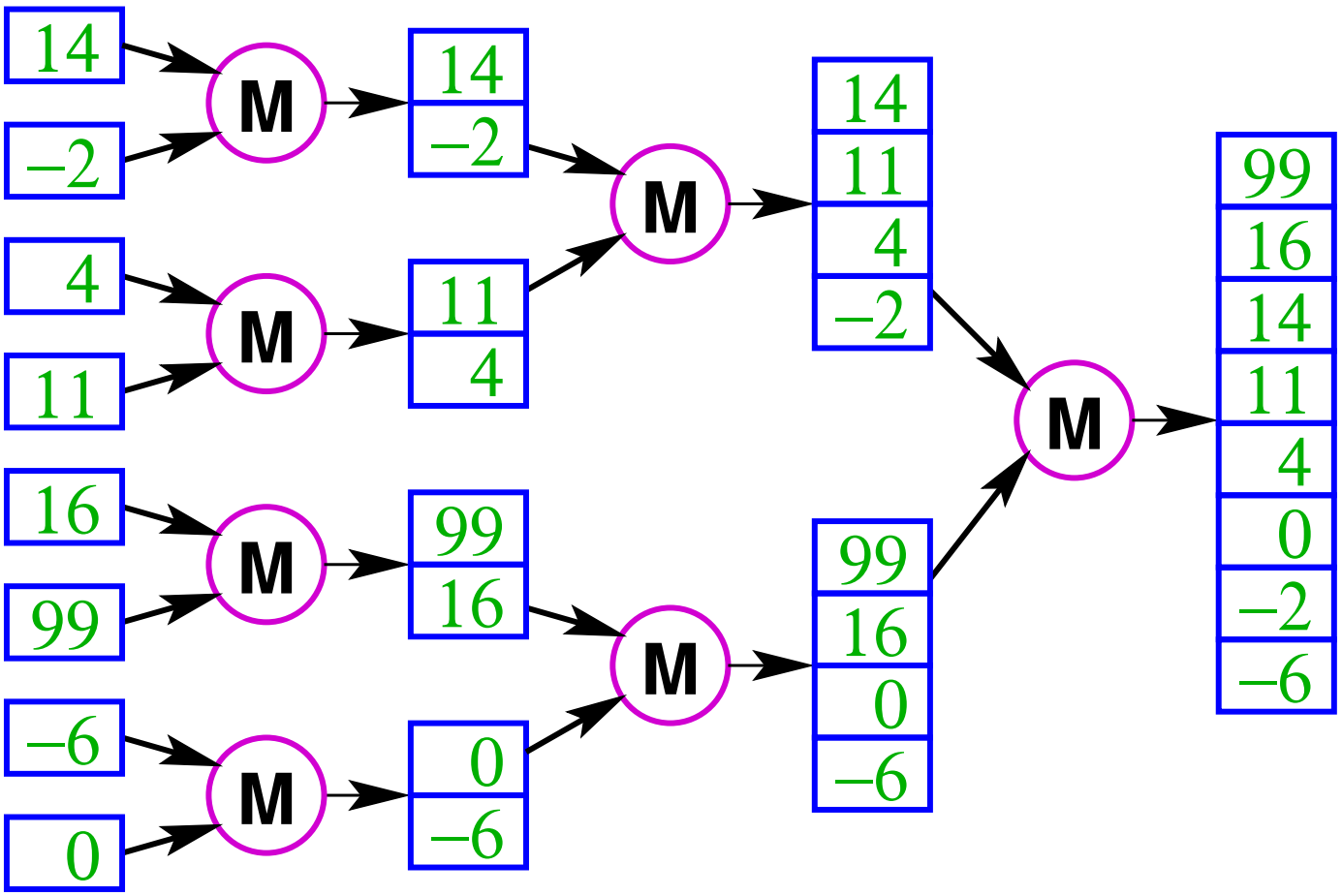
Sorting Networks

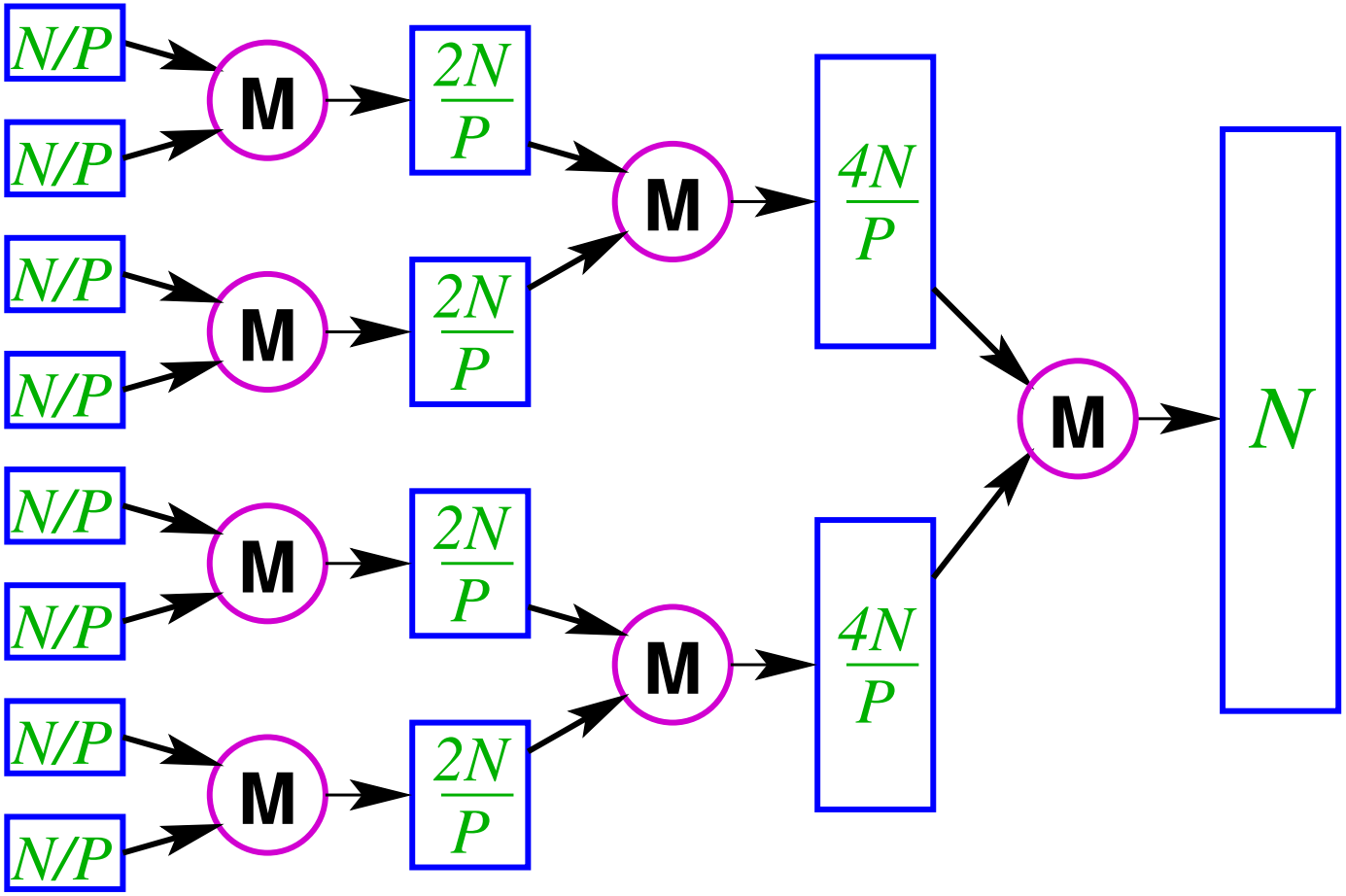
Mark Greenstreet

CpSc 418 – Mar. 9, 2016

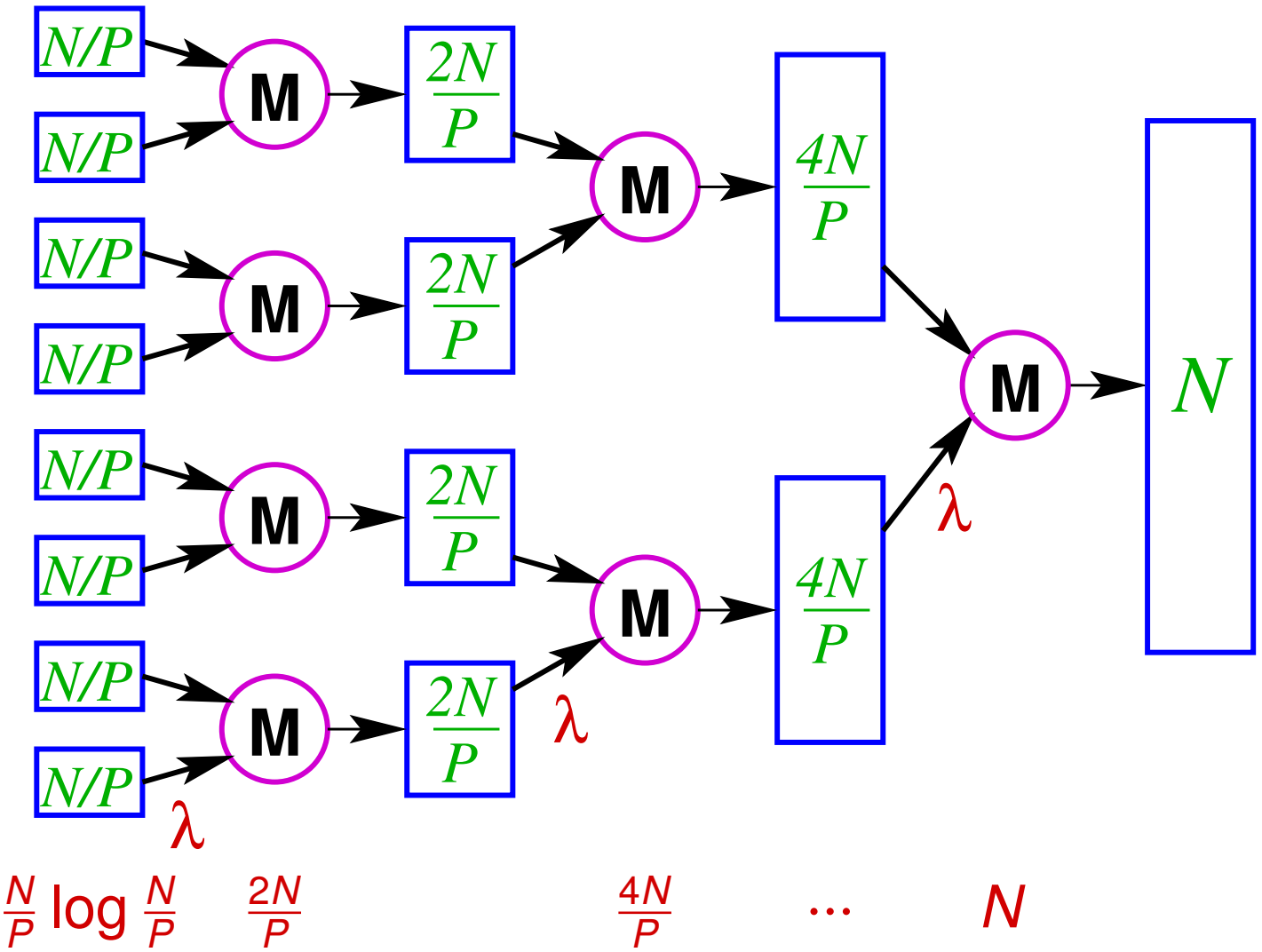
Parallelizing Mergesort



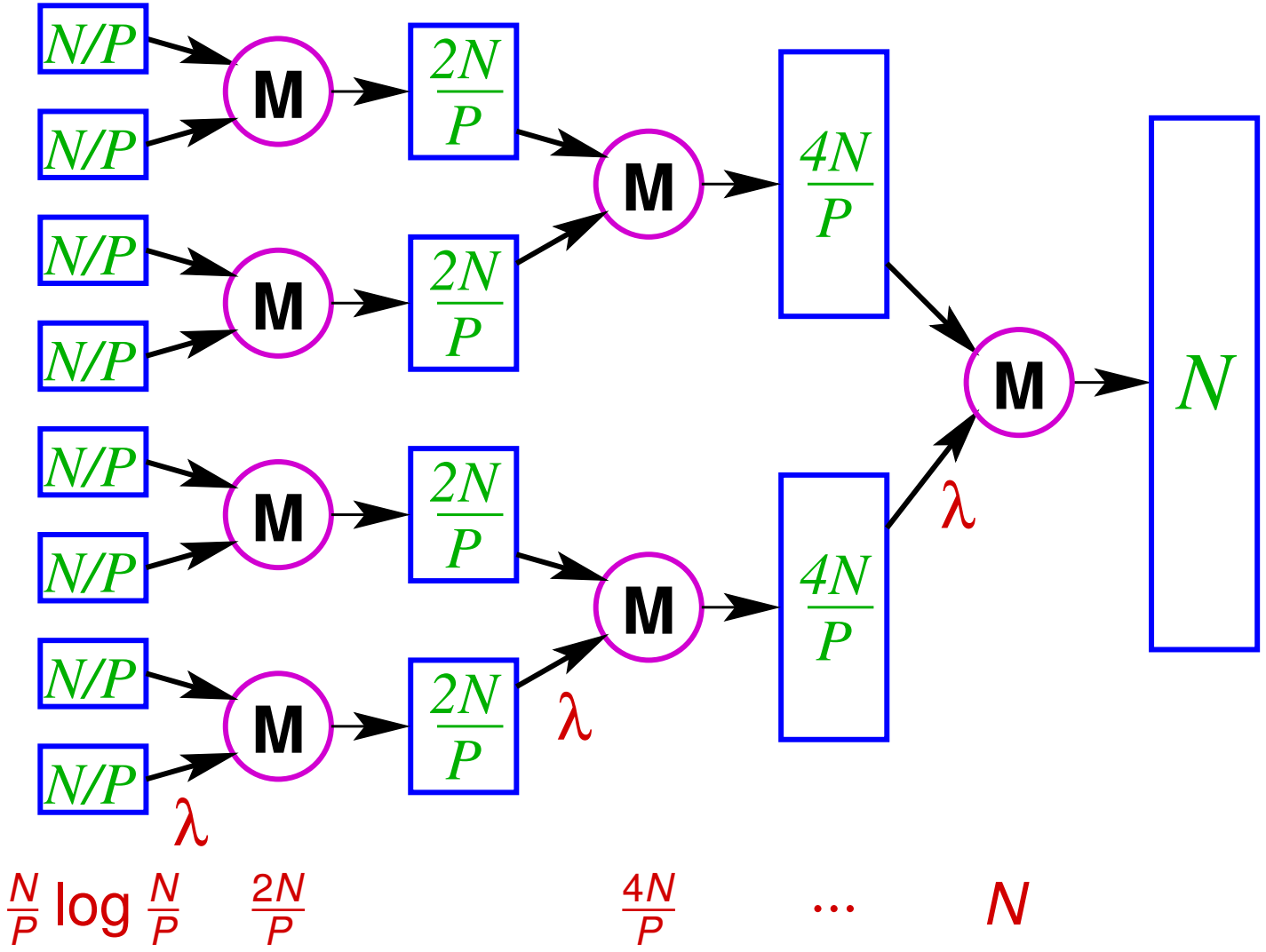
Parallelizing Mergesort



Parallelizing Mergesort



Parallelizing Mergesort



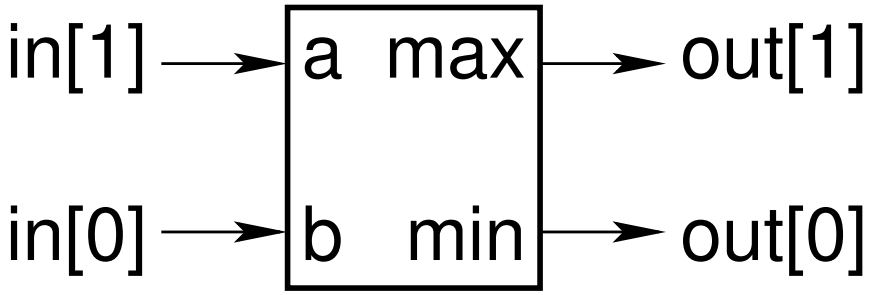
Total time: $\frac{N}{P}(\log N + 2(P-1) - \log P) + (\log P)\lambda$

Parallelizing Quicksort

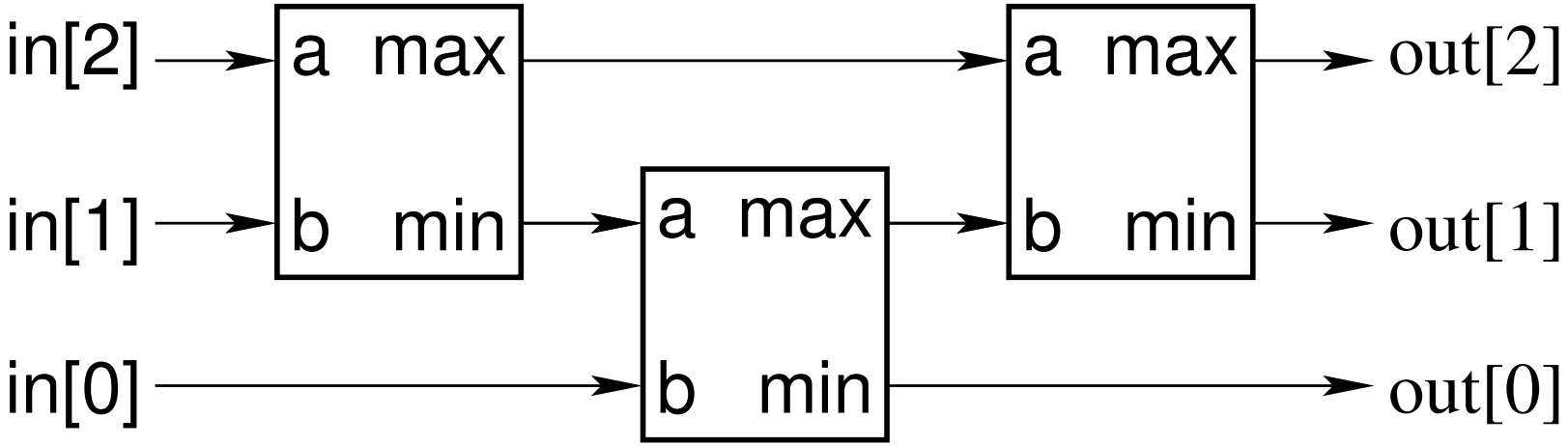
How would you write a parallel version of quicksort?

Sorting Networks

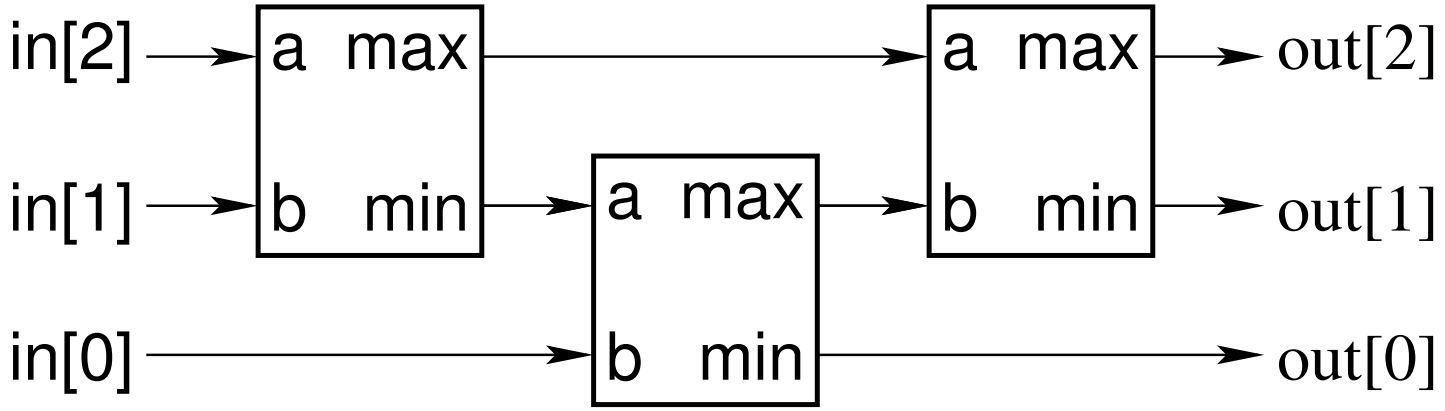
Sorting Network for 2–elements

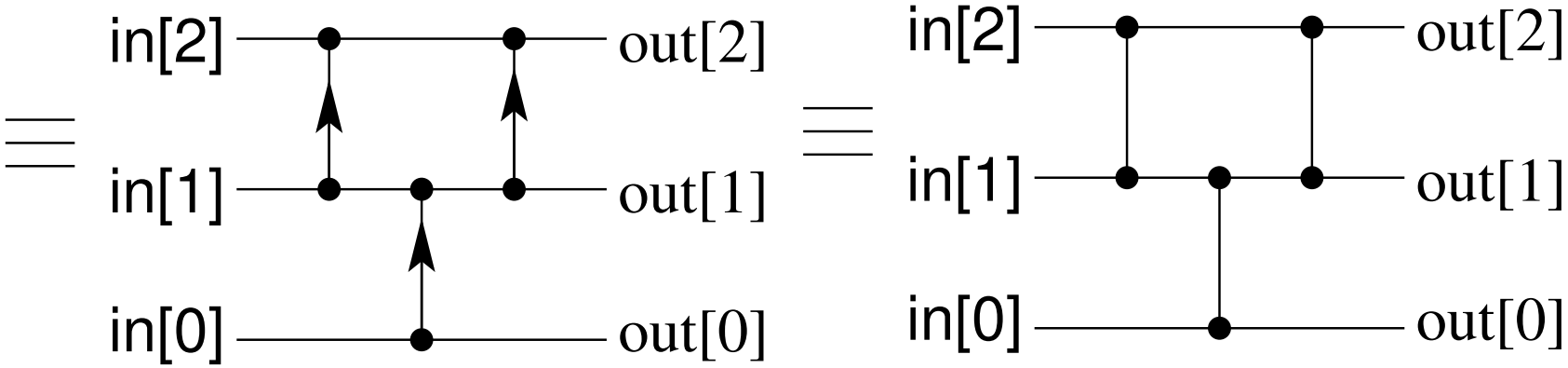


A Sorting Network for 3–elements

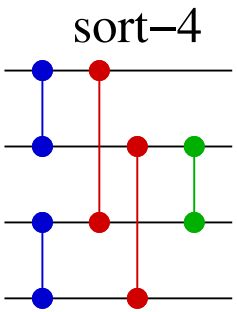


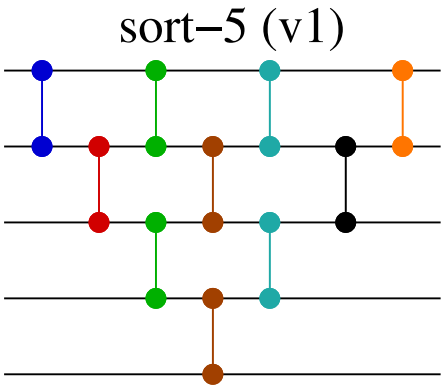
Sorting Networks – Drawing

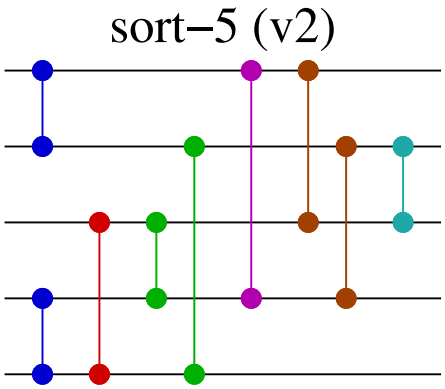


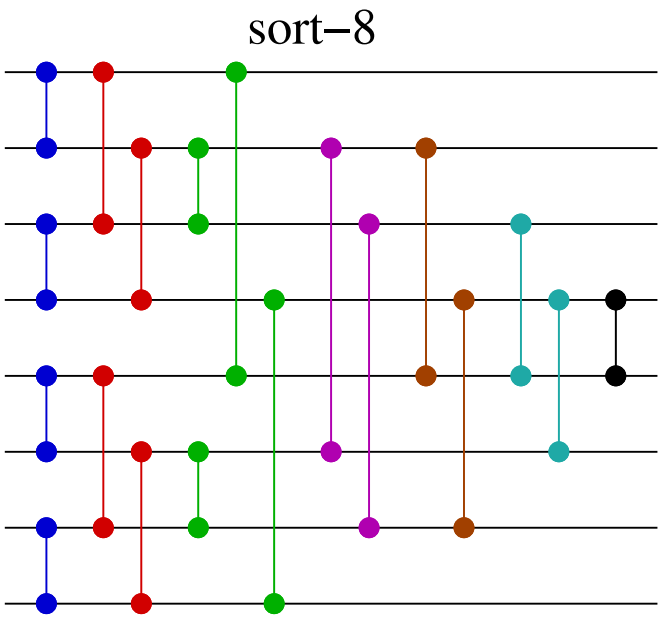


Sorting Networks – Examples









Operations of the same color can be performed in parallel.

See: http://pages.ripco.net/~jgamble/nw.html

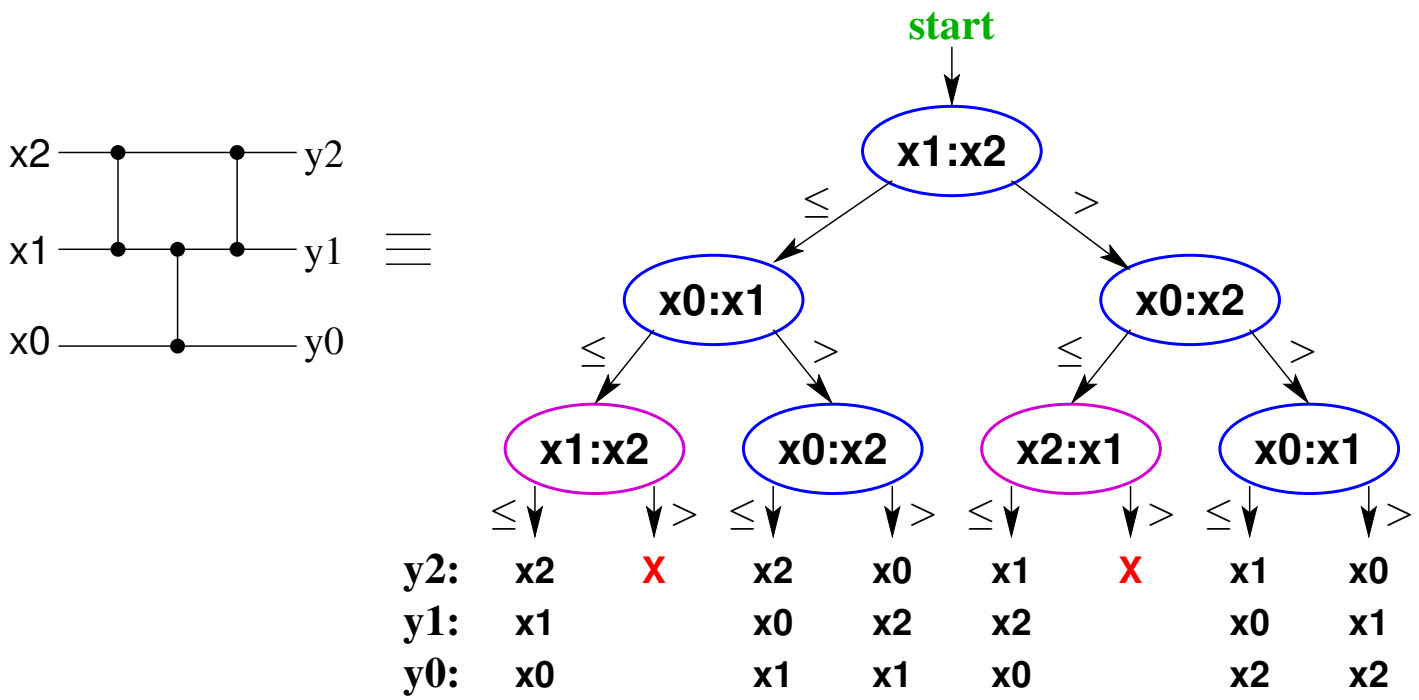
Sorting Networks: Definition

Structural version:

- A sorting network is an acyclic network consisting of compare-and-swap modules.
- I'll give an "inductive" version that makes this more rigorous on slide 10.

Sorting Networks: Definition

Decision-tree version:



- Let v be an arbitrary vertex of a decision tree, and let x_i and x_j be the variables compared at vertex v.
- A decision tree is a sorting network iff for every such vertex, the left subtree is the same as the right subtree with x_i and x_j exchanged.

The 0-1 Principle

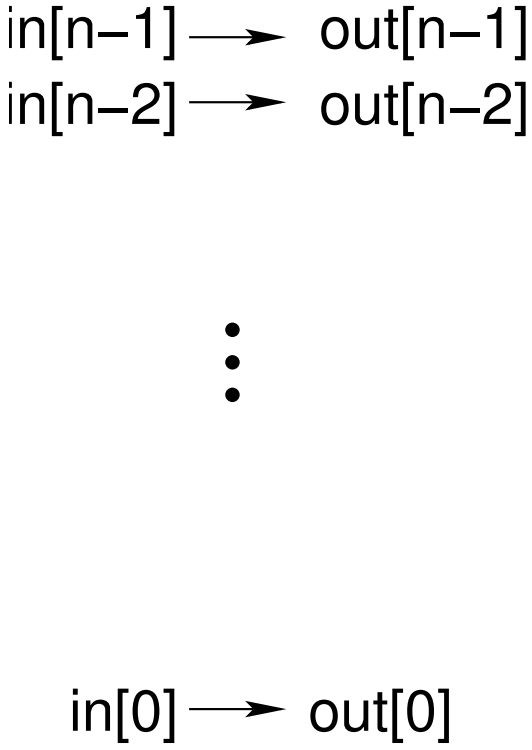
If a sorting network correctly sorts all inputs consisting only of 0s and 1s, then it correctly sorts inputs consisting of arbitrary (comparable) values.

- The 0-1 principle doesn't hold for arbitrary algorithms:
  - Consider the following linear-time "sort"
  - In linear time, count the number of zeros, *nz*, in the array.
  - Set the first *nz* elements of the array to zero.
  - Set the first remaining elements to one.
  - This correctly sorts any array consisting only of 0s and 1s, but does not correctly sort other arrays.
- By restricting our attention to sorting networks, we can use the 0-1 principle.

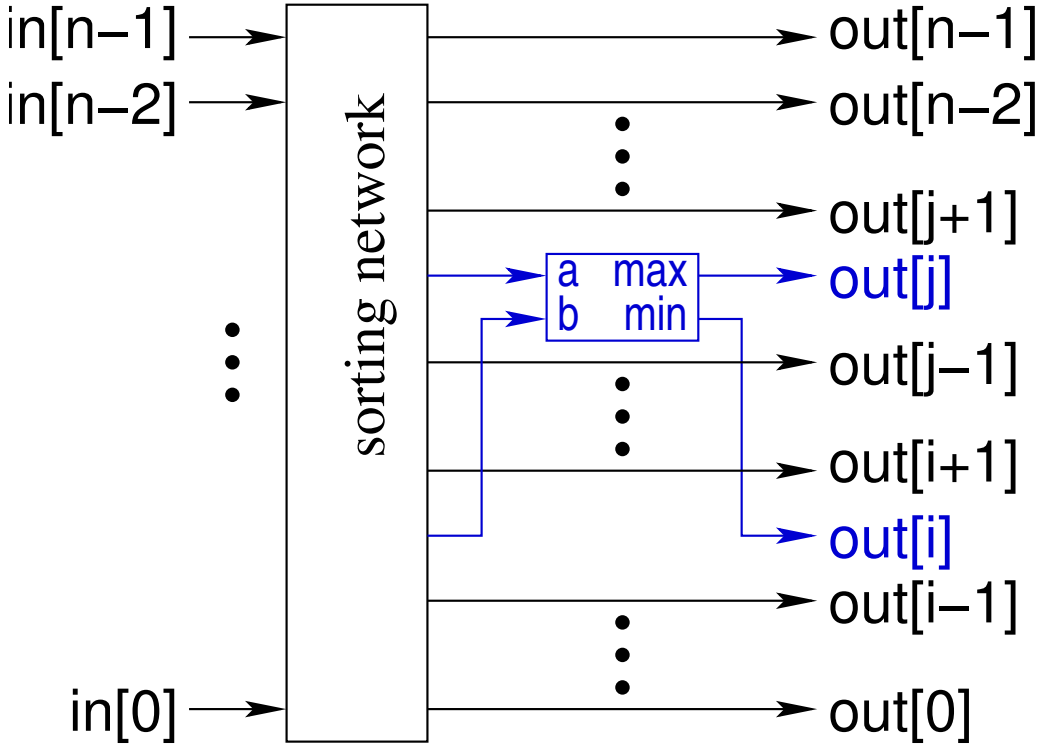
Sorting Networks: Inductive Definition

An N-input sorting network is either:

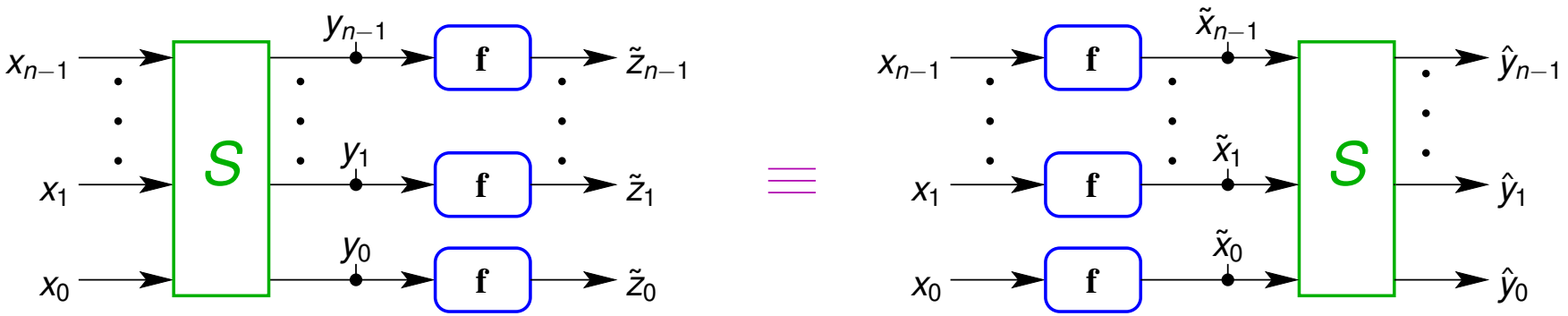
The identity function,



or a sorting network composed with a compare-and-swap element



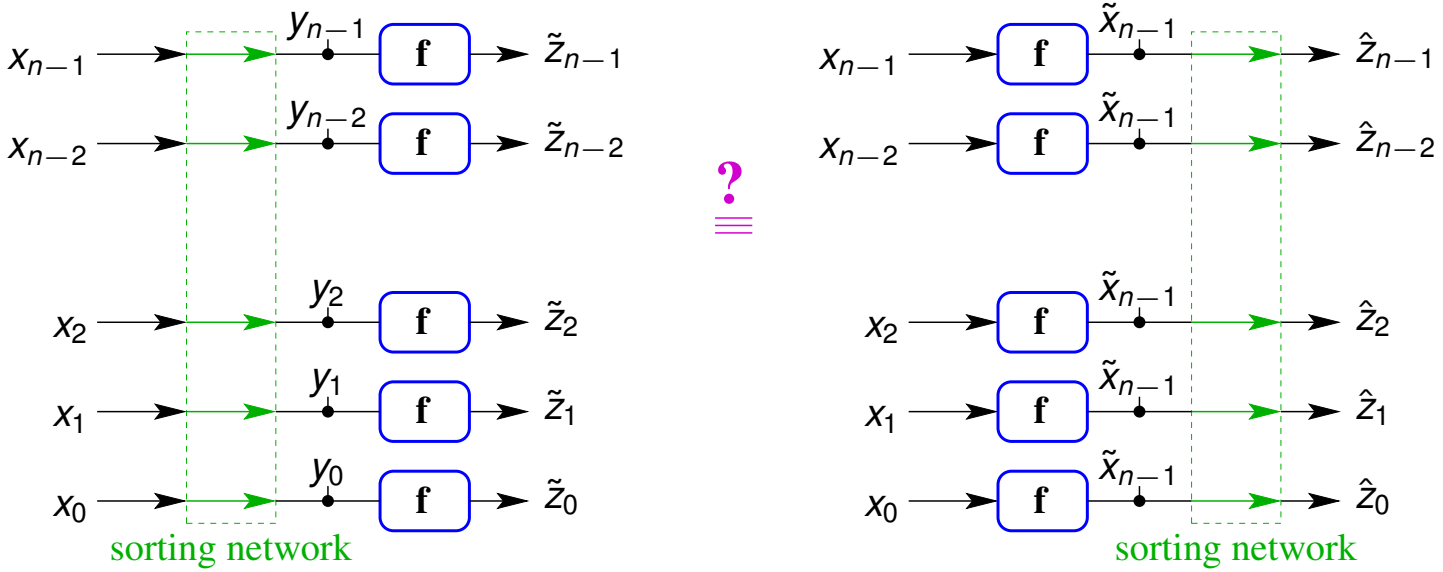
Monotonicity Lemma



Lemma: sorting networks commute with monotonic functions.

- Let S be a sorting network with n inputs an N outputs.
  - I'll write $x_0, \ldots, x_{n-1}$ to denote the inputs of S.
  - I'll write $y_0, \ldots, y_{n-1}$ to denote the outputs of S.
- Let f be a monotonic function.
  - If $x \leq y$, then $f(x) < f(y)$.
- The monotonicity lemma says
  - applying S and then f produces the same result as
  - applying f and then S.

Monotonicity Lemma – proof, Base Case

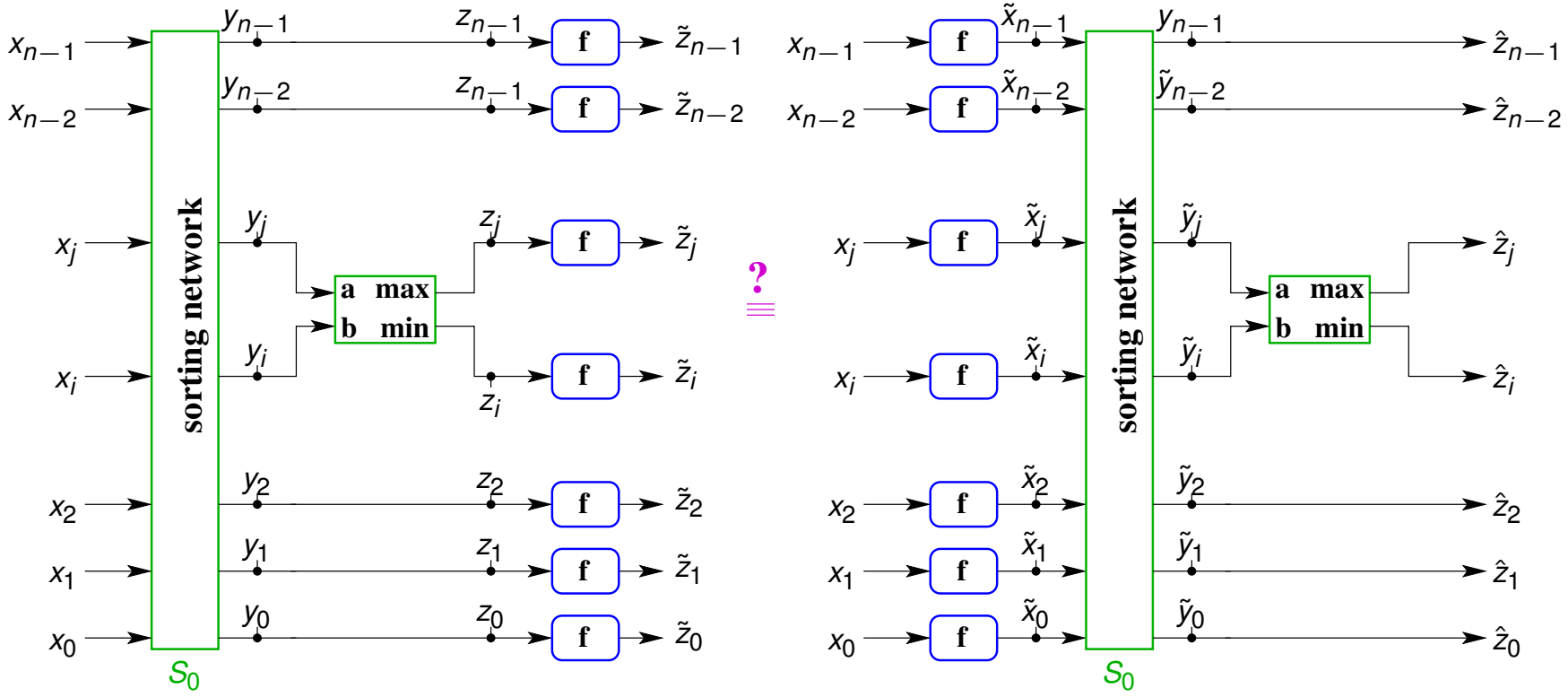


The sorting network, S, is the identity function.

$$f \bullet S \;=\; f \bullet \text{ident} \;=\; f \;=\; \text{ident} \bullet f \;=\; S \bullet f$$

Monotonicity Lemma – proof, Induction Step

Induction step: Let S_0 be a sorting network, and append a compare-and-swap to outputs i and j.



Monotonicity Lemma – proof, Induction Details

- All outputs other than i and j are unaffected by adding the new compare-and-swap module.
  - The claim follows directly from the induction hypothesis.
- For outputs i and j, we just note that the compare-and-swap operation commutes with f because f is monotonic.

```
y_i = min(x_i, x_j);
y_j = max(x_i, x_j);
z_i = f(y_i);
z_j = f(y_j);
```

$\equiv$

```
w_i = f(x_i);
w_j = f(x_j);
z_i = min(w_i, w_j);
z_j = max(w_i, w_j);
```

The 0-1 Principle

If a sorting network correctly sorts all inputs consisting only of 0s and 1s, then it correctly sorts inputs of any values.

I'll prove the contrapositive.

- If a sorting network does not correctly sort inputs of any values, then it does not correctly sort all inputs consisting only of 0s and 1s.
- Let S be a sorting network, let x be an input vector, and let $y = S(x)$, such that there exist i and j with $i < j$ such that $y_i > y_j$.
- Let
$$\begin{aligned} f(x) &= 0, && \text{if } x < y_i \\ &= 1, && \text{if } x \geq y_i \\ \tilde{y} &= S(f(x)) \end{aligned}$$
- By the definition of f, $f(x)$ is an input consisting only of 0s and 1s.
- By the monotonicity lemma, $\tilde{y} = f(y)$. Thus,
$$\tilde{y}_i = f(y_i) = 1 > 0 = f(y_j) = \tilde{y}_j$$
- Therefore, S does not correctly sort an input consisting only of 0s and 1s.
- $\square$

Summary

- Sequential sorting algorithms don't parallelize in an "obvious" way because they tend to have sequential bottlenecks.
  - Later, we'll see that we can combine ideas from sorting networks and sequential sorting algorithms to get practical, parallel sorting algorithms.
- Sorting networks are a restricted class of sorting algorithms
  - Based on compare-and-swap operations.
  - The parallelize well.
  - They don't have control-flow branches – this makes them attractive for architectures with large branch-penalties.
- The zero-one principle:
  - If a sorting-network sorts all inputs of 0s and 1s correctly, then it sorts all inputs correctly.
  - This allows many sorting networks to be proven correct by counting arguments.

Preview

April 1: Bitonic Sort – the algorithm
April 4: Bitonic Sort – the code
April 6: Map Reduce
Reading: MapReduce: Simplified Data Processing on Large Clusters
April 8: PReach: Parallel Model Checking
Reading: Industrial Strength Distributed Explicit State Model Checking

But of course, we'll adjust this as we go.

Review

- Why don't traditional, sequential sorting algorithms parallelize well?
- Try to parallelize another sequential sorting algorithm such as heap sort? What issues do you encounter?
- I claimed that `max` and `min` can be computed without branches. We could work out the hardware design for a compare-and-swap module. Instead, consider an algorithm that takes two "words" as arguments – each word is represented as a list of characters. The algorithm is supposed to output the two words, but in alphabetical order. For example:

```
compareAndSwap("cat", "dog") -> {"cat", "dog"};
compareAndSwap("dog", "cat") -> {"cat", "dog"};
compareAndSwap(X, Y) when X =< Y -> {X, Y};
compareAndSwap(X, Y) -> {Y, X}.
```

  Show that `compareAndSwap` can be implemented as a scan operation.