Ablauf

- Max Haustein (haustein@mi.fu-berlin.de)
- Aufgaben
  - Dienstag im Netz
  - Mittwoch Vorbesprechung
  - Abgabe/Bearbeitung bis nächsten Freitag
  - ~2er Gruppen
  - Abgabeform...

Java New IO (NIO)

- "neu" seit Java 1.4
- Stream-Konzept wird ersetzt
- betriebssystemnah, effizient
- Lesen und Schreiben über Kanäle
  `java.nio.Channel, FileChannel, SocketChannel, ...`
- Daten müssen in Puffer verpackt werden
  `java.nio.Buffer, ByteBuffer, IntBuffer, ...`

java.nio.Buffer

- **Array + Verwaltung**
- **Attribute** `position, limit, capacity`
- **Verwendung:**
  - `clear()`**: löschen/initialisieren**
    `(position=0, limit=capacity)`
  - **Füllen inkrementiert** `position`
  - `flip()`**: vorbereiten zum Auslesen**
    `(limit=position, position=0)`
  - **Auslesen**
    `(position = 0..limit)`

Datei kopieren

```
FileInputStream fin = new FileInputStream(args[0]);
FileChannel fcin = fin.getChannel();

FileOutputStream fout = new FileOutputStream(args[1]);
FileChannel fcout = fout.getChannel();

ByteBuffer buffer = ByteBuffer.allocate(1024);
for (;;) {
    buffer.clear(); // initialisieren
    int r = fcin.read(buffer);
    if (r == -1)
        break;
    buffer.flip(); // zum Schreiben vorbereiten
    fcout.write(buffer);
}
```

Disjunktives Warten

- "gleichzeitig" auf mehrere Ereignisse warten,
  ohne Polling, ohne Threads
- Abbildung auf den Systemaufrufs `select`
- Vorgehen:
  - interessante Kanäle bei `Selector` registrieren
  - select()-Aufruf
  - Teilmenge mit bereiten Kanälen traversieren

```
SocketChannel sc1 = ..., sc2 = ...;

// Selector erzeugen und Kanäle registieren
Selector selector = Selector.open();
sc1.register(selector, SelectionKey.OP_READ);
sc2.register(selector, SelectionKey.OP_READ); // ACCEPT | CONNECT | WRITE

while (true) {
    int ready = selector.select(5000);
    if (ready == 0) continue; // ...Timeout behandeln...
    Set selectedKeys = selector.selectedKeys();
    Iterator it = selectedKeys.iterator();
    while (it.hasNext()) {
        SelectionKey key = (SelectionKey) it.next();
        if ((key.readyOps() & SelectionKey.OP_READ)==SelectionKey.OP_READ){
            SocketChannel sc = (SocketChannel) key.channel();
            // ...Daten vom Kanal lesen...
        }
        it.remove();
    }
}
```

Alternating-Bit-Protokoll

- zuverlässiger Simplex-Kanal über unzuverlässigen Duplex-Kanal realisiert
- Empfang von Nachrichten wird quittiert
- Durchnummerierung ermöglicht Zuordnung von Bestätigung zu Nachricht
- Durchnummerierung mod 2

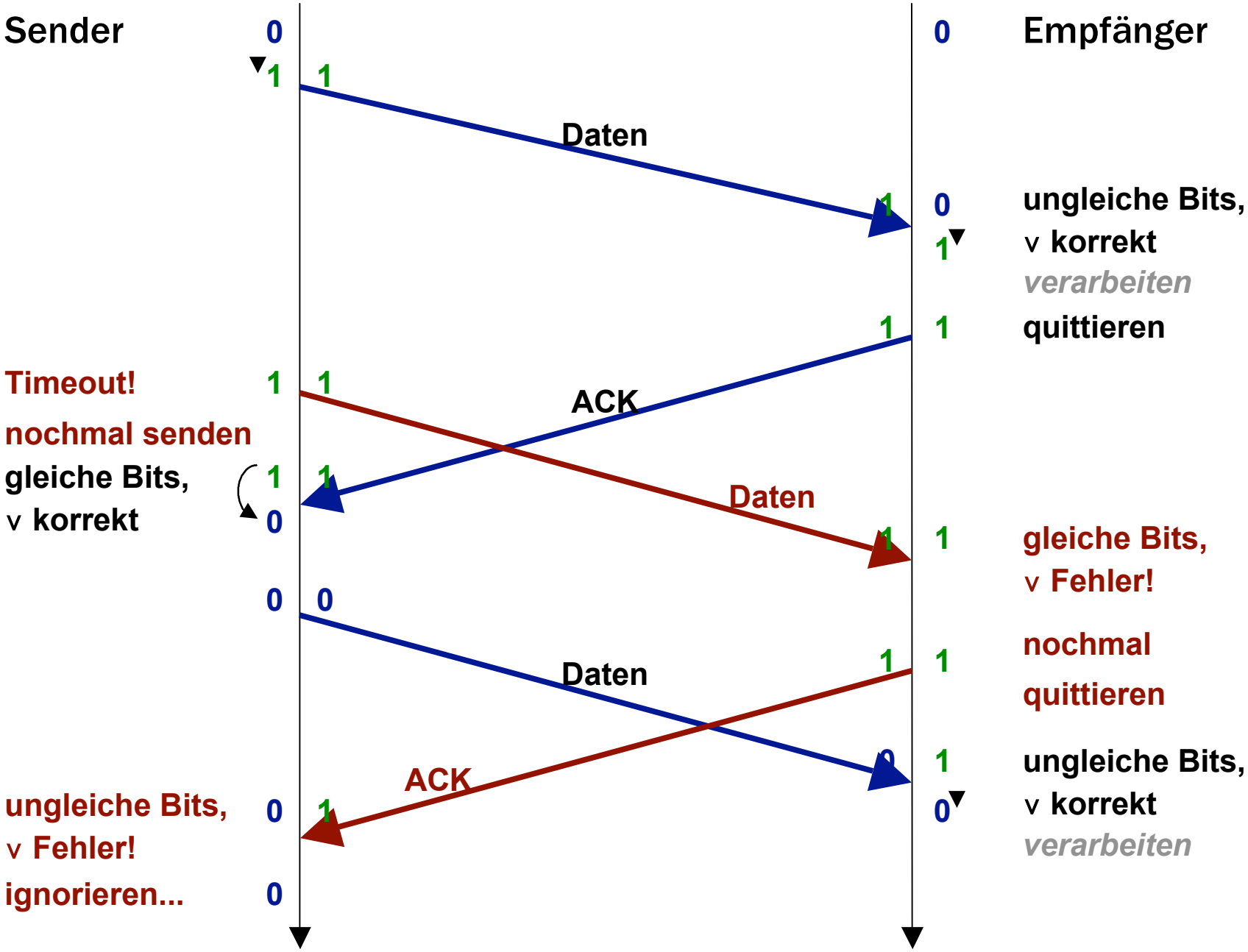
Alternating-Bit-Protocol (Pseudocode)

Sender:

```
boolean bit = false;
do {
    if(ack.bit == bit) {
        data = produce();
        bit = !bit;
    }
    msg.send(data,bit);
    ack.receive(timeout);
} while(true);
```

Empfänger:

```
boolean bit = false;
do {
    msg.receive();
    if(msg.bit != bit) {
        consume(msg.data);
        bit = !bit;
    }
    ack.send(null,bit);
} while(true);
```


Sender
Empfänger
0
1
1
Daten
1
0
1
ungleiche Bits,
v korrekt
verarbeiten
1
1
quittieren
Timeout!
nochmal senden
gleiche Bits,
v korrekt
1
1
ACK
1
1
Daten
1
1
gleiche Bits,
v Fehler!
0
0
Daten
1
1
nochmal
quittieren
1
0
ACK
ungleiche Bits,
v Fehler!
ignorieren...
0
1
1
0
ungleiche Bits,
v korrekt
verarbeiten
0

HTTP-Server/-Client

- java HttpServer <port> <docroot>
  z.B. java HttpServer 8080 /home/me/mysite
- java HttpClient <url>
  z.B. java HttpClient http://myhost.fu-berlin.de:8080/mydir/page.html
- Ablauf
  - Client baut Verbindung zum Server auf
    (Parsen über java.net.URL)
    und schickt Anfrage
    `GET mydir/page.html`
    terminiert mit CR LF (\r\n)
  - Server liefert Datei
    /home/me/mysite/mydir/page.html
  - Client gibt Datei aus
- Sockets verwenden (TCP)