ProActive Parallel Suite

Denis Caromel Arnaud Contes

Univ. Nice ActiveEon

Agenda

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

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
Programming, Composing, Deploying on the Grid

ProActive

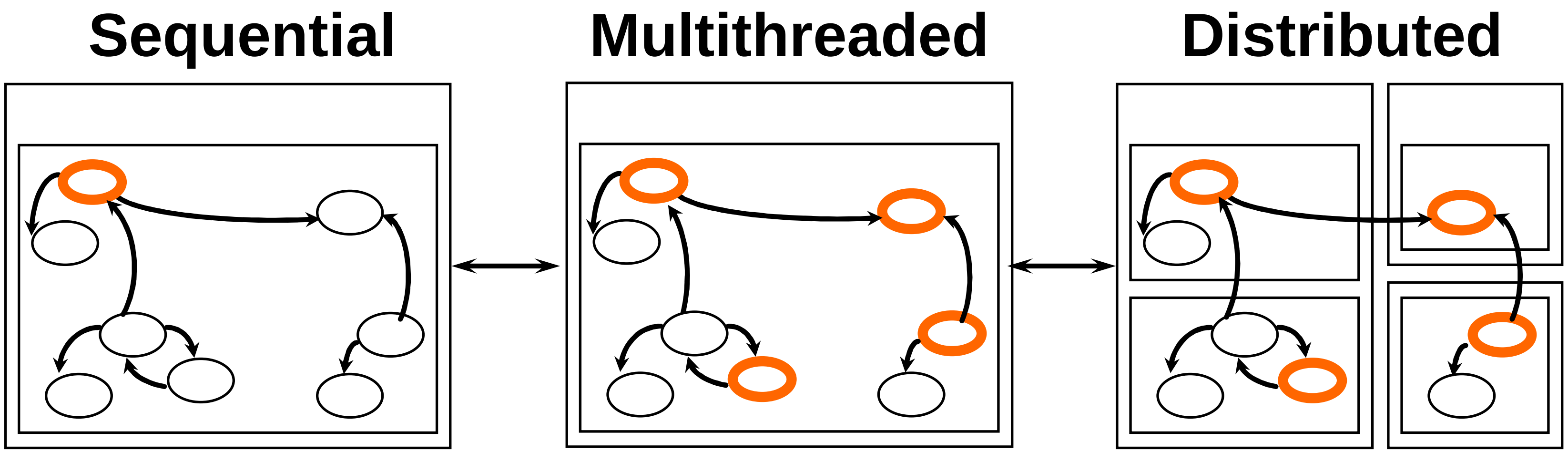
- ProActive is a JAVA middleware for parallel, distributed and multi-threaded computing.

- 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.

Unification of Multi-threading and Multi-processing

Seamless



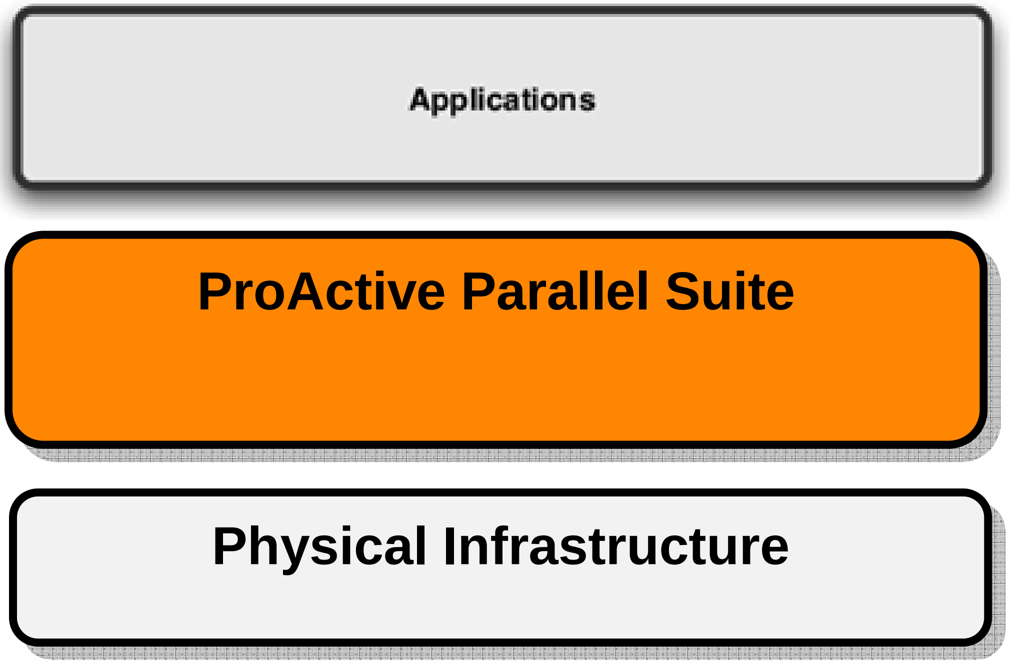
- Most of the time, activities and distribution are not known at the beginning, and change over time
- Seamless implies reuse, smooth and incremental transitions

ProActive Parallel Suite

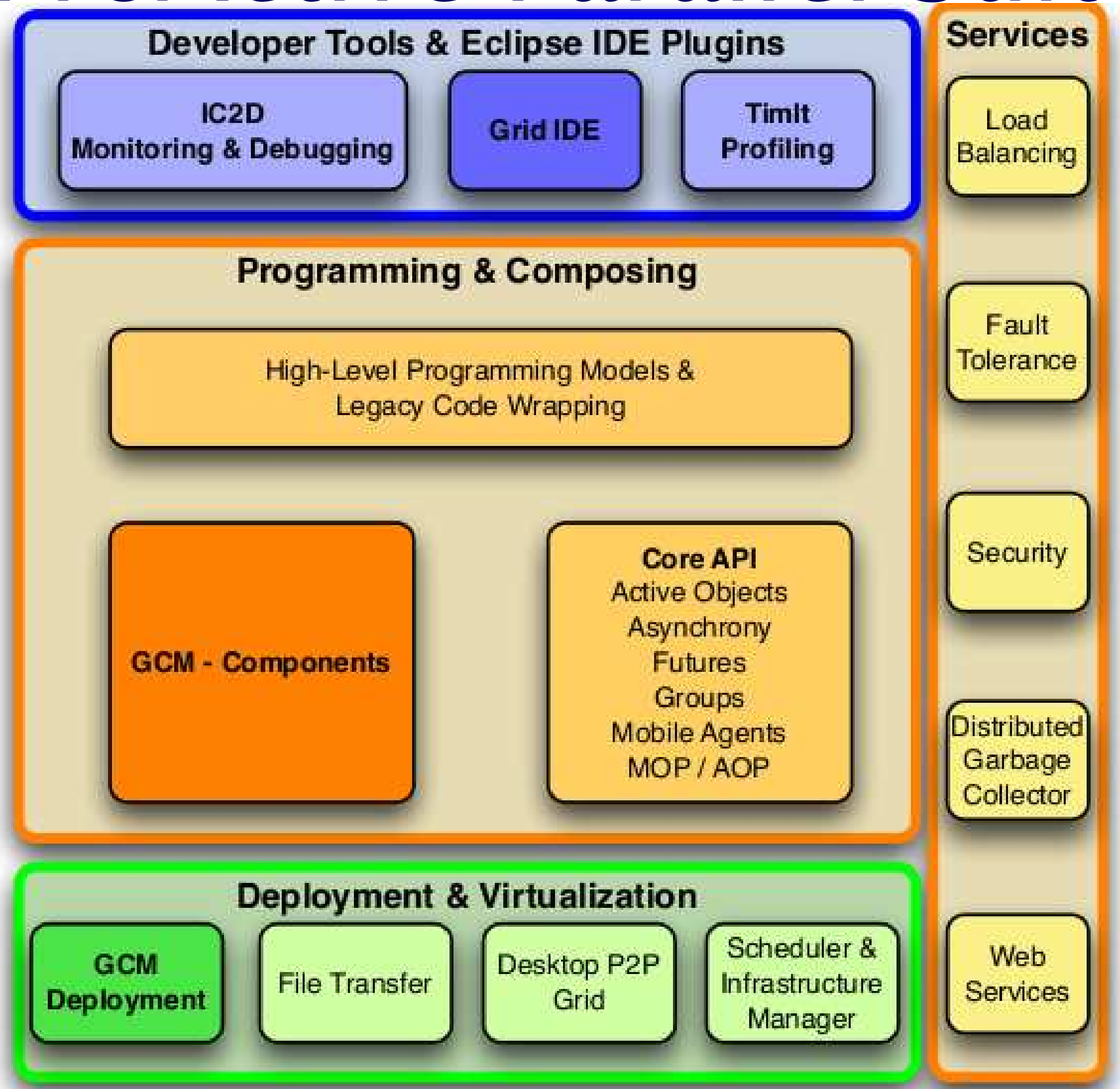
- ProActive Parallel Suite includes:
  - The ProActive middleware featuring services like:
    - Fault tolerance, Load balancing, Distributed GC, Security, WS
    - A set of parallel programming frameworks
    - A framework for deploying applications on distributed infrastructures
  - Software for scheduling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

ProActive
Programming, Composing, Deploying on the Grid

ProActive Parallel Suite



ProActive Parallel Suite



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?

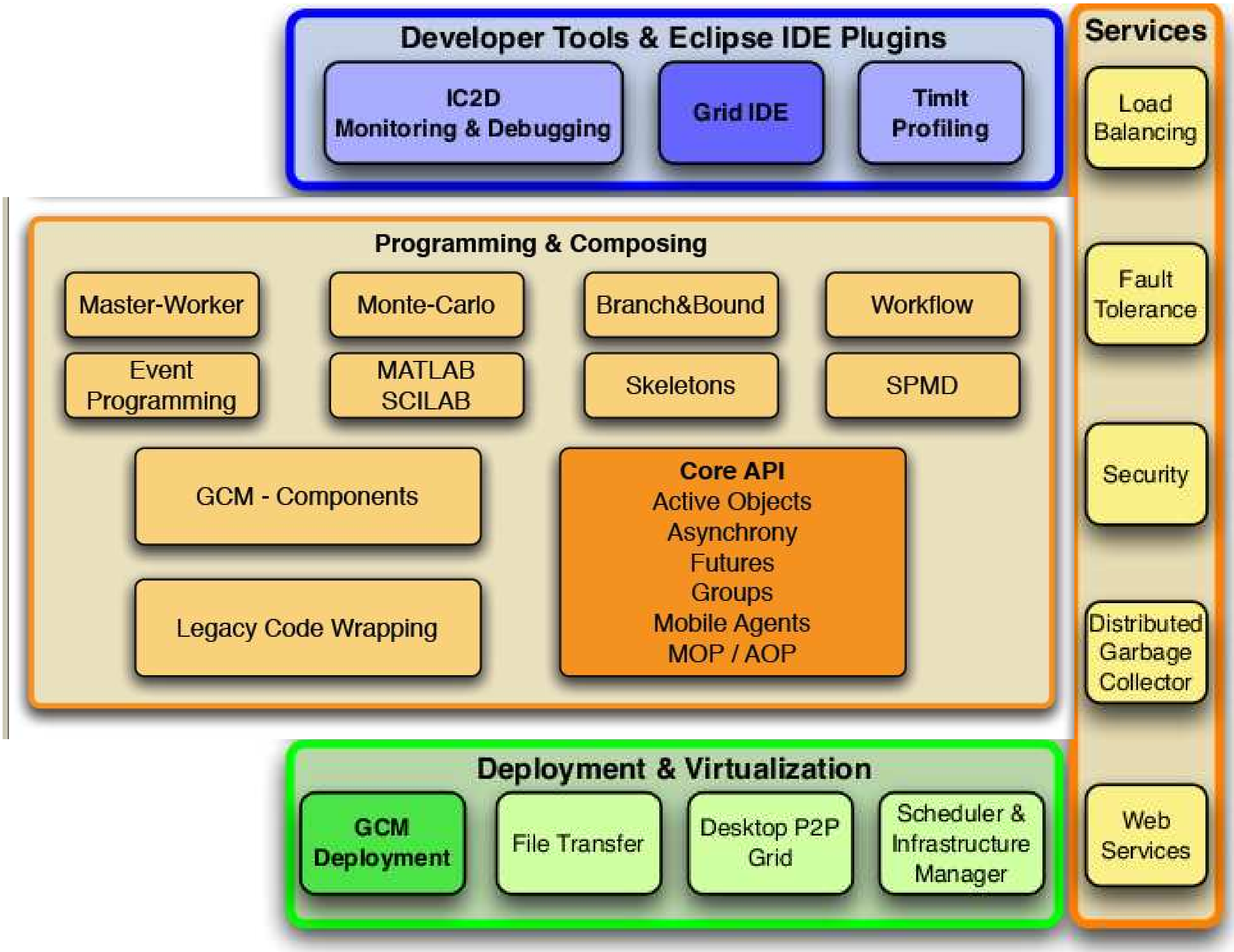
- To wrap your native application with ProActive in order to distribute it

- 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



ProActive Core

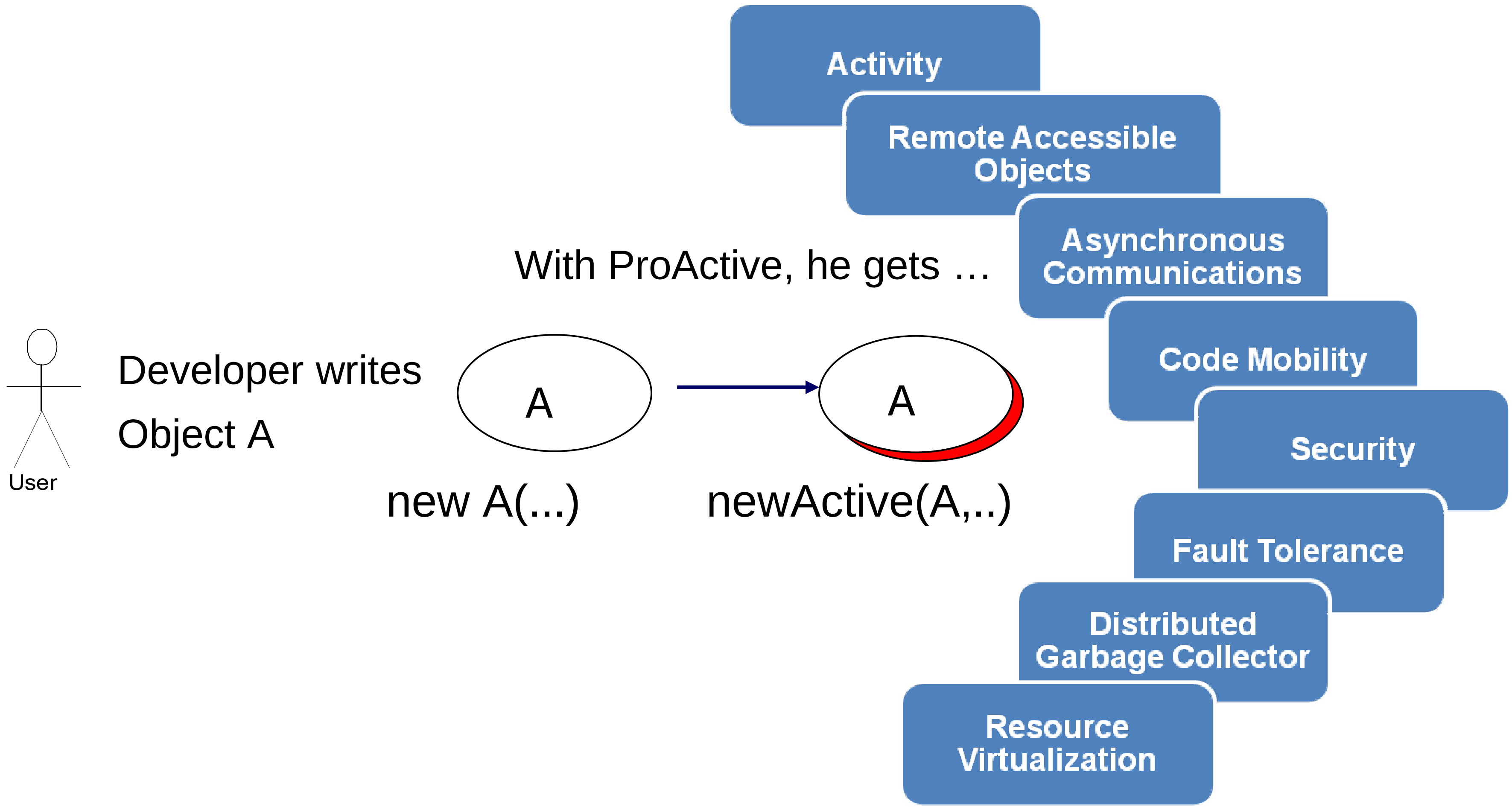
ACTIVE OBJECTS

ProActive

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

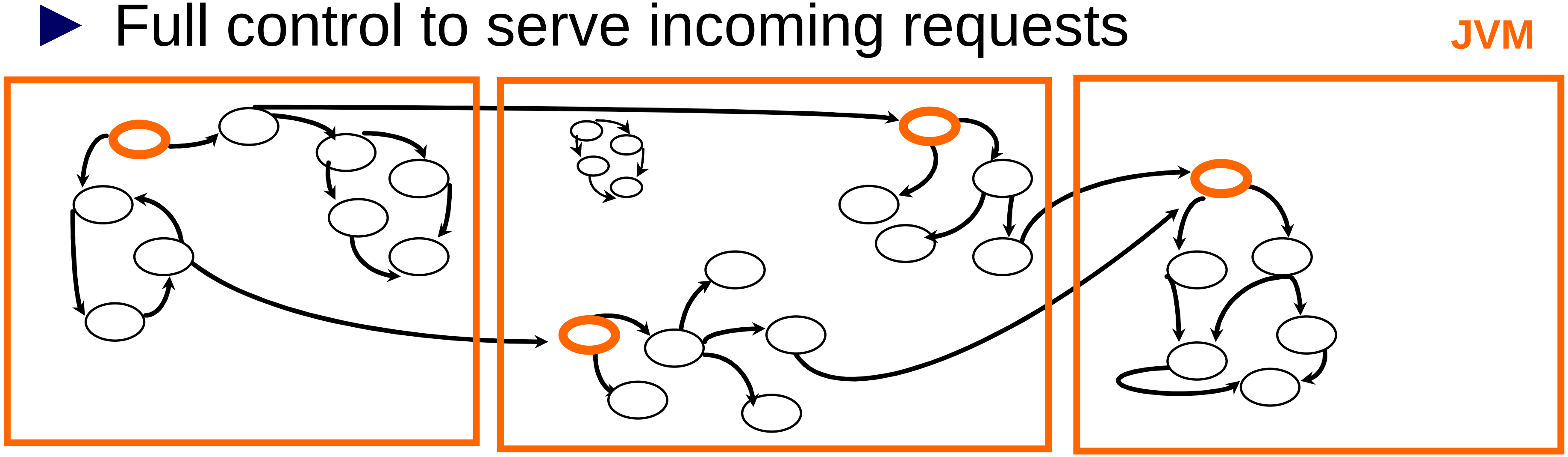
- A programming model: Active Objects
  - Asynchronous Communications, Wait-By-Necessity, Groups, Mobility, Components, Security, Fault-Tolerance
- A formal model behind: Determinism (POPL'04)
  - Insensitive to application deployment
- A uniform Resource framework
  - Resource Virtualization to simplify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

Active Objects



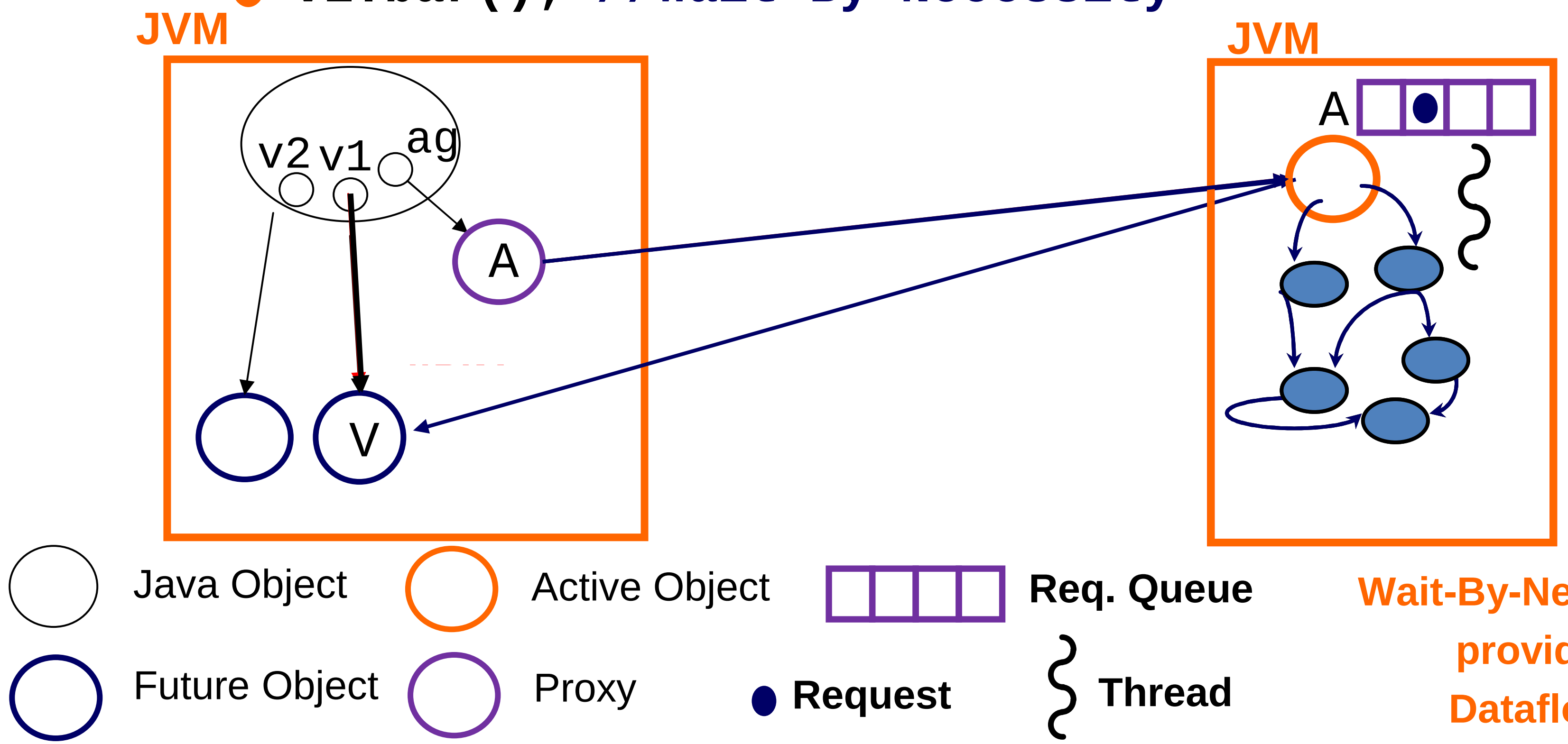
ProActive model : Basis

- 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
```



Wait-By-Necessity provides Dataflow Synchronization

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

Proofs in GREEK

$$\frac{(a,\sigma)\to_S(a',\sigma')}{\alpha[a;\sigma;\iota;F;R;f]\parallel P\longrightarrow\alpha[a';\sigma';\iota;F;R;f]\parallel P}\ (\text{LOCAL})$$

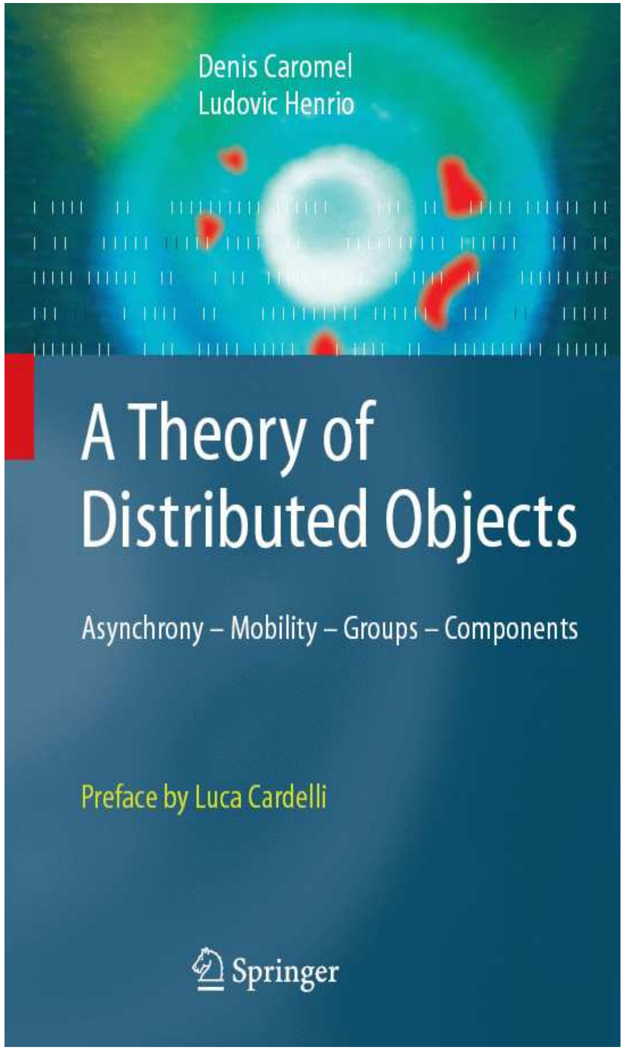
$$\frac{\begin{array}{c}\gamma\text{ fresh activity}\quad \iota'\notin dom(\sigma)\quad \sigma'=\{\iota'\mapsto AO(\gamma)\}::\sigma\\ \sigma_\gamma=copy(\iota'',\sigma)\quad Service=(\text{ if } m_j=\emptyset\text{ then } FifoService\text{ else } \iota''.m_j())\end{array}}{\begin{array}{c}\alpha[\mathcal{R}[Active(\iota'',m_j)];\sigma;\iota;F;R;f]\parallel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

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

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

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

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



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 update of references:
  - requests (remote references remain valid)
  - replies (its previous queries will be fulfilled)
- Migration is initiated by the active object itself
- Can be initiated from outside through any public method

Migration 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

ProActive Core

PROACTIVE GROUPS

ProActive Groups

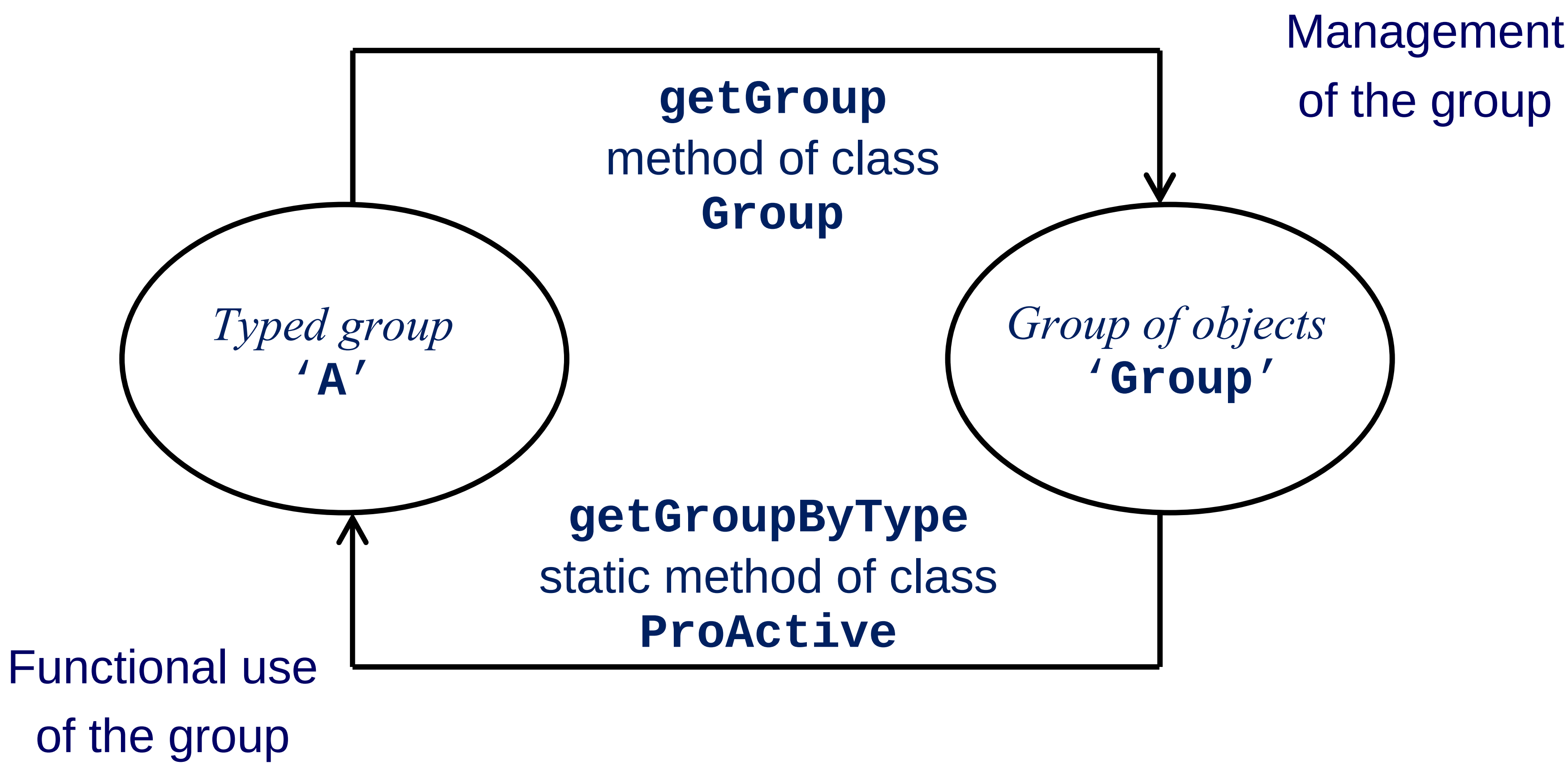
- Manipulate groups of Active Objects, in a simple and typed manner:
    - Typed and polymorphic Groups of local and remote objects
    - Dynamic generation of group of results
    - Language centric, Dot notation
- Be able to express high-level collective communications (like in MPI):
    - broadcast,
    - scatter, gather,
    - all to all

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

ProActive Groups

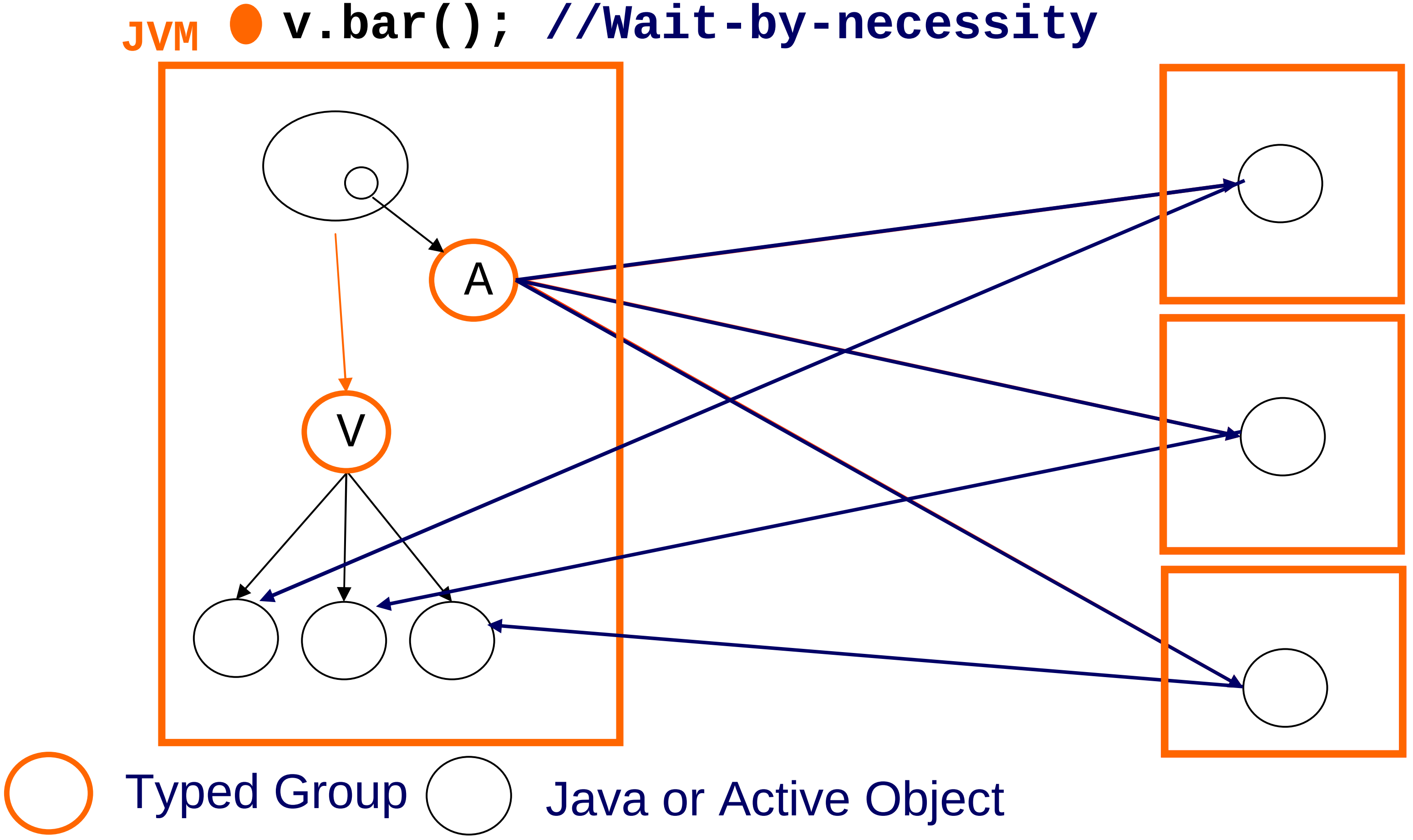
- Group Members
  - Active Objects
  - POJO
  - Group Objects
- Hierarchical Groups
- Based on the ProActive communication mechanism
  - Replication of N ' single ' communications
  - Parallel calls within a group (latency hiding)
- Polymorphism
  - Group typed with member's type

Two Representations Scheme



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, ...

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

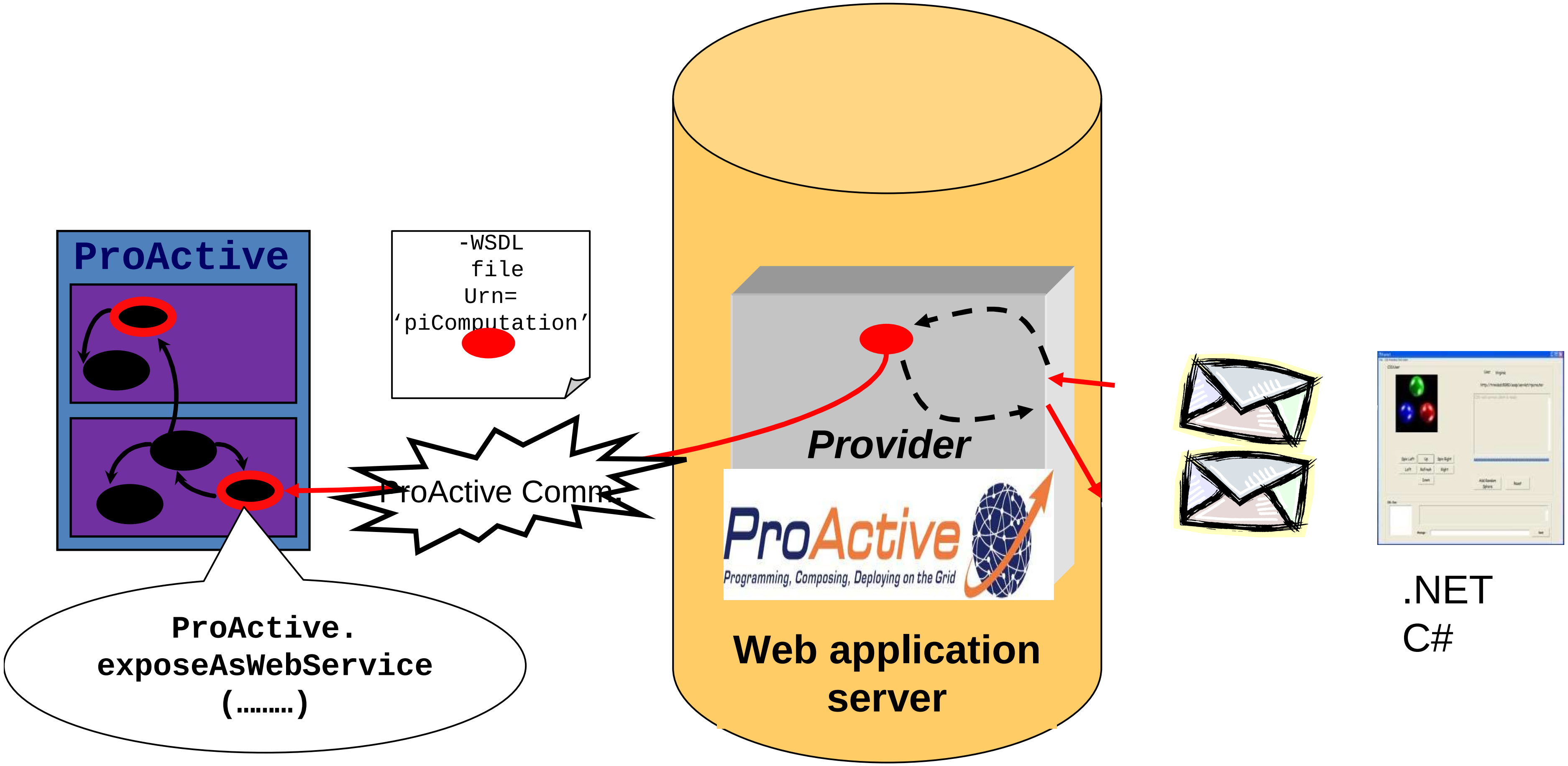
ProActive Core

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 );`**

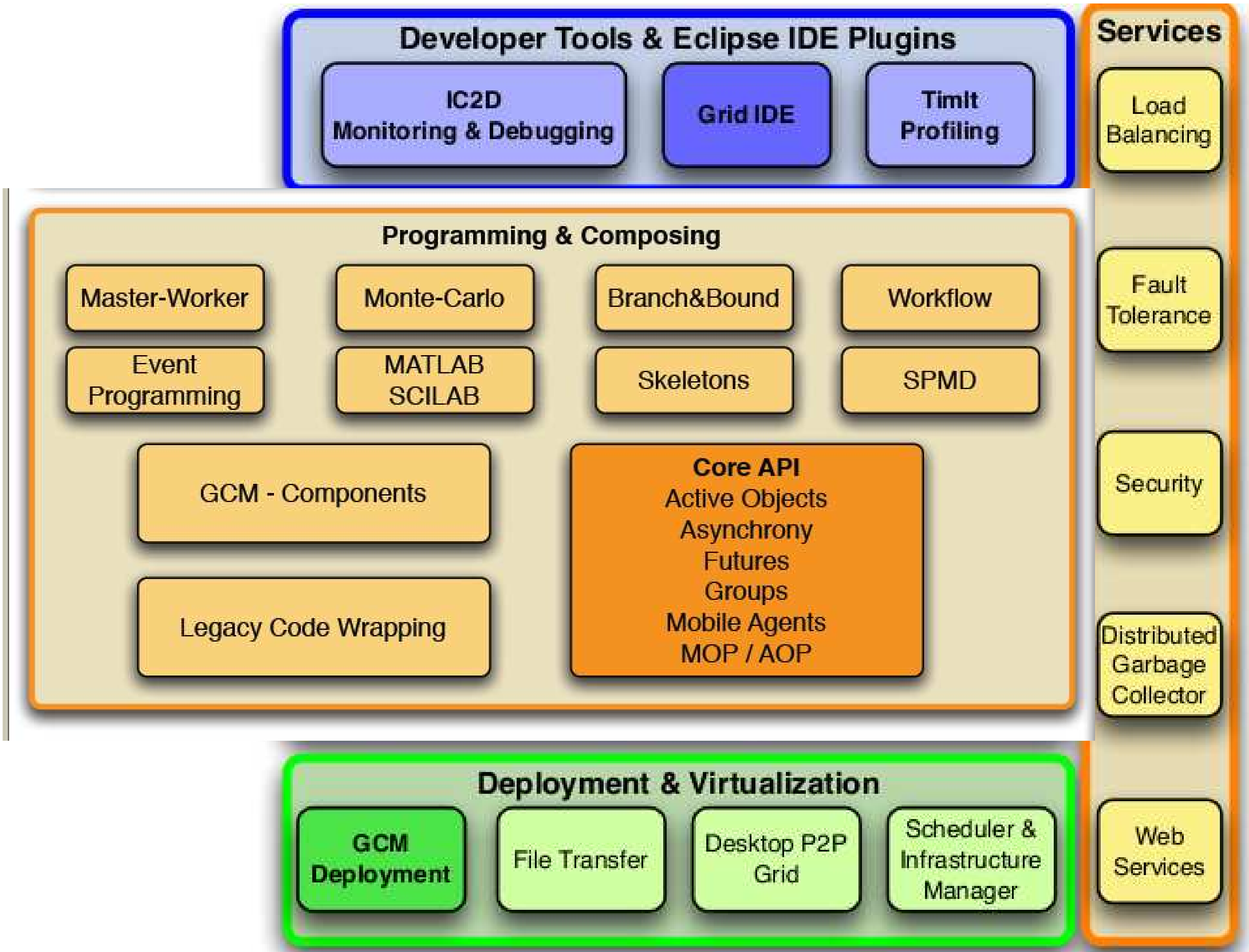


3. Client Call ...poseAsWebService ()

Agenda

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

ProActive Parallel Suite



High Level Programming models

Master-Worker Framework

Motivations

- Embarrassingly parallel problems : simple and frequent model

- Write embarrassingly parallel applications with ProActive :

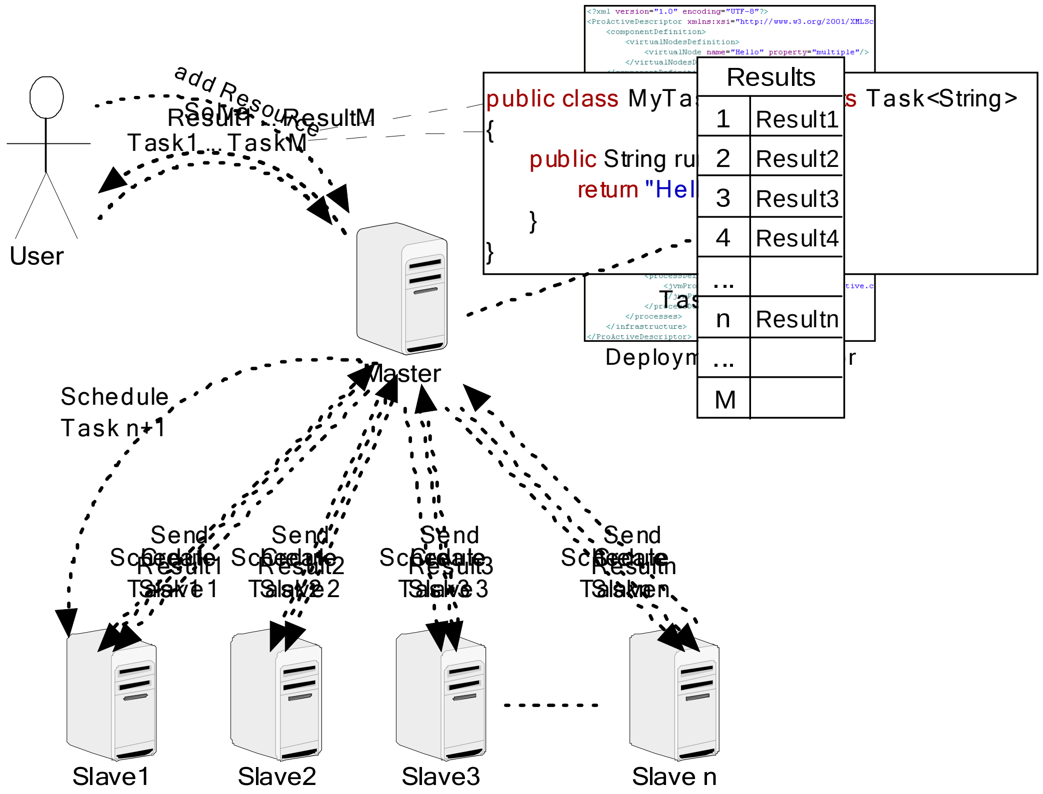
  - May require a sensible amount of code (fault-tolerance, load-balancing, …).

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

Goals of the M/W API

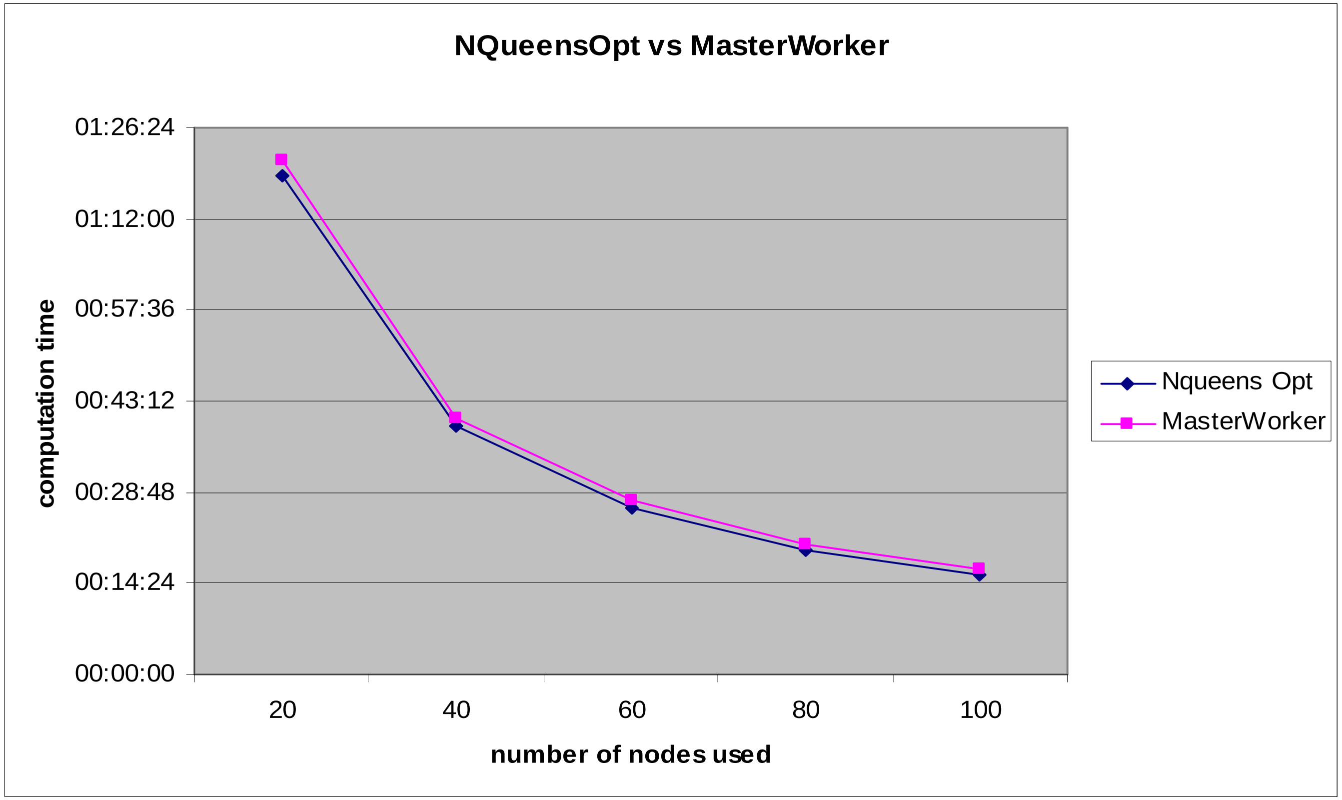
- Provide a easy-to use framework for solving embarrassingly parallel problems:
  - Simple Task definition
  - Simple API interface (few methods)
  - Simple & efficient solution gathering mechanism
- Provide automatic fault-tolerance and load-balancing mechanism
- Hide ProActive concepts from the user

How does it work?



Comparison between specific implementation and M/W

- Experiments with nQueens problem
- Runs up to 25 nodes

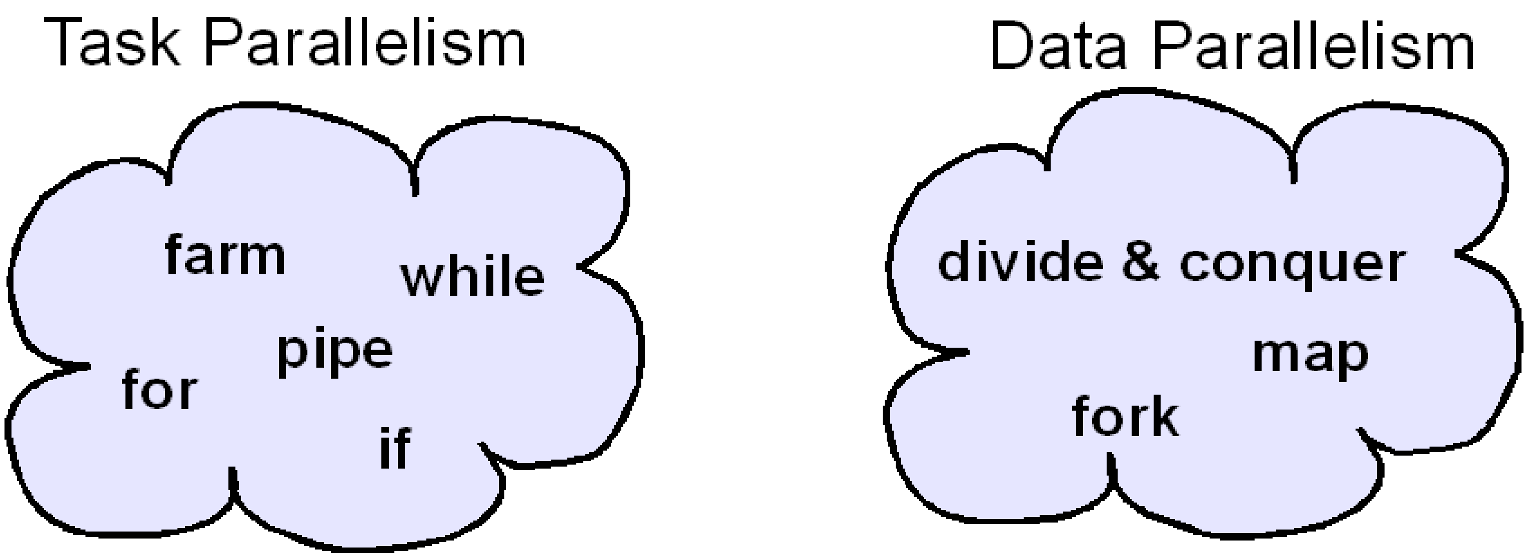


High Level Programming models

Skeletons Framework

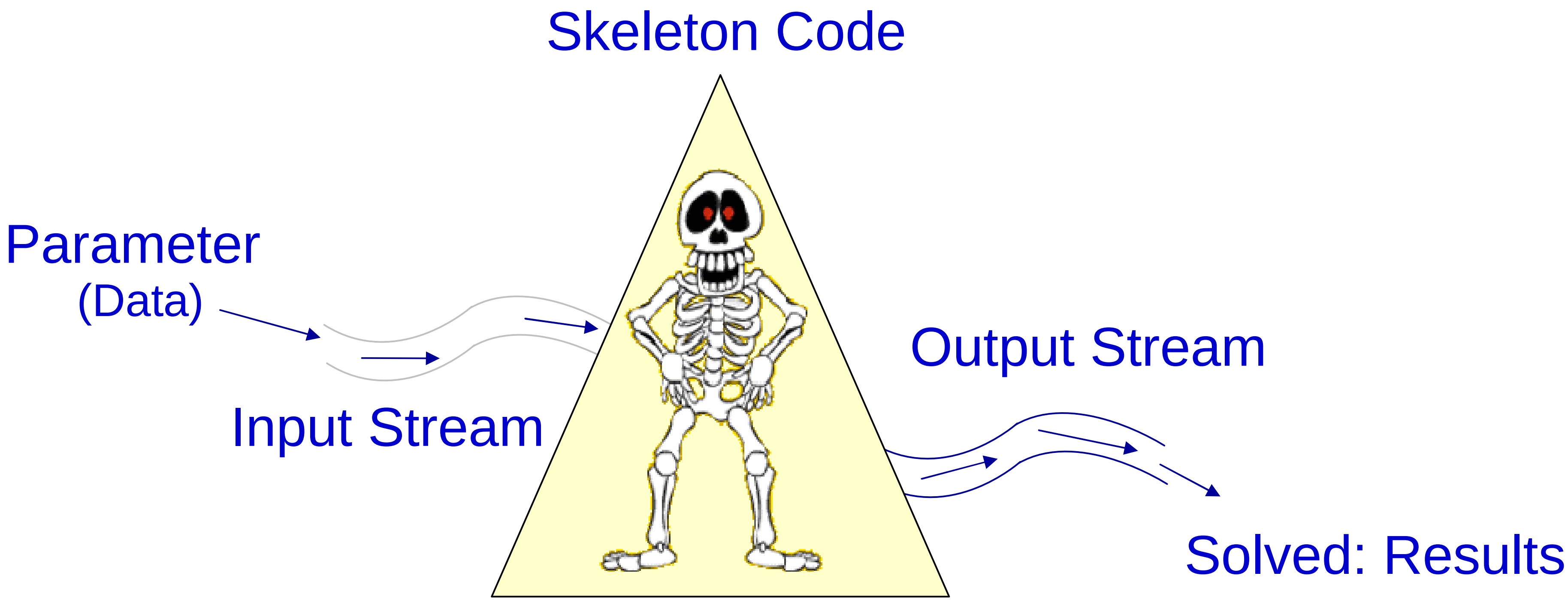
Algorithmic Skeletons

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



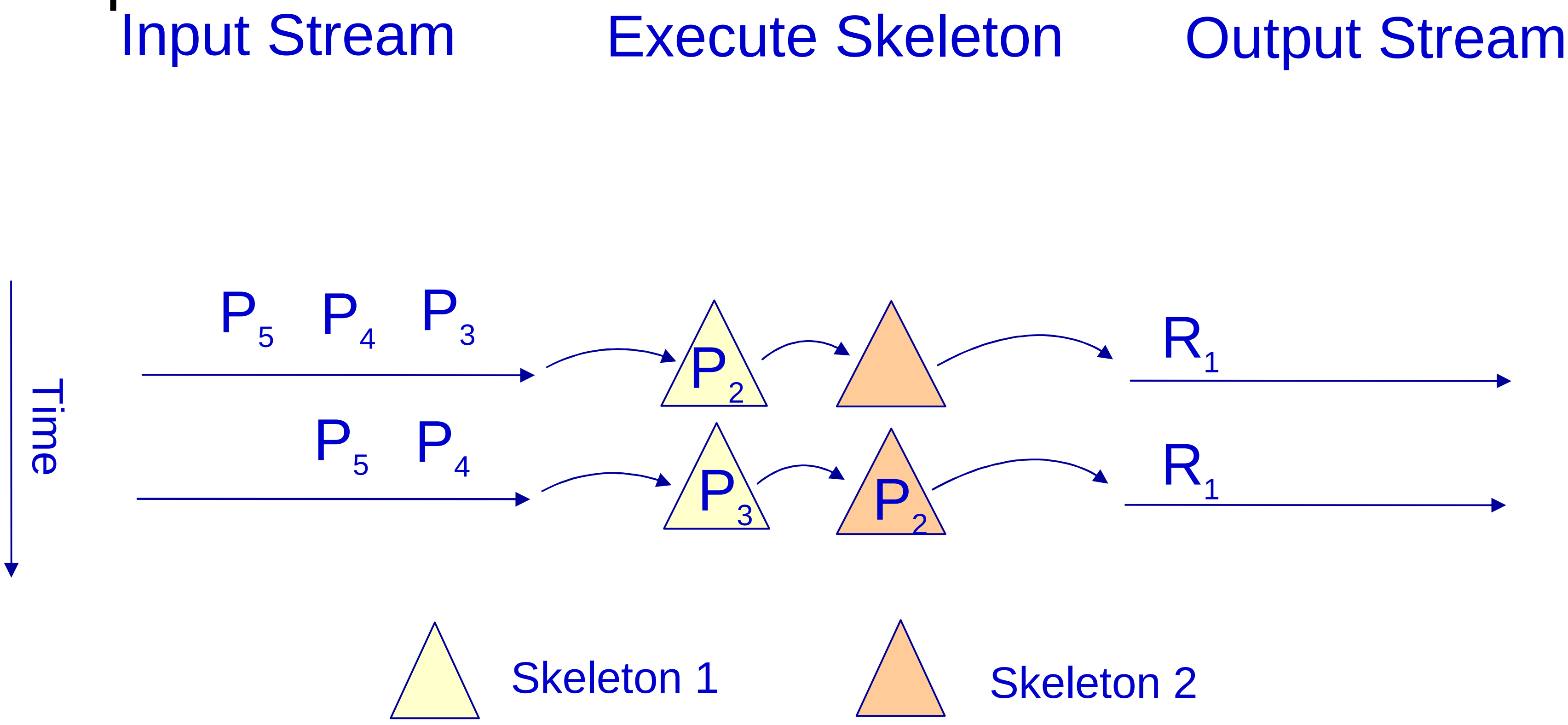
Skeletons Big Picture

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



Pipe Skeleton

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



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);
```

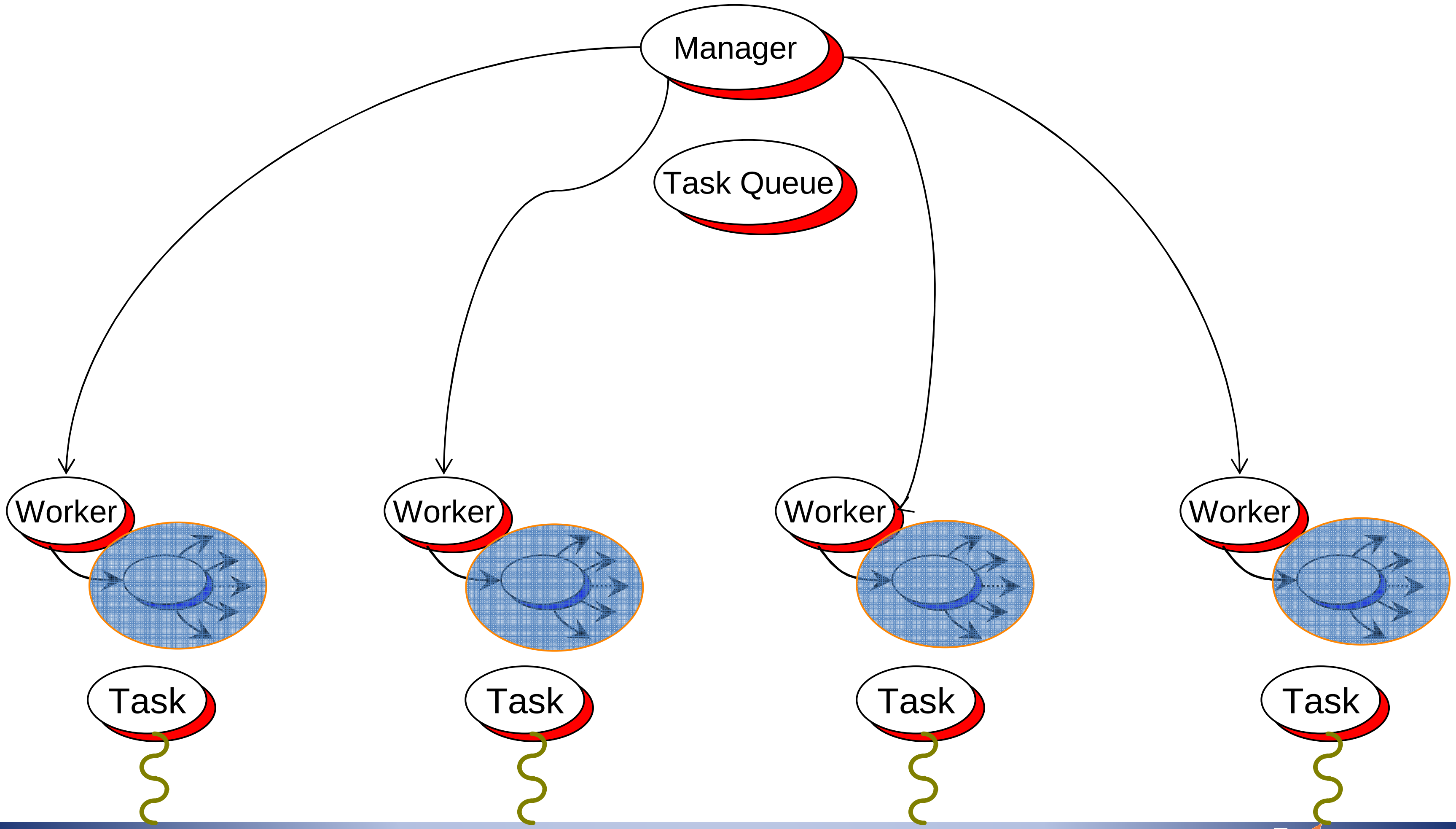
High Level Programming models

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

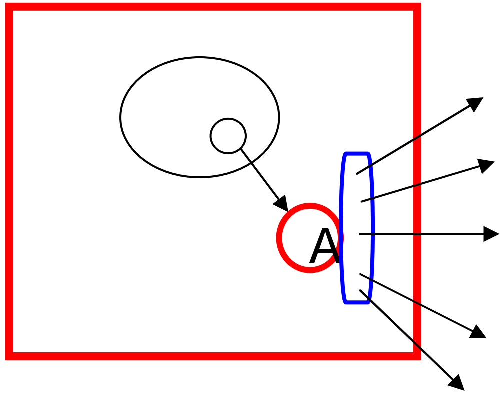
Object-Oriented Single Program Multiple Data

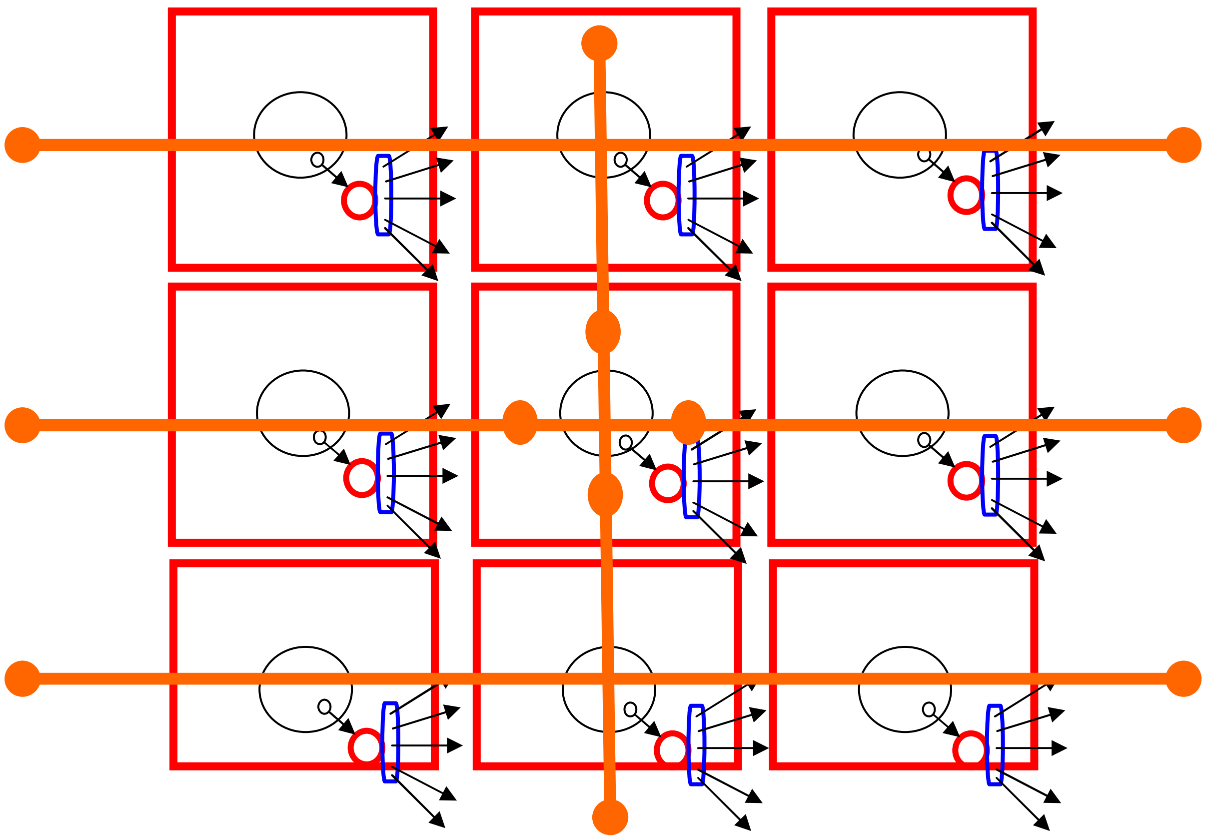
- Motivation
  - Cluster / GRID computing
  - SPMD programming for many numerical simulations
  - Use enterprise technology (Java, Eclipse, etc.) for Parallel Computing
- Able to express most of MPI's
  - Collective Communications (broadcast, gathercast, scattercast,..)
  - Barriers
  - Topologies
-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

Execution example

- A ag = newSPMDGroup ("A", […], VirtualNode)

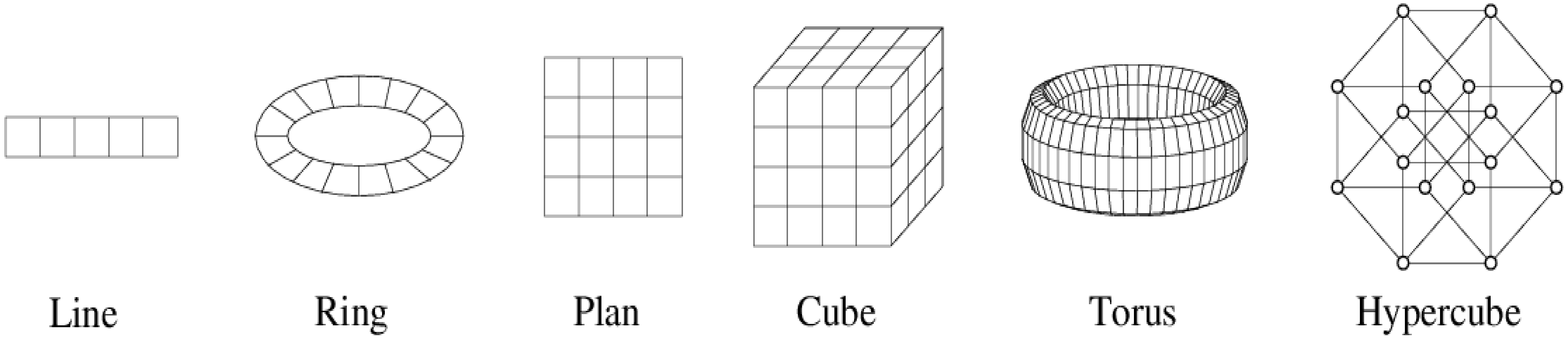
```
// In each member
myGroup.barrier ("2D"); // Global Barrier
myGroup.barrier ("vertical"); // Any Barrier
myGroup.barrier ("north","south","east","west");
```





Topologies

- Topologies are typed groups
- Customizable
- Define neighborhood



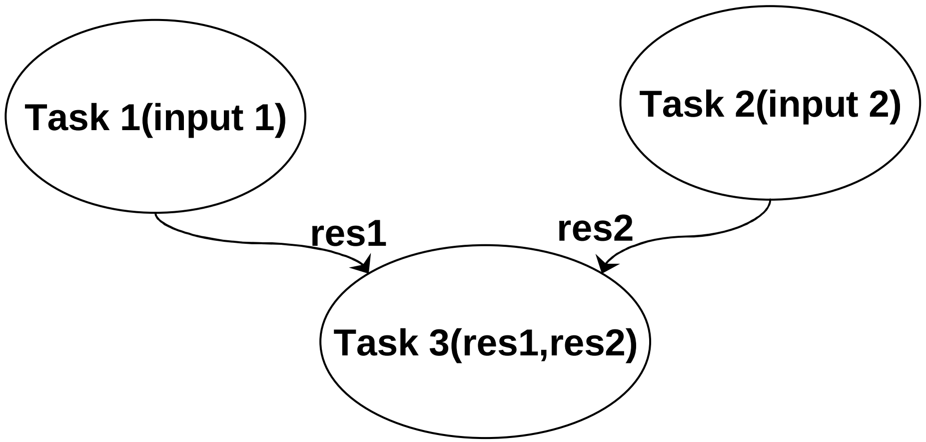

```
Plan plan = new Plan(groupA, Dimensions);
Line line = plan.getLine(0);
```

High Level Programming models

Scheduler

Programming with flows of tasks

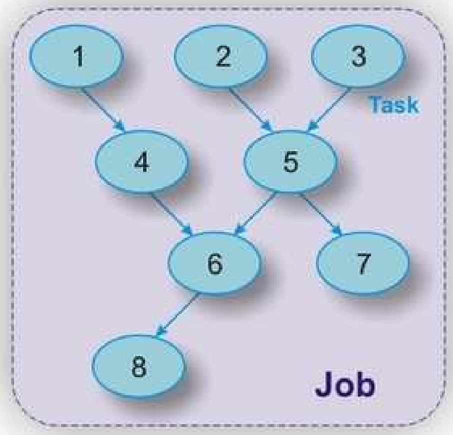
- Program an application as an ordered tasks set
  - **Logical flow** : Tasks execution are orchestrated
  - **Data flow** : Results are forwarded from ancestor tasks to their children as parameter



- 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

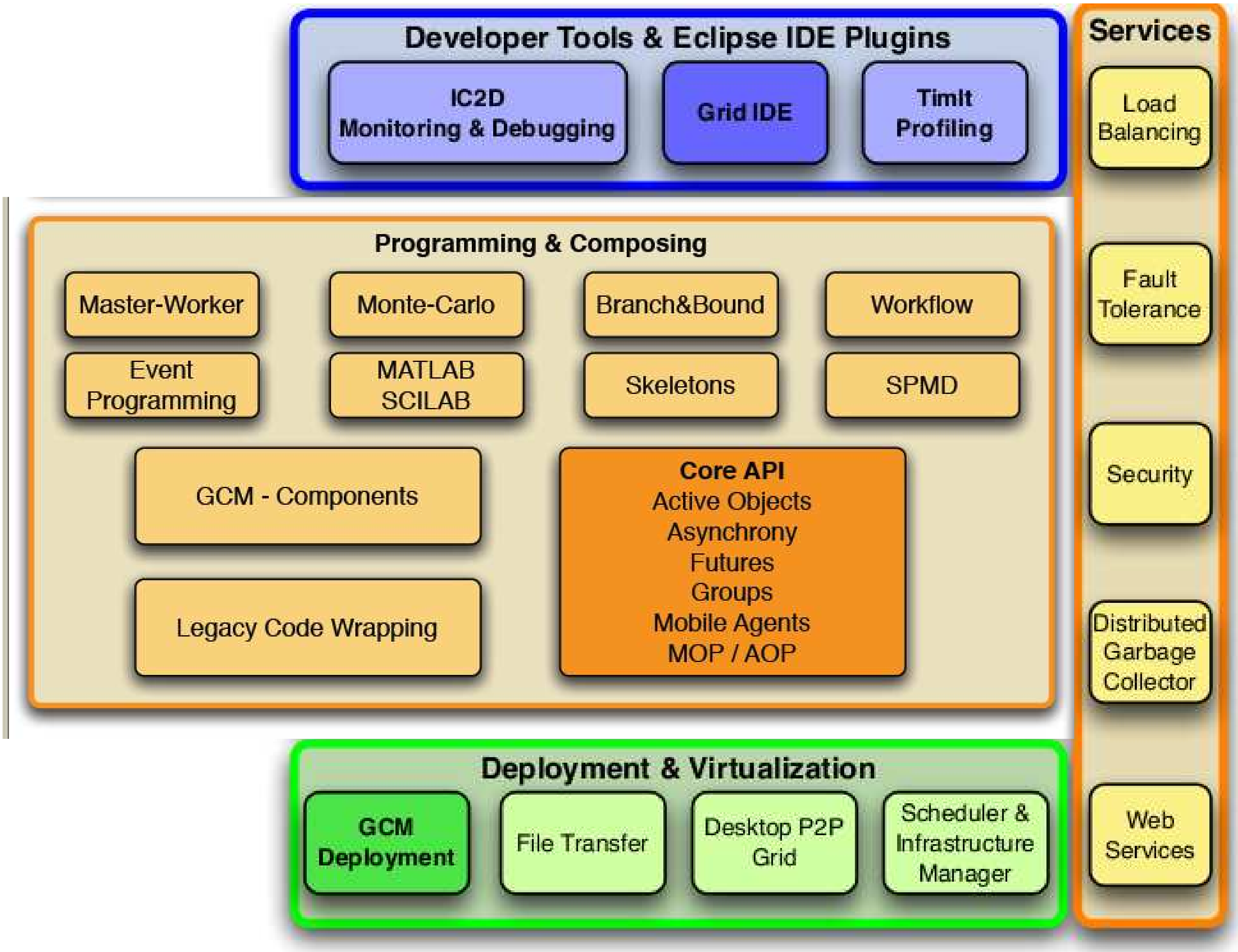
- Java or XML interface
  - Dynamic job creation in **Java**
  - Static description in **XML**

- 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



A framework for Grid components

- Facilitating the design and implementation of complex distributed systems

- Leveraging the ProActive library
  ProActive components benefit from underlying features

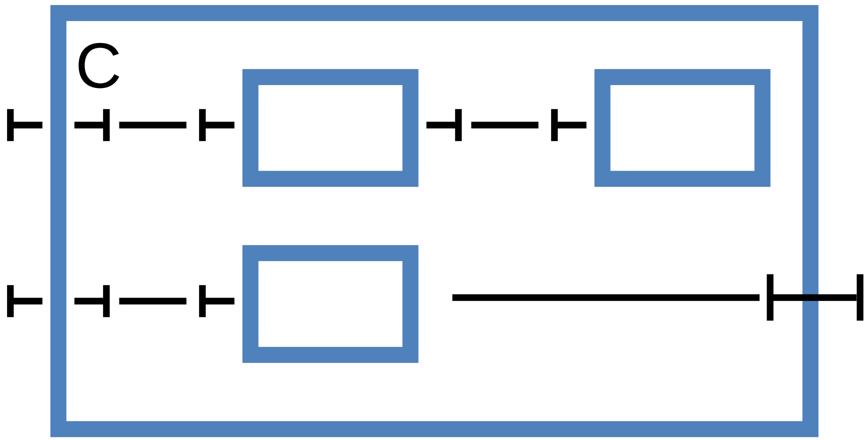
- Allowing reuse of legacy components (e.g. MPI)

- Providing tools for defining, assembling and monitoring distributed components

Component - What is it ?

- A component in a given infrastructure is:

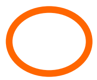
a software module,
with a standardized description of what it needs and provides,
to be manipulated by tools for Composition and Deployment



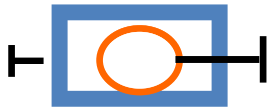
ProActive Component Definition

- 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

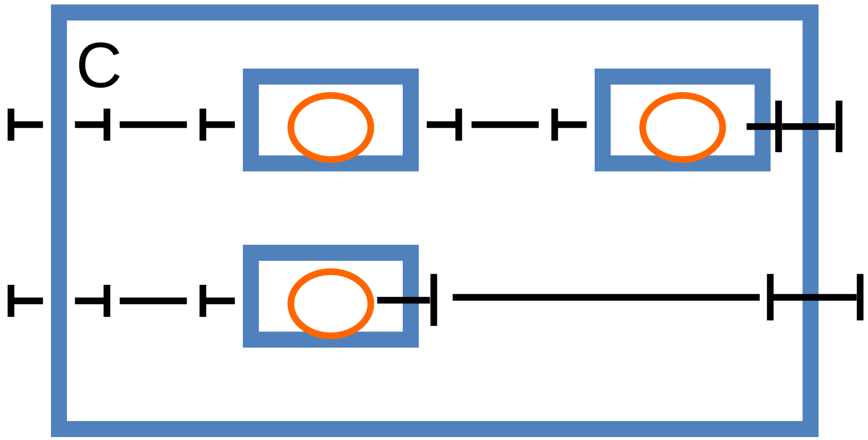
Components for the GRID



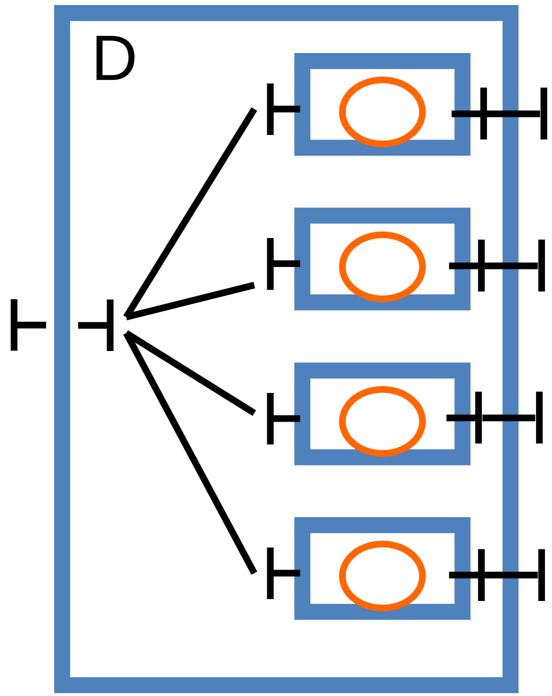
An activity, a process, …
potentially in its own JVM



1. Primitive component



2. Composite component

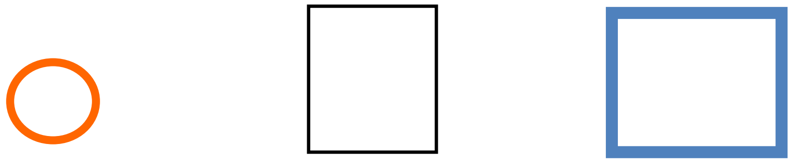


3. Parallel and composite component

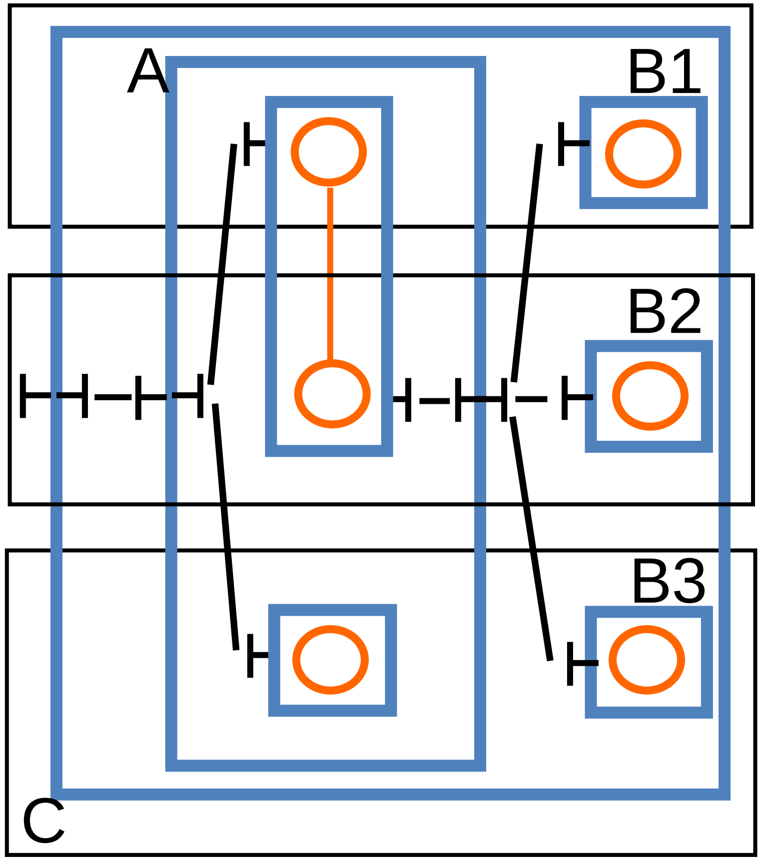
Composite: Hierarchical, and
Distributed over machines

Parallel: Composite
+ Broadcast (group)

Components vs. Activity and JVMs



Activity **JVM** **Component**

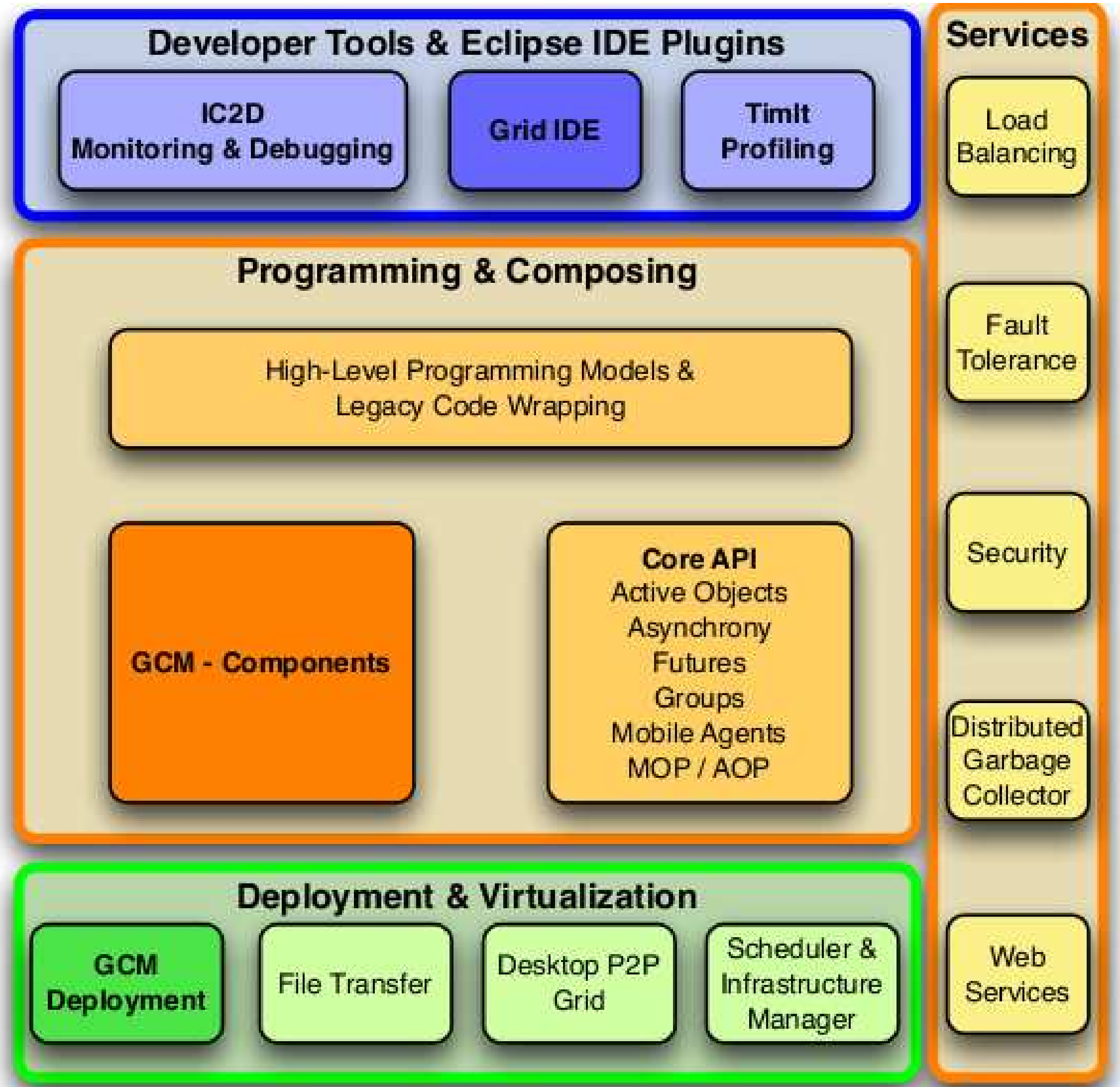


- Components are orthogonal to activities and JVMs
  - They contain activities, span across several JVMs
- Components are a way to globally manipulate distributed, and running activities

Agenda

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

GCM Deployment



Abstract Deployment Model

Problem

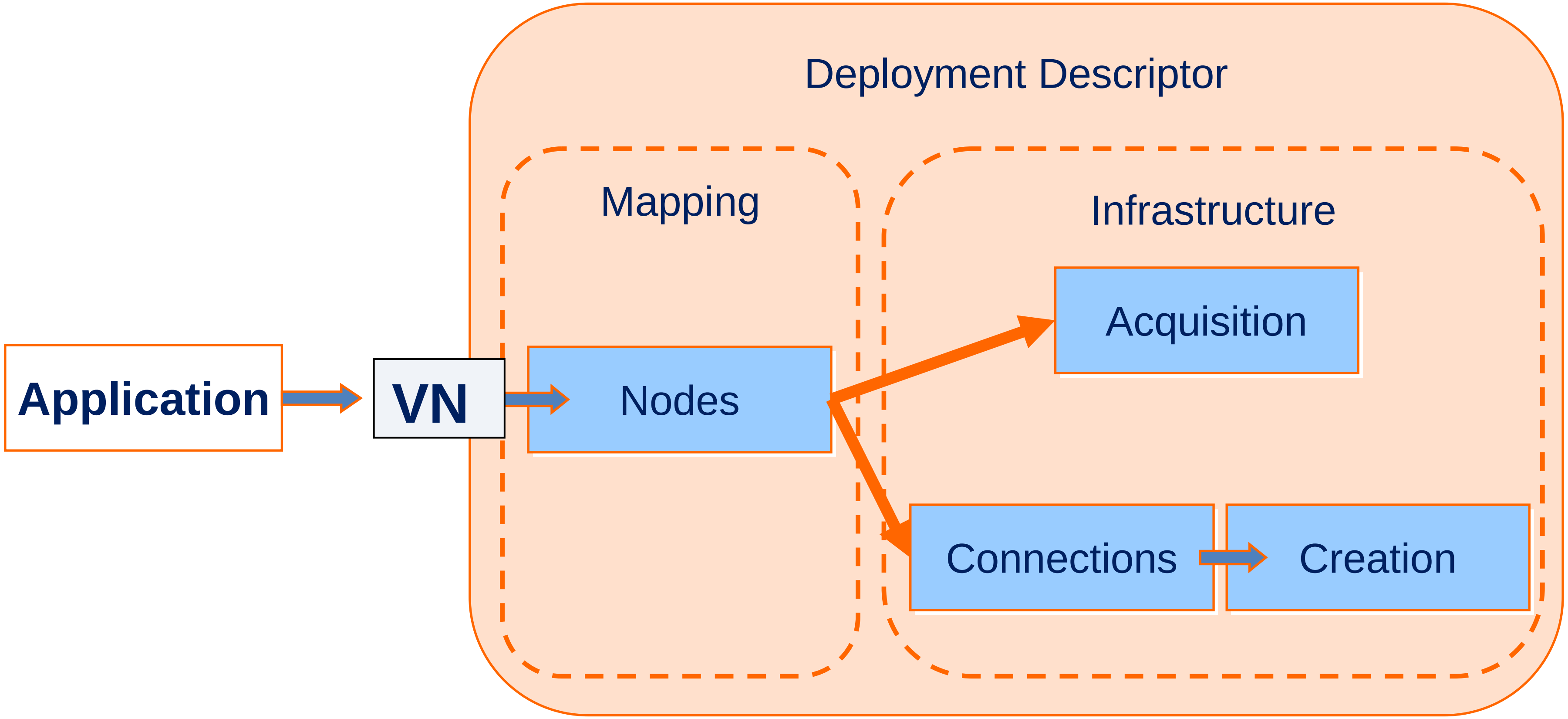
Difficulties and lack of flexibility in deployment

Avoid scripting for configuration, getting nodes, connecting…

A key principle: Virtual Node (VN)

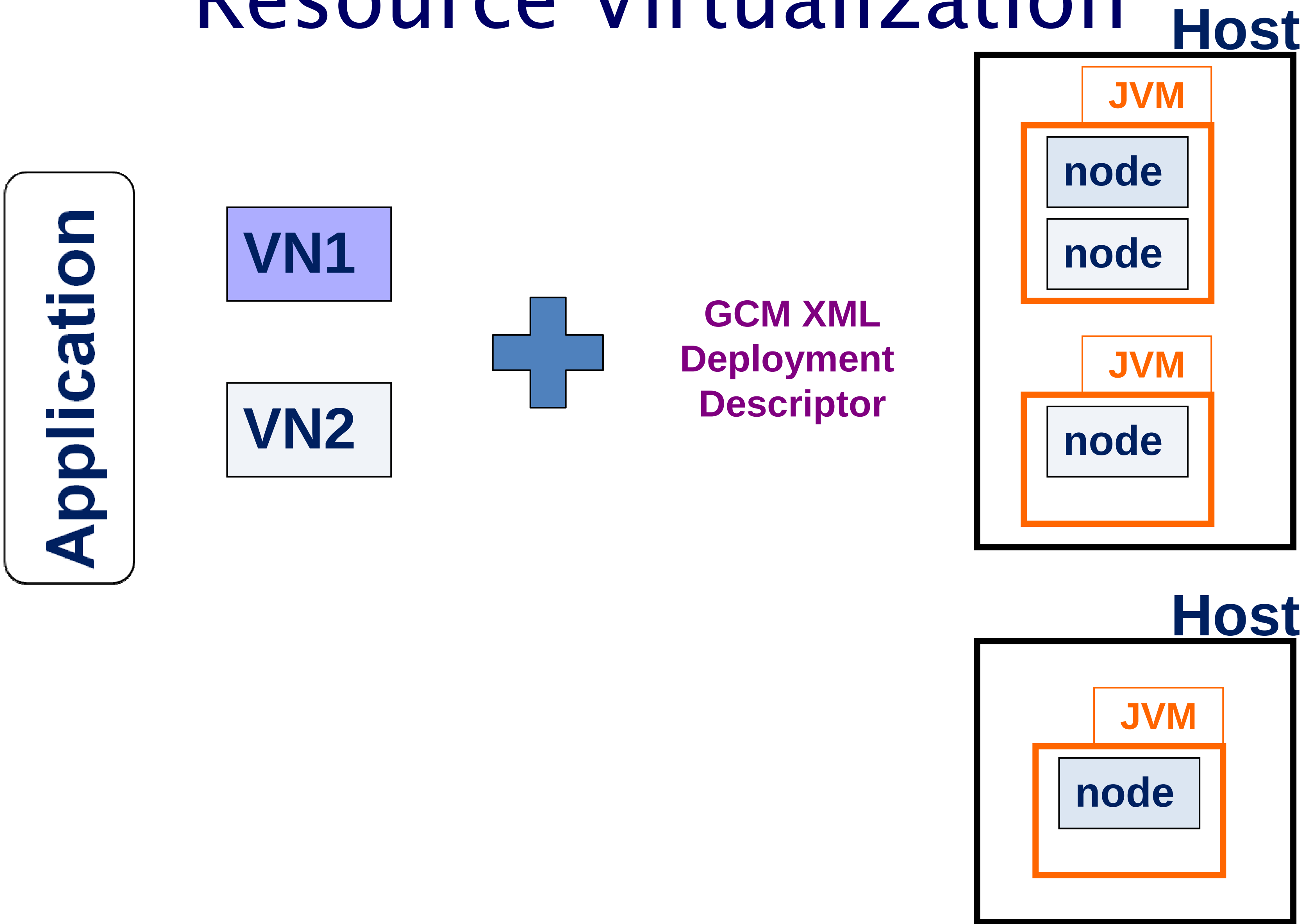
- Abstract Away from source code:
  - Machines names
  - Creation/Connection Protocols
  - Lookup and Registry Protocols
- Interface with various protocols and infrastructures:
  - Cluster: LSF, PBS, SGE , OAR and PRUN(custom protocols)
  - Intranet P2P, LAN: intranet protocols: rsh, rlogin, ssh
  - Grid: Globus, Web services, ssh, gsissh

Resource Virtualization

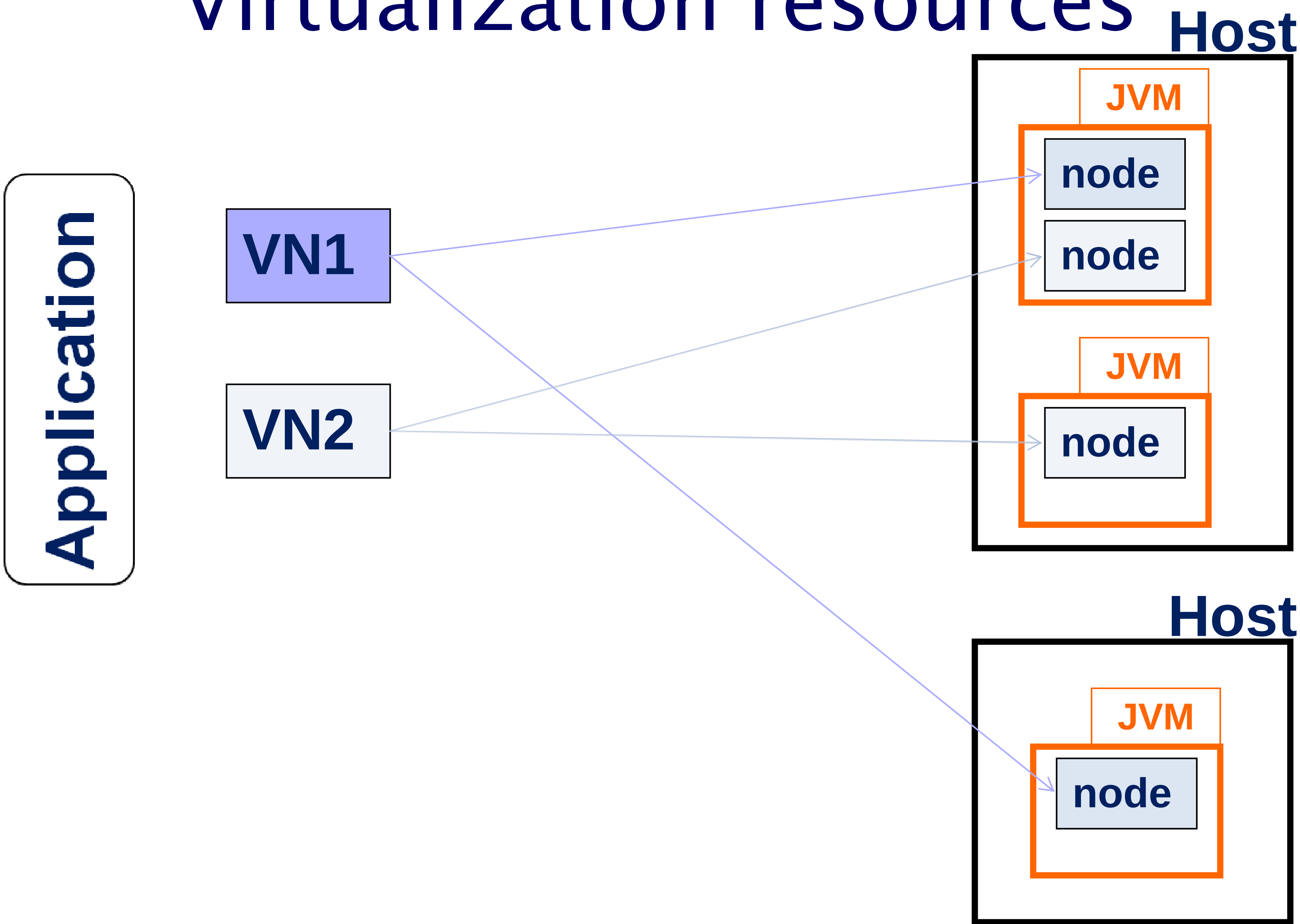


Runtime structured entities: 1 VN --> n Nodes in m JVMs on k Hosts

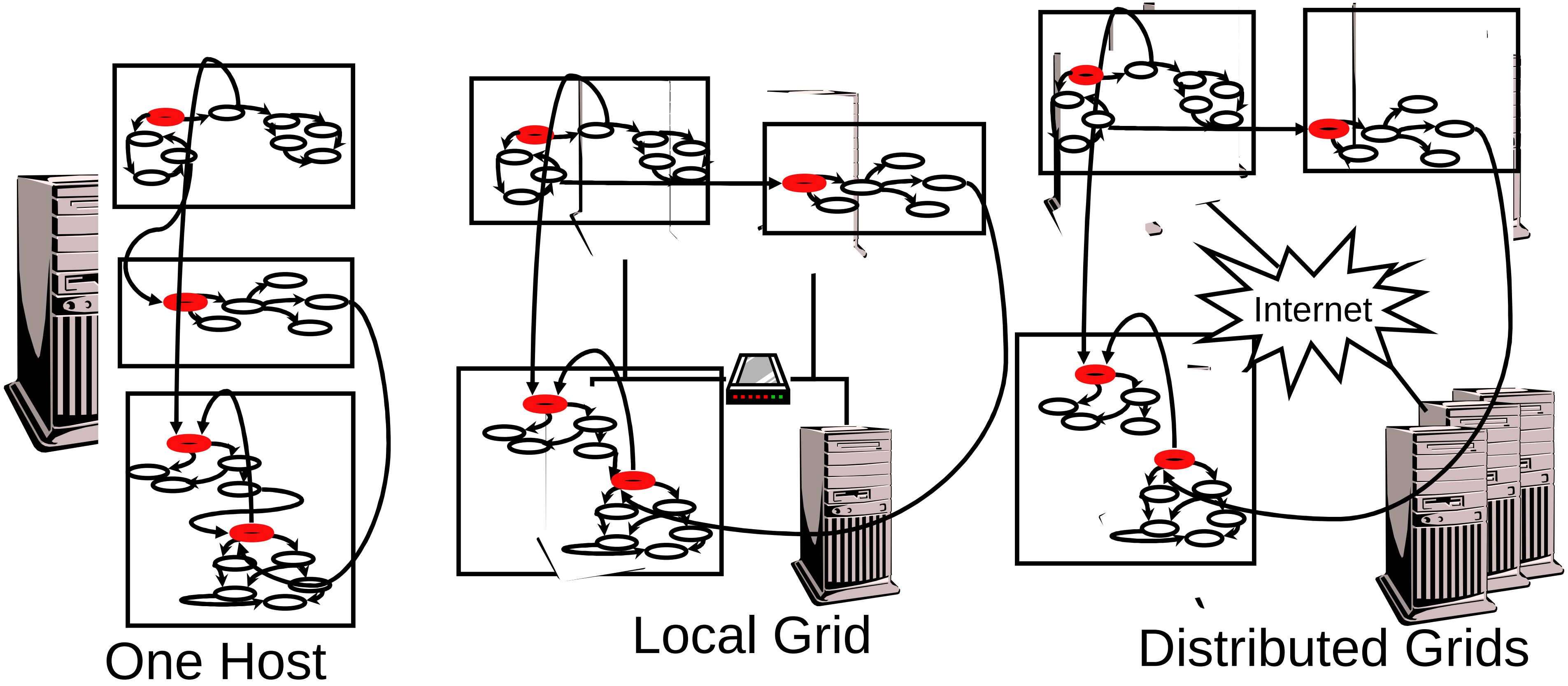
Resource Virtualization



Virtualization resources



Multiple Deployments



Rmissh : SSH Tunneling

- **A fact : overprotected clusters**
  - Firewalls prevent incoming connections
  - Use of private addresses
  - NAT, IP Address filtering, …

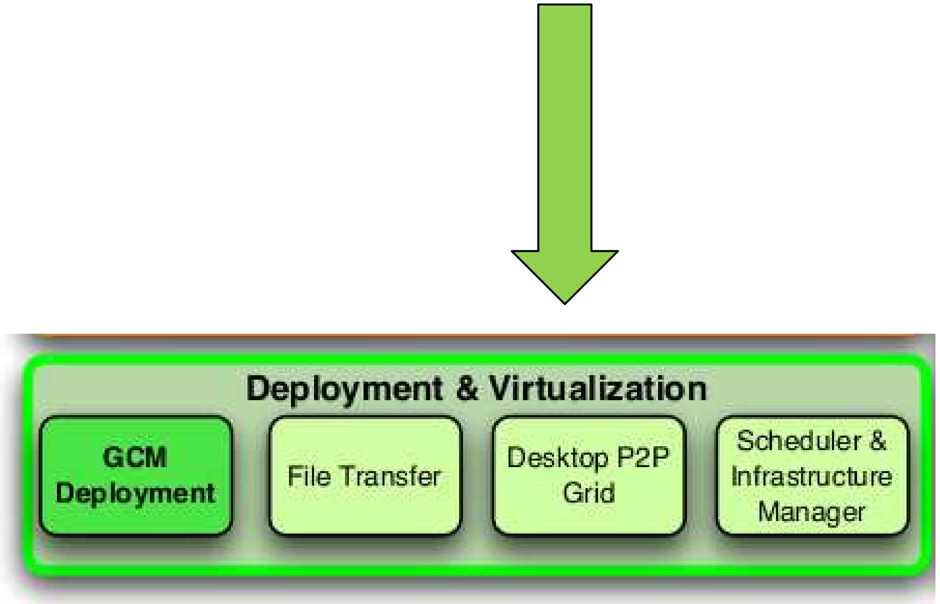
- **A consequence :**
  - Multi clustering is a nightmare

- **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

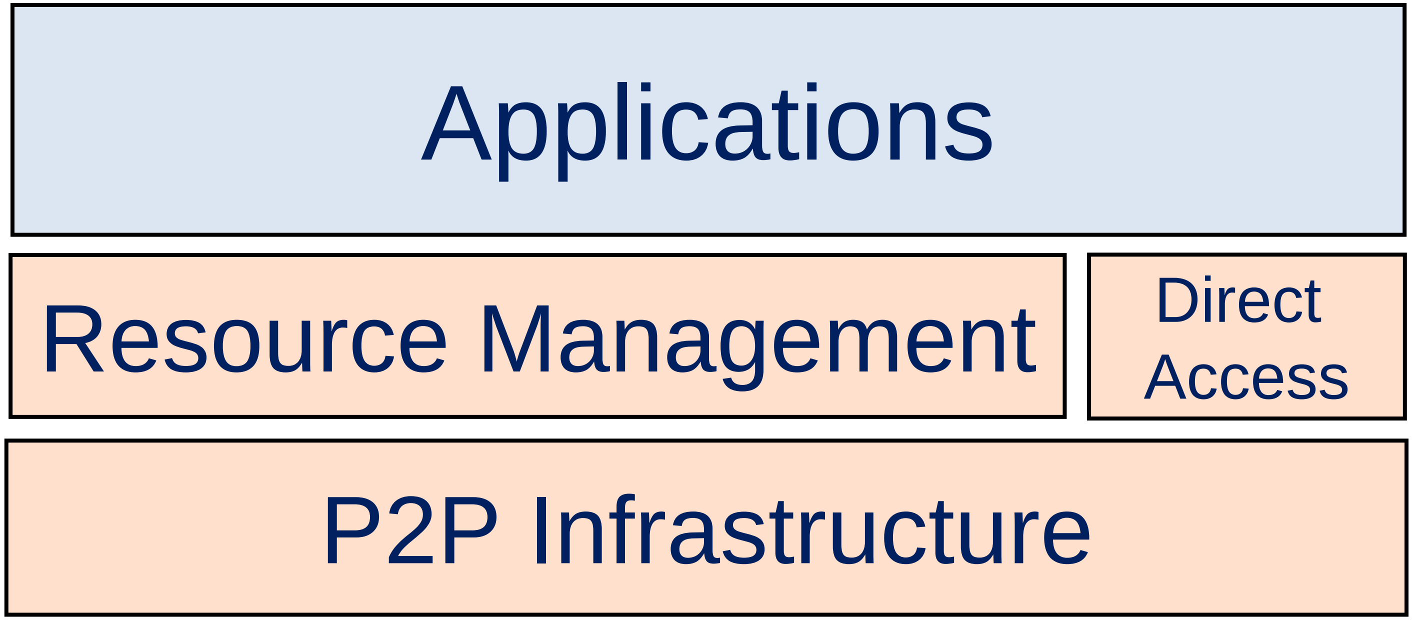
The ProActive P2P



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)



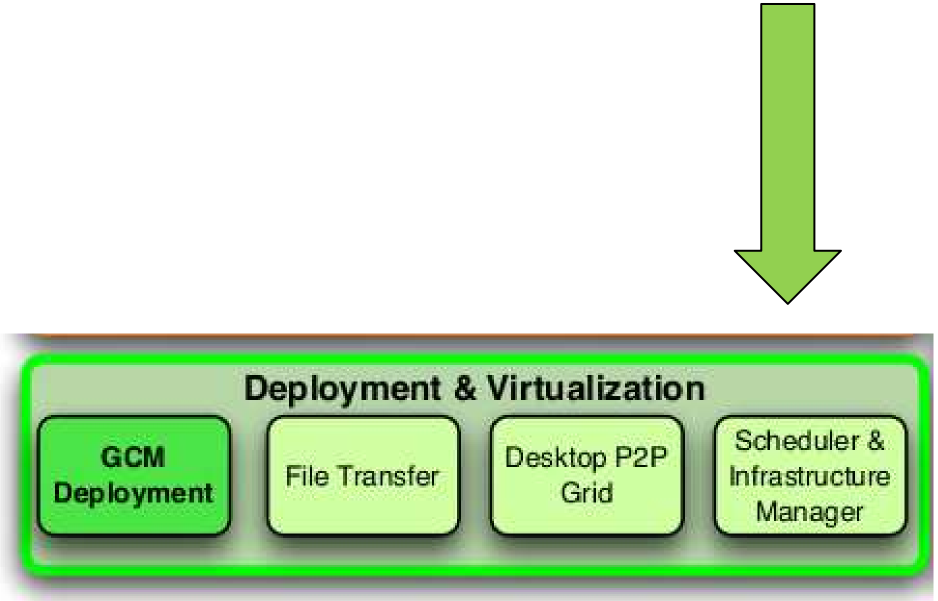
Infrastructure

- 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

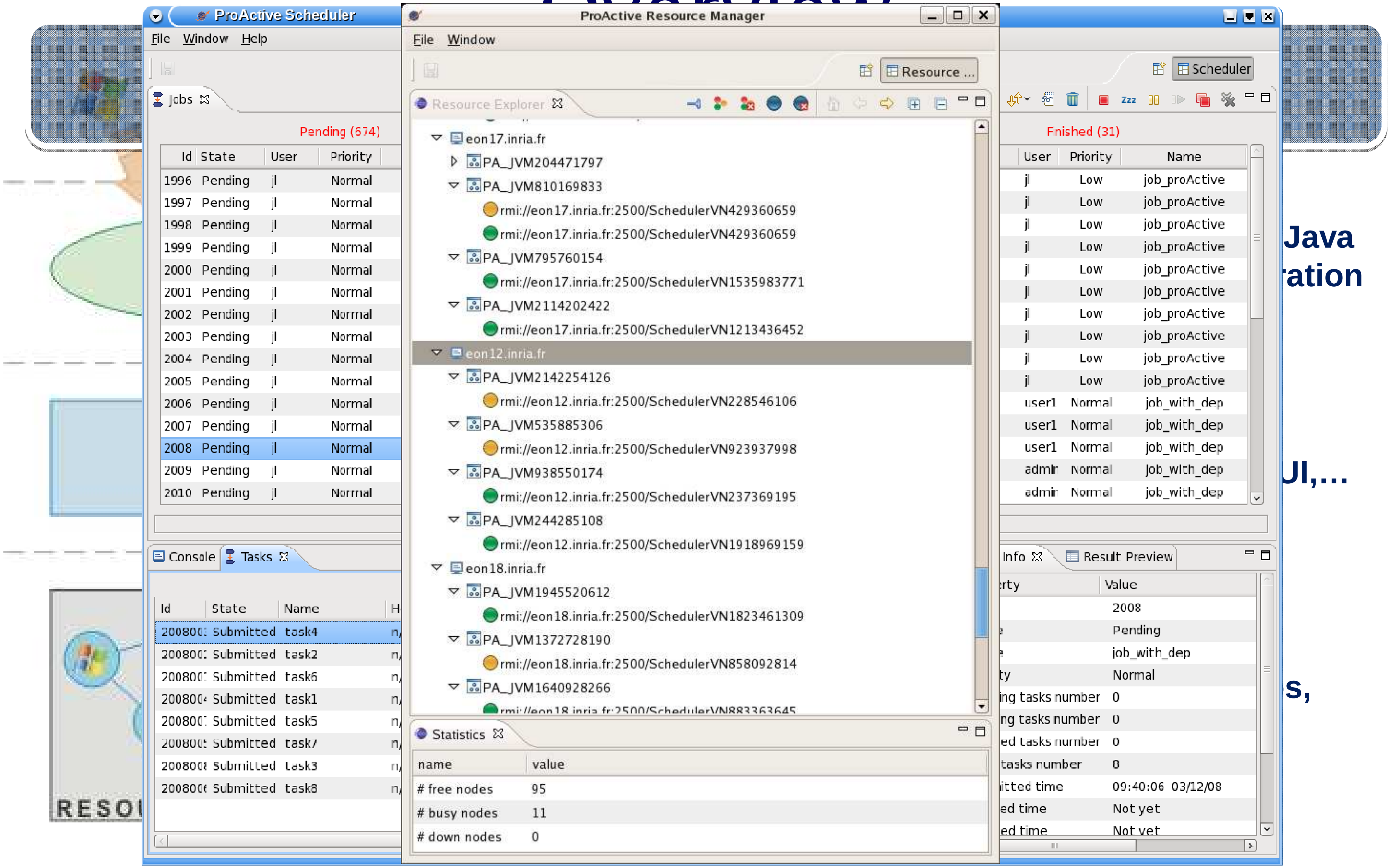
Requesting Nodes

- 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



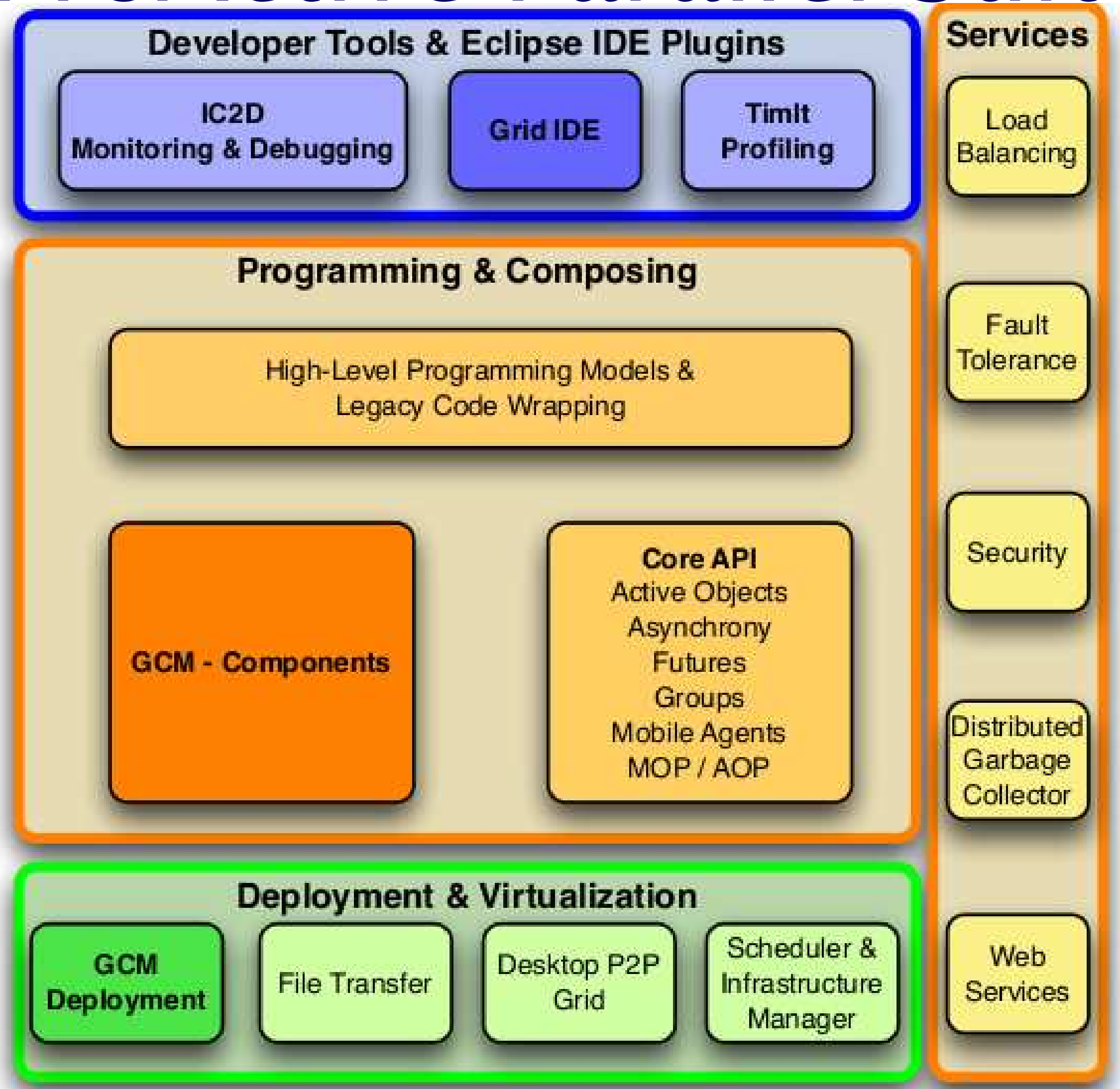
Scheduler / Resource Manager Overview



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

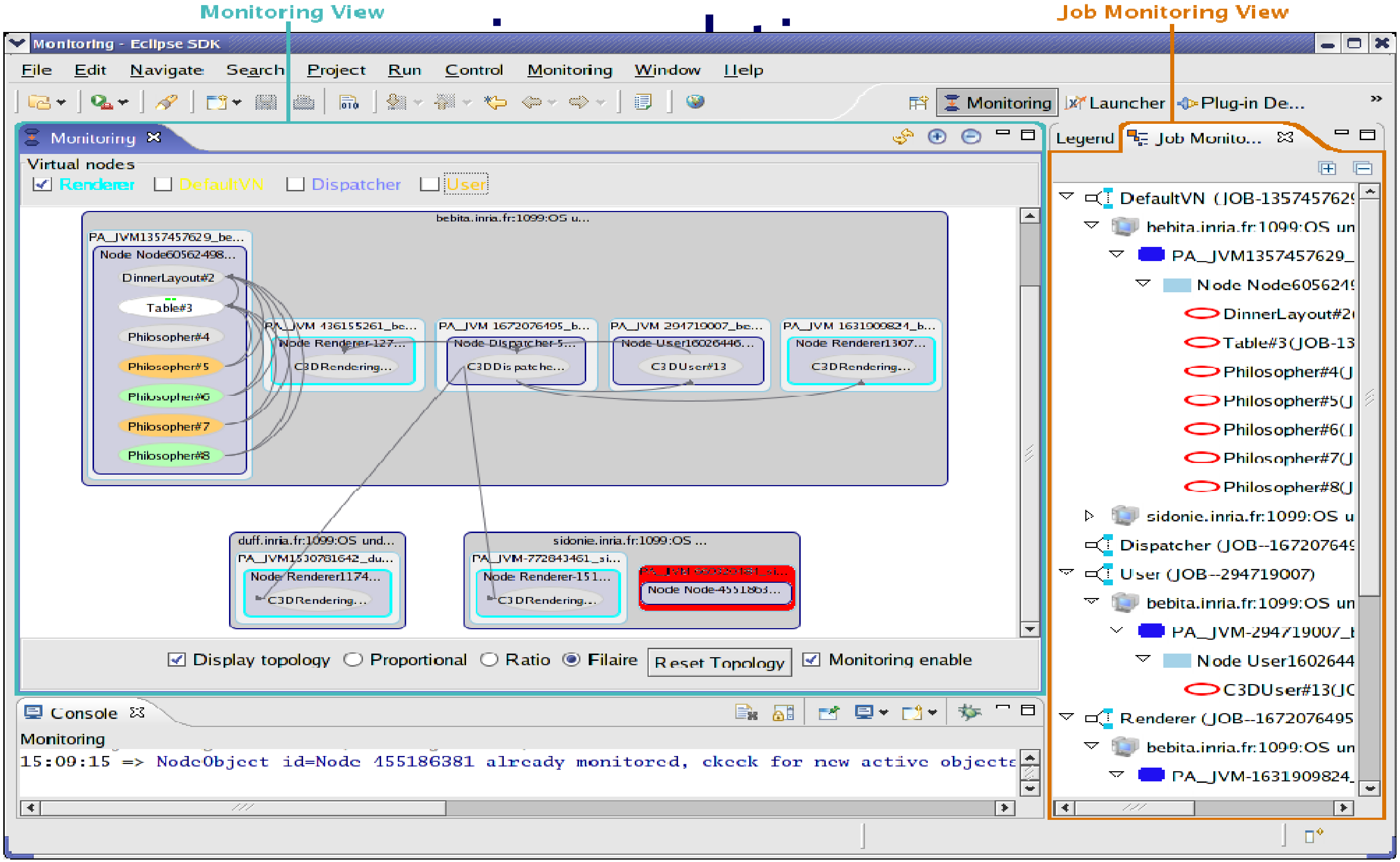


IC2D

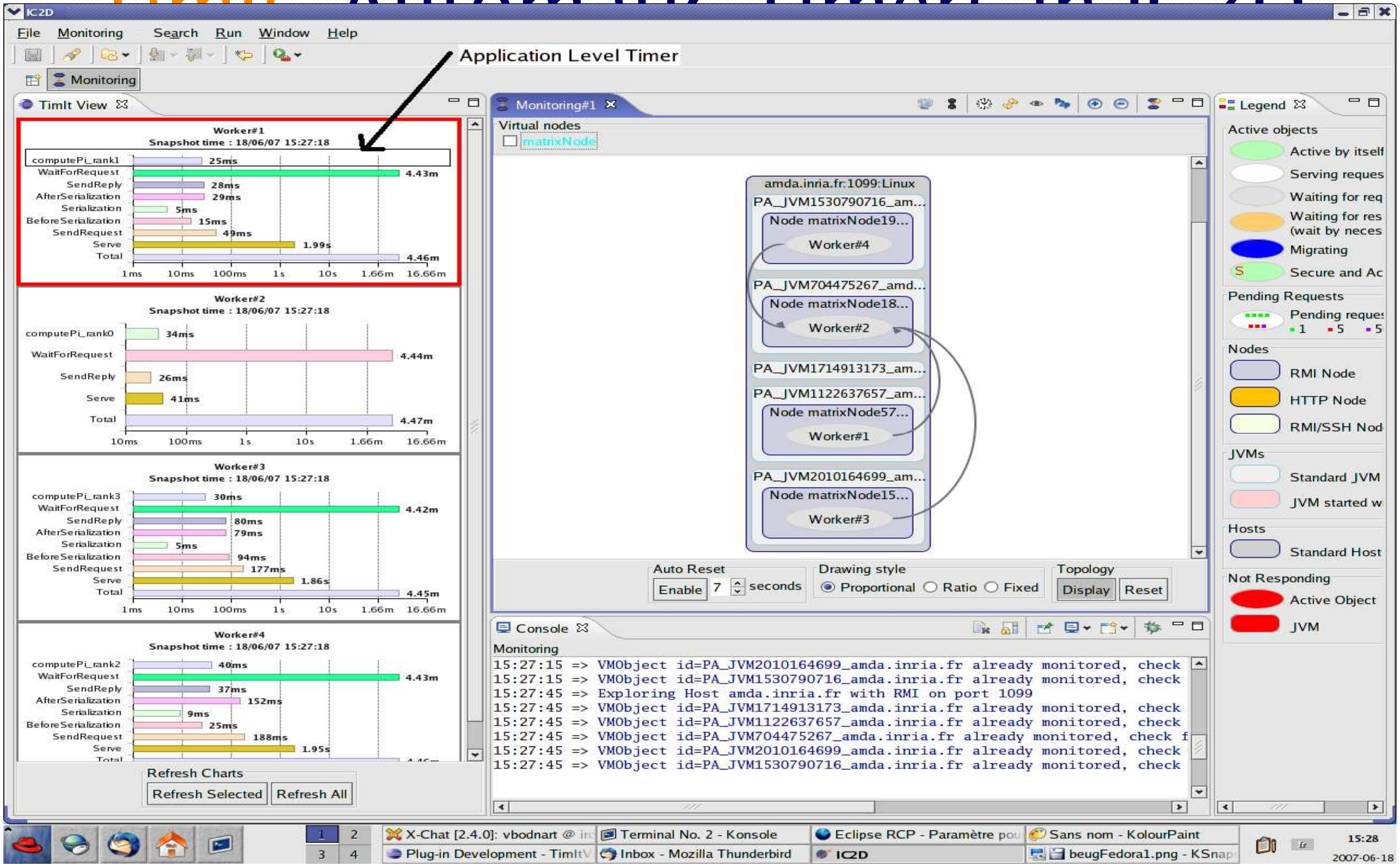
Interactive Control & Debug for Distribution

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

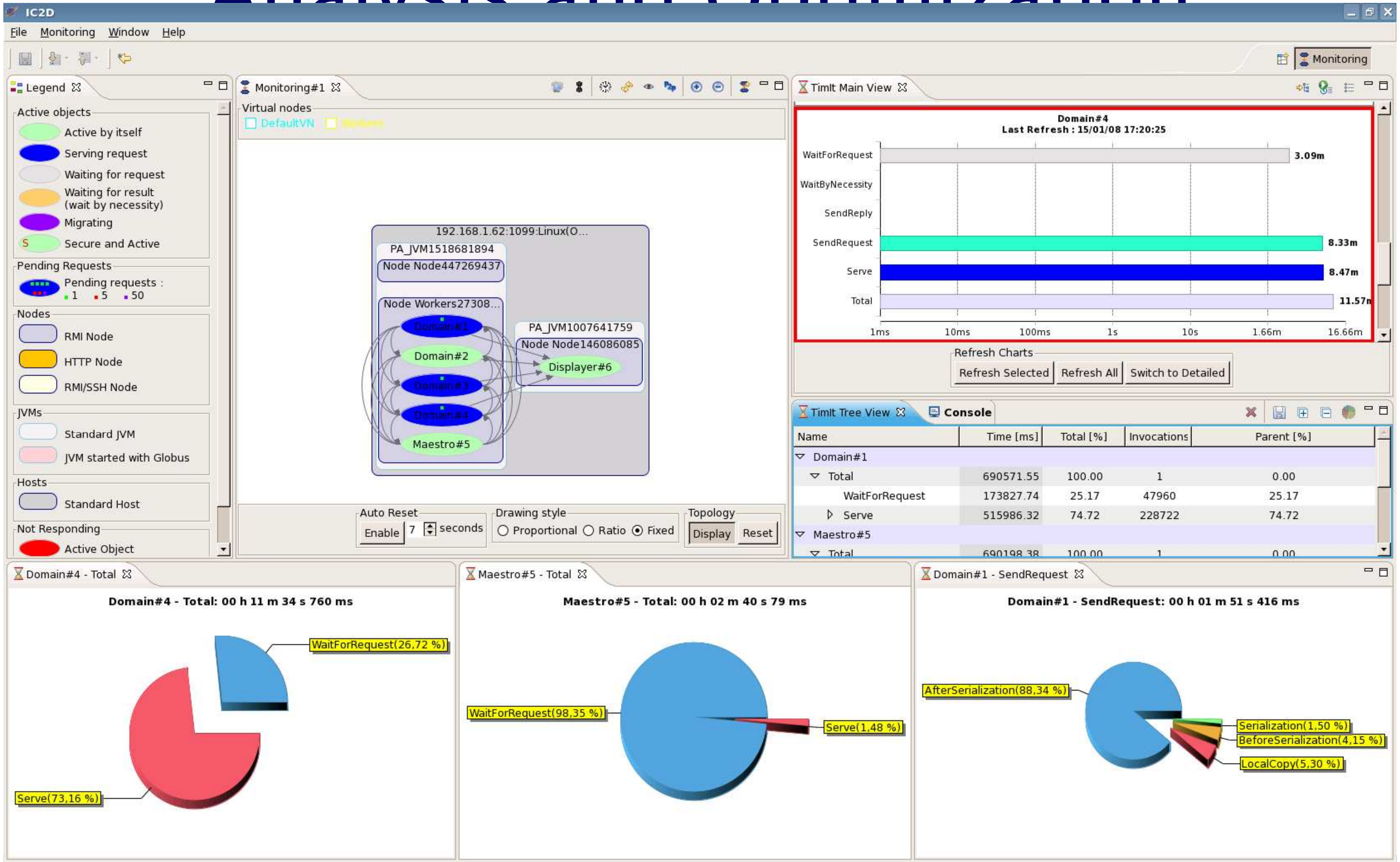
IC2D: Monitor your application



TimIt: Automatic Timers in IC2D



Analysis and Optimization

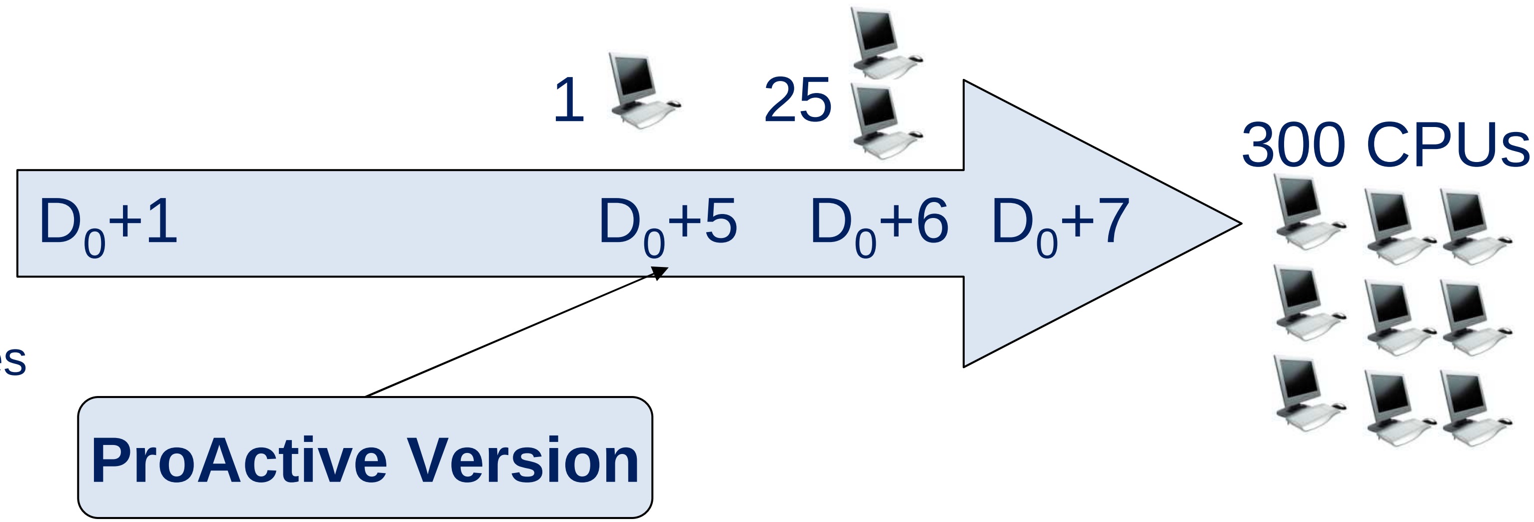


M/W Success Story: Artificial Life Generation

Sylvain Cussat-Blanc, Yves Duthen – IRIT TOULOUSE

Application

Development of artificial creatures



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

Price-It workload distribution with ProActive

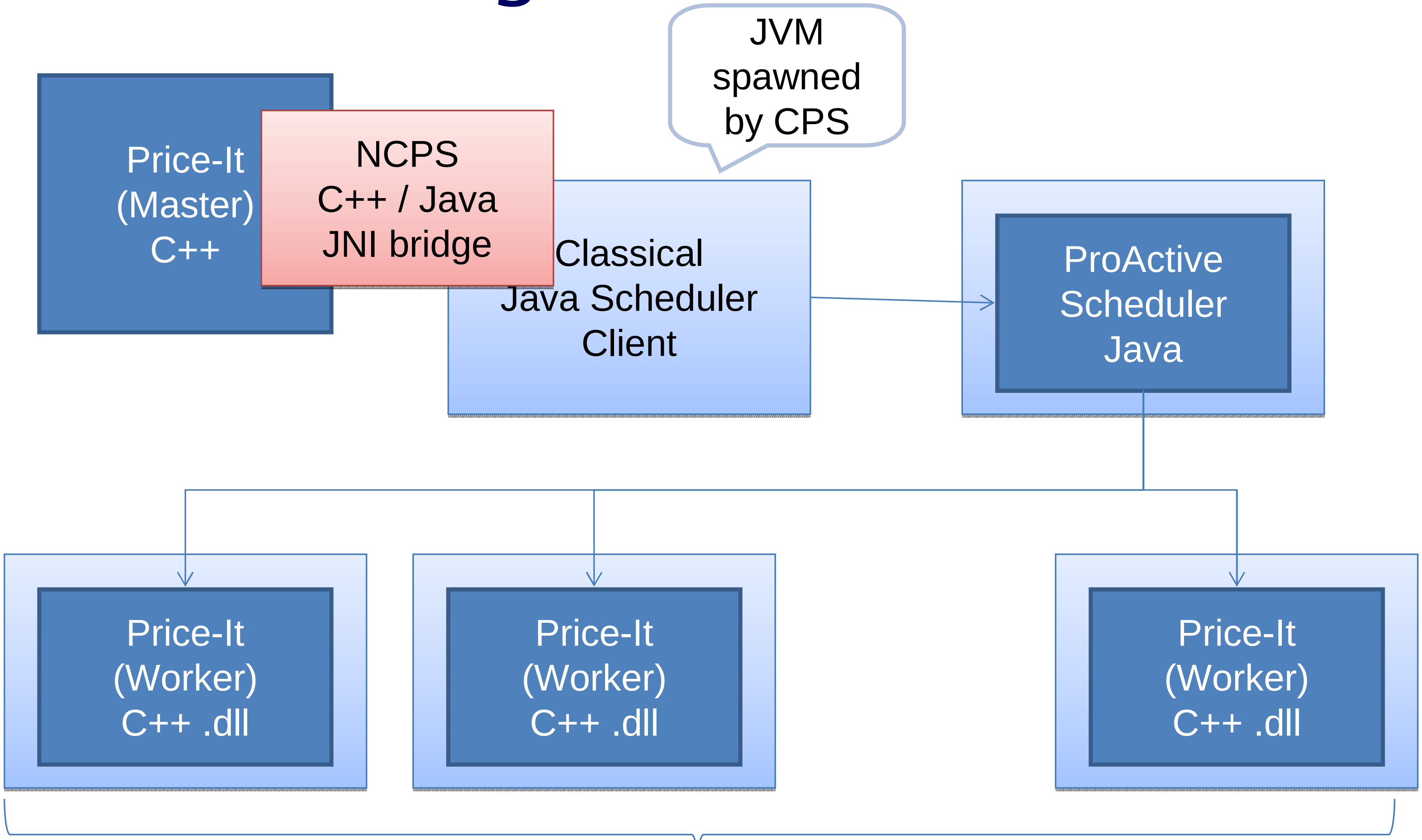
- Low level parallelism : shared memory
- Written in c++
- Originally written for microsoft compiler
- JNI, Com interface
- No thread safe

- 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



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`)