Verteiltes Rechnen mit JPVM

- Verteiltes Raytracing
  - mehrere Arbeiter berechnen Teilbilder
- Koordinator
  - erzeugt (entfernte) Arbeiter
  - verteilt wiederholt Rechenjobs und empfängt Ergebnisse
  - stellt Ergebnis dar
  - beendet Arbeiter

JPVM Beispiel

- Hauptprogramm
  - erzeugt zwei entfernte Tasks
  - schickt jedem eine Nachricht
  - empfängt Antwort und gibt sie aus
- Task
  - empfängt Nachricht (blockierend)
  - berechnet Ergebnis
  - schickt Antwort

```
import jpvm.*;

public class jpvmDemo {

    public static void main(String[] args) throws jpvmException {

        jpvmEnvironment jpvm = new jpvmEnvironment();
        int tasks = 2;

        // Tasks erzeugen
        jpvmTaskId tids[] = new jpvmTaskId[tasks];
        jpvm.pvm_spawn("jpvmDemoTask", tasks, tids);

        // Nachricht an den ersten Task schicken
        jpvmBuffer buf = new jpvmBuffer();
        buf.pack(3); // überladen für primitive Datentypen; auch mehrfach hintereinander
        jpvm.pvm_send(buf, tids[0], 42);

        // Nachricht an den zweiten Task schicken
        buf = new jpvmBuffer();
        buf.pack(2);
        jpvm.pvm_send(buf, tids[1], 42);
```

```
...

// Antworten lesen
        for (int task = 0; task < tasks; task++) {
            // Nachricht mit Tag 23 empfangen
            jpvmMessage message = jpvm.pvm_recv(23);
            jpvmTaskId from = message.sourceTid;

            // Daten (int[]) aus dem Puffer lesen
            int n = message.buffer.upkint();
            int[] answer = new int[n];
            message.buffer.unpack(answer, n, 1);
            System.out.println("antwort von " + from);
            for (int i = 0; i < n; i++)
                System.out.print(answer[i] + " ");
            System.out.println();
        }
        jpvm.pvm_exit();
    }
}
```

```
public class jpvmDemoTask {
    public static void main(String[] args) throws jpvmException {
        jpvmEnvironment jpvm = new jpvmEnvironment();

        // Nachricht mit Tag 42 empfangen
        jpvmMessage message = jpvm.pvm_recv(42);
        int n = message.buffer.upkint();

        // Berechnung durchführen
        int[] answer = new int[n];
        for (int i = 0; i < n; i++)
            answer[i] = i*2;

        // zurückschicken
        jpvmBuffer buf = new jpvmBuffer();
        buf.pack(n);
        buf.pack(answer, n, 1);
        jpvm.pvm_send(buf, jpvm.pvm_parent(), 23);

        jpvm.pvm_exit();
    }
}
```

System aufsetzen

- CLASSPATH setzen(!)
- Daemons starten
  - host1> **java jpvm.jpvmDaemon**
    jpvm daemon: host1.inf.fu-berlin.de, port #37046
    host2> **java jpvm.jpvmDaemon**
    jpvm daemon: host2.inf.fu-berlin.de, port #36030
- auf einem Host Console starten und *andere* Hosts registrieren
  - host1> **java jpvm.jpvmConsole**
    jpvm> **add**
    Host name : **host2.inf.fu-berlin.de**
    Port number : **37046**
    jpvm> **ps**

JPVM Hinweise

- modifizierte Versionen auf der Veranstaltungs-Webseite
- aufgesetztes System nur bedingt "wiederverwendbar"
- erzeugte Tasks haben keine Standardausgabe

Java: berechnete Bilddaten darstellen

```
class PixelPanel extends JPanel {
    BufferedImage bi;
    public void renderImage() {
        int width = getWidth();
        int height = getHeight();
        bi = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        for (int row = 0; row < height; row++) {
            for (int col = 0; col < width; col++) {
                int r = (int) ( (double)col/width*255 );
                int g = (int) ( (double)(width-col)/width*255 );  // Farbverlauf berechnen
                int b = (int) ( (double)row/height*255 );
                bi.setRGB(col, row, (r<<16 | g<<8 | b));  // alternativ: setRGB(..., int[] data, ...)
            }
        repaint();
    }
    public void paint(Graphics g) {
        if (bi != null) g.drawImage(bi, 0, 0, this);
    }
}
```