

---

# Entwurfsmuster für Enterprise Application Integration

Prof. Dr. Christian Pape

---

# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

# Überblick

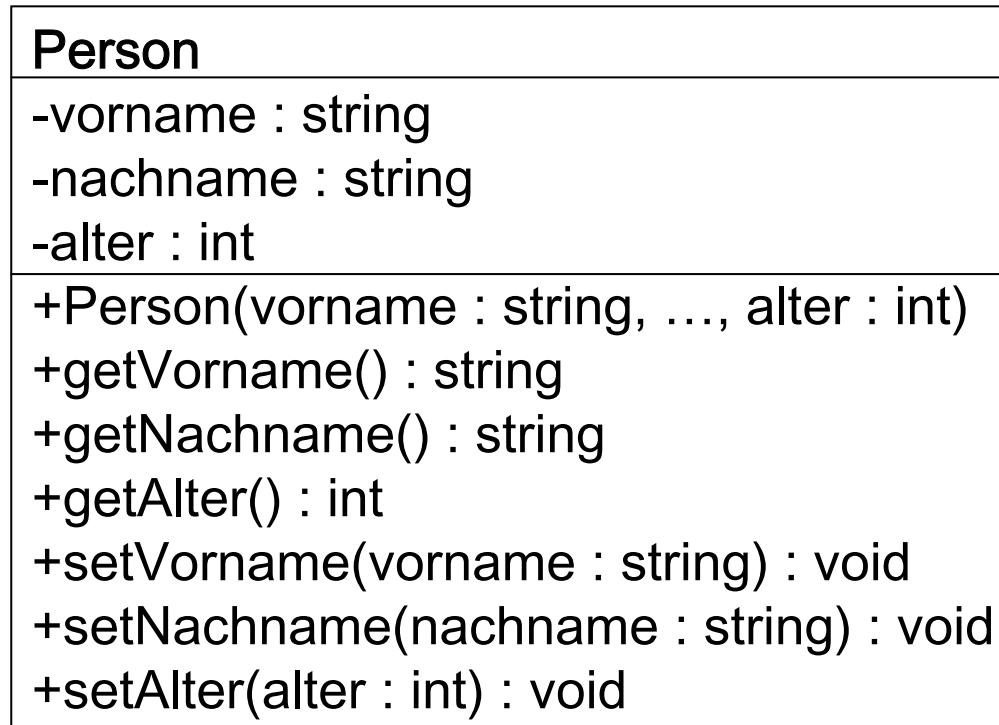
---

- Ziel
  - Einige *Entwurfsmuster* kennen und auf Integrationsprobleme anwenden können
  - *Implementierungsvarianten* eines Entwurfs angeben und abschätzen können
  - Zwischen Entwurf der Lösung und dessen Implementierung unterscheiden können
- Entwurf
  - Wie soll Lösung strukturell aufgebaut werden?
- Implementierung
  - Womit sollen Lösungsbausteine gebaut werden?

# Überblick

---

- Entwurf UML Klasse



- Implementierung in C?

# Überblick

---

- Quelle
  - Gregor Hoppe, Bobby Woolf: Enterprise Integration Patterns, Addison Wesley
- Weitere Literatur über Entwurfsmuster
  - (GoF) Erich Gamma, et. al.: Design Pattern
  - Frank Buschmann, et. al.: Architectural Patterns
  - Martin Fowler: Patterns of Enterprise Application Architecture

# Überblick

---

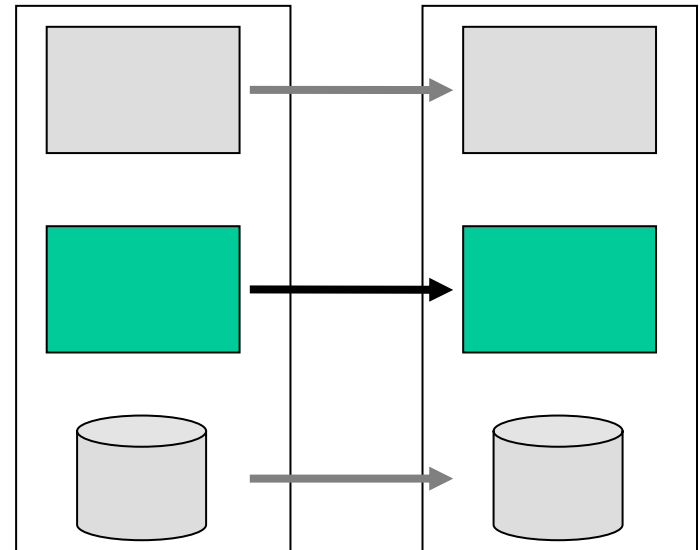
- Elemente eines Entwurfsmuster (GoF)
  - Name zur Identifikation und Kommunikation (Titel)
  - Problembeschreibung (Frage, Beispiel)
  - Lösungsentwurf
  - Konsequenzen (positive, negative)
  - (Implementierung)
- Implementierung normalerweise nicht Bestandteil eines Entwurfsmuster (höchstens zur Illustration)
  - Entwurfsmuster nicht Implementierungsmuster

# Überblick

- Fokus der folgenden Entwurfsmuster
  - Entwurfsmuster für asynchrone Koppelung von Systemen mit Nachrichten, z.B. mit MQ
  - Nachrichten in XML (wenn nicht anders erwähnt)
  - Integration auf Funktionsebene

- Vertikale Integration

- Siehe Patterns of Enterprise Application Architecture
- Verteilte Systeme
- J2EE Entwurfsmuster



# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

# Pipes and Filters

---

- Unix
  - Pipe: Zwischenspeicher für Dateneingabe/-ausgabestrom von Anwendungen
  - Filter: Unix-Kommandos
  - `cat Personen.txt | grep "Christian" | wc -l`
- Compilerbau
  - Pipe: In-memory-Zwischenspeicher oder Datei
  - Filter: Präprozessor, lexikalische Analyse, Parsing, Semantische Analyse, Codegenerierung

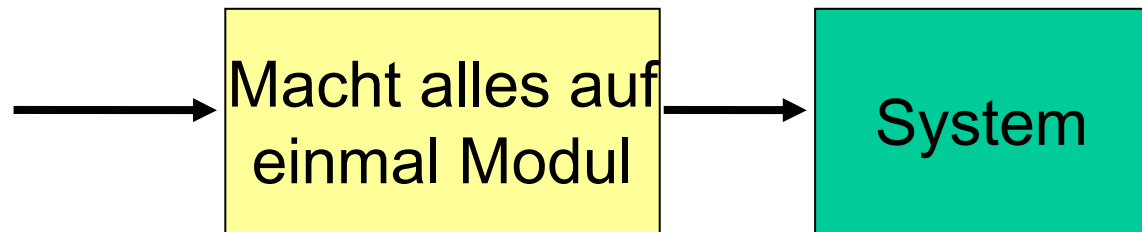
# Pipes and Filters

---

- Problem
  - Wie können komplexe Verarbeitungen einer Nachricht durchgeführt werden aber mit unabhängigen und flexibeln Verarbeitungsschritten?
- Beispiel: Bestellung wird in Form einer Nachricht von einer anderen Firma empfangen
  - Bestellnachricht verschlüsselt
  - Bestellnachricht mit digitalen Schlüssel authentifiziert
  - Bestellungen könnten doppelt auftreten
  - Zielsystem benötigt unverschlüsselte einfache Textnachrichten

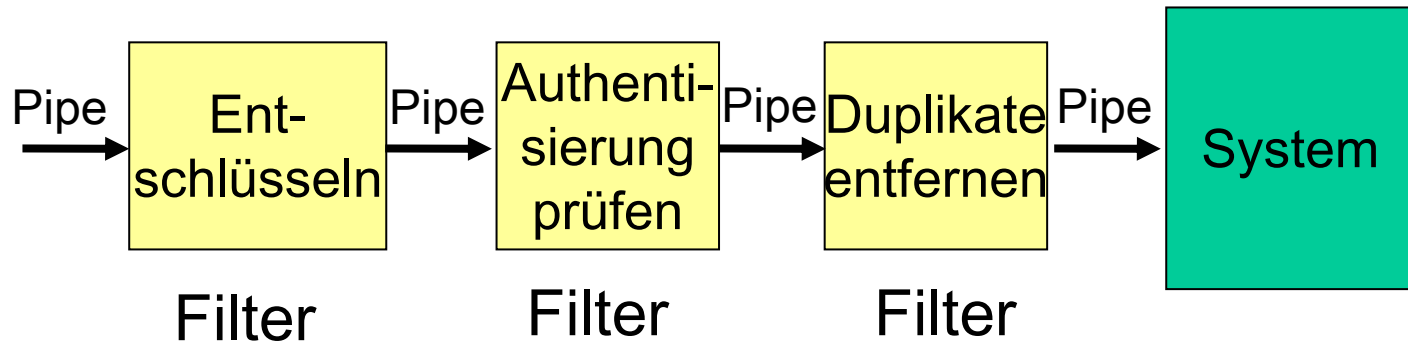
# Pipes and Filters

- Schlechte Lösung
  - Entschlüsselung, Authentisierung, Bereinigung von doppelten Nachrichten in ein monolithisches Modul mit genau einem Eingang für die Nachricht und einem Ausgang
  - Z.B. ein einziges Message Driven Bean in einem EJB Container, oder ein einziger ContentHandler (Java SAX API)
- Schlecht, da:
  - Verschlüsselung sich ändern kann
  - Verschlüsselung ggf. nur auf dem System mit privatem Schlüssel ausführbar
  - Authentisierung kann pro Verkaufskanal anders sein
  - Schlechte Wiederverwendbarkeit



# Pipes and Filters

- Bessere Lösung
  - Pro Verarbeitungsschritt ein Filter
  - Filter mit Pipes zusammenschließen



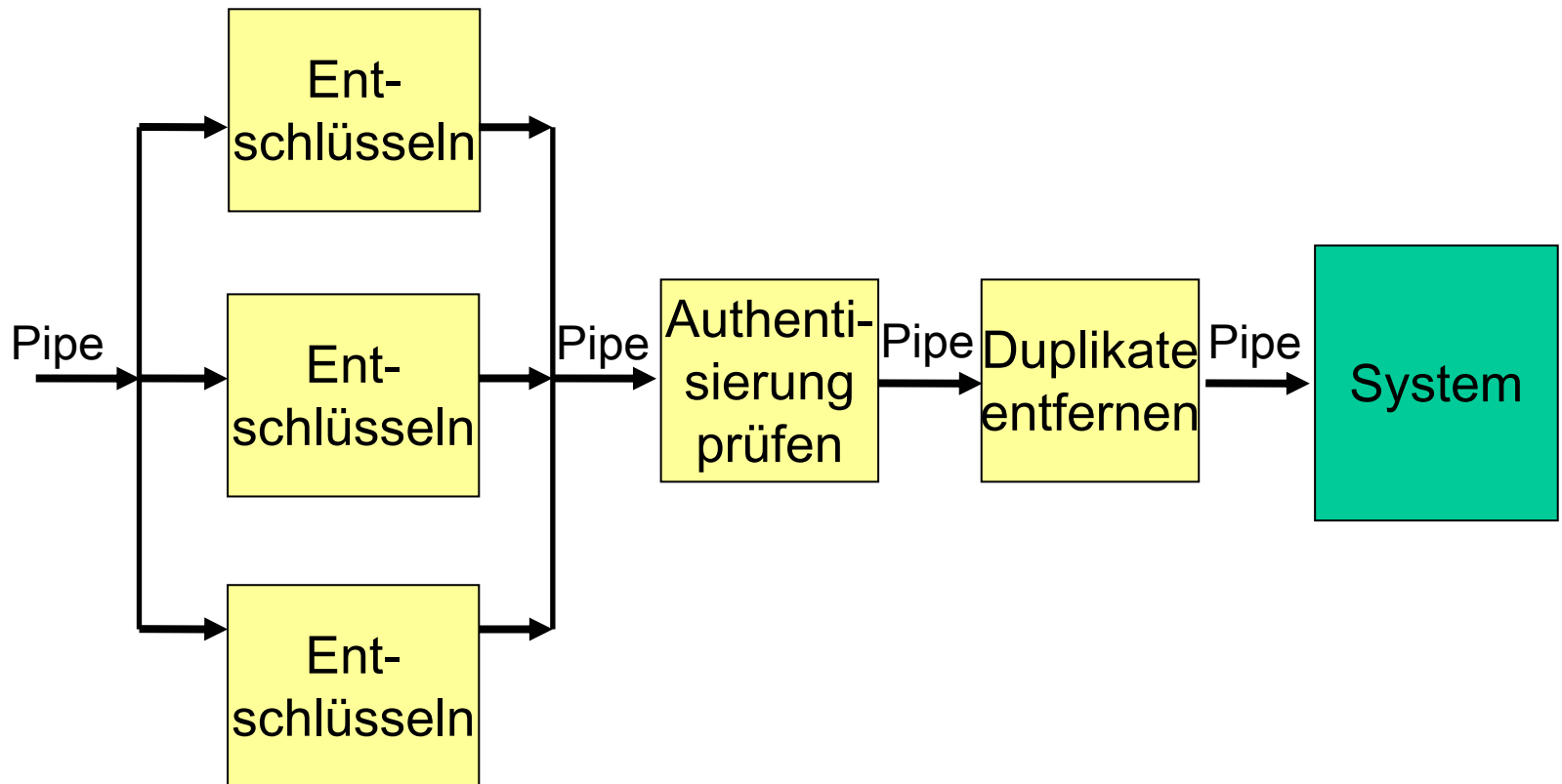
- Pro Filter klar umgrenzte Aufgabe, definiertes Eingabeformat, definiertes Ausgabeformat
- Pipe kann eine Queue sein, Dateiverzeichnis

# Pipes and Filters

---

- Konsequenzen (positive)
  - Filter arbeiten unabhängig (asynchrone Kopplung).
  - Pipe kann noch nicht verarbeitete Nachrichten zwischenspeichern
  - Filter austauschbar, Reihenfolge änderbar
  - Bei ungleicher Auslastung können Filter parallel betrieben werden (Skalierbarkeit besser)
- Konsequenzen (negative)
  - Viele Pipes benötigt (kann Ressourcen kosten)

# Pipes and Filters



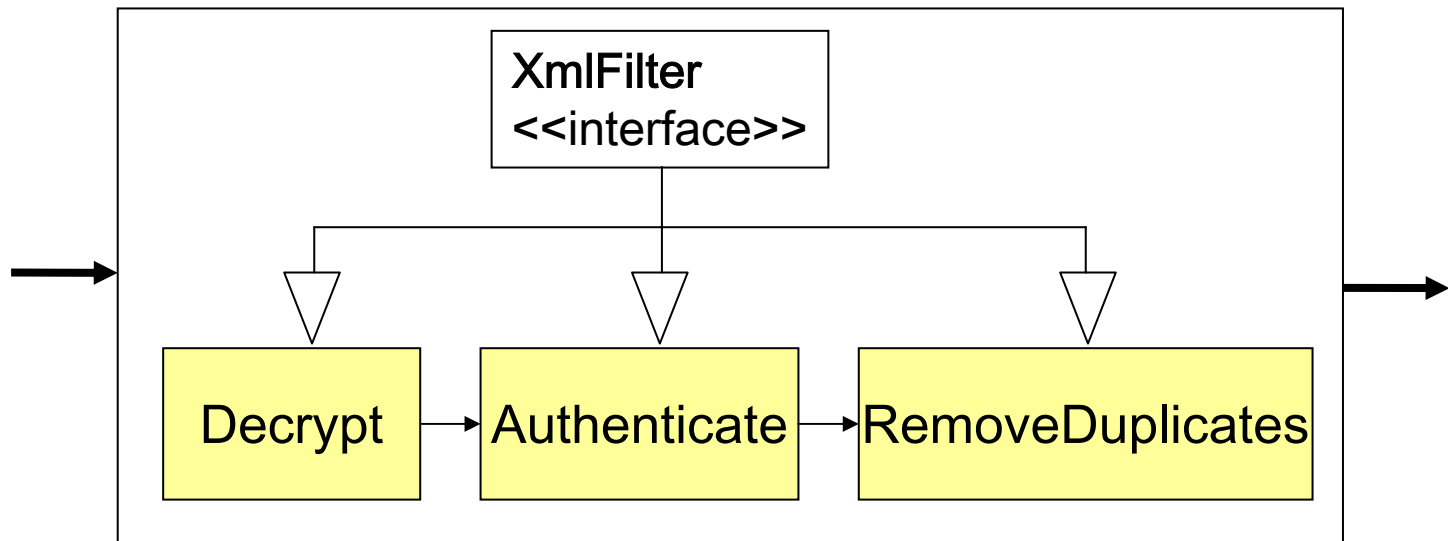
# Pipes and Filters

---

- Implementierung
  - Pipes and Filters mit Message Queue
  - Pipe
    - (persistente) Queue
  - Filter
    - Programm, welches Nachricht von einer Queue entgegennimmt, verarbeitet und auf die Ausgabequeue stellt
    - Message Driven Bean, mit JMS, BizTalk

# Pipes and Filters

- Implementierung mit SAX (Simple Java API for XML)
  - Filter: XmlFilter (eine Klasse)
  - Pipe: Beziehung zu nächstem Filter und Methodenaufruf des nächsten Filters
- Vorteil: In-memory-Pipe



# Pipes and Filters

Create two filters:

- Add currency element with default EUR
- Print output XML with indentation of elements

```
<products>
  <item>
    <last-change>2001-03-05</last-change>
    <id>a-123123-xs</id>
    <name>Wooden &ref; chair</name>
    <price>120.00</price>
  </item>
  <item>
    <last-change>2005-06-11</last-change>
    <id>123-das3</id>
    <name>Table (wood)</name>
    <price>523.00</price>
  </item>
</products>
```

1

```
<products>
  <item>
    <last-change>2001-03-05</last-change>
    <id>a-123123-xs</id>
    <name>Wooden &ref; chair</name>
    <price>120.00</price>
  </item>
  <item>
    <last-change>2005-06-11</last-change>
    <id>123-das3</id>
    <name>Table (wood)</name>
    <price>523.00</price>
    <currency>EUR</currency>
  </item>
</products>
```

2

```
<products>
  <item>
    <last-change>
      2001-03-05
    </last-change>
    <id>
      a-123123-xs
    </id>
    <name>
      Wooden
      AREF
      chair
    </name>
    <price>
      120.00
    </price>
    <currency>
      EUR
    </currency>
  </item> ...
```

# Pipes and Filters

---

```
public class CurrencyFilter extends XMLFilterImpl {

    public CurrencyFilter(XMLReader xmlReader) {
        super(xmlReader);
    }

    public void endElement(String uri, String localName,
                          String qName) throws SAXException {
        if ("item".equals(localName)) {
            super.startElement(uri, "currency", localName, null);
            super.characters("EUR".toCharArray(), 0, 3);
            super.endElement(uri, "currency", localName);
        }
        super.endElement(uri, localName, qName);
    }
}
```

# Pipes and Filters

```
public class IdentFilter extends XMLFilterImpl {
    private int indentation = 0; // constructor missing
    public void startElement(String uri, String localName,
        String qName, Attributes atts) throws SAXException {
        super.startElement(uri, localName, qName, atts);
        printSpaces();
        System.out.println("<" + localName + ">");
        indentation++;
    }
    public void endElement(String uri, String localName,
        String qName) throws SAXException {
        super.endElement(uri, localName, qName);
        indentation--;
        printSpaces();
        System.out.println("</" + localName + ">");
    }
    public void characters(char [] ch, int startIndex, int length) {
        super.characters(ch, startIndex, length);
        printSpaces();
        System.out.println(new String(ch, startIndex, length));
    }
}
```

# Inhalt

---

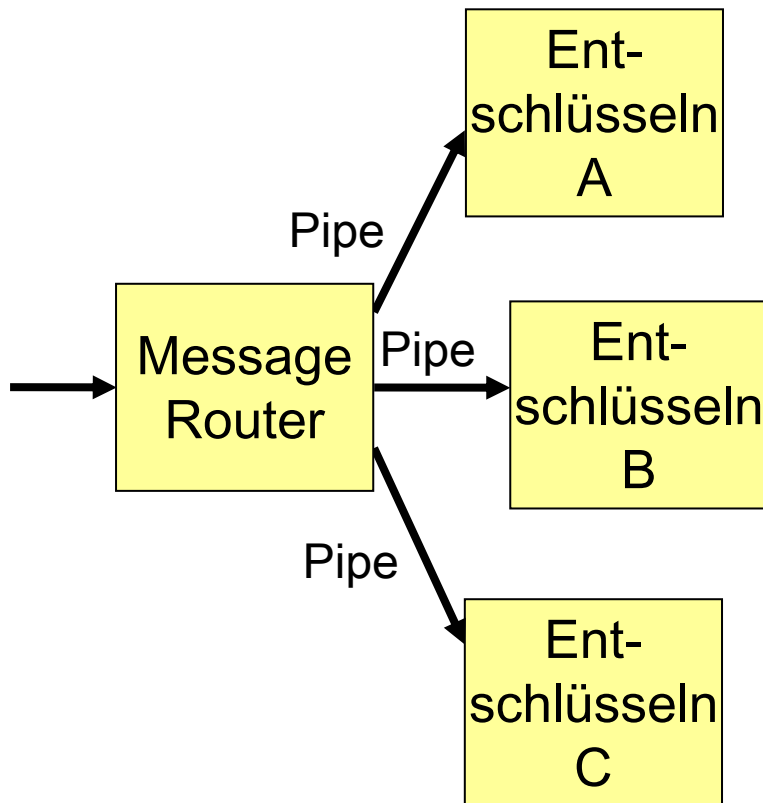
- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - **Message Router**
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

# Message Router

---

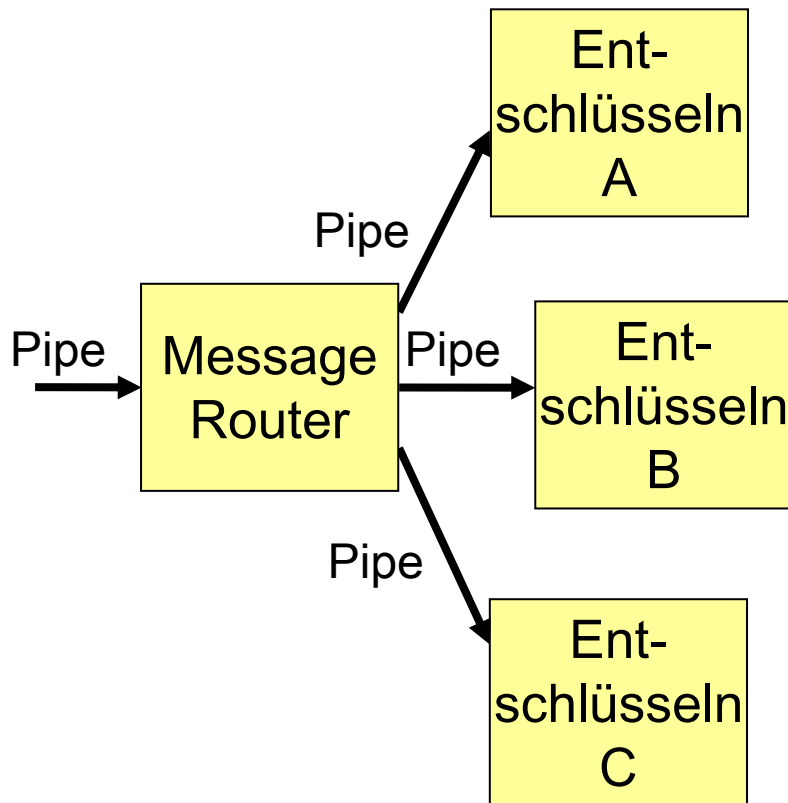
- Problem
  - Wie können Nachrichten in Abhängigkeit von Bedingungen an unterschiedliche Filter gesendet werden ohne direkte Kopplung?
  - Beispiel: Nachricht verschlüsselt mit Alg A, Alg B, Alg C.
- Schlechte Lösung mit Pipe and Filter
  - direkte Kopplung der Filter durch Pipes
  - Zu viele Sequenzen von Pipes notwendig
- Weitere schlechte Lösung
  - Publish-Subscriber: Filter sollen selbst entscheiden, ob Nachricht verarbeitet werden soll (höhere Kopplung, schlechte Wiederverwendbarkeit)

# Message Router



- Lösung
  - Einen Filter verwenden, der Nachricht entgegennimmt, Bedingung auswertet und in Abhängigkeit der Bedingung die Nachricht auf einer anderen Pipe neu publiziert

# Message Routing



- Konsequenzen (positive)
  - Bestehende Filter unangetastet
  - Entschlüsselung muss nicht entscheiden, ob Nachricht verarbeitet werden soll/kann
  - Router / Entschlüsselung lose gekoppelt
  - Anzahl zusätzlicher Pipes gering
- Konsequenzen (negative)
  - Zu viele Router lassen erzeugen komplizierten Datenfluss (Testen, Debugging schwierig)
  - Ggf. (wie bei Email) Nachricht um Stationen, die durchlaufen wurden, erweitern
  - Höhere Latenzzeiten, falls Pipes Netzwerkaufrufe sind / Router auf anderen Rechner

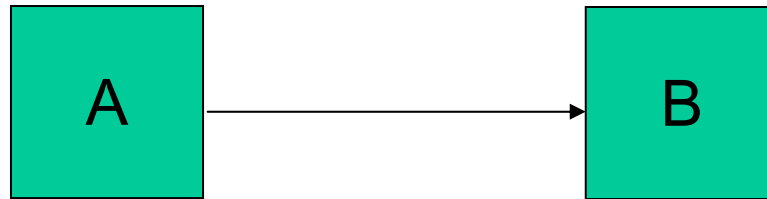
# Message Routing

---

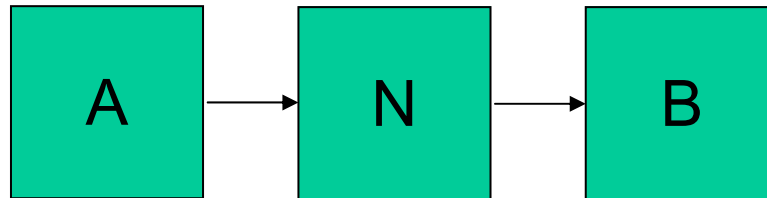
- Implementierung Routing
  - Verzweigung in einem Geschäftsprozess
- Hub and Spoke
  - Hub: Routing Funktionen wichtiger Bestandteil (auch Pipes und Filter)
  - Spoke: Systeme, Filter
- J2EE
  - Java Programm, dass via JMS/Message Driven Bean eine Nachricht entgegennimmt, auswertet und an andere Queues verteilt
- SAX
  - XmlFilter, der in Abhängigkeit von Inhalt Daten an andere Filter verteilt

# Kopplung / Entkopplung

- Starke Kopplung: A hängt direkt von (einem Teil von B) ab



- A von B entkoppeln: Etwas Neues dazwischenschalten



- Gefahr: Den Teufel mit Belzebub vertrieben zu haben
  - N sollte wesentlich weniger komplex als B sein
- Generelles Ziel Entwurf / Architektur
  - Wenige und kleine Komponenten mit geringen Abhängigkeiten
  - Trade-off: Kleine Komponenten zu wenigen und geringen Abhängigkeiten

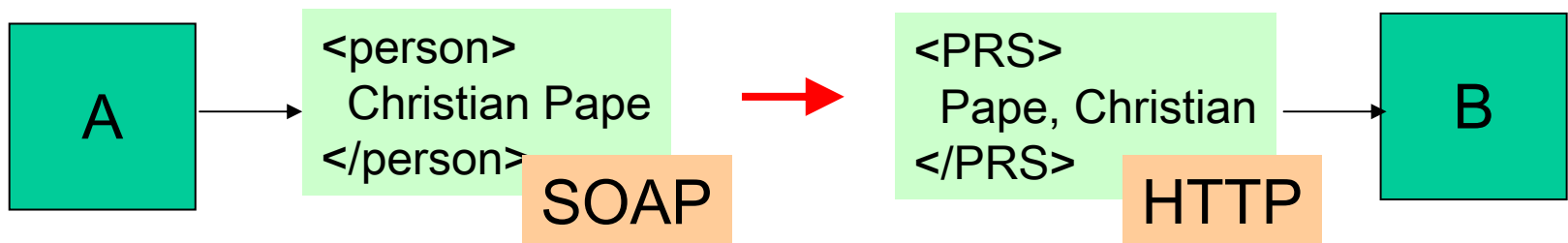
# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - **Transformation**
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

# Message Translator

- Problem
  - Wie können Systeme Nachrichten austauschen, die jeweils ein unterschiedliches Format besitzen?
- Verschiedenen System besitzen (für vergleichbare Daten) unterschiedliche Datenmodelle
  - Schnittstellen (API oder Nachrichtenformate) sind meist ein Abbild des Datenmodells
  - Jedes System erzeugt Nachrichten in eigenem proprietären Format (eigenes XML, CSV, alte EDI Formate, ...)
  - Protokolle für den Versand können auch unterschiedlich sein (SOAP, HTTP, Sockets, MQ)



# Message Translator

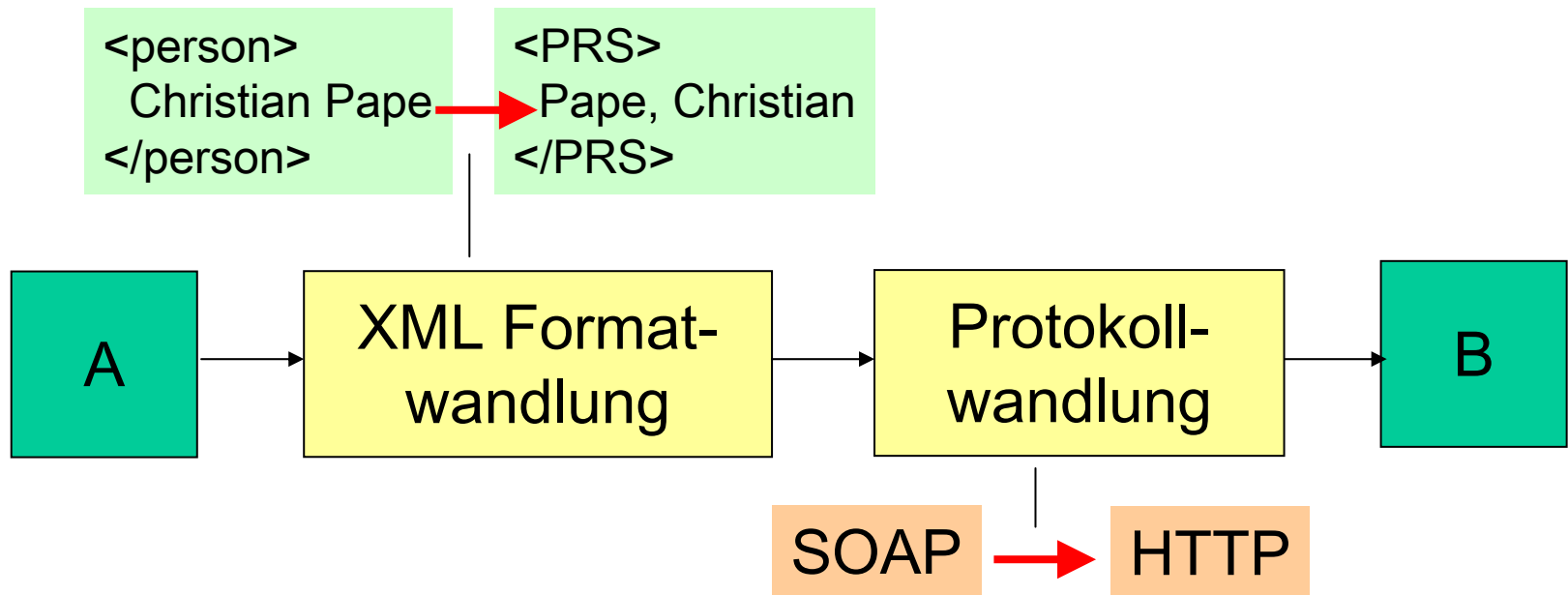
---

- Schlechte Lösung(en)
  - Gemeinsame Datenbank / Datenformat
  - Quellsystem wandelt eigenes Format in Zielformat um (oder beim Zielsystem)
- Konsequenzen
  - Starke Kopplung von Quellsystem zu Zielsystem (oder umgekehrt)
  - Protokollwandlung technisch nicht immer durchführbar bei Quellsystem (Mainframe Hostmaske zu XML und SOAP wandeln)
  - Neue Version vom Quellsystem nicht isoliert installierbar
  - Wiederverwendung von Code schwierig

# Message Translator

## ■ Lösung

- Verwende einen oder mehrere speziellen Filter, die das Quelldatenformat in das Zieldatenformat *umwandelt*
- Bei komplizierten Aufgaben: Pipes
- Analog GoF Adapter



# Message Translator

---

- Konsequenzen
  - Anwendungen entkoppelt (Filter dazwischen sind)
  - Filter potentiell wieder verwendbar (SOAP / HTTP Filter)
  - Latenzzeiten ggf. erhöht, falls Pipes nicht im Speicher / Filter auf anderen Rechner

# Message Translator

Schicht (ähnlich ISO/OSI)	Inhalt	Transformation (Bspl)	Werkzeuge (Bspl)
Struktur der Daten	Relationen, Beziehungen, Kardinalitäten	n-m Beziehung in 1-n und m-1 Beziehung auflösen	Programm, SQL
Datentypen	Feldnamen, Wertebereiche, Einschränkungen	Datum in ISO-Code Umwandeln	XSL, SQL, Programm
Datenrepräsentation	XML, CSV, EDI, Herstellerabhängig ASCII, UTF-8 Verschlüsselung	Datenformat wandeln	XSL, XML Parser
Transport	TCP/IP, MQ, IIOP, JRM		EAI Adapter, Programm

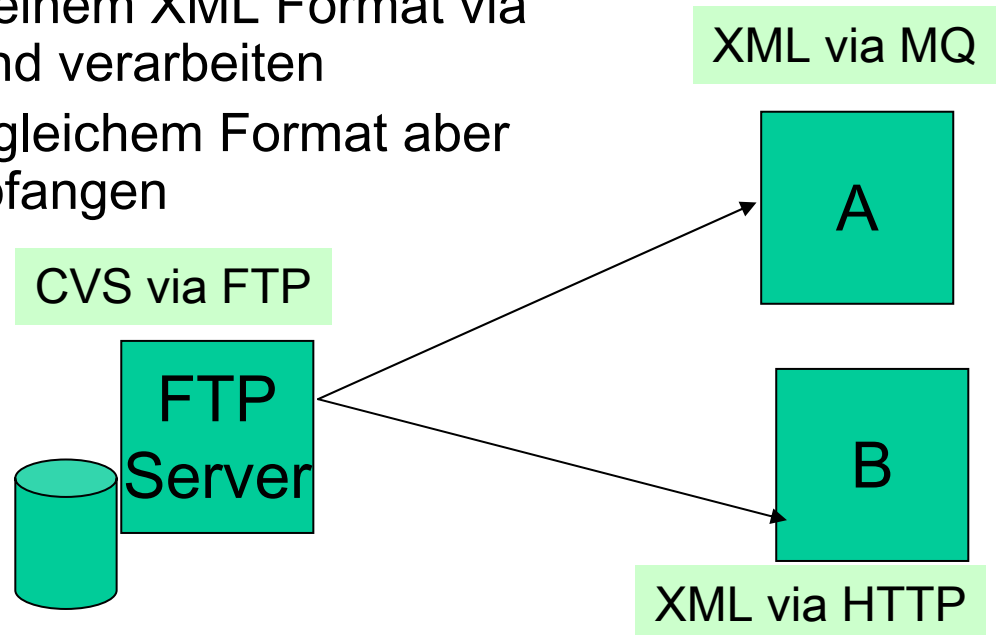
# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - **Beispiel**
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

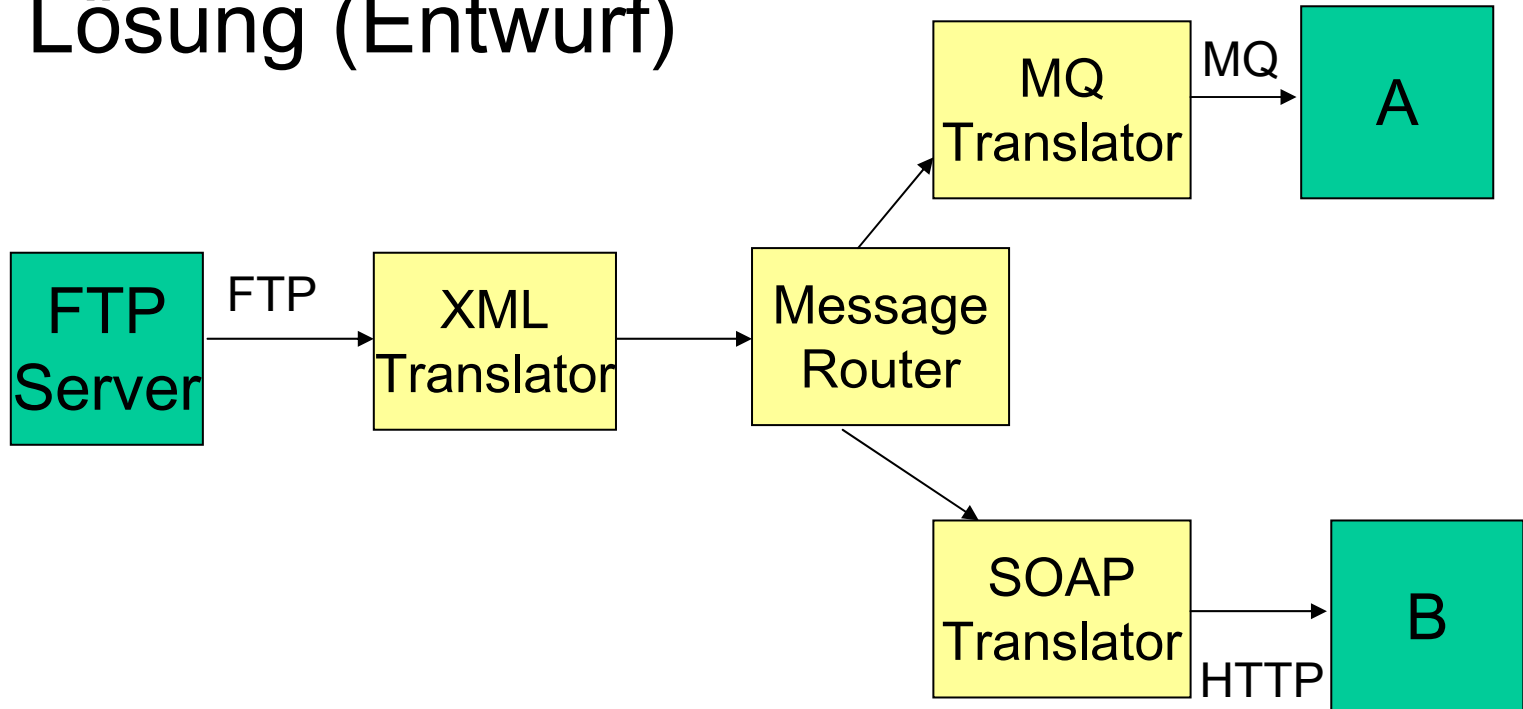
# Beispiel

- Quellsystem
  - Ein Datenbankextrakt liegt als CSV Datei auf einem FTP Server
- Zwei Zielsysteme
  - A: Kann Daten in einem XML Format via MQ empfangen und verarbeiten
  - B: Kann Daten in gleichem Format aber nur via HTTP empfangen



# Beispiel

- Lösung (Entwurf)



- Alternativen

- Zusätzliche Protokollwandlung vor XML Translator

# Message Translator

---

- Implementierung (viele Varianten)
  - Hub and Spoke mit EAI Werkzeug (z.B. IBM Business Integrator). Visuelle Programmierung.
  - Eine EJB Anwendung (Programmieren z.B. mit SAX API oder mit XSL)
- Beide beiden Varianten
  - XML Translator mit XSL möglich
  - Sollte möglichst auf einem Integrationsserver ablaufen (Debugging, Testen, Latenzzeiten, Installation)
- Wahl hängt von Komplexität der Aufgaben ab
  - Programmieren bei komplexeren Aufgaben, z.B. komplexe Datenformatwandlung
  - Mischen je nach Technologie und Werkzeug möglich

# Inhalt

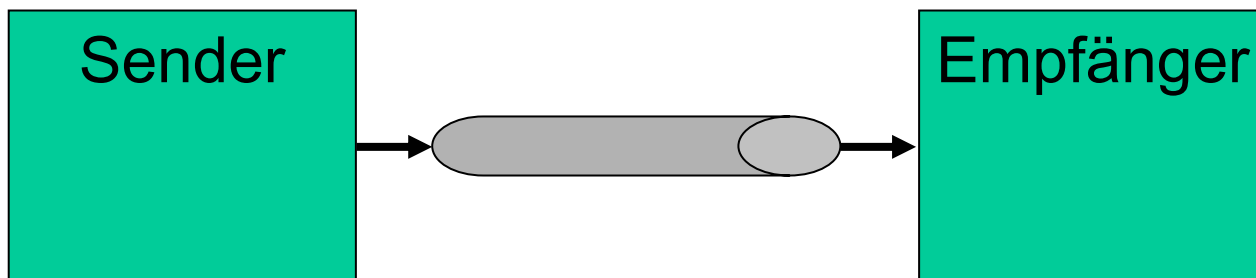
---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

# Message Channel (1/3)

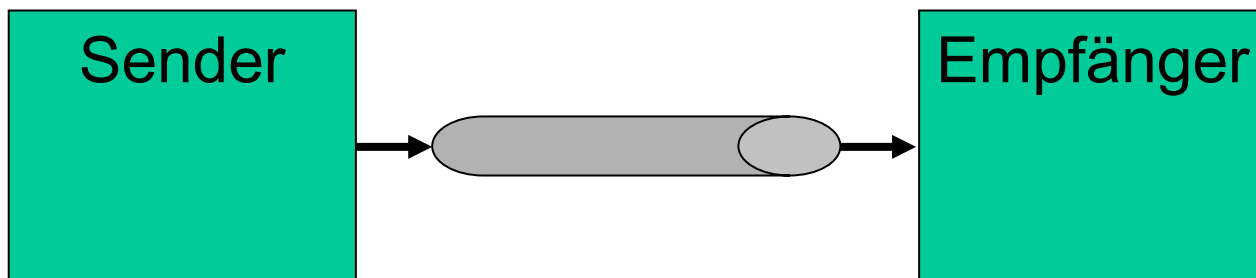
---

- Problem
  - Wie kann ein System mit einem anderen System kommunizieren?
- (bekannte) Lösung
  - Verbinde beide Systeme mit einem Nachrichten Kanal, in dem ein System Informationen hineinschreibt und das andere diese Information herausliest




# Message Channel (2/3)

- Beispiele
  - MQ, Sockets, FTP
- Kanal-Identifikation
  - Eindeutiger Name
- Identifikation der Endpunkt (Sender, Empfänger)
  - Zum Beispiel: IP Adressen+Ports, URL



# Message Channel (3/3)

---

- Pipe – Message Channel
  - Pipe kann durch Message Channel „implementiert“ werden, muss aber nicht
  - Jeder Message Channel ist eine Pipe
- Pipe: Pfeil
- Message Channel: 
- Notation geht vielleicht ab und zu durcheinander

# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

# Message Filter (1/4)

---

- Problem
  - Wie kann eine Komponente es vermeiden, uninteressante Nachrichten zu erhalten?
- Beispiel
  - Webshop: will nur Nachrichten von Produkte, die auch im Webshop verkauft werden.



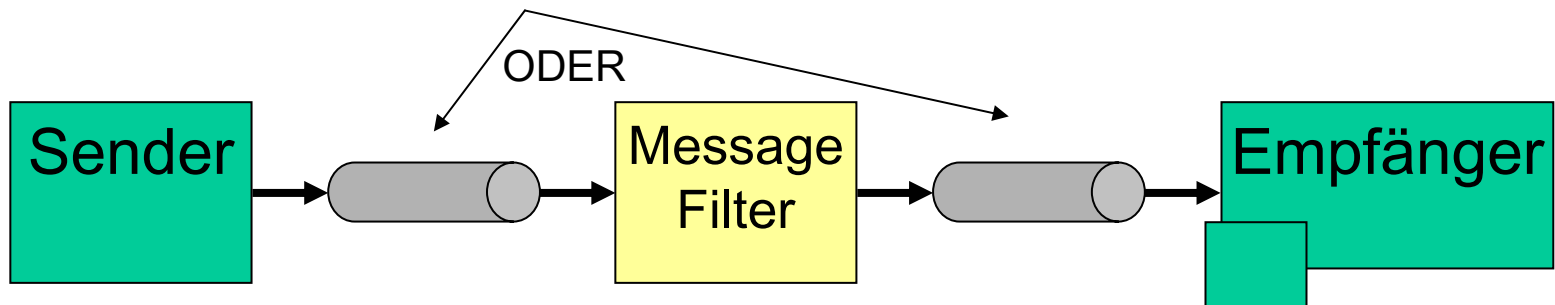
# Message Filter (2/4)

---

- Eine Lösung
  - Publish-Subscriber verwenden
  - **Subscribe oft nur nach Informationen im Header möglich, nicht nach Inhalt Nachricht**
  - **Header kann oft nur ja/nein Eigenschaften enthalten**
  - **Abfragefunktionalität der Header-Informationen limitiert**
- Weitere Lösung
  - Verschiedenen Kanäle verwenden
  - **Zu viele Kombinationen möglich: Zwei Produktkategorien Festnetztelefon + Mobiltelefon, drei Rabatte 5%, 10%, 15%. 2 x 3 mögliche Kombinationen also 6 Kanäle im schlimmsten Fall.**
- Noch eine Lösung
  - Komponente filtert Nachrichten selbst
  - **Bei unveränderlicher, gekaufter Software ist dies oft nicht möglich**

# Message Filter (3/4)

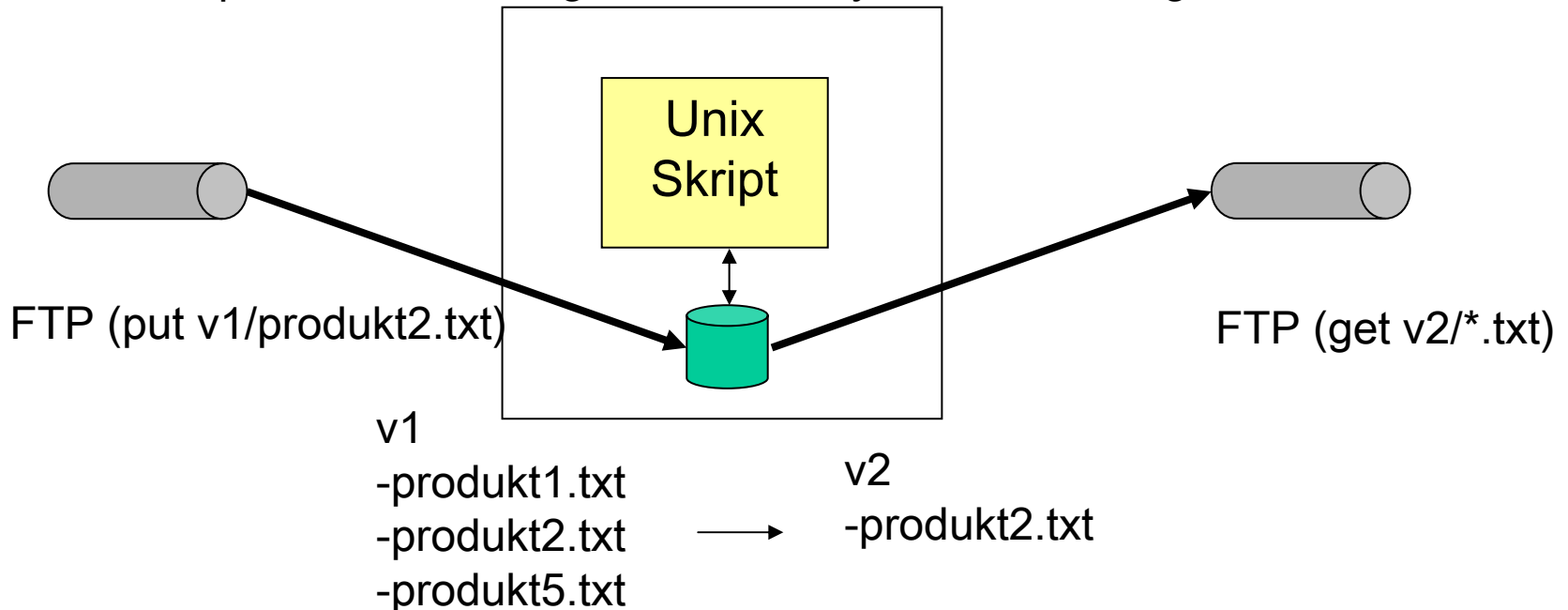
- Lösung
  - Verwende für die *gewünschte Komponente* einen speziellen Router, der ungewünschte Nachrichten filtert.
  - Filtern: „/dev/null“
  - Weiteren Kanal verwenden



# Message Filter (4/4)

## ■ Implementierungsbeispiel

- Nachrichten kommen via FTP in ein Verzeichnis v1
- Komponente holt Nachrichten via FTP aus einem Verzeichnis v2
- Separates Programm (Unix Skript mit Kombination aus find, rm und sed Skript) filtert Nachrichten durch Löschen der Dateien in v1 und kopiert sie in v2. Zugriff auf Dateisystem notwendig.



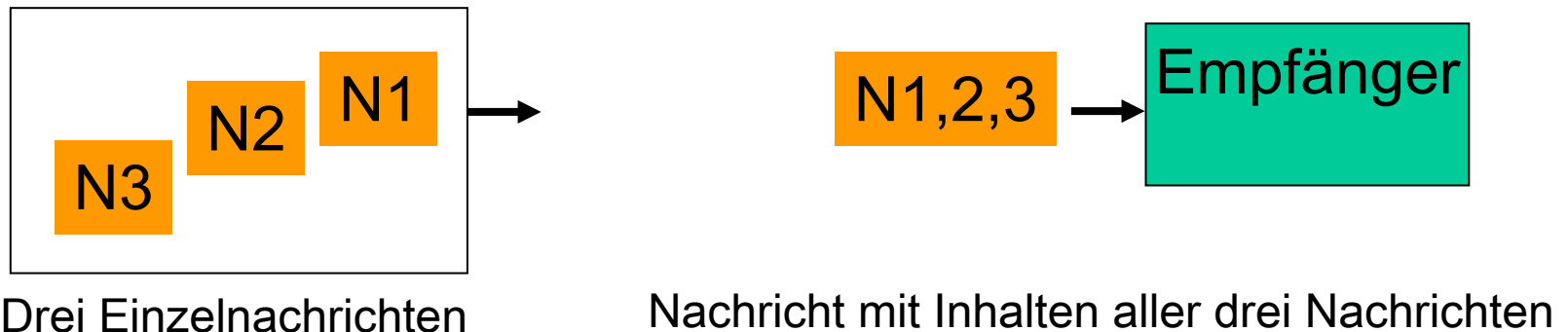
# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - **Aggregator**
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

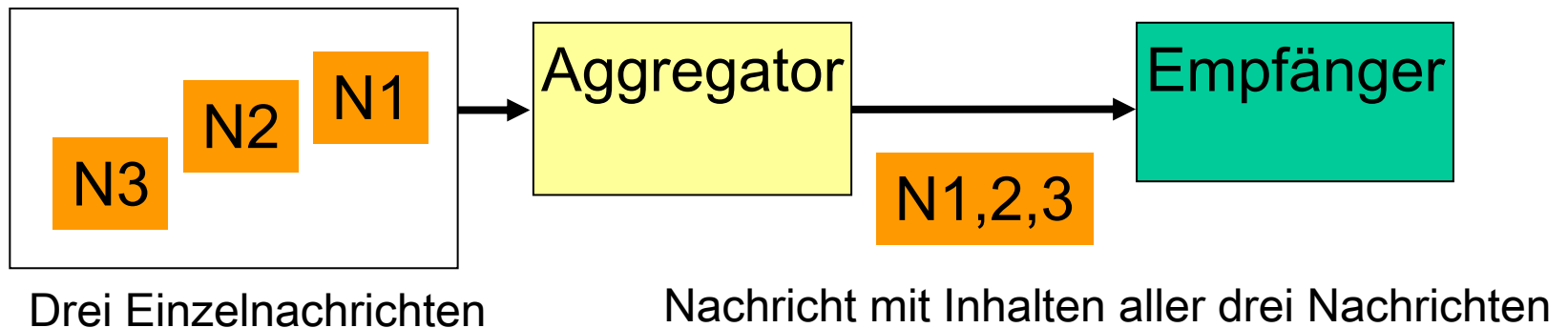
# Aggregator (1/6)

- Problem
  - Wie können mehrere, inhaltlich zusammenhängende Nachrichten als Ganzes verarbeitet werden.
- Beispiele
  - Das beste Preisangebot mehrere Hersteller verarbeiten.
  - Einem Kunden soll erst dann eine Gesamtrechnung gestellt werden, wenn alle Produkte ausgeliefert wurden.
  - Mehrere Produkte werden zu einem Paket (Bundle) geschnürt und zusammen unter anderen Konditionen verkauft.



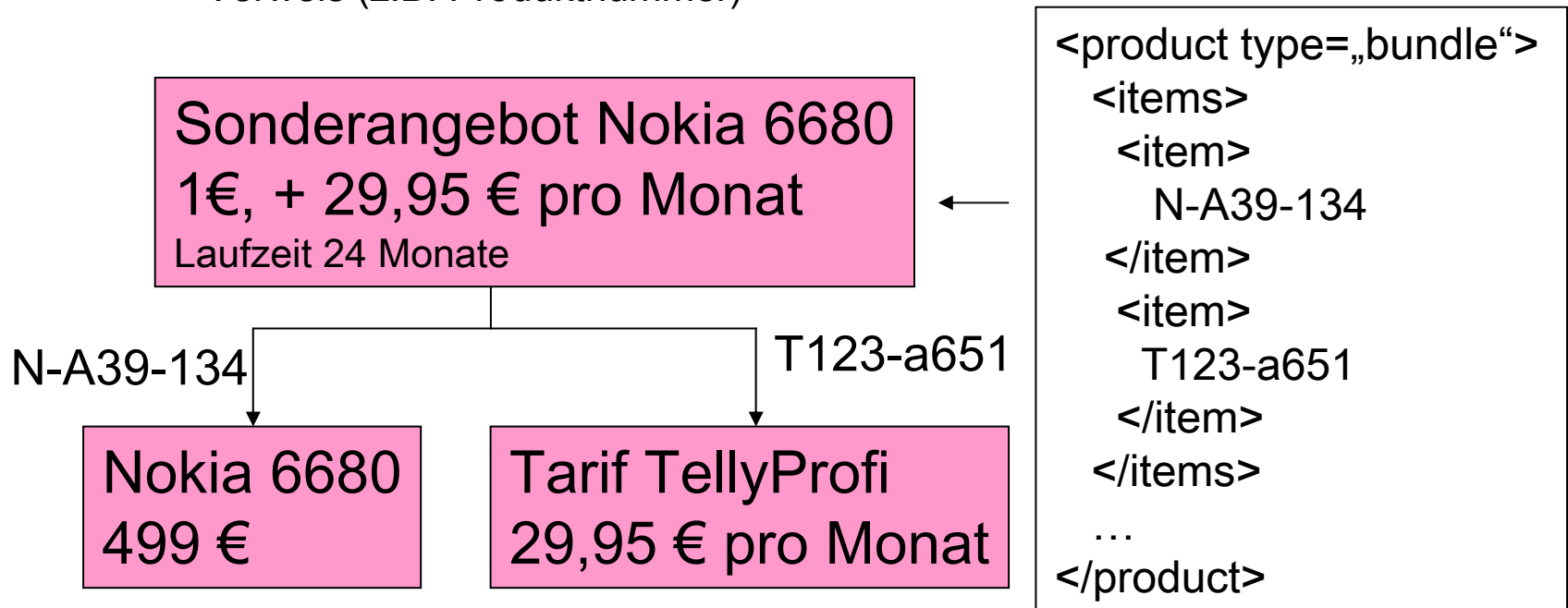
# Aggregator (2/6)

- Lösung
  - Verwende einen *zustandsabhängigen* Filter, der individuelle Nachrichten speichert, bis alle zusammenhängende Nachrichten empfangen wurden. Dann sendet der Aggregator eine aus den Informationen der Einzelnachrichten bestehende Nachricht.
- Folgende Fragen müssen beantwortet sein:
  - Welche Nachrichten gehören zusammen? (Korrelation)
  - Wann sind alle zusammengehörige Nachrichten empfangen? (Vollständigkeit)
  - Wie können die Nachrichten zusammengefügt werden? (Aggregation)



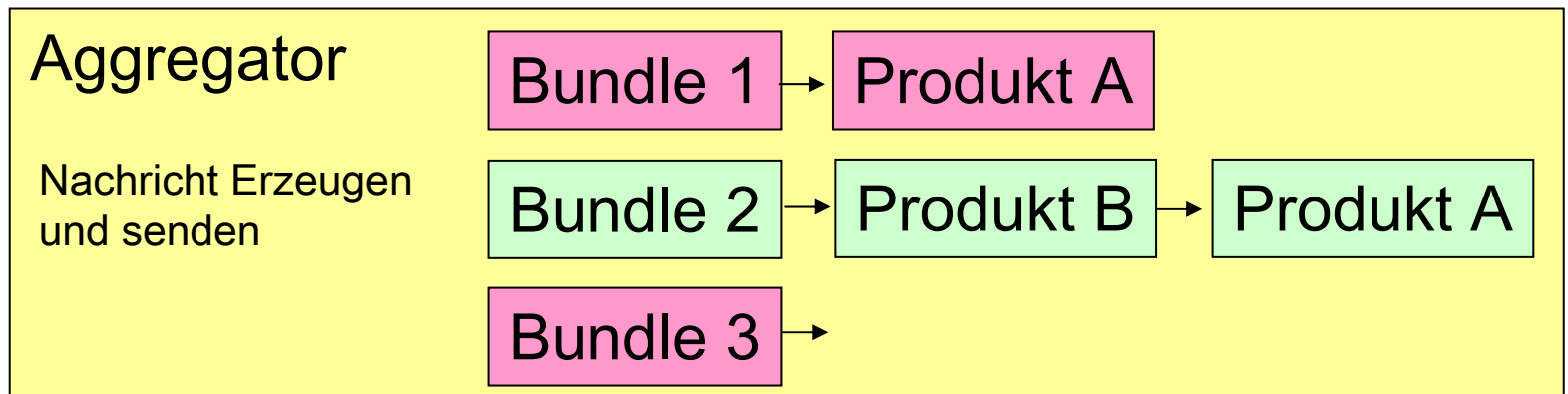
# Aggregator (3/6)

- Korrelation
  - Gemeinsame(n) Identifikator verwenden
- Beispiel: Produktnummer
  - Bundle: Ein Produkt kennt sein Bundle normalerweise nicht. Normalerweise ist Bundleinformation eine weitere Nachricht. (in R/3 z.B. als Stückliste)
  - Bundle ist selbst wieder ein Produkt, welches andere Produkte enthält, als Verweis (z.B. Produktnummer)



# Aggregator (4/6)

- Korrelation mit Produktnummern
- 1. Bundle wird immer vor Produkten gesendet
  - Für jedes Bundle eine Liste aufbauen mit Produktnummern. Eingehendes Produkt in zugehörigen Listen einfügen.
  - Wenn Liste voll ist, dann Gesamtnachricht senden.
- 2. Bundle wird irgendwann gesendet, auch nach Produkten
  - Alle Produkte eine Zeit lang zwischenspeichern Zeitperiode hängt von Geschäftsanforderungen ab
  - Produkte des Bundles senden, wenn Bundle gesendet wird (Quellsystem anpassen)
- Wie lange auf Produkte warten?



# Aggregator (5/6)

---

- Wie lange auf Nachrichten warten?
  1. Auf alle Warten (unbegrenzt)
  2. Ein bestimmte Zeit warten (Timeout)
  3. Nur die ersten Besten nehmen (z.B. Auktion, das beste Gebot)
  4. Signal von Außen (z.B. Handelsschluss an Börse)

# Aggregator (6/6)

---

- Konsequenzen
  - Zustand muss gespeichert werden
  - Je nach Strategie und Anzahl Nachrichten werden viel Ressourcen verbraucht (Speicher)
  - Korrelationsidentifikator nicht immer vorhanden
  - Erzeugen der neuen Nachricht aus den gespeicherten Informationen (nicht notwendigerweise direkt aus den ursprünglichen Nachrichten, also kein Message Translator)

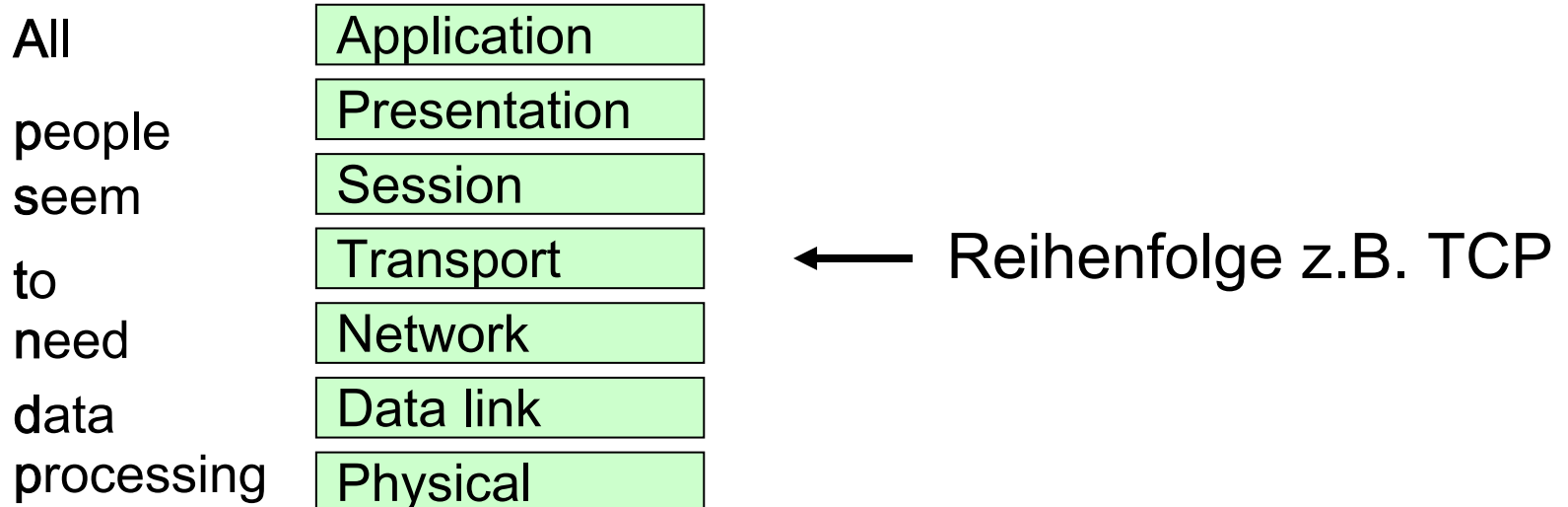
# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - **Resequencer**
- Monitoring & System Management
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

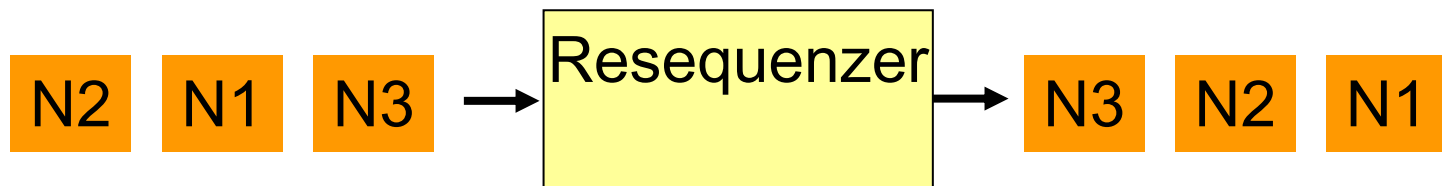
# Resequenzer (1/4)

- Problem
  - Wie kann eine Folge von Nachrichten, deren Reihenfolge durcheinander geraten ist, wieder in die richtige Reihenfolge gebracht werden?
- Beispiel
  - ISO/OSI Modell bei Paket-orientierter Vermittlung
  - Parallele identische Filter, können Reihenfolge zerstören



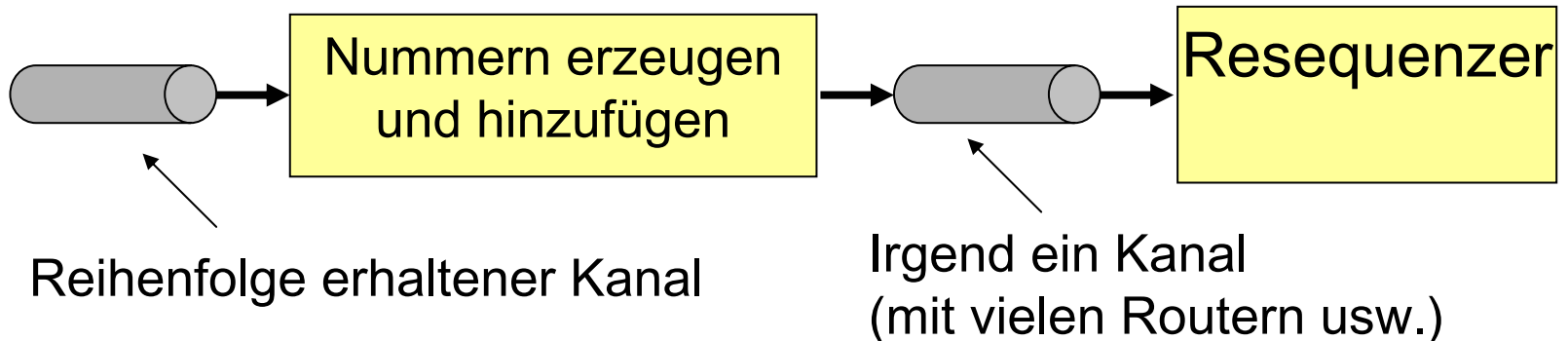
# Resequenzer (2/4)

- Lösung
  - Verwende eine *zustandsabhängigen* Filter, der überholende Nachrichten speichert, bis alle überholten Nachrichten empfangen wurden. Danach werden alle Nachrichten in richtiger Reihenfolge gesendet.
  - Resequenzer benötigt dazu für jede Nachrichten eindeutige Sequenznummern
- Kein Aggregator
  - Korrelationsidentifikation ist „zufällig“ (Produktnummer), braucht nicht eindeutig zu sein
  - Dieser wartet auf alle Nachrichten.
  - Nachrichten haben keine Reihenfolge.
  - Er publiziert eine aus den Nachrichten neu erzeugte Nachricht.



# Resequenzer (3/4)

- Etwas muss Nachrichten mit eindeutigen Sequenznummern anreichern
  - Nachrichten könnten von mehrere Sender kommen
  - Eindeutigkeit über mehrere Systeme notwendig
- Erzeugen mit MAC Adresse + aktuelle Zeit funktioniert nicht
  - Sprünge zwischen Sequenznummern (1, 7, 123)
- Einzelner, zentraler Zähler notwendig (Filter)



# Resequenzer (4/4)

---

- Implementierung
  - Zähler in einer Datenbank speichern und erhöhen
  - Zähler in einem statefull Session Bean (Beginn Zähler z.B. aktuelle Zeit in ms, da Zähler nicht persistent).
- Konsequenzen
  - Zähler kann zum Flaschenhals werden
  - Zwischenspeichern kostet Ressourcen (Buffer overrun Problem)

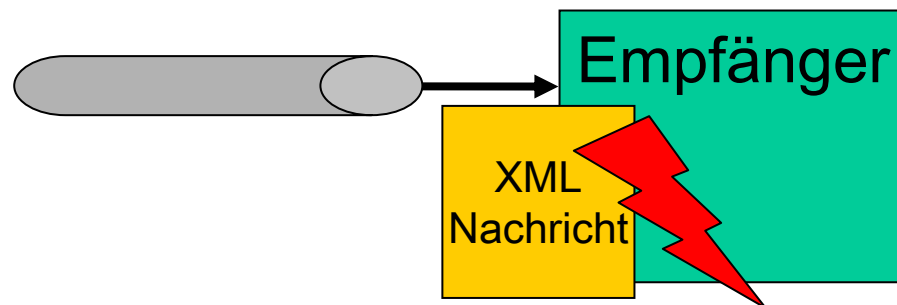
# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- **Monitoring & System Management**
  - Invalid Message Channel
  - Dead Letter Channel
  - Test Message

# Invalid Message Channel (1/5)

- Problem
  - Wie kann ein Empfänger Nachrichten sinnvoll behandeln, die keinen Sinn ergeben?
- Beispiele
  - Inhalt der Nachricht kann nicht korrekt interpretiert werden (Wert eines Datenfelds hat ein nicht erwartetes Format, z.B. kein wohlgeformtes XML)
  - Nachricht ist in nicht mehr unterstütztem XML Dialekt formuliert, das nach Update der Empfängersoftware nicht mehr gelesen werden kann



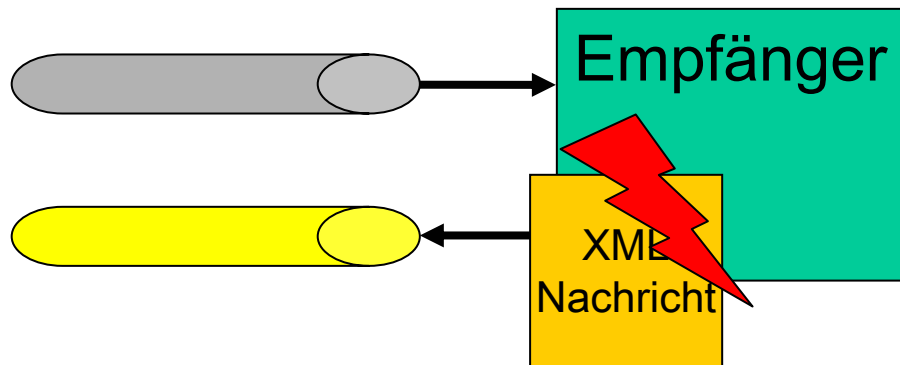
# Invalid Letter Channel (2/5)

---

- Keine Lösungen:
  1. Eintrag in ein Logfile und Nachricht einfach „vergessen“
    - Inhalt kann wichtig gewesen sein
    - Für Sender ist Verarbeitung abgeschlossen (asynchron)
  2. Nachricht wieder zurücksenden
    - Absender vielleicht gar nicht bekannt (z.B. bei publish-subscriber)
    - Absendersystem kann Fehler nicht beheben
- Offenbar muss ein Mensch die Nachricht untersuchen, da vermutlich ein Programm- oder Systemfehler vorliegt

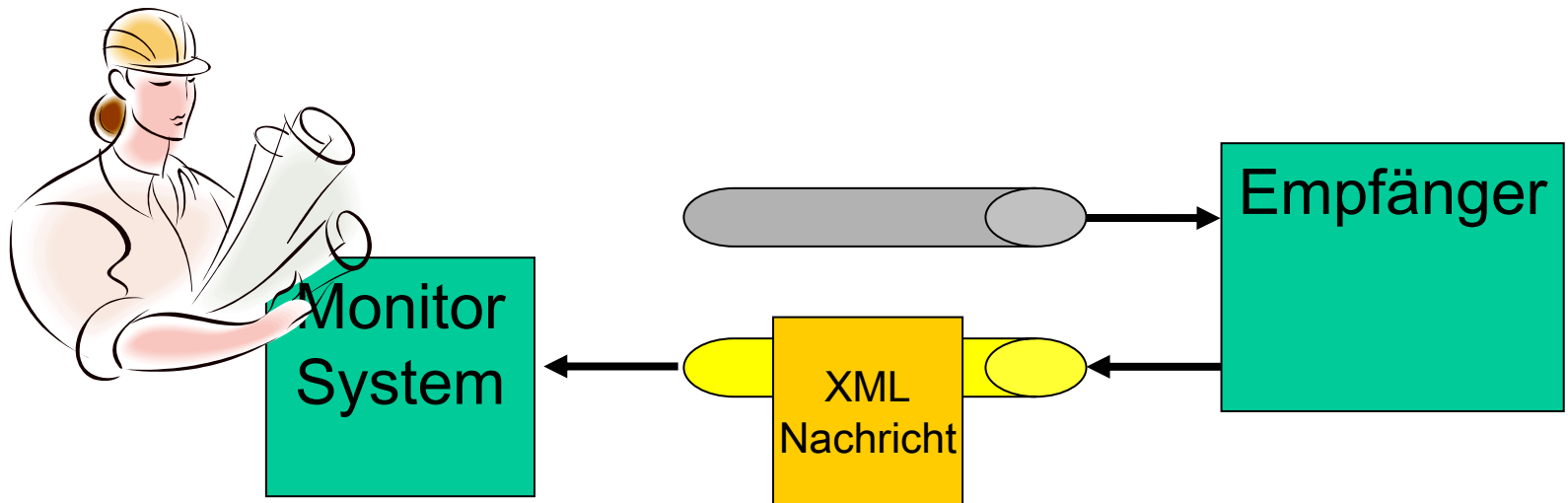
# Invalid Letter Channel (3/5)

- Lösung
  - Sende Nachrichten, die nicht verarbeitet werden können, in einen speziellen Kanal für ungültige Nachrichten (*invalid letter channel*)
- Was soll dort damit getan werden?

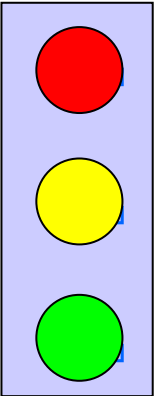


# Invalid Letter Channel (4/5)

- Kanal muss überwacht werden
  - Wenn Nachricht dort hin gesendet wurde, dann muss ein Mensch alarmiert werden
  - Mensch muss die Nachricht ansehen, sie analysieren und entscheiden, was damit getan werden soll
- Sofortiger Alarm oft nicht wünschenswert
  - Verfügbarkeitsanforderungen entscheidend
  - Asynchrone Übermittlung (auf ein paar Stunden kommt es nicht an)
  - Hohe Kosten, wenn Sa/So immer jemand nachts zum Notdienst muss

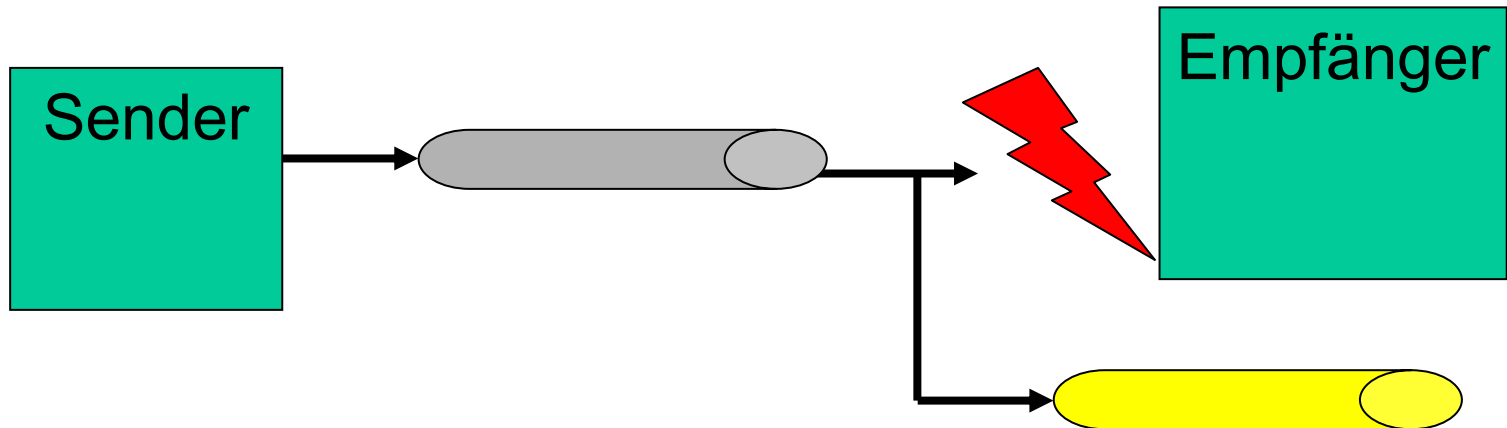


# Invalid Letter Channel (5/5)

- Swisscom (Error queue)
    - Ampelmodell
    - Überwachungsprogramm hat zwei Schwellwerte für Anzahl Nachrichten in Error queue
- 
- Rot: 2. Schwellwert überschritten, es wird jemand alarmiert
  - Gelb: 1. Schwellwert überschritten, es wird noch nicht alarmiert
  - Grün: 1. Schwellwert noch nicht erreicht
- Zu jeder (asynchronen) Queue muss eine Error queue eingerichtet werden
  - Empfänger stellt Nachrichten dort hinein

# Dead Letter Channel (1/1)

- Variante
  - Was soll mit Nachrichten geschehen, die nicht an Empfänger übermittelt werden konnten?
- Beispiel
  - Empfänger ist off-line
- Lösung
  - Sende Nachricht an einen Dead Letter Channel



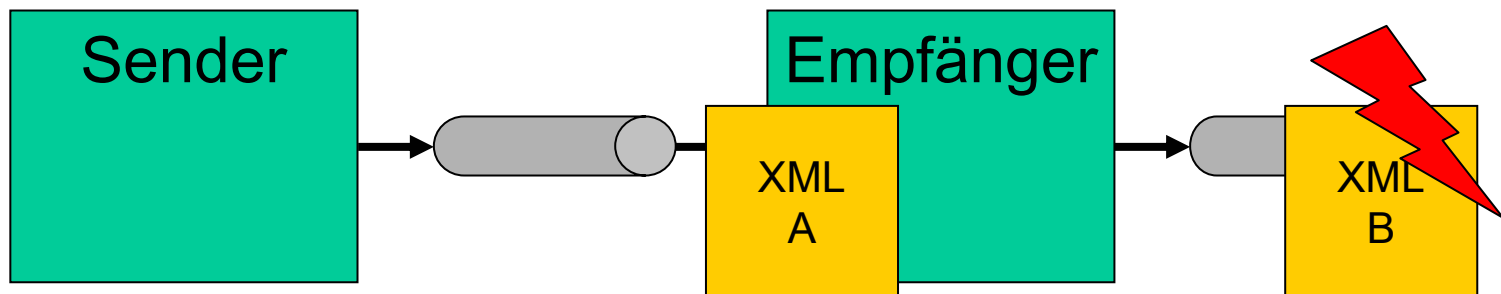
# Inhalt

---

- Überblick
- Übergeordnete Entwurfsmuster
  - Pipes and Filters
  - Message Router
  - Transformation
  - Beispiel
- Message Channel
- Message Routing
  - Message Filter
  - Aggregator
  - Resequencer
- Monitoring & System Management
  - Invalid Message Channel
  - **Dead Letter Channel**
  - Test Message

# Test Message (1/9)

- Problem
  - Eine Komponente eines System verarbeitet Nachrichten, aber zugehörige ausgehende Nachrichten sind fehlerhaft aufgrund interner Fehler. Wie kann diese Situation entdeckt werden?
- Beispiel
  - System hat periodisch zu wenig Hauptspeicher und verursacht deswegen Fehler
  - Daten in der Datenbank stimmen nicht mehr mit Daten in der Nachricht überein.



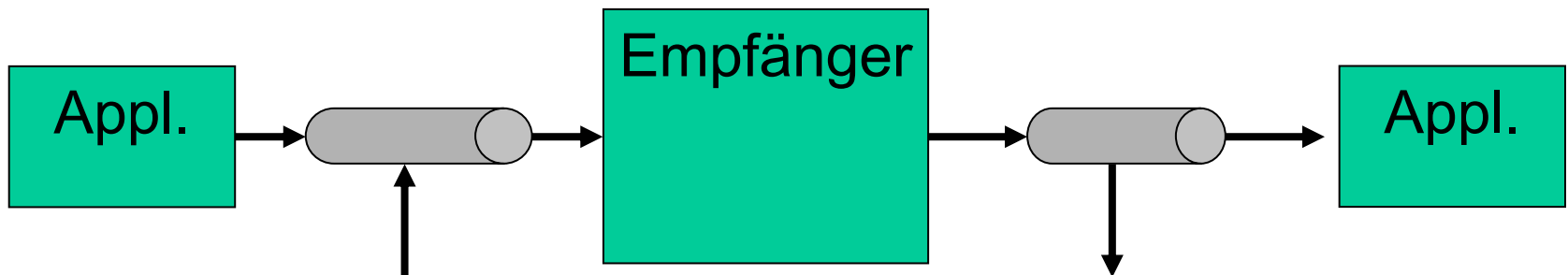
# Test Message (2/9)

---

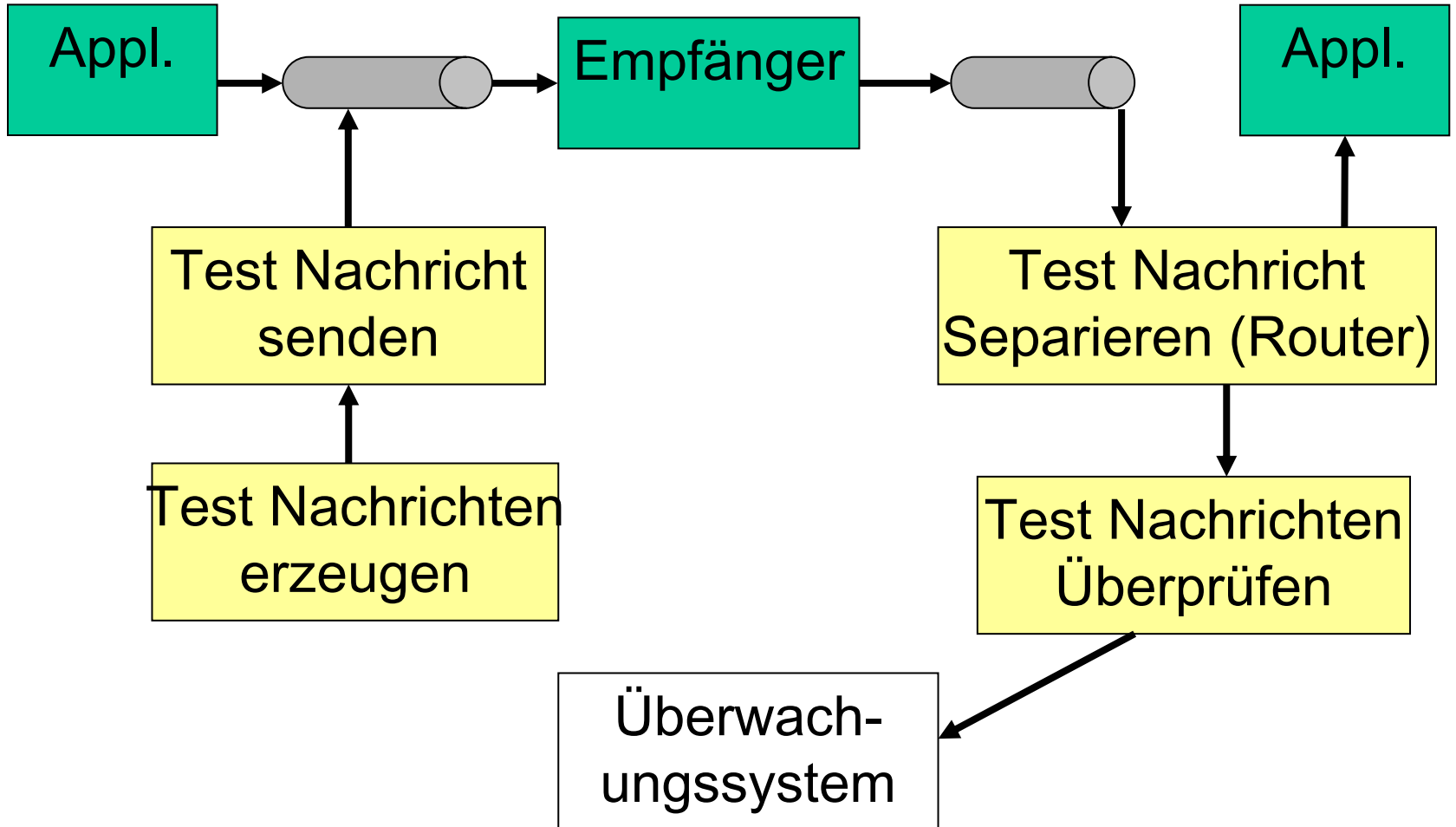
- Problem (etwas anders formuliert)
  - Die Methoden einer Klasse/Objekts einer Software können aufgerufen werden, die zurückgebenden Werte stimmen aber nicht mehr. Wie kann diese Situation entdeckt werden?
- Beispiele
  - Teile der Software von denen die Klasse abhängt haben sich geändert
  - Die Klasse ist geändert worden und hat Bugs.
- Lösung?

# Test Message (3/9)

- Lösung (Regressionstest)
  - Sende eine Test Nachricht und überprüfe den Inhalt der resultierende Folgenachricht.
  - Aber wie kann die Test Nachricht erzeugt, eingeschleust, herausgefiltert und ausgewertet werden?



# Test Message (4/9)

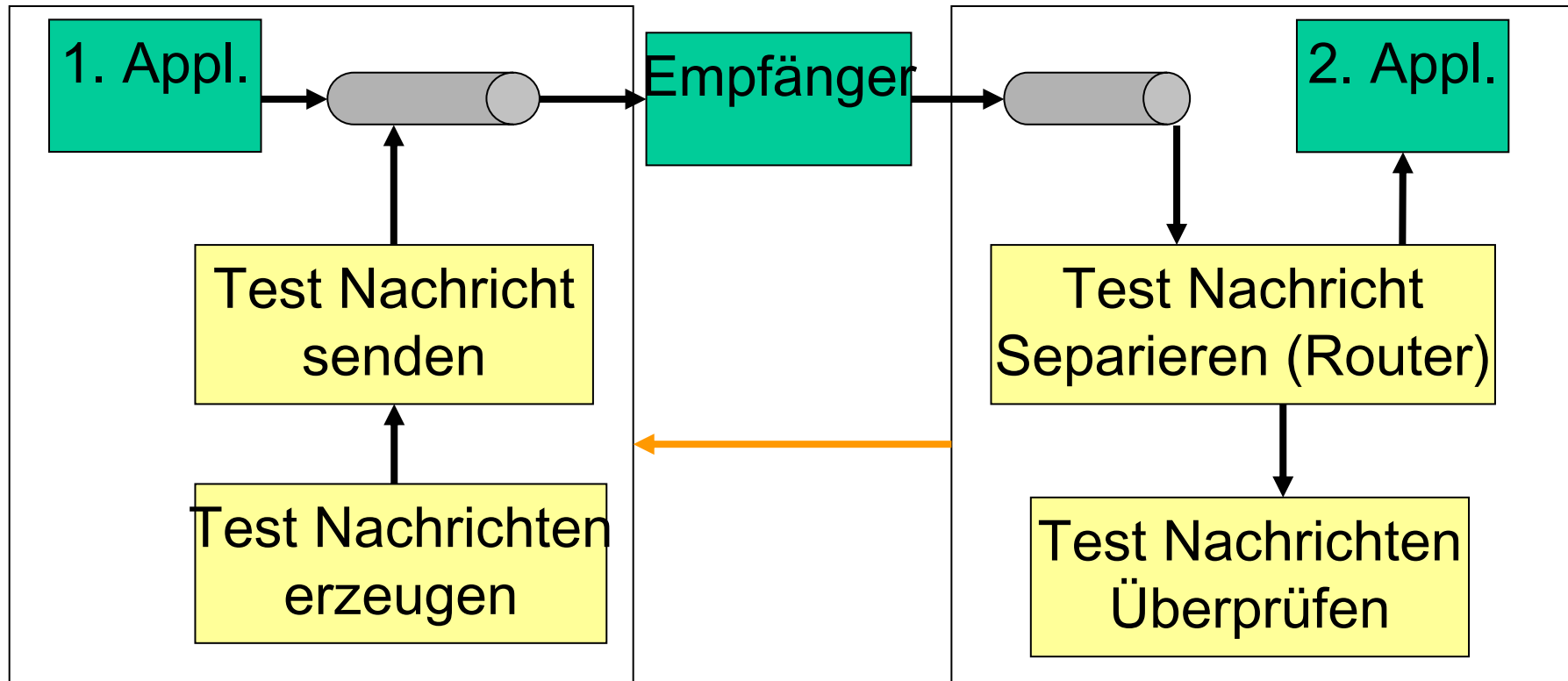


# Test Message (5/9)

---

- Wo sollen die Komponenten implementiert und verteilt werden?
  - Bei 1. Appl., die eigentliche Nachrichten sendet?
  - Bei Empfänger, der Antworten für 2. Appl. Erzeugt?
  - Bei 2. Appl. die Antworten einliesst?
  - Woanders.

# Test Message (6/9)



- 1. Appl. hat Daten, um Nachrichten zu erzeugen
- 2. Appl. hat Daten, um Nachrichten zu überprüfen
- Aber 2. Appl. ist dann stärker an 1. Appl. gekoppelt (und auch etwas in umgekehrter Richtung)

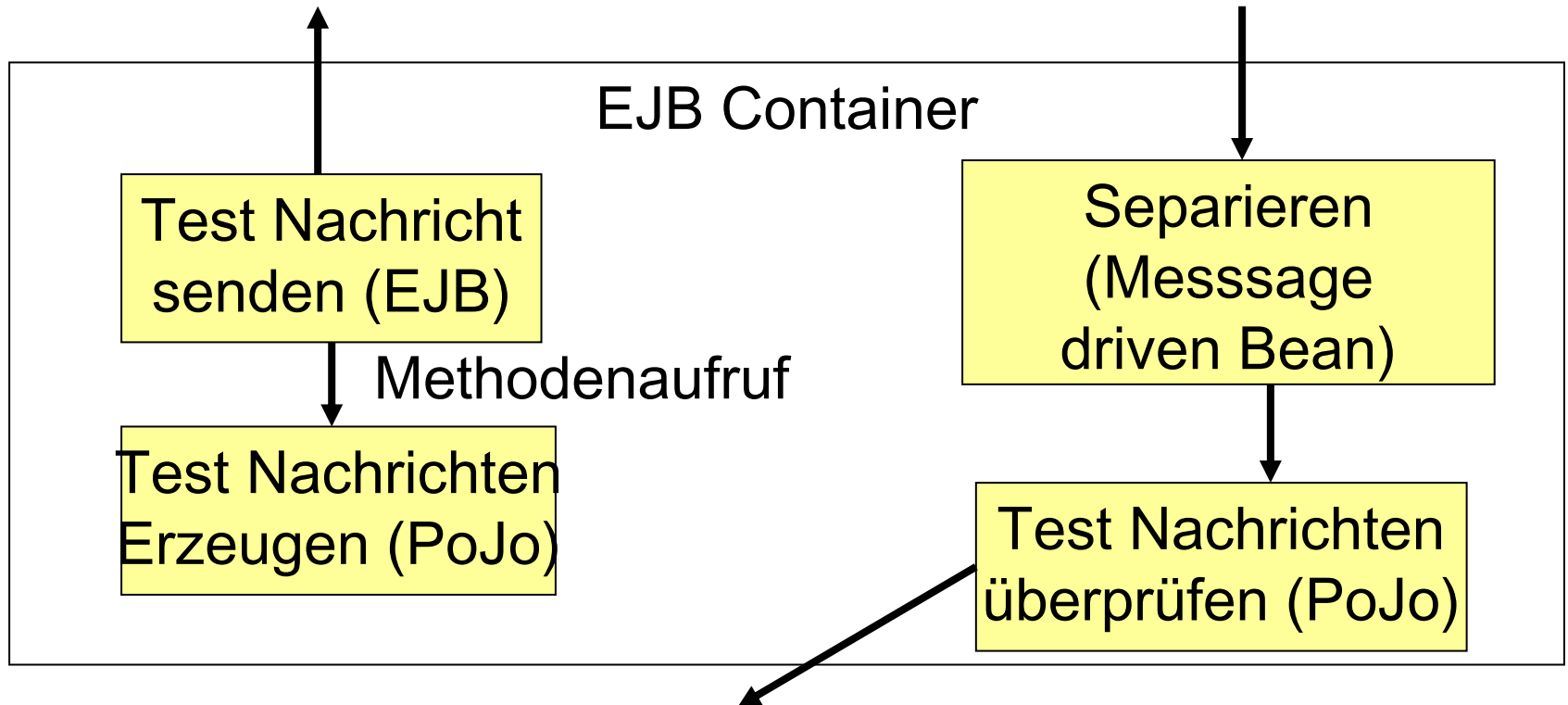
# Test Message (7/9)

---

- Fiktive einfache, heterogene Implementierung
- Testgenerierung nah bei 1. Appl.
  - Testgenerierung: Feste Nachrichten als Datei
  - Testnachrichten senden: Dateien in die Queue stellen mit zusätzlicher Headerinformation „test“ (Herstellerprogramm von MQ Anbieter aus Kommandozeile aufrufen, periodisch Aufrufen)
- Überprüfung nah bei 2. Appl.
  - Separierung: Subscriber, der nur Nachrichten mit „test“ haben will.
  - Test Nachrichten Überprüfen: Nachricht mit vorgefertigten Testergebnissen gegeben als Datei vergleichen (Skript diff, grep unter Unix)

# Test Message (8/9)

- Fiktive homogenen Lösung mit J2EE
  - Eigenständige Applikation
  - Beide Applikation nicht stark gekoppelt
  - Auch Publish-subscriber



# Test Message (9/9)

---

- Konsequenzen
  - Aktive Überwachung einzelner *Komponenten* im Nachrichtenfluss
  - Zusätzliche Last für Empfänger
  - Testnachrichten nur sporadisch Senden im Vergleich zu Rest der Nachrichten
  - Zusätzlicher Aufwand in Entwicklung und Wartung zusätzlicher, verteilter Komponenten
  - Empfänger muss zwischen Testdaten und produktiven Daten unterscheiden können
- Oft Invalid Letter Channel ausreichende Lösung