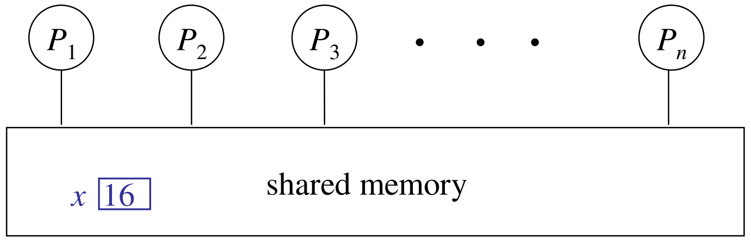
# Shared-Memory Computers, Basic Techniques, and Brent's Theorem

## ■ Shared-Memory Computers



1. *EREW* (exclusive read, exclusive write): Simultaneous access to the same memory location is not allowed.

2. *CREW* (concurrent read, exclusive write): Simultaneous read to the same memory location is allowed, but simultaneous write is not allowed.

3. *ERCW* (exclusive read, concurrent write): Simultaneous read is not allowed, but simultaneous write is allowed.

4. *CRCW* (concurrent read, concurrent write): Both simultaneous read and simultaneous write are allowed.

* Shared-memory computer is of great theoretical interest, but current technology prohibits its realization.

* The shared-memory computer can be regarded as a completely connected interconnection network.

* Communication between any two processors takes $O(1)$ time through the shared memory.

* The SIMD shared-memory computer is also called *Parallel Random Access Machine* (*PRAM*)

* **Resolution rules for write conflicts** (L. Kucera, "Parallel computation and conflicts in memory access," *Information Processing Letters*, vol. 14, no. 2, pp. 93-96, 1982.)

1. Weak conflict-resolution rule (also known as W-PRAM):
   (i) Simultaneous writing is allowed only to selected memory locations which can contain the numbers 0 or 1 only.
   (ii) Processors write simultaneously into the same location must write the value 1 and the final value remaining is 1.

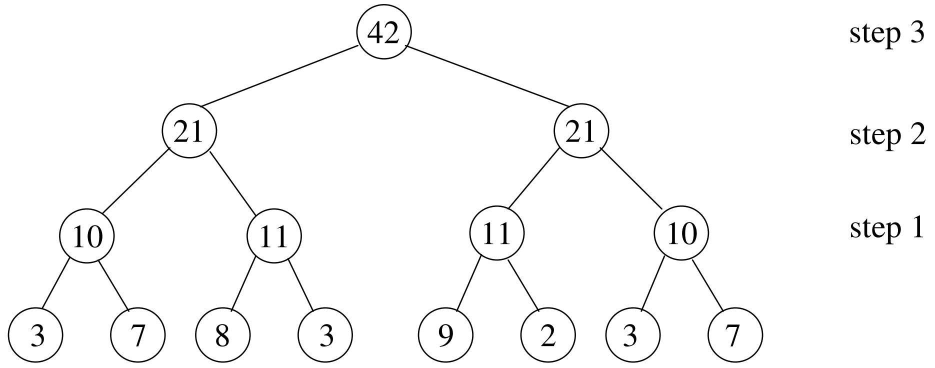
2. Equality conflict-resolution rule.

3. Priority conflict-resolution rule (fixed or dynamically changeable).

* How to find the maximum in $O(1)$ time???

## ■ Basic Techniques

**Problem SM-1**: Summing
**Input**: $A[1\ldots n] = \{3, 7, 8, 3, 9, 2, 3, 7\}$ ($n = 8$)
**Output**: $A[1] + A[2] + \ldots + A[n]$
**Model**: EREW PRAM of $n/2$ processors ($n = 2^m$)



```
for $k = m - 1$ step $-1$ to 0 do
   for $i$, $1 \le i \le 2^k$, pardo $A[i] = A[2i-1] + A[2i]$
```

* $T(n) = O(\log n)$
* *Balanced binary tree method*: Build a balanced binary tree on the input (or output) elements and traverse the tree forward and backward to and from the root.

Simple Applications: (1) Computing maximum
(2) Broadcasting
(3) OR, AND, XOR

* Prefix sums (two phases) (tree machine)

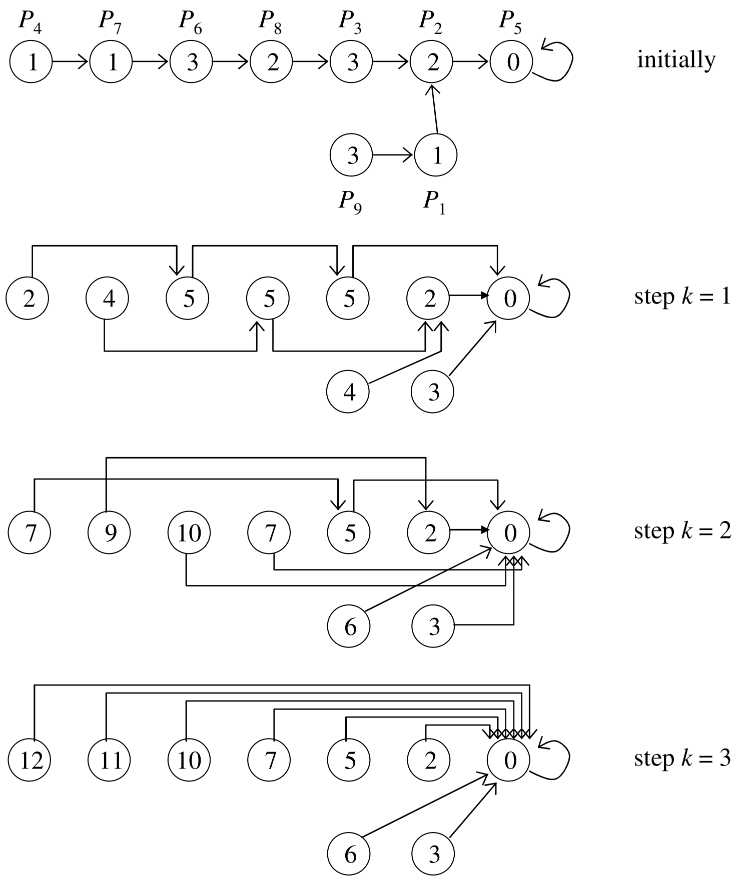
**Problem SM-2**: Parallel prefix on a rooted directed tree
**Input**: $P[1\ldots n] = \{2, 5, 2, 7, 5, 8, 6, 3, 1\}$ and
$W[1\ldots n] = \{1, 2, 3, 1, 0, 3, 1, 2, 3\}$.
(A rooted directed tree of $n$ nodes; $P[i]$ and $W[i]$, $1 \le i \le n$, is the parent and weight of node $i$, respectively. The tree is self-loop at its root and the weight of the root is 0.)

**Output**: For each node $i$, $W[i]$ is set equal to the sum of the weights of nodes on the path from $i$ to the root of its tree.
**Model**: CREW PRAM of $n$ processors

```
for $i$, $1 \le i \le n$, pardo $S[i] = P[i]$
for $k = 1$ to $\log n$ do
     for $i$, $1 \le i \le n$, pardo
     begin
          $W[i] = W[i] + W[S[i]]$
          $S[i] = S[S[i]]$
     end
```

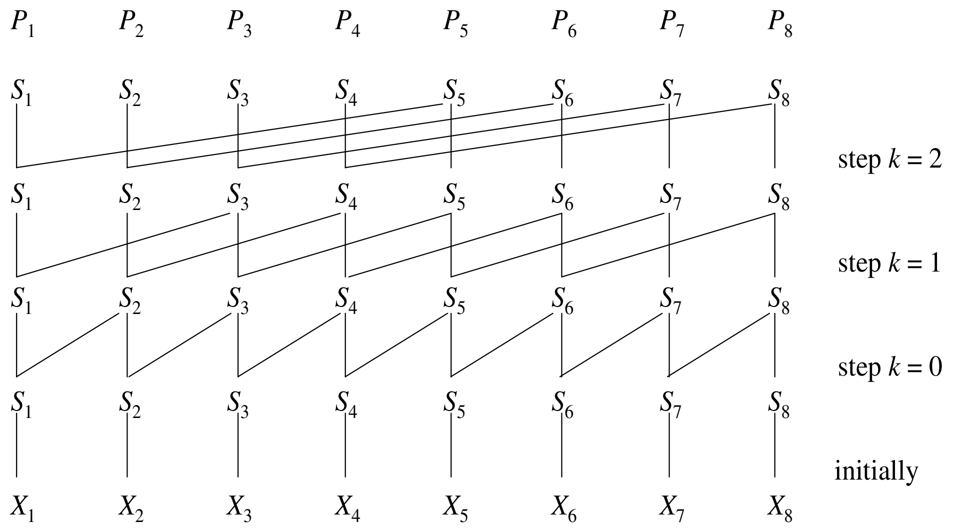
* Correctness???

* $T(n) = O(\log n)$
* *Pointer Jumping* (*Doubling*): The computation proceeds by a recursive application of the calculation in hand to all elements over a certain distance (in the data structure) from each individual element. This distance doubles in successive steps.



Simple Applications:
(1) Parallel prefix (computing prefix sums on linked list)
(2) List ranking (EREW, $n/\log n$ processors, $O(\log n)$ time)
(3) Parallel prefix on a rooted directed forest
(4) Finding the root for each node of a given forest
(5) Determining the length of the path from each node (of a given forest) to its root.

**Problem SM-3**: Prefix sums (prefix computation)
**Input**: $X[1\ldots n]$
**Output**: $S[1\ldots n]$, where $S[i] = X[1] + X[2] + \ldots + X[i]$
**Model**: EREW PRAM of $n$ processors



```
for $i$, $1 \le i \le n$, pardo $S[i] = X[i]$
for $k = 0$ to $\log n - 1$ do
    for $i$, $2^k+1 \le i \le n$, pardo $S[i] = S[i] + S[i - 2^k]$
```

* $T(n) = O(\log n)$
* *Doubling*

**Problem SM-4**: Parallel *m*-way search
**Input**: a sorted sequence $A[1, n] = \{1, 2, \ldots, 18\}$, and a value $x = 11$
**Output**: $k$ such that $A[k] = x$ (Assume that $k$ uniquely exists.)
**Model**: CREW PRAM of $p$ processors

step 0: $beginning = 0$; $length = n$;

step 1: If ($length \le p$)
each processor $P_i$, $1 \le i \le length$, sets $k = beginning + i$ if $A[beginning + i] = x$; and then, terminates.

step 2: Each processor $P_i$, $1 \le i \le n$, sets $M[i]$ as 1 if $x \le A[beginning + i \times (length/p)]$ and 0 otherwise.

step 3: Each processor $P_i$, $1 \le i \le p$, sets $M[i] = M[i] - M[i-1]$. (Assume $M[0]=0$.) And then, if $M[i] = 1$, sets $beginning = beginning + (i-1) \times (length/p)$ and $length = length/p$.

step 4: Repeat 1~3 until $k$ is found.

**Example:** $x = 11$, $n=18$, $p=3$

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $P_1$ | | | | | | $P_2$ | | | | | | $P_3$ |
| $A$: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| $M$: | | | | | | 0 | | | | | | 1 | | | | | | 1 |

stage 1

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $P_1$ | | $P_2$ | | $P_3$ | | | | | | |
| $A$: | * | * | * | * | * | * | 7 | 8 | 9 | 10 | 11 | 12 | * | * | * | * | * | * |
| $M$: | | | | | | | | 0 | | 0 | | 1 | | | | | | |

stage 2

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | $P_1$ | $P_2$ | | | | | | |
| $A$: | * | * | * | * | * | * | * | * | * | * | $\underline{11}$ | 12 | * | * | * | * | * | * |

$k = 11$

stage 3

* A simple extension of binary search

* $T(n) = T(n/p) + O(1) = O(\log_p n)$

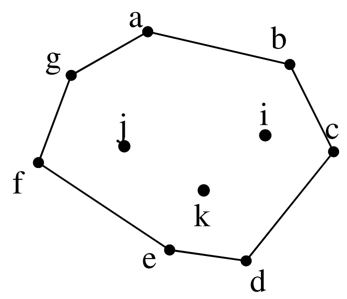
* While $p = n^{1/k}$ for some constant $k$, the proposed algorithm requires $O(1)$ time. We will have an example of $k = 2$ in **Problem** SM-5 (convex hull).

**Problem SM-5**: Convex hull

**Input**: a set $S = \{v_1, v_2, \ldots, v_n\}$ of $n$ points

**Output**: The convex hull $CH(S)$ of $S$ (clockwise)

**Model**: CREW PRAM of $n$ processors



$S = \{f, g, j, a, e, k, d, i, b, c\}$

$CH(S) = \{f, g, a, b, c, d, e\}$

$UH(S) = \{f, g, a, b, c\}$

$LH(S) = \{c, d, e, f\}$

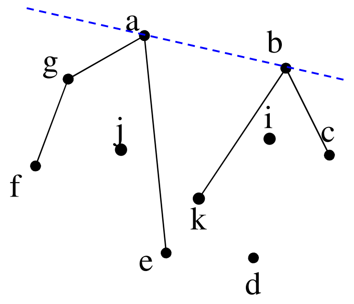
(1) Sort the points in $S$ by their $x$ coordinates. After the sorting, we have $x(v_1) < x(v_2) < \ldots < x(v_n)$.

**Remark:** Sorting $n$ numbers can be done in $O(\log n)$ time on the EREW PRAM of $n$ processors. (R. Cole, "Parallel merge sort," *SIAM Journal on Computing*, vol. 17, no. 4, pp. 770-785, 1988.)

(2) Let $p$ and $q$ be the points of the smallest and the largest $x$ coordinates, respectively. Clearly, $p$ and $q$ belong to $CH(S)$ and partition $CH(S)$ into an upper hull $UH(S)$ and a lower hull $LH(S)$. In the following, we concentrate on determining $UH(S)$.

For simplicity, assume that no two points have the same $x$ or $y$ coordinates and $n=2^k$.

(3) **Algorithm** $UH(S)$



$S_1 = \{f, g, j, a, e\}$
$S_2 = \{k, d, i, b, c\}$

$UH(S_1) = \{f, g, a, e\}$
$UH(S_2) = \{k, b, c\}$
upper common tangent: (a, b)

$UH(S) = \{f, g, a, b, c\}$

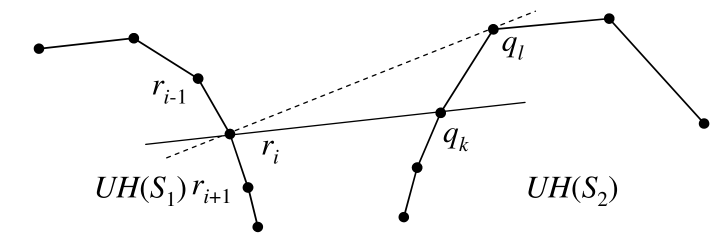
**step 1**: If $|S| \leq 4$, then use a brute-force method to determine $UH(S)$, and **exit**.

**step 2**: Let $S_1 = \{v_1, v_2, \ldots, v_{n/2}\}$ and $S_2 = \{v_{n/2+1}, v_{n/2+2}, \ldots, v_n\}$. Recursively, compute $UH(S_1)$ and $UH(S_2)$ in parallel.

**step 3**: Find the *upper common tangent* between $UH(S_1)$ and $UH(S_2)$, and deduce $UH(S)$.

(4) Let $UH(S_1) = \{r_1, r_2, \ldots, r_s\}$ and $UH(S_2) = \{q_1, q_2, \ldots, q_t\}$.

**Remark:** Given a point $r_i$ and a point $q_k$, we can determine in $O(1)$ sequential time whether $l < k$, $l = k$, or $l > k$, where $r_iq_l$ be the tangent from a point $r_i$ to $UH(S_2)$.



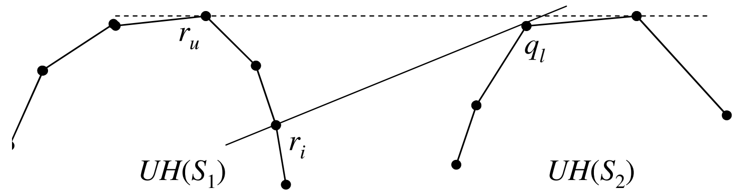
In this example, we can conclude that $l > k$.

**Remark:** In $O(1)$ time, the tangent from a point $r_i$ to $UH(S_2)$ can be obtained on the CREW PRAM of $t^{1/2}$ processors.

step 1: From the points $q_{t^{1/2}}, q_{2\times t^{1/2}}, \ldots, q_{t^{1/2}\times t^{1/2}}$, determine the index $m$ that satisfies $m\times t^{1/2} < l \le (m+1)\times t^{1/2}$.

step 2: From the points $q_{m\times t^{1/2}+1}, q_{m\times t^{1/2}+2}, \ldots, q_{m\times t^{1/2}+t^{1/2}}$, determine the index $l$.

**Remark:** Let $(r_u q_v)$ be the upper tangent of $UH(S_1)$ and $UH(S_2)$. Given a point $r_i$ and $r_i q_l$ (the tangent from $r_i$ to $UH(S_2)$). We can determine in $O(1)$ sequential time whether $u<i$, $u=i$, or $u>i$.



In this example, we can conclude that $u < i$.

**Remark:** The upper tangent of $UH(S_1)$ and $UH(S_2)$ can be determined in $O(1)$ time on a CREW PRAM of $s+t$ processors.

step 1: For each point $r_{i\times s^{1/2}}$, determine the tangent to $UH(S_2)$. ($s^{1/2}\times t^{1/2}$ processors) Then, determine the index $m$ that satisfies $m\times s^{1/2} < u \le (m+1)\times s^{1/2}$.

step 2: From the points $r_{m\times s^{1/2}+1}, r_{m\times s^{1/2}+2}, \ldots, r_{m\times s^{1/2}+s^{1/2}}$, determine the index $u$. (Also, index $v$)

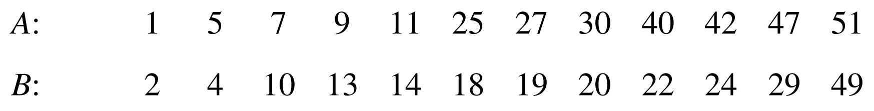
(5) * $T(n)=O(1)+T(n/2)=O(\log n)$

* *Divide and Conquer*: [1] partition the input into the same subproblems of almost equal sizes, [2] solve recursively the subproblems in parallel, and [3] combine the solutions of subproblems.

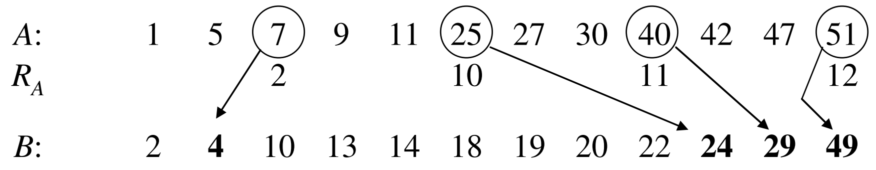
**Problem SM-6**: Merging
**Input**: two sorted sequences $A[1\ldots n]$ and $B[1\ldots n]$
**Output**: a sorted sequence $C[1\ldots 2n]$
**Model**: CREW PRAM of $n/\log n$ processors ($n = 2^m$)

**Remark:** This problem can be solved in $O(n)$ sequential time.

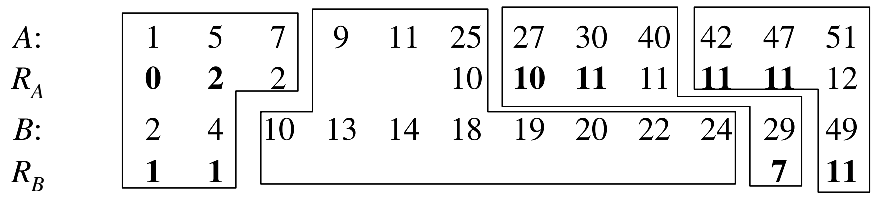
Let $R_A[i]$ be the rank of $A[i]$ in $B$ (i.e., the number of values $\le A[i]$ in $B$.) Define $R_B[i]$ similarly. The merge problem can be viewed as that of determining $R_A[1\ldots n]$ and $R_B[1\ldots n]$. For simplicity, assume that all elements in $A$ and $B$ are distinct.


A: 1 5 7 9 11 25 27 30 40 42 47 51
B: 2 4 10 13 14 18 19 20 22 24 29 49

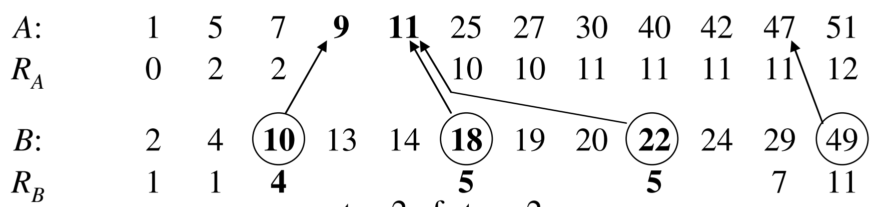
initially


A: 1 5 7 9 11 25 27 30 40 42 47 51
R_A 2 10 11 12
B: 2 4 10 13 14 18 19 20 22 24 29 49

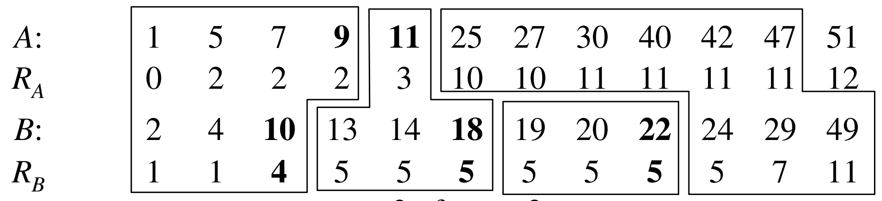
stage 1


A: 1 5 7 9 11 25 27 30 40 42 47 51
R_A 0 2 2 10 10 11 11 11 11 12
B: 2 4 10 13 14 18 19 20 22 24 29 49
R_B 1 1 7 11

step 1 of stage 2


A: 1 5 7 9 11 25 27 30 40 42 47 51
R_A 0 2 2 10 10 11 11 11 11 12
B: 2 4 10 13 14 18 19 20 22 24 29 49
R_B 1 1 4 5 5 7 11

step 2 of stage 2


A: 1 5 7 9 11 25 27 30 40 42 47 51
R_A 0 2 2 2 3 10 10 11 11 11 11 12
B: 2 4 10 13 14 18 19 20 22 24 29 49
R_B 1 1 4 5 5 5 5 5 5 5 7 11

step 3 of stage 2

**stage 1: (partition)** Determine $R_A[i \times m]$, $1 \le i \le n/m$. Then, partition $A$ and $B$ into $A_i$'s and $B_i$'s respectively, where $A_i = \{A[(i-1)\times m+1], \ldots, A[i\times m]\}$ and $B_i = \{B[R_A[(i-1)\times m]+1], \ldots, B[R_A[i\times m]]\}$.

**stage 2:** For each pair of $A_i$ and $B_i$, $1 \le i \le n/m$, if $|B_i|=O(\log n)$, then rank all elements in $A_i$ and $B_i$ sequentially. Otherwise, apply the partition technique used in **stage 1** to partition $A_i$ and $B_i$ into $A_{i,j}$'s and $B_{i,j}$'s of size $O(\log n)$ (in this case, $B_i$ plays the role of $A$ and $A_i$ plays the role of $B$). And then, for each pair of $A_{i,j}$ and $B_{i,j}$, rank all elements in $A_{i,j}$ and $B_{i,j}$ sequentially.

* $T(n)=O(\log n)$

* *Partition*:(1) break up the given problem into $p$ independent subproblems of almost equal size, and then (2) solve the $p$ subproblems concurrently, where $p$ is the number of processors available.

**Remark:** The main work in the divide-and-conquer strategy usually lies in the merging of sub-solutions, whereas the main work in the partition strategy lies in carefully partitioning the problem into independent sub-problems.

**Problem SM-7**: Inserting a sorted sequence into a 2-3 tree
**Input**: a 2-3 tree $T$ holding $n$ items $a_1 < a_2 < \ldots < a_n$, and a sequence $b_1 < b_2 < \ldots < b_k$ (Assume $k$ is much smaller that $n$.)

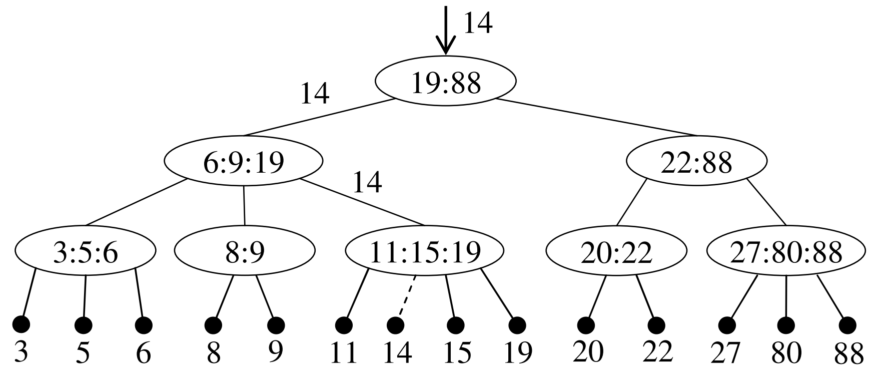
**Output**: the 2-3 tree obtained from *T* by inserting $b_1, b_2, \ldots, b_k$
**Model**: CREW PRAM of *k* processors

(1) A 2-3 tree is a rooted tree in which (i) each internal node has either 2 or 3 children, and (ii) every path from the root to a leaf is of the same length. Clearly, if the number of leaves is *n* the height of the tree is $O(\log n)$.

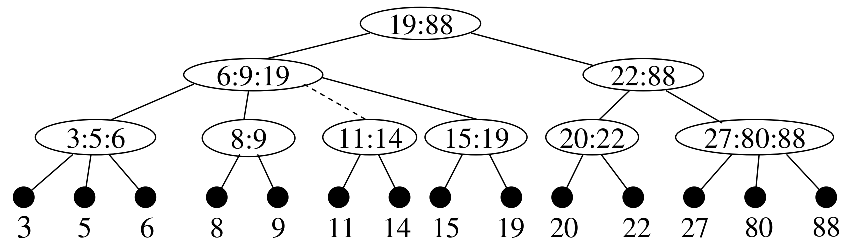
(2) A sorted list $a_1, a_2, \ldots, a_n$, can be represented by a 2-3 tree *T*, where the leaves hold the data items in a left-to-right order. And, an internal node will hold the largest data items stored in its subtrees. Searching for a data item on *T* can be performed in $O(\log n)$ time.

(3) Inserting an item *b* into a 2-3 tree in $O(\log n)$ time

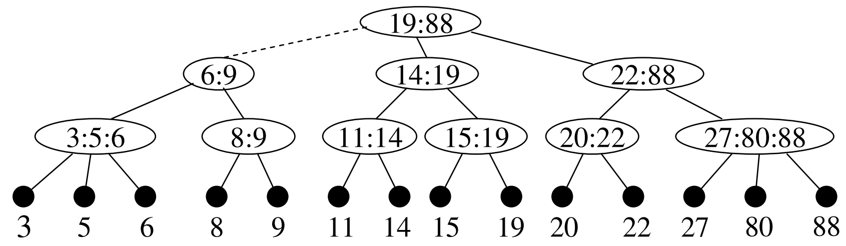
step 1: (1) locate 14, (2) create a leaf to hold 14, and (3) then insert it into <11:15:19>.



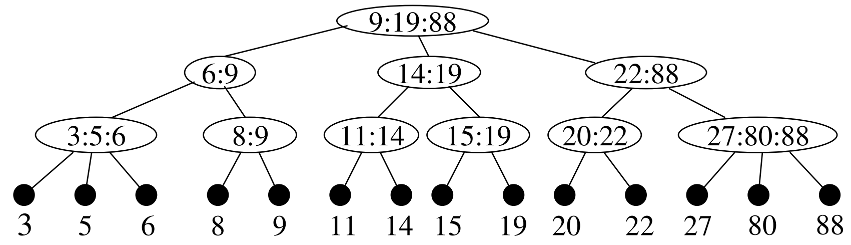
step 2: (1) split the node <11:15:19> into two nodes <11:14> and <15:19>, and then (2) insert <11:14> into <6:9:19>.



step 3: (1) split the node <6:9:19> into two nodes <6:9> and <14:19>, and then (2) insert <6:9> into <19:88>.

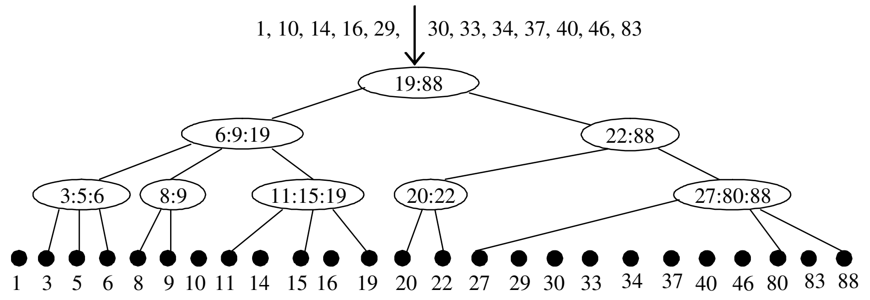


* after step 3

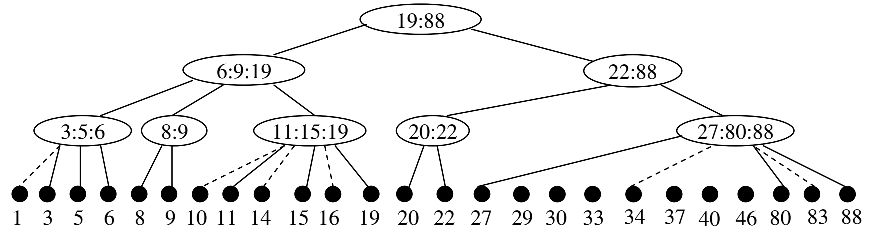


(4) Inserting a sorted sequence into a 2-3 tree

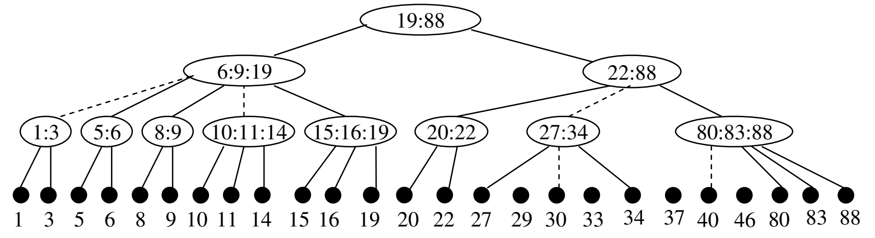
step 1: (1) locate $b_i$'s, and (2) create leaves to hold them.



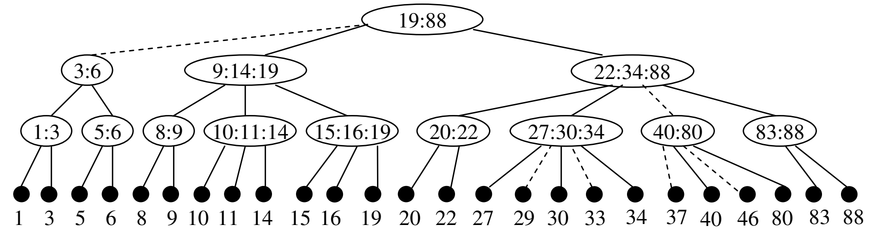
step 2: Let us call a **chain** the order set of elements among the $b_i$'s that have to fit between two consecutive leaves of $T$. Insert the medians of all **chains** into appropriate parents. (An internal node at height 1 may now have at most six children.)



step 3: (1) split nodes at height 1 that have more than 3 children and insert them into appropriate parents. (An internal node at height 2 may now have at most six children.) And, (2) then start the inserting of the medians of current chains.



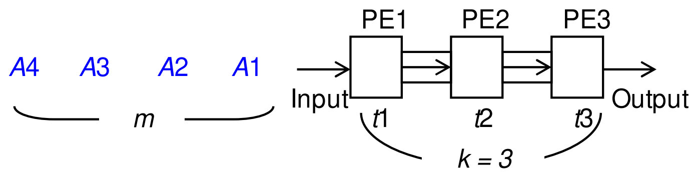
step 4: (1) split nodes at height 1 and 2 that have more than 3 children and insert them into appropriate parents. And, then (2) start the inserting of the medians of current chains.



* $T(n, k) = O(\log n + \log k)$

* *Pipelining*: Breaking up a task into a sequence of subtasks $t_1$, $t_2$, …, $t_k$, such that, once $t_1$ is completed, the sequence corresponding to a new task can begin and can proceed at the same rate as the previous task.

**pipelining**



time : $3m \Rightarrow 3 + (m - 1)$
time : $km \Rightarrow k + (m - 1)$ (Speedup = k)

* How to adapt the algorithm to EREW PRAM???

**Problem SM-8**: Finding maximum
**Input**: $A[1…n]$={3, 7, 8, 4} (Assume all elements are distinct.)
**Output**: **max**{ $A[1]$, $A[2]$, …, $A[n]$ }
**Model**: CRCW PRAM of $n^2$ processors
(For simplicity, we assume that each processor is indexed with a unique pair of $(i, j)$.)

step 1: (1) $P_{i,1}$ sets $MARK[i]$ as 0. (2) $P_{i,j}$ sets $C[i, j]$ as 0 if ($A[i] \geq A[j]$) and 1 otherwise.

$MARK[i]$ $C[i, j]$ (comparison matrix)

$$\begin{bmatrix}0\\0\\0\\0\end{bmatrix} \quad \begin{array}{c|cccc} & 3 & 7 & 8 & 4 \\ \hline 3 & 0 & 1 & 1 & 1 \\ 7 & 0 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

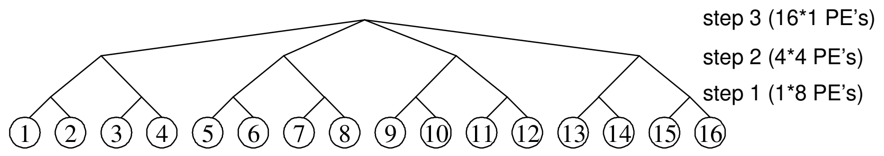
step 2: (1) $P_{i,j}$ sets $MARK[i] = 1$ if $C[i, j] = 1$. (2) $P_{i,1}$ sets $max = A[i]$ if $MARK[i] = 0$.

$MARK[i]$ $C[i, j]$ (comparison matrix)

$$\begin{bmatrix}1\\1\\\underline{0}\\1\end{bmatrix} \quad \begin{array}{c|cccc} & 3 & 7 & 8 & 4 \\ \hline 3 & 0 & 1 & 1 & 1 \\ 7 & 0 & 0 & 1 & 0 \\ \underline{8} & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

* OR operation can be done in $O(1)$ time using $n$ PEs.

**Problem SM-9**: Finding maximum
**Input**: $A[1\ldots n]$
**Output**: **max**{ $A[1], A[2], \ldots, A[n]$ }
**Model**: CRCW PRAM of $n$ processors ($n = 2^{2^k}$)



double logarithmic-depth tree of 16 leaves

**Algorithm**: The double logarithmic-depth tree can be used for computing the maximum as follows. Each internal node holds the maximum of the elements stored in its subtrees. The algorithm proceeds level by level, bottom up, starting with the nodes at height 1.

* $T(n) = O(\log\log n)$

* *Double logarithmic-depth tree* (of $n = 2^{2^k}$ leaves): the root of the tree has $2^{2^{k-1}}$ ($n^{1/2}$) children, each of its children has $2^{2^{k-2}}$ ($n^{1/4}$) children, and, in general, each node in the $i$-th level has $2^{2^{k-1-i}}$ children, for $0 \le i \le k-1$. Each node at level $k$ will have 2 leaves as children. (The depth of the tree is $O(\log\log n)$.)

* $\sqrt{n}$-way D&C

**Problem SM-10**: Finding maximum
**Input**: $A[1\ldots n]$
**Output**: **max**{ $A[1], A[2], \ldots, A[n]$ }
**Model**: CRCW PRAM of $n/\log\log n$ processors

**Remark**: The problem can be solved in $O(n)$ sequential time, which is optimal.

**Algorithm:**

**stage 1:** For each $i$, $1 \le i \le n/\log\log n$, computing the maximum $m_i$ of $A[(i-1)*\log\log n+1], A[(i-1)*\log\log n+2], \ldots, A[i*\log\log n]$ in $O(\log\log n)$ sequential time.

**stage 2:** Apply the algorithm proposed in **Problem SM-9** to compute the maximum of $m_1, m_2, \ldots, m_k$, using $k = n/\log\log n$ processors, in $O(\log\log k) = O(\log\log n)$ time.

* $T(n) = O(\log\log n)$

* *Accelerated Cascading*: Suppose there are two algorithms, one optimal but relatively slow, the other very fast but nonoptimal. The strategy is to (1) start with the optimal one until the size of the problem is reduced to a certain threshold value. And, then (2) shift to the fast one. This technique is useful for deriving very fast parallel optimal algorithms.

**Problem SM-11**: 3-coloring on a directed cycle

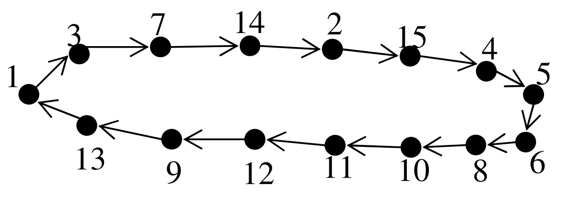
**Input**: $S[1\ldots n]=\{3, 15, 7, 5, 6, 8, 14, 10, 13, 11, 12, 9, 1, 2, 4\}$ (a directed cycle : $S[i]$, $1\le i\le n$, is the next node of node $i$.)

**Output**: A coloring $C[1\ldots n]$, in which $C[i]=0$, 1, or 2, and $C[i]\neq C[j]$ if $S[i]=j$.

**Model**: EREW PRAM of $n$ processors

**Remark 1:** We can obtain a $2*\lceil \log k\rceil$-coloring $C'$ from a $k$-coloring $C$ ($k>3$) in $O(1)$ time on an EREW PRAM of $n$ processors.

```
for $1\le i\le n$ pardo
begin
    1. Set $k(i)$ to the least significant bit position in which
       $C[i]$ and $C(S[i])$ disagree.
    2. Set $C'[i]=2k(i)+C[i]_{k(i)}$ (the $k$-th least significant bit of
       $C[i]$)
end
```



| $i$ | 1 | 3 | 7 | 14 | 2 | 15 | 4 | 5 | 6 | 8 | 10 | 11 | 12 | 9 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $C[i]$ | 1 | 3 | 7 | 14 | 2 | 15 | 4 | 5 | 6 | 8 | 10 | 11 | 12 | 9 | 13 |
| | $00\underline{0}1$ | $0\underline{0}11$ | $011\underline{1}$ | $1\underline{1}10$ | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |
| $k$ | 1 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 2 |
| $C'[i]$ | 2 | 4 | 1 | 5 | 0 | 1 | 0 | 1 | 3 | 2 | 0 | 1 | 0 | 4 | 5 |

* Suppose that $C'[i]=C'[S[i]]$ for some $i$. We have, $2k(i)+C[i]_{k(i)} = 2k(S[i])+C[S[i]]_{k(S[i])}$. Then, $C[i]_{k(i)} = C[S[i]]_{k(S[i])}$, which contradicts the definition of $k$. Thus, $C'$ is a coloring.

**Remark 2:** Sorting $n$ integers, each of which is in the range $[0, O(\log n)]$, can be done in $O(\log n)$ time on an EREW PRAM of $n/\log n$ processors. (S. Rajasekaran, and J. Reif, "Optimal and sublogarithmic time randomized parallel sorting algorithms," *SIAM J. Computing*, vol. 18, no. 3, pp. 594-607, 1989.)

**Algorithm**

```
step 1.  for $i$, $1\le i\le n$, pardo $C[i]=i$
step 2.  By Remark 1, obtain an $O(\log n)$ coloring.
step 3.  By Remark 2, sort the nodes by their colors.
step 4.  for $c=3$ to $2*\lceil \log (n+1)\rceil -1$  do
             for all nodes $i$ of color $c$ pardo
                 color $i$ with the smallest color from {0, 1, 2} that
                 is different from the colors of its two neighbors.
```

| $i$ | 1 | 3 | 7 | 14 | 2 | 15 | 4 | 5 | 6 | 8 | 10 | 11 | 12 | 9 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | $C[i]$'s | | | | | | |
| step 1 | 1 | 3 | 7 | 14 | 2 | 15 | 4 | 5 | 6 | 8 | 10 | 11 | 12 | 9 | 13 |
| step 2 | 2 | 4 | 1 | 5 | 0 | 1 | 0 | 1 | $\underline{3}$ | 2 | 0 | 1 | 0 | 4 | 5 |
| $c=3$ | 2 | $\underline{4}$ | 1 | 5 | 0 | 1 | 0 | 1 | **0** | 2 | 0 | 1 | 0 | $\underline{4}$ | 5 |
| $c=4$ | 2 | **0** | 1 | $\underline{5}$ | 0 | 1 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | **1** | $\underline{5}$ |
| $c=5$ | 2 | 0 | 1 | **2** | 0 | 1 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 1 | **0** |

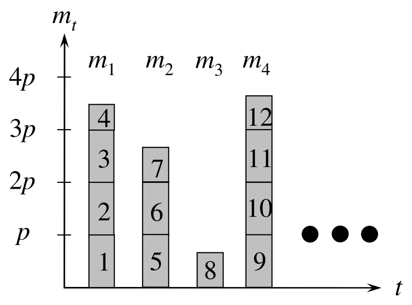
* $T(n)=O(\log n)$

* *Symmetry Breaking.*

## ■ Brent's Theorem

**Theorem:** Let $A$ be a given algorithm with a parallel computation time of $T$. Suppose that $A$ involves a total number of $M$ computational operations. Then $A$ can be implemented using $p$ processors in $O(M/p+T)$ parallel time. (R. P. Brent, "The parallel evaluation of general arithmetic expressions," *JACM*, vol. 21, no. 2, pp. 201-208, 1974.)

**proof:** Let $m_t$ be the number of computational operations performed (in parallel) in step $t$ of $A$. Using $p$ processors this can be simulated in $\lceil m_t/p \rceil$ time (if succeed).



Now summing all $t$, $1 \le t \le T$, we get the stated parallel computation time

$$\begin{aligned} & \lceil m_1/p \rceil + \lceil m_2/p \rceil + \ldots + \lceil m_T/p \rceil \\ \le \; & m_1/p + m_2/p + \ldots + m_T/p + T \\ = \; & O(M/p + T). \end{aligned}$$

Q.E.D.

* Note that the simulation in the proof may not succeed! (SM-11.)

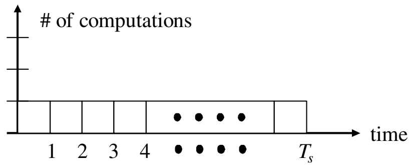
* As the accelerated cascading strategy, Brent's Theorem may be used to reduce the number of processors used in a derived algorithm that is nonoptimal.

* For example, consider the summing algorithm proposed in **Problem SM-1**, in which $T=O(\log n)$ and $M=1+2+4+\ldots+n/2 =O(n)$. By Brent's theorem, the problem can be solved in $O(n/p+\log n)$ time using $p$ processors. Let $p=n/\log n$. We obtained an optimal parallel algorithm, which solved the problem in $O(\log n)$ time using $O(n/\log n)$ processors.

**For a given problem PROB:**

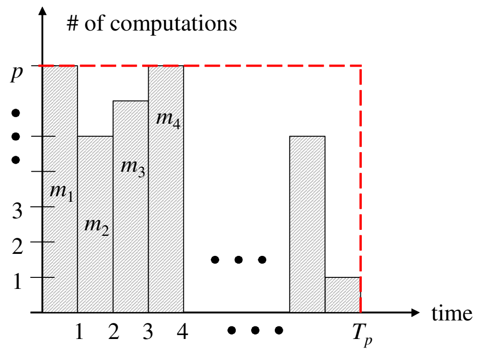
**1. The fastest sequential algorithm (known)**

| | |
|---|---|
| # of PEs | : 1 |
| time | : $T_s$ |



**A parallel algorithm A**

| | |
|---|---|
| # of PEs | : $p$ |
| time | : $T_p$ |



Cost = $p * T_p$
Speedup = $T_s / T_p$
Efficiency = $T_s / (p * T_p)$

**2. Simulate (2) sequentially**

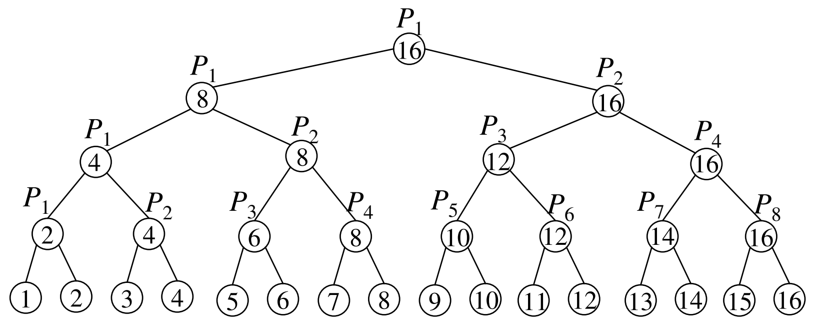
# of PEs : 1
time: $M = m_1 + m_2 + \ldots + M_{Tp}$

$\Rightarrow$ Speedup of A = $T_s/T_p \le M/T_p = p(M/(p * T_p)) \le p$

**Brent's Theorem v.s Accelerated cascading**

**Example : Finding Maximum**

1. Balanced binary tree method
   ($n$=16, number of processors=8)



**Algorithm 1**

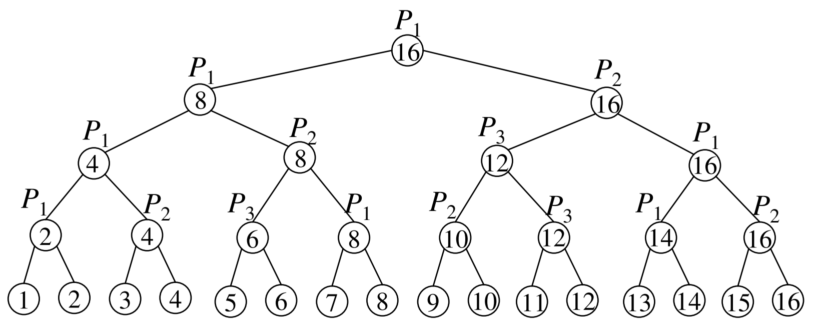
```
for k=log n–1 to 0 do
    for i, 1≤i≤2^k, pardo A[i]=MAX{A[2i–1], A[2i]}
```

**Analysis**

| | | |
|---|---|---|
| processors | $n/2$ | |
| time $T$ | $O(\log n)$ | |
| cost | $n \log n$ | (non-optimal) |
| total computation $M$ | $O(n)=O(n/2+n/4+\ldots+1)$ | |

* Sequentially, this problem can be solved in $O(n)$ time.

2. Improving **Algorithm 1** by Brent's theorem ($n$=16, $p$=3)



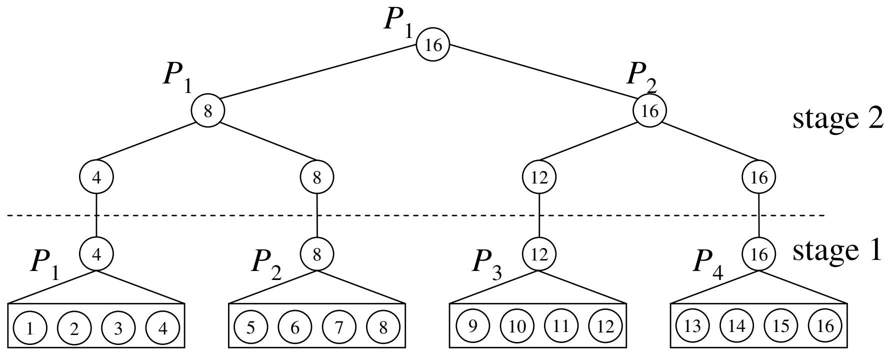
**Algorithm 2 (simulate Algorithm 1)**

```
for $k$=log $n$–1 to 0 do
  for $s$, $1 \le s \le 2^k/p$, do
    for $i$, $1 \le i \le p$, pardo
      $A[i+(s-1)*p]$=MAX$\{A[2(i+(s-1)*p)-1], A[2(i+(s-1)*p)]\}$
```

**Analysis**

| | |
|---|---|
| processors | $p$ |
| time | $O((n/2)/p + (n/4)/p + \ldots)=O(n/p + \log n)$ |
| cost | $n + p\log n$ (optimal for $p \le n/\log n$) |

3. Improving **Algorithm 1** by accelerated cascading
($n$ = 16, $p$ = 4)



**Algorithm 3** **(combine (1) sequential algorithm and (2) Algorithm 1)**

stage 1: Each $P_i$ determines $A'[i]$=**MAX**$\{A[(i-1)n/p+1], A[(i-1)n/p+2], \ldots, A[(i-1)n/p+n/p]$ in $O(n/p)$ sequential time.

stage 2: Perform **Algorithm 1** to determine MAX$\{A'[1], A'[2], \ldots, A'[p]\}$ in $O(\log p)$ time.

**Analysis**

| | |
|---|---|
| processors | $p$ |
| time | $O(n/p+ \log p)$ |
| cost | $n+ p\log p$ (optimal for $p \le n/\log n$) |

* Brent's Theorem: Simulation

* Accelerated cascading: Combine two algorithms