Receive ohne Blockieren

Wenn mehrere Prozesse von einem Kanal empfangen, kann

```
if (!ch.empty()) msg = ch.receive();
```

trotz der Bedingung blockieren.
Wenn von mehreren Kanälen empfangen werden soll, ist das unerwünscht.

Die Kanaloperationen werden deshalb um eine Operation erweitert:

```
public class Channel
{  ...
   public synchronized Object receiveMsgOrNull ()
   {  if (msgQueue.empty()) return null;
      Object result = msgQueue.front();
      msgQueue.dequeue();
      return result;
}  }
```

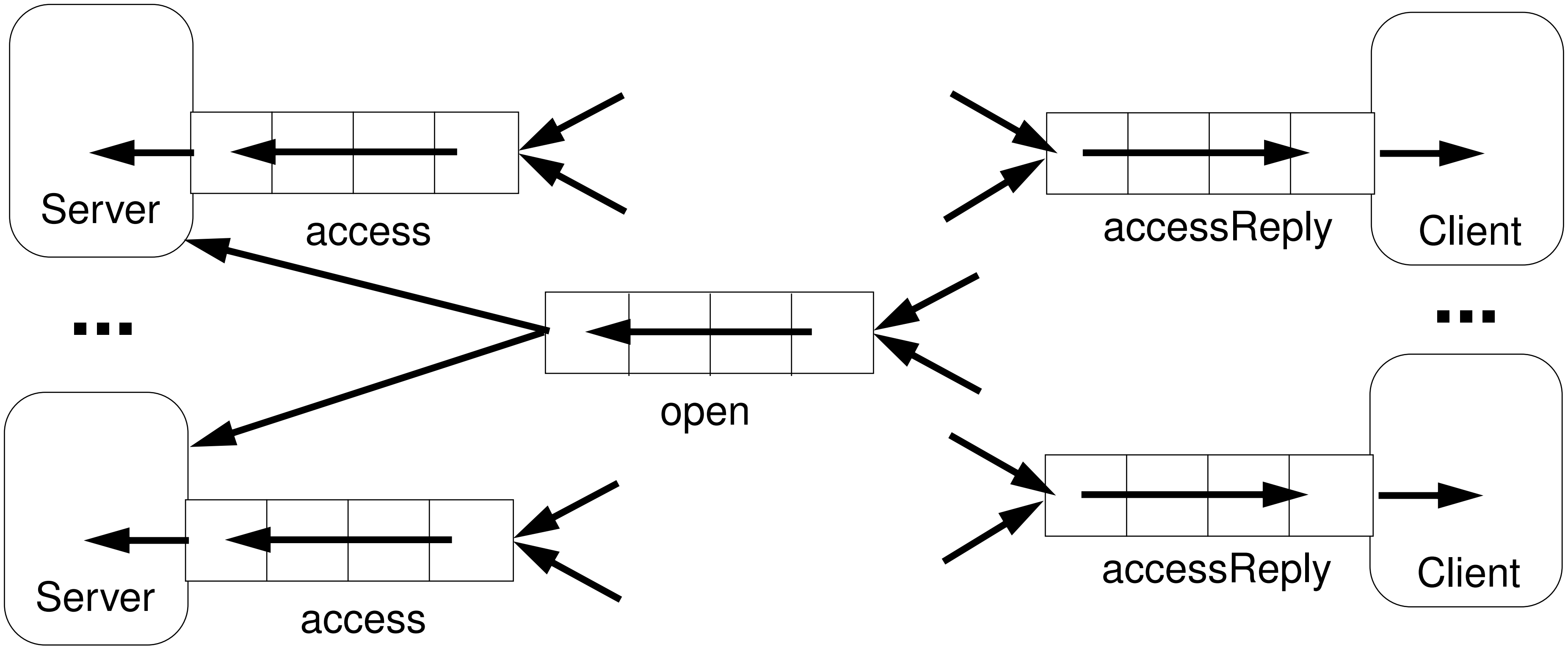
Abfrage mehrerer Kanäle:

```
while (msg == null)
{  if ((msg = ch1.receiveMsgOrNull()) == null)
   if ((msg = ch2.receiveMsgOrNull()) == null)
      Thread.sleep (500);
}
```

Konversationsfolgen zwischen Client und Server

Anwendungsmuster (z. B. Fileserver):

- mehrere gleichartige Server bedienen mehrere Clients
- Client schickt Eröffnungsanfrage auf allgemeinem Kanal (open)
- ein Server übernimmt den Auftrag und führt mit dem Client eine Konversation nach bestimmten Protokoll, z. B. ((read readReply)* | (write writeReply)) close closeReply
- Antwortkanäle werden mit den Botschaften versandt.



Broadcast im Netz

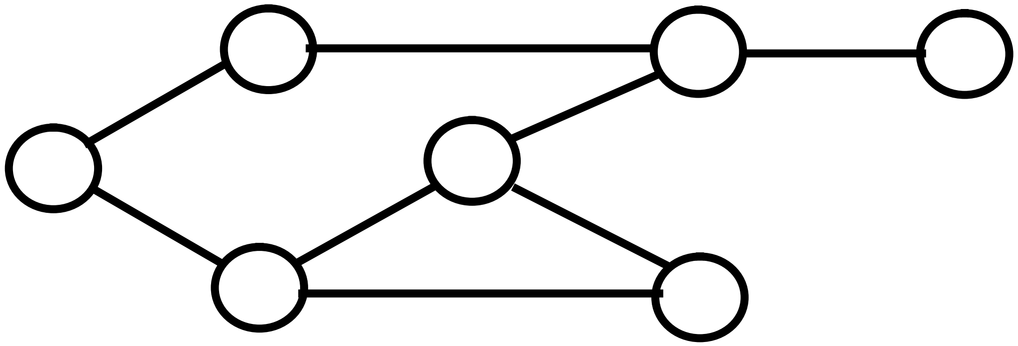
Netz: unregelmäßiger, zusammenhängender, bidirektionaler Graph;
Knoten: Prozess mit einem Input-Port;
Kante: Paar von Referenzen auf Input-Ports der beiden Prozesse an der Kante

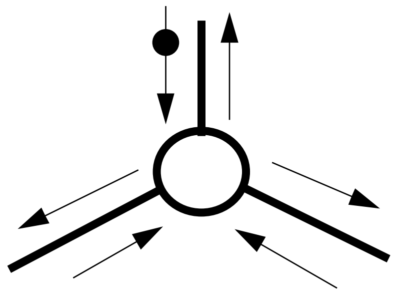
Knoten kennen nur ihre unmittelbaren Nachbarn, deren Identität und Input-Ports

Aufgabe: Von einem Initiator-Knoten aus eine Botschaft an alle Knoten im Netz verteilen.
Am Ende sollen alle Kanäle leer sein.

Methode:

- Knoten wartet auf Botschaft am Input-Port
- Nach erstem Empfang sendet er an **alle** n Nachbarn (auch an den Sender der Botschaft)
- Kanal des ersten Empfangs ist eine Kante des **Spannbaumes**
- Knoten empfängt n-1 redundante Botschaften von den nicht-Spannbaumkanten





insgesamt 2*|Kanten| Botschaften

Probe/Echo im Netz

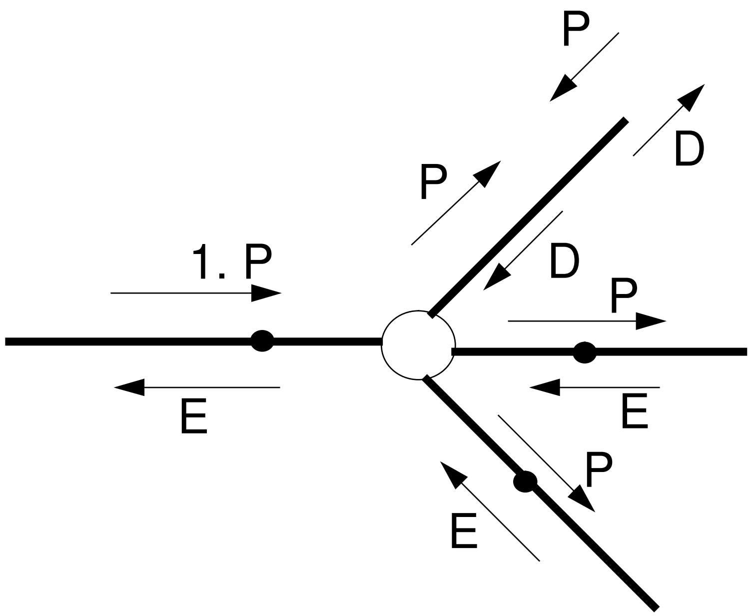
Ein Initiator ruft von allen Knoten im Netz Information ab (Probe);
auf dem Weg durch das Netz wird sie zusammengefasst (Echo),
z. B. Summe lokaler Werte, Topologie des Graphen, globalen Zustand berechnen

Methode:

- Anfragen (Probe) verteilen wie Broadcast
- dabei Spannbaumkanten bestimmen (jeweils erster Empfang)
- Antwort (Echo) auf Spannbaumkanten zurücksenden

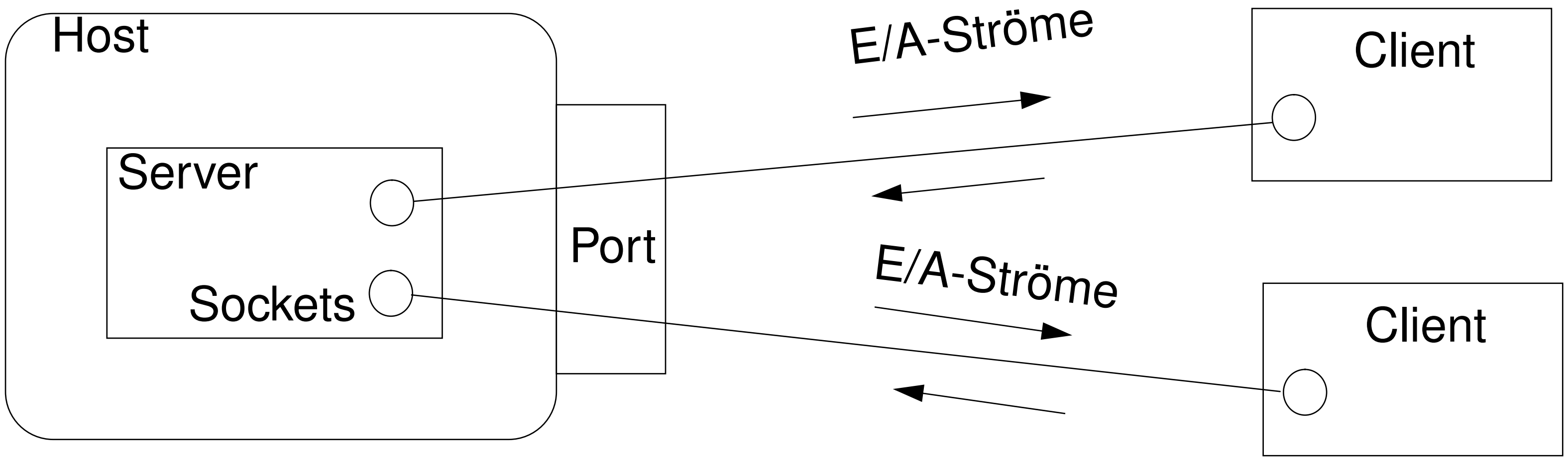
Aktionen eines Knotens im Einzelnen:

- Jeder Knoten hat einen Input-Port für Probes und Echos zusammen
- Nach erstem Empfang eines Probes von s: Probe an alle n Nachbarn außer s senden
- Alle weiteren eintreffenden Probes mit einer Dummy-Botschaft beantworten
- Insgesamt müssen n-1 Dummies und Echos eintreffen, dann Echo berechnen, an s senden



Rechnerverbindungen über Ports und Sockets

Port:
- Abstrakte Verbindungsstelle eines Rechners; numerisch codiert;
- Server-Prozesse können bestimmte Ports bedienen, z. B. 13: Datum und Uhrzeit;
- Client-Prozesse auf anderen Rechnern können über
- Rechnernamen und Port-Nummer Verbindung aufnehmen



Socket:
- Abstraktion der Netzwerk-Software zur Kommunikation über Ports.
- Sockets werden aus Rechneradresse und Portnummer erzeugt
- Mehrere Sockets auf einem Port zur Bedienung mehrerer Clients
- Auf einem Socket können E/A-Ströme eingerichtet werden.

Sockets und E/A-Ströme

Rechneradresse bestimmen:

```
InetAddress  addr1 = InetAddr.getByName ("java.sun.com"),
             addr2 = InetAddr.getByName ("206.26.48.100"),
             addr3 = InetAddr.getLocalHost();
```

Client-Seite, Socket mit Verbindung zum Server-Rechner erzeugen

```
Socket myServer = new Socket (addr, port);
```

E/A-Ströme auf Socket einrichten:

```
BufferedReader in =
   new BufferedReader
      (new InputStreamReader (myServer.getInputStream()));

PrintWriter out =
   new PrintWriter (myServer.getOutputStream(), true);
```

Server-Seite, speziellen Socket erzeugen, eintreffende Verbindungen annehmen:

```
ServerSocket listener = new ServerSocket (port);
...
Socket client = listener.accept(); ... client.close();
```