Kapitel 3

Message-Oriented Middleware (MOM)

Inhalt

- Transaktionale Warteschlangen in TP-Monitoren
- Message Queuing
  - Beispiel: JPMQ (Java Persistent Message Queuing)
  - Nutzung im Workflow-Management
- Message Brokering
- Zusammenfassung

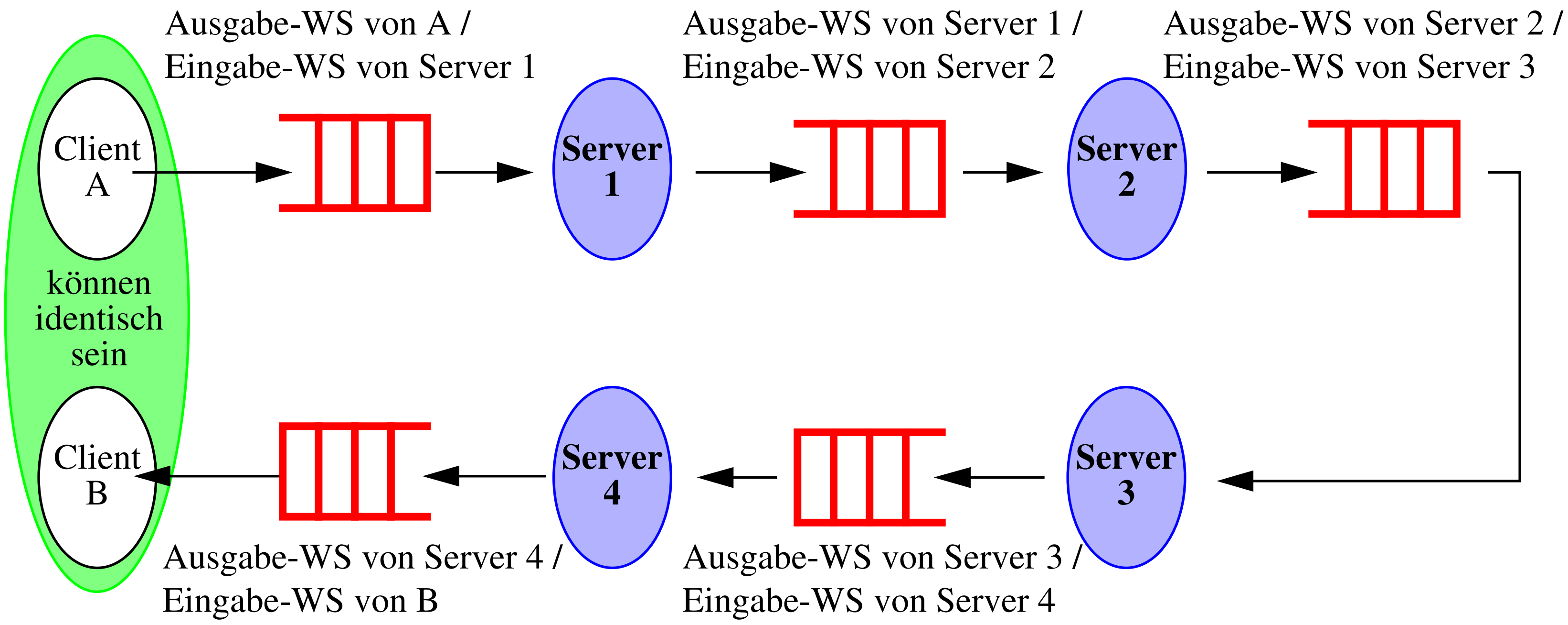
Transaktionale Warteschlangen in TP-Monitoren (1)

- ❑ Zwei Arten von Transaktionen
  - ➪ direkt (typisch für interaktive AW)
  - ➪ asynchron (erfordert *Queueing*)

- ❑ Einsatz von Warteschlangen
  - ➪ Lastkontrolle
    - Normalisierung von Lastspitzen
    - statt Erzeugung eines weiteren Prozesses wird Auftag in flüchtiger Schlange (vor der Server-Klasse) vermerkt
  - ➪ Endbenutzerkontrolle
    - Aushändigung der Ausgabe von asynchronen Vorgängen ist kritisch (Text, Bilder, Geld, ...)
    - Wiederholung der Ausgabe kann erforderlich werden, solange der Empfang nicht explizit quittiert wurde
  - ➪ Wiederherstellbare Dateneingabe
    - AW, die durch Eingabenachrichten hoher Frequenz getrieben werden und für hohen Durchsatz ausgelegt sind
    - Eingabe wird von einer Warteschlange gelesen
    - Eingabe darf nicht verloren gehen; auch nicht bei Crash

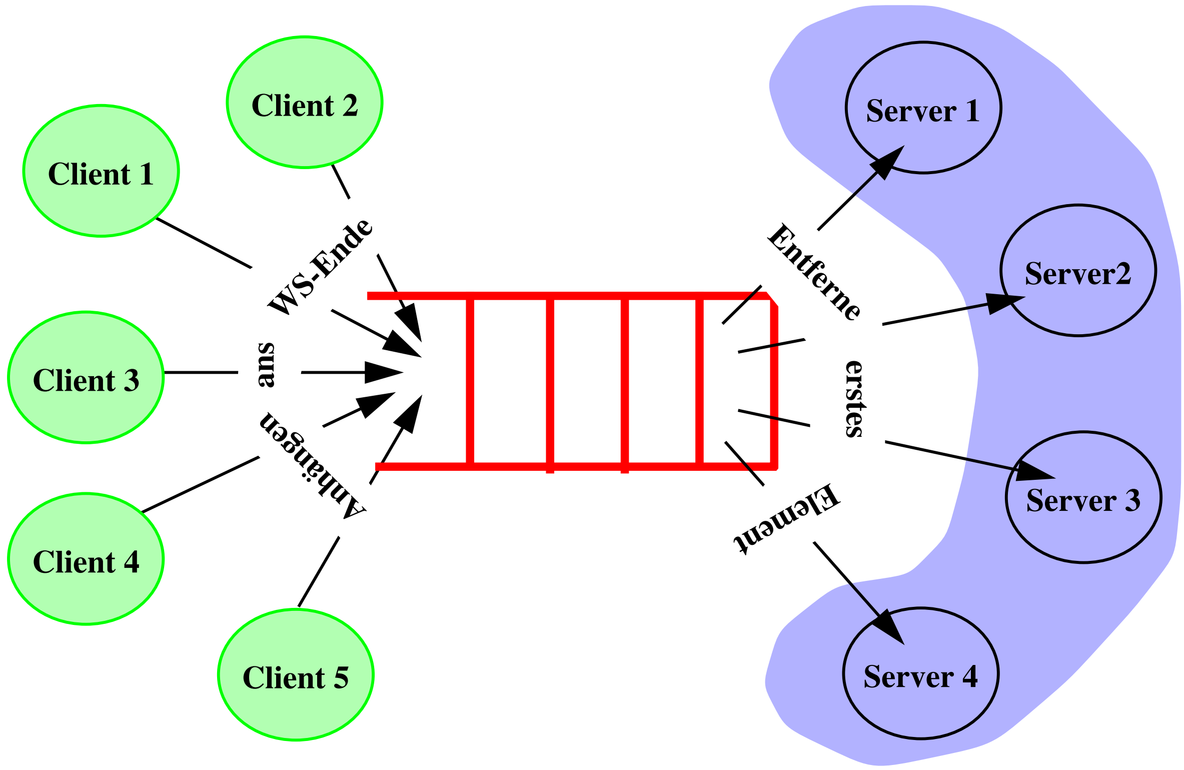
Transaktionale Warteschlangen in TP-Monitoren (2)

- Einsatz von Warteschlangen (Forts.)
  - *Mehrtransaktionen-Anforderungen*
    - Beispiel: synchrone Verarbeitung von aufeinanderfolgenden Transaktionen - basierend auf wiederherstellbaren Eingabedaten -, die auf hohen Durchsatz ausgelegt sind



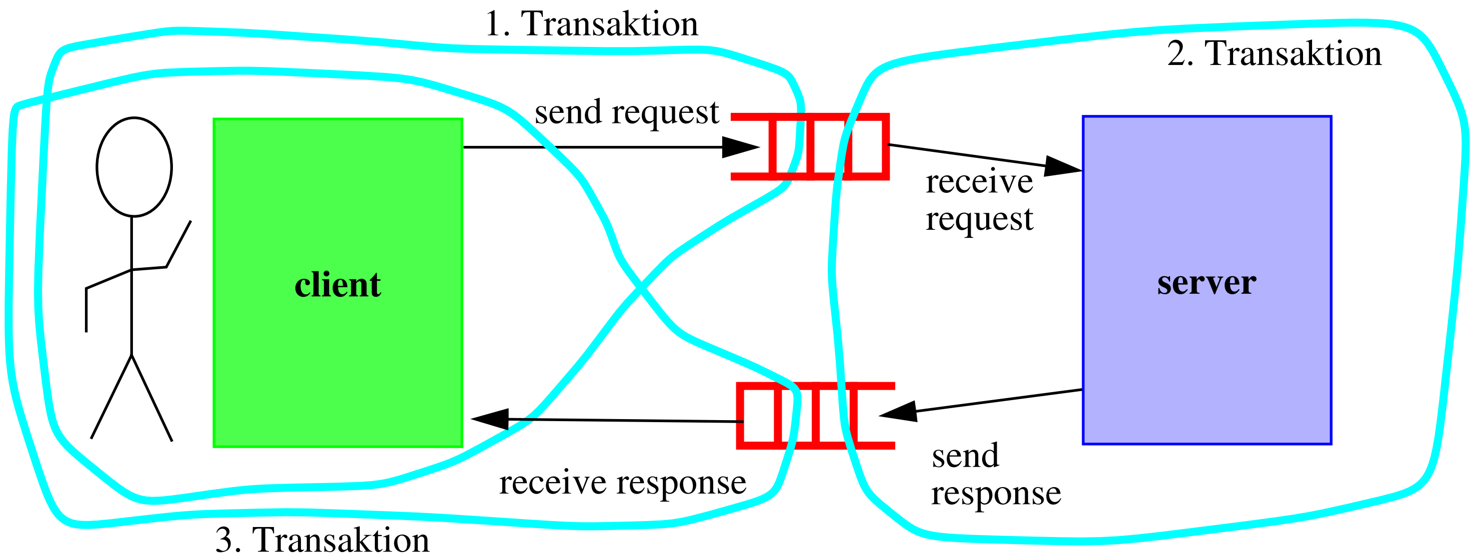
Transaktionale Warteschlangen in TP-Monitoren (3)

- ❑ Zwei Typen von WS
  - ➪ flüchtig für direkte TA
  - ➪ dauerhaft (persistent, recoverable) für asynchrone TA
- ❑ Flüchtige, Server-Klassen-spezifische WS
  - ➪ Möglichkeiten der Behandlung von Überlast
    - Client bekommt RetCode “Überlast” und entscheidet (Warten - Wie?, Rollback)
    - Flüchtige WS “vor” Server-Klasse
    - es muß “nur” “exactly-once” zugesichert werden; Dequeue muß nicht Teil der Server-TA sein



Transaktionale Warteschlangen in TP-Monitoren (4)

❏ Dauerhafte WS für asynchrone Transaktionsverarbeitung



➪ Entkopplung von Eingabe, Verarbeitung und Ausgabe

- Einsatz von drei oder mehr separaten TA
- Durchsatzoptimierung anstelle von Antwortzeitminimierung
- Anforderungs- und Ergebnis-WS sind dauerhafte Objekte
- Ein- und Auslagern muß jeweils Teil der zugehörigen TA sein

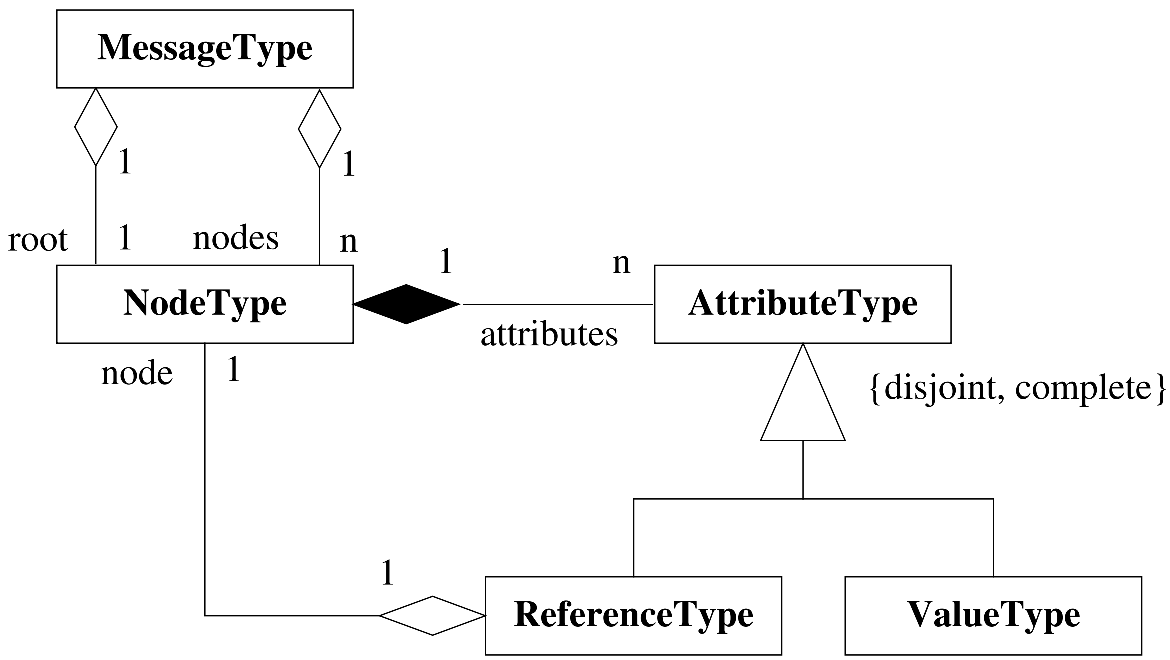
JPMQ[1] (1)

- QS (Queuing Systems) im allgemeinen bieten:
  - Bereitstellung persistenter Warteschlangen
    - zur Abwicklung asynchroner Kommunikation
    - als zuverlässige Nachrichtenpuffer
  - Grundlegende Operationen:
    - **Enqueue**:
      - Einlagern von Nachrichten in eine Warteschlange
      - Garantie der persistenten Speicherung
    - **Dequeue**:
      - (Multiples) Auslagern von Nachrichten
      - Garantie von "exactly once"
- Neben allgemeinen bietet JPMQ[2] spezielle Dienste:
  - strukturierte Nachrichtentypen
  - inhaltsbasierte Selektion von Nachrichten
  - anwendungsspezifische Sortierung von Nachrichten
  - automatische Zuweisung inhalts-basierter Prioritäten
  - flexible Transaktionssemantik

1. Steiert, H.-P., Zimmermann, J.: JPMQ - An Advanced Persistent Message Queuing Service, Proc. 16th British Nat. Conf. on Data Management (BNCOD16), LNCS 1405, Springer, 1998, pp. 1-18.
2. JPMQ steht für "Java Persistent Message Queuing".

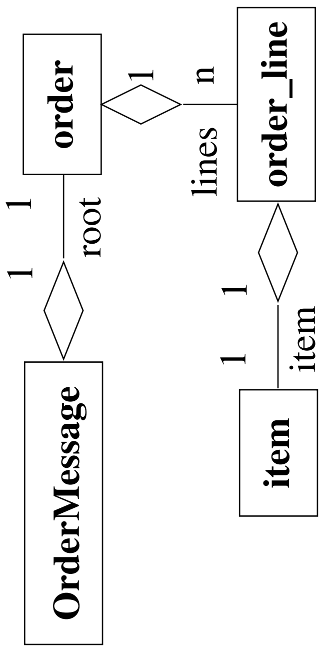
JPMQ (2)

- Message-Types
  - Jede Nachricht muß einen Typ haben
  - Modellierungselemente von Message Types
    - Knoten mit Attributen
    - Referenzen zur Verbindung von Knoten
    - Message Type selbst
  - Knoten und Referenzen bilden gerichteten, azyklischen Graphen mit einem ausgezeichneten Wurzelknoten
  - Message Types werden auf RDBMS-Schema abgebildet
  - Meta-Modell



JPMQ (3)

- Message-Types (Forts.)
  - Beispiel



Order			
Name:	Hans-Peter Steiert		
Address:	37th Street		
City:	Somewhere		
Date:	23.02.1998		
No.:	Amount	Item	
1	50	4711	apples
2	10	3478	orange juice
...	...	...	...

```
DEFINE MESSAGETYPE OrderMessage;
   NODE order;
      id:            IDENT;
      name:          VARCHAR(50);
      address:       VARCHAR(50);
      city:          VARCHAR(50);
      date:          DATE;
      lines:      REFERENCE TO order_line;
   END;
   NODE order_line;
      line_no:       INTEGER;
      amount:        INTEGER;
      price:         REAL;
      item:        REFERENCE TO item;
   END;
   NODE item;
      id:          IDENT;
      name:        STRING;
      ean:         INTEGER;
   END;
   ROOT order;
END;
```

JPMQ (4)

- Operationale Granulate

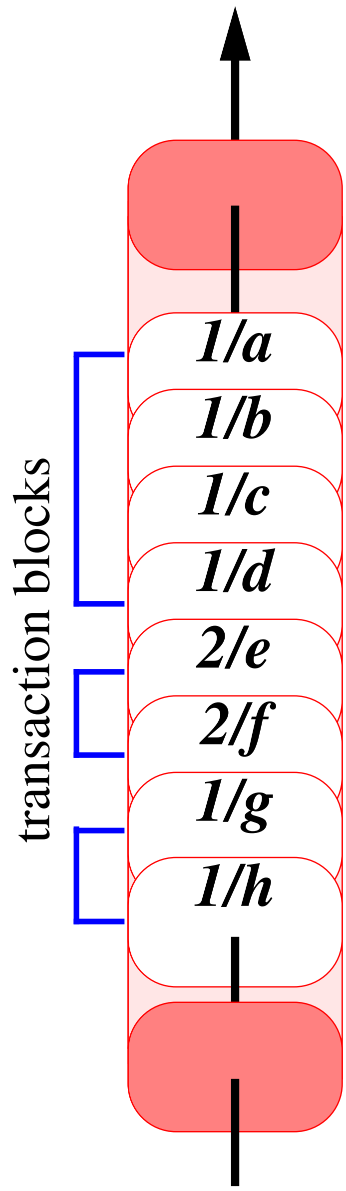
Granulat	Isolations-eigenschaft	
Operation	*recoverable*	*“exactly once” für Enqueue und Dequeue von Nachrichten*
Transaktion	*fifo (ta)*	*Nachrichten, die von derselben Transaktion eingelagert (Enqueue) wurden, werden in FIFO Reihenfolge ausgelagert (Dequeue)*
	blocked (ta)	*fifo (ta) + Nachrichten, die von derselben Transaktion eingelagert wurden, werden als Folge ausgelagert*
Connection	*fifo (c)*	*Nachrichten, die von derselben Connection eingelagert wurden, werden in FIFO Reihenfolge ausgelagert*
	blocked (c)	*fifo (c) + Nachrichten, die von derselben Connection eingelagert wurden, werden als Folge ausgelagert*

JPMQ (5)

- Transaktionen (Forts.)
  - Menge von queue Operationen (möglicherweise mehrere Queues betreffend) können als ***atomare*** Einheit betrachtet werden (queue transaction)
  - ***Dauerhaftigkeit*** ist notwendig, um ‘exactly once’ Semantik zu gewährleisten
  - ***Isolation***ssemantik pro Queue ‘einstellbar’
    - isolation level: “**none**”
      - Operationen sind ‘recoverable’
    - isolation level: “**ordered**“
      - fifo(c)
    - isolation level: “**blockwise**”
      - blocked(ta) + TA Grenzen (blocks) des Schreibers müssen vom Leser beachtet werden
    - isolation level: “**ordered blockwise**”
      - fifo(c) + isolation level “blockwise”
  - ***Konsistenz***
    - Bedingungen (Prädikate) auf Nachrichteninhalten

JPMQ (6)

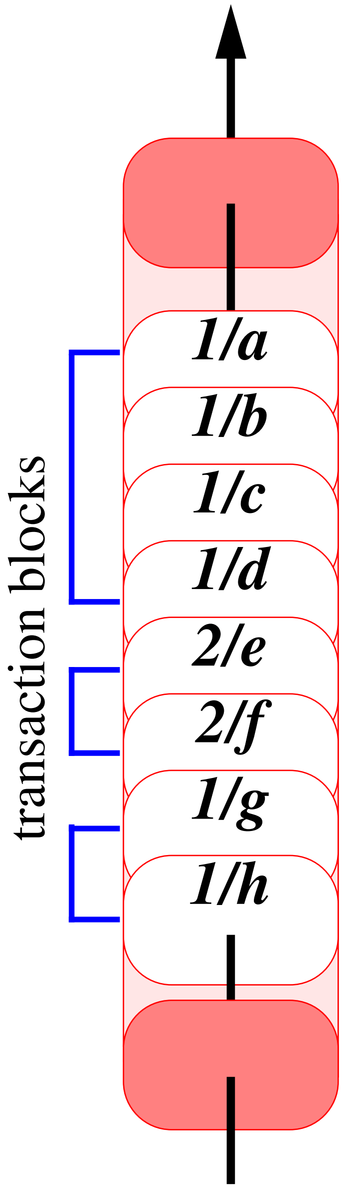
- Transaktionen (Forts.)
  - isolation level “**none**”



Leser1:	***Leser2***
d1[0] = 1/a	
d1[1] = 1/b	
	d2[0] = 1/c
d1[2] = 1/d	
	d2[1] = 2/e
	abort!
d1[3] = ?	

JPMQ (7)

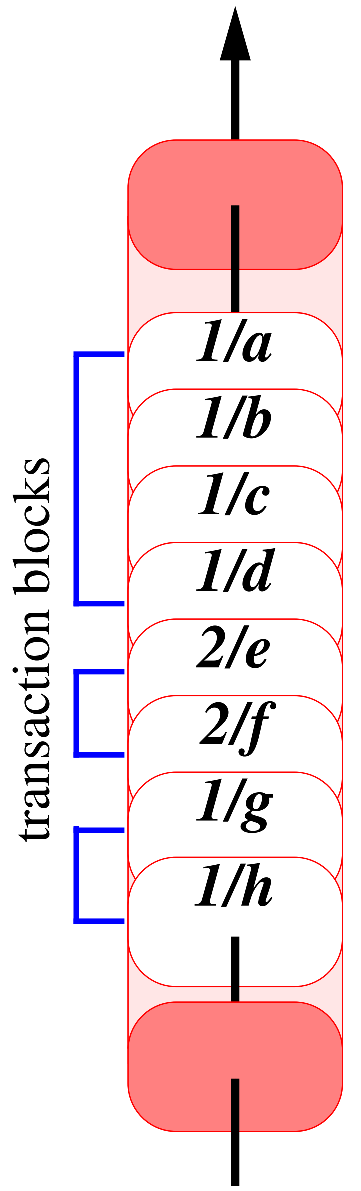
- Transaktionen (Forts.)
  - isolation level “**ordered**”



Leser1:	***Leser2***
d1[0] = 1/a	
d1[1] = 1/b	
	d2[0] = 2/e
d1[2] = 1/c	
	d2[1] = 2/f
	abort!
d1[3] = ?	

JPMQ (8)

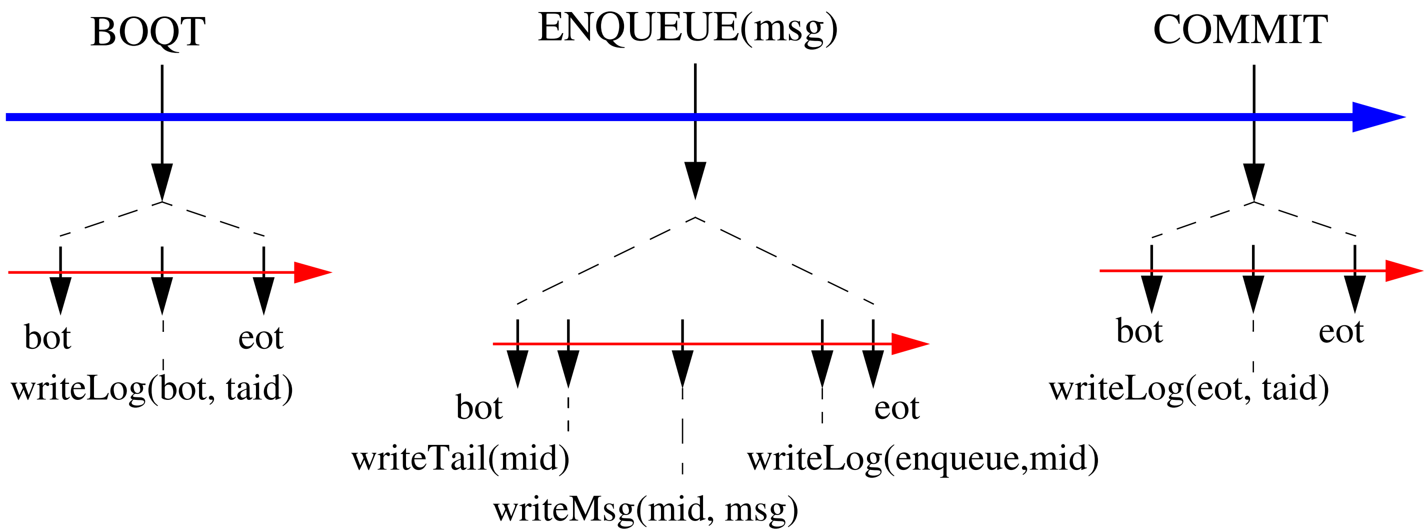
- Transaktionen (Forts.)
  - isolation level “**blockwise**”



Leser1:	***Leser2***
d1[0] = 1/a	
d1[1] = 1/b	
	d2[0] = 2/e
d1[2] = 1/c	
	d2[1] = 2/f
d1[3] = 1/d	
commit!	
	d2[2] = 1/g
	commit!

JPMQ (9)

- Transaktionen (Forts.)
  - Implementierung



- RDBMS-Transaktionen zu strikt
- jede JPMQ-Operation wird auf genau eine RDBMS-Transaktion abgebildet (DB ist immer ‘operations-konsistent’)
- Transaktionsmodell ähnlich zu “Open-Nested-Transactions”

JPMQ (10)

- Selektive Dequeue
  - Erweiterte Dequeue Operation:
    - Dequeue Operation unterstützt inhalts-basierte Selektion
    - Selektionskriterium (**SC**) kann dynamisch spezifiziert werden (OQL-ähnliche Anweisung)
    - Evaluation des Selektionskriteriums durch RDBMS
  - Beispiel

dequeue the next order of customer “Smith”

```
orders = outstandingOrdersQueue.dequeue
                        ( “ORDER”,
                          “select om
                           from mt_OrderType om, om.orders o
                           where o.name = ‘Smith’” );
```

message type

SC

JPMQ (11)

- Ordered Dequeue
  - Erweiterte Dequeue Operation:
    - Dequeue Operation unterstützt inhalts-basierte Sortierung von Nachrichten
    - Ordering function (**OF**) wird als OQL-ähnliche Anweisung spezifiziert
    - automatische Zuweisung von inhalts-basierten Prioritäten bei Enqueue
    - OF wird durch RDBMS bei Dequeue ausgewertet
    - Priority calculation function (**PCF**) wird als OQL-ähnliche Anweisung spezifiziert
    - PCF wird durch RDBMS bei Enqueue ausgewertet
  - Beispiel: *dequeue the order of customer "Smith" which includes the most items*

```
orders = OrdersQueue.dequeue
    ("ORDER",
     "count ( select  ol.items
              from    mt_OrderType om, order o, o.orderline ol )",
     "select  om
      from    mt_OrderType om, om.orders o
      where   o.name = 'Smith'" );
```

OF

JPMQ (12)

- Query Transformation
  - Beispielanfrage (Selective Ordered Dequeue:)

```
select     m1.mid as mid, of.sort1 as sort1
from       mt_order m1, (<OF>) as of
where      m1.mid = of.mid and m1.mid in (<SC>)
order by   sort1
```

  - Ersetzung von **<OF>** durch

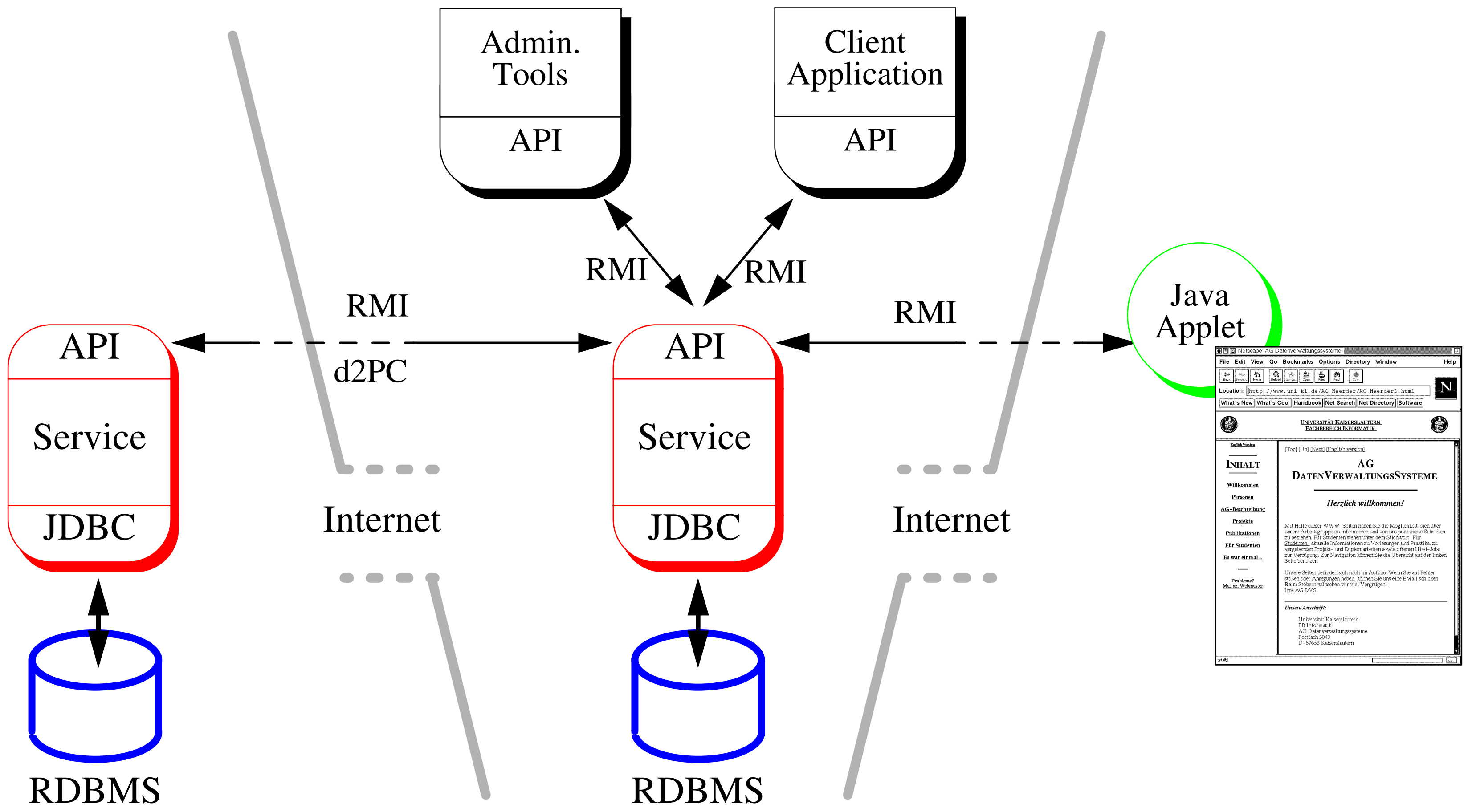
```
select     m2.mid as mid, count (i1.oid) as sort1
from       mt_order m2, orders o1, ref_order_orderline ref1,
           orderline ol1, ref_orderline_item ref2, item i1
where      m2.mid = o1.mid and o1.oid = ref1.order
           and ref1.orderline = ol1.oid
           and ol1.oid = ref2.orderline and ref2.item = i1.oid
group by   m2.mid
```

  - Ersetzung von **<SC>** durch

```
select     m3.mid
from       mt_order m3, order o2
where      m3.mid = o2.mid and o2.name = 'Smith'"
```

JPMQ (13)

- System Architektur



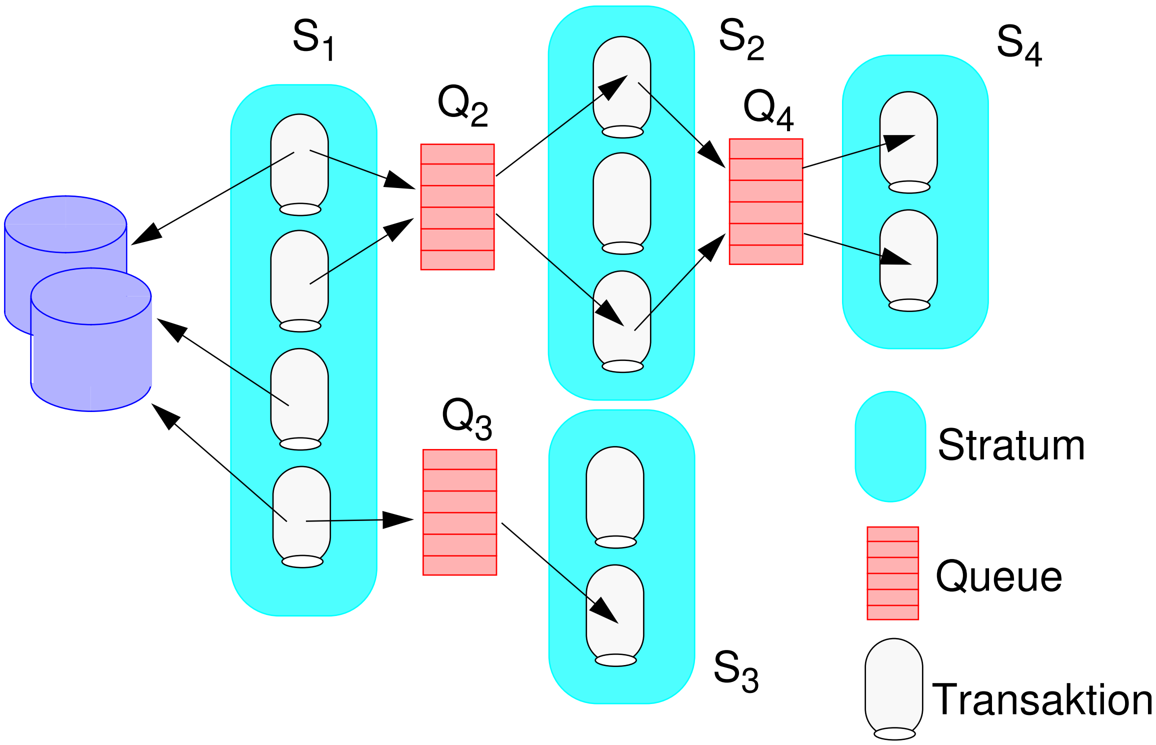
Nutzung im Workflow-Managment (1)

❑ Stratifizierte Transaktionen

- AW-orientierte Zerlegung der Transaktion T
  - in $T_1, \ldots T_n$;
  - Verkettung: jede T_i erhält persistente Warteschlange Q_i , aus der sie Anforderungen erhält, bestimmte Operationen auszuführen
  - lineare Reihenfolge nicht zwingend
- WICHTIG:
  - alle von den Transaktionen T_i manipulierten Ressourcen (also insbes. auch die Nachrichten) sind wiederherstellbar
  - dies bedeutet, daß sich die von den Transaktionen T_i benutzten Resource-Manager (DBVS, MOM) in atomares Commit einbinden lassen (XA-Protokoll, 2PC)
- Struktur startifizierter Transaktionen
  - einige T_i sollen gemeinsam zum erfolgreichen Ende kommen
  - disjunkte Zerlegung von T in Transaktionsmengen $S_1, \ldots, S_m$
  - ($S_i \subseteq T$ mit $S_i \neq 0$, und $S_i \cap S_j = 0$ für $i \neq j$,
    und $\bigcup_{j=1}^{m} S_j = T$)
  - Transaktionen von S_i werden durch 2PC-Protokoll synchronisiert
  - Menge S_i von Transaktionen heißt ***Stratum***

Nutzung im Workflow-Managment (2)

- **Stratifizierte Transaktionen** (Forts.)
  - Verkettung der Strata innerhalb der stratifizierten Transaktion T durch Baumstruktur



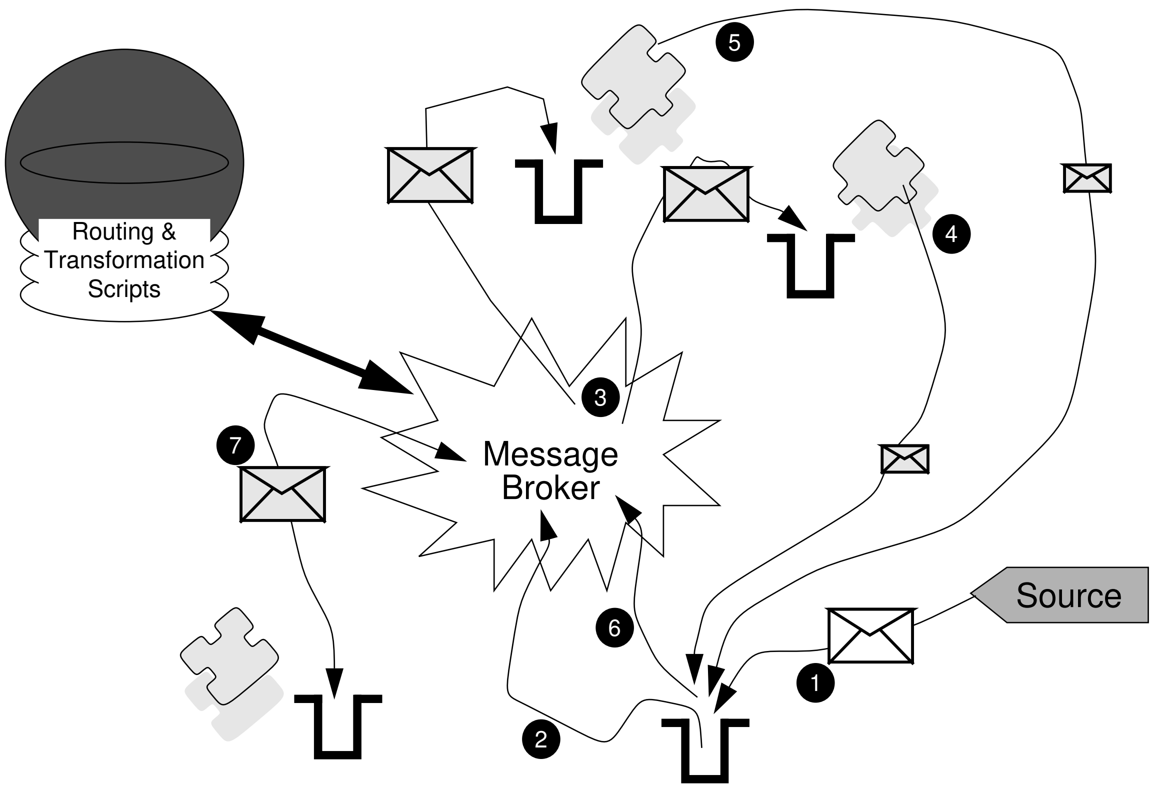
  - alle Strata führen schließlich Commit aus unter der Bedingung, daß die jeweiligen Vaterstrata zu irgendeinem Zeitpunkt vorher Commit ausgeführt haben
  - falls Stratum wiederholt scheitert (echte Ausnahme): stratifizierte Transaktion muß zurückgesetzt werden (Kompensation)

Nutzung im Workflow-Managment (3)

- **Stratifizierte Transaktionen** (Forts.)
  - Vorteile:
    - im Vergleich zu T: früheres Commit der einzelnen Strata S_i; dies impliziert frühere Freigabe der Sperren und damit höhere Nebenläufigkeit
    - Antwortzeit für Benutzer: Ausführungszeit des Wurzelstratums
    - 2PC-Protokolle für alle Strata können weniger Nachrichten umfassen als das 2PC-Protokoll einer globalen Transaktion (*Kolokation* als mögliches Kriterium für Stratifikation von T: lokale 2PC-Nachrichten)

Message Brokering (1)

- bisher (Message Queuing):
  - explizite Zieldefinition
  - vereinbarte Nachrichtenstruktur
- nun (Message Brokering):
  - Verteilung von Nachrichten ohne Zieldefintion
  - identische Struktur von gesendeter und empfangener Nachricht wird nicht vorausgesetzt
  - Publish/Subscribe
  - Regeln zur Weiterleitung von Nachrichten ausgehend von ihrem Inhalt
  - Anpassung von Format und Inhalt an ‘Empfängerwünsche’

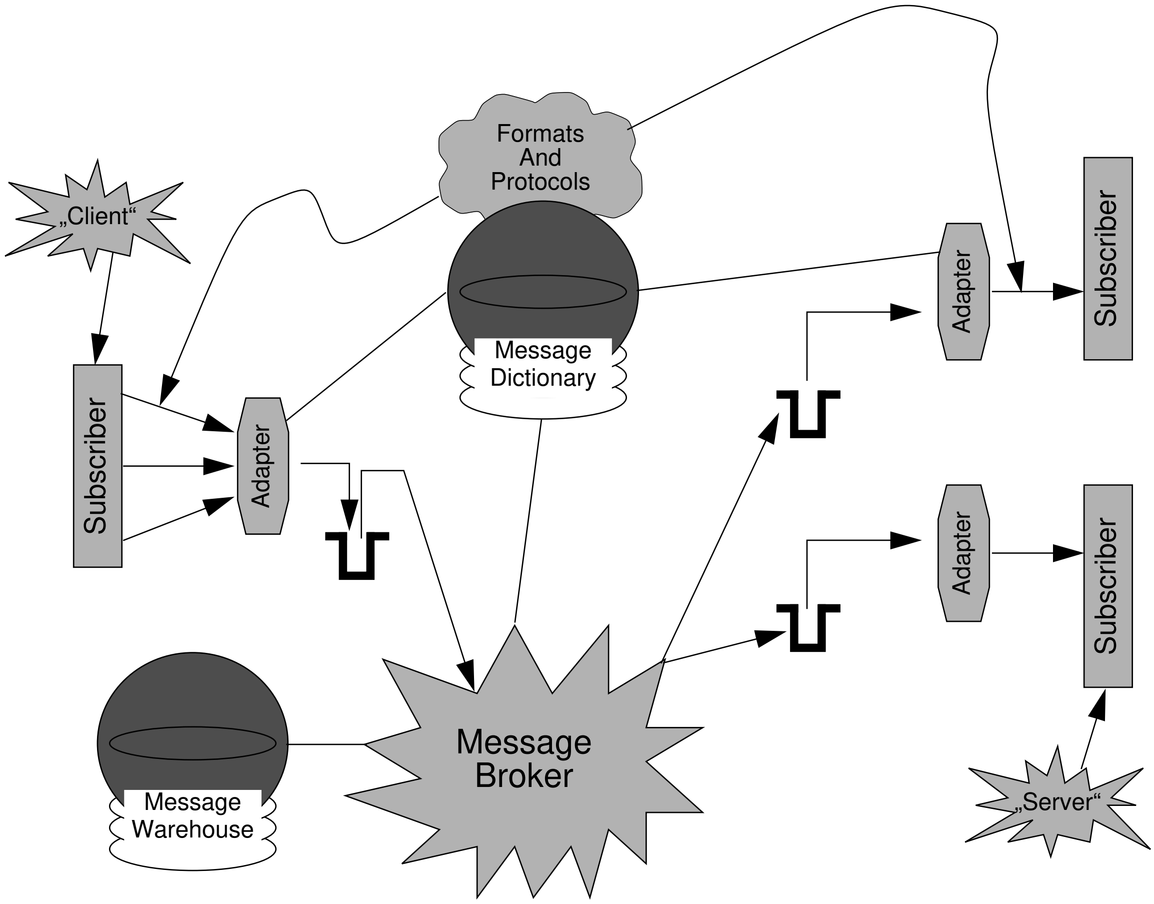


Message Brokering (2)

- ❑ Charakteristika
  - ➪ Quellen: *Publishers*
  - ➪ Senken: *Subscribers*
  - ➪ nach ‘Einreichung’ einer Nachricht bestimmt der Broker die Empfänger aufgrund der ‘Subscribe-Spezifikationen’
  - ➪ *Subscription*:
    - repräsentiert Intresse eines Empfängers an einer Nachricht ausgehend von ihrem Originalinhalt
    - definiert Format und Inhalt der ‘Zustellung’
    - inhaltliche Abweichungen
      - Annotierung
      - ‘Reinigung’
  - ➪ Filter
  - ➪ möglicherweise komplizierte Nachrichtentransformationen erforderlich

Message Brokering (3)

❑ Struktur[1]



- Dictionary:
  - Formatspezifikationen
  - Transformationsregeln/-funktionen/-Skripte
- Warehouse
  - weitere Auswertung und Analyse von Nachrichten

1. weitere Informationen in:
   F. Leymann: A Practitioners Approach to Data Federation bzw.
   F. Leymann, D. Roller: Production Workflow, Prentice Hall, 2000.

Zusammenfassung (1)

- Integration durch MOM
- Message Queuing
  - Nutzung
    - TP-Monitore, z. B. QTP in IMS/DC
    - WfMS, z. B. MQSeries-Workflow (IBM)
    - eigenständige Middleware zur asynchr. Kommunikation
  - DB-Aspekt
    - Persistenz der Nachrichten
    - Einbindung von Queue-Operationen in Sender- und Empfängertransaktionen
    - Folgen von Queue-Operationen als semantische Einheiten
  - Funktionalität
    - allgemeine Dienste
      - Enqueue, Dequeue, Browsing, ...
    - erweiterte Funktionalität (JPMQ)
      - strukturierte Message Types
      - anwendungsspezifische Prioritäten
      - inhalts-basierte Sortierung
      - inhalts-basierte Selektion
      - einstellbare Isolation zwischen Empfängertransaktionen
- Message Brokering
  - Publish/Subscribe
  - Nachrichtentransformation