Traçage événementiel pour l'analyse d'applications réparties

francoisgael.ottogalli@rd.francetelecom.fr

France-Télécom R&D DLT/ASR

Jean-Marc.Vincent@imag.fr

Projet Mescal

Guest stars : Jacques Chassin de Kergommeaux (LSR-IMAG)
Benhur Stein (Univ. Santa Maria)
Eric Maillet (Astra)

Expérimentation et mesure

Je dis souvent que lorsqu'on peut mesurer ce dont on parle et l'exprimer en chiffres, on en sait quelque chose; en revanche, si on ne peut l'exprimer en chiffres, on en a une bien piètre connaissance.

Lord William Thomson Kelvin (1883)

On pourrait déterminer les différents âges d'une science par la technique de ses instruments de mesure.

Gaston Bachelard

Organisation

- Motivations
- Modèles et mesure
- Mise en œuvre : application à objets répartis
- Exemple d'analyse : serveur de données
  - client / courtier / serveur
- Perspectives

Motivations

Contexte des applications parallèles/distribuées

Spécifications qualitatives :

→Le résultat est correct (preuve, tests, …)

Spécifications quantitatives :

→ Le résultat est-il obtenu dans de bonnes conditions

Identification du problème

→ Déboguage

→Analyse de performances

Modification du code source / librairies / OS / architecture …

Objectif scientifique

Comprendre le **comportement** dynamique d'une application **distribuée**

- identification de comportements (patterns répartis)
- vérification (aide au débogage)
- évaluation des durées
- analyse globale de l'exécution
- évaluation des performances
- politique de contrôle (monitoring)
- évaluation du coût

Analyser l'**utilisation** des **ressources**

- applicatif
- support exécutif
- système d'exploitation
- architecture matérielle

Approches

précision, complexité, coût,…

généralisation

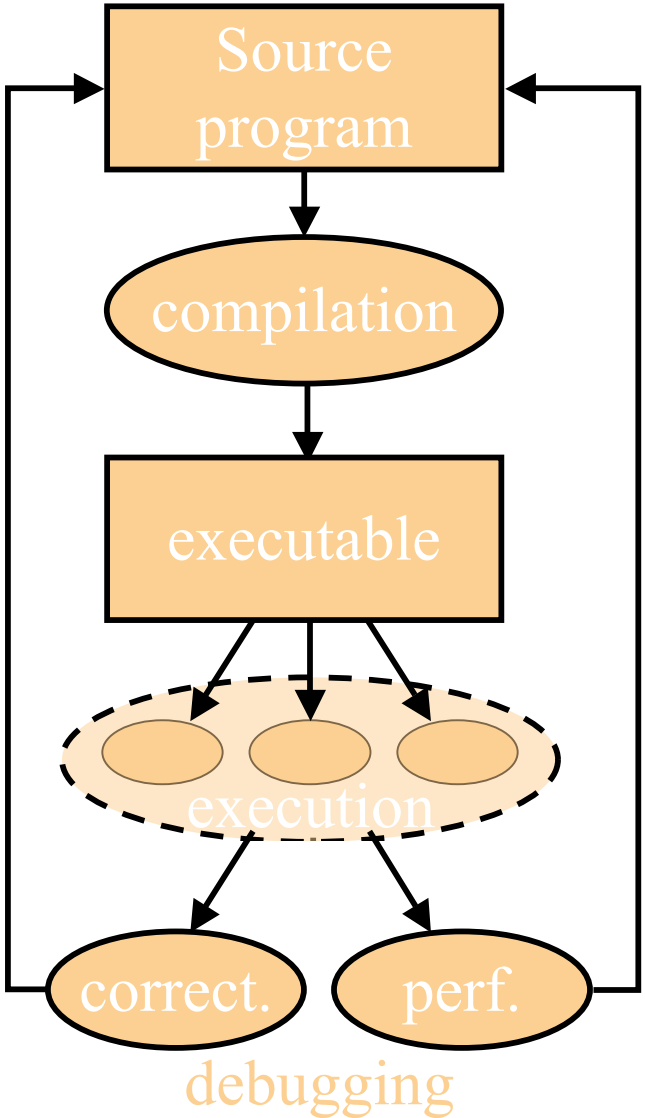
Modèle		Méthode
mathématique	→	analyse (formelle, numérique)
logiciel	→	simulation (évts discrets,…)
mesure	→	observation

Remarque : méthodes hybrides (émulation, générateurs de trafic,…)

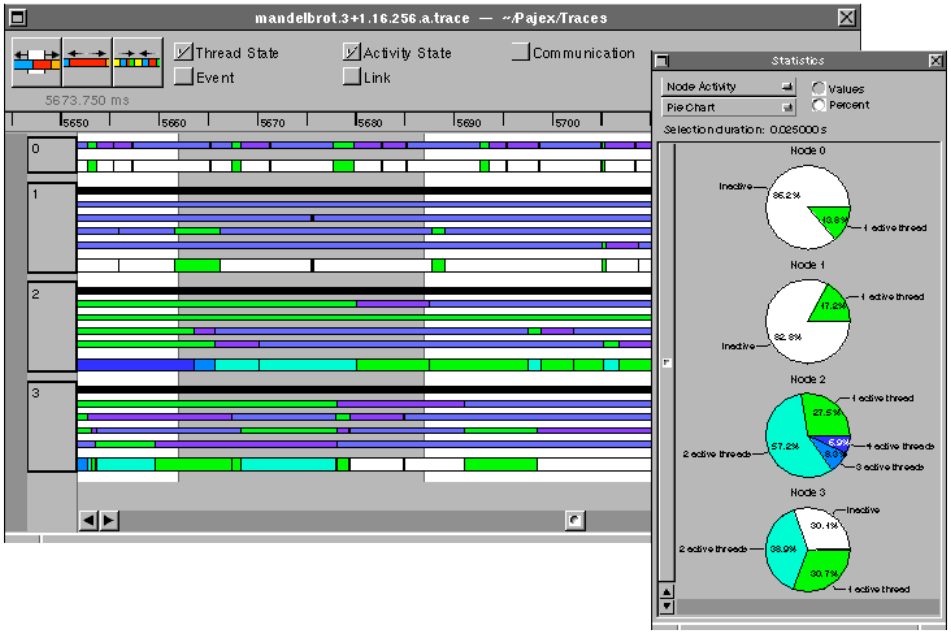
Expérimentation → Plan d'expérience

choix des paramètres, facteurs,…

Distributed program development cycle



Visualization



Context

(Potentially large size) parallel/distributed applications.

Executing on (potentially large size) distributed systems:

- Clusters of PCs.
- Grids of processors and clusters.

Keypoints

Distributed **heterogeneous** resources

Dynamicity of the architecture

Scalability of the middleware

Objective

Help users find performance errors:

- Lack of parallelism, bottlenecks, overheads.

Behavior analysis methodology

- **Execution** model
- **Measurement** environment
- **Visualization** model

Performance debugging tools

Monitoring to gather performance data

Data analysis:

- Collection of raw data
- Reconstruction (simulation) of the behavior of the application and computation of performance indexes.
- Presentation (visualization).

Existing visualization tools

Paragraph:

Large number of visualizations.

Not easily extensible/ Not interactive.

Pablo:

Easily extensible.

No behavioral visualization / Not extensible.

Paradyn:

Scalable. Automatic research of errors.

No behavioral visualization.

ParaGraph

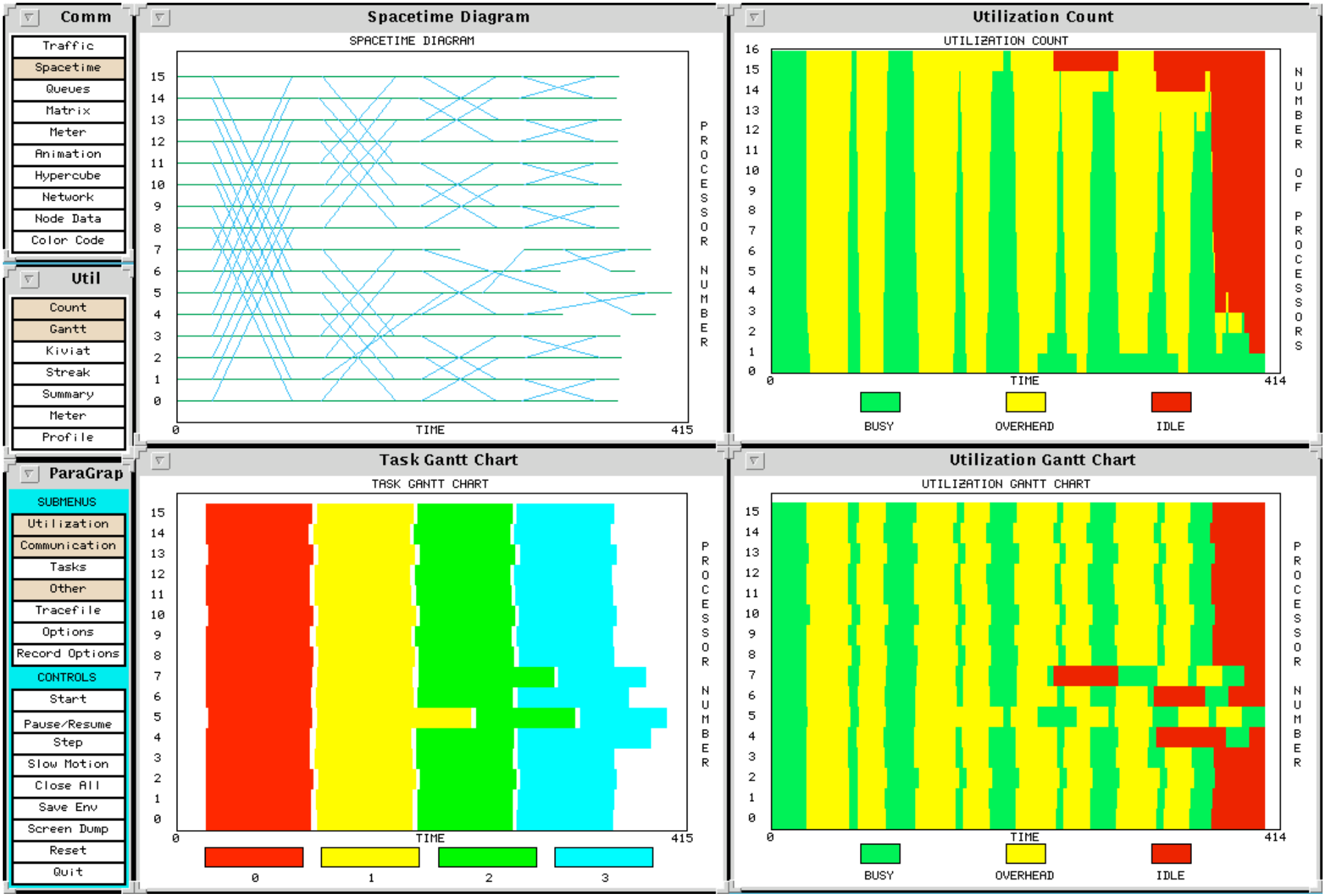
The first widespread public domain visualization tool.

Execution traces recorded by the parallel programming library PICL.

Large number of visualizations:

- Processor utilization.
- Communications: fréquency, volume, distribution.
- Various: critical path, statistics, etc.

Some Visualizations of ParaGraph



Limitations of ParaGraph

Most visualizations are "instantaneous".

Lack of scalability:

- Visualizations become cryptic for a couple of dozens of processors.
- No filtering: impossible to visualize a subset of the system.

Limited extensibility: monolithic implementation.

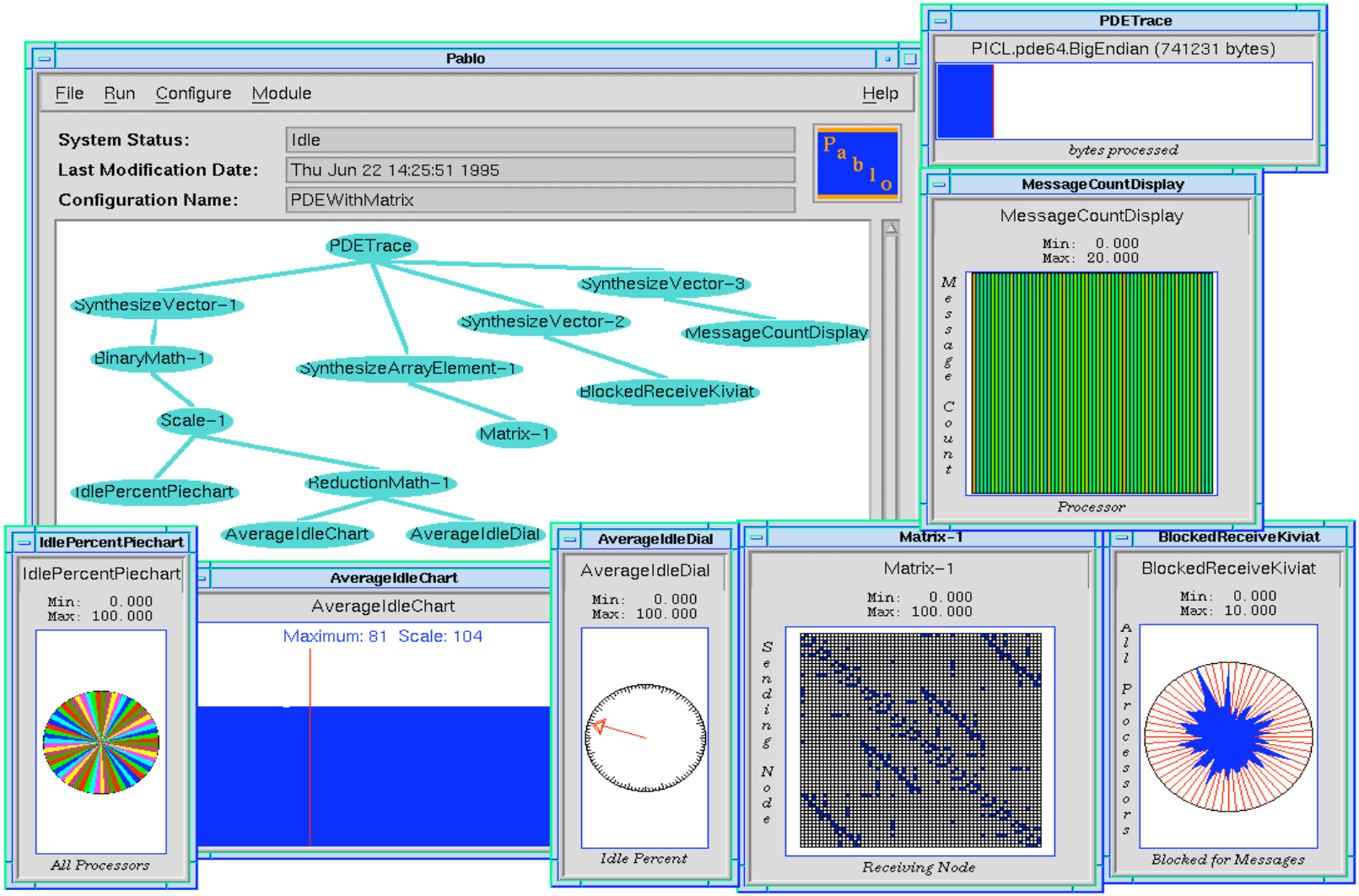
PABLO

Architecture : graph of components connected by the user to build some visualizations.

Large number of predefined components:

- Transformations : arithmetic opérations, statistics, etc.
- Visualizations: pie chart, Kiviat, clouds of points, etc.

Pablo visualization



Assessment of Pablo

Extensibility, flexibility:

- Relatively simple to add a new component.
- Indépendence of programming model.

Scalable visualizations.

Non interactive, no relation between source code and visualizations.

No behavioral visualization of program executions.

Difficult to use? (meta tool).

SvPablo

One single component: monitoring, data analysis, etc.

Data monitoring: counting,timing.

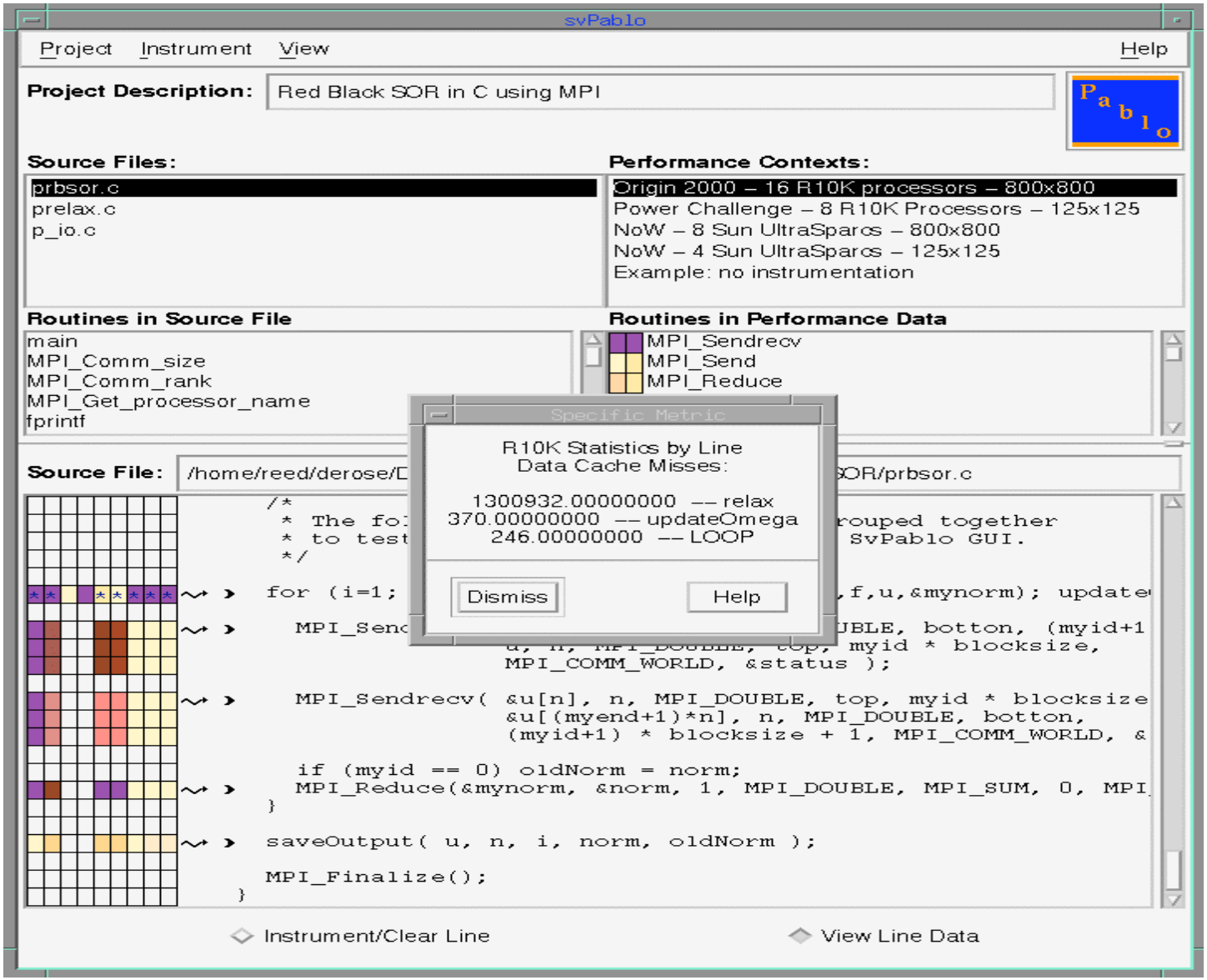
Independence of the implementation language of the monitored application.

Correlation between input and application source program.

No trace ⇒ no behavioral visualization.

Scalability: counting + statistical visualizations.

SvPablo Visualization



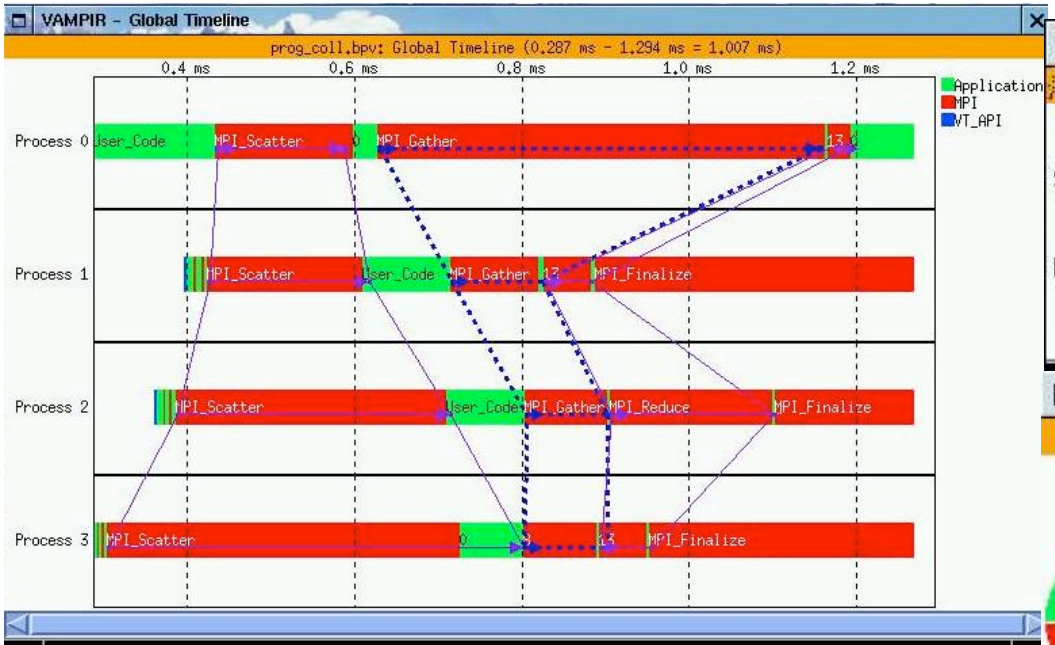
Vampirtrace and Vampir

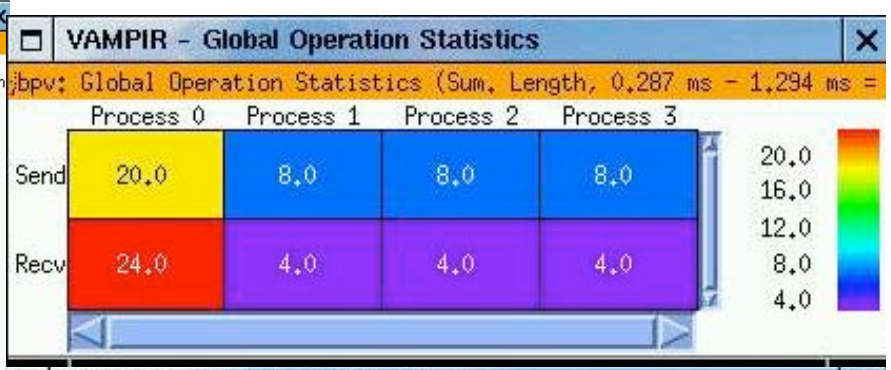
Toolkit for performance visualization of MPI programs.

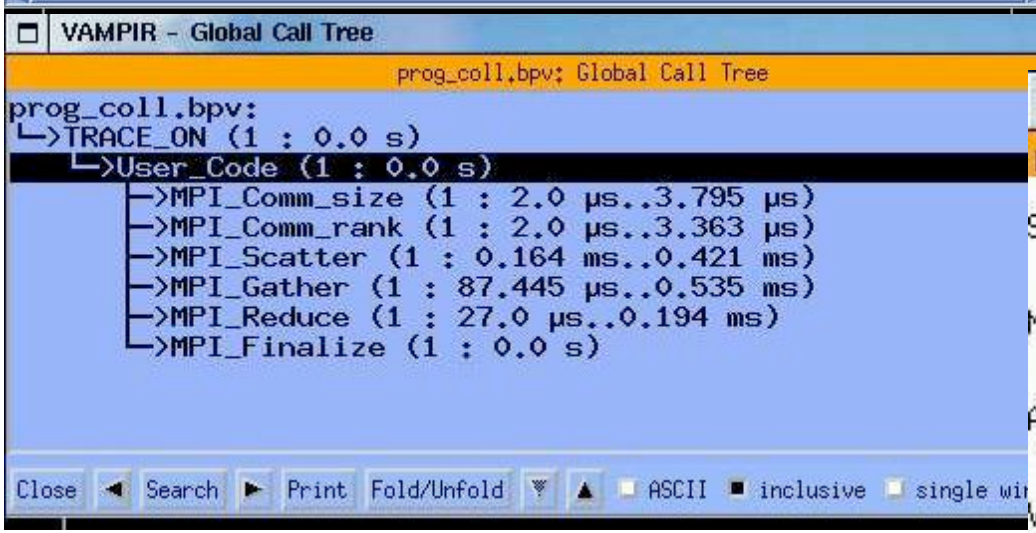
Scalability:

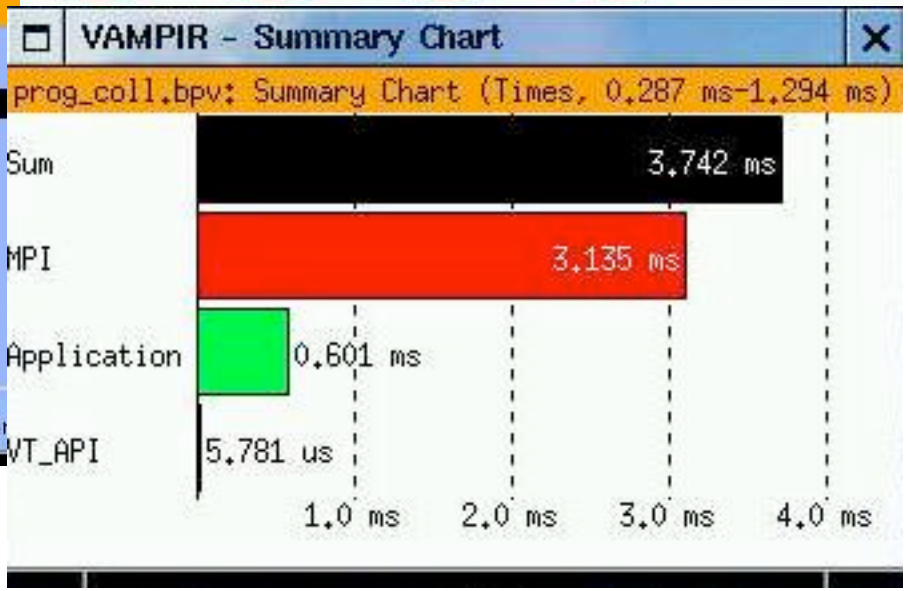
- Numerous zooming possibilities.
- Filtering of visualized objects.
- Filtering of traces during monitoring (when using Vampirtrace).
- Possibility to load a subset of a trace file: the set of events of a time period.

Vampir Visualizations









Vampir

Interactivity:

- Inspection of visual objects.
- Relation with source code (click back).
- Zooming along time axis.
- Not possible to move back and forth in time.

Extensibility? (commercial tool).

Standard?

- Translators to the Vampir format (ex. Tau → Vampir).
- Widespread commercial tool.

Pajé*: interactive and extensible and scalable visualization tool

*doctor in Tupi language

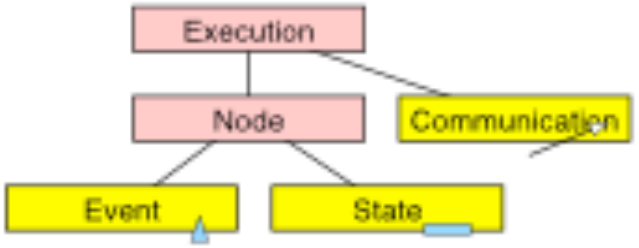
Originally: visualizations of **communicating thread** program executions.

Main issue: **large number** of visual objects:

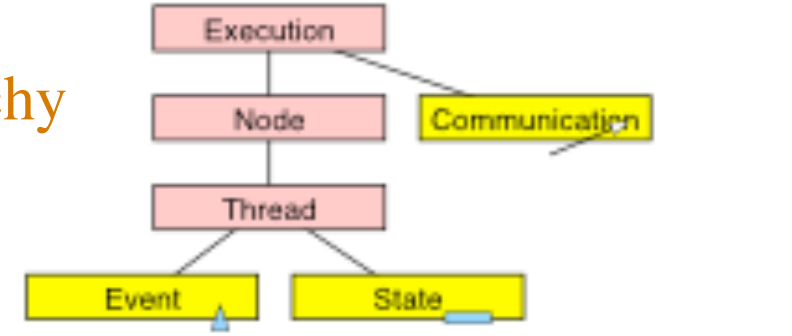
- Large number of nodes.
- One each node, dynamically created and terminated threads.
- **Synchronizations and communications**.

Later: variety of programming models.

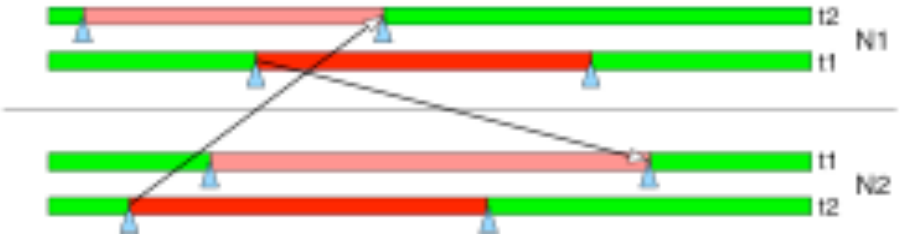
Example of type hierarchy definition and instance



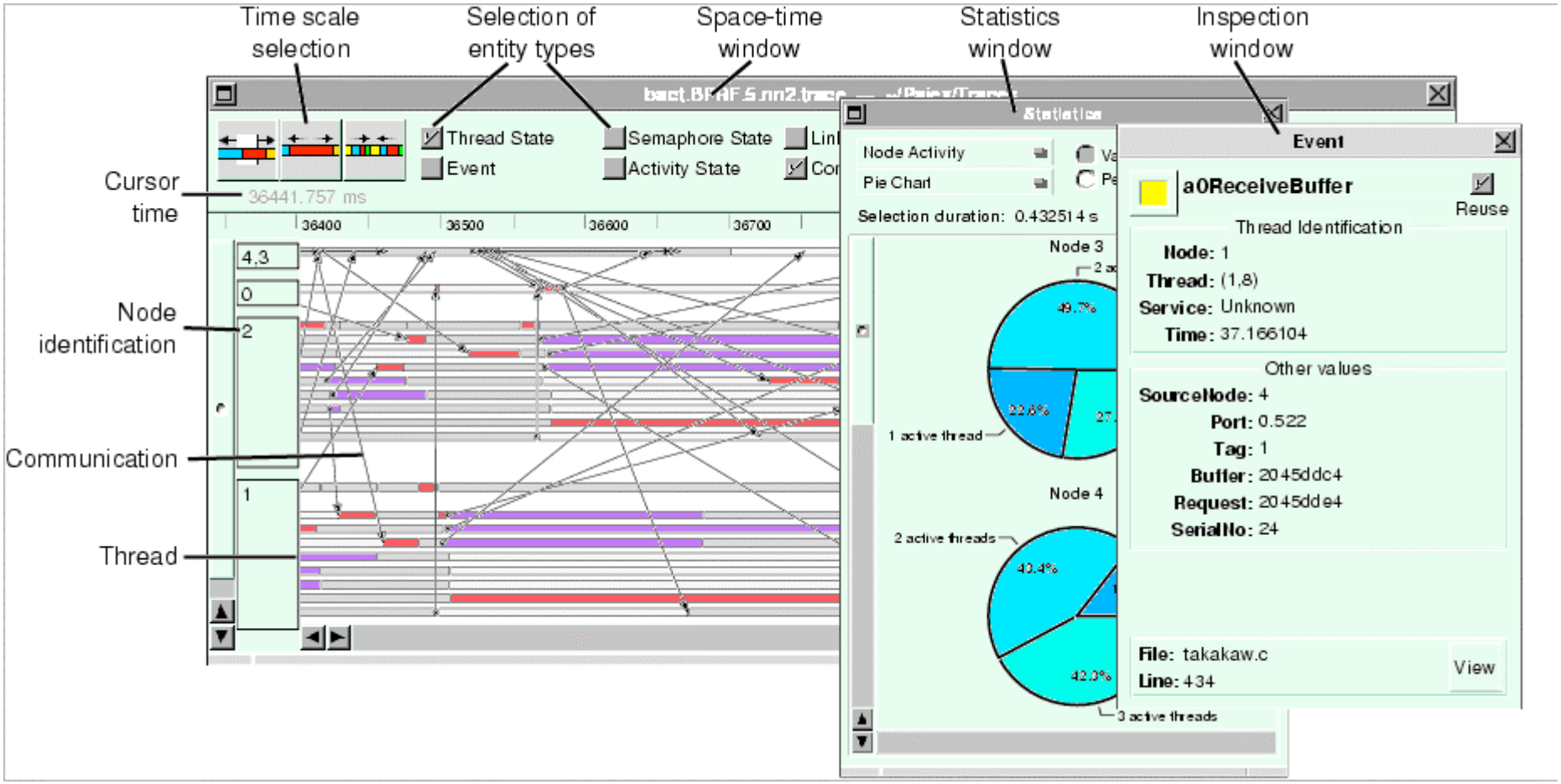
Type hierarchy definition



Instance of type hierarchy

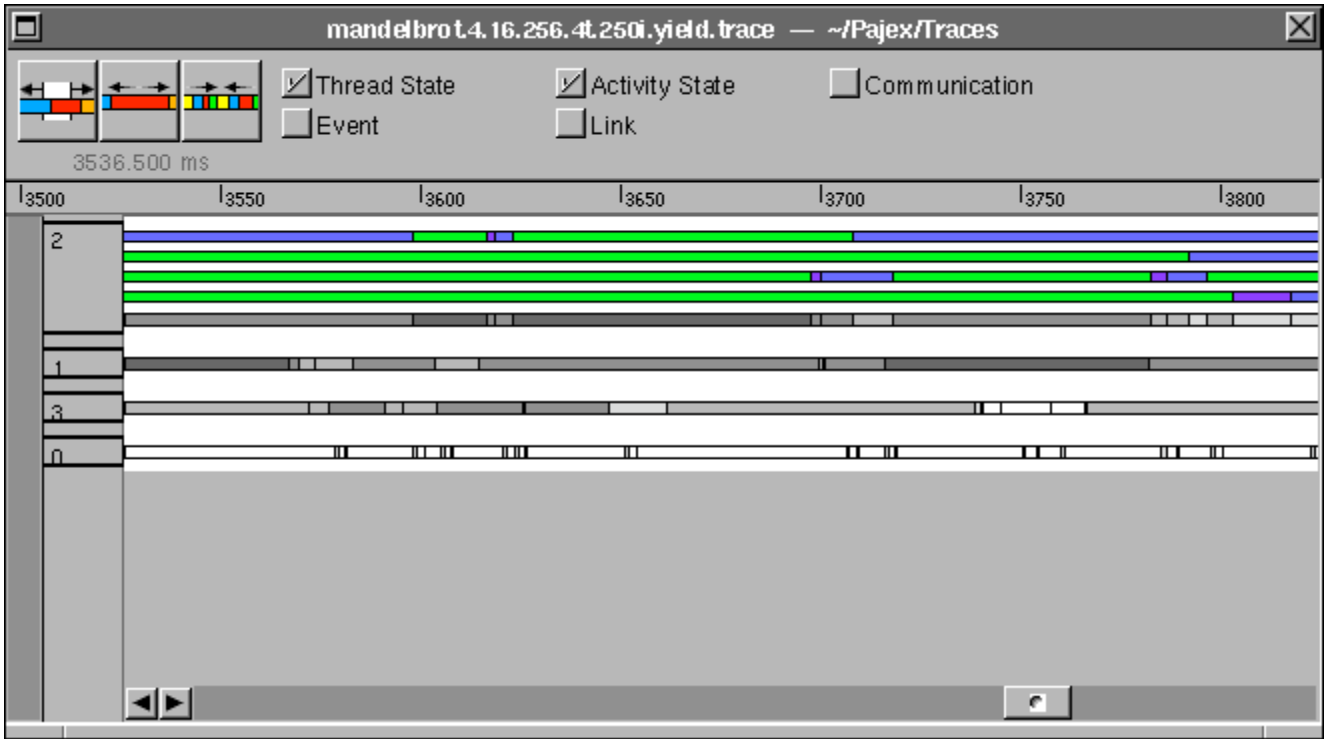


Pajé visualization example



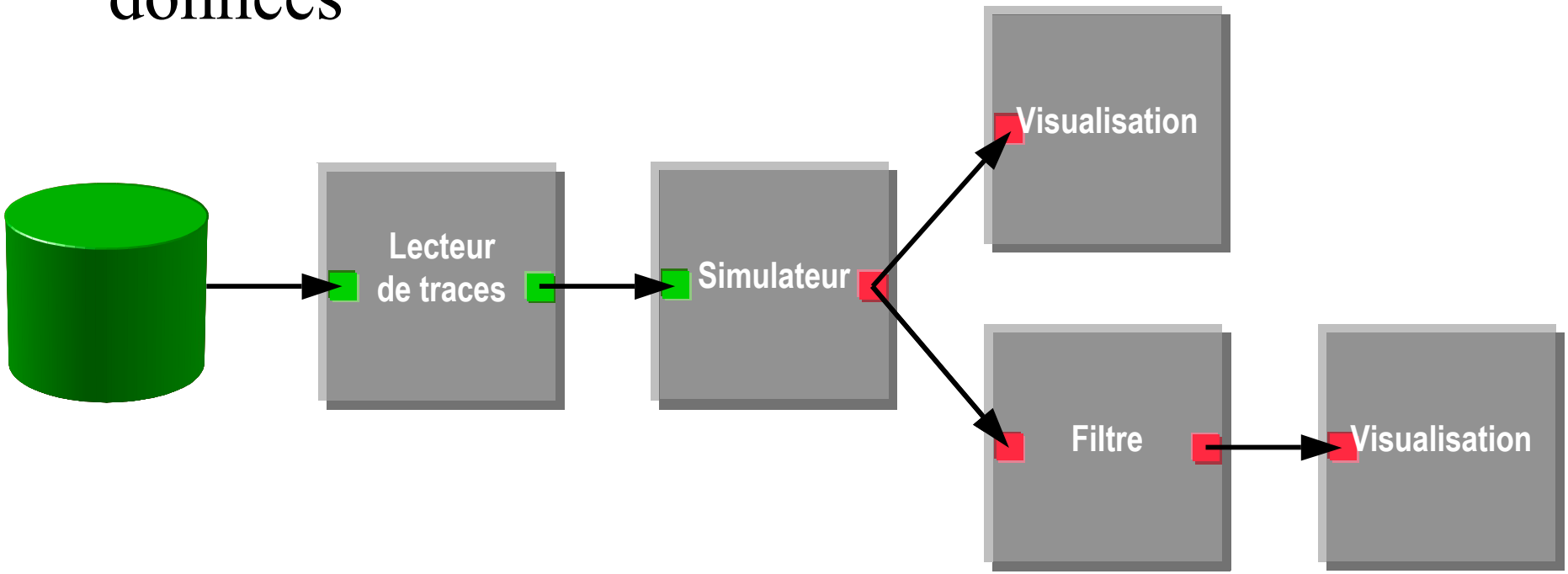
« Scalabilité »

- Réduction des informations d'un nœud



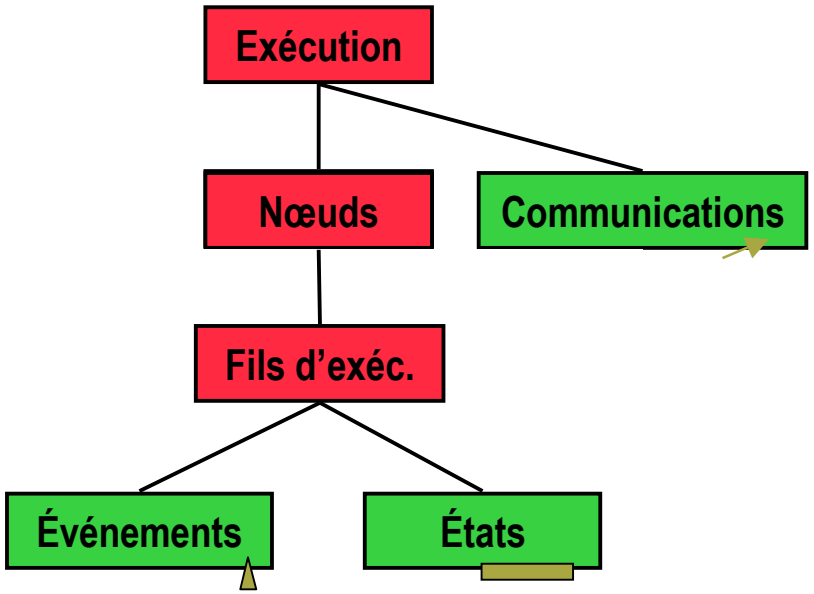
Extensibilité

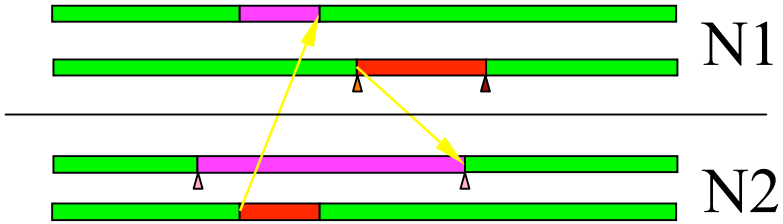
- Architecture de l'outil : graphe flot de données



Extensibilité

- Pajé : ·description de la structure des objets à visualiser
  - ·simulateur générique
    - description de la hiérarchie des types d'objets visualisés dans la trace.





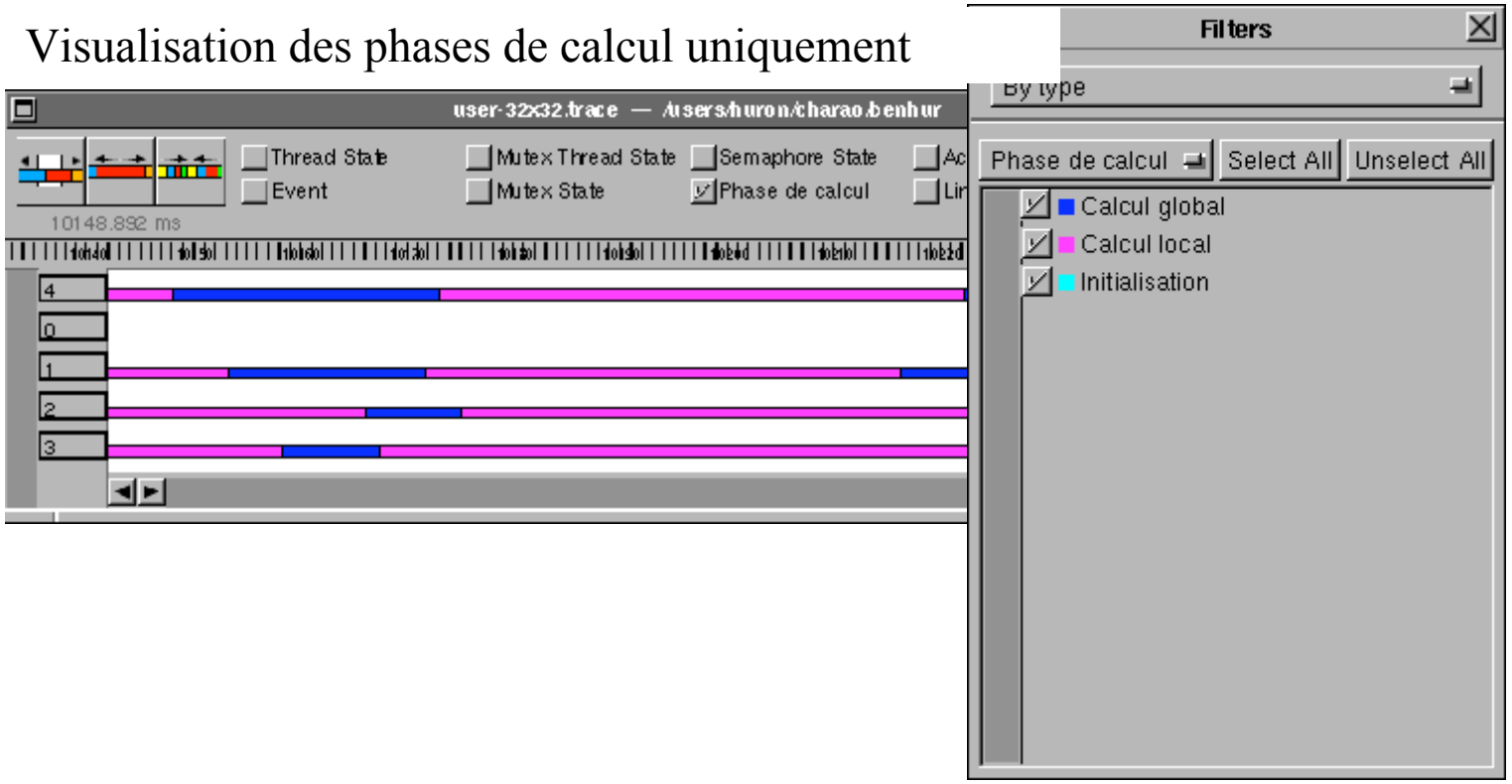
Extensibilité

Création d ’objets du nouveau type

```
unsigned phase_state, init_phase, local_phase, global_phase;
phase_state  = pajeDefineUserStateType ( A0_NODE, "Phase de calcul" );
init_phase   = pajeNewUserEntityValue ( phase_state, "Initialisation" );
local_phase  = pajeNewUserEntityValue ( phase_state, "Calcul local" );
global_phase = pajeNewUserEntityValue ( phase_state, "Calcul global" );

pajeSetUserState ( phase_state, 0, init_phase, "" );
initialisation();
while (!converge) {
   iter++;
   pajeSetUserState ( phase_state, 0, local_phase, 0 );
   calcul_local();
   send (local_data);
   receive (remote_data);
   pajeSetUserState ( phase_state, 0, global_phase, 0 );
   calcul_global();
}
```

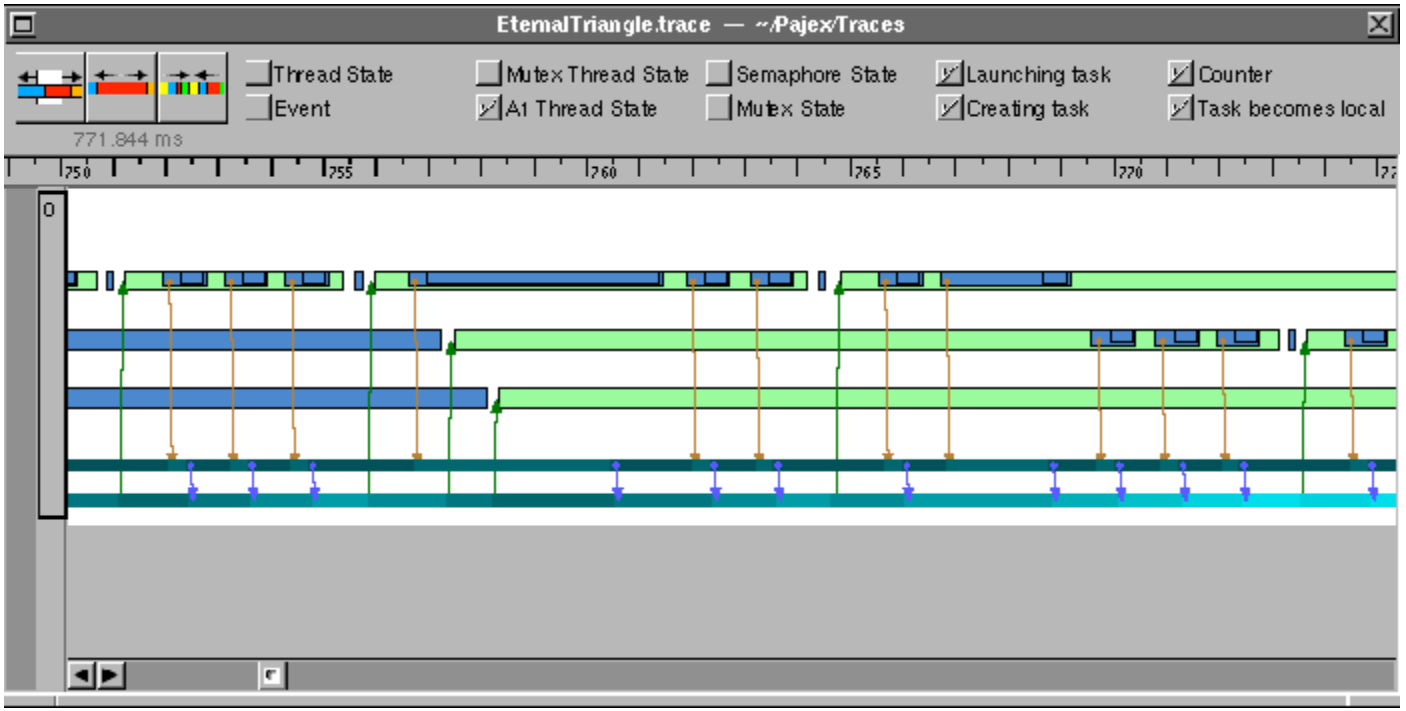
Extensibilité

Visualisation des phases de calcul uniquement



Extensibilité

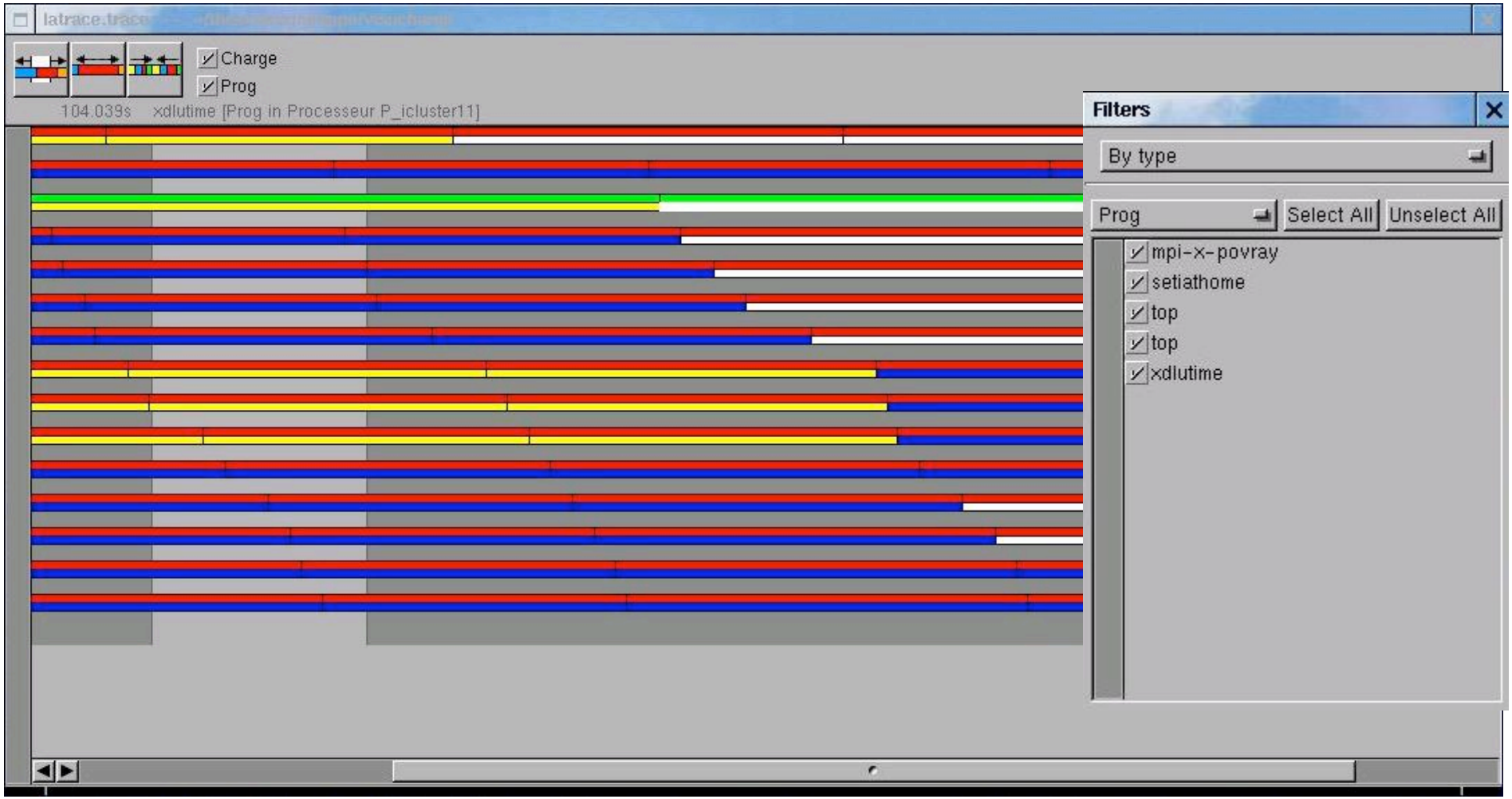
- Visualisation de l'ordonnanceur d'Athapascan-1
- Addition de quelques lignes dans Athapascan-1
- Aucun changement dans Pajé



Exploitation de la généricité de Pajé

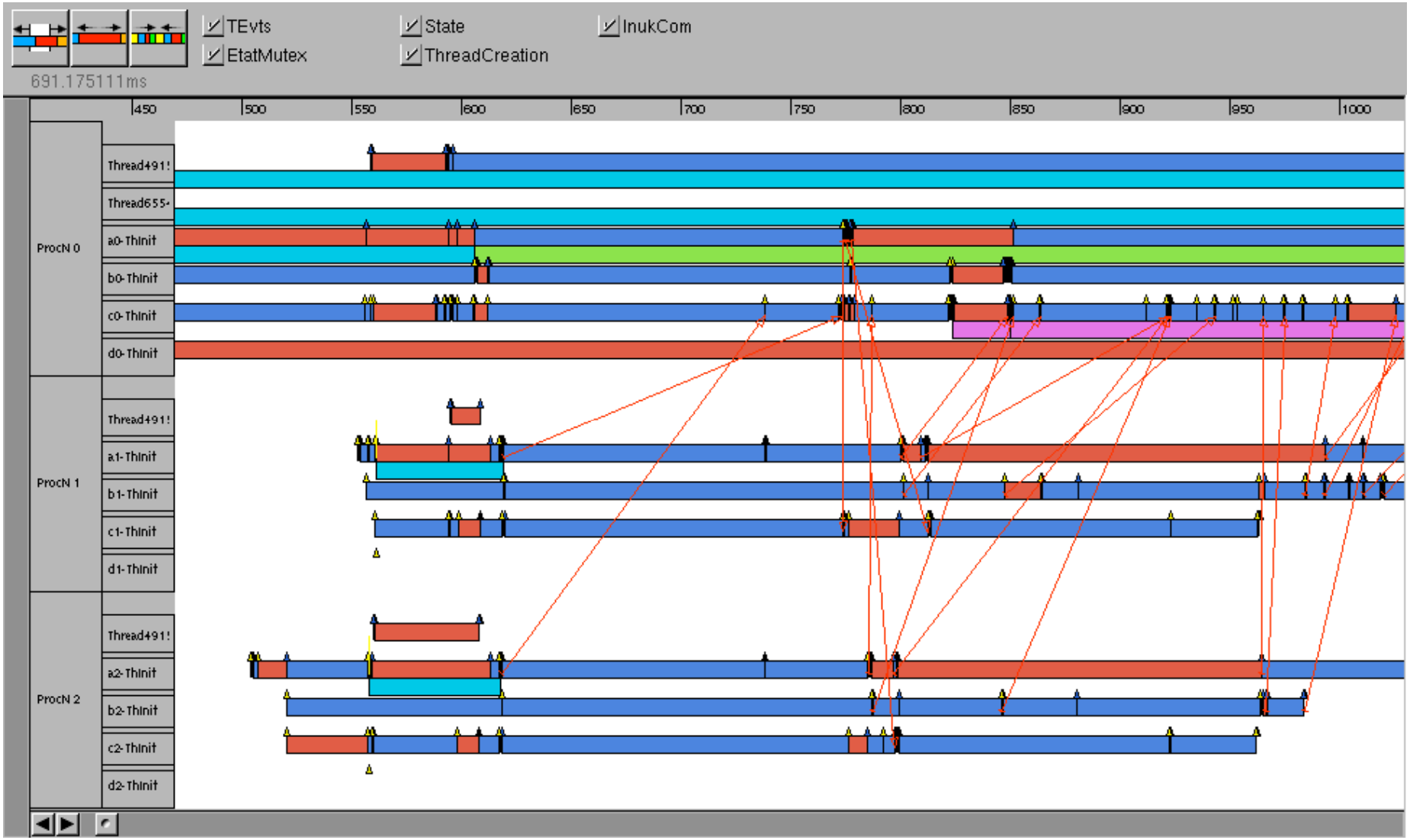
- Utilisation de Pajé pour l ’administration système de grappes (Cyril Guilloud).
- Mesure de performances de programmes Java distribués (François Ottogali):
  - Corrélation traces du niveau application et indices de performance du niveau système.
  - Utilisation de Pajé pour visualisation du comportement des applications.

Cluster administration

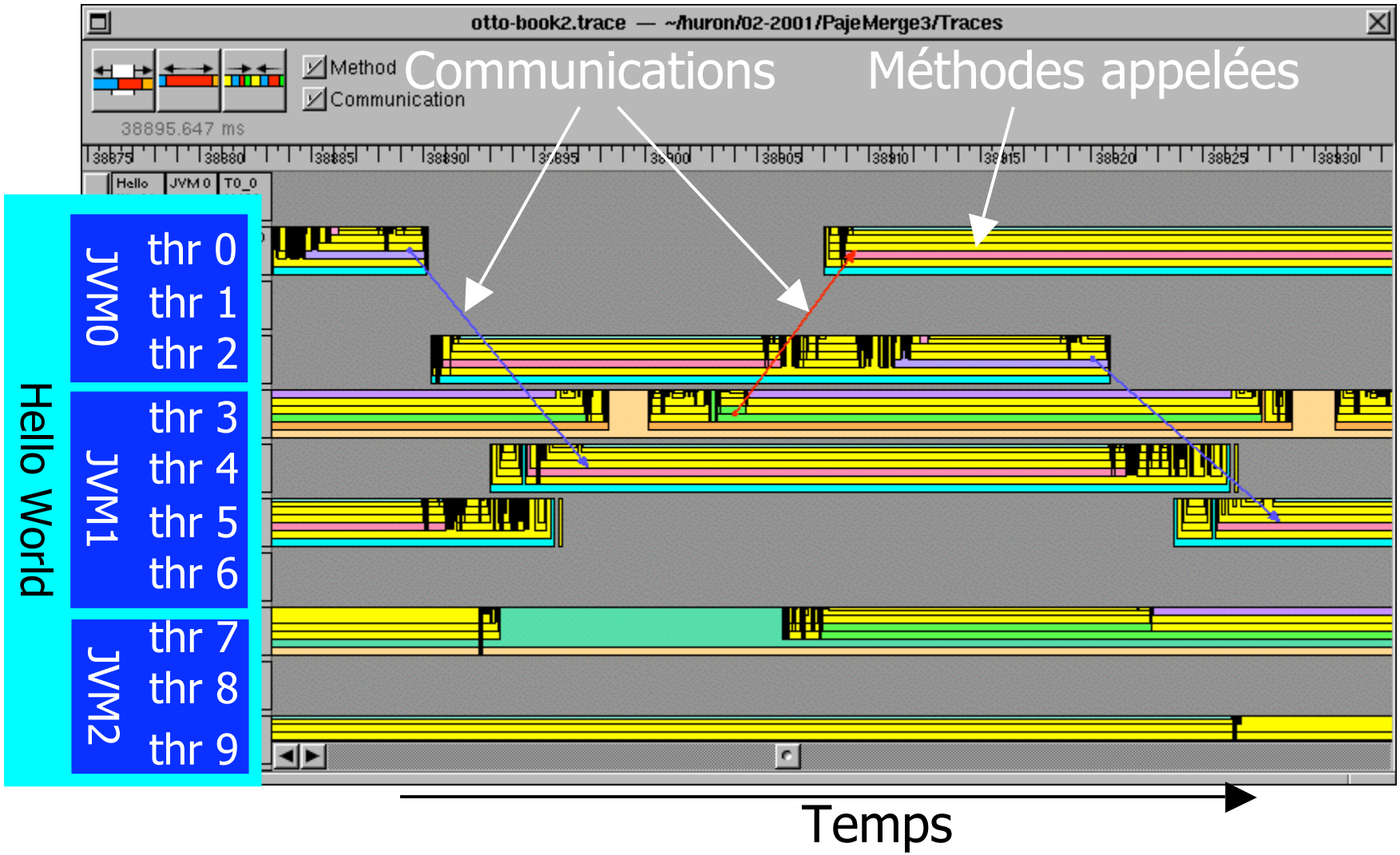



pbs3003.trace — ~/nouveau/Paje
pbs-task
2.136081 d
0
5
10
15
20
switch2
switch3

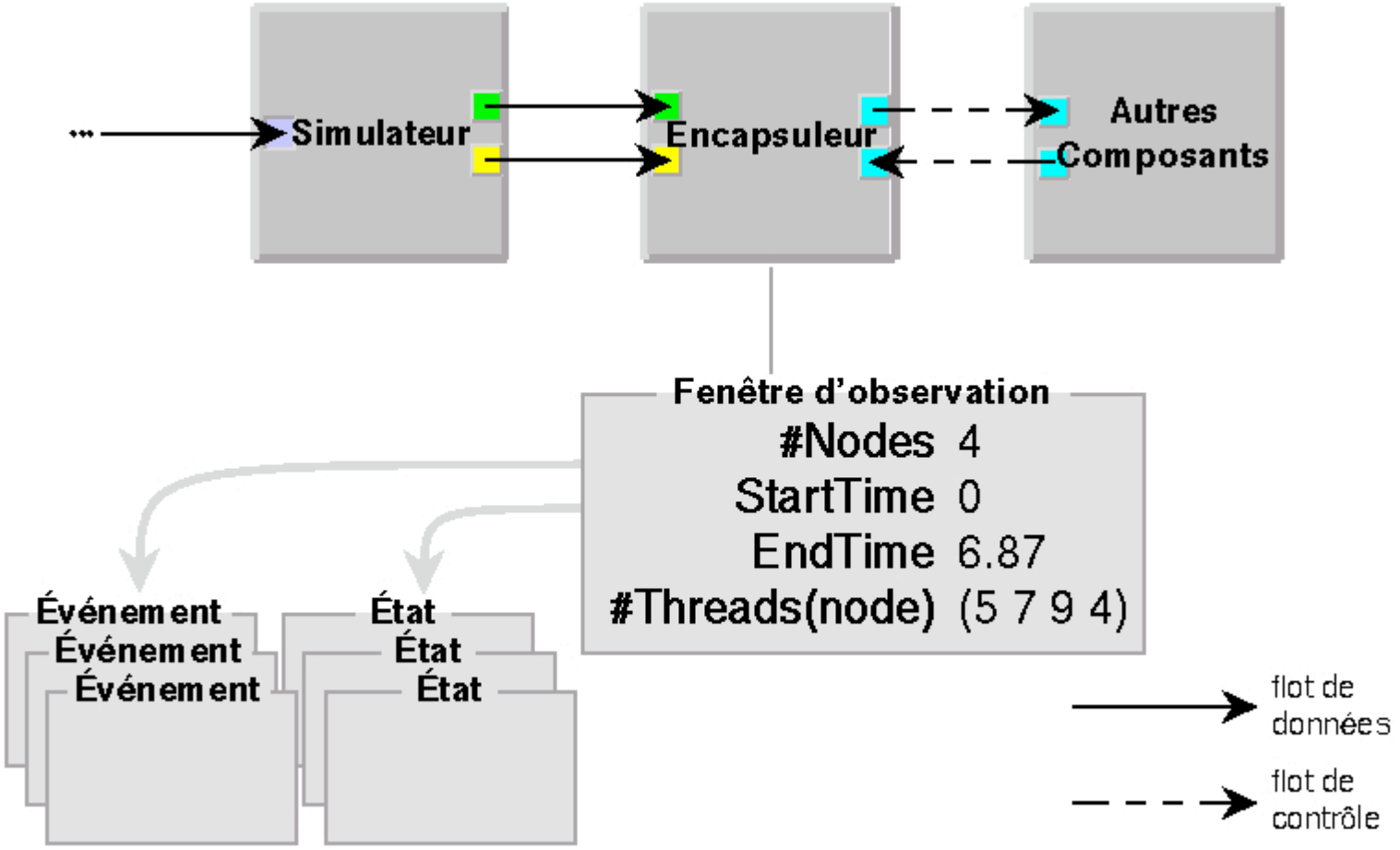
Molecular biology code



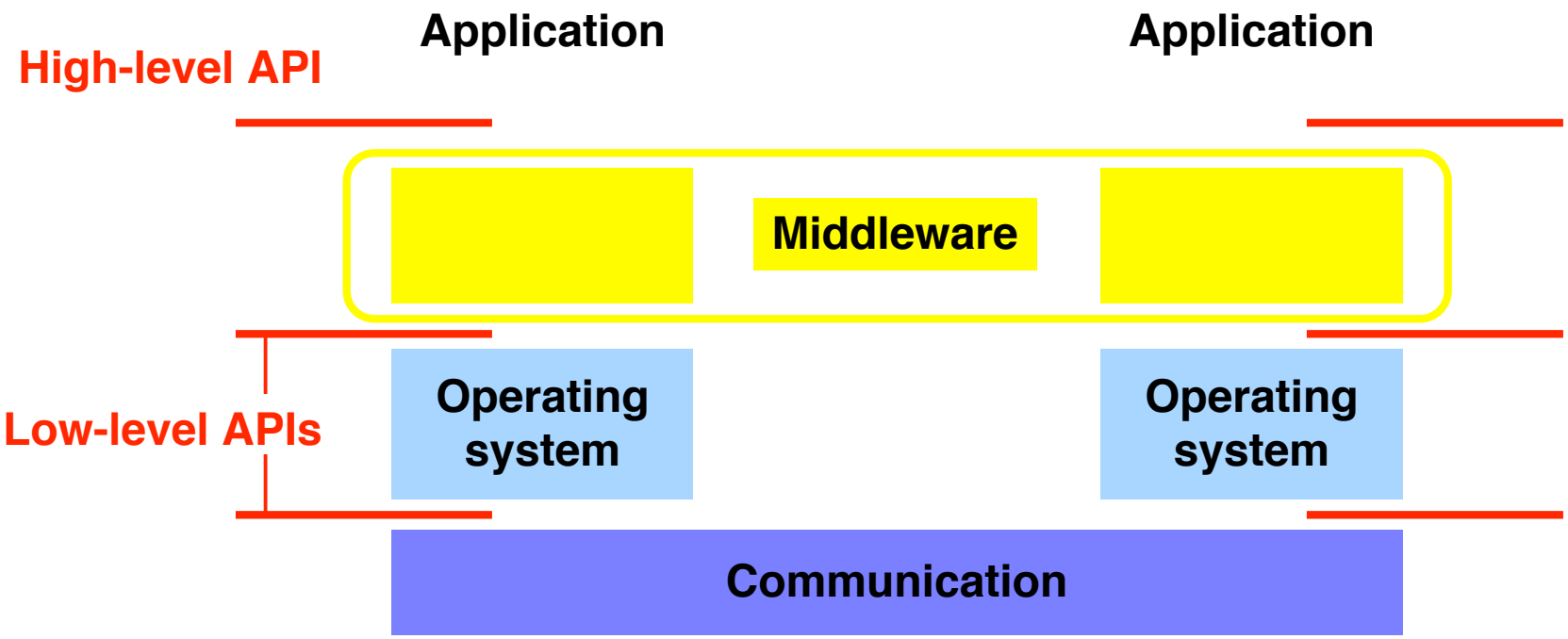
Programme Java distribué

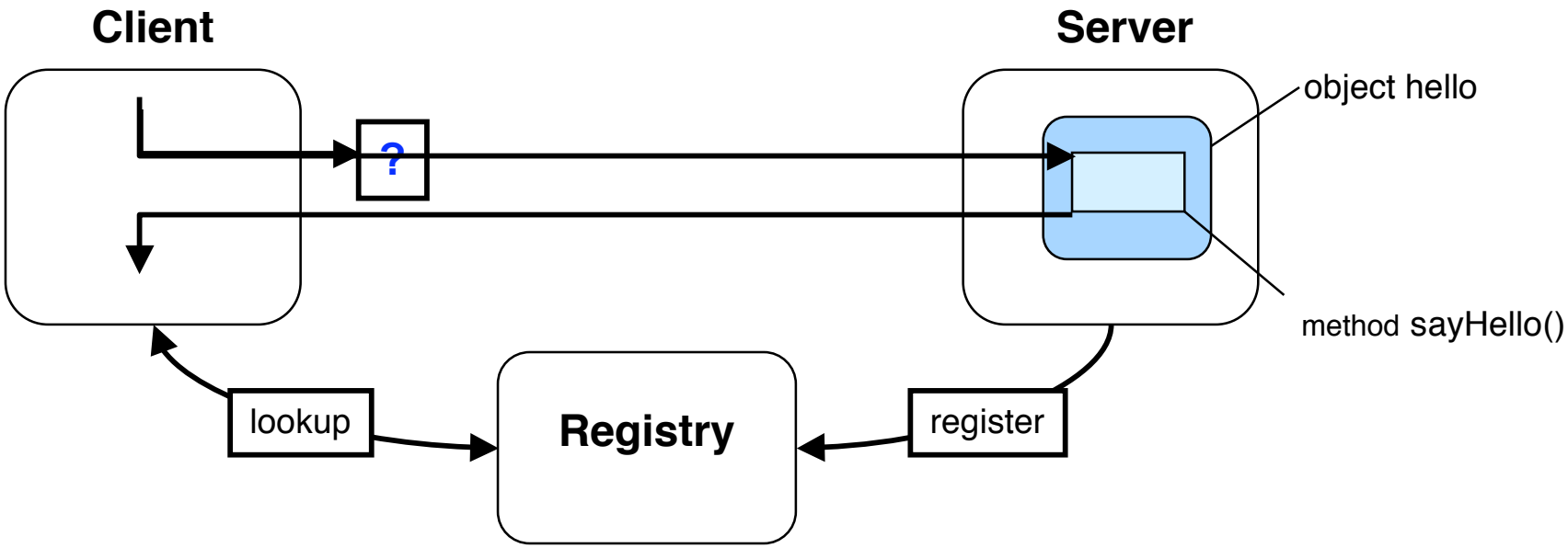


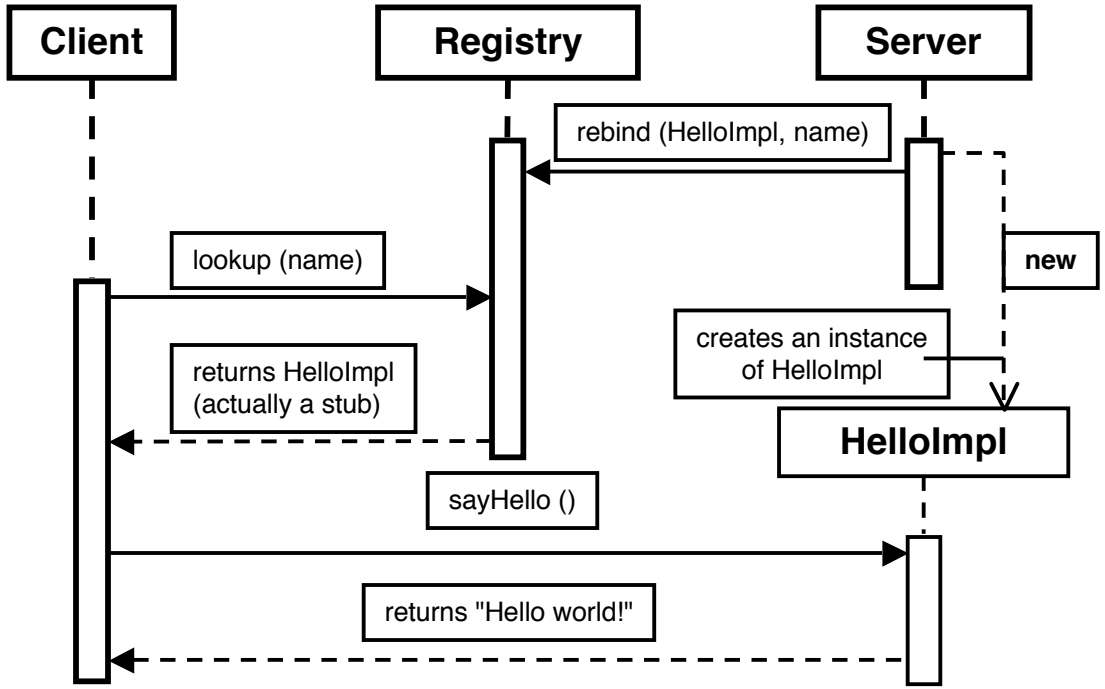
Extensibilité et Interactivité et «scalabilité»

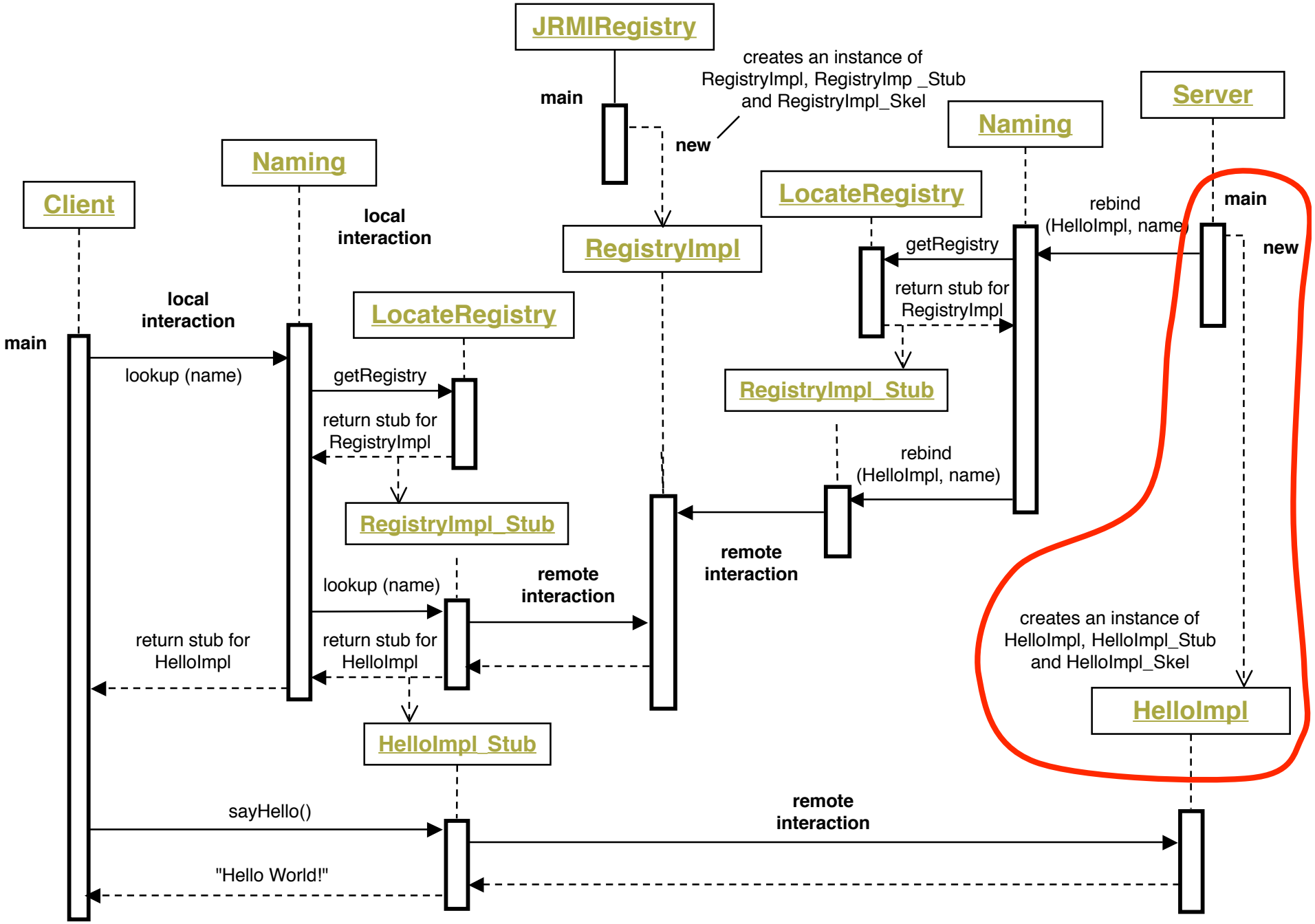


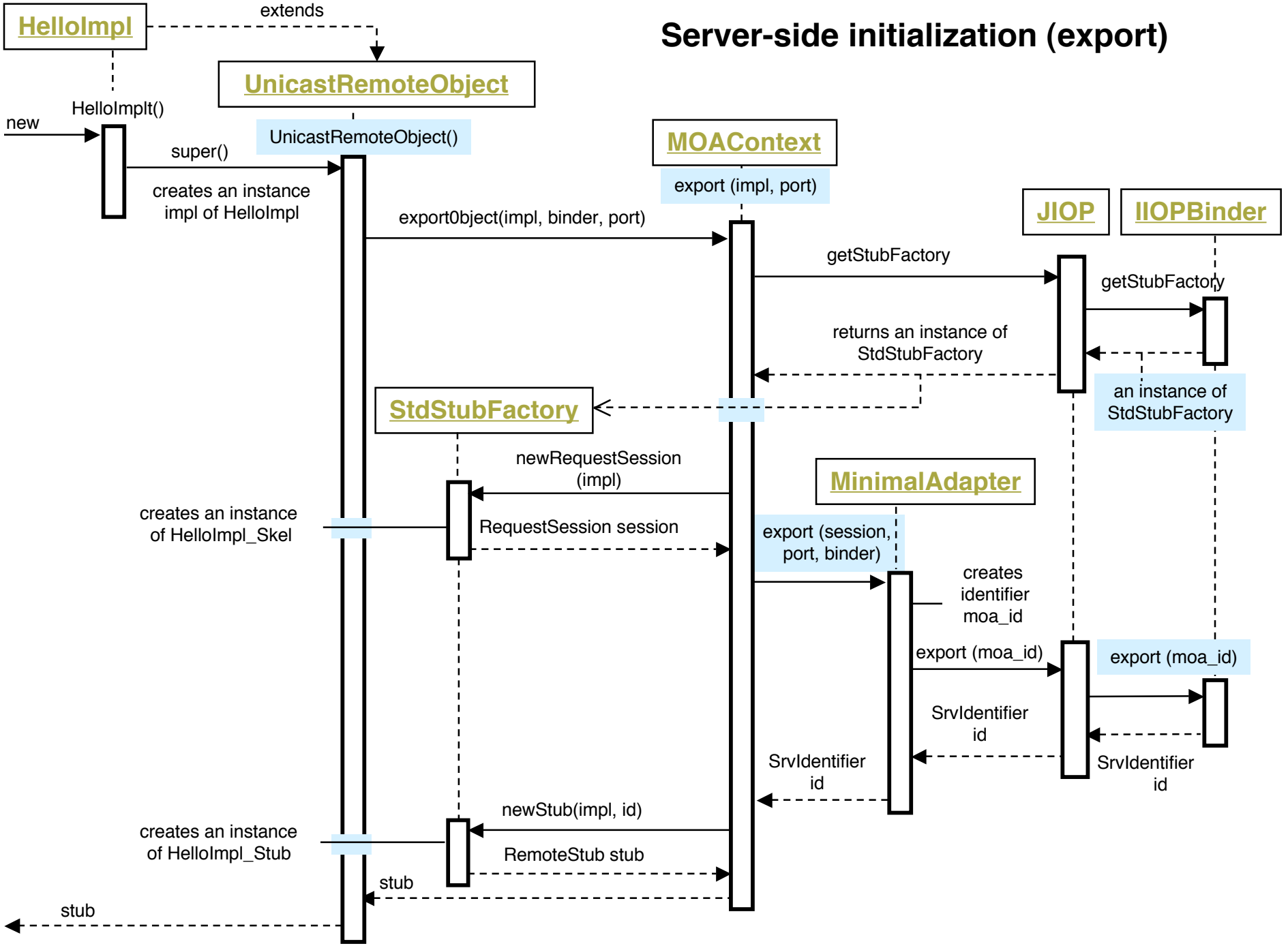
Architecture logicielle

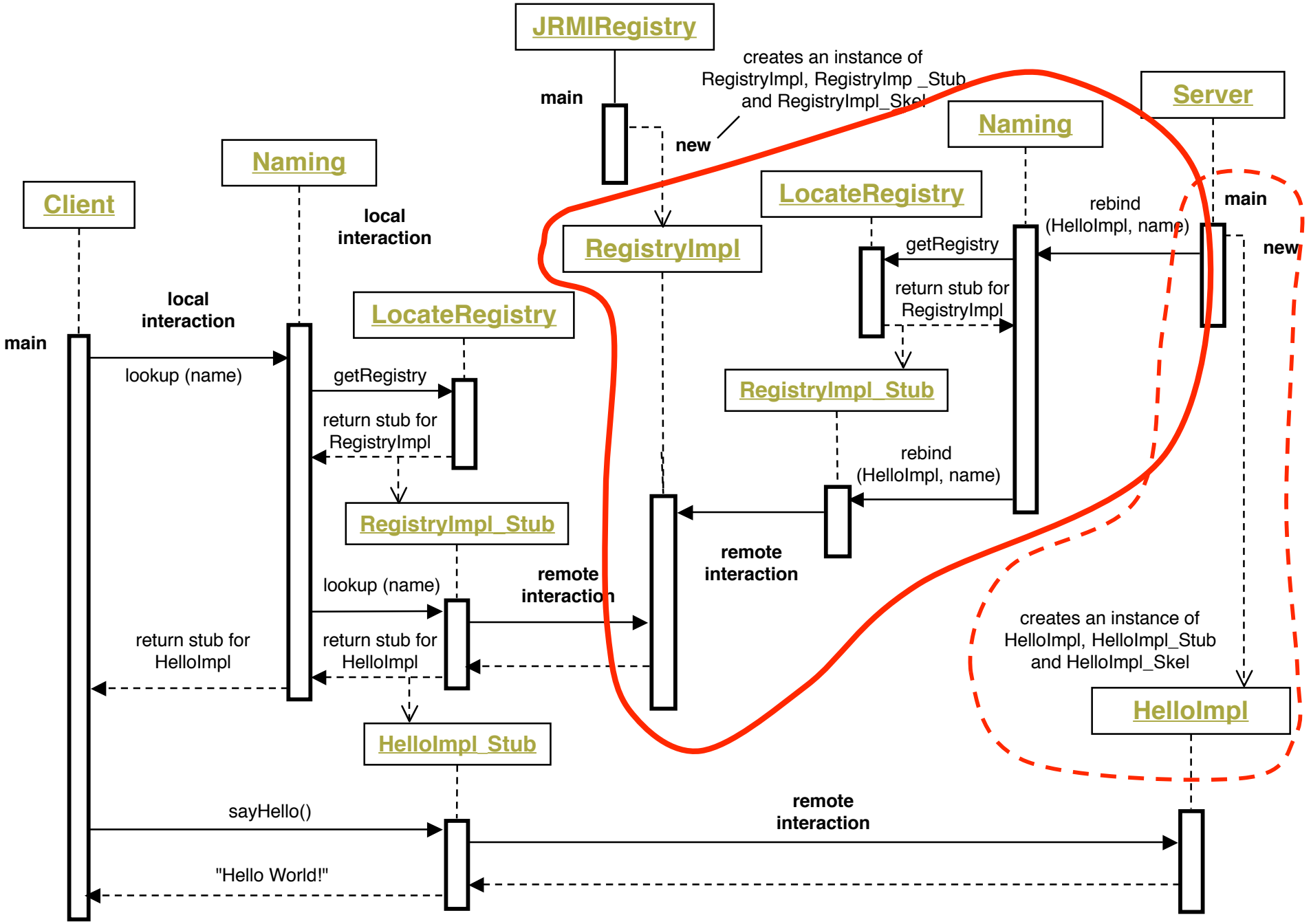



Client
Server
object hello
?
method sayHello()
lookup
Registry
register

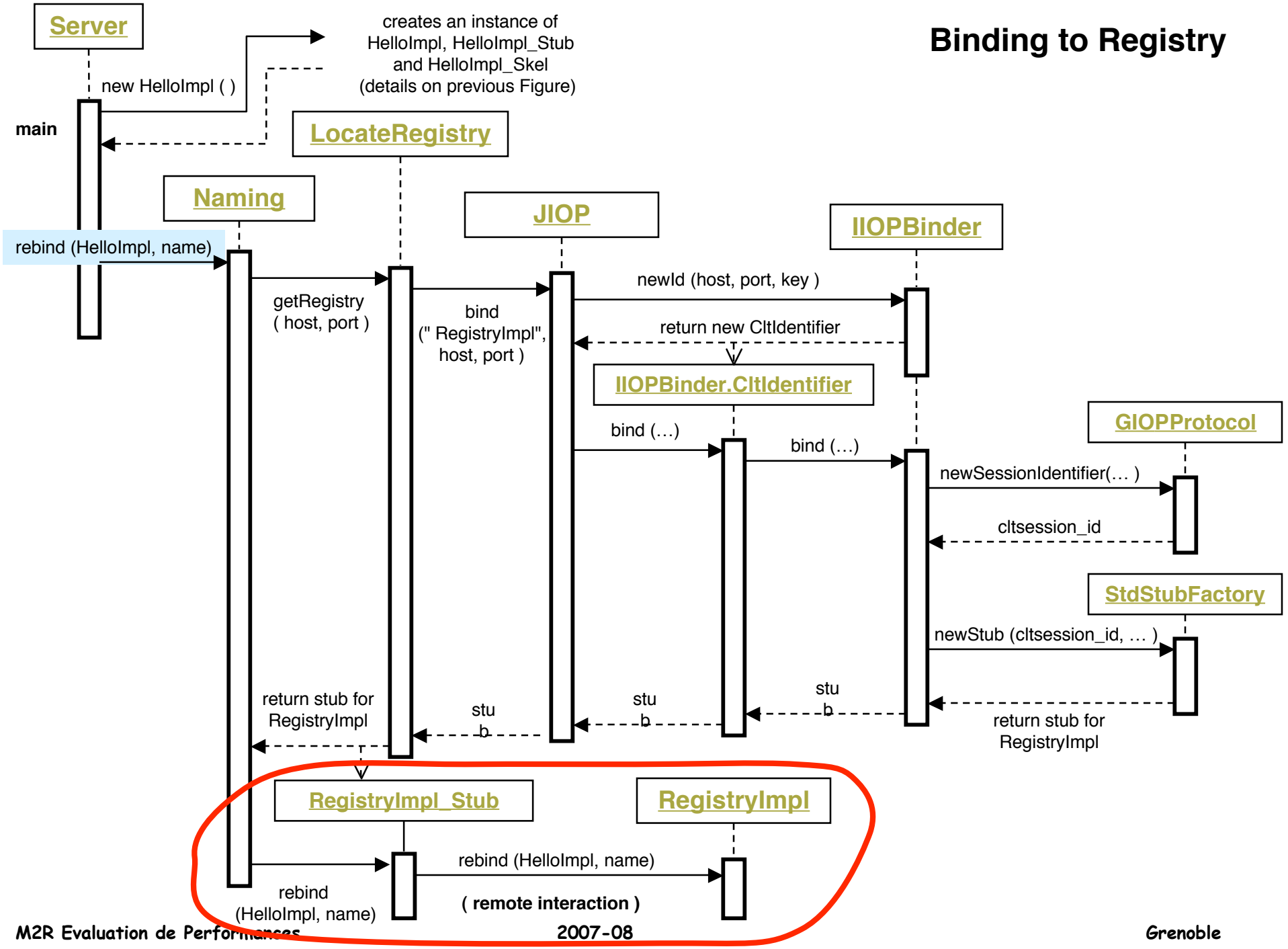

Client
Registry
Server
rebind (HelloImpl, name)
lookup (name)
new
creates an instance
of HelloImpl
returns HelloImpl
(actually a stub)
HelloImpl
sayHello ()
returns "Hello world!"

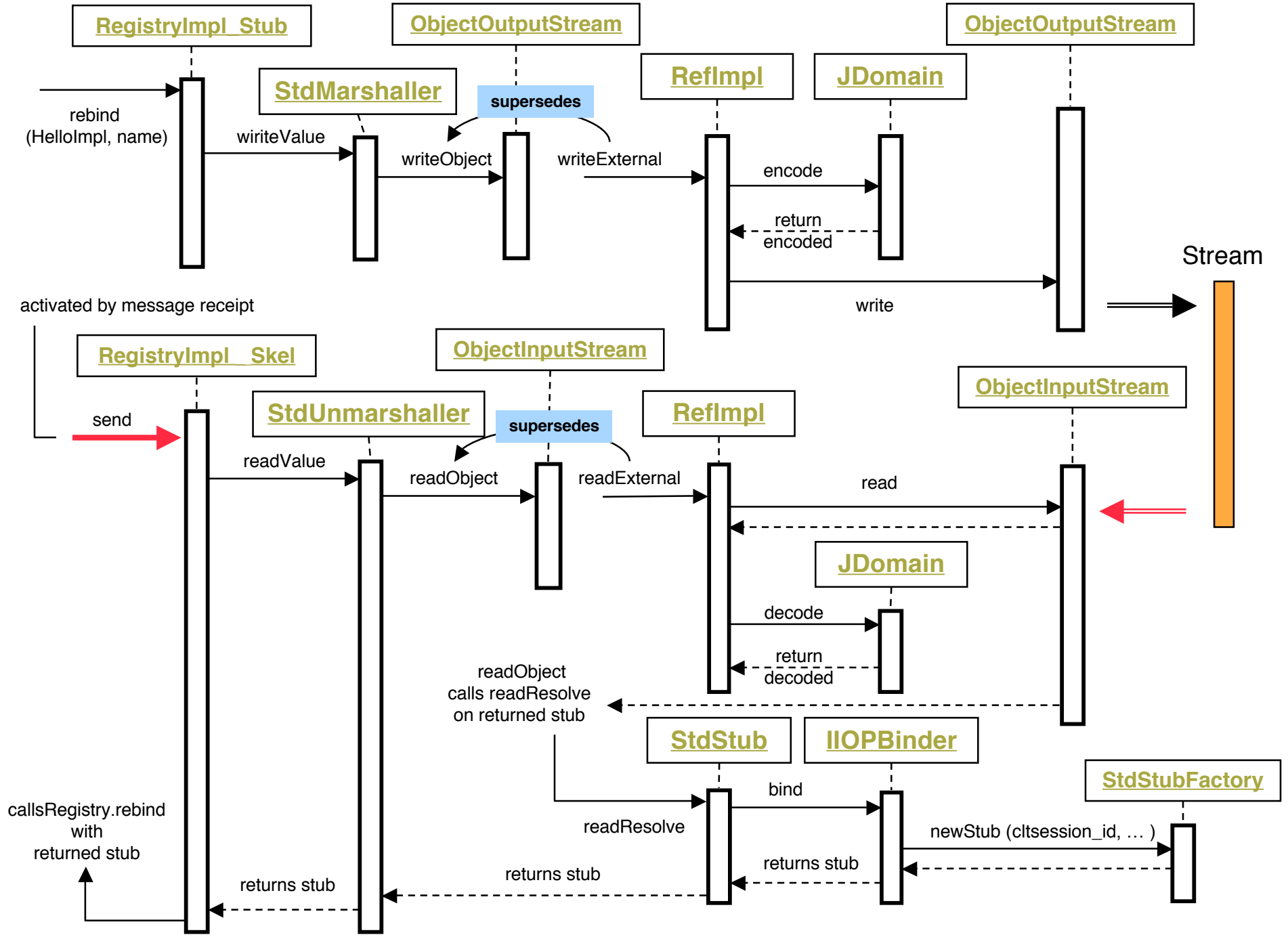

JRMIRegistry
main
creates an instance of
RegistryImpl, RegistryImp _Stub
and RegistryImpl_Skel
new
Server
Naming
Naming
Client
LocateRegistry
local
interaction
RegistryImpl
rebind
(HelloImpl, name)
main
new
getRegistry
return stub for
RegistryImpl
local
interaction
LocateRegistry
main
lookup (name)
getRegistry
RegistryImpl_Stub
return stub for
RegistryImpl
rebind
(HelloImpl, name)
remote
interaction
RegistryImpl_Stub
lookup (name)
remote
interaction
creates an instance of
HelloImpl, HelloImpl_Stub
and HelloImpl_Skel
return stub for
HelloImpl
return stub for
HelloImpl
HelloImpl
HelloImpl_Stub
sayHello()
remote
interaction
"Hello World!"


Server-side initialization (export)
HelloImpl
extends
UnicastRemoteObject
HelloImplt()
new
super()
UnicastRemoteObject()
creates an instance
impl of HelloImpl
MOAContext
export (impl, port)
export0bject(impl, binder, port)
JIOP
IIOPBinder
getStubFactory
getStubFactory
returns an instance of
StdStubFactory
an instance of
StdStubFactory
StdStubFactory
newRequestSession
(impl)
MinimalAdapter
creates an instance
of HelloImpl_Skel
RequestSession session
export (session,
port, binder)
creates
identifier
moa_id
export (moa_id)
export (moa_id)
SrvIdentifier
id
SrvIdentifier
id
SrvIdentifier
id
newStub(impl, id)
creates an instance
of HelloImpl_Stub
RemoteStub stub
stub
stub


JRMIRegistry
main
creates an instance of
RegistryImpl, RegistryImp _Stub
and RegistryImpl_Skel
new
Server
Naming
Client
Naming
local
interaction
LocateRegistry
RegistryImpl
rebind
(HelloImpl, name)
main
new
getRegistry
return stub for
RegistryImpl
local
interaction
main
lookup (name)
LocateRegistry
getRegistry
return stub for
RegistryImpl
RegistryImpl_Stub
rebind
(HelloImpl, name)
RegistryImpl_Stub
remote
interaction
remote
interaction
lookup (name)
return stub for
HelloImpl
return stub for
HelloImpl
creates an instance of
HelloImpl, HelloImpl_Stub
and HelloImpl_Skel
HelloImpl
HelloImpl_Stub
sayHello()
remote
interaction
"Hello World!"

Binding to Registry




RegistryImpl_Stub
ObjectOutputStream
ObjectOutputStream
StdMarshaller
supersedes
RefImpl
JDomain
rebind
(HelloImpl, name)
wiriteValue
writeObject
writeExternal
encode
return
encoded
Stream
write
activated by message receipt
RegistryImpl_Skel
ObjectInputStream
ObjectInputStream
send
StdUnmarshaller
supersedes
RefImpl
readValue
readObject
readExternal
read
JDomain
decode
return
decoded
readObject
calls readResolve
on returned stub
StdStub
IIOPBinder
StdStubFactory
bind
readResolve
newStub (cltsession_id, …)
returns stub
returns stub
callsRegistry.rebind
with
returned stub
returns stub

Modèles et mesure

Définition états – transitions

→ niveau d'abstraction

→ abstraction de l'exécution

État **intéressant**

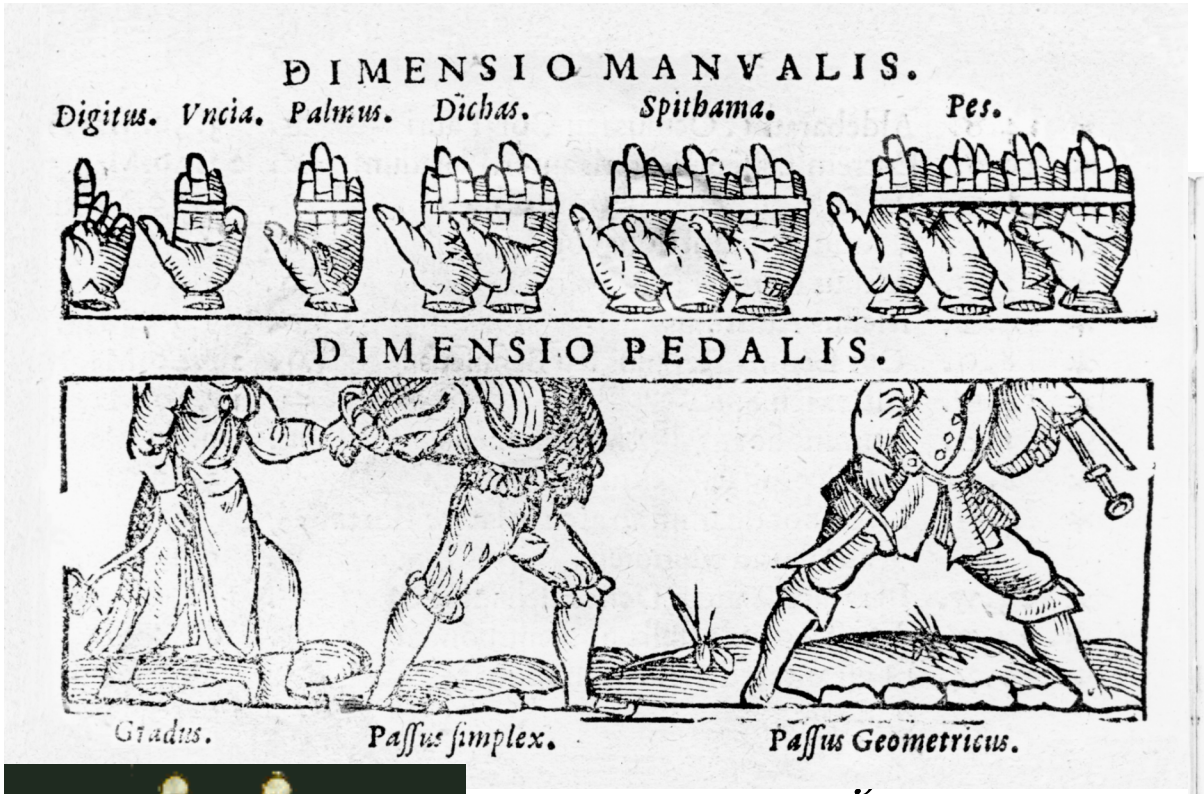
➔ action de l'utilisateur (code, ordonnancement,…)

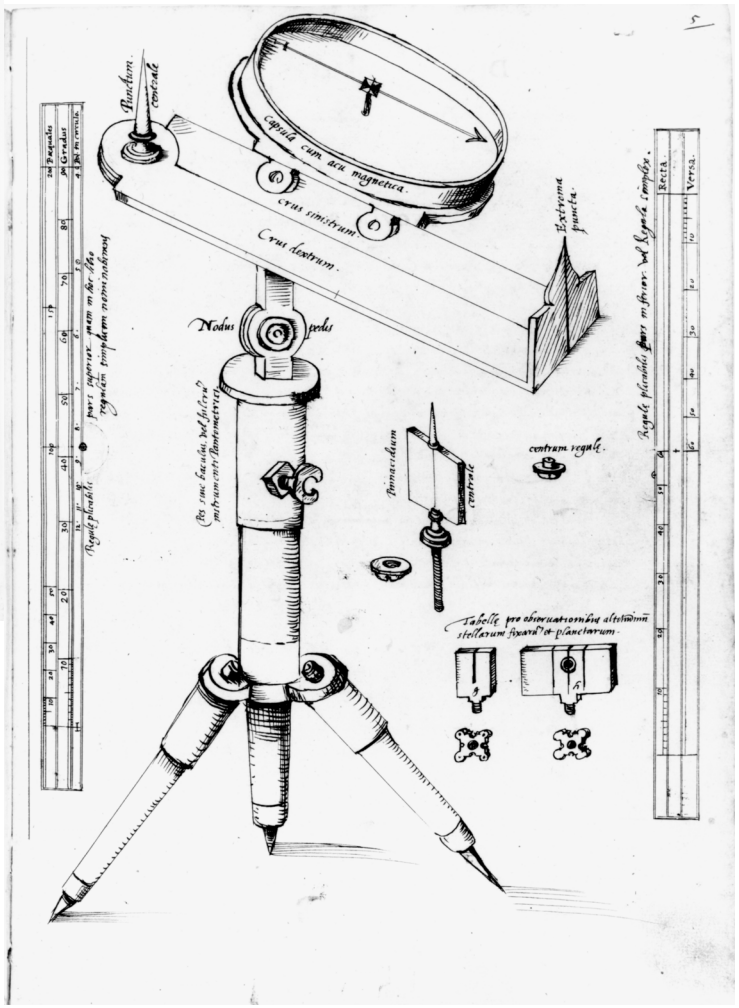
État **observable**

➔ instrument de mesure associé

Conception et instrumentation ←→ Intégration d'instruments

Exécution séquentielle





Techniques : time references

› Clock driven measurments

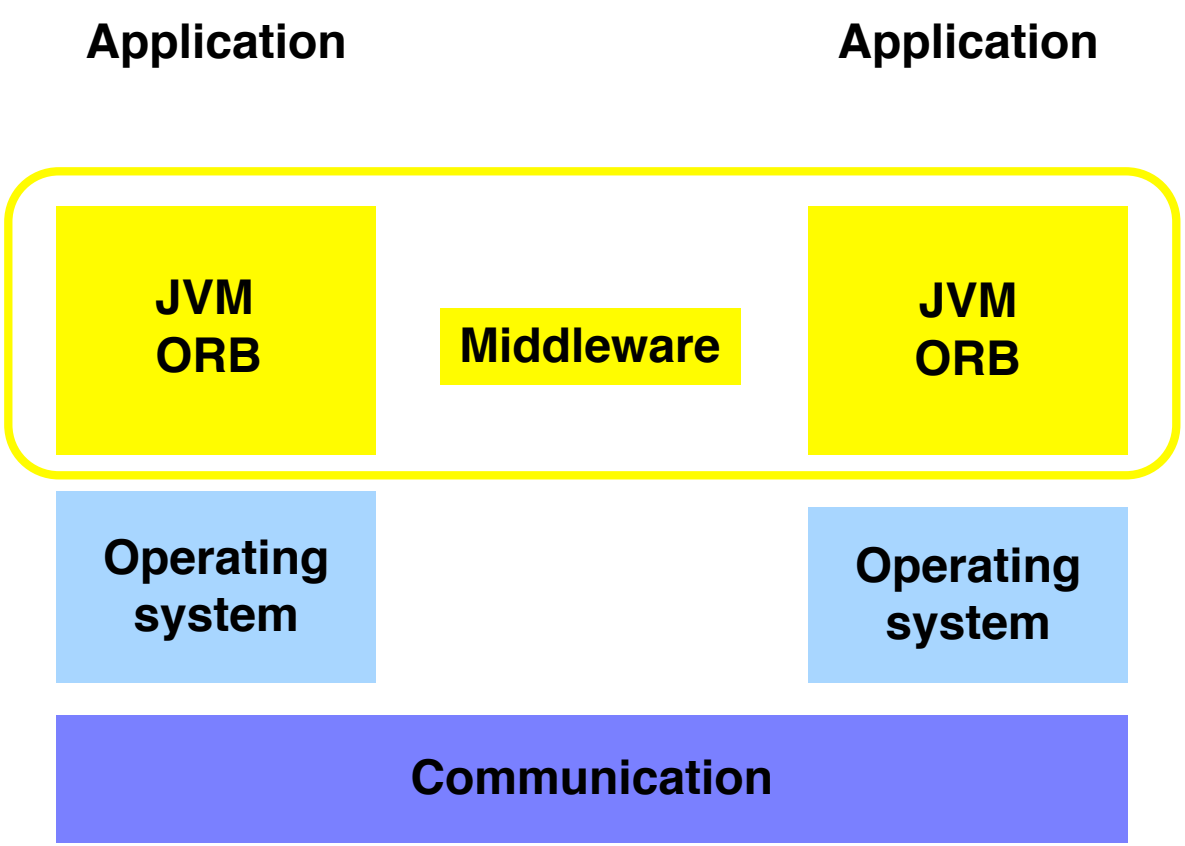
(a)

(b)

(c)

(d)

Architecture logicielle



événements **utilisateur**
- état de variables,…

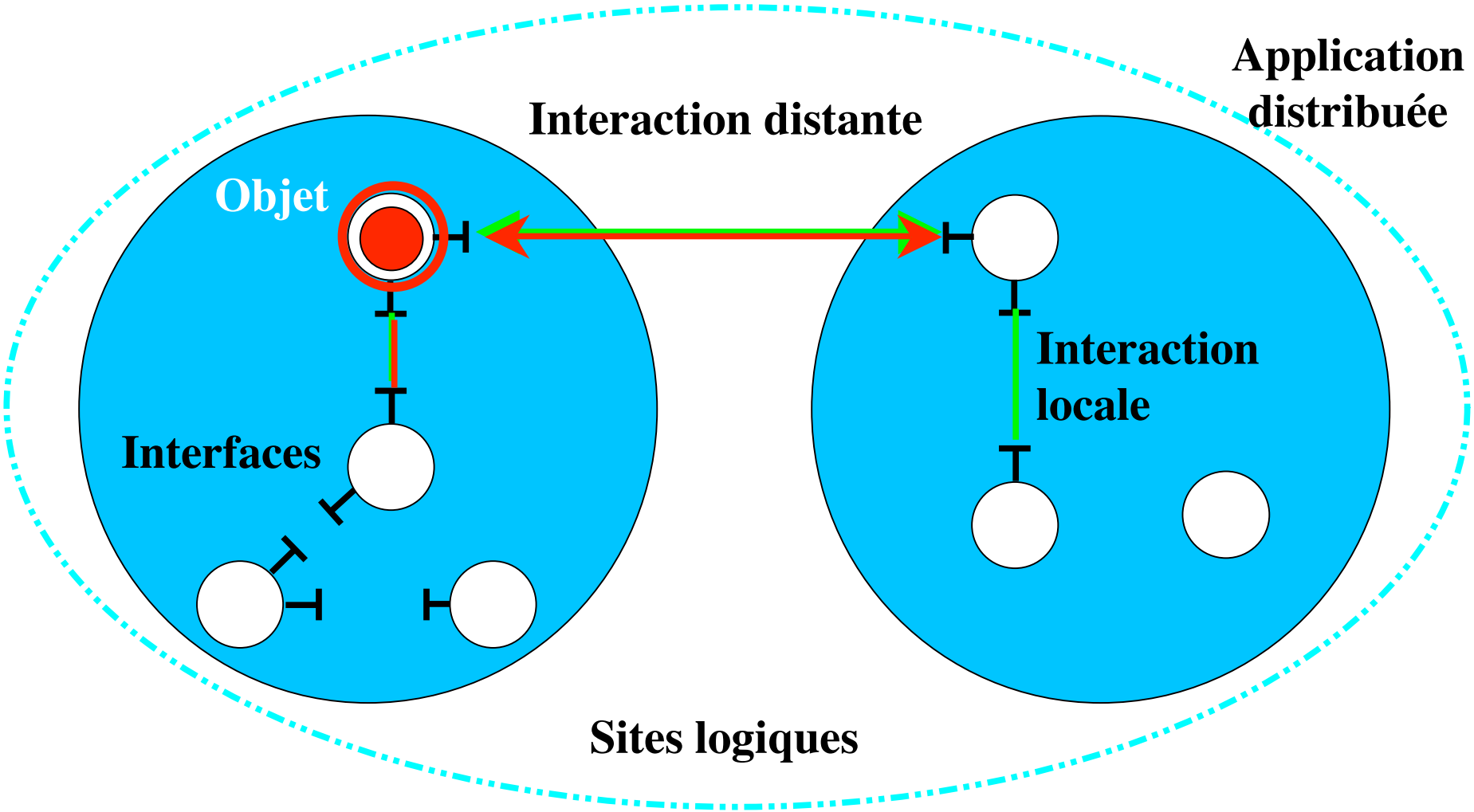
événements **utilisateur JVM**
-appels de méthode,
-chargement de classe,…

événements **gestion JVM**
-threads,
-moniteurs,…

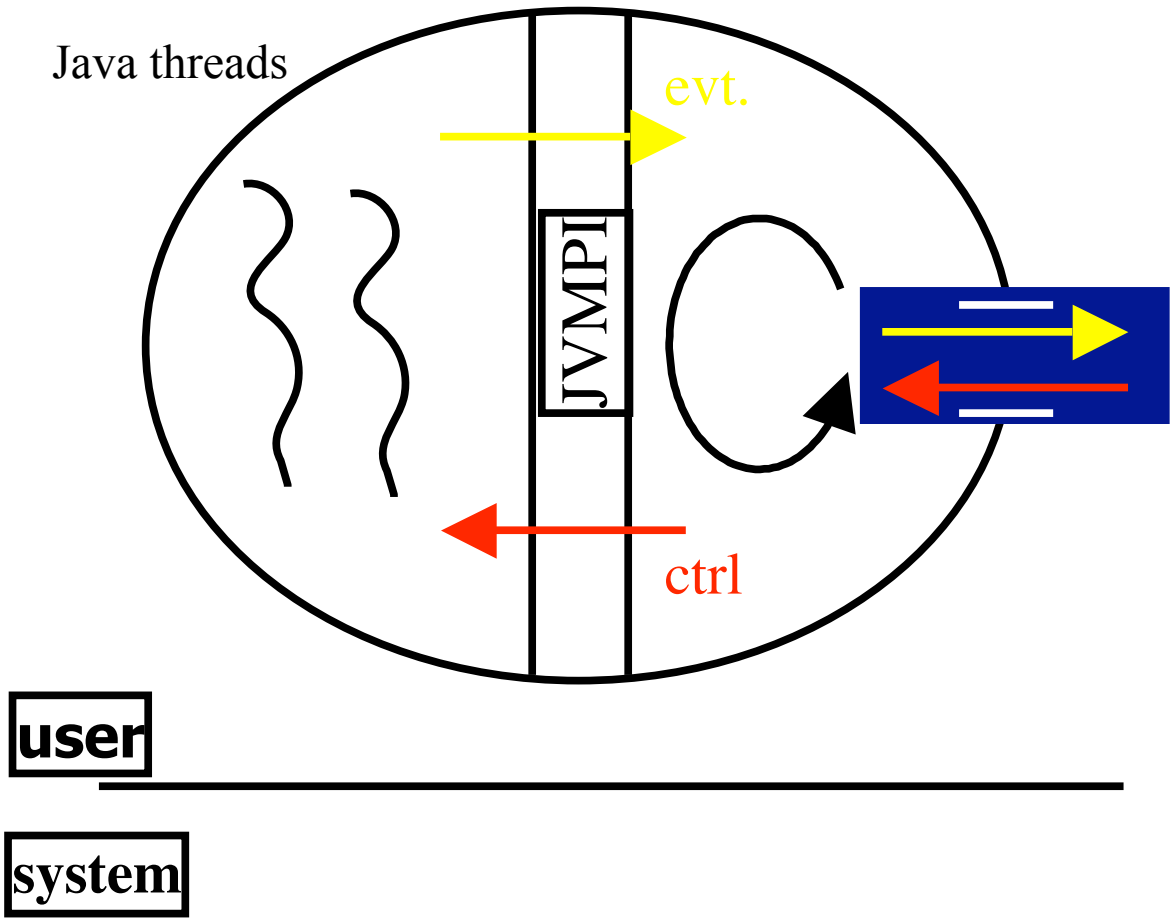
événements **système**
- état des threads système,
- sockets, mémoire,…

événements **réseau**
- état des liens,routeurs,…

Application distribuée à objets



JVM Profiling Interface



Timestamped events:

- **classes**
- **objects**
- **methods**
- ***monitors***
- ***GC***
- ***threads***
- ***JVM***

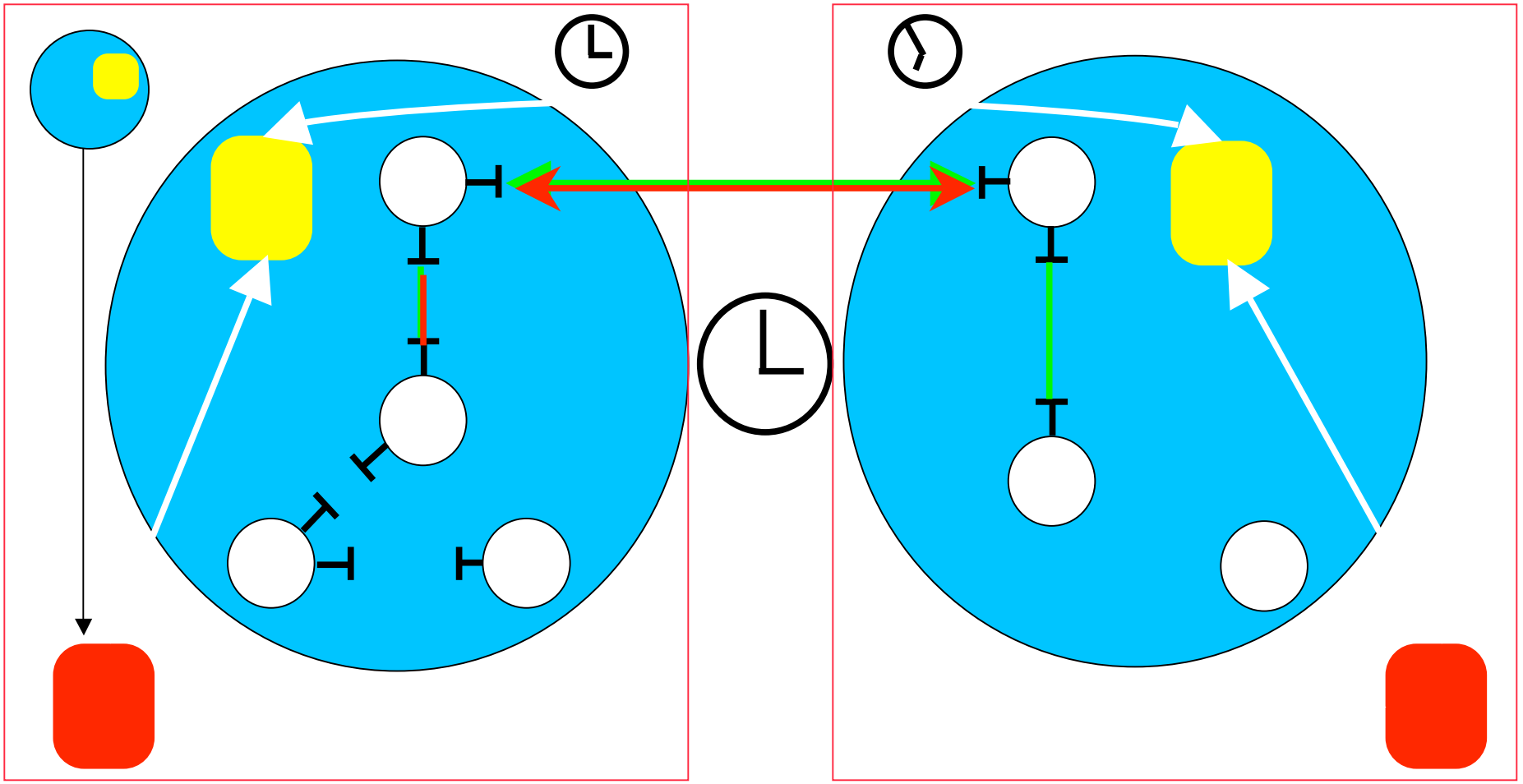
Control :

- **threads scheduling**
- **GC**
- **execution context**

Trace locale filtrée et indentée

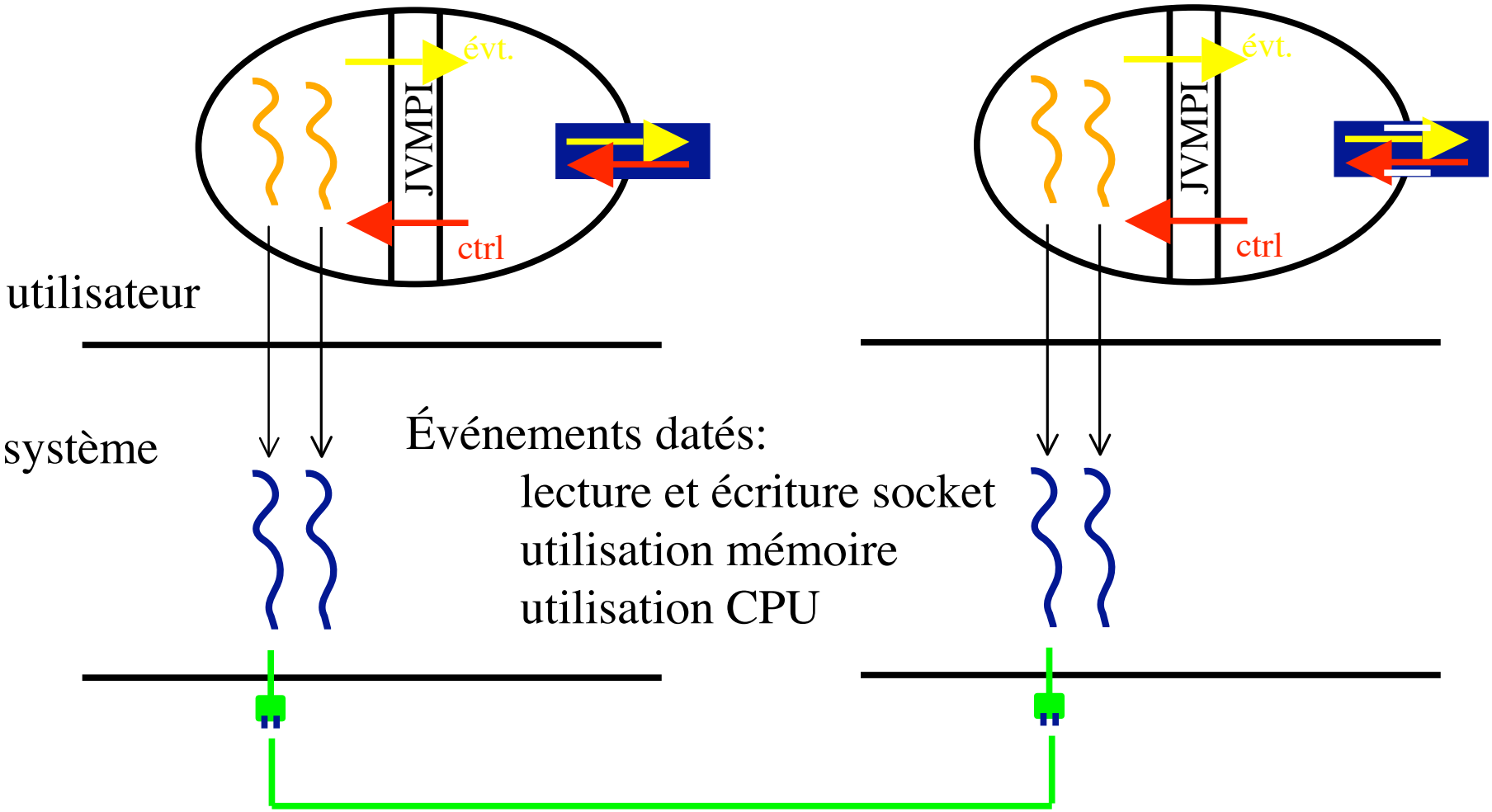
```
Client/<clinit> : D F
Client/main : D
|  Client/<init> : D F
|  org/objectweb/david/libs/contexts/orbs/ORBSingletonClass/<init> : D F
|  org/objectweb/david/libs/contexts/orbs/ORBClass/<clinit> : D F
|  org/objectweb/david/libs/contexts/orbs/iiop/IIOPORB/<clinit> : D F
|  org/objectweb/david/libs/contexts/orbs/iiop/IIOPORB/<init> : D
|  |  org/objectweb/david/libs/contexts/orbs/ORBClass/<init> : D
|  |  |  org/objectweb/david/libs/contexts/orbs/ORBSingletonClass/<init> : D F
|  |  |  org/objectweb/david/libs/contexts/orbs/ORBSingletonClass/setDefault : D F
|  |  org/objectweb/david/libs/contexts/orbs/ORBClass/<init> : F
|  org/objectweb/david/libs/contexts/orbs/iiop/IIOPORB/<init> : F
```

Cohérence d'observations distribuées

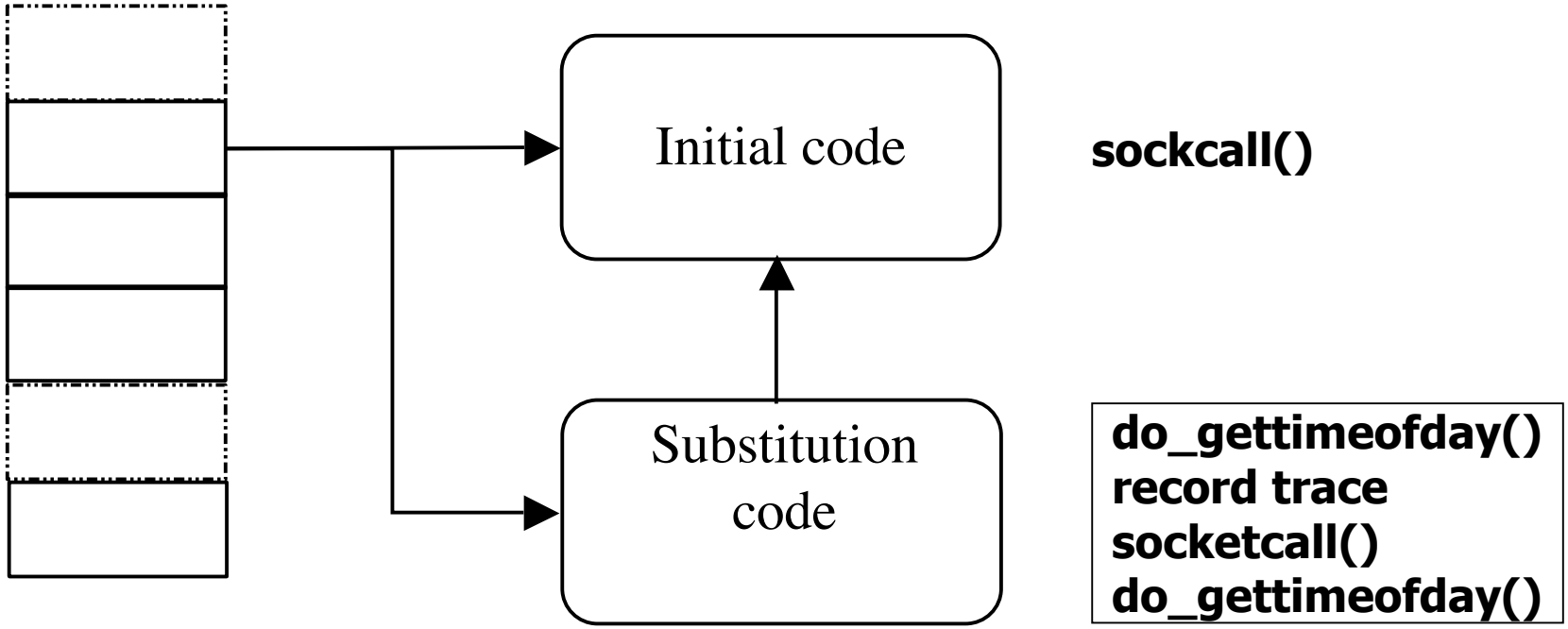


Architecture distribuée

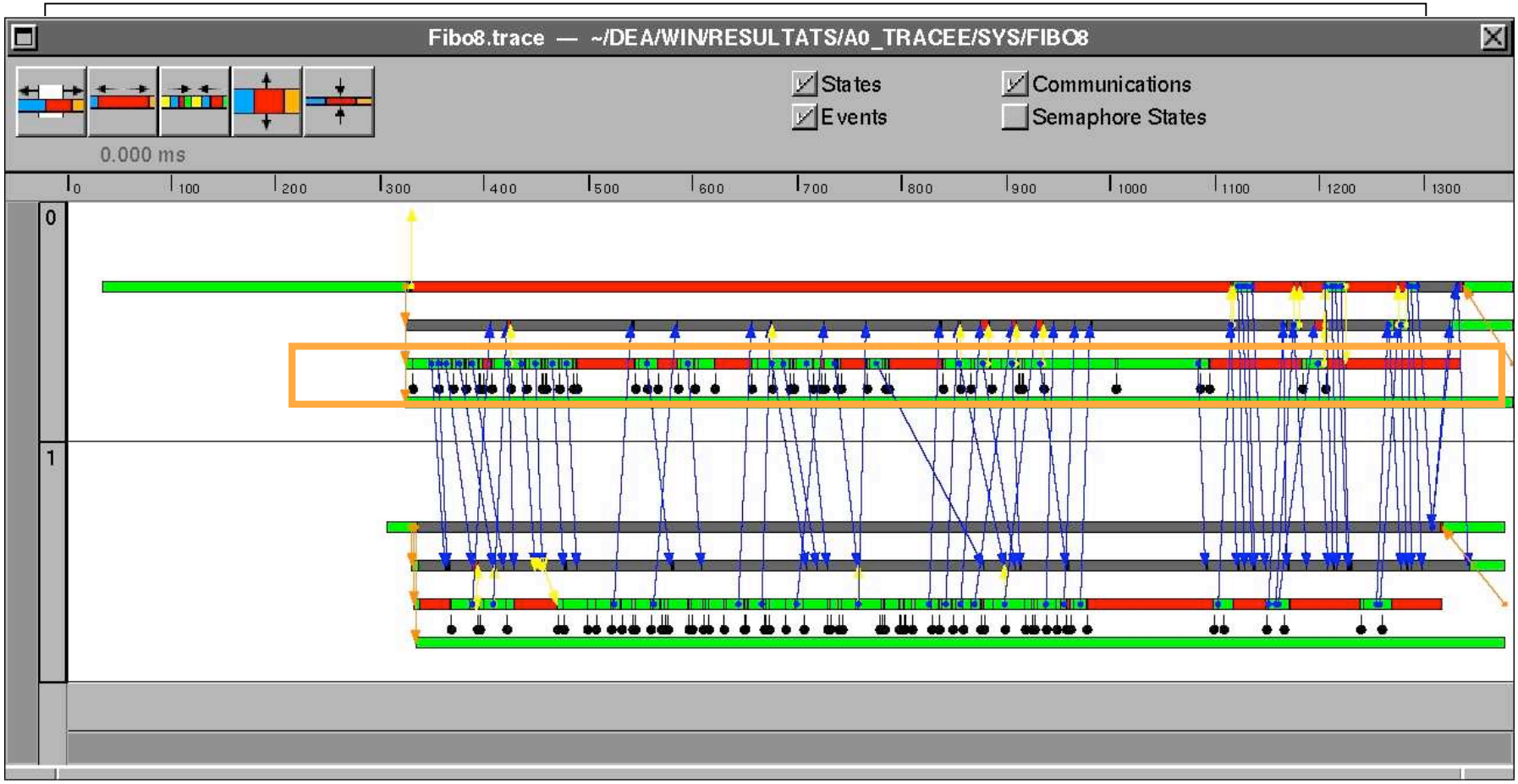
Observations systèmes



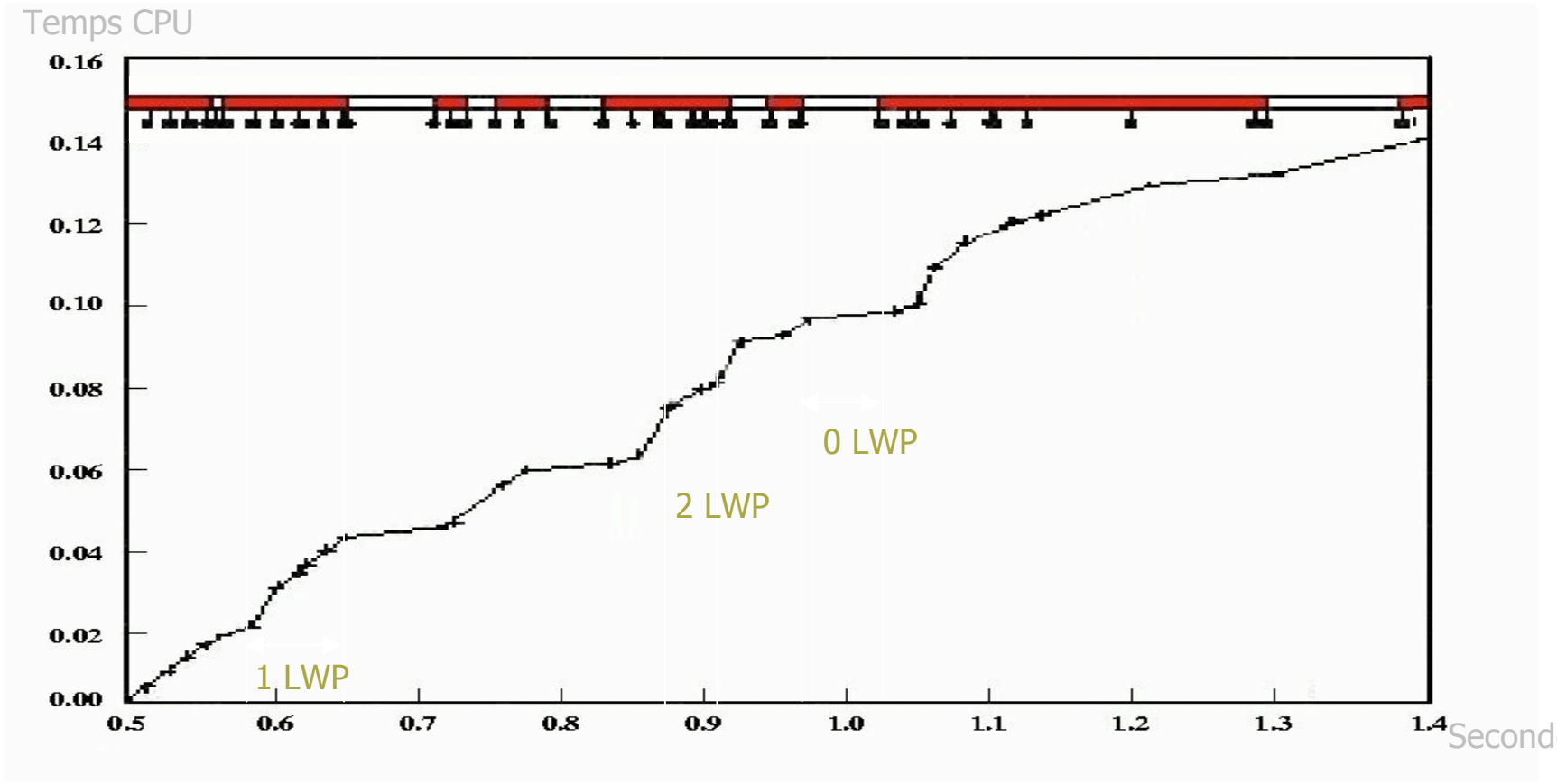
Derouting of system calls



Modèle
Threads + communications



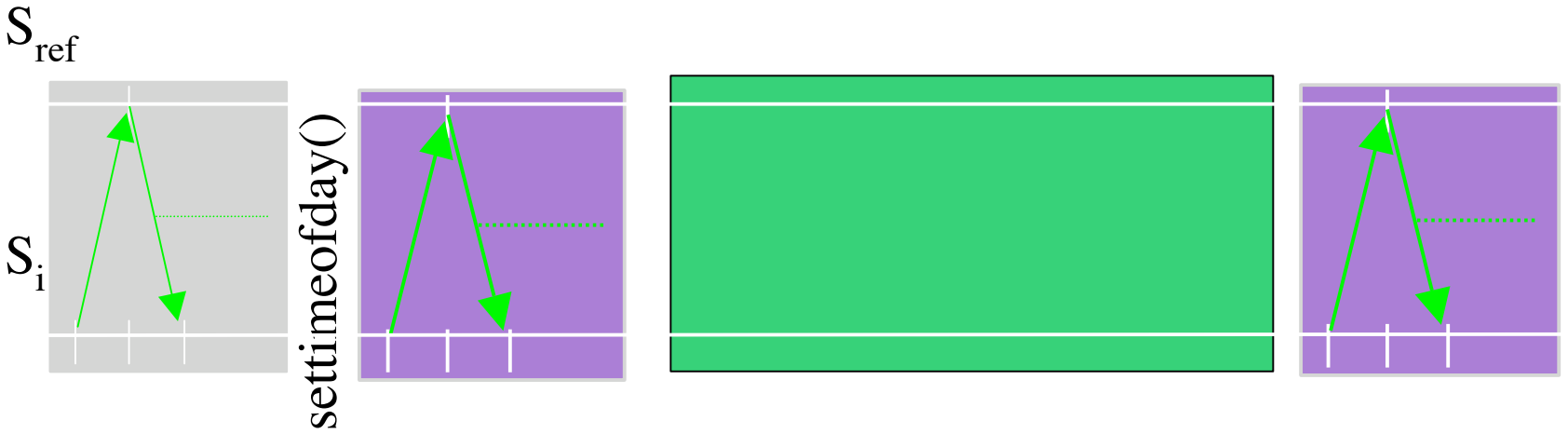
Traceurs systèmes / événements applicatifs



[Ottogalli Vincent 99a,99b

Time in distributed systems

Experimental protocol



retiming **Estimation of clocks parameters** **execution** **Estimation of clocks parameter**

correction post-mortem
analysis
visualization

Intégration et mise en cohérence

- Événements utilisateurs → Threads Java (filtre JVMPI sur appel de méthode)
- Événements JVM locaux →JVMPI
- Interactions distantes

  Thread Java → thread système → socket id
- Horloge globale

Observation = perturbation

Traceur et application partagent les mêmes ressources

Calcul : modification du temps d'exécution

- estimation, correction

Mémoire : stockage local des traces

- gestion de cache, ralentissement,

Réseau : extraction des traces locales

- latences, débits,

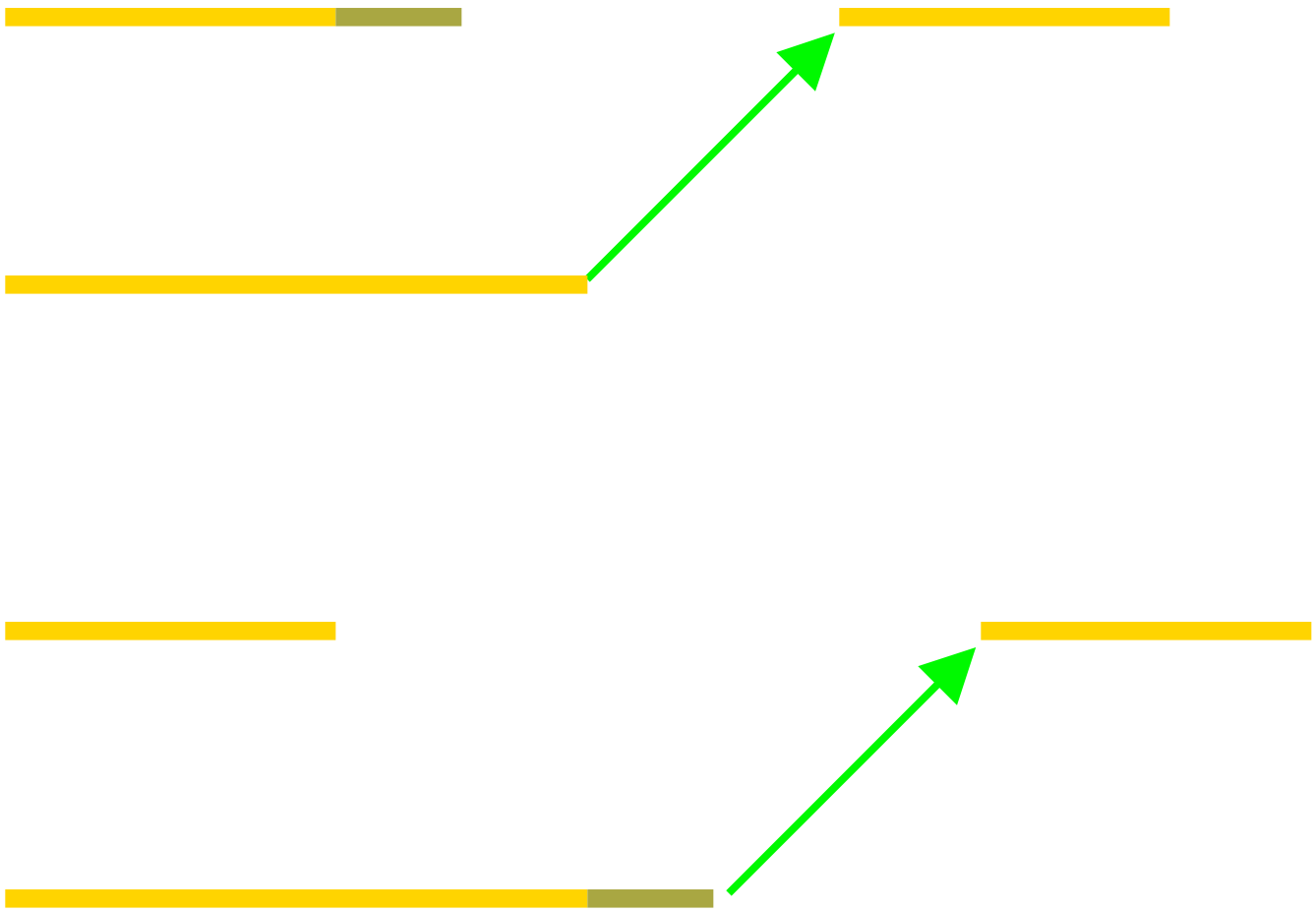
Analyse des perturbations

	Delays (µs)	Size (octet)
gettimeofday()	1.049	8
do_gettimeofday()	0.195	8
event JVMPI	3	60
Writing « socket »	2.13	36 + 32
Reading « socket »	1.55	36 + 32

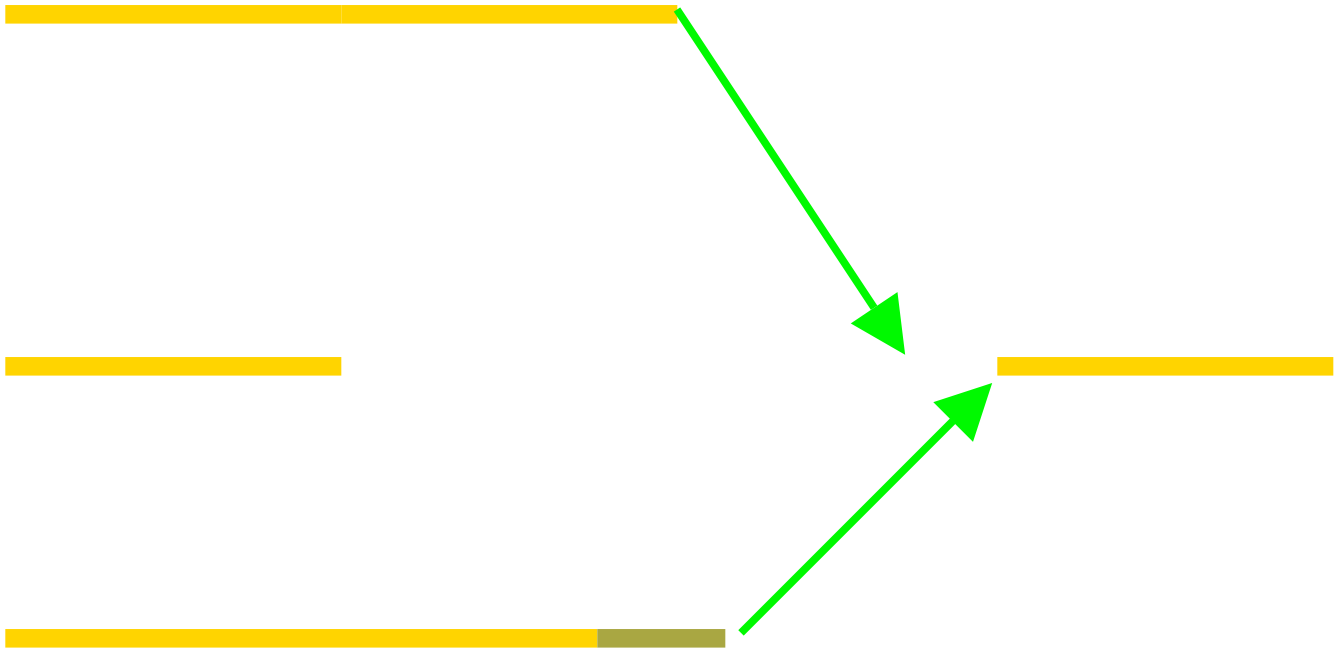
Perturbations application level : **≈ 6 %**
Perturbations OS level : **≈ 3 %**

Correction post-mortem

Estimation of recording time
Perturbation accumulation
Possible transfert



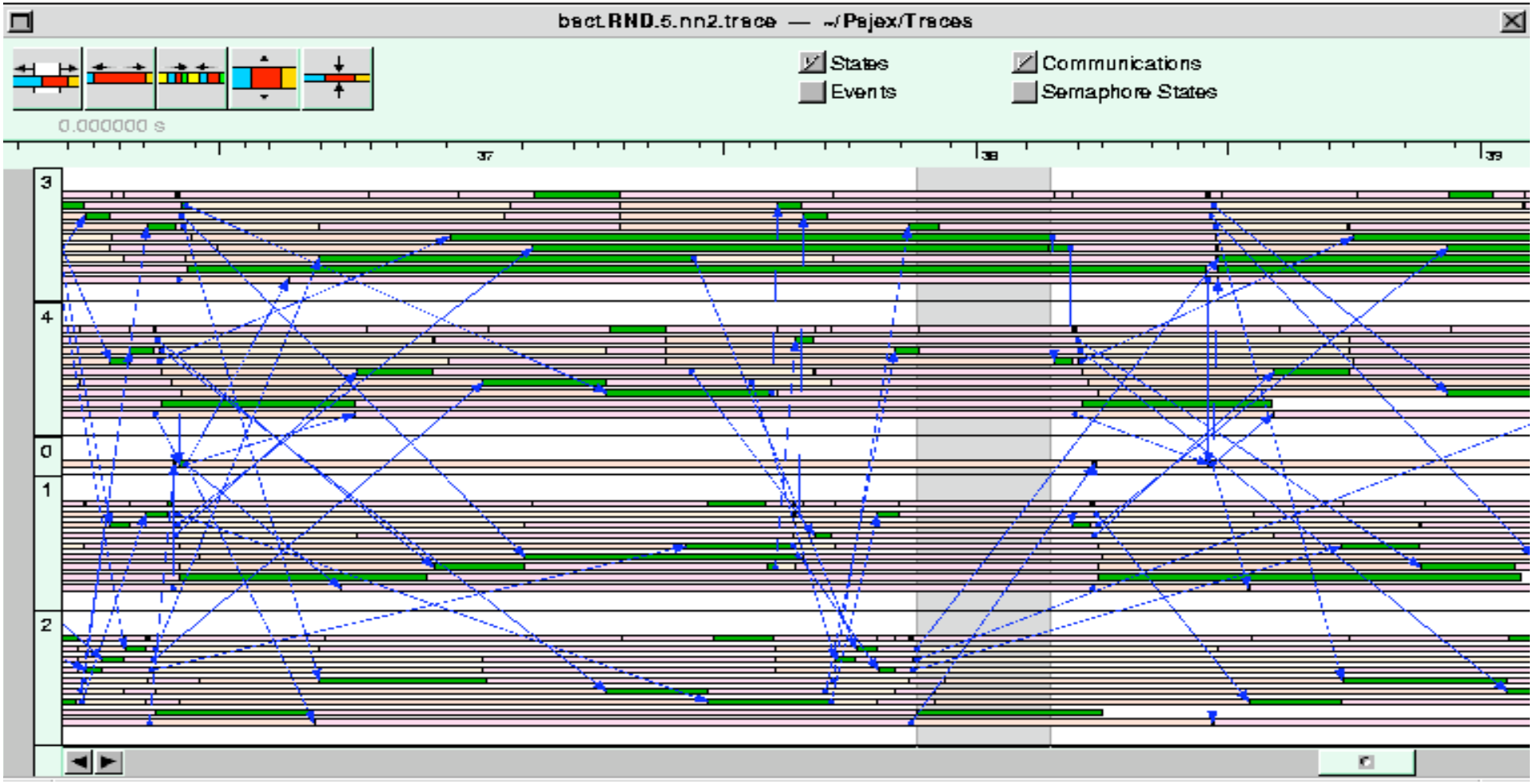
Race conditions



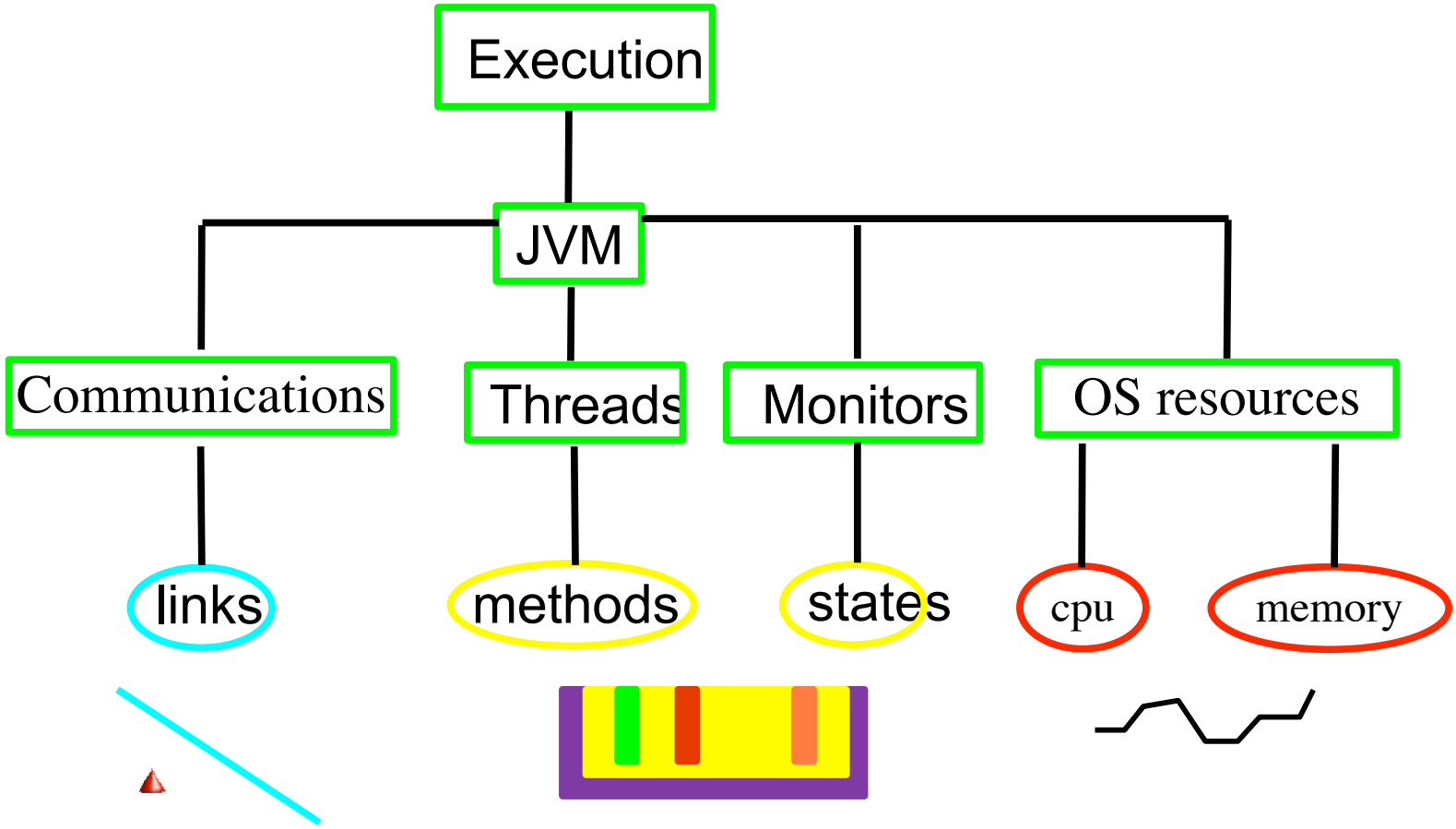
Conservative policy

Visualisation « Paje »

[J. Chassin, B. Stein]



Hierarchy of visual objects in « Pajé »

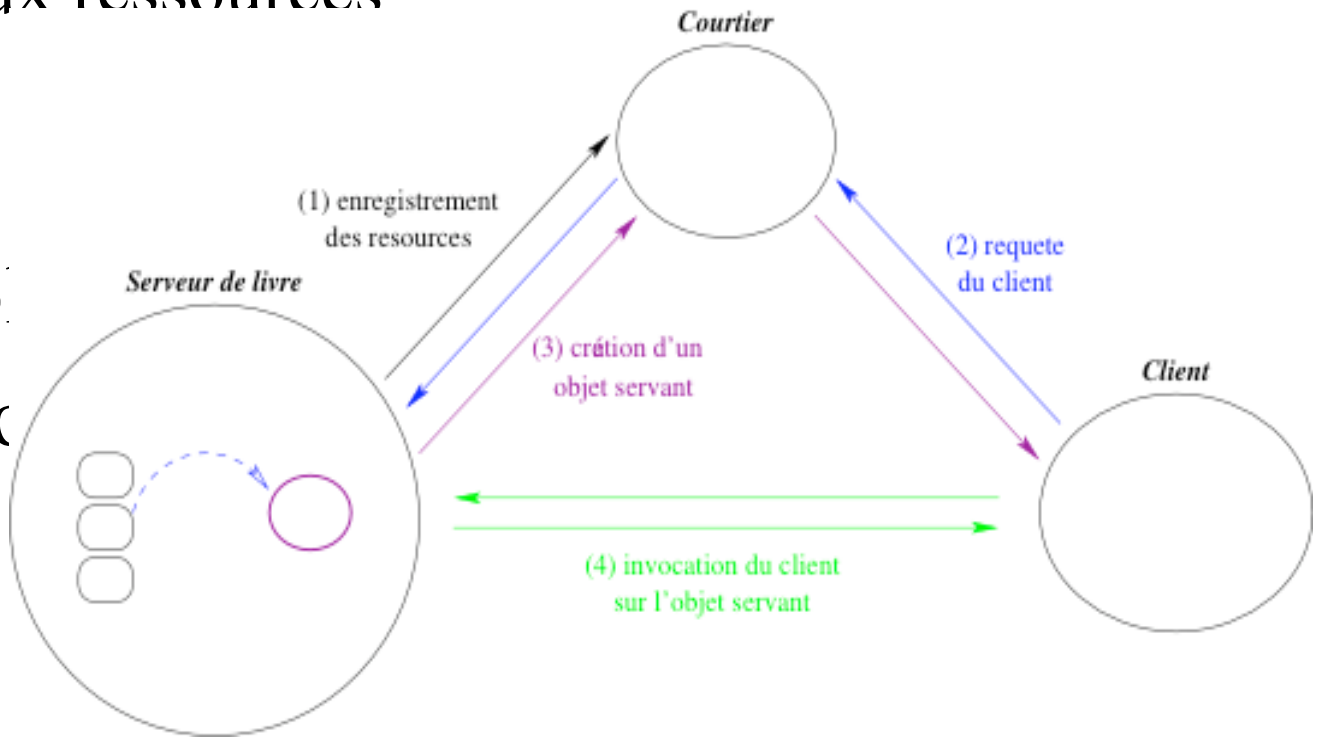


Traçage événementiel pour l'analyse d'exécutions réparties : de la théorie à la pratique

Application au serveur multimédia

- Serveur : livre, musique
- Courtier : enregistrement des ressources et requêtes
- Client : accès aux ressources

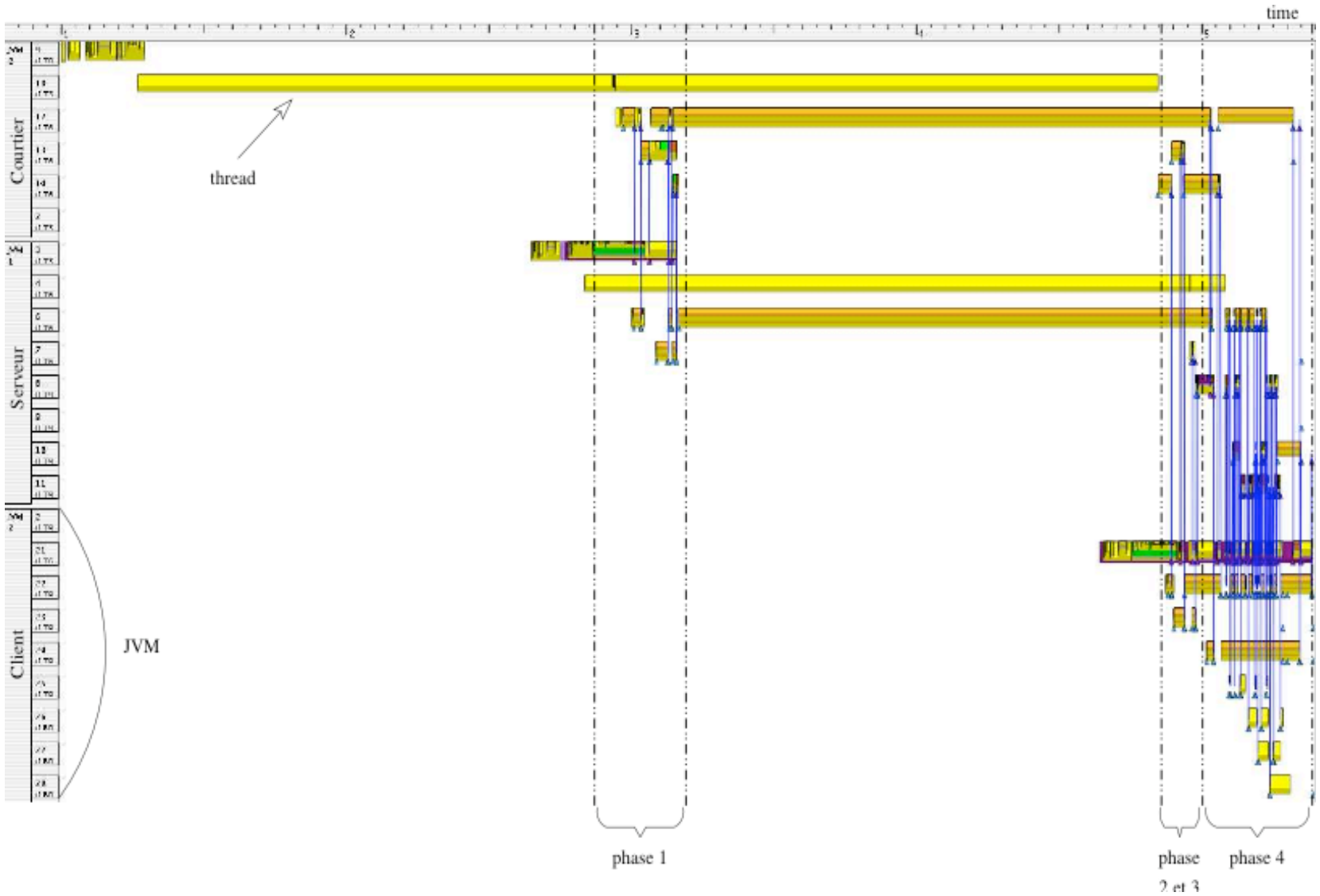
- Modèle d'app
- Modèle d'invo



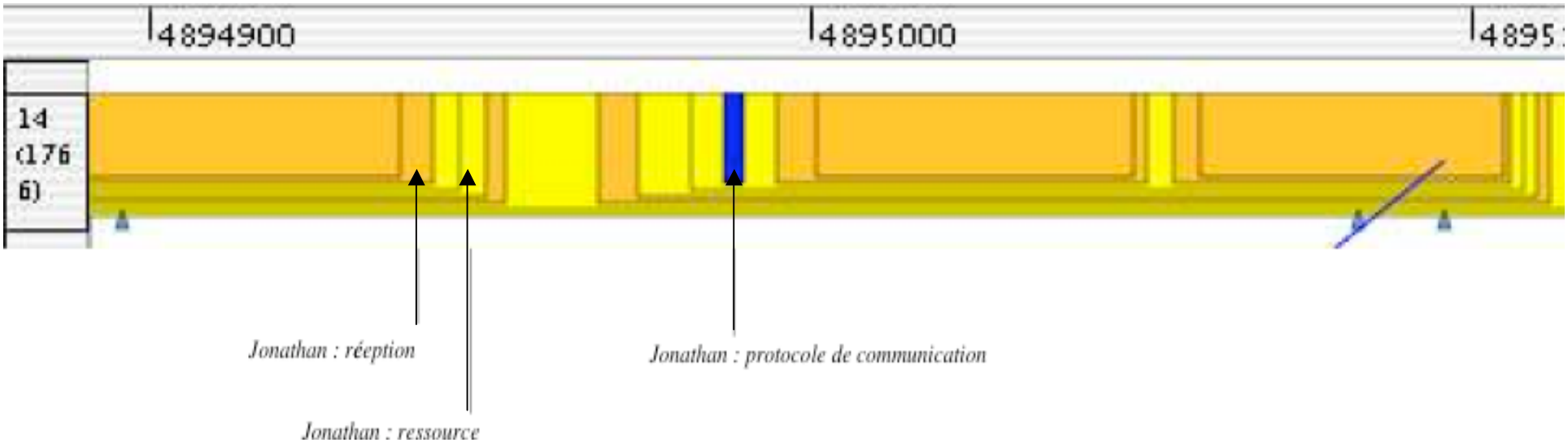
Problèmes à résoudre

- Exécution locale :
  - causalité des appels de méthode
  - utilisation des objets de synchronisation
  - gestion des ressources d'exécution : thread, GC…
- Exécution répartie :
  - interaction distante : communications
  - blocage : du retour d'une invocation
- Performances :
  - utilisation des ressources systèmes : thread, CPU, mémoire
  - utilisation des ressources d'infrastructure : ORB, JVM

Interprétation d'une trace

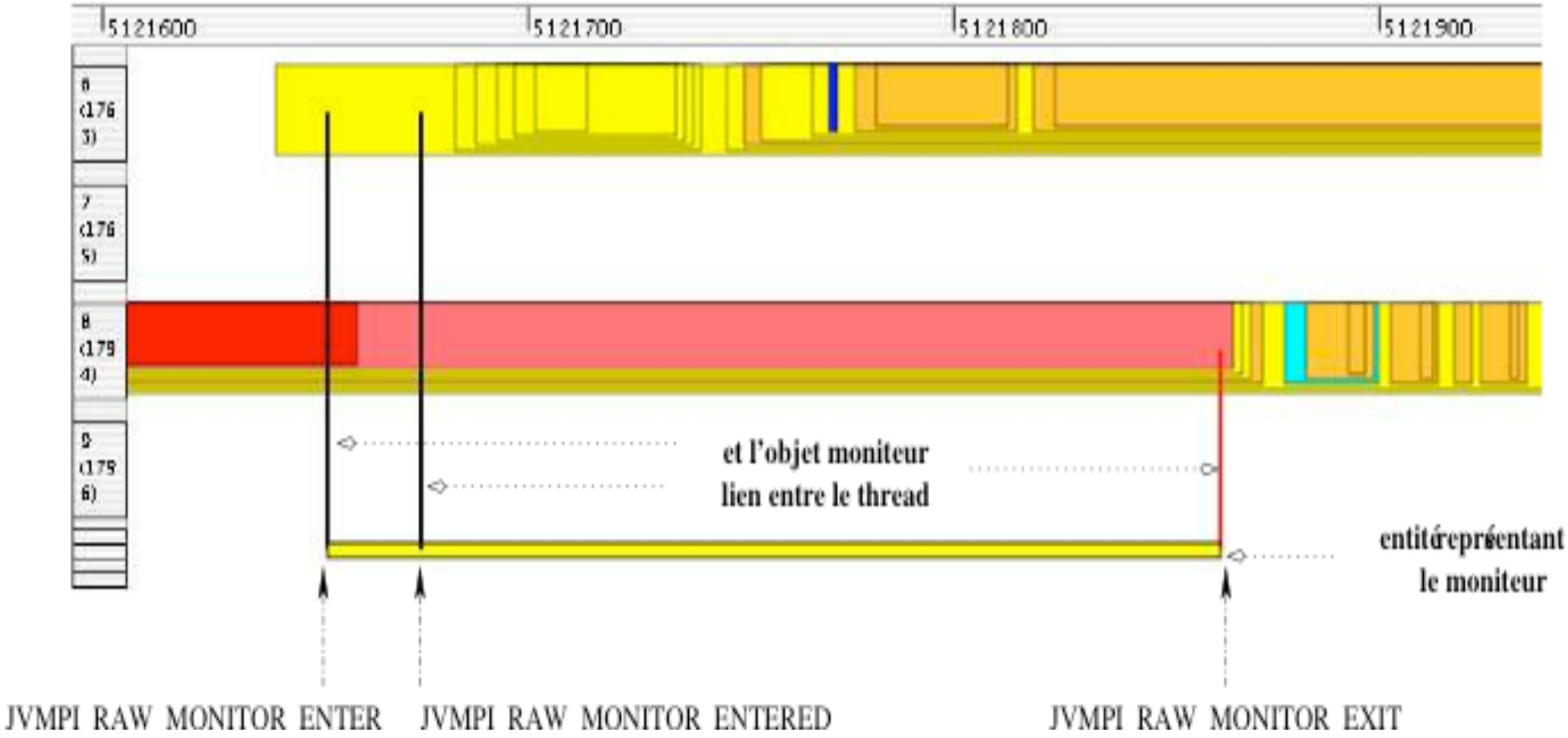


Causalité des appels de méthode

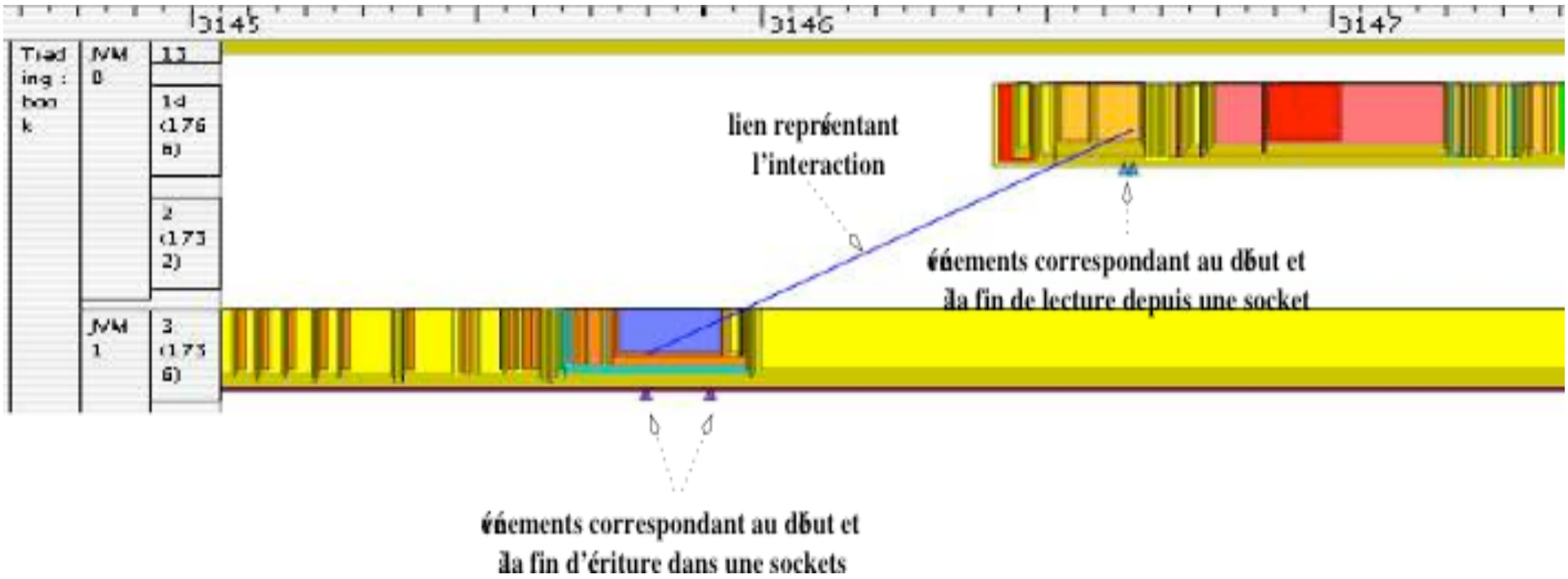


- Thread : exécution séquentielle
- Emboîtement = causalité
- Enchainement = séquentialité

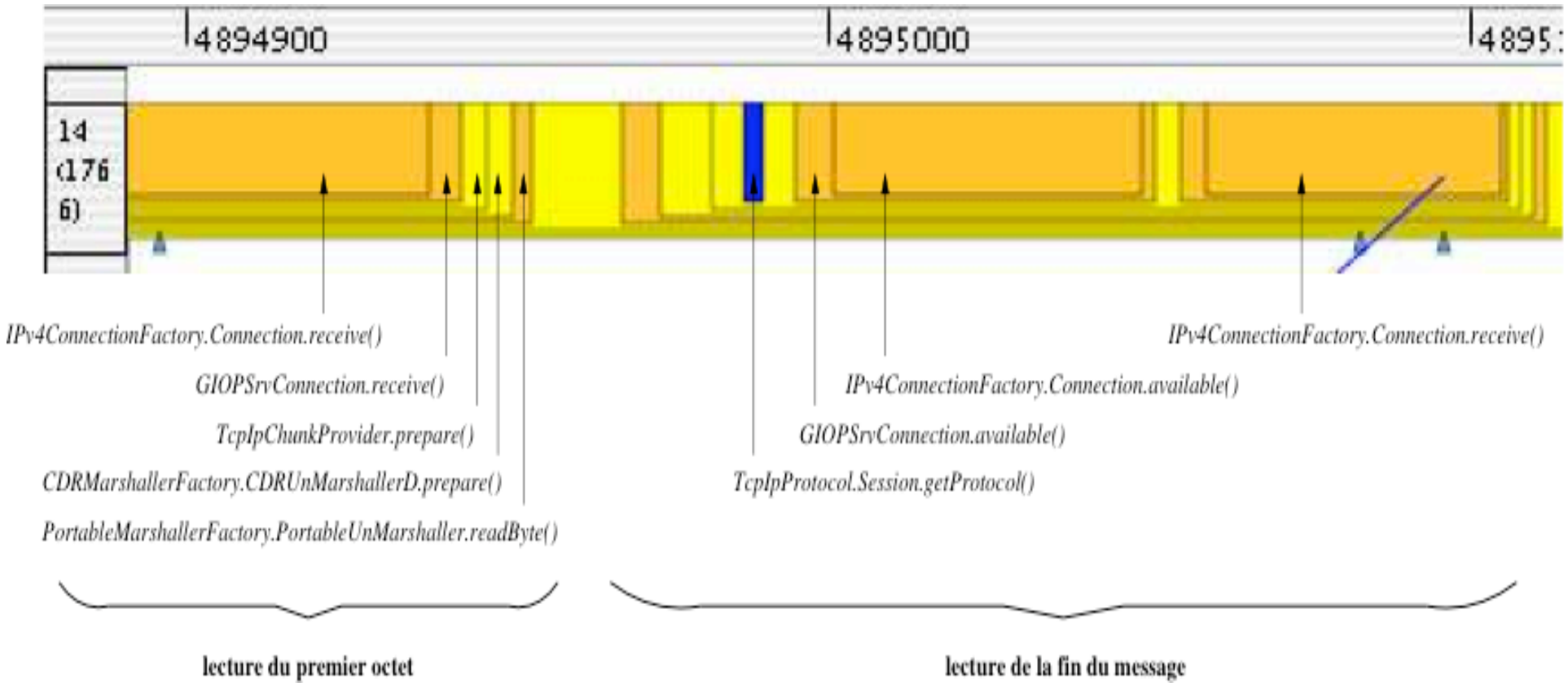
Objet de synchronisation



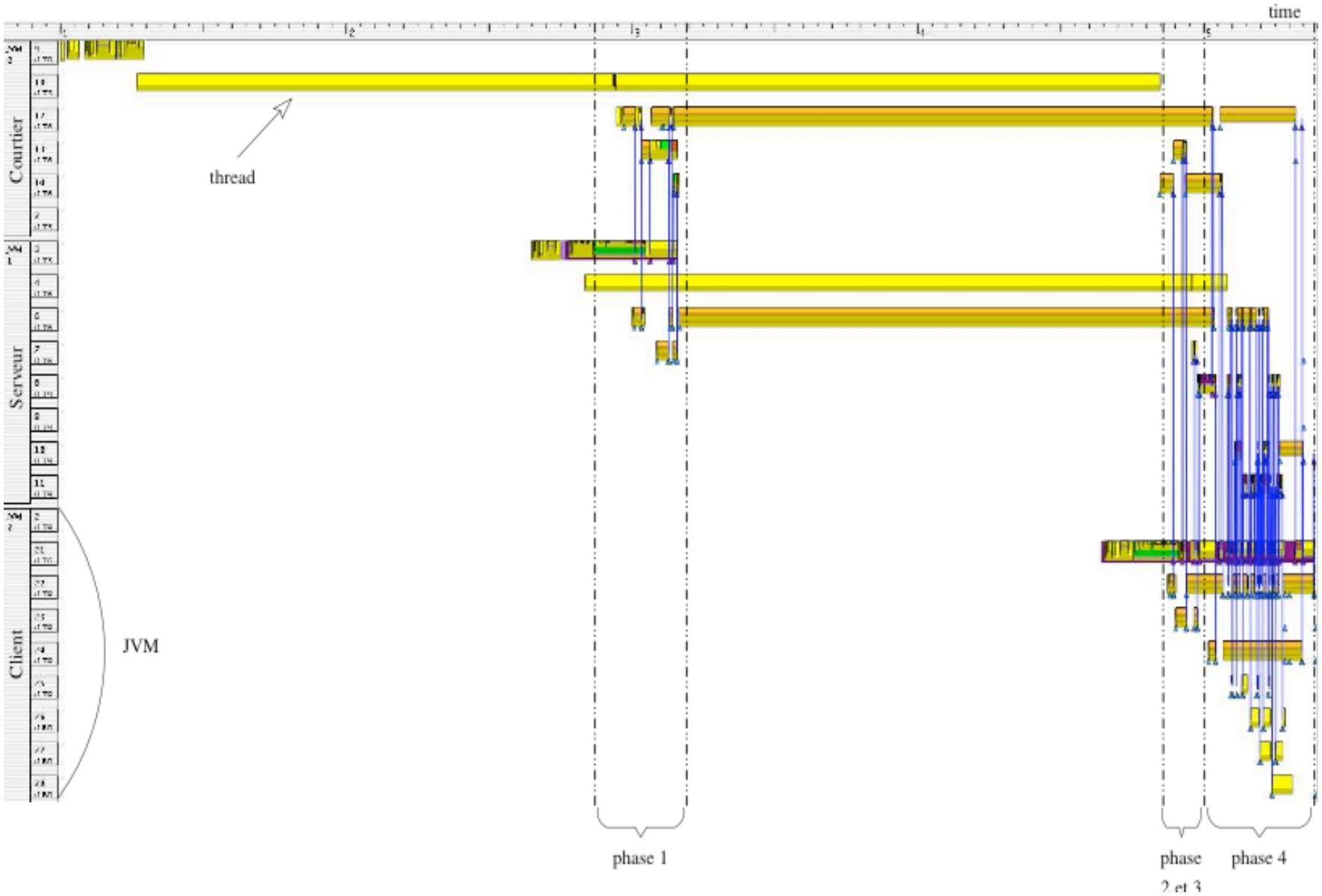
Interaction distante



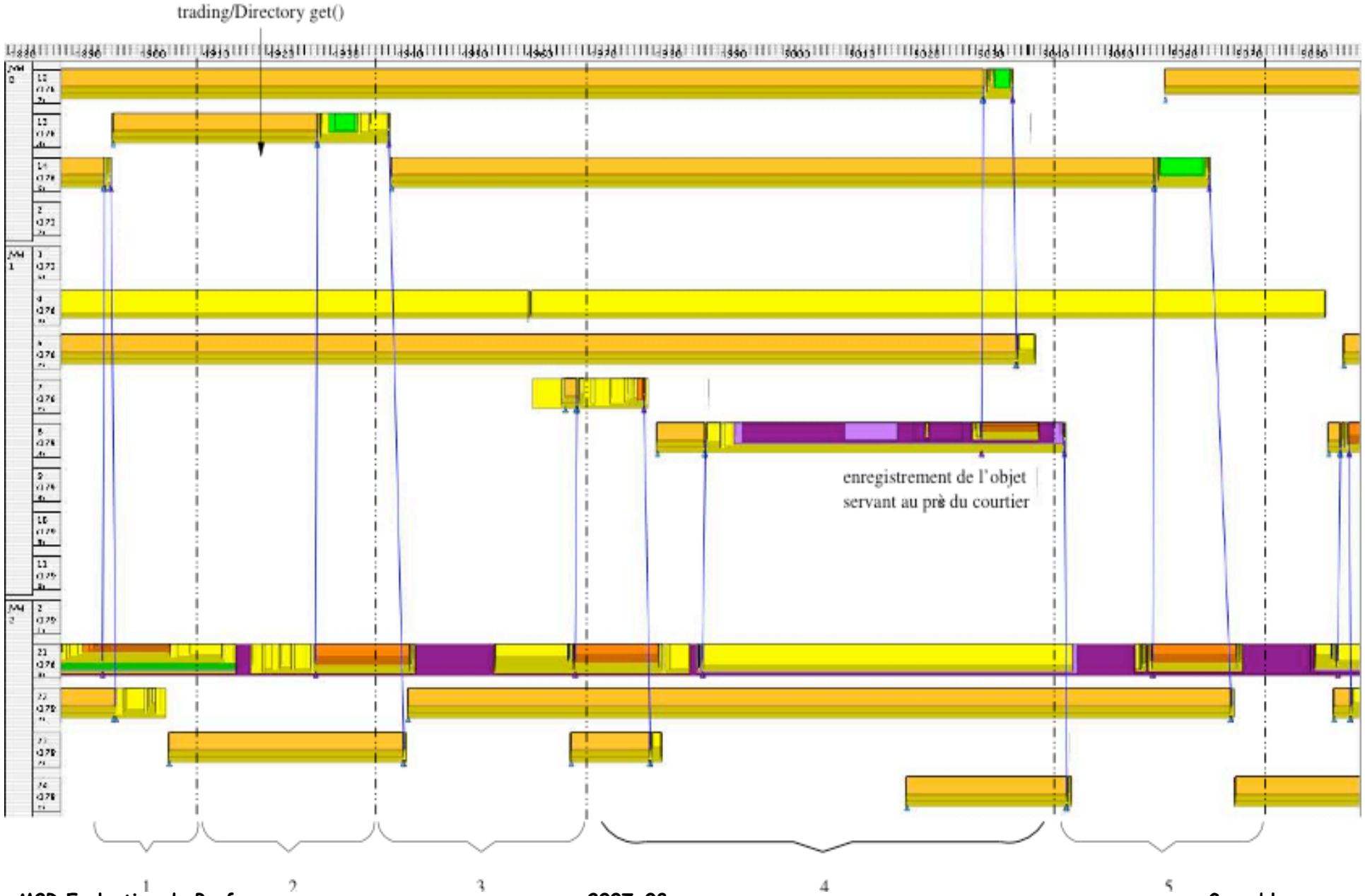
Pattern de réception



Vue globale


time
Courtier
Serveur
Client
thread
JVM
phase 1
phase 2 et 3
phase 4

Interprétation de la phase 2



Utilisation des ressources systèmes

Approche d'analyse proposée

Phase d'étude	Identification des besoins Identification du modèle de performances à étudier
Phase d'instrumentation	Instrumentation du programme à observer
Phase d'observation	Exécution du programme et observation des évènements (traçage des évènements)
Phase de traitement	Fusions, correction, compression des traces
Phase de présentation et d'analyse	Visualisation et analyse des traces collectées

État de l'art et choix technologiques

Phase d'étude

- **Qu'est ce qu'on veux étudier?**
  ->Les threads et les objets de synchronisation Java
- **Quel est le modèle de performances à étudier?**
  ->Le modèle de programmation concurrente

Phase d'instrumentation

- **Portable/non portable**
- **Code source/code binaire**
- **Automatique/manuelle/semi-automatique**

Phase d'observation

- **Traçage/échantillonnage/chronométrage/comptage**
- **Capture évènementielle/à la demande/périodique**

Phase de traitement

- **Filtrage à la volé + filtrage post-mortem**

Phase de présentation et d'analyse

- **Pajé**

Plateforme proposée

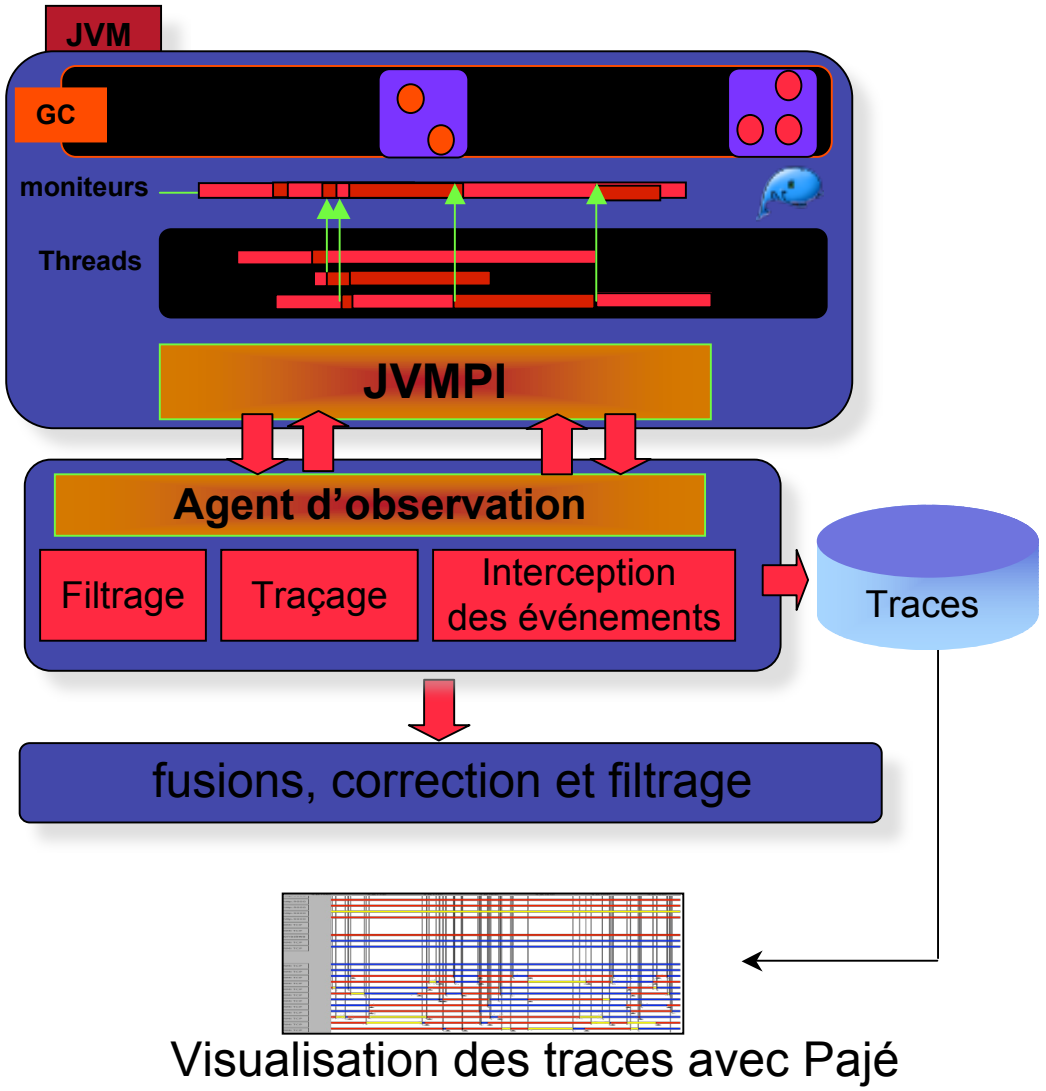
Phase d'étude

Phase d'instrumentation

Phase d'observation

Phase de traitement

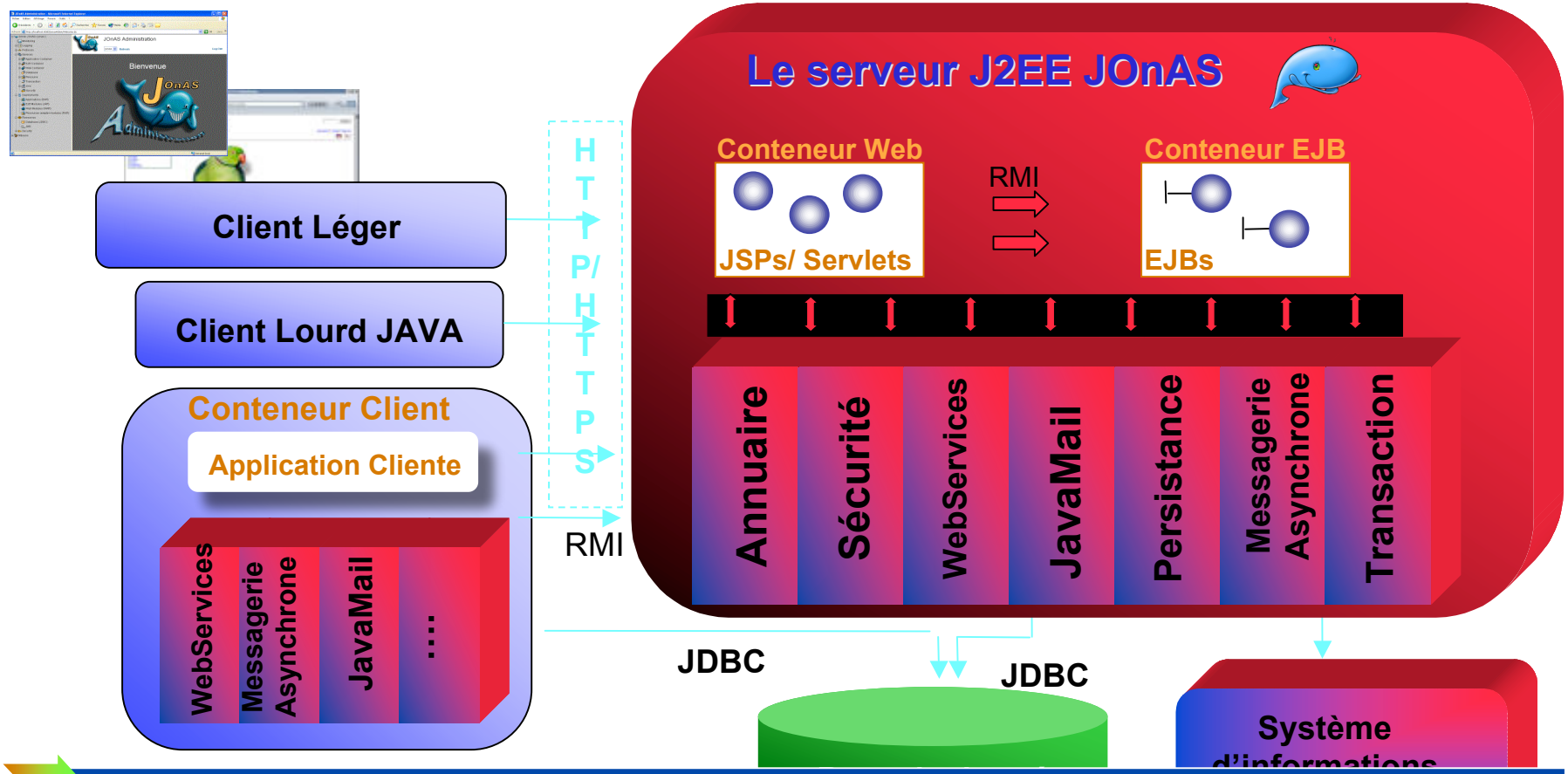
Phase de présentation et d'analyse



Visualisation des traces avec Pajé

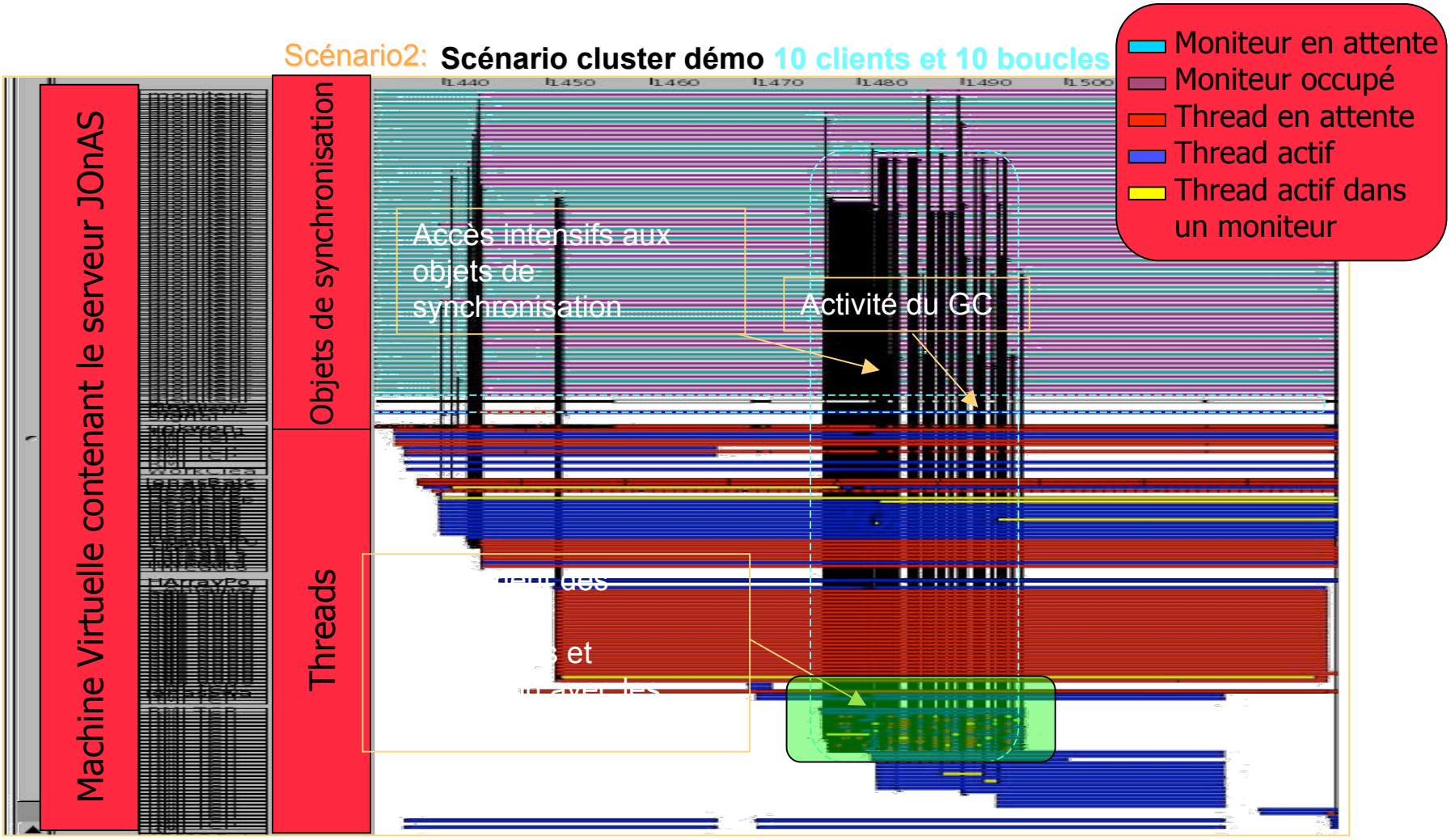
Le serveur J2EE JOnAS (2)

Architecture

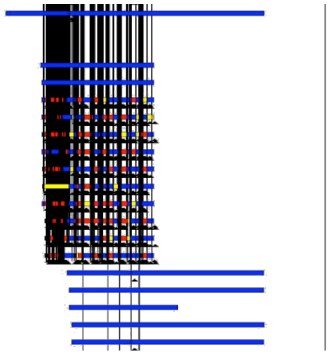


JOnAS est l'exemple type d'un programme complexe très intéressant à étudier dans le cadre du test logiciel et l'analyse des comportements et des performances

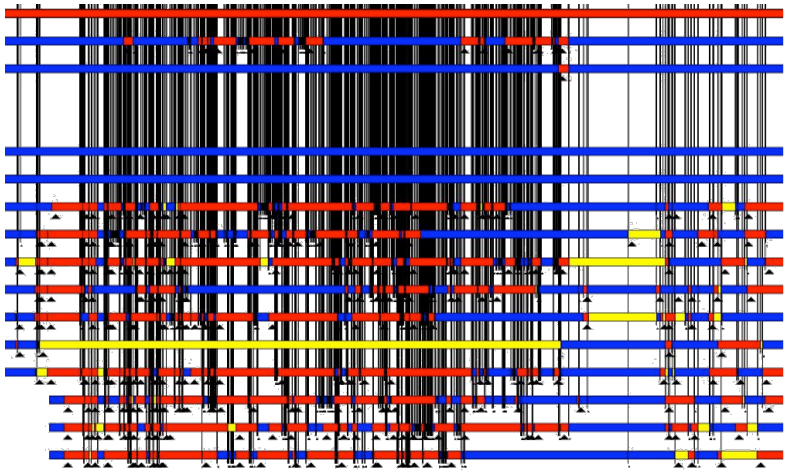
Exemple de visualisation des threads JOnAS



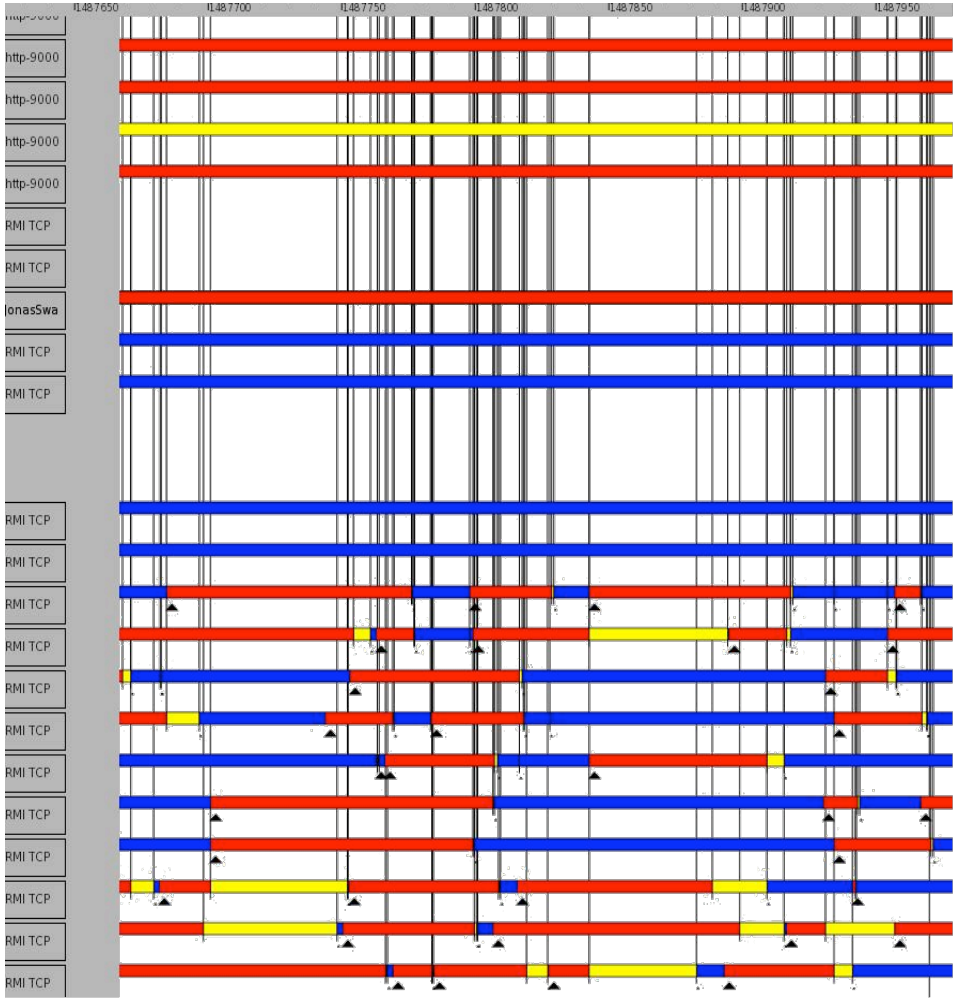
Agrandissement des visualisations



Visualisation initiale



Premier niveau d'agrandissement

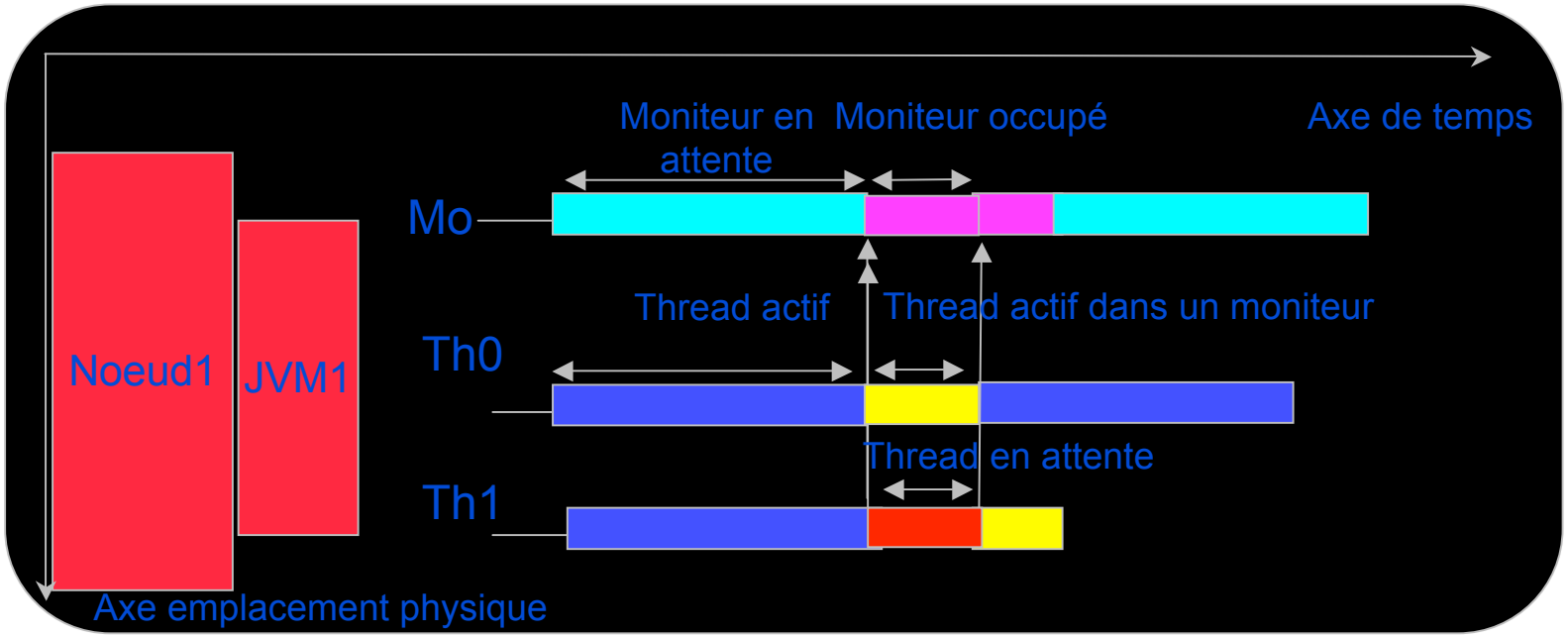


Deuxième niveau d'agrandissement

Interprétations des visualisations

Pajé permet la visualisation :

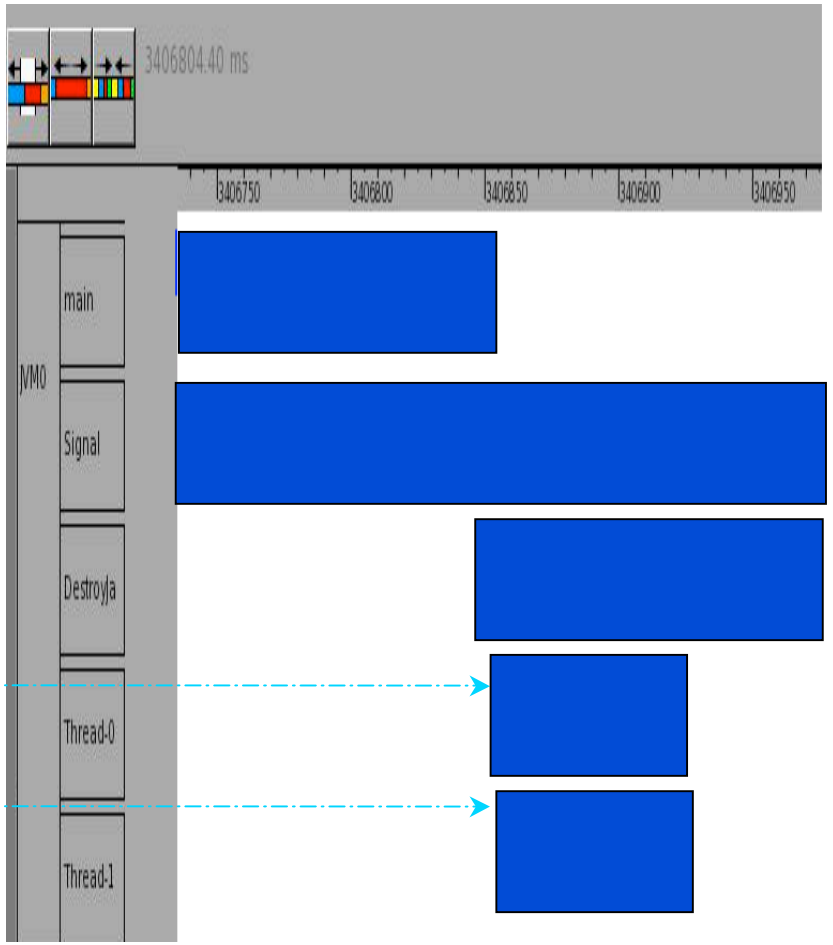
- des threads dans un diagramme spatio-temporel
- des objets de synchronisation
- de l'interaction entre threads et objets de synchronisation



Correspondance entre code Java et visualisation Pajé

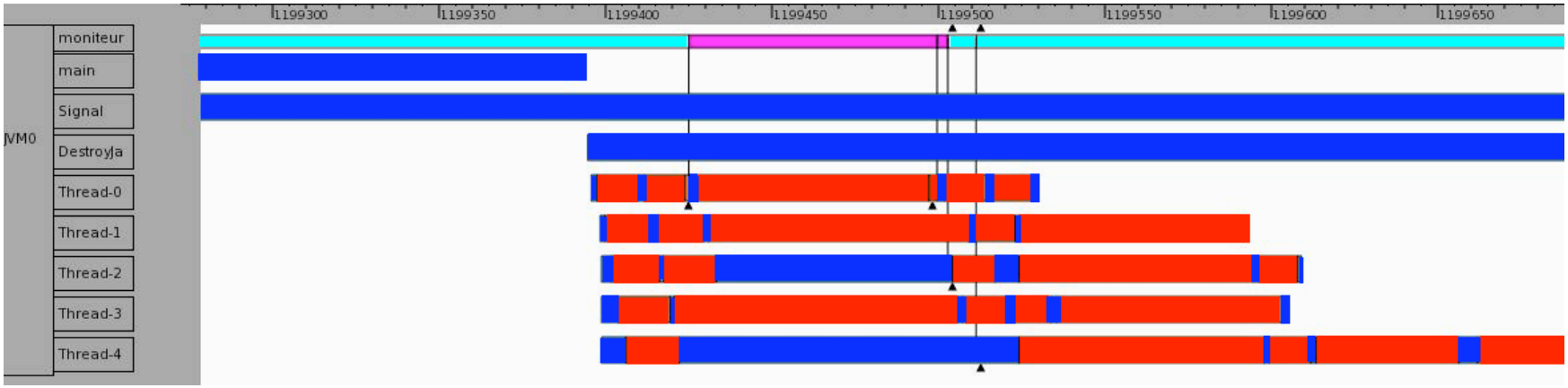
```
public class Demo1 extends Thread {
        public int x;
        public Demo1(int param){
                x=param;
                }
    public synchronized void Task1(){…}
   public void run(){
        for (int i=0;i<5;i++){
            Task1();}
  }

   public static void main(String[] args) {
        Demo1 Thread0=new Demo1(1);
        Thread0.start();
        Demo1 Thread1=new Demo1(2);
        Thread1.start();
   }
}
```

Code Java

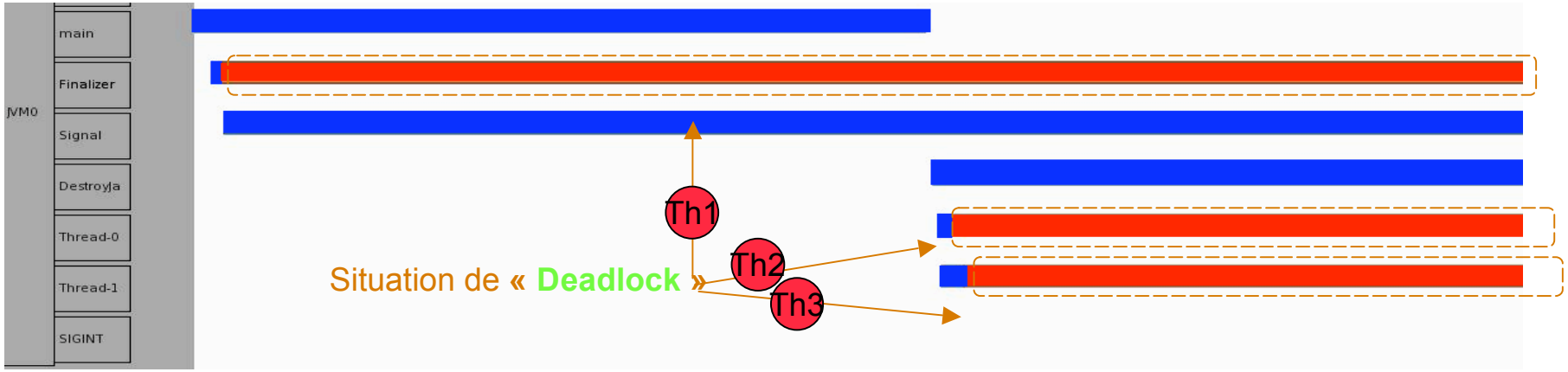


Environnement de visualisation Pajé

Exemples de situations possibles



Entrelacement des threads



Inter blocage

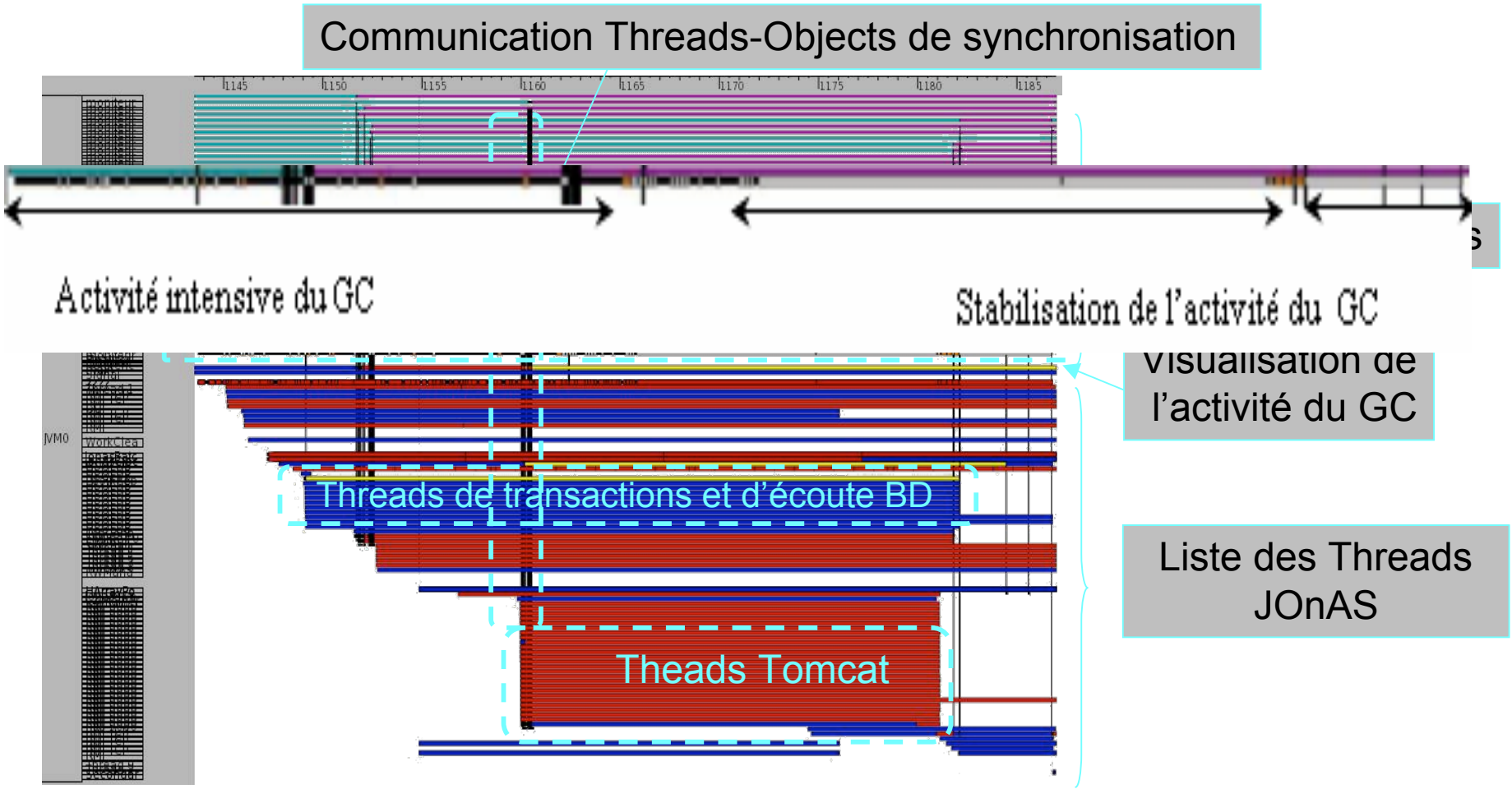
Évaluation de la plateforme

Taux de perturbation:

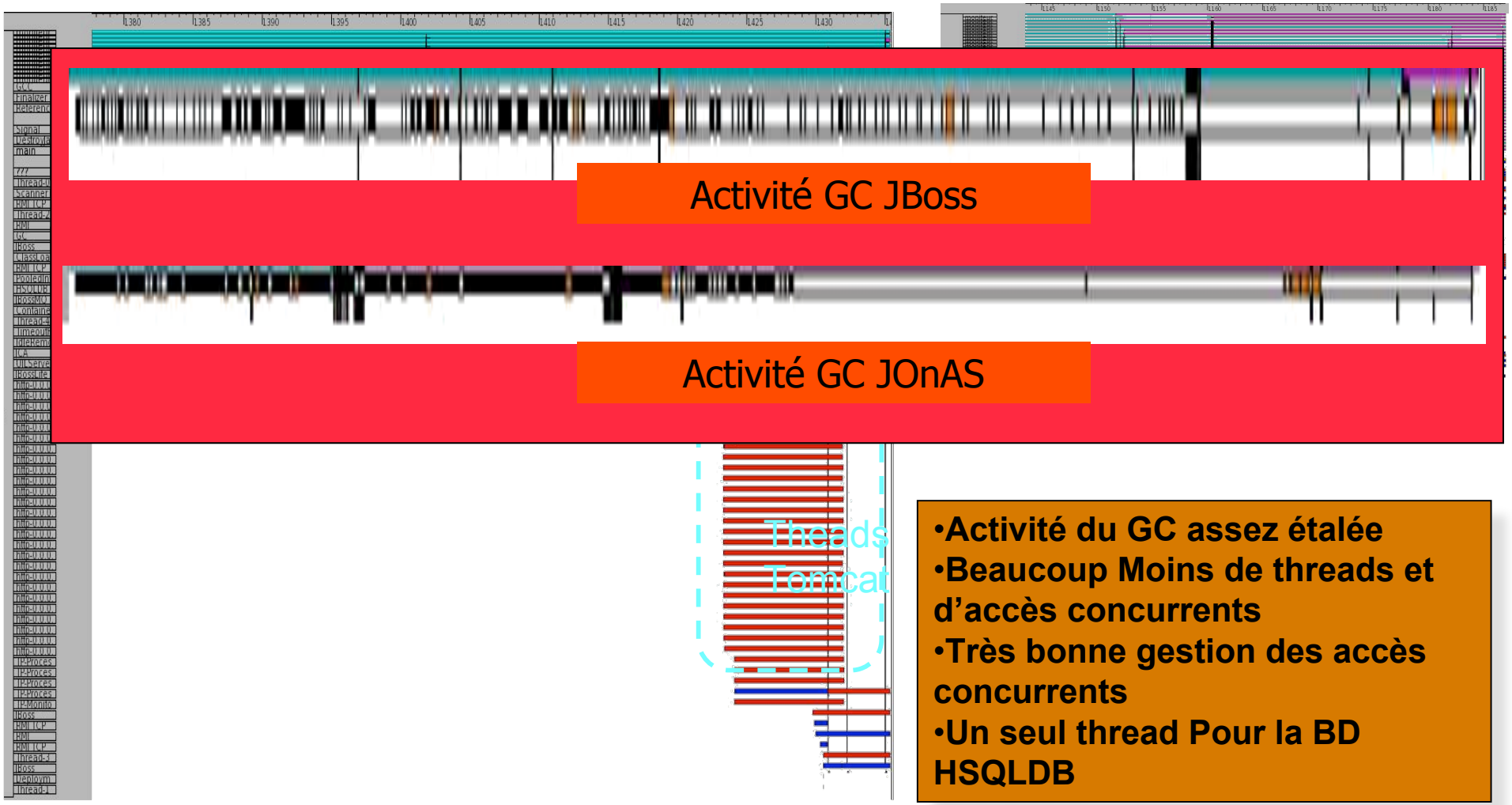
« Le surcoût du à l'instrumentation du programme »

	Nbr évènements	Temps d'exécution sans instrumentation (Milliseconde)	Temps d'exécution avec instrumentation (Milliseconde)	Taux de perturbation
Démarrage et arrêt de JOnAS	2081	16.341	29.763	**82.112%**
Scénario src	4008	27.57	32.778	**18.89%**
Scénario Cluster Démo (10 Clients/10 boucles)	26253	81.311	151	**85,7%**
Scénario Cluster Démo (10 Clients/100 boucles)	81077	462.677	562.92	**17.80 %**

Analyse de l'activité du GC de JOnAS



Comparaison avec le serveur JBoss



- Activité du GC assez étalée
- Beaucoup Moins de threads et d'accès concurrents
- Très bonne gestion des accès concurrents
- Un seul thread Pour la BD HSQLDB

Scénario démarrage du serveur JBoss

Perspectives

Observation : Définition d'un modèle état/événements **MULTI-NIVEAUX**

Observation = perturbation ←→compromis

Challenges :

- facteurs d'échelle : clusters, grids, grid5000;
- niveaux hétérogènes pour l'instrumentation
- hétérogénéité des environnements

Analyse de données et visualisation

- recherche de motifs automatique, abstraction automatique, modèles de programmation…


Oxford virtual museum: The Measurers

Sonorisation

- Possibilités du son :
  - traitement passif : repérage d ’anomalie, schéma répétitif, etc.
  - combinaison de sons (traitement parallèle par exemple).
  - Plusieurs dimensions utilisables : timbre, hauteur, durée, intensité, positionnement spatial.
- Représentation du comportement temporel d ’un programme : communications, charge des processeurs, etc.
- Exemple