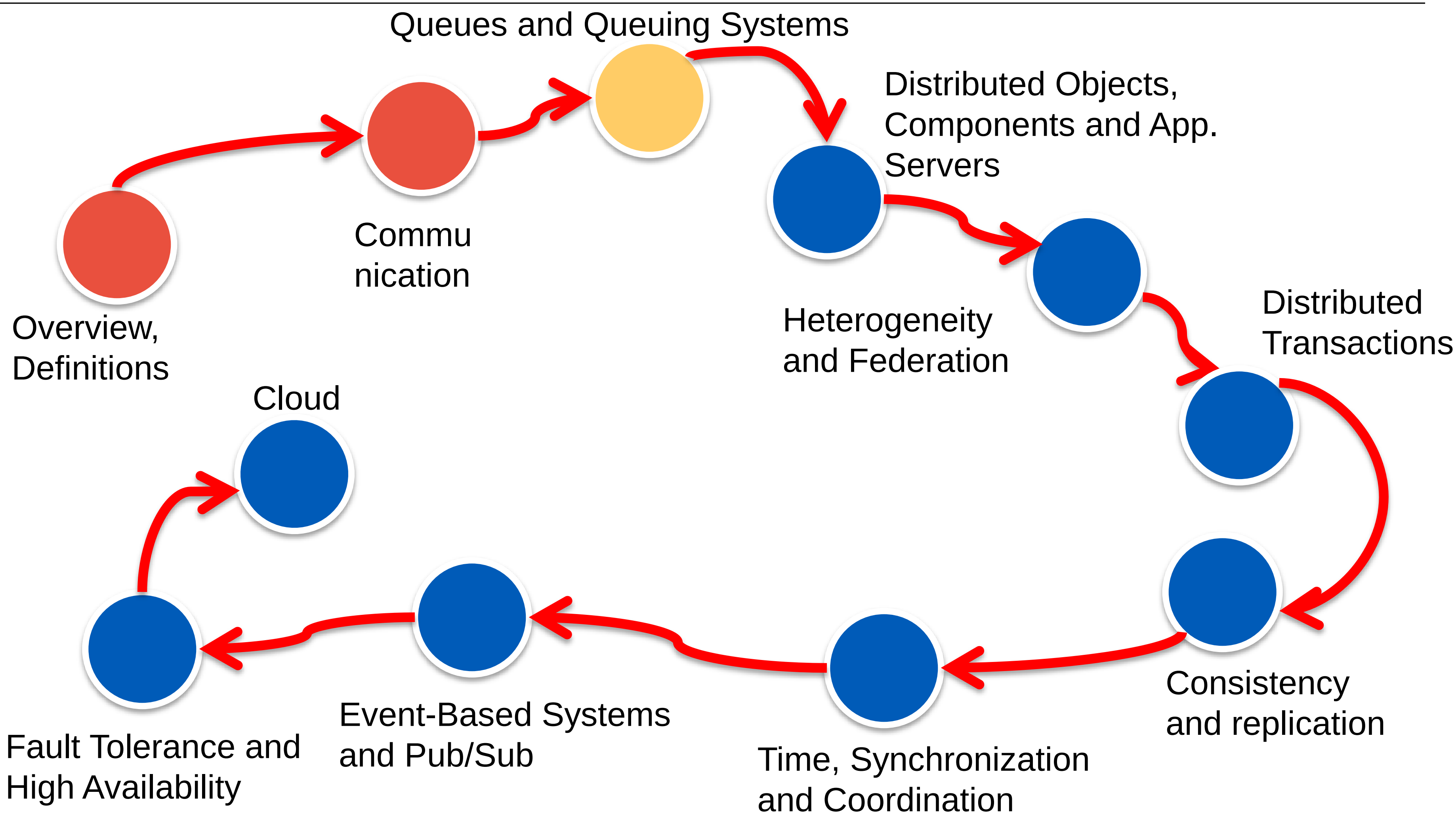
# Middleware:
## 3. Queues, Systems and Message-Oriented Communication

I. Petrov, A. Buchmann
Wintersemester

# Topics


Queues and Queuing Systems
Distributed Objects, Components and App. Servers
Commu nication
Overview, Definitions
Heterogeneity and Federation
Distributed Transactions
Cloud
Consistency and replication
Event-Based Systems and Pub/Sub
Time, Synchronization and Coordination
Fault Tolerance and High Availability

# Topics

- Queues
- Queuing Systems
- Message-oriented communication
- Asynchronous RPC

# Reading for THIS Lecture

- The slides for the lecture are based on material from:
  - Andrew S. Tanenbaum and Maarten Van Steen. 2001. **Distributed Systems: Principles and Paradigms.** Prentice Hall.
    - Chapter 2.4
  - Philip A. Bernstein, Eric Newcomer. 1997.
    **Principles of Transaction Processing.** Morgan Kaufmann
    - **Chapter 4**
- WebSphere MQ Primer: An Introduction to Messaging and WebSphere MQ
  http://www.redbooks.ibm.com/redpapers/pdfs/redp0021.pdf
  IBM, Dec 2012
  (last accessed Nov. 7, 2013)

# Message Oriented Communication

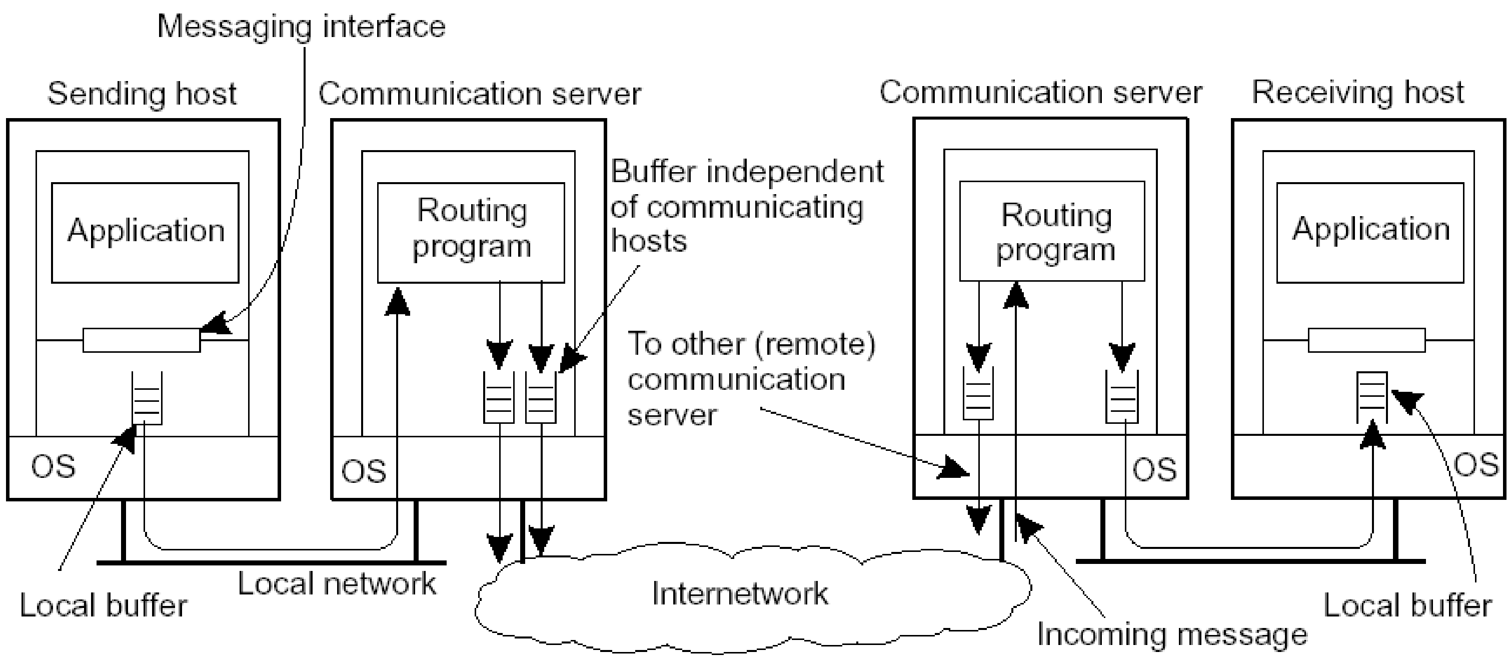

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

# Synchrony

- Request/Reply mechanisms based on synchronous communication model:
  - Pros:
    - Simple programming
    - High access transparency
  - Cons:
    - Blocking → Scalability problems
      - Client sends request and is blocked until it receives a reply
      - Must handle failures immediately (the client is waiting)
    - Client and server must be simultaneously active!

- Many situations where synchronous processing is not adequate:
  - Application Integration
  - Mobility
  - Messaging, e.g. eMail

# Communicating Systems - Schematic




Q: What happens to messages? What factors influence the fate of a message?

# Communication Dimensions and Decoupling

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Synchronicity**
  - Synchronous
  - Asynchronous

- **Persistence**
  - Transient Communication
    - Message ‘buffered’ as long as sender and receiver are active
    - If delivery impossible – drop message

  - Persistent Communication
    - Store Message before transmission
    - Successful Delivery – guarantees, decoupling

# Asynchronous Communication: Messaging

- Message-oriented middleware (MOM): Aims at high-level asynchronous communication
  - Processes send each other messages, which are queued
  - Sender need not wait for immediate reply, but can do other things
  - Middleware often ensures fault tolerance
- Examples: IBM WebSphere MQ (used to be MQSeries), Hornet Q, Active MQ
- Integrate applications
  - Large scale, Dispersed networks, (Geographical) scalability
  - Decoupling in time
  - Guarantees (depending on persistence)
  - Interoperability and portability – interfaces, message formats, conversions



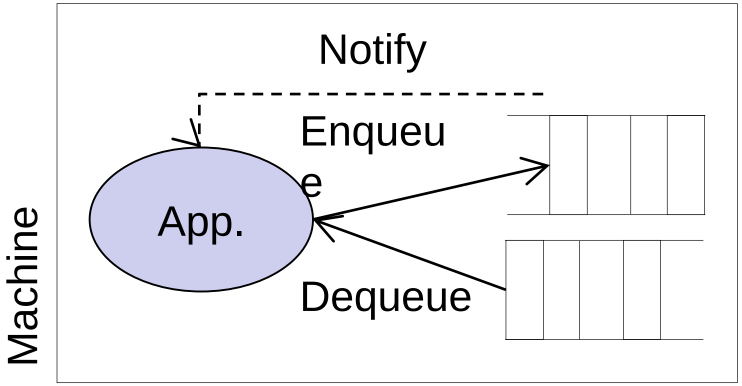
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

# Queues


Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

# Message-Queuing Model

- An application is associated with an:
  - input (request) queue
  - output (reply) queue
  - these can be dedicated or shared (between several applications)
- Queue Interface – simple
  - Enqueue – put/append a message in the queue (non-blocking)
  - Dequeue – remove the first message from the queue (blocking)
  - Poll – check if the queue is non-empty. If yes dequeue first message (non-blocking)
  - Notify – install a callback function. Notify application upon arrival of a message
  - Scan (open-scan and get-next-element) – iterate over all messages in queue



# Message-Queuing Model

- Messages
  - Different formats (Msg. can contain any data)
    - there are typed queues
  - Arbitrary size – the queuing system handles assembly, disassembly, transmission
  - Addressing – messages should contain the name of destination queue

- Guarantees: Message delivered to recipient queue
    - No guarantees given about delivery time and whether message is read by recipient
    - Decoupling in space and time
      - Sender and receiver execute independently

- Durability: Persistent vs. Transient Queues
  - NOTE: persistent to communication specialists means that a message remains in the buffer, persistent to database and middleware people means message is stored and a non-persistent queue is one that only buffers

- Transactional vs. Non-Transactional Queues
  - Queue operations are executed transactionally (new Tx or as part of an existing Tx)

# Message-Queuing Model

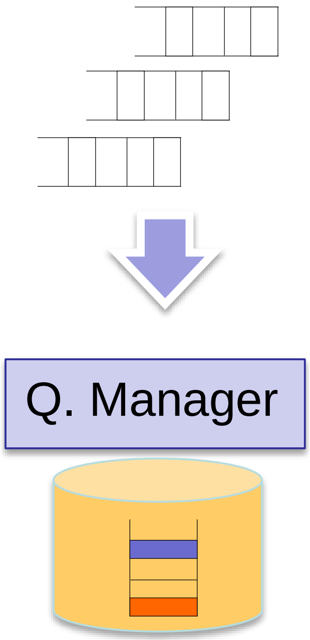
- Priority queues

  - Prioritize queue elements
  - Dequeue by priority

  Operations are now
  - Put-with-priotity
  - Get-highest (or get-lowest)

# Queues and Queue Managers

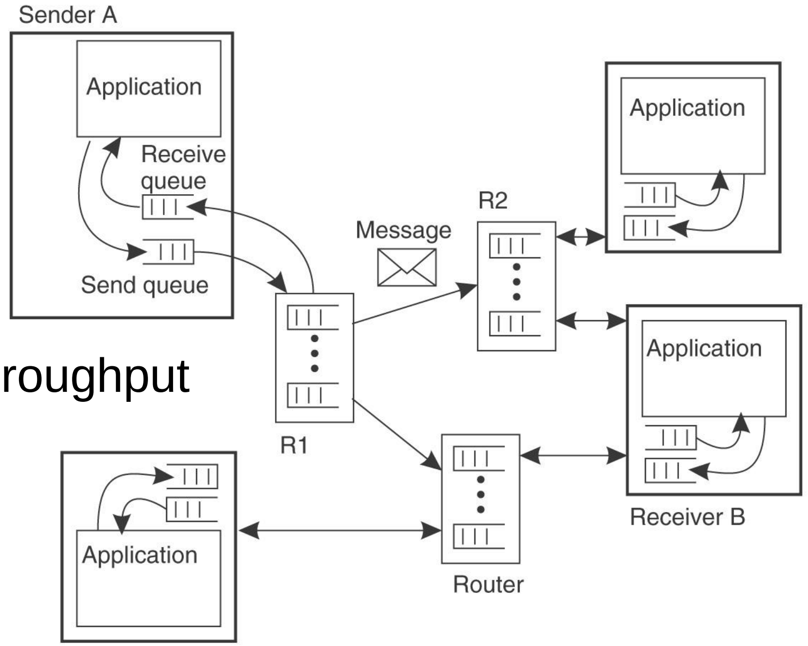
- Queues are managed by Queue Managers
  - One Q.Manager manages many queues
- Queue managers
  - Interact directly with applications
  - Act as ‘message routers’ → relay msg. to other queues




- Queue Manager’s Interface:
  - Queue Interface: Enqueue, Dequeue, Poll, Scan, Notify
  - Housekeeping Operations:
    - Create and Destroy Queue or Queue database
    - Start and Stop a queue
    - Modify Queue’s attributes
    - Perform Recovery
  - Monitoring:
    - Monitor load
    - Reporting and auditing

# Message-Queuing System

- System Interaction is done by Queues
- 'Relay'/Router nodes
  - No naming service in a queue system → Queue-to-location mapping in Q.Mgr.
  - Message transformation
  - Implement multicast
- Forwarding of messages between queues
  - transactional, to avoid lost messages
  - batch forwarding of messages → better throughput
  - can be implemented as an ordinary transaction server

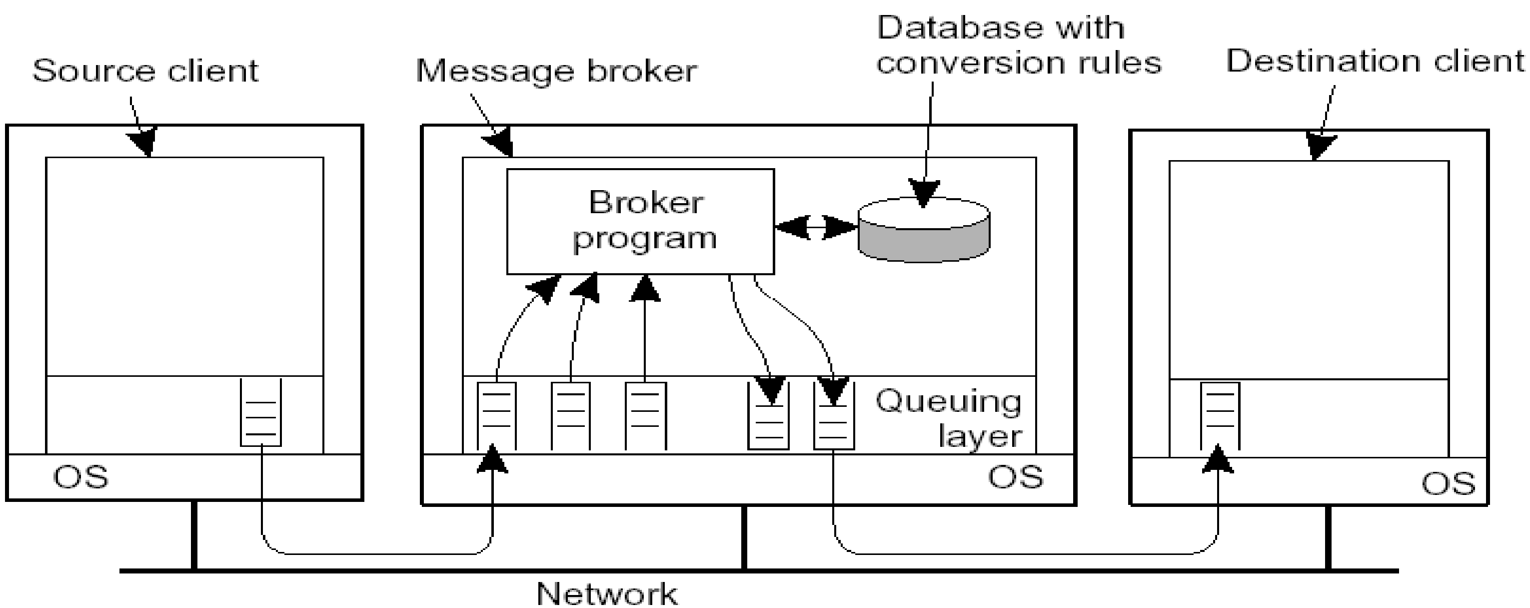





Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

# Message Brokers

- Relax the assumption of a common message format → different formats
- Message broker:
  - Centralized component
  - Transforms incoming messages to target format → heterogeneity
  - May provide subject-based routing capabilities
  - Acts very much like an application gateway





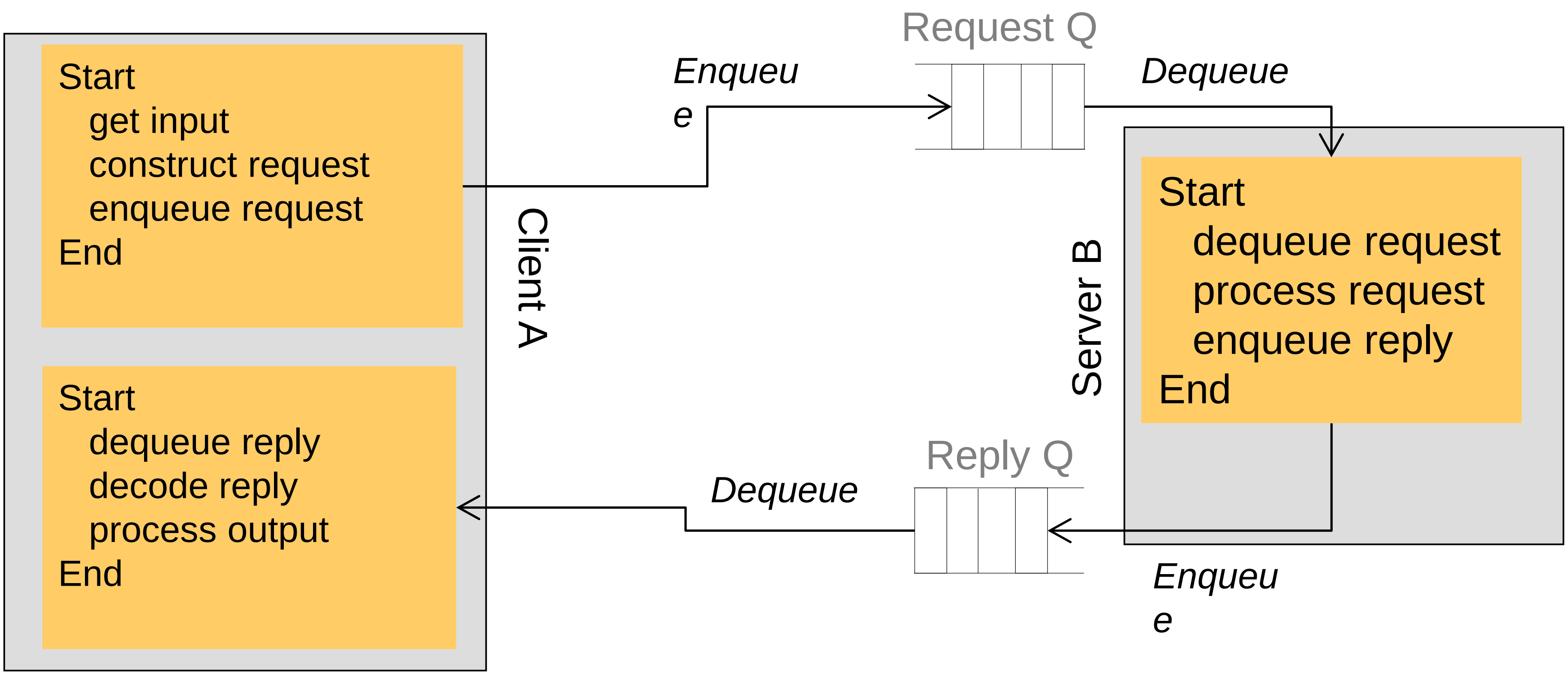

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

# Asynchronous Request/Reply, RPC


Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

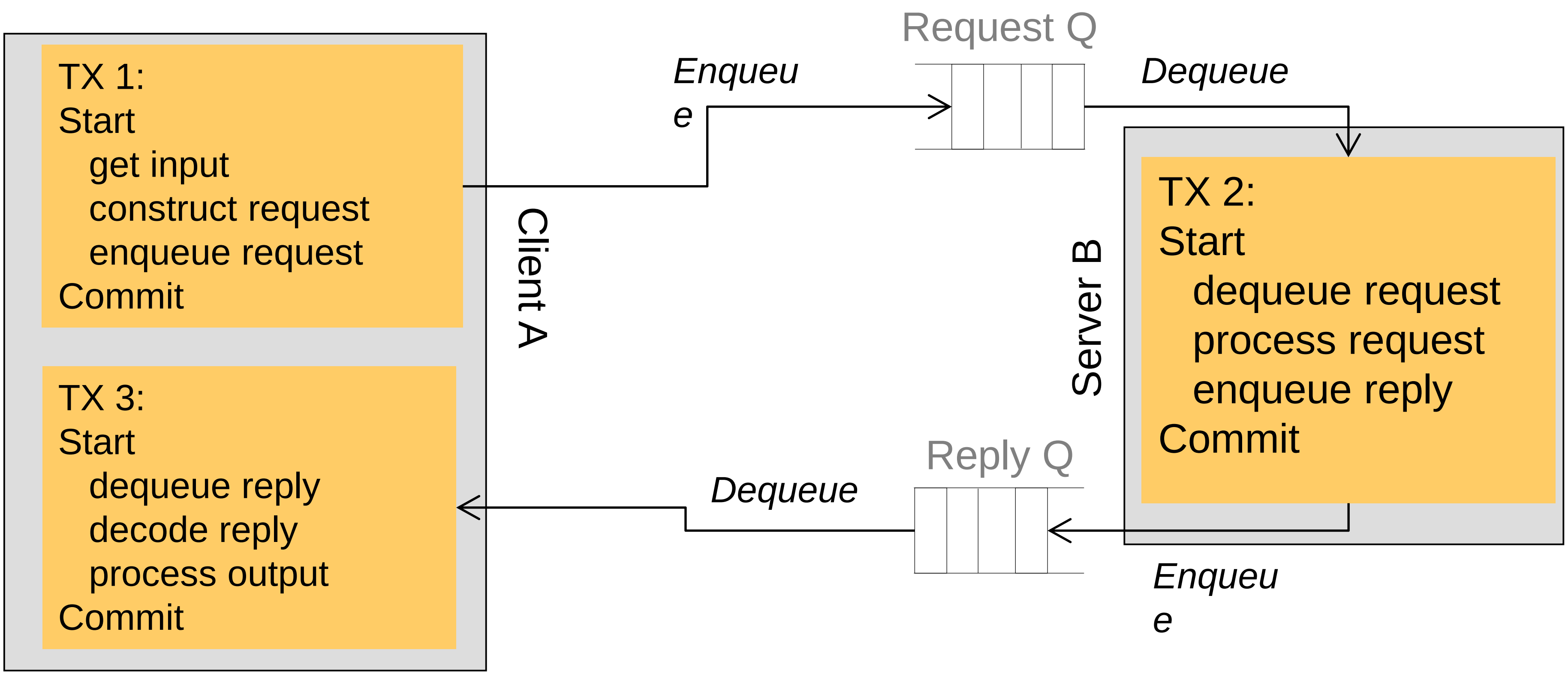
# Request/Reply with Queues

- Asynchronous request/reply can be realized with message queues



# Request/Reply with Queues

- Asynchronous request/reply can be realized with message queues
  - Reliability – transactional queues
  - Three Transactions



# Server's View of Queuing

- Assume each request for execution is a single transaction
- Server dequeues a request, executes the request, enqueues the result, and commits
- If the transaction aborts
  - the dequeue operation is undone
  - the enqueue operation is undone if already started
- If client checks queues, request is either in request queue, in process, or result in result queue

# Client's View of Queuing

- Client perceives three transactions for each request:
  - one transaction to enqueue request
    - receive input from user, construct request, enqueue request, commit
  - one server transaction (described before)
  - one transaction to dequeue results
    - dequeue reply from result queue, convert to proper output format, deliver output, commit (wiping out result in result queue)

# Cost/Benefit of Operating with Queues

- Using queues buys flexibility
  - communication with unavailable clients or servers
  - load balancing across servers
  - easy implementation of priorities
  - easier integration of legacy systems
- Using queues is expensive
  - 3 transactions instead of one
  - transactional queues must be managed by a (specialized) DBMS to guarantee persistence and transaction semantics
- **Question**: Can queues be used to implement transactional RPC?

# IBM MQseries


Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

# Queue Managers

- TP Monitors (TUXEDO, Encina, TOP END) offer(ed) queue managers
- Originally some standalone products (IBM's MQSeries most widely used)
- Integration into tool suites as part of Application Servers
  - IBM MQSeries → WebSphere Suite, Weblogic → BEA messageQ → Oracle Weblogic (Oracle Fusion), SUN JMQ → Oracle, …
- Standalone products
  - SonicMQ, ActiveMQ, Hornet Q, …
  - Initially tied to JMS, now integrating other protocols (REST, AMQP)

# IBM's MQSeries
# (now part of WebSphere Suite)

- MQSeries provides interoperable queue management across many Operating Systems
- works with all IBM TP monitors and any system supporting the X/Open XA interface (including CORBA OTS), Java connectivity included
- when working with a TPM, MQSeries uses the TPM transactions, otherwise it provides its own

# IBM MQSeries - concepts

- Basic concepts:
  - Application-specific messages are put into (MQPUT), and removed from (MQGET) queues
    - Queues managed by queue managers
    - Messages enqueued mainly in local queues ( or remote Q managed by Q Mgr.)
- multiple named queues supported
- queue forwarding among queues (e.g. for load balancing)
- queue forwarding occurs within channel agent's own transaction
- queue manager consists of
  - connection manager, data manager
  - lock manager, buffer manager
  - recovery manager, log manager
- pub/sub brokering possible



Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

# IBM MQSeries: Qs, API ...

- types of Qs
  - Local
  - Remote
- interaction through MQI verbs
  - MQBEGIN, MQCMIT
  - MQPUT, MQGET (browsing, consuming, blocking/non-blocking)
  - control operations
    - connect/disconnect Qmanager (MQCONN, MQDISC)
    - set configurations, manage Q processing (MQOPEN, MQSET, MQCLOSE)
- interaction through C++/ Java APIs
- interaction through JMS API

# IBM MQSeries - Messages

- messages can be
  - persistent
    - more secure, more expensive, logged, exactly once semantic
  - non-persistent
    - less secure, faster since in main memory, at most once semantic
- both types of messages can be enqueued in same queue
- message data
  - user defined format
  - default format and encodings

- Addresses – combination of:
  - Name of destination Queue manager – unique systemwide
  - Name of destination Queue (managed by dest. Queue manager)

# IBM MQSeries - Messages

- message consists of descriptor and data
- descriptor includes context
  - identity
  - origin
  - system message ID
  - application message ID
  - message type
    - datagram, request, reply, report
  - persistence flag
  - name of destination queue
  - ID of reply queue
  - correlation ID
  - expiry
  - application-defined format
  - report options
    - confirm on arrival, on delivery, on positive/negative action, on expiration, or on exception
  - priority

# IBM MQSeries – Message Transfer

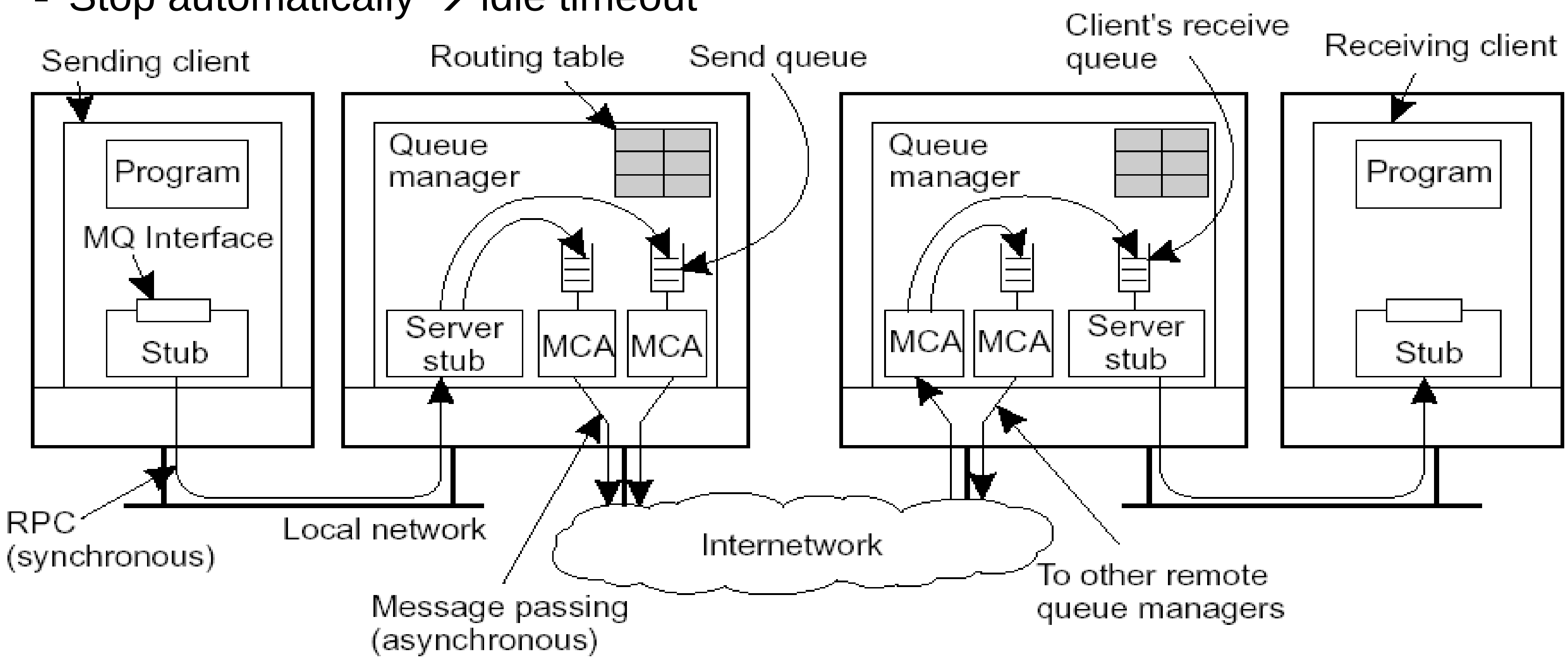
- Message transfer:
  - Messages are transferred between queues
  - Message transfer requires a channel
  - At each endpoint of channel is a Message Channel Agent (MCA)
    - Channels use lower-level network communication facilities (e.g., TCP/IP)
    - (Dis)Assemble messages into/from transport-level packets
    - Sending/receiving packets



Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

# IBM MQSeries - Channels

- Channels are unidirectional
- Each channel associated with exactly one send queue
  - Message transfer → sending and receiving MCA running. Alternatives:
    - Start sending MCA upon enqueue in sending queue
    - Remote start → send control message to a daemon
    - Stop automatically → idle timeout






Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

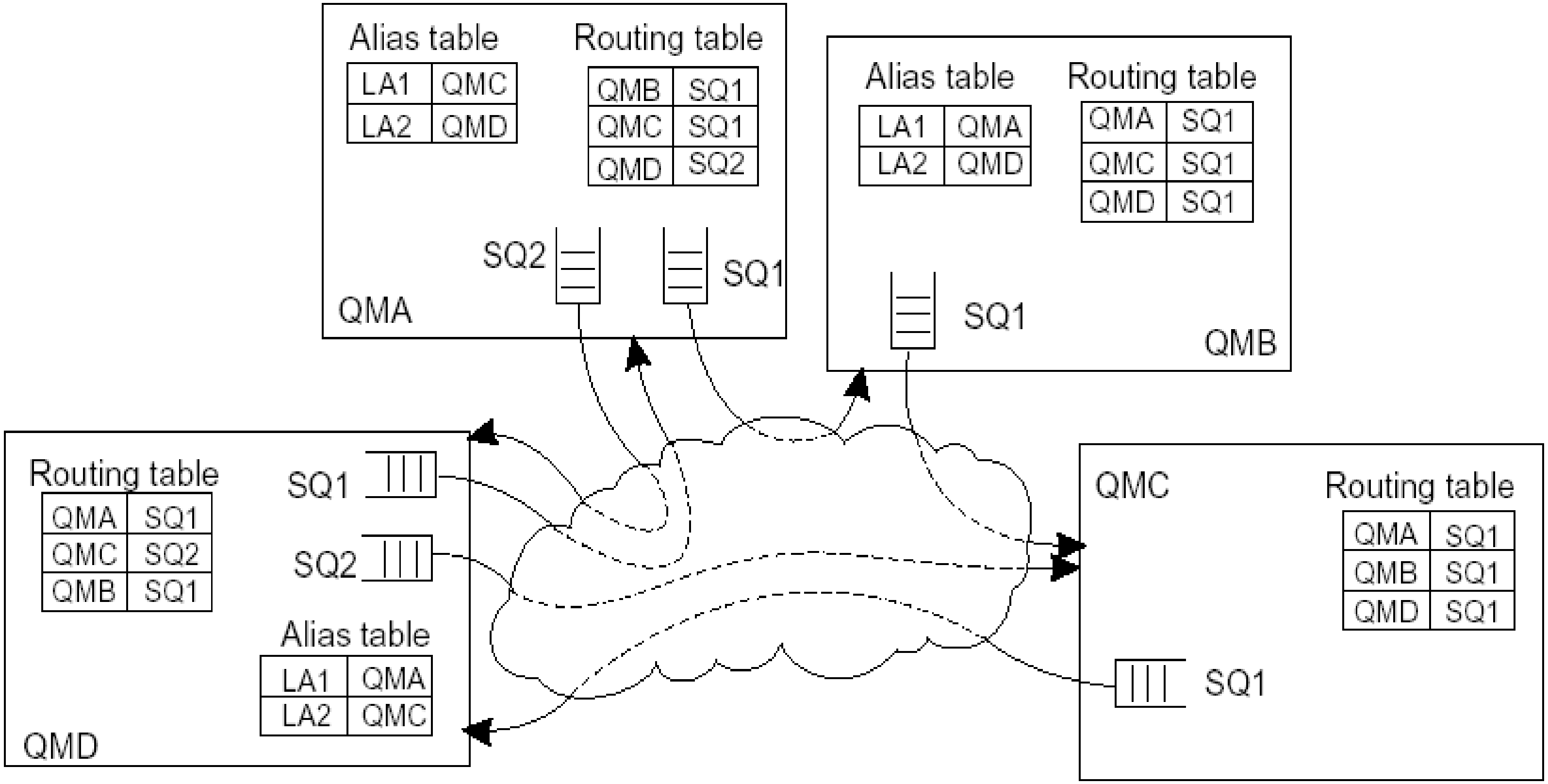
# IBM MQSeries – Message Transfer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Any network of queue managers can be created
  - routes are set up manually (system administration)
- Route must be specified, besides dest. address (Dest. Q.Mgr+dest.Queue)
  - Route: (local send queue; dest.address→dest.queue)
  - Routes stored in queue manager's routing table
  - If retransmission needed (intermediate queues) – message relayed by lookup in local routing table
    - Dest_queue = lookup( msg.header→dest_queue )
- Manage changes in queue manager's names
  - Use logical names → local aliases
  - Name change queue manager → change the name in ALL routing tables
  - Problem Solved: Location/Address transparency

# IBM MQSeries – Message Transfer

- **Routing:** By using **logical names**, in combination with name resolution to local queues, it is possible to put a message in a **remote queue**






Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

# Summary

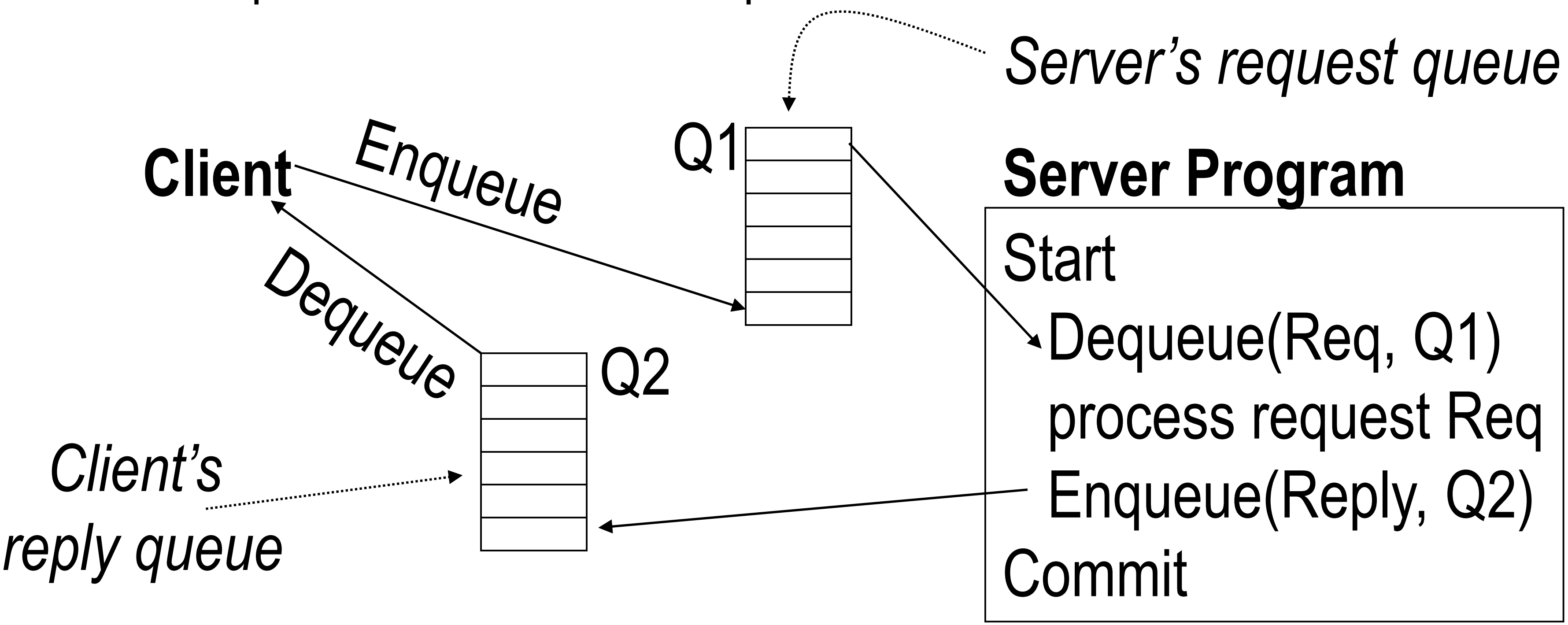
- Asynchronous communication
- Queues
- Queuing Systems
- Message-oriented communication
- Asynchronous RPC
- IBM MQ Series

# Transaction Processing with Queues


Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

# Transaction Semantics - Server View

- The queue is a transactional resource manager
- Server dequeues request within a transaction
- If the transaction aborts, the dequeue is undone, so the request is returned to the queue



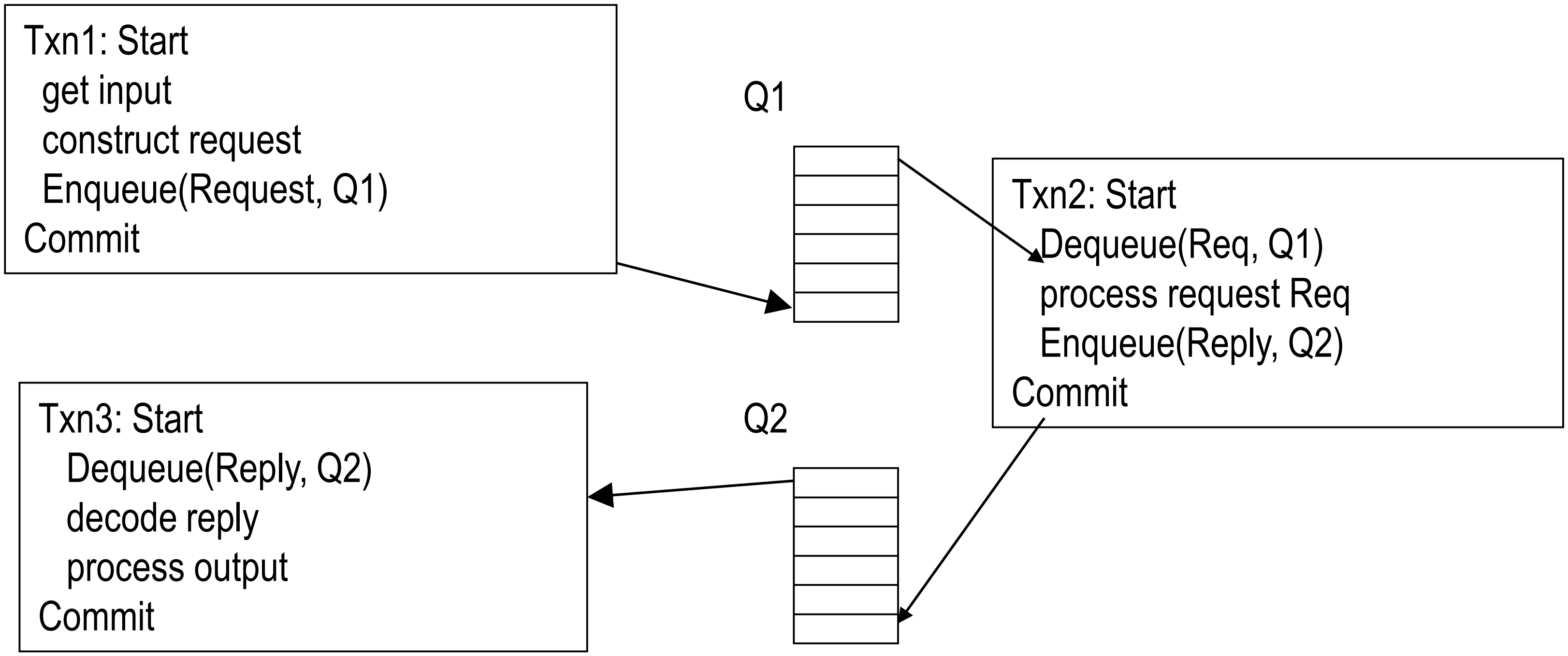
# Transaction Semantics - Server View

- Server program is usually a workflow controller
- It functions as a dispatcher to
  - get a request,
  - call the appropriate transaction server, and
  - return the reply to the client.
- Abort-count limit and error queue to deal with requests that repeatedly lead to an aborted transaction

# Transaction Semantics - Client View

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Client runs one transaction to enqueue a request and a second transaction to dequeue the reply



# Client Recovery

- If client fails and then recovers, a request R could be in one of 4 states:
  - A. Txn1 didn't commit – Local DB says R is NotSubmitted.
  - B. Txn1 committed but server's Txn2 did not – Local DB says R is Submitted and R is either in request queue or being processed
  - C. Txn2 committed but Txn3 did not – Local DB says R is Submitted and R's reply is in the reply queue
  - D. Txn3 committed – Local DB says R is Done

- To distinguish B and C, client first checks request queue (if desired) and then polls reply queue.

# Persistent Sessions



- Suppose client doesn’t have a local database that runs 2PC with the queue manager.

- The queue manager can help by persistently remembering each client’s last operation, which is returned when the client connects to a queue … amounts to a persistent session

# Client Recovery with Persistent Sessions

- Now client can figure out
  - A – if last enqueued request is not R
  - D – if last dequeued reply is R
  - B – no evidence of R and not in states A, C, or D.

- // Let R be id of client's last request
- // Assume client ran Txn0 for R before Txn1
- Client connects to request and reply queues;
- If (id of last request enqueued ≠ R) { resubmit request }
- elseif (id of last reply message dequeued ≠ R)
- { dequeue (and wait for) reply with id R }
- else // R was fully processed, nothing to recover

# Non-Undoable Operations

TECHNISCHE UNIVERSITÄT DARMSTADT

- How to handle non-undoable non-idempotent operations in txn3 ?

```
Txn3: Start
  Dequeue(Reply for R, Q2)
  decode reply & process output
  State(R) = "Done"
Commit
```

← Crash!

=> R was processed.
But Txn3 aborts.
So R is back on Q2.

- If the operation is undoable, then undo it.
- If it's idempotent, it's safe to repeat it.
- If it's neither, it had better be testable → compensation

# Testable Operations

- Testable operations
  - After the operation runs, there is a test operation that the client can execute to tell whether the operation ran
  - Typically, the non-undoable operation returns a description of the state of the device (before-state) and then changes the state of the device
  - the test operation returns a description of the state of the device.
  - E.g., State description can be a unique ticket/check/form number under the print head

# Thank You!

# Questions?