The Power of CRCW – Minima

- Two points of interest
  - Illustrate the power of CRCW models
  - Illustrate another optimality technique.
- Find the minima of n elements.
  - Input: An array A of n elements
  - Output: The minimum element in A.
- From what we already know:
  - Standard sequential algorithm not good enough
  - Can use an upward traversal, with min as the operator at each internal node. Time = O(log n), work = O(n).

The Power of CRCW – Minima

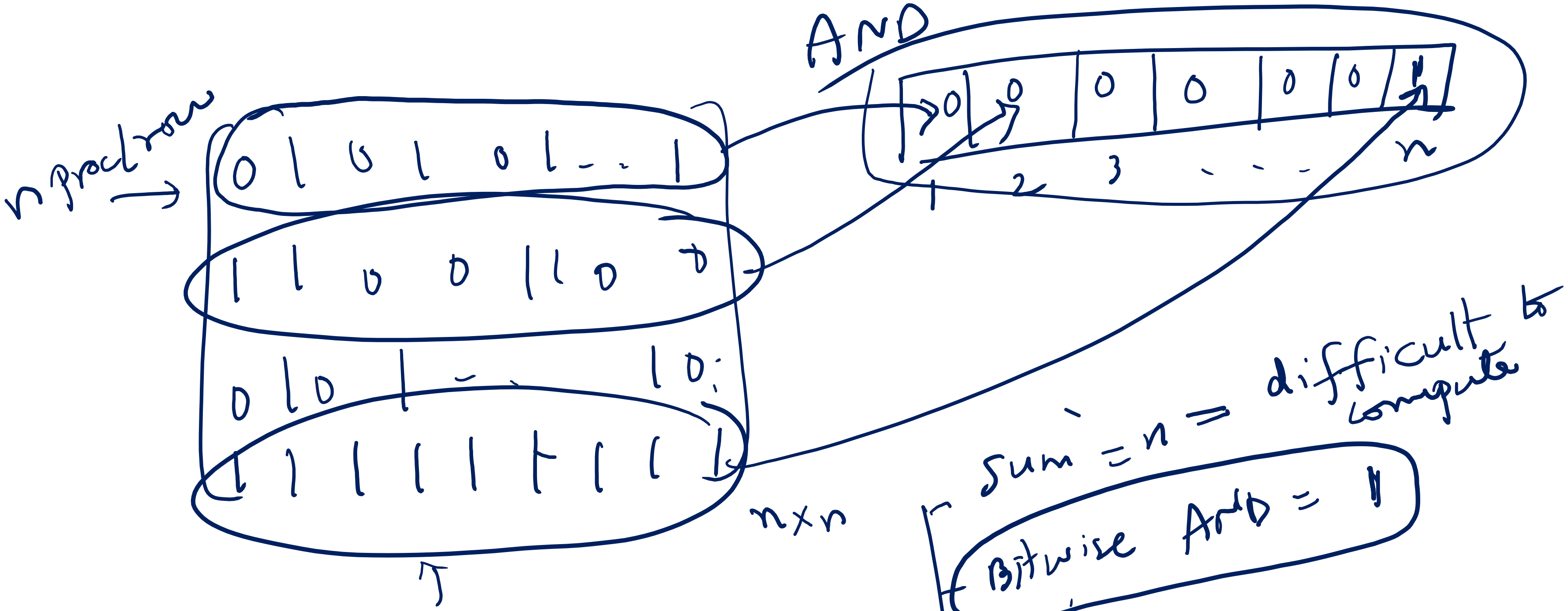
- Our solution steps:
  - Design a $O(n^2)$ work, $O(1)$ time algorithm.
  - Gain optimality by sacrificing runtime to $O(\log \log n)$.

An O(1) Time Algorithm

	12	17	8	18	26
12	--	1	0	1	1
17	0	--	0	1	1
8	1	1	--	1	1
18	0	0	0	--	1
26	0	0	0	0	--

- Use n^2 processors.
- Compare A[i] with A[j] for each i and j.
- Now can identify the minimum.

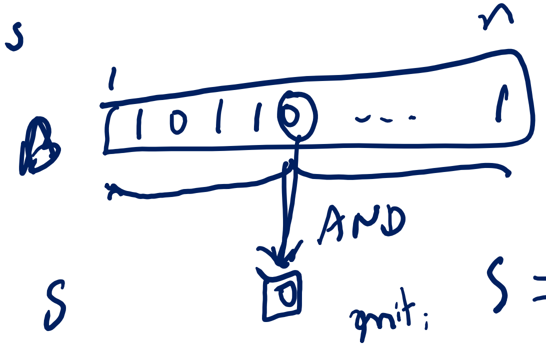
n^2 Processors



sum = n ⇒ difficult to compute

Bitwise AND = 1

CRCW helps



init: $S = \cancel{1}\ 0$

$$S = \begin{cases} 1 & \text{if all } B(i) = 1 \\ 0 & \text{if any } B(i) = 0 \end{cases}$$

if $B(i) = 0$ write 0 to S // for all i in parallel

An O(1) Time Algorithm

	12	17	8	18	26
12	--	1	0	1	1
17	0	--	0	1	1
8	1	1	--	1	1
18	0	0	0	--	1
26	0	0	0	0	--

- Use n^2 processors.
- Compare A[i] with A[j] for each i and j.
- Now can identify the minimum
  - How?

An O(1) Time Algorithm

	12	17	8	18	26
12	--	1	0	1	1
17	0	--	0	1	1
8	1	1	--	1	1
18	0	0	0	--	1
26	0	0	0	0	--

- Use n^2 processors.
- Compare A[i] with A[j] for each i and j.
- Now can identify the minimum.
  - How?
- Where did we need the CRCW model?

Towards Optimality

- The earlier algorithm is heavy on work.
- To reduce the work, we proceed as follows.
- We derive an O(nlog log n) work algorithm running in O(log log n) time.
- For this, use a doubly logarithmic tree.
  - Defined in the following.

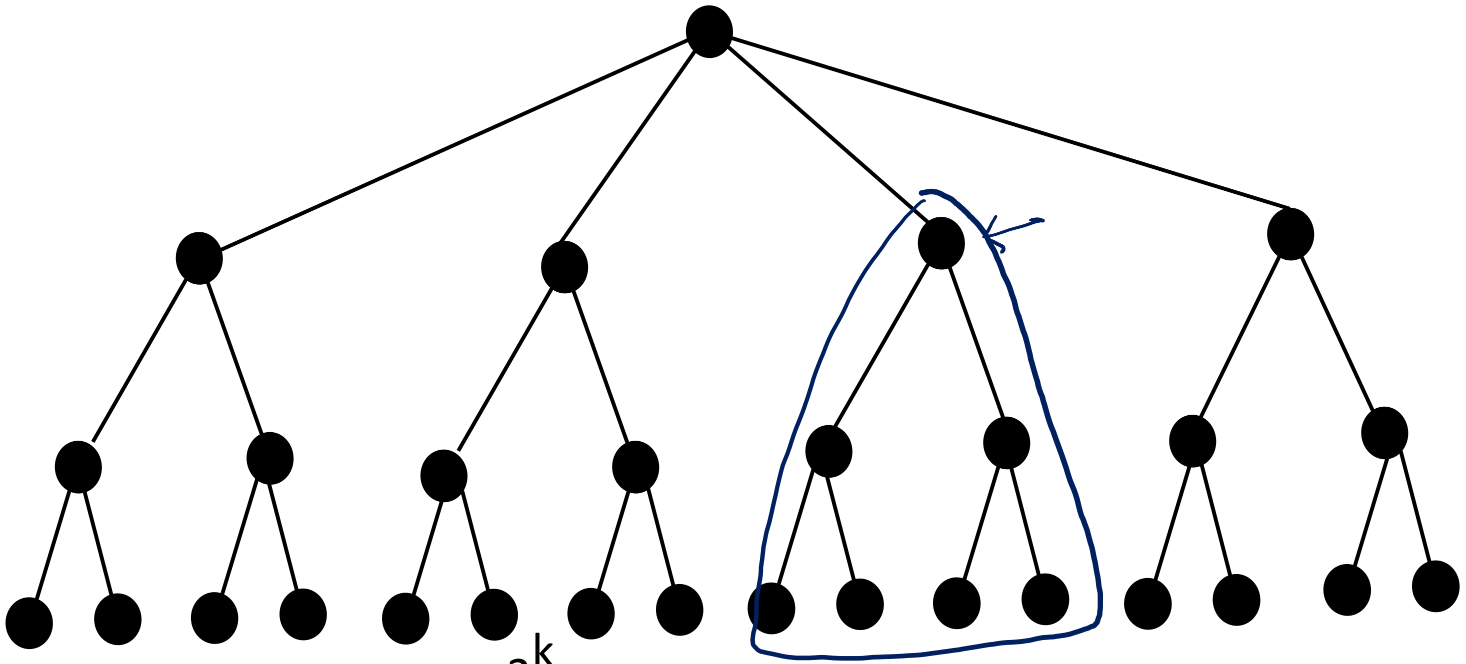
Work : $n^2 \longrightarrow n\log\log n$ not optimal

Time : $O(1) \longrightarrow O(\log\log n)$

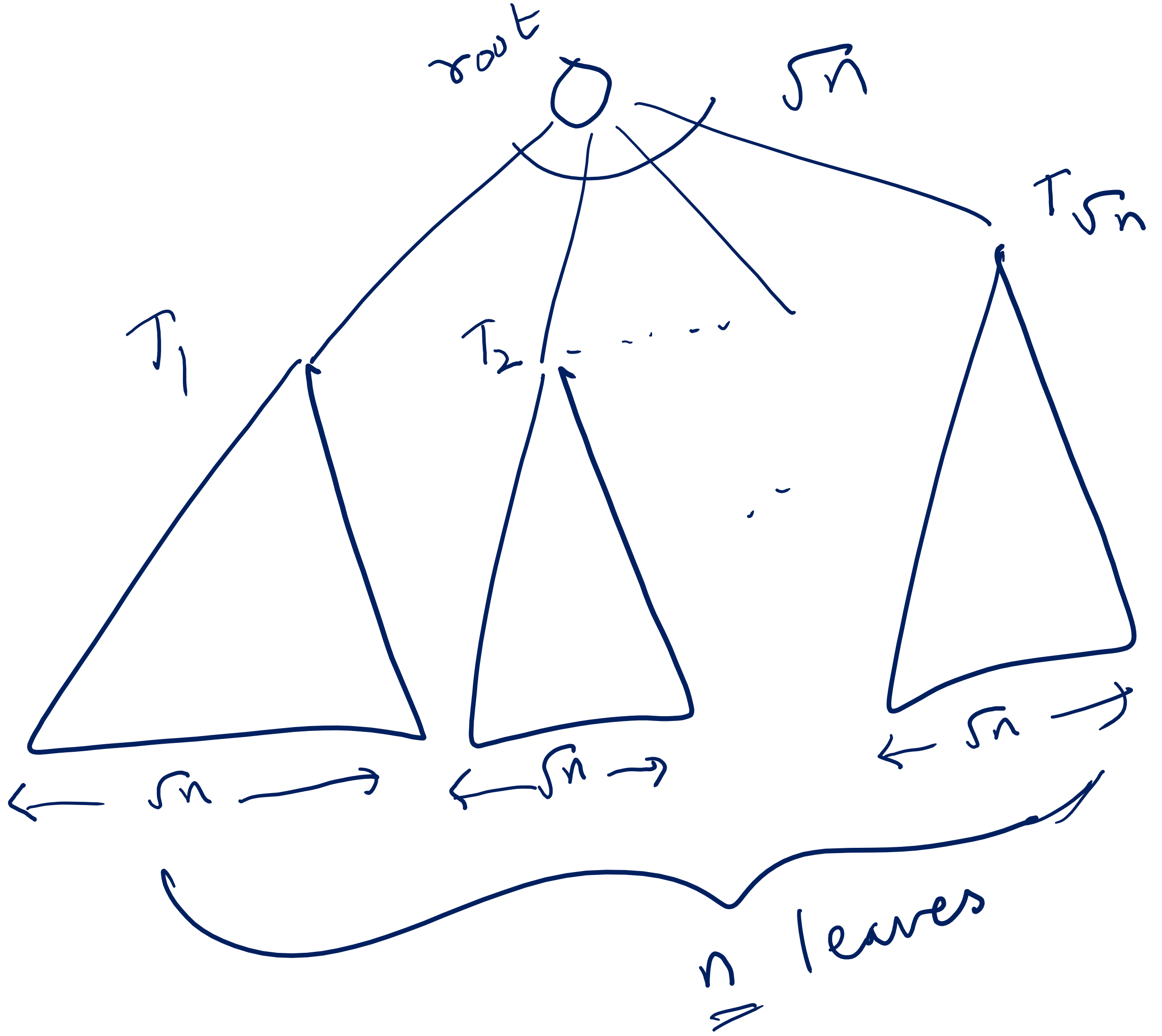
optimal

start with a smaller problem size

Doubly Logarithmic Tree



- Let there be $n = 2^{2^k}$ leaves, the root is level 0. The root has $\sqrt{n} = 2^{2^{k-1}}$ children.
- In general, a node at level i has $2^{2^{k-i-1}}$ children, for $0 \leq i \leq k$.
- Each node at level k has two leaf nodes as children.



Root : $\sqrt{n}$ children

$\rightarrow$ $\sqrt{n}$ subtrees

T_1 : a D.L. tree for $\sqrt{n}$ nodes as leaves

Doubly Logarithmic Tree

- Some claims:
  - Number of nodes at level i is $2^{2^k - 2^{k-i}}$.
  - Number of nodes at the kth level is n/2.
  - Depth of a doubly logarithmic tree of n nodes is k+1 = log log n + 1.
- To compute the minimum using a doubly logarithmic tree:
  - Each internal node performs the min operation does not suffice.
  - Why?

Minima Using the Doubly Logarithmic Tree

- Intuition:
  - Should spend only O(1) time at each internal node.
  - Use the O(1) time algorithm at each internal node.
- At each internal node of level i, if there are c_i children, use c_i^2 processors.
  - Minima takes O(1) time at each level.
  - Also, No. of nodes at level i x No. of processors used =

    $2^{2^k - 2^{k-i}} . (2^{2^{k-i-1}})^2 = 2^{2^k} = n.$

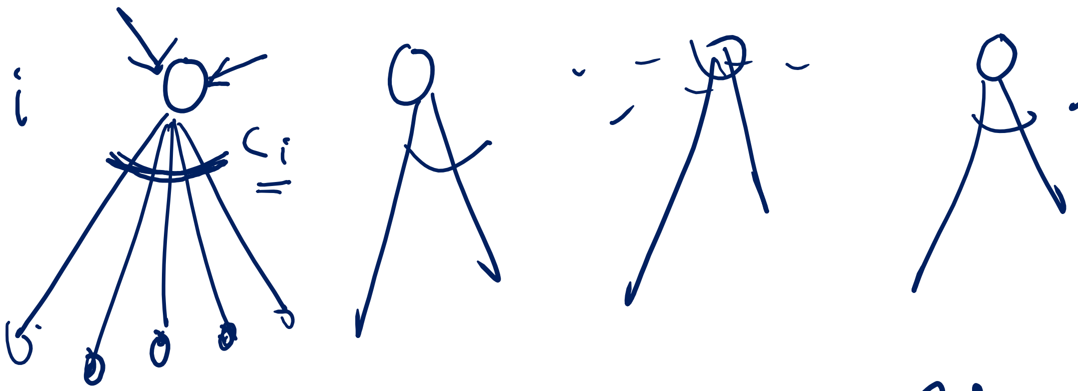
degree $= c_i$

$\#$Proc's $= c_i^2$

Time for minima $= O(1)$

$\#$Proc's at level $i = n$

Level i



$\#$ nodes at level i $= n_i$

$\#$ n_i's

g^2 comparisons

$n_i \times c_i^2$ Processors $= 2^{2^k - 2^{k-i}} \times \left(2^{2^{k-i-1}}\right)^2$

$= 2^{2^k - 2^{k-i}} \times 2^{2^{k-i}} = 2^{2^k} = n$

Minima Using a Doubly Logarithmic Tree

- Second, slightly improved result:
  - With n processors, can find the minima of n numbers in O(log log n) time.
  - Total work = O(n log log n).
- Still suboptimal by a factor of O(log log n).
- We now introduce a technique to achieve optimality.

Accelerated Cascading

- Our two algorithms:
  - Algorithm 1: A slow but optimal algorithm.
    - Binary tree based: O(log n) time, O(n) work.
  - Algorithm 2: A fast but non-optimal algorithm
    - Doubly Logarithmic tree based: O(log log n) time, O(nlog log n) work.
- The accelerated cascading technique suggests combining two such algorithms to arrive at an optimal algorithm
  - Start with the slow but optimal algorithm till the problem is small enough
  - Switch over to the fast but non-optimal algorithm.

Accelerated Cascading

- The binary tree based algorithm starts with an input of size n.
- Each level up the tree reduces the size of the input by a factor of 2.
- In log log log n levels, the size of the input reduces to $n/2^{\text{logloglog } n}$ = n/loglog n.

- Now switch over to the fast algorithm with n/loglog n processors, needing O(log log (n/log log n)) time.

Final Result

- Total time = O(log log log n) + O(log log n).
- Total work = O(n).
- Need CRCW model.
- Where did we need the CRCW model?