主要内容

1. MPI简介及环境搭建
2. **MPI点对点通信**
3. MPI集合通信
4. MPI 派生数据类型
5. MPI程序的性能评估
6. MPI实例——并行排序算法
7. 作业

太原理工大学
TAIYUAN UNIVERSITY OF TECHNOLOGY

MPI点对点通信函数

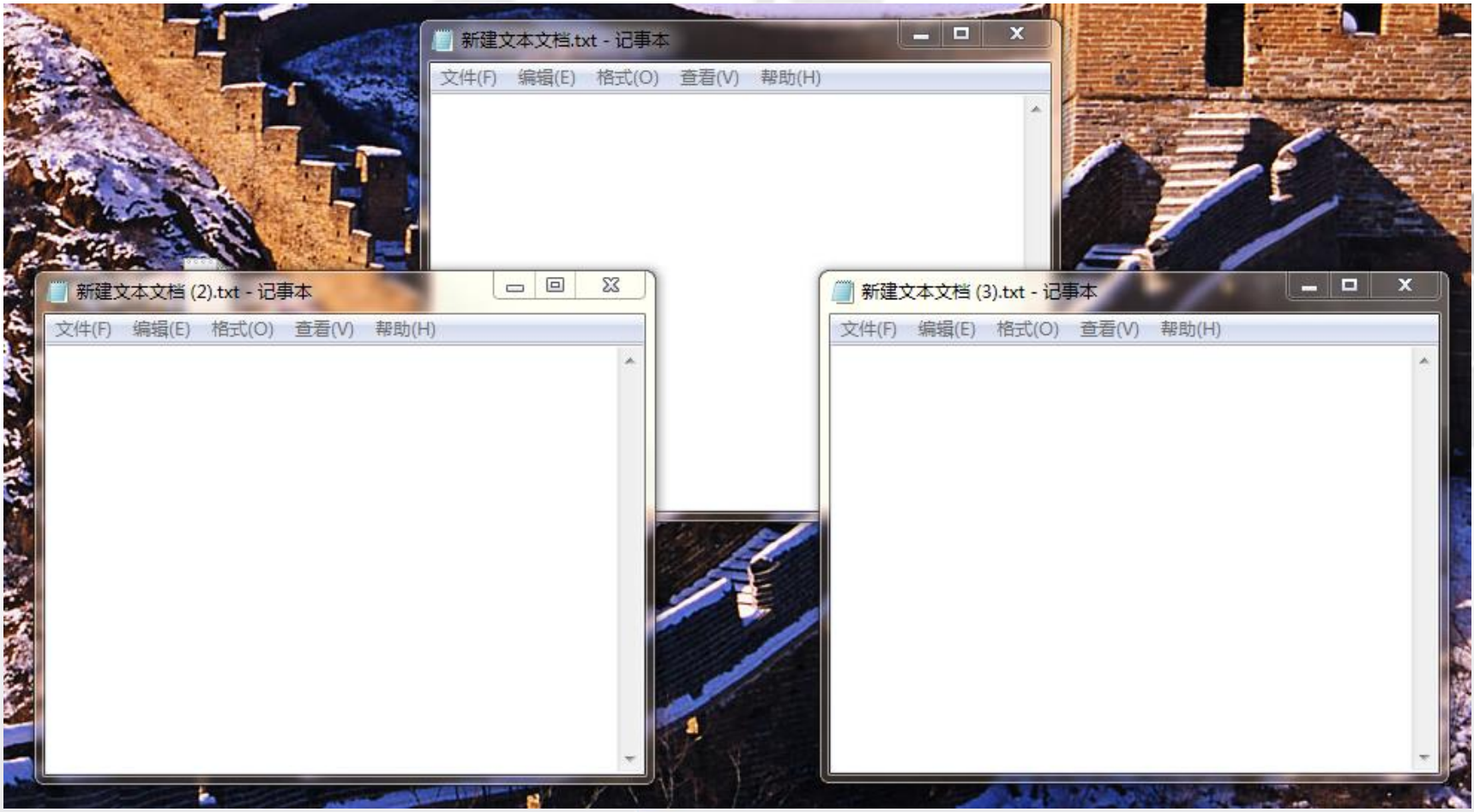
岳俊宏
E-mail:yuejunhong@tyut.edu.cn; Tel:18234095983

目录

并行程序运行方式

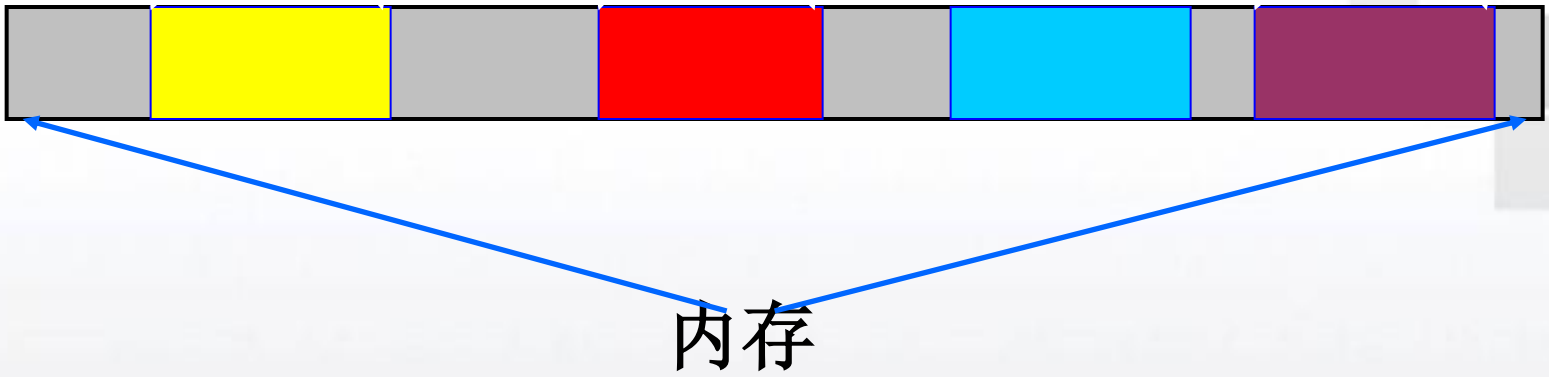
正在单机上运行的进程

- 进程是正在运行的程序的实例



多个进程在单机运行方式

进程0　进程1　进程2　进程3



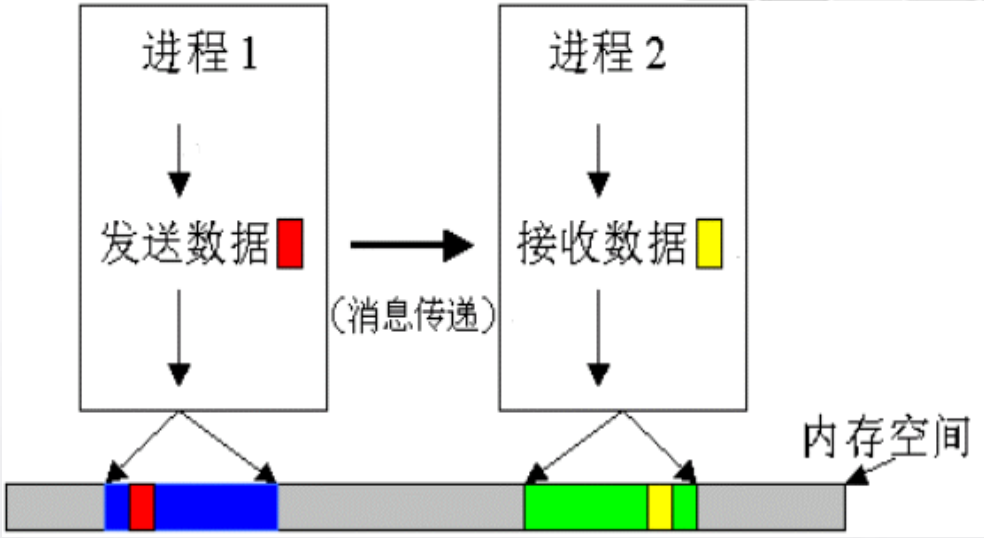
进程间通信

进程间可以相互交换信息：例如数据交换、同步等待；

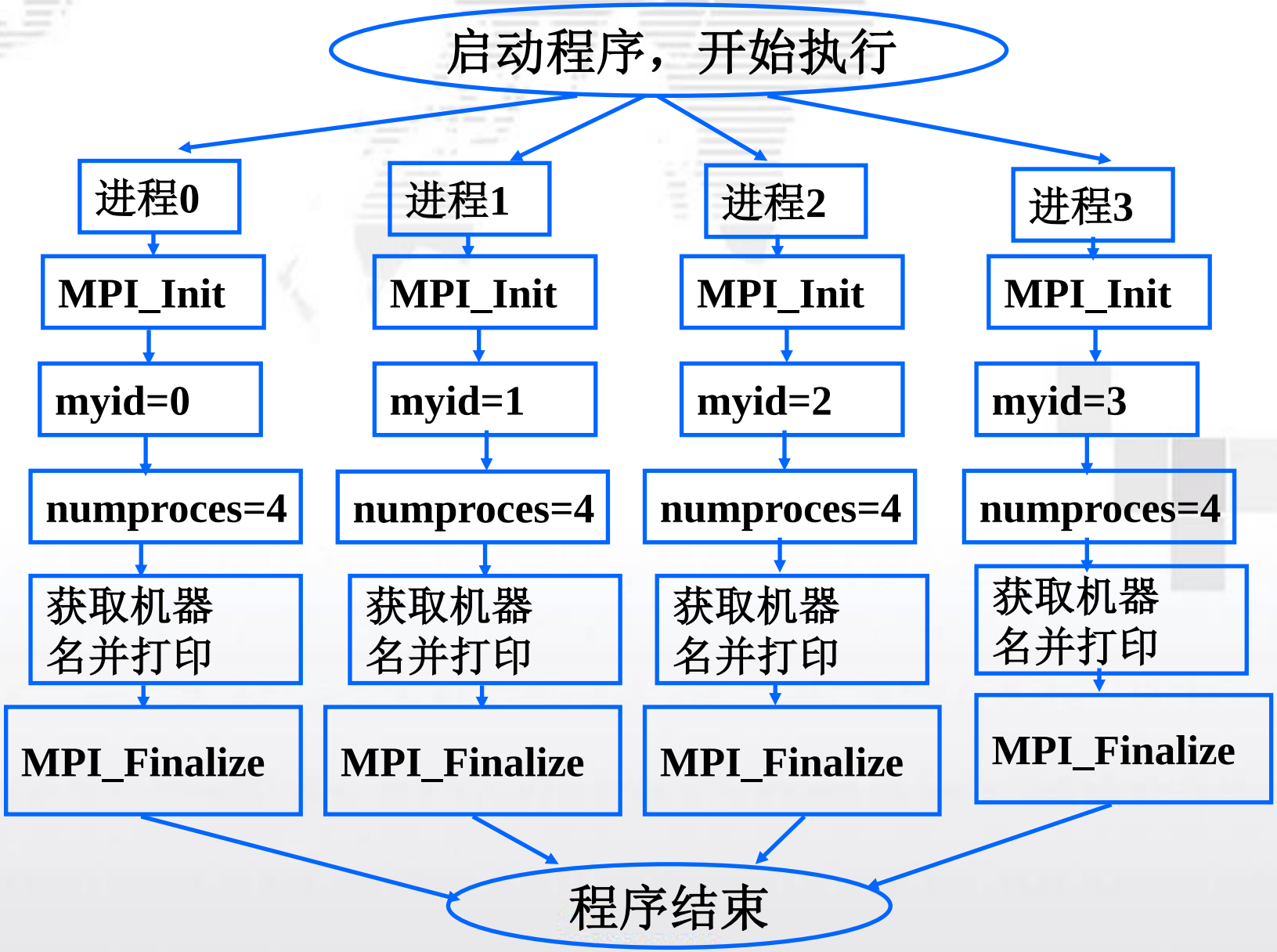
消息传递是指这些信息在进程间的相互交换，其中**消息**是这些交换信息的基本单位，

最基本的消息传递操作：

- ✓ 发送消息（send）
- ✓ 接收消息（receive）
- ✓ 进程同步（barrier）
- ✓ 规约（reduction）



并行程序的单机运行方式



并行程序集群上的运行方式

- 进程独立存在：进程位于不同的计算机，由各自独立的操作系统调度，享有独立的CPU和内存资源。
- 通过网络连接不同计算机的多个进程
- 进程间相互信息交换：消息传递。
- 消息传递的实现：用户不必关心。

并行程序集群上的运行方式

- 通过网络进行消息传递

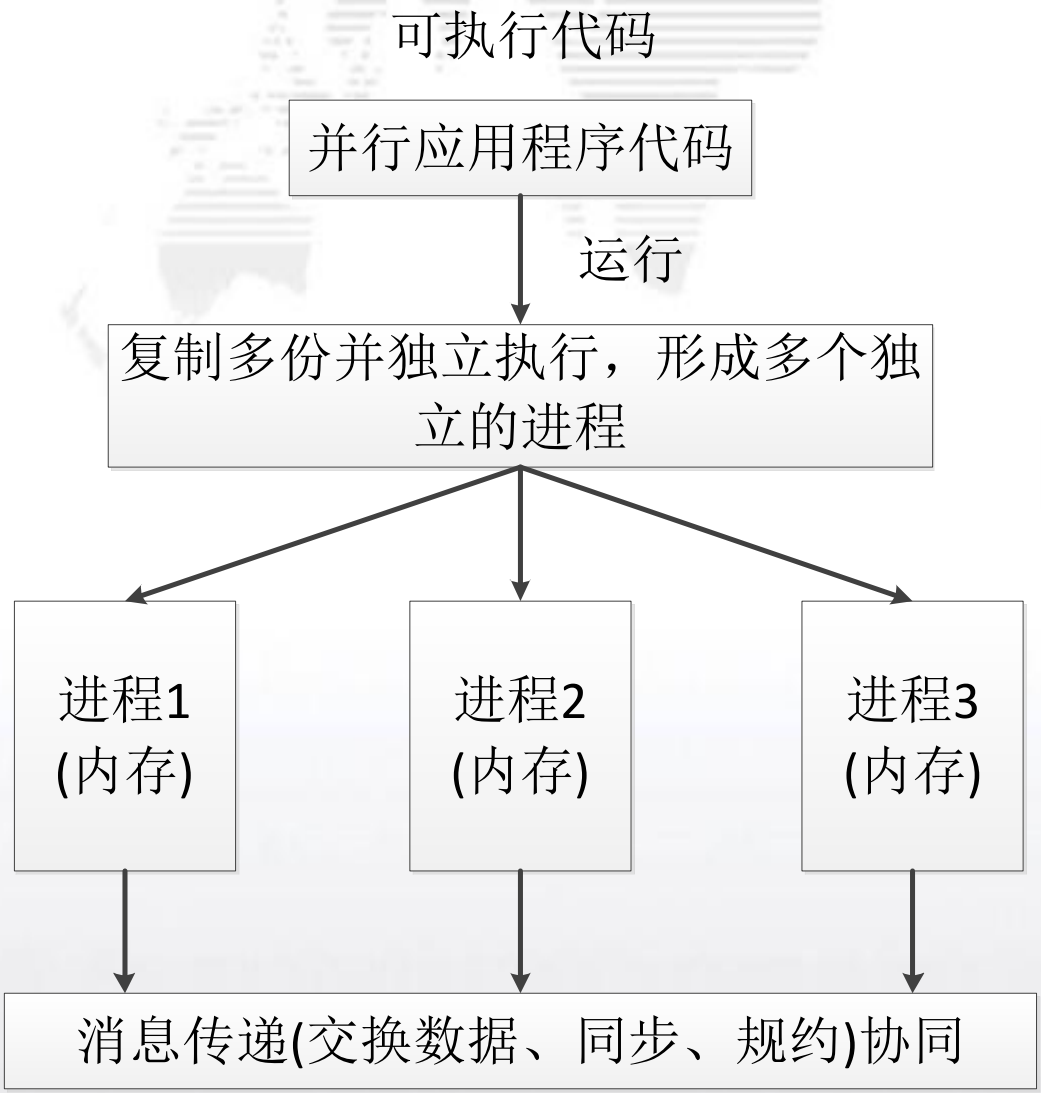
进程0

进程1

进程2

进程3

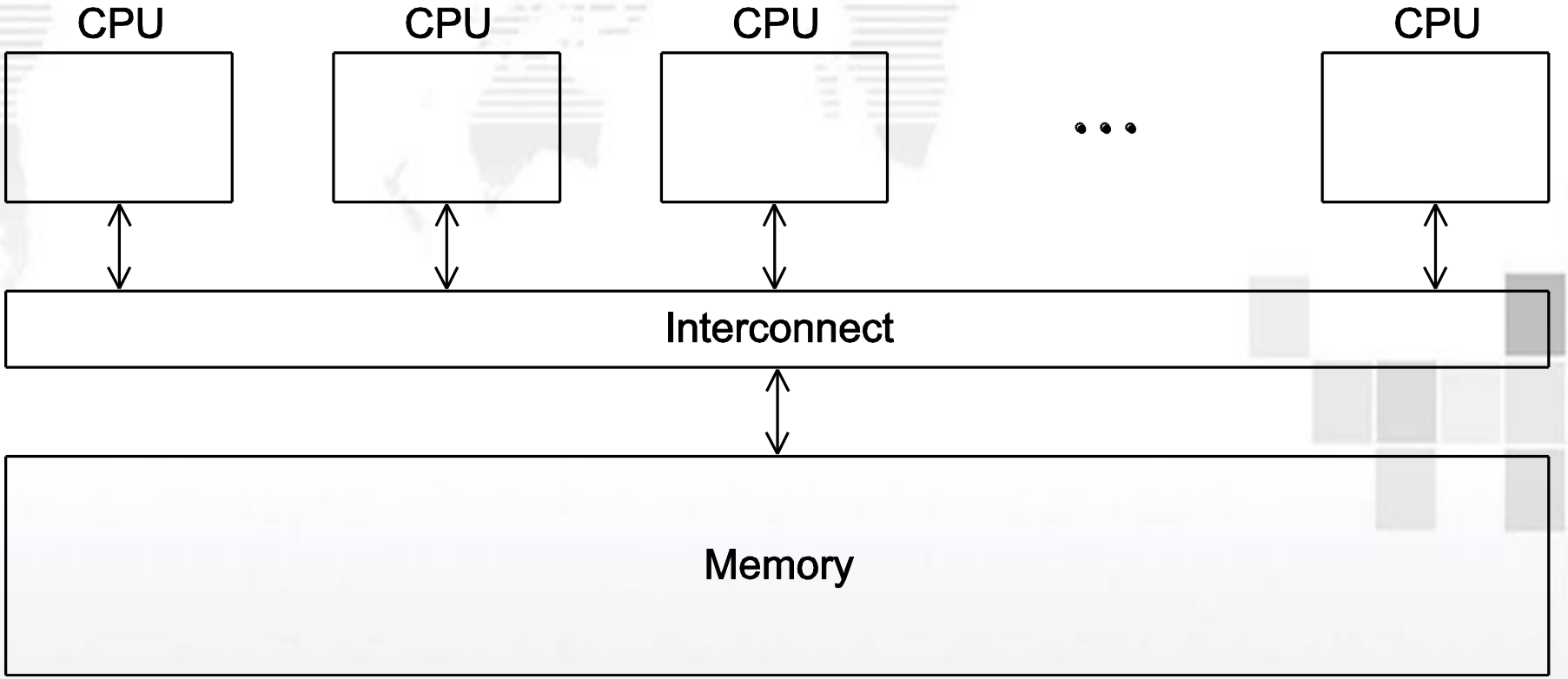
基于消息传递的并行程序执行模式



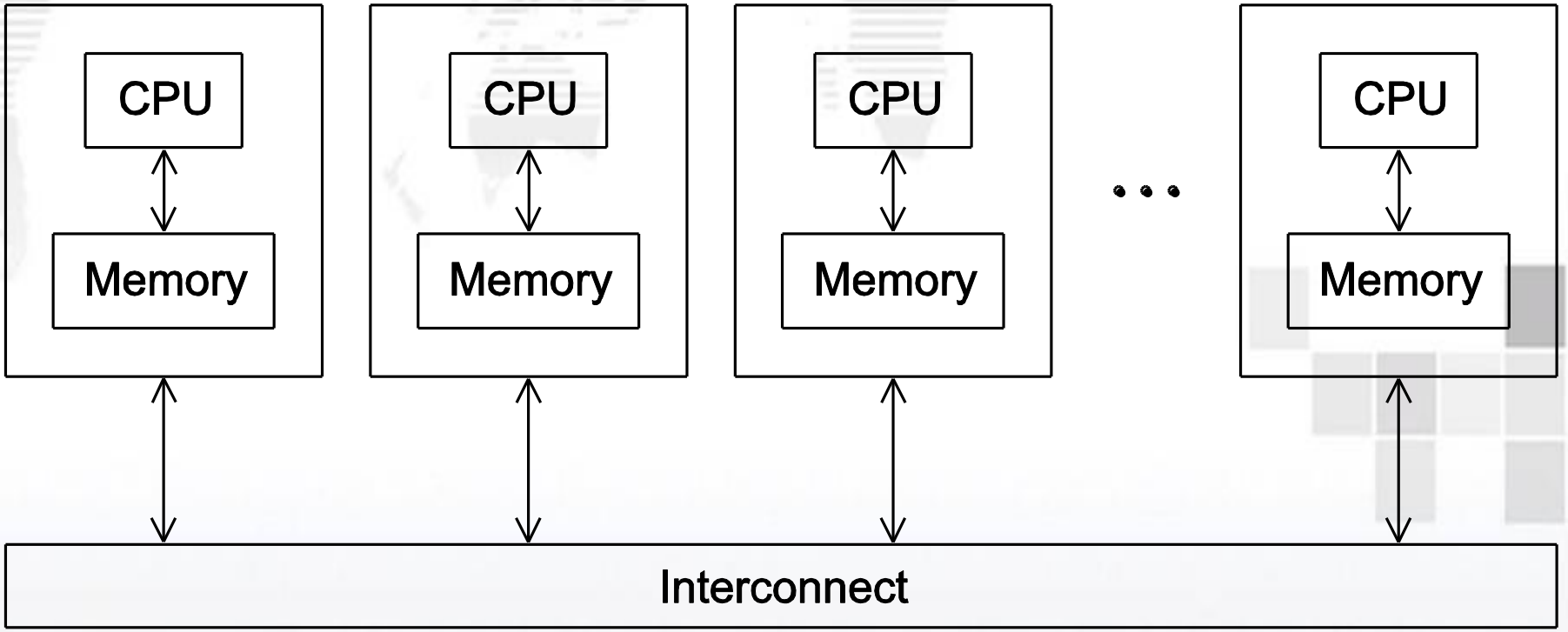
消息传递

- **消息传递**是相对于进程间通信方式而言的，与具体并行机存储模式无关，任何支持进程间通信的并行机，均可支持消息传递并行程序设计。
- 几乎所有**共享**和**分布式内存系统**的并行计算环境均支持进程间的消息传递通信。

共享式内存系统



分布式内存系统



MPI程序在并行机上的执行模式

- MPI并行程序设计平台由标准消息传递函数及相关辅助函数构成，多个进程通过调用这些函数（类似调用子程序），进行通信。
- SPMD执行模式：一个程序同时启动多份，形成多个独立的进程，在不同的处理机上运行，拥有独立的内存空间，进程间通信通过调用MPI函数来实现。

SPMD 单程序多数据流

- Single-Program Multiple-Data
- 只编译一个程序，并不是为不同的进程编译不同的程序
- 0号进程做的事情与其他进程不同
- 当其他进程生成和发送消息时，他负责接收消息并输出
- 主要靠 if-else 语句实现SPMD

标识 MPI 进程

- 每个进程开始执行时，将获得一个唯一的序号（rank）。将进程按照非负整数进行标注。
- 例如启动P个进程，序号依次为0，1，…，P-1。


MPI程序
的惯例

MPI 程序惯例

- 用C语言编写.
  - 有一个主函数main.
  - 也需要加stdio.h, string.h, etc.
- 必须加mpi.h头文件.
- 所有MPI定义的标识符都由“MPI_”开始.
- 下划线后的第一个字母大，表示函数名和MPI定义的类型.

```
MPI_Comm_rank(MPI_COMM_WORLD,&myid)
```

MPI 程序惯例

- MPI 定义的宏和常量的所有字母都是大写。
- 在自己编写的程序中不准定义或声明以前缀“MPI_”开始的任何变量和函数
- 目的：避免与MPI存在的名字混淆

```
MPI_Comm_rank(MPI_COMM_WORLD,&myid)
```

编译与运行

```
mpicc  -g  -Wall  -o  mpi_hello  mpi_hello.c
```

```
mpiexec  -n  <number of processes>   <executable>
```

```
mpiexec  -n  4  ./mpi_hello
```

run with 4 processes

mpirun

六个基本MPI函数

一、MPI初始化

调用说明：

```
int MPI_Init(
        int*      argc_p,   /* in/out */
        char*** argv_p    /* in/out */)
```

功能：完成MPI程序的所有初始化工作，

- ✓ 资源分配—如存储空间、进程编号，通信子MPI_COMM_WORLD的创建等；

所有MPI程序的第一条可执行语句都是它。

返回int型错误码，MPI_SUCCESS

4指针数组作main函数的形参

指针数组的一个重要应用是作为main函数的形参。

如 void main(int argc, char *argv[])

argc和argv就是main函数的形参。main函数是由操作系统调用的。当处于操作命令状态下，输入main所在的文件名（经过编译、连接后得到的可执行文件名，后缀为.exe），操作系统就调用main函数。

main函数的形参的值从何处得到?

显然不可能在程序中得到。实际上，实参是和命令一起给出的，也就是在一个命令行中包括命令名和需要传给main函数的参数。

指针数组作main函数的形参

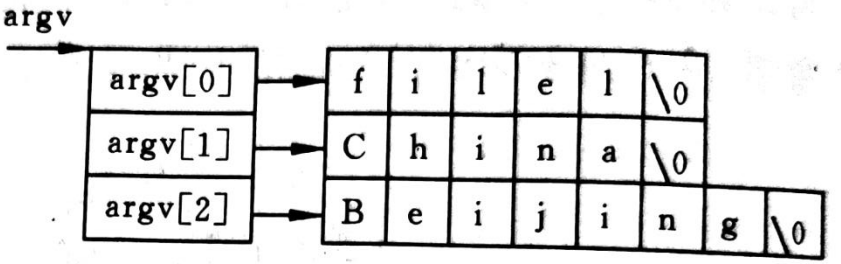
命令行的一般形式为：

命令名 参数1 参数2 … 参数n

命令名和各参数之间用空格分隔。

如：

file1 China Beijing



其中file1为文件名，可包括盘符、路径及文件的扩展名。

main函数中形参argc是指命令行中参数的个数，argc的值等于3

二、MPI结束

调用说明:

int MPI_Finalize (void)

功能:

结束MPI程序的运行（释放资源及环境等），所有MPI程序的最后一条可执行语句都是它，否则程序的运行结果是不可预知的。

示例

```c
#include <stdio.h>
#include <mpi.h>
void main(int argc, char *argv[]){
    MPI_Init(&argc,&argv);
    printf("Hello world!\n");
    MPI_Finalize();
}
```

```
[hdusr@Node2 mpi-work]$ mpiexec -n 4 ./mpi_hello1
Hello world!
Hello world!
Hello world!
Hello world!
```

三、获取当前进程标识

调用说明:

MPI_Comm_rank(MPI_Comm comm, int *rank)

功能:

返回调用进程在给定的通信子中的进程标识号，有了这一标识号，不同的进程就可以将自身和其它的进程区别开来，实现各进程的并行和协作。

通信子

通信子是所有可以进行相互通信的进程的集合。

MPI_COMM_WORLD是MPI预定义的常量，是一个包含所有用户启动进程的通信子。（MPI_Init）

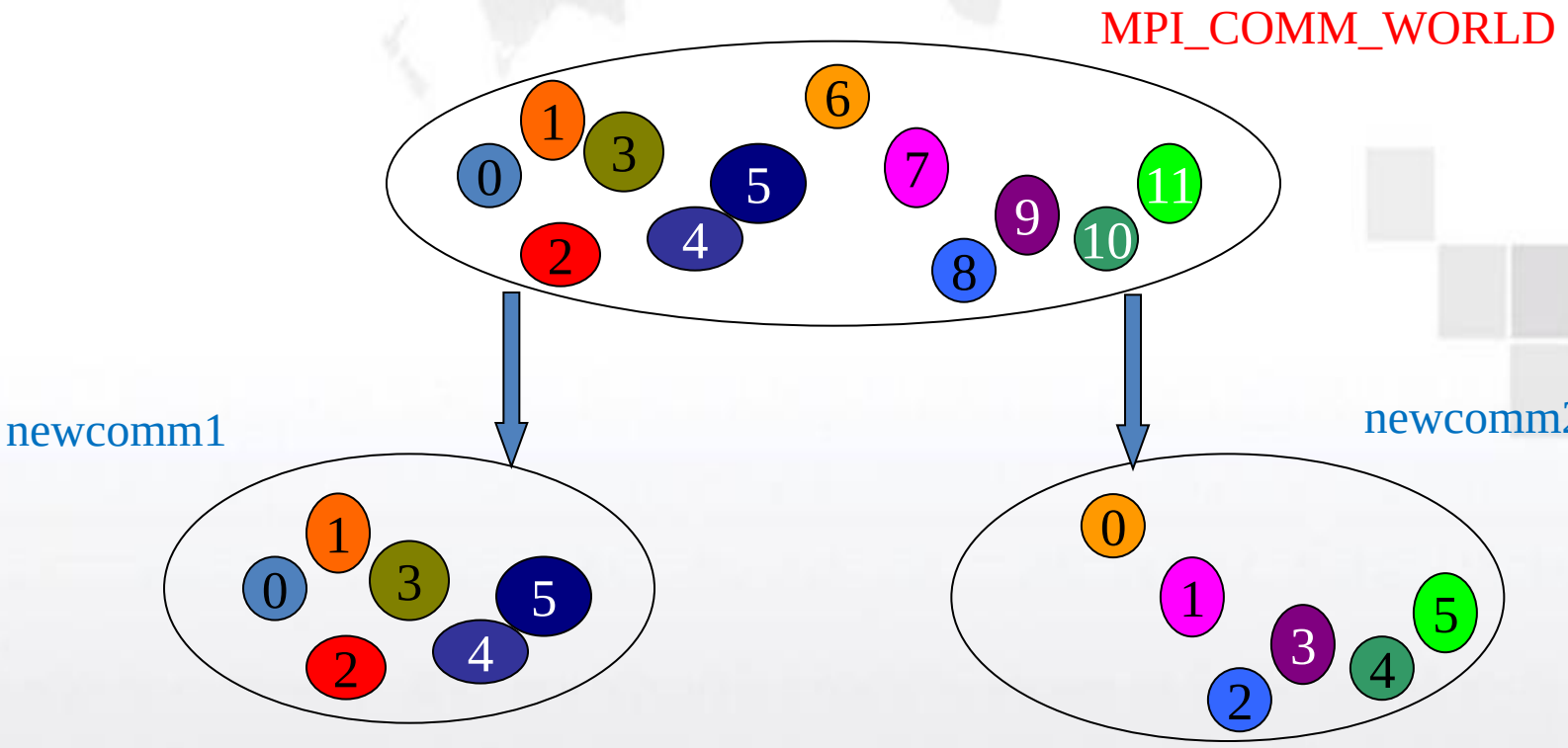
进程标识号（进程id）：

- 进程在某个通信子中唯一的标识号;
- 一个进程可以属于不同的通信子，它在不同通信子中的标识号不同;
- 进程号为从0开始的连续非负整数。

MPI_Comm_rank(MPI_COMM_WORLD，&my_rank)

通信子

如果一共有N个进程参加通信，则进程编号从0到N－1；可提供一个相对独立的通信区域。



MPI_Comm_create(MPI_COMM_WORLD,grp, newcomm) <扩展>

四、获取通信域中的进程总数

调用说明:

MPI_Comm_size(MPI_Comm comm, int *size)

功能:

返回给定通信子中所包括的进程个数，不同进程通过这一调用得知在给定的通信子中一共有多少个进程在并行执行。

MPI_Comm_size (MPI_COMM_WORLD，&comm_sz)

MPI点对点通信 示例

```
#include <stdio.h>
#include <mpi.h>
void main(int argc, char *argv[]){
        int rc;
        int comm_sz;
        int my_rank;
        rc = MPI_Init(NULL,NULL);
        if(rc!= MPI_SUCCESS){
                printf("Error starting MPI program. Terminating.\n");
                MPI_Abort(MPI_COMM_WORLD, rc);
        }
        MPI_Comm_size(MPI_COMM_WORLD,&comm_sz);
        MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
        printf("Hello, I am %d of %d\n",my_rank,comm_sz);
        MPI_Finalize();
}
```

```
[hdusr@Node1 mpi-work]$ mpicc -g -o mpi_hello5 mpi_hello5.c
[hdusr@Node1 mpi-work]$ mpiexec -n 4 ./mpi_hello5
Hello, I am 1 of 4
Hello, I am 0 of 4
Hello, I am 3 of 4
Hello, I am 2 of 4
```

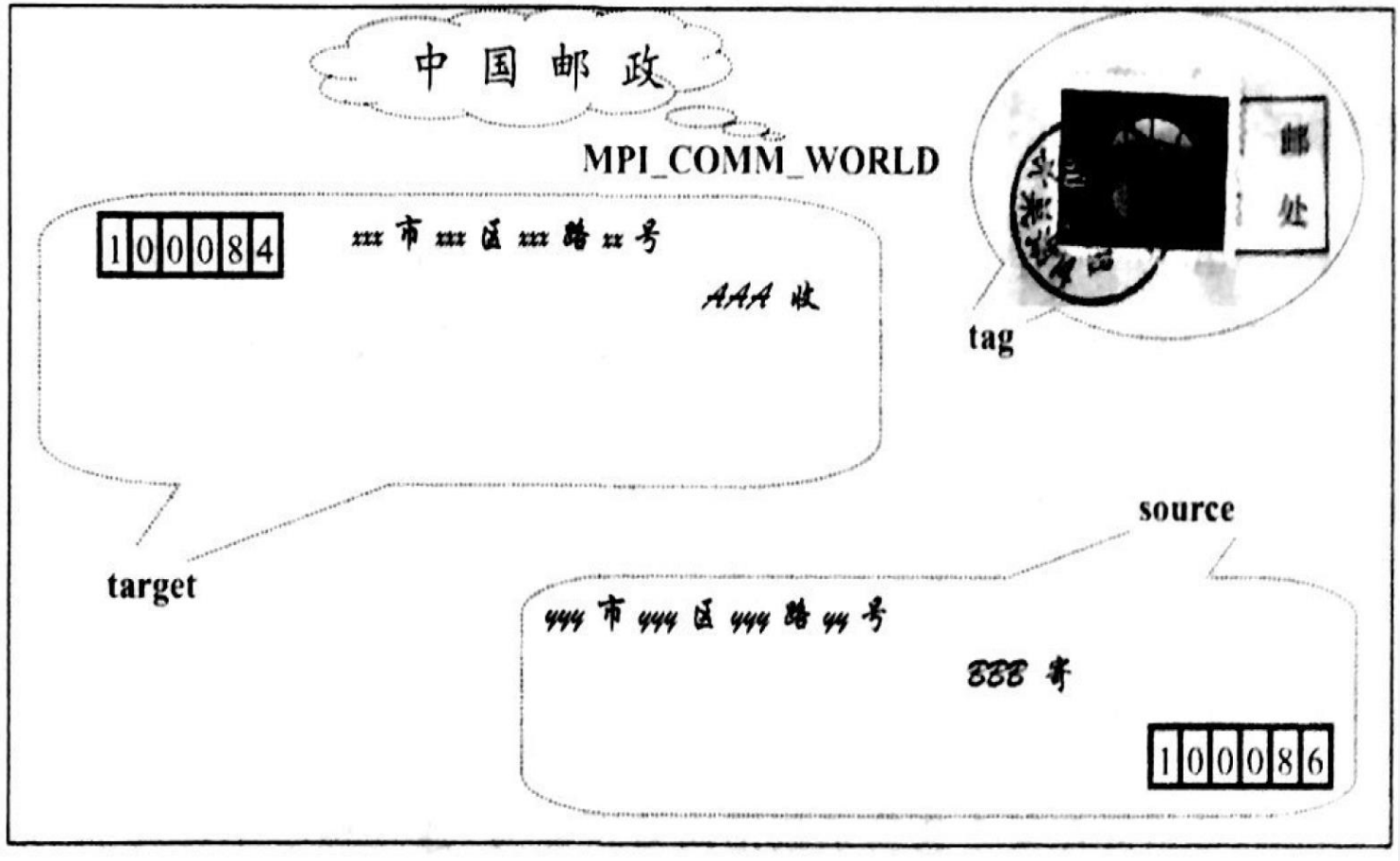
五、消息发送

写一封信需要已知些什么？

信封
- 收信人姓名：张三
- 收信人地址和邮编：……
- 邮局邮戳

内容
- 信中写些什么内容
- 用什么语言书写 汉语？英语？
- ……

MPI消息的组成


中 国 邮 政
MPI_COMM_WORLD
100084
xxx 市 xxx 区 xxx 路 xx 号
AAA 收
邮
处
tag
target
source
yyy 市 yyy 区 yyy 路 yy 号
BBB 寄
100086

MPI消息的组成

- MPI消息包括信封和数据两部份
- 信封：＜源/目，标识，通信子＞
- 数据：＜起始地址，数据个数，数据类型＞

消息数据　　消息信封

- MPI_Send(buf,count,datatype,dest,tag,comm)

消息数据　　消息信封

- MPI_Recv(buf,count,datatype,source,tag,comm,status)

五、消息发送

```
int MPI_Send(
    void*          msg_buf_p      /* in */,
    int            msg_size       /* in */,
    MPI_Datatype   msg_type       /* in */,
    int            dest           /* in */,
    int            tag            /* in */,
    MPI_Comm       communicator   /* in */);
```

调用说明:

int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

五、消息发送

buf,count,datatype

- buf: 发送缓冲区的起始地址

  发送哪些连续数据？从内存的哪个位置开始发送?

- count: 发送数据的个数

  发送多少个数？100个？200个?

- datatype: 发送的数据类型

  发送什么样的数据？是整数？实数？还是其它?

五、消息发送

MPI_Send(buf,count,datatype,dest,tag,comm)

dest 目的进程标识号(整型)

发送给谁？接收方是谁？

tag 消息标志(整型)

如果短时间内向同一进程发送了两次消息，如何区分它们？

comm 通信子

在哪个范围内发送？消息在哪个通信子内传递？

五、消息发送

注意：

- 发送缓冲区中的count不是以字节计数而是以数据类型为单位；
- count若是字符串，则其大小需加上\0所占的字符数量。
- datatype数据类型可以是MPI的预定义类型也可以是用户自定义的类型.（C语言中类型不能作为参数传递给函数）
- buf的大小与count和datatype所指定的消息的大小并不相同。

tag标志可以把本次发送的消息和本进程向同一目的进程发送的其它消息区别；

五、消息发送

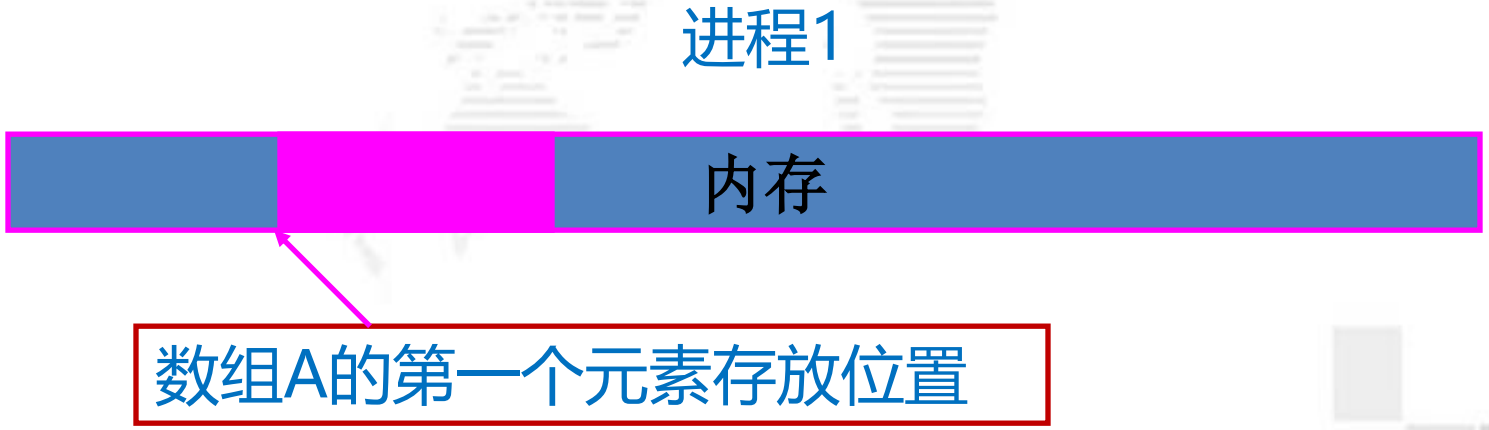
说明： MPI预定义的数据类型

MPI预定义了一些数据类型可以直接使用

MPI预定义的数据类型

MPI预定义的数据类型	相应的C语言数据类型
MPI_CHAR	signed char (一个字节)
MPI_SHORT	signed short int （2个字节）
MPI_INT	signed int （4字节）
MPI_LONG	signed long int（4字节）
MPI_UNSIGNED_CHAR	unsigned char （0-255，一个字节）
MPI_ UNSIGNED_SHORT	unsigned short int（2字节）
MPI_UNSIGNED	unsigned int 取决于系统
MPI_UNSIGNED_LONG	unsigned long int （4字节）
MPI_FLOAT	float （4字节）
MPI_DOUBLE	double （8字节）
MPI_LONG_DOUBLE	long double （10字节）
MPI_BYTE	无对应类型
MPI_PACKED	无对应类型

五、消息发送——举例

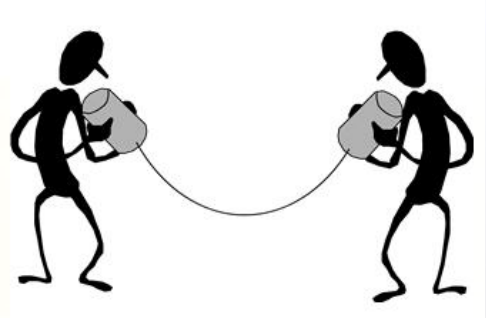


MPI_Send(A,100,MPI_FLOAT,2,1,MPI_COMM_WORLD)

意义?

六、消息接收

```
int MPI_Recv(
      void*          msg_buf_p      /* out */,
      int            buf_size       /* in  */,
      MPI_Datatype   buf_type       /* in  */,
      int            source         /* in  */,
      int            tag            /* in  */,
      MPI_Comm       communicator   /* in  */,
      MPI_Status*    status_p       /* out */);
```



六、消息接收

MPI_RECV(buf,count,datatype,source,tag,comm,status)

buf发送缓冲区的起始地址(可选类型)

接收到的数据放在哪儿?

count将发送的数据的个数(非负整数)

最多接收多少个数，100个？200个?

datatype 发送数据的数据类型

接收什么样的数据？是整数？实数？还是其它？

Source 源进程标识号(整型)

从谁那儿接收？发送方是谁?

tag消息标志(整型)

如果短时间有一个进程向我发送了两次消息，如何区分它们?

comm 通信域

在哪个范围内接收？消息在哪个通信域内传递?

六、消息接收

注意：

- 从进程source接收消息，并且该消息的数据类型和消息标识与本接收进程指定的datatype和tag相一致
- 接收到的消息所包含的数据元素的个数最多不能超过count。
  - ✓ MPI没有截断，接收到的消息大于接收缓冲区会发生溢出错误。
  - ✓ 如果一个短于接收缓冲区的消息到达，那么只有相应于这个消息的那些地址被修改。
  - ✓ count可以是零，这种情况下消息的数据部分是空的

思考

接收函数可以在不知道以下信息的情况下接收消息：

- ✓ 消息中的数据量
- ✓ 消息中发送者
- ✓ 消息的标签

接收者是如何找出这些值的?

status_p 参数

```
MPI_Recv(recv_buf_p, recv_buf_sz, recv_type, src, recv_tag,
         recv_comm, &status);
```

MPI_Status*

MPI_Status status;
status.MPI_SOURCE
status.MPI_TAG
status. MPI_ERROR

MPI_SOURCE
MPI_TAG
MPI_ERROR

- MPI定义的一个数据类型，使用之前需要用户为它分配空间。
- 在C实现中，状态变量是由**至少三个**域组成的结构类型，这三个域分别是: MPI_SOURCE、MPI_TAG和MPI_ERROR

status_p 参数

MPI_RECV(buf,count,datatype,source,tag,comm,&**status**)

- status.MPI_SOURCE:发送数据进程的标识
- status.MPI_TAG:发送数据使用的tag标识
- status.MPI_ERROR:本接收操作返回的错误代码
- 说明方式： MPI_Status status

举例

MPI_Recv(A,100,MPI_FLOAT,1,1,MPI_COMM_WORLD,&status)

则:

status. MPI_SOURCE=1

status.MPI_TAG=1

status.MPI_ERROR=MPI_SUCCESS　　(MPI_ERRTYPE)

若不用这个参数，也可以用MPI预定义的常量MPI_STATUS_IGNORE

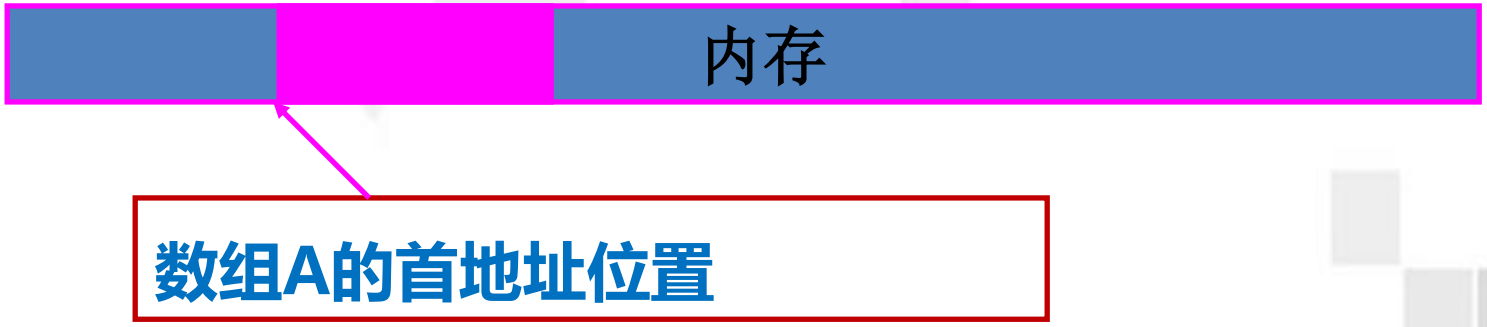
如何确定接收数据量

```
int MPI_Get_count(
      MPI_Status*   status_p   /* in  */,
      MPI_Datatype  type       /* in  */,
      int*          count_p    /* out */);
```

如何得到接收的数据量（需要一次计算）

举例

进程2



MPI_Recv(A,100,MPI_FLOAT,1,1,MPI_COMM_WORLD,&status)

消息匹配

```
MPI_Send(send_buf_p, send_buf_sz, send_type, dest, send_tag,
         send_comm);
```

r

MPI_Send

src = q

MPI_Recv

dest = r

```
MPI_Recv(recv_buf_p, recv_buf_sz, recv_type, src, recv_tag,
         recv_comm, &status);
```

q

消息匹配

发送：send_buf_p,send_buf_sz,send_type;

接收：recv_buf_p,recv_buf_sz,recv_type;

还需满足：

- ✓ recv_type=send_type
- ✓ recv_buf_sz>=send_buf_sz

由q号进程发送的消息就可以被r号进程成功地接收。

类型匹配——举例

```
……
float   a[100];
……
if(myid==0)
MPI_Send(a,20,MPI_FLOAT,1,99,MPI_COMM_WORLD);
if(myid==1)
MPI_Recv(a,20,MPI_FLOAT,0,99,MPI_COMM_WORLD,&status
);
……
```

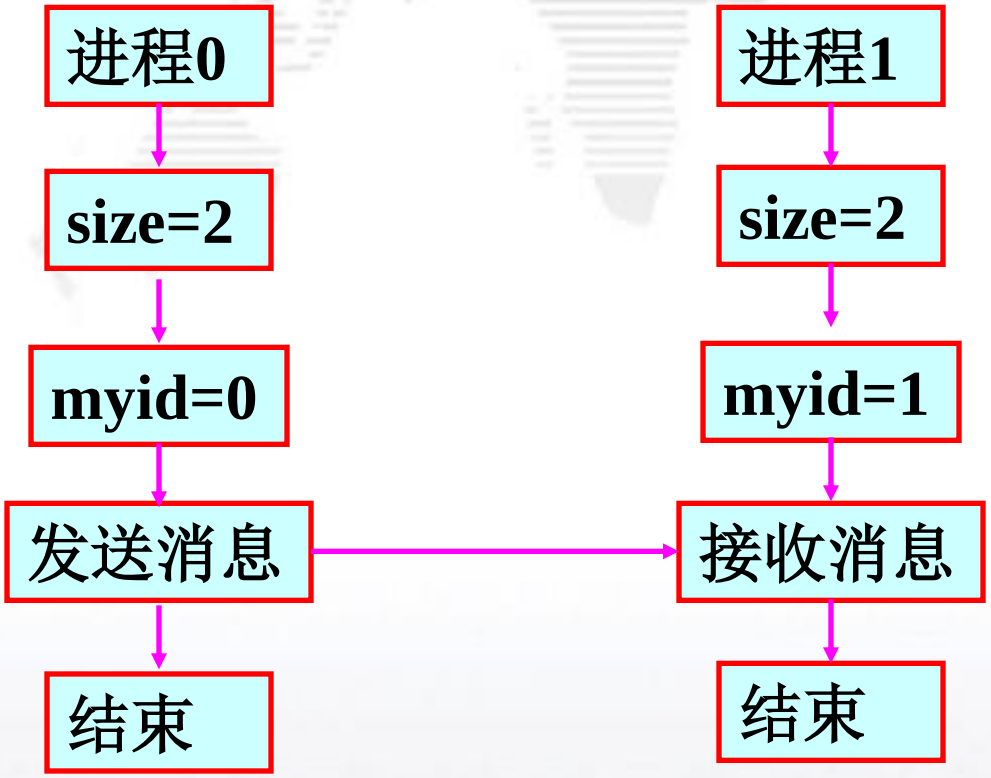
宿主语言的类型和通信操作所指定的类型相匹配

发送方和接收方的类型相匹配

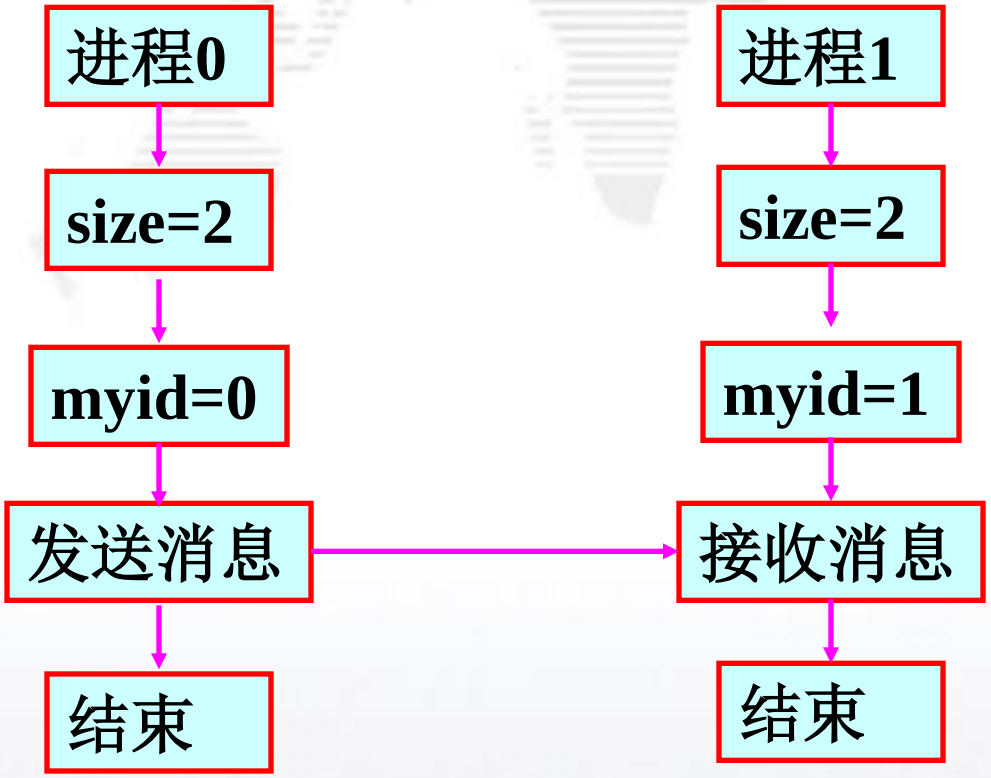
消息发送、接收程序举例

```
#include “mpi.h”
main(int argc, char ** argv)
{   int a[60],rank,size; MPI_Status  status;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank((MPI_COMM_WORLD,&rank);
  if(rank==0)
    MPI_Send(a,20,MPI_INT,1,99,MPI_COMM_WORLD);
  if(rank==1)
   MPI_Recv(a,20,MPI_INT,0,99,
MPI_COMM_WORLD,&status);
 MPI_Finalize( );
}
```

执行过程



执行过程



执行过程

发送、接收前：

进程0：　a[i]=i

进程1：　a[i]=10*i

发送、接收后：

进程0：　a[i]=i

进程1：　a[i]=i

举例

```
……
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
if(myid==0)
{ MPI_Recv( buf1,10,MPI_INT,1,1,MPI_COMM_WORLD,&status);
  MPI_Recv (buf2,10,MPI_INT,2,1,MPI_COMM_WORLD,&status);
}
if(myid==1)
 MPI_Send (buf1,10 ,MPI_INT,0,1,MPI_COMM_WORLD );
if ( myid ==2 )
 MPI_Send(buf2,10 ,MPI_INT,0,1,MPI_COMM_WORLD )
……
```

消息来源于进程1

消息来源于进程2

消息发送给进程0

消息发送给进程0

程序的执行过程

相关问题 任意源和任意标识

MPI_ANY_SOURCE，MPI_ANY_TAG

1、MPI_ANY_SOURCE：

任何进程发送的消息都可以接收，但其它要求必须满足；

2、MPI_ANY_TAG：

任何标签的消息都可以接收，但其它要求必须满足；

3、两者可同时使用，也可单独使用；

4、不能给通信子comm指定任意值；

5、发送操作与接收操作不对称。

举例

```
……
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
if(myid==0)
{ MPI_Recv( buf1,10,MPI_INT,MPI_ANY_SOURCE,1,
                              MPI_COMM_WORLD,&status);
  MPI_Recv (buf2,10,MPI_INT,
MPI_ANY_SOURCE,1,MPI_COMM_WORLD,&status);
}
if(myid==1)
 MPI_Send (buf1,10 ,MPI_INT,0,1,MPI_COMM_WORLD );
if ( myid ==2 )
 MPI_Send(buf2,10 ,MPI_INT,0,1,MPI_COMM_WORLD )
……
```

消息来源：可能是进程1，也可能是进程2

消息来源：可能是进程1，也可能是进程2

消息发送给进程0

消息发送给进程0

接收任意源和任意标签的消息

进程1　进程2　……　进程N-1

进程0接收消息并打印

程序分析

进程0： 接收来自于其它进程的消息

MPI_Recv(……);

source=MPI_ANY_SOURCE

tag=MPI_ANY_TAG

其它进程：

向进程0发送消息

MPI_Send(……);

dest=0;

tag=任意值

思考

strlen(greeting)代替strlen(greeting)+1来计算进程1、2、…、comm_sz-1发送消息的长度，会发生什么情况？如果用MAX_STRING代替strlen(greeting)+1又会是什么结果？你可以解释这些结果吗？

```
16    if (my_rank != 0) {
17       sprintf(greeting, "Greetings from process %d of %d!",
18             my_rank, comm_sz);
19       MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
20             MPI_COMM_WORLD);
21    } else {
22       printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
23       for (int q = 1; q < comm_sz; q++) {
24          MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
25             0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26          printf("%s\n", greeting);
27       }
28    }
```

思考

```
16    if (my_rank != 0) {
17       sprintf(greeting, "Greetings from process %d of %d!",
18             my_rank, comm_sz);
19       MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
20             MPI_COMM_WORLD);
21    } else {
22       printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
23       for (int q = 1; q < comm_sz; q++) {
24          MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
25             0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26          printf("%s\n", greeting);
27       }
28    }
```

程序运行过程，打印顺序?

send and receive的语义

发送 阻塞和缓冲

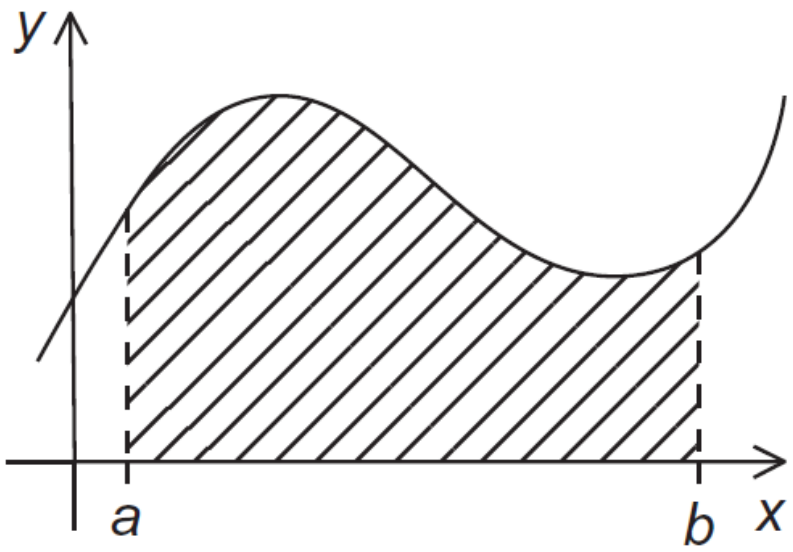
接收 总是阻塞的

补充

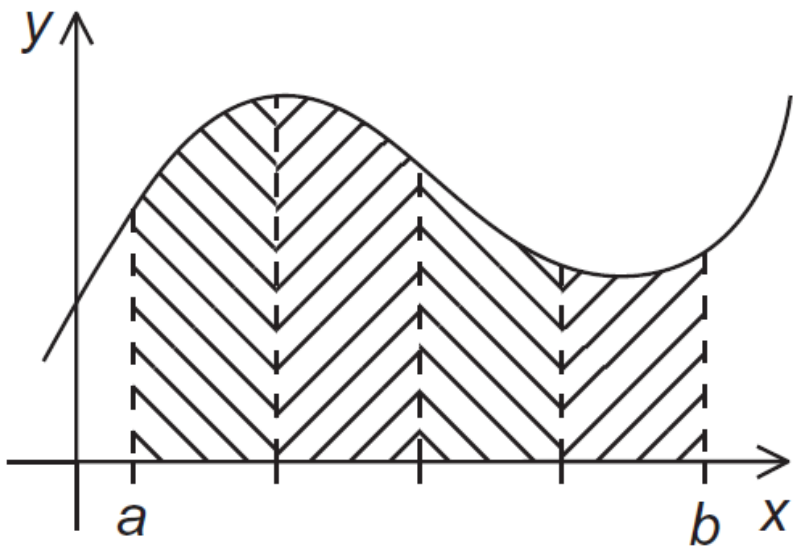
```
int MPI_Get_processor_name( char *name,
                            int *resultlen );
                                /*得到机器名*/
MPI_Get_processor_name(processor_name,&namelen);
```


MPI 实现
梯形积分

梯形积分法

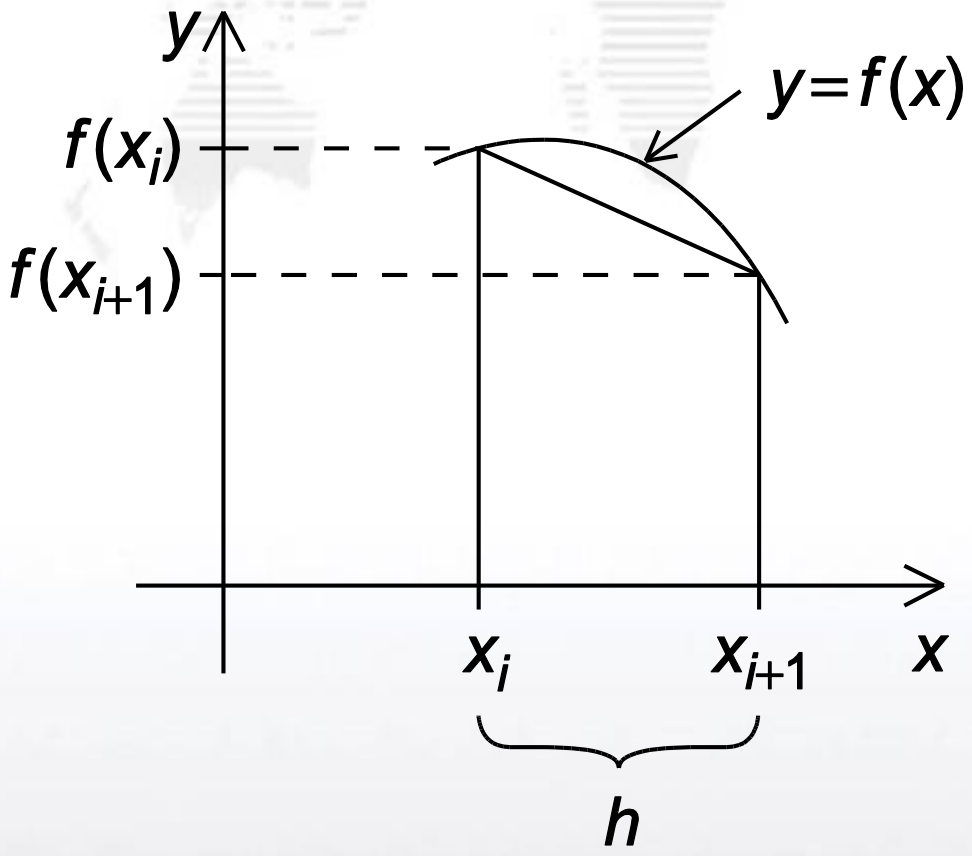


(a)



(b)

梯形



梯形

$$\text{Area of one trapezoid} = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

$$h = \frac{b-a}{n}$$

$$x_0 = a,\ x_1 = a+h,\ x_2 = a+2h,\ \ldots,\ x_{n-1} = a+(n-1)h,\ x_n = b$$

$$\text{Sum of trapezoid areas} = h[f(x_0)/2 + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + f(x_n)/2]$$

串行程序

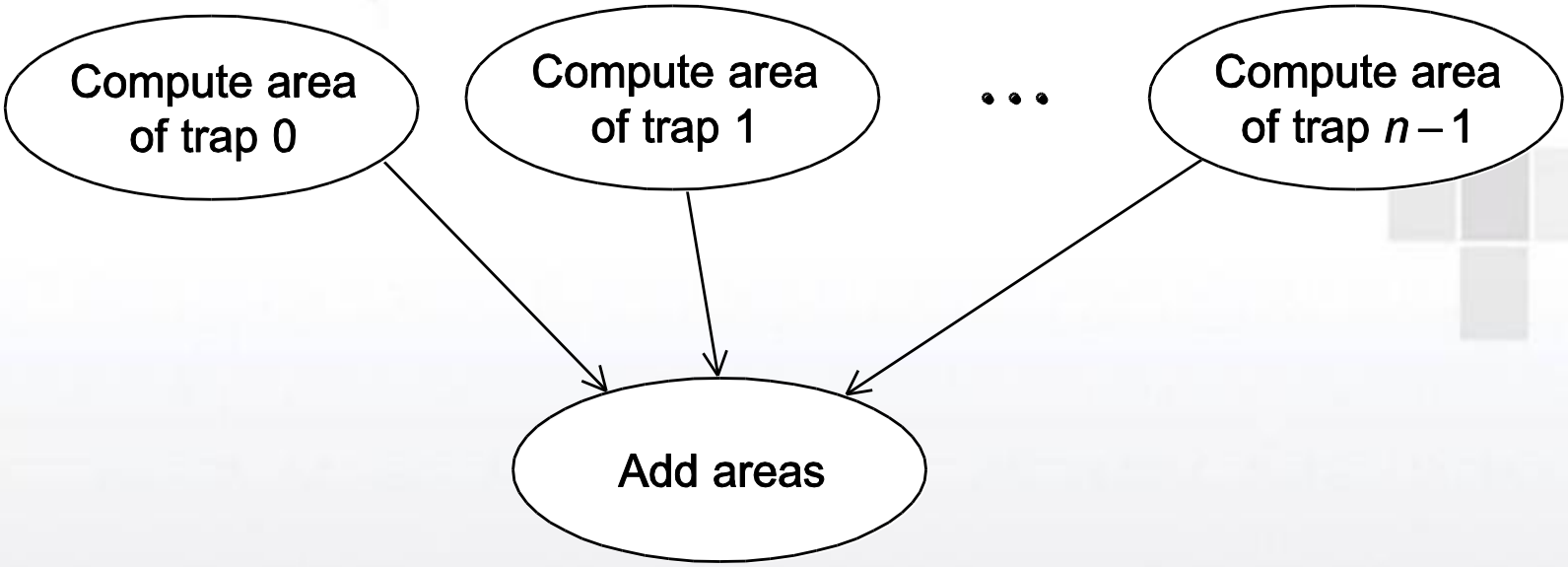
```
/* Input:  a, b, n */
h = (b-a)/n;
approx = (f(a) + f(b))/2.0;
for (i = 1; i <= n-1; i++) {
    x_i = a + i*h;
    approx += f(x_i);
}
approx = h*approx;
```

Foster方法：

1. 将问题的解决方案划分成多个任务；
2. 在任务间识别出需要的通信信道；
3. 将任务聚合成复合任务；
4. 在核上分配复合任务

任务与通信

1. 获取单个梯形区域的面积；
2. 计算所有区域的面积和



并行程序伪代码

```
1     Get a, b, n;
2     h = (b-a)/n;
3     local_n = n/comm_sz;
4     local_a = a + my_rank*local_n*h;
5     local_b = local_a + local_n*h;
6     local_integral = Trap(local_a, local_b, local_n, h);
7     if (my_rank != 0)
8        Send local_integral to process 0;
9     else /* my_rank == 0 */
10       total_integral = local_integral;
11       for (proc = 1; proc < comm_sz; proc++) {
12          Receive local_integral from proc;
13          total_integral += local_integral;
14       }
15    }
16    if (my_rank == 0)
17       print result;
```

MPI点对点通信 第一版

```
1  int main(void) {
2     int my_rank, comm_sz, n = 1024, local_n;
3     double a = 0.0, b = 3.0, h, local_a, local_b;
4     double local_int, total_int;
5     int source;
6
7     MPI_Init(NULL, NULL);
8     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
9     MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
10
11    h = (b-a)/n;           /* h is the same for all processes */
12    local_n = n/comm_sz;   /* So is the number of trapezoids  */
13
14    local_a = a + my_rank*local_n*h;
15    local_b = local_a + local_n*h;
16    local_int = Trap(local_a, local_b, local_n, h);
17
18    if (my_rank != 0) {
19       MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
20             MPI_COMM_WORLD);
```

MPI点对点通信 第一版

```
21      } else {
22         total_int = local_int;
23         for (source = 1; source < comm_sz; source++) {
24            MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
25                  MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26            total_int += local_int;
27         }
28      }
29
30      if (my_rank == 0) {
31         printf("With n = %d trapezoids, our estimate\n", n);
32         printf("of the integral from %f to %f = %.15e\n",
33             a, b, total_int);
34      }
35      MPI_Finalize();
36      return 0;
37   } /*  main  */
```

MPI点对点通信 第一版

```
1  double Trap(
2        double left_endpt  /* in */,
3        double right_endpt /* in */,
4        int    trap_count  /* in */,
5        double base_len    /* in */) {
6     double estimate, x;
7     int i;
8
9     estimate = (f(left_endpt) + f(right_endpt))/2.0;
10    for (i = 1; i <= trap_count-1; i++) {
11       x = left_endpt + i*base_len;
12       estimate += f(x);
13    }
14    estimate = estimate*base_len;
15
16    return estimate;
17 } /*  Trap  */
```

局部变量与全局变量

局部变量：只在使用他们的进程内有效。

全局变量：在所有进程中都有效。

梯形积分法程序汇总哪些变量是局部的，哪些是全局的?

I/O 处理

```
#include <stdio.h>
#include <mpi.h>

int main(void) {
   int my_rank, comm_sz;

   MPI_Init(NULL, NULL);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

   printf("Proc %d of %d > Does anyone have a toothpick?\n",
          my_rank, comm_sz);

   MPI_Finalize();
   return 0;
}  /* main */
```

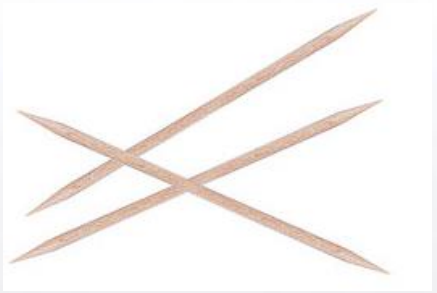
每一个进程都会输出一条信息

思考：结果如何?

MPI点对点通信

```
Proc 0 of 6 > Does anyone have a toothpick?
Proc 1 of 6 > Does anyone have a toothpick?
Proc 2 of 6 > Does anyone have a toothpick?
Proc 4 of 6 > Does anyone have a toothpick?
Proc 3 of 6 > Does anyone have a toothpick?
Proc 5 of 6 > Does anyone have a toothpick?
```

无序的输出



输入

- 大部分 MPI程序只允许 MPI_COMM_WORLD中的process 0访问 stdin.
- Process 0 负责读取数据并将数据发送给其他进程。

```
. . .
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

Get_data(my_rank, comm_sz, &a, &b, &n);

h = (b-a)/n;
. . .
```

使用用户输入数据的函数

```
void Get_input(
      int        my_rank    /* in  */,
      int        comm_sz    /* in  */,
      double*    a_p        /* out */,
      double*    b_p        /* out */,
      int*       n_p        /* out */) {
   int dest;

   if (my_rank == 0) {
      printf("Enter a, b, and n\n");
      scanf("%lf %lf %d", a_p, b_p, n_p);
      for (dest = 1; dest < comm_sz; dest++) {
         MPI_Send(a_p, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
         MPI_Send(b_p, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
         MPI_Send(n_p, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
      }
   } else { /* my_rank != 0 */
      MPI_Recv(a_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
      MPI_Recv(b_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
      MPI_Recv(n_p, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
   }
}  /* Get_input */
```

扩展学习

任务：自学接收发送的其他几种函数

点到点通信共有 12 对，分别对应阻塞通信方式 1 组(4 个)和非阻塞通信方式 2 组(分别为非重复的非阻塞和可重复的非阻塞)，详细分类见表 2-1。

表 2-1　点到点通信分类

分类		发送	接收	说　明
阻塞通信		MPI_Send	MPI_Recv MPI_Irecv MPI_Recv_Init	如果接收动作使用了 MPI_Irecv，MPI_Recv_init，则使用 MPI_Request 对象以及 MPI_Wait，MPI_Waitall，MPI_Waitany，MPI_Waitsome，MPI_Test，MPI_Testany，MPI_Testall，MPI_Testsome 进行测试
		MPI_Bsend		
		MPI_RSend		
		MPI_SSend		
非阻塞通信	非重复	MPI_Isend	MPI_Recv MPI_Irecv MPI_Recv_Init	需用到 MPI_Request 对象，以及 MPI_Wait，MPI_Waitall，MPI_Waitany，MPI_Waitsome，MPI_Test，MPI_Testany，MPI_Testall，MPI_Testsome 进行相关测试与运行
		MPI_IBSend		
		MPI_IRSend		
		MPI_ISSend		
	重复	MPI_Send_Init	MPI_Recv MPI_Irecv MPI_Recv_Init	需用到 MPI_Request 对象，结合 MPI_Start，MPI_Startall，以及 MPI_Wait，MPI_Waitall，MPI_Waitany，MPI_Waitsome，MPI_Test，MPI_Testany，MPI_Testall，MPI_Testsome 进行相关的测试与运行
		MPI_Bsend_Init		
		MPI_RSend_Init		
		MPI_SSend_Init		

其中发送函数模式均为 MPI_??Send，其中 B 表示缓存模式(Buffer)，R 表示就绪模式(Ready)，S 表示同步方式(Synchonous)，I 表示立即发送，即非阻塞方式发送(Imediately)。不带任何修饰，单纯以 MPI_Send 方式发送的称为标准模式。I 可分别与 B，R，S 组合，最后得出如表 2-1 所示的各种通信模式。

随堂练习1

1 strlen(greeting)代替strlen(greeting)+1来计算进程1、2、…、comm_sz-1发送消息的长度，会发生什么情况?

如果用MAX_STRING代替strlen(greeting)+1又会是什么结果?

你可以解释这些结果吗?

```
16    if (my_rank != 0) {
17       sprintf(greeting, "Greetings from process %d of %d!",
18             my_rank, comm_sz);
19       MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
20             MPI_COMM_WORLD);
21    } else {
22       printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
23       for (int q = 1; q < comm_sz; q++) {
24          MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
25             0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26          printf("%s\n", greeting);
27       }
28    }
```

随堂练习1

答:

1) “+1”是因为字符串结束符 ‘\0’(null) 占一个字符。因此，如果使用strlen(greeting)代替strlen(greeting)+1，则终止空字符将不会被发送。所以有的程序可能会发生错误，有的可能会显示正确的结果。

2) 如果我们使用MAX_String而不是strlen(greeting)+1，则程序的输出是相同的。但是，在本例中，我们发送的是整个数组问候语，而不仅仅是问候语的字符串和终止空字符。

随堂练习2

2 梯形积分法程序汇总哪些变量是局部的，哪些是全局的?

答：在梯形积分法程序中局部变量有：

my_rank, local_n, local_a, local_b, local_int, source ,

total_int.

全局变量有：

comm_sz, n, a, b, h.

```
1  int main(void) {
2      int my_rank, comm_sz, n = 1024, local_n;
3      double a = 0.0, b = 3.0, h, local_a, local_b;
4      double local_int, total_int;
5      int source;
6
```

随堂练习3

3 下列哪个命令是MPI分布式并行程序的编译命令_____.

A、mpirun

B、mpicc

C、mpiexec

D、mpigcc

随堂练习4

4 下列关于MPI函数库中MPI_Send()与MPI_Recv()说法不正确的是（）

A、 MPI中消息发送函数MPI_Send()可能是阻塞的

B、 MPI中消息接收函数MPI_Recv()总是阻塞的

C、 MPI中消息发送函数MPI_Send()可能是缓冲的

D、 MPI中消息接收函数MPI_Recv()可能是缓冲的

随堂练习5

5 下列关于通信子的说法不正确的是（）。

A、 进程在某个通信子中具有唯一的标识号

B、 一个进程只能属于一个通信子

C、 进程号为从0开始的连续非负整数

D、 进程在不同通信子中的标识号可以不同

随堂练习6

6 填空

（1）MPI的英文全称是_____.

（2） MPI并行程序中，进程间通信的方式是_____.

（3） MPI中通配符有MPI_ANY_TAG和_____.

上机作业(编程练习)

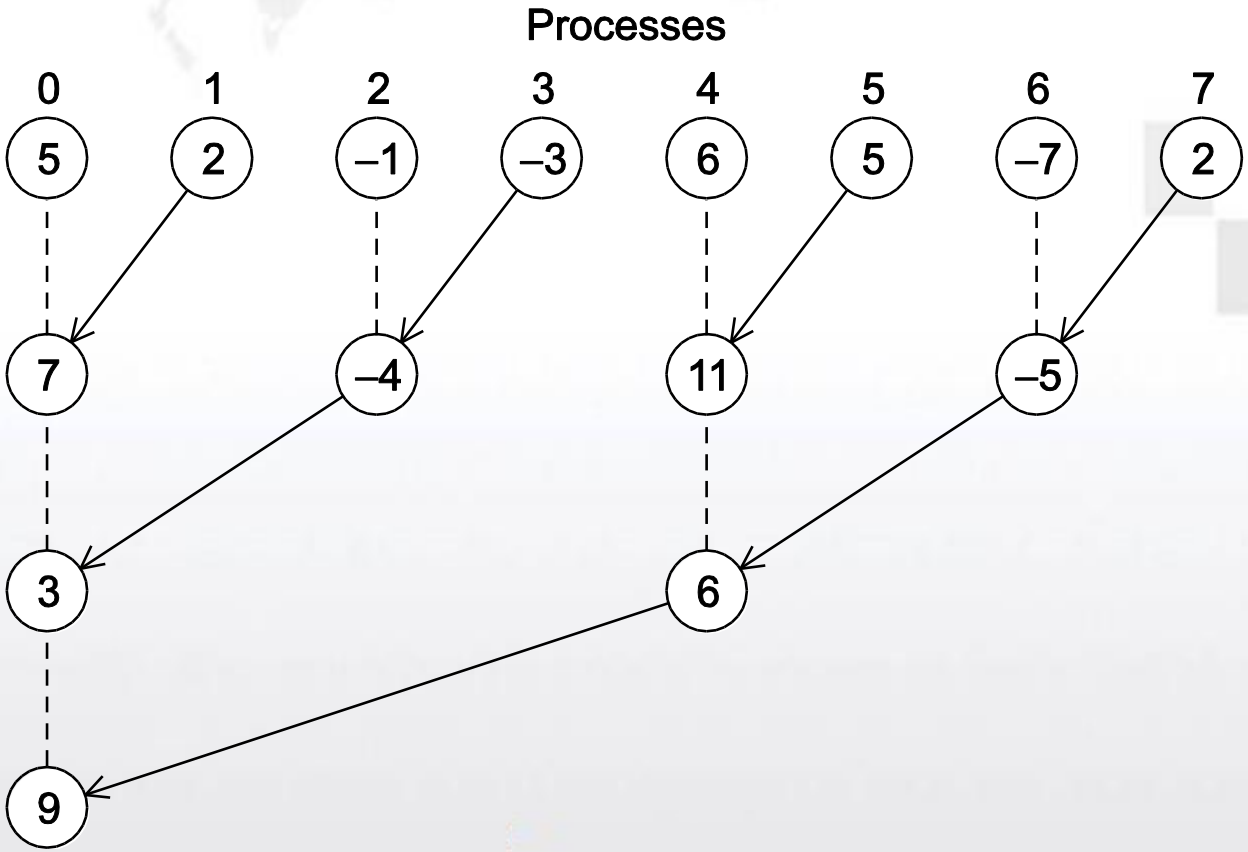
- 编写梯形积分串行程序
- 编写一个完整的并行梯形积分（要求：梯形积分区间的端点和划分的个数是固定.）
- 编写一个完整的并行梯形积分（要求：允许用户输入梯形积分区间的端点和划分的个数.）
- 考虑通配符的影响

上机作业(编程练习)

- 将接收函数中的源进程和标签修改为通配符，观察输出结果并给出解释。
- 改变梯形积分法，使其能够在comm_sz无法被n整除的情况下，正确估算积分值（假设n>=comm_sz）

上机作业(编程练习)

编写一个MPI程序，采用如下的树形通信结构来计算全局总和。考虑comm_sz是2的幂的特殊情况.



实验报告要求

- 特别注意Word 格式排版
- 上机按照实验报告模板进行书写

结束!

思考

上述讲解的点点通信并没有采用更公平的工作分配方式，思考如何通过上述方法实现树形结构的通信?