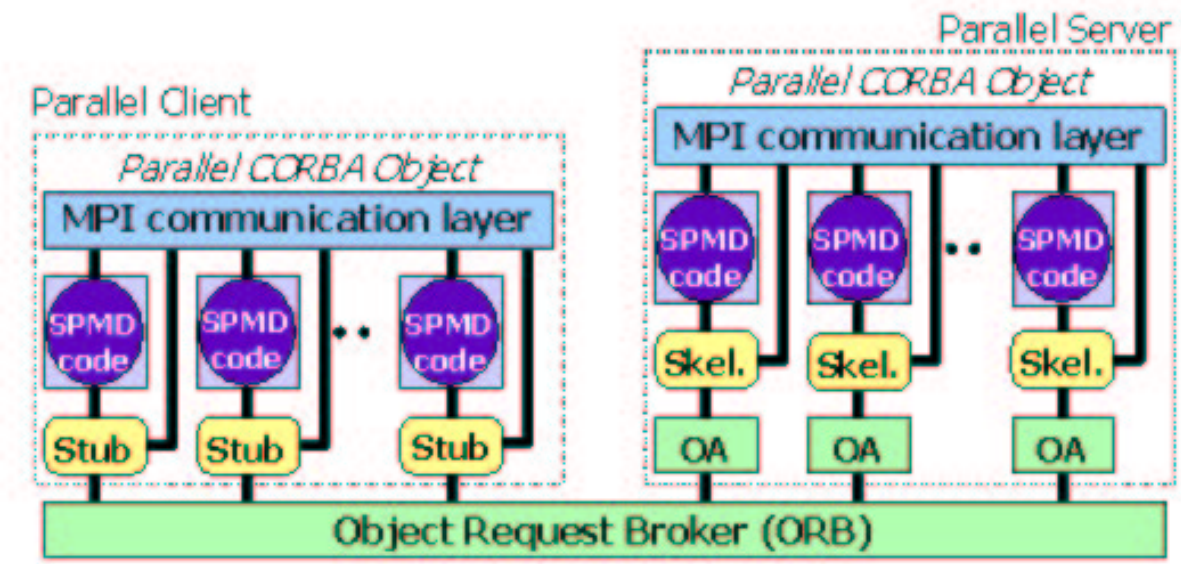
- a CORBA server started in a dedicated thread
- a CORBA thread pool for the steering treatments
- thread synchronization based on semaphores and signals
- data access through shared memory (zero copy)
- asynchronous communication (*oneway* CORBA request)
- communication overlapping

The Whole Infratructure



EPSN Parallel Extension

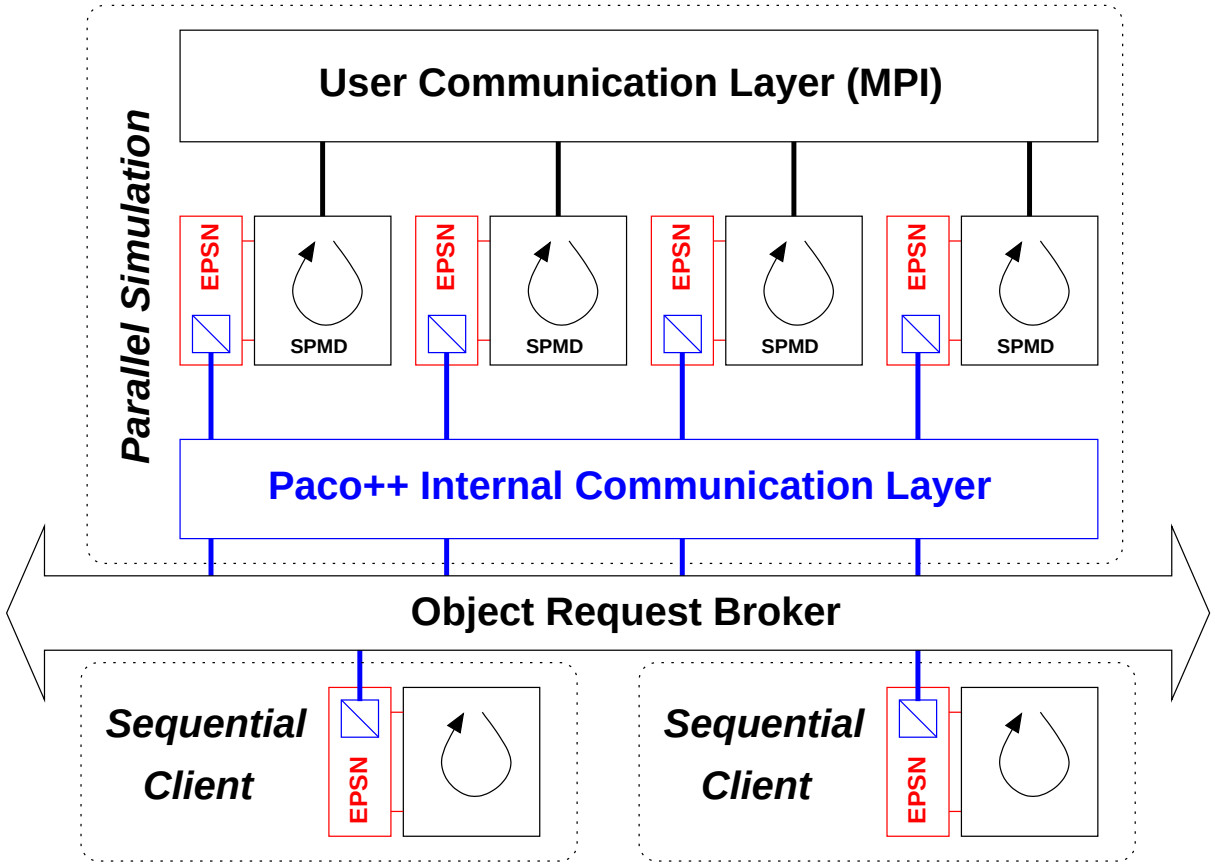
Parallel CORBA Objects with PaCO++



Paris Project (IRISA)

- collection of identical CORBA objects
- no modification of the CORBA specifications $\rightarrow$ portable
- communication thanks to an external mechanism (MPI)
- data distribution transparency (XML description)

Parallel Architecture Overview

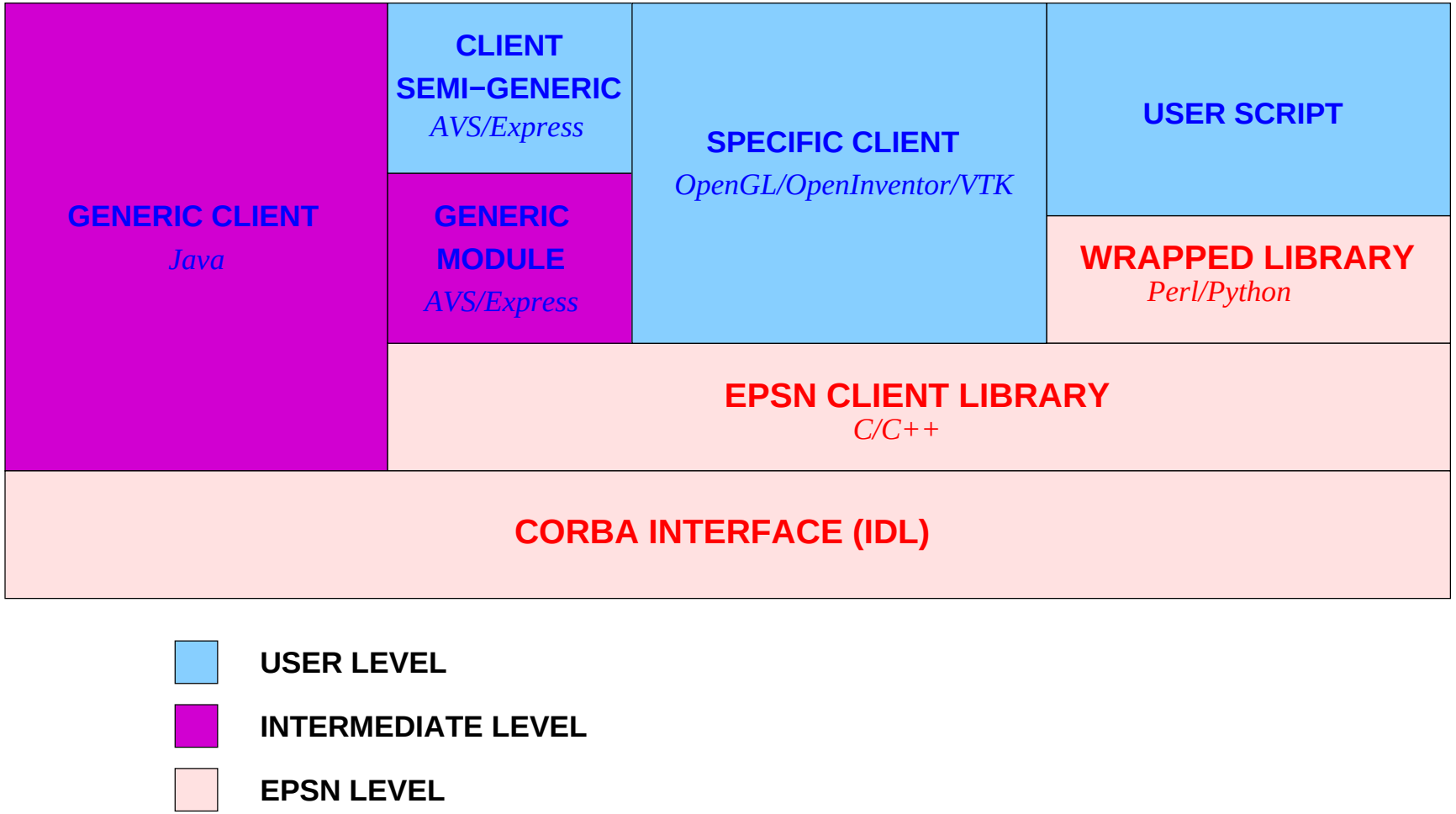


- instrumentation of each simulation process
- synchronization mechanisms

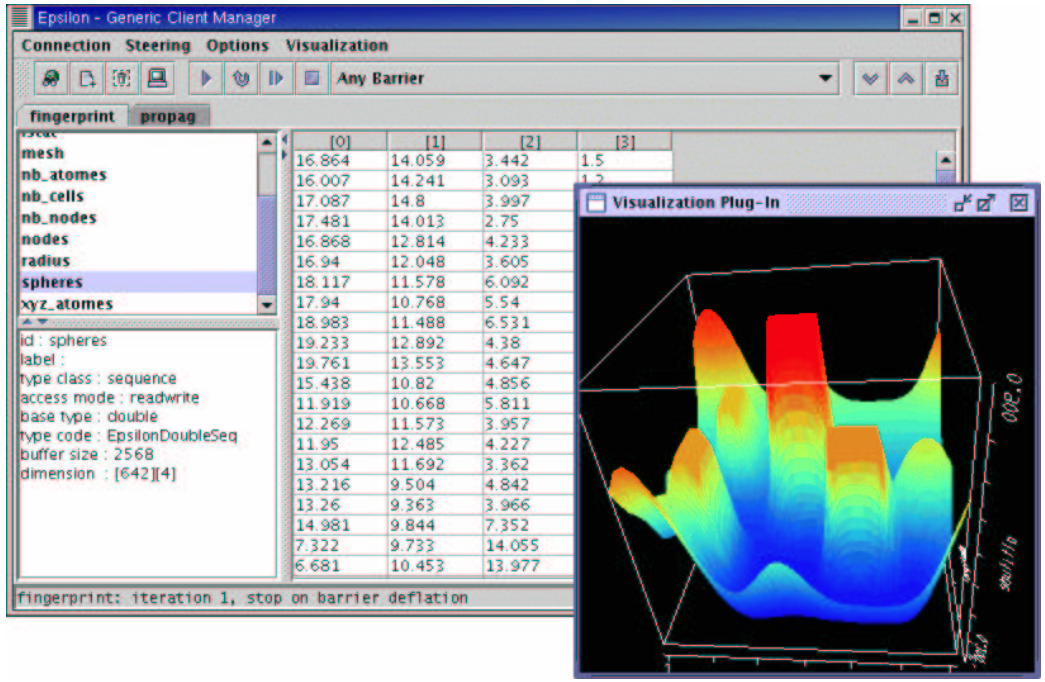
EPSN Steering System

Steering System Design

Multiple strategies to interact with your simulation.

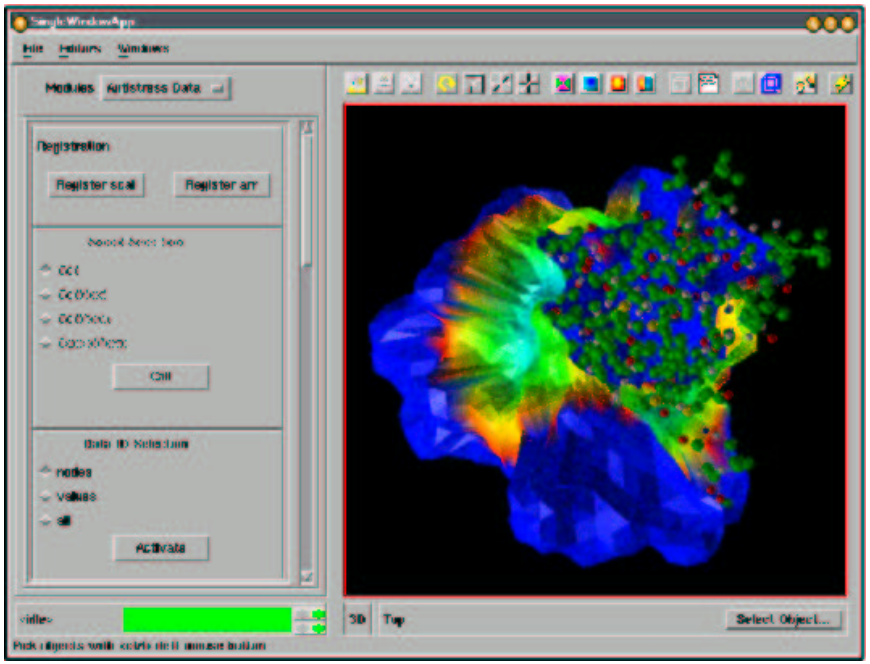


EPSN Generic Client



- implemented in Java/Swing
- control and data access of any EPSN simulation
- presenting data through simple numerical data-sheets and basic visualization plug-ins

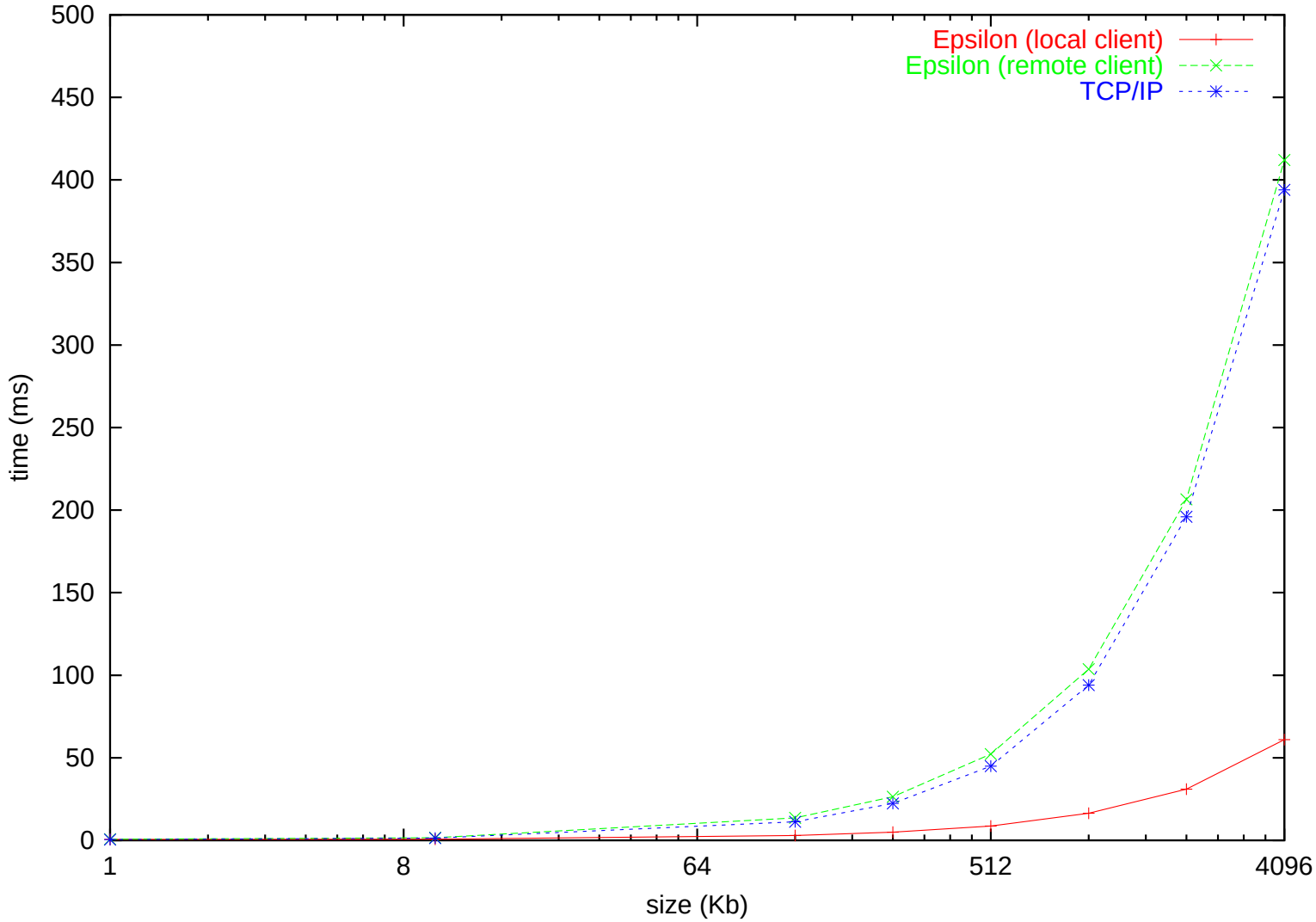
AVS/Express Semi-Generic Client



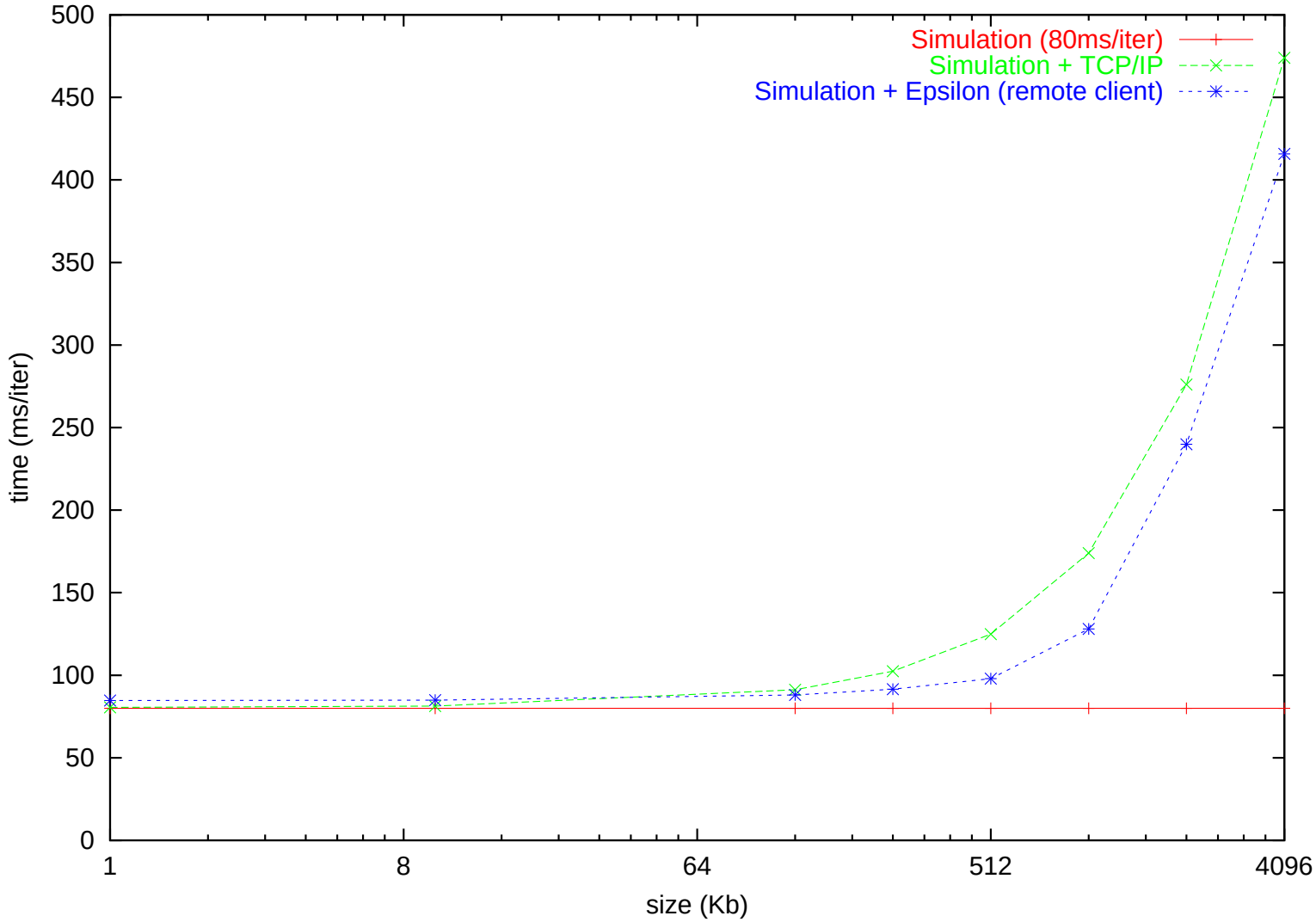
- visual programming model (dataflow oriented)
- EPSN generic module (data extraction and configuration)
- visualization of complex objects (meshes, molecules, etc.)

Performance Evaluation of the Sequential Prototype

Communication



Data Extraction from a Simulation



Conclusion & Prospects

Conclusion:

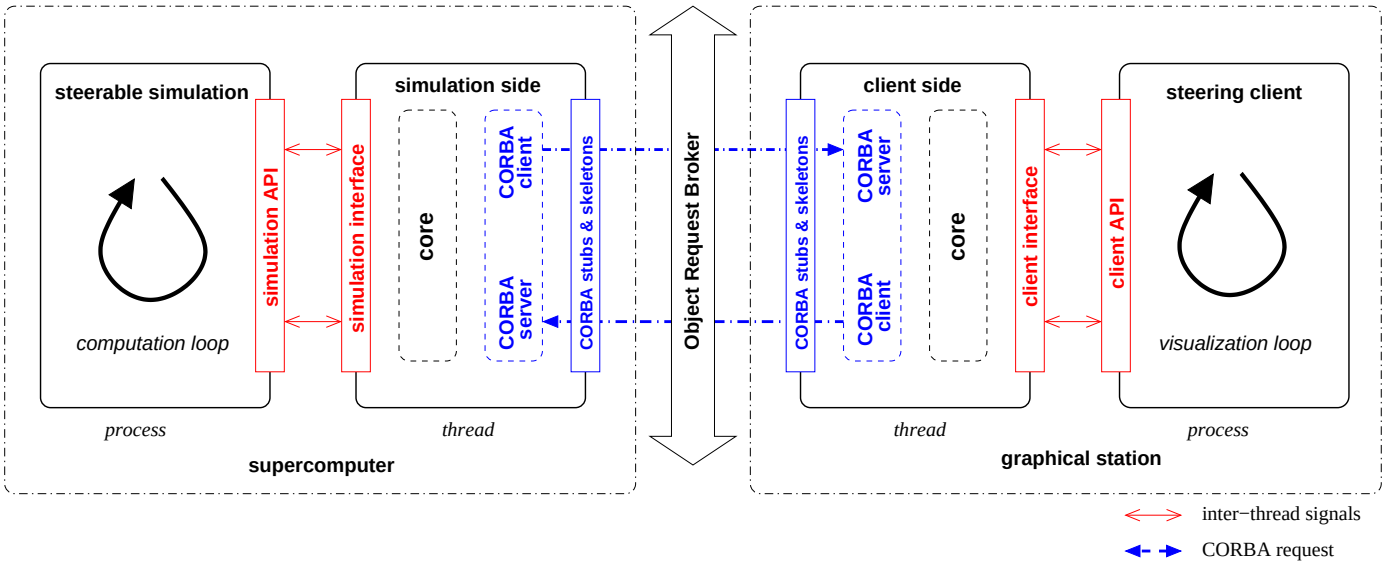
- a first step toward a computational steering environment
- XML description + source code annotations
- communication infrastructure based on CORBA
- first developement for SPMD simulation with regular data distribution
- several strategies to implement steering clients
- encouraging performances with the sequential prototype

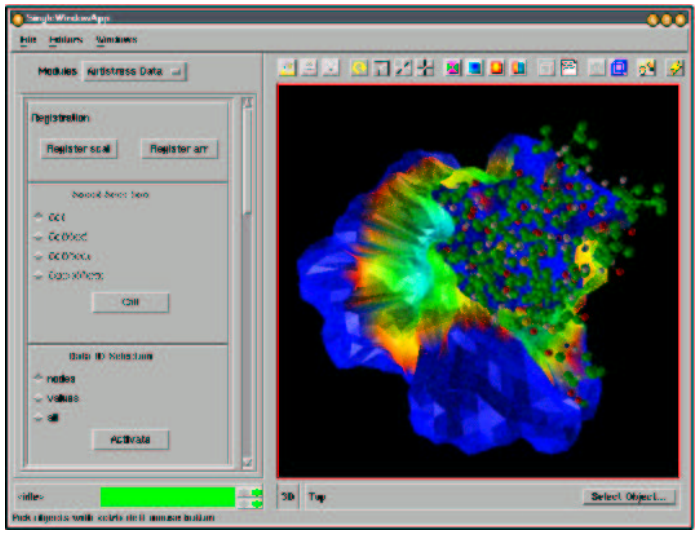
Prospects:

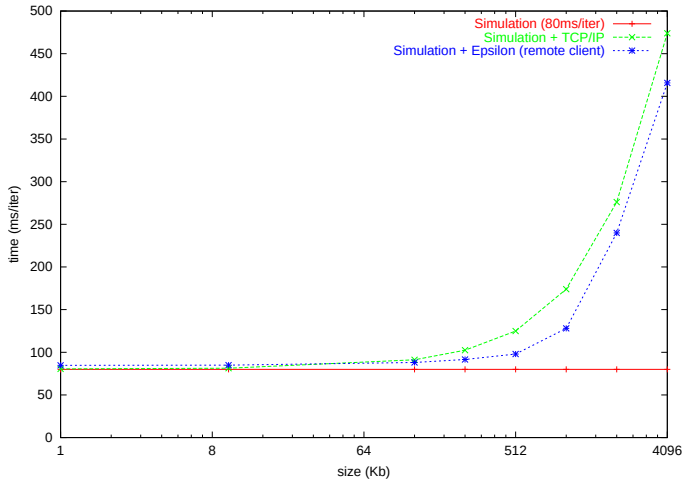
- PaCO++ full integration
- irregular data distribution on SPMD
- extension to distributed application (MPMD)

THE END.

Questions?

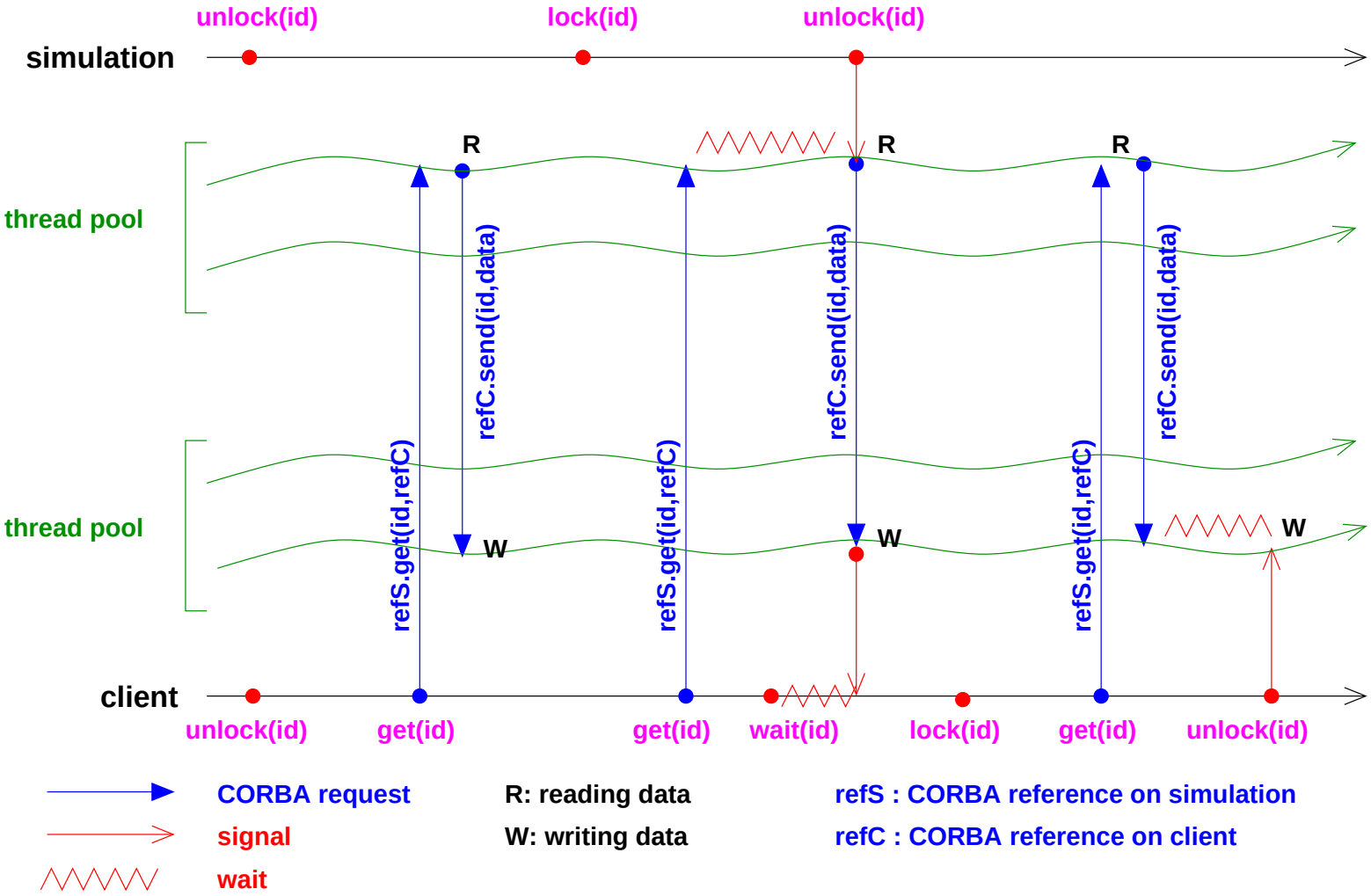

steerable simulation
simulation side
client side
steering client
simulation API
simulation interface
core
CORBA client
CORBA server
CORBA stubs & skeletons
Object Request Broker
CORBA stubs & skeletons
CORBA server
CORBA client
core
client interface
client API
computation loop
visualization loop
process
thread
thread
process
supercomputer
graphical station
inter−thread signals
CORBA request



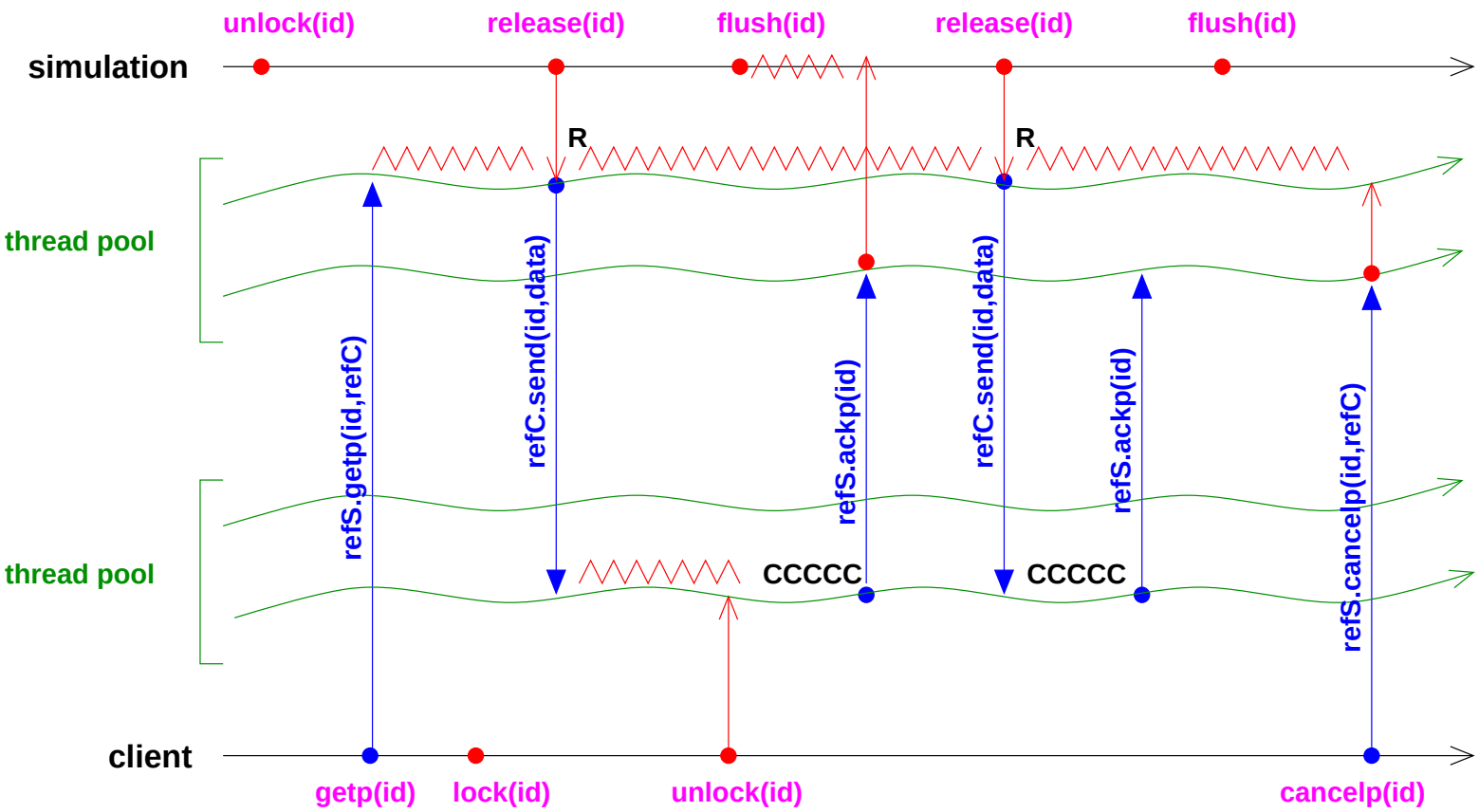

Simulation (80ms/iter)
Simulation + TCP/IP
Simulation + Epsilon (remote client)
time (ms/iter)
size (Kb)
0
50
100
150
200
250
300
350
400
450
500
1
8
64
512
4096

Appendix

Get Request


unlock(id)
lock(id)
unlock(id)
simulation
thread pool
thread pool
client
R
R
R
W
W
W
refS.get(id,refC)
refC.send(id,data)
refS.get(id,refC)
refC.send(id,data)
refS.get(id,refC)
refC.send(id,data)
unlock(id)
get(id)
get(id)
wait(id)
lock(id)
get(id)
unlock(id)
CORBA request
signal
wait
R: reading data
W: writing data
refS : CORBA reference on simulation
refC : CORBA reference on client

Getp Request



Performance Comparison