Advanced Topics in DS:
Distributed Sorting

Matteo Di Giovanni
Claudio Di Sipio
Andrea Di Stefano
Cintia Scafa

Outline of the talk

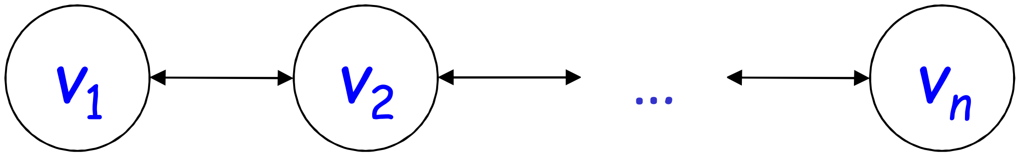
Sorting is a classic problem that we are going to present in a distributed computing setting, analyzing some interesting topologies.

Distributed sorting is applied to managing and processing large data sets with a parallel and distributed algorithm, such as the MapReduce technique.

DEFINITION (Sorting): We choose a graph with n nodes $v_1, \ldots, v_n$. Initially each node stores a value. After applying a sorting algorithm, node v_k stores the k^{th} smallest value.

First steps: Array Topology

Let's start analyzing the problem with a simple topology, the array.



Although this is a quite simple topology, we can use it to prove some interesting properties that we will reuse later on.

Here is a first algorithm for the sorting problem.

Odd/Even Sort

1: Given an array of n nodes $(v_1,\dots,v_n)$, each storing a value (not sorted)

2: REPEAT

3: Compare and exchange the values at the nodes i and $i+1$, i odd

4: Compare and exchange the values at the nodes i and $i+1$, i even

5: UNTIL DONE

The "Compare and exchange" primitive works in this way: if value v_i is stored in node i, after the operation i stores $\min(v_i, v_{i+1})$ and node $i+1$ stores $\max(v_i, v_{i+1})$.

0-1 Sorting Lemma

LEMMA 1: If an oblivious comparisons-exchange algorithm sorts all inputs of 0's and 1's, then it sorts arbitrary inputs.

Remark: "oblivious" means that the exchange between two values must only depend on their relative order. From now on, we will always restrict our inputs to 0's and 1's.

PROOF

Let's prove the contrapositive: does not sort arbitrary input ➔ does not sort 0's and 1's.

Suppose we have an input $x=x_1,\dots,x_n$ which is not sorted correctly by the algorithm. Then, there is a smallest value k such that the value at node v_k after running the sorting algorithm is strictly larger than the k^{th} smallest value, namely $x[k]$. We define a new input

$$x_i^* = \begin{cases} 0 \Leftrightarrow x_i \leq x[k] \\ 1 \text{ otherwise} \end{cases}$$

This input must be sorted by the algorithm in the same way as before, so in any case in which $x_i^* = 0$ and $x_j^* = 1$, this means that $x_i \leq x[k] < x_j$. Therefore, if the previous input was not correctly sorted, then also the new one can not be correctly sorted.

QED

Correctness/Efficiency

THEOREM: Odd/Even algorithm sorts correctly in n steps.

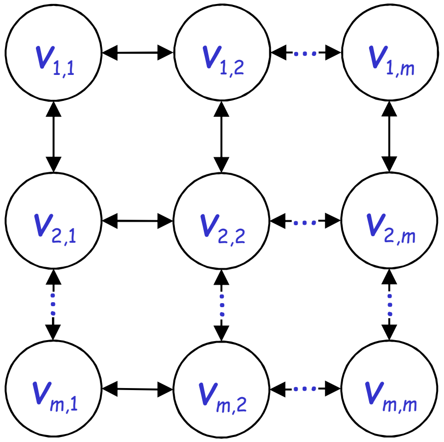
PROOF: Thanks to Lemma 1, we can consider an array of 0's and 1's. The proof follows by induction.

- **BASIS**: Let j_1 be the node with the rightmost 1. If j_1 is odd (even), it will move in the first (second) step, and it will continue moving right until it reaches the rightmost node v_n.
- **INDUCTION:** Let j_k be the k^{th} rightmost 1. By induction, after step k j_k can move constantly right until it reaches destination. Since j_{k-1} moves after step k-1, for each step after step k j_k gets a right 0-neighbor.

QED

Mesh topology

Let's now analyze a different topology, the mesh (also known as grid).



SHEARSORT

1: We are given a mesh with *m* rows and *m* columns, *m* even, $n=m^2$.

2: REPEAT

3: in the odd phases 1, 3, … we sort all the rows; in the even phases 2, 4, … we sort all the columns, such that:

4: Columns are sorted such that small values move up

5: Odd rows (1, 3, …, *m*-1) are sorted such that small values move left

6: Even rows (2, 4, …, *m*) are sorted such that small values move right

5: UNTIL DONE

Mesh topology: Shearsort

Theorem: Shearsort sorts correctly *n* values in $\sqrt{n}(\log(n)+1)$ time in snake-like order.

PROOF: Thanks to Lemma 1, we can consider a mesh of 0's and 1's. We call a row **clean** if it contains only 0's or only 1's, **dirty** otherwise. Initially all rows can be dirty. The rows can be divided in 3 regions: the north, with only clean 0's rows, the center, with dirty rows, and the south, with only clean 1's rows.

After an odd phase, let us consider a pair of dirty rows, which will be sorted in opposite direction, as follows:

00000…11111

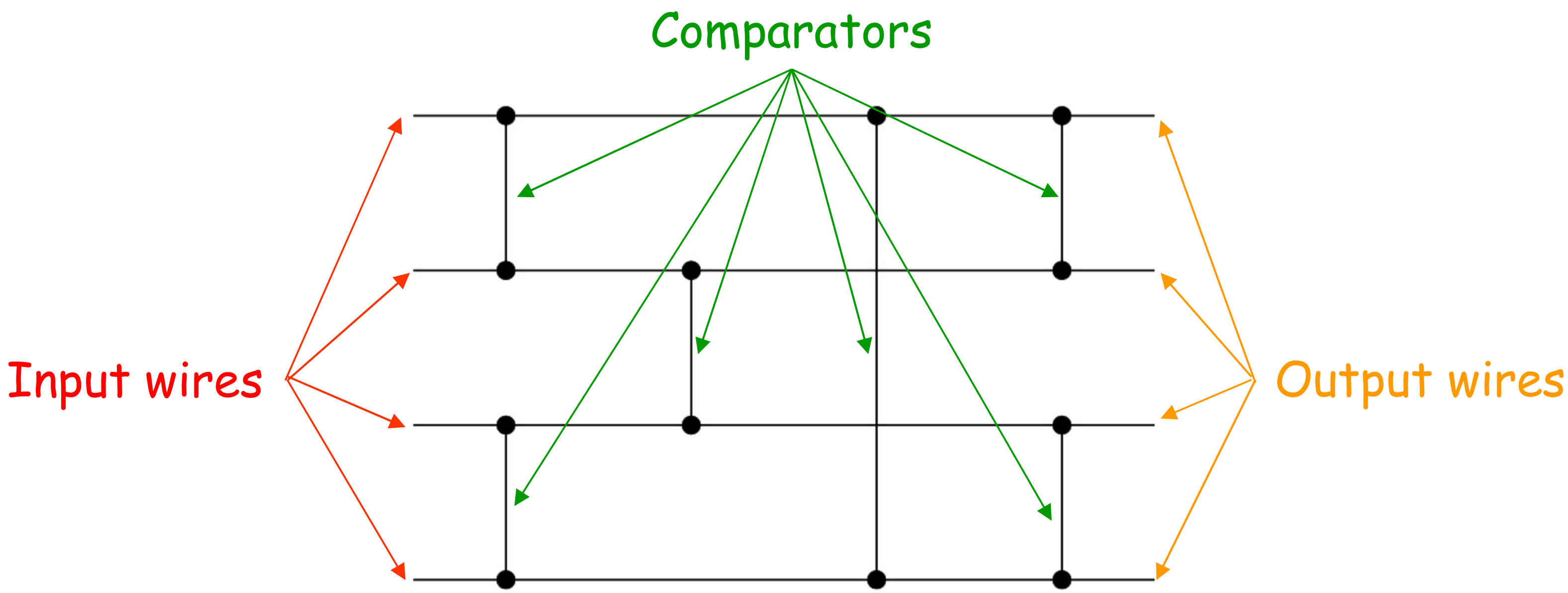
11111…00000

There are 3 possible cases: #0's>#1's, #0's=#1's, #0's<#1's. After column sorting, we are left with 1 clean row (or 2 in case 2) which moves to the correct region.

Each iteration halves the number of dirty rows, so the sorting requires log(*n*) phases, and in the end only 1 dirty row remains, which is sorted with the last row sorting phase. Each phase requires at most $m=\sqrt{n}$ operations, so the total time is $\sqrt{n}(\log(n)+1)$.

Sorting networks

We now propose a better topology, which is also used in many real application such as p2p networks. A **sorting network** is a set of **input wires**, **comparators** and **output wires**, that ensures that input values will be sorted on the output wires. A comparator takes two inputs x and y and returns two outputs x' and y' such that $x'=\min(x, y)$ and $y'=\max(x, y)$. In the following picture, wires are horizontal lines and comparators are vertical lines.



A sorting network

Some definitions

Width

The **width** of a comparison network is the number of wires.

Depth

- The **depth** of an input wire is 0.
- The **depth** of a comparator is the maximum depth of its input wires plus 1.
- The **depth** of an output wire of a comparator is the depth of the latter.
- The **depth** of a comparison network is the maximum depth of an output wire.

Bitonic sequence

A **bitonic sequence** is a sequence of numbers that first monotonically increases, then monotonically decreases, or vice versa.

Example:

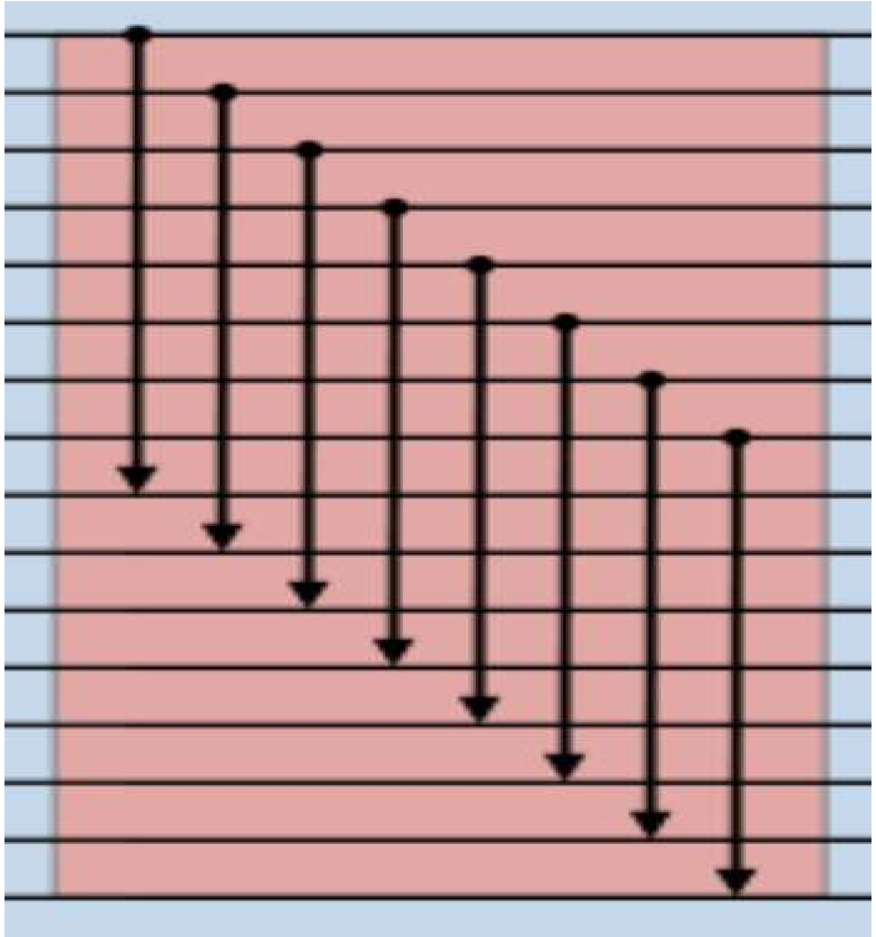
$\langle 1, 4, 5, 9, 7, 6, 2 \rangle$ is a bitonic sequence.

$\langle 2, 5, 4, 7 \rangle$ is not a bitonic sequence.

Remark: by using Lemma 1, a bitonic sequence has the form $0^i 1^j 0^k$ or $1^i 0^j 1^k$, with $i, j, k \geq 0$.

Half Cleaner

A **half cleaner** is a comparison network of depth 1, where we compare wire i with wire $i+n/2$, for $i=1,\ldots,n/2$ (assume n even).



Half cleaner of width 16

Half Cleaner

LEMMA 2

If we feed a bitonic sequence into a half cleaner, it cleans (makes all 0's or all 1's) either the upper or the lower half of the n wires. The other half is bitonic.

PROOF

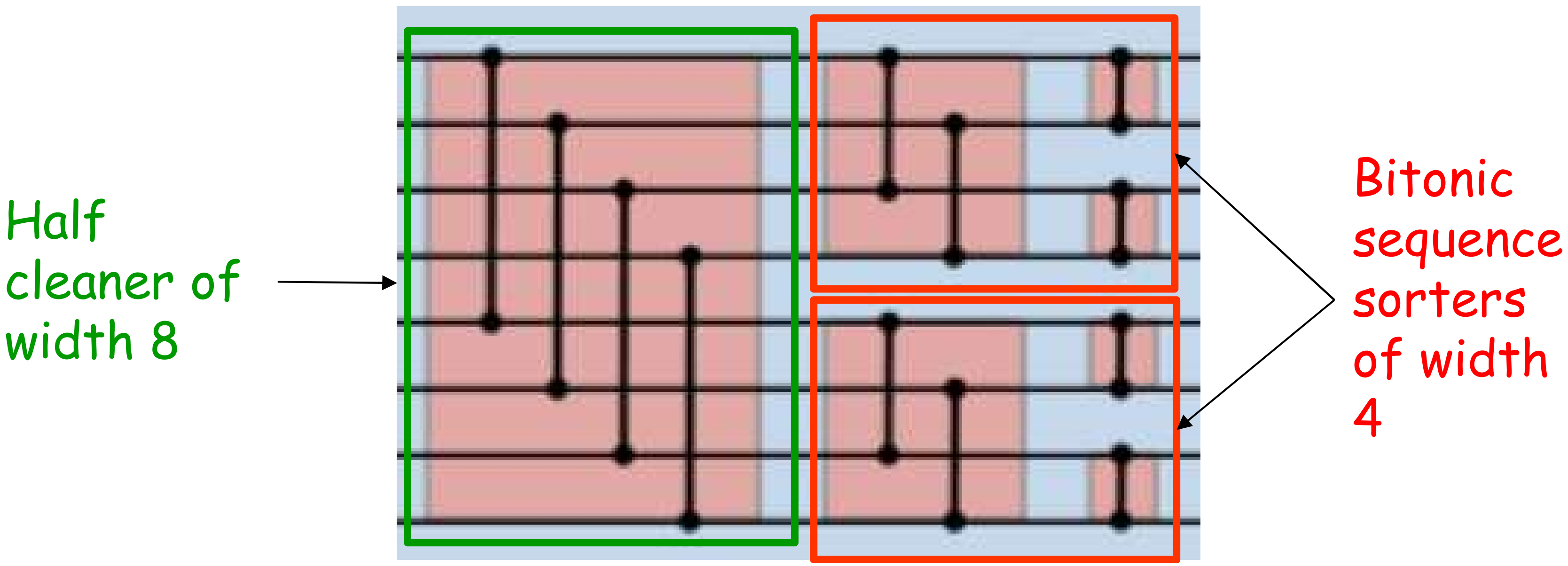
Assume the input is of the form $0^i1^j0^k$, for $i,j,k \geq 0$. If the midpoint falls into the 0's the input is already clean/bitonic and it will stay so. Otherwise, the half cleaner acts as Shearsort with 2 adjacent rows, as we previously proved.

The case $1^i0^j1^k$ is symmetric.

QED

Bitonic Sequence Sorter

A **bitonic sequence sorter** of width n ($n=2^k$) is made up of a half cleaner of width n, followed by 2 bitonic sequence sorters of width $n/2$. A bitonic sequence sorter of width 1 is empty.



Bitonic sequence sorter of width 8

Bitonic Sequence Sorter

LEMMA 3

A bitonic sequence sorter of width *n* sorts bitonic sequences of length *n*. It has depth log *n*.

PROOF

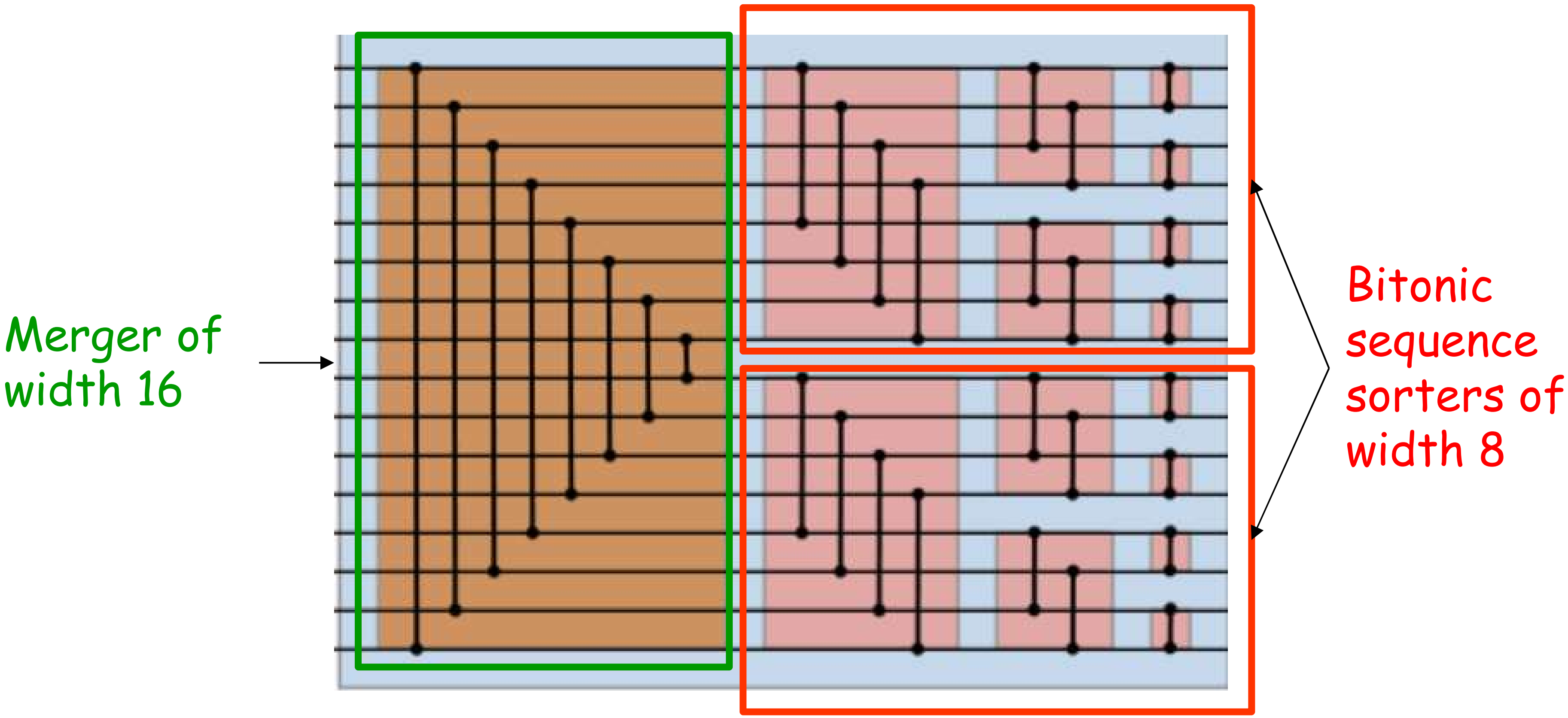
All the components of a bitonic sequence sorter are half cleaners, that correctly sort bitonic sequences. At each step, the original sequence is divided in 2 parts, which are still bitonic and are fed in 2 half cleaners. The claim follows by proof of Lemma 2.

The bitonic sequence sorter has depth log *n*, since the sequence is halved at every step.

QED

Merging Network

A **merging network** of width n is a merger of width n followed by two bitonic sequence sorters of width $n/2$. A **merger** is a depth-one network where we compare wire i with wire $n-i+1$, for $i=1,\dots,n/2$.



Merging network of width 16

Merging Network

LEMMA 4

A merging network of width *n* merges two sorted input sequences of length *n*/2 each into one sorted sequence of length n.

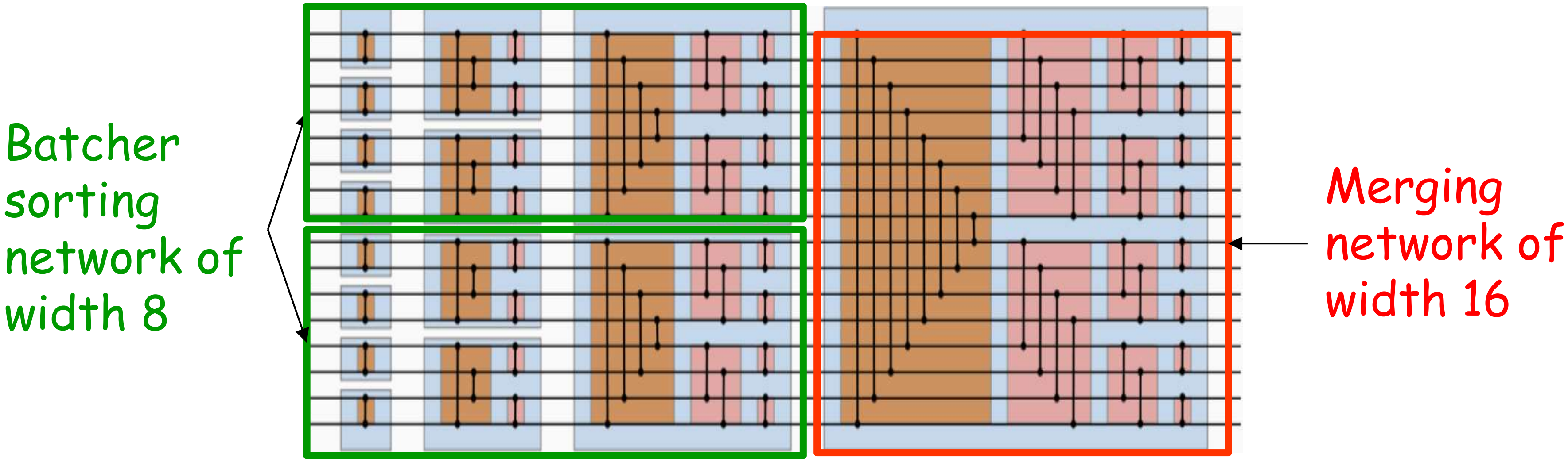
PROOF

The intuition follows directly from proofs of Lemmas 2 and 3. After the merger step, one half of the sequence is clean and the other is bitonic, so it will be correctly sorted by the bitonic sequence sorter.

QED

Batcher's "Bitonic" Sorting Network

A **batcher sorting network** of width *n* consists of two batcher sorting networks of width *n*/2 followed by a merging network of width *n*. A batcher sorting network of width 1 is empty.



Batcher sorting network of width 16

Batcher's "Bitonic" Sorting Network

THEOREM

A sorting network sorts an arbitrary sequence of n values. It has depth $O(\log^2 n)$.

PROOF

At recursive stage k (k=1,2,…,log n) we merge 2^k sorted sequence into 2^{k-1} sorted sequences. The depth d(n) of the sorting network of level n is the depth of a sorting network of level $n/2$ plus the depth m(n) of a merging network.

$d(n) = d(n/2) + m(n)$

$d(1) = 0$ (the sorter is empty)

Since a merging network of width n has the same depth as a bitonic sequence sorter of width n, we know by Lemma 3 that $m(n)=\log(n)$.

This gives a recursive formula for d(n) which solves to $d(n)=\frac{1}{2}\log^2(n)+\frac{1}{2}\log(n)$ (derived from case 2 of Master Theorem).

QED

Concluding remarks

Simulating a Batcher's sorting network on an ordinary sequential computer takes time $O(n \log^2 n)$.

As you know, there exist sequential sorting algorithms that sort in asymptotically optimal time $O(n \log n)$. So, is there a sorting network with depth $O(\log n)$?

Yes, indeed in 1983 Ajtai, Komlos and Szemeredi proposed a $O(\log n)$ sorting network, but the constant hidden in big-O is too large to be practical.

THANK YOU!