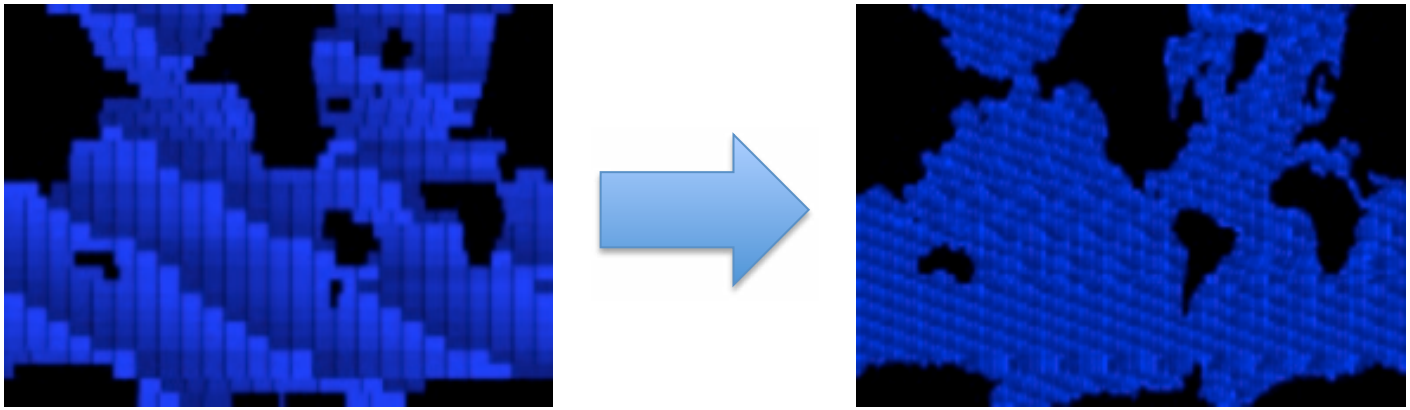
総合演習第1ラウンド資料
No. 1

滝澤 真一朗

スパコンてなに？

- スパコン（スーパーコンピュータ）とは
  - 内部の演算処理速度がその時代の一般的なコンピュータより非常に高速なコンピュータ
- スパコンの主な用途
  - 天文学、金融工学、核融合シミュレーション
  - 高速、高精度に計算



パソコンのなかみ

アプリケーション
ソフトウェア

- 文書作成
- 音楽、動画再生
- インターネット
- プログラム開発
- シミュレーション

OS

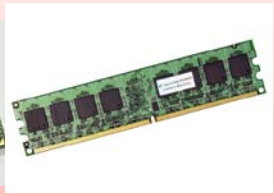
Windows

Mac OS

Linux

ハードウェア

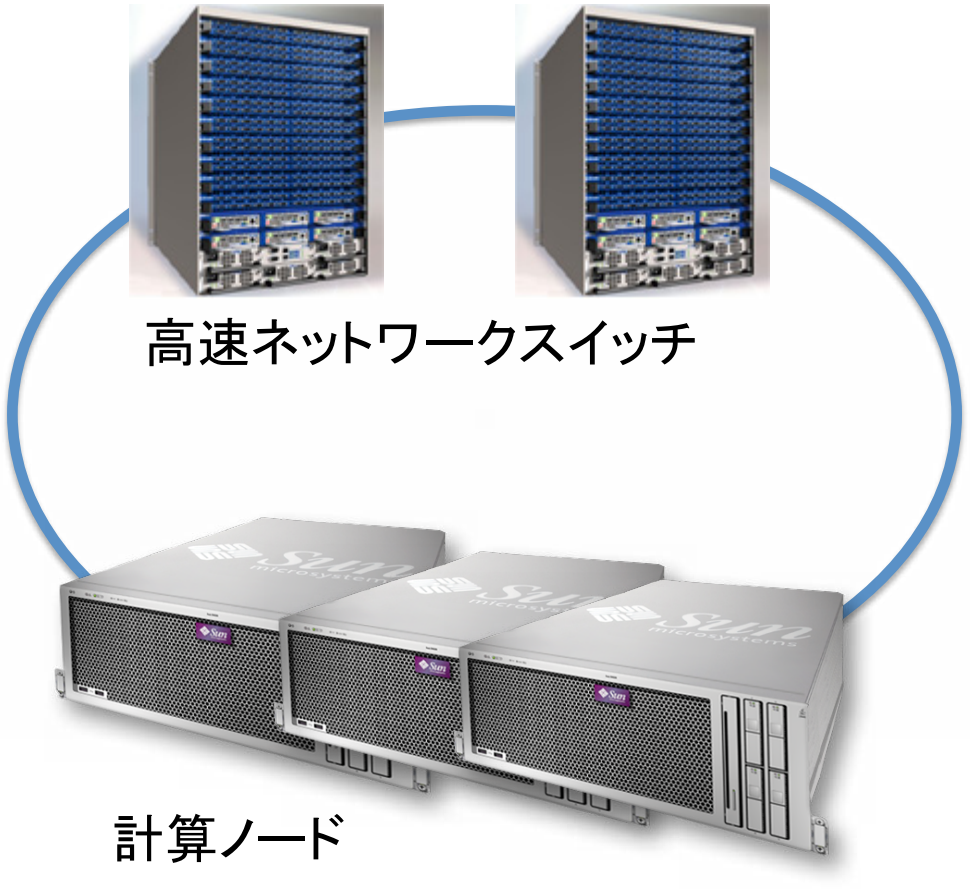
CPU



メモリー

ハードディスク

スパコンのなかみ
- ハードウェア -



- 計算ノード
  - パソコンの10数倍の高性能コンピュータ
  - 多くのCPU、メモリ、ハードディスクを搭載
- 高速ネットワーク
  - ノード間: ～20000Mbps
  - パソコン同士の直接通信: ～1000Mbps
  - インターネット接続: ～100Mbps

東京工業大学 学術国際情報センター TSUBAME Grid Cluster

655 nodes

Sun Fire X4600

CPU : AMD Opteron(Dual Core)
5,240CPU / 10,480Core
メモリ : 21.4TB
演算性能 : 50TFlops(Peak)
38.18TFlops(Linpack)

ClearSpeed CSX600 SIMD accelerator

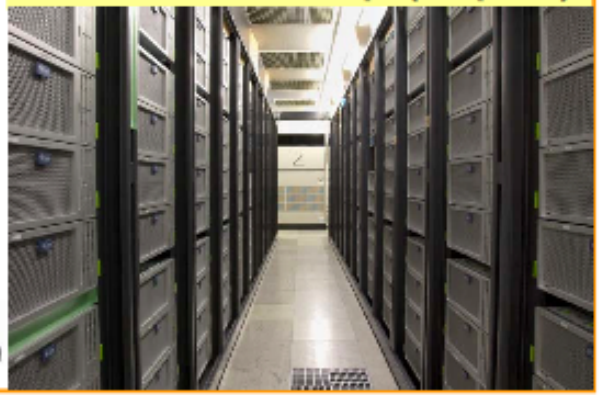
演算性能 : 35TFlops(Peak)/360slots

分子動力学アクセラレータ:
15TFlops(Peak)

高速フーリエ変換加速装置:
59TFlops(Peak)

TSUBAME Grid Cluster

総合演算性能: 124TFlops(Peak)
77.48TFlops(Linpack)



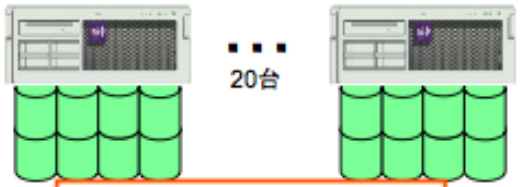
100ギガビット級ネットワーク装置

InfiniBand Network Voltaire ISR 9288 ×8

24Gbps
(片方向)

SuperTitanet

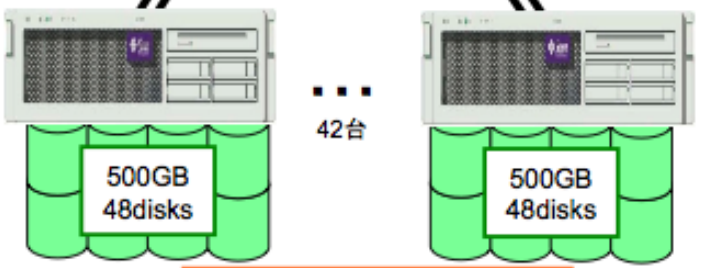
10Gbps
(片方向)



20台

Sun Fire X4500

高度研究用大規模ストレッジ基盤
総物理容量: 0.5PB



42台

500GB
48disks

500GB
48disks

Sun Fire X4500

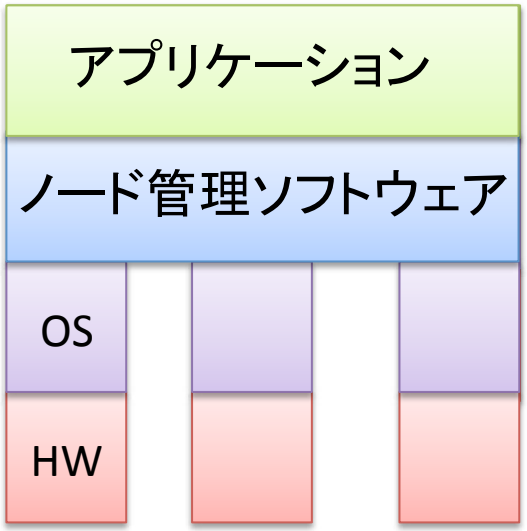
物理容量: 1PB

ペタバイト級ストレッジサーバ
総物理容量: 1.1PB

NEC iStorage S1800AT

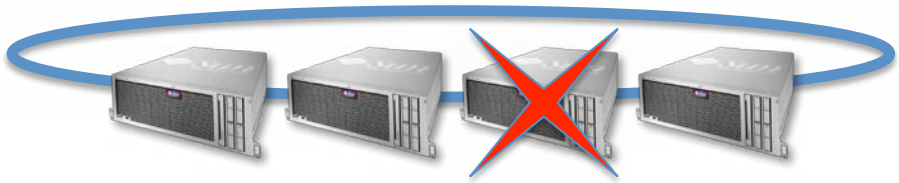
物理容量: 0.1PB

スパコンのなかみ
- ソフトウェア -



- 高速通信ライブラリ
  - 全ノード同時通信も
- 計算割り当てスケジューラ
  - どのアプリケーション(計算)を
    どのノードで実行するか
- 故障対応
  - 故障したノードで実行していた
    計算の続きを別ノードで再開

スケジューラ



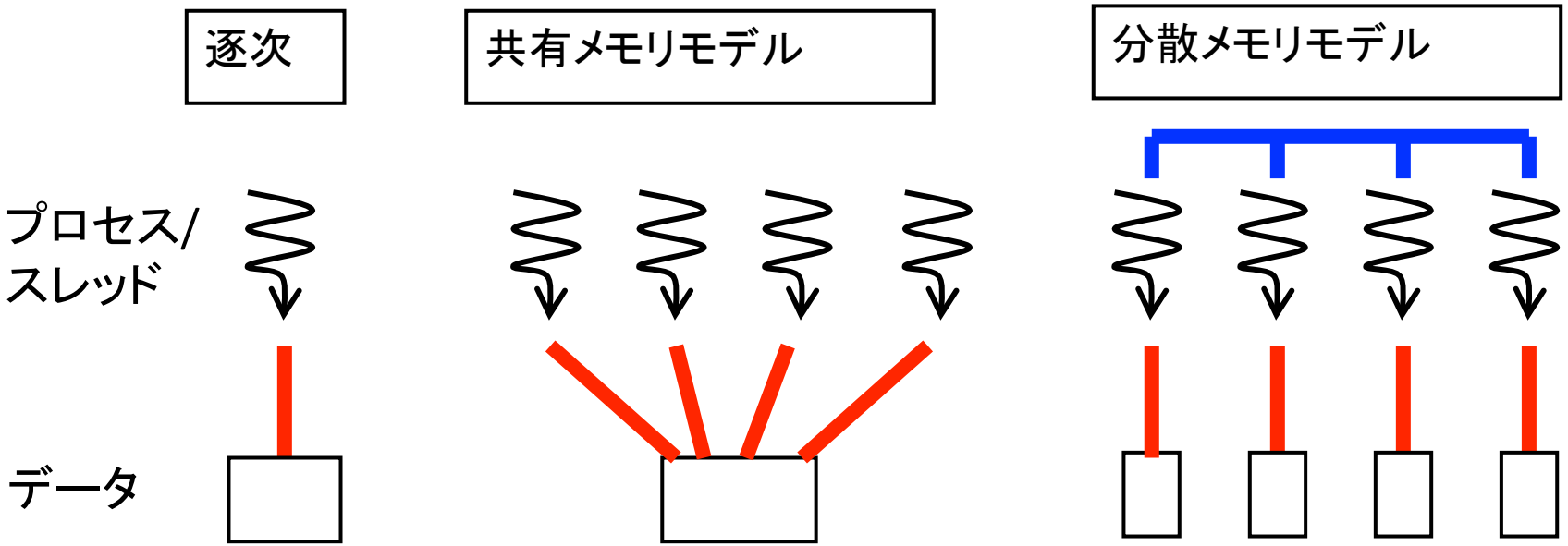
スパコンプログラミングのキモ

- 結局スパコンはたくさんのコンピュータをまとめたシステム
  - 1台のコンピュータ用に作られたプログラムをそのまま動かしても効果なし
- たくさんのCPU、ノードを使うようにプログラムを書き換える必要がある
  - 個々のCPUに同時に異なる計算をさせる
  - 個々のCPUに同時に異なるデータを処理させる

並列プログラミング

並列プログラミングモデルの分類

- メモリモデルによる分類



スレッド達が共通の
データにアクセス可能
- OpenMP(言語拡張)
- pthread(ライブラリ)

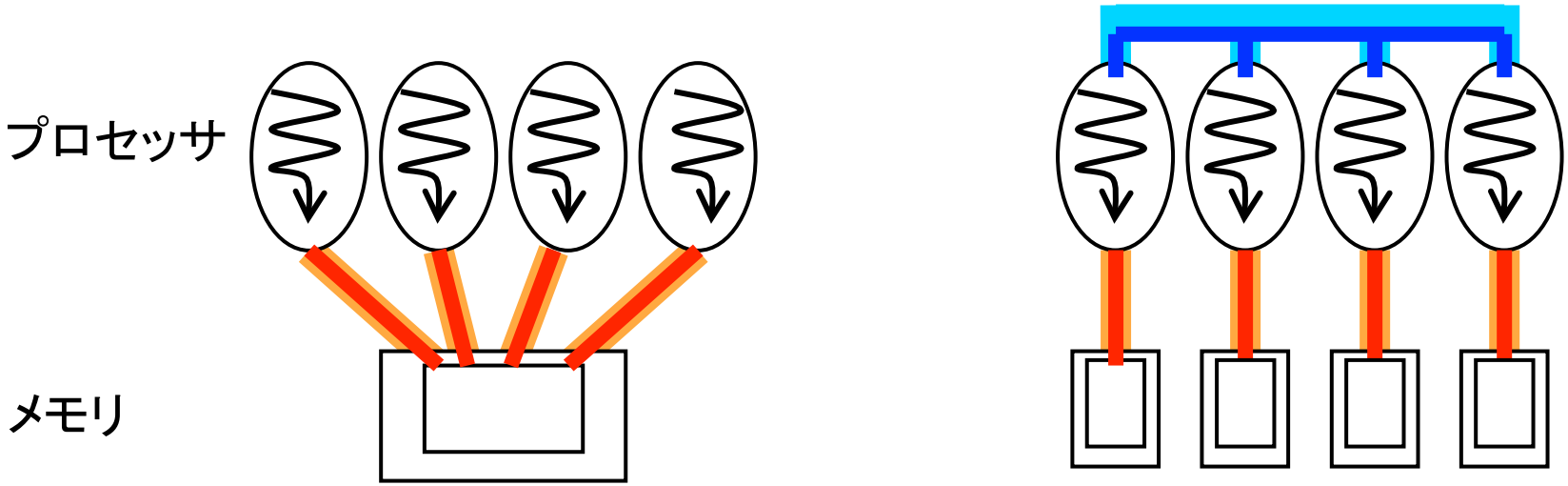
プロセス間では通信
が必要
- MPI (ライブラリ)
- socket (ライブラリ)

並列アーキテクチャの分類

- メモリアーキテクチャによる分類
- どちらもMIMD(multiple instruction, multiple data)

共有メモリアーキテクチャ

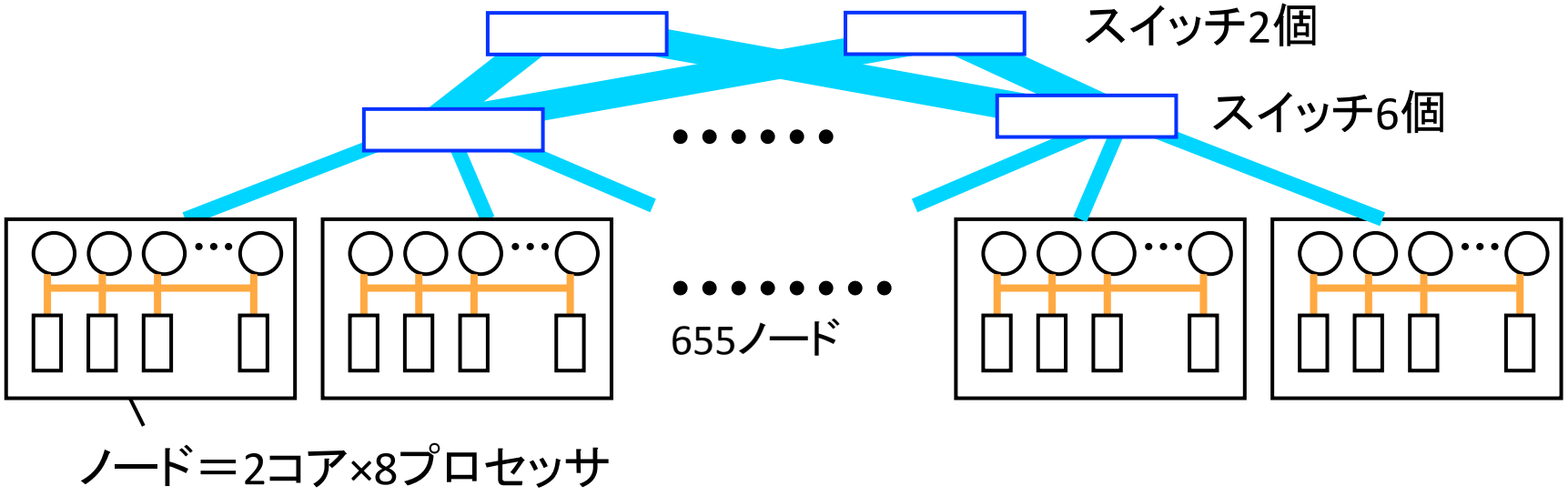
分散メモリアーキテクチャ



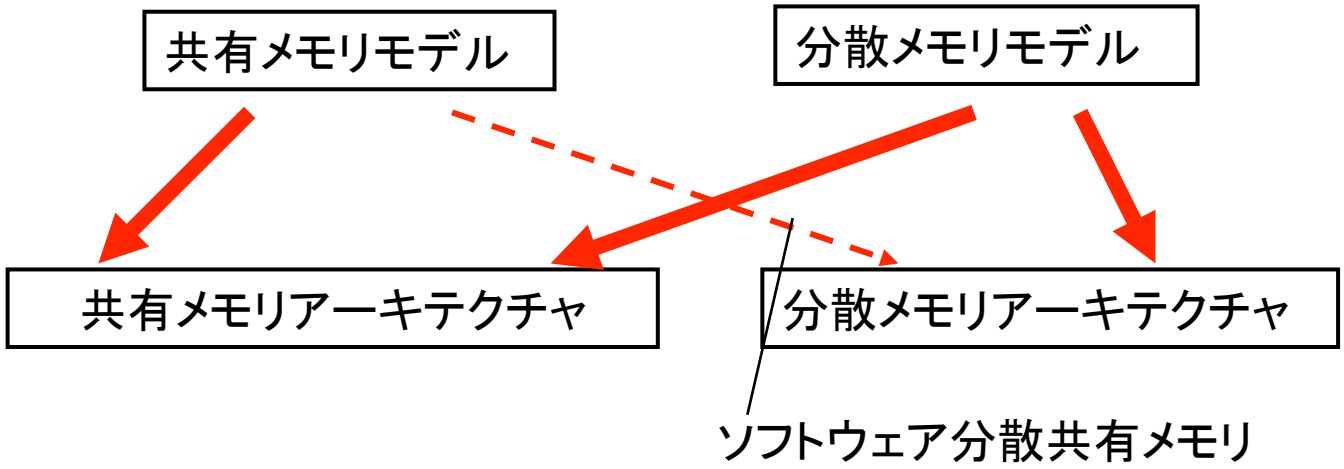
必ずしもモデルとアーキテクチャが一対一の対応ではない

TSUBAMEのアーキテクチャ

- ノード内: 共有メモリ
- ノード間: 分散メモリ
- 近年のスパコンのほとんどは共有と分散の組み合わせ
  - BlueGene, RoadRunner, T2K, 地球シミュレータ・・・



プログラミングモデルとアーキテクチャ

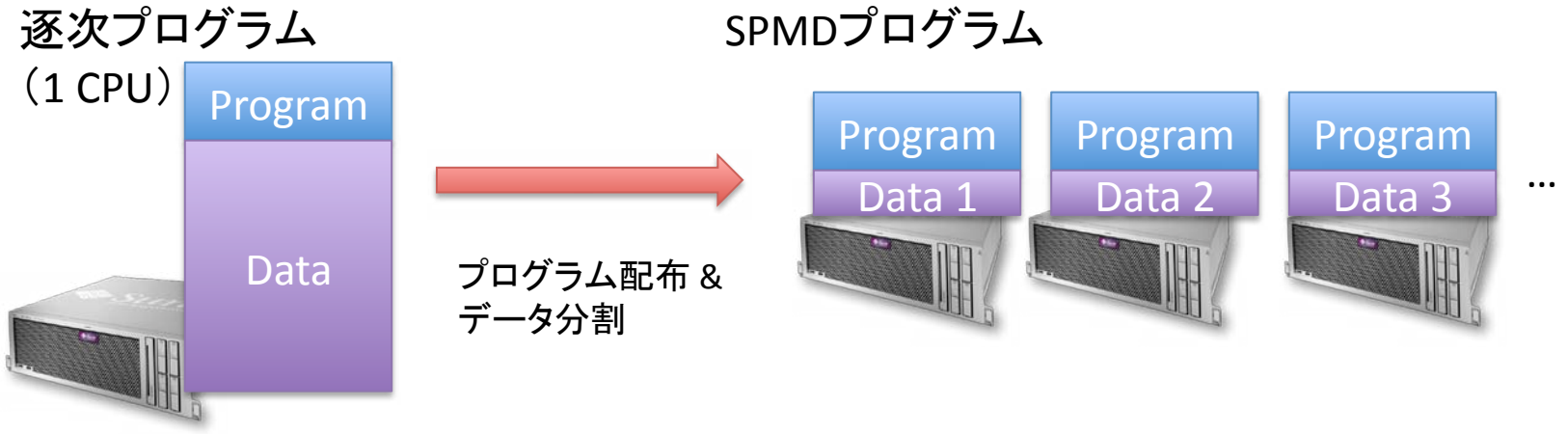


- ノード内の場合
  - 共有メモリモデル (OpenMPなど)
  - 分散メモリモデル (MPIなど)
- ノードをまたぐ場合
  - 分散メモリモデル
  - 分散＋共有 (MPI + OpenMPなど)

MPI
(Message Passing Interface)

MPIとは

- 分散メモリ並列プログラミングの規格
- C, C++, Fortranに対応
- メッセージパッシングのためのライブラリ
- SPMD (Single Program Multiple Data)モデル
- プロセス間の相互作用はメッセージで



MPIプロセスとメモリ (1/2)

- 複数のプロセスが同一プログラムを実行(SPMDモデル)
  - プロセス: プログラム実行の単位
  - MPIでは、1つのプロセスが1つのCPU上で実行されるように、プロセス数を調整するのが一般
  - ➡ TSUBAMEなら、1ノード16プロセス以下 (並列度16)
- MPIプロセスには，0, 1, 2・・・という番号(rank)がつく
  - MPI_Comm_rank(MPI_COMM_WORLD, &rank); ランク取得
  - MPI_Comm_size(MPI_COMM_WORLD, &size); 全プロセス数取得
  - 0 ≦ rank < size
  - MPI_COMM_WORLDは，「全プロセスを含むプロセス集団(=コミュニケータ) 」

MPIプロセスとメモリ (2/2)

- プロセス毎に異なる処理をしたい場合は、rankをif節等で分岐

```
void function() {
  … //すべてのMPIプロセスで実行
  if (rank == 0) {
    //rankが0のプロセス(唯一)でのみ実行
  } else {
    //rankが0以外のプロセスで実行
  }
}
```

- プロセスごとに別のメモリ空間
  → 全ての変数(大域変数・局所変数)は各プロセスで別々
  - 明示的に通信を行わない限り、特定のプロセスで変更したデータ(変数等)は他のプロセスには伝わらない
- メッセージの送信先，受信元にrankを指定

MPIプログラムの概要

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

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);          ← MPIライブラリの初期化

     (計算・通信)

    MPI_Finalize();                  ← MPIライブラリの終了
}
```

メッセージの送信・受信

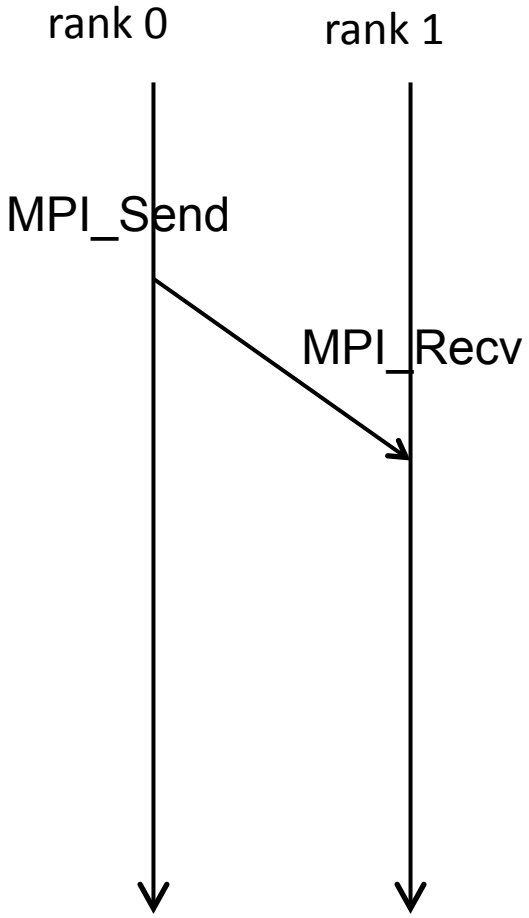
rank 0からrank1へ，int a[16]の中身を送りたい場合

- 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, ・・・
- 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);

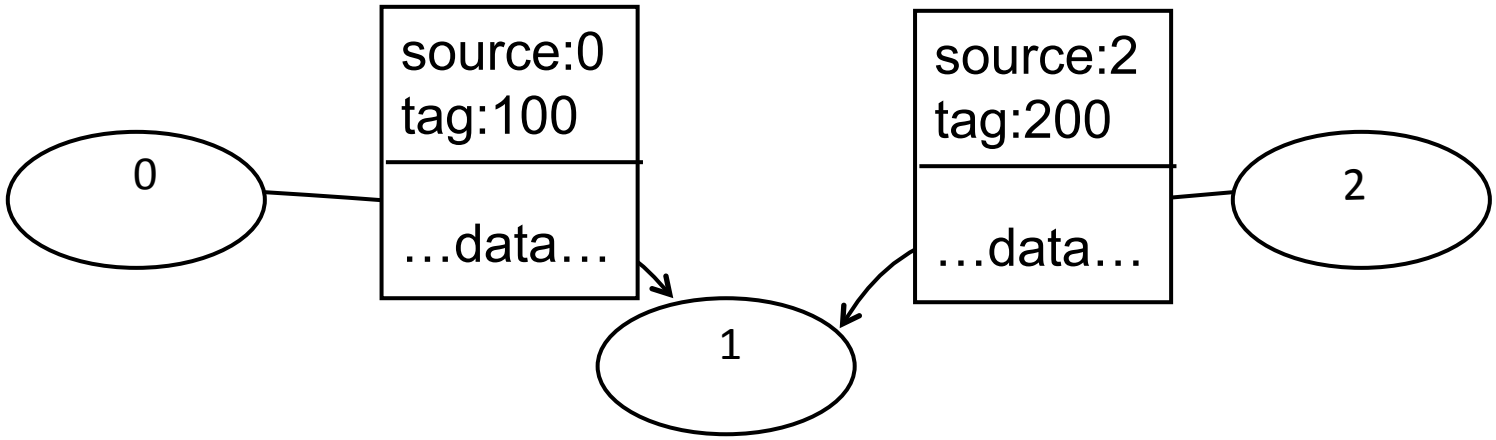
- b: メッセージを受け取るメモリ領域の先頭アドレス
  - 十分な領域を確保しておくこと
- 16: 受け取るデータ個数
- MPI_INT: 受け取るデータ型
- 0: 受け取りたいメッセージの送信元プロセスのrank
- 100: 受け取りたいメッセージのタグ
  - MPI_Sendで指定したものと同じなら受け取れる
- MPI_COMM_WORLD: コミュニケータ
- &stat: メッセージに関する補足情報が受け取れる

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)



MPIサンプルプログラム

- rankが0のプロセスが、rankが1のプロセスに文字列を送る
- MPIプログラム実行時のプロセス数が1以下の場合、エラー終了

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

int main(int argc, char *argv[])
{
  int myrank, nprocs;
  char msg0[100];
  MPI_Status stat;

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  if (nprocs < 2) {
    fprintf(stderr, "#processes must be larger than 2.¥n");
    MPI_Abort(MPI_COMM_WORLD, 1);
  }
```

- プロセス数は実行時に指定

強制終了のための命令

MPIサンプルプログラム

```
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if (myrank == 0) {
    sprintf(msg0, "hello, this is rank[%d]", myrank);
    MPI_Send(msg0, 100, MPI_CHAR, 1, 1000, MPI_COMM_WORLD);
  } else if (myrank == 1) {
    // printf("rank[%d]: I got a message ¥"%s¥"¥n", myrank, msg0);
    MPI_Recv(msg0, 100, MPI_CHAR, 0, 1000, MPI_COMM_WORLD, &stat);
    printf("rank[%d]: I got a message ¥"%s¥"¥n", myrank, msg0);
  } else {
    // do nothing
  }

  MPI_Finalize();
  return 0;
}
```

Rank0でのみ
行われる処理

Rank1でのみ
行われる処理

この時点では変数msg0は初期化されていないため、おかしな結果が出力される

一般的なコンパイル・実行方法

- コンパイル (on Terminal)

  $ mpicc -Wall -o hello hello.c

  – -wall: 警告メッセージを表示

  – -o hello: helloという名前の実行ファイルを生成

- 実行方法 (on Terminal)

  $ mpirun -np 2 hello

  – -np 2: 2プロセス使用

- 実行結果

  rank[1]: I got a message "hello, this is rank[0]”

- ただし、環境によって使い方は変わる

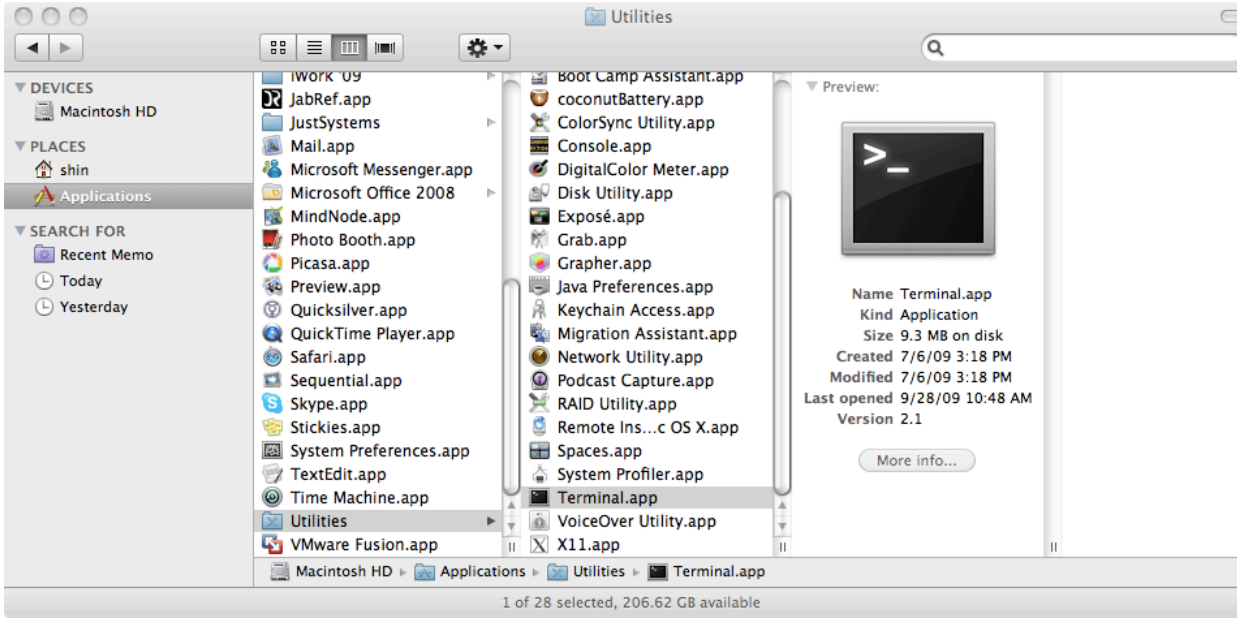
演習室MACでの
MPIプログラミング

演習室MAC環境

- CPU: Quad Core CPU x2 (合計8 CPUコア)
  - 最大MPIプロセス数 => 8
- Memory: 4GB
  - 本来ならば、ノードあたりのメモリ量も考えて、オーバーしないように問題の分割を考える必要がある
- MPI: OpenMPIがインストール済み
  - 2ページ前の一般的なコンパイル・実行方法で可
  - MAC上である程度動作確認してから、TSUBAMEで実行すると効率的

プログラミングの進め方

- エディタ: Emacs
- Terminal上で作業




```
$ mkdir enshu_r1
$ cd enshu_r1
$ ls
hello.c
$ mpicc -Wall -o hello hello.c
$ ls
hello  hello.c
$ mpirun -np 2 hello
rank[1]: I got a message "hello, this is rank[0]”
$
```

Terminal（Shell）のTips

- 長いファイル名を入力したくない
  - 途中まで入力して、「Tab」キーを打てば、補完してくれる
- 直前に実行したコマンドを再度実行したい
  - カーソルキー↑ or Cntl + P
    - 1つ前のコマンドをTerminalに挿入
  - カーソルキー↓ or Cntl + N
    - 1つ後のコマンドをTerminalに挿入
- 以前実行したコマンドを探し出して実行したい
  1. 「Ctrl + r」で検索モードに入る
  2. コマンド文字列の一部を入力すると、過去の実行履歴の中から該当するモノがTerminalに挿入される
  3. 「Ctrl + r」を再度押すことで、異なる候補を挿入