2. メッセージ通信計算

2.1 メッセージ通信プログラミングの基本

プログラミングの選択肢

- 特別な並列プログラミング言語を設計する。
  - occam (Inmos, 1984, 1986)
- 既存の逐次言語の文法／予約語をメッセージ通信を処理できるように拡張する。
- 既存の逐次言語を用い、メッセージ通信のための拡張手続のライブラリを用意する。
  - どのプロセスを実行するのか
  - メッセージ通信のタイミング、中身を明示的に指定する必要がある。

- 基本的な機能
  - 異なるコンピュータで実行する別のプロセスを生成する
  - メッセージを送受信する

2.1 メッセージ通信プログラミングの基本

- プロセス
  - 処理の単位
  - 一つのプロセッサに複数のプロセスが割当てられることもある
- プロセスの生成
  - 静的プロセス生成
    - 全てのプロセスは実行前に指定され、一定数のプロセスを実行する
  - 動的プロセス生成
    - プロセス生成のための構文あるいはライブラリ／システムコールを用いる
    - プロセスの破壊も可能
    - プロセス生成時に大きなオーバーヘッドが発生する
    - プロセスのコードは実行前にコンパイルしておく必要がある

spawn(name_of_process);

PVMで可能．MPIの最近のバージョンでは可能

2.1 メッセージ通信プログラミングの基本

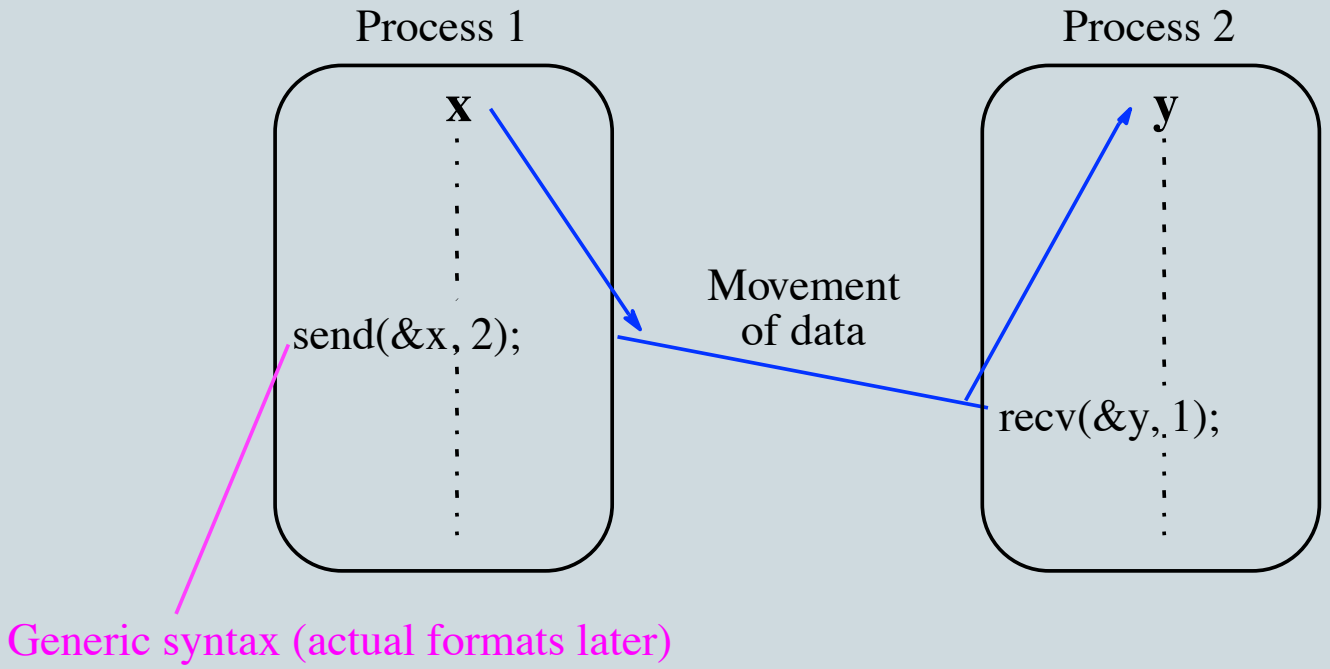
メッセージ通信ルーチン

- 基本送受信ルーチン
  - send(parameter_list)
  - メッセージ発信元プロセス中にある
  - recv(parameter_list)
  - 送信されたメッセージを受け取る宛先プロセス内に配置される

例:

- 送信元プロセス: send(&x, destination_id);
- 宛先プロセス: recv(&y, source_id);
  - パラメータの順番はシステム依存
  - xに送信データが予め設定されている必要がある
  - xとyは同じデータ型で同じサイズである必要がある

メッセージ送受信


Process 1
Process 2
x
y
send(&x, 2);
Movement
of data
recv(&y, 1);
Generic syntax (actual formats later)

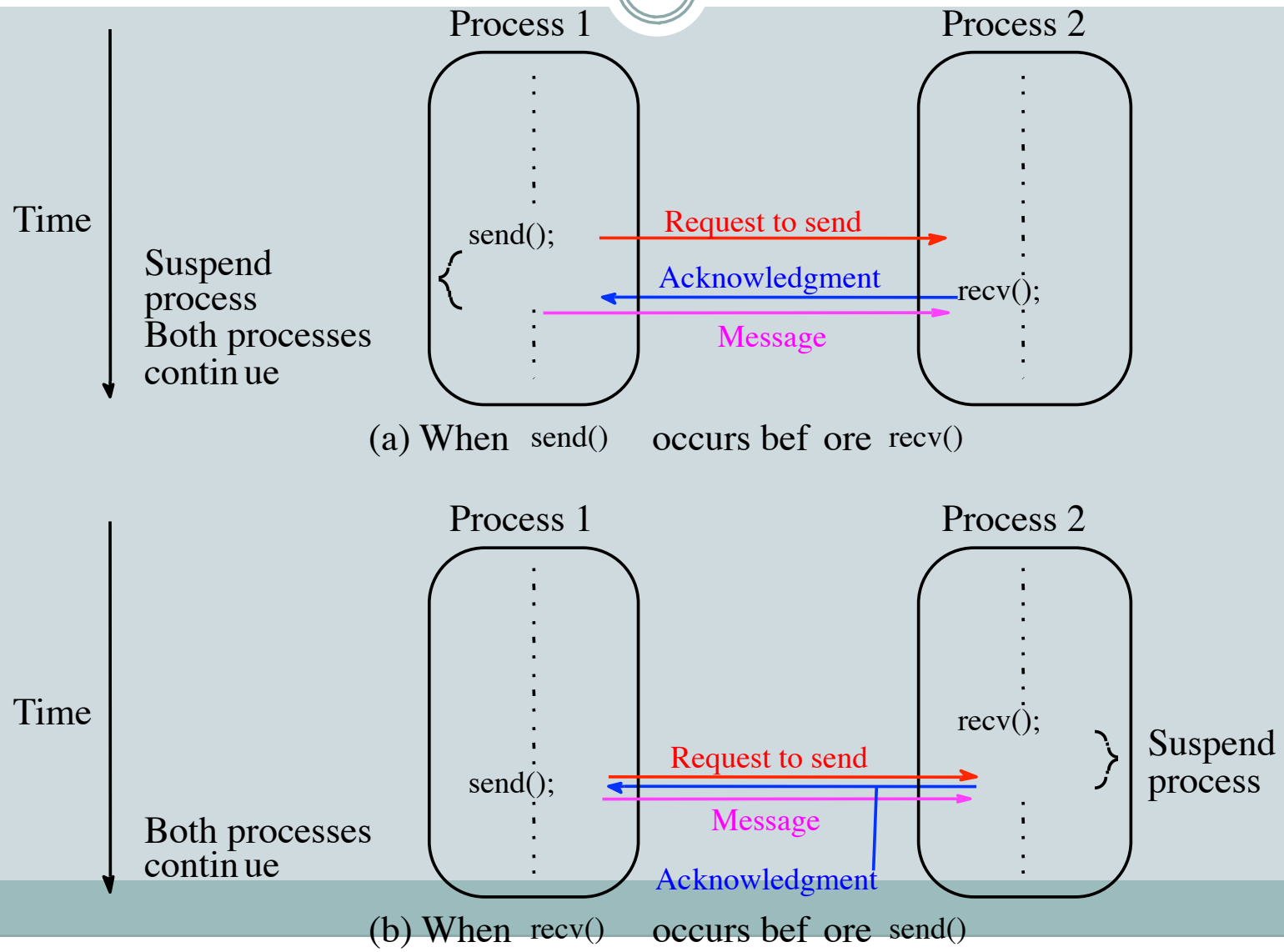
2.1 メッセージ通信プログラミングの基本

メッセージ通信ルーチン

- 同期式メッセージ通信
  - メッセージ転送が完了した時点でリターンするルーチン
- 同期式送信ルーチン
  - メッセージ全体が受信プロセスに受け取られまで待つ
- 同期式受信ルーチン
  - 予期するメッセージが到着するまで待つ
- データ転送とプロセス間の同期の二つの役割がある

- 三段階プロトコル
  - 発信元プロセスが「送信リクエスト」メッセージを送付する
  - 受信者は「了解」メッセージを送り返す
  - 実際のデータが送信される
  - タイミングによって、送信者あるいは受信者が一時待たされる

3-wayプロトコルによる同期式通信

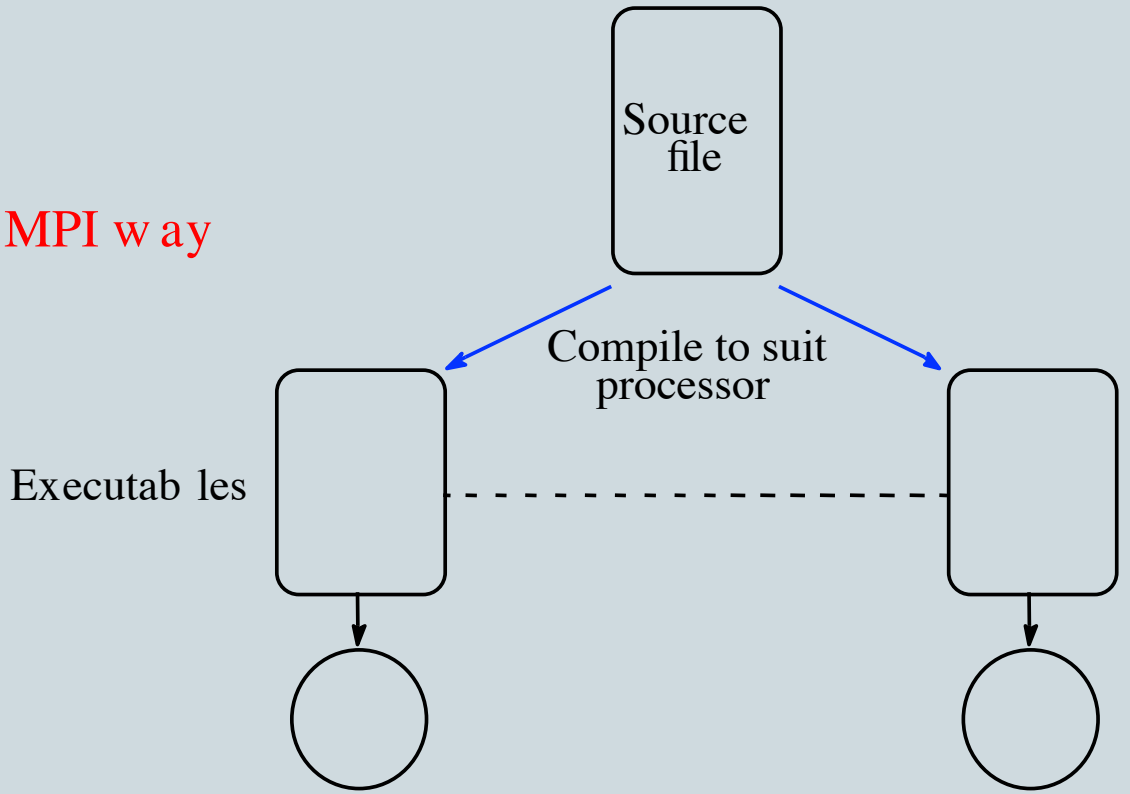


2.1 メッセージ通信プログラミングの基本

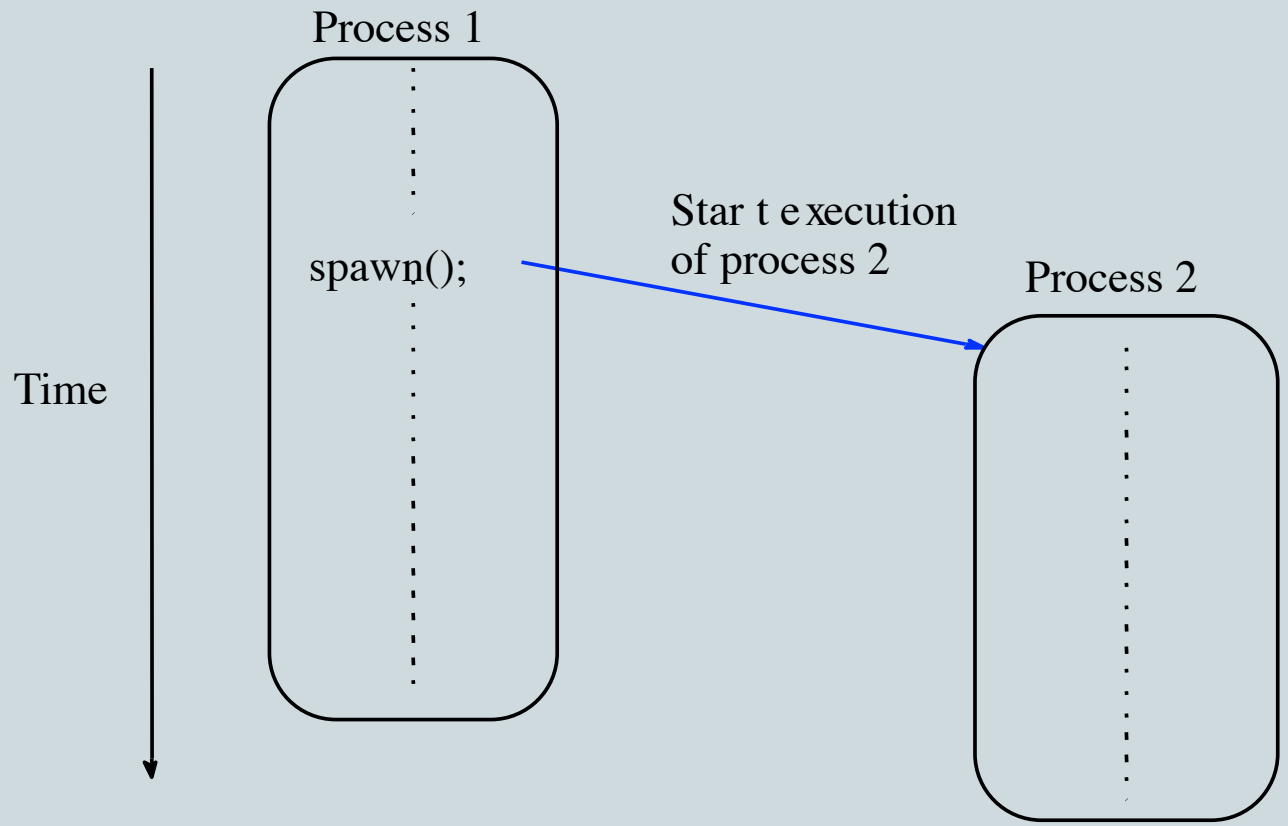
- プログラムモデル
  - SPMD(Single Program Multiple Data)
    - 複数の異なるプロセスを一つのプログラムとして記述
  - MPMD(Multiple Program Multiple Data)
    - プロセス毎にプログラムを記述

SPMD

Basic MPI w ay



MPMD

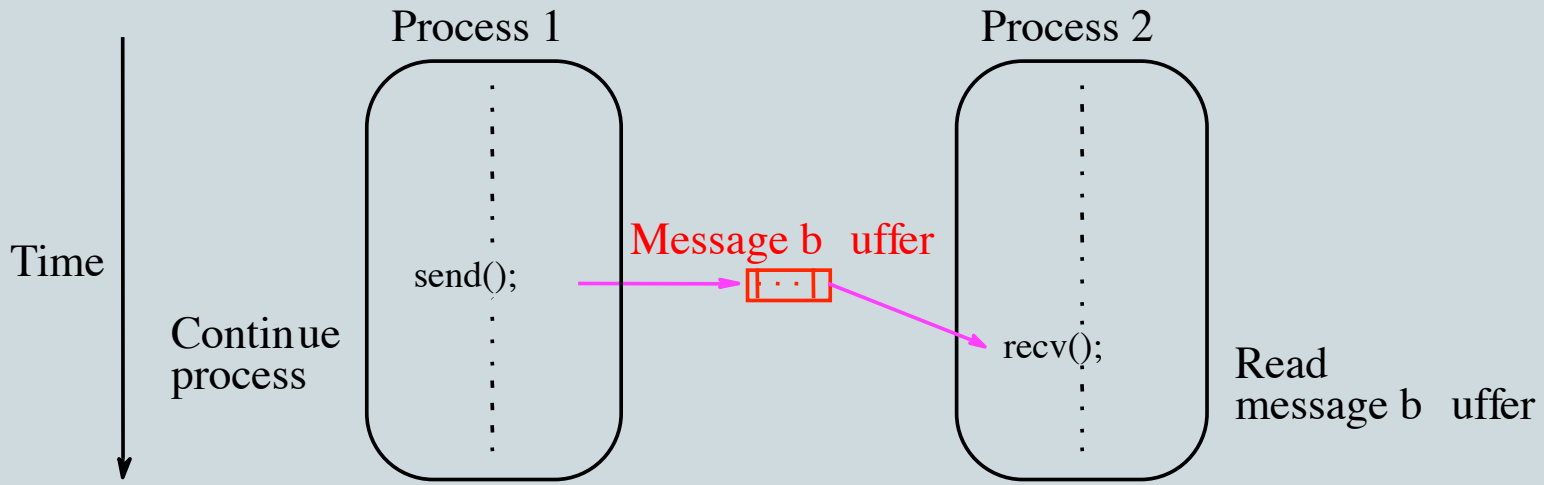


2.1 メッセージ通信プログラミングの基本

メッセージ通信ルーチン

- 非同期式メッセージ通信
  - メッセージ転送が完了する以前にリターンするルーチン
  - 受信バッファが必要になる
- 非同期式送信ルーチン
  - メッセージを受信バッファに置いてリターンする
- 非同期式受信ルーチン
  - 受信バッファからデータを受け取る
  - 受信バッファが空だったならメッセージを待つ

非同期通信のためのメッセージバッファ



2.1 メッセージ通信プログラミングの基本

- ブロッキングメッセージ送信
  - 送信のための局所処理を終えたらリターンする
  - 局所処理とは，送り出されたメッセージのメモリ領域を書き換えても，送信データには影響がないことを保証するような処理
    - MPI_Send(buf, count, datatype, dest, tag, comm)
    - MPI_Recv(but, count, datatype, src, tag, comm, status)

2.1 メッセージ通信プログラミングの基本

- ノンブロッキングメッセージ送信
  - 局所処理を完了する前にリターンするような送信
  - 受信側はメッセージが届いていなかったらすぐにリターンする
    - MPI_Isend(buf, count, datatype, dest, tag, comm, request)
    - MPI_Irecv(buf, count, datatype, source, tag, comm, request)
  - 送信完了は別の関数で確認される
  - MPI_Wait()は送受信が完了するまで待ち，完了直後にリターンする
  - MPI_Test()は送受信が完了したかどうかをテストし，すぐに戻る

- ノンブロッキング送受信のメリット
  - 送受信と別の処理がオーバーラップできる
  - 送信関数呼び出しに続く命令が，送信データに影響を与えるものでなければ問題は生じない

2.1 メッセージ通信プログラミングの基本

メッセージ通信ルーチン

- メッセージ選択
  - 特定の発信元ではなくて、どの発信元からのメッセージも受信したい
    - 発信元アドレスとしてワイルドカードを利用する。
    - メッセージタグによる選択で柔軟な送受信が可能となる
    - メッセージタグが一致したときに受信する

例:

発信プロセス: send(&x, 2, 5)

/*プロセス2へメッセージタグ5で送信*/

受信プロセス: recv(&y, 1, 5)

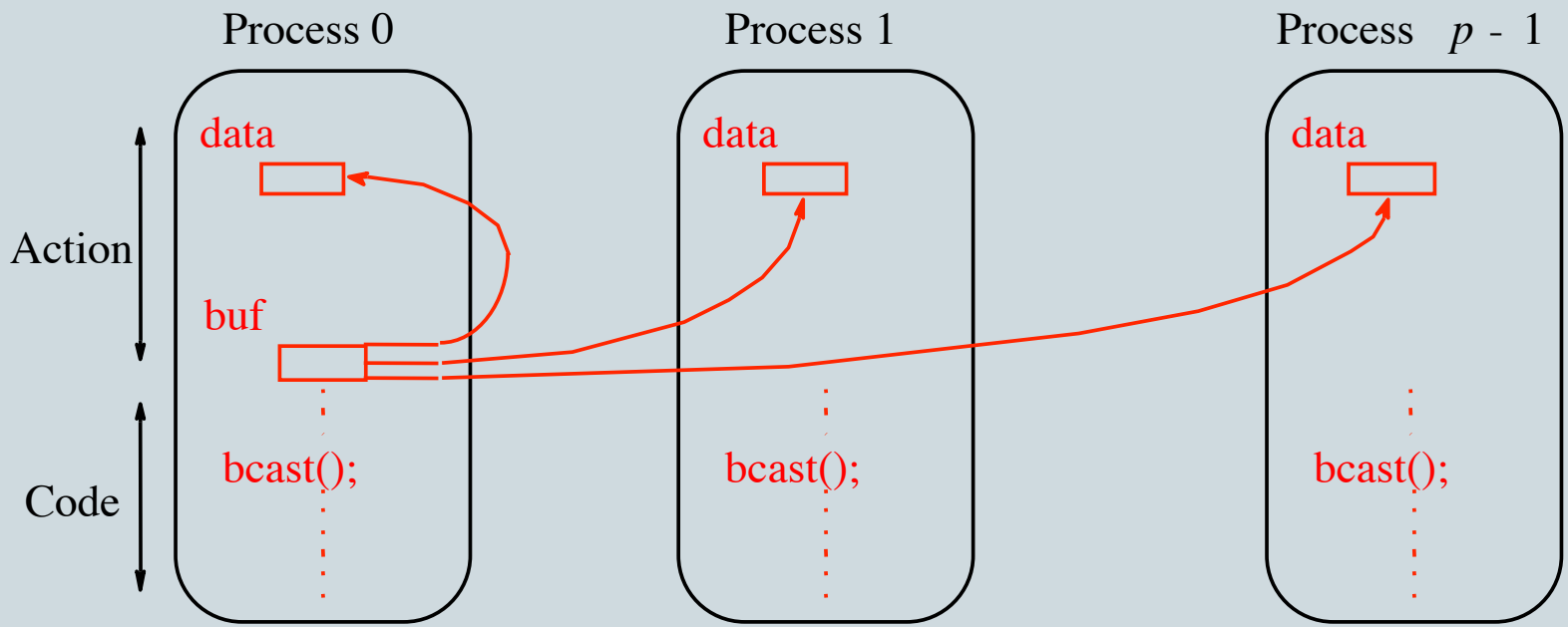
/*プロセス1からメッセージタグ5で受信*/

2.1 メッセージ通信プログラミングの基本

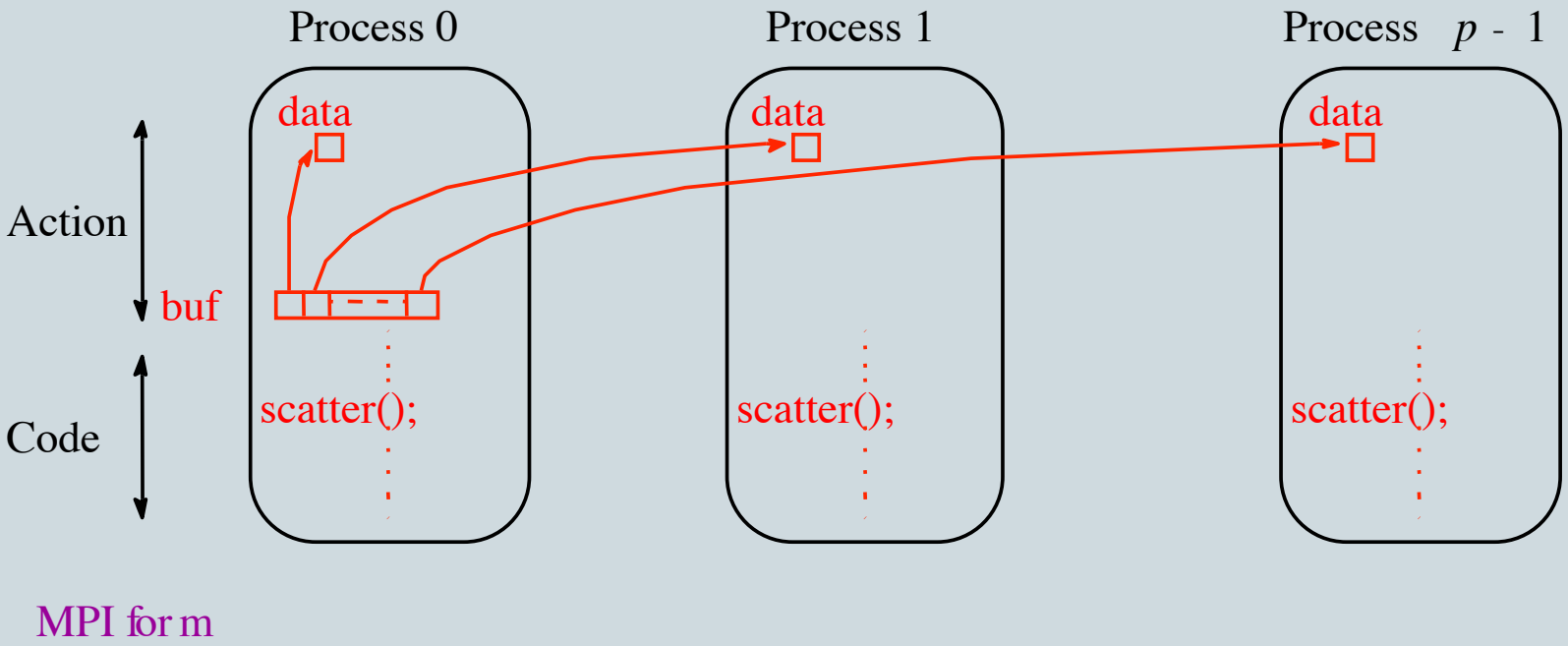
メッセージ通信ルーチン

- 一斉同報通信と収集通信と分配通信
  - ブローキャスト(broadcast)
    - 同じメッセージを全プロセスに送信すること
  - マルチキャスト(multicast)
    - 定められたプロセスのグループに同じメッセージを送信すること
  - スキャタ(scatter)(分配)
    - あるプロセスの配列のi番目の要素をi番目のプロセスに送信する
  - ギャザー(gather)(収集)
    - i番目のプロセスからのデータを配列のi番目に格納する

ブロードキャスト

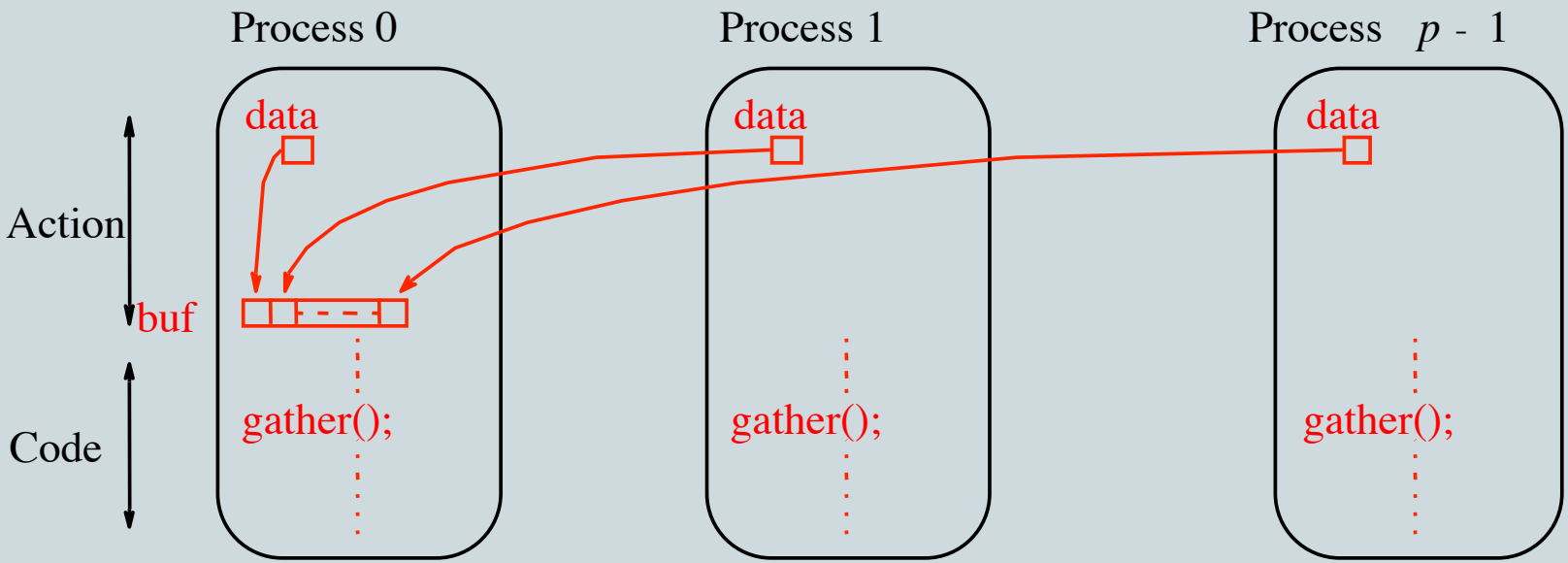

Process 0
Process 1
Process p - 1
data
data
data
Action
buf
Code
bcast();
bcast();
bcast();

SCATTER

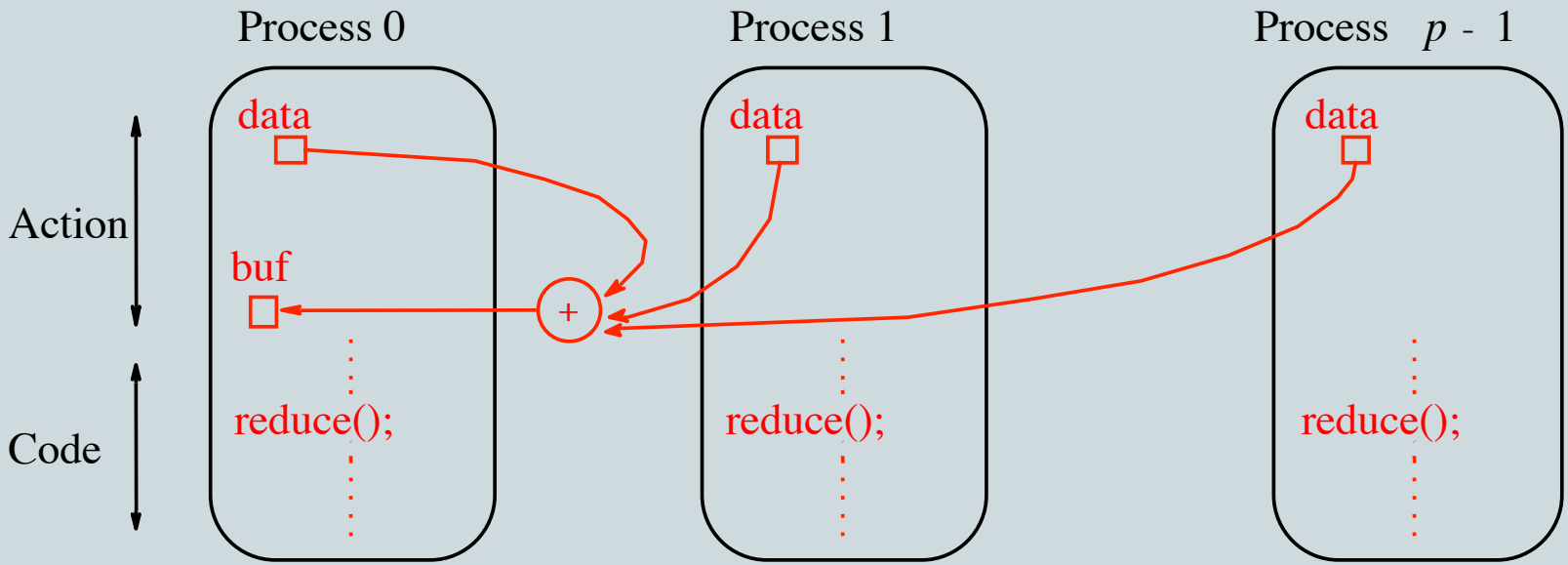


MPI for m

GATHER


Process 0
Process 1
Process p - 1
data
data
data
Action
buf
Code
gather();
gather();
gather();

REDUCE


Process 0
Process 1
Process p - 1
data
data
data
Action
buf
+
Code
reduce();
reduce();
reduce();

例題(Blocking)

To send an integer x from process 0 to process 1,

```
MPI_Comm_rank(MPI_COMM_WORLD,&myrank); /* find rank */
if (myrank == 0) {
   int x;
   MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
} else if (myrank == 1) {
   int x;
   MPI_Recv(&x, 1, MPI_INT,
   0,msgtag,MPI_COMM_WORLD,status);
}
```

例題(Non-blocking)

To send an integer x from process 0 to process 1 and allow process 0 to continue,

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* find rank */
if (myrank == 0) {
   int x;
   MPI_Isend(&x,1,MPI_INT, 1, msgtag, MPI_COMM_WORLD, req1);
   compute();
   MPI_Wait(req1, status);
} else if (myrank == 1) {
   int x;
   MPI_Recv(&x,1,MPI_INT,0,msgtag, MPI_COMM_WORLD, status);
}
```

Sample MPI program

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
    int myid, numprocs;
    int data[MAXSIZE], i, x, low, high, myresult, result;
    char fn[255];
    char *fp;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    if (myid == 0) {  /* Open input file and initialize data */
            strcpy(fn,getenv("HOME"));
            strcat(fn,"/MPI/rand_data.txt");
            if ((fp = fopen(fn,"r")) == NULL) {
                    printf("Can't open the input file: %s\n\n", fn);
                    exit(1);
            }
            for(i = 0; i < MAXSIZE; i++) fscanf(fp,"%d", &data[i]);
    }
    MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD); /* broadcast data */
    x = n/nproc; /* Add my portion Of data */
    low = myid * x;
    high = low + x;
    for(i = low; i < high; i++)
            myresult += data[i];
    printf("I got %d from %d\n", myresult, myid); /* Compute global sum */
    MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    if (myid == 0) printf("The sum is %d.\n", result);
    MPI_Finalize();
}
```

2.1 メッセージ通信プログラミングの基本

PVM (Parallel Virtual Machine)

- オークリッジ国立研究所とエモリー大学の研究者によって開発
- 異機種分散計算のためのプロジェクトで開発
- Web SITE:
  - PVMの概要 http://erpc1.naruto-u.ac.jp/~geant4/pvm/pvm.html
  - PVM3インデックスhttp://netlib.org/pvm3/

MPI (Message Passing Interface)

  - MPIフォーラムという任意参加の会議で議論され作成された
  - メッセージ通信のAPI仕様であり、特定のソフトウエアではない
  - MPICHは、最も有名なフリーのMPIライブラリ
  - LAM/MPIはMPICHより高速
- Web SITE:
  - MPICH http://www-unix.mcs.anl.gov/mpi/mpich/
  - LAM/MPI: http://www.lam-mpi.org/