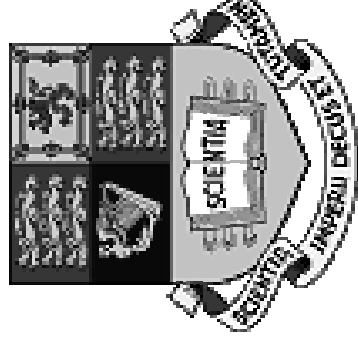


Integrating Java/ Jini into existing Grid Systems

Steven Newhouse

s.newhouse@doc.ic.ac.uk



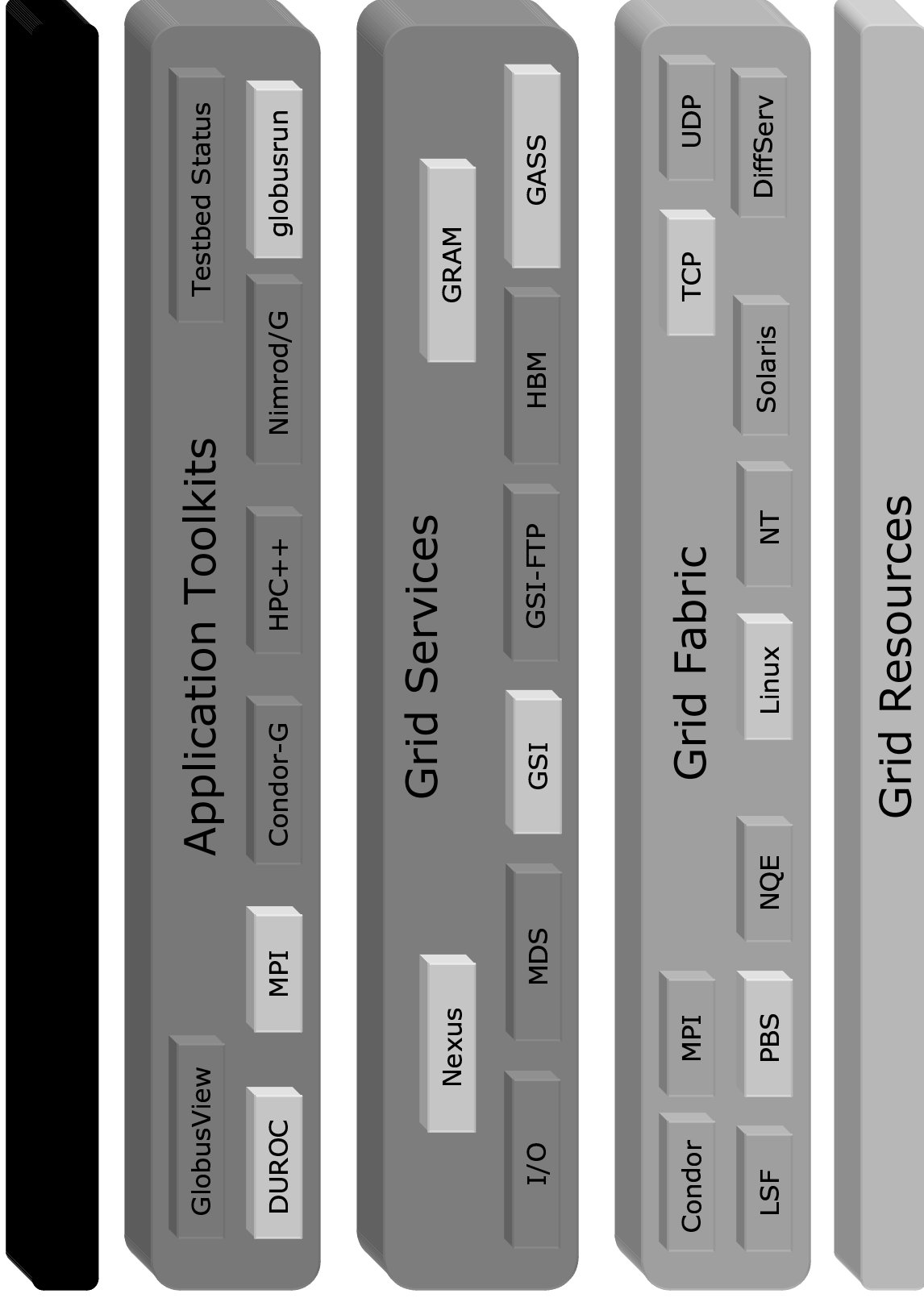
ParallelSoftwareGroup/
ImperialCollegeParallelComputingCentre/
LondonE -ScienceCentre

DepartmentofComputing
ImperialCollege,London

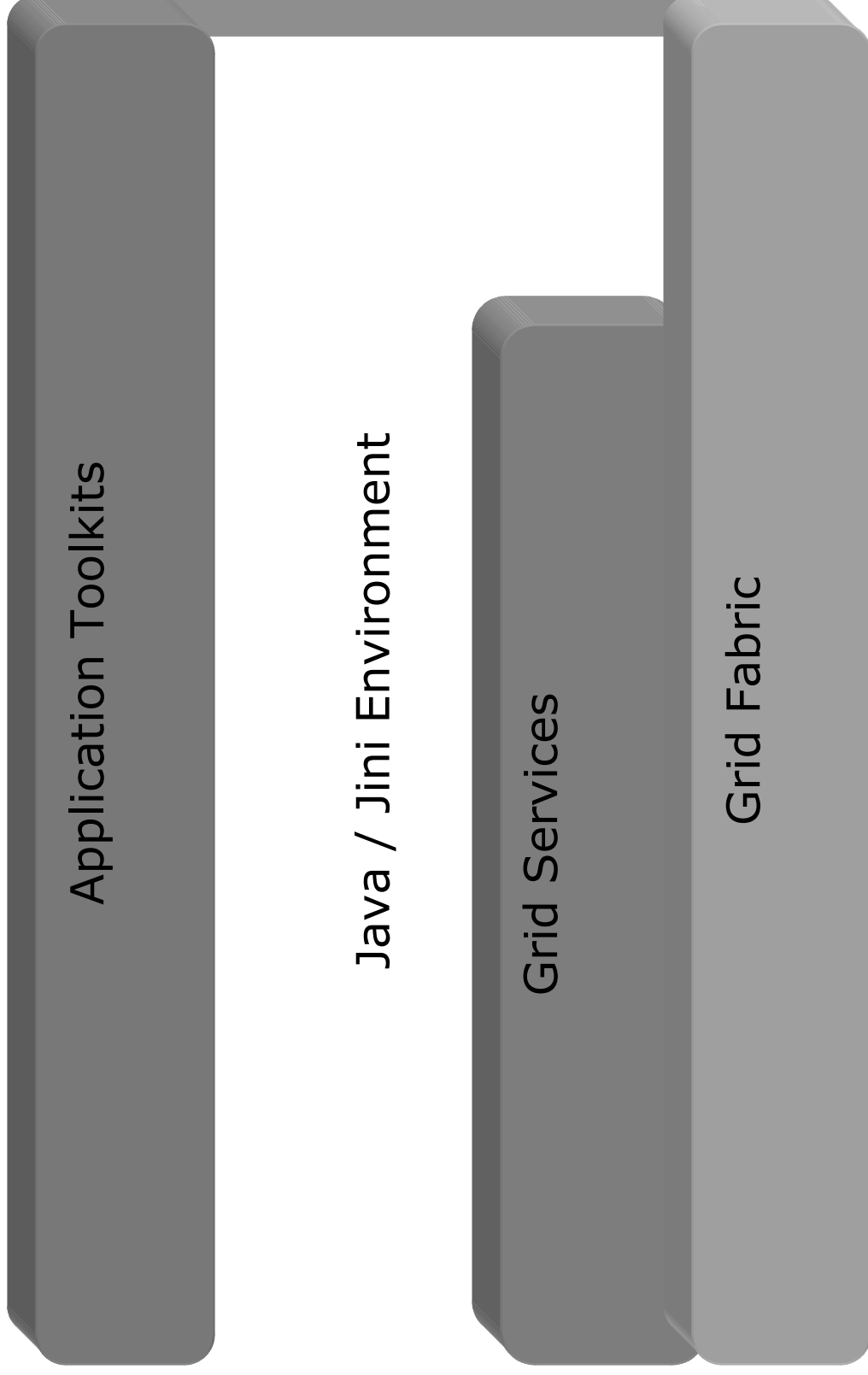
The Grid Story So Far...

- Globus
- Globus
- Globus
- What's wrong with Globus?
 - Implementation and deployment issues NOT design.
- What's missing from the Grid (and Globus!)?
 - Essential higher level services such as policy and scheduling.

Layered Architecture



Exploit Layered Architecture



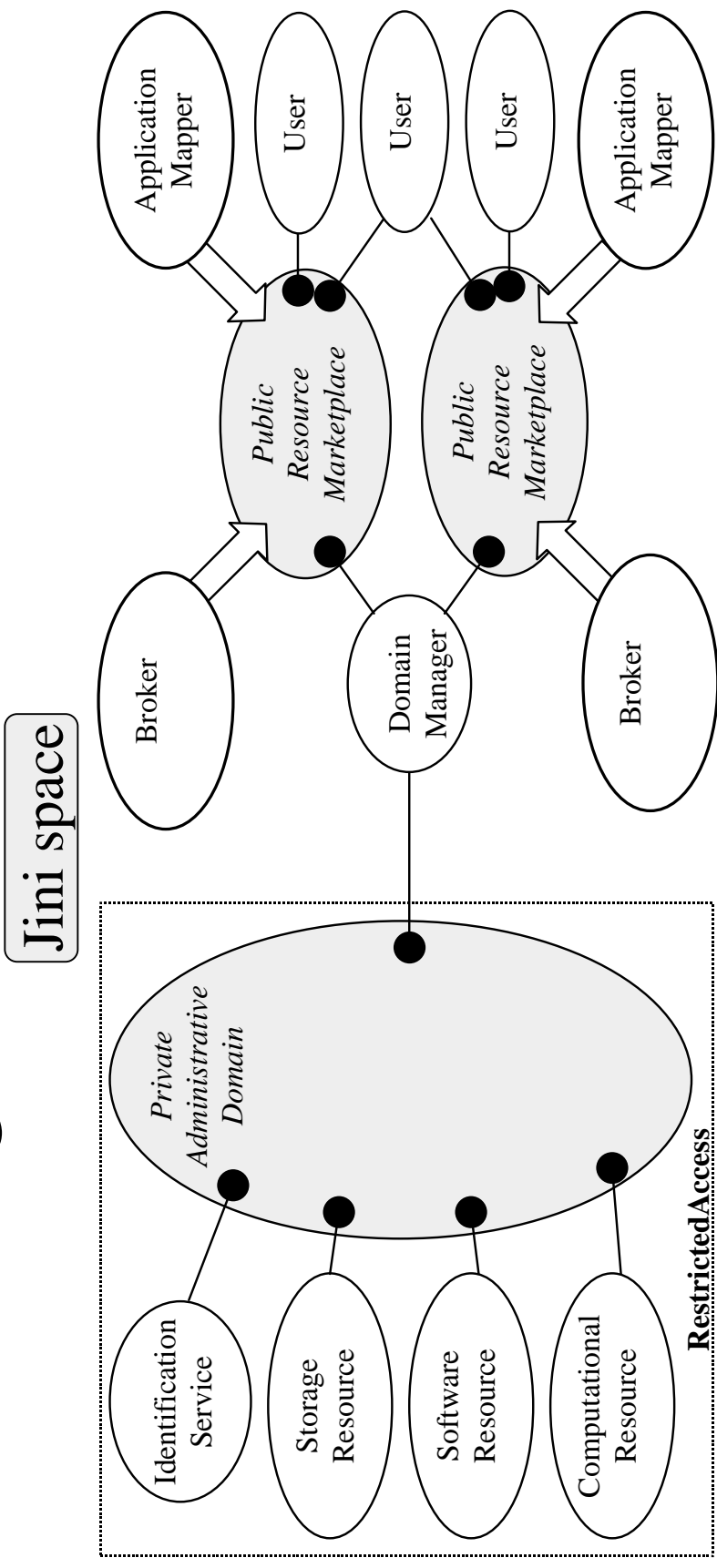
Use Java+ Jini to fill the gap

- Usability
 - Higher-level abstractions for the developer by encapsulating lower-level grid service abstractions, e.g. Java CoG
 - Construct higher-level Grid services from system-level Grid services
 - Interface that is resource transparent and natural, i.e. PSEs, VPEs, Portals
- Capture Information
 - Resource Policy: Who can use what and when?
 - User Intent: Completion deadline or regular job?
 - Application Requirements: Optimal resources for job?
- Exploit Information to optimise resource allocation

Grid Middleware Development

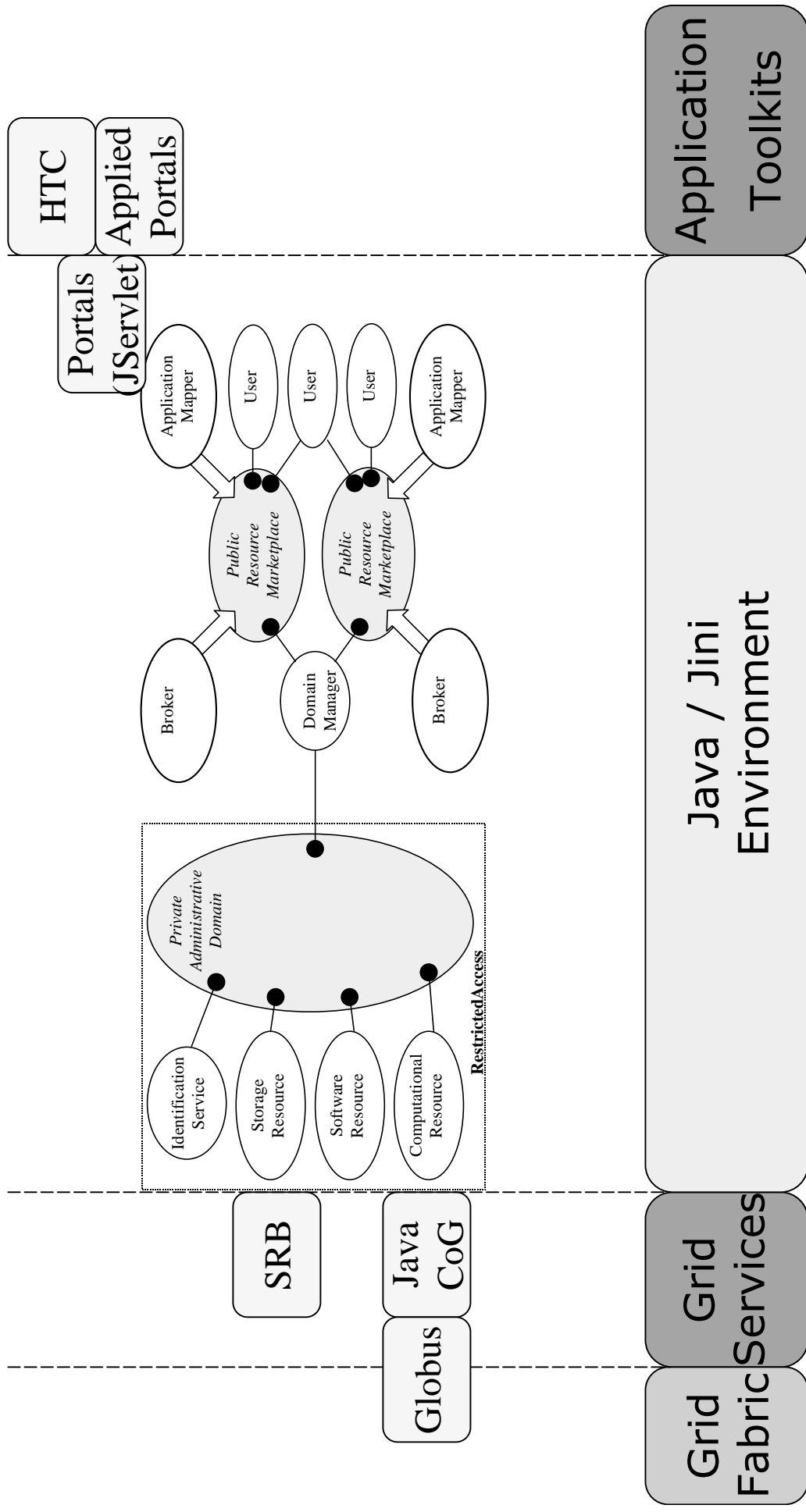
- Federating Resources through Computational Communities
 - Java/ Jini infrastructure to express access and usage policy
- High Performance Software Components
 - Express an application as a composition of components (high level abstract datatypes) – Decoupled definition from implementation.
 - Use component meta -information (CXML) to inform all stages of application construction and execution
 - Use Java to construct the framework and interface to the assembled components
- Optimise Application to Resource mapping
 - Use Performance Models
 - Computational Economics

Federating Grid Resources with Jini



- X509 based security
- Policy + Management Tools
- Mapping GridID to local (pool) accounts

Java and Jini within the Grid

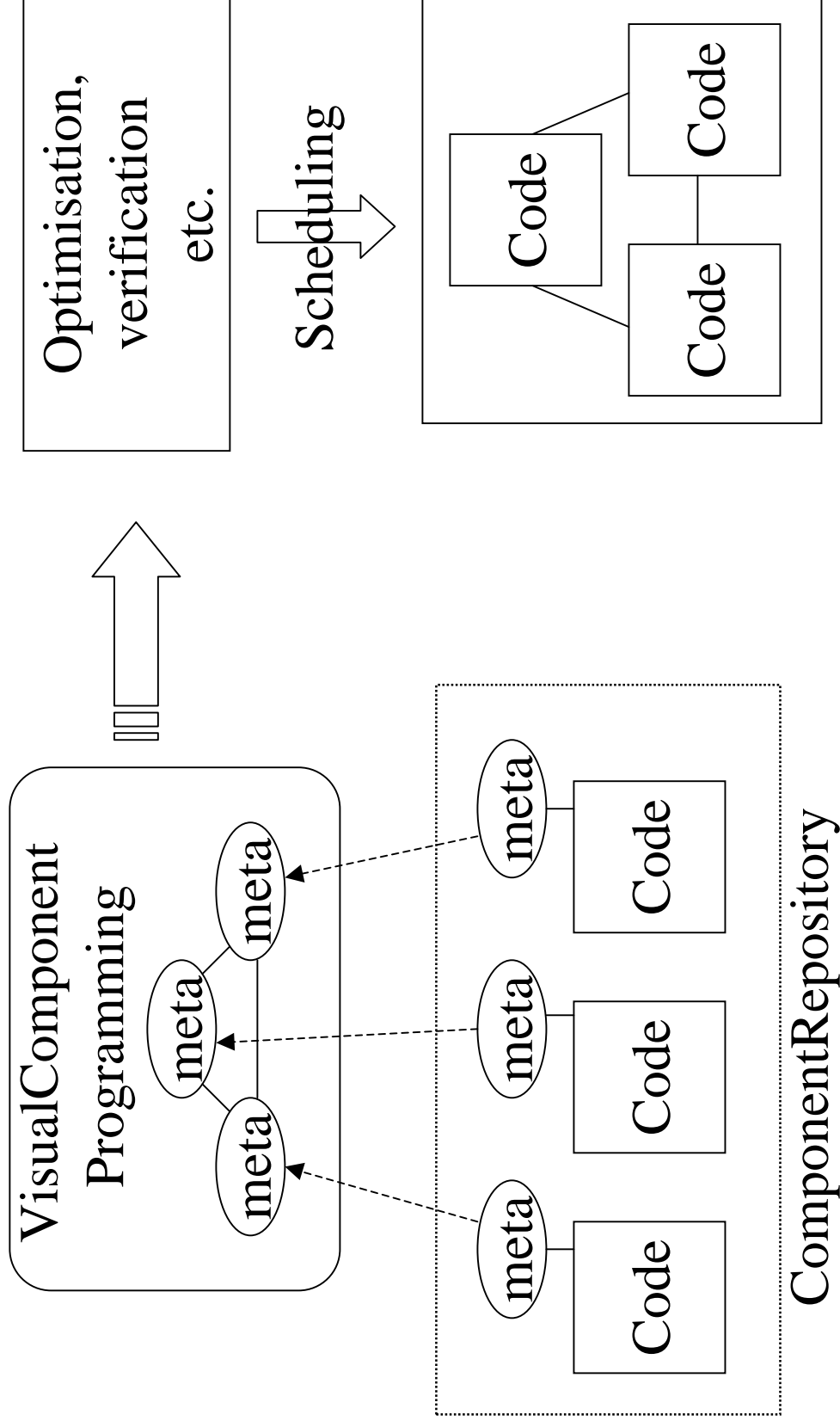


Whatnext...

- P2Pdistributedshareablefilesystem
- Scalabilitytests:
 - Increase number of clients & services
 - Test infrastructure using departmental Condor pool
- Integration of higher level services:
 - Application Composition
 - Application Mapping
 - Broker to support Computational Economics
- Web portal to access Jini infrastructure
- Bank Infrastructure to support scheduling
- More sophisticated brokers

Separating Implementation from

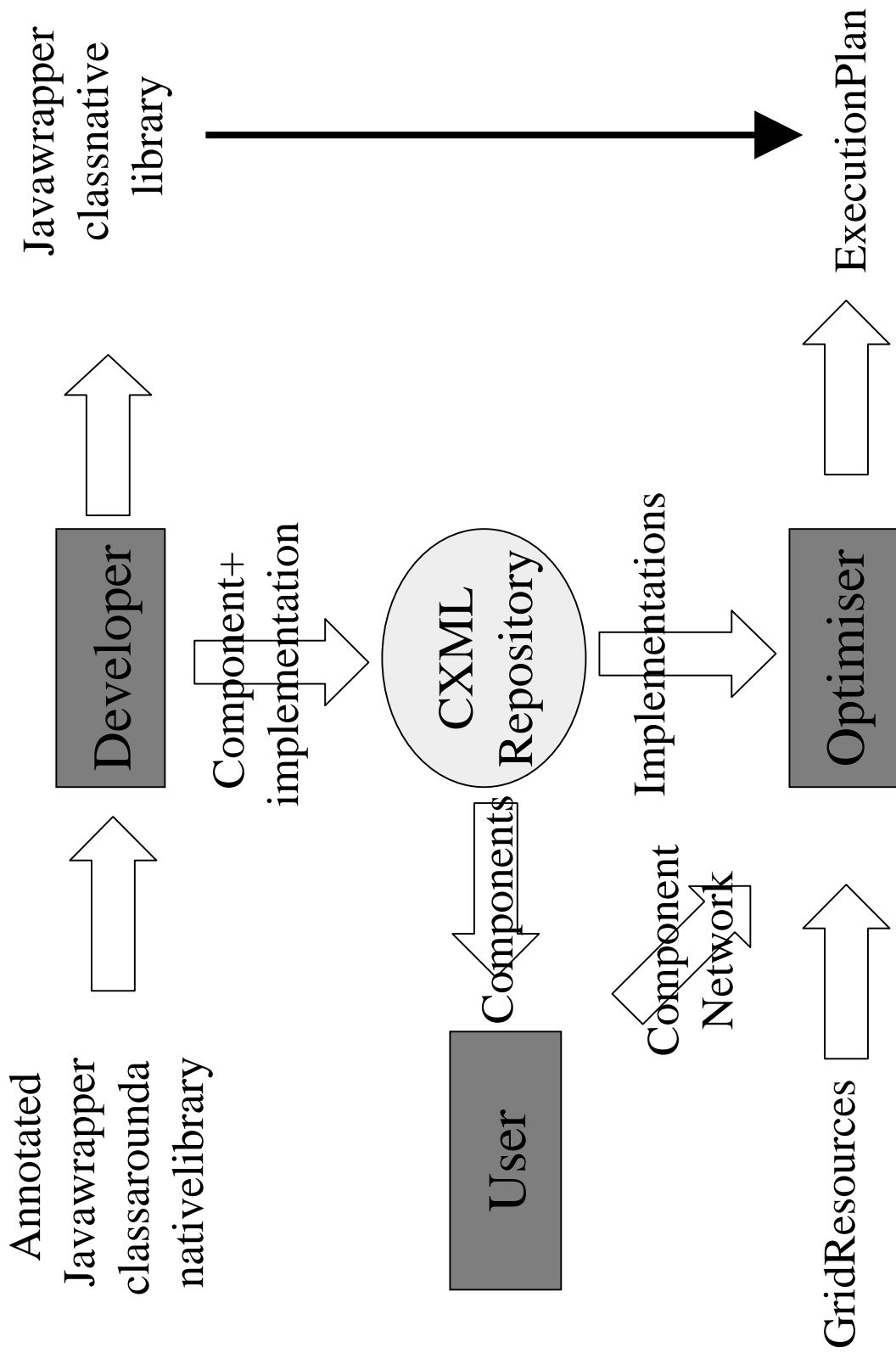
Meta-information



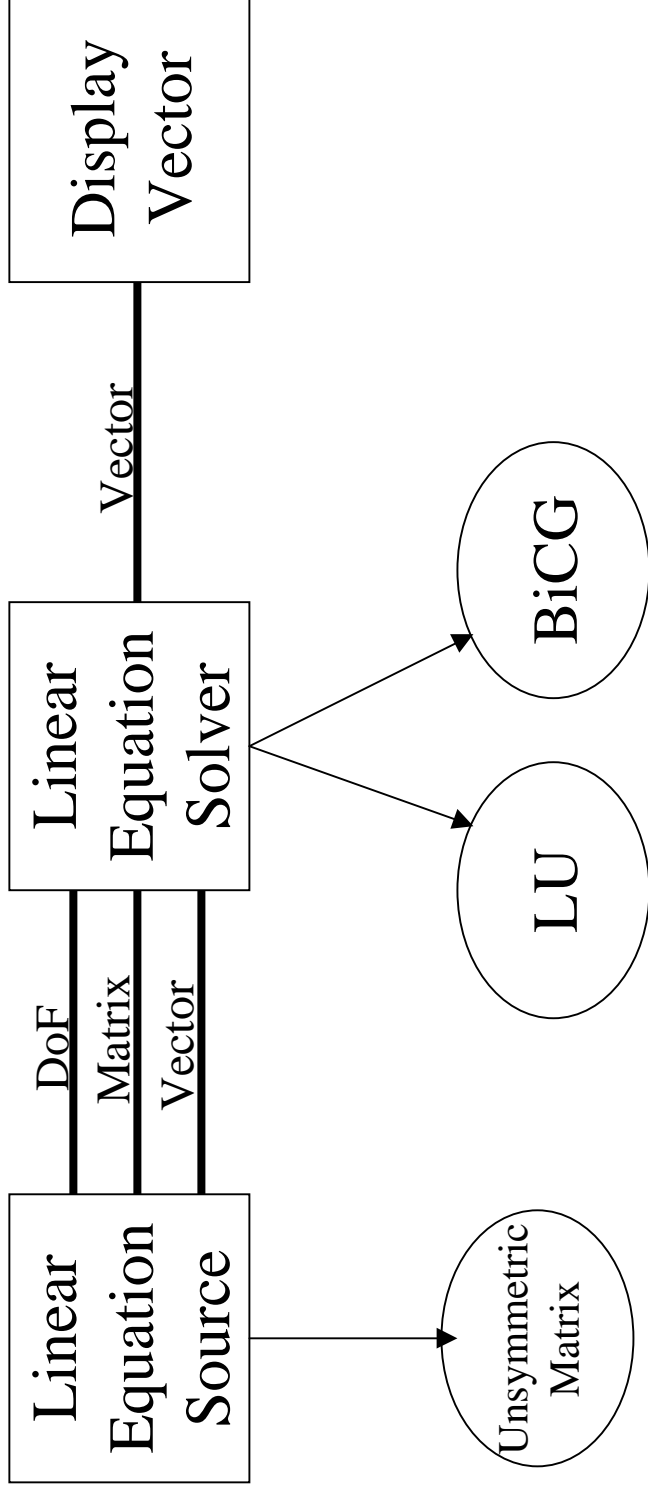
LinearSolverCXML - 1

```
<repository>
  <component package="icpc.denseLinearAlgebra.real"
    name="LinearEquationSourceRowsColumnsUnsymmetric" version="1">
    <propertyDefinition type="external" name="degrees of freedom" value="1000"/>
    <port behaviour="OUT" objectPackage="icpc.matrix.real"
      objectName="MatrixRowsColumnsUnsymmetric" portName="matrix"/>
    <port behaviour="OUT" objectPackage="icpc.vector.real"
      objectName="Vector" portName="vector"/>
    <implementation language="java" platform="java" url="file:.">
      <action portName="matrix">
        <binding method="getMatrix"> ... </binding>
        <classPerformanceModel type="initial" url="http:" />
      </action>
    <implementation language="C" platform="Linux" url="file:."> ... </implementation>
  </component>
  <object package="icpc.matrix.real" name="MatrixRowsColumnsUnsymmetric" version="1">
    ...
    <method name="getMatrix" type="action">
      <argument mode="out" typeName="MatrixRowsColumnsUnsymmetric"
        typePackage="icpc.matrix.real" />
    </method>
  </object>
</repository>
```

CXML as an Intermediate Language



Example: LinearSolver



LinearSolverCXML - 2

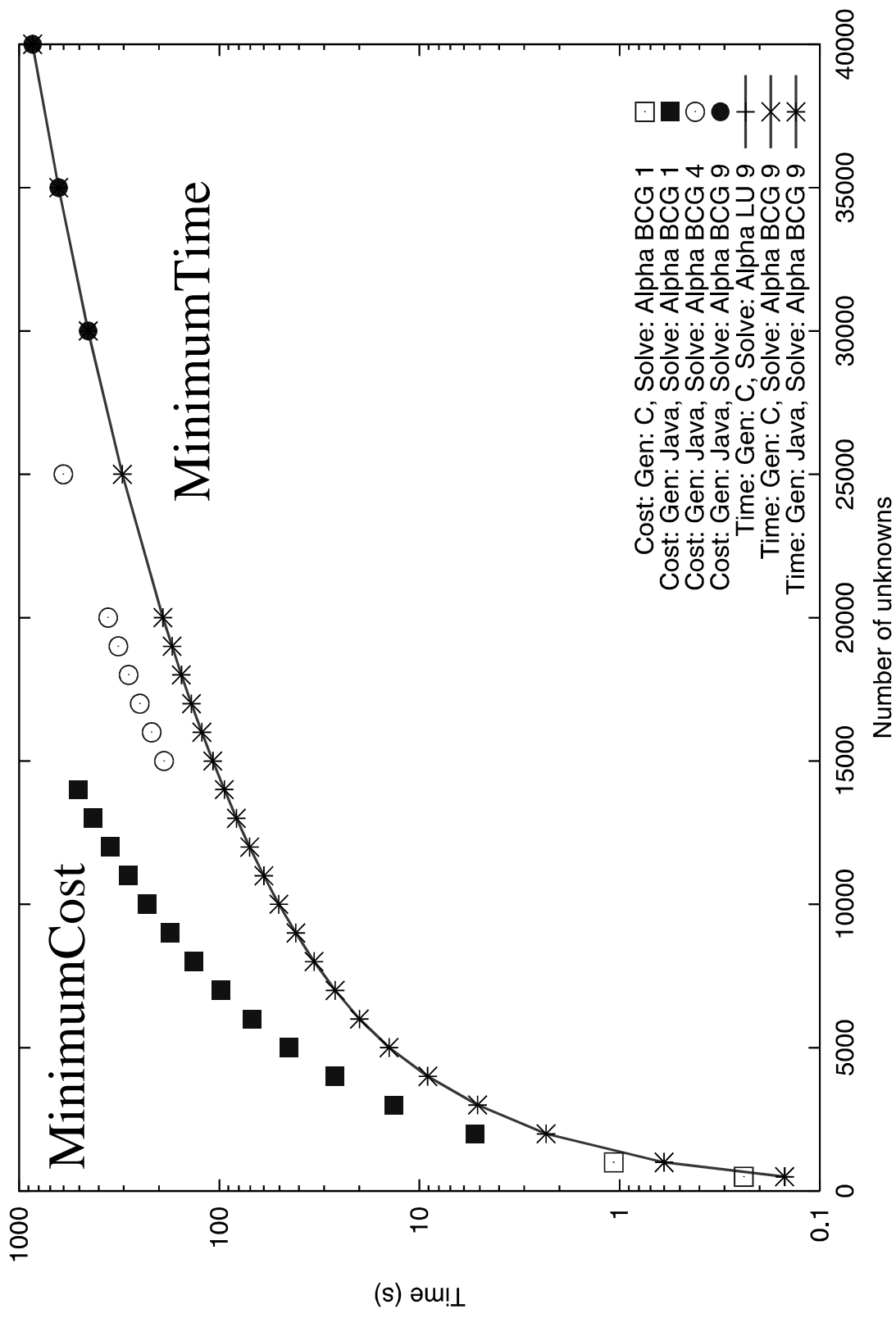
```
<application>
  <network>
    <instance componentName="LinearEquationSourceRowsColumnsUnsymmetric"
      componentPackage="icpc.denseLinearAlgebra.real" id="1">
      <property name="degrees of freedom" value="100"/>
    </instance>
    <instance componentName="LinearSolverRowsColumnsUnsymmetric"
      componentPackage="icpc.denseLinearAlgebra.real" id="2"/>
    <instance componentName="DisplayVector"
      componentPackage="icpc.vector.real" id="3"/>
  </network>
  <dataflow sinkComponent="2" sinkPort="matrix"
    sourceComponent="1" sourcePort="matrix"/>
  <dataflow sinkComponent="2" sinkPort="vector"
    sourceComponent="1" sourcePort="vector"/>
  <dataflow sinkComponent="3" sinkPort="vector"
    sourceComponent="2" sourcePort="solution"/>
</application>
```

Local Computational Community

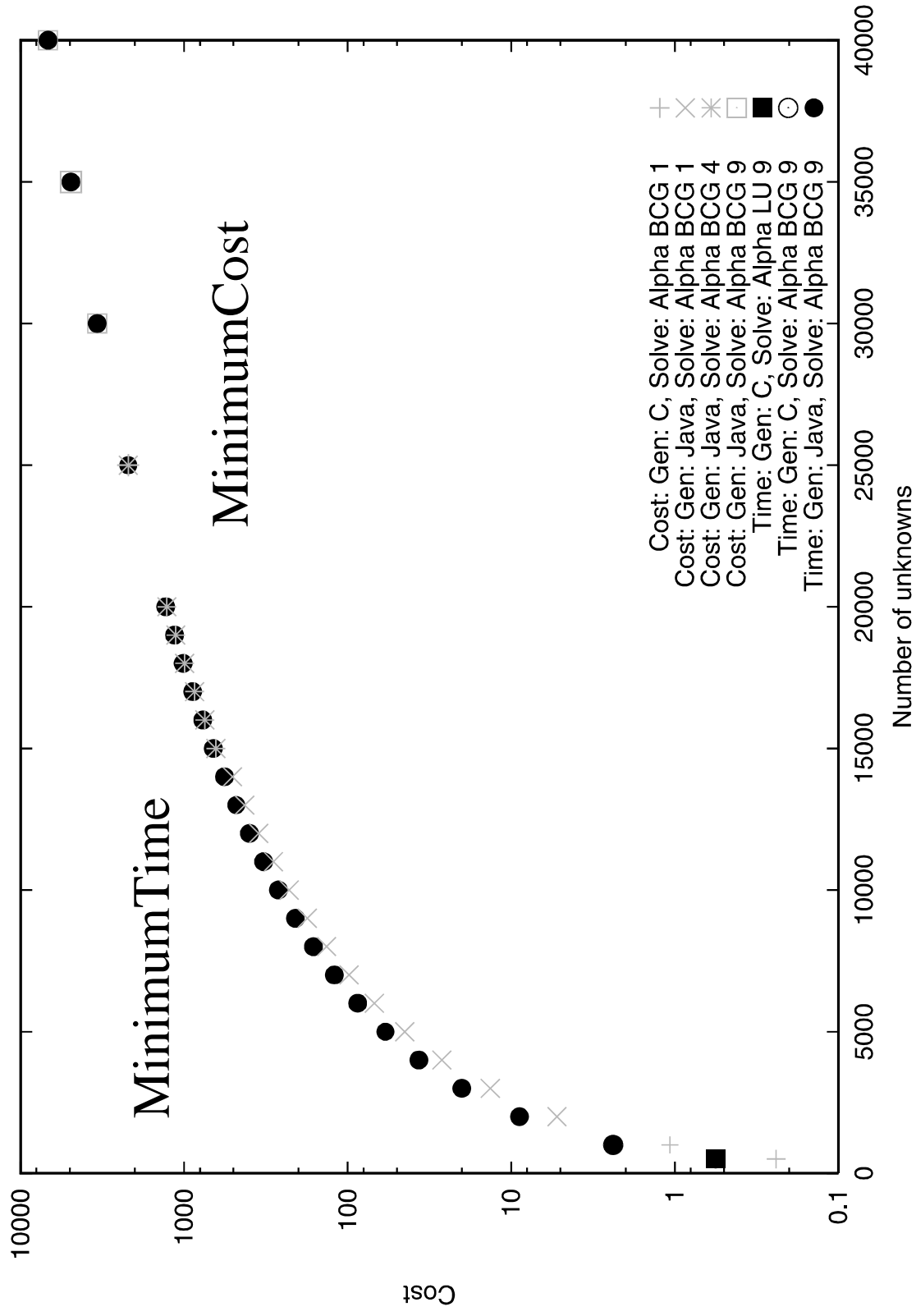
System	Processor	Processors	Language	Solution
PC	AMD	1	Java+C	LU+BCG
(Linux)	900MHz		LAPACK	LU
Atlas	Alpha 667MHz	1,4,9	ScaLAPACK	LU+BCG
AP3000	UltraSparcIII 300MHz	4,9,16	ScaLAPACK	LU+BCG

All processors have equal cost

Influence of Policy on Execution Time



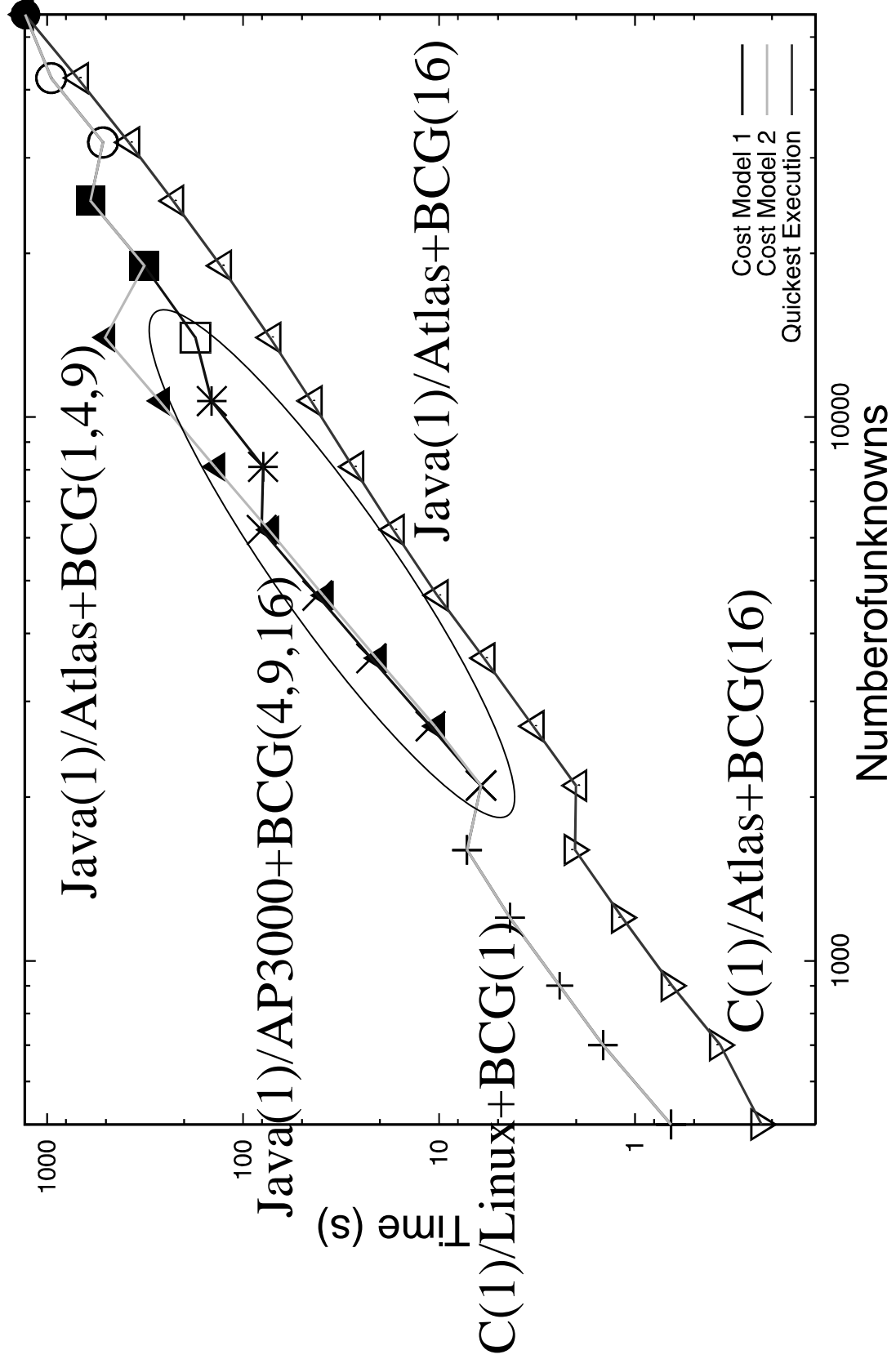
Influence of Policy on Execution Cost



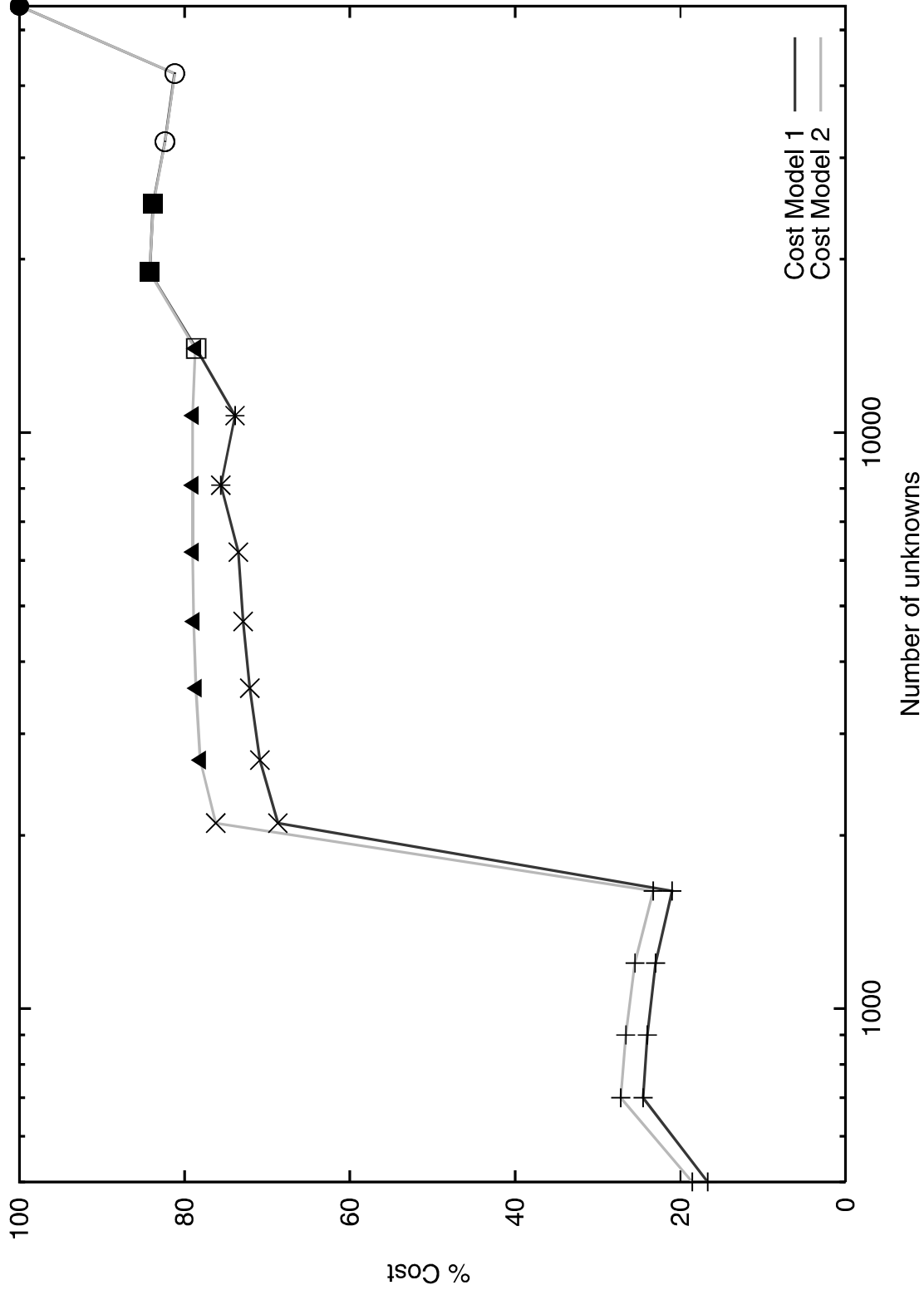
DynamicResourceCost

System	CostModel1	CostModel2
PC (Linux)	80	80
Atlas	475	425
AP3000	100	100

Influence of Cost and Policy



Cost saving relative to minimum time



Summary

- High Performance Software Components
 - Separated definition from implementation
 - Metadata relating to composition
- Computational Communities
 - Define the policy to federate resources
 - Jini provides fault tolerant
- Optimise usage
 - Users specify minimum time or cost
 - Resource providers able to control their job mix

Acknowledgements

- ParallelSoftwareGroup:
 - JohnDarlington
 - Steven Newhouse
 - AnthonyMayer
 - Nathalie Furmento
 - Stephen McGough
- Furtherinformation:
 - <http://www-icpc.doc.ic.ac.uk/components>
- Contact: icpc-sw@doc.ic.ac.uk
- Funding:EPSRCGR/N13371