Parallel Algorithm Design

CS 263

Michael T. Goodrich

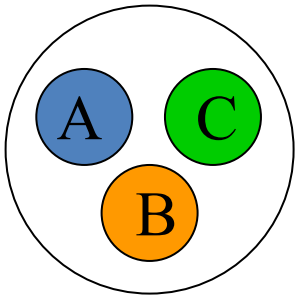
University of California, Irvine

Some slides adapted from content by Joseph Jaja, Wikipedia, Guy Blelloch

Fundamental Algorithms

- **Prefix sums**: O(log n) time, linear work, EREW PRAM, binary fork-join
- **List ranking**: O(log n) time, linear work, EREW PRAM, binary fork-join (randomized)
- **Sorting**: O(log n) time, O(n log n) work, EREW PRAM, binary fork-join
- **Maximum/minimum finding**: O(log log n) time, linear work, CRCW PRAM

Binary tree



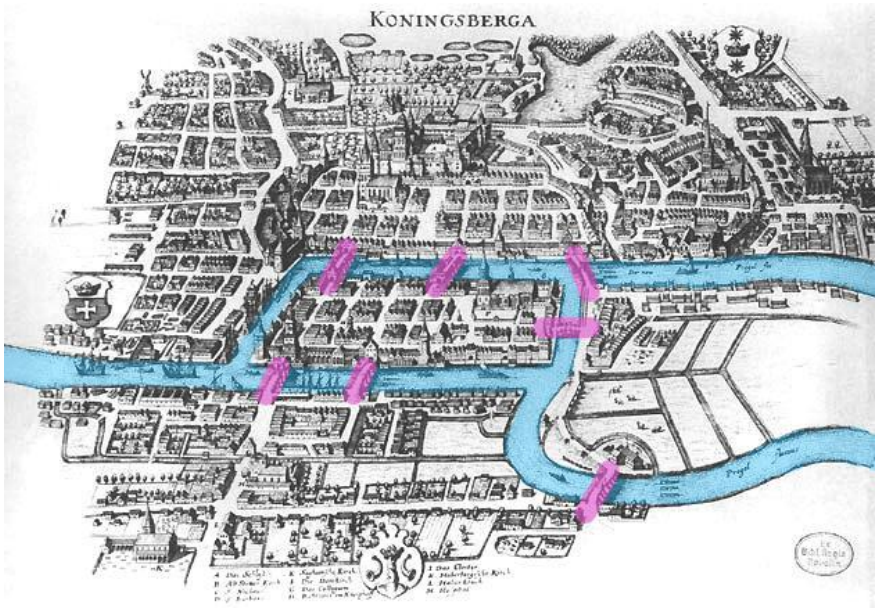
- Let T be a binary tree stored in a PRAM
- Each node *i* has fields parent[i], left[i] and right[i], which point to node i's parent, left child and right child respectively
- Let's assume that each node is identified by a non-negative integer
- Also we associate not one but 3 processes with each node; we call these node's A,B and C processors

Computing depth of each node in an n node tree takes O(n) time on a serial RAM

- A simple parallel algorithm to compute depths propagates a “wave” downward from the root of the tree.
  - The wave reaches all nodes at the same depth simultaneously, and thus by incrementing a counter carried along with the wave, we can compute the depth of each node.
- This parallel algorithm works well on a complete binary tree, since it runs in time proportional to the tree’s height.
- But the height of the tree could be as large as n-1

The Euler-tour technique

- An Euler tour of a graph is a path that traverses each edge exactly once
- A connected, directed graph has an Euler tour if and only if for all vertices v, the in-degree of v equals the out degree of v
- A connected, undirected graph has an Euler tour if and only if all vertices, except possibly the start and finish, have even degree.

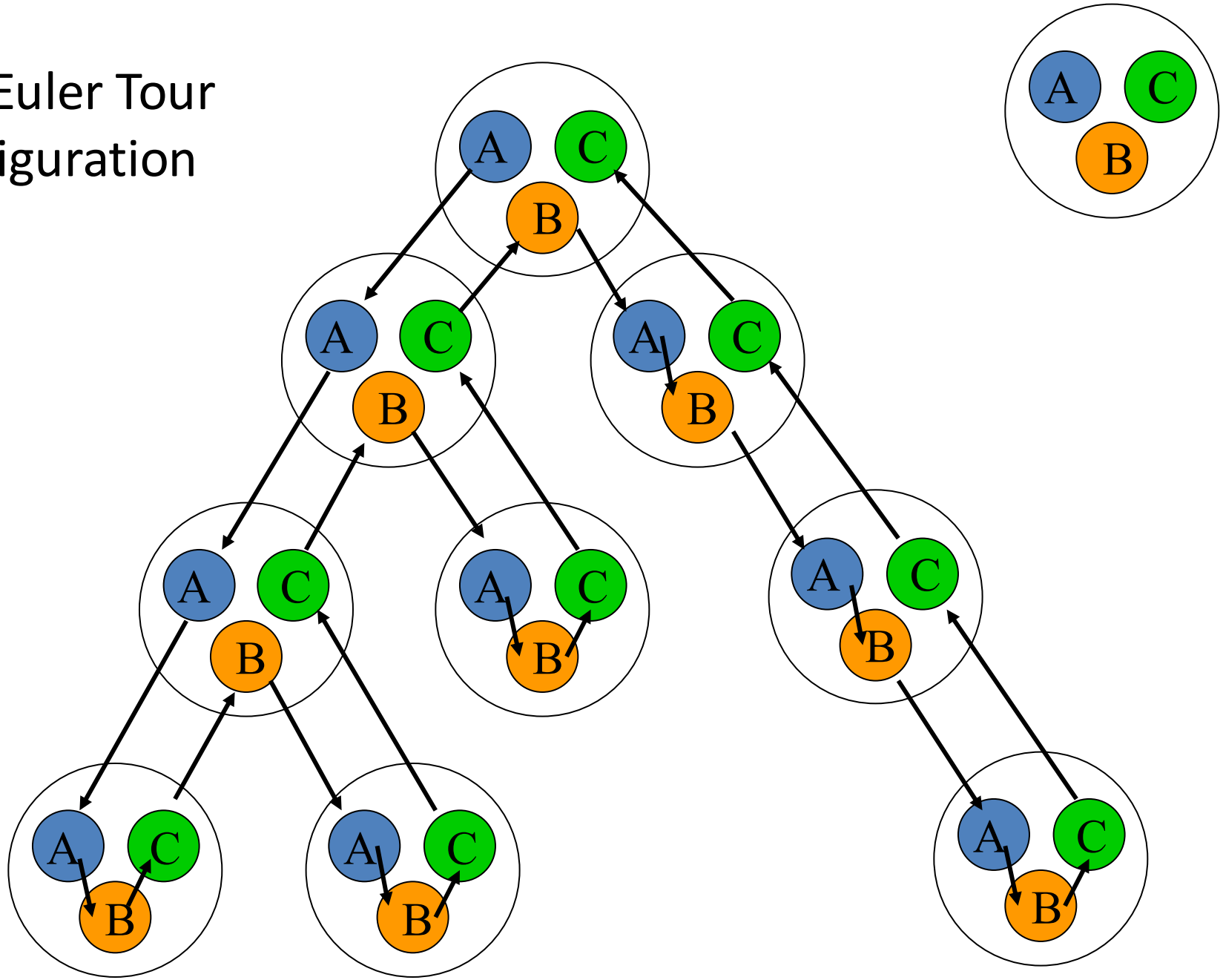


Leonhard Euler

Depth of nodes computation

- First we form an Euler tour of the directed version of T.
- The tour corresponds to walk of the tree with the following structure:
  - A node's A processor points to the A processor of its left child, if it exist, and otherwise to its own B processor
  - A node's B processor points to the A processor of its right child, if it exist, and otherwise to its own C processor
  - A node's C processor points to the B processor of its parent, if it is a left child and to the C processor of its parent if it is a right child. The root's C processor points to NIL.

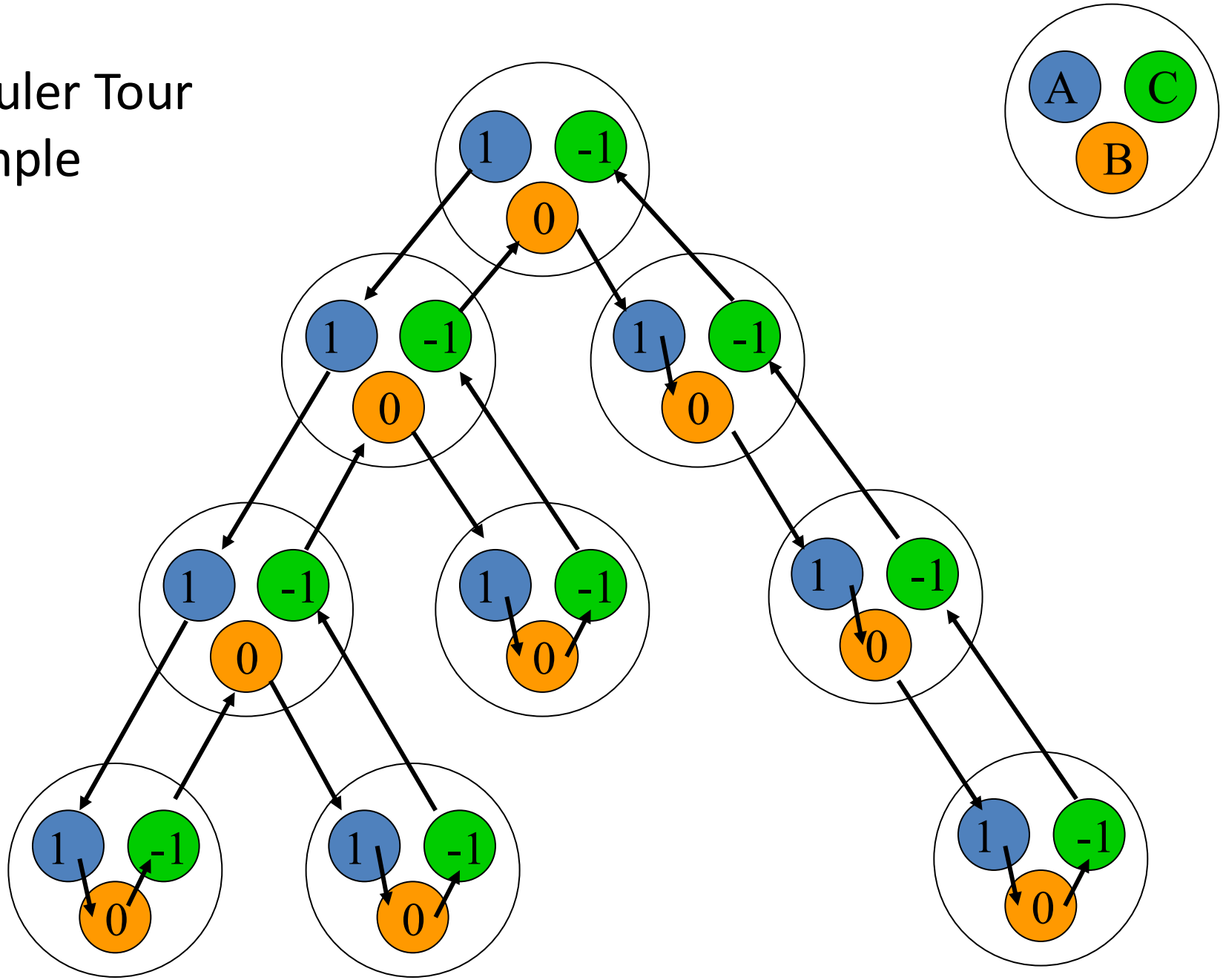
The Euler Tour Configuration



First step

- The head of the linked list formed by the Euler tour is the root's A processor, and the tail is the root's C processor.
- Given the pointers composing the original tree, an Euler tour can be constructed in O(1) time.
- Once we have linked list representing the Euler tour of T, we place
  - a 1 in each A processor,
  - a 0 in each B processor and
  - a –1 in each C processor

An Euler Tour Example



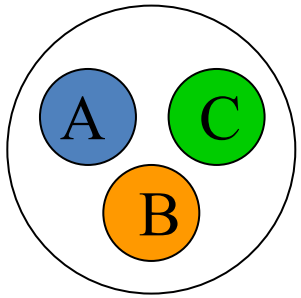
Second step

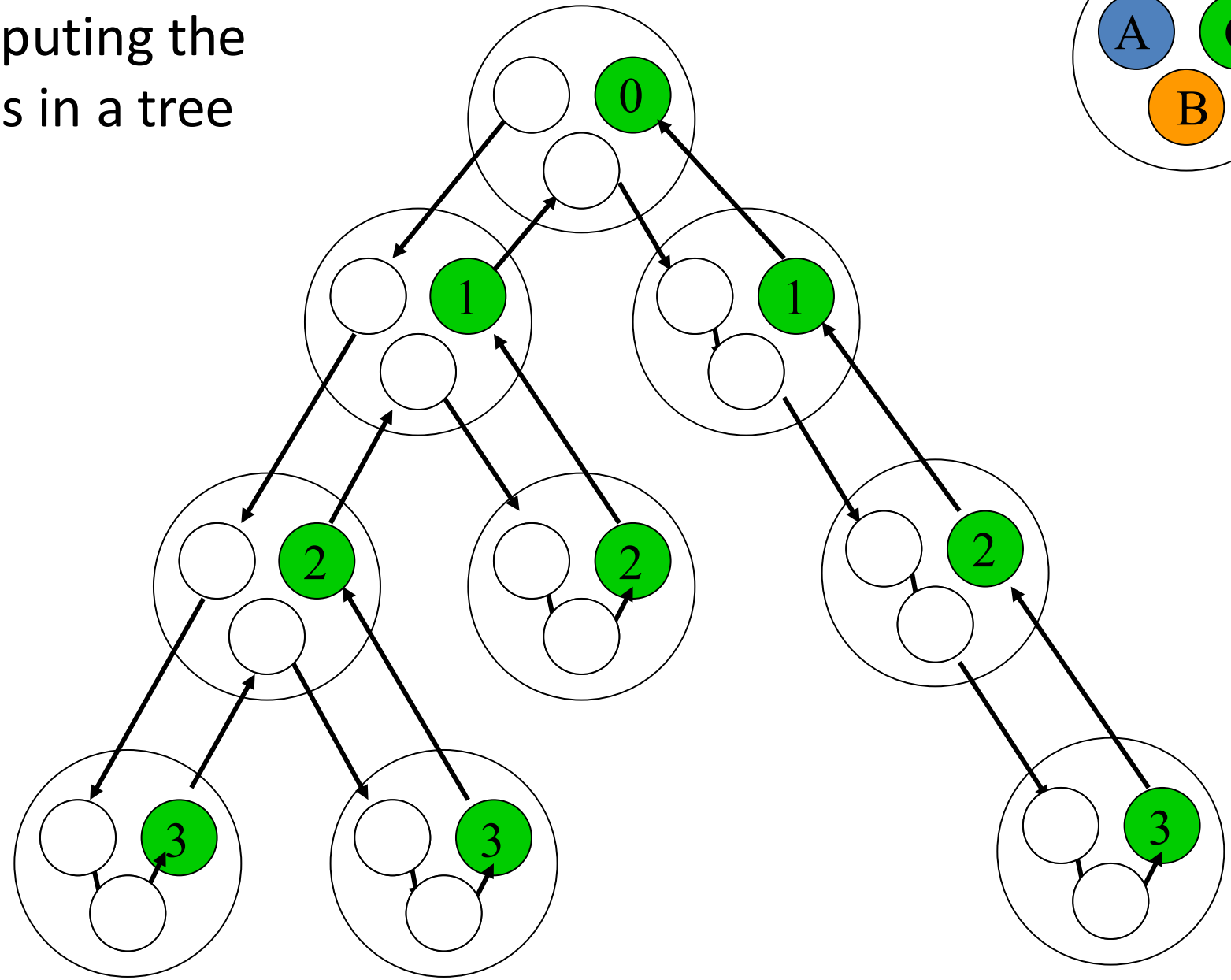
- We then perform list ranking and a parallel prefix computation using ordinary addition as the associative operation.

- We claim that after performing the parallel prefix computation, the depth of each node resides in the node's C processor.

- Why?

Computing the levels in a tree





WHY?

- The numbers are placed into the A,B and C processors in such a way that the net effect of visiting a subtree is to add 0 to the running sum
- The A processor of each node *i* contributes 1 to running sum
- The B processor of each node *i* contributes 0 because the depth of the node *i's* left child equals the depth of the node *i's* right child
- The C processor contributes –1, so the entire visit to the subtree rooted at node *i* has no effect on the running sum.

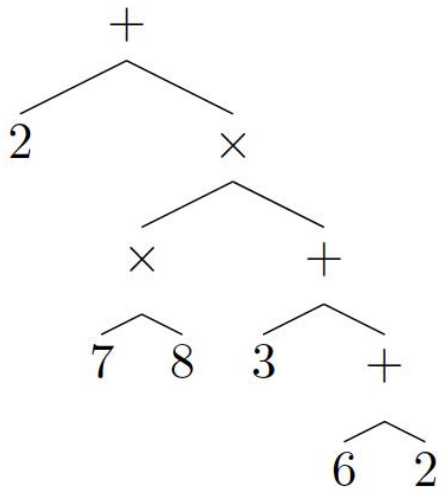
Euler-tour Technique Analysis

- The list representing Euler-tour can be computed in O(1) time.
- It has 3n objects, and thus the parallel prefix computation takes only O(log n) time
- Thus the total amount of time to compute all node depths is O(log n).
- Work is O(n).
- Because no concurrent memory accesses are needed, the algorithm is an EREW algorithm.
- Can be used for lots of tree computations: pre-order, in-order, post-order, number-of-descendants, etc.

Arithmetic Expressions

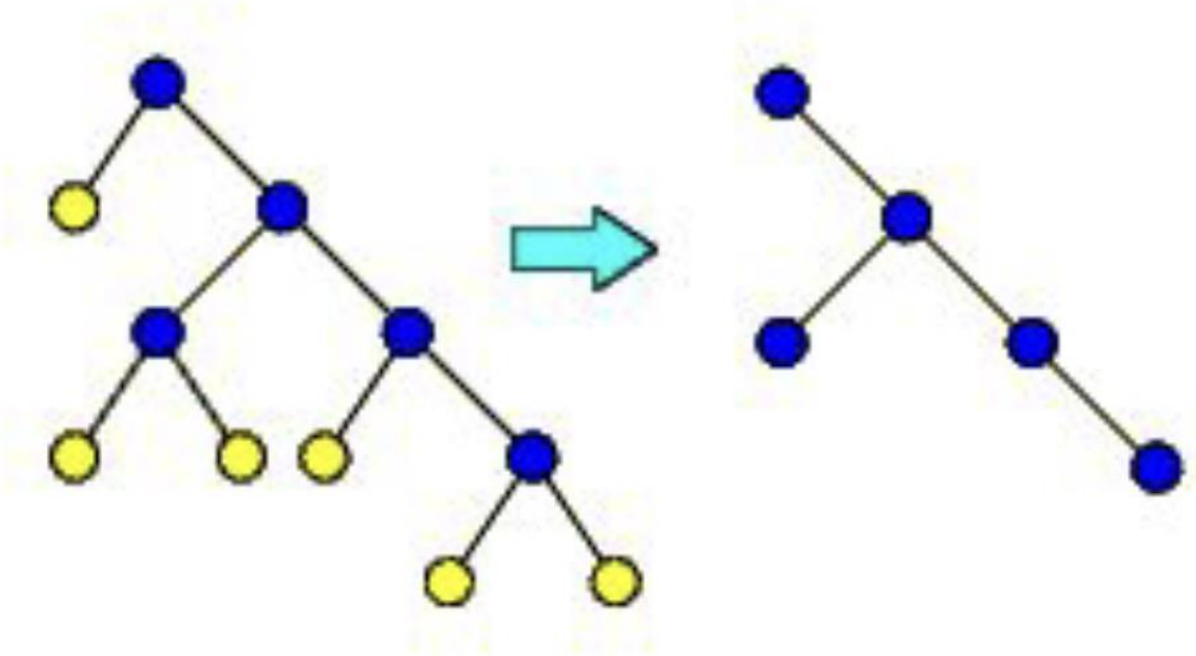
- Given a binary tree representing an arithmetic expression, involving addition and multiplication, compute the value associated with the root.

$2 + ((7 \times 8) \cdot (3 + (6 + 2)))$ corresponds to the following tree:



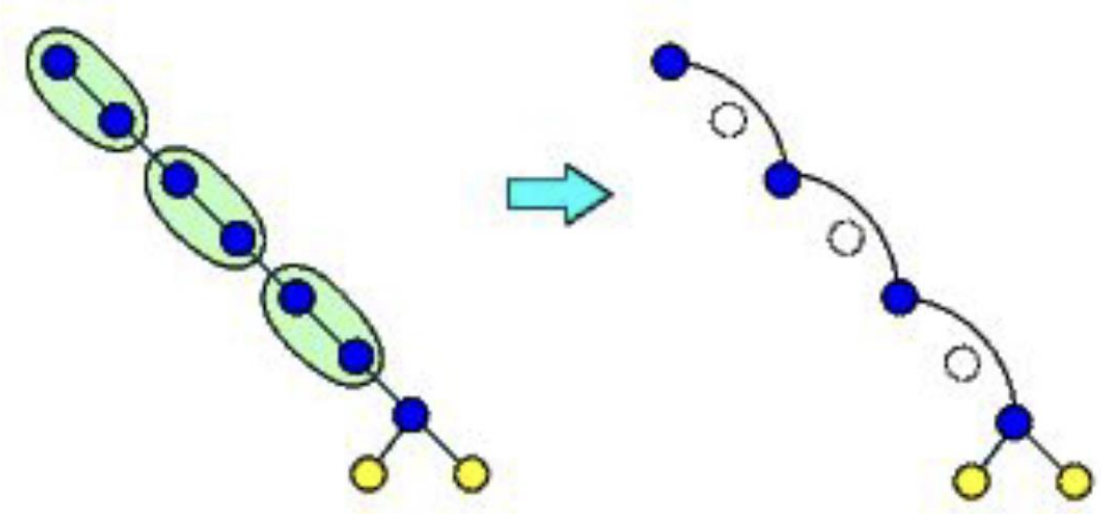
Rake

- The rake operation removes all the leaves in a tree.



Compress

- The compress operation compresses odd-depth nodes with their parents in chains of nodes with only one child.

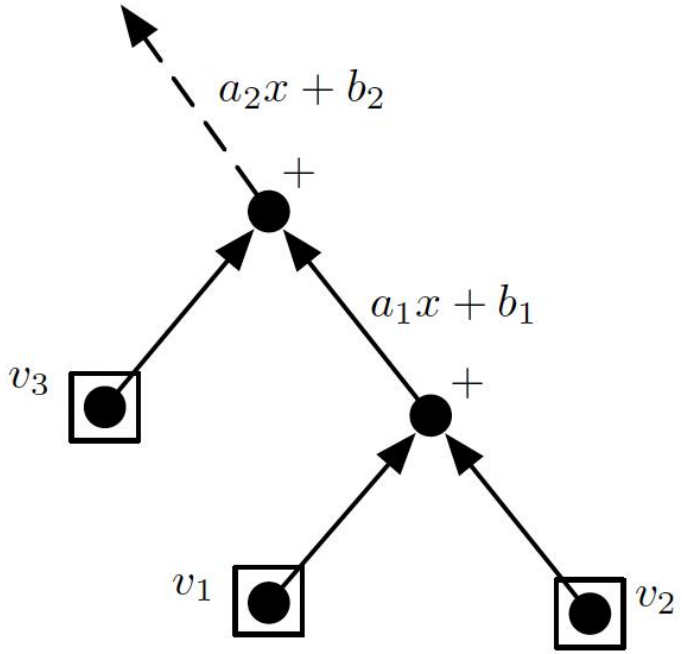


Rake-and-Compress

- Alternate rake and compress operations until there is only one node left.
- There are O(log n) steps and linear work:
- Every step removes the leaves and halves the depth of chains of nodes with only one child.

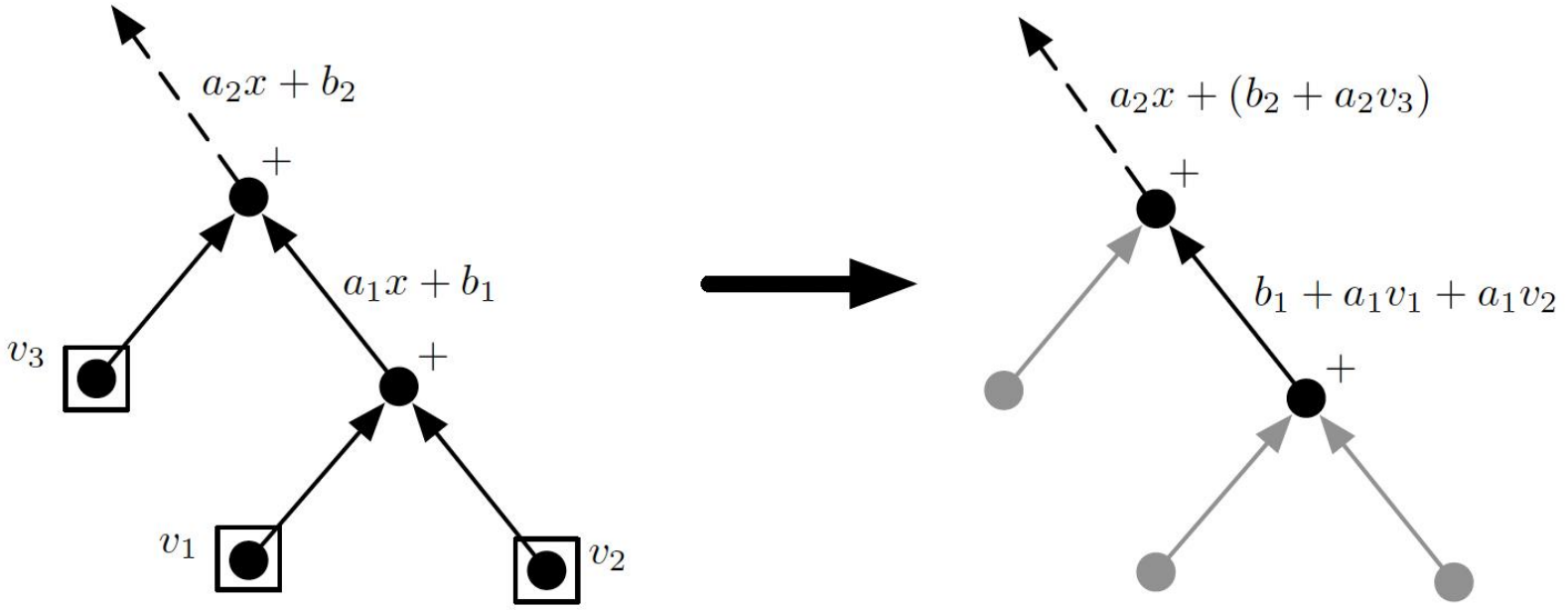
Use for Arithmetic Tree Operations

- Represent the value of each edge as a linear expression, ax+b, which is initially just 1x+0 for each edge.



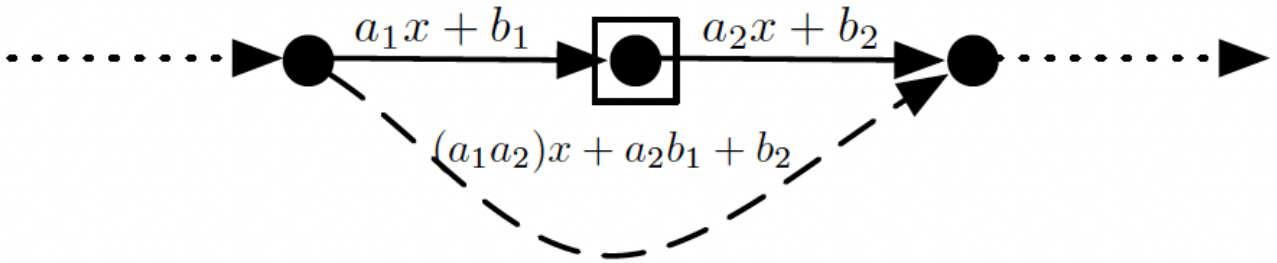
Update for Rake

- Update the edge for the parent for each removed leaf.



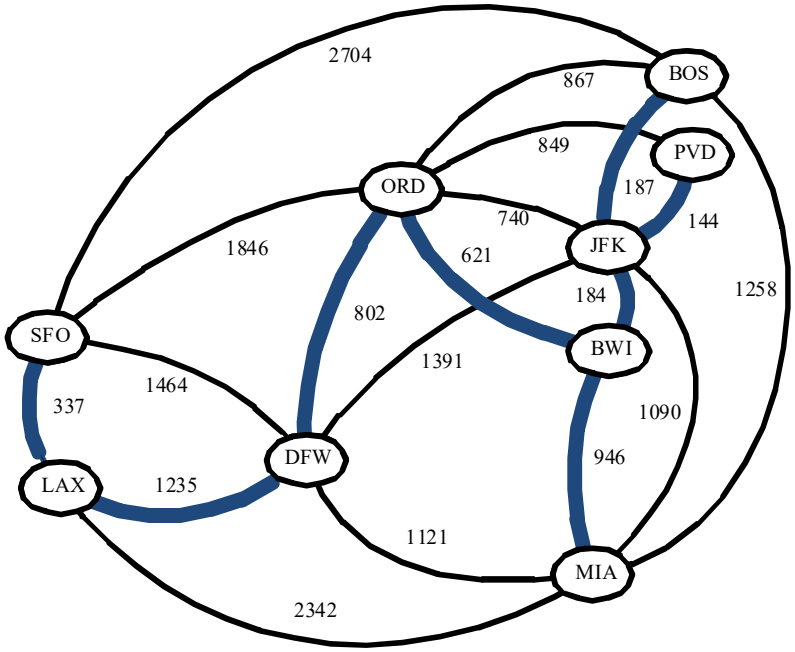
Update for Compress

- Update the edge for the parent based on plugging in the child equation.



- After O(log n) time and linear work in the EREW PRAM model we will have the root value, no matter how much the arithmetic tree is unbalanced.

Minimum Spanning Trees



Minimum Spanning Tree

Spanning subgraph

- Subgraph of a graph G containing all the vertices of G

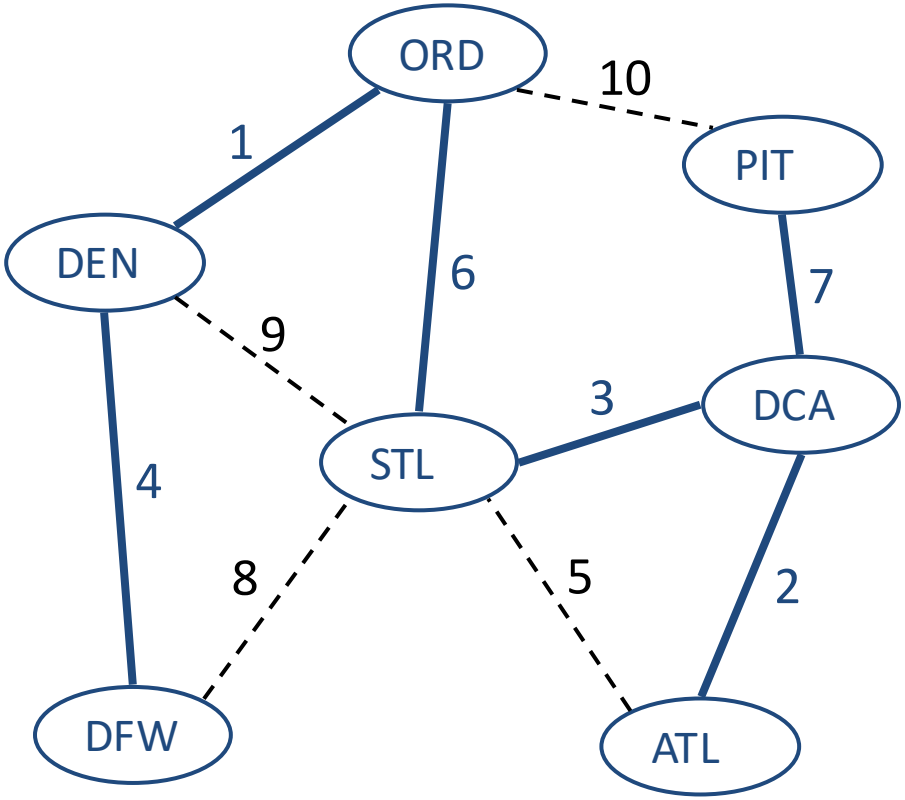
Spanning tree

- Spanning subgraph that is itself a (free) tree

Minimum spanning tree (MST)

- Spanning tree of a weighted graph with minimum total edge weight

- Applications
  - Communications networks
  - Transportation networks



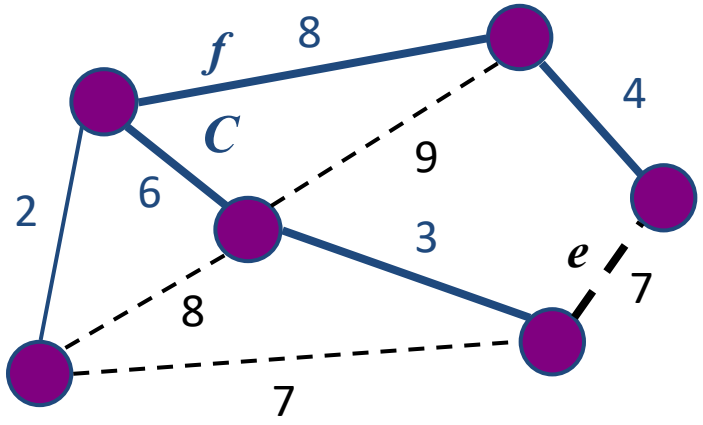
Cycle Property

Cycle Property:

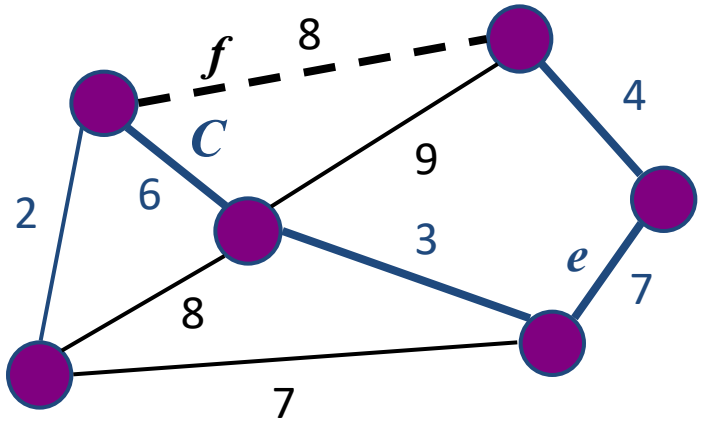
- Let T be a minimum spanning tree of a weighted graph G
- Let e be an edge of G that is not in T and C let be the cycle formed by e with T
- For every edge f of C, $weight(f) \leq weight(e)$

Proof:

- By contradiction
- If $weight(f) > weight(e)$ we can get a spanning tree of smaller weight by replacing e with f



Replacing f with e yields a better spanning tree



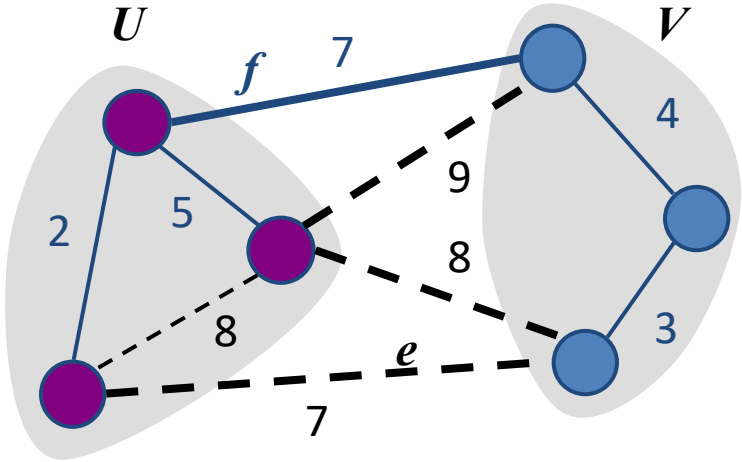
Partition Property

Partition Property:

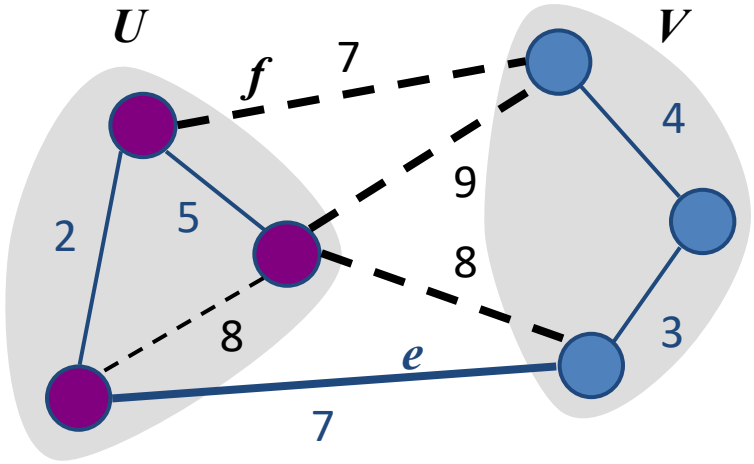
- Consider a partition of the vertices of G into subsets U and V
- Let e be an edge of minimum weight across the partition
- There is a minimum spanning tree of G containing edge e

Proof:

- Let T be an MST of G
- If T does not contain e, consider the cycle C formed by e with T and let f be an edge of C across the partition
- By the cycle property,
  $$weight(f) \leq weight(e)$$
- Thus, $weight(f) = weight(e)$
- We obtain another MST by replacing f with e



Replacing f with e yields another MST



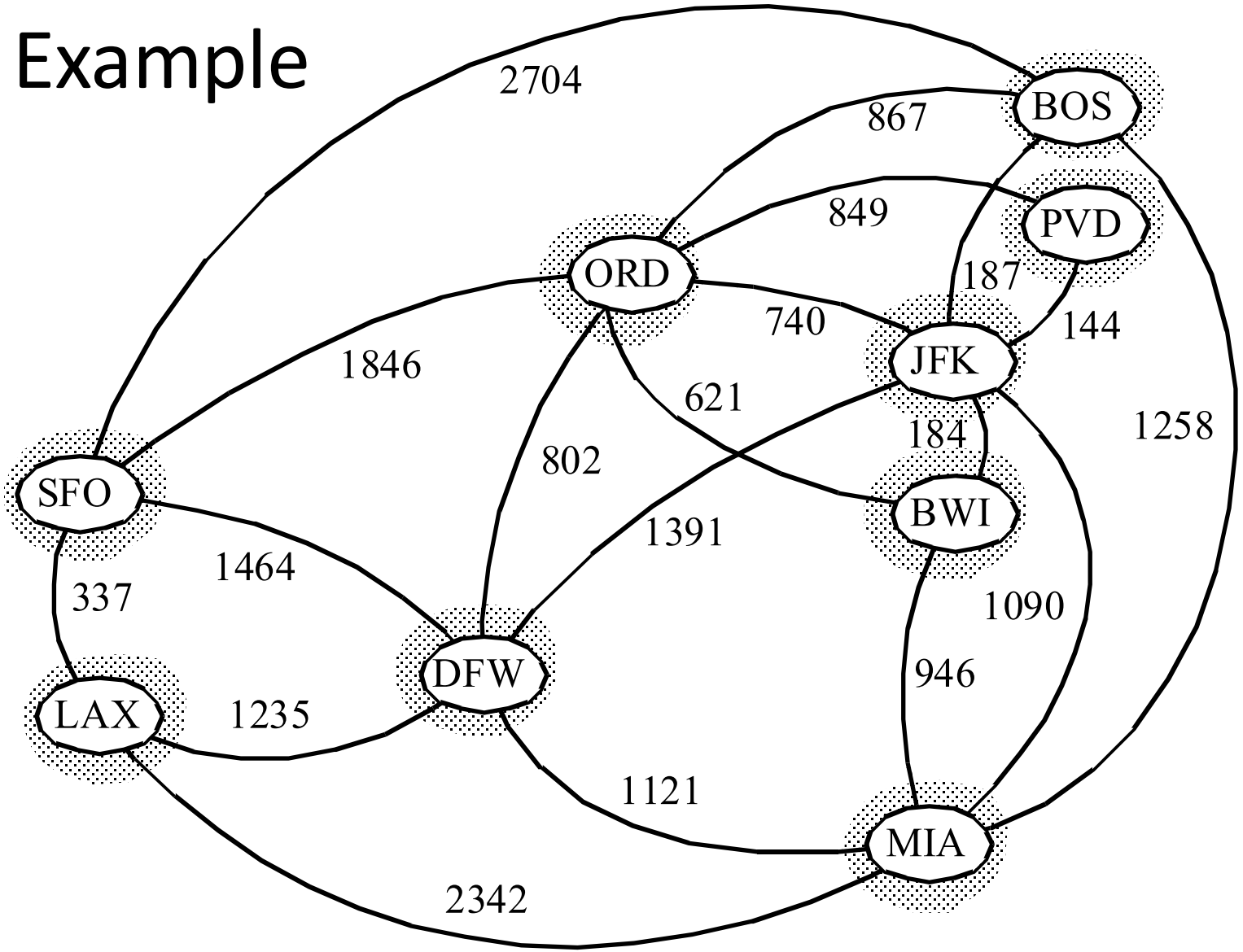
Baruvka's Algorithm

- Baruvka's algorithm grows many "clouds" of vertices at once.

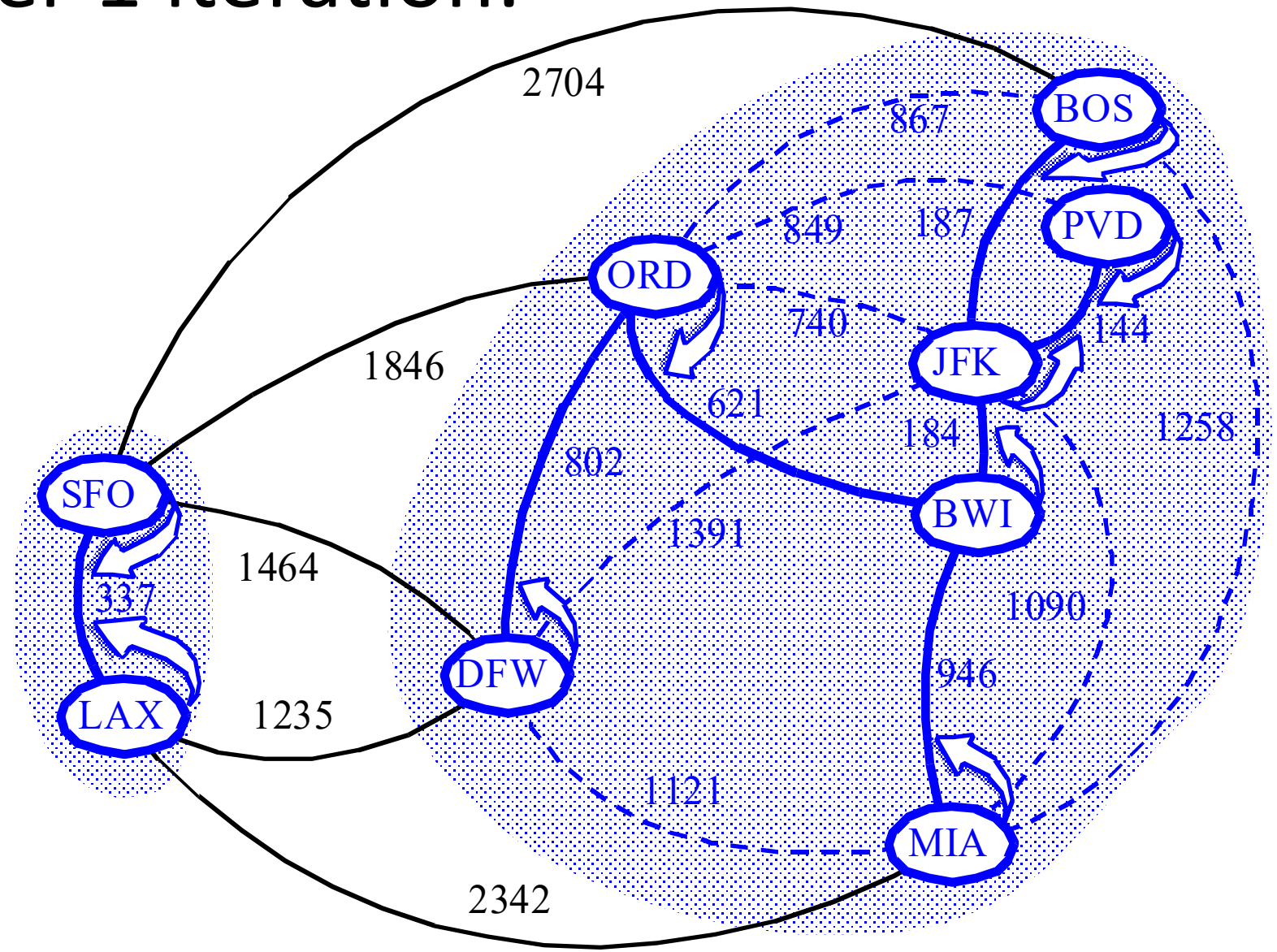
```
Algorithm BaruvkaMST(G)
    T ← V  {just the vertices of G}
  while T has fewer than n-1 edges do
    for each connected component C in T do
      Let edge e be the smallest-weight edge from C to another component in T.
      if e is not already in T then
        Add edge e to T
  return T
```

- Each iteration of the while-loop halves the number of connected compontents in T.
  - The sequential running time is O(m log n).
  - Implement this in parallel: O(log n) time O(m) work for each iteration of the while-loop
  - $O(\log^2 n)$ time and O(m log n) work.

Baruvka Example



After 1 iteration:



After 2 iterations: