

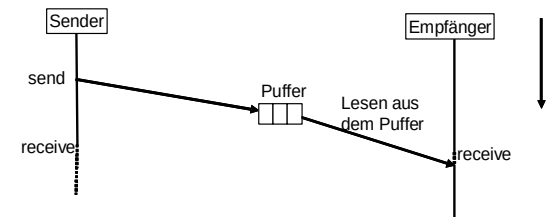
# MAS: Betriebssysteme

## Kommunikation von Prozessen und Threads

T. Pospíšek

# Gesamtüberblick

1. Einführung in Computersysteme
2. Entwicklung von Betriebssystemen
3. Architekturansätze
4. Interruptverarbeitung in Betriebssystemen
5. Prozesse und Threads
6. CPU-Scheduling
- 7. Synchronisation und Kommunikation**
8. Speicherverwaltung
9. Geräte- und Dateiverwaltung
10. Betriebssystemvirtualisierung



# Überblick

---

- 1. Einführung in die Grundbegriffe der Kommunikation**
2. Lokale Kommunikation
3. Verteilte Kommunikation

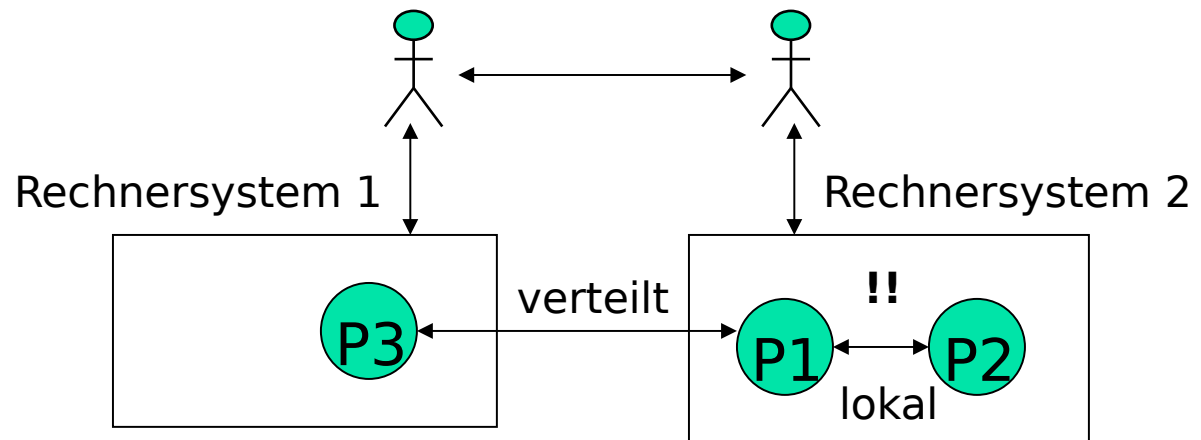
# Zielsetzung

---

- Grundlegende Begriffe der Prozess-Prozess-Kommunikation im lokalen und verteilten Umfeld kennenlernen
- Beispielmeechanismen zur Programmierung von Kommunikationsanwendungen kennenlernen

# Motivation

- **Kommunikation** ist der Austausch von Informationen zwischen:
  - Menschen
  - Mensch und Maschine (Rechnersystem)
  - Maschinen: Zwischen Prozessen/Threads innerhalb einer Maschine (lokal) oder Rechner-übergreifend (verteilt)



## Wichtige Aspekte

---

- **Speicherbasierte** versus **nachrichtenbasierte** Kommunikation
- **Verbindungsorientierte** versus **verbindungslose** Kommunikation
- **Synchrone** versus **asynchrone** Kommunikation
- Senderichtung im Kommunikationskanal: **Halbduplex-** versus **Vollduplex-**Betrieb, auch simplex möglich
- Varianten der **Empfängeradressierung**

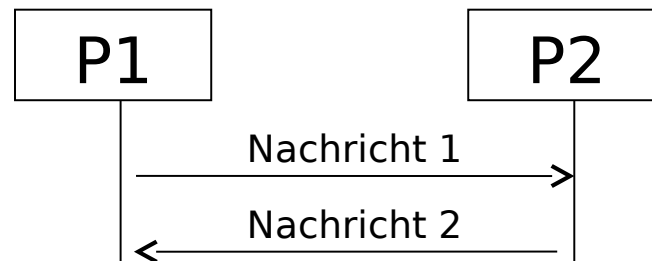
# Speicherbasierte und nachrichtenbasierte Kommunikation

## ■ **Speicherbasiert:**

- Daten werden in einem gemeinsamen Speicher ausgetauscht oder „geshared“
  - Gleicher Adressraum, Shared Memory
  - Datenbank
  - Datei

## ■ **Nachrichtenbasiert:**

- Zwischen Prozessen/Threads werden Nachrichten ausgetauscht → einheitliches Protokoll



# Verbindungsorientierte und verbindungslose Kommunikation

---

## ■ **Verbindungsorientiert:**

- Vor dem Datenaustausch wird eine logische Verbindung aufgebaut und danach wieder abgebaut
- Kommunikationspartner kennen Verbindungszustand

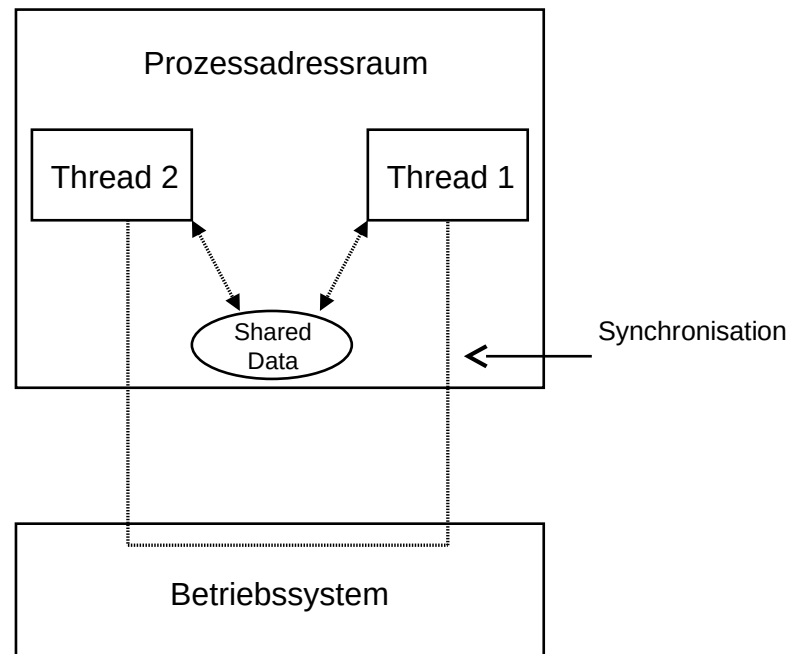
## ■ **Verbindungslos:**

- Daten werden ohne Verbindungsmanagement vom Sender zum Empfänger übertragen
- Senderadresse wird in der Nachricht mit übertragen



# Speicherbasierte Kommunikation über einen Adressraum

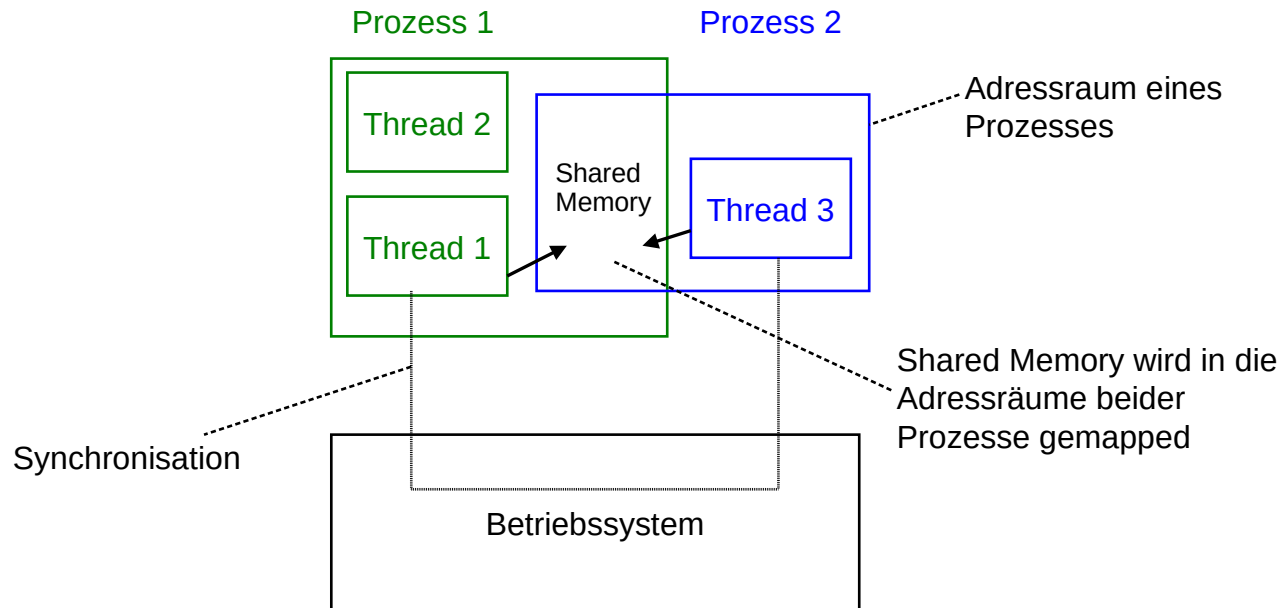
- Zwei Threads kommunizieren in einem Adressraum miteinander



# Speicherbasierte Kommunikation

## Variante Shared Memory

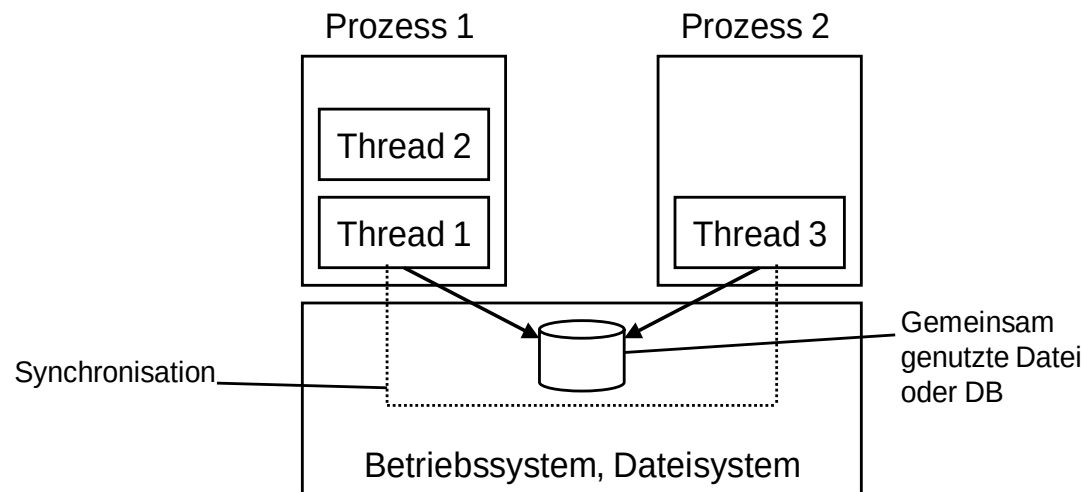
- Zwei Threads aus getrennten Adressräumen kommunizieren über Shared Memory



# Speicherbasierte Kommunikation

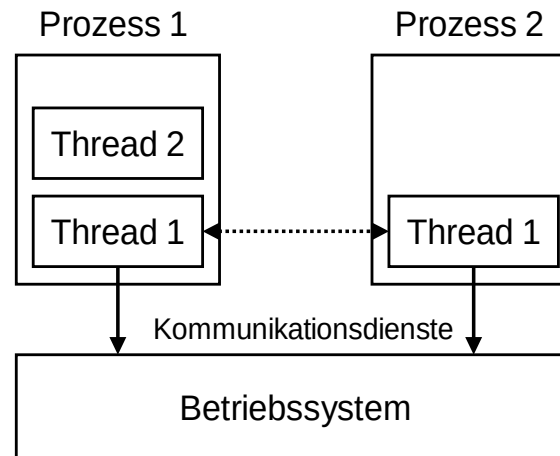
## Variante Gemeinsamer Dateizugriff

- Zwei Threads aus getrennten Adressräumen kommunizieren über Dateiaustausch



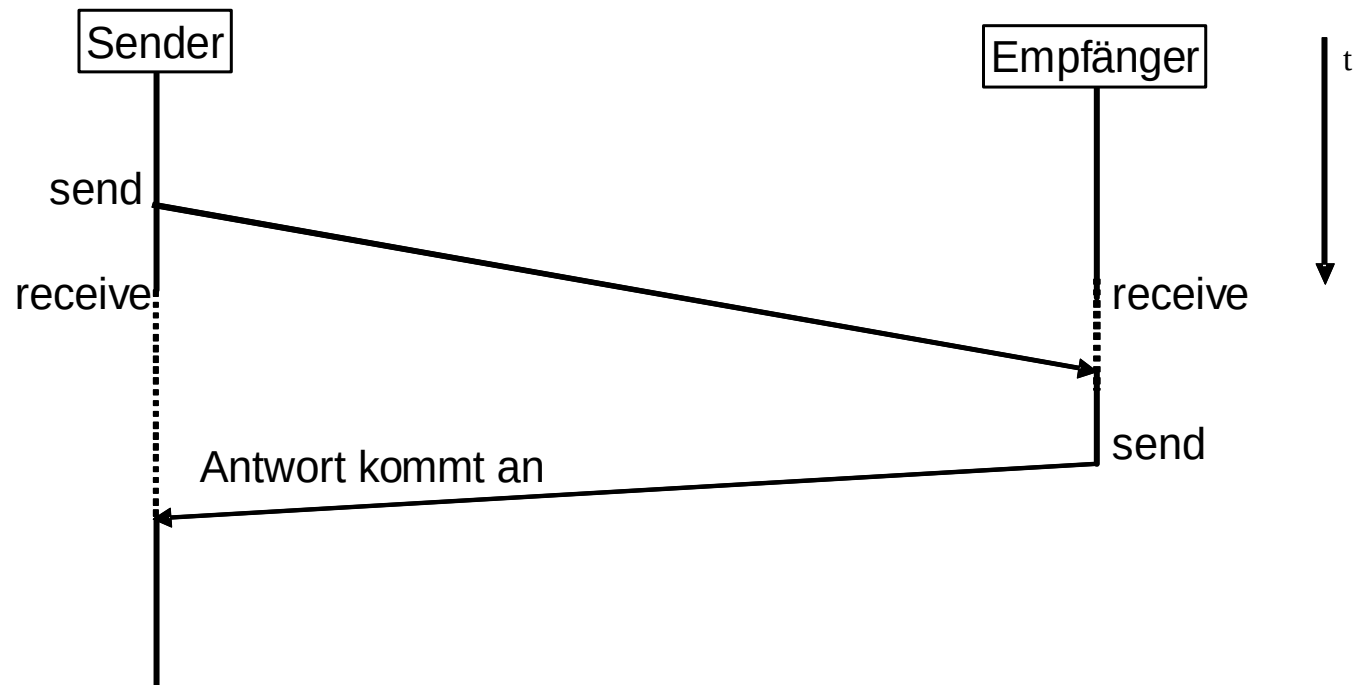
# Nachrichtenbasierte Kommunikation über Kommunikationsdienste

- Zwei Threads aus getrennten Adressräumen kommunizieren über Kommunikationsdienste
  - Z.B. über TCP oder UDP
  - Sockets-Schnittstelle



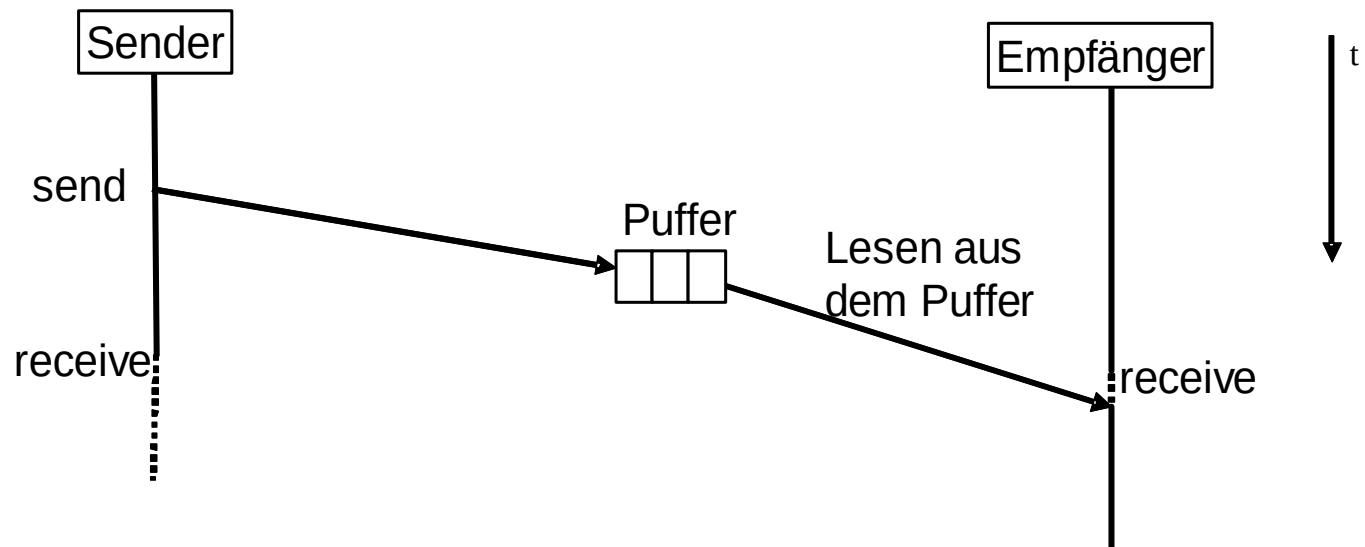
# Synchron versus asynchron

- Synchrone Kommunikation ist blockierend



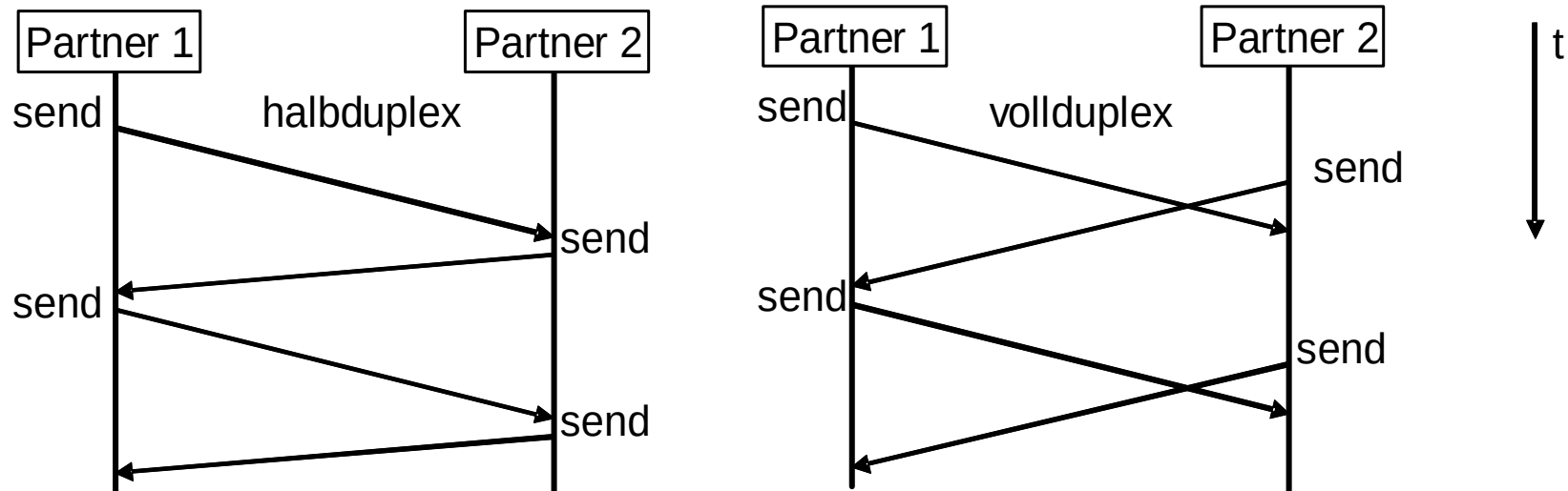
# Synchron versus asynchron

- Asynchrone Kommunikation ist beim Senden nicht blockierend



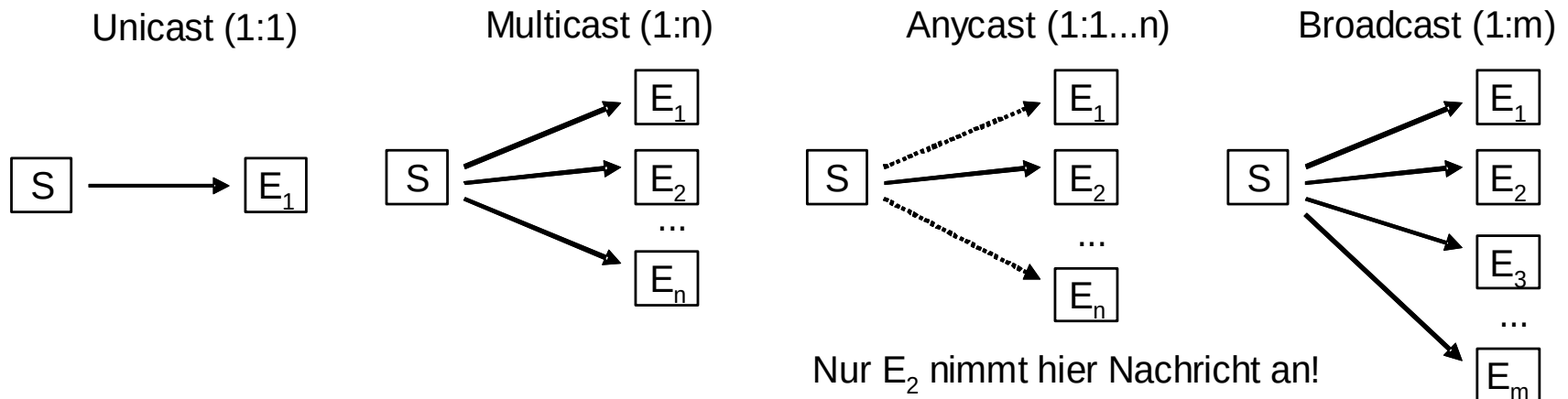
# Halbduplex- versus Vollduplexbetrieb

- **Halbduplex:** Nur einer der Partner sendet zu einer Zeit (Wechselbetrieb)
- **Vollduplex:** beide Partner können unabhängig voneinander senden (Gegenbetrieb)
- Auch simplex möglich (nur in eine Richtung)



# Varianten der Empfängeradressierung

- Unicast: nur ein Empfänger wird adressiert
- Alle anderen Varianten adressieren mehrere Empfänger





# Überblick

---

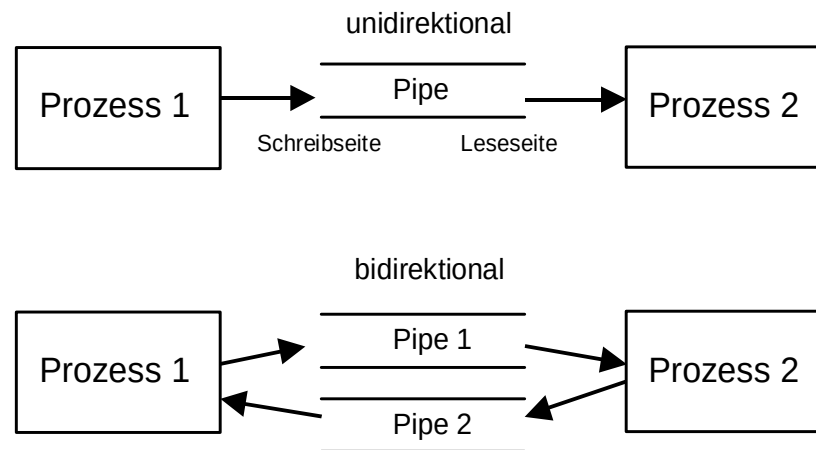
1. Einführung in die Grundbegriffe der Kommunikation
- 2. Lokale Kommunikation**
3. Verteilte Kommunikation

# Beispiele

- Unter Unix (je nach Derivat) und Windows gibt es verschiedene IPC-Mechanismen:
  - **Pipes** und **FIFO's** (Named Pipes) als Nachrichtenkanal
  - Nachrichtenwarteschlangen (Message Queues)
  - **Gemeinsam genutzter Speicher** (Shared Memory)
  - **Sockets** z.B. ia Loopback Netzwerk Interface
  - **Dateien**
- Vorhandene Synchronisationsmechanismen:
  - Semaphore, Mutexe, Signale, Datei-Locks

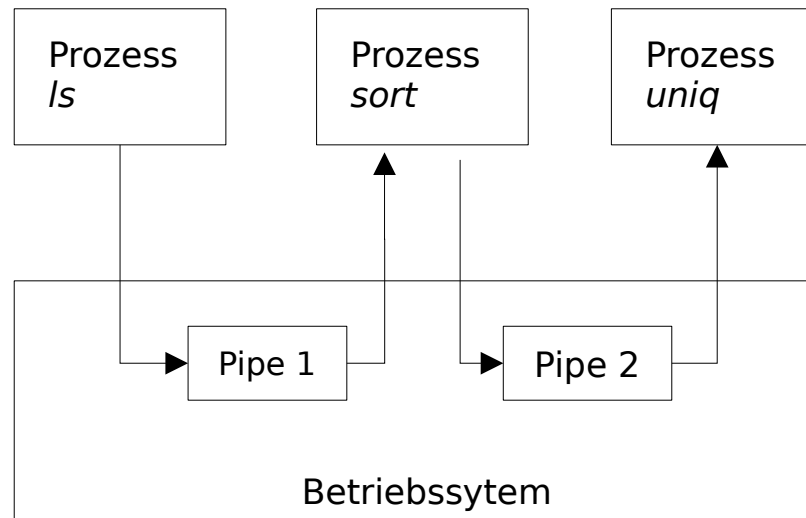
# Pipes

- Pipes: Spezieller unidirektionaler Mechanismus
- Unidirektionale und bidirektionale Kommunikation (bidi → Nutzung mehrerer Pipes)
- Bidirektionale Kommunikation über zwei Pipes kann sowohl halb- als auch vollduplex betrieben werden



# Pipes unter Unix

- **Pipes** werden u.a. genutzt, um die Standardausgabe eines Prozesses mit der Standardeingabe eines weiteren Prozesses zu verbinden
- Beispiel in Unix: *ls | sort | uniq* (vgl.: Stevens)



Unix-Kommandos:  
*ls | sort | uniq*

## Pipes: Programmierung (1)

- **Erzeugen** einer Pipe:
  - Unter Unix mit dem Systemaufruf *pipe()* oder *popen()*
  - Unter Windows mit *CreatePipe()*
- **Schließen** einer Pipe:
  - Unter Unix mit dem Systemaufruf *close()* oder *pclose()*
  - Unter Windows mit *closeHandle()*
- Elternprozess erzeugt Pipe und **vererbt** sie an den Kindprozess
- Man kann Pipes **blockierend** (Normalmodus) und nicht blockierend einsetzen. Blockierend bedeutet:
  - Wenn die Pipe voll ist blockiert der Sendeprozess
  - Wenn die Pipe leer ist blockiert der Leseprozess
  - Sinnvoll für Erzeuger-Verbraucher-Problem

# Pipes: Programmierung (2)

## ■ Beispielprogramm

```
// Filedeskriptoren für Pipe
int pipe_fds[2];

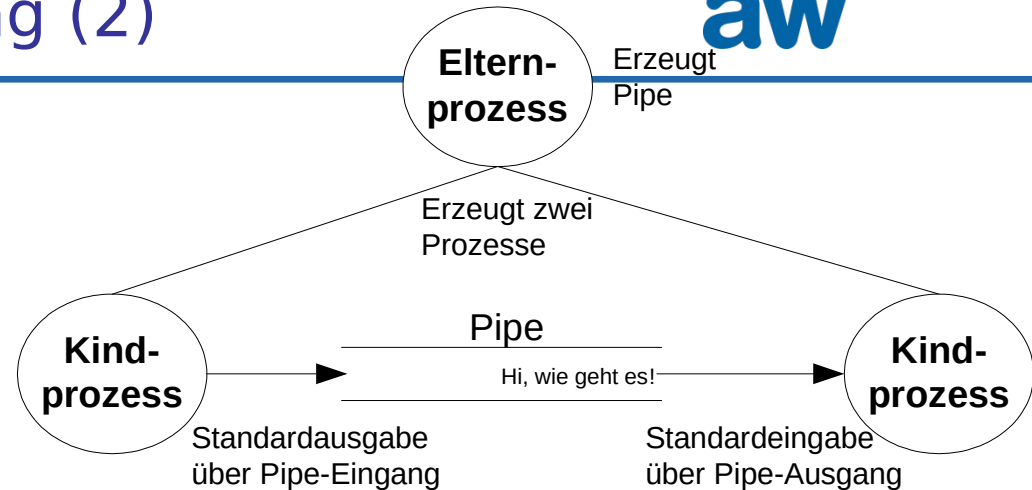
char *text = "Hi, wie geht's!\n";
char buffer[5];
int status;

pipe(pipe_fds);

if (fork() == 0) {
    // 1. Kindprozess
    // * Standardausgabe auf Pipe-Schreibseite (Pipe-Eingang) legen
    // * Pipe-Leseseite (Pipe-Ausgang) schließen (wird nicht benötigt)

    // pipe_fds[1] wird auf 1 (Standardausgabe) gelegt
    dup2(pipe_fds[1], 1);

    close(pipe_fds[0]);
    write (1, text, strlen(text)+1);
}
else{    // ... nächste Seite...
```



# Pipes: Programmierung (3)

## ■ Beispielprogramm ...

...

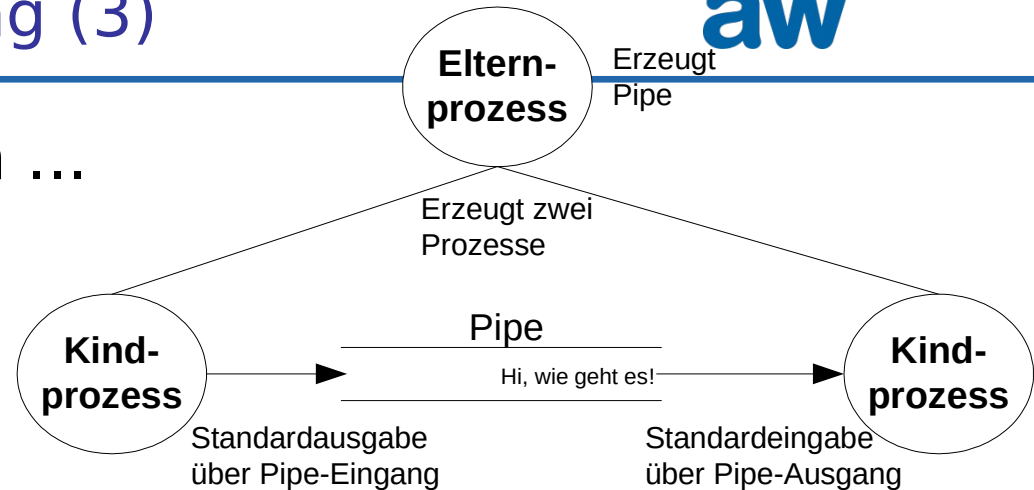
```

else{
    if (fork() == 0) {
        // 2. Kindprozess, Pipe-Leseseite (Pipe-Ausgang) auf
        // Standardeingabe umlenken und Pipe-Schreibseite
        // (Pipe-Eingang) schließen

        // pipe_fds[0] wird auf 0 (Standardeingabe) gelegt
        dup2(pipe_fds[0], 0);

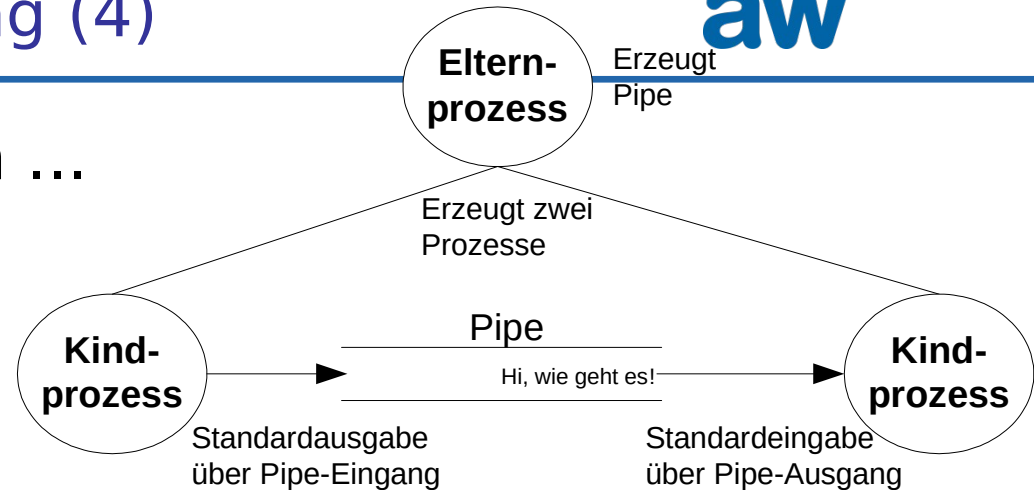
        close(pipe_fds[1]);
        while (count = read(0, buffer, 4))
        {
            // Pipe in einer Schleife auslesen
            buffer[count] = 0;    // String terminieren
            printf("%s", buffer) // und ausgeben
        }
    }
    // ... nächste Seite ...

```



# Pipes: Programmierung (4)

## ■ Beispielprogramm ...



```
...
else {
    // Im Vaterprozess: Pipe an beiden Seiten schließen und
    // auf das Beenden der Kindprozesse warten
    close(fds[0]);
    close(fds[1]);
    wait(&status);
    wait(&status);
}
exit(0);
}
```



# Überblick

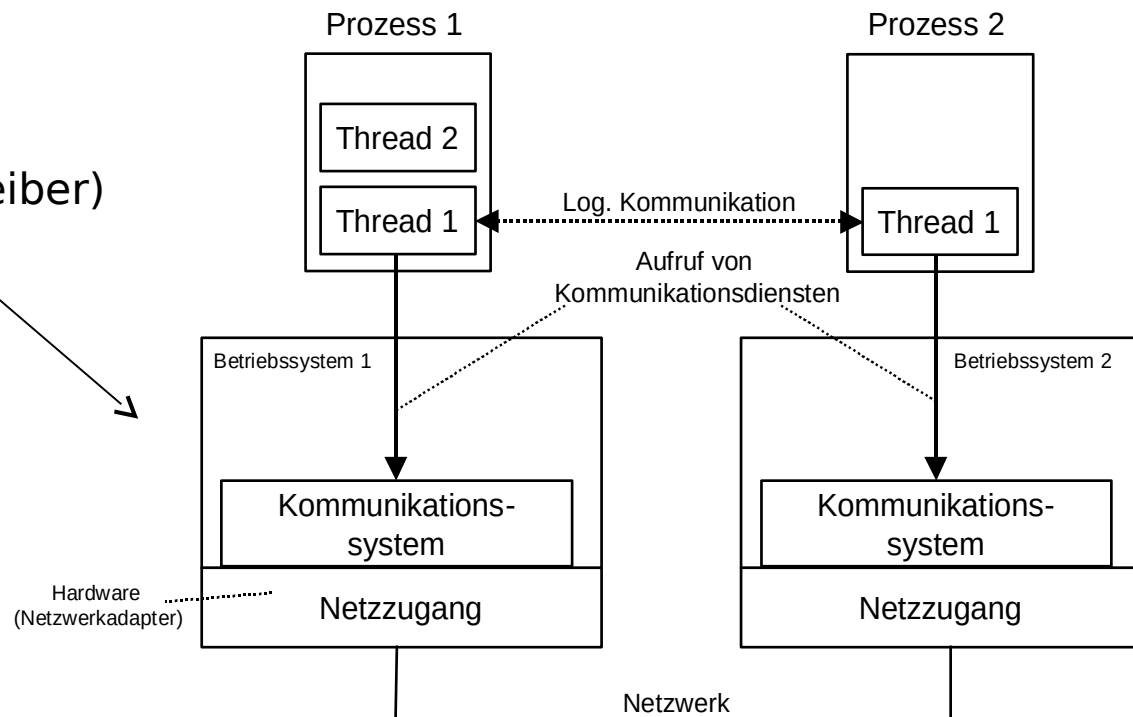
---

1. Einführung in die Grundbegriffe der Kommunikation
2. Lokale Kommunikation
- 3. Verteilte Kommunikation  
(über Rechnergrenzen hinweg)**

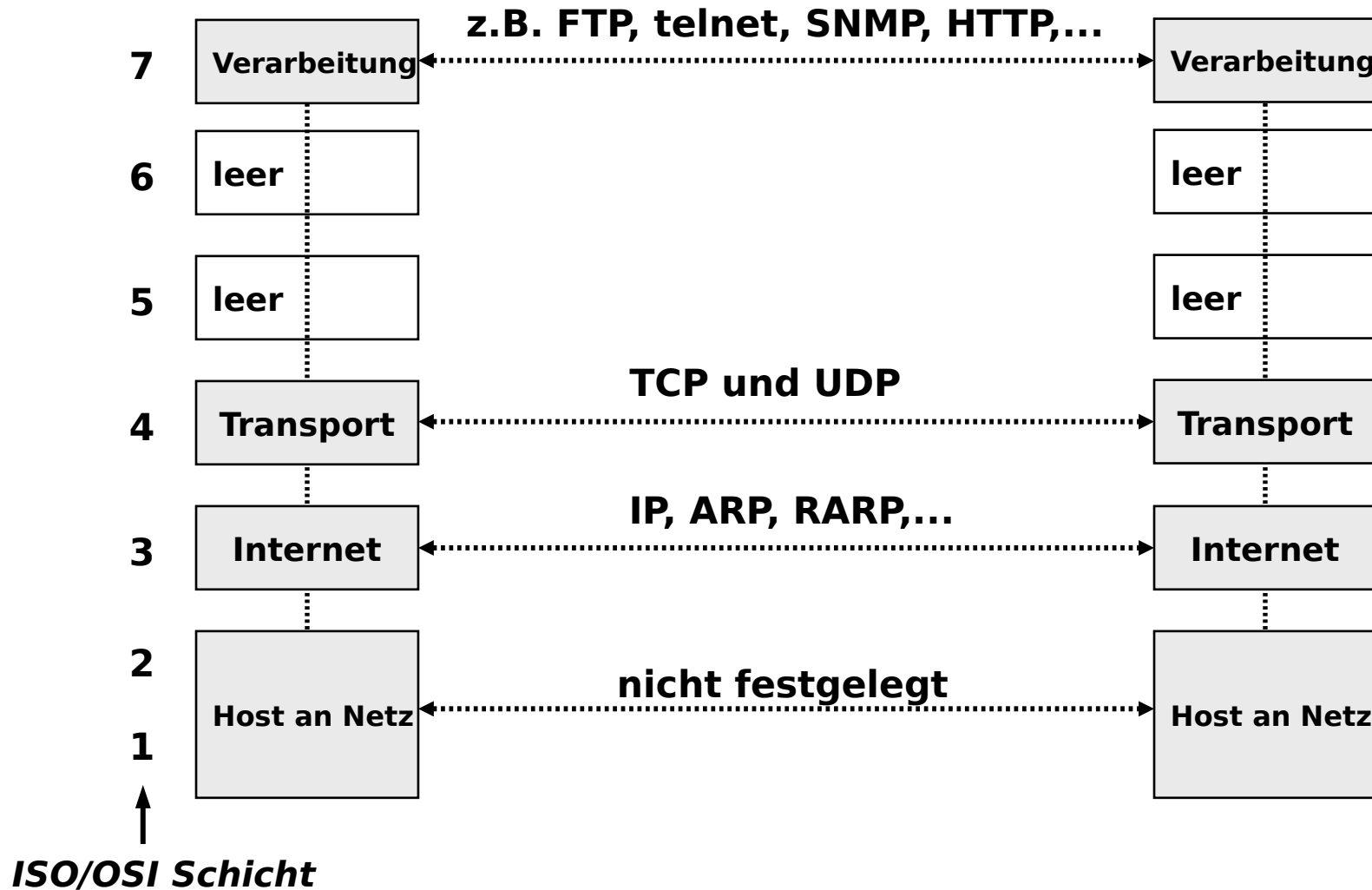
# Verteilte Kommunikation zwischen Prozessen/Threads

- Prozesse/Threads verschiedener Rechner kommunizieren
- Betriebssystem stellt Kommunikationssystem bereit
- Netzzugang erforderlich

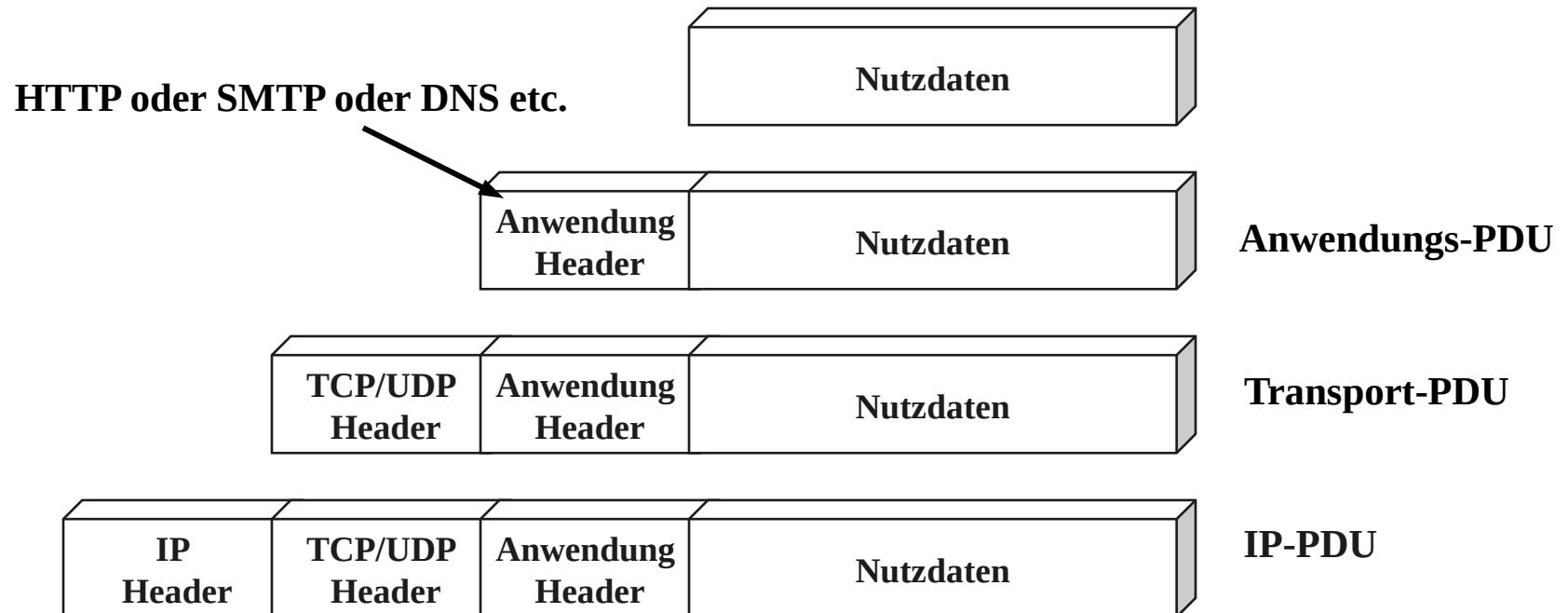
Einbettung ins Betriebssystem (Treiber)



# TCP-Referenzmodell



# Protokolle

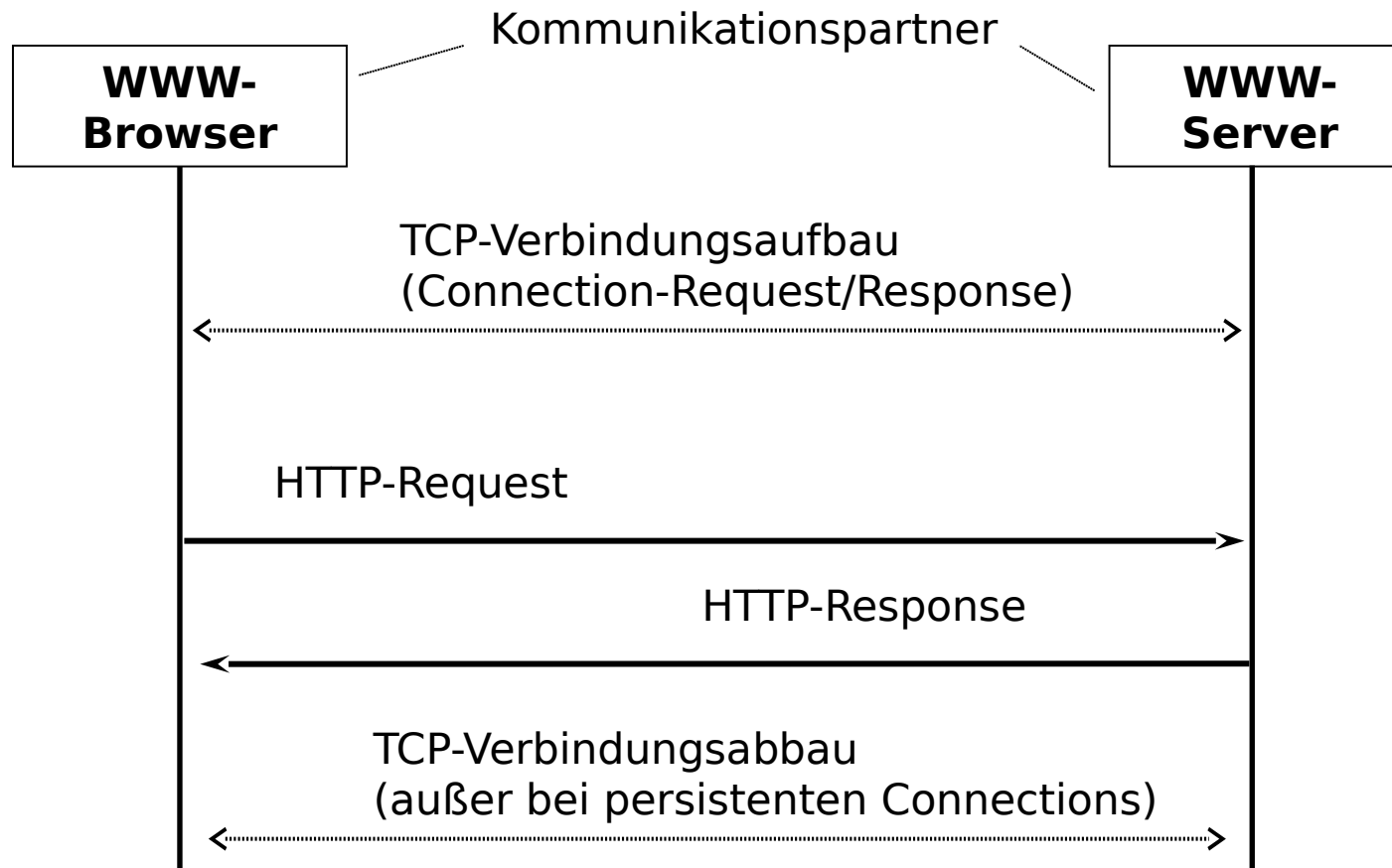


**PDU = Protocol Data Unit**

Protokolle sind über Regeln definiert  
→ Kommunizierende Automaten

# Kommunikation am Beispiel des HTTP-Protokolls

- Verbindungsorientiert, nachrichtenbasiert



# Zusammenfassung

---

1. Einführung in die Grundbegriffe der Kommunikation
2. Lokale Kommunikation
3. Verteilte Kommunikation → *mehr dazu in den Vorlesungen Datenkommunikation und Verteilte Systeme*

# Gesamtüberblick

---

- ✓ Einführung in Computersysteme
- ✓ Entwicklung von Betriebssystemen
- ✓ Architekturansätze
- ✓ Interruptverarbeitung in Betriebssystemen
- ✓ Prozesse und Threads
- ✓ CPU-Scheduling
- ✓ Synchronisation und Kommunikation
- 8. Speicherverwaltung
- 9. Geräte- und Dateiverwaltung
- 10. Betriebssystemvirtualisierung