

# **Jini Technology for Building a Service-Oriented Grid System**

David W. Walker  
School of Computer Science  
Cardiff University

# Outline

- Part 1: Introduction to Jini and Web Service technologies.
- Part 2: JISGA — A Jini-based Web Service Oriented Grid Architecture.
- Part 3: Examples and Demonstration
- Part 4: Conclusions and Future Work

# Part 1:

## Introduction to Jini and Web Service technologies

# Part 1: Outline

- Jini network technology
- Jini and the Grid
- Web service technology
- Web services and the Grid

# What is a Jini System?

- A set of components that provides an infrastructure for federating services in a distributed system.
- A programming model that supports and encourages the production of reliable services.
- Services that can be made part of a federated Jini system and that offer functionality to any other member of the federation.

# Purpose of Jini

- Enable users to share services or resources over a network.
- Provide users easy access to the resources anywhere on the network.
- Simplify the task of building, maintaining, and altering a network of devices, software, and users

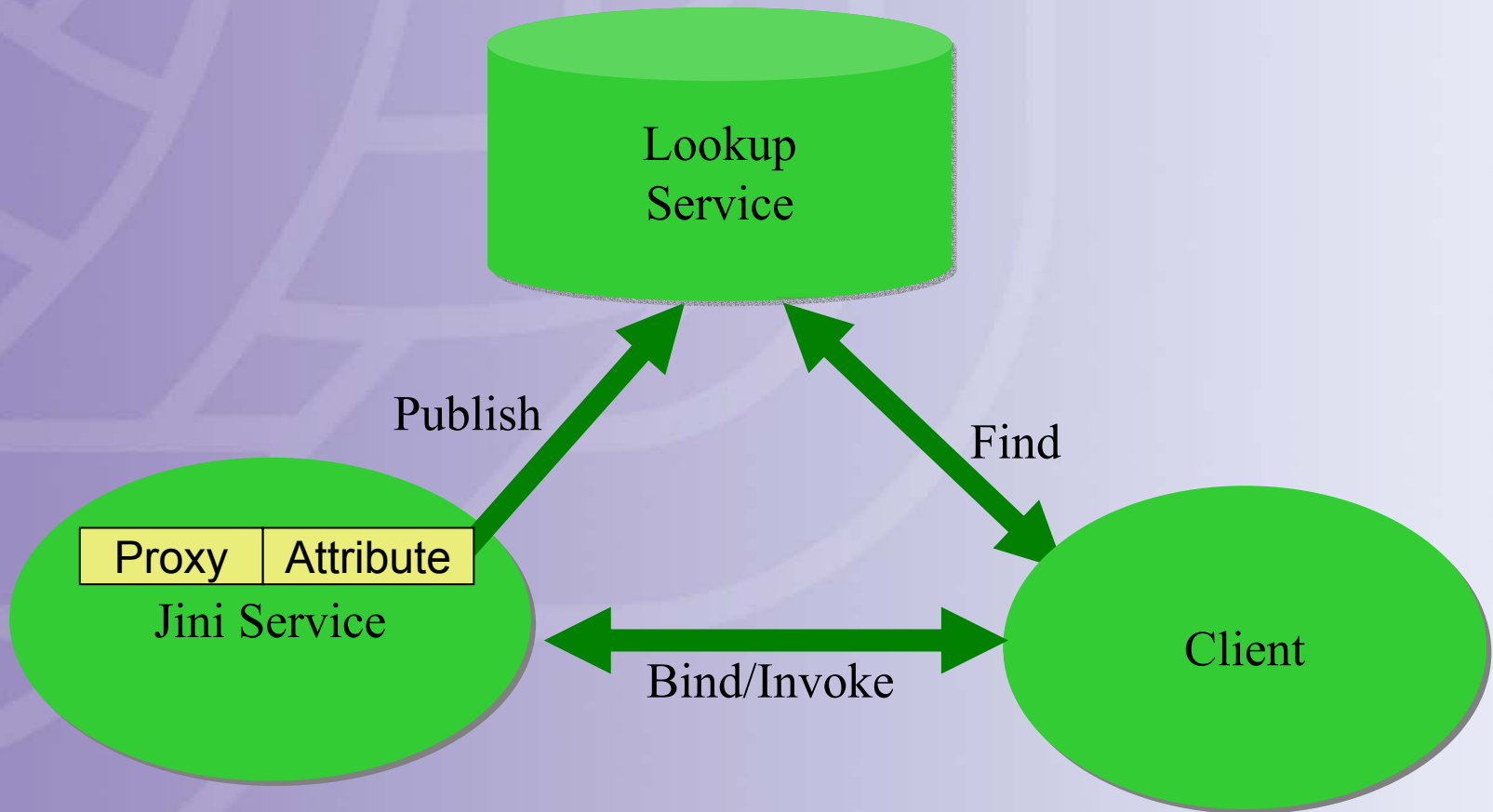
# Why Jini ?

- Code mobility: Both code and data can move from machine to machine.
- Protocol agnostic: Service protocol is a set of interfaces written in Java. This provides a high degree of design flexibility.
- Leasing: Enables network robustness and self-healing

# Central Concepts of Jini

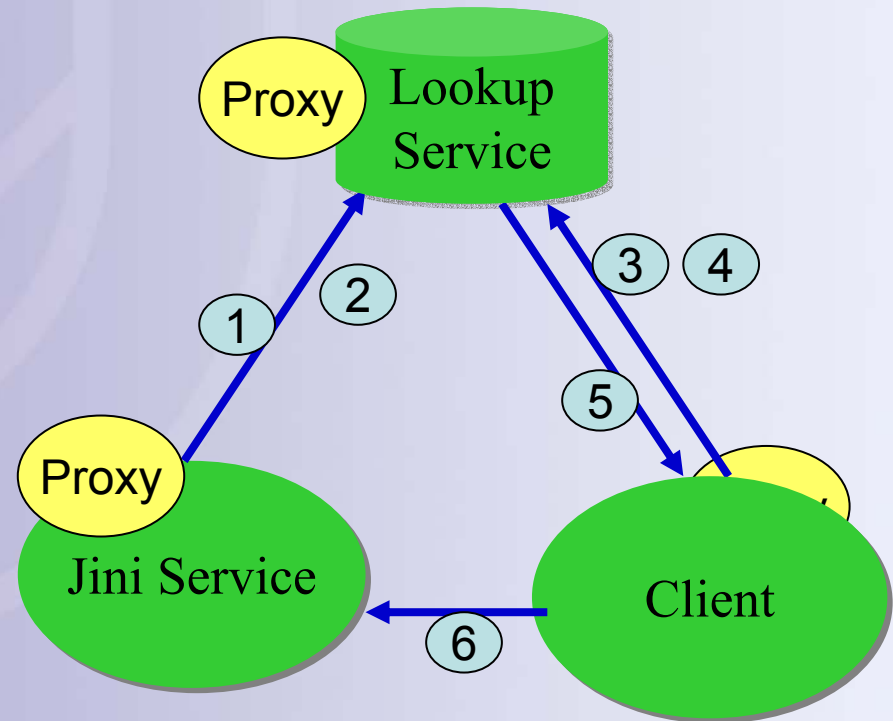
- A Service is a piece of independent functionality that is made available to other users and can be accessed remotely across the network.
- A client is a device or software component that would like to make use of a service.
- A lookup service helps clients find and connect to services.

# Jini Service-Oriented Architecture

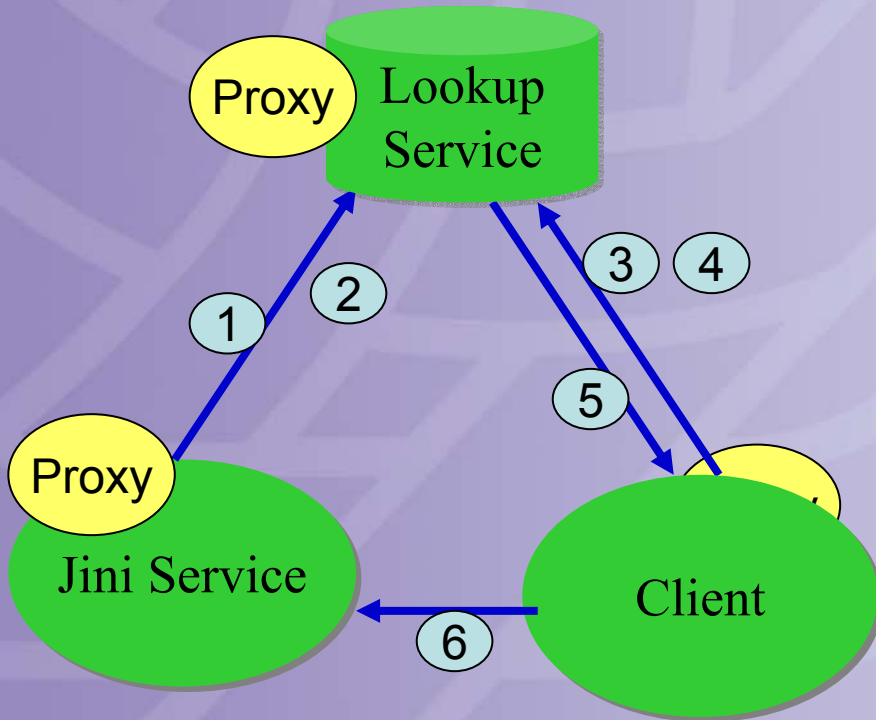


# Jini Process

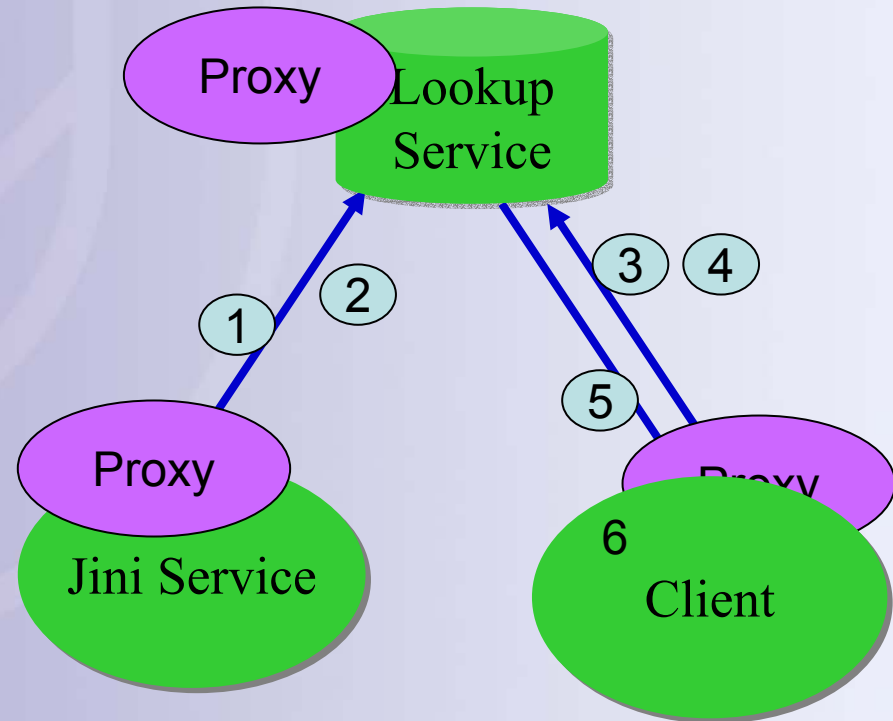
1. Discover: find a Lookup service.
2. Join: send a copy of the service proxy to the Lookup service.
3. Discover: find a Lookup service.
4. Lookup: request a service.
5. Receive a copy of the service proxy.
6. Access service.



# “Thin” and “Fat” Proxy



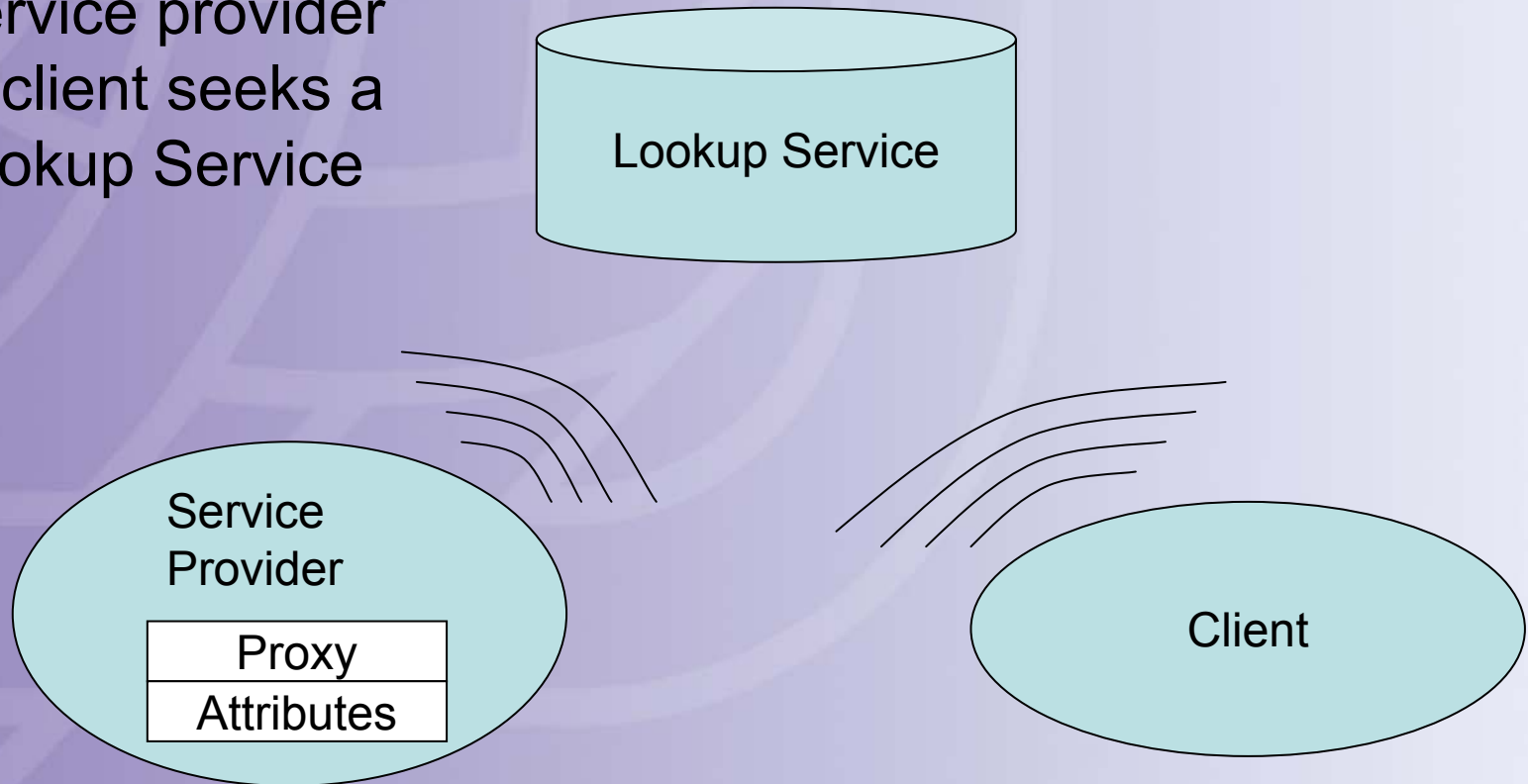
“Thin” Proxy



“Fat” Proxy

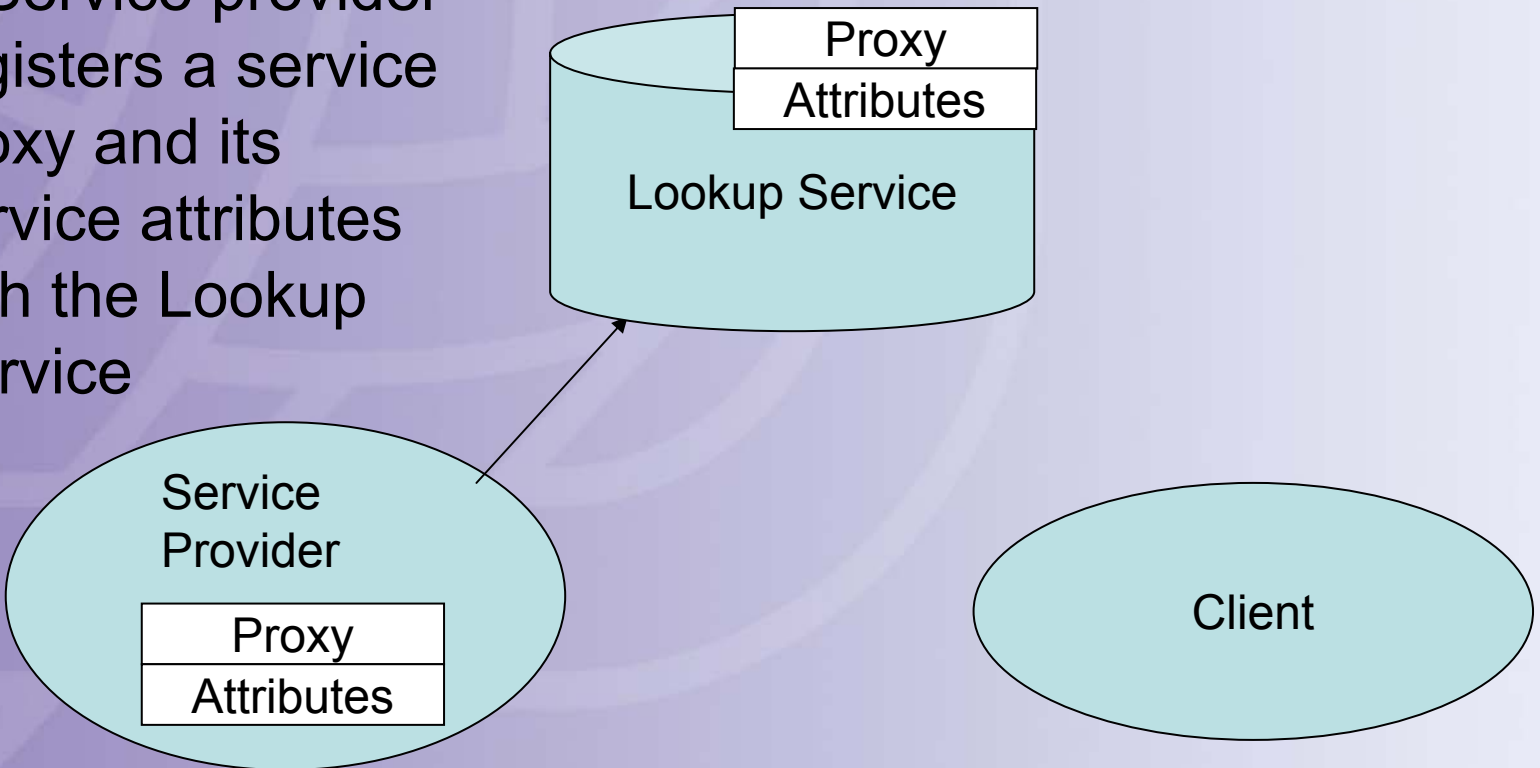
# Discovery

Service provider  
or client seeks a  
Lookup Service



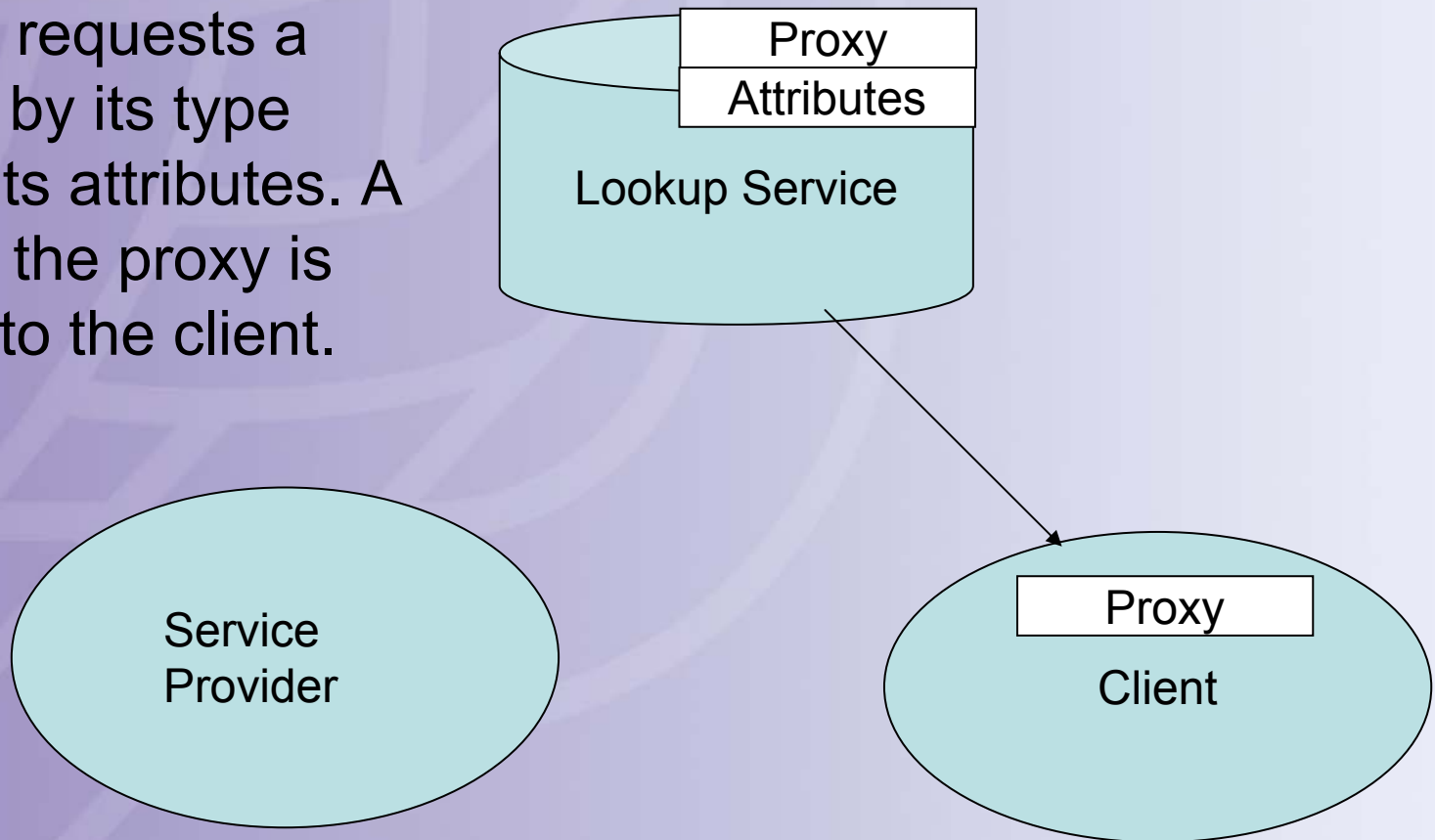
# Join

A Service provider registers a service proxy and its service attributes with the Lookup Service



# Lookup

A client requests a service by its type and/or its attributes. A copy of the proxy is moved to the client.



# Discovery Protocols

- The process of finding the available lookup services.
- Discovery protocols
  - The Unicast Discovery Protocol — For applications and services that know about particular lookup services.
  - The Multicast Request Protocol – For applications and services that do not know about any particular lookup services.
  - The Multicast Announcement Protocol — is used by a lookup service to announce its presence.

# Leasing

- Services, when registering, receive a "lease" on their entry in a lookup service for a small period of time, and the "lease" must be renewed at regular intervals if the service is to continue to be available.
- This makes Jini systems self-healing and provides a consistent means to free unused or unneeded resources throughout Jini.

# Transactions

- By grouping a set of operations together we can ensure that all of the operations complete successfully, or that none of them complete at all.
- Defines an atomic series of operations that can occur either in a single service or across multiple services.
- Is based on a Two-Phase Commit protocol: Prepare and Commit.
- Is used to prevent a distributed application from partially failing.

# Remote Events

- Allow an object (service or application) to register its interest in events and receive a notification of the occurrence of such events.
- Jini provides a simple but powerful remote event programming model.

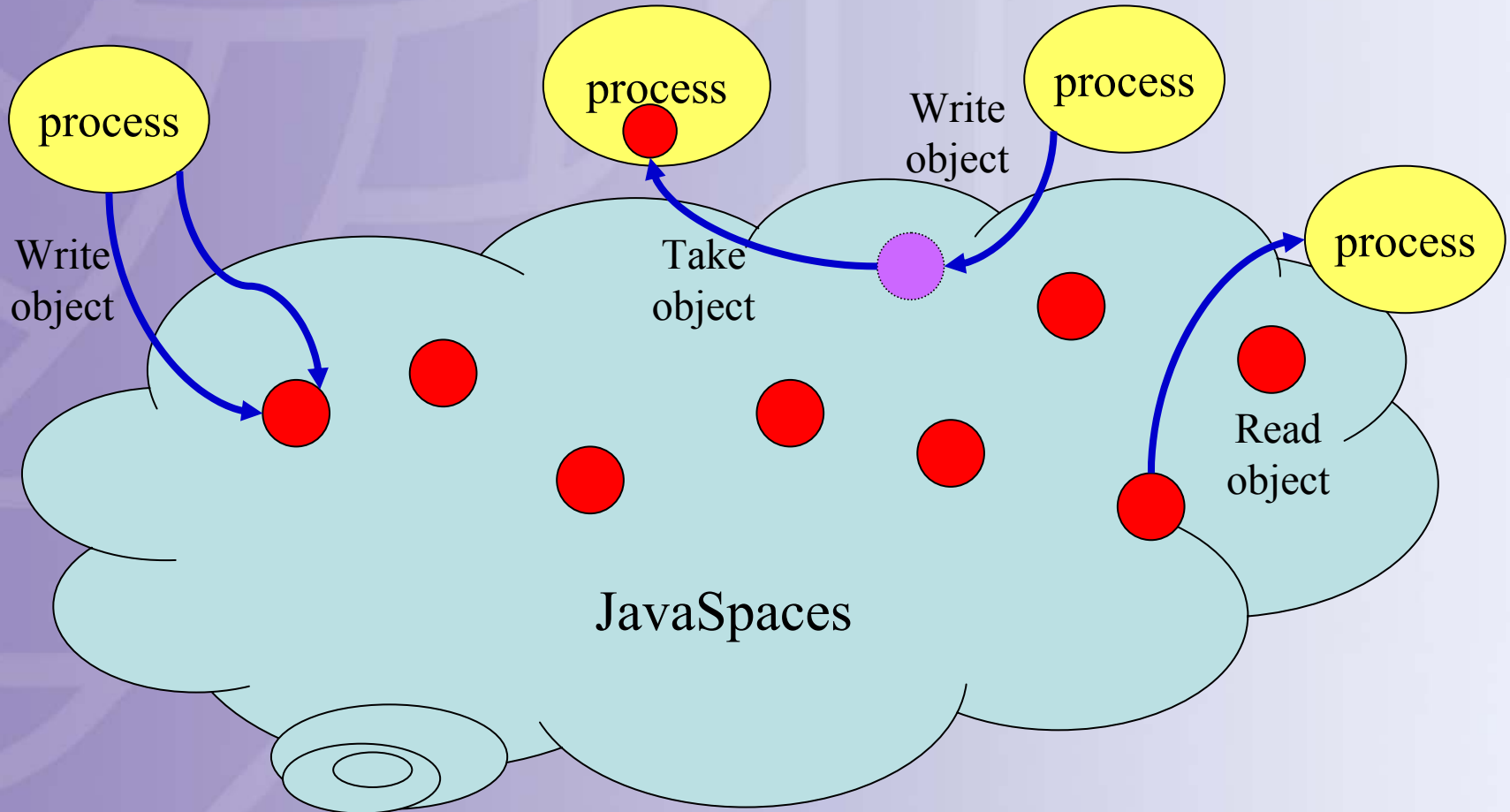
# Using Java Activation Framework

- An activatable service can be inactive until it is needed — it is activated automatically when there is remote call to it.
- An activatable service can be automatically “restarted”, either after a crash or when a machine boots up.
- References to activatable service objects are persistent.

# JavaSpaces

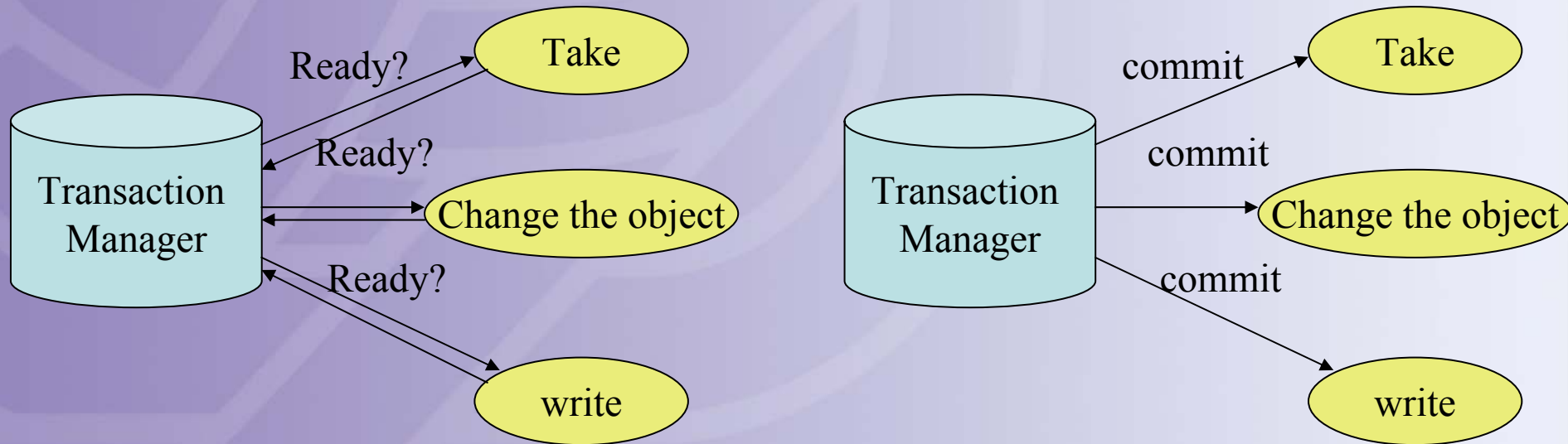
- JavaSpaces provides a shared object storage service for Java objects.
- Operations on JavaSpaces.
  - Processes write objects into the space by copying a local object into the space.
  - Process take objects from the space by removing the object from the JavaSpaces.
  - Process read objects in the space by making a local copy of an object in the space.

# JavaSpaces



# Transactions and JavaSpaces

- Update an object in JavaSpaces



# Jini 2.0

- Supports a secure Jini technology-based programs (Security)
- Provide mechanisms to configure applications
- Unify client-side and server-side programming models (ProxyPreparer and Exporter)
- Provide a new Java RMI implementation – JERI (Jini Extensible Remote Invocation) that supports pluggable invocation layer behavior and pluggable transport providers.

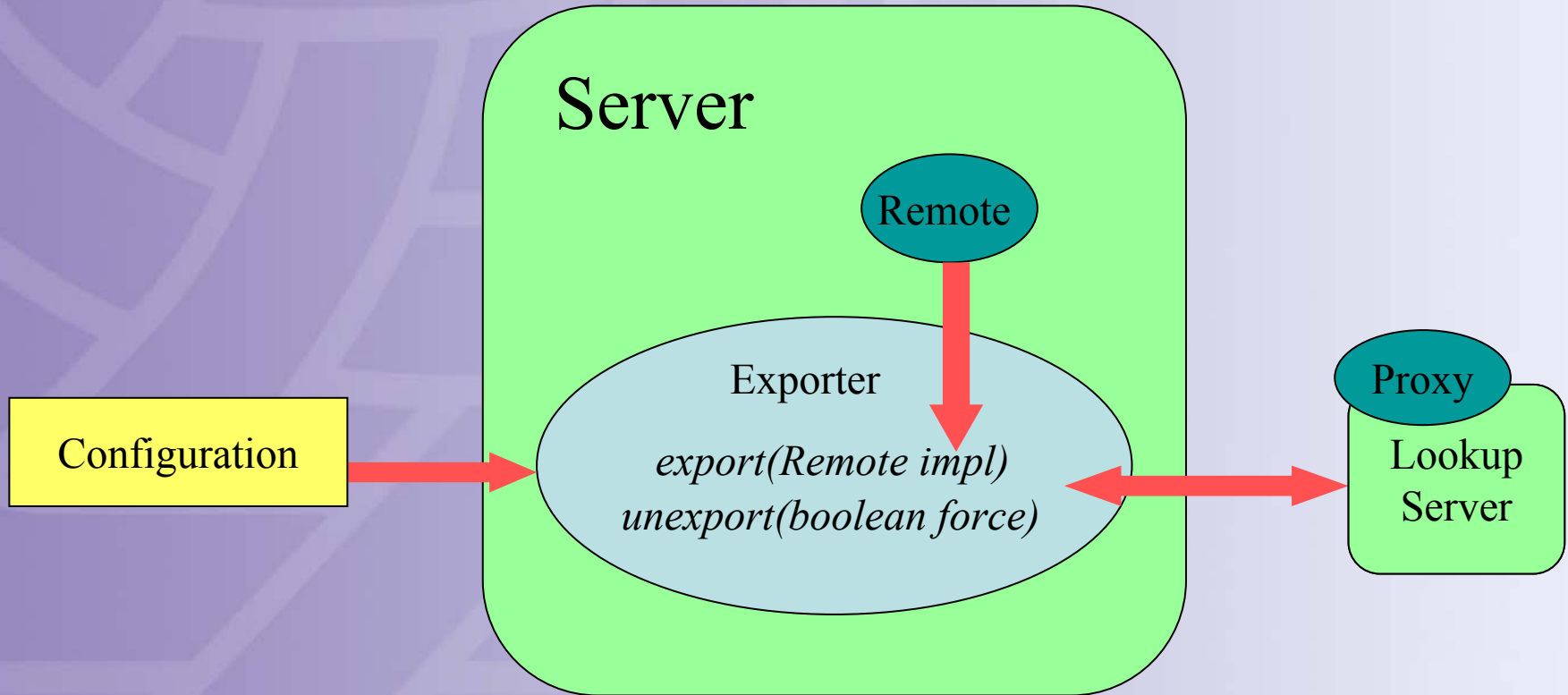
# Jini 2.0 Programming Model

- Configuration—separates deployment-specific information from the application source code, allowing this information to be configured at deployment time.

# Jini 2.0 Programming Model

- Exporter—unifies the server-side implementation model by providing an abstraction for exporting and unexporting a remote object.
  - JrmpExporter---using Java RMI
  - IiopExporter---using Java RMI-IIOP
  - BasicJeriExporter --- using JERI

# Jini 2.0 Programming Model

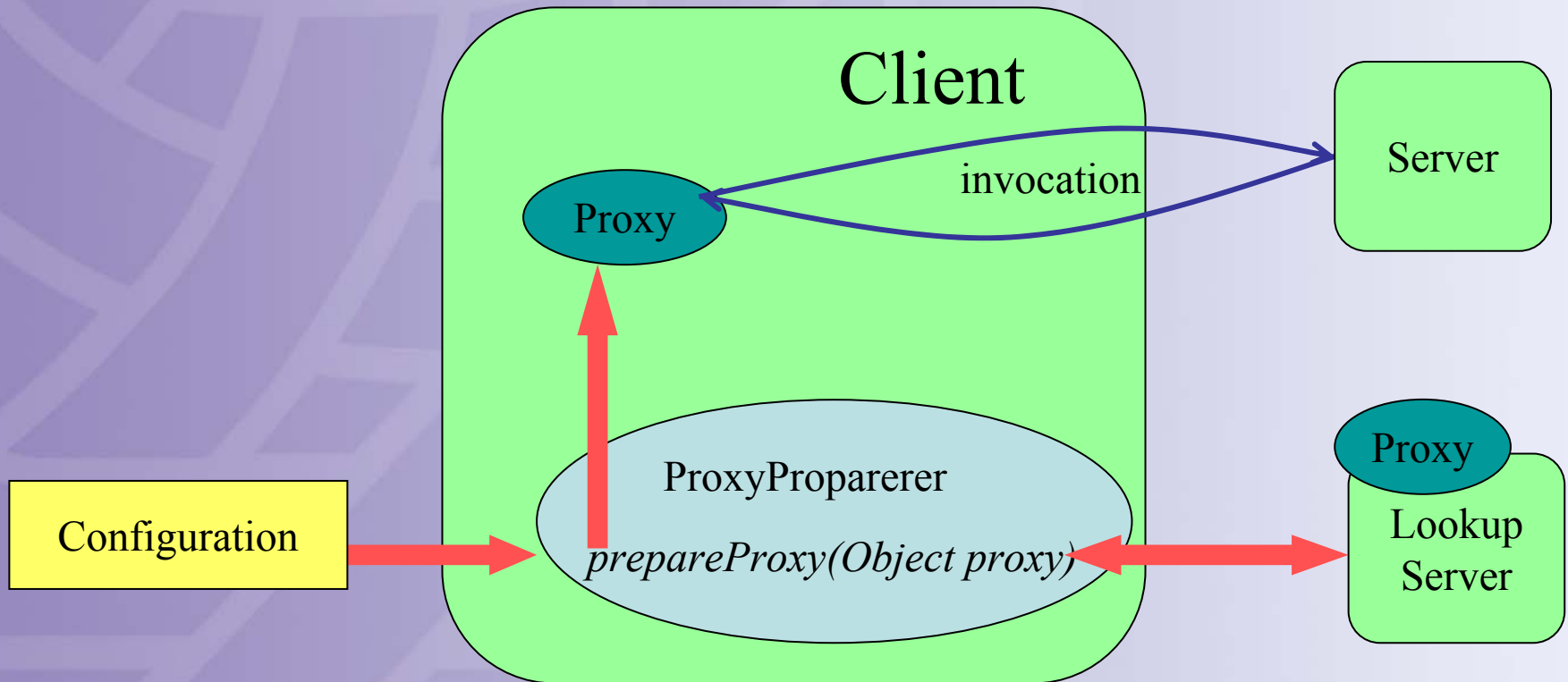


Server-side Programming model

# Jini 2.0 Programming Model

- ProxyPreparer—used server-side implementation model
  - Before using a proxy, preparation is needed to verify the trust in the proxy and/or granting permissions to the proxy.

# Jini 2.0 Programming Model



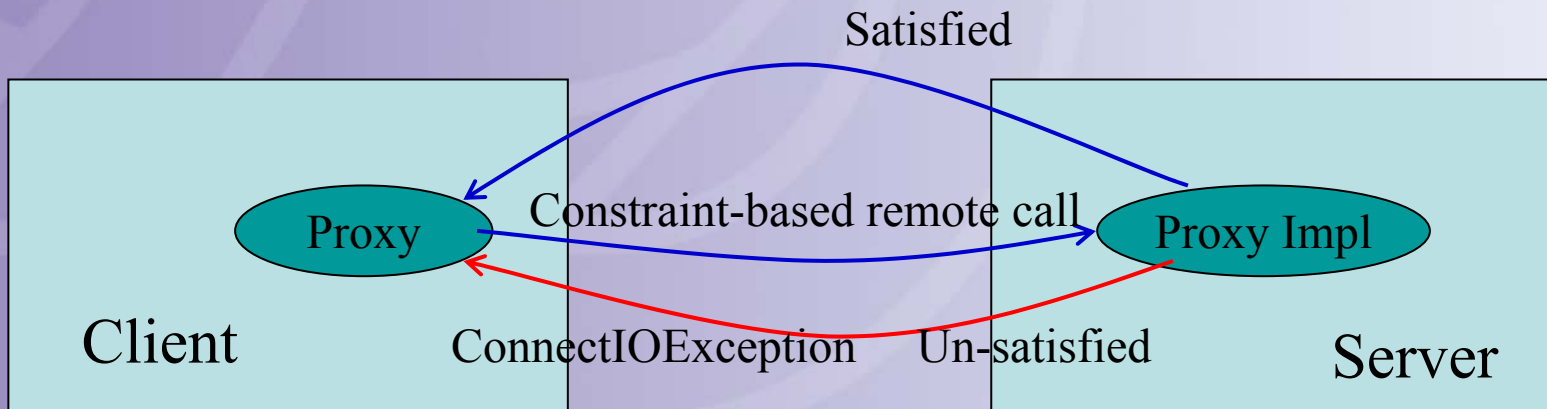
Client-side Programming model

# Jini 2.0 Security Supports

- To secure a Jini remote data communication requires one or more of following
  - Mutual authentication (client and server)
  - Authorization (access control)
  - Integrity (cryptographic checksums)
  - Confidentiality (encryption)
- Mechanisms to ensure code integrity.

# Jini 2.0 Security Support

- Constraint-based remote invocation



# Jini 2.0 Security Supports

- Constraint types
  - Basic trust-related constraints
    - Integrity
    - Server authentication
    - Client authentication
    - Delegation
  - Others
    - Time control
    - Confidentiality

# Jini Extensible Remote Invocation

- JERI Features:
  - Full RMI semantics support
  - Remote object export model
  - Constraint-based remote invocation
  - Trust model
  - Customization of client-side and server-side remote invocation behavior.

# Jini Extensible Remote Invocation

- Use of a variety of communication transports
- No need of build-time generation of stub classes
- Optional use of distributed garbage collection
- Control of all customizable features per exported remote object.

# Jini Extensible Remote Invocation

- JERI Protocol Stack:

Invocation Layer	Deals with remote call semantics: methods, arguments, return values, and exceptions; marshalling and unmarshalling objects.
Object Identification Layer	Identifies distinct exported remote objects and implements behaviour to individual exported remote object (such as DGC)
Transport Layer	Communicates requests and responds over the network.

# Jini and the Grid

- Jini meets the general requirements of federating network resources.
- This generality makes Jini important in a broader view of the Grid.
- The elegance and simplicity of Jini's design, as well as its growing importance as a community standard, make Jini an important technology for building Grids.

# What is a Web Service

- IBM

*“A Web Service is an interface that describes a collection of operations that are network accessible through standardized XML messaging. Web services fulfil a specific task or a set of tasks. A web service is described using a standard, formal XML notion, called its service description, that provides all of the details necessary to interact with the service.”*

# What is a Web Service?

- Microsoft

*“A Web service is a unit of application logic providing data and services to other applications. Applications access Web services via ubiquitous Web protocols and data formats such as HTTP, XML, and SOAP, with no need to worry about how each Web service is implemented.”*

# What is a Web Service?

- Sun

*“Web services are software components that can be spontaneously discovered, combined and recombined to provide a solution to the user’s problem/request. The Java language and XML are the prominent technologies for Web services.”*

# What is a Web Service?

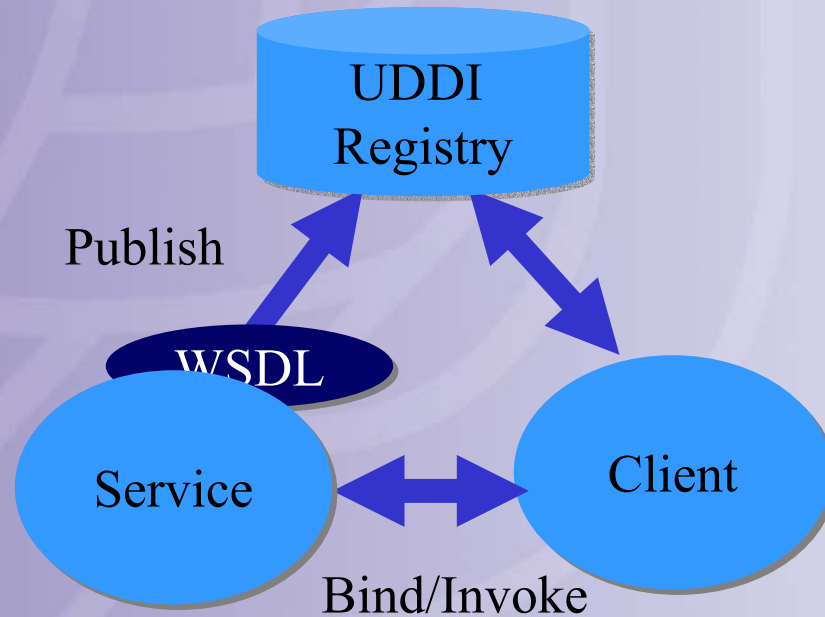
A Web service is a platform and implementation independent software component that

- Is described in an XML-based service description language.
- Is published to a registry of services and can be discovered through a standard mechanism.
- Can be invoked based on its service description.
- Can be composed with other services.

# Web Service technology

- Is not an implementation technology, but about access and integration: application and systems integration.
- Is used to form a system of loosely coupled applications/components.
  - Existing legacy systems and Web applications.
  - Interoperability: the need to find and access resources automatically, without human intervention.
  - Trends in application design are moving from rigid structures to flexible architectures.

# Web Service-Oriented Architecture



# Web Service Standards

- SOAP: provides a standardized messaging service for enabling the service invocation calls between service provider and service requester.
- WSDL: provides a standardized way to describe web services.
- UDDI: provides a standard way of registering and discovering web services

# SOAP (Simple Object Access Protocol)

- A lightweight XML-based protocol for exchange of information in a decentralized, distributed environment.
- It consists of three parts:
  - an envelope that defines a framework for describing what is in a message and how to process it.
  - A set of encoding rules for expressing instances of application-defined data types.
  - A convention for representing remote procedure calls and responses.

# Web Service Description Language (WSDL)

- WSDL is an XML-based standardized language used to describe Web Services based on an abstract model of what the service offers.
- It describes
  - What a services does — the operations.
  - How is a service is accessed — data formats and protocols
  - Where a service is located — network address.

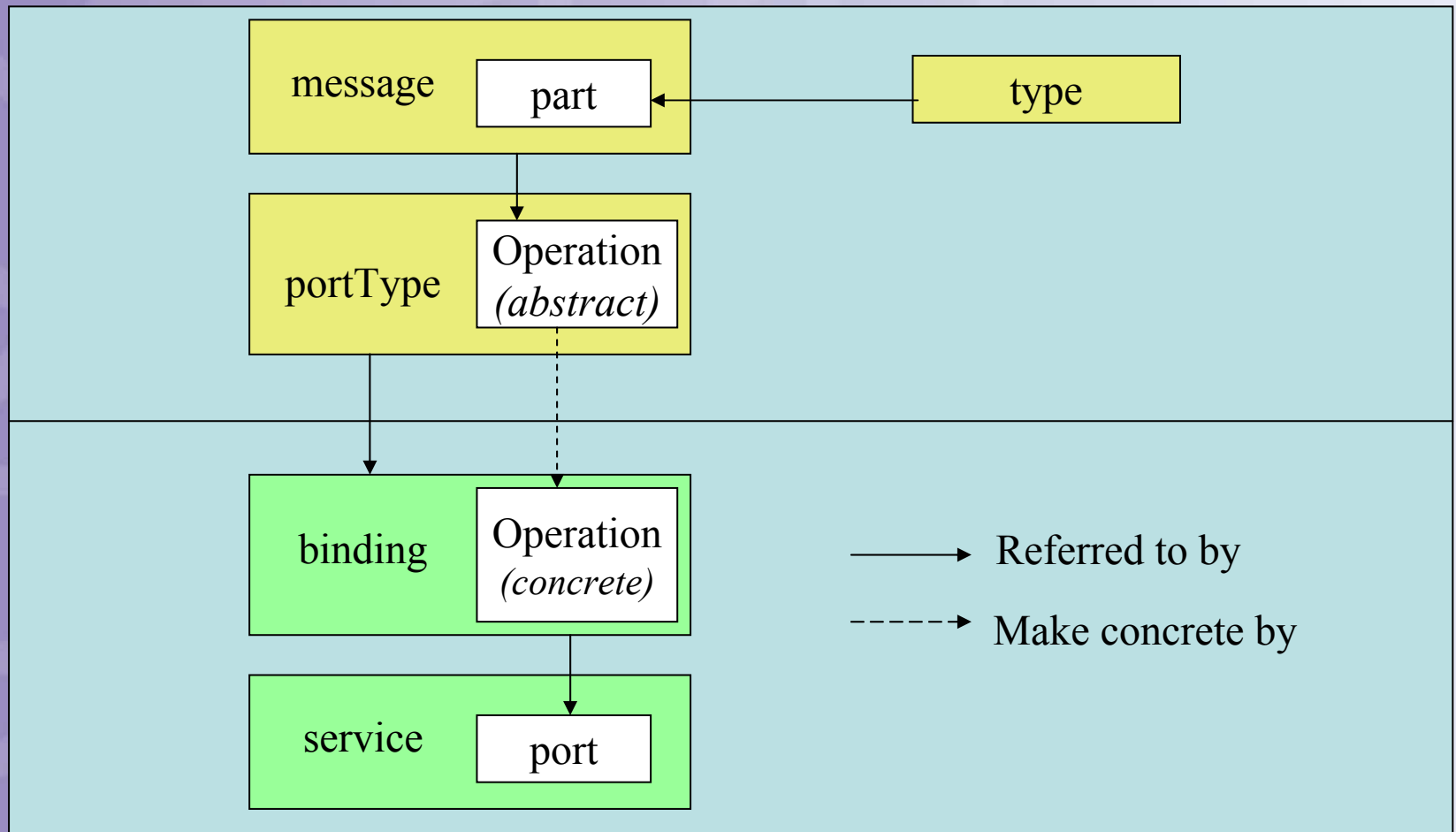
# Major elements in WSDL

- WSDL elements dealing with abstract definitions for Web services.
  - portType — a service's abstract interface definition.
  - Message — defines a set of parameters referred to by the method signatures or operations.
  - Types — defines all the data types referred to by the parts in the message definitions.

# Major elements in WSDL

- WSDL elements deal with abstract definitions for Web services.
- Binding — Specifies bindings of each operation in the PortType.
- Port — Specifies how a binding is deployed at a particular network endpoint.
- Service — Specifies port address of each binding.

# WSDL Elements



# Mapping between Java and WSDL

WSDL Elements	Java File
Type child element	An encode file a data variable
portType operation input (message) output (message)	An interface file a method parameters return
binding	A client-side stub file
service	A client-side factory file

# Universal Description Discovery and Integration (UDDI)

- The purpose of UDDI is to facilitate service discovery both at design time and dynamically at runtime.
- UDDI is a business and service registry.
- UDDI also defines a set of data structures and an programmatic API for registering and finding services
  - Publication APIs
  - Inquiry APIs

# Combining Web services with the Grid

- A set of Web services techniques can be used in Grid systems.
- Different Grid systems and non-Grid systems can be easily integrated.
- All the services are accessible for any user in any organization by presenting service in a standard, universal way.

# Open Grid Service Architecture

- Is an evolution of the current Grid system architecture based on an integration of Grid and Web services concepts and technologies.

# Open Grid Service Architecture

- Defines a uniform exposed service semantics (the Grid services)
- Defines standard mechanisms for creating discovering, and naming transient Grid services
- Provides location transparency and multiple protocol bindings for Grid services
- Supports integration with underlying native platform facilities
- Defines mechanisms required for creating and composing sophisticated distributed applications

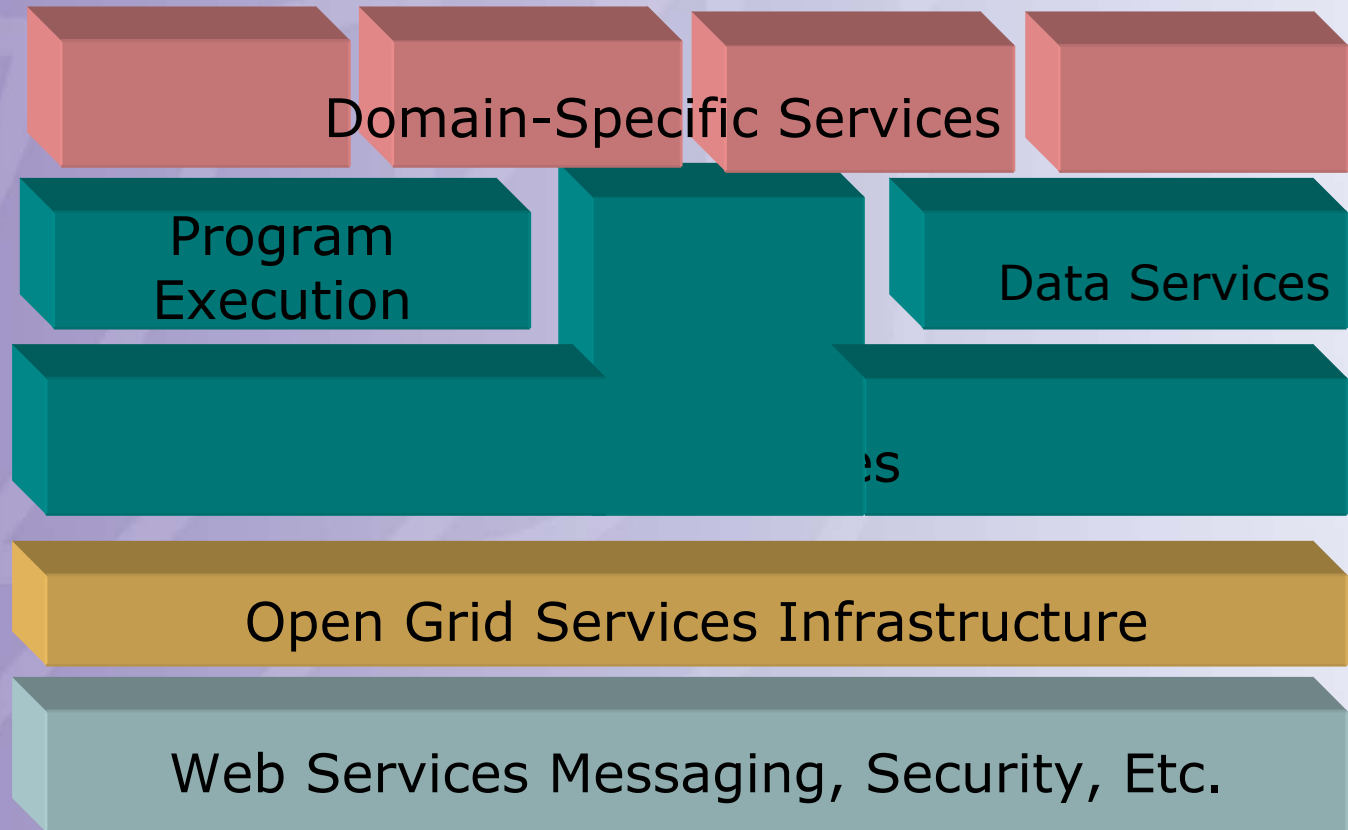
# Web service-based Grid

- Its services are available by any user by presenting machine-independent and context-drive information. (XML, WSDL)
- Its applications are a composition of services and described in the same machine-independent language (XML, WSFL, BPEL4WS, WSCI).

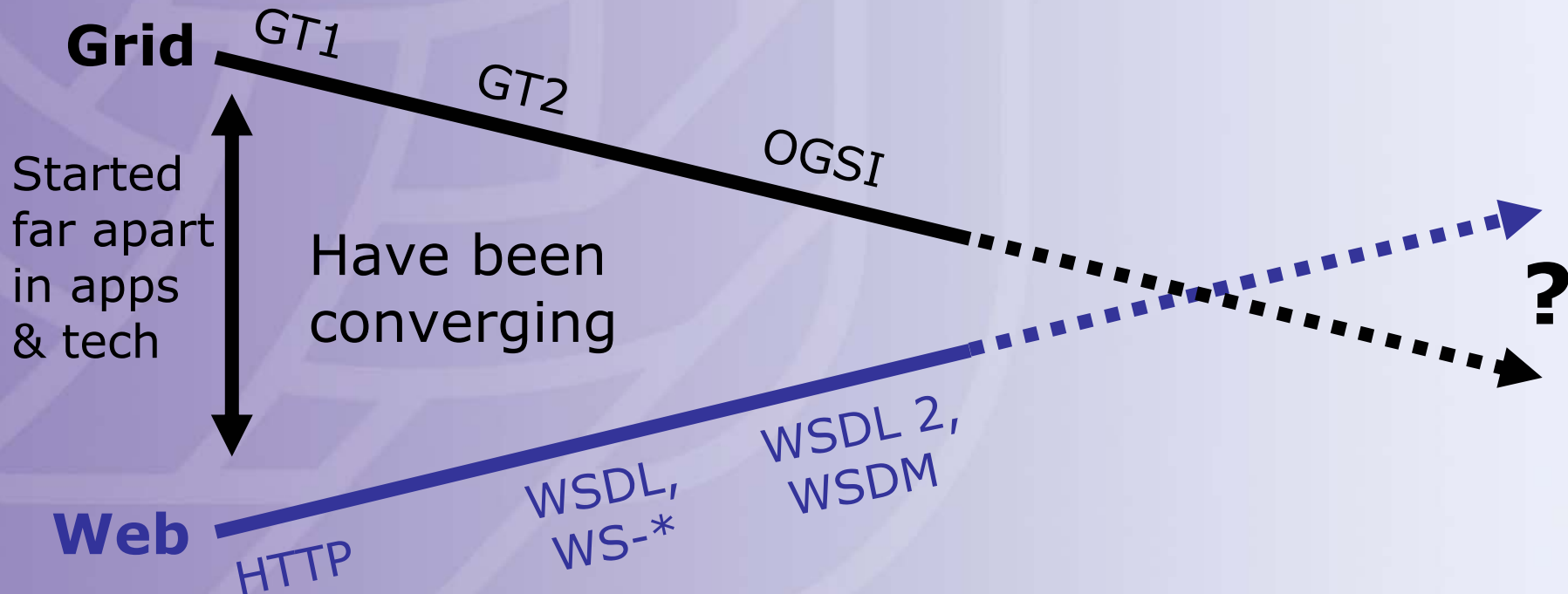
# OGSI vs Web Services



# Open Grid Services Architecture



# Grid and Web Services: Convergence?

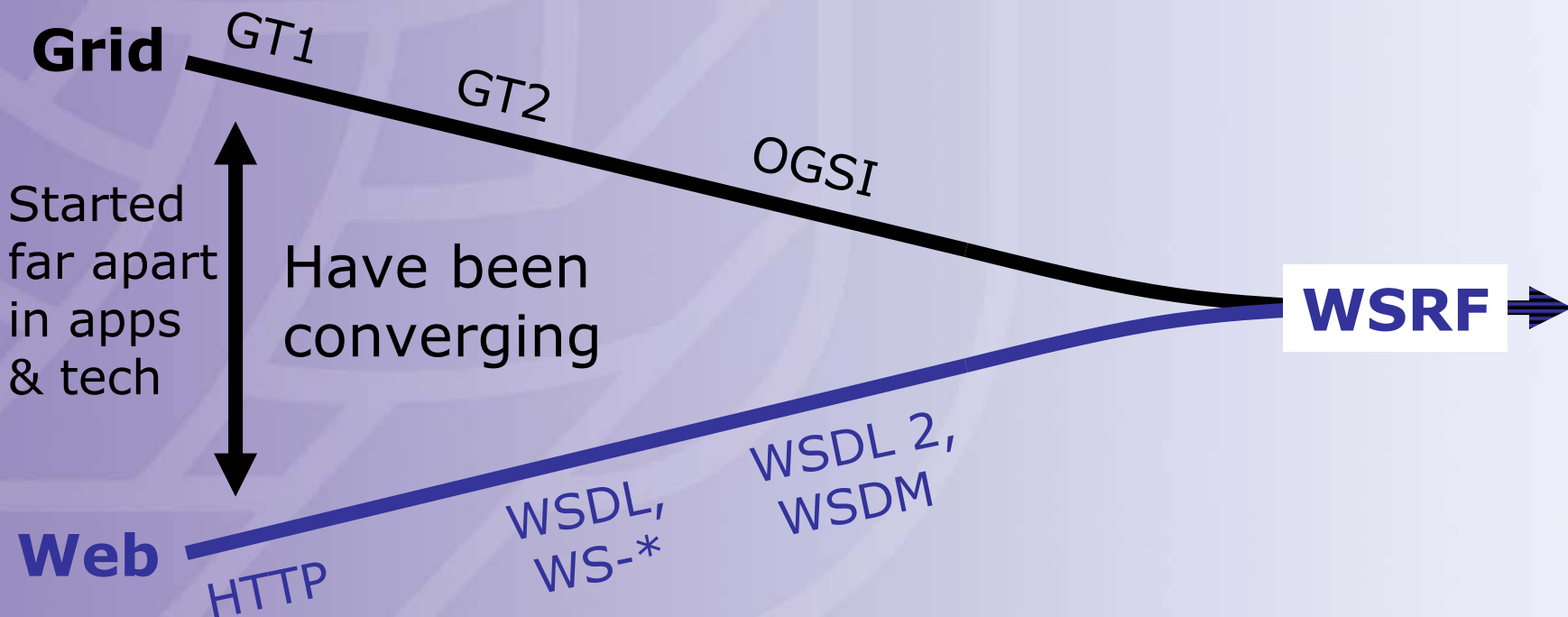


However, despite enthusiasm for OGSI, adoption within Web community turned out to be problematic

# Three Major Web Services Concerns about OGSF

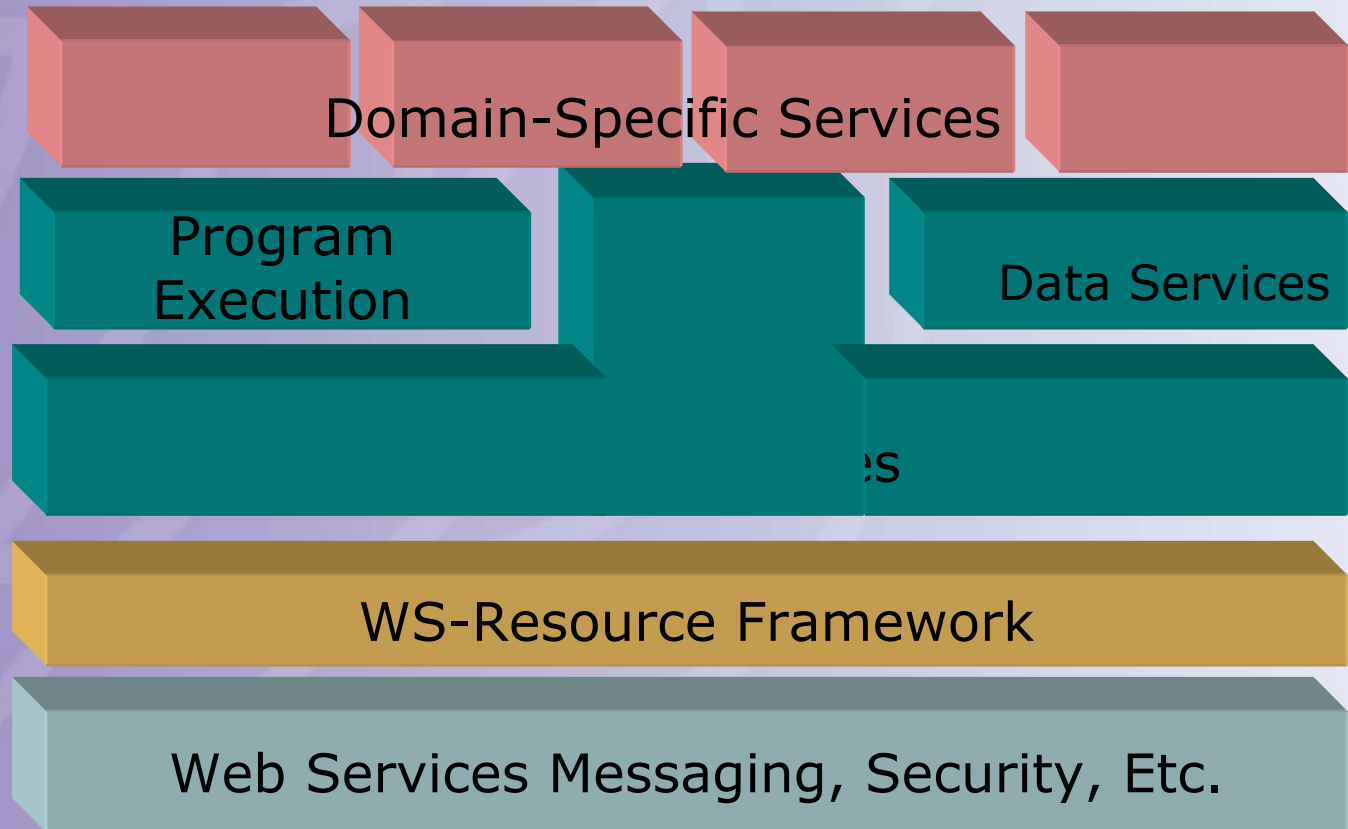
- Too much stuff in one specification
- Does not work well with existing Web services tooling
- Too “object oriented”

# Grid and Web Services: Convergence: Yes!



The definition of WSRF means that Grid and Web communities can move forward on a common base

# Open Grid Services Architecture



# More On WSRF

- Specifications, architecture documents, FAQ, and other information
  - <http://www.globus.org/wsrf>
- Discussion forum
  - <http://www.ggf.org/ogsi-wg>

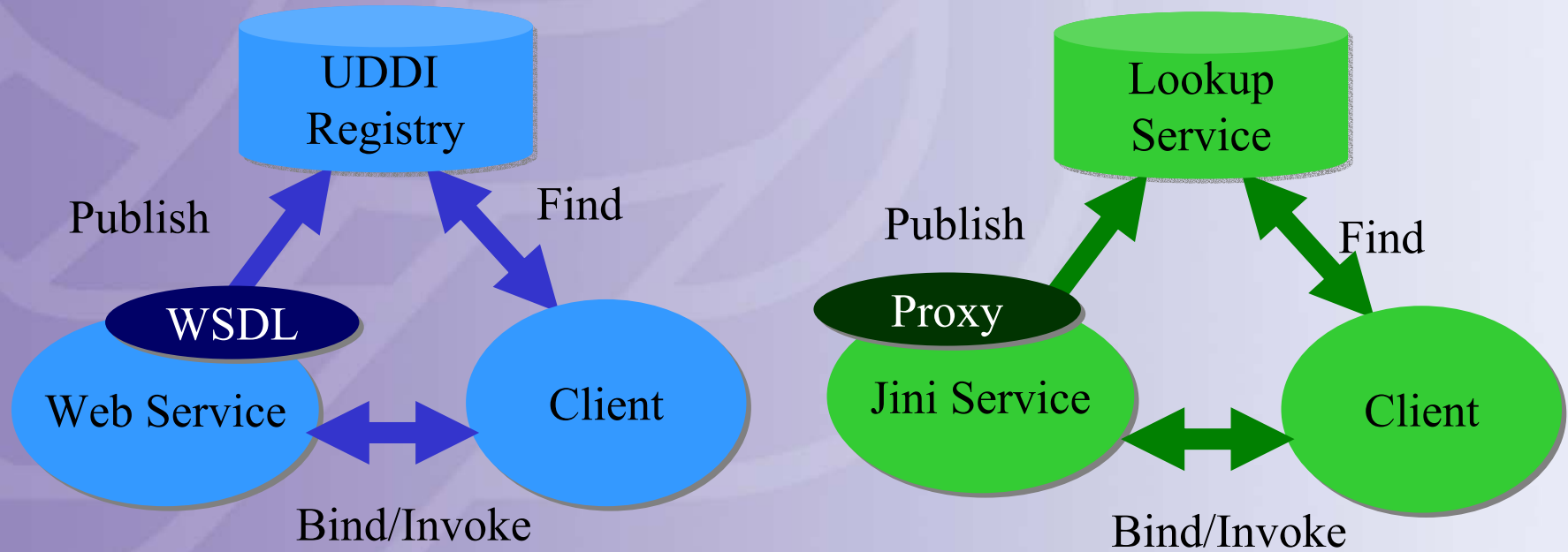
# Part 2

## JISGA — A Jini-Based Service Oriented Grid Architecture

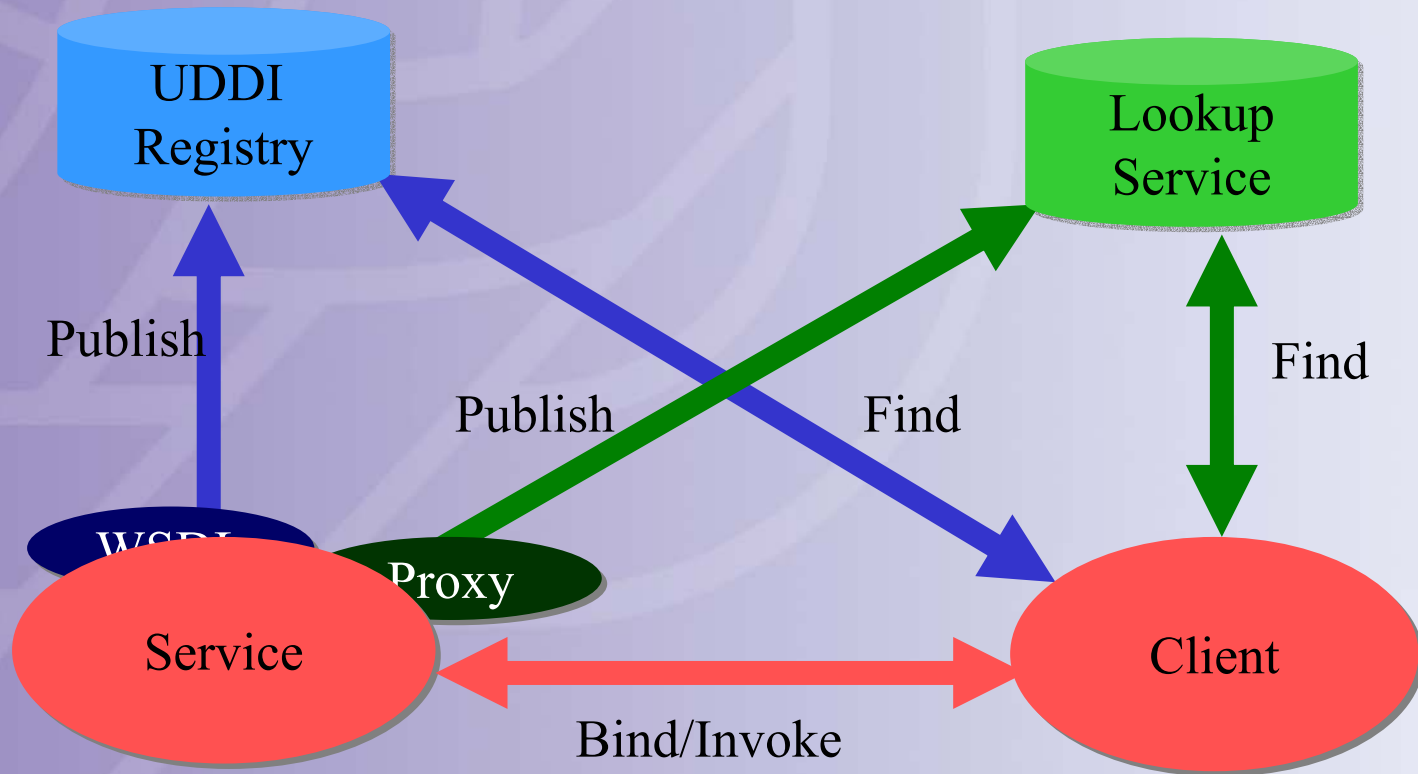
# Part 2: Outline

- Background – Jini technology and Web service technology
- Introduction to JISGA
- Service workflow language
- Implementation aspects — Workflow Engine

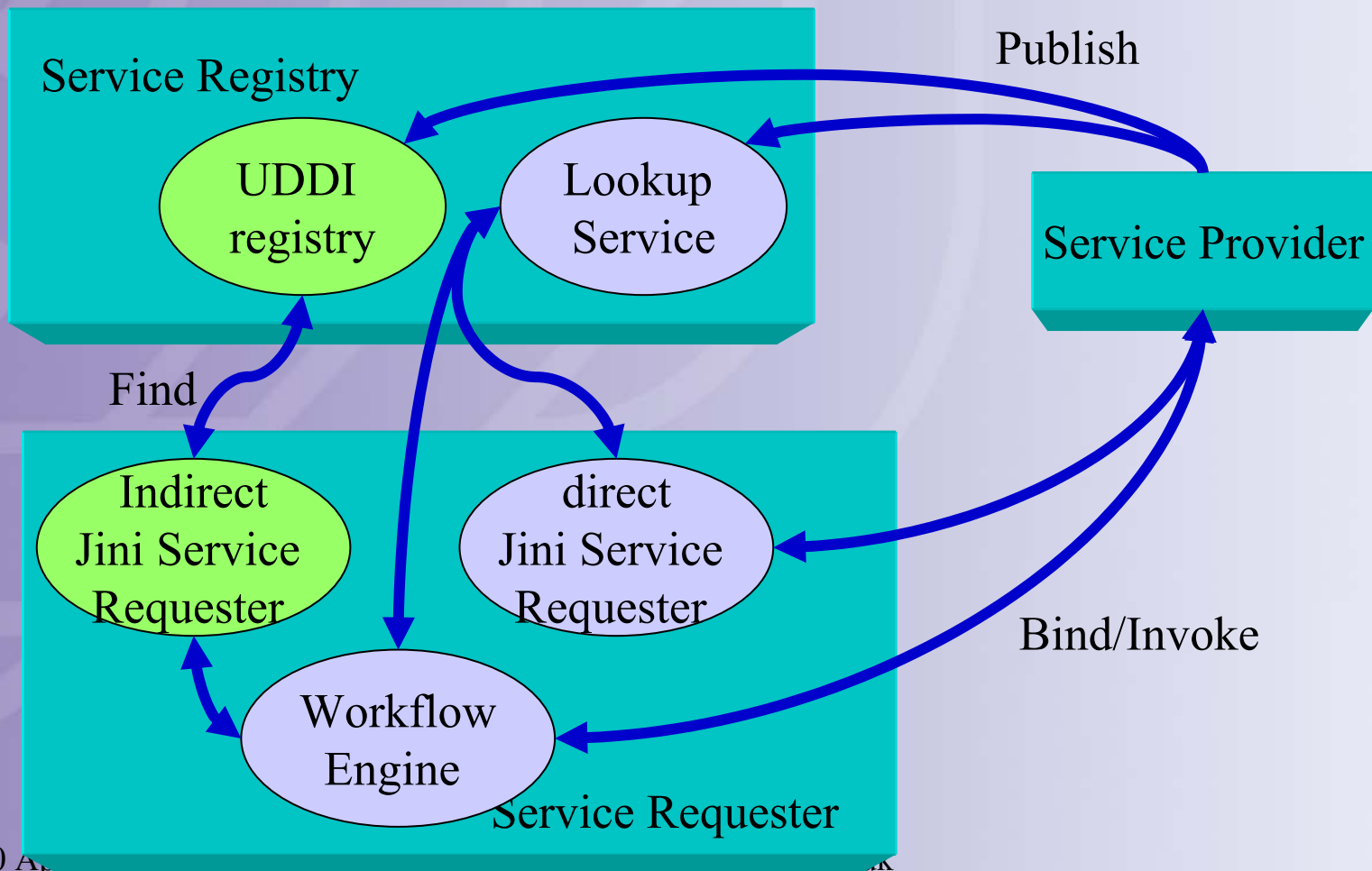
# Web service and Jini service SOAs



# Combining Web Services and Jini SOAs



# A Jini-based Web Service-Oriented Grid Architecture



# Benefits

- Jini services can be accessed from outside of a Jini community.
- Jini services can be invoked in the same way as any other Web service.
- Jini services can be integrated with other Web services
- The Jini Grid will be developed into a OGSA-compliant system.

# Functionalities of JISGA

- Provides the functionalities of a general Jini system.
- Gives users easy access to services by supporting an XML-based, dynamic, context-driven, job description language.
- Automatically processes the job for the user by dynamically creating and executing the job processes.
- It allows both blocking and non-blocking job submission.
- It allows both sequential and parallel job processing.

# A JISGA application

- Is not a traditional software application written in a programming language or an executable code generated by a compiler
- Is a service-based application described by an XML document

# A JISGA service

- Is defined in WSDL
- Is “well-behaved” in the sense of having a LeaseRenewalManager service manage its lease renewal.
- Can be “thin” or “fat”.
- Is activatable.
- Has a persistent service ID
- Has three required attributes associated with it: a service name, a service description, and a service location.

# JISGA Components

- *JavaSpaces* is used as a shared memory mechanism.
- A *ServiceManager* service is used by service providers to publish and remove services.
- A *JSHousekeeper* service is used to deal with certain tasks relating to the use of *JavaSpaces*, such removing objects that have not been recently used, and updating the state of certain objects.

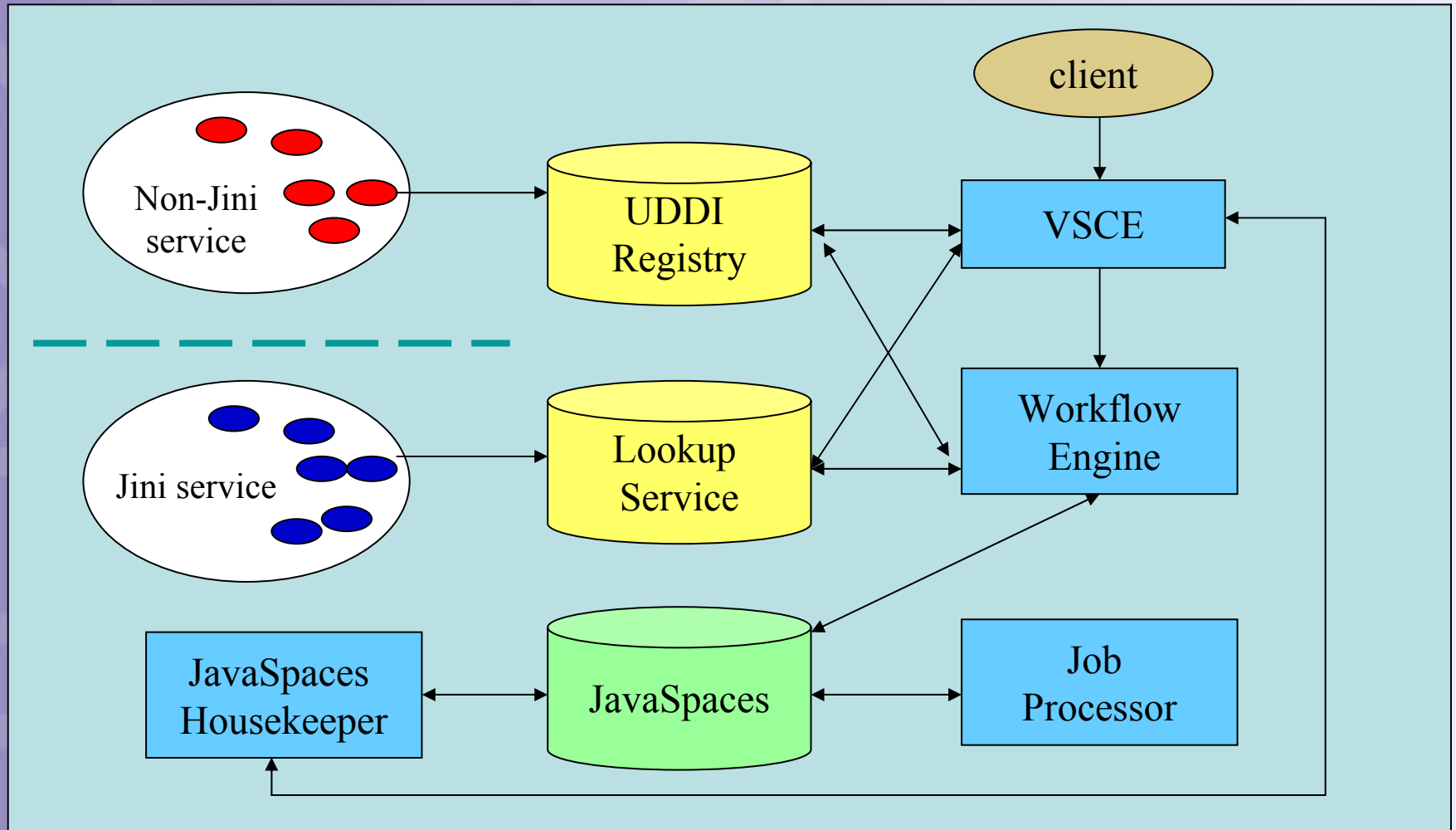
# JISGA Components

- A *WorkflowEngine* service provides the interface for job (described in SWFL) submission and the processing of sequential jobs.
- A *JobProcessor* service mainly processes parallel jobs, which includes partitioning parallel jobs into sequential sub-jobs and then processing each sequential sub-job.

# JISGA Components

- The Visual Service Composition Environment (VSCE) provides a graphical tool for generating composite service-based applications through a "drag-and-drop" interface. This generates an application described by an SWFL document.
- A ServiceBrowser service allows a user to graphically and dynamically monitor the services within a JISGA system
- Other service maintenance tools.

# JISGA Structure

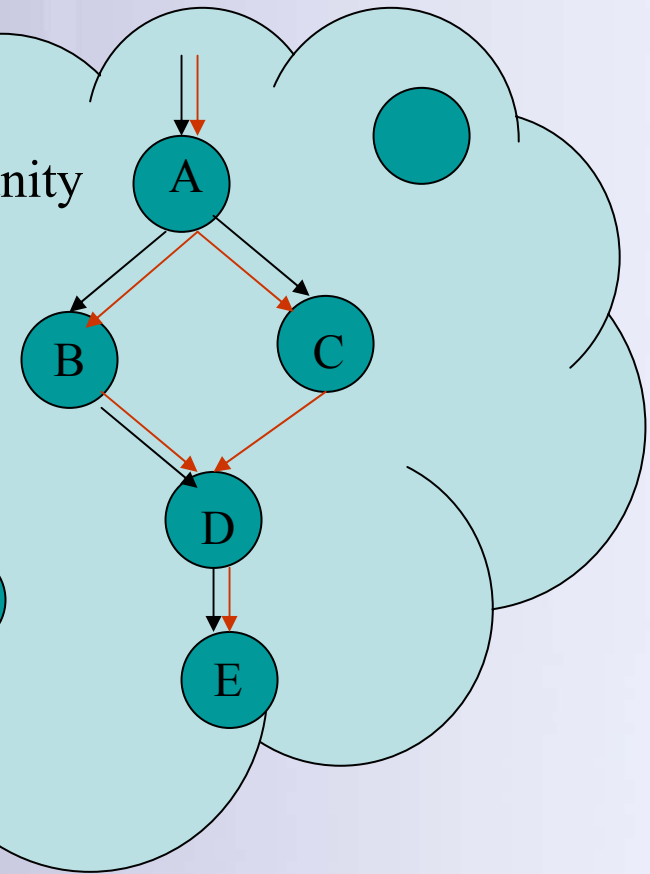


# An Example of Service-based Application

```
<?xml ...>
<JFlowModel>
...
<ControlLink name="AtoB" source="A" target="B"/>
<ControlLink name="AtoC" source="A" target="C"/>
<ControlLink name="BtoD" source="B" target="D"/>
<ControlLink name="DtoE" source="D" target="E"/>

<DataLink name="AtoB" source="A" target="B">..</DataLink>
<DataLink name="AtoC" source="A" target="C">..</DataLink>
<DataLink name="BtoD" source="B" target="D">..</DataLink>
<DataLink name="CtoD" source="C" target="D">..</DataLink>
<DataLink name="DtoE" source="D" target="E">..</DataLink>
...
</JFlowModel>
```

JINI Community



# SWFL: a Service Workflow Language

- XML based for describing interacting Web services.
- extends Web Service Flow Language (WSFL) by
  - supporting the application of all the conditional and loop control constructs of the Java language to the composition of Web services.
  - allowing more general data mapping such as arrays and compound objects

# Why SWFL?

- Both Web Services and OGSA services present their implementation-free interfaces in a platform-free, XML-based language such as WSDL
- Presenting service composite applications in traditional programming languages limits the applications to a range of specific services and a specific time

# Why SWFL?

- Representing applications in an XML-document allows applications be run in any environment where a set of services are available.
- Having a graph-based approach, SWFL gives the flexibility to dynamically partition and schedule services at run time in a distributed environment.

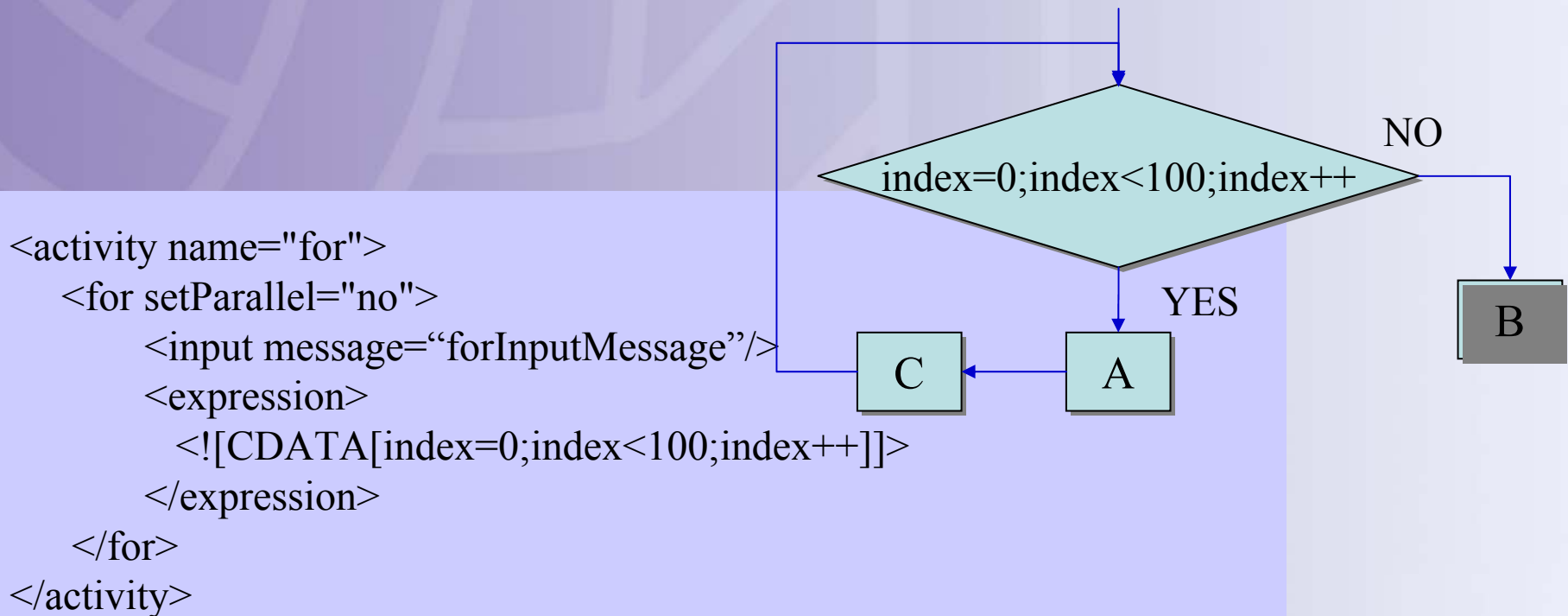
# Loop and Control constructs

- SWFL supports six kinds of activities: *normal*, *if*, *while*, *dowhile*, *for* and *switch*.
- The *normal* type activity is a WSFL activity.
- The *for*, *while*, *dowhile* activities are of type *loopType*, A “*setParallel*” attribute is added to allow parallelism in a loop.
- The *if* and *switch* activities are of type *controlType*.
- Both *loopType* and *controlType* are extensions of *wsdl:operationType*

# Loop and Control constructs

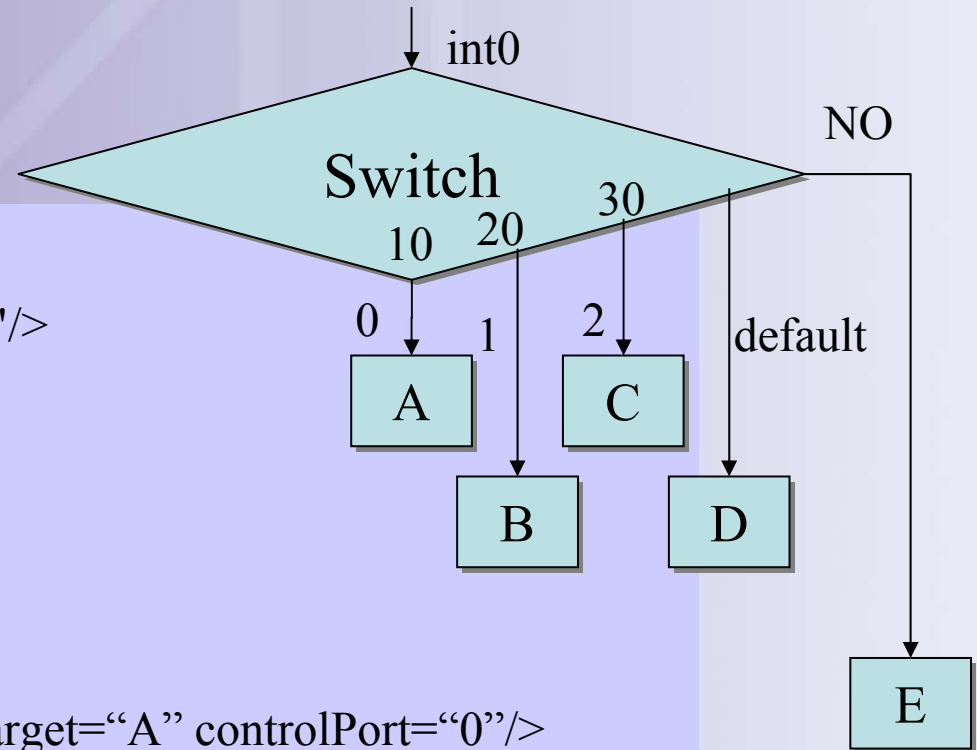
```
<xsd:complexType name="jactivityType">
  <xsd:choice>
    <xsd:element name="normal" type="wsfl:activityType"/>
    <xsd:element name="while" type="loopType"/>
    <xsd:element name="dowhile" type="loopType"/>
    <xsd:element name="for" type="loopType"/>
    <xsd:element name="if" type="controlType"/>
    <xsd:element name="switch" type="controlType">
      <xsd:key name="CasePortName">
        <xsd:selector xpath="case"/>
        <xsd:field xpath="@port"/>
      </xsd:key>
    </xsd:element>
  </xsd:choice>
  <xsd:attribute name="operation" type="NCName"/>
</xsd:complexType>
```

# An Example of *for* Loop Activity



```
<controlLink name="for_A" source="for" target="A" controlPort="YES"/>
<controlLink name="for_B" source="for" target="B" controlPort="NO"/>
<controlLink name="A_C" source="A" target="C" />
<controlLink name="C_for" source="C" target="For" />
```







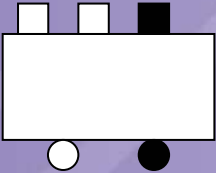
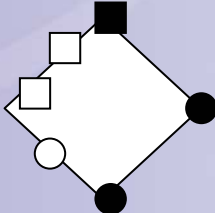
# An Example of *switch* Loop Activity



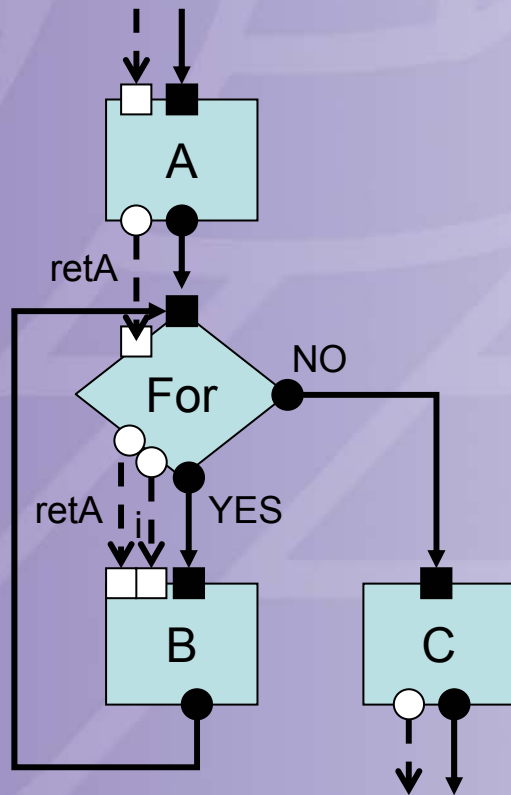
```
<switch name="S" expression="int0">  
  <input message="switchInputMessage"/>  
  <case port="0">10</case>  
  <case port="1">20</case>  
  <case port="2">30</case>  
  <defaultCase port="default"/>  
</switch>
```

```
<controlLink name="S_A" source="S" target="A" controlPort="0"/>  
<controlLink name="S_B" source="S" target="B" controlPort="1"/>  
<controlLink name="S_C" source="S" target="C" controlPort="1"/>  
<controlLink name="S_D" source="S" target="D" controlPort="default" />  
<controlLink name="S_E" source="S" target="E" controlPort="NO" />
```

# Graphical Representation of SWFL Job Descriptions

Graphical Symbol	SWFL Concept	Graphical Symbol	SWFL Concept
	Data input port		Control input port
	Data output port		Control output port
	Data Link		Control Link
	Normal and assign activity		Control activity

# An Example



```
retA = activityA(..);  
for( i=0; i<100; i++){  
    activityB(retA, i);  
}  
retC = activityC();
```

# Service Workflow Language (SWFL)

```
<activity name="for">
  <for setParallel="no">
    <input message="ForInput"/>

    <expression><![CDATA[index=0;index<100;index++]]></expression>
  </for>
</activity>

<dataLink name="A_For" source="A" target="For">
  <swflMap sourceMessage="AOutput" targetMessage="ForInput">
    <part>
      <sourcePart name="retA"/><targetPart name="retA"/>
    </part>
  </swflMap>
</dataLink>
```

# Service Workflow Language (SWFL)

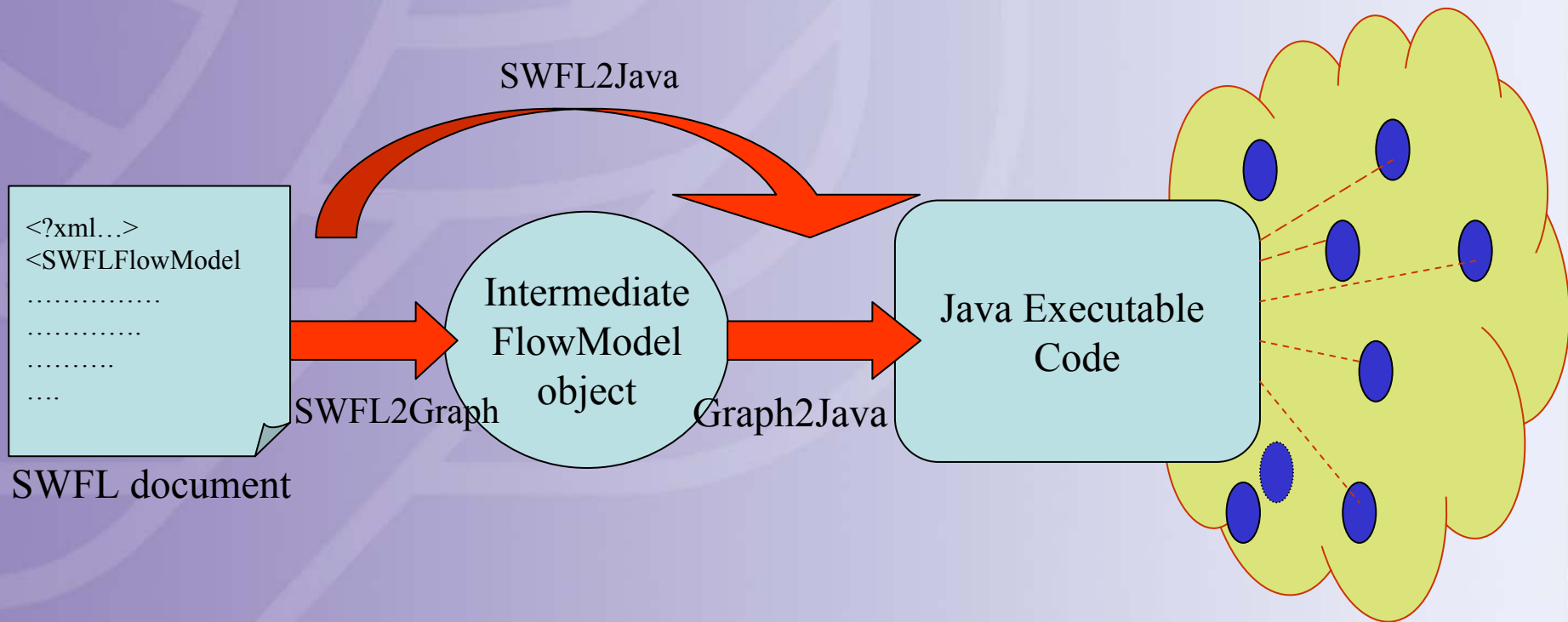
```
<dataLink name="A_For" source="A" target="For">
  <swflMap sourceMessage="ForOutput" targetMessage="BInput">
    <part>
      <sourcePart name="retA"/><targetPart name="retA"/>
      <sourcePart name="i"/><targetPart name="i"/>
    </part>
  </swflMap>
</dataLink>

<controlLink name="A_For" source="A" target="For" />
<controlLink name="For_B" source="For" target="B" controlPort="YES"/>
<controlLink name="For_C" source="For" target="C" controlPort="NO"/>
<controlLink name="B_For" source="B" target="For" />
```

# Implementation Aspects

- A tool called SWFL2Java converts the description of a job in SWFL into Java executable code.
- A Workflow Engine provides an execution environment to run the jobs described in SWFL.

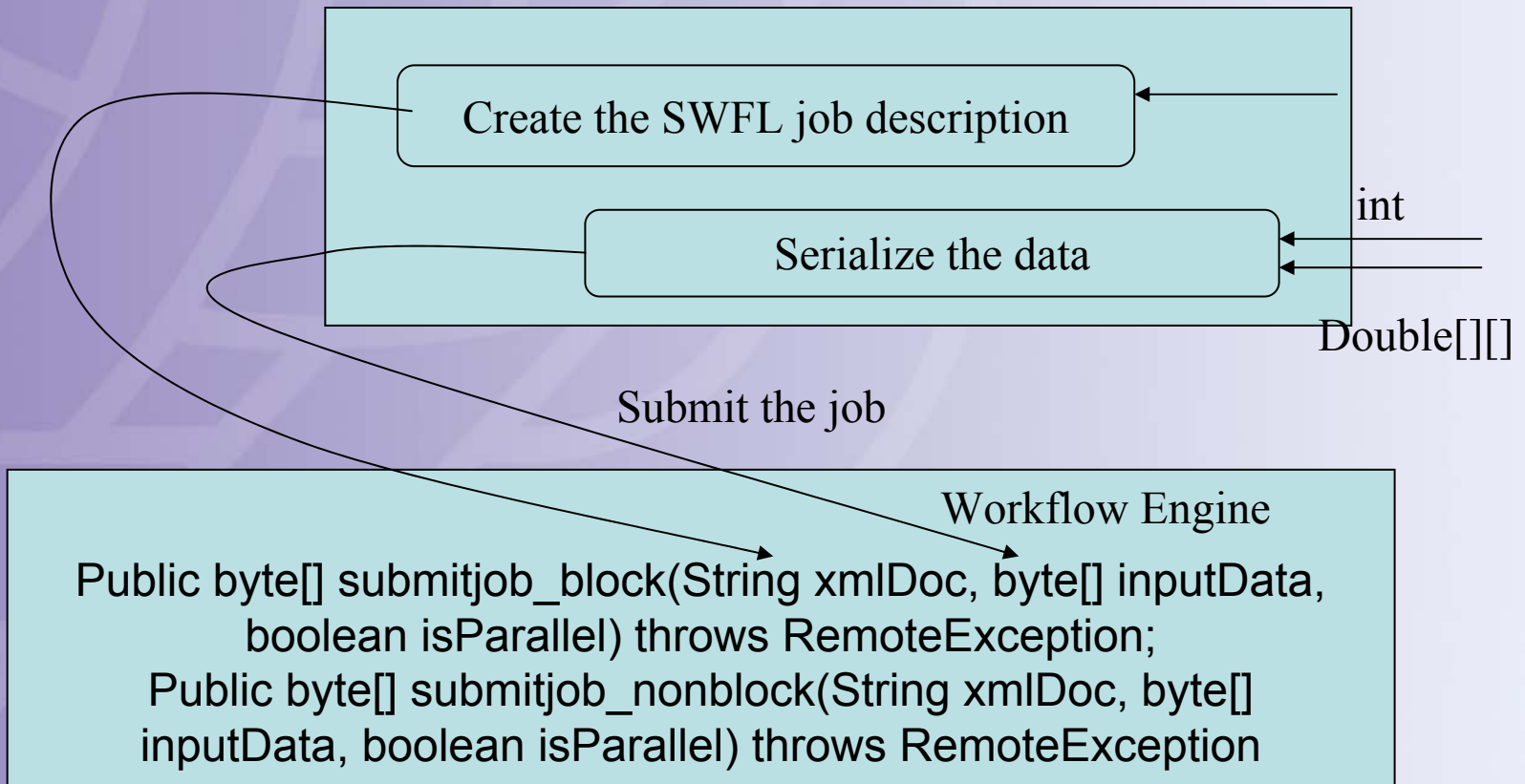
# Workflow Engine



# Workflow Engine Interfaces

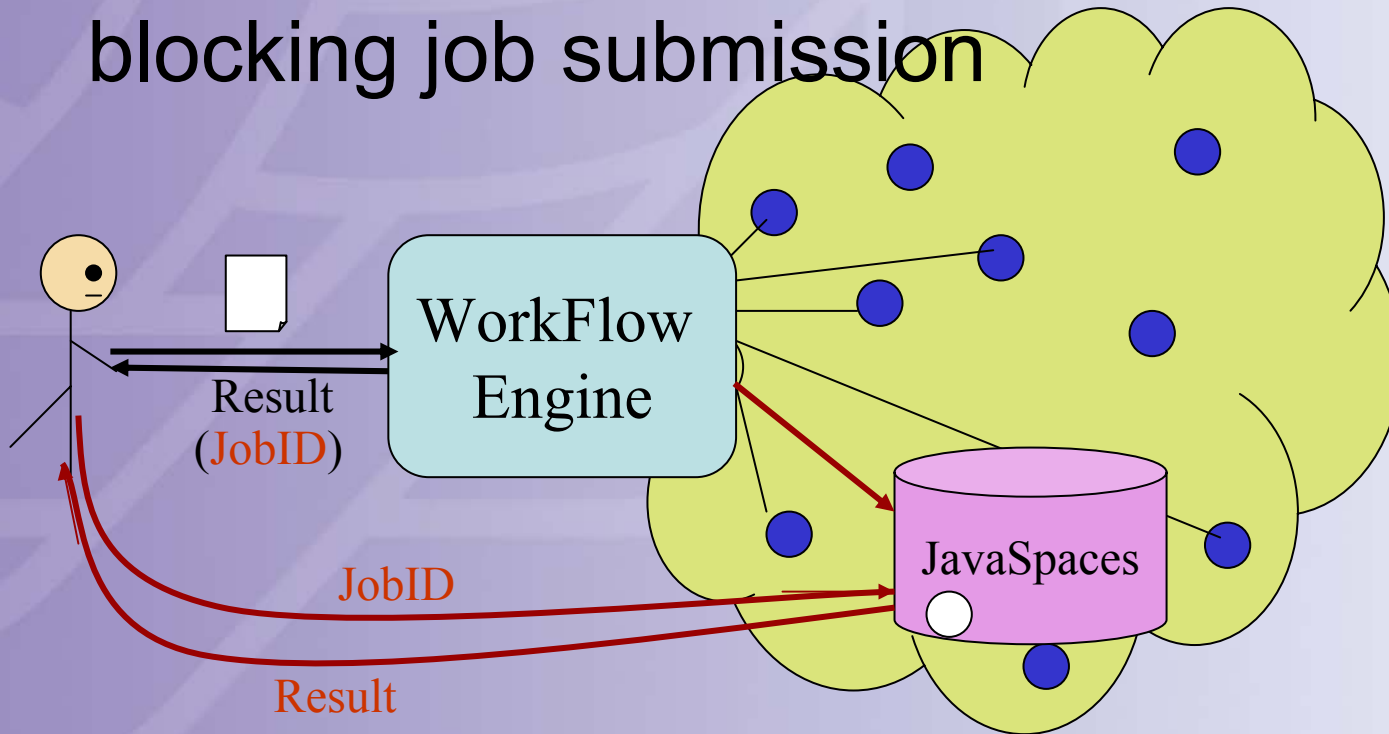
- `Public byte[] submitjob_block(String xmlDoc, byte[] inputData, boolean isParallel) throws RemoteException;`
- `Public byte[] submitjob_nonblock(String xmlDoc, byte[] inputData, boolean isParallel) throws RemoteException;`

# Submit a Job

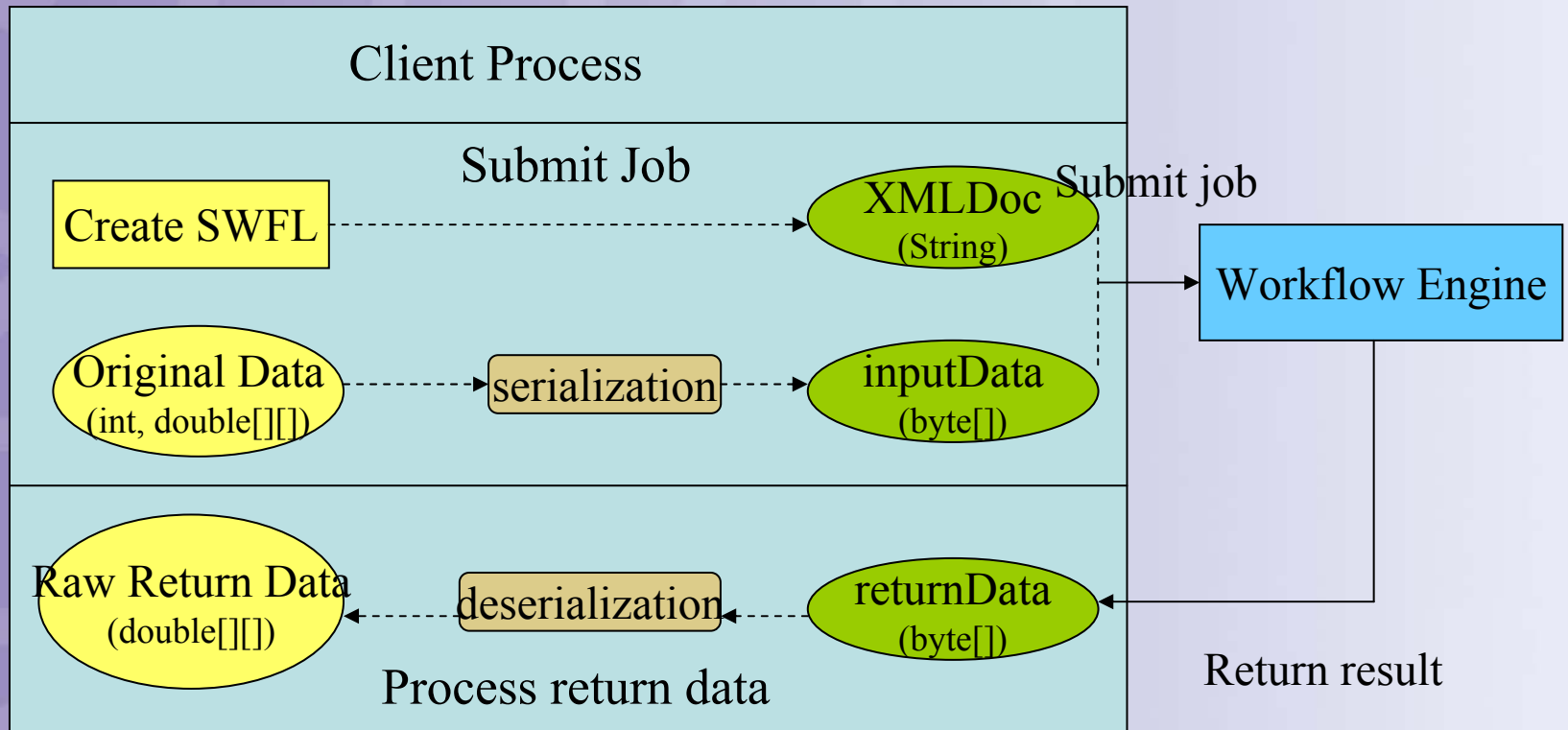


# Non-Blocking and Blocking Call

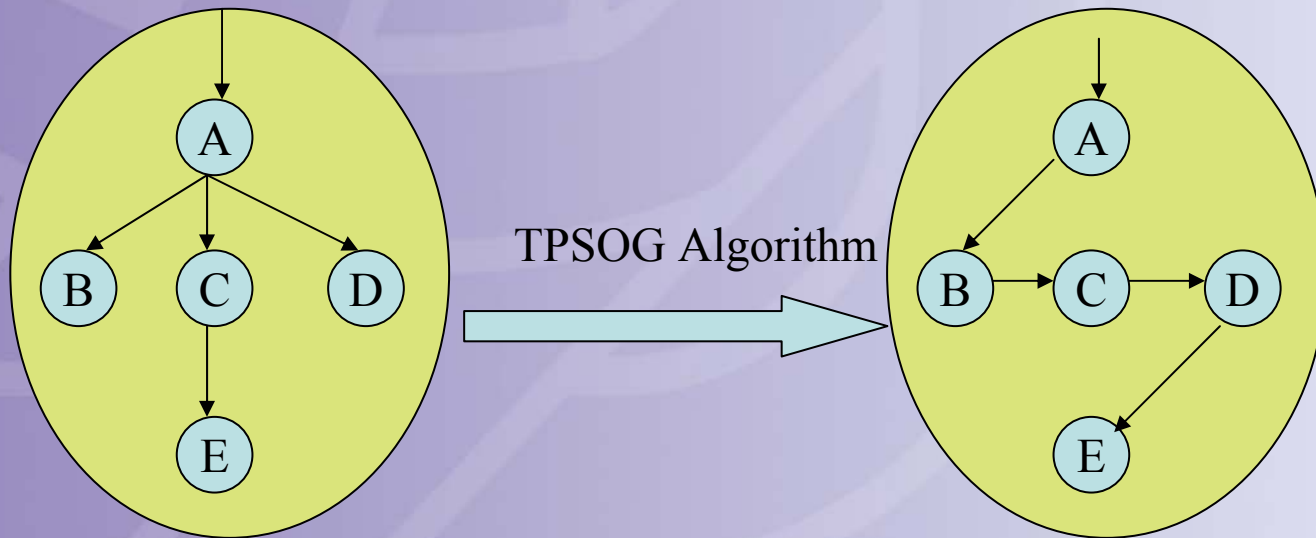
- Allows both non-blocking and blocking job submission



# Data Transformation in a client process

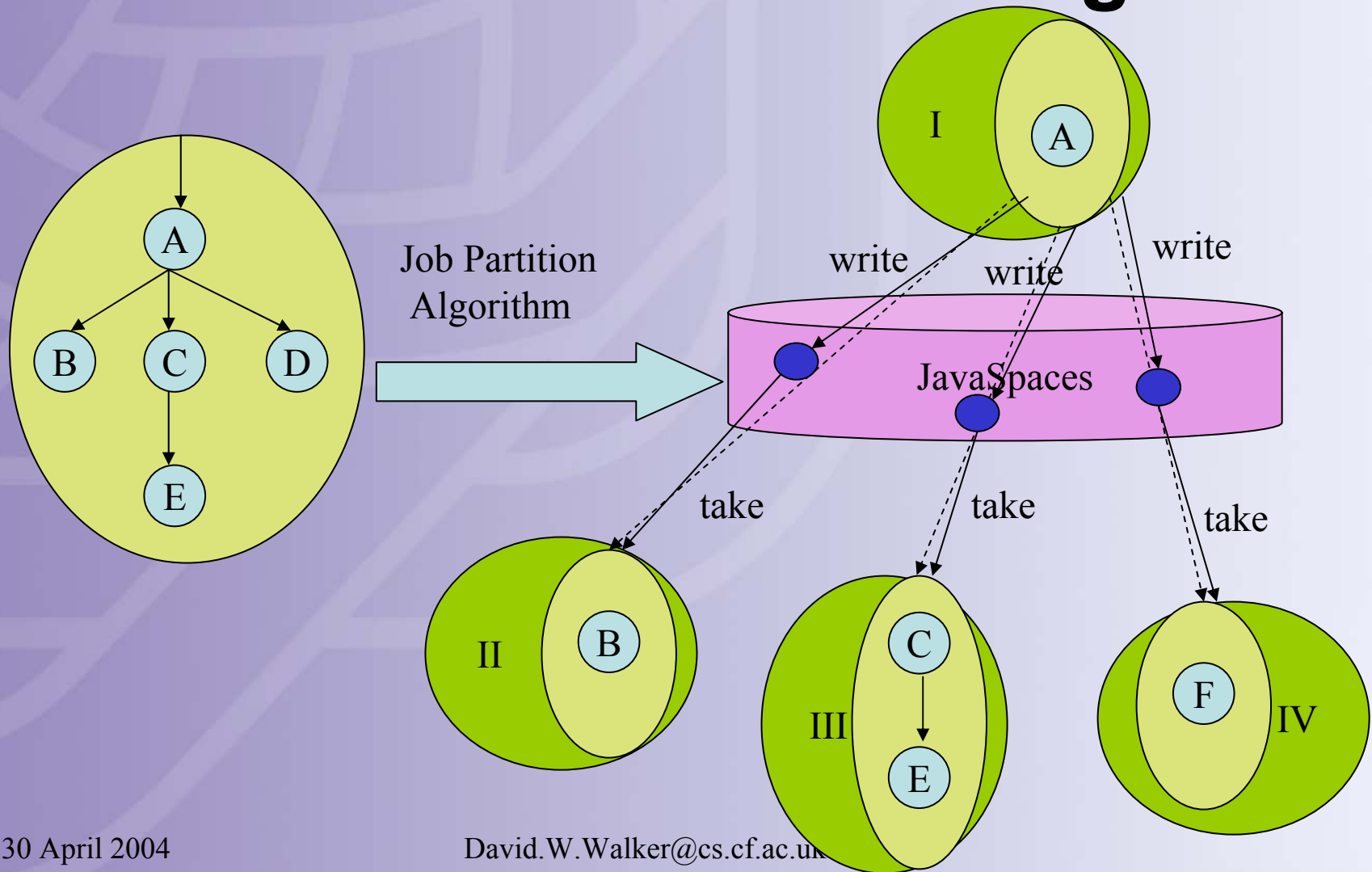


# Sequential Job Processing



TPSOG: Task Processing Sequential Order Generating

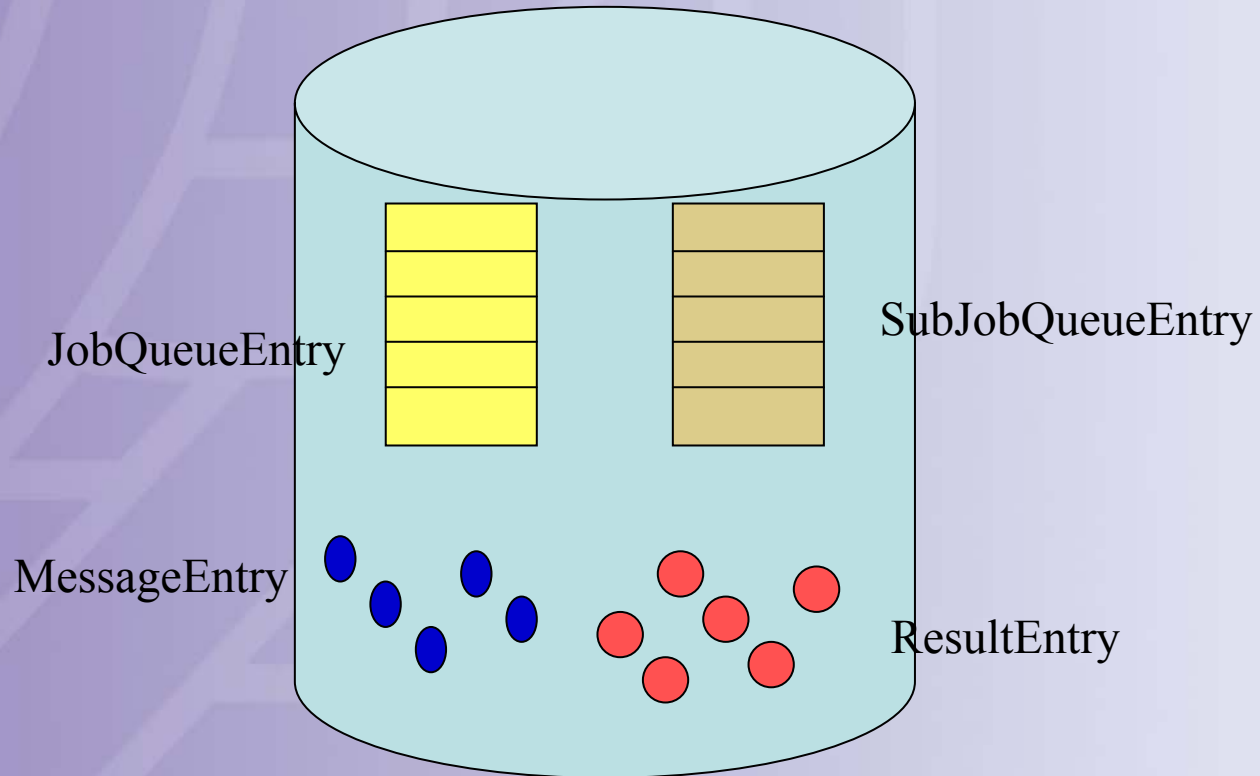
# Parallel Job Processing



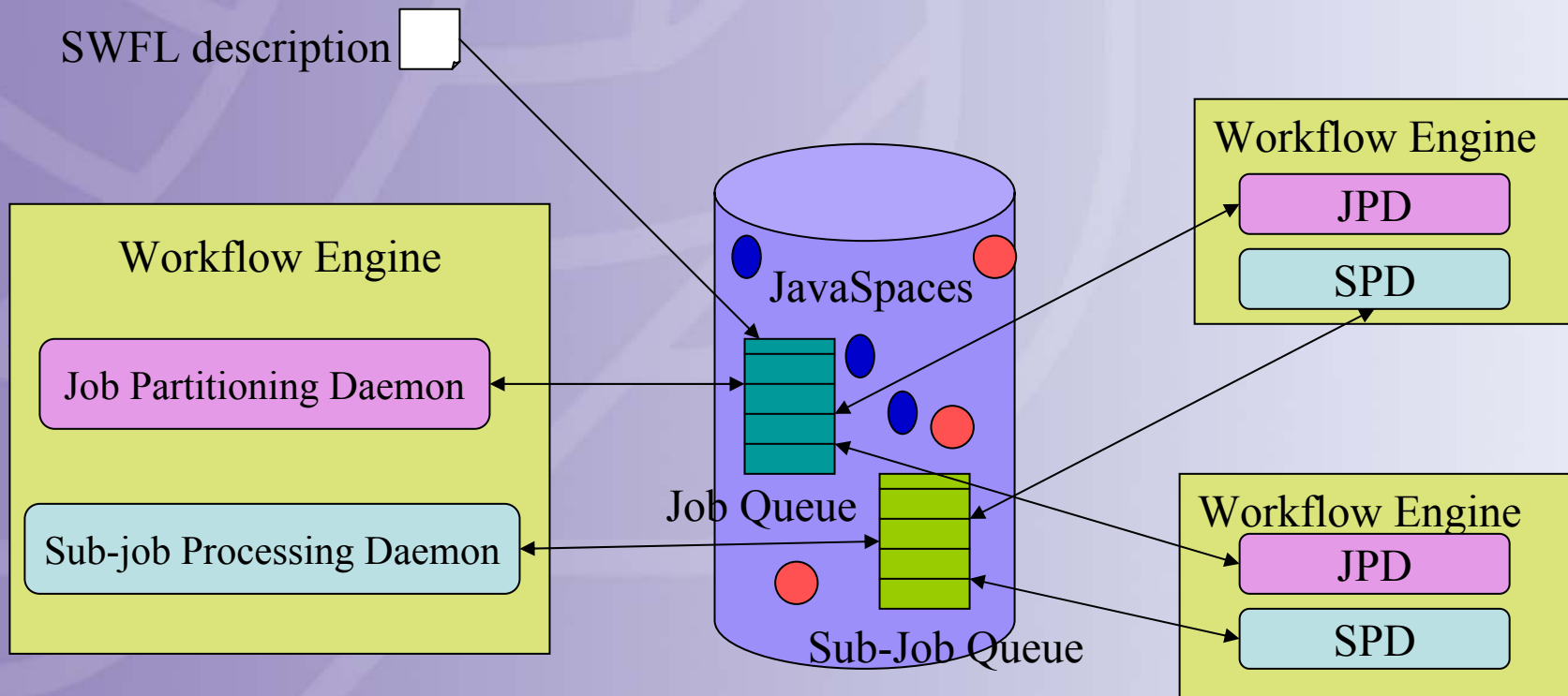
# JavaSpaces in JISGA

- Four kinds of objects can be stored in a shared JavaSpaces space:
  - A JobQueueEntry object
  - A SubJobQueueEntry object
  - MessageEntry objects
  - ResultEntry objects

# JavaSpaces in JISGA



# Parallel Job Processing



# JavaSpace Housekeeper

- Updating the *isready* state of each sub-job in the sub-job queue, removing the sub-jobs in the queue that have been forgotten for a long time.
- Removing any message that has not been used for a sufficiently long time, and then storing them locally.
- Removing any result objects, storing them locally and waiting for the result to be claimed.

# Limitations of SWFL

- In the original definition of a parallel for loop, there is no mechanism provided to allow message-passing between the parallel processes, which assumes that all the processes are running the same code but on different data with no communication between them.

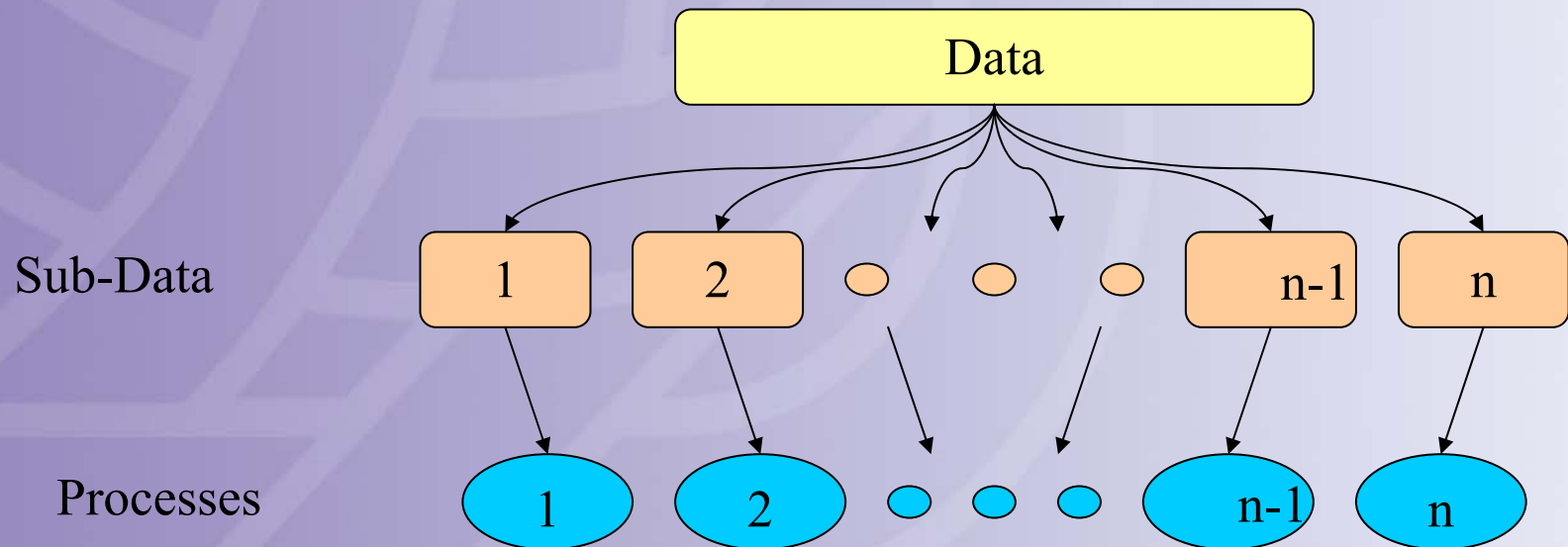
# Extensions to SWFL

- A label is specified for each parallel process.
  - A particular job can be assigned to a particular parallel process to support MIMD parallelism.
  - A message-passing path can be specified
- Allow specification of *shared* and *non-shared* variables to support shared-memory applications
  - Accessing a *shared* variable requires mutual exclusion

# Extensions to SWFL

- Extensions to data mapping specification in SWFL to specify how a data structure is to be distributed across the processes.
  - In order to adequately support SPMD (Single-Program Multiple-Data) parallelism.
  - The mapping is implemented in two steps
    - a single-to-multiple data decomposition mapping
    - Mapping sub-data to processes

# Extensions to SWFL

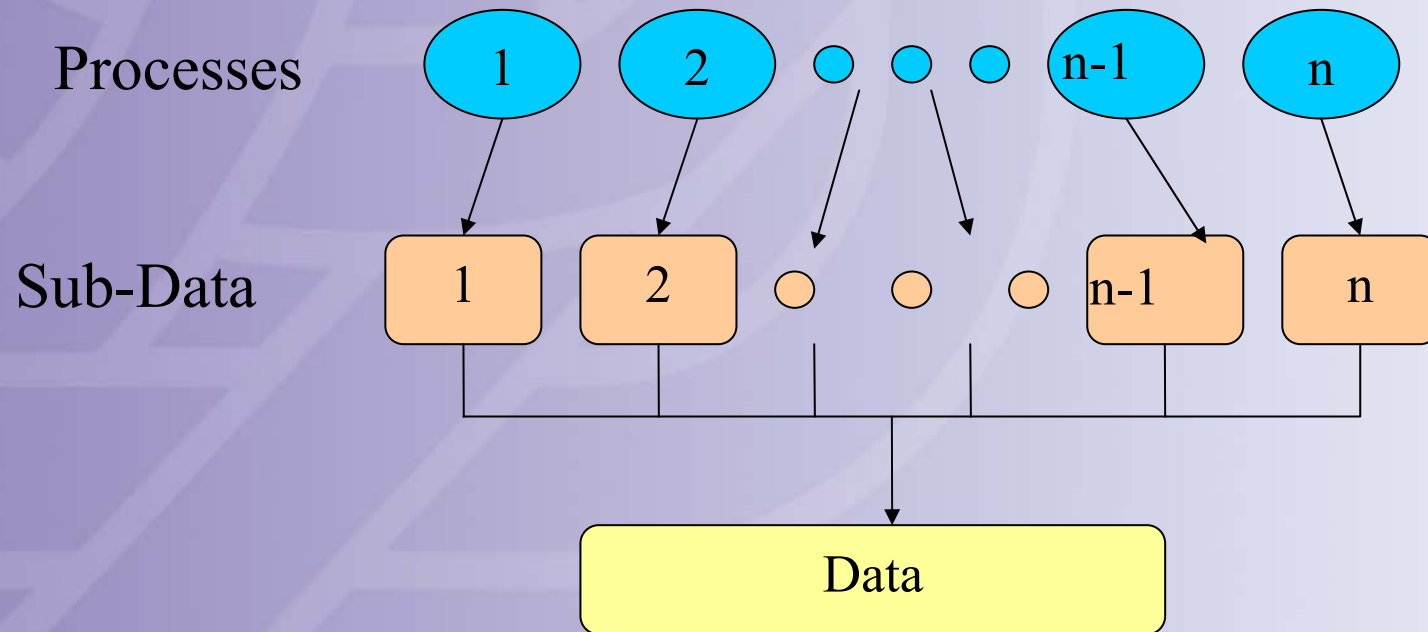


Mapping a data structure to parallel processes.

# Extensions to SWFL

- A “multiple-to-single” data mapping mechanism in SWFL to support data merging. It can be
  - data reduction in which all the data sub-parts are combined using a binary operation such as addition, multiplication, max, min, logical and, etc.
  - combining a data sub-parts into a larger data structure by allocating each element in the sub-parts to a specific location in the target data structure.

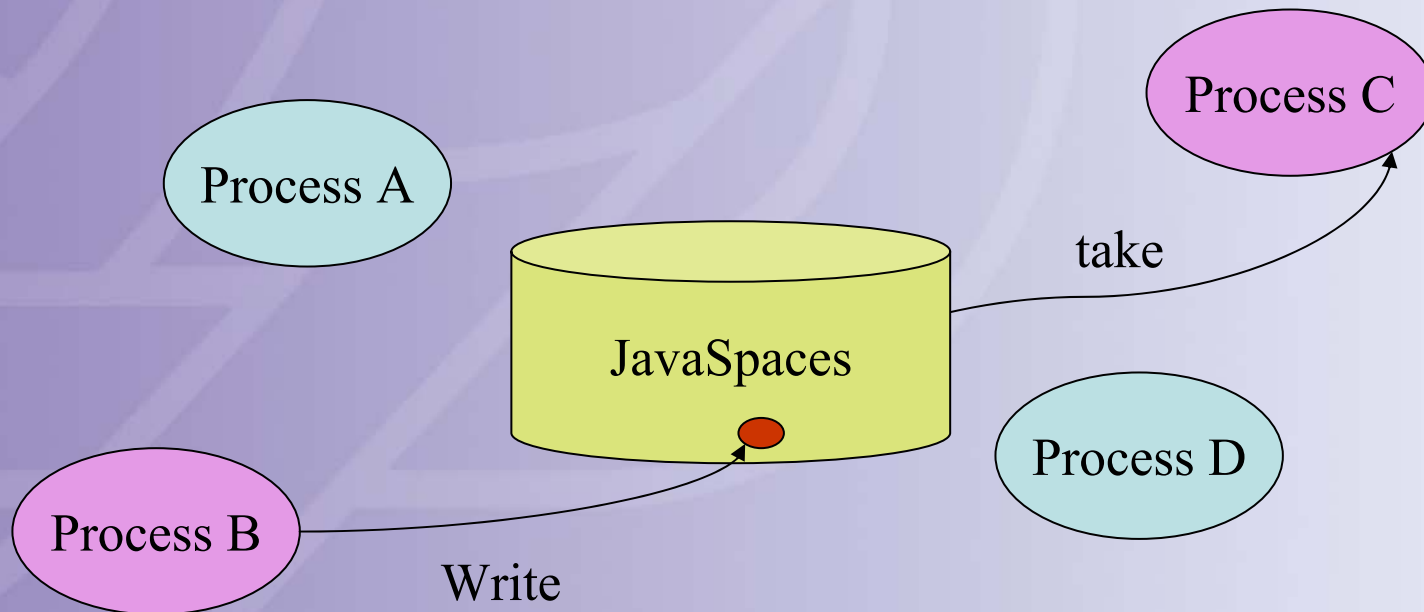
# Extensions to SWFL



Data merging

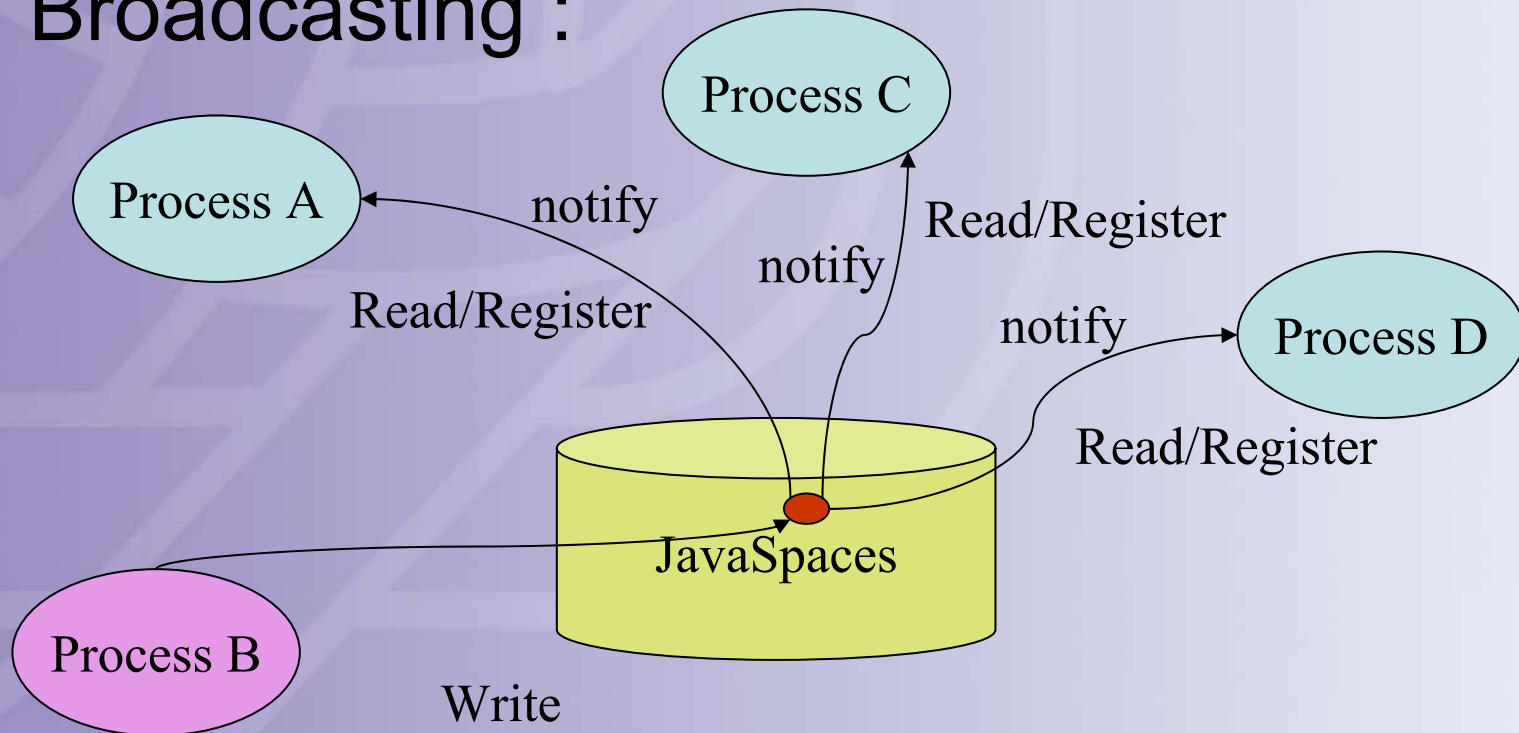
# Implementation Issues

- Point-to-point message passing:



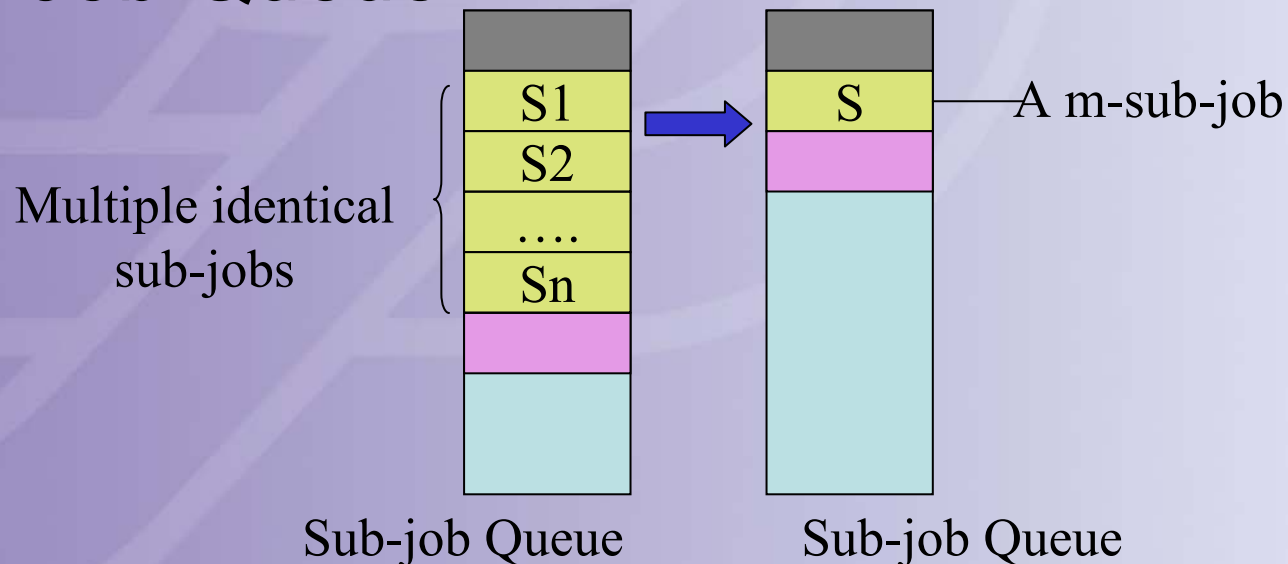
# Implementation Issues

- Broadcasting :



# Implementation Issues

- Allow multiple identical sub-jobs to be represented in a single sub-job description and submitted to the Sub-Job Queue.

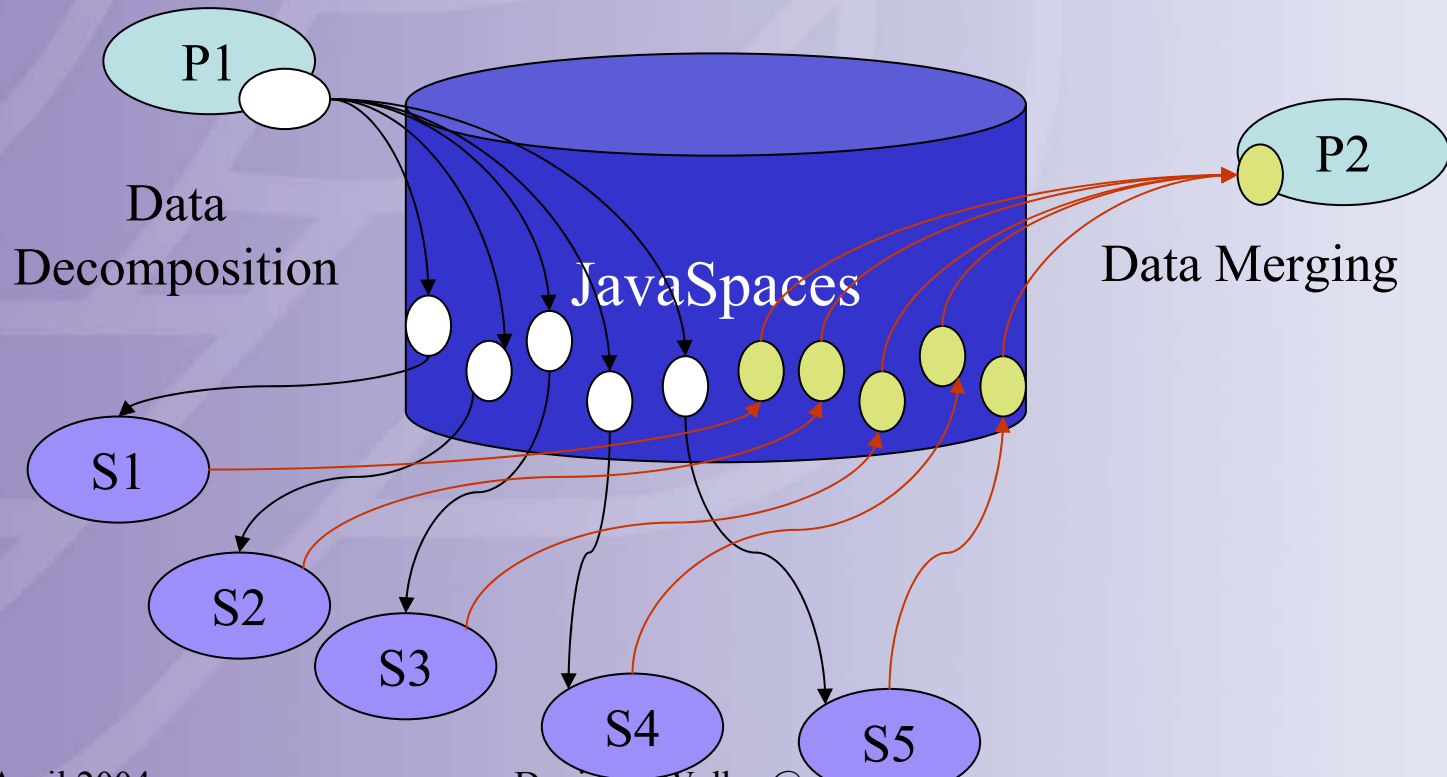


# Implementation Issues

- Shared Data: only *read* and *update* operations allowed.
  - Update is combined *take-and-write* operations.
- Non-shared Data: only *read* and *write* operations.

# Implementation Issues

- Data decomposition and merging



# Part 3:

## Examples/Demonstration

# Part 3 Outline

- Create a Web Service
- Access a Web Service
- Undeploy a Web service
- Deploy a Jini service as a Web Service.

# Techniques used

- Apache Axis 1.1: an implementation of the SOAP (Also WSDL2Java and Java2WSDL tools).
- Java Web Service Development Pack (JWS DP) 1.2.: an integrated toolkit for building, testing and deploying XML applications, Web services, and Web applications
- UDDI4J: a tool for interacting with a UDDI server

# Create a Web service by using Axis.

- Start a Web Server and the JWSDP UDDI Registry Server.
  - Simply start Tomcat and Xindice.
- An Example Calculator.java

```
package MyTest;

public class Calculator{
    public int add(int p1, int p2){ return p1+p2; }
    public int subtract(int p1, int p2){ return p1*p2; }
}
```

# Create a Web service by using Axis.

- Compile Calculator.java.
- Copy the class with its package structure into axis' Tomcat servlet distribution (usually *webapps\axis\WEB-INF\*)
- Create an XML-based WSDD (Web Service Deployment Document) to specify the class.)

# Create a Web service by using Axis

## Example of a WSDD

```
<deployment name="test" xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
  <service name="Calculator" provider="java:RPC" >
    <parameter name="className" value="MyTest.Calculator"/>
    <parameter name="allowedMethods" value="*/>
  </service>
</deployment>
```

# Create a Web service by using Axis

- Deploy the Calculator class as a Web Service:

```
java org.apache.axis.client.AdminClient  
    deploy.wsdd
```

# Directly Access a Web Service

- String endpoint = "http://danny.cs.cf.ac.uk:8080/axis/servlet/AxisServlet";
- 
- String method = args[1]; // "add" or "subtract"
- Integer i1 = new Integer(args[1]);
- Integer i2 = new Integer(args[2]);
  
- Service service = new Service();
- Call call = (Call) service.createCall();
  
- call.setTargetEndpointAddress( new java.net.URL(endpoint) );
- call.setOperationName( new QName("Calculator", method) );
- call.addParameter( "op1", XMLType.XSD\_INT, ParameterMode.IN );
- call.addParameter( "op2", XMLType.XSD\_INT, ParameterMode.IN );
- call.setReturnType( XMLType.XSD\_INT );
- Integer ret = (Integer) call.invoke( new Object [] { i1, i2 } );
- System.out.println("Got result : " + ret);

# Indirectly Access a Web Service

- Obtain WSDL for deployed services by using ?WSDL
- Build stubs, skeletons, and data types from WSDL by using WSDL2Java.
  - A package path is created based on the URL address of the service.
  - Also create
    - Calculator.java
    - CalculatorService.java
    - CalculatorServiceLocator.java
    - CalculatorSoapBingStub.java

# Indirectly Access a Web Service

- Test.java

```
CalculatorService service = new CalculatorServiceLocator();  
  
Calculator calc = service.getCalculator();  
  
// Make the actual call  
int ret1 = calc.add(200, 300);  
  
int ret2 = calc.subtract(200, 300);
```

# Undeploying a Web service

- undeployment.wsdd

```
<undeployment
xmlns="http://xml.apache.org/axis/wsdd/">
  <service name="Calculator"/>
</undeployment>
```

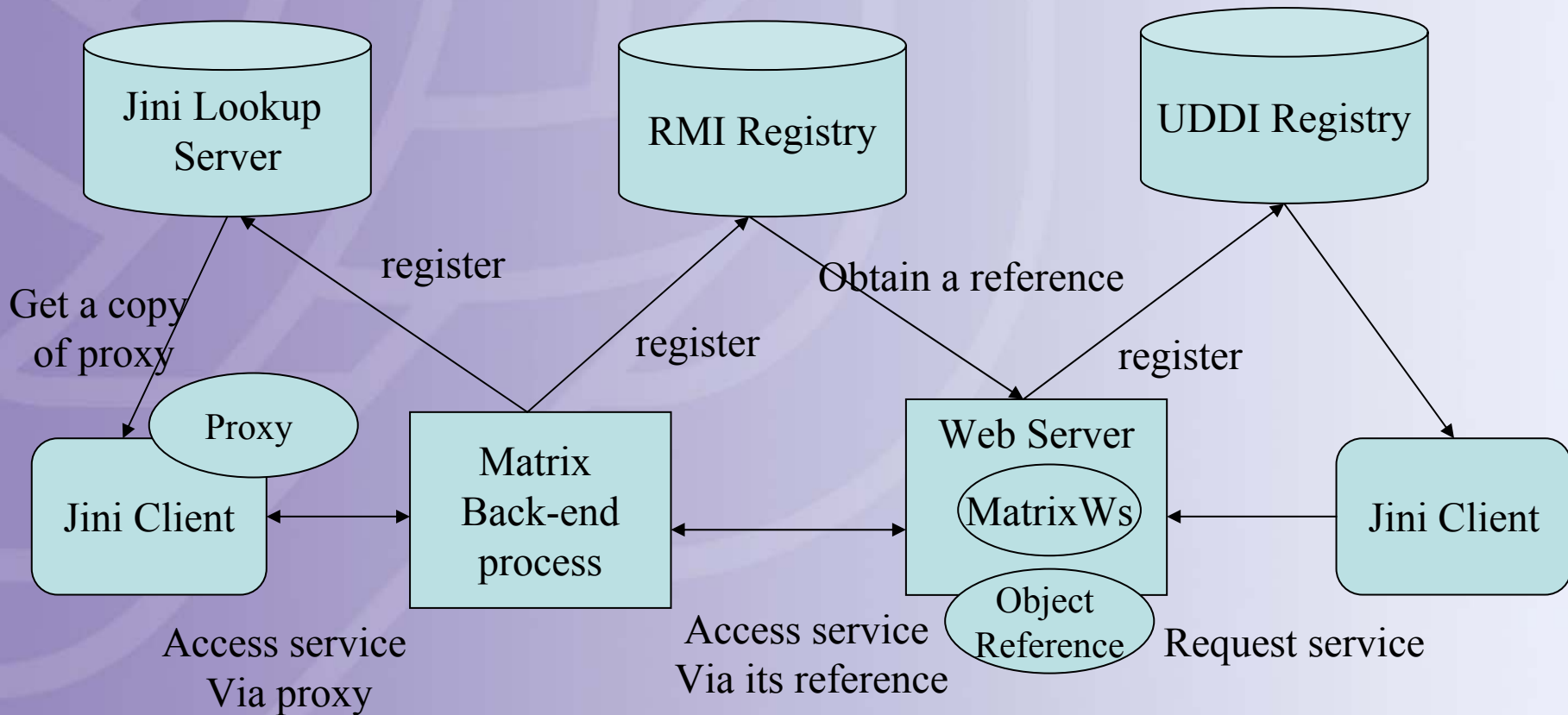
- Run

```
java org.apache.axis.client.AdminClient
undeployment.wsdd
```

# Deploying a Jini Service as a Web Service

- Register the backend object to an RMI registry.
- Create a *WSTrigger* for the Jini Service.
- Deploy the *WSTrigger* as a Web Service.

# Deploy a Jini Service as a Web Service



# Part 4:

## Conclusions and Future Work

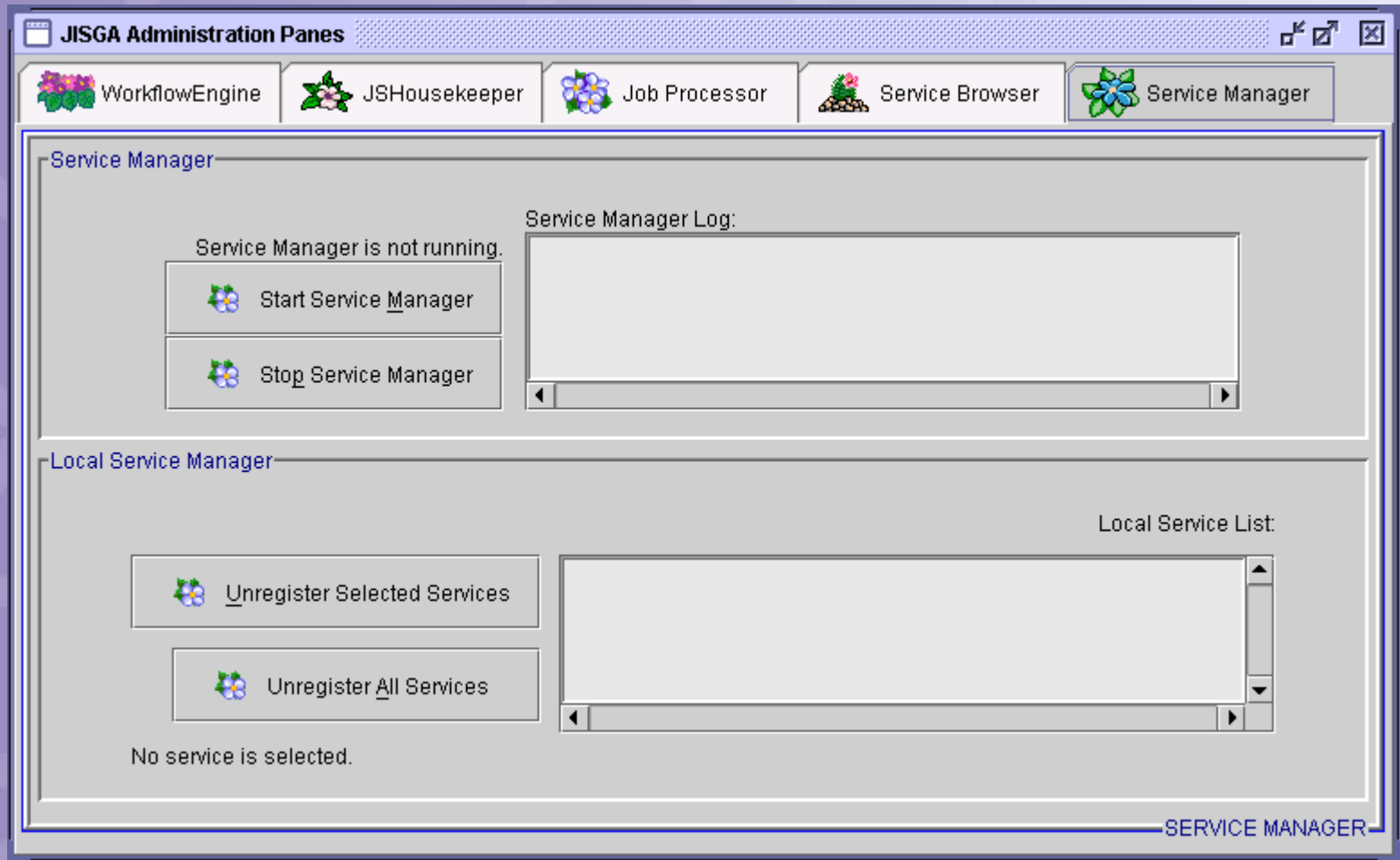
# Conclusions

- Introduction of standard Web services into existing Grid architecture provides an efficient way to build a real, global Grid.
- An implementation-free job description language and corresponding execution environment are two critical issues in building such a Grid

# Current and Future Research

- Updating JISGA---JISGA1.2
  - Extended SWFL supports
  - More attractive interface
  - Use Jini 2.0
  - Security supports
  - Job tracking and performance monitoring.
  - Testbeds and applications

# JISGA 1.1 Administration Pane



# JISGA 1.2 Administration Pane

The screenshot displays the JISGA Administrator application window, which is divided into several functional panels:

- Initial Panel:** Contains a "Register a new lookup server:" section with a text input field, a "Register" button, and an "Unregister" button. Below this is a "Lookup Server List: (\*--local LU server)" area with a large empty list box. A "Groups:" dropdown menu is set to "JISGA1 2", with "Remove" and "Add" buttons. At the bottom, it indicates "Event Manager is not running." and has a "Start Event Manager" button.
- Control Panel:** Features a "Current Registered Servers:" list box. To its right are three buttons: "Start As a JP", "Start As a WE", and "Start As a JSH". Below these is a legend defining abbreviations: JP --Job Processor, WE --Workflow Engine, JSH --JavaSpace Housekeeper, J --running as a Job Processor, W --running as a Workflow Engine, H --running as a JavaSpace Housekeeper, j --ready to run as a Job Processor, w --ready to run as a Workflow Engine, h --ready to run as a JavaSpace Housekeeper.
- Event Display Panel:** A large teal-colored area on the right side of the top section, currently empty.
- Service Browser Panel:** Located in the bottom-left, it shows "Current running services:" and "Current ready-to-run services:" as empty list boxes. It includes "Unregister" and "Register" buttons.
- JavaSpace Monitor Panel:** Occupies the bottom-right, containing a "Visualized Monitor for JavaSpace" with a play button and a green square icon. Below it are input fields for "Message's maximum stay time:" (1440 mins) and "Job's maximum stay time:" (1440 mins), with "Clear" and "Update" buttons.
- Administrative Panels:** A central section with "Result entries:", "Message entries:", "Old Unused Messages:", "Job queue:", "Sub-job queue:", and "Old Undone Jobs:" labels, each followed by an empty list box.

The Windows taskbar at the bottom shows the Start button, several open applications (David - O..., JiscaCourse, foster\_wsrif, Inter..., Jini packa..., AllHands2..., Command..., C:\WIND...), the JISGA Administrator window, system tray icons, and the time 15:53.

# Current and Future Research

- Also
  - Development of a Visual Service Composition system to build service composite applications in an easy-to-use, graphical environment.
  - More experiments in integration of Jini services, CORBA services and OGSA Grid services(GT4?) by using a workflow engine
  - A new version of the workflow engine supporting BPEL4WS and WSCI.

-----> **GSiB project**

# **Grid-Service-in-an-Box (GSiB)**

## **— Background**

- Building on Grid and Web service technologies, the OGSA proposes an architecture for a new generation of Grid infrastructure
- In general, scientists cannot be expected to keep breast of the rapid development of Grid technologies.

# GSiB – Motivation and Targets

- Easy-to-use tools are needed to allow scientists to take full advantage of the new Grid without them needing detailed knowledge of the underlying infrastructure.
- GSiB project is developing a visual problem-solving environment for all service users (clients and providers).

# **GSiB Interfaces and Functionalities**

- GSiB consists of two packages:
  - The service provider GUI package
  - The service client GUI package

# The Service Provider GUI Package

- Service Deployment/Undeployment Interface
  - Automatic deployment of existing software routines or libraries as OGSA-compliant Grid services.
  - Visual Service Composition Environment (VSCE) for building and deploying a composite service.
  - Updating/undeployment of existing Grid Services. Consideration of how to avoid interrupting higher-level services and running applications.

# The Service Provider GUI Package

- Service Configuration interface for setting up or changing a service's security policy and lifetime.
- Service Monitor Interface for dynamically monitoring the distribution, performance and workload of Grid services and resources.
- Other interfaces include Authorization interface, and Lifecycle Management Interface.

# The Service Client GUI Package

- Visual Service Composition Environment (VSCE) for building a composite Grid application.
  - It allows a user to create a representation of a Grid application by drawing its workflow.
  - Then VSCE transforms the graphic workflow model into a XML-based workflow description document.

# The Service Client GUI Package

- Workflow Engine component provides an execution environment for Grid applications described in an XML-based workflow description language.
- Job Tracking and Monitoring Interface
- Service Browser for browsing and querying services.

# GSiB

The screenshot displays the Visual Service Composition, Deployment and Execution Environment. The main workspace shows a workflow diagram with a 'Flowsource' component connected to a 'Flowsink' component. A 'Service Discovery' dialog box is open, showing search results for 'Global110' services. The dialog includes search criteria and a list of results with details like description, service key, and overview URL.

**Visual Service Composition, Deployment and Execution Environment**

File Edit View Insert Tools Service Browser Deployment Monitor Other

Flowmodel Design Service Browser Deployment and Monitor Browser

none

**Working Frame**

Workflow Composition: untitled0.sfm

Property Inspector/Service Browser

- Web Services
  - #Global110
  - #Global110 #1
  - #Global110 #2
  - #Global110 #3
  - #Global110 #4
- Jini Services
- Other Services

**Service Discovery**

UDDI Inquiry URL(s) (Multiple inputs separated by '+')  
http://uddi.microsoft.com/inquire

Search by:  
 Service name  
 Service key  
 TModel key

Qualifier:  
 Case sensitive match  
 Exact name match  
 Sort by name ascent  
 Sort by name descent  
 Sort by date ascent  
 Sort by date descent

Row Num: 10 Search: %

1 Web Service found.

- 1. #Global110
  - Description: The Global110 is designed to build a bridge between consumers and providers of information. This struct
  - Service Key: 6ef54665-1731-442e-88e7-35d6f24cf25d
  - TModel: Key = uuid:98e42d82-505a-42ad-b4d8-29958d9aec07
    - Overview URL = <http://www.global110.com/webservices/global110/global110.asmx?WSDL>

Result WSDL Tree

For Help, click Help Topics on the Help Menu.

start Inbox - Out... JiscaCourse 5 Internet... Jini package... main untitled - Paint Command Pr... 3 java EN 100% 16:08

30 April 2004

David.W.Walker@cs.cf.ac.uk

# Web Site

- <http://www.cs.cf.ac.uk/user/Yan.Huang/GridWF/index.htm>