ProActive Parallel Suite

Denis Caromel Arnaud Contes et al.

Univ. Nice/INRIA/CNRS ActiveEon

January 2009

Agenda

- ProActive and ProActive Parallel Suite
- Programming and Composing
  - ProActive Core
  - High Level Programming models
  - ProActive Components
- Deployment Framework
- Development Tools

ProActive

- ProActive is a JAVA middleware for parallel, distributed and multi-threaded computing.

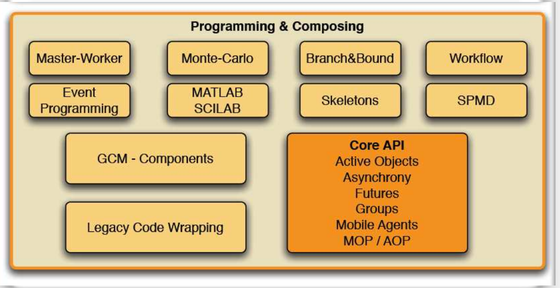
- ProActive features:
  - ☐ A programming model
  - ☐ A comprehensive framework

To simplify the programming and execution of parallel applications within multi-core processors, distributed on Local Area Network (LAN), on clusters and data centers, on intranet and Internet Grids.

Current Open Sour Tools:

PROGRAMMING

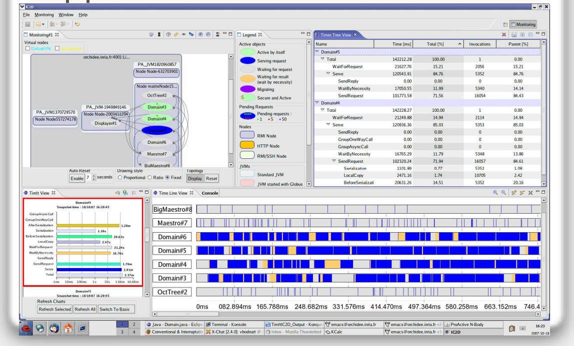
Java Parallel Frameworks
for HPC, Multi-Cores, Distribution, Enterprise Grids and Clouds.


Programming & Composing

- Master-Worker
- Monte-Carlo
- Branch&Bound
- Workflow
- Event Programming
- MATLAB SCILAB
- Skeletons
- SPMD
- GCM - Components
- Legacy Code Wrapping
- Core API
  - Active Objects
  - Asynchrony
  - Futures
  - Groups
  - Mobile Agents
  - MOP / AOP

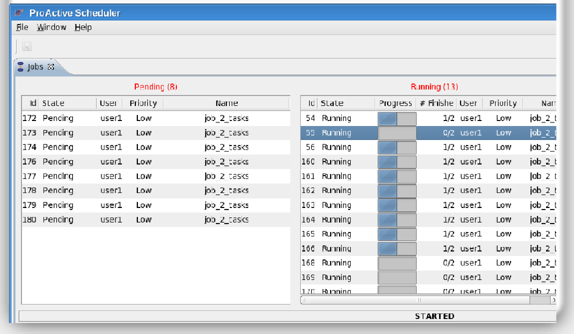
OPTIMIZING

Eclipse GUI (IC2D)
for Developing, Debugging, Optimizing your parallel applications.



SCHEDULING

Multi-Language Scheduler
for Workflows made of C, C++, Java, Scripts, Matlab, Scilab tasks.



Pending (8)

Id	State	User	Priority	Name
172	Pending	user1	Low	job_2_tasks
173	Pending	user1	Low	job_2_tasks
174	Pending	user1	Low	job_2_tasks
176	Pending	user1	Low	job_2_tasks
177	Pending	user1	Low	job_2_tasks
178	Pending	user1	Low	job_2_tasks
179	Pending	user1	Low	job_2_tasks
180	Pending	user1	Low	job_2_tasks

Running (13)

Id	State	Progress	# Finishe	User	Priority	Nar
54	Running		1/2	user1	Low	job_2_t
55	Running		0/2	user1	Low	job_2_t
56	Running		1/2	user1	Low	job_2_t
160	Running		1/2	user1	Low	job_2_t
161	Running		1/2	user1	Low	job_2_t
162	Running		1/2	user1	Low	job_2_t
163	Running		1/2	user1	Low	job_2_t
164	Running		1/2	user1	Low	job_2_t
165	Running		1/2	user1	Low	job_2_t
166	Running		1/2	user1	Low	job_2_t
168	Running		0/2	user1	Low	job_2_t
169	Running		0/2	user1	Low	job_2_t
170	Running		0/2	user1	Low	job_2_t

STARTED

Acceleration Toolkit :
Concurrency+Parallelism +Distributed

Unification of Multi-Threading and Multi-Processing

Multi-Threading

Multi-Core Programming

- **SMP**
  - Symmetric Multi-Processing
  - Shared-Memory Parallelism
- Solutions : OpenMP, pThreads, Java Threads...

Multi-Processing

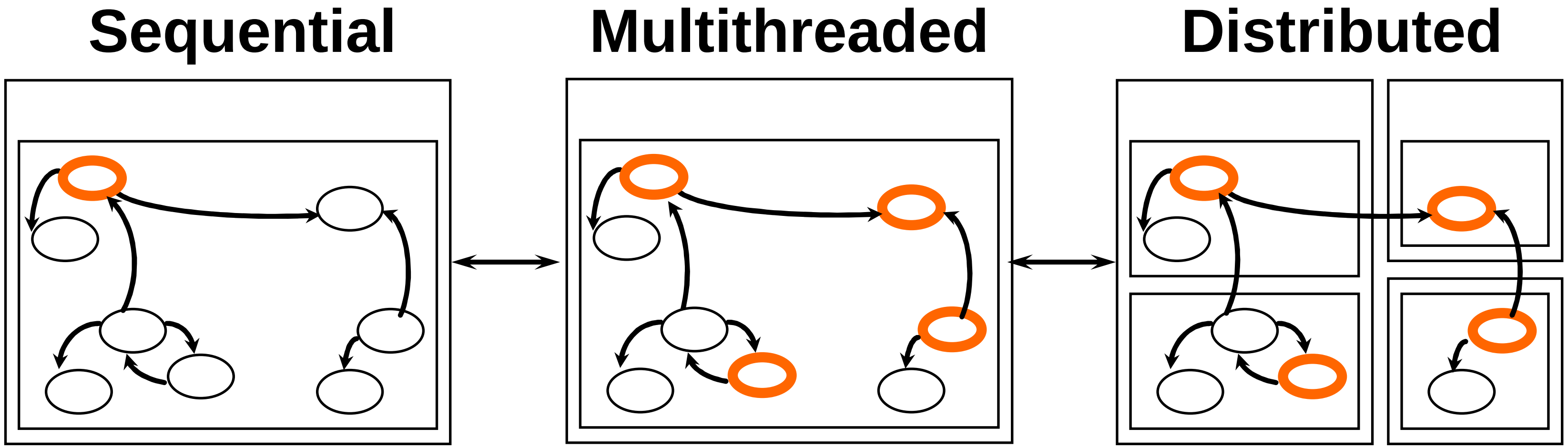
Distributed programming,
Grid Computing

- **MPP**
  - Massively Parallel Programming or
  - Message Passing Parallelism
- Solutions: PVM, MPI, RMI, sockets ,…

ProActive
Parallel Suite

Unification of Multi-threading and Multi-processing

Seamless

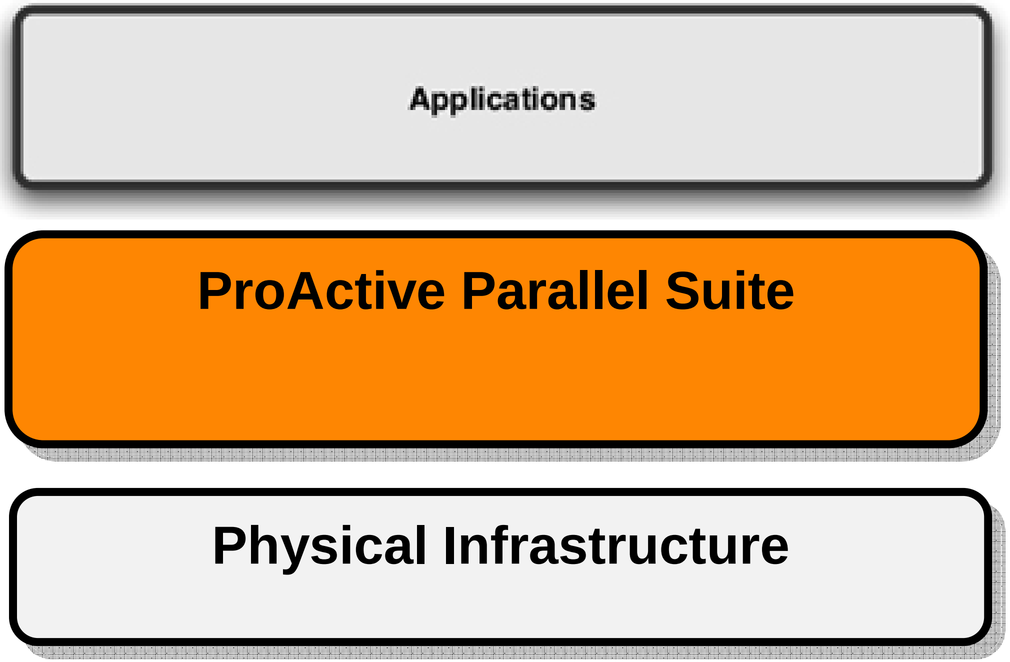


- Most of the time, activities and distribution are not known at the beginning, and change over time
- Seamless implies reuse, smooth and incremental transitions

ProActive Parallel Suite

- ProActive Parallel Suite includes:
  - The ProActive middleware featuring services like:
    - Fault tolerance, Load balancing, Distributed GC, Security, WS
    - A set of parallel programming frameworks
    - A framework for deploying applications on distributed infrastructures
  - Software for scheduling applications and resource management
  - Software for monitoring and profiling of distributed applications
  - Online documentation
  - Full set of demos and examples

ProActive Parallel Suite

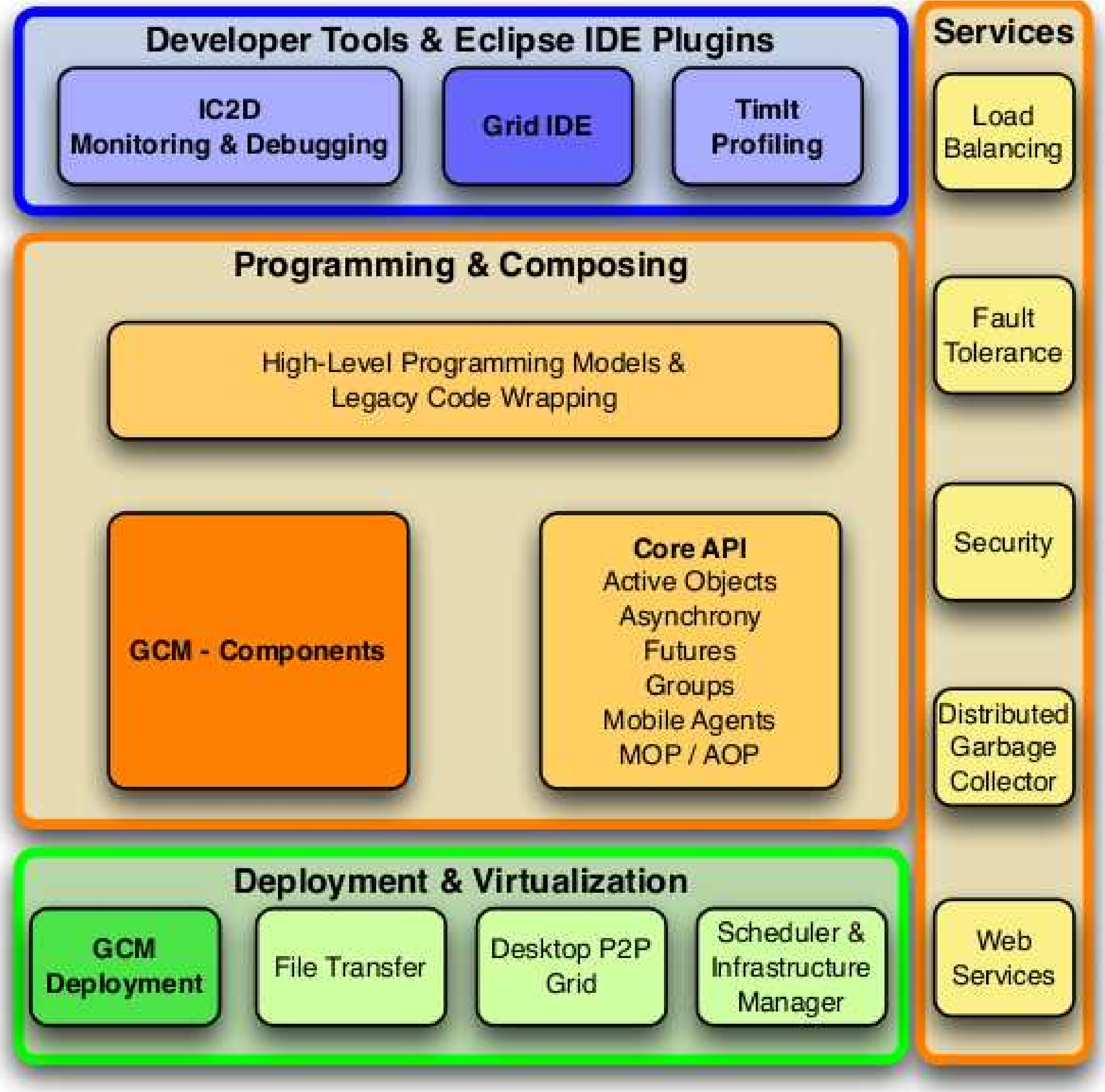


Applications

ProActive Parallel Suite

Physical Infrastructure

ProActive Parallel Suite



Developer Tools & Eclipse IDE Plugins

IC2D Monitoring & Debugging

Grid IDE

TimIt Profiling

Programming & Composing

High-Level Programming Models & Legacy Code Wrapping

GCM - Components

Core API
Active Objects
Asynchrony
Futures
Groups
Mobile Agents
MOP / AOP

Deployment & Virtualization

GCM Deployment

File Transfer

Desktop P2P Grid

Scheduler & Infrastructure Manager

Services

Load Balancing

Fault Tolerance

Security

Distributed Garbage Collector

Web Services

Ways of using Proactive Parallel Suite?

- To easily develop parallel/distributed applications from scratch

- Develop applications using well-known programming paradigms thanks to our high-level programming frameworks (master-worker, Branch&Bound, SPMD, Skeletons)

- To transform your sequential mono-threaded application into a multi-threaded one (with minimum modification of code) and distribute it over the infrastructure.

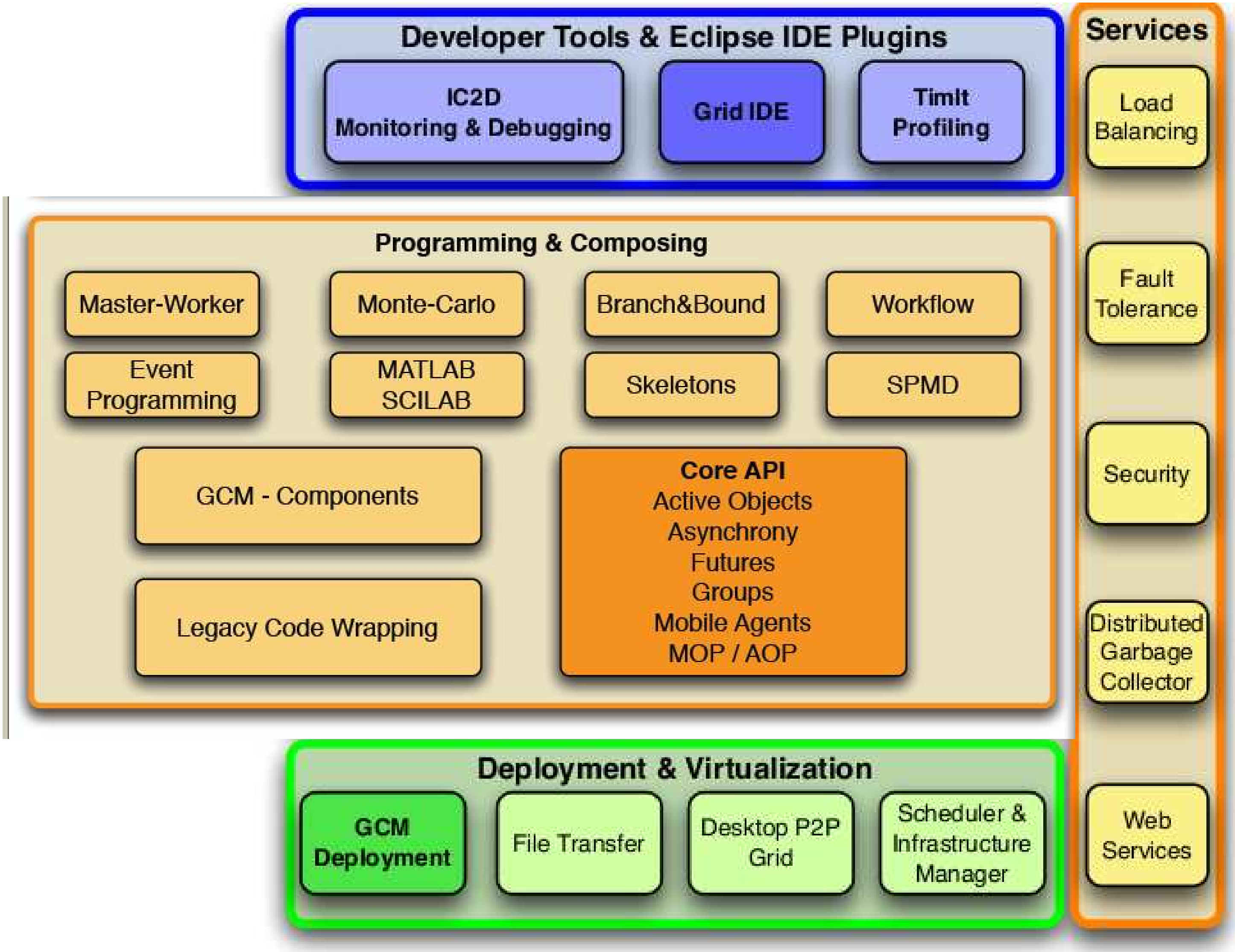
Ways of using Proactive Parallel Suite?

- To wrap your native application with ProActive in order to distribute it
- Define jobs containing your native-applications and use ProActive to schedule them on the infrastructure

Agenda

- ProActive and ProActive Parallel Suite
- Programming and Composing
  - ProActive Core
  - High Level Programming models
  - ProActive Components
- Deployment Framework
- Development Tools

ProActive Parallel Suite



Developer Tools & Eclipse IDE Plugins

IC2D Monitoring & Debugging

Grid IDE

TimIt Profiling

Programming & Composing

Master-Worker

Monte-Carlo

Branch&Bound

Workflow

Event Programming

MATLAB SCILAB

Skeletons

SPMD

GCM - Components

Legacy Code Wrapping

Core API

Active Objects

Asynchrony

Futures

Groups

Mobile Agents

MOP / AOP

Deployment & Virtualization

GCM Deployment

File Transfer

Desktop P2P Grid

Scheduler & Infrastructure Manager

Services

Load Balancing

Fault Tolerance

Security

Distributed Garbage Collector

Web Services

ProActive Core

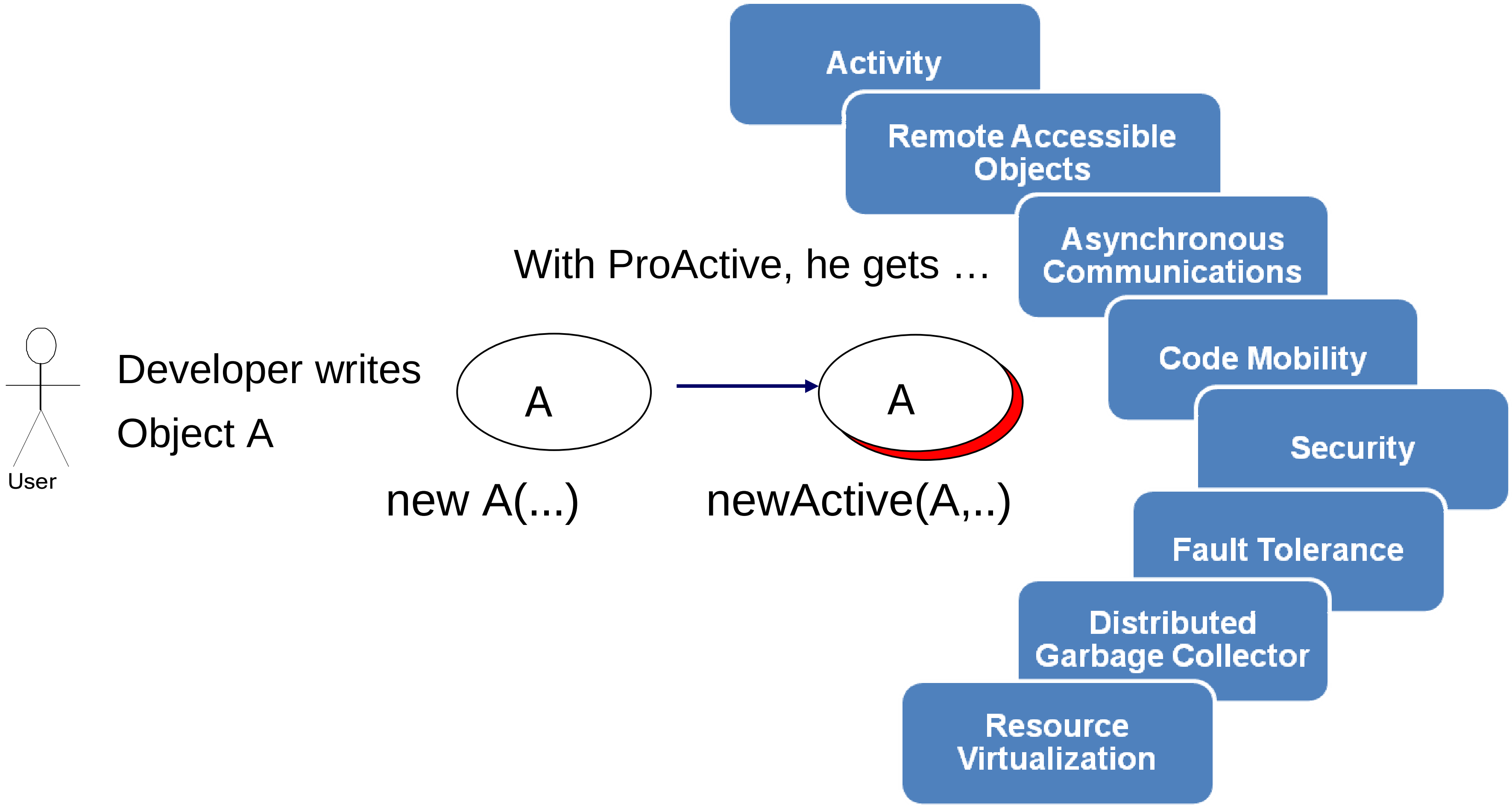
ACTIVE OBJECTS

ProActive

A 100% Java API + Tools for Parallel, Distributed Computing

- A programming model: Active Objects
  - ☐ Asynchronous Communications, Wait-By-Necessity, Groups, Mobility, Components, Security, Fault-Tolerance
- A formal model behind: Determinism (POPL’04)
  - ☐ Insensitive to application deployment
- A uniform Resource framework
  - ☐ Resource Virtualization to simplify the programming

Active Objects



With ProActive, he gets …

Developer writes

Object A

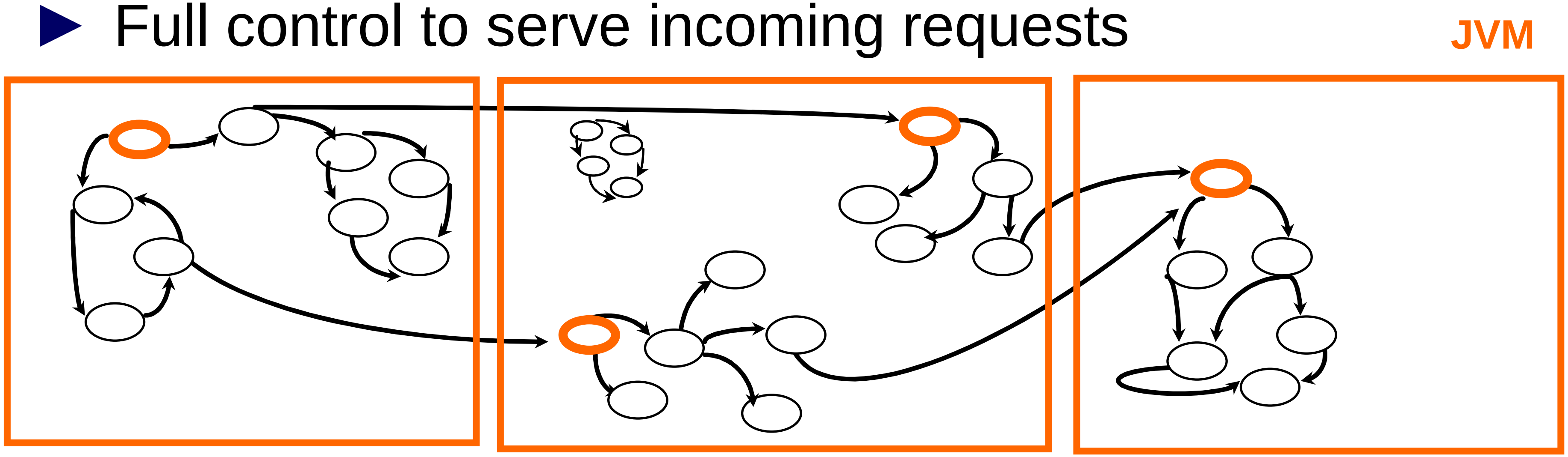
new A(...)

newActive(A,..)

- Activity
- Remote Accessible Objects
- Asynchronous Communications
- Code Mobility
- Security
- Fault Tolerance
- Distributed Garbage Collector
- Resource Virtualization

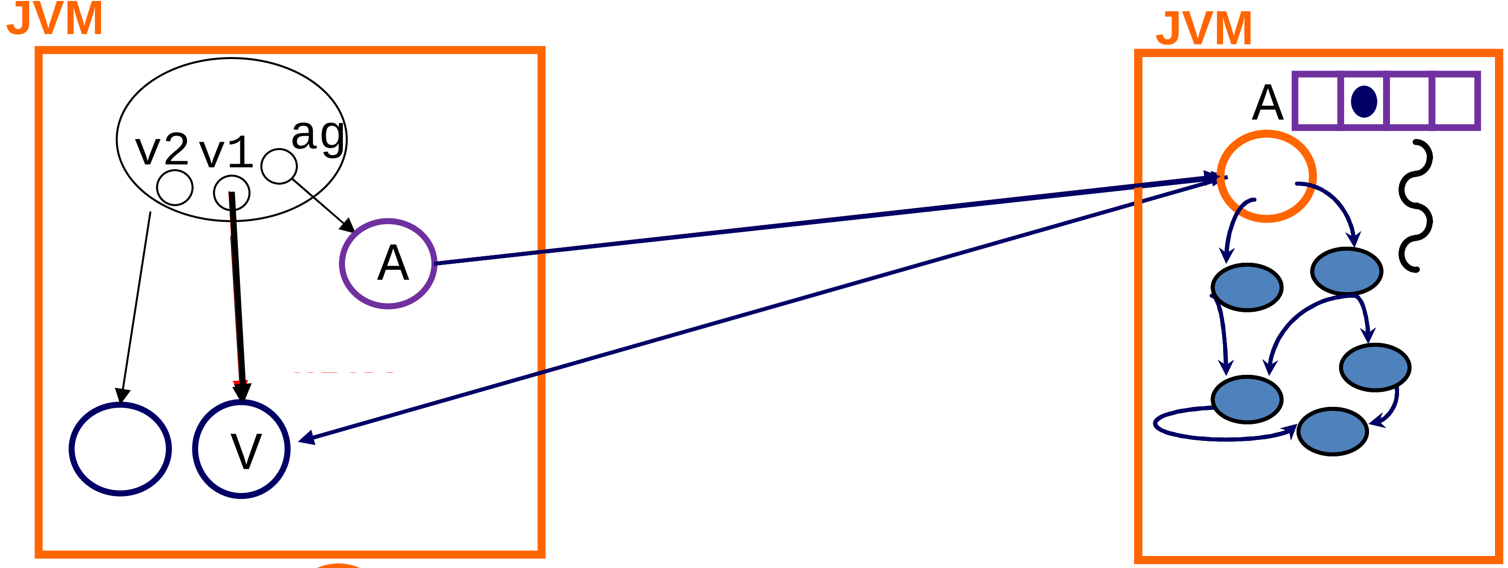
ProActive model : Basis

- Active objects
  - coarse-grained structuring entities (subsystems)
  - has exactly one thread.
  - owns many passive objects (Standard Java Objects, no thead)
  - No shared passive objects -- Parameters are deep-copy
- Remote Method Invocation
  - Asynchronous Communication between active objects
- Full control to serve incoming requests



Active objects

```
A ag = newActive ("A", […], Node)
V v1 = ag.foo (param);
V v2 = ag.bar (param);
...
v1.bar(); //Wait-By-Necessity
```



○ Java Object ○ Active Object ▭▭▭▭ **Req. Queue**

○ Future Object ○ Proxy ● **Request** **Thread**

Wait-By-Necessity provides Dataflow Synchronization

ProActive : Creating active objects

- ❑ An object created with `A a = new A (obj, 7);`
- ❑ can be turned into an active and remote object:

❑ Instantiation-based: The most general case.

❑
```
A a = (A)ProActive.newActive(«A», params, node);
```

❑ Class-based: In combination with a static method as a factory

To get a non-FIFO behavior (Class-based):

❑
```
class pA extends A implements RunActive { … };
```

❑ Object-based:

```
A a = new A (obj, 7);
...
...
a = (A)ProActive.turnActive (a, node);
```

Wait by necessity

- A call on an active object consists in 2 steps
  - ☐ **A query** : name of the method, parameters…
  - ☐ **A Reply** : the result of the method call
- A query returns a Future object which is a placeholder for the result
- The callee will update the Future when the result is available
- The caller can continue its execution event if the Future has not been updated

```
foo ()
{
  Result r = a.process();
  //do other things
  ...
  r.toString();
}
```

will block if not available

```
Result process()
{
  //perform long
  //calculation

  return result;
}
```

ProActive : Explicit Synchronizations

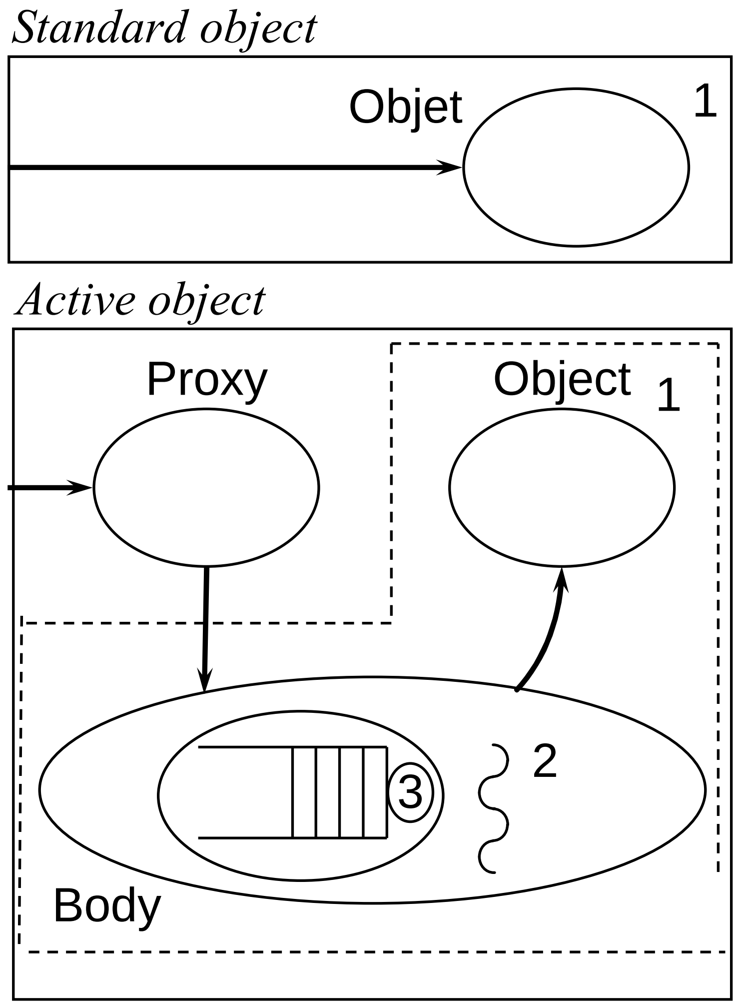
```
A ag = newActive ("A", […], VirtualNode)
V v = ag.foo(param);
...
v.bar(); //Wait-by-necessity
```

- Explicit Synchronization:
  - `- ProActive.isAwaited (v);   // Test if vailable`
  - `-          .waitFor (v);     // Wait until availab.`
- Vectors of Futures:
  - `-         .waitForAll (Vector); // Wait All`
  - `-         .waitForAny (Vector); // Get First`

ProActive : Active object

An active object is composed of several objects :

- The object being activated: Active Object (1)
- A set of standard Java objects
- A single thread (2)
- The queue of pending requests (3)

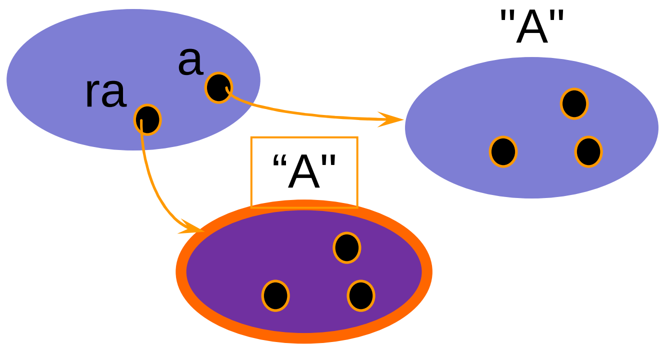


ProActive : Reuse and seamless

- Two key features:

- **Polymorphism** between standard and active objects
  - ☐- Type compatibility for classes (and not only interfaces)
  - ☐- Needed and done for the future objects also
  - ☐- Dynamic mechanism (dynamically achieved if needed)

- **Wait-by-necessity**: inter-object synchronization



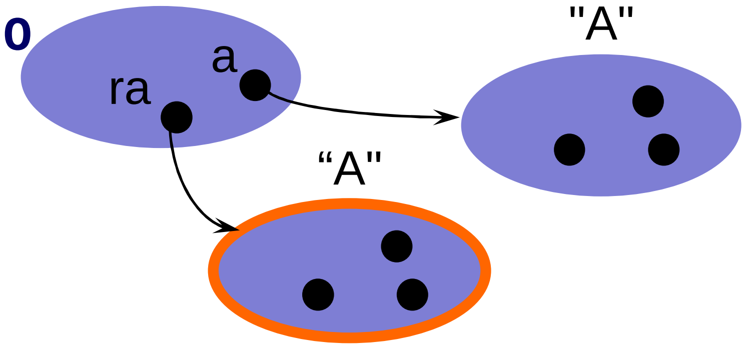
```
foo (A a)
{
   a.g (...);
   v = a.f (...);
   ...
   v.bar (...);
}
```

ProActive : Reuse and seamless

- Two key features:

- **Polymorphism** between standard and active objects
  - ☐- Type compatibility for classes (and not only interfaces)
  - ☐- Needed and done for the future objects also
  - ☐- Dynamic mechanism (dynamically achieved if needed)

- **Wait-by-necessity**: inter-object synchronization
  - ☐- Systematic, implicit and transparent futures
  - ☐- Ease the programming of synchronizations, and the reuse of routines



```
foo (A a)
{
   a.g (...);
   v = a.f (...);
   ...
   v.bar (...);
}
```

O.foo(a)
a.g() and a.f() are « local »

O.foo(ra):
a.g() and a.f()are «remote + Async.»

ProActive : Reuse and seamless

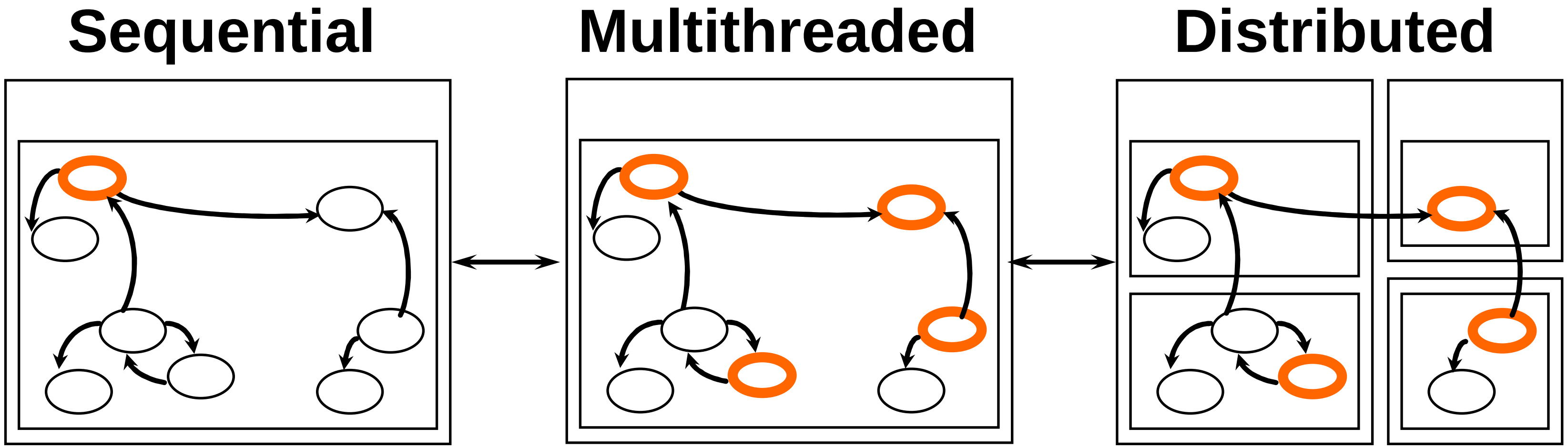
- **Polymorphism** between standard and active objects
  - Type compatibility for classes (and not only interfaces)
  - Needed and done for the future objects also

- **Wait-by-necessity**: inter-object synchronization
  - Systematic, implicit and transparent futures
  - Ease the programming of synchronizations, and the reuse of routines

Intra Active Object Synchronizations

ProActive:
Inter- to Intra- Synchronization

Inter-Synchro: mainly Data-Flow



Synchronizations do not dependent upon

the physical location (mapping of activities)

ProActive : Intra-object synchronization

- Explicit control:
- Library of service routines:
  - Non-blocking services,...
    - **serveOldest ();**
    - **serveOldest (f);**
  - Blocking services, timed, etc.
    - **serveOldestBl ();**
    - **serveOldestTm (ms);**
  - Waiting primitives
    - **waitARequest();**
    - **etc.**

```
class BoundedBuffer extends FixedBuffer
              implements RunActive {

     // Programming Non FIFO behavior
 runActivity (ExplicitBody myBody) {
    while (...) {
      if (this.isFull())
           serveOldest("get");
      else if (this.isEmpty())
           serveOldest ("put");
      else serveOldest ();

      // Non-active wait
      waitArequest ();
  }
}}
```

Implicit (declarative) control: library classes

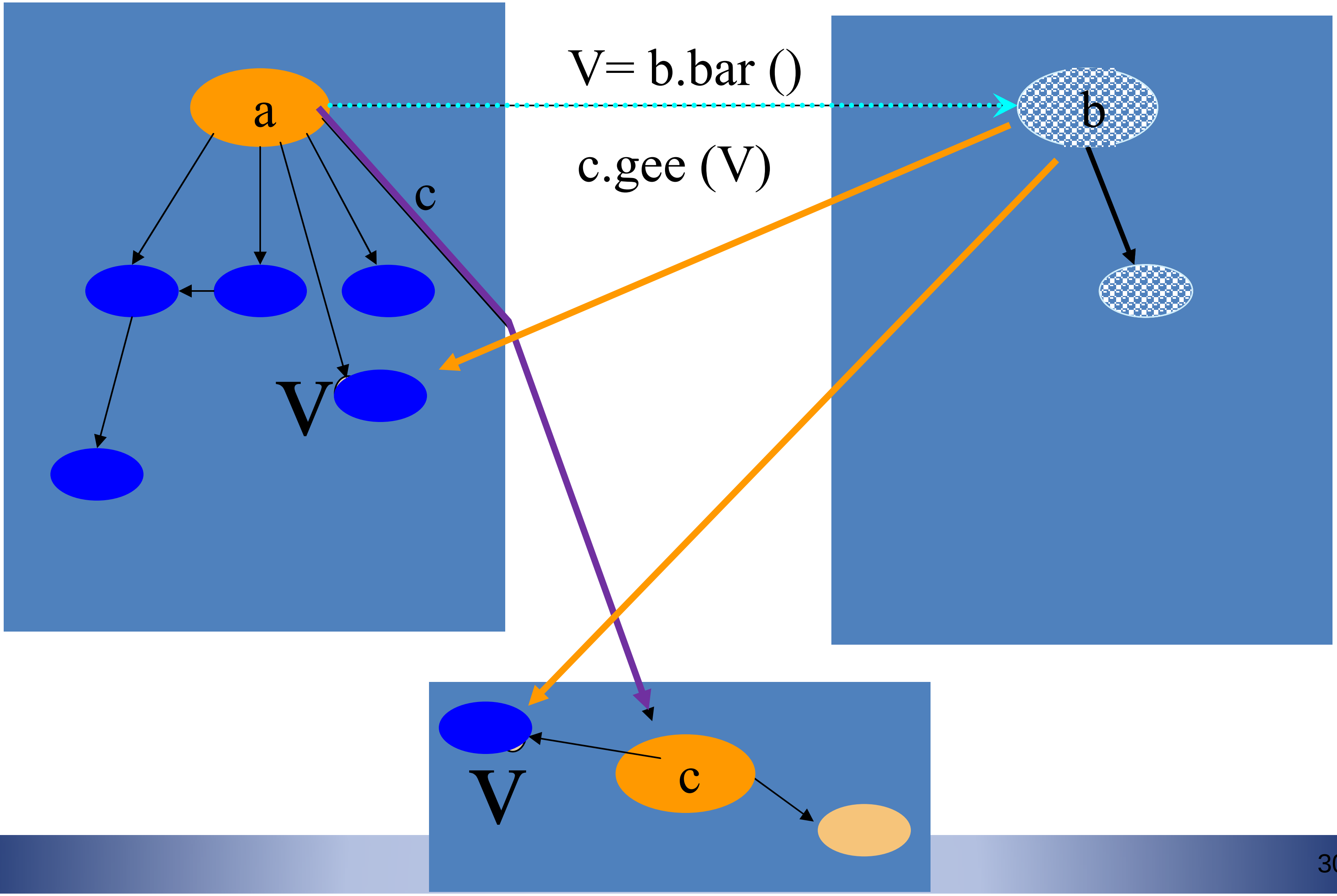
`e.g. : Blocking Condition Abstraction for concurrency control:`

`doNotServe ("put", "isFull");`

First-Class Futures Update

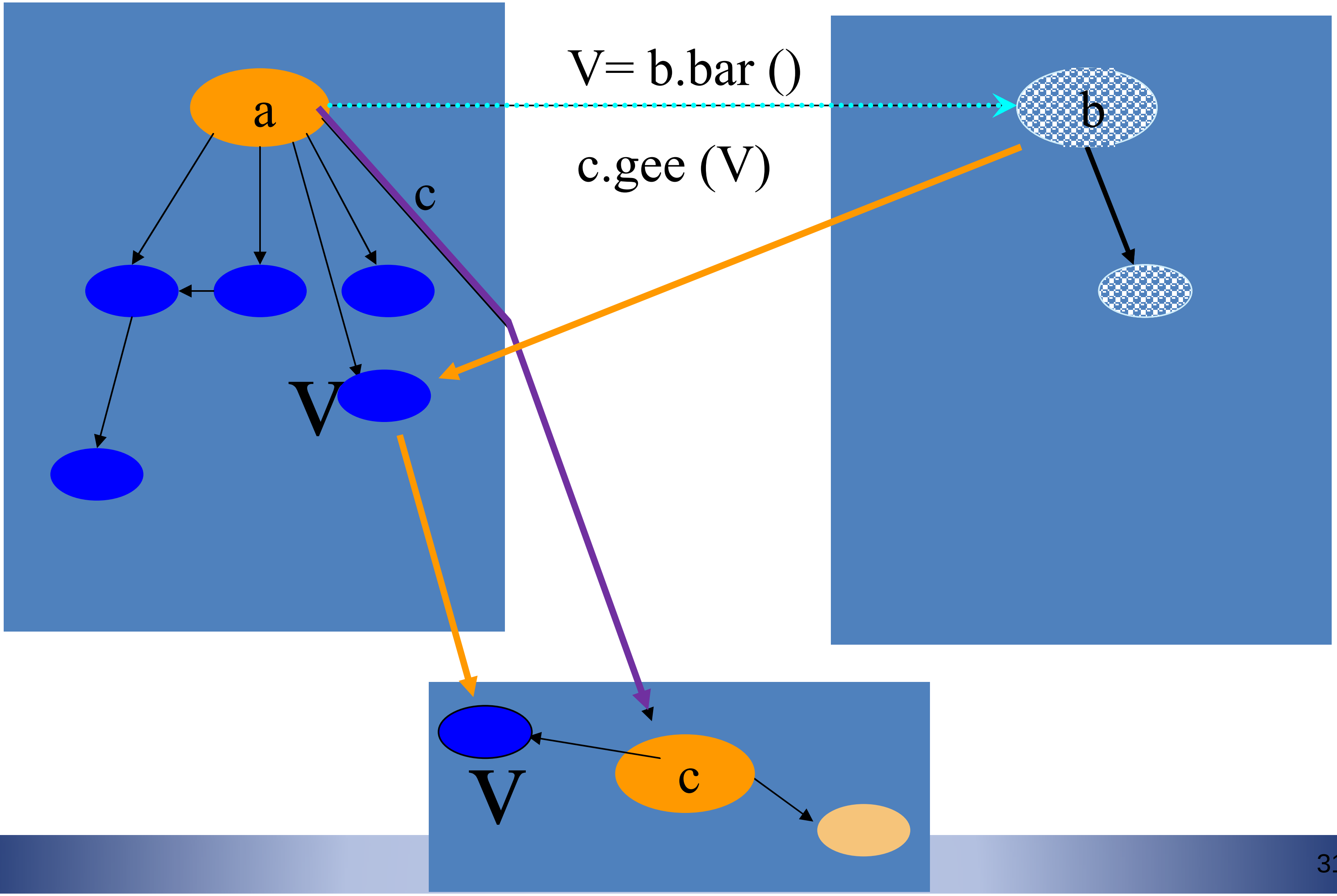
Wait-By-Necessity: First Class Futures

Futures are Global Single-Assignment Variables



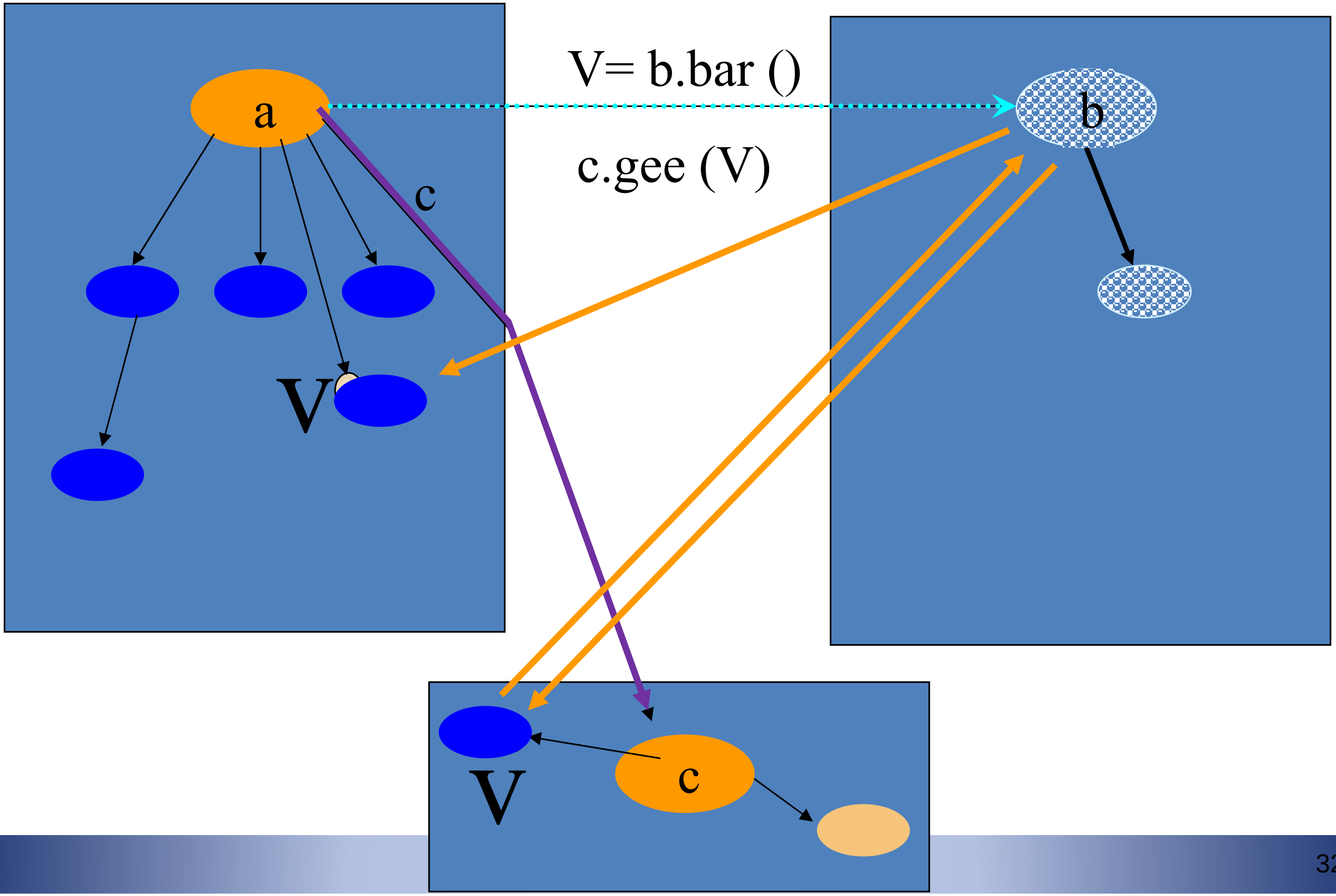
Wait-By-Necessity: Eager Forward Based

AO forwarding a future: will have to forward its value



Wait-By-Necessity: Eager Message Based

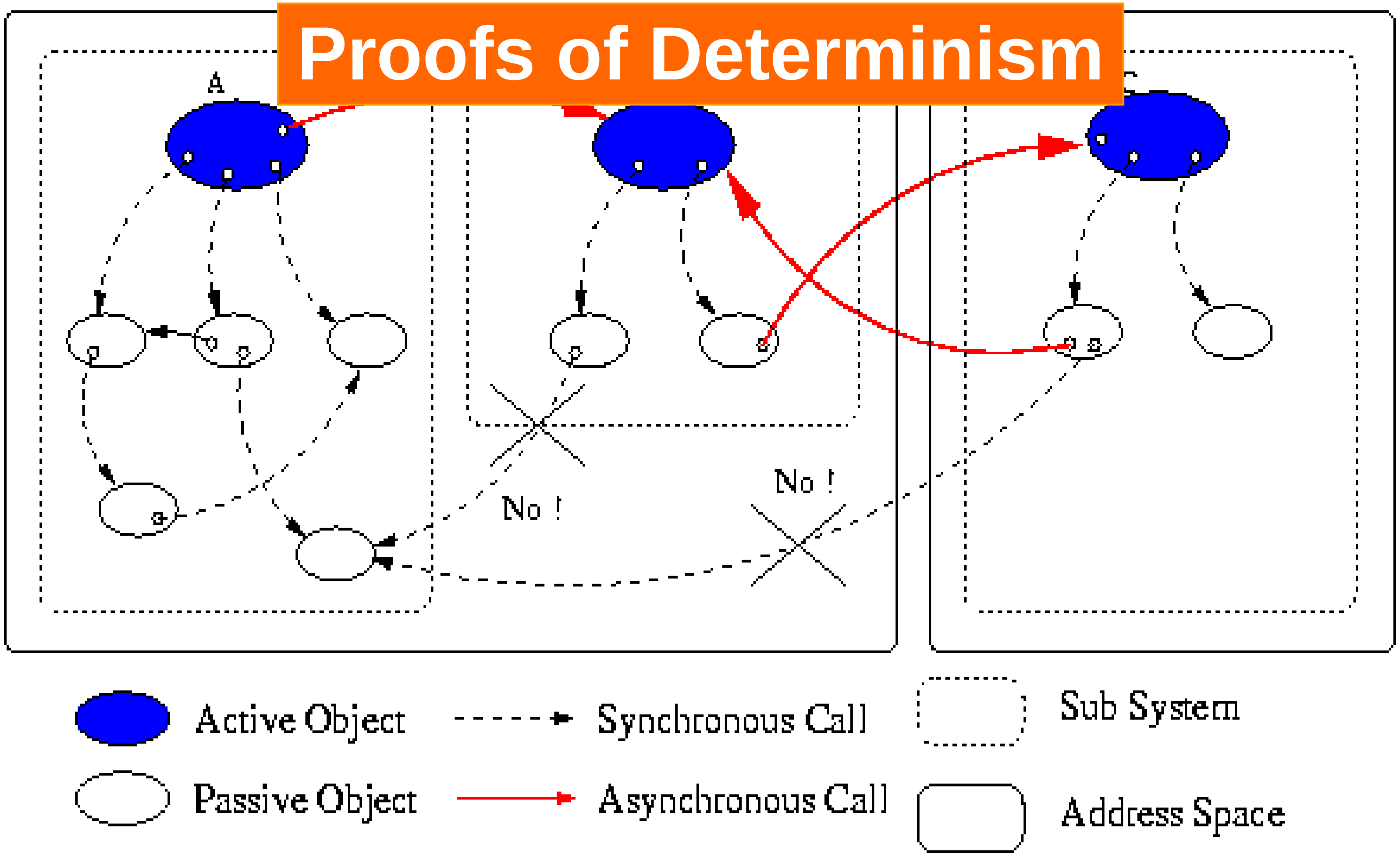
AO receiving a future: send a message



Standard system at Runtime: No Sharing

NoC: Network On Chip

Proofs of Determinism



Proofs in GREEK

$$\frac{(a,\sigma) \rightarrow_S (a',\sigma')}{\alpha[a;\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a';\sigma';\iota;F;R;f] \parallel P}\ (\textsc{Local})$$

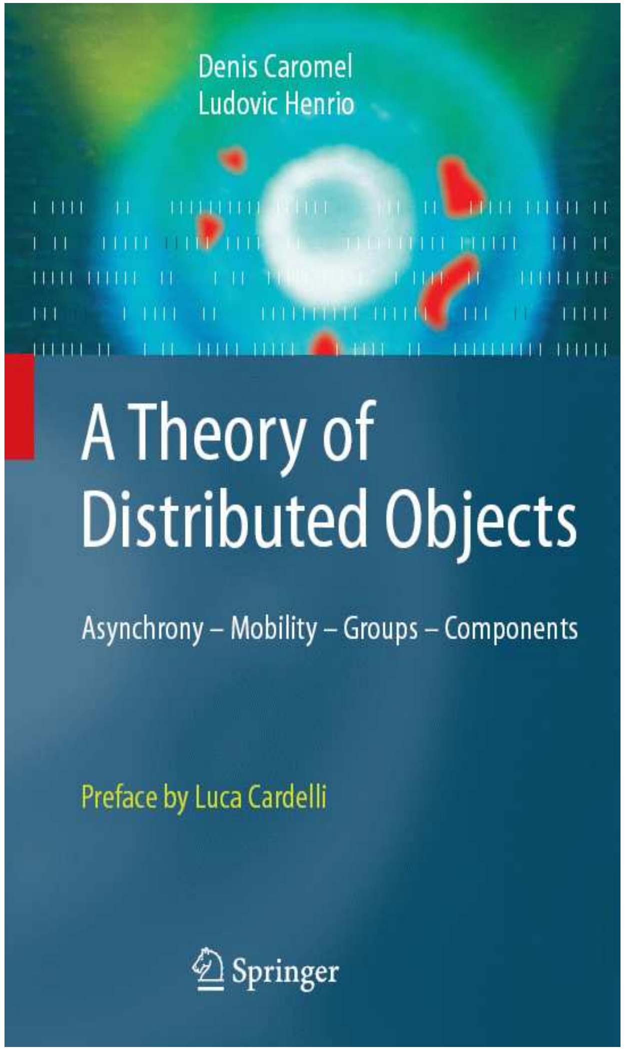
$$\frac{\begin{array}{c}\gamma \text{ fresh activity} \qquad \iota' \notin \mathit{dom}(\sigma) \qquad \sigma' = \{\iota' \mapsto AO(\gamma)\} :: \sigma \\ \sigma_\gamma = \mathit{copy}(\iota'',\sigma) \qquad \mathit{Service} = (\text{ if } m_j = \emptyset \text{ then } \mathit{FifoService} \text{ else } \iota''.m_j())\end{array}}{\begin{array}{c}\alpha[\mathcal{R}[\mathit{Active}(\iota'',m_j)];\sigma;\iota;F;R;f] \parallel P \\ \longrightarrow \alpha[\mathcal{R}[\iota'];\sigma';\iota;F;R;f] \parallel \gamma[\mathit{Service};\sigma_\gamma;\iota'';\emptyset;\emptyset;\emptyset] \parallel P\end{array}}\ (\textsc{NewAct})$$

$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = AO(\beta) \qquad \iota'' \notin \mathit{dom}(\sigma_\beta) \qquad f_i^{\alpha\rightarrow\beta} \text{ new future} \qquad \iota_f \notin \mathit{dom}(\sigma_\alpha) \\ \sigma'_\beta = \mathit{Copy\&Merge}(\sigma_\alpha,\iota'\ ;\ \sigma_\beta,\iota'') \qquad \sigma'_\alpha = \{\iota_f \mapsto \mathit{fut}(f_i^{\alpha\rightarrow\beta})\} :: \sigma_\alpha\end{array}}{\begin{array}{c}\alpha[\mathcal{R}[\iota.m_j(\iota')];\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[\mathcal{R}[\iota_f];\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma'_\beta;\iota_\beta;F_\beta;R_\beta :: [m_j;\iota'';f_i^{\alpha\rightarrow\beta}];f_\beta] \parallel P\end{array}}\ (\textsc{Request})$$

$$\frac{R = R' :: [m_j;\iota_r;f'] :: R'' \qquad m_j \in M \qquad \forall m \in M, m \notin R'}{\alpha[\mathcal{R}[\mathit{Serve}(M)];\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[\iota.m_j(\iota_r) \Uparrow f, \mathcal{R}[[]];\sigma;\iota;F;R' :: R'';f'] \parallel P}\ (\textsc{Serve})$$

$$\frac{\iota' \notin \mathit{dom}(\sigma) \qquad F' = F :: \{f \mapsto \iota'\} \qquad \sigma' = \mathit{Copy\&Merge}(\sigma,\iota\ ;\ \sigma,\iota')}{\alpha[\iota \Uparrow (f',a);\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a;\sigma';\iota;F';R;f'] \parallel P}\ (\textsc{EndService})$$

$$\frac{\sigma_\alpha(\iota) = \mathit{fut}(f_i^{\gamma\rightarrow\beta}) \qquad F_\beta(f_i^{\gamma\rightarrow\beta}) = \iota_f \qquad \sigma'_\alpha = \mathit{Copy\&Merge}(\sigma_\beta,\iota_f\ ;\ \sigma_\alpha,\iota)}{\begin{array}{c}\alpha[a_\alpha;\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[a_\alpha;\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P\end{array}}\ (\textsc{Reply})$$



Denis Caromel
Ludovic Henrio

A Theory of Distributed Objects

Asynchrony – Mobility – Groups – Components

Preface by Luca Cardelli

Springer

ProActive Core

PROACTIVE GROUPS

ProActive Groups

- Manipulate groups of Active Objects, in a simple and typed manner:
  - ➡ Typed and polymorphic Groups of local and remote objects
  - ➡ Dynamic generation of group of results
  - ➡ Language centric, Dot notation

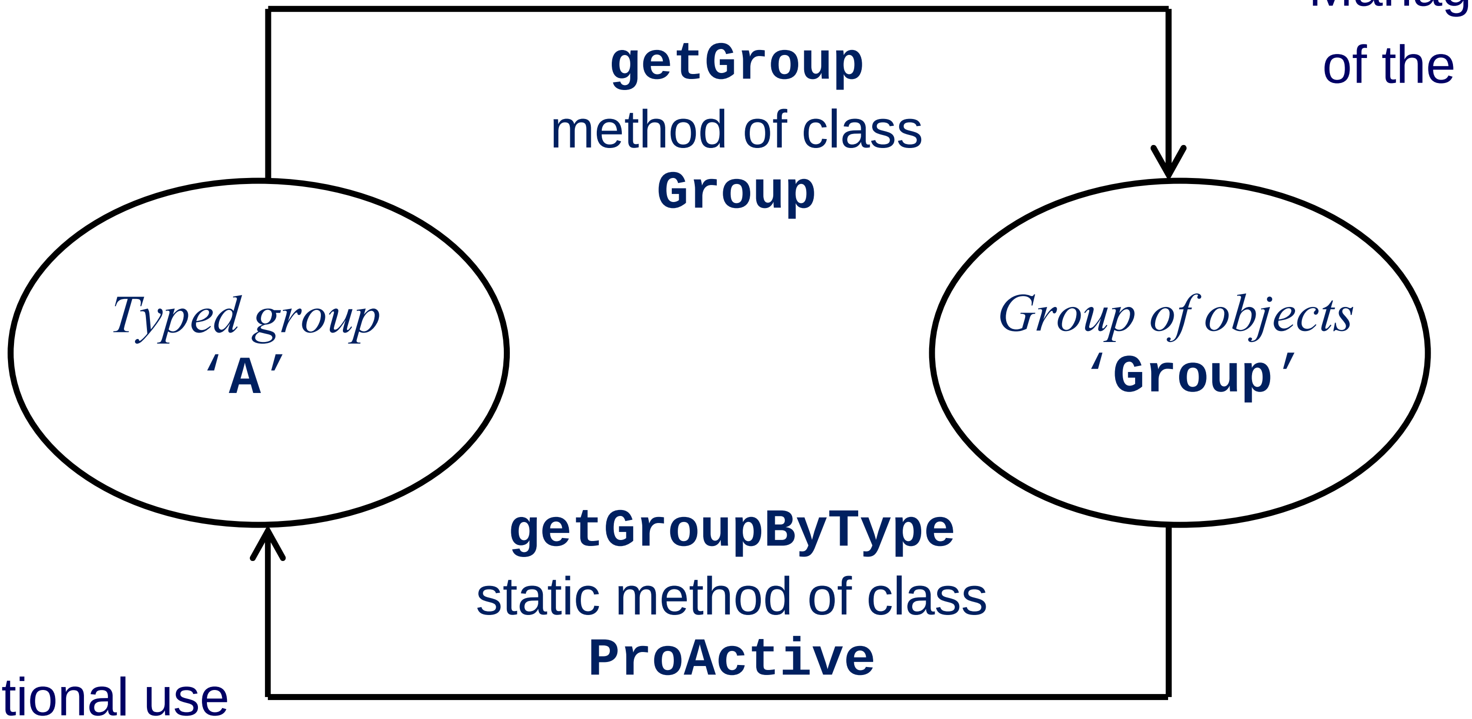
- Be able to express high-level collective communications (like in MPI):
  - broadcast,
  - scatter, gather,
  - all to all

```
A ag=(A)ProActiveGroup.newGroup(«A»,{{p1},...},{Nodes,..});
V v = ag.foo(param);
v.bar();
```

ProActive Groups

- Group Members
  - Active Objects
  - POJO
  - Group Objects
- Hierarchical Groups
- Based on the ProActive communication mechanism
  - Replication of N ‘ single ’ communications
  - Parallel calls within a group (latency hiding)
- Polymorphism
  - Group typed with member’s type

Two Representations Scheme



Management

of the group

`getGroup`
method of class
`Group`

Typed group
`'A'`

Group of objects
`'Group'`

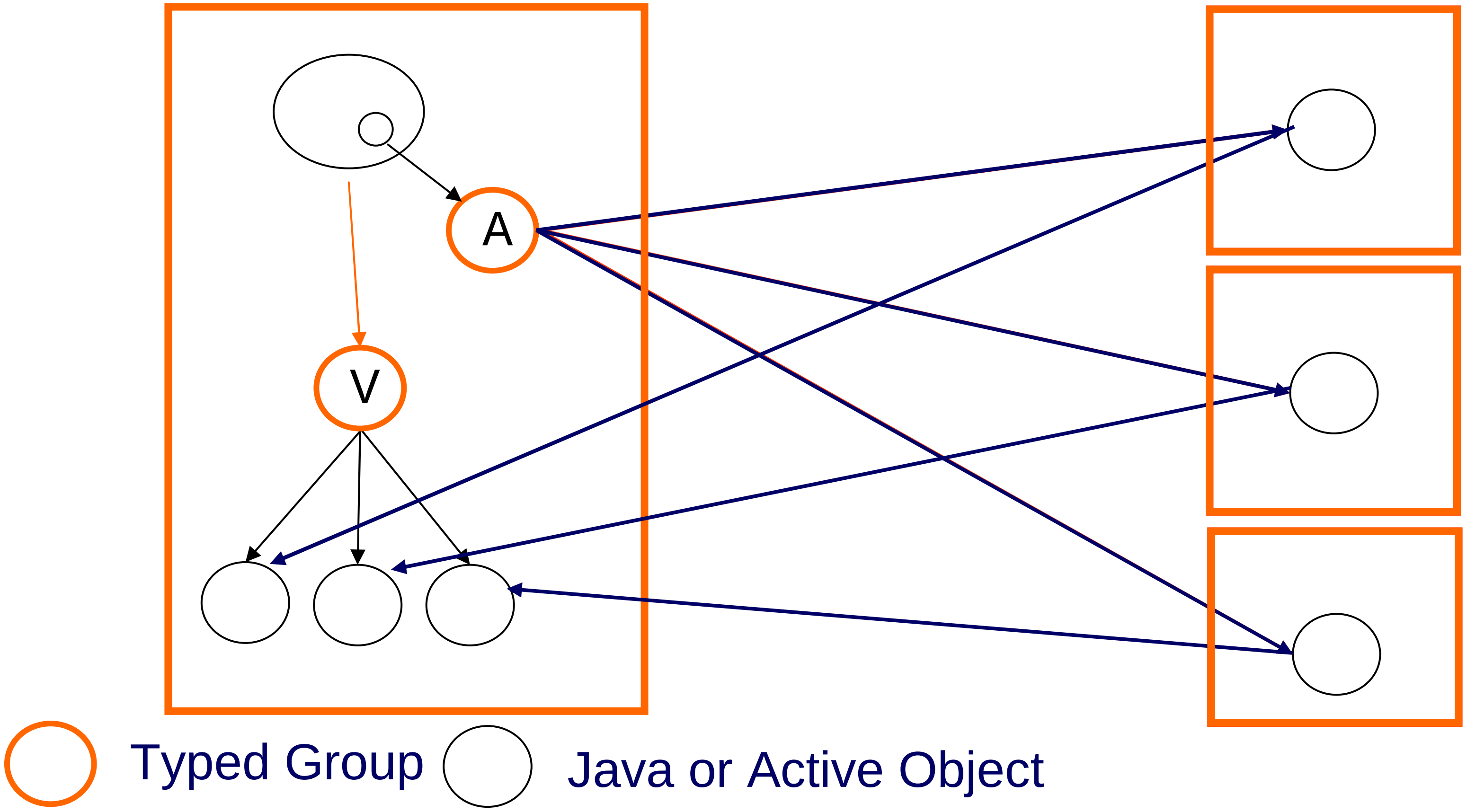
`getGroupByType`
static method of class
`ProActive`

Functional use

of the group

Creating AO and Groups

- `A ag = newGroup ("A", […], Node[])`
- `V v = ag.foo(param);`
- `…`
- `v.bar(); //Wait-by-necessity`

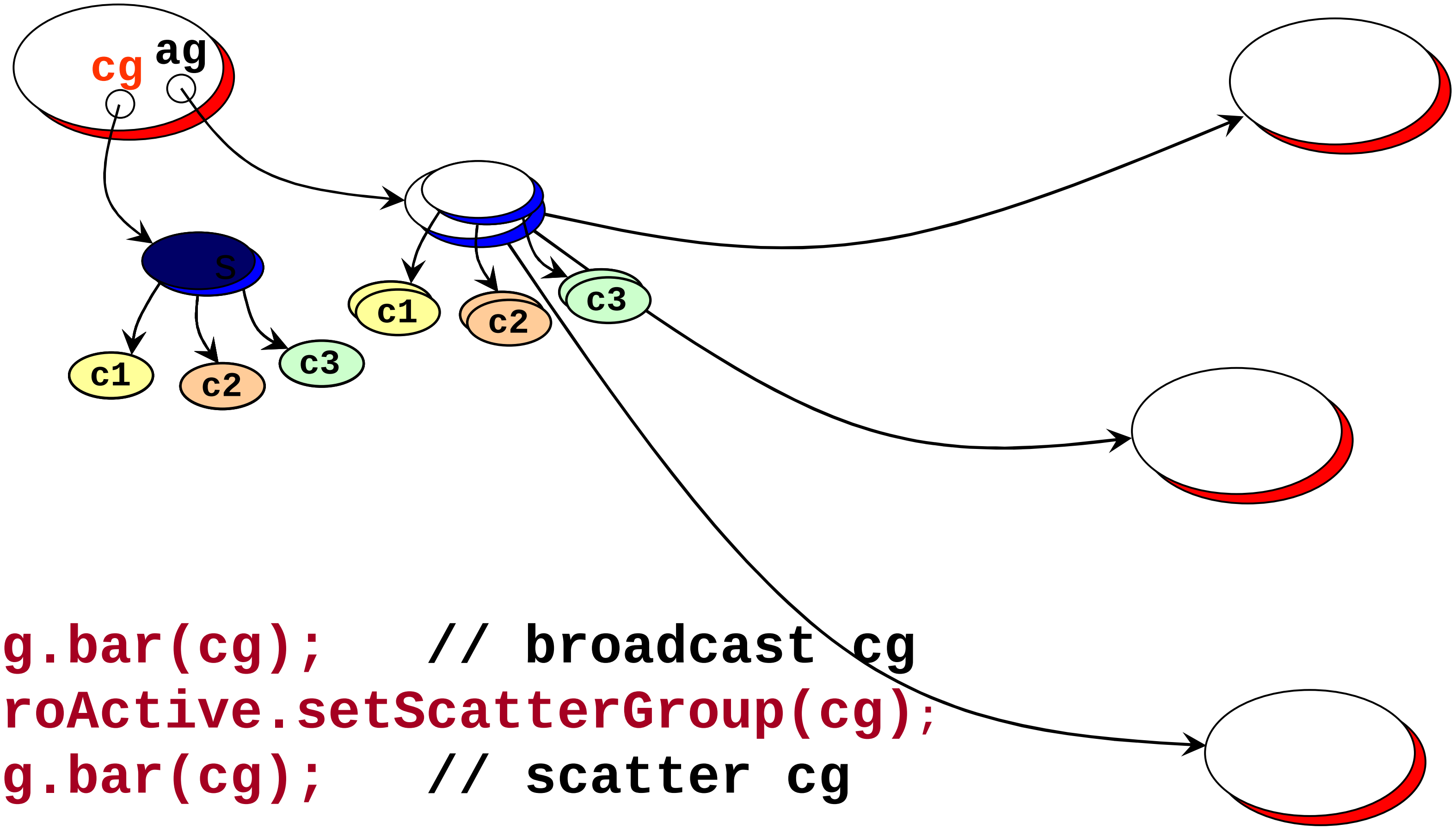


Typed Group as Result of Group Communication

- Ranking Property:
  - Dynamically built and updated
    - B groupB = groupA.foo();
  - Ranking property: order of result group members = order of called group members
- Explicit Group Synchronization Primitive:
  - Explicit wait
    - ProActiveGroup.waitOne(groupB);
    - ProActiveGroup.waitAll(groupB);
  - Predicates
    - noneArrived
    - kArrived
    - allArrived, ...

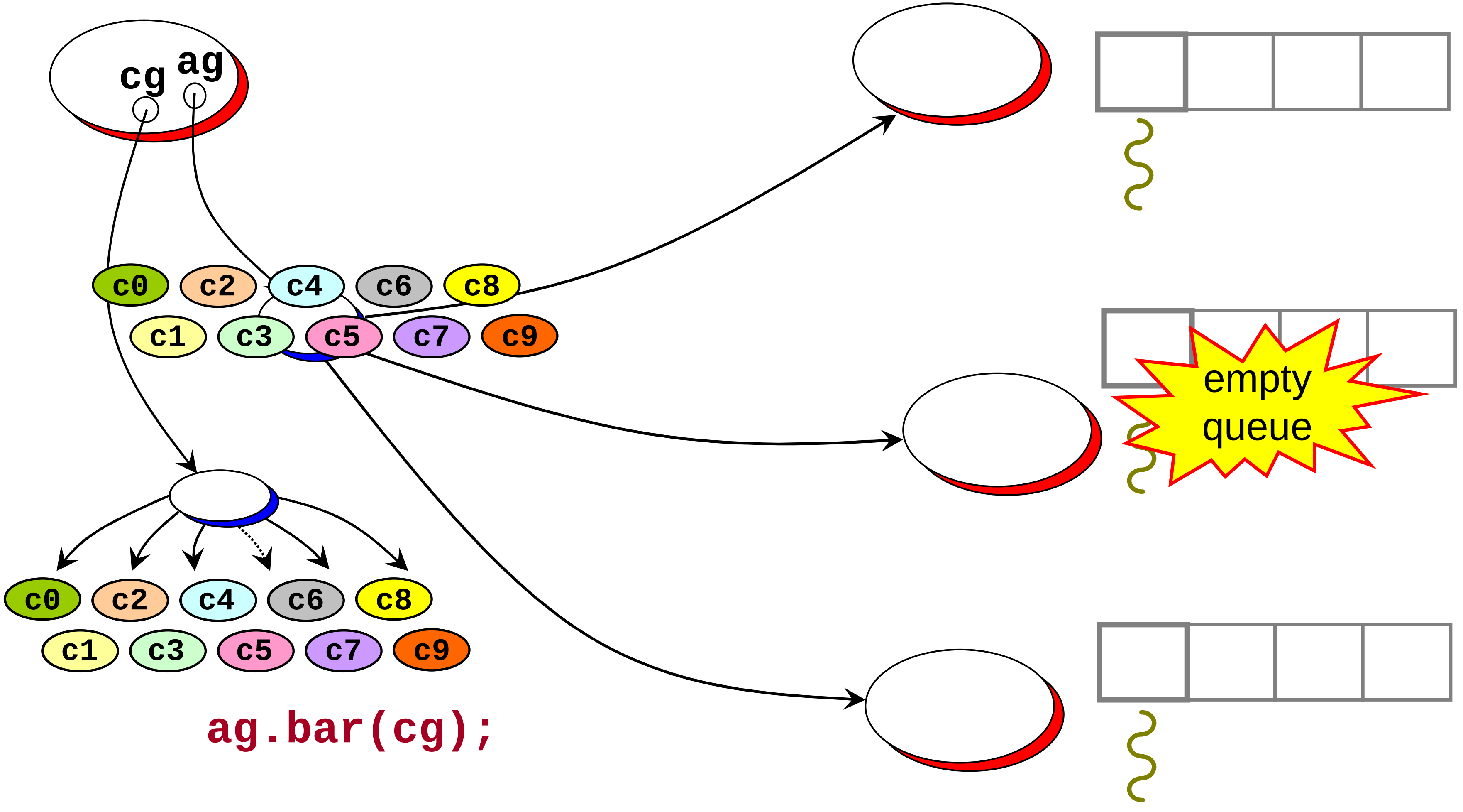
Broadcast and Scatter

Broadcast is the default behavior
Use a group as parameter, Scattered depends on rankings



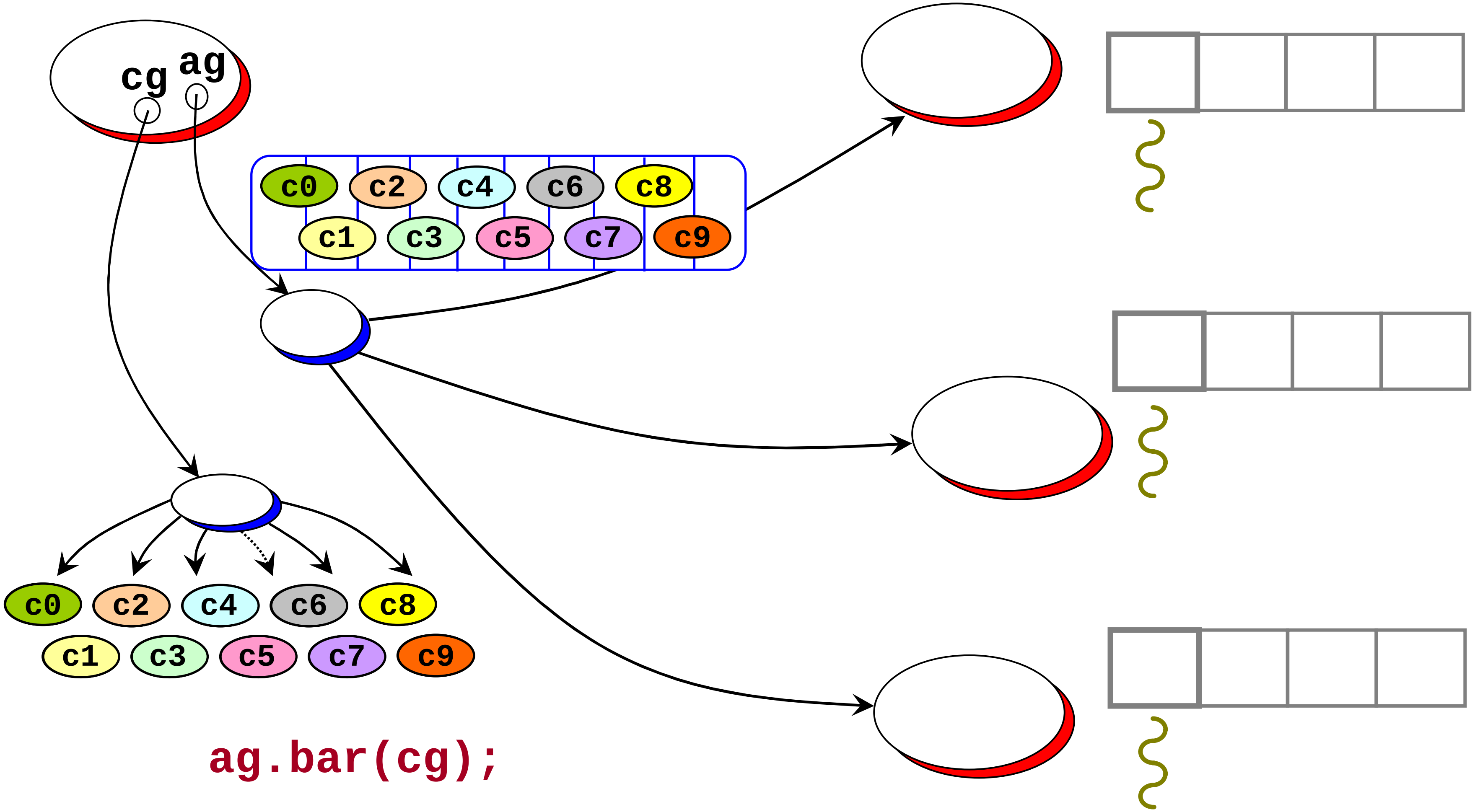
```
ag.bar(cg);   // broadcast cg
ProActive.setScatterGroup(cg);
ag.bar(cg);   // scatter cg
```

Static Dispatch Group



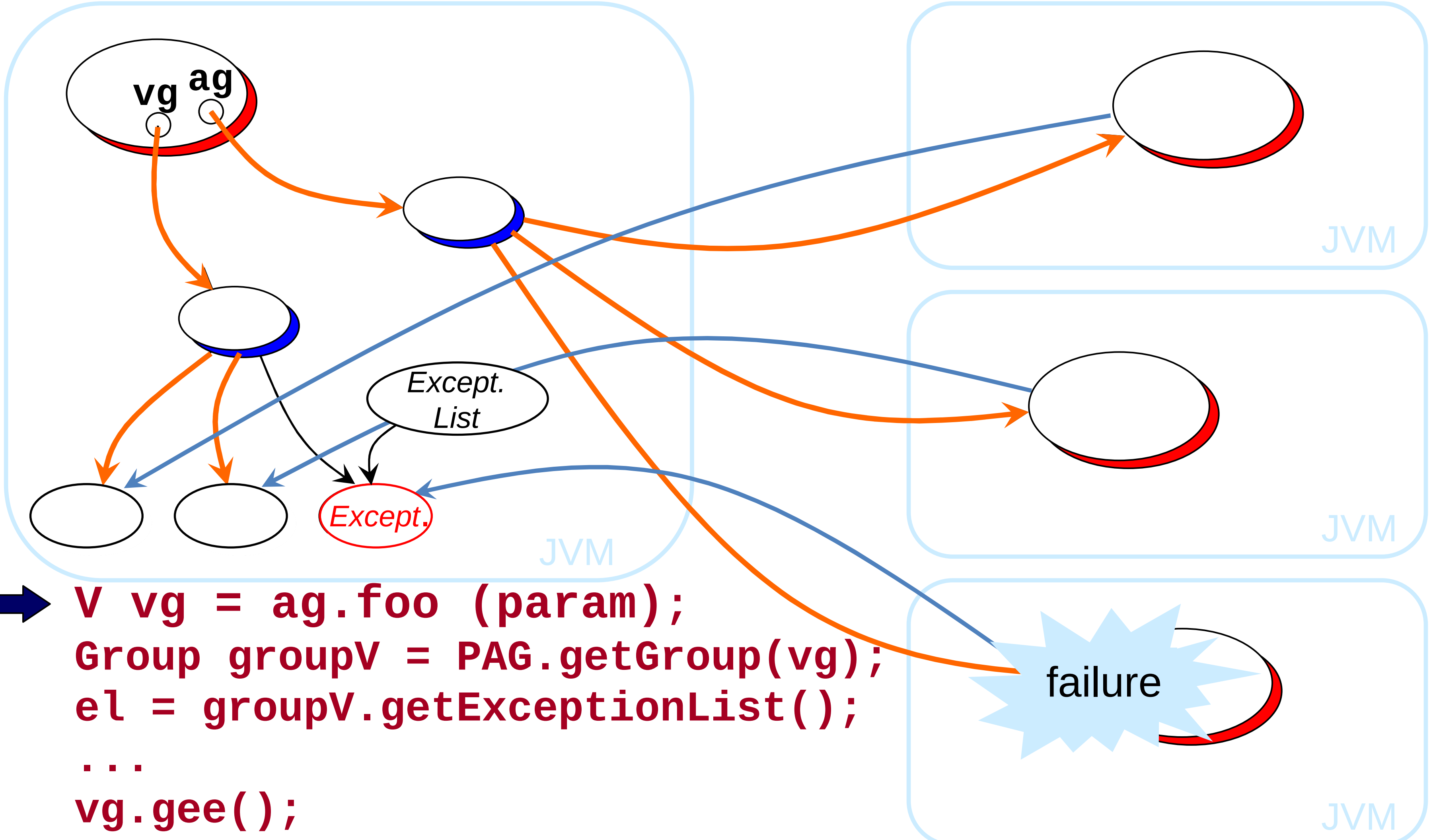
`ag.bar(cg);`

Dynamic Dispatch Group



`ag.bar(cg);`

Handling Group Failures (2)



```
V vg = ag.foo (param);
Group groupV = PAG.getGroup(vg);
el = groupV.getExceptionList();
...
vg.gee();
```

ProActive Core

MIGRATION: MOBILE AGENTS

Mobile Agents: Migration

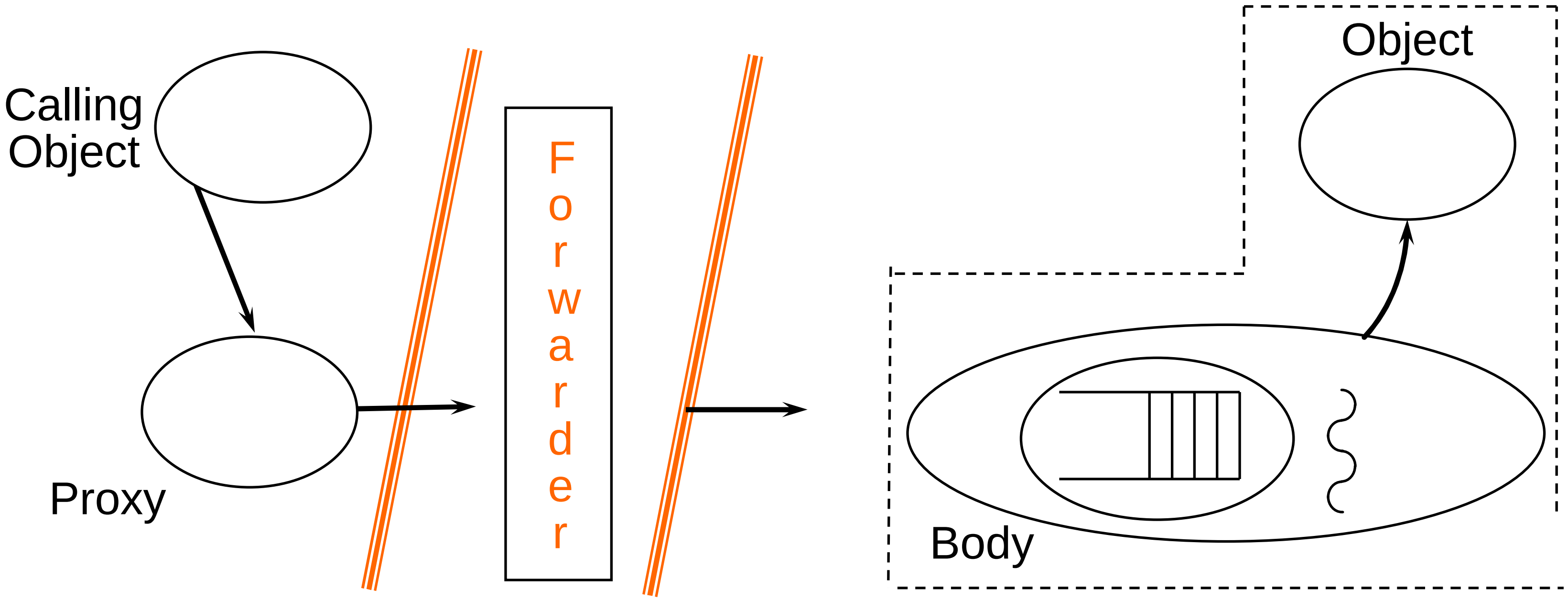
- The active object migrates with:
  - ☐ its state
  - ☐ all pending requests
  - ☐ all its passive objects
  - ☐ all its future objects
- Automatic management of references:
  - ☐ Remote references remain valid: Requests to new location
  - ☐ Previous queries will be fulfilled: Replies to new location
- Migration is initiated by the active object itself
- API: static migrateTo

- Can be initiated from outside through any public method

Migration: Localization Strategies

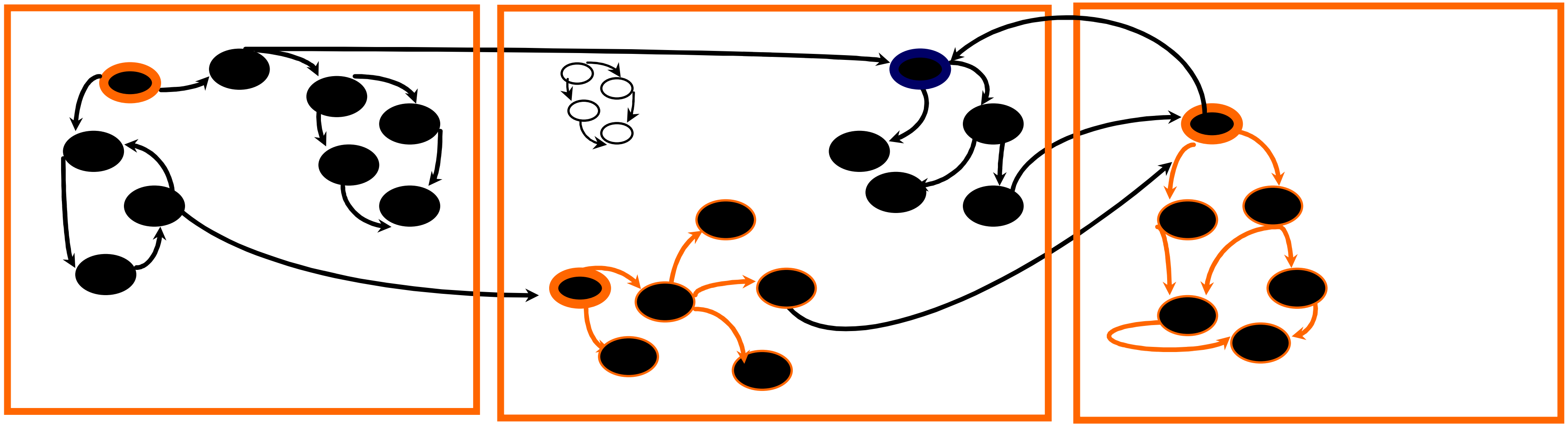
- Forwarders
  - Migration creates a chain of forwarders
  - A forwarder is left at the old location to forward requests to the new location
  - Tensioning: shortcut the forwarder chains by notifying the sender of the new location of the target (transparently)
- Location Server
  - A server (or a set of servers) keeps track of the location of all active objects
  - Migration updates the location on the server
- Mixed (Forwarders / Local Server)
  - Limit the size of the chain up to a fixed size

Migration of AO with Forwarders



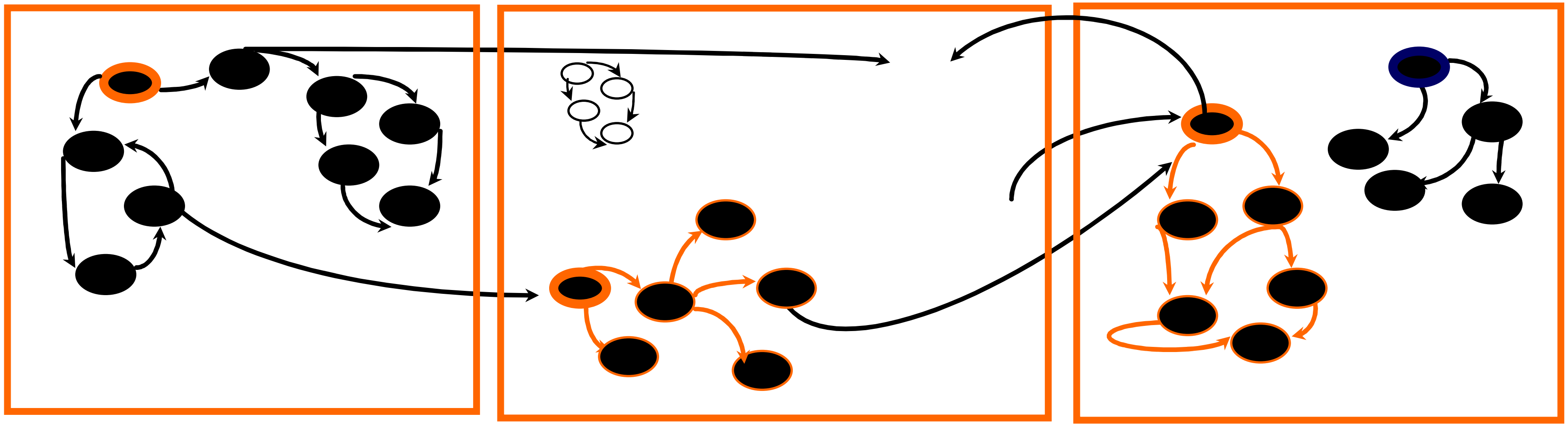
Principles and optimizations

- Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
- Safe migration (no agent in the air!)
- Local references if possible when arriving within a VM
- Tensionning (removal of forwarder)



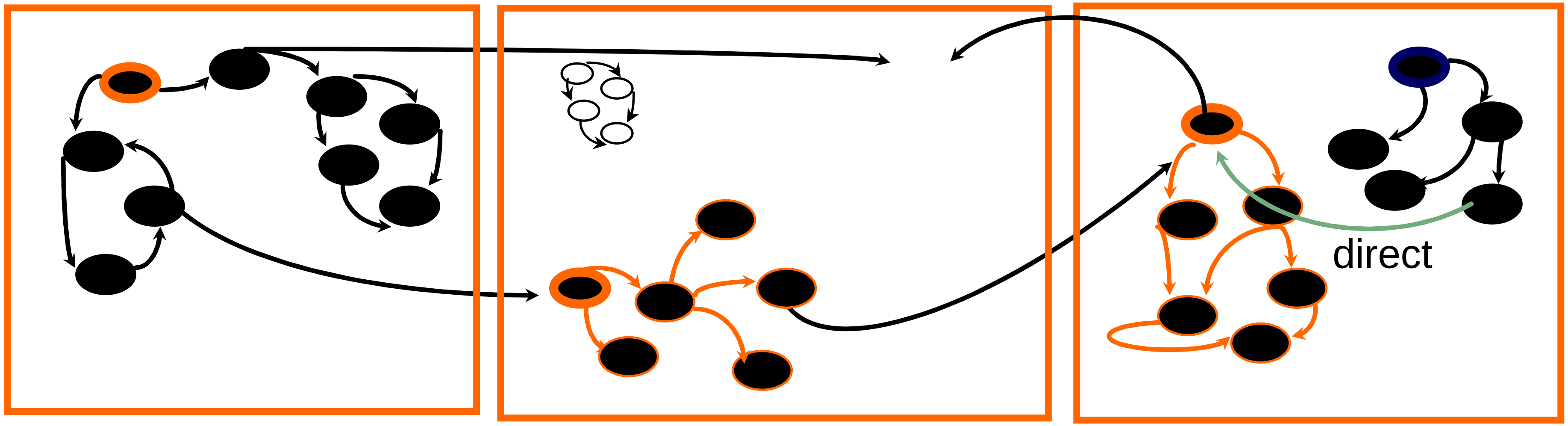
Principles and optimizations

- Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
- Safe migration (no agent in the air!)
- Local references if possible when arriving within a VM
- Tensionning (removal of forwarder)



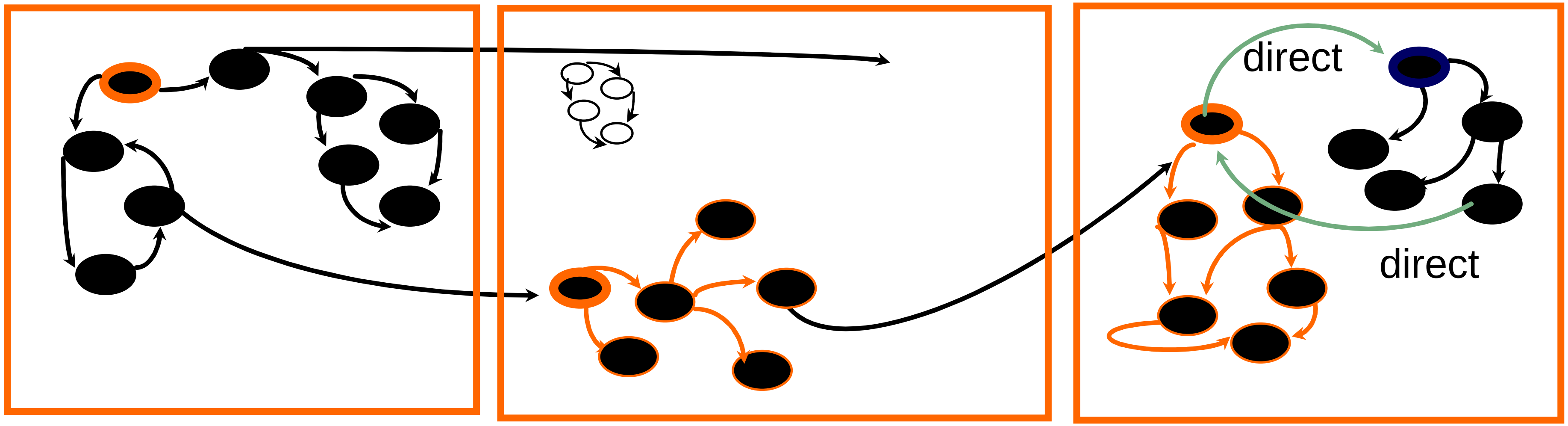
Principles and optimizations

- Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
- Safe migration (no agent in the air!)
- Local references if possible when arriving within a VM
- Tensionning (removal of forwarder)



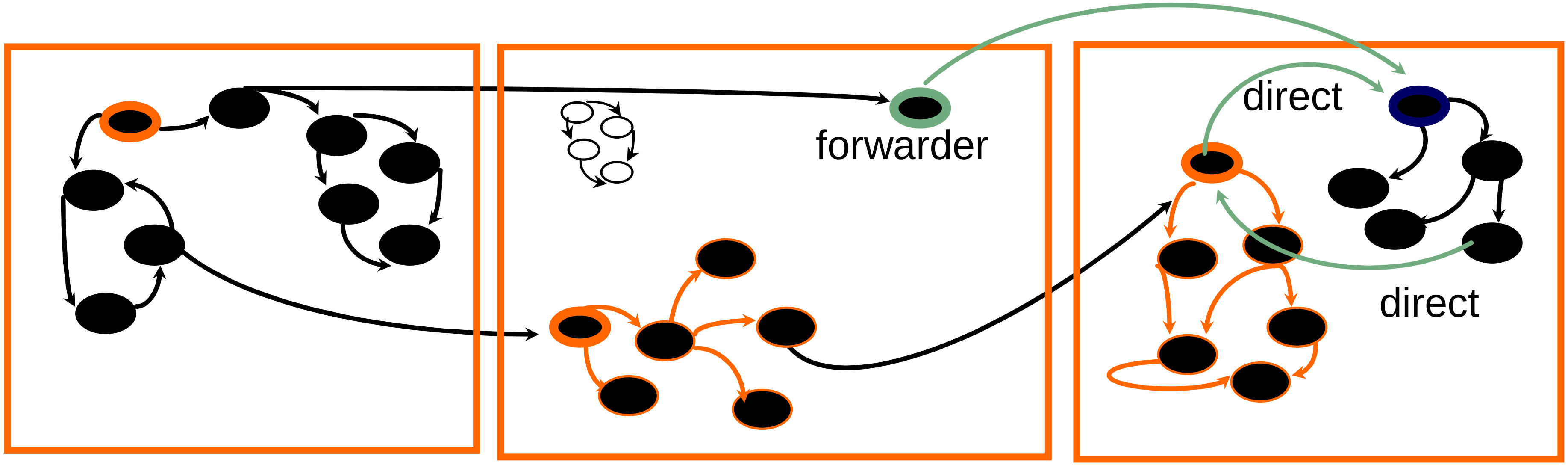
Principles and optimizations

- Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
- Safe migration (no agent in the air!)
- Local references if possible when arriving within a VM
- Tensionning (removal of forwarder)



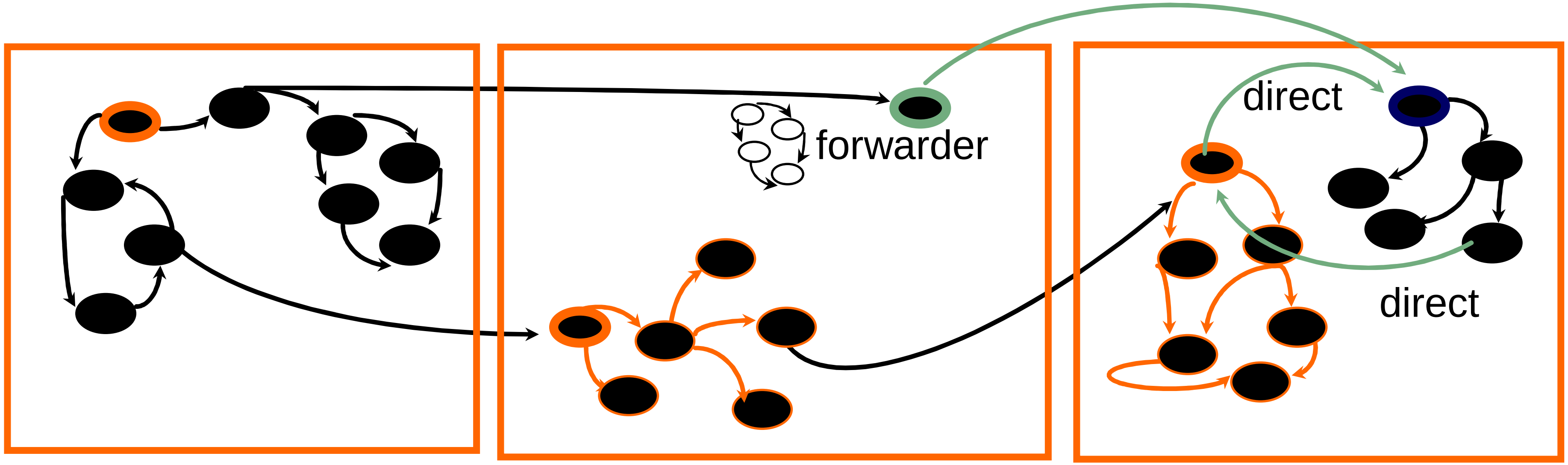
Principles and optimizations

- Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
- Safe migration (no agent in the air!)
- Local references if possible when arriving within a VM
- Tensionning (removal of forwarder)



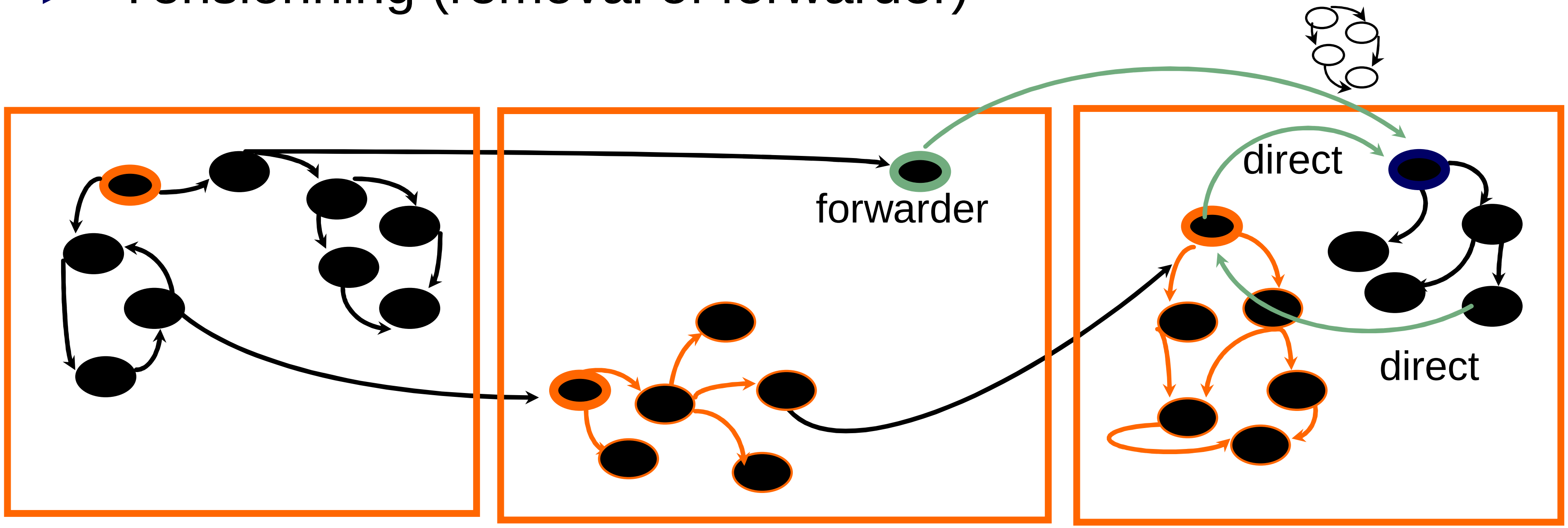
Principles and optimizations

- Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
- Safe migration (no agent in the air!)
- Local references if possible when arriving within a VM
- Tensionning (removal of forwarder)



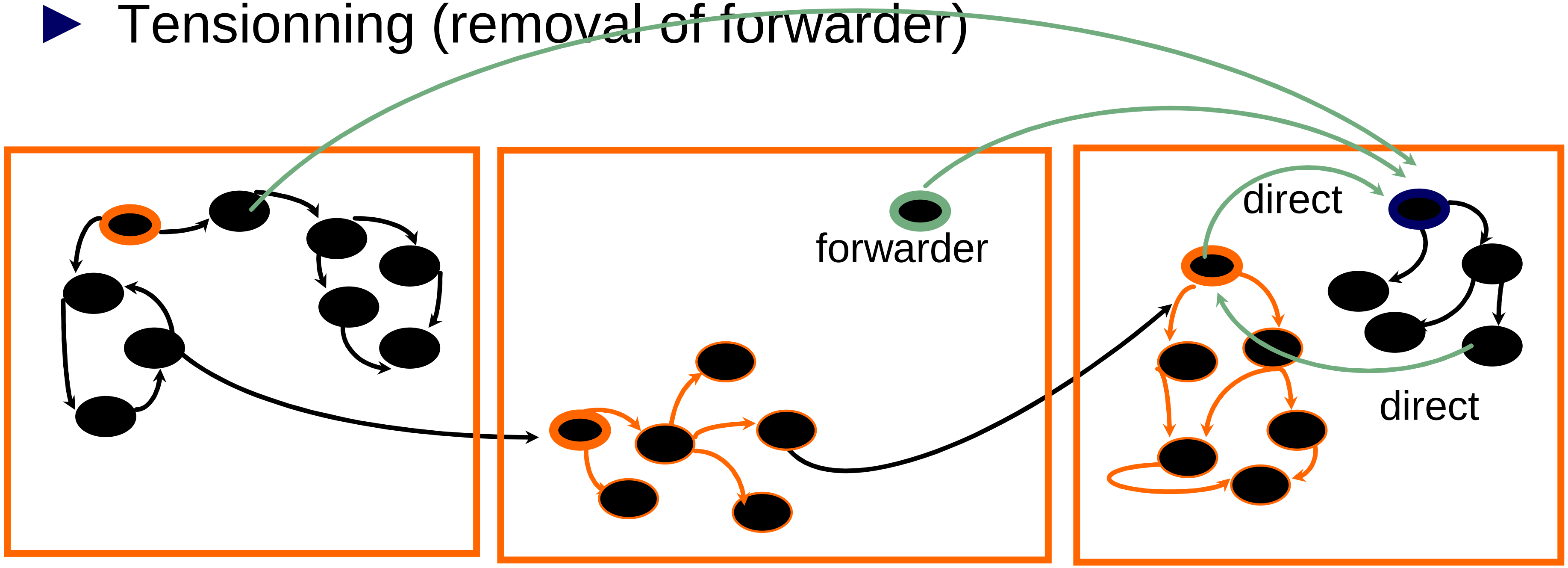
Principles and optimizations

- Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
- Safe migration (no agent in the air!)
- Local references if possible when arriving within a VM
- Tensionning (removal of forwarder)



Principles and optimizations

- Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
- Safe migration (no agent in the air!)
- Local references if possible when arriving within a VM
- Tensionning (removal of forwarder)



ProActive : API for Mobile Agents

- Mobile agents (active objects) that communicate
- Basic primitive: migrateTo
  - public static void migrateTo (String u)
    // string to specify the node (VM)
  - public static void migrateTo (Object o)
    // joinning another active object
  - public static void migrateTo (Node n)
    // ProActive node (VM)
  - public static void migrateTo (JiniNode n)
    // ProActive node (VM)

API for Mobile Agents

- Mobile agents (active objects) that communicate

```
// A simple agent
class SimpleAgent     implements runActive, Serializable {
  public SimpleAgent () {}

  public void moveTo (String t){ // Move upon request
             ProActive.migrateTo (t);
    }
    public String whereAreYou (){ // Repplies to queries
             return (“I am at ” + InetAddress.getLocalHost ());
    }
    public runActivity (Body myBody){
    while (… not end of itinerary …){
       res = myFriend.whatDidYouFind () // Query other agents
             …
             }
             myBody.fifoPolicy(); // Serves request, potentially moveTo
    }
 }
```

API for Mobile Agents

Mobile agents that communicate

Primitive to automatically execute action upon migration

public static void **onArrival** (String r)

// Automatically executes the routine r upon arrival

// in a new VM after migration

public static void **onDeparture** (String r)

// Automatically executes the routine r upon migration

// to a new VM, guaranted safe arrival

public static void **beforeDeparture** (String r)

// Automatically executes the routine r before trying a migration

// to a new VM

API for Mobile Agents Itinerary abstraction

- Itinerary : VMs to visit
  - specification of an itinerary as a list of (site, method)
  - automatic migration from one to another
  - dynamic itinerary management (start, pause, resume, stop, modification, …)

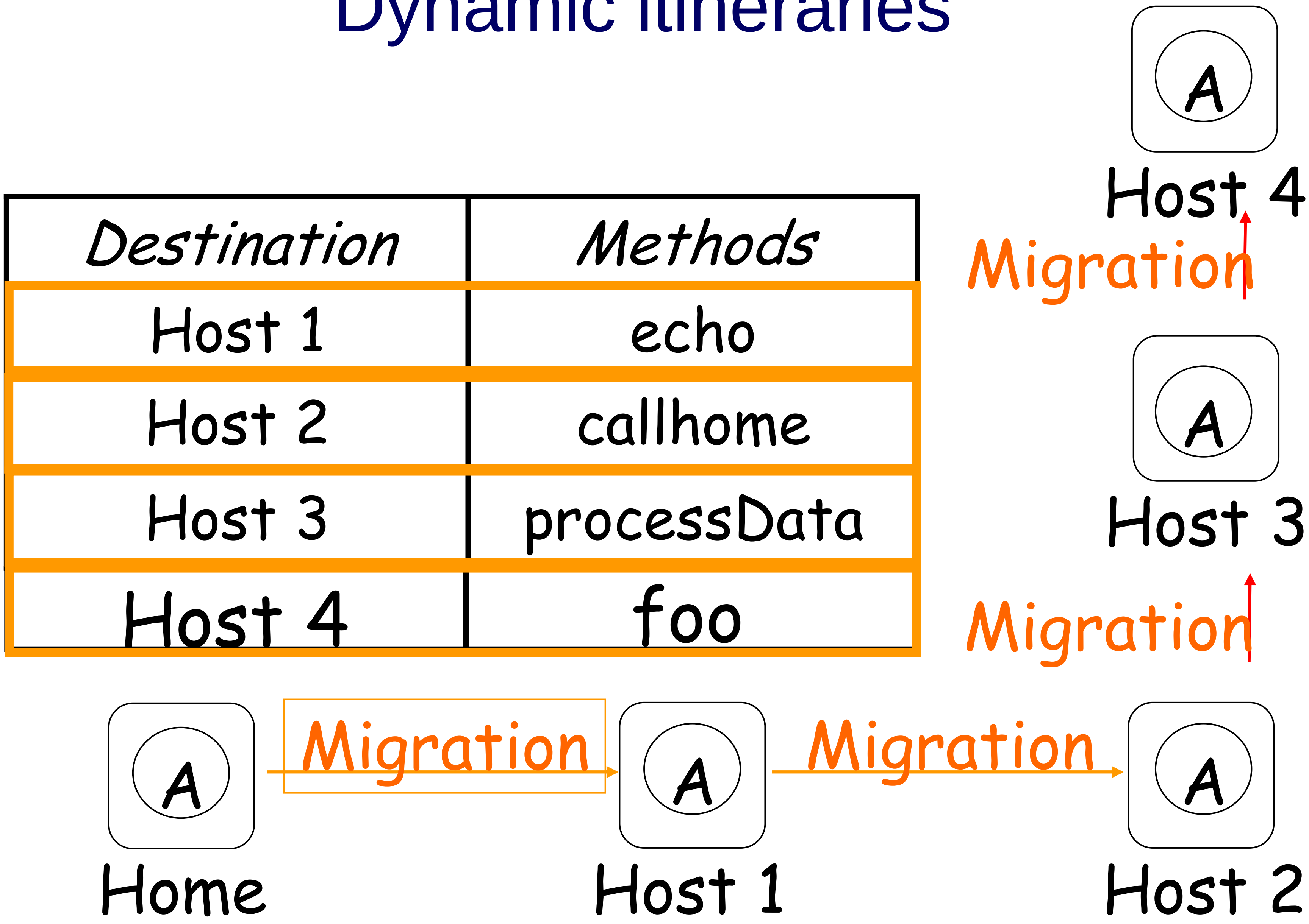
- API:
  - **myItinerary.add ("machine1'', "routineX"); ...**
  - **itinerarySetCurrent, itineraryTravel, itineraryStop, itineraryResume, …**

- Still communicating, serving requests:
  - **itineraryMigrationFirst ();**

    // Do all migration first, then services, Default behavior
  - **itineraryRequestFirst ();**

    // Serving the pending requests upon arrival before migrating again

Dynamic itineraries



Destination	*Methods*
Host 1	echo
Host 2	callhome
Host 3	processData
Host 4	foo

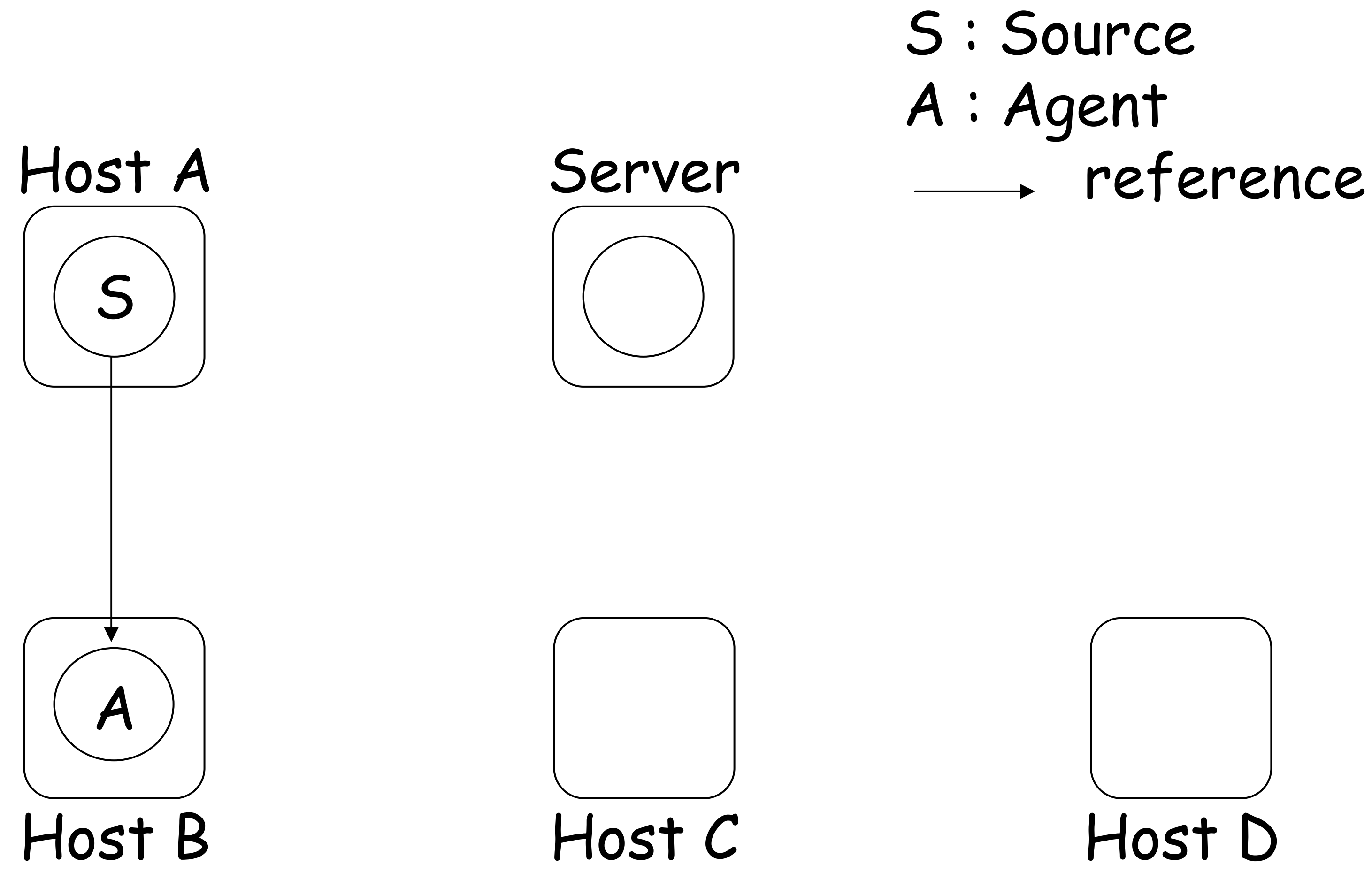
Communicating with mobile objects

- Ensuring communication in presence of migration
- Should be transparent (i.e. nothing in the application code)
- Impact on performance should be limited or well known
- ProActive provides 2 solutions to choose from at object creation
-
    - Location Server
    - Forwarders
- also, it is easy to add new ones!

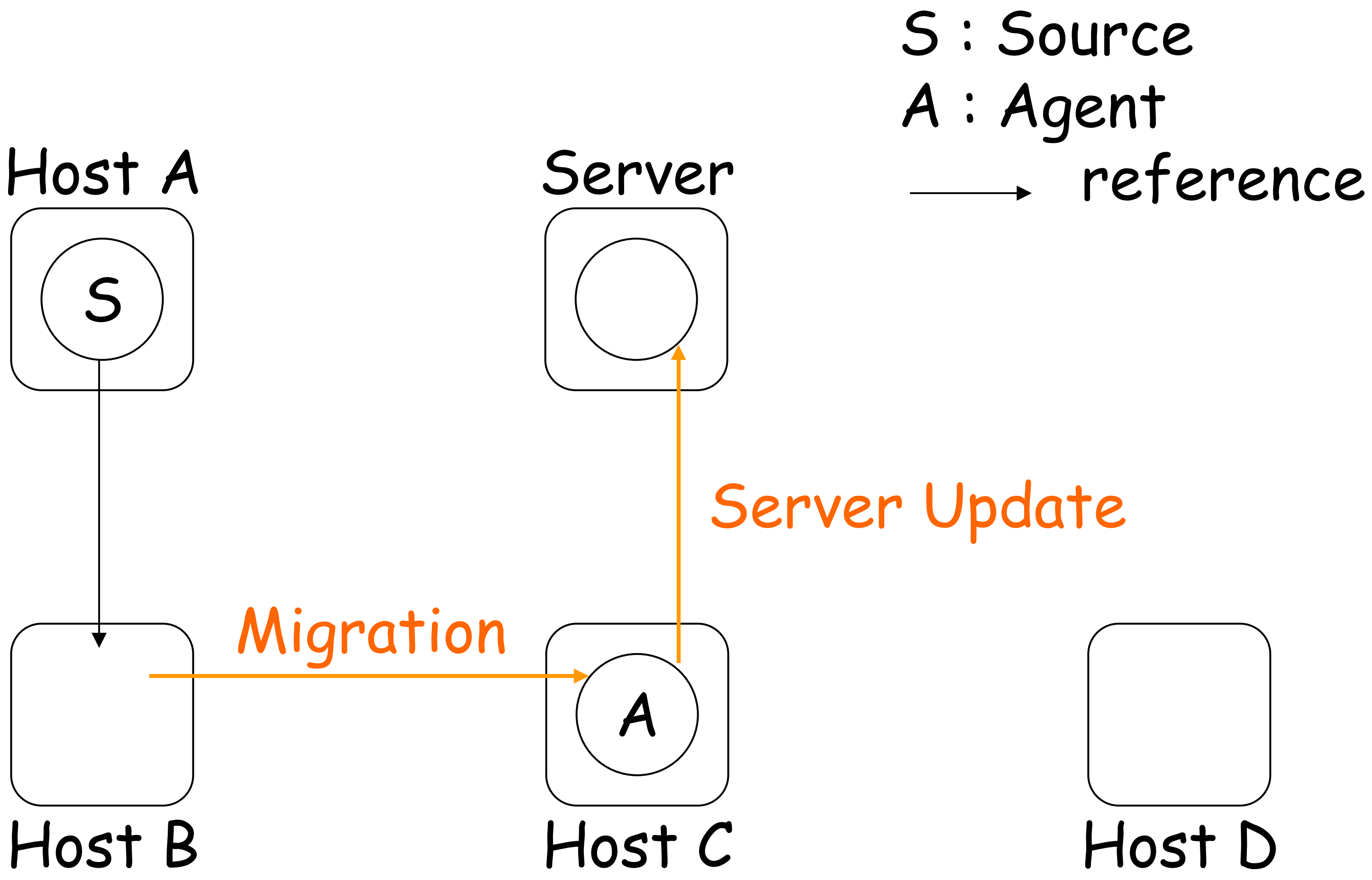
Forwarders

- Migrating object leaves forwarder on current site
- Forwarder is linked to object on remote site
  - Possibly the mobile object
  - Possibly another forwarder => a *forwarding chain* is built
- When receiving message, forwarder sends it to next hop
- Upon successful communication, a tensioning takes place

Other Strategy: Centralized (location Server)



Centralized Strategy (2)



S : Source
A : Agent
reference

Host A

Server

S

Server Update

Migration

A

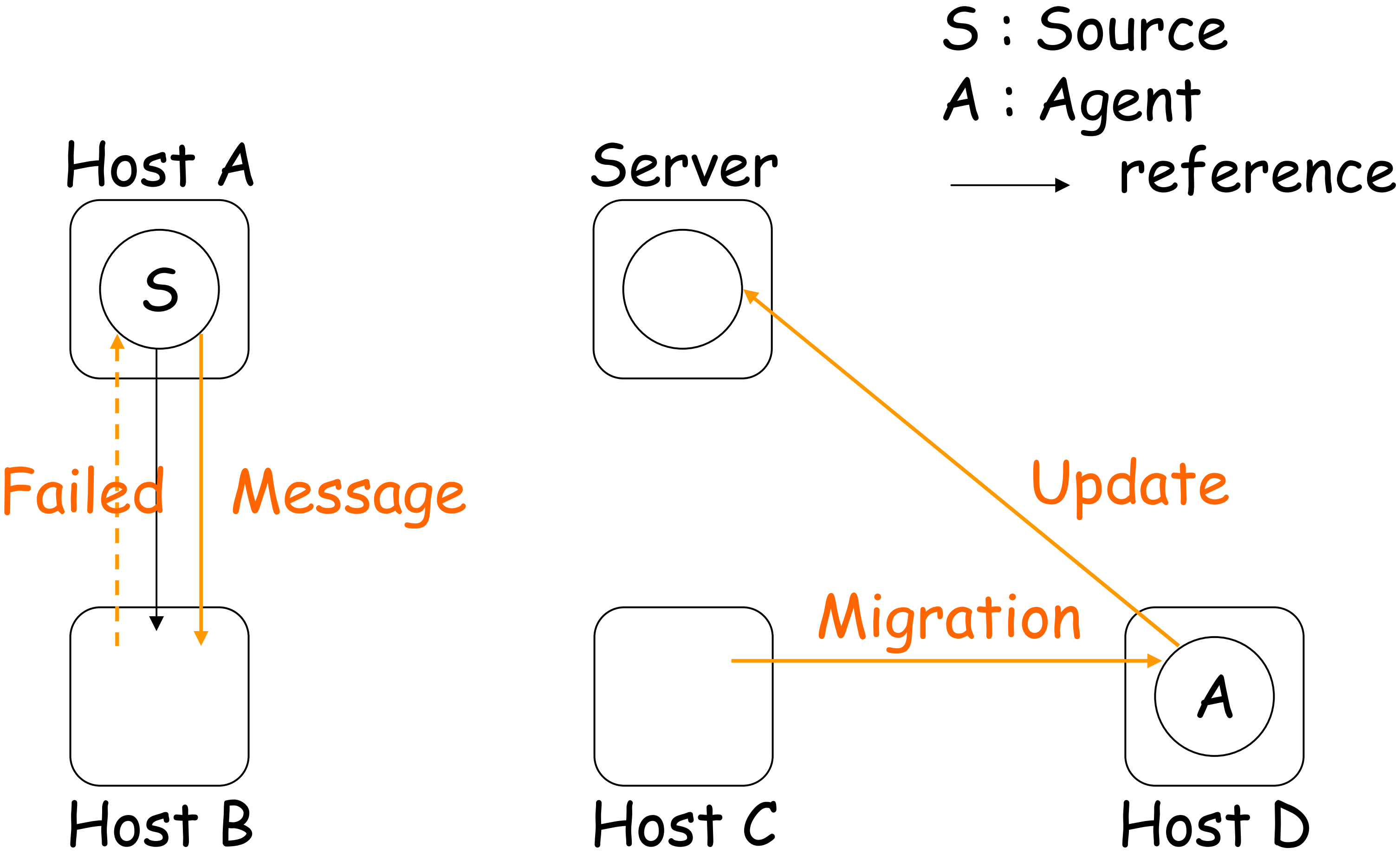
Host B

Host C

Host D

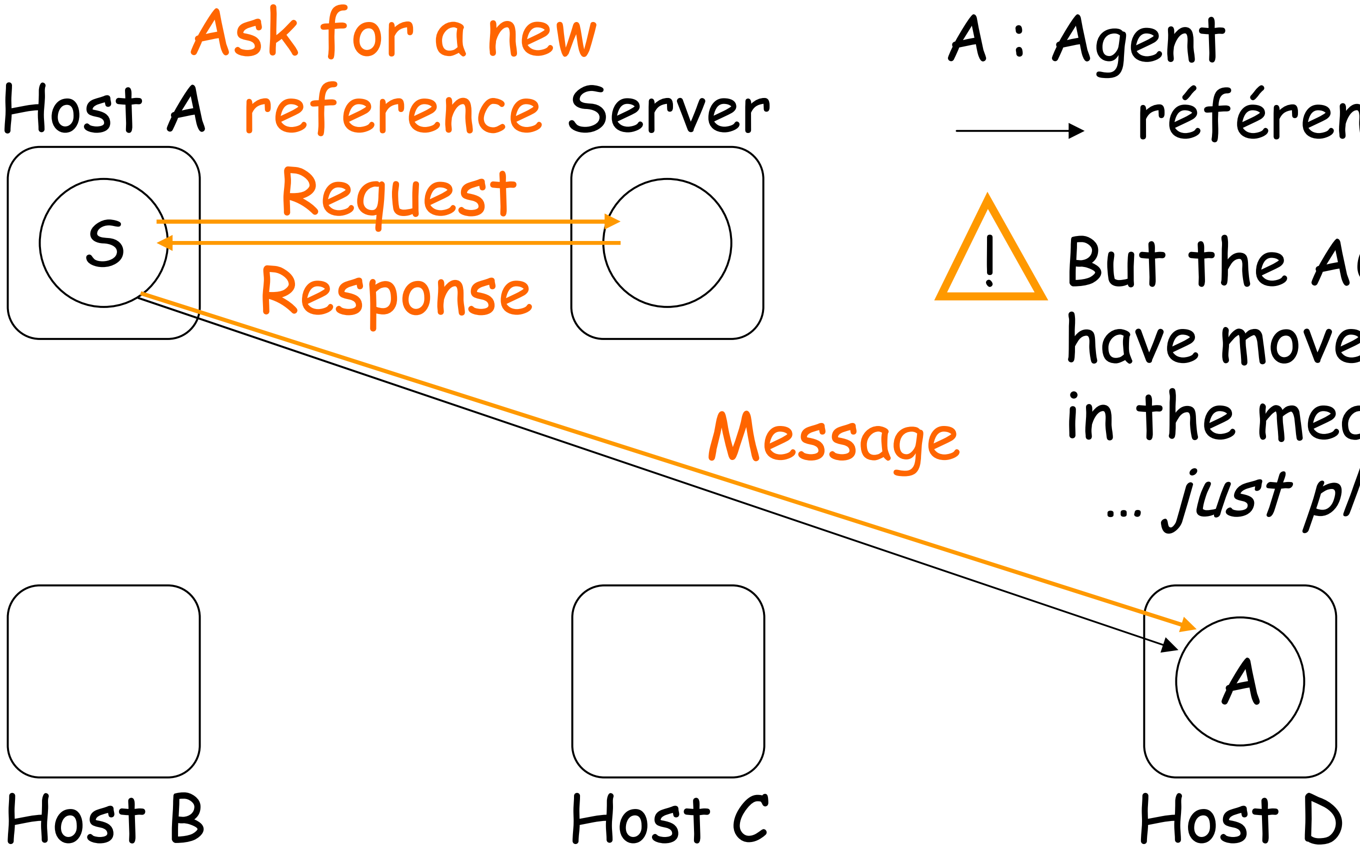
A migrating object updates the server

Centralized Strategy (3)



A migrating object updates the server

Centralized Strategy (4)



S : Source
A : Agent
→ référence

⚠ But the AO might have moved again in the meantime
… *just play again.*

The source get a new reference from the server

Location Server vs Forwarder

- Server
  - No fault tolerance if single server
  - Scaling is not straightforward
  - Added work for the mobile object
  - The agent can run away from messages

- Forwarders
  - Use resources even if not needed
  - The forwarding chain is not fault tolerant
  - An agent can be lost

- **What about performance?**

On the cost of the communication

- Server:
  - The agent must call the server => the migration is longer
  - Cost for the source:
    - Call to site where the agent was
    - Call to the server and wait for the reply
    - Call to the (maybe) correct location of the agent
- Forwarder:
  - The agent must create a forwarder (< to calling server)
  - Cost for the source:
    - Follow the forwarding chain
    - Cost of the tensioning (1 communication)

Conclusion

- Weak Migration of any active object
- Communications using two schemes: server and forwarders
- Current applications:
  - ☐ Network Administration
  - ☐ Desktop to Laptop
- Perspective: Taking the best of the forwarders and the server
  - ☐ Forwarder with limited lifetime
  - ☐ Server as a backup solution

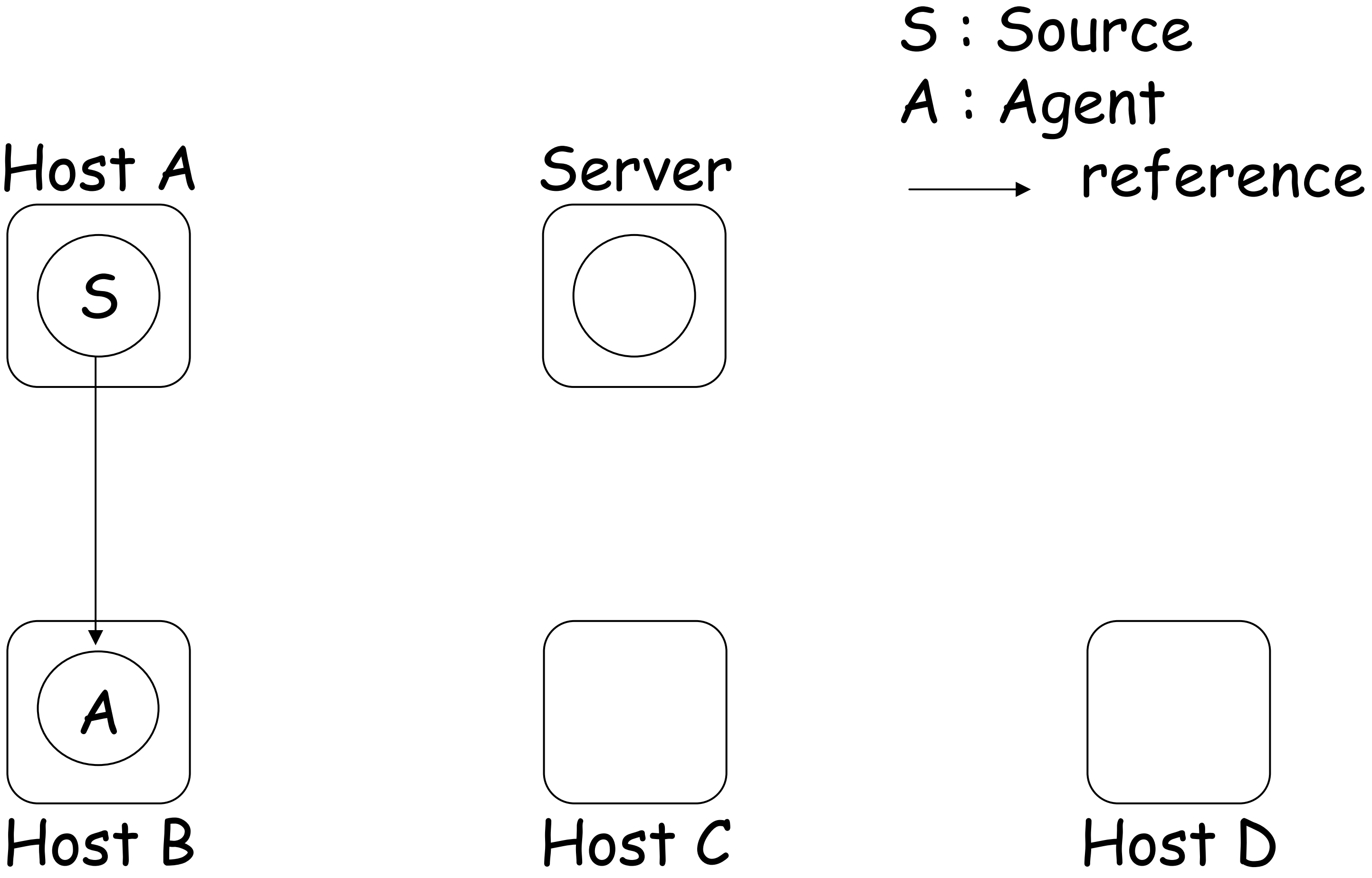
TTL-TTU mixed parameterized protocol

- TTL: Time To Live + Updating Forwarder: **5 s.**
  - After TTL, a forwarder is subject to self destruction
  - Before terminating, it updates server(s) with last agent known location

- TTU: Time To Update mobile AO:
  - After TTU, AO will inform a localization server(s) of its current location

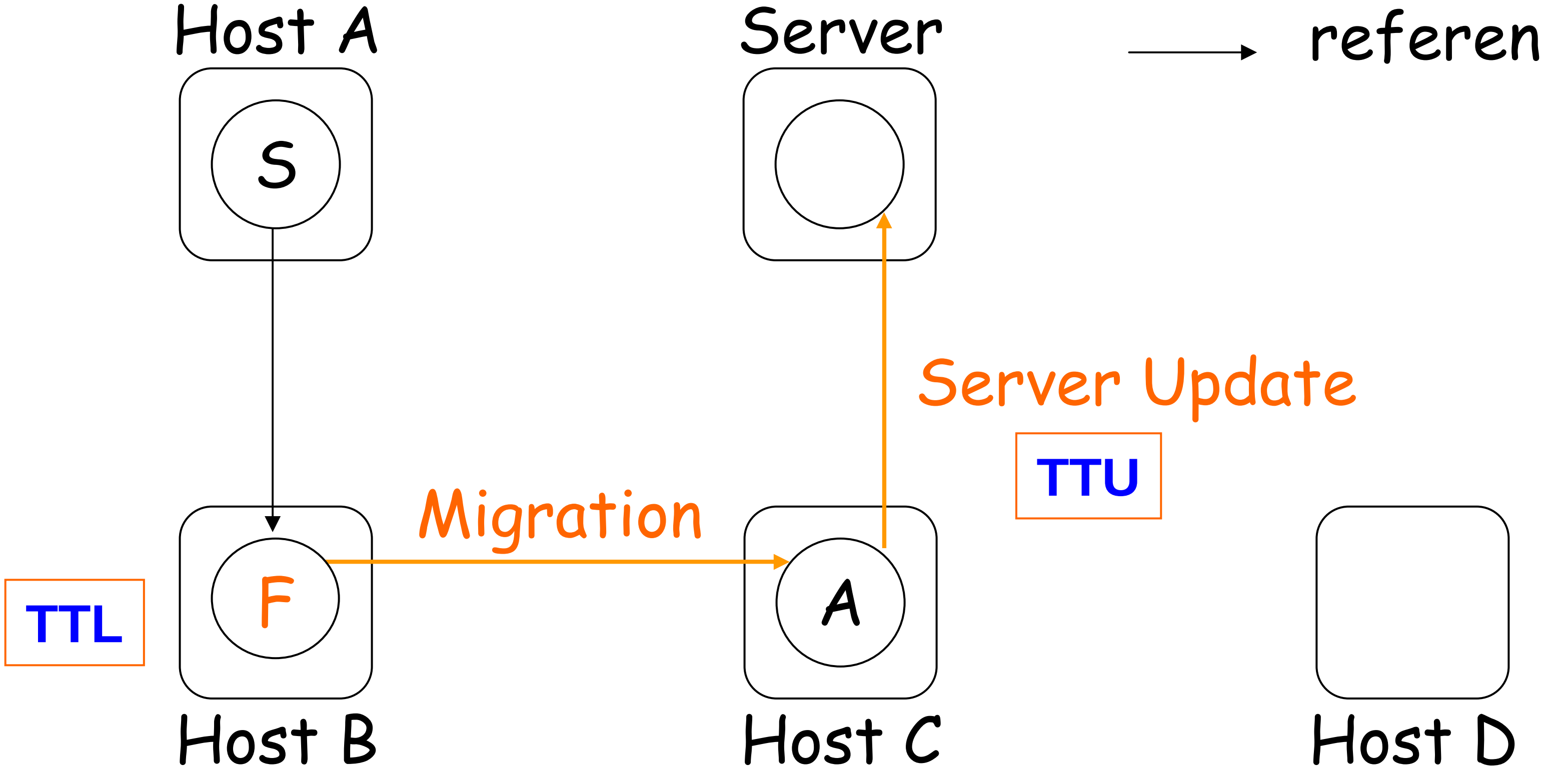
- Dual TTU: first of two events:
  - maxMigrationNb: the number of migrations without server update **10**
  - maxTimeOnSite: the time already spent on the current site **5 s.**

TTL-TTU mixed parameterized protocol



TTL-TTU mixed parameterized protocol

S : Source
A : Agent
→ reference



Conclusion on Mobile Active Objects

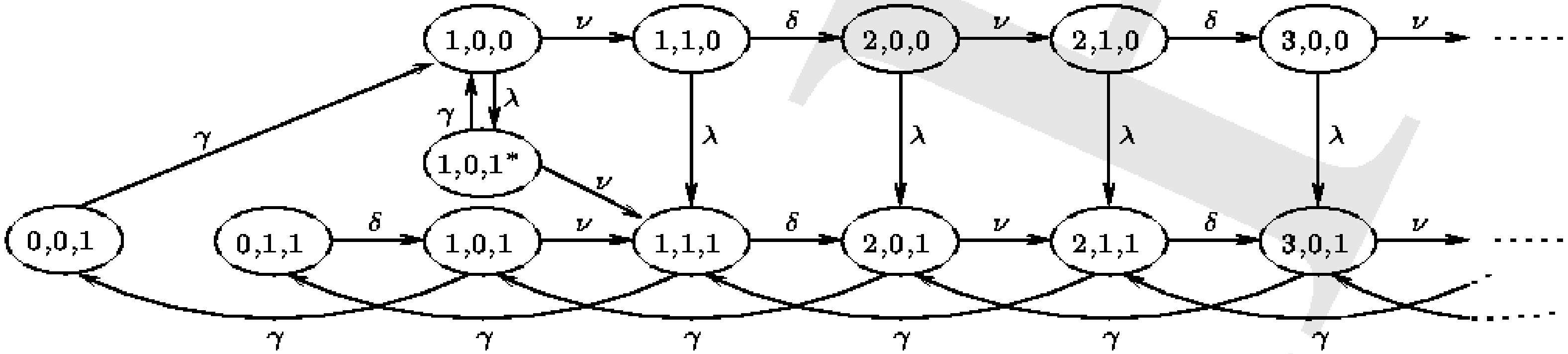
- AO = a good unit of Computational Mobility
- Weak Migration OK (even for Load Balancing)
- Both Actors and Servers

- Ensuring communications: several implementation to choose from:
  - ☐ Location Server
  - ☐ Forwarders
  - ☐ Mixed: based on TTL-TTU

- Primitive + Higher-Level abstractions:
  - ☐ migrateTo (location)
  - ☐ onArrival, onDeparture
  - ☐ Itinerary, etc.

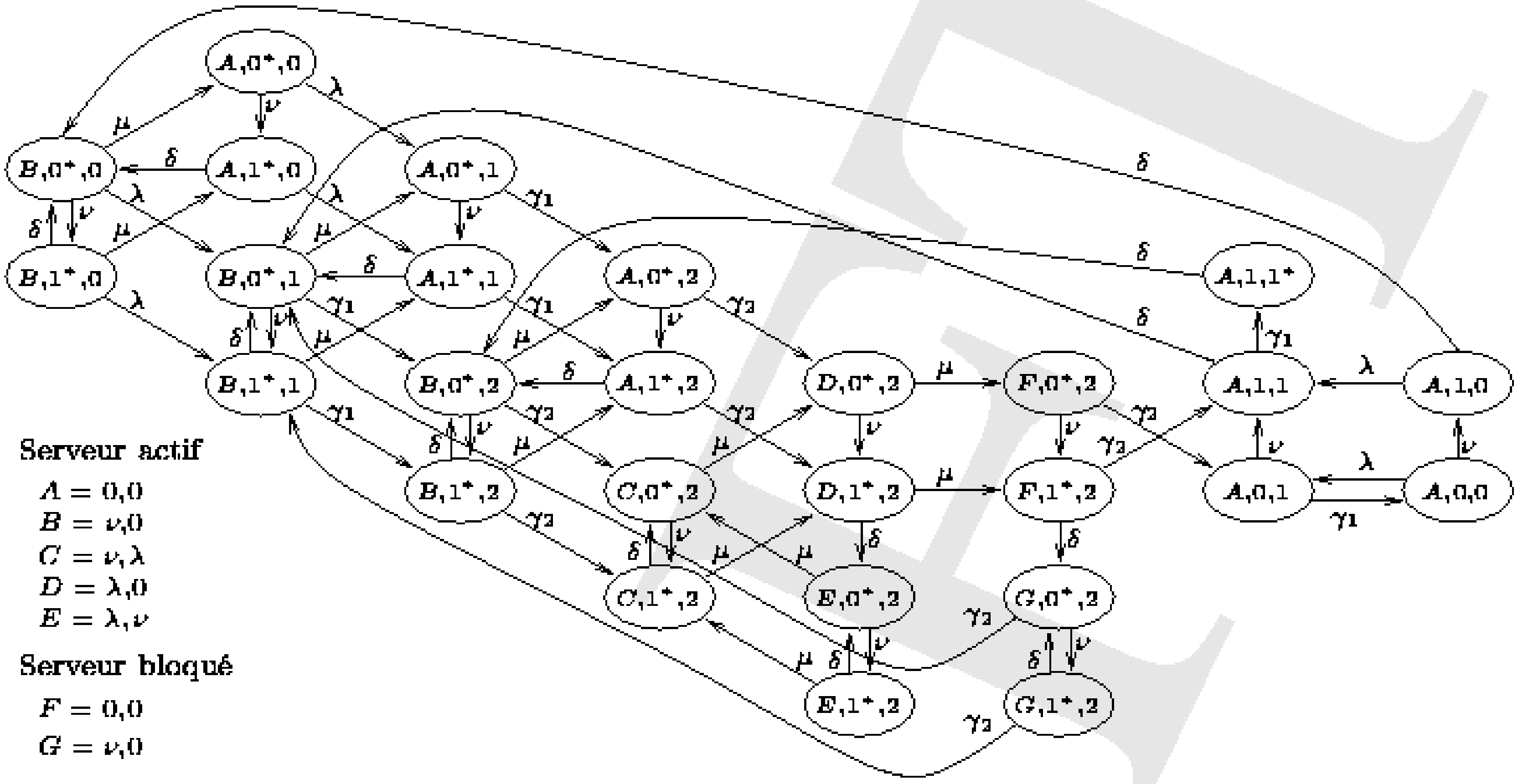
Formal Performance Evaluation of Mobile Agents: Markov Chains

- Objectives:
  - Formally study the performance of Mobile Agent localization mechanism
  - Investigate various strategies (forwarder, server, etc.)
  - Define adaptative strategies



Forwarder Strategy

Modeling of Server Strategy



Serveur actif

$A = 0,0$
$B = \nu,0$
$C = \nu,\lambda$
$D = \lambda,0$
$E = \lambda,\nu$

Serveur bloqué

$F = 0,0$
$G = \nu,0$

ProActive Core

FAULT TOLERANCE SERVICE

Fault-tolerance in ProActive

- Restart an application from latest valid checkpoint
  - Avoid cost of restarting from scratch
- Fault-tolerance is non intrusive
  - set in a deployment descriptor file
  - Fault-tolerance service attached to resources
  - No source code alteration
    - Protocol selection , Server(s) location, Checkpoint period

Fault-tolerance in ProActive

- Rollback-Recovery fault-tolerance
  - After a failure, revert the system state back to some earlier and correct version
  - Based on periodical checkpoints of the active objects
  - Stored on a stable server

- Two protocols are implemented
  - Communication Induced Checkpointing (CIC)
    - \+ Lower failure free overhead
    - – Slower recovery
  - Pessimistic Message Logging (PML)
    - – Higher failure free overhead
    - \+ Faster recovery

- Transparent and non intrusive

Built-in Fault-tolerance Server

- Fault-tolerance is based on a global server
- This server is provided by the library, with
  - Checkpoint storage
  - Failure detection
    - Detects fail-stop failures
  - Localization service
    - Returns the new location of a failed object
  - Resource management service
    - Manages a set of nodes on which restart failed objects

ProActive Core

SECURITY SERVICE

ProActive Security Framework

Issue

Access control, communication privacy and integrity

- Unique features
  - SPKI: Hierarchy of certificates
  - No security related code in the application source code
  - Declarative security language
  - Security at user- and administrator-level
  - Security context dynamic propagation
- Configured within deployment descriptors
  - Easy to adapt according the actual deployment

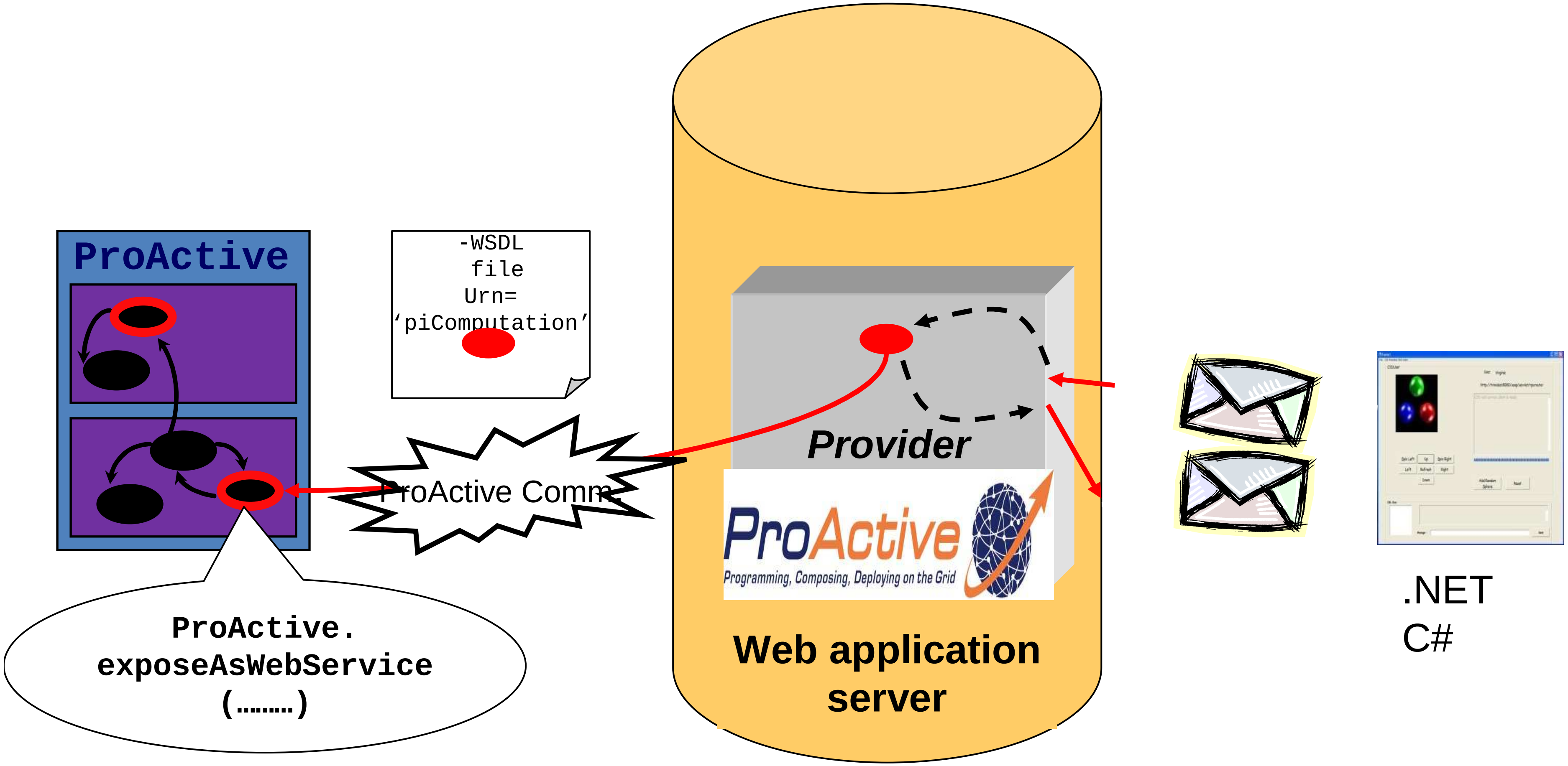
ProActive Core

WEB SERVICES

Web Service Integration

- Aim
  - Turn active objects and components interfaces into Web Services

    ➔ interoperability with any foreign language or any foreign technology.
- API
  - Expose an active object as a web Service (the user can choose the methods he wants to expose)
    - **`exposeAsWebService(Object o, String url, String urn, String [] methods );`**
  - Expose component's interfaces as web services
    - **`exposeComponentAsWebService(Component component, String url, String componentName );`**



ProActive

-WSDL
file
Urn=
‘piComputation’

ProActive Comm.

Provider

Web application server

.NET
C#

`ProActive.`
`exposeAsWebService`
`(………)`

1. ProActive.exposeAsWebService ()

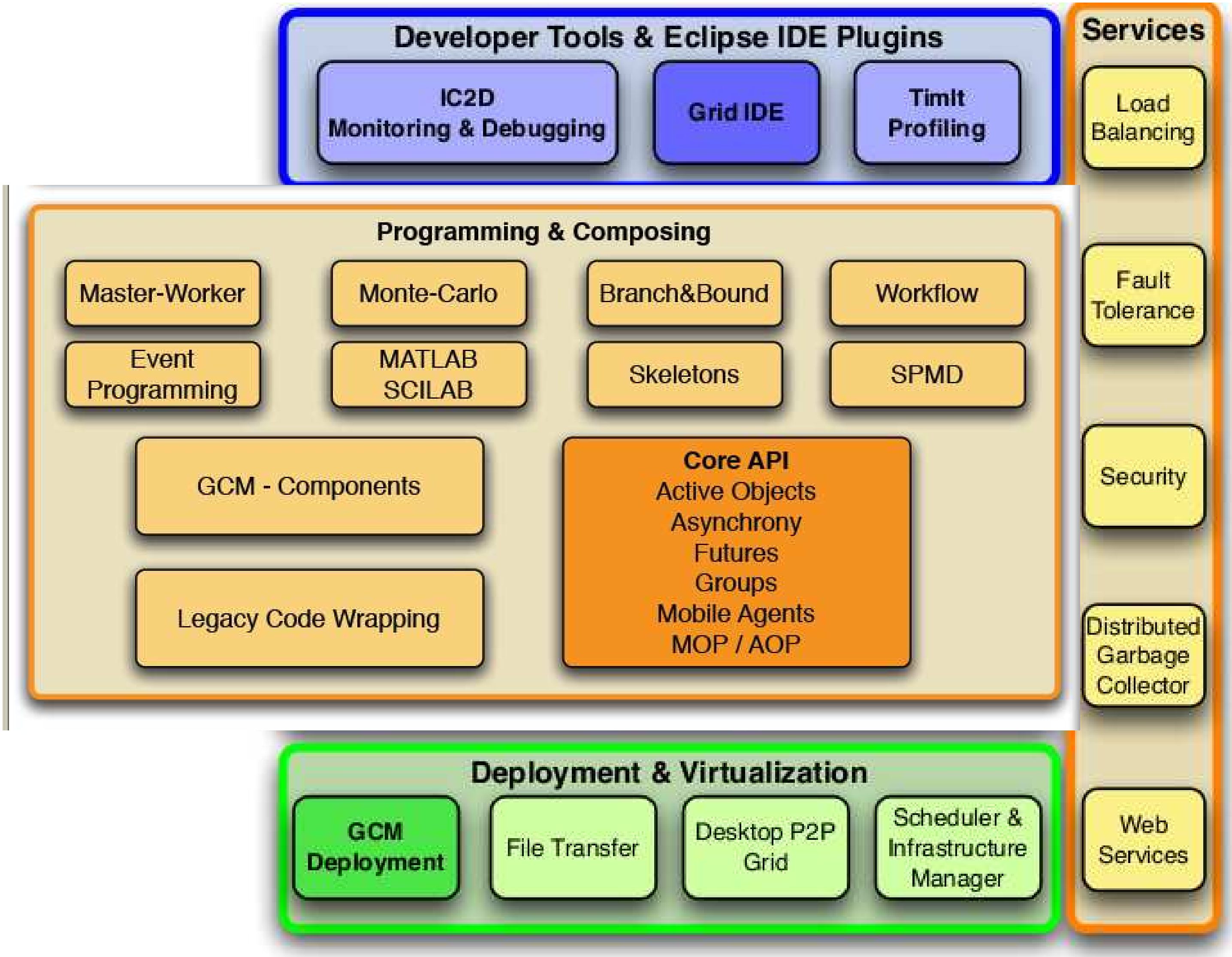
2. Deployment

3. Client Call

Agenda

- ProActive and ProActive Parallel Suite
- Programming and Composing
  - ProActive Core
  - High Level Programming models
  - ProActive Components
- Deployment Framework
- Development Tools

ProActive Parallel Suite



Developer Tools & Eclipse IDE Plugins

- **IC2D Monitoring & Debugging**
- **Grid IDE**
- **TimIt Profiling**

Programming & Composing

- Master-Worker
- Monte-Carlo
- Branch&Bound
- Workflow
- Event Programming
- MATLAB SCILAB
- Skeletons
- SPMD
- GCM - Components
- Legacy Code Wrapping

Core API
Active Objects
Asynchrony
Futures
Groups
Mobile Agents
MOP / AOP

Deployment & Virtualization

- **GCM Deployment**
- File Transfer
- Desktop P2P Grid
- Scheduler & Infrastructure Manager

Services

- Load Balancing
- Fault Tolerance
- Security
- Distributed Garbage Collector
- Web Services

High Level Programming models

Master-Worker Framework

Motivations

- Embarrassingly parallel problems : simple and frequent model

- Write embarrassingly parallel applications with ProActive :

  - May require a sensible amount of code (fault-tolerance, load-balancing, …).

  - Requires understanding of ProActive concepts (Futures, Stubs, Group Communication)

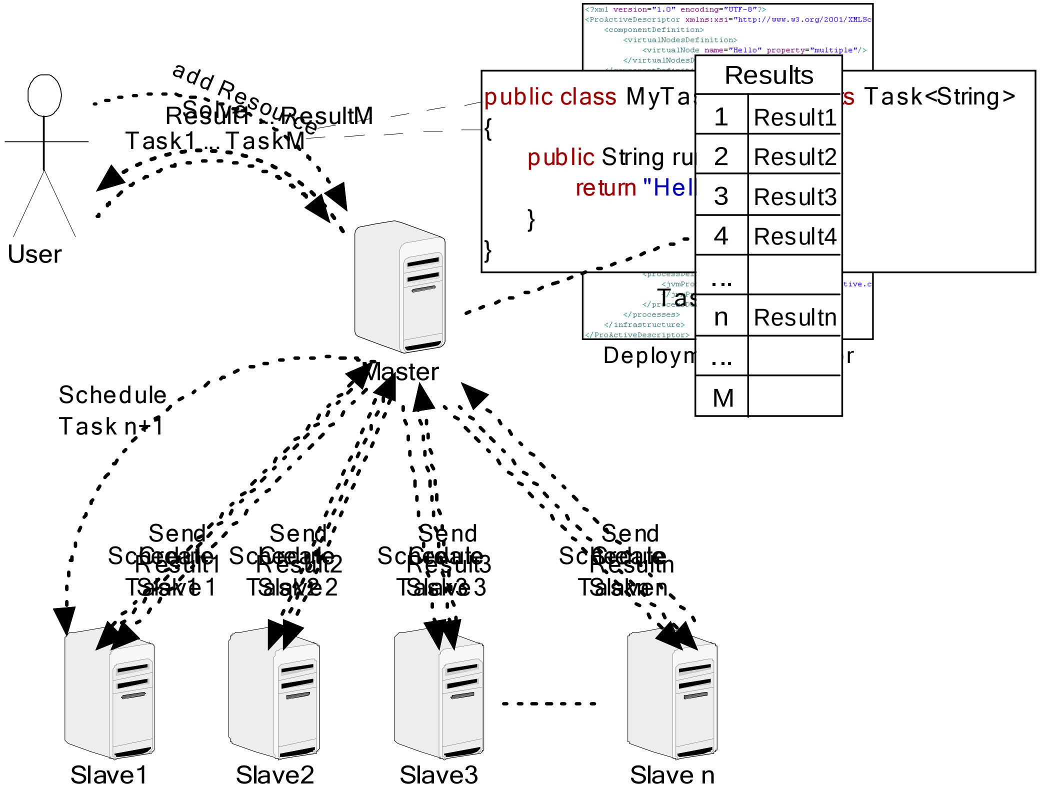
Goals of the M/W API

- Provide a easy-to use framework for solving embarrassingly parallel problems:
  - ❑ Simple Task definition
  - ❑ Simple API interface (few methods)
  - ❑ Simple & efficient solution gathering mechanism

- Provide automatic fault-tolerance and load-balancing mechanism

- Hide ProActive concepts from the user

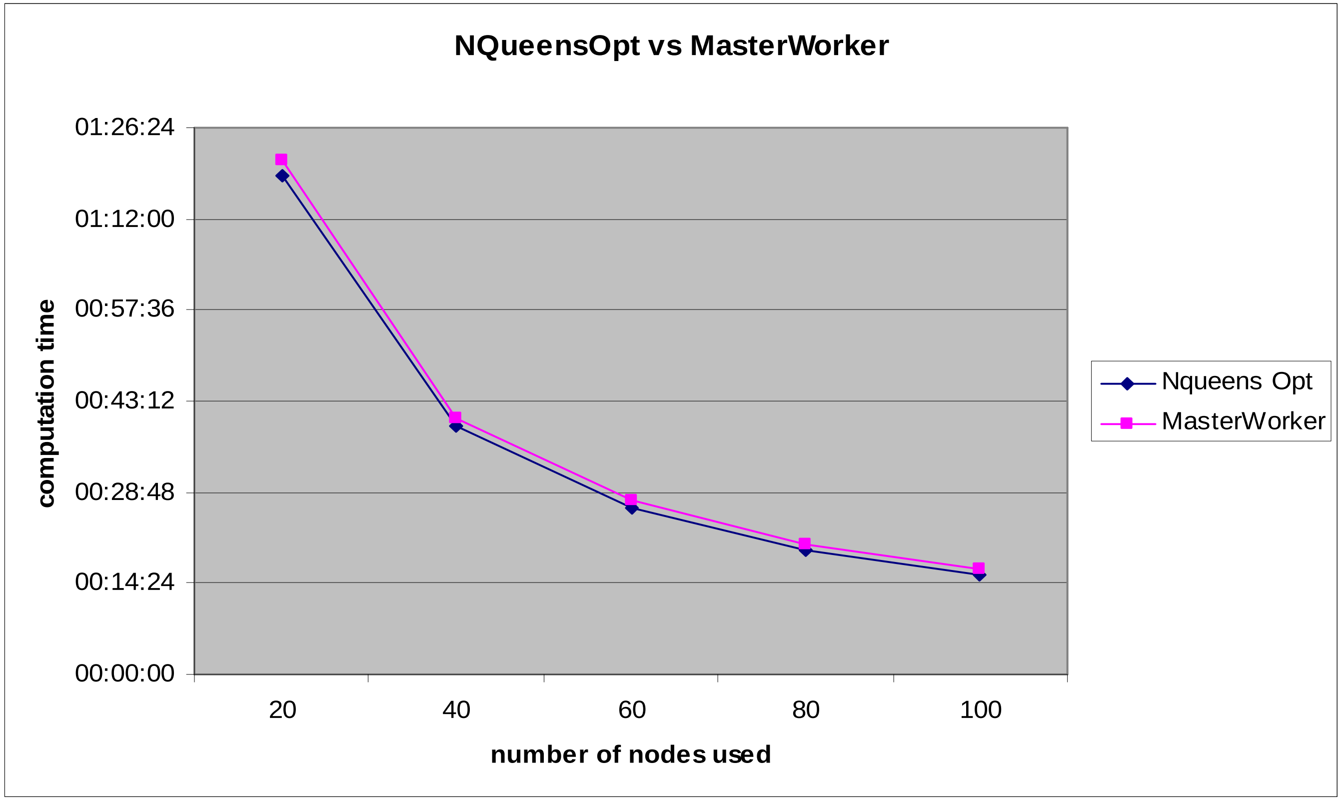
How does it work?



Results	
1	Result1
2	Result2
3	Result3
4	Result4
...	
n	Resultn
...	
M	

Comparison between specific implementation and M/W

- Experiments with nQueens problem
- Runs up to 25 nodes

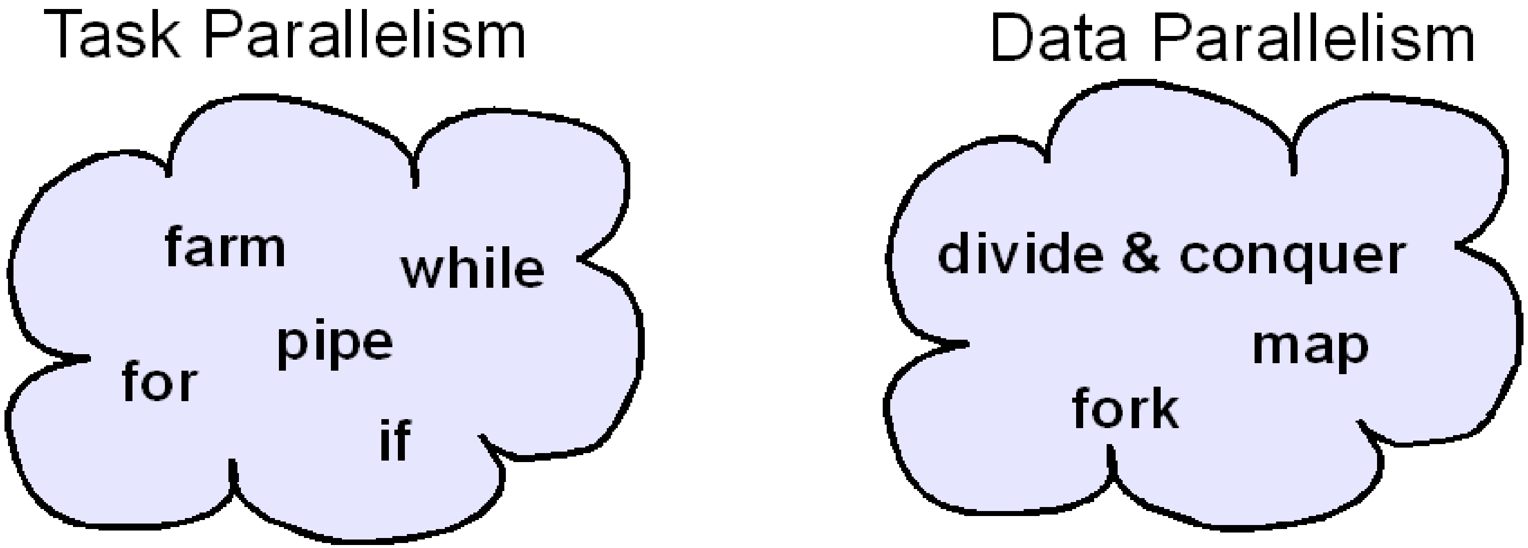


High Level Programming models

Skeletons Framework

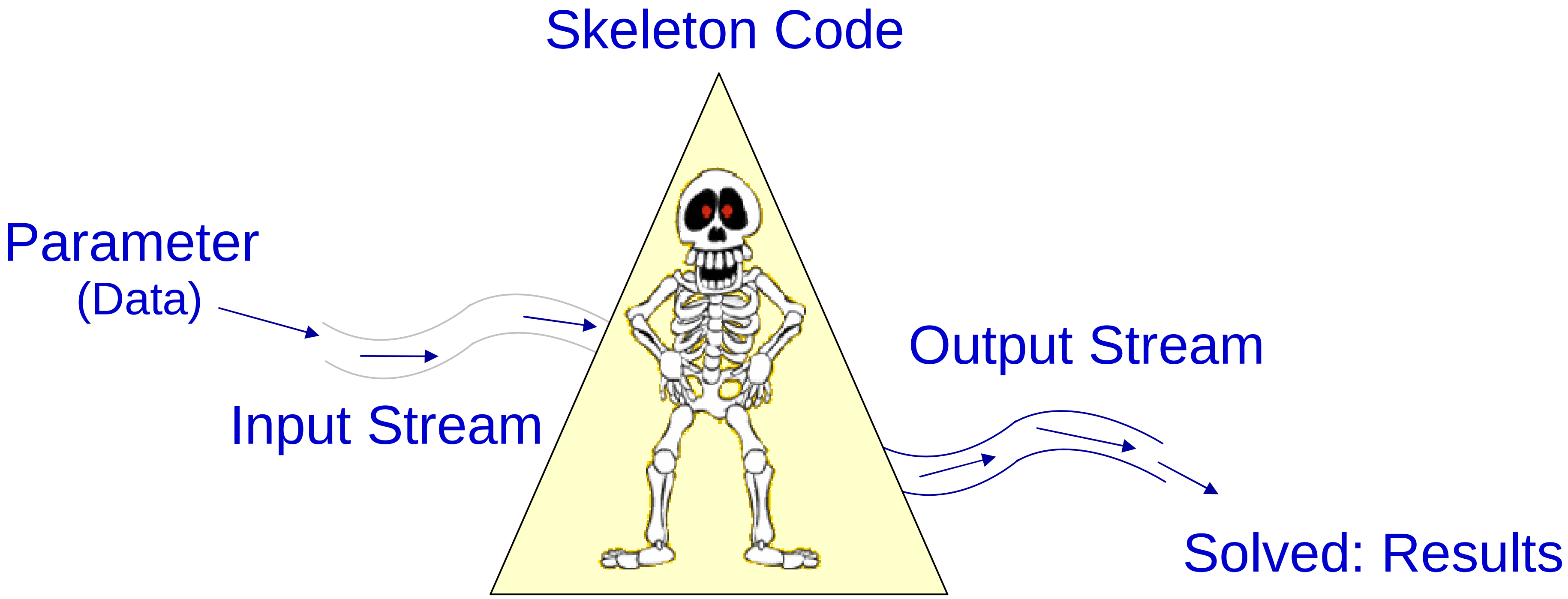
Algorithmic Skeletons

- High Level Programming Model
- Hides the complexity of parallel/distributed programming.
- Exploits nestable parallelism patterns



Skeletons Big Picture

- ►Parameters/Results are passed through streams
- ►Streams are used to connect skeletons (CODE)



Skeleton Code

Parameter
(Data)

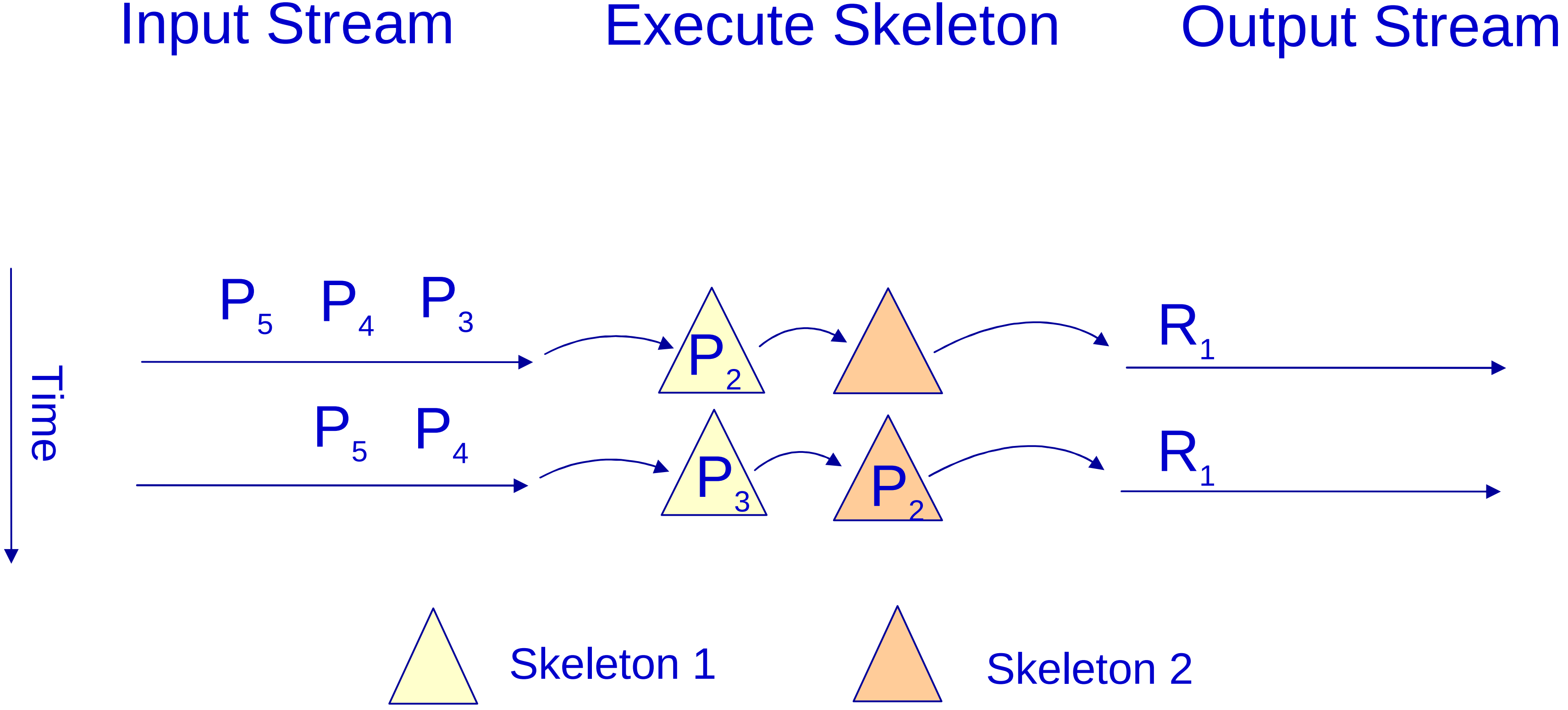
Input Stream

Output Stream

Solved: Results

Pipe Skeleton

- ►Represents computation by stages.
- ►Stages are computed in parallel for different parameters.



Input Stream Execute Skeleton Output Stream

Simple use of Pipe skeleton

```
Skeleton<Eggs, Mix>        stage1 =
                     new Seq<Eggs,Mix>(new Apprentice());

Skeleton<Mix, Omelette> stage2 =
                     new Seq<Mix,Omelette>(new Chef());

Skeleton<Eggs, Omelette> kitchen =
                    new Pipe<Eggs, Omelette>(stage1, stage2);
```

High Level Programming models

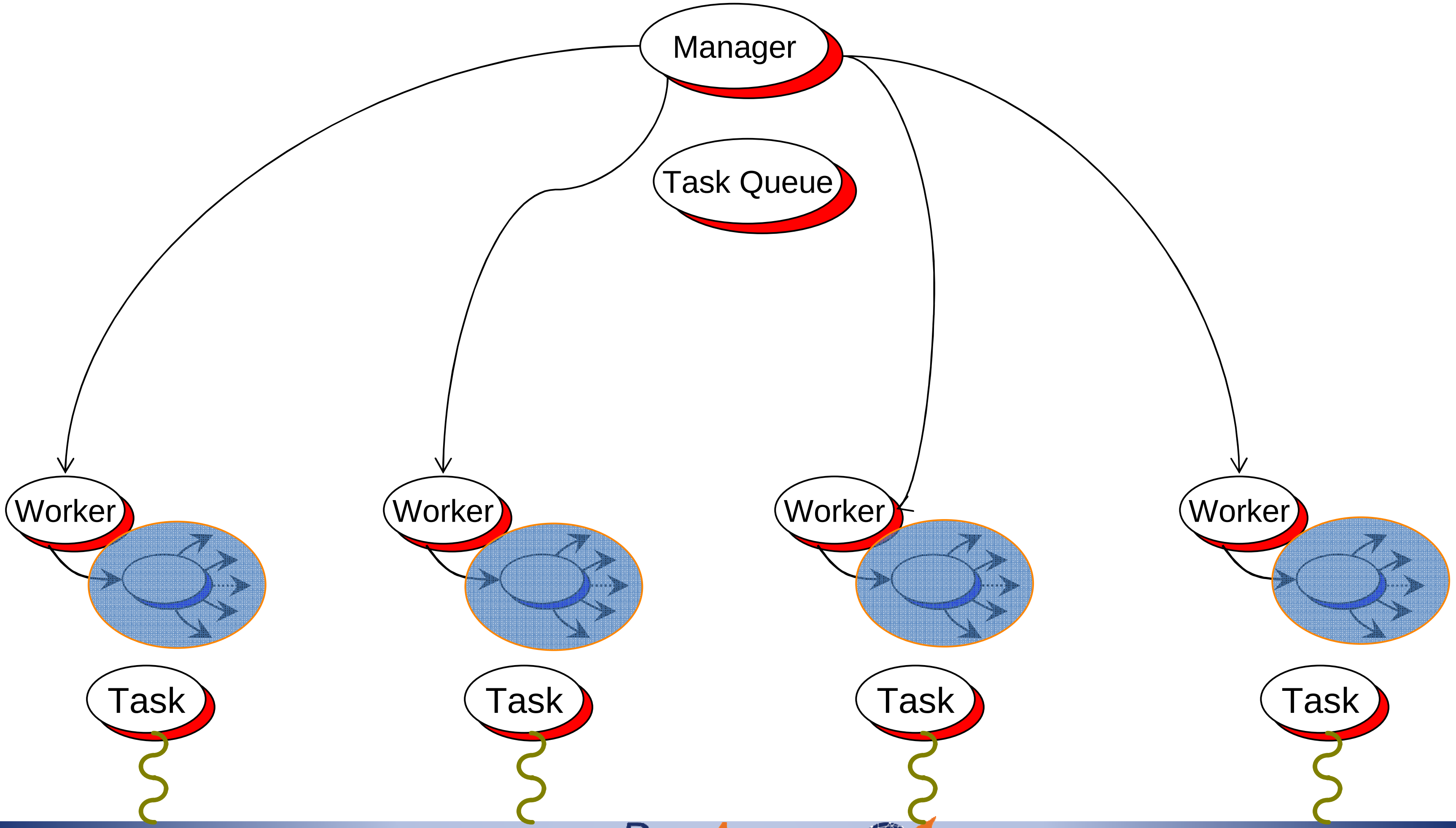
Branch-and-Bound Framework

Branch & Bound API (BnB)

- Provide a high level programming model for solving BnB problems:
  - manages task distribution and provides task communications

- Features:
  - Dynamic task split
  - Automatic result gather
  - Broadcasting best current result
  - Automatic backup (configurable)

Global Architecture : M/W + Full connectivity



High Level Programming models

OO SPMD

Object-Oriented
Single Program Multiple Data

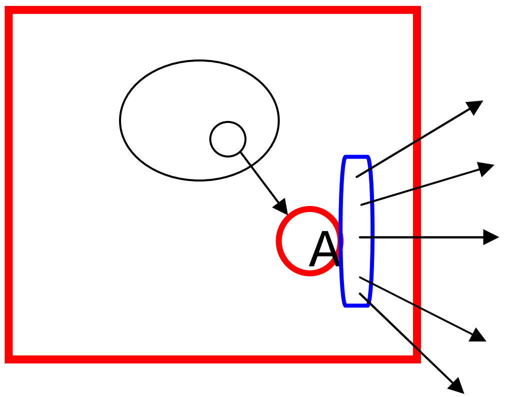
- Motivation
  - Cluster / GRID computing
  - SPMD programming for many numerical simulations
  - Use enterprise technology (Java, Eclipse, etc.) for Parallel Computing
- Able to express most of MPI’s
  - Collective Communications (broadcast, gathercast, scattercast,..)
  - Barriers
  - Topologies

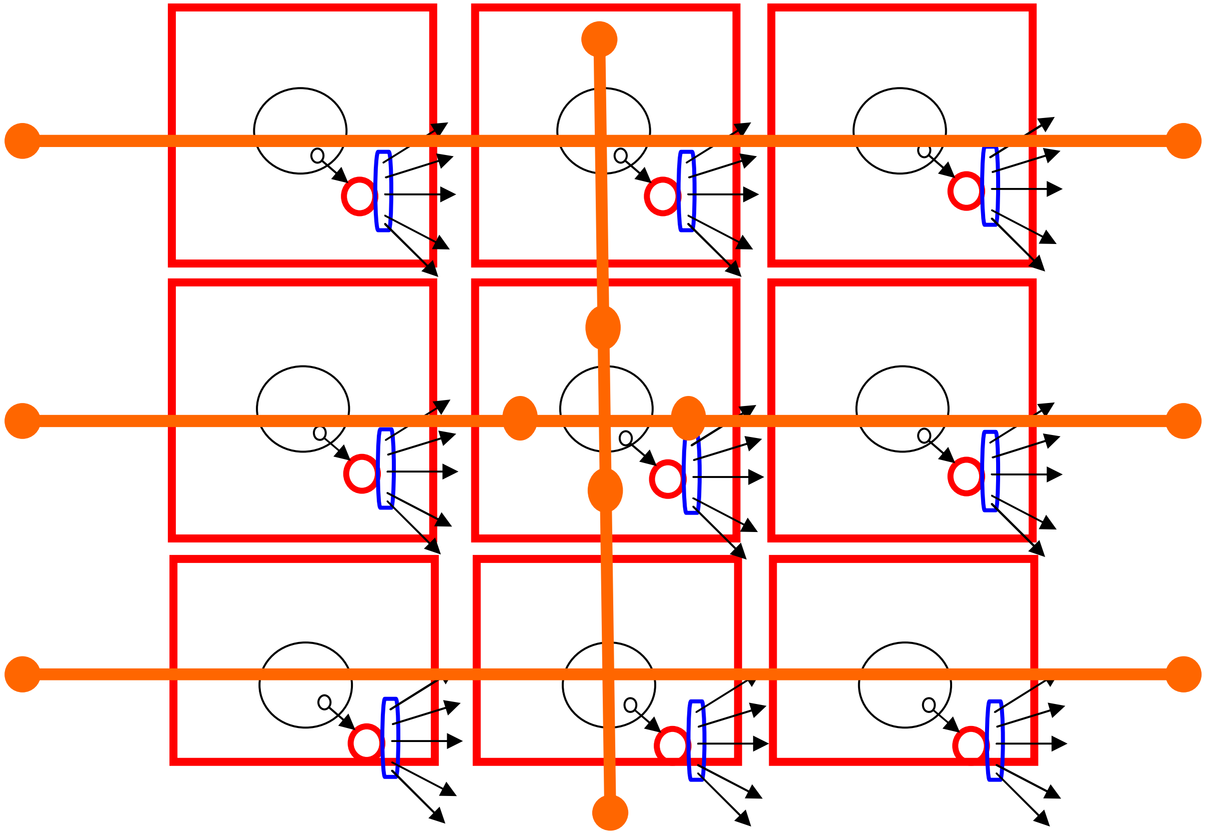
ProActive OO SPMD

- A simple communication model
  - Small API
  - No “Receive” but data flow synchronization
  - No message passing but RPC (RMI)
  - User defined data structure (Objects)
  - SPMD groups are dynamics
  - Efficient and dedicated barriers

Execution example

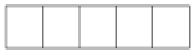
```
A ag = newSPMDGroup ("A", […], VirtualNode)
                    // In each member
    myGroup.barrier ("2D"); // Global Barrier
    myGroup.barrier ("vertical"); // Any Barrier
    myGroup.barrier ("north","south","east","west");
```

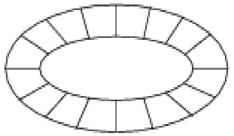


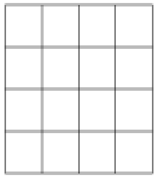


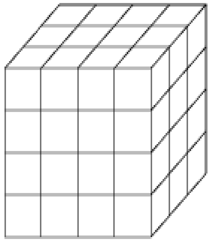
Topologies

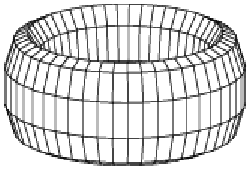
- Topologies are typed groups
- Customizable
- Define neighborhood

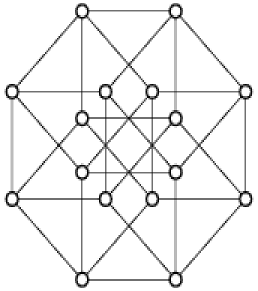

Line


Ring


Plan


Cube


Torus


Hypercube

```
Plan plan = new Plan(groupA, Dimensions);
Line line = plan.getLine(0);
```

MPI Communication primitives

- For some (historical) reasons, MPI has many com. Primitives:

MPI_Send	Std	MPI_Recv	Receive
MPI_Ssend	Synchronous	MPI_Irecv	Immediate
MPI_Bsend	Buffer	… (any) source, (any) tag,	
MPI_Rsend	Ready		
MPI_Isend	Immediate, async/future		
MPI_Ibsend, …			

- **I'd rather put the burden on the implementation, not the Programmers !**
  - **How to do adaptive implementation in that context ?**
- Not talking about:
  - the combinatory that occurs between send and receive
  - the semantic problems that occur in distributed implementations

- Is Recv at all needed ? (Dynamic Control of Message Asynchrony)

MPI and Threads

- MPI was designed at a different time
- When OS, languages (e.g. Fortran) were single-threaded
  - No longer the case.
  - Programmers can write more simple, ”sequential” code,
- the implementation, the middleware, can execute things in parallel.

Main MPI problems for the GRID

- Too static in design
- Too complex in Interface (API)
- Too many specific primitives to be adaptive
- Type Less
    - … and you do not ’’lamboot’’ / ’’lamhalt’’ the GRID !

Performance & Productivity

- HPC vs. HPC:

High Performance Computing

vs.

High Productivity Computing

Sum up: MPI vs. ProActive OO SPMD

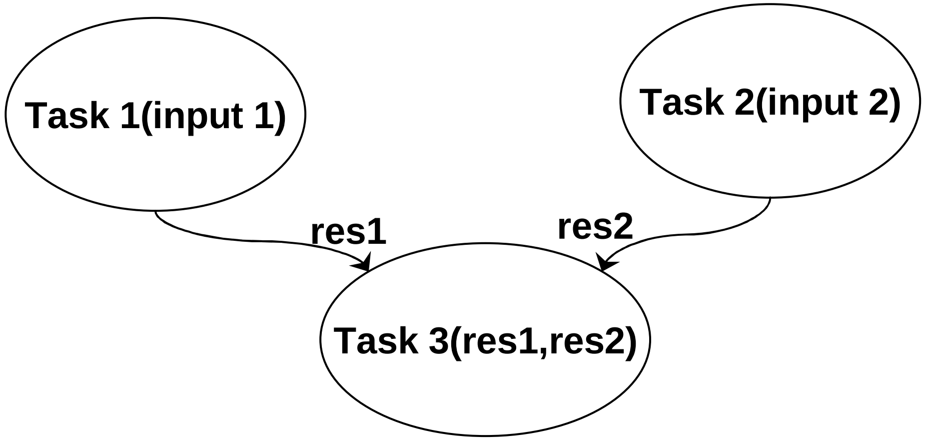
- A simple communication model, with simple communication primitive(s):
  - No RECEIVE but data flow synchronization
  - Adaptive implementations are possible for:
    - » // machines, Cluster, Desktop, etc.,
    - » Physical network, LAN, WAN, and network conditions
    - » Application behavior
- **Typed Method Calls:**
  - **==> Towards Components**
- Reuse and composition:
  - No main loop, but asynchronous calls to myself

High Level Programming models

Scheduling

Programming with flows of tasks

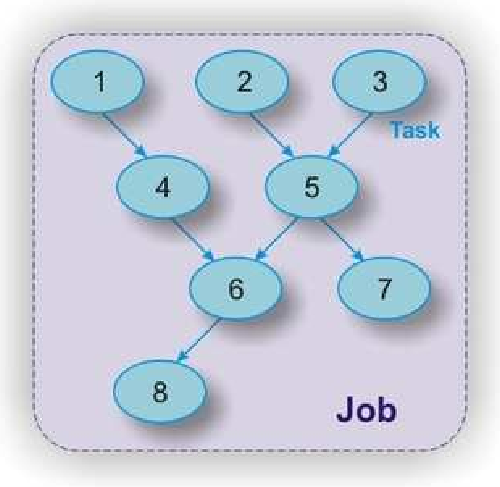
- Program an application as an ordered tasks set
  - **Logical flow** : Tasks execution are orchestrated
  - **Data flow** : Results are forwarded from ancestor tasks to their children as parameter



- The task is the smallest execution unit
- Two types of tasks:
  - Standard Java
  - Native, i.e. any third party application

Defining and running jobs with ProActive

- A workflow application is a job
  - a set of tasks which can be executed according to a dependency tree
- Rely on ProActive Scheduler only
- Java or XML interface
  - Dynamic job creation in **Java**
  - Static description in **XML**

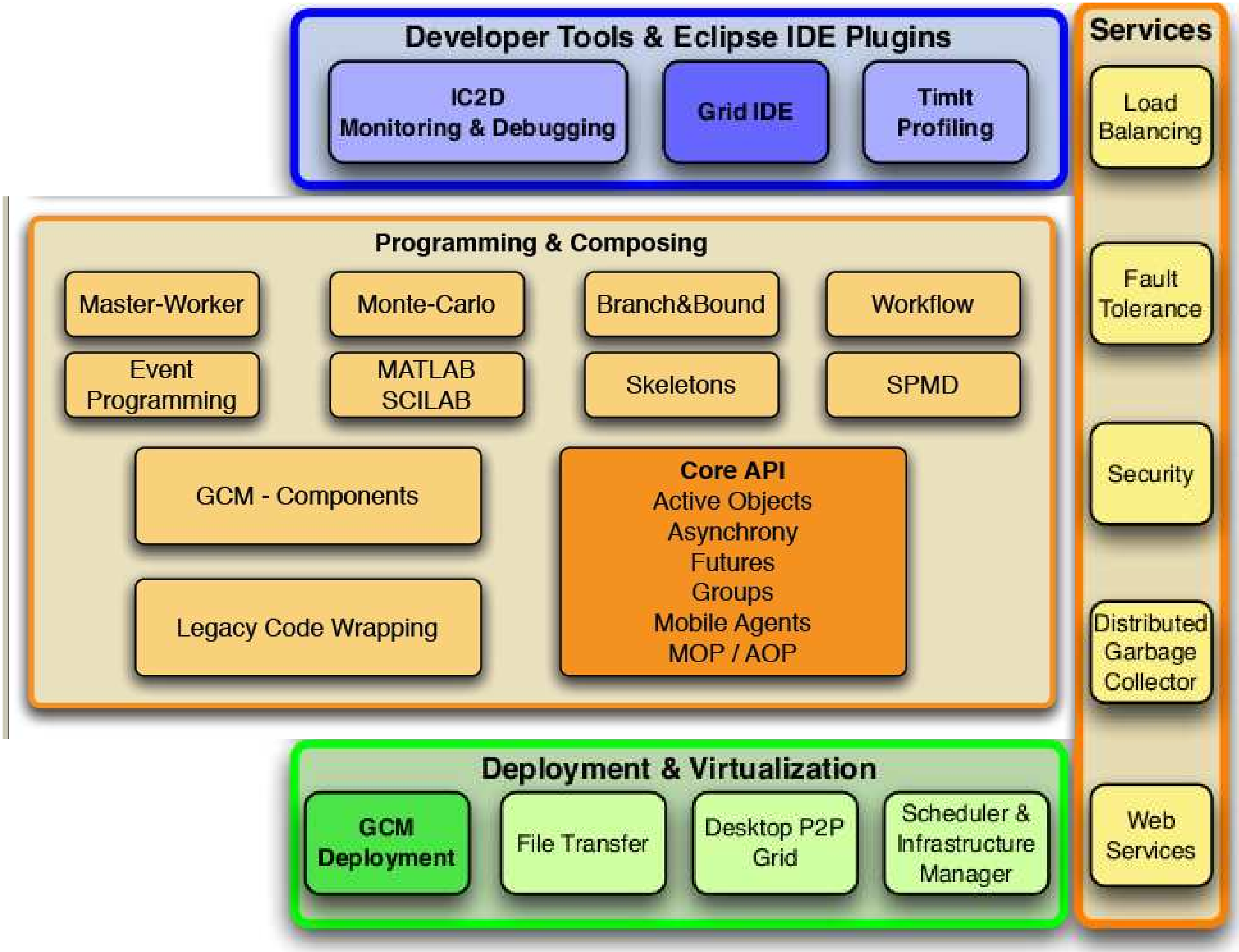


- Task failures are handled by the ProActive Scheduler
  - A task can be automatically re-started or not (with a user-defined bound)
  - Dependant tasks can be aborted or not
  - The finished job contains the cause exceptions as results if any

Agenda

- ProActive and ProActive Parallel Suite
- Programming and Composing
  - ProActive Core
  - High Level Programming models
  - ProActive Components
- Deployment Framework
- Development Tools

ProActive Parallel Suite



Developer Tools & Eclipse IDE Plugins

IC2D Monitoring & Debugging

Grid IDE

TimIt Profiling

Programming & Composing

Master-Worker

Monte-Carlo

Branch&Bound

Workflow

Event Programming

MATLAB SCILAB

Skeletons

SPMD

GCM - Components

Legacy Code Wrapping

Core API
Active Objects
Asynchrony
Futures
Groups
Mobile Agents
MOP / AOP

Deployment & Virtualization

GCM Deployment

File Transfer

Desktop P2P Grid

Scheduler & Infrastructure Manager

Services

Load Balancing

Fault Tolerance

Security

Distributed Garbage Collector

Web Services

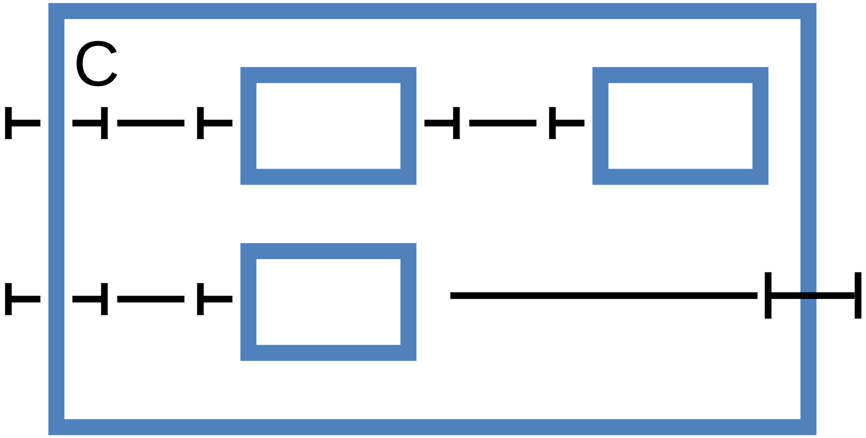
A framework for Grid components

- Facilitating the design and implementation of complex distributed systems
- Leveraging the ProActive library
  ProActive components benefit from underlying features
- Allowing reuse of legacy components (e.g. MPI)
- Providing tools for defining, assembling and monitoring distributed components

Component - What is it ?

- A component in a given infrastructure is:

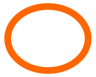
a software module,
with a standardized description of what it needs and provides,
to be manipulated by tools for Composition and Deployment



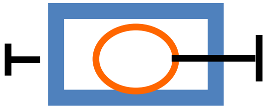
ProActive Component Definition

- A component is:
  - Formed from one (or several) Active Object
  - Executing on one (or several) JVM
  - Provides a set of server ports: Java Interfaces
  - Uses a set of client ports: Java Attributes
  - Point-to-point or Group communication between components
- Hierarchical:
  - Primitive component: define with Java code and a descriptor
  - Composite component: composition of primitive + composite
  - Parallel component: multicast of calls in composites
- Descriptor:
  - XML definition of primitive and composite (ADL)
  - Virtual nodes capture the deployment capacities and needs
- Virtual Node:
  - a very important abstraction for GRID components

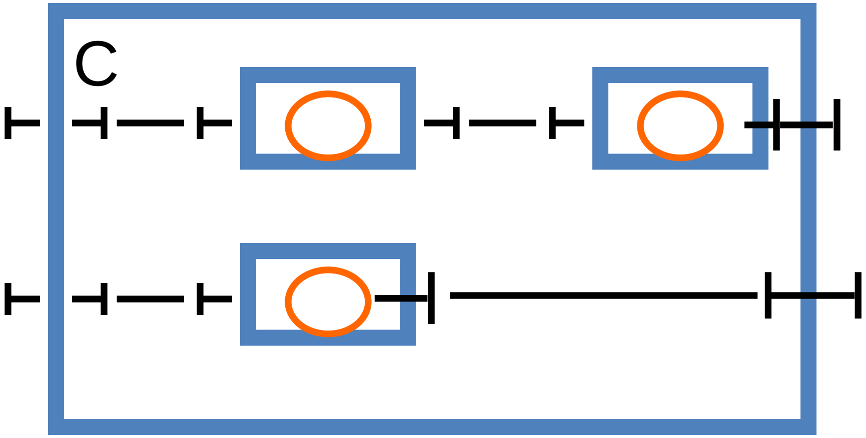
Components for the GRID



An activity, a process, …
potentially in its own JVM



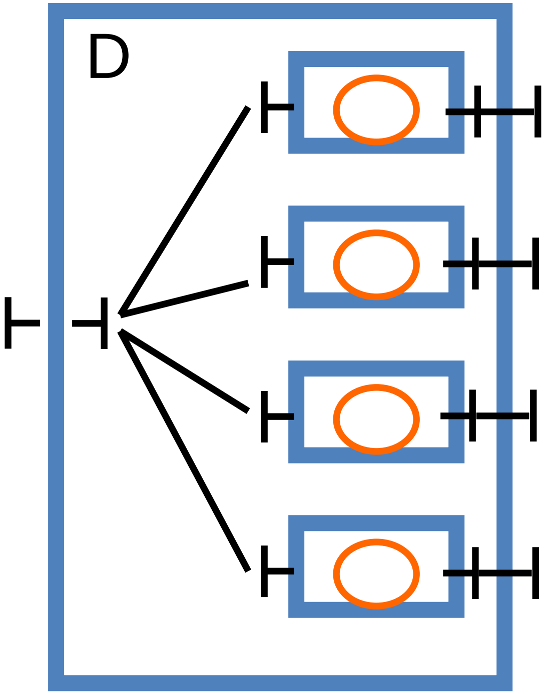
1. Primitive component



2. Composite component

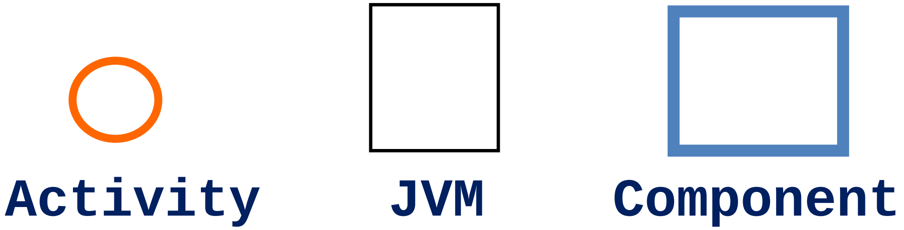
Composite: Hierarchical, and
Distributed over machines

Parallel: Composite
+ Broadcast (group)



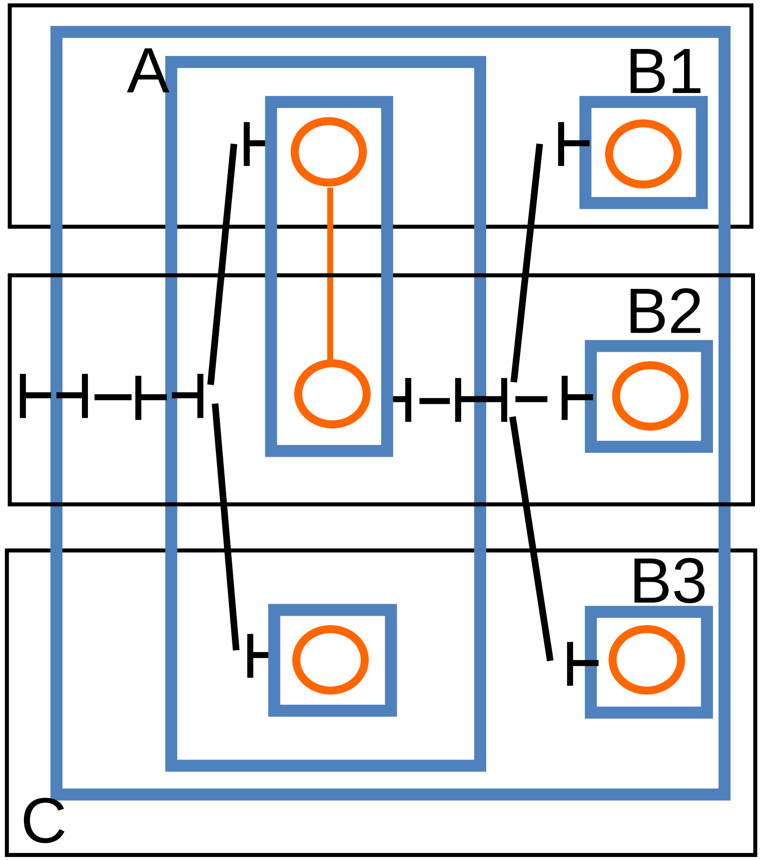
3. Parallel and composite component

Components vs. Activity and JVMs



- Components are orthogonal to activities and JVMs
    - They contain activities, span across several JVMs

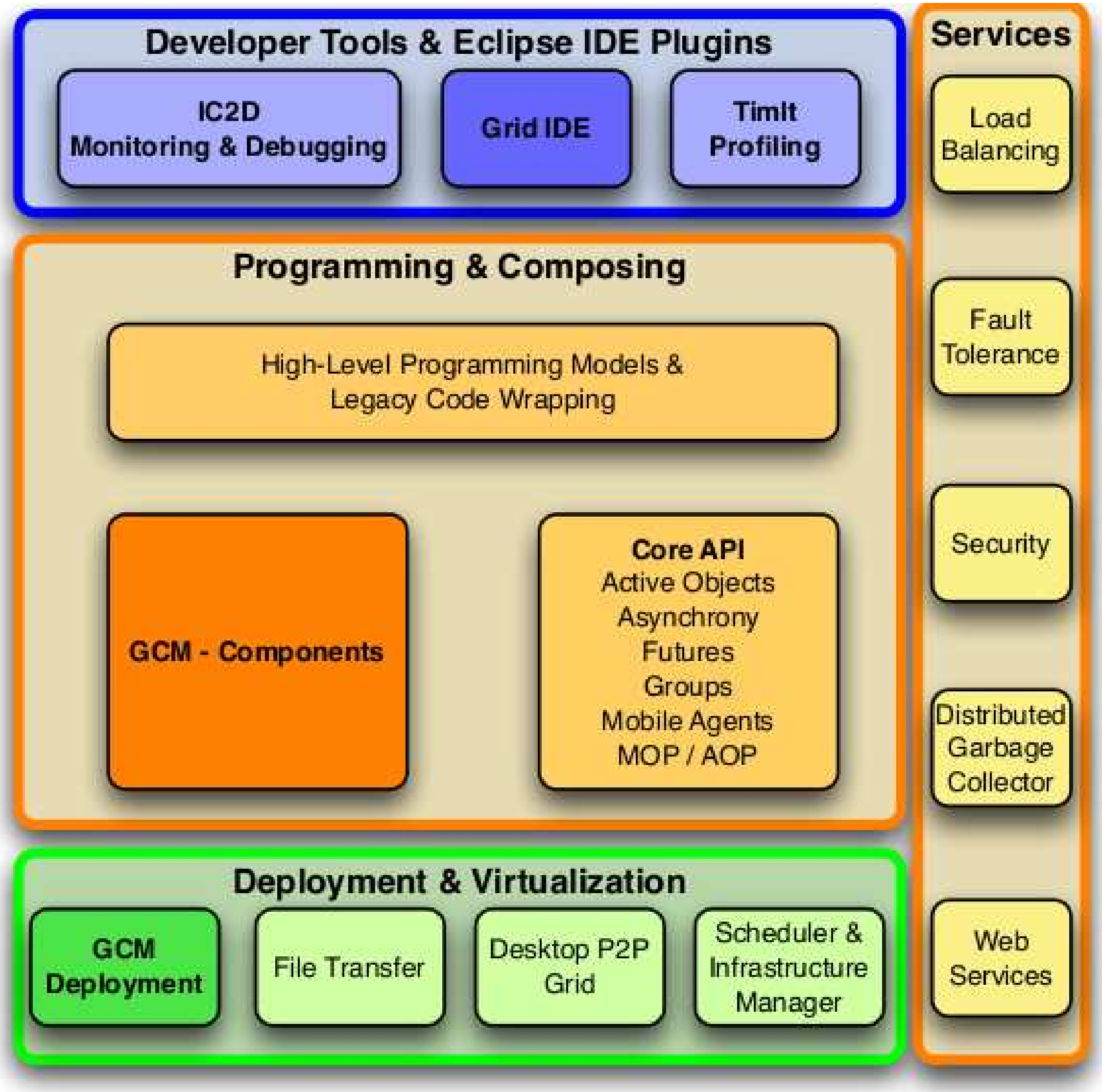
- Components are a way to globally manipulate distributed, and running activities



Agenda

- ProActive and ProActive Parallel Suite
- Programming and Composing
  - ProActive Core
  - High Level Programming models
  - ProActive Components
- Deployment Framework
- Development Tools

GCM Deployment



Developer Tools & Eclipse IDE Plugins

IC2D Monitoring & Debugging

Grid IDE

TimIt Profiling

Programming & Composing

High-Level Programming Models & Legacy Code Wrapping

GCM - Components

Core API
Active Objects
Asynchrony
Futures
Groups
Mobile Agents
MOP / AOP

Deployment & Virtualization

GCM Deployment

File Transfer

Desktop P2P Grid

Scheduler & Infrastructure Manager

Services

Load Balancing

Fault Tolerance

Security

Distributed Garbage Collector

Web Services

Abstract Deployment Model

Problem

Difficulties and lack of flexibility in deployment

Avoid scripting for configuration, getting nodes, connecting…

A key principle: Virtual Node (VN)

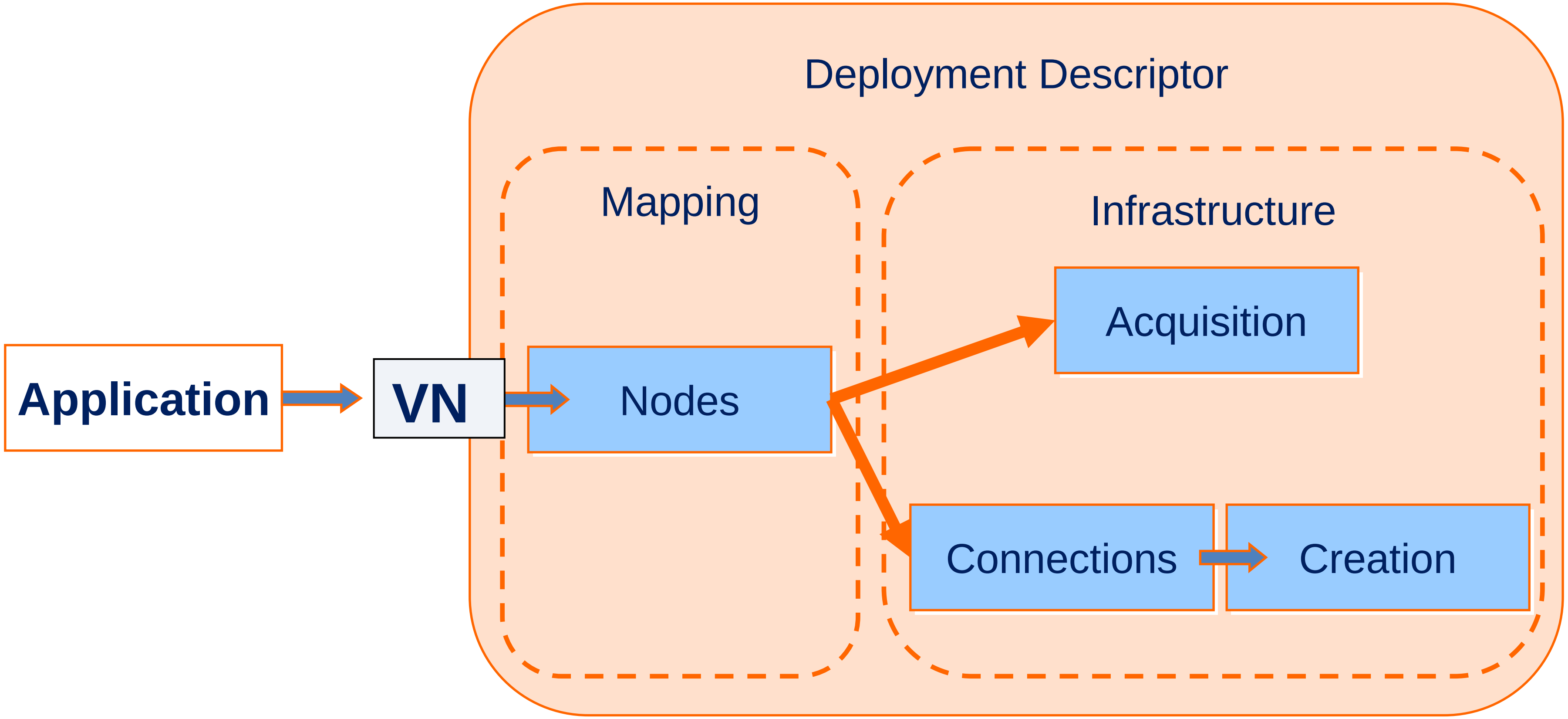
Abstract Away from source code:

- Machines names
- Creation/Connection Protocols
- Lookup and Registry Protocols

Interface with various protocols and infrastructures:

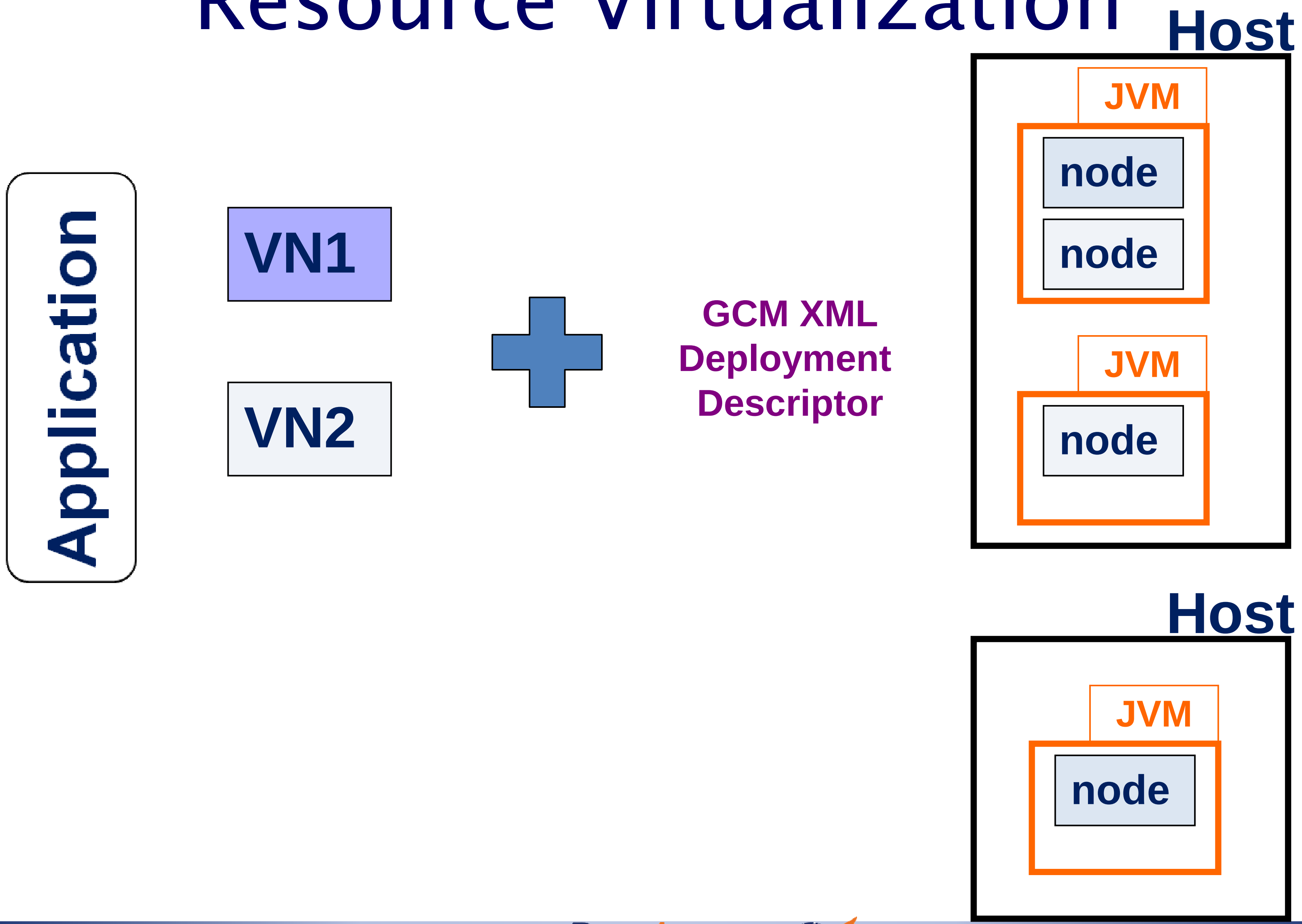
- Cluster: LSF, PBS, SGE , OAR and PRUN(custom protocols)
- Intranet P2P, LAN: intranet protocols: rsh, rlogin, ssh
- Grid: Globus, Web services, ssh, gsissh

Resource Virtualization

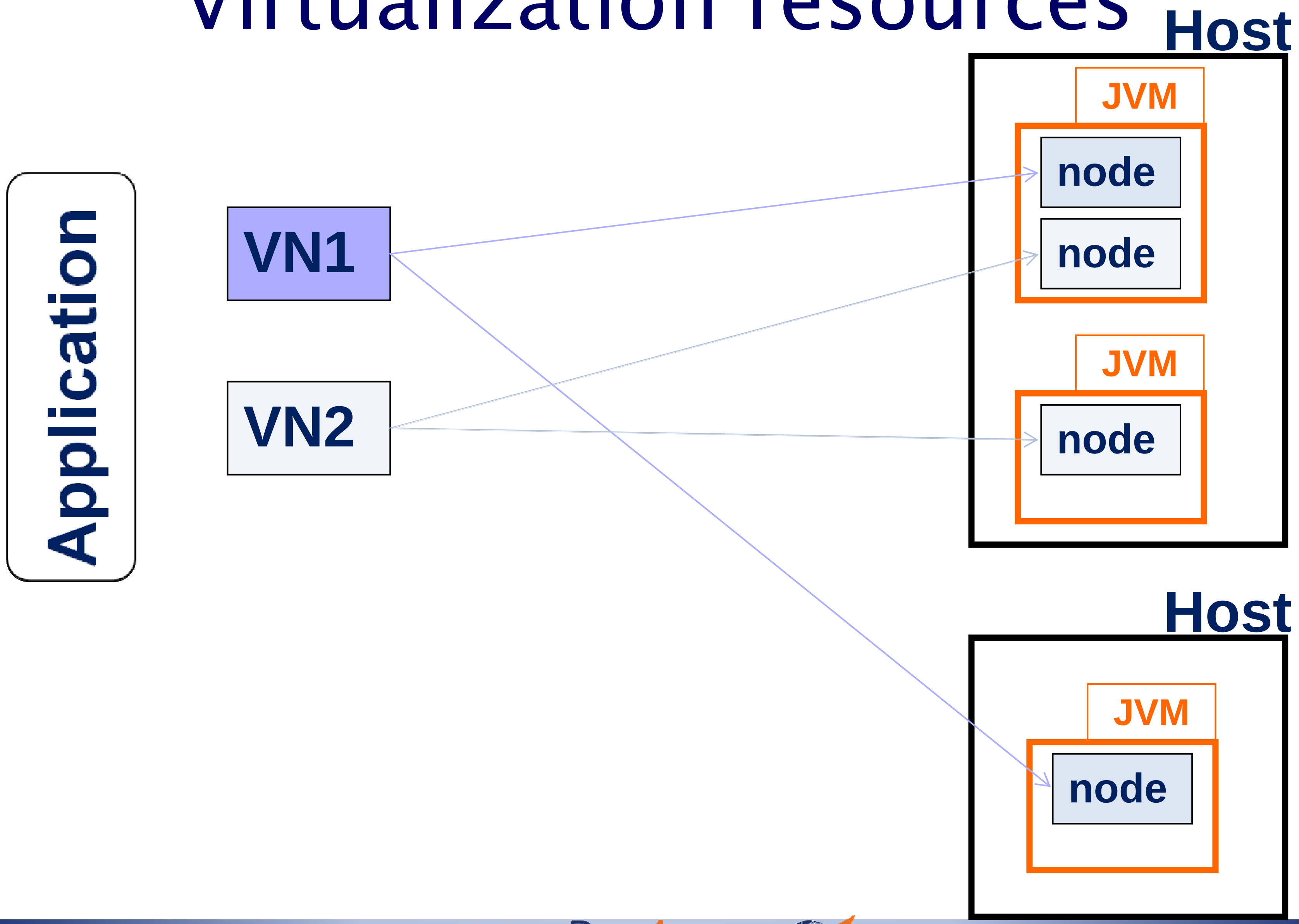


Runtime structured entities: 1 VN --> n Nodes in m JVMs on k Hosts

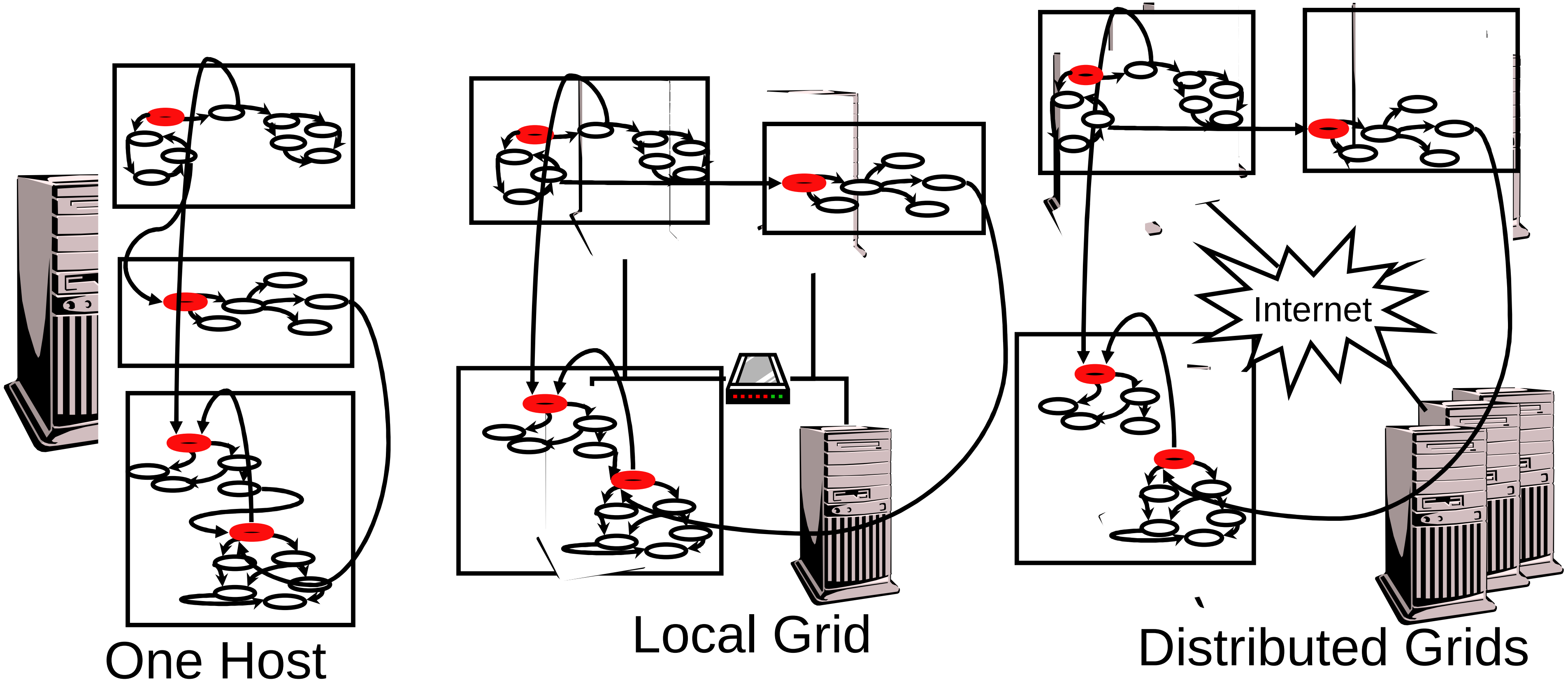
Resource Virtualization



Virtualization resources



Multiple Deployments



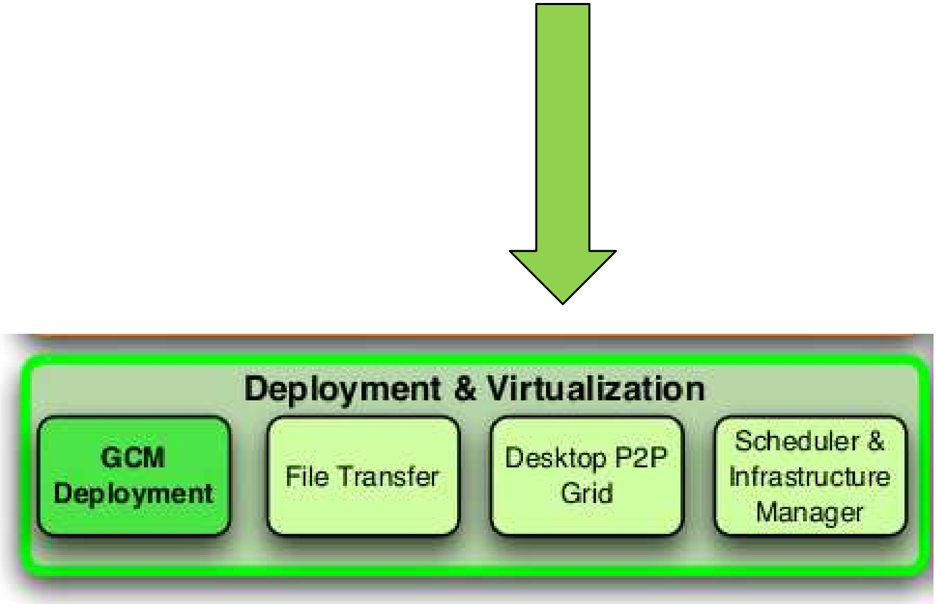
Rmissh : SSH Tunneling

- **A fact : overprotected clusters**
  - Firewalls prevent incoming connections
  - Use of private addresses
  - NAT, IP Address filtering, …
- **A consequence :**
  - Multi clustering is a nightmare
- **Context :**
  - SSH protocol : encrypt network traffic
  - Administrators accept to open SSH port
  - SSH provides encryption

Rmissh : SSH Tunneling (2)

- Create a communication protocol within ProActive that allows firewall transversal
- Encapsulates rmi streams within ssh tunnels
- Avoid ssh tunneling costs when possible by first trying a direct rmi connection then fallbacking with rmissh

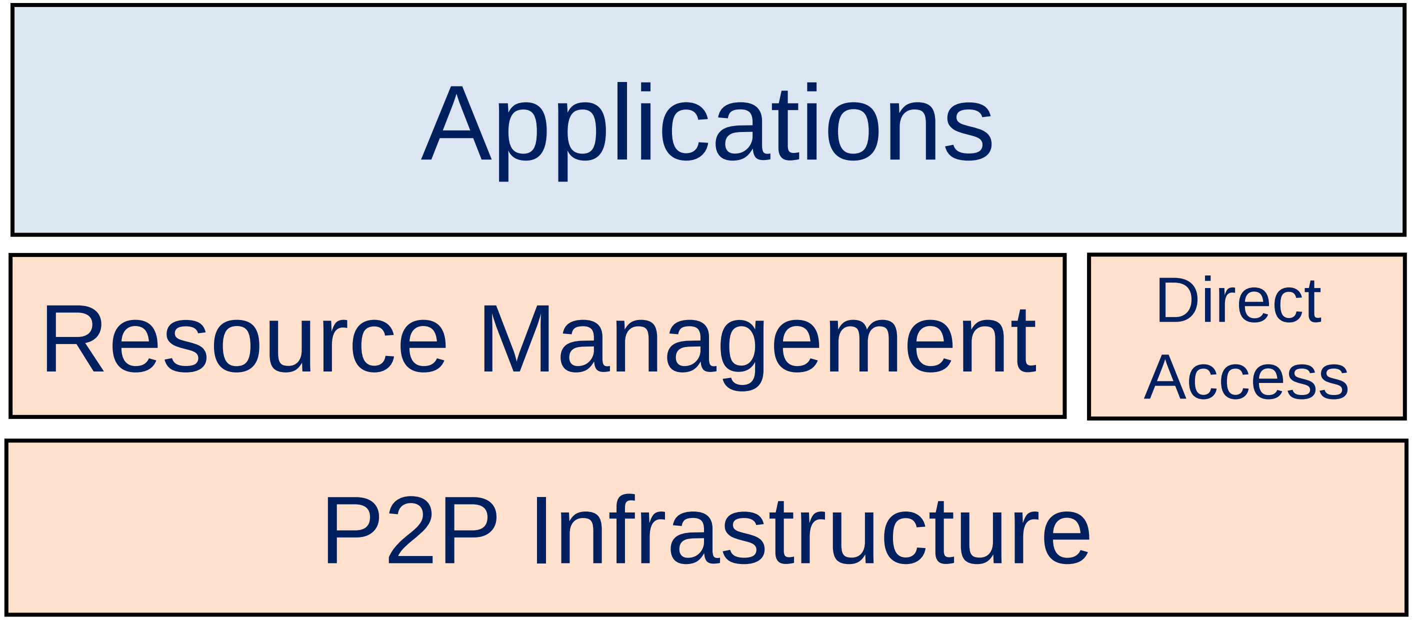
The ProActive P2P



The ProActive P2P

- Unstructured P2P
  - Easier to deploy/manage
  - Only 1 resource : CPU
- Java code
  - Each peer is written in Java and can run any Java application
- Direct communications
  - Peers are reachable using their name (URLs)
  - One peer can send/receive a reference on another peer

The ProActive P2P (2)



Applications

Resource Management

Direct Access

P2P Infrastructure

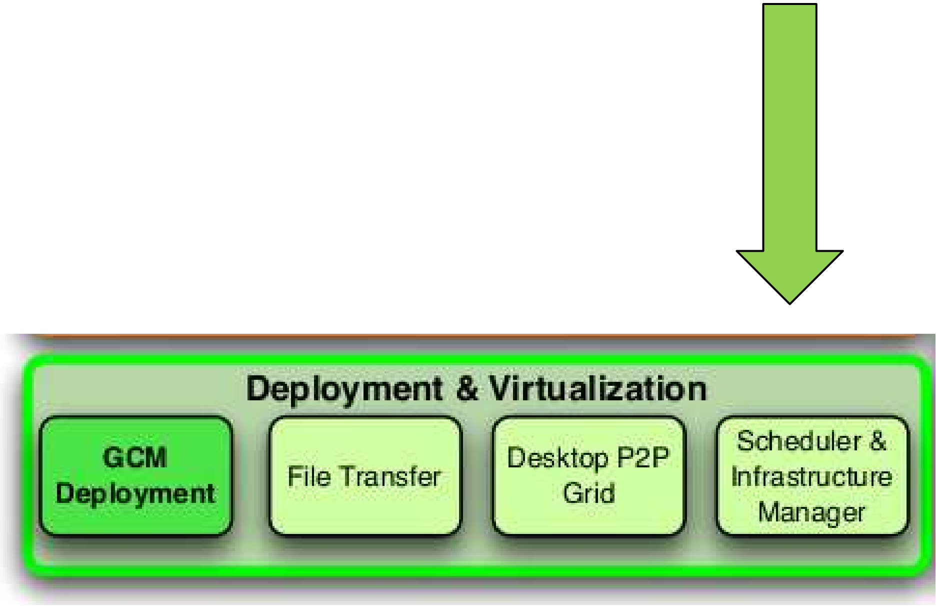
Infrastructure

- A peer is an Active Object in a JVM
- Each peer knows a limited number of other peers (bi-directional links)
  - Its *acquaintances*
  - The number is set by a variable (*NOA*)
- Goal of a peer
  - A peer will always try to maintain the number of its acquaintances equals to its *NOA*
- 2 basic operations
  - Adding an acquaintance
  - Removing an acquaintance

Requesting Nodes

- To request a node
  - Contact only a Peer (URLs)
- The infrastructure will handle the reservation
- The application has to wait until the nodes are available
- Using the P2P network
  - Programmatically at runtime using the Java API
  - At Deployment time through the GCMDeployment

Scheduler and Resource manager



Deployment & Virtualization

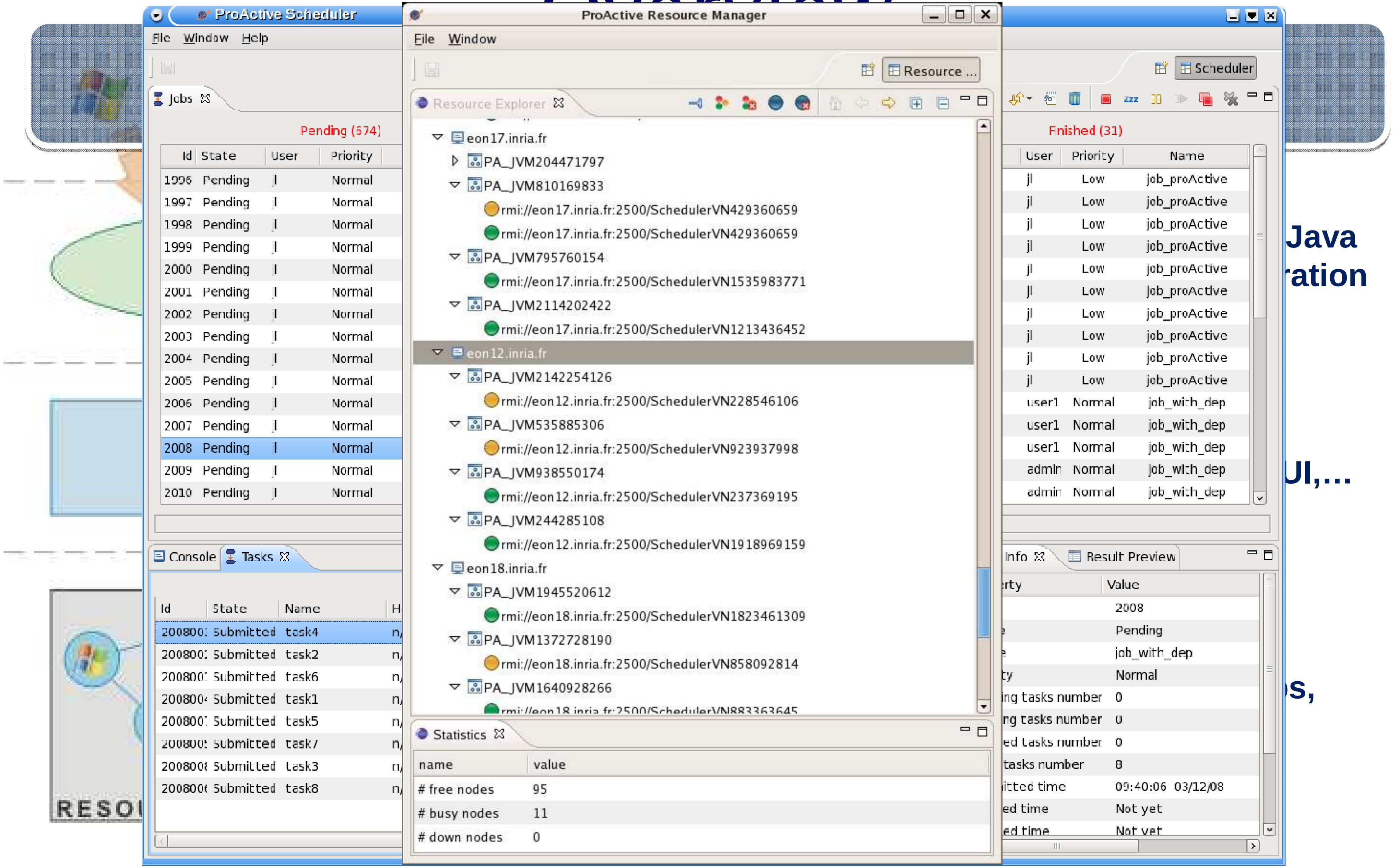
GCM Deployment

File Transfer

Desktop P2P Grid

Scheduler & Infrastructure Manager

Scheduler / Resource Manager Overview



ProActive Scheduler

Jobs

Pending (574)

Id	State	User	Priority
1996	Pending	jl	Normal
1997	Pending	jl	Normal
1998	Pending	jl	Normal
1999	Pending	jl	Normal
2000	Pending	jl	Normal
2001	Pending	jl	Normal
2002	Pending	jl	Normal
2003	Pending	jl	Normal
2004	Pending	jl	Normal
2005	Pending	jl	Normal
2006	Pending	jl	Normal
2007	Pending	jl	Normal
2008	Pending	jl	Normal
2009	Pending	jl	Normal
2010	Pending	jl	Normal

Console | Tasks

Id	State	Name	H
200800	Submitted	task4	n
200800	Submitted	task2	n
200800	Submitted	task6	n
200800	Submitted	task1	n
200800	Submitted	task5	n
200800	Submitted	task7	n
200800	Submitted	task3	n
200800	Submitted	task8	n

Finished (31)

User	Priority	Name
jl	Low	job_proActive
jl	Low	job_proActive
jl	Low	job_proActive
jl	Low	job_proActive
jl	Low	job_proActive
jl	Low	job_proActive
jl	Low	job_proActive
jl	Low	job_proActive
jl	Low	job_proActive
jl	Low	job_proActive
user1	Normal	job_with_dep
user1	Normal	job_with_dep
user1	Normal	job_with_dep
admin	Normal	job_with_dep
admin	Normal	job_with_dep

Info | Result Preview

rty	Value
	2008
e	Pending
e	job_with_dep
ty	Normal
ing tasks number	0
ng tasks number	0
ed tasks number	0
tasks number	8
itted time	09:40:06 03/12/08
ed time	Not yet
ed time	Not yet

ProActive Resource Manager

Resource Explorer

- eon17.inria.fr
  - PA_JVM204471797
  - PA_JVM810169833
    - rmi://eon17.inria.fr:2500/SchedulerVN429360659
    - rmi://eon17.inria.fr:2500/SchedulerVN429360659
  - PA_JVM795760154
    - rmi://eon17.inria.fr:2500/SchedulerVN1535983771
  - PA_JVM2114202422
    - rmi://eon17.inria.fr:2500/SchedulerVN1213436452
- eon12.inria.fr
  - PA_JVM2142254126
    - rmi://eon12.inria.fr:2500/SchedulerVN228546106
  - PA_JVM535885306
    - rmi://eon12.inria.fr:2500/SchedulerVN923937998
  - PA_JVM938550174
    - rmi://eon12.inria.fr:2500/SchedulerVN237369195
  - PA_JVM244285108
    - rmi://eon12.inria.fr:2500/SchedulerVN1918969159
- eon18.inria.fr
  - PA_JVM1945520612
    - rmi://eon18.inria.fr:2500/SchedulerVN1823461309
  - PA_JVM1372728190
    - rmi://eon18.inria.fr:2500/SchedulerVN858092814
  - PA_JVM1640928266
    - rmi://eon18.inria.fr:2500/SchedulerVN883363645

Statistics

name	value
# free nodes	95
# busy nodes	11
# down nodes	0

Java

ration

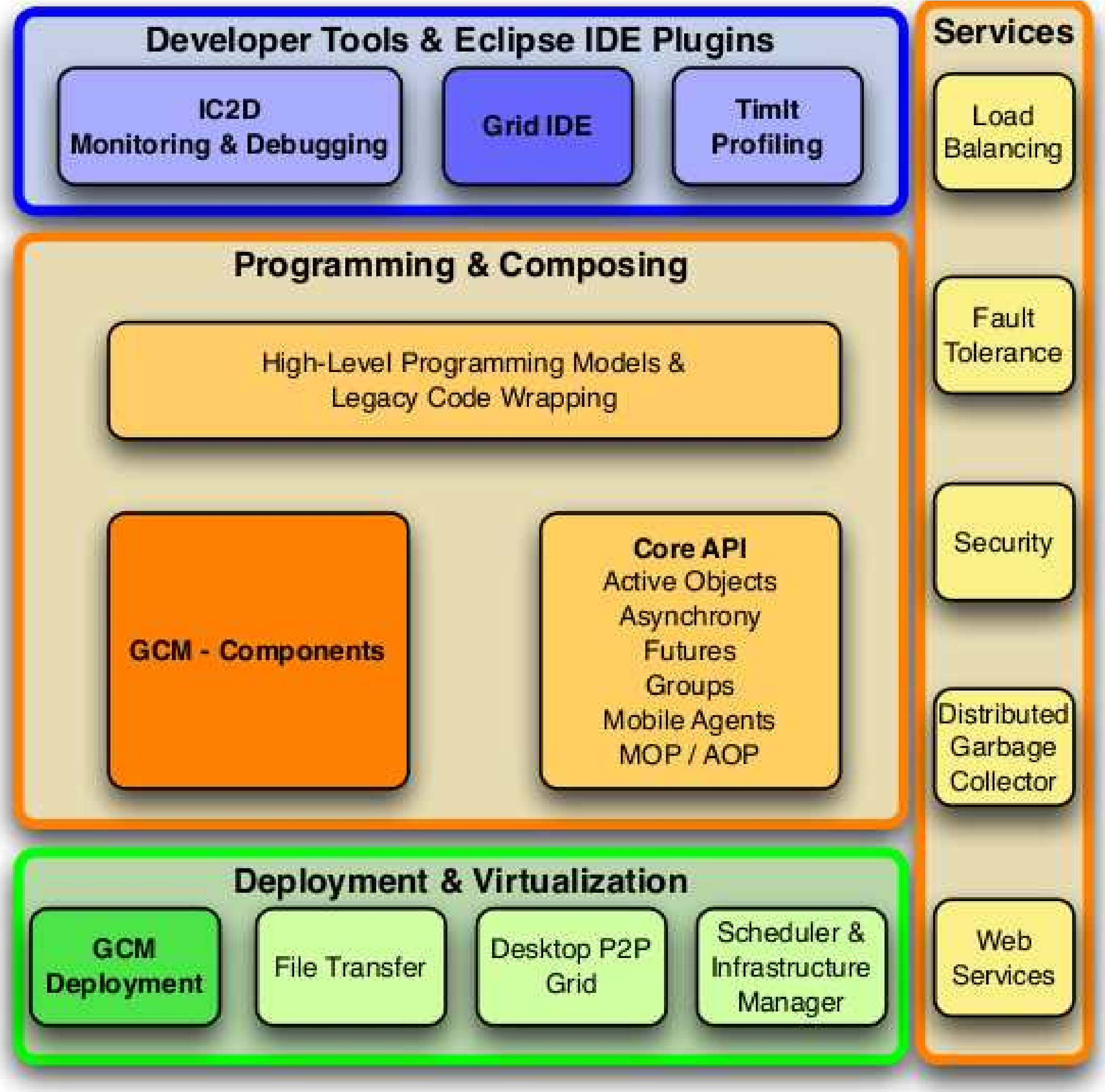
UI,...

s,

Agenda

- ProActive and ProActive Parallel Suite
- Programming and Composing
  - ProActive Core
  - High Level Programming models
  - ProActive Components
  - Legacy code wrapping
- Deployment Framework
- Development Tools

ProActive Parallel Suite



Developer Tools & Eclipse IDE Plugins

IC2D
Monitoring & Debugging

Grid IDE

TimIt
Profiling

Programming & Composing

High-Level Programming Models &
Legacy Code Wrapping

GCM - Components

Core API
Active Objects
Asynchrony
Futures
Groups
Mobile Agents
MOP / AOP

Deployment & Virtualization

GCM
Deployment

File Transfer

Desktop P2P
Grid

Scheduler &
Infrastructure
Manager

Services

Load
Balancing

Fault
Tolerance

Security

Distributed
Garbage
Collector

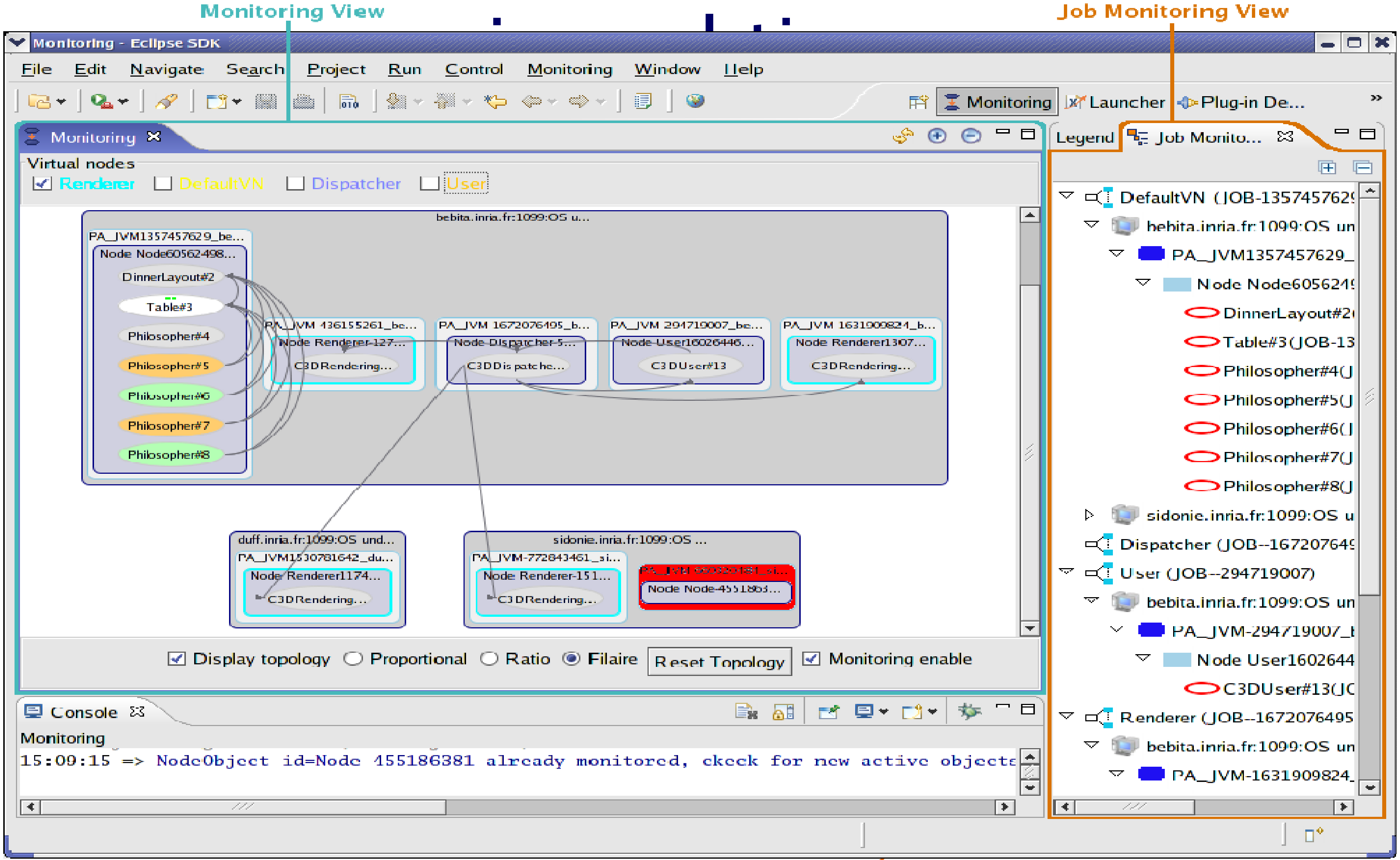
Web
Services

IC2D

Interactive Control & Debug for Distribution

- Basic Features:
  - Graphical visualization
  - Textual visualization
  - Monitoring and Control
- Extensible through RCP plug-ins
  - TimIt
  - ChartIt
  - P2P view
  - DGC view

IC2D: Monitor your application



Monitoring View

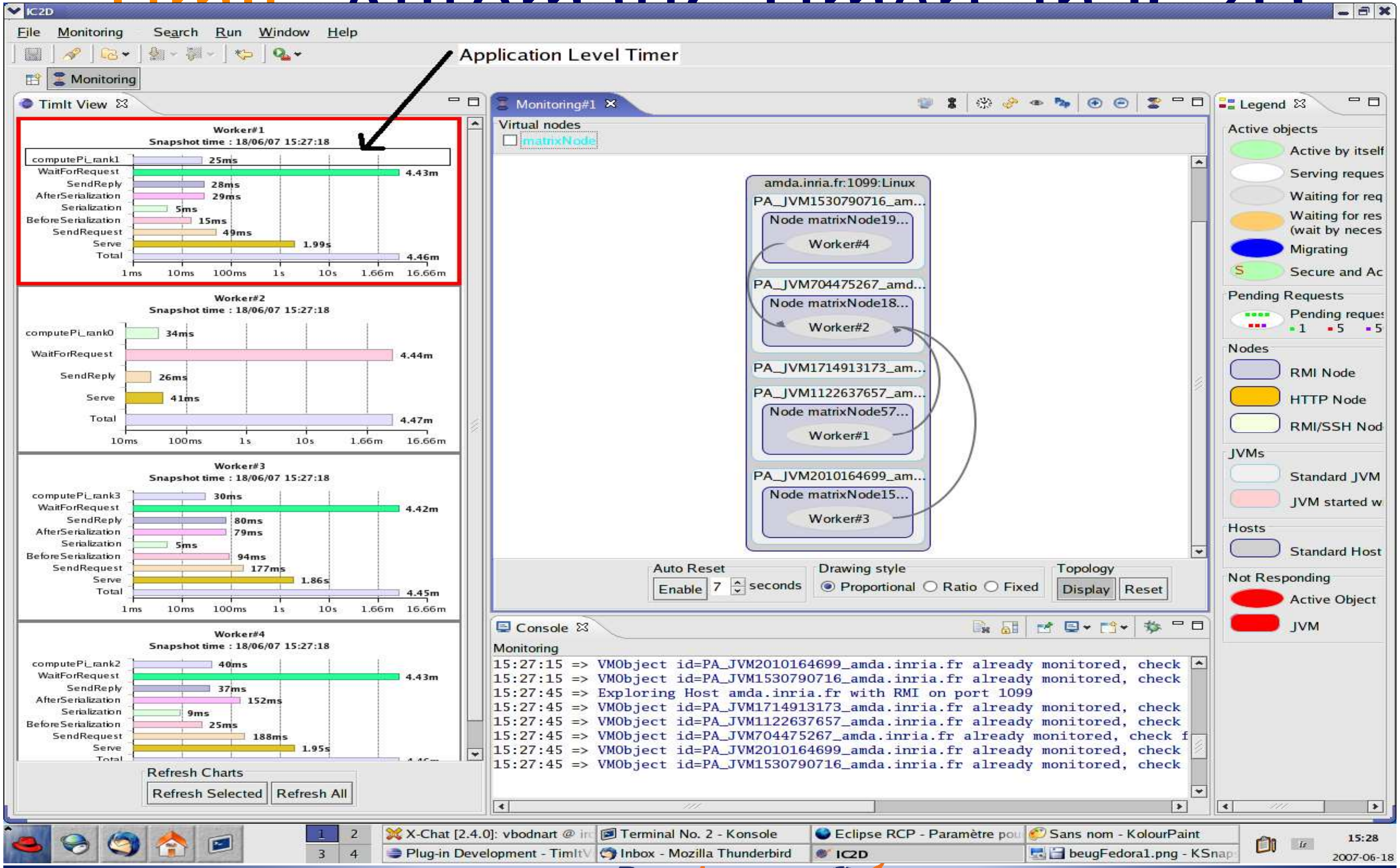
Job Monitoring View

☑ Renderer ☐ DefaultVN ☐ Dispatcher ☐ User

☑ Display topology ○ Proportional ○ Ratio ● Filaire [Reset Topology] ☑ Monitoring enable

```
Monitoring
15:09:15 => NodeObject id=Node 455186381 already monitored, ckeck for new active object
```

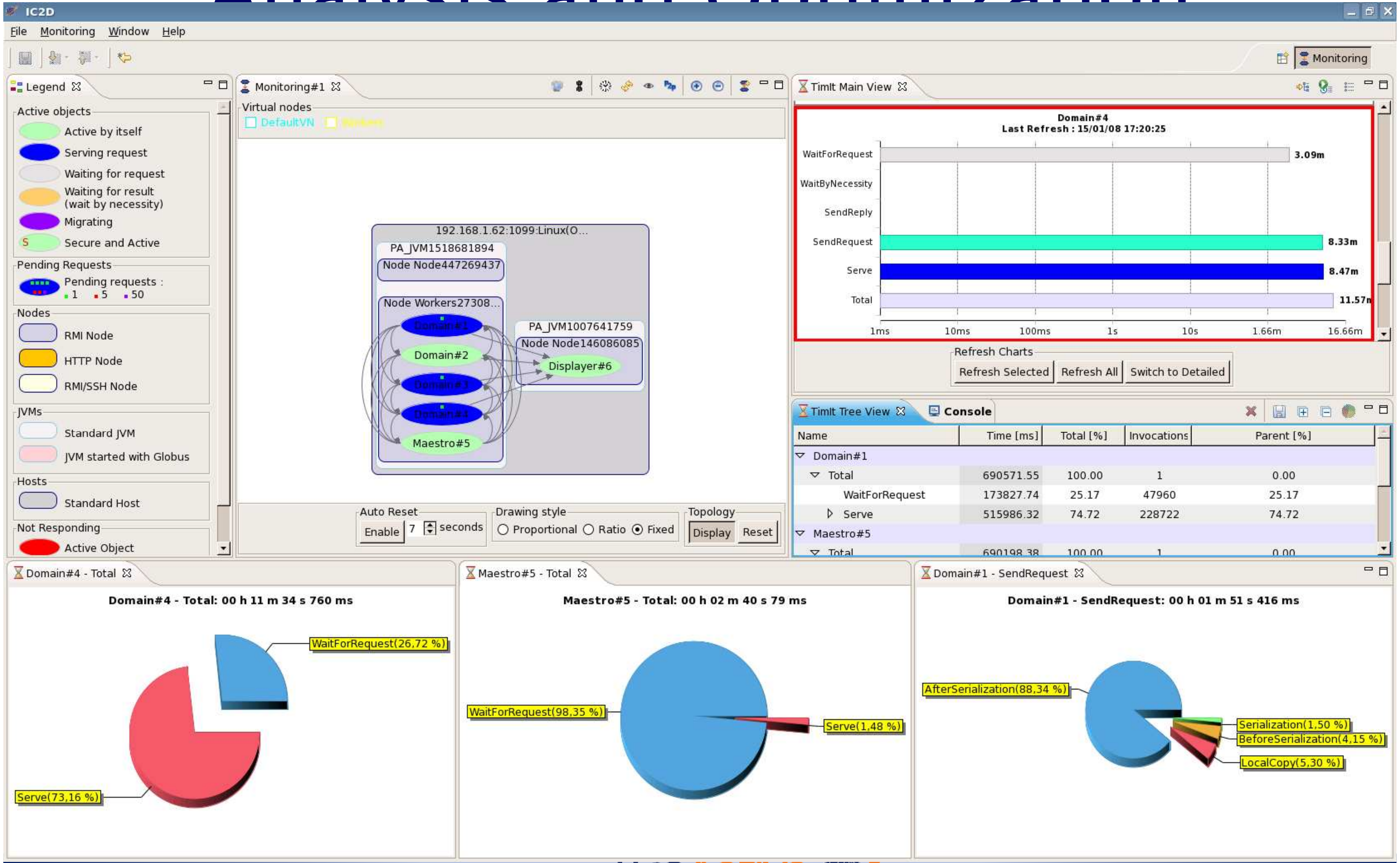
TimIt: Automatic Timers in IC2D



Application Level Timer

```
15:27:15 => VMObject id=PA_JVM2010164699_amda.inria.fr already monitored, check
15:27:15 => VMObject id=PA_JVM1530790716_amda.inria.fr already monitored, check
15:27:45 => Exploring Host amda.inria.fr with RMI on port 1099
15:27:45 => VMObject id=PA_JVM1714913173_amda.inria.fr already monitored, check
15:27:45 => VMObject id=PA_JVM1122637657_amda.inria.fr already monitored, check
15:27:45 => VMObject id=PA_JVM704475267_amda.inria.fr already monitored, check f
15:27:45 => VMObject id=PA_JVM2010164699_amda.inria.fr already monitored, check
15:27:45 => VMObject id=PA_JVM1530790716_amda.inria.fr already monitored, check
```

Analysis and Optimization



Auto Reset: Enable 7 seconds

Drawing style: ○ Proportional ○ Ratio ● Fixed

Domain#4
Last Refresh : 15/01/08 17:20:25

Name	Time [ms]	Total [%]	Invocations	Parent [%]
▽ Domain#1				
▽ Total	690571.55	100.00	1	0.00
WaitForRequest	173827.74	25.17	47960	25.17
▷ Serve	515986.32	74.72	228722	74.72
▽ Maestro#5				
▽ Total	690198.38	100.00	1	0.00

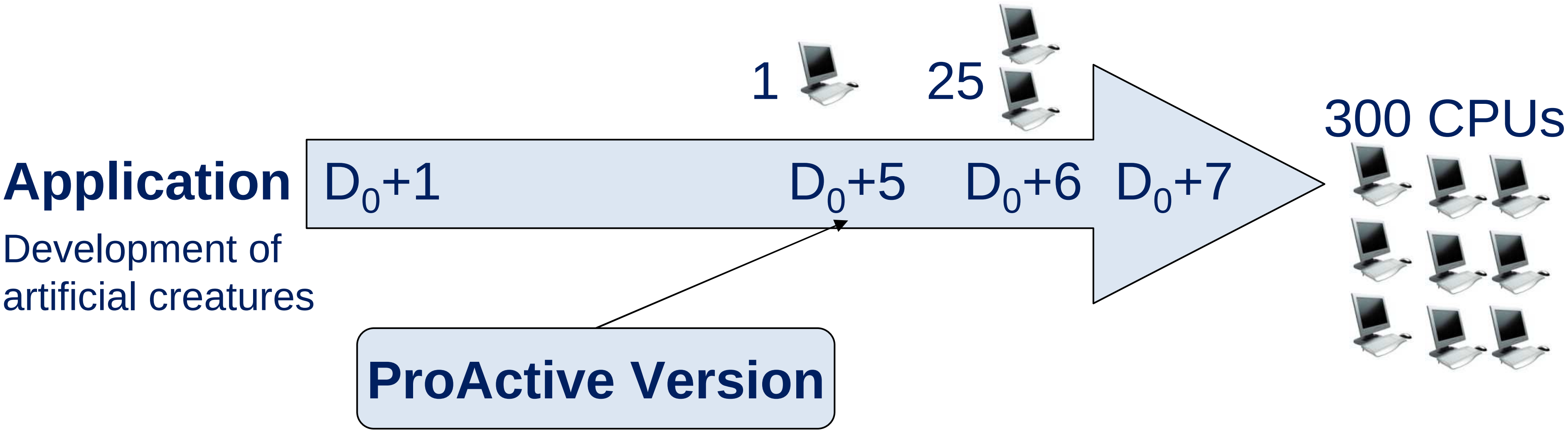
Domain#4 - Total: 00 h 11 m 34 s 760 ms

Maestro#5 - Total: 00 h 02 m 40 s 79 ms

Domain#1 - SendRequest: 00 h 01 m 51 s 416 ms

M/W Success Story: Artificial Life Generation

Sylvain Cussat-Blanc, Yves Duthen – IRIT TOULOUSE



Application

Development of artificial creatures

ProActive Version

Initial Application (C++)	1 PC	56h52 => Crashed
ProActive Version	**300 CPUs**	**19 minutes**

Price-It workload distribution with ProActive

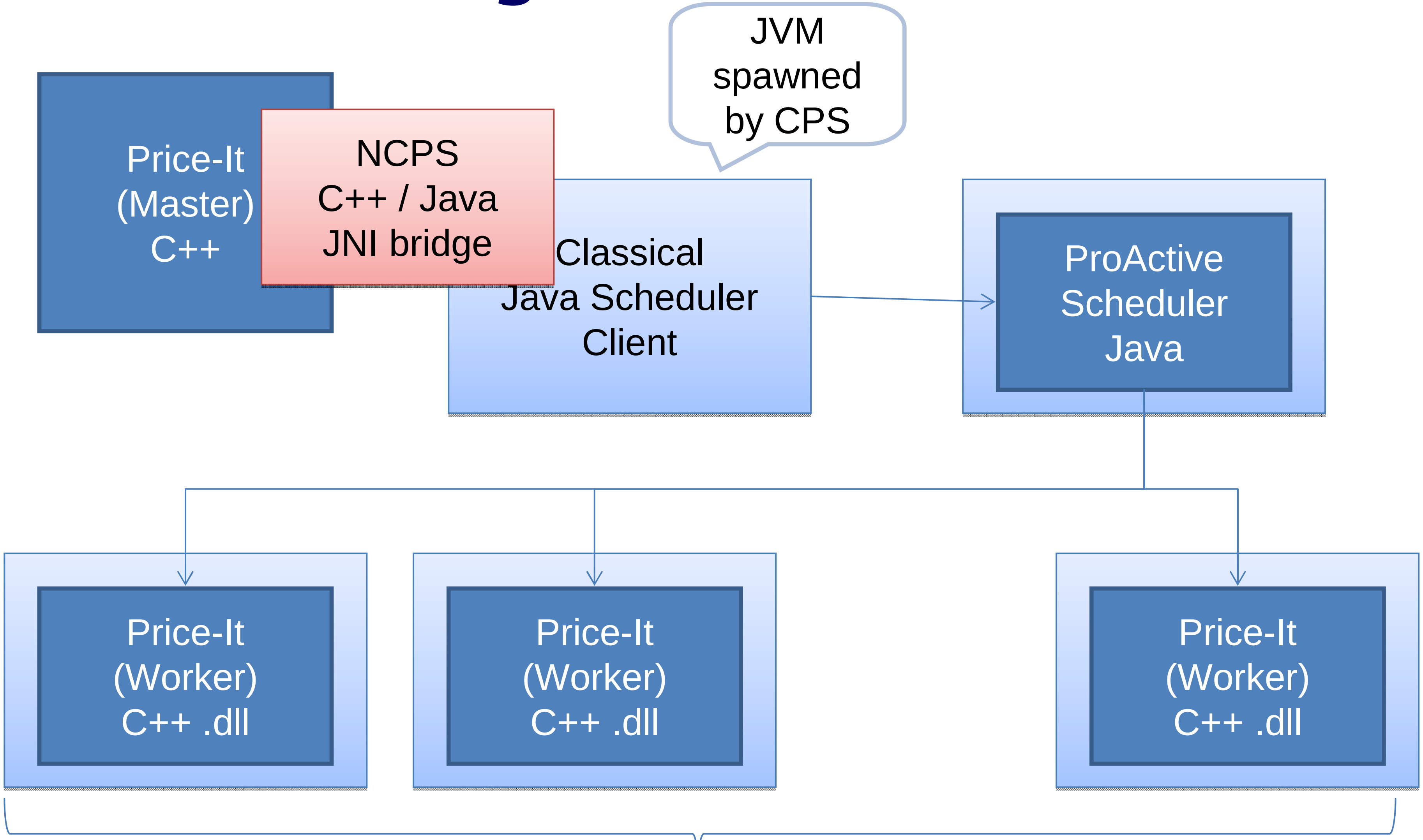
- Low level parallelism : shared memory
- Written in c++
- Originally written for microsoft compiler
- JNI, Com interface
- No thread safe

- Upgrading the code base to thread safe code might be costly
- Is there any easier and cheaper alternative to extract parallelism from Price-it Library ?

CPS : C++ API Client for ProActive Scheduler

- CPS : Client for ProActive Scheduler
- Shipped as .so/.dll
- A set of C++ methods to submit jobs to the Scheduler
  - ☐ SchedulerClient::init() and dispose()
  - ☐ SchedulerClient::submitJob(Job* jobPtr)
  - ☐ SchedulerClient::getJobResult(int jobId)
- Internally uses JNI

Using CPS in Price-It



JVM spawned by CPS

Price-It (Master) C++

NCPS C++ / Java JNI bridge

Classical Java Scheduler Client

ProActive Scheduler Java

Price-It (Worker) C++ .dll

Price-It (Worker) C++ .dll

Price-It (Worker) C++ .dll

Workers are shipped as .dll then loaded by JVMS and executed through JNI

Conclusion

- Simple and Uniform programming model
- Rich High Level API
- Write once, deploy everywhere (GCMD)

- Let the developer concentrate on his code, not on the code distribution

- Easy to install, test, validate on any network

Now, let's play with ProActive…

- Start and monitor with IC2D the ProActive examples, and have a look at the source code

  `org.objectweb.proactive.examples.*`

Features	Applications
Basics, Synchronization	Doctors problem (`doctors.bat`), Reader/Writer problem (`readers.bat`),…
Futures, Automatic Continuation	Binary Search Tree (`bintree.bat`)
Migration	Migrating Agent (`/migration/penguin.bat`)
Group	Chat (`/group/chat.bat`)
Fault-Tolerance	N-body problem (`/FT/nbodyFT.bat`)
All	Distributed 3D renderer (`c3d*.bat`)