総合演習第1ラウンド資料
No. 2

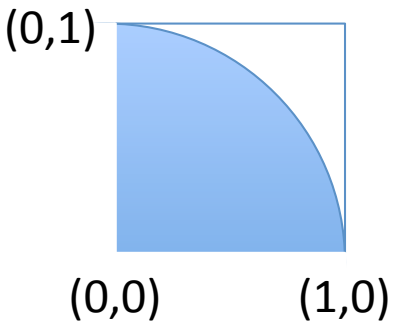
滝澤 真一朗

本日の内容

- 逐次プログラムからMPI並列プログラムの書き換え
  - モンテカルロ法を用いたπ計算プログラム
- TSUBAMEの使い方
  - ログイン方法
  - MACからのファイル転送方法
  - ジョブ（プログラム）実行方法
- 課題

モンテカルロ法によるπ計算

- モンテカルロ法
  - 乱数を用いた数値計算、シミュレーション手法
- π計算への応用
  - 面積1（1x1）の正方形内に適当に点を打ち、半径1の扇形図形に入った点の数を数える



正方形の面積	1
扇形の面積	π/4
正方形内の全点数	N
扇形内の点数	x

Nはプログラム実行時に、xはすべての点を打ち終わった後に決まる

$$1 : \pi/4 = N : x$$

$$\pi = \frac{4x}{N}$$

逐次プログラム

```
#include <stdio.h>
#include <stdlib.h> /* for random */
#include <math.h> /* for fabs and M_PI */
#define N_POINTS 100000000
```

打ち込む点の数

```
int main(int argc,char *argv[])
{
  int intotal, i;
  double x, y, dist, pi;

  srandom(0);
  intotal = 0;
```

扇形内に入った点の数

```
  for (i = 0; i < N_POINTS; i++) {
    x = (double)random() / RAND_MAX;
    y = (double)random() / RAND_MAX;
    dist = sqrt(x*x + y*y);
    if (dist < 1) intotal +=1;
  }
  pi = 4 * (double)intotal / N_POINTS;

  printf("pi is approximately %.16f, Error is %.16f\n",
         pi, fabs(pi - M_PI));
  return 0;
}
```

点を打ち、半径1の扇形に入るか判断
入った場合、intotalをインクリメント

並列化の進め方

1. 逐次プログラムの処理の重たい部分を探し出す
   - カーネル という(プログラムのメイン処理部分)
2. カーネル処理の分割
3. データ送受信関数の挿入

- その他、MPIを使うためのヘッダファイル、関数を挿入する必要がある

さぁ、みんなで考えよう！

カーネル処理の分割・並列化

- MPIプロセスで処理対象の点を分ける
  - 各プロセスの担当数 = N_POINTS / N_PROCS

```
MPI_Comm_size(MPI_COMM_WORLD,&n_procs);
n = N_POINTS / n_procs;

srandom(myid);
incnt = 0;                扇形内の点数(各プロセス担当分)
for (i = 0; i < n; i++) {
  x = (double)random() / RAND_MAX;
  y = (double)random() / RAND_MAX;
  dist = sqrt(x*x + y*y);
  if (dist < 1) incnt +=1;
}
```

データ送受信関数の挿入

- すべてのプロセスのincntを1つのプロセスに集め、和を求める

```
if (myid == 0) {
  intotal = incnt;
  for (i = 1; i < n_procs; i++) {
    MPI_Status stat;
    MPI_Recv(&incnt, 1, MPI_INT, i, 100, MPI_COMM_WORLD, &stat);
    intotal += incnt;
  }
  pi = 4 * (double)intotal / N_POINTS;

} else {
  MPI_Send(&incnt, 1, MPI_INT, 0, 100, MPI_COMM_WORLD);
}
```

この時点では、incntにはrank0のデータが入っている

incntにはrank “i”のデータが入っている

ちょっとめんどくさい・・・

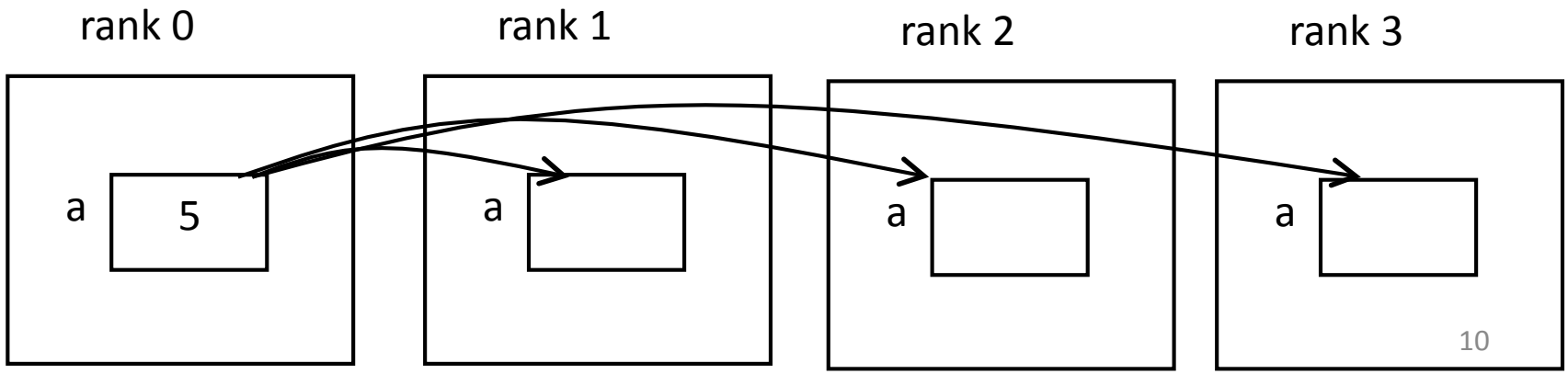
集団通信とは

- 一対一通信: MPI_Send対MPI_Recv
- 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)
  - 例：MPI_Barrier(MPI_COMM_WORLD);
    全プロセスがMPI_Barrier関数を呼び出すまで待つ
    (バリア同期)

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を呼び出すこと
  - 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

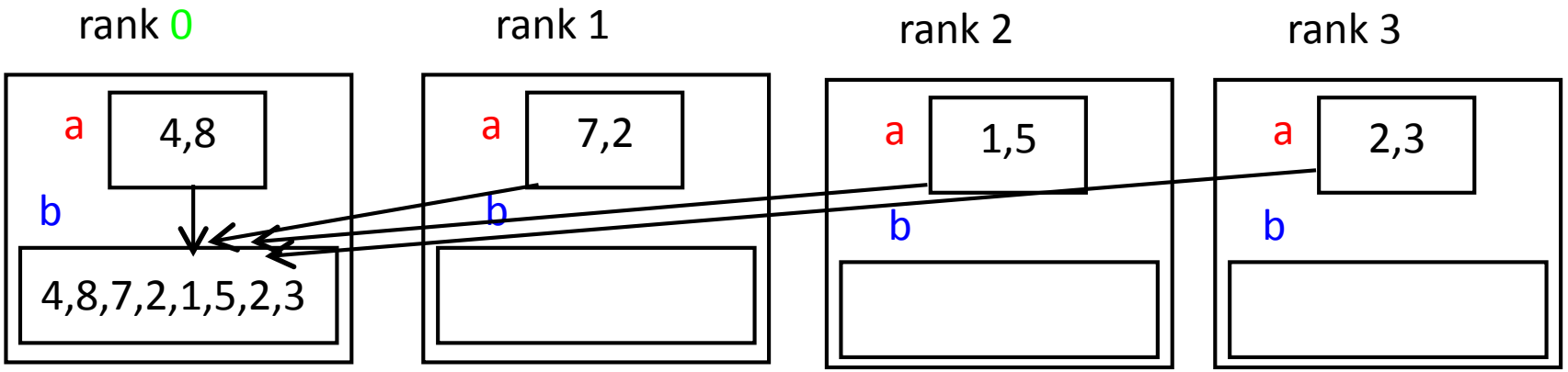


集団通信：MPI_Gather, MPI_Allgather

- 例：各プロセスが一部ずつ持っている配列を合体させたい

  MPI_Gather(&a, 2, MPI_INT,

  &b, 8, MPI_INT, 0, MPI_COMM_WORLD)



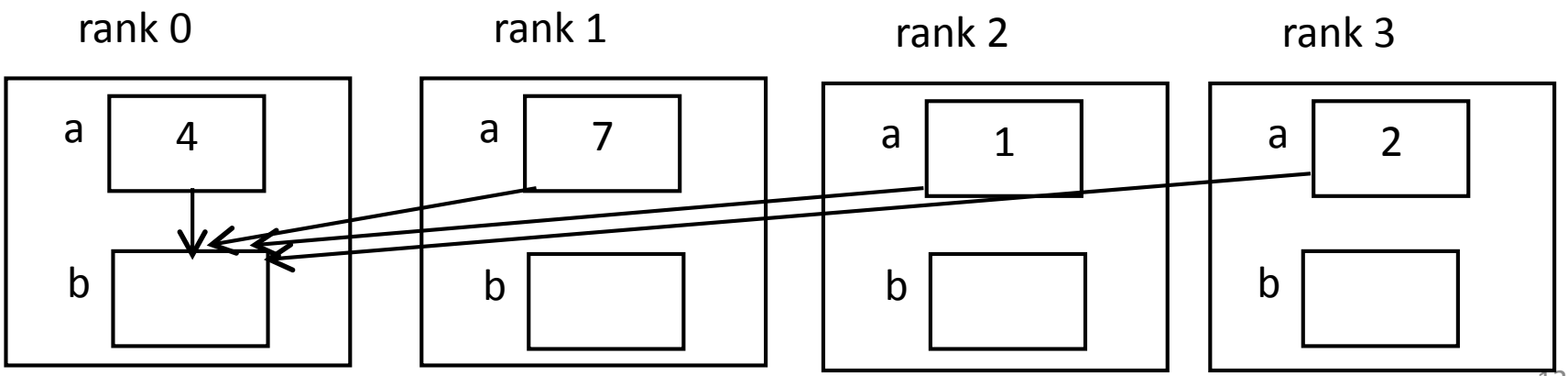
- MPI_Gatherは1プロセス(上ではrank0)にだけ集める
- MPI_Allgatherは全プロセスに合体配列を作る

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()を呼び出すこと
  - 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



MPI_Reduceでの実装

```
MPI_Reduce(&incnt, &intotal, 1, MPI_INT,
           MPI_SUM, 0, MPI_COMM_WORLD);

if (myid == 0) {
  pi = 4 * (double)intotal / N_POINTS;
  結果の表示
}

MPI_Finalize();
return 0;
```

- すべてのMPIプロセスが持つincnt変数の値を、rank0プロセスに転送し、intotal変数にその和を保存
  - rank0以外でのintotal変数の値は不定

TSUBAMEの使い方

TSUBAME情報源

- http://www.gsic.titech.ac.jp　または検索「GSIC」
  ⇒「研究用計算機システム」
  - マニュアル
  - 統計情報(混雑具合など)
  - メンテナンス予定
- 相談ML:
  - soudan@cc.titech.ac.jp
  - ただし、総合演習に関する
    ことは滝澤に聞いてください。

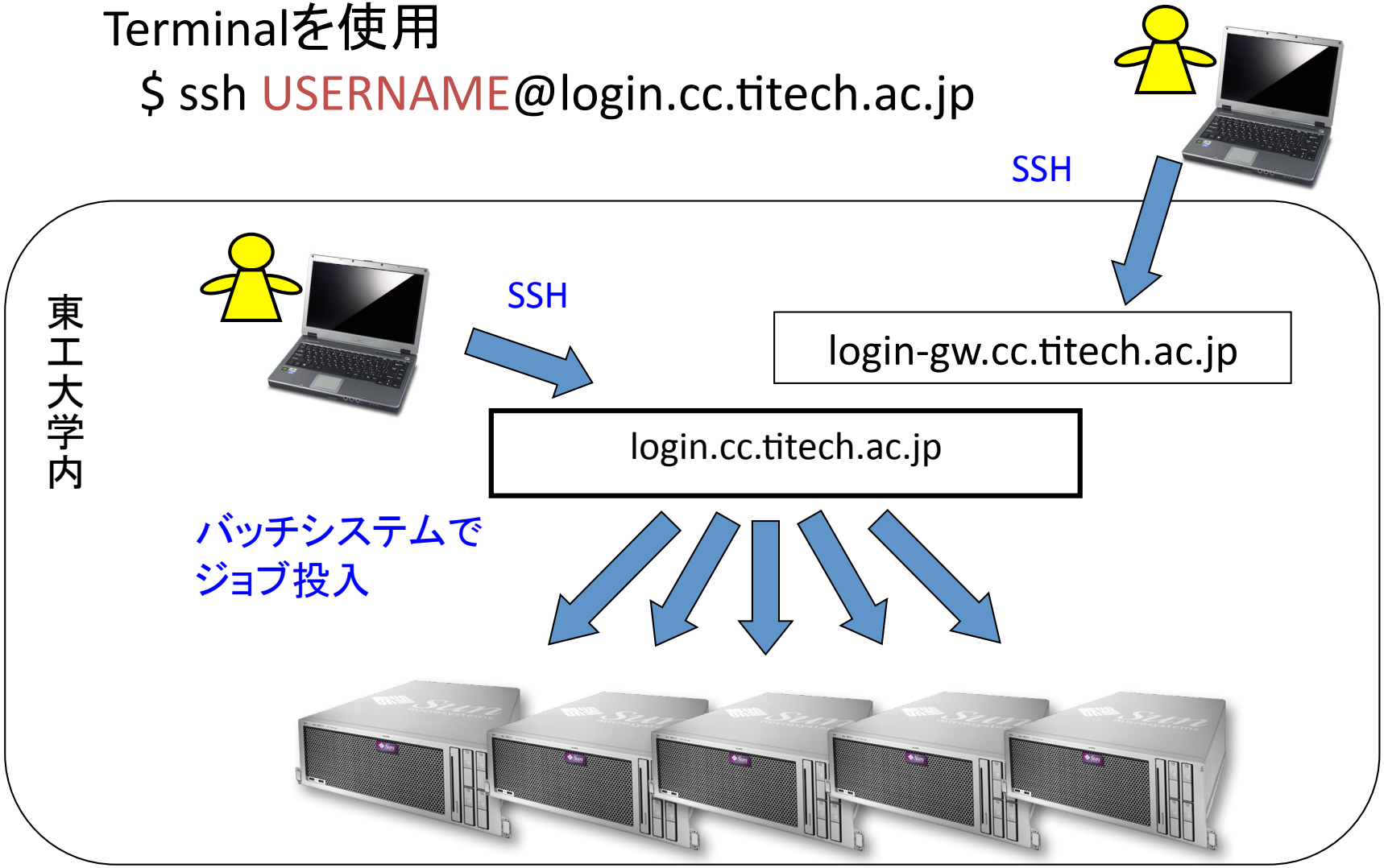
TSUBAME基礎情報

- 16プロセッサ(CPUコア)×655ノード=10480プロセッサ
  - ただし、同時にすべて使えるわけではない
- ノードあたりメモリ32GB
- OSはLinux
  - SUSE Linux Enterprise Server 64bit, kernel 2.6
- プログラム実行法
  - インタラクティブノードにログイン
  - バッチキューシステムを介してプログラム実行
    - Sun N1 Grid Engine
- 利用料金あり (一部除く)
  - 課金グループ制

TSUBAMEへのログイン

Terminalを使用

$ ssh USERNAME@login.cc.titech.ac.jp



TSUBAMEへのファイル転送

- SCPを使用
  - SSHを使った、ファイルコピー
- Terminalを使用
  - TSUBAMEではなく、MACにログインしていることを確認
  - TSUBAMEへのファイルコピー （TSUBAMEに転送）

    $ scp mpi_pi.c USERNAME@login.cc.titech.ac.jp:mpi_pi.c
  - TSUBAMEからのファイルコピー （TSUBAMEから取得）

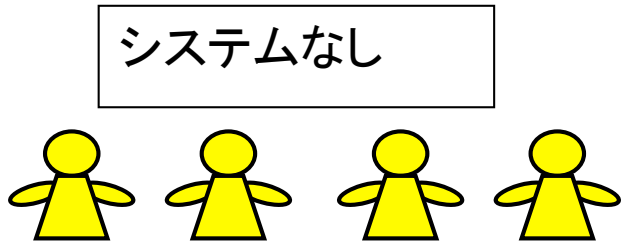
    $ scp USERNAME@login.cc.titech.ac.jp:mpi_pi.c mpi_pi.c

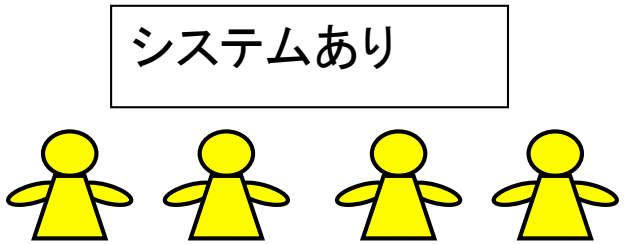
インタラクティブノードでできること

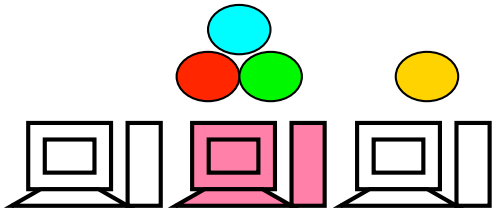
- 基本的なLinuxコマンド
- エディタ(vi, emacs)でプログラム開発
- デバッグ，短時間(10分以下)のプログラム実行
- 長時間かかるプログラム実行はバッチキューで！

バッチキューシステムとは

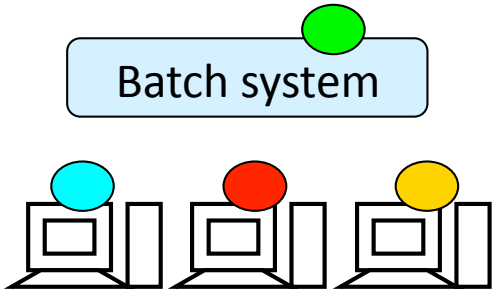
- 多数のプログラムの「交通整理」
  - OSはノード内、バッチキューシステムはノード間







ユーザが自分でノード決定
混雑すると実行が遅くなる



システムが自動決定
ジョブ開始が待たされることあり

バッチキュー構成

- ベストエフォートキュー
  - bes1(120), bes2(120)
  - CPUコア毎にジョブ割り当て
- 性能保証キュー
  - sla1(120), sla2(120), sla3(60)
  - ノード毎にジョブ割り当て
  - 並列プログラム向き
- 初心者キュー
  - novice

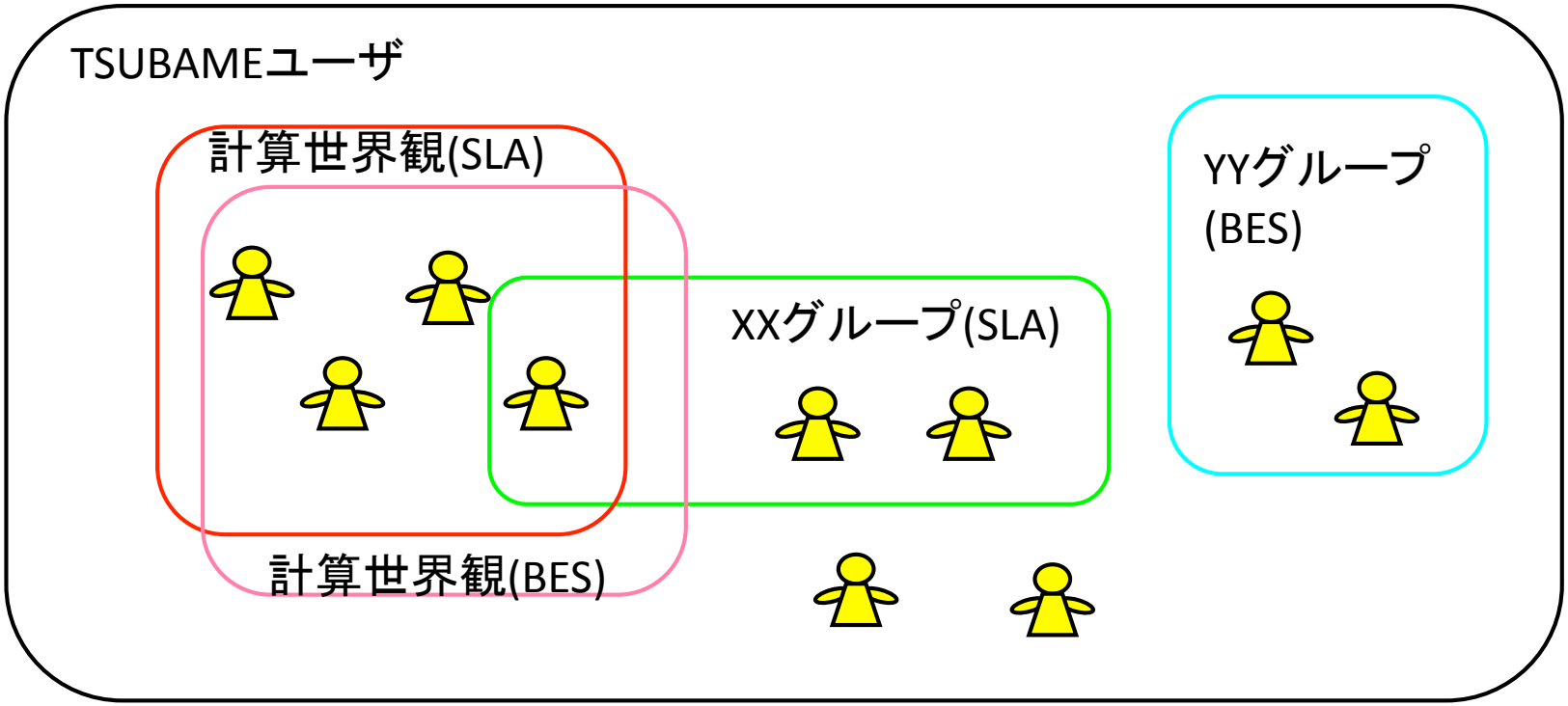
BES課金グループから利用

SLA課金グループから利用

課金なし
短いジョブのみ

※ 括弧内はノード数

課金グループについて



- 自分の所属グループはqgroup –aコマンドで確認できます

ジョブの投入法 (1)

./myprog a b c
というジョブを実行したい場合
例 :

% n1ge –g 1B080157 ./myprog a b c

% n1ge –g [group] –q [queue] ./myprog a b c

課金グループID　キュー名

- 実行されると、標準出力の内容ファイルが作成される
- -q は省略可能
- noviceキューを使用する際には、-gオプションも必要ない

ジョブの投入法(2)

mpirun –np 32 ./parprog a b c

というジョブを実行したい場合

% n1ge –mpi 32 –g … –q … ./parprog a b c

- バッチキューシステムは、32CPUが同時に空いたときにジョブ実行を開始させる

実行後、XXX.oJOBID、XXX.poJOBIDファイルが生成される

- 例: OTHERS.o6082523, OTHERS.po6082523
- XXX.oJOBID: プログラムの出力
- XXX.poJOBID: スケジューラの出力
- lessコマンドで内容を確認

ジョブの管理方法

- ジョブ一覧　(自分が所属する課金グループ内)

  % qstat

  % qstat –u <user-name>

- ジョブ削除

  % qdel <job-id>　　・・・bes/noviceの場合

  % qdel_sla <job-id>　　・・・*sla*の場合

- 課金グループ情報取得

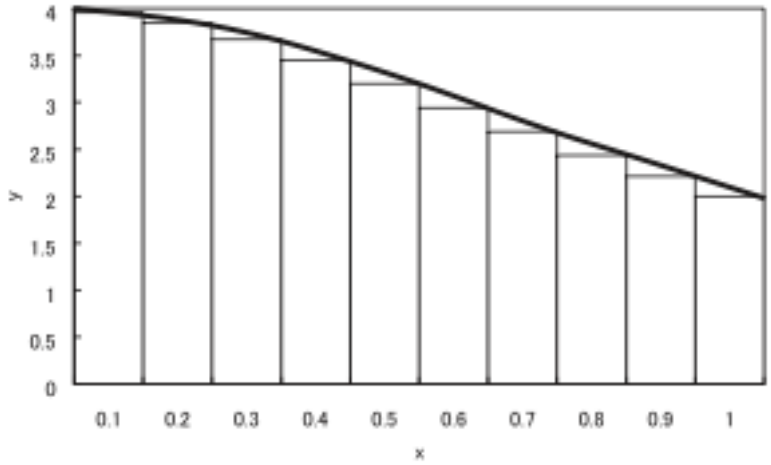
  % qgroup –a

  % qgroup –g <group-id>

課題

- 台形公式を用いたπ計算

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$



$$y = \frac{4}{1+x^2}$$

- 配布した逐次プログラムの並列化
  - mpi_pi_trapezoid.cを編集
- 実行速度を計測
  - 10億分割以上でテスト
  - プロセス数: 1, 2, 4, 8, 16

MPIでの時間記録方法

- MPI_Wtime関数を使用
  - 詳しくは mpi_pi_mc.cを参照

```
double start_time, end_time;

start_time = MPI_Wtime();
なんか計算
end_time = MPI_Wtime();

printf(“Time: %f¥n”, end_time – start_time);
```