Zustandsändernde / -invariante Aufträge

- Verändern Aufträge den Zustand des Servers?
  - typische zustandsinvariante Aufträge: Anfrage bei Auskunftsdienst (z.B. Namensdienst oder Zeitservice)
  - typische zustandsändernde Aufträge: Schreiben bei Datei-Server
- Idempotente Aufträge
  - Wiederholung eines Auftrags führt zum gleichen Effekt
  - Beispiel: „Schreibe in Position 317 von Datei XYZ den Wert W“ (ist aber nicht zustandsinvariant!)
  - Gegenbeispiel: „Schreibe ans Ende der Datei XYZ den Wert W“
  - Gegenbeispiel: „Wie spät ist es?“ (ist aber zustandsinvariant!)
- Bei Idempotenz oder Zustandsinvarianz kann bei fehlgeschlagenem Auftrag (timeout beim Client) dieser problemlos erneut abgesetzt werden (→ einfache Fehlertoleranz)

Zustandslose („stateless“) / zustandsbehaftete („statefull“) Server

- Hält der Server Zustandsinformation über Aufträge hinweg?
  - z.B. (Protokoll)zustand des Clients
  - z.B. Information über frühere damit zusammenhängende Aufträge
- Beispiel: Datei-Server

```
open(“XYZ”);
read;
read;
close;
```

In klassischen Systemen hält sich das Betriebssystem Zustandsinformation, z.B. über die Position des Dateizeigers geöffneter Dateien

- Bei zustandslosen Servern entfällt open/close; jeder Auftrag muss vollständig beschrieben sein (Position des Dateizeigers etc.)
  - zustandsbehaftete Server daher i.Allg. effizienter
  - Dateisperren bei echten zustandslosen Servern nicht einfach möglich
- Zustandsbehaftete Server können wiederholte Aufträge erkennen (z.B. Speichern von Sequenznummern) → Idempotenz
- Crash eines Servers: Weniger Probleme im zustandslosen Fall

Sind Web-Server zustandslos?

- Beim HTTP-Zugriffsprotokoll wird über den Auftrag hinweg keine Zustandsinformation gehalten
  - jeder link, den man anklickt, löst eine neue „Transaktion“ aus
- Stellt z.B. ein Problem E-Commerce-Anwendungen dar
  - oft gewünscht: Transaktionen über mehrere Klicks hinweg und
  - Wiedererkennen von Kunden (beim nächsten Klick oder Tage später)
  - erforderlich z.B. für Realisierung von „Warenkörben“ von Kunden

Wiedererkennung von Kunden?

- „URL rewriting“ und dynamische Web-Seiten
  - der Einstiegsseite eine eindeutige Identität anheften, wenn der Kunde diese erstmalig aufruft
  - diese Identität jedem link der Seite anheften und mit zurückübertragen
- „Cookie“ als Context-Handle
  - kleine Textdatei, die ein Server einem Browser (= Client) schickt und die im Browser gespeichert wird
  - der Server kann das Cookie später wieder lesen und damit den Kunden wiedererkennen
- Evtl. auch Wiedererkennung über IP-Adresse
  - aber oft Probleme bei dynamischen IP-Adressen, Proxies etc.

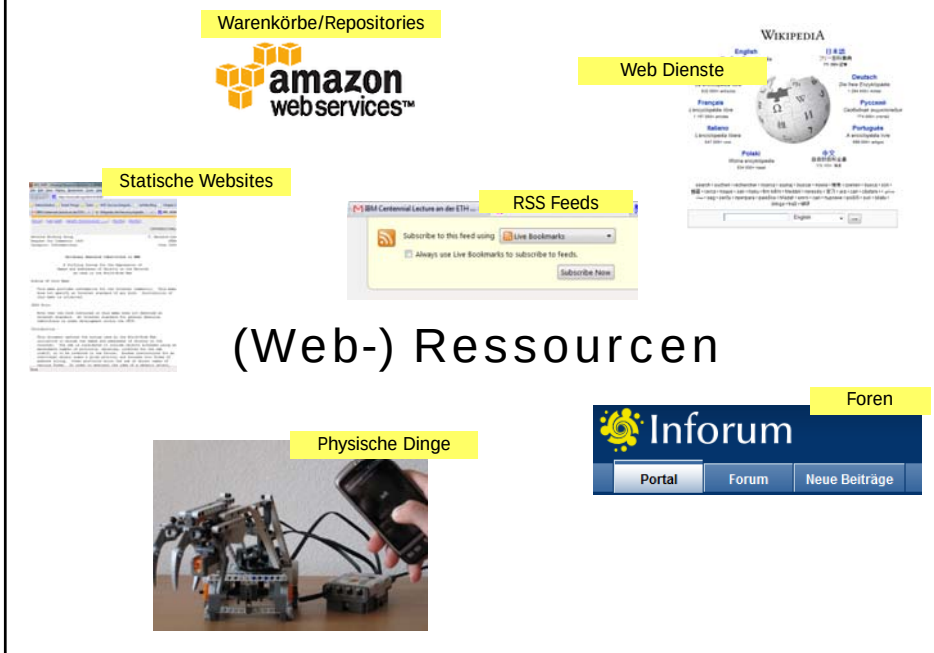
Ressourcen-orientierte Architektur (ROA)

- Funktionalität wird nicht durch Services („SOA“), sondern durch (Web-) Ressourcen angeboten
- Ressource? Bezugsobjekt eines Uniform Resource Identifiers
  - RFC 1630 „URL“ (1994) — Implizit: „Etwas, das adressiert werden kann“
  - RFC 2396 „URI“ (1998)

> *A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources...*

  - RFC 3986 „URI“ (2005)

> *...Likewise, abstract concepts can be resources, such as the operators and operands of a mathematical equation, the types of a relationship...*



Repräsentation – Beispiel „Buch“

- Das Buch als abstraktes Konzept
  - es gibt verschiedene Ausgaben, Exemplare etc.
  - identifiziert per ISBN: *ISBN-13: 978-3780200075*
- Was wir kaufen oder ausleihen ist eine Repräsentation des Buches
  - z.B. Hardcover, PDF, E-Book,...
  - auch ein Bild des Covers kann eine Repräsentation sein
  - oder ein maschinenlesbares XML-Dokument für das Bibliothekssystem

REST

- REST (als idealisierte Architektur des Web) steht für
  - Representational: Nicht Ressourcen, sondern deren Repräsentationen werden übertragen
  - State Transfer: Die Übertragung löst Zustandsübergänge aus und verändert damit die Ressourcen
- Motivation und Entwicklung
  - Erfolg des WWW in technischer Hinsicht (z.B. bzgl. Skalierbarkeit) beruht auf Eigenschaften der zugrundeliegenden Protokolle und Mechanismen
  - Idealisierung dieser Architektur durch Abstraktion von HTTP
  - Mit REST wird versucht, die Möglichkeiten, die das Web (bzw. HTTP) bietet, optimal auszunutzen

Eigenschaften von REST

- Zustandslosigkeit
  - entschärft Crash-Problematik und Orphans
  - erlaubt Caching und bessere Skalierbarkeit
- Einheitliche und a-priori bekannte Schnittstelle für alle Ressourcen
  - einheitliche Aufrufe, z.B. GET, POST bei HTTP
  - Adressierung direkt durch URIs
  - selbstbeschreibende Nachrichten: alle benötigten Metadaten sind enthalten, z.B. im HTTP-Header
  - bekannte Repräsentationen, z.B. MIME-Typen
- Bevorzugte Repräsentation wählbar
  - HTML, XML, JSON,...

auch andere Protokolle möglich!

Entwicklung von REST-Komponenten mit Java-IDE

- JAX-RS: Java API for RESTful Web Services (Beispiel: Eclipse)

ShoppingCartResource.java

```
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;

@Path("/cart/{cartID}")
public class ShoppingCartResource {

    @GET @Produces("text/html")
    public Response getCartContent(
            @PathParam("cartID") String cartID) {

        return Response.ok("Books in your cart: ...").build();
    }

    @POST @Consumes("application/json") @Produces("text/html")
    public Response addBookToCart(
            @PathParam("cartID") String cartID,
            @QueryParam("bookID") String bookID) {

        return Response.created(c.bookURIInCart).build();
    }
}
```

Erweiterung von einfachen Java-Objekten zu Ressourcen per Java Annotations

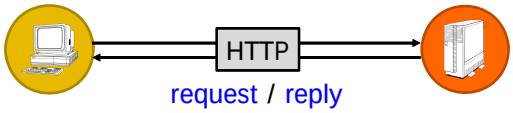
@Path: Pfad zur Ressource

Definition der verwendeten Media Types (@Consumes und @Produces)

Extraktion von Parametern aus dem Request mittels:
@PathParam: z.B. *{cartID}*
@QueryParam: z.B *?bookID=123*

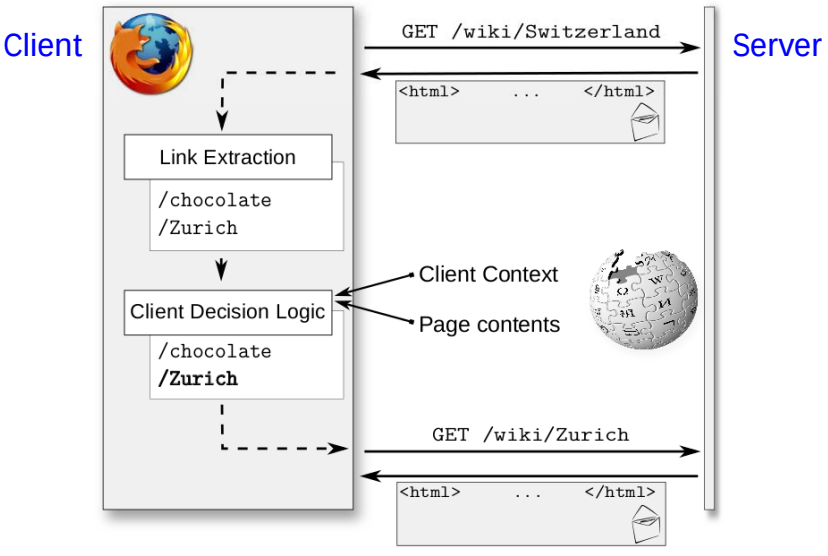
REST-Anwendungsmodell

- „Hypermedia as the Engine of Application State“
  - Client kennt ausschliesslich die Basis-URI des Dienstes
  - Server leitet durch die Anwendungszustände durch Bekanntgabe von Wahlmöglichkeiten (hyperlinks)



  - Der Anwendungszustand kann beim Client oder beim Server liegen, oder auch über beide verteilt sein
  - HTTP-Kommunikationsprotokoll selbst bleibt aber zustandslos

Beispiel:



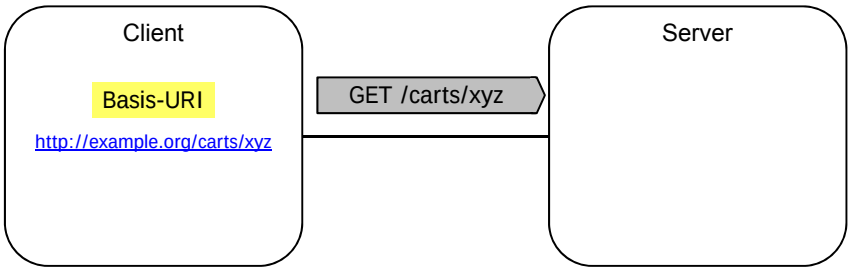
REST: Zustandsspeicherung

- Wenn der Server keinen Zustand hält, dann muss der Client alleine durch den Zustandsraum navigieren
  - Beispiel: man klickt sich durch eine Hypertext-Struktur (und merkt sich die früher besuchten Dokumente bzw. Zustände)
  - Aktueller Zustand = derzeitiges Hypertextdokument beim Client
  - Client und Server sind völlig entkoppelt
  - Dem Server ist es egal (und nicht bewusst), welches Hypertext-Dokument (d.h. Zustand) der Client vorher hatte
  - Der Client übergibt jeweils eine vollständige URI, die den Folgezustand bezeichnet
  - Bookmarks ergeben Sinn (sind vollständige URI losgelöst von jeglichem Kontext)
  - Back button im Browser ergibt Sinn: er führt zu einem früheren (im Client gecachten) Zustand

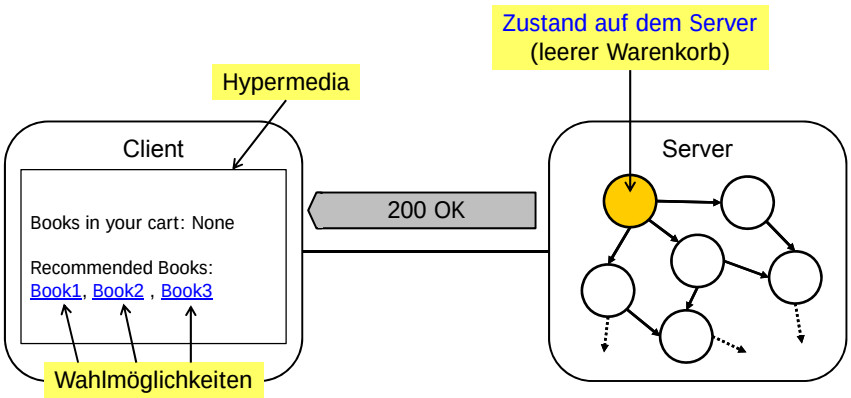
REST: Zustandsspeicherung

- Bei komplexeren Anwendungen residiert typischerweise der Anwendungszustand beim Server
  - z.B. Flugbuchung, wo Server mit externen Diensten kommuniziert
  - e-shop, wo der Warenkorb beim Server geführt wird
- Dann funktioniert eine Kopie einer URI (bookmark) später meistens nicht, weil dem Server der Kontext dazu fehlt
  - auch back button im Browser ist problematisch: führt zu einer früheren Zustandskopie, ohne dass der Server dies mitbekommt – Client meint fälschlicherweise, in einem gewissen Zustand zu sein, der tatsächliche Zustand wird aber auf dem Server gehalten

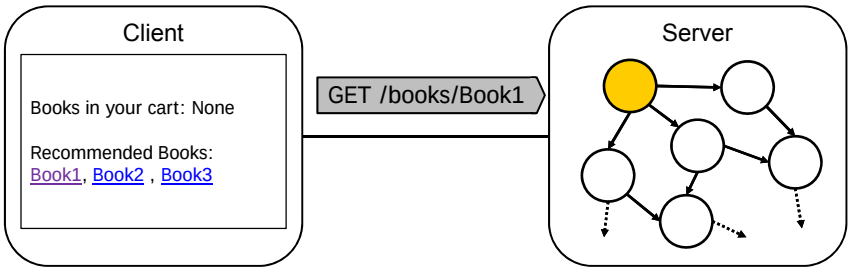
Zustandsspeicherung beim Server



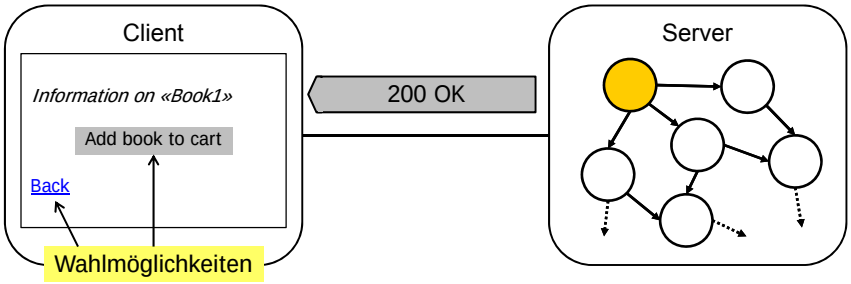
Zustandsspeicherung beim Server



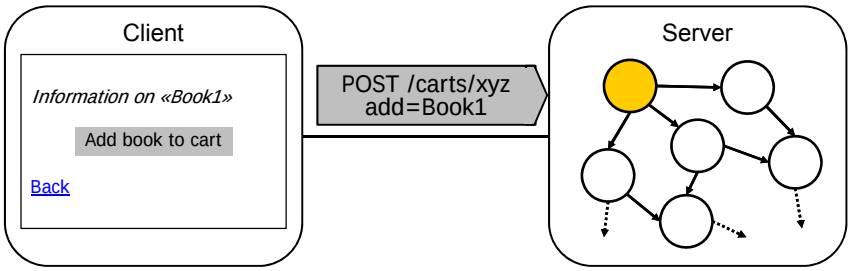
Zustandsspeicherung beim Server



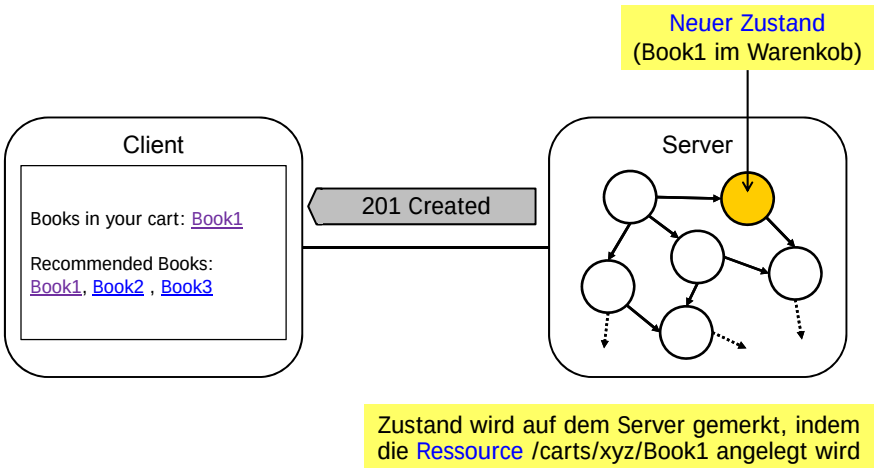
Zustandsspeicherung beim Server



Zustandsspeicherung beim Server

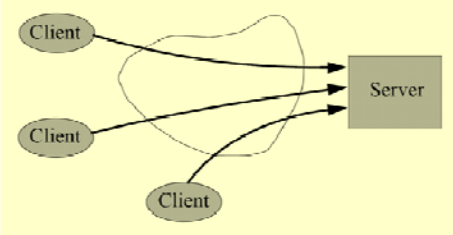


Zustandsspeicherung beim Server

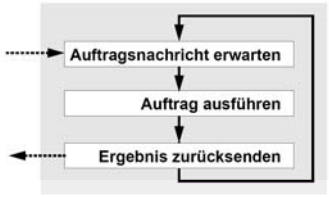


Gleichzeitige Server-Aufträge

- Problem: Oft viele „gleichzeitige“ Aufträge



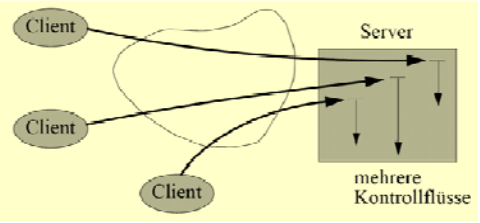
Iterativer Server („single threaded“):



- Iterative Server bearbeiten nur einen einzigen Auftrag pro Zeit
  - eintreffende Anfragen während der Auftragsbearbeitung: abweisen, in Warteschlange puffern oder schlichtweg ignorieren
  - einfach zu realisieren
  - bei trivialen Diensten mit kurzer Bearbeitungszeit sinnvoll

Konkurrente („nebenläufige“) Server

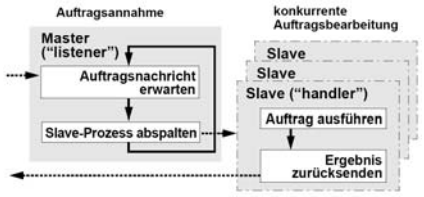
- (Quasi)gleichzeitige Bearbeitung mehrerer Aufträge
  - sinnvoll (d.h. effizienter für Clients) bei längeren Aufträgen



Verschiedene Realisierungen
- mehrere Prozessoren bzw. Multicore-Prozessoren
- Verbund verschiedener Server-Maschinen (Cluster)
- dynamische oder feste Anzahl vorgegründeter Prozesse

- Beachte: Auch bei Monoprozessor-Systemen ist Timesharing sinnvoll: Nutzung erzwungener Wartezeiten während einer Auftragsbearbeitung für Aufträge anderer Klienten; kürzere mittlere Antwortzeiten bei Jobmix aus langen und kurzen Aufträgen

Konkurrente Server mit dynamischen Handler-Prozessen



Für jeden Auftrag gründet der Master einen neuen Slave-Prozess und wartet dann auf einen neuen Auftrag

Alternative: „Process preallocation“: Feste Anzahl statischer Slave-Prozesse (evtl. effizienter, da Wegfall der Erzeugungskosten)

- Neu gegründeter Slave („handler“) übernimmt den Auftrag
- Client kommuniziert dann direkt mit dem Slave
- Slaves sind typischerweise Leichtgewichtsprozesse („threads“)
- Slaves terminieren i.Allg. nach Beendigung des Auftrags
- Die Anzahl gleichzeitiger Slaves sollte begrenzt werden

Master/Slave

Subject: Identification of equipment sold to LA County
Date: Tue, 18 Nov 2003 14:21:16 -0800
From: "Los Angeles County"

The County of Los Angeles actively promotes and is committed to ensure a work environment that is free from any discriminatory influence be it actual or perceived. As such, it is the County's expectation that our manufacturers, suppliers and contractors make a concentrated effort to ensure that any equipment, supplies or services that are provided to County departments do not possess or portray an image that may be construed as offensive or defamatory in nature.

One such recent example included the manufacturer's labeling of equipment where the words "Master/Slave" appeared to identify the primary and secondary sources. Based on the cultural diversity and sensitivity of Los Angeles County, this is not an acceptable identification label.

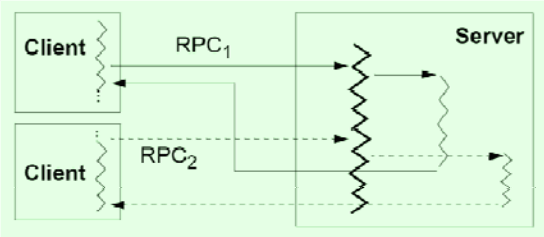
We would request that each manufacturer, supplier and contractor review, identify and remove/change any identification or labeling of equipment or components thereof that could be interpreted as discriminatory or offensive in nature before such equipment is sold or otherwise provided to any County department.

Thank you in advance for your cooperation and assistance.

Joe Sandoval, Division Manager
Purchasing and Contract Services
Internal Services Department
County of Los Angeles

Multithreading beim Client-Server-Konzept

- Server-Threads: quasiparallele Bearbeitung von Aufträgen
  - Server bleibt ständig empfangsbereit



- Client-Threads: Möglichkeit zum „asynchronen RPC“
  - Hauptkontrollfluss delegiert RPCs an nebenläufige Threads
  - keine Blockade durch Aufrufe im Hauptfluss
  - echte Parallelität von Client (Hauptkontrollfluss) und Server

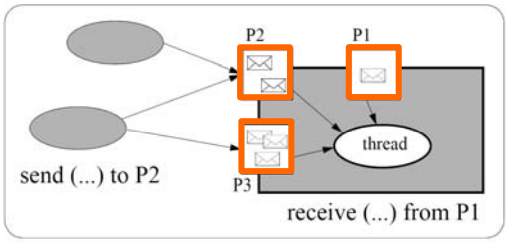
Mehr zu allgemeinen Kommunikationsprinzipien

Im Folgenden:

- Port-Konzept
- Kommunikationskanäle
- Ereigniskanäle
- Timeouts bei der Kommunikation
- Broadcast / Multicast

Das Port-Konzept

- Port = adressierbarer Kommunikationsendpunkt, der die interne Struktur eines Nachrichtenempfänger abkapselt
- Ein Prozess kann mehrere (evtl. typisierte) Ports haben

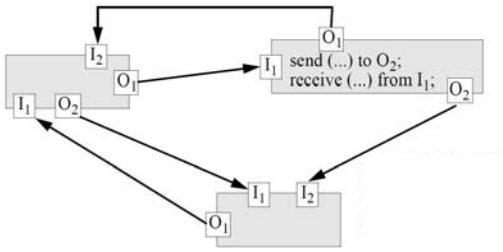


- Manchmal bilden Ports Stauräume („message queues“) für Nachrichten
- Manchmal können Ports dynamisch gegründet oder auch geschlossen / geöffnet werden

- Neben Eingangsports („In-Port“) sind manchmal auch Ausgangsports („Out-Port“) möglich

Kommunikationskanäle

- Kanäle, z.B. eingerichtet mit Ports als Endpunkten
  - dazu je einen In- und Out-Port miteinander verbinden

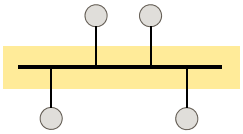


- Alternativ: Kanäle benennen und etwas *auf den Kanal senden* bzw. von ihm lesen
- Evtl. Broadcast-Kanäle: Nachricht geht an alle angeschlossenen Empfänger

- Flexibilität durch Umkonfiguration der Verbindungsstruktur
  - eigentlicher Adressat wird den Prozessen verborgen („virtualisiert“)

Ereigniskanäle für autonome Software-Komponenten

- Kooperierende autonome Software-Komponenten
  - nicht notwendigerweise geographisch weit verteilt
  - mit i.Allg. getrennten Lebenszyklen
  - anonym: kennen nicht die Identität der Anderen
  - Auslösen von „Ereignissen“ durch Sender
  - Reagieren auf Ereignisse beim Empfänger
- Ereigniskanal als „Softwarebus“
  - agiert als Zwischeninstanz und verknüpft die Komponenten
  - registriert Interessenten (vgl. LUS)
  - Dispatching eingehender Ereignisse
  - evtl. Puffern, Filtern, Umlenken von Ereignissen

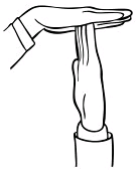


Ereigniskanäle (2)

- Probleme
  - Ereignisse können „jederzeit“ ausgelöst werden, werden von Empfängern aber i.Allg. nicht jederzeit entgegengenommen (→ Pufferung?)
  - falls Komponenten nicht lokal, sondern geographisch verteilt sind → „übliche“ Probleme nachrichtenbasierter Kommunikation: Verzögerungen, evtl. verlorene Ereignisse, falsche Reihenfolge,...
- Beispiele
  - Microsoft-Komponentenarchitektur (.NET etc.)
  - „Distributed Events“ bei JavaBeans und Jini (event generator bzw. remote event listener)

Zeitüberwachter Nachrichtenempfang

- Idee: Receive soll max. eine gewisse Zeit lang blockieren
  - z.B. über Rückgabewert abfragen, ob Kommunikation geklappt hat oder der Timeout zugeschlagen hat
  - Timeout-Wert adäquat setzen (oft schwierig)



- Im Timeout-Fall geeignete Recovery-Massnahmen treffen oder Exception auslösen
- Verwendung bei:
  - Echtzeitprogrammierung
  - Aufheben von Blockaden im Fehlerfall (z.B. bei abgestürztem Kommunikationspartner)
- Timeout evtl. auch beim blockierenden Senden sinnvoll

Zeitüberwachter Nachrichtenempfang (2)

- Sprachliche Einbindung z.B. so:

```
receive ... delay t
   {...}
 else
   {...}
end
```

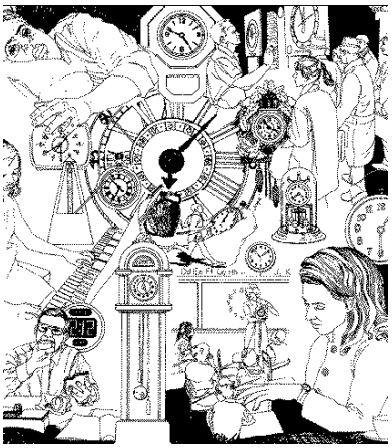
Wird nach mindestens t Zeiteinheiten ausgeführt, wenn bis dahin noch keine Nachricht empfangen

- Beachte: Es wird mindestens so lange auf Kommunikation gewartet – danach kann (wie immer!) noch beliebig viel Zeit bis zur Fortsetzung des Ablaufs verstreichen
- Was könnte „delay 0“ bedeuten? Ist das sinnvoll?

Logische Zeit

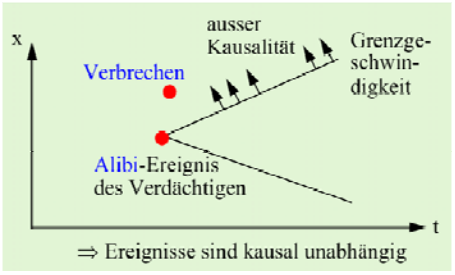
Zeit?

Ich halte ja eine Uhr für überflüssig. Sehen Sie, ich wohne ja ganz nah beim Rathaus. Und jeden Morgen, wenn ich ins Geschäft gehe, da schau ich auf die Rathausuhr hinauf, wie viel Uhr es ist, und da merke ich's mir gleich für den ganzen Tag und nütze meine Uhr nicht so ab. -- Karl Valentin



Kommt Zeit, kommt Rat

1. Volkszählung: Stichzeitpunkt in der Zukunft
   - liefert eine gleichzeitige „Beobachtung“ im Nachhinein
2. Kausalitätsbeziehung zwischen Ereignissen („Alibi-Prinzip“)
   - Ausschluss potentieller Kausalität
   - wurde Y später als X geboren, dann kann Y unmöglich Vater von X sein



Kommt Zeit, kommt Rat (2)

3. Fairer wechselseitiger Ausschluss
   - bedient wird derjenige, wer am längsten wartet
4. Viele weitere nützliche Anwendungen von „Zeit“ in unserer verteilten realen Welt
   - z.B. kausaltreue Beobachtung durch „Zeitstempel“ der Ereignisse

---

- Zeit ist vor allem natürlich auch dann wichtig, wenn es um Interaktionen mit der realen Welt geht
  - Prozesssteuerung, Realzeitsysteme, Cyber Physical Systems,...

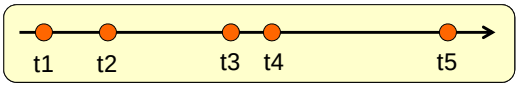
Eigenschaften der „Realzeit“

Struktureigenschaften eines „natürlichen“ Zeitpunktmodells:

- asymmetrisch (Zeit ist „gerichtet“)
- transitiv
- irreflexiv
- linear

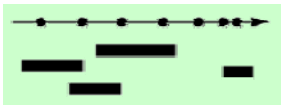
} lineare Ordnung („später als“)

- unbeschränkt („Zeit ist ewig“: Kein Anfang oder Ende)
- dicht (es gibt immer einen Zeitpunkt dazwischen)
- kontinuierlich
- metrisch
- vergeht „von selbst“
  → jeder Zeitpunkt wird schliesslich erreicht



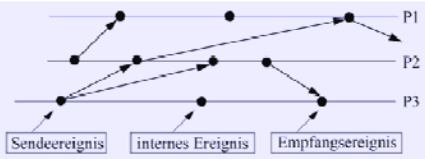
Eigenschaften der „Realzeit“ (2)

- Ist das Zeitpunktmodell adäquat? Oder sind Zeitintervalle besser?
  - wann tritt das Ereignis (?) „Sonne wird rot“ am Abend ein?



- Welche Eigenschaften benötigen wir wirklich?
  - Idee: „billigeren“ Ersatz für fehlende globale Realzeit konstruieren (z.B.: sind die reellen / rationalen / ganzen Zahlen gute Modelle?)
  - wann genügt welche Form „logischer“ statt „echter“ Zeit?
  - dazu vorher klären: was wollen wir mit „Zeit“ anfangen?

Raum-Zeitdiagramme und die Kausalrelation



Interessant dabei: von links nach rechts verlaufende „Kausalitätspfade“

Bezeichnung oft: „happened before“
- eingeführt von Leslie Lamport (1978)
- aber Vorsicht: damit ist nicht direkt eine „zeitliche“ Aussage getroffen!

Definiere eine Kausalrelation $\prec$ auf der Menge aller Ereignisse:

- Es sei $x \prec y$ genau dann, wenn:
  1) x und y auf dem gleichen Prozess stattfinden und x vor y kommt, oder
  2) x ist ein Sendeereignis und y korrespondierendes Empfangsereignis, oder
  3) $\exists$ z mit $x \prec z \ \wedge \ z \prec y$ (Transitivität)

links von

zur gleichen Nachricht gehörend

Logische Zeitstempel von Ereignissen

- Zweck: Ereignissen E eine Zeit geben
  - welche Zeit *zwischen* Ereignissen herrscht, ist irrelevant
  - gesucht ist also eine Abbildung $C: E \rightarrow \mathbb{N}$ („C“ steht für „Clock“)
  - $\mathbb{N}$ genügt hier, $\mathbb{Z}$ oder $\mathbb{R}$ ist nicht nötig, wie wir sehen werden
- C(e) nennt man den Zeitstempel von e
  - wenn C(e) < C(e’), dann nennt man e früher als e’

Ordnungshomomorphismus

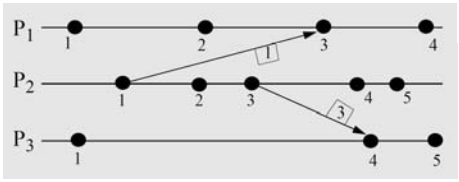
- Sinnvolle Forderung: Uhrenbedingung: $e \prec e' \Rightarrow C(e) < C(e')$
  - Interpretation („Zeit ist kausaltreu“):

    Kann ein Ereignis e ein anderes Ereignis e’ beeinflussen, dann muss e einen kleineren Zeitstempel als e’ haben

Communications of the ACM 21(7), 558-565, 1978: *Time, Clocks, and the Ordering of Events in a Distributed System*

Logische Uhren von Lamport

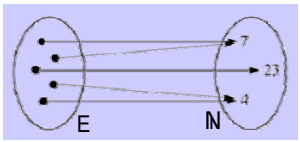
- $C: (E,\prec) \rightarrow (\mathbb{N},<)$
  Zeitstempel-Zuordnung
- $e \prec e' \Rightarrow C(e) < C(e')$
  Uhrenbedingung



- Protokoll zur Implementierung der Uhrenbedingung:
  - bei jedem Ereignis tickt die lokale Uhr (= „Zähler“)
  - Sendeereignis: Uhrwert mitsenden (→ Zeitstempel der Nachricht)
  - Empfangsereignis: Uhr = max(Uhr, Zeitstempel der Nachricht)
    (zuerst max, danach erst tickt die Uhr)
- *Behauptung:* Protokoll respektiert Uhrenbedingung
  - *Beweis:* Entlang von Kausalitätspfaden wächst logische Zeit monoton...

Lamport-Zeit: injektive Variante

- Abbildung ist nicht injektiv
  - wäre wichtig z.B. für: „Wer die kleinste Zeit hat, der gewinnt“



- Lösung: lexikographische Ordnung (C(e),i), wobei i die Prozessnummer bezeichnet, auf dem e stattfindet
  - Def.: $(a,b) < (a',b') \Leftrightarrow a<a' \vee a=a' \wedge b<b'$ (ist lineare Ordnung!)
- Alle Ereignisse haben nun verschiedene Zeitstempel
  - Abbildung ist injektiv
  - jede (nicht-leere) Menge von Ereignissen hat nun ein „frühestes“
- Abb. $(E,\prec) \rightarrow (\mathbb{N}\times\mathbb{N},<)$ respektiert die Uhrenbedingung

Umkehrung der Uhrenbedingung?

- Wieso gilt folgendes eigentlich nicht?

  $e \prec e' \Leftrightarrow C(e) < C(e')$

  - Was kann man überhaupt über die beiden Ereignisse e und e' sagen, wenn man die Zeitstempel C(e) und C(e') vergleicht?

- Kann man eine andere Art von Zeitstempeln finden, für die die Umkehrung der Uhrenbedingung gilt?
  - wofür wäre das nützlich?

Resümee (5a)

- Zustandsändernde / -invariante Aufträge
- Idempotente und wiederholbare Aufträge
- Stateless / statefull Server
- Zustandshaltung über Einzelaufträge hinweg bei Web-Servern
  - URL rewriting, cookies
- Ressourcen-orientierte Architekturen
  - Ressource
  - Repräsentation einer Ressource
  - REST: Representational State Transfer
- Zustandshaltung bei REST
  - Anwendungszustand als Hypermedia-Dokument
  - Speicherung dieses Zustands beim Server vs. zustandsloser Server

Resümee (5b)

- Kommunikationskonzepte
  - Ports; Kanäle; Ereigniskanäle als „Softwarebus“
  - Timeouts beim Empfangen von Nachrichten
- Konkurrente Server
  - z.B. dynamische / statische Handler-Prozesse („slaves“)
- Logische Zeit
  - Raum-Zeitdiagramme, Ereignisse, Kausalrelation
  - Zeitstempel von Ereignissen
  - Uhrenbedingung (als Ordnungshomomorphismus)
- Logische Uhren von Lamport
  - Definition
  - Realisierung
  - injektive Variante, eindeutige Zeitpunkte
  - Umkehrung der Uhrenbedingung?