Programming Parallel Computers

Jukka Suomela · Aalto University · ppc.cs.aalto.fi

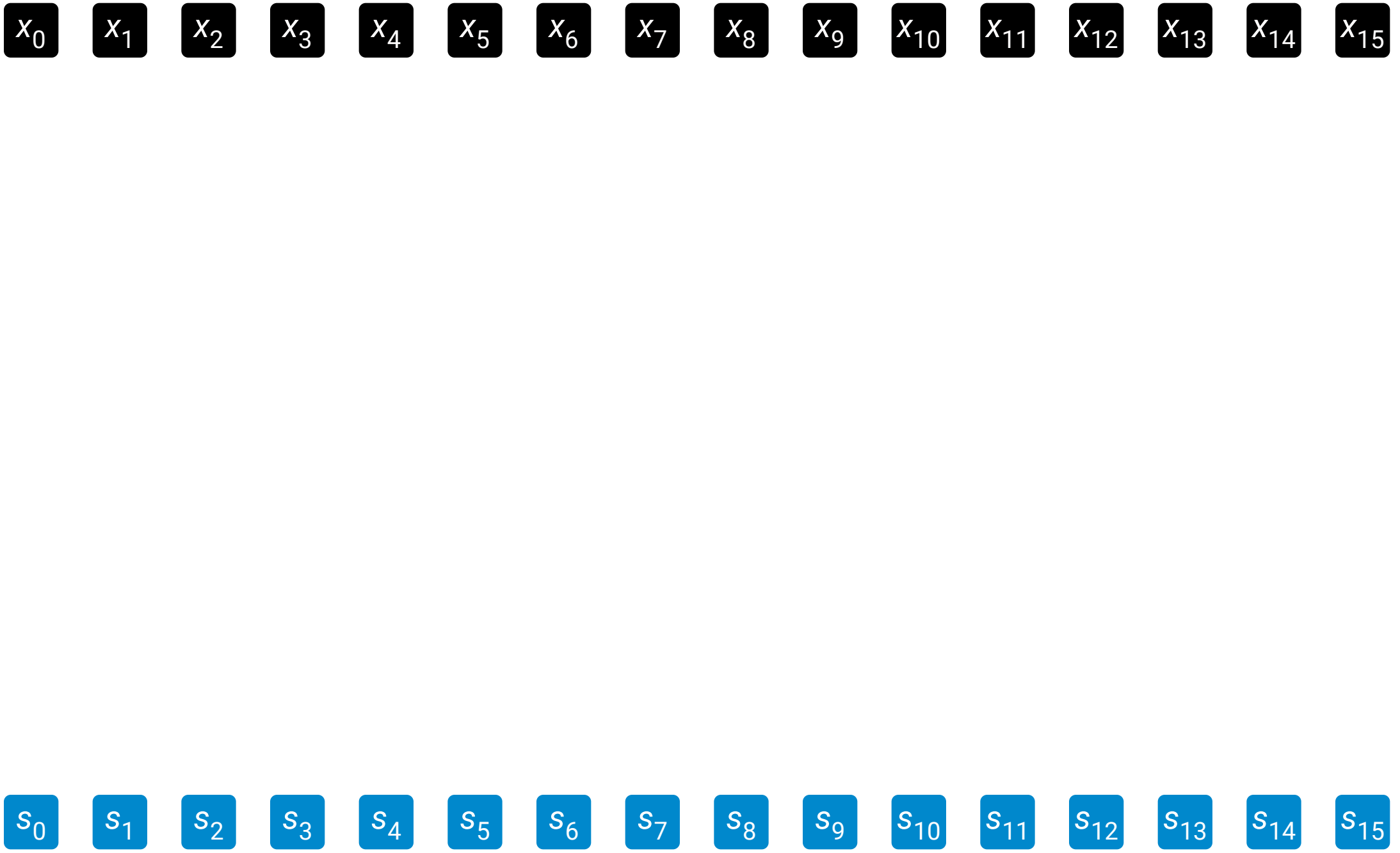
Part 6B:
Parallel prefix sum

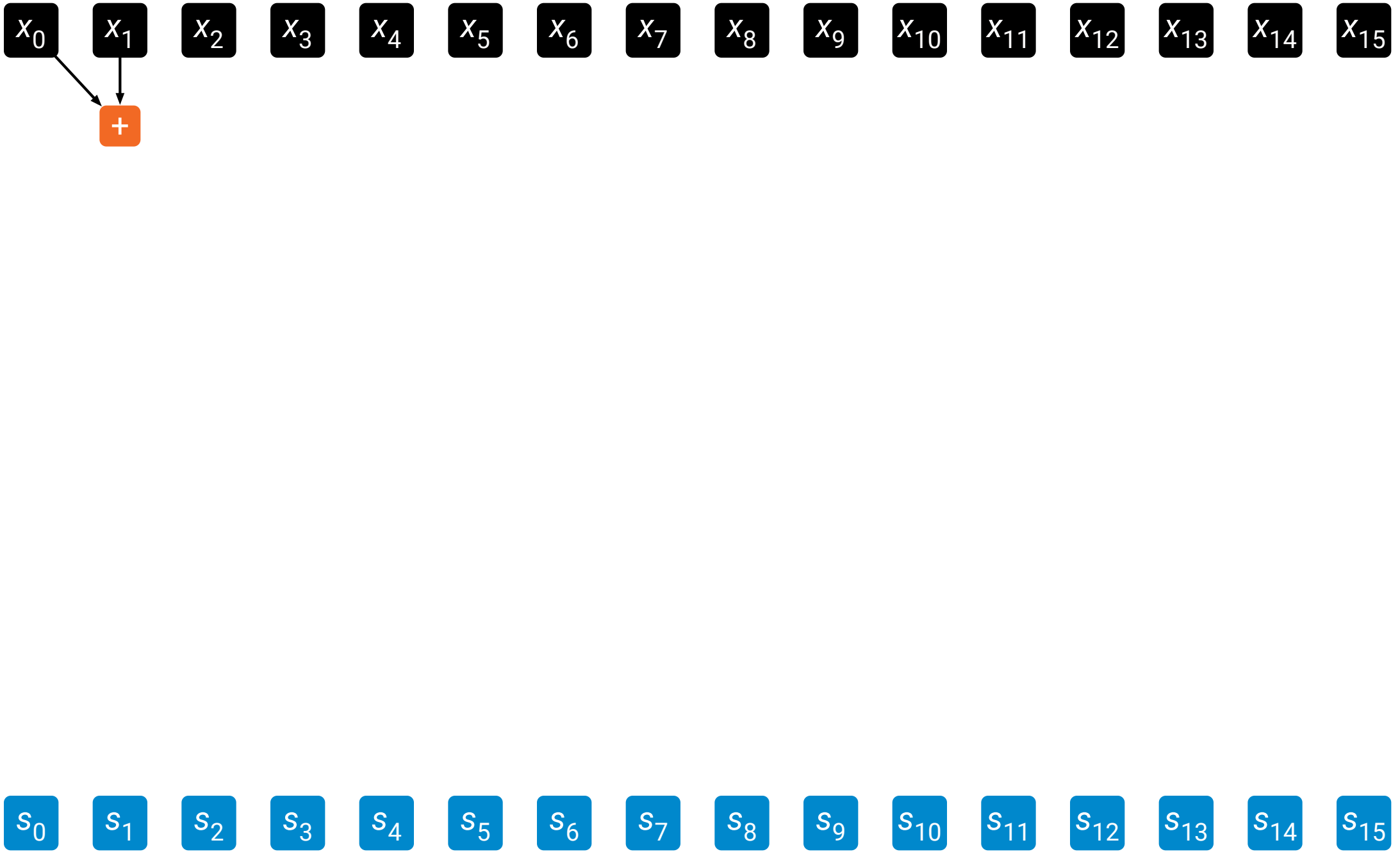
Prefix sum

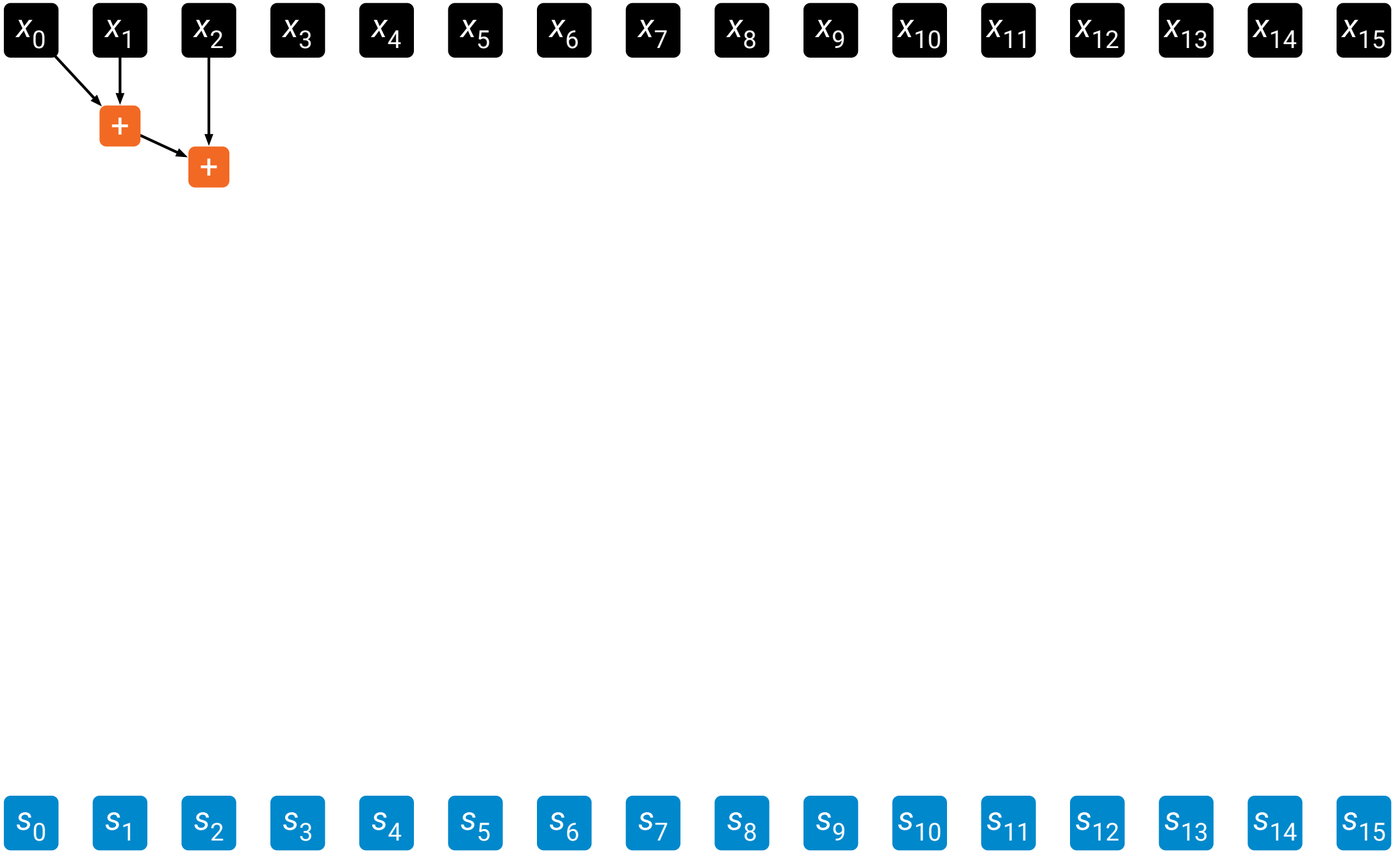
- Input: $x_0, x_1, \ldots, x_{n-1}$
- Output:
  - $s_0 = x_0$
  - $s_1 = x_0 + x_1$
  - $s_2 = x_0 + x_1 + x_2$

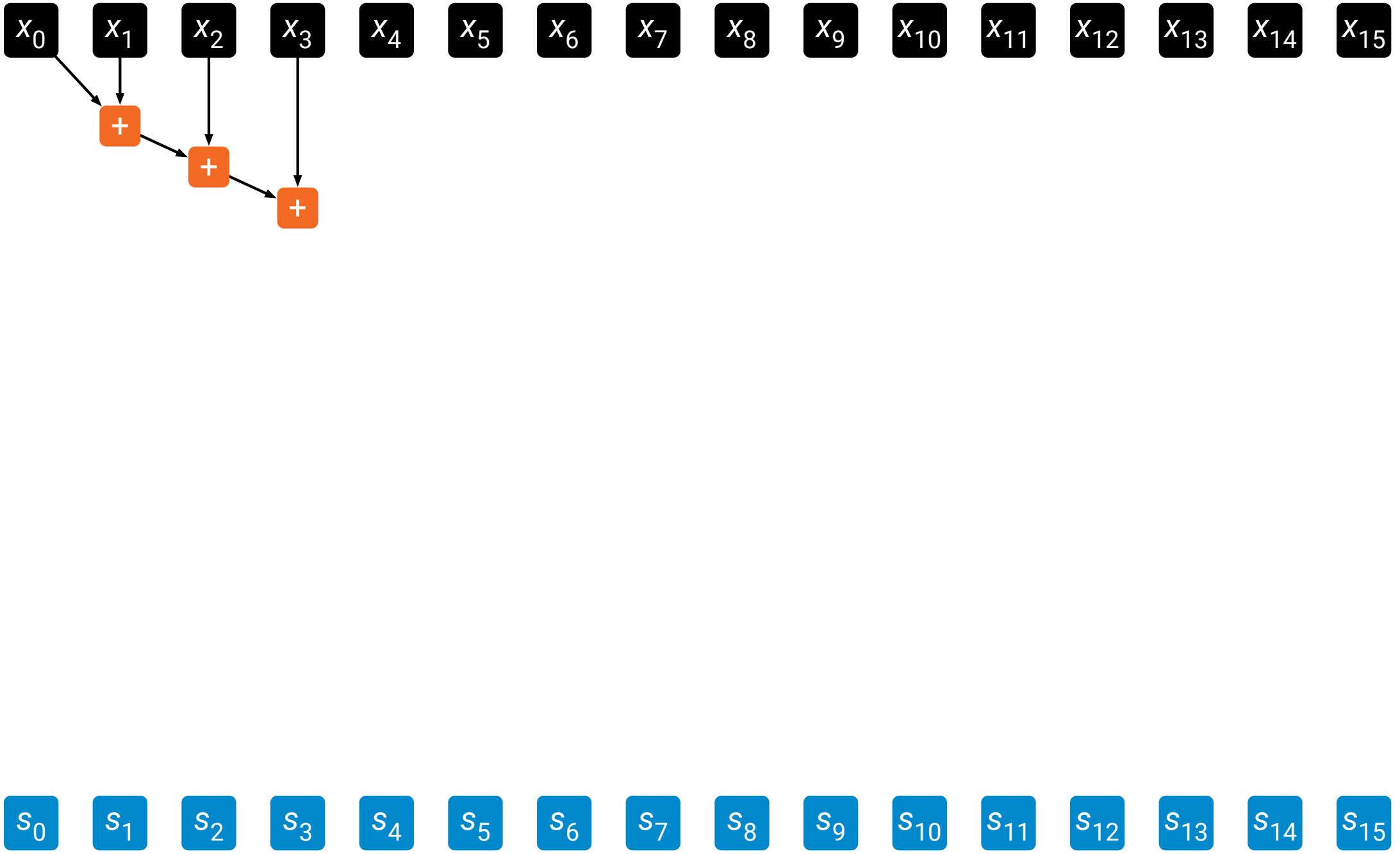
    …
  - $s_{n-1} = x_0 + x_1 + \ldots + x_{n-1}$
- Trivial sequential implementation
- Can be parallelized efficiently!

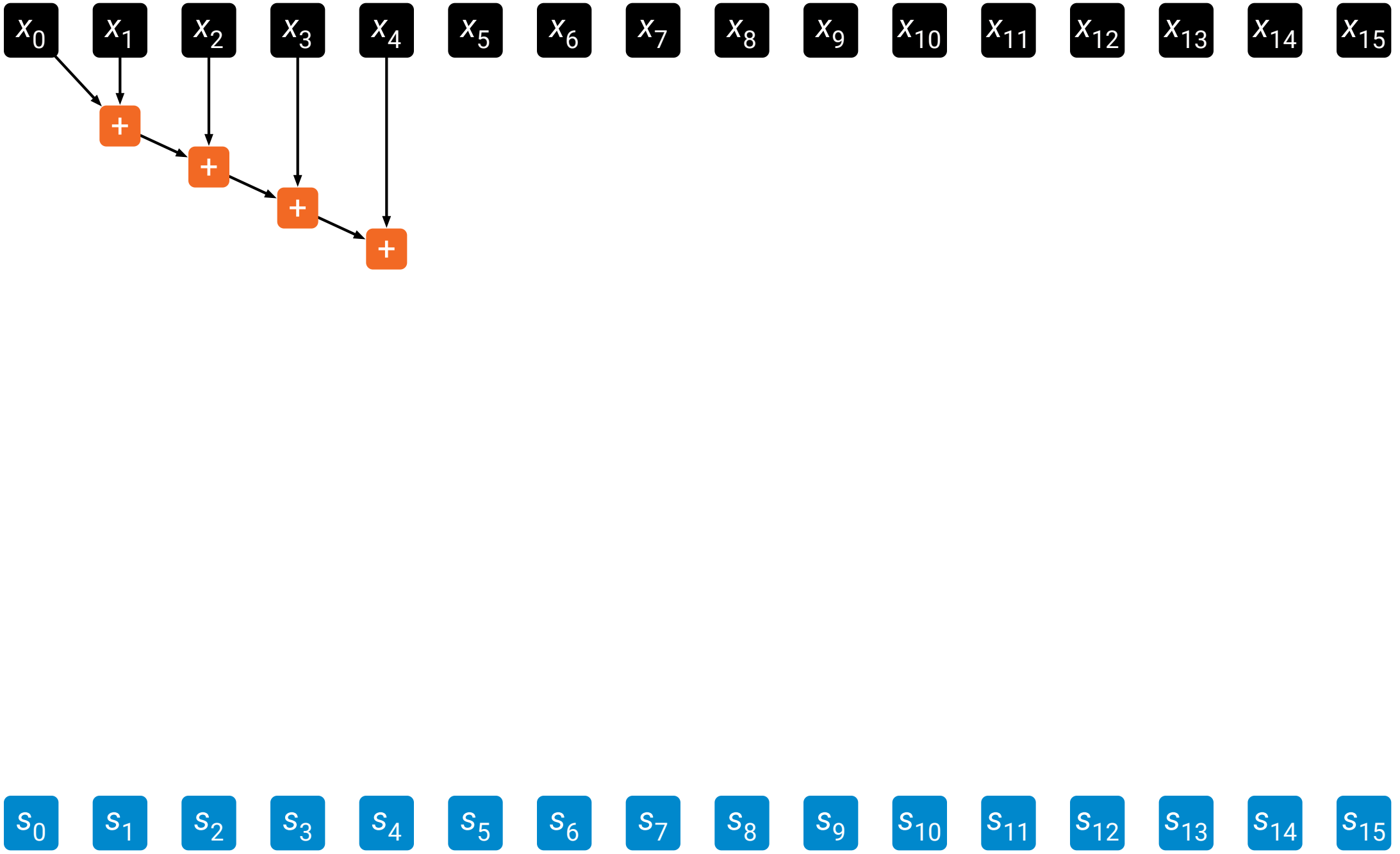
Sequential prefix sum

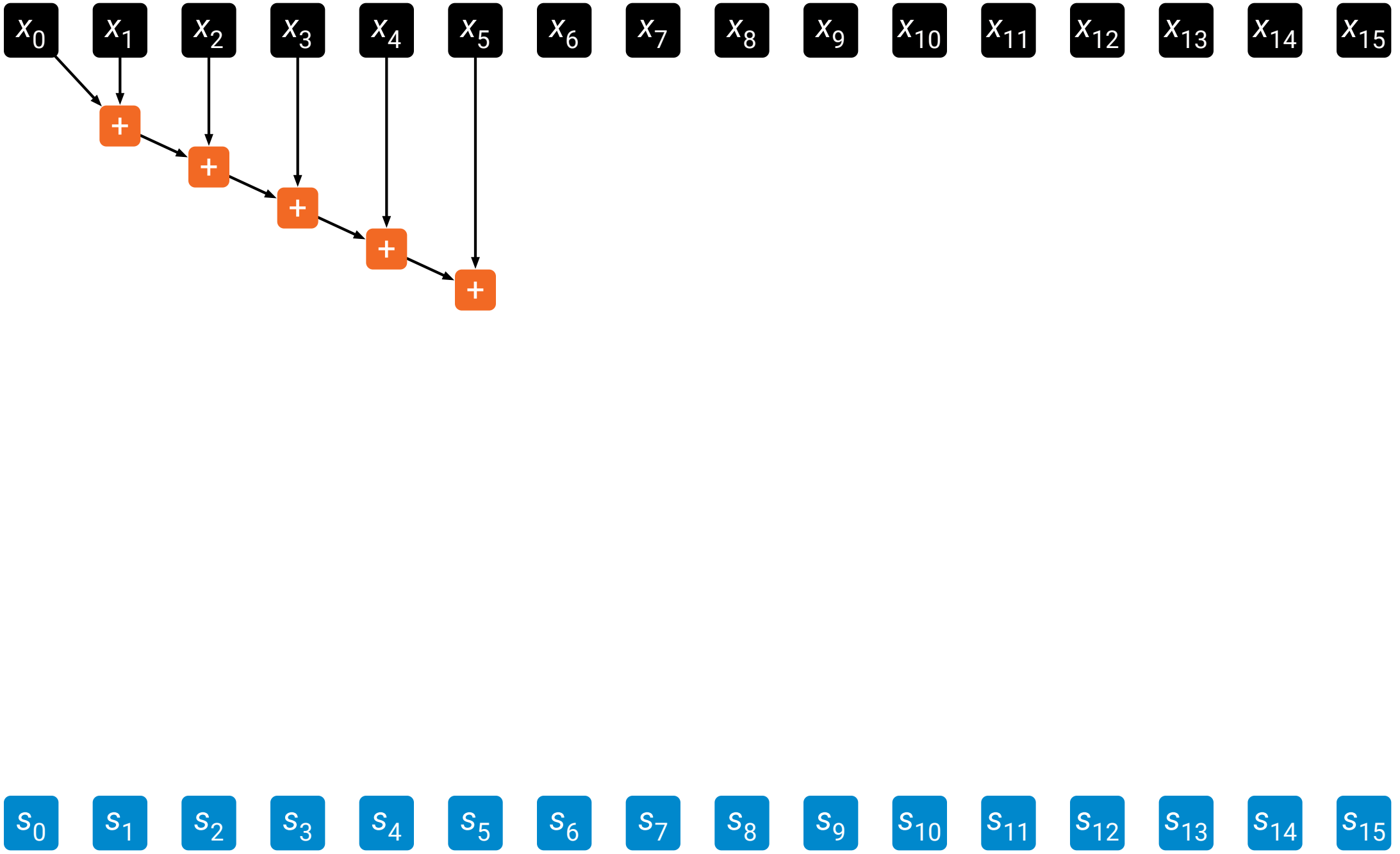

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

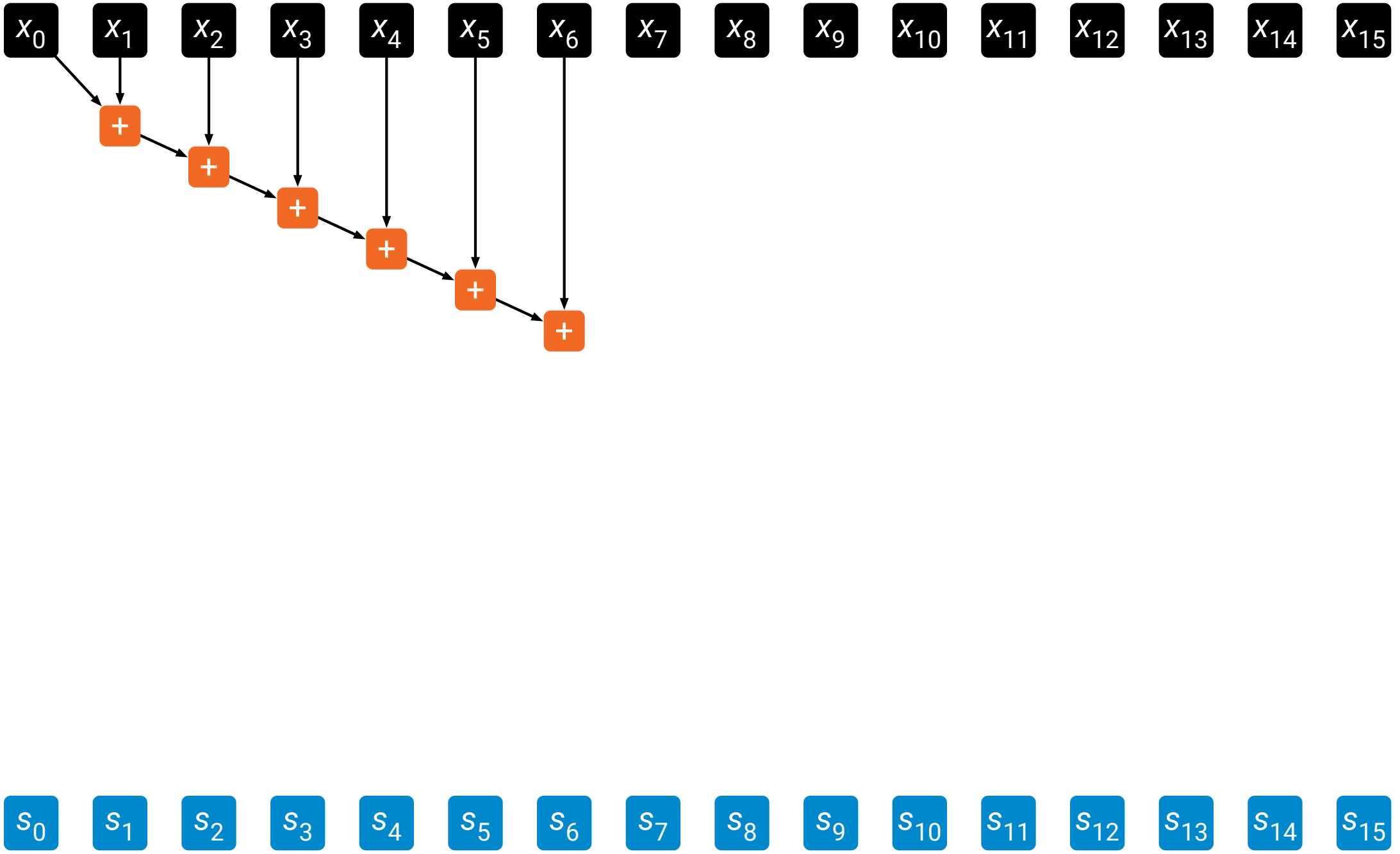

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

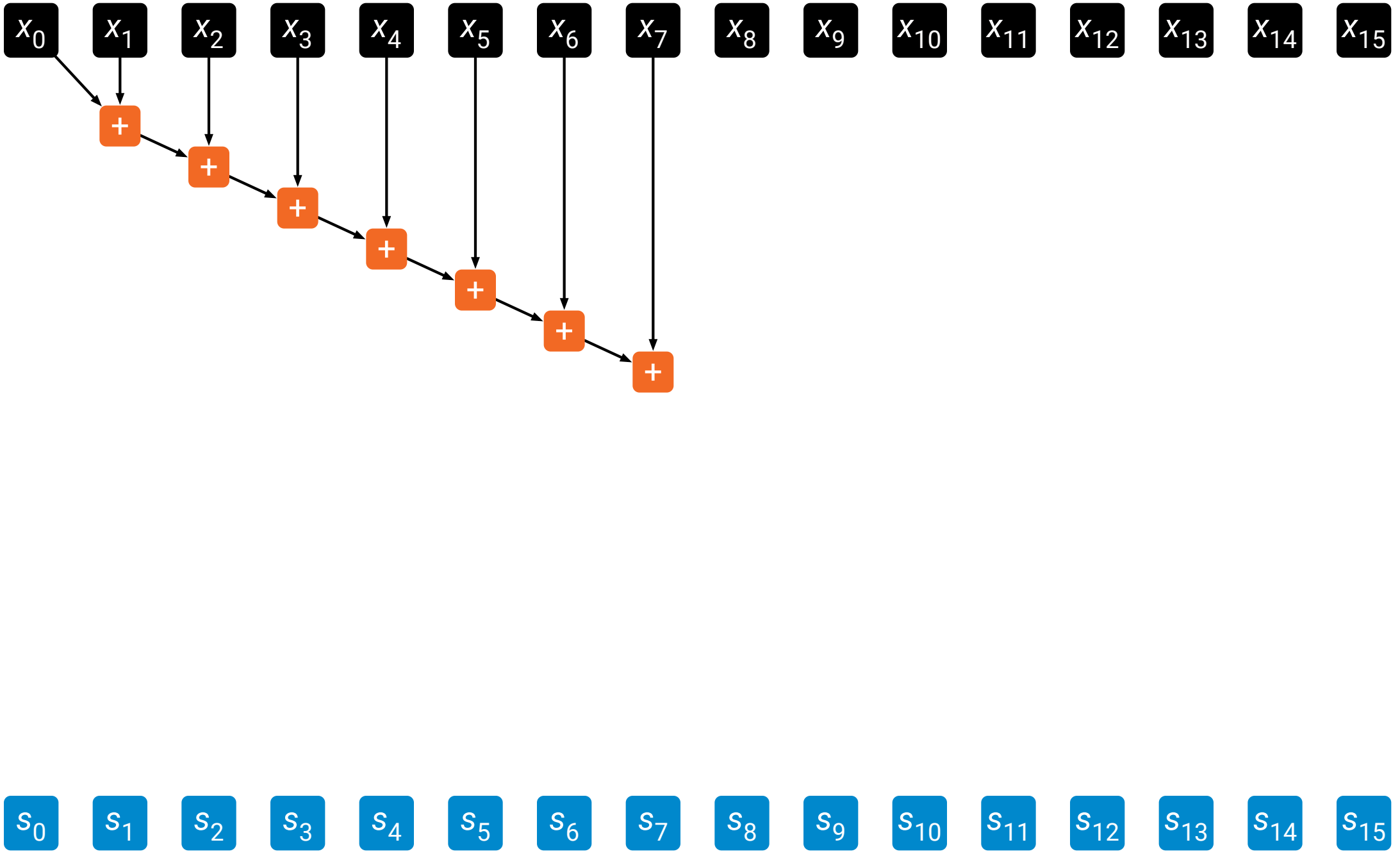

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

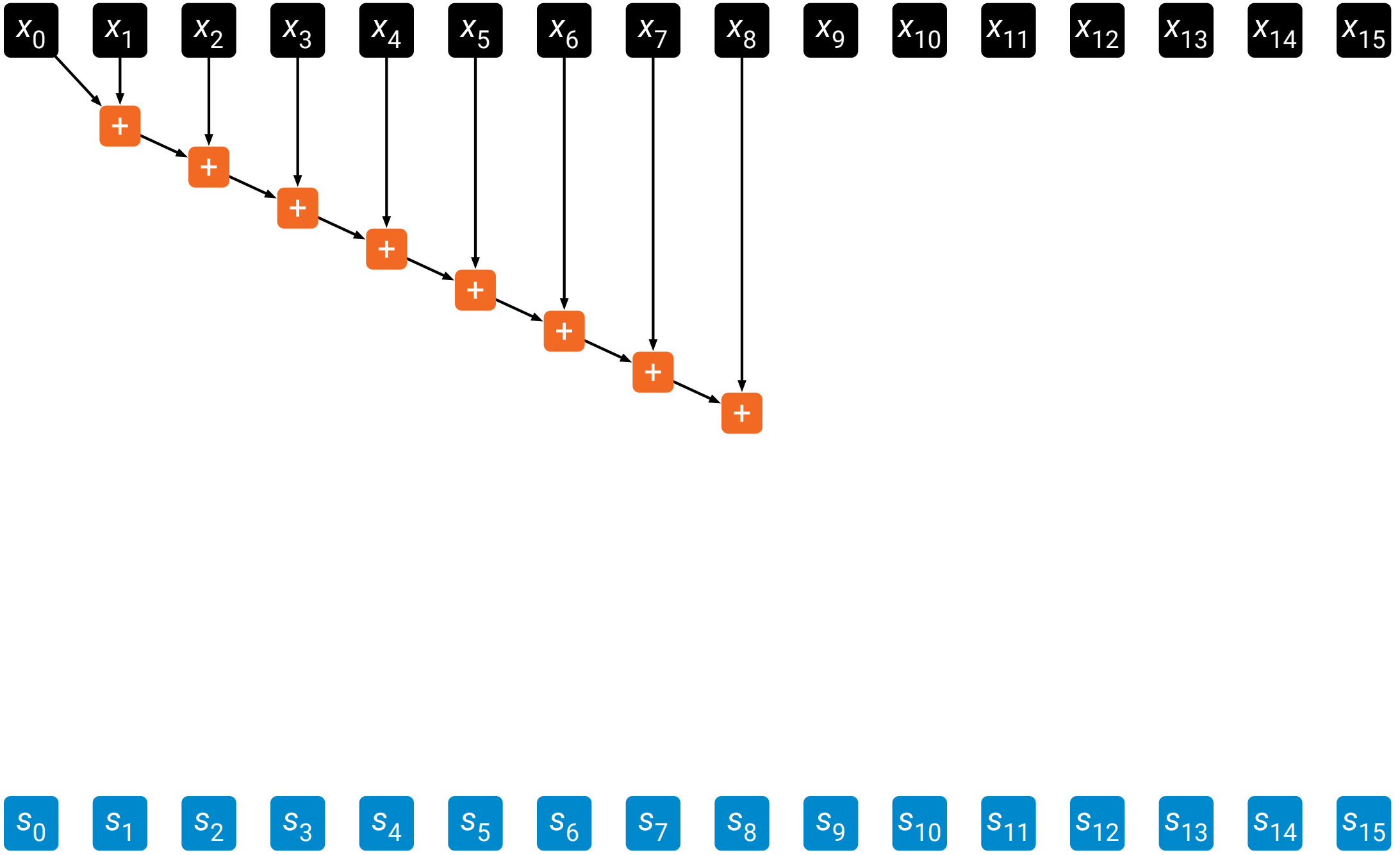

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
+
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

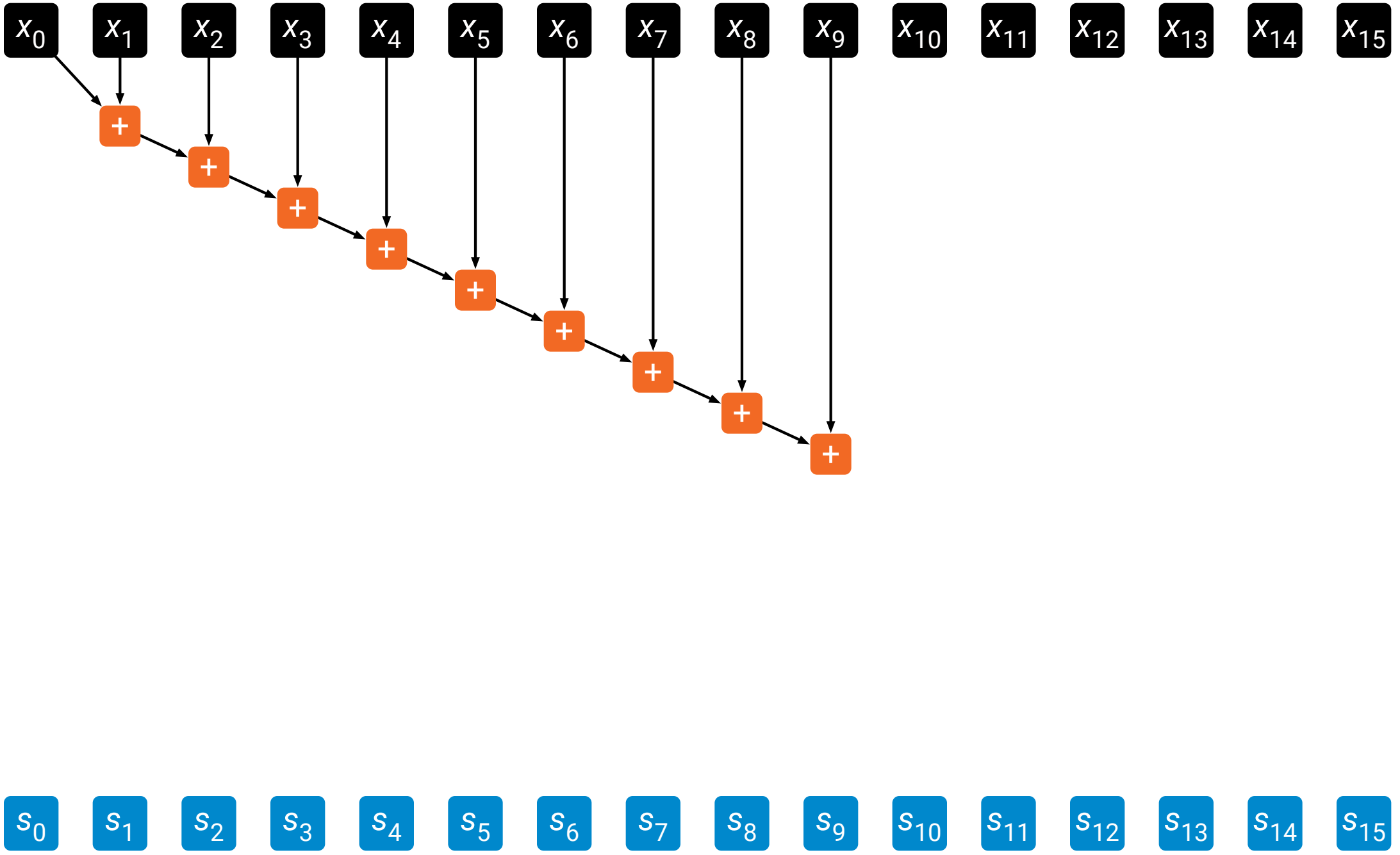

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
+
+
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

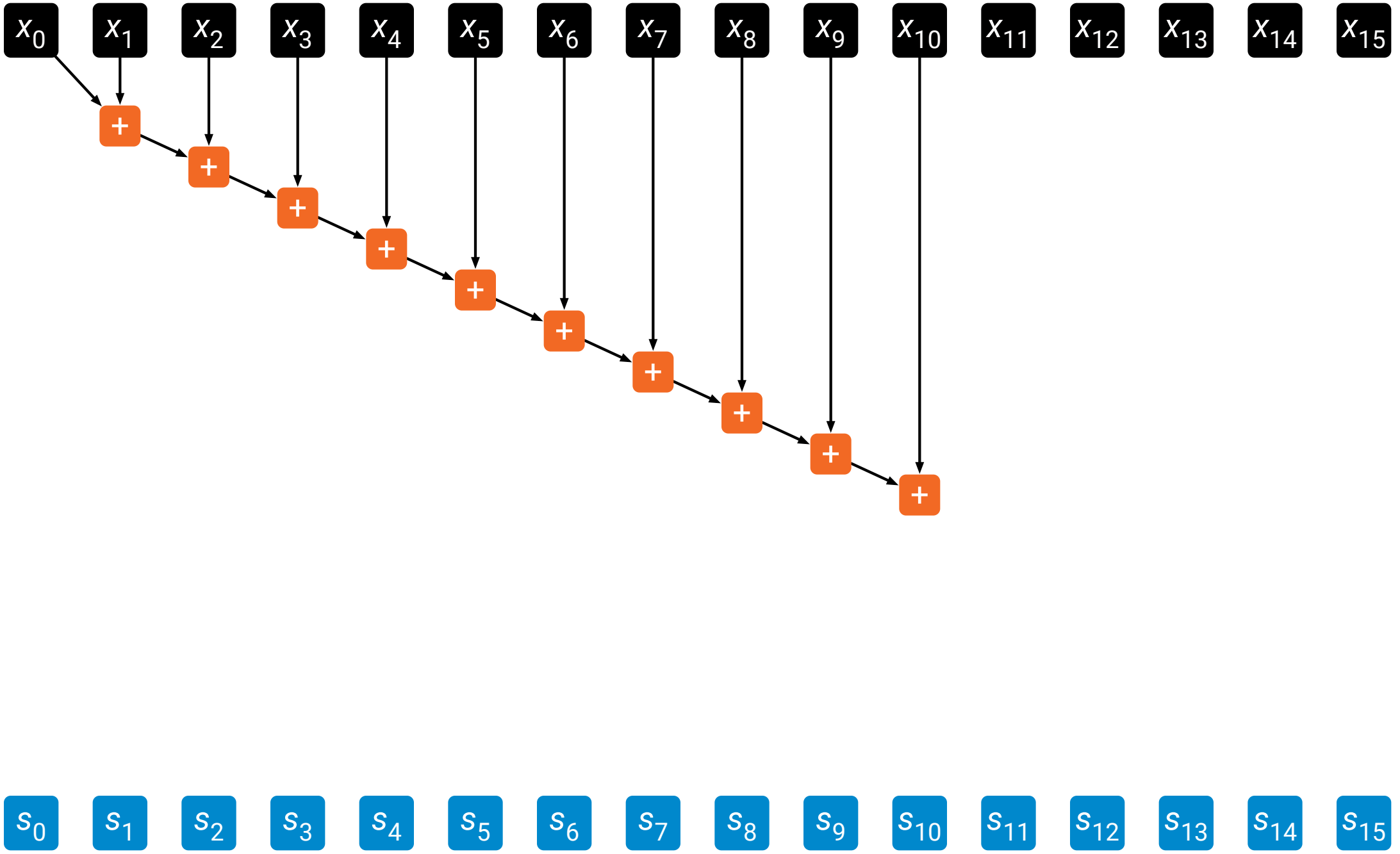

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
+
+
+
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

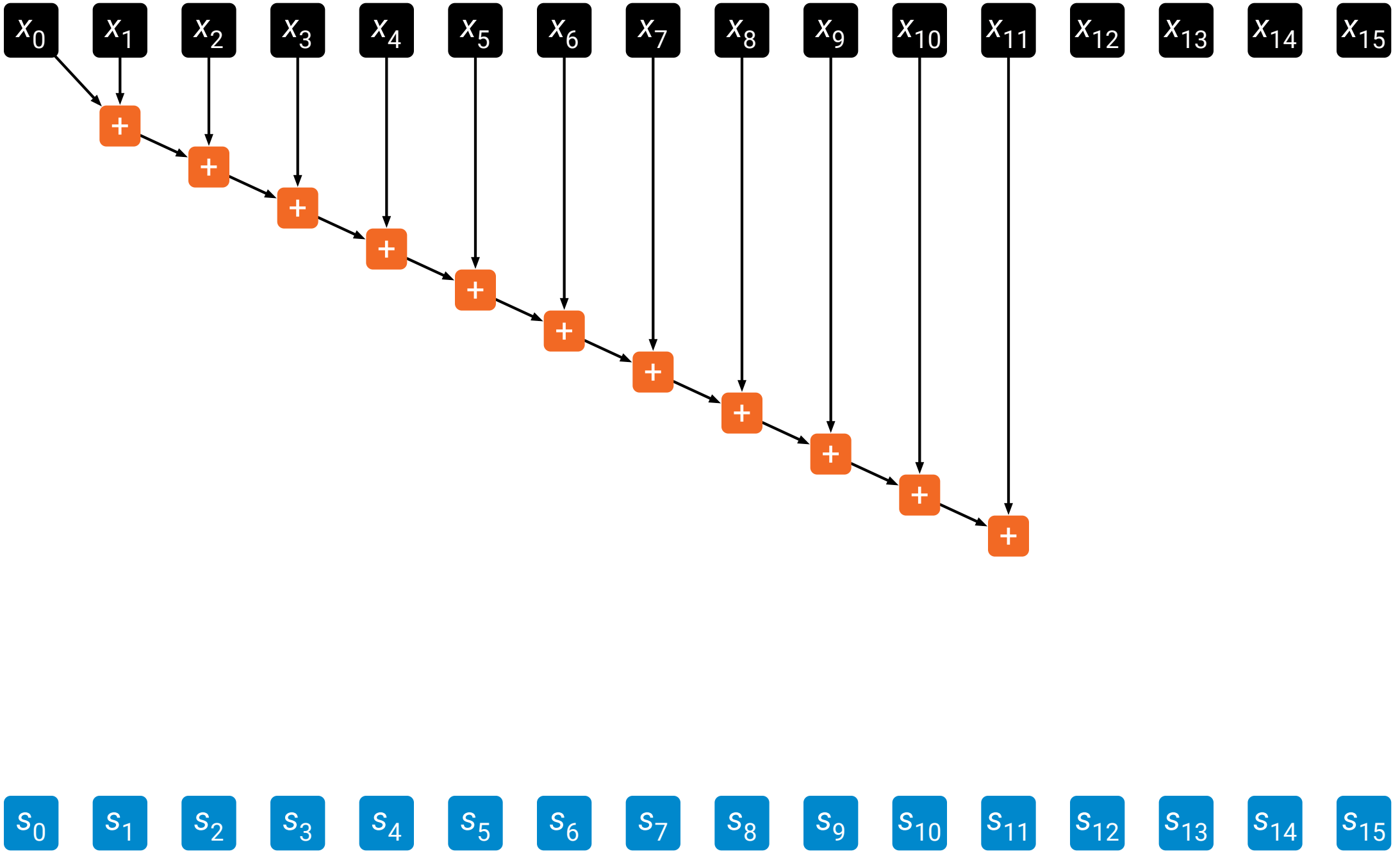

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+ + + + + +
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

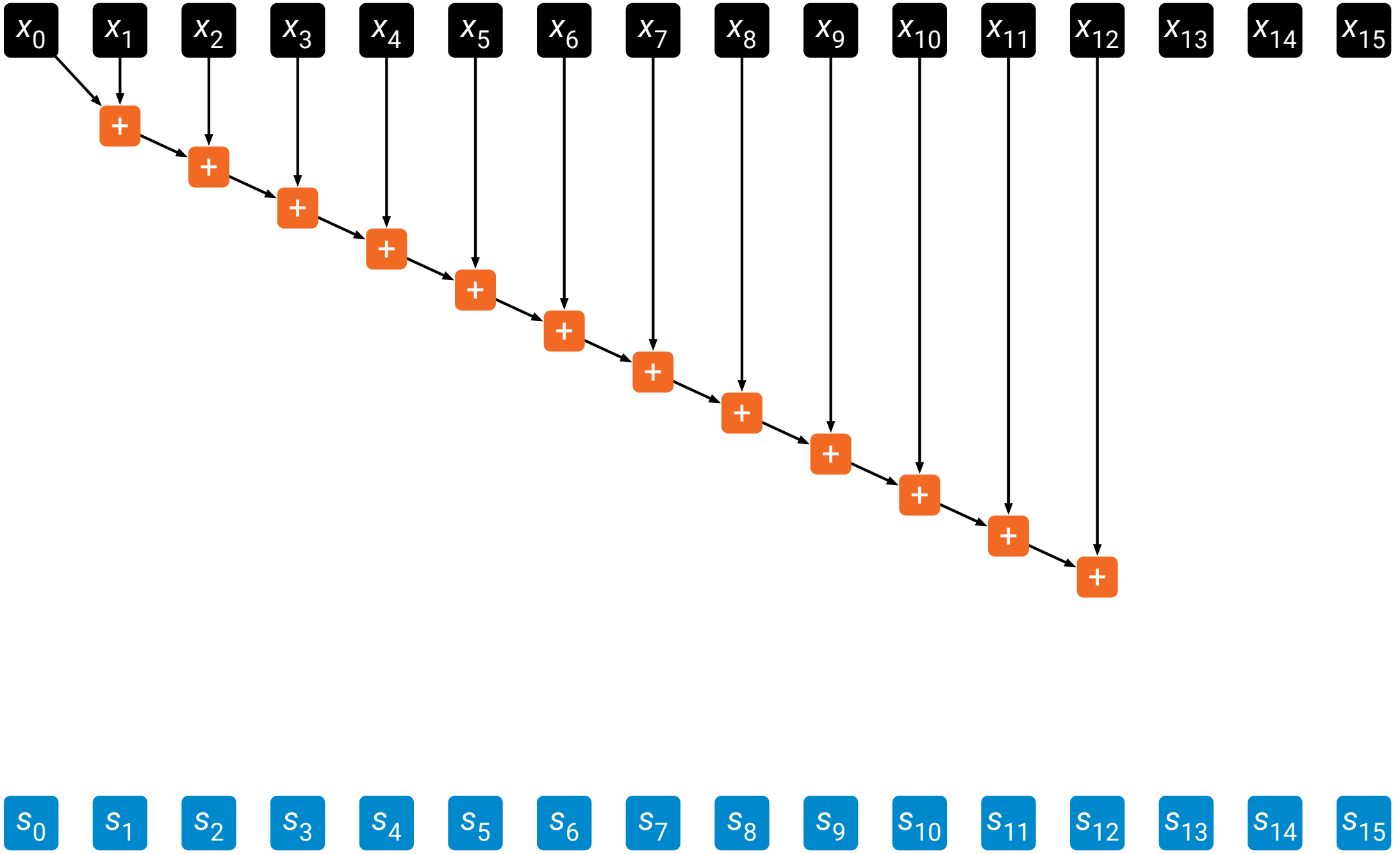

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+ + + + + + +
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

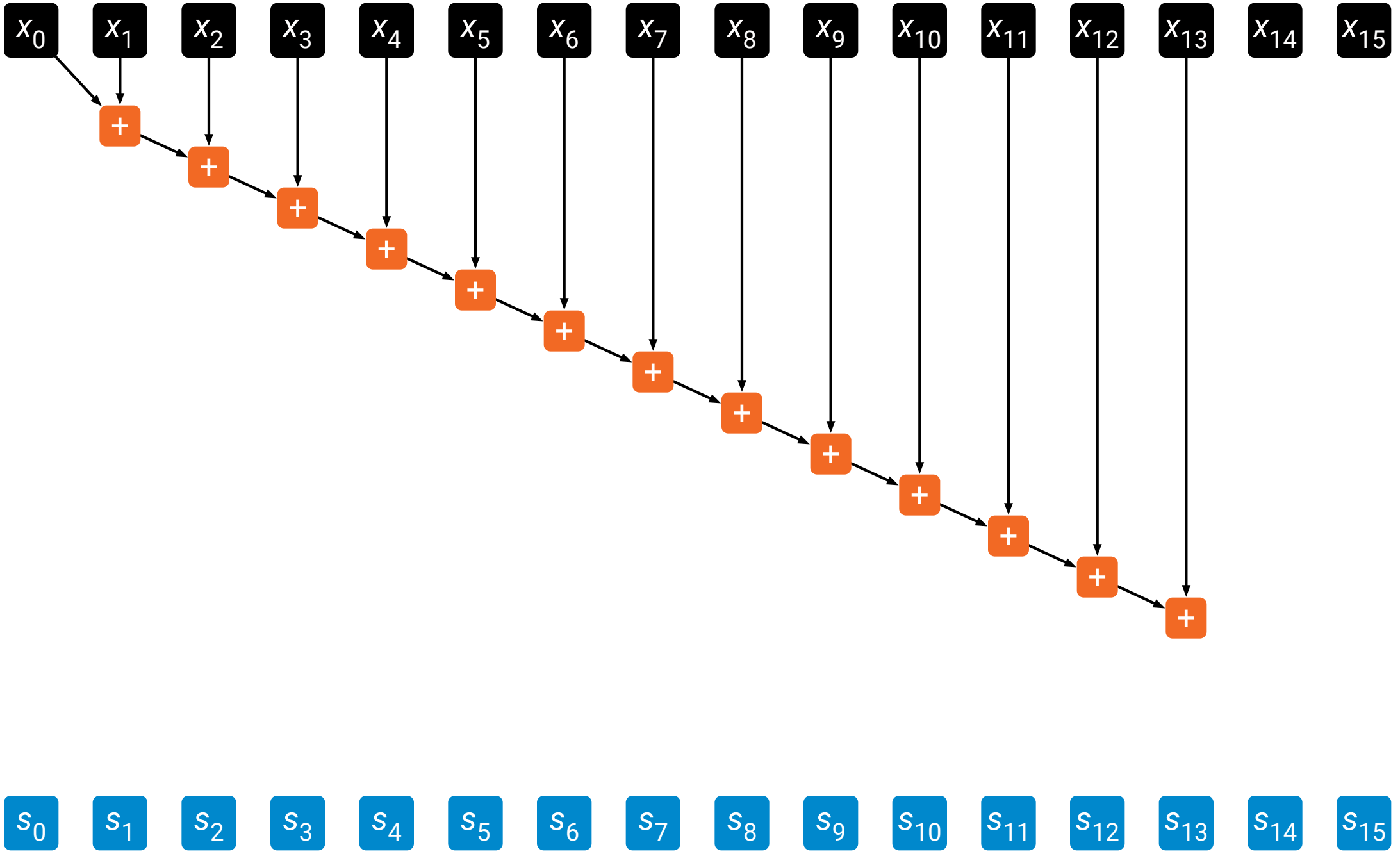

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
+
+
+
+
+
+
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

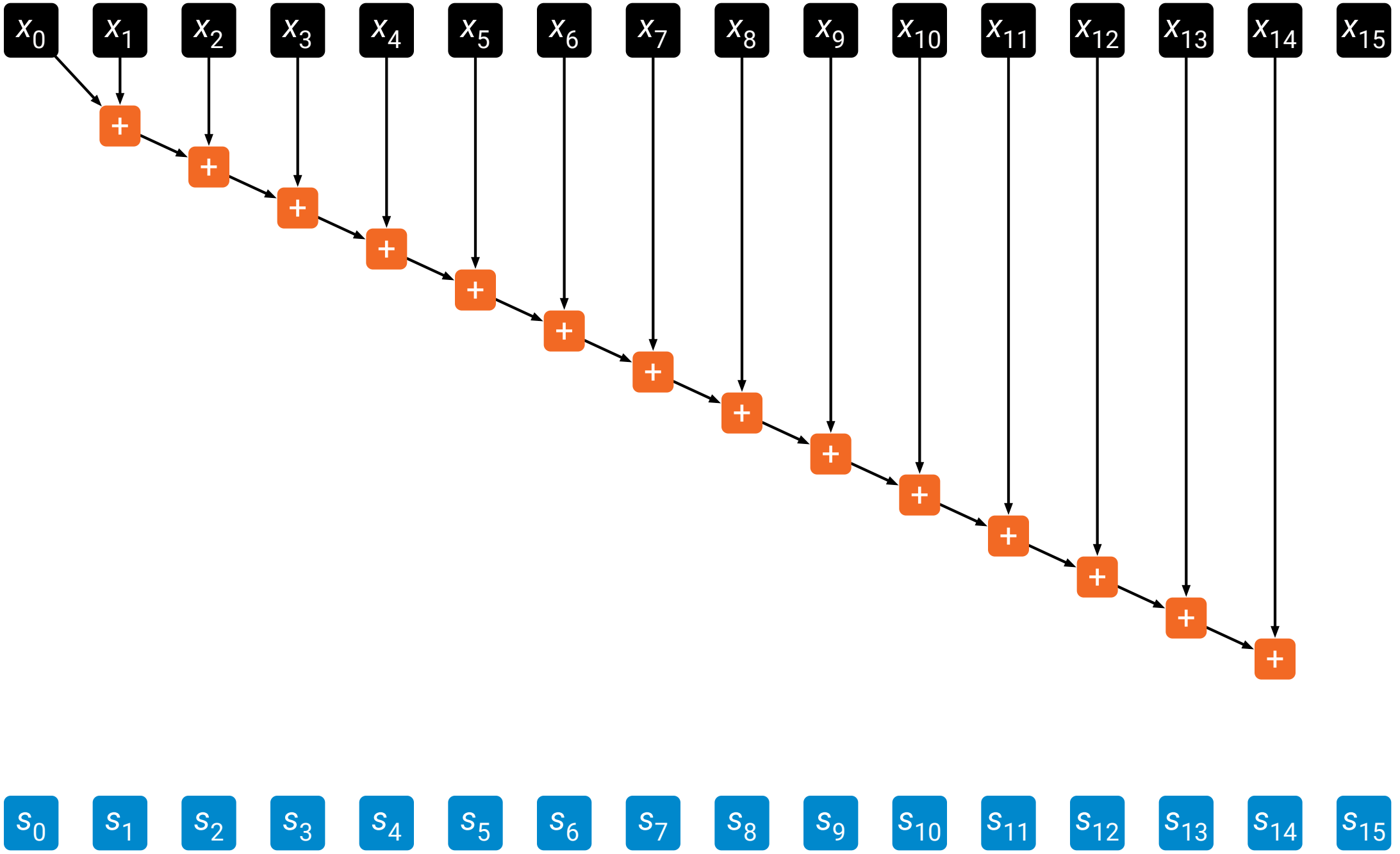

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

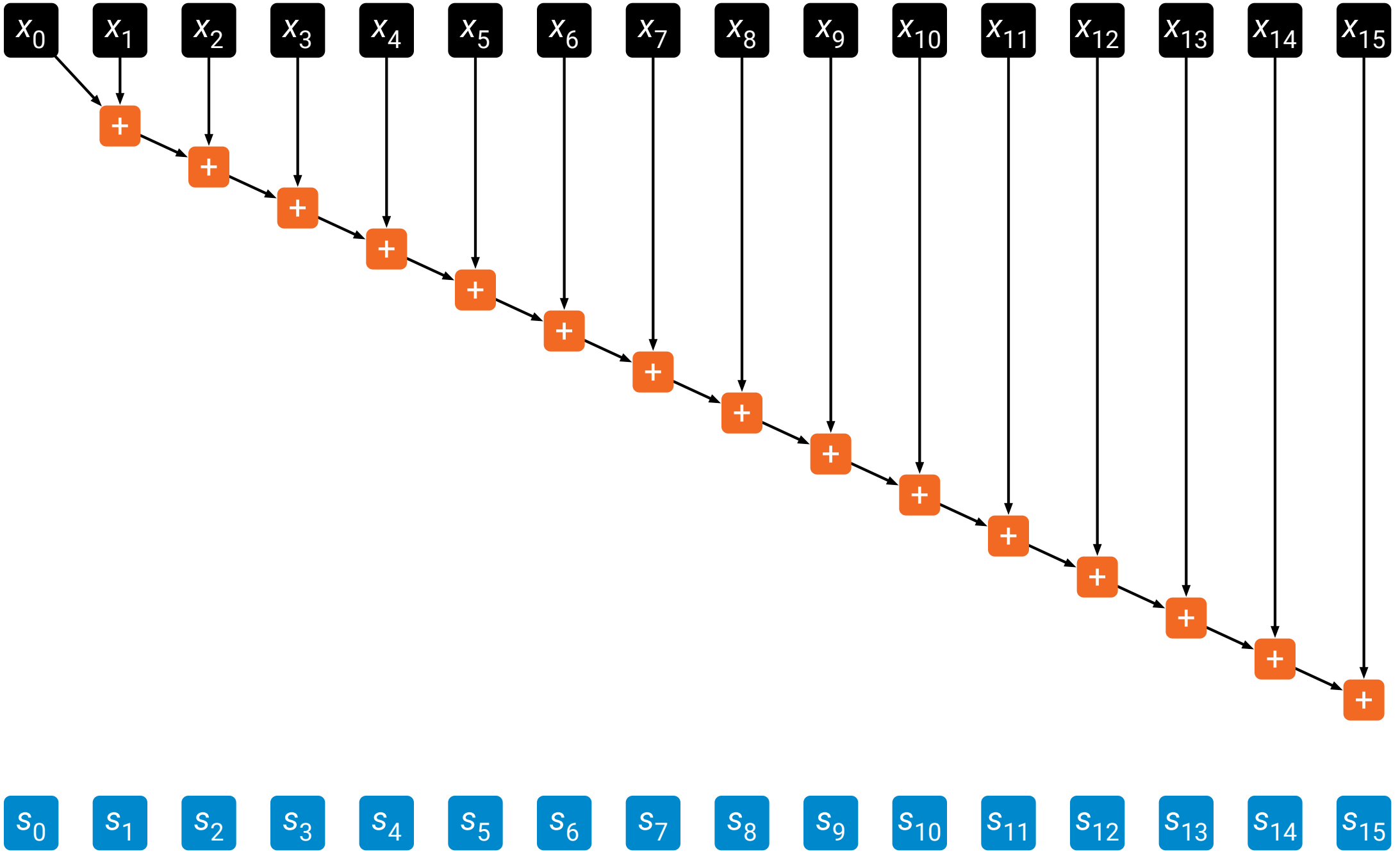

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

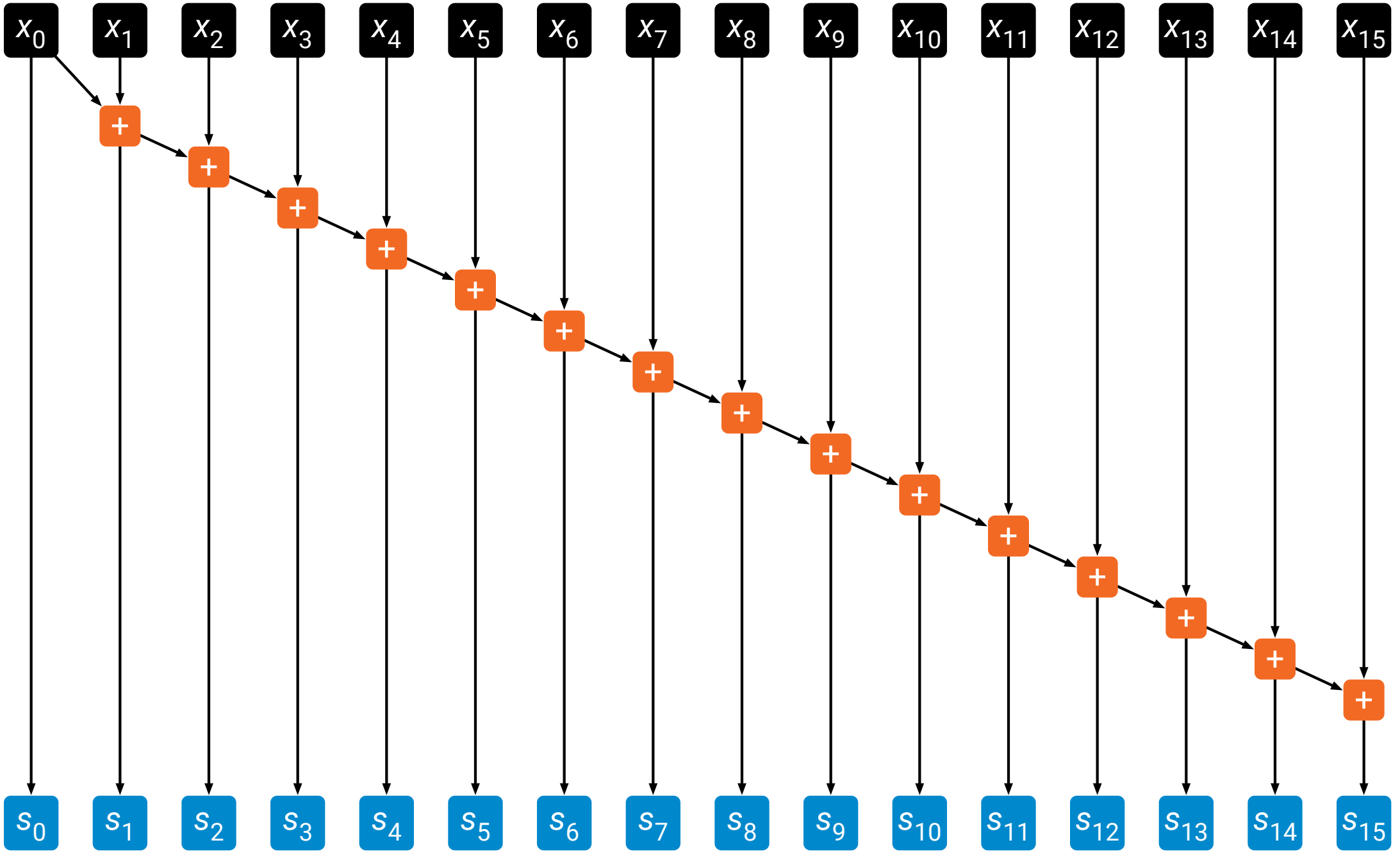

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}


x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

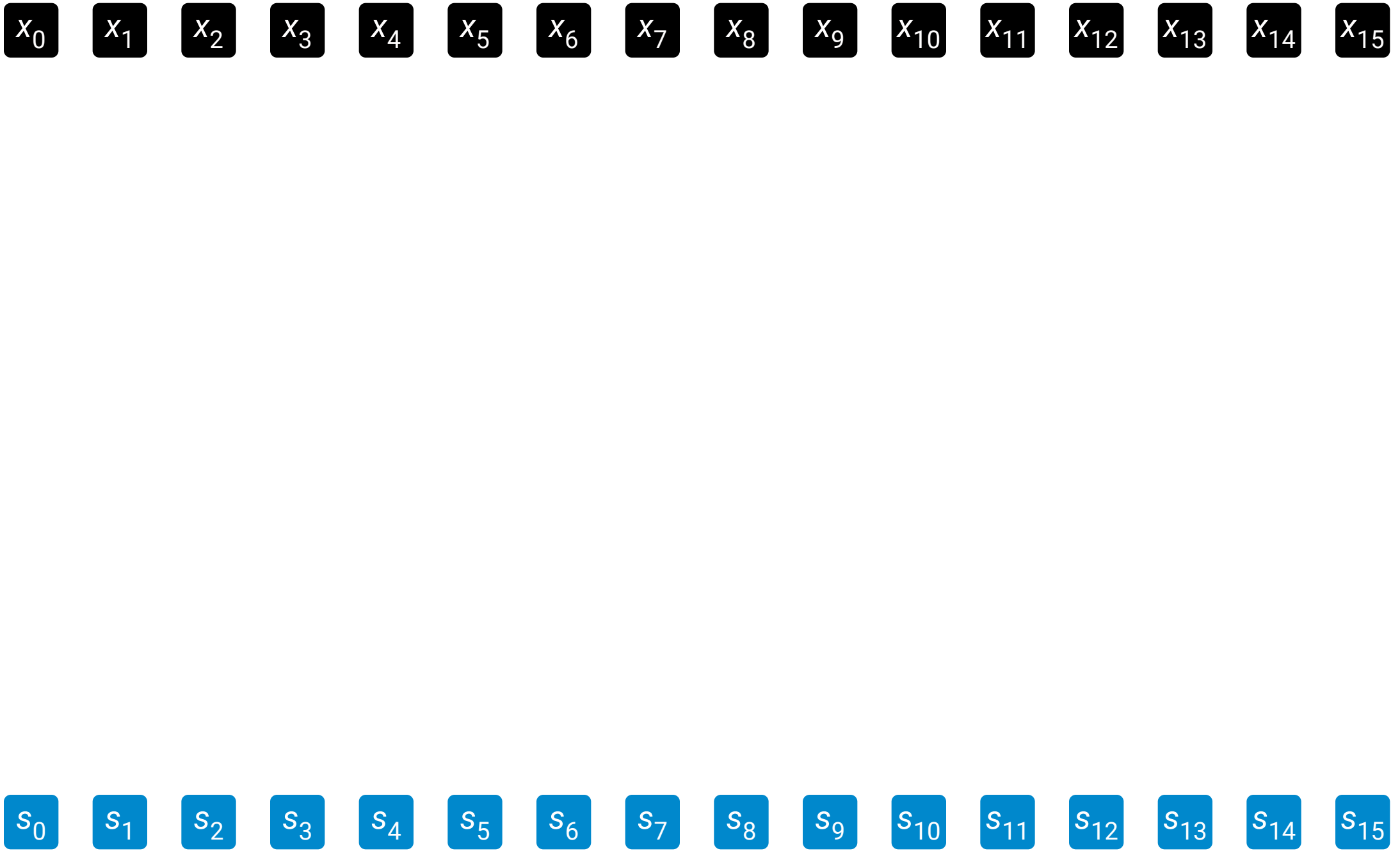

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

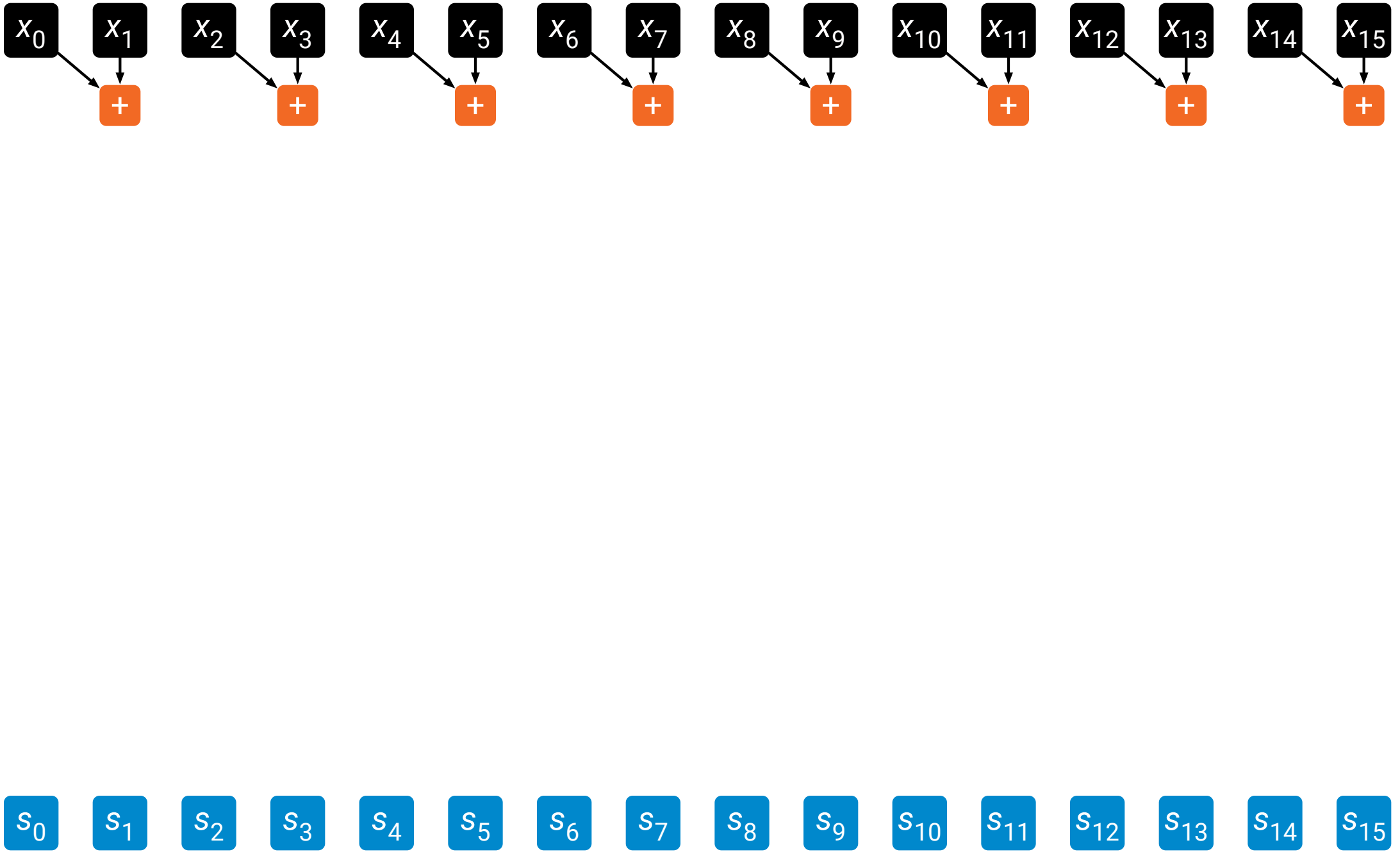

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

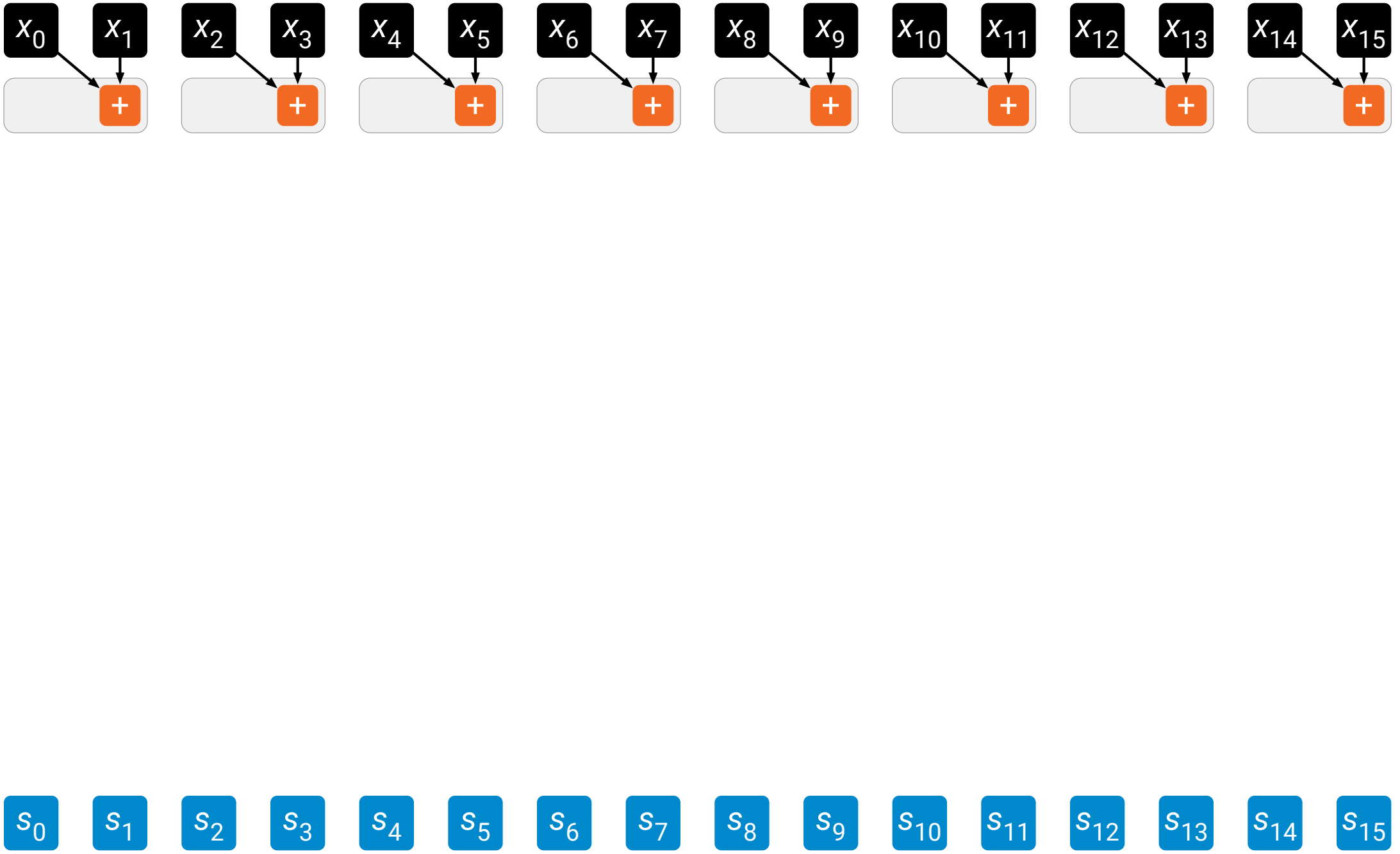

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

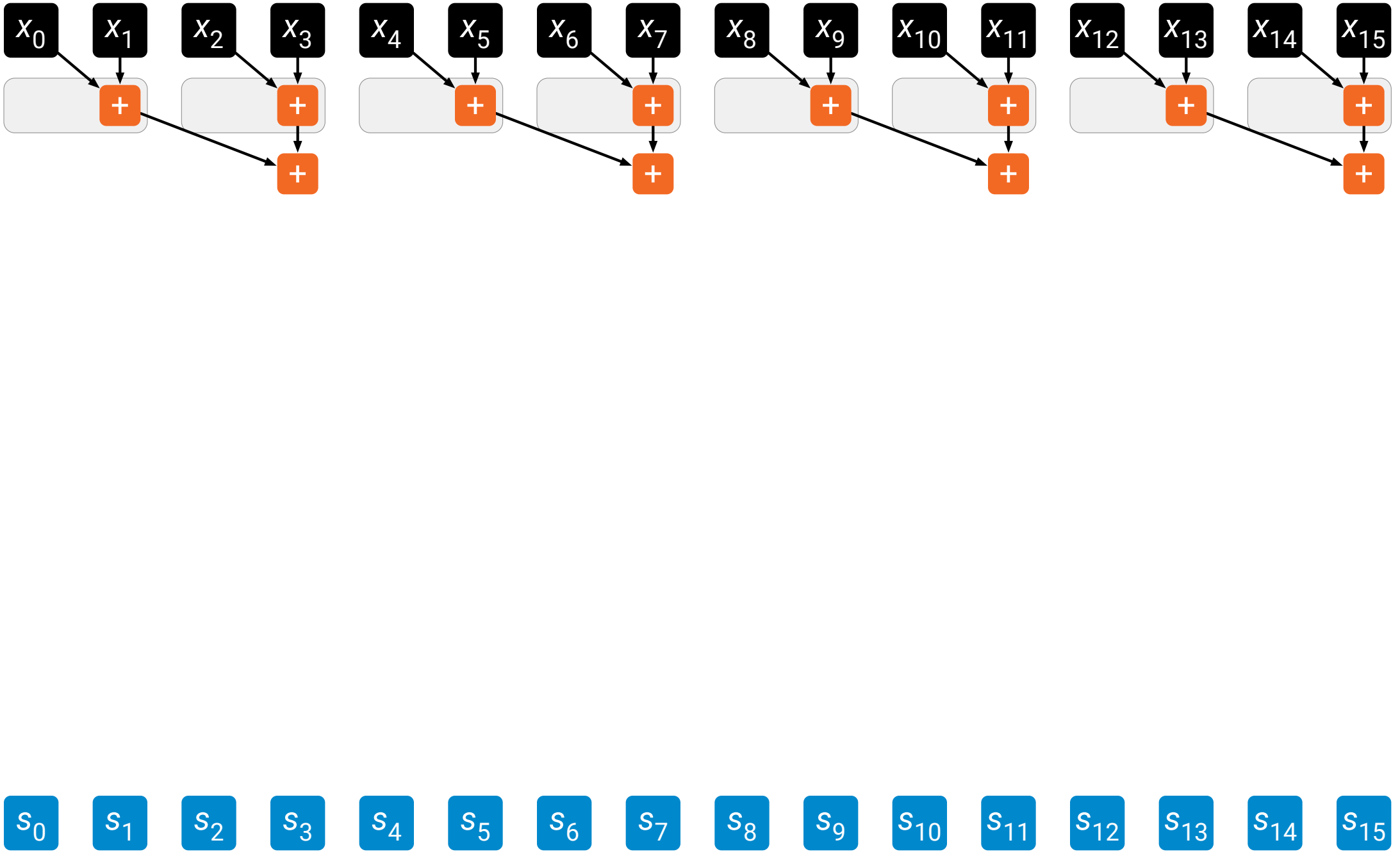

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

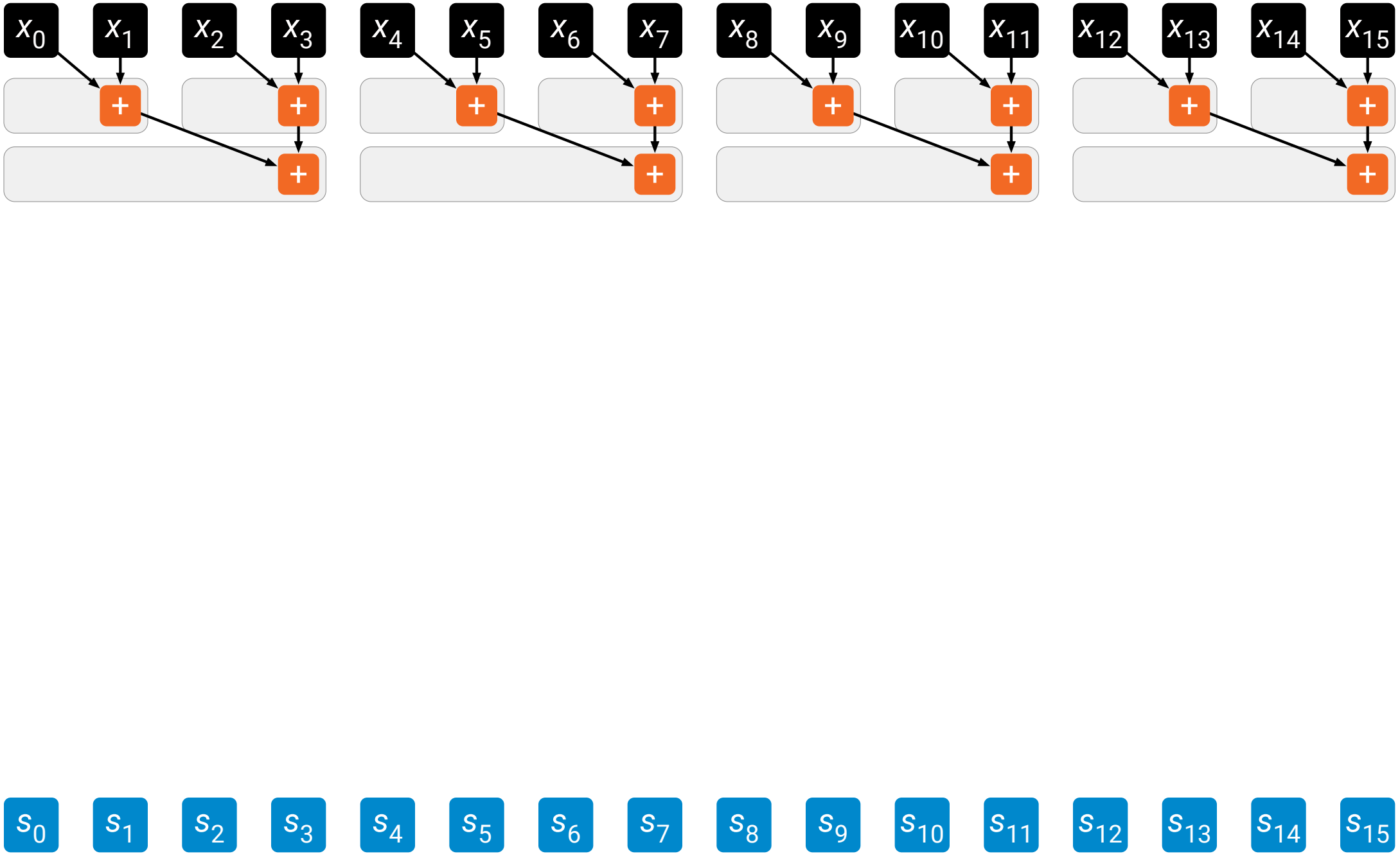
Parallel prefix sum

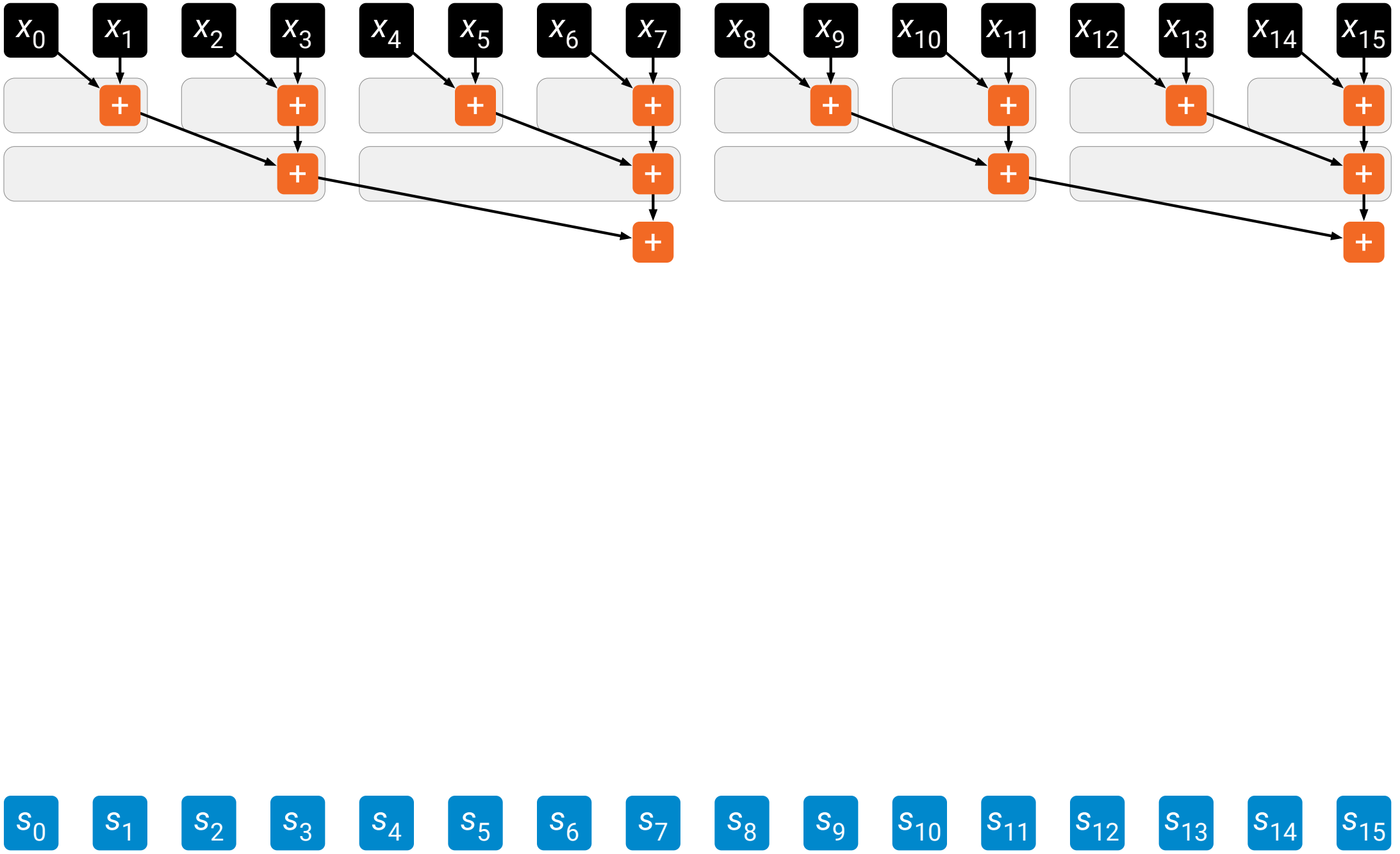

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

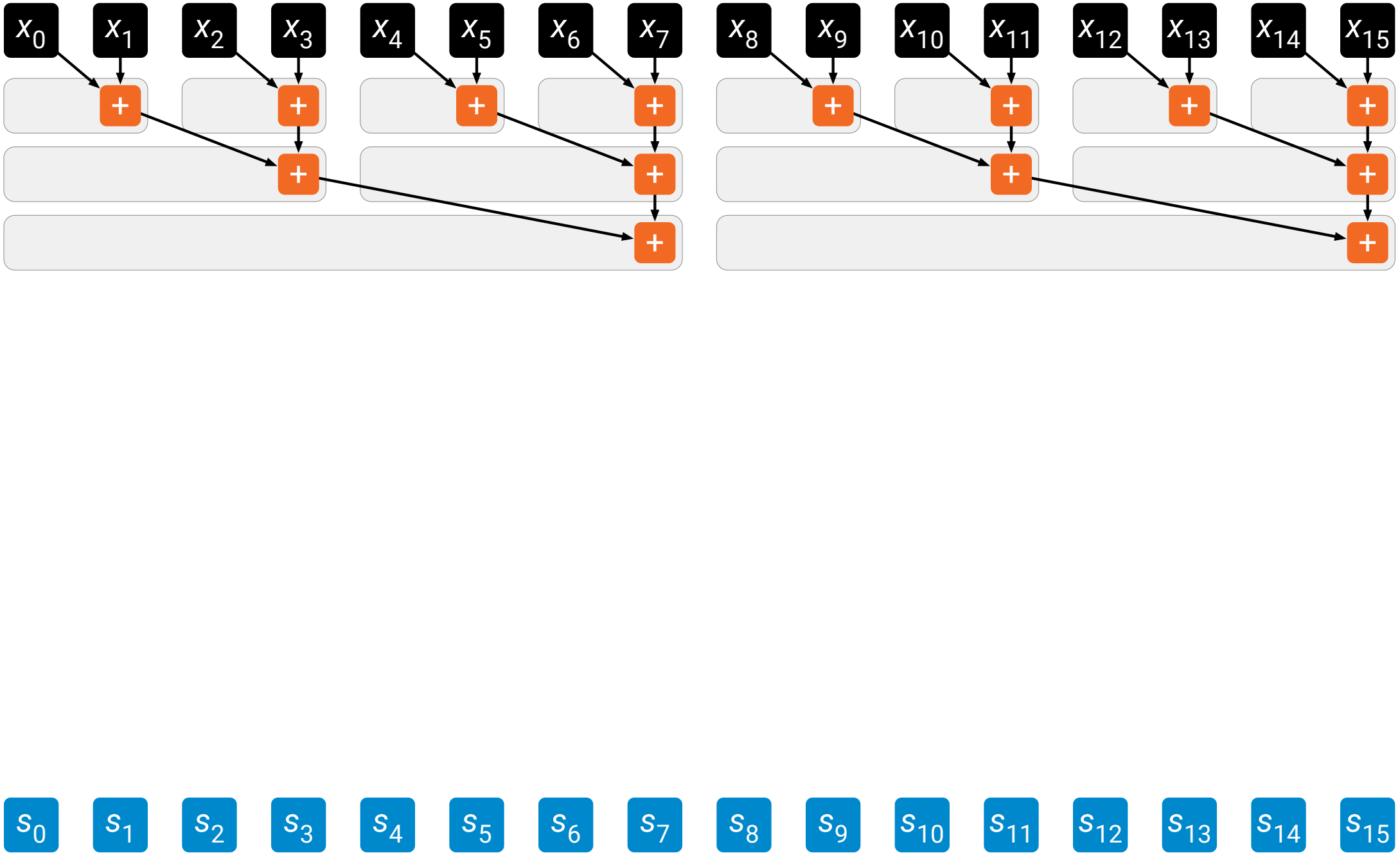

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+ + + + + + + + + + + + + + + +
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

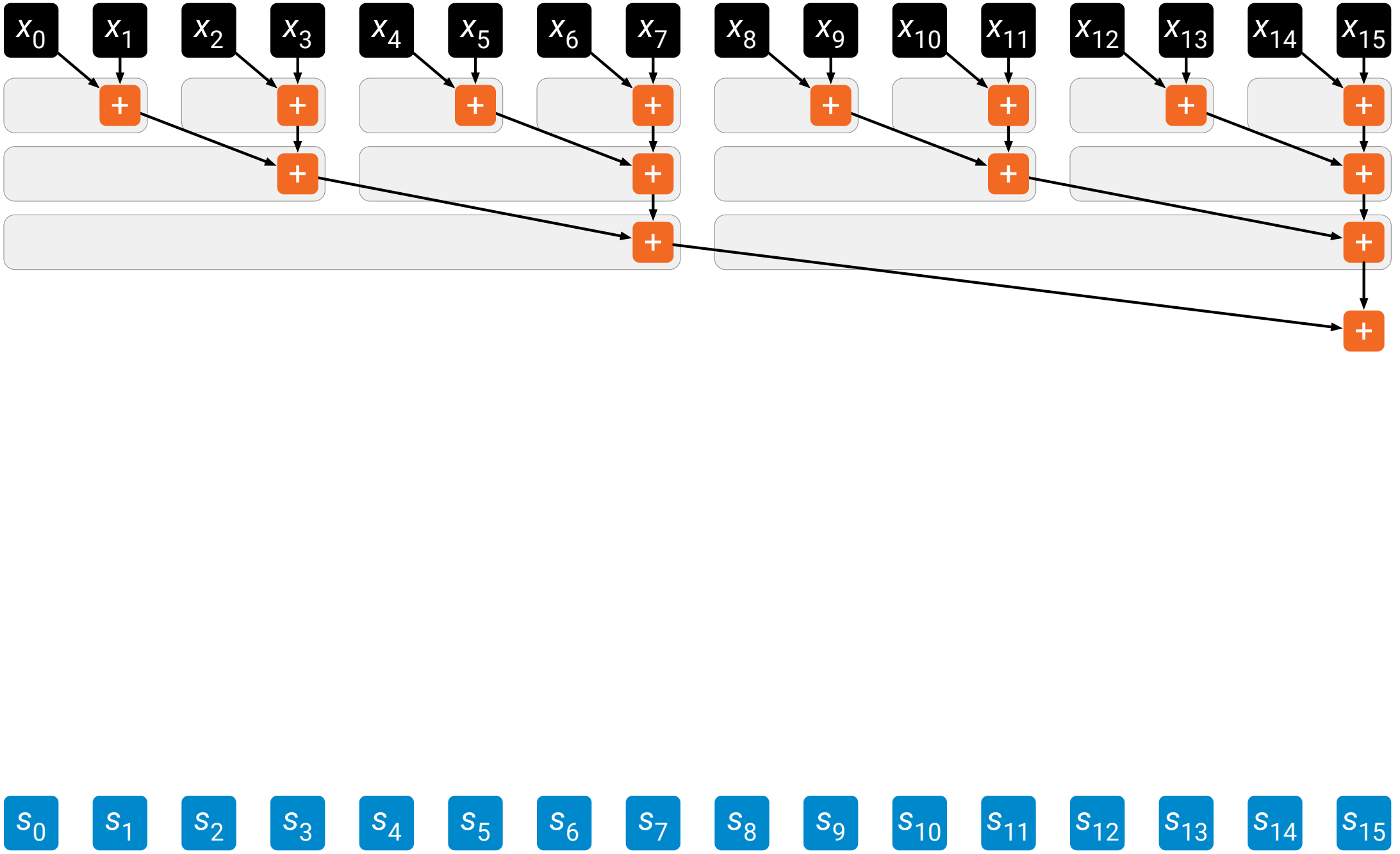

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

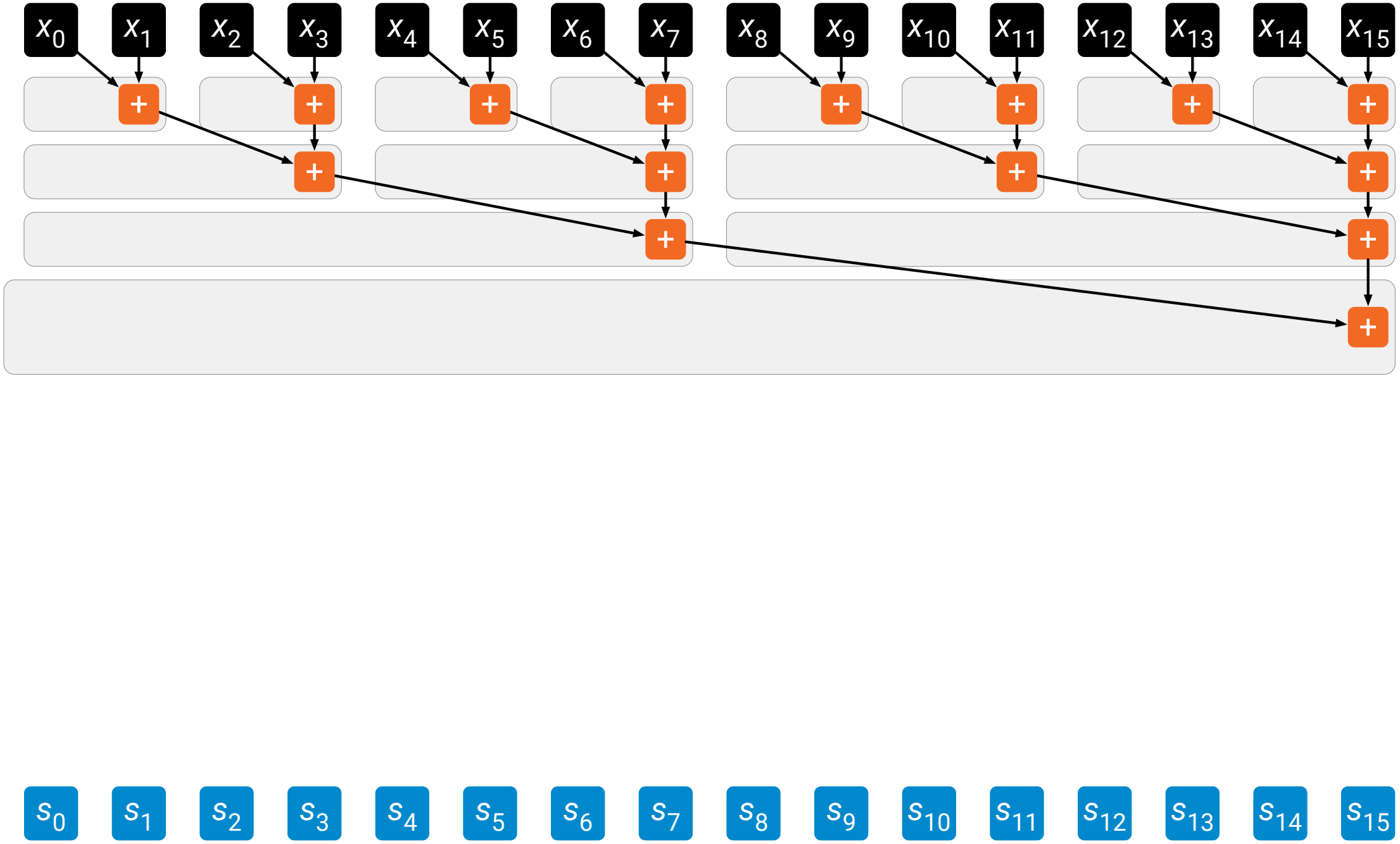

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

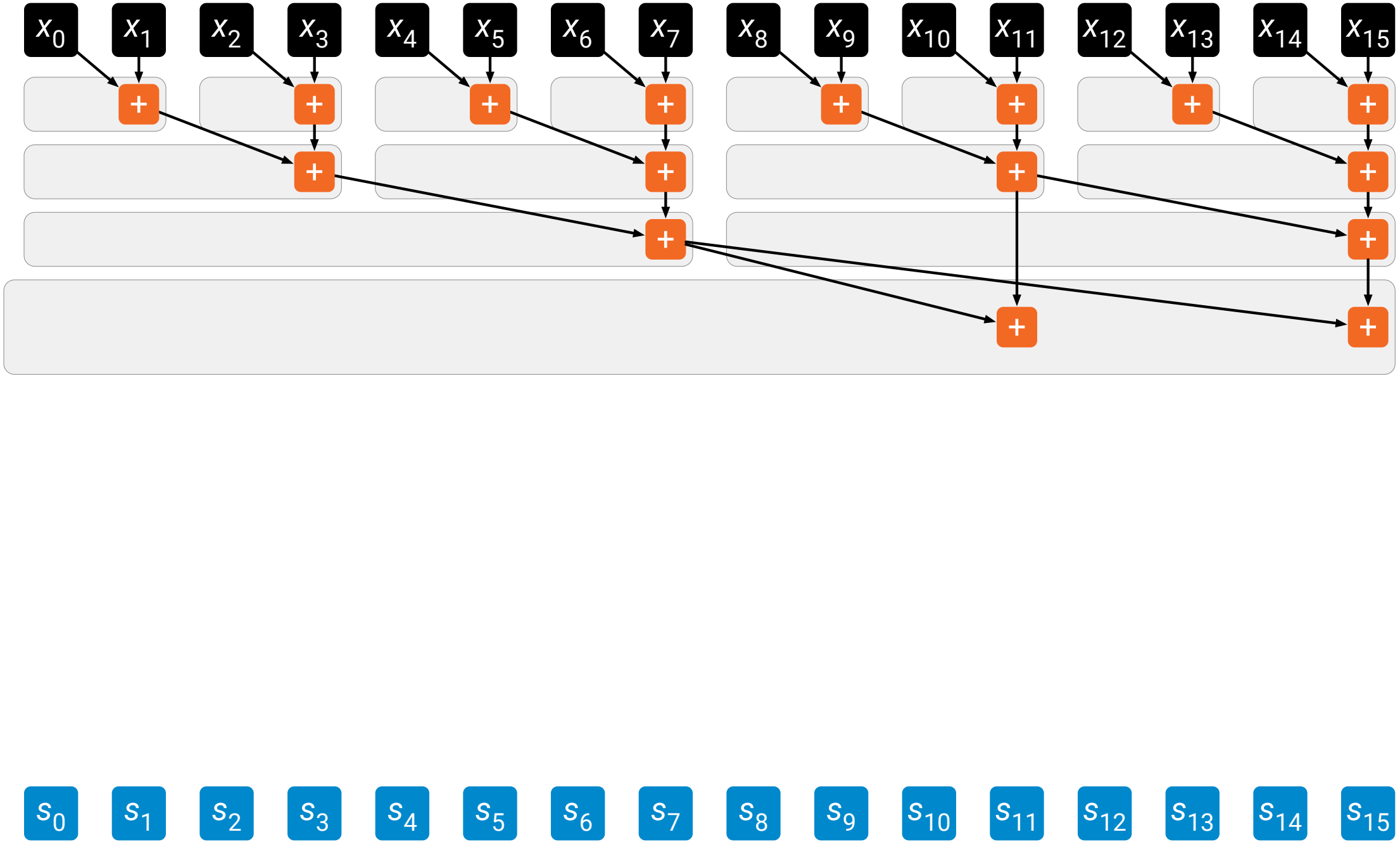

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

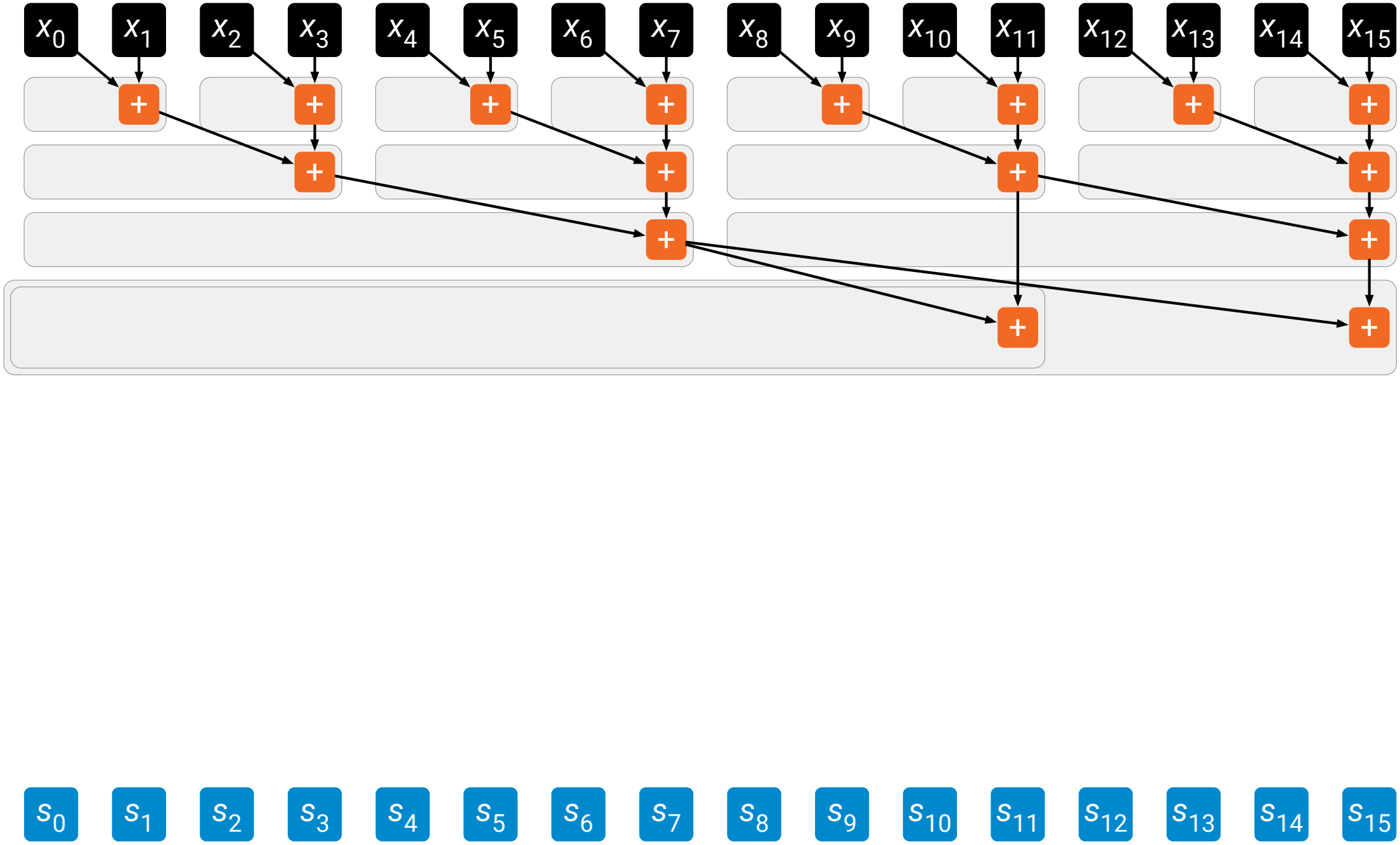

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

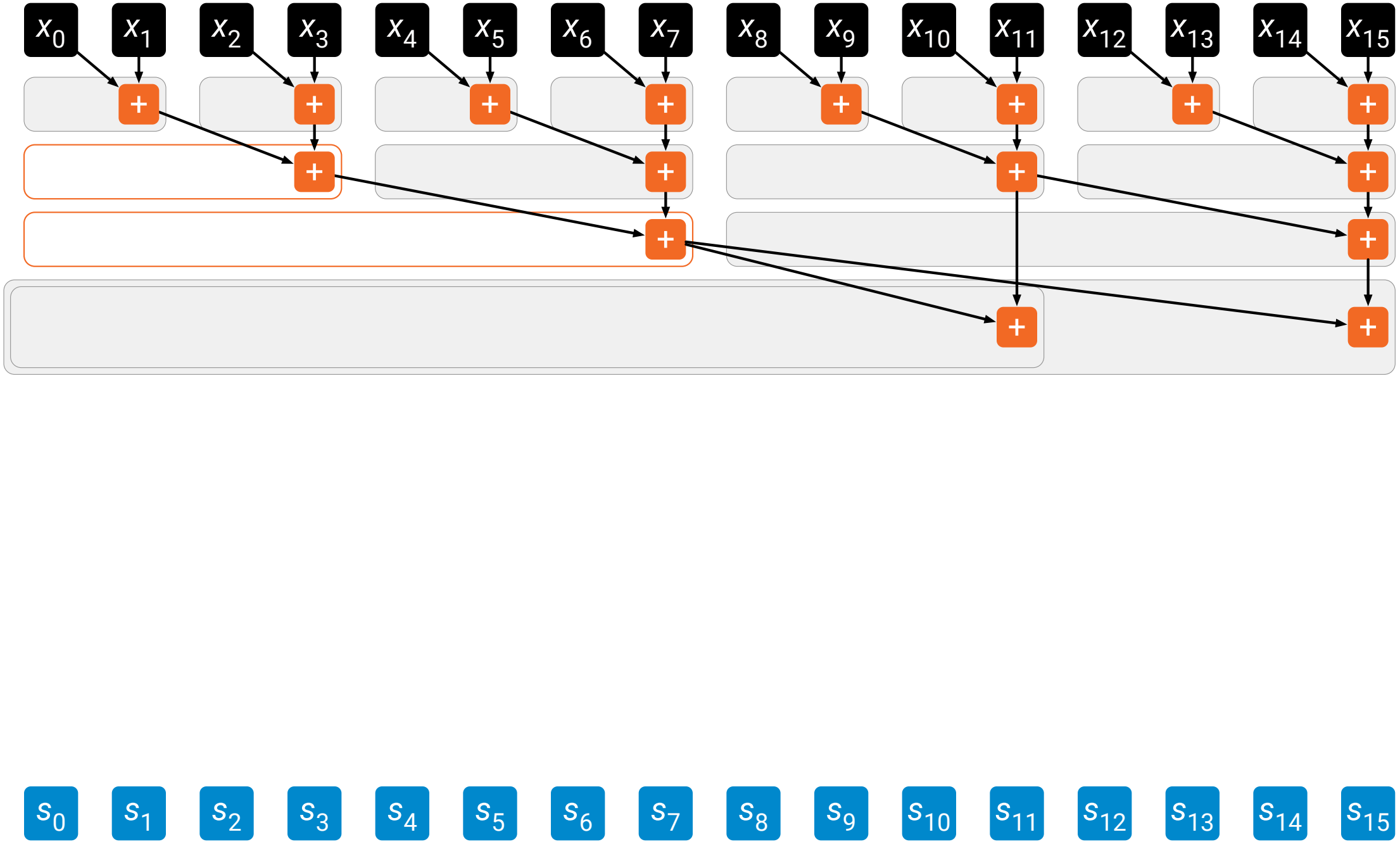

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

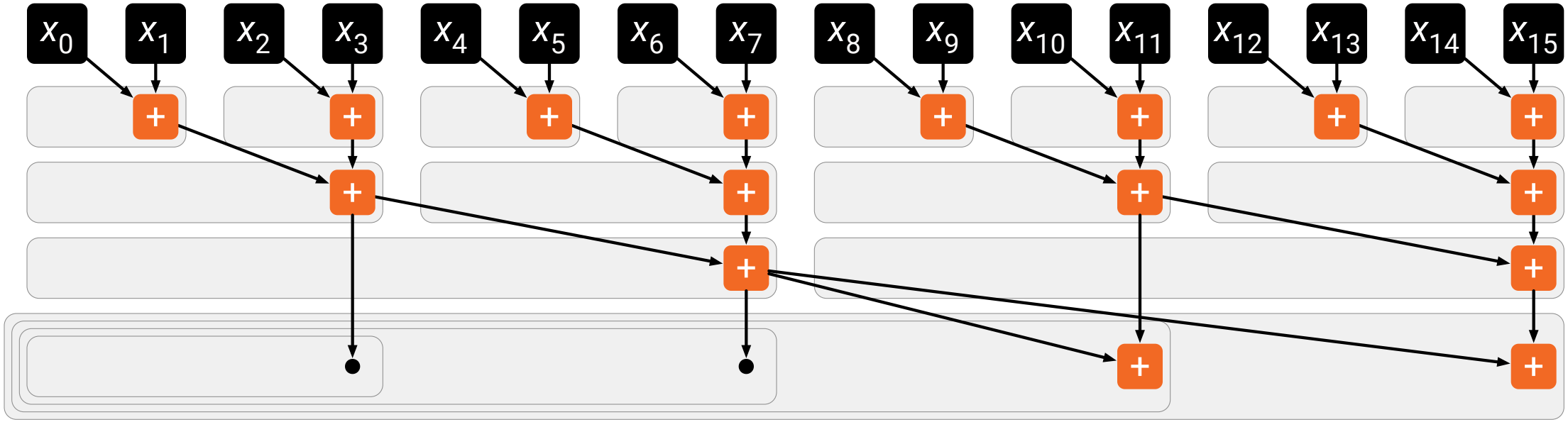

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}


x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

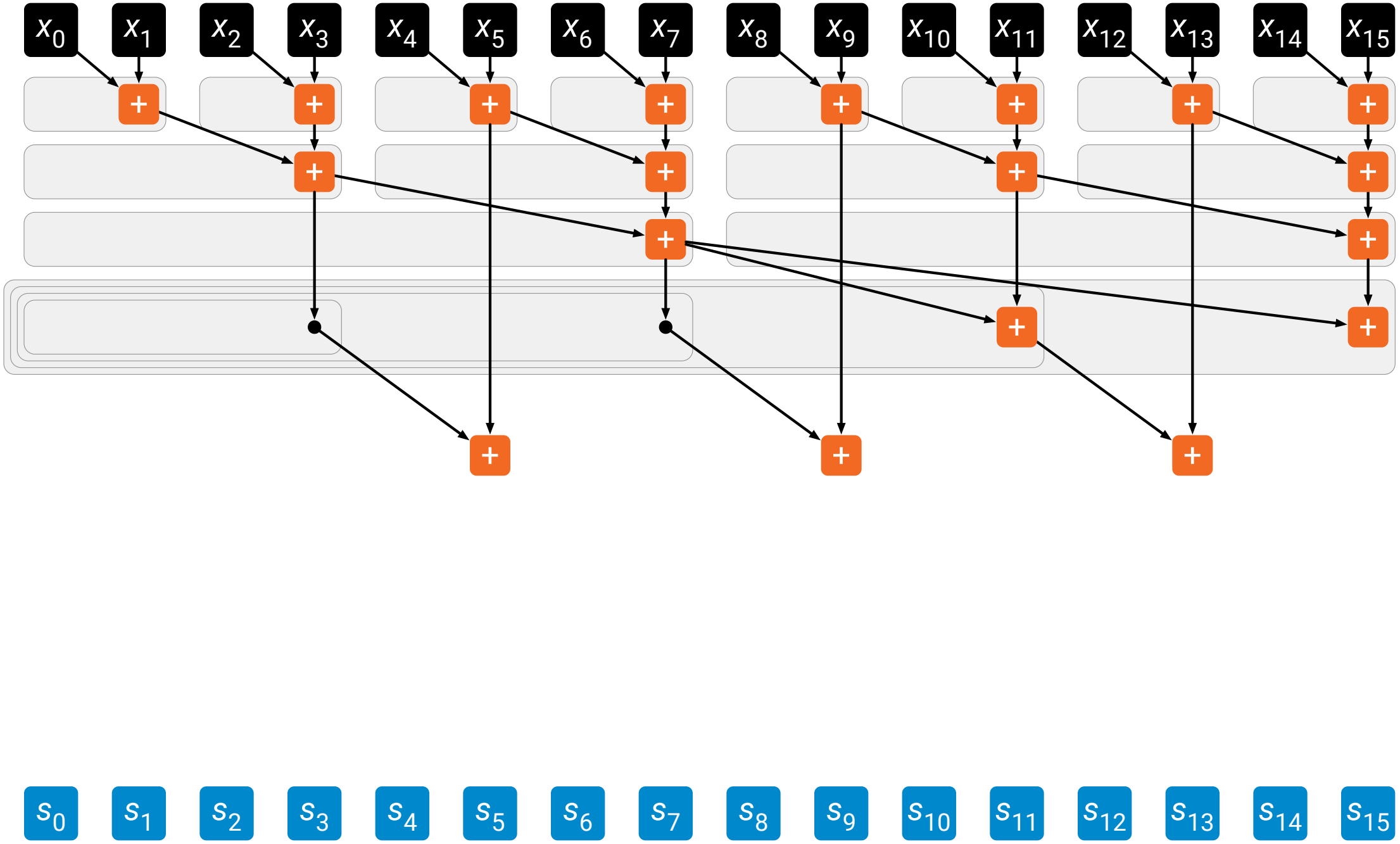

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

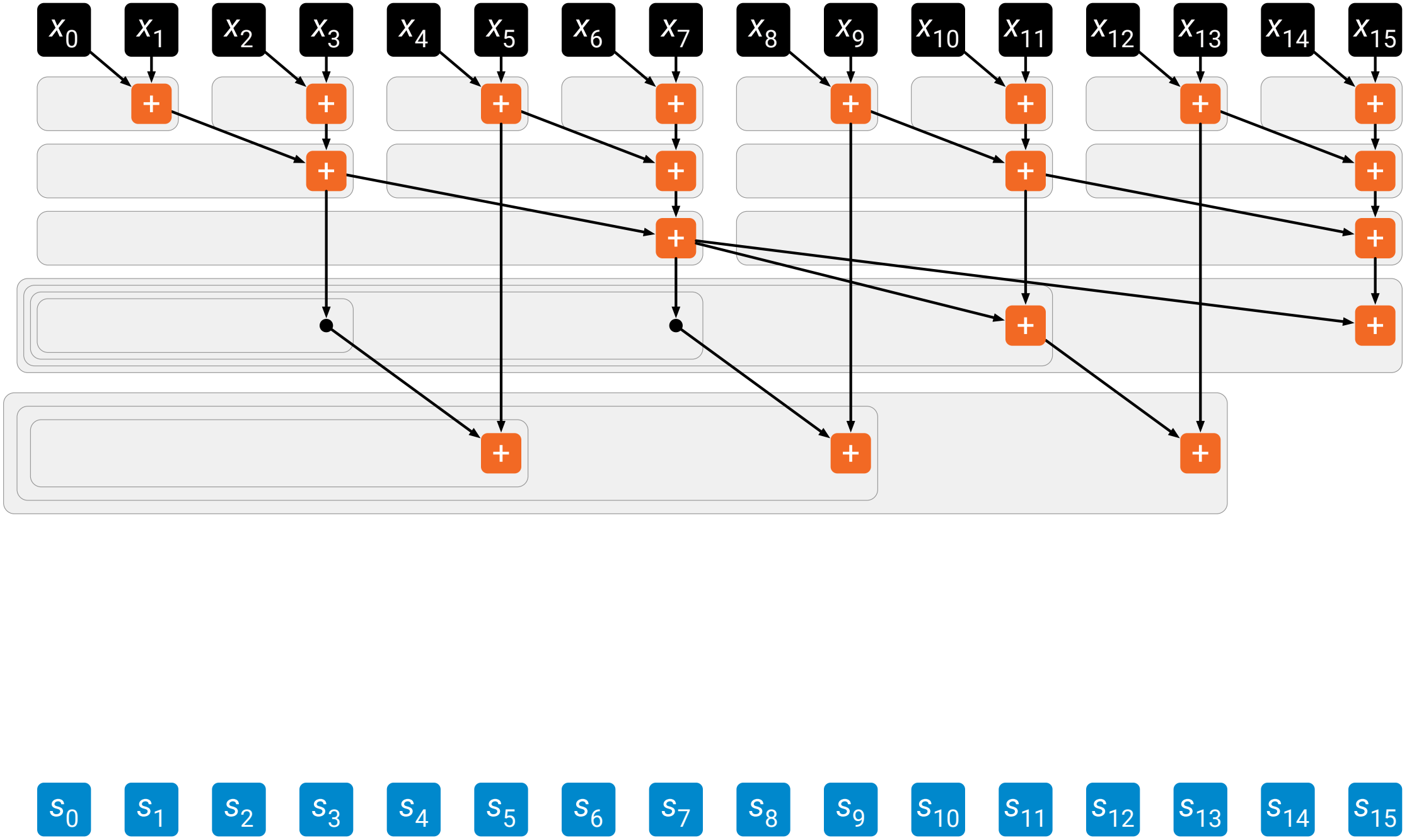

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

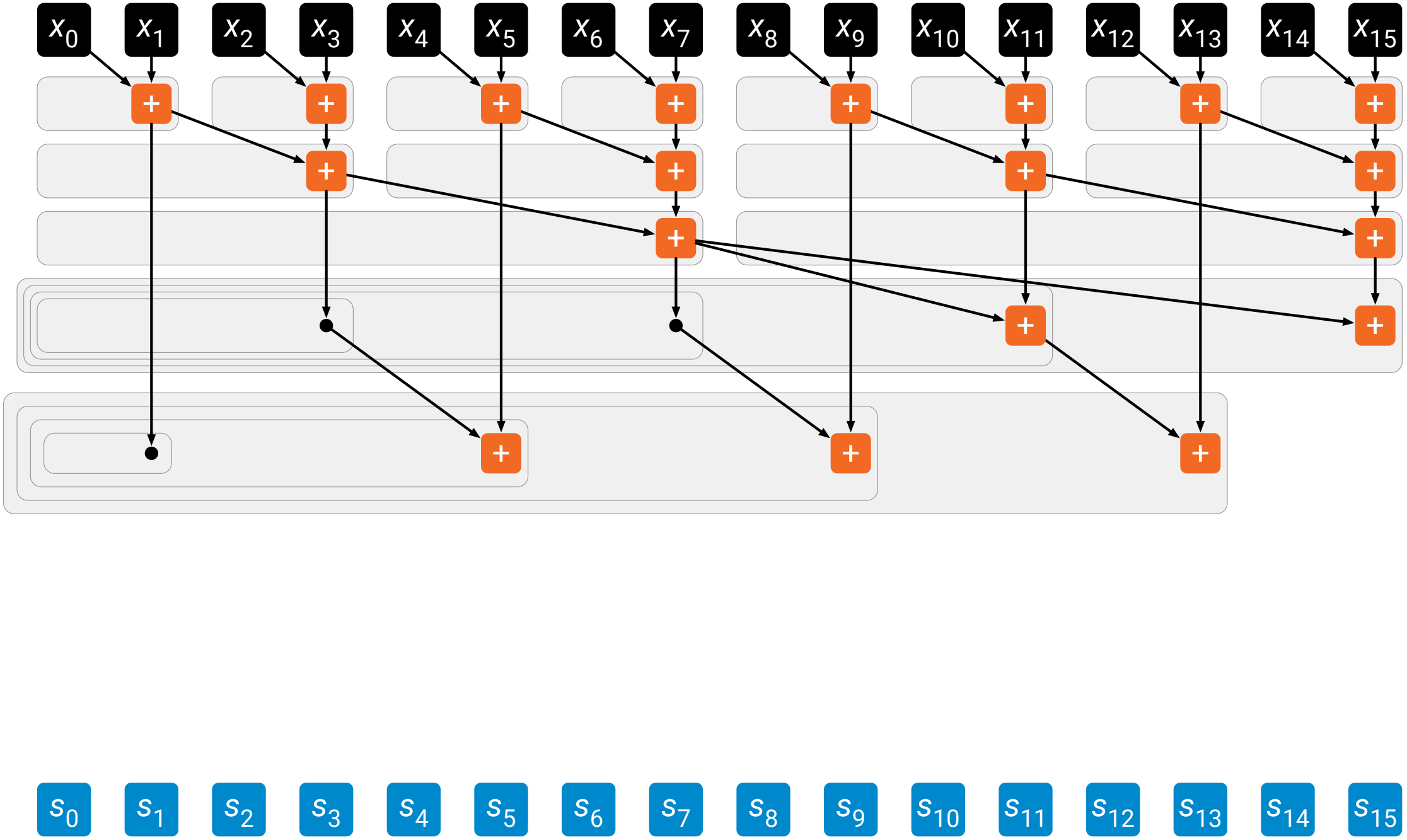

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

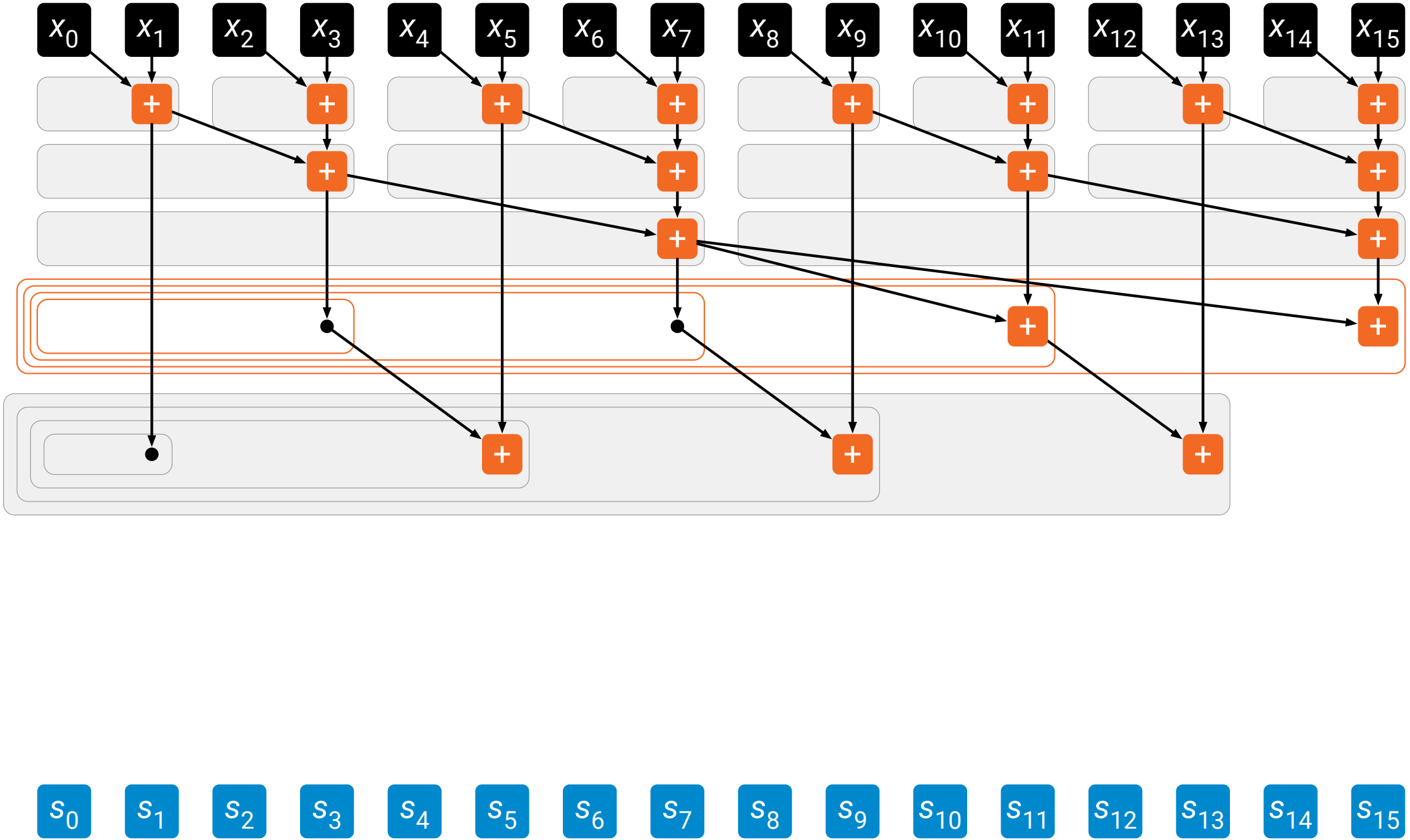


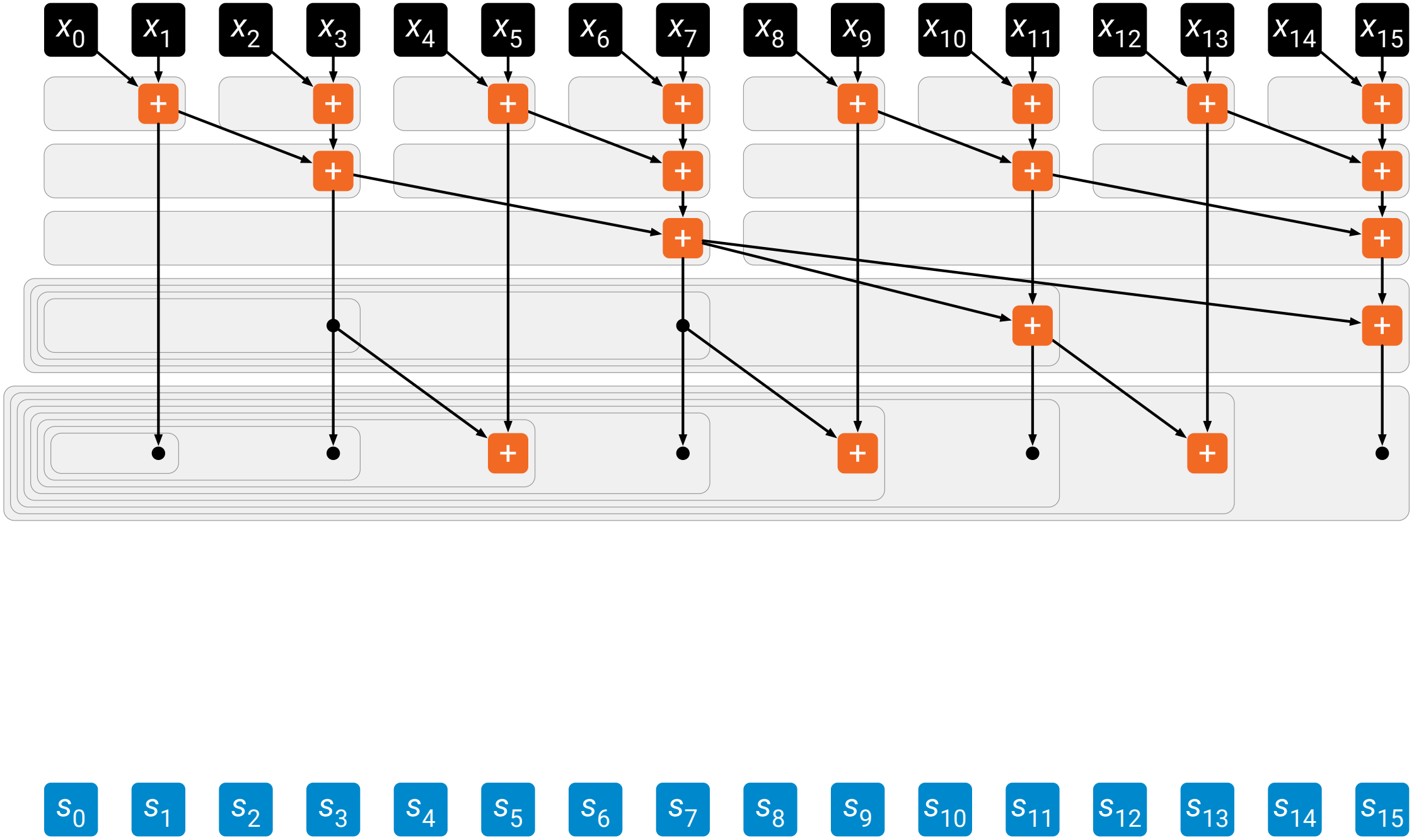
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

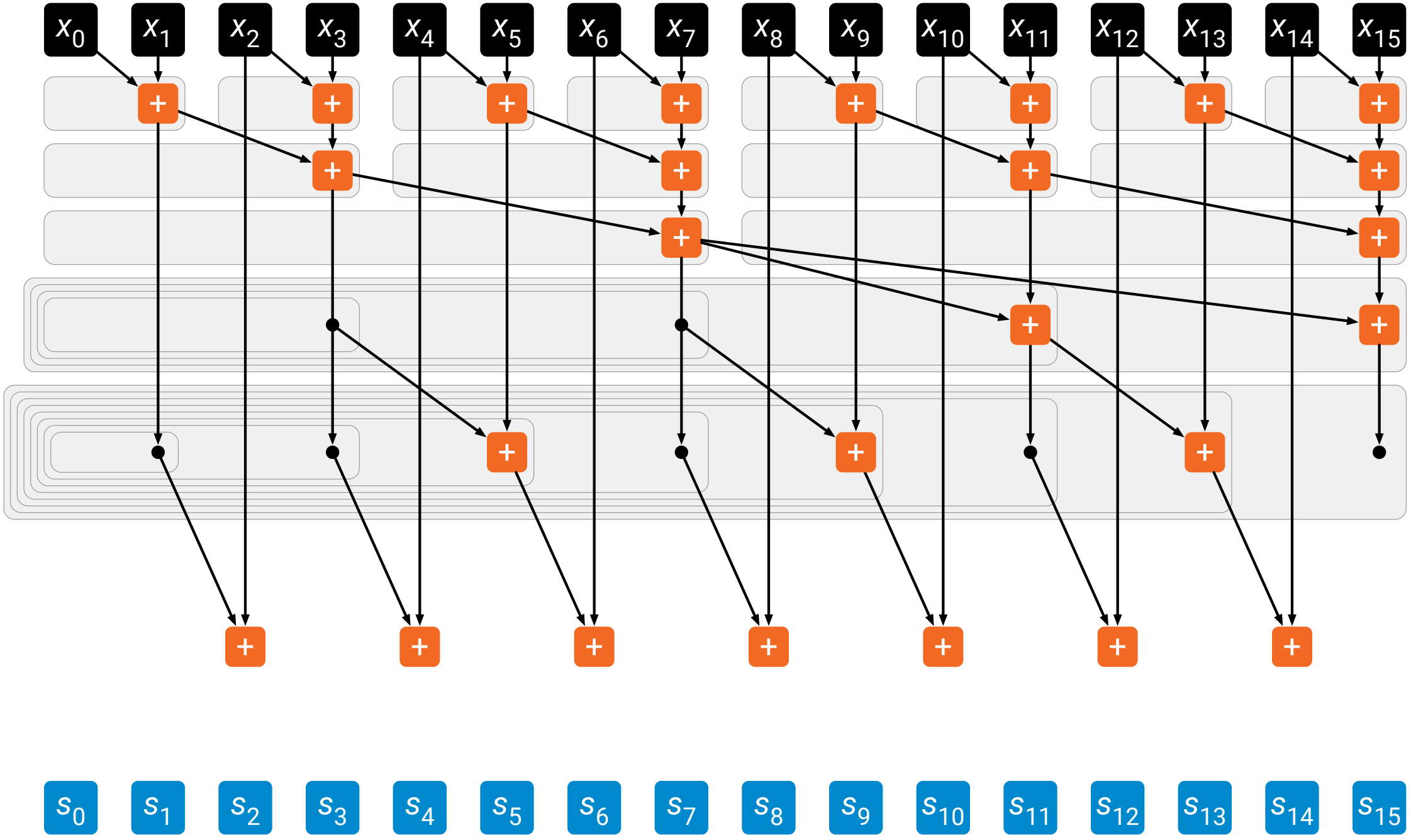

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

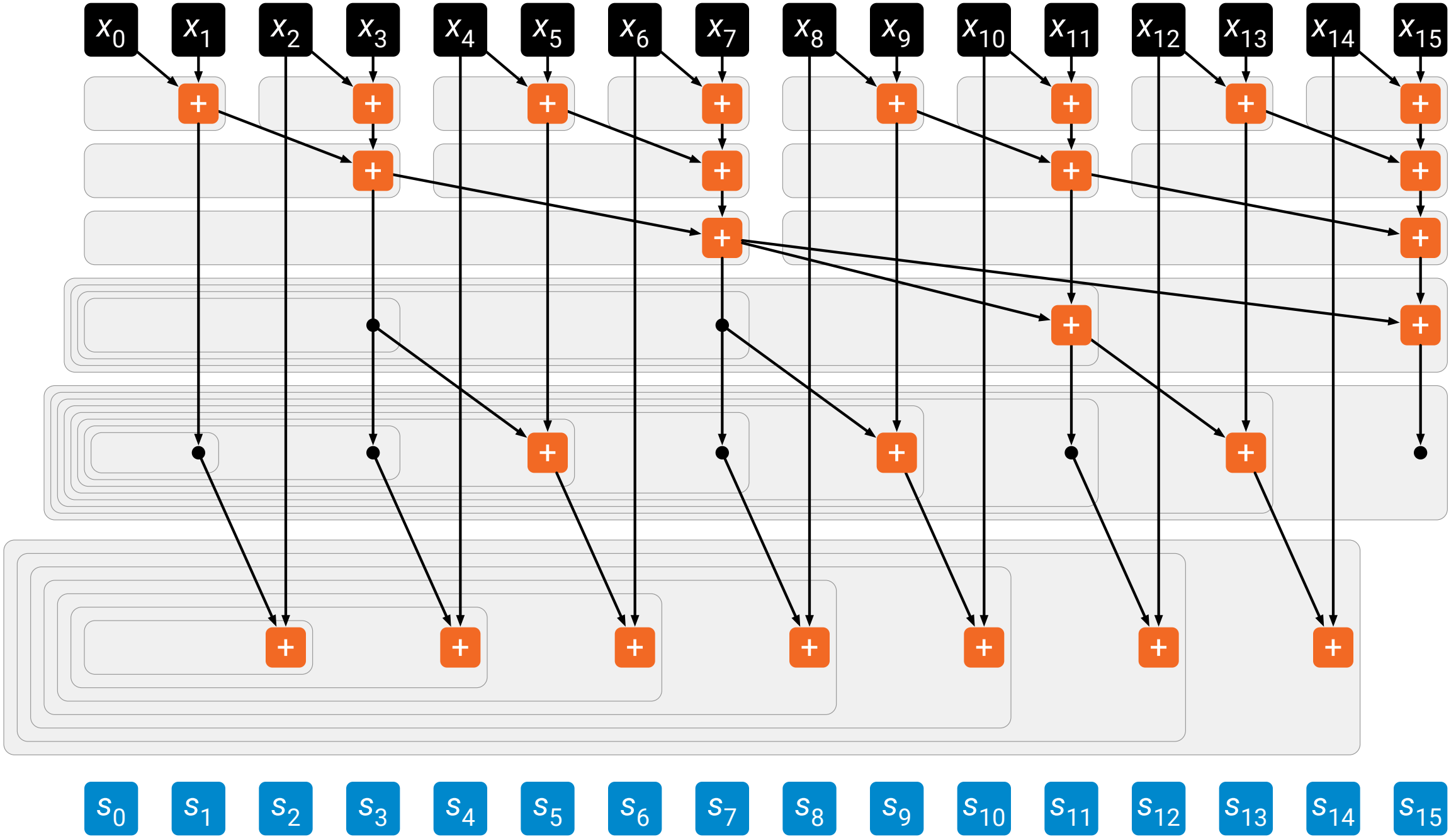

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

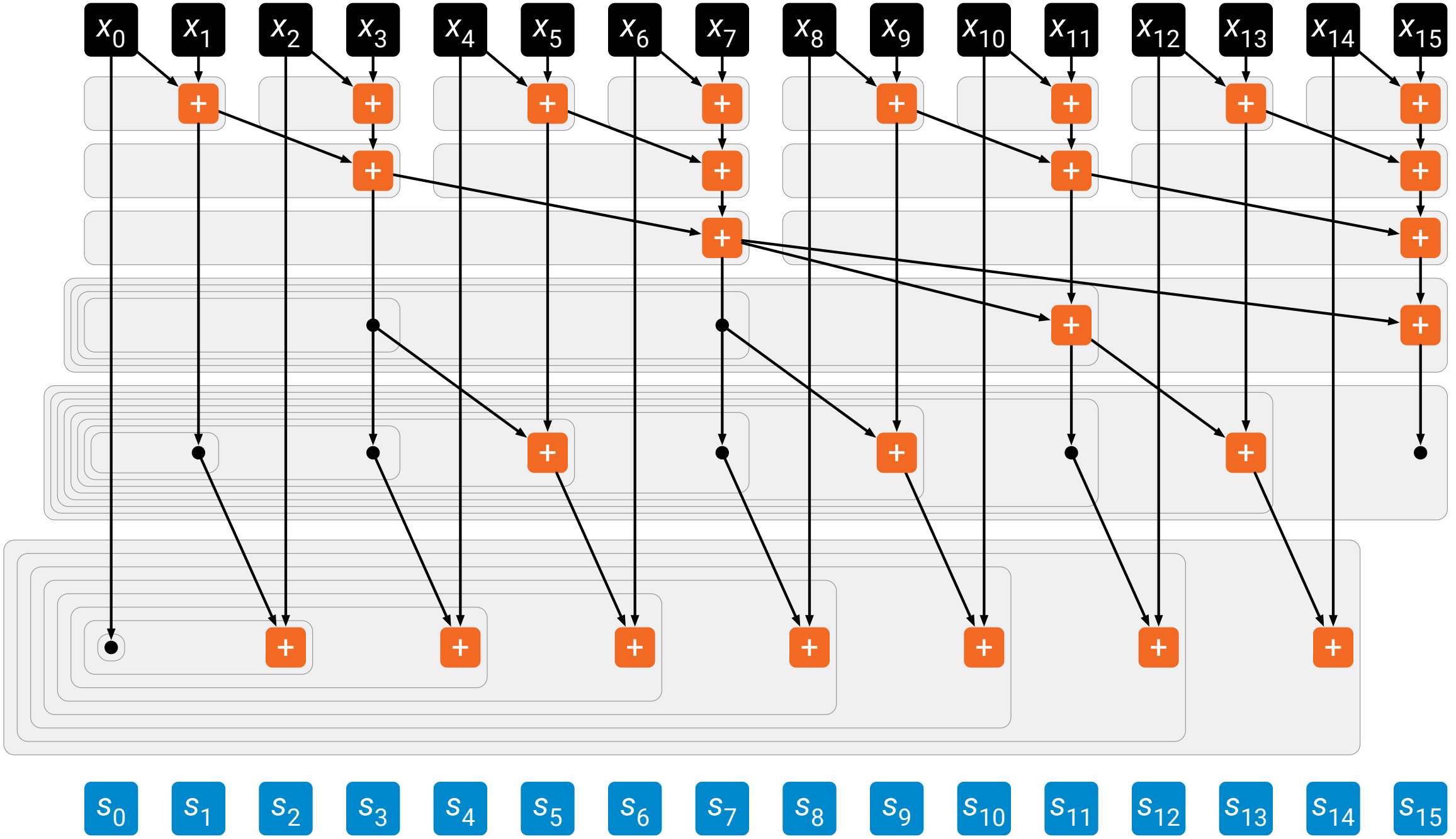

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

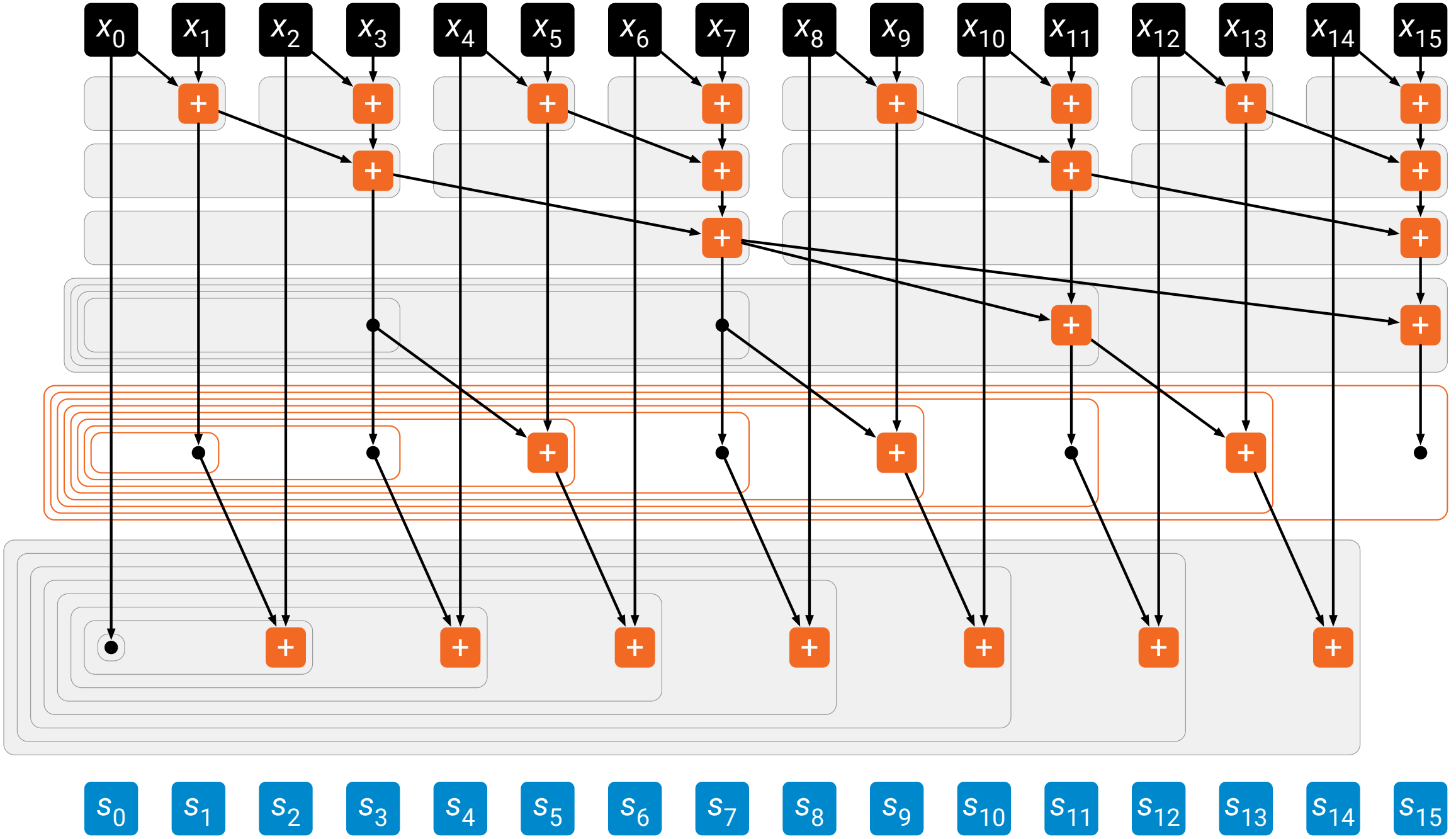

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

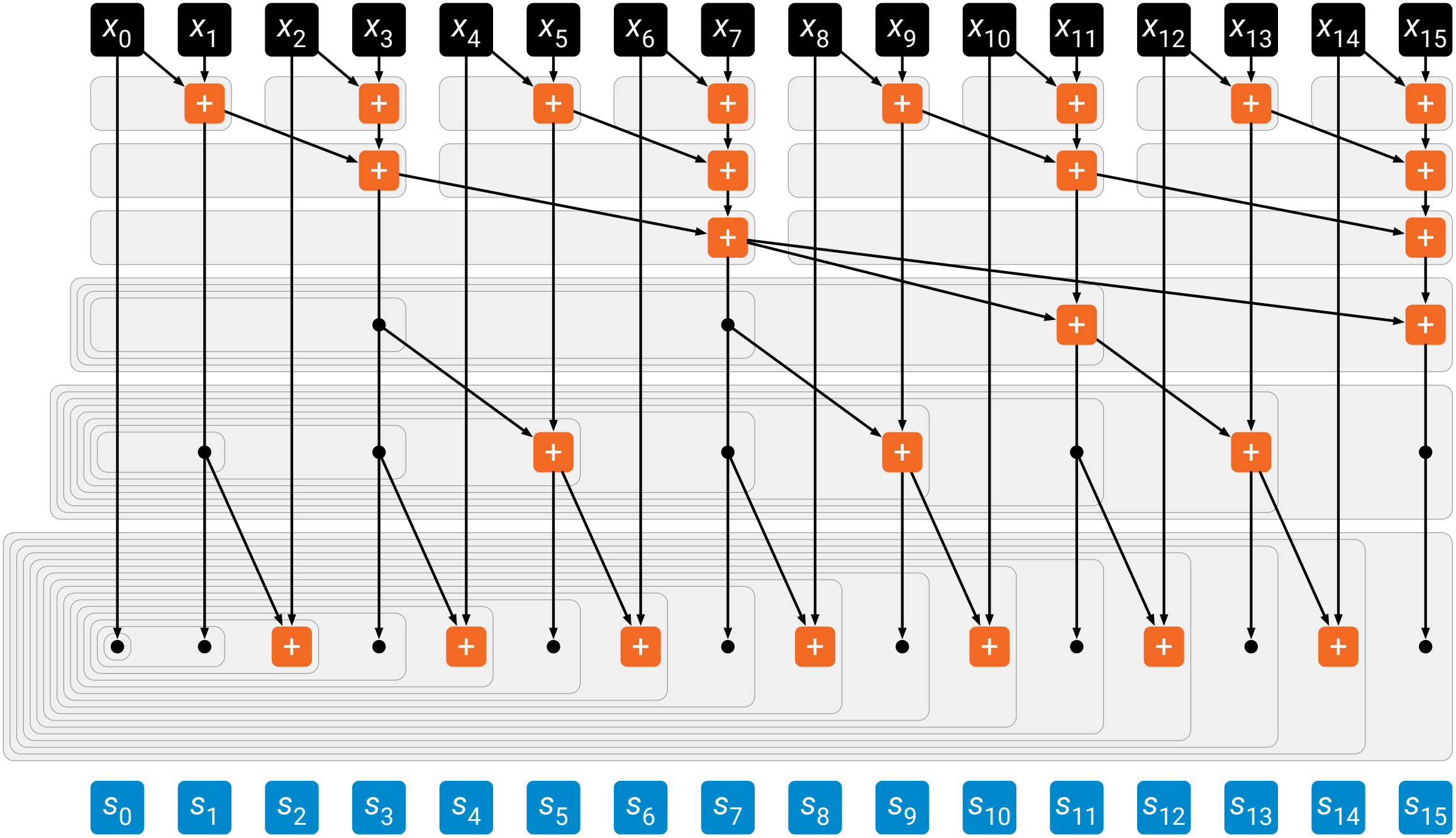

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
+
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

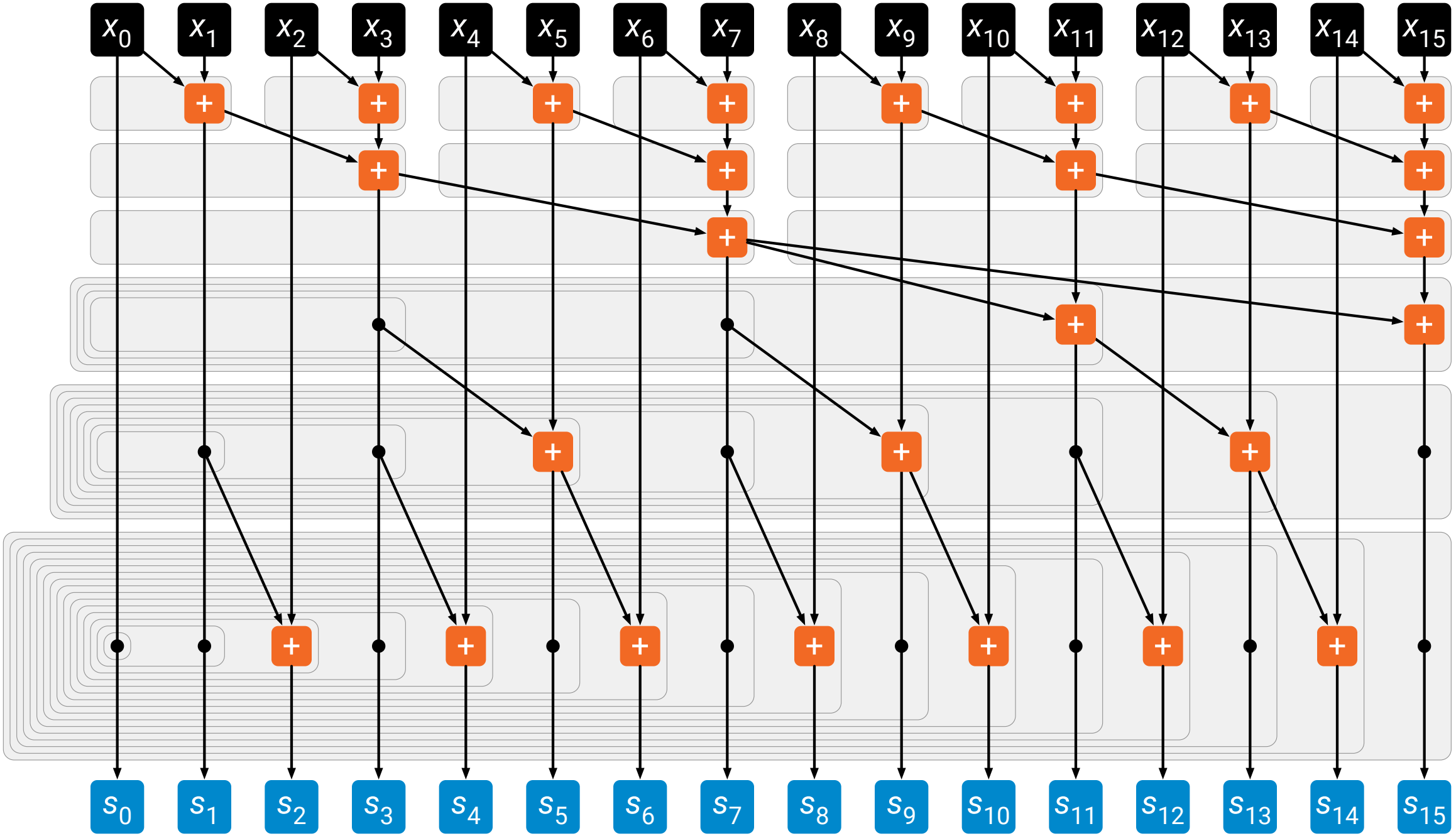

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}


x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}


x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}


x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}


x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}


x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

Prefix sums in parallel in practice

Prefix sum

- Simple practical implementation for *p* threads:

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:												
z:												
***s*:**												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts

x:	1	2	3	4	5	6	7	8	9	10	11	12
***y*:**												
***z*:**												
s:												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*

x:	1	2	3	4	5	6	7	8	9	10	11	12
***y*:**												
***z*:**												
***s*:**												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:	**1**				**5**				**9**			
z:												
s:												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:		**3**				**11**				**19**		
z:												
s:												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*

x:	1	2	3	4	5	6	7	8	9	10	11	12
***y*:**			**6**				**18**				**30**	
***z*:**												
***s*:**												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:				**10**				**26**				**42**
z:												
s:												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:				**10**				**26**				**42**
z:												
s:												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:				**10**				**26**				**42**
z:				**10**								
s:												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:				**10**				**26**				**42**
z:				**10**				**36**				
s:												

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:				**10**				**26**				**42**
z:				**10**				**36**				**78**
s:												

p = 3 threads
n = 12 input values

Prefix sum

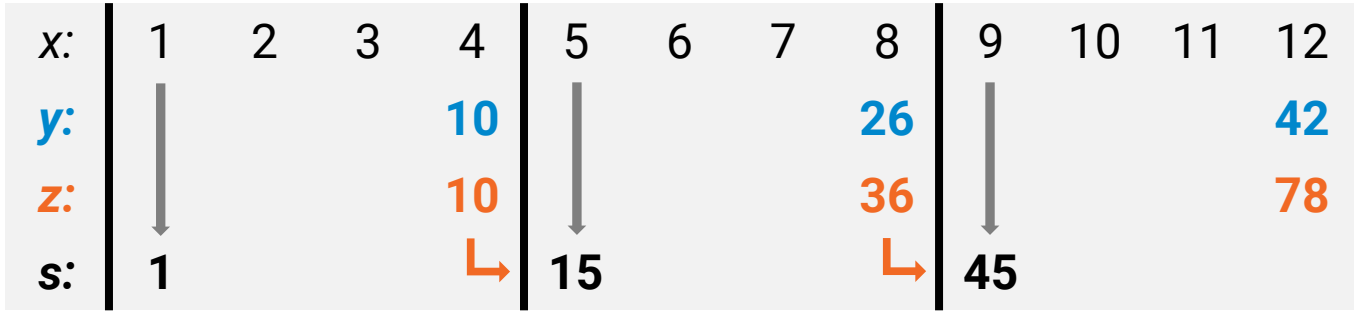
- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated
  - ***in parallel:*** calculate prefix sums for each part
    - part *i* uses z(*i*) as the initial value

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:				**10**				**26**				**42**
z:				**10**				**36**				**78**
s:												

p = 3 threads
n = 12 input values

Prefix sum

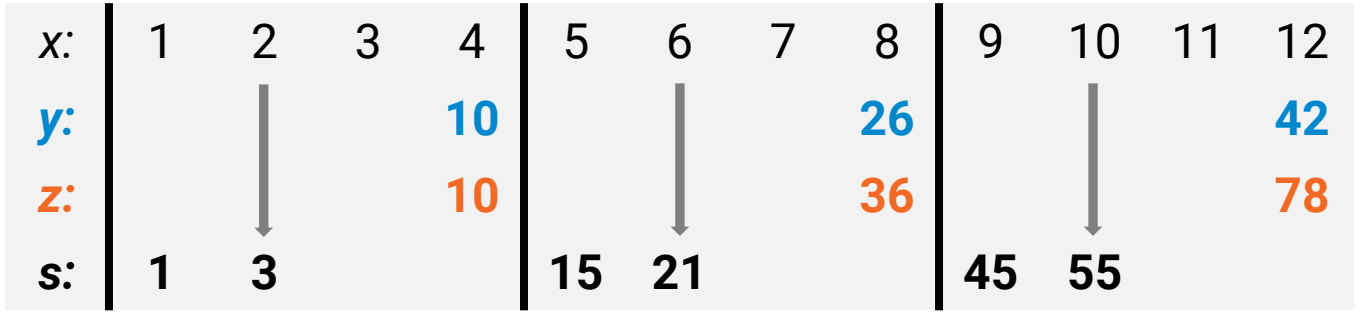
- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated
  - ***in parallel:*** calculate prefix sums for each part
    - part *i* uses z(*i*) as the initial value



p = 3 threads
n = 12 input values

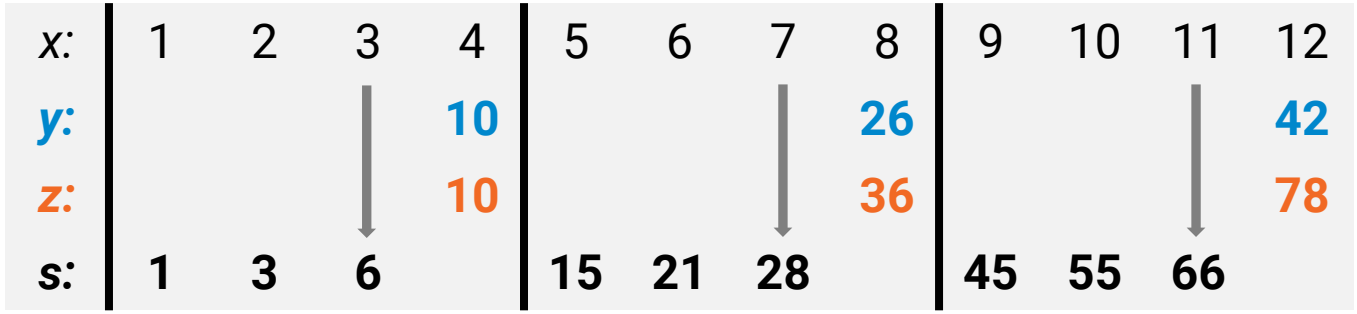
Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated
  - ***in parallel:*** calculate prefix sums for each part
    - part *i* uses z(*i*) as the initial value



Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated
  - ***in parallel:*** calculate prefix sums for each part
    - part *i* uses z(*i*) as the initial value



p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated
  - ***in parallel:*** calculate prefix sums for each part
    - part *i* uses z(*i*) as the initial value

x:	1	2	3	4	5	6	7	8	9	10	11	12
y:				**10**				**26**				**42**
z:				**10**				**36**				**78**
s:	**1**	**3**	**6**	**10**	**15**	**21**	**28**	**36**	**45**	**55**	**66**	**78**

p = 3 threads
n = 12 input values

Prefix sum

- Simple practical implementation for *p* threads:
  - split in *p* parts
  - ***in parallel:*** calculate ***y(i)*** = sum of part *i*
  - ***sequentially:*** calculate ***z(i)*** = sum of all parts up to *i*
    - using *y*(*i*) values that we just calculated
  - ***in parallel:*** calculate prefix sums for each part
    - part *i* uses z(*i*) as the initial value

Smaller prefix sum calculation, could be further parallelized if needed

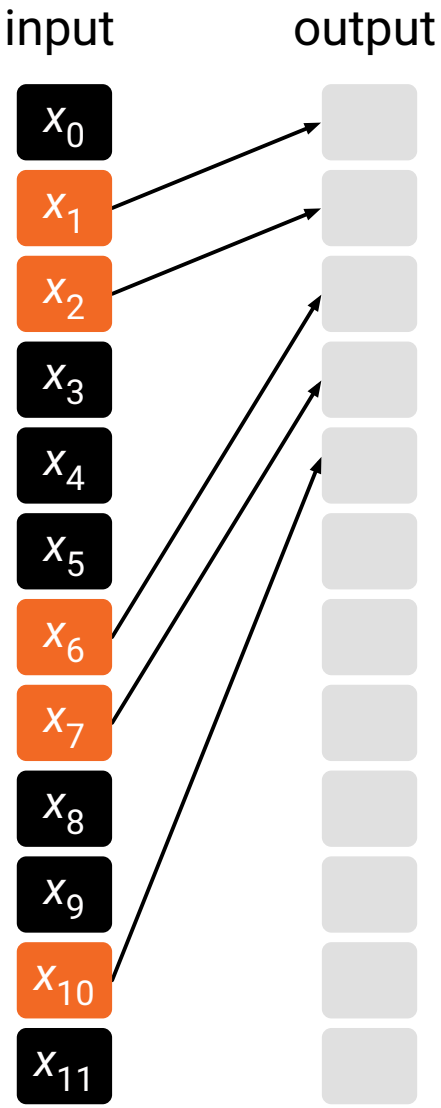
x:	1	2	3	4	5	6	7	8	9	10	11	12
y:				**10**				**26**				**42**
z:				**10**				**36**				**78**
s:	**1**	**3**	**6**	**10**	**15**	**21**	**28**	**36**	**45**	**55**	**66**	**78**

p = 3 threads
n = 12 input values

Using parallel prefix sum to solve other problems

Select

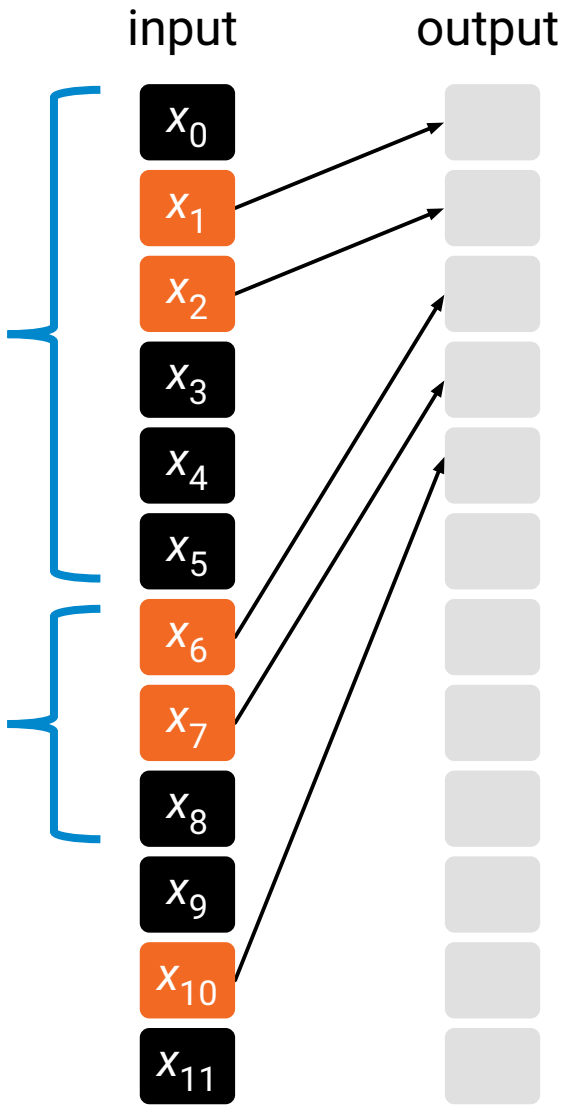
- Find all ***orange*** elements and put them in the output array in consecutive positions
  - cf. **“partition”** in quicksort
- Trivial sequential algorithm
- How to parallelize?



Select

If we know how many orange elements are here…

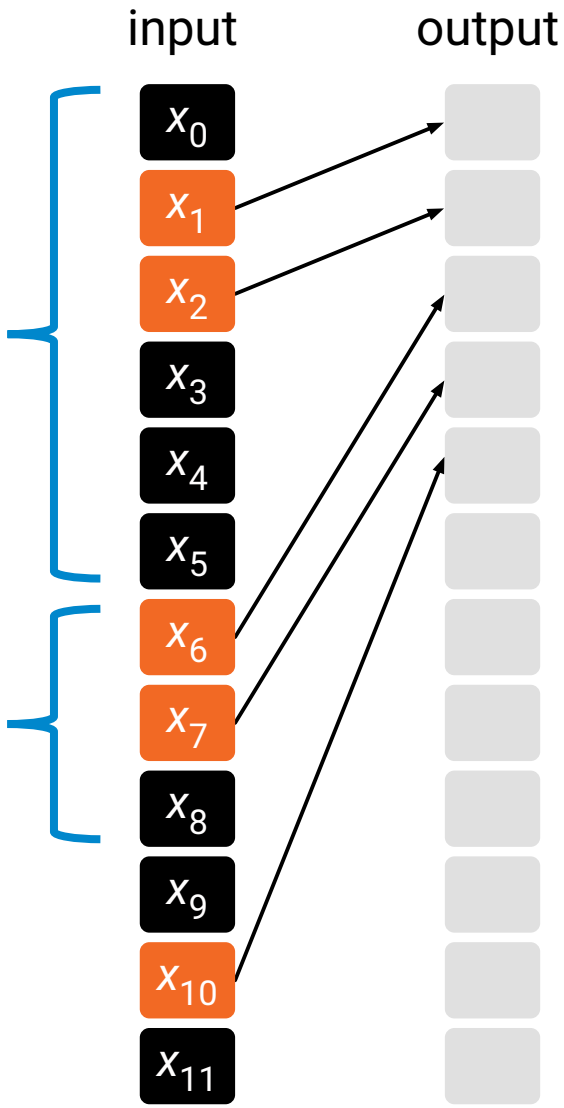
… we know where to put these elements



Select

If we know how many orange elements are here…

… we know where to put these elements



Special case of prefix sum!