Parallel and Distributed Computing (B4B36PDV)

Matěj Kafka, Michal Jakob

kafkamat@fel.cvut.cz

https://pdv.pages.fel.cvut.cz

A case study

PARALLEL SORTING ALGORITHMS

Parallel sorting algorithms

- Common and well-understood problem…
  - …with many interesting solutions.
- You should already know common algorithms from ALG.
- Comparison-based vs non-comparison-based sorting
  - We will focus on comparison-based algorithms.

- Good case study for designing parallel algorithms.
  - How to directly parallelize simple sorting algorithms?
  - How do real-world implementations sort?

Serial quicksort

We already saw this one last week.

```
template<typename Iter>
void quicksort(Iter begin, Iter end) {
    if (end - begin <= 1) {
        return;
    }

    // select a pivot and partition the data around it
    auto pivot_it = partition(begin, end);

    quicksort(begin, pivot_it);
    quicksort(pivot_it + 1, end);
}
```

Parallel quicksort

We already saw this one last week.

```
template<typename Iter>
void quicksort(Iter begin, Iter end, size_t task_count) {
    if (end - begin <= 1) return;

    if (task_count == 1) {
        quicksort_seq(begin, end);
        return;
    }

    // select a pivot and partition the data around it
    auto pivot_it = partition(begin, end);

    #pragma omp task
    quicksort(begin, pivot_it, task_count / 2);

    // run this branch in the current task
    quicksort(pivot_it + 1, end, task_count / 2);
}
```

Cutoff -> switch to serial version.

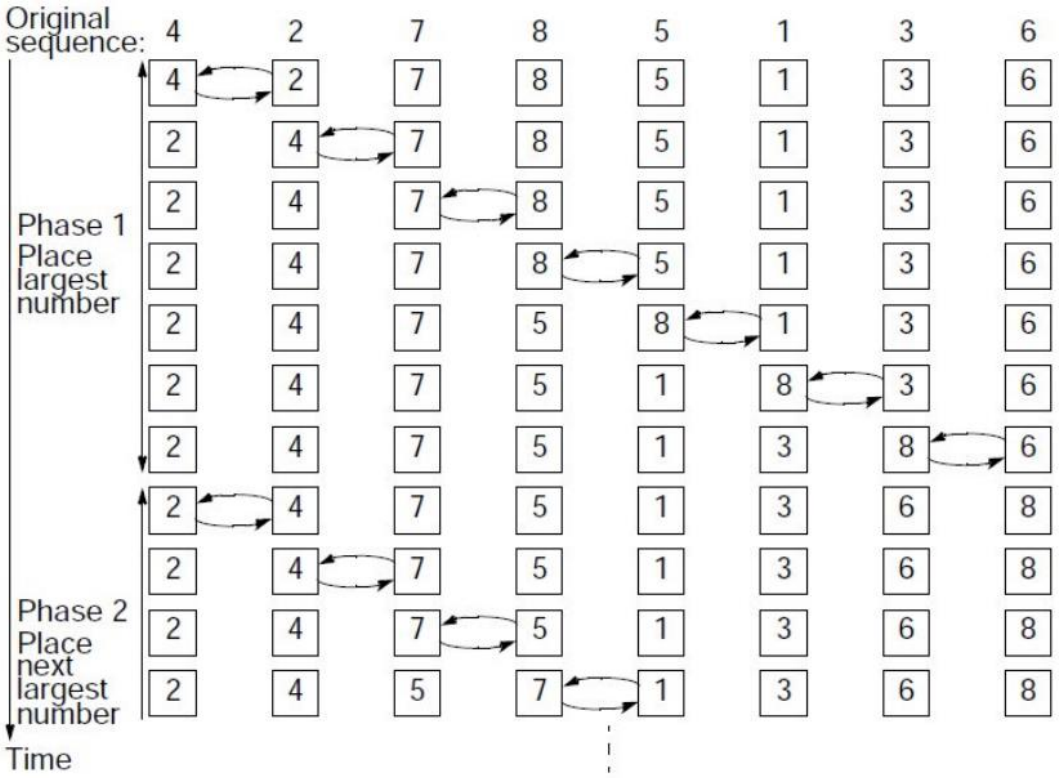
Create a task for one branch.

Avoid task overhead.

Bubble sort

The simplest usable sorting algorithm.

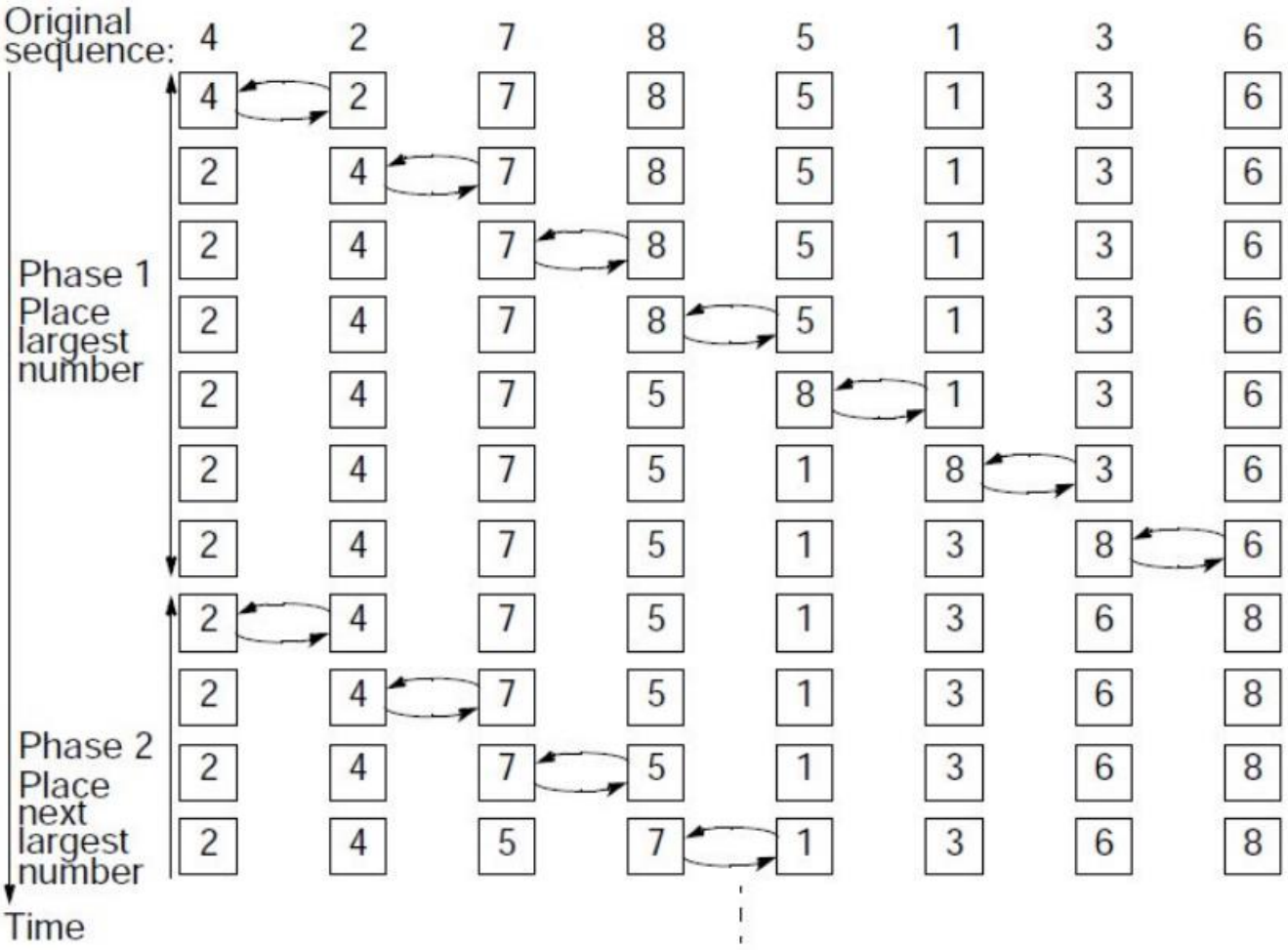
```
template<typename It>
void bubble_sort(It begin, It end) {
    for (end--; end != begin; end--) {
        for (auto it = begin; it != end; ++it) {
            minmax(*it, *(it + 1));
        }
    }
}
```



Parallel bubble sort

How to parallelize it?

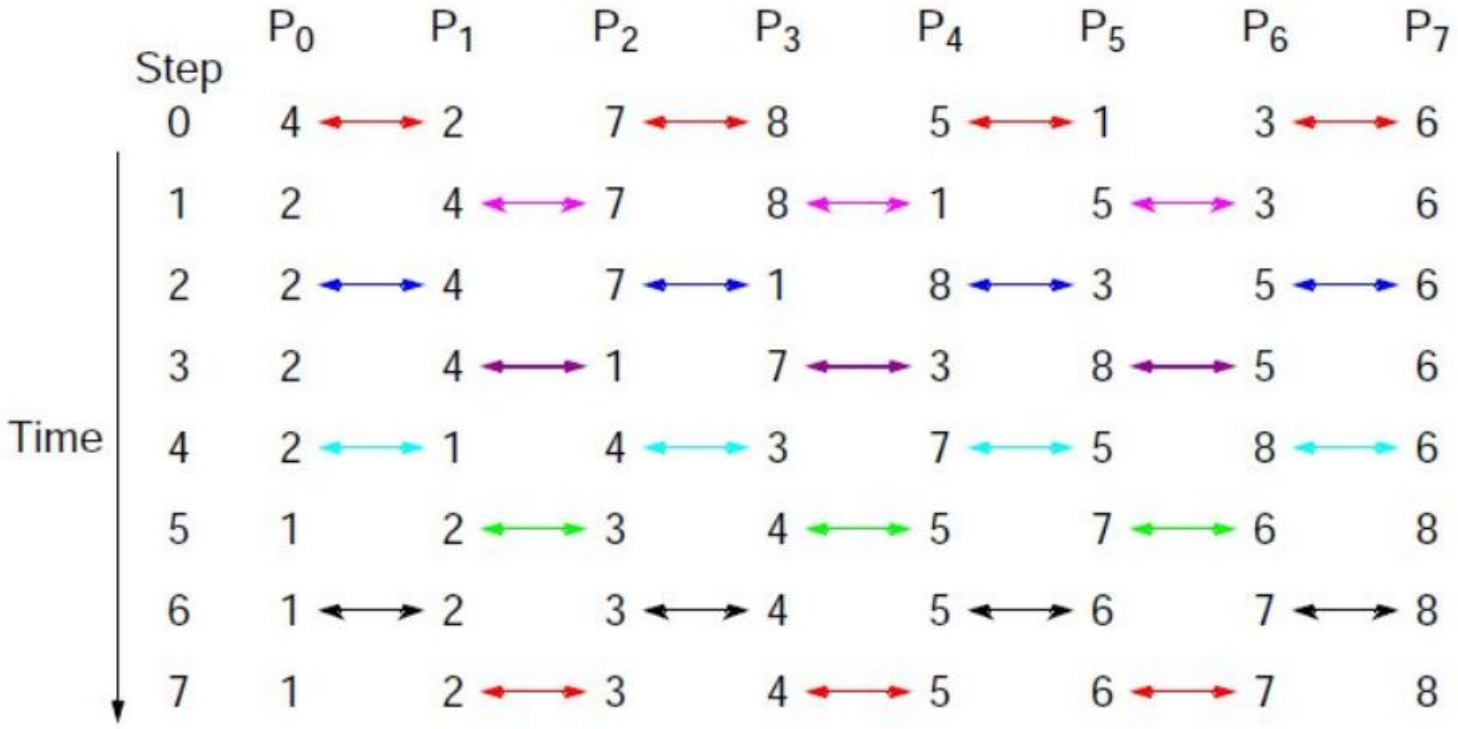
- Which steps can be done in parallel?



Odd-even sort

Bubble sort with a different comparison order.

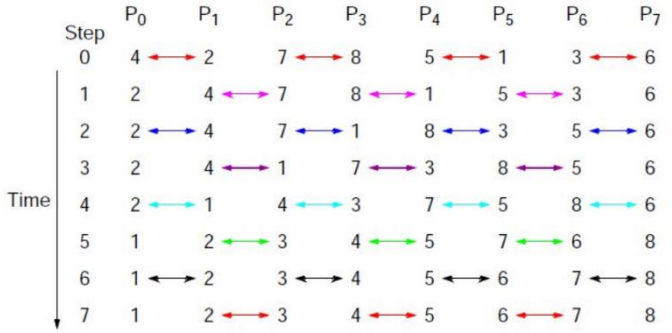
- Which steps can be done in parallel?
  - Remember heat diffusion from last lecture.



Odd-even sort

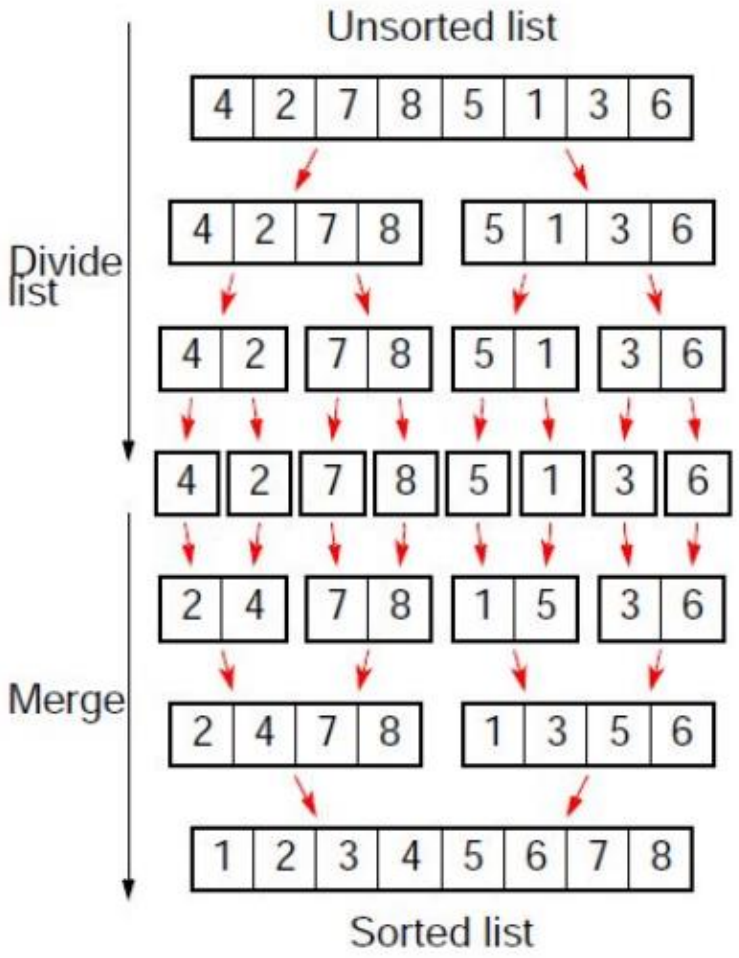
Bubble sort with a different comparison order.

```
template<typename It>
void odd_even_sort(It begin, It end) {
    end--;
    auto n = end - begin;

    #pragma omp parallel
    for (size_t i = 0; i < n; i++) {
        #pragma omp for
        for (auto it = begin; it < end; it += 2) {
            minmax(*it, *(it + 1));
        }

        #pragma omp for
        for (auto it = begin + 1; it < end; it += 2) {
            minmax(*it, *(it + 1));
        }
    }
}
```



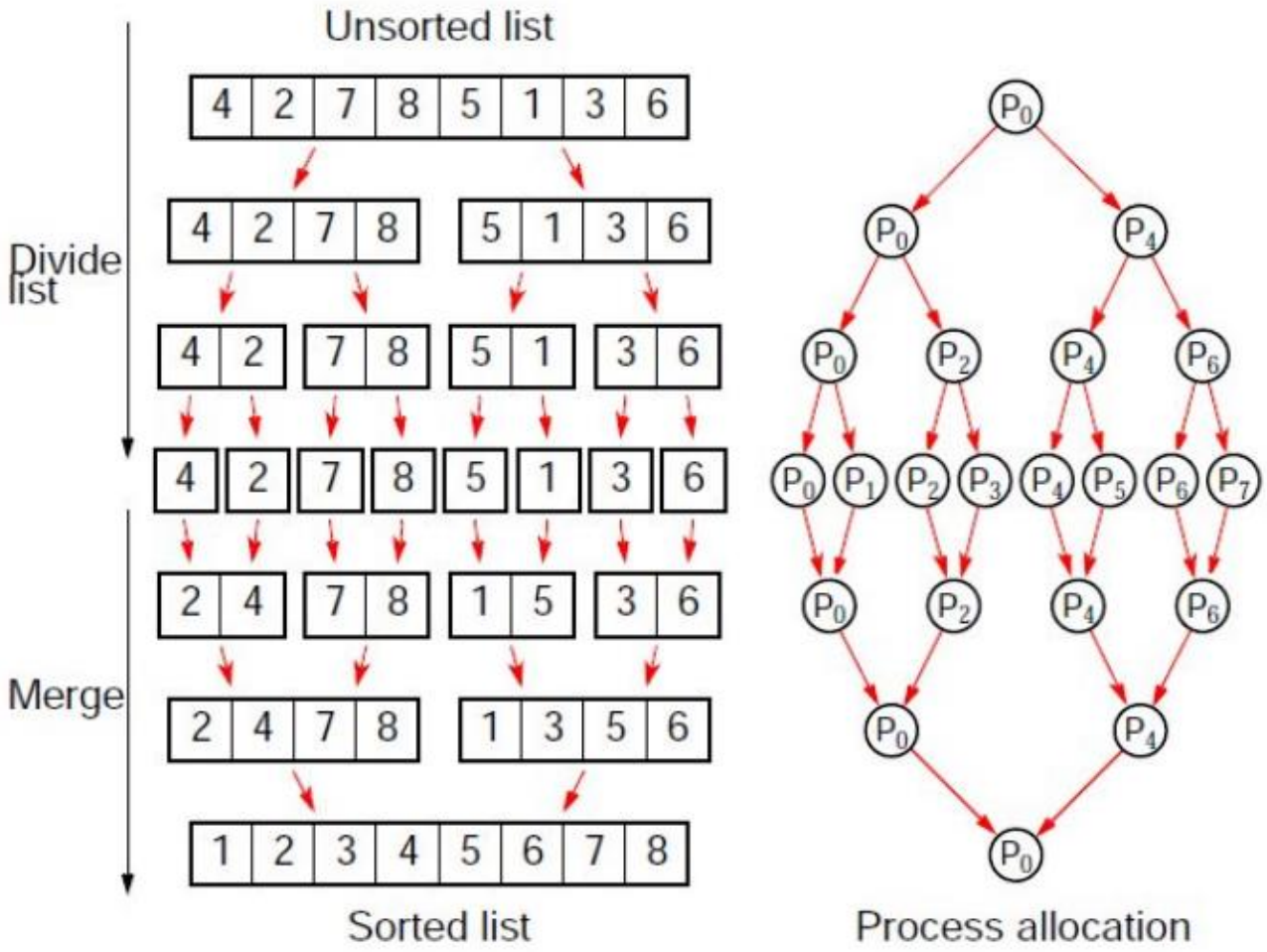
Merge sort

Another divide-and-conquer algorithm.



Parallel merge sort

Recursion? Tasks! (most likely)

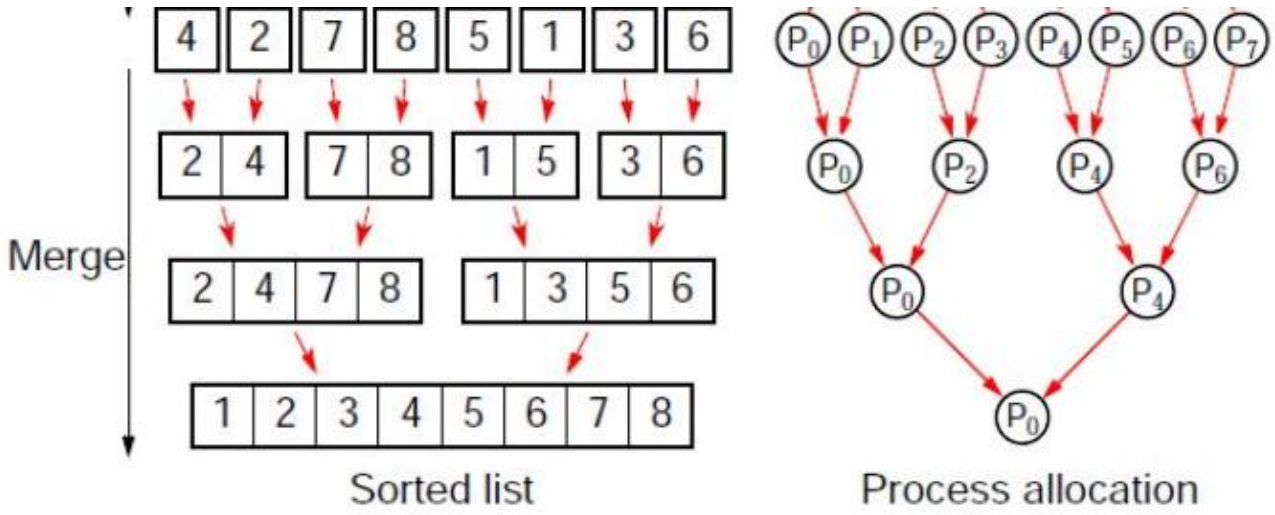


Parallel merge sort

Bottom-up formulation

We can also implement merge sort as an iterative algorithm by starting from the lowest level (merging pairs) and going up in powers of two, skipping the "divide" part of the recursion.

Can be easily implemented with 2 for loops, we can parallelize the inner loop.

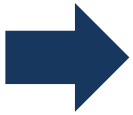


Sorting networks

Sorting small arrays quickly

- So far, we parallelized large arrays using CPU threads.
- How to efficiently sort small arrays in parallel?
  - CPU threads are too heavy, but we could implement the sorting algorithm in hardware (or an FPGA), or with lighter-weight "threads", e.g., CPU SIMD or GPUs (next lecture).
- We cannot easily branch control flow in hardware!
  - Could we avoid branching?

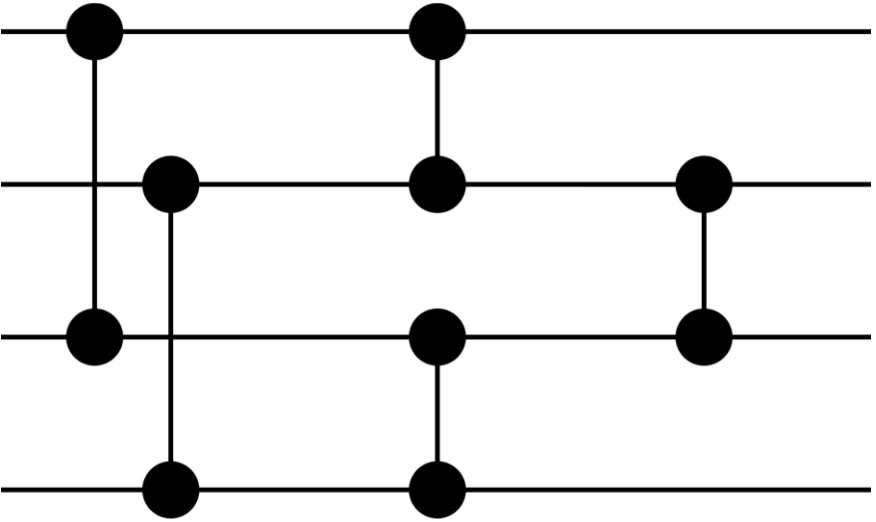
```
template<typename T>
void minmax(T& v1, T& v2) {
    if (v1 > v2) {
        std::swap(v1, v2);
    }
}
```



```
template<typename T>
void minmax(T& v1, T& v2) {
    auto min = std::min(v1, v2);
    auto max = std::max(v1, v2);
    v1 = min;
    v2 = max;
}
```

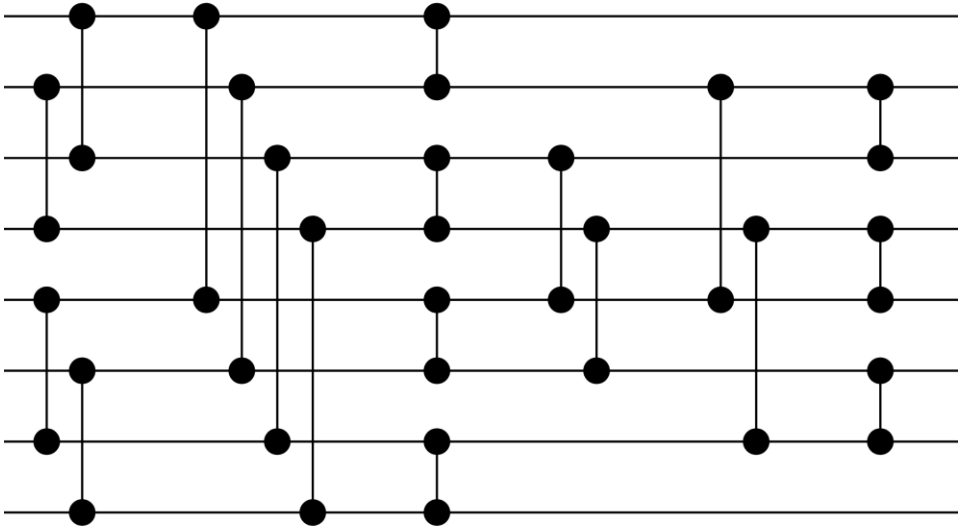
Sorting networks

Sorting small arrays quickly



Optimal sorting network for n=4

Optimal sorting network for n=8



Source: https://bertdobbelaere.github.io/sorting_networks.html

Sorting networks

Why are they useful?

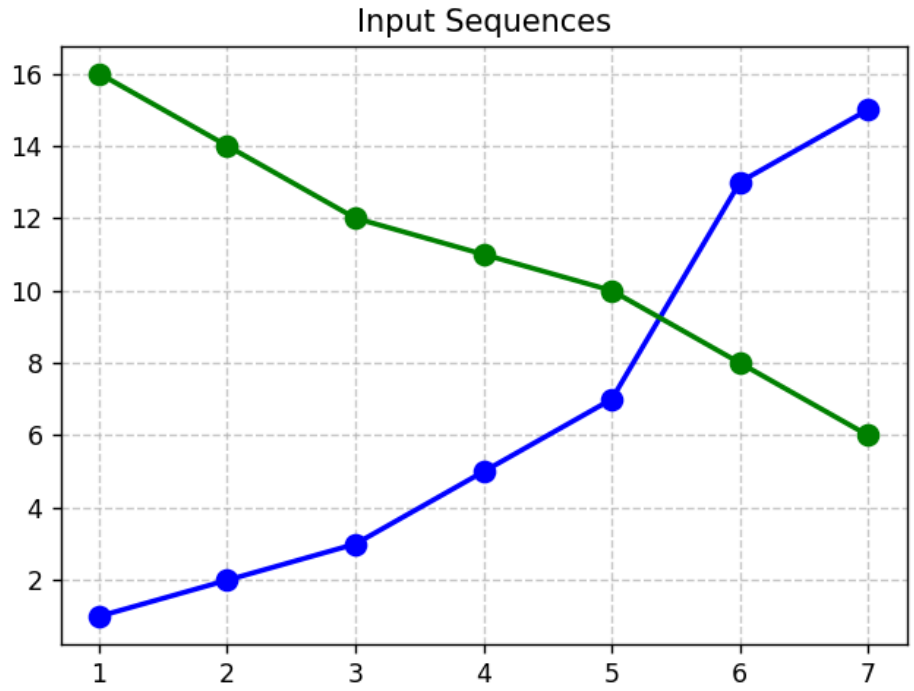
- We get a static tree of operations that's easy to parallelize on the small scale.
- Branchless code avoids penalties from branch misprediction on CPUs.
- Branchless code can be trivially implemented in hardware and in SIMD (next lecture).

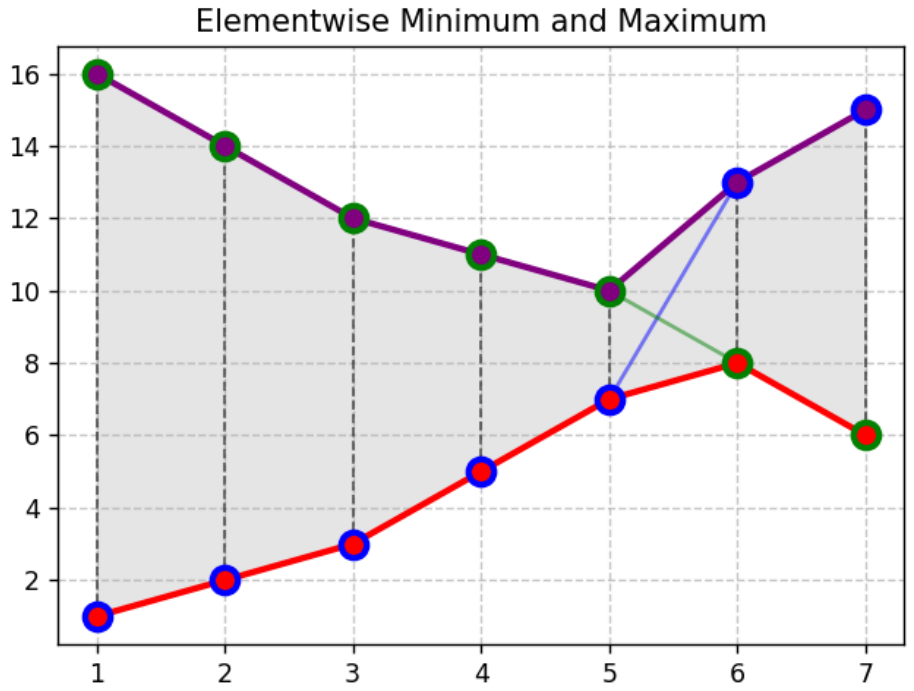
- We can construct sorting networks for small inputs by hand or using brute-force.
- What about larger inputs?

Bitonic merge

Branchless way to merge two sorted sequences.

- Let us go back to the idea of parallel merging.
- We have two ascending sequences of numbers.
  - Flip the second sequence (one ascending, one descending).
- What if we do parallel min/max between the two?

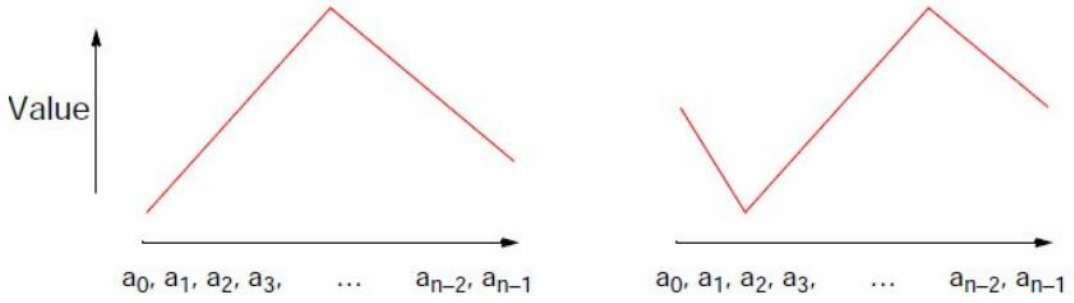


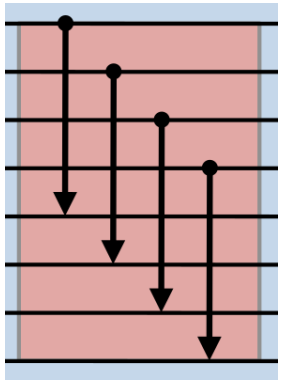


Bitonic merge

Branchless way to merge two sorted sequences.

- Let us go back to the idea of parallel merging.
- We have two ascending sequences of numbers.
  - Flip the second sequence (one ascending, one descending).
- What if we do parallel min/max between the two?
- The resulting sequences have 2 interesting properties:
  - The min sequence contains the lower half of all numbers.
  - Both sequences are *bitonic* = there is only a single "direction change", otherwise they're monotonic (or a cyclic shift of such a sequence).

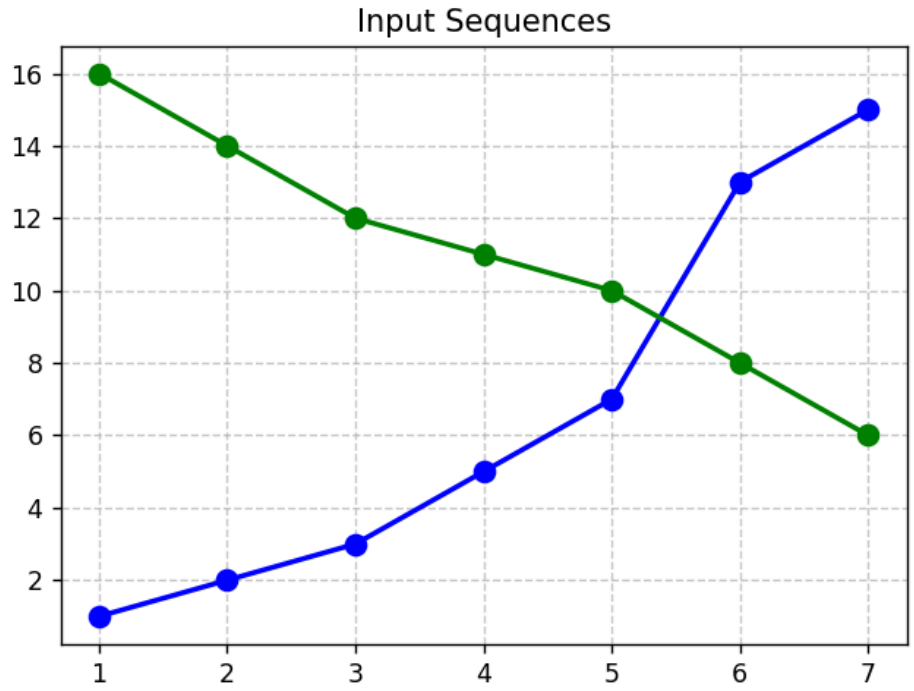


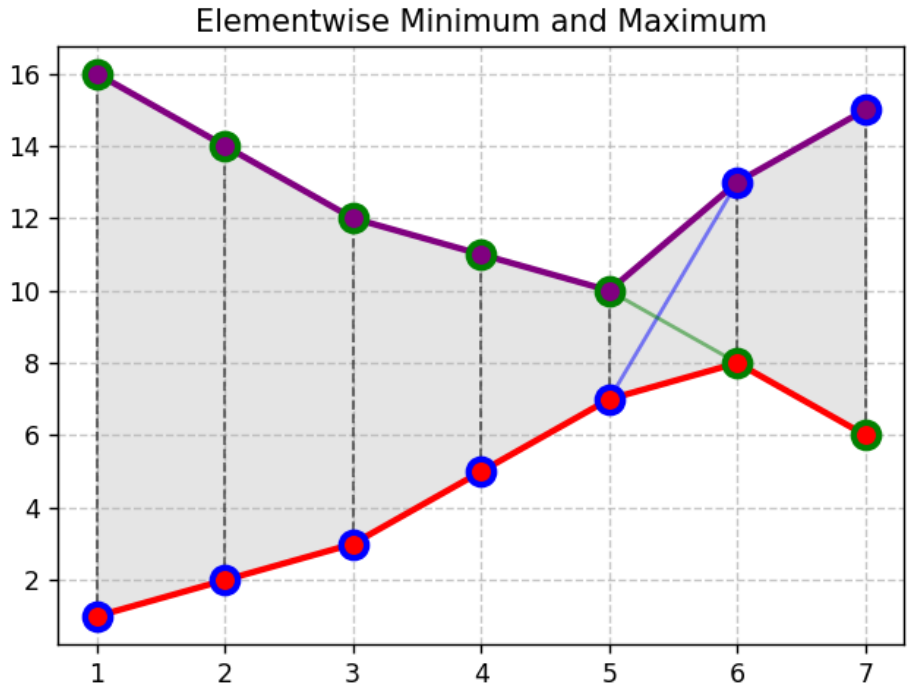


Bitonic merge

Branchless way to merge two sorted sequences.

- Proposition: "If we have **2 sorted sequences**, we can turn them into **2 bitonic sequences**, one containing lower half of all numbers, second the upper half."
- Can we turn a bitonic sequence into a sorted one?
  - If we could do that, we have a merge algorithm.

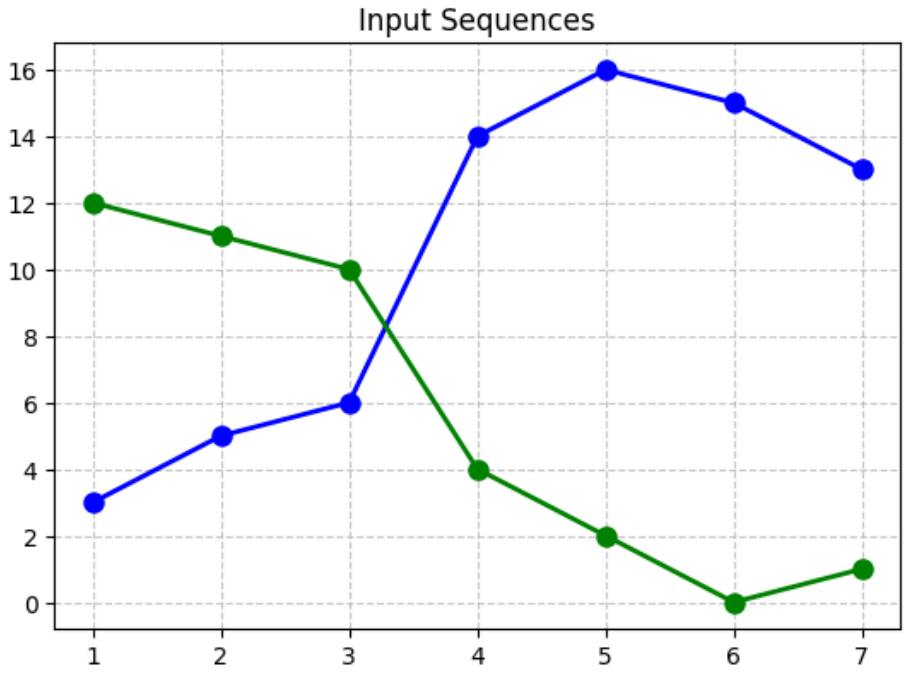


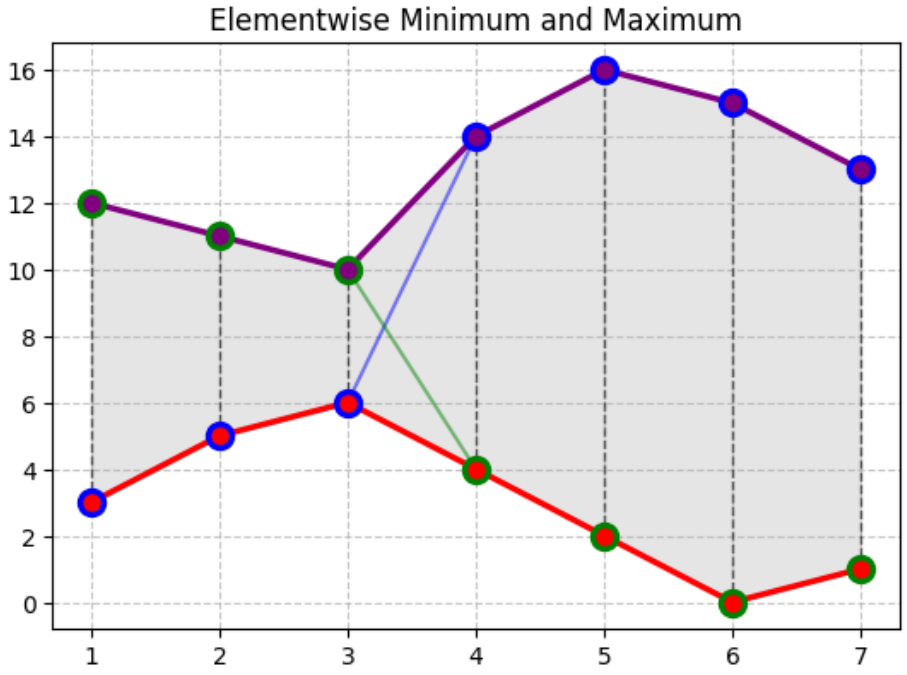


Bitonic merge

Branchless way to merge two sorted sequences.

- Observation: "Two sorted sequences can be viewed as a bitonic sequence."
- Idea: Wouldn't the proposition hold if we had a single bitonic sequence instead of two sorted ones?





Bitonic merge

Branchless way to merge two sorted sequences.

- Observation: "Two sorted sequences can be viewed as a bitonic sequence."
- Idea: Wouldn't the proposition hold if we had a single bitonic sequence instead of two sorted ones?

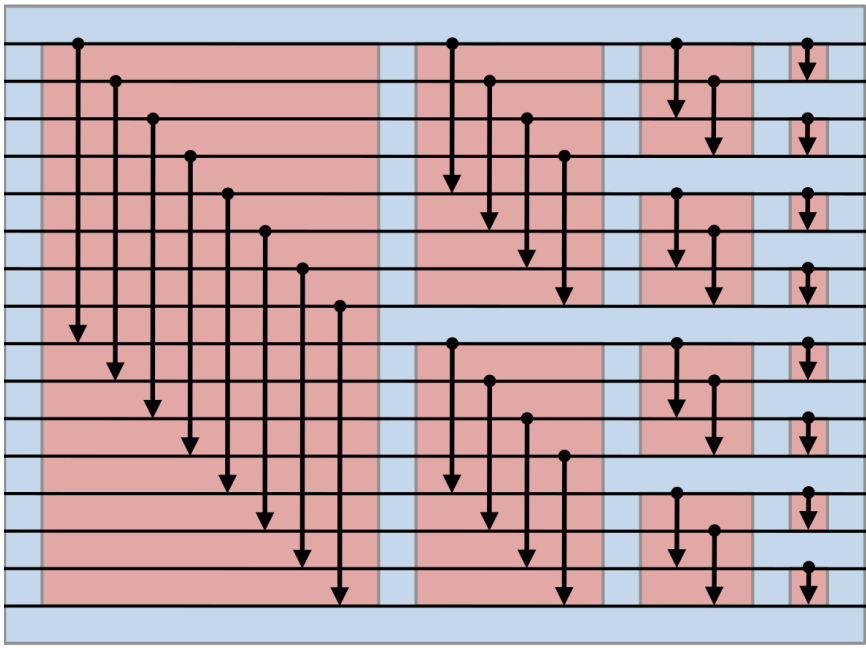
- If that holds, we have the following modified proposition:

  "If we have **a bitonic sequence**, we can turn it into **2 bitonic sequences**, one containing lower half of all numbers, second the upper half."

Bitonic merge

Branchless way to merge two sorted sequences.

- By recursive application of the min/max operation, we can convert a bitonic sequence to a sorted sequence.
- Proof is non-trivial, utilizing bitonicity and the zero-one principle.
- Intuitively, in each round, we move the values closer to the correct place, while the bitonic property ensures that a high value does not prevent a slightly lower value from moving to the upper half (and vice versa).



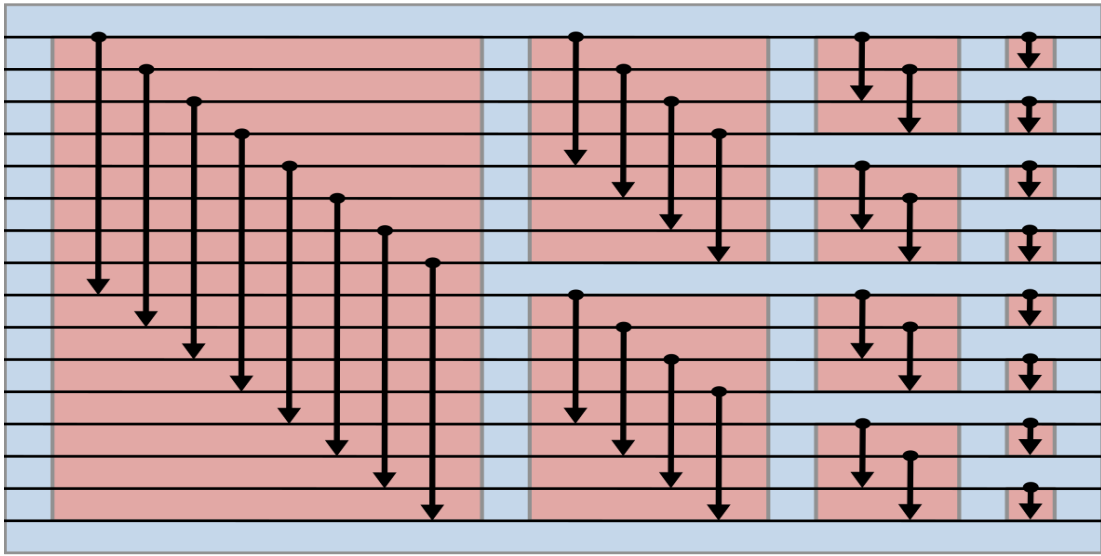
Source: https://commons.wikimedia.org/wiki/File:BitonicSort1.svg

Bitonic merge

Branchless way to merge two sorted sequences.

"If we have **a bitonic sequence**, we can turn it into 2 bitonic sequences, one containing lower half of all numbers, second the upper half."

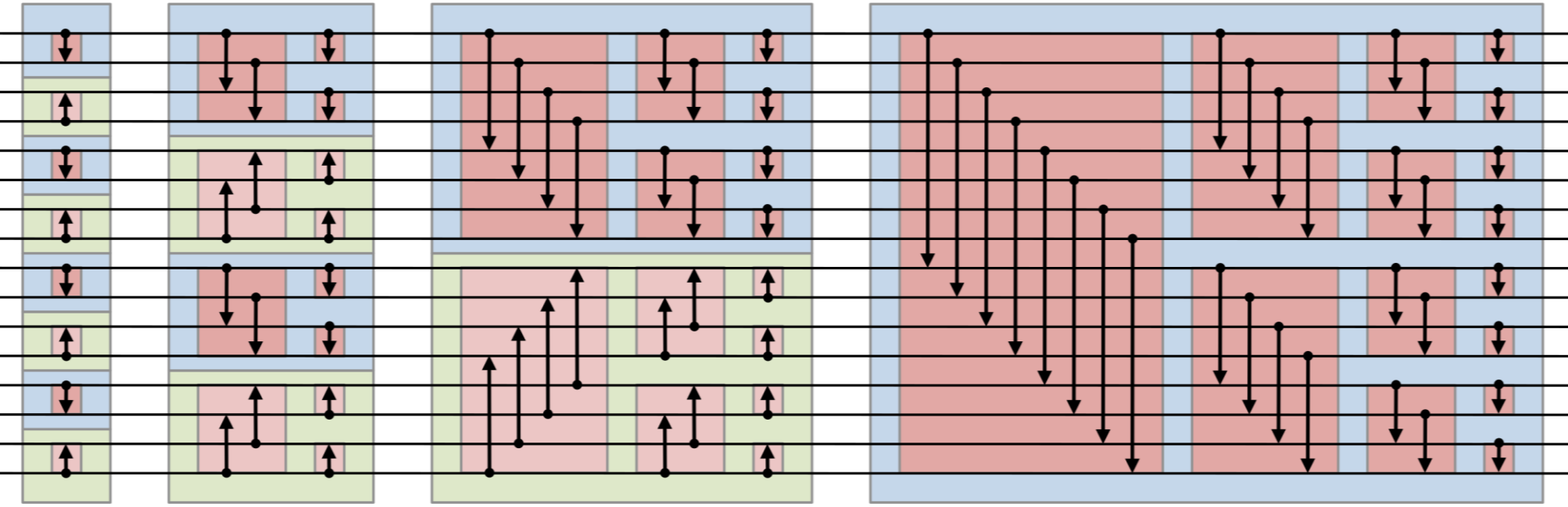
"If we have **2 sorted sequences** of the same length, we can merge them into **a single sorted sequence** using only repeated application of the vector min/max operation.



Bitonic sort

An algorithm for constructing larger sorting networks.

- A singleton array is trivially sorted.
- Using bitonic merge, we can combine two sorted sequences of the same length into a single sorted sequence.
- -> We just inductively described how to create a sorting network for any $\mathrm{N} = 2^x$.

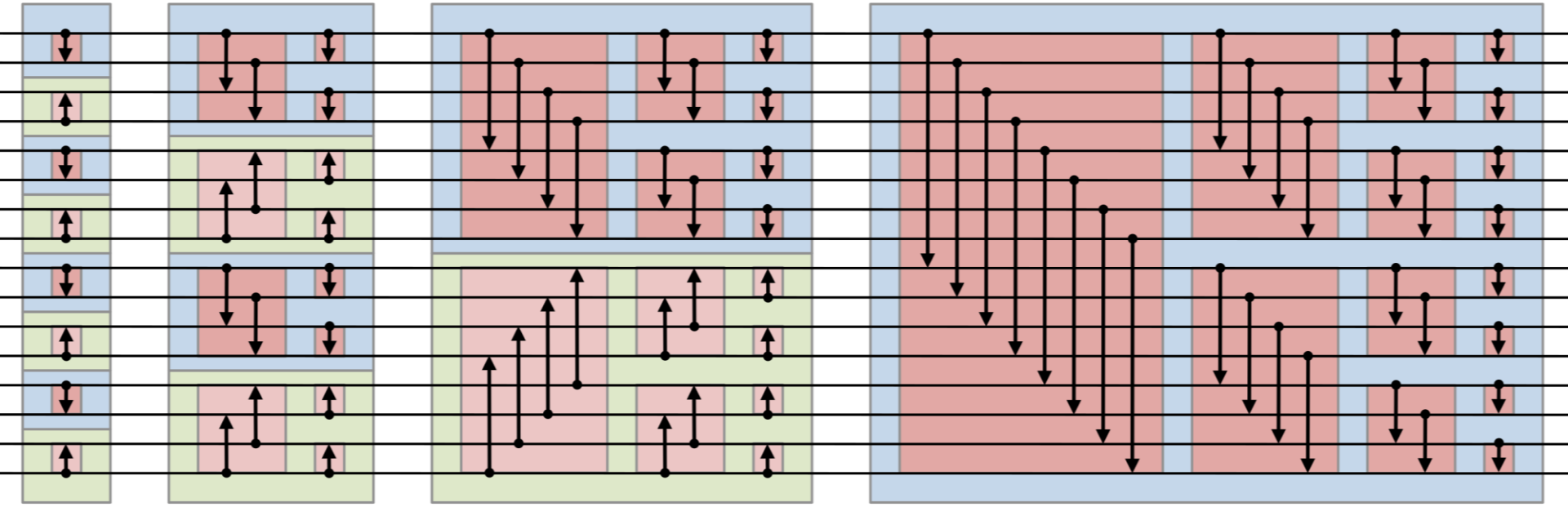


Source: https://commons.wikimedia.org/wiki/File:BitonicSort1.svg

Bitonic sort

Why is it interesting?

- Algorithm for creating larger sorting networks.
- Very amenable to parallelization, $O(\log^2 N)$ time complexity on sufficiently parallel hardware.
- Great fit for GPUs and SIMD.



Source: https://commons.wikimedia.org/wiki/File:BitonicSort1.svg

How does the C++ standard library sort?

SORTING IN THE REAL WORLD

Sorting in the real world

How does the C++ standard library sort?

- We already saw multiple parallel sorting algorithms.
- And we saw how OpenMP works internally.
- Let us apply both and see how widely-used libraries, such as the C++ standard library, sort things.

```cpp
auto vec = std::vector<uint32_t>{3, 4, 1, 5, 2};

// serial sort
std::sort(vec.begin(), vec.end());

// parallel sort
std::sort(std::execution::par_unseq, vec.begin(), vec.end());
```

C++ standard libraries

Not just a single C++ standard library.

- There are 3 mainstream implementations of the C++ standard library:
  - libstdc++ (GCC, most common on Linux)
  - libc++ (Clang, most common on macOS)
  - STL (MSVC, Windows-only)

- The parallel sort implementation in STL is the simplest and most readable (and somewhat slower than the other two), we'll explore it more in-depth.

- Warning: Stdlib code often looks ugly on first sight, you'll get used to it (if you look long enough).

Coding session 1

EXPLORING STD::SORT IN MICROSOFT'S STL

During the 6th week, we've started part of the next topic (SIMD).