Scan

Mark Greenstreet

CpSc 418 – Jan. 20, 2016

Objectives

- Prefix sum
  - Spawning processes.
  - Sending and receiving messages.
- The source code for the examples in this lecture is available here: procs.erl.

Prefix Sum

- Scan is similar to reduce, but every process calculates its cumulative total.
- Example:

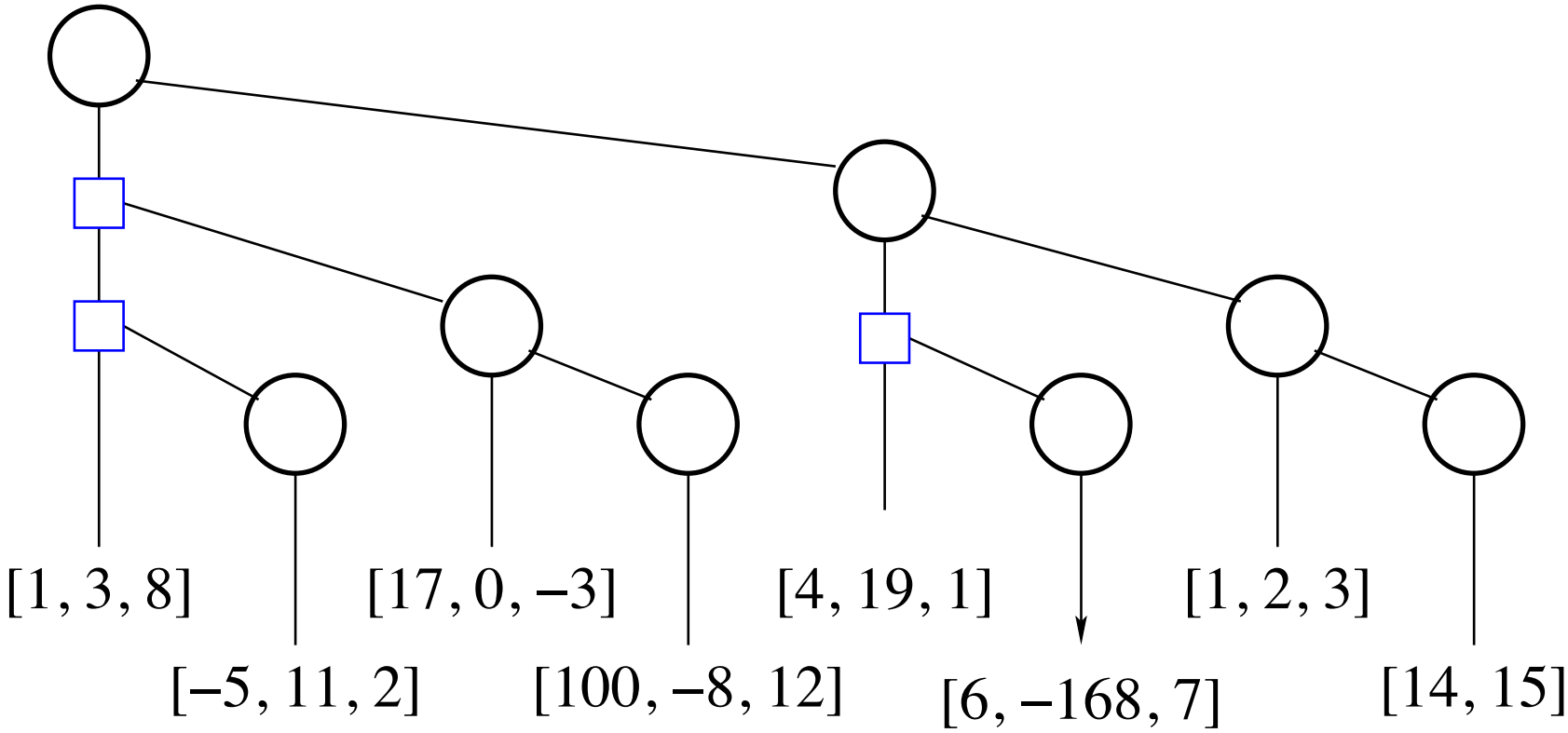
```
% prefix_sum: compute prefix sum.
prefix_sum(L) when is_list(L) -> prefix_sum_tr(L, 0).
prefix_sum_tr([], Acc) -> [];
prefix_sum_tr([H | T], Acc) ->
    MySum = H+Acc,
    [MySum | prefix_sum_tr(T, MySum)].
```

- Let's try it:

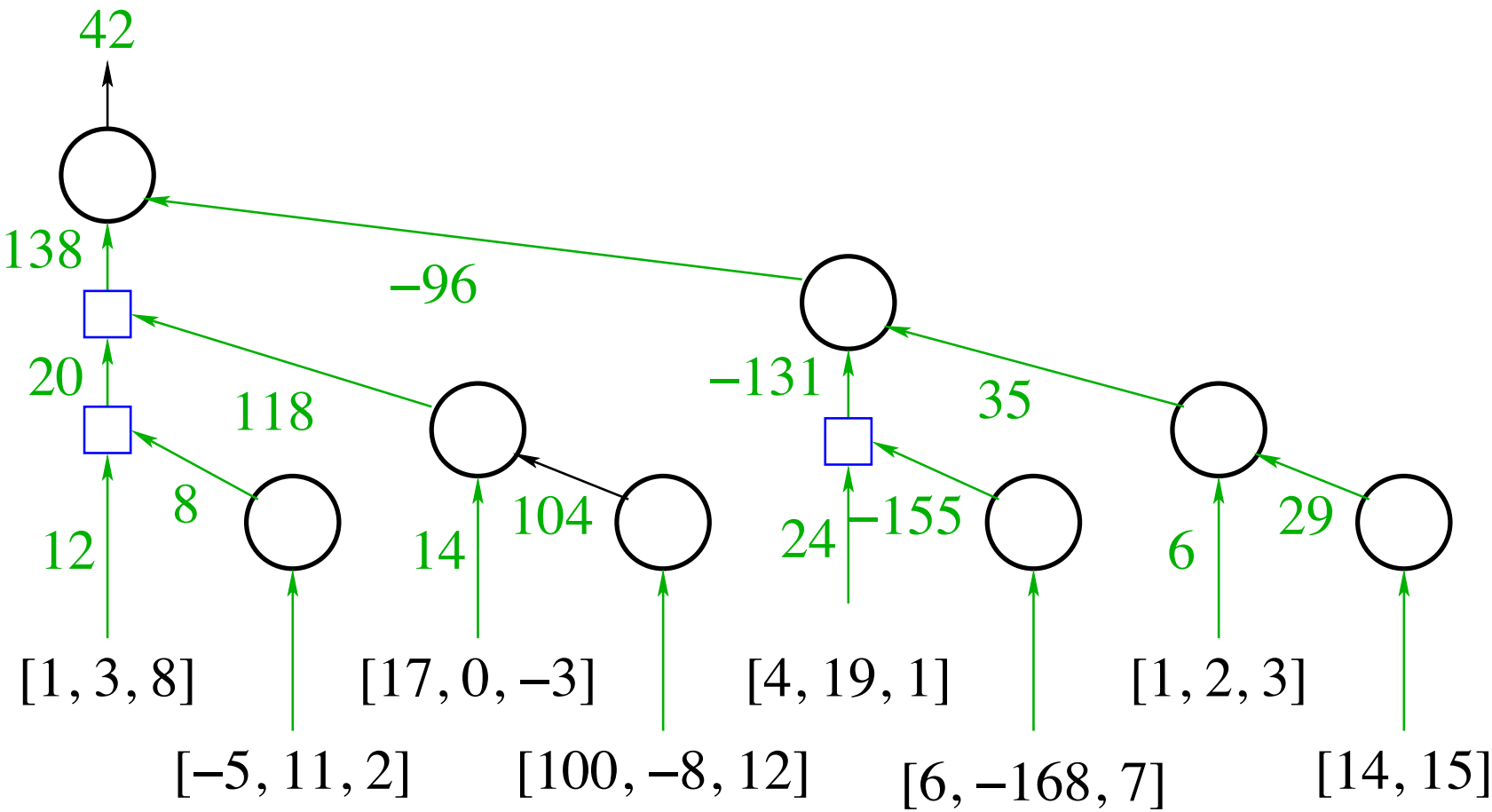
```
1> examples:prefix_sum([1, 13, 2, -5, 17, 0, 33]).
[1,14,16,11,28,28,61]
```

- How can we do this in parallel?

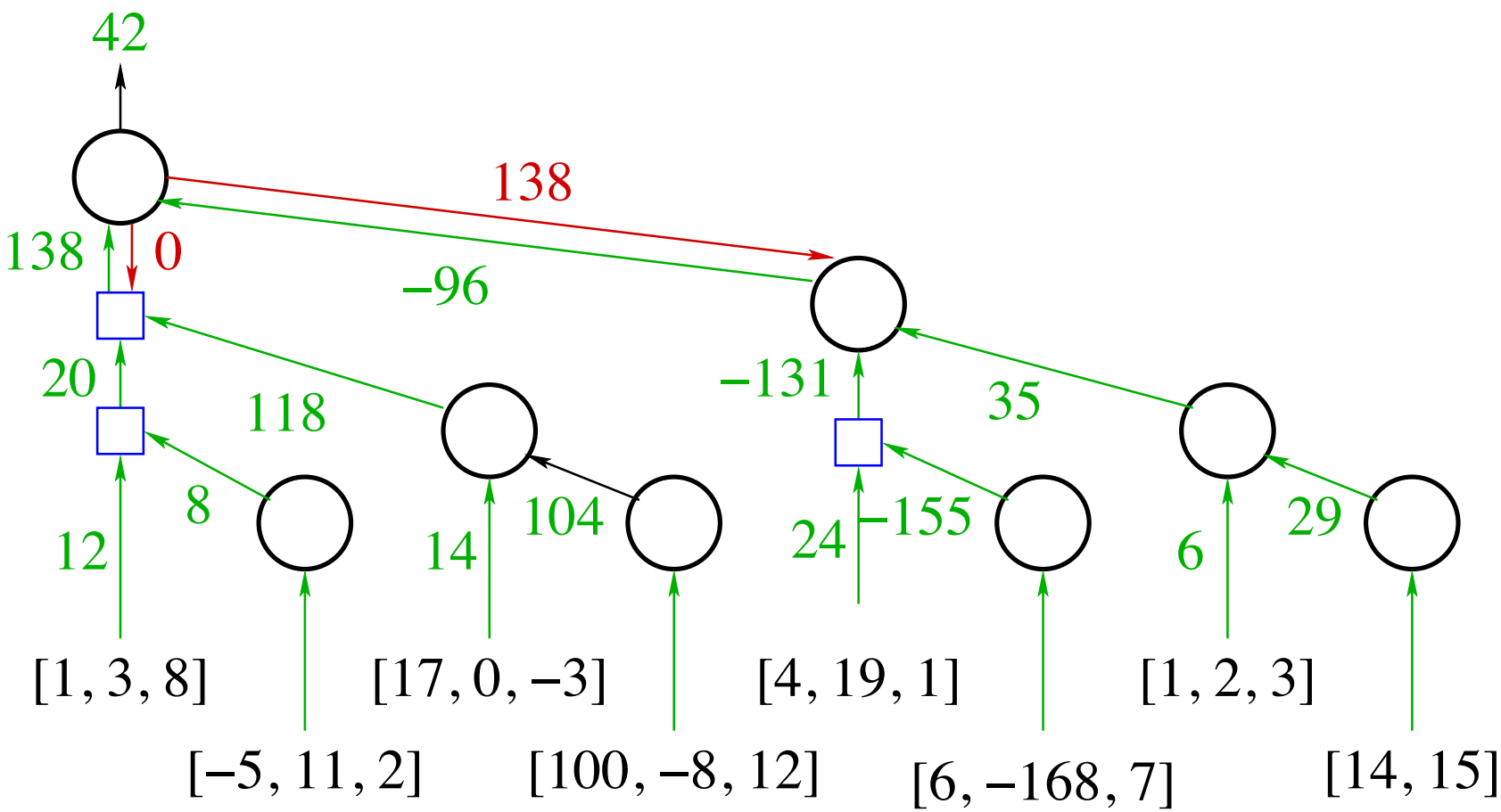
Parallel Prefix Sum



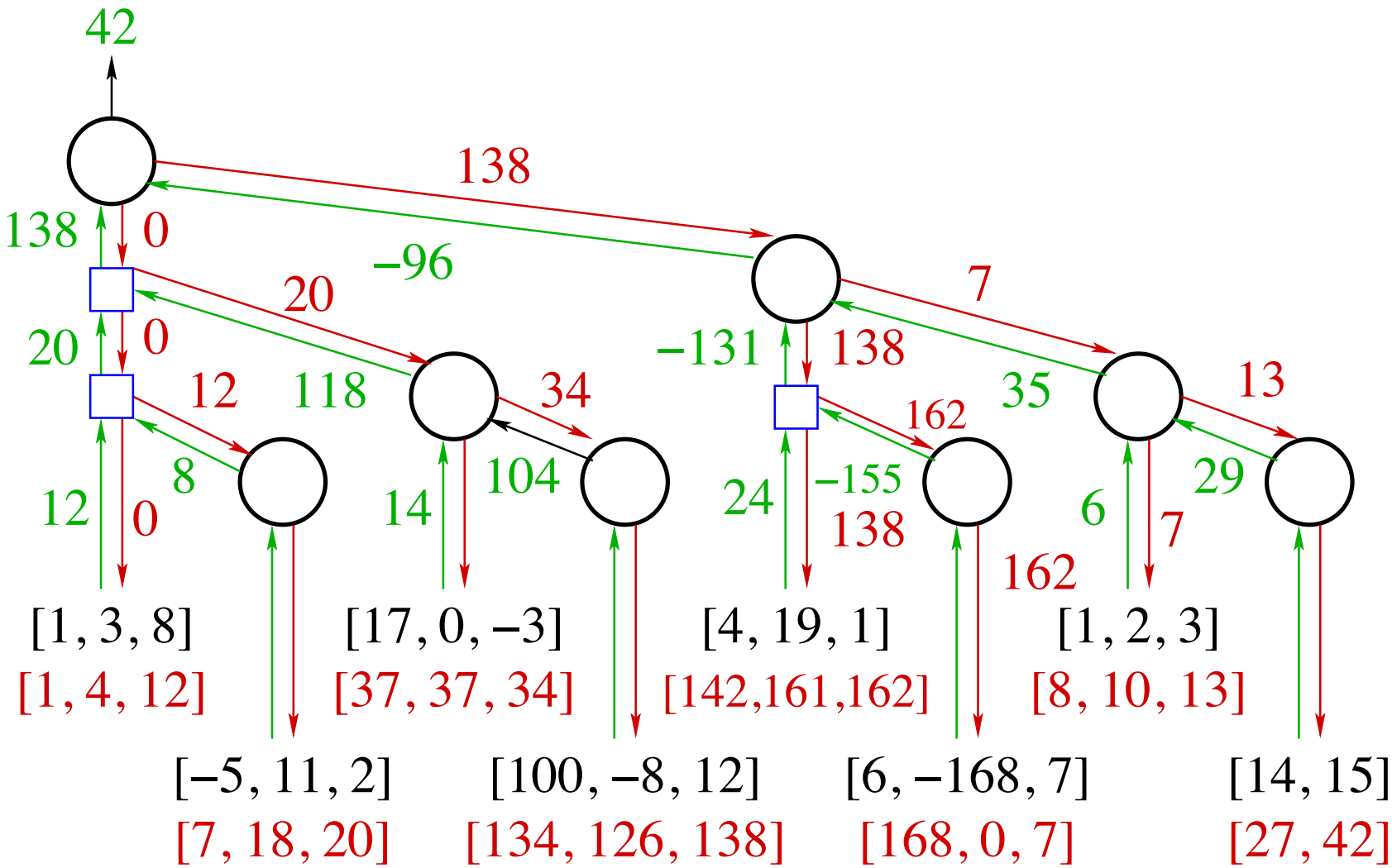
Parallel Prefix Sum



Parallel Prefix Sum



Parallel Prefix Sum



The Scan Pattern

- It's a parallel version of *mapfold*, e.g. `lists:mapfoldl` and `lists:mapfoldr`.
- `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.

Scan example: prefix sum

```
prefix_sum_par(W, Key1, Key2) ->
   wtree:scan(W,
      fun(ProcState) -> % Leaf1
         lists:sum(wtree:get(ProcState, Key1)) end,
      fun(ProcState, AccIn) -> % Leaf2
         wtree:put(ProcState, Key2,
            prefix_sum(wtree:get(ProcState, Key1), AccIn)
         ) end,
      fun(Left, Right) -> % Combine
         Left + Right end,
      0 % Acc0
   ).

prefix_sum(L, Acc0) ->
   element(1,
      lists:mapfoldl(fun(X, Y) -> Sum = X+Y, {Sum,Sum} end,
                     Acc0, L)).
```

Prefix Sum Using Scan, example (part 1 of 4)

- Consider the example from slide 4.
  - We'll assume that the original lists for each processes are associated with the key `raw_data`.
  - We'll store the cummulative sum using the key `cooked_data`.
- `Leaf1`: each worker computes the sum of the elements in its list:
  - Worker 0:

```
Leaf1(ProcState) ->
    lists:sum(wtree:get(ProcState, raw_data)) ->
    lists:sum([1,3,8]) ->
12.
```

  - Worker 1:

```
Leaf1(ProcState) -> lists:sum([-5,11,2]) -> 8.
```

  - Worker 2:

```
Leaf1(ProcState) -> lists:sum([17,0,-3]) -> 14.
```

  - Workers 3–6: ...
  - Worker 7:

```
Leaf1(ProcState) -> lists:sum([14,15]) -> 29.
```

Prefix Sum Using Scan, example (part 2 of 4)

- `Combine` (upward, first round):
  - Worker 0: `Combine(12, 8) -> 20`.
  - Worker 2: `Combine(14, 104) -> 118`.
  - Worker 4: `Combine(24, -155) -> -131`.
  - Worker 6: `Combine(6, 29) -> 35`.
- `Combine` (upward, second round):
  - Worker 0: `Combine(20, 118) -> 138`.
  - Worker 4: `Combine(-131, 35) -> -96`.
- `Combine` (upward, final round):
  - Worker 0: `Combine(138, -96) -> 42`.
  - This value is returned to the caller of `wtree:scan`.

Prefix Sum Using Scan, example (part 3 of 4)

- `Combine` (downward)
- The root sends `AccIn`, `0` to the left subtree.
- Each worker that did a combine remembers the arguments from the upward combines, and uses them in the downward sweep. In the code, each upward step is a recursive function call, and each downward step is a return.
- `Combine` (downward, first round)
  - Worker 0: `Combine(0, 138) -> 138`.
  - The `0` is `AccIn` from the root.
  - The `138` is the stored value from the left subtree.
  - Worker 0 sends this result to its right subtree, worker 4.
- `Combine` (downward, second round)
  - Worker 0: `Combine(0, 20) -> 20`. Send to worker 2.
  - Worker 4: `Combine(138, -131) -> 7`. Send to worker 6.
- `Combine` (downward, third round)
  - Worker 0: `Combine(0, 12) -> 12`. Send to worker 1.
  - Worker 2: `Combine(20, 14) -> 34`. Send to worker 3.
  - Worker 4: `Combine(138, 24) -> 162`. Send to worker 5.
  - Worker 6: `Combine(7, 6) -> 13`. Send to worker 7.

Prefix Sum Using Scan, example (part 4 of 4)

- `Leaf2` (update worker state)
  - Worker 0:

```
Leaf2(ProcState, 0) ->
   wtree:put(ProcState, Key2,
      prefix_sum(wtree:get(ProcState, Key1), 0)) ->
   wtree:put(ProcState, Key2,
      prefix_sum([1, 3, 8], 0)) ->
   wtree:put(ProcState, Key2, [1, 4, 12]).
```

  - Worker 1:

```
Leaf2(ProcState, 0) ->
   wtree:put(ProcState, Key2,
      prefix_sum(wtree:get(ProcState, Key1), 0)) ->
   wtree:put(ProcState, Key2,
      prefix_sum([-5, 11, 2], 12)) ->
   wtree:put(ProcState, Key2, [7, 18, 20]).
```

  - Workers 2–7: . . .

Let's Try It

```
2> W = wtree:create(8).
[<0.65.0>,<0.66.0>,<0.67.0>,<0.68.0>
 <0.69.0>,<0.70.0>,<0.71.0>,<0.72.0>]
3> workers:update(W, raw_data,
   [ [1,3,8], [-5,11,2], [17,0,-3], [100,-8,12],
     [4,19,1], [6,-168,7], [1,2,3],  [14,15]]).
ok
4> examples:prefix_sum_par(W, raw_data, cooked_data).  42
5> workers:retrieve(W, cooked_data).
[ [1,4,12], [7,18,20], "%%\"", [134,126,138],
  [142,161,162], [168,0,7], "\b\n\r", "\e*"] 6> $37
```

- Likewise, `$" == 34`, `$== 8`, `$\n == 10`, `$\r == 13`, `$\e == 27`, and `$* == 42`.
- All is well.

More Examples of scan

- Account balance with interest:
  - Input: a list of transactions, where each transaction can be a deposit (add an amount to the balance), a withdrawal (subtract an amount from the balance), or interest (multiply the balance by an amount). For example:

    ```
    [{deposit, 100.00}, {withdraw, 5.43}, {withdraw, 27.75}
    ```

  - Output: the account balance after each transaction. For example, if we assume a starting balance of $1000.00 in the previous example, we get

    ```
    [1100.00, 1094.57, 1066.82, 1067.40, ...]
    ```

- Delete 3s
  - Given a list that is distributed across *NProc* processes, delete all 3s, and rebalance the list so each process has roughly the same length sublisth.
  - Solution (sketch):
    - ★ Using scan, each process determines how many 3s preceed its segment, the total list length preceeding it, and the total list length after deleting 3s.
    - ★ Each process deletes its 3s and send portions of its lists and/or receives list portions to rebalance.

More^2 Examples of scan

- Carry-Lookahead Addition:
  - Given two large integers as a list of bits (or machine words), compute their sum.
    - ★ Note that the "pencil-and-paper" approach works from the least significant bit (or digit, or machine word) and works sequentially to the most-significant bit. This takes $O(N)$ time where N is the number of bits in the work.
  - Carries can be computed using scan.
    - ★ This allows a parallel implementation that adds two integers in $O(\log N)$ time.
    - ★ This is how the hardware in your CPU does addition – the adder takes $O(\log N)$ gate delays to add two, machine words, where N is the number of bits in a word.
- See *Principles of Parallel Programming*, pp. 119f.
- See homework 2 (later today, I hope).

Preview

January 23: Architecture Review	
Reading:	Pacheco, Chapter 2, Sections 2.1 and 2.2.
January 25: Shared-Memory Machines	
Reading:	Pacheco, Chapter 2, Section 2.3
January 27: Distributed-Memory Machines	
Reading:	Pacheco, Chapter 2, Sections 2.4 and 2.5.
Mini Assignments	Mini 4 goes out.
January 30: Parallel Performance: Speed-up	
Reading:	Pacheco, Chapter 2, Section 2.6.
Homework:	HW 2 earlybird (11:59pm). HW 3 goes out.
February 1: Parallel Performance: Overheads	
Homework:	HW 2 due (11:59pm).
February 3: Parallel Performance: Models	
Mini Assignments	Mini 3 due (10am)
February 6: Parallel Performance: Wrap Up	
January 8–February 15: Parallel Sorting	
Homework (Feb. 15):	HW 3 earlybird (11:59pm), HW 4 goes out.
February 17: Map-Reduce	
Homework:	HW 3 due (11:59pm).
February 27: TBD	
March 1: Midterm	

Review Questions

- What is scan? Give an example.
- Compare scan with `lists:mapfoldl`?
- What property must an operator have to be amenable use with scan?
- What are the components of a generalized scan?
  As an example, what functions do you need to define to use `wtree:scan`?
- Consider the following variations on the bank account problem:
  - Add a transaction `{reset, Balance}`, where `Balance` is a number. The account balance is set to this amount. For example, this can be used to open an account with an initial balance. We'll also assume that a `reset` can be done at any point in a sequence of transactions.
  - Change interest computations so that the bank charges a daily interest of X% for negative balances, neither charges nor pays interest for positive balances less than \$1000, and pays a daily interest of Y% for positive balances greater than \$1000.
  - For each of these:
    - ★ Can the account balance still be computed using scan?
    - ★ If yes, explain how to do. If no, explain why it's not possible.