Chapter 2

Abstract Machine Models & Multi-threading

Thoai Nam

Faculty of Computer Science and Engineering

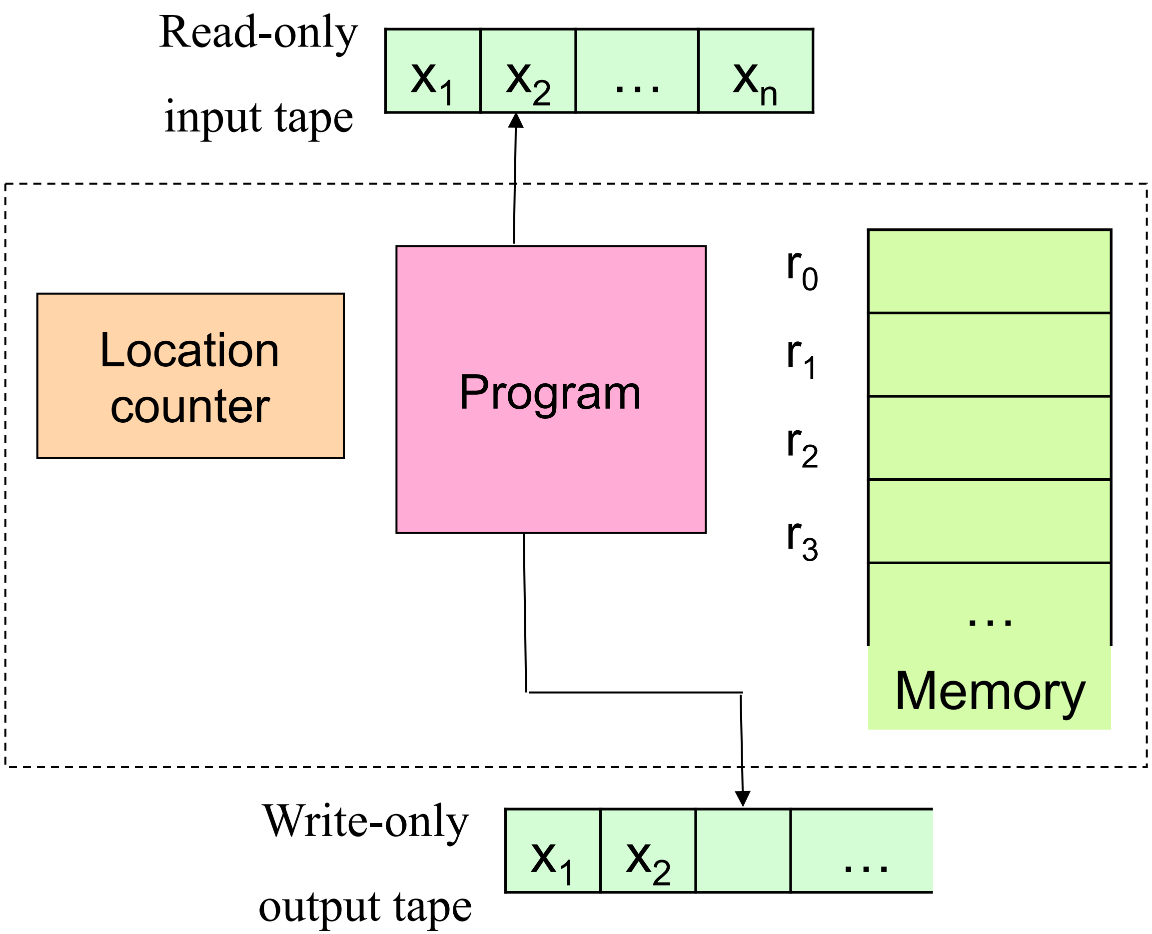
HCMC University of Technology

Abstract Machine Models

- An abstract machine model is mainly used in the design and analysis of parallel algorithms without worry about the details of physics machines.
- Three abstract machine models:
  - PRAM
  - BSP
  - Phase Parallel

RAM (1)

- RAM (Random Access Machine)



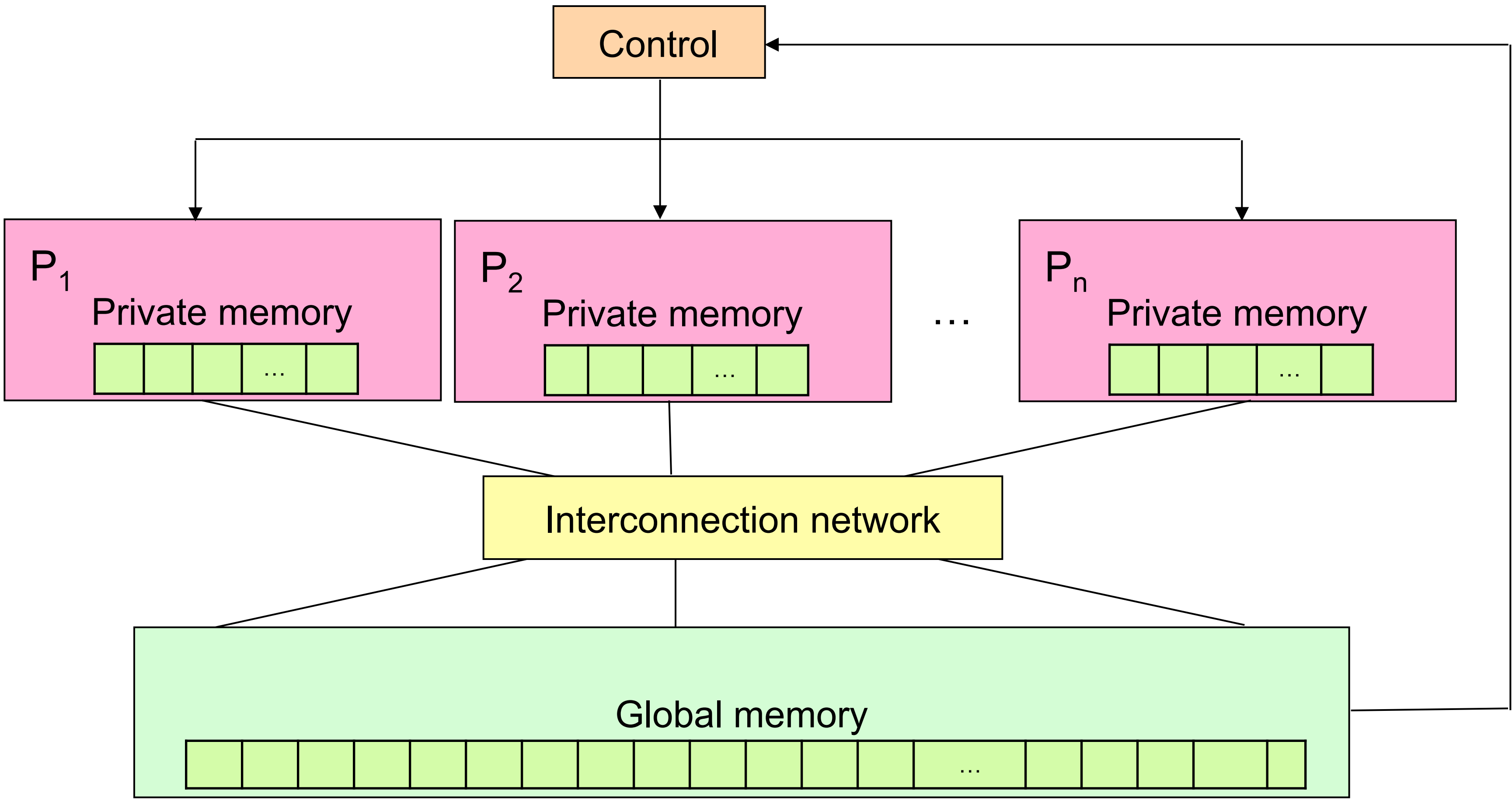
RAM (2)

RAM model of serial computers

- Memory is a sequence of words, each capable of containing an integer
- Each memory access takes one unit of time
- Basic operations (add, multiply, compare) take one unit time
- Instructions are not modifiable
- Read-only input tape, write-only output tape

PRAM (1)

Parallel Random Access Machine (Introduced by Fortune and Wyllie, 1978)



PRAM (2)

- A control unit
- An unbounded set of processors, each with its own private memory and an unique index
- Input stored in global memory or a single *active* processing element
- Step: (1) read a value from a single private/global memory location

  (2) perform a RAM operation

  (3) write into a single private/global memory location
- During a computation step: a processor may activate another processor
- All active, enable processors must execute ***the same instruction*** (albeit on different memory location)???
- Computation terminates when the last processor halts

PRAM (3)

PRAM composed of:

- P processors, each with its own unmodifiable program
- A single shared memory composed of a sequence of words, each capable of containing an arbitrary integer
- a read-only input tape
- a write-only output tape

PRAM model is a synchronous, MIMD, shared address space parallel computer

- Processors share a common clock but may execute different instructions in each cycle

PRAM(4)

- Definition:

  The **cost** of a PRAM computation is the product of the parallel time complexity and the number of processors used.

Ex: a PRAM algorithm that has time complexity O(log ***p***) using ***p*** processors has cost O(***p*** log ***p***)

Time Complexity Problem

- Time complexity of a PRAM algorithm is often expressed in the big-O notation
- Machine size n is usually small in existing parallel computers
- Ex:
  - Three PRAM algorithms A, B and C have time complexities if 7n, (n log n)/4, n log log n.
  - Big-O notation: A(O(n)) < C(O(n log log n)) < B(O(n log n))
  - Machines with no more than 1024 processors:

    log n ≤ log 1024 = 10 and log log n ≤ log log 1024 < 4

    and thus: B < C < A

Conflicts Resolution Schemes (1)

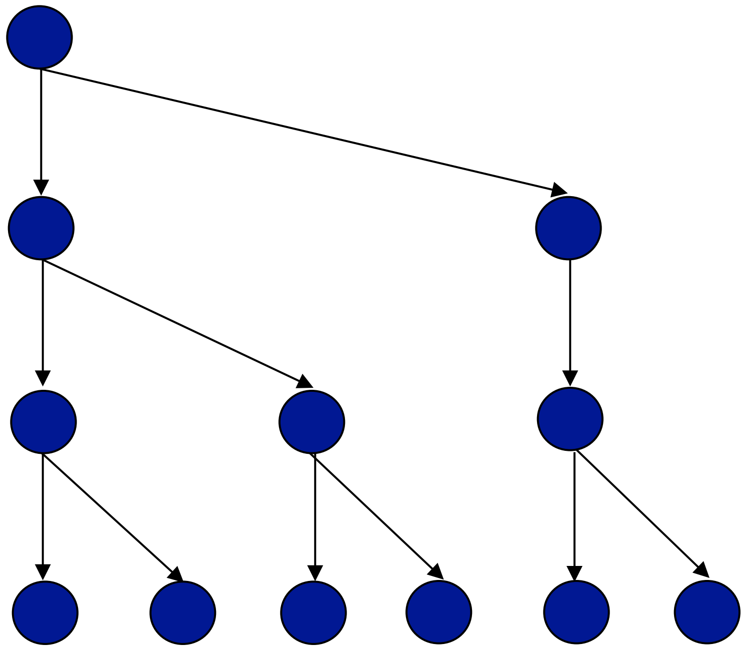
- PRAM execution can result in simultaneous access to the same location in shared memory.
  - Exclusive Read (ER)
    - » No two processors can simultaneously read the same memory location.
  - Exclusive Write (EW)
    - » No two processors can simultaneously write to the same memory location.
  - Concurrent Read (CR)
    - » Processors can simultaneously read the same memory location.
  - Concurrent Write (CW)
    - » Processors can simultaneously write to the same memory location, using some conflict resolution scheme.

Conflicts Resolution Schemes (2)

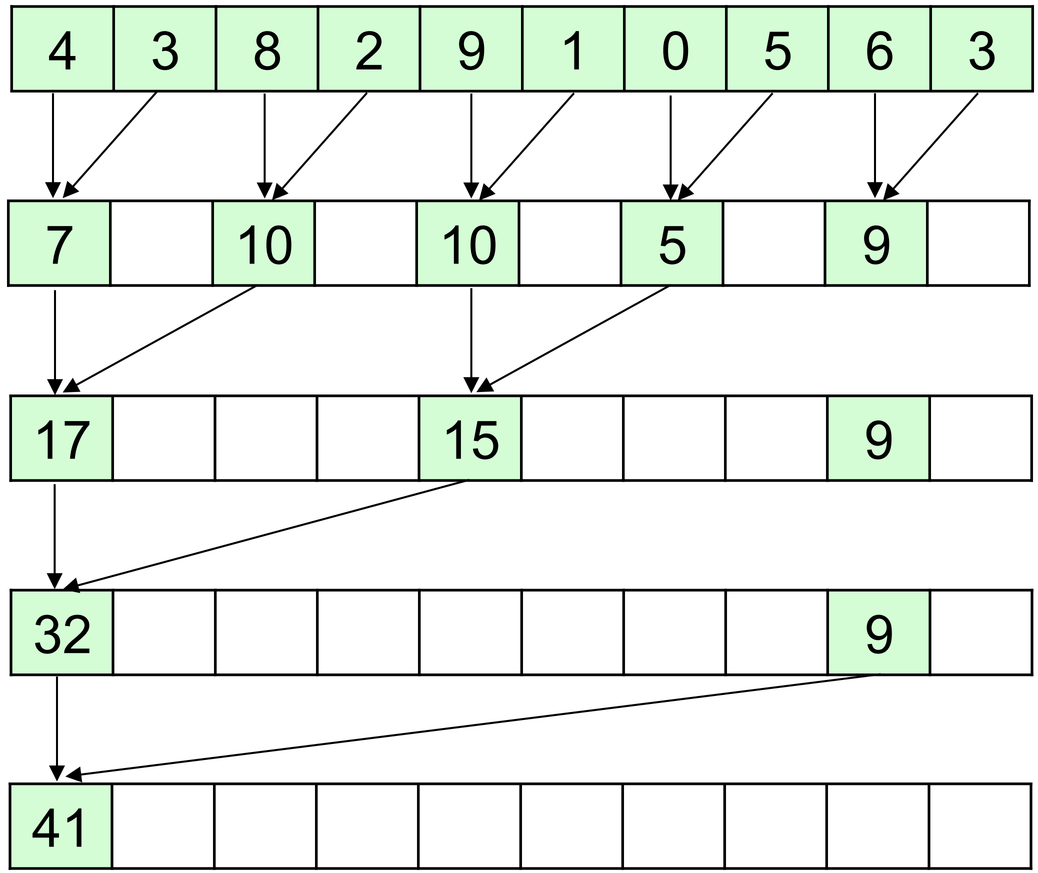
- Common/Identical CRCW
  - All processors writing to the same memory location must be writing the same value.
  - The software must ensure that different values are not attempted to be written.
- Arbitrary CRCW
  - Different values may be written to the same memory location, and an arbitrary one succeeds.
- Priority CRCW
  - An index is associated with the processors and when more than one processor write occurs, the lowest-numbered processor succeeds.
  - The hardware must resolve any conflicts

PRAM Algorithm

- Begin with a single active processor active
- Two phases:
  - A sufficient number of processors are activated
  - These activated processors perform the computation in parallel
- $\lceil \log p \rceil$ activation steps: p processors to become active
- The number of active processors can be double by executing a single instruction



Parallel Reduction (1)



Parallel Reduction (2)

(EREW PRAM Algorithm in Figure2-7, page 32, book [1])

Ex: SUM(EREW)

Initial condition: List of $n \geq 1$ elements stored in $A[0..(n-1)]$

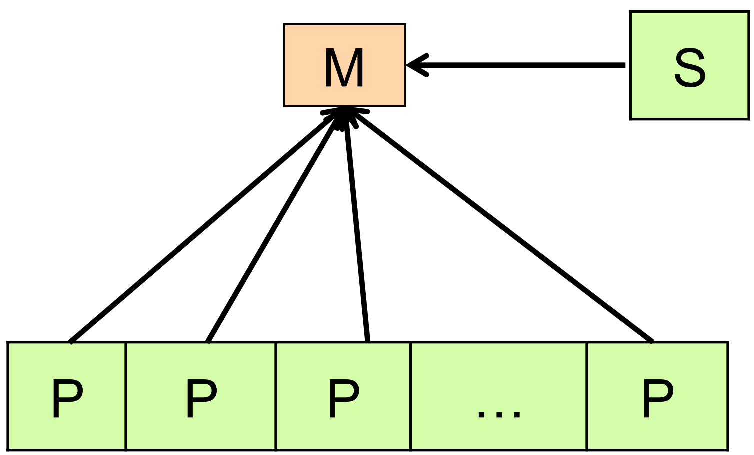
Final condition: Sum of elements stored in $A[0]$

Global variables: n, $A[0..(n-1)]$, j

```
begin
    spawn (P_0, P_1,..., P_{⌊n/2⌋-1})
    for all P_i where 0 ≤ i ≤ ⌊n/2⌋ -1 do
        for j ← 0 to ⌈log n⌉ – 1 do
            if i modulo 2^j = 0 and 2i+2^j < n the
                A[2i] ← A[2i] + A[2i+2^j]
            endif
        endfor
    endfor
end
```

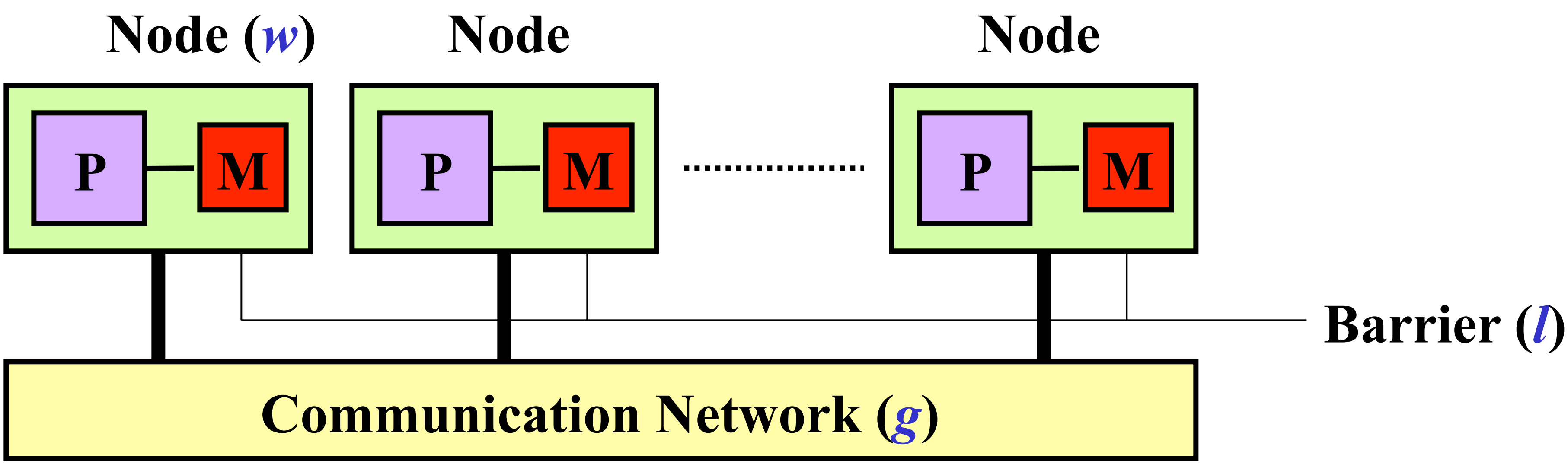
Broadcasting on a PRAM

- "Broadcast" can be done on CREW PRAM in O(1) steps:
  - Broadcaster sends value to shared memory
  - Processors read from shared memory
- Requires log*P* steps on EREW PRAM



BSP – Bulk Synchronous Parallel

- BSP Model
  - Proposed by Leslie Valiant of Harvard University
  - Developed by W.F.McColl of Oxford University



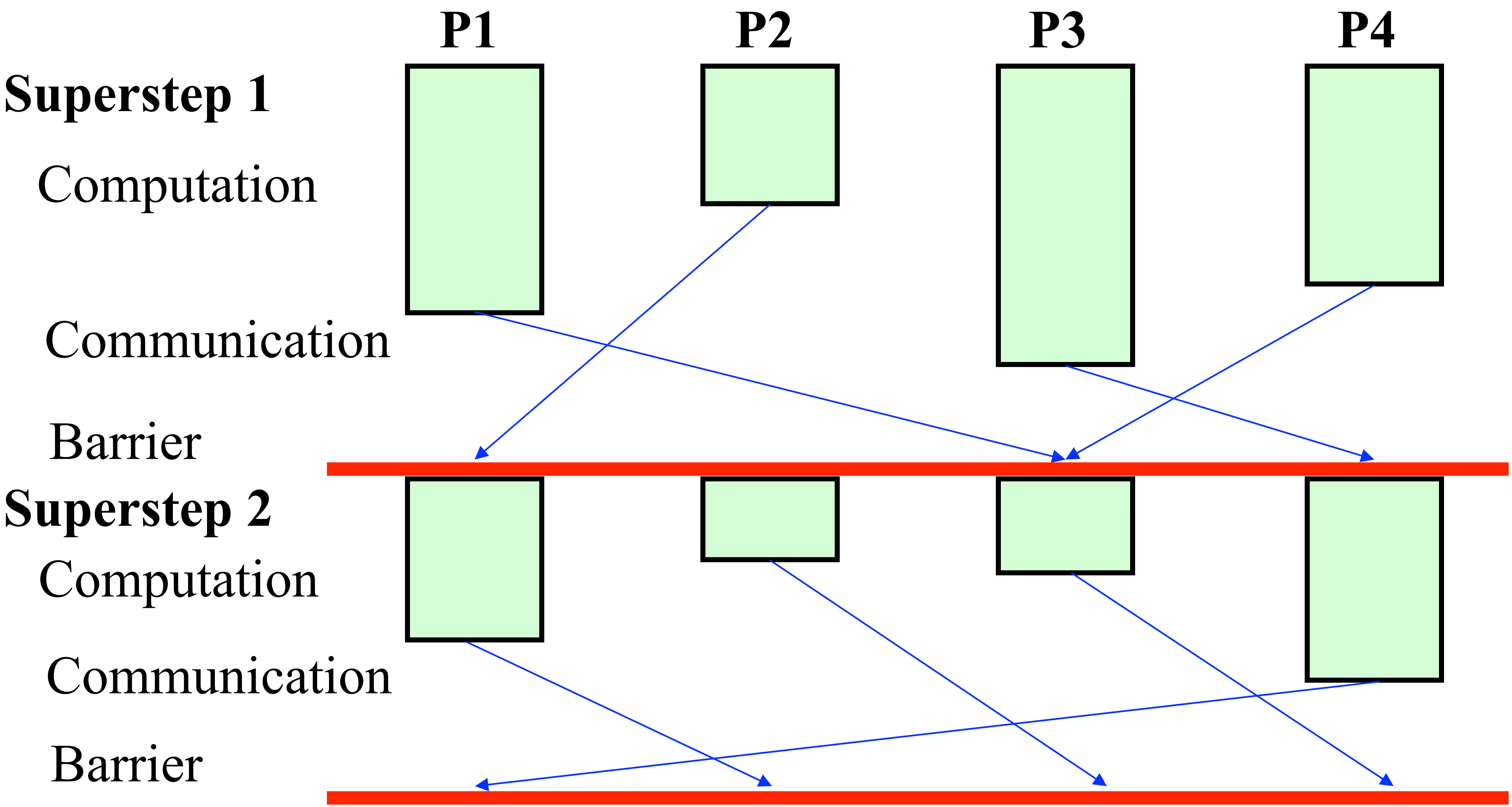
BSP Model

- A set of n nodes (processor/memory pairs)
- Communication Network
  - Point-to-point, message passing (or shared variable)
- Barrier synchronizing facility
  - All or subset
- Distributed memory architecture

BSP Programs

- A BSP program:
  - *n* processes, each residing on a node
  - Executing a strict sequence of *supersteps*
  - In each superstep, a process executes:
    - » Computation operations: *w* cycles
    - » Communication: *gh* cycles
    - » Barrier synchronization: *l* cycles

A Figure of BSP Programs



Three Parameters

- The basic time unit is a cycle (or time step)
- ***w*** parameter
  - Maximum computation time within each superstep
  - Computation operation takes at most ***w*** cycles.
- ***g*** parameter
  - Number of cycles for communication of unit message when all processors are involved in communication - network bandwidth
  - (total number of local operations performed by all processors in one second) / (total number of words delivered by the communication network in one second)
  - ***h*** relation coefficient
  - Communication operation takes ***gh*** cycles.
- ***l*** parameter
  - Barrier synchronization takes ***l*** cycles.

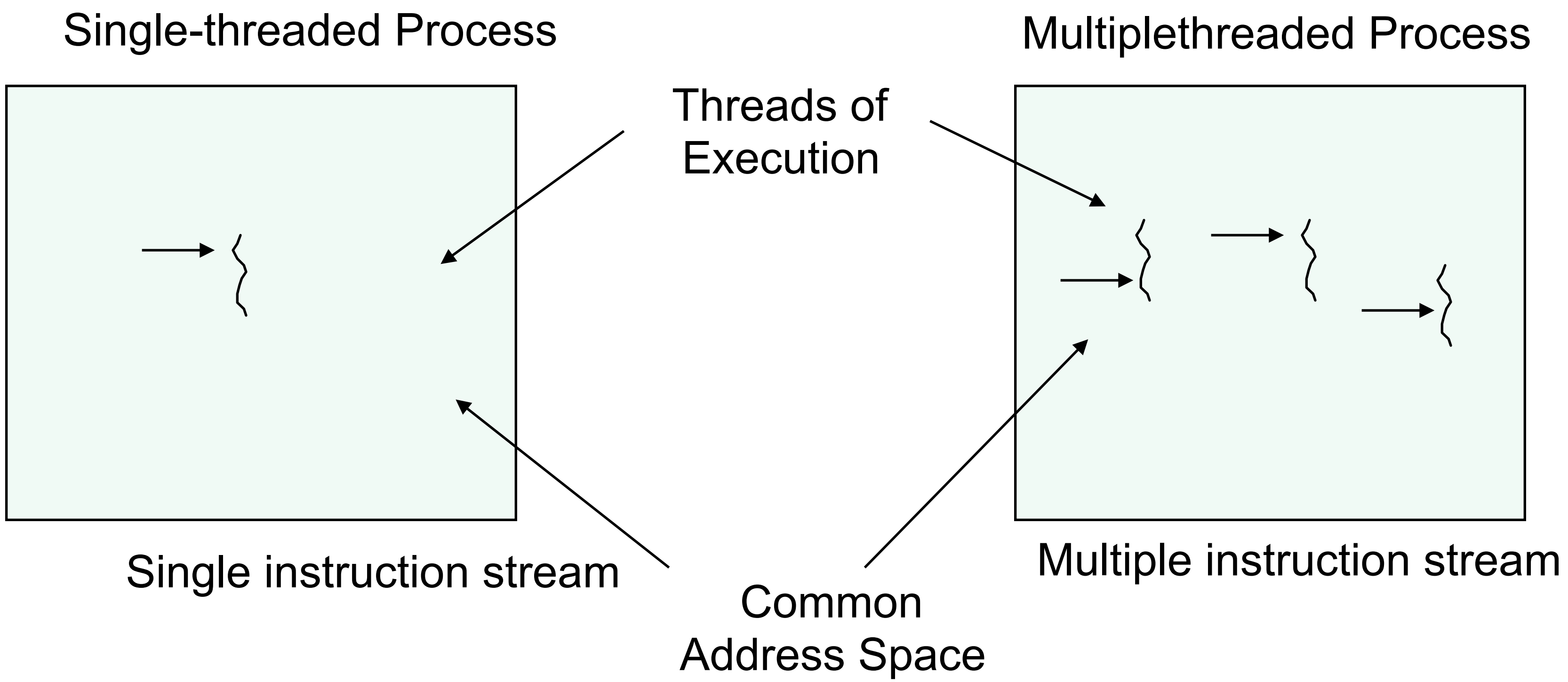
Time Complexity of BSP Algorithms

- Execution time of a superstep:
  - Sequence of the computation, the communication, and the synchronization operations: $w + gh + l$
  - Overlapping the computation, the communication, and the synchronization operations: $\max\{w, gh, l\}$

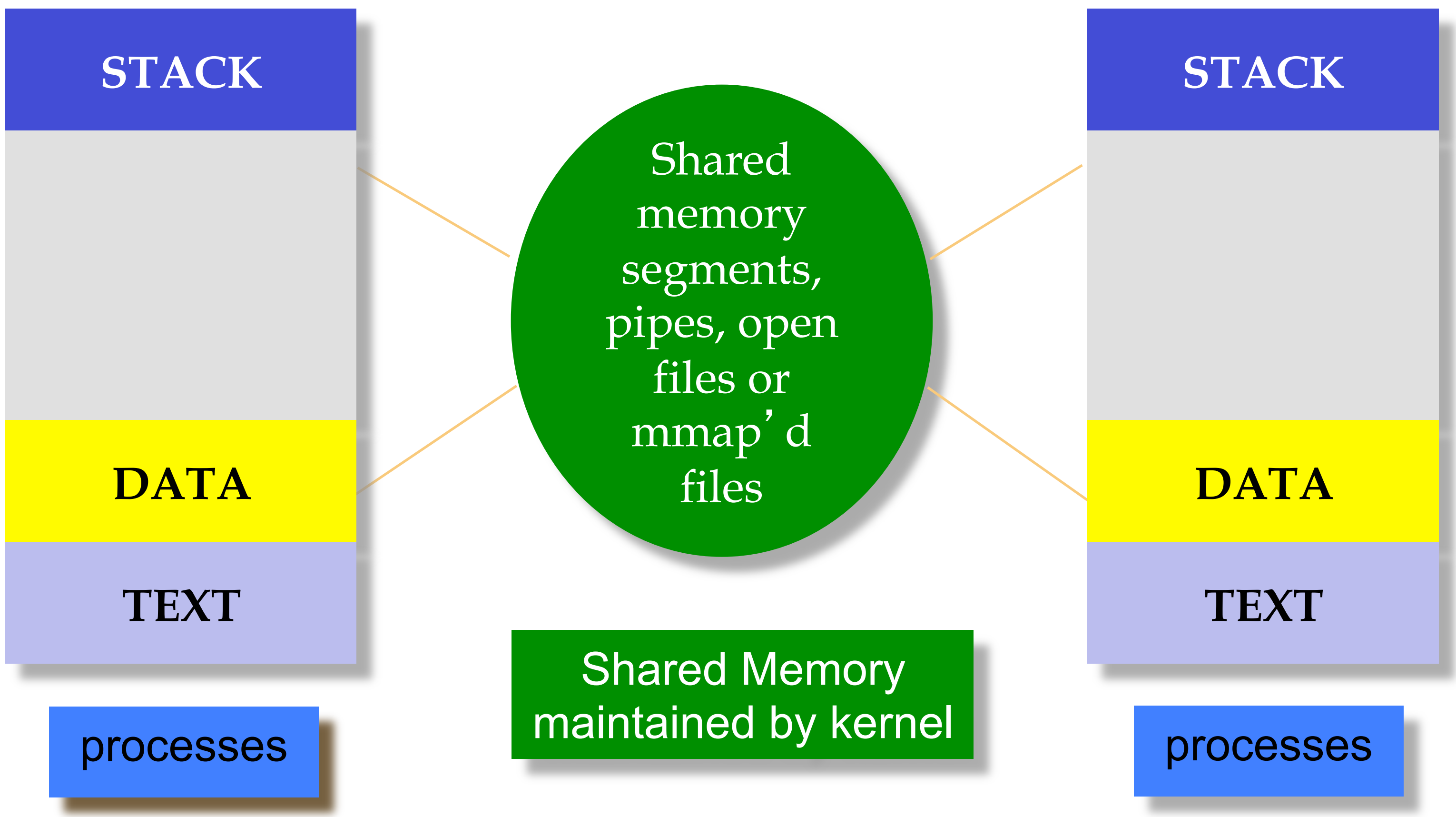
Phase Parallel

- Proposed by Kai Hwang & Zhiwei Xu
- Similar to the BSP:
  - A parallel program: sequence of phases
  - Next phase cannot begin until all operations in the current phase have finished
  - Three types of phases:
    - Parallelism phase: the overhead work involved in process management, such as process creation and grouping for parallel processing
    - Computation phase: local computation (data are available)
    - Interaction phase: communication, synchronization or aggregation (e.g., reduction and scan)
- Different computation phases may execute different workloads at different speed.

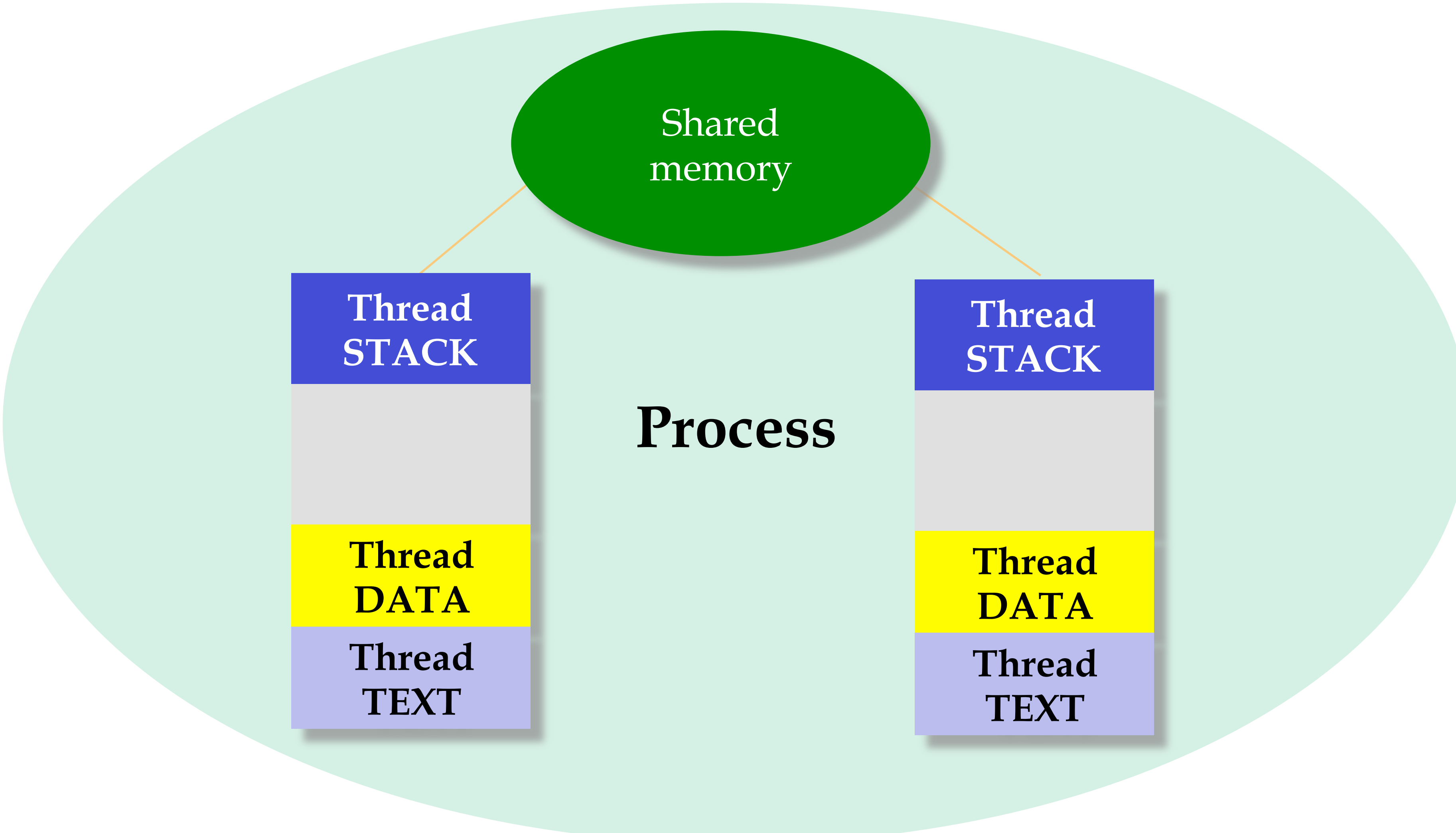
Process: single & multithreaded



Process Model



Threaded Process Model

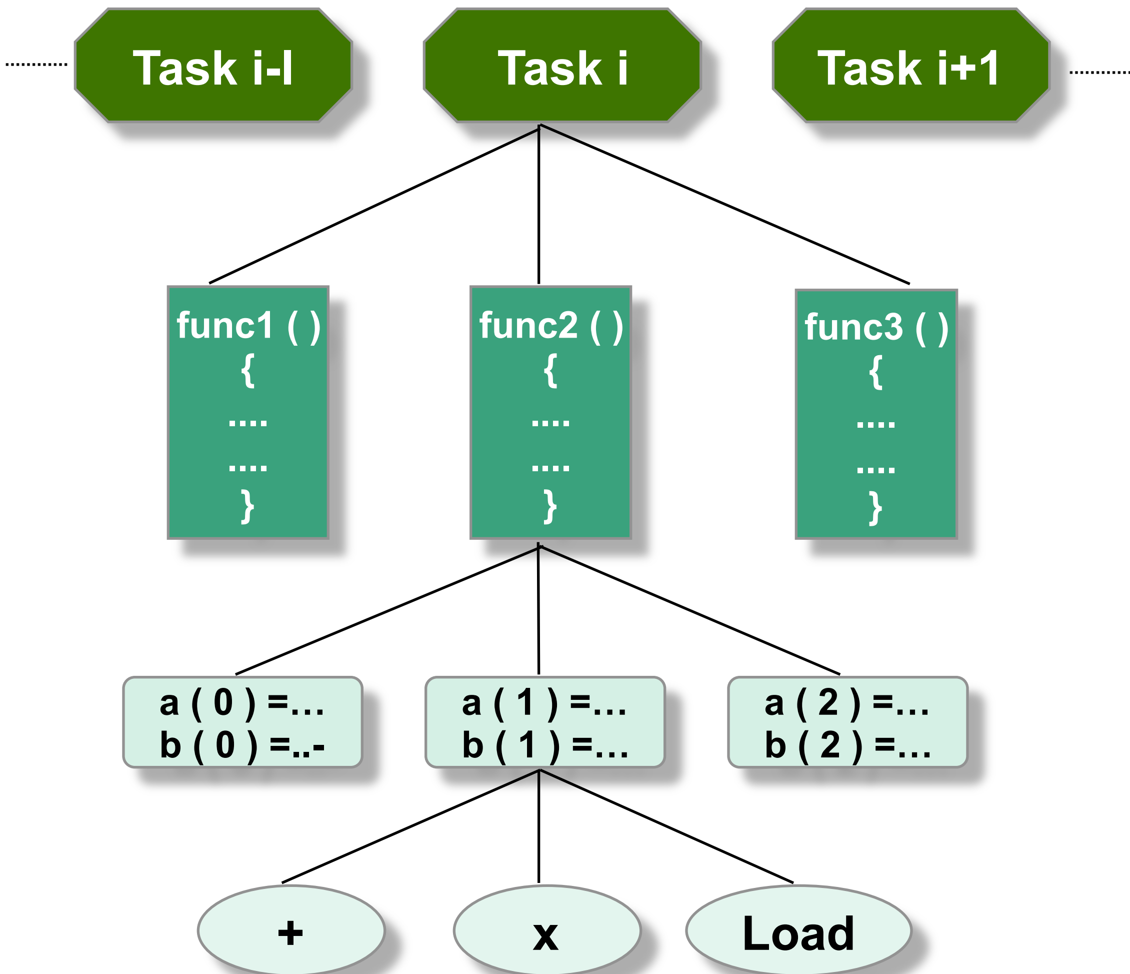


What are Threads

- Thread is a piece of code that can execute in concurrence with other threads
- It is a schedule entity on a processor

  - **Local state**
  - **Global/shared state**
  - **PC**
  - **Hard/Software context**

Levels of Parallelism



Code-Granularity
Code Item

Large grain (task level)
Program

Medium grain (control level)
Function (thread)

Fine grain (data level)
Loop

Very fine grain (multiple issue)
With hardware

Thread Example

```
void *func ( )
{
    /* define local data */
    - - - - - - - - - - -
    - - - - - - - - - - -   /* function code */
    - - - - - - - - - - -
    thr_exit(exit_value);
}

main ( )
{
    thread_t tid;
    int exit_value;
    - - - - - - - - - - -
    thread_create (0, 0, func (), NULL, &tid);
    - - - - - - - - - - -
    thread_join (tid, 0, &exit_value);
    - - - - - - - - - - -
}
```

Ref

– BSP: http://www.computingreviews.com/hottopic/hottopic_essay.cfm?htname=BSP

Bubble Sort

```
procedure BubbleSort( A : list of sortable items )
   n = length(A)
   repeat
      newn = 0
      for i = 1 to n-1 inclusive do
         if A[i-1] > A[i] then
            swap(A[i-1], A[i])
            newn = i
         end if
      end for
      n = newn
   until n = 0
end procedure
```

Quick Sort

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p – 1)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo     // place for swapping
    for j := lo to hi – 1 do
        if A[j] ≤ pivot then
            swap A[i] with A[j]
            i := i + 1
    swap A[i] with A[hi]
    return i
```