Parallel Scan

Bryan Mills, PhD

Spring 2017

Scan

- Prefix Sum Example

INPUT	1	2	3	4	5	6	7	8
OUTPUT	1	3	6	10	15	21	28	36

Generalized Scan

- f(x,y) = operation
  - Can be any binary associative operation
    - sum, multiply, logical and/or, max, min
- 0 = Identity element
  - zero for sum, one for multiplication, true for logical, false for logical or, 0 for unsigned max

Generalized Scan

```
Type acc = identity;
for (i = 0; I < elems.length(); i++) {
   acc = f(acc, elems[i]);
   out[i] = acc;
}
```

- How many steps?

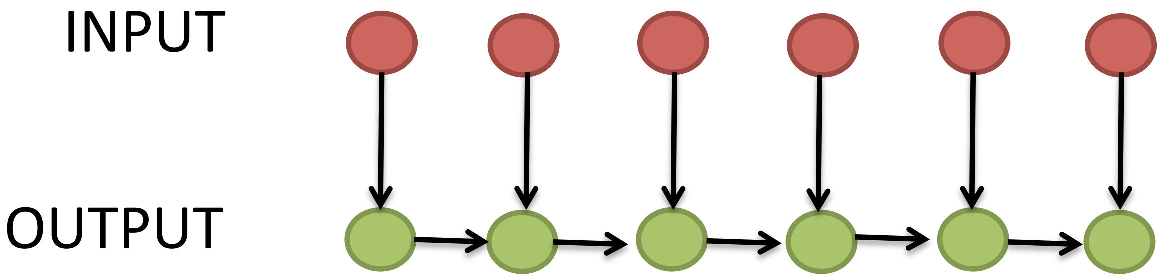
  n

- How much work?

  n

Why?

- Many problems look like this
  - Output depends on one new input and previous output
  - Balancing checkbook, regressions, even sorting can.



INPUT

OUTPUT

Types of Scan

INPUT	1	2	3	4	5	6	7	8

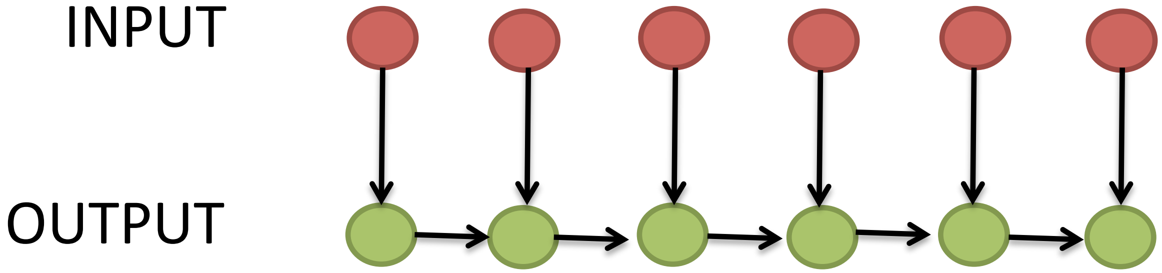
- Exclusive
  - Output all elements **ex**cluding the current

OUTPUT	0	1	3	6	10	15	21	28

- Inclusive
  - Output all elements **in**cluding the current

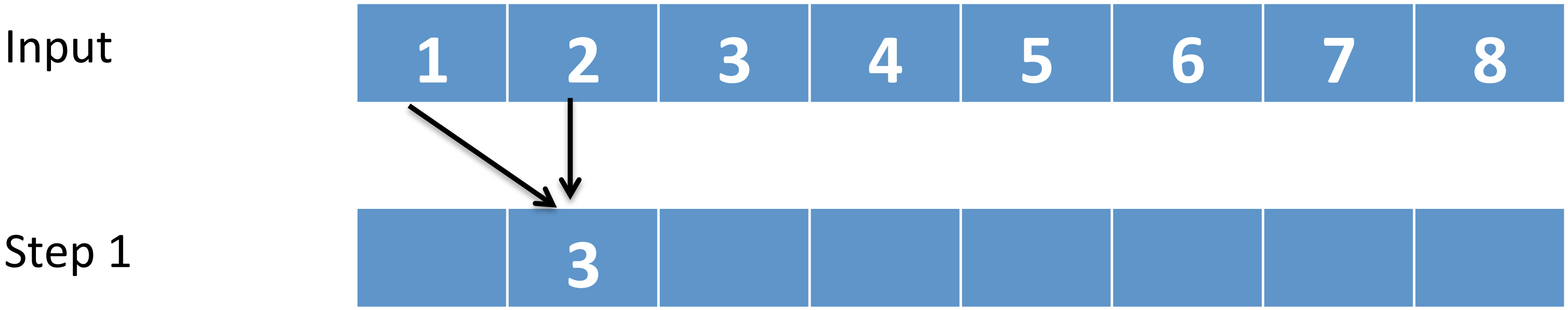
OUTPUT	1	3	6	10	15	21	28	36

Looks serial



- It is non-obvious how to convert to parallel however we will look at two algorithms
  – Hillis/Steele
  – Blelloch

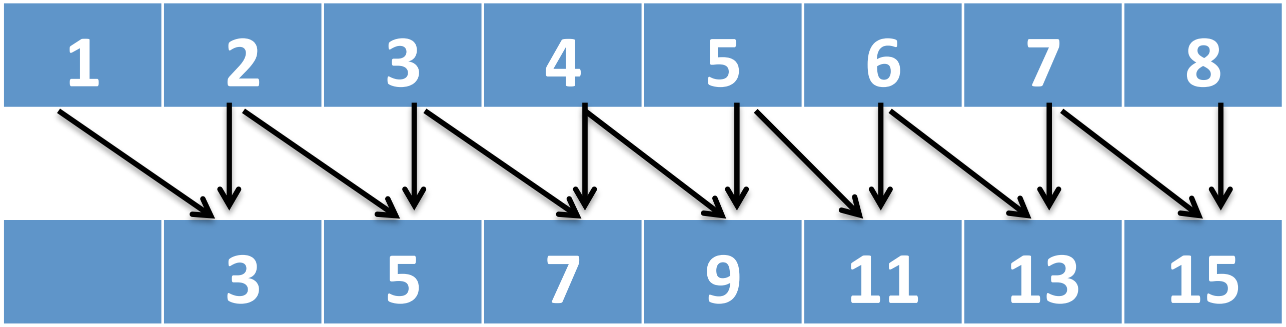
Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 1		3						

Add direct neighbors n = 1

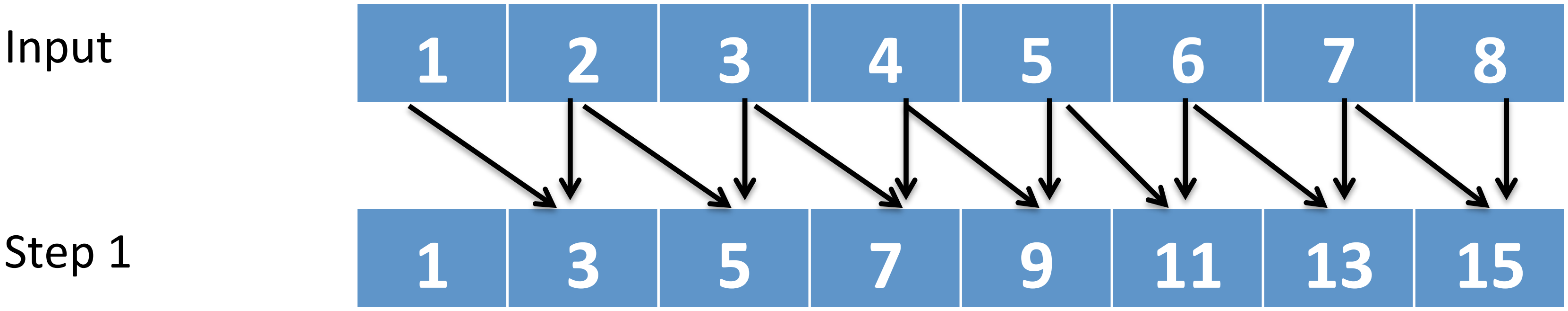
Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 1		3	5	7	9	11	13	15

Add direct neighbors n = 1

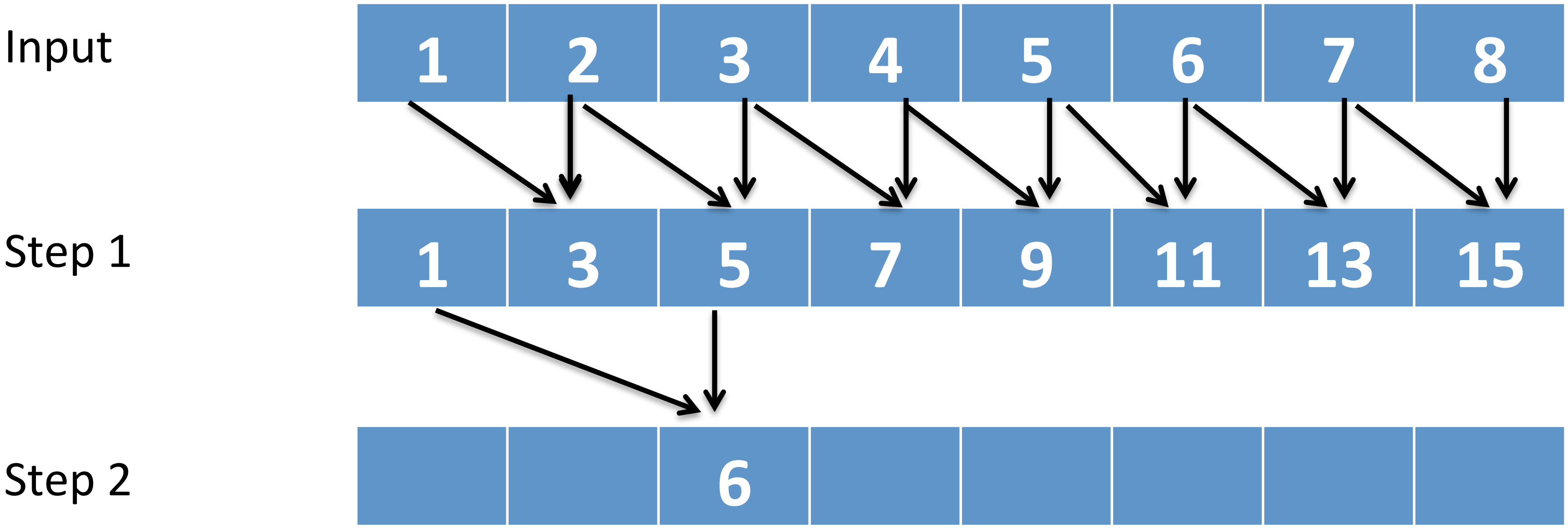
Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 1	1	3	5	7	9	11	13	15

If no neighbors then copy previous step

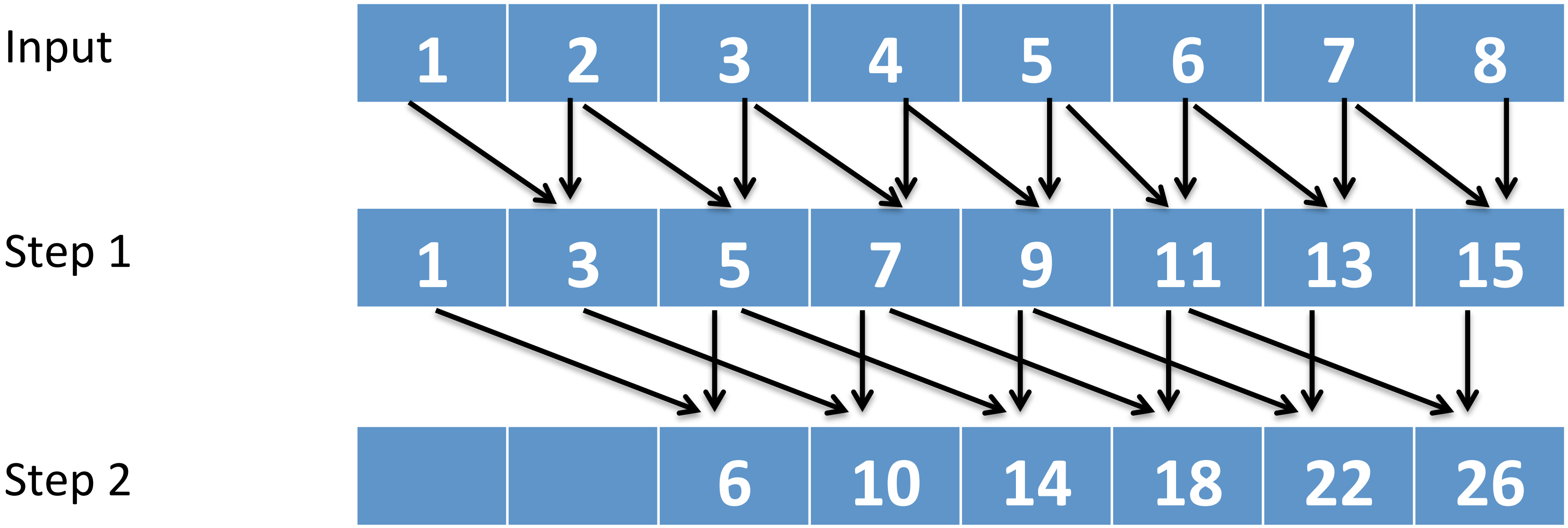
Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 1	1	3	5	7	9	11	13	15
Step 2			6					

Add neighbors two away n = 2

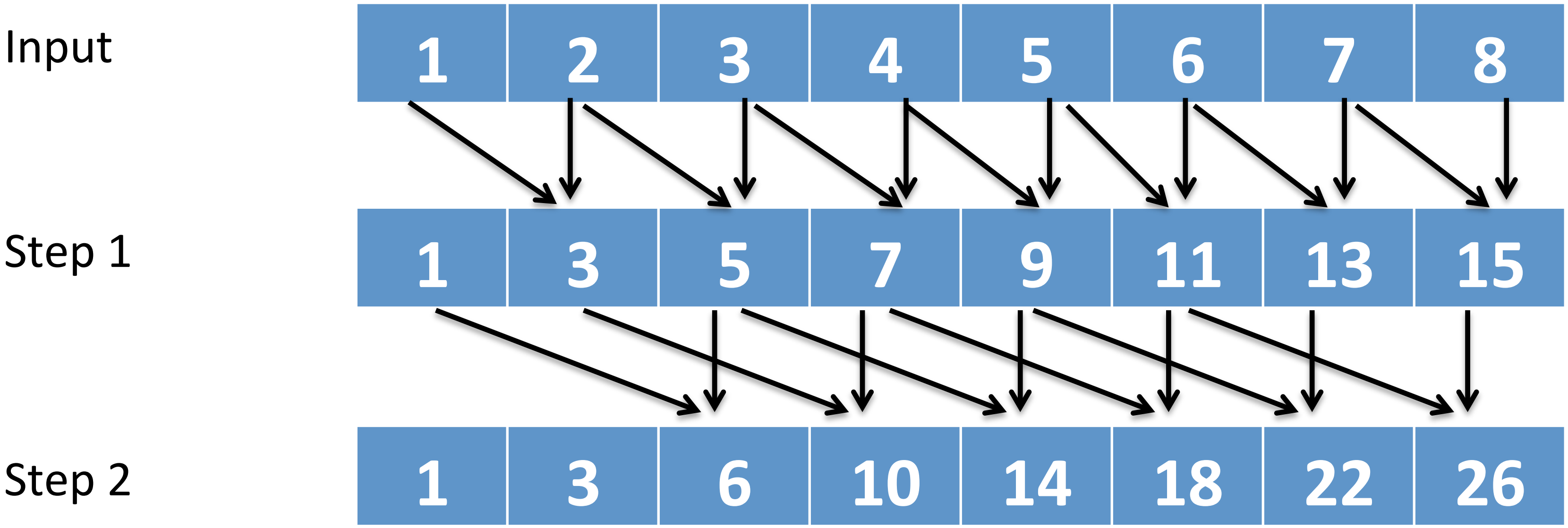
Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 1	1	3	5	7	9	11	13	15
Step 2			6	10	14	18	22	26

Add neighbors two away n = 2

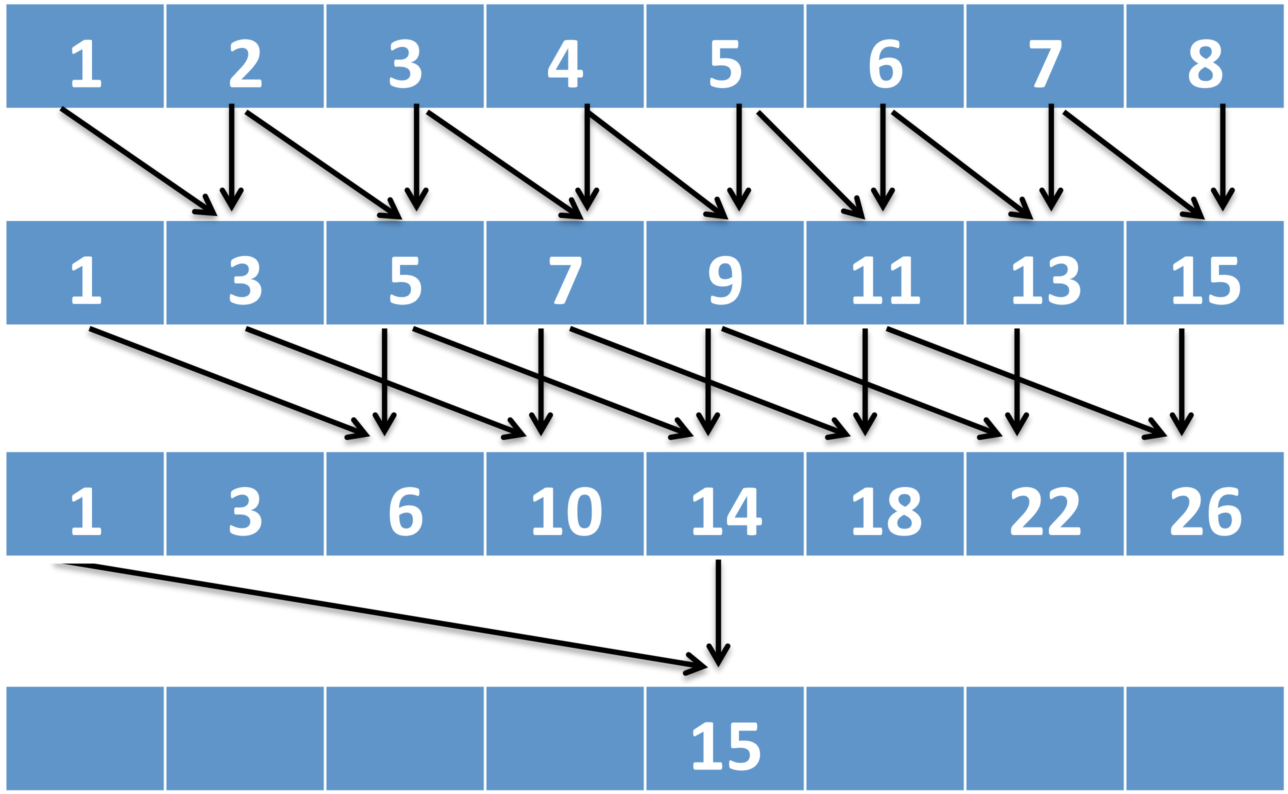
Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 1	1	3	5	7	9	11	13	15
Step 2	1	3	6	10	14	18	22	26

If no neighbors copy down

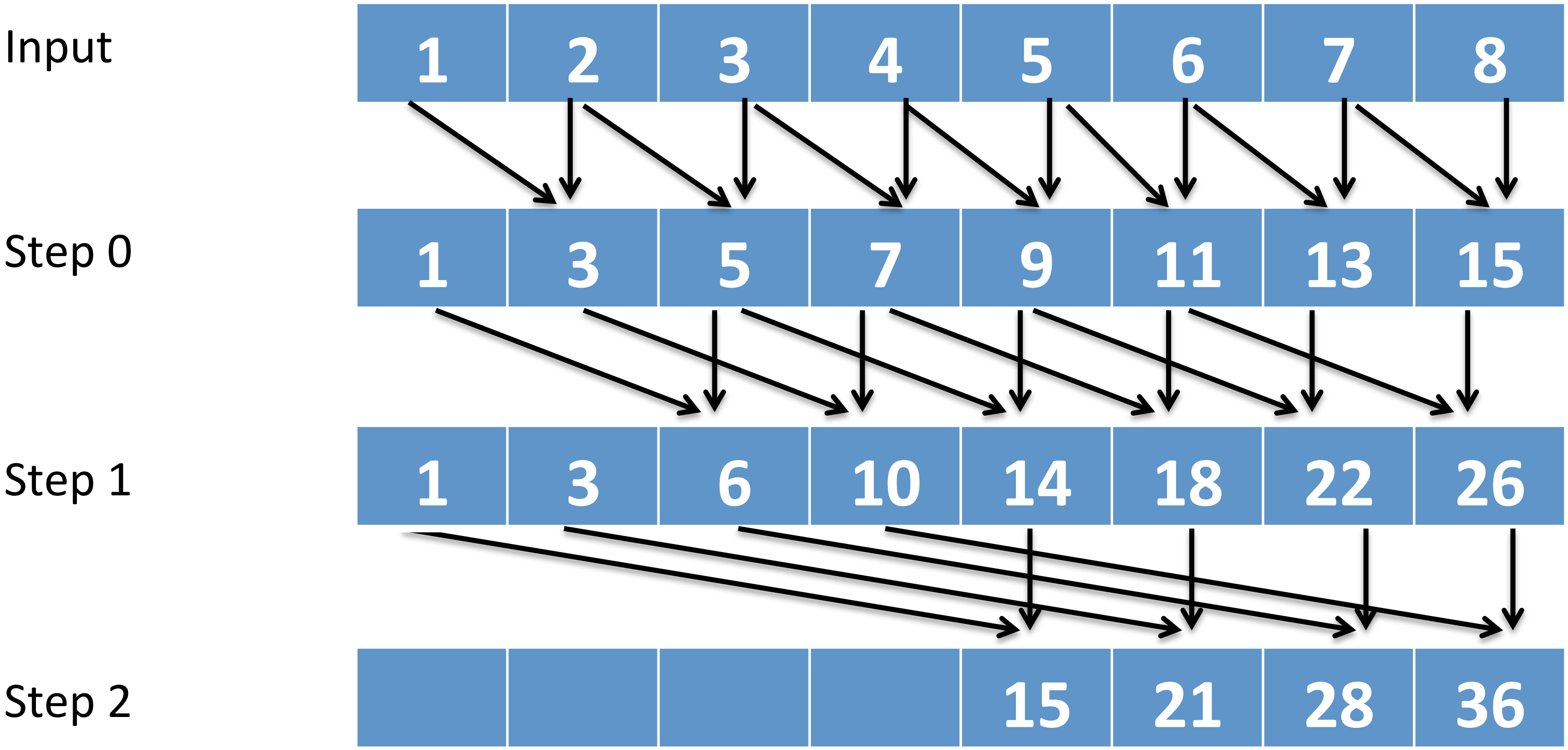
Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 0	1	3	5	7	9	11	13	15
Step 1	1	3	6	10	14	18	22	26
Step 2					15			

Add neighbors four away n = 4
Generally n = 2^{step}

Hillis/Steele Inclusive Scan

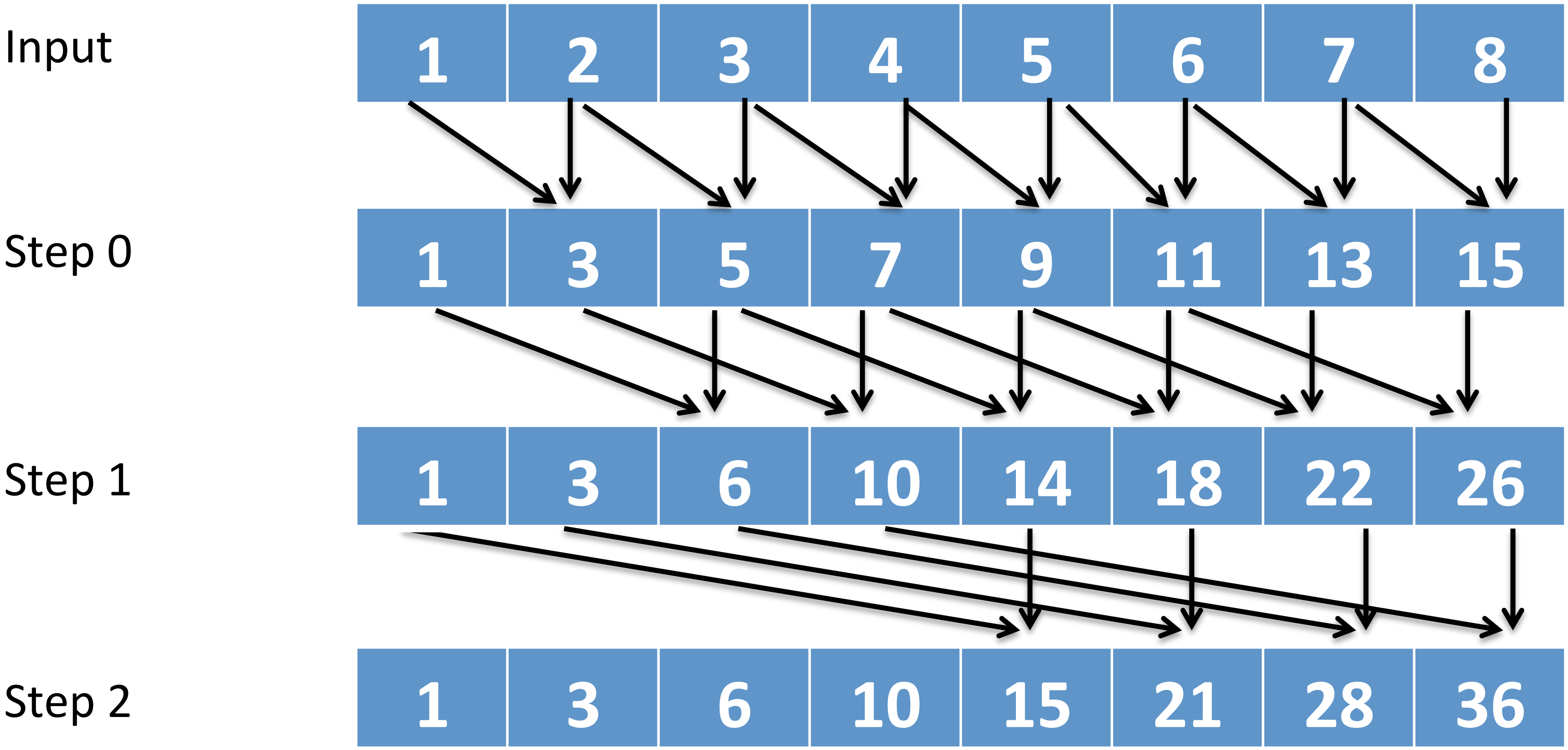


Input	1	2	3	4	5	6	7	8
Step 0	1	3	5	7	9	11	13	15
Step 1	1	3	6	10	14	18	22	26
Step 2					15	21	28	36

Add neighbors four away n = 4

Generally n = 2^{step}

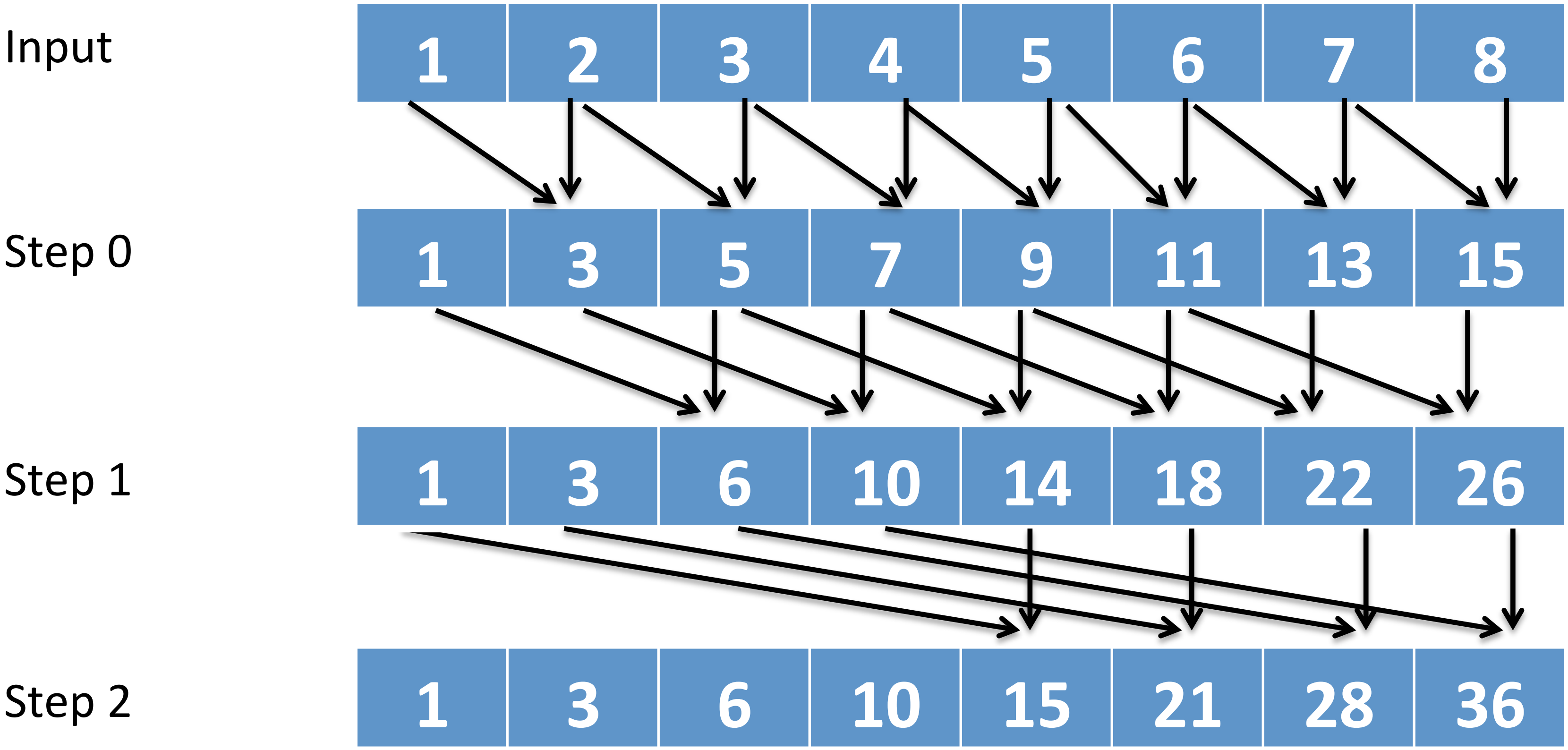
Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 0	1	3	5	7	9	11	13	15
Step 1	1	3	6	10	14	18	22	26
Step 2	1	3	6	10	15	21	28	36

Copy down others

Hillis/Steele Inclusive Scan



Input	1	2	3	4	5	6	7	8
Step 0	1	3	5	7	9	11	13	15
Step 1	1	3	6	10	14	18	22	26
Step 2	1	3	6	10	15	21	28	36

You now have the inclusive scan.

Steps = O(log n)
Work = O(n log n) <- dimensions of rectangle above

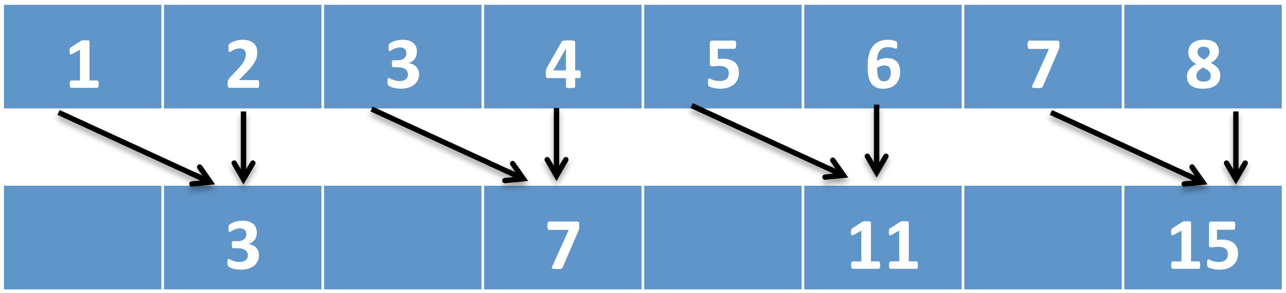
Blelloch Exclusive Scan

- Happens in two passes:
  - Reduce
    - Like previous reduce steps but keep around intermediate results
  - Down sweep
    - New operation

Blelloch Exclusive Scan

Input	1	2	3	4	5	6	7	8

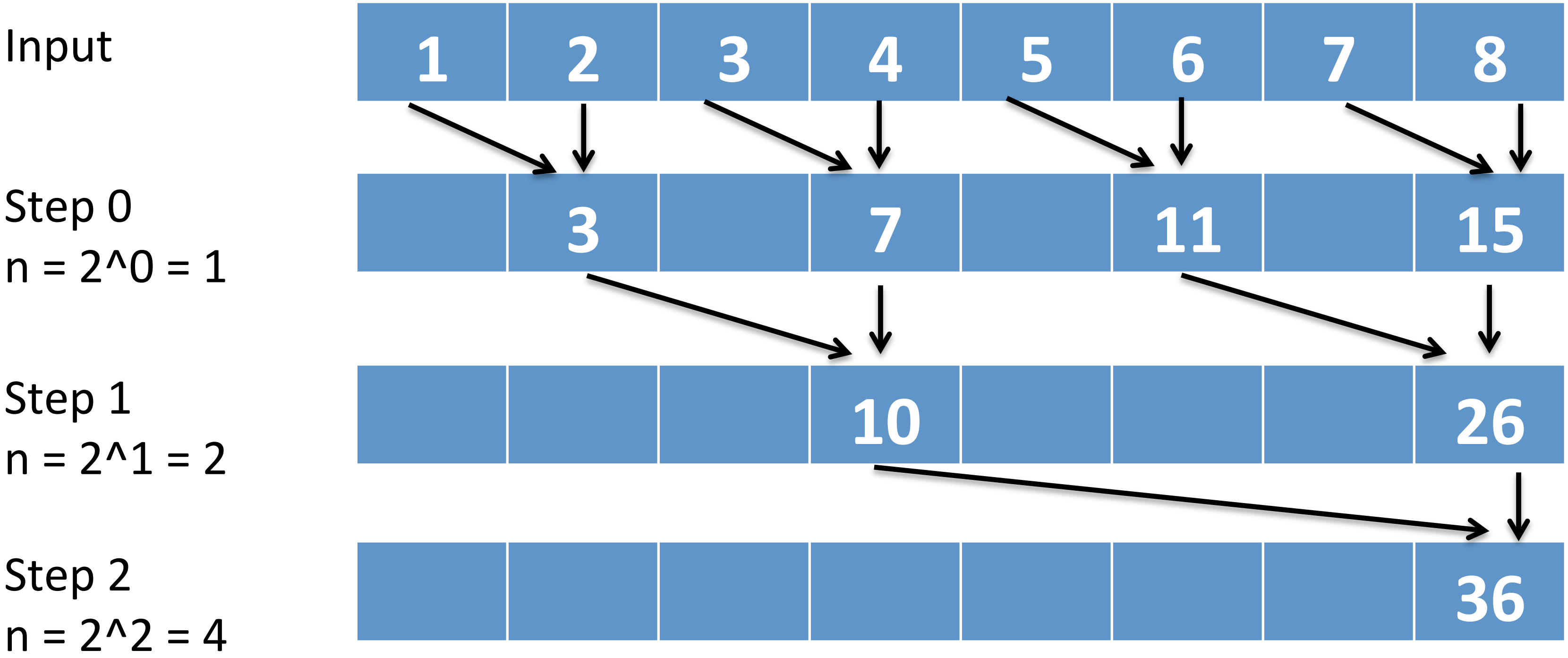
Blelloch Exclusive Scan



Input	1	2	3	4	5	6	7	8
Step 0 n = 2^0 = 1		3		7		11		15

Start a simple reduce

Blelloch Exclusive Scan

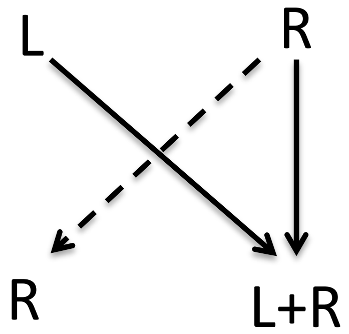


Input	1	2	3	4	5	6	7	8
Step 0 n = 2^0 = 1		3		7		11		15
Step 1 n = 2^1 = 2				10				26
Step 2 n = 2^2 = 4								36

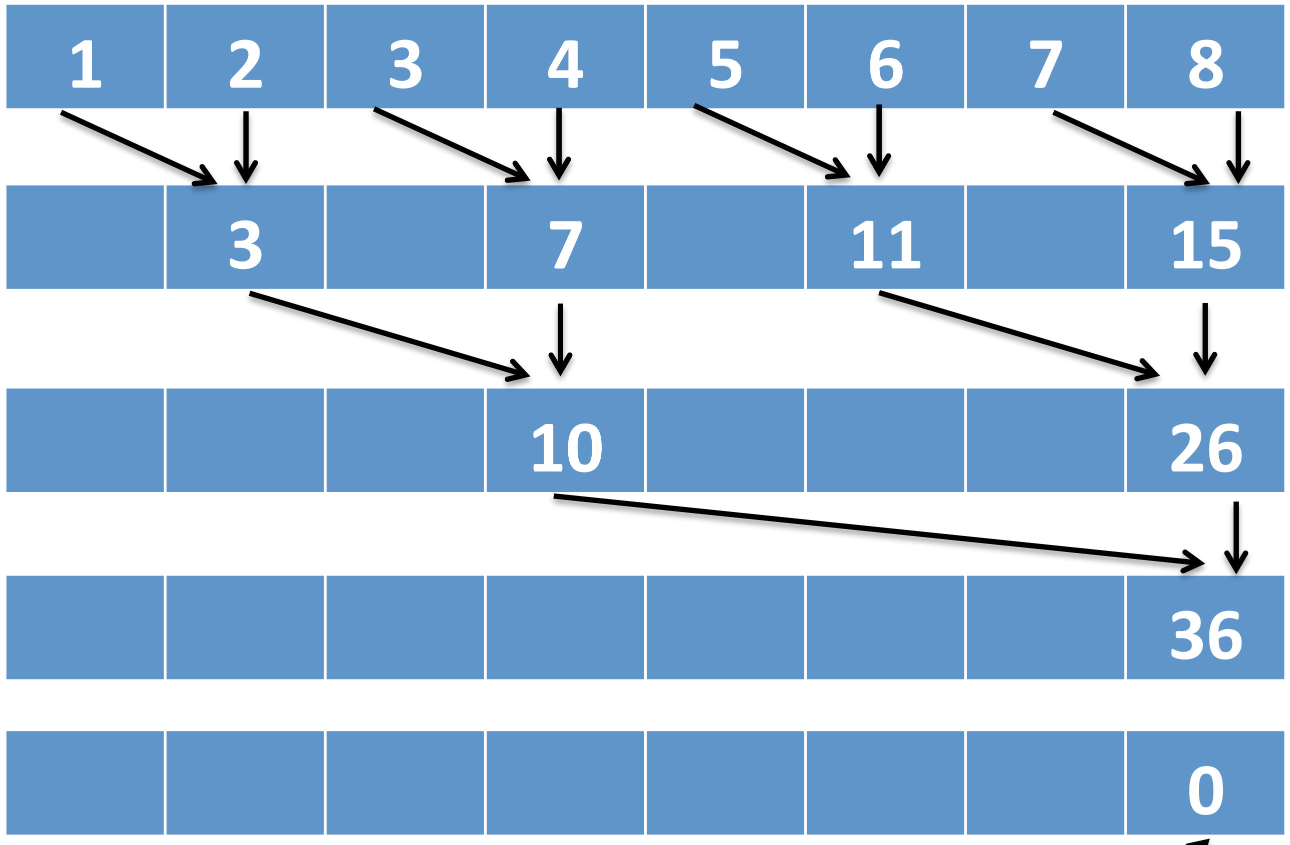
Similar to other reduces, however keep around the intermediate results: 3, 7, 11, 15, 10, 26

Blelloch Down Sweep Operation

- Reverse reduce step
  - Same inputs (left and right)
  - But two outputs also left and right
    - Add L+R and put on right
    - Copy down R and put it on left



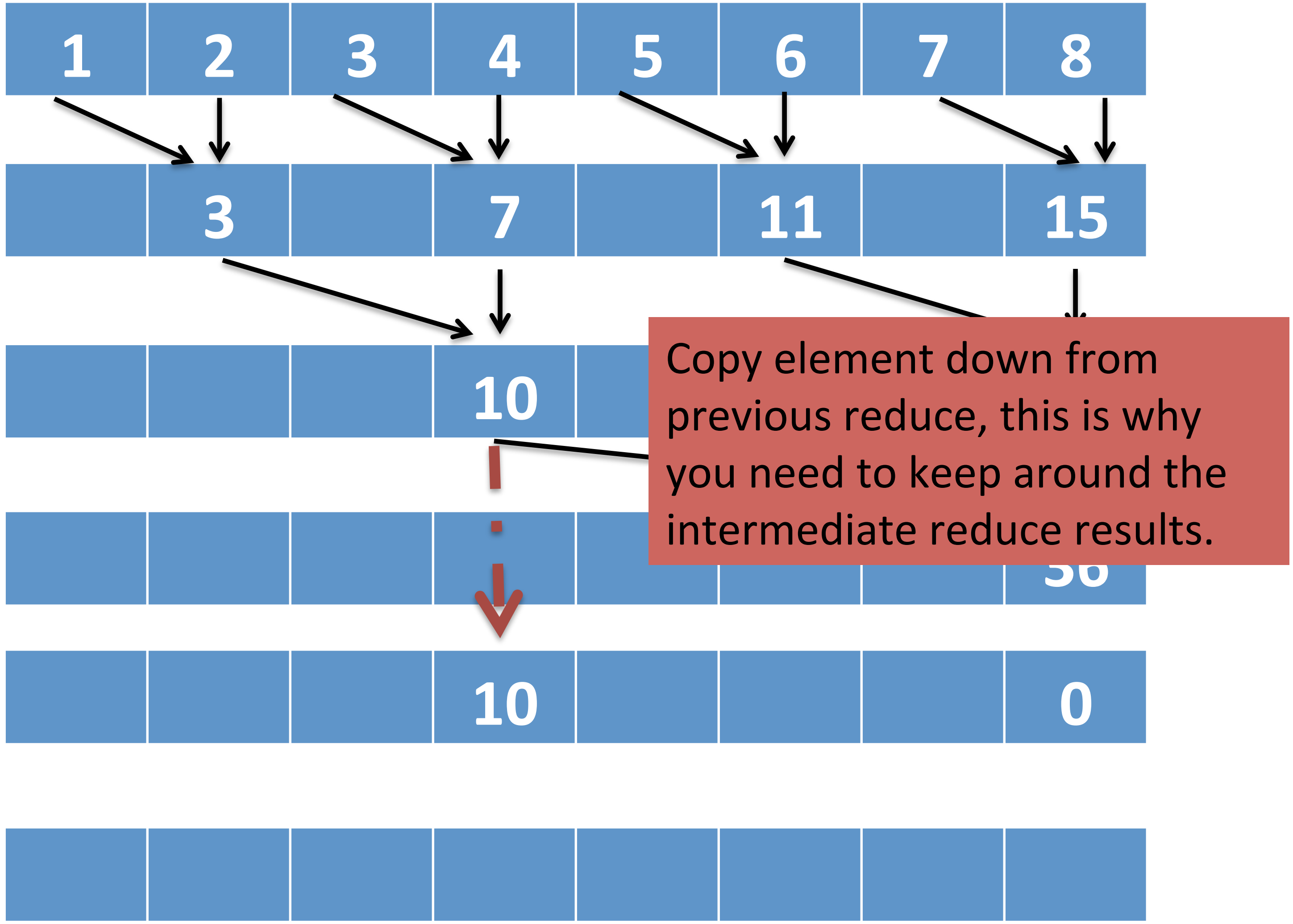
Blelloch Down Sweep



Input	1	2	3	4	5	6	7	8
Step 0 n = 2^0 = 1		3		7		11		15
Step 1 n = 2^1 = 2				10				26
Step 2 n = 2^2 = 4								36
Replace with Identity elem								0

First replace last column with identity element, for sum this is zero (0)

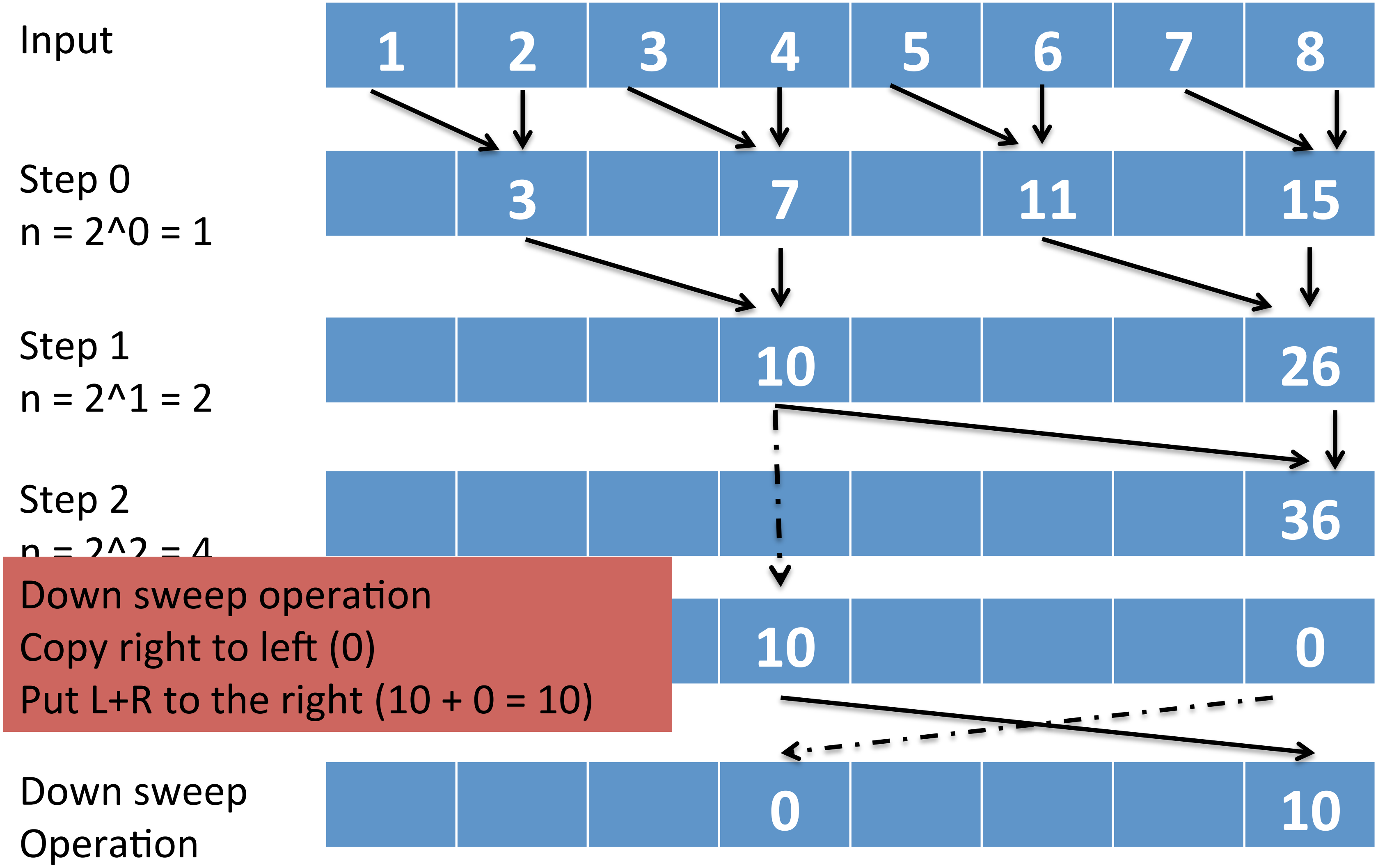
Blelloch Down Sweep



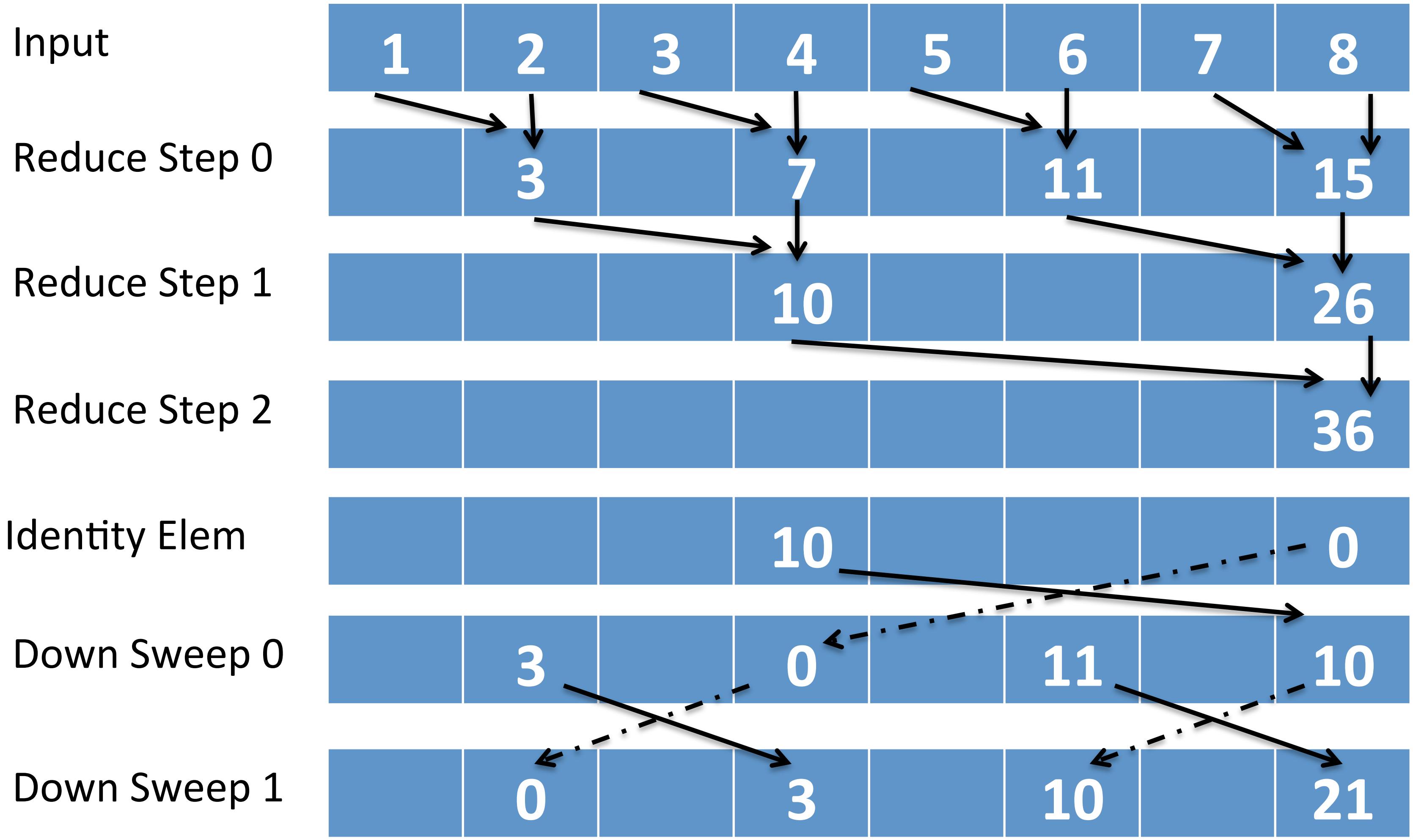
Input	1	2	3	4	5	6	7	8
Step 0 n = 2^0 = 1		3		7		11		15
Step 1 n = 2^1 = 2				10				
Step 2 n = 2^2 = 4								36
Replace with Identity elem				10				0
Down sweep Operation								

Copy element down from previous reduce, this is why you need to keep around the intermediate reduce results.

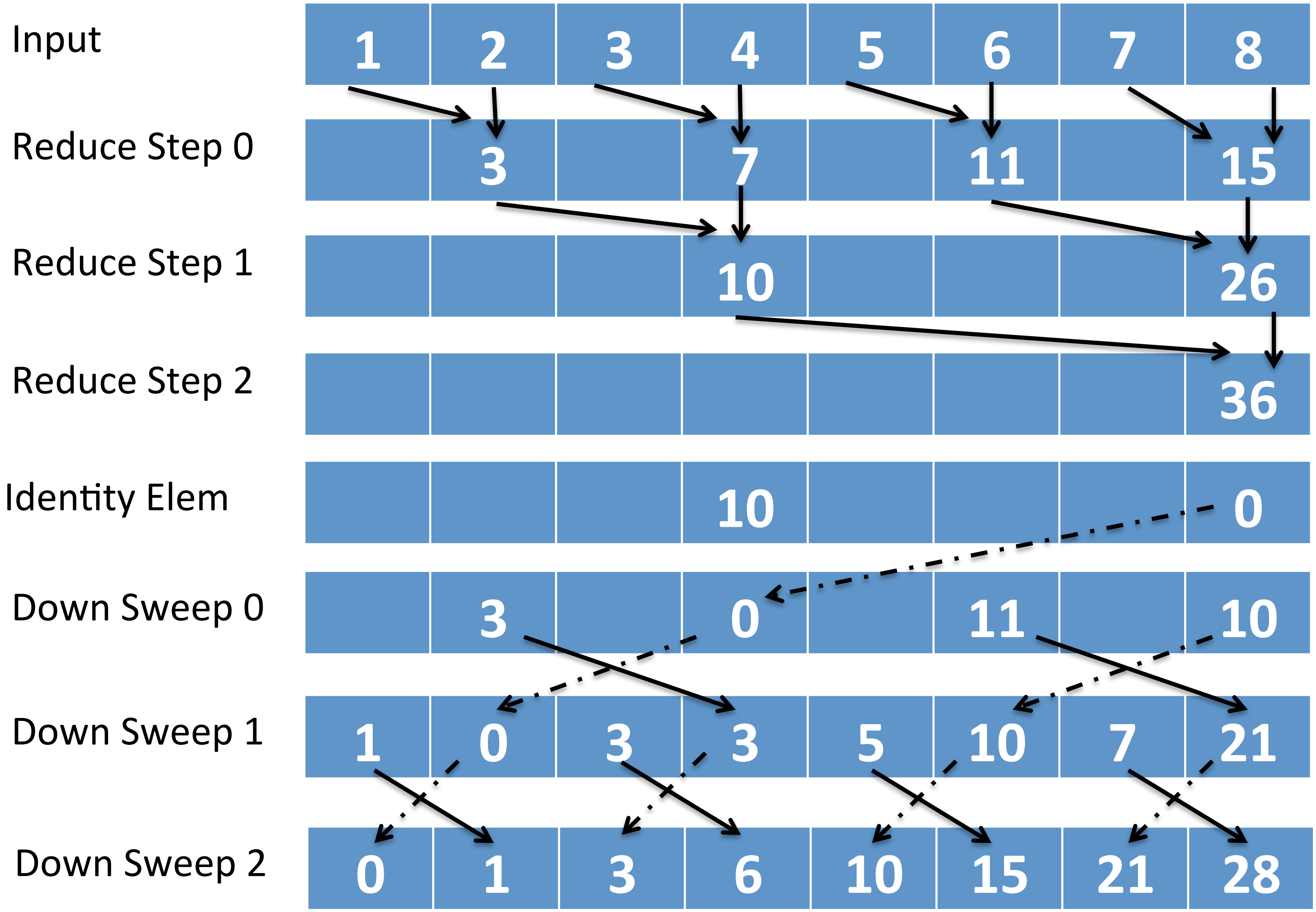
Blelloch Down Sweep



Down sweep operation
Copy right to left (0)
Put L+R to the right (10 + 0 = 10)



Input	1	2	3	4	5	6	7	8
Reduce Step 0		3		7		11		15
Reduce Step 1				10				26
Reduce Step 2								36
Identity Elem				10				0
Down Sweep 0		3		0		11		10
Down Sweep 1		0		3		10		21



Input	1	2	3	4	5	6	7	8
Reduce Step 0		3		7		11		15
Reduce Step 1				10				26
Reduce Step 2								36
Identity Elem				10				0
Down Sweep 0		3		0		11		10
Down Sweep 1	1	0	3	3	5	10	7	21
Down Sweep 2	0	1	3	6	10	15	21	28

Blelloch Exclusive Scan

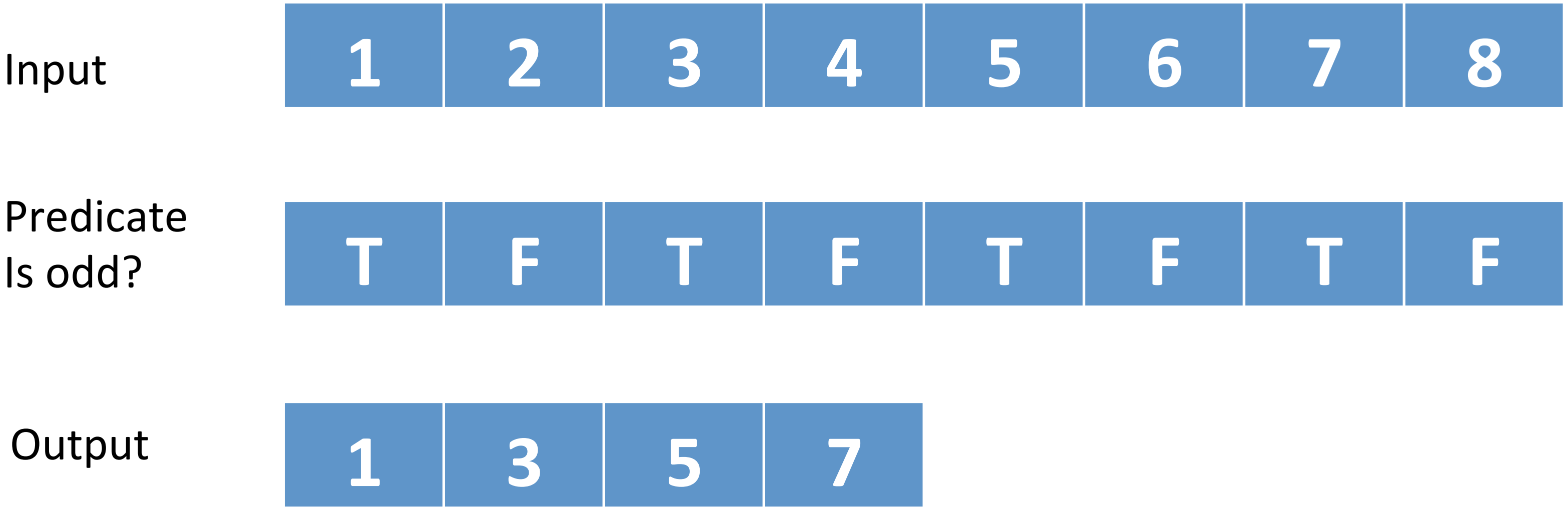
- # Steps?
  - 2 log n = O(log n)
- Work?
  - O(n)

Compact

- Many times you lots of data and you only want to perform some computation on a subset of that data.
  - Logs analysis: only look at logs containing a certain search term or type of search term
  - Graphics: only perform ray tracing on elements in the viewport
  - Big Data: Calculate histogram of incomes for everyone with a dog

What is compact

- Given some predicate function remove those elements which return false and “squeeze” the data into the required space.



Input	1	2	3	4	5	6	7	8
Predicate Is odd?	T	F	T	F	T	F	T	F
Output	1	3	5	7				

Parallel Compact

- Just have each thread evaluate predicate and copy only on true.

Input	1	2	3	4	5	6	7	8
Predicate Is odd?	T	F	T	F	T	F	T	F
Sparse Output	1	-	3	-	5	-	7	-
Dense Output	1	3	5	7				

Sparse is easy in parallel, how do we get dense output in parallel?

Parallel Compact

- Just have each thread evaluate predicate and copy only on true.

Input	1	2	3	4	5	6	7	8
Predicate Is odd?	T	F	T	F	T	F	T	F
Sparse Output	1	-	3	-	5	-	7	-
Dense Output	1	3	5	7				

Sparse is easy in parallel, how do we get dense output in parallel?

Parallel Compact

Input	1	2	3	4	5	6	7	8
Predicate Is odd?	T	F	T	F	T	F	T	F
Sparse Output	1	-	3	-	5	-	7	-
Address in dense output	0	-	1	-	2	-	3	-
Dense Output	1	3	5	7				

Look at the desired address locations in the dense output for each for each element in the sparse output

Parallel Compact

Input	1	2	3	4	5	6	7	8
Predicate Is odd?	T	F	T	F	T	F	T	F
True/False = 1/0	1	0	1	0	1	0	1	0
Exclusive Scan on 1/0	0	1	1	2	2	3	3	4
Address in dense output	0	-	1	-	2	-	3	-
Dense Output	1	3	5	7				

To get addresses for dense output run a SCAN ! !

Parallel Compact Steps

1. Run Predicate
2. Create a scan-in array
   - True = 1
   - False = 0
3. Run exclusive scan over scan-in array
   - Output is the scatter addresses for input
4. Scatter the input into output addresses