Verteilte Algorithmen und Datenstrukturen

Kapitel 4: Einführung in verteilte Programmierung

Prof. Dr. Christian Scheideler

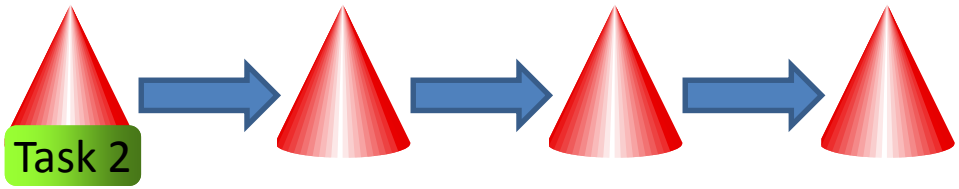
Insitut für Informatik

Universität Paderborn

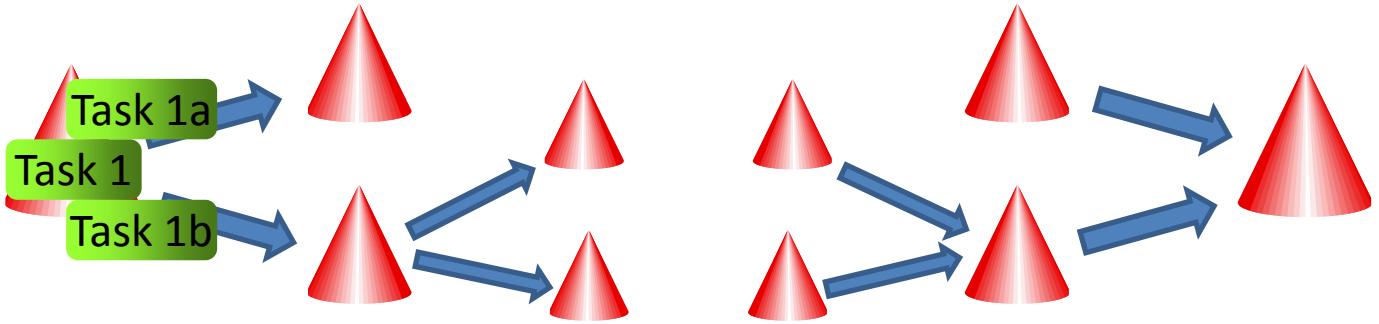
Verteilte Programmierung

Grundprinzip: Arbeitsteilung

- Sequentielle Arbeitsteilung (Flow-Based Progr.)



- Parallele Arbeitsteilung (MapReduce Paradigma)



Verteilte Programmierung

Zoo an Programmiersprachen:

Actor Modell:

- Axum, Elixir, Erlang, Janus, SALSA, Scala, Smalltalk,…

Dataflow:

- CAL, E, Joule, LabView, Lustre, Signal, SISAL,…

Verteilt:

- Bloom, Emerald, Go, Julia, Limbo, MPD, Oz, Sequoia, SR,…

Funktional:

- Concurrent Haskell, Concurrent ML, Clojure, Elixir, Erlang, Id, MultiLisp, SequenceL,…

Multithreaded:

- C=, Cilk, Clojure, Go, Java, ParaSail, SequenceL,…

Objektorientiert:

- mC++, Ada, C*, C++ AMP, Charm++, D, Emerald, Go, Java, ParaSail, Smalltalk,…

Siehe z.B. http://en.wikipedia.org/wiki/Concurrent_computing

Verteilte Programmierung

Unser Ansatz:

- Prozesse, die sich beliebig miteinander vernetzen können
- Prozesse können eigenständig sequentiell Aktionen ausführen
- Prozesse arbeiten und kommunizieren asynchron
- Daten müssen Prozessen zugeordnet sein, da es keinen Speicher außerhalb von Prozessen gibt

Pseudocode

Wie in objektorientierter Programmierung:

Subject ⟨Subjektname⟩: // deklariert Prozesstyp
    lokale Variablen
    Aktionen

Allgemeine Formen einer Aktion:

⟨Aktionsname⟩(Objektliste) →
    Befehle in Pseudocode

⟨Aktionsname⟩: ⟨Prädikat⟩ →
    Befehle in Pseudocode

Spezielle Aktionen:

init(Objektliste) → // Konstruktur
    Befehle in Pseudocode

timeout → // bei jedem timeout (was periodisch erfolgt)
    Befehle in Pseudocode

Pseudocode

- Zuweisung durch :=
- Schleifen (for, while, repeat)
- Bedingtes Verzweigen (if – then – else)
- (Unter-)Programmaufruf/Übergabe (return)
- Kommentar durch { }
- Blockstruktur durch Einrückung
- Aufruf einer Aktion über Subjektreferenz: $\leftarrow$
- Referenzvariable leer: $\perp$, Menge leer: $\varnothing$
- Erzeugung neuer Subjekte und Objekte: new
  (new aktiviert init im Subjekt)

Programmieren mit Java

- Wir verwenden eine einfache Subject.java Library, die einen Prozesstyp namens „Subject“ deklariert (welcher vom Java-Thread abgeleitet ist), für den folgende Methoden vom Nutzer implementiert werden müssen:
- init(): initialisiert den Prozess
- onMessageReceived(Object message): wird aufgerufen, wenn Anfrage empfangen wird
- onTimeout(): wird aufgerufen, wenn Timeout ausgelöst wird. Das geschieht laut Vorgabe alle 100 ms.

Beispiele

- Hello World: erzeugt Prozess, der „Hello World“ ausgibt
- Chain: erzeugt Kette aus 5 Prozessen
- Clique: erzeugt eine Clique aus 10 Prozessen
- Client & Server: erzeugt Server mit Broadcast Service und 3 Clients
- Grid & Node: erzeugt ein 10x10-Gitter und demonstriert das x-y Routing

Alle Programme dazu sind auf der Vorlesungswebseite zu finden.

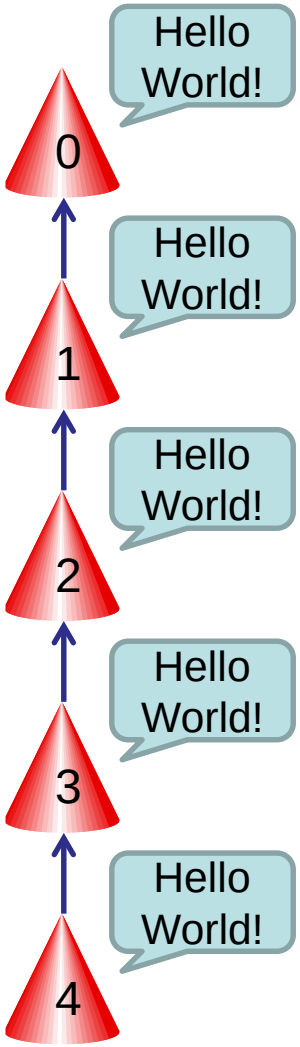
Hello World


```
public class Hello extends Subject {
      public Hello() {     }

      protected void init() {
            println("Hello World!");
      }

      protected void onMessageReceived(Object message) { }

      protected void onTimeout() { }

      public static void main(String[] args) {
            Hello hello = new Hello();
            hello.start();
      }
}
```

Chain Beispiel

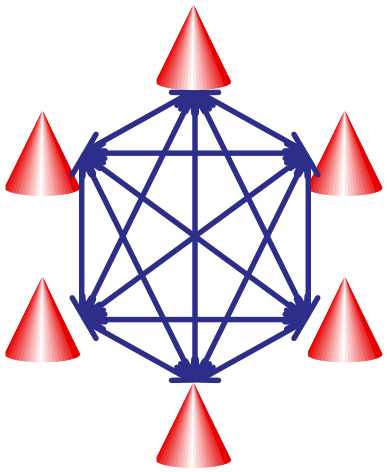

```
public class Chain extends Subject {
    private int n;
    private Chain parent;

    public Chain(int n, Chain parent) {
        // do not create new subjects in the constructor!
        // instead, use the init-method
        this.n = n;
        this.parent = parent;
    }

    protected void init() {
        if(n < 5) {
            // create and start child
            Chain child = new Chain(n + 1, this);
            child.start();
        } else {
            finish();
        }
    }
```

Chain Beispiel



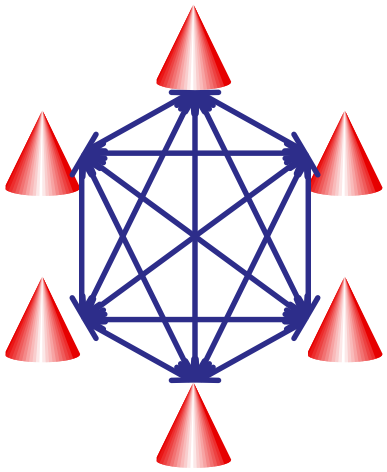

```
    protected void onMessageReceived(Object message) {
        if(message instanceof Boolean) {
            finish();
        }
    }

    protected void onTimeout() {    }

    // say hello, notify parent, and stop execution
    protected void finish() {
        println("Subject " + n + " says \"Hello World\"");
        if(parent != null) {
            // the sent boolean simply acts as a token
            parent.send(true);
        }
    }

    public static void main(String[] args) {
        Chain chain = new Chain(0, null);
        chain.start();
    }
}
```

Clique Beispiel



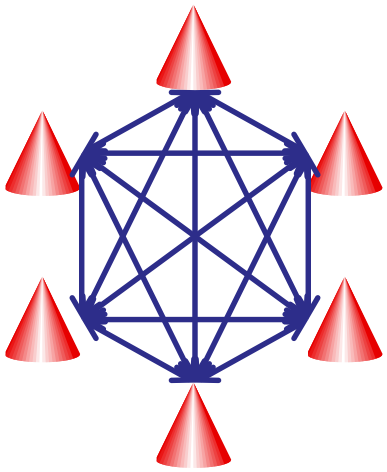
```
import java.util.LinkedList;

public class Clique extends Subject {
    private int id;
    private int cliqueSize;
    private LinkedList<Clique> subjects;

    public Clique(int id, int cliqueSize) {
        // do not create new subjects in constructor!
        // instead, use the init-method
        this.id = id;
        this.cliqueSize = cliqueSize;
        subjects = new LinkedList<Clique>();
        subjects.add(this);
    }

    protected void init() {    }
```

Clique Beispiel



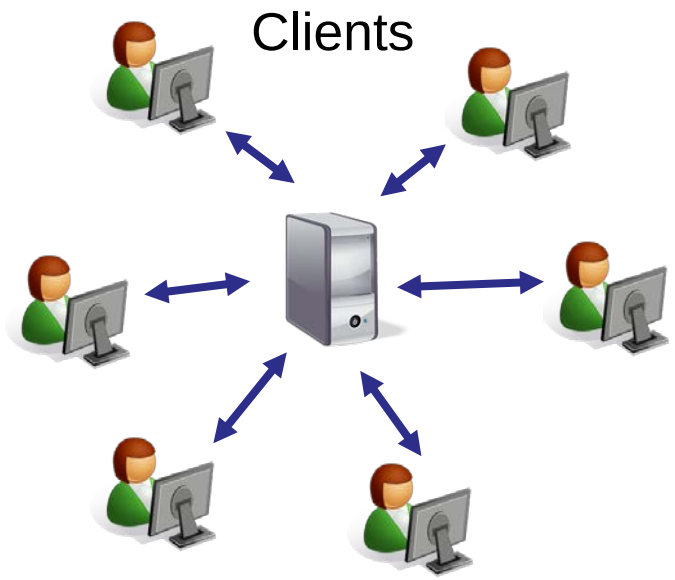
```
protected void onMessageReceived(Object message) {
    if(message instanceof Clique) {
        Clique other = (Clique) message;
        // if the other subject is already known, continue
        // this indirectly implements FUSION
        if(subjects.contains(other)) {
            return;
        }
        // implementation of INTRODUCTION
        for(int i = 0; i < subjects.size(); i++) {
            Subject knownSubject = subjects.get(i);
            // introduce other subject to all known subjects
            knownSubject.send(other);
             // and introduce all known subjects to other subject
            other.send(knownSubject);
        }
        subjects.add(other);
        if (subjects.size() == cliqueSize) {
            println("Subject " + id + " knows all other subjects.");
        }
    }
}

protected void onTimeout() {    }
```

Clique Beispiel



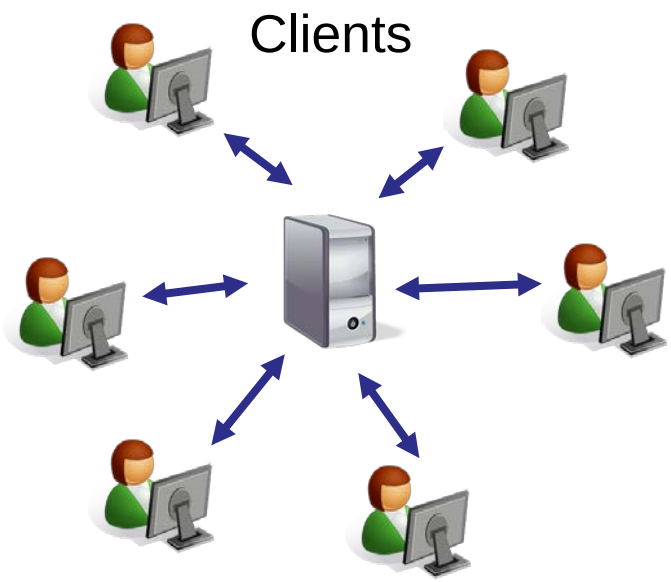
```
public static void main(String[] args) {
        int n = 10;
        Clique clique[] = new Clique[n];
        // create n subjects
        for(int i = 0; i < n; i++) {
            clique[i] = new Clique(i, n);
        }
        // connect them to a line
        for(int i = 1; i < n; i++) {
            clique[i].send(clique[i - 1]);
        }
        // start subjects
        for(int i = 0; i < n; i++) {
            clique[i].start();
        }
    }
}
```

Broadcast Beispiel



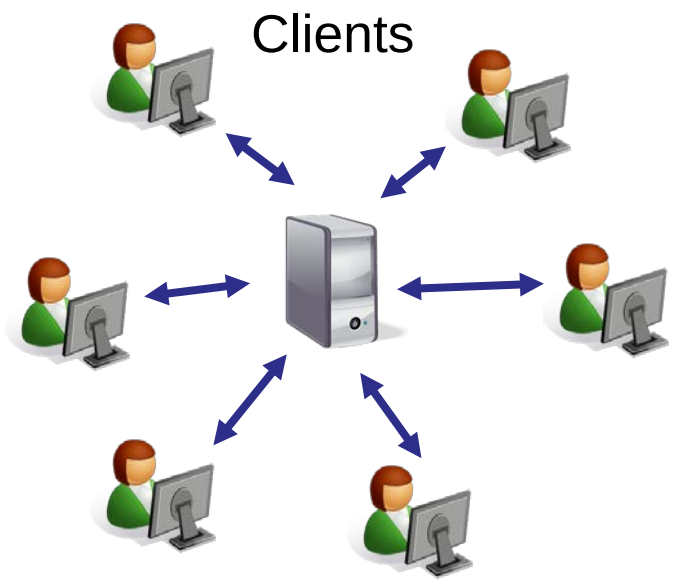

```
public class Client extends Subject {
	private String name;
	private Server server;

	public Client(String name, Server server) {
	    this.name = name;
	    this.server = server;
	}

	protected void init() {
	    // subscribe to server
	    server.send(this);
	}

	protected void onMessageReceived(Object message) {
	    if(message instanceof String) {
	        // message received from server, print it
	        String text = (String) message;
	        println(name + " received message \"" + text + "\"");
	    }
	}

	protected void onTimeout() {
	    // with probability 1/10, send a message to server
	    if(Math.random() < 0.1) {
	        String text = "Hello from " + name + ".";
	        println(name + " send message \"" + text + "\"");
	        server.send(text);
	    }
	}
}
```

Broadcast Beispiel



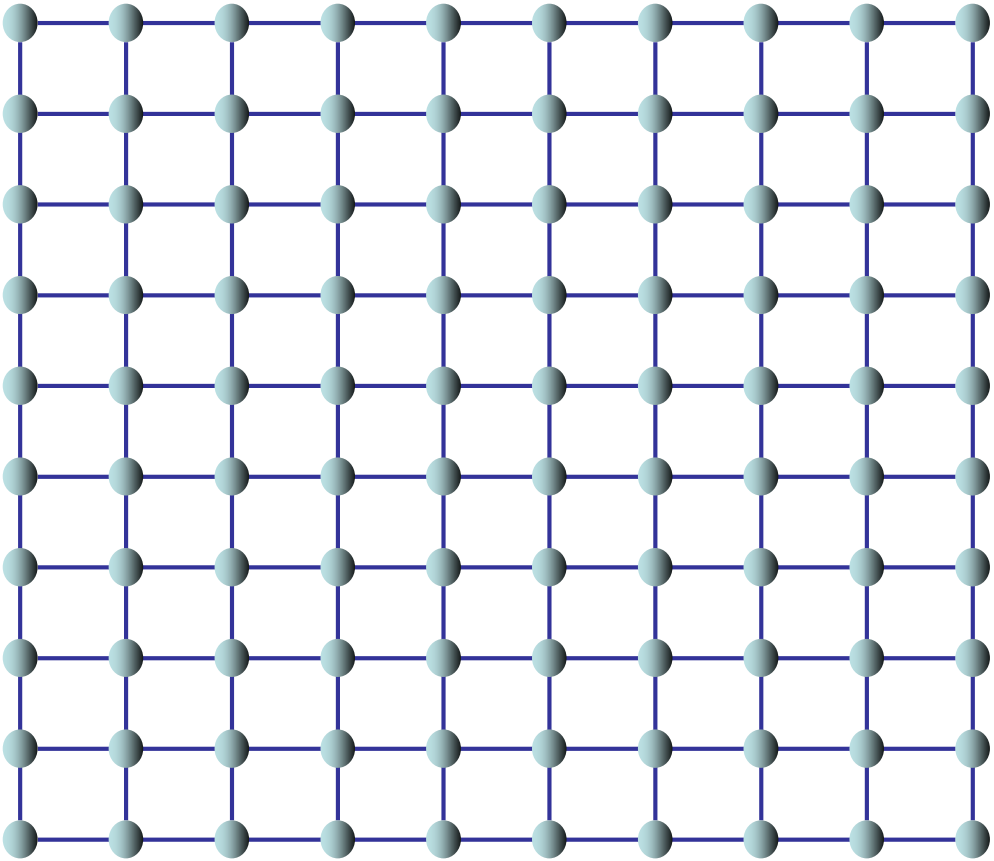

```
import java.util.LinkedList;

public class Server extends Subject {
    private LinkedList<Client> clients;

    public Server() {
        clients = new LinkedList<Client>();
    }

    protected void init() {   }

    protected void onMessageReceived(Object message) {
        if(message instanceof Client) {
            // client subscribes, add it to the list of clients
            clients.add((Client) message);
        }
        if(message instanceof String) {
            // message received, broadcast it to all clients
            for(int i = 0; i < clients.size(); i++) {
                clients.get(i).send(message);
            }
        }
    }
```

Broadcast Beispiel

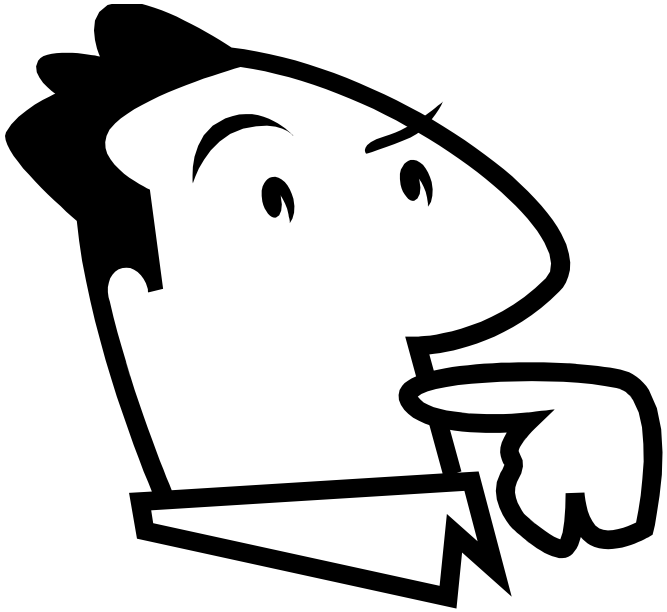



```
    protected void onTimeout() {   }

      public static void main(String[] args) {
          Server server = new Server();
          Client client1 = new Client("client 1", server);
          Client client2 = new Client("client 2", server);
          Client client3 = new Client("client 3", server);
          server.start();
          client1.start();
          client2.start();
          client3.start();
      }
}
```

Gitter Beispiel



Siehe Grid.java und Node.java auf der Vorlesungswebseite.



Fragen?