

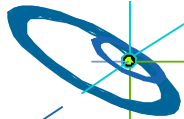
Die Pointer-Datenstruktur

(am Beispiel von FORTRAN und CFDFV)

M. Haas, A. Stock, A. Beck

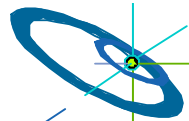
Universität Stuttgart
Institut für Aerodynamik und Gasdynamik (IAG)

www.iag.uni-stuttgart.de



Die Pointer-Datenstruktur

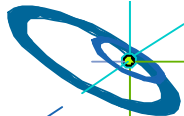
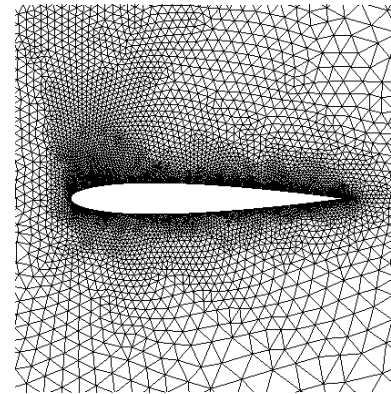
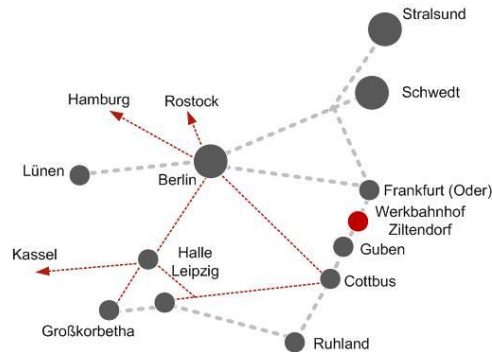
- Motivation
- Definition & Implementierung
- Pointerstrukturen in CFDFV
- Zusammenfassung / Literatur



Datentypen und Abstraktion

➤ Wie lassen sich „reale“ (komplexe) Datensätze/Zusammenhänge in einer Datenstruktur erfassen?

- Größe/Wert ➡ Abstraktion ➡ Datentyp in der Programmiersprache
- Trivial: Ganze Zahlen ➡ Integer, Ja/Nein-Entscheidung ➡ Boolean
- Nicht-(so)-trivial: Streckennetz, Adressbuch, Rechnernetze....



Datentyp für komplexere Datenstrukturen

➤ Problem: Abbildung komplexer, (topo)logisch zusammenhängender, (zeit)flexibler Strukturen im Code

Statische Datentypen:

- Größe vor dem Programmaufruf festgelegt
- Zusammenhänge müssen explizit abgebildet werden („wer ist mit wem verknüpft?“)

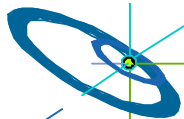
```
TYPE tConstants
  TYPE (tICons)      :: IC
  TYPE (tTime)      :: TIME
  TYPE (tAnalyse)   :: ANALYSE
  REAL              :: Pi
  INTEGER           :: EqType
  REAL              :: gamma
  ...
```

Dynamische Datentypen:

- Größe während des Programmablaufs modifizierbar
- Zusammenhänge/Strukturen in der Datenstruktur bereits implizit enthalten



Datentyp: Pointer

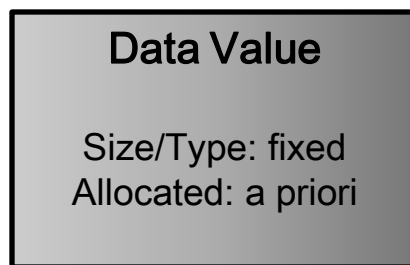


Pointerdefinition

➤ **Statische Variable:**

- Deskriptor einer Speicherstelle, incl. Typ und Speicheradresse
- Alles bis auf Inhalt/Wert festgelegt
- Speicherplatz wird für entsprechenden Typ beim Programmstart reserviert und aufgeräumt

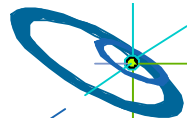
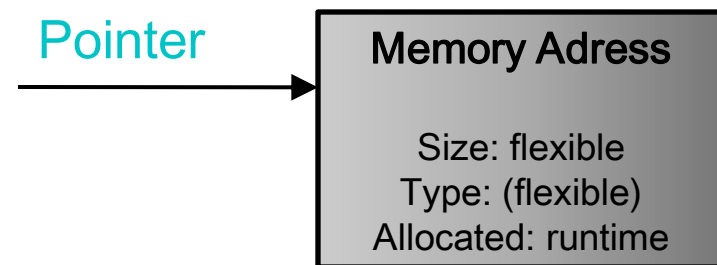
Static Variable



➤ **Pointer:**

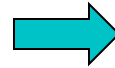
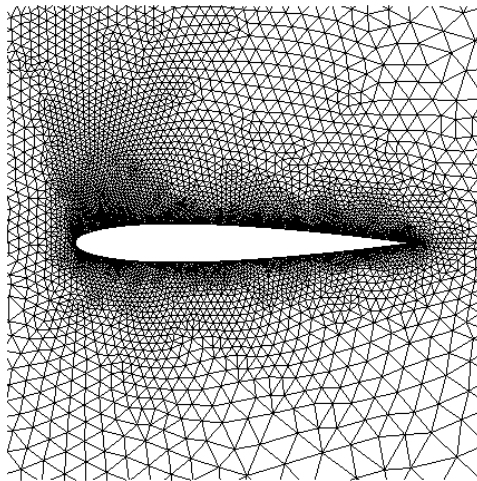
- Zeiger auf eine Speicheradresse
- Flexibel, „umbiegbar“
- Speicherplatz wird für bei Allokation für den Pointer angelegt
- Typ flexibel (C), fest (Fortran)
- (De)Allozierung

Pointer



Anwendungsbeispiele

Ausgangsproblem:
Unstrukturiertes Netz mit flexibler Konnektivität

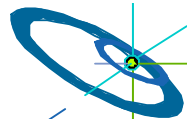


Statische Arraystruktur:

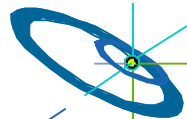
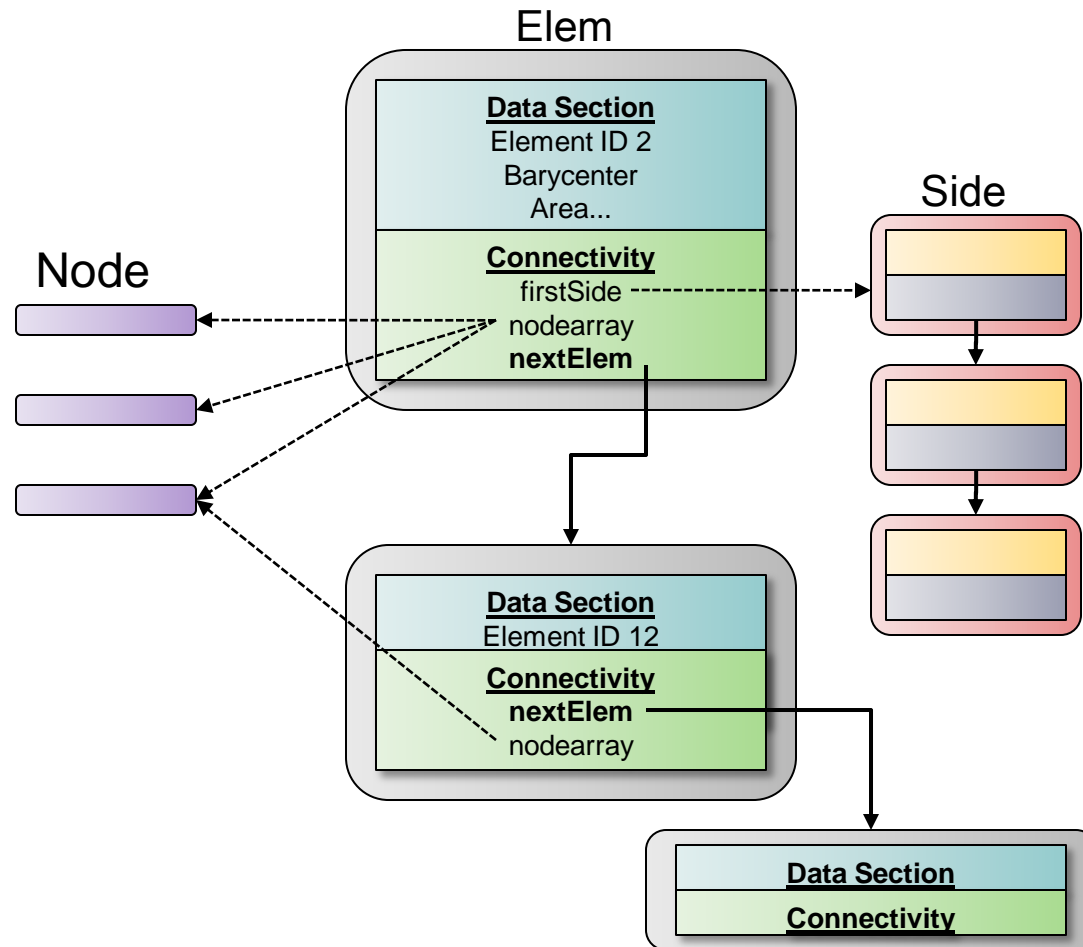
| Element 2 | |
|-----------|--------------------------|
| | Verbunden mit Knoten 23 |
| | Verbunden mit Knoten 219 |
| | Verbunden mit Knoten 33 |
| | Nachbar von Elem 13 |
| | Nachbar von Elem 19 |
| | Nachbar von Elem 333 |
| | Verbunden mit Seite 56 |
| | Verbunden mit Seite 22 |
| | Verbunden mit Seite 90 |



Overhead, Flexibilität?



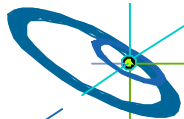
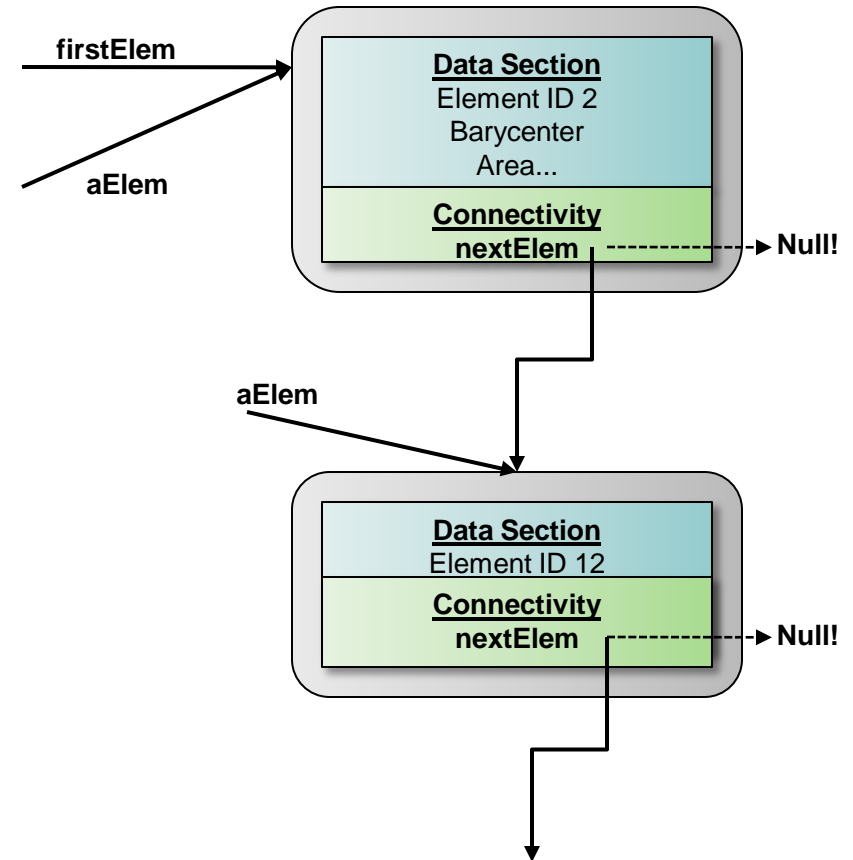
Anwendungsbeispiel: Verkettete Liste



Anwendungsbeispiel: Verkettete Liste II

➤ Erstellen der Liste:

1. Allokation und Befüllen des ersten Elements
2. Setzen des Laufpointers aElem
3. Allokation und Befüllen von aElem%nextElem
4. Setzen des Laufpointers aElem
5. ...



Linked Lists

➤ Linked Lists: Zusammenfassung

➤ **Pros:**

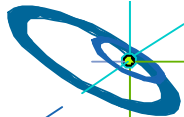
- Flexible, komplexe Datenstrukturen
- Löschen/Einfügen zur Laufzeit

➤ **Cons:**

- Elemente finden/ordnen nur durch Traversierung
- Nur unidirektional
- Langsamer als Arrays

➤ **Best Practise:**

- Pointer => Nullify bei der Definition
- Listenende klar markieren
- Einstiegs/Firstpointer nur auf der rechten Seite der Zuweisung!
- (De)Allozierung nicht vergessen



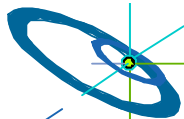
Anwendungsbeispiel: Pointerarrays

➤ Problem: Parallelisierung mit OpenMP

- Generell: Parallelisierung des Rechenkerns wünschenswert (kaum noch Rechnungen auf SingleCores)
- Für FV: Flußberechnung maßgeblich, daher schneller Zugriff auf die Zellseiten nötig
- Problem: Linked Lists mit OpenMP sehr schlecht parallelisierbar, Array-Struktur vorteilhafter



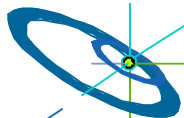
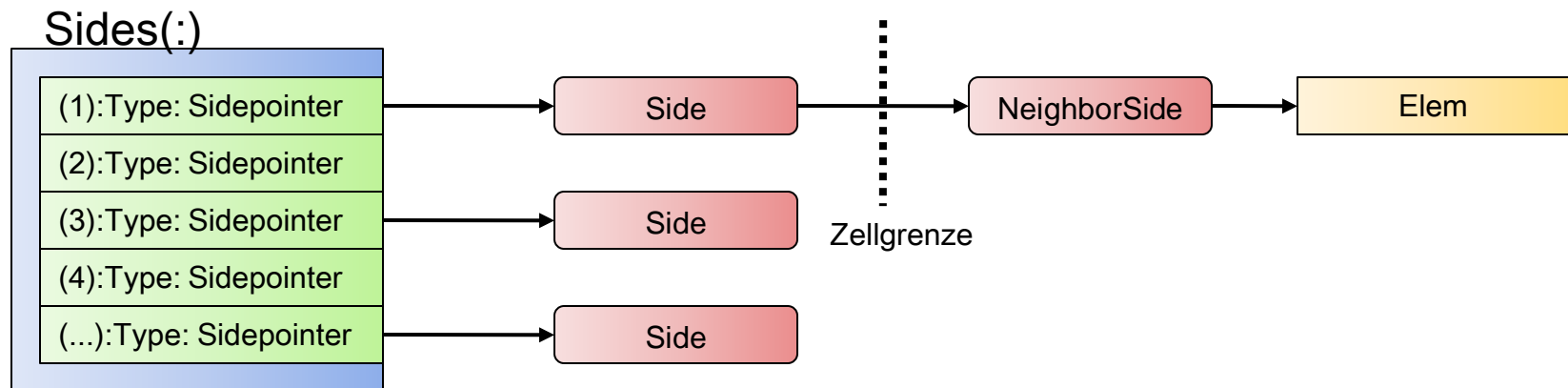
Array aus Pointern für Seitenadressierung



Anwendungsbeispiel: Pointerarrays II

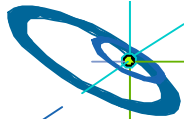
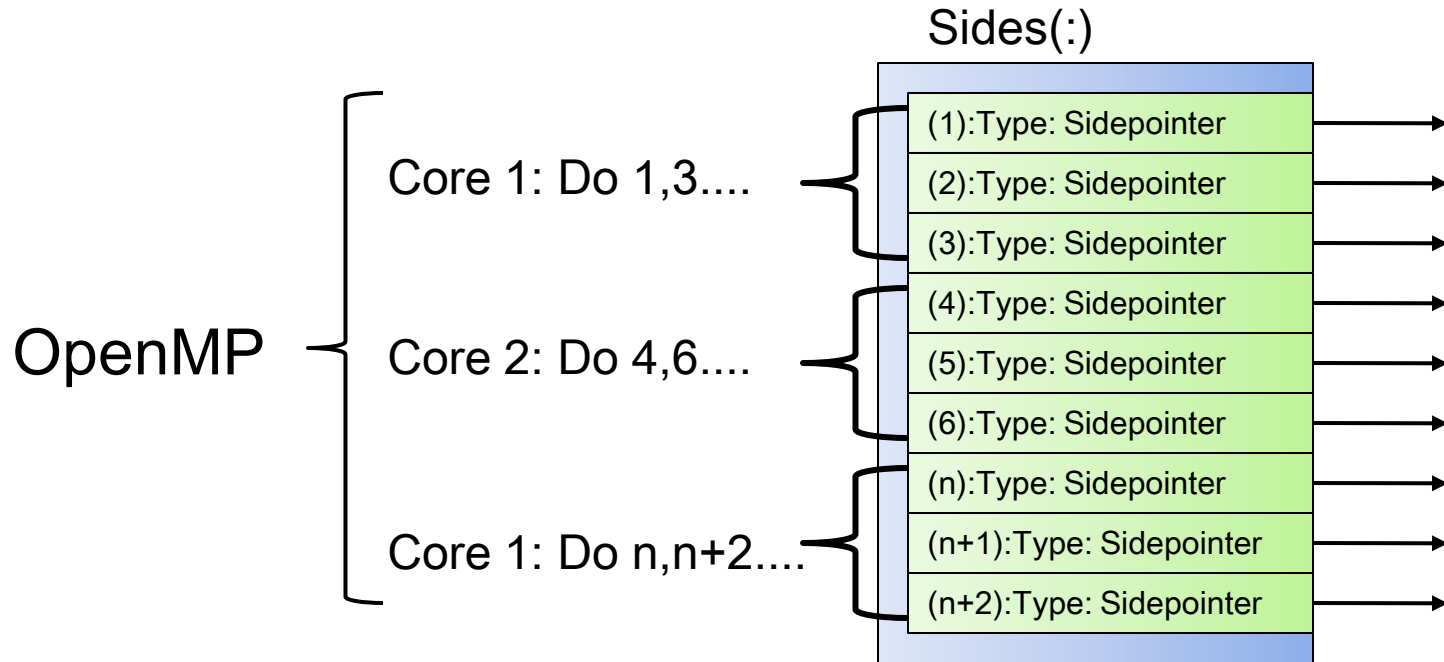
➤ Fortran-Problem: Array aus Pointern nicht direkt deklarierbar

- Daher: Hilfskonstrukt „Type“, der nur einen Pointer enthält
- Daraus: Array of Records, die nur einen Pointer auf die jew. Seite enthalten
- Somit alle Seiten in einer Array-Struktur ansprechbar



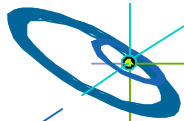
Anwendungsbeispiel: Pointerarrays III

➤ Arraystruktur der Sides leicht parallelisierbar



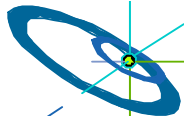
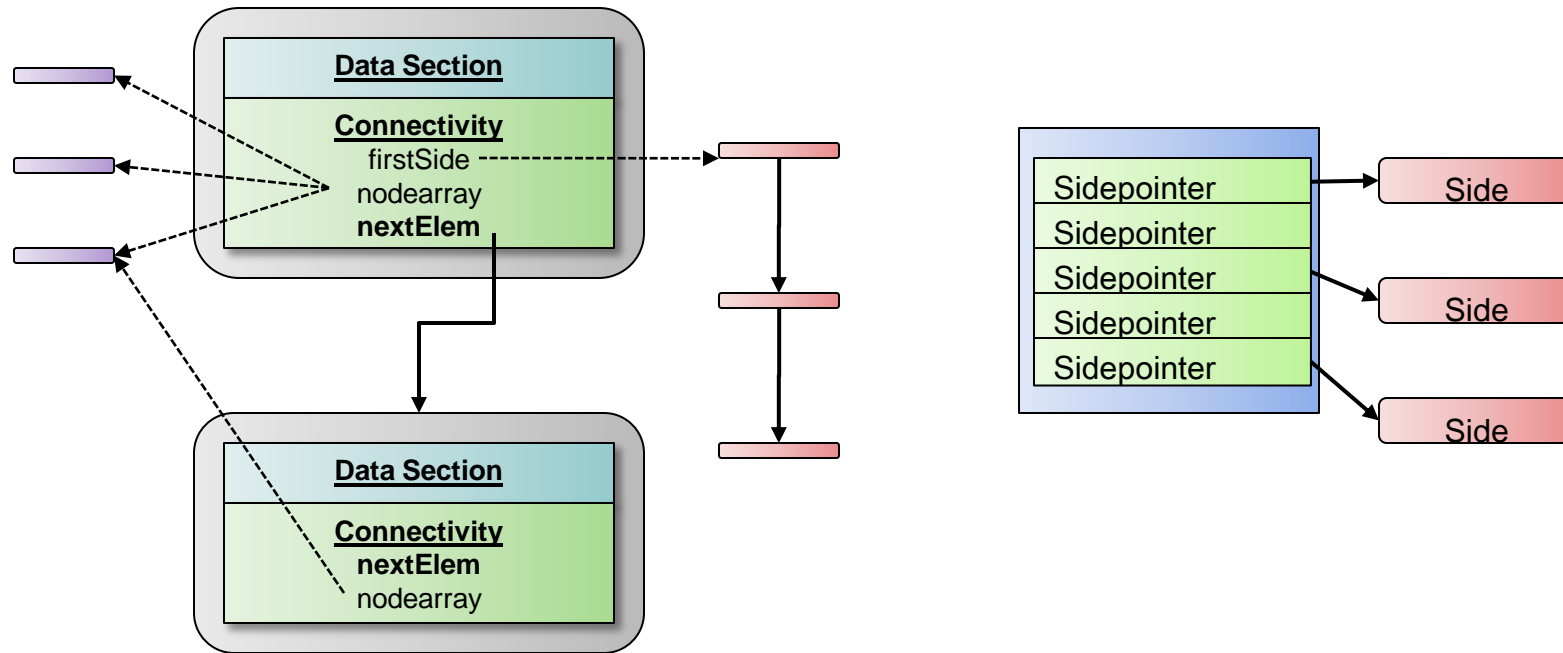
Zusammenfassung I - Pointer allgemein

- Pointer sind allozierbare Datentypen, die flexibel auf Speicheradressen zeigen können
- Abbildung komplexerer Datenstrukturen
- Erstellung flexibler logischer Verknüpfungen zur Laufzeit
- Vorsicht beim Umgang mit Allozierung, Deallozierung, „Gargabe Collection“, Segmentation-Faults
- Sorgfältigere Programmierweise erforderlich



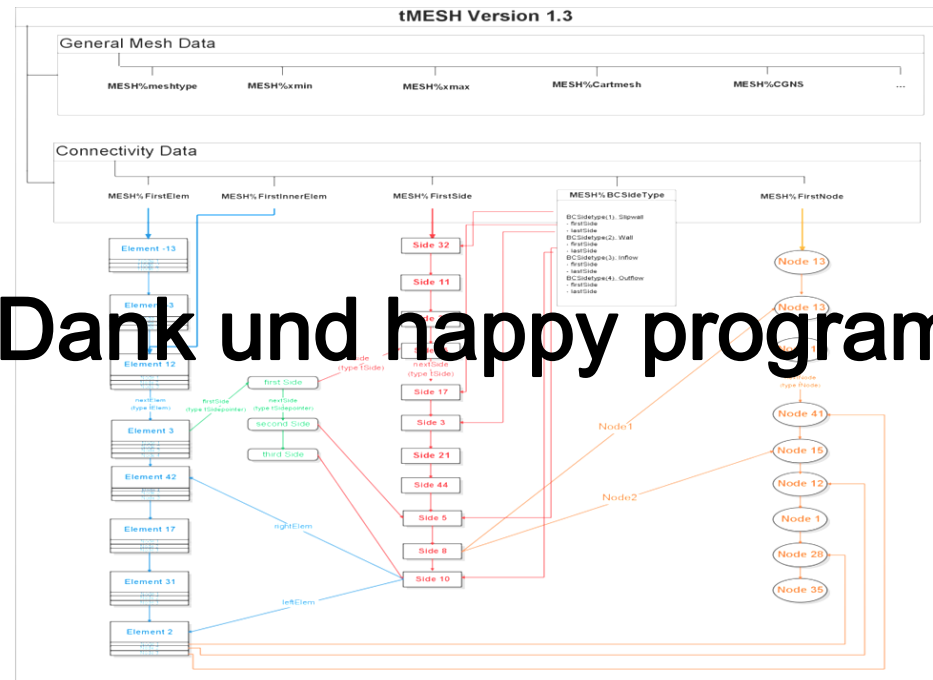
Zusammenfassung II – Pointer in CFDFV

Zwei Konzepte: Linked Lists und Pointerarrays



Literatur

1. Stephen J. Chapman: Fortran 90/95 for Scientists and Engineers
2. Michael Metcalf and John Reid: Fortran 90/95 explained
3. http://de.wikibooks.org/wiki/Fortran:_Fortran_95:_Zeiger



Vielen Dank und happy programming!

