Using AQ to Access Java Services from PL/SQL

Philipp Salvisberg
Senior Principal Consultant

@phsalvisberg www.salvis.com/blog/

Agenda

1. Motivation
2. Message Queuing Essentials
3. Database Setup for JMS (Java Message Service)
4. Java AQ Listener Setup
5. Demo (incl. Synchronous Call from SQL, Monitoring)
6. Core Messages

Motivation

Why Calling Services Outside of the Database?

- PL/SQL limitations
- Oracle database server environment limitations
- Java components exist, but
  - required Java version is not supported within the database
  - loading java into the database is not appealing (e.g. large libraries with a lot of dependencies, performance impact of multi-threading restrictions, …)
- Save database server resources (license costs)

Why Queue Messages instead of HTTP Calls?

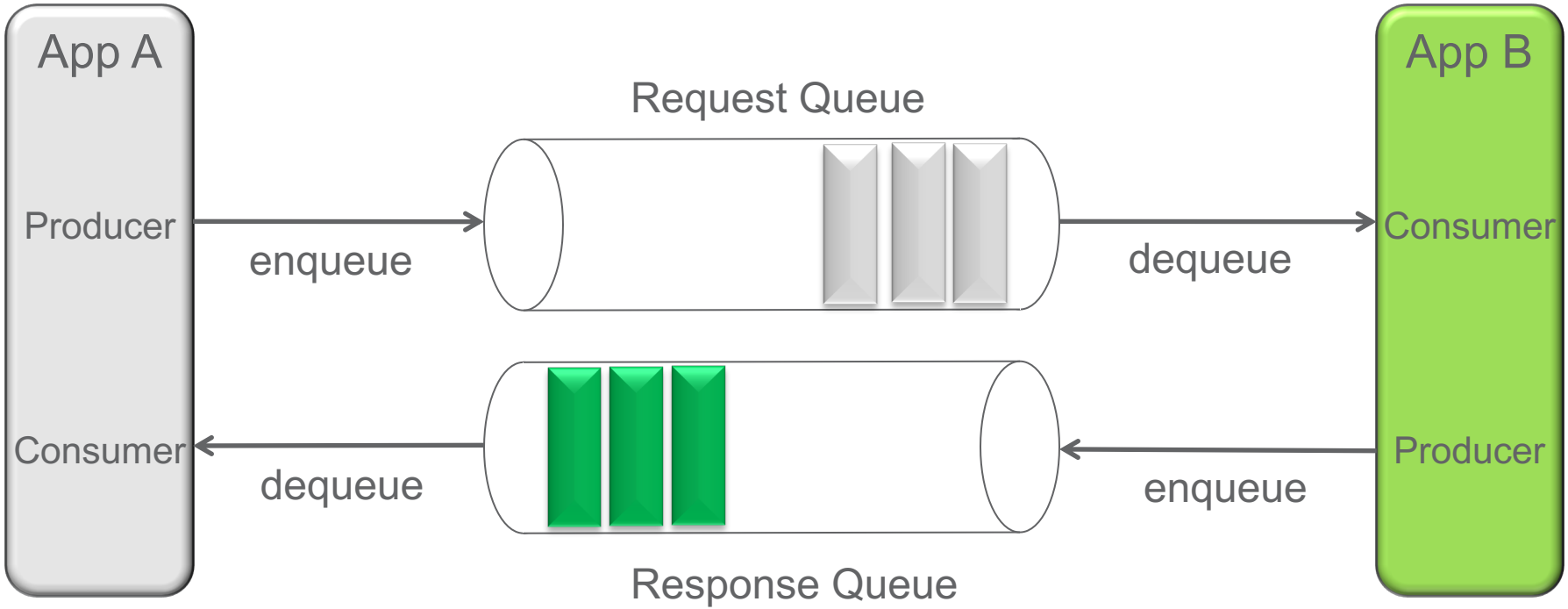
- Transactions – for enqueuer and dequeuer processes
- Asynchronous, concurrent execution to increase overall throughput
- Prioritization of service calls
- Using UTL_HTTP is not simpler, especially over SSL
- UTL_DBWS is challenging to install, and buggy (e.g. Bug 18705437 : UTL_DBWS REQUESS LARGER THAN 65599 ARE TRUNCATED)

Some Use Cases

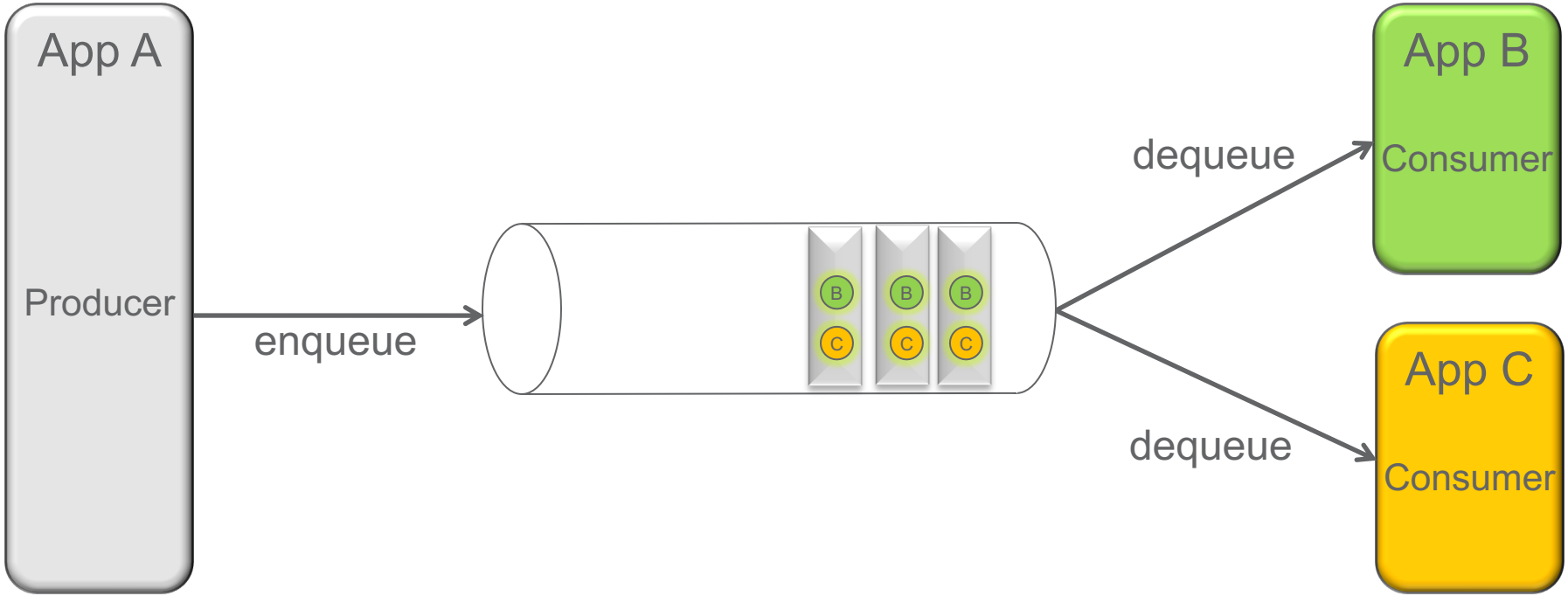
- Send E-Mail or SMS asynchronously
- Applying resource demanding optimization algorithms
- Synchronize MS Exchange items based on database changes
- Start various tasks from an Oracle APEX application
  - Database deployments
  - Automic (UC4) job network generation and deployment
  - Populating staging area tables from non Oracle database sources
- Create Reports (e.g. PDF) or Office documents (e.g. PowerPoint, Excel)

Message Queuing Essentials

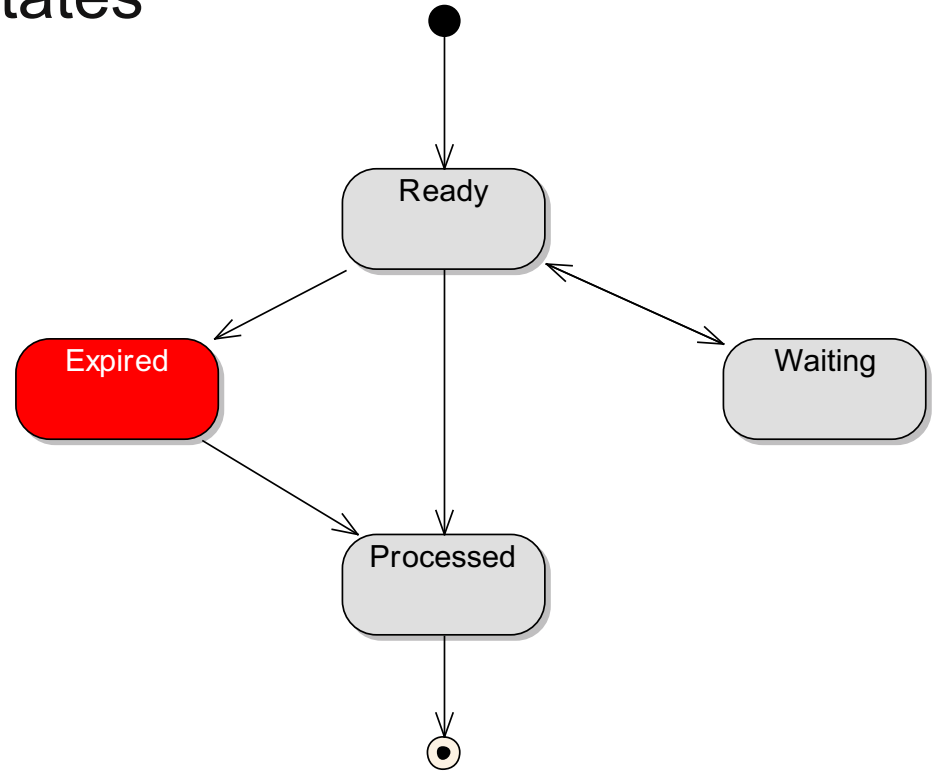
Single Consumer (Point-to-Point / Queue)



Multi-Consumer (Publish-Subscribe / Topic)



Message States*



* = additional sub-states exist, e.g. for ready: “in memory”, “deferred”, “spilled”, “deferred spilled”

Database Setup for JMS

Create Queue Table

```
BEGIN
   dbms_aqadm.create_queue_table (
      queue_table         => 'REQUESTS_QT'
     ,queue_payload_type  => 'SYS.AQ$_JMS_TEXT_MESSAGE'
     ,sort_list           => 'PRIORITY,ENQ_TIME'
     ,multiple_consumers  => TRUE
   );
END;
/
```

Oracle provides a generic JMS type and specific types for BytesMessage, MapMessage, ObjectMessage, StreamMessage, TextMessage

sys.aq$_jms_text_message ≈ javax.jms.TextMessage

- header — aq$_jms_header
  - replyto — sys.aq$_agent
    - name — varchar2(10)
    - address — varchar2(1024)
    - protocol — number
  - type — varchar(100)
  - userid — varchar(100)
  - appid — varchar(100)
  - groupid — varchar(100)
  - groupseq — int
  - properties — varray(100) of aq$_jms_userproperty
    - name — varchar(100)
    - type — int
    - str_value — varchar(2000)
    - num_value — number
    - Java_type — int

- text_len — int
- text_vc — varchar2(4000)
- text_lob — clob

Some member procedures/functions:

- set_text / get_text
- set_replyto / get_replyto
- set_string_property / get_string_property
- set_int_property / get_int_property
- set_long_property / get_long_property
- set_double_property / get_double_property

Create Queue

```
BEGIN
   dbms_aqadm.create_queue (
      queue_name           => 'REQUESTS_AQ'
     ,queue_table          => 'REQUESTS_QT'
     ,max_retries          => 0
     ,retry_delay          => 5 -- 5 seconds
     ,retention_time       => 60*60*24*7 -- 1 week
   );
END;
/
```

Start Queue

```
BEGIN
   dbms_aqadm.start_queue(
      queue_name => 'REQUESTS_AQ'
     ,enqueue    => TRUE
     ,dequeue    => TRUE
   );
END;
/
```

Enqueue Test

```
DECLARE
   l_enqueue_options sys.dbms_aq.enqueue_options_t;
   l_message_props   sys.dbms_aq.message_properties_t;
   l_jms_message     sys.aq$_jms_text_message := sys.aq$_jms_text_message.construct;
   l_msgid           RAW(16);
BEGIN
   l_message_props.correlation := sys_guid;
   l_message_props.recipient_list(1) := sys.aq$_agent('APP_B', 'REQUESTS_AQ', 0);
   l_jms_message.set_text('Hello App B');
   dbms_aq.enqueue(queue_name         => 'REQUESTS_AQ',
                   enqueue_options    => l_enqueue_options,
                   message_properties => l_message_props,
                   payload            => l_jms_message,
                   msgid              => l_msgid);
   COMMIT;
END;
/
```

Dequeue Test

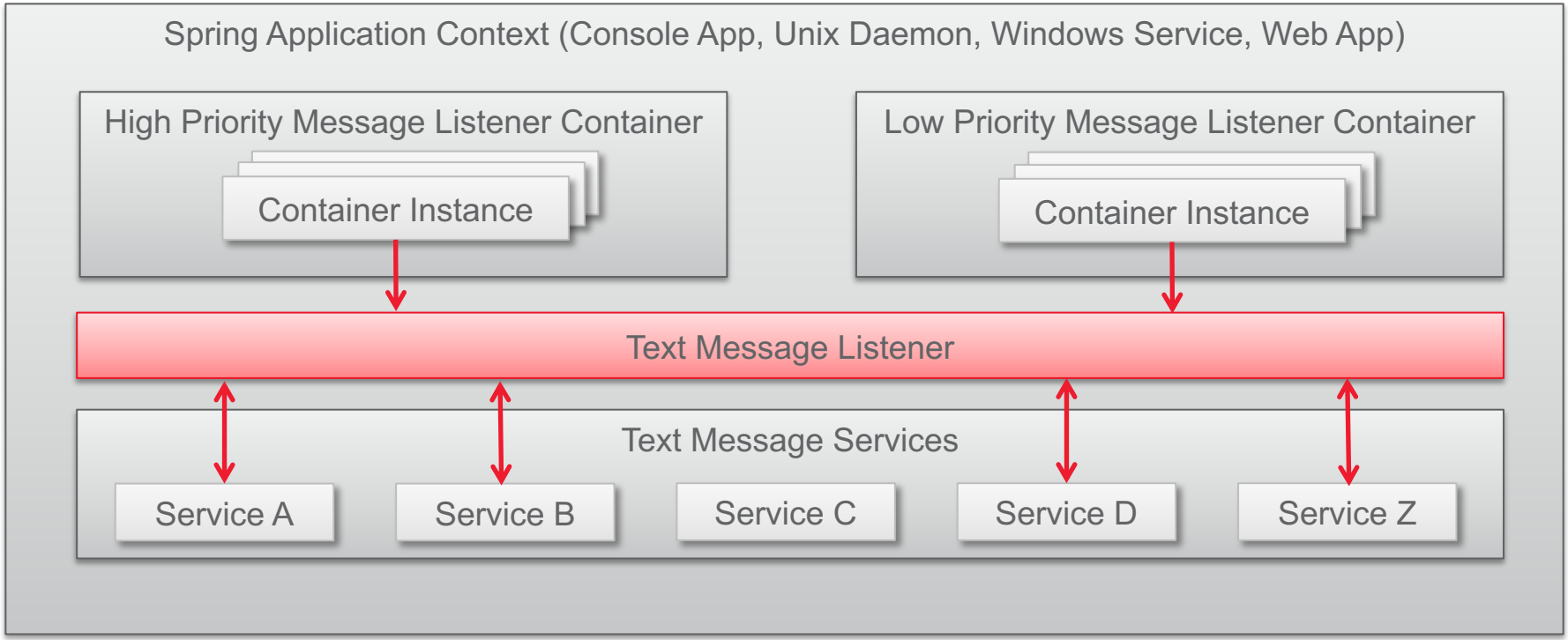
```
…  LOOP
      l_dequeue_options.consumer_name := 'APP_B';
      l_dequeue_options.navigation    := sys.dbms_aq.first_message;
      l_dequeue_options.wait          := 5;
      BEGIN
         dbms_aq.dequeue(queue_name         => 'REQUESTS_AQ',
                         dequeue_options    => l_dequeue_options,
                         message_properties => l_message_props,
                         payload            => l_jms_message,
                         msgid              => l_msgid);
         l_jms_message.get_text(l_text);
         dbms_output.put_line(l_text);
         COMMIT;
      EXCEPTION
         WHEN e_no_msg THEN
            EXIT;
      END;
   END LOOP; …
```

Java AQ Listener Setup

Wanted Solution Properties

- Robust
  - Shutdown of the database does not require restart of the Java application
- Reliable
  - Request and response are processed in the same database transaction
  - A message is never processed twice
  - Unprocessed messages are logged
- Simple
  - Adding a new service is as easy implementing a single class without configuration changes
- Fast
  - Processing around 100 messages per second
  - Adding additional consumer processes (threads) is a matter of configuration
  - Long-running low priority calls never block high priority calls

Architecture Overview



Interface TextMessageService

```
public interface TextMessageService {
   public void process(TextMessage request, TextMessage response);
}
```

Service Implementation

```
@Service("PrintTextService")
@Scope("prototype")
public class PrintTextService implements TextMessageService {

   private final Logger logger = Logger.getLogger(PrintTextService.class);

   @Override
   public void process(TextMessage request, TextMessage response) {
      String payload = null;
      try {
         payload = request.getText();
      } catch (JMSException e) {
         throw new RuntimeException("cannot extract payload from request message.");
      }
      logger.info("printing request payload: " + payload);
   }
}
```

MessageListenerContainer – Powerful!

```
@Configuration
public class AppConfig {
   @Bean
   public DefaultMessageListenerContainer highPriorityJmsContainer() {
      DefaultMessageListenerContainer cont = new DefaultMessageListenerContainer();
      cont.setMessageListener(new TextMessageListener());
      cont.setConnectionFactory(connectionFactory());
      cont.setDestinationName("aqdemo.requests_aq");
      cont.setPubSubDomain(true);
      cont.setSubscriptionName("Java_High_Priority");
      cont.setSubscriptionDurable(true);
      cont.setMessageSelector("(JMSPriority IN (1,2) and appName = 'Java')");
      cont.setSessionAcknowledgeMode(Session.SESSION_TRANSACTED);
      cont.setSessionTransacted(true);
      cont.setConcurrency("1-4");
      cont.setMaxMessagesPerTask(1);
      return cont;
   }
}
```

TextMessageListener – Handles all Messages


```
@Component
public class TextMessageListener implements SessionAwareMessageListener<TextMessage> {
   public void onMessage(final TextMessage request, final Session session) {
      TextMessageService service;
      String messageId = null;
      try {
         messageId = request.getJMSMessageID();
         TextMessage response = null; ...
         String beanName = request.getStringProperty("beanName");
         if (beanName != null) {
            service = ctx.getBean(request.getStringProperty("beanName"),
                           TextMessageService.class);
            service.process(request, response);
            if (response != null) {...}
         } ...
      } catch (JMSException e) {...}
   }
}
```

Demo

(incl. Synchronous Call from SQL, Monitoring)

Demo

- Single message (demo1.sql)
- 200 messages (demo2.sql)
- Mixed workload – fast and slow services (demo3.sql, demo4.sql)
- Processing response message asynchronously (demo5.sql)
- Processing response message synchronously from SQL (demo6.sql)
- Monitoring queues (demo7.sql)

complete source code is available on https://github.com/PhilippSalvisberg/aqdemo

Function for Synchronous Message Processing

```
CREATE OR REPLACE FUNCTION get_prime_fact(in_number  IN INTEGER,
                                          in_timeout IN INTEGER DEFAULT 30)
   RETURN VARCHAR2 IS
   PRAGMA AUTONOMOUS_TRANSACTION;
   l_correlation VARCHAR2(128);
   --
   FUNCTION enqueue(in_number IN INTEGER, in_timeout IN INTEGER) RETURN VARCHAR2 IS
      ...
   END enqueue;
   --
   FUNCTION dequeue(in_correlation IN VARCHAR2, in_timeout IN INTEGER)
      RETURN VARCHAR2 IS
      ...
   END dequeue;
BEGIN
   l_correlation := enqueue(in_number, in_timeout);
   RETURN dequeue(l_correlation, in_timeout);
END get_prime_fact;
```

Function for Synchronous Message Processing

Query

```
 SELECT rownum                              AS input_number,
        aqdemo.get_prime_fact(ROWNUM) AS prime_factorization
   FROM dual
CONNECT BY rownum <= 100;
```

Result

```
INPUT_NUMBER PRIME_FACTORIZATION
------------ ----------------------------------------
           1
…         99 3,3,11
          100 2,2,5,5
```

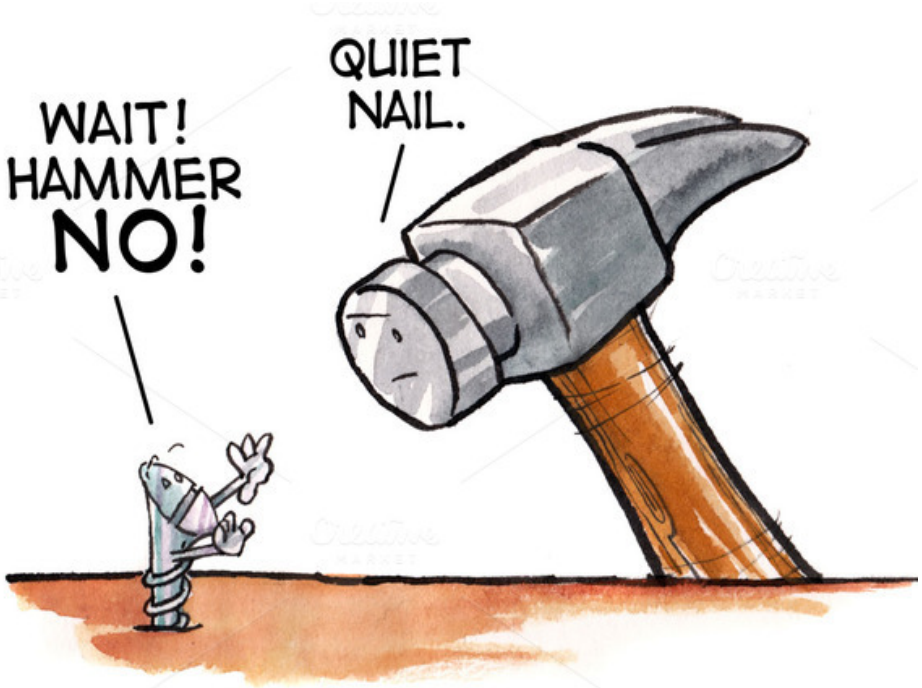
Monitoring

- Configure queues with a retention time (e.g. 1 day or 1 week)
- Query the provided AQ$ views, e.g. for MSG_STATE != 'PROCESSED'
- Query AQ$<queue_table_name>_R, USER_RULE_SETS, USER_RULE_SET_RULES and USER_RULES to check subscription filter criteria
- Create convenience views
  - Subset of columns provided in queue tables, views
  - Provide the content of AQ$_JMS_TEXT_MESSAGE as Oracle primitive data types
    - Text message
    - User properties
  - See MONITOR_REQUESTS_V and MONITOR_RESPONSES_V

Core Messages

Just Because You Can Doesn't Mean You Should

- Consider carefully what belongs into an Oracle Database
- Oracle AQ is a cost free option and part of every database edition – just use it



Happy Queuing

- Make the granularity of a message a part of your design considerations, that's the key to improve overall messaging performance
- Use dedicated queue tables and queues for topics with high throughput requirements
- Messages are processed row by row, contention issues are not unusual, mitigation is similar to conventional tables (e.g. increase PCTFREE)
- Just be aware that AQ has a limited scalability

Questions and answers…

Philipp Salvisberg
Senior Principal Consultant

Tel. +41 58 459 52 31
philipp.salvisberg@trivadis.com

Limited Scalability of AQ

@phsalvisberg

www.salvis.com/blog/

trivadis
makes IT easier.