Колективни операции

Увод в паралелното програмиране #5

Материалите са разработени в рамките на договор ДО-02-136/2008 с Фонд Научни Изследвания
Проект: IRC-CoSiM Ръководител на проекта: проф. дфзн Ана Пройкова, ФзФ на СУ
http://www.irc-cosim.uni-sofia.bg/

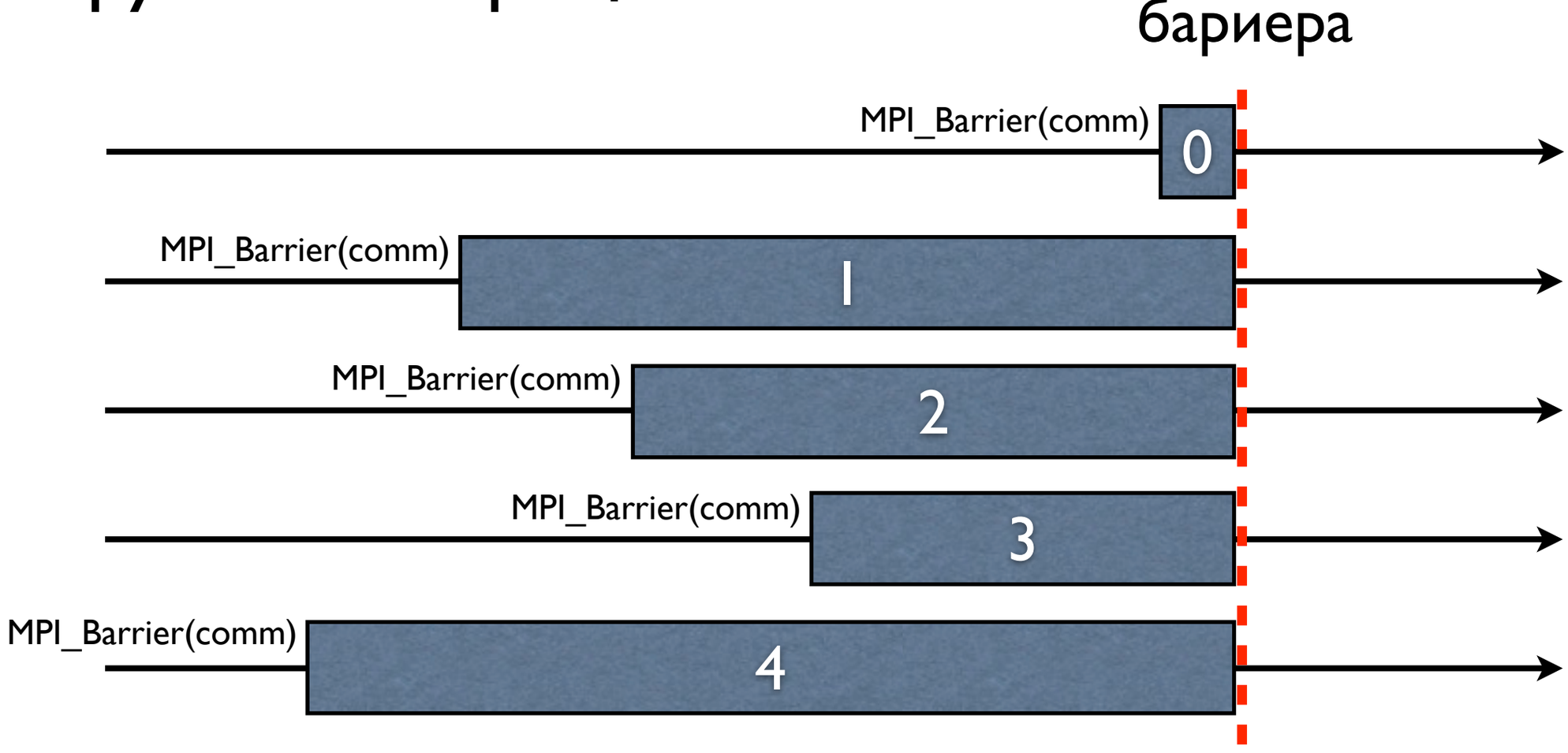
Общи положения

- Операции, в които участват всички процеси в групата на даден комуникатор
- Колективните операции са синхронни, блокиращи и приключват тогава, когато се изпълнят от всички процеси в групата
- Видове:
  - бариерна синхронизация
  - глобално разпращане
  - разпределяне на работата (scatter/gather)
  - глобална редукция

Бариерна синхронизация

MPI_Barrier(comm)

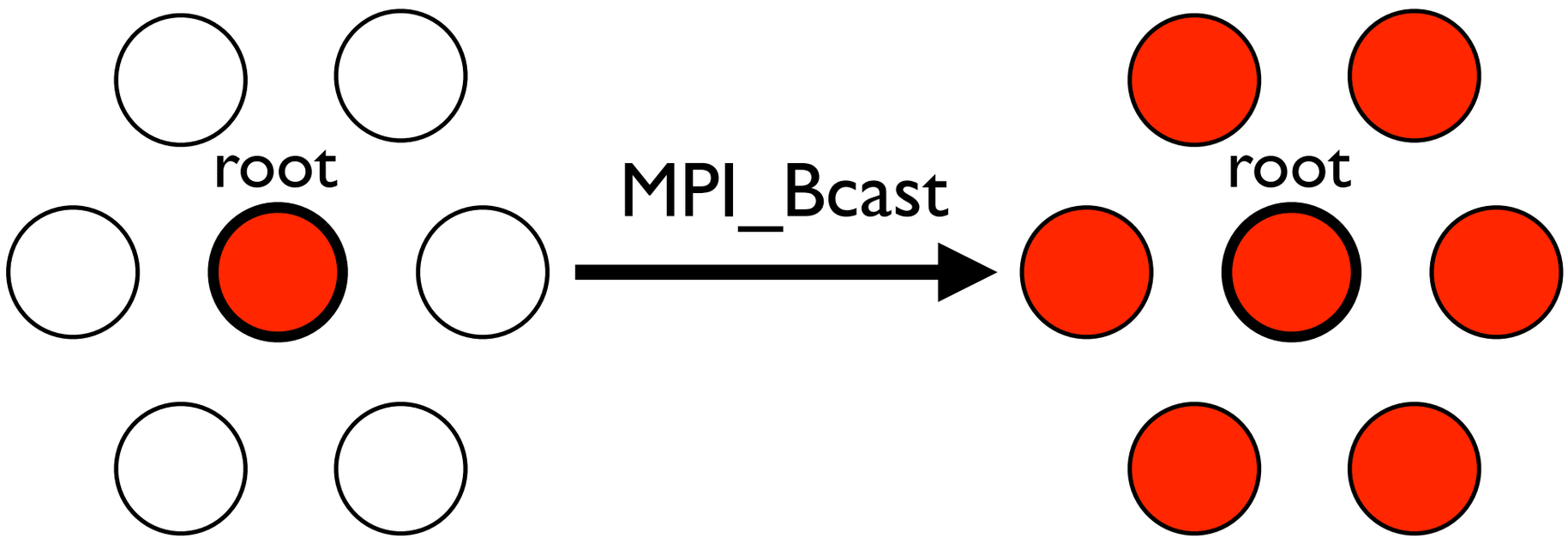
- Синхронизира във времето изпълнението на групата от процеси



Глобално разпръщане

MPI_Bcast (buf, count, dtype, root, comm)

- Копира съдържанието на *buf* от процеса с ранг *root* в буферите на останалите процеси в комуникатора *comm*



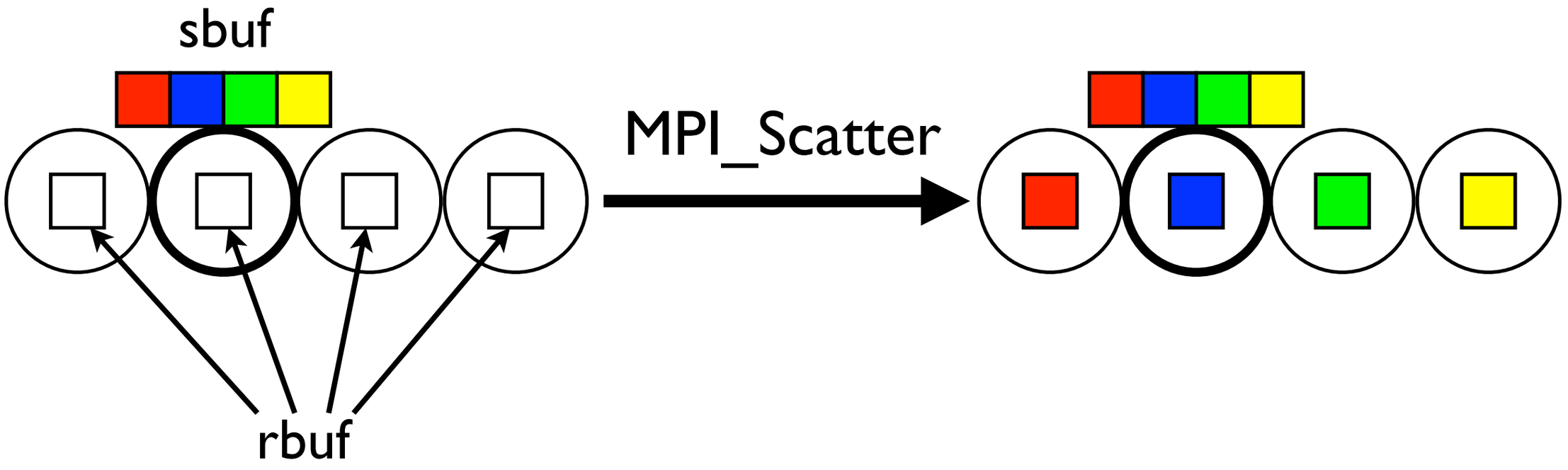
MPI_Bcast

- Процесът с ранг *root* е инициатор на разпращането и източник на информацията
- Всички останали процеси са цели и получатели на информацията
- Операцията приключва успешно след като всички процеси в комуникатора *comm* изпълнят *MPI_Bcast()*
- Всички процеси трябва да подадат едни и същи стойности на параметрите *root* и *comm*
- Горното важи за всички глобални операции

Разпределяне на работата

MPI_Scatter (sbuf, scount, sdtype, rbuf, rcount, rdtype, root, comm)

- Разпределя съдържанието на буфера *sbuf* в процеса с ранг *root* по буферите *rbuf* на всички процеси в комуникатора



MPI_Scatter

- Аргументите *sbuf*, *scount* и *sdtype* са съществени само в процес с ранг *root*
- Аргументите *rbuf*, *rcount* и *rdtype* са съществени във всички процеси, вкл. *root*
- На всеки процес се изпращат по *scount* на брой елемента, взети последователно от *sbuf*
- *sbuf* трябва да съдържа поне *scount* × *size* елемента, където *size* е броят на процесите в *comm*
- Важат същите правила за размерите на буферите и типа/броя елементи като при *MPI_Send*/*MPI_Recv*

Действие на място

- Операциите могат да работят на място като използват изходния буфер за входен и презаписват, ако е необходимо, предното съдържание с резултата от операцията
- *rbuf == MPI_IN_PLACE* само в процес с ранг *root*
- Действието на място се заключава в пълно бездействие в процес с ранг *root* – частта от *sbuf* не се премества в началото на буфера
- Използването на *MPI_IN_PLACE* във всеки друг процес, различен от *root*, води до грешка

MPI_IN_PLACE

- Без *MPI_IN_PLACE* (с отделен *rbuf* в *root*):

root изпраща на себе си парче от *sbuf* и го поставя в *rbuf*

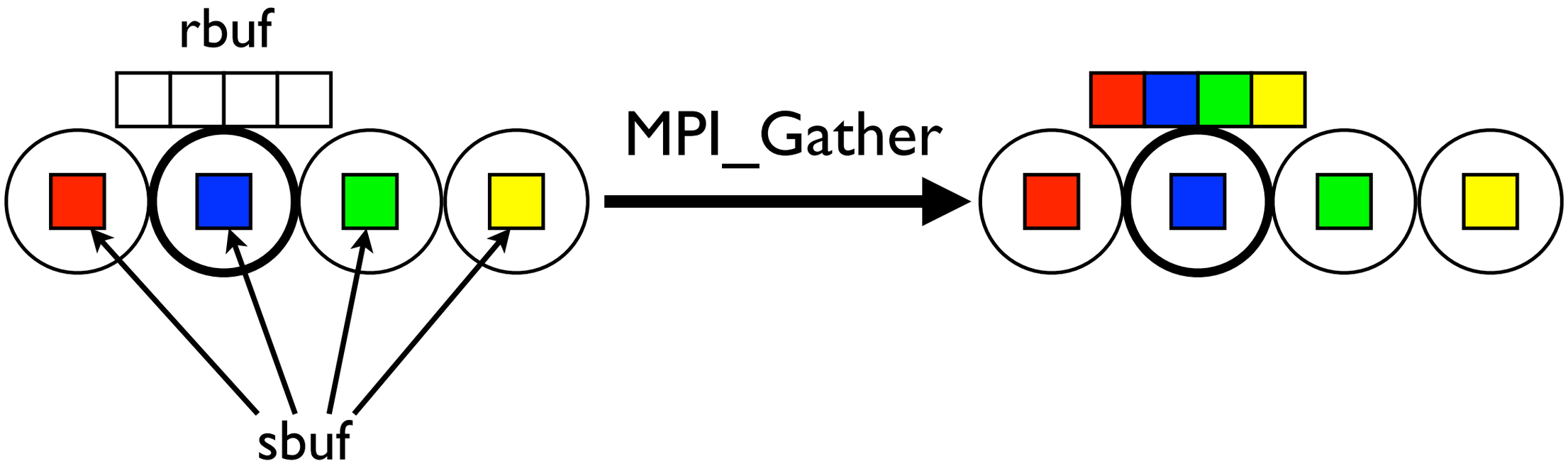
- С *MPI_IN_PLACE* (действие на място):

root използва направо своето парче от *sbuf*

MPI_Gather

MPI_Gather (sbuf, scount, sdtype, rbuf, rcount, rdtype, root, comm)

- Събира съдържанието на буферите *sbuf* на всички процеси в комуникатора в буфера *rbuf* на процеса с ранг *root*
- Обратната операция на *MPI_Scatter()*



Действие на място

- *sbuf* == *MPI_IN_PLACE* в процес с ранг *root*
- Частта от *rbuf*, която се пада на процес с ранг root, не се презаписва от *MPI_Gather()*

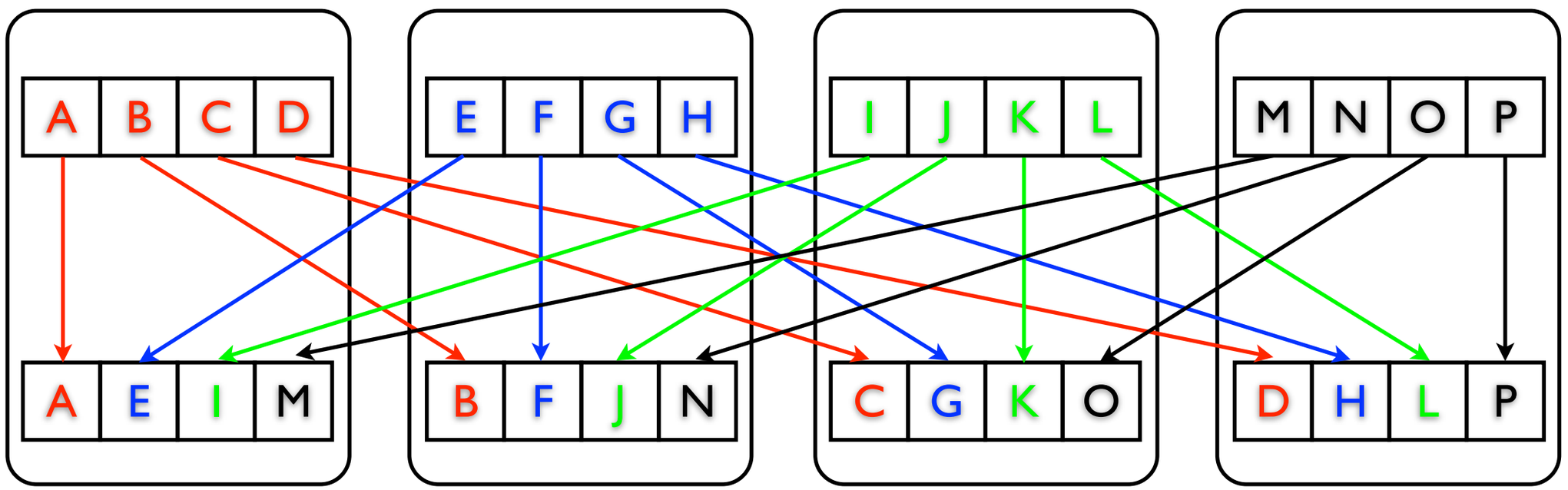
```
if (rank == root)
  MPI_Gather(MPI_IN_PLACE, count, dtype,
             rbuf, count, dtype,
             root, MPI_COMM_WORLD);
else
  MPI_Gather(sbuf, count, dtype,
             rbuf, count, dtype,
             root, MPI_COMM_WORLD);
```

- Параметрите в червено са несъществени в извикващия процес и се игнорират

MPI_Alltoall

MPI_Alltoall (sbuf, scount, sdtype, rbuf, rcount, rdtype, comm)

- Комбинирано разпределяне на съдържанието на всеки *sbuf* във всеки *rbuf*
- Няма отделен процес *root* – всички процеси изпращат и получават
- Обратната операция се постига с размяна на местата на *sbuf* и *rbuf*



MPI_Allgather

MPI_Allgather (sbuf, scount, sdtype, rbuf, rcount, rdtype, comm)

- Работи като комбинация на *MPI_Gather()* и последващ *MPI_Bcast()* на съдържанието на *rbuf*
- Няма отделен процес цел *root* – всички процеси получават съдържанието

Неравноделие

MPI_Scatterv (sbuf, scounts, displs, sdtype,
rbuf, rcount, rdtype, root, comm)

- *scounts* и *displs* са целочислени масиви от *size* на брой елемента (*size* е броят на процесите)
- На процес с ранг *i* се изпращат $count_i$ елемента от отместване $displs_i$
- Един и същ елемент от *sbuf* не трябва да бъде четен повече от един път – не трябва да има застъпване на отделните парчета
- Обратната операция е *MPI_Gatherv()*

Глобална редукция

- Прилага бинарен редукционен оператор ⊕ върху данни, разпределени по всички процеси в комуникатора:

$$R = S_0 \oplus S_1 \oplus S_2 \oplus S_3 \oplus \ldots$$

- Операторът ⊕ може да бъде:
  - аритметичен (събиране, умножение)
  - логически и побитови операции (AND, OR, XOR)
  - намиране на минимален/максимален елемент
  - дефинирана от потребителя операция
- Предефинираните оператори са асоциативни и комутативни
- Аргументите и резултатът най-общо са вектори

Предефинирани оператори

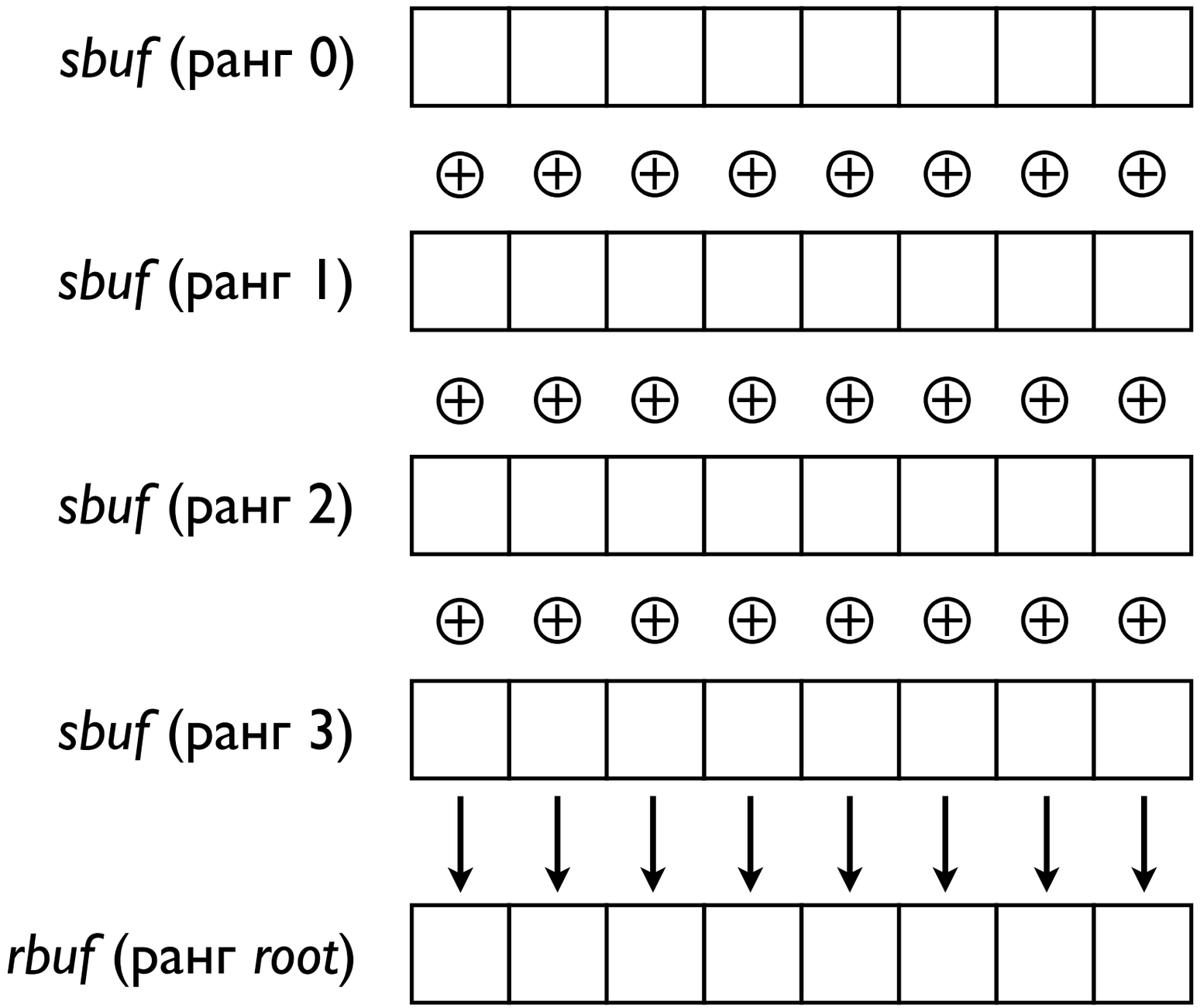
Манипулатор на оператора	Значение
MPI_MAX	Стойност на максималния елемент
MPI_MIN	Стойност на минималния елемент
MPI_SUM	Сума от елементите
MPI_PROD	Произведение на елементите
MPI_LAND	Логическо И между елементите
MPI_BAND	Побитово И между елементите
MPI_LOR	Логическо ИЛИ между елементите
MPI_BOR	Побитово ИЛИ между елементите
MPI_LXOR	Логическо изключващо ИЛИ между елементите
MPI_BXOR	Побитово изключващо ИЛИ между елементите
MPI_MAXLOC	Ранг на процеса с максималния по стойност елемент
MPI_MINLOC	Ранг на процеса с минималния по стойност елемент

Редукция

MPI_Reduce (sbuf, rbuf, count, dtype, op, root, comm)

- *op* е манипулатор на предефиниран оператор или оператор, дефиниран от потребителя
- *rbuf* има смисъл само в процес с ранг *root*
- Ако *sbuf* == *MPI_IN_PLACE*, стойността в процес с ранг *root* се взима от *rbuf*, след което съдържанието на последния се заменя със стойността на резултата от операцията

Редукция в картинки



Потребителски оператори

```
MPI_Op_create(func, commute, op)
MPI_Op_free(op)
```

- Регистрира потребителската функция *func* като редукционен оператор и връща манипулатор в *op*
- *commute* указва дали операторът е комутативен
- Операторът трябва задължително да е асоциативен
- Когато операторът не е комутативен, редукцията се изчислява в каноничен ред, като се прилага върху данните в процеси с последователни рангове, което води до известно забавяне
- Разрегистрирането на оператора става с *MPI_Op_free()*

Потребителски оператори

- Прототип на *func* в C/C++:

```
void func (void *invec, void *inoutvec,
           int *count, MPI_Datatype *dtype)
```

- Fortran:

```
SUBROUTINE FUNC (INVEC(*), INOUTVEC(*),
                 COUNT, DTYPE)
<тип> INVEC(COUNT), INOUTVEC(COUNT)
INTEGER COUNT, DTYPE
```

- Действието се свежда до:

  $inoutvec_i = invec_i \oplus inoutvec_i$

- *func* сигнализира грешки по време на изпълнение посредством извикване на *MPI_Abort()*

Варианти на редукцията

- *MPI_Allreduce* – резултатът се получава във всички процеси
- *MPI_Reduce_scatter* – върху резултата се прилага операция scatter
- *MPI_Scan* – всеки процес получава резултата от частичната редукция:

  $R_0 = S_0$

  $R_1 = S_0 \oplus S_1$

  $R_2 = S_0 \oplus S_1 \oplus S_2$

  $R_3 = S_0 \oplus S_1 \oplus S_2 \oplus S_3$

  Само процесът с най-висок ранг получава резултата от пълната редукция