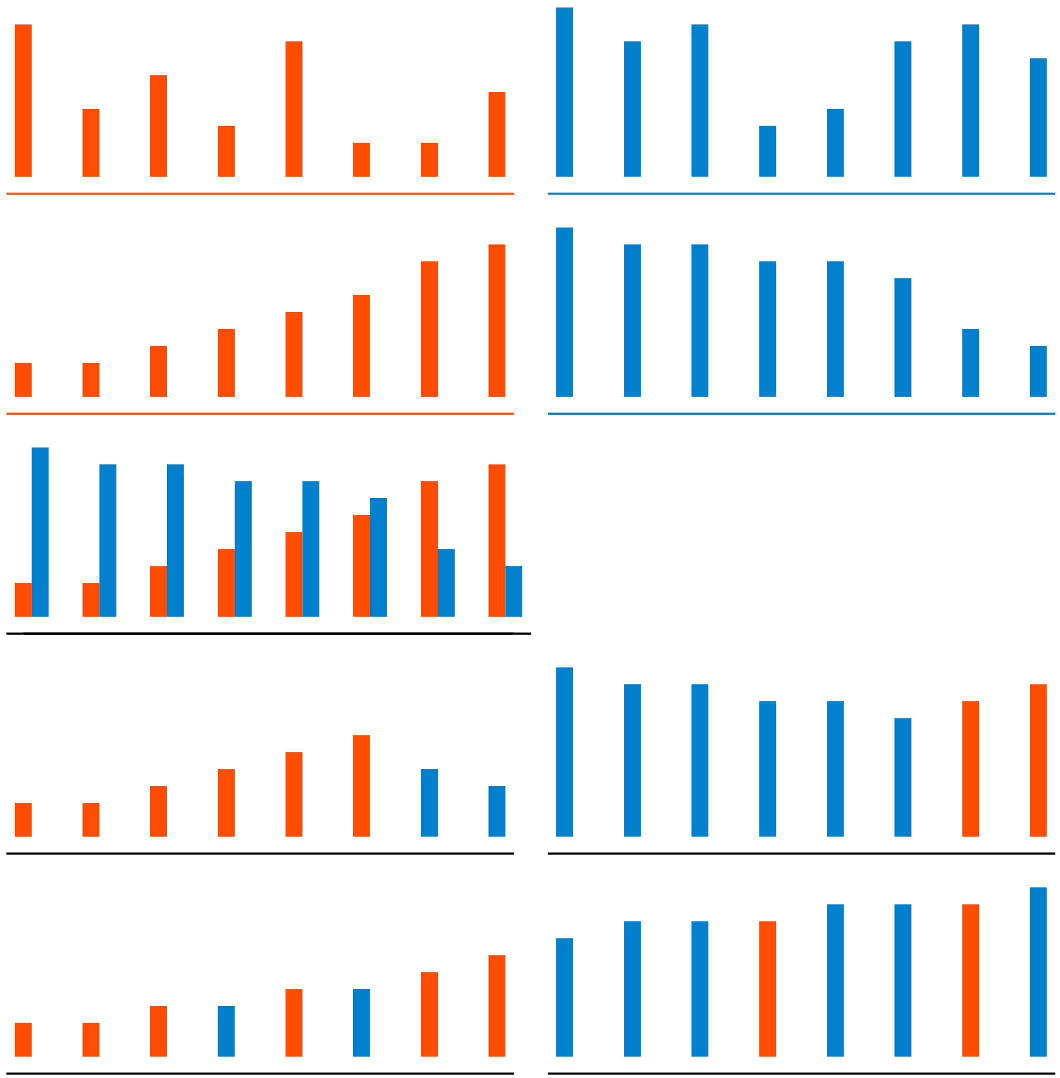
Visualizing Bitonic Sorting on a Linear Array



Initial data sequence

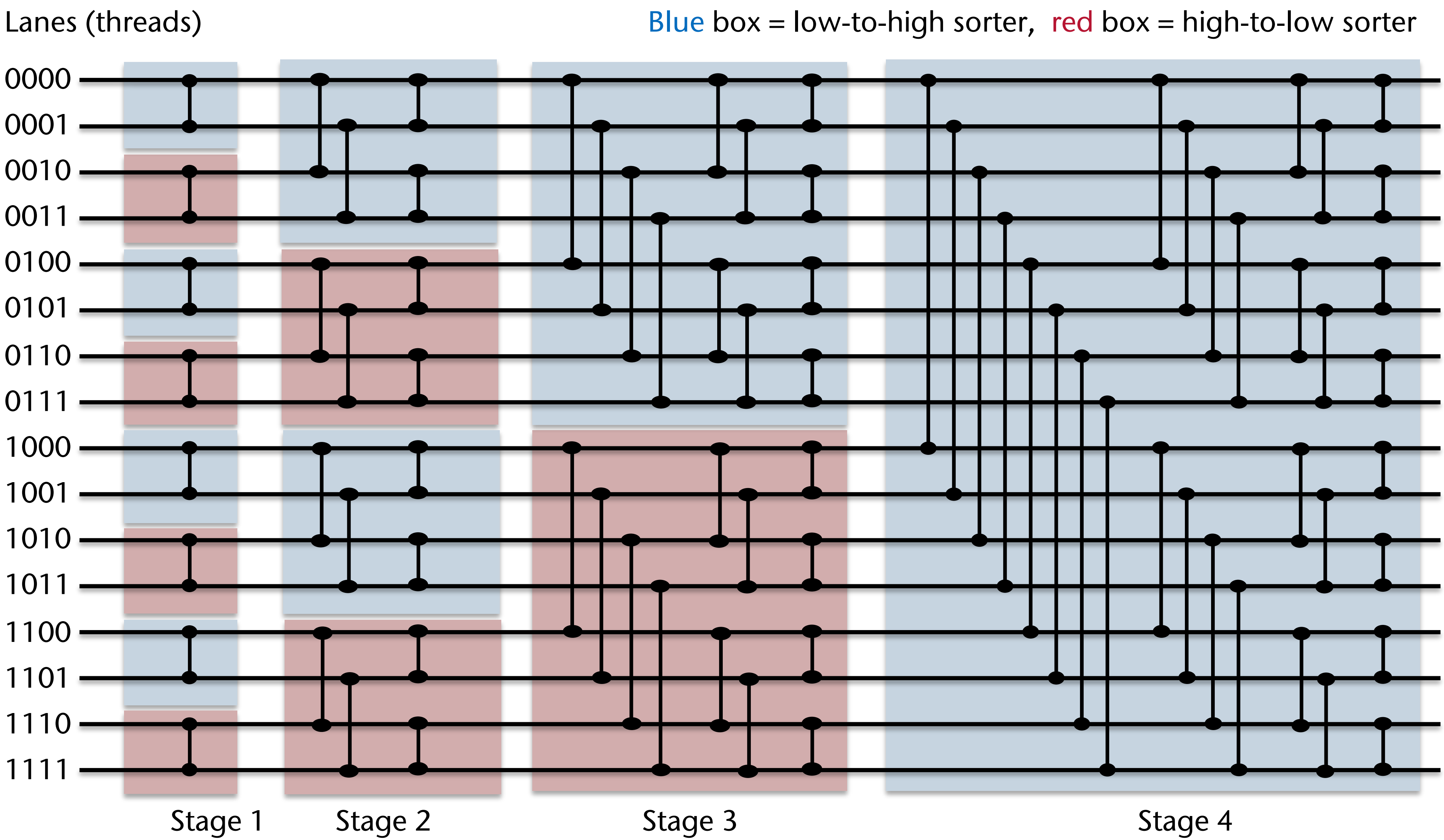
1: Sort half-arrays in opposite directions

2: Compare half-arrays

3: Send larger item in each pair to the right

Perform 2 & 3 recursively on each half

Example Bitonic Sorting Network



Example Run

3

7

4

8

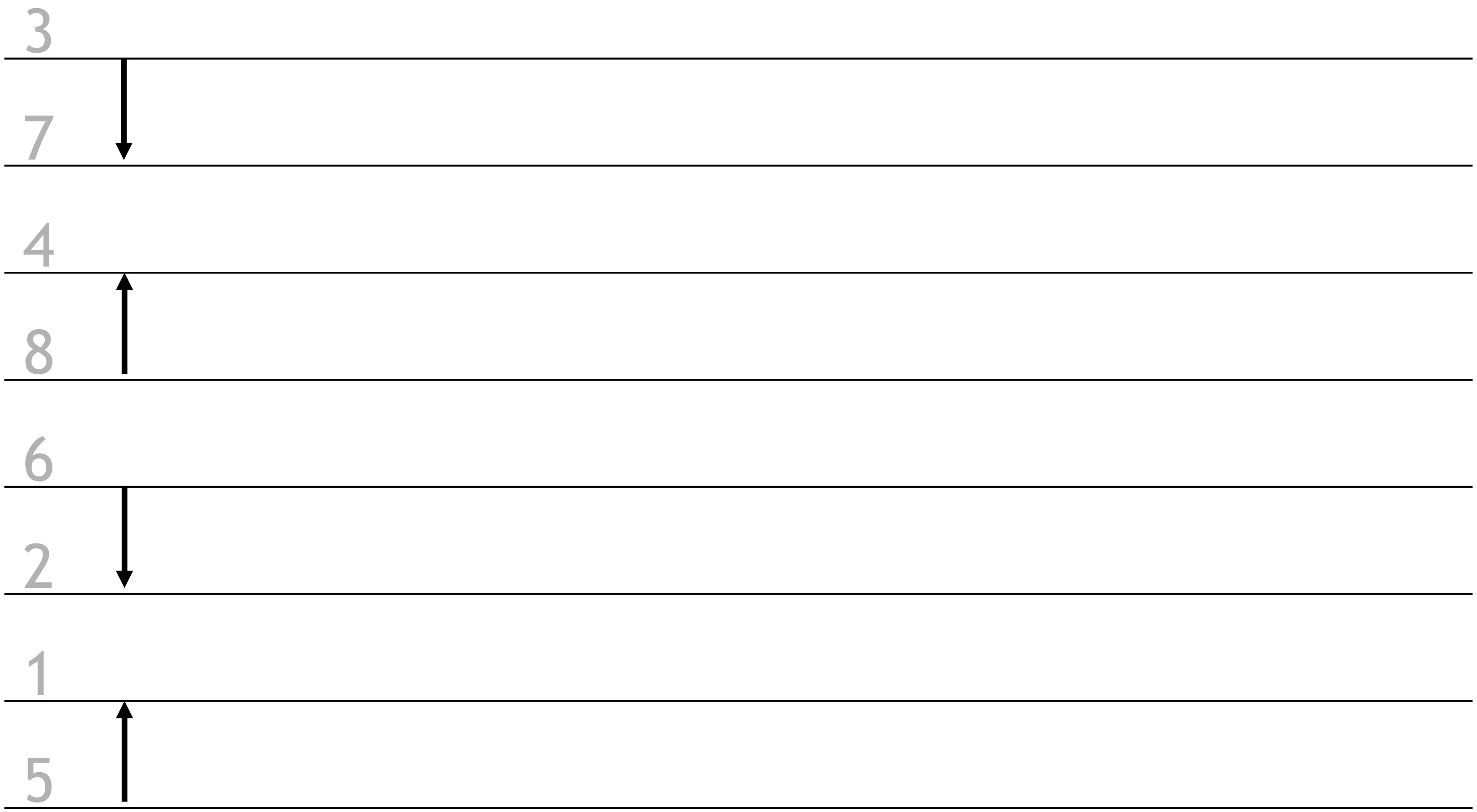
6

2

1

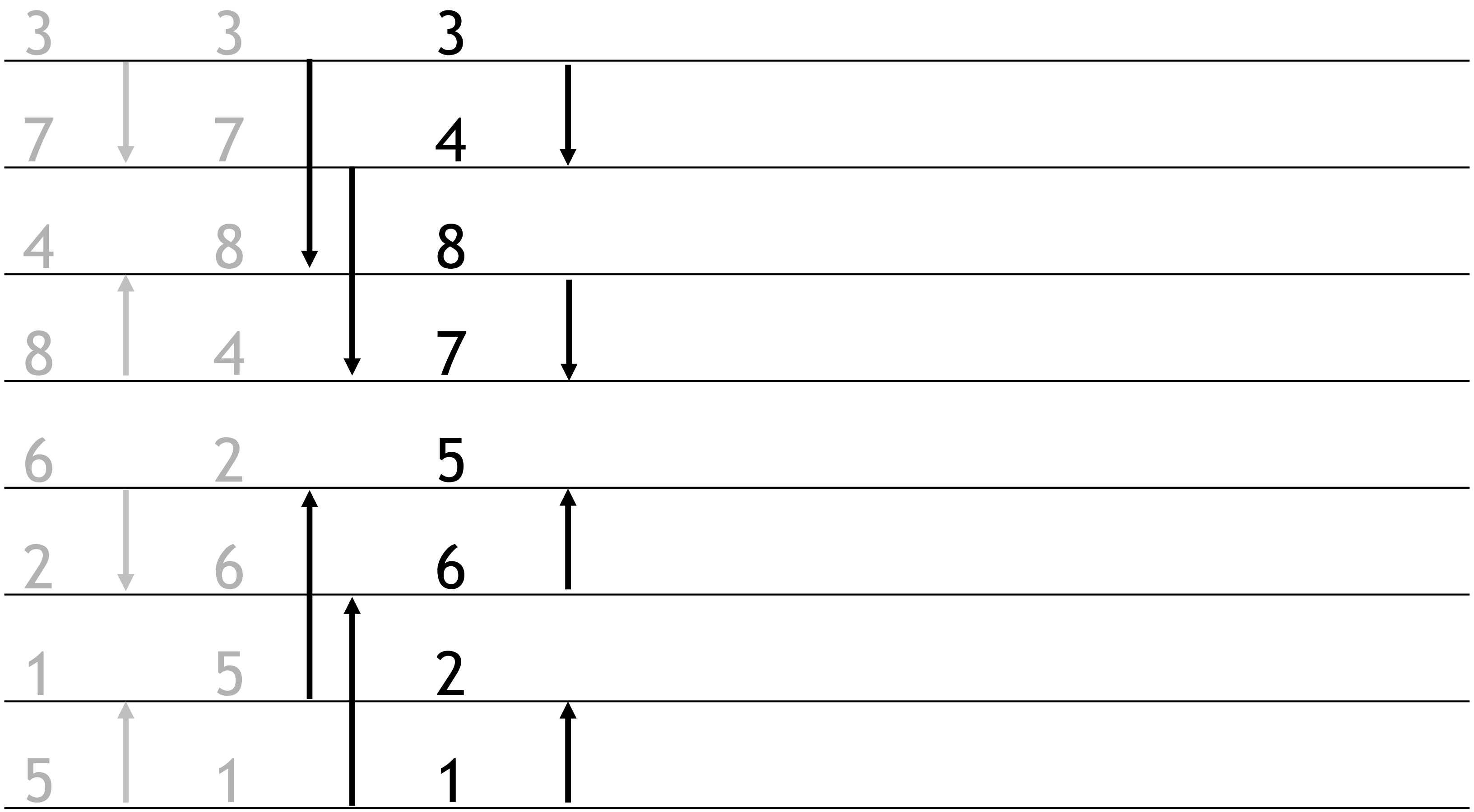
5

8x monotonic lists: (3) (7) (4) (8) (6) (2) (1) (5)
4x bitonic lists: (3,7) (4,8) (6,2) (1,5)

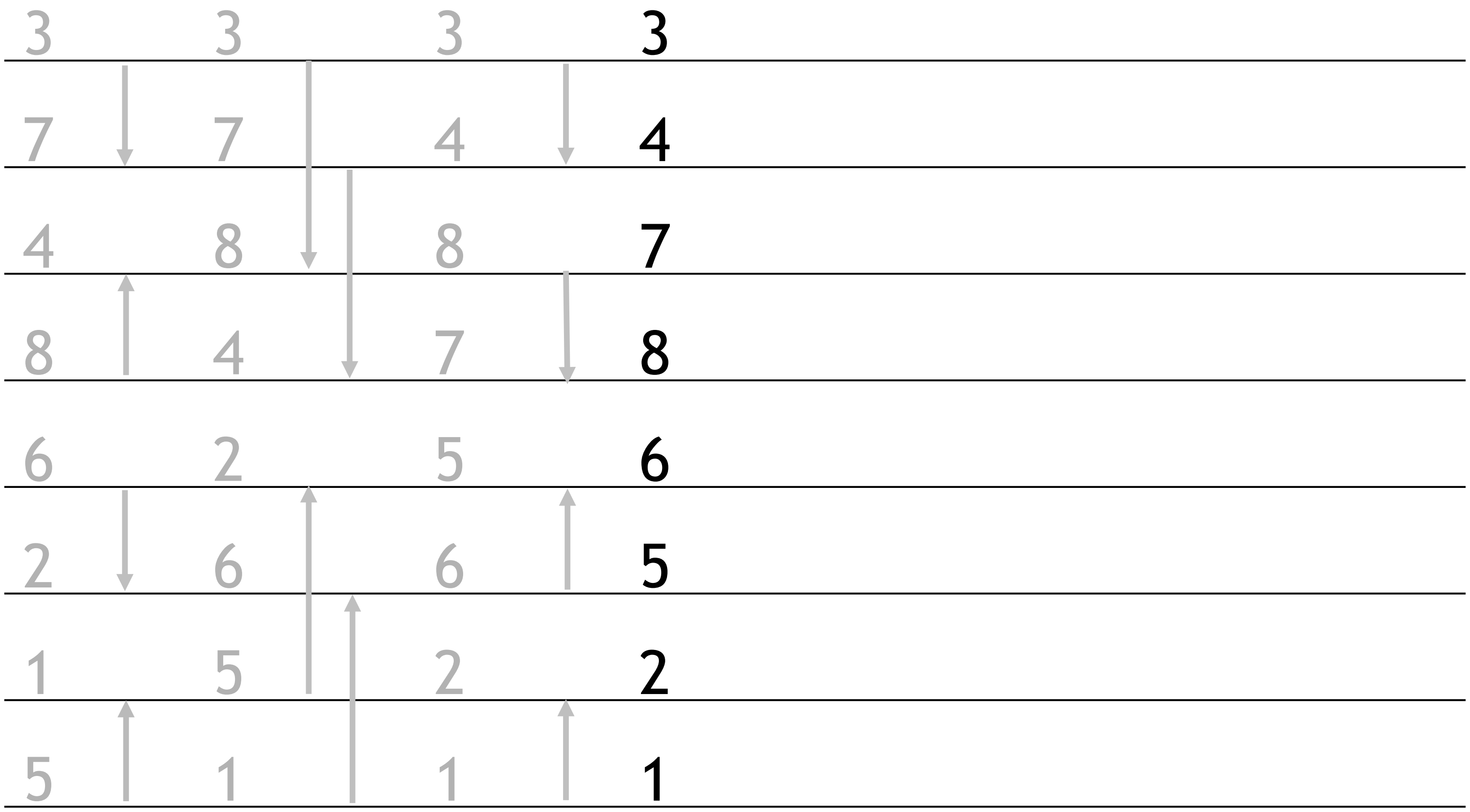


Sort the bitonic lists

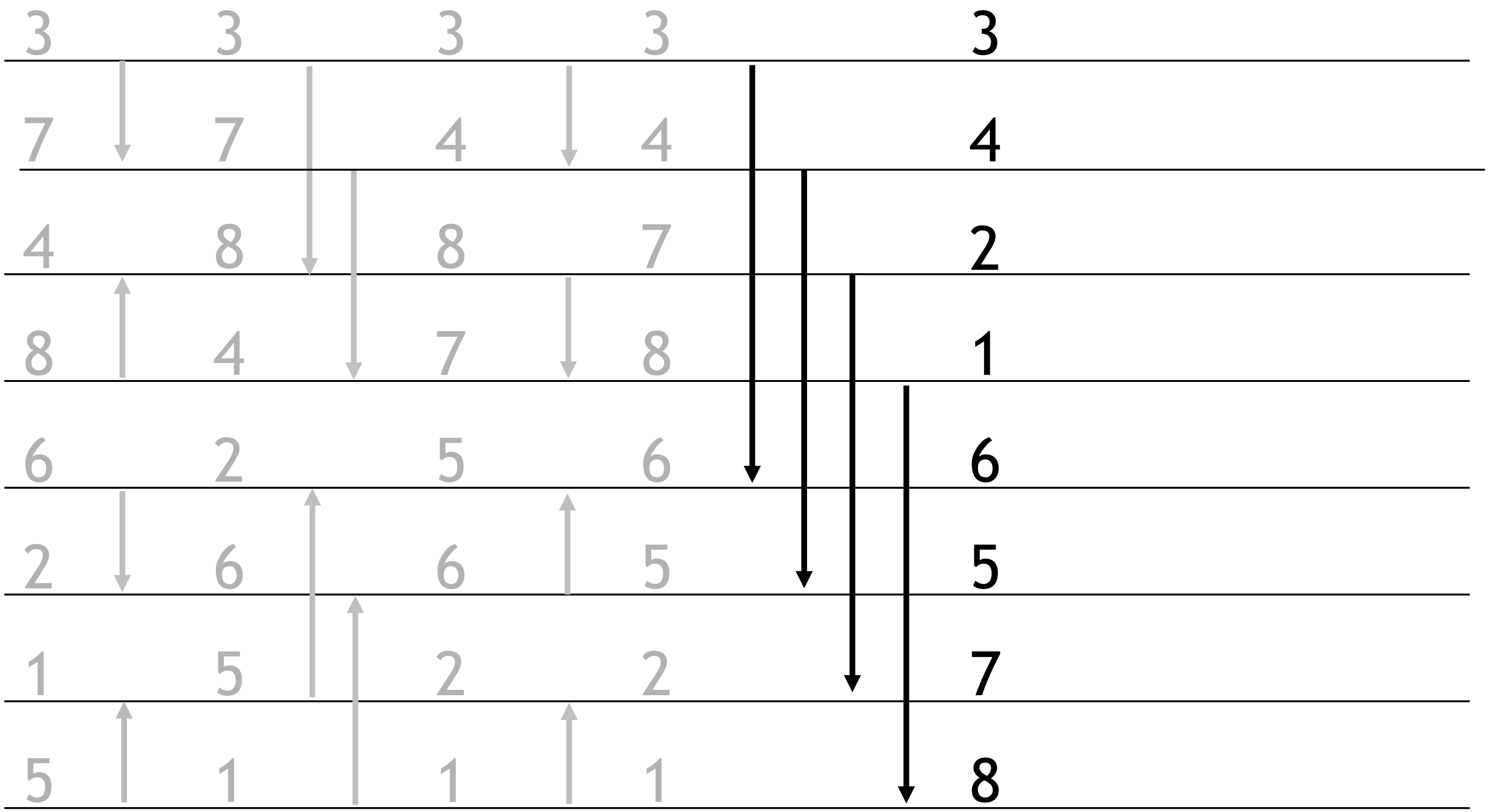
4x monotonic lists: (3,7) (8,4) (2,6) (5,1)
2x bitonic lists: (3,7,8,4) (2,6,5,1)



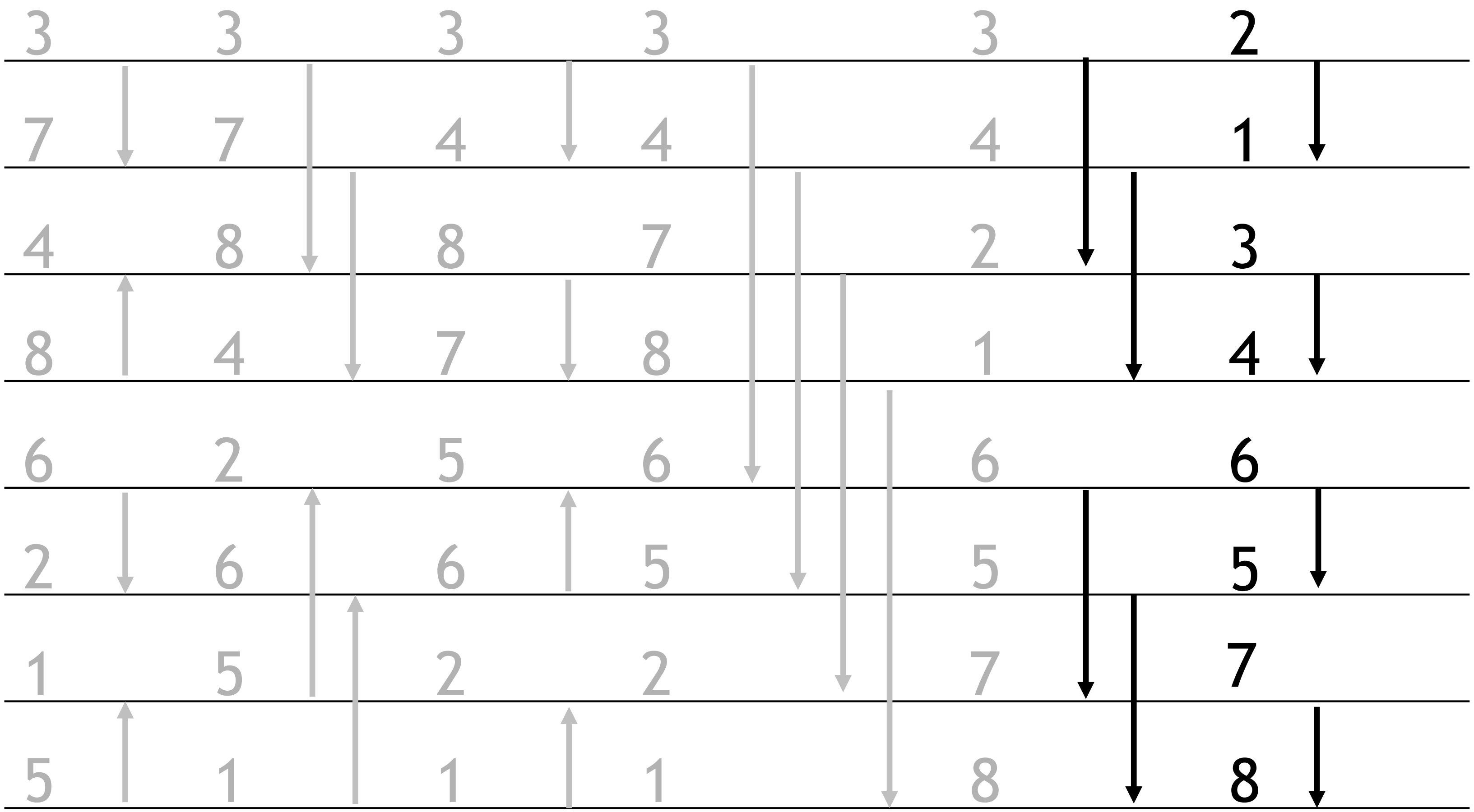
Sort the bitonic lists



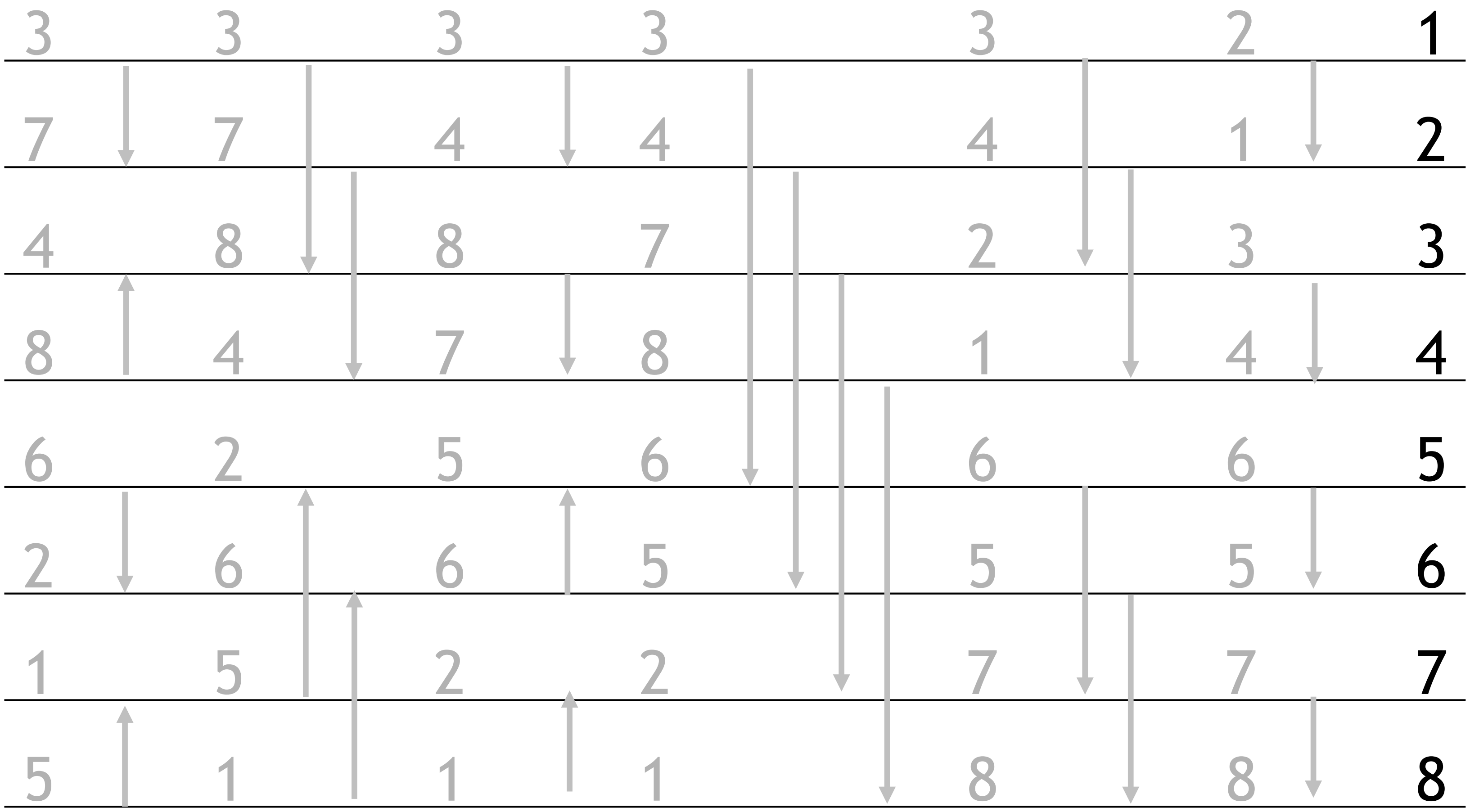
2x monotonic lists: (3,4,7,8) (6,5,2,1)
1x bitonic list: (3,4,7,8, 6,5,2,1)



Sort the bitonic lists



Sort the bitonic lists


3 3 3 3 3 2 1
7 7 4 4 4 1 2
4 8 8 7 2 3 3
8 4 7 8 1 4 4
6 2 5 6 6 6 5
2 6 6 5 5 5 6
1 5 2 2 7 7 7
5 1 1 1 8 8 8

Done!

Complexity of the Bitonic Sorter

- Depth complexity (= parallel time complexity):
  - Bitonic merger: $O\left(\log n\right)$
  - Bitonic sorter: $O\left(\log^2 n\right)$
- Work complexity of bitonic merger:
  - Means number of comparators $C(n)$ here
  - Recursive equation for C: $C(n) = 2C(\frac{n}{2}) + \frac{n}{2}$, with $C(2) = 1$
  - Overall $C(n) = \frac{1}{2} n \log n$
- Remark: there must be some redundancy in the sorting network, because we know (from merge sort) that n comparisons are sufficient for merging two sorted sequences
- Reason for the redundancy?
  $\longrightarrow$ because the network is data-independent!

Remarks on Bitonic Sorting

- Probably most well-known parallel sorting algo / network
- Fastest algorithm for "small" arrays (or, is it?)

- Lower bound on depth complexity is

$$\frac{O\left(n \log n\right)}{n} = O\left(\log n\right)$$

assuming we have n processors

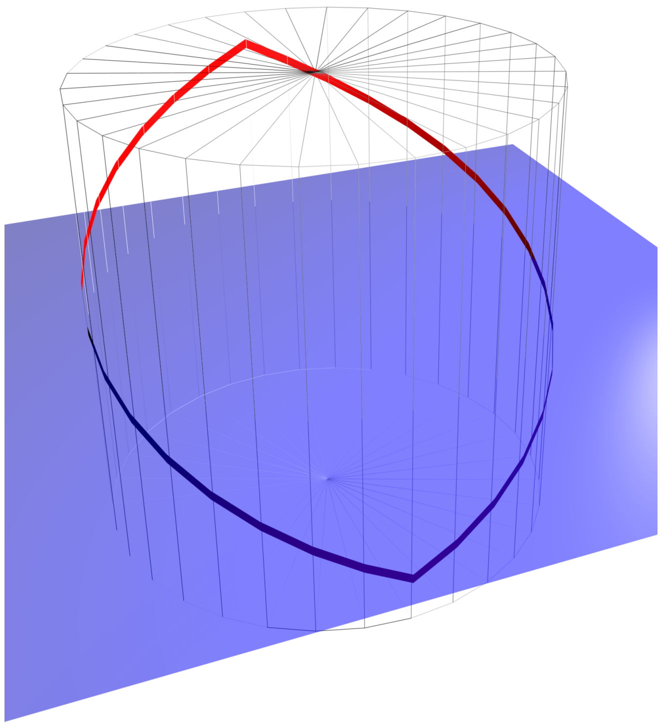
- A nice property: comparators in a bitonic sorter network only ever compare lines whose label (= binary line number) differs by exactly one bit!
- Consequence for the implementation:
  - One kernel for all threads
  - Each thread only needs to determine
    which bit of its own thread ID to "flip"
    $\longrightarrow$ gives the "other" line with which to compare
- Hence, bitonic sorting is sometimes pictured as well suited for a log(n)-dimensional hypercube parallel architecture:
  - Each node of the hypercube = one processor
  - Each processor is connected directly to log(n) many other processors
  - In each step, each processor talks to one of its direct neighbors

Adaptive Bitonic Sorting

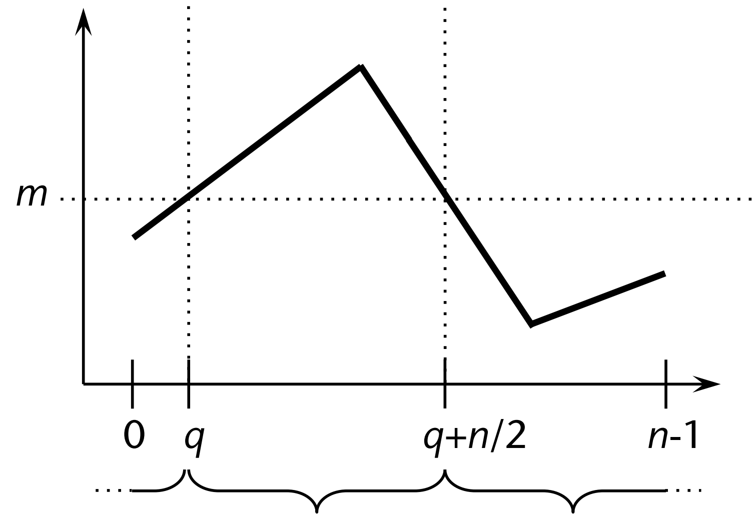
- Theorem 2:
  Let **a** be a bitonic sequence.
  Then, we can always find an index q such that

$$\max\left(a_q, \ldots, a_{q+\frac{n}{2}-1}\right) \leq \min\left(a_{q+\frac{n}{2}}, \ldots, a_{q-1}\right)$$

- Sketch of proof:
  - Assume (for sake of simplicity) that all elements in **a** are distinct
  - Imagine the bitonic sequence as a "line" on a cylinder
  - Since **a** is bitonic ⟶ only two inflection points ⟶ each horizontal plane cuts the sequence at exactly 2 points, and both sub-sequences are contiguous
  - Use the median m as "cut plane" ⟶ each sub-sequence has length $n/2$, and max("lower sequ.") ≤ m ≤ min("upper sequ.")
  - These must be L**a** and U**a** , resp.
  - The index of m is exactly index q in Theorem 2



- Visualization of the theorem:



- Theorem 3:
  Any bitonic sequence **a** can be partitioned into four sub-sequences $(\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3, \mathbf{a}^4) = \mathbf{a}$, such that

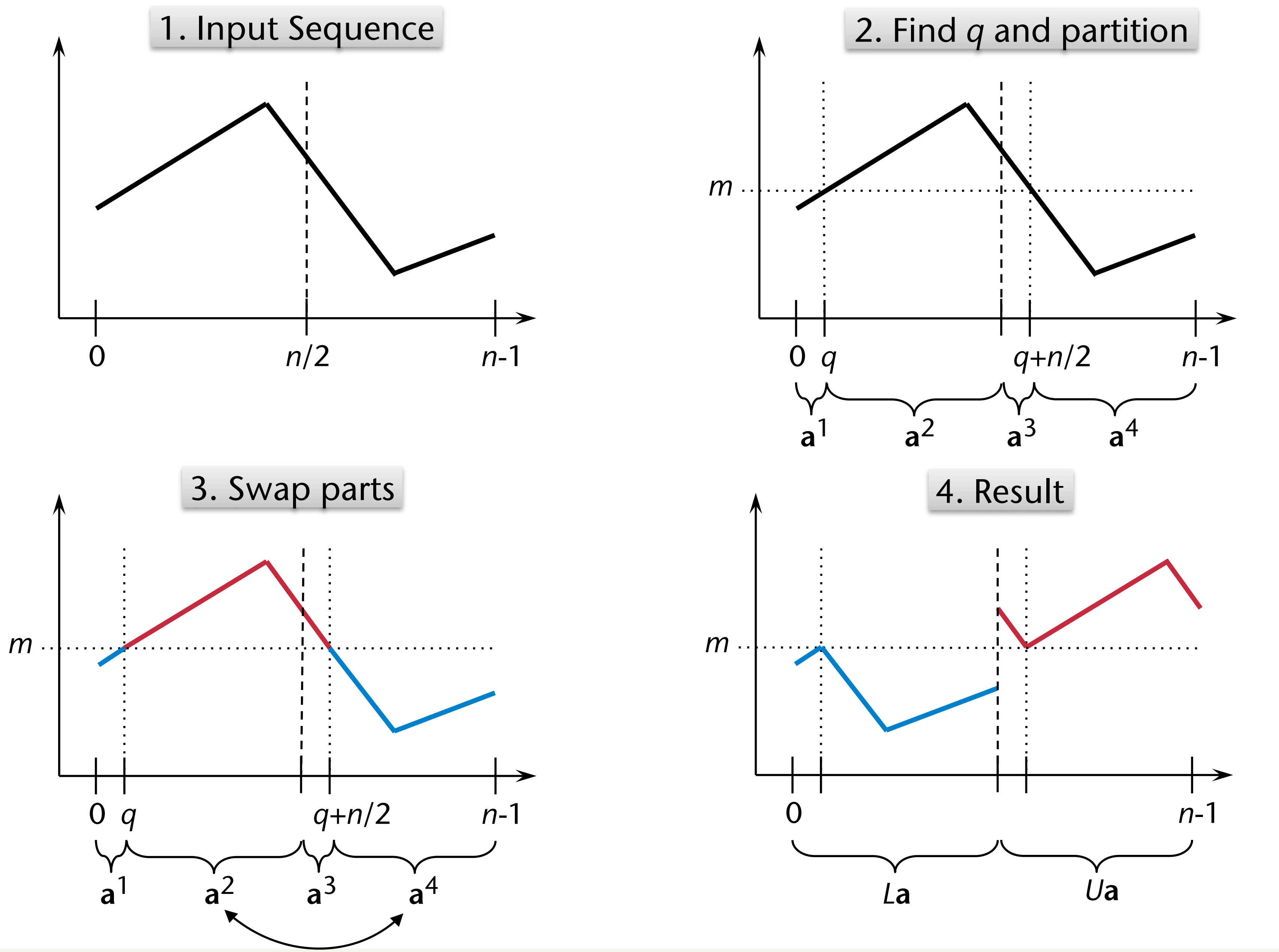
$$|\mathbf{a}^1| + |\mathbf{a}^2| = |\mathbf{a}^3| + |\mathbf{a}^4| = \frac{n}{2} \quad , \quad |\mathbf{a}^1| = |\mathbf{a}^3| \quad , \quad |\mathbf{a}^2| = |\mathbf{a}^4|$$

and

either $\quad (L\mathbf{a}, U\mathbf{a}) = (\mathbf{a}^1, \mathbf{a}^4, \mathbf{a}^3, \mathbf{a}^2) \qquad$ or $\quad (L\mathbf{a}, U\mathbf{a}) = (\mathbf{a}^3, \mathbf{a}^2, \mathbf{a}^1, \mathbf{a}^4)$

Optional

Visual "Proof"



Complexity

- Finding the median in a bitonic sequence $\longrightarrow \log n$ steps
- Remark: this algorithm is no longer data-independent!
- Depth complexity: $\longrightarrow$ exercise
- Work complexity of adaptive bitonic merger:
  - Number of comparisons

$$C(n) = 2C(\frac{n}{2}) + \log(n) = \sum_{i=0}^{k-1} 2^i \log(\frac{n}{2^i}) = 2n - \log n - 2$$

  - This is optimal!
  - Need a trick to avoid actually copying the subsequences
    - Otherwise the total complexity of a BM(n) would be $O(n \log n)$
  - Trick = *bitonic tree* (see orig. paper for details)

Optional

How to find the median in a bitonic sequence

- We have

$$\text{median}(a) = \min(U\mathbf{a})$$

or

$$\text{median}(a) = \max(L\mathbf{a})$$

(depending on the definition of the median)

- Finding the minimum in a bitonic sequence takes $\log(n)$ steps

Optional

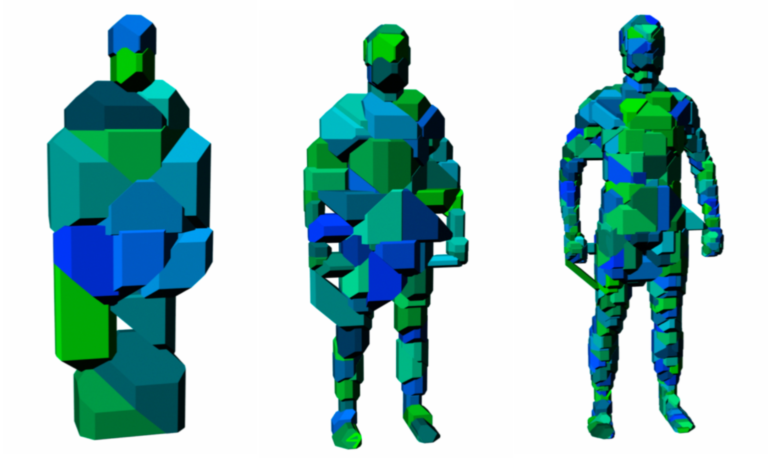
Topics for Master Theses

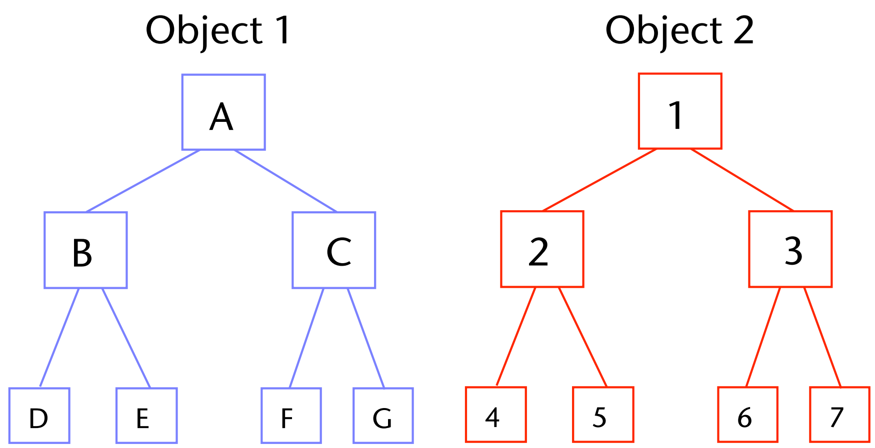
- Lots of different parallel sorting algorithms
- Our implementation of Adaptive Bitonic Sorting is ancient (on an ancient architecture [shaders ...])
- Do you love algorithms?
  - Thinking about them?
  - Proving properties?
  - Implementing them super-fast?
- Then we should talk about a possible master's thesis topic! 😀

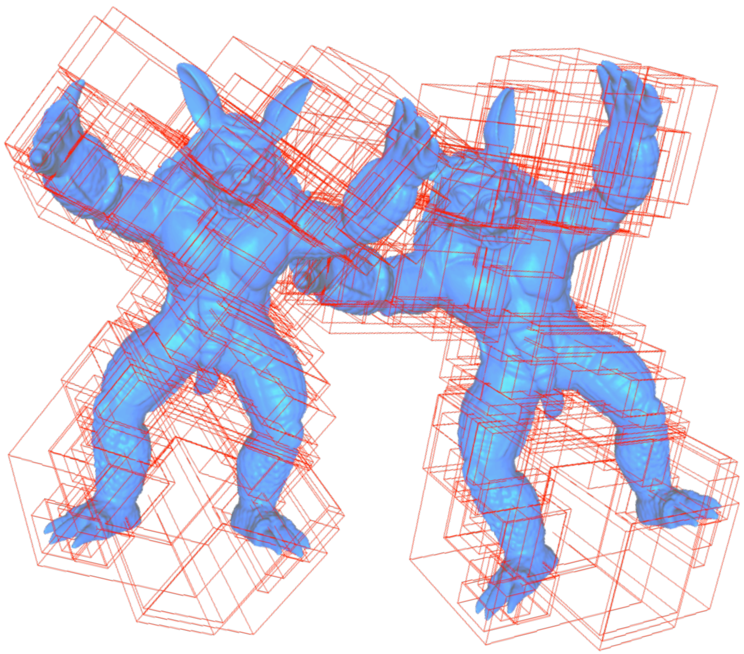
Application: BVH Construction

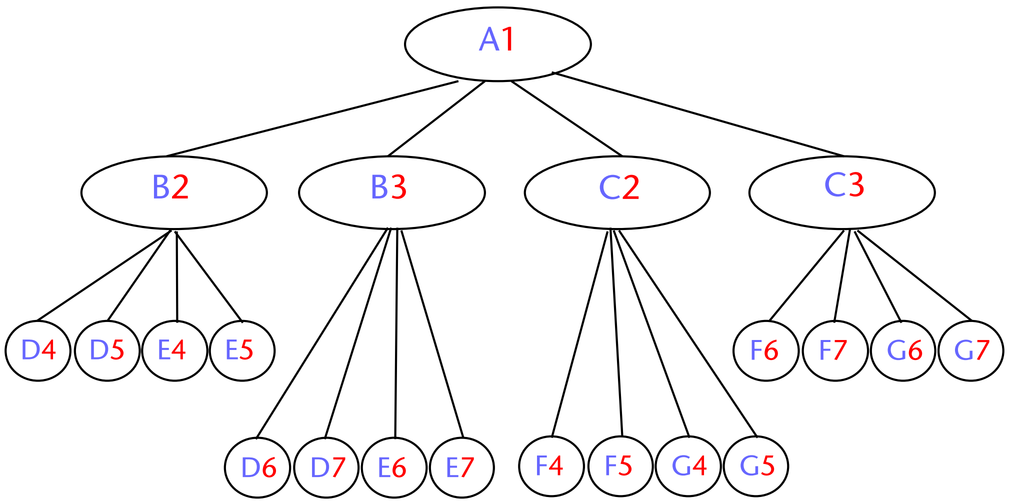
- Bounding volume hierarchies (BVHs): very important data structure for accelerating geometric queries
- Applications: ray-scene intersection, collision detection, spatial data bases, etc.
  - Database people call it often "R-tree" …

BVHs in Collision Detection





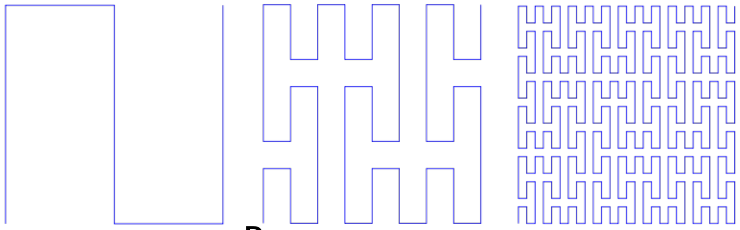




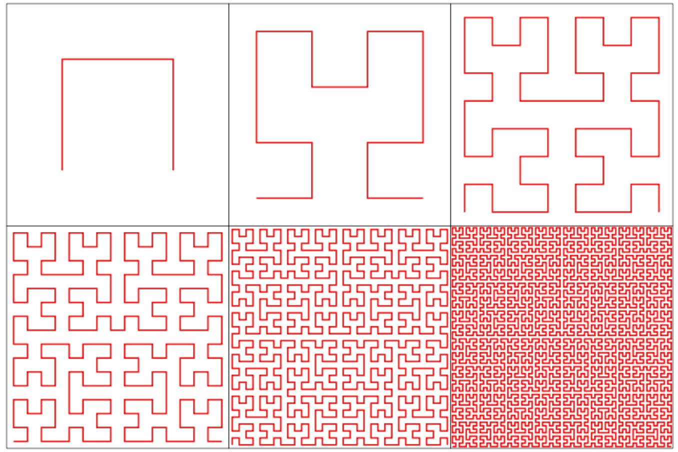
Parallel Construction of BVHs

- First idea: linearize 3D points/objects by space-filling curve
- Definition curve:
  A curve (with endpoints) is a continuous function with *domain* in the unit interval [0,1] and *range* in some *d*-dimensional space.
- Definition space-filling curve:
  A space-filling curve is a curve with a range that covers the entire 2-dimensional unit square (or, more generally, an *n*-dimensional hypercube).

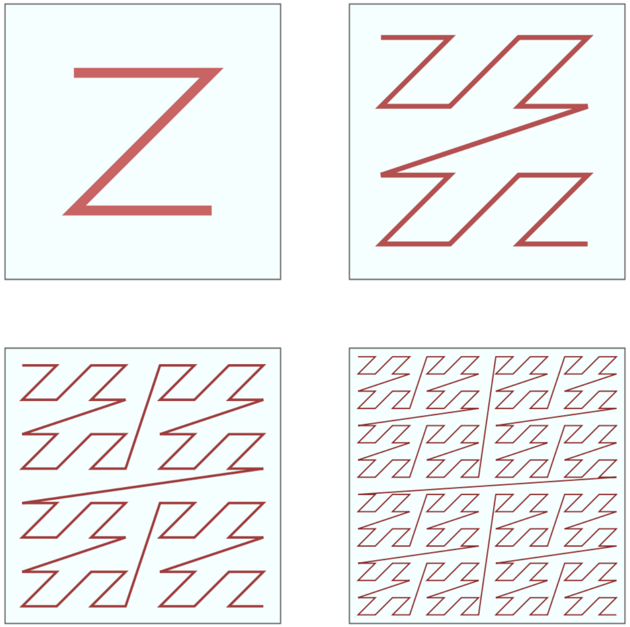
Examples of Space-Filling Curves



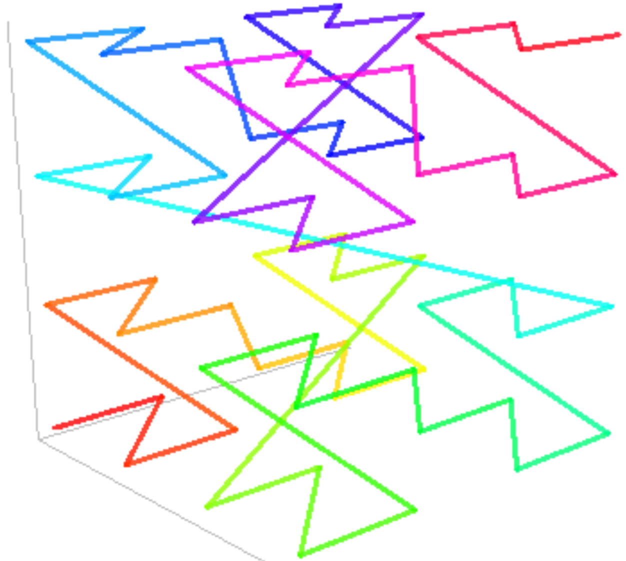
Peano curve



Hilbert curve

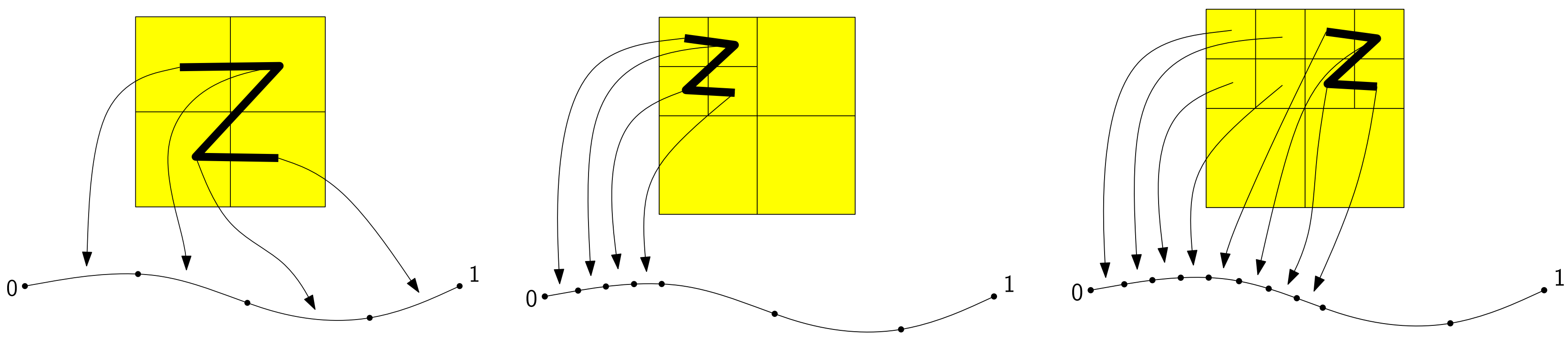


Z-order curve
(a.k.a. Morton curve)



Z-order curve in 3D

- Benefit: a space-filling curve gives a mapping from the unit square to the unit interval
  - At least, the limit curve does that …



  - We can construct a "space-filling" curve only on some specific (recursion) level, i.e., in practice space-filling curves are never really *space-filling*