Reduce & Scan Examples

Mark Greenstreet

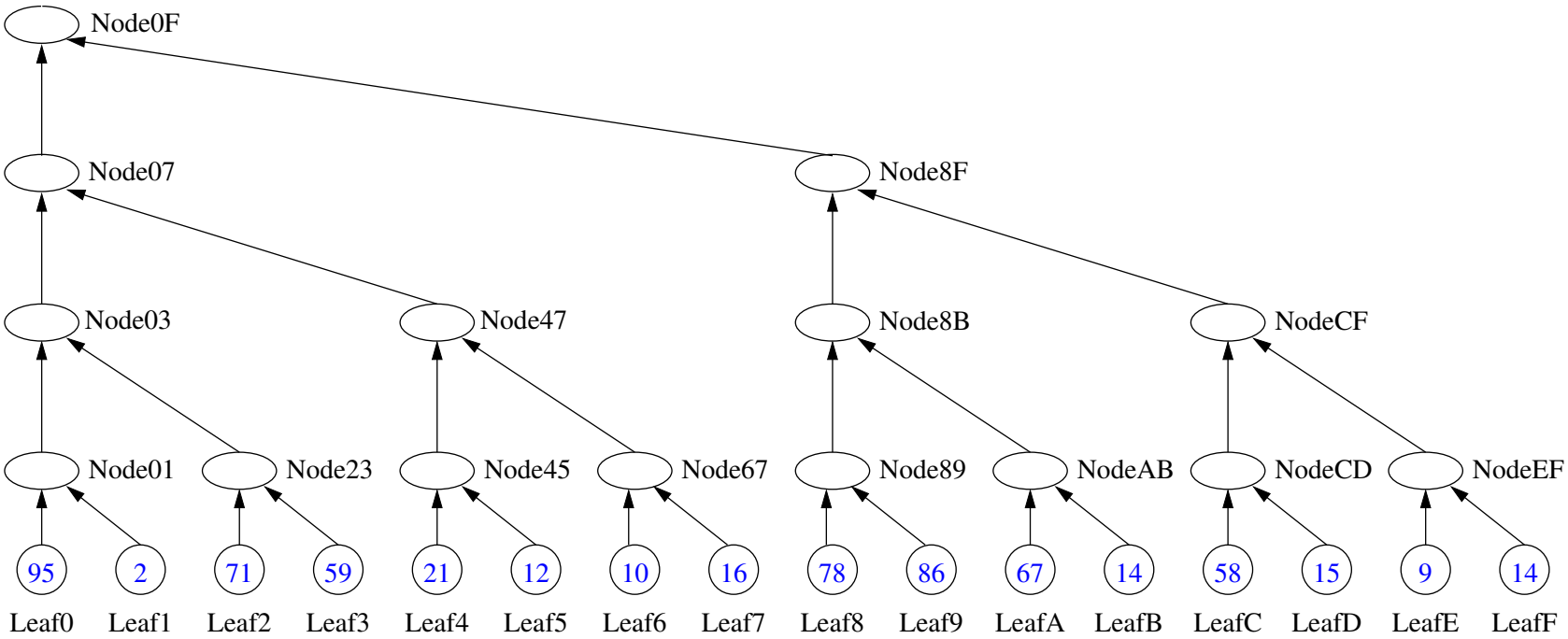
CpSc 418 – September 19, 2018

- Finishing Scan
- Patterns for Reduce and Scan

Scan in two passes

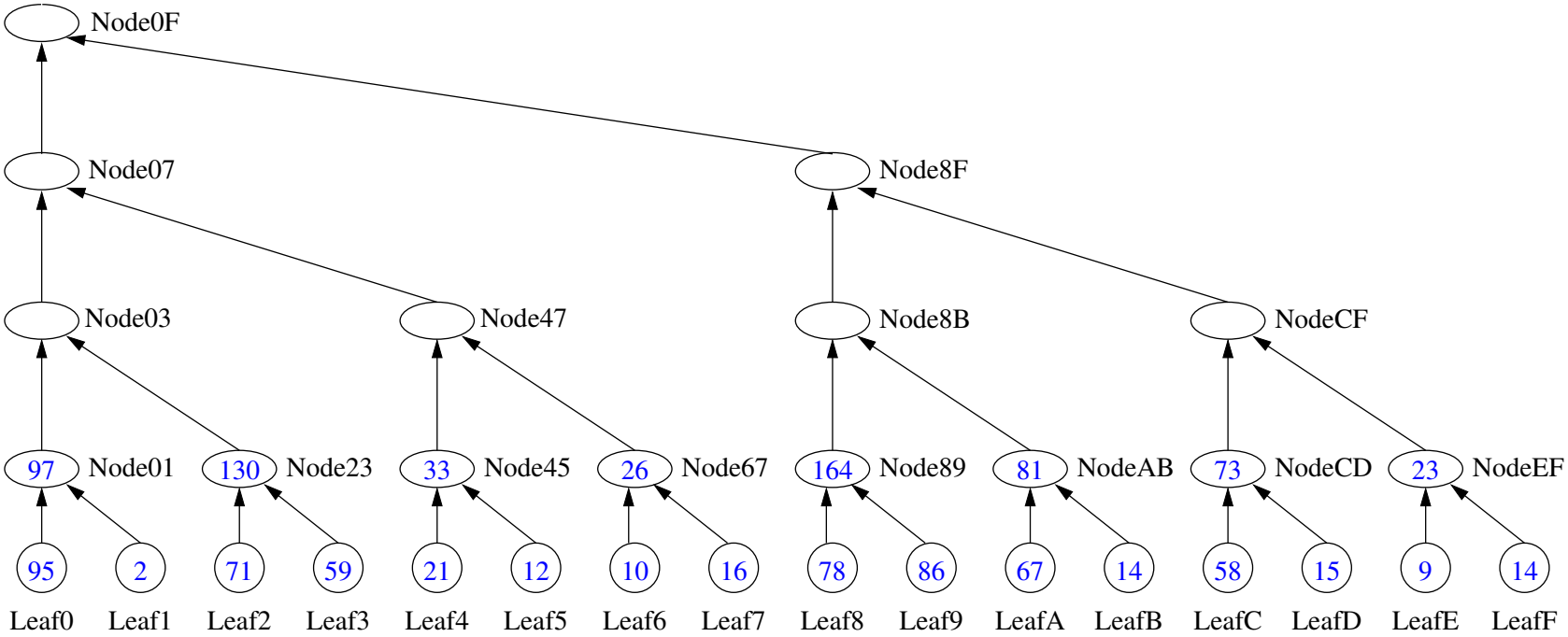
- Upward Pass: reduce
  - Each node receives values from left and right subtrees.
  - The node combines these values, and sends the results to the parent.
  - The combine at the root node produces the total for all nodes.
- Downward Pass:
  - Each node has the values from its left and right children
  - When a node receives a value from its parent:
    - ★ it sends the parent's value to its left subtree – this is the "sum" of everything to the left of the left subtree.
    - ★ it combines the parent's value with the value from the left subtree and sends this to the right subtree – this is the "sum of everything to the left of the right subtree.
  - When a leaf receives a value from its parent
    - ★ It records the value as its result (exclusive scan),
    - ★ Or, it combines this value with its own value and records that as its result (inclusive scan)

The upward pass



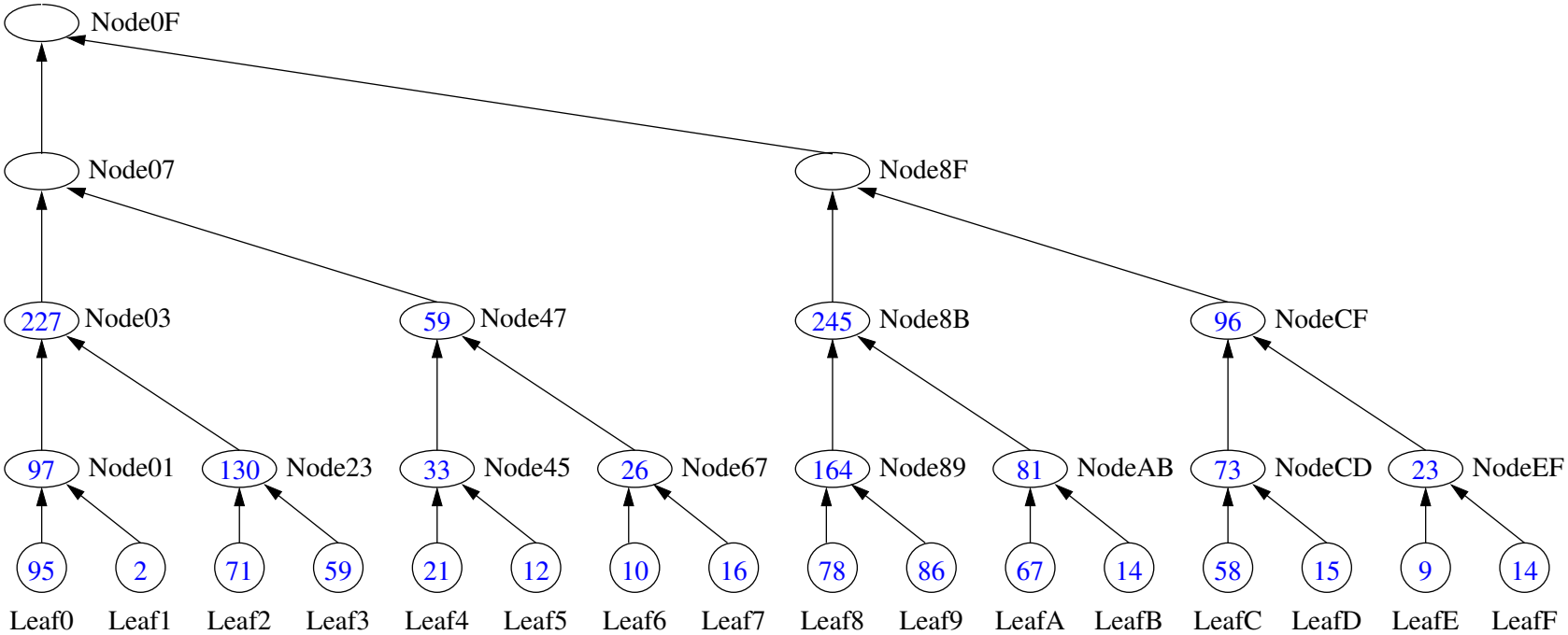
Each leaf sends its value to its parent.

The upward pass



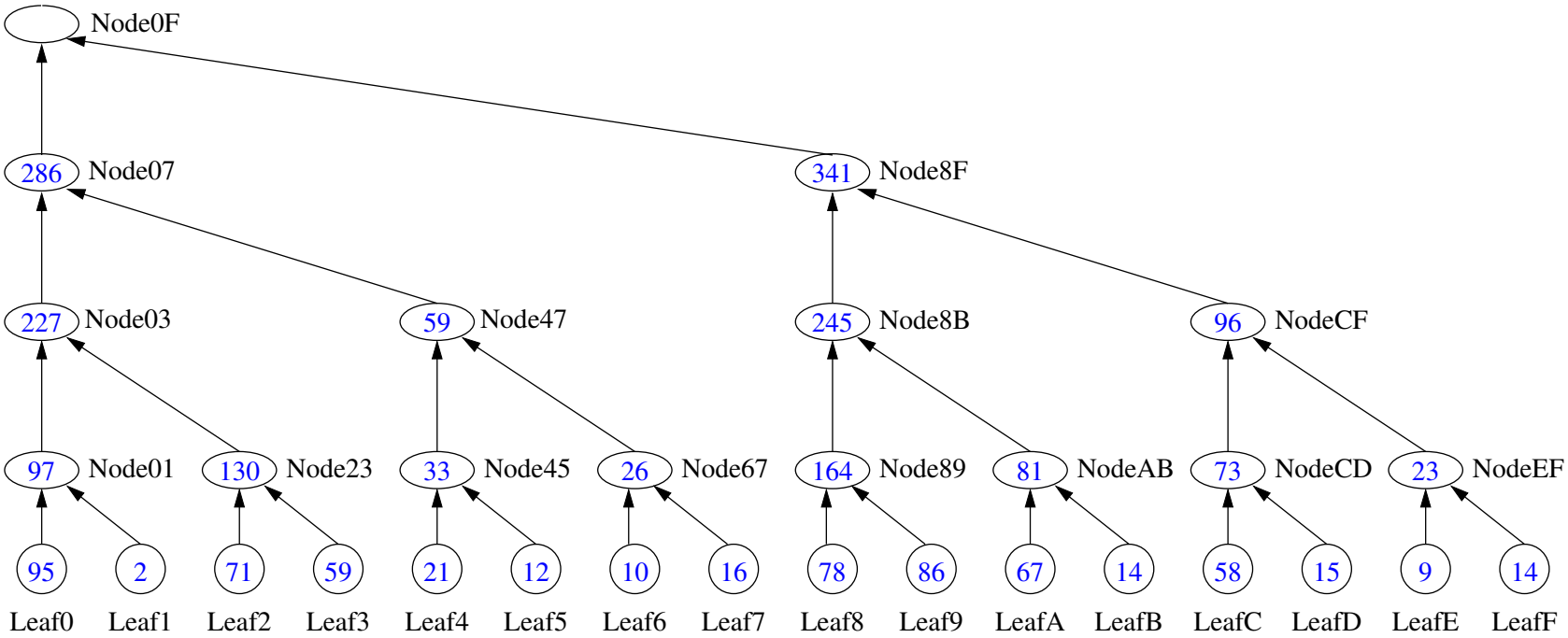
Node01, Node23, . . . , NodeEF combine values from their leaves and send the results to their parents.

The upward pass



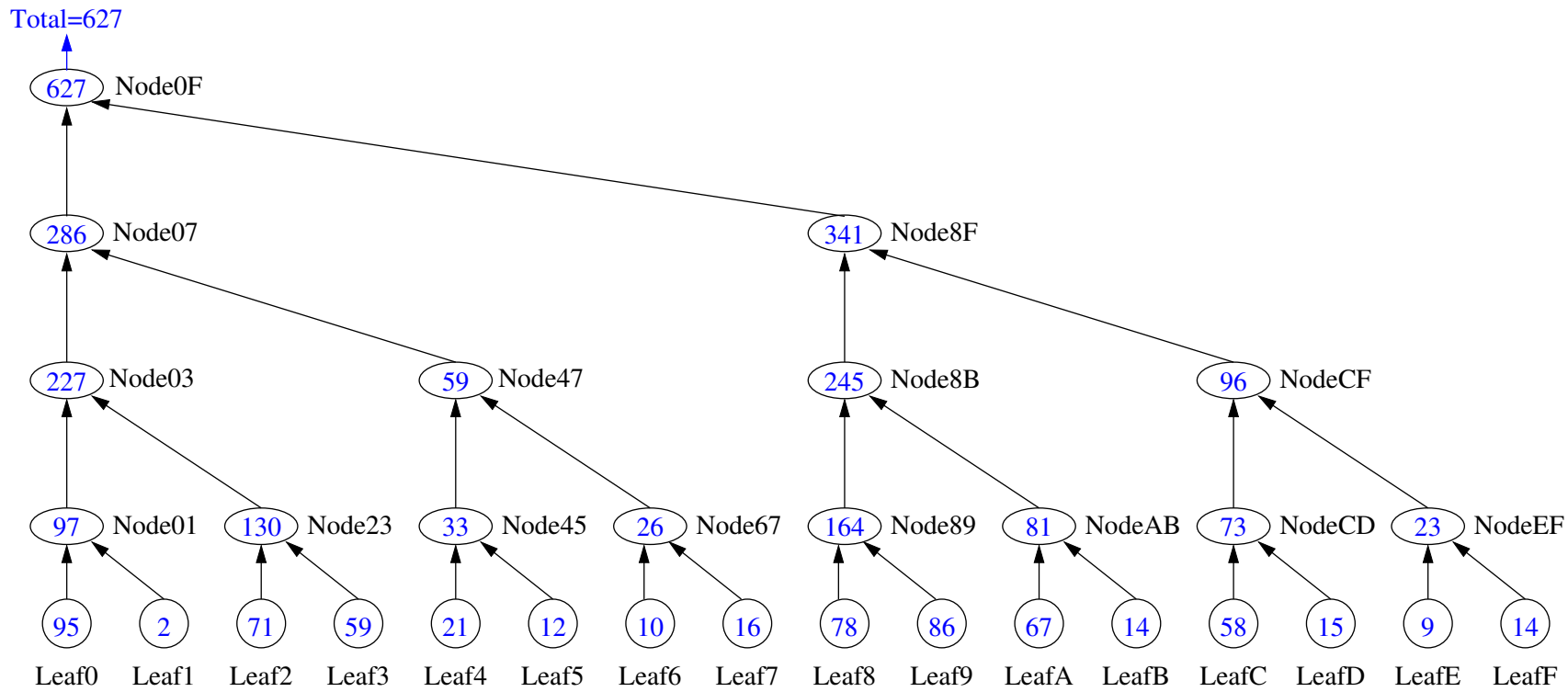
Node03, Node47, Node8B, and NodeCF combine values from their subtrees and send the results to their parents.

The upward pass



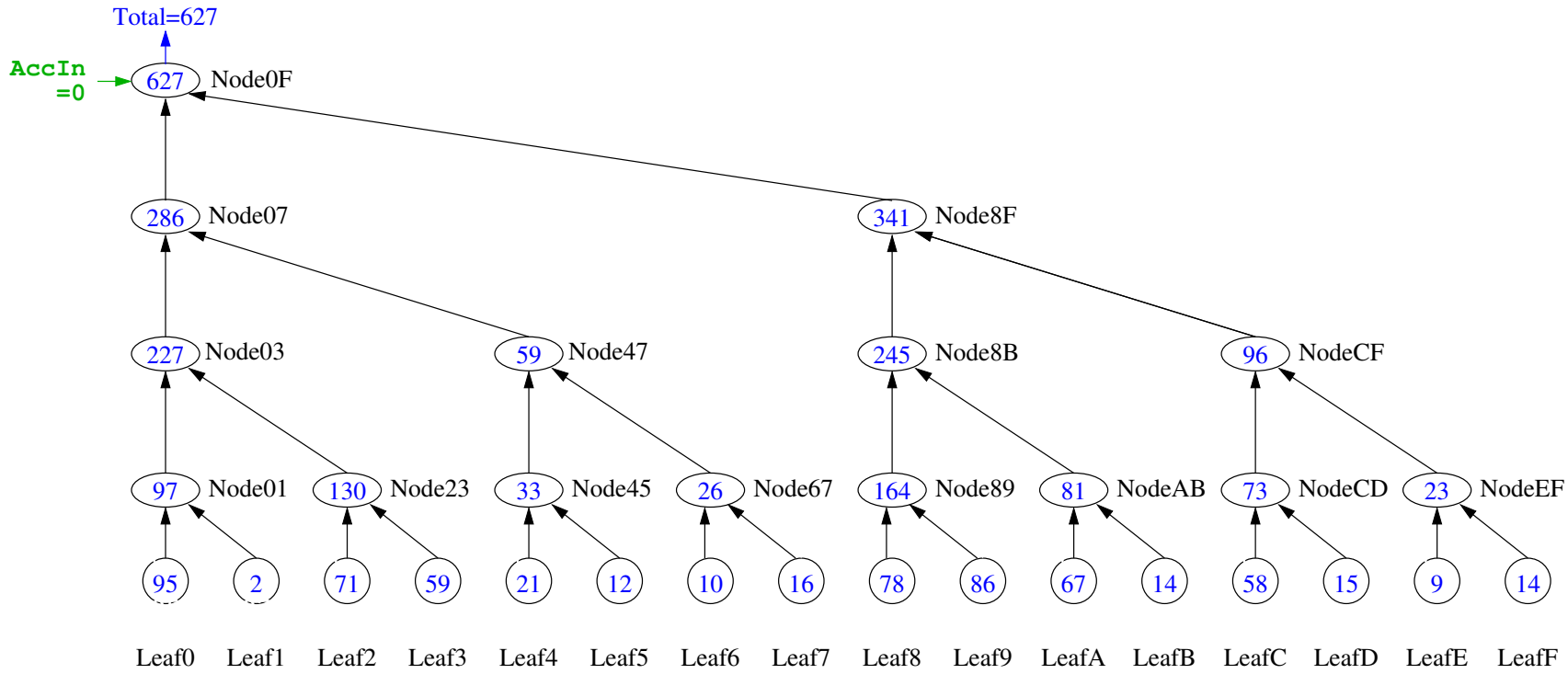
Node07, and Node8F combine values from their subtrees and send the results to the root.

The upward pass



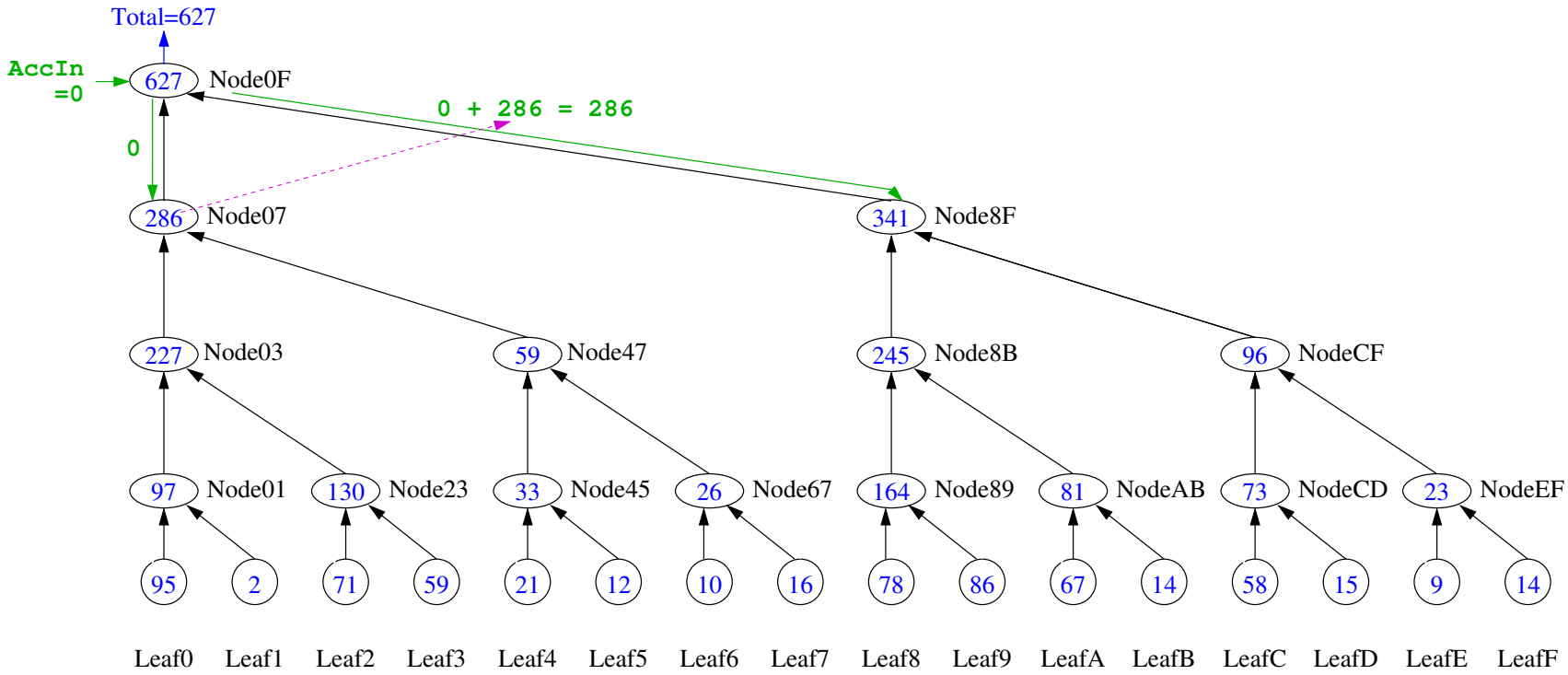
Node0F (the root) computes the grand total.

The downward pass



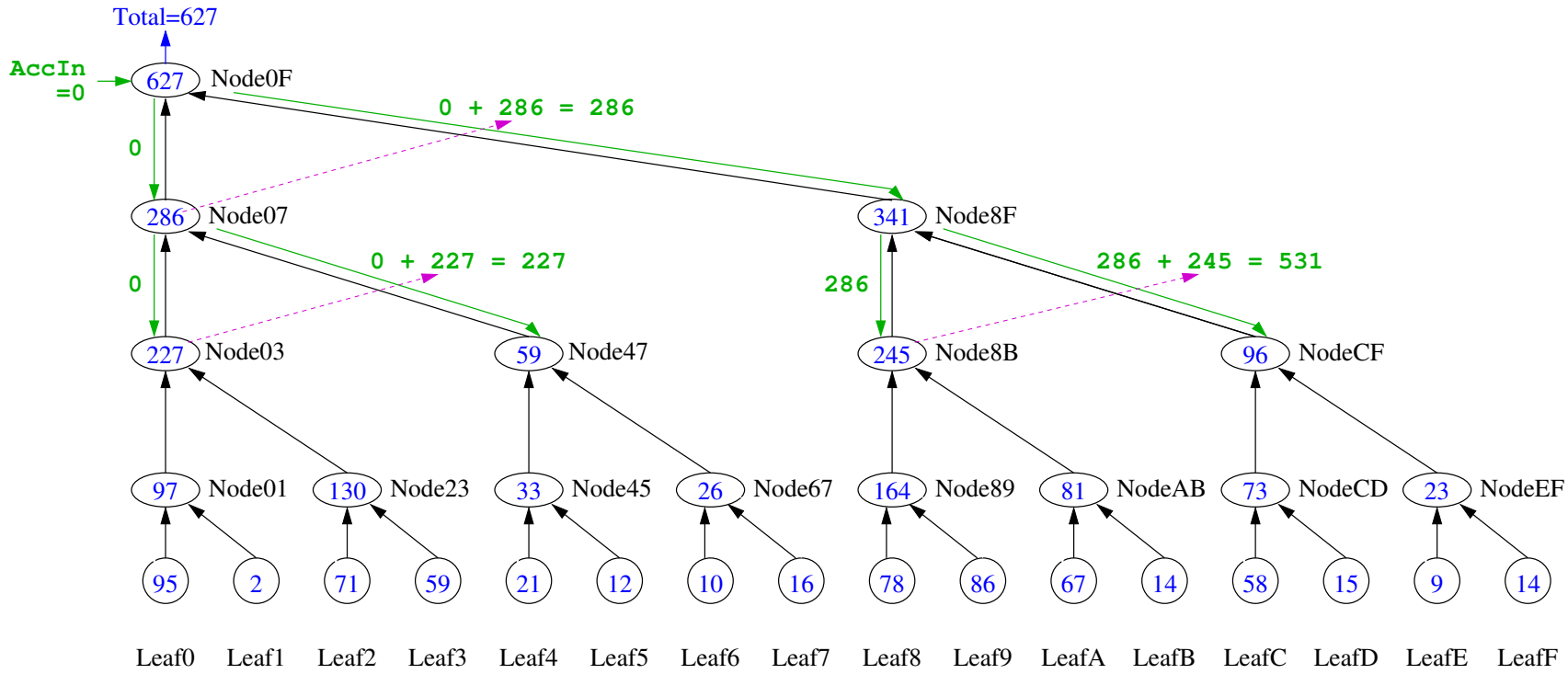
The root node takes *AccIn* as the value of “everything to the left”.

The downward pass



The root node sends *AccIn* to the left and *AccIn* + *Total* to the right – each subtree receives the value of everything to its left.

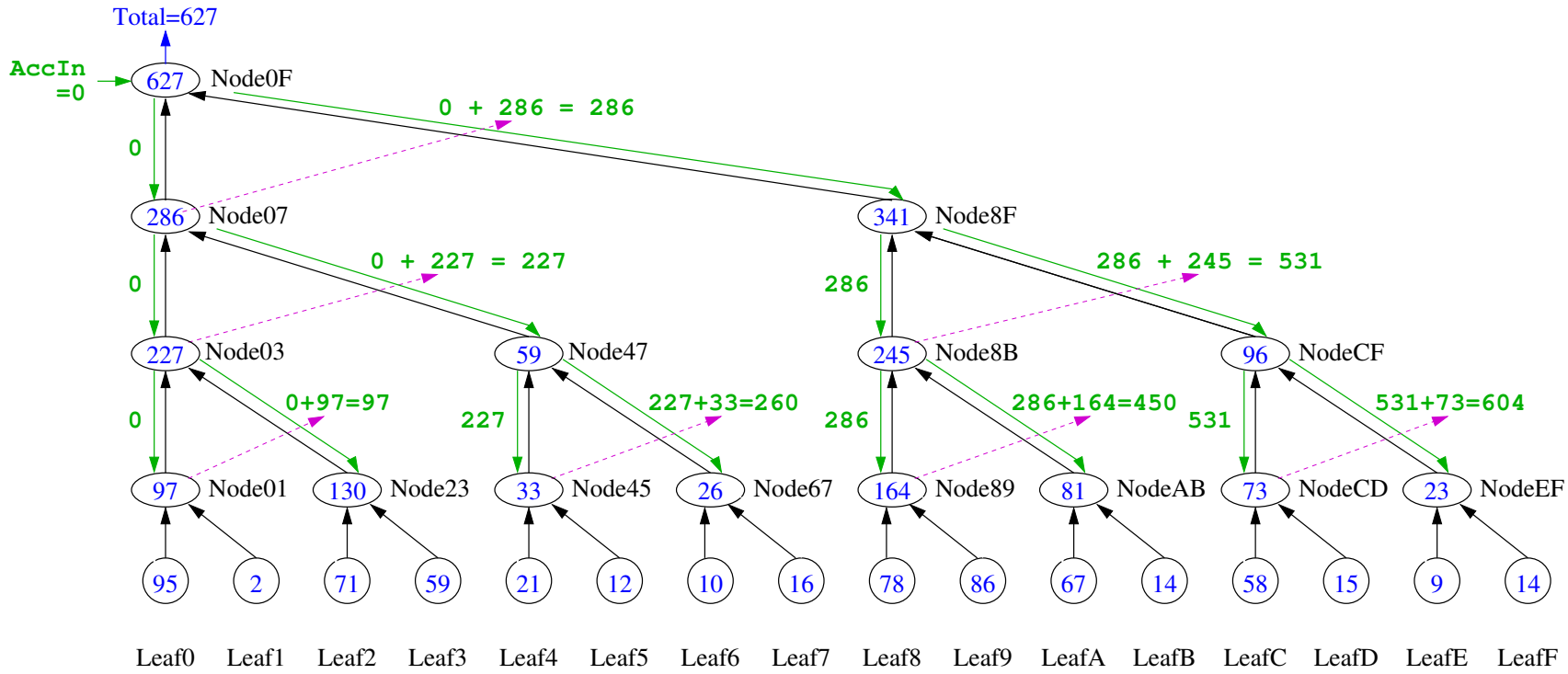
The downward pass



Node07 and Node8F each

- send the value from their parent to their left subtree.
- combine the value from their parent with the value from their left subtree and send that to their right subtree.

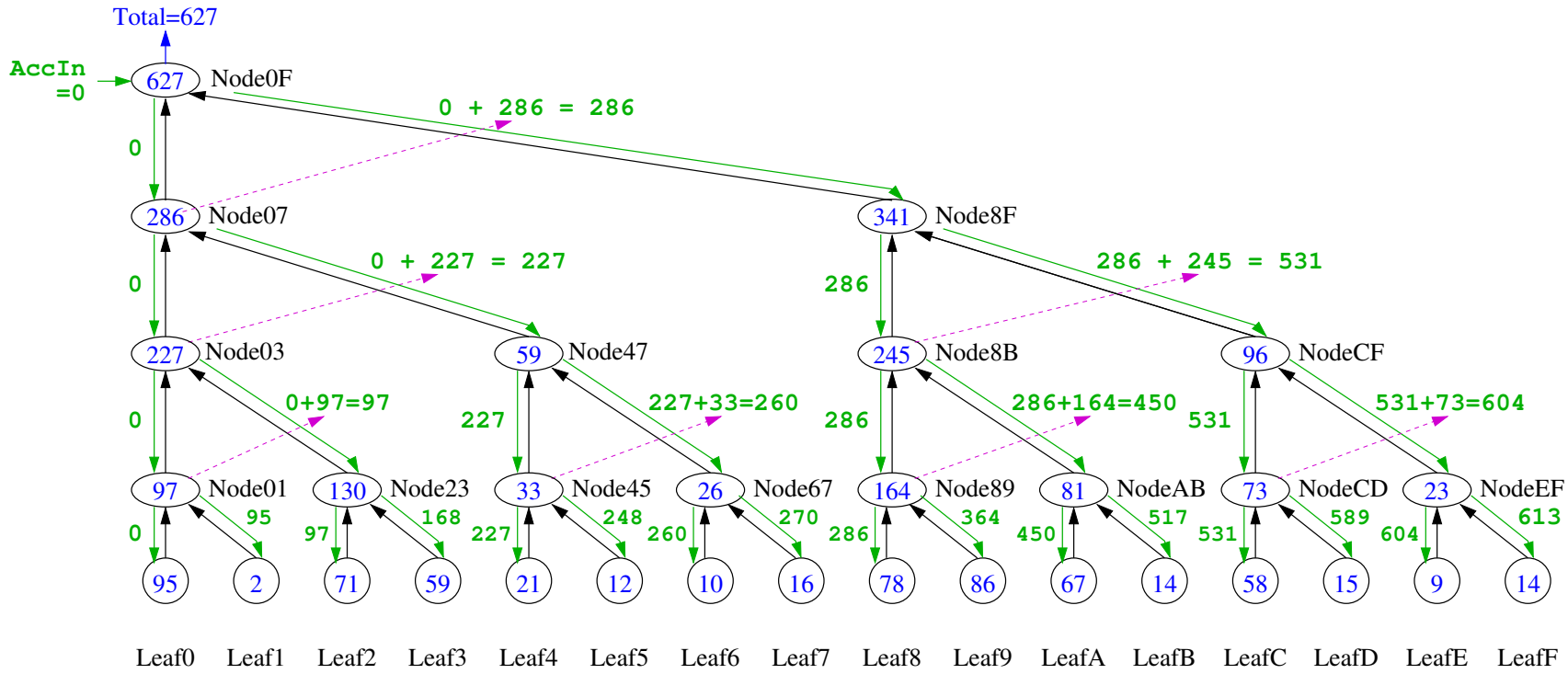
The downward pass



Node03, Node47, Node8B, and NodeCF each

- send the value from their parent to their left subtree.
- combine the value from their parent with the value from their left subtree and send that to their right subtree.

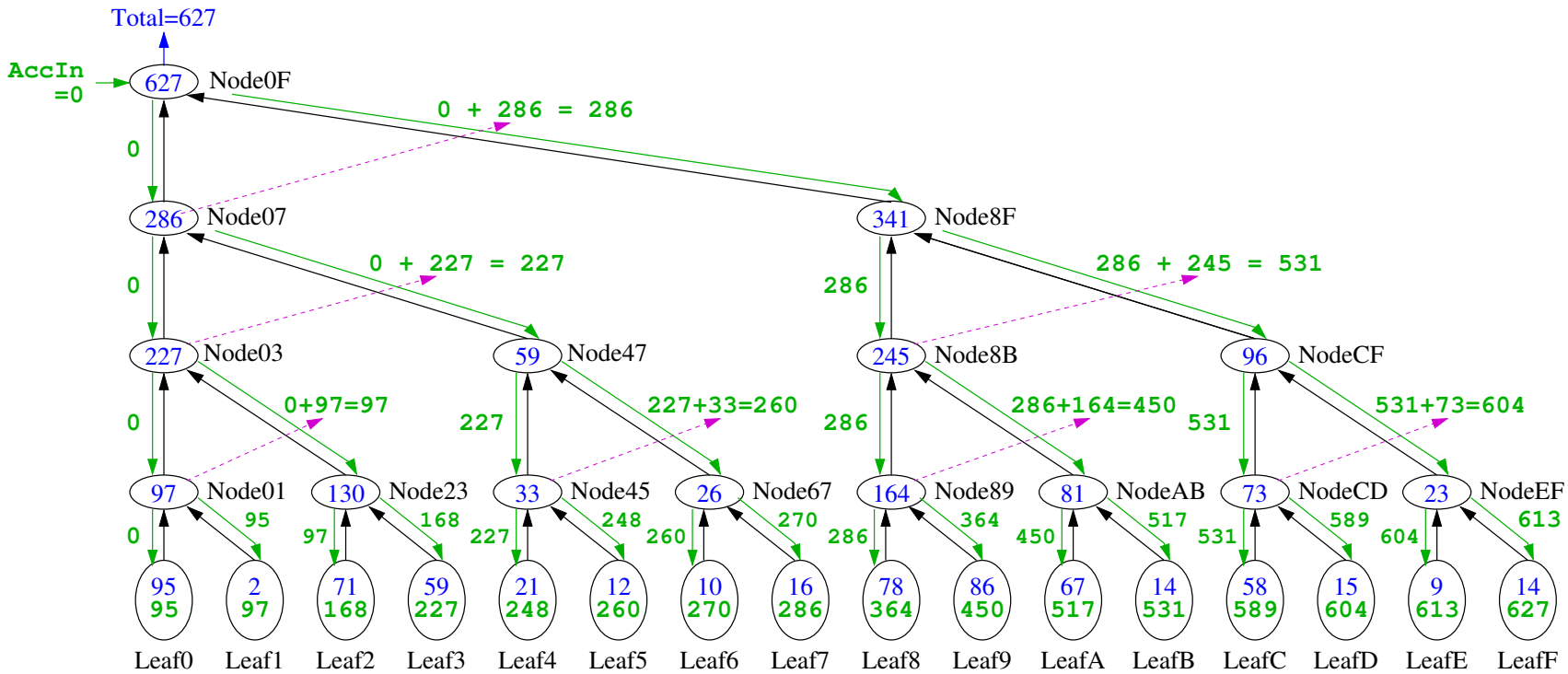
The downward pass



Node01, Node 23, . . . , NodeEF each

- send the value from their parent to their left subtree.
- combine the value from their parent with the value from their left subtree and send that to their right subtree.

The downward pass



The leaves:

- receive values from their parents,
- and add their own value (inclusive scan)

`wtree:scan(`*Leaf1, Leaf2, Combine, Acc0*`)`

- *Leaf1* `(` *ProcState* `)` `->` *Value*
  Each worker process computes its *Value* based on its *ProcState*.
- *Combine* `(` *Left*`,` *Right* `)` `->` *Value*
  Combine values from sub-trees.
- *Leaf2* `(` *ProcState*`,` *AccIn* `)` `->` *ProcState*
  Each worker updates its state using the *AccIn* value – i.e. the accumulated value of everything to the worker's "left".
- *Acc0*: The value to use for *AccIn* for the leftmost nodes in the tree.

Implementing cumulative sum with `wtree:scan`

```
sum_scan(W, SrcKey, DstKey) ->
  wtree:scan(W,
    fun(ProcState) -> % Leaf1
      sum(workers:get(ProcState, SrcKey))
    end,
    fun(ProcState, AccIn) -> % Leaf2
      MyList = workers:get(ProcState, SrcKey),
      {Result, _Total} = mapfoldl(
                           fun(E,Acc) -> V = E+Acc, {V,V} end,
                           AccIn, MyList),
      % workers:put returns the updated ProcState
      workers:put(ProcState, DstKey, Result)
    end,
    fun(Left, Right) -> Left + Right end, % Combine
    0, % Acc0
  ).
```

Let's Try It!

```
1> code:add_path("/home/c/cs418/public_html/resources/erl").
true
2> c(scan).
{ok,scan}
3> W = wtree:create(4).
[<0.76.0>,<0.77.0>,<0.78.0>,<0.79.0>]
4> workers:rlist(W, 20, 100, data).
ok
5> scan:sum_scan(W, data, cumsum).
974 % the total of all elements of 'data'
6> lists:append(workers:retrieve(W, data)).
% the original list
[2,77,37,26,34,91,58,45,68,39,82,3,29,8,32,97,67,57,77,45]
7> lists:append(workers:retrieve(W, cumsum)).
% cumulative sum of 'data'
[2,79,116,142,176,267,325,370,438,477,
 559,562,591,599,631,728,795,852,929,974]
```

Simple Reduce & Scan Examples

- We can replace `+` with any associative operator.
- Simple examples include:
  - `max`, `min`
  - `*` – but often not very useful
    - ★ Floating point `*` prone to overflow or underflow for folding long lists.
    - ★ Integer `*` has an implementation in Erlang that is $\mathcal{O}(N^2)$ where N is the number of digits – the time is dominated by the final multiplication at the root.
  - `and`, `or`, `band`, `bor`, `bxor`

Composing with filters and/or maps

- We can apply reduce or scan to a filtered list
  - Only combine the elements that satisfy the filter.
- We can apply reduce or scan to a mapped list
  - Transform each element of the list and then

Second Largest

The Algebraic Pattern

The Longest Pattern

Preview

September 21: Shared Memory Architectures
Reading: Pacheco chapter 2.1 - 2.3
Homework: Homework 1 due due **11:59pm**

September 24: Message Passing Architectures

September 26: Superscalar Architectures

Sept. 28 – Oct. 5: Performance Analysis

October 8 – 15: Sorting

October 17: Intro. to CUDA

October 19: Midterm review

October 22: Midterm

Oct. 24 – Nov. 30: Data Parallel Computing, GPUs, and CUDA