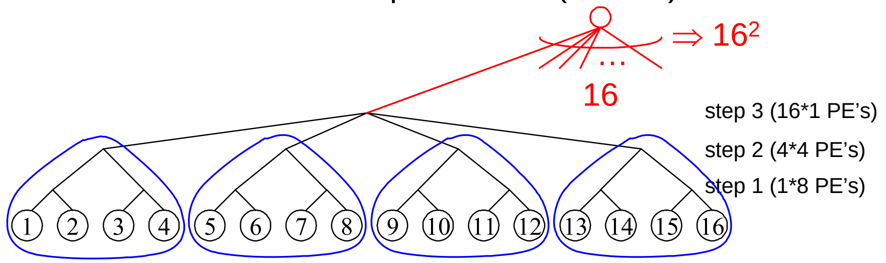
Problem SM-9: Finding maximum
Input: $A[1\ldots n]$
Output: **max**$\{ A[1], A[2], \ldots, A[n] \}$
Model: CRCW PRAM of n processors ($n = 2^{2^k}$)



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 method

* *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 (top-down view)

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

Sm-22ab

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

Algorithm:

n PEs, $O(\log\log n)$ time
1 PE. $O(n)$ time

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. $O(\log\log n)$

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.

$O(\log\log (n/\log\log n)) = O(\log\log n)$

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

* *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.

Sm-22c

Sm-23a

* sequential: ???

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

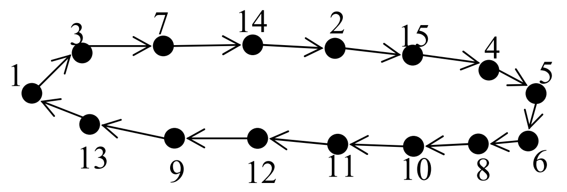
Input: $S[1\ldots n]=\{3, 15, 7, 5, 6, 8, 14, 10, 13, 11, 12, 9, 1, 2, 4\}$
(a directed cycle : $S[i]$, $1 \le i \le n$, is the next node of node i.)

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 $(2^*\log k+1)$-coloring C' from a k-coloring C ($k>3$) in $O(1)$ time on an EREW PRAM of n processors.

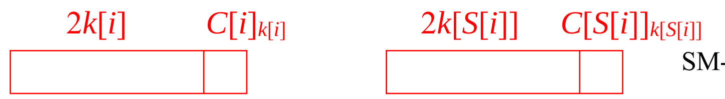
```
for 1≤i≤n pardo
begin
    1. Set $k(i)$ to the least significant bit position in which
       $C[i]$ and $C(S[i])$ disagree.    $C[i]_{k(i)} \neq C[S[i]]_{k(i)}$
    2. Set $C'[i]=2k(i)+C[i]_{k(i)}$ (the $k$-th least significant bit of
       $C[i]$)
end
```



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
	$\underline{0001}$	$\underline{0011}$	$\underline{0111}$	$\underline{1110}$	--	--	--	--	--	--	--	--	--	--	--
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

* how to find k(i) ???

Sm-23b



* 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≤i≤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*log n 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**

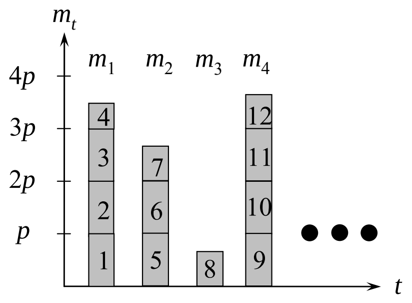
* $T(n)=O(\log n)$

* *Symmetry Breaking*. **cyclic dependency in computation*

■ 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

$$
\begin{aligned}
& \lceil m_1/p \rceil + \lceil m_2/p \rceil + \dots + \lceil m_T/p \rceil \\
\le\ & m_1/p + m_2/p + \dots + m_T/p + T \\
=\ & O(M/p + T).
\end{aligned}
$$
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. Sm-26a

* For example, consider the summing algorithm proposed in **Problem SM-1**, in which $T=O(\log n)$ and $M=1+2+4+\dots+n/2 =O(n)$. 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.

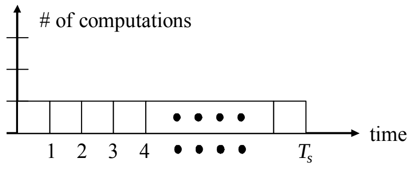
* How to find the best p???

* Prove: no super-linear speedup

For a given problem PROB:

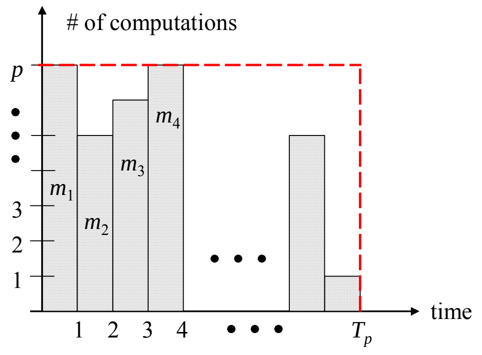
1. The fastest sequential algorithm (known)

# 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 = $T_s/(p*T_p)$

linear speedup:
speedup = $\Theta(p)$

2. Simulate (2) sequentially

# of PEs : 1 → a sequential algorithm: $M \geq T_s$

time: $M = m_1 + m_2 + \ldots + M_{T_p}$

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

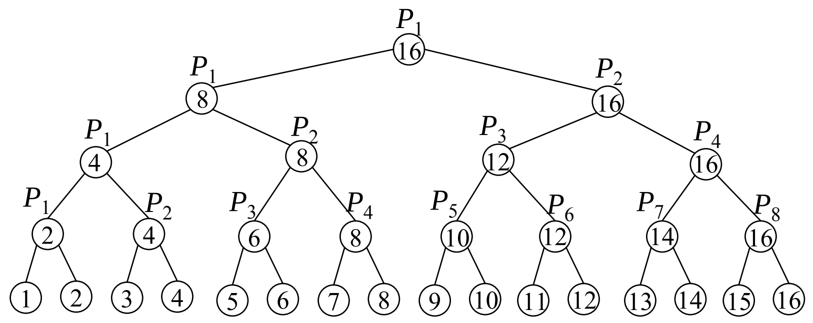
* speedup > p => new best seq. algo (or caused by some hardware)

* Exercise: prove that efficiency ≤ 1

Brent's Theorem v.s Accelerated cascading

Example : Finding Maximum

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



P_1 16
P_1 8 P_2 16
P_1 4 P_2 8 P_3 12 P_4 16
P_1 2 P_2 4 P_3 6 P_4 8 P_5 10 P_6 12 P_7 14 P_8 16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Algorithm 1

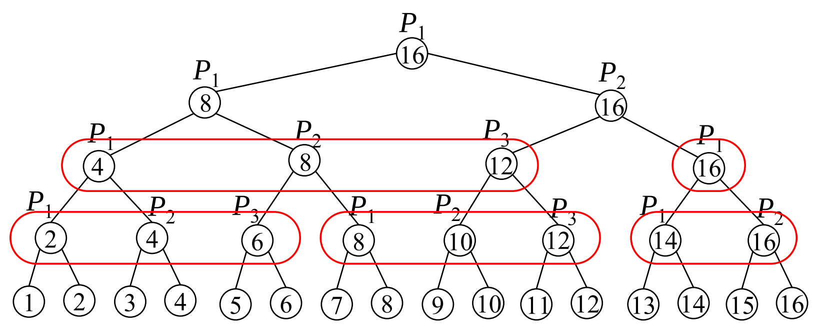
```
for $k$=log $n$–1 to 0 do
    for $i$, $1 \le i \le 2^k$, pardo $A[i]$=MAX{$A[2i–1]$, $A[2i]$}
```

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)

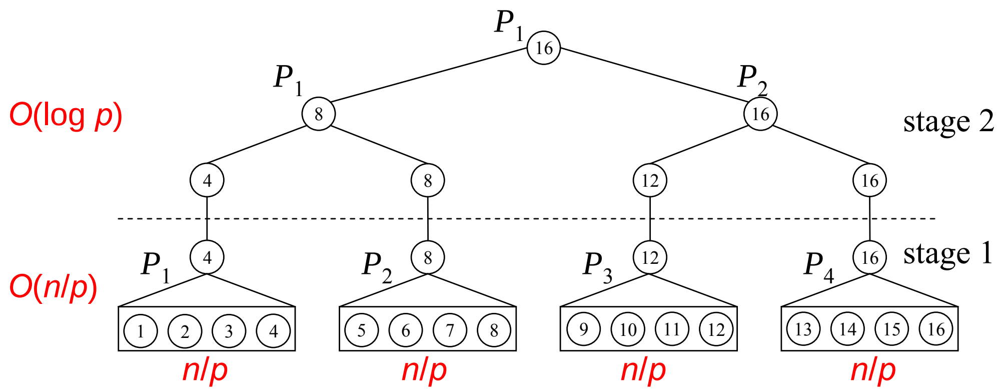
```
for $k$=log $n$–1 to 0 do
  for $s$, $1 \le s \le 2^k/p$, do
    for $i$, $1 \le i \le p$, pardo
      $A[i+(s-1)*p]$=MAX{$A[2(i+(s-1)*p)-1]$, $A[2(i+(s-1)*p)]$}
```

Analysis

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

$p=n/\log n \Rightarrow \log n$ time!

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**{$A[(i-1)n/p+1]$, $A[(i-1)n/p+2]$, …, $A[(i-1)n/p+n/p]$ in $O(n/p)$ sequential time.

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

Analysis $p=n/\log n \Rightarrow \log n$ time!

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

A cost optimal prefix sums algorithm

SM-3:	n PEs, $O(\log n)$ time => cost: $n \log n$, M: $n\log n$
sequential:	1 PEs, $O(n)$ time => cost: n

$n = 12$ $p = n/\log n = 4$

$$A:\quad \underbrace{1\quad 1\quad 2}_{\log n}\quad \underbrace{1\quad 4\quad 2}\quad \underbrace{2\quad 3\quad 1}\quad \underbrace{2\quad 1\quad 2}$$

initially ($|A|=n$)

$$A':\quad 4\quad 7\quad 6\quad 5$$

stage 1 ($|A'|=n/\log n$) O(n/p)=O(log n) (sequential)

$$S':\quad 4\quad 11\quad 17\quad 22$$

stage 2 O(log p) (parallel)

A:	1	1	2	1	4	2	2	3	1	2	1	2
S:			4			11			17			22

initially of stage 3

A:	1	1	2	1	4	2	2	3	1	2	1	2
S:	1	2	4	5	9	11	13	16	17	19	20	22

stage 3 O(n/p) (sequential)

* Accelerated cascading

$$\begin{array}{ll} A_1 & \boxed{\text{fast, non-optimal}} \\ & \quad + \\ A_2 & \boxed{\text{slow, optimal}} \end{array} = \boxed{\text{fast, optimal}}$$