A Model

Fixed Connection Network

- Processors Labeled $P_1, P_2, \ldots, P_N$
- Each Processor knows its Unique *ID*
- Local Control
- Local Memory
- Fixed Bi-directional Connections
- Synchronous

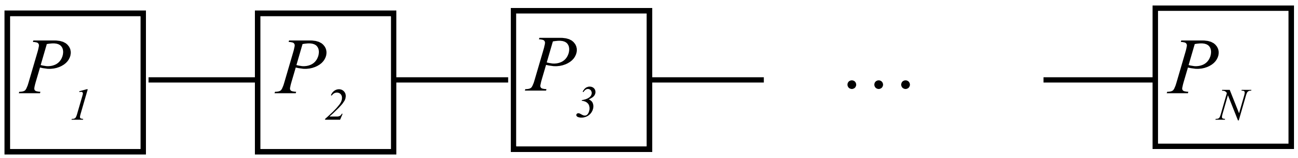
  Global Clock Signals Next Phase

Operations at Each Phase

Each Time the Global Clock Ticks

- Receive Input from Neighbors
- Inspect Local Memory
- Perform Computation
- Generate Output for Neighbors
- Update Local Memory

A Model of Cooperation: Bucket Brigades

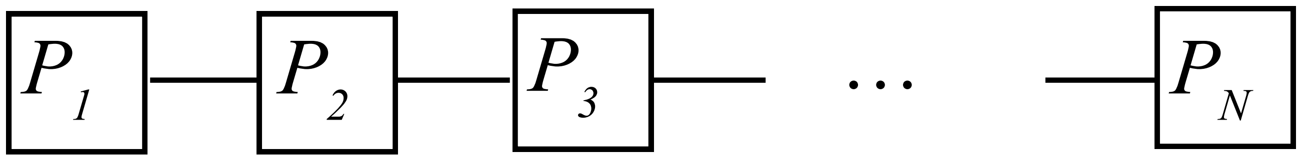


- N Processors, Labeled P_1 to P_N

- Processor P_i is connected to P_{i+1}, $i<N$ and P_{i-1}, $i>0$

A Sort Algorithm

Odd-Even Transposition on Linear Array



- The Array is $X[1 : N]$
- P_i's Local Variable X is $X[i]$
- P_i's have a Local Variables Y and a Global/Singular variable *Step*
- *Step* is initialized to Zero (0) at all Pi
- Compares and Exchanges are done alternately at Odd/Even - Even/Odd Pairs

Odd-Even Transposition

Algorithmic Description of Parallel Bubble Sort

At Each Clock Tick and For Each P_i do {

$Step$ ++;

if parity(i) = = parity($Step$) & $i < N$ then

Read from P_{i+1} to Y;

$X = \min(X,Y)$

else if $i > 1$ then

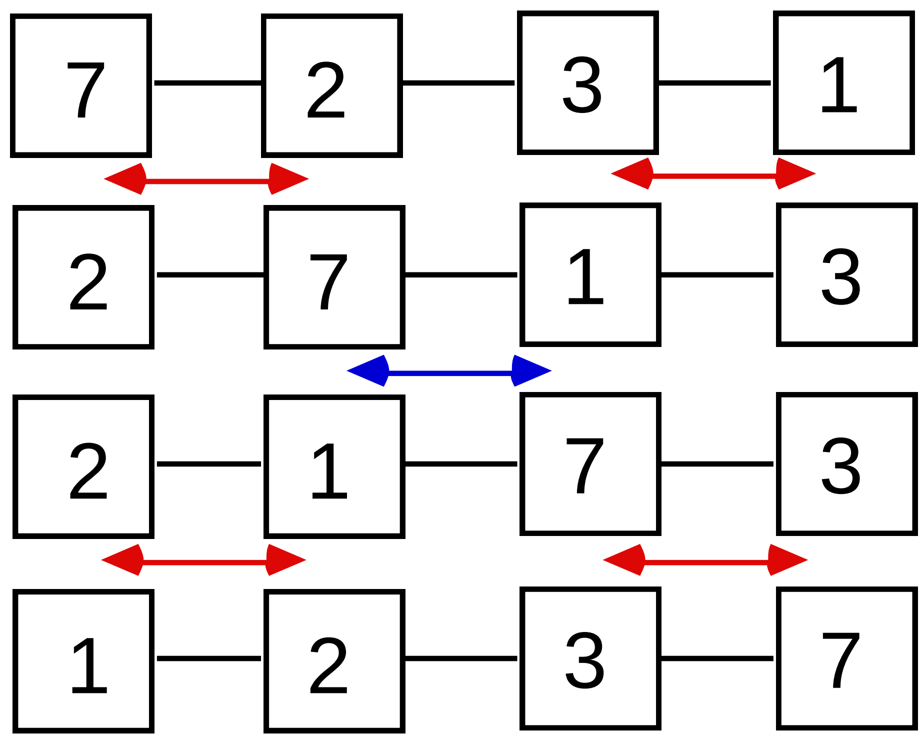
Read from P_{i-1} to Y;

$X = \max(X,Y)$;

}

Example of Parallel Bubble Sort

Sort 4 Numbers 7, 2, 3, 1 on an Array of 4 Processors



Case of 4, 3, 2, 1 Takes 4 Steps

Measuring Benefits

How Do We Measure What We Have Gained?

- Let $T_1(N)$ be the Best Sequential Algorithm
- Let $T_P(N)$ be the Time for Parallel Algorithm (P processors)
- The Speedup $S_P(N)$ is $T_1(N)/T_P(N)$
- The Cost $C_P(N)$ is $P \times T_P(N)$, assuming P processors
- The Work $W_P(N)$ is the summation of the number of steps taken by each of the processors. It is often, but not always, the same as Cost.
- The Cost Efficiency $CE_P(N)$ (often called efficiency $Ep(N)$) is
  $$S_P(N)/P = C_1(N) / C_P(N) = T_1(N) / (P \times T_P(N))$$
- The Work Efficiency $WE_P(N)$ is
  $$W_1(N) / W_P(N) = T_1(N) / W_P(N)$$

Napkin Analysis of Parallel Bubble

How'd We Do ? - Well, Not Great !

- $T_1(N) = N \lg N$ — Optimal Sequential
- $T_N(N) = N$ — Parallel Bubble
- $S_N(N) = \lg N$ — Speedup
- $C_N(N) = W_N(N) = N^2$ — Cost and Work
- $E_N(N) = \lg N / N$ — Cost and Work Efficiency

But Good Relative to Sequential Bubble

$$S_N(N) = N^2/N = N \; ; \; E_N(N) = S_N(N) / N = 1 \; !$$

Non-Scalability of Odd-Even Sort

Assume we start with 1 processor sorting 64 values, and then try to scale up by doubling number of values (N), each time we double number of processors (P) in a ring. The cost of the parallel sort requires each processor to sort its share of values (N/P), and then do P swaps and merges. Since P processors are busy, the cost is N lg N/P. After the local sort, sets are exchanged, merged, and parts thrown away. The merge costs N/P on each of P processors, for a Cost of N, and P-1 such merges occur, for a total cost of N×(P-1). Efficiency is then

E = N lg N / (N lg N/P + N×(P-1)) = lg N / (P - 1 + lg N - lgP)

First 2 columns double N as P doubles. Second three try to increase N to keep efficiency when P doubles.

N	P	E	N	P	E
64	1	1.0000	64	1	1.0000
128	2	1.0000	4096	2	1.0000
256	4	0.8889	16777216	4	0.9600
512	8	0.6923	2.81475E+14	8	0.9231
1024	16	0.4762	7.92282E+28	16	0.8972
2048	32	0.2973	6.2771E+57	32	0.8807
4096	64	0.1739	3.9402E+115	64	0.8707
8192	128	0.0977	1.5525E+231	128	0.8649

Cost for Finding Max Value in a List

Given a sequence A of n elements find the largest of these elements.

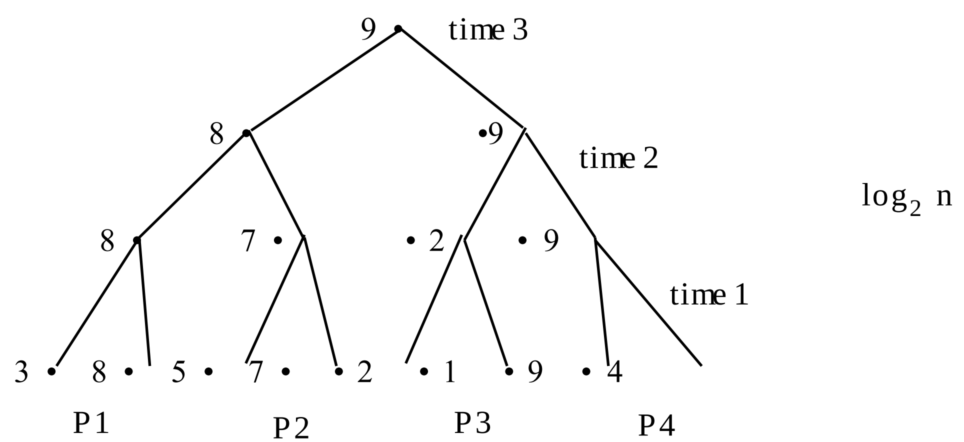
Serial Algorithm.

Largest = A [0]

For i = 1 to n-1 do { if A [i] > Largest then Largest = A [i] }

n - 1 comparison.

A Parallel Algorithm



Efficiency of Binary Tree Max

Assume Full Binary Tree

- $T_{N/2}(N) = T_{N/4}(N/2) + 1, N > 1$

  $T_1(2) = 1$

  $T_{N/2}(N) = lg\ N = O(lg\ N)$

- $C_N(N) = N\ lg\ N = O(N\ lg\ N)$

  $\mathrm{E}_N(N) = N / N\ lg\ N = O(1 / lg\ N)$

- $W_{N/2}(N) = W_{N/4}(N/2) + N/2, N > 2$

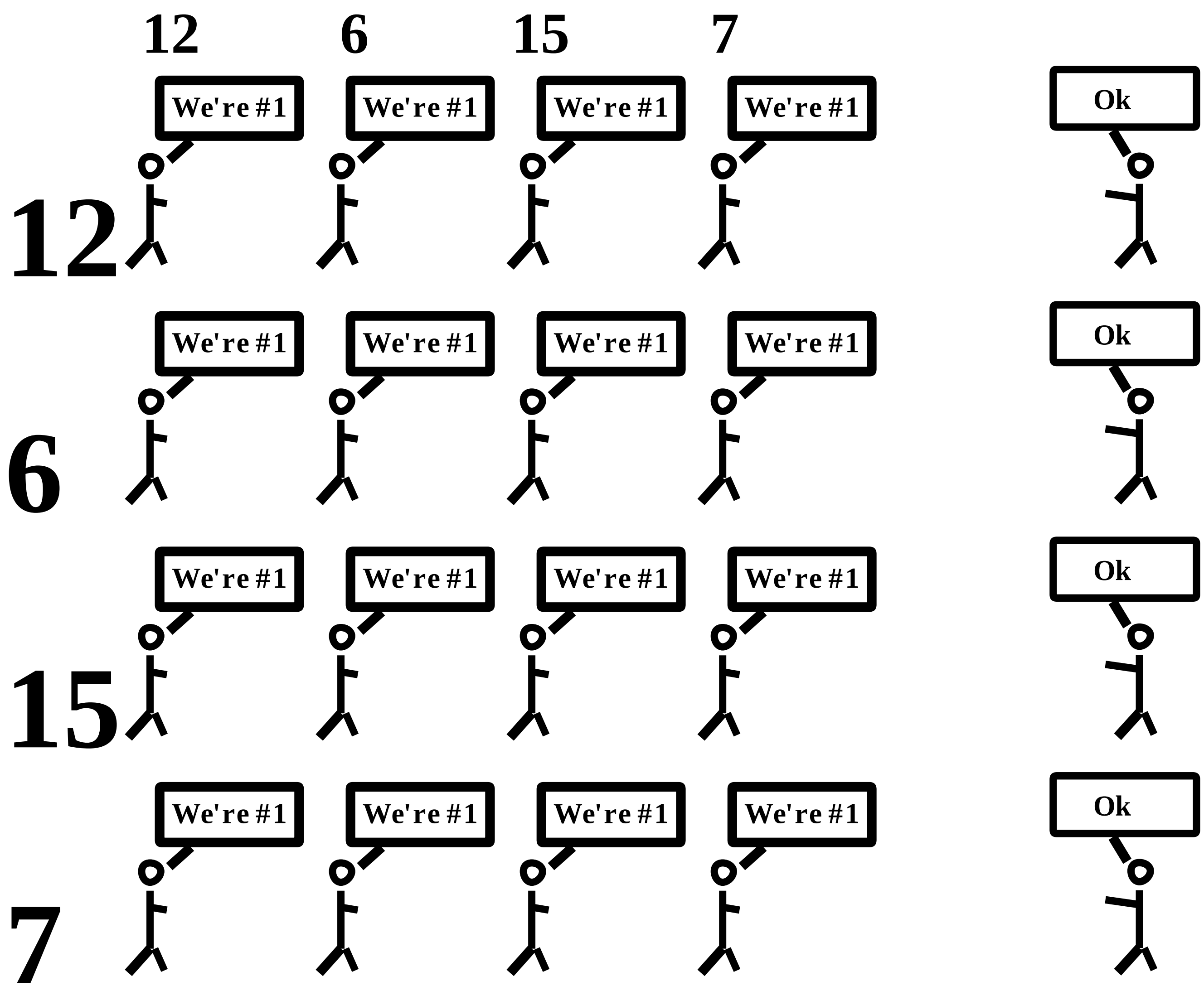
  $W_1(2) = 1$

  $W_{N/2}(N) = N - 1 = O(N)$

- This is optimally work efficient.
- But it is not optimally cost efficient.

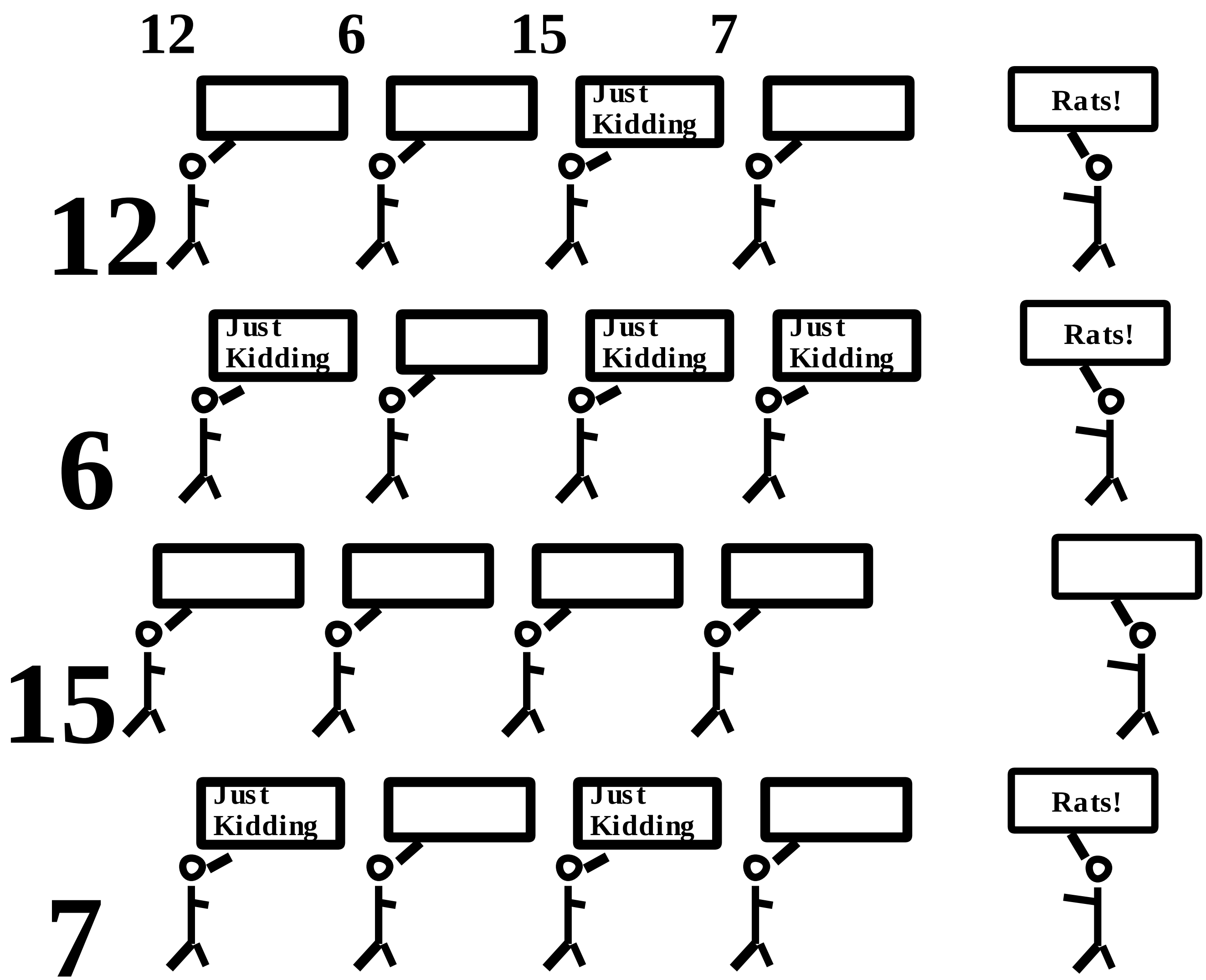
Finding the Maximum by Controlled Anarchy

Step#1: Everyone's an Optimist



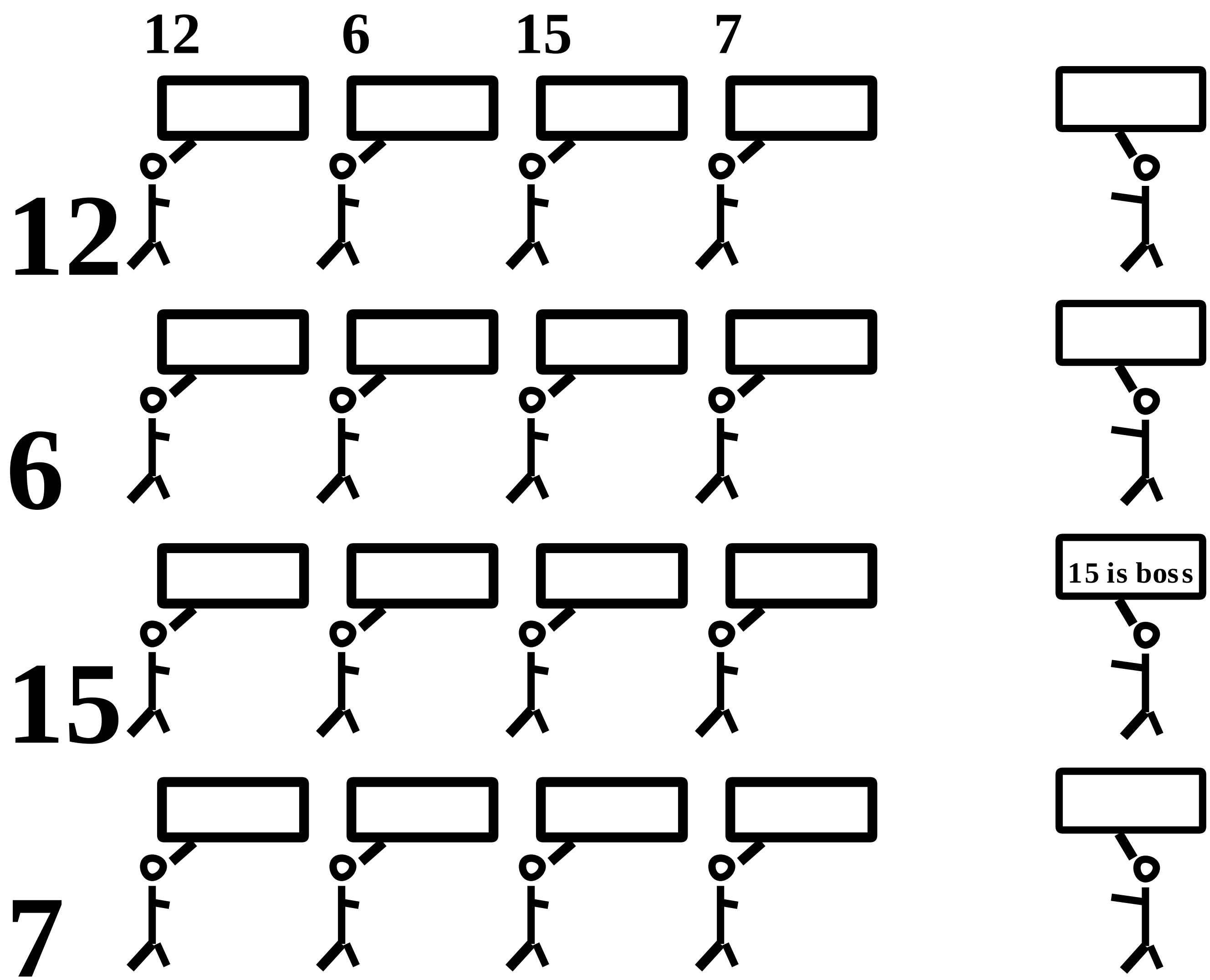
This is the Meatiest Part

Step#2: Realism Sets In



That's All Folks

Step#3: Reporting the Answer



Analysis of Very Fast Max

Optimal in Time, Not Work on CRCW (Concurrent Read Concurrent Write) PRAM (Parallel Random Access Machine)

- Assign **N** processors to initialize **M** in 1 step.
- Assign all $\mathbf{N^2}$ processors to first statement to fill **B** in 1 step.
- Assign all $\mathbf{N^2}$ processors to 2nd statement to fill **M** in 1 step.
- Assign **N** processors to 3rd statement to select **maxVal** in 1 step.

That Was Inefficient but Real Fast

- Can Solve Any Size Problem in 3 Steps
  But we need to make unreasonable assumptions about memory (CRCW)

- Use Lots of Processors
  Over a Million to Find Max of 1000

- We Want Fast but Not Too Expensive