Treating Your Data Legacy

Björn Häuser, reBuy reCommerce GmbH, Berlin

Wer steht überhaupt da vorne?

- PostgreSQL-Nutzer seit mehr als 5 Jahren
- Datenbanken seit mehr als 8 Jahren

- „The right tool for the right job.“

Agenda

1. Voraussetzungen / Situationsdarlegung
2. Was ist data Legacy?
3. Lösungsansatz

Was Sie nicht erwarten wird:

- Exzessives MySQL-Bashing
- Exzessives PHP-Bashing
- Vorgefertigte Lösungen

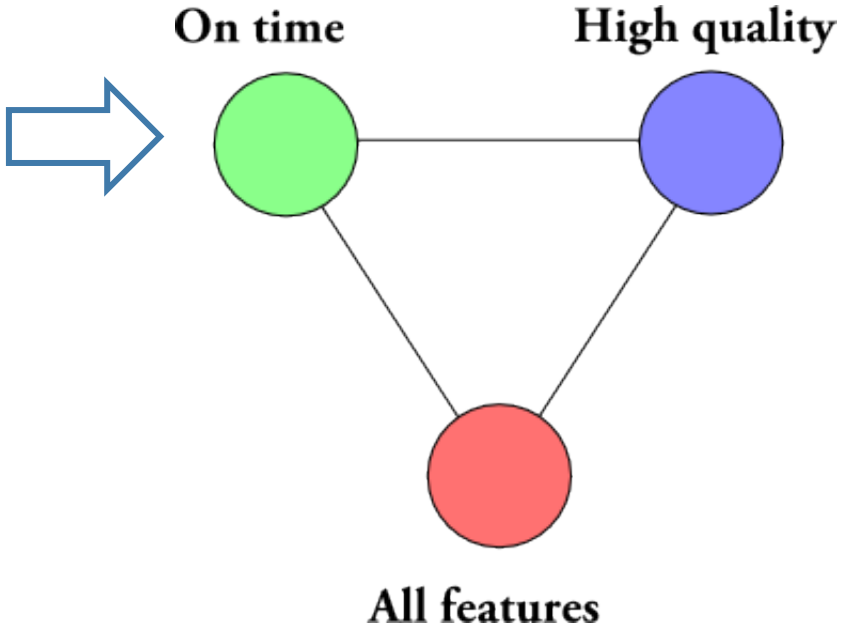
Was Sie erwarten können:

- Work-In-Progress
- Lösungsrezept
- Don‘ts

Was ist reBuy.de?

- Startup, schon über 5 Jahre am Markt aktiv
- Der einfache An- und Verkaufsshop im Internet
  - Bücher, DVDs, Blurays, Konsolen, Tablets
  - Vertragspartner ist reBuy reCommerce GmbH
- Ca. 4.5 Millionen Kunden
- Ca. 100.000 abgewickelte Artikel am Tag

Voraussetzungen I



- Time-To-Market

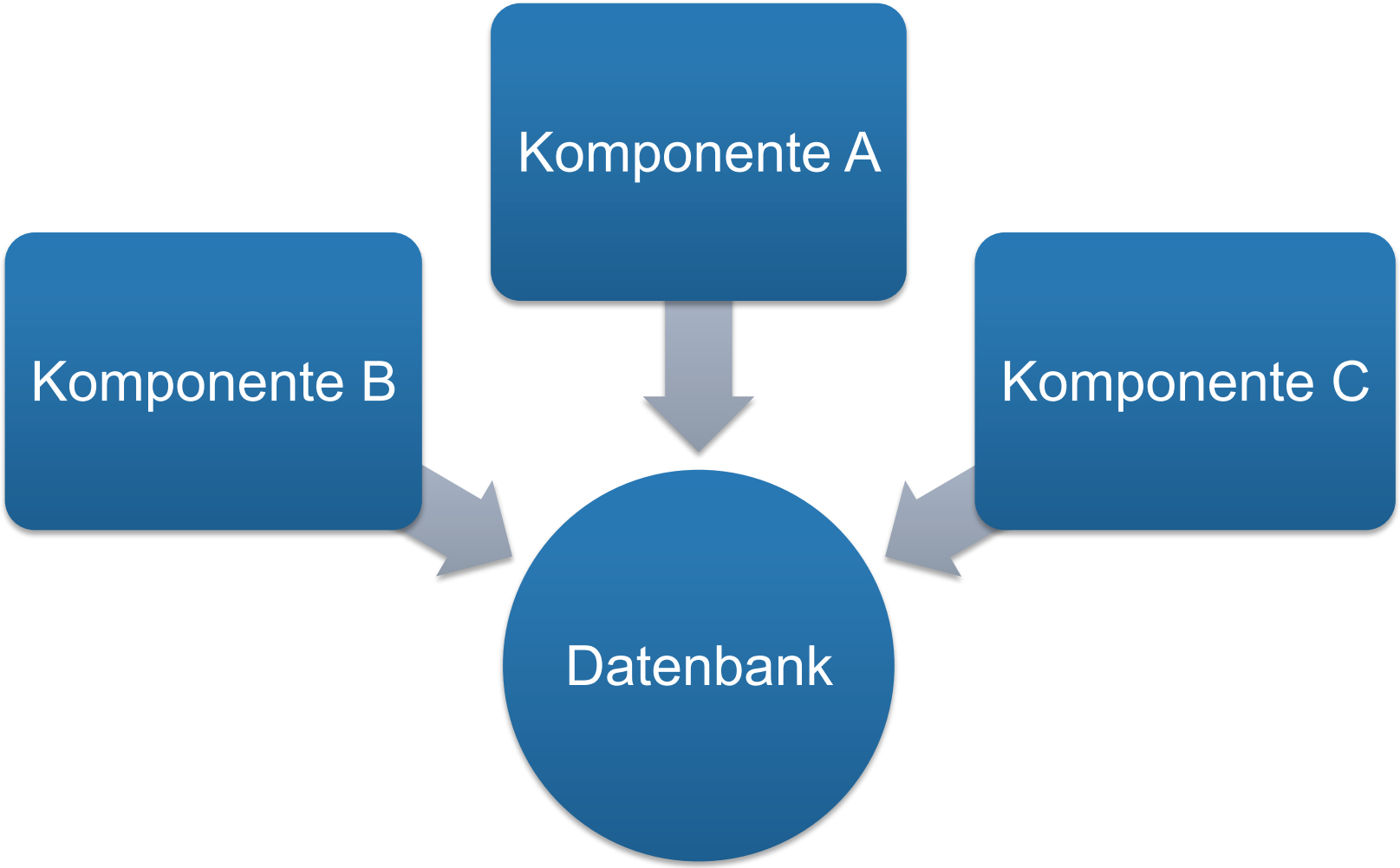
Voraussetzungen II

- Anfänglich viele Komponenten vollständig selbst entwickelt
- Großer angebrochener Monolith
- Mit MySQL begonnen und gewachsen
  - Hohe Investitionen um Unzulänglichkeiten auszugleichen
- Mit PHP begonnen und gewachsen

Aktuelle Baustellen I

- Umstellung zu einer Service-Orientieren-Architektur
  - Message-Driven
  - RabbitMQ
  - Java
  - PostgreSQL
- MySQL
  - Probleme mit:
    - Triggern
    - Funktionen
    - Komplexeren Abfragen

Beispiel: Prozess Stornierung



Was ist Data Legacy?

Falsche Daten

- Auftragstyp
  - Mit Livegang einer neuer Komponente wurde der Typ des Auftrags nicht korrekt gesetzt
  - Lange Zeit unbemerkt
  - Daten werden benötigt
  - Daten konnten wieder hergestellt werden
    - Anhand der Teilaufträge kann der Typ des Auftrages errechnet werden

reBuy reCommerce GmbH | Kanalstraße 139 | 12357 Berlin

Falsches Datenbankschema

- Auftrag storniert?
  - Stornierungsdatum gesetzt?
  - Status ist „storniert“?

- Retouren vs. Stornierung
  - Stornierungsdatum > Lieferdatum

- Mit einer einfachen Schemaänderung nicht reparierbar
- Umschreiben aller Komponenten notwendig
- Berücksichtigung der Sonderfälle auch in neuen Komponenten notwendig

Find your Abstraction!

Zielsetzung

- Neue Komponenten unabhängig von Alten entwickeln
- Abstraktion zwischen alter und neuer Datenhaltung
- Integration von Komponenten unabhängig von der Datenbank
- Unabhängig von:
  - Tabellen
  - Schemaversionen
  - Datenbankgrenzen

Think big!

- Nicht nur Lösung des eigentlichen Problems:
  - Data Legacy
- Sondern auch Vorbereitung für:
  - Wechsel zu PostgreSQL
  - Prozessunabhängigkeit

Warum PostgreSQL?

- Gute Erfahrungen (Als Anwender und Admin)
- Feature-Rich
- The better Open-Source?

Lösungsansatz I

- Entwicklung einer „Fire-Hose“ / „Wormhole“

Lösungsansatz II

- Eigener MySQL-Replikation-Client
- Dekodieren des binären Logs
- Erzeugen von Nachrichten für RabbitMQ:
  - <tblname><INSERT|UPDATE|DELETE>Message
- Probleme:
  - Richtige Handhabung der Logposition
  - Performance

Lösungsansatz III

- Message-Broker
  - Abstrahiert die generischen Messages
  - Pro Business-Message ein Broker
  - Let it crash!
- Java
  - Spring Integration AMQP
- Probleme:
  - Kompletter Rewrite der rebuy-messaging-library notwendig

Kunden anlegen I

- Allgemeine Message wird auf den Bus gelegt
- Diese Message entspricht dem Datenbankschema
- Bis zu diesem Zeitpunkt noch keine Abstraktion

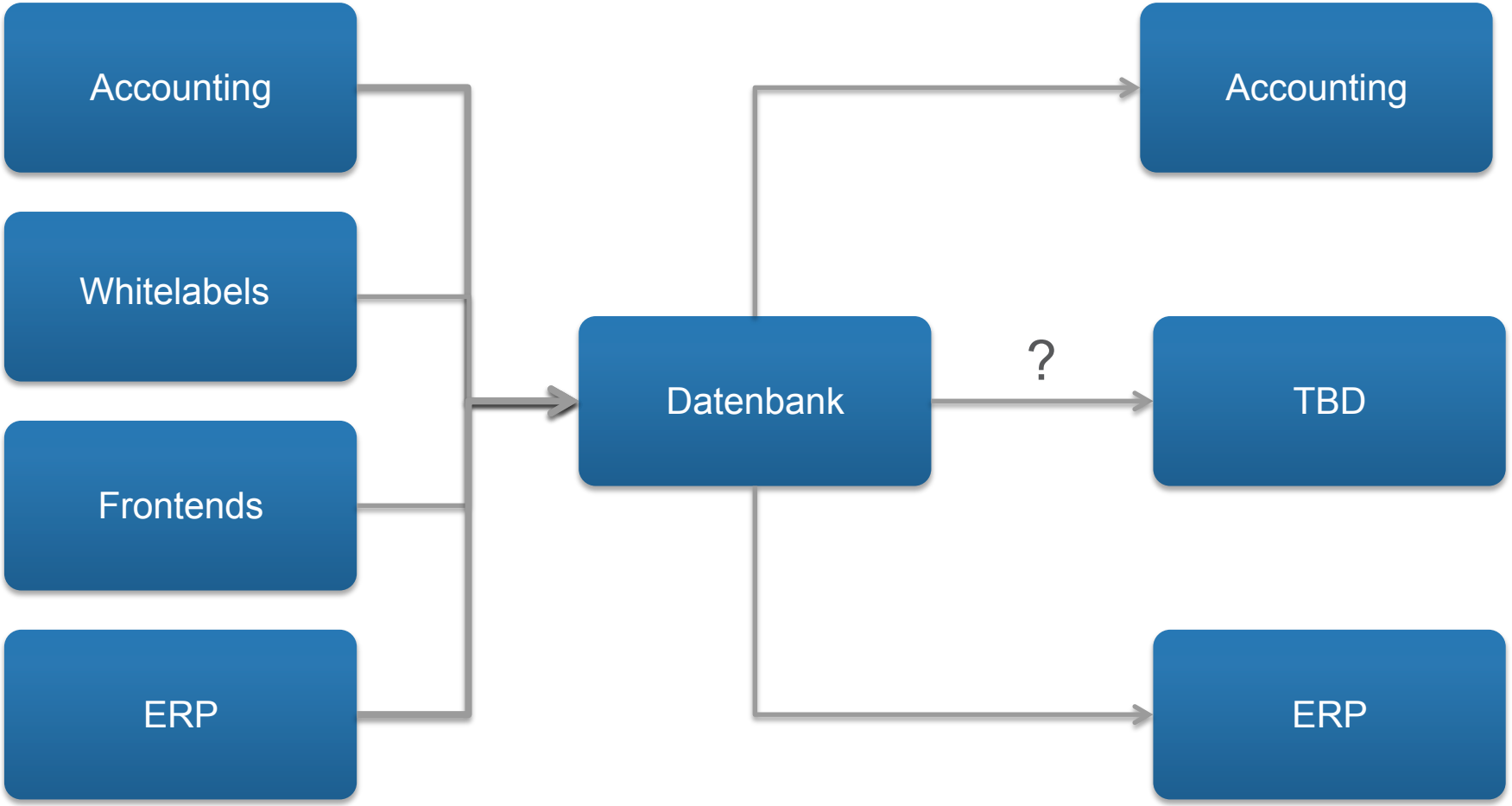
Kunden anlegen II

- Einführung einer neuer Abstraktionsebene
- Neue Komponenten sind nicht mehr abhängig von konkreten Tabellenspalten, Schemas, Datenbanken

Aufgabenstellung

- Neue Komponente benötigt Informationen, dass ein Auftrag retourniert worden ist
- Problem: Storno- und Retourenprozess eng verzahnt
- Integrationsmöglichkeiten:
  1. Direkt an den bestehenden Code
  2. Cron-Job, der überprüft, ob es neue Retouren gibt
  3. Message-Driven

Beispiel: Retoure



Möglichkeit I

- Direkt an die bestehenden Komponenten dranhängen
- Vorteil:
  - Geringe Time-To-Market (wenn Q&D)
- Nachteile:
  - Alle Komponenten müssen angefasst werden
  - Keine Abstraktion
  - Tight-Coupling

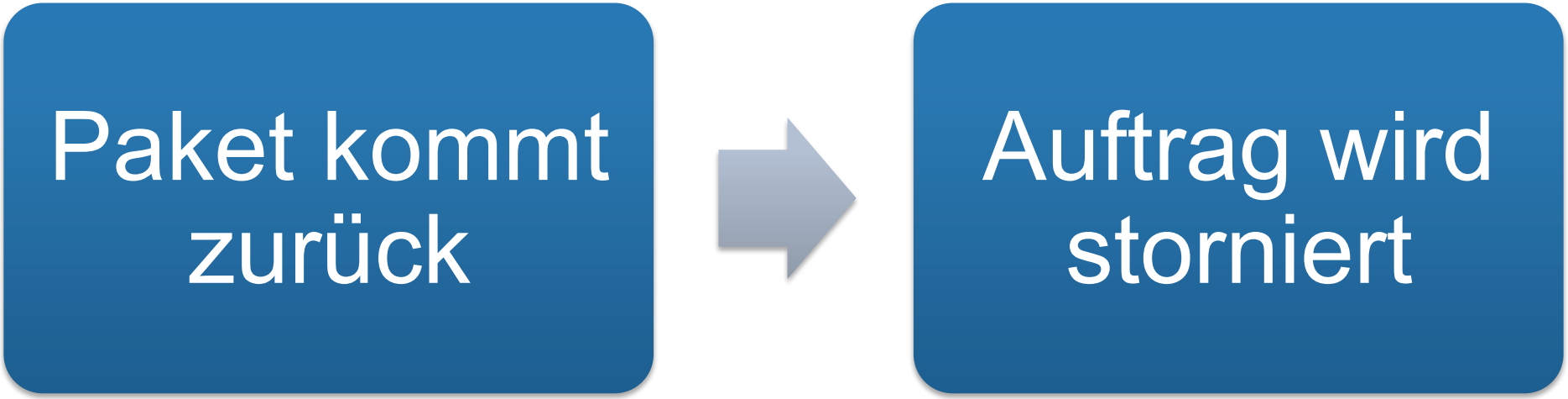
Möglichkeit II

- Per Cron-Job alle neuen Retouren abfragen
- Vorteil:
  - Kompromiss aus Abstraktion und Time-To-Market
- Nachteile:
  - Kopplung an das Datenbankschema
  - Berücksichtigung der Data-Legacy
  - Fehleranfälligkeit

Möglichkeit III

- Message-Driven
- Vorteile:
  - Vollständige Abstraktion
  - Lose-Coupling
  - Bestehende Komponenten müssen nicht angefasst werden
- Nachteile:
  - Duplizierung der Daten notwendig

Möglichkeit III

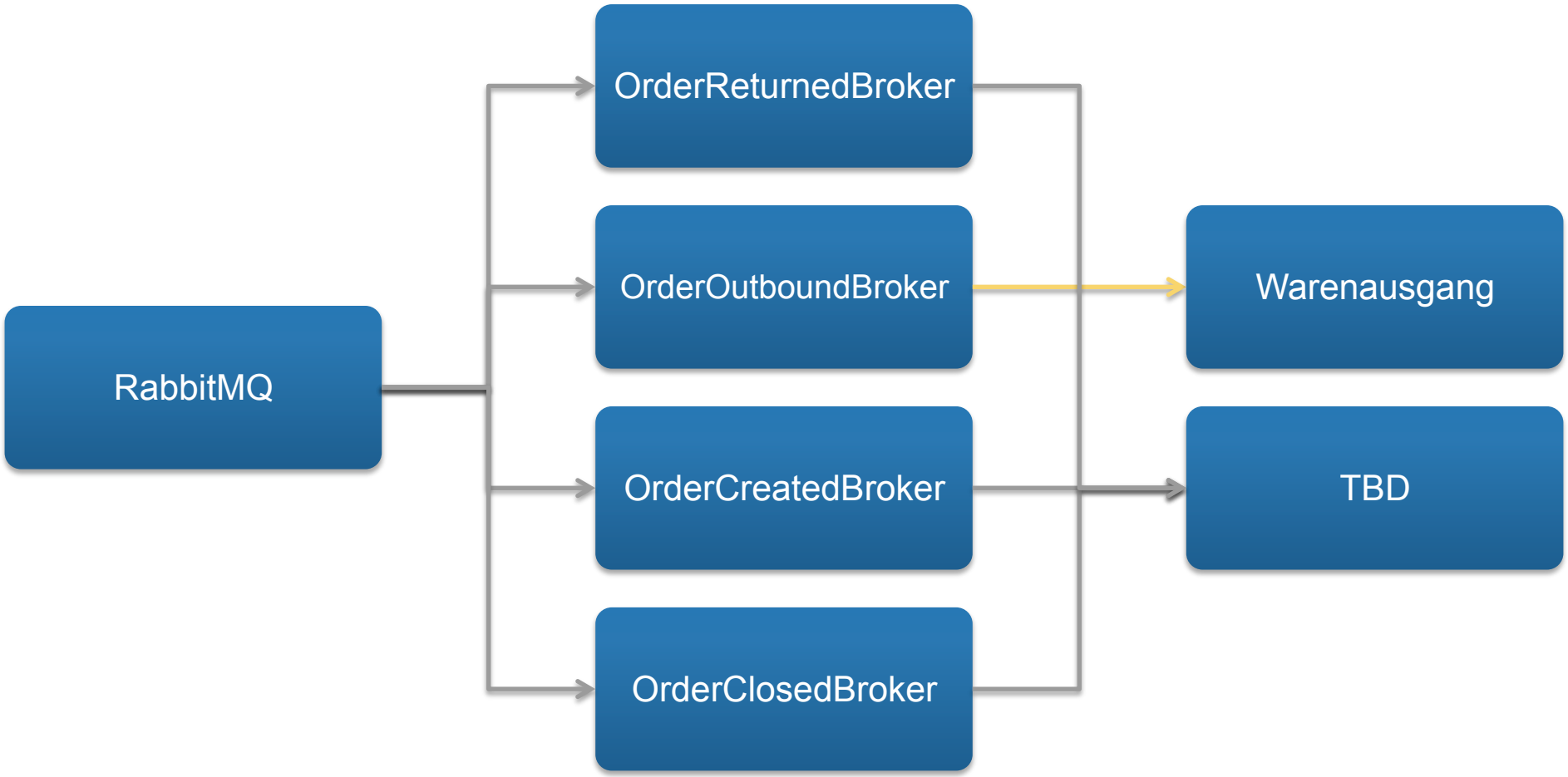


- UPDATE orders SET cancellation_date = NOW() WHERE id = ...;
- Bleibt vollständig unberührt

Möglichkeit III


```
{“schema“: “rebuy“, “table“: “orders“,
 “values“: {
   “cancellation_date“: {
      “old“: null, “new“: “2013-11-06 11:12:12“
   }
  }, “id_values“: {“id“: 123456, ...}}
```

Möglichkeit III



OrderCreatedMessage

```
{
 “uuid“: “3dfbd817-fd3f-4659-a4bd-ed771f0e0625“,
 “customer_uuid“: “...“,
 ...
}
```

OrderClosedMessage

```
{
 “uuid“: “3dfbd817-fd3f-4659-a4bd-ed771f0e0625“,
 “closed_stamp“: “2013-10-06 07:47:49“
}
```

OrderReturnedMessage

```
{
 “uuid“: “3dfbd817-fd3f-4659-a4bd-ed771f0e0625“,
 “returned_stamp“: “2013-10-20 12:30:00“
}
```

Zusammenfassung Möglichkeit III

- Neue Komponente kann unabhängig entwickelt werden
- Vollständige Abstraktion der Data Legacy
- Storno- und Retourenprozess können unabhängig von neuen Komponenten getrennt und refaktorisiert werden
- Es muss nur sichergestellt werden, dass weiterhin eine OrderReturnedMessage erstellt wird

Erreichte Ziele

- Abstraktion zwischen neuen und alten Komponenten
- Möglichkeit geschaffen:
  - Data Legacy Stück für Stück aufzuräumen
    - Und damit auch alte Komponenten
  - Großer Gewinn an Freiheit bei der Entwicklung von Komponenten
- Zentralisierung von Funktionalität

Facts

- Pro Tag:
  - Zwischen 3 und 6 Millionen generische Messages
  - Zwischen 800k und 1.2 Millionen Businessmessages
- Derzeit:
  - Ca. 40 Broker
  - Ca. 60 Listener
- Und RabbitMQ langweilt sich dabei immernoch.

Danke schön!

Looking for a new Job?
E-Mail your github profile to it-jobs@rebuy.de

reBuy.de