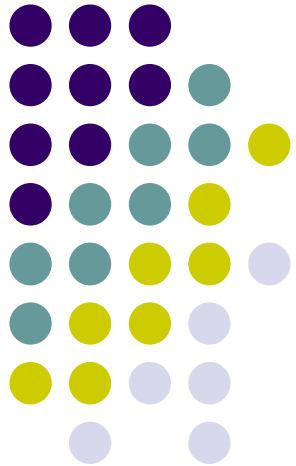
Supercon
MPIプログラミング
自習用資料



MPIの基本中の基本: メッセージの送信・受信

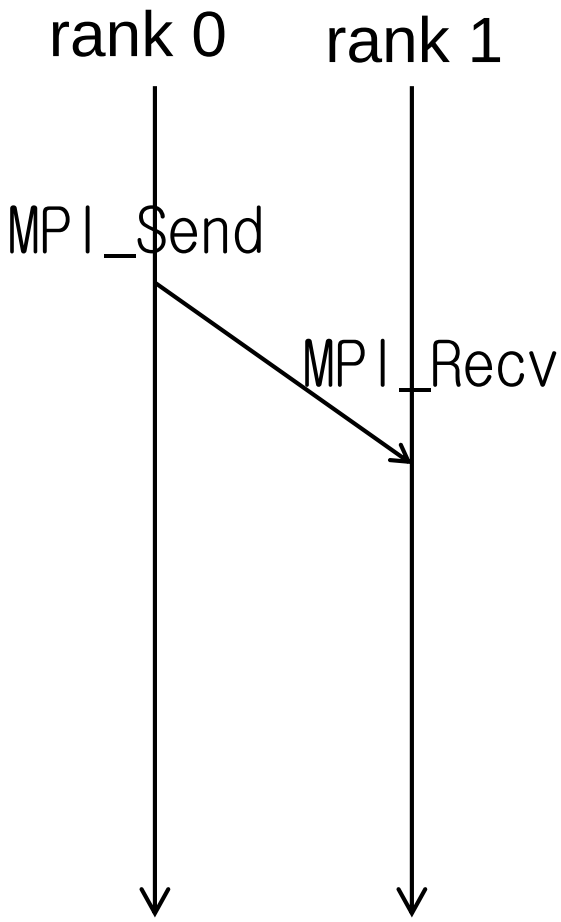
rank 0からrank1へ, int a[16]の中身を送りたい場合

- rank0側で

```
MPI_Send(a, 16, MPI_INT, 1,
    100, MPI_COMM_WORLD);
```

- rank1側で

```
MPI_Recv(b, 16, MPI_INT, 0,
    100, MPI_COMM_WORLD, &stat);
```



MPI_Send

```
MPI_Send(a, 16, MPI_INT, 1, 100, MPI_COMM_WORLD);
```

- a: メッセージとして送りたいメモリ領域の先頭アドレス
- 16: 送りたいデータ個数
- MPI_INT: 送りたいデータ型
  - 他にはMPI_CHAR, MPI_LONG. MPI_DOUBLE, MPI_BYTE・・・
- 1: メッセージの宛先プロセスのrank
- 100: メッセージにつけるタグ(整数)
- MPI_COMM_WORLD: コミュニケータ

MPI_Recv

```
MPI_Status stat;
MPI_Recv(b, 16, MPI_INT, 0, 100, MPI_COMM_WORLD, &stat);
```

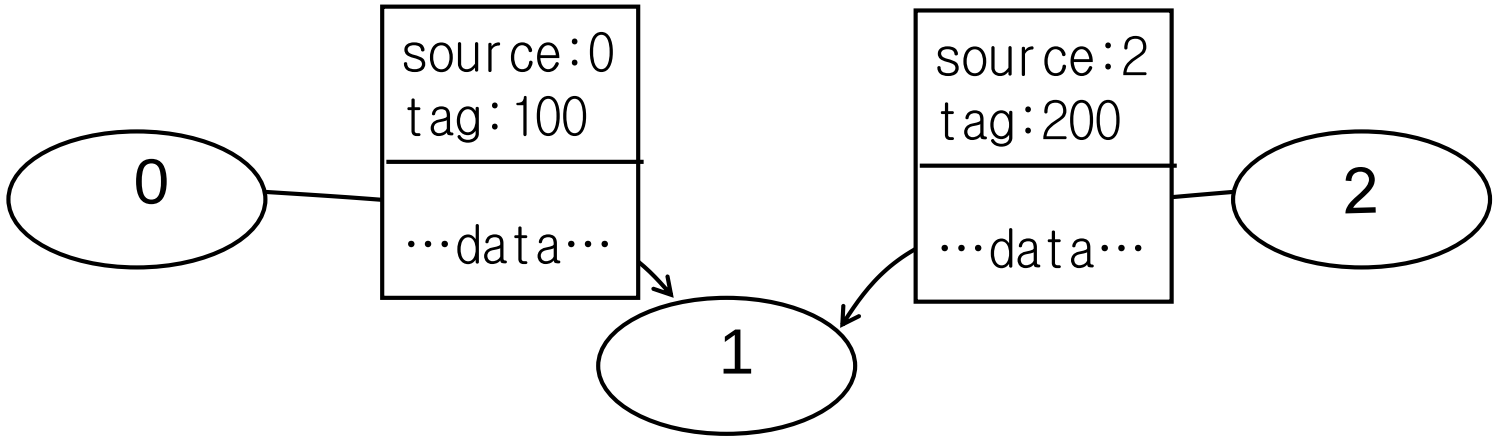
- b: メッセージを受け取るメモリ領域の先頭アドレス
  - 十分な領域を確保しておくこと
- 16: 受け取るデータ個数
- MPI_INT: 受け取るデータ型
- 0: 受け取りたいメッセージの送信元プロセスのrank
- 100: 受け取りたいメッセージのタグ．ユーザが決める整数
  - MPI_Sendで指定したものと同じなら受け取れる
- MPI_COMM_WORLD: コミュニケータ
- &stat: メッセージに関する補足情報が受け取れる

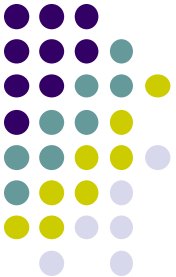
MPI_Recvを呼ぶと，メッセージが到着するまで待たされる（ブロッキング）

MPI_Recvのマッチング処理

受信側には複数メッセージがやってくるかも → 受け取りたい条件を指定する

- 受け取りたい送信元を指定するか，MPI_ANY_SOURCE (誰からでもよい)
- 受け取りたいタグを指定するか，MPI_ANY_TAG(どのタグでもよい)





集団通信

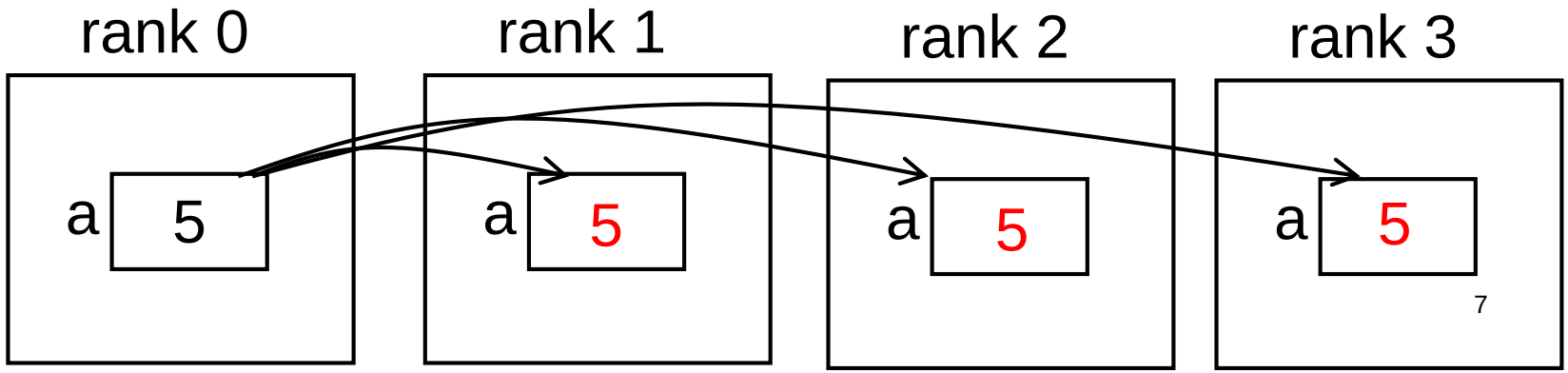
- 一対一通信: MPI_Send対MPI_Recv
  - これがあれば、原理的にはなんでも書ける
- 集団通信とは，多数プロセスを巻き込んだ通信
  - Reduce, Bcast, Gather, Scatter, Barrier・・・
  - 一対一の組み合わせでも実現できるが，専用関数の方が早い・速い
    - プログラムが楽
    - プロセスの木構造・binary exchangeなどの効率的アルゴリズムが使われている(はず)

集団通信:MPI_Bcast

- 例:rank 0のプロセスが持っているint aの値を全プロセスに知らせたい(broadcast処理)

```
MPI_Bcast(&a, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

  - 全プロセスがMPI_Bcastを呼び出す必要
  - この結果，全プロセスの領域aに結果が格納される
  - 第一引数はrootプロセス(ここではrank 0)では入力，それ以外のプロセスでは出力として扱われる

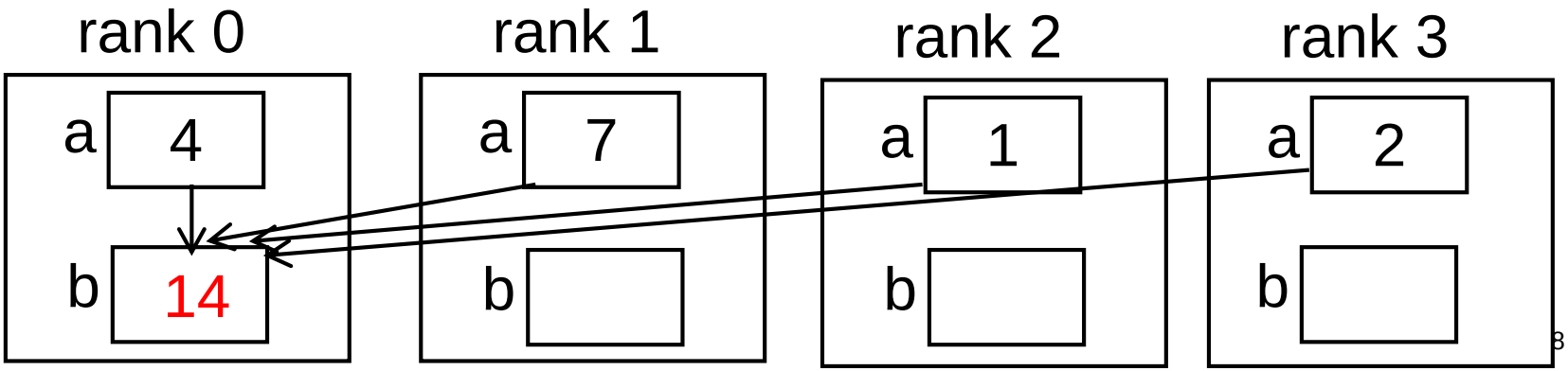


集団通信:MPI_Reduce

- 例: 全プロセスのint aの合計を求めたい (reduction処理)

```
MPI_Reduce(&a, &b, 1, MPI_INT, MPI_SUM, 0,
           MPI_COMM_WORLD);
```

  - 全プロセスがMPI_Reduce()を呼び出す必要
  - この結果, rank 0の領域bに合計(SUM)が格納される
  - 演算は他に, MPI_PROD(積), MPI_MAX, MPI_MIN, MPI_LAND(論理積)など

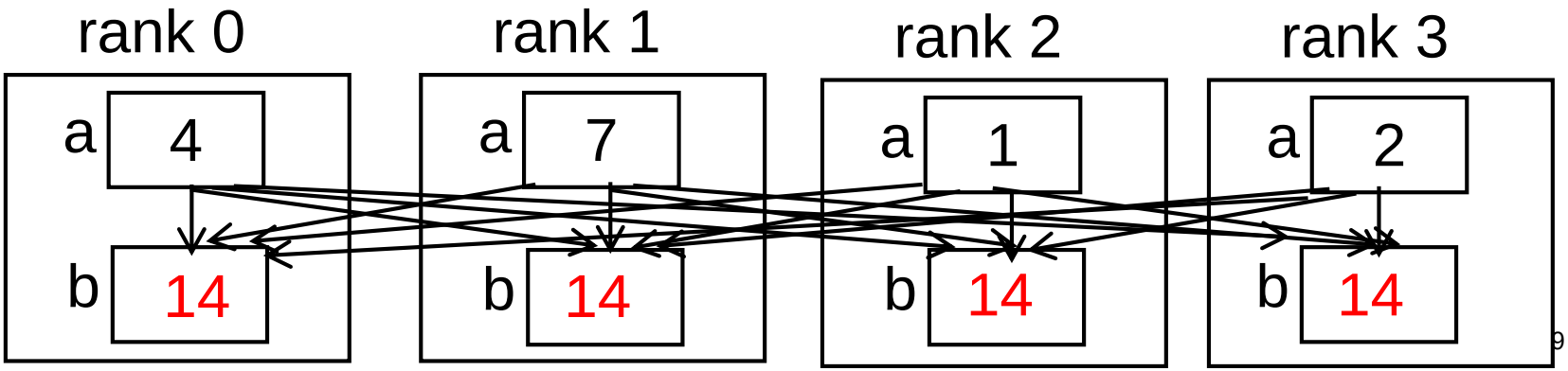


集団通信:MPI_Allreduce

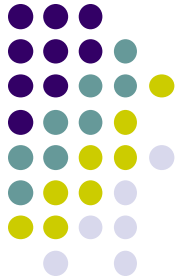
- Reduction処理を行い、その結果を全プロセスが知りたい

```
MPI_Allreduce(&a, &b, 1, MPI_INT, MPI_SUM,
              MPI_COMM_WORLD);
```

  - この結果，全員の領域bに合計(SUM)が格納される



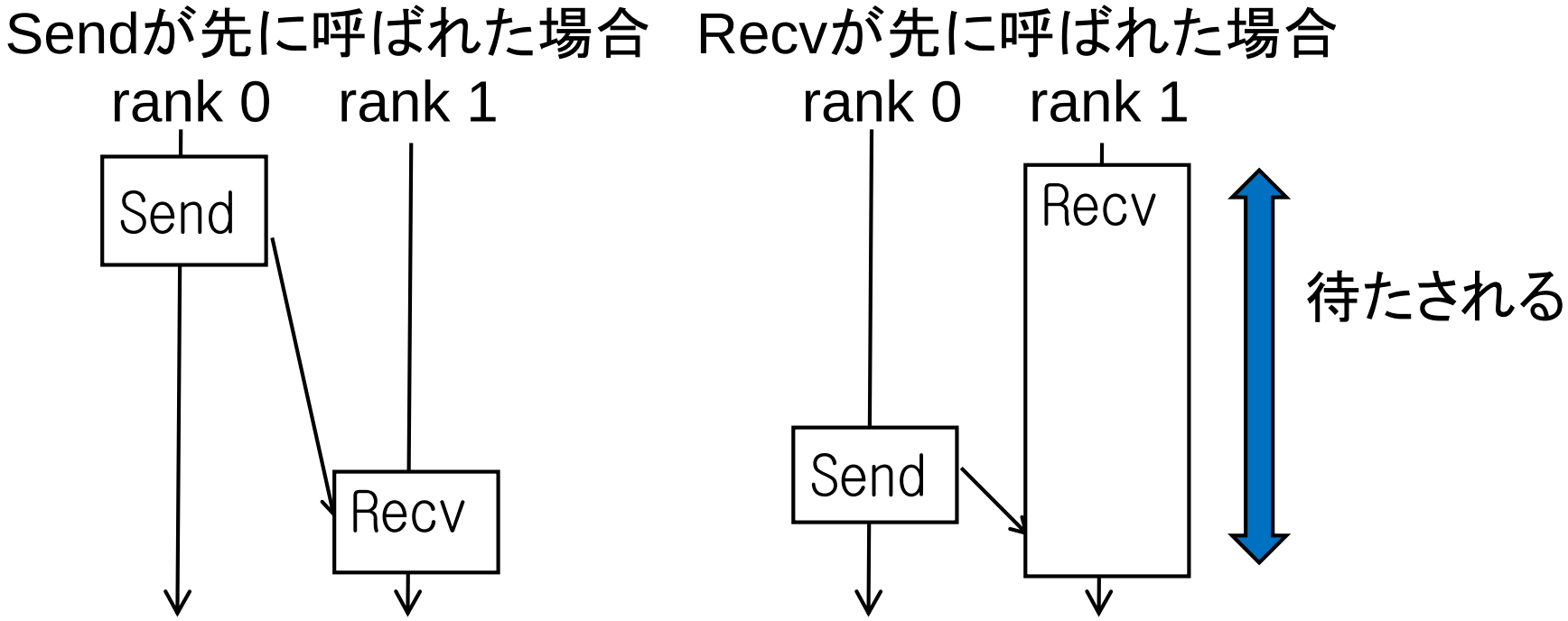
集団通信: MPI_Barrier



- 全プロセスで足並みをそろえる。バリア同期と呼ぶ

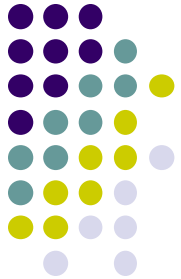
```
MPI_Barrier(MPI_COMM_WORLD);
```

ふつうの通信の問題



- MPI_Recvは、メッセージが届くまで返ってこない⇒その間プロセスはなにもできない
- MPI_Sendも、通信相手のMPI_Recv終了まで待たされる「ことがある」

ノンブロッキング通信とは



- データの送受信の指示だけまず行い、その完了を待たないこと
  - 送信、受信とも、ノンブロッキング用MPI関数が存在
  - その後、他の処理を行うことができる
- 後で別途、完了の確認を行う必要がある

ノンブロッキング(non-blocking)通信: Recvの場合

```
MPI_Status stat;
MPI_Recv(buf, n, type, src, tag, comm, &stat);
```

```
MPI_Status stat;
MPI_Request req;
MPI_Irecv(buf, n, type, src, tag, comm, &req);   ←受信開始
      :  別の仕事をしてよい
MPI_Wait(&req, &stat);     ←完了待ち
```

MPI_Irecv: 受信を開始するが，すぐ終了

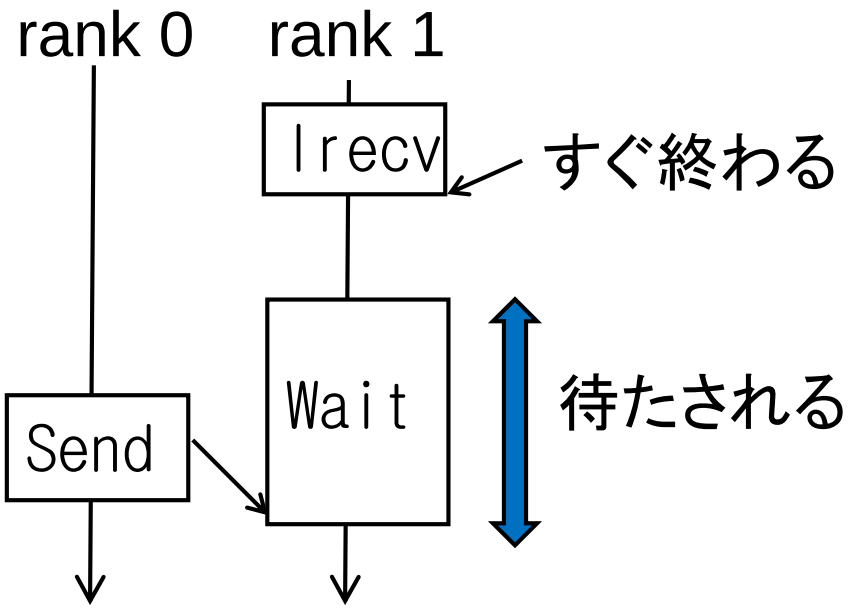
MPI_Waitを行うと受信を待つ → その終了後，メッセージを利用可能

IrecvとWaitの間に別の仕事をすることができる

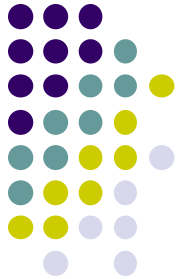
MPI_Requestは、通信を表す「チケット」のようなもの

MPI_Irecvの動き

- Irecv自身はすぐ終わる(non-blocking)
- が、データを使えるのはWait以降



ノンブロッキング通信: Sendの場合



```
MPI_Send(buf, n, type, dest, tag, comm);
```

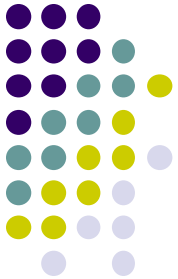
```
MPI_Status stat;
MPI_Request req;
MPI_Isend(buf, n, type, dest, tag, comm, &req);
MPI_Wait(&req, &stat);
```

MPI_Isend: 送信を開始するが，すぐ終了

MPI_Waitを行うと送信完了を待つ

注：MPI_Waitが終了するまで，buf領域を他目的に使ってはいけない

MPI_Waitの仲間



MPI_Wait(&req, &stat); → reqに対応する通信を待つ

MPI_Status stats[…]; MPI_Request reqs[…];
MPI_Waitall(n, reqs, stats); → reqs全部を待つ
MPI_Waitany(n, reqs, &idx, &stat); → reqsのどれかを待つ

MPI_Test(&req, &flag, &stat);
→ MPI_Waitに似るが, すぐ終了. 通信完了していればflagに0以外が帰る.
MPI_Testall, MPI_Testanyあり