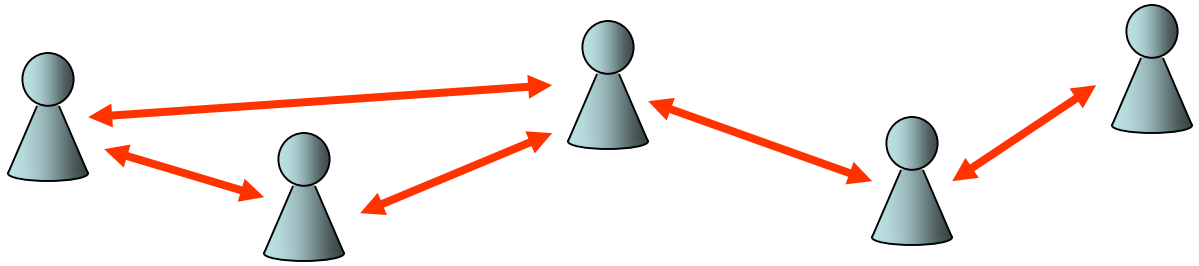
Verteilte Algorithmen und Datenstrukturen



Prof. Dr. Christian Scheideler

Institut für Informatik

Universität Paderborn

Verteilte Algorithmen und Datenstrukturen

Vorlesung: Mi 11:15 - 12:45 Uhr, F0.530
Übung: Mi 13:00 - 13:45 Uhr, F0.530

Webseite:
http://www-old.cs.upb.de/fachgebiete/fg-ti/lehre0/ss2016/verteilte-algorithmen-und-datenstrukturen.html

Prüfung:

- mündliche Prüfung (zwei Blöcke) 50%
- Programmierprojekt 50%
- Beide Teile müssen bestanden sein, um Kurs zu bestehen
- Notenverbesserung um 0,3: Präsentation in Übung

Voraussetzungen: Grundkenntnisse in Algorithmen und Datenstrukturen

Verteilte Algorithmen und Datenstrukturen

Übungsaufgaben:

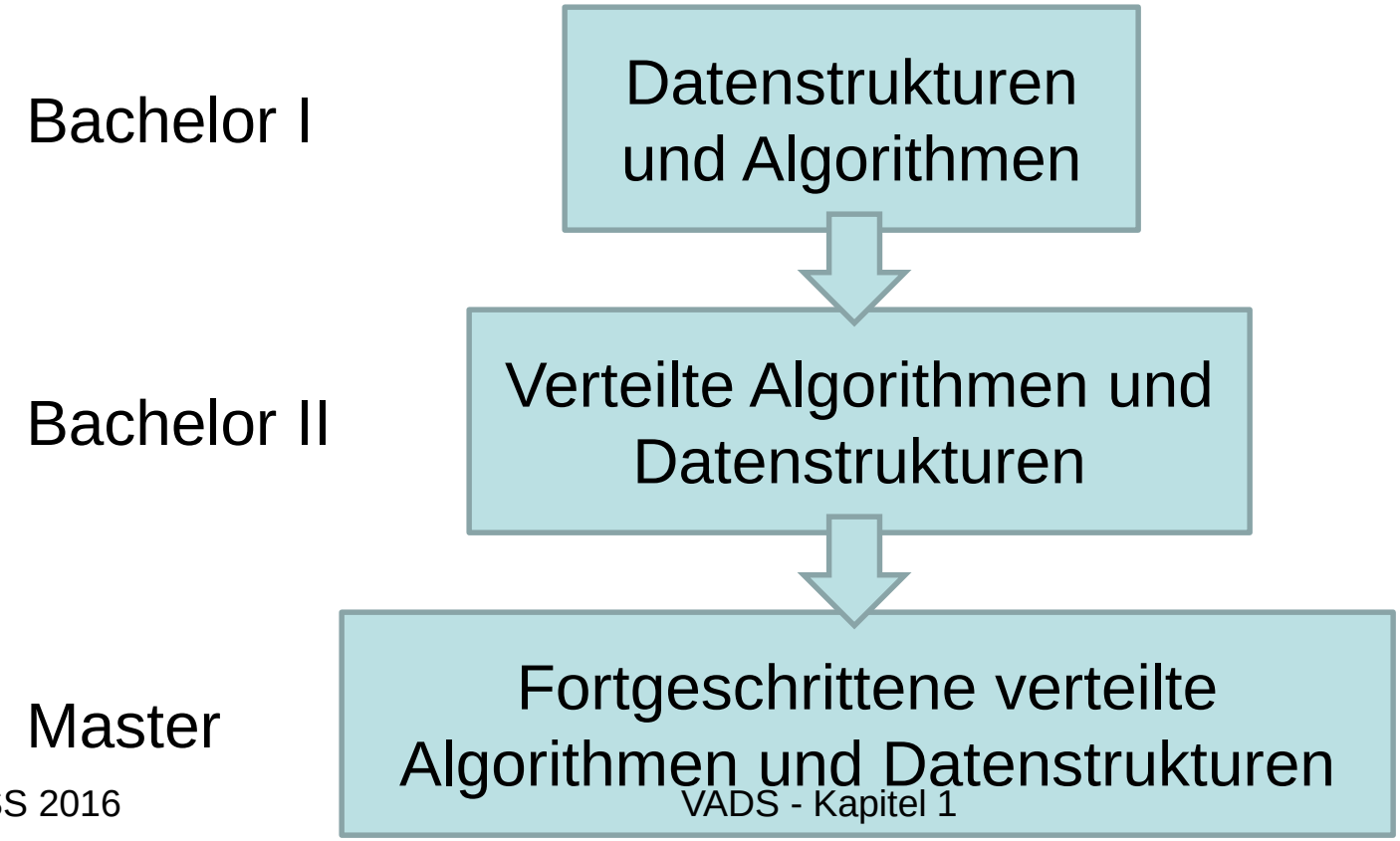
- wöchentliche Ausgabe und Abgabe am Mittwoch (ab nächste Woche)
- teils theoretisch, teils praktisch

Folien und Übungsaufgaben: Webseite

Bücherempfehlungen: kein Buch (Vorlesung basiert auf neuesten Ergebnissen)

Verteilte Algorithmen und Datenstrukturen

Einordnung in Informatikstudium:

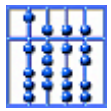


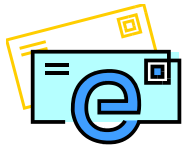
Verteilte Algorithmen und Datenstrukturen

Ziele:

1. Grundlegende verteilte Algorithmen und Datenstrukturen kennen zu lernen.
2. Wichtige Techniken und Entwurfsmethoden kennen und *anwenden* zu lernen.
3. Wichtige Analysemethoden kennen und *anwenden* zu lernen.

Einleitung





Sequentielle Algorithmen und Datenstrukturen

Verteilte Algorithmen und Datenstrukturen

Einleitung

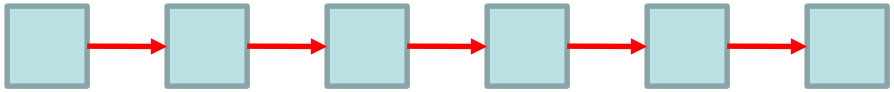
Was sind die Kernprobleme für verteilte Algorithmen und Datenstrukturen?

Einleitung

Definition 1.1: Eine *Datenstruktur* ist eine bestimmte Art, Daten im Speicher eines Computers so anzuordnen, dass Operationen wie z.B. *Suchen, Einfügen, Löschen* einfach zu realisieren sind.

Einfache Beispiele:

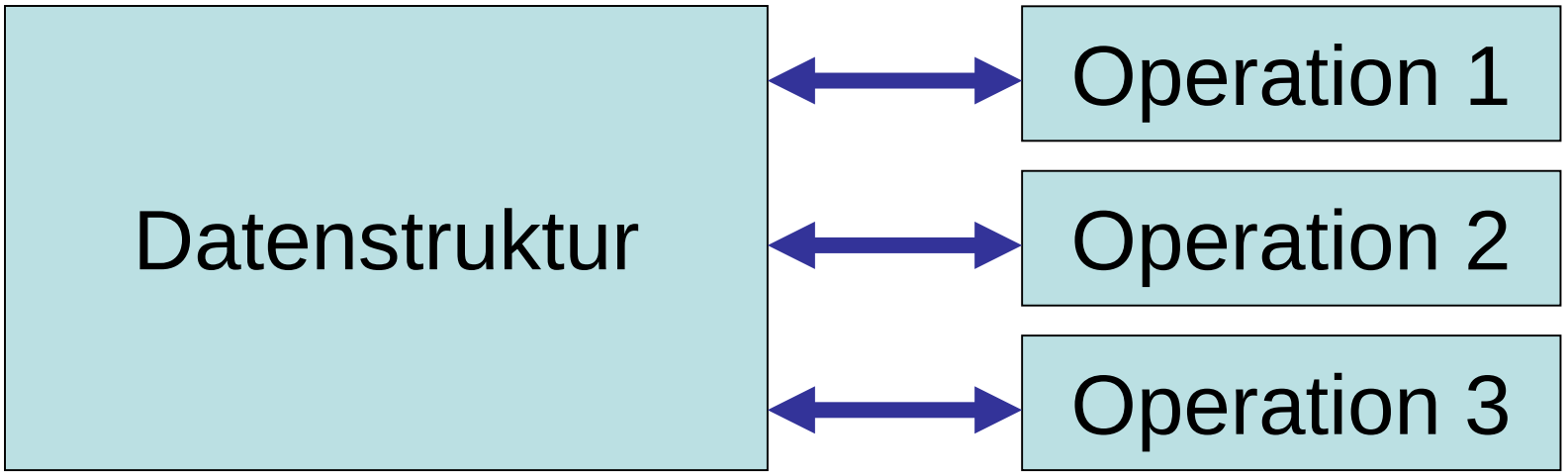
- Listen



- Arrays

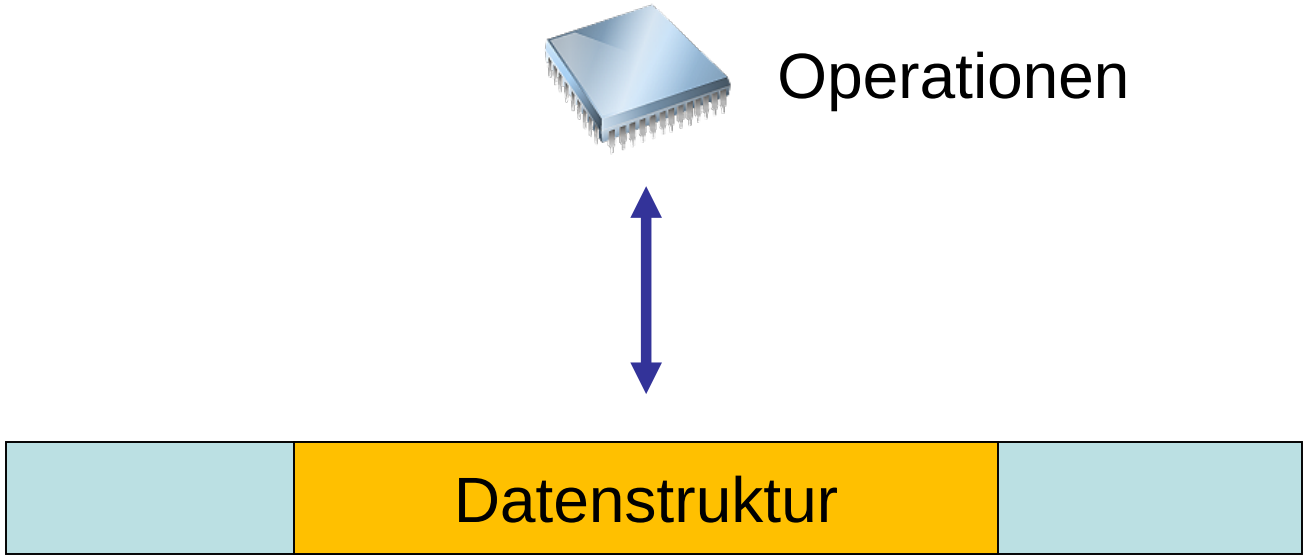
Einleitung

Anschaulich:



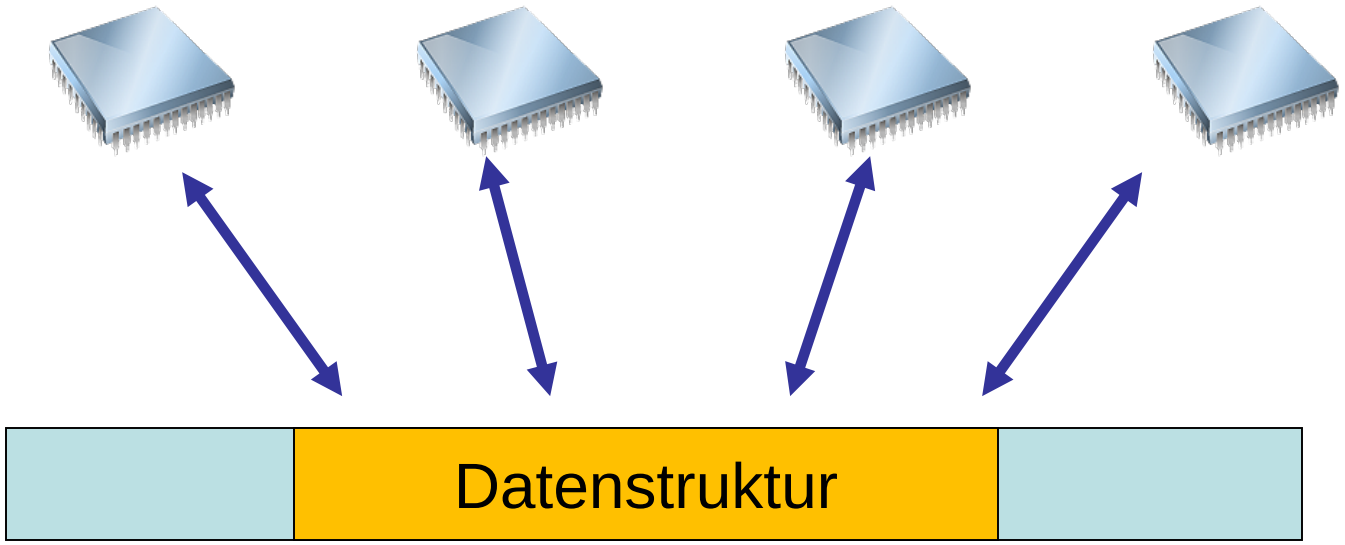
Einleitung

Klassischer Fall: 1-Prozessor Rechner



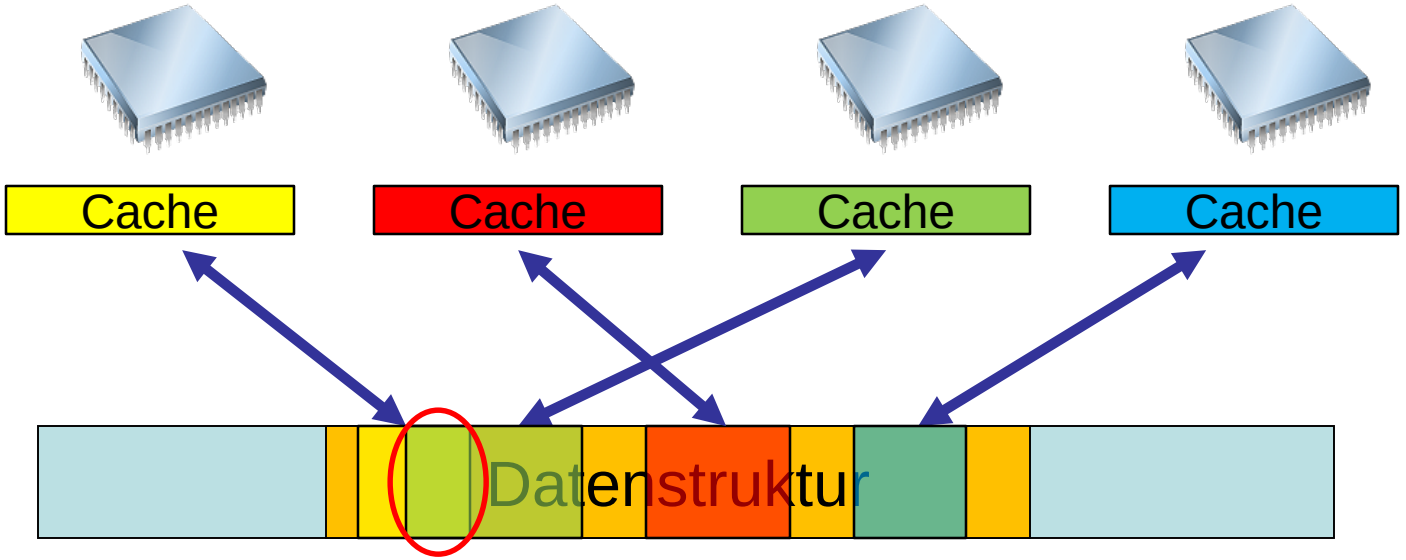
Einleitung

Rechner mit mehreren Prozessoren/Kernen:



Einleitung

Rechner mit mehreren Prozessoren/Kernen:

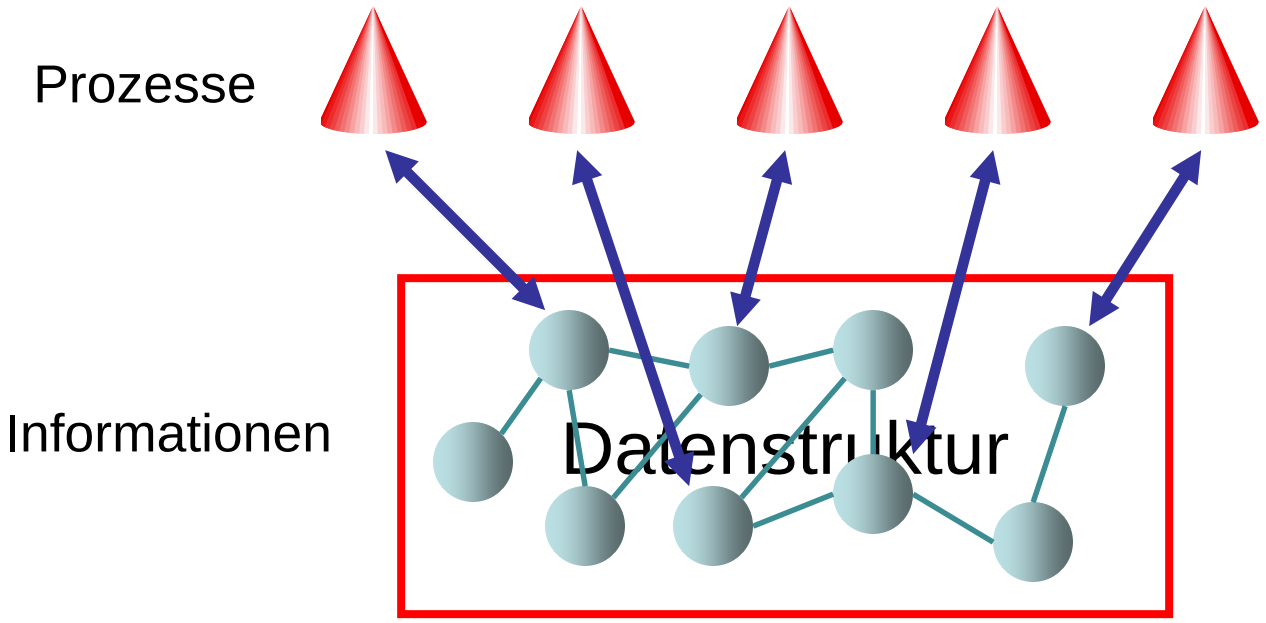


Überlappung:

- Zugriffskonflikte (Korrektheit)
- Leistungsprobleme (Effizienz)

Einleitung

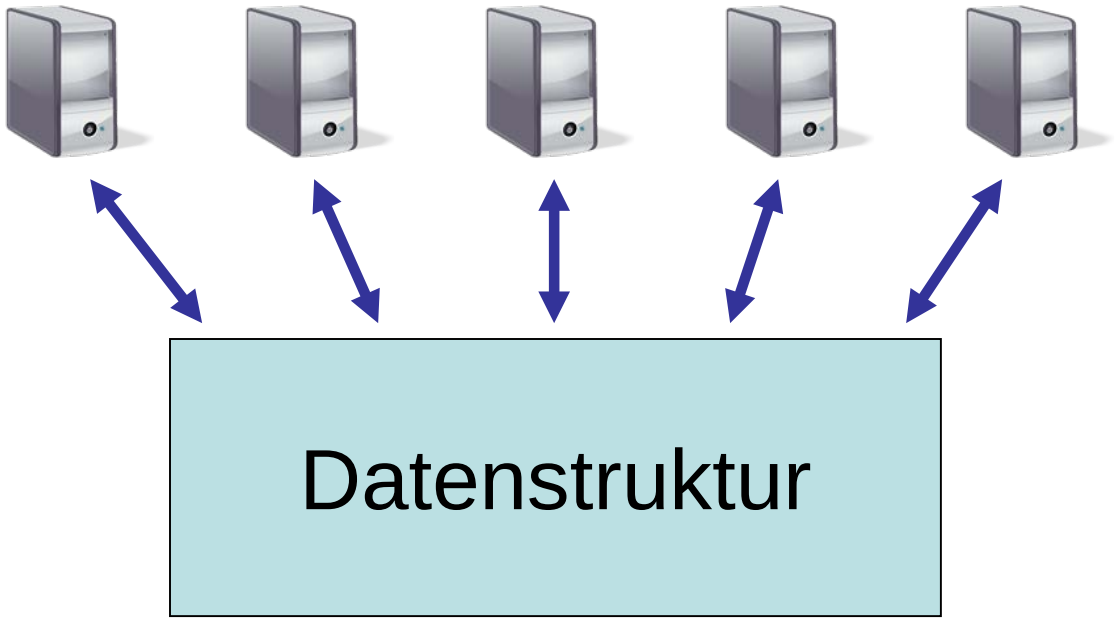
Abstrakte Sichtweise:



Externe Datenstruktur

Einleitung

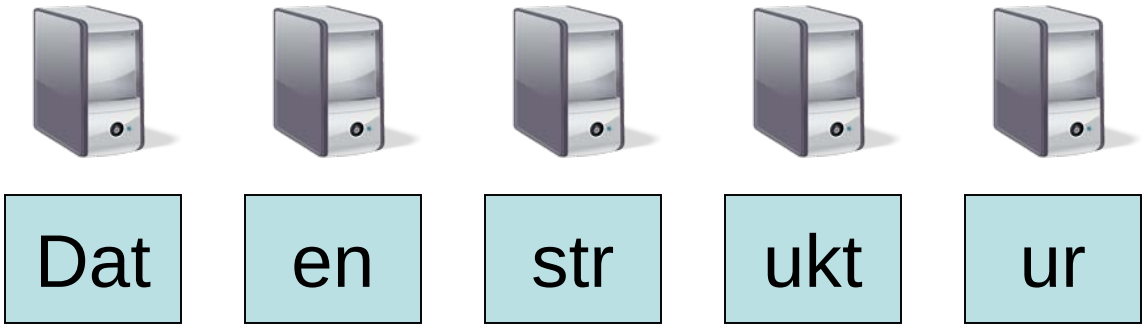
Mehrere Rechner:



Problem: Verteilung der Datenstruktur auf Rechner

Einleitung

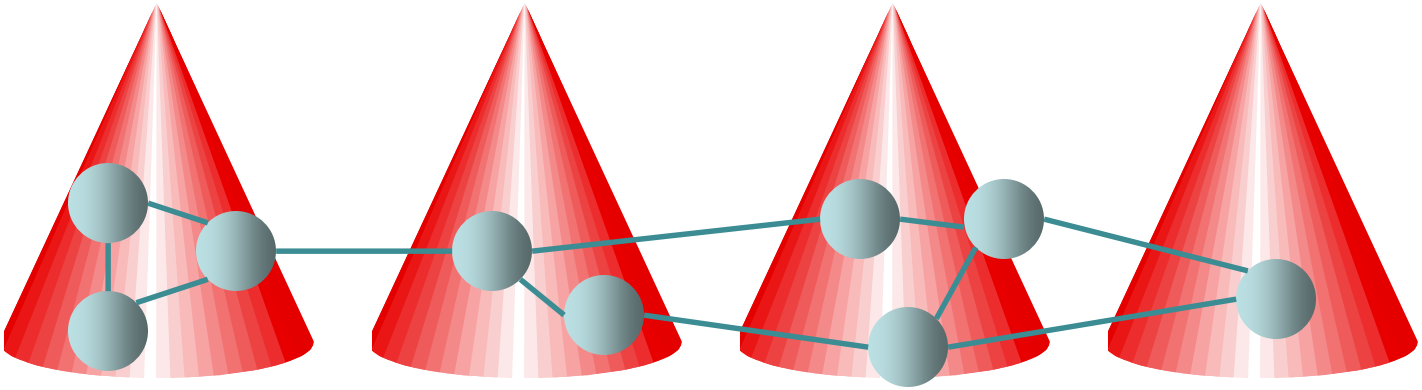
Mehrere Rechner:



Problem: Verteilung der Datenstruktur auf Rechner

Einleitung

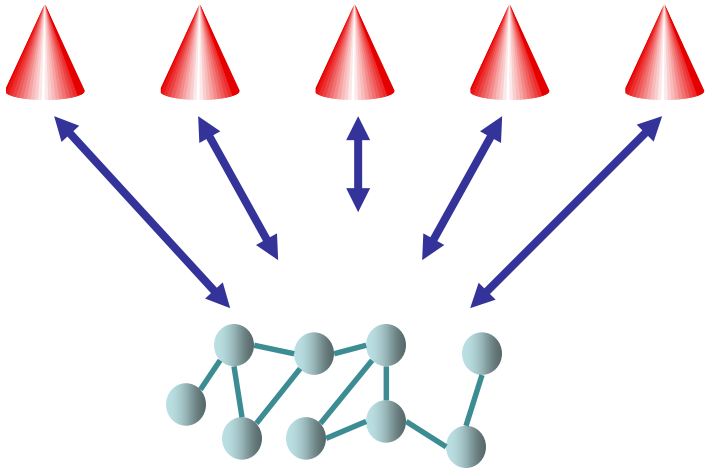
Abstrakte Sichtweise:



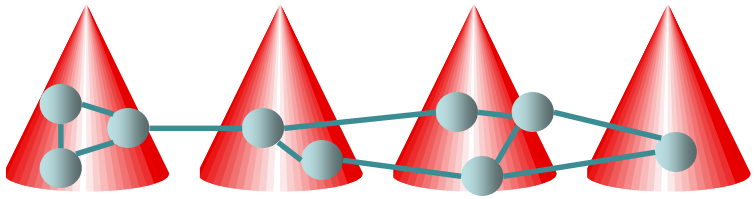
Interne Datenstruktur

Verteilte Datenstrukturen
Einheitliche Sicht

Externe Datenstruktur



Interne Datenstruktur



Zentrale Fragen:

- Wie genau kann auf die Daten zugegriffen werden?
- Wie können Prozesse miteinander kommunizieren?

Datenzugriff

Zwei Alternativen:

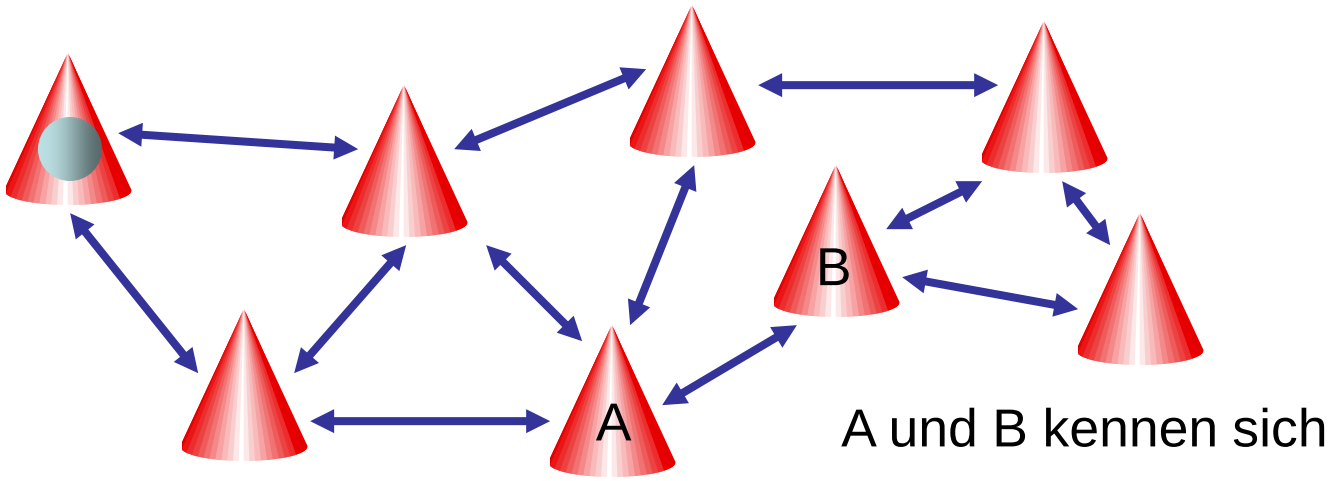
- Anforderung des Datums (oder einer Kopie)

- Antwort auf Anfrage zu einem Datum

Beste Strategie problemspezifisch.

Kommunikation zwischen Prozessen

Problem: finde geeignetes Kommunikations-netzwerk für die Prozesse

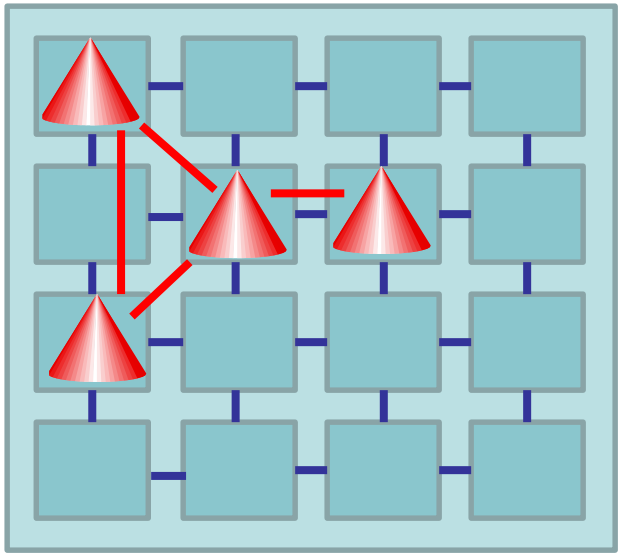


Bestes Netzwerk problemspezifisch.

Kommunikation zwischen Prozessen

Problem: finde geeignetes Kommunikations-netzwerk für die Prozesse

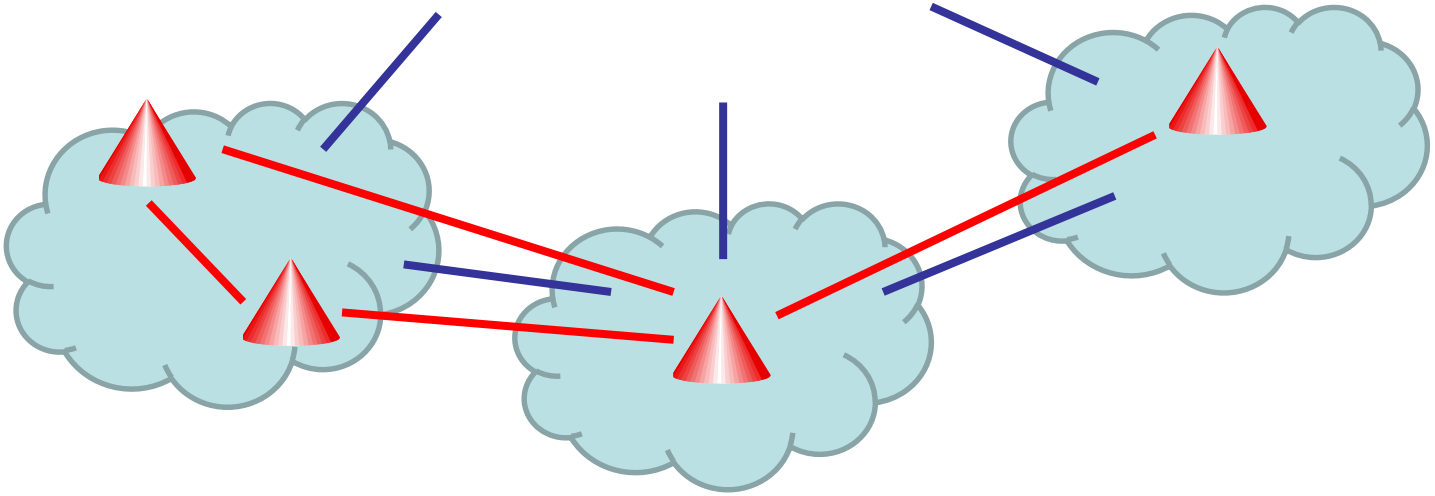
Beispiel: Prozesse innerhalb eines Multicore-Chips



Kommunikation zwischen Prozessen

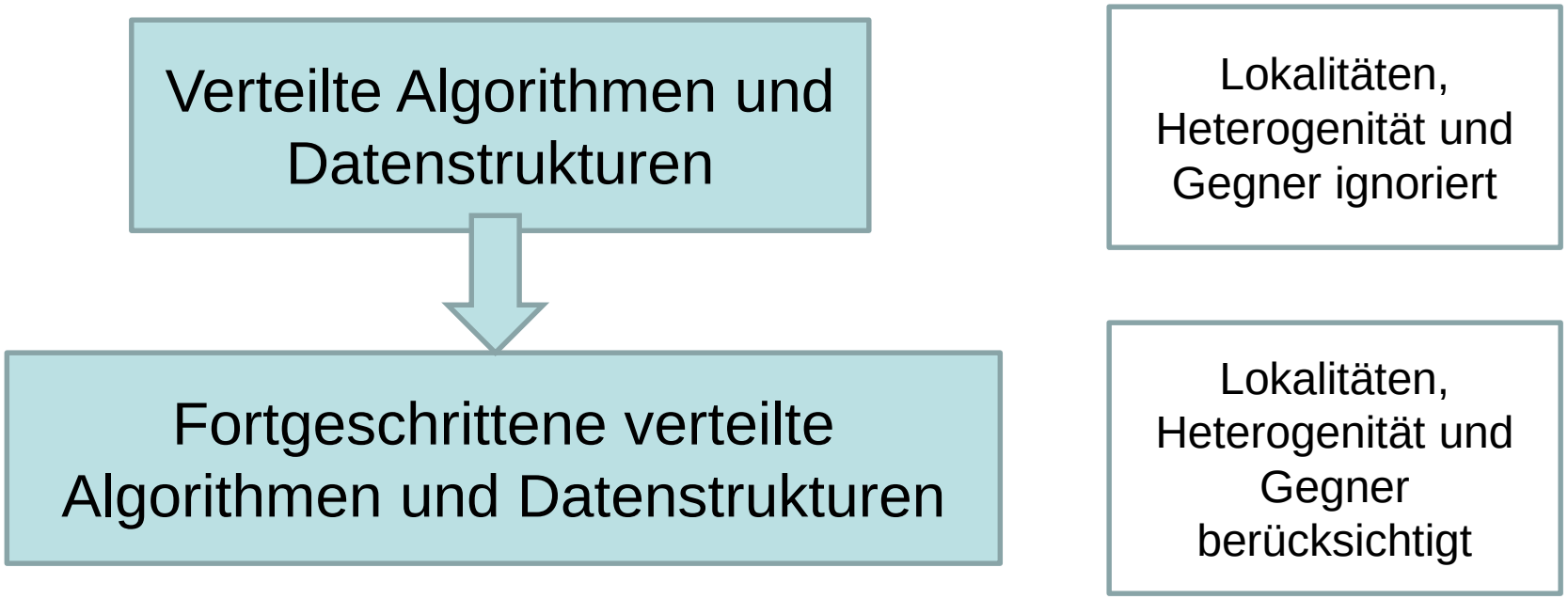
Problem: finde geeignetes Kommunikations-netzwerk für die Prozesse

Beispiel: Prozesse im Internet



Kommunikation zwischen Prozessen

Problem: finde geeignetes Kommunikations-netzwerk für die Prozesse



Lokalität und Heterogenität

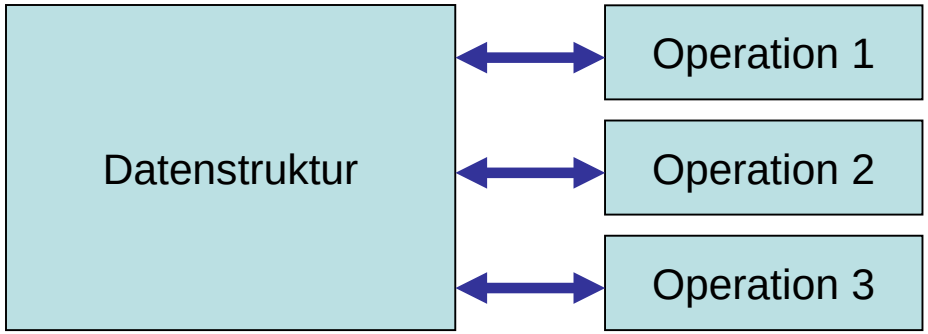
Beispiel: Hybride Netze

- long range (LTE, Satellit)
- mid range (WLAN)
- short range (Bluetooth, NFC)
- …

Anwendungen:

Einleitung

Abstrakte Sicht auf Datenstruktur:



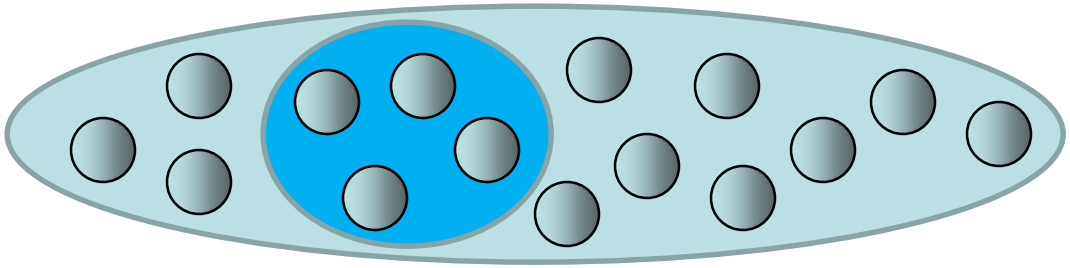
Zentrale Frage: mit welchen Zugriffsmethoden und welchem Kommunikationsnetzwerk kann die Datenstruktur am effizientesten umgesetzt werden?

Geeignetes Maß notwendig!

Einleitung

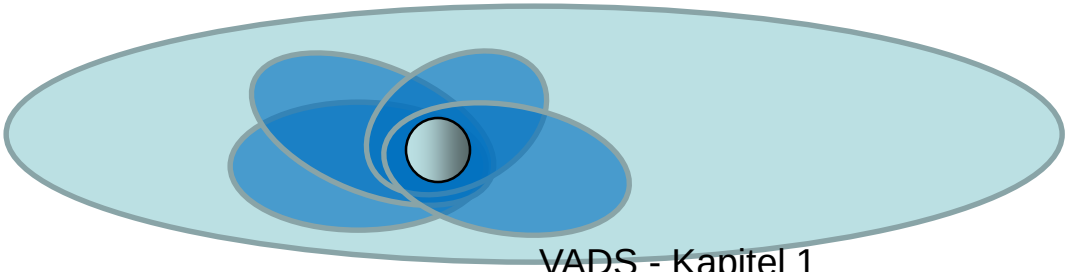
Effizienz: externe Datenstruktur

- Jede Operation involviert nur kleine Menge der Daten



geringer Ressourcenbedarf

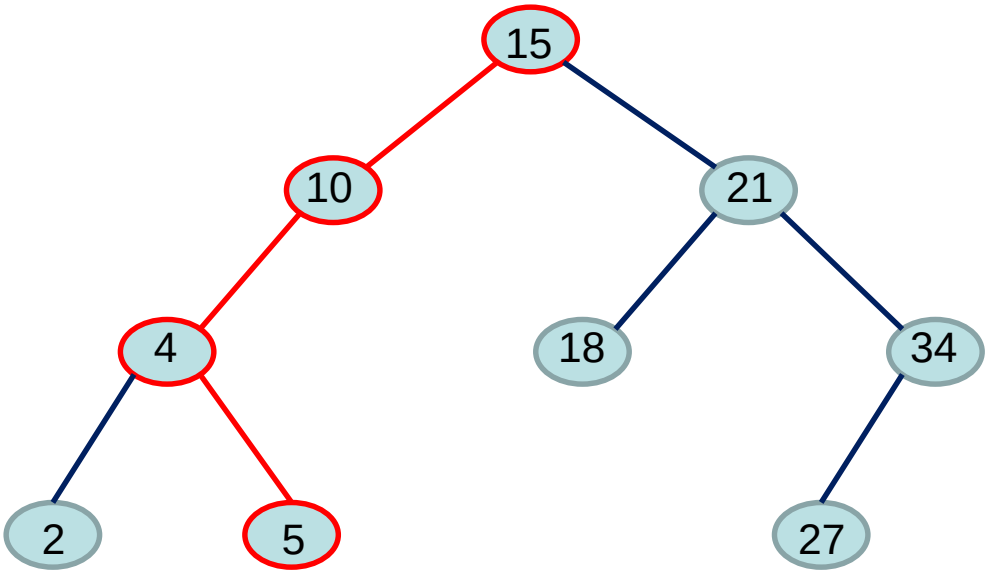
- Für eine beliebige Menge von Operationen mit maximal einer Initiierung pro Prozess ist die maximale Anzahl an Operationen, in die ein Datum involviert ist, gering.



hoher Durchsatz

Einleitung

Sequentieller Fall: Suchbaum, Search(5)

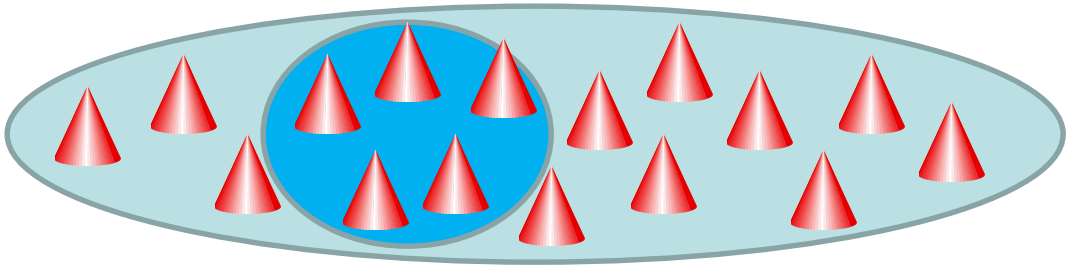


- Search involviert nur wenige Daten, falls Suchbaum balanciert
- Wegen Start bei Wurzel haben Search Operationen aber hohe Überlappung

Einleitung

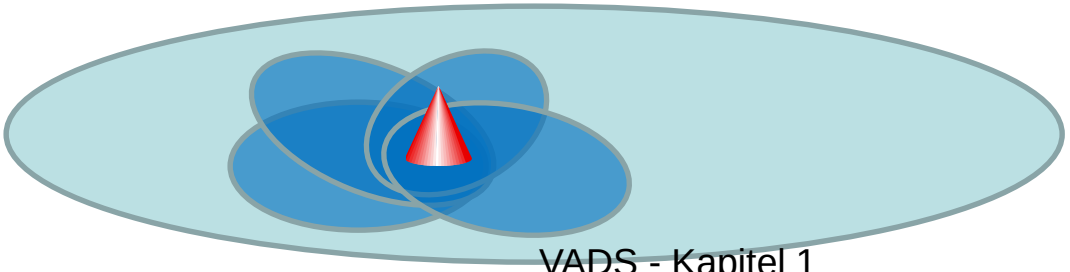
Effizienz: interne Datenstruktur

- Jede Operation involviert nur kleine Menge der Prozesse



geringer Ressourcenbedarf

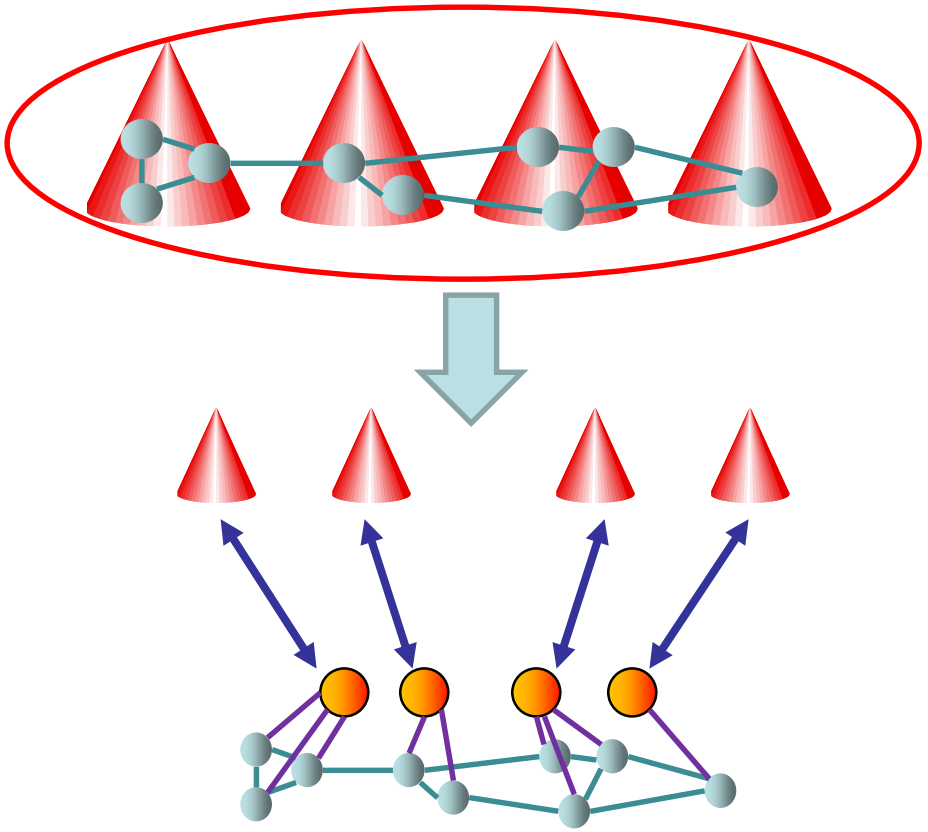
- Für eine beliebige Menge von Operationen mit maximal einer Initiierung pro Prozess ist die maximale Anzahl an Operationen, in die ein Prozess involviert ist, gering.



hoher Durchsatz

Einleitung

Transformation interne → externe Datenstruktur

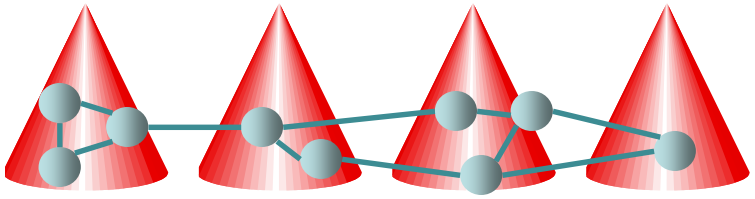


Fokus dieser Vorlesung

Einleitung

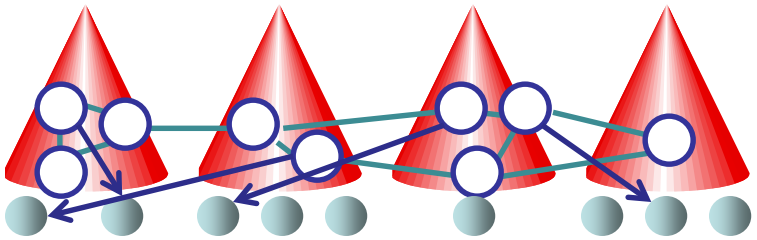
Szenarien für interne Datenstrukturen:

- Verwaltung von Daten (falls z.B. Daten beweglich und klein)
  → direkte Datenstruktur



SANs oder
RAID Systeme

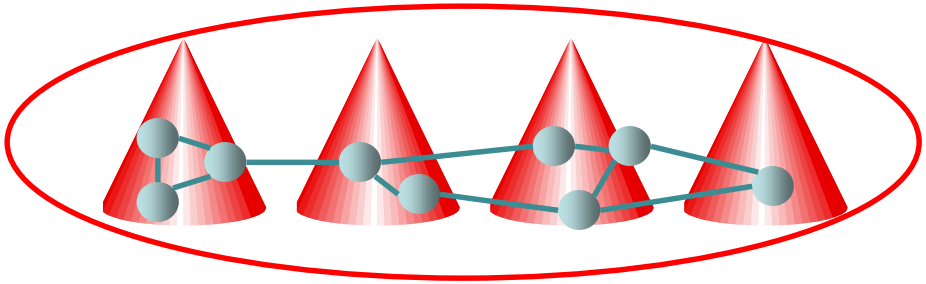
- Verwaltung von Referenzen auf Daten (falls z.B. Daten verankert oder groß)
  → indirekte Datenstruktur



Standard für
P2P Systeme

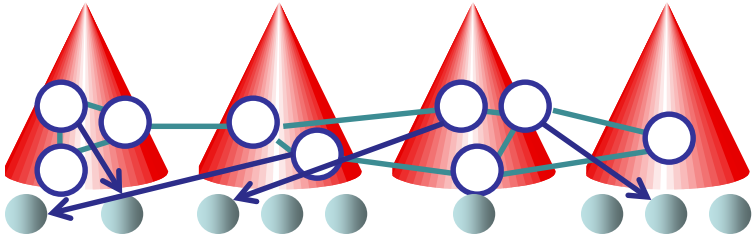
Einleitung

Transformation direkte $\rightarrow$ indirekte Datenstruktur



Fokus dieser Vorlesung

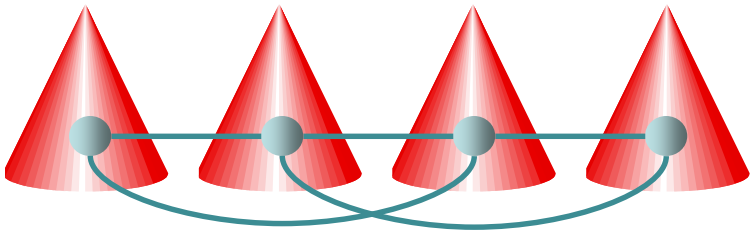
eine Indirektionsstufe



Einleitung

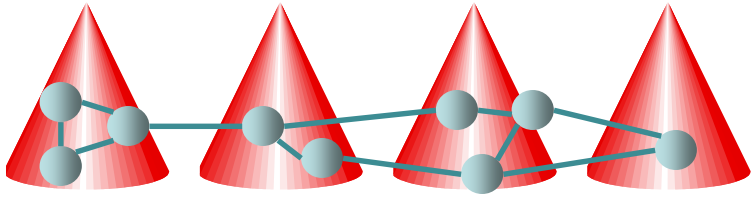
Szenarien für direkte Datenstrukturen:

- verankerte Daten → prozessorientierte Datenstruktur
  (nur Vernetzung ändert sich)



Beispiel:
Suchstruktur für
Prozesse (→DNS)

- bewegliche Daten → informationsorientierte Datenstruktur
  (Datenpositionen und Vernetzungen ändern sich)



Beispiel:
Suchstruktur für
Daten (→Google)

Einleitung

Ziel der Vorlesung: Prozess/Informations-orientierte verteilte Datenstrukturen für elementare Datentypen

- (zyklische) Listen
- Queues und Stacks
- Hashing
- Suchstrukturen
- Priority Queues

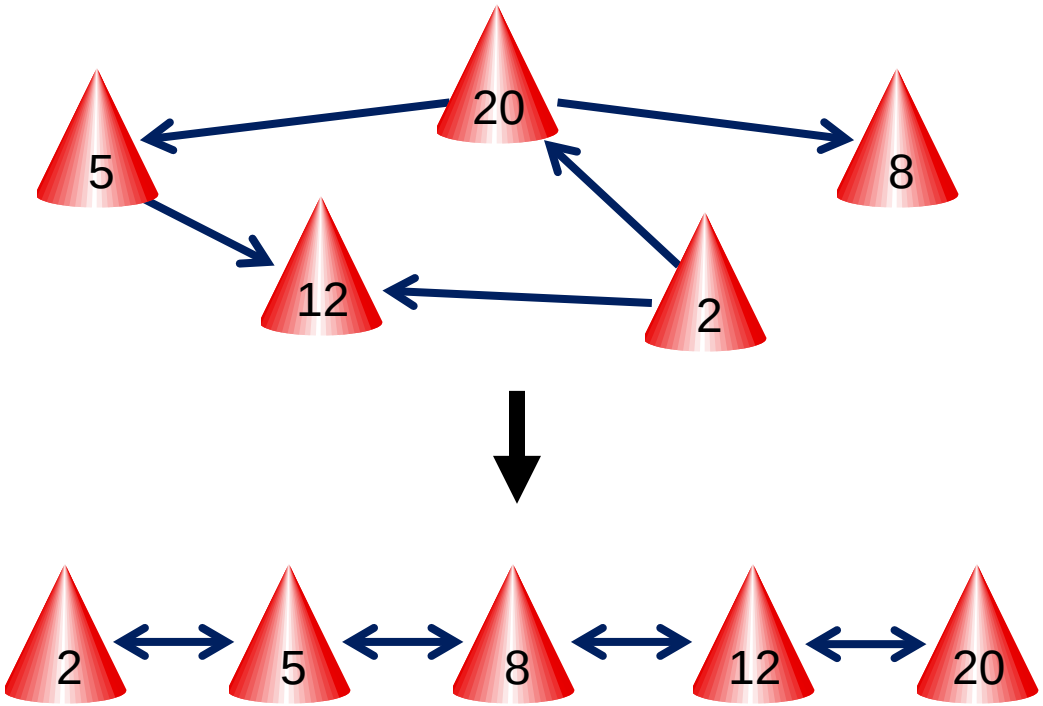
Beispiele in DuA:

- Linear probing
- AVL-Bäume
- binäre Heaps

Einleitung

Liste: Beispiel für prozessorientierte Datenstruktur

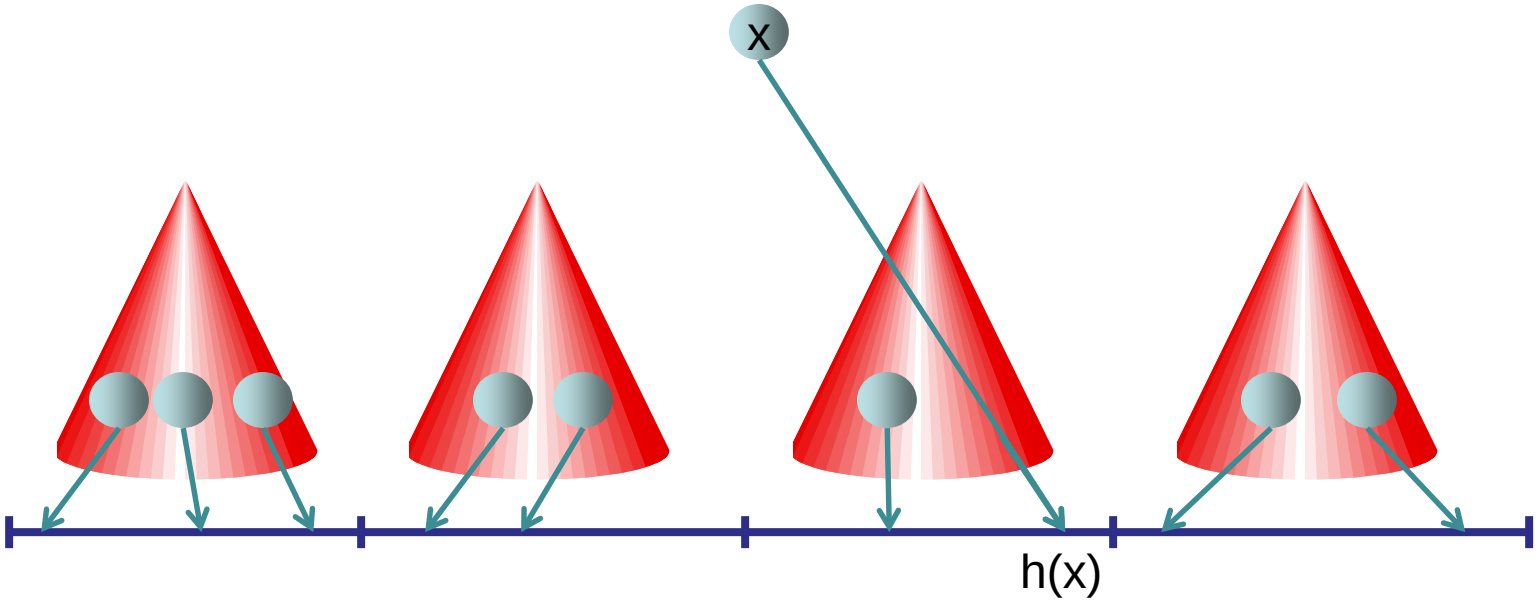
Konkrete Operation: BuildList: organisiere Prozesse in sortierter Liste gemäß ihrer Namen



Einleitung

Hashing: Beispiel für informationsorientierte Datenstruktur

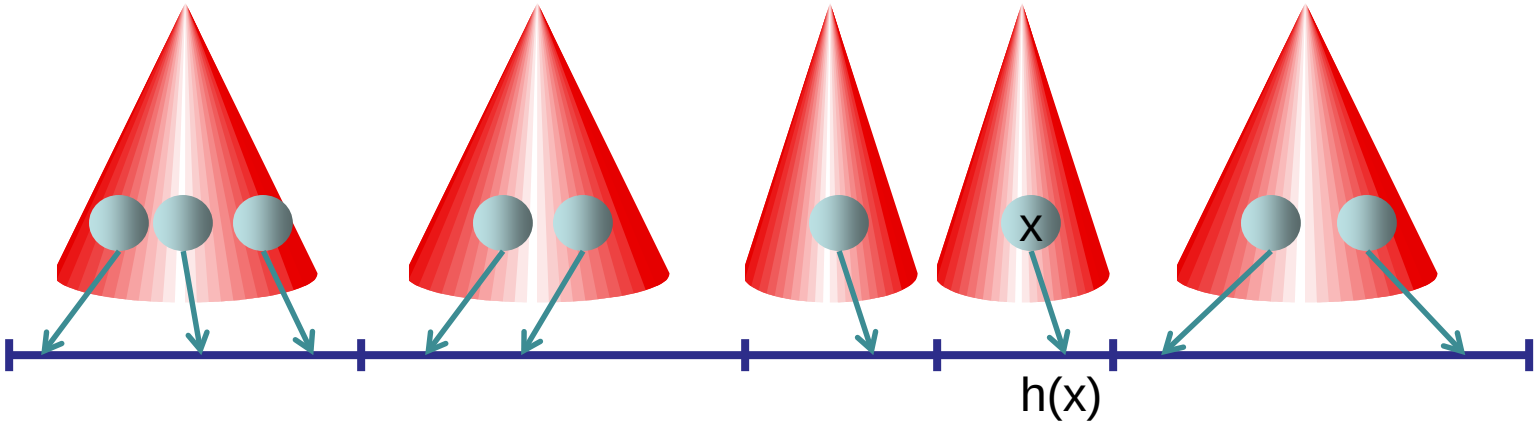
Konkrete Operation: Insert(x): füge Element x in Hash-tabelle ein



Einleitung

Hashing: Beispiel für informationsorientierte Datenstruktur

Konkrete Operation: Insert(x): füge Element x in Hash-tabelle ein



Hashtabelle

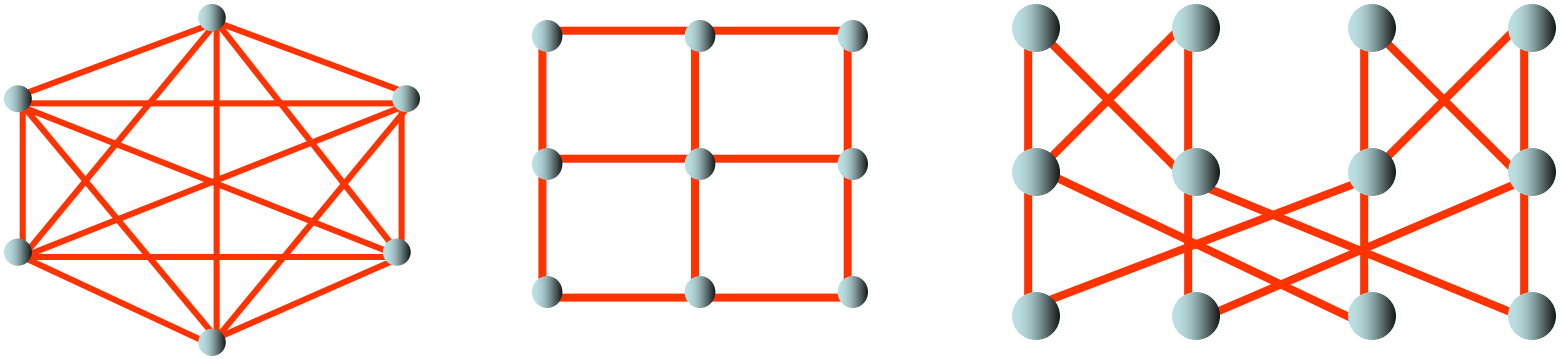
Verteilte Algorithmen und Datenstrukturen

Inhalt:

1. Einführung
2. Netzwerktheorie
3. Designprinzipien für verteilte Algorithmen und Datenstrukturen
4. Einführung in die verteilte Programmierung
5. Prozessorientierte Datenstrukturen
   a. (zyklische) Listen
   b. Cliquen
   c. De Bruijn Graphen (prozessorientiertes Hashing)
   d. Skipgraphen (prozessorientierte Suche)
6. Informationsorientierte Datenstrukturen
   a. Verteiltes Hashing
   b. Verteilter Stack
   c. Verteilte Queue
   d. Verteilter Heap

Netzwerktheorie

- Einführung in grundlegende Topologien



- Grundlegende Graphparameter (Grad, Durchmesser, Expansion,…)
- Routing (finde Weg von A nach B)

Verteilte Algorithmen und Datenstrukturen

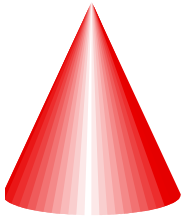
Inhalt:

1. Einführung
2. Netzwerktheorie
3. Designprinzipien für verteilte Algorithmen und Datenstrukturen
4. Einführung in die verteilte Programmierung
5. Prozessorientierte Datenstrukturen
   a. (zyklische) Listen
   b. Cliquen
   c. De Bruijn Graphen (prozessorientiertes Hashing)
   d. Skipgraphen (prozessorientierte Suche)
6. Informationsorientierte Datenstrukturen
   a. Verteiltes Hashing
   b. Verteilter Stack
   c. Verteilte Queue
   d. Verteilter Heap

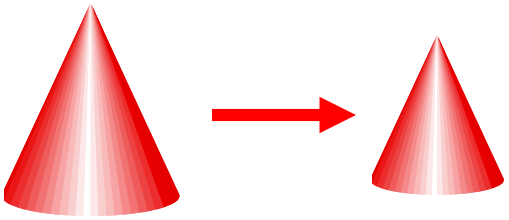
Designprinzipien für verteilte Algorithmen und Datenstrukturen

- Prozess- und Netzwerkmodell
- Pseudocode
- Primitive zur Manipulation von Verbindungen zwischen Prozessen
- Selbststabilisierung
- Konsistenzmodelle

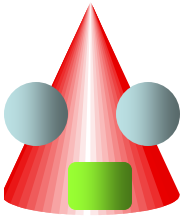
Prozessmodell

- Prozesse 
- Objekte  (repräsentieren Daten)
- Aktionen 

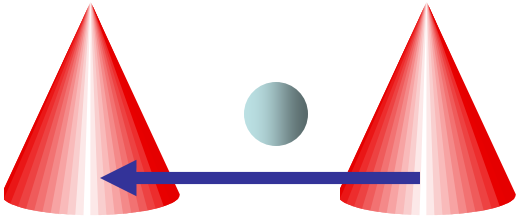
Prozess



kann Prozess (am selben Ort wie Mutterprozess) erzeugen



kann Objekte besitzen, führt Aktionen sequentiell aus



kann Verbindungen aufbauen

Pseudocode

Ähnlich zu objektorientierter Programmierung:

Subject <Subjektname>: { deklariert Prozesstyp }
    lokale Variablen
    Aktionen

Aktionstypen:

⟨Aktionsname⟩(Objektliste) →
    Befehle in Pseudocode

⟨Aktionsname⟩: ⟨Prädikat⟩ →
    Befehle in Pseudocode

Spezielle Aktionen:

init(Objektliste) → // Konstruktur
    Befehle in Pseudocode

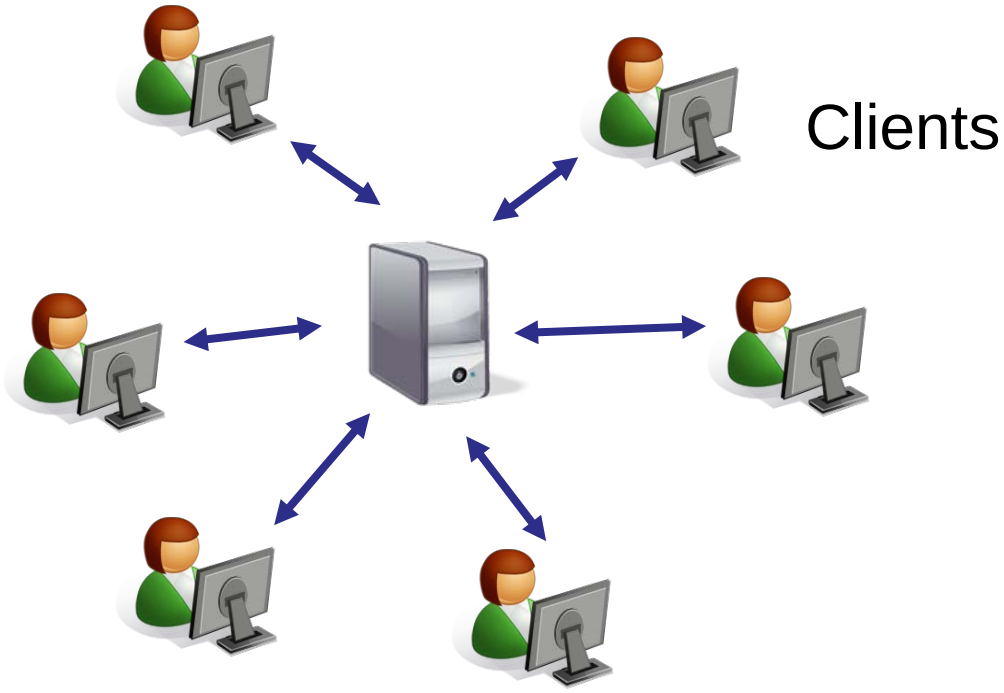
timeout → // bei jedem timeout (was periodisch erfolgt)
    Befehle in Pseudocode

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: ←
- Erzeugung neuer Subjekte und Objekte: new
  (new aktiviert init im Subjekt)

Beispiel

Einfacher Broadcast Service über Server



Broadcast Service

```
Subject Server:
    n: Integer              { aktuelle Anzahl Clients }
    Client: Array[1..MAX] of Subject    { für Subjektreferenzen}

    init() →                { Konstruktor }
       n:=0

    register(C) →           { registriere neuen Client mit Referenz C }
       n:=n+1
       Client[n]:=C

    broadcast(M) → { schicke M an alle Clients }
       for i:=1 to n do
          M´:=new Object(M)    {neues Objekt mit Inhalt von M}
          Client[i]←output(M´)
```

Broadcast Service

```
Subject Client:
   Server: Subject

   init(S) →                    { Konstruktor }
      Server:=S                 { S: Referenz auf Server }
      Server←register(this)
                                { eigene Ref. an Server }

   broadcast(M) →               { verteile M über Server }
      Server←broadcast(M)

   output(M) →                  { gib M aus }
      print M
```

Beispiel

Konventioneller Mergesort Algorithmus:

```
Mergesort(A, l, r):
 if l<r then
     m:= (l+r)/2
     Mergesort(A, l, m)      {sortiere linke Teilfolge}
     Mergesort(A, m+1, r)    {sortiere rechte Teilfolge}
     Merge(A, l, r)          {Merge-Schritt}
```

Aufruf zu Beginn mit Mergesort(*A*,1,*length*(*A*)) für ein Zahlenarray A[1..length(A)].

Mergesort

Verteilter Mergesort Algorithmus:

```
Subject Mergesort:
    caller: Subject
    replies: Integer
    A: Array[1..n] of Integer
    l, r: Integer

    init(A´, l´, r´, C) →
    A:=A´; l:=l´; r:=r´; caller:=C
    replies:=0
    if l<r then
        m:= (l+r)/2
        new Mergesort(new Object(A), l, m, this)        { ein Kind sortiert die linke Hälfte }
        new Mergesort(new Object(A), m+1, r, this)      { das andere Kind die rechte Hälfte }
    else caller←done(A,l,r)

    done(A´,l´,r´) →
      for i:=l´ to r´ do A[i]:=A´[i]
      if replies=0 then replies:=1      { schon beide Antworten erhalten? }
                   else merge()

    merge() →
        … { wie normales Merge }
        caller←done(A,l,r)
```

Verteilte Algorithmen und Datenstrukturen

Inhalt:

1. Einführung
2. Netzwerktheorie
3. Designprinzipien für verteilte Algorithmen und Datenstrukturen
4. Einführung in die verteilte Programmierung
5. Prozessorientierte Datenstrukturen
   a. (zyklische) Listen
   b. Cliquen
   c. De Bruijn Graphen (prozessorientiertes Hashing)
   d. Skipgraphen (prozessorientierte Suche)
6. Informationsorientierte Datenstrukturen
   a. Verteiltes Hashing
   b. Verteilter Stack
   c. Verteilte Queue
   d. Verteilter Heap

Beispiel in Java

```
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();
    }
}
```

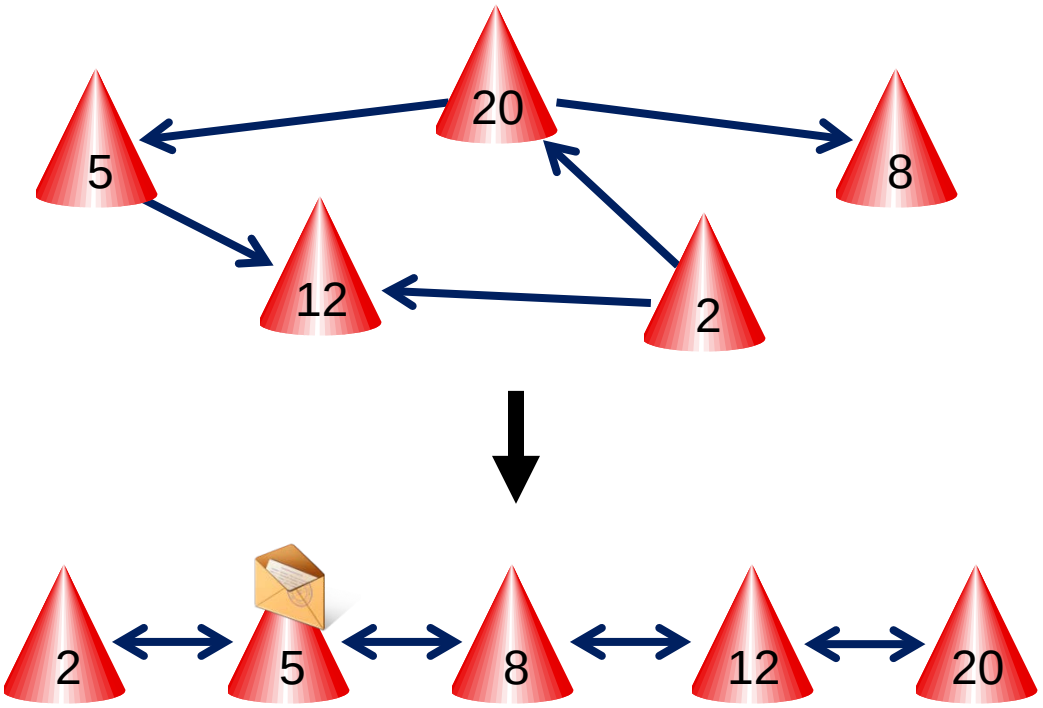
Verteilte Algorithmen und Datenstrukturen

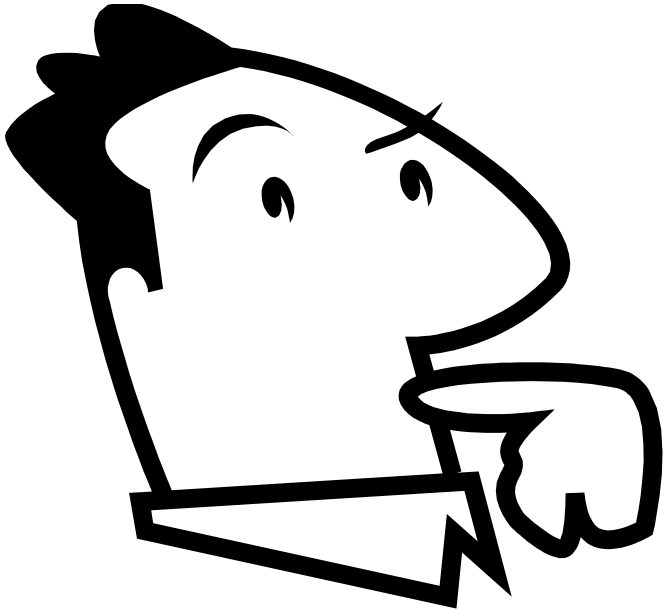
Inhalt:

1. Einführung
2. Netzwerktheorie
3. Designprinzipien für verteilte Algorithmen und Datenstrukturen
4. Einführung in die verteilte Programmierung
5. Prozessorientierte Datenstrukturen
   a. (zyklische) Listen
   b. Cliquen
   c. De Bruijn Graphen (prozessorientiertes Hashing)
   d. Skipgraphen (prozessorientierte Suche)
6. Informationsorientierte Datenstrukturen
   a. Verteiltes Hashing
   b. Verteilter Stack
   c. Verteilte Queue
   d. Verteilter Heap

Prozessorientierte Datenstrukturen

Beipiel: sortierte Liste





Fragen?