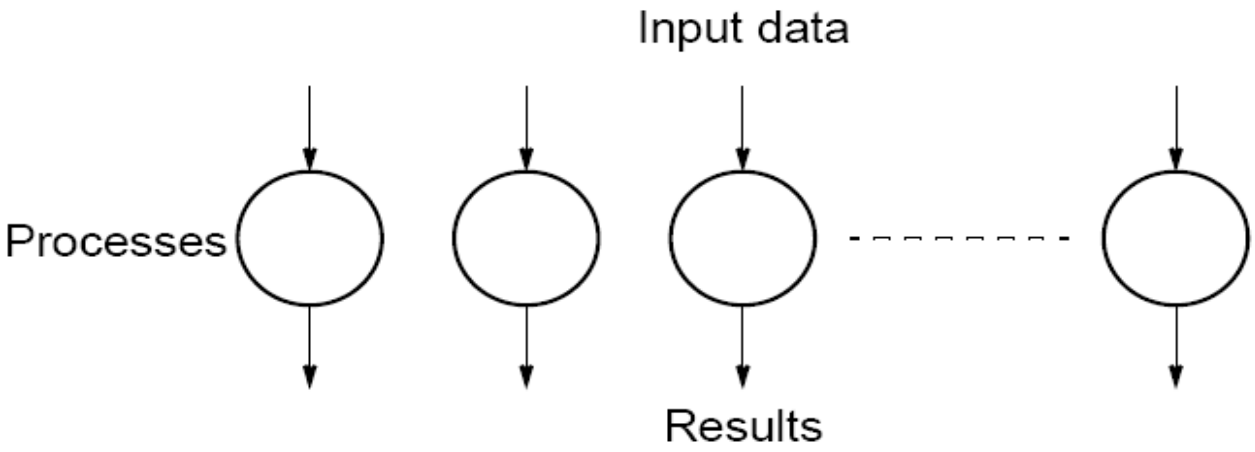
Técnicas de Paralelização

1 – Computações embaraçosamente paralelas (naturalmente)

Computações embaraçosamente paralelas

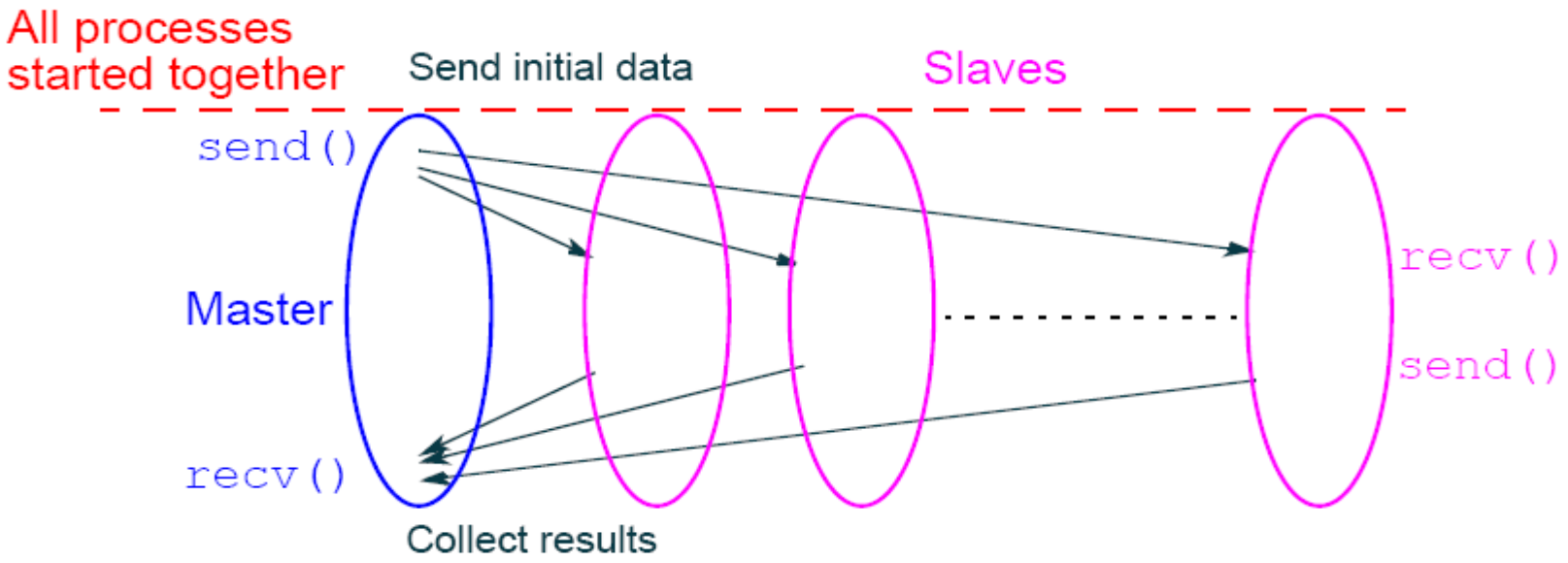
Computação que pode ser dividida num conjunto de tarefas independentes que podem ser executadas em processadores diferentes.



- Não há comunicação entre os processos.

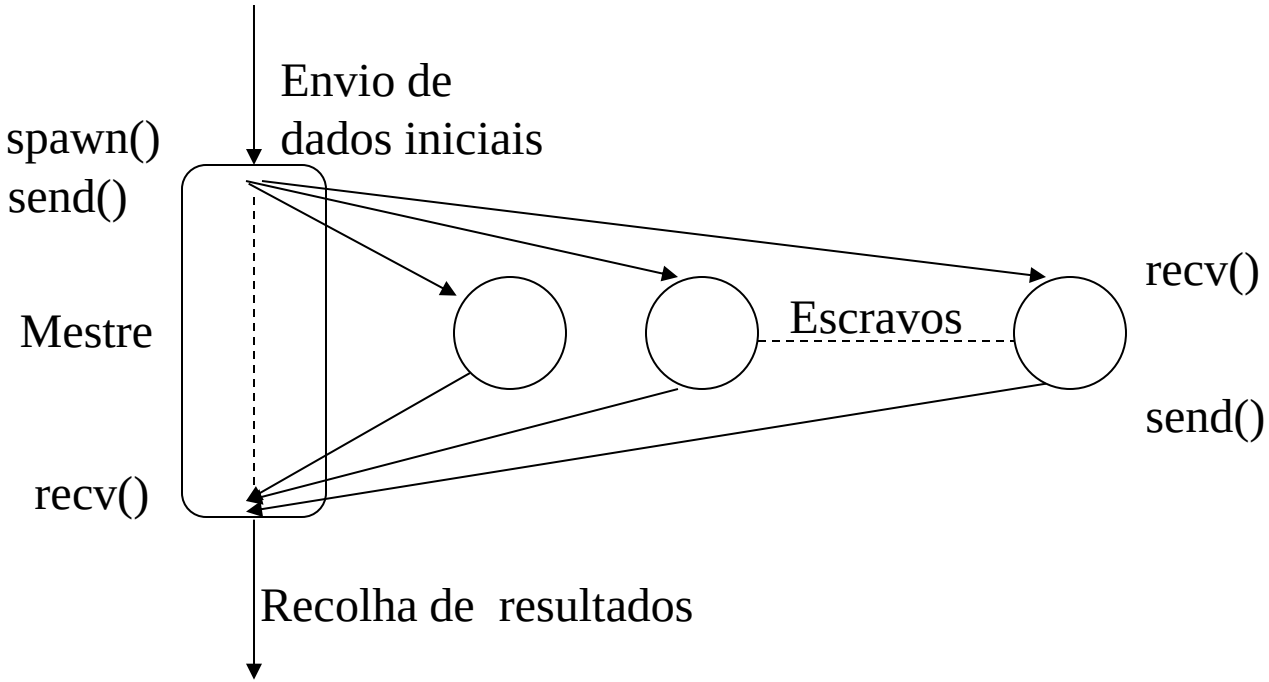
- Cada processo pode executar as suas tarefas sem interação com os outros.

Computações embaraçosamente paralelas, com criação estática de processos, abordagem mestre-escravo



Usual MPI approach

Computações embaraçosamente paralelas, com criação dinâmica de processos, abordagem mestre-escravo



Exemplos de Computações embaraçosamente paralelas

- Processamento de imagens
- Conjunto de Mandelbrot
- Métodos de Monte Carlo

Processamento de imagem

Muitas operações só envolvem dados locais com comunicações entre processos muito limitadas.

Some geometrical operations

Shifting - deslocamento

Deslocar um objeto de Δx na dimensão x e Δy na dimensão y

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

com x e y *as coordenadas originais e* x' e y' as novas coordenadas..

Scaling – mudança de escala

Mudar a escala de um objeto por um fator S_x na dimensão x e *por* S_y na dimensão y

$$x' = xS_x$$
$$y' = yS_y$$

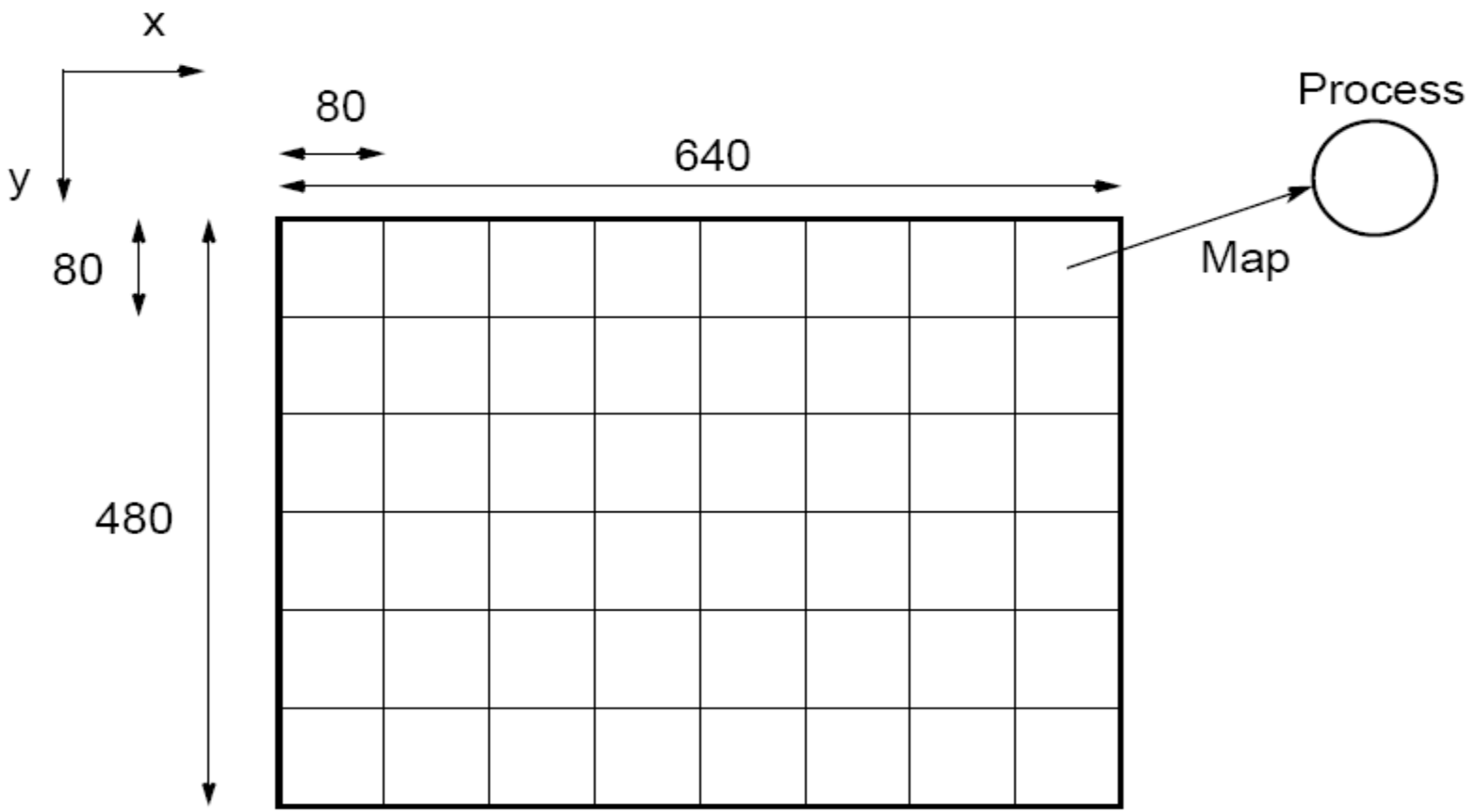
Rotation - rotação

Rodar um objeto de um ângulo θ em relação à origem do sistema de coordenadas:

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

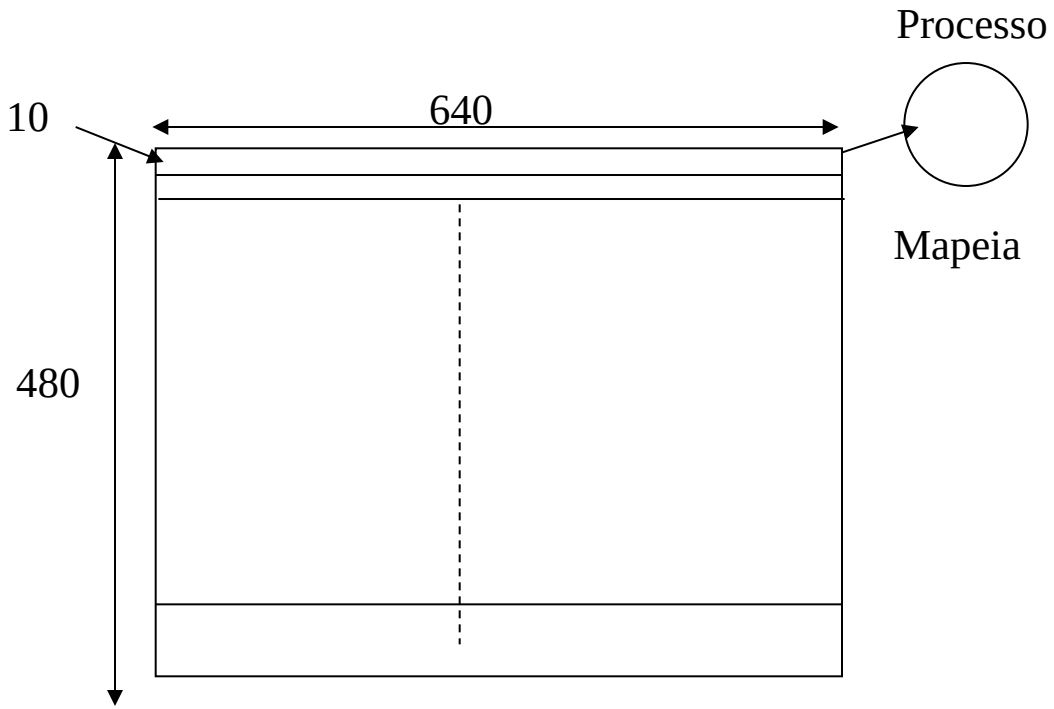
Supondo, uma imagem com 640 x 480 pixels:

Partição em regiões para os processos individuais



Podemos atribuir a cada processo uma secção de 80 x 80 pixels.

Partição em regiões para os processos individuais



Alternativamente podemos dividir a área em 48 linhas, de 640 x 10.

Supondo que usamos 1 processo master, 48 slaves e particionamos a imagem em grupos de 10 linhas, cada escravo processa a sua área devolvendo as novas coordenadas ao master para visualização:

Pseudocódigo para executar deslocamento de imagem

- Mestre

```
for (i=0, row=0; i < 48; i++, row = row+10) //para cada slave
    send (row, Pi); }        // envia número da 1ª linha, das 10 a processar
for (i=0; i < 480; i++)
    for (j=0; j < 640; j++)
        temp_map[i][j]=0;    // buffer para obter os resultados
for (i=0; i < (640*480); i++) {    // para cada pixel
    recv(oldrow, oldcol, newrow, newcol, Pany);   //recebe novas coord.
    if !((newrow < 0) || (newrow >=480) || (newcol < 0) || newcol >=640))
        temp_map[newrow][newcol]=map[oldrow][oldcol]; }
for (i=0; i < 480; i++)
    for (j=0; j < 640; j++)
        map[i][j]= temp_map[i][j]; // atualiza a imagem
```

Se imagem sai dos limites, fica a 0

Pseudocódigo para executar deslocamento de imagem

- Slave

```
recv(row, P_mestre);          // recebe nº de linha

for (oldrow=row; oldrow < (row +10); oldrow++)
    for (oldcol=0; oldcol < 640; oldcol++) {
                              // transforma as coordenadas
        newrow = oldrow + delta_x;
        newcol = oldcol + delta_y;
        // envia novas coord para o master
        send (oldrow, oldcol,newrow,newcol,P_mestre);
}
```

Pseudocódigo para executar deslocamento de imagem

Possíveis melhoramentos:

- A linha de início, depende do id do processo, pode ser determinado pelo processo.

- Os resultado são enviados um a um, podiam ser envidos em conjunto.

- Deve ser introduzido código para terminar os slaves.

- Parametrizar o tamanho da figura, número de linhas para cada processo, e numero de processos.

- …

Análise de complexidade

- Versão Sequencial

$t_s = 2\,n^2 = O(n^2)$ // n x n pixeis, 2 operações cada

- Versão Paralela
  - Comunicação

$t_{comm} = p\,(t_{startup} + t_{data}) + n^2\,(t_{startup} + 4\,t_{data}) = O(p + n^2)$

//Envio de p números de linha, recebe $4n^2$ coordenadas

  - Computação

$T_{comp} = 2\,(n^2/p) = O(n^2/p)$

  - Tempo total de execução

$$t_p = t_{comp} + t_{comm}$$

Para um número fixo de processos, p, $t_p = O(n^2)$

Speedup?, Rácio computaçao /comunicação ??

Mandelbrot

Conjunto de pontos no plano complexo que são quase estáveis (diminuem e aumentam, mas não passam de certo limite) quando calculados pela iteração da função,

$$z_{k+1} = z_k^2 + c$$

Com z_{k+1} a iteração (k + 1) do número complexo $z = a + b_i$ e c um número complexo que dá a posição do ponto no plano complexo. O valor inicial de z é zero.

As iterações continuam até que a magnitude de z seja maior que 2 ou o número de iterações atinja um determinado limite. A magnitude de *z é o comprimento do vetor z dado por:*

$$z_{\text{length}} = \sqrt{a^2 + b^2}$$

Mandelbrot

- Simplificando o cálculo de: $z_{k+1} = z_k^2 + c$

$$z^2 = a^2 + 2abi + b^2 i^2 = a^2 - b^2 + 2abi$$

Cálculo do valor de z para cada iteração

$$z_{real} = z_{real}^{\ 2} - z_{imag}^{\ 2} + c_{real}$$

$$z_{imag} = 2 z_{real} z_{imag}^{\ 2} + c_{imag}$$

Rotina sequencial para calcular um ponto (devolve o número de iterações)

```
structure complex {
float real;
float imag;
};
int cal_pixel(complex c)
{
int count, max;
complex z;
float temp, lengthsq;
max = 256;
z.real = 0; z.imag = 0;
count = 0;                          /* number of iterations */
do {
temp = z.real * z.real - z.imag * z.imag + c.real;
z.imag = 2 * z.real * z.imag + c.imag;
z.real = temp;
lengthsq = z.real * z.real + z.imag * z.imag;
count++;
} while ((lengthsq < 4.0) && (count < max));
return count;
}
```

Mudança de escala do sistema de coordenadas

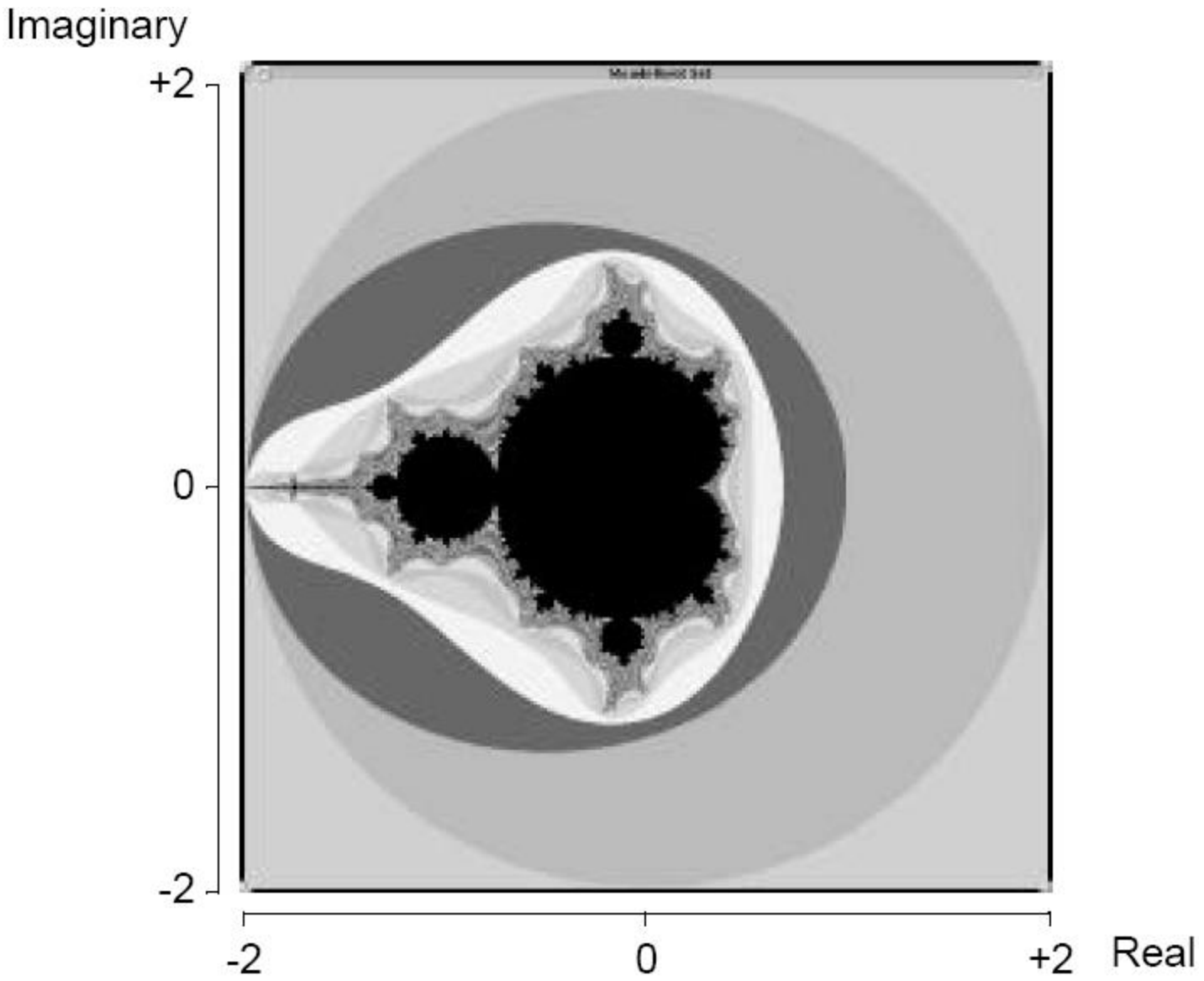
- Para visualizar os pontos numa janela com uma determinada área é necessário fazer uma mudança de escala (scaling):

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

- Incluindo a mudança de escala, temos o seguinte código

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

Mandelbrot set



Computação paralela do conjunto de Mandelbrot

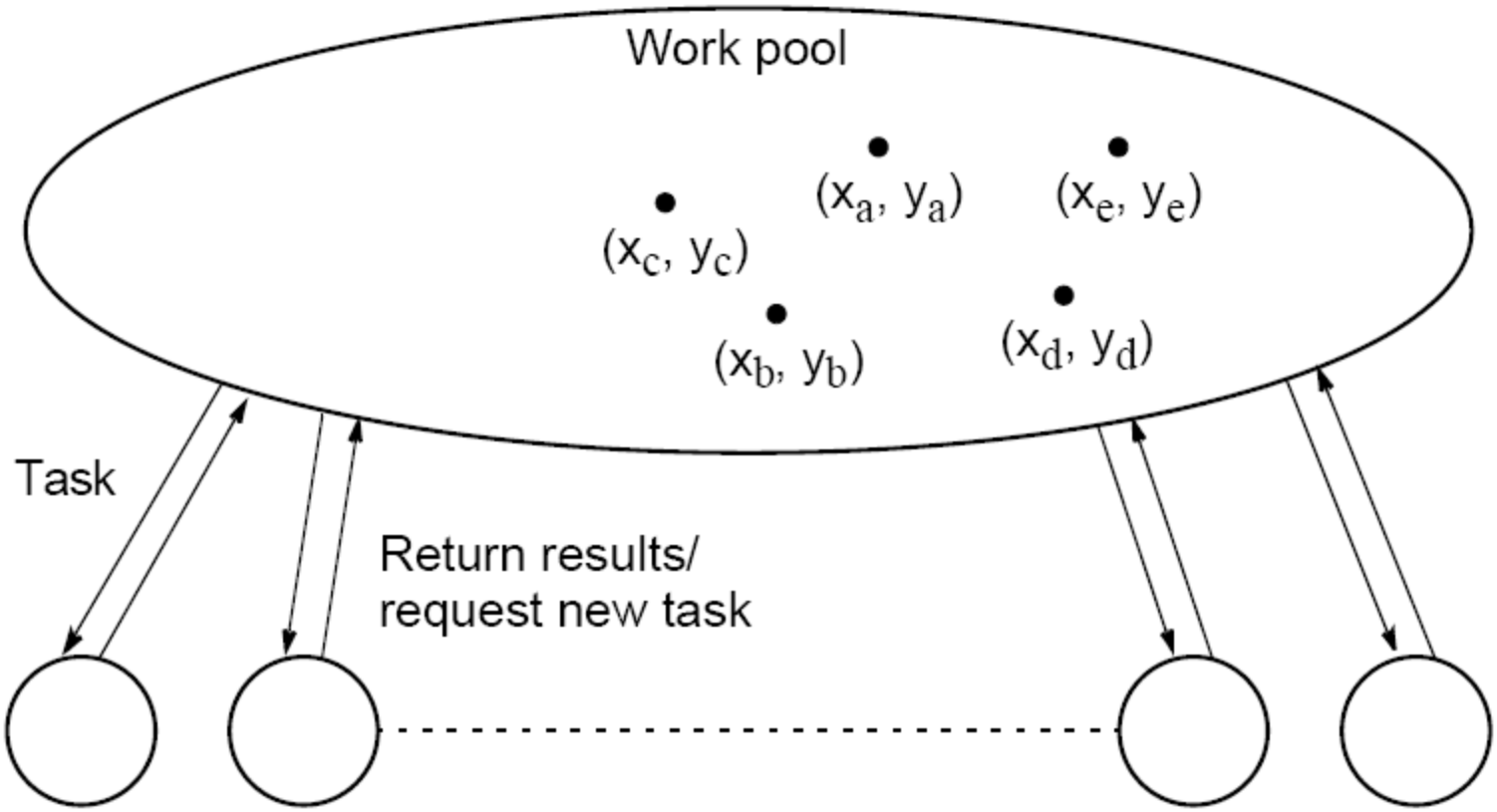
Atribuição estática de tarefas

Dividir a região num número fixo de secções. Cada secção é atribuída a um processo/processador

Não é muito eficiente porque cada região necessita um número diferente de iterações.

Atribuição dinâmica de tarefas

Quando um processo termina a sua tarefa, é-lhe atribuído um novo ponto para iterar.



Supondo que cada processo recebe as linhas a processar =>

Atribuição dinâmica de tarefas

- Mestre

```
count = 0;
row = 0;                                    //nº da linha
for (k=0; k < procno; k++) {                // procno – id do processo < disp_heigh
    send(&row, P_k, data_tag);              // envio da primeira linha para o proc
    row++;
 }
do {
        recv (&slave, &r, color, P_any, result_tag); //  recebe resultado
        count--;
        if (row < disp_height) {
           send (&row, P_escravo, data_tag);  // enviar nova linha
           row ++;
           count ++;
        } else
              send (&row, P_escravo, terminator_tag);   // terminar
        display ( r, color);
} while (count > 0);
```

Atribuição dinâmica de tarefas

- Escravo

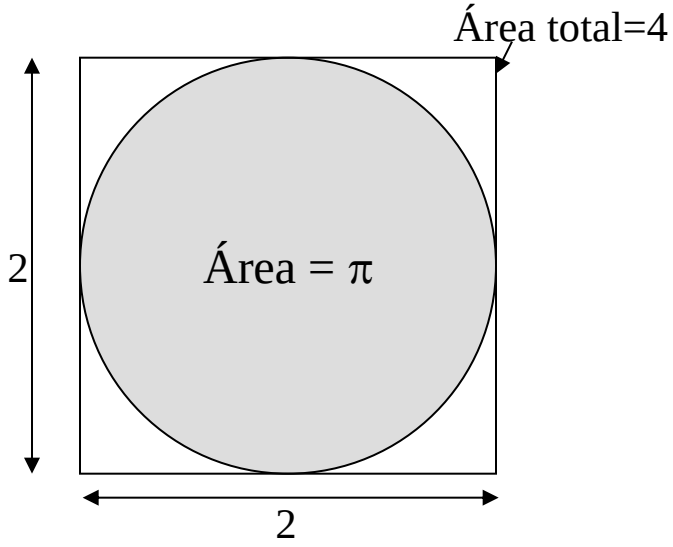
```
recv(y ,  P_mestre , source_tag);  // recebe 1ª linha

while (source_tag == data_tag) {
    //scaling
    c.imag = imag_min + ((float) y * scale_imag);
    for (x = 0; x < disp_width; x++) {
        c.real = real_min + ((float) x * scale_real);
         color[x] = cal_pixel ( c )   // calcula as cores da linha
     }
    send (&x, &y, color, P_mestre, result_tag);  //linha de cores p/ master
    recv (y, P_mestre, source_tag);
 };
```

Métodos de Monte Carlo

A base dos métodos de Monte Carlo é a geração de uma grande quantidade de amostragens aleatórias para se chegar a resultados próximos de resultados reais.

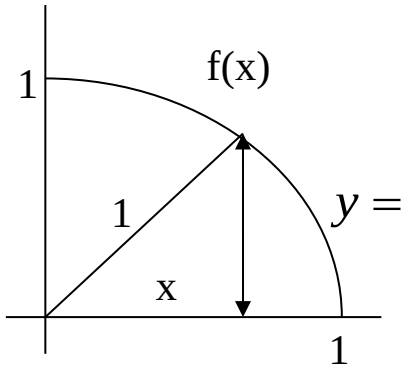
Exemplo – calcular o valor de π



$$\frac{\text{Área do círculo}}{\text{Área do quadrado}} = \frac{\pi(1)^2}{2 \times 2} = \frac{\pi}{4}$$

- Pontos dentro do quadrado são escolhidos aleatoriamente e verifica-se a fração desses pontos que está dentro do círculo
- Dado um número suficiente de amostras aleatórias, essa fração será $\pi/4$

Calcular um integral



$$y = \sqrt{1 - x^2}$$

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

Um par de números aleatórios (x_r, y_r) pode ser gerado entre 0 e 1 e contado como dentro do círculo se

$$y_r \leq \sqrt{1 - x_r^2}$$

ou seja, $y_r^2 + x_r^2 \leq 1$

Método Alternativo para calcular um integral

Usar valores aleatórios de x para calcular *f*(*x*) e somar os valores de *f*(*x*):

$$\text{Area} = \int_{x_1}^{x_2} f(x)\,dx = \lim_{N \to \infty} \frac{1}{N} \sum_{r=1}^{N} f(x_r)(x_2 - x_1)$$

com x_r *valores gerados aleatoriamente entre* x_1 e x_2.

Exemplo

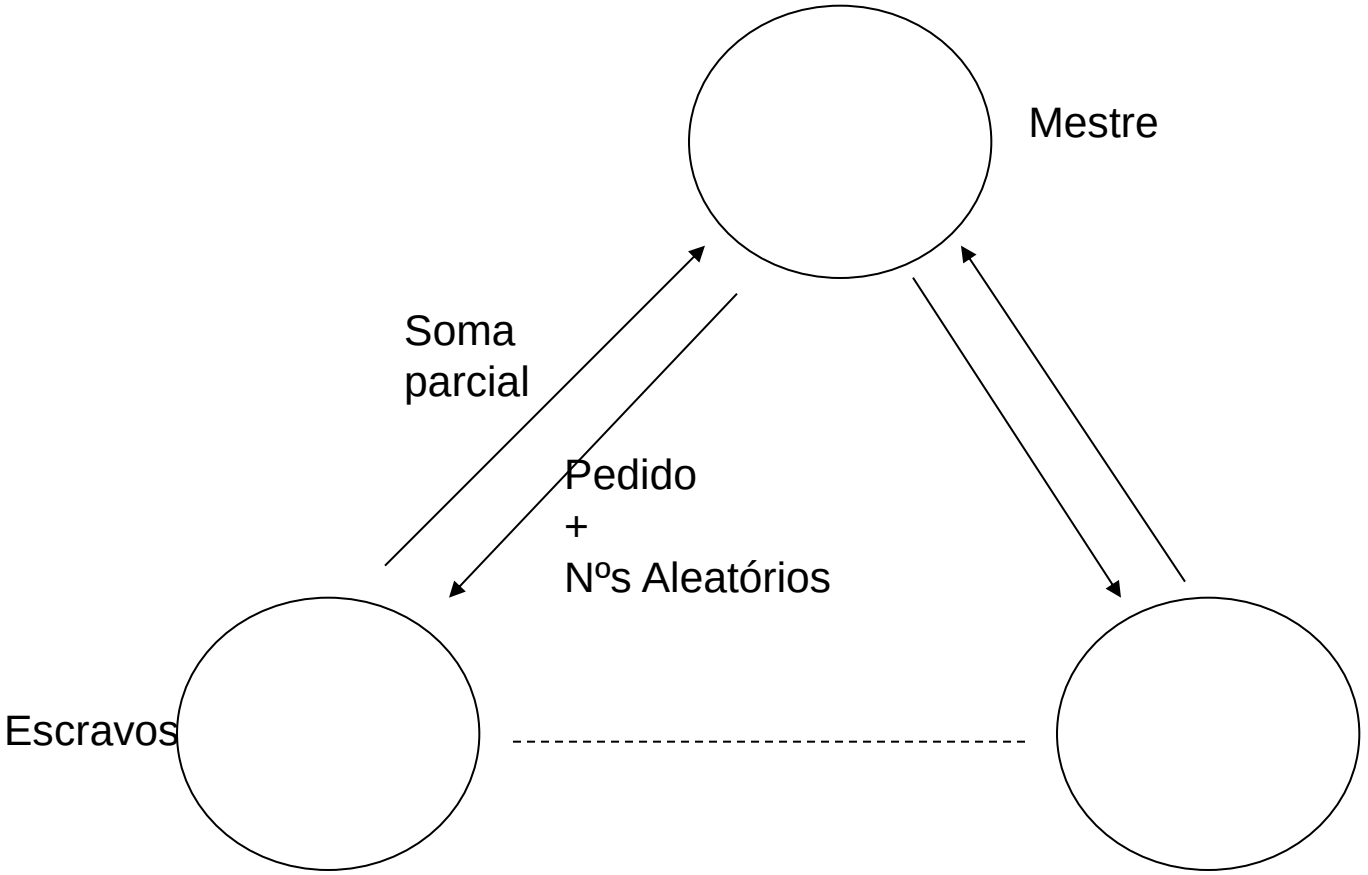
Calcular o integral:

$$I = \int_{x_1}^{x_2} (x^2 - 3x)\, dx$$

Versão sequencial

```
sum = 0;
for (i = 0; i < N; i++) {              /* N amostras aleatórias */
    xr = rand_v(x1, x2);               /* gerar próximo aleatório */
    sum = sum + xr * xr - 3 * xr;      /* calcular f(xr) e acumular */
}
area = (sum / N) * (x2 - x1);
```

Versão paralela



É necessário garantir que cada computação usa um número aleatório diferente e não há correlação entre eles.

Pseudocódigo

- Mestre

```
for (i = 0; i < N/n; i++) {        //  n = nº de aleatórios por escravo
    for (j = 0; j < n; j+=)
        xr[j] = rand [j];
    recv(P_any, req_tag, P_fonte);  // espera por pedido do escravo !!
    send(xr, &n, P_fonte, compute_tag);  // envia aleatórios
 }
 for (i = 0; i < slave_no; i++) {
     recv(P_i, req_tag);
     send(P_i, stop_tag);      // mensagem para terminar
  }
sum=0;
reduce_add(&sum, P_group);  // soma dos valores parciais
…
```

Pseudocódigo

- Escravo

```
sum = 0;
send(P_master, req_tag);
recv(xr, &n, P_mestre, source_tag);

while (source_tag == compute_tag) {
   for (i = 0; i < n; i++)
      sum = sum +xr[i] * xr[i] -3 * xr[i];
   send(P_mestre, req_tag);
   recv(xr, &n, P_master, source_tag);
}
reduce_add(&sum, P_group);
```