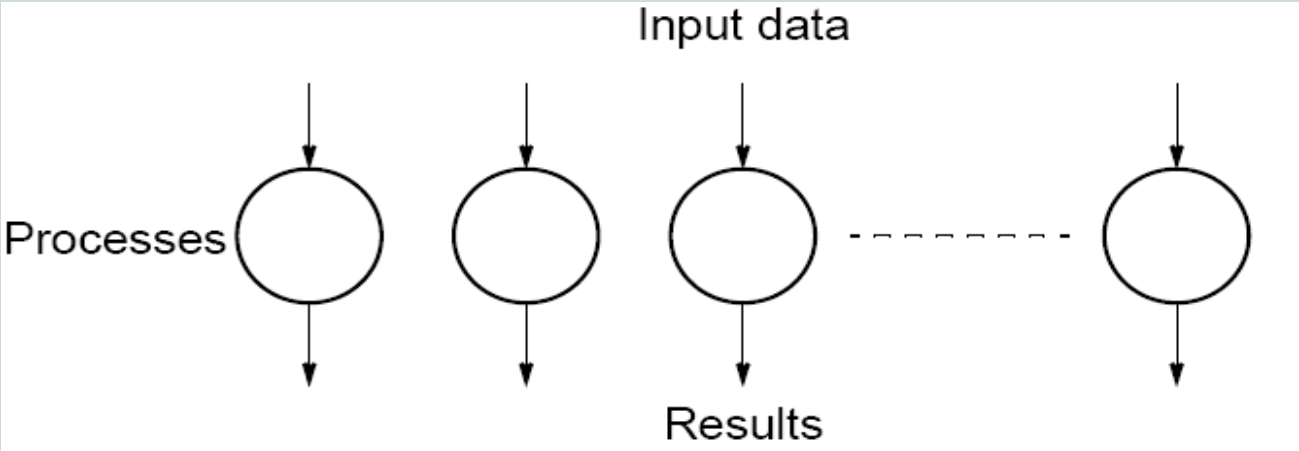
驚異的並列計算

独立並列処理

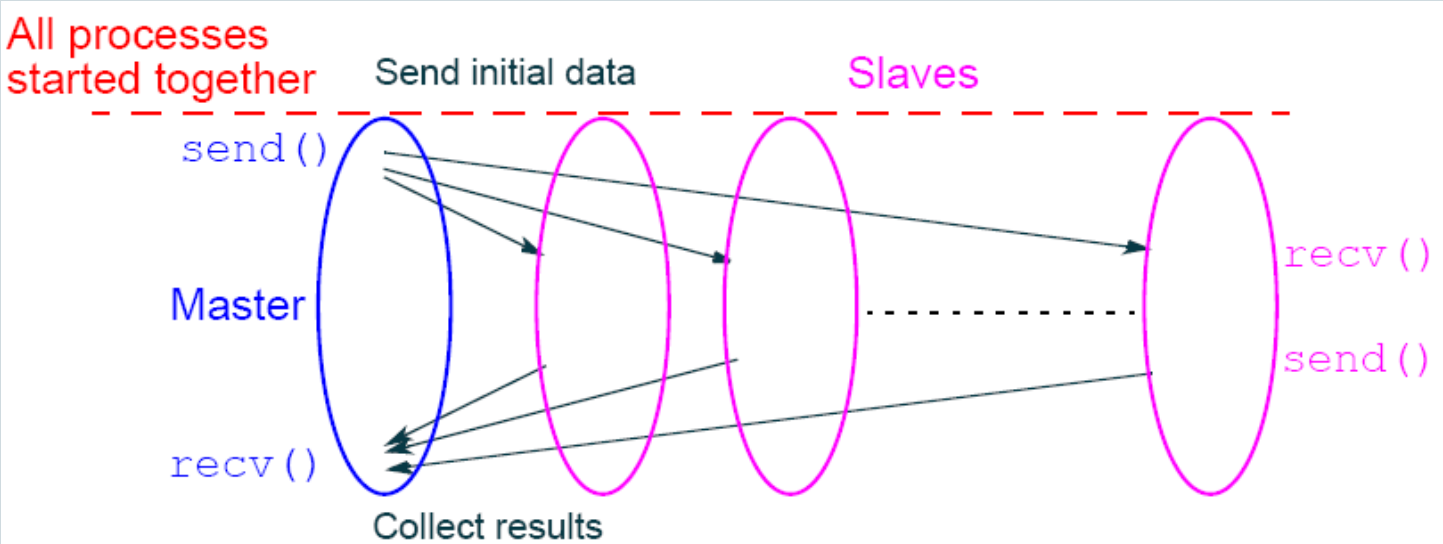
3.1 理想的並列処理

- 驚異的並列処理(embarassingly parallel) -> 図
  - 各プロセッサがほぼ完全に独立に処理できる
  - 通常は困難であるが，幾つかの事例が該当する



3.1 理想的並列処理

- 近驚異的並列処理
  - データの分配，結果の収集等が必要なもの
  - マスタースレーブ構成が一般的
  - 動的プロセス生成，静的プロセス生成どちらでも可能



Usual MPI approach

3.2 驚異的並列の例

画像の幾何学的変換

- 画像データ
  - カメラ、スキャナ等から得られた画像
  - コンピュータグラフィックス
- 2次元画像の格納(bitmap)
  - 2次元配列に格納可能
  - モノクロであれば各ピクセル1ビットで表現できる
  - カラーであれば、RGB各256階調で各ピクセル24bit -> tiff形式
  - 使用している色は限定されているため実際は24x(ピクセル数)は必要ない
    -> gif

3.2 驚異的並列の例

画像の幾何学的変換

シフト

(a) 2次元オブジェクトをx軸方向にΔx, y軸方向にΔyシフトしたとき

$$x' = x + \Delta x$$

$$y' = y + \Delta y$$

スケーリング

(b) オブジェクトをx方向にSx倍, y方向にSy倍したときの座標

$$x' = x\ Sx$$

$$y' = y\ Sy$$

Sx, Syが1より大きいときは拡大, 0と1の間の時は縮小

3.2 驚異的並列の例

画像の幾何学的変換

回転

(c) 座標系の原点に対して角度 θ 回転したオブジェクトの座標は

$$x' = x \cos\theta + y \sin\theta$$

$$y' = -x \sin\theta + y \cos\theta$$

クリッピング

(d) 定義された長方形の境界を図に適用して，定義された領域外の点を表示図形から削除する

$$xl \leqq x' \leqq xh$$

$$yl \leqq y' \leqq yh$$

3.2 驚異的並列の例(画像の幾何学的変換)(1)

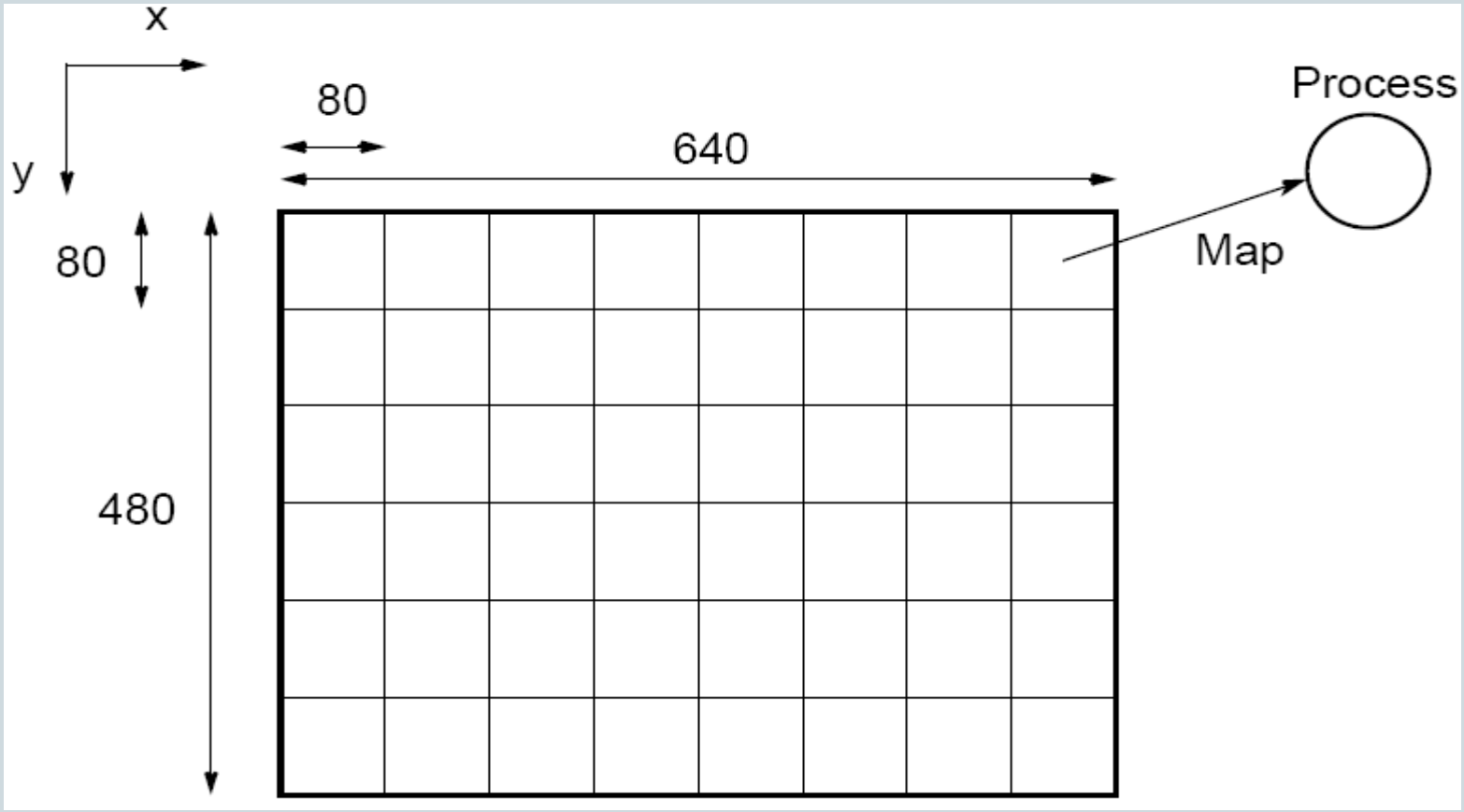
画像の幾何学的変換

- シフト, スケーリング, 回転, クリッピング
  - 各画素に対する処理
  - 各画素は独立に計算可能 -> 驚異的並列処理が可能
- 並列プログラム
  - 各プロセッサに画素を割当てる -> 通常は, 画素数 ≫ プロセッサ数
  - 長方形／正方形の領域毎
  - 行／列毎

例: 640x480の画像を処理したい

- マスタプロセスと48個のスレーブプロセスを用いて10行ずつ割当てる
- マスタプロセス:
  - 各プロセスに最初の行を送信する;
- スレーブプロセス:
  - 640 x 10の領域を処理する;

3.2 驚異的並列の例(画像の幾何学的変換)(2)



3.2 驚異的並列の例(画像の幾何学的変換)(プログラム)

```
Master:
    for(i=0, row=0; i<48; i++, row=row+10)
       send(row, Pi);
    for(i=0; i<480; i++)
       for(j=0; j<640; j++)
          temp_map[i][j]=0;
    for(i=0; i<(640*480); i++){
       recv(oldrow,oldcol,newrow,newcol, PANY);
                              /*PANY = 任意のプロセッサ*/
       if (!((newrow < 0)||(newrow >=480)||
           (newcol <0)||(newcol>=640))) /* 範囲内？ */
          temp_map[newrow][newcol]=
                              map[oldrow][oldcol];/* データをシフト */
    }
    for(i=0; i<480; i++)
       for(j=0; j<640; j++)
          map[i][j]=temp_map[i][j]
```

3.2 驚異的並列の例(画像の幾何学的変換)(プログラム)

```
Slave:
1.   recv(row, Pmaster)
2.   for(oldrow=row; oldrow<(row+10); oldrow++)
3.      for(oldcol=0; oldcol<640; oldcol++){
4.         newrow = oldrow + delta_x;
5.         newcol = oldcol + delta_y;
6.         send(oldrow, oldcol, newrow, newcol,
7.              Pmaster);
8.      }
```

- マスタは各スレーブに先頭行を送っているが，タスクIDより計算できるので必要ない
- 結果の転送は画素毎だと通信オーバーヘットが大きくなる
- グループ化することにより低減できる

3.2 驚異的並列の例(画像の幾何学的変換)(解析)

逐次処理の計算量

- n x n画素だとすると, ts =n^2

並列処理における通信時間

- t_comm = t_startup + m・t_data
  - t_startup: メッセージの作成+送信の開始(一定)
  - t_data: 1データを送信するのに要する時間(一定)
  - m: データ数
- t_comm = p(t_startup + t_data)+n^2(t_startup+4t_data)
  - p:プロセス数

並列処理における計算時間

- t_comp = 2(n^2/p)
  - 各プロセッサはn^2/p個の画素を計算
  - 各計算で, 2回の加算が必要

全実行時間

- t_p = t_comp + t_comm
- 通常はt_commが大きい. イーサーネットの最小起動時間500 μ s

3.2 驚異的並列の例(マンデルブロー)

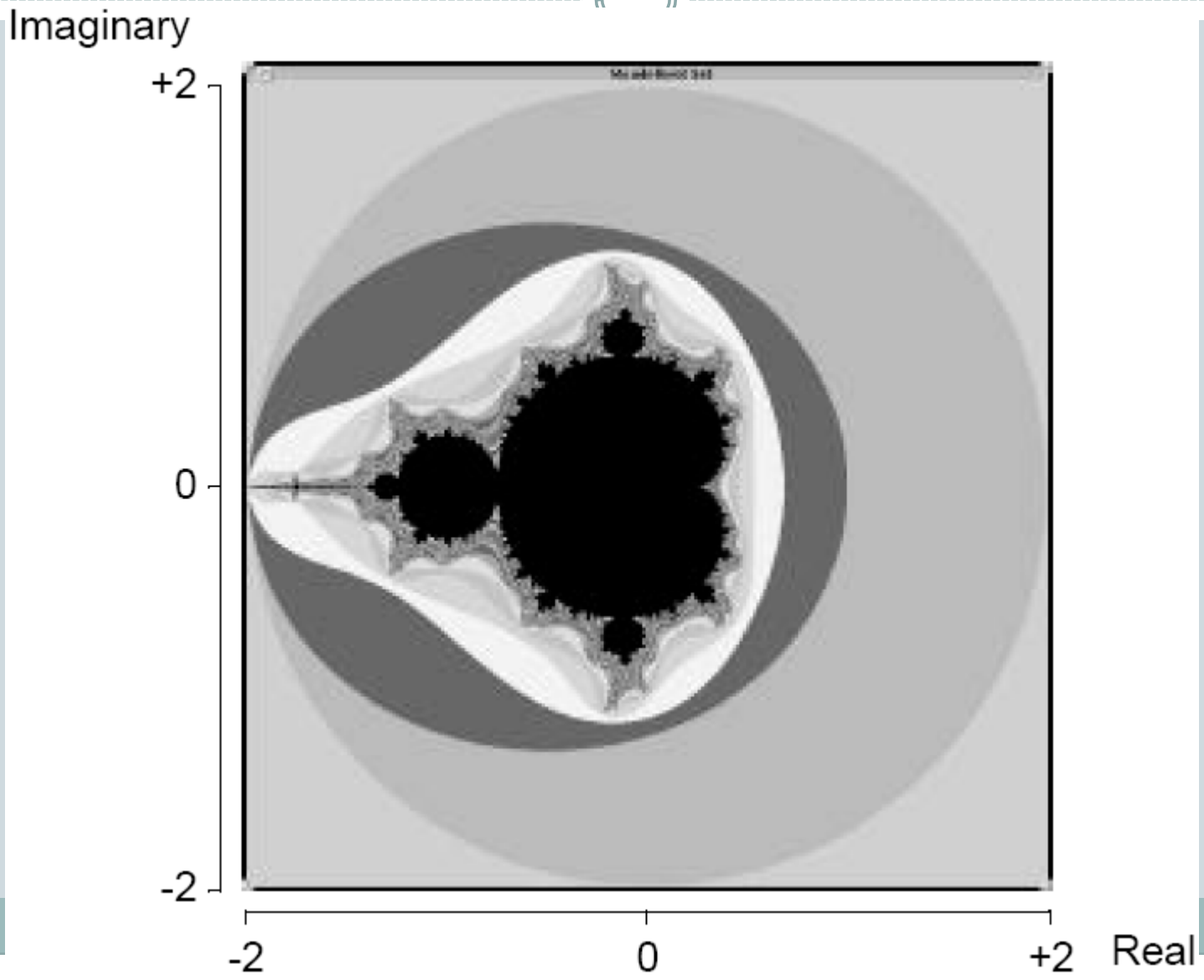
マンデルブロー集合

- ある関数を繰り返し計算したとき，擬安定である複素数平面上の点の集合
- 通常，$z_{k+1} = z_k^2 + c$ が用いられる.
- 繰り返しの式は

  $z\_real = z\_real^2 - z\_image^2 + c\_real$

  $z\_imag = 2z\_real\ z\_imag + c\_imag$

3.2 驚異的並列の例(マンデルブロー) (2)



3.2 驚異的並列の例(マンデルブロー)(プログラム)

```
1.   int cal_pixel(complex c)
2.   {
3.      int count, max;
4.      complex z;
5.      float temp, lengthsq;
6.      max = 256;
7.      z.real = 0;
8.      z.imag = 0;
9.      count = 0;
10.     do{
11.        temp = z.real*z.real - z.imag*z.imag  + c.real;
12.        z.imag = 2*z.real*z.imag + c.imag;
13.        z.real = temp;
14.        lenghsq = z.real*z.real + z.imag*z.imag;
15.        count++;
16.     }while((lenghsq < 4.0) && (count < max));
17.     return count;
18.  }
```

3.2 驚異的並列の例(マンデルブロー)(プログラム)

- ディスプレイのサイズdisp_height, disp_widthの領域内の点(x,y)を, 最小値real_min, imag_min, 最大値real_max, imag_maxに表示する.

```
c.real = real_min + x*(real_max-real_min)/disp_width;
c.imag = imag_min + x*(imag_max-imag_min)/disp_height;
```

- 計算を簡略化するために

```
scale_real = (real_max - real_min)/disp_width;
scale_imag = (imag_max - imag_min)/disp_height;
```

- スケーリングを含むコードは

```
1. for (x=0; x<disp_width; x++)
2.    for(y=0; y<disp_height; y++){
3.       c.real = real_min + ((float)x * scale_real);
4.       c.imag = imag_min + ((float)y * scale_imag);
5.       color = cal_pixel(c);
6.       display(x, y, color)
7.    }
```

- 独立性:各画素が周囲の画素の情報を使用しないで計算可能

3.2 驚異的並列の例(マンデルブロー)(並列化)

静的タスク割当て:長方／正方、行／列のグループ化が可能

Master:

```
1.   for(i=0, row=0; i<48; i++,row=row+10) send(&row, Pi);
2.   for(i=0; i<(480*640); i++){
3.      recv(&c, &color, Pany);
4.      display(c, color);
5.   }
```

Slave:

```
1.   recv(&row, Pmaster);
2.   for(x=0; x<disp_width; x++)
3.      for(y=row; y<(row+10); y++){
4.         c.real=min_real+((float)x*scale_real);
5.         c.imag=min_imag+((float)x*scale_imag);
6.         color = cal_pixel(c);
7.         send(&c, &color, P_master);
8.      }
```

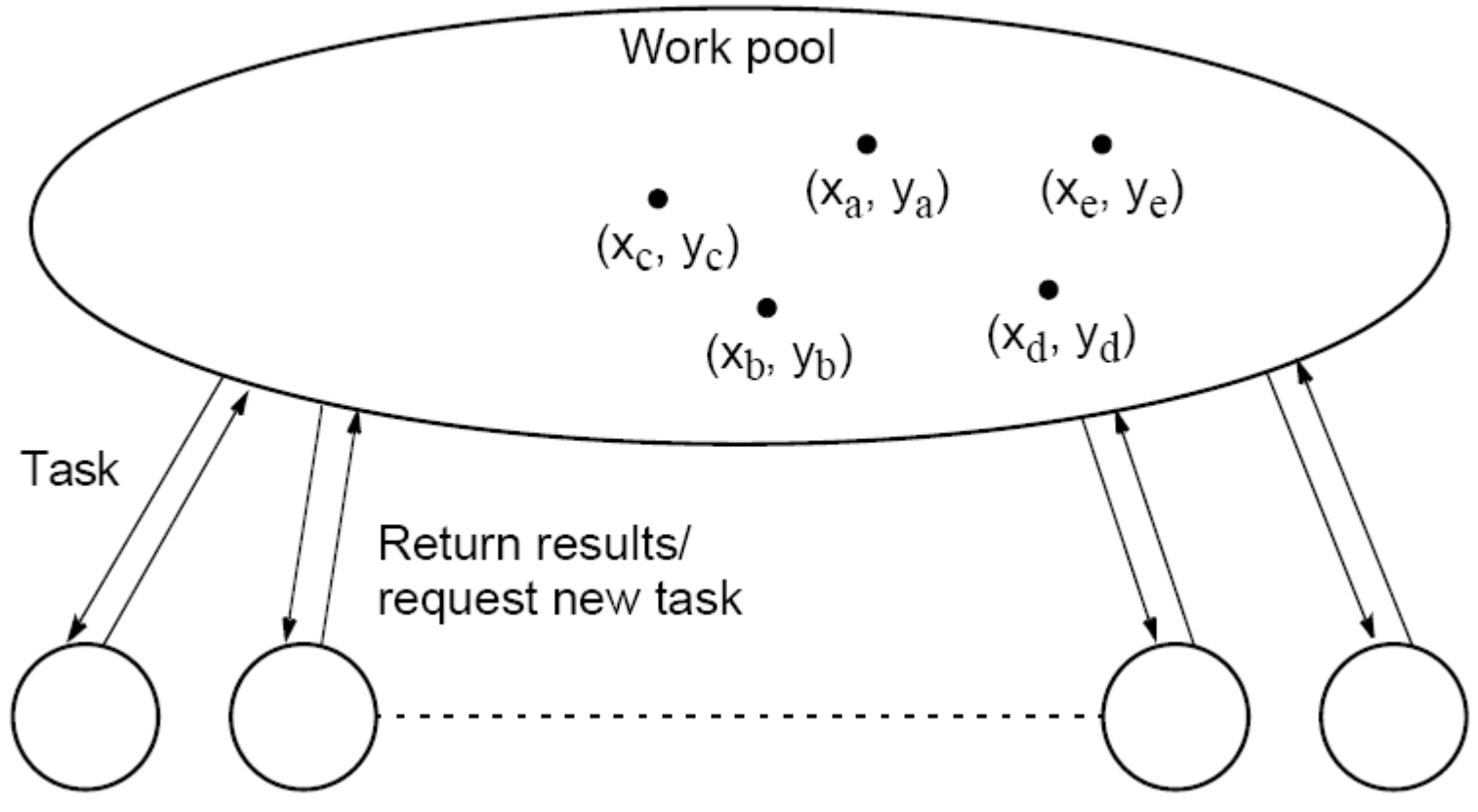
3.2 驚異的並列の例(マンデルブロー)(プログラム)

動的タスク割当て: 各プロセスは一度の1行の計算を行う。終了すると別の行。

Master:

```
1.    count=0; row=0;
2.    for(k=0; k<procno; k++){
3.       send(&row, Pk, data_tag);
4.       count++;  row++;
5.    }
6.    do{
7.       recv(&slave, &r, color, Pany, result_tag);
8.       count--;
9.       if(row<disp_height){
10.         send(&row, P_slave, data_tag);
11.         row++; count++;
12.      }else
13.         send(&row, P_slave, terminator_tag);
14.      rows_recv++;
15.      display(r, color);
16.   }while(count>0);
```

動的タスク割当て


Work pool
(x_a, y_a)
(x_e, y_e)
(x_c, y_c)
(x_b, y_b)
(x_d, y_d)
Task
Return results/
request new task

3.2 驚異的並列の例(マンデルブロー)(プログラム)

```
Slave:
1.   recv(&y, P_master, source_tag);
2.   while(source_tag == data_tag){
3.      c.imag=imag_min + ((float)y*scale_imag);
4.      for(x=0;x<disp_width; x++){
5.         c.real=real_min + ((float)x*scale_real);
6.         color[x]=cal_pixel(c);
7.      }
8.      send(&i, &y, color, P_master, result_tag);
9.      recv(&y, P_master, source_tag);
10.  };
```

3.2 驚異的並列の例(マンデルブロー)(解析)

逐次処理の計算量

- t_s = ≦ max × n
- 各画素の繰り返しの回数はmaxを越えない

この並列プログラムは、通信、計算、通信の構成になっている

- フェーズ１：通信
  - t_comm１ = s(t_startup + t_data)
  - s: スレーブの数
- フェーズ２：計算
  - 画素が均一に全プロセッサに分配されると仮定すると
  - t_comp ≦ (max × n)/s
- フェーズ３：通信
  - 個別の結果がマスタに送り返される
  - t_comm2 = n(t_startup + t_data)/s
- 全体：
  - t_p ≦ (max × n)/s +(n/s + s)(t_startup + t_data)
  - maxが大きければ、速度向上率はs(=p-1)に近づく