

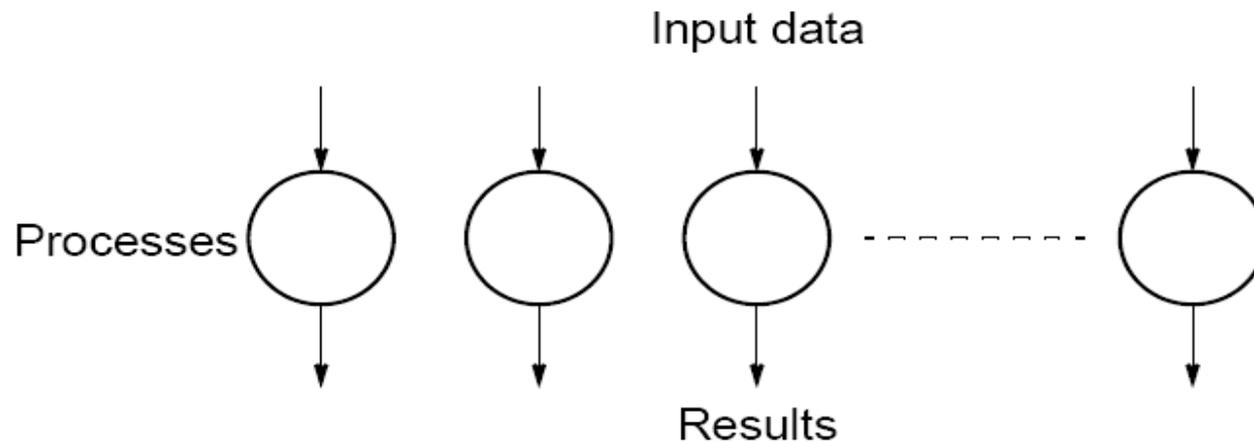
# Parallel Techniques

- Embarrassingly Parallel Computations
- Partitioning and Divide-and-Conquer Strategies
- Pipelined Computations
- Synchronous Computations
- Asynchronous Computations
- Load Balancing and Termination Detection

# Embarrassingly Parallel Computations

# Embarrassingly Parallel Computations

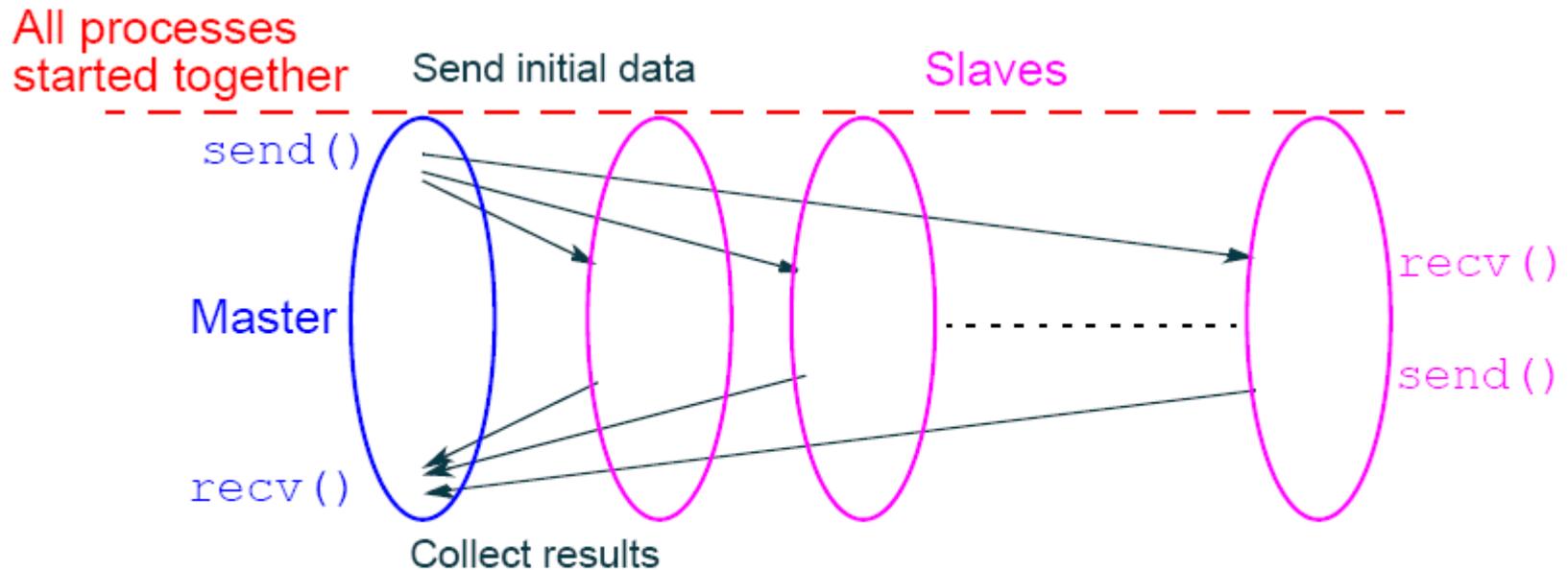
A computation that can **obviously** be divided into a number of completely independent parts, each of which can be executed by a separate process(or).



No communication or very little communication between processes

Each process can do its tasks without any interaction with other processes

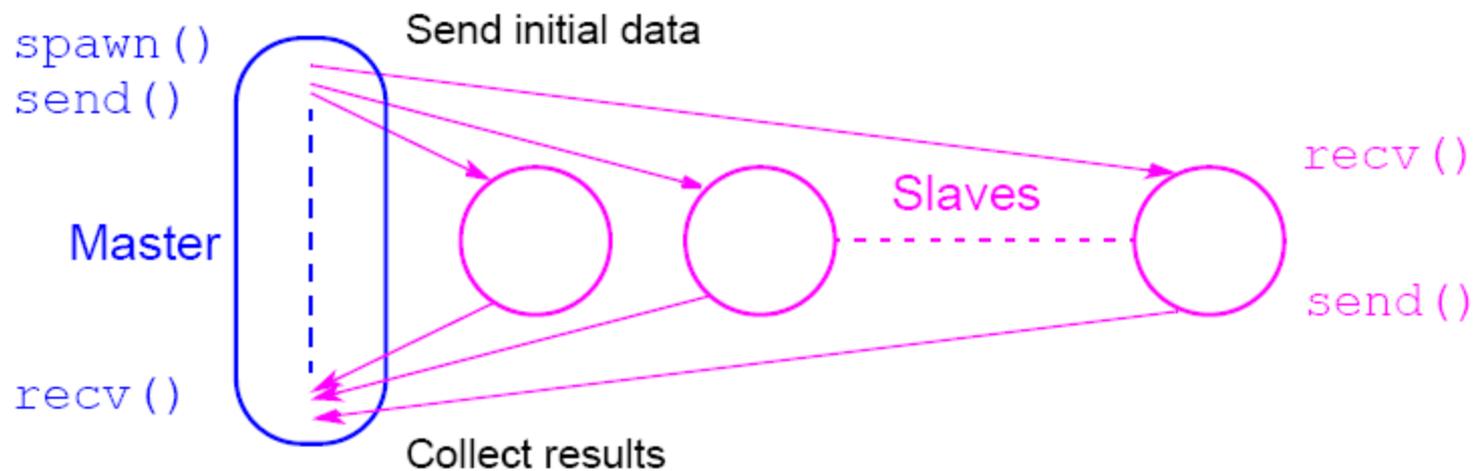
# Practical embarrassingly parallel computation with static process creation and master-slave approach



Usual MPI approach

# Practical embarrassingly parallel computation with dynamic process creation and master-slave approach

Start Master initially



(PVM approach)

# Embarrassingly Parallel Computation Examples

- Low level image processing
- Mandelbrot set
- Monte Carlo Calculations

# Low level image processing

Many low level image processing operations only involve local data with very limited if any communication between areas of interest.

# Some geometrical operations

## Shifting

Object shifted by  $\Delta x$  in the  $x$ -dimension and  $\Delta y$  in the  $y$ -dimension:

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

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

where  $x$  and  $y$  are the original and  $x'$  and  $y'$  are the new coordinates.

## Scaling

Object scaled by a factor  $S_x$  in  $x$ -direction and  $S_y$  in  $y$ -direction:

$$x' = xS_x$$

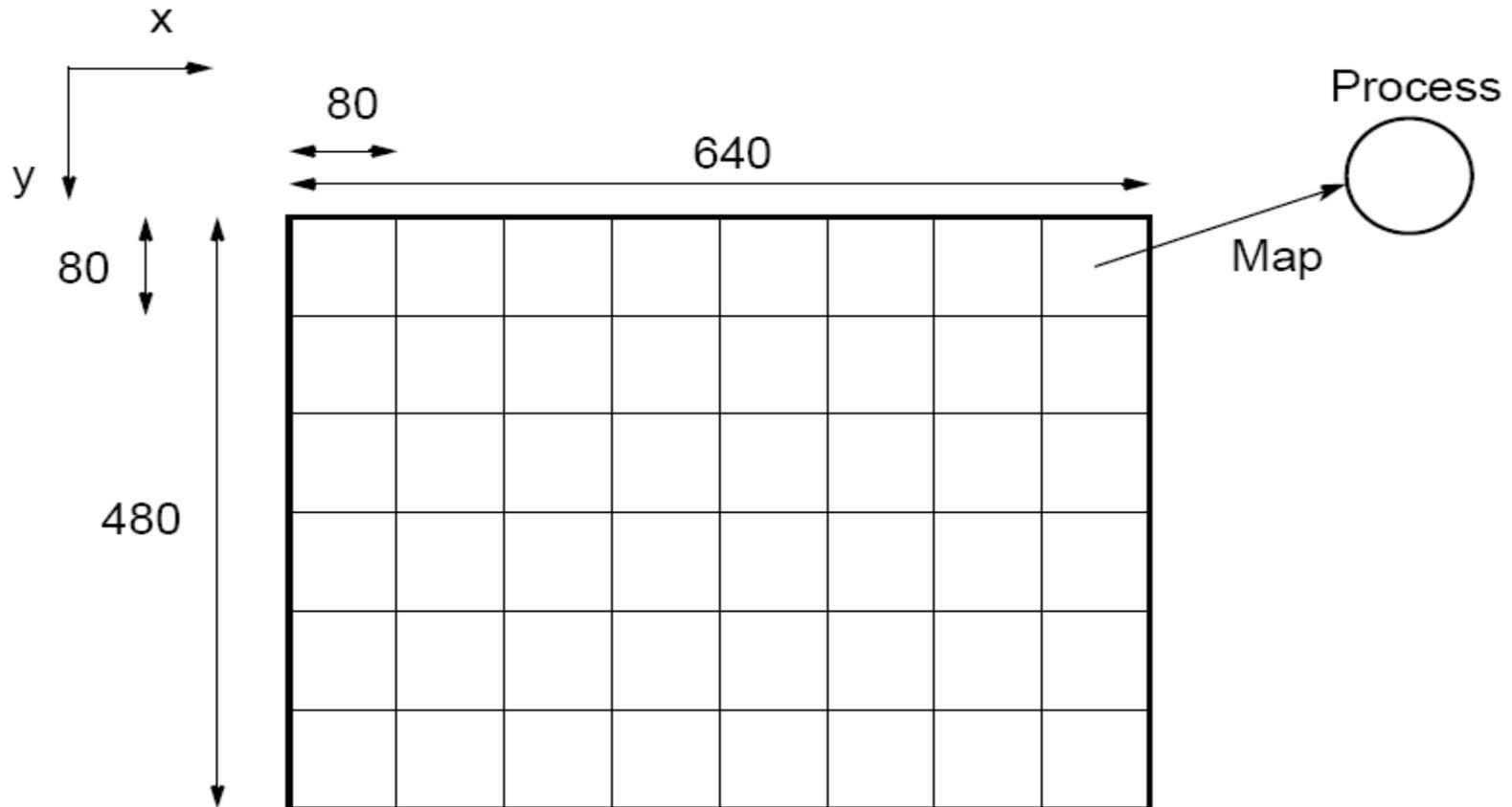
$$y' = yS_y$$

# Rotation

Object rotated through an angle  $\theta$  about the origin of the coordinate system:

$$\begin{aligned}x' &= x \cos\theta + y \sin\theta \\y' &= -x \sin\theta + y \cos\theta\end{aligned}$$

# Partitioning into regions for individual processes



Square region for each process (can also use strips)

# Mandelbrot Set

Set of points in a complex plane that are quasi-stable (will increase and decrease, but not exceed some limit) when computed by iterating the function

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

where  $z_{k+1}$  is the  $(k + 1)$ th iteration of the complex number  $z = a + bi$  and  $c$  is a complex number giving position of point in the complex plane. The initial value for  $z$  is zero.

Iterations continued until magnitude of  $z$  is greater than 2 or number of iterations reaches arbitrary limit. Magnitude of  $z$  is the length of the vector given by

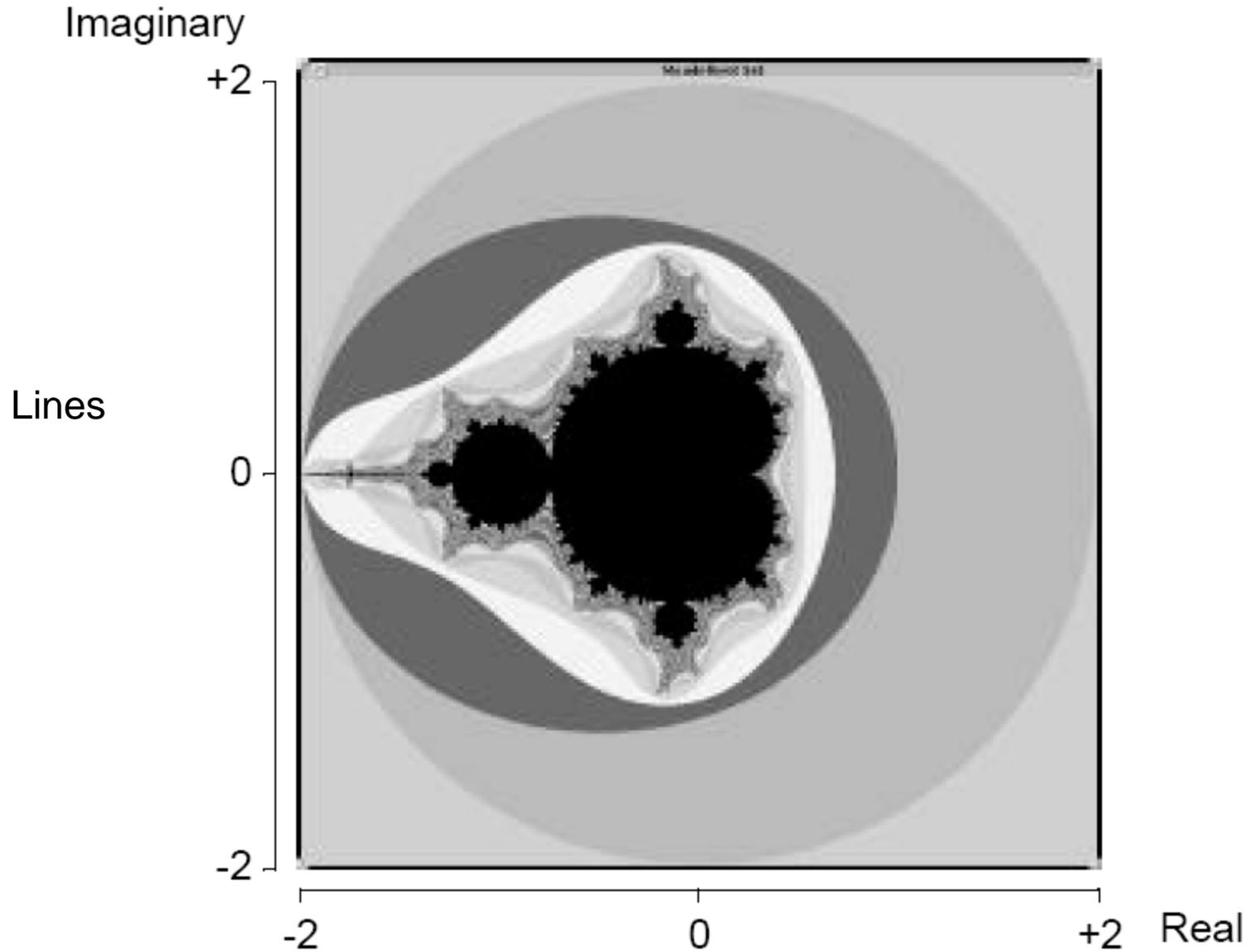
$$z_{\text{length}} = \sqrt{a^2 + b^2}$$

Plot the number of iterations by assigning colors

# Sequential routine computing value of one point returning number of iterations

```
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;
}
```

# Mandelbrot set



# Parallelizing Mandelbrot Set Computation

## Static Task Assignment

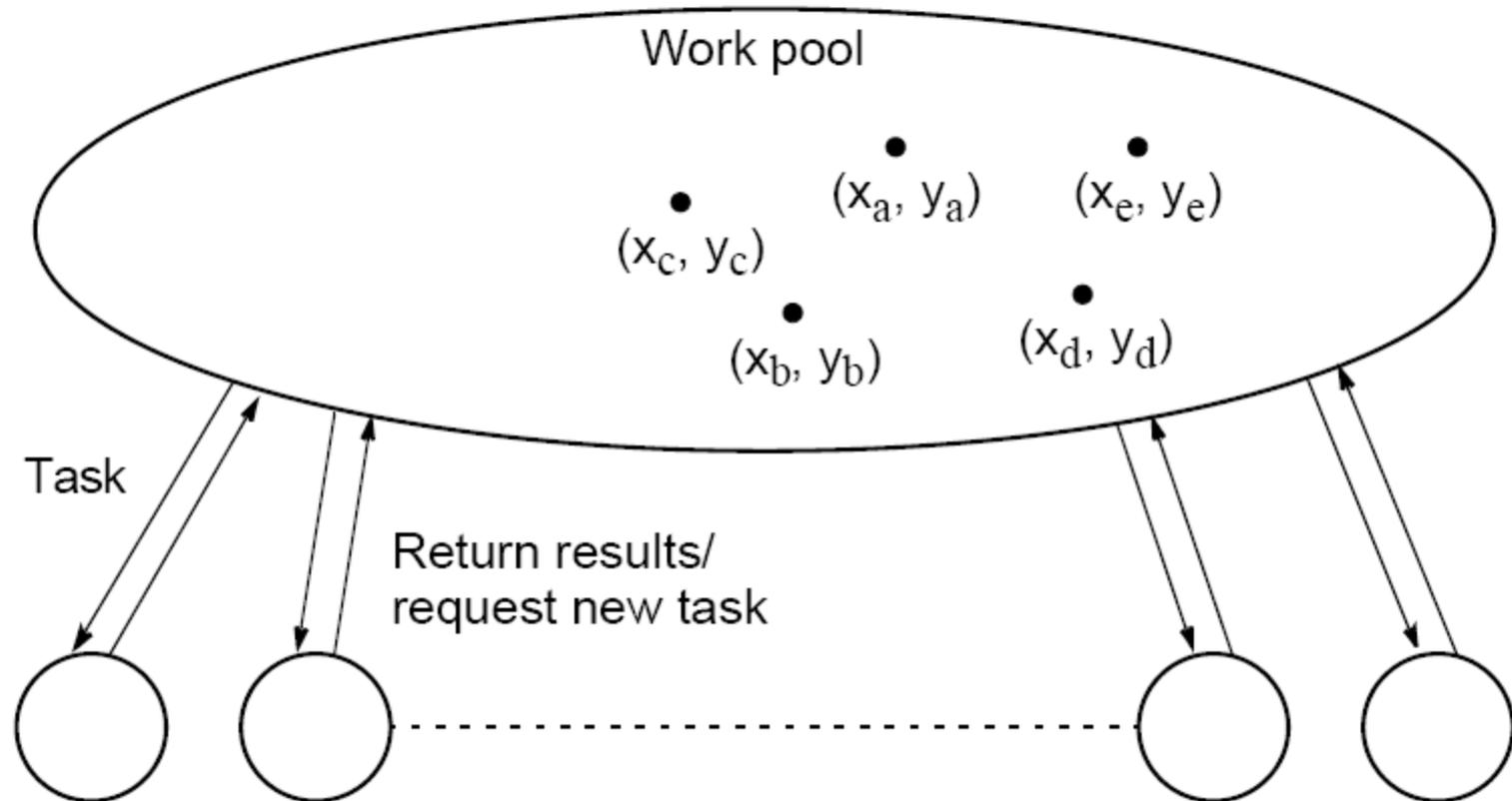
Simply divide the region in to fixed number of parts, each computed by a separate processor.

Not very successful because different regions require different numbers of iterations and time.

## Dynamic Task Assignment

Have processor request regions after computing previous regions

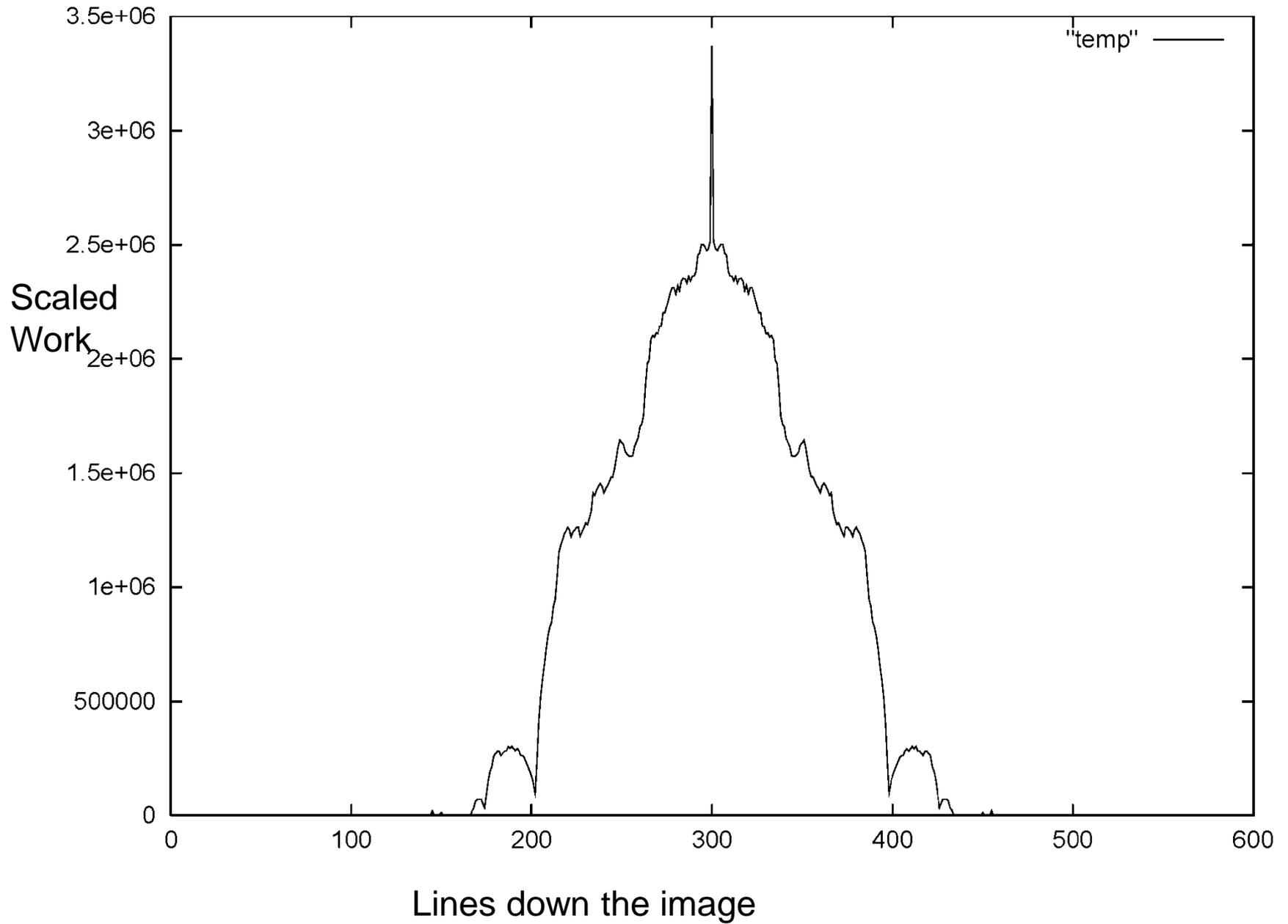
# Dynamic Task Assignment Work Pool/Processor Farms



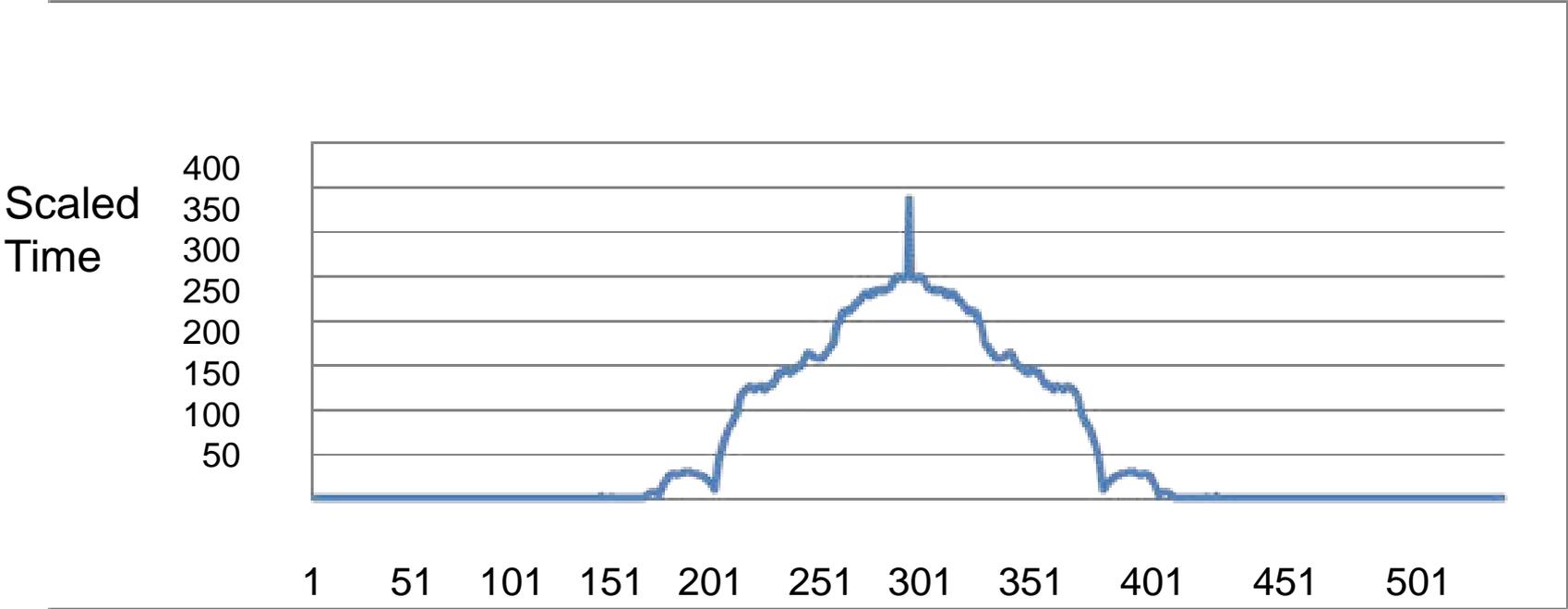
# Allocation Options

- Static allocation – use symmetry
- Use cyclic allocation
- Random allocation can work well when no obvious choices.

# Work for Mandelbrot Set Example

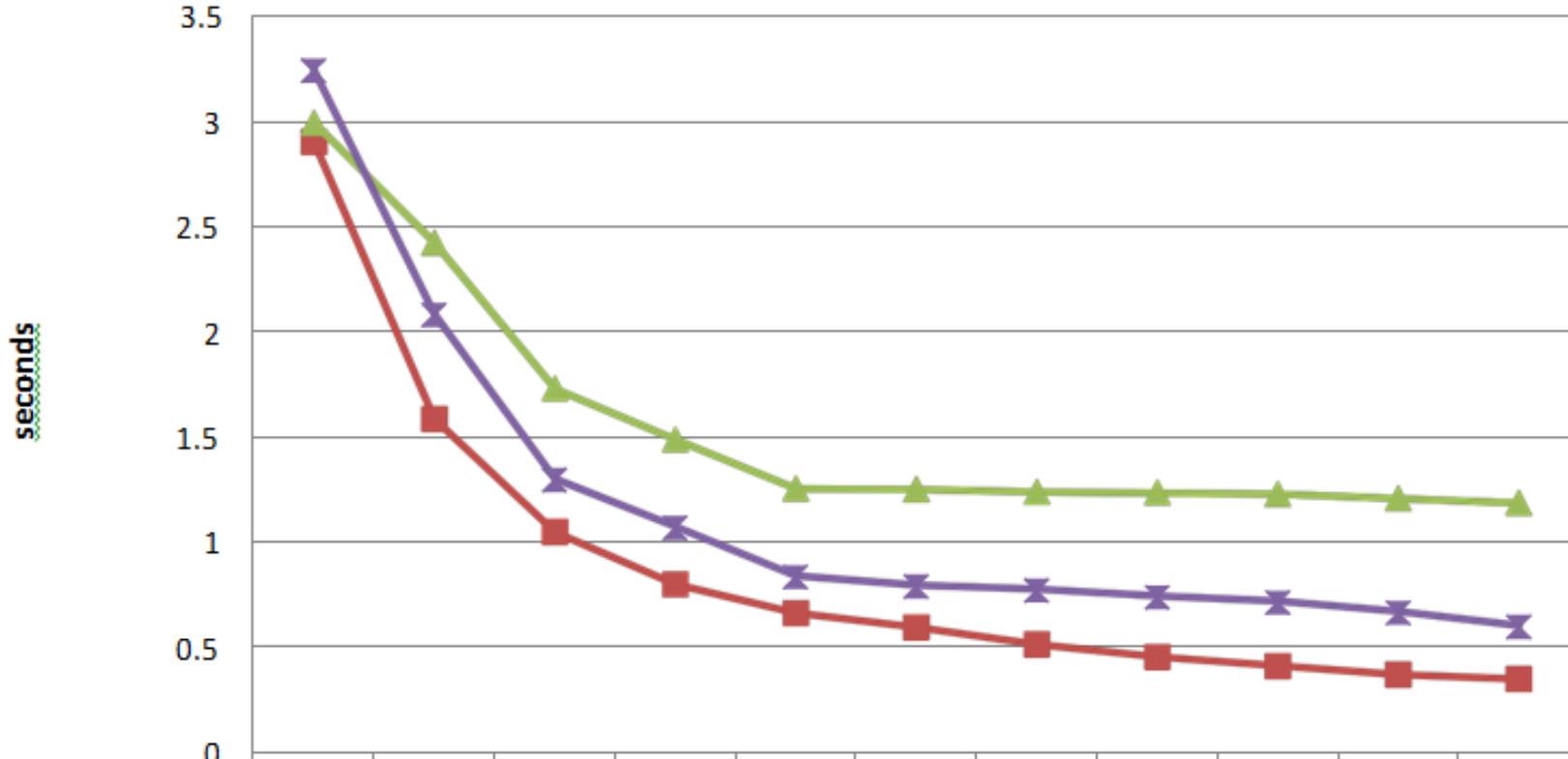


# Work Distribution for Mandelbrot Set



Lines down the image

# Overall time



	2	4	6	8	10	12	14	16	18	20	22
static	2.901	1.58	1.042	0.792	0.655	0.587	0.506	0.446	0.402	0.362	0.341
work pool	2.992	2.42	1.725	1.482	1.248	1.245	1.232	1.229	1.222	1.202	1.18
random	3.243	2.08	1.295	1.067	0.833	0.788	0.769	0.737	0.712	0.663	0.596

Running result from raven cluster

# Monte Carlo Methods

Another embarrassingly parallel computation.

Monte Carlo methods use of random selections.

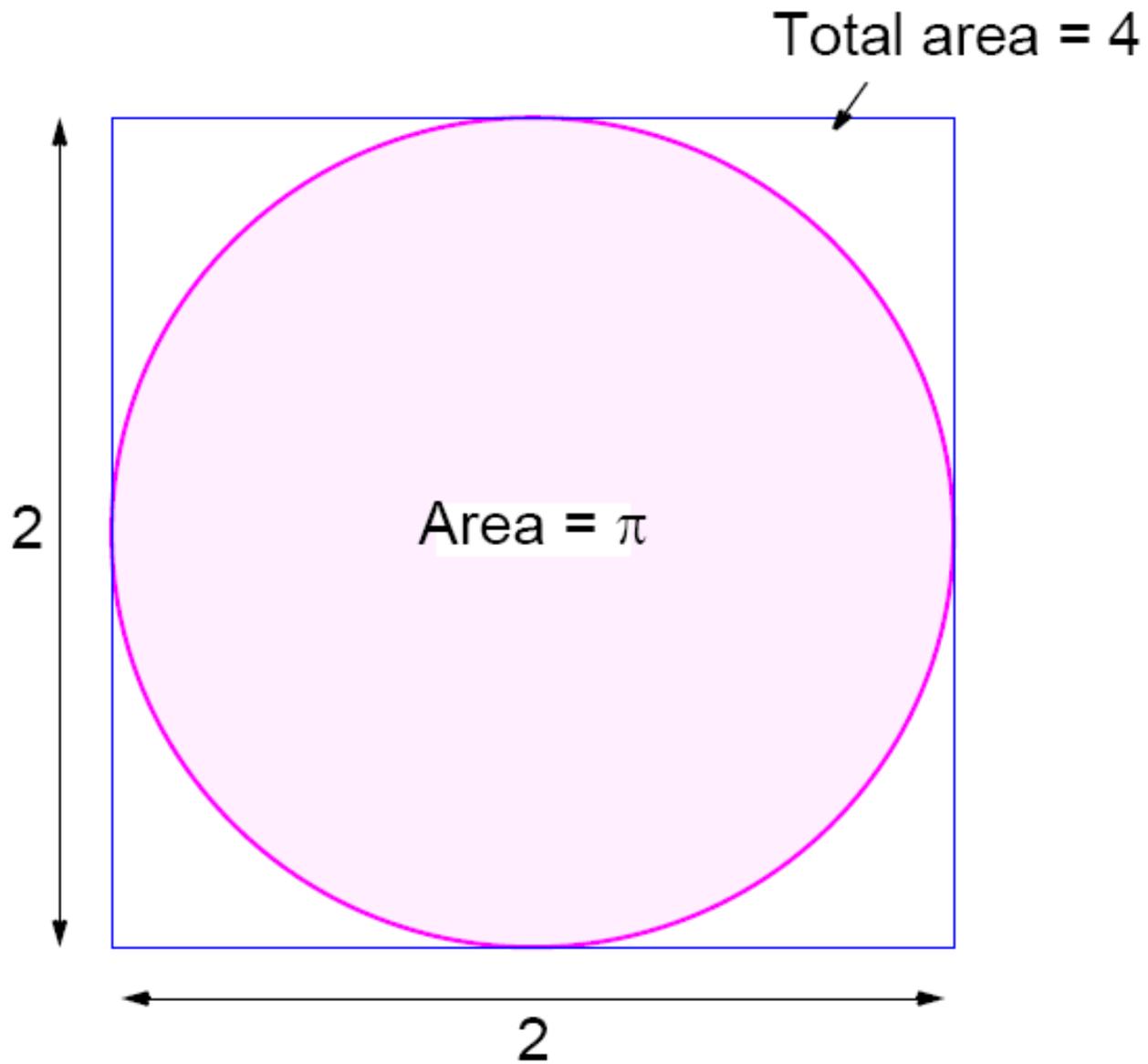
# Example - To calculate $\pi$

Circle formed within a 2 x 2 square. Ratio of area of circle to square given by:

$$\frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi(1)^2}{2 \times 2} = \frac{\pi}{4}$$

Points within square chosen randomly. Score kept of how many points happen to lie within circle.

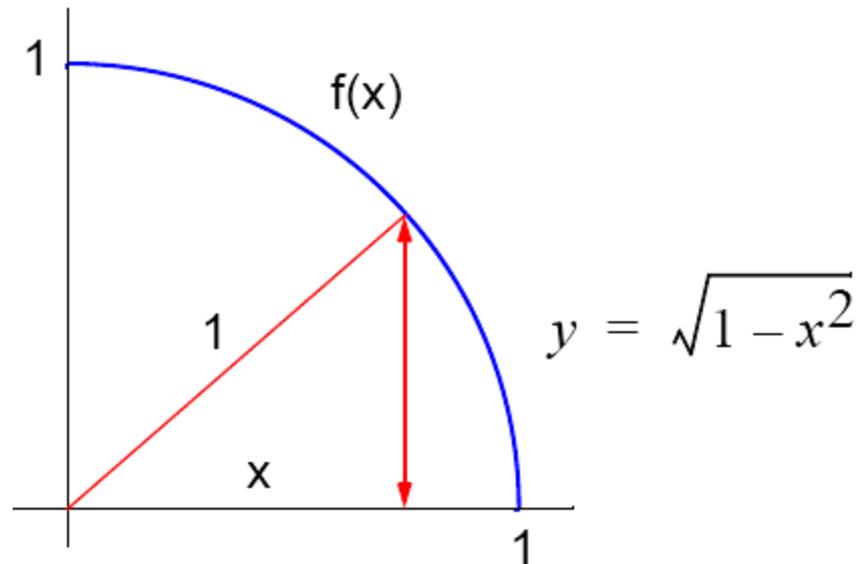
Fraction of points within the circle will be  $\pi/4$  , given a sufficient number of randomly selected samples.



# Computing an Integral

One quadrant can be described by integral

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



Random pairs of numbers,  $(x_r, y_r)$  generated, each between 0 and 1.

Counted as in circle if  $y_r \leq \sqrt{1-x_r^2}$ ; that is,  $y_r^2 + x_r^2 \leq 1$ .

# Alternative (better) Method

Use random values of  $x$  to compute  $f(x)$  and sum values of  $f(x)$ :

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

where  $x_r$  are randomly generated values of  $x$  between  $x_1$  and  $x_2$ .

Monte Carlo method very useful if the function cannot be integrated numerically (maybe having a large number of variables)

# Example

Computing the integral

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

## Sequential Code

```
sum = 0;
for (i = 0; i < N; i++) {           /* N random samples */
xr = rand_v(x1, x2);               /* generate next random value */
sum = sum + xr * xr - 3 * xr;      /* compute f(xr) */
}
area = (sum / N) * (x2 - x1);
```

Routine randv(x1, x2) returns a pseudorandom number between x1 and x2.

*For parallelizing Monte Carlo code, must address best way to generate random numbers in parallel - see textbook*