Last-minute slide deck for QCon London 2008 (Fill-in for a sick speaker)

by

John Davies

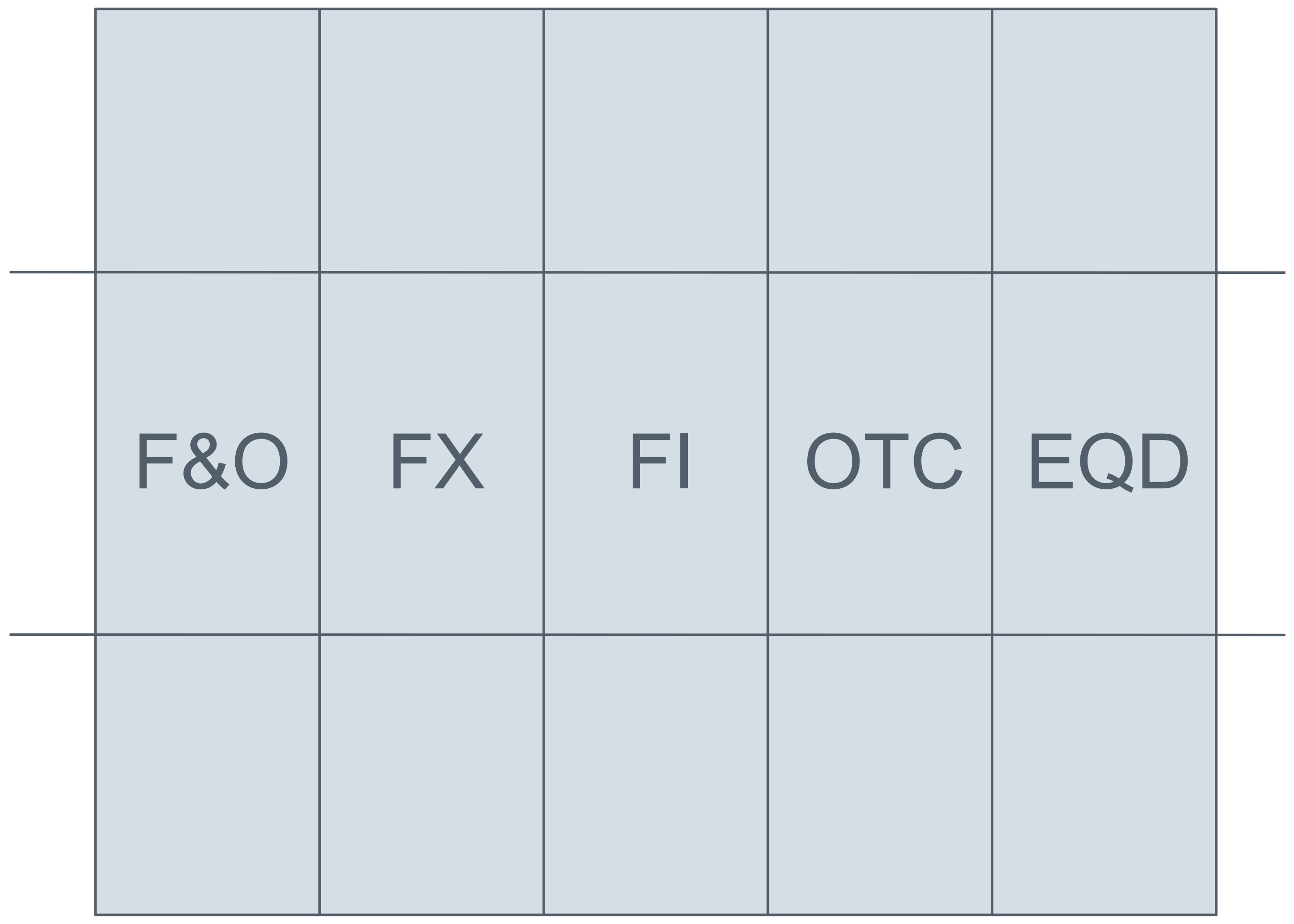
Agenda

- Overview of the Investment Banking Technology Space
- The basics of Integration
- The Issues with persisting today’s complex data models
- The argument for distributed architectures
- Example - High performance matching engine
- Example - SWIFT MT to MX migration

Enterprise!

- **If we're looking for an enterprise solution then it has to solve problems in the front, middle and back office across all lines of business (LoBs)**

- **This structure creates small businesses within the business**
  - Not unlike a large department store selling fruit & veg., white goods, books and music



A reminder of the problem…

- **Front Office**
  - Very high volume (100-100,000 / sec), usually simple messages
  - Latency is critical (< 10ms)
  - FIX, FAST, ASN1, IIOP are most common payloads
  - Light-weight XML only (if any)

- **Middle Office**
  - Volumes are high (1-1000 / sec), very complex messages
  - Calculations are complex and grid/HPC is usually requires
  - FpML, ISO-20022, Murex, SwapsWire, CSVs are most common formats
  - XML widely used but usually over MQ & JMS

- **Back Office**
  - Low volume (100-1000 / hour)
  - Very high value messages, strict compliance and validation
  - Proprietary networks, mostly SWIFT

Back to basics

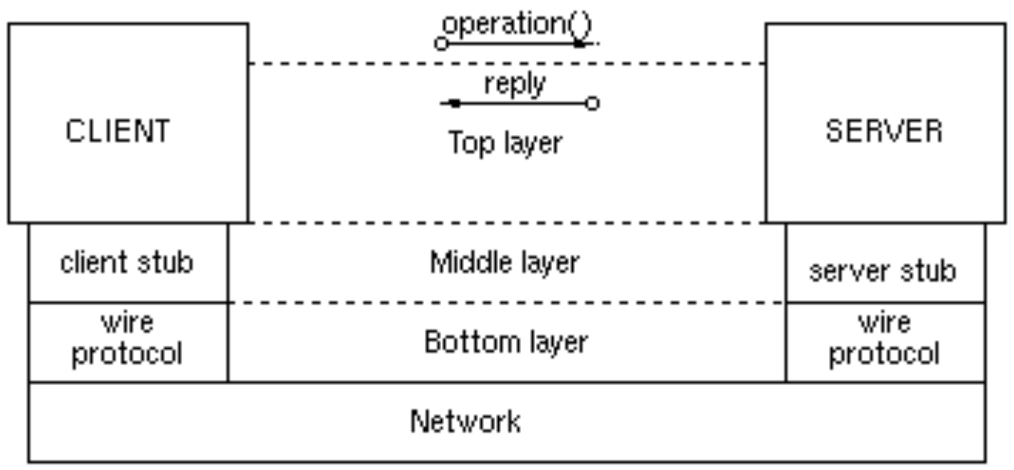
- **All computers have inputs and outputs**
- **The vast majority of input and outputs are non-human interfaces**
  - Network, Sockets, Messages (JMS etc.), Databases etc.
  - The human version being keyboards, screens, printers and faxes
- **Attaching the output of one system to the input of another only works if they “speak” the same protocol over the same transport**
- **Matching these two (transport and protocol) is called “Integration”**
- **There are two levels of integration**
  - Transport integration
  - Data integration

Basic Integration

- **There are two parts to integration**
  - Transport integration
  - Data integration

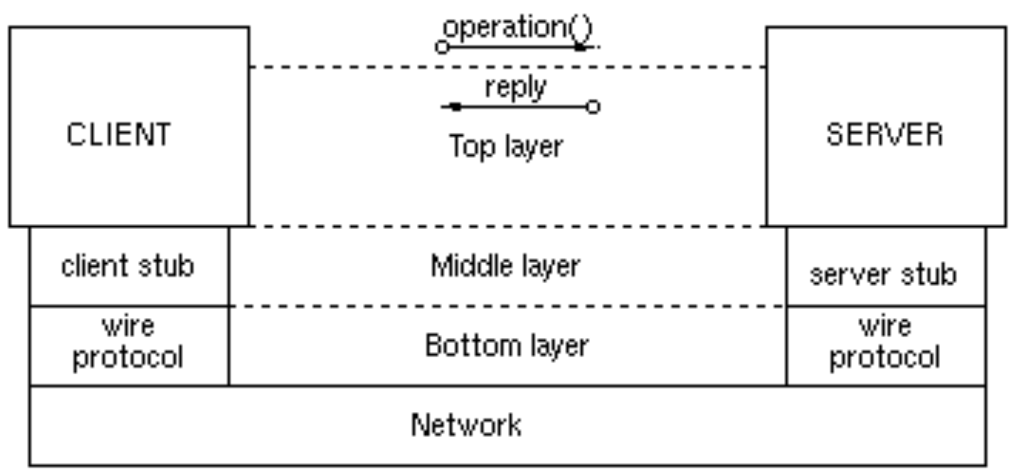
- **Thirty years ago we were using RS-232 (serial), then came token-ring and finally Ethernet**
  - Today we see InfiniBand, RDMA etc. becoming mainstream

- **Going back to basics, one of the early standards for programmers was CORBA (Common Object Request Broker Architecture)**
  - A well defined protocol over a well defined transport, fast too



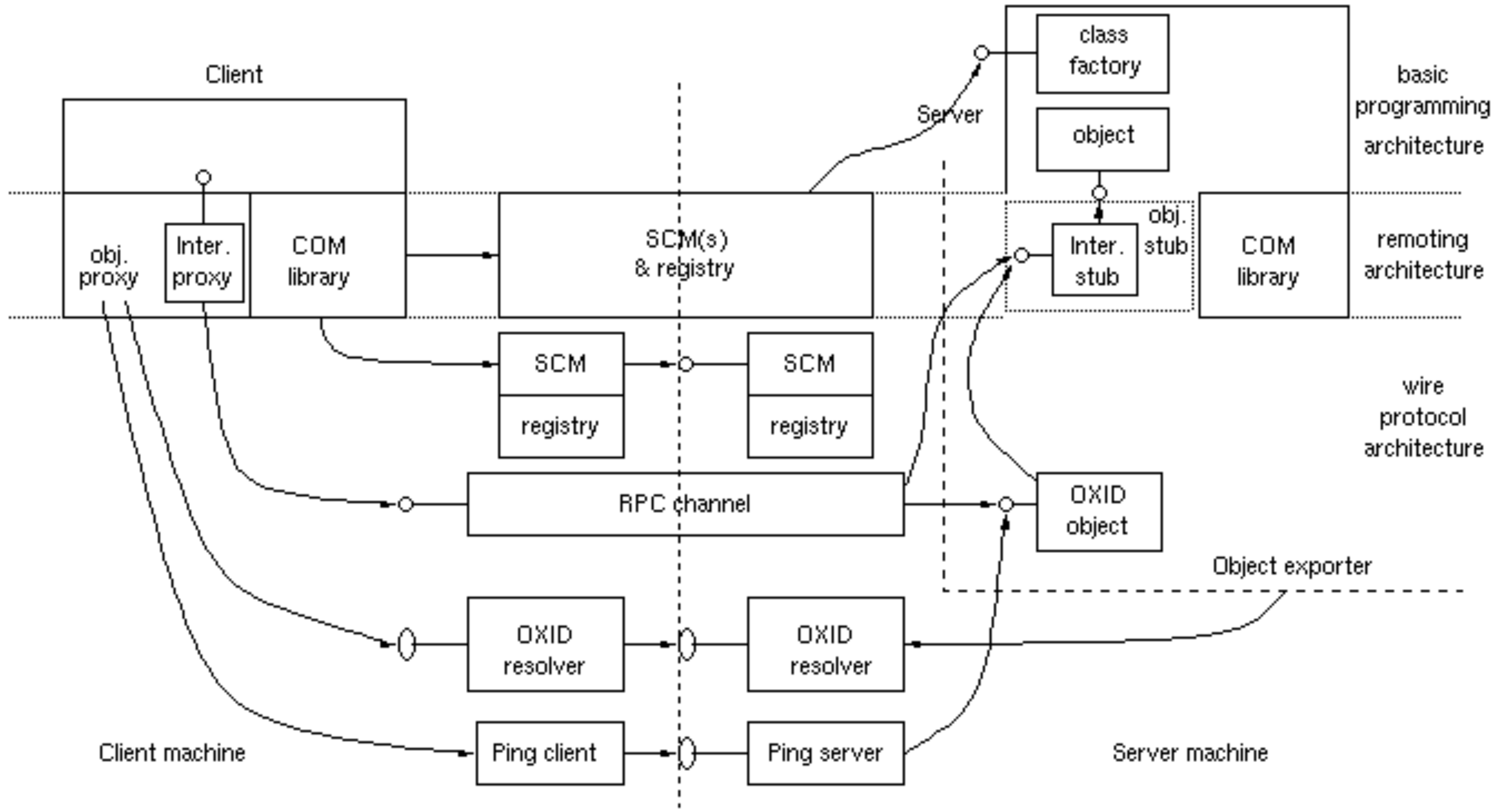
CORBA worked well

- **So Microsoft replaced it with something far simpler… DCOM**



CORBA worked well

- **So Microsoft replaced it with something far simpler… DCOM**



CORBA worked well

- **So Microsoft replaced it with something far simpler… DCOM**

CORBA was too tightly coupled

- **It was the synchronous coupling that let CORBA down**
  - Some users however found the tight coupling to be an advantage
    - If the network died you knew immediately
    - It was fast and guaranteed delivery

- **The standards were improved but too many nails were already in the coffin**
  - Microsoft had a “better” idea and Java was too young at the time

- **As time went on CORBA was often replaced with SOAP or XML over HTTP and MQ**
  - Not all bad but we took a huge step back in performance
  - CORBA will still the backbone of most telcos and many banks
  - It is very debatable as to what Web Services has given us over CORBA

XML - good or bad?

- **Imagine a comma delimited file (CSV), a header and many rows of data**
  - Take the header and repeat it for every row - pointless?
  - Now replace the commas with “>” and “<“ and space it out for clarity
    - Clarity for who?

- **CSVs are very limited in what they can represent, typically we need to represent a hierarchy of related fields**
  - CSVs are just “string” fields, they are not type-safe and create integration problems

- **In the investment banking world derivative trades can be extremely complex and XML is a perfect tool for representing them**
  - Not only are CSVs out of the question but databases are also of questionable use for this level of complexity

- **XML is good for complex messages**
  - It’s not ideal for simple message particularly in low-latency or high volume situations

Non-XML formats

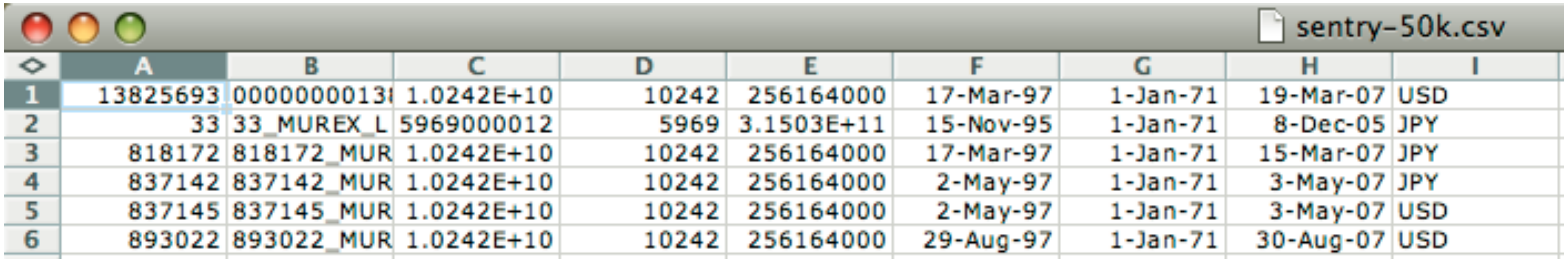
- **A large proportion of applications (internal and external) still send CSV files**
  - We need to be able to deal with these natively and not have to convert them to XML each time
  - Conversion adds risk, risk is money
  - Technically it's not always that easy, some of these files are millions of lines long

- **Banks sell their services to Brokers and Hedge Funds, amongst others, being able to "onboard" clients (mostly non-XML formats) is critical to their ability to sell their services**

- **Many international banking standards are non-XML and will remain so for some time to come**
  - The Federal Reserve Bank put through $½ trillion through our SWIFT systems every day
  - We can run the added risk of translating this to and from XML just for the convenience of the vendors

Complex validation

- **You can define an ISO-8601 DateTime in XML Schema, the format is well defined**
  - Well almost, there are still inconsistencies about time-zones and time offsets
- **The problem comes though when you want to restrict one field based on the content or existence of another field(s)**
  - If //@AlternateEmail then at least two emails must be defined
  - //TradeDate must be before or the same as the //SettlementDate
- **This problem isn't unique to XML, it is true for almost any type of data**
- **Excel imports CSVs reasonably well but it can't semantically validate the content**


sentry-50k.csv

	A	B	C	D	E	F	G	H	I
1	13825693	0000000013	1.0242E+10	10242	256164000	17-Mar-97	1-Jan-71	19-Mar-07	USD
2	33	33_MUREX_L	5969000012	5969	3.1503E+11	15-Nov-95	1-Jan-71	8-Dec-05	JPY
3	818172	818172_MUR	1.0242E+10	10242	256164000	17-Mar-97	1-Jan-71	15-Mar-07	JPY
4	837142	837142_MUR	1.0242E+10	10242	256164000	2-May-97	1-Jan-71	3-May-07	JPY
5	837145	837145_MUR	1.0242E+10	10242	256164000	2-May-97	1-Jan-71	3-May-07	USD
6	893022	893022_MUR	1.0242E+10	10242	256164000	29-Aug-97	1-Jan-71	30-Aug-07	USD

Persistence

- **How do you store something as complex as FpML or SEPA's ISO-20022 messages in a relational database?**
  - FpML is typical, it has over 4000 elements and umpteen levels of depth
  - Normalising FpML would result in the mother of all databases and SQL queries up to a page long
- **How do you manage multiple versions?**

- **Answer**
  - Don't use a relational mapping
  - Simply store it as XML (in a CLOB) and extract the indices you need with XPath
  - Many databases (Oracle, Sybase, DB2 V9 etc.) offer XML data types but they usually don't implement all the schema features and slow the insert times down to a crawl
  - Using this technique leaves many more powerful mechanisms open such as using Tangosol's (now Oracle's) Coherence rather than a database

The end of the Database?

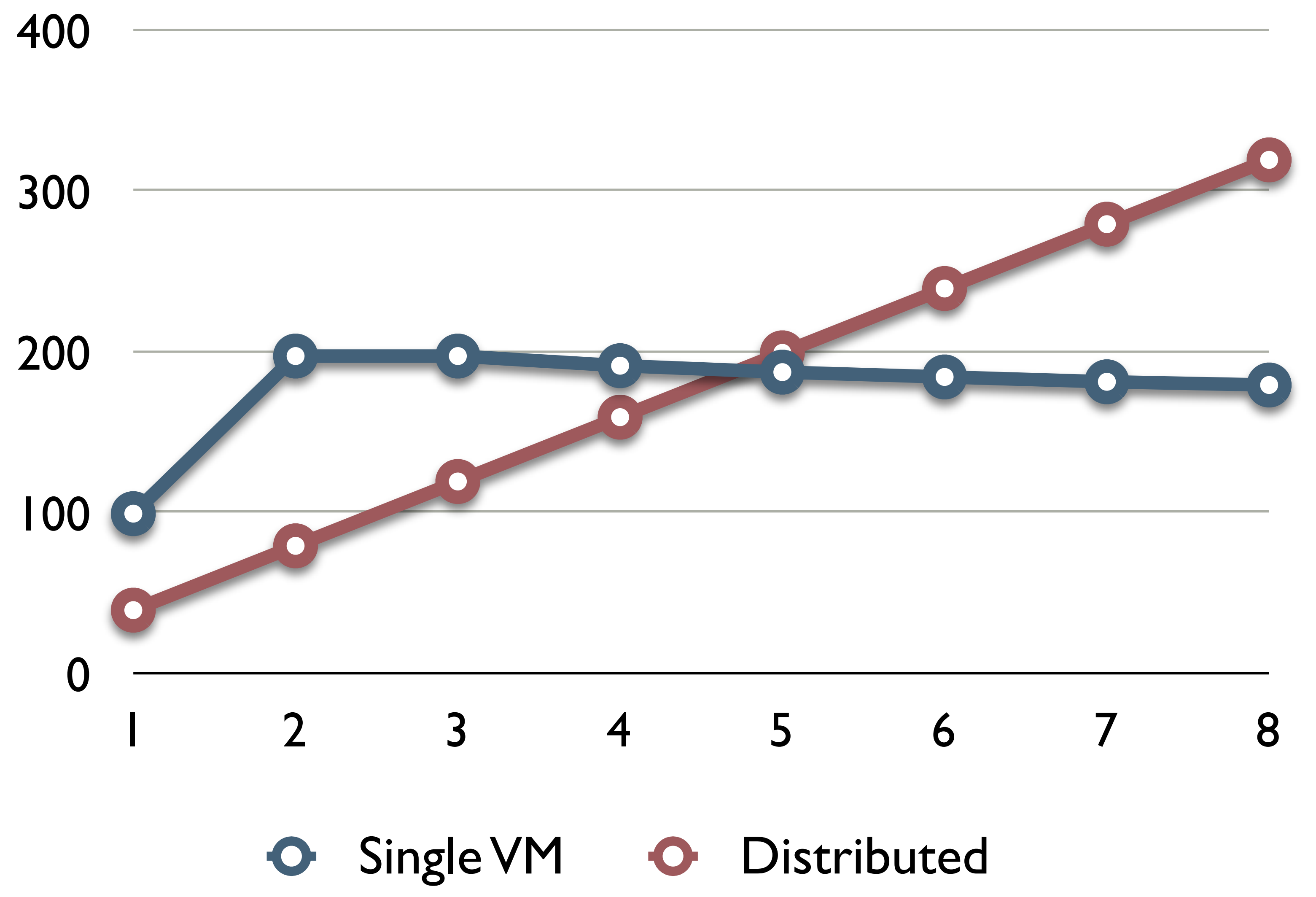
- **Databases are commodity**
- **We can now use memory (local and distributed) in places where we used to use a database**
- **We can now use disk where we used to use tape**
  - Throw your tapes away
- **In that past everything revolved around the database**
- **Now everything revolves around messaging (and ESBs)**
- **Relational database will out-live us but their position in the centre of the enterprise is long gone**

We can scale through distribution

- 100 Problems to solve?
  - Send them to 100 machines!
- Serialise every object, send it over the network and then de-serialise it again on the other side
  - Good, we can scale but what about all that object serialisation?
- What about queries over distributed data
  - Good if we can distribute the query but a real pain of the query doesn't match the distribution slice
- Dynamic load balancing can also be a pain
- When we move to a distributed system we take a hit with the serialisation

Distributed vs. Single VM

- Throughput on a single VM on a 2-core box vs. the same application distributed (unintelligently) on 4 2-core boxes
  - x = number of threads (one thread uses roughly 100% of one core)
  - y = throughput scaled to 100 being 1 core fully used



Java vs. C/C++

- Most (80-85%) of banking applications are written in Java
  - The remaining languages include C & C++, Perl, C#, VB
- C and C++ still play a serious role in high performance applications
  - It's not that it's faster than Java, it's just more predictable - no GC
- Java's lack of "real time" is its biggest hindrance in HPC
  - BEA and Sun have "real time" version but they are still a long way off what we need
  - 5ms to run the GC is still too long, today we're struggling to get under 20ms

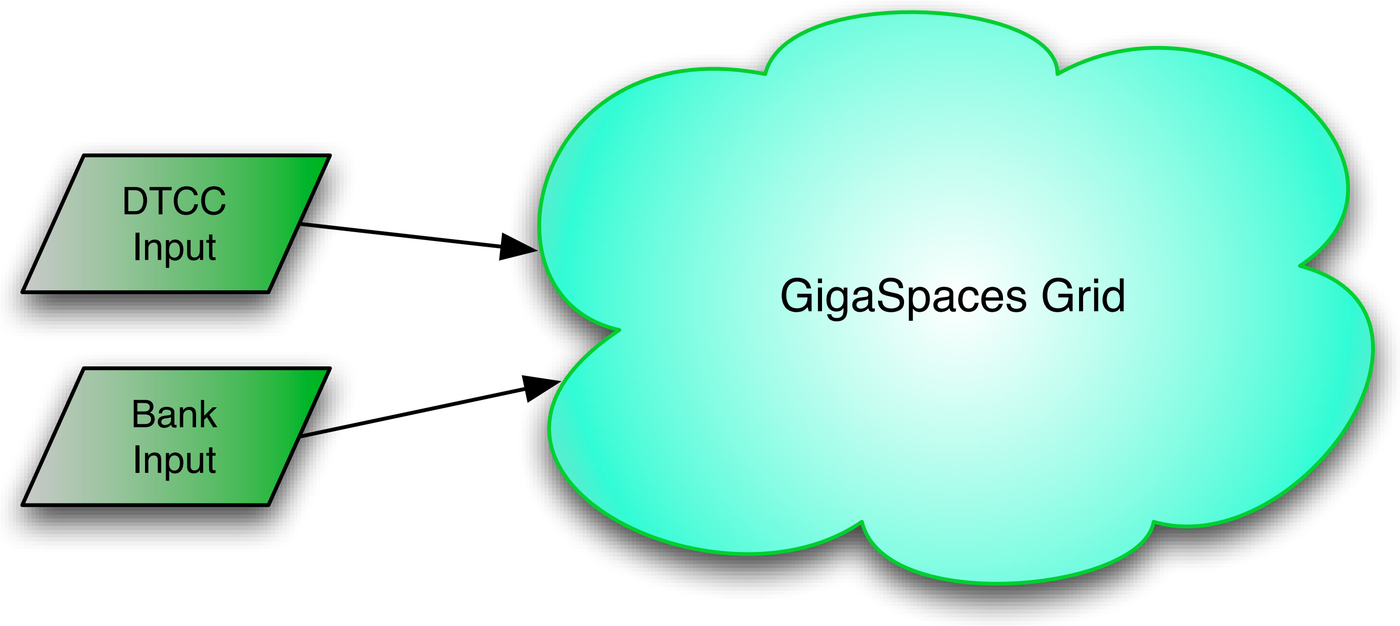
A Derivatives Matching Engine - Example

- Extremely complex derivatives, in FpML need matching as they come back from the DTCC
  - Using ORM (Hibernate, iBatis etc.) is way too complex
  - A database is too slow for transient data
- Put the trades into memory and match them using XPath expressions
  - 2 sets of 200,000 trades of roughly 15k each that's 6 GBytes minimum
  - Remember the GC problem and the serialisation delay?
- There's really only one obvious solution to remain in the market at this level

JavaSpaces

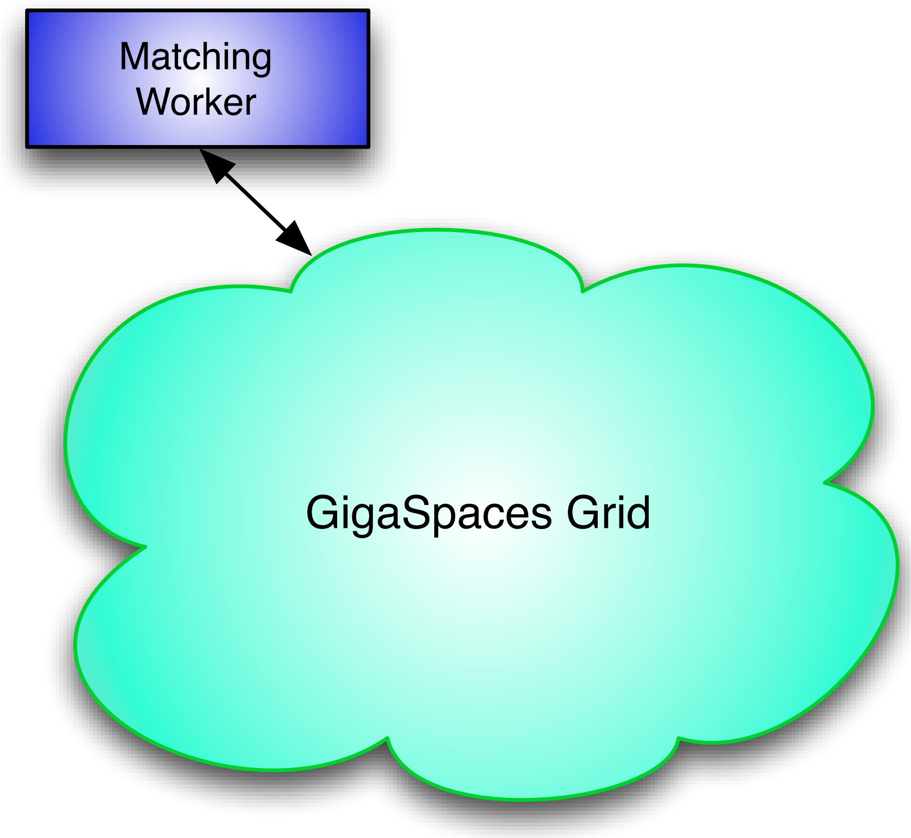
- While Azul can provide impressive acceleration to JEE application servers to me that’s like putting nitro-injection into a bus
  - It will go faster but you need a lot of power to get it going in the first place
- JavaSpaces provide a much better platform for this level of scalability
  - JavaSpaces is part of Jini (from Sun) and is in fact older than JEE
  - Blitz - Open source JavaSpaces implementation
  - GigaSpaces
- Provides the perfect implementation of Master-Worker pattern

Writing to the Space



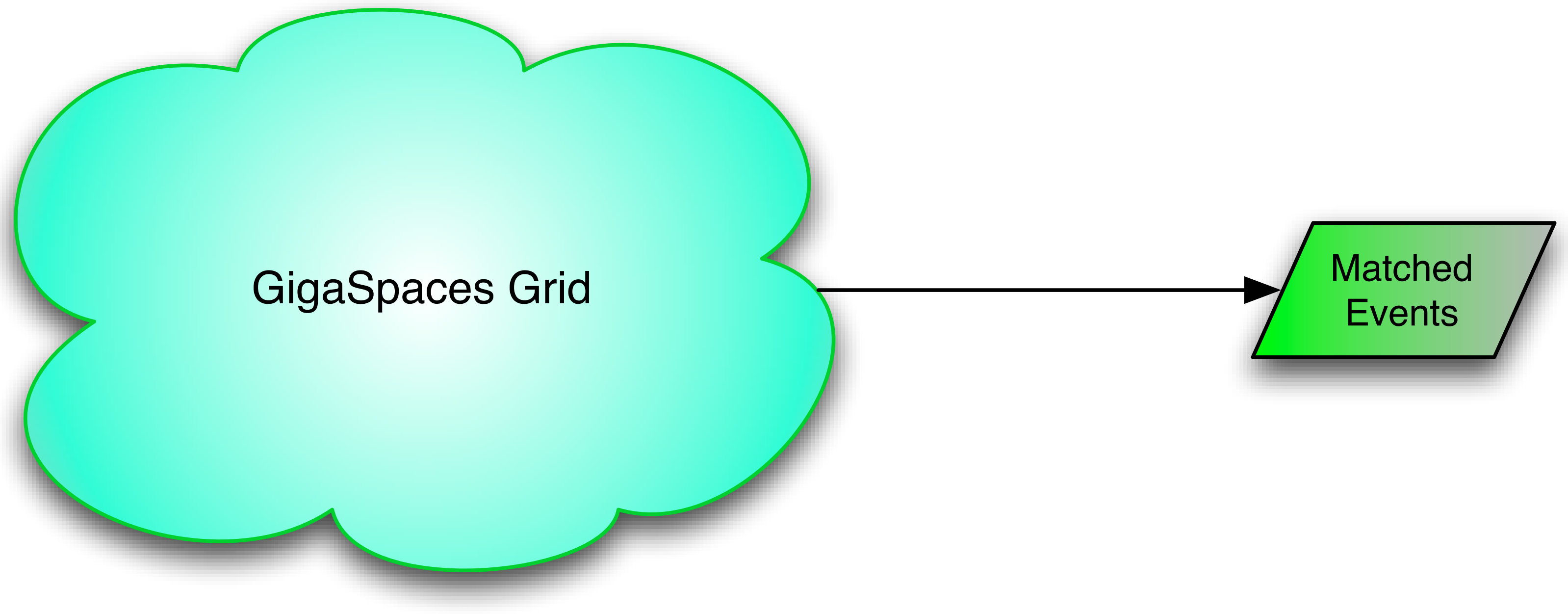
- Strings from of data are inserted into the space, a local (pass by reference) operation
- This requires relatively little complexity, just basic ESB functionality

Matching



- Now the clever bit
  - Strings are read out of the Space and converted to Objects using IONA's Artix Data Services (formally C24's Integration Objects) - This provides Java-Binding functionality beyond XML such as CSVs, SWIFT, FIX etc. with full XPath 2.0
  - The Object is assigned a matching template and the matching pair is sought
  - Once the match is found (now of later) an event is kicked off

Output



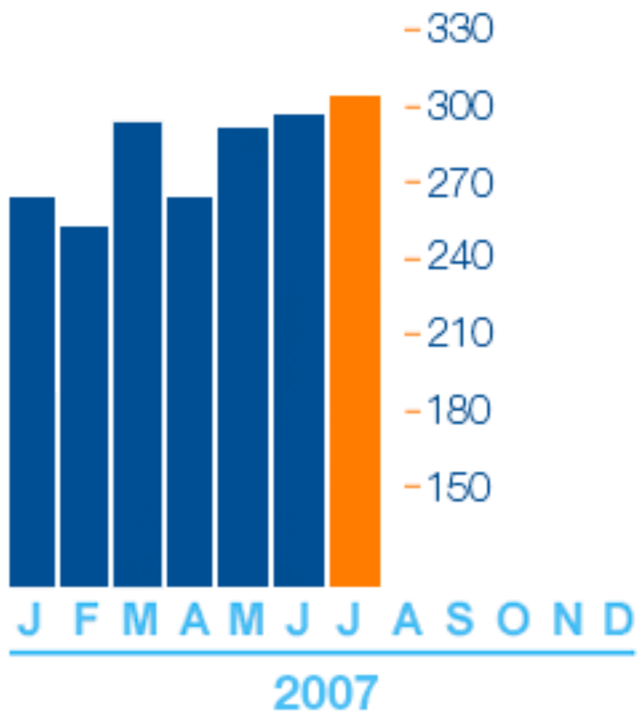
- This is relatively simple again
  - The output is usually driven by events from matched pairs
  - Results are usually written to a database but as XML not as an ugly relational entry
  - The Output(s) sinks can be running in the same VM and once again the passing of the data is by reference and not through serialisation (as in JMS or remote EJBs)

Another one of today's problems

- Messages are more and more complex
  - ISO-20022 and FpML
- Increasing volumes out strip Moore's law
  - Banks now ask for 100,000 per second on the trading floor
- Bad trading days can triple (3x) daily volumes
  - Bad days used to be 2x but recent stock market "blips" have produced 3x volumes
- New initiatives from regulatory bodies (i.e. European Commission) add extra requirements, many of them verging on the impossible
  - MiFID, SEPA

SWIFT

- SWIFT is 3 things, a secure network, a standards body and a connectivity provider
  - It is used by over 8,000 banks (>80,000 nodes), in over 200 countries handling over 15 million messages a day (>2 billion/year)
  - Mostly payments and securities, Europe is >65% of the volume
  - SWIFT is over 30 years old, has a systems availability of 99.986% (<1½ minutes/week) , they've NEVER lost a message



- The figures are impressive but the messages are a bastard!
  - Around 330 types of message
  - Over 400 complex types used in the messages
  - Over 1000 complex validation rules

The SWIFT message

```
{1:F01DRESGB2LAXXX0548034693}{2:I541DRESDEFFXXXXN2}{3:{108:MT541}}{4:
:16R:GENL
 :20C::SEME//FRTJ123REC2
 :23G:NEWM
 :16S:GENL
 :16R:TRADDET
 :98A::TRAD//20000519
 :98A::SETT//20000524
 :90A::DEAL//PRCT/101,001283
 :35B:ISIN GB0987654321
 :16S:TRADDET
 :16R:FIAC
 :36B::SETT//FAMT/4000000,
 :97A::SAFE//222S
 :16S:FIAC
 :16R:SETDET
 :22F::SETR//TRAD
 :16R:SETPRTY
 :95P::SELL//DEUTDEFF
 :16S:SETPRTY
 :16R:SETPRTY
 :95R::DEAG/CRST/456
 :16S:SETPRTY
 :16R:SETPRTY
 :95R::REAG/CRST/123
 :16S:SETPRTY
 :16R:SETPRTY
 :95P::RECU//DRESDEFF
 :16S:SETPRTY
 :16R:SETPRTY
 :95P::BUYR//MGTCDE55
 :97A::SAFE//111S
 :16S:SETPRTY
 :16R:SETPRTY
 :95P::PSET//CRSTGB22
 :16S:SETPRTY
 :16R:AMT
 :19A::SETT//GBP4047151,3
 :16S:AMT
 :16S:SETDET
 -}
```

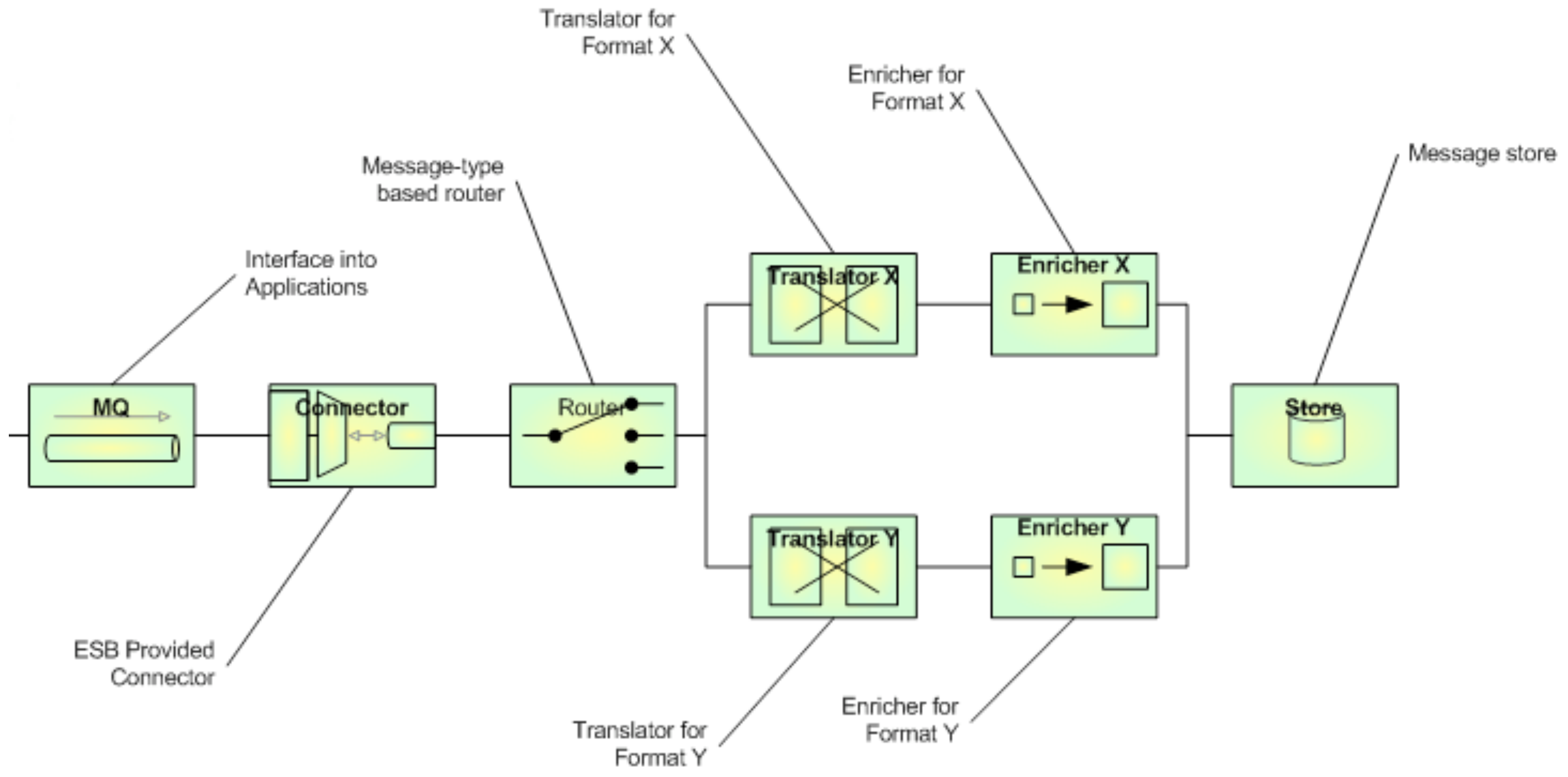
- This is one of the simpler messages
  - It's easy to parse into strings but there are complex rules about how the fields relate to one another
  - Constructing the message might also seem simple but get it wrong and SWIFT will fine you, big-time!

SWIFT MT to MX

- SWIFT are moving to XML, it’s based on ISO-20022
  - ISO-20022 is not a message definition but a framework for creating standards - it contains metadata
- Virtually every bank on the planet will have to migrate their systems from “old” MT format (previous slide) to new ISO-20022 messages
- To add to the pressure SEPA (Single European Payments Area) initiative uses parts of ISO-20022
  - This will happen sooner than SWIFT MX messages
- ISO-20022 is becoming a very important standard in the banking world

A large client...

- This is the logical architecture of a large European clearing house



- Most of the hard work is in transformation and enrichment

High Performance Transformation

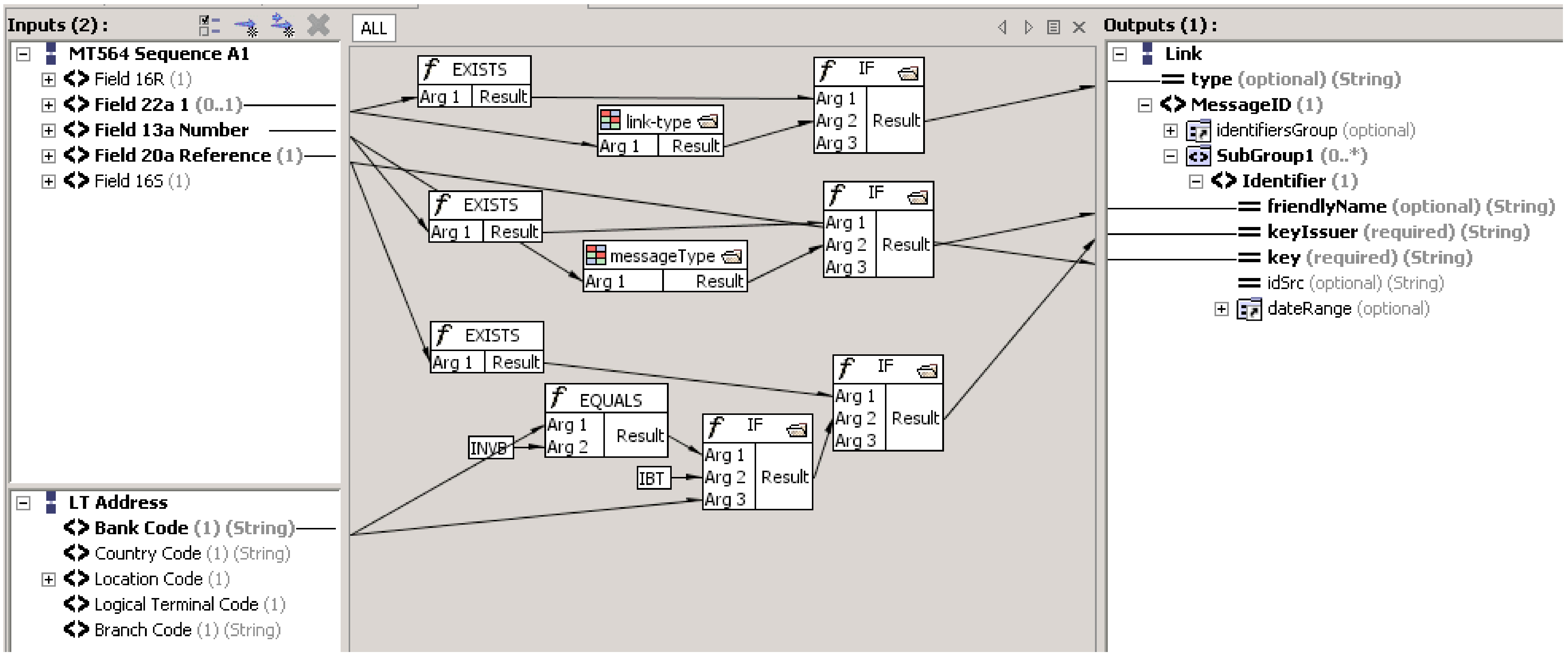
- Banks need a migration plan, SWIFT MT to SWIFT MX and visa-versa
- The task is complex and the volumes can be large
    - One message transformation alone has been estimated to take over 6 man-months coded by hand, there are over 300 messages in total
- Larger banks have around 2 million messages to transform a day
    - Windows can be short though, frequently batches of >100,000 arrive and need to be out within 15 minutes

How we do it

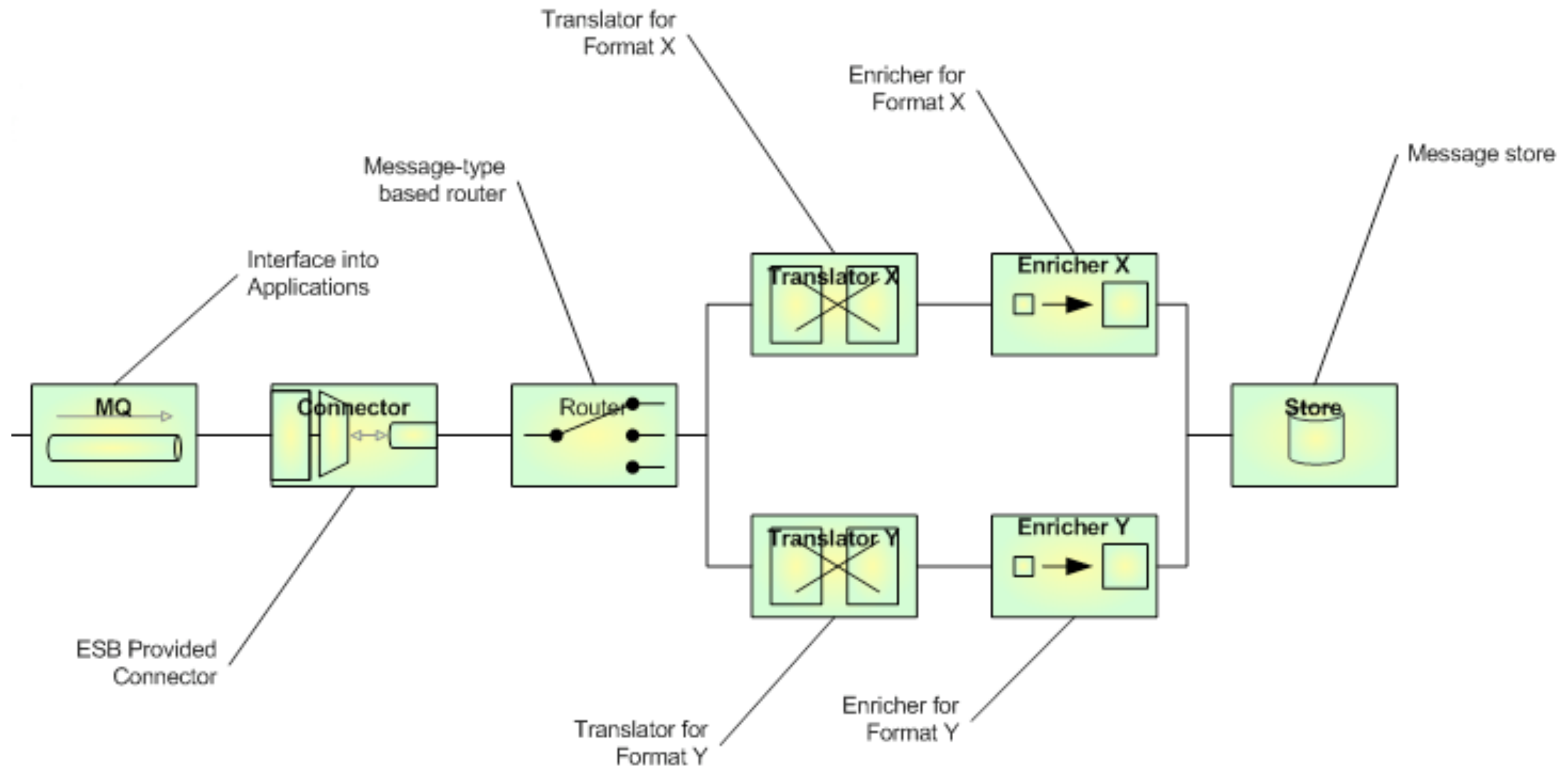
- We use Java-binding
- We model the SWIFT message as if it were XML
- We have a customised parser that reads SWIFT
  - It's not looking for the < and > but for other character indicators
- Like JAXB, JIBX or Castor we now have complex Java Beans that exactly represent the original SWIFT message
- Apply some clever transformation and some enrichment to this and the job is done
- Processed messages are stored in a database

Transformation

- Transformation needs a lot of business experience, the section below is a "function" within a larger transform



The flow...



1. Message arrives on MQ from the SWIFT gateway
2. Into a Mule connector
3. Messages are routed and potentially load-balanced
4. Messages are parsed, transformed, enriched in the grid
    - This is done using IONA's Artix Data Services (ADS)
5. Finally transformed messages are loaded into a database

Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability

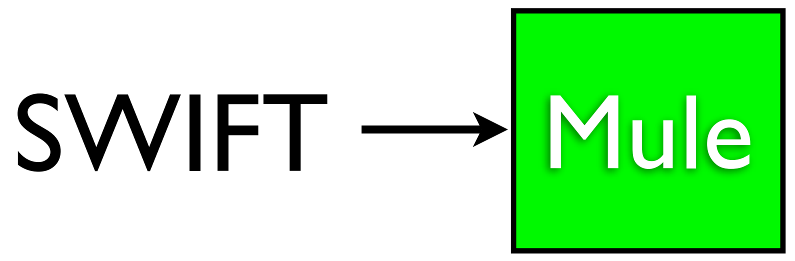
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability

SWIFT

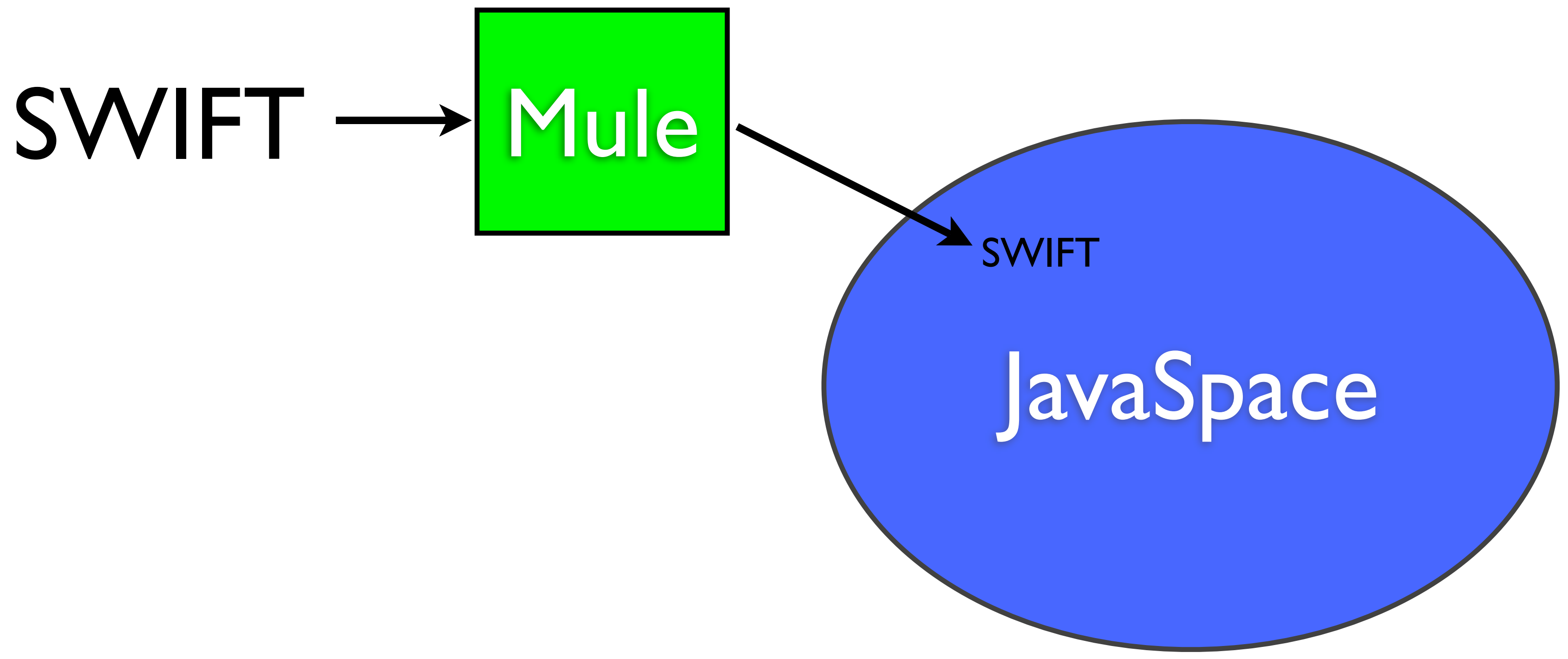
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability



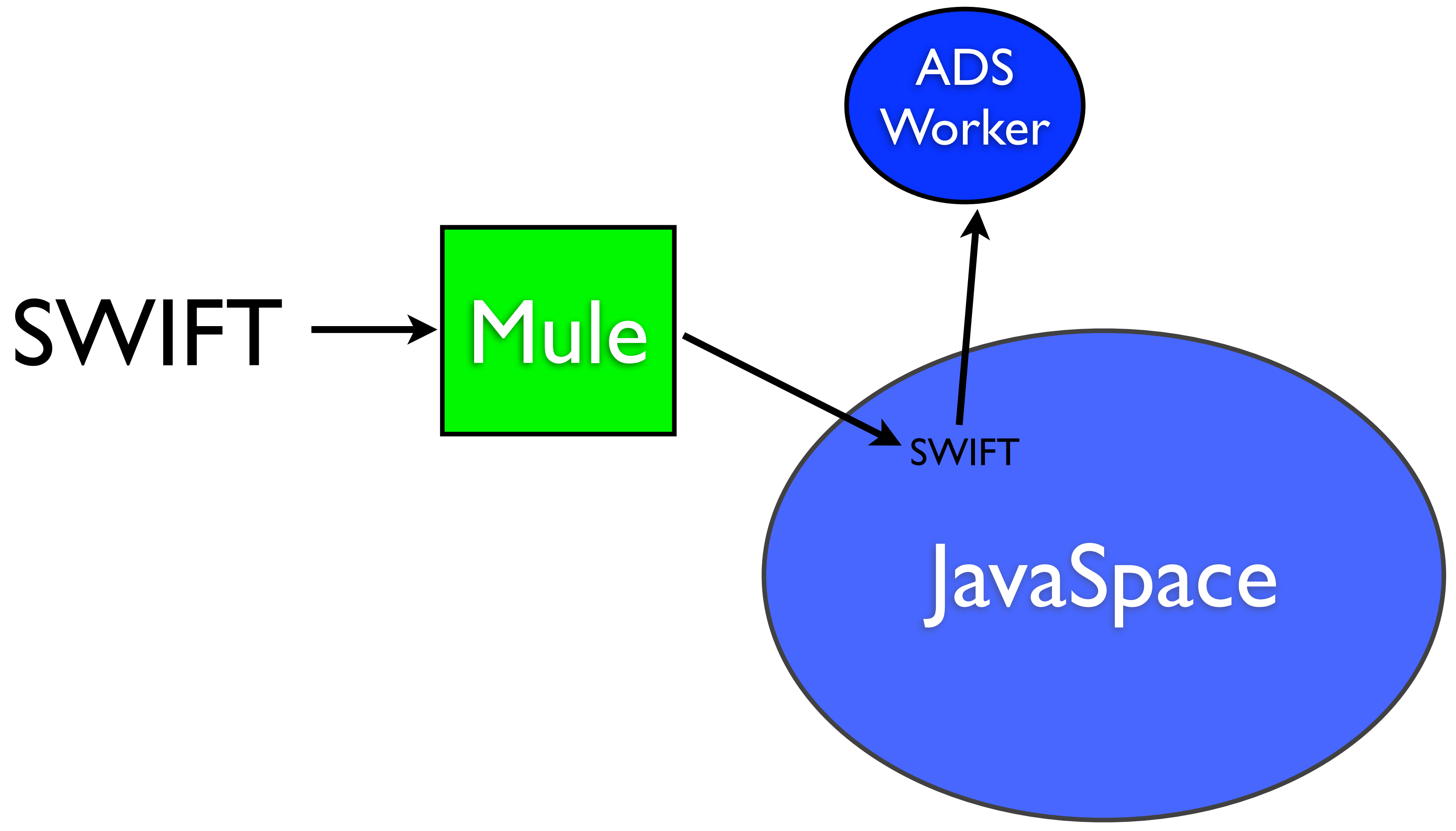
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability



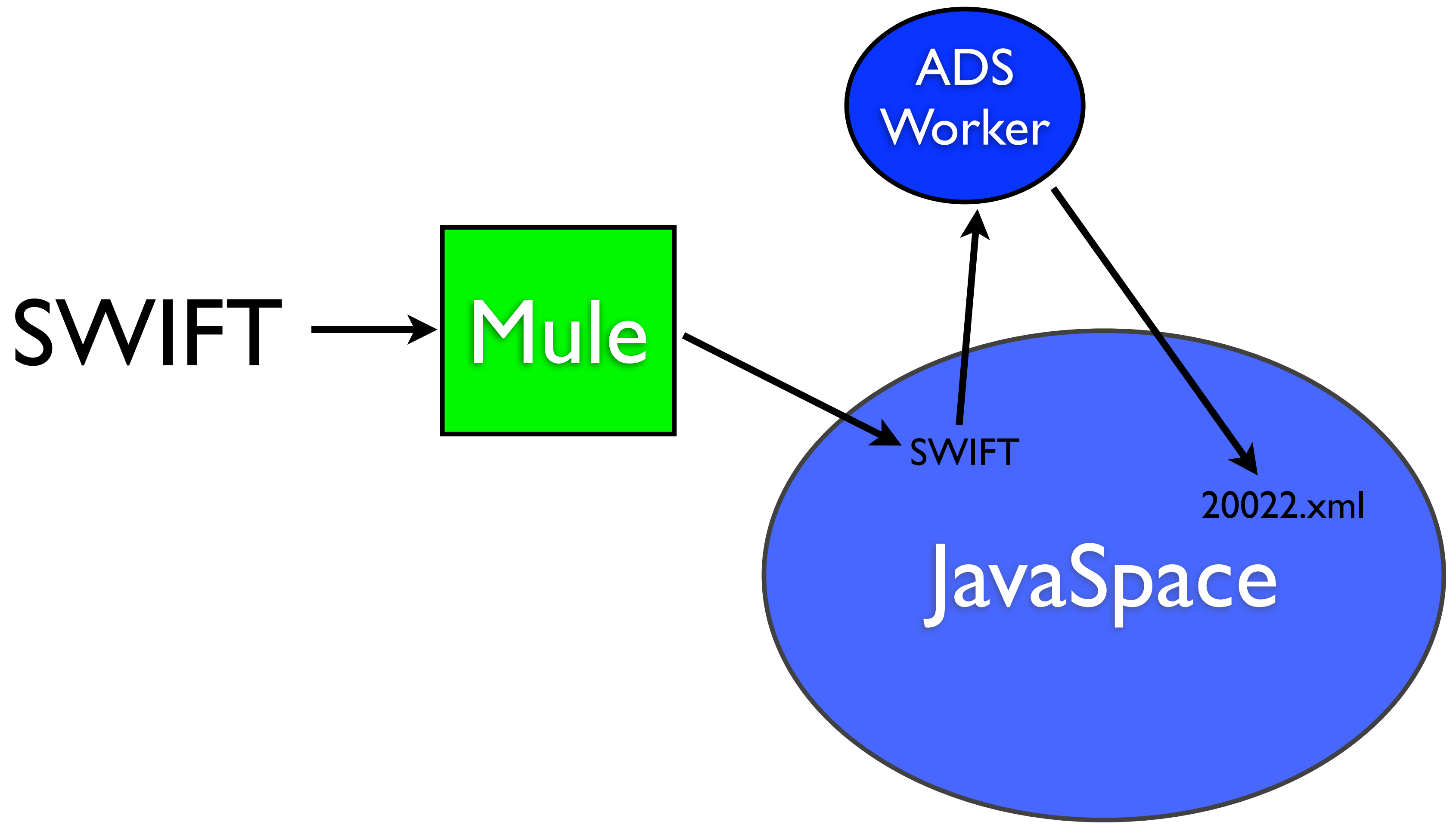
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability



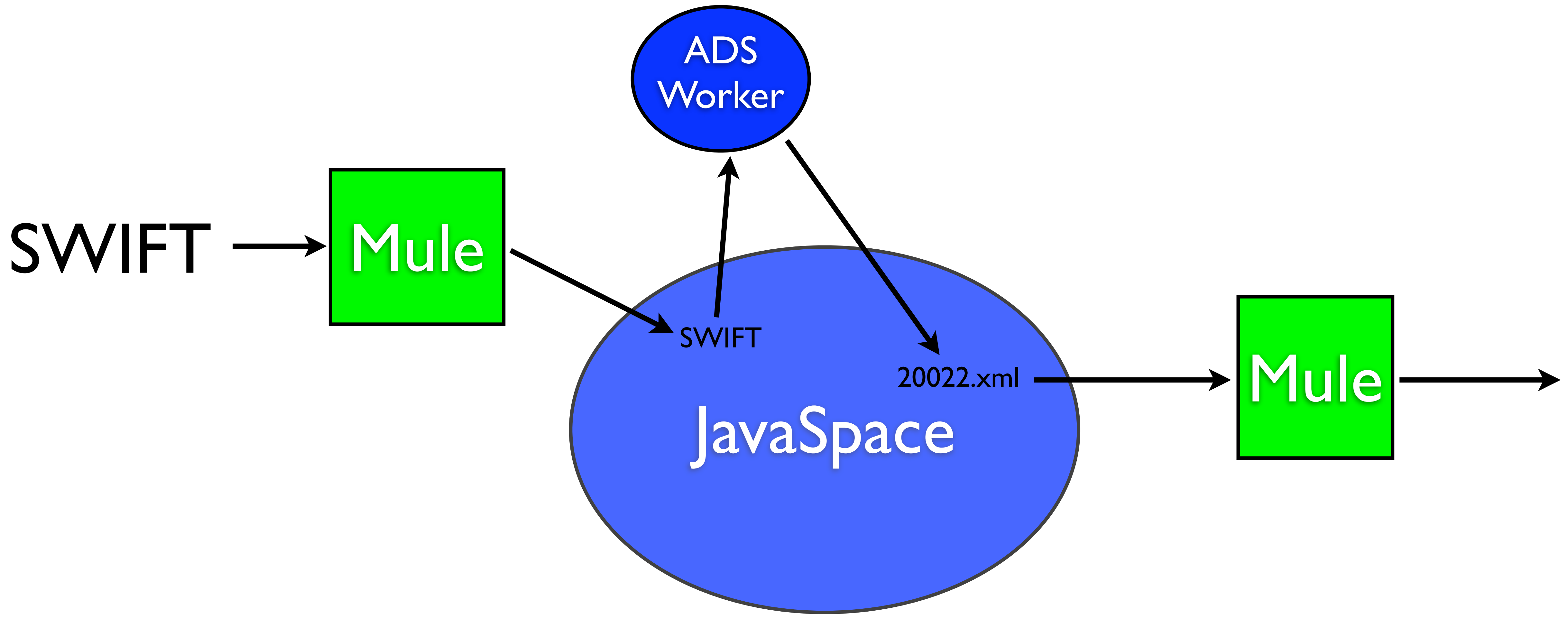
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability



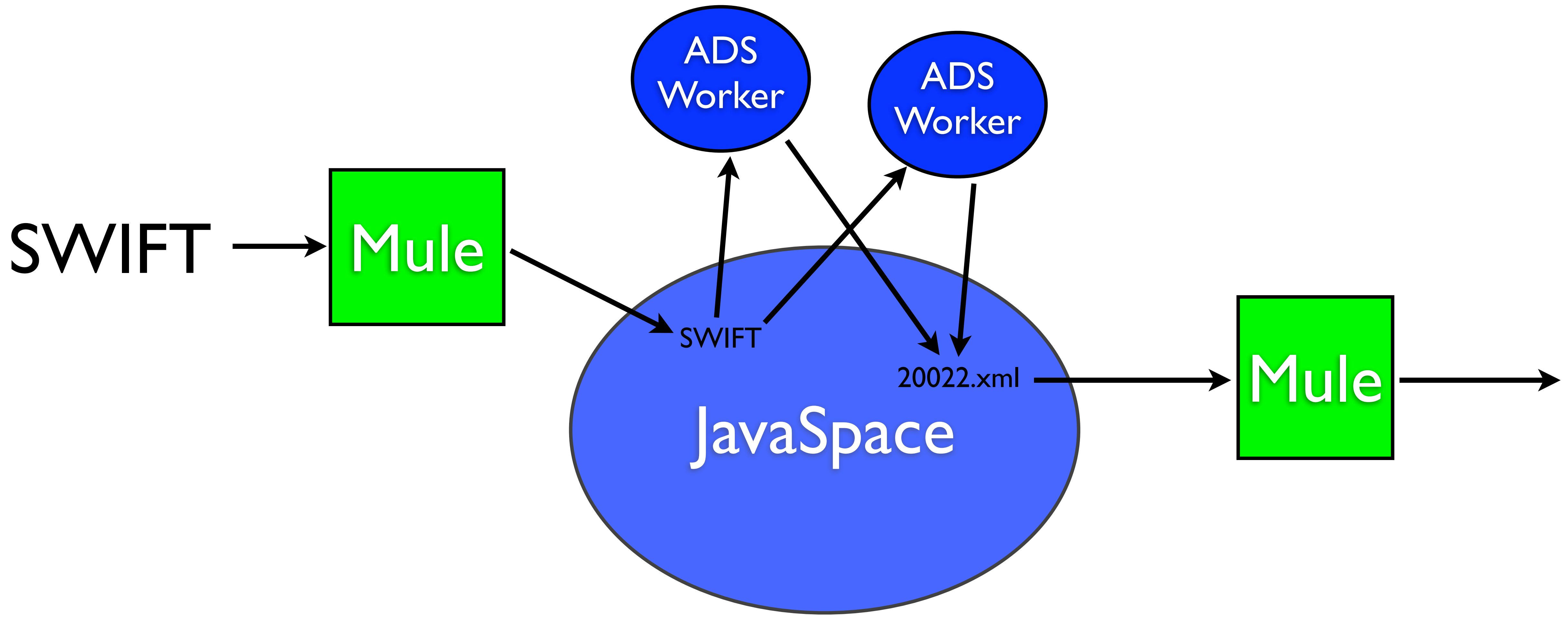
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability



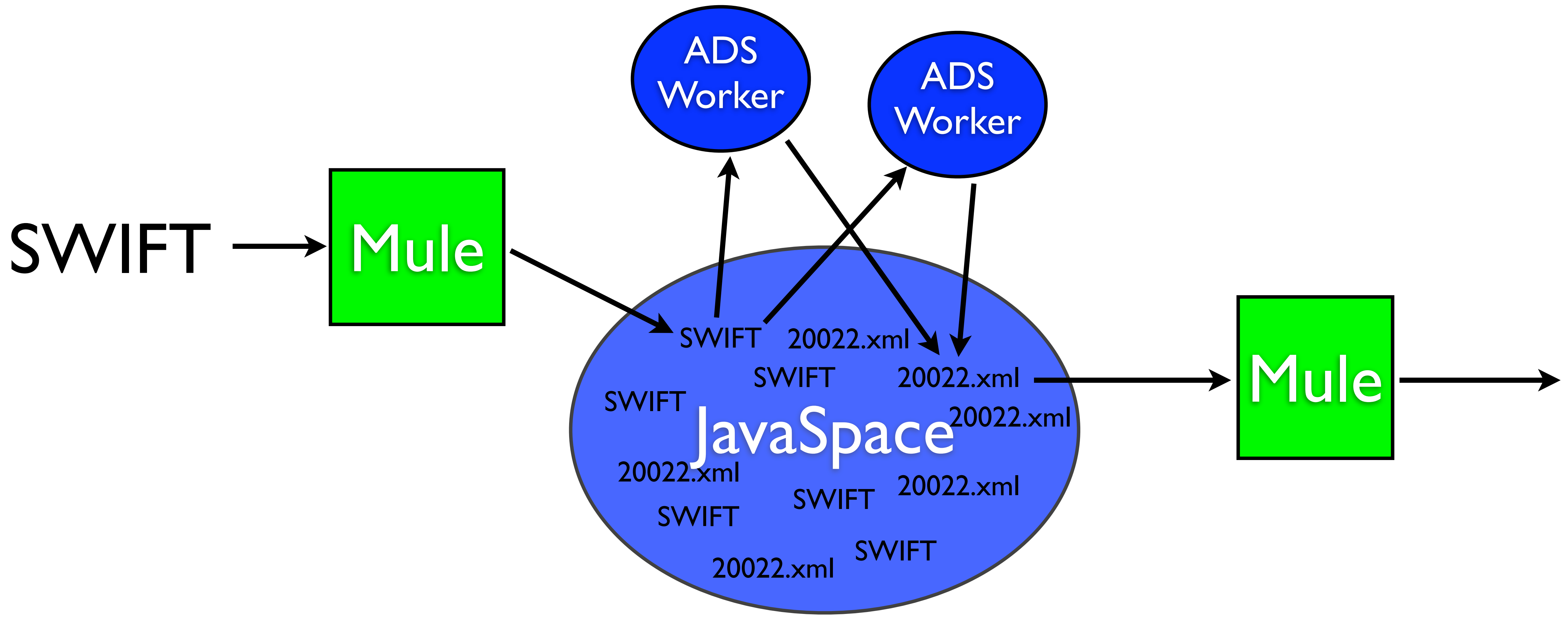
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability



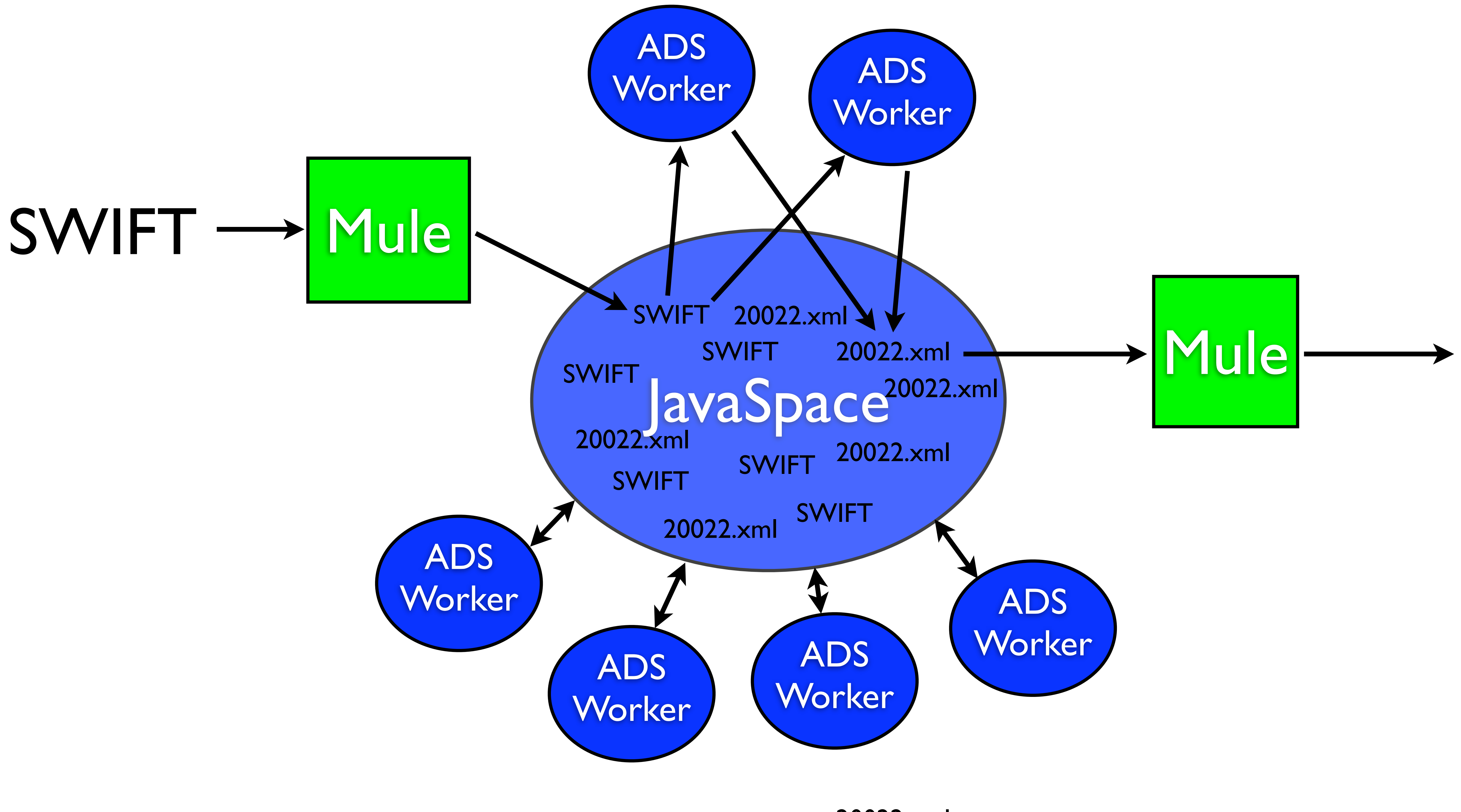
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability



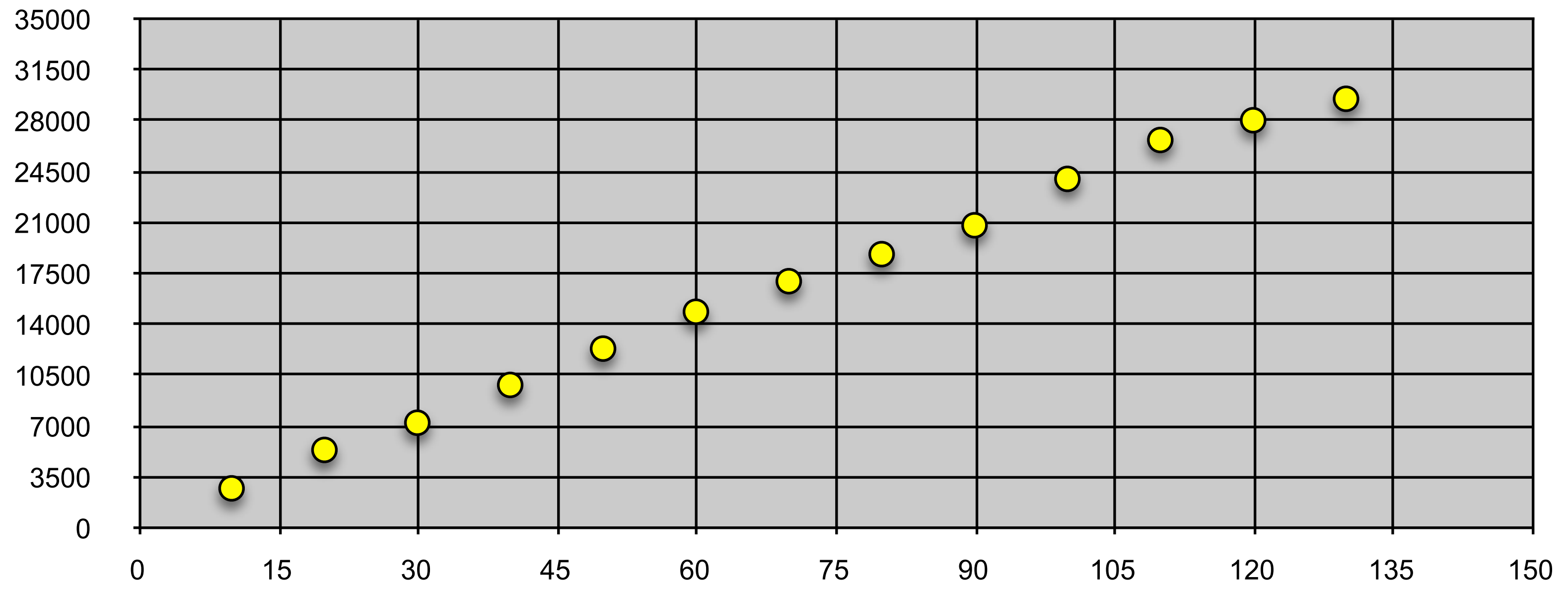
Scaling

- This is where GigaSpaces comes in...
- It adds a Spring 2 based container with huge scalability



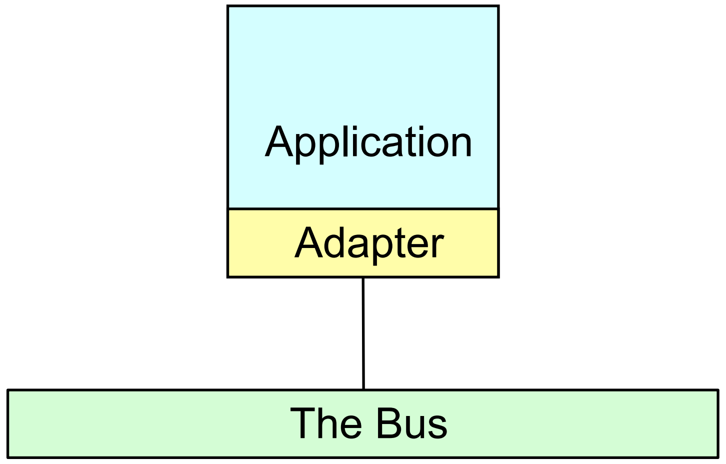
Linear scalability

- The graph below is actual data from Azul after scaling the number of threads on a 192 core box
  - x axis is the number of worker threads
  - y axis is SWIFT messages/second on the vega2 CPU



In a Nutshell

- **Use a Model Driven Architecture**
- **Use a standard like XML Schema or UML 2.0 to define the models**
  - But importantly don't assume implementations are always XML
- **Use a standard like WSDL to define the data/transport bindings**
  - But importantly and in most cases don't use classic Web Services
- **Use XSLT and XQuery to define transformations**
  - But again don't assume XML is the source or target
- **Validate extensively, syntactically and semantically**
  - This vastly reduces errors further down the line
- **Keep the "bus" simple, it just delivers canonical messages**
  - But make sure you have abstracted it


Application
Adapter
The Bus