Space-based approach to high throughput computations in UNICORE 6 Grids

Bernd Schuller, Miriam Schumacher

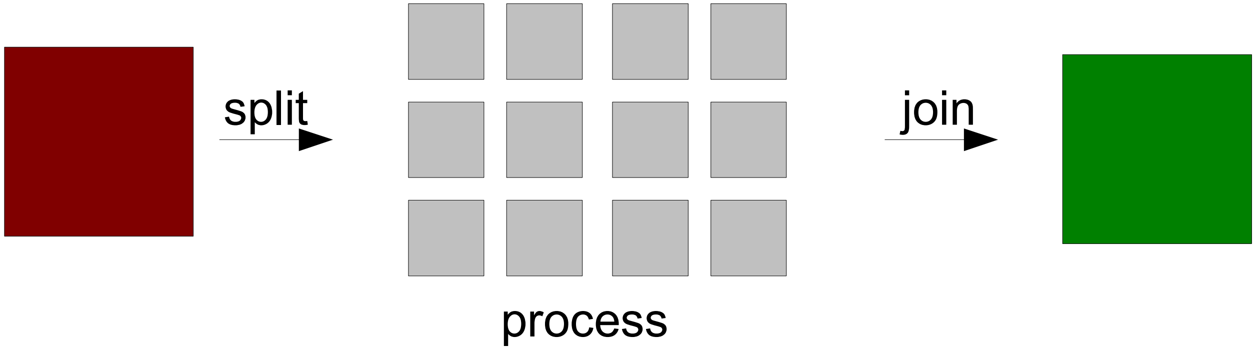
Jülich Supercomputing Cente
Distributed Systems and Grid Computing
Forschungszentrum Jülich GmbH

Outline

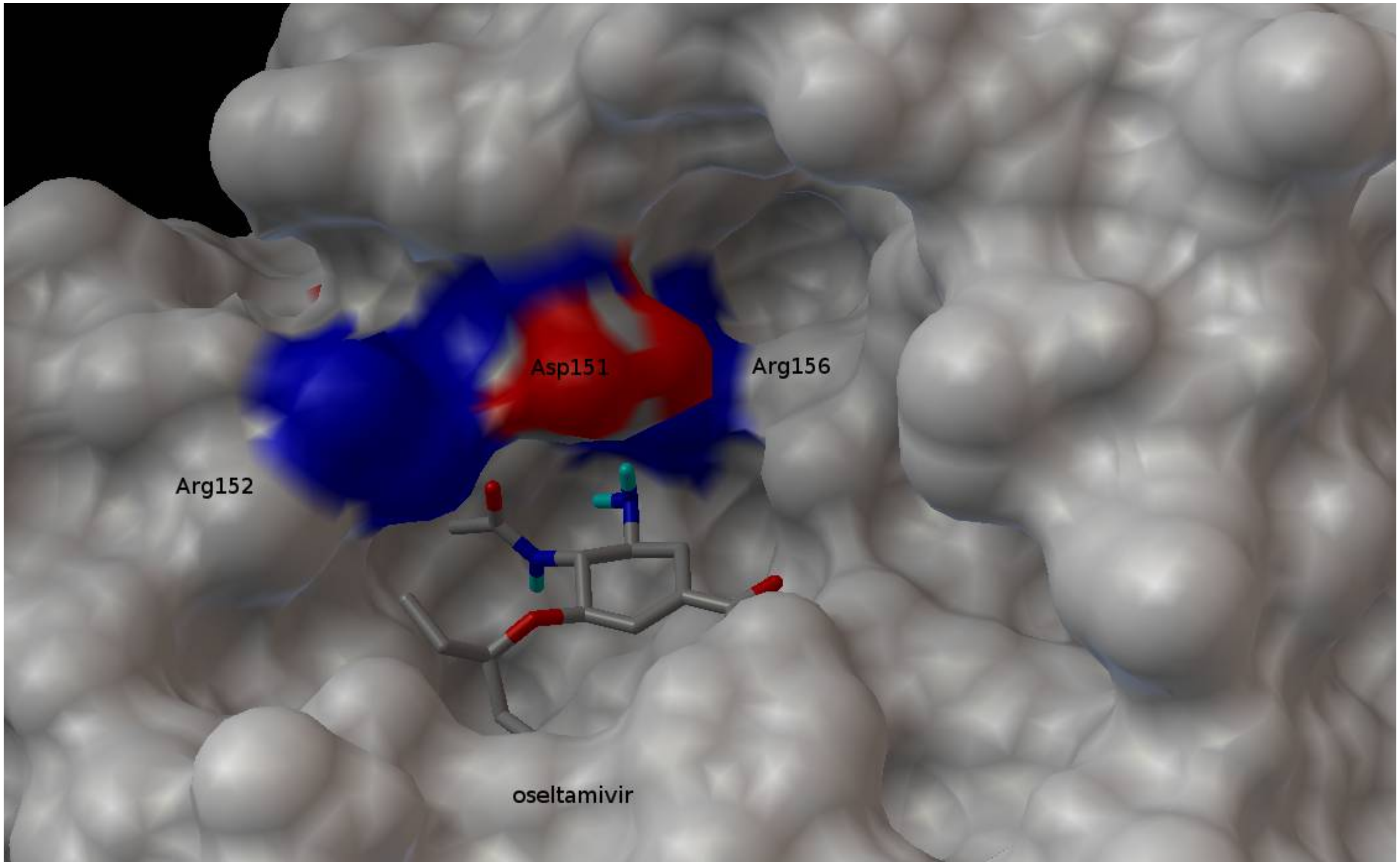
- Motivation: high-throughput computing
- What is a tuple space?
- „XML Spaces“ based on WSRFlite / UNICORE 6
- Job execution using the tuple space
- Performance measurements and results

High-throughput computing – characteristics and challenges

- Characteristics
  - Many (small) jobs, many (small) resources
- Examples
  - Docking (e.g. WISDOM),
  - High-throughput screening, e.g. apply a QSAR model for property prediction for a very high number of structures



Example: find a drug for combating avian flu



Garcia-Sosa, A.T., Sild, S., Maran, U. *ChemMedChem*, submitted **2008.**

Using docking for virtual screening

- Docking is very well suited for massive parallelization: 1 job per docking run
- Docking was run through the UNICORE command client. UNICORE was used for the distribution, running and output recollection of the jobs
- Single UNICORE site
- ~ 1,500 jobs per day on 20 cluster nodes
- Each job took around 15 mins. average real time
- 50-100,000 ligands per virtual screening
- 33-66 days on 20 nodes (or 7-12 days on 100 nodes)
- Promising strategies and molecules for new inhibitors of avian flu have been obtained

Garcia-Sosa, A.T., Sild, S., Maran, U. *ChemMedChem*, submitted **2008.**

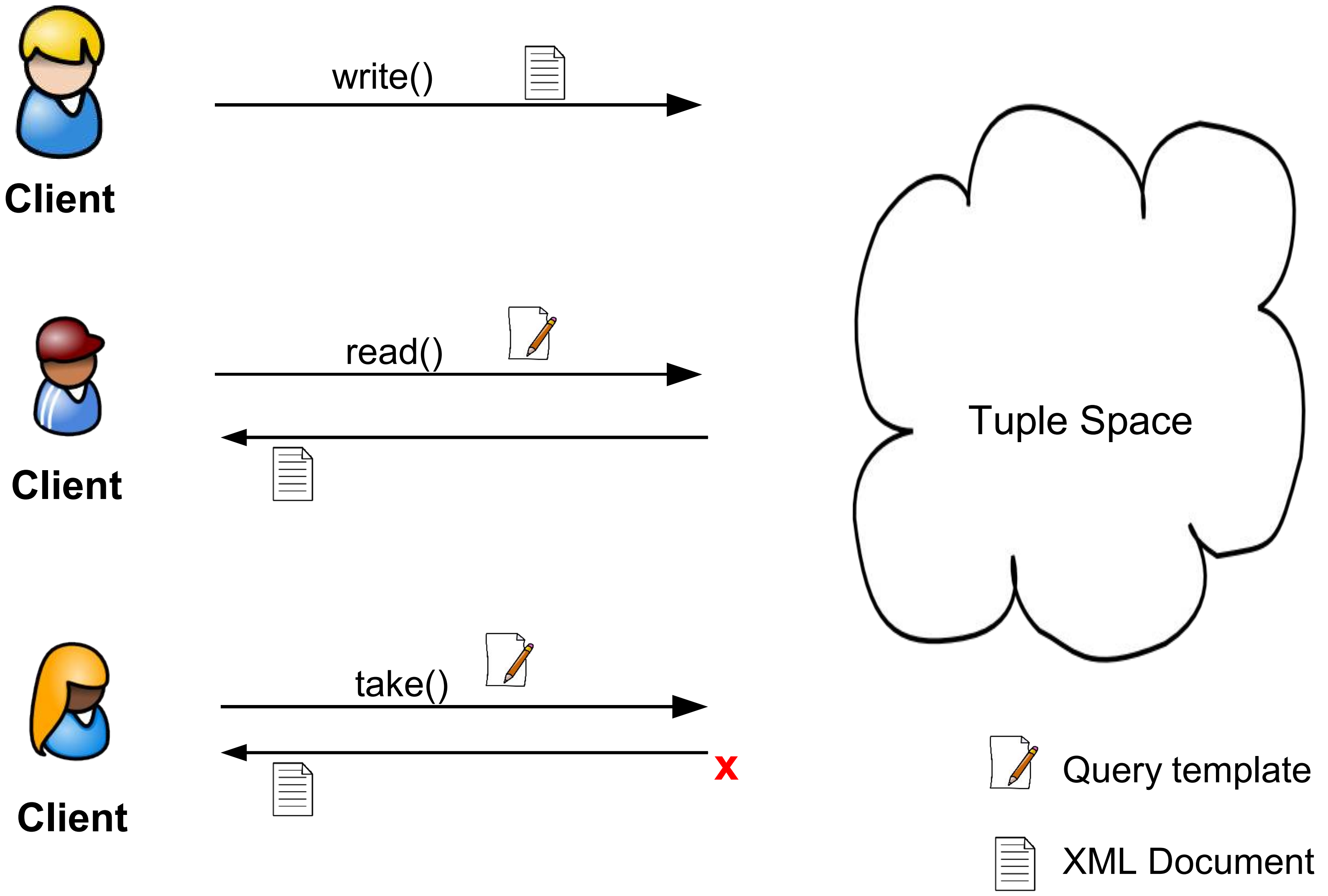
High throughput computing – Problems of conventional architectures

- Challenges
  - Scalable resource discovery, Efficient resource usage
- the „information gap“: a lot of state information must be available to the information systems to allow efficient resource usage
- Scalability: information systems and schedulers (usually) become bottlenecks as execution nodes are added

High-throughput computing – UNICORE 6 based approaches

- Commandline client (UCC) batch mode
- UNICORE 6 Workflow system
- **Tuple Space based approach**
- Other(s)

Tuple Space basics



Tuple Spaces principles

- Tuple space stores entries
- Tuple space entries have a lifetime („lease time“)
- Basic API
  - write(Entry, LeaseTime)
    - inserts new entry into the tuple space
  - read(Template, Timeout)
    - returns matching entry
  - take(Template, Timeout)
    - returns matching entry
      and removes it from the tuple space
  - notify(Template)
    - tuple space will notify client upon insertion of matching entry

Template-based queries

- read(), take() use „query by example“
- Supply a template for querying with fields set
- Example:
  - give me an entry where the field „status“ has the value „DONE“
- Intuitive and easy to use
- Not as powerful as a real query language (such as SQL, XPath or XQuery)

JavaSpaces

- Java based tuple space (stores Java objects)
- Part of Sun's JINI specification
- Opensource and commercial implementations exist
  - Gigaspaces (commercial)
  - Sun JINI
  - Blitz
- Stores Java objects
- Communicates using Java RMI (but also SOAP etc.)

Tuple spaces: pro&con

- Pro
  - Decouple communications
  - Enables highly scalable („share-nothing“) architectures
- Con
  - Tuple space itself is **hard** to distribute
  - Tuple space itself may become the bottleneck
  - Not all applications fit this model

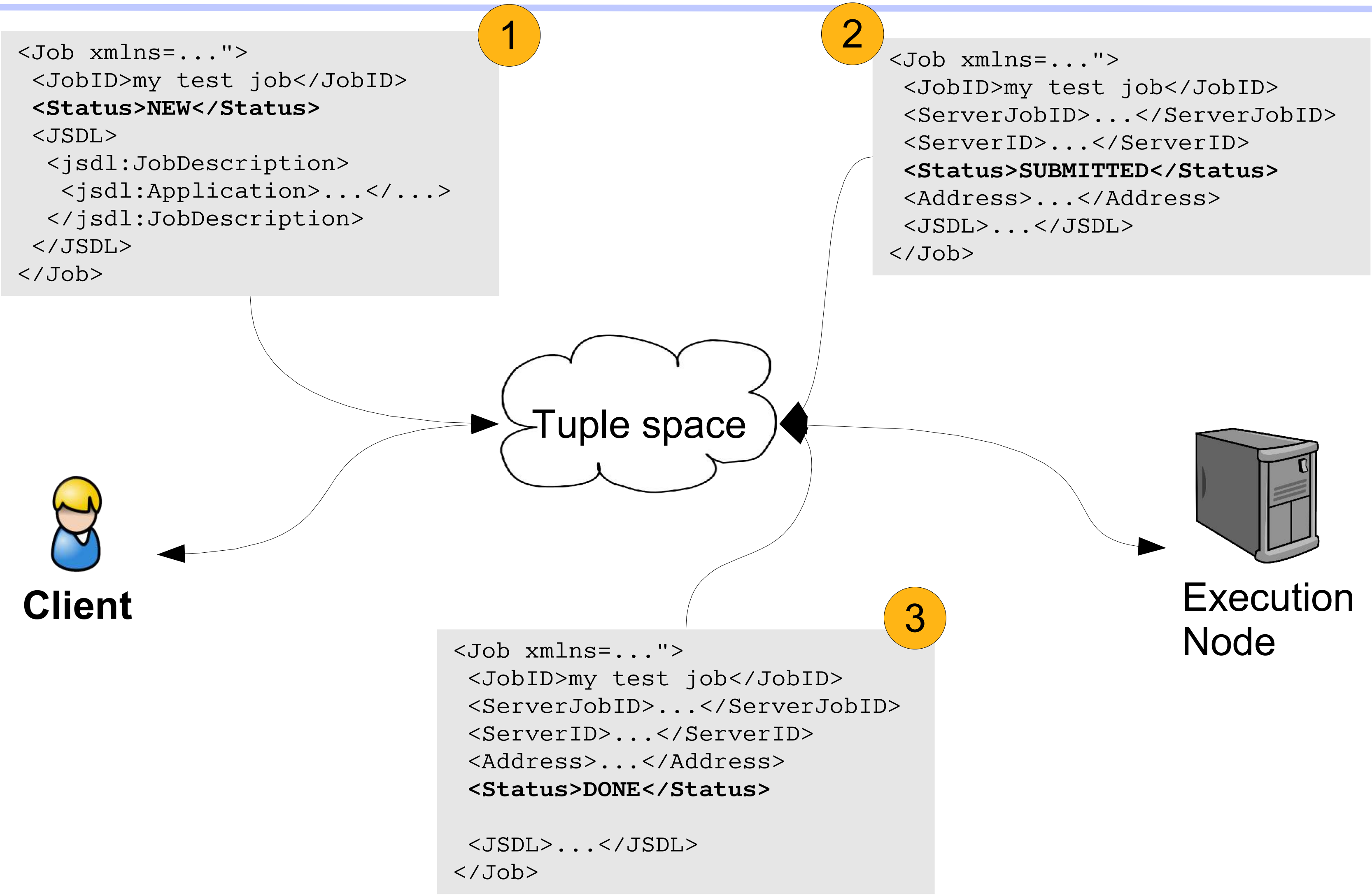
Idea: „XML Space“

- Store any XML documents
- Use UNICORE 6 protocols and tools
- WSRF fits the tuple space model very nicely
  - resource + lifetime concepts
  - XML centric
- Diploma thesis by Miriam Schumacher
  - used UNICORE 6 / WSRFlite to implement such an „XML space“
  - Prototype + example application available
    http://unicore.svn.sf.net/svnroot/unicore/contributions/unicore-spaces/trunk

UNICORE Spaces

- Small add-on to UNICORE 6
- Two services
  - Space
    - web service, offering write(), read(), take()
  - SpaceEntry
    - WSRF service
    - Each instance corresponds to one entry in the spaceWS-
    - XML document is stored as a WSRFresource property
- Example Client (SpaceClient)
- ca. 300 lines of code

Job execution example



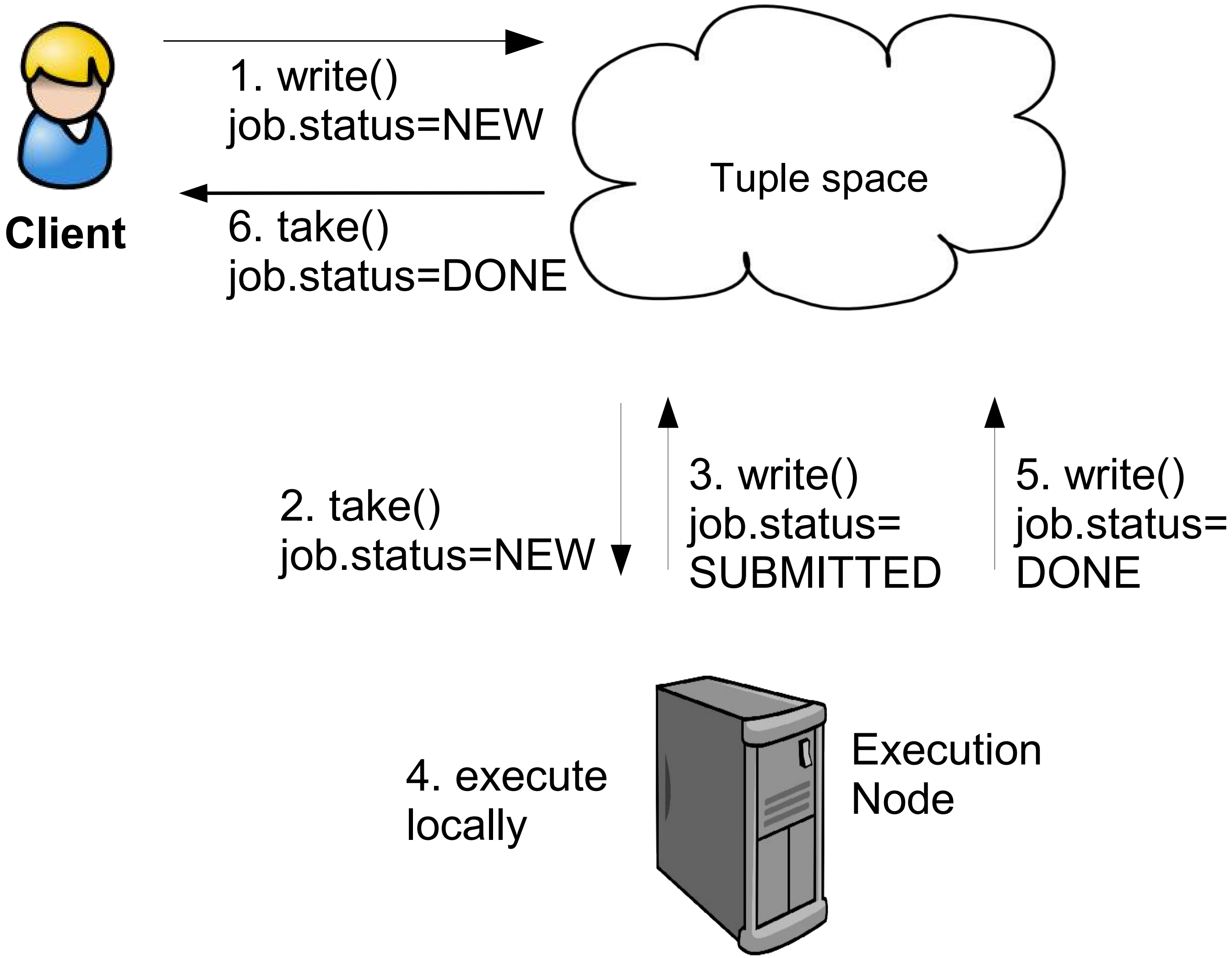
```
<Job xmlns=...">
 <JobID>my test job</JobID>
 <Status>NEW</Status>
 <JSDL>
  <jsdl:JobDescription>
   <jsdl:Application>...</...>
  </jsdl:JobDescription>
 </JSDL>
</Job>
```

```
<Job xmlns=...">
 <JobID>my test job</JobID>
 <ServerJobID>...</ServerJobID>
 <ServerID>...</ServerID>
 <Status>SUBMITTED</Status>
 <Address>...</Address>
 <JSDL>...</JSDL>
</Job>
```

```
<Job xmlns=...">
 <JobID>my test job</JobID>
 <ServerJobID>...</ServerJobID>
 <ServerID>...</ServerID>
 <Address>...</Address>
 <Status>DONE</Status>

 <JSDL>...</JSDL>
</Job>
```

Job execution



Client

1. write()
job.status=NEW

6. take()
job.status=DONE

Tuple space

2. take()
job.status=NEW

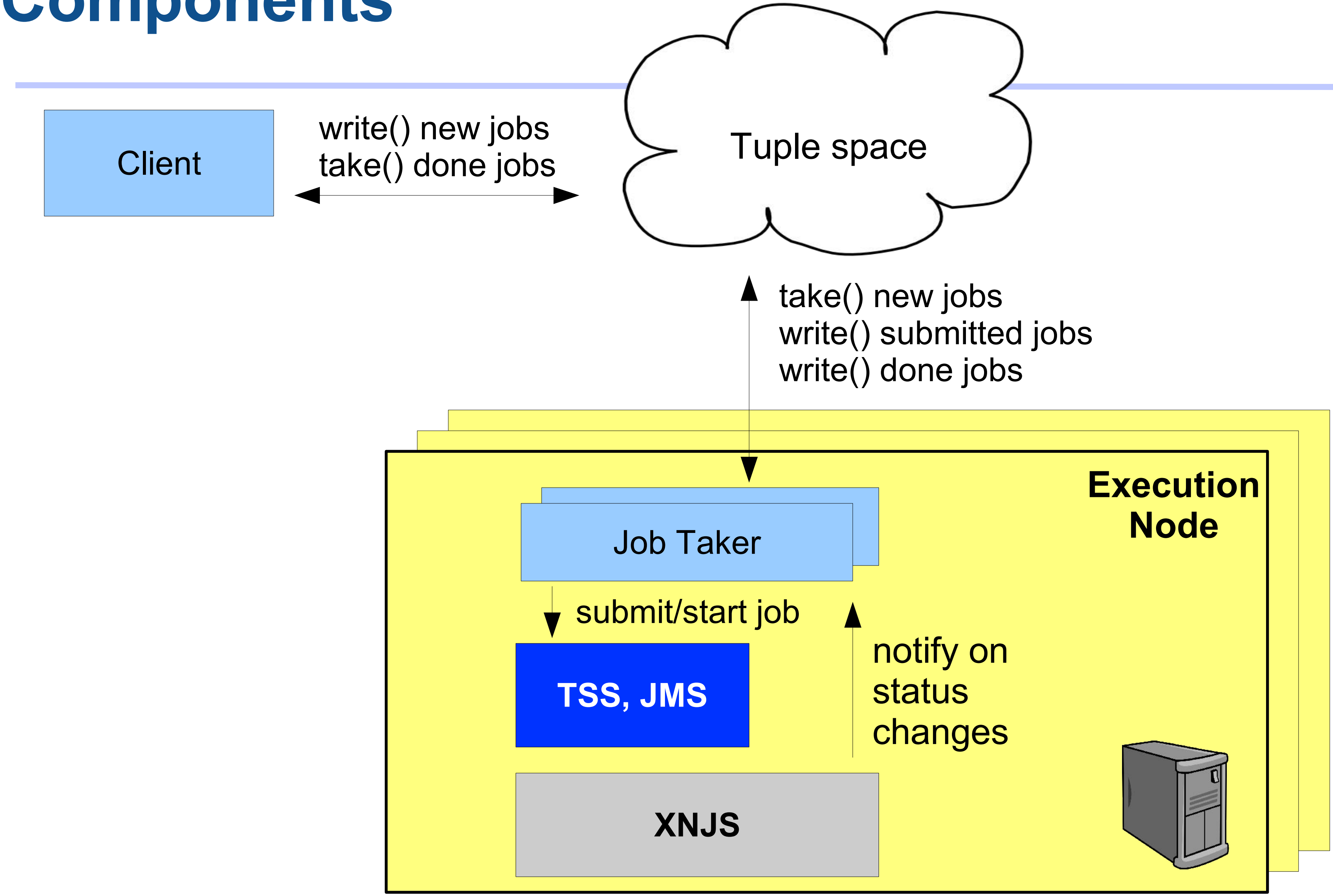
3. write()
job.status=
SUBMITTED

5. write()
job.status=
DONE

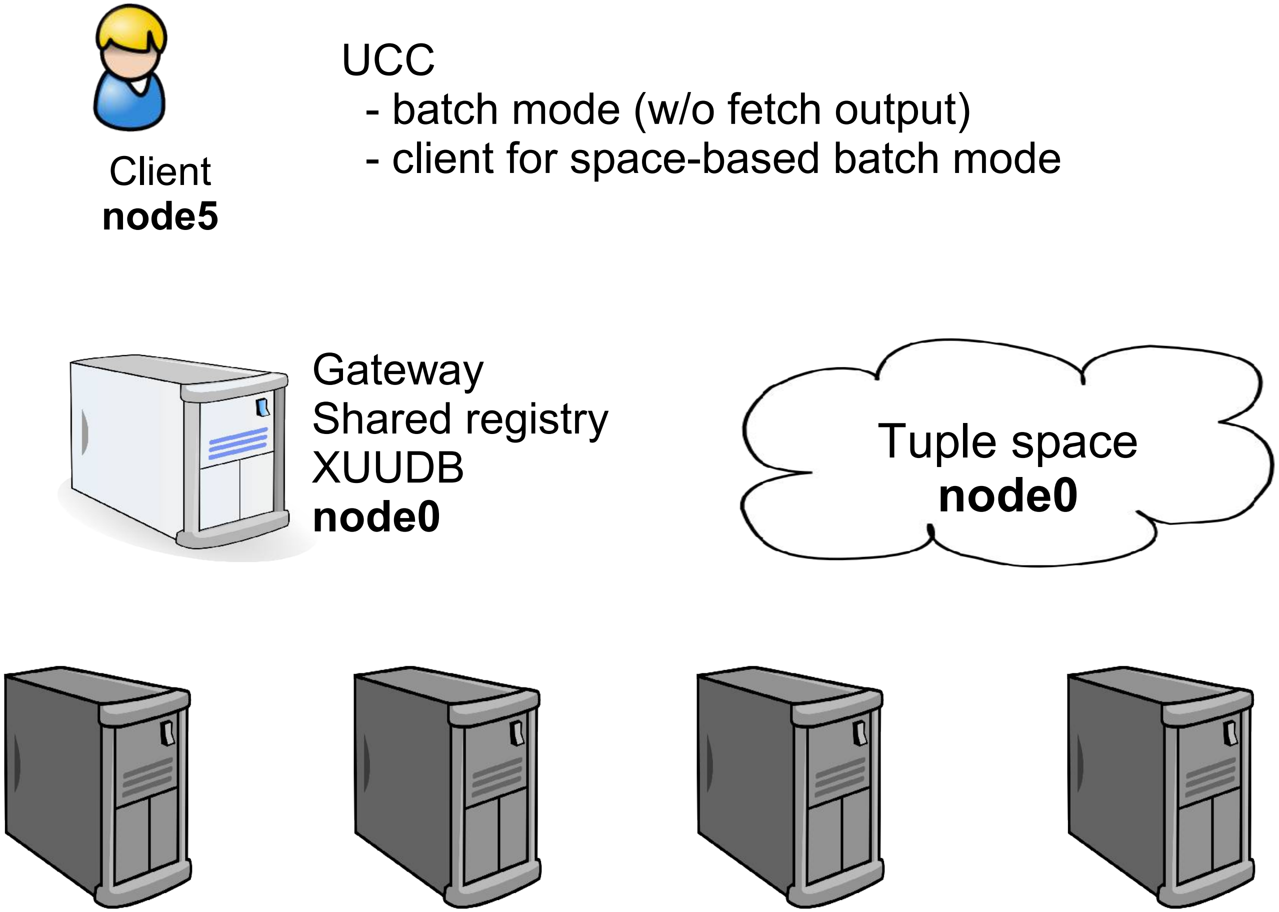
4. execute
locally

Execution
Node

Components


Client
write() new jobs
take() done jobs
Tuple space
take() new jobs
write() submitted jobs
write() done jobs
Execution Node
Job Taker
submit/start job
notify on status changes
TSS, JMS
XNJS

Performance test configuration



Client
node5

UCC
- batch mode (w/o fetch output)
- client for space-based batch mode

Gateway
Shared registry
XUUDB
node0

Tuple space
node0

Execution
node1

Execution
node2

Execution
node3

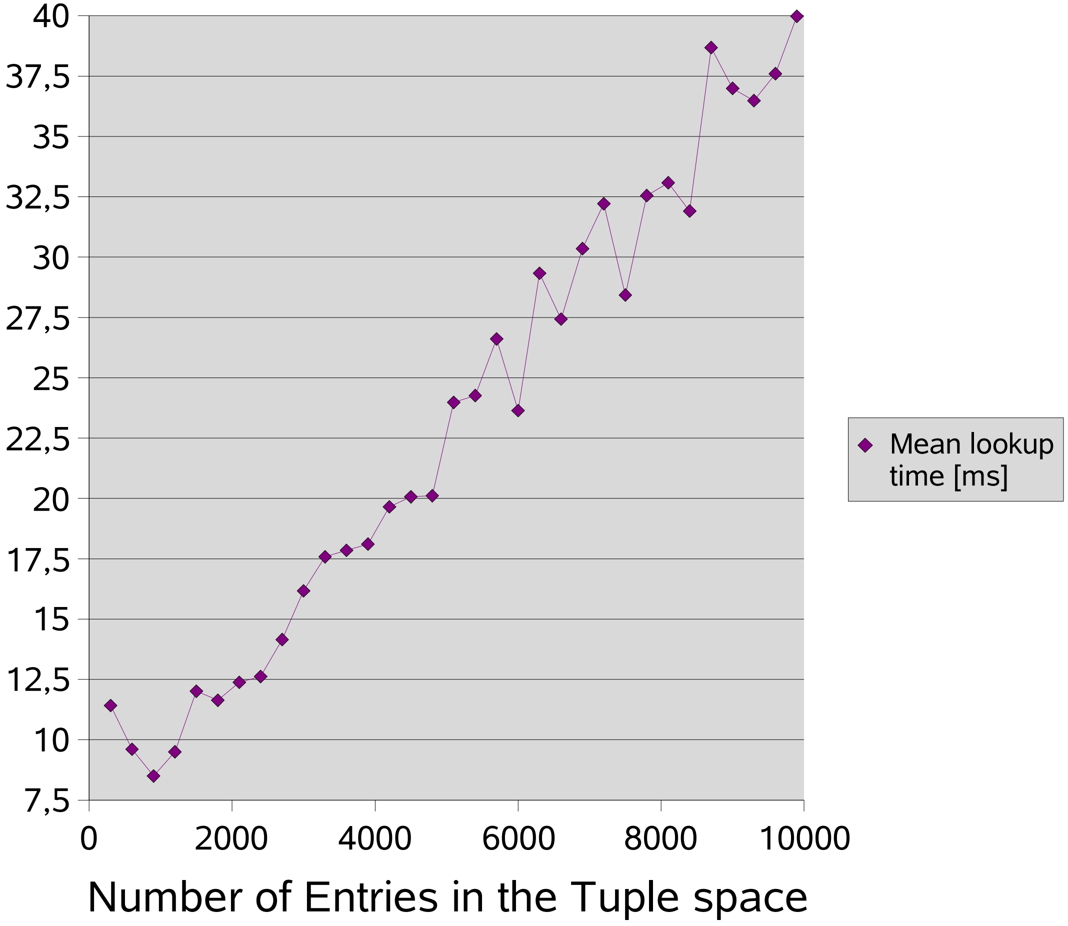
Execution
node4

Some first results: job throughput

Nodes	Jobs			
	100	**400**	**1000**	**5000**
1	97			
2	48	146	520	
4	26	80	231	940
5	22			

for **comparison**: **UCC batch-mode, 100 jobs @ 4 nodes = 126 seconds**
(ucc „tuned“ to not check resource availability and to not get any output files)

read() / take() : performance



Unit test!

Measure mean
time for read() of
100 random entries

read()/take()
becomes a **bottleneck**
when many clients
access the same space

Summary:

- Very promising!
- Pro
  - Excellent for simple requirements
  - Highly scalable
  - Very simple to setup
- Con
  - Difficult for complex requirements (e.g. co-scheduling)
  - The Tuple space might become the bottleneck eventually!

Outlook

- Other use cases for the UNICORE Spaces?
  - any „document-oriented state machine“ will be easy
- Implementation aspects
  - improve read() / take() performance (partitioning, indexing...)
  - investigate distributed/clustered space (hard!)
- Job processing example application:
  - **more than a toy**
  - Security
    - need to delegate trust to the workers
  - Input/Output data
    - stage-in from shared storage? From client?
    - getting results: not a problem once TD is in place

Thank you!

Chemomentum

Sixth Framework Programme

Project website: **http://www.chemomentum.org**

Funded by the European commission, IST-5-033437

UNICORE

Downloads, documentation, tutorials, mailing lists, community links, and more: **http://www.unicore.eu**