



## Was sind Web Services?

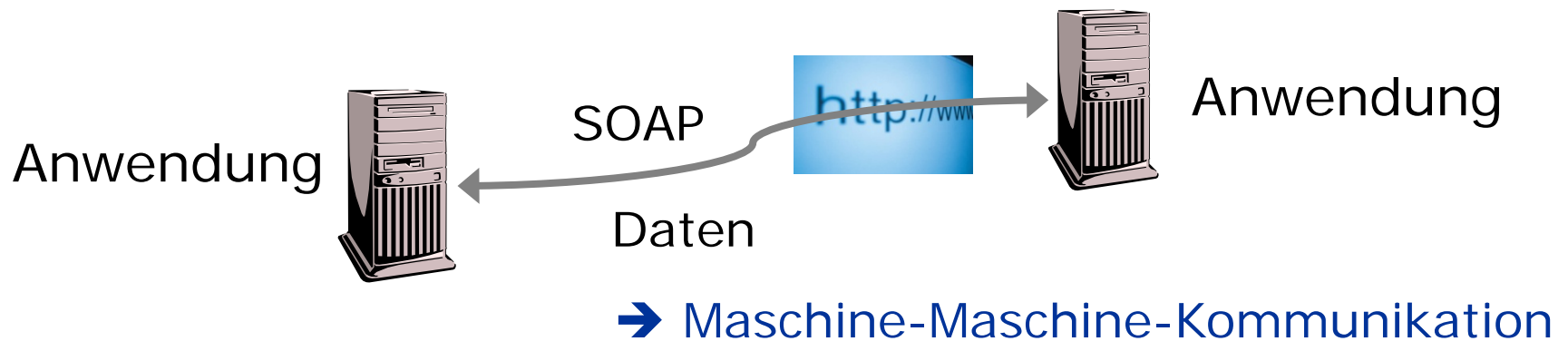
Marko Harasic

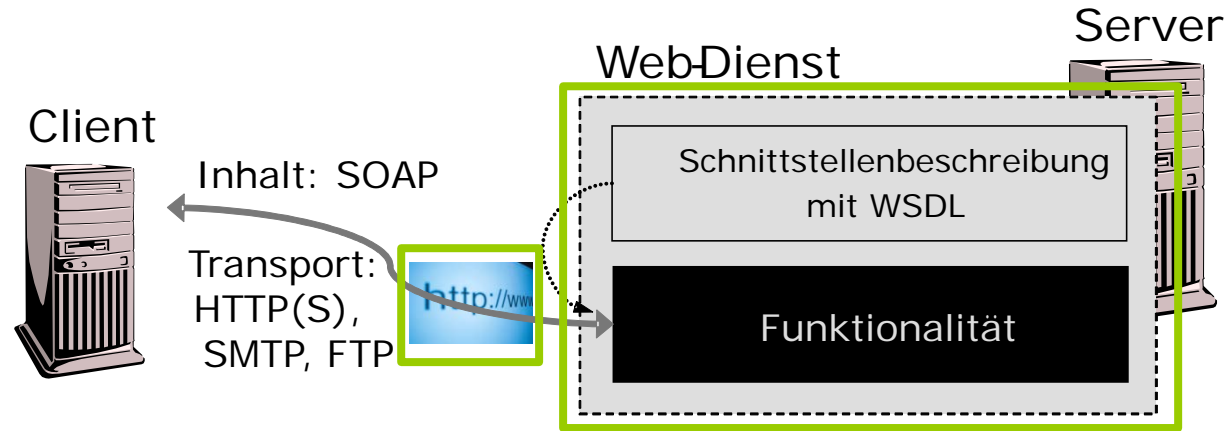
Freie Universität Berlin  
Institut für Informatik  
Netzbasierte Informationssysteme  
[harasic@inf.fu-berlin.de](mailto:harasic@inf.fu-berlin.de)

## traditionelle Web-Anwendung



## Web Service





Ein **Web Service** ist eine **Softwareanwendung**, die

1. mit einer **URI** eindeutig identifizierbar ist,
2. über eine **WSDL**-Schnittstellenbeschreibung verfügt,
3. nur über die in ihrer WSDL beschriebenen Methoden zugreifbar ist und
4. über **gängige Internet-Protokolle** unter Benutzung von XML-basierten Nachrichtenformaten wie z.B. **SOAP** zugreifbar ist.

## Web Services

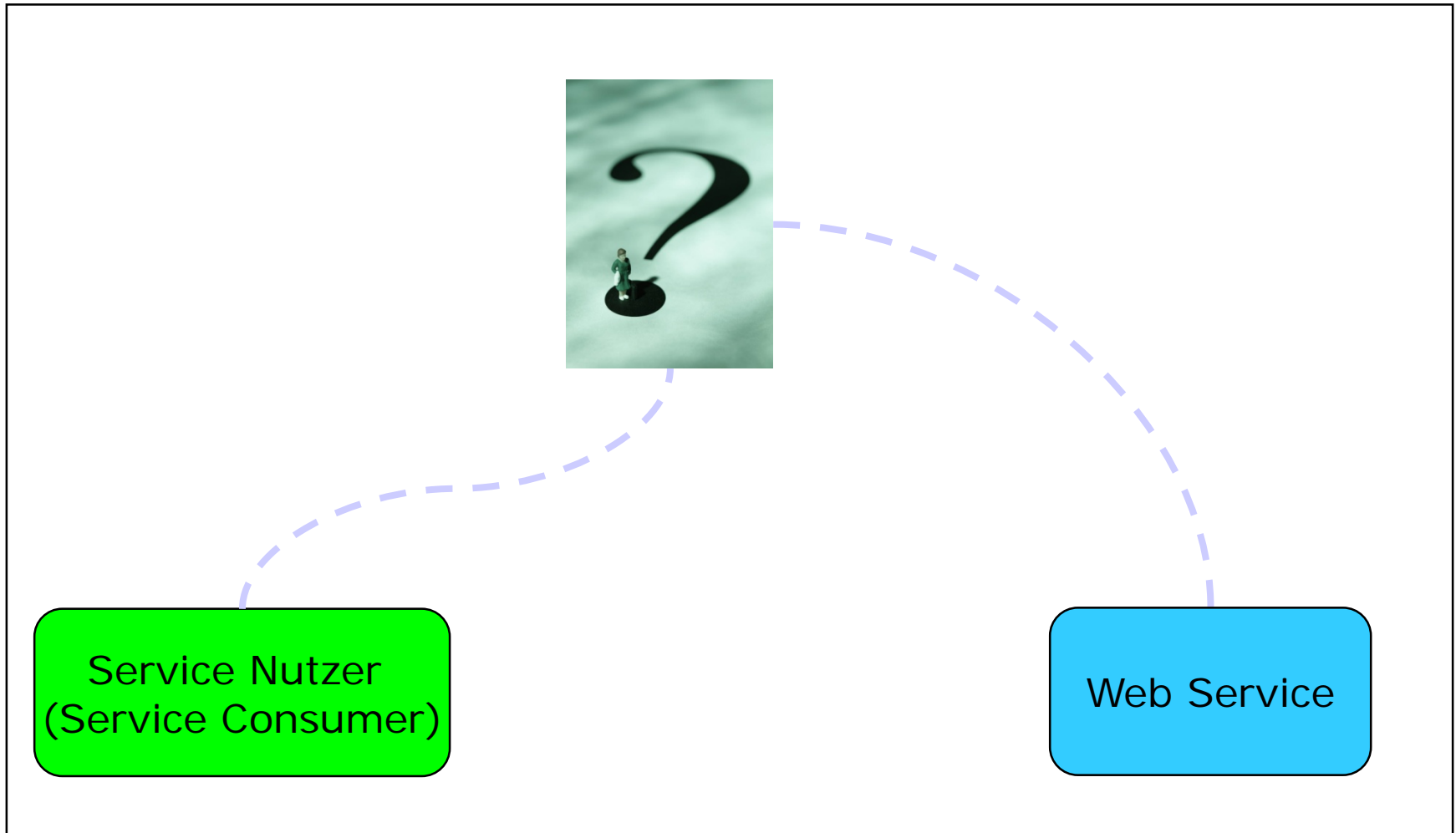
- implementieren häufig keine neuen Systeme, sondern sind **Fassade** für bestehende Systeme
- abstrahieren von Programmiersprache und Plattform mit der die Anwendung realisiert ist:  
**Virtualisierung von Software**
- zwei Erscheinungsformen:
  - **RPCs** (synchron)
  - **Messaging** (asynchron)

Layer	Protokoll/Standards
Messaging	HTML
Transport	HTTP, FTP, SMPT
Network	TCP/IP, UDP

<b>Layer</b>	<b>Protokoll/Standards</b>
Content	Content Information
Messaging	SOAP, XML-RPC
Transport	HTTP, FTP, SMTP
Network	TCP/IP, UDP

# Web Service Technology Stack

<b>Layer</b>	<b>Protokoll/Standards</b>
Discovery <i>(holt Service-Beschreibung von Prodiver)</i>	UDDI, DISCO, WSIL
Description <i>(beschreibt Services)</i>	WSDL
Messaging	SOAP, XML-RPC
Transport <i>(Applikation-zu-Applikation Kommunikation)</i>	HTTP, FTP, SMPT
Network <i>(Netzwerk Layer)</i>	TCP/IP, UDP



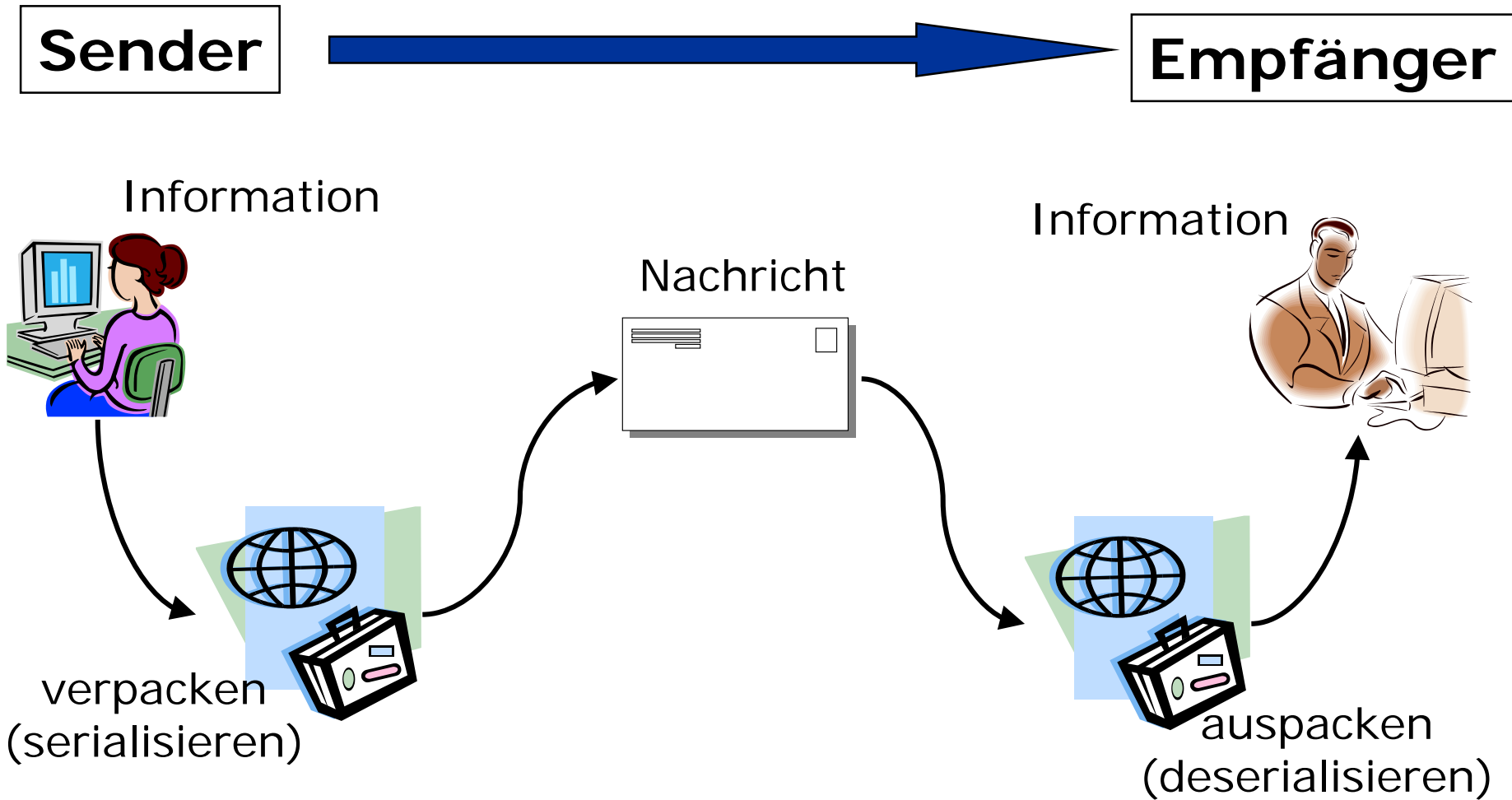




## Web Service Basiskomponenten: *SOAP*

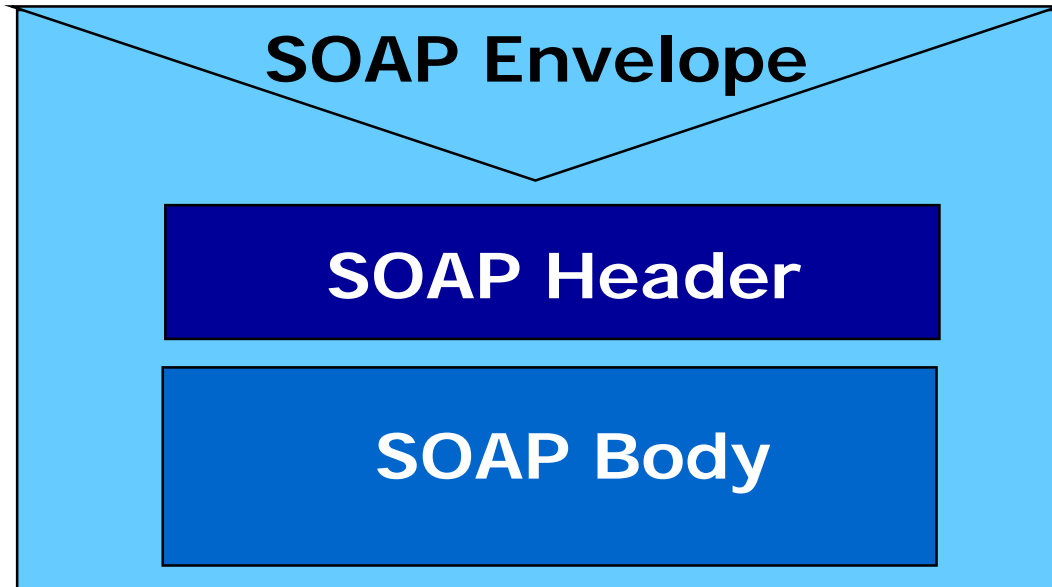


# Austausch einer SOAP-Nachricht



- ...eine **Kommunikationskomponente** von Web Services
- ...ein **Protokoll** für Nachrichtenaustausch zwischen Web Service-Konsument und Web Service-Anbieter
- ...**XML-basiert** (nutzt XML für die Darstellung von Nachrichten)
- ...Plattform- & Programmierspracheunabhängig

Die SOAP-Spezifikation legt fest, wie eine Nachricht übertragen wird. Die Umsetzung der Nachricht ist nicht Gegenstand der SOAP-Spezifikation



XML-Deklaration

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap/envelope/">
```

```
<!-- SOAP Header -->
```

```
<!-- SOAP Body -->
```

```
</env:Envelope>
```

SOAP Version 1.2

SOAP Version 1.1:  
<http://schemas.xmlsoap.org/soap/envelope/>

## HTML

```
<html>
  <head>
    Zusatzinformationen
  </head>
  <body>
    Inhalt: Webseite
  </body>
</html>
```

## SOAP

```
<Envelope>
  <Header>
    Zusatzinformationen
  </Header>
  <Body>
    Inhalt: XML-Daten
  </Body>
</Envelope>
```

- XML-basierter W3C-Standard
  - SOAP 1.1: W3C Note von 2000
  - SOAP 1.2: W3C Recommendation von 2003

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<env:Envelope
```

```
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
```

```
>
```

```
<env:Header>
```

*Zusatzinformationen*

```
</env:Header>
```

```
<env:Body>
```

*Nachrichtinhalt*

```
</env:Body>
```

```
</env:Envelope>
```

- Wurzel-Element: Envelope aus SOAP-Namensraum
- Header: optional
- Body: obligatorisch
- Im Beispiel: kein W3C-Namensraum → SOAP 1.1

SOAP Nachricht → ein XML Dokument das beinhaltet:

- obligatorisches **Envelope Element** – identifiziert ein XML Dokument als SOAP Nachricht
- optionales **Header Element** – Header Informationen
- obligatorisches **Body Element** – Call & Response Informationen
- optionales **Fault Element** – Informationen über Fehler

```
<?xml version="1.0"?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-
  envelope">
  ...
</env:Envelope>
```

■ Im Beispiel: W3C-Namensraum  
→ SOAP 1.2

- Name des Elements: Envelope
- Envelope: Wurzel-Element einer SOAP Nachricht
- beinhaltet SOAP Namespace
- identifiziert SOAP Nachricht



```
<env:Envelope ...>
  <env:Header xmlns:ns="URI" >
    <ns:Zusatzinformation-1>...</ns:Zusatzinformation-1>
    ...
    <ns:Zusatzinformation-n>...</ns:Zusatzinformation-n>
  </env:Header>
  <env:Body>...</env:Body>
</env:Envelope>
```

**Header  
Blöcke**

- **Header:** beliebige XML-Inhalte erlaubt
- **Struktur von Anwendung festgelegt**
- **Header Block**
  - Kind-Element von Header
  - Zusatzinformation zur eigentlichen Nachricht

```
<env:Envelope ...>  
  <env:Body xmlns:ns="URI">  
    <ns:Nachrichtinhalt-Teil-1>...</ns:Nachrichtinhalt-Teil-1>  
    ...  
    <ns:Nachrichtinhalt-Teil-n>...</ns:Nachrichtinhalt-Teil-n>  
  </env:Body>  
</env:Envelope>
```

- **Body**: beliebige XML-Inhalte erlaubt
- Struktur von Anwendung festgelegt, z.B. durch:
  - speziellen Namensraum und/oder
  - WSDL-Beschreibung

```
<env:Envelope ...>
  <env:Header>
    <alertcontrol xmlns="http://example.org/alertcontrol">
      <priority>1</priority>
      <expires>2007-06-24T14:00:00-05:00</expires>
    </alertcontrol>
  </env:Header>
  <env:Body>
    <alert-msg xmlns="http://example.org/alert">
      Pick up Mary at 2pm!
    </alert-msg>
  </env:Body>
</env:Envelope>
```

Zusatz-information  
(Header Block)

Nachricht

```
<env:Header>  
  <alertcontrol xmlns="http://example.org/alertcontrol"  
    env:mustUnderstand="true" >  
    ...  
  </alertcontrol>  
</env:Header>
```

- **mustUnderstand="true"**: Empfänger muss Header Block verstehen oder mit Fehlermeldung antworten
- **mustUnderstand="false"**: Empfänger kann Header Block (ohne Fehlermeldung) ignorieren
- kann für jeden Header Block unterschiedlich sein
- Beachte: Standard-Wert ist "false"

## Empfänger **muss verarbeiten**:

- Body
- Header Blocks mit `mustUnderstand="true"`

## Empfänger **darf ignorieren**:

- Header Blocks mit `mustUnderstand="false"`
  - Header Blocks ohne `mustUnderstand`-Attribut
- Grund: "false" Standardwert von `mustUnderstand`

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="..." xmlns:xsi="...">
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key xsi:type="xsd:string">3289754870548097</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
      ...
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

- **doGoogleSearch**(key, q, start, maxResults,...)
- hier kein Header
- Beachte: **Web Service-Aufruf kann als URL kodiert werden (REST)**

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="..." xmlns:xsi="...">
  <env:Body>
    <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" ...>
      <return xsi:type="ns1:GoogleSearchResult">
        ...
      </return>
    </ns1:doGoogleSearchResponse>
  </env:Body>
</env:Envelope>
```

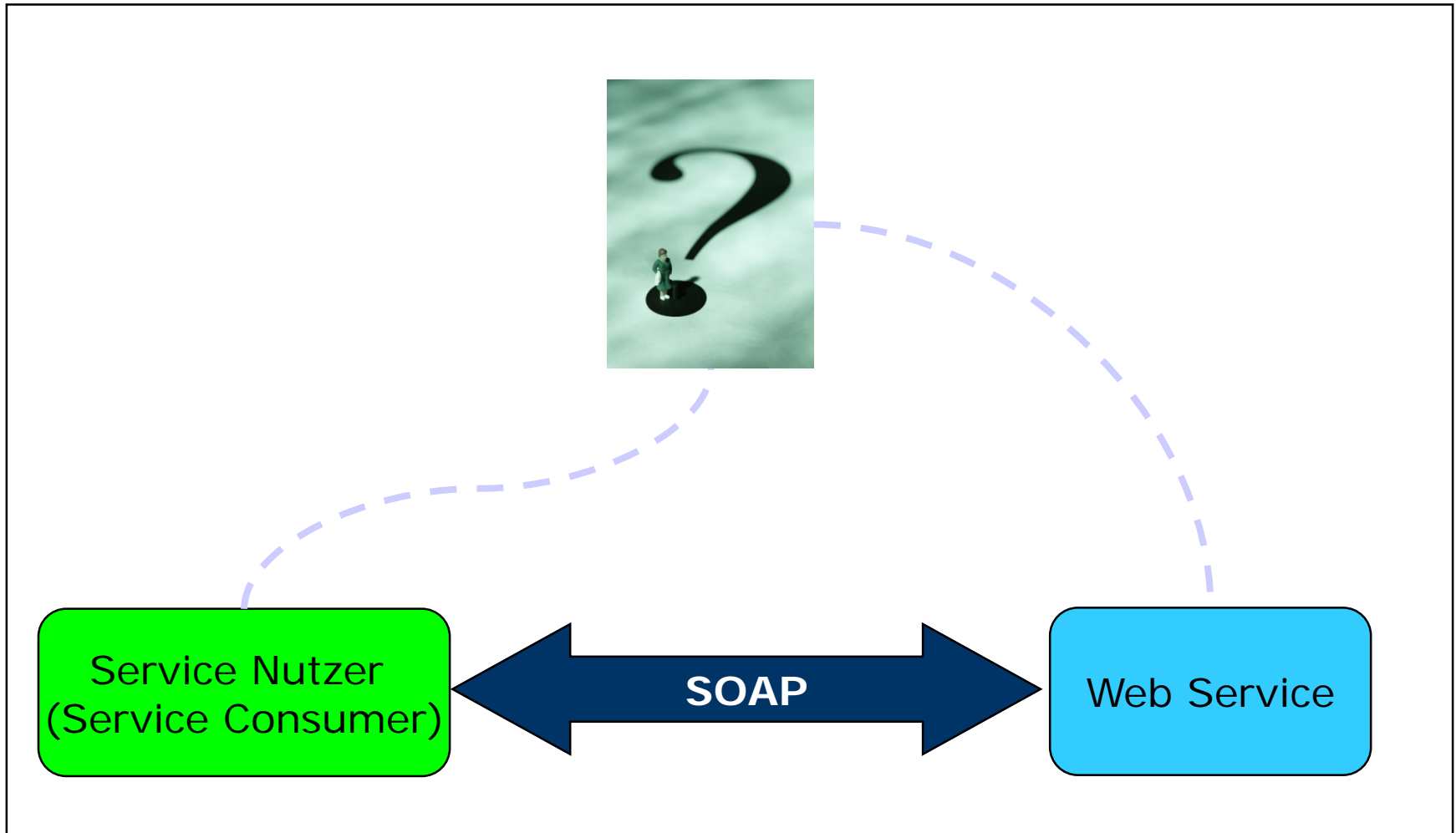
- Antwort: **doGoogleSearchResponse**(return(...))
- Datentyp ns1:GoogleSearchResult in WSDL-Beschreibung definiert

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="..." xmlns:xsi="...">
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key xsi:type="xsd:string">3289754870548097</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
      ...
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

- **Was ist eine URN?**
  - **Ortsunabhängiger, globaler, lebenslanger Identifier**
  - **Übersetzung in URI übernimmt Anwendung**



- heute meist über HTTP oder HTTPS
- Request-Response-Verhalten von HTTP unterstützt entfernte Prozeduraufrufe (RPCs)
- auch z.B. mit SMTP (Messaging) möglich



- + unabhängig von Übertragungsprotokollen
- + sowohl für RPCs als auch für Messaging geeignet
- + einfach erweiterbar
- + Erweiterungen unabhängig voneinander
- + Plattformunabhängig
- + Programmiersprachenunabhängig

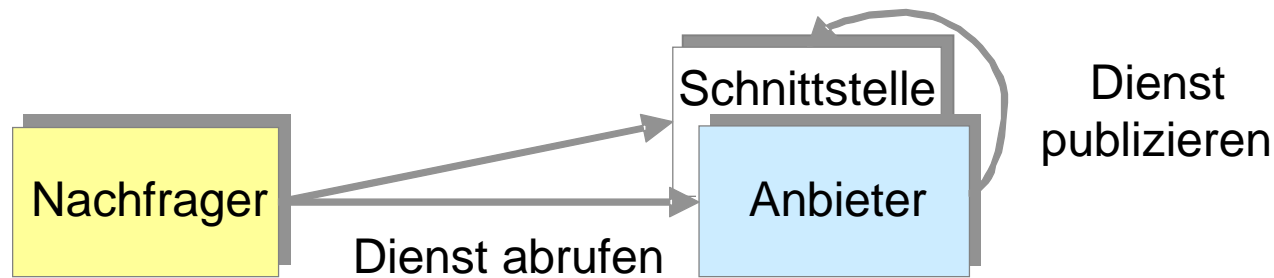
- zusätzlicher Verarbeitungsaufwand
- nicht so einfach zu erlernen
- für viele notwendige Erweiterungen noch kein etablierter Standard

Beispiel: wsu:identifier vs. wsa:MessageID

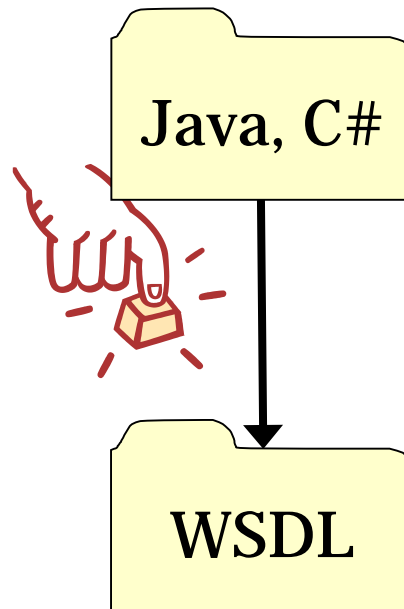


## Web Service Basiskomponenten: *WSDL*

## Formale Beschreibung der Schnittstelle von Services



- Client möchte bestimmten Web Service nutzen
- Client benötigt hierfür:
  - Struktur des Aufrufes: Name, Parameter, Ergebnis, Fehlermeldungen
  - Übertragungsprotokoll und Web-Adresse
- genau dies wird mit WSDL beschrieben



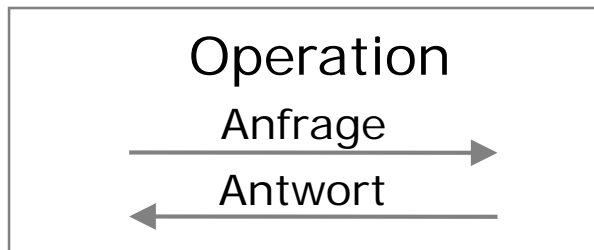
- WSDL = zu veröffentlichende Schnittstellenbeschreibung (Vertrag)
- Nutzer des Web Service kennt nur WSDL, nicht Programm-Code
- ⇒ Web-Service-Anbieter/Nutzer sollten WSDL (Vertrag) verstehen!
- mögliche Probleme bei generierten WSDLs:
  - Fehlermeldungen nicht korrekt beschrieben
  - optionale RPC-Parameter ungünstig beschrieben



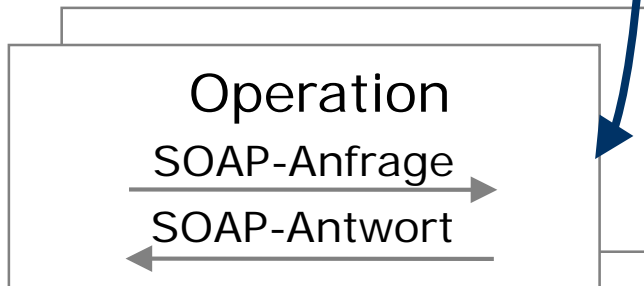
- beschreibt Netzwerkdienste als Kommunikationsendpunkte (Ports), die bestimmte Nachrichten über bestimmte Protokolle austauschen



## abstrakte Schnittstelle



## versch. Bindungen



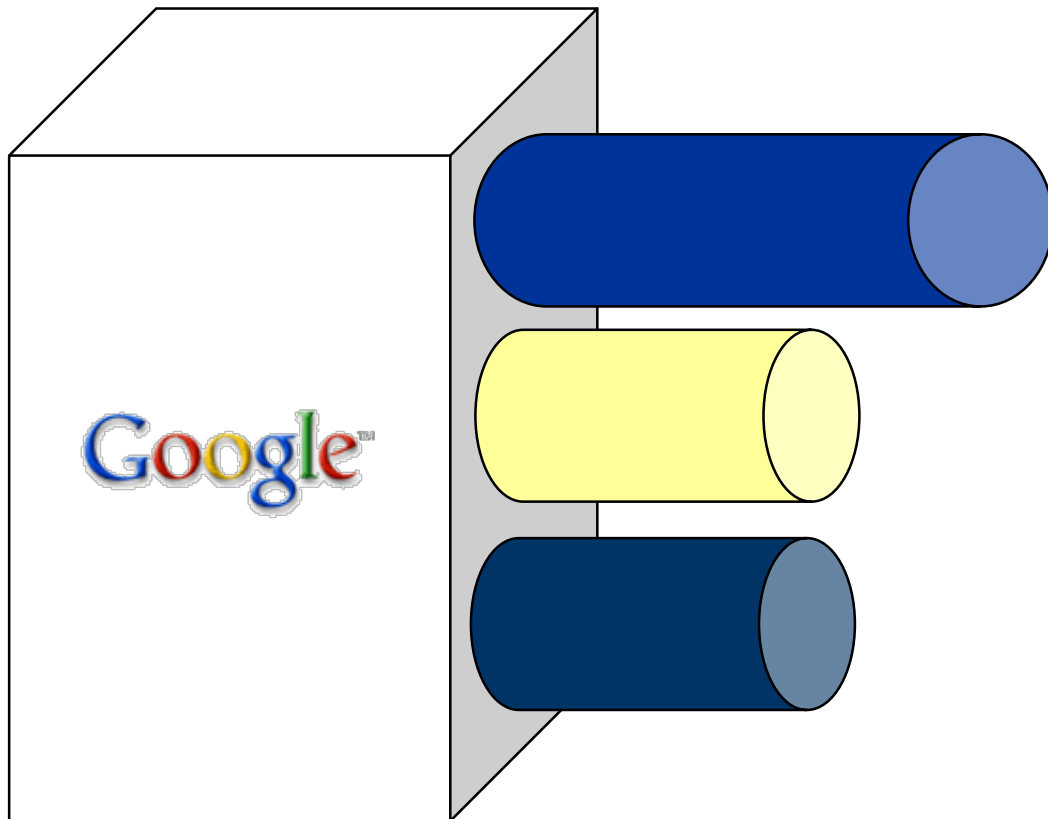
## abstrakte Schnittstelle

- Beschreibung der Schnittstelle unabhängig von
  - Nachrichtenformaten wie SOAP
  - Übertragungsprotokollen wie HTTP

## Bindung

- Realisierung einer abstrakten Schnittstelle mit bestimmtem Nachrichtenformat und Übertragungsprotokoll

- ein Dienst (**abstrakte Schnittstelle**):
  - Name der Operation: `doGoogleSearch`
  - Eingangsparameter: `key:string`, `q:string`, ...
  - Rückgabewert: `doGoogleSearchResponse`
    - Kind-Elemente von *doGoogleSearchResponse*: Rückgabewerte (komplexer Datentyp)
  
- eine Beschreibung (**WSDL**), aber 4 Zugriffsmöglichkeiten (**Bindungen**):
  1. SOAP/HTTP-POST
  2. SOAP/HTTP-GET (REST)
  3. SOAP/SMTP (asynchron)
  4. HTML/HTTP-GET (Browser/REST)

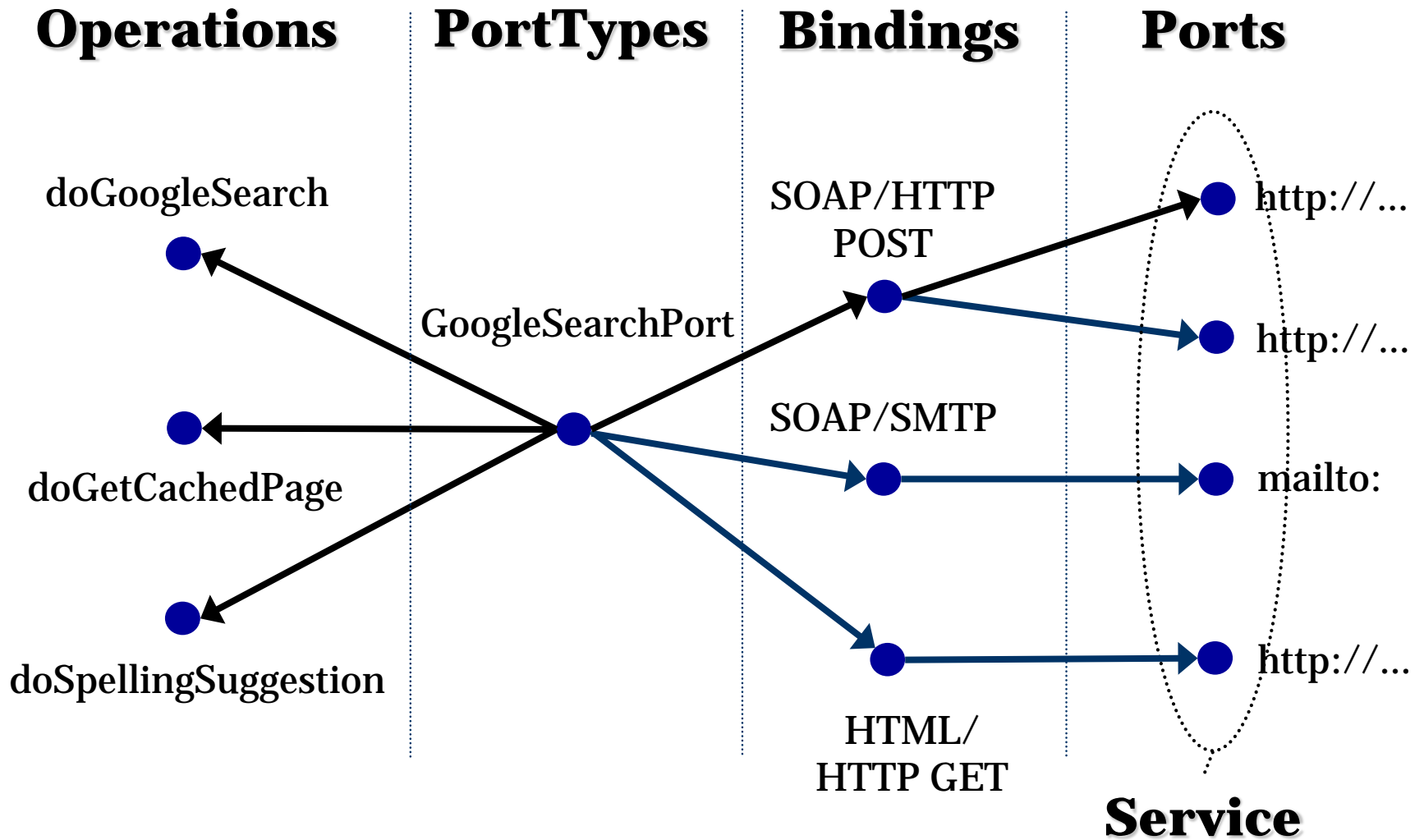


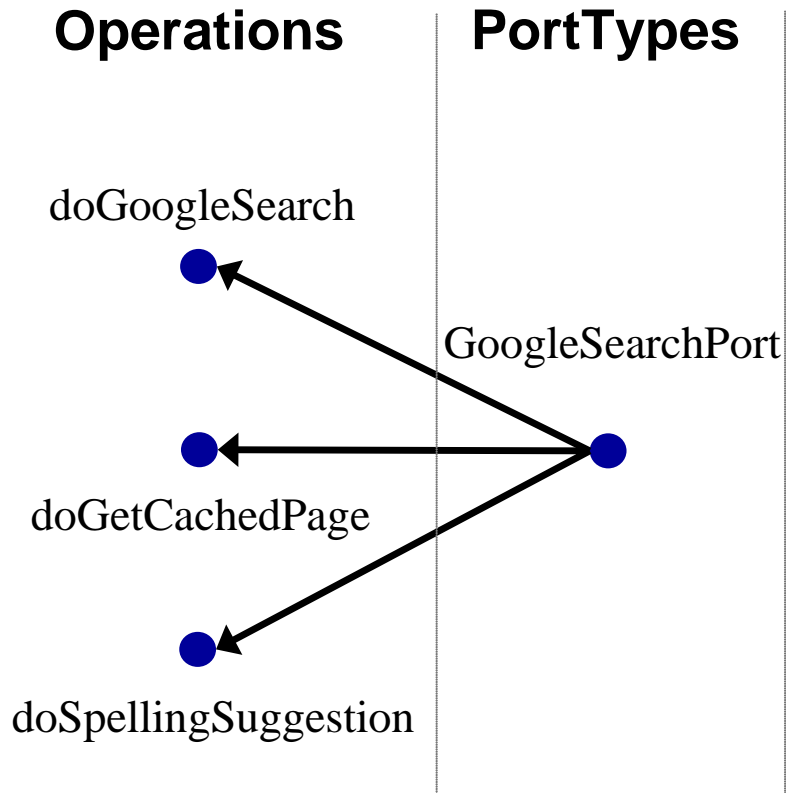
Dienst: Suche

Name der Operation:  
doGoogleSearch

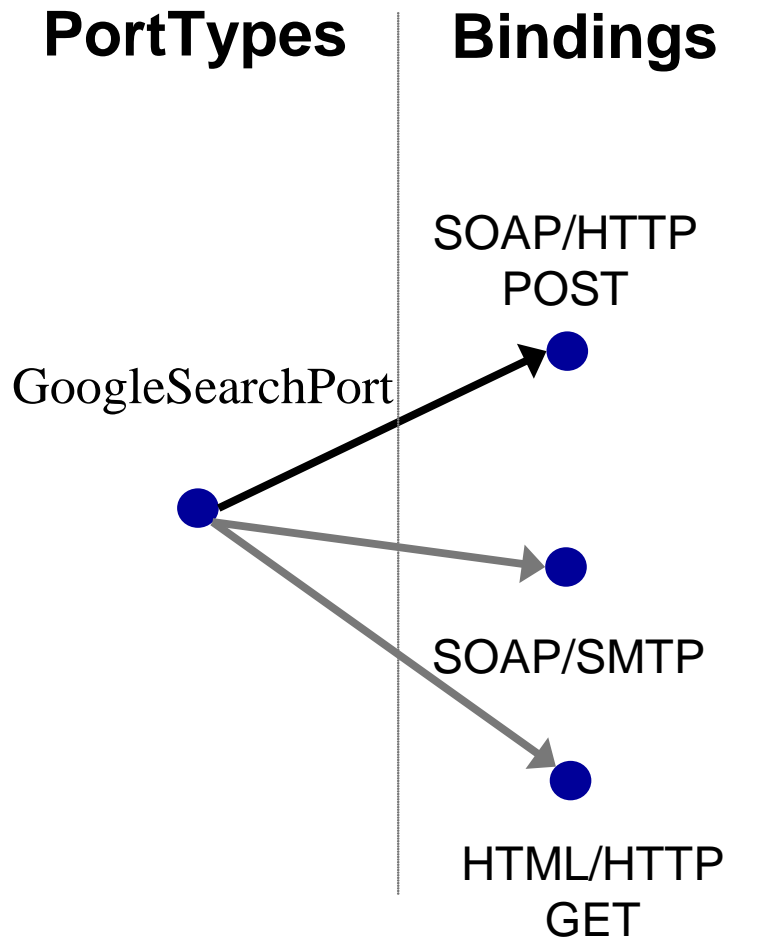
Rückgabe:  
doGoogleSearchResponse

- Was? → Typen, Messages, PortTypes (Interfaces)
  - Deklaration der verfügbaren Operationen
  - Struktur der ausgetauschten Nachrichten (Aufruf und Rückruf, Fehlermeldungen)
- Wie → Bindings
  - unterstützte Transportprotokolle
  - verwendete Nachrichtenformate
- Wo → Service
  - Wie heißt der Service?
  - Unter welchen URLs kann er gefunden werden?

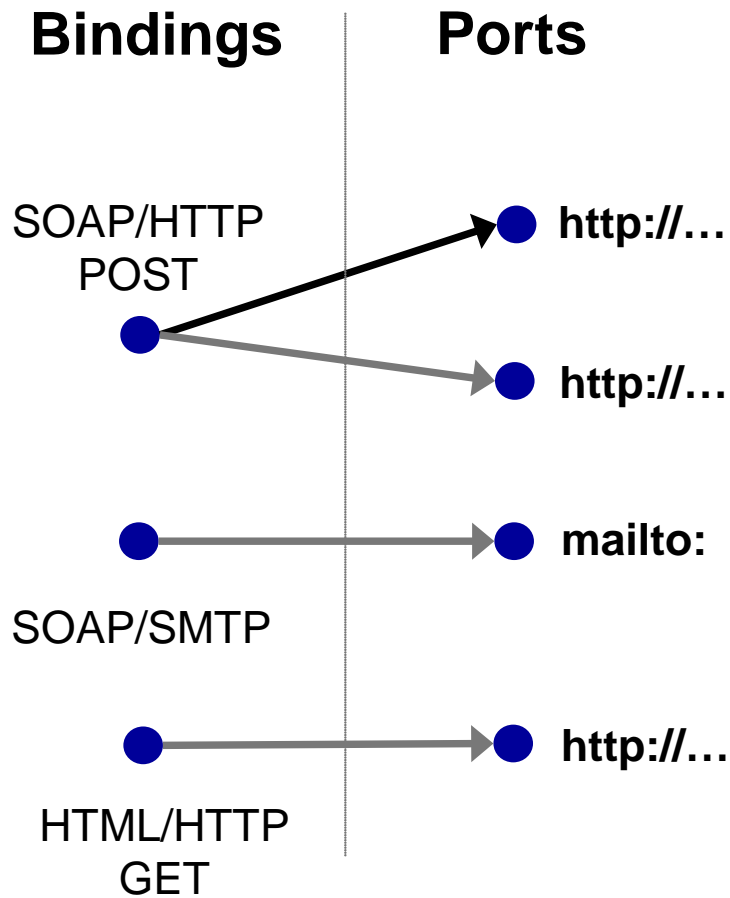




- **portType** (WSDL 1.1) = **interface** (WSDL 2.0)
- portType = Menge von abstrakten Operationen
- jede abstrakte Operation beschreibt Eingangs- und Ausgangsnachricht
- meist nur ein portType, aber in WSDL 1.1 auch mehrere möglich

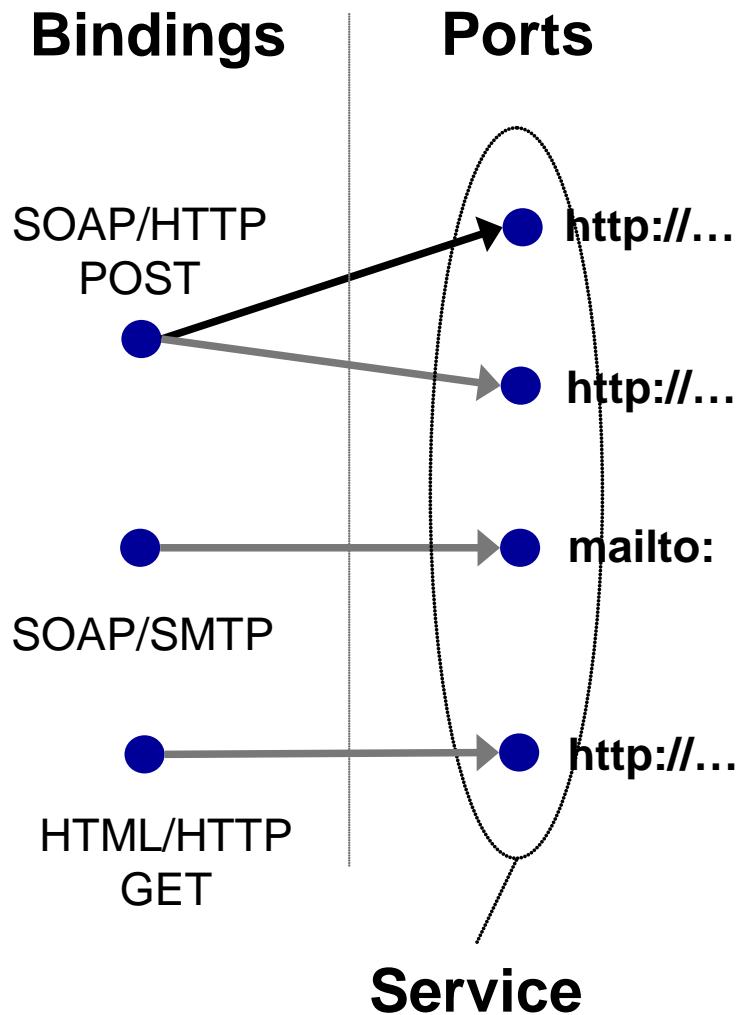


- in WSDL **binding** genannt
- für jede abstrakte Schnittstelle (**portType**) mindestens eine Bindung
- ein **portType** kann also mit **unterschiedlichen Bindungen** realisiert sein

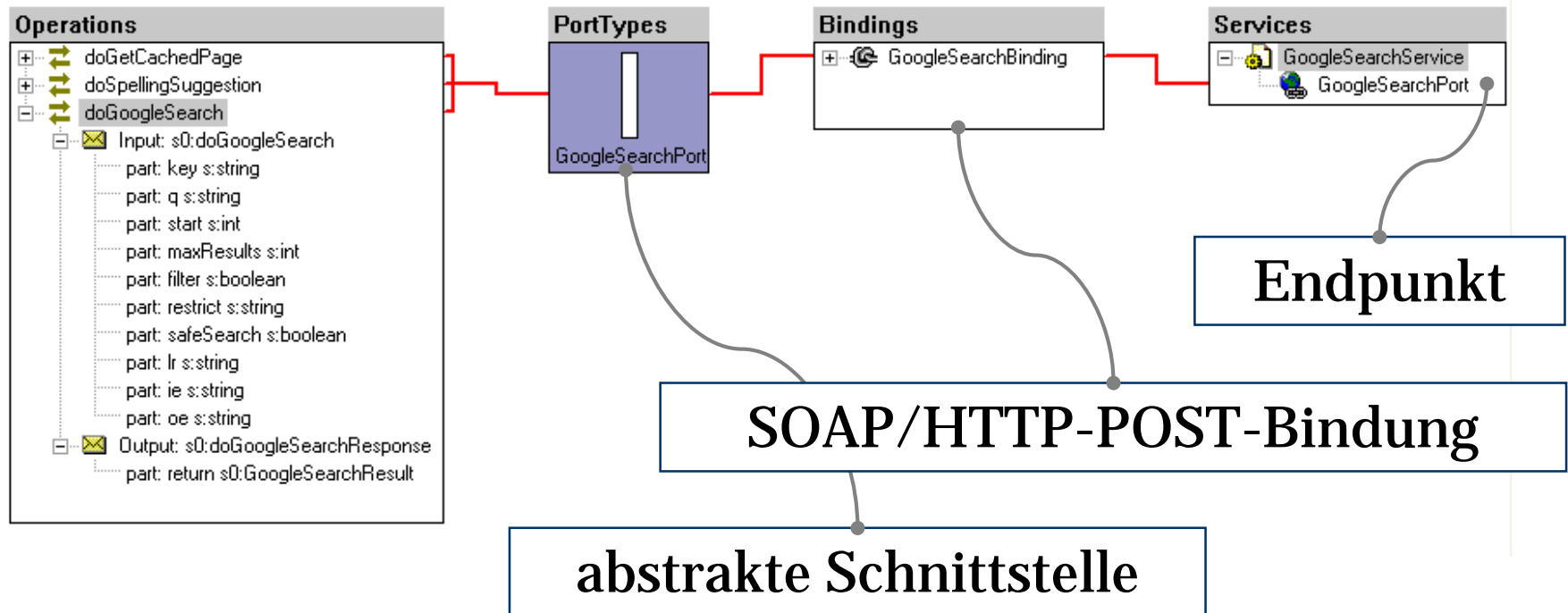


- **port** (WSDL 1.1) = **endpoint** (WSDL 2.0)
- **port** = Bindung + Web-Adresse
- für jede Bindung (**binding**) mindestens ein **port**
- ein **binding** kann also über unterschiedliche Web-Adressen zugänglich sein





- Menge von **ports** bilden zusammen einen **Service**
- **ports** können in verschiedene Services gruppiert werden
- **ports** eines Service = semantisch äquivalente Alternativen

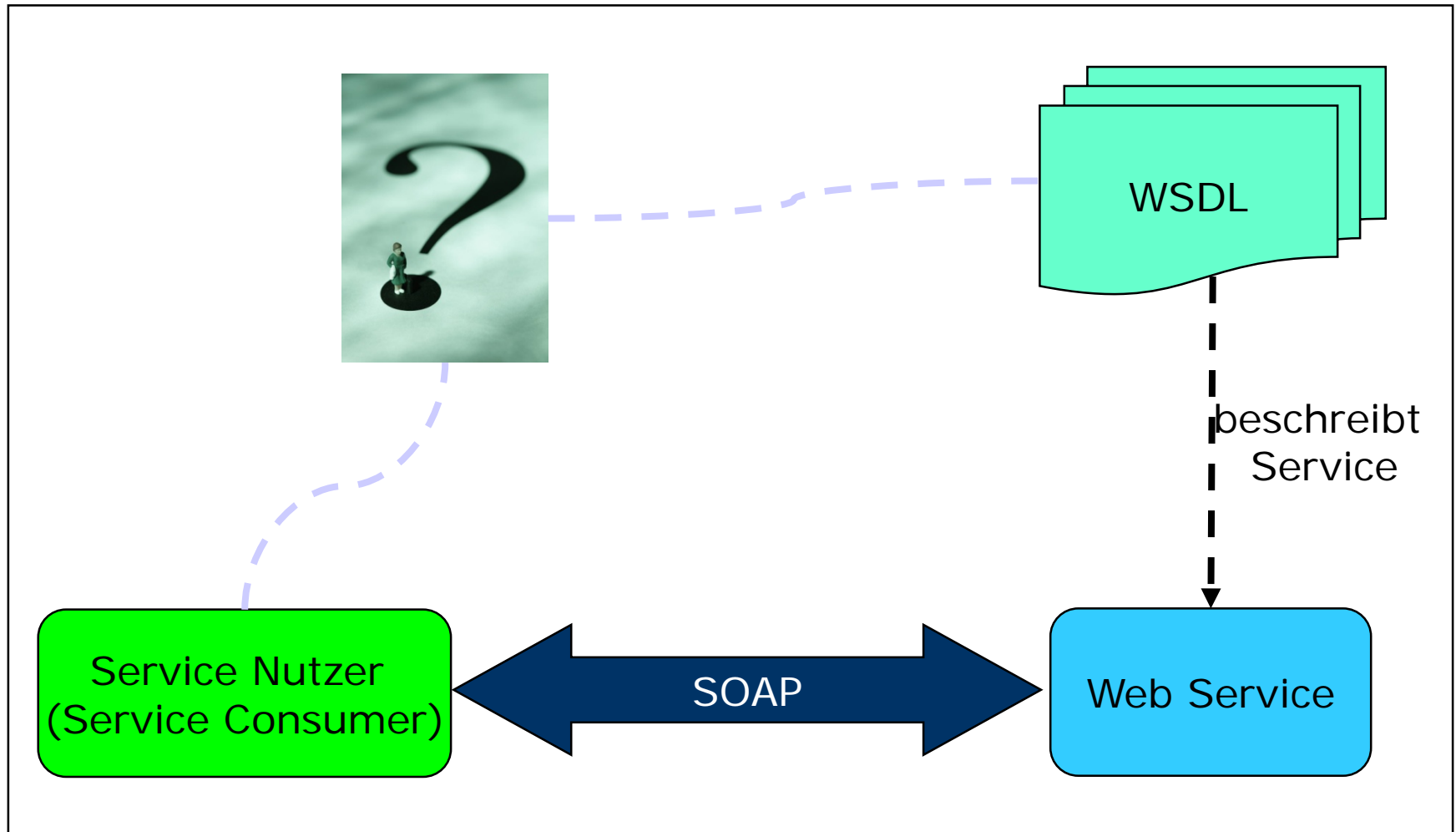


Element	Beschreibung
<b>Abstrakte Beschreibung</b>	
<code>&lt;types&gt;</code> ... <code>&lt;/types&gt;</code>	- Maschinen- und sprachunabhängige Typdefinitionen → definiert die verwendeten <b>Datentypen</b>
<code>&lt;message&gt;</code> ... <code>&lt;/message&gt;</code>	- <b>Nachrichten</b> , die übertragen werden sollen - Funktionsparameter (Trennung zwischen Ein- und Ausgabeparameter) oder Dokumentbeschreibungen
<code>&lt;portType&gt;</code> ... <code>&lt;/portType&gt;</code>	- Nachrichtendefinitionen im Messages-Abschnitt - definiert <b>Operationen</b> , die beim Web Service ausgeführt werden

Element	Beschreibung
<b>Konkrete Beschreibung</b>	
<code>&lt;binding&gt;...&lt;/binding&gt;</code>	<ul style="list-style-type: none"> <li>- <b>Kommunikationsprotokoll</b>, das beim Web Service benutzt wird</li> <li>- Gibt die Bindung(en) der einzelnen Operationen im portType-Abschnitt an</li> </ul>
<code>&lt;service&gt;...&lt;/service&gt;</code>	<ul style="list-style-type: none"> <li>- gibt die Anschlussadresse(n) der einzelnen Bindungen an (Sammlung von einem oder mehreren Ports)</li> </ul>

# Eigenschaften von WSDL

- baut auf **XML-Schema** auf
  - ⇒ Syntax einer Schnittstelle kann bis ins kleinste Detail festgelegt werden
- beschreibt **Schnittstelle(n)** eines Web Services und **wo** dieser **abgerufen** werden kann
- **grundlegende Interaktionsmuster** (wie Anfrage-Antwort)
- keine Möglichkeit semantische Eigenschaften zu beschreiben



# Vorteile von WSDL

- **Ziel: Interoperabilität**

→ Interoperabilität zwischen unterschiedlichen Implementierungsplattformen

## *Vorteile*

- + Plattformunabhängig
- + Syntax der Schnittstelle kann genau festgelegt werden
- + Unterschiedliche Realisierungen einer abstrakter Schnittstelle möglich (z.B. SOAP über HTTP und SMTP)

## *schafft neue Probleme*

- nicht alle Entwicklungen werden akzeptiert (vgl. UDDI, REST vs. SOAP)
- nicht alle geforderte Funktionalitäten sind verfügbar (Sicherheit, Transaktionen, Schnittstellenversionierung, etc.)
- eine weitere Schnittstellentechnologie, die gewartet werden muss

## *Nachteile*

- verschiedene Protokoll-Bindungen (wie HTTP vs. SMTP) können unterschiedliche Semantik haben
- keine komplexen Interaktionsmuster
- keine qualitativen Aspekte (quality of service)
- keine Sicherheitsaspekte
- unzureichend, um automatisch die Kompatibilität (Interoperabilität) zweier Web Services feststellen zu können → Semantic Web Services





## RPC vs. Messaging

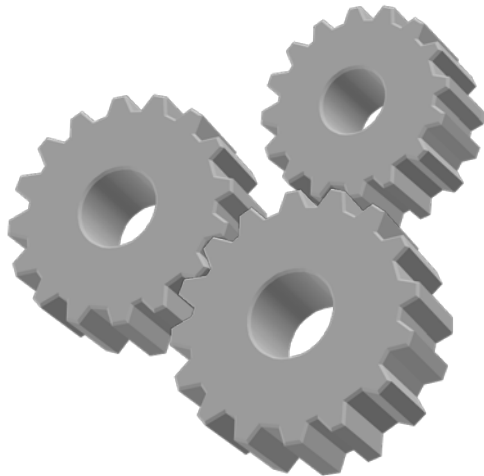
## über HTTP

- heute üblich
- Request-Response-Verhalten von HTTP unterstützt **RPCs**
- **verbindungsorientiert**
- Übertragung: Sender und Empfänger müssen präsent sein.
- typischerweise **synchron**

## über SMTP

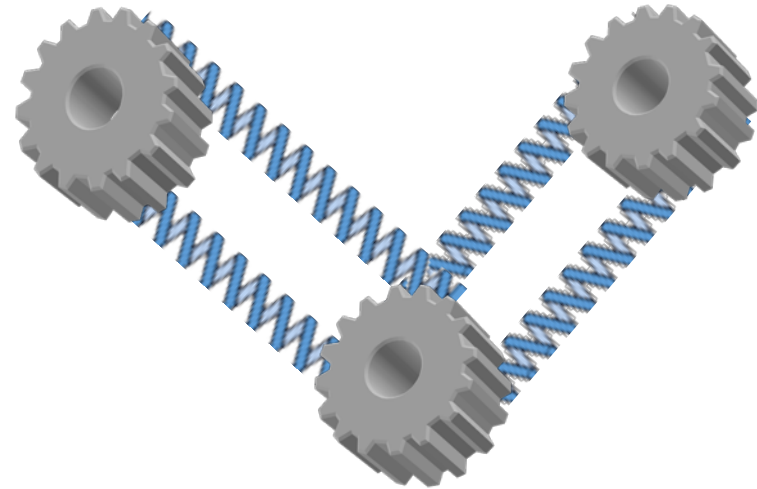
- heute eher selten
- realisiert **Messaging**
- **persistente** Kommunikation
- Übertragung: Weder Sender noch Empfänger muss präsent sein.
- typischerweise **asynchron**
- erlaubt Lastverteilung und Priorisierung

⇒ weitreichende Designentscheidung!



## enge Kopplung

- **verbindungsorientierte, synchrone** Kommunikation
- z.B. SOAP/HTTP



## lose Kopplung

- **gepufferte, asynchrone** Kommunikation
- z.B. SOAP/SMTP
- robuster, aber auch komplexer zu entwerfen

- **Nachrichten konform:**
  - zu einer vordefinierten Beschreibung des Funktionsaufrufes
  - zu der zu erwartenden Rückantwort
- **Arten der Kommunikation:**
  - Anfrage (Request)
  - Antwort (Response)
  - Fehlerfall (Fault)

## **asynchroner Nachrichtenaustausch**

- auch nach Timeout Antwort noch möglich
- Was tun mit der Antwort?
- häufig muss Absender informiert werden, dass Antwort nicht verarbeitet werden konnte
- Was tun, wenn diese Warnung nicht rechtzeitig beim Absender ankommt?
- Absender hat vielleicht im falschen Glauben, dass seine Antwort verarbeitet wurde, weitergearbeitet.
- Dieser Vorgang muss dann evtl. rückgängig gemacht werden.

- Anwendungen interagieren durch Austausch von Nachrichten miteinander:
- Kommunikation in Anwendung sichtbar: senden und empfangen
- verschiedene Formen des Messaging:
  1. Kommunikationsstruktur
  2. Interaktionsmuster
  3. flüchtig vs. persistent
  4. synchron vs. asynchron

- Anzahl und Organisation der Kommunikationspartner
- wichtigste Kommunikationsstrukturen:
  - One-to-One-Kommunikation
  - One-to-Many-Kommunikation

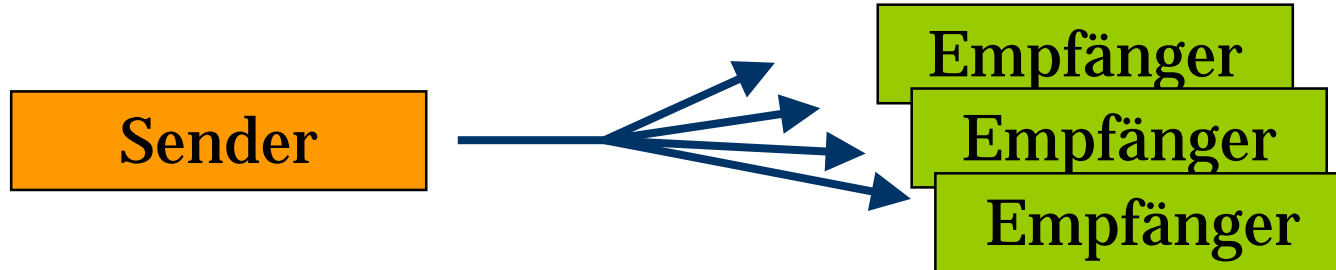
# 1. Kommunikationsstruktur: One-to-One-Kommunikation



- Sender sendet Nachricht an bestimmten Empfänger.
- Beispiel: Kunde sendet Bestellung per E-Mail an Firma



# 1. Kommunikationsstruktur: One-to-Many-Kommunikation



- Sender sendet identische Kopie gleichzeitig an mehrere Empfänger
  - Sender (publisher) veröffentlicht Nachricht zu bestimmten Thema (topic), zu dem sich die Empfänger (subscriber) angemeldet haben.
- ⇒ auch **publish-subscribe** oder **topic-based messaging** genannt
- Beispiel: Mailing-Liste

## 2. Interaktionsmuster

Client  Server

**Einweg** (one way,  
fire and forget)

Client  Server

**Anfrage-Antwort**  
(request-response)

---

Client  Server

**Benachrichtigung**  
(notification)

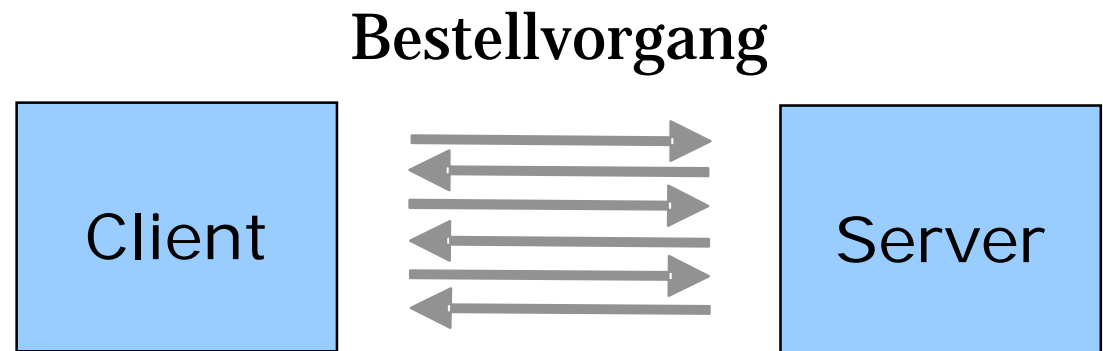
Client  Server

**Benachrichtigung-  
Antwort** (notification-  
response)

Beachte: „Antwort“ bezieht sich auf jeweilige Anwendung,  
nicht auf das Netzwerk.

## 2. Interaktionsmuster: Komplexe Interaktionsmuster

- Bestellanfrage mit spätestem Liefertermin
- ← Angebot mit zugesichertem Liefertermin
- Bestellung
- ← Bestätigung des Eingangs der Bestellung
- Bestätigung der Lieferung
- ← Rechnung

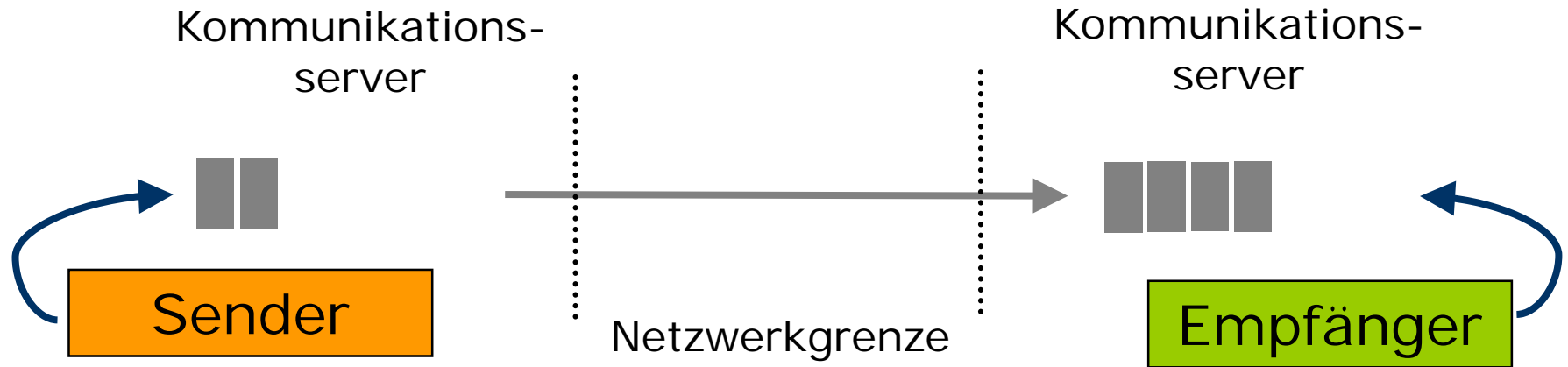


### 3. Flüchtig vs. Persistent



#### flüchtige Kommunikation

- Sender und Empfänger kommunizieren direkt ohne Puffer miteinander.
- Sender und Empfänger müssen während der gesamten Übertragung präsent sein
- engl. transient
- Beispiele: HTTP, UDP



## **persistente Kommunikation**

- Nachricht solange gespeichert, bis sie tatsächlich zugestellt wurde.
- Weder Sender noch Empfänger müssen während Übertragung präsent sein.
- Beispiel: E-Mail

## 4. Synchron vs. Asynchron

- **Asynchron**
  - Senden und Empfangen zeitlich versetzt
  - ohne Blockieren des Prozesses (ohne Warten auf die Antwort)
- **Synchron**
  - Senden/Empfangen synchronisieren → warten (blockieren) bis die Kommunikation abgeschlossen ist.
- bei **persistenter Kommunikation** → Messaging normalerweise asynchron
- bei **flüchtiger Kommunikation** → Messaging kann sowohl synchron als auch asynchron sein

## RPC $\Rightarrow$ eng gekoppelte, starre Systeme

- + einfach, abstrahiert von Kommunikation
- nur Eins-zu-Eins-Kommunikation
- Client und Server müssen präsent sein
- skaliert weniger gut

## Messaging $\Rightarrow$ lose gekoppelte, robuste Systeme

- abstrahiert nicht von Kommunikation
- + erlaubt auch One-to-Many-Kommunikation
- + Weder Sender noch Empfänger müssen präsent sein
- + erlaubt Priorisierung und Lastverteilung
- + skaliert sehr gut

## heutige Vorlesung

- ☑ Was sind Web Services?
- ☑ Basistechnologien (SOAP, WSDL, UDDI)
- ☑ RPC vs. Messaging