Partitioning and Divide-and-Conquer

Thoai Nam

High Performance Computing Lab (HPC Lab)

Faculty of Computer Science and Technology

HCMC University of Technology

Introduction

Partitioning

Partitioning simply divides the problem into parts.

Divide and Conquer

Characterized by dividing problem into sub-problems of same form as larger problem. Further divisions into still smaller sub-problems, usually done by recursion.

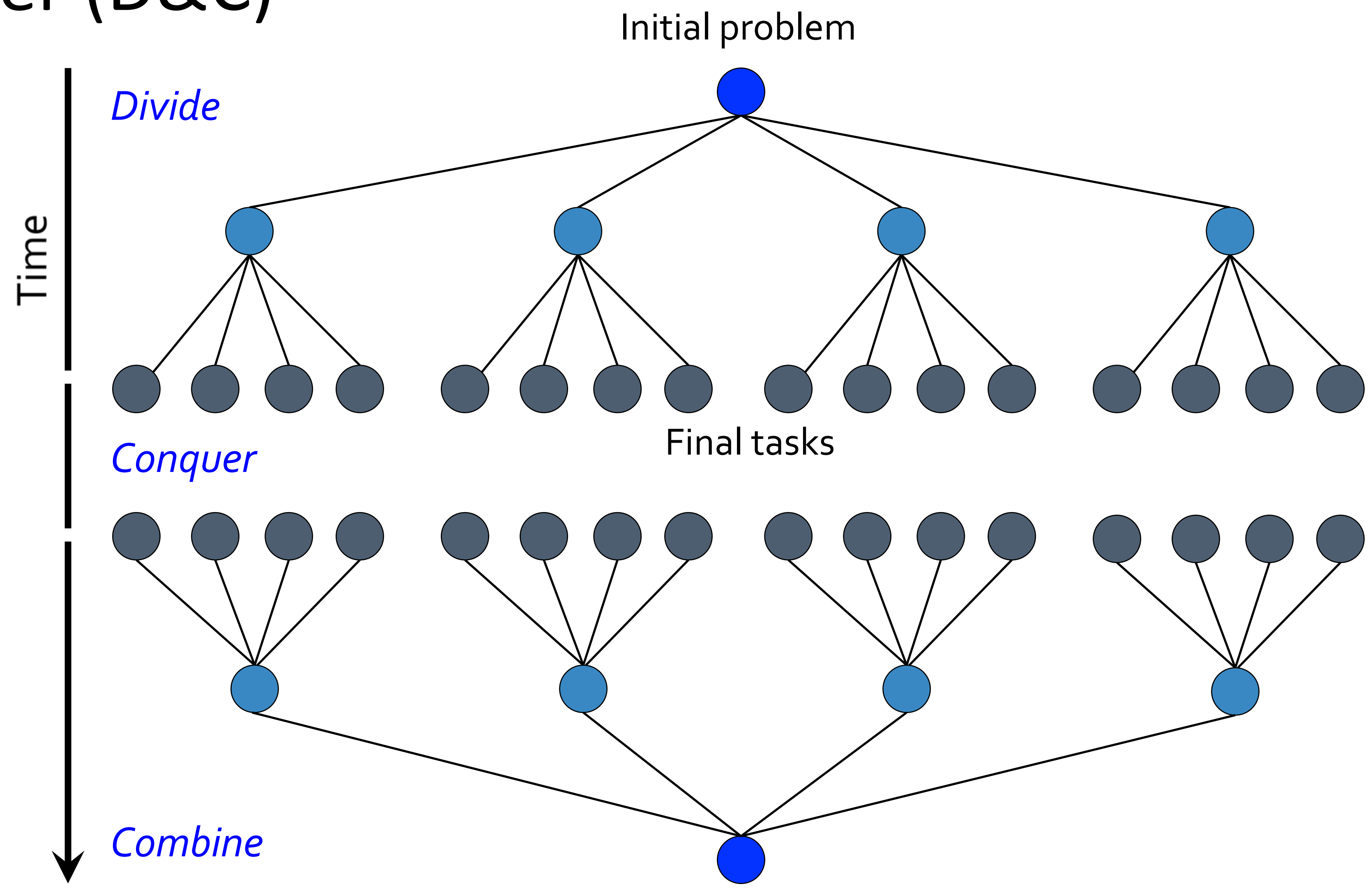
- ✓ Recursive divide and conquer amenable to parallelization because separate processes can be used for divided parts
- ✓ Also usually data is naturally localized.

Partitioning

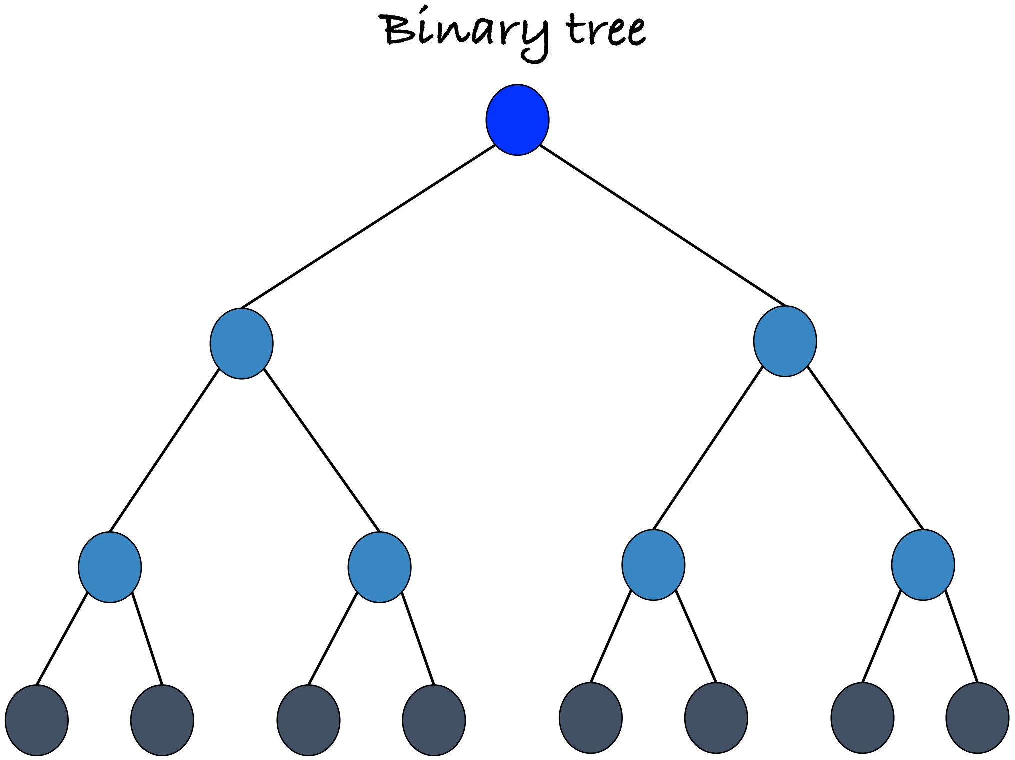
- Data partitioning or domain decomposition
    - Used in EPC?
- Functional decomposition

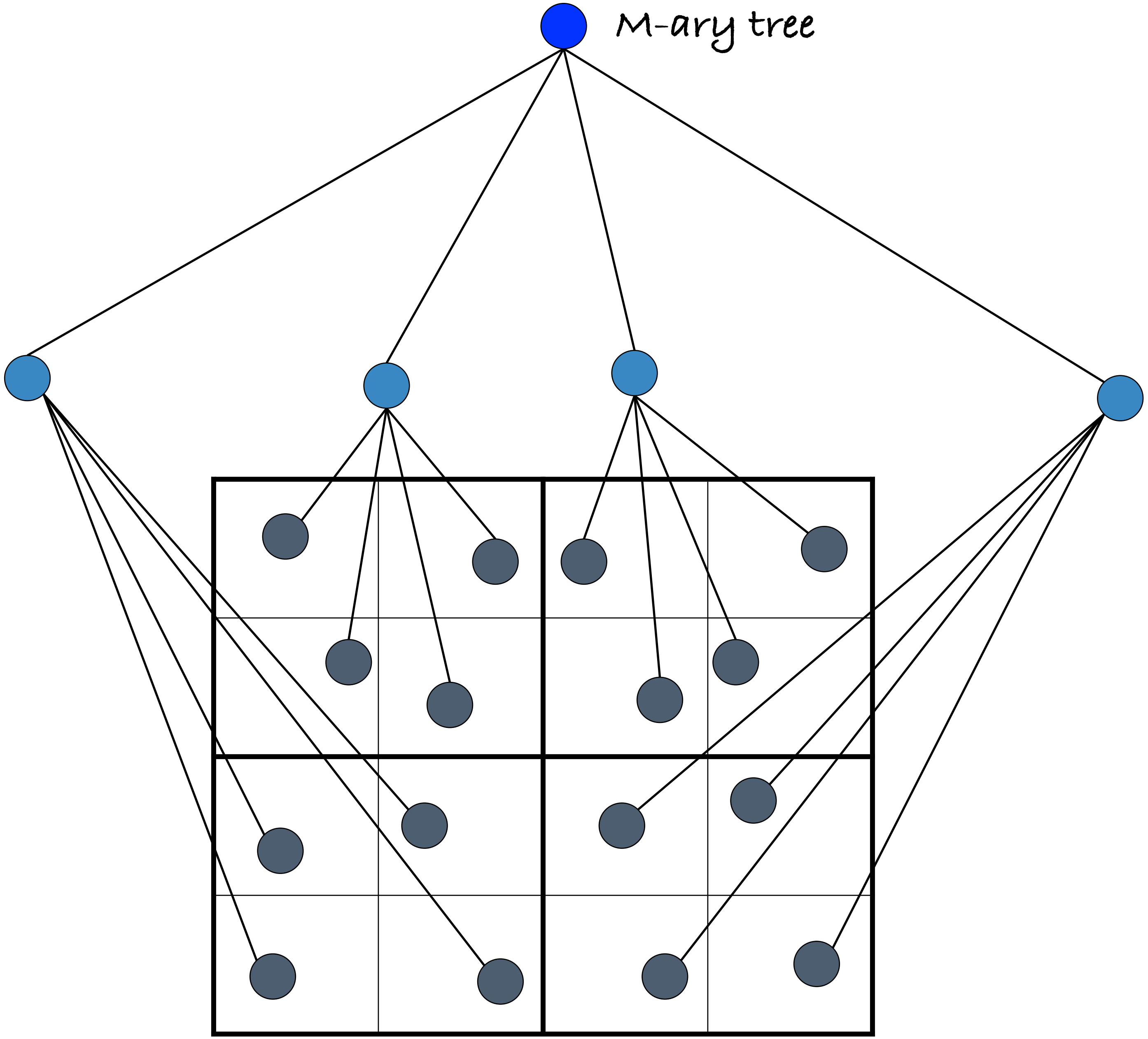
Divide-and-Conquer (D&C)

- Divide
  - Divide a problem into sub-problems that are of the same form as the large problem
  - Recursion: smallest sub-problems called final tasks
- Conquer
  - Run final tasks
- Combine
  - Aggregation

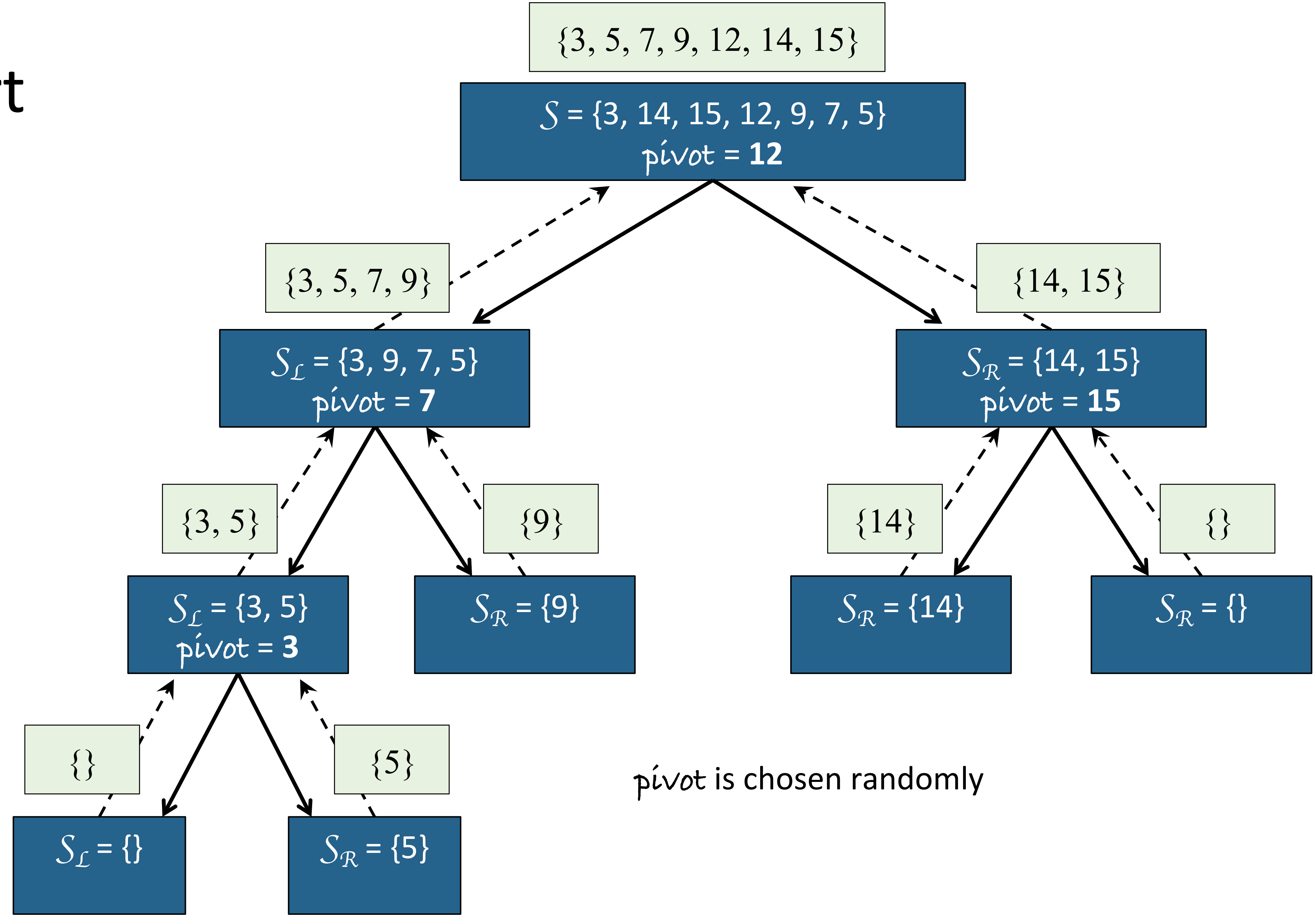


D&C trees





D&C Quicksort



pivot is chosen randomly

D&C matrix multiplication

- Divide
  - Divide A, B sizeof $n \times n$ into 8 matrix sizeof $n/2 \times n/2$:

    $A^{11}, A^{12}, A^{21}, A^{22}, B^{11}, B^{12}, B^{21}, B^{22}$
  - Matrix multiplication

$$X^{11} = A^{11}.B^{11} \qquad Y^{11} = A^{12}.B^{21}$$
$$X^{12} = A^{11}.B^{12} \qquad Y^{12} = A^{12}.B^{22}$$
$$X^{21} = A^{21}.B^{11} \qquad Y^{21} = A^{22}.B^{21}$$
$$X^{22} = A^{21}.B^{12} \qquad Y^{22} = A^{22}.B^{22}$$

- Combine

$$C^{11} = X^{11} + Y^{11} \qquad C^{12} = X^{12} + Y^{12}$$
$$C^{21} = X^{21} + Y^{21} \qquad C^{22} = X^{22} + Y^{22}$$

$$A = \begin{pmatrix} A^{11} & A^{12} \\ A^{21} & A^{22} \end{pmatrix},\ B = \begin{pmatrix} B^{11} & B^{12} \\ B^{21} & B^{22} \end{pmatrix},\ C = A\,.\,B = \begin{pmatrix} C^{11} & C^{12} \\ C^{21} & C^{22} \end{pmatrix}$$

$$C^{11} = A^{11}.B^{11} + A^{12}.B^{21} \qquad C^{12} = A^{11}.B^{12} + A^{12}.B^{22}$$
$$C^{21} = A^{21}.B^{11} + A^{22}.B^{21} \qquad C^{22} = A^{21}.B^{12} + A^{22}.B^{22}$$

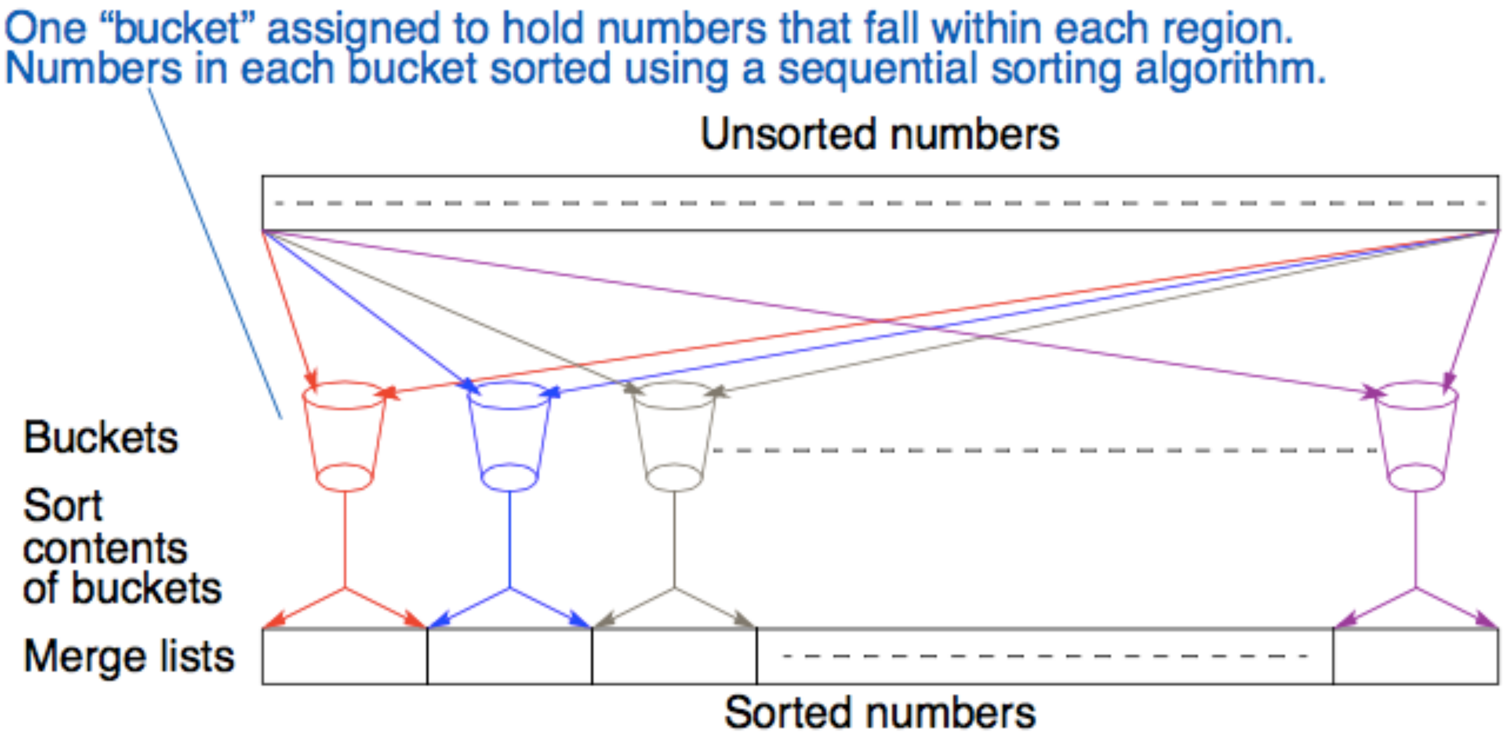
- Conquer
  - If matrix size is small enough, then C = A.B
  - Others, recursion & parallel computation

$$X^{11} = A^{11}.B^{11} \qquad Y^{11} = A^{12}.B^{21}$$
$$X^{12} = A^{11}.B^{12} \qquad Y^{12} = A^{12}.B^{22}$$
$$X^{21} = A^{21}.B^{11} \qquad Y^{21} = A^{22}.B^{21}$$
$$X^{22} = A^{21}.B^{12} \qquad Y^{22} = A^{22}.B^{22}$$

D&C matrix multiplication algorithm

```
Matrix_Multiplication (A, B, C, n) {    // n là kích thước
        if (n ≤ n0) { // n0 là kích thước bài toán nhỏ nhất
            C = A.B;  // Giải thuật tuần tự
            return;
        }
    // Bước (1)
        Phân rả A, B thành A11,A12,A21,A22,B11,B12,B21,B22 với
                kích thước n/2xn/2;
    // Bước (2)
        #pragma parallel for  // Song song vòng lặp for
        for (i=0; i<8; i++) {
            Matrix_Multiplication(A11, B11, X11, n/2);
            Matrix_Multiplication(A11, B12, X12, n/2);
            Matrix_Multiplication(A21, B11, X21, n/2);
            Matrix_Multiplication(A21, B12, X22, n/2);
            Matrix_Multiplication(A12, B21, Y11, n/2);
            Matrix_Multiplication(A12, B22, Y12, n/2);
            Matrix_Multiplication(A21, B22, Y21, n/2);
            Matrix_Multiplication(A22, B22, Y22, n/2);
        }
    // Bước (3)
        C11 = X11 + Y11;
        C12 = X12 + Y12;
        C21 = X21 + Y21;
        C22 = X22 + Y22;
        return;
}
```

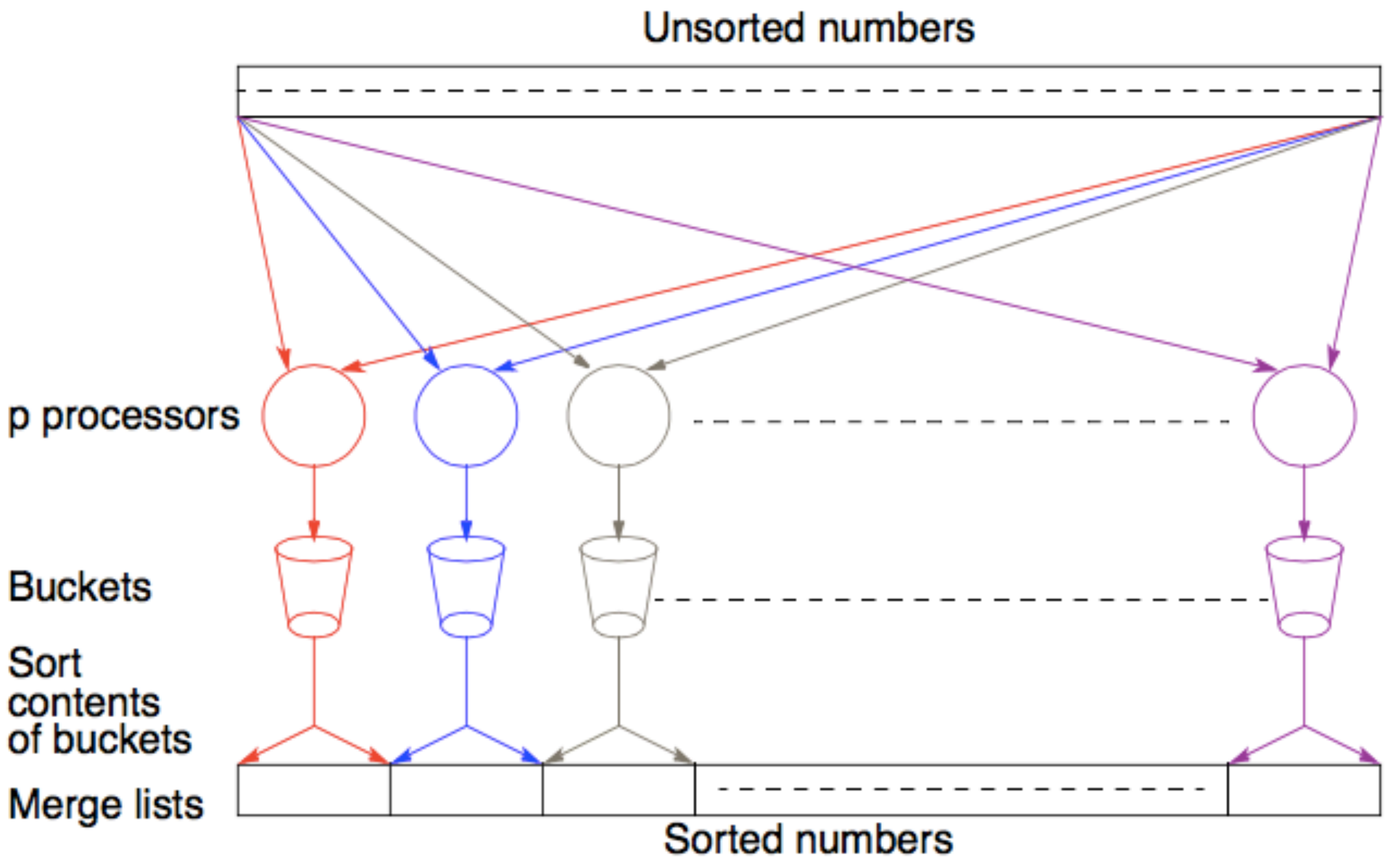
Bucket sort



- Sequential sorting time complexity: $O(n\log(n/m))$
- Works well if the original numbers uniformly distributed across a known interval, say 0 to a−1

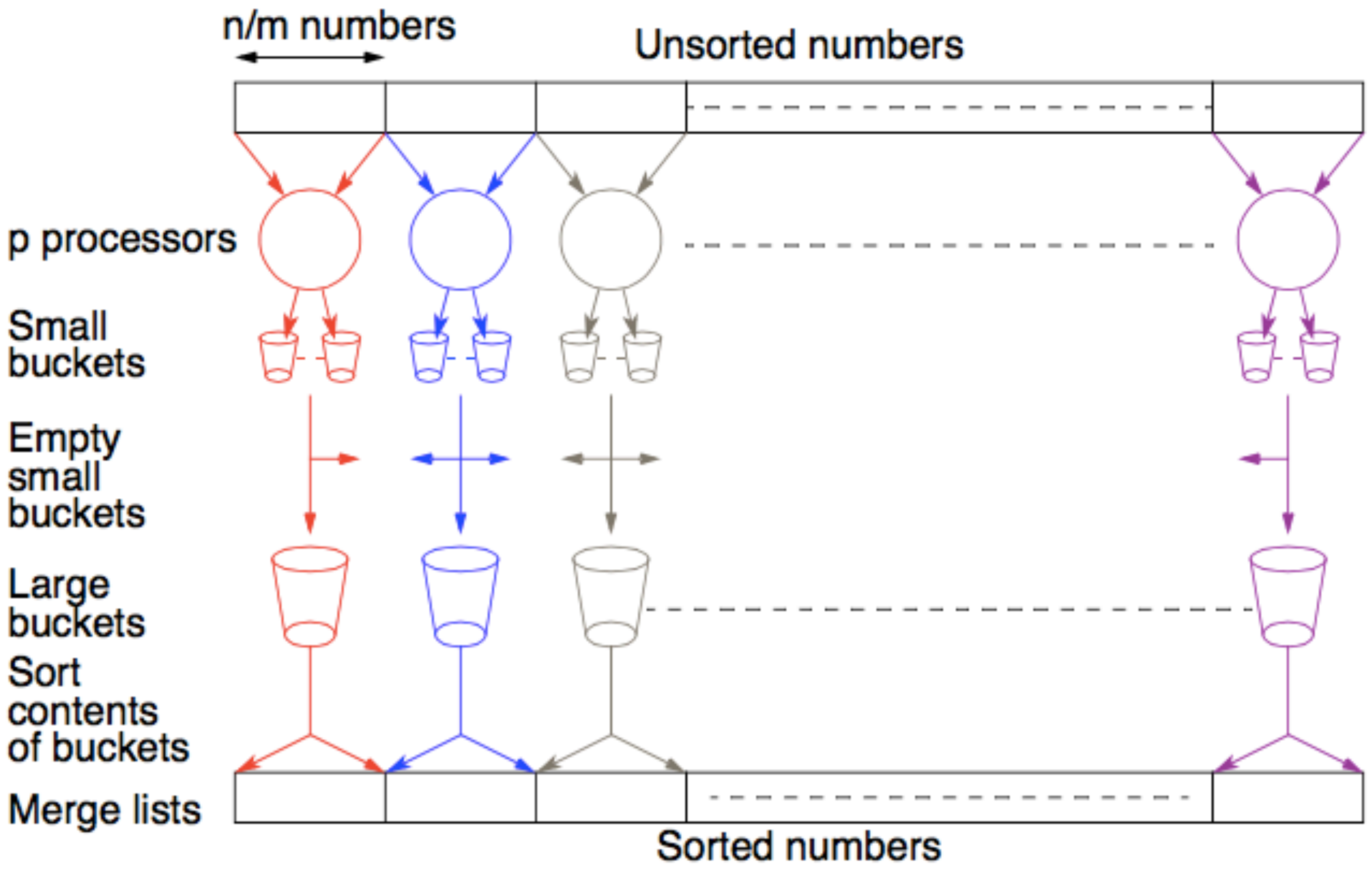
Parallel Bucket sort: version 1

- Assign one processor for each bucket



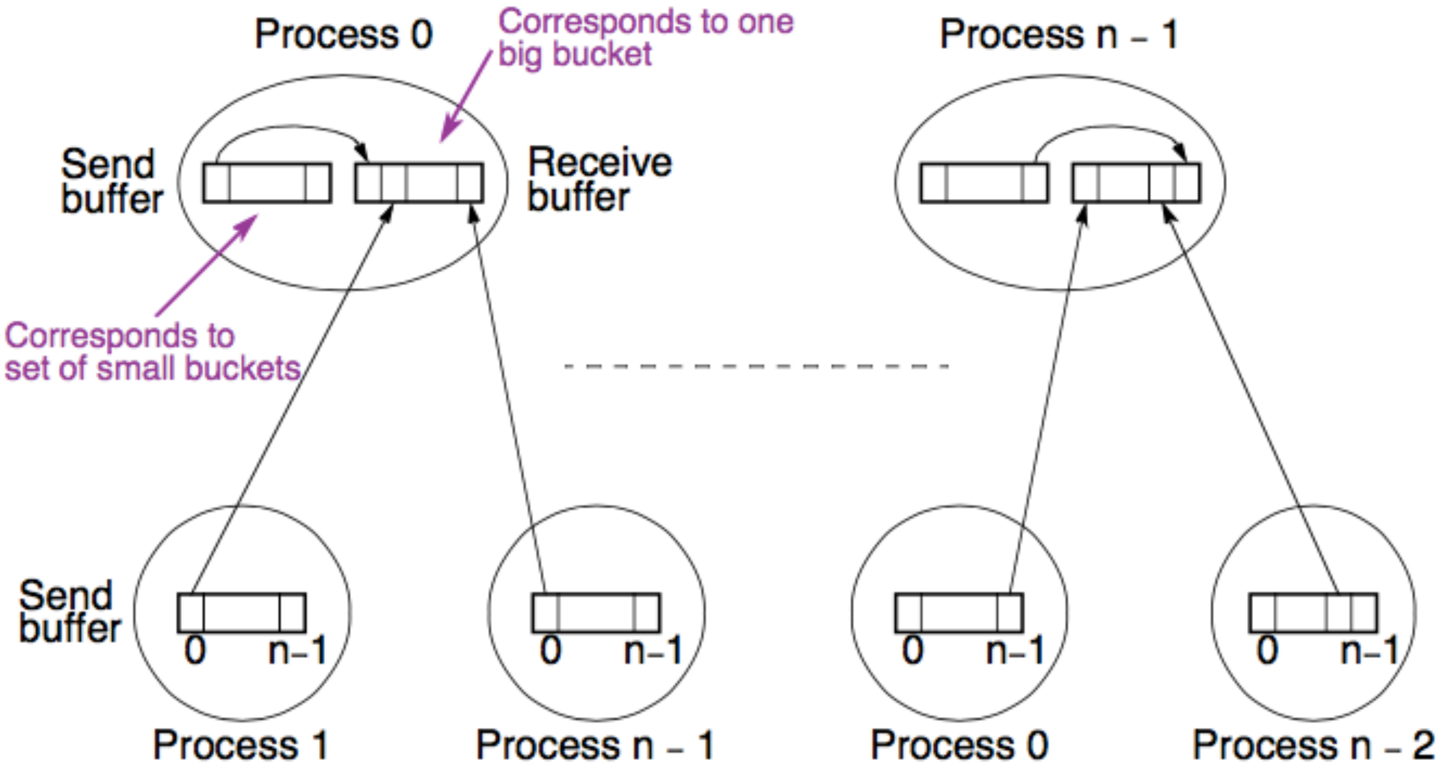
Parallel Bucket sort: version 2

- Partition sequence into m regions, one region for each processor
- Each processor maintains p "small" buckets and separates the numbers in its region into its own small buckets
- Small buckets then emptied into p final buckets for sorting, which requires each processor to send one small bucket to each of the other processors (bucket i to processor i)



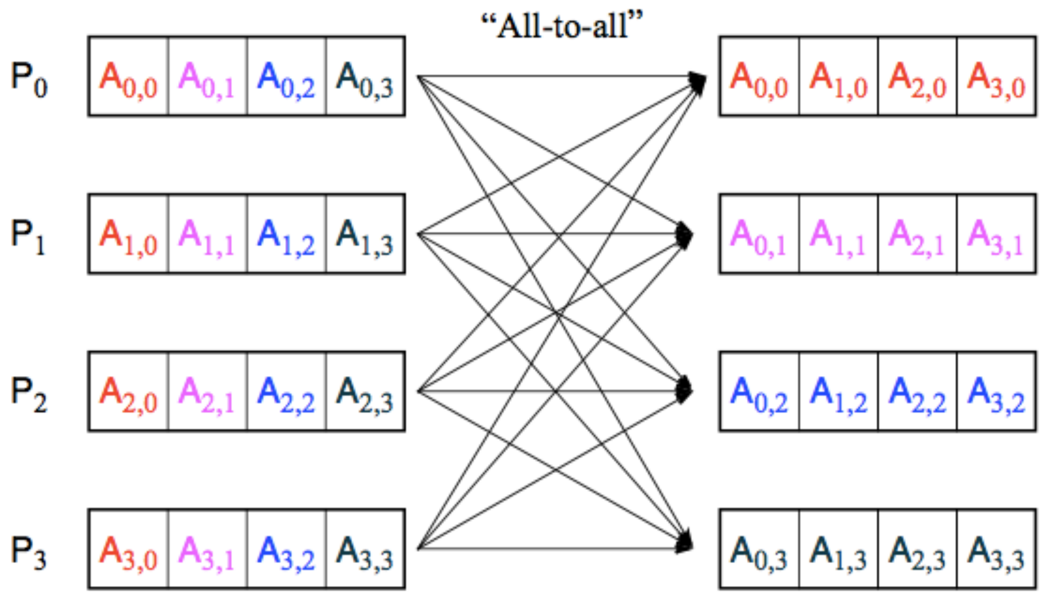
- Message-passing operation
  all-to-all broadcast

All-to-all broadcast



"all-to-all" routine actually transfers rows of an array to columns

Transpose a matrix



Numerical integration using rectangles

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

- Each region calculated using an approximation given by rectangles
- Aligning the rectangles: (a) & (b)

Figure (a)

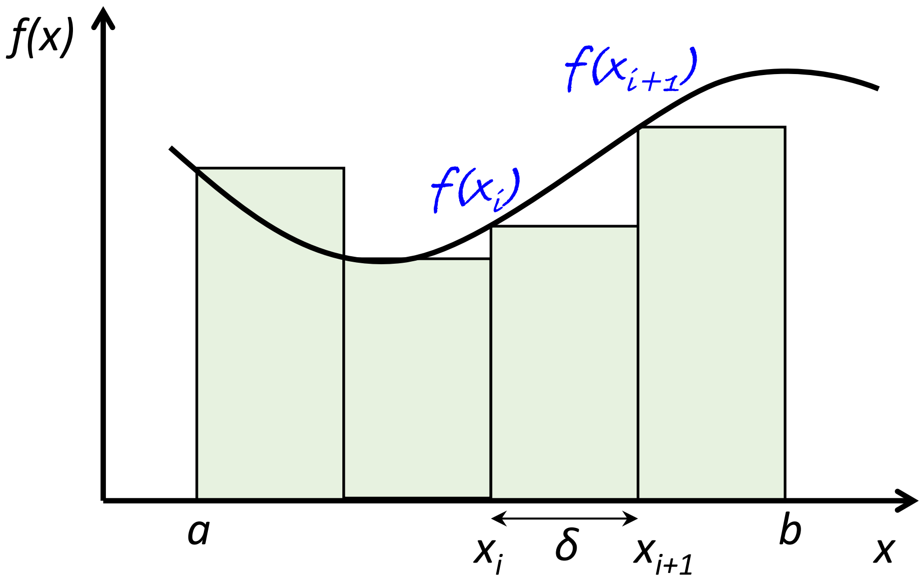
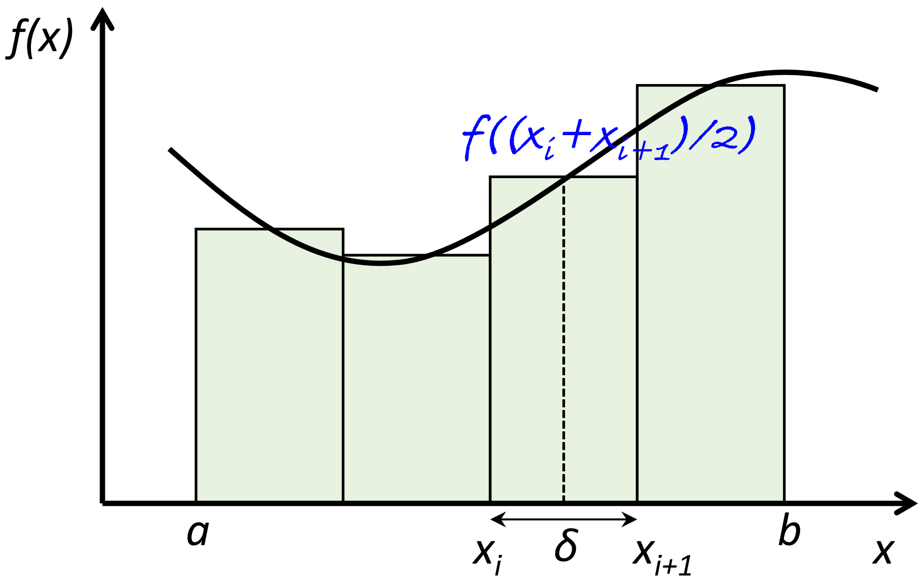


Figure (b)



Numerical integration using rectangles (1)

- Master/slave
- SPMD (Single-Program Multiple-Data)
- Static task assignment

- A better solution

```
11.        area = 0;
       // Tính diện tích trong vùng con tại mỗi P_i
12.        for (x=start; x<end; x=x+δ)
13.1.          area = area + f(x) + f(x+δ);
13.2.      area = 0.5 * area * δ;
```

```
// Tính $\int_a^b f(x)dx$ với chu kỳ lấy mẫu (b-a)/N_0
// P_group là nhóm p tiến trình/bộ xử lý (P_i)
1.  Integration(a, b) {
2.          i = Get_Rank(); // P_i có Rank=i
3.          if (i == 0) { // Master có Rank=0
4.              n = N_0;    // Số mẫu cầu lấy
5.          }
      // p bộ xử lý cùng thực hiện, Master cũng là một Slave
6.          bcast(&n, P_group); // Master truyền, tất cả P_i nhận
7.          δ = (b-a)/n;     // Chu kỳ lấy mẫu
8.          region = (b-a)/p;        // độ dài vùng con
9.          start = a + region * i; // điểm bắt đầu
10.         end = start + region;   // điểm kết thúc
11.         area = 0;               // diện tích vùng con
       // Tính diện tích trong vùng con tại mỗi P_i
12.         for (x=start; x<end; x=x+δ)
13.             area = area + 0.5 * (f(x) + f(x+δ)) * δ;
       // Tính tổng gộp diện tích các vùng con tại tất cả P_i
14.         Reduce_Add(&S, &area, P_group);
15.         Return(S);
16. }
```

Numerical integration using rectangles (2)

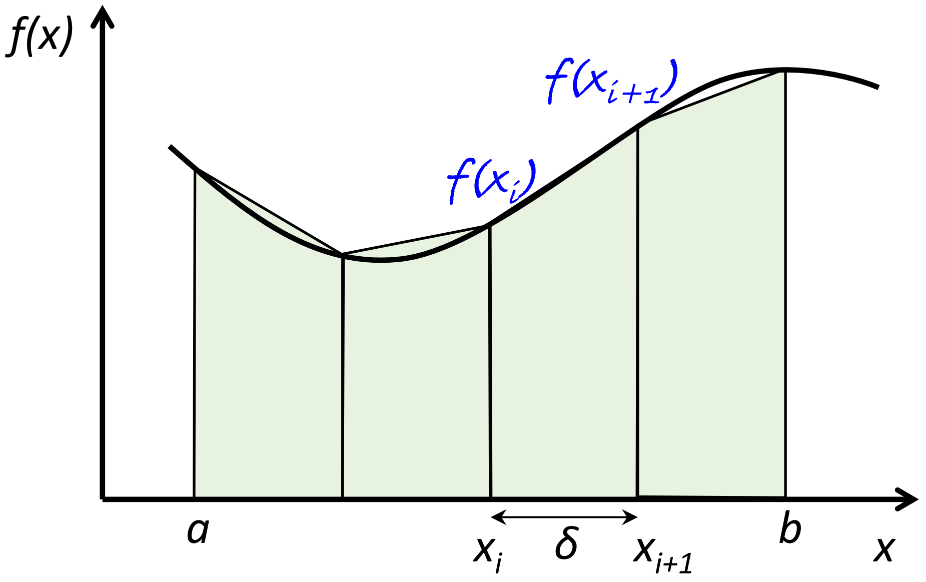
$$area = \frac{\delta(f(a)+f(a+\delta))}{2} + \frac{\delta(f(a+\delta)+f(a+2\delta))}{2} + \ldots + \frac{\delta(f(a+(n-1)\delta)+f(b))}{2}$$

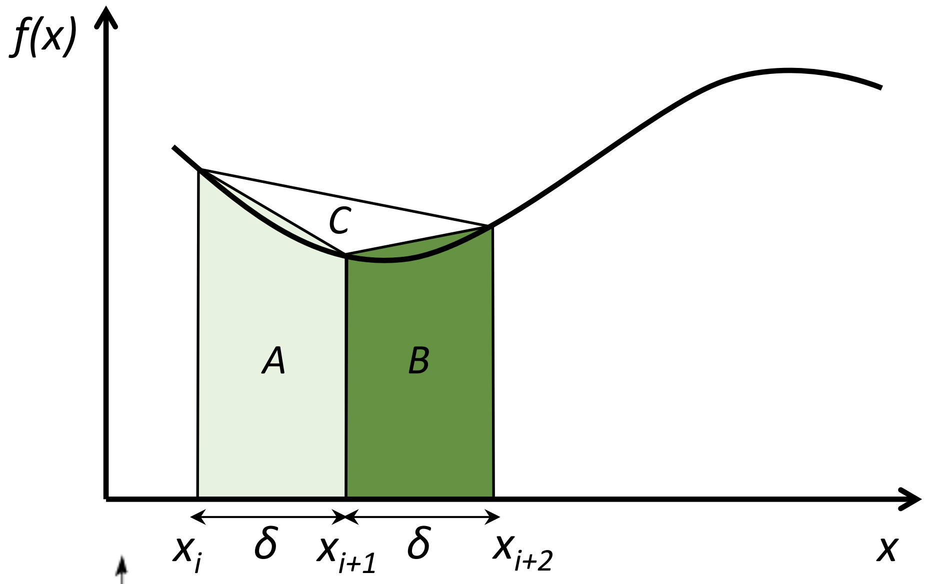
$$\Leftrightarrow area = \delta(\frac{f(a)}{2} + f(a+\delta) + f(a+2\delta) + \ldots + f(a+(n-1)\delta) + \frac{f(b)}{2}).$$

```
11.         area = 0;
        // Tính diện tích trong vùng con tại mỗi Pi
12.         for (x=start; x<end; x=x+δ)
13.1.           area = area + f(x) + f(x+δ);
13.2.       area = 0.5 * area * δ;
```

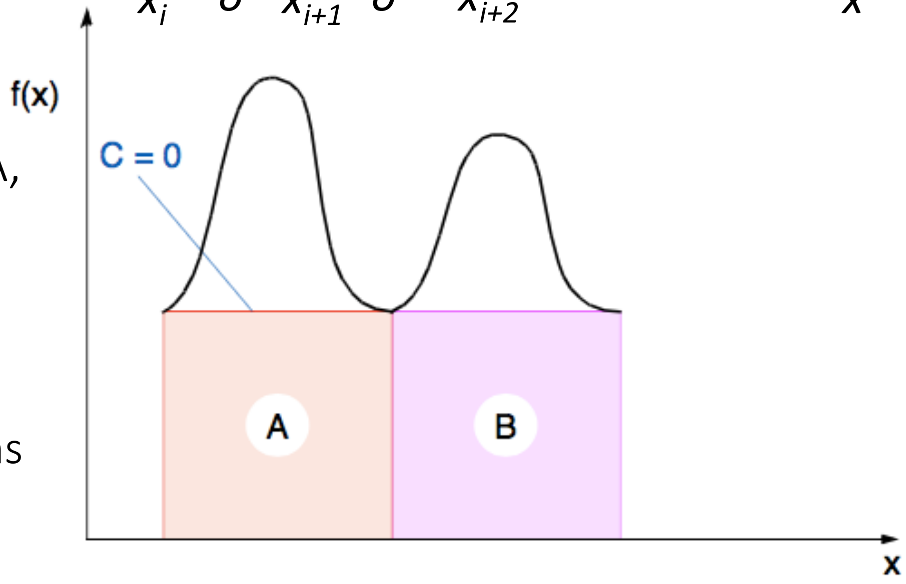
```
11.         area = 0.5 * (f(start) + f(end));
        // Tính diện tích trong vùng con tại mỗi Pi
12.         for (x=start+δ; x<end; x=x+δ)
13.1.           area = area + f(x);
13.2.       area = area * δ;
```

Numerical integration using trapezoidal method





- Adaptive Quadrature
  - Solution adapts to shape of curve. Use three areas, A, B, and C. Computation terminated when largest of A and B sufficiently close to sum of remain two areas
  - Adaptive quadrature with false termination
    - Might cause us to terminate early, as two large regions are the same (i.e., C = 0)



Numerical integration using trapezoidal method

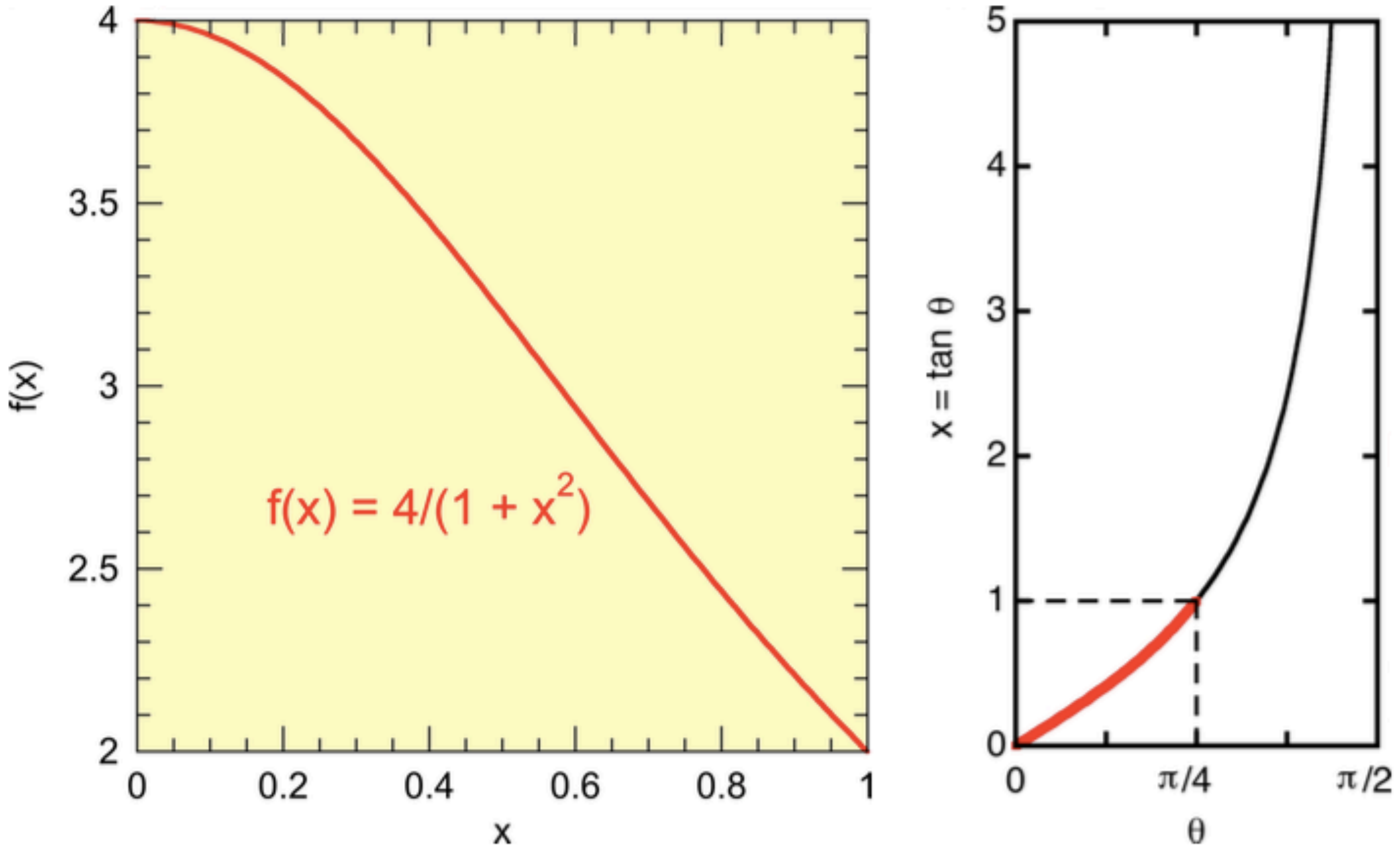
$F(L, \delta)$ with $L=[a, b]$ is an integral of $f(.)$ from a to b with the sampling frequency $1/\delta$.

- Divide
  - Divide $L=[a, b]$ into p parts $L_1, L_2, .., L_p$ with the same length $(b-a)/p$;
  - Concurrent computation $F(L_k, \delta)$ with all k from 1 to p
- Combine
  - Return $\sum Area_k$
- Conquer
  - Input: L_k and δ_k;
  - Compute $F(L_k, \delta_k)$
    - $Area_k$ = total area of sampling trapezoids in L_k;
  - If the accuracy Δ_k is hold, then
    - Return $Area_k$

    else
    - Divide L_k into $L_{k(L)} = [a_k, (b_k-a_k)/2]$ & $L_{k(R)} = [(b_k-a_k)/2 + 1, b_k]$
    - Compute $F(L_{k(L)}, \delta_k/2)$ & $F(L_{k(R)}, \delta_k/2)$

Integral representation of π

$$\int_0^1 dx \frac{4}{1+x^2} = \int_0^{\pi/4} \frac{d\theta}{\cos^2\theta} \frac{4}{1+\tan^2\theta} = \int_0^{\pi/4} 4 d\theta = \pi$$



Integral representation of π

```
/* Calculate value of π and compares with actual value (to 25 digits) of pi to give error.
Integrates function f(x)=4/(1+x^2)*/

#include <math.h>   //include files
#include <iostream.h>
#include "mpi.h"

void printit(); //function prototypes

int main(int argc, char *argv[])
{
    double actual_pi = 3.141592653589793238462643;
    int n, rank, num_proc, i;
    double temp_pi, calc_pi,
    int_size, part_sum, x;
    char response = 'y', resp1 = 'y’;
```

```
MPI::Init(argc, argv); //initiate MPI
num_proc = MPI::COMM_WORLD.Get_size();
rank = MPI::COMM_WORLD.Get_rank();
if (rank == 0)
   printit(); /* I am root node, print out welcome */
   while (response == 'y') {
         if (resp1 == 'y') {
              if (rank == 0) {   /*I am root node*/
              cout <______________________” <<endl;
              cout <<"\nEnter the number of intervals: (0 will exit)" << endl;
              cin >> n;
         }  else n = 0;
   MPI::COMM_WORLD.Bcast(&n, 1, MPI::INT, 0); //broadcast n
   if (n==0) break; //check for quit condition
   else {   int_size = 1.0 / (double) n; //calcs interval size
             part_sum = 0.0;
             for (i = rank + 1; i <= n; i += num_proc) { //calcs partial sums
                  x = int_size * ((double)i - 0.5);
                  part_sum += (4.0 / (1.0 + x*x));
         }
```

```
        temp_pi = int_size * part_sum; //collects all partial sums computes pi
        MPI::COMM_WORLD.Reduce(&temp_pi, &calc_pi, 1, MPI::DOUBLE, MPI::SUM, 0);
        if (rank == 0) { /*I am server*/
           cout << "pi is approximately " << calc_pi
           << ". Error is "
           << fabs(calc_pi - actual_pi)
           << endl
           <<"_______________________" << endl;
        }
     } //end else
     if (rank == 0) { /*I am root node*/
           cout << "\nCompute with new intervals? (y/n)" << endl;
           cin >> resp1;
     }
  } //end while
  MPI::Finalize(); //terminate MPI
  return 0;
} //end main
```

```
//functions
void printit() {
    cout << "\n_______________________" << endl
         << "Welcome to the pi calculator!" << endl
         << "Programmer: K. Spry" << endl
         << "You set the number of divisions \nfor estimating the integral: \n\tf(x)=4/(1+x^2)"
         << endl
         << "_______________________"
         << endl;
}  //end printit
```