Vorlesung Datenstrukturen und Algorithmen
Letzte Vorlesung 2018

Felix Friedrich, 30.5.2018

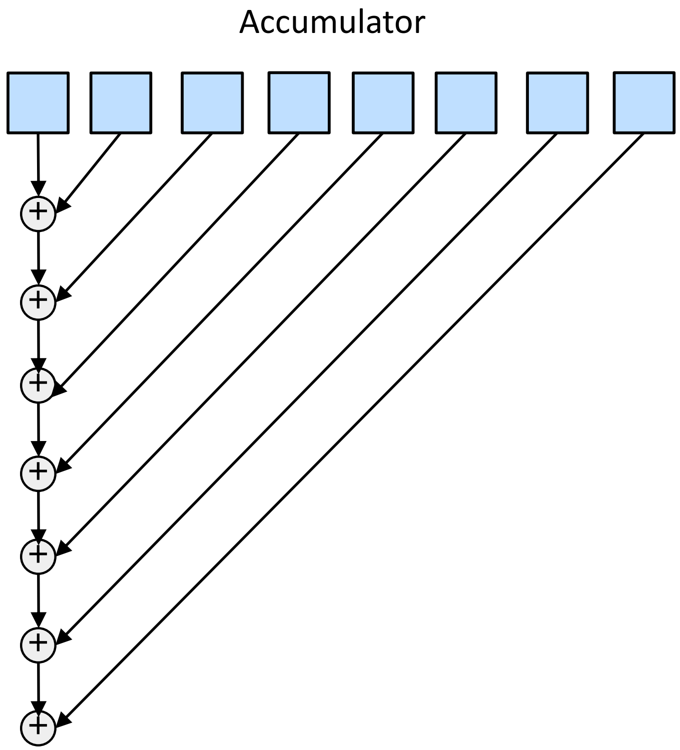
Map/Reduce

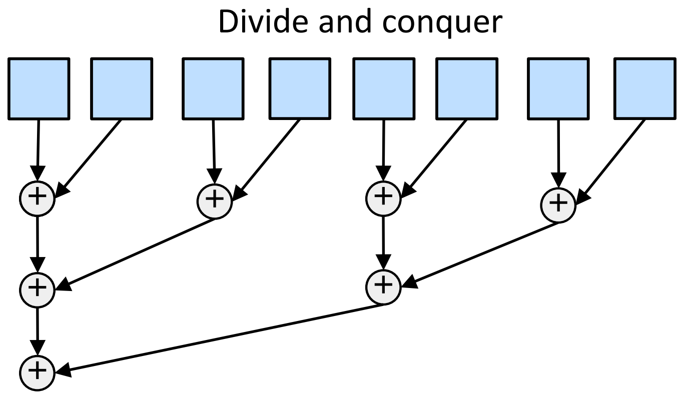
Sorting Networks

Prüfung

MAP AND REDUCE AND MAP/REDUCE

Summing a Vector



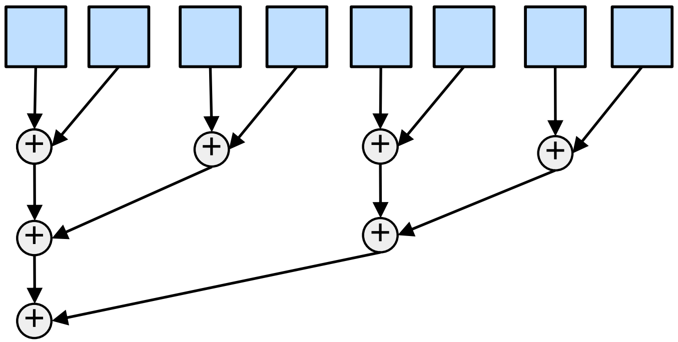


Q: Why is the result the same?

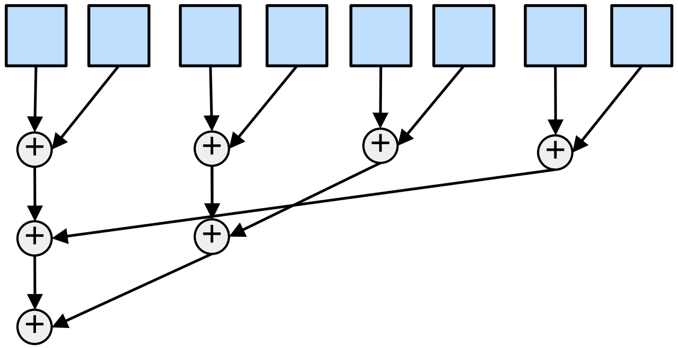
A: associativity:
(a+b) + c = a + (b+c)

Summing a Vector

Divide and conquer



Is this correct?



Only if the operation is commutative:

$$a + b = b + a$$

Reductions

Simple examples: sum, max

Reductions over programmer-defined operations

- operation properties (associativity / commutativity) define the correct executions
- supported in most parallel languages / frameworks
- powerful construct

C++ Reduction

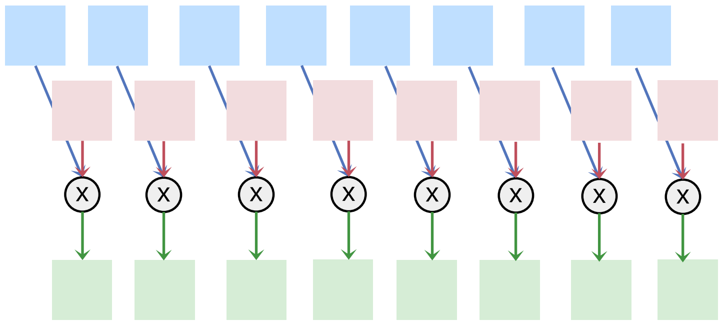
- std::accumulate (requires associativity)
- std::reduce (requires commutativity, from C++17, can specify execution policy)

```
std::vector<double> v;
...
double result = std::accumulate(
     v.begin(), v.end(), 0.0,
     [](double a, double b){return a + b;}
);
```

Elementwise Multiplication

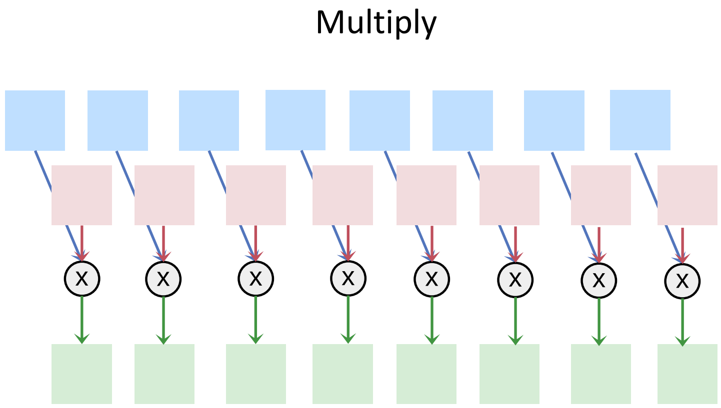
Map

Multiply

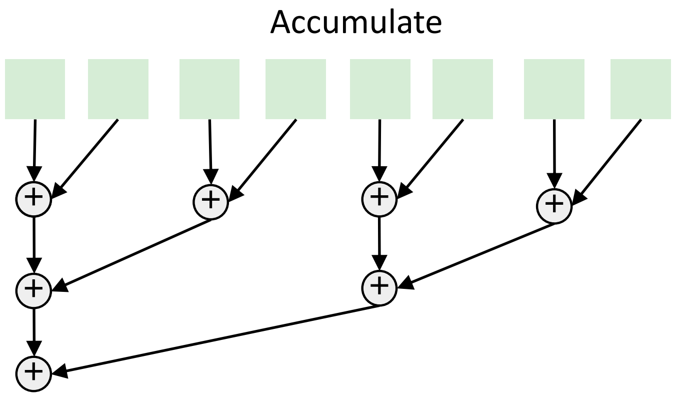


Scalar Product

Map



Reduce



C++ Scalar Product (map + reduce)

```
// example data
std::vector<double> v1(1024,0.5);
auto v2 = v1;
std::vector<double> result(1024);
// map
std::transform(v1.begin(), v1.end(), v2.begin(), result.begin(),
                     [](double a, double b){return a*b;});
// reduce
double value = std::accumulate(result.begin(), result.end(), 0.0); // = 256
```

Map & Reduce = MapReduce

Combination of two parallelisation patterns

$$\text{result} = f(\text{in}_1) \oplus f(\text{in}_2) \oplus f(\text{in}_3) \oplus f(\text{in}_4)$$

f = map

$\oplus$ = reduce (associative)

Examples: numerical reduction, word count in document, (word, document) list, maximal temperature per month over 50 years (etc.)

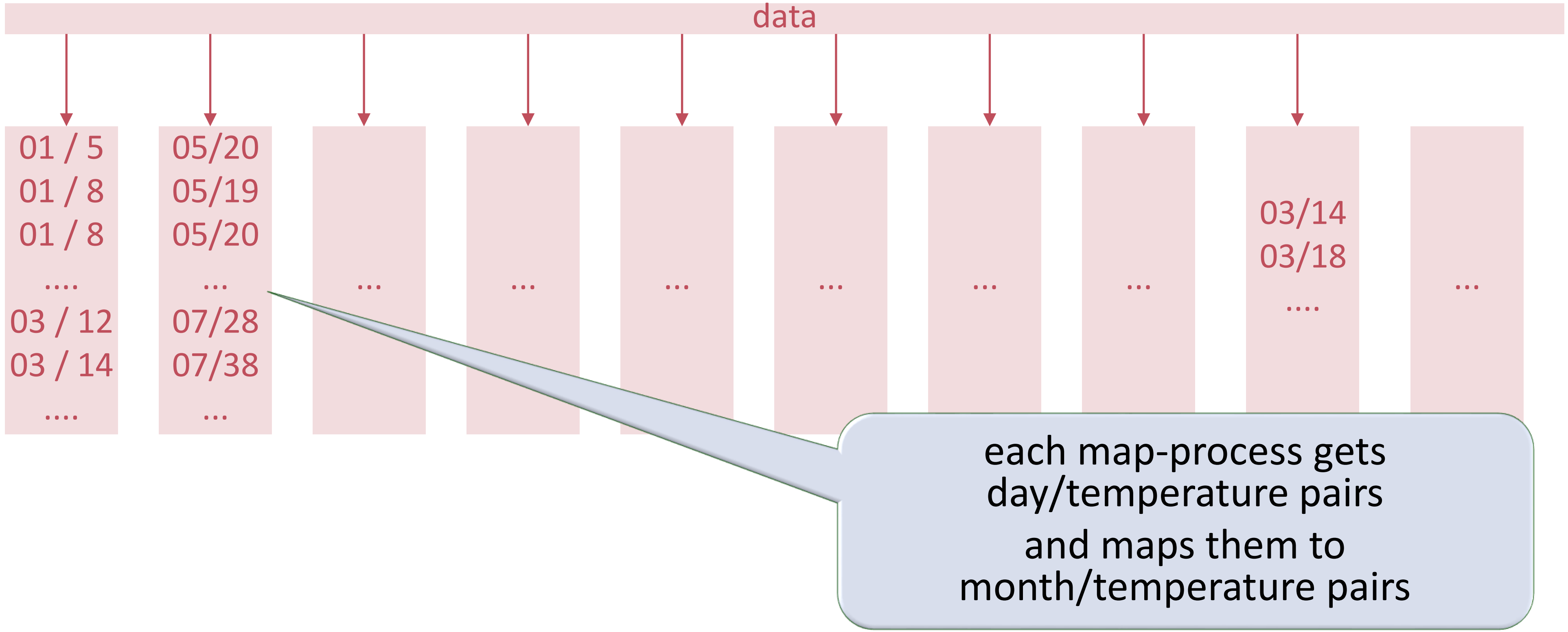
Motivating Example

Maximal Temperature per Month for 50 years

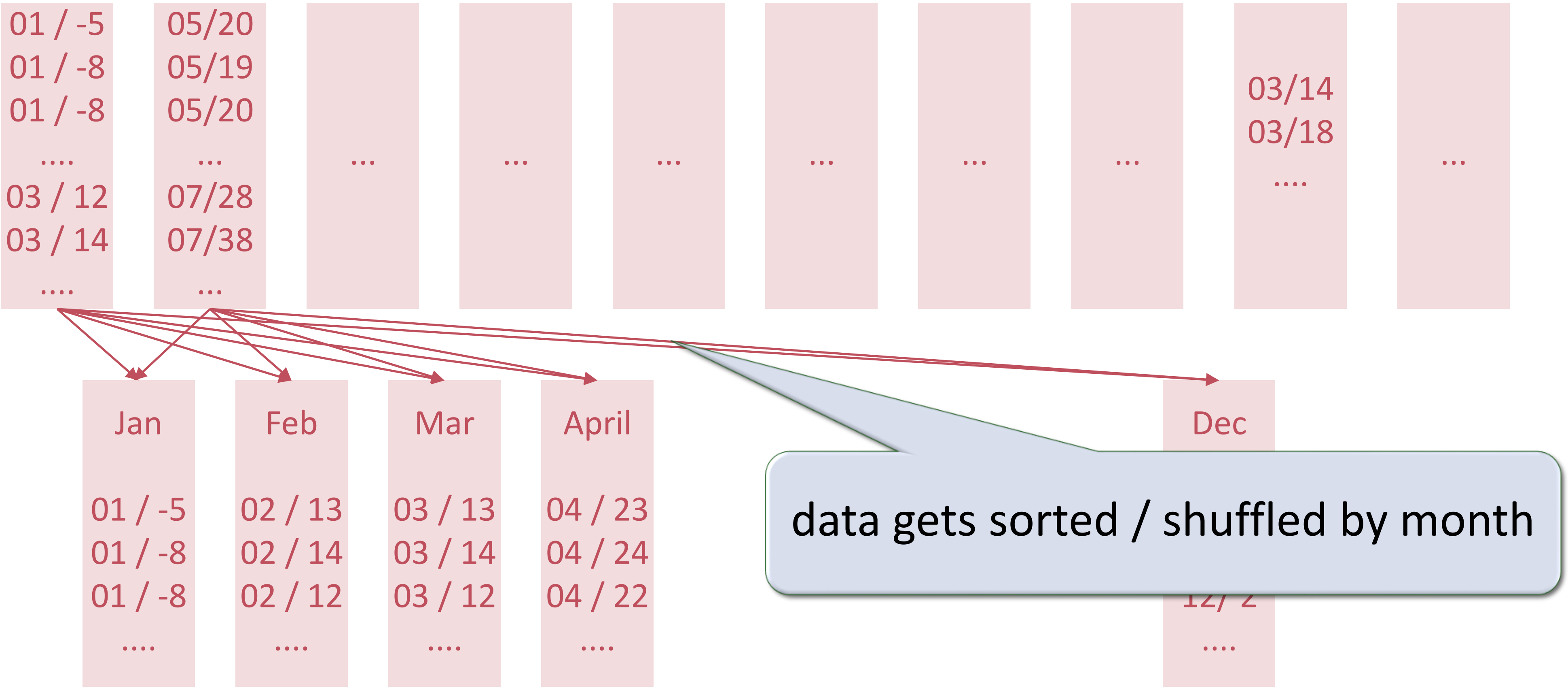
- Input: 50 * 365 Days / Temperature pairs
- Output: 12 Months / Max Temperature pairs

Assume we (you and me) had to do this together.
How would we distribute the work?
What is the generic model?
How would we be ideally prepared for different reductions (min, max, avg)?

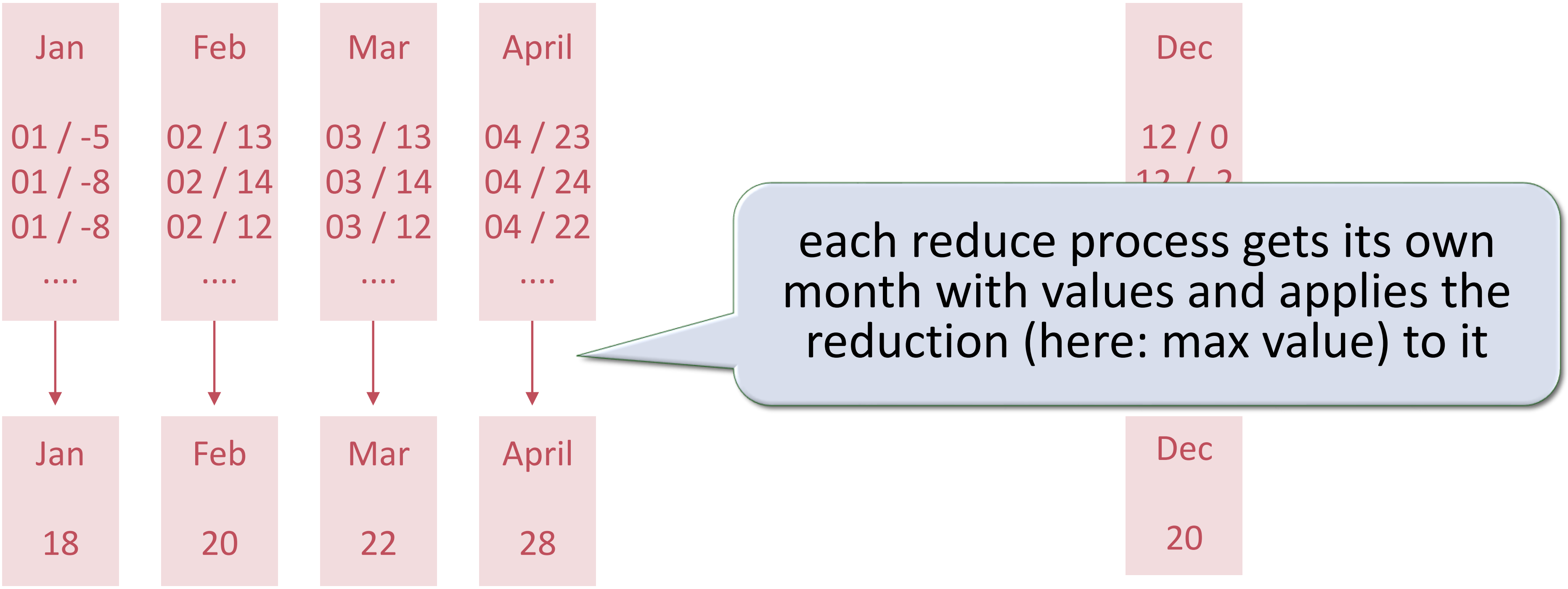
Maximal Temperature per Month: Map



Maximal Temperature per Month: Shuffle



Maximal Temperature per Month: Reduce



Map/Reduce

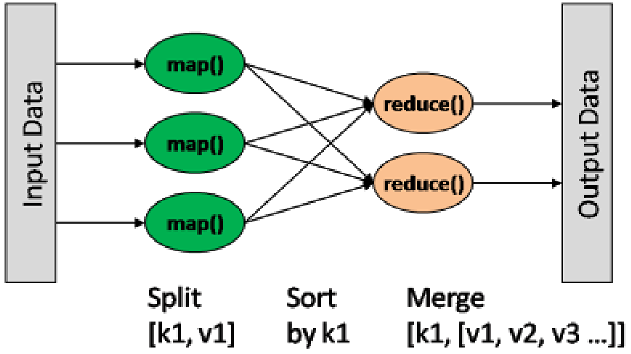
A strategy for implementing parallel algorithms.

- *map*: A master worker takes the problem input, divides it into smaller sub-problems, and distributes the sub-problems to workers (threads).
- *reduce*: The master worker collects sub-solutions from the workers and combines them in some way to produce the overall answer.

Map/Reduce

Frameworks and tools have been written to perform map/reduce.

- MapReduce framework by Google
- Hadoop framework by Yahoo!
- related to the ideas of *Big Data* and *Cloud Computing*
- also related to *functional programming* (and actually not that new) and available with the Streams concept in Java (>=8)



- Map and reduce are user-supplied plug-ins, the rest is provided by the frameworks.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (January 2008), 107-113. DOI=10.1145/1327452.1327492 http://doi.acm.org/10.1145/1327452.1327492

MapReduce on Clusters

You may have heard of Google's "map/reduce" or Amazon's Hadoop

Idea: Perform maps/reduces on data using many machines

- The system takes care of distributing the data and managing fault tolerance
- You just write code to **map** one element (key-value part) and **reduce** elements (key-value pairs) to a combined result

Separates how to do recursive divide-and-conquer from what computation to perform

- Old idea in higher-order functional programming transferred to large-scale distributed computing

Example

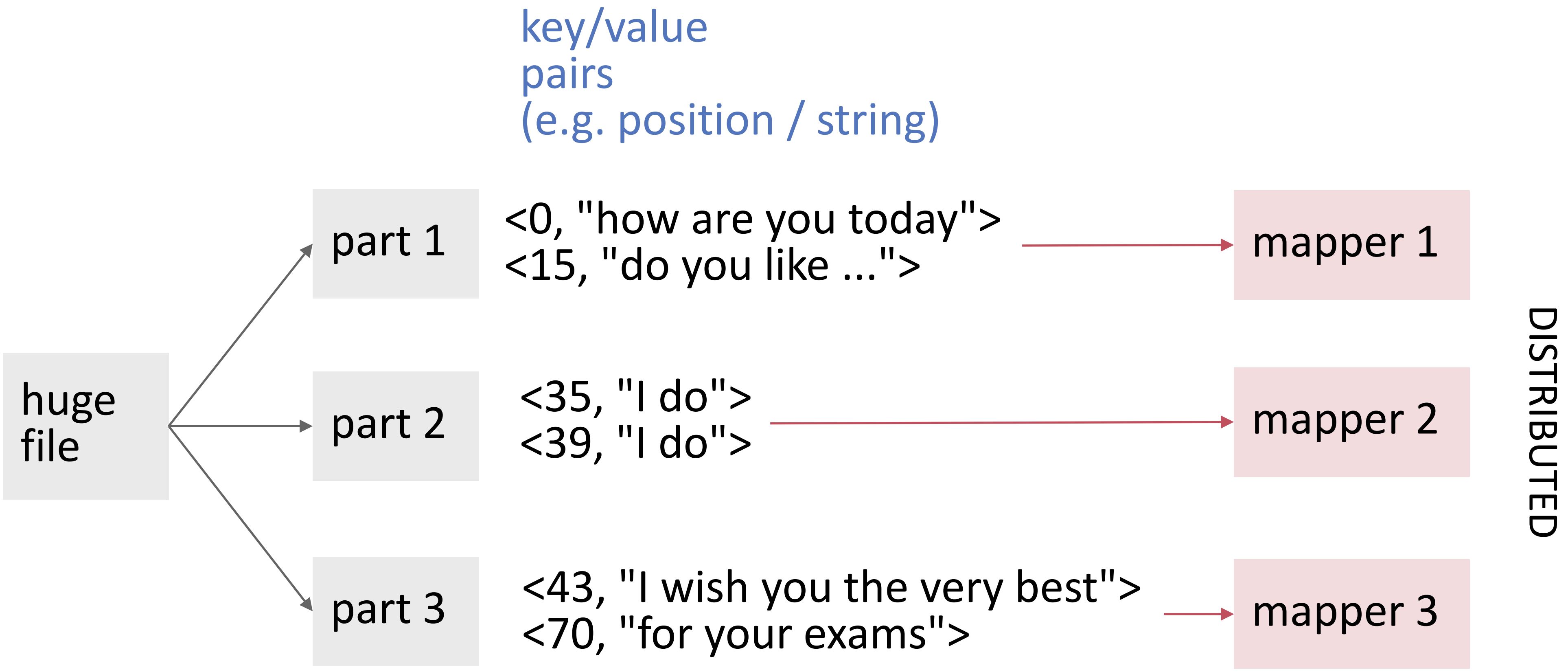
Count word occurences in a very large file

File =

GBytes

```
how are you today
do you like the weather outside
I do
I do
I wish you the very best
for your exams.
```

Mappers


key/value
pairs
(e.g. position / string)
huge
file
part 1
part 2
part 3
<0, "how are you today">
<15, "do you like ...">
<35, "I do">
<39, "I do">
<43, "I wish you the very best">
<70, "for your exams">
mapper 1
mapper 2
mapper 3
DISTRIBUTED

Mappers

input

key/value
pairs
(e.g. position / string)

output

key/value
pairs
(word, count)

<0, "how are you today">
<15, "do you like ...">

→ mapper 1 →

<"how",1>
<"are",1>
<"you",1>
...

<35, "I do">
<39, "I do">

→ mapper 2 →

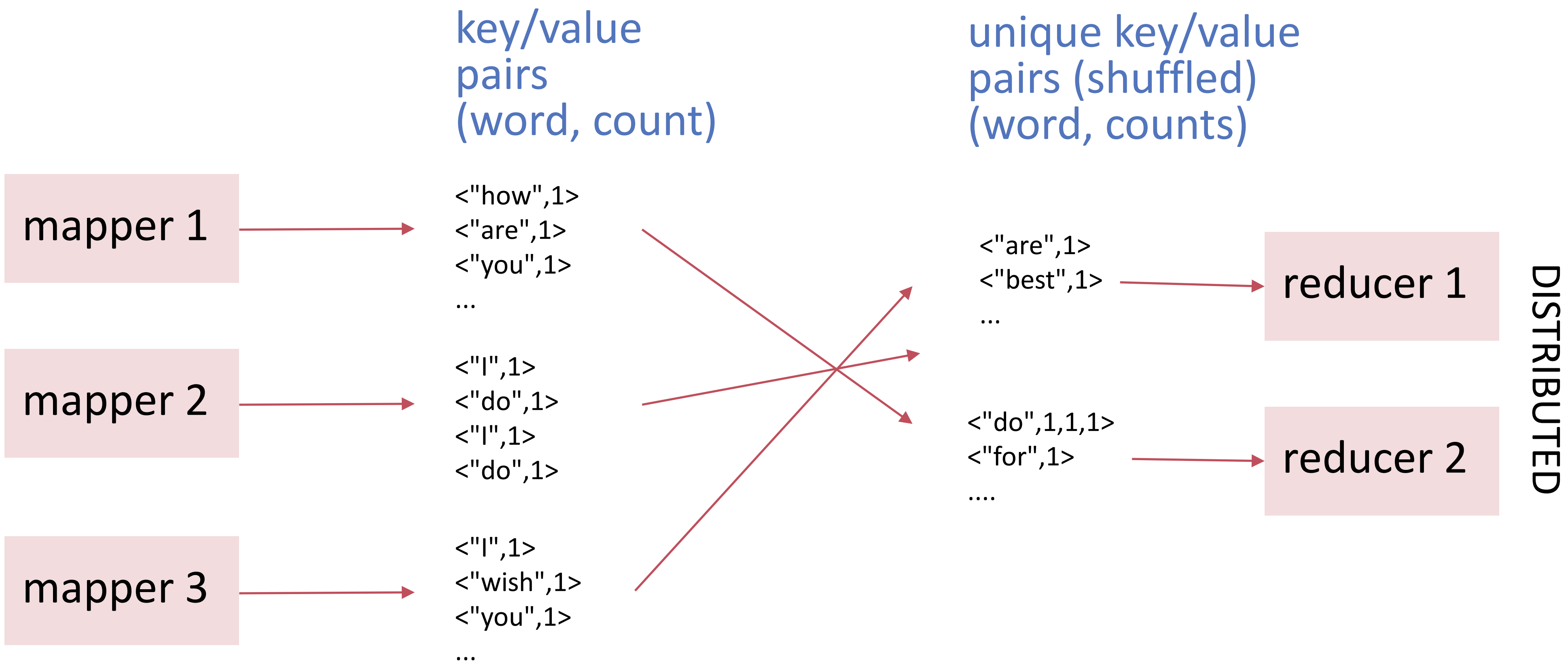
<"I",1>
<"do",1>
<"I",1>
<"do",1>

<43, "I wish you the very best">
<70, "for your exams">

→ mapper 3 →

<"I",1>
<"wish",1>
<"you",1>
...

Shuffle / Sort



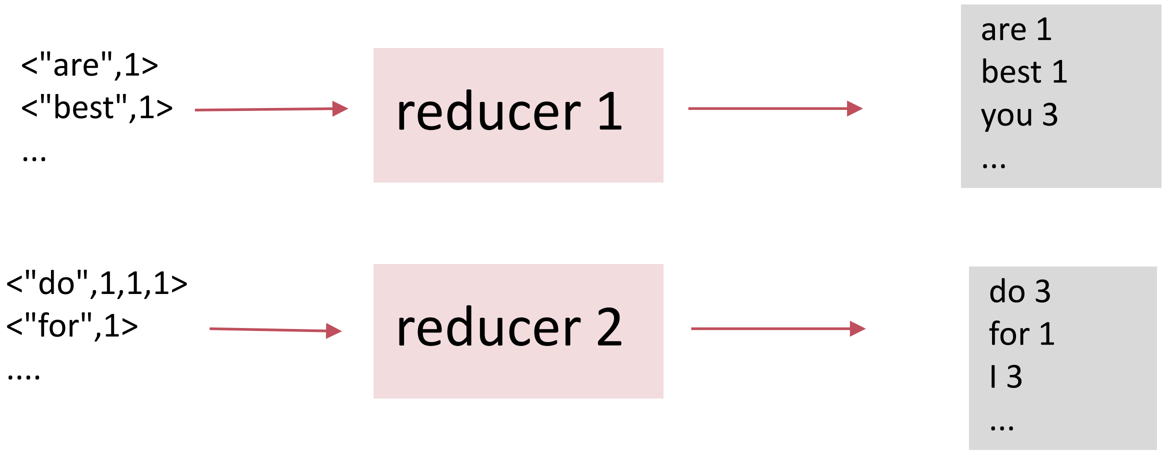
Reduce

input

unique key/value
pairs (shuffled)
(word, counts)

output

target file(s)



SORTING NETWORKS

Lower bound on sorting

Horrible algorithms: $\Omega(n^2)$

Bogo Sort
Stooge Sort

Simple algorithms: $O(n^2)$

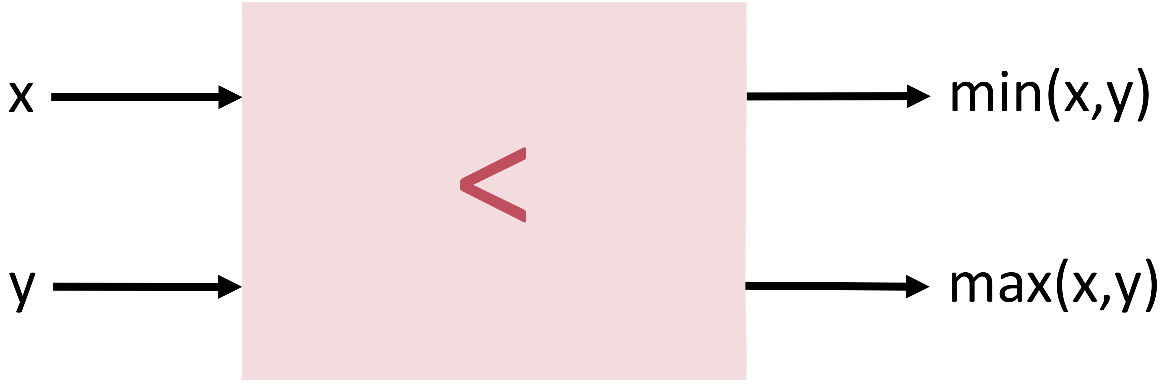
Insertion sort
Selection sort
Bubble Sort
Shell sort
…

Fancier algorithms: $O(n \log n)$

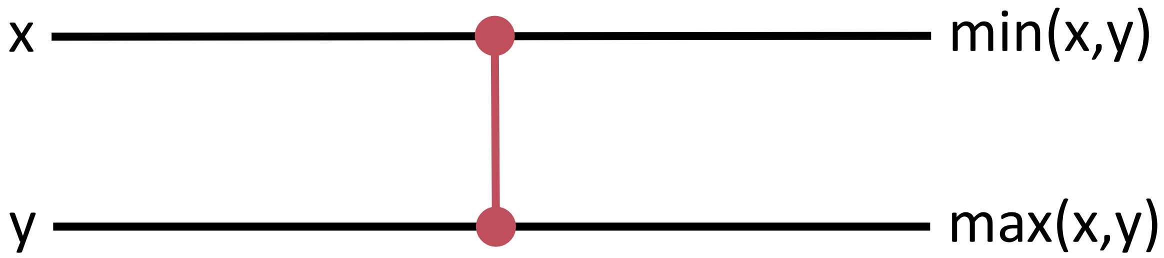
Heap sort
Merge sort
Quick sort (avg)
…

Comparison lower bound: $\Omega(n \log n)$

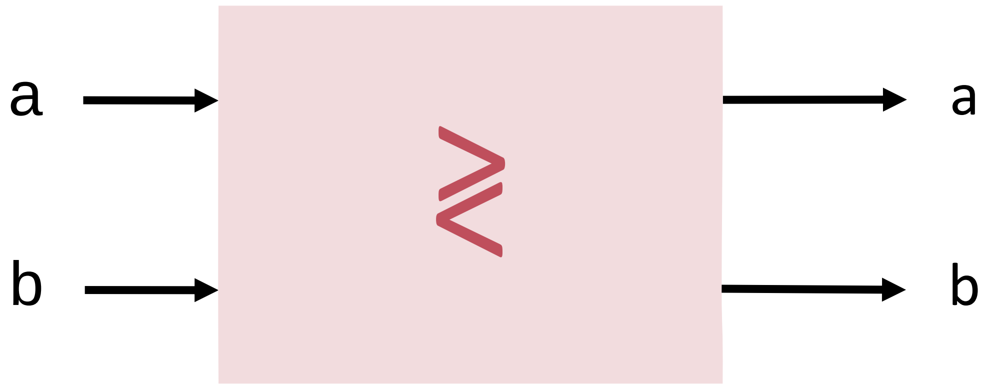
Comparator



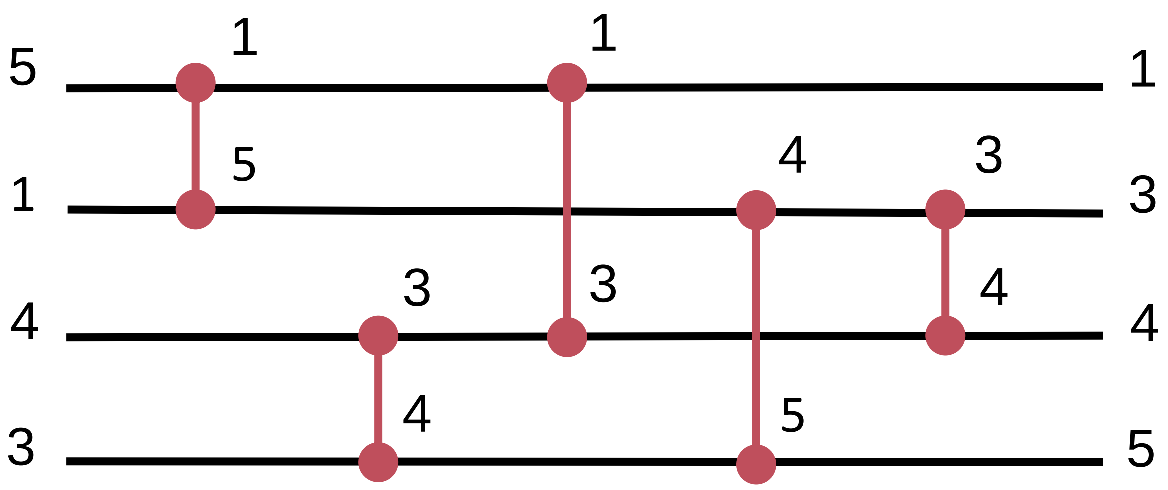
shorter notation:



```
void compare(int&a, int&b, boolean dir) {
   if (dir==(a,b)){
      std::swap(a,b);
   }
}
```

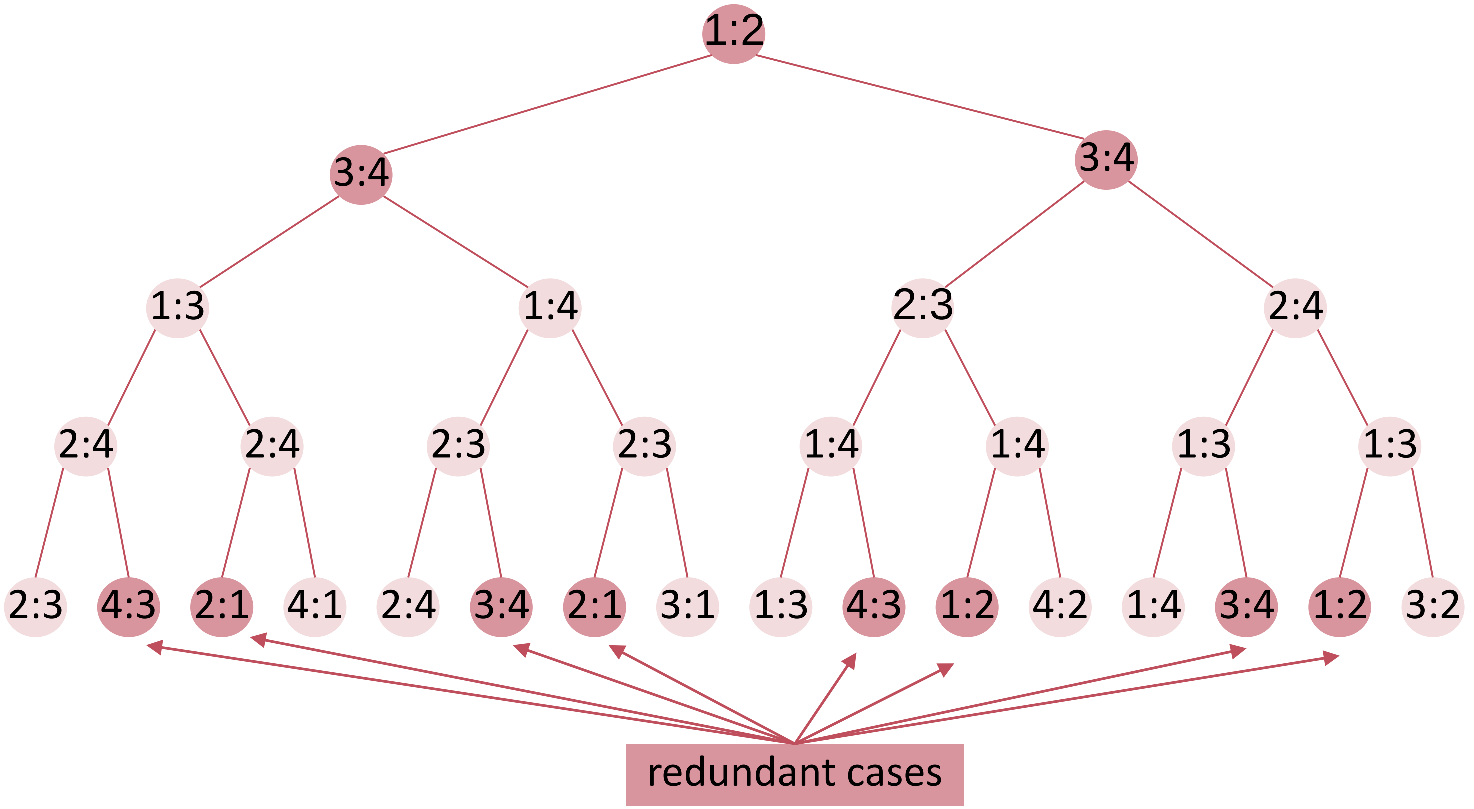


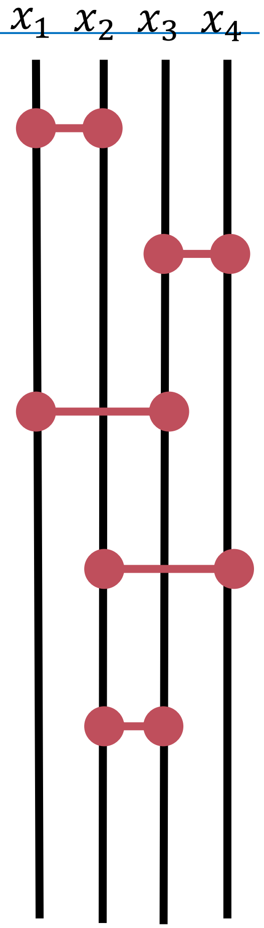
Sorting Networks



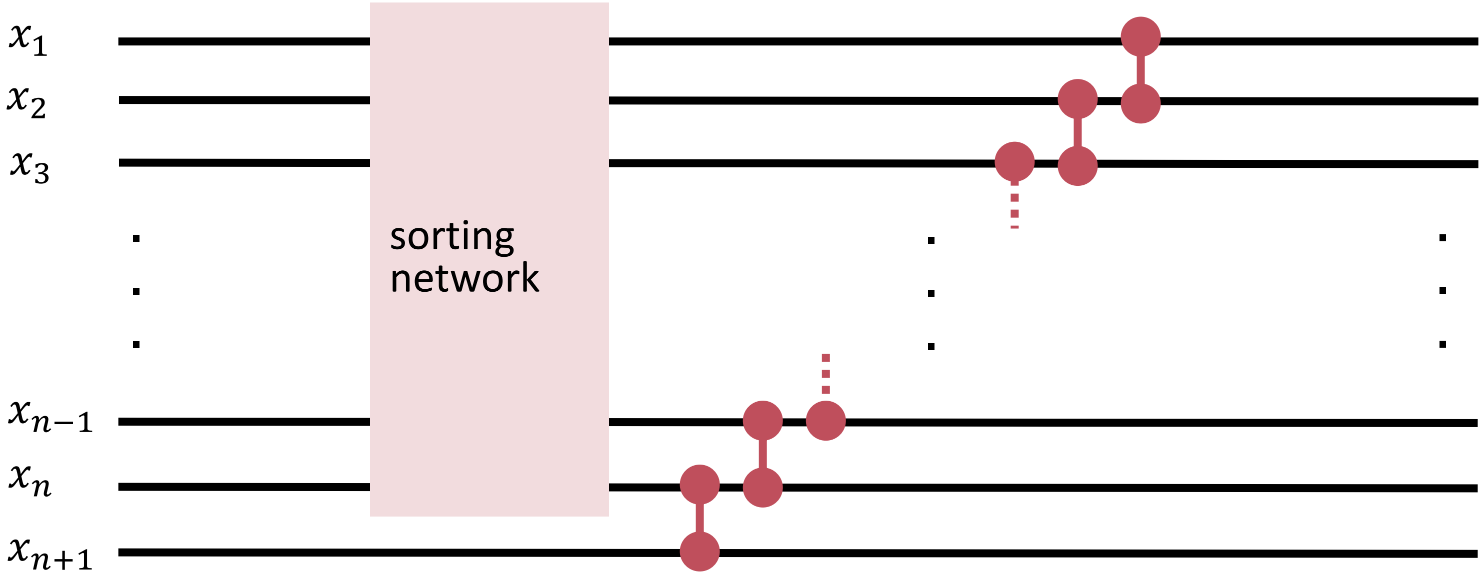
Sorting Networks are Oblivious (and Redundant)

Oblivious comparison tree

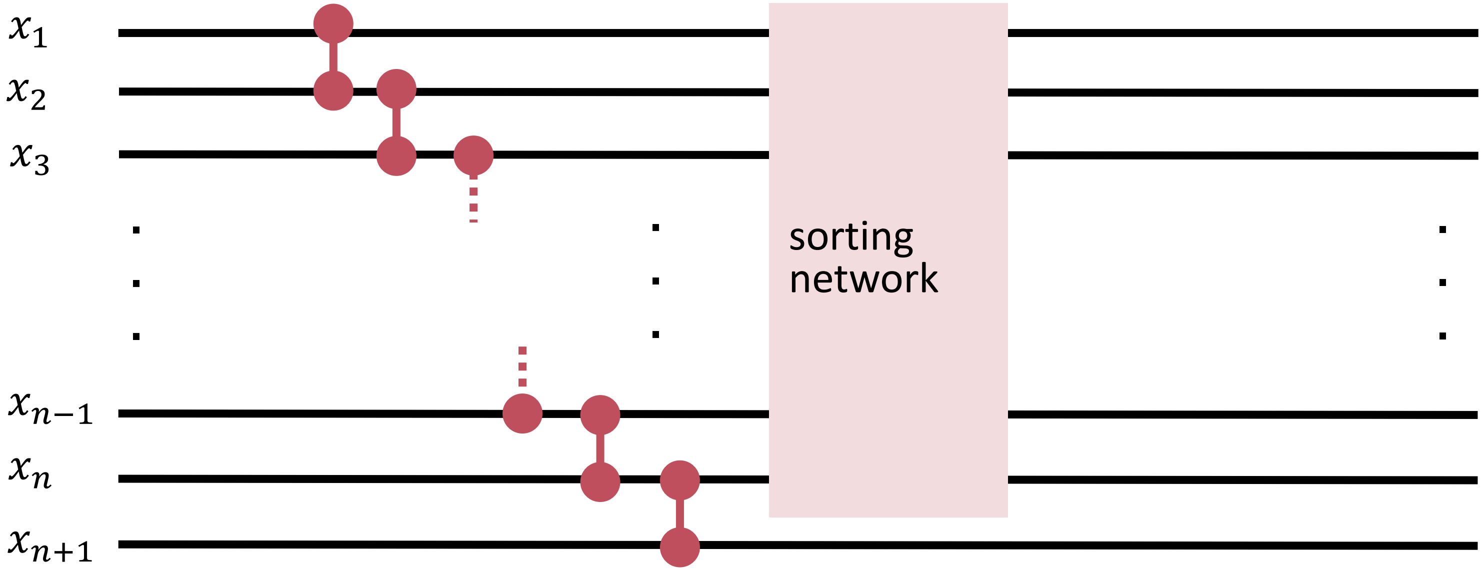




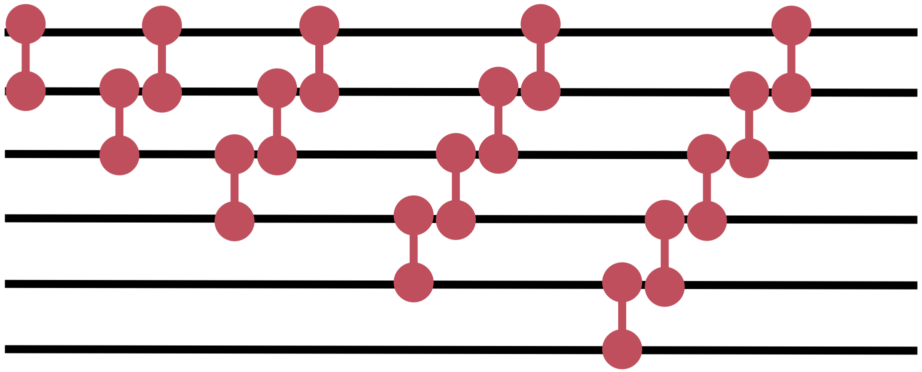
Recursive Construction : Insertion



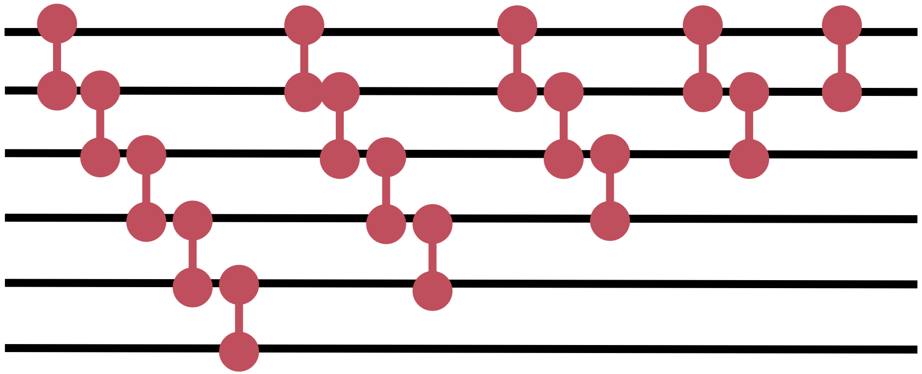
Recursive Construction: Selection



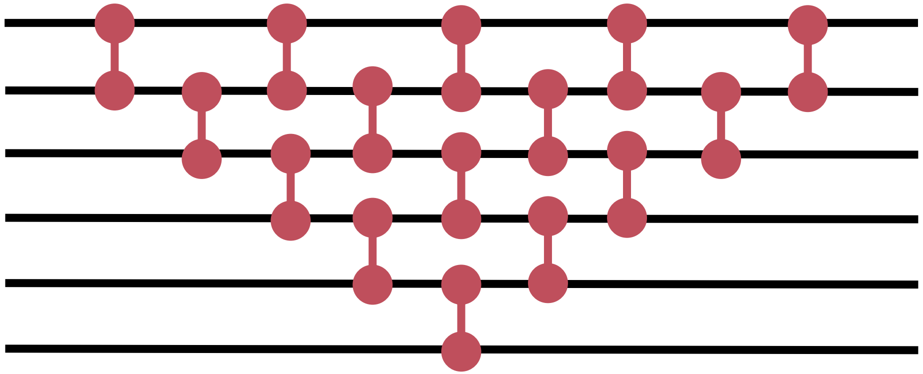
Applied recursively..



insertion sort



bubble sort

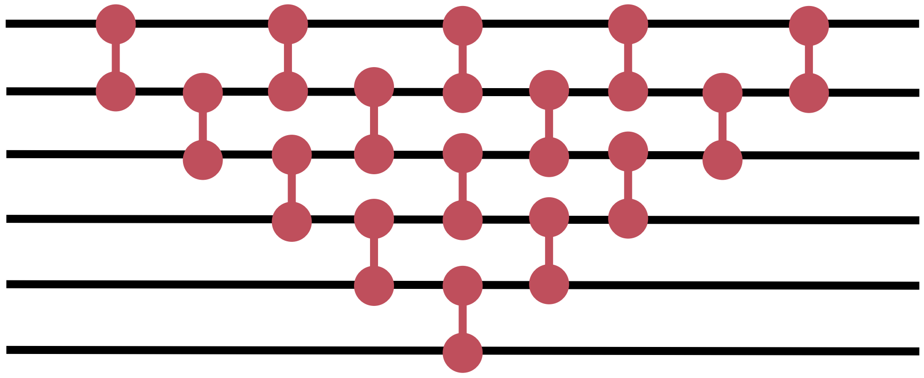


with parallelism: insertion sort = bubble sort !

Question

How many steps does a computer with infinite number of processors (comparators) require in order to sort using parallel bubble sort?

Answer: 2n – 3
Can this be improved ?



How many comparisons ?

Answer: (n-1) n/2

How many comparators are required (at a time)?

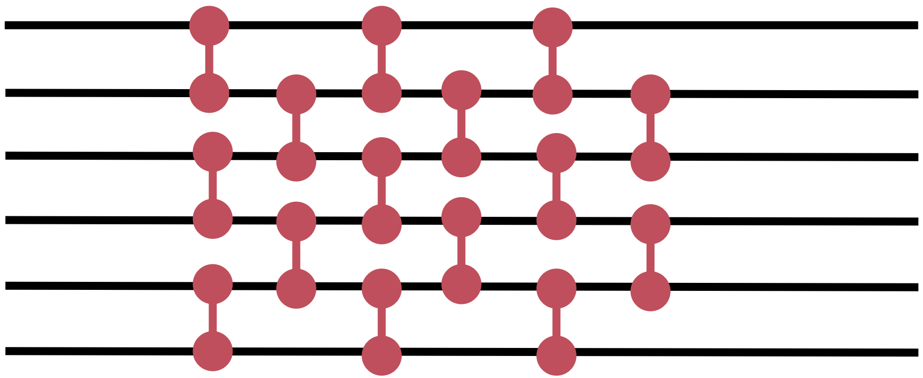
Answer: n/2
Reusable comparators: n-1

Improving parallel Bubble Sort

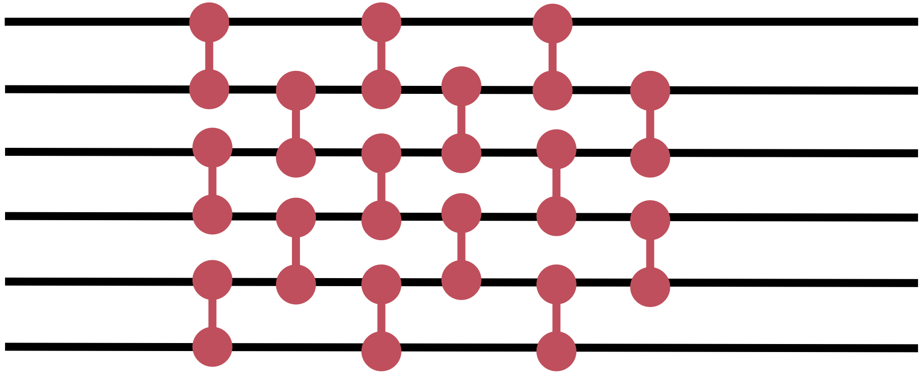
Odd-Even Transposition Sort:

0	9	↔	8		2	↔	7		3	↔	1		5	↔	6		4
1	8		9	↔	2		7	↔	1		3	↔	5		6	↔	4
2	8	↔	2		9	↔	1		7	↔	3		5	↔	4		6
3	2		8	↔	1		9	↔	3		7	↔	4		5	↔	6
4	2	↔	1		8	↔	3		9	↔	4		7	↔	5		6
5	1		2	↔	3		8	↔	4		9	↔	5		7	↔	6
6	1	↔	2		3	↔	4		8	↔	5		9	↔	6		7
7	1		2	↔	3		4	↔	5		8	↔	6		9	↔	7
8	1	↔	2		3	↔	4		5	↔	6		8	↔	7		9
	1		2		3		4		5		6		7		8		9

```
void oddEvenTranspositionSort(std::vector<T>& v, boolean dir) {
   for (int i = 0; i<v.size(); ++i){
      for (int j = i % 2; j+1<n; j+=2)
         compare(v[i],v[j],dir);
   }
}
```



Improvement?



Same number of comparators (at a time)

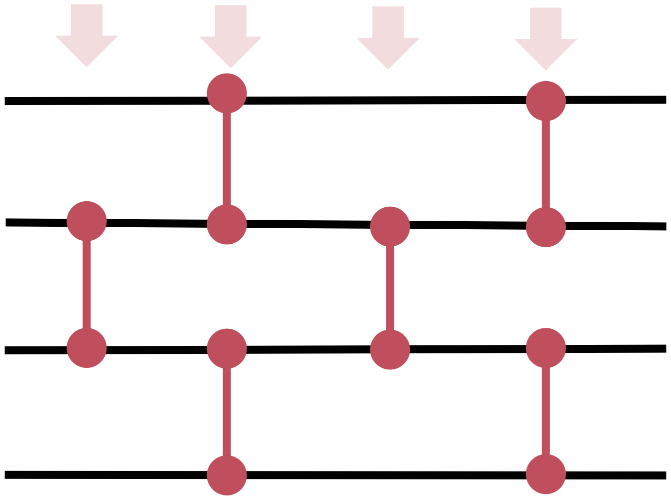
Same number of comparisons

But less parallel steps (depth): n

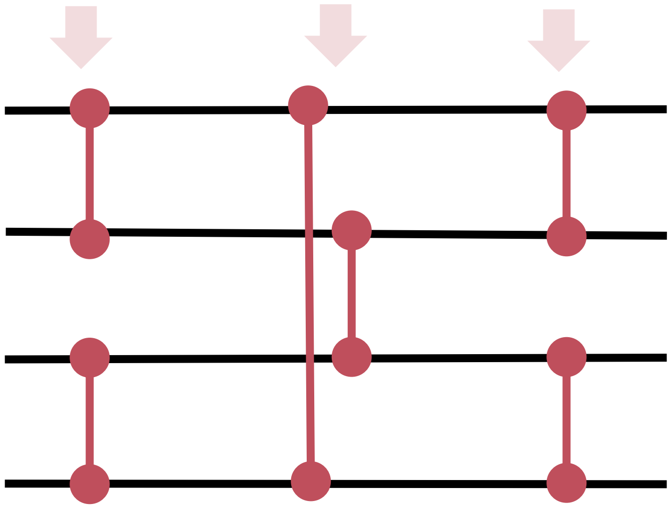
In a massively parallel setup, bubble sort is thus not too bad.

But it can go better...

Parallel sorting



depth = 4



depth = 3

Prove that the two networks above sort four numbers. Easy?

Zero-one-principle

Theorem: If a network with n input lines sorts all 2^n sequences of 0s and 1s into non-decreasing order, it will sort any arbitrary sequence of n numbers in nondecreasing order.

Proof

Argue: If x is sorted by a network N
then also any monotonic function of x.

1	8	20	30	5	9	→	1	5	8	9	20	30
-1	4	10	99	2	9	→	-1	2	4	9	10	99

Show: If x is **not** sorted by the network, **then** there is monotonic function f that maps x to 0s and 1s and **f(x) is not sorted** by the network

1	8	20	30	5	9	→	1	5	9	8	20	30
0	0	1	1	0	1	→	0	0	1	0	1	1

$$x \text{ not sorted by } N \Rightarrow \text{there is an } f(x) \in \{0,1\}^n \text{ not sorted by N}$$

$$\Leftrightarrow$$

$$f \text{ sorted by N for all } f \in \{0,1\}^n \Rightarrow x \text{ sorted by N for all x}$$

Proof

Assume a monotonic function $f(x)$ with $f(x) \leq f(y)$ whenever $x \leq y$ and a network N that sorts. Let N transform $(x_1, x_2, \ldots, x_n)$ into $(y_1, y_2, \ldots, y_n)$, then it also transforms $(f(x_1), f(x_2), \ldots, f(x_n))$ into $(f(y_1), f(y_2), \ldots, f(y_n))$.

All comparators must act in the same way for the $f(x_i)$ as they do for the x_i

Assume $y_i > y_{i+1}$ for some i, then consider the monotonic function

$$f(x) = \begin{cases} 0, if\ x < y_i \\ 1, if\ x \geq y_i \end{cases}$$

➔ N converts

$(f(x_1), f(x_2), \ldots, f(x_n))$ into $(f(y_1), f(y_2), \ldots f(y_i), f(y_{i+1}), \ldots, f(y_n))$

1 0

Bitonic Sort

Bitonic (Merge) Sort is a parallel algorithm for sorting

If enough processors are available, bitonic sort breaks the lower bound on sorting for comparison sort algorithm

Asymptotic Runtime of $O(n \log^2 n)$ (sequential execution)

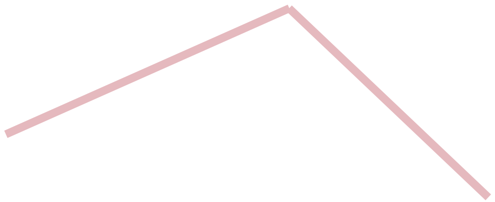
Very good asymptotic runtime in the parallel case (as we'll see below).

Worst = Average = Best case

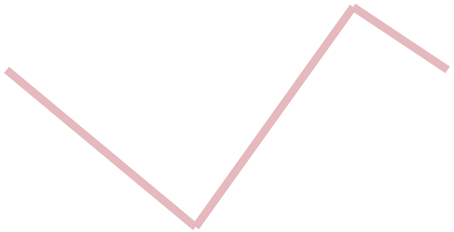
Bitonic

Sequence $(x_1, x_2, \ldots, x_n)$ is bitonic, if it can be circularly shifted such that it is first monotonically increasing and then monontonically decreasing.

(1, 2, 3, 4, 5, 3, 1, 0)



(4, 3, 2, 1, 2, 4, 6, 5)



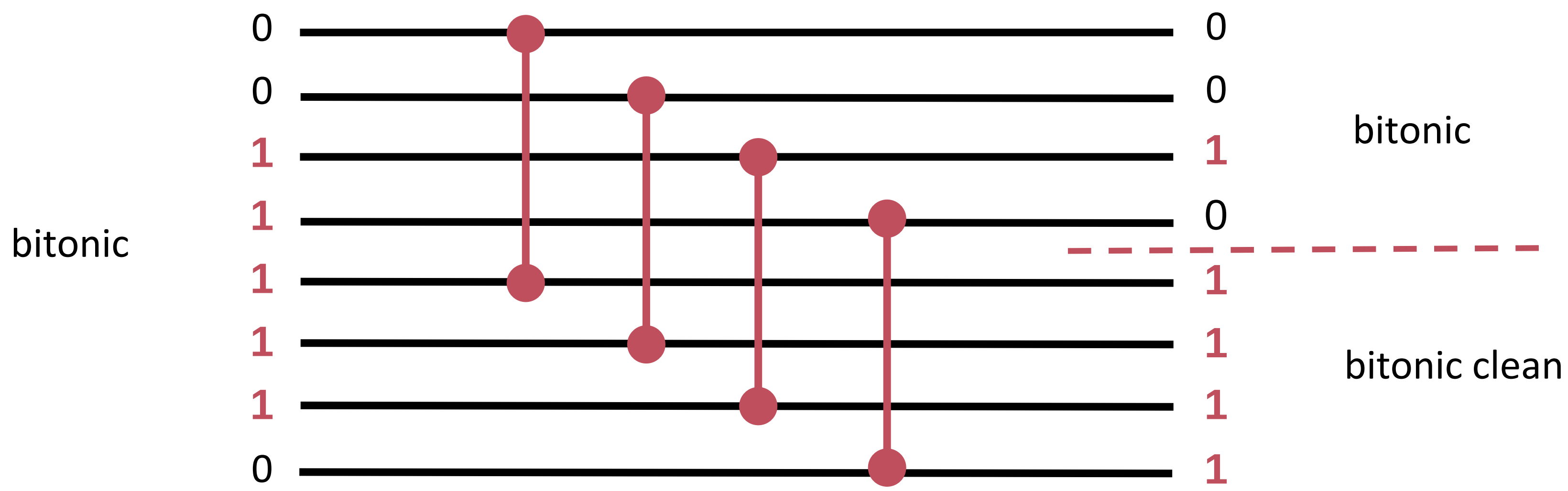
Bitonic 0-1 Sequences

$$0^i 1^j 0^k$$

$$1^i 0^j 1^k$$

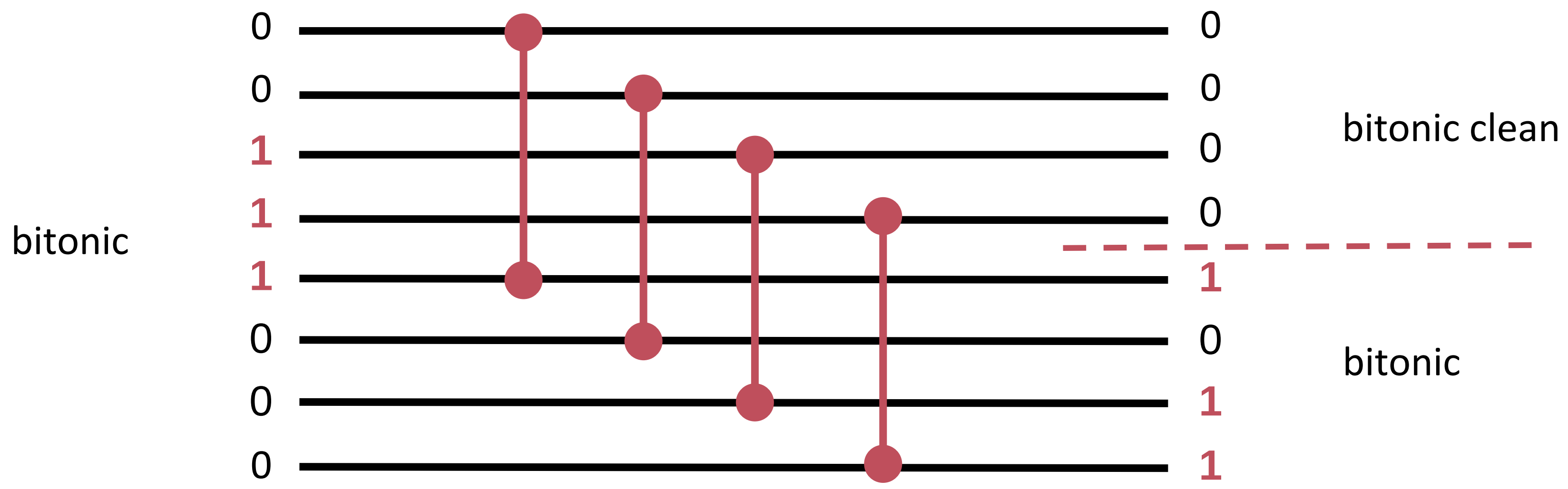
The Half-Cleaner

```
void halfclean(std::vector<T>& a, int lo, int n, boolean dir){
   for (int i=lo; i<lo+n/2; i++)
      compare(a[i],a[i+n/2], dir);
}
```

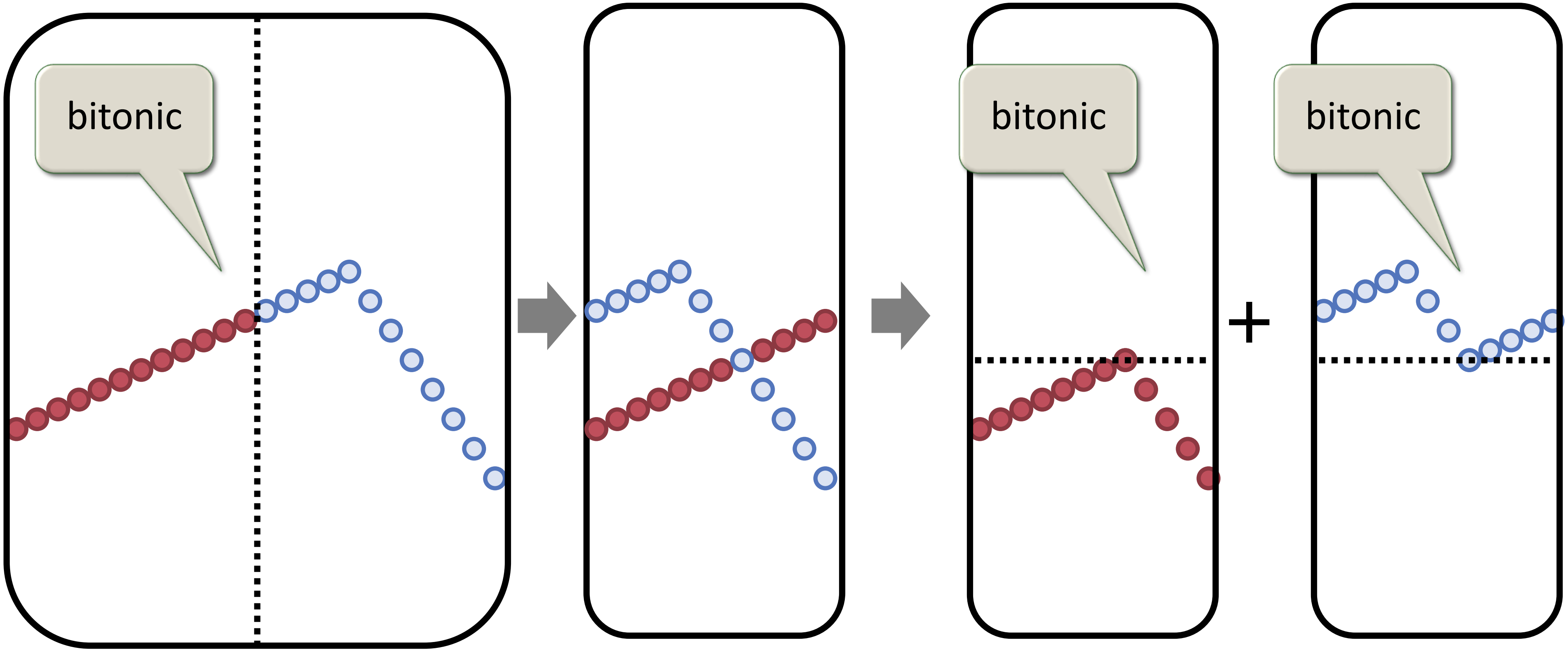


The Half-Cleaner

```
void halfclean(std::vector<T>& a, int lo, int n, boolean dir){
   for (int i=lo; i<lo+n/2; i++)
      compare(a[i],a[i+n/2], dir);
}
```



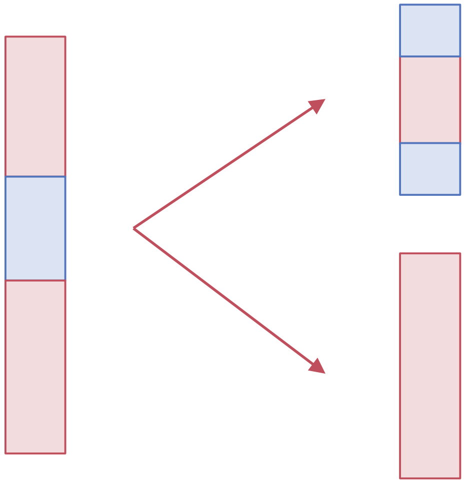
Bitonic Split Example



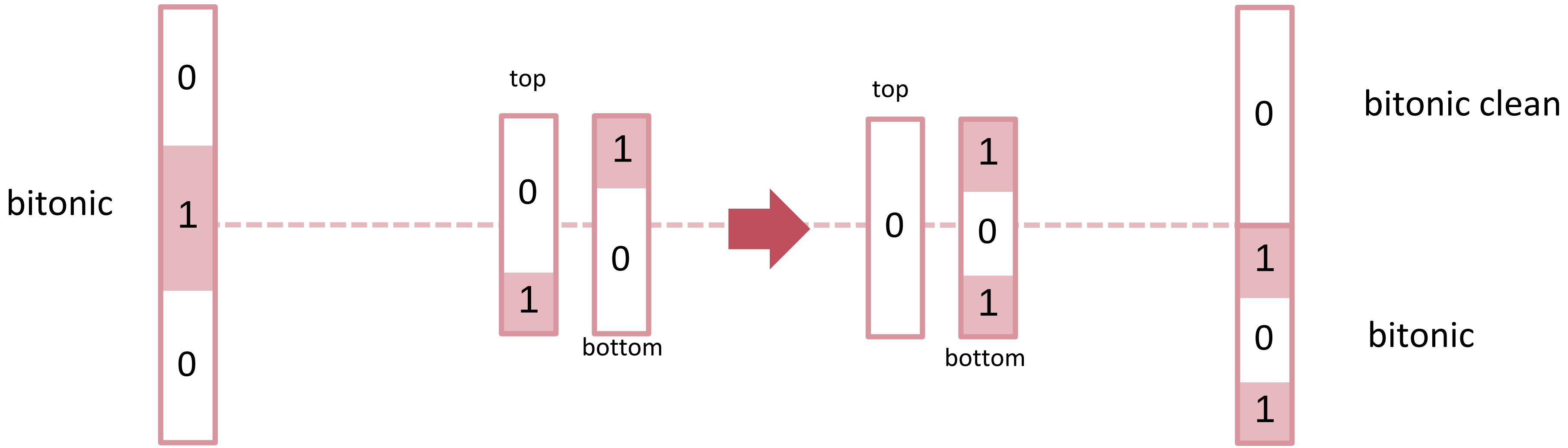
Lemma

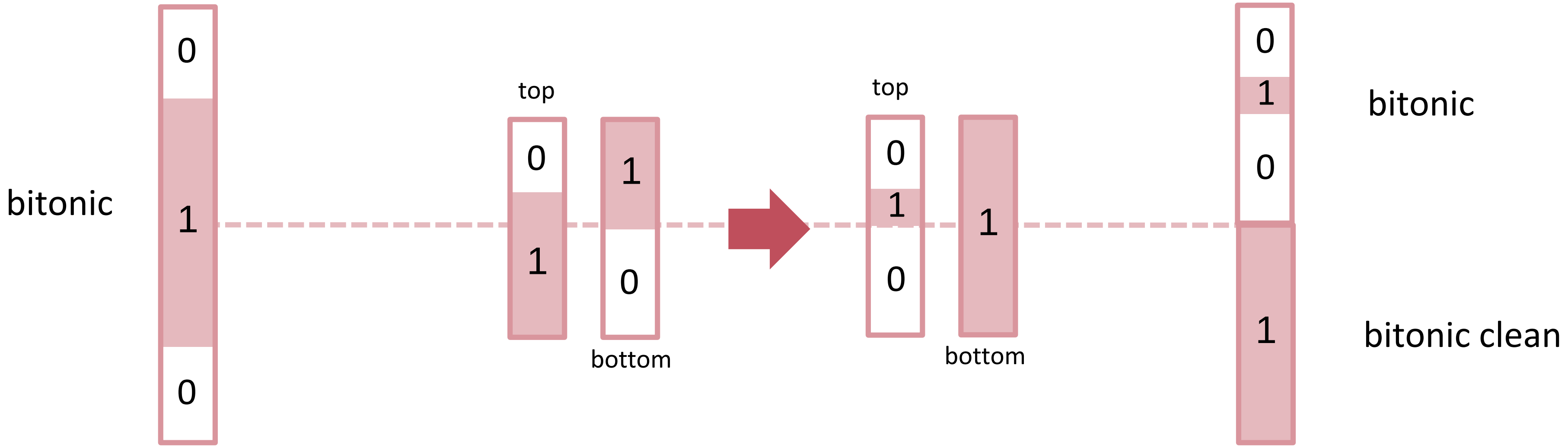
Input: a bitonic sequence of 0s and 1s,
then for the output of the half-cleaner it holds that

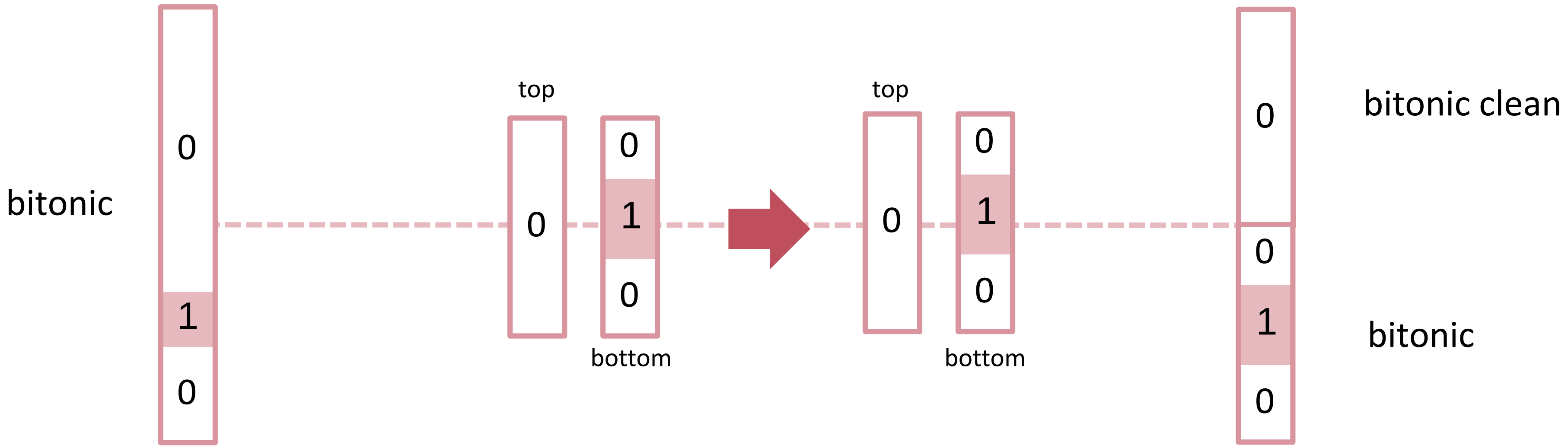
- Upper and lower half is bitonic
- [One of the two halfs is bitonic *clean* (only 0s or 1s)]
- Every number in upper half $\leq$ every number in the lower half

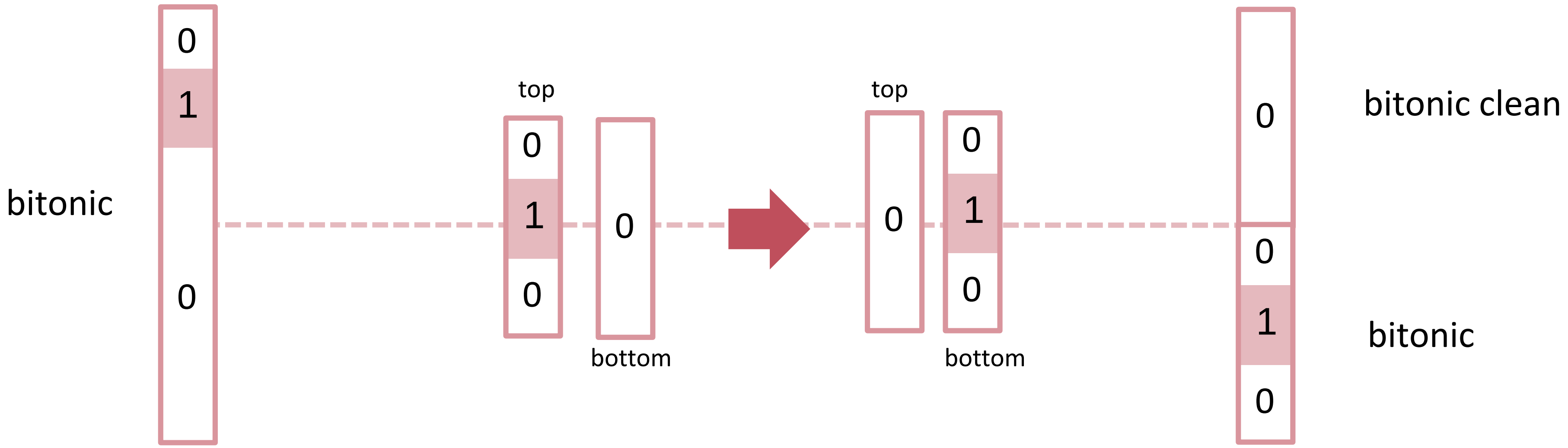


Proof: All cases

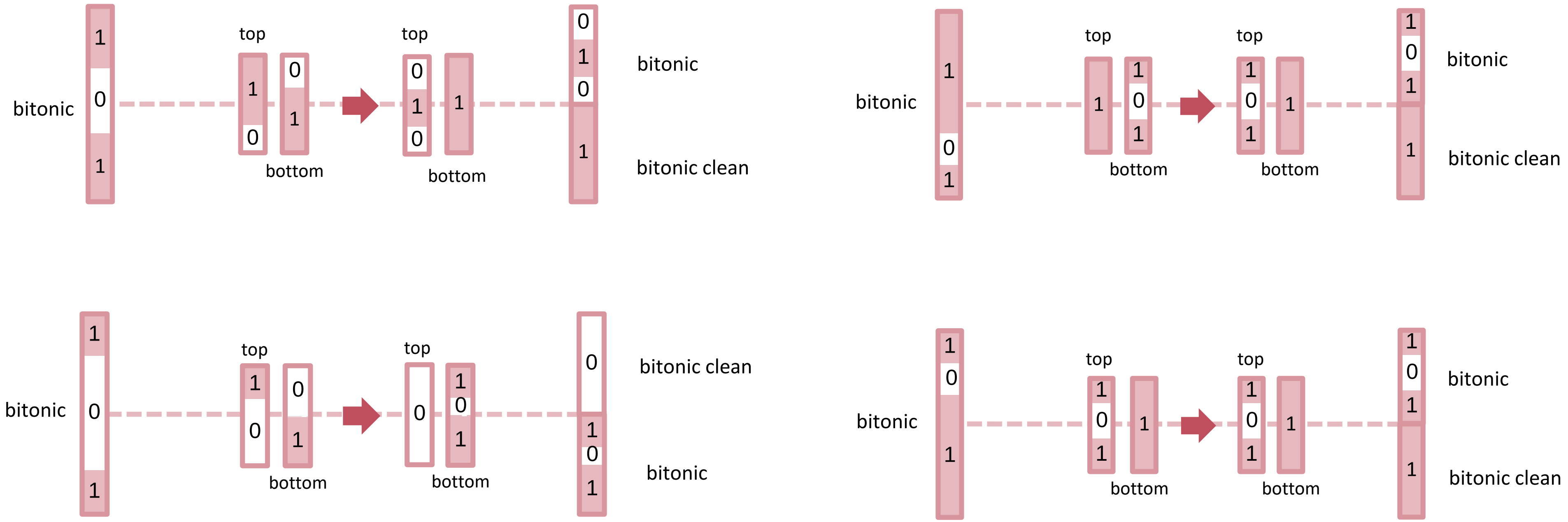



bitonic
0
1
0
top
0
1
1
0
bottom
top
0
1
0
1
bottom
0
1
0
1
bitonic
bitonic clean

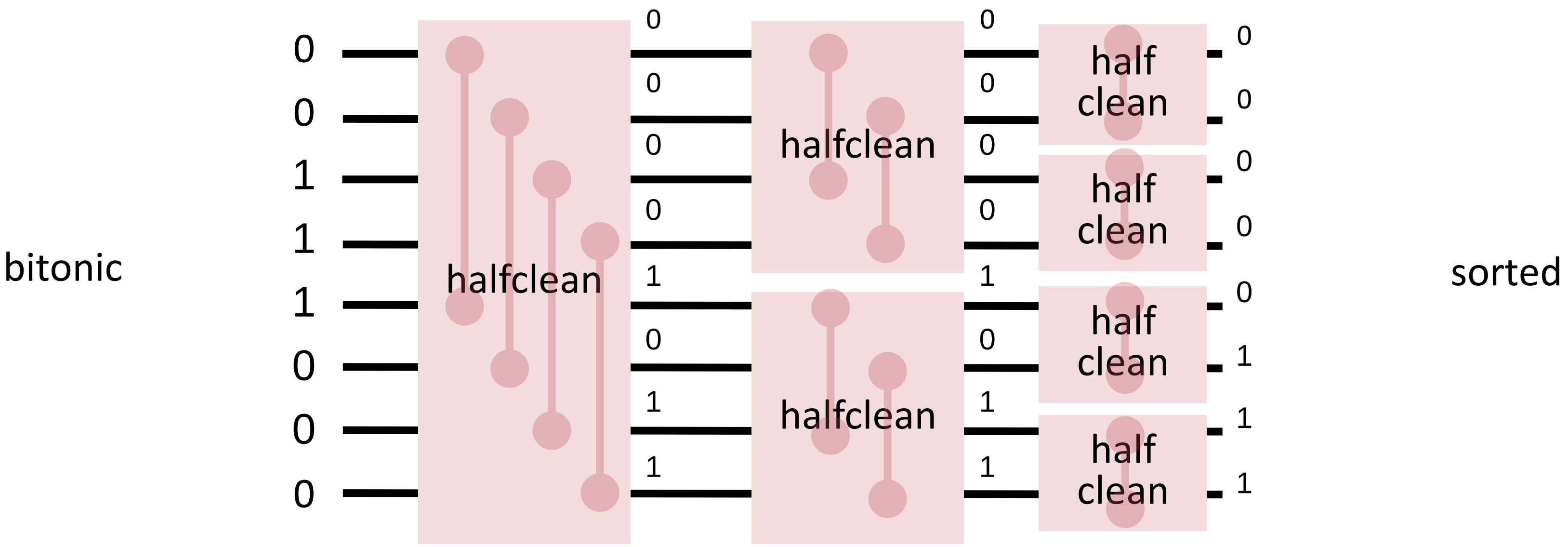

bitonic
0
1
0
top
0
0
1
0
bottom
top
0
0
1
0
bottom
0
0
1
0
bitonic clean
bitonic


bitonic
0
1
0
top
0
1
0
0
bottom
top
0
0
1
0
bottom
0
0
1
0
bitonic clean
bitonic

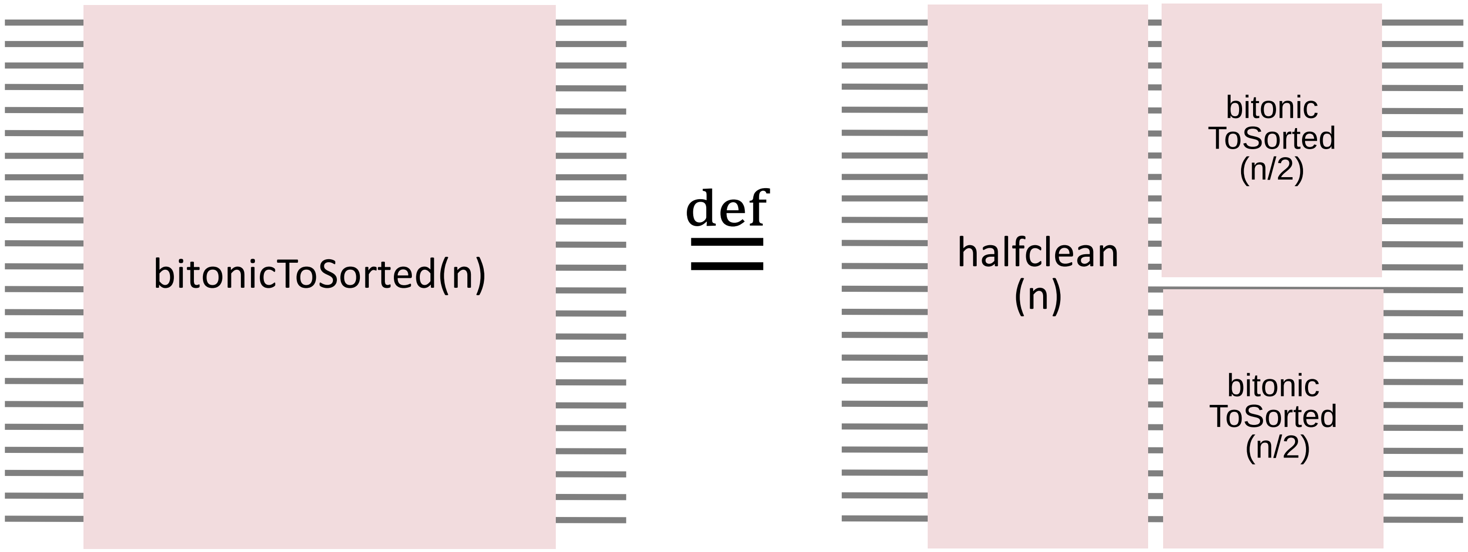
The four remaining cases (010 → 101)



Construction BitonicToSorted

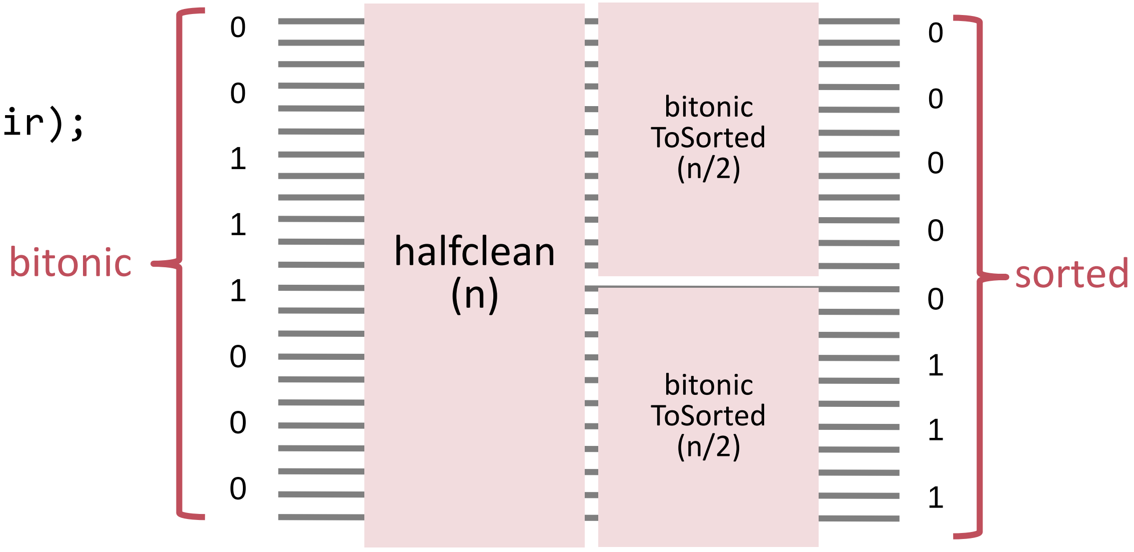


Recursive Construction

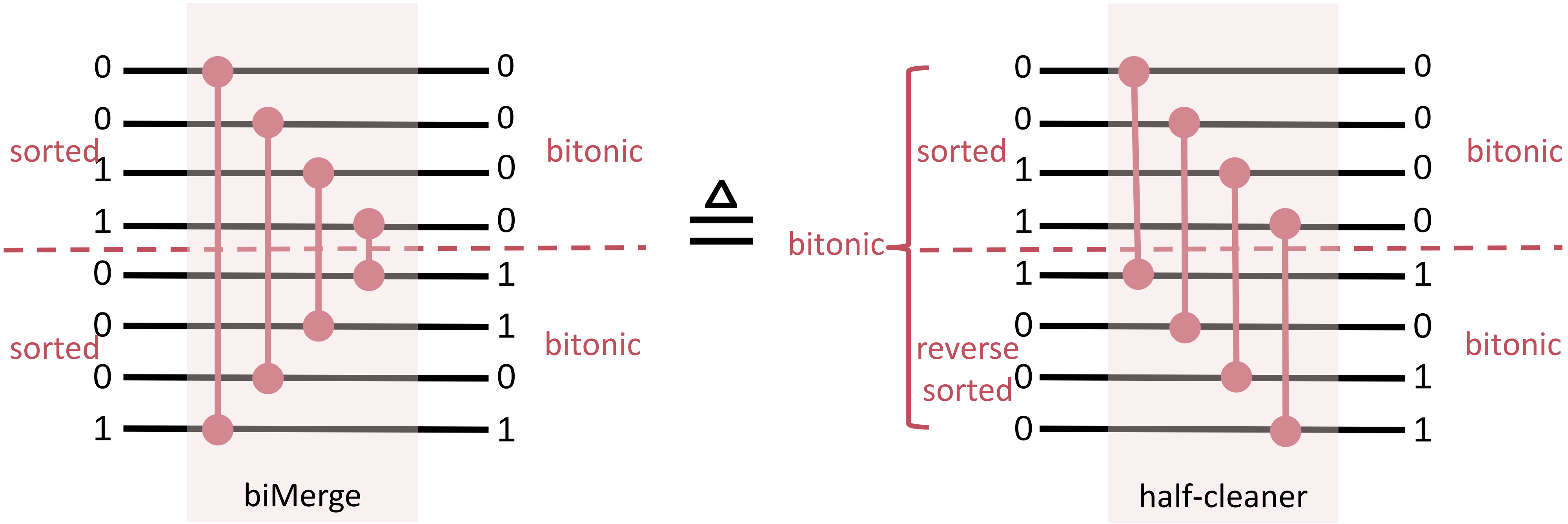


BitonicToSorted sorts a Bitonic Sequence

```
void bitonicToSorted (std::vector<int>& a, int lo, int n, boolean dir){
   if (n>1){
      halfClean(a, lo, n, dir);
      bitonicToSorted(a, lo, n/2, dir);
      bitonicToSorted(a, lo+m, n-n/2, dir);
   }
}
```

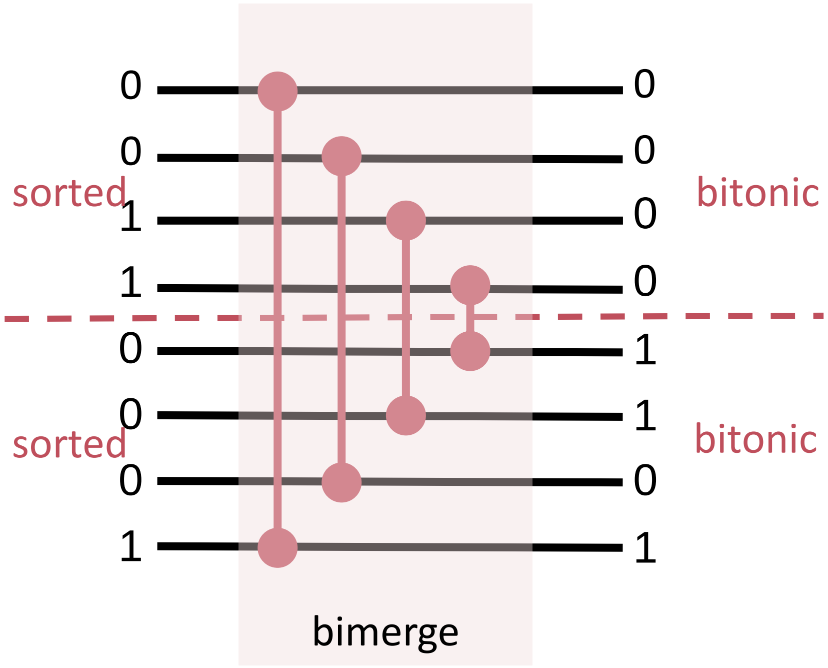


Bi-Merger



Bi-Merger on two sorted sequences acts like a half-cleaner on a bitonic sequence (when one of the sequences is reversed)

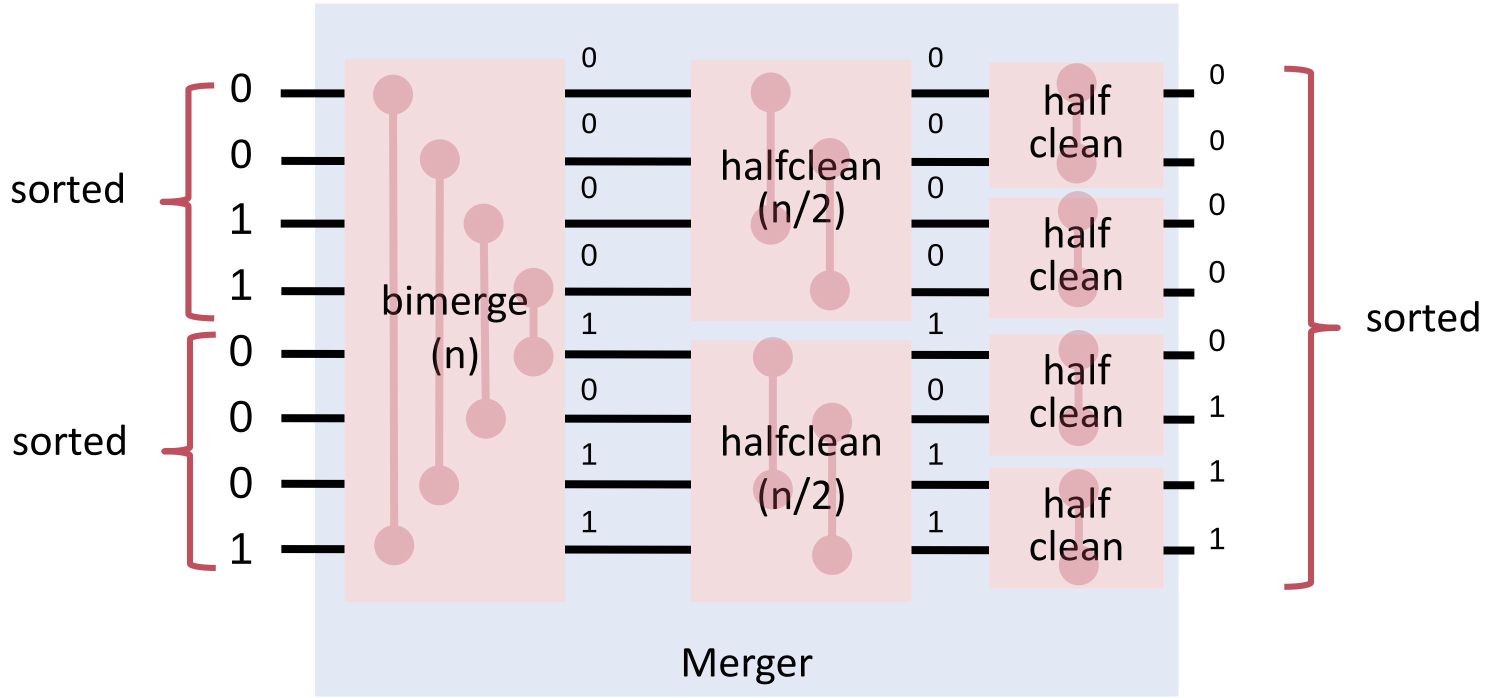
Bi-Merger



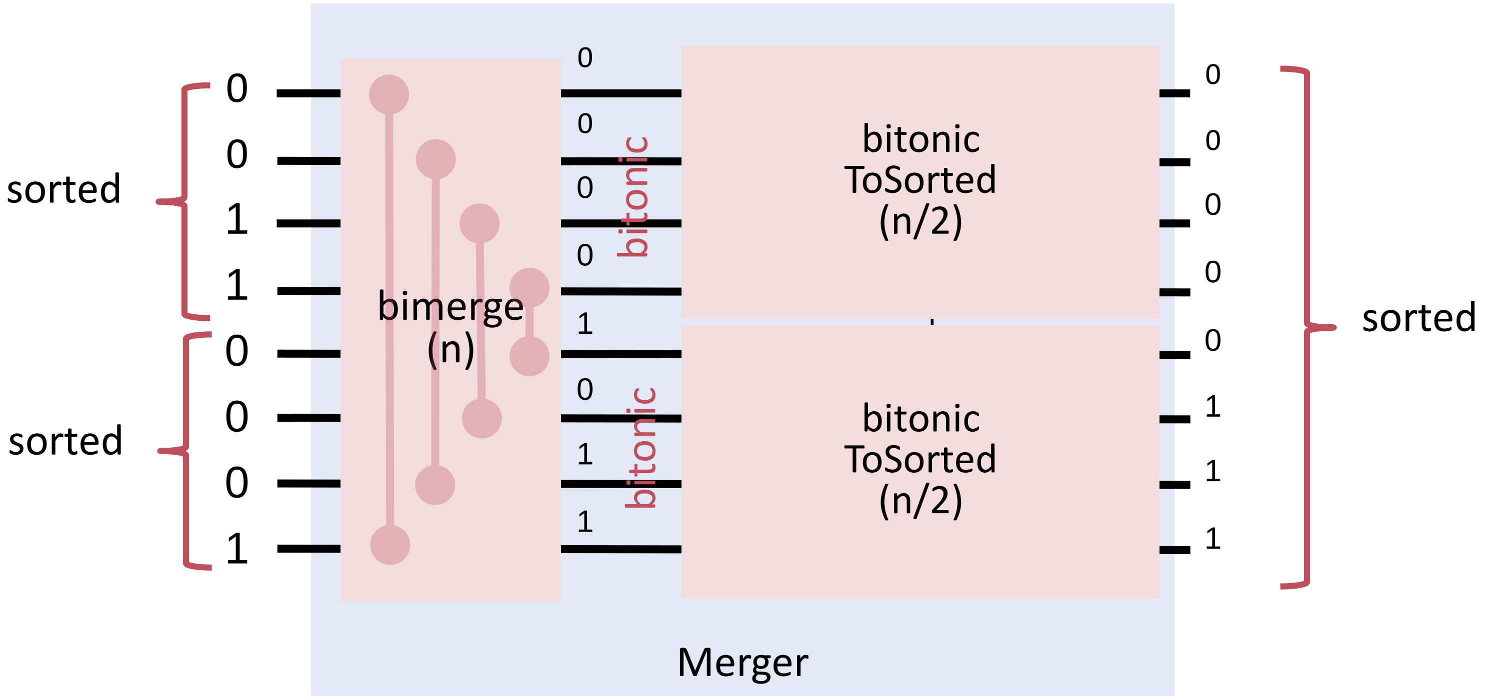

```
void bimerge(std::vector<T>& a,
                    int lo, int n, boolean dir){
   for (int i=0; i<n/2; i++)
      compare(a[lo+i],a[lo+n-i-1], dir);

}
```

Bi-Merger on two sorted sequences acts like a half-cleaner on a bitonic sequence (when one of the sequences is reversed)

Merger

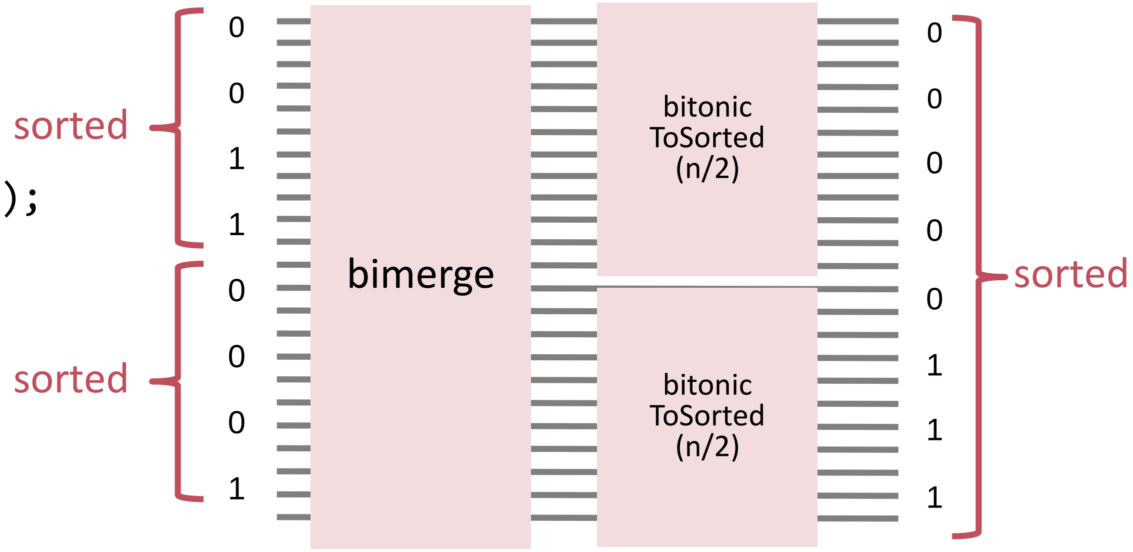

sorted
sorted
0
0
1
1
0
0
0
1
bimerge
(n)
0
0
0
0
1
0
1
1
halfclean
(n/2)
halfclean
(n/2)
0
0
0
0
1
0
1
1
half
clean
half
clean
half
clean
half
clean
0
0
0
0
0
1
1
1
sorted
Merger

Merger

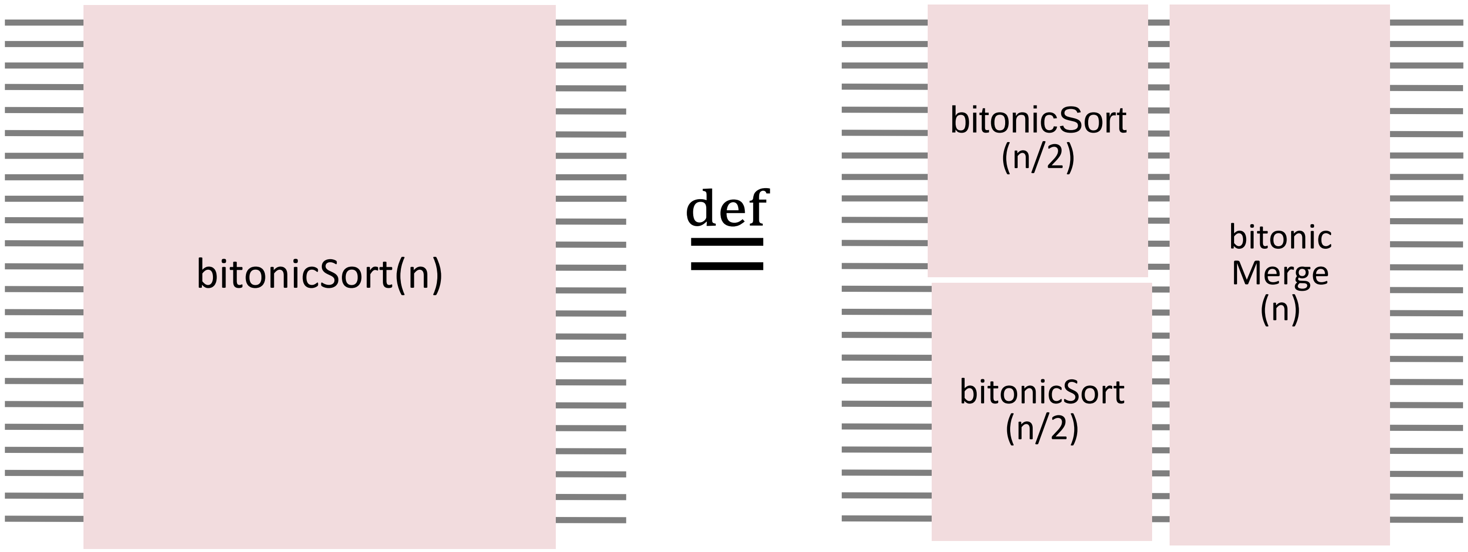

sorted
sorted
0
0
1
1
0
0
0
1
bimerge
(n)
0
0
0
0
1
0
1
1
bitonic
bitonic
bitonic
ToSorted
(n/2)
bitonic
ToSorted
(n/2)
0
0
0
0
0
1
1
1
sorted
Merger

BitonicMerge sorts a Halfsorted Sequence

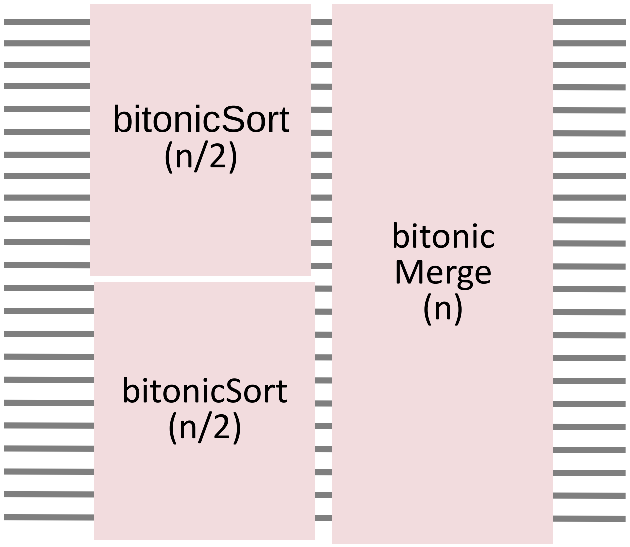
```
void bitonicMerge (std::vector<int>& a, int lo, int n, boolean dir){
   if (n>1){
      int m=n/2;
      bimerge(a,lo,n,dir);
      bitonicToSorted(a, lo, m, dir);
      bitonicToSorted(a, lo+m, n-m, dir);
   }
}
```



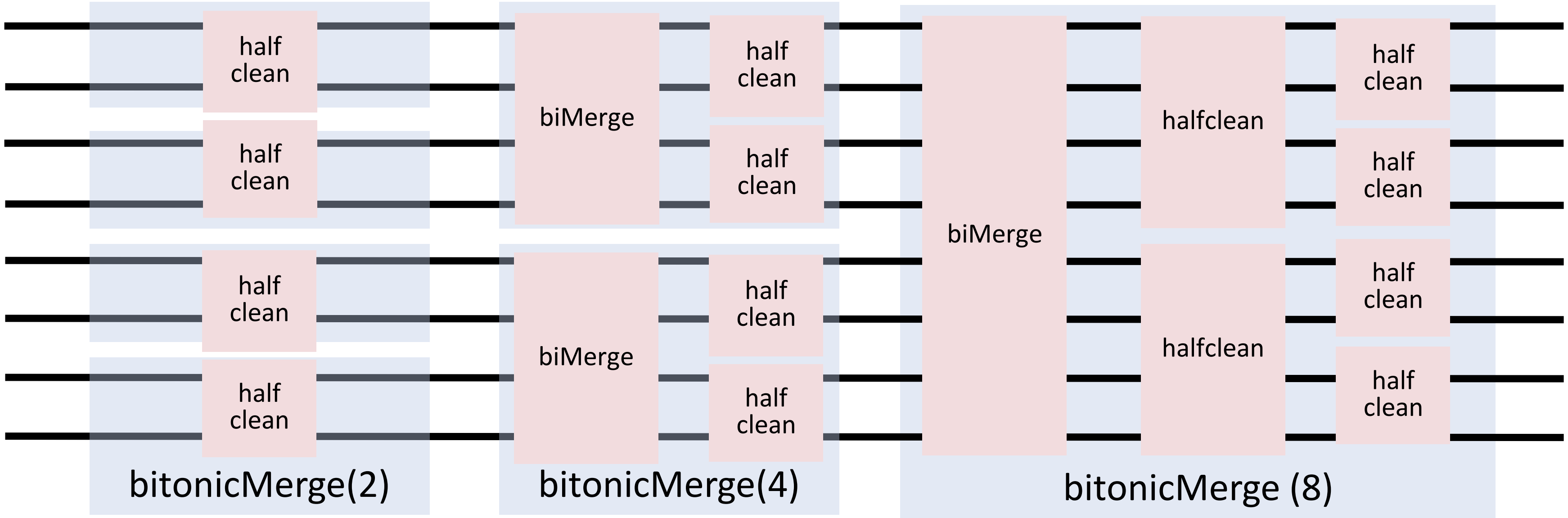
Recursive Construction of a Sorter



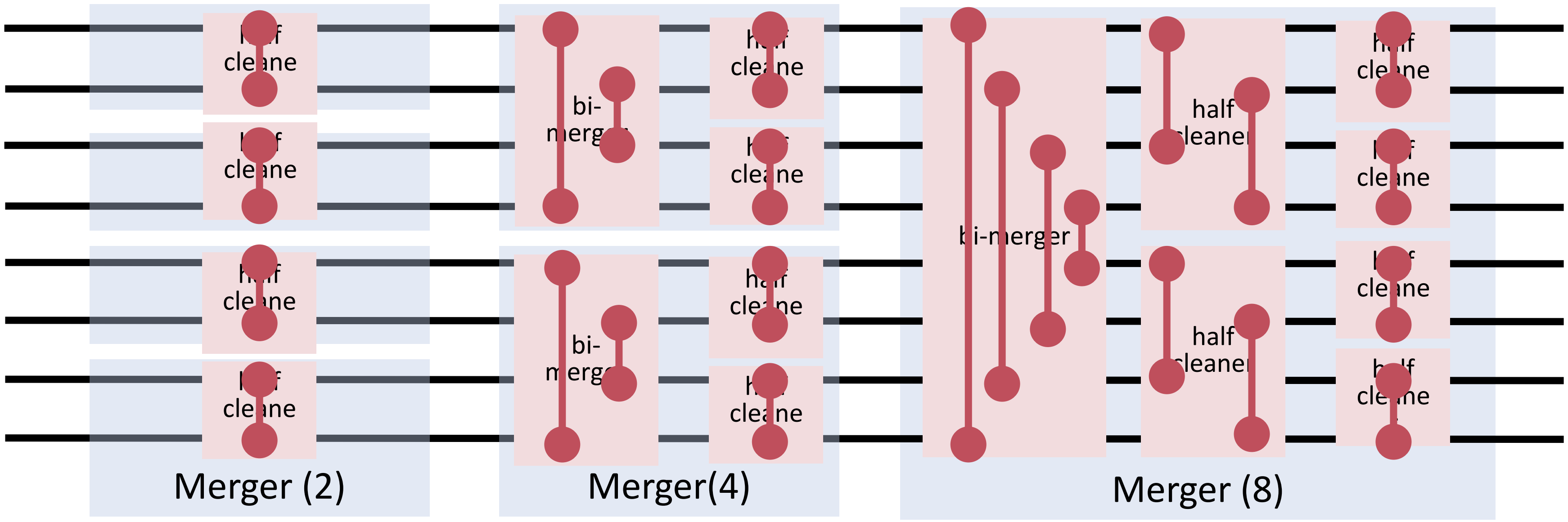
```
private void bitonicSort(std::vector&<T> a, int lo, int n, boolean dir){
   if (n>1){
      int m=n/2;
      bitonicSort(a, lo, m, ASCENDING);
      bitonicSort(a, lo+m, n-m, DESCENDING);
      bitonicMerge(a, lo, n, dir);
   }
}
```



Example


half clean
half clean
half clean
half clean
bitonicMerge(2)
biMerge
half clean
half clean
biMerge
half clean
half clean
bitonicMerge(4)
biMerge
halfclean
halfclean
half clean
half clean
half clean
half clean
bitonicMerge (8)

Example



Bitonic Merge Sort

How many steps?

#mergers

$$\sum_{i=1}^{\log n} \log 2^i = \sum_{i=1}^{\log n} i \log 2 = \frac{\log n \cdot (\log n + 1)}{2} = \boldsymbol{O(\log^2 n)}$$

#steps / merger

Zur Prüfung

- findet statt am **6.8.2018 von 9:00 – 11:00** (2h)

- Inhalt: Datenstrukturen und Algorithmen, C++ Advanced, Parallel Programming,
- Hilfsmittel: 4 A4 Seiten, handgeschrieben oder min. 11Pt Fontsize. Kopieren ist erlaubt aber nicht clever. Handgeschriebenes vom Tablet drucken ist auch erlaubt, wenn dabei die Schrift nicht wesentlich verkleinert wird (in Relation zur sonst üblichen Schreibschrift).

Vorschlag: Q&A vor der Prüfung.

- Schulferien ab 16.Juli -- Termin sollte vorher sein.
- Für Sie: je später desto besser. Vorschlag: +/- 12. Juli.

Vorbereitung

- Übungen machen / gemacht haben.

  [morgen, Freitag 1. Juni, finden Übungen statt. Besprechung Übung 13.]

- Können Sie die Vorlesungsinhalte einem Kollegen (ohne Folien) erklären?
- Alte Prüfungen stehen auf der Webseite zur Verfügung.
  - Die alten Prüfungen von Prof. Widmayer enthalten Material, das nicht behandelt wurde (geometrische Algorithmen, branch-and-bound)
  - Die "neuen" Prüfungen bei mir enthalten Material, das in den Vorlesungen von Widmayer/Püschel nicht behandelt wurden (insbesondere C++ / Parallel Programming)

Ausschlüsse

NICHT Prüfungsstoff sind

- Details der längeren Beweise (Laufzeit Algorithmus Blum, Analyse Ranomisierter Quicksort, Amortisierte Analyse von Move-To-Front, Beweis Theorem Universelles Hashing, Beweis Fibonacci Zahlen mit Erzeugendenfunktion, Beweis Amortisierte Kosten Fibonacci Heap,
- Atomare Register / RMW Operationen / Lock Free Programming
- Hardware Architekturen, Pipelining, Peterson Algorithmus

Ich bin weiterhin erreichbar unter

felix.friedrich@inf.ethz.ch