Introduction to data-parallelism

and OpenCL

Sylvain Lefebvre - INRIA

Sorting

- Sort is everywhere

- Sequential sorts:
  - Selection sort $O(N^2)$
  - Heapsort $O(N \log N)$
  - Quicksort $O(N \log N)$

- Known best complexity: $O(N \log N)$

Parallel sort

- How can we sort in parallel?
  - For now, not concerned about performance
- Most sorting algorithms:
  - Number of comparisons depends on data
  - Example: Quicksort
    - Average O(N log N)
    - Worst case O(N^2)
- Data dependency makes parallelism complex

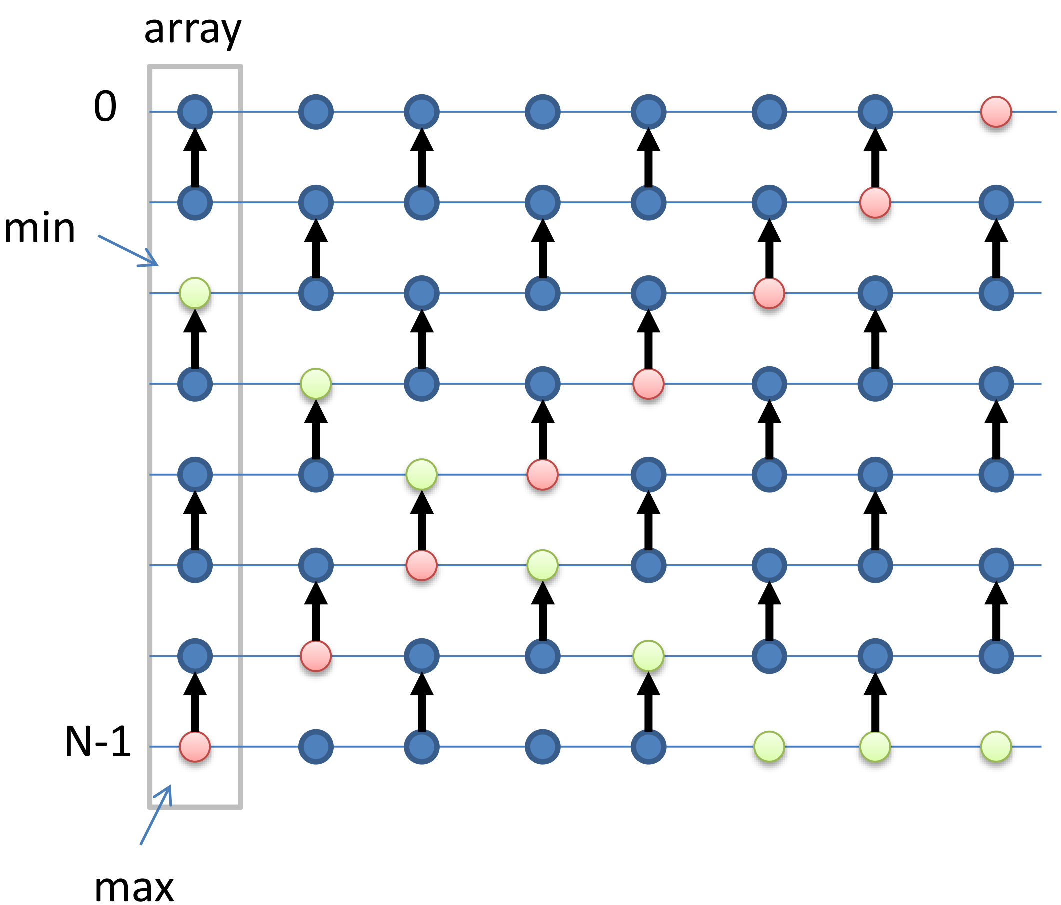
Sorting networks

- **Data-independent** sorting algorithms
  - Whatever the data, the same operations occur

- There are several such algorithms:
  - Odd-even sort
  - Bitonic merge sort
  - …

Odd-even sort

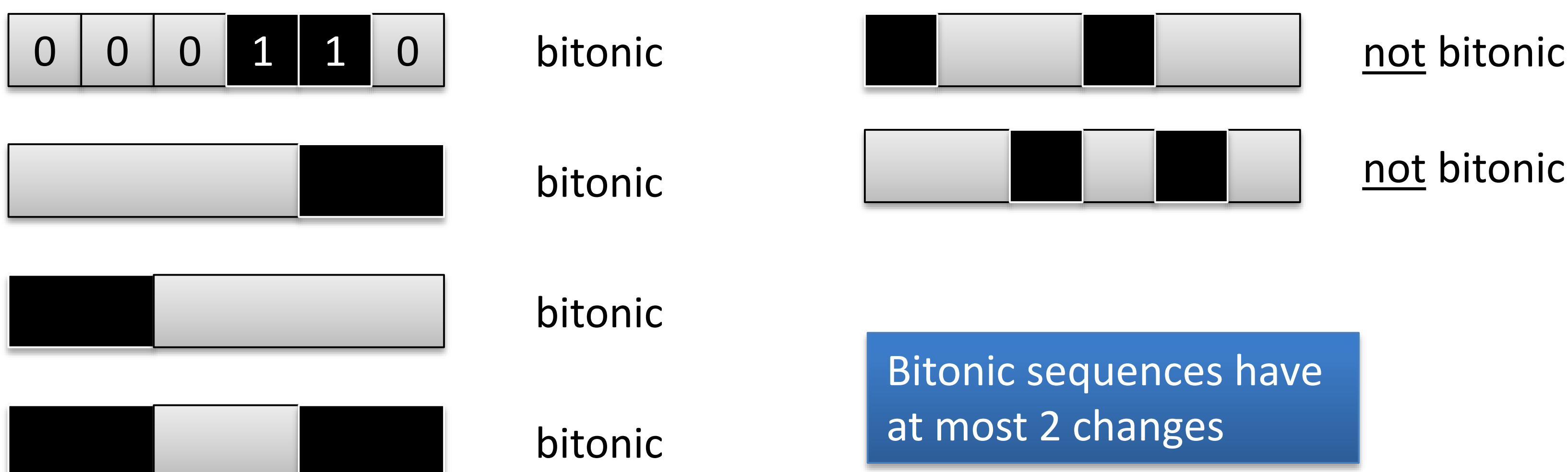
- Simple, first approach:



$O(N^2)$

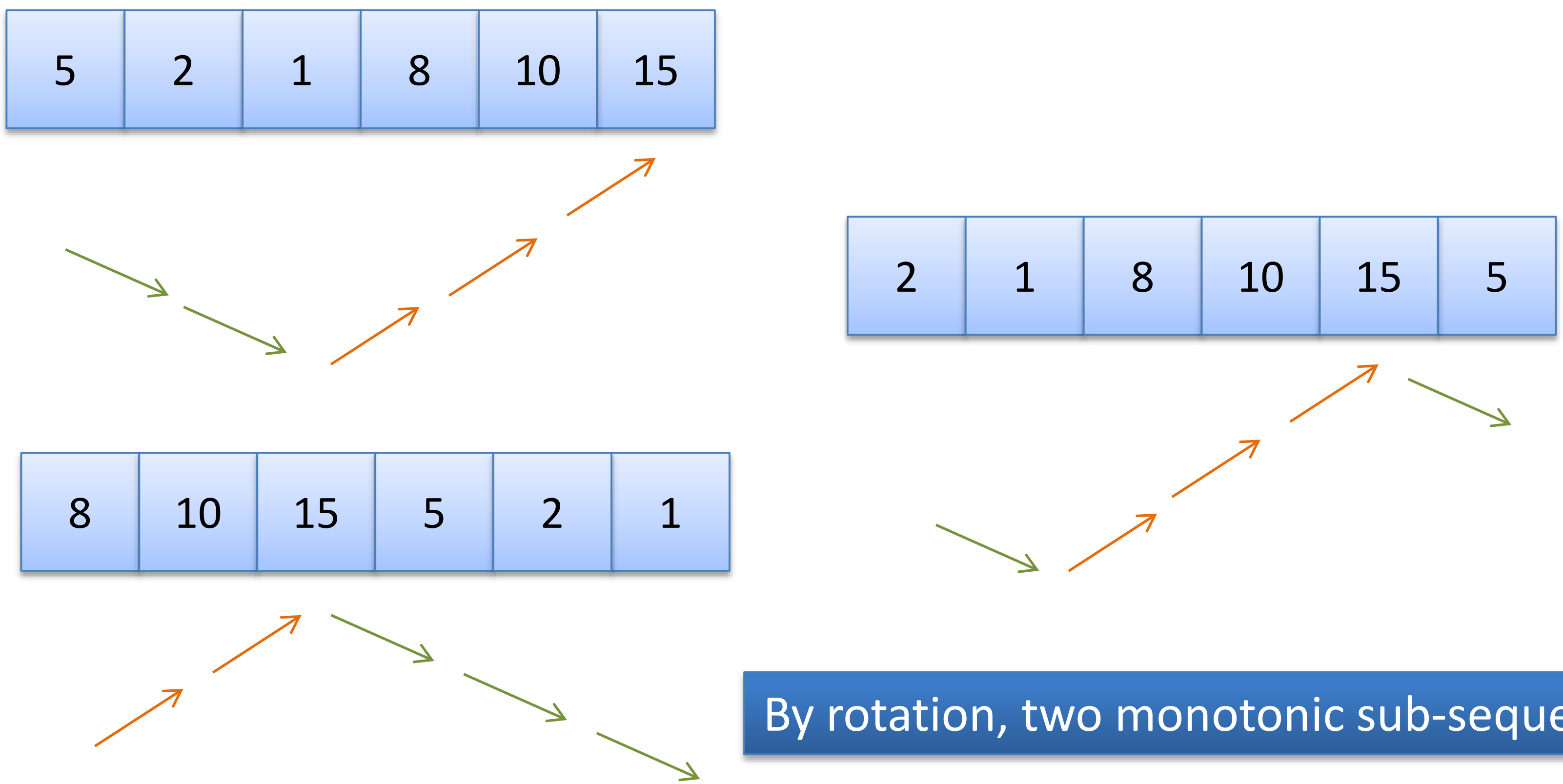
Bitonic merge sort

- Better complexity: $O(\ N \log^2 N\ )$
- Based on ***bitonic*** sequences
  - Let's consider we only have values in {0,1}



Bitonic merge sort

- Bitonic sequences of numbers:



By rotation, two monotonic sub-sequences

0-1 principle

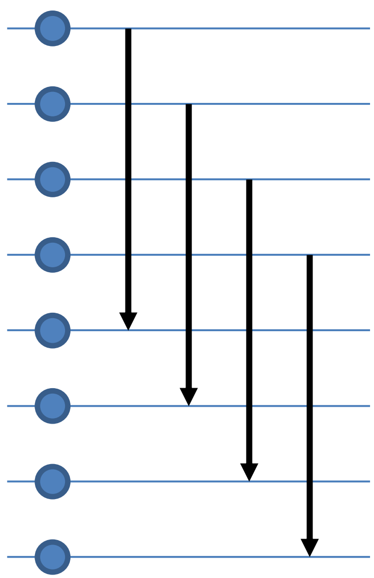
Theorem:

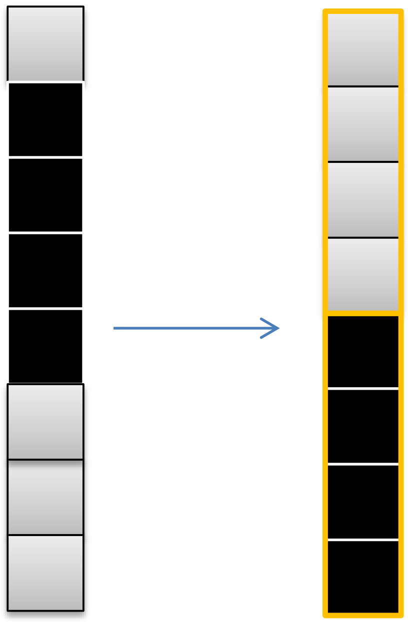
If a sorting network sorts every sequence of 0's and 1's, then it sorts every arbitrary sequence of values.

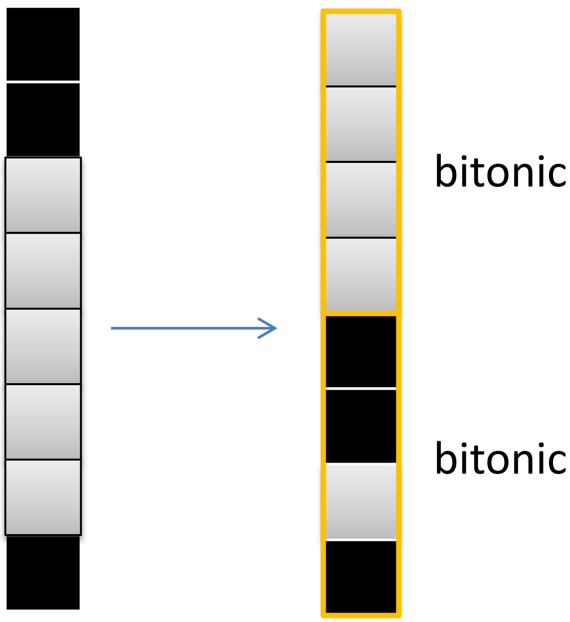
[Knuth 73]

Comparator

- Compares two halves of a bitonic 0-1 sequence

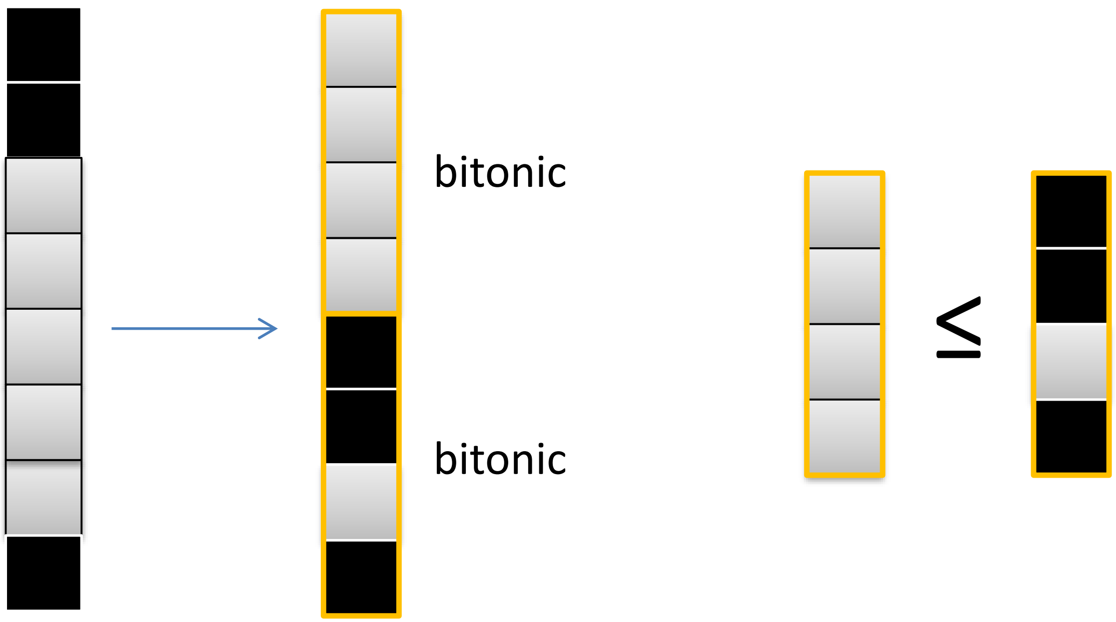




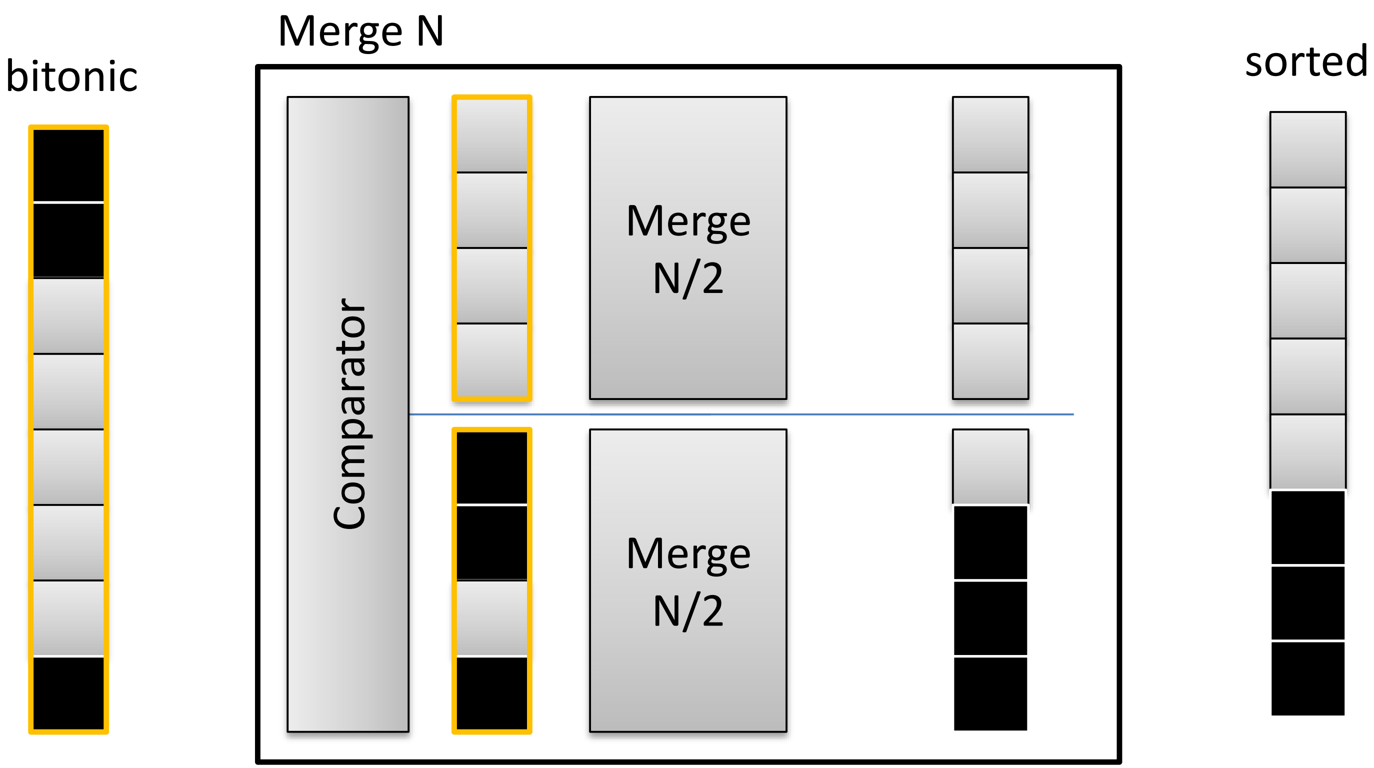


Comparator

- The two sub-sequence are bitonic
- The first is less-or-equal to the second

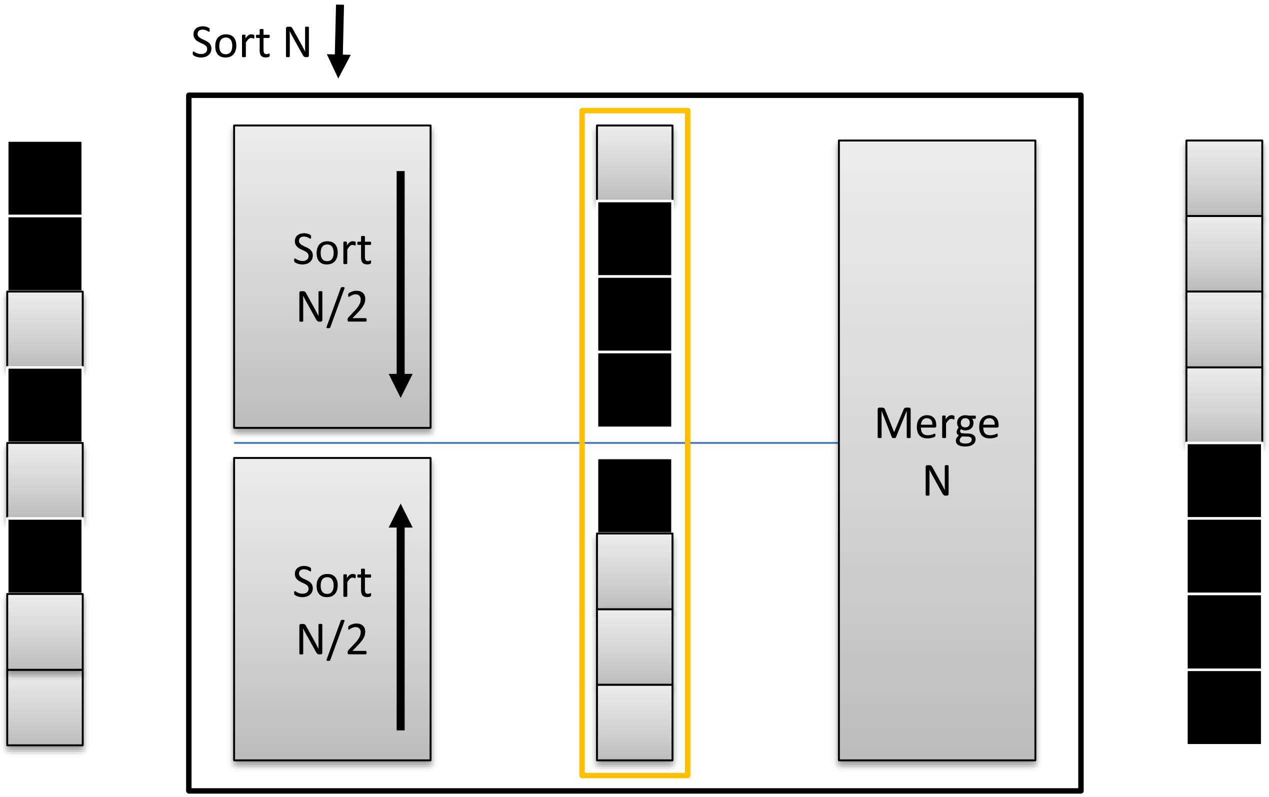


Bitonic merge



Bitonic sort

- Divide and conquer approach (recursive)
  - Two operations: **Sort** and **Merge**



Implementation

- Trivial cases:
  - Sort and Merge on N = 1

- Recursion (CPU):

```
Sort( T, L,R, d) {
  if ( R-L+1 > 1 ) {
    Sort( T, L,(R+L)/2,   ASC  )
    Sort( T, (R+L)/2+1,R, DESC )
    Merge( T , L,R, d)
  }
}
```

```
Merge( T, L,R, d) {
  if ( R-L+1 > 1 ) {
    Compare( T, L,R ,d )
    Merge( T , L,(L+R)/2   ,d )
    Merge( T , (L+R)/2+1,R ,d )
  }
}
```

Implementation

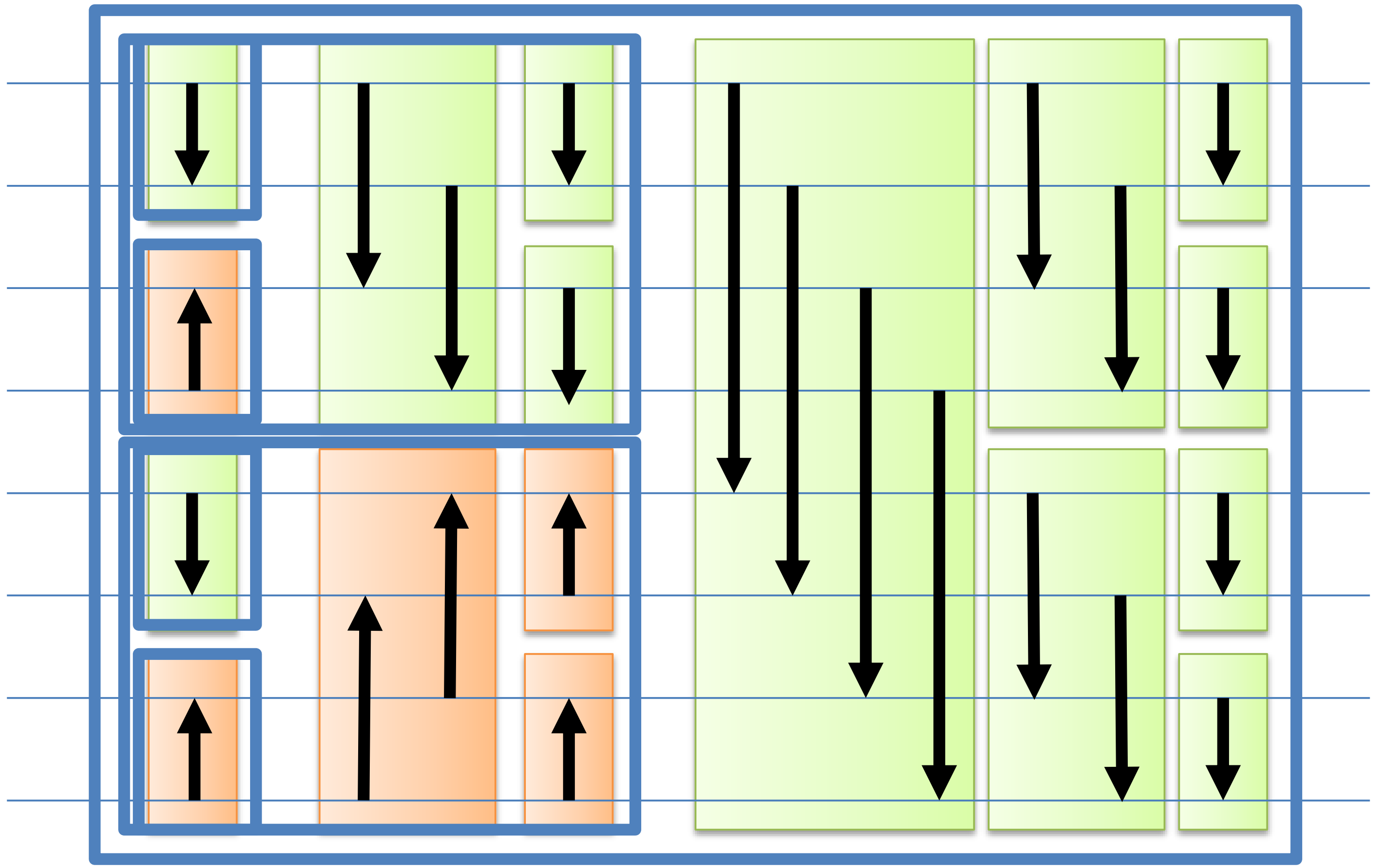
```
Compare( T, R,L, d )
  k = (R-L+1)/2
  for (i = 0 ; i < k ; i++) {
    if (d == ASC) {
      if ( T[L+i] < T[L+i+k] )
        swap(T,L+i,L+i+k)
    } else {
      if ( T[L+i] > T[L+i+k] )
        swap(T,L+i,L+i+k)
    }
  }
```

```
Sort( T, L,R, d) {
  if ( R-L+1 > 1 ) {
    Sort( T, L,(R+L)/2,   ASC  )
    Sort( T, (R+L)/2+1,R, DESC )
    Merge( T , L,R, d)
  }
}
```

```
Merge( T, L,R, d) {
  if ( R-L+1 > 1 ) {
    Compare( T, L,R ,d )
    Merge( T , L,(L+R)/2    ,d )
    Merge( T , (L+R)/2+1,R ,d )
  }
}
```

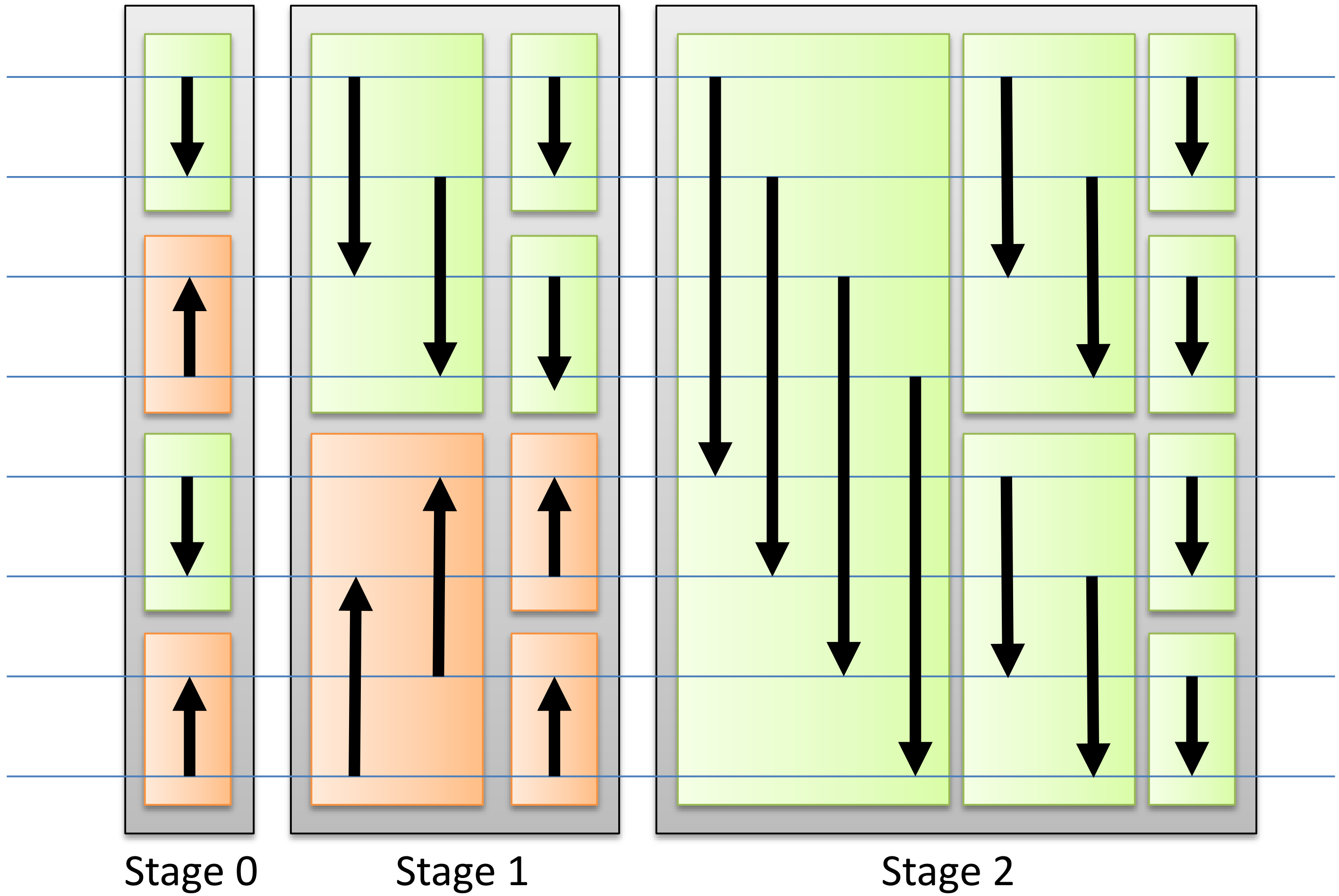
Implementation in OpenCL

- Consider global comparison pattern



Implementation in OpenCL

- Log(N) stages



Number of comparisons

- log(N) stages
- Stage i has i columns
- Number of columns:

  Sum(i=0…log N) (i) = log N (log N + 1) / 2

- Number of comparisons:

  N/2 * (log N (log N + 1) / 2) ➔ O(N $\log^2$ N)

Let’s practice

- Odd-even sort
  - Global memory

- Bitonic sort
  - First, recursive on CPU
  - Second, on the GPU (global memory)

Credits

- http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter46.html
- http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm