



Enabling Grids for E-scienceE

# Building a robust distributed system: some lessons from R-GMA

*CHEP-07, Victoria, Canada, 3-7 September 2007*

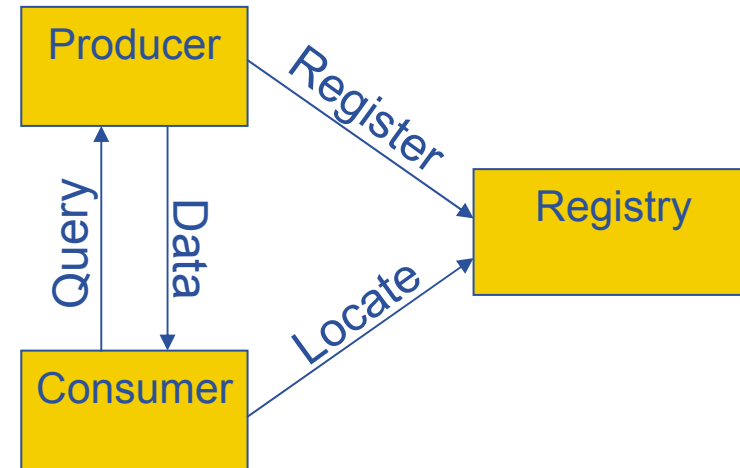
*Steve Fisher/RAL on behalf of JRA1-UK*

[www.eu-egee.org](http://www.eu-egee.org)  
[www.glite.org](http://www.glite.org)



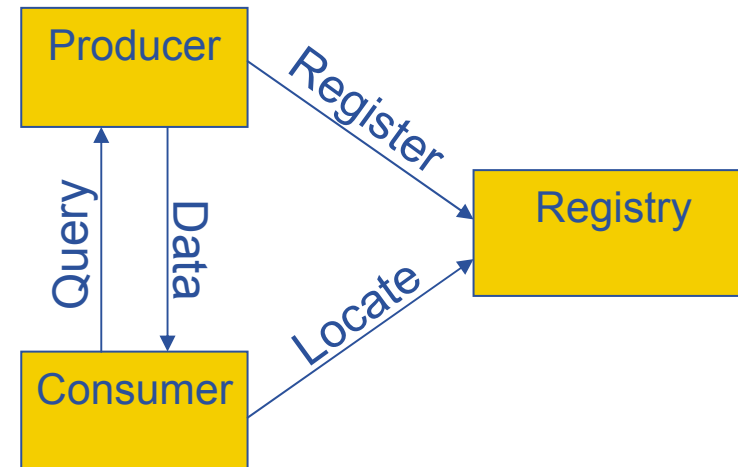
- **GMA and R-GMA overview**
- **Managing Memory Usage**
- **Buffers**
  - Consumer
  - Primary producer
  - Secondary producer
- **Loss of control messages**
- **Replication**
  - Schema
  - Registry
- **Conclusions**

- **Defined by the GGF**
  - Now OGF
- **3 Components**
  - Producer
  - Consumer
  - Registry
  
- **Real system needs to tie down message formats**
  - This has been done by R-GMA

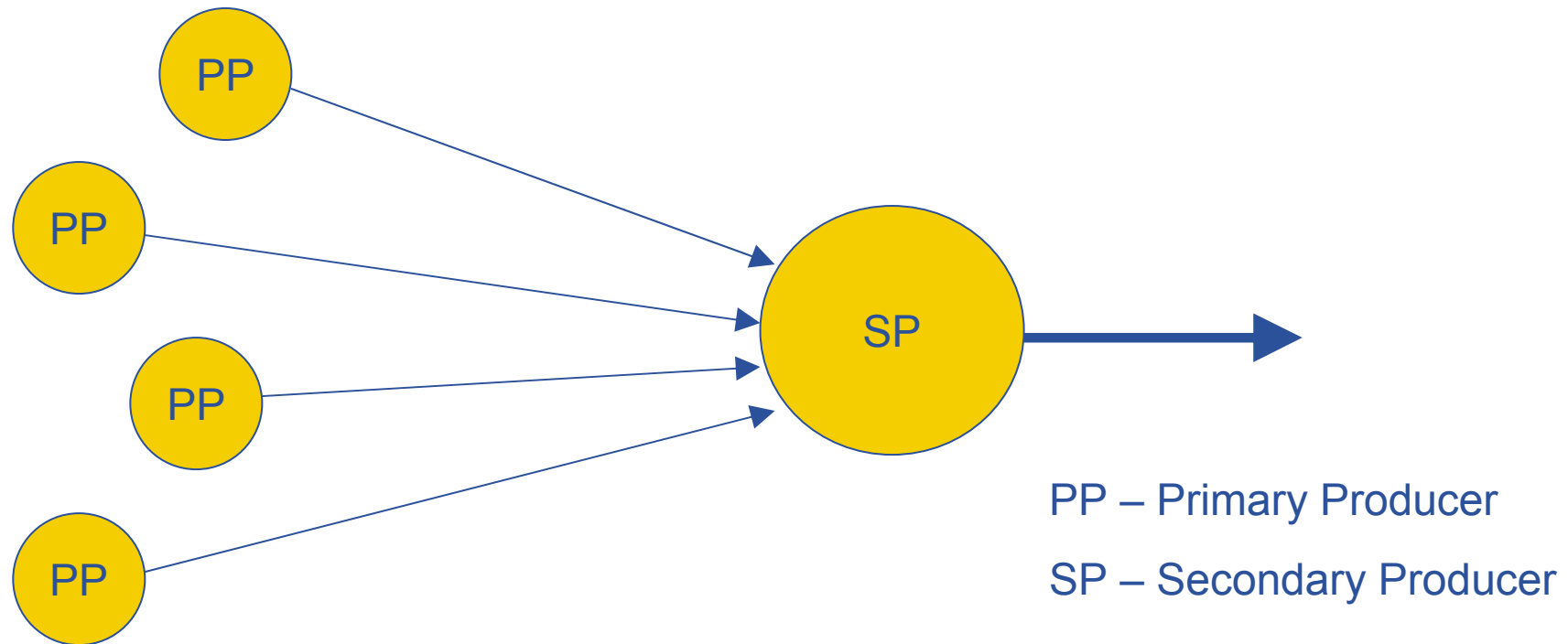


- **The INFOD-WG at GGF**
  - IBM, Oracle and others have defined a GMA compliant specification

- Relational implementation of the GGF's GMA
- Provides a uniform method to publish and access both information and monitoring data
- Registry is hidden
- It is intended for use by:
  - Other middleware components
  - End users
- Easy for individuals to define, publish and retrieve data
- All data has a timestamp, enabling its use for monitoring

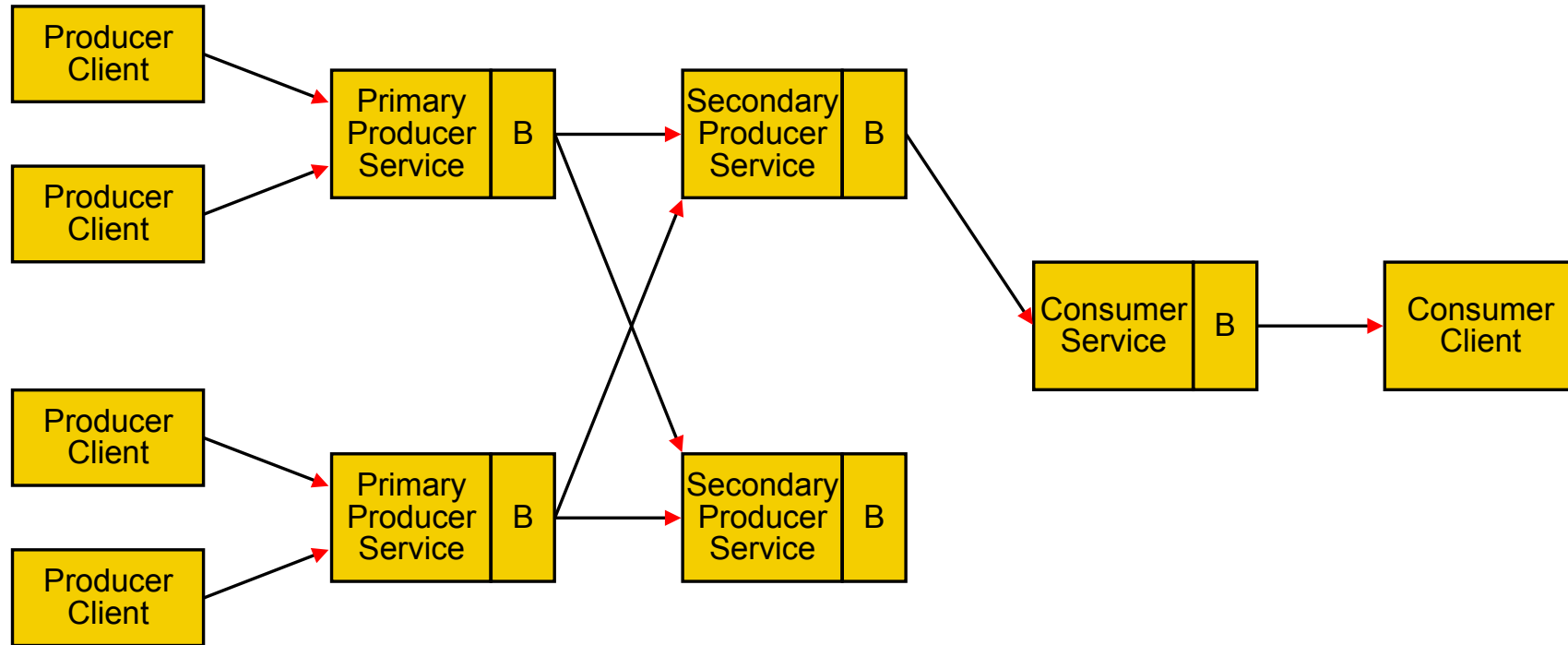


- **Primary – source of data**
- **Secondary – republish data**
  - Co-locate information to speed up queries
  - Reduce network traffic



- **EDG**
  - corresponding to the version developed within EDG.
- **EGEE-I**
  - for the version deployed in gLite 3.0
- **EGEE-II**
  - for the version that will be rolled out late Summer and Autumn of 2007 as upgrades to gLite 3.1
    - Designed to address properly all the long term problems

- **R-GMA may have varying amounts of memory available**
  - May share servlet container (Tomcat) with other servlets
  - JVM may be badly configured
- **EGEE-II solution**
  - Use JDK 5 Observer to monitor memory usage
  - When memory low RGMABusyException returned for all user calls that may take extra memory
    - inserting data into the system
    - creating new producer or consumer resources
- **We try to be fair**
  - If you behave reasonably you should not be penalised
  - If problem is caused by too many reasonable demands must reject requests with the RGMABusyException.



- **Buffers are shown “B”**
  - Primary producer
  - Secondary producer
  - Consumer



- **Consumer has a buffer for each client where the results of the query are stored until they are popped**
  - If the application is slow to pop() then buffers can fill up.
  - Cannot send an RGMABusyException to a pop() as this call reduces memory use.

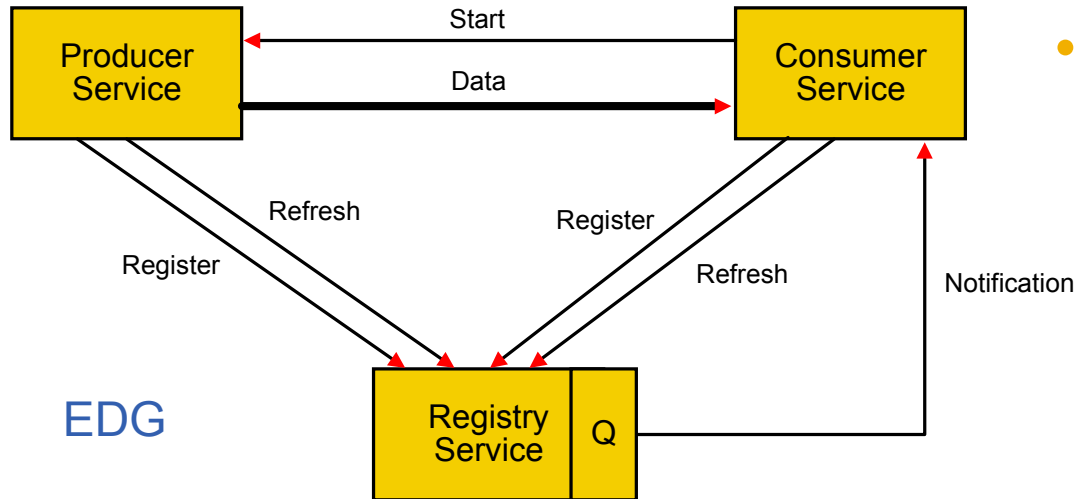
- **EGEE-I solution**

- Allocate each instance a certain amount of memory
- When this is full data are written to disk
- Once the data are all read from disk, the disk file is removed and memory is used again.
- If allocation on disk runs out we close the consumer.
- This allows us to cope with peaks of data and is working well

- **Problem only exists with memory storage**
  - Latest store to answer latest queries
    - Must hold tuples up to their LRT
  - History store for history and continuous queries
    - Must hold enough tuples to satisfy the history retention period
    - Must hold tuples for which delivery to existing continuous queries has not been attempted.

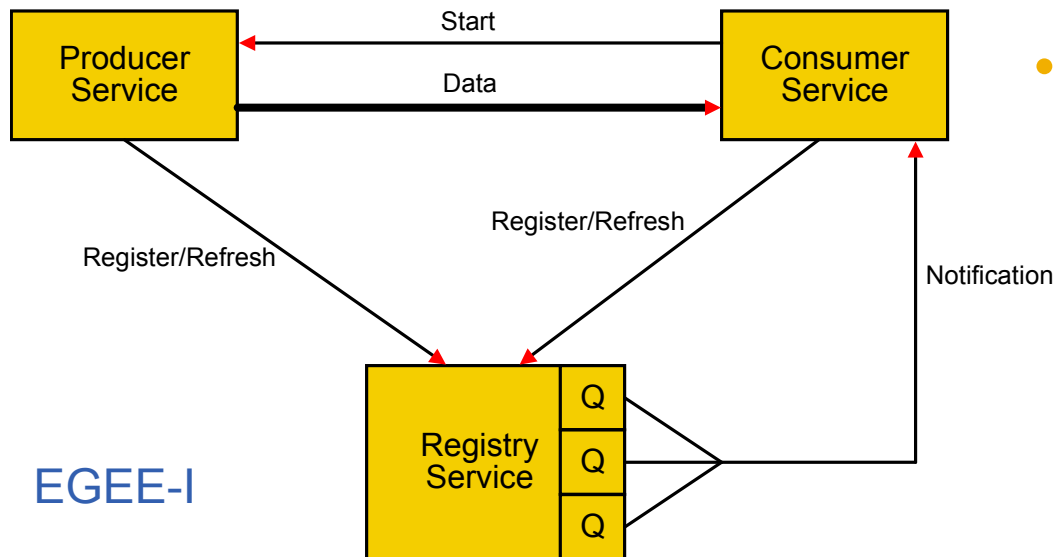
- **EGEE-I solution**
  - RGMABufferFullException which is thrown when a producer tries to publish a new tuple and the producer has exceeded a server defined limit.
  - Works most of the time
- **EGEE-II extra**
  - We have the RGMABusyException to fall back on.

- In EDG and EGEE-I designs the secondary producer was made up of consumers and one producer.
- In the EGEE-II design incoming data are stored directly in the tuple store.
- A memory based tuple store can grow very large but nobody to send an RGMABusyException to.
- For this reason we do not generally recommend using memory based tuple stores for secondary producers.
- Will close the secondary producer when unable to deal with memory demands implied by retention periods.
  - This will be added to the servlet code that would normally be sending the RGMABusyException.



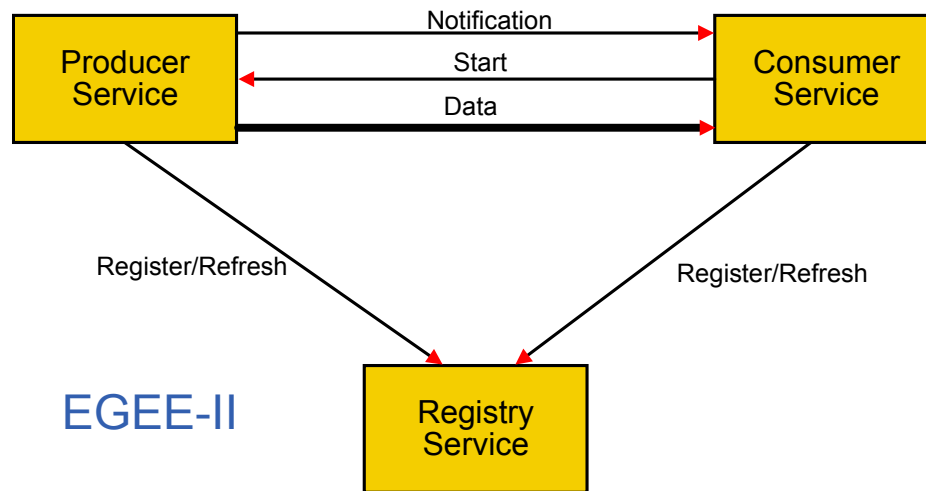
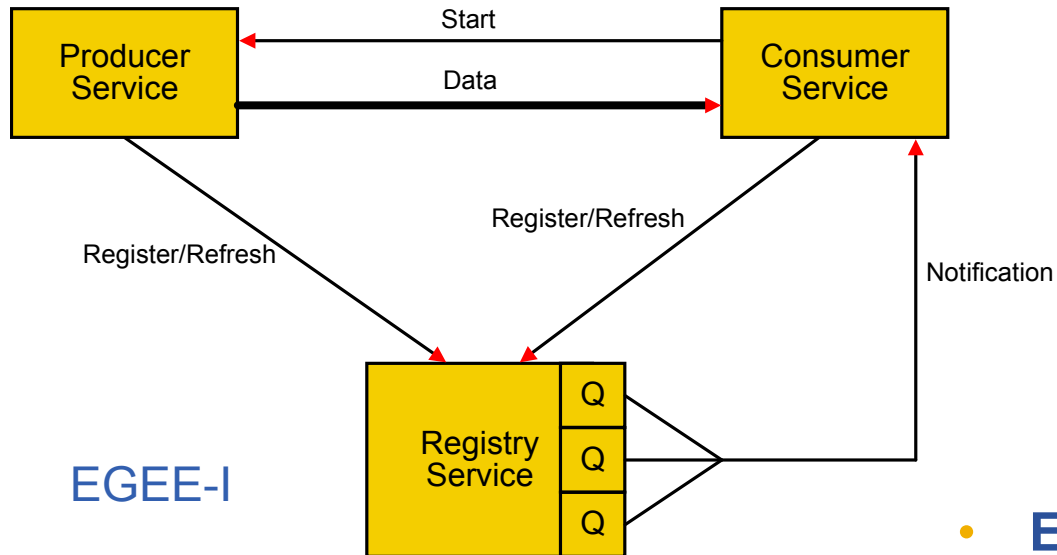
- **EDG**

- Register once
- Refresh periodically
- Only register results in notification and start
- Network problems can block everything



- **EGEE-I**

- Use register as refresh
  - No longer need messages to get through
  - But much more traffic
- Split queue into slow medium and fast queue

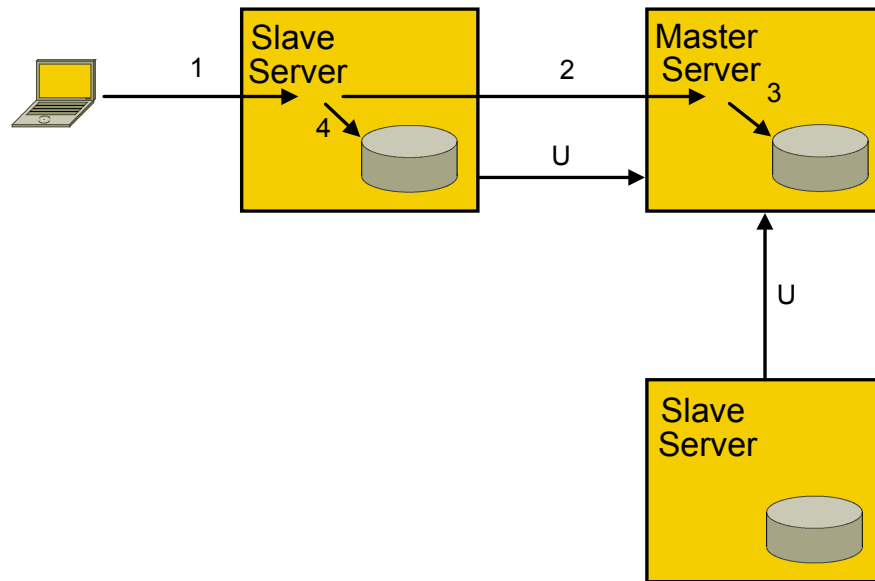


- **EGEE-II**

- For a producer a register message now return the consumers of interest and vice versa.
- Producers now notify consumers themselves
- Messages to other servers go via a task on the task queue

- **Assumption is that tasks dependent upon some unreliable resource**
  - e.g. network connection to a server and that server
- **Assign a key to each resource**
- **One queue of tasks but a pool of task invocators**
- **Initially empty set of good keys**
- **If a task is successful on its first try its key is added to good set**
- **If a task fails its key is removed from good set**
- **Only if key is in good set will more than one task be run with that key**
- **Some invocators only take tasks with a good key**





- **VDB is defined by a configuration file identifying the master server**
- **Each server has full information for each VDB served**
- **Tried to avoid a master but very difficult**
- **Updates are first done on the master**
- **Replication request is “updates since”**

- **A registry “owns” those records that were last changed by direct calls and is responsible for pushing updates of these records.**
- **The registry is updated:**
  - by direct calls – sets master flag
  - upon receipt of replication messages – clear master flag
- **If registry unavailable direct update requests are routed to a different registry instance and records in the new registry will get the master flag set.**
  - The system will clean up when a registry assumes mastership for the record and replicates its records.
- **A hash table is used to hold the add registration and delete registration requests keyed on the primary key of the entry.**
  - Each replication cycle a new hash table is created to take new entries and the old one is processed.
- **Time stamps are associated with the replication messages**
  - Can recognise missed messages and recover

- Try to think of everything that can go wrong.
- Keep it simple.
- Polling is much simpler though less efficient than notification.
- Make the system self correcting and avoid critical messages.
- Avoid single points of failure.
- Reject incoming requests if a server cannot cope rather than just going slowly or crashing.
- Server code should protect itself against running out of memory.
- External conditions can change at any time: it is not good enough to just check at service startup.

This will give us a highly robust and scalable R-GMA.