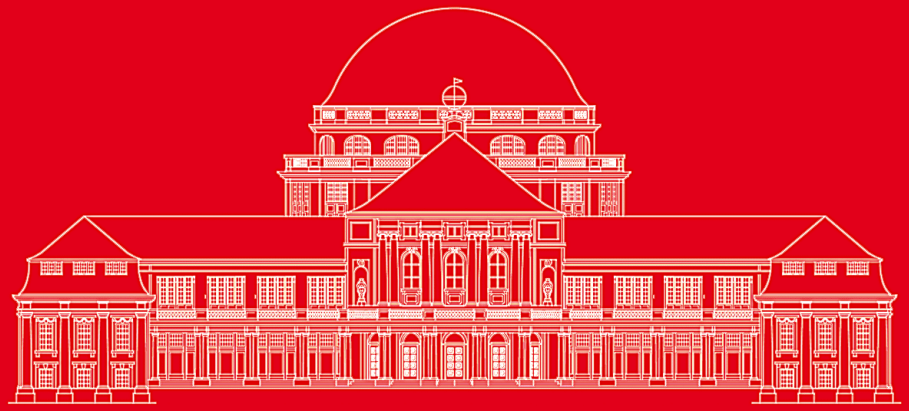
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart & Enno Zickler



Dr. Hermann-J. Lenhart
hermann.lenhart@informatik.uni-hamburg.de

MPI Einführung II:

- Kollektive Operationen
- Reduce Operation
- Scatter / Gather => gleichmäßige Matrixaufteilung
- ScatterV/ GatherV => nicht gleichmäßige Matrixaufteilung
- Halo Lines für Programmablauf

MPI Kollektive Operationen

Neben **Point-to-Point Kommunikation** mittels Send & Recv

verfügt MPI über umfangreiche Operationen zum **kollektiven Bewegen von Daten**.

MPI_BROADCAST Eine Info an alle Prozesse versenden

MPI_REDUCE „aggregierende“ Operationen (Summe; Prod) auf Matrix ausführen

MPI_SCATTER Teilarrays an Prozesse übertragen

MPI_GATHER Teilarrays zusammenführen

MPI Reduce I

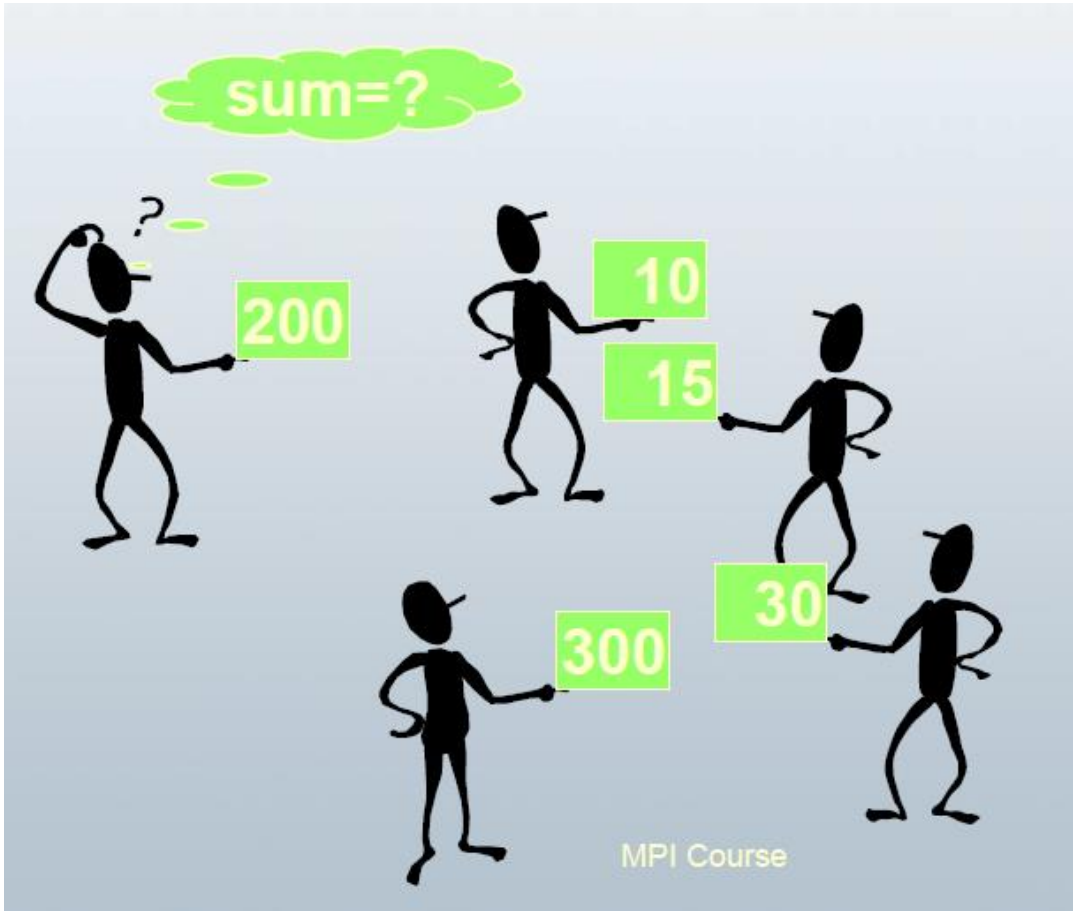
Um die Ergebnisse der einzelnen Prozesse zusammenzuführen gibt es eine Auswahl an „Reduce“ Operationen, z.B:

MPI_REDUCE(Operand, Result, Count, Datatype, Operation, Root, Comm, Ierror)

Call MPI_REDUCE(temp, sum,1, MPI_Real, MPI_SUM,0, MPI_COMM_World, Ierror)

Über die Operation MPI_SUM werden alle Resultate der Größe temp von allen Prozessen aufaddiert und in der Variable sum abgelegt.

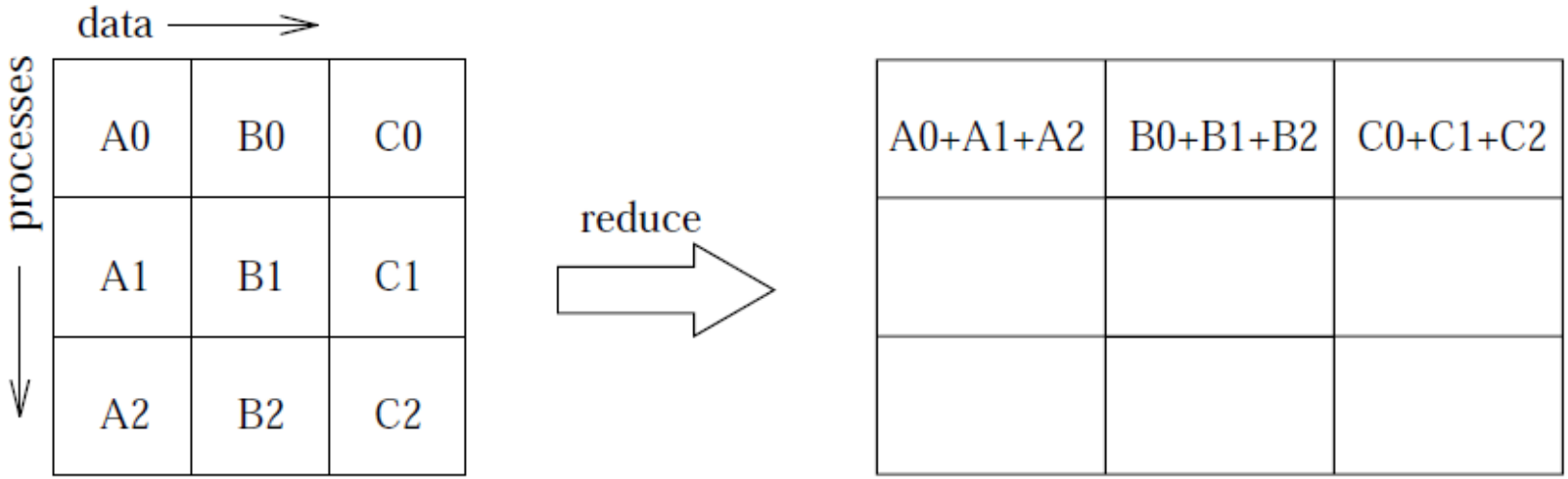
MPI Reduce I



DKRZ MPI Einführungs Kurs

MPI Reduce III

Operator SUM



(William Gropp ANL, MPI Tutorial)

MPI Reduce IV

Übersicht der möglichen MPI_Reduce Opertionen:

MPI_SUM	Summe
MPI_PROD	Produkt
MPI_MAX /MPI_MIN	Maximum/Minimum
MPI_MAXLOC	Maximum und Position des Maximums
MPI_LAND / MPI_LOR	Logical And / Logical Or

MPI Scatter I

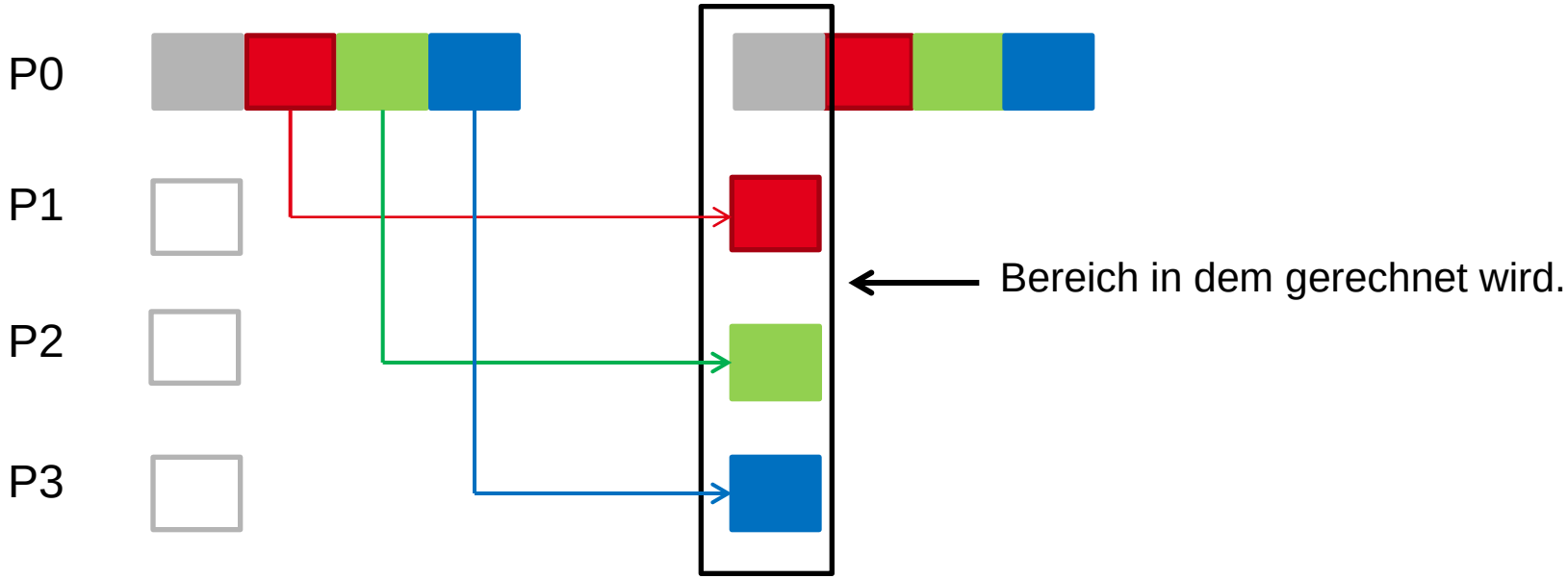
Eine Möglichkeit z.B. eine Anfangsbelegung auf die Teilarrays der Prozesse, zu übertragen bietet MPI_SCATTER:

Syntax: MPI_Scatter(Sendbuffer, Sendcount, Sendtype,
Recvbuffer, Recvcount, Recvtype,
Root, Comm, Ierror)

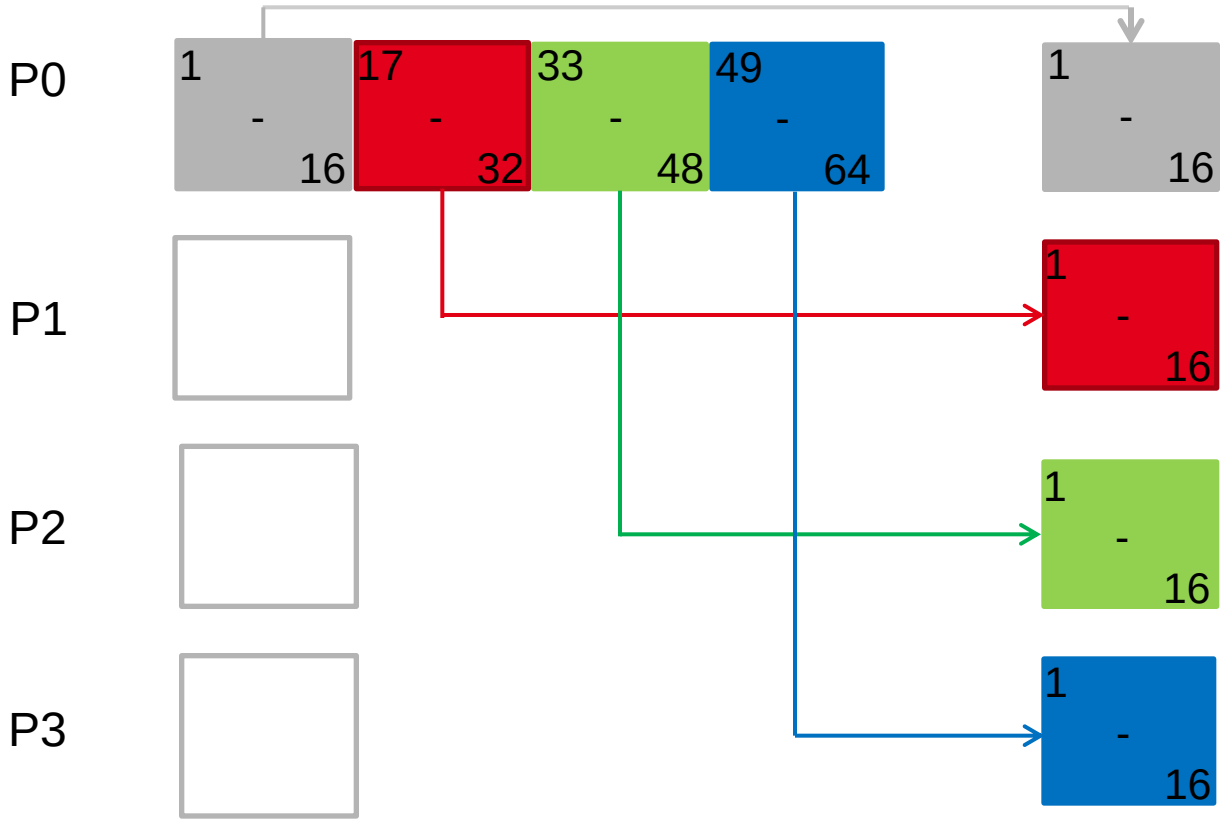
Call MPI_SCATTER(Send_Message, Send_Count, Send_Datatype,
Recv_Message, Recv_Count, Recv_Datatype,
0, MPI_COMM_World, Ierror)

MPI Scatter II

Die Daten werden von P0 an die anderen Prozesse P1 – P3 gesendet.



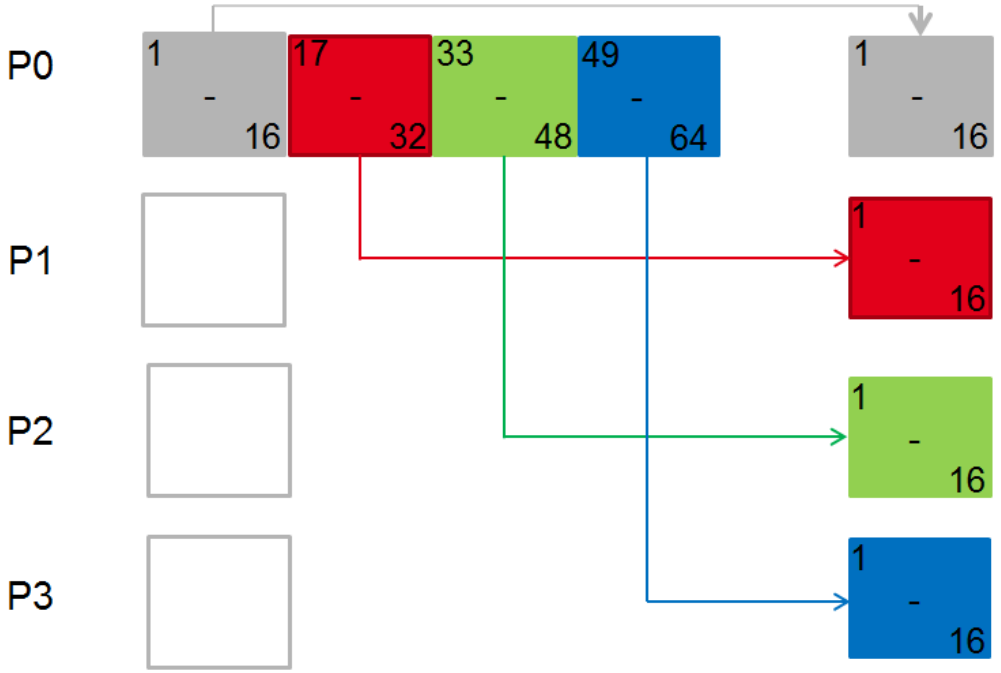
MPI Scatter: Beispiel array2D(8X8) => 4 Teile chunk2D(4X2)



MPI Scatter:

```
integer, dimension(8, 8) :: array2D !   all the 2D data
integer, dimension(8, 2) :: chunk2D ! piece of 2D data each process works on

 ! int MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)
 ! 16 = 8*2 = size(chunk)
CALL MPI_SCATTER(array2D, 16, MPI_INTEGER, chunk2D, 16, MPI_INTEGER, &
0, MPI_COMM_WORLD, mpi_ierr)
```

Abbildung von Matrix
array2D auf 4 Teile chunk2D



MPI Gather I

Eine Möglichkeit Teilarrays der Prozesse (z.B. für I/O Zwecke) wieder zusammenzuführen, bietet MPI_GATHER:

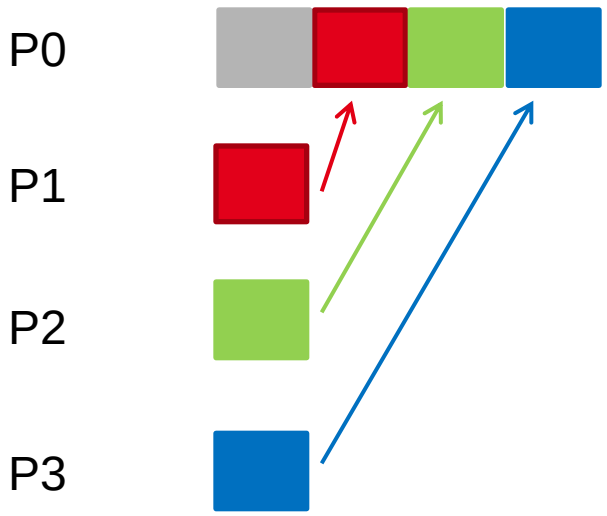
Syntax: MPI_Gather(Send_Message, Send_Count, Send_Datatype,
Recv_Message, Recv_Count, Recv_Datatype,
Root, Comm, Ierror)

Call MPI_GATHER (temp, 1, MPI_Real,
temps,1, MPI_Real,
0, MPI_COMM_World, Ierror)

MPI Gather II

Call MPI_GATHER(temp, 1, MPI_Real,

tempAll,1, MPI_Real, 0, MPI_COMM_World, Ierror)



MPI Gather III

```
   integer, dimension(8, 8) :: array2D ! all the 2D data
   integer, dimension(8, 2) :: chunk2D ! piece of 2D data each process works on
   integer :: mpi_ierr, mpi_rank, mpi_size


    ! int MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype,
root, comm, ierr)
    ! 16 = 8*2 = size(chunk)
    CALL MPI_SCATTER(array2D, 16, MPI_INTEGER, chunk2D, 16, MPI_INTEGER, 0,
MPI_COMM_WORLD, mpi_ierr)

    CALL computation(chunk2D)  ! where the magic happens

    ! Chunks are gathered by process 0.
    CALL MPI_GATHER(chunk2D, 16, MPI_INTEGER, array2D, 16, MPI_INTEGER, 0,
MPI_COMM_WORLD, mpi_ierr)
```

MPI Gather III

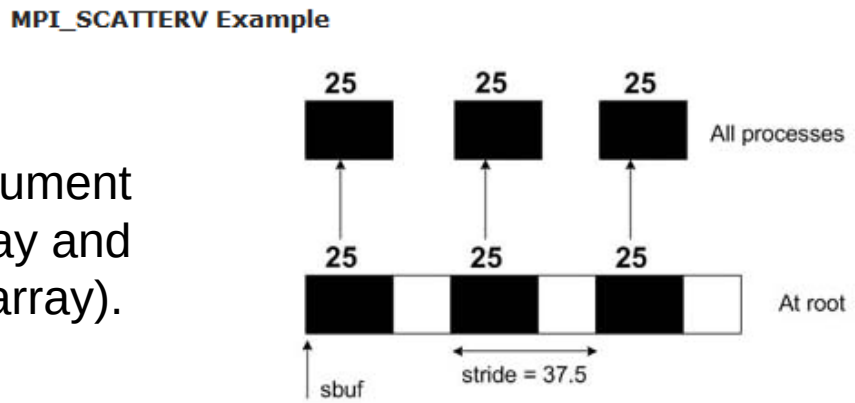
```
   integer, dimension(8, 8) :: array2D ! all the 2D data
   integer, dimension(8, 2) :: chunk2D ! piece of 2D data each process works on
   integer :: mpi_ierr, mpi_rank, mpi_size


    ! int MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype,
root, comm, ierr)
    ! 16 = 8*2 = size(chunk)
    CALL MPI_SCATTER(array2D, 16, MPI_INTEGER, chunk2D, 16, MPI_INTEGER, 0,
MPI_COMM_WORLD, mpi_ierr)

    CALL computation(chunk2D)  ! where the magic happens

    ! Chunks are gathered by process 0.
    CALL MPI_GATHER(chunk2D, 16, MPI_INTEGER, array2D, 16, MPI_INTEGER, 0,
MPI_COMM_WORLD, mpi_ierr)
```

MPI Scatter-Gather: In der Literatur

MPI_Gatherv and MPI_Scatterv are the variable-message-size versions of MPI_Gather and MPI_Scatter.

MPI_Scatterv extends the functionality of MPI_Scatter to permit a varying count of data from each process.

It does this by changing the count argument from a single integer to an integer array and providing a new argument displs (an array).



(Quelle: https://www.cac.cornell.edu/vw/MPIcc/gathervscatterv.aspx)

MPI Scatter -> ScatterV

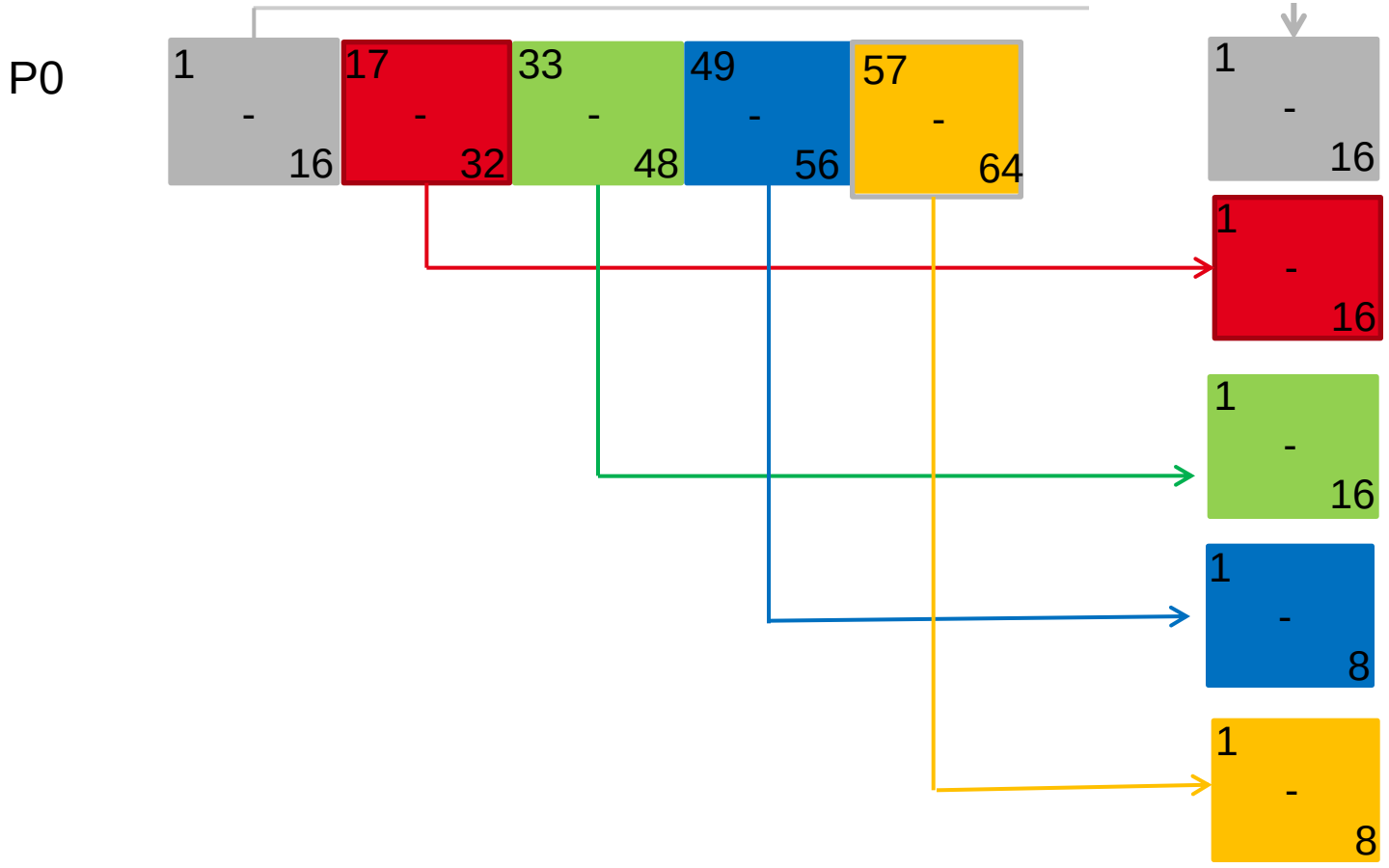
Call MPI_SCATTER (Send_Message, Send_Count, Send_Datatype,
Recv_Message, Recv_Count, Recv_Datatype,
Root, *Comm*, Ierror)

Call MPI_SCATTERV(Send_Mesage, Send_Count, **Displacement**, Send_Datatype,
Recv_Mesage, Recv_Count, Recv_Datatype,
Root, *Comm*, Ierror)

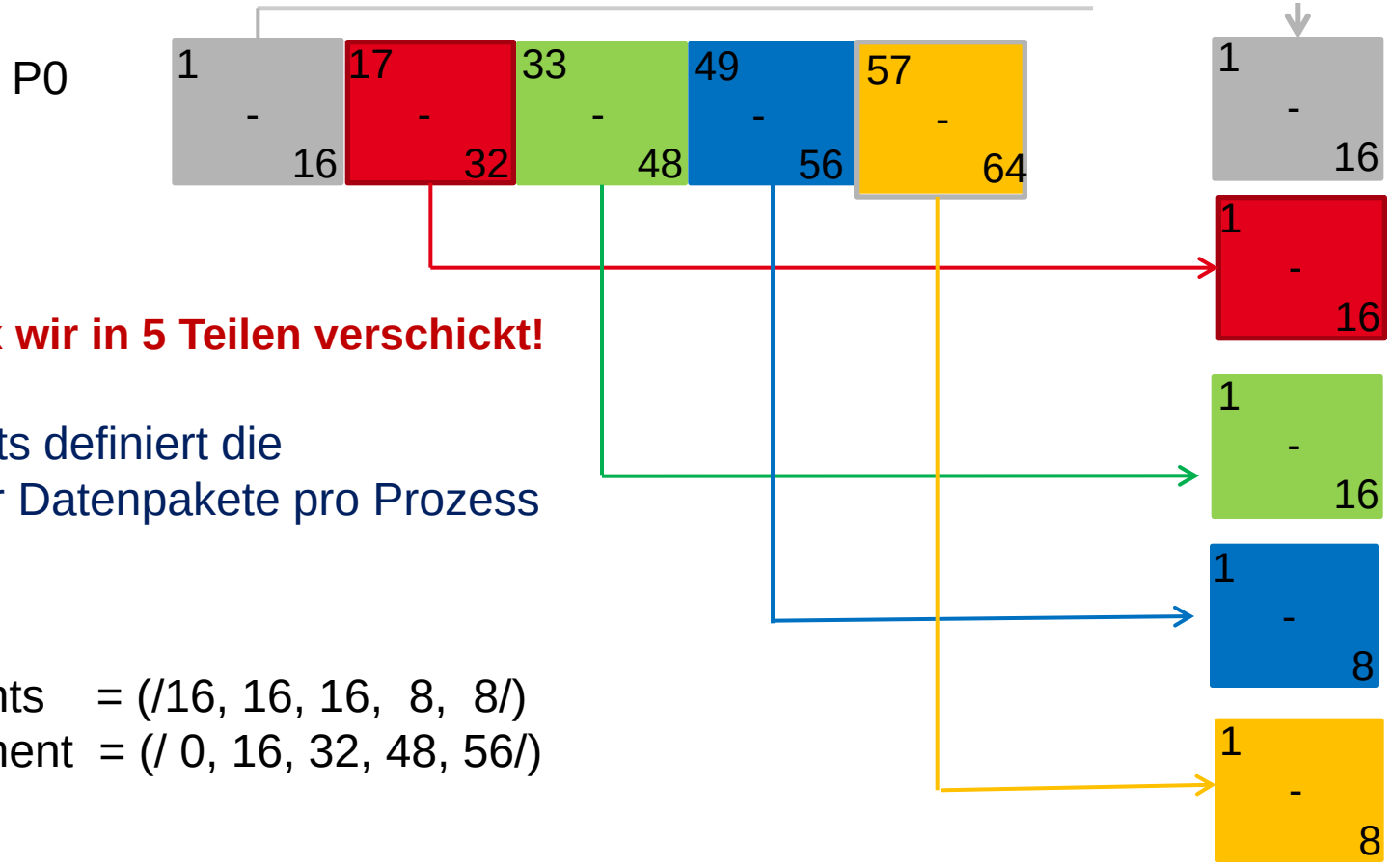
Änderung: ***Send_count*** ***von Integer - Zahl*** → ***Integer-Vektor***

Neu: ***Displacement-Vektor***

MPI ScatterV: Beispiel array2D(8X8) => 5 Teile chunk2D



MPI ScatterV: Beispiel array2D(8X8) => 5 Teile chunk2D



Die Matrix wir in 5 Teilen verschickt!

sendcounts definiert die
Größe der Datenpakete pro Prozess

Sendcounts = (/16, 16, 16, 8, 8/)
displacement = (/ 0, 16, 32, 48, 56/)

MPI ScatterV / GatherV

```
! für 5 prozesse

integer, allocatable, dimension(8,8) :: array
integer, allocatable, dimension(:,:) :: chunk, displacement, sendcounts


sendcounts      = (/16, 16, 16,  8,  8/)
displacement    = (/ 0, 16, 32, 48, 56/)

call MPI_SCATTERV(array, sendcounts, displacement, MPI_INTEGER, chunk,
sendcounts(rank+1), MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)

call MPI_GATHERV(chunk, sendcounts(rank+1), MPI_INTEGER, array,
sendcounts, displacement, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

Displacement:
Beginnt mit Null und bezieht sich jeweils auf das letzte Element des vorherigen Abschnitts!

Beispiel: Die Matrix wir in Vektoren zerlegt.

sendcounts = (/2, 10, 5, 8, 5, 2, 2, 2, 2, 2/)

displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)

displacement den Versatz in der Matrix

```
call MPI_SCATTERV(array, sendcounts, displacement, MPI_INTEGER,
                  chunk, sendcounts(rank+1),       MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

MPI Scatterv – Beispiel A – **Zuweisung II**

Beispiel: Die Matrix wir in Vektoren zerlegt.

sendcounts = (/2, 10, 5, 8, 5, 2, 2, 2, 2, 2/)

displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)

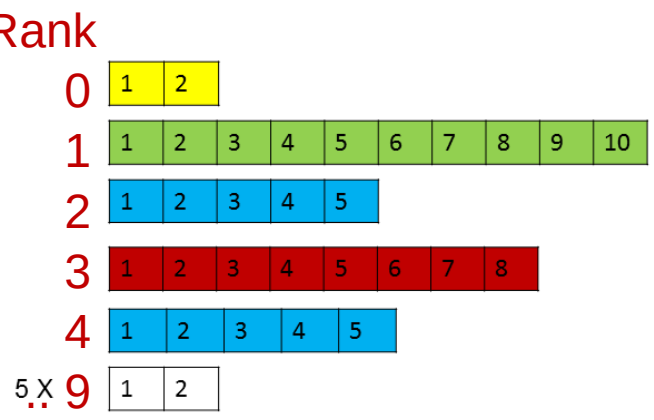
1	2								10
11	12					17			20
				25					30
	32		34		36		38		40

displacement den Versatz in der Matrix

```
call MPI_SCATTERV(array, sendcounts, displacement, MPI_INTEGER,
                  chunk, sendcounts(rank+1),       MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

Die Datenpakete werden den einzelne Vektoren (chunk) pro Prozess zugewiesen

Entsprechend der variablen Größe pro Prozess definiert durch sendcount(rank+1)



MPI Scatterv – Beispiel mit Lücke

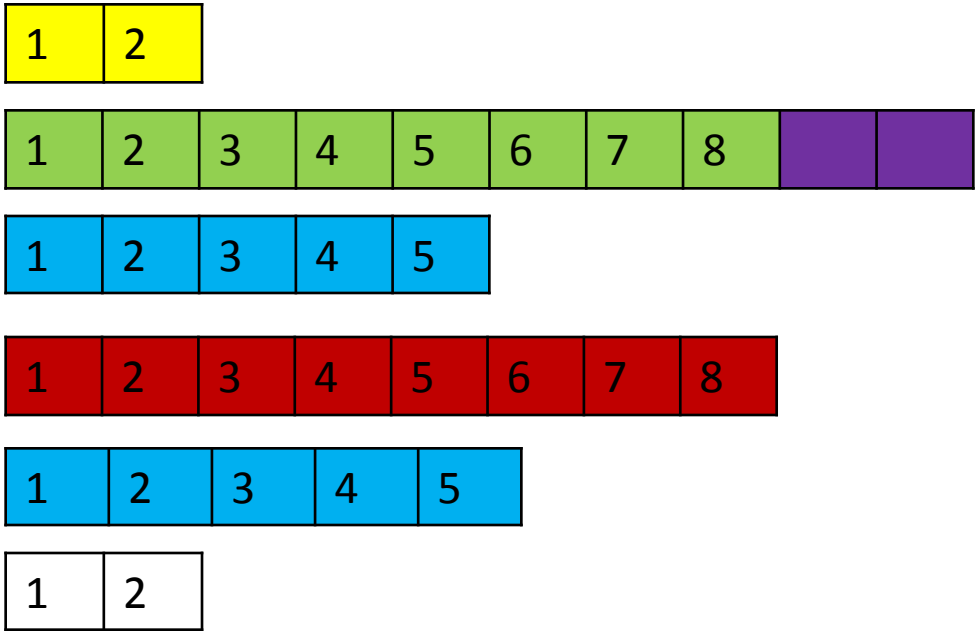
1	2								10
11	12					17			20
				25					30
	32		34		36		38		40

Aufteilung Matrix 10x4

auf einzelne Vektoren

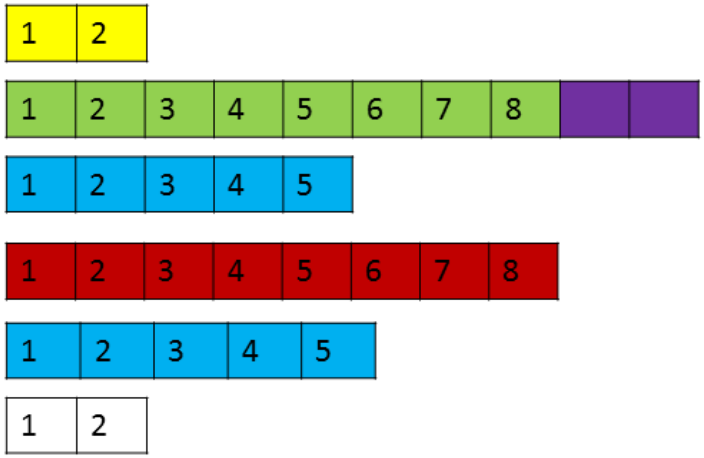
unterschiedlicher Länge

mit Lücke in den Daten



Für 10 Prozesse:

integer :: sendcount (10)
Integer :: displacement (10)



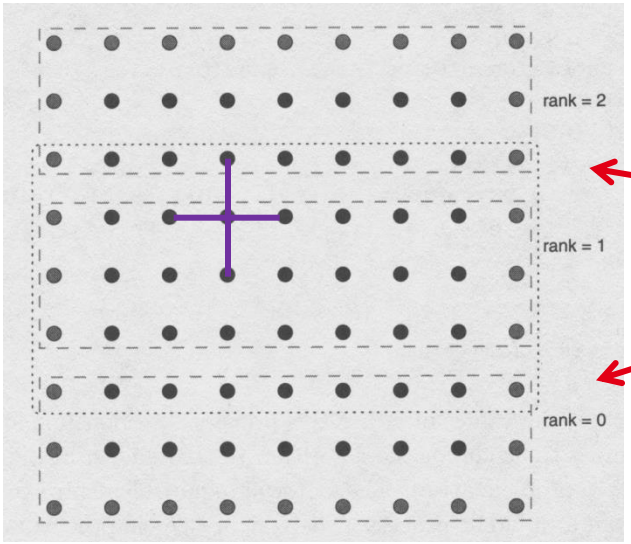
sendcounts = (/2, **8,** 5, 8, 5, 2, 2, 2, 2, 2/)

displacement = (/0, 2, 12, 17, 25, 30, 32, 34, 36, 38/)

sendcounts wird an neue Datenstruktur angepasst, Position 11 und 12 wird ausgelassen

displacement: hier ändert sich in diesem Beispiel nichts.

Parallele Bearbeitung einer Matrix I



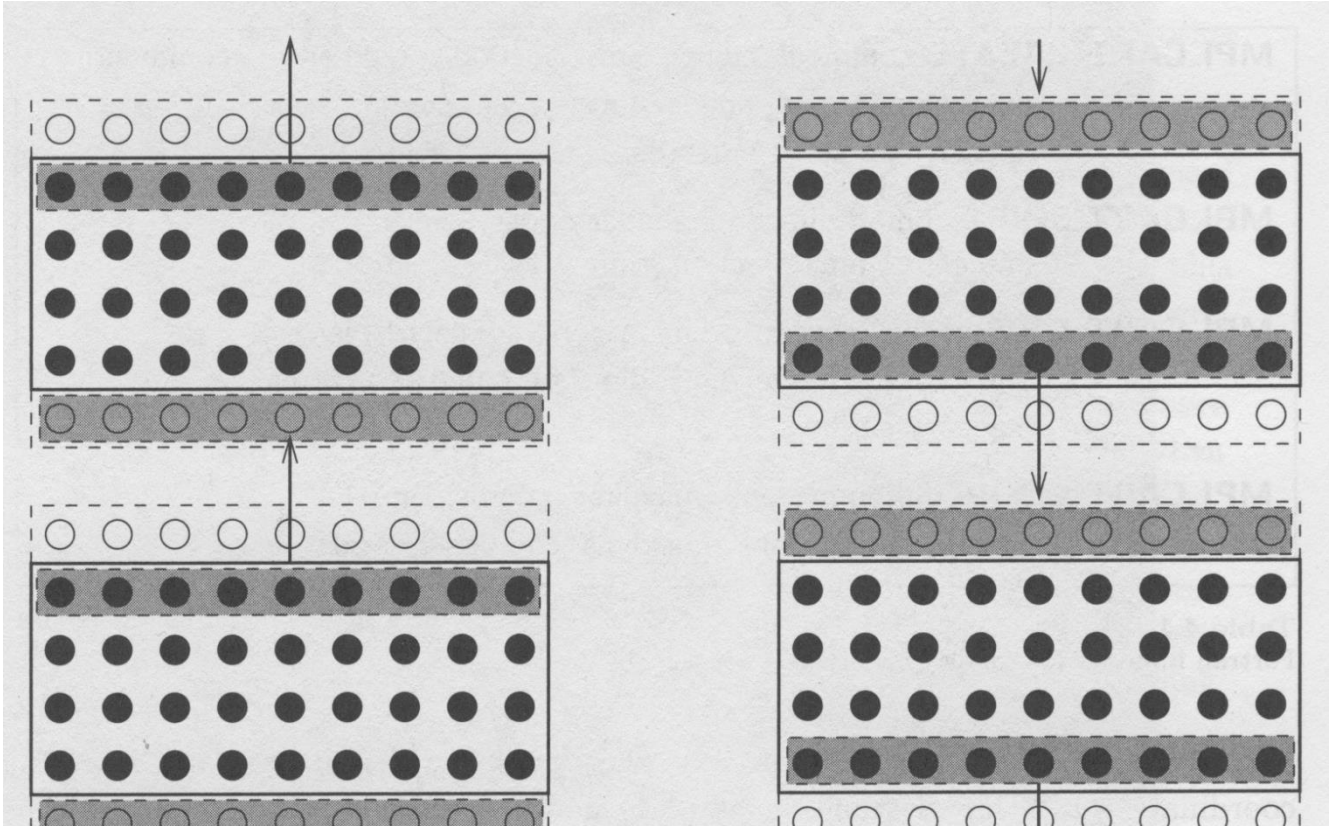
Überlappung der Teilmatrizen der einzelnen Prozesse

Berechnung der Poisson Gl.

auf *Star*

Quelle:
Gropp, Lusk & Skjellum
Using MPI

Parallele Bearbeitung einer Matrix II



Austausch
der Randstreifen
im Programmlauf

Quelle:
Gropp, Lusk & Skjellum
Using MPI

Danke das wars!