平行運算簡介／實例操作
企鵝也會的 MPICH

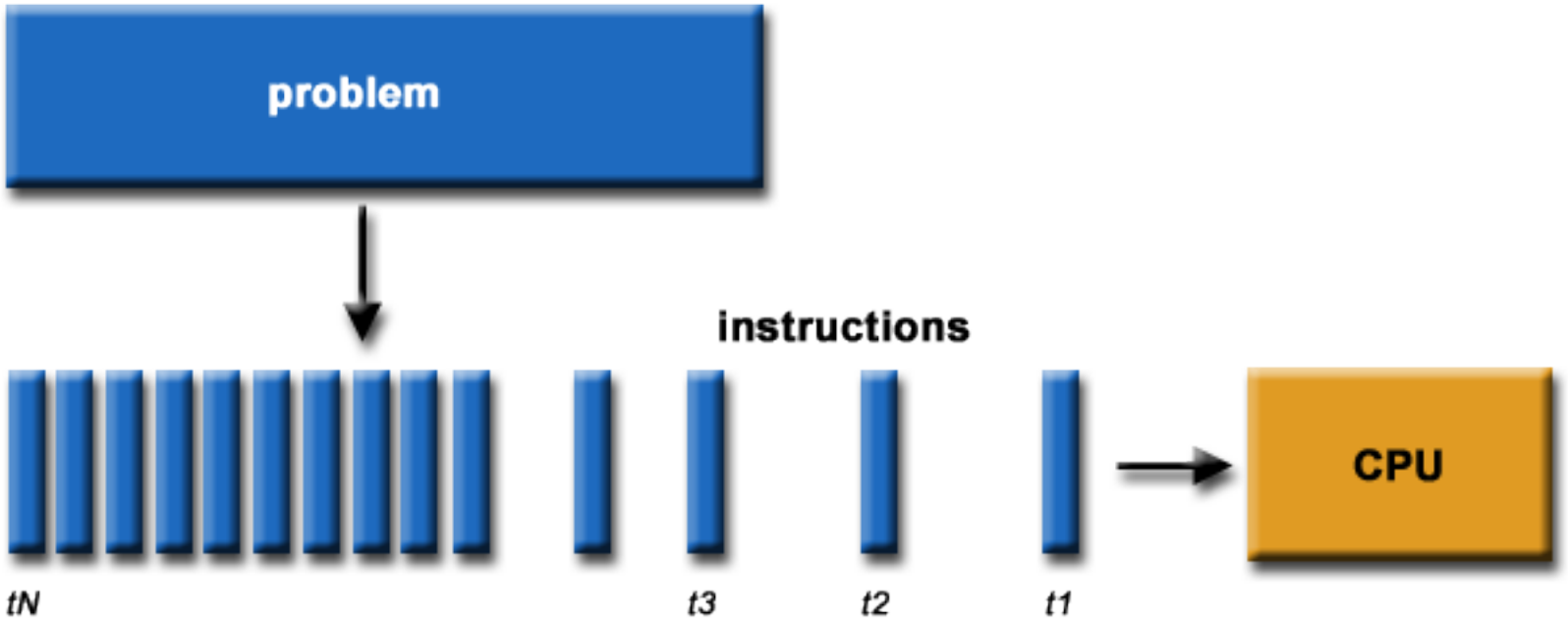
研究員：鄧偉華

wade@nchc.org.tw

財團法人國家實驗研究院
國家高速網路與計算中心
National Center for High-Performance Computing

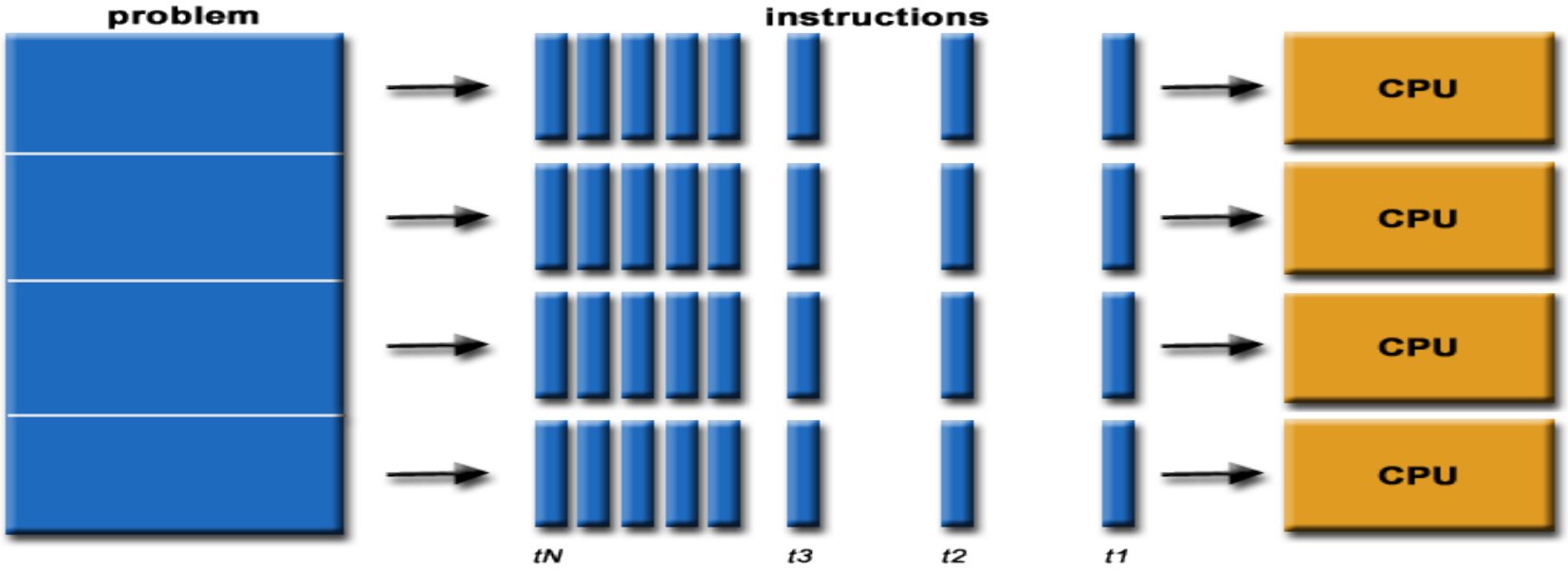
什麼是平行計算

- 傳統：
  - 單一程序
  - 單一 CPU



什麼是平行計算 (續)

- 平行計算
  - 程序切割
  - 多 CPUs



爲什麼要平行計算

- 簡省時間
- 解決大型問題
- 即時性
- 使用更多來自網路上的資源
- 使用大量「便宜」 PCs 取代超級電腦
- 記憶體不足

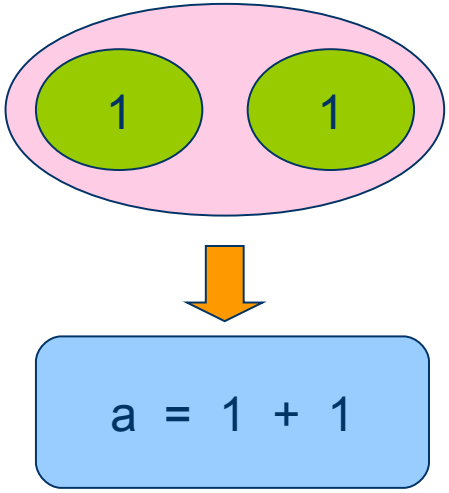
平行計算種類

- Flynn's taxonomy
- 多處理器架構

	單一程序 Single Instruction	多程序 Multiple Instruction
單一資料 Single Data	SISD	MISD
多資料 Multiple Data	SIMD	MIMD

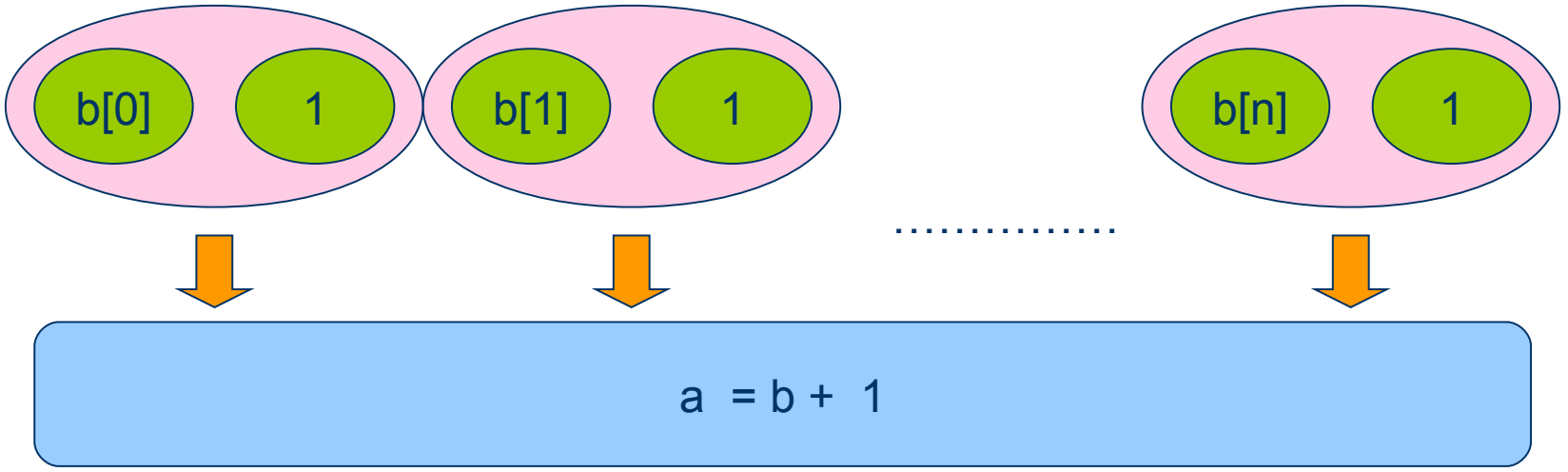
Single Instruction, Single Data (SISD)

- 非平行計算
- 單一程序
- 單一資料
- ex ： a = 1 + 1



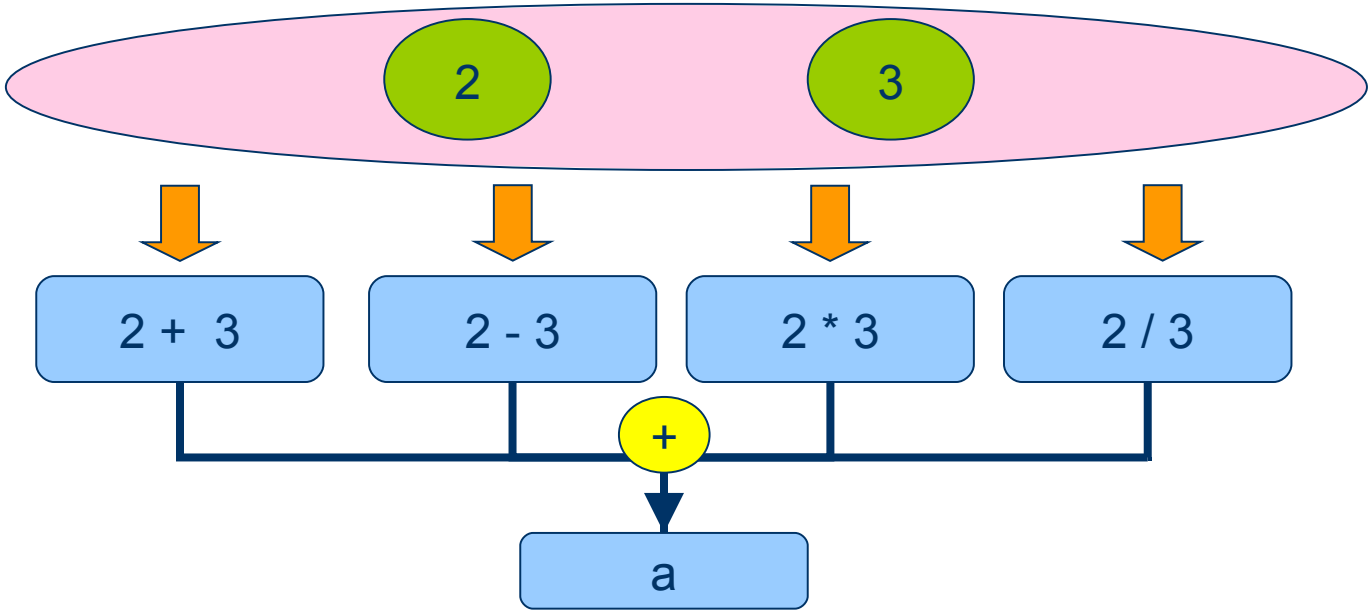
Single Instruction, Multiple Data(SIMD)

- 平行計算
- 單一程序
- 多資料
- ex ： a = b + 1 ， a, b 是二組陣列，則同一時間就可以計算完成。而不需要算完 a[0] = b[0] + 1 ，再算 a[1] = b[1] + 1… 再算 a[n] = b[n] + 1 。



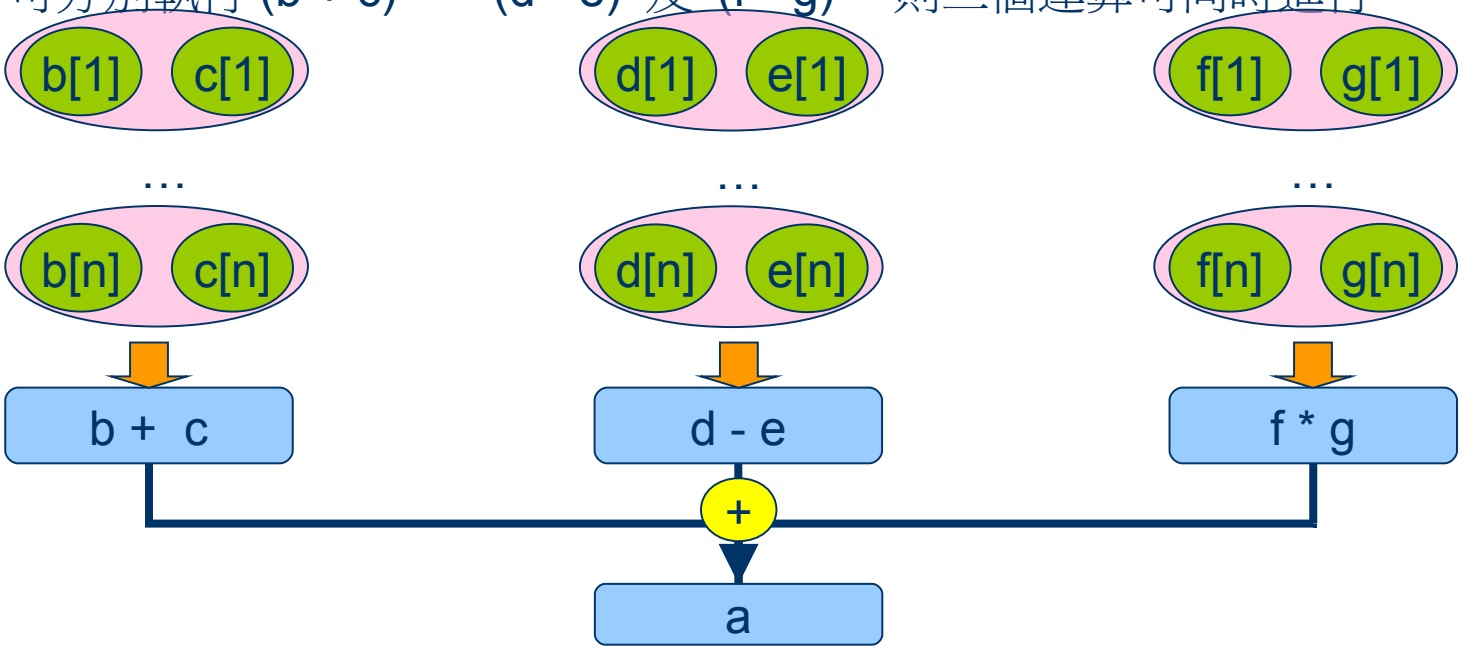
Single Instruction, Multiple Data(MISD)

- 平行計算
- 多程序
- 單一資料
- a = (2 + 3) + (2 - 3) + (2 * 3) + (2 / 3)



Multiple Instruction, Multiple Data(MIMD)

- 平行計算
- 多程序
- 多資料
- a = (b + c) + (d - e) + (f * g)， a, b, c, d, e, f, g 爲陣列如果有三臺機器可分別執行 (b + c) 、 (d - e) 及 (f * g)，則三個運算可同時進行。



如何執行 MPICH 平行程式

- 啓動 MPD
  - mpdboot -n 4 -f machine_file
  - -n how many mpds to start
  - -f hostsfile
- 列出所有 nodes
  - mpdtrace
- 執行
  - mpiexec -n 12 ./mpi/a.out
- 結束 MPD
  - mpdallexit

MPICH 設計注意事項

- MPICH 屬於 SIMD 架構
- 指令及資料量切割
  - 優：有助於平行運算
  - 缺：增加傳輸次數及傳輸量

如何使用載入／啓動 MPICH

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

main (int argc, char **argv)
{
  MPI_Init(&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD,  &numprocs);
  MPI_Comm_rank (MPI_COMM_WORLD,  &myid);
  . . .
  . . .
  MPI_Finalize();
  return 0;
}
```

啓動該程式在多個 CPU 上的平行計算工作

得知參與平行計算的 CPU 個數 (nproc)

得知我是第幾個 CPU (myid) from 0

結束平行計算工作

HELLO MPICH

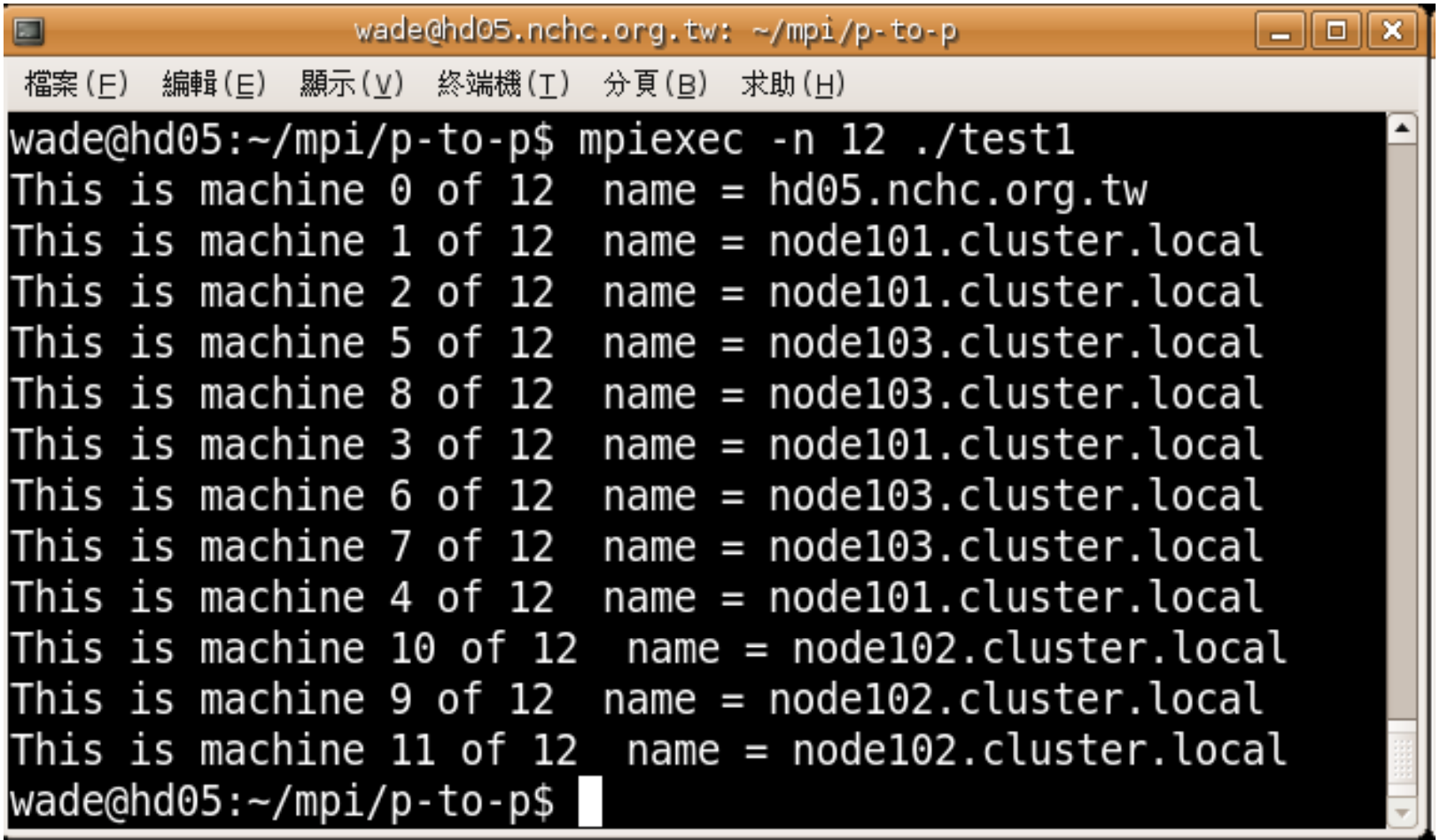
目標：

- 每個 node 將自己的 id 印出，並且將所有的參與活動的 node 總數也印出，顯示出自己的主機名稱。

HELLO MPICH

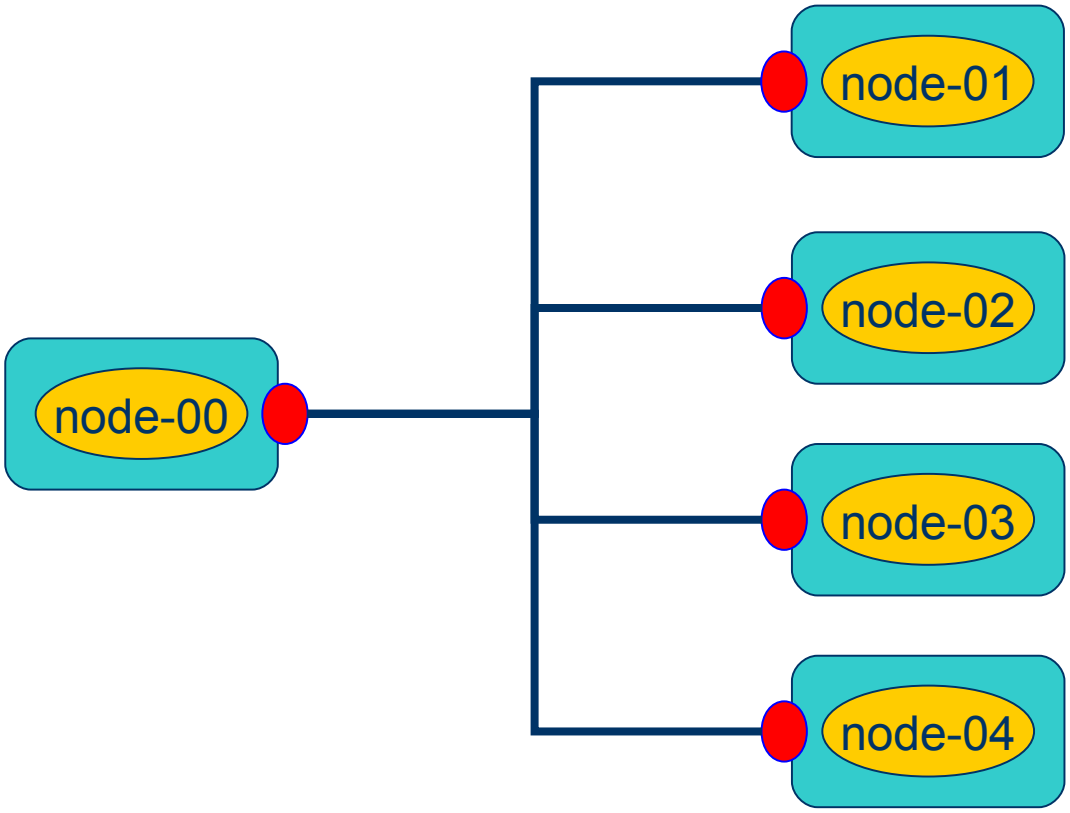
```
#include <stdio.h>
#include <mpi.h>
main (int argc, char **argv) {
 int rank, size, len;
 char name[MPI_MAX_PROCESSOR_NAME];
 MPI_Init(&argc, &argv);
 int myid, numprocs;

 /*  取得  node  總數  */
 MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
 /*  取得本身  node id / rank */
 MPI_Comm_rank(MPI_COMM_WORLD,&myid);
 /*  取得本身  host name */
 MPI_Get_processor_name(name, &len);
 printf("This is machine %d of %d name = %s\n“, myid, numprocs, name);

 MPI_Finalize();
 }
}
```

結果

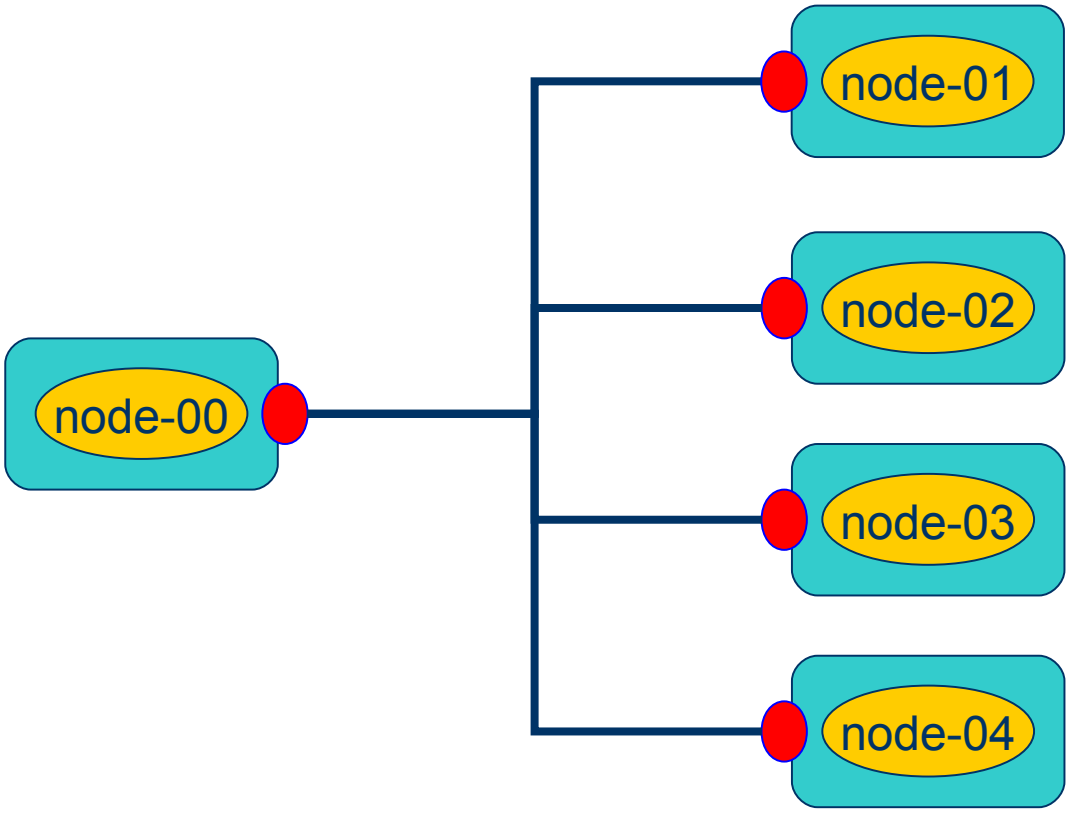

wade@hd05.nchc.org.tw: ~/mpi/p-to-p
檔案(F) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
wade@hd05:~/mpi/p-to-p$ mpiexec -n 12 ./test1
This is machine 0 of 12 name = hd05.nchc.org.tw
This is machine 1 of 12 name = node101.cluster.local
This is machine 2 of 12 name = node101.cluster.local
This is machine 5 of 12 name = node103.cluster.local
This is machine 8 of 12 name = node103.cluster.local
This is machine 3 of 12 name = node101.cluster.local
This is machine 6 of 12 name = node103.cluster.local
This is machine 7 of 12 name = node103.cluster.local
This is machine 4 of 12 name = node101.cluster.local
This is machine 10 of 12 name = node102.cluster.local
This is machine 9 of 12 name = node102.cluster.local
This is machine 11 of 12 name = node102.cluster.local
wade@hd05:~/mpi/p-to-p$

Point to point communication

Point to point communication



Point to point communication



MPI_Send

MPI_Send ((void *)&data, icount, DATA_TYPE, idest, itag, MPI_COMM_WORLD)

- data 要送出去的資料起點，可以是純量 (scalar) 或陣列 (array) 資料。
- icount 要送出去的資料數量，當 icount 的值大於一時， data 必須是陣列。
- DATA_TYPE 要送出去的資料類別， MPI 內定的資料類別。
- idest 是收受資料的 CPU id 。
- itag 要送出去的資料標籤。
- MPI_COMM_WORLD 通信域。

MPI_Recv

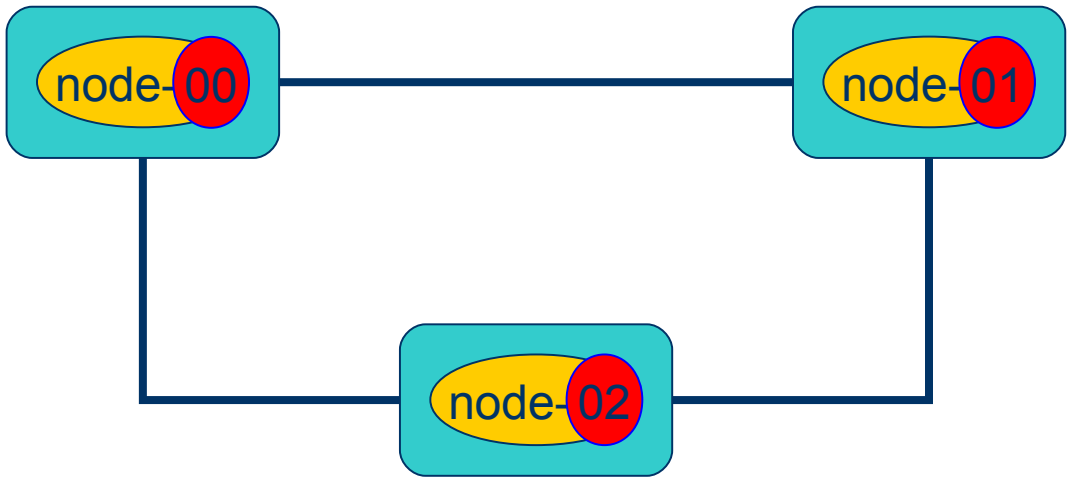
MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

- buf 要接收的資料起點，可以是純量 (scalar) 或陣列 (array) 資料。
- icount 要接收的資料數量，當 icount 的值大於一時， data 必須是陣列。
- DATA_TYPE 要接收的資料類別， MPI 內定的資料類別。
- source 是收受資料的 CPU id 。
- itag 要接收的資料標籤。
- MPI_COMM_WORLD 通信域。

範例

目標：

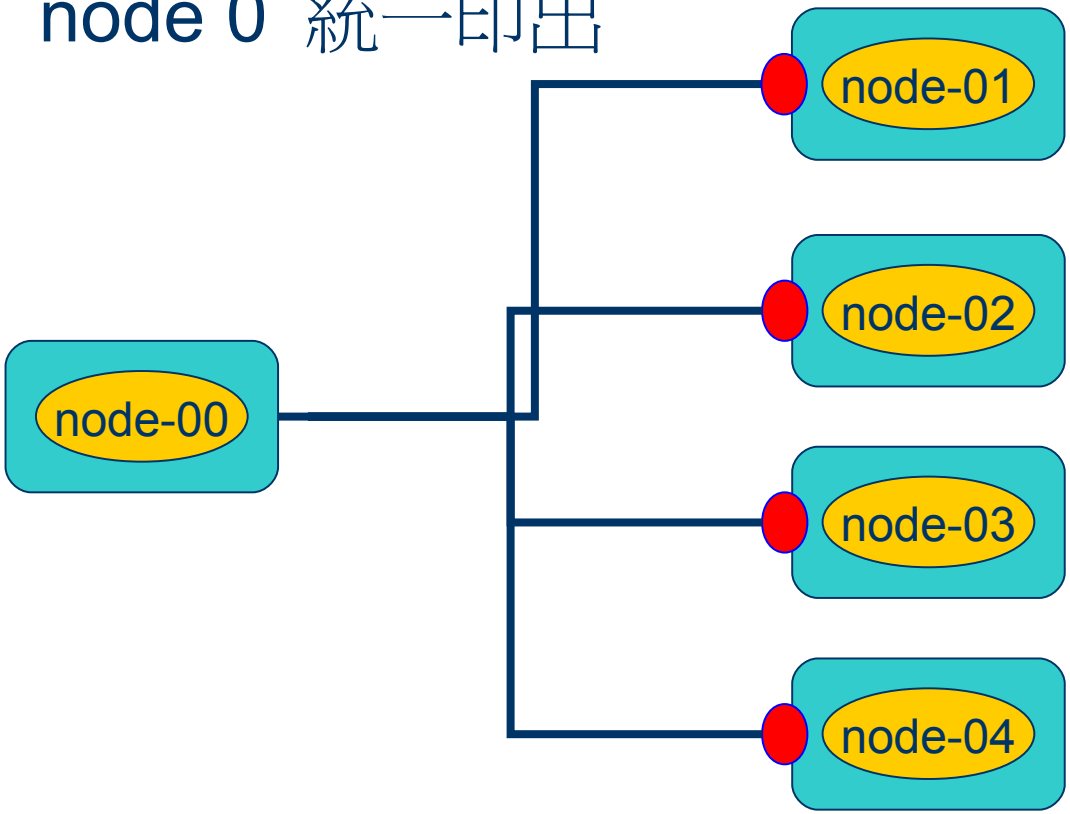
- 每個 node 將自己的 id 資訊送給下一個 node 。



```
#include <mpi.h>
#include <stdio.h>
main(int argc,char **argv) {
  int n, myrank, numprocs;
  MPI_Status status;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

  /* node 0 will send the first message */
  if(myrank == 0) {
    n = myrank;
    MPI_Send(&n, 1, MPI_INT, 1, 99, MPI_COMM_WORLD);
    printf("[Ndde %d]「 %d 」 >> [Node %d]\n\n", myrank, n, myrank+1);
  }

  /* node 1 to node n-2 will send message to the next node */
  if(myrank>0 && myrank<numprocs-1) {
    MPI_Recv(&n, 1, MPI_INT, myrank-1, 99, MPI_COMM_WORLD, &status);
    printf("[Node %d] << 「 %d 」 [Node %d]\n", myrank, n, status.MPI_SOURCE);
    n = myrank; MPI_Send(&n, 1, MPI_INT, myrank+1, 99, MPI_COMM_WORLD);
    printf("[Ndde %d]「 %d 」 >> [Node %d]\n\n", myrank, n, myrank+1);
  }

 /* the final node n-1 will not send any message but receive*/
  if(myrank==numprocs-1) {
    MPI_Recv(&n, 1, MPI_INT, myrank-1, 99, MPI_COMM_WORLD, &status);
    printf("[Node %d] << 「 %d 」 [Node %d]\n", myrank, n, status.MPI_SOURCE);
    }

  MPI_Finalize();
}
```

動手作

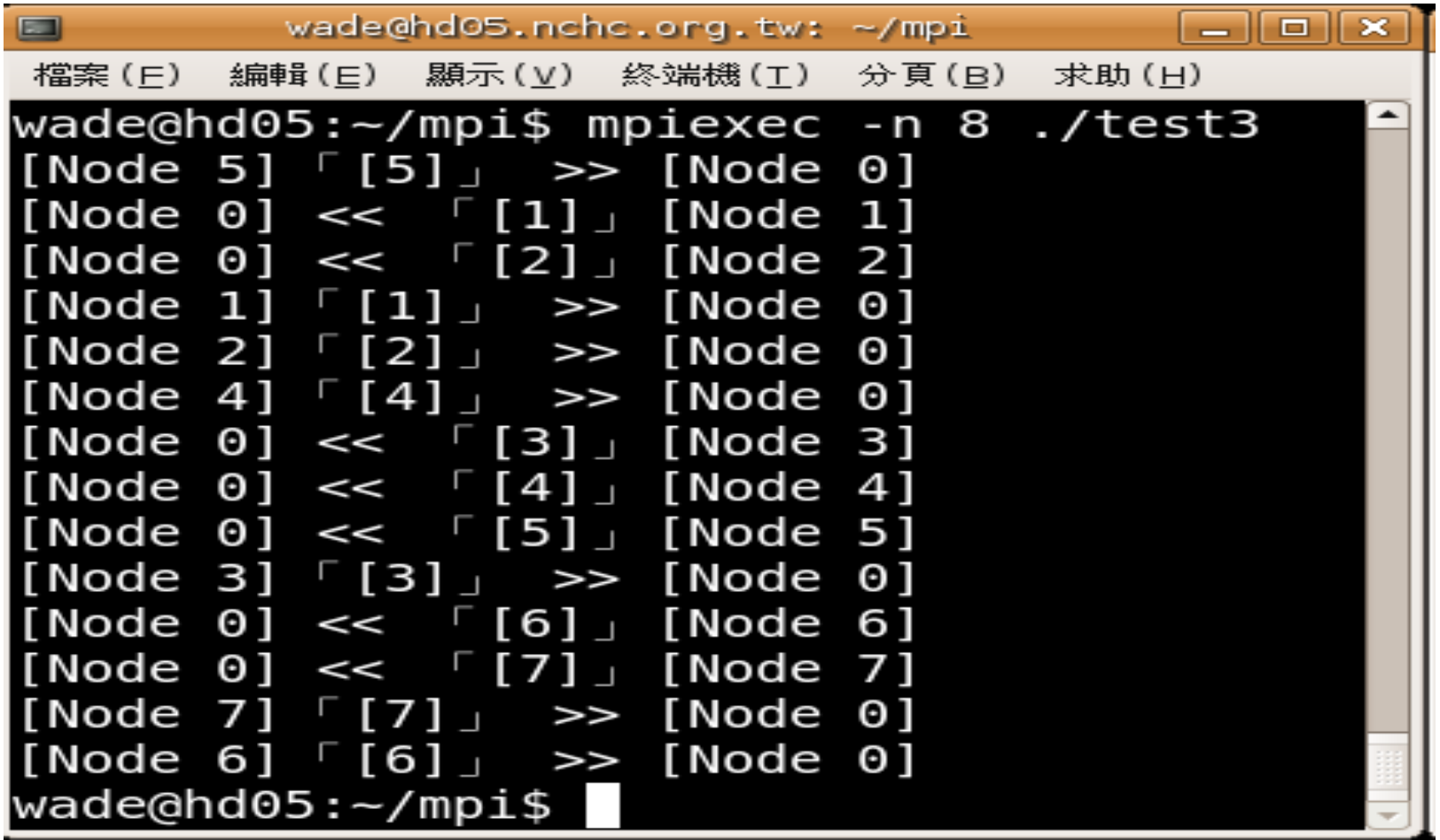
- 每個 node 將訊息傳送給 node 0 ，由， node 0 統一印出



提示

```
main(int argc, char **argv){
  /* Node 0 will do the following */
  if(myrank == 0) {
    /* receive messages from other nodes */
    ……
    MPI_Recv();
  }

  /* other Nodes will do the following */
  if(myrank != 0) {
    /* send node's rank to Node 0 */
    MPI_Send(message, 20, MPI_CHAR, 0, 99,
     MPI_COMM_WORLD);
  }
```

結果


wade@hd05.nchc.org.tw: ~/mpi
檔案(F) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
wade@hd05:~/mpi$ mpiexec -n 8 ./test3
[Node 5] 「[5]」 >> [Node 0]
[Node 0] << 「[1]」 [Node 1]
[Node 0] << 「[2]」 [Node 2]
[Node 1] 「[1]」 >> [Node 0]
[Node 2] 「[2]」 >> [Node 0]
[Node 4] 「[4]」 >> [Node 0]
[Node 0] << 「[3]」 [Node 3]
[Node 0] << 「[4]」 [Node 4]
[Node 0] << 「[5]」 [Node 5]
[Node 3] 「[3]」 >> [Node 0]
[Node 0] << 「[6]」 [Node 6]
[Node 0] << 「[7]」 [Node 7]
[Node 7] 「[7]」 >> [Node 0]
[Node 6] 「[6]」 >> [Node 0]
wade@hd05:~/mpi$

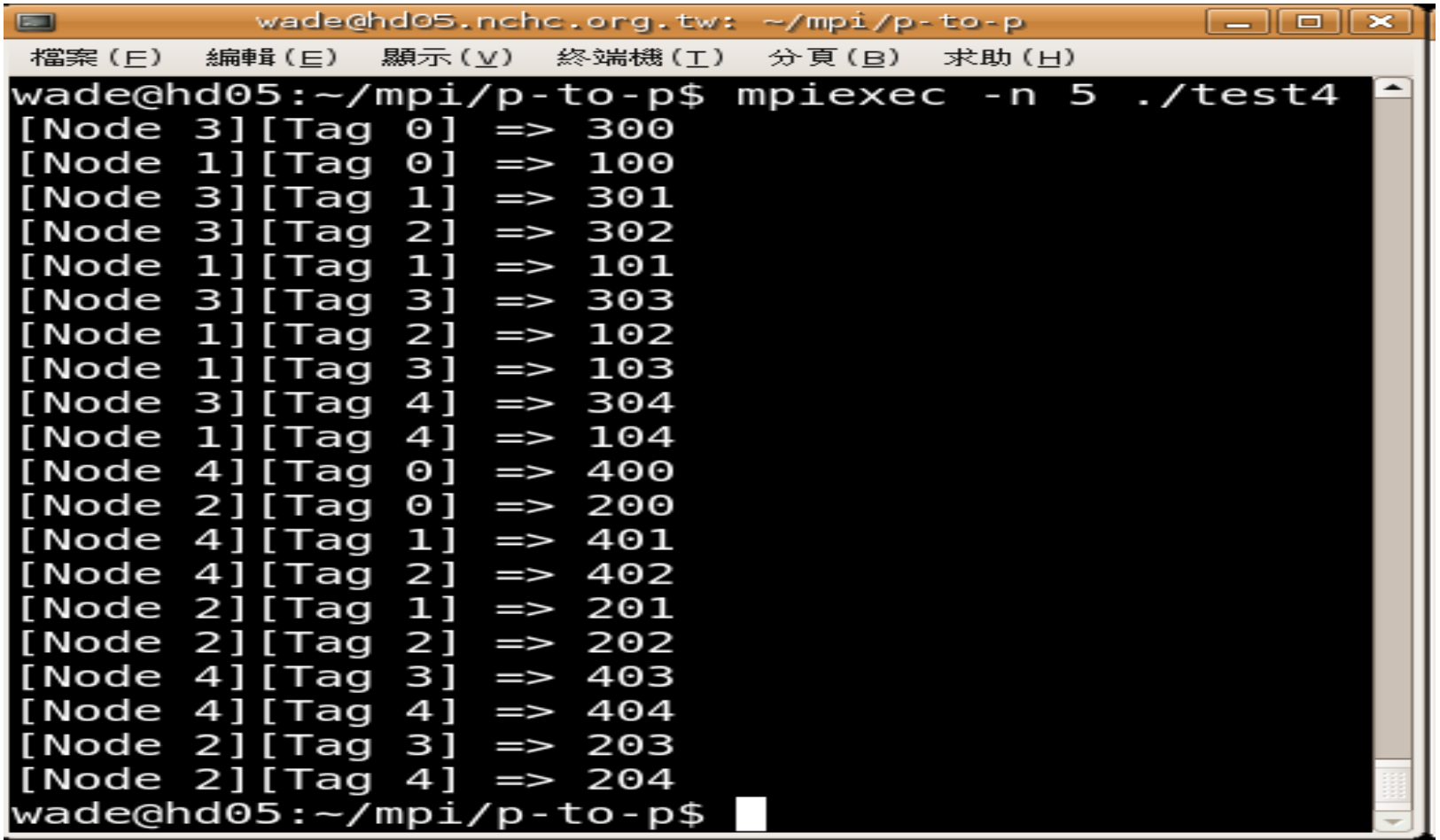
再動手作

- 讓 node 0 可以接受來自任何 node 的訊息，每個 node 將訊息標上不同 tag (0, 1, 2...) 後傳給 node 0

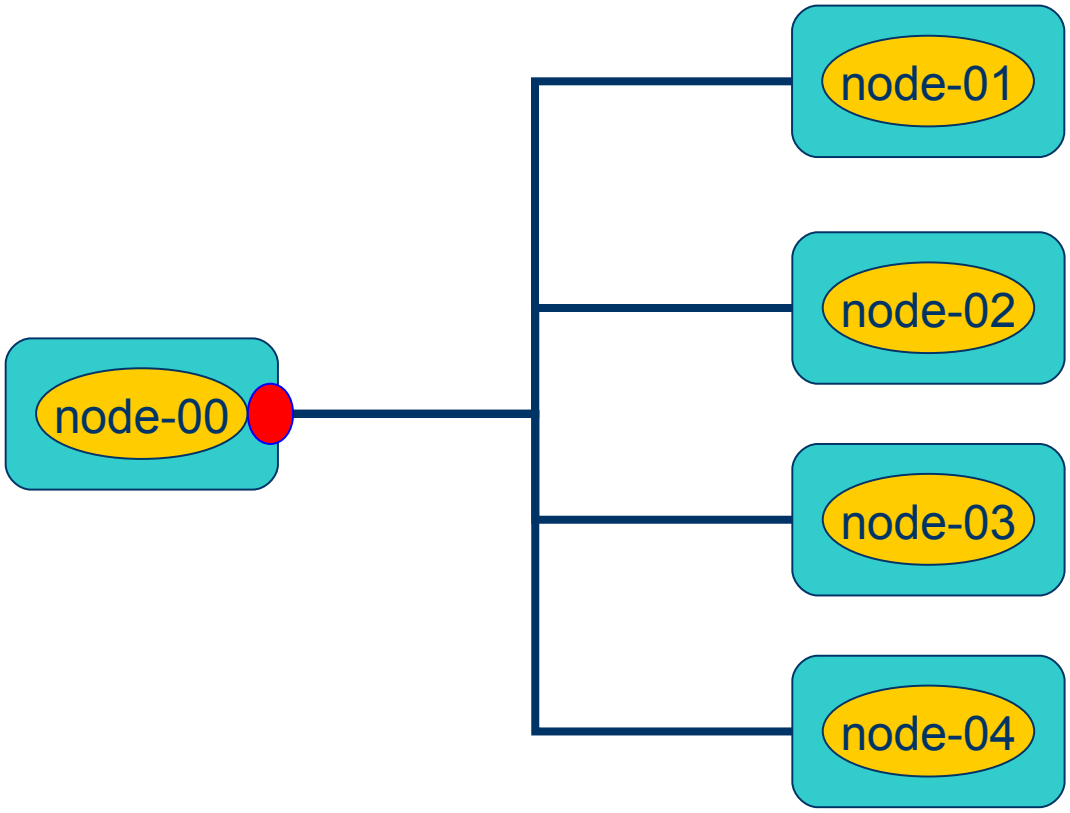
提示

- MPI_ANY_SOURCE 接收來自任何 node 訊息。
- MPI_ANY_TAG 接收任何 tag 。

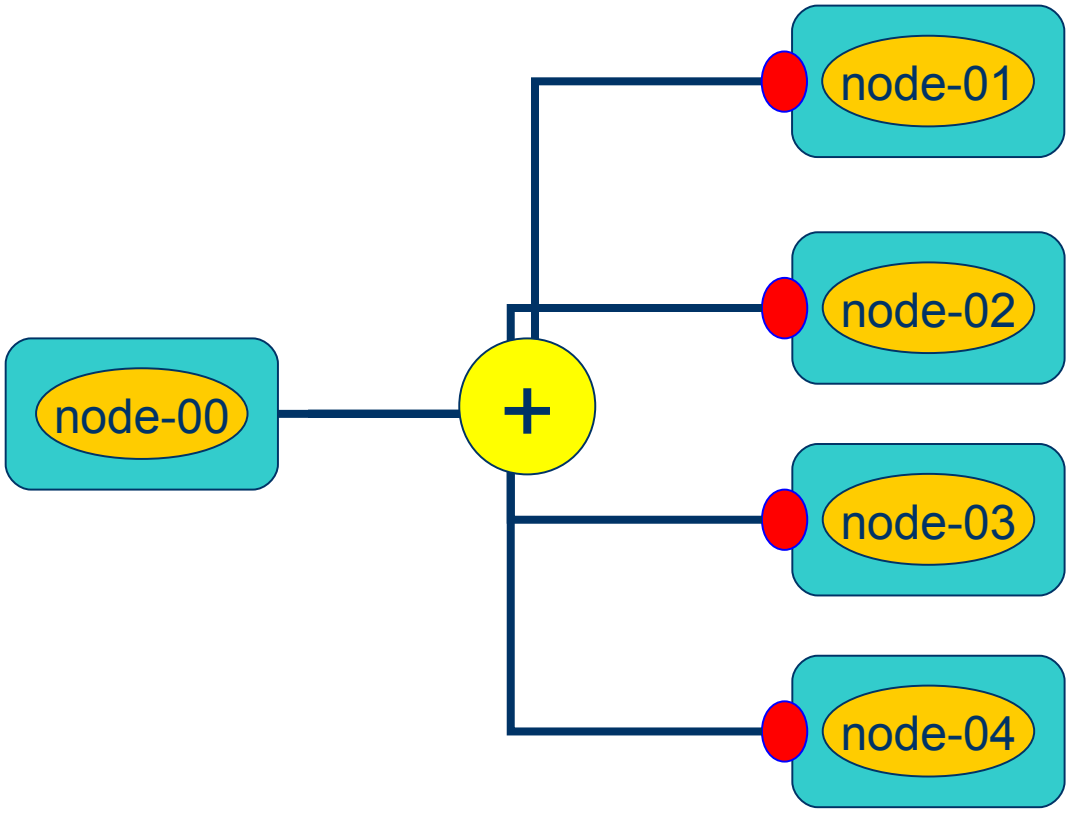
結果


wade@hd05.nchc.org.tw: ~/mpi/p-to-p
檔案(F) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
wade@hd05:~/mpi/p-to-p$ mpiexec -n 5 ./test4
[Node 3][Tag 0] => 300
[Node 1][Tag 0] => 100
[Node 3][Tag 1] => 301
[Node 3][Tag 2] => 302
[Node 1][Tag 1] => 101
[Node 3][Tag 3] => 303
[Node 1][Tag 2] => 102
[Node 1][Tag 3] => 103
[Node 3][Tag 4] => 304
[Node 1][Tag 4] => 104
[Node 4][Tag 0] => 400
[Node 2][Tag 0] => 200
[Node 4][Tag 1] => 401
[Node 4][Tag 2] => 402
[Node 2][Tag 1] => 201
[Node 2][Tag 2] => 202
[Node 4][Tag 3] => 403
[Node 4][Tag 4] => 404
[Node 2][Tag 3] => 203
[Node 2][Tag 4] => 204
wade@hd05:~/mpi/p-to-p$

Collective communication



Collective communication



MPI_Reduce

MPI_Reduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

- sendbuf address of send buffer (choice)
- count number of elements in send buffer (integer)
- datatype data type of elements of send buffer (handle)
- op reduce operation (handle)
- root rank of root process (integer)
- comm communicator (handle)

計算 1 + 2 + 3 + ... + 100

```
#include <stdio.h>
#include <mpi.h>
main (int argc, char **argv) {
  int rank, size, i; int myid, numprocs; int myTotal = 0;
  int Total = 0; MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  for(i = 1; i <= 100; i++) {
    if((i % numprocs) == myid) {
      myTotal += i;
    }
  MPI_Reduce(&myTotal, &Total, 1, MPI_INT, MPI_SUM, 0,MPI_COMM_WORLD);
  }
  if(myid==0) {
    printf("Total = %d\n", Total);
  }
  MPI_Finalize();
}
```

結果


wade@hd05.nchc.org.tw: ~/mpi
檔案(F) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
wade@hd05:~/mpi$ mpiexec -n 4 ./sum_1_to_n
Total = 5050
wade@hd05:~/mpi$

動手作

- 計算 Fibonacci number 從第 1 項加至第 n 項總合。
- Fibonacci number
  - f(n) = f(n - 1) + f(n - 2)
  - f(0) = 1, f(1) = 1
  - 1, 1, 2, 3, 5, 8, ...

參考

- http://www-unix.mcs.anl.gov/mpi/mpich1/docs
- http://www-unix.mcs.anl.gov/mpi/www/www3/
- http://www.mpi-forum.org/
- https://trac.nchc.org.tw/grid/wiki/mpich