MPI Datentypen

Seminar Effiziente Programmierung

Michel Böker

Universität Hamburg

Fachbereich Informatik

19. Januar 2021

Gliederung

1. Grundlagen
   - Was und wofür ist MPI?
   - MPI-Routinen
   - Demo
2. Arten der Kommunikation
   - Point-To-Point
   - Kollektive Kommunikation
3. Datentypen
   - Vordefinierte
   - Abgeleitete

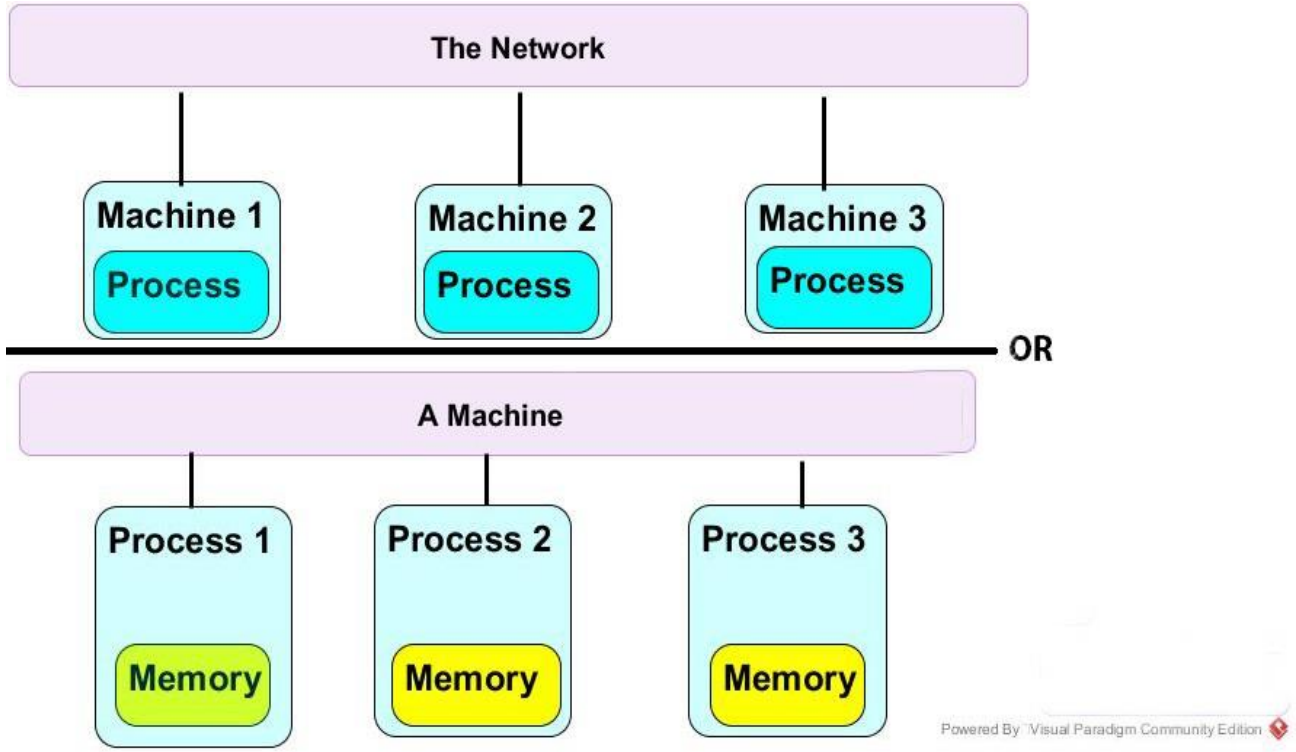
Grundlagen

1.Grundlagen
- **Was und wofür ist MPI?**
- MPI-Routinen
- Demo

2. Arten der Kommunikation
3.Datentypen

Message Passing Interface

- Basiert auf dem Message-Passing-Programmiermodell
- Prozess kann nicht auf Speicher eines anderen Prozesses zugreifen
- Berechnung ⟷ Übertragung



Message Passing Interface

- Standard-Library für den Nachrichtenaustausch bei parallelen Berechnungen auf verteilten Systemen
- legt eine Programmierschnittstelle fest
- Allgegenwärtig im HPC
- Zwei große MPI-Anbieter

Message Passing Interface

Vorteile von MPI:

- ✓ standardisiert
- ✓ portabel
- ✓ weit verbreitet
- ✓ skalierbar
- ✓ Funktionalität

MPI Routinen

Fortran

MPI_Init (ierror)

- Muss als erste MPI-Funktion aufgerufen werden

MPI_Finalize (ierror)

- Muss als letzte MPI-Funktion aufgerufen werden

MPI_Comm_size (comm, size, ierror)

- Liefert Anzahl der Prozesse in der MPI-Umgebung

MPI_Comm_rank (comm, rank, ierror)

- Liefert den Rang des aktuellen Prozesses in der MPI-Umgebung

Aufbau MPI Programm

first_program.F90 | send_recv.f90

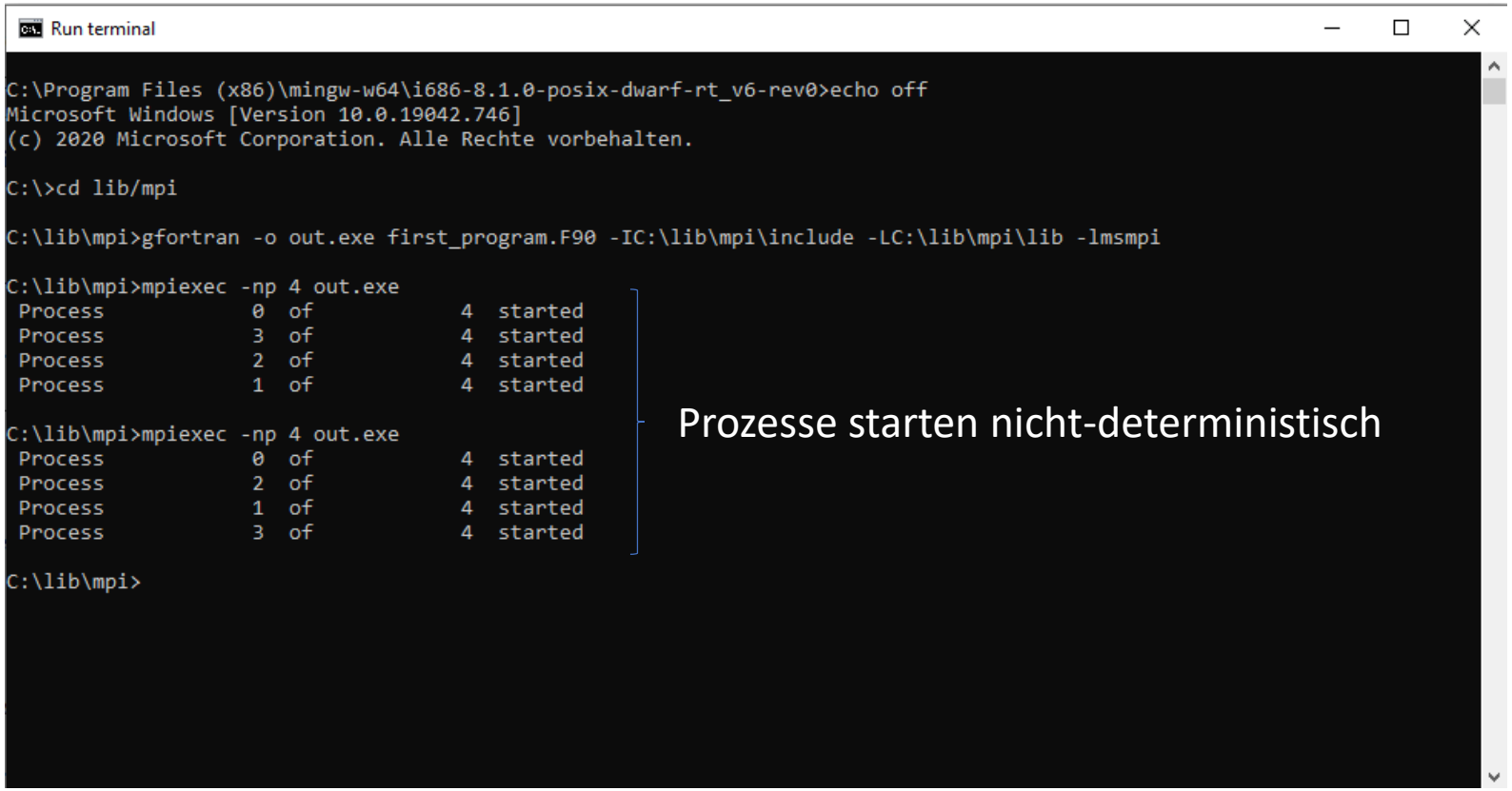
C: > lib > mpi > first_program.F90

```
1   program first_program
2   ! Include MPI Module
3   use mpi
4   implicit none
5       ! Data declarations for MPI
6       integer :: rank, num_procs, ierror
7
8       call MPI_INIT(ierror)
9       call MPI_COMM_SIZE(MPI_COMM_WORLD, num_procs, ierror)
10      call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
11      print*, 'Process ', rank, ' of ', num_procs, ' started'
12      call MPI_FINALIZE(ierror)
13  end
```

Aufbau MPI Programm

1.Grundlagen
- Was und wofür ist MPI?
- MPI-Routinen
- **Demo**

2. Arten der Kommunikation
3.Datentypen



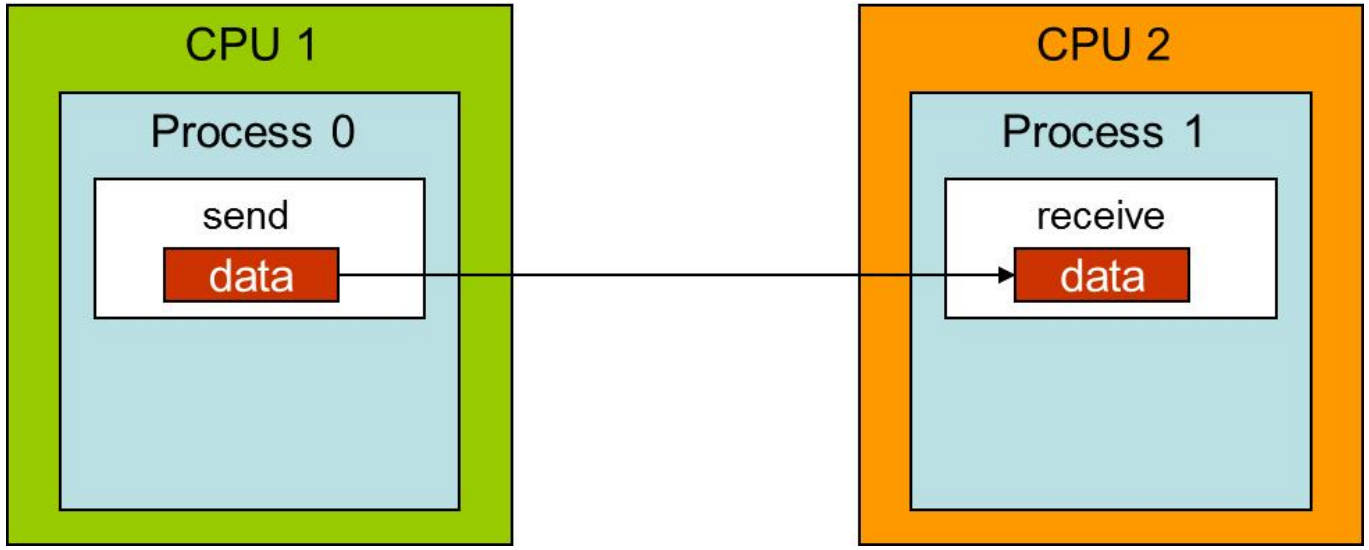
Arten der Kommunikation

Arten der Kommunikation

- Point-To-Point
- Kollektive Kommunikation

Point-To-Point Kommunikation

- Genau zwei Prozesse
- Sender und Empfänger
- Identifiziert durch ihren Rang im Kommunikator



Point-To-Point Kommunikation

Fortran

Blockierend

MPI_SEND (buf, count, datatype, dest, stag, comm, ierror)

Input	Beschreibung
buf	Adresse des Sendepuffers
count	Anzahl der zu versendenden Elemente
datatype	Typ der zu versendenden Daten
dest	Rang des Empfängers im Kommunikator
stag	Etikett zur Unterscheidung der Nachrichten
comm	Kommunikator

1.Grundlagen
2. Arten der Kommunikation
- **Point-To-Point**
- Kollektive

3.Datentypen

Point-To-Point Kommunikation

Fortran

Blockierend

MPI_RECV (buf, count, datatype, src, rtag, comm, status, ierror)

Input	**Beschreibung**
count	Anzahl der zu empfangenden Elemente
datatype	Typ der zu empfangenden Daten
src	Rang des Senders im Kommunikator
rtag	Tag zur Unterscheidung der Nachrichten
comm	Kommunikator
Output	**Beschreibung**
buf	Adresse des Empfangspuffers
status	Information über empfangene Nachricht

Point-To-Point Kommunikation

Phasen

1. Daten aus Sendepuffer kopieren & Nachricht erstellen
2. Nachricht zum Empfänger senden
3. Daten aus Nachricht lesen und im Empfangspuffer auspacken

Datentypen

- Daten im Sendepuffer zu *MPI_DATATYPE* in Send
- Gleicher *MPI_DATATYPE* in Send und Recv
- Daten im Empfangspuffer zu *MPI_DATATYPE* in Recv

Verantwortung des Programmierers

Point-To-Point Kommunikation

1.Grundlagen
2. Arten der Kommunikation
- **Point-To-Point**
- Kollektive

3.Datentypen

first_program.F90 | send_recv.f90

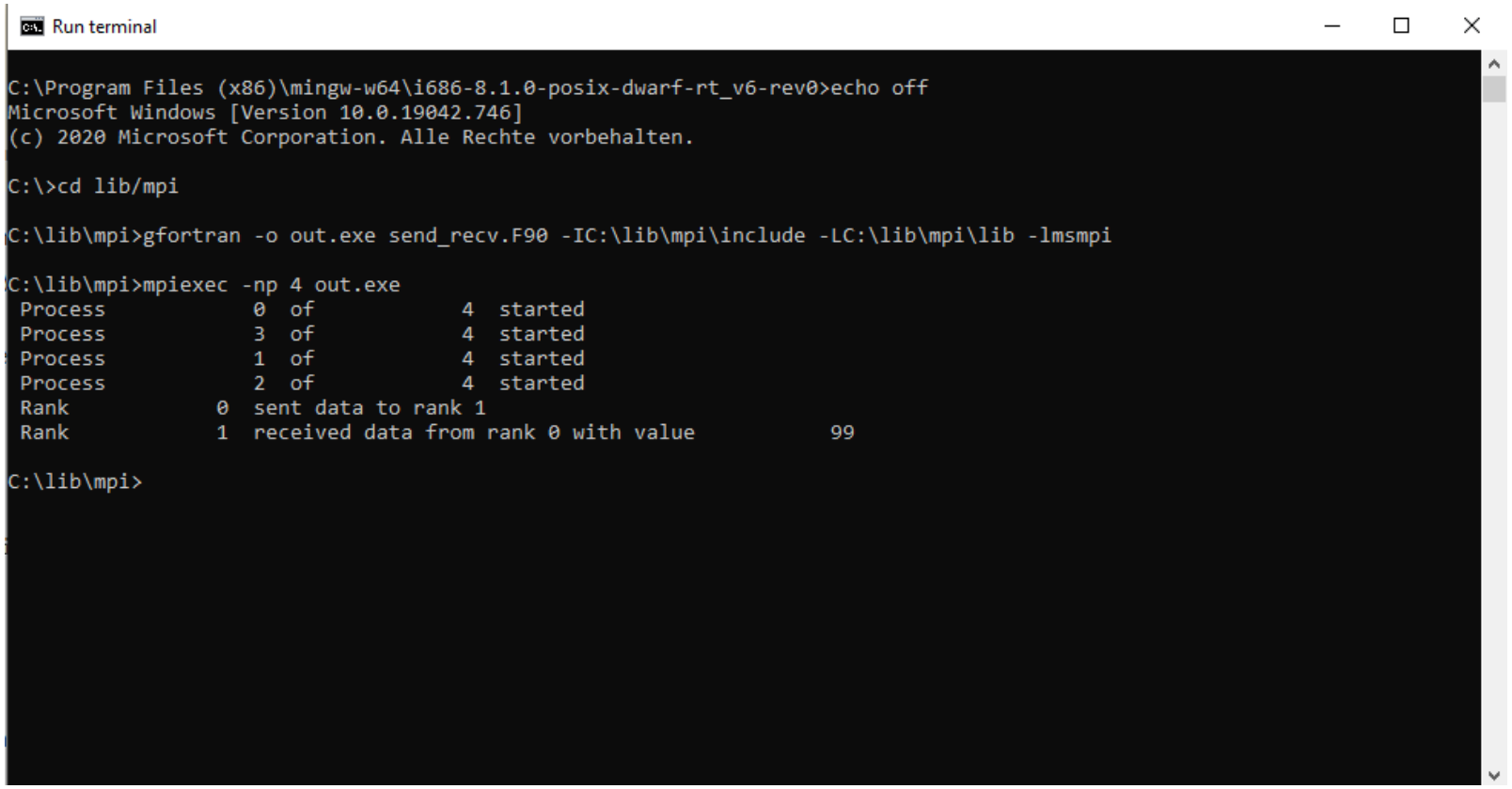
C: > lib > mpi > send_recv.f90

```
1   program send_recv
2   ! Include MPI Module
3   use mpi
4   implicit none
5       ! Data declarations for MPI
6       integer :: rank, num_procs, ierror
7       integer :: data0 = 0, data1 = 0
8       integer :: tag = 18, dest = 1, src = 0
9       integer, dimension(MPI_STATUS_SIZE) :: status
10
11      call MPI_INIT(ierror)
12      call MPI_COMM_SIZE(MPI_COMM_WORLD, num_procs, ierror)
13      call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
14      print*, 'Process ', rank, ' of ', num_procs, ' started'
15
16      ! Invoke Master Process
17      if (rank == 0) then
18          data0 = 99
19          call MPI_SEND(data0, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, ierror)
20          print*, 'Rank ', rank, ' sent data to rank 1'
21      end if
22
23      ! Invoke Non-Master Process
24      if (rank == 1) then
25          call MPI_RECV(data1, 1, MPI_INT, src, tag, MPI_COMM_WORLD, status, ierror)
26          print*, 'Rank ', rank, ' received data from rank 0 with value ', data1
27      end if
28
29      call MPI_FINALIZE(ierror)
30      end
```

Point-To-Point Kommunikation

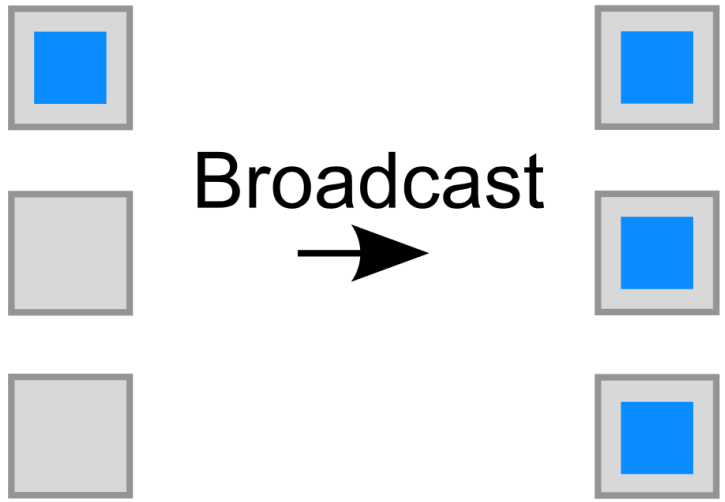
1.Grundlagen
2. Arten der Kommunikation
- **Point-To-Point**
- Kollektive

3.Datentypen



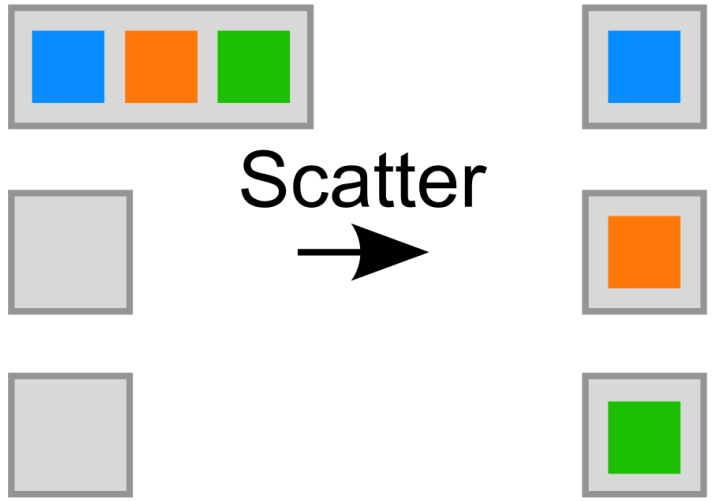
Kollektive Kommunikation

- Alle Prozesse innerhalb des Kommunikators beteiligt



Kollektive Kommunikation

- Alle Prozesse innerhalb des Kommunikators beteiligt

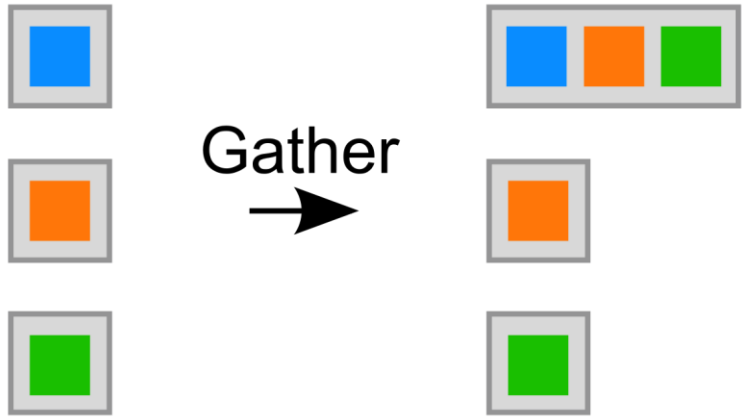


Kollektive Kommunikation

1.Grundlagen
2. Arten der Kommunikation
- Point-To-Point
- **Kollektive**

3.Datentypen

- Alle Prozesse innerhalb des Kommunikators beteiligt



Datentypen

Datentypen

- Vordefinierte
- Abgeleitete

Vordefinierte Datentypen

1.Grundlagen
2. Arten der Kommunikation
3.Datentypen
- **Vordefinierte**
- Abgeleitete

MPI Datentyp	Fortran Datentyp
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE_PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

Abgeleitete Datentypen

Wozu?

- Kommunikation verursacht Kosten

Beachte:

- Moderner Parallelrechner schafft ca. 3 Mrd. floating point operations / sec
- Nachrichtenaustausch aber nur 10 Mio. Wörter / sec

Faktor 300 !

Besser: Eine große Nachricht als viele kleine

Abgeleitete Datentypen

1.Grundlagen
2. Arten der Kommunikation
3.Datentypen
- Vordefinierte
- **Abgeleitete**

Wie?

- Konstruiere den neuen Datentyp mithilfe der dazugehörigen MPI-Routine:

MPI_TYPE_CONTIGUOUS

MPI_TYPE_VECTOR

MPI_TYPE_INDEXED

MPI_TYPE_CREATE_STRUCT

- Übergebe den neuen Datentyp:

MPI_TYPE_COMMIT

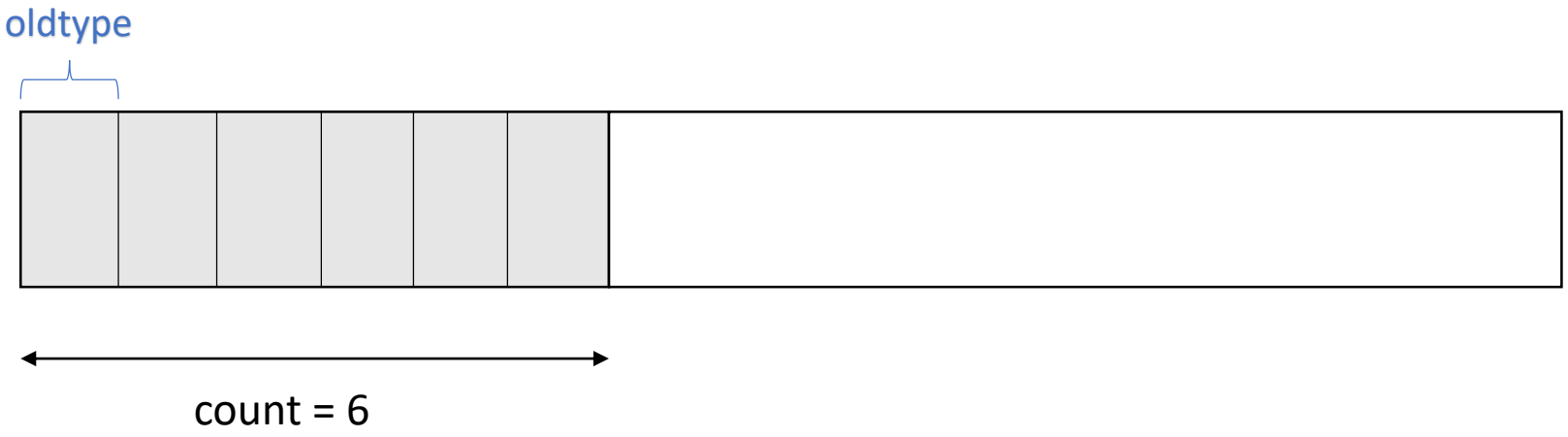
- Benutze den neuen Datentyp in send/receive etc.

Datentyp Contiguous

Fortran

MPI_TYPE_CONTIGUOUS (count, oldtype, newtype, ierror)

- Einfachste Art einen Datentyp abzuleiten
- Besteht aus einer bestimmten Anzahl gleichartiger Element



1.Grundlagen
2. Arten der Kommunikation
3.Datentypen
- Vordefinierte
- **Abgeleitete**

Datentyp Vector

Fortran

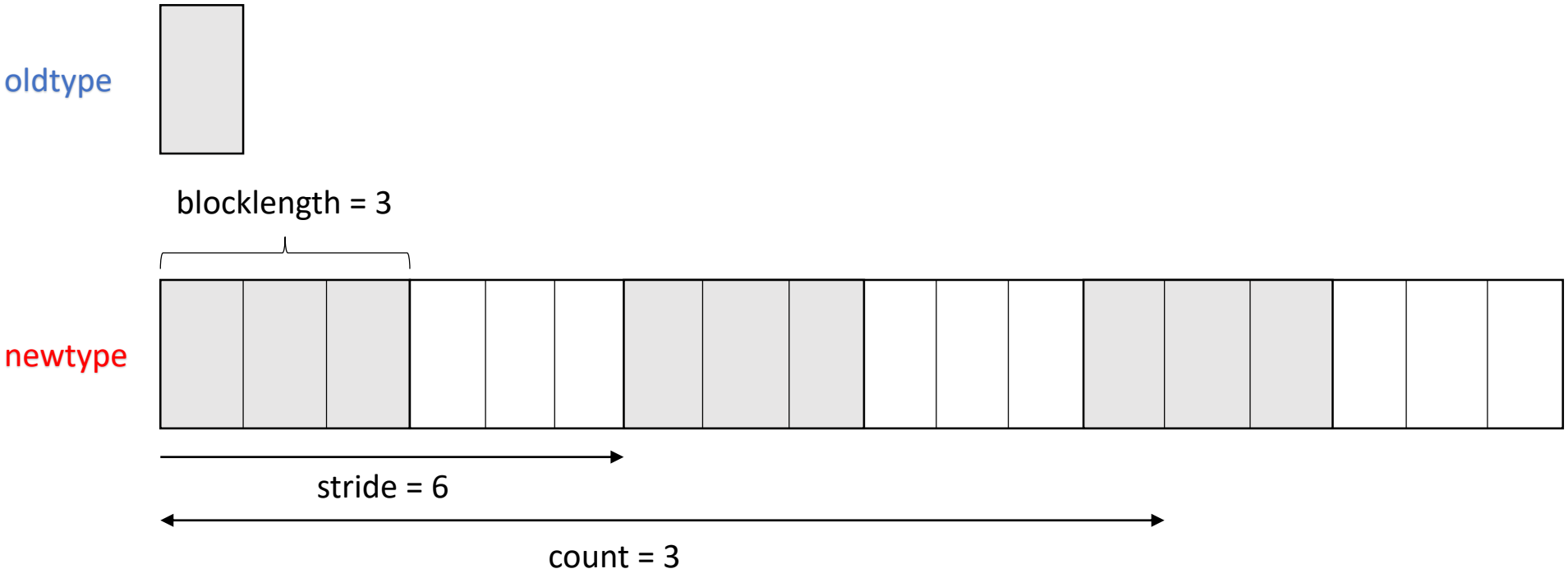
MPI_TYPE_VECTOR (count, blocklength, stride, oldtype, newtype, ierror)

- Replikation eines Datentyps in mehrere gleichgroße Blöcke
- Abstand zwischen den Blöcken

Datentyp Vector

Fortran

MPI_TYPE_VECTOR (count, blocklength, stride, oldtype, newtype, ierror)



Beispiel

count = 4

MPI_TYPE_CONTIGUOUS (count, oldtype, newtype, ierror)

5	11	25	1
6	9	31	2
71	44	3	8
5	12	24	99

MPI_SEND (a[2][0], 1, rowtype …)

Beispiel

count = 4 | blocklength = 1 | stride = 4

MPI_TYPE_VECTOR (count, blocklength, stride, oldtype, newtype, ierror)

5	11	25	1
6	9	31	2
71	44	3	8
5	12	24	99

MPI_SEND (a[0][3], 1, columntype …)

Datentyp Indexed

1.Grundlagen
2. Arten der Kommunikation
3.Datentypen
- Vordefinierte
- **Abgeleitete**

Fortran

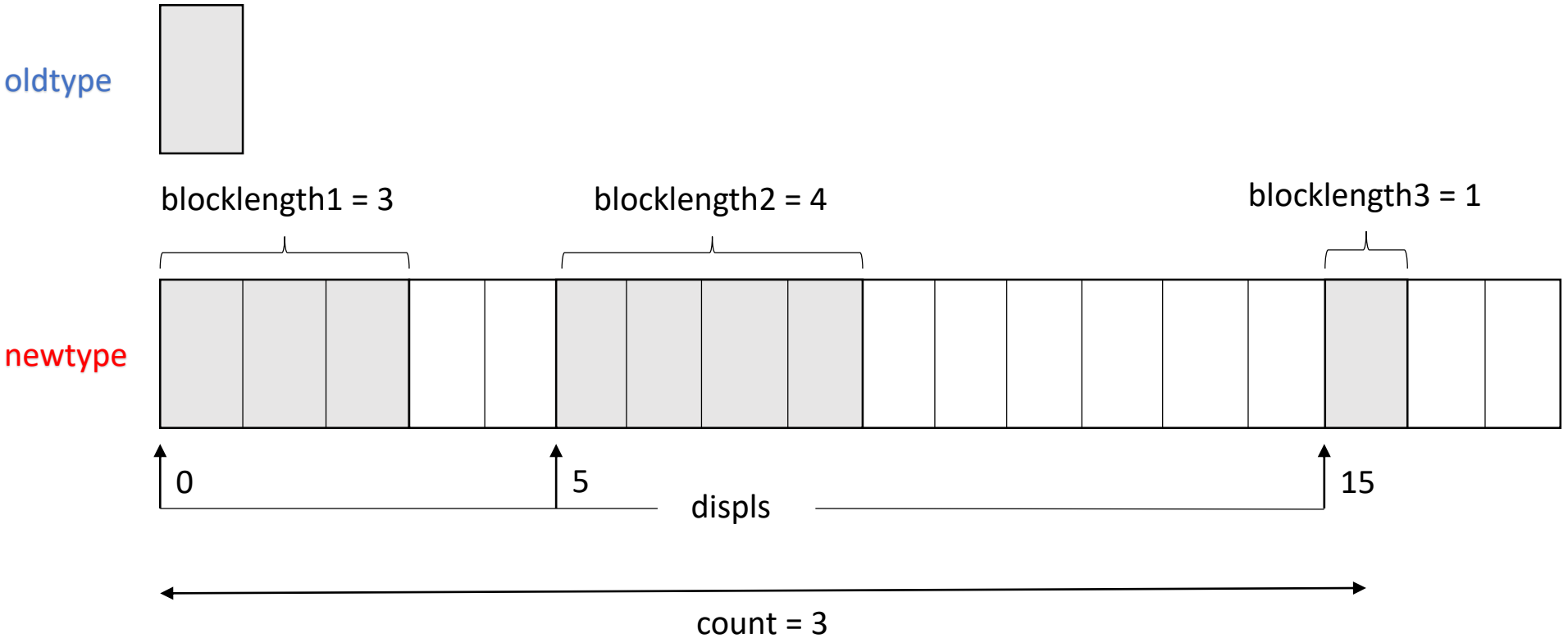
MPI_TYPE_INDEXED (count, blocklengths, displs, oldtype, newtype, ierror)

- Allgemeiner als Datentyp Vector
- Jeder Block kann eine unterschiedliche Anzahl Elemente enthalten
- Abstand zwischen Blöcken kann ebenfalls unterschiedlich sein

Datentyp Indexed

Fortran

MPI_TYPE_INDEXED (count, blocklengths, displs, oldtype, newtype, ierror)



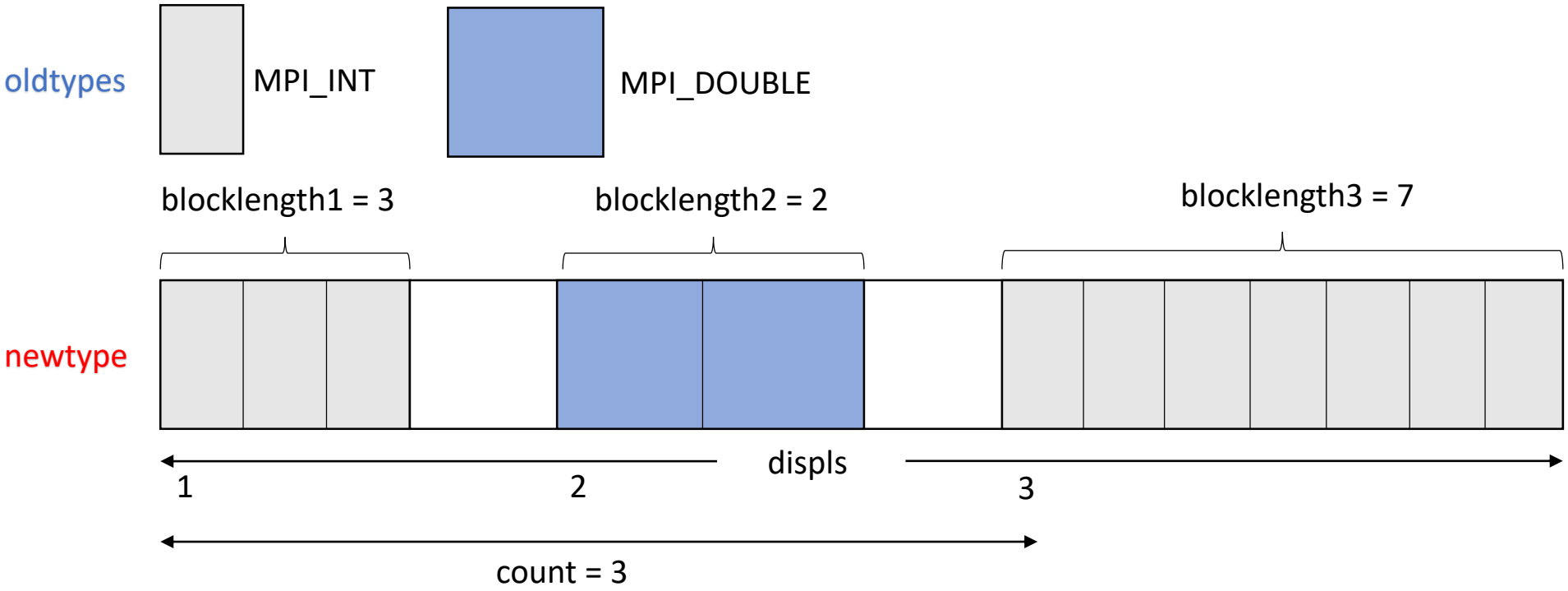
Datentyp Struct

MPI_TYPE_CREATE_STRUCT (count, blocklengths, displs, oldtypes, newtype, ierror) ***Fortran***

- Allgemeinste Form eines Typ-Konstruktors
- Jeder Block enthält Elemente eines bestimmten Datentyps
- Die verschiedenen Blöcken können sich in Datentyp, Länge und Blockanfang unterscheiden

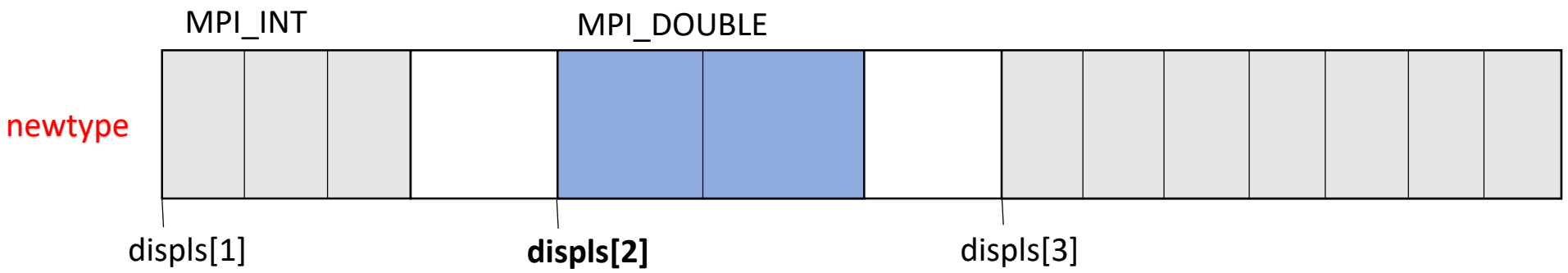
Datentyp Struct

MPI_TYPE_CREATE_STRUCT (count, blocklengths, displs, oldtypes, newtype, ierror) ***Fortran***



Displacements

- Anzahl der Bytes zwischen den Blockanfängen



MPI_GET_ADRESS (buffer.i[0], adress0, ierror)

MPI_GET_ADRESS (buffer.d[0], adress1, ierror)

displs[2] = **MPI_AINT_DIFF** (adress0, adress1, ierror)

Zusammenfassung

- MPI ist ein Standard für die parallele Berechnung auf verteilten Systemen
- Point-To-Point Kommunikation für Nachrichtenaustausch zwischen genau zwei Prozessen
- Vordefinierte Datentypen können unterschiedlich abgeleitet werden
- Abgeleitete Datentypen für die effizientere Kommunikation

Literatur

1. MPI-Forum Docs, https://www.mpi-forum.org/docs/, letzter Zugriff: 02.01.2021
2. MPICH Docs, https://www.mpich.org/static/docs/v3.3, letzter Zugriff: 02.01.2021
3. Marc Snir, Steve Otto, Steven Huss Lederman, David Walker, Jack Dongarra: MPI The Complete Reference: The MIT Press
4. The Shared Memory and Message Passing Models of Interprocess Communication, http://www.umsl.edu/~siegelj/CS4740_5740/Overview/MPSM.html, letzter Zugriff: 11.01.2021
5. Message Passing Interface Wikipedia, https://de.wikipedia.org/wiki/Message_Passing_Interface, letzter Zugriff: 09.01.2021
6. MPI Instructions for Windows, https://abhilashreddy.com/writing/3/mpi_instructions.html, letzter Zugriff: 09.01.2021
7. Using MPI with Fortran, https://curc.readthedocs.io/en/latest/programming/MPI-Fortran.html, letzter Zugriff: 11.01.2021
8. MPI Standard, https://www.mcs.anl.gov/research/projects/mpi/, letzter Zugriff: 12.01.2021