



**Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom**

TELEPHONE:
INTERNATIONAL:
FAX:
E-MAIL:

**Cambridge (01223) 515010
+44 1223 515010
+44 1223 359779
apm@ansa.co.uk**

Training

ANSAwise - Distributed Communications Techniques [Eurocontrol]

Mark Madsen

Abstract

Existing computer systems may already use an approach to distribution.

This module of the ANSAwise programme describes the Remote Data Access, Remote Procedure Call, and Robust Queued Messaging approaches. It examines Robust Queued Messaging in detail, and concludes that although the technique may be valid, it is not inherently any more robust than the others.

This module also shows why failures in a distributed file system cannot be made transparent (and maintain availability) without the applications being ultimately responsible for handling errors.

APM.1618.01

Approved
Briefing Note

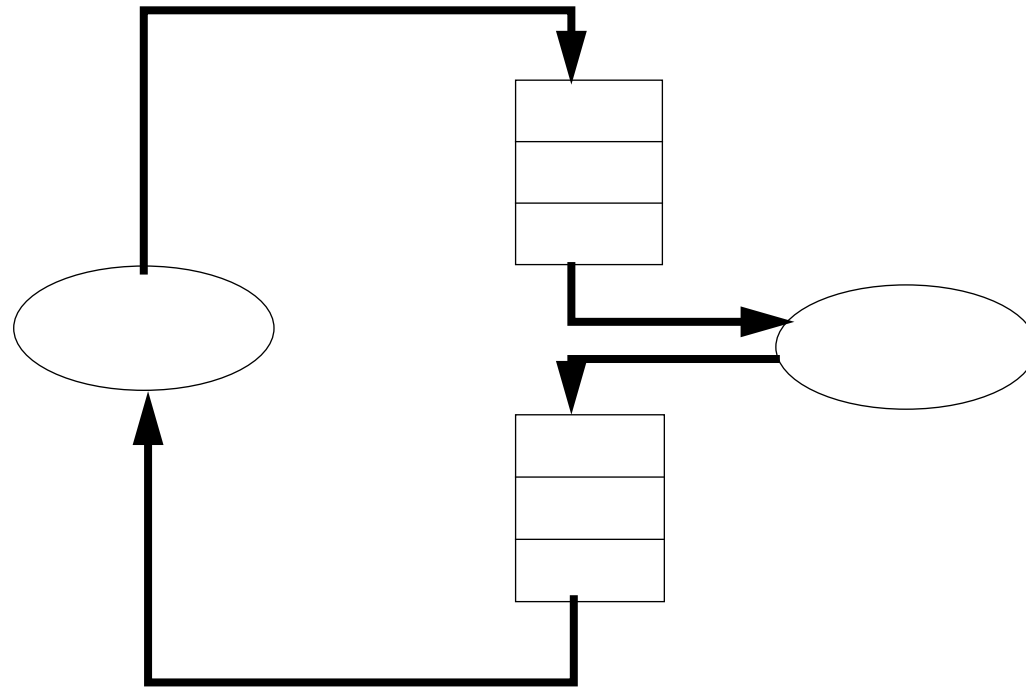
16th October 1995

Distribution:

Supersedes:

Superseded by:

Distributed Communications Techniques



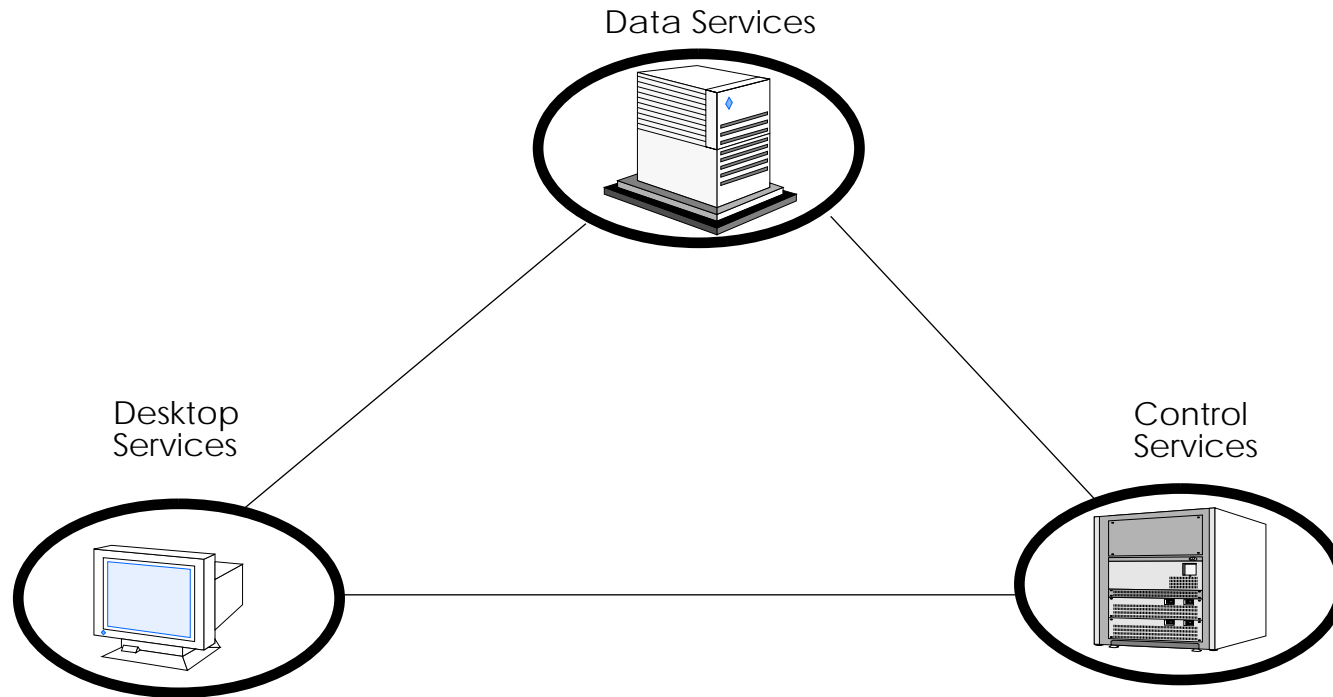


In this session

- *Explain three basic techniques for communications in distributed systems*
- *Explore one of these techniques in detail*
- *Look at standards in each of these techniques*



Communications techniques for all kinds of services





The three techniques of communication

- ***Remote Data Access (RDA)***
 - also known as remote database access
- ***Remote Procedure Call (RPC)***
- ***Robust Queued Messaging (RQM)***
 - also known as reliable messaging (RM) or message-oriented middleware (MOM)



Why use more than one technique?

- *For interoperability with existing products and standards*
 - each has different availability implications
 - each has different performance trade-offs
- *Systems can use different techniques in different places*
 - interfacing between techniques is possible

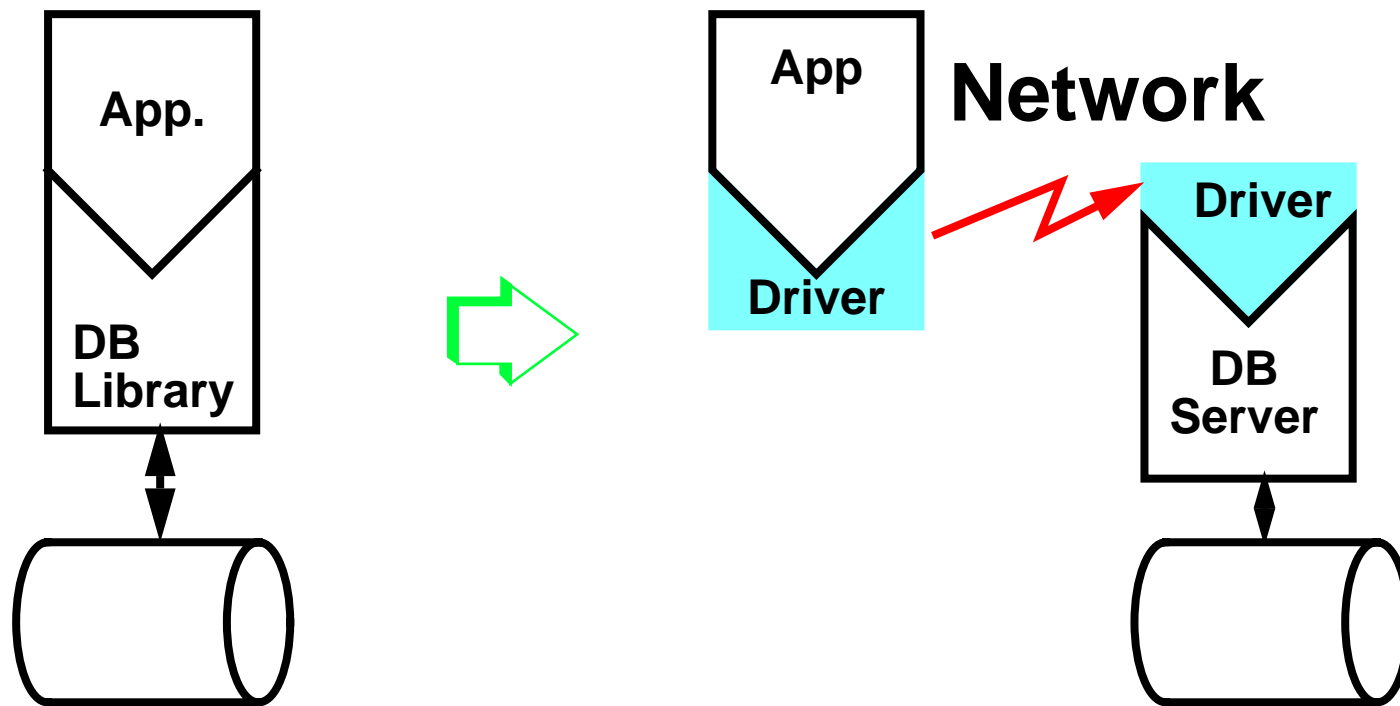


Will one technique predominate?

- *Probably not...*
 - the need to interface with legacy systems will persist
 - new products continue to refine each technique
- *...it is therefore important to understand the strengths and weaknesses of each*

Remote Data Access (RDA)

- *Local database access can be transformed into remote database access*





Applications for RDA

- *Desktop front-ends to remote databases*
 - from spreadsheets
 - from report writers
 - from 4GLs

Remote Data Access Is Not File Sharing

- *Do not confuse RDA with database file sharing...*
 - ... where a database is placed on a file server



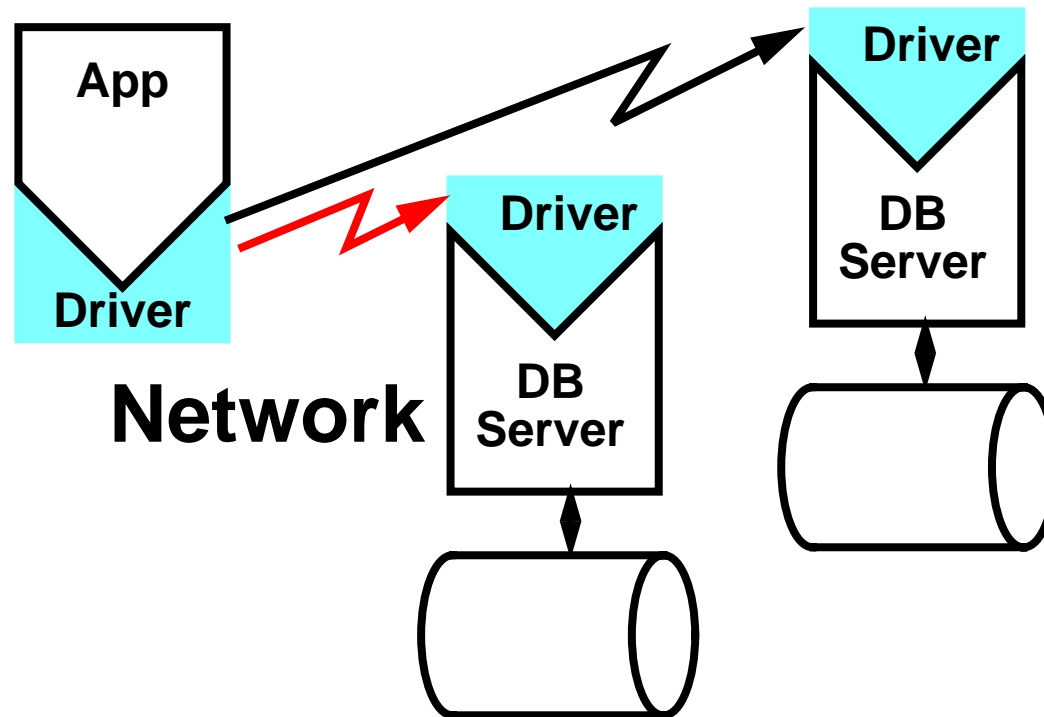


RDA and database file sharing

- *Database file sharing merely transmits file reads, writes, and locks over the network*
 - there is no database server, only a file server
- *Remote data access transmits database queries and updates over the network*

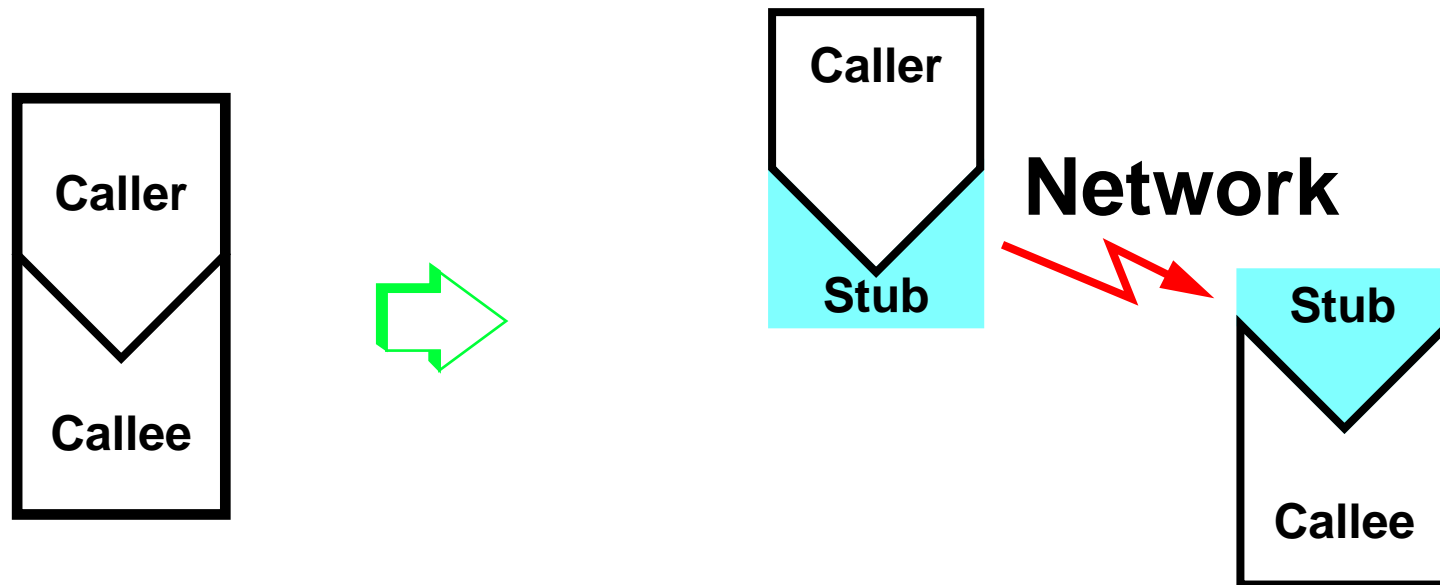
Remote Data Access Is Not Distributed Database

- *Database servers do not communicate with each other*



Remote Procedure Call (RPC)

- *Local procedure call can be transformed into a remote procedure call*



- *The caller is the client, the callee is the server*



Applications for RPC

- *Most distributed computing applications*
 - typically, applications integration
- *Particularly suited for building or integrating distributed object-oriented applications*

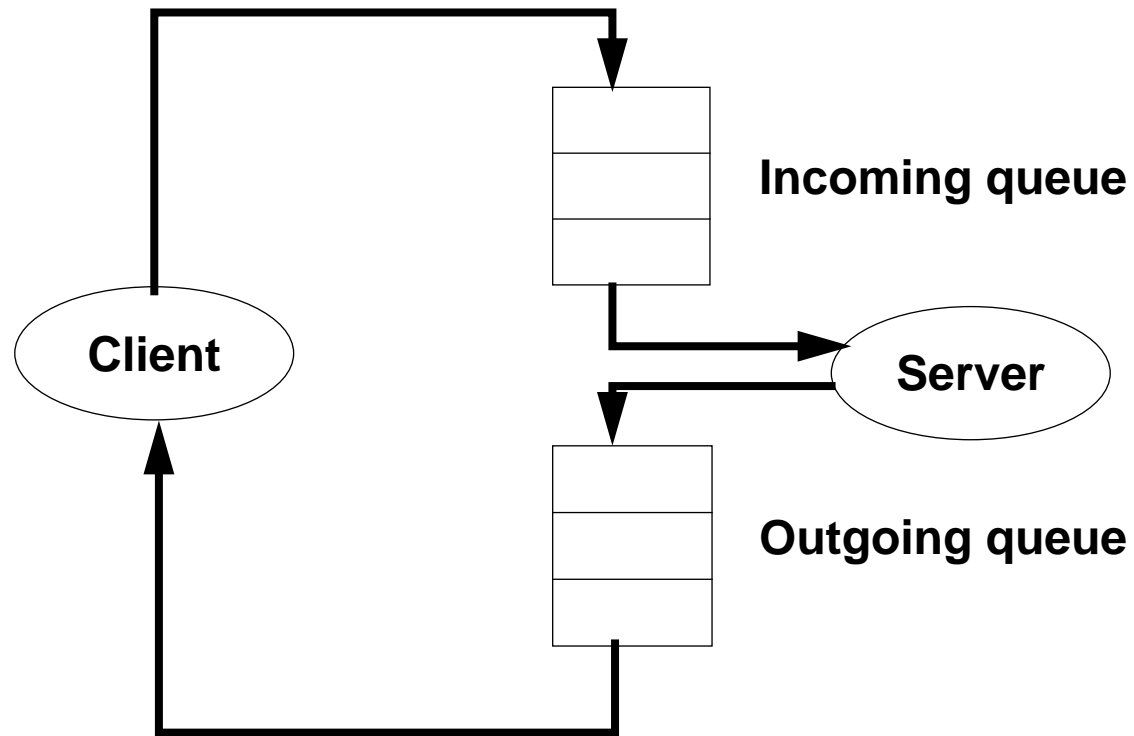


Stubs in RPC

- *The stubs in RPC are responsible for packing and unpacking the call parameters, and the call results*
 - this is called marshalling/unmarshalling
- *Stubs must allow for the fact that client and server may be machines of different types*
 - for example, integers may be represented differently (byte-ordering)

Robust Queued Messaging (RQM)

- *Requests and responses are queued*





Applications for RQM

- *real-time systems*
- *command and control*
- *transaction processing*
- *financial trading*



When is robustness important?

- *High cost of failure*
- *Rapid recovery from failure*

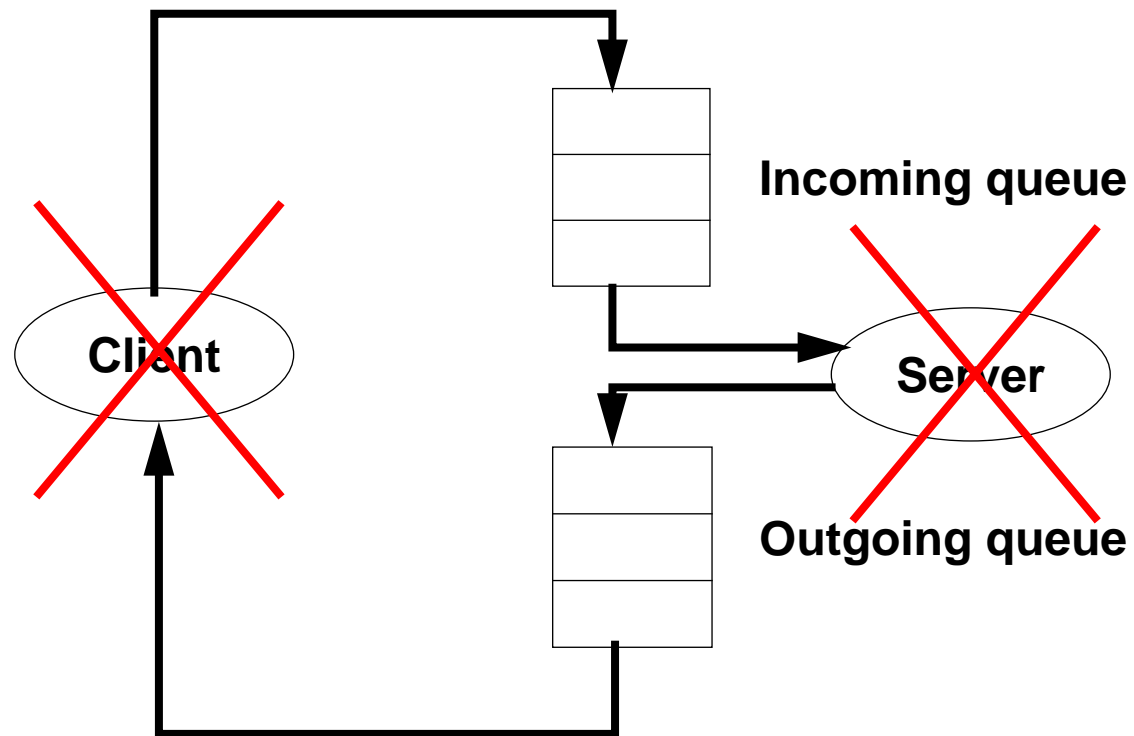


Information flow in Robust Queued Messaging

- *Client and server do not communicate directly, but via intermediate queues*
 - client places request on queue
 - server retrieves request from queue
 - server places response on queue
 - client retrieves response from queue

Failure in Robust Queued Messaging

- *Client and server can fail independently; queued messages not lost*





Client Failure and Recovery

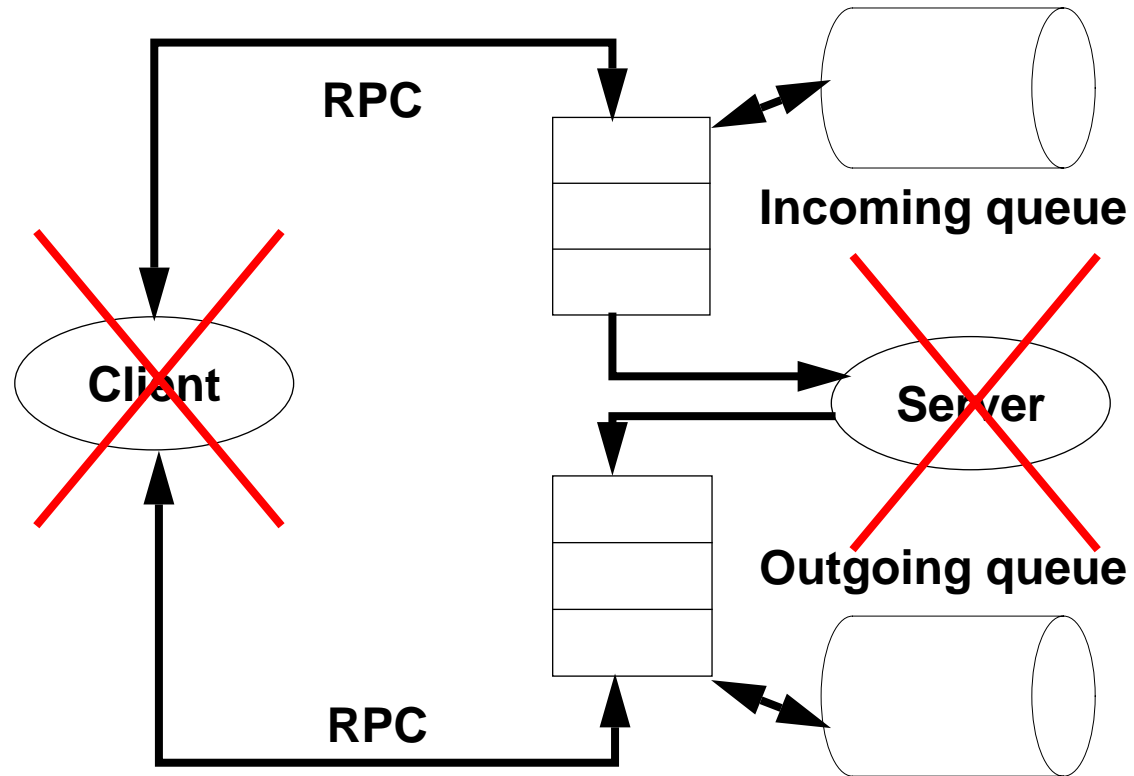
- *If a client fails and is restarted, it can receive pending messages from the outgoing queue*
 - responses to a request from an earlier 'incarnation' of the client
- *This is an unfamiliar challenge for most application designers*
 - how to handle these particular responses?



Failure and Recovery in Robust Queued Messaging

- *Client and server can recover messages from incoming and outgoing queues*
- *What other potential points of failure are there?*
 -
 -
- *What can be done about them?*
 -
 -

RQM using RPC





Handling queue overload

- *Some commercial products simply discard messages on overload*
 - this is inconsistent with the aim of robustness
- *Queue overload control requires either flow control or rate control*
 - this is more complex than in RPC, where flow control/rate control can be part of the RPC protocol itself
- *Do not confuse queuing with buffering*
 - large messages may be fragmented and require buffering for defragmentation...
 - ... but this is a communications issue, not a distributed systems issue

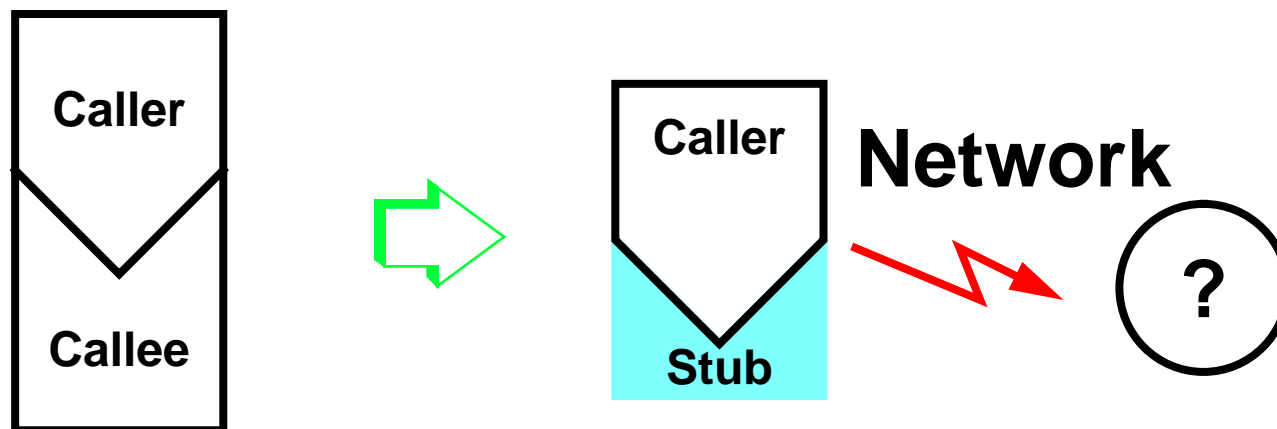


Dependability - beyond robustness

- *For many applications, robustness is not enough...*
- *...they need dependability, including*
 - **Atomicity (transactions)**
 - **Fault tolerance**
 - **Security**

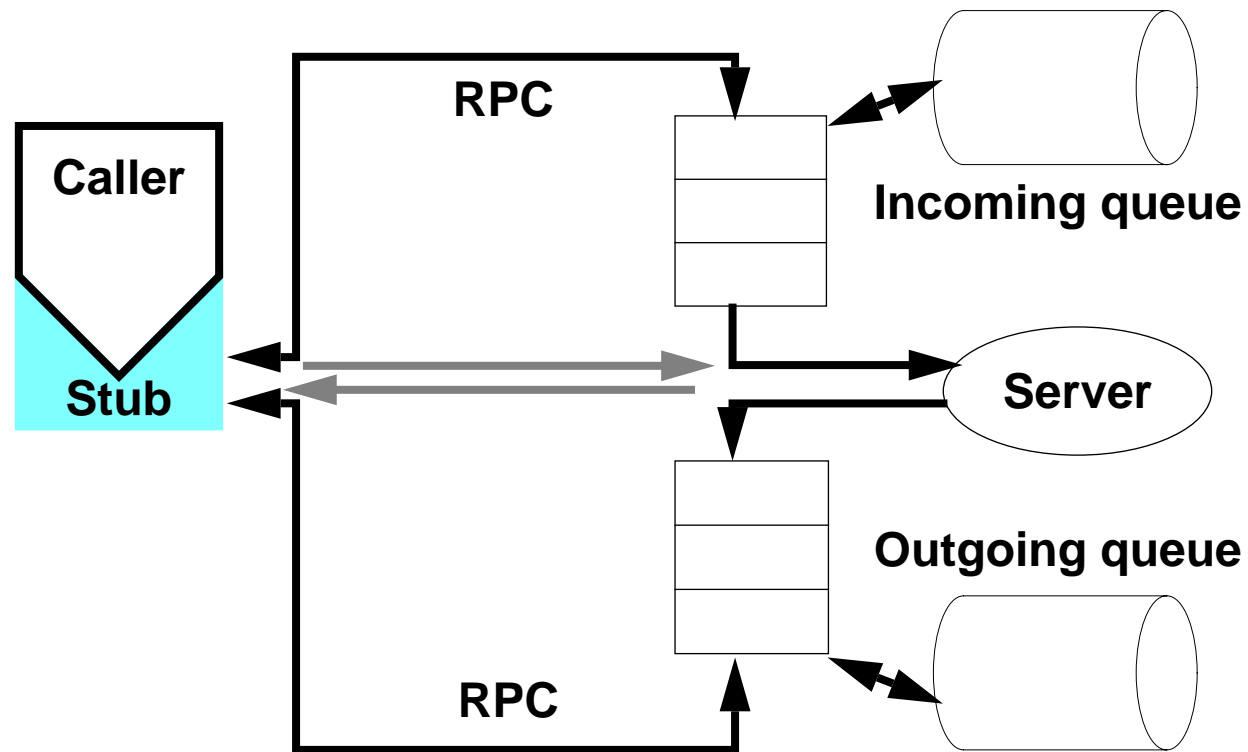
Queue transparency

- *Should the client even be aware of the queue?*



RPC interface to RQM

- **Caller just sees a simple RPC interface**





RPC interfacing to RQM

- *The programming interface to RQM can be complex...*
 - most programming languages don't support message queuing
 - some programming languages are described as 'message-passing'...
 - ... but this is a different meaning of the word 'message'
 - ... they don't support message queuing any better than ordinary programming languages
- *...using a standard RPC simplifies RQM for the application designer*
 - whichever programming language you use



RQM management

- *Queues naturally support testing and debugging...*
 - message logging, filtering, and tracing
 - message insertion and deletion
 - load monitoring and balancing, flow control
 - management
- *... these can be separate interfaces to the queue itself*



Exercise: RQM using objects

- *The following exercises refer to “RPC interface to RQM”*
- *Replace all components of the diagram with objects*
- *How is the state of each queue represented?*
- *Add interfaces to the queue objects to support*
 - *queue management*
 - *message insertion/deletion*
 - *message logging and tracing*
- *How can message filtering be implemented?*
- *How could load balancing be implemented?*



Other RQM possibilities

- *RQM can be a good interface to legacy systems*
 - the two sides of the queue can use different protocols
- *RQM allows complex message patterns (the response can come from a third process, multiple responses, one-way messages,...)*
 - such designs do not scale well and can be difficult to maintain
- *RQM can support off-line (e-mail style) messaging*
 - messages can remain in queues for long periods...
 - ... servers can poll queues, or only run periodically



How effective is RQM?

- ***Efficiency***
 - mainframe transaction processing (TP) monitors gain efficiency from queuing
- ***Managability***
 - queuing does offer an effective point of control
- ***Robustness - arguable!***
 - ... in practice RQM tends to be robust, but this is probably due to the care and attention given to typical design and implementation
 - ... plain RPC should be at least as robust under the same conditions; there is less to go wrong



RQM in practice

- *The queue manager can be made very simple*
- *The queue manager can be integrated with the system...*
 - giving high throughput
- *... or integrated into applications libraries*
 - leaving a more lightweight micro-kernel, and giving the application control of queuing policies and mechanisms
- *The latter is more likely in the future*



Selecting an RQM system

- *If you believe RQM is appropriate technology, you need to be clear about your specific needs*
 - Different systems offer different kinds of robustness (for example, in overload)
 - RQM systems may or may not offer transactional (atomic action) capabilities
 - Different RQM systems do not interoperate
 - RQM systems may or may not offer real-time quality-of-service
 - RQM systems support online reconfiguration differently



Other issues for RQM

- *Technical expertise can be hard to acquire*
 - it is spread across specialist markets

- *Off-the-shelf products are available but can be expensive*
 - IBM MQSeries, DECtr, Tandem Pathway, ISIS



Standards for RPC

- *Several standards are competing*
 - OSF DCE RPC
 - Sun ONC
 - proprietary (for example, Netwise RPC)
- *CORBA will support at least one mandatory interoperable RPC, plus others as options*
 - multiple protocols simultaneously under one API



Standards for RDA

- ***APIs (also known as call level interfaces (CLIs))***
 - Based on the X/Open and SQL Access Group (SAG) CAE specification
 - Based on SQL as the data definition and manipulation language
 - Microsoft Open Database Connectivity (ODBC) is the best-known variant
- ***Protocols (also known as formats-and-protocols (FAPs))***
 - IBM DRDA (based on LU6.2)
 - OSI-SAG-X/Open RDA/SQL (based on OSI)
- ***Vendors are continuing to promote proprietary APIs and protocols***

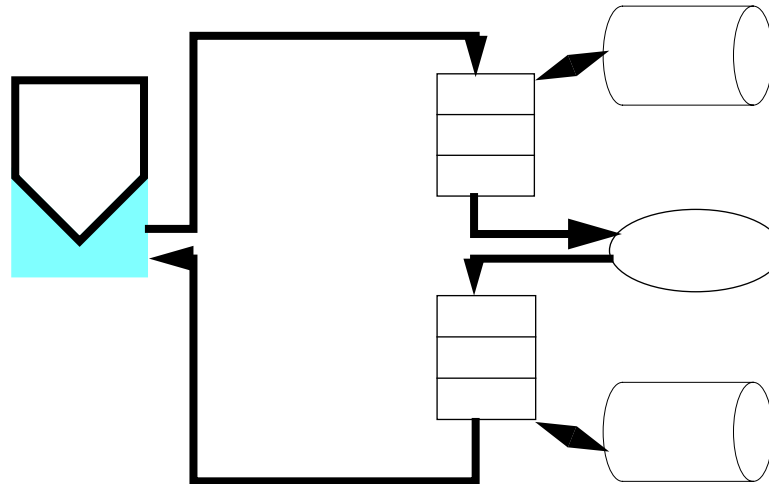


Standards for RQM

- *Currently there are no generic standards for RQM*
 - the Message Oriented Middleware Association (MOMA) is attempting to resolve this...
 - ... by providing interoperability, at least

Hybrid techniques - using RPC with RQM

- *RPC can be implemented over RQM, (transparently or not)...*



- *...RQM can be implemented over RPC*
 - as in Channels in the CORBA Event Service



Summary

- ***Robust Queued Messaging (RQM) is a misnomer; the queuing is the important aspect***
 - queuing improves performance, not robustness; it belongs in application libraries, not in the system
- ***100% reliability is impossible***
 - cost-effectiveness depends on careful application design and system engineering
- ***RDA, RPC, and RQM each have their place***
 - RPC is the most natural fit to most programming languages
- ***For more information on robust messaging***
 - see *Transaction Processing Concepts and Techniques* by Jim Gray and Andreas Reuter (Morgan Kaufman)