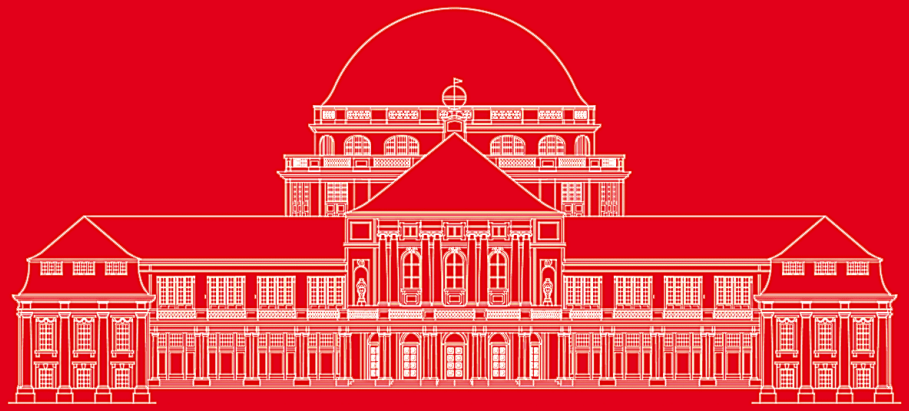
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart



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

MPI Einführung II:

- Send/Receive Syntax
- Broadcast
- Reduce Operation
- Barrier
- Scatter / Gather -> Matrixbehandlung

MPI Send/Receive Syntax I

MPI_SEND(Message, Count, Datatype, Dest, Tag, Comm, Ierror)

z.B:

Call MPI_SEND(temp, 1, MPI_Real, dest, tag, MPI_COMM_World, Ierror)

temp	Adresse des Sendepuffers; Real :: temp
1	Count – Anzahl der Elemente im Puffer
MPI_Real	Datentyp des gesendeten Elementes
dest	Angabe des Ranges des Zielprozesses; integer :: dest
tag	Nachrichtenkennung; integer :: tag
MPI_COMM_World	Kommunikator (Gruppe, Kontext)
Ierror	Fehlerstatus; integer :: Ierror

MPI Send/Receive Syntax II

MPI Datentypen in Anlehnung an Fortran

MPI Datentyp	FORTRAN Datentyp
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)

MPI Send/Receive Syntax III

MPI_RECV(Message, Count, Datatype, Source, Tag, Comm, status, Ierror)

Call MPI_RECV(temp, 1, MPI_Real, dest, tag, MPI_COMM_World, status, Ierror)

temp	Adresse des Sendepuffers; Real :: Vector
1	Count – Anzahl der Elemente im Puffer
MPI_Real	Datentyp des gesendeten Elementes
source	Angabe des Ranges des Sendeprozesses; integer :: source
tag	Nachrichtenkennung (Reihenfolge); integer :: tag
MPI_COMM_World	Kommunikator (Gruppe, Kontext)
status	Empfangsstaus der Nachricht (angekommen?); **integer status(MPI_STATUS_SIZE)**
Ierror	Fehlerstatus; integer :: Ierror

MPI Kollektive Operationen

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

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

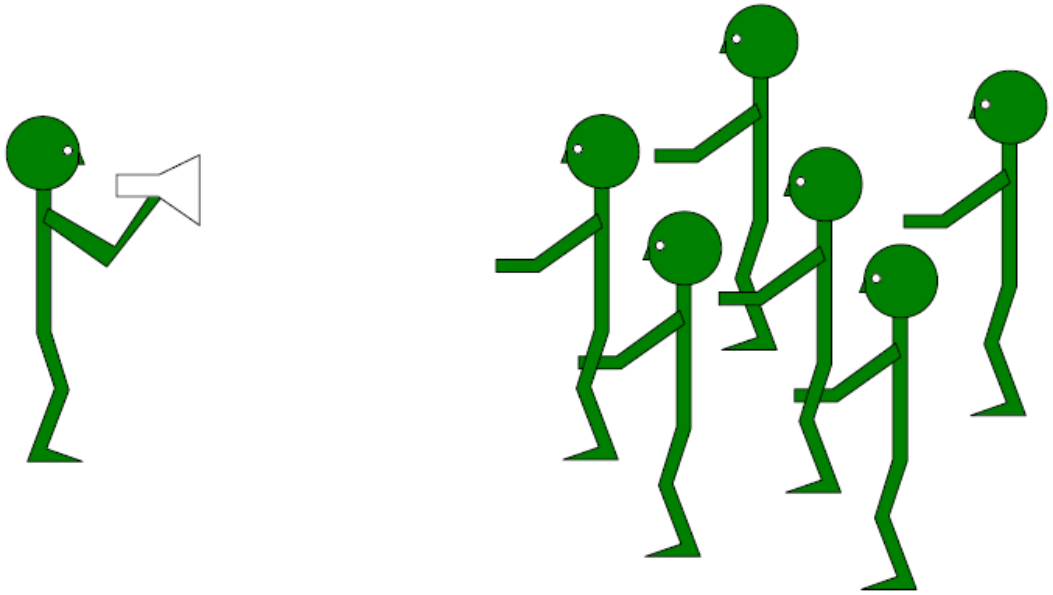
MPI_BROADCAST Eine Info an alle Prozesse versenden

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

MPI_SCATTER Teilarrays an Prozesse übertragen

MPI_GATHER Teilarrays zusammenführen

MPI Broadcast I



(Wolfgang Baumann ZIB, 2009;
Parallel Programming with MPI)

MPI Broadcast

Neben dem Versenden von Nachrichten zwischen einzelnen Prozesse mittels

Call MPI_SEND(temp, 1, MPI_Real, dest, tag, MPI_COMM_World, Ierror)

gibt es auch die Möglichkeit **eine Nachricht an alle anderen Prozesse** (z.B. in MPI_COMM_World) zu senden:

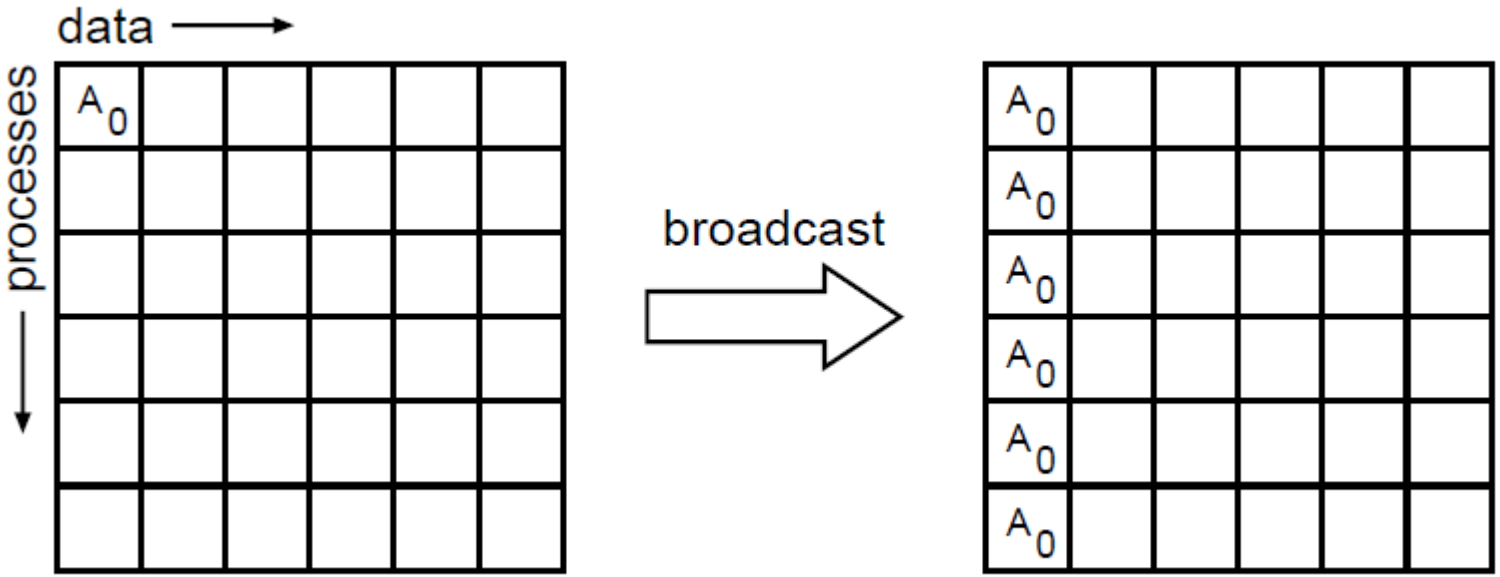
MPI_BCAST(Message, Count, Datatype, Root, Comm, Ierror)

Call MPI_BCAST(temp, 1, MPI_Real, source, MPI_COMM_World, Ierror)

Für Initialisierung oder zum Programmabbruch genutzt.

MPI Broadcast III

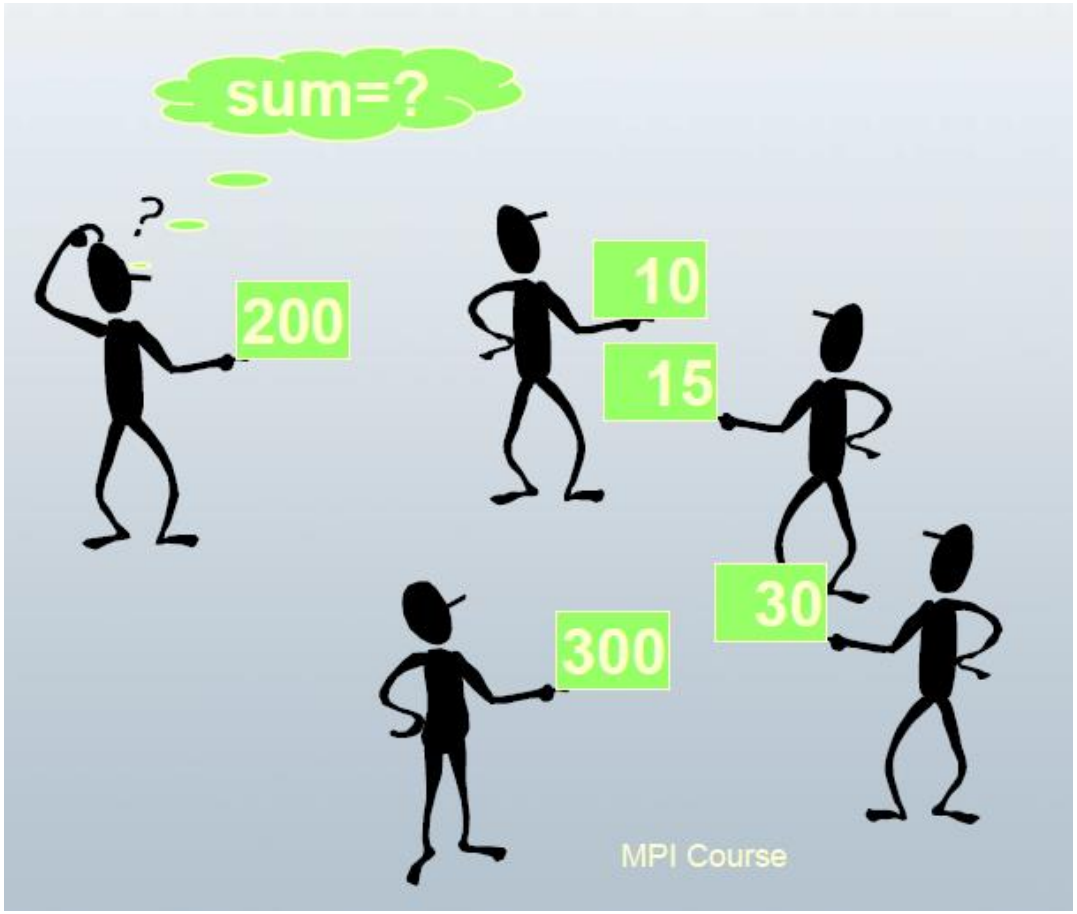
Verwendung zur Initialisierung



(William Gropp ANL, MPI Tutorial)

Es gibt eine interne Hirarchie der Zuteilung!

MPI Reduce I



DKRZ MPI Einführungs Kurs

MPI Reduce II

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

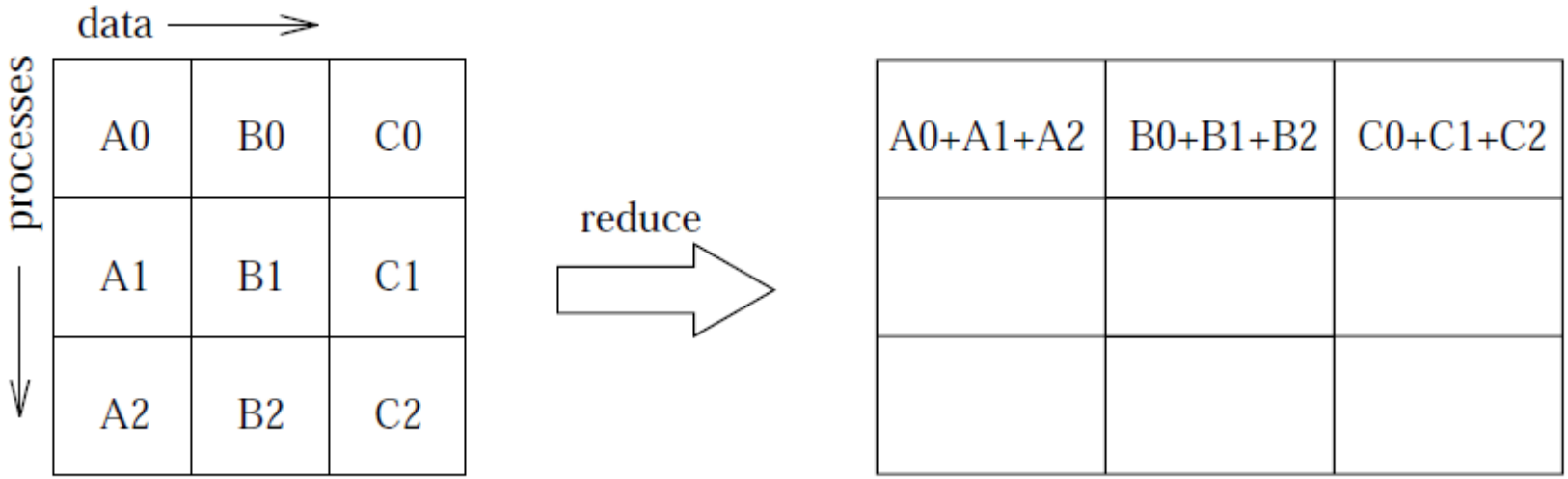
MPI_REDUCE(Operand, Result, Count,Datatype, Operation, Root, Comm, Ierror)

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 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

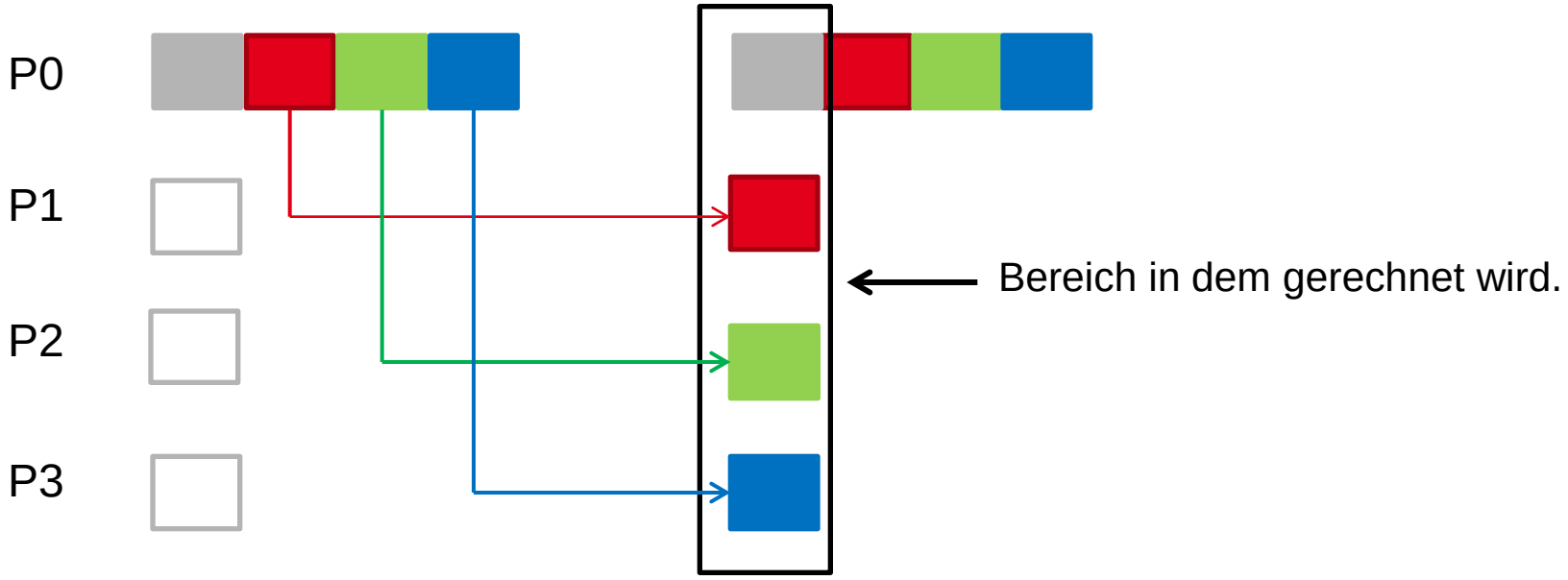
Eine Möglichkeit z.B. eine Anfangsbelegung auf die Teilarrays der Prozesse , zu übertragen bietet MPI_SCATTER:

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

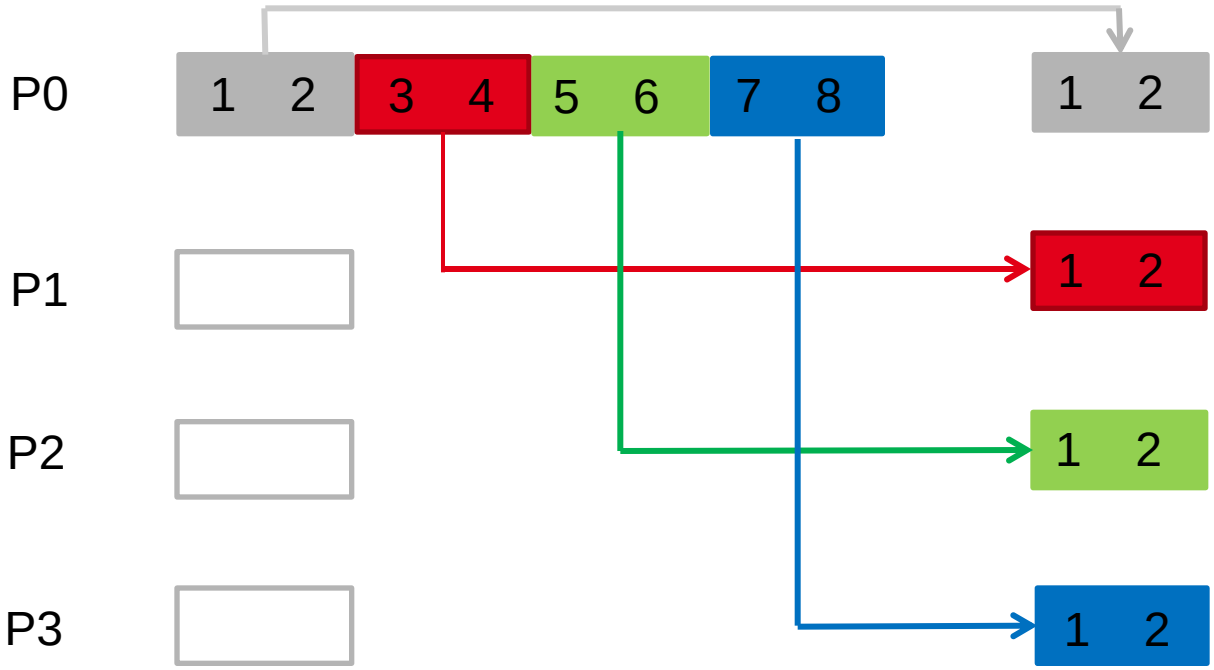
Call MPI_SCATTER(Send_Message, Send_Count, Send_Datatype,
Recv_Message, Recv_Count, Recv_Datatype,
0, MPI_COMM_World, Ierror)

MPI Scatter II

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



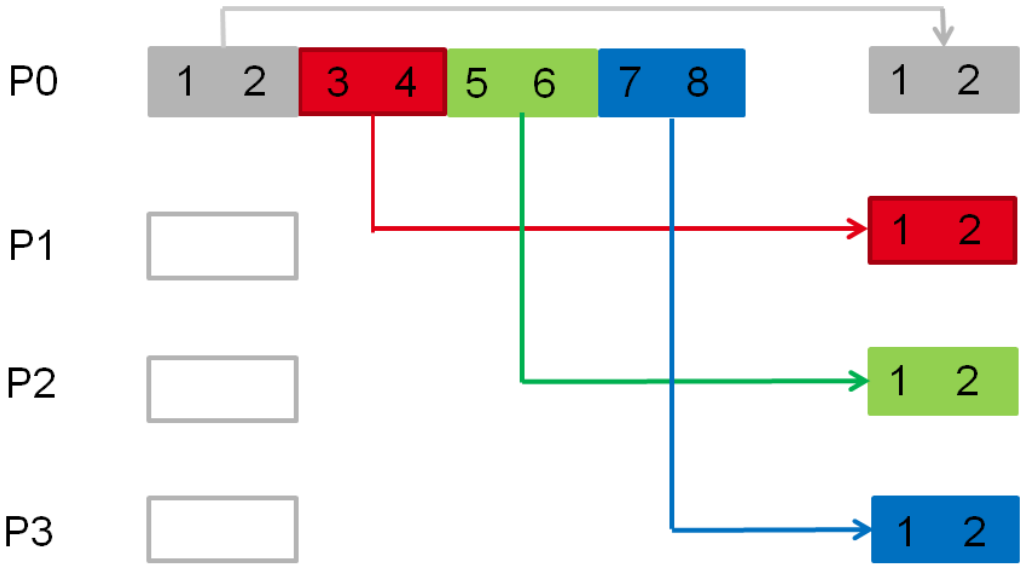
MPI Scatter: array1D(8) => 4 Teile chunk1D(2)



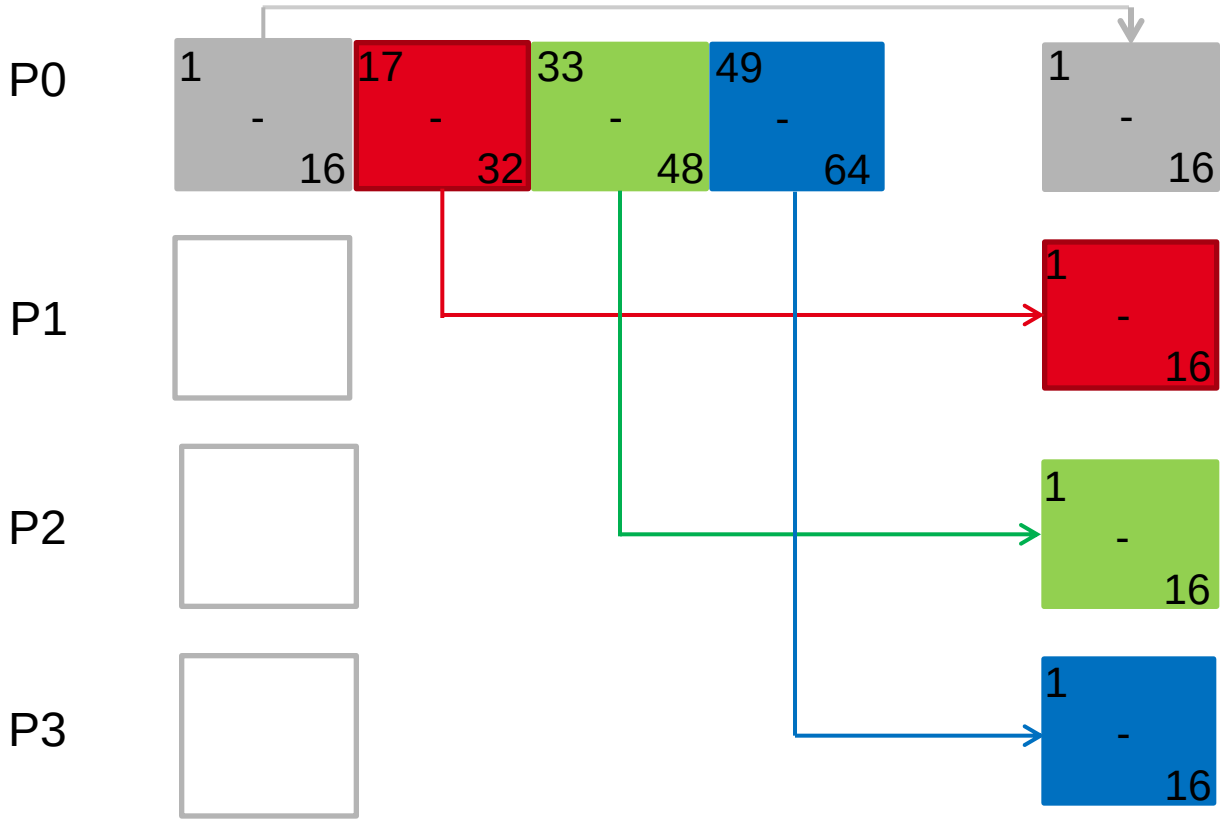
MPI Scatter:

```
integer, dimension(8) :: array1D      ! all the 1D data
integer, dimension(2) :: chunk1D      ! piece of 1D data each process works on


 ! int MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)

CALL MPI_SCATTER(array1D, 2, MPI_INTEGER, chunk1D, 2, MPI_INTEGER, &
0, MPI_COMM_WORLD, mpi_ierr)
```

Abbildung von Vektor
array1D auf 4 Teile chunk1D



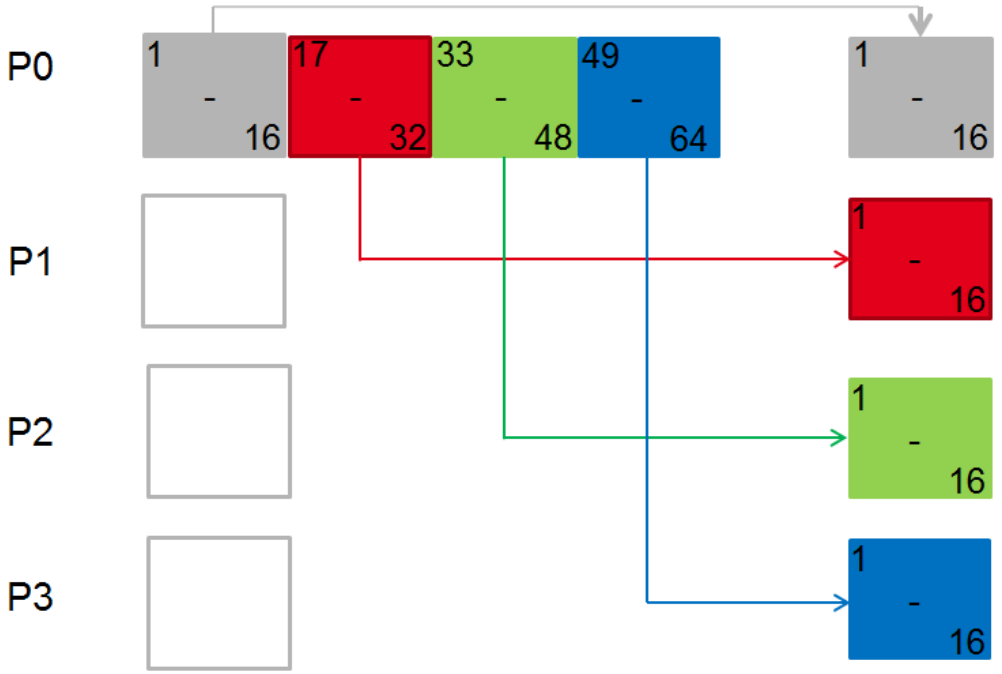
MPI Scatter: Beispiel array2D(8X8) => 4 Teile chunk2D(4X2)



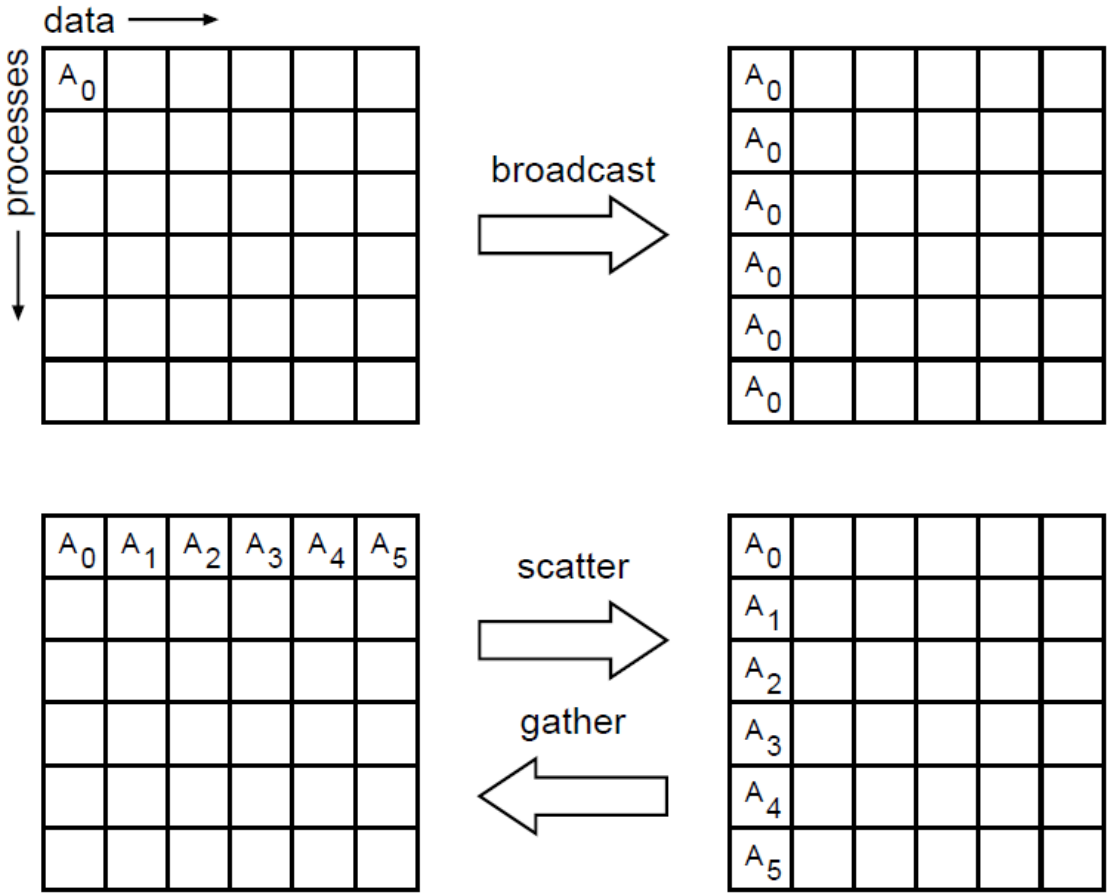
MPI Scatter:

```
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 Scatter-Gather



(William Gropp ANL, MPI Tutorial)

MPI Gather I

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

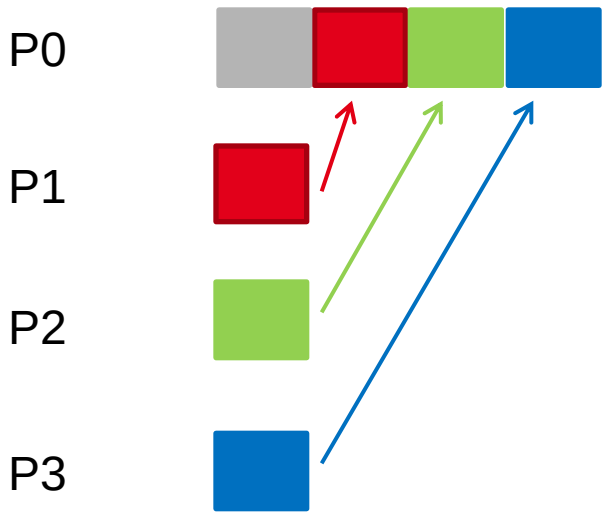
Syntax: MPI_Gather(Send_Message, Send_Count, Send_Datatype,
Recv_Message, Recv_Count, Recv_Datatype,
Root, Comm, Ierror)

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 Barrier I

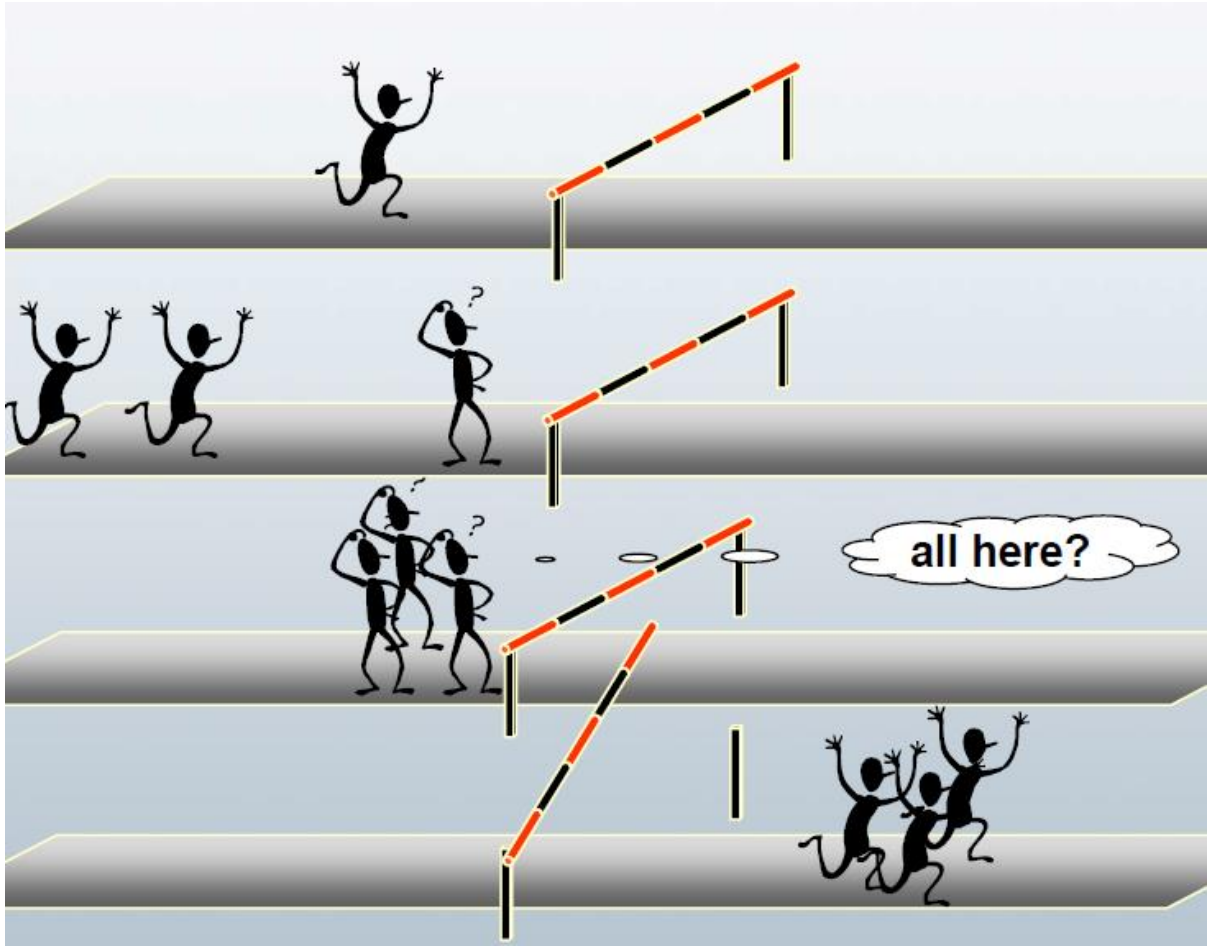
Der MPI_BARRIER Befehl wird zur Programmsteuerung eingesetzt.

Call MPI_BARRIER(MPI_COMM_World, Ierror)

Der MPI_Barrier Befehl erzwingt dass alle Prozesse den gleichen Punkt im Code erreicht haben bevor das Programm weiterläuft.

MPI Barrier II

DKRZ MPI Einführungs Kurs



MPI Barrier III

Der MPI_BARRIER Befehl wird vorranging zur Zeitmessung eingesetzt, z.B.

….

```
Call MPI_BARRIER(MPI_COMM_World, ierror)
t1 = MPI_WTIME()
….
Call MPI_BARRIER(MPI_COMM_World, ierror)
total_time = MPI_WTIME() – t1
```

Danke das wars!