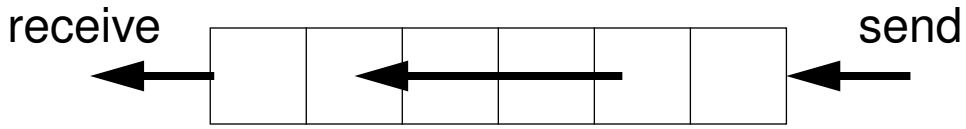
Asynchrone Botschaften

Prozesse senden und empfangen Botschaften über Kanäle.

Botschaft: Wert eines evtl. zusammengesetzten Datentyps oder Objekt einer Klasse.

Kanal: beliebig lange Schlange von Botschaften mit Operationen send und receive.



Operationen mit Kanälen:

- **send (b)**: fügt die Botschaft b an die Schlange des Kanals an;
  blockiert nicht (im Gegensatz zu send bei synchronen Botschaften)
- **receive()**: liefert die älteste Botschaft und entfernt sie aus dem Kanal;
  blockiert den aufrufenden Prozess solange der Kanal leer ist.
- **empty()**: liefert true, falls der Kanal leer ist; das Ergebnis ist **nicht notwendig aktuell**.

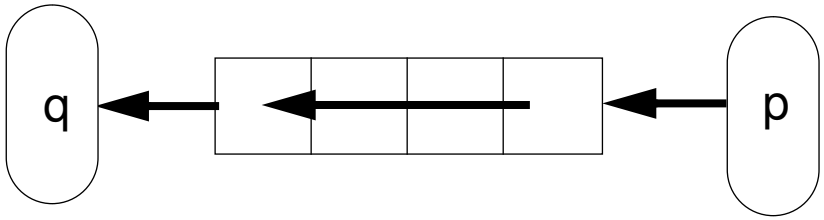
Alle Operationen werden atomar ausgeführt.

Kanal mit leeren Botschaften: dient nur zur Synchronisation; entspricht zählendem Semaphor

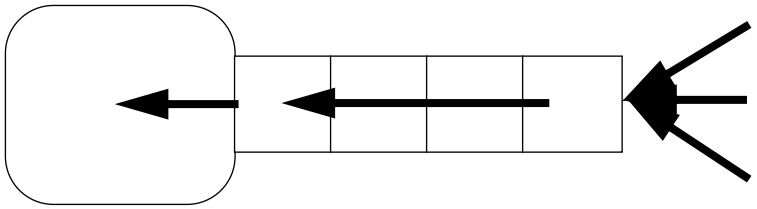
© 2000 bei Prof. Dr. Uwe Kastens

Prozesse und Kanäle

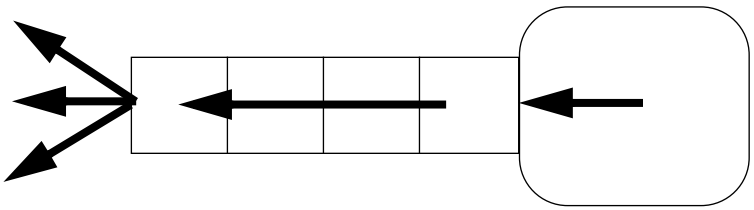
Link:
ein Sender mit **einem** Empfänger verbunden;
z. B. Prozesse bilden Ketten von Transformationen (Pipeline)



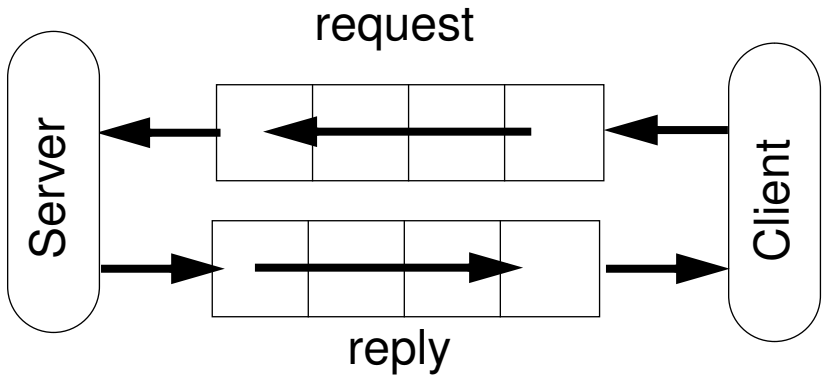
Input-Port eines Prozesses:
viele Sender ein Empfänger;
Kanal gehört zum empfangenden Prozess;
z. B. Server-Prozess empfängt Aufträge von vielen Client-Prozessen



Output-Port eines Prozesses:
ein Sender viele Empfänger;
Kanal gehört zum sendenden Prozess;
z. B. Prozess verteilt Aufträge an viele Bearbeite
(ungewöhnlich Struktur)



Paar von **Anfrage- und Antwort-Kanal**
request/reply;
enge Synchronisation,
z.B. von Client und Server



© 2000 bei Prof. Dr. Uwe Kastens

Kanäle in Java

```
public class Channel
{  // Implementation of a Channel using a Queue of messages
   private Queue msgQueue;

   public Channel ()
      { msgQueue = new Queue (); }

   public synchronized void send (Object msg)
      { msgQueue.enqueue (msg); notify(); }// wake a receiving process

   public synchronized Object receive ()
      {  while (msgQueue.empty())
            try { wait(); } catch (InterruptedException e) {}
         Object result = msgQueue.front();// if this process is woken,
         msgQueue.dequeue();                 // the Queue is not empty
         return result;
      }

   public synchronized boolean empty ()
      { return msgQueue.empty (); }
}
```

© 2000 bei Prof. Dr. Uwe Kastens

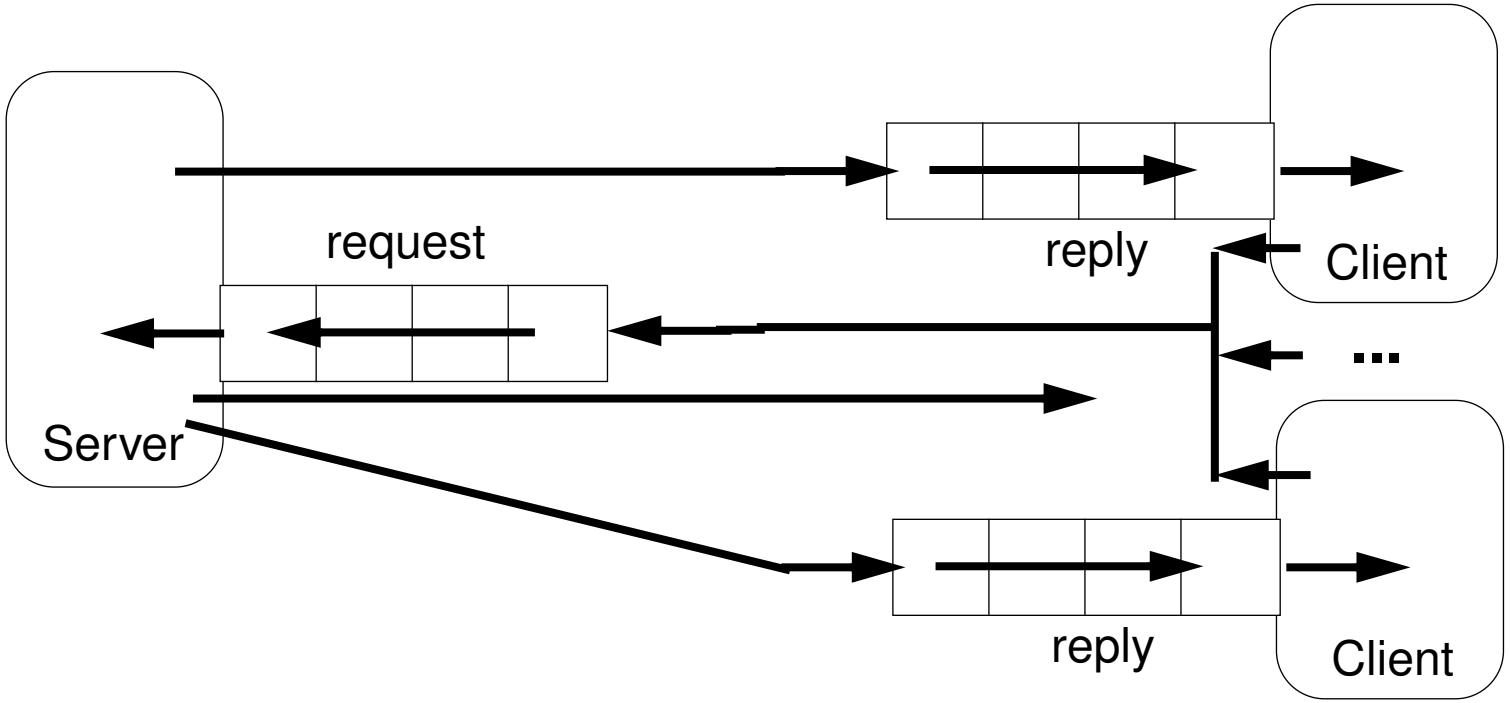
Botschaften in Java

```
import java.util.Vector;

public class BasicMessage
{              // Messages with an integral code and arbitrary arguments
   protected int kind;                    // encoding of the message kind
   protected Vector argList;// arbitrary number and types of arguments

   public BasicMessage () { argList = new Vector(); } // constructors
   public BasicMessage (int k) { kind = k; argList = new Vector (); }
   public BasicMessage (int k, Vector args) { kind=k; argList=args;}

   protected void setKind (int k) { kind = k; }    // kind of message
   protected int getKind () { return kind; }

   public void setArg (int index, Object arg) // set and get arguments
      { argList.insertElementAt (arg, index); }

   public Object getArg (int index)
      { return argList.elementAt (index); }

   public Vector getArgList ()                          // copy arguments
      { Vector listCopy = (Vector) argList.clone();
         return listCopy;
}     }
```

© 2000 bei Prof. Dr. Uwe Kastens

Client/Server mit asynchronen Botschaften

Ein oder mehrere gleichartige **Server-Prozesse bedienen Anfragen von Client-Prozessen**.



request-Kanal:

Input-Port des Servers;
auch bei mehreren Servern nur ein request-Kanal

reply-Kanal

zu jedem Client (Input-Port);
wird z. B. mit der request-Botschaft versandt

© 2000 bei Prof. Dr. Uwe Kastens

Server-Prozess

Der Server ist ein Prozess, er verarbeitet Anfragen in permanenter Schleife.

Unterschiedliche Anfragen (Operationen) durch verschiedene Arten von Botschaften.

```
public void run ()
{  while (running)
   {                                                  // Anfrage empfangen
      BasicMessage msg = (BasicMessage)servChan.receive();
      switch (msg.getKind())
      {  case REQUEST:  // Operation bearbeiten und Antwort senden:
            ...   ((Channel)msg.getArg(0)).send(...);
         case RELEASE:
            ...   ((Channel)msg.getArg(0)).send(...);
}  }  }
```

Antwortkanal ist in der Anfragebotschaft enthalten.

Falls **Anfragen nur unter bestimmten Bedingungen** erfüllt werden können,
werden sie in einer **Schlange** des Servers gespeichert.

Der Server-Prozess bearbeitet die Anfragen streng sequentiell,
deshalb ist gesichert, dass kritische Abschnitte nicht Verzahnt werden.

Server-Prozess als **„aktiver Monitor“**.

© 2000 bei Prof. Dr. Uwe Kastens

Server-Prozesse in Java

```
public class RessourceServer extends Thread
{  private Channel servChan; // this server receives messages from it
   private String id;

   public static final int REQUEST = 1;          // kinds of messages
   public static final int RELEASE = 2;

    private int avail = 5;           // number of available ressources
   private Queue pending;                          // pending requests

   public RessourceServer (Channel c, String name)     // constructor
      { servChan = c; id = name; pending = new Queue ();}

   public void stopIt () { running = false; } // external termination
   private boolean running = true;

   public void run ()                          // requests are processed
   {  while (running)
      {  BasicMessage msg = (BasicMessage)servChan.receive();
         switch (msg.getKind())
         {  case REQUEST: ...
            case RELEASE: ...
}  }  }  }
```

© 2000 bei Prof. Dr. Uwe Kastens

Client-Prozesse in Java

```
import java.util.Random;

public class RessourceClient extends Thread
{  private Channel clientReply;       // reply channel of this client
   private Channel serverChan;// the request channel of the server(s)

   private String id; private Random rand;

   public RessourceClient (Channel serv, String name)
      {  serverChan = serv; clientReply = new Channel ();
         id = name; rand = new Random (name.hashCode());
      }

   public void stopIt () { running = false; }
   private boolean running = true;

   public void run ()
   {  BasicMessage reqMsg = new BasicMessage(RessourceServer.REQUEST);
      reqMsg.setArg (0, clientReply); ...// reusable message instances

      while (running)
      {  serverChan.send (reqMsg); clientReply.receive (); ...
         serverChan.send (relMsg); clientReply.receive (); ...

}  }  }
```

© 2000 bei Prof. Dr. Uwe Kastens