UCIrvine | The Henry Samueli School of Engineering

Parallel Algorithms

EECS 221: Intro to High-Performance Computing

Aparna Chandramowlishwaran

April 20, 2017

Inclusive Prefix-Sums (Scan)

- The prefix-sums operation takes a binary associative operator ⊕, and an array of *n* elements

  $$[x_0, x_1, \ldots, x_{n-1}]$$

  and returns the array

  $$[x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1})]$$

- Example: If ⊕ is addition

3	1	7	0	4	1	6	3

Inclusive Prefix-Sums (Scan)

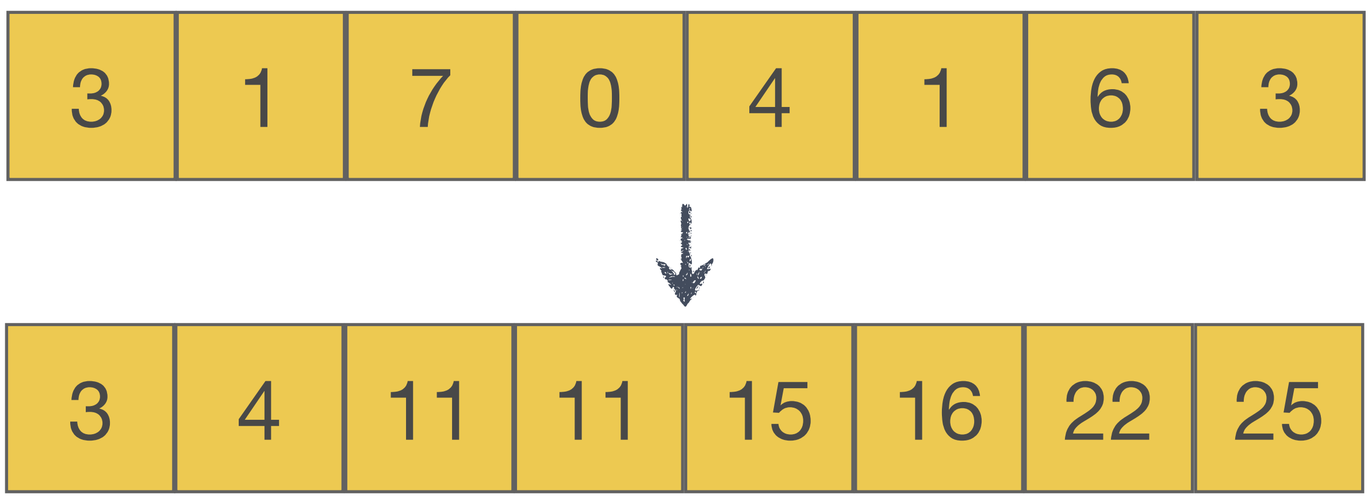
- The prefix-sums operation takes a binary associative operator ⊕, and an array of *n* elements

  $$[x_0, x_1, \ldots, x_{n-1}]$$

  and returns the array

  $$[x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1})]$$

- Example: If ⊕ is addition



Exclusive Prefix-Sums (Scan)

- The prefix-sums operation takes a binary associative operator $\oplus$, and an array of *n* elements

$$[x_0, x_1, \ldots, x_{n-1}]$$

and returns the array

$$[0, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-2})]$$

- Example: If $\oplus$ is addition

3	1	7	0	4	1	6	3

Exclusive Prefix-Sums (Scan)

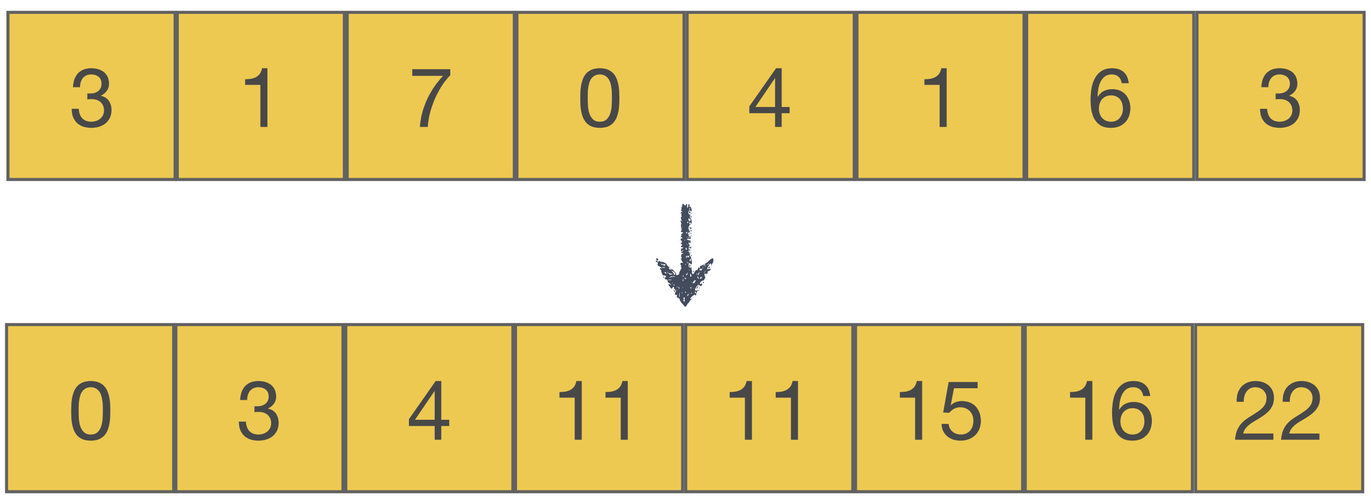
- The prefix-sums operation takes a binary associative operator $\oplus$, and an array of *n* elements

$$[x_0, x_1, \ldots, x_{n-1}]$$

and returns the array

$$[0, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-2})]$$

- Example: If $\oplus$ is addition



Work Efficient Sequential Scan

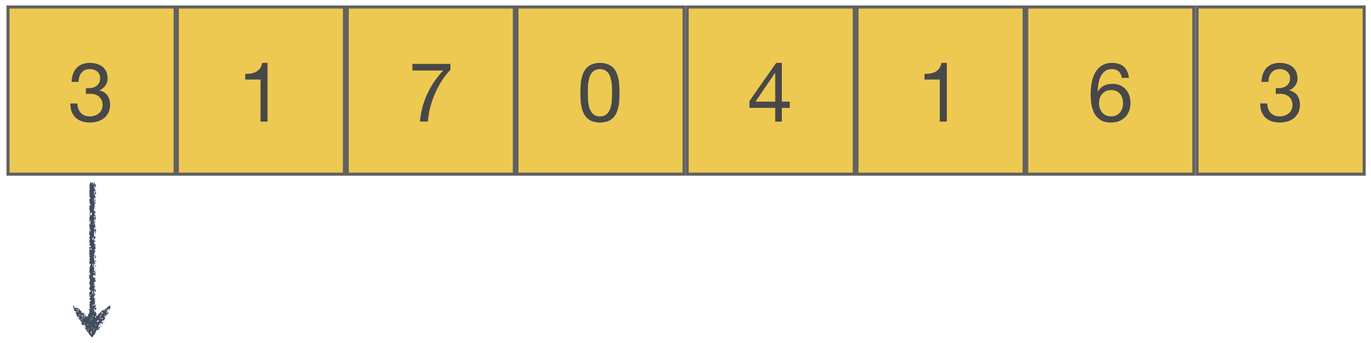
```
for i ← 1 to n − 1 do
    A[i] ← A[i − 1] + A[i]
end for
```

$$W_{scan}(n) = \Theta(n)$$

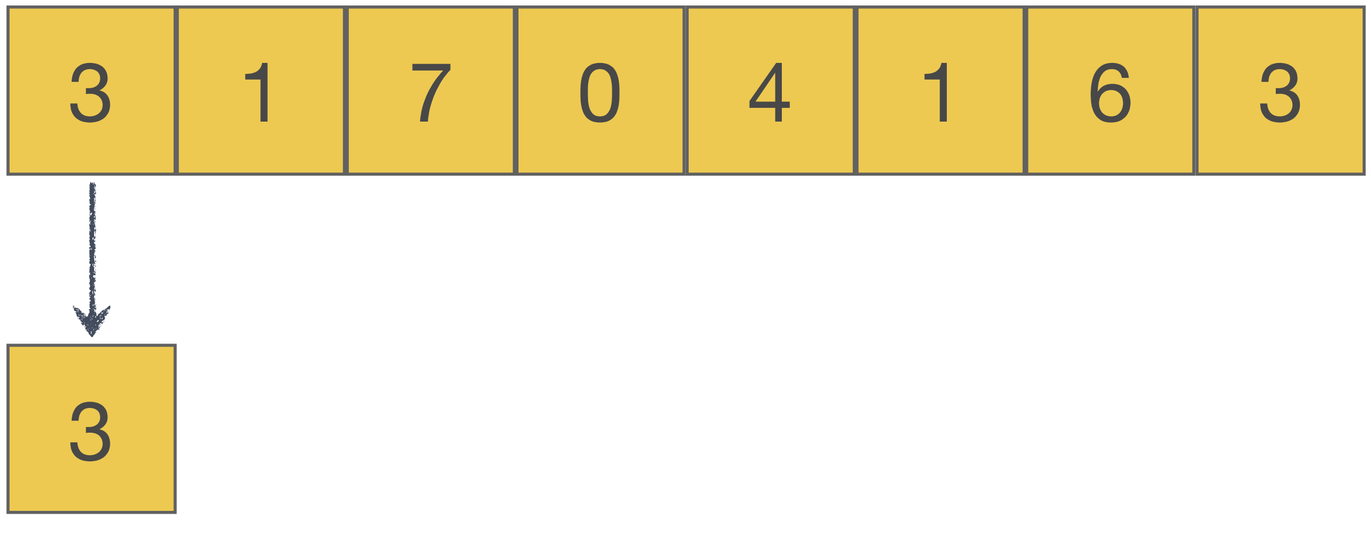
$$D_{scan}(n) = \Theta(n)$$

Inclusive Parallel Scan

Inclusive Parallel Scan



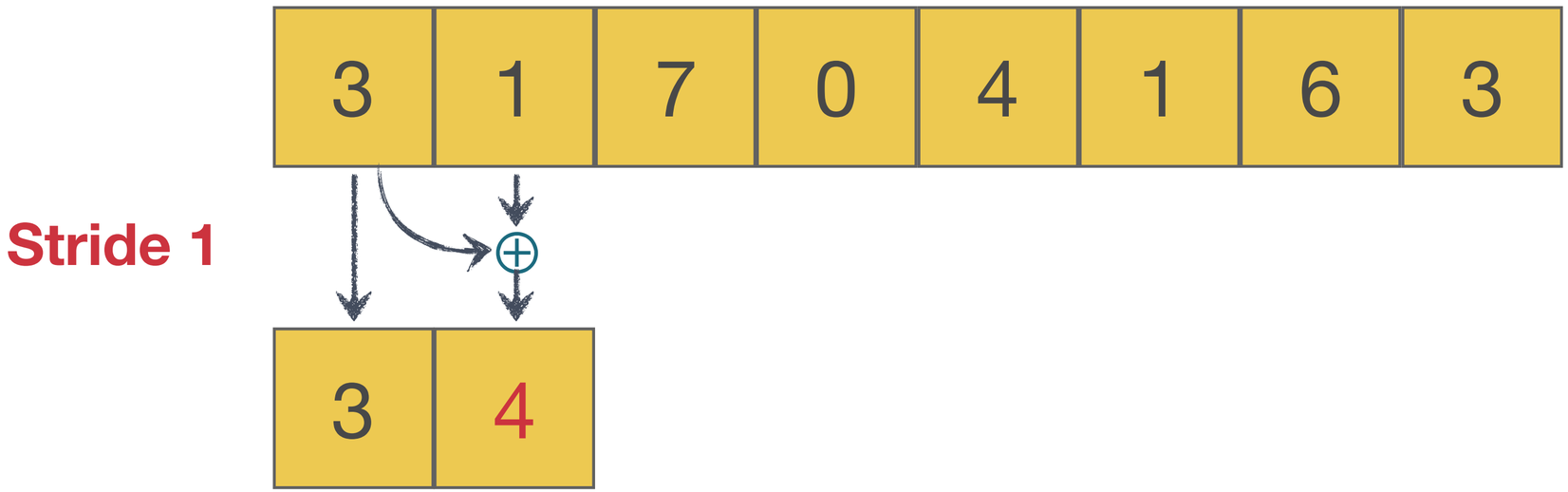
Inclusive Parallel Scan



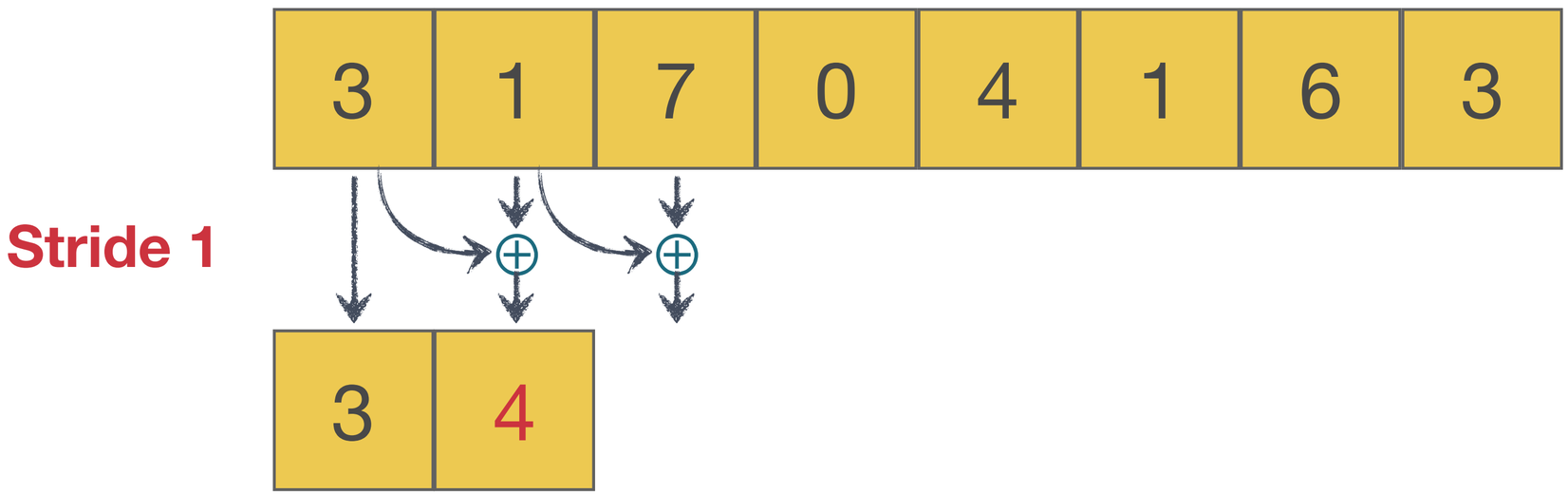
Inclusive Parallel Scan

Inclusive Parallel Scan

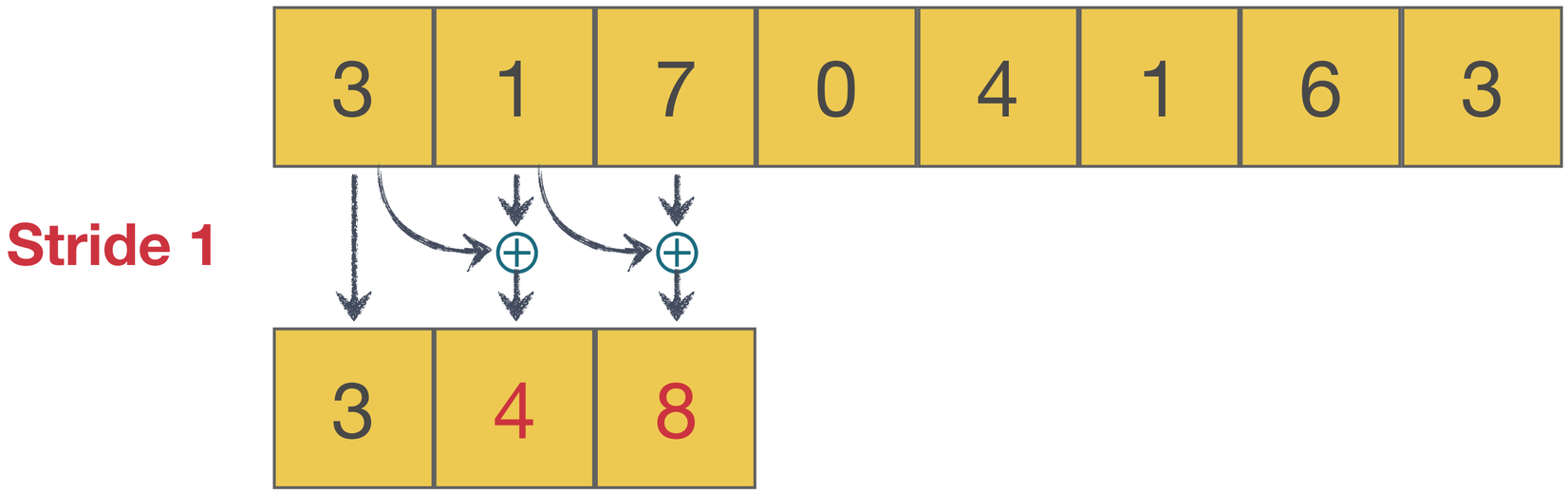
Inclusive Parallel Scan



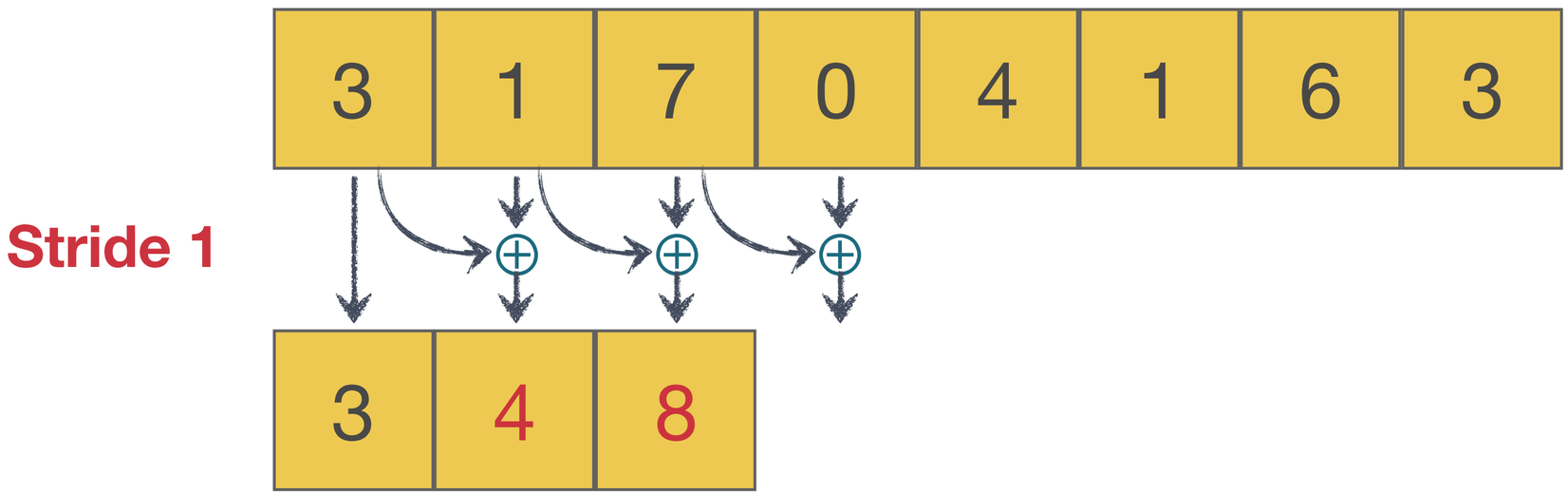
Inclusive Parallel Scan



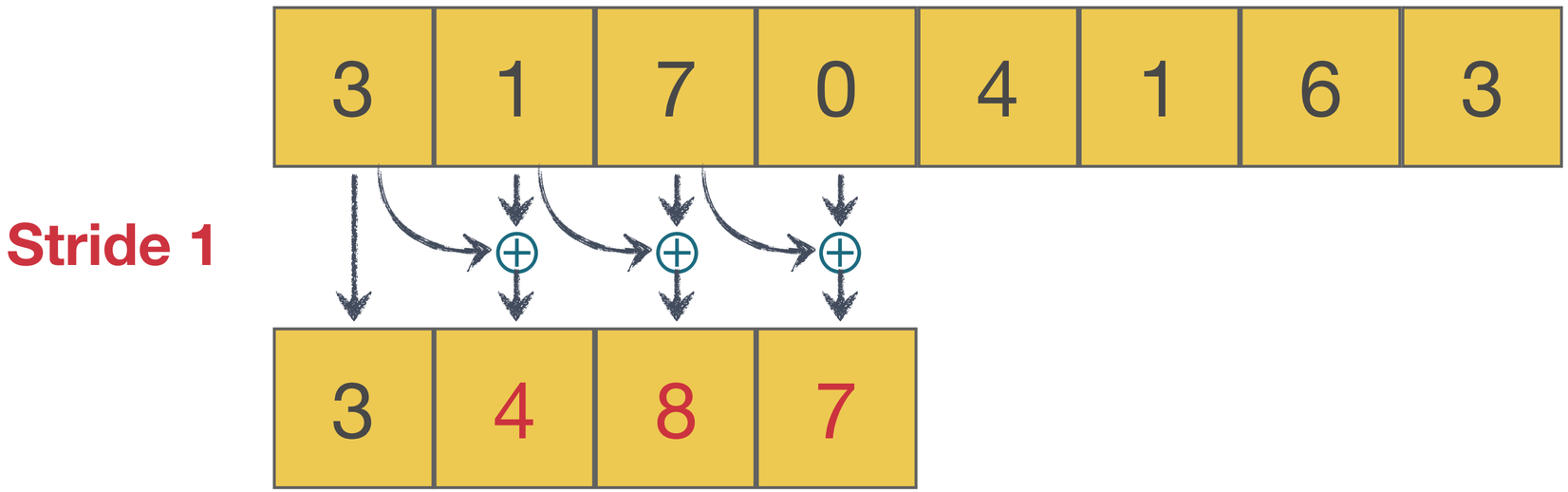
Inclusive Parallel Scan



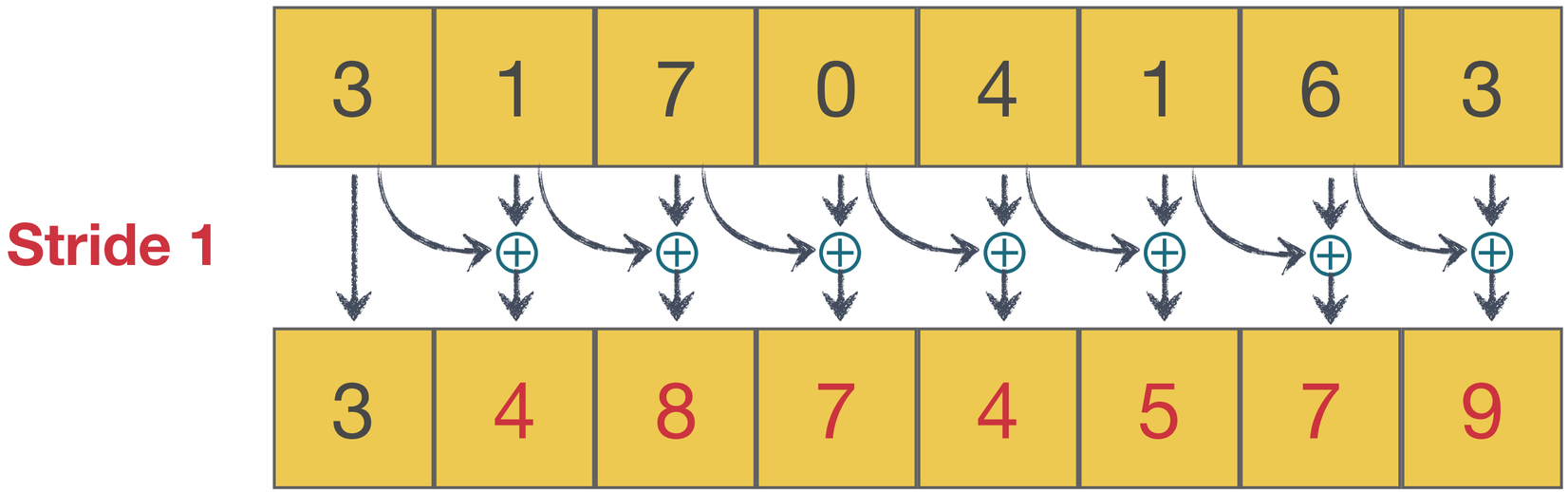
Inclusive Parallel Scan



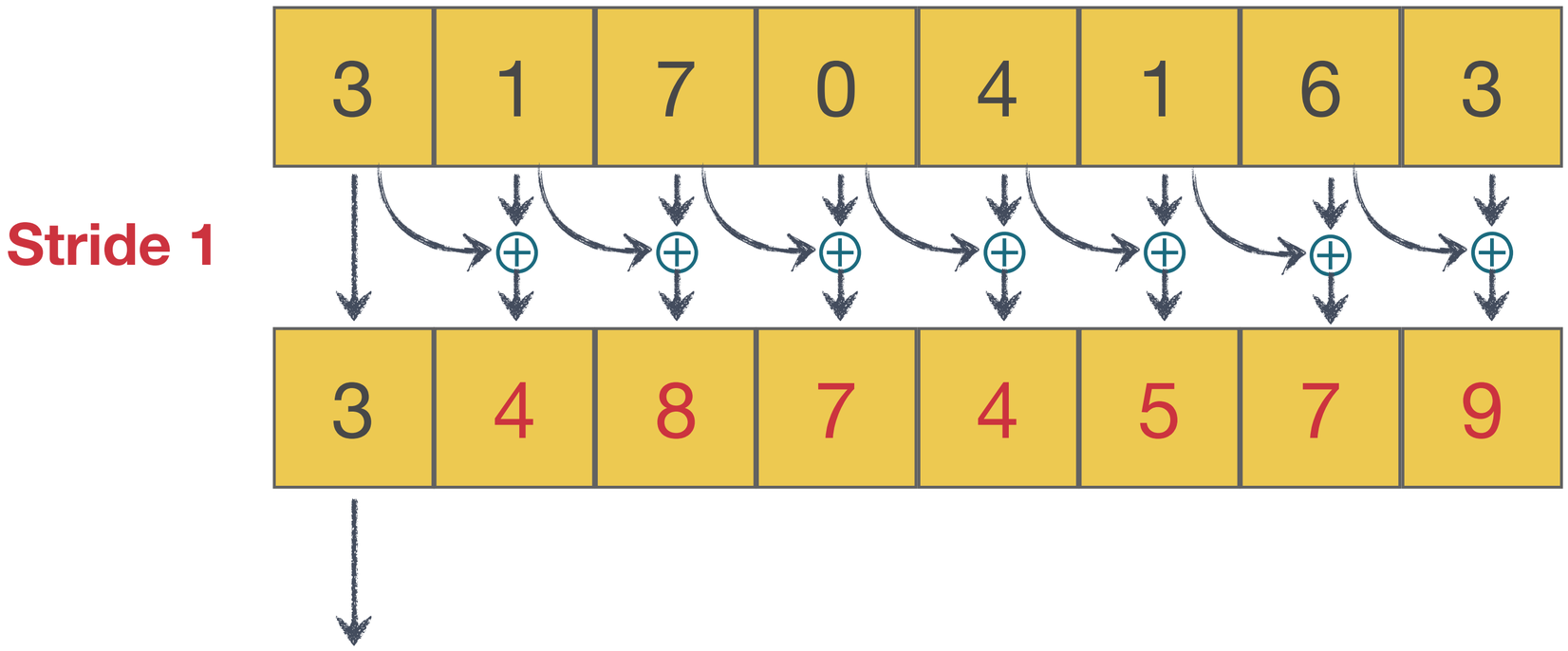
Inclusive Parallel Scan



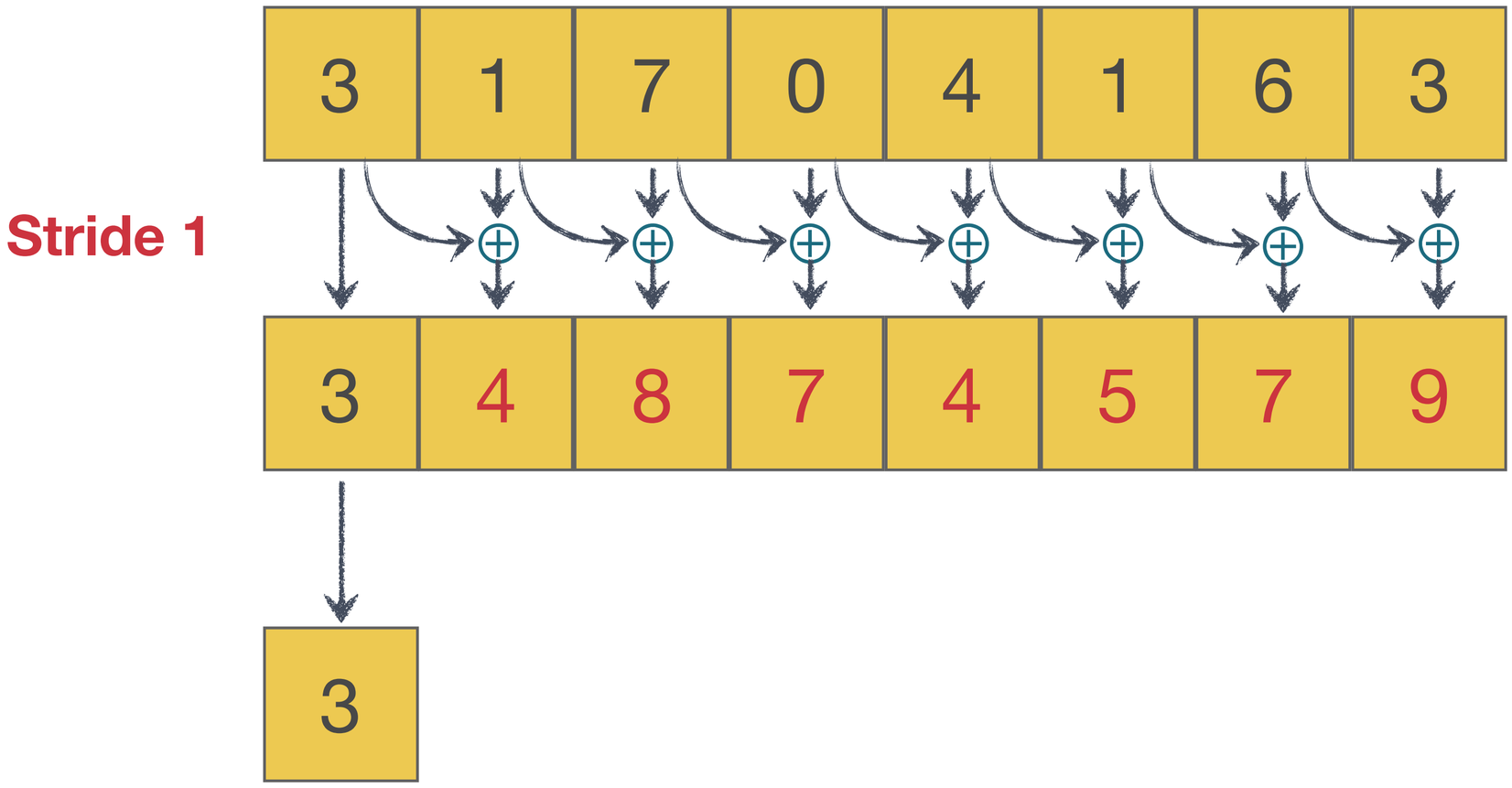
Inclusive Parallel Scan



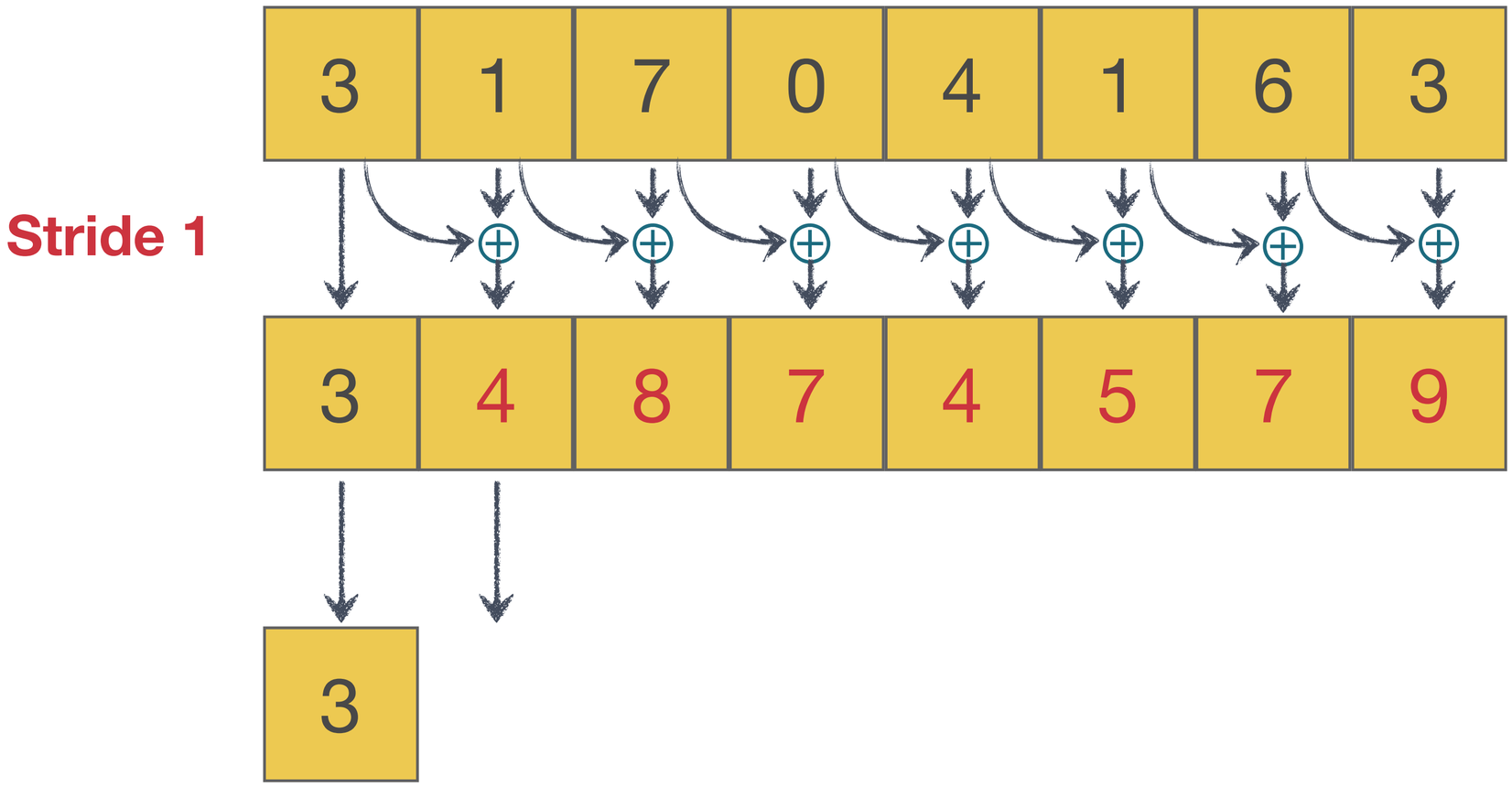
Inclusive Parallel Scan



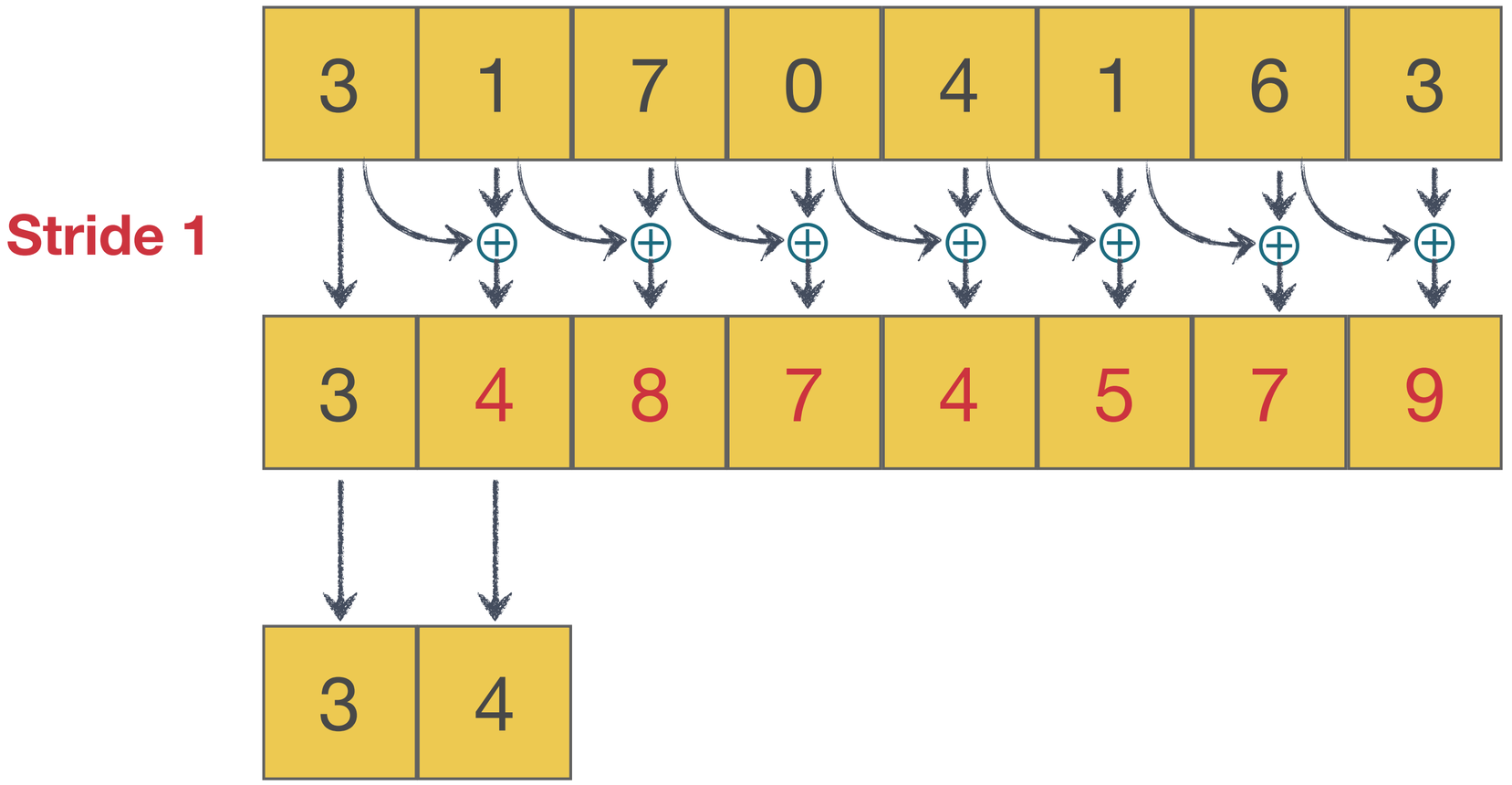
Inclusive Parallel Scan



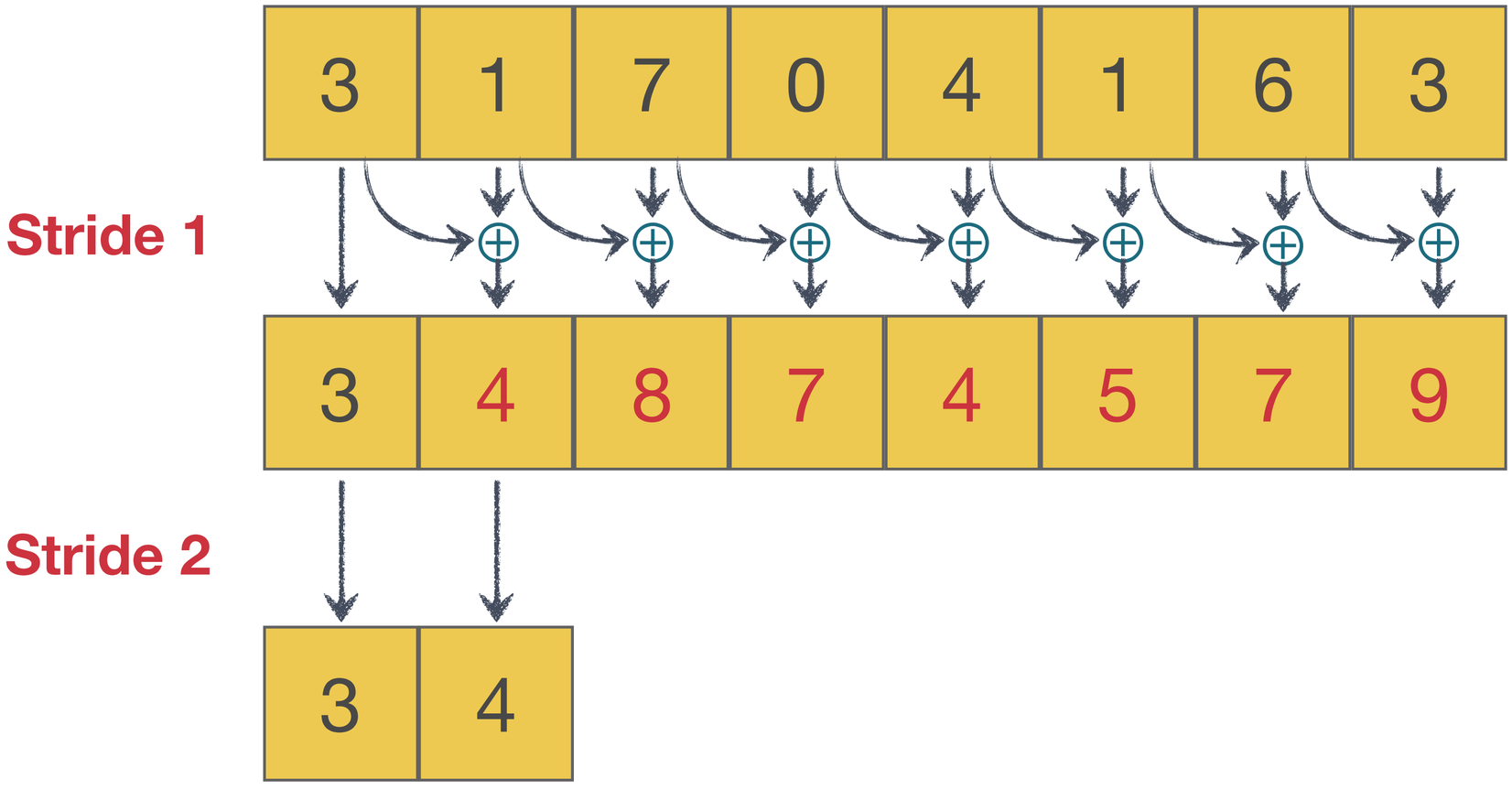
Inclusive Parallel Scan



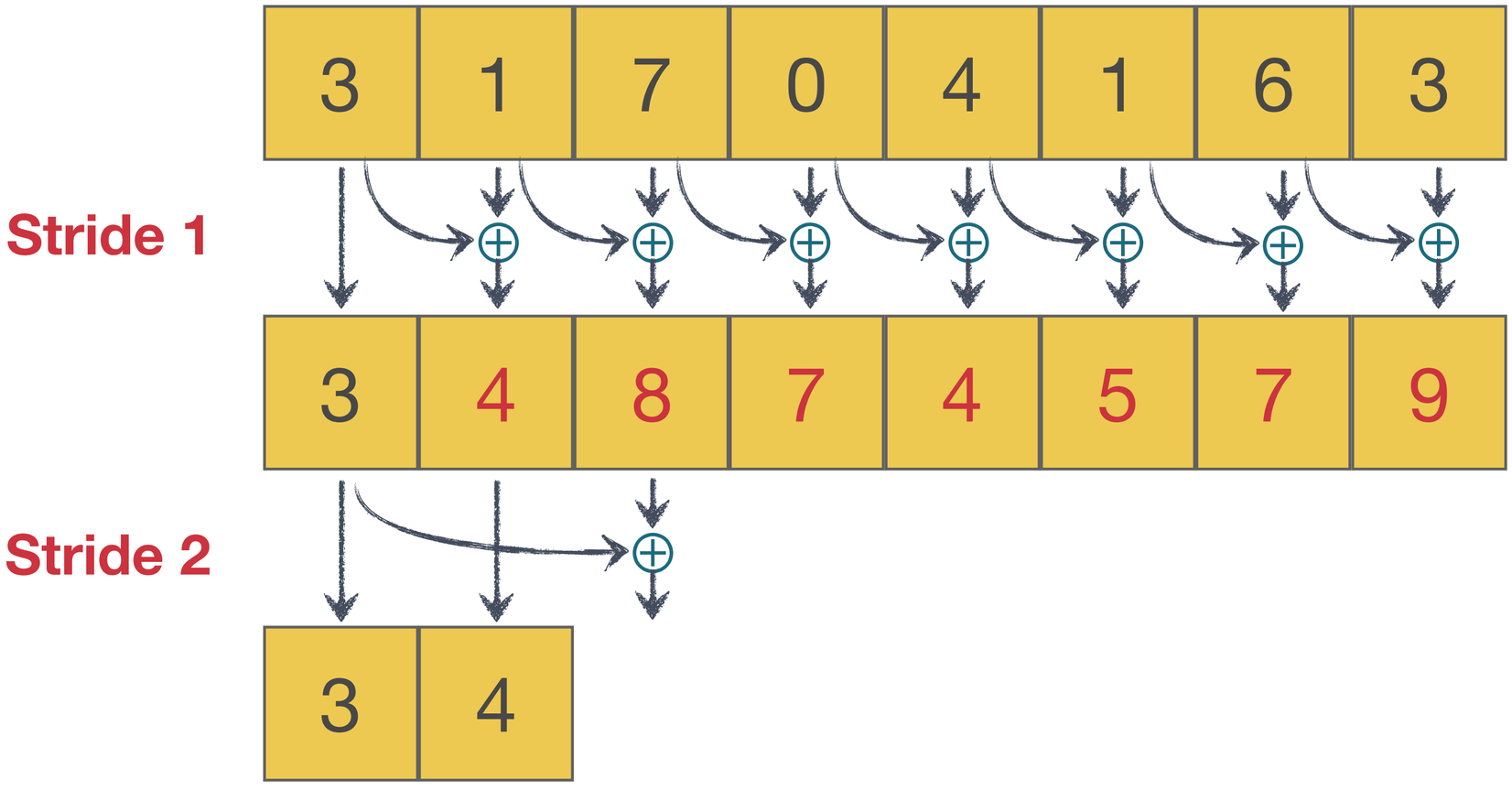
Inclusive Parallel Scan



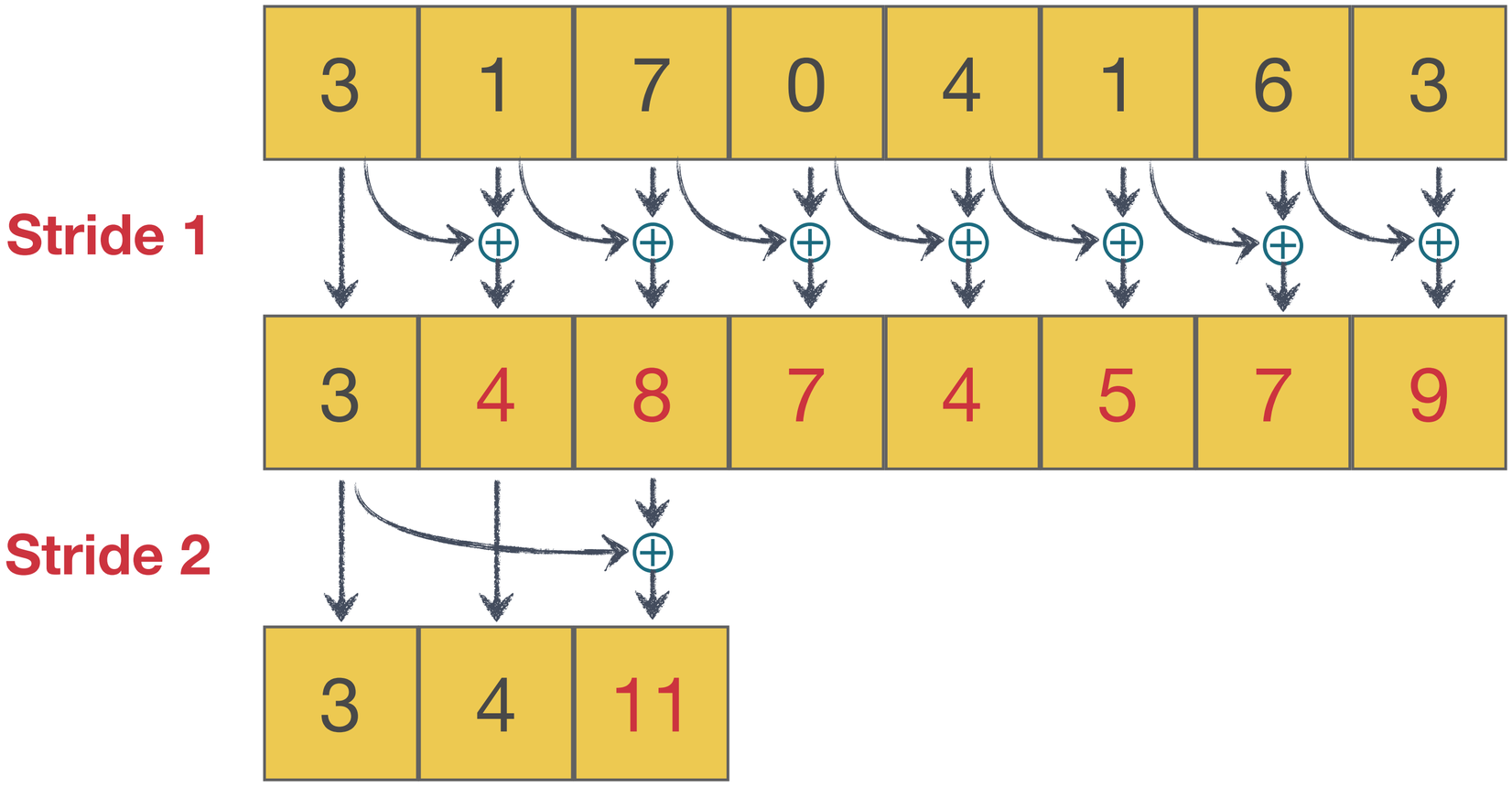
Inclusive Parallel Scan



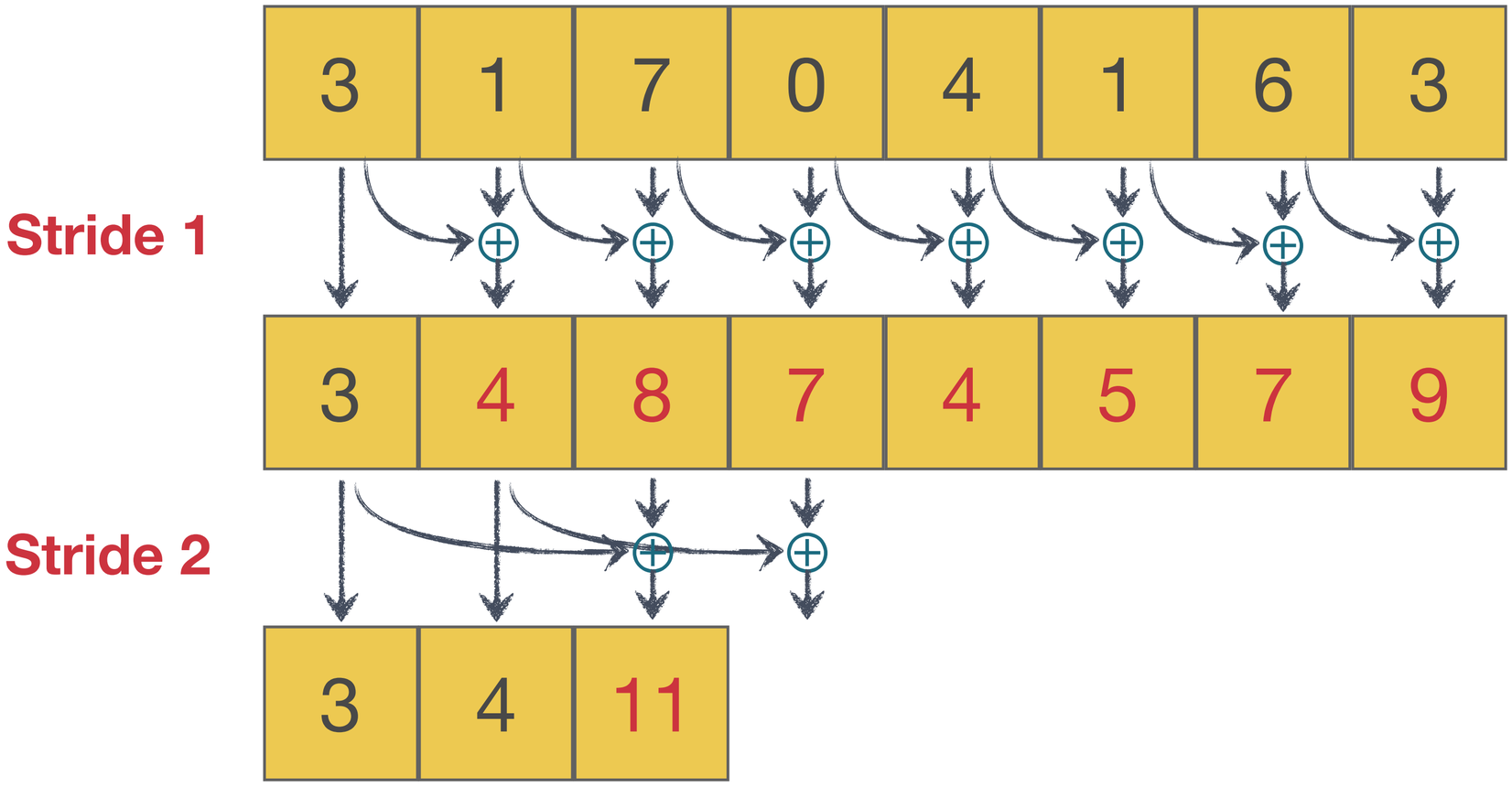
Inclusive Parallel Scan



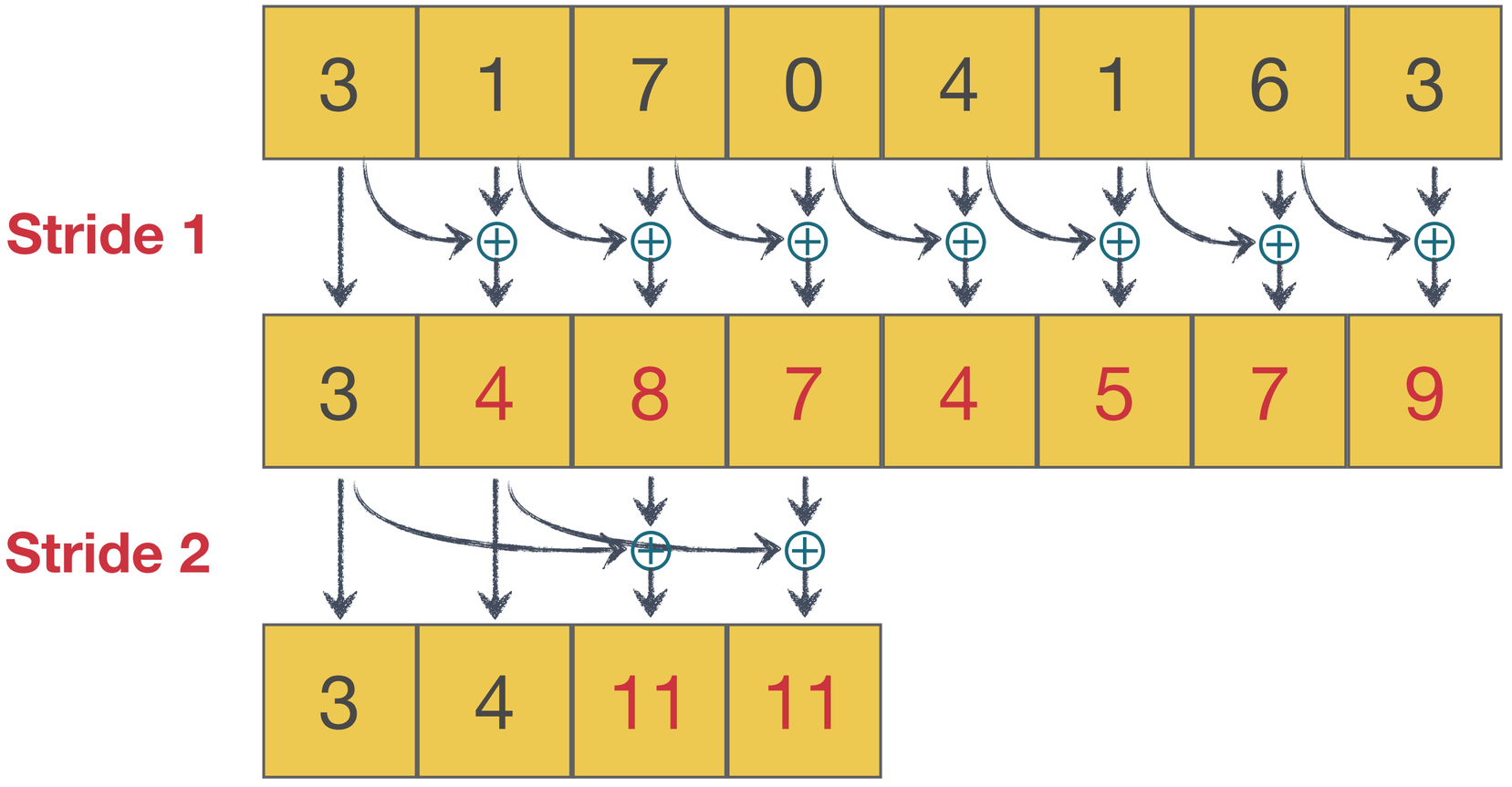
Inclusive Parallel Scan



Inclusive Parallel Scan

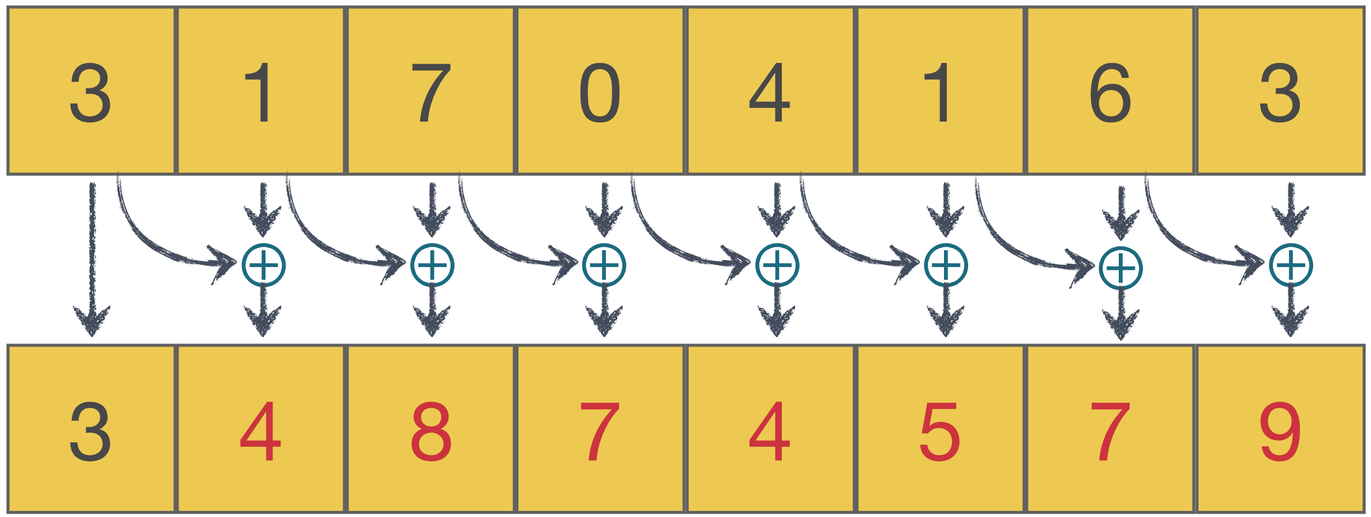


Inclusive Parallel Scan

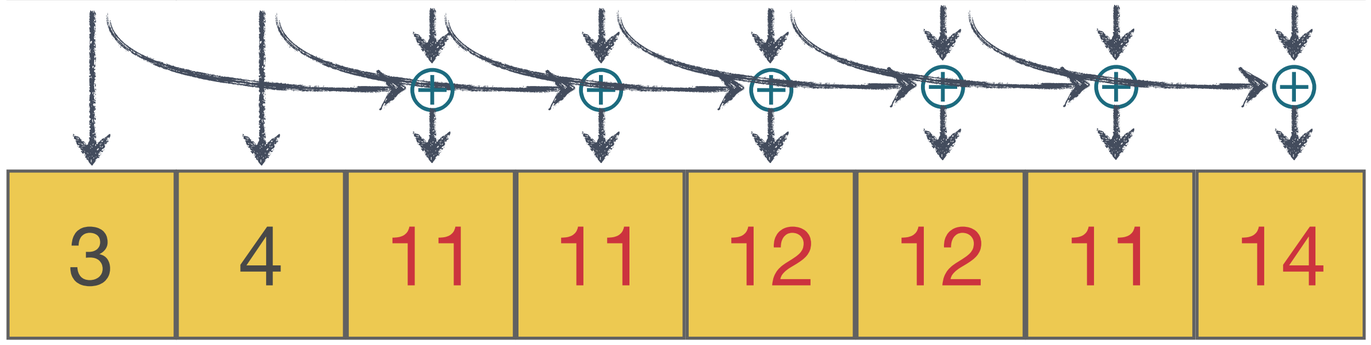


Inclusive Parallel Scan

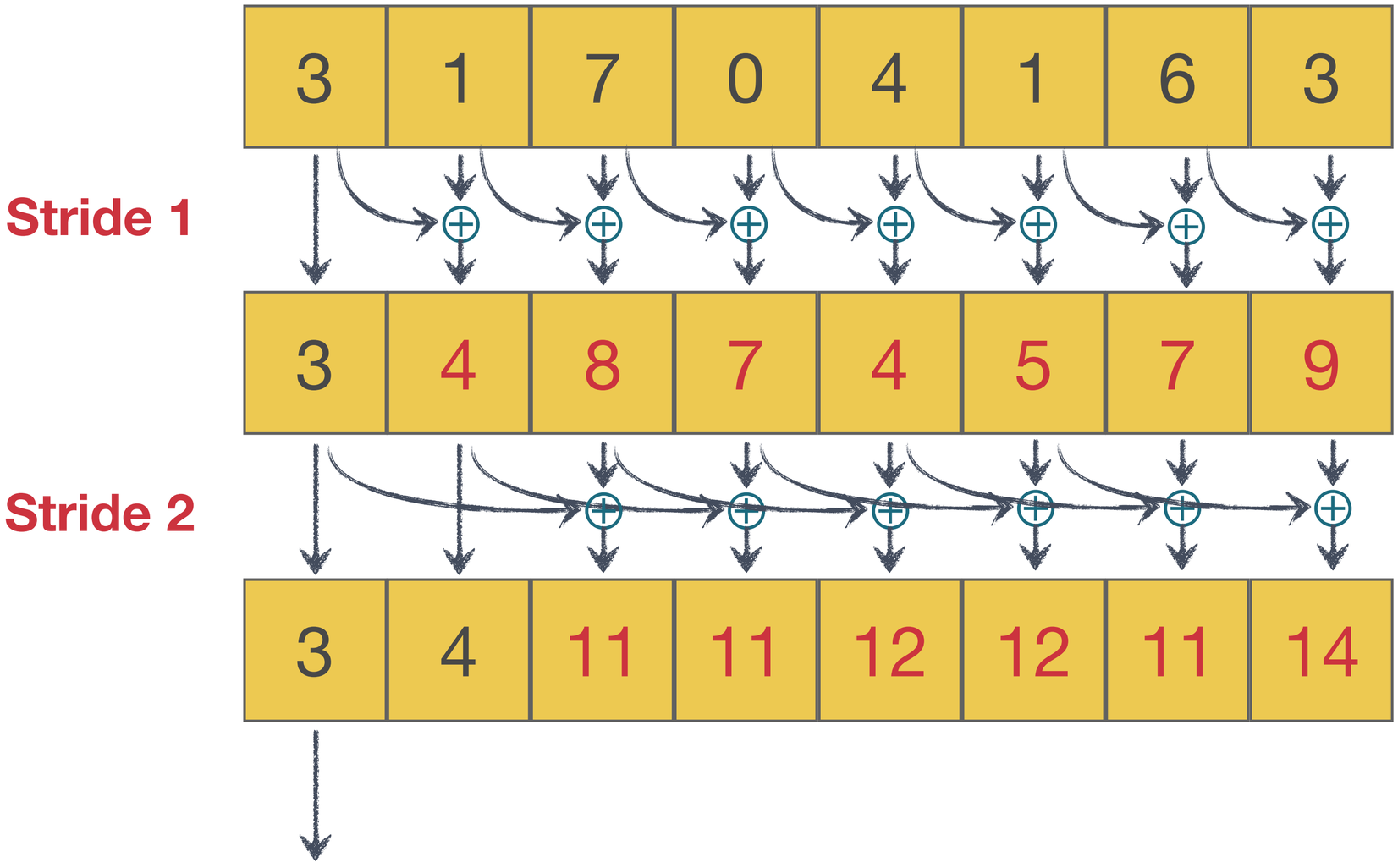
Stride 1



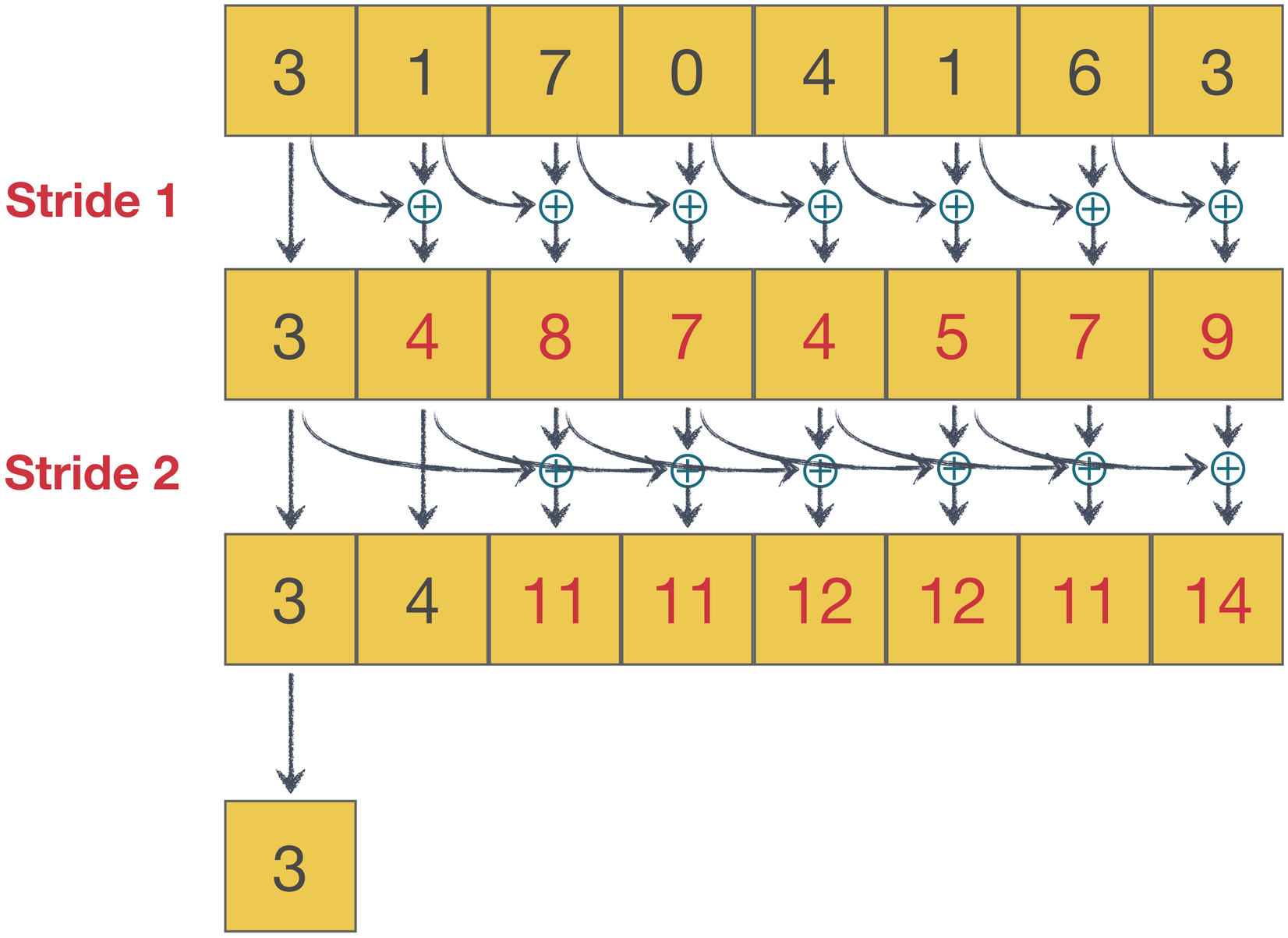
Stride 2



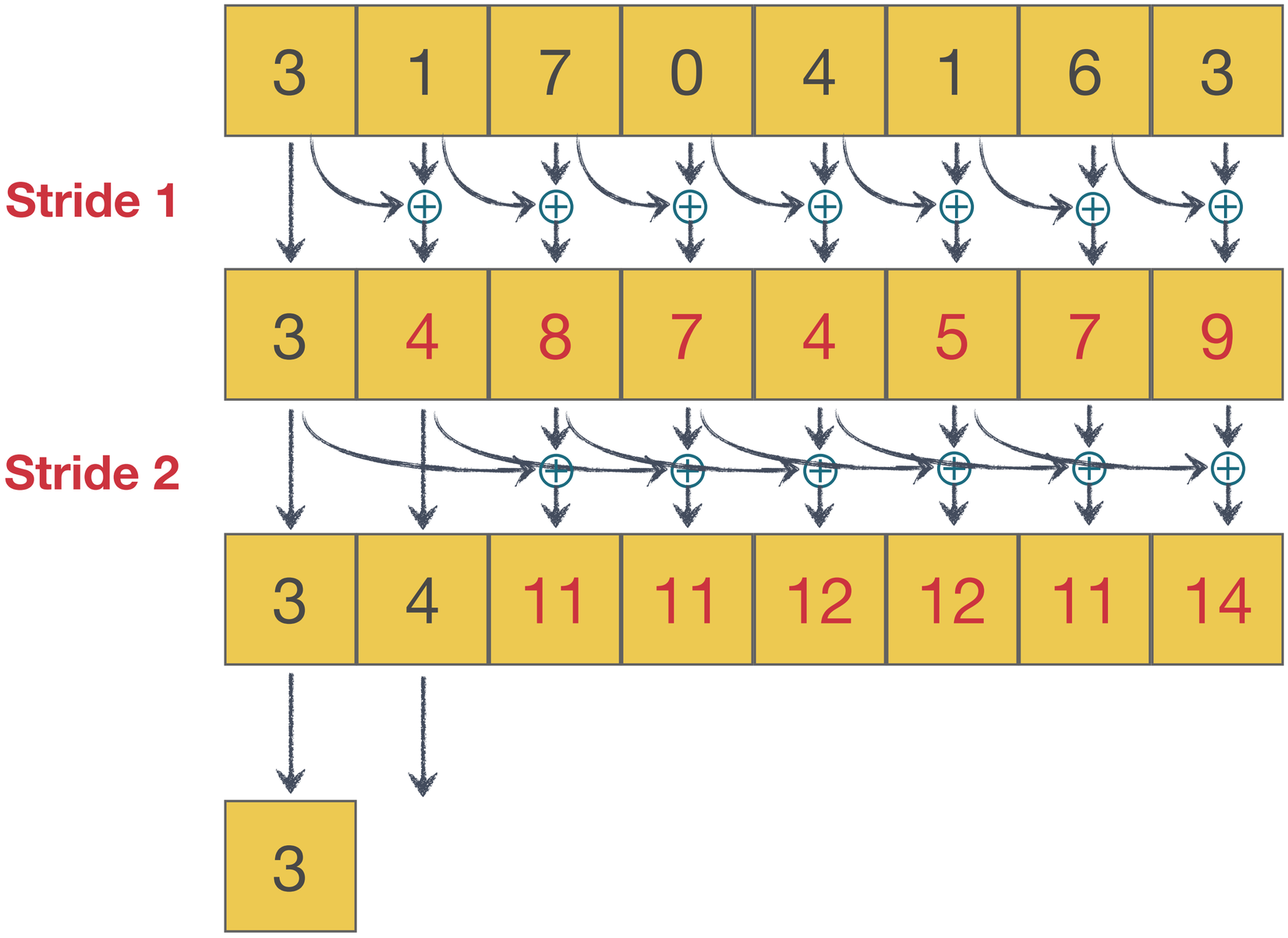
Inclusive Parallel Scan



Inclusive Parallel Scan

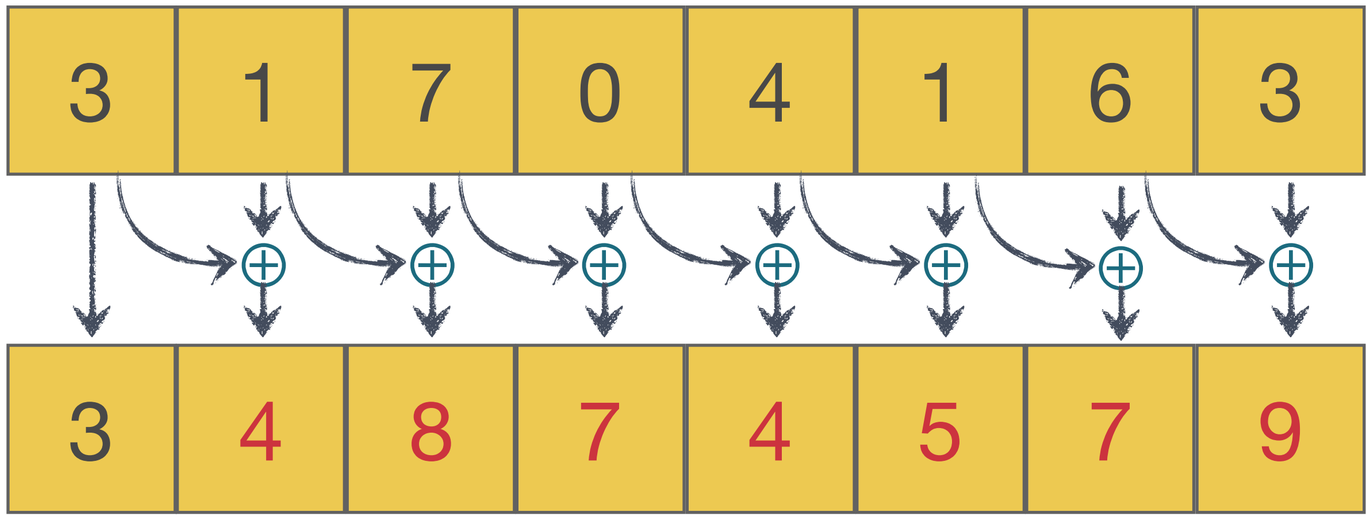


Inclusive Parallel Scan

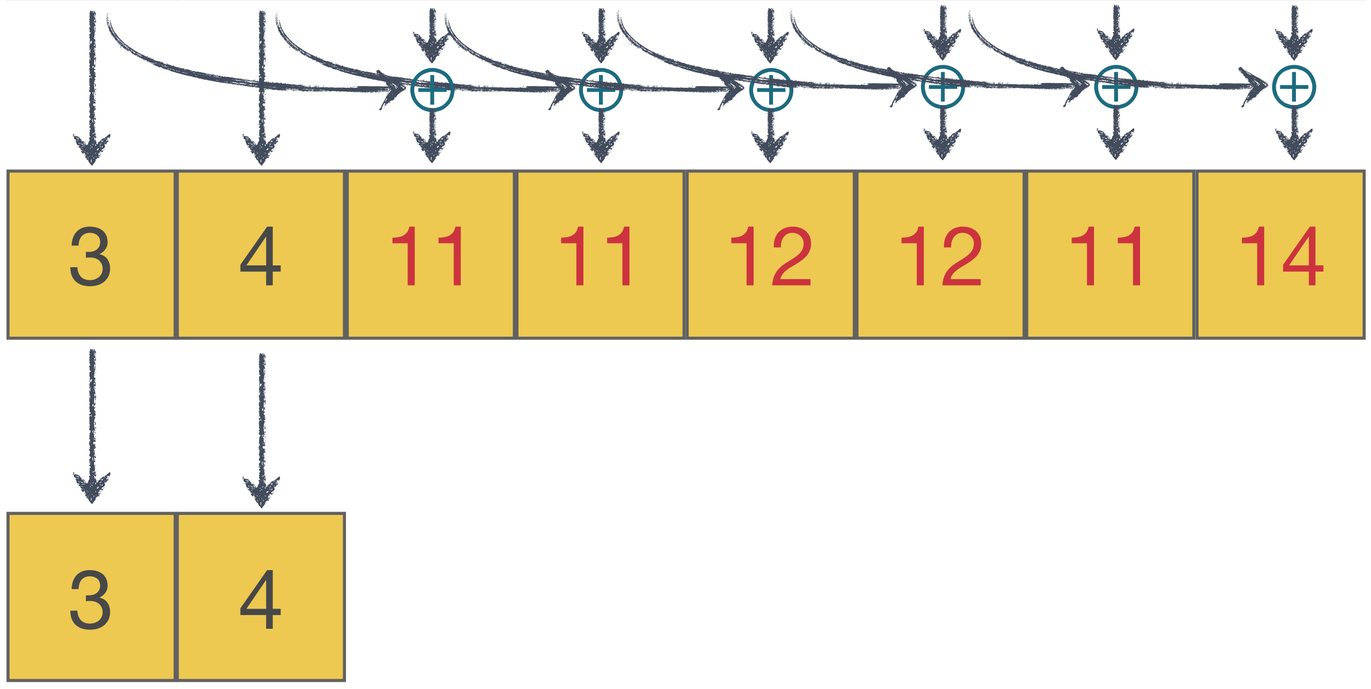


Inclusive Parallel Scan

Stride 1

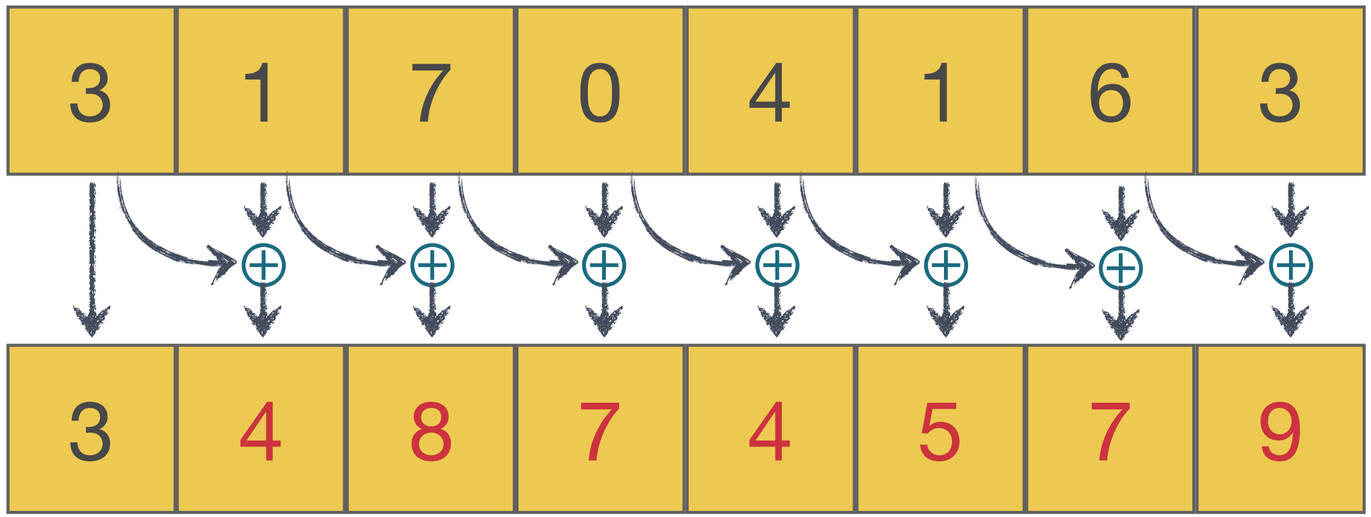


Stride 2

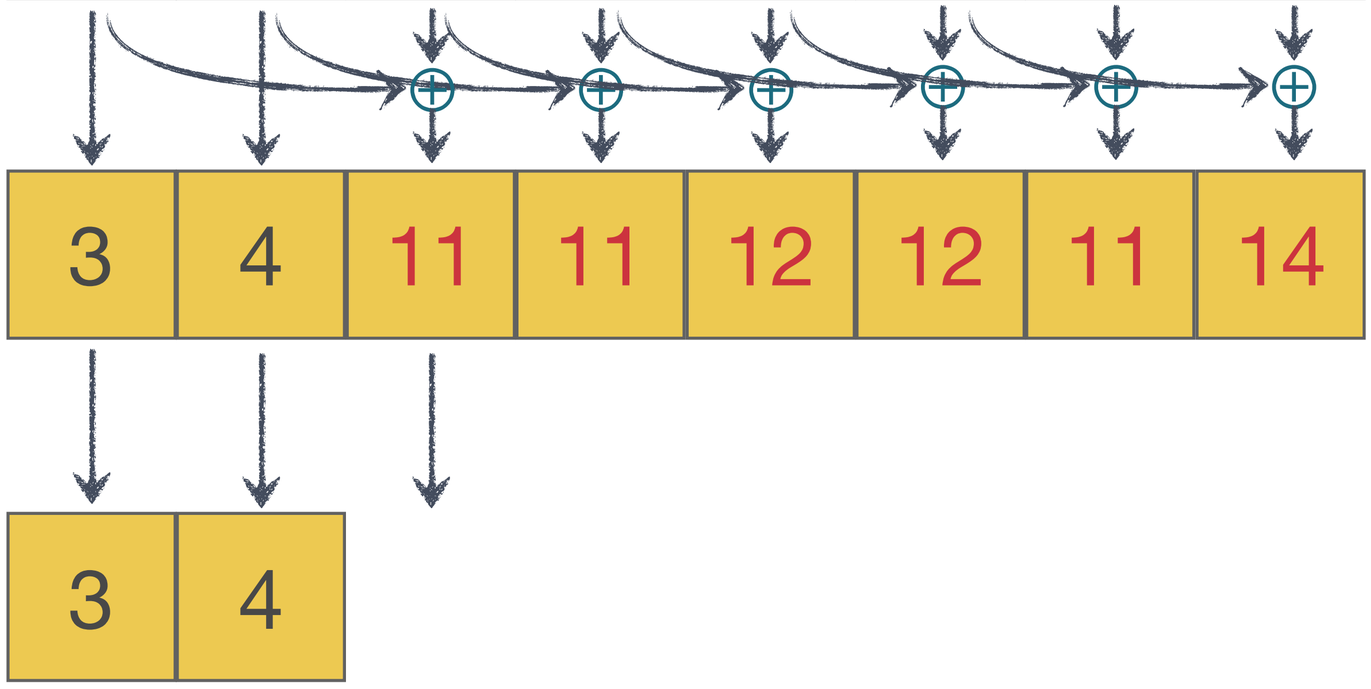


Inclusive Parallel Scan

Stride 1

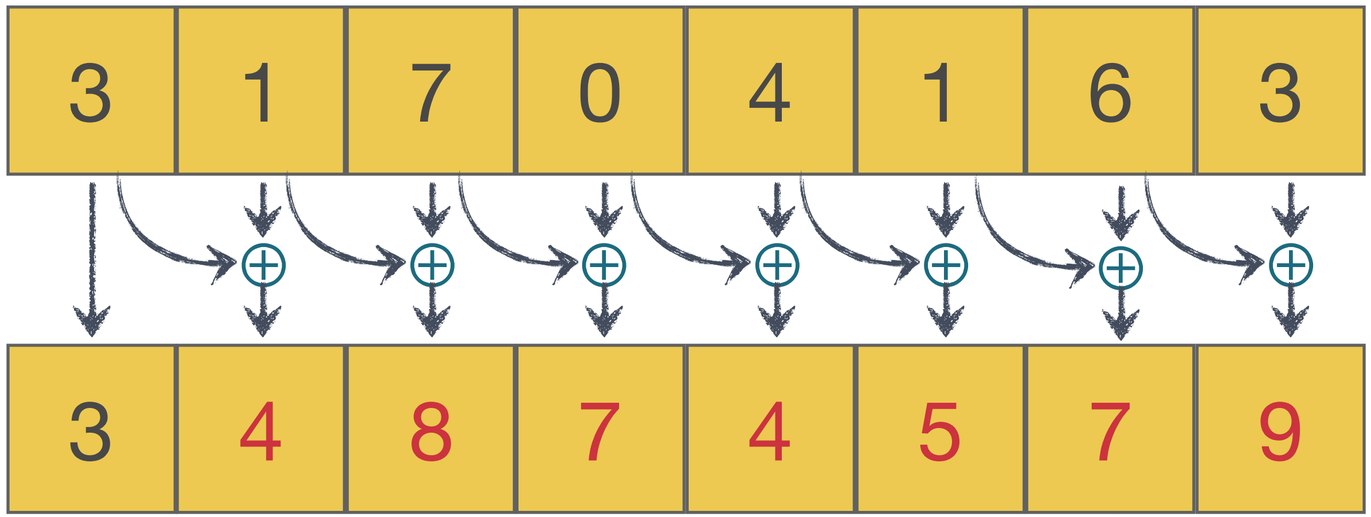


Stride 2

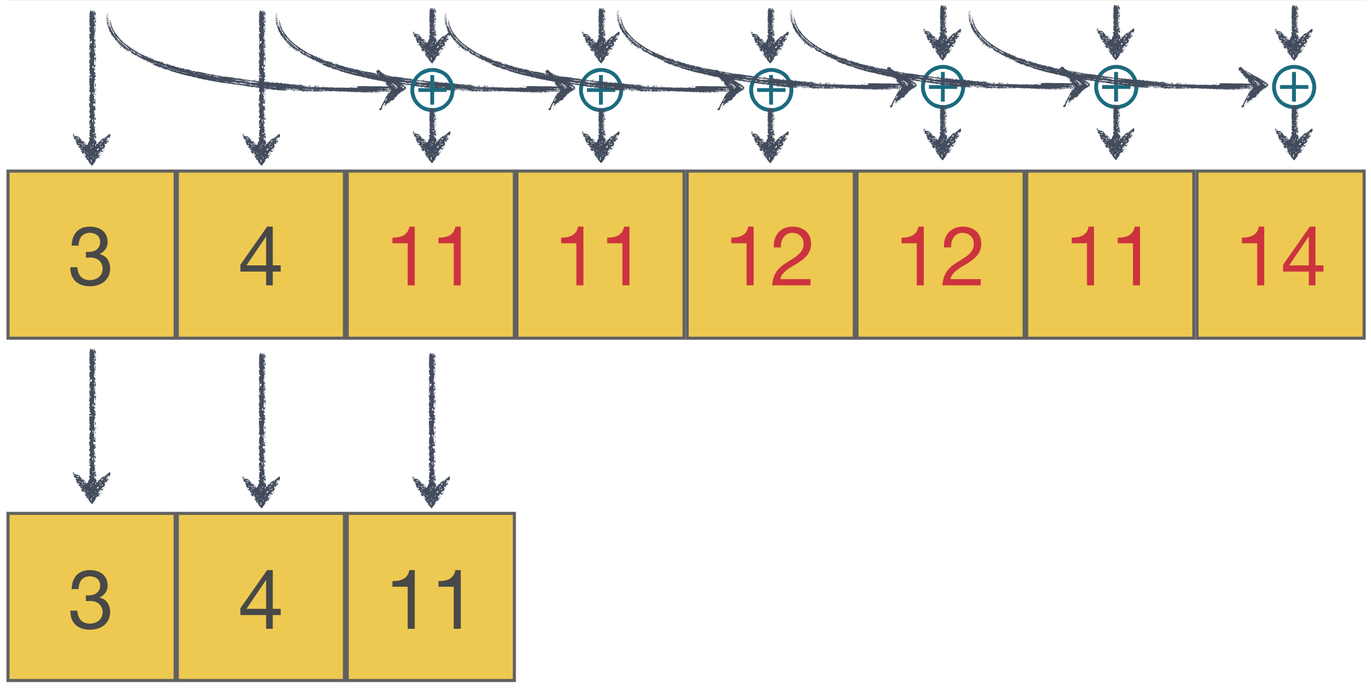


Inclusive Parallel Scan

Stride 1

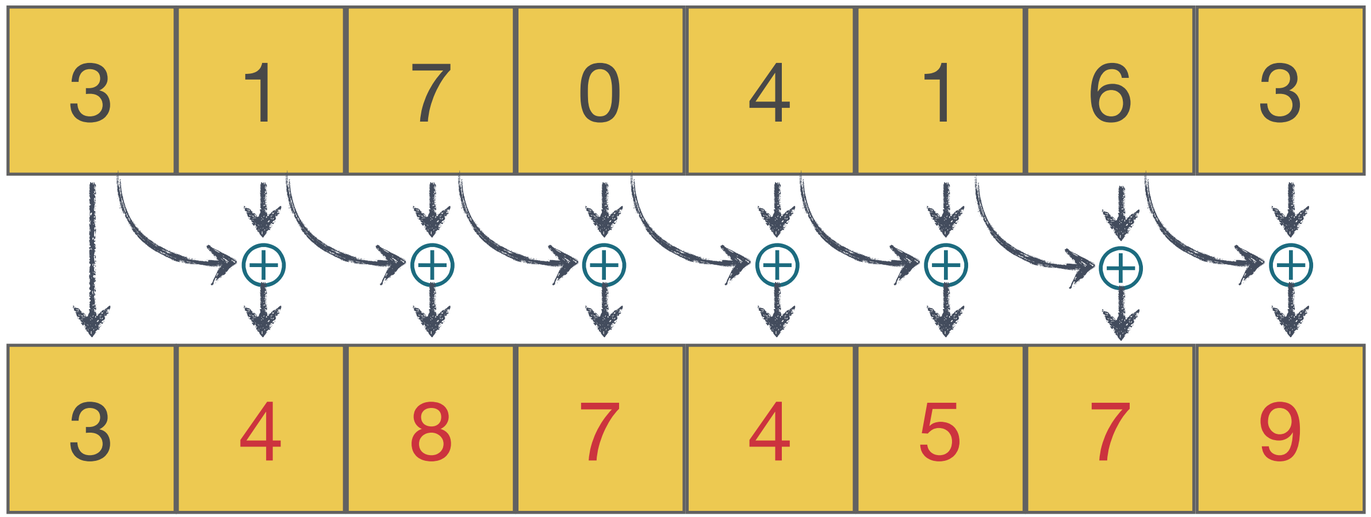


Stride 2

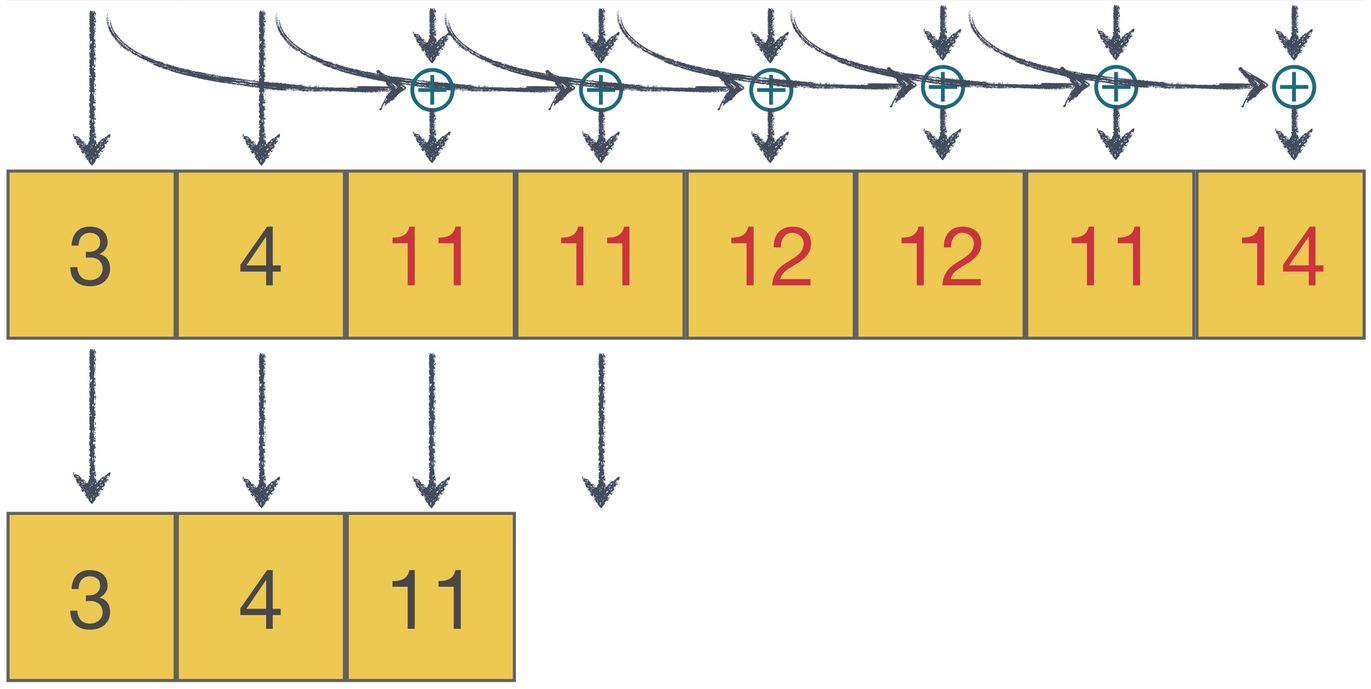


Inclusive Parallel Scan

Stride 1

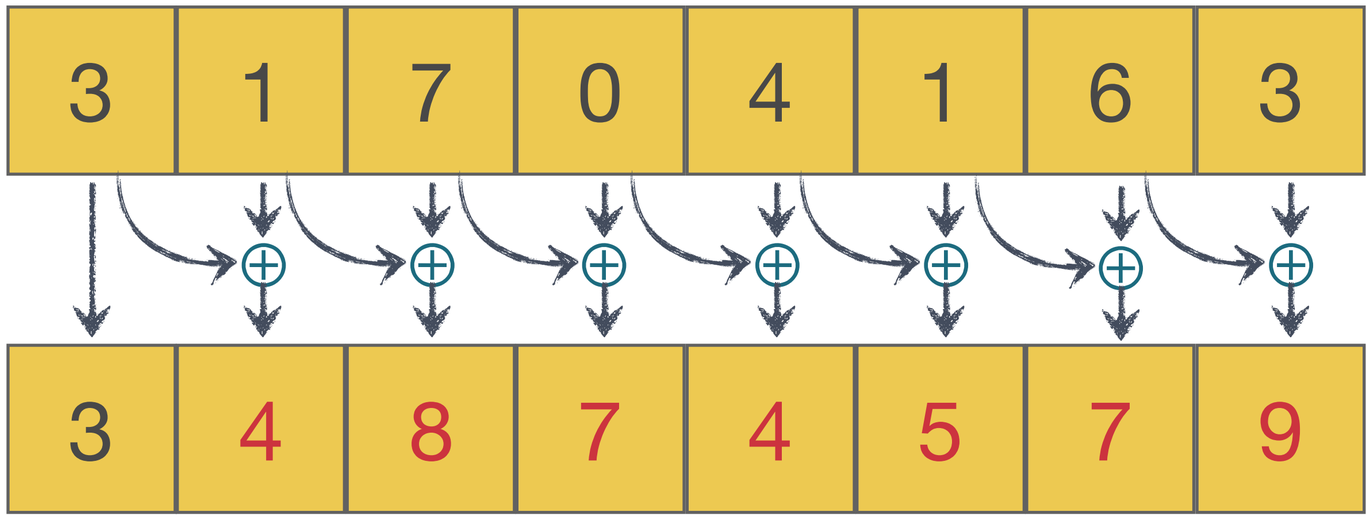


Stride 2

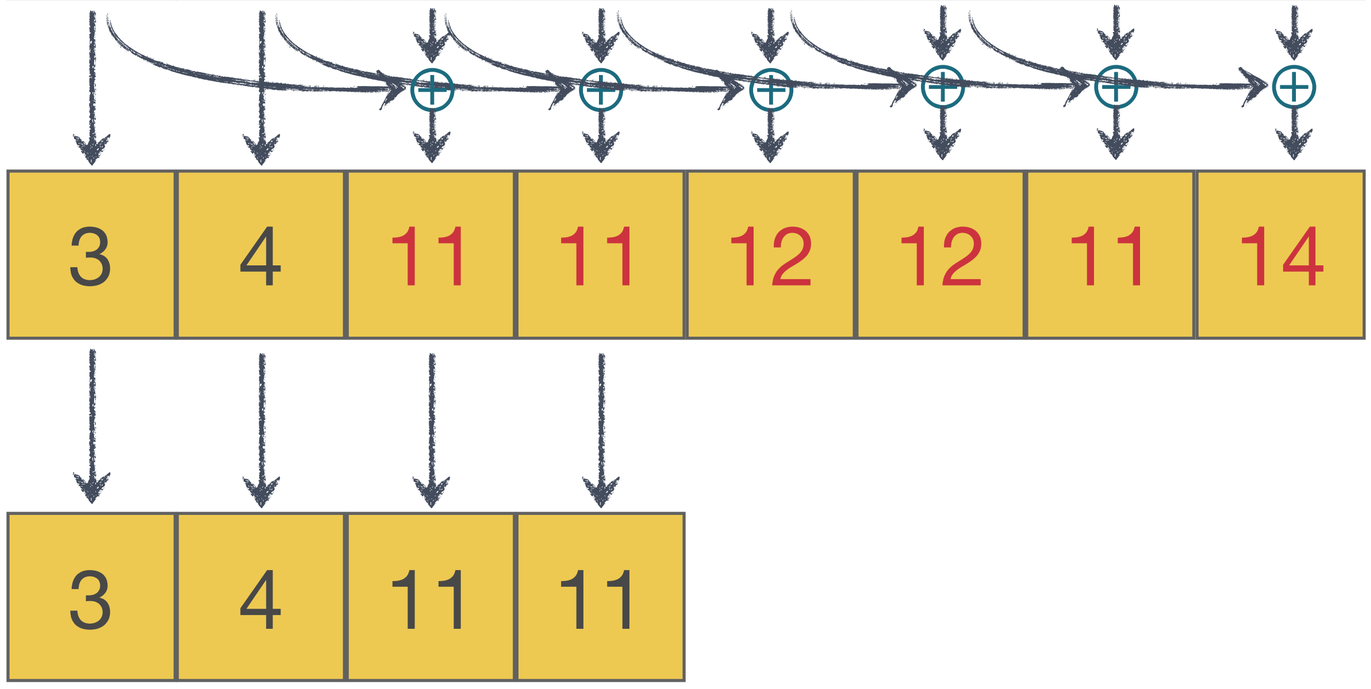


Inclusive Parallel Scan

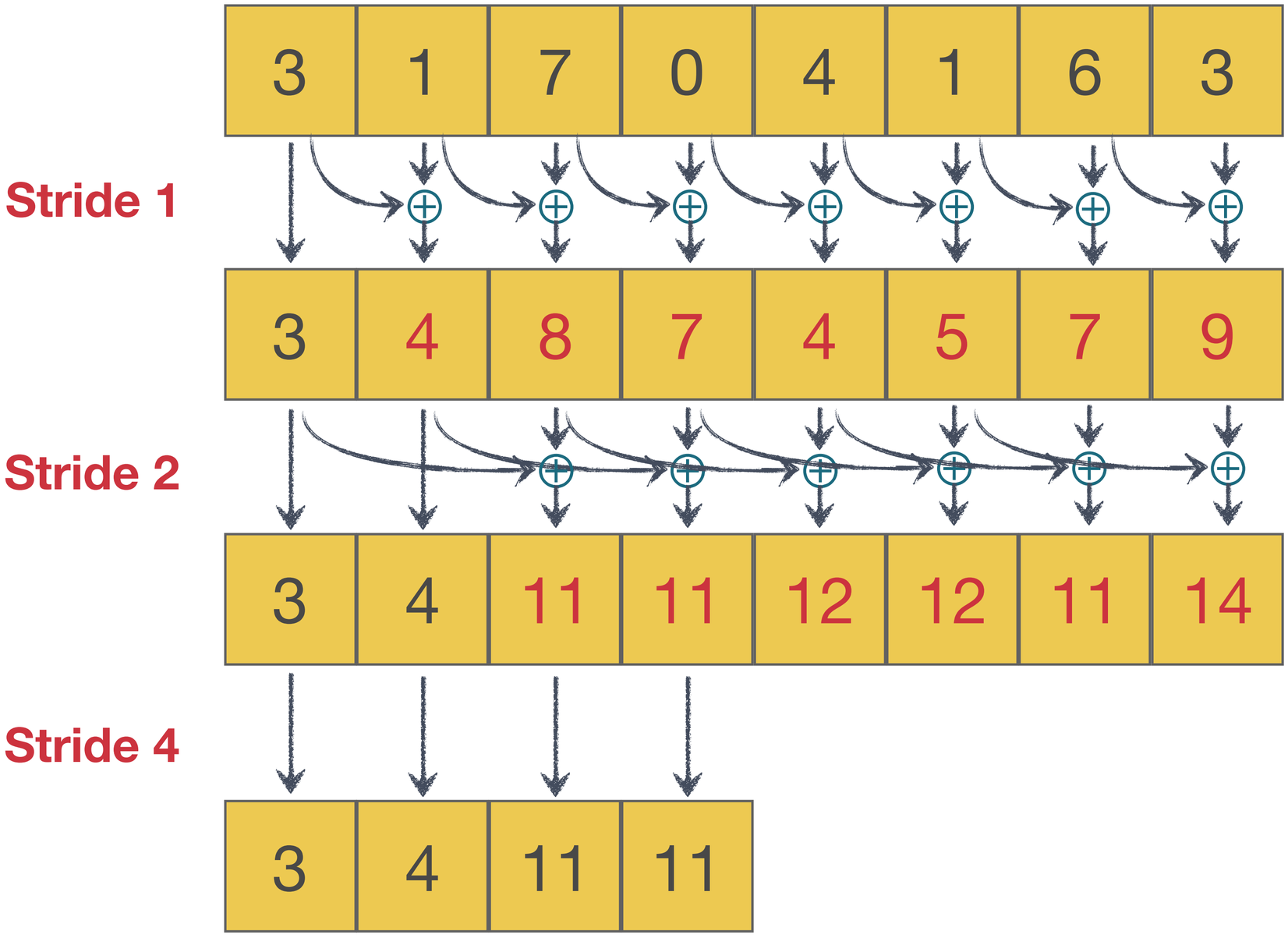
Stride 1



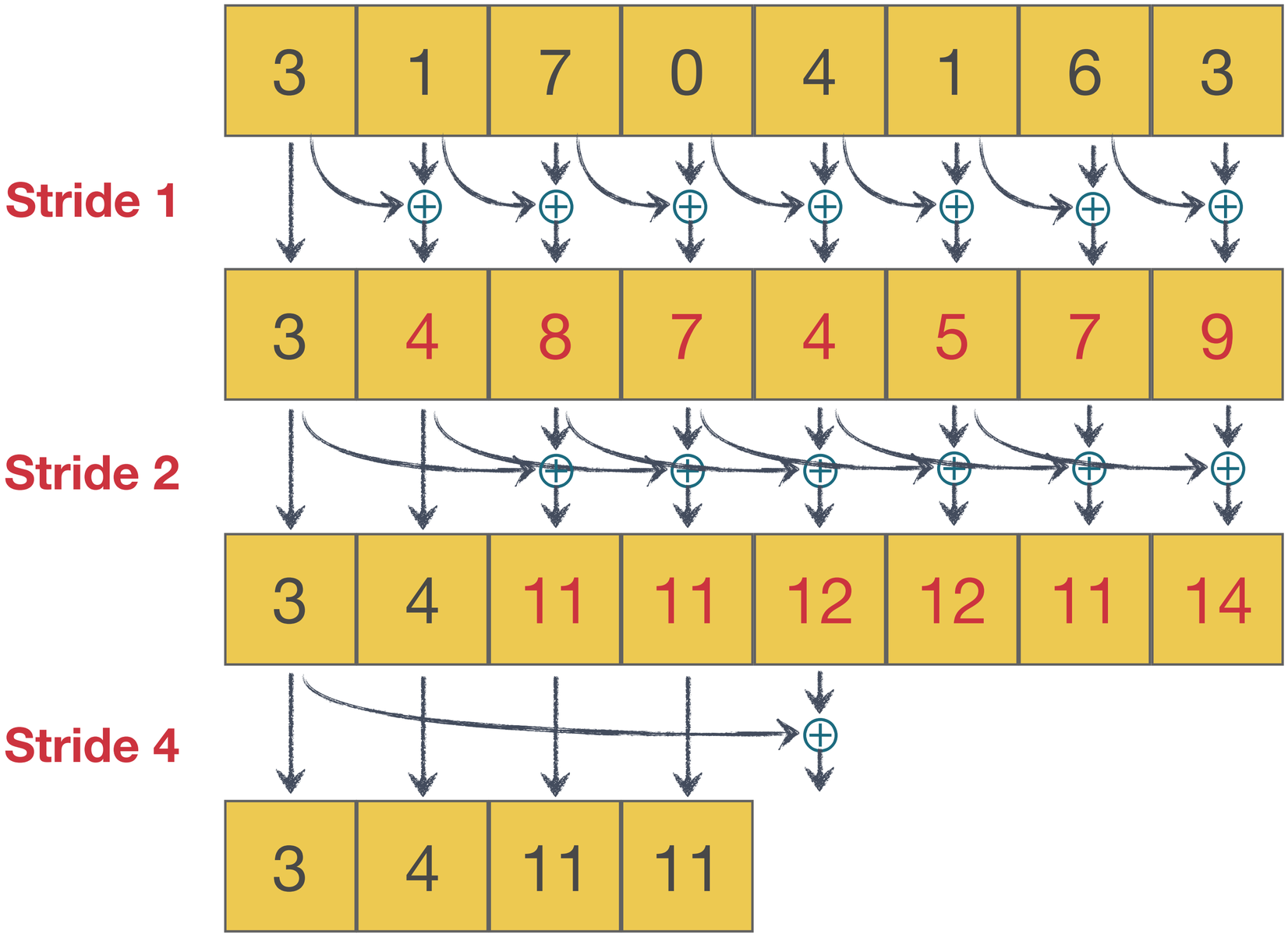
Stride 2



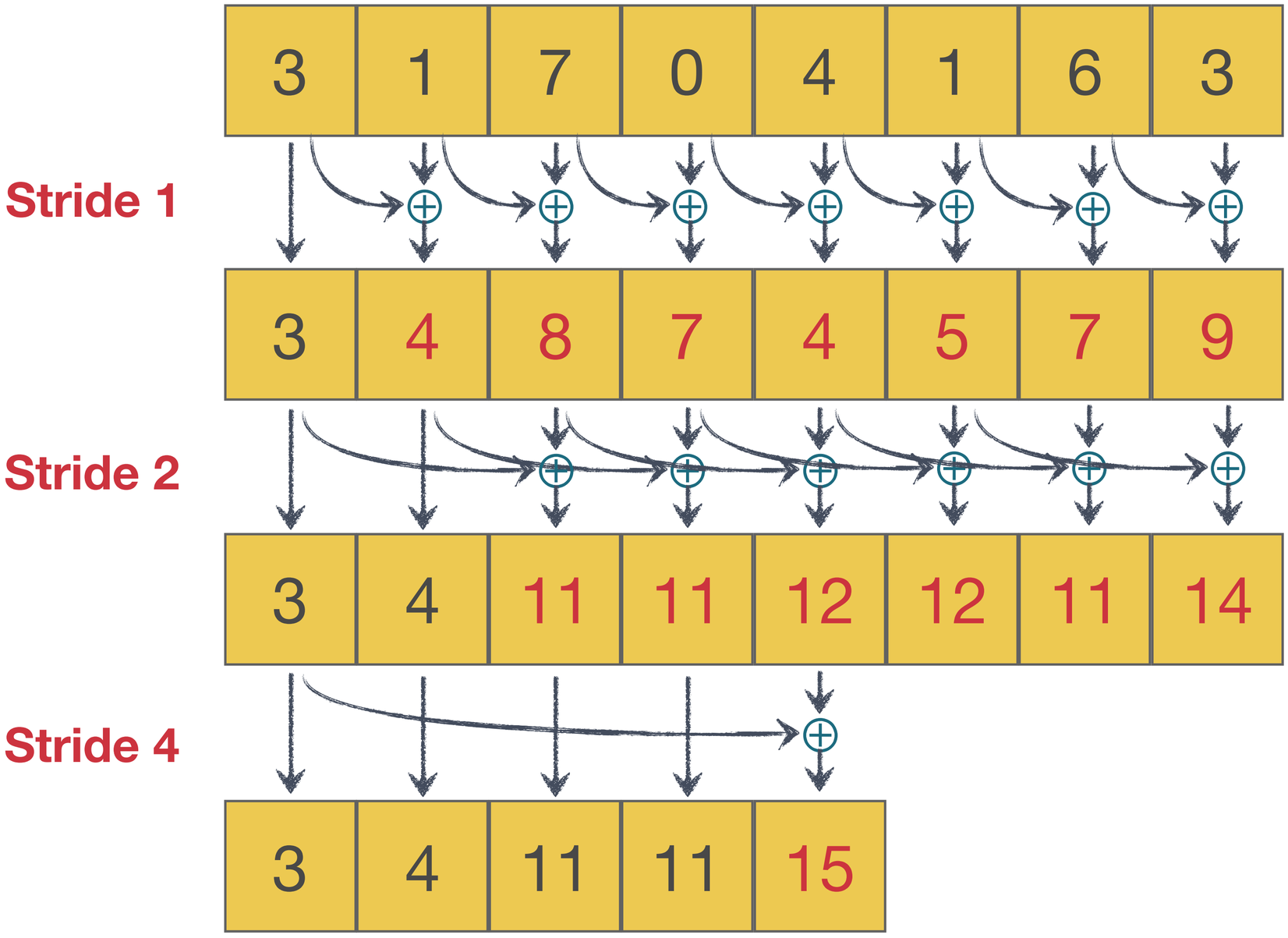
Inclusive Parallel Scan



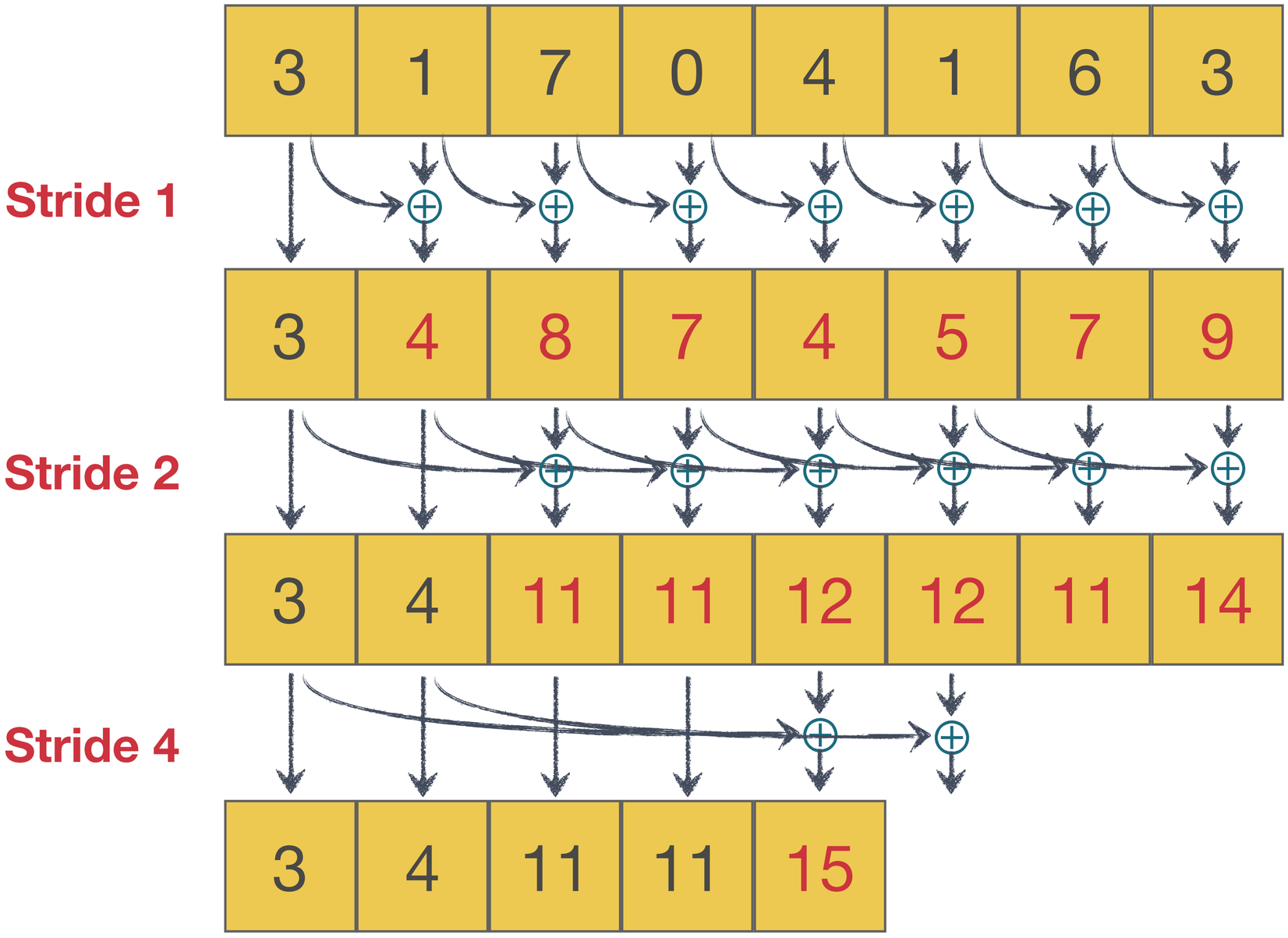
Inclusive Parallel Scan



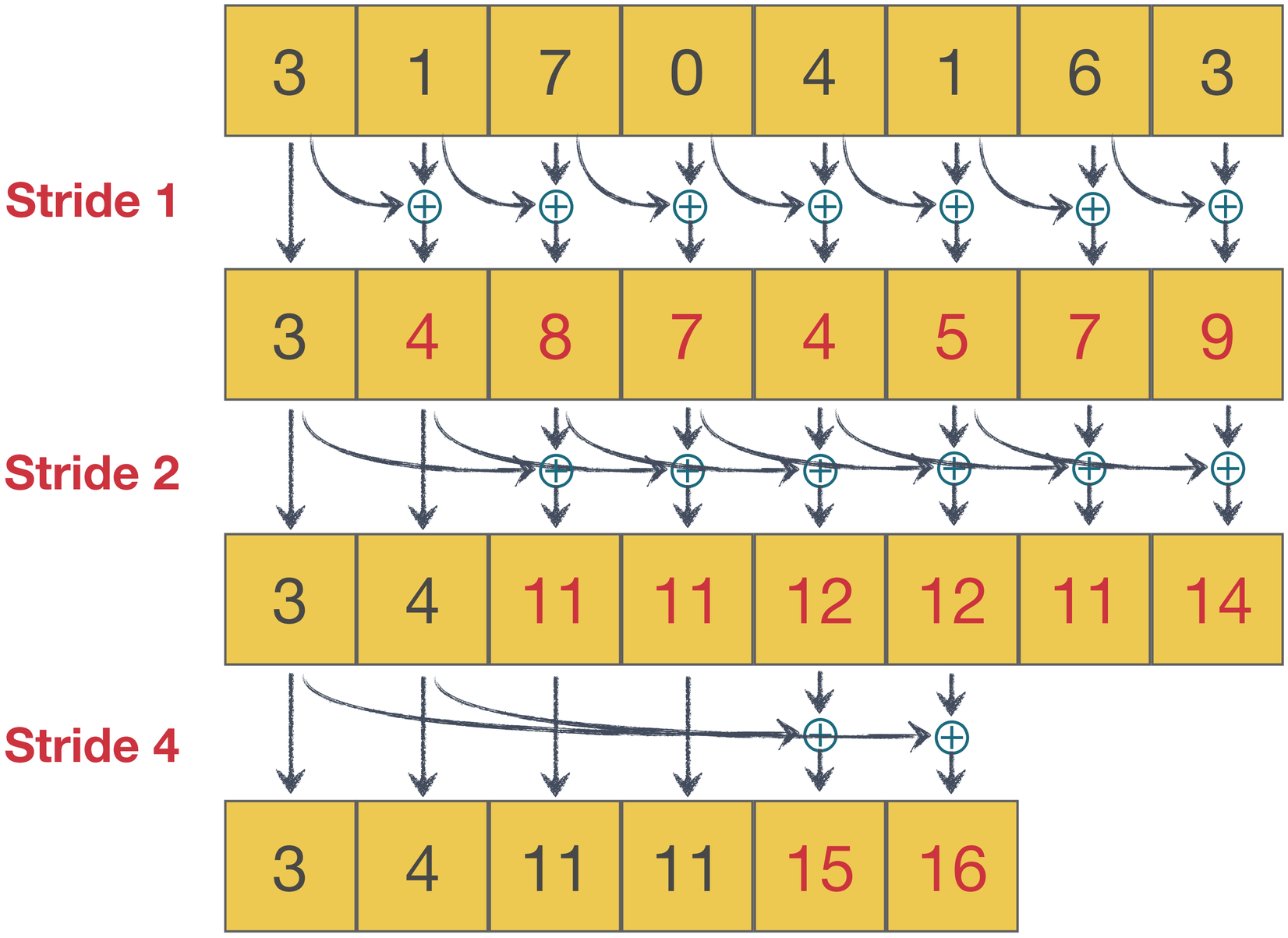
Inclusive Parallel Scan



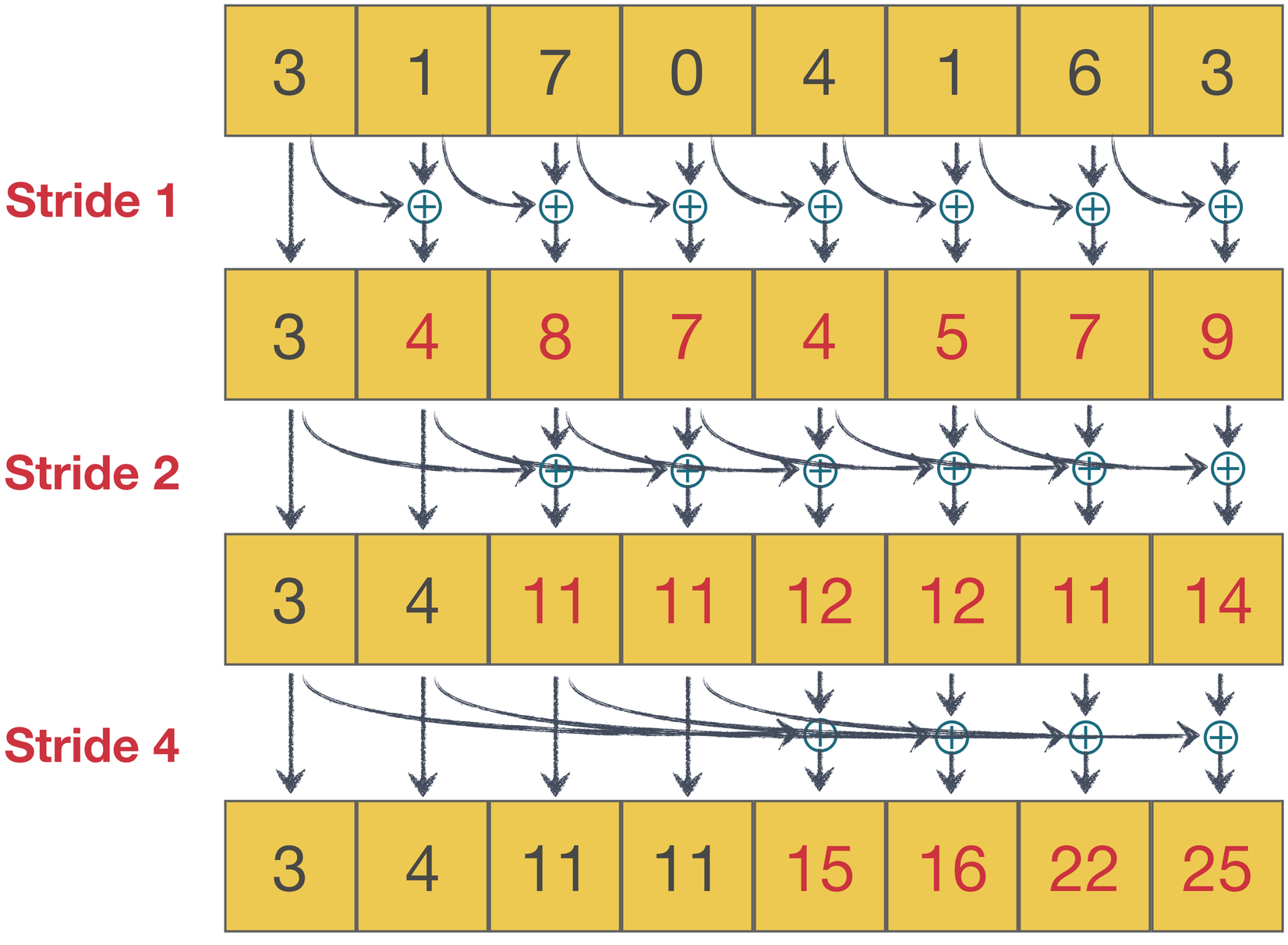
Inclusive Parallel Scan



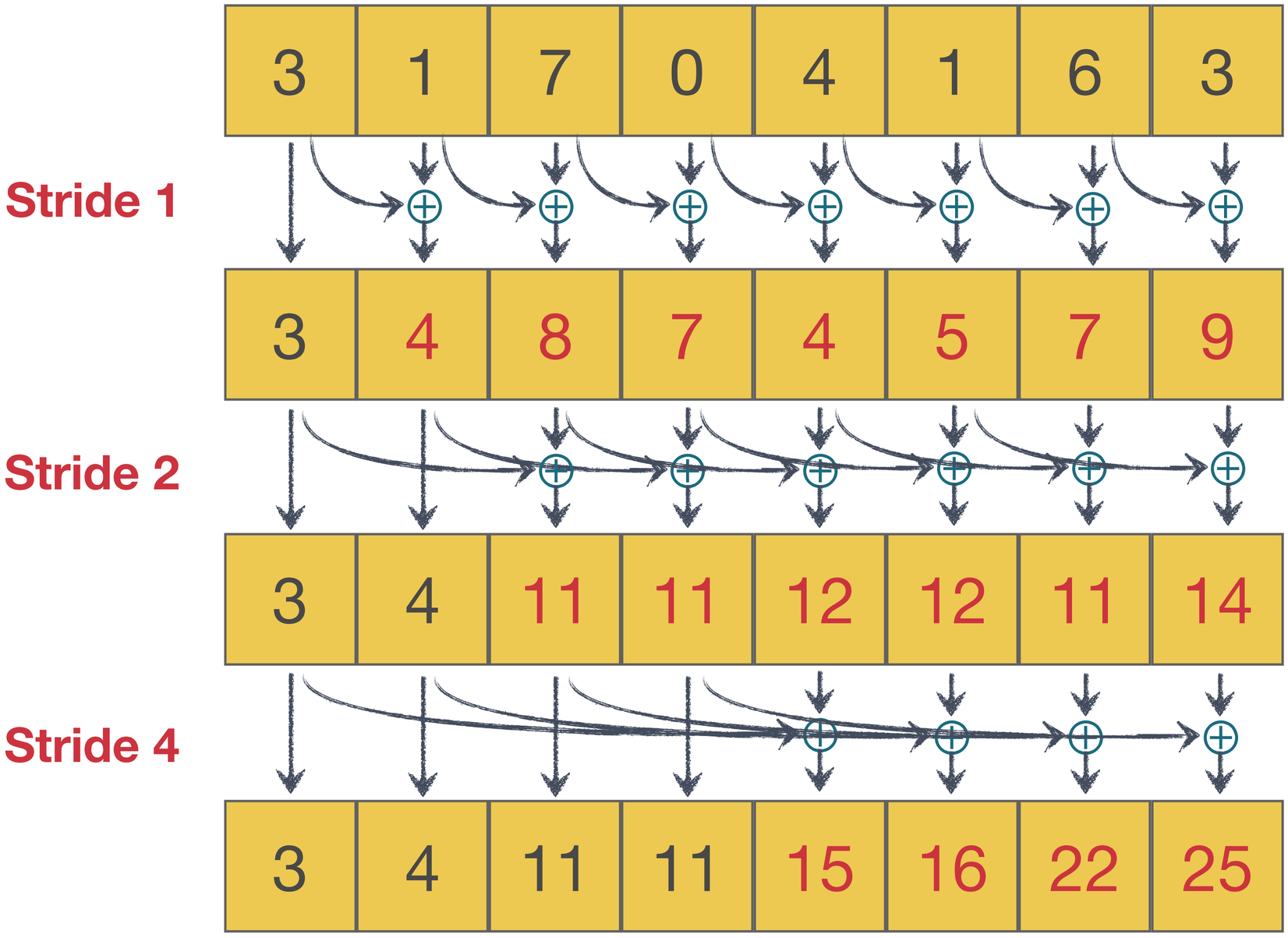
Inclusive Parallel Scan



Inclusive Parallel Scan

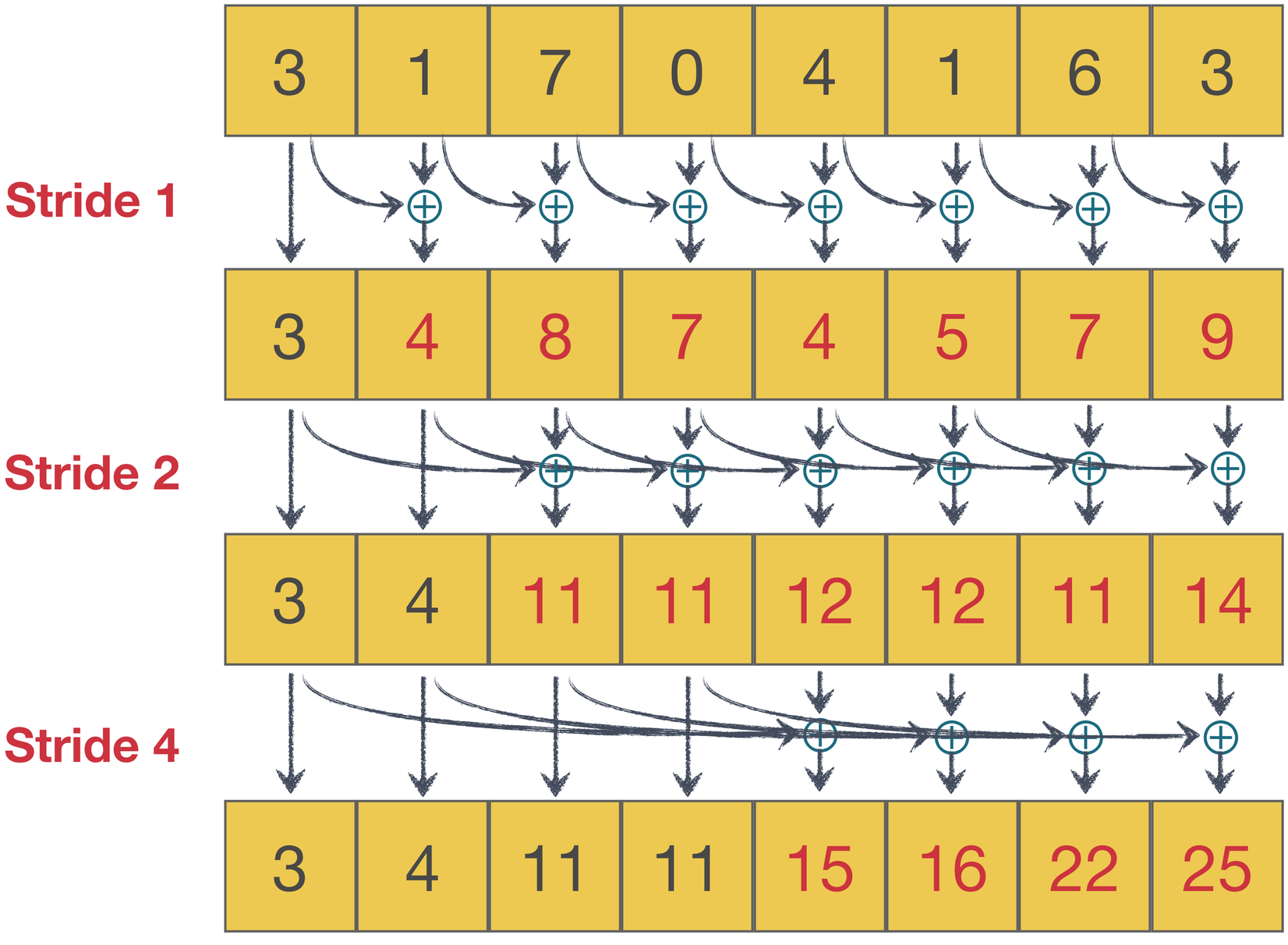


Inclusive Parallel Scan



Iterate log(n) times adding pairs of elements *stride* elements apart.

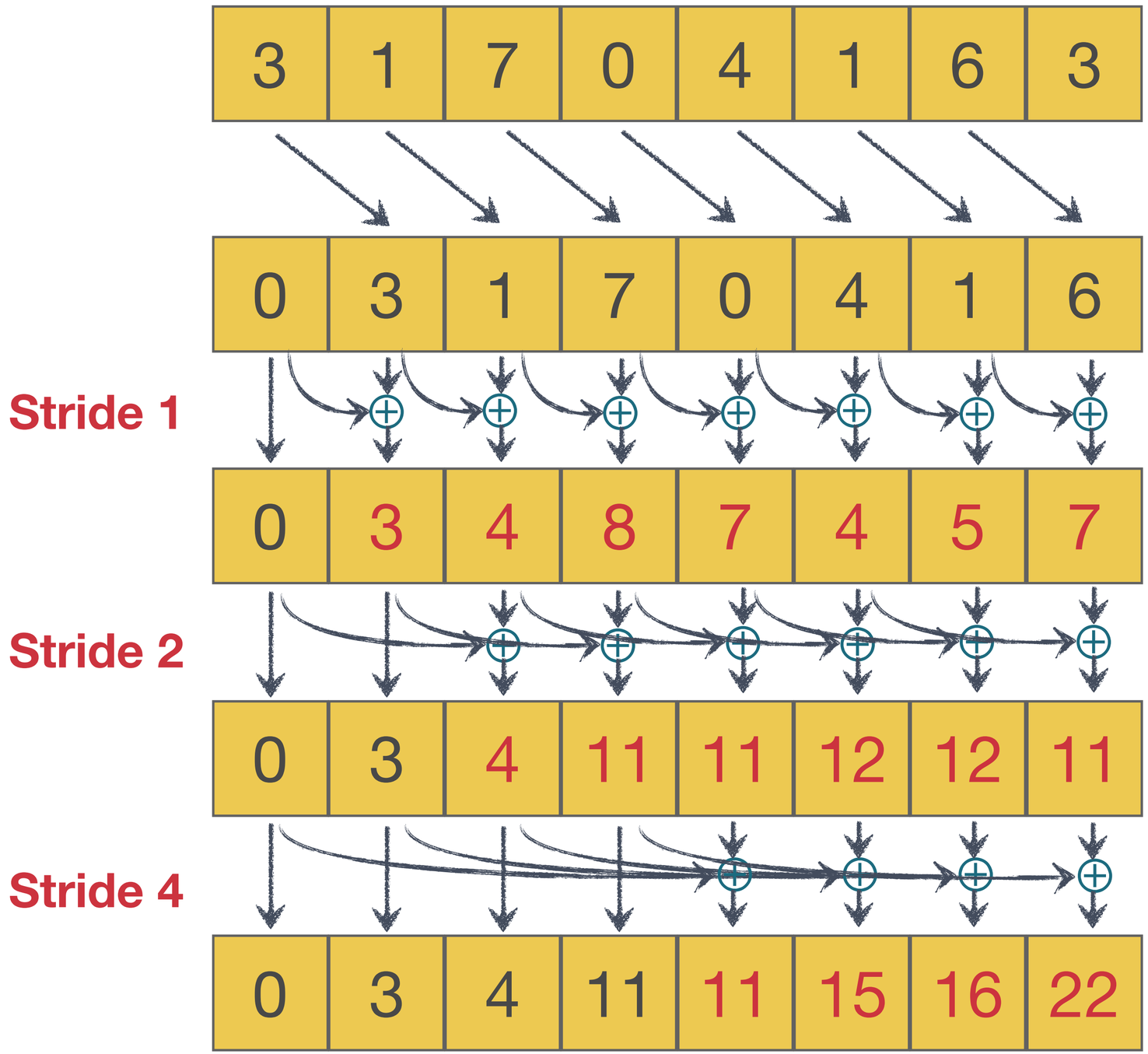
Inclusive Parallel Scan



Iterate log(n) times adding pairs of elements *stride* elements apart.

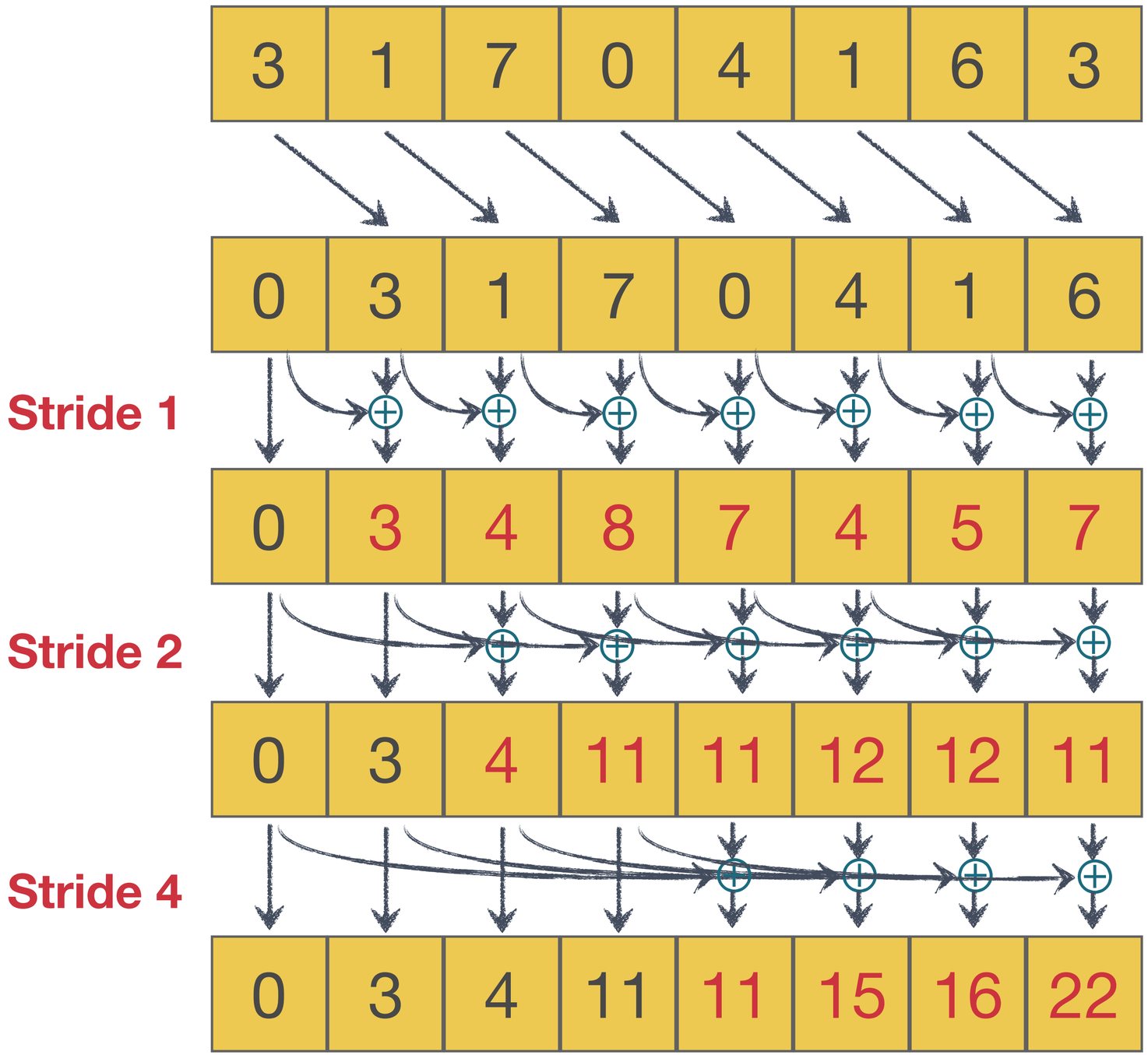
$$W(n) = \mathcal{O}(n \log n)$$
$$D(n) = \mathcal{O}(\log n)$$

Exclusive Parallel Scan



Iterate log(n) times adding pairs of elements *stride* elements apart.

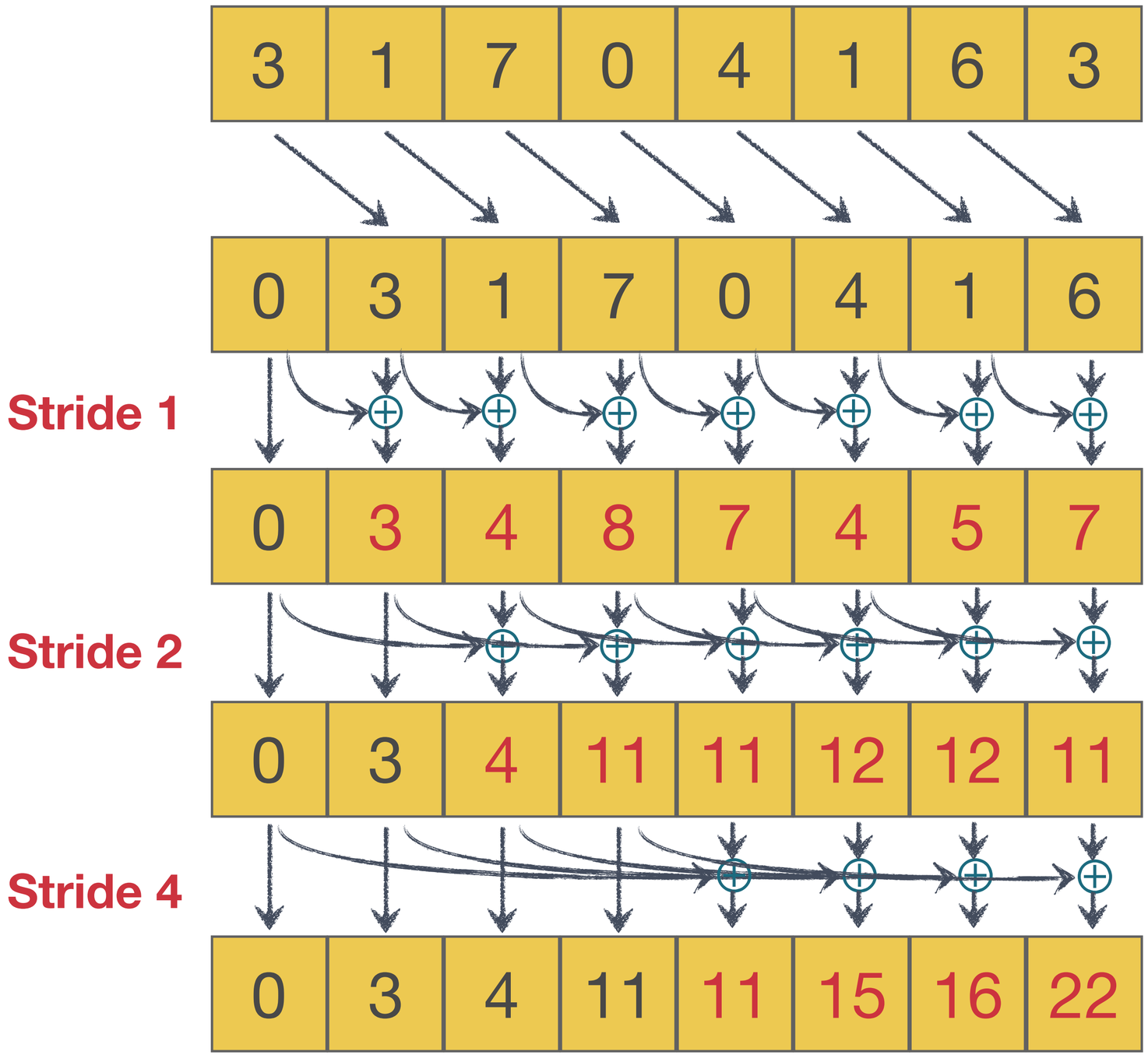
Exclusive Parallel Scan



Iterate log(n) times adding pairs of elements *stride* elements apart.

$$W(n) = \mathcal{O}(n \log n)$$
$$D(n) = \mathcal{O}(\log n)$$

Exclusive Parallel Scan



Iterate log(n) times adding pairs of elements *stride* elements apart.

$$W(n) = \mathcal{O}(n \log n)$$
$$D(n) = \mathcal{O}(\log n)$$

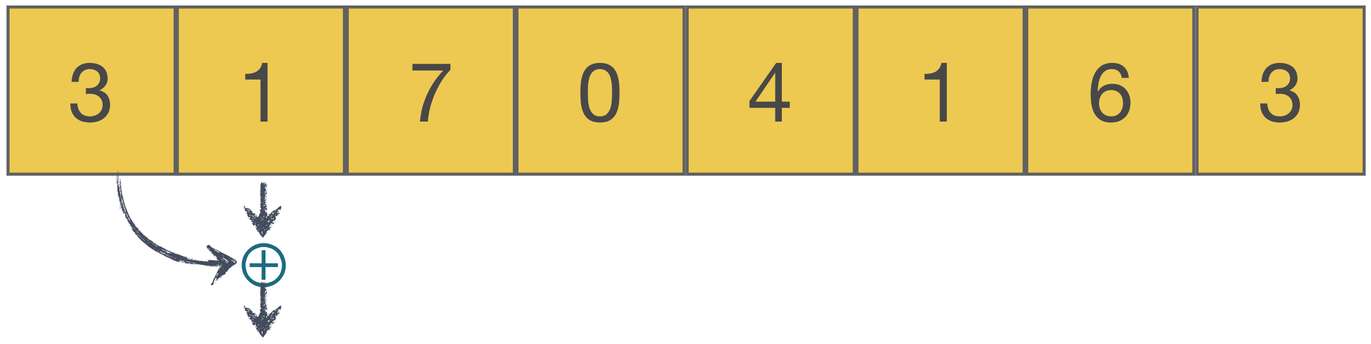
Can we do better?

Parallel Scan: Reduction

Parallel Scan: Reduction

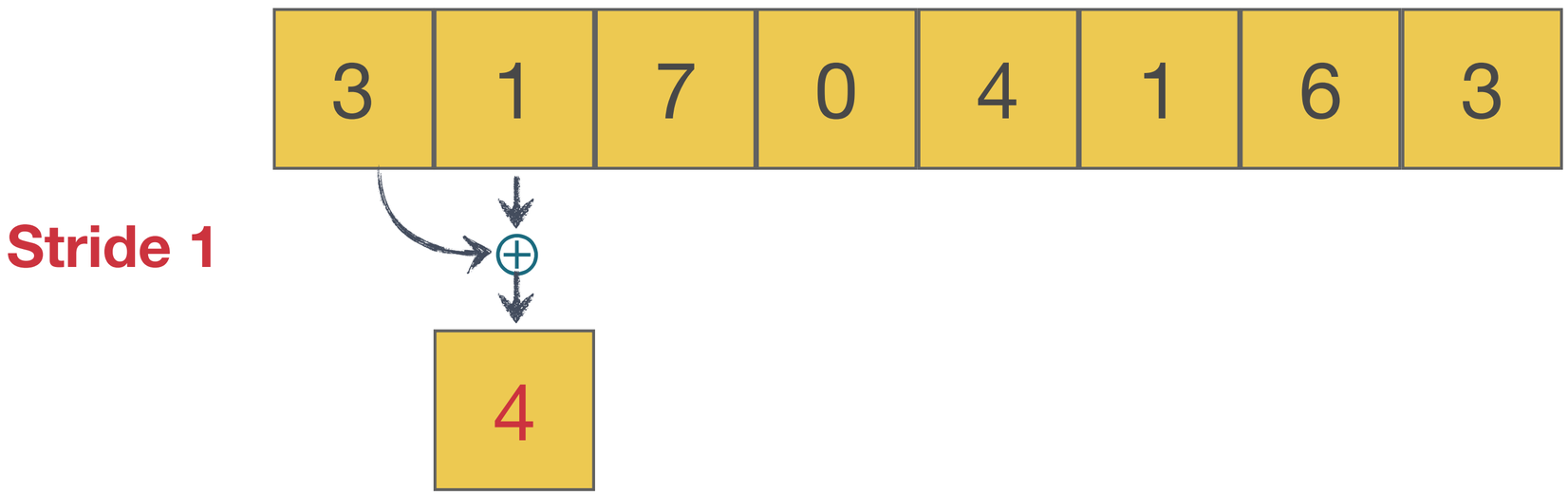
Stride 1

Parallel Scan: Reduction

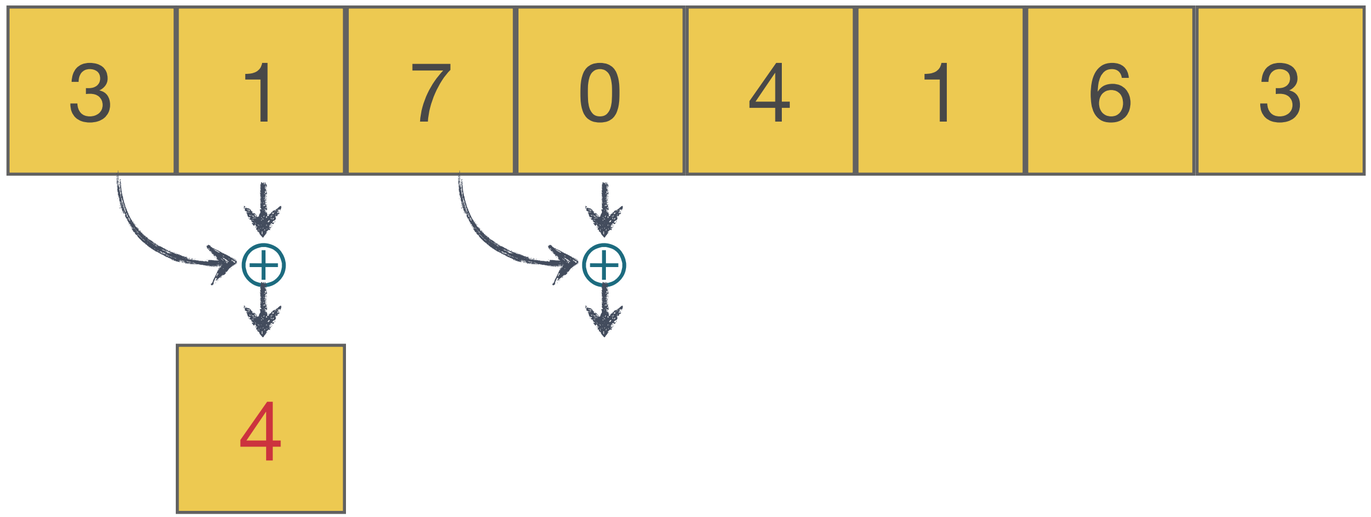


Stride 1

Parallel Scan: Reduction

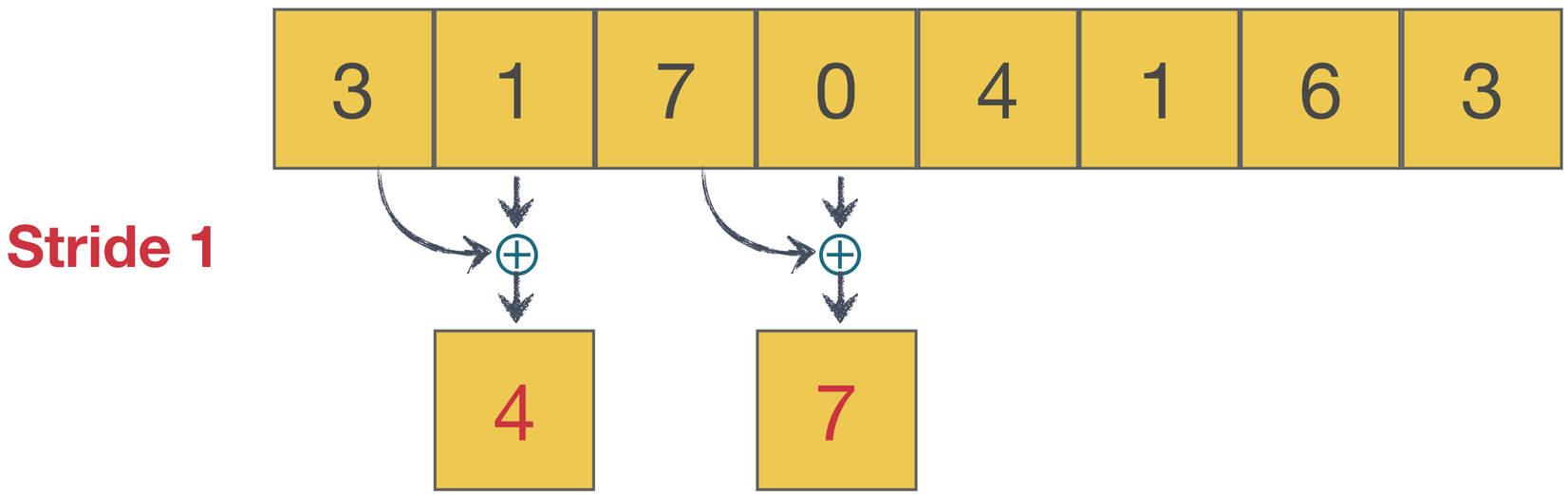


Parallel Scan: Reduction

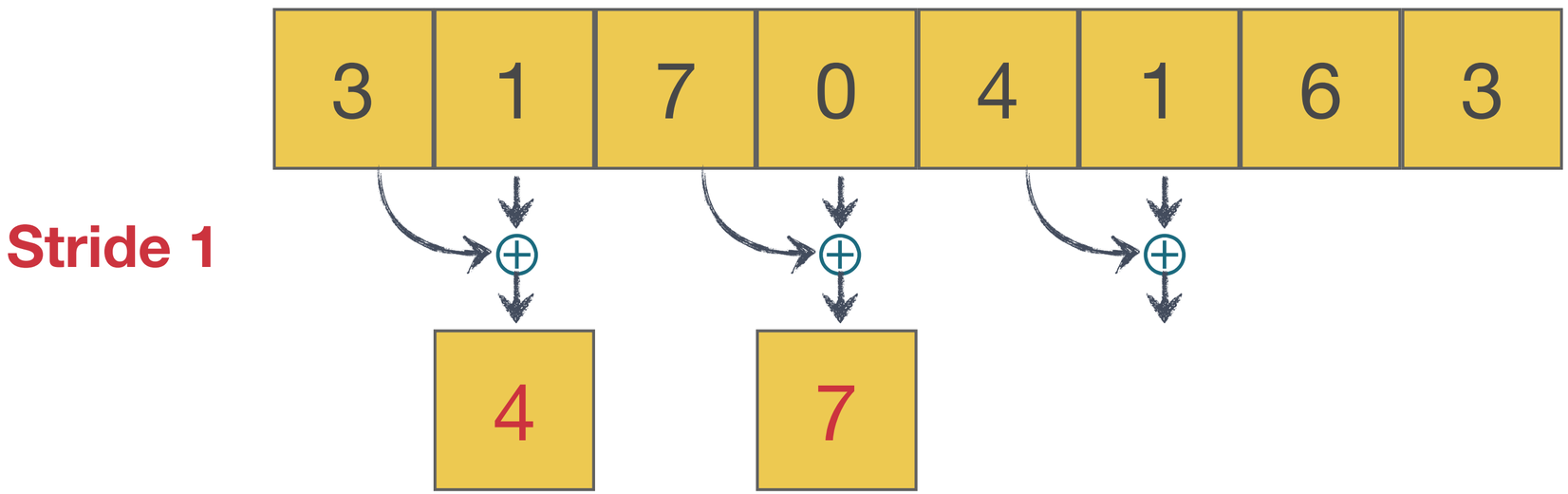


Stride 1

Parallel Scan: Reduction

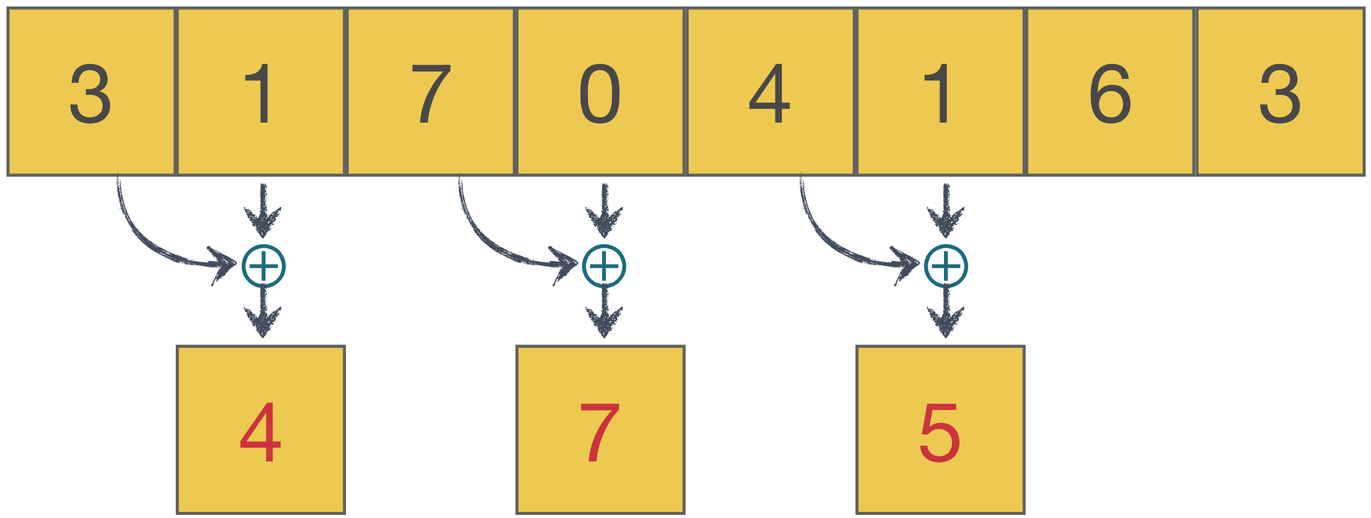


Parallel Scan: Reduction



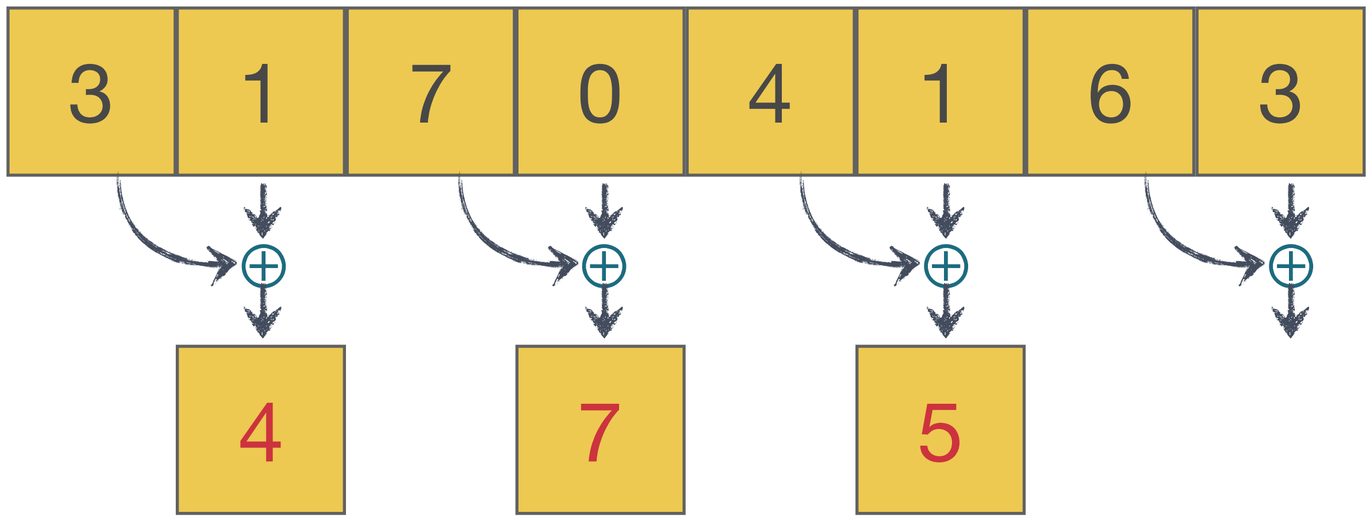
Parallel Scan: Reduction

Stride 1



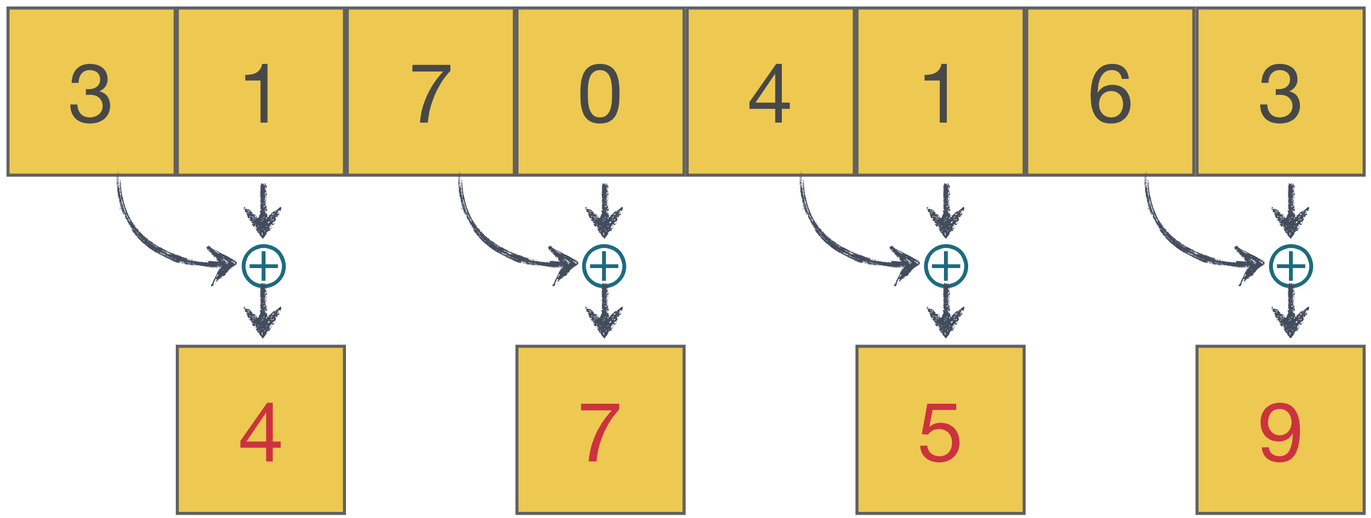
Parallel Scan: Reduction

Stride 1



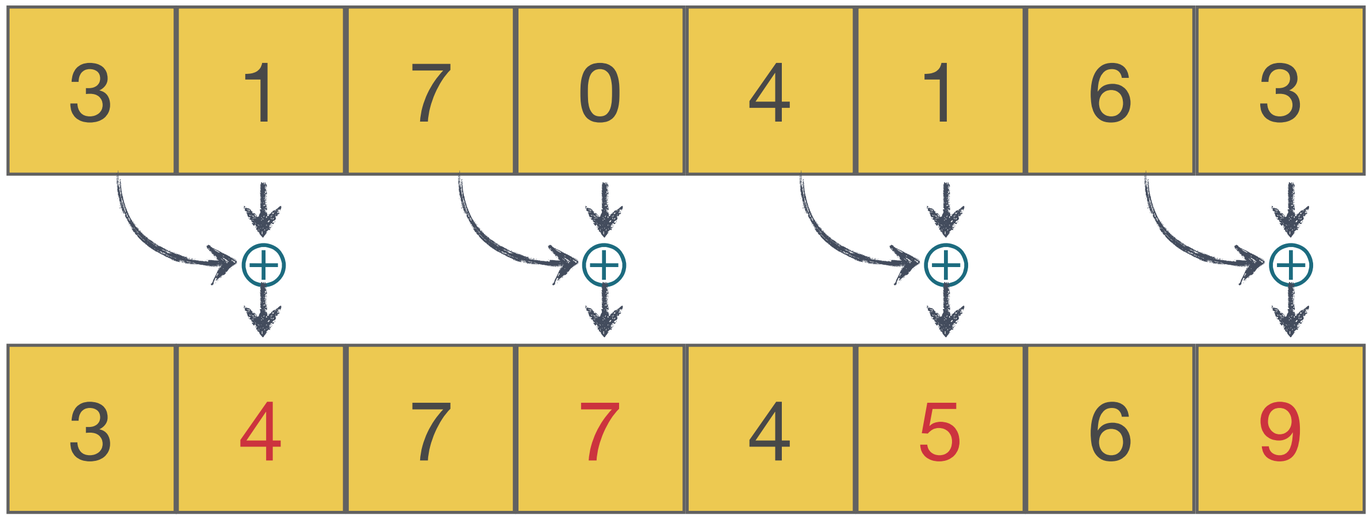
Parallel Scan: Reduction

Stride 1



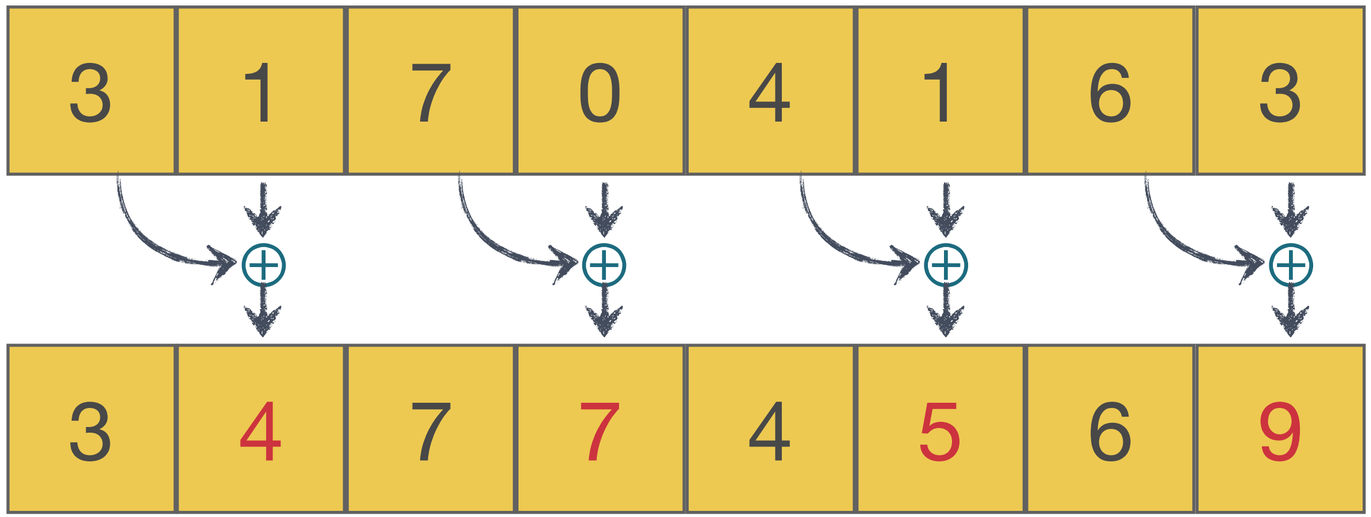
Parallel Scan: Reduction

Stride 1



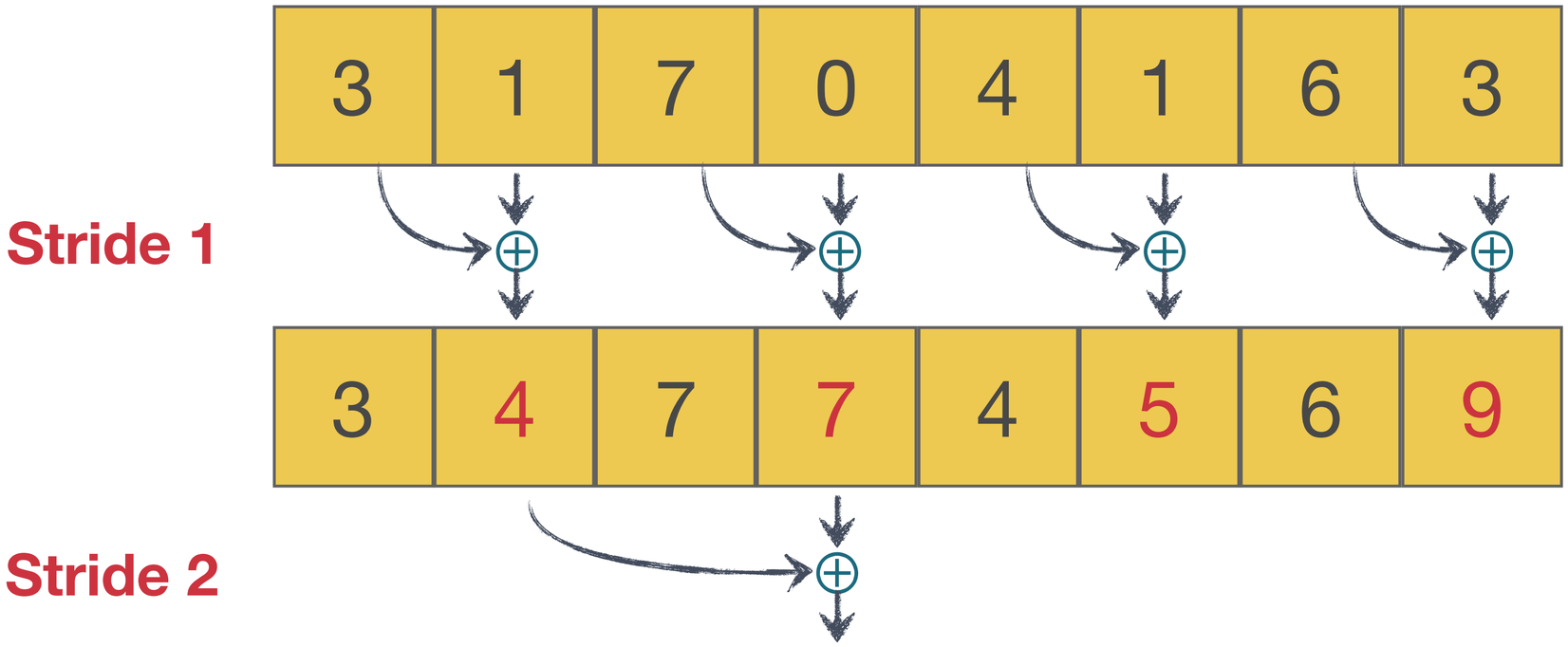
Parallel Scan: Reduction

Stride 1

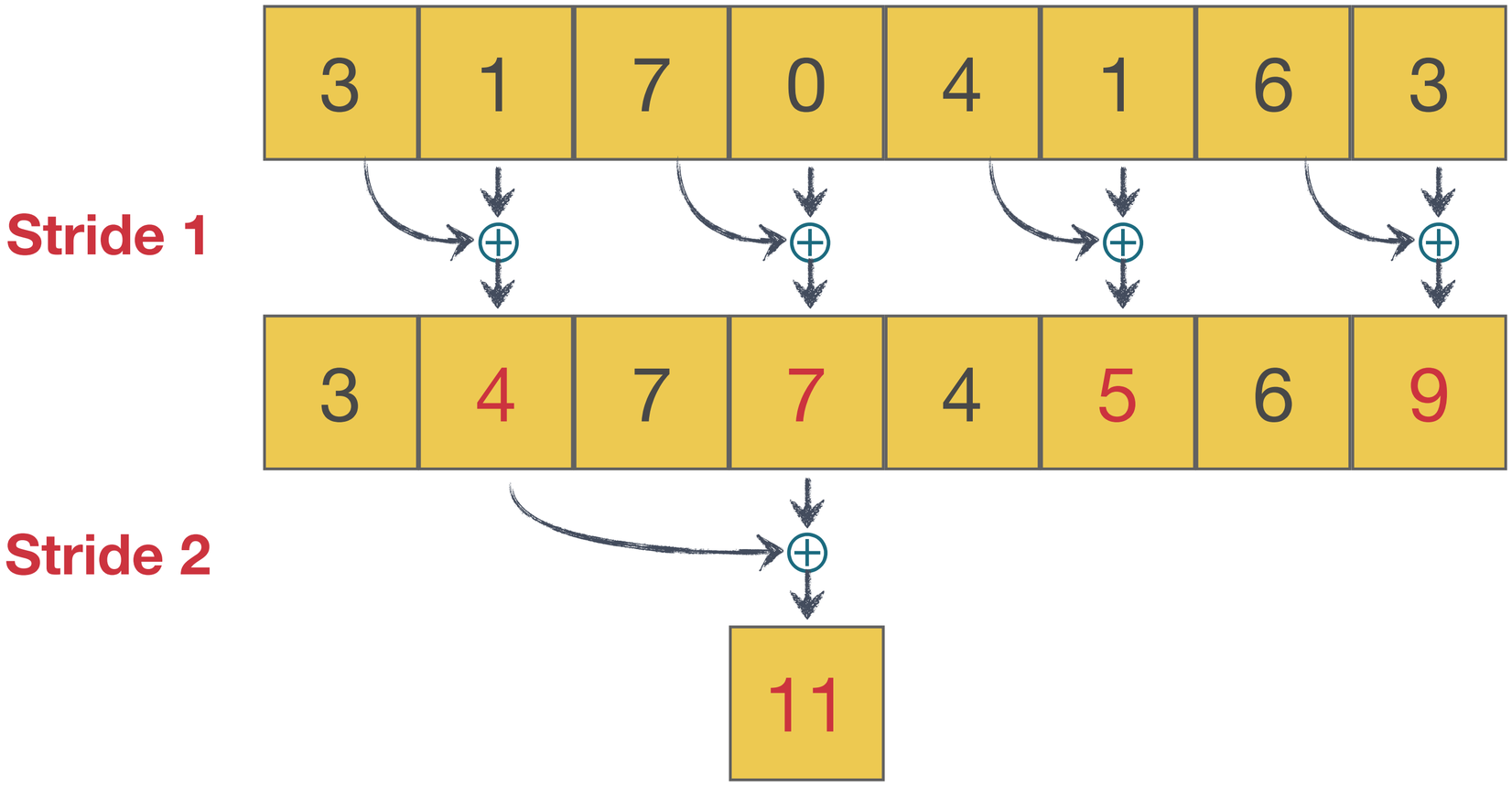


Stride 2

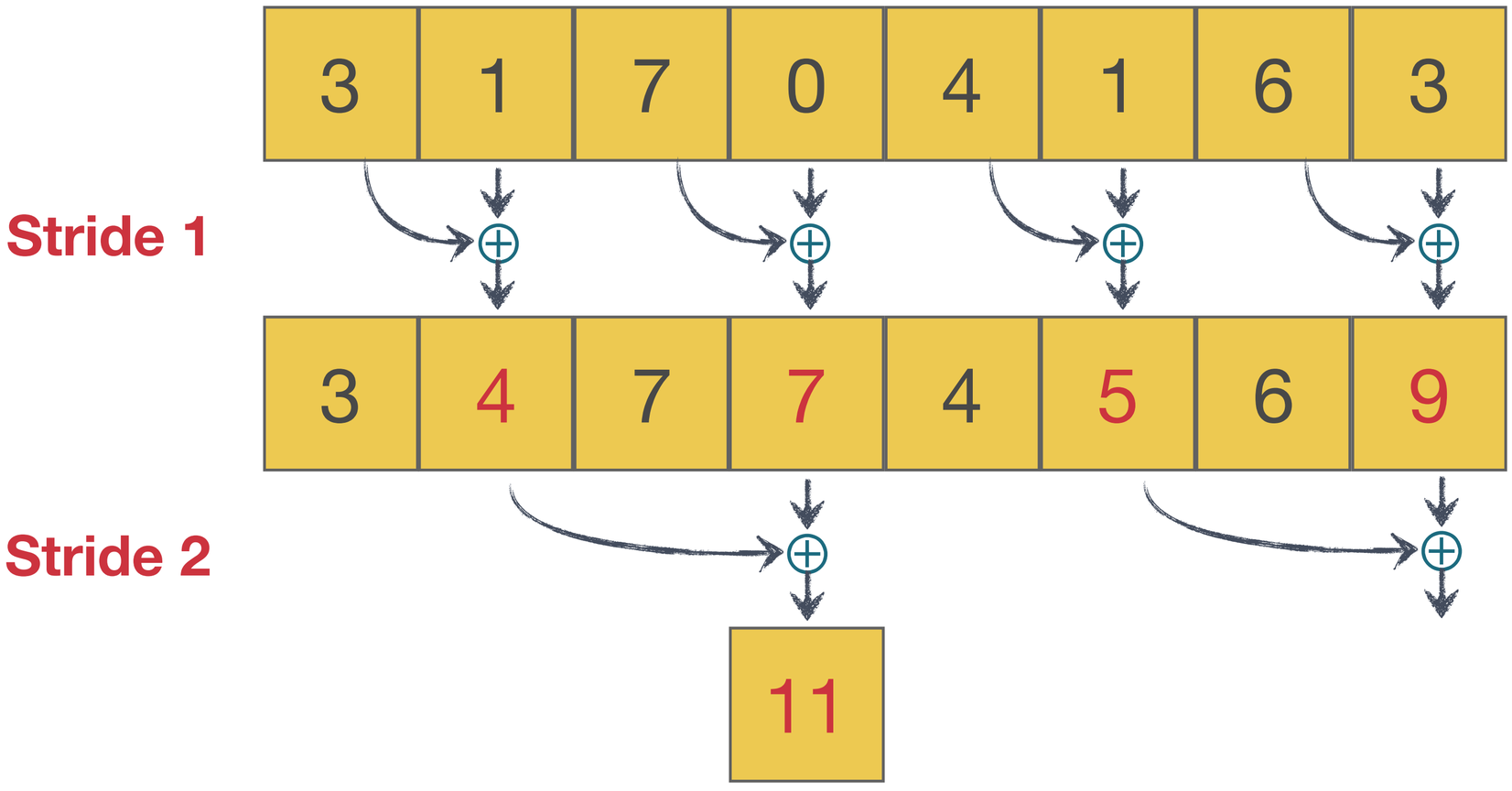
Parallel Scan: Reduction



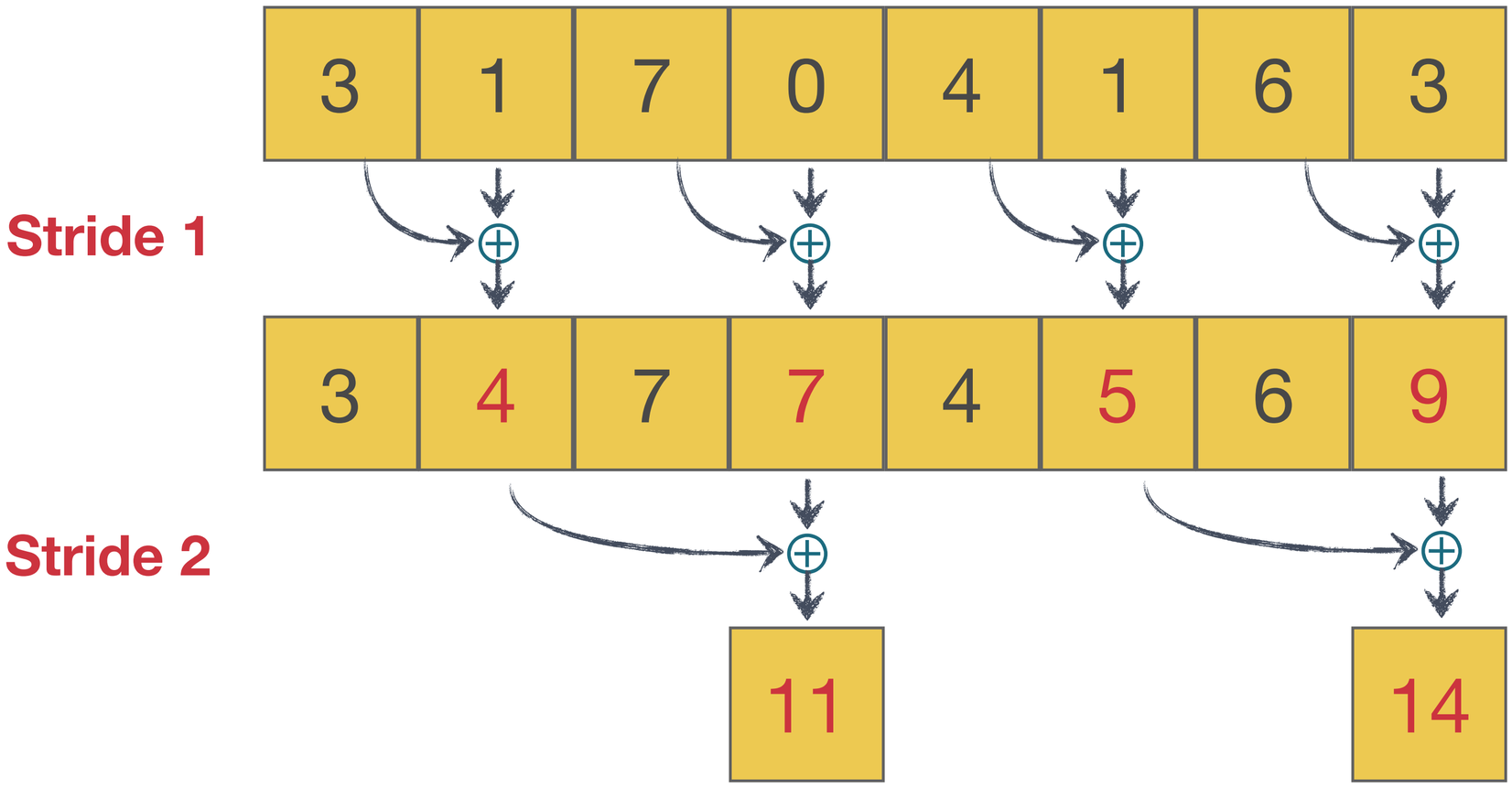
Parallel Scan: Reduction



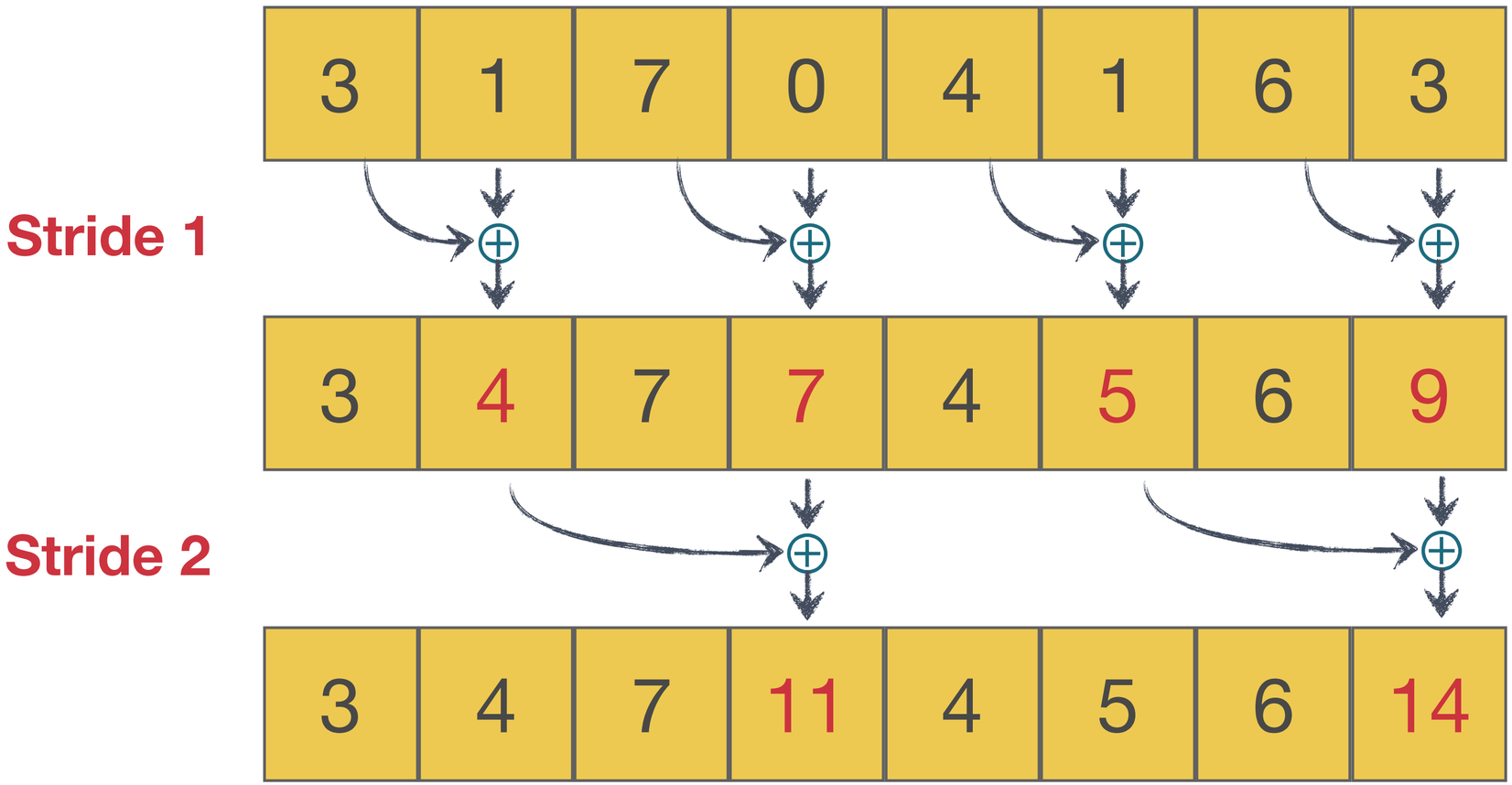
Parallel Scan: Reduction



Parallel Scan: Reduction

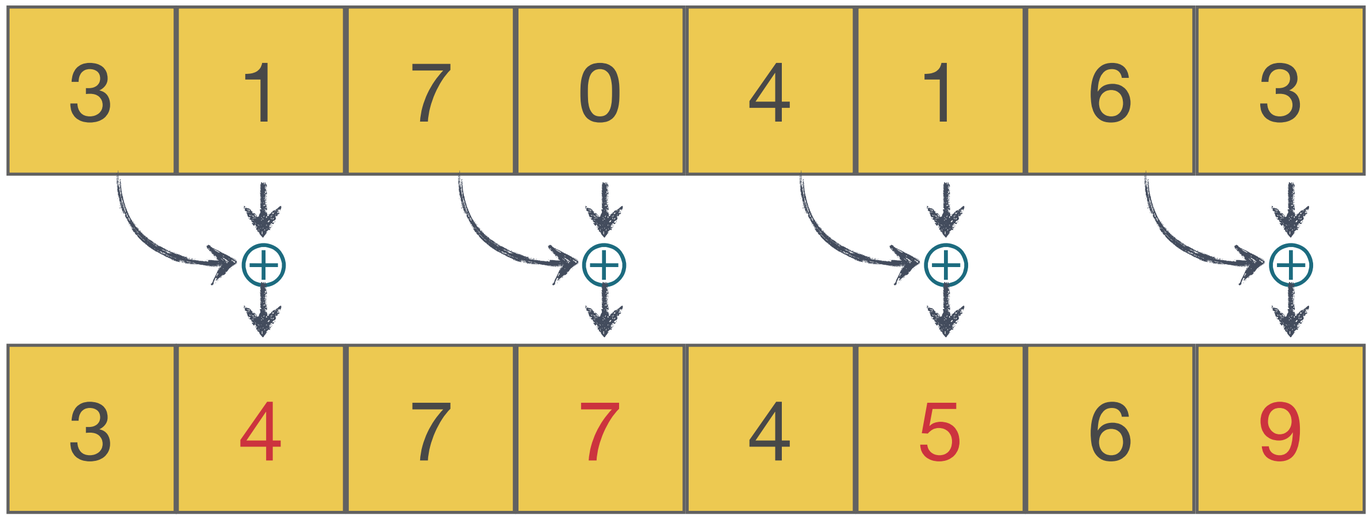


Parallel Scan: Reduction

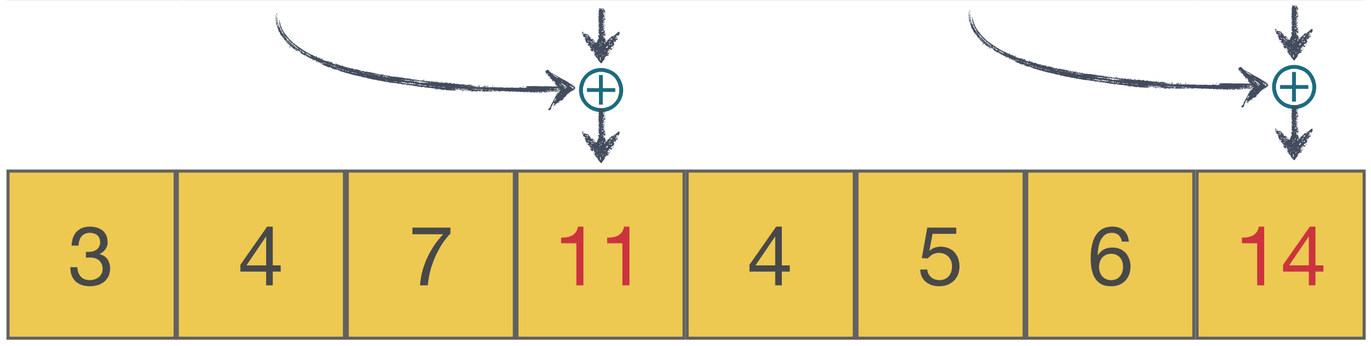


Parallel Scan: Reduction

Stride 1

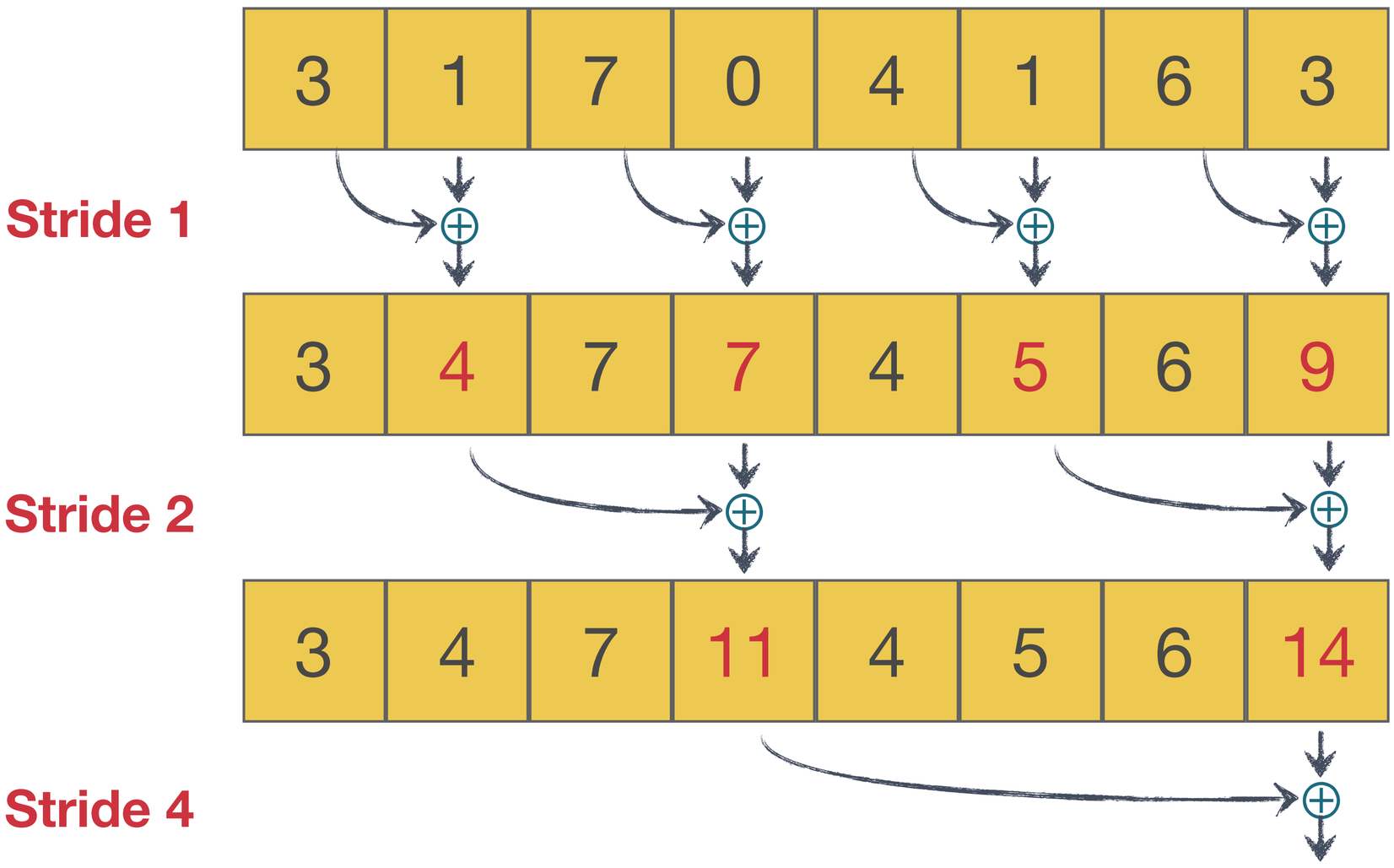


Stride 2

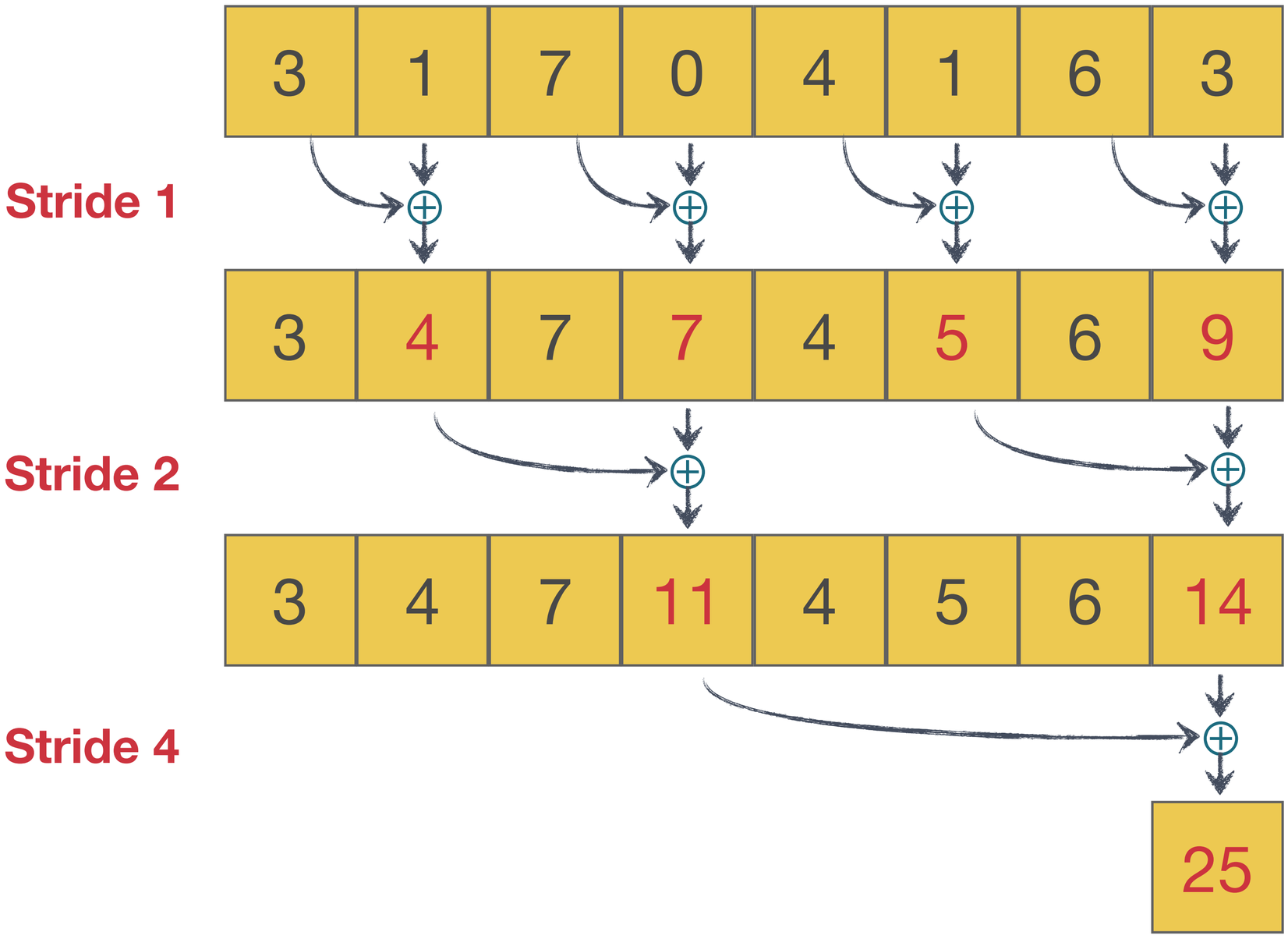


Stride 4

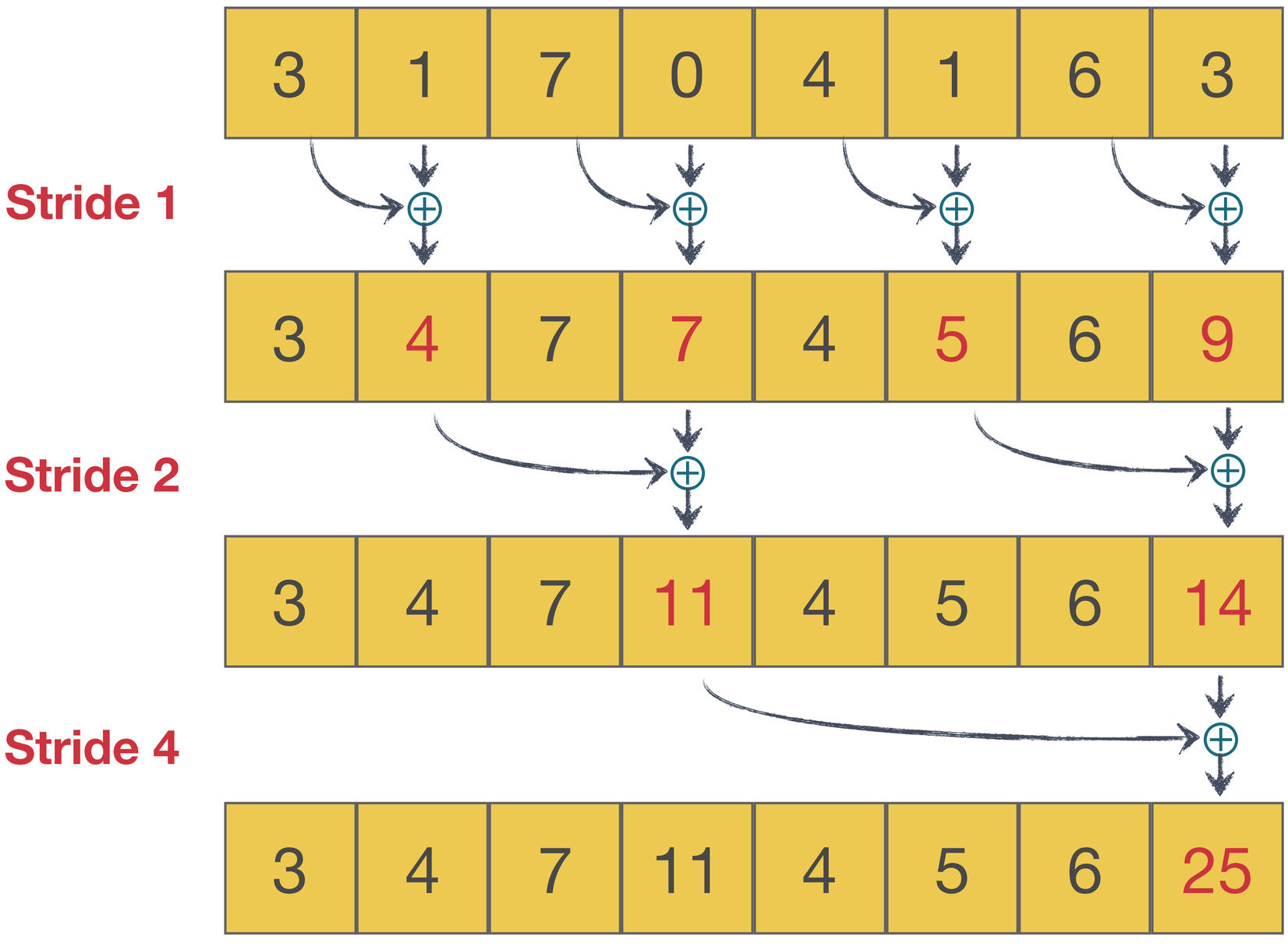
Parallel Scan: Reduction



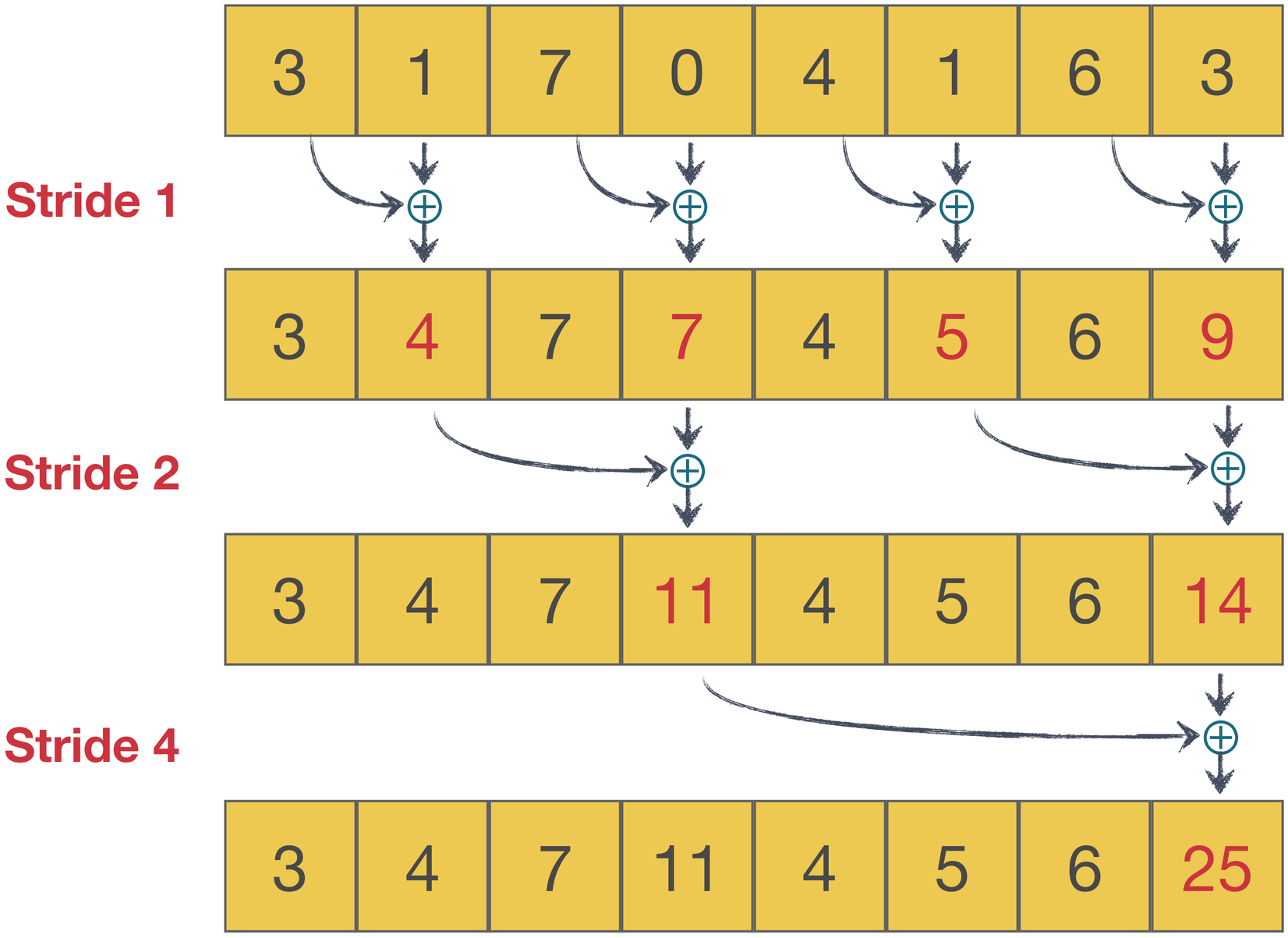
Parallel Scan: Reduction



Parallel Scan: Reduction

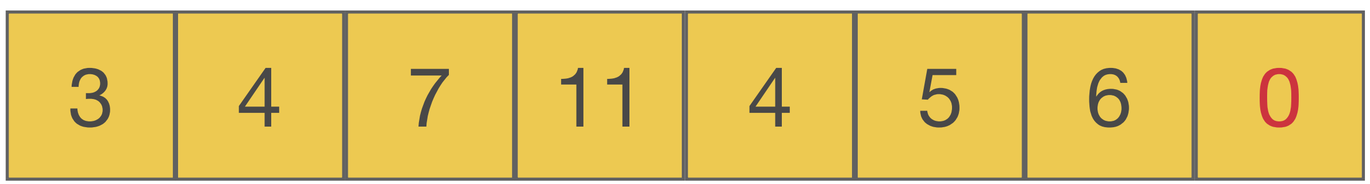


Parallel Scan: Reduction



Iterate log(n) times adding value *stride* elements away to its own value.

Parallel Scan: Zero the last element



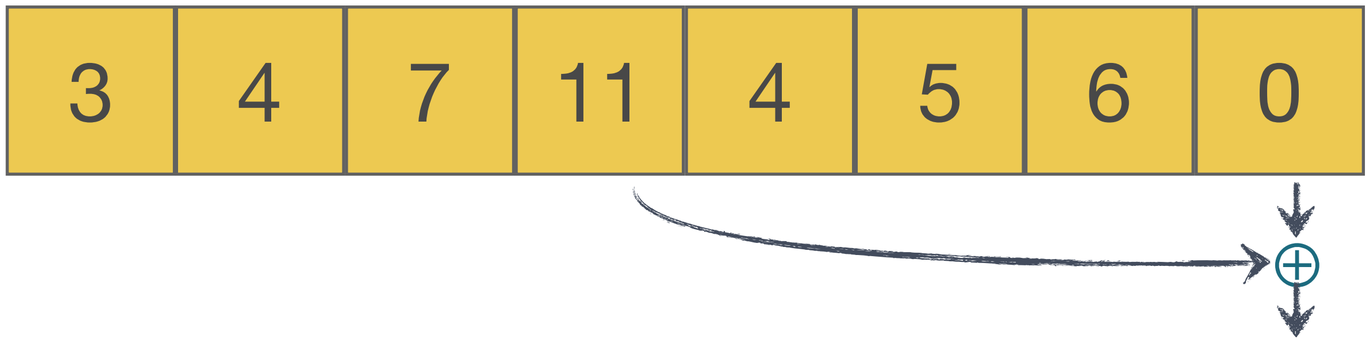
Since this is an exclusive scan,set the last element to zero. It will propagate back to the first element.

Parallel Scan: Post Scan

Parallel Scan: Post Scan

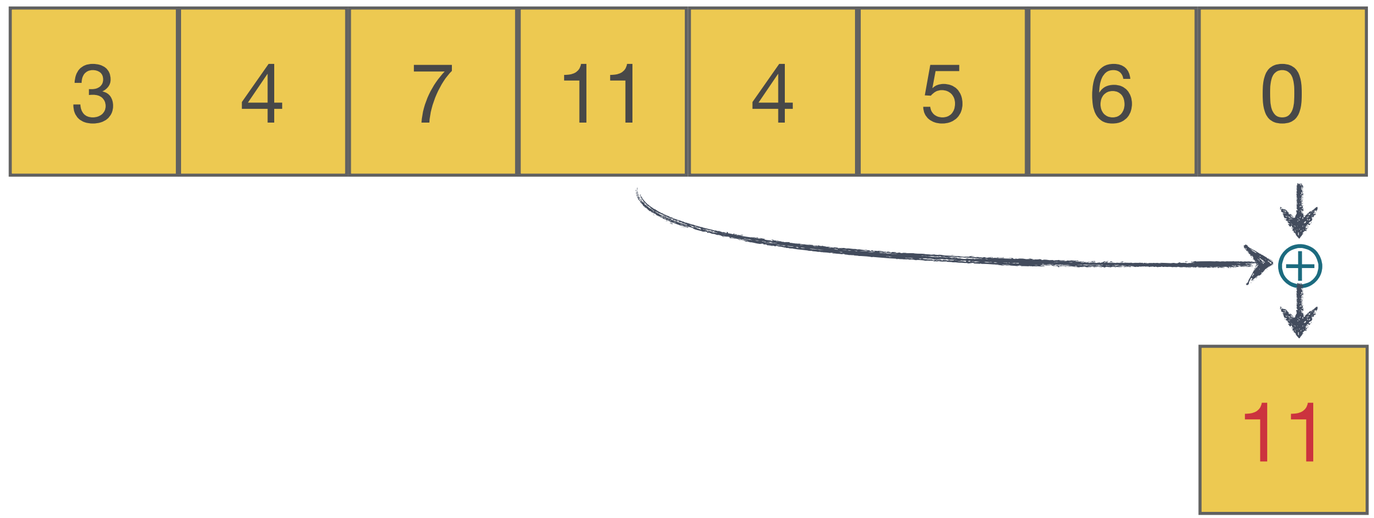
Stride 4

Parallel Scan: Post Scan



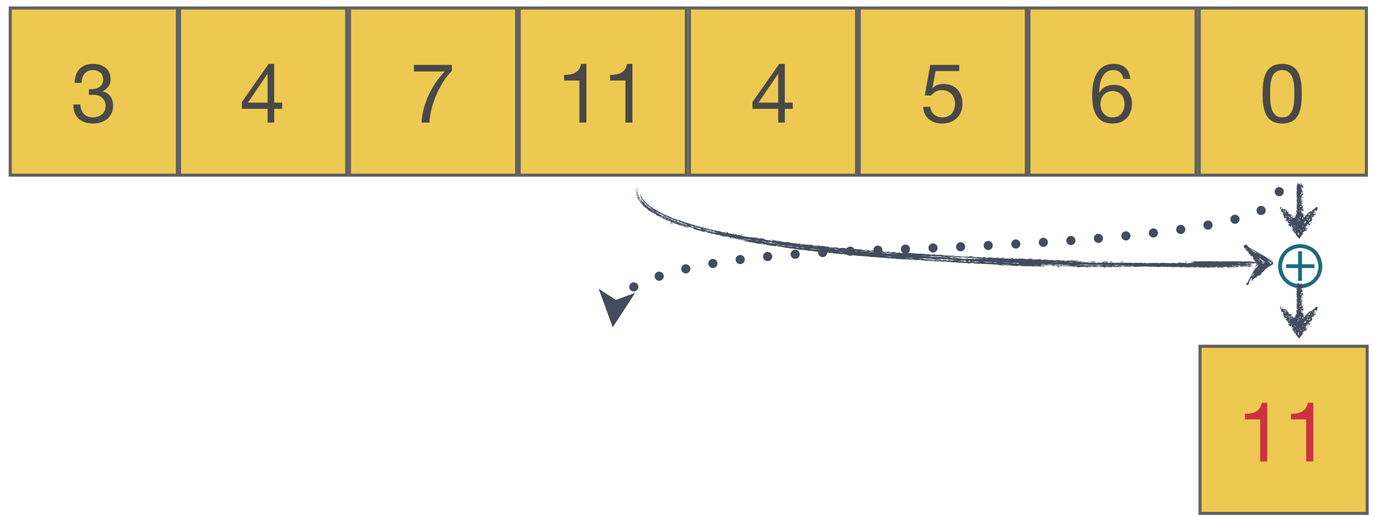
Stride 4

Parallel Scan: Post Scan



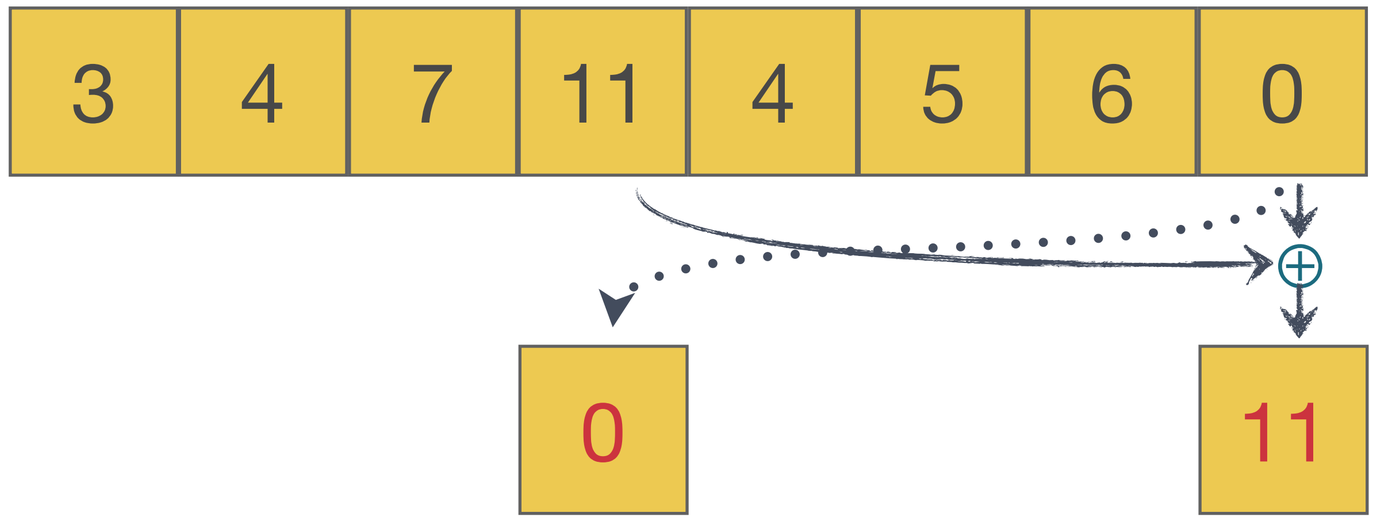
Stride 4

Parallel Scan: Post Scan



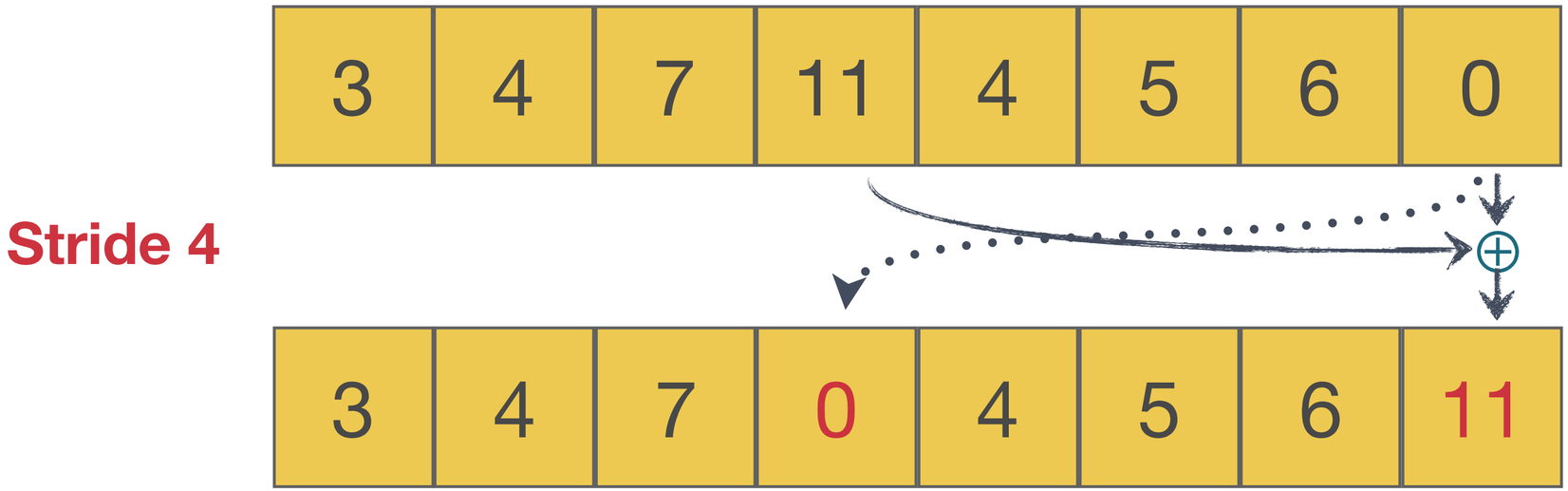
Stride 4

Parallel Scan: Post Scan

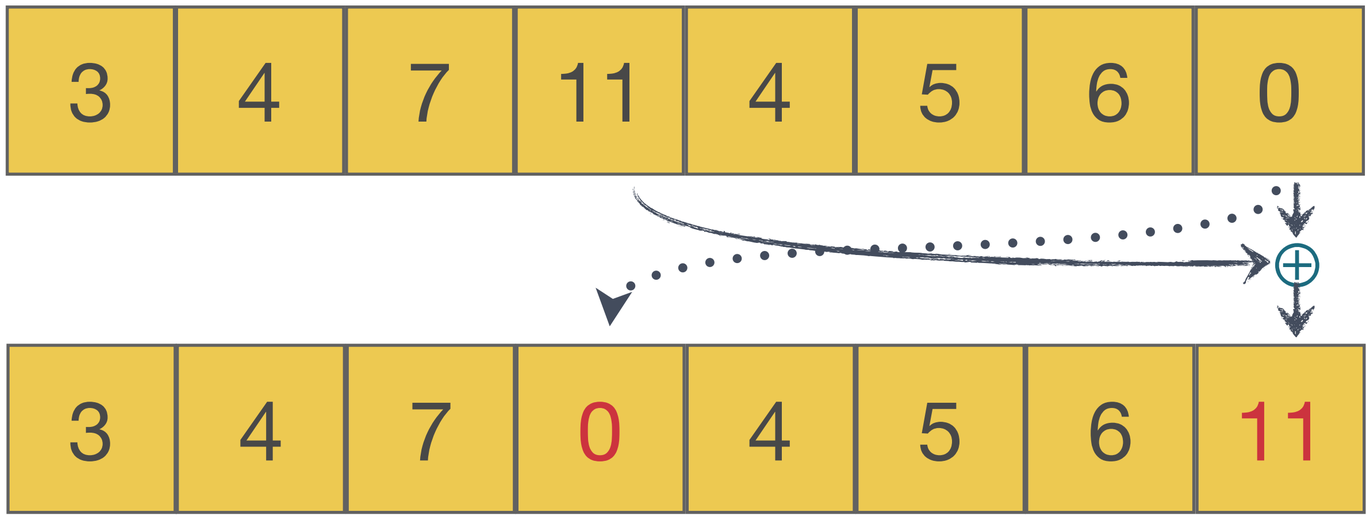


Stride 4

Parallel Scan: Post Scan

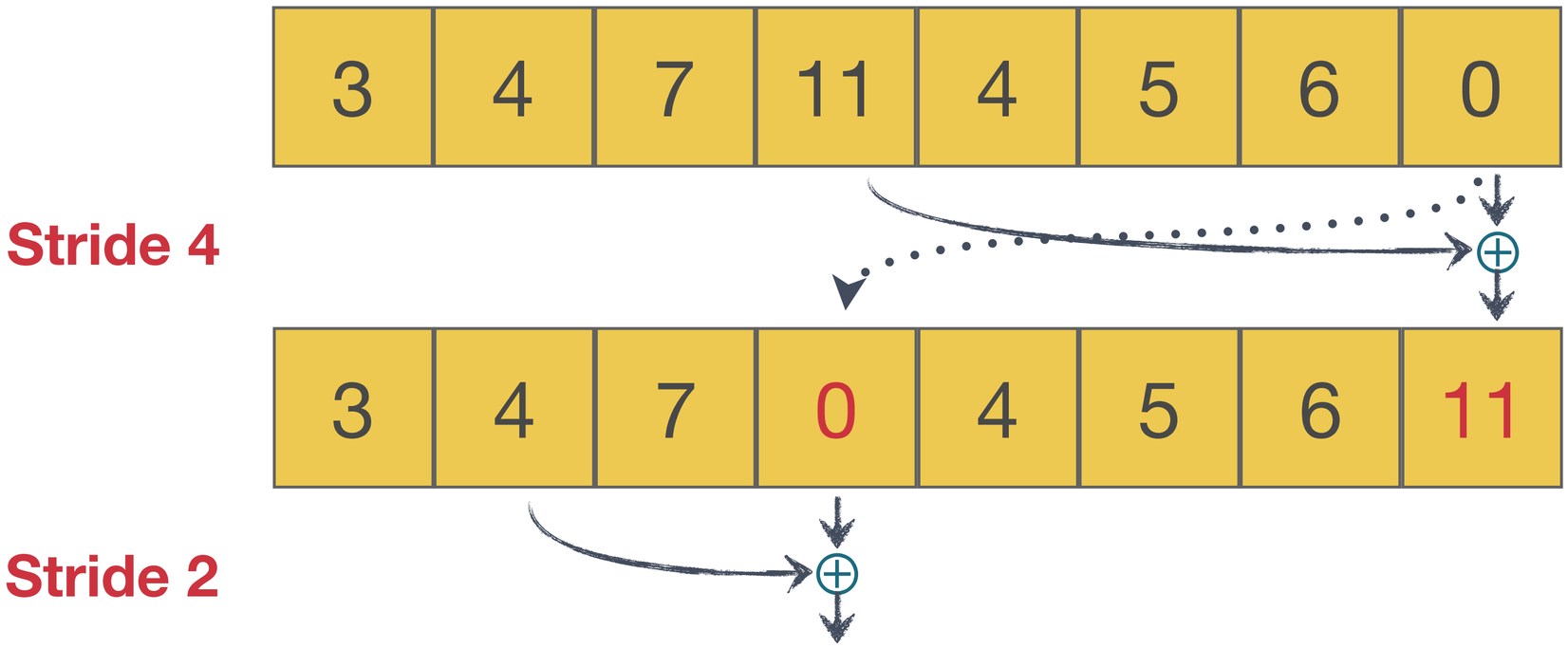


Parallel Scan: Post Scan

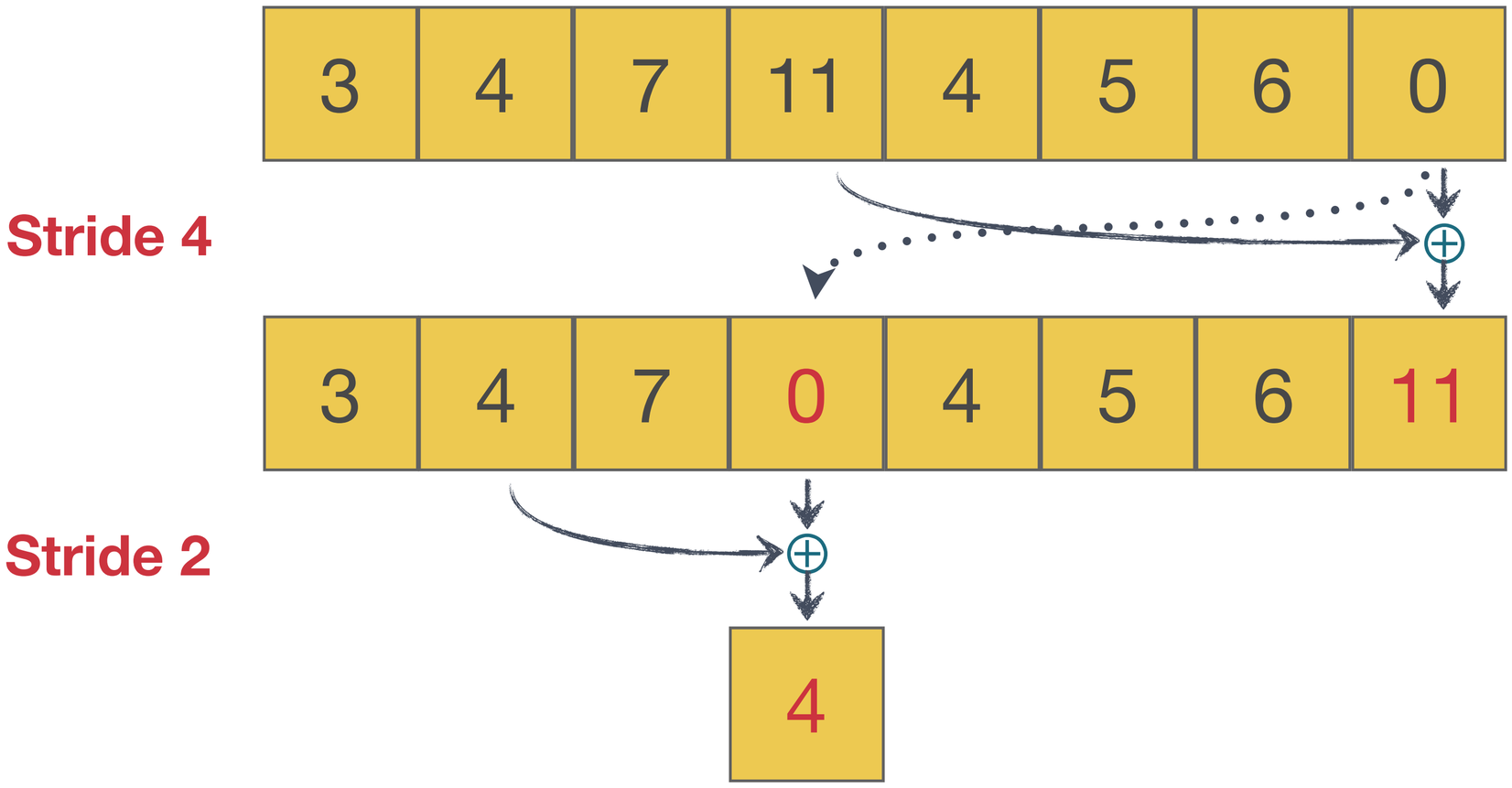


Stride 2

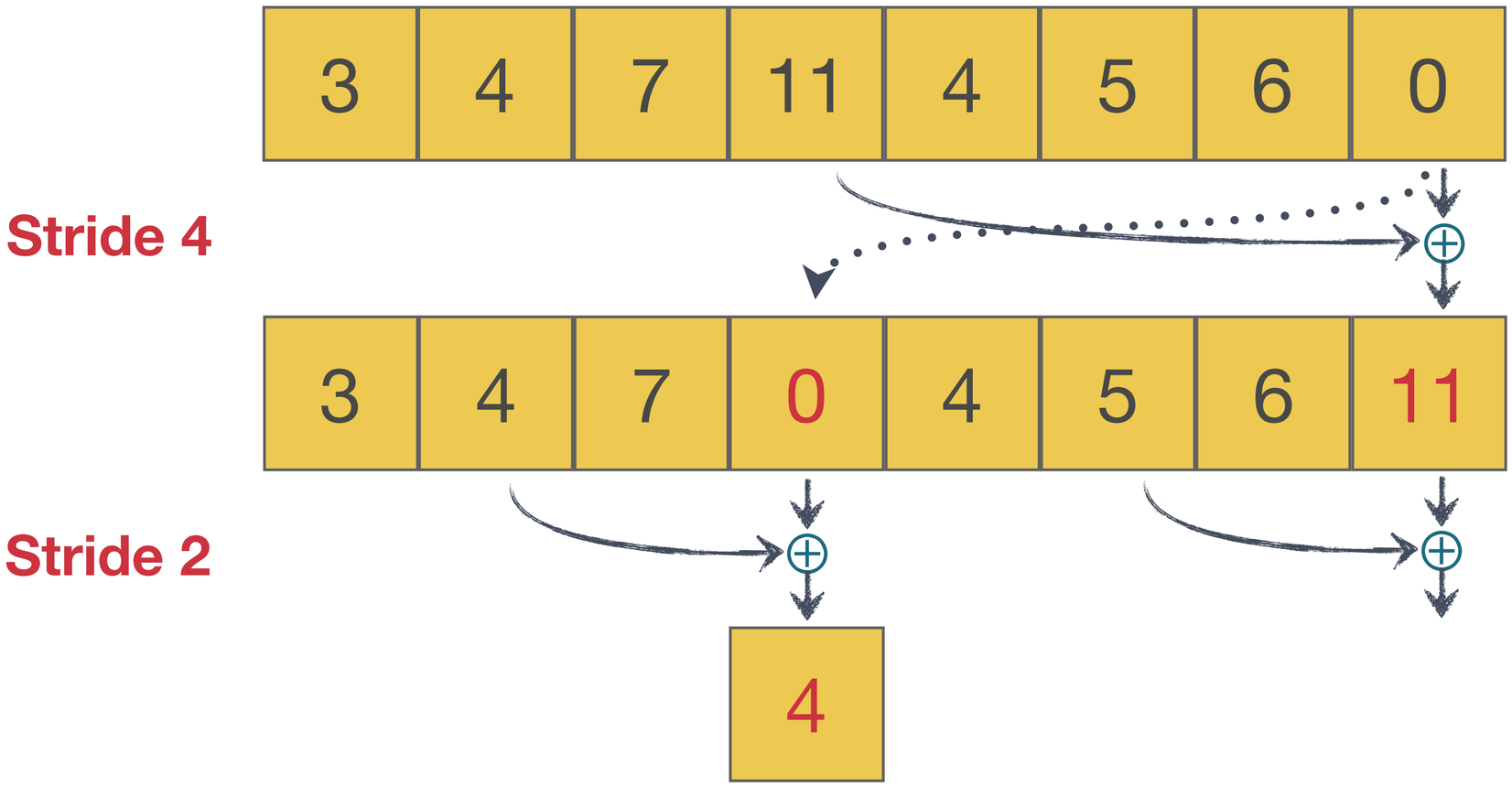
Parallel Scan: Post Scan



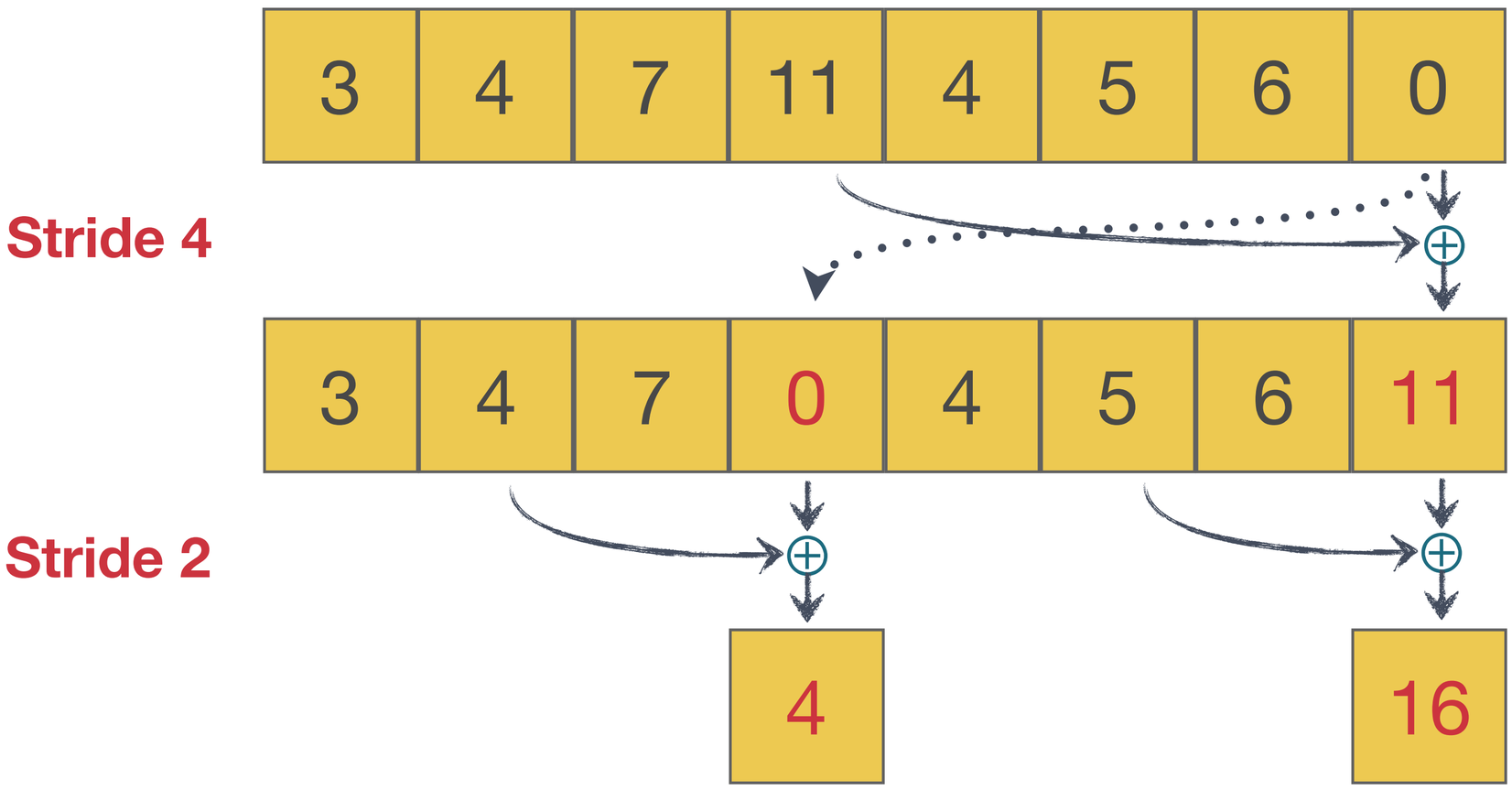
Parallel Scan: Post Scan



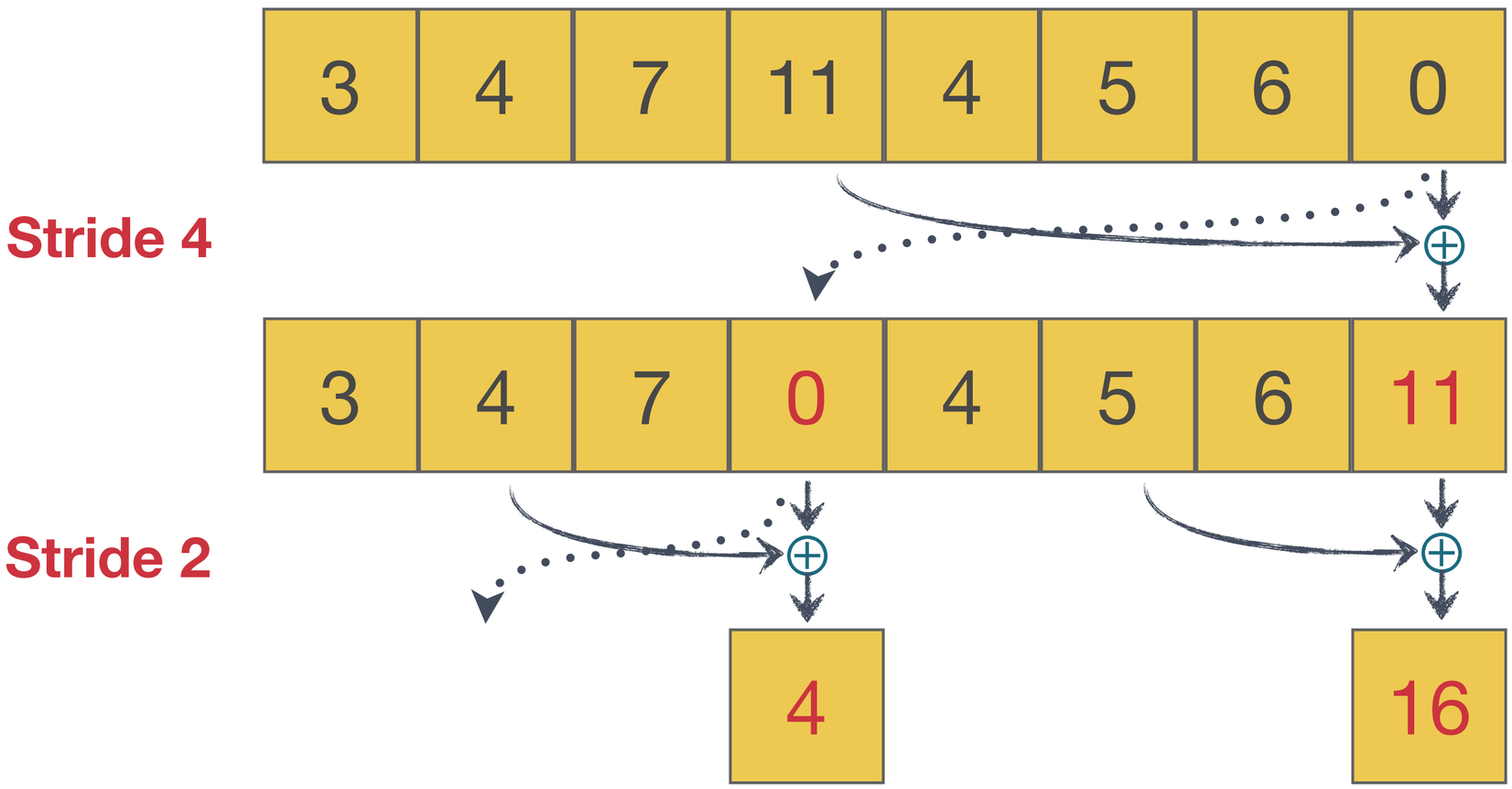
Parallel Scan: Post Scan



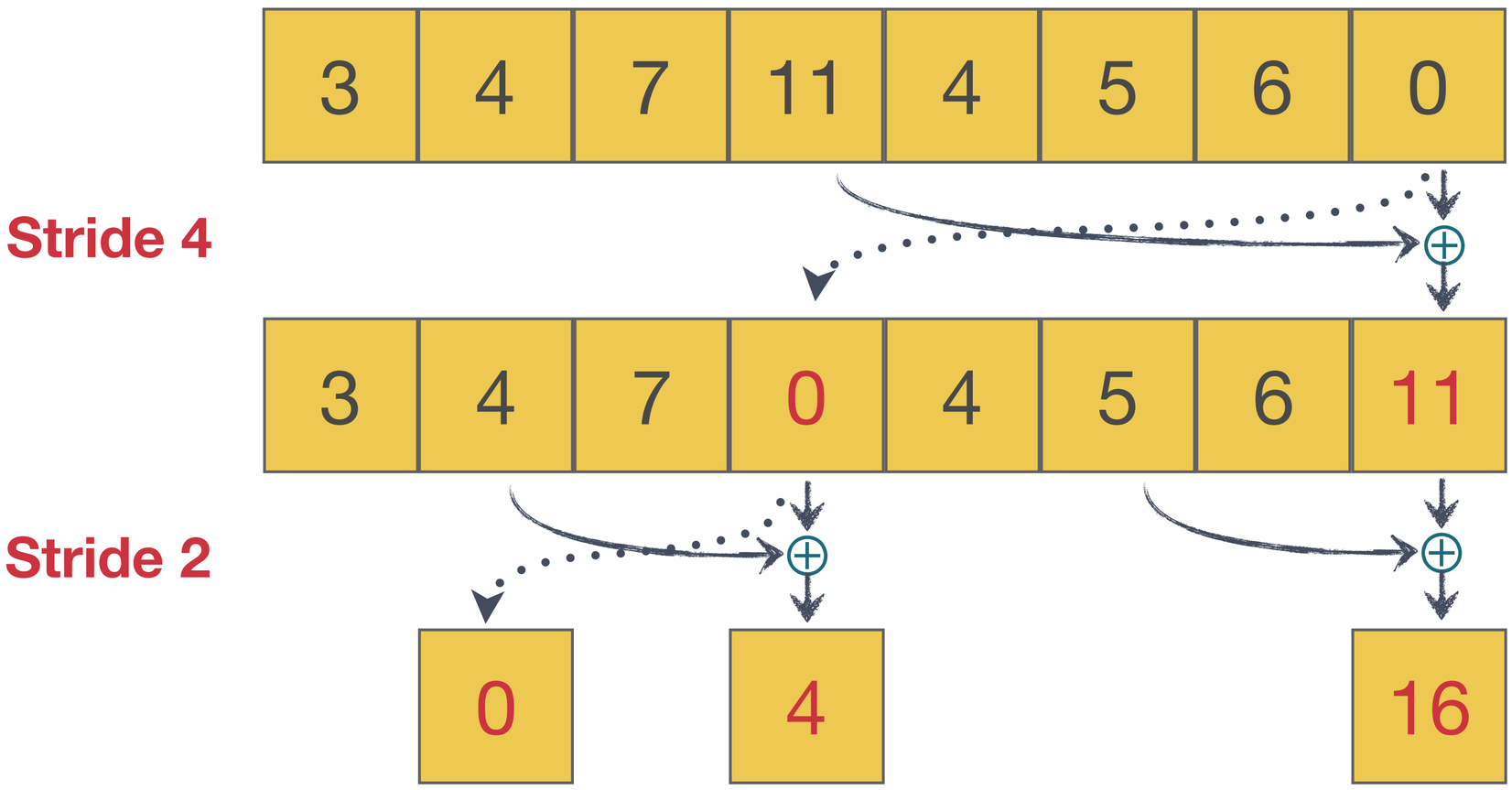
Parallel Scan: Post Scan



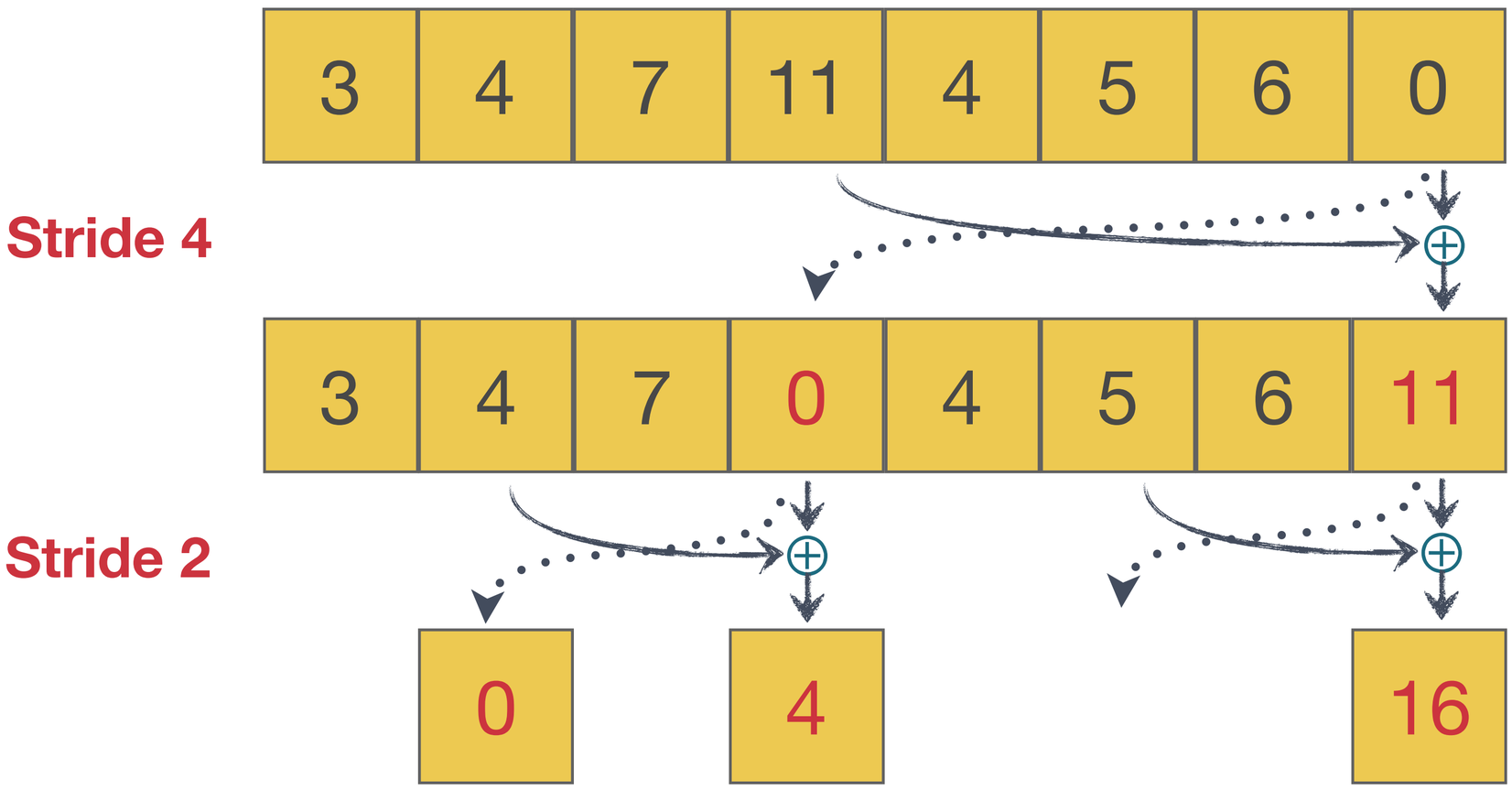
Parallel Scan: Post Scan



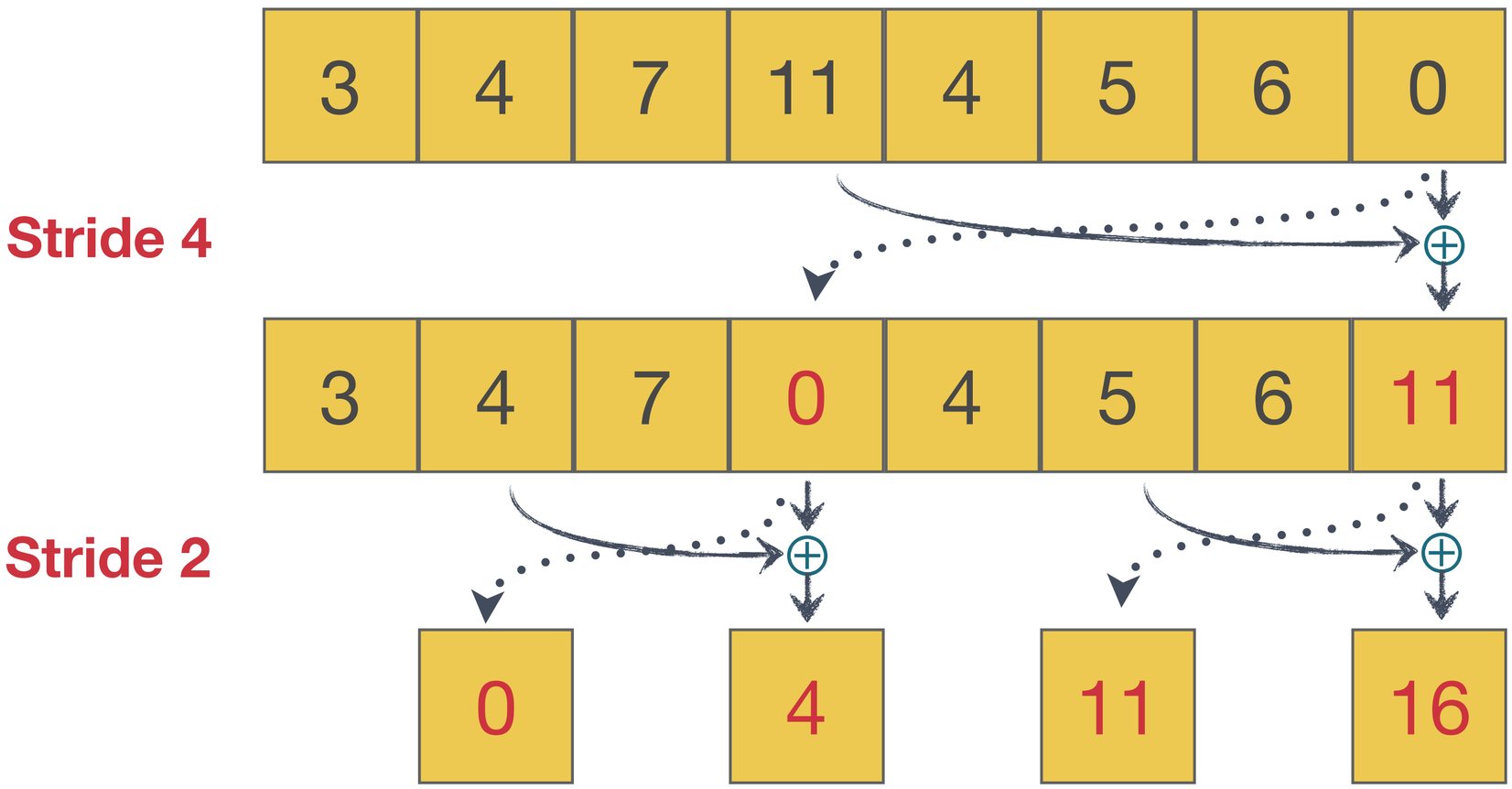
Parallel Scan: Post Scan



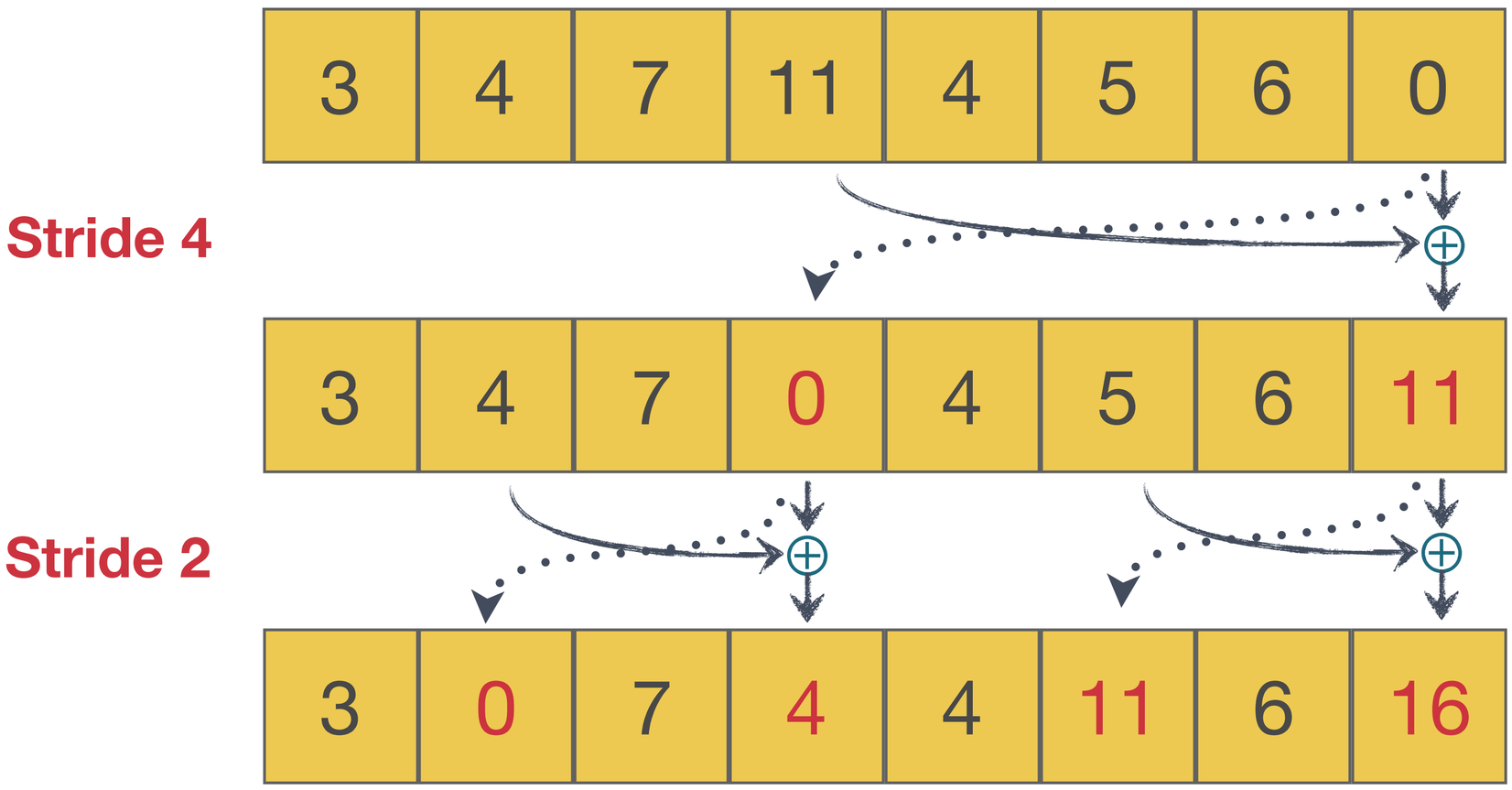
Parallel Scan: Post Scan



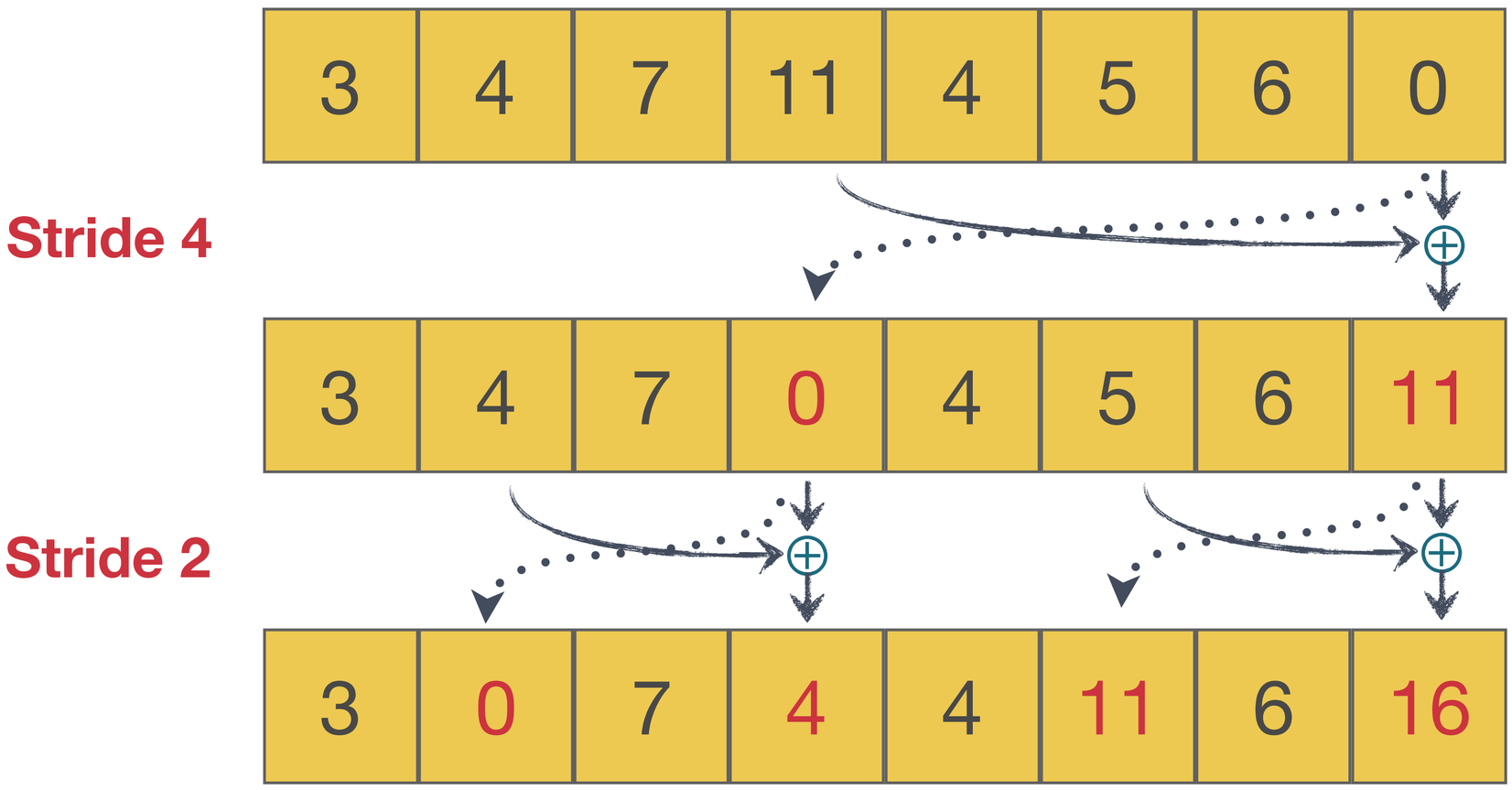
Parallel Scan: Post Scan



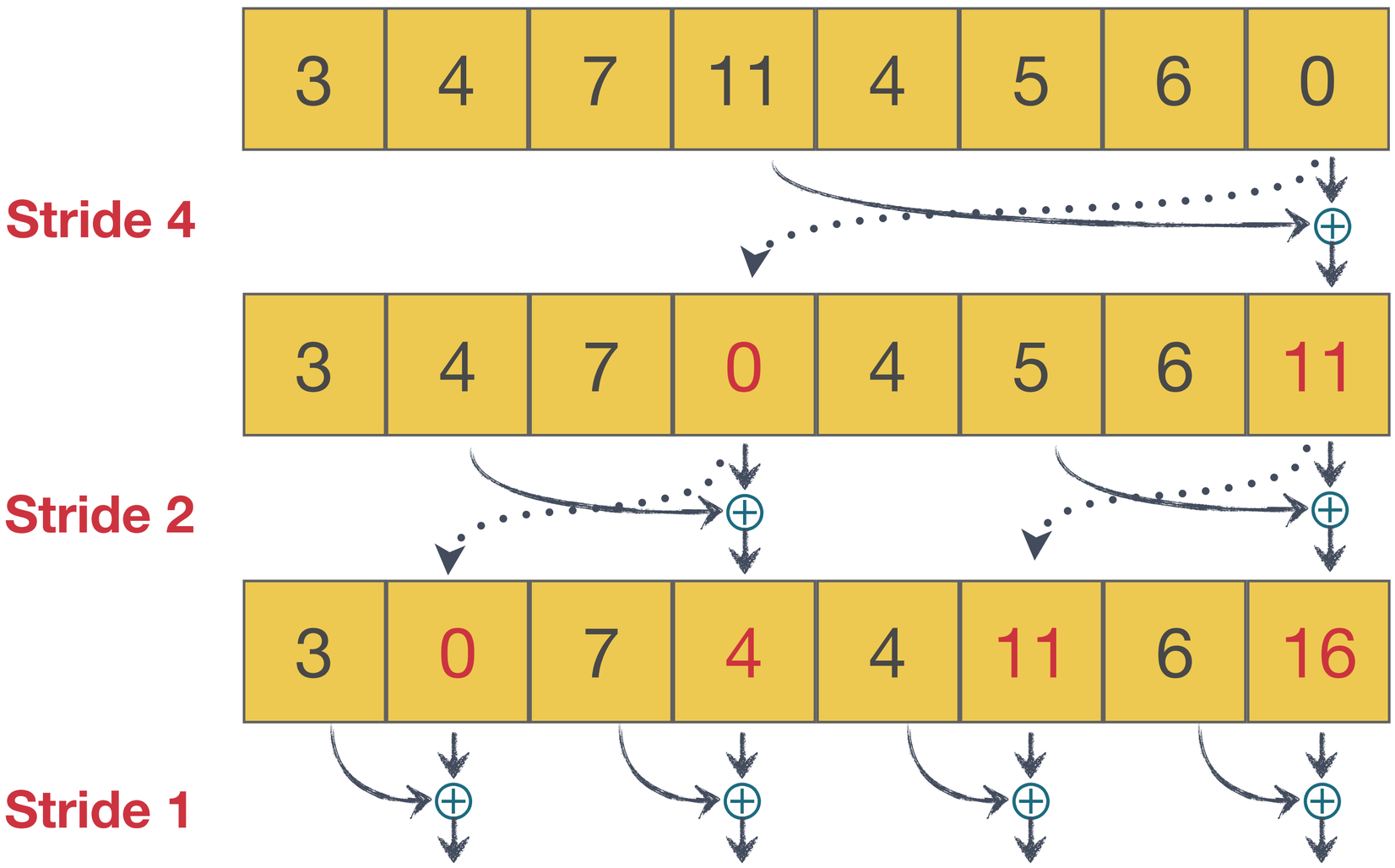
Parallel Scan: Post Scan



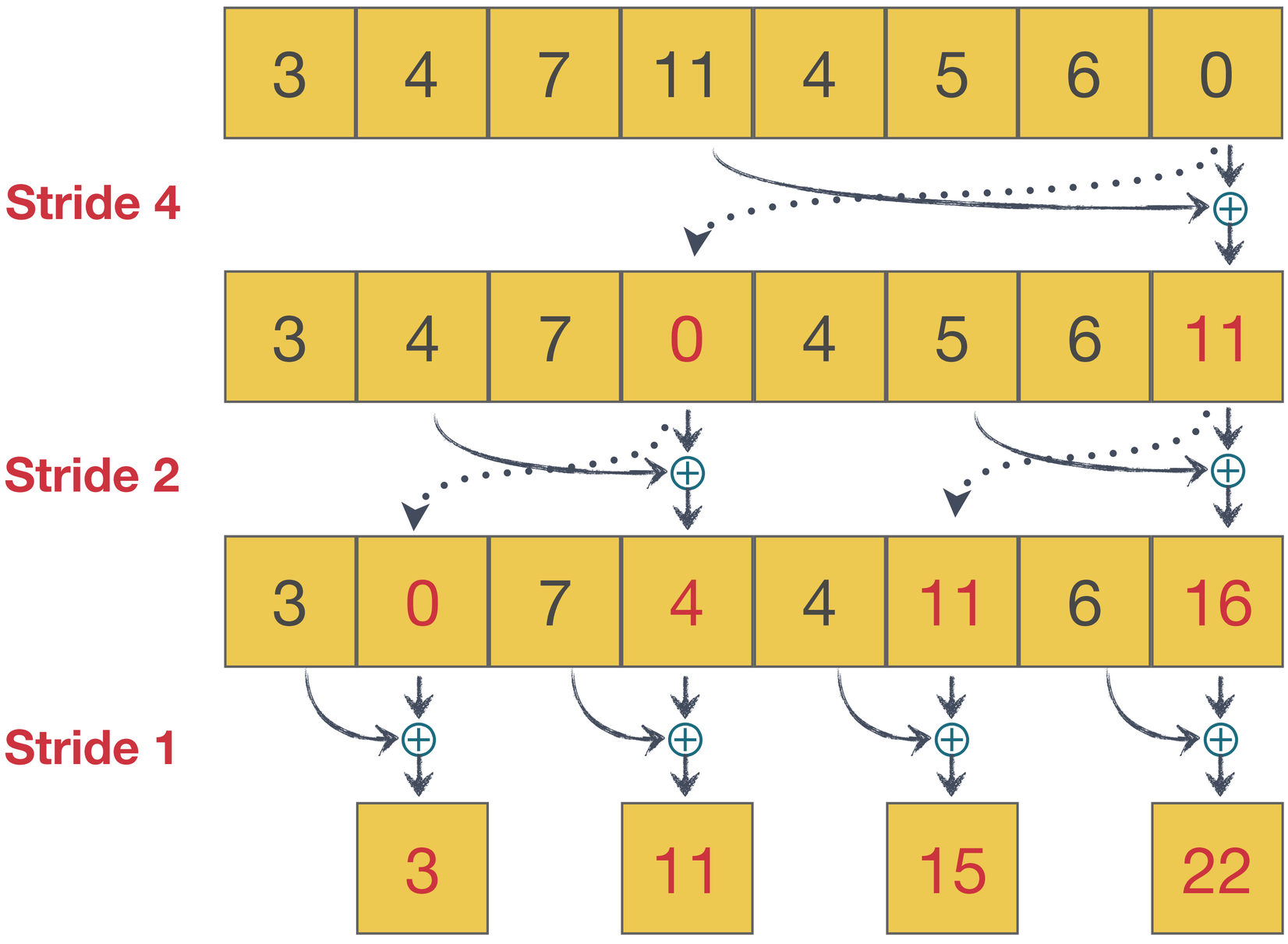
Parallel Scan: Post Scan



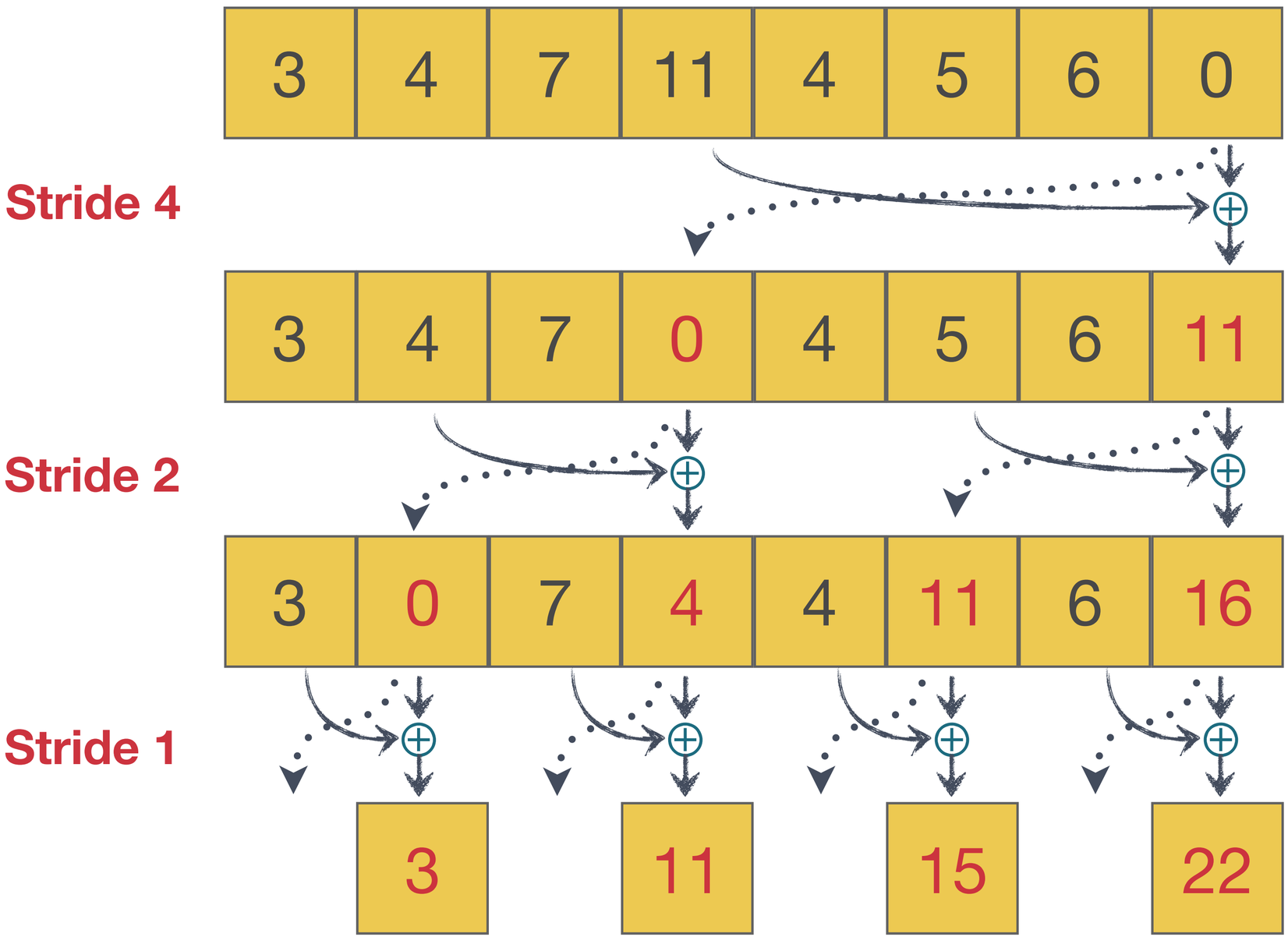
Parallel Scan: Post Scan



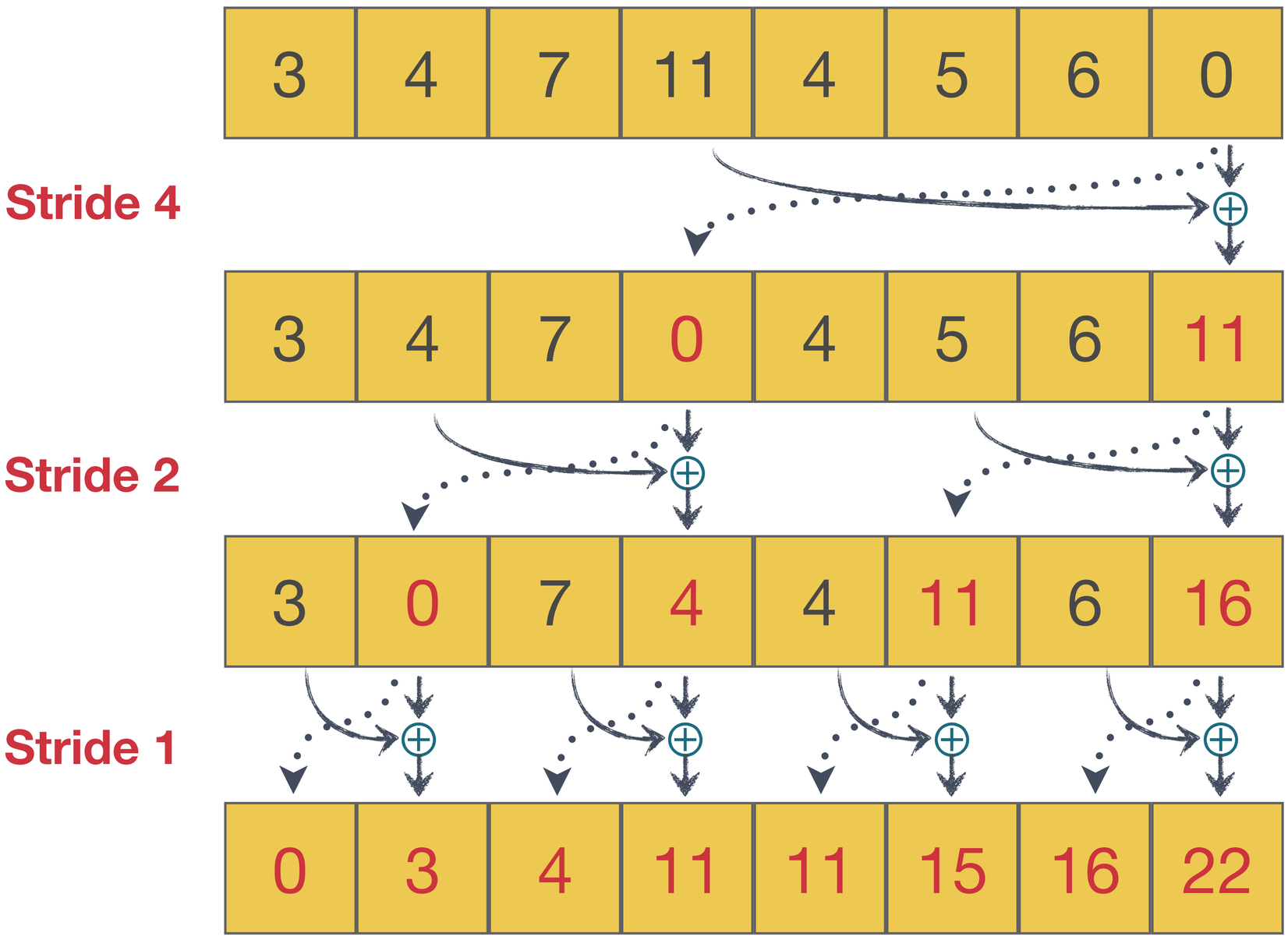
Parallel Scan: Post Scan



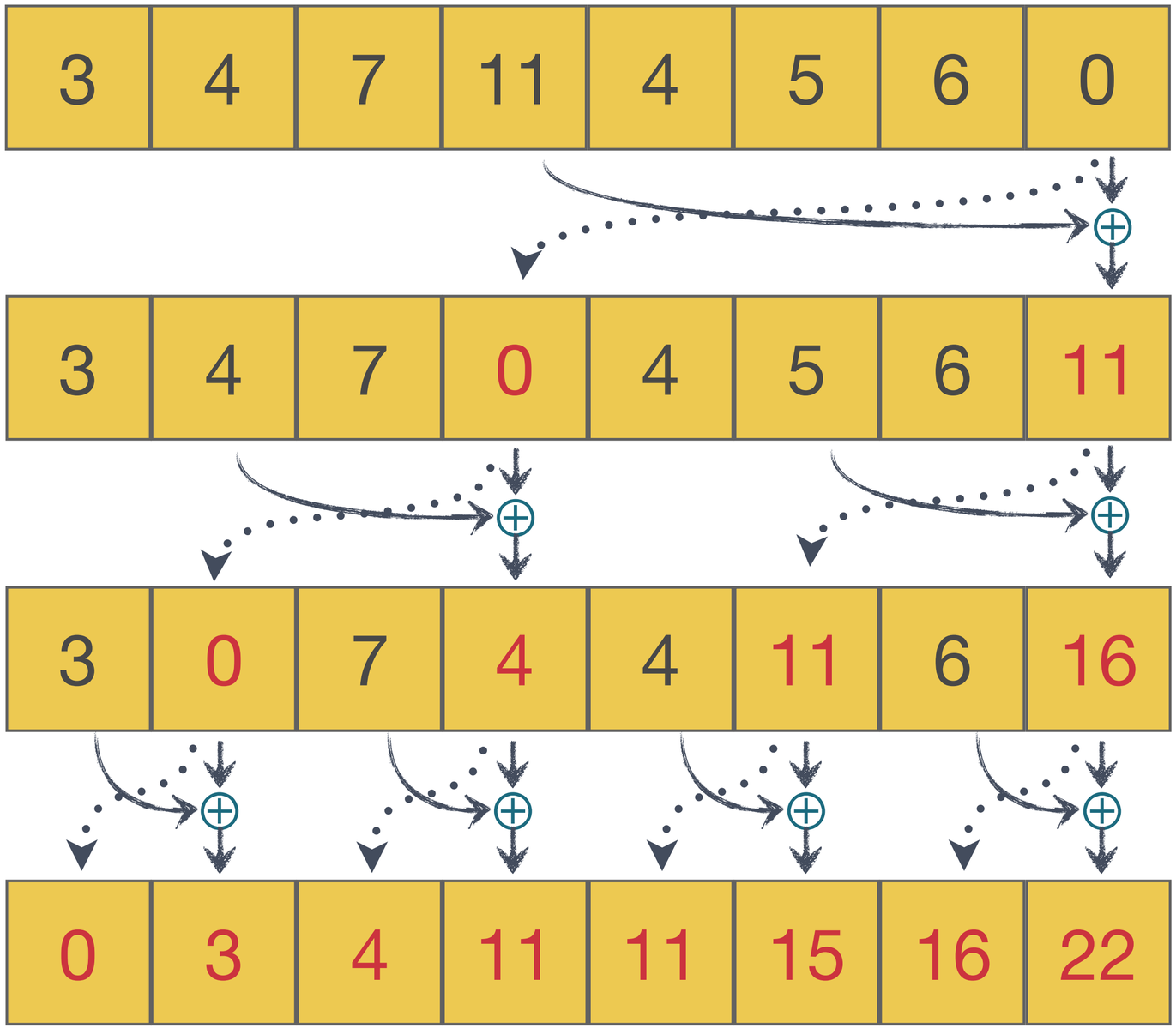
Parallel Scan: Post Scan



Parallel Scan: Post Scan



Parallel Scan: Post Scan



Iterate log(n) times adding value *stride* elements away to its own value, and set the value *stride* elements away to its previous value.

$$W_{scan}(n) = \Theta(n)$$

$$D_{scan}(n) = \Theta(\log n)$$

$$p \equiv \frac{W}{D} = \Theta\left(\frac{n}{\log n}\right)$$