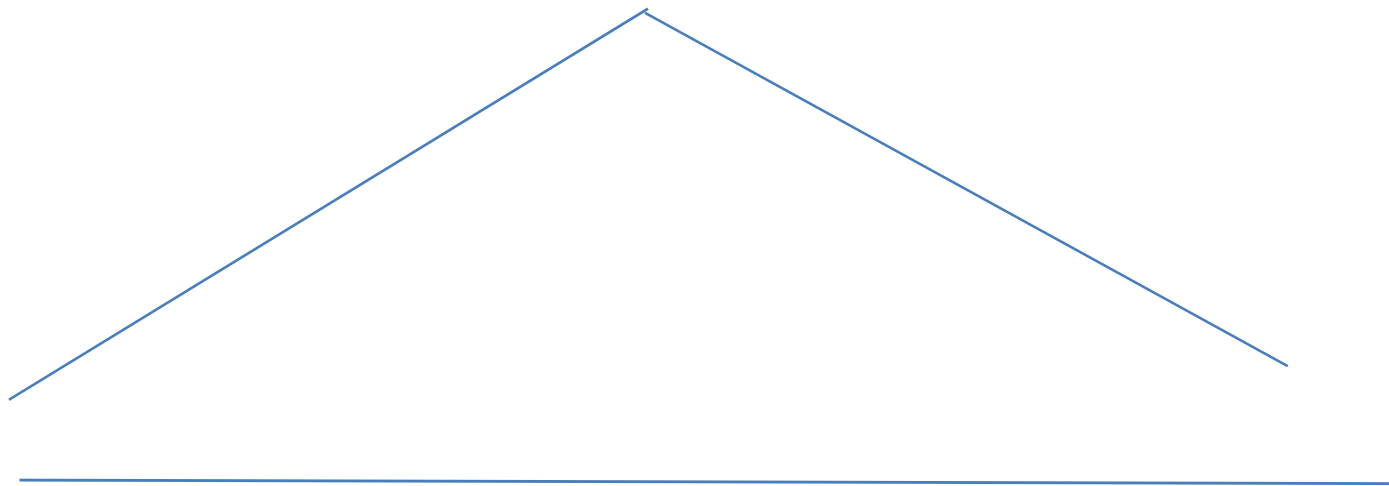


CSL 860: Modern Parallel Computation

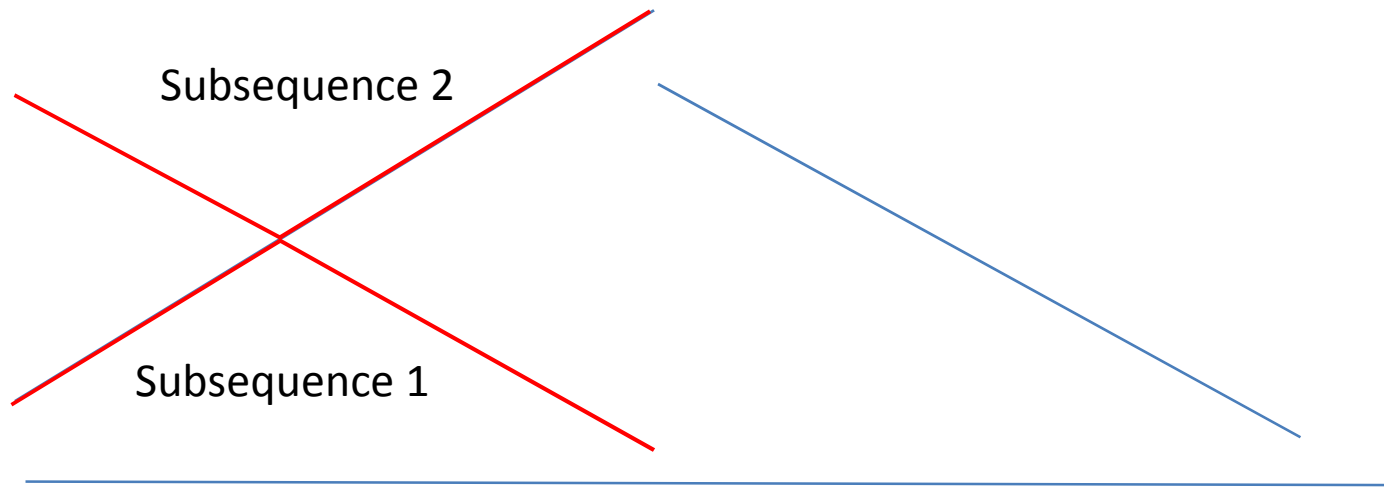
PARALLEL SORTING

Bitonic Merge and Sort

- Bitonic sequence: $\{a_0, a_1, \dots, a_{n-1}\}$:
 - A sequence with a monotonically increasing part and a monotonically decreasing part
 - For some i , $\{a_0 \leq \dots \leq a_i\}$ and $\{a_{i+1} \geq \dots \geq a_{n-1}\}$
 - Or, a cyclic-shift of indices makes it bitonic



Two Bitonic Sequences

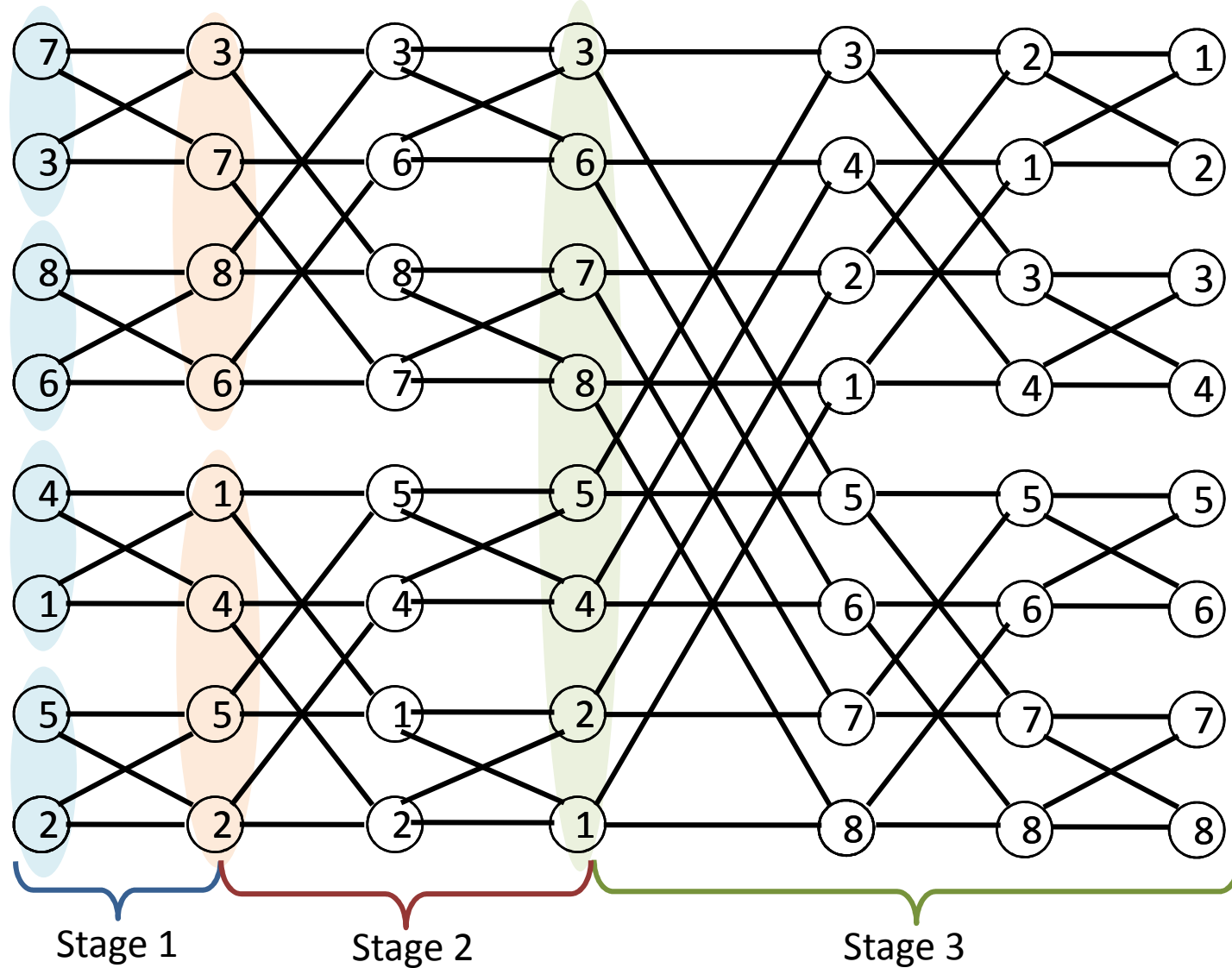


- Say $\{a_0 \leq \dots \leq a_{n/2} \geq a_{n/2+1} \geq \dots \geq a_{n-1}\}$
 - Subsequence 1: $\{\min(a_0, a_{n/2+1}), \min(a_1, a_{n/2+2}), \dots\}$
 - Subsequence 2: $\{\max(a_0, a_{n/2+1}), \max(a_1, a_{n/2+2}), \dots\}$
- Recursively sort each bitonic subsequence
 - Subsequence 1 \leq Subsequence 2

Bitonic Sort

- Sort each pairs
 - alternately in increasing and decreasing orders
- Every sequence of length four is now bitonic
- Sort recursively
 - Again alternate increasing and decreasing orders
 - Forming bitonic sequences of length eight now
 - And so on ..

Bitonic Network



log n stages
log²n time
n log²n work

Batcher's Odd-Even Merge

- Merge the even indices
 - Get $c_0, c_1, c_2, c_3, \dots$
- Merge the odd indices
 - Get $d_0, d_1, d_2, d_3, \dots$
- Perform an odd-even merge
 - $c_0, \min(c_1, d_0), \max(c_1, d_0), \min(c_2, d_1), \max(c_2, d_1), \dots, d_{n-1}$

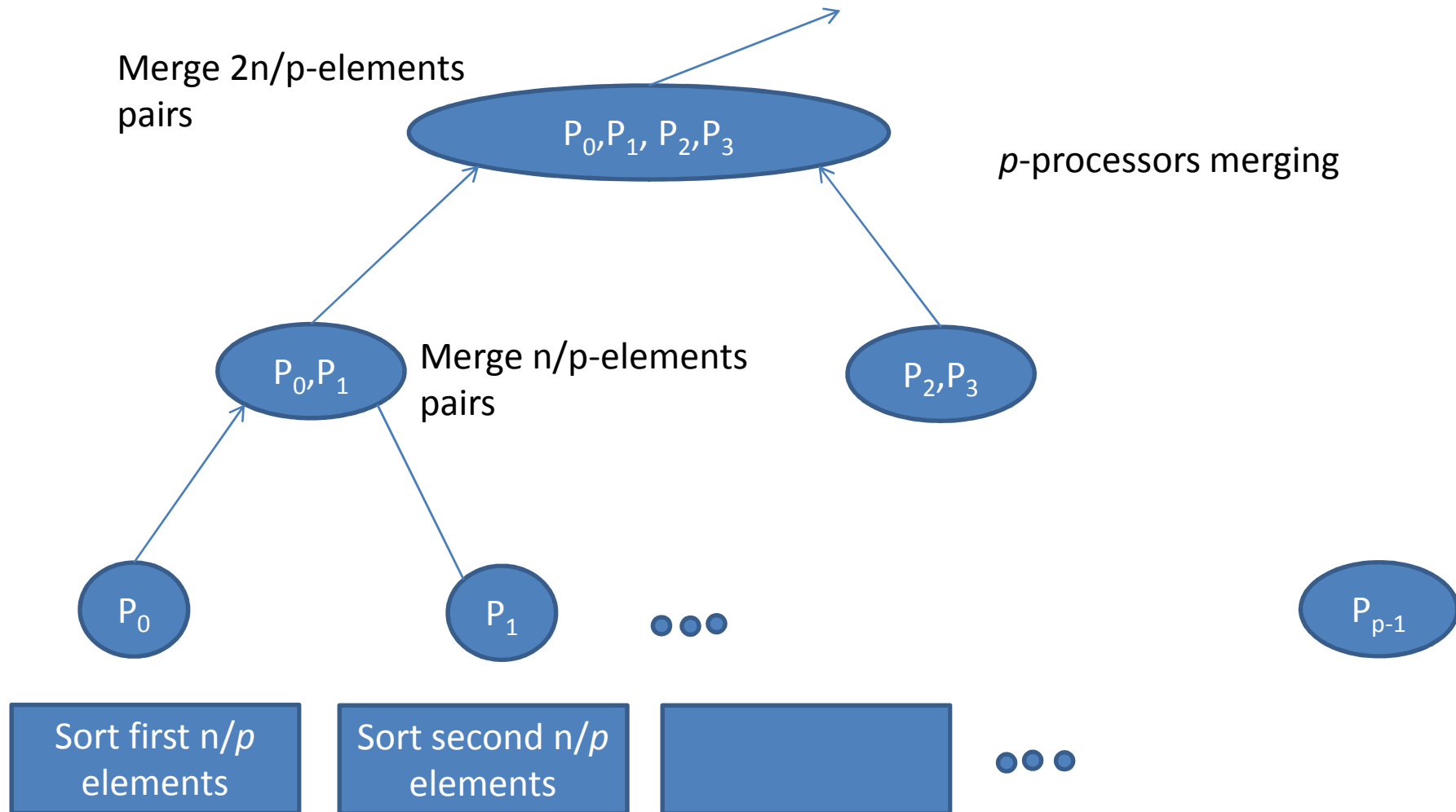
Example of a Fast Sort

- Rank sort
- Given Array A,
 - For each i , find rank $A[i]$
 - $A[\text{rank}[i]] = A[i]$
- How fast can you find $\text{Rank}(A:A)$?
 - If you had n^2 processors (Quiz)
 - If you had n^3 processors

Optimal Merge Sort

- Sort recursively similarly to a sequential
- Merge at each level of recursion using the optimal parallel merge
 - $O(\log \log n)$ time
 - $O(n)$ work
- Log n merge-tree levels =>
 - $O(\log n \log \log n)$ time
 - $O(n \log n)$ work

Sort n/p elements, then Merge



HOW EFFICIENTLY CAN YOU MERGE?

c-Cover Merging

- Consider sequence A and B
- X: a c-cover of A and B
 - Two consecutive elements of X have at most c elements of A (and at most c elements of B) between them
- Given Rank(X:A) and Rank(X:B)
 - Find Rank(A:B) and Rank(B:A)
 - In $O(1)$ time, with $O(n)$ work
- If X is a c-cover of B, and we know Rank(A:X) and Rank(X:B)
 - Compute Rank(A:B) in $O(1)$

Optimal $O(\log n)$ -time Merge Sort

- Works for any proper binary tree
 - Not necessarily balanced
- The sorted list $L[i]$ at a given node i is divided into stages
 - s^{th} stage generates list $L_s[i]$
- Initially:
 - $L_0[i] = \text{null}$ for internal nodes
 - $L_0[i] = \text{value}$ at leaf node i

Fast Optimal Merge Sort: Definitions

- Algorithm proceeds up the tree, one **stage** at a time
- At stage s , node n is **active** if
 - $\text{height}(\text{node}) \leq s \leq 3 * \text{height}(n)$
 - $\text{height}(n) = \text{height}(\text{Tree}) - \text{path-length from root to } n$
- At each stage a node is active,
 - It merges a sample of lists of its children

Algorithm

- parallel for $n = \text{active nodes}$:
 - $L_{s+1}[n] = \text{Merge}(\text{Sample}_s(\text{left}), \text{Sample}_s(\text{right}))$
- $\text{SUB}_i(\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_3 \dots) = (\mathcal{L}_i, \mathcal{L}_{2i}, \mathcal{L}_{3i} \dots)$
 - $\text{Sample}_s(n) =$
 - $\text{SUB}_4(L_s[n])$, if $s \leq 3 * \text{height}(n)$
 - $\text{SUB}_2(L_s[n])$, if $s = 3 * \text{height}(n) + 1$
 - $\text{SUB}_1(L_s[n]) = L_s[n]$, if $s \geq 3 * \text{height}(n) + 2$

Analysis

- Number of elements at a node doubles
 - $L_{s+1}[n] \leq 2 |L_s[v]| + 4$
- $\text{Sample}_s(n)$ is a 4-cover for $\text{Sample}_{s+1}(n)$
 - i.e., No more than 4 items of $L_{s+1}[n]$ between two consecutive items of $L_s[n]$
- For each stage $s > \text{height}(n)$,
 - $L_s[n]$ is a 4-cover for $\text{Sample}_s(\text{left})$ and $\text{Sample}_s(\text{right})$
- For $s \geq 2$, Merge. I.e., compute:
 - $\text{Rank}(\text{Sample}_s(\text{left}), \text{Sample}_s(\text{right}))$
 - $\text{Rank}(\text{Sample}_s(\text{right}), \text{Sample}_s(\text{left}))$
 - Use $\text{Rank}(L_s:\text{Sample}_s(\text{left}))$ (similarly right) computed by:
 - $\text{Rank}(L_s:\text{Sample}_{s-1}(\text{left}))$ and $\text{Rank}(\text{Sample}_{s-1}(\text{left}):\text{Sample}_s(\text{left}))$

Parallel Quick-Sort

- Group of p processors sort a sub-sequence
- Initially all processors sort the entire sequence
- The sequence is divided into n/p blocks
- Together processors choose a pivot
- Processors rearrange elements into two '*halves*'
 - Prefix sum
- The groups is subdivided into two and assigned to each 'half'
 - The size of each subgroup may be proportional to the size of the 'half'
 - If subgroup size = 1, stop subdividing and sort serially

Bucket Sort

- Decide buckets
- Parallel for element i :
 - Put it in bucket b
 - Keep incoherent cache of each bucket per processor
 - Merge buckets
 - Sort each bucket separately
- All buckets need not be equal
 - Load imbalance
- Sample sort:
 - Choose a sample of size s
 - Sort the samples
 - Choose $\mathbf{B}-1$ evenly spaced element from the sorted list
 - These elements (*splitters*) demarcate \mathbf{B} buckets

Parallel Splitter Selection

- Divide n elements into B (equi-sized) blocks
- (Quick)Sort each block
- For each sorted block:
 - Choose $B-1$ evenly spaced splitters
- Use the $B*(B-1)$ elements as samples
- Sort the samples
 - Choose $B-1$ Splitters
- No bucket contains more than $2*n/B$ elements

Radix Sort

- Bucket-sort by each key
 - Sort stably
- For each key, find its rank
 - Count number of keys ' $<$ ' in the sequence
 - Count number of keys ' $=$ ' before it in the sequence
 - Use parallel prefix sum
 - Notbit = !bit
 - psum = escan(bit)
 - nZeros = psum[n] + notbit[n]
 - nBefore = idx - psum + nZeros
 - Rank = bit? nBefore : psum

Blocked Radix-sort

- Consider b bits at a time as “key”
 - Prefix-sum still required per-bit?
- Satish/Harris/Garland:
 - Divide sequence into blocks, Divide key into nibbles
 - Load each block into shared memory
 - Sort by 4 iterations of single-bit stable sorts (How?)
 - For each block, write its 16-entry digit histogram and the sorted block to global memory.
 - Perform a prefix sum over the 16B histogram tables
 - Copy elements of each block to their correct output position