October 30, 2025

6.S894
Accelerated Computing

Lecture 9: Data-Parallel Primitives

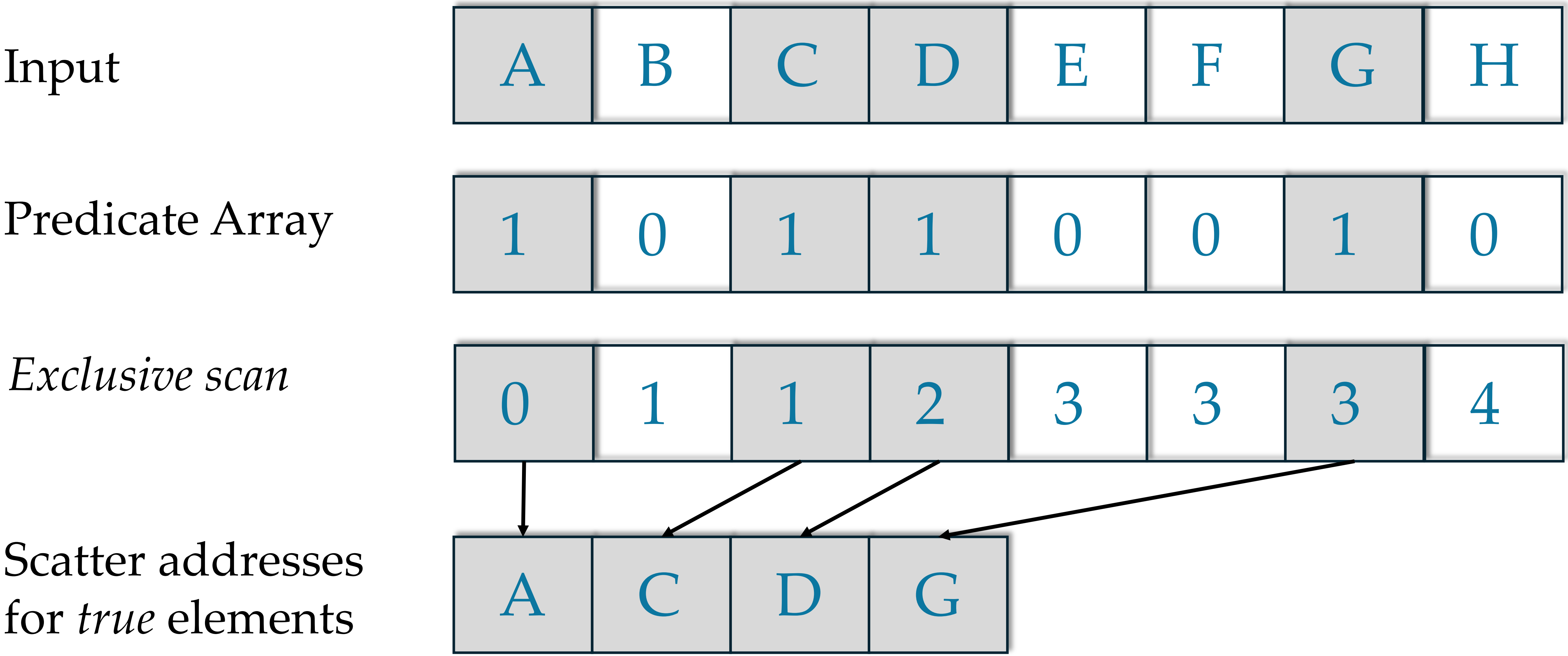
Ahmed Mahmoud

Outline:

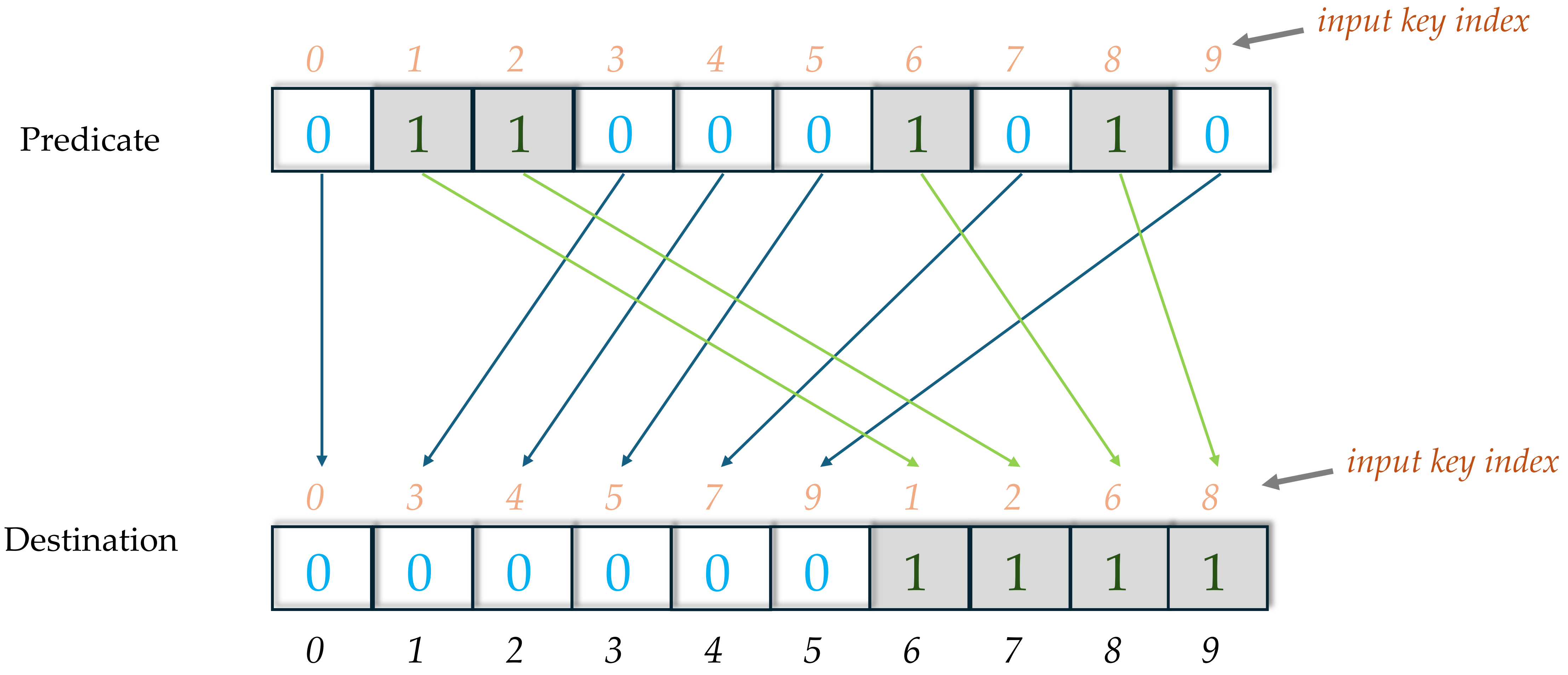
- Map
- Stencil
- Reduce
- Compaction
- Scan
  - Segmented Scan
- Histogram
- Merge
- Reduction using tensor core!

Main idea: high-performance parallel implementations of data-parallel primitives exist, allowing programs written with these primitives to run efficiently on the GPU.

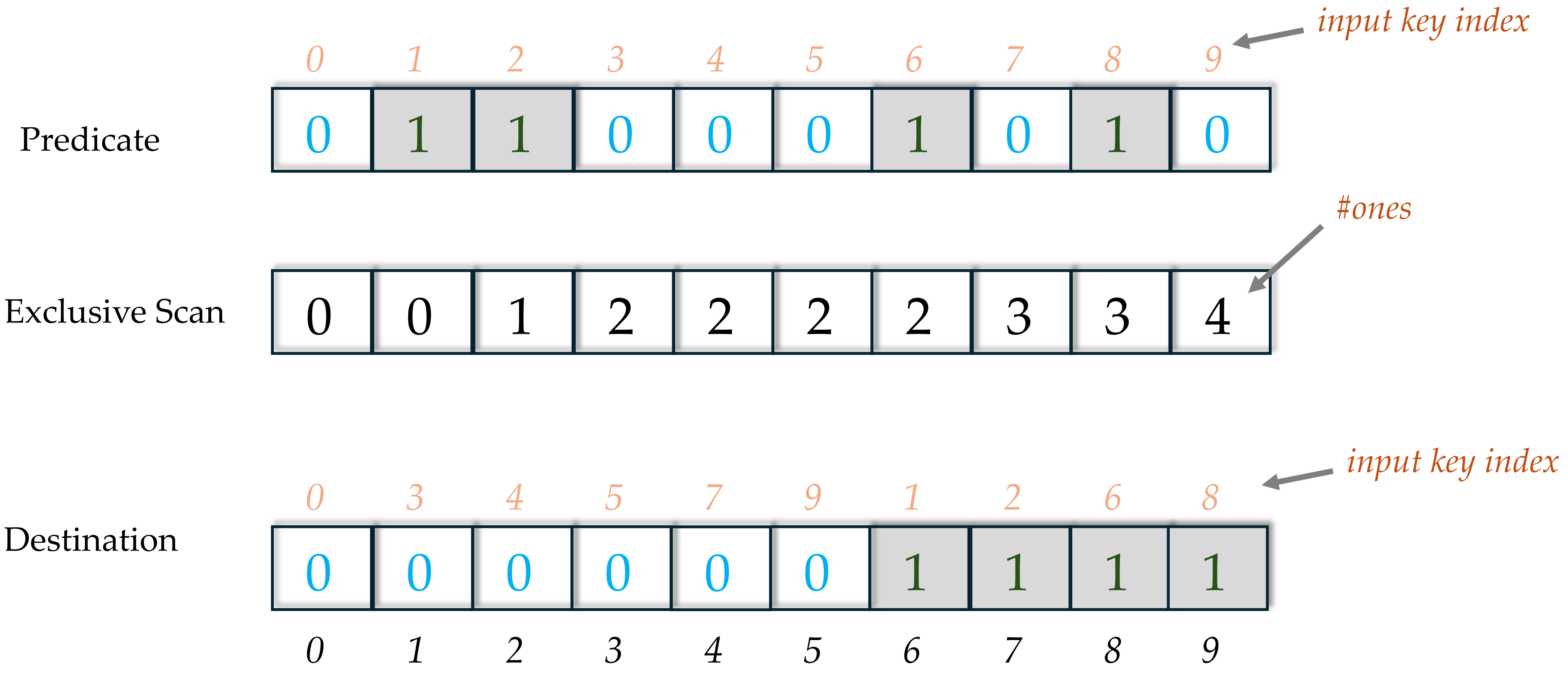
Pattern #4: Stream Compaction



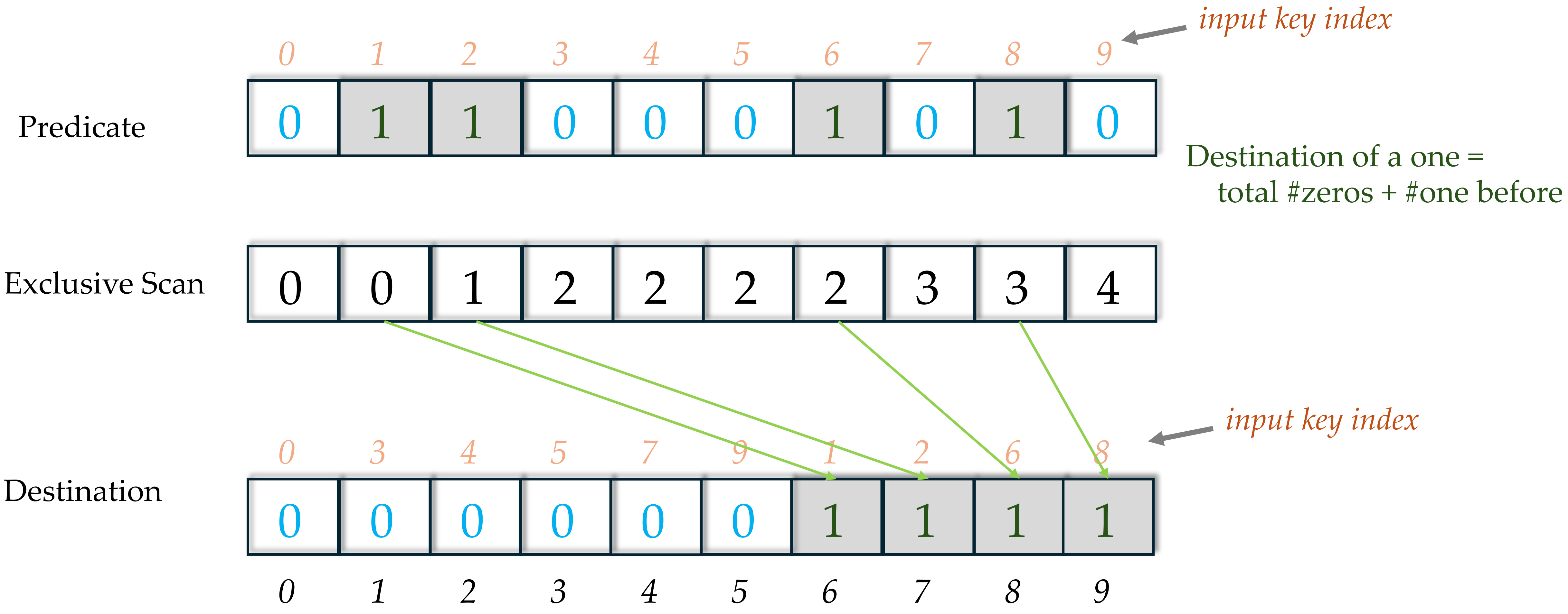
Pattern #4.5: Split



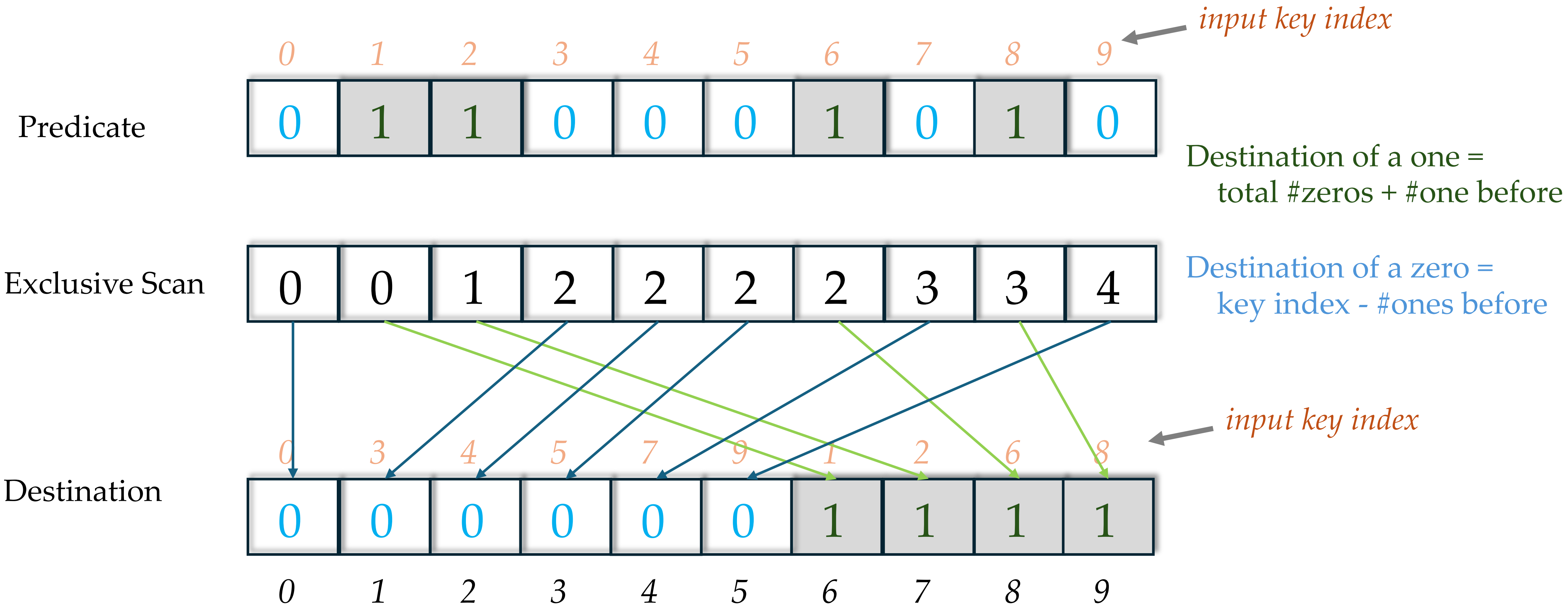
Pattern #4.5: Split



Pattern #4.5: Split



Pattern #4.5: Split



Load-balance Search

"Coordinates work-items with the objects that scheduled them, allowing perfect load-balancing for functions that expand and contract data"

[A B C D E F] Alphabet (#size 6)

[2 1 0 0 7 3] Count

Load-balance Search

“Coordinates work-items with the objects that scheduled them, allowing perfect load-balancing for functions that expand and contract data”

[A B C D E F] Alphabet (#size 6)

[2 1 0 0 7 3] Count

[A A B E E E E E E E F F F] Output

https://moderngpu.github.io/intro.html

Load-balance Search

"Coordinates work-items with the objects that scheduled them, allowing perfect load-balancing for functions that expand and contract data"

[A B C D E F] Alphabet (#size 6)

[2 1 0 0 7 3] Count

[0 0 1 4 4 4 4 4 4 4 5 5 5]

[A A B E E E E E E E F F F] Output

https://moderngpu.github.io/intro.html

Load-balance Search

"Coordinates work-items with the objects that scheduled them, allowing perfect load-balancing for functions that expand and contract data"

[A B C D E F] Alphabet (#size 6)

[2 1 0 0 7 3] Count

[0 2 3 3 3 10 13] Count Scanned

[0 0 1 4 4 4 4 4 4 4 5 5 5]

[A A B E E E E E E E F F F] Output

https://moderngpu.github.io/intro.html

Load-balance Search

"Coordinates work-items with the objects that scheduled them, allowing perfect load-balancing for functions that expand and contract data"

[A B C D E F] Alphabet (#size 6)

[2 1 0 0 7 3] Count

[0 2 3 3 3 10 13] Count Scanned

[0 0 0 0 0 0 0 0 0 0 0 0 0] init with zero

[0 0 1 4 4 4 4 4 4 4 5 5 5]

[A A B E E E E E E E F F F] Output

https://moderngpu.github.io/intro.html

Load-balance Search

"Coordinates work-items with the objects that scheduled them, allowing perfect load-balancing for functions that expand and contract data"

[A B C D E F] Alphabet (#size 6)

[2 1 0 0 7 3] Count

[0 2 3 3 3 10 13] Count Scanned

[0 0 0 0 0 0 0 0 0 0 0 0 0] init with zero

[0 0 1 4 0 0 0 0 0 0 5 0 0] Scatter Alphabet *index* based on Count Scanned if Count != zero

[0 0 1 4 4 4 4 4 4 4 5 5 5]

[A A B E E E E E E E F F F] Output

https://moderngpu.github.io/intro.html

Load-balance Search

"Coordinates work-items with the objects that scheduled them, allowing perfect load-balancing for functions that expand and contract data"

[A B C D E F] Alphabet (#size 6)

[2 1 0 0 7 3] Count

[0 2 3 3 3 10 13] Count Scanned

[0 0 0 0 0 0 0 0 0 0 0 0 0] init with zero

[0 0 1 4 0 0 0 0 0 0 5 0 0] Scatter Alphabet *index* based on Count Scanned if Count != zero

[0 0 1 4 4 4 4 4 4 4 5 5 5] Scan with *max()* operator

[A A B E E E E E E E F F F] Output

https://moderngpu.github.io/intro.html

Pattern #5.5: Segmented Scan

- Simultaneously performs scans on arbitrary contiguous partitions of input sequences

- Example for when $\oplus$ is addition

Input	3	1	7	0	4	1	6	3
Inclusive Segmented Scan	3	4	11	0	4	5	6	9

Pattern #5.5: Segmented Scan

- Simultaneously performs scans on arbitrary contiguous partitions of input sequences
- Same computational complexity as scan, but additionally must keep track of segments using ***head flags***

Input	3	1	7	0	4	1	6	3
Head flag	1	0	0	1	0	0	1	0
Inclusive Segmented Scan	3	4	11	0	4	5	6	9

Pattern #5.5: Segmented Scan

Up-sweep:

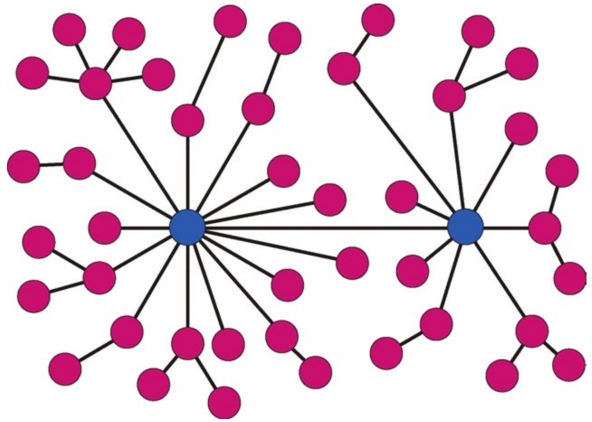
```
for d=0 to (log2n - 1) do:
  forall k=0 to n-1 by 2^(d+1) do:
    if flag[k + 2^(d+1) - 1] == 0:
      data[k + 2^(d+1) - 1] = data[k + 2^d - 1] + data[k + 2^(d+1) - 1]
    flag[k + 2^(d+1) - 1] = flag[k + 2^d - 1] || flag[k + 2^(d+1) - 1]
```

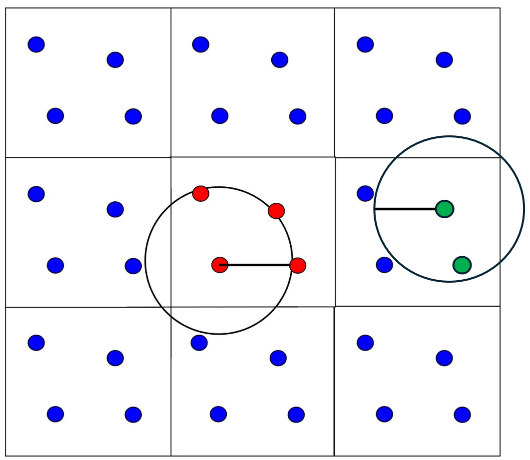
Down-sweep:

```
data[n-1] = 0
for d=(log2n - 1) down to 0 do:
  forall k=0 to n-1 by 2^(d+1) do:
    tmp = data[k + 2^d - 1]
    data[k + 2^d - 1] = data[k + 2^(d+1) - 1]
    if flag_original[k + 2^d] == 1:   # must maintain copy of original flags
      data[k + 2^(d+1) - 1] = 0       # start of segment
    else if flag[k + 2^d - 1] == 1:
      data[k + 2^(d+1) - 1] = tmp
    else:
          data[k + 2^(d+1) - 1] = tmp + data[k + 2^(d+1) - 1]
      flag[k + 2^d - 1] = 0
```

Pattern #5.5: Segmented Scan

- Used in operating on sequence of *variable-length* sequences
  1. For each vertex in a graph:
     - For each edge incoming to the vertex
  2. For each particle in simulation:
     - For each particle within a cutoff radius
  3. For each document in a collection
     - For each word in the document





Pattern #5.5: Segmented Scan

- Used in operating on sequence of *variable-length* sequences
  1. For each vertex in a graph:
     - For each edge incoming to the vertex
  2. For each particle in simulation:
     - For each particle within a cutoff radius
  3. For each document in a collection
     - For each word in the document
- Two levels of parallelism to exploit but it's *irregular*:
  1. The size of edge lists
  2. Particle neighbor list
  3. Words per document

Pattern #5.5: Segmented Scan

Quicksort

1. Pick a *pivot*
2. Partition remaining elements into two sub-arrays
   - Elements ≤ pivot
   - Elements > pivot
3. Repeat for each sub-array until sorted

Pattern #5.5: Segmented Scan

Quicksort

previous pivot

[3 1 4 2 6 9 8 9] Input

Blelloch "Prefix sums and their applications." (1990).

Pattern #5.5: Segmented Scan

Quicksort

[3 1 4 2 X 9 8 9] Input

Blelloch "Prefix sums and their applications." (1990).

Pattern #5.5: Segmented Scan

Quicksort

[3	1	4	2	X	9	8	9]	Input
[1	0	0	0	X	1	0	0]	Flags

Blelloch "Prefix sums and their applications." (1990).

Pattern #5.5: Segmented Scan

Quicksort

[3	1	4	2	X	9	8	9]	Input
[1	0	0	0	X	1	0	0]	Flags
[3	0	0	0	X	9	0	0]	Heads write pivot

Blelloch "Prefix sums and their applications." (1990).

Pattern #5.5: Segmented Scan

Quicksort

[3	1	4	2	X	9	8	9]	Input
[1	0	0	0	X	1	0	0]	Flags
[3	0	0	0	X	9	0	0]	Heads write pivot
[3	3	3	3	X	9	9	9]	Pivot (*max-scan*)

Blelloch "Prefix sums and their applications." (1990).

Pattern #5.5: Segmented Scan

Quicksort

[3	1	4	2	X	9	8	9]	Input
[1	0	0	0	X	1	0	0]	Flags
[3	0	0	0	X	9	0	0]	Heads write pivot
[3	3	3	3	X	9	9	9]	Pivot (*max-scan*)
[=	<	>	<	X	=	<	=]	Compare

Pattern #5.5: Segmented Scan

Quicksort

[3	1	4	2	X	9	8	9]	Input
[1	0	0	0	X	1	0	0]	Flags
[3	0	0	0	X	9	0	0]	Heads write pivot
[3	3	3	3	X	9	9	9]	Pivot (*max-scan*)
[=	<	>	<	X	=	<	=]	Compare
[1	2	3	4	X	8	9	9]	2-way Seg Split (req: segmented scan)

Blelloch "Prefix sums and their applications." (1990).

Pattern #6: Histogram

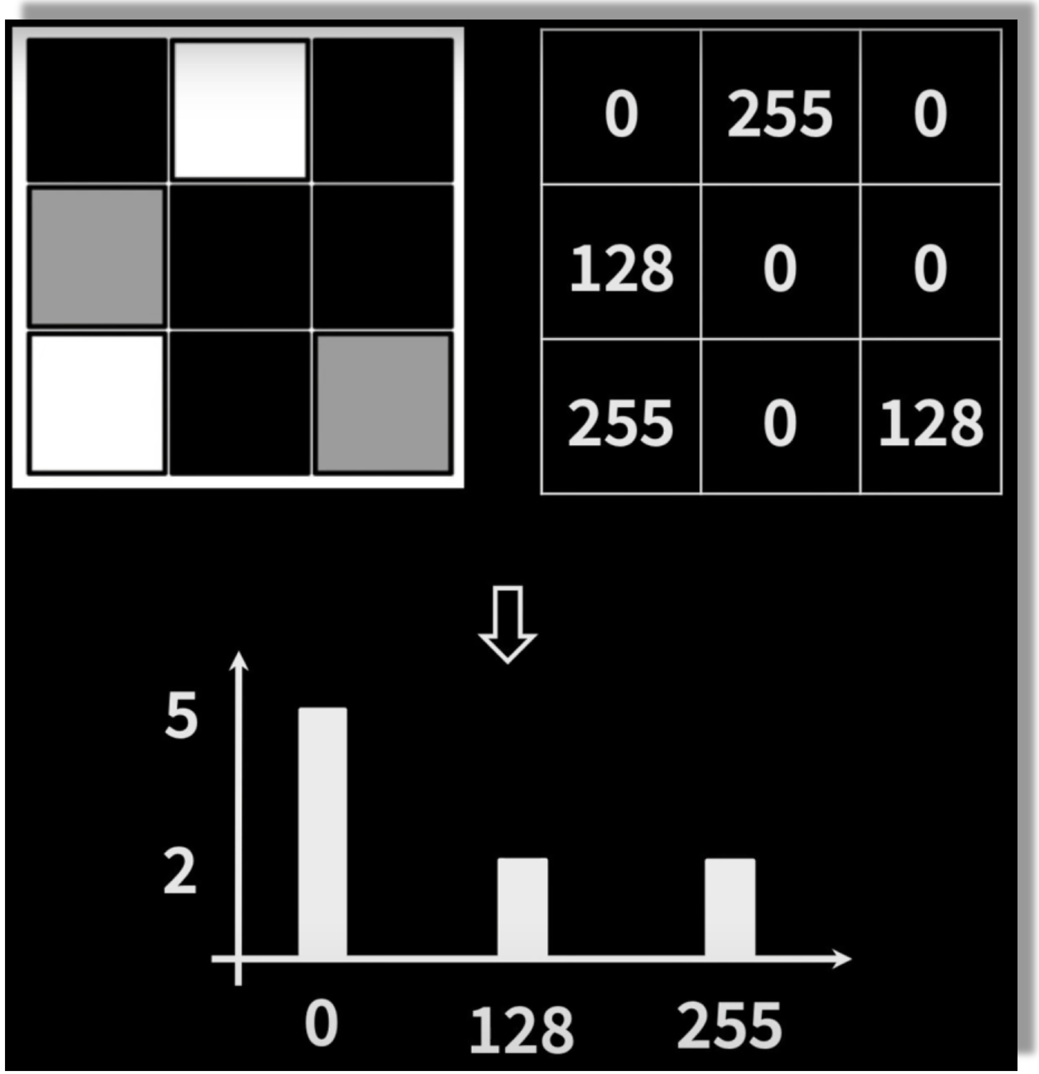
Given:

- Sequence of data elements A and sequence of bins B
- Function f(x) that assigns A's elements into the bins B

- *Histogram(A, B, f)* = populates B with A's elements according to f

Pattern #6: Histogram

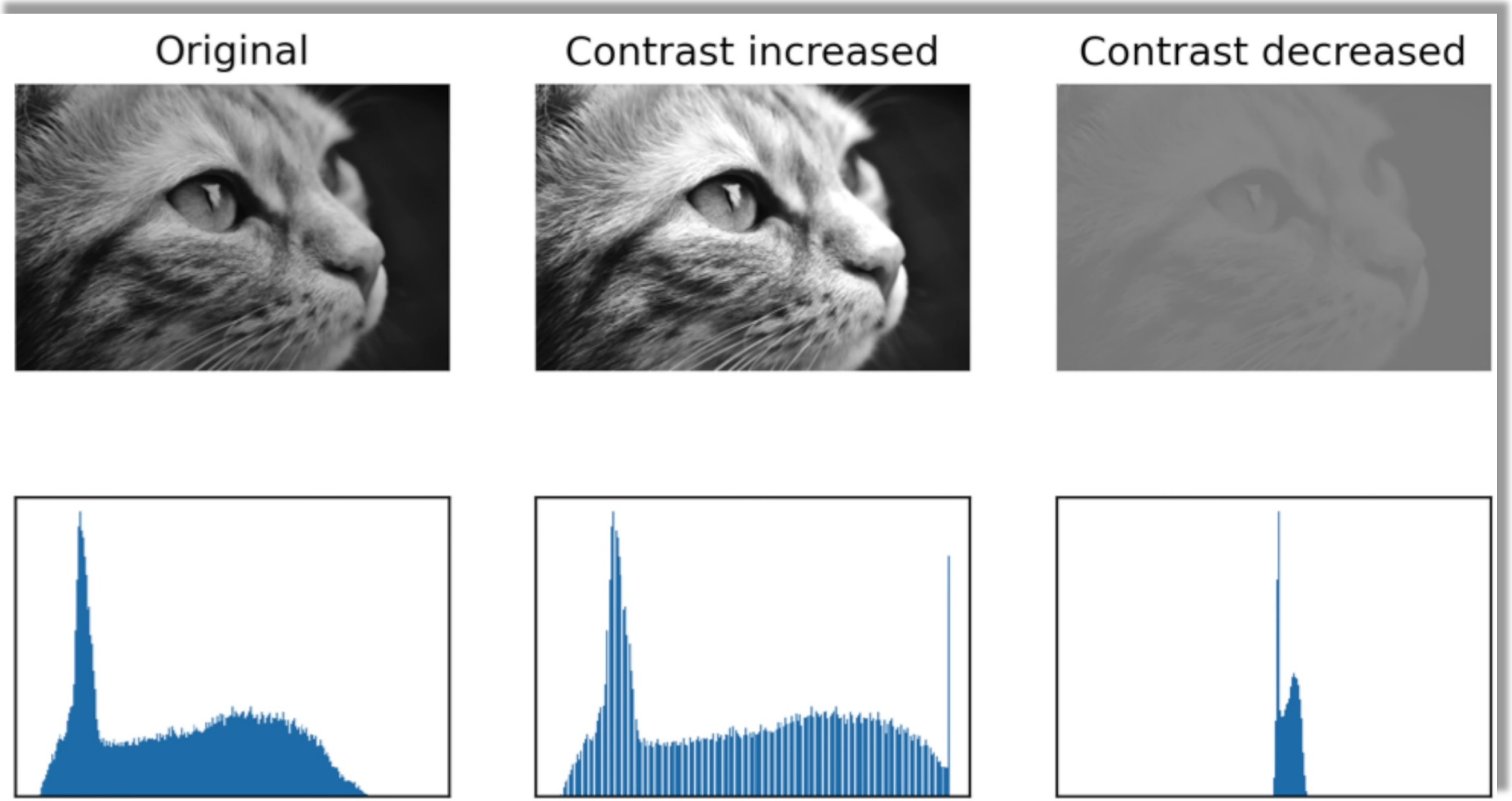
Example: image intensity values



https://www.youtube.com/watch?v=flI_Umo_VAU

Pattern #6: Histogram

Example: image intensity values



https://www.youtube.com/watch?v=flI_Umo_VAU

Pattern #6: Histogram

Example: Parallel Top-k

K=4

		1100	0100	0001	1000	0110	0101	1101	0000	1110
Input	**Value**	**12**	**4**	**1**	**8**	**6**	**5**	**13**	**0**	**14**
	Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Zhang, Naruse, Li, and Wang. "Parallel Top-K Algorithms on GPU: A Comprehensive Study and New Methods" SC '23

Pattern #6: Histogram

Example: Parallel Top-k

K=4

		1100	0100	0001	1000	0110	0101	1101	0000	1110
Input	**Value**	**12**	**4**	**1**	**8**	**6**	**5**	**13**	**0**	**14**
	Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

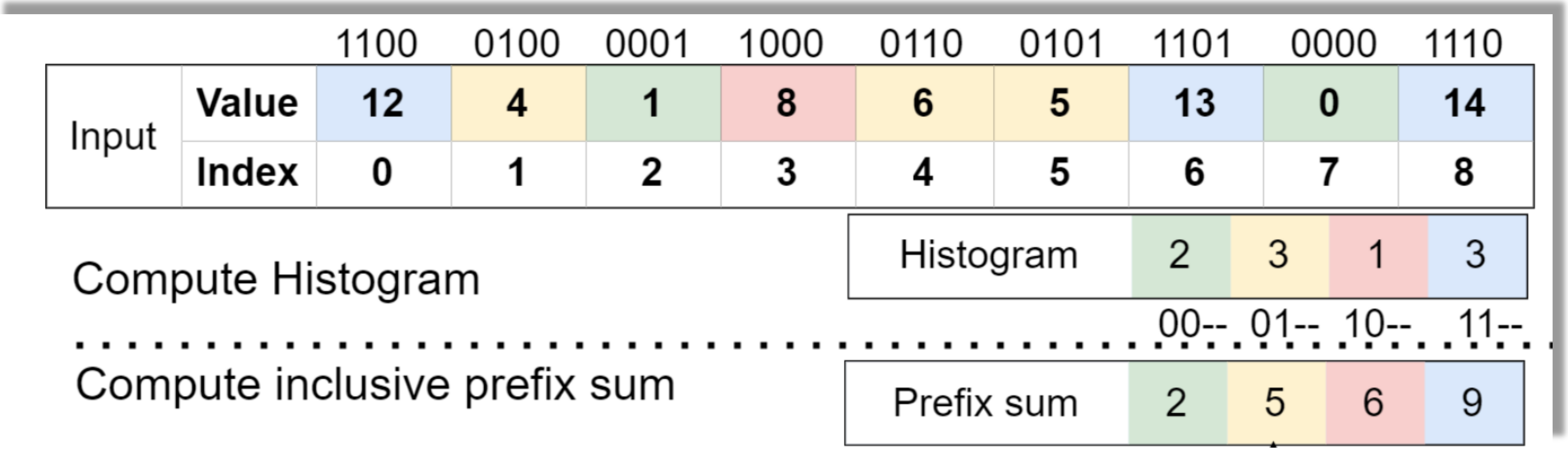
Compute Histogram

	00--	01--	10--	11--
Histogram	2	3	1	3

Zhang, Naruse, Li, and Wang. “Parallel Top-K Algorithms on GPU: A Comprehensive Study and New Methods” SC '23

Pattern #6: Histogram

Example: Parallel Top-k

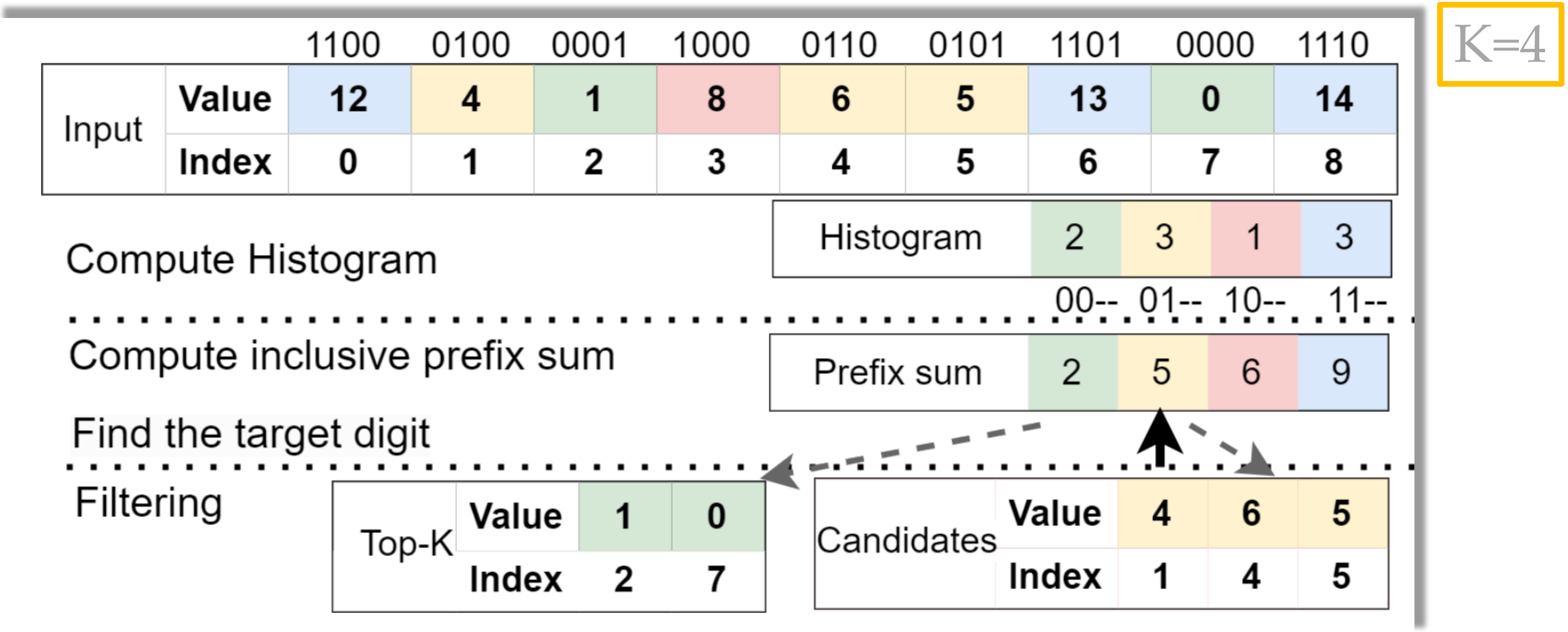


K=4

Zhang, Naruse, Li, and Wang. "Parallel Top-K Algorithms on GPU: A Comprehensive Study and New Methods" SC '23

Pattern #6: Histogram

Example: Parallel Top-k



Zhang, Naruse, Li, and Wang. "Parallel Top-K Algorithms on GPU: A Comprehensive Study and New Methods" SC '23

Pattern #6: Histogram

Example: Bucketing for load balance

- Sparse Matrix Vector Multiplication (SpMV)

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 3 & 0 & 1 & \cdots & 0 \\ 0 & 2 & 0 & \cdots & 0 \\ 0 & 0 & 4 & \cdots & 0 \\ & \vdots & & & \\ 0 & 2 & 6 & \cdots & 8 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

values = [[3,1], [2], [4], ..., [2,6,8]]

cols = [[0,2], [1], [2], ...,]

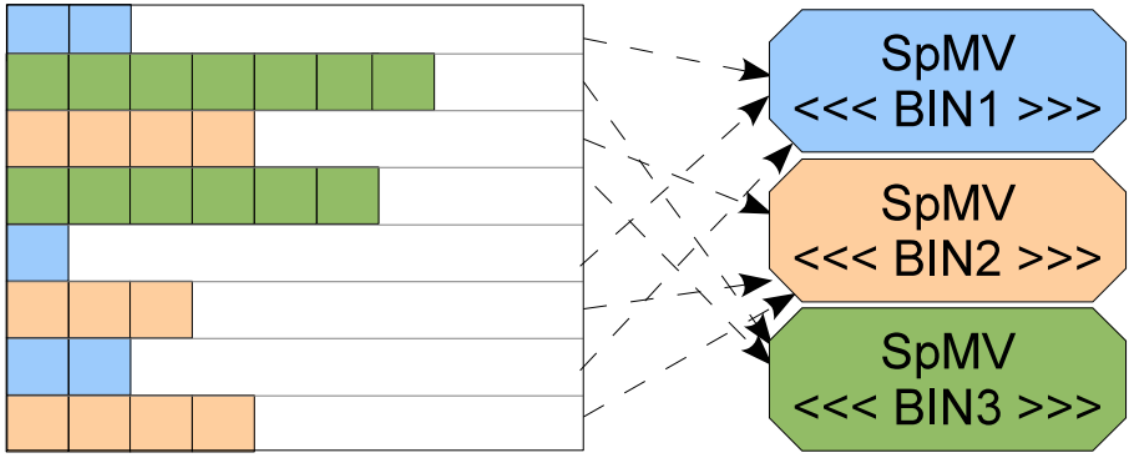
row_starts = [0, 2, 3, 4, ...]

Ashari, Sedaghati, Eisenlohr, Parthasarath and Sadayappan, "Fast Sparse Matrix-Vector Multiplication on GPUs for Graph Applications," SC '14

Pattern #6: Histogram

Example: Bucketing for load balance

- Sparse Matrix Vector Multiplication (SpMV)



$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 3 & 0 & 1 & \cdots & 0 \\ 0 & 2 & 0 & \cdots & 0 \\ 0 & 0 & 4 & \cdots & 0 \\ & \vdots & & & \\ 0 & 2 & 6 & \cdots & 8 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

values = [[3,1], [2], [4], ..., [2,6,8]]

cols = [[0,2], [1], [2], ...,]

row_starts = [0, 2, 3, 4, ...]

Ashari, Sedaghati, Eisenlohr, Parthasarath and Sadayappan, "Fast Sparse Matrix-Vector Multiplication on GPUs for Graph Applications," SC '14

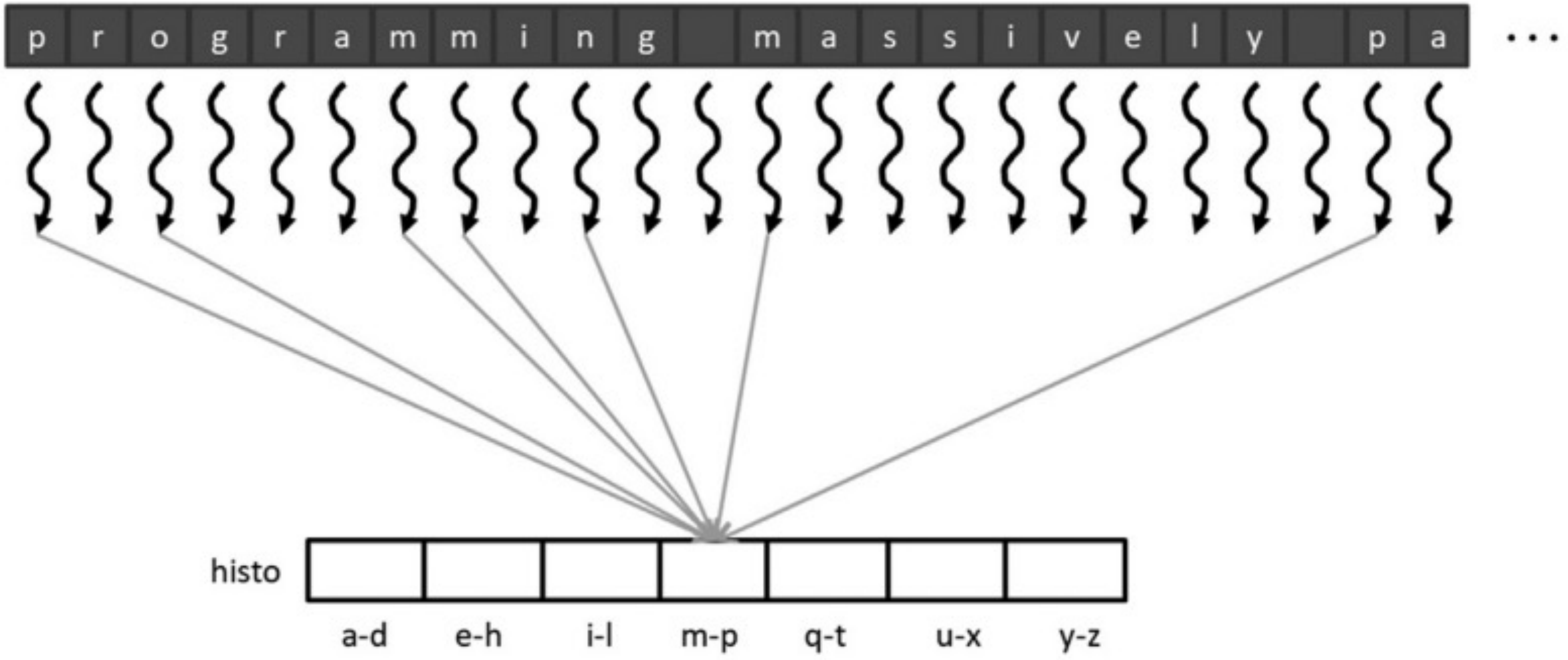
Pattern #6: Histogram

High-level algorithm sketch:

1. Partition the input into sections
2. Each compute unit iterates through its section
3. For each element in the section, the compute unit increments the appropriate bin counter

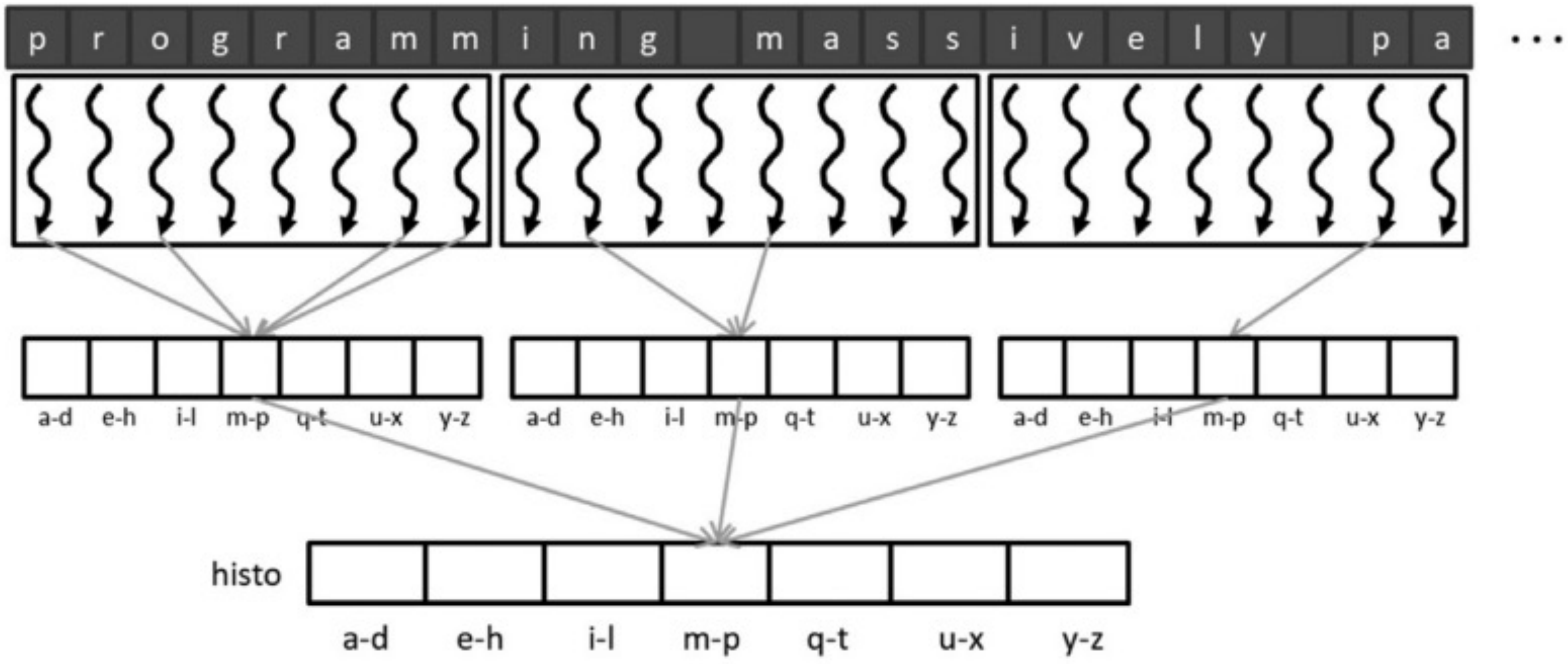
Pattern #6: Histogram

Using atomics



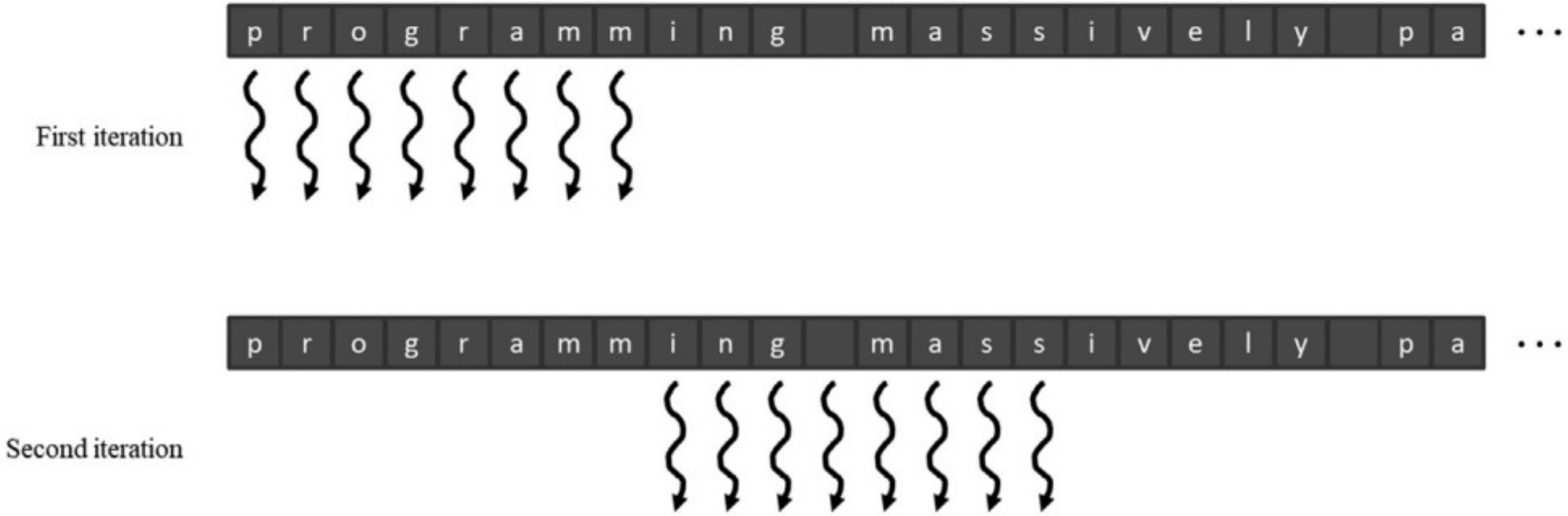
Pattern #6: Histogram

Privatization



Pattern #6: Histogram

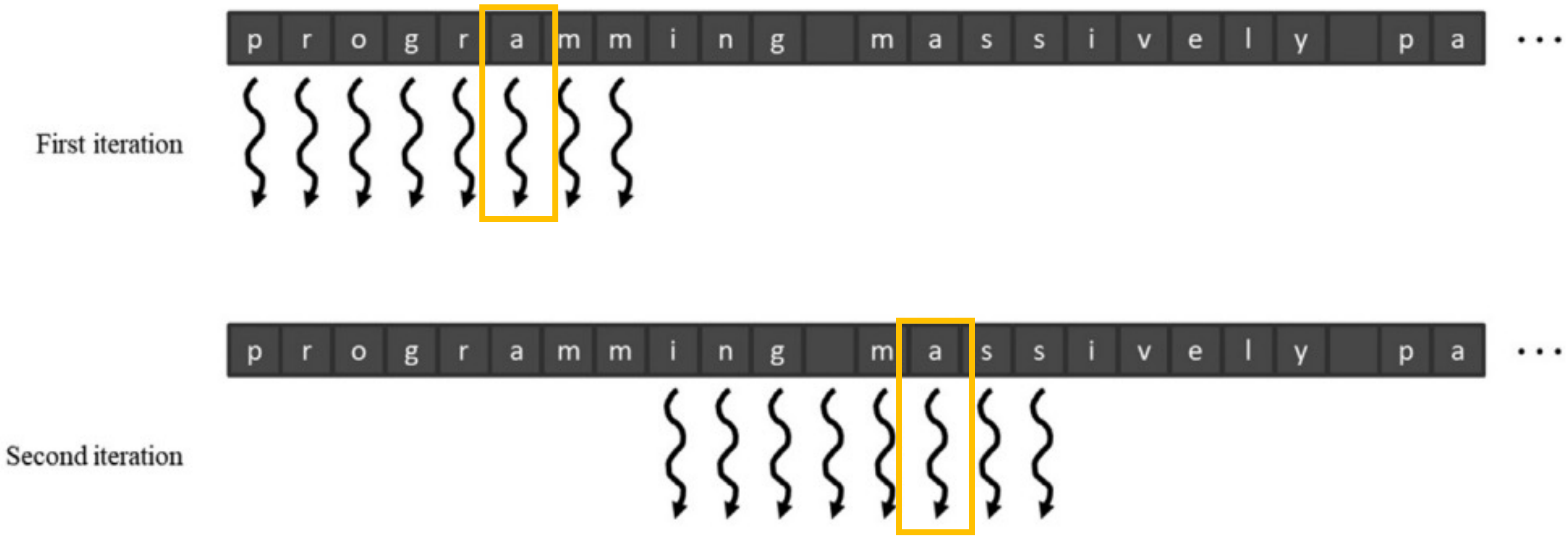
Coarsening



Pattern #6: Histogram

Aggregation

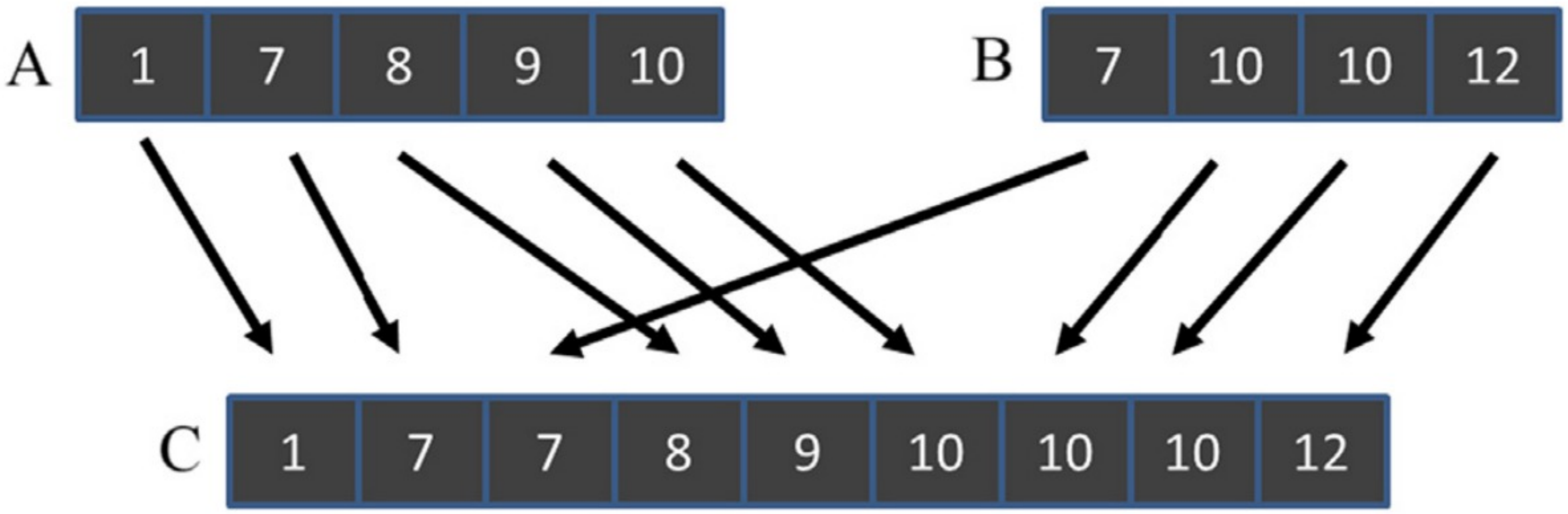
Per thread private accumulator



Pattern #7: Merge

Given two sorted sequences A and B

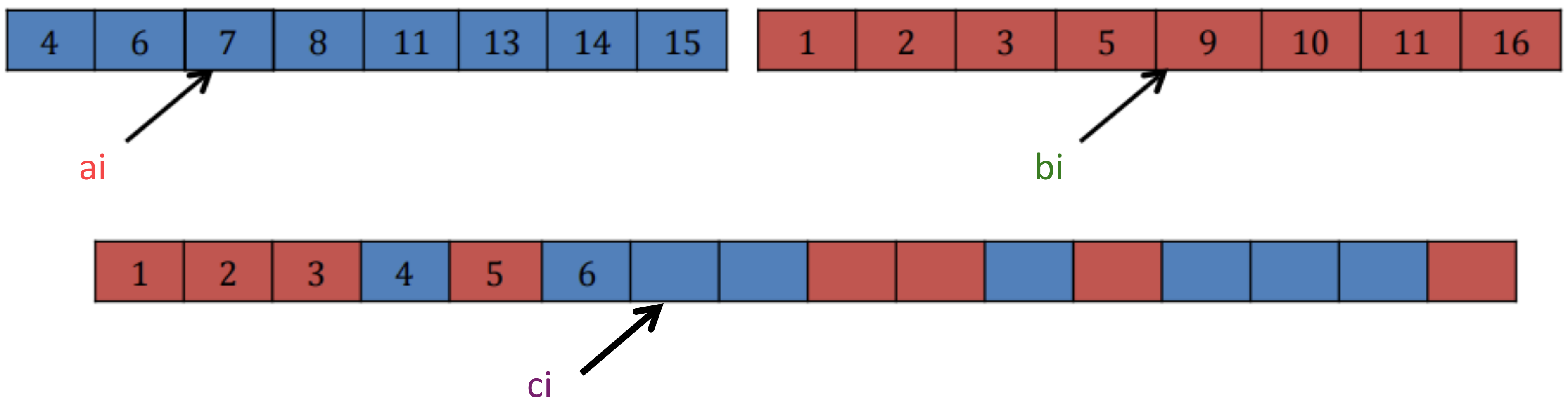
- Generate another sorted sequence C that combines A and B



Pattern #7: Merge

Serial implementation is $O(n)$

```
ai = 0
bi = 0
ci = 0
if (A[ai] < B[bi])
  C[ci++] = A[ai++]
else
  C[ci++] = B[bi++]
```



Pattern #7: Merge

Naïve parallel implementation

1. For each item in A, *binary search* in B
2. For each item in B, *binary search* in A
3. 1. and 2. can be done *concurrently*

Complexity $O(N \log N)$

Pattern #7: Merge

Merge Matrix:

if (A[i]>B[j])
    M[i,j]=1
else:
    M[i,j] =0

A \ B	3	5	12	22	45	64	69	82
17	1	1	1	0	0	0	0	0
29	1	1	1	1	0	0	0	0
35	1	1	1	1	0	0	0	0
73	1	1	1	1	1	1	1	0
86	1	1	1	1	1	1	1	1
90	1	1	1	1	1	1	1	1
95	1	1	1	1	1	1	1	1
99	1	1	1	1	1	1	1	1

Pattern #7: Merge

Merge Path:

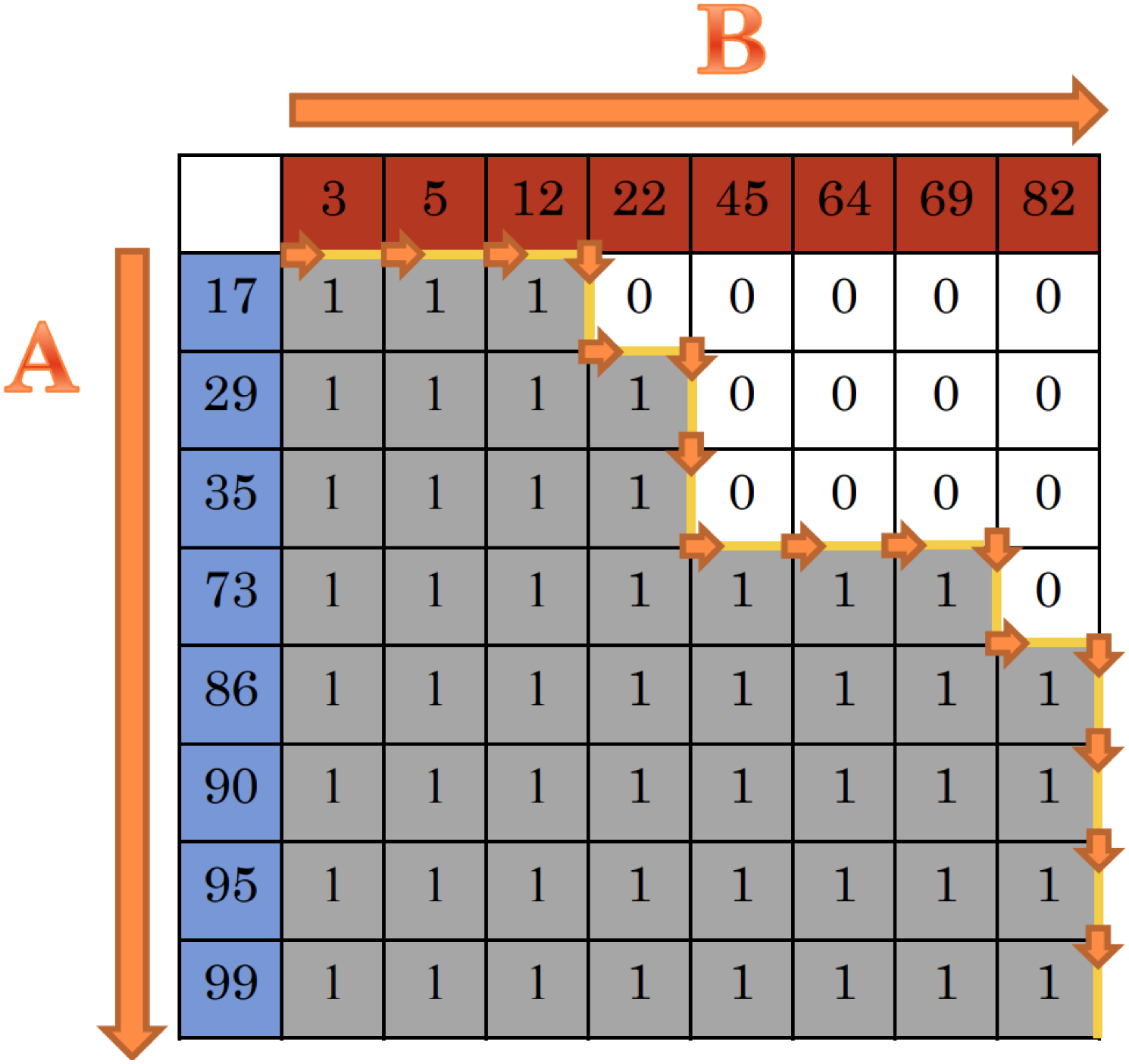
- Start at top-left
- Stop at bottom-right

if (A[i]>B[j])

    move right

else:

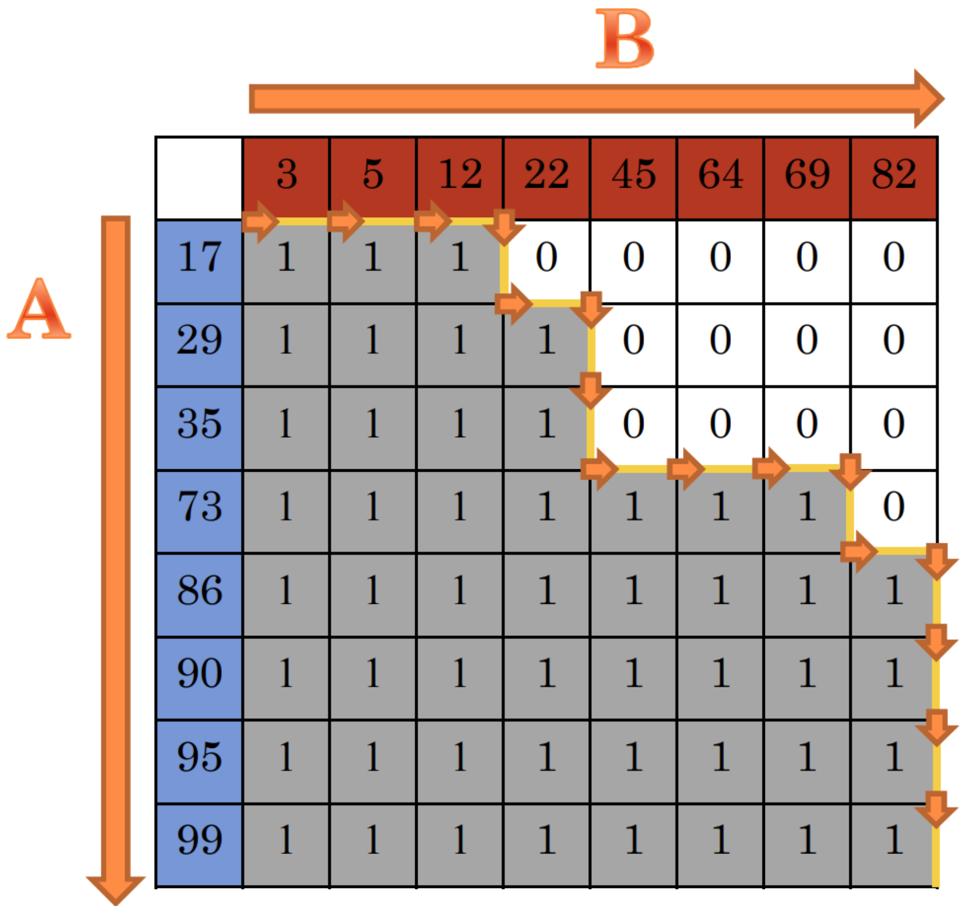
    move down



Pattern #7: Merge

Merge Path Properties:

- The merge path is the same as the output sequence.

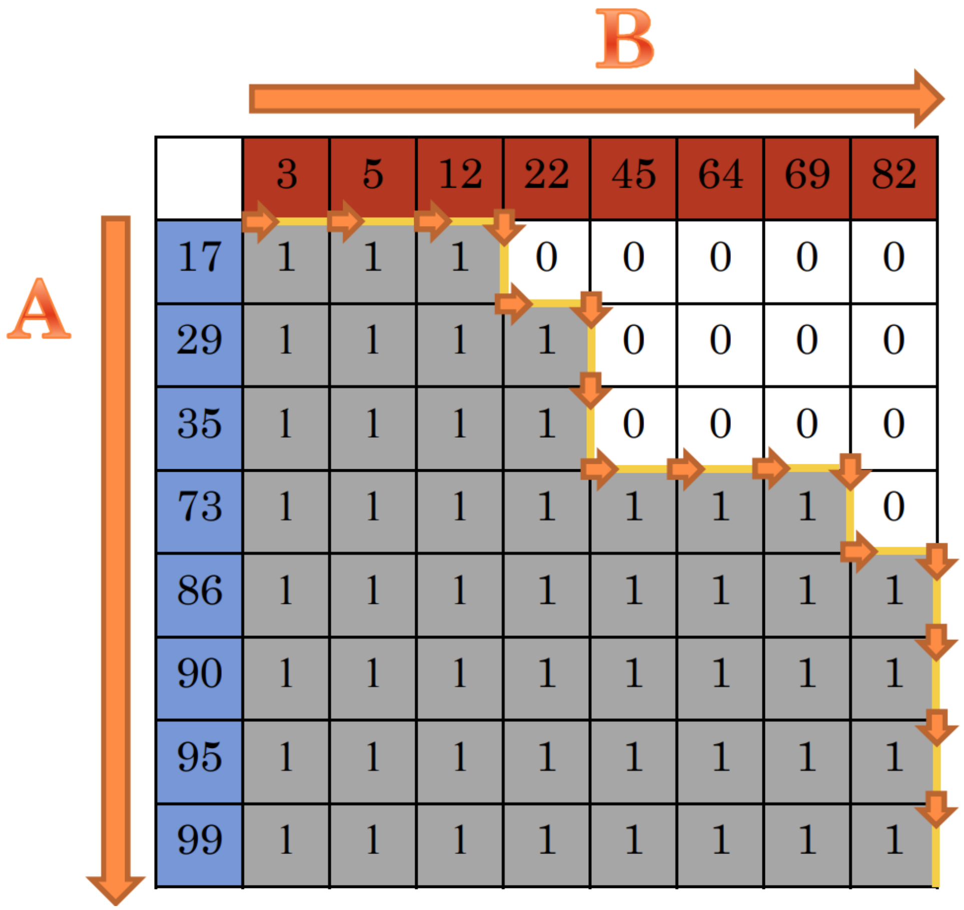


Pattern #7: Merge

Merge Path Properties:

- The merge path is the same as the output sequence.

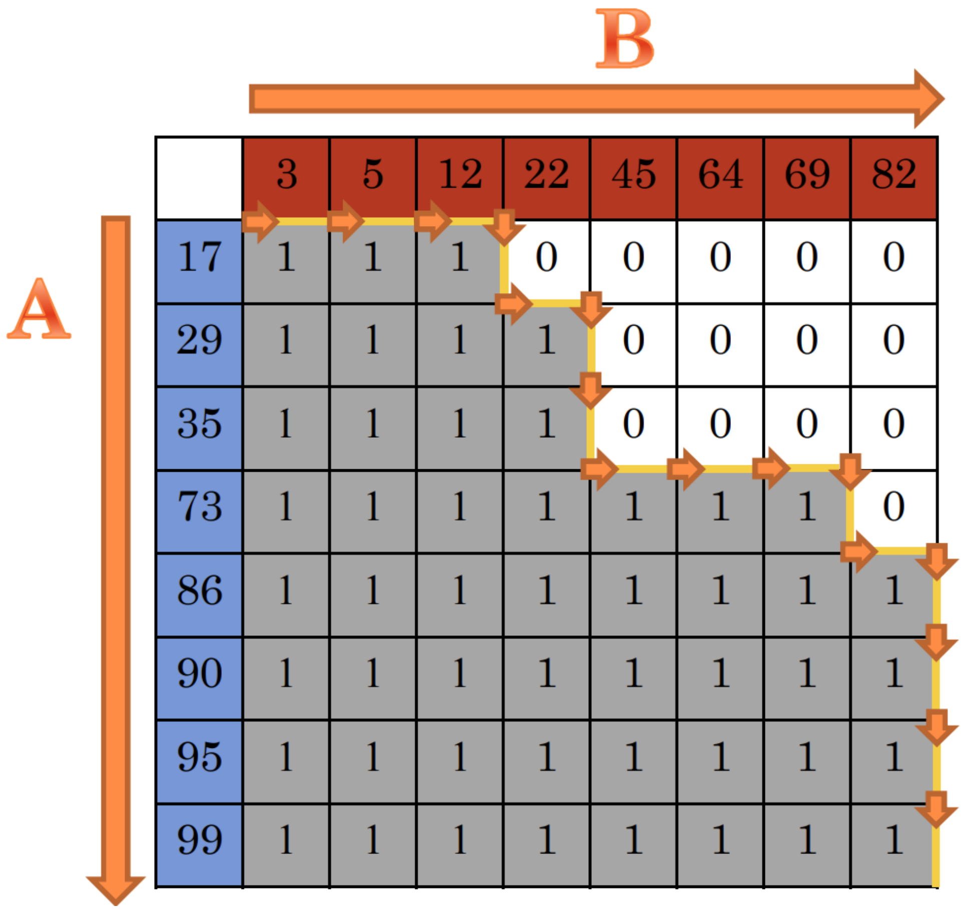
- Each segment of the merge bath is composed of a contiguous sequence of elements of A and B



Pattern #7: Merge

Merge Path Properties:

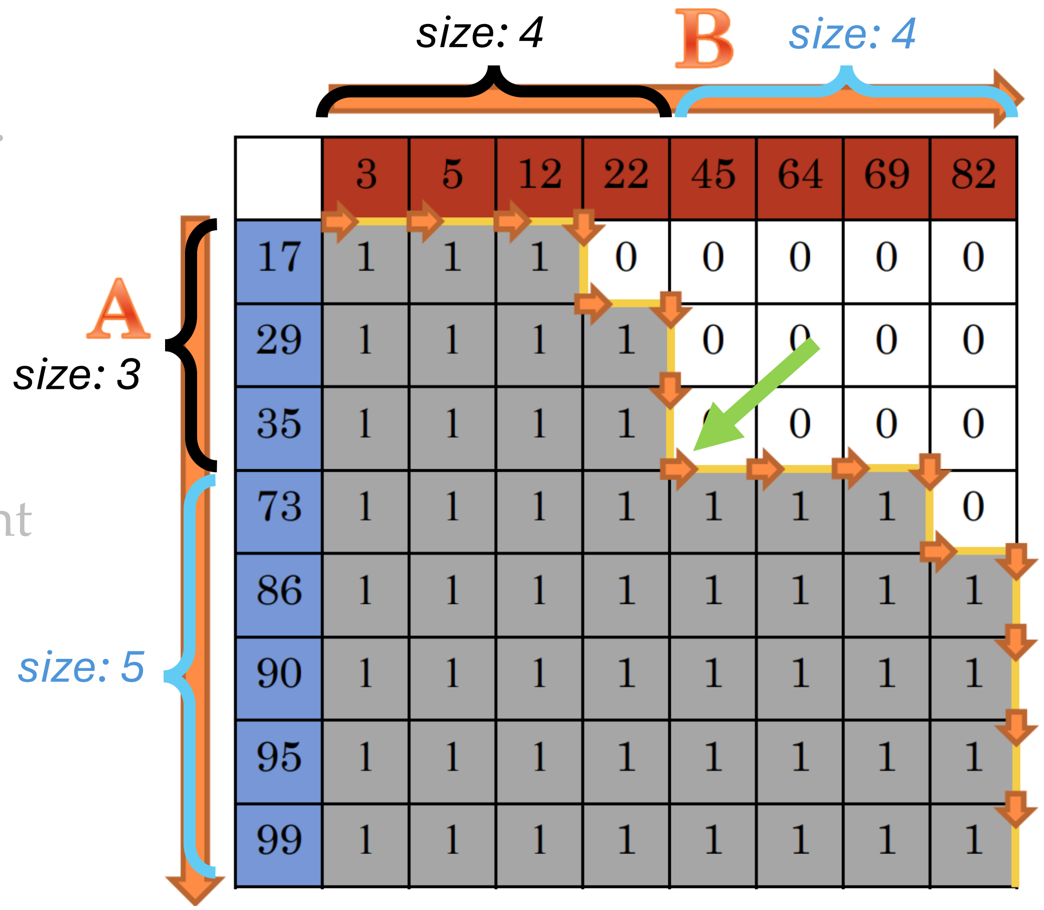
- The merge path is the same as the output sequence.
- Each segment of the merge bath is composed of a contiguous sequence of elements of A and B
- It is possible to partition the merge path into disjoint sets



Pattern #7: Merge

Merge Path Properties:

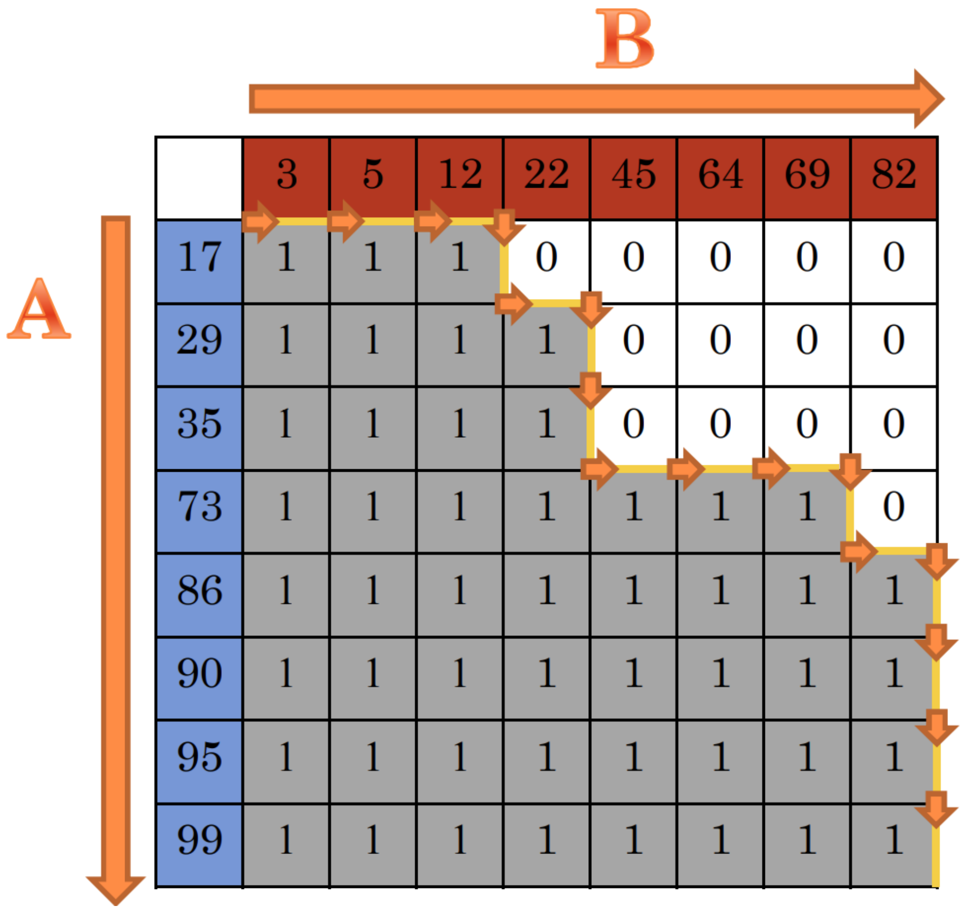
- The merge path is the same as the output sequence.
- Each segment of the merge bath is composed of a contiguous sequence of elements of A and B
- It is possible to partition the merge path into disjoint sets
- For each segment, we know where to write it in parallel.



Pattern #7: Merge

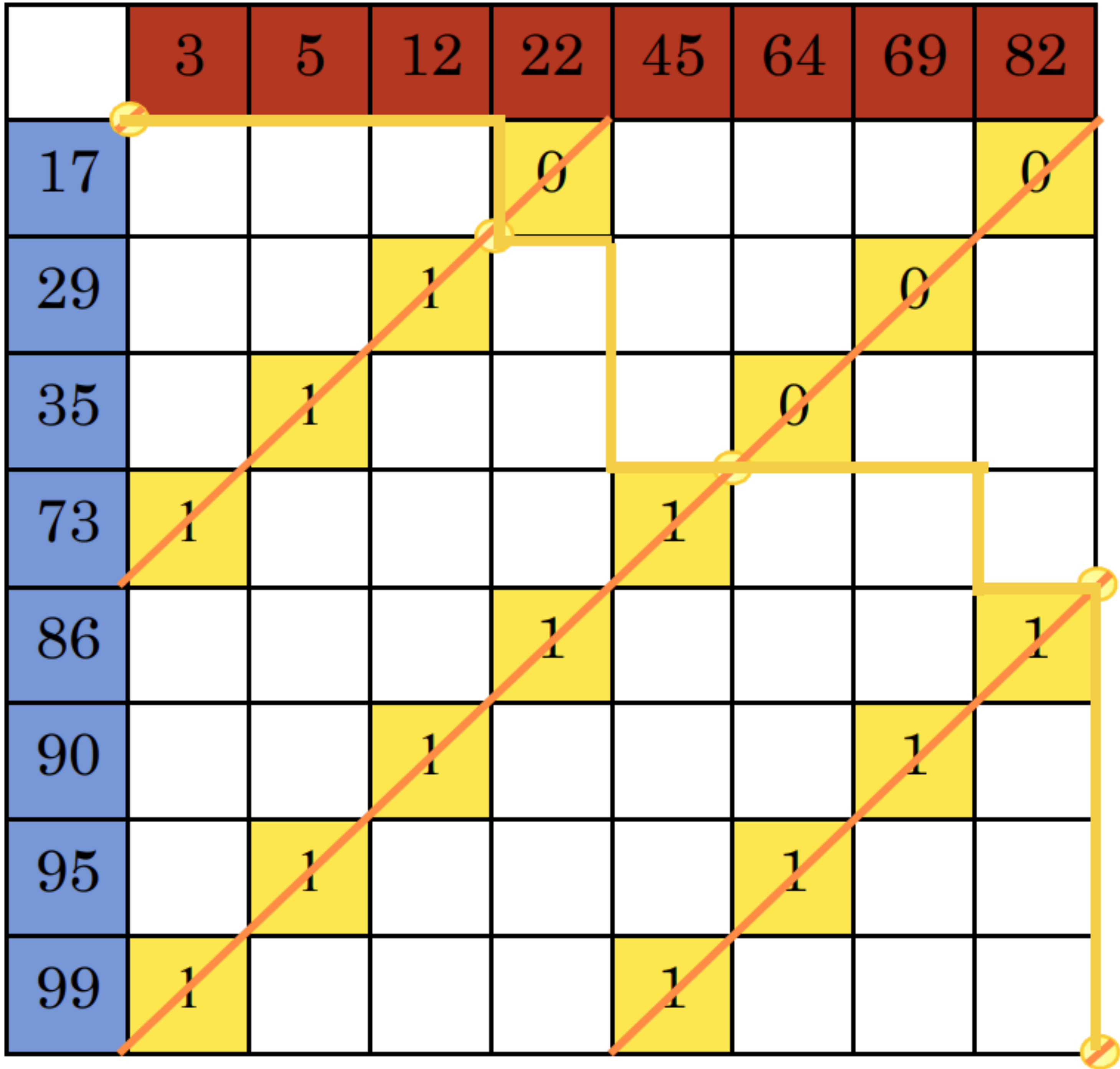
Merge Path Properties:

- The merge path is the same as the output sequence.
- Each segment of the merge bath is composed of a contiguous sequence of elements of A and B
- It is possible to partition the merge path into disjoint sets
- For each segment, we know where to write it in parallel.
- Partitioning the merge path will lead to load balance.



Pattern #7: Merge

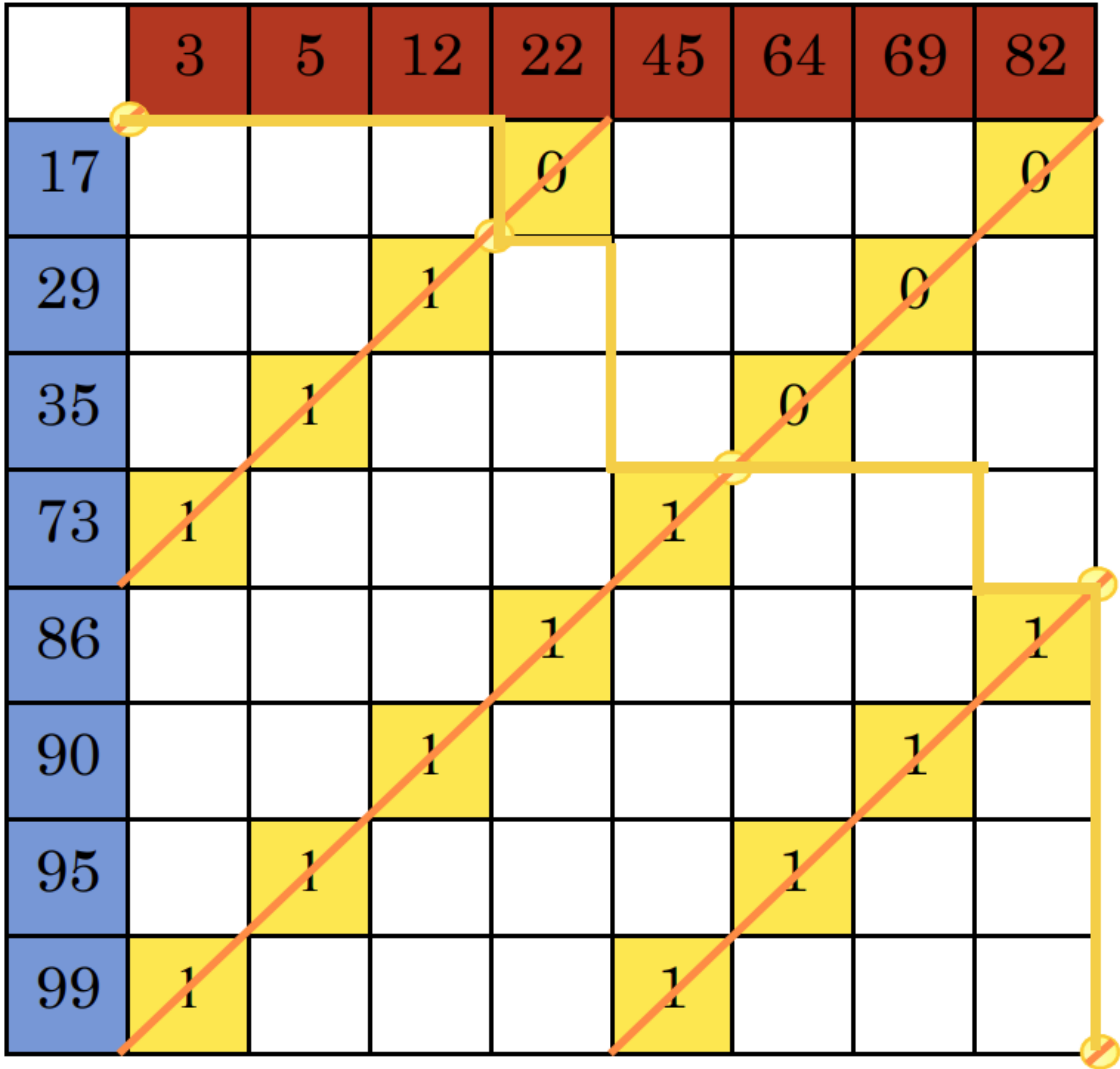
Cross diagonals:



Pattern #7: Merge

Cross diagonals:

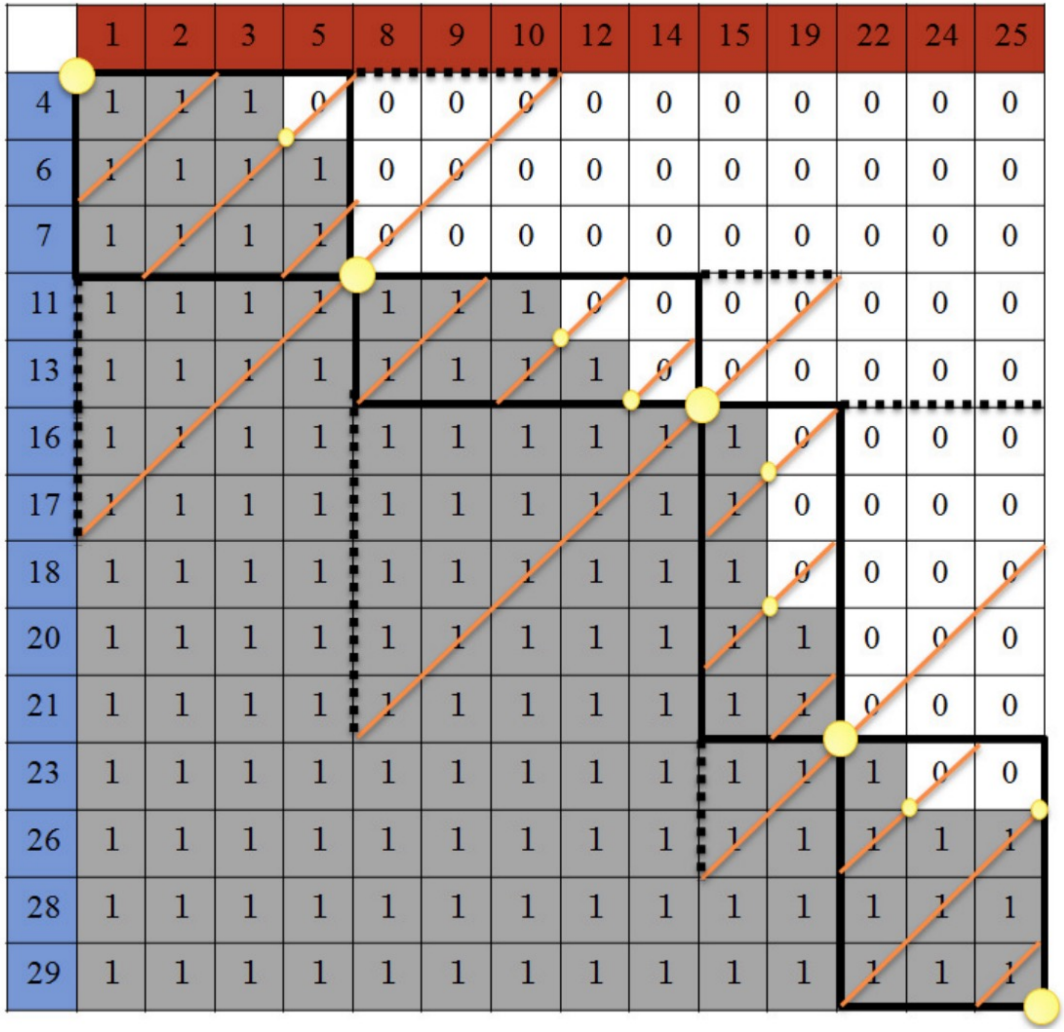
- Entries along the cross diagonals are monotonically non-increasing



Pattern #7: Merge

Cross diagonals:

- Partitions the merge path equally
- Every processor (i.e., thread/block/grid) does a constrained single binary search
- Can be done hierarchically



Saher Odeh, Oded Green ‡, Zahi Mwassi, Oz Shmueli, Yitzhak Birk
"Merge Path - Parallel Merging Made Simple". IPDPS 2012

Reduction using Tensor Cores

Reduction as matrix multiplication

Dakkak, Li, Xiong, Gelado, and Hwu.
“Accelerating reduction and scan using tensor core units”. ICS '19

Reduction using Tensor Cores

Reduction as matrix multiplication

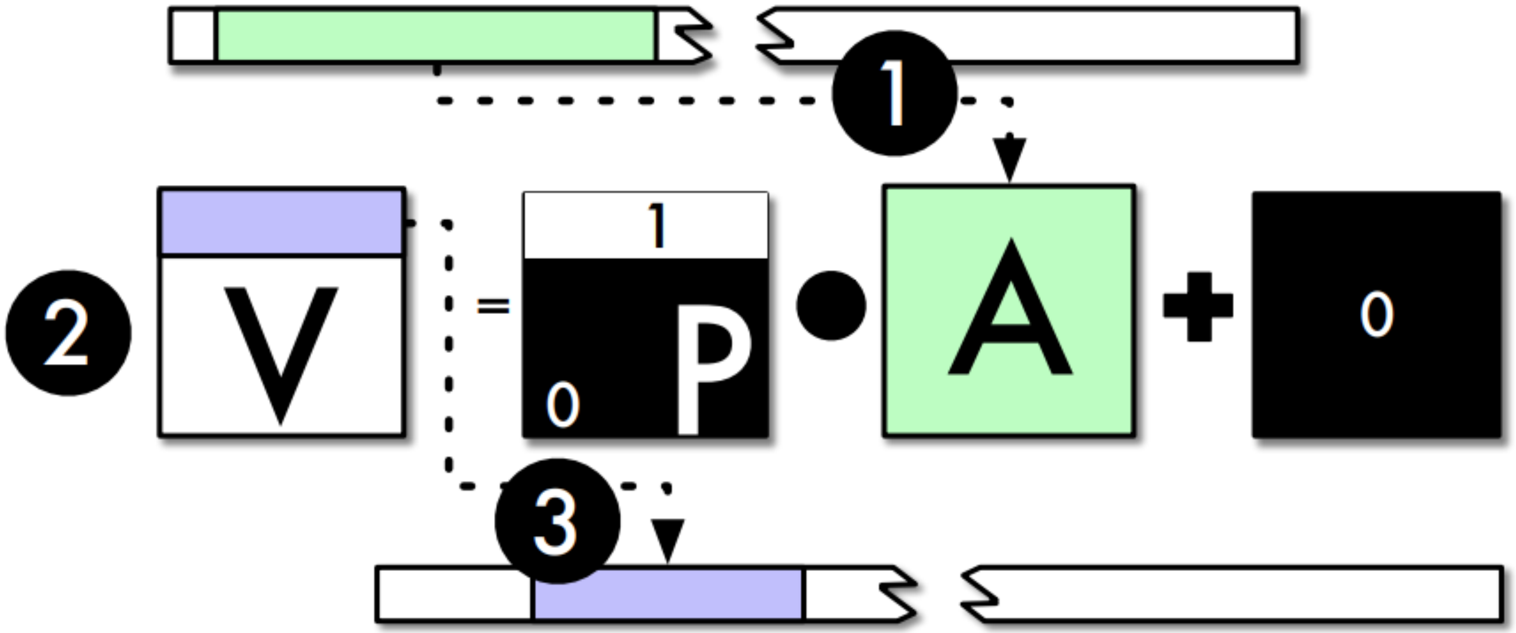
$$Reduction\left[a_1, a_2, \ldots, a_n\right] =$$

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} a_1 & 0 & \cdots & 0 \\ a_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_n & 0 & \cdots & 0 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} a_i & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

Dakkak, Li, Xiong, Gelado, and Hwu.
"Accelerating reduction and scan using tensor core units". ICS '19

Reduction using Tensor Cores

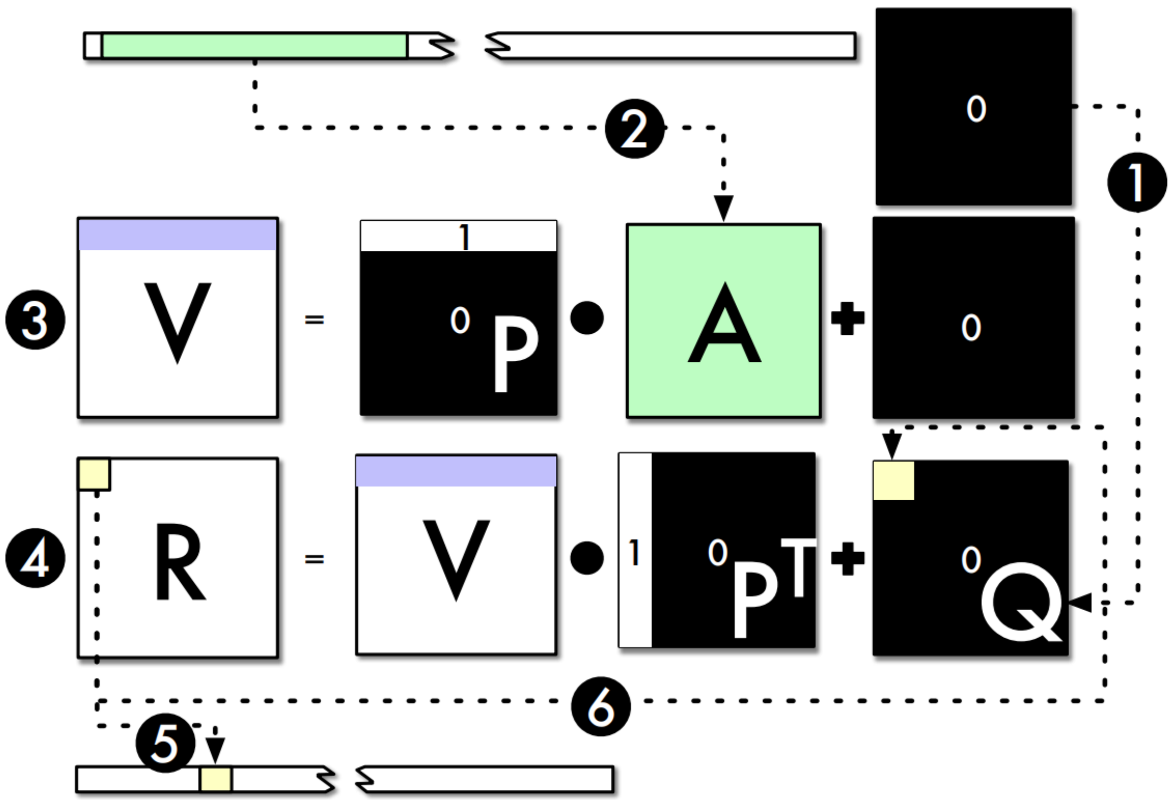
Reduction as fused-multiply add (FMA)



Dakkak, Li, Xiong, Gelado, and Hwu.
"Accelerating reduction and scan using tensor core units". ICS '19

Reduction using Tensor Cores

Reduction as fused-multiply add (FMA)



Dakkak, Li, Xiong, Gelado, and Hwu.
"Accelerating reduction and scan using tensor core units". ICS '19

Data-Parallel Primitives in Thrust

Algorithms

- Copying
  - Gather
  - Scatter
  - thrust::swap_ranges
  - thrust::copy
  - thrust::swap_ranges
  - thrust::copy_n
  - thrust::uninitialized_copy
  - thrust::copy
  - thrust::copy_n
  - thrust::uninitialized_copy_n
  - thrust::uninitialized_copy_n
  - thrust::uninitialized_copy
- Merging
  - thrust::merge_by_key
  - thrust::merge
  - thrust::merge
  - thrust::merge_by_key
  - thrust::merge_by_key
  - thrust::merge
  - thrust::merge
  - thrust::merge_by_key
- Reductions
  - Comparisons
  - Counting
  - Extrema
  - Logical
  - Predicates
  - Transformed Reductions
  - thrust::reduce_by_key
  - thrust::reduce_by_key
  - thrust::reduce
  - thrust::reduce_by_key
  - thrust::reduce_by_key
  - thrust::reduce
  - thrust::reduce
  - thrust::reduce
  - thrust::reduce_by_key
  - thrust::reduce_by_key
  - thrust::reduce
  - thrust::reduce
- Reordering
  - Partitioning
  - Shuffling
  - Stream Compactation
- Prefix sums
  - Segmented Prefix Sums
  - Transformed Prefix Sums
  - thrust::inclusive_scan
  - thrust::exclusive_scan
  - thrust::exclusive_scan
  - thrust::inclusive_scan
  - thrust::inclusive_scan
  - thrust::inclusive_scan
  - thrust::exclusive_scan
  - thrust::exclusive_scan
  - thrust::exclusive_scan
  - thrust::inclusive_scan
  - thrust::inclusive_scan
  - thrust::exclusive_scan
- Transformations
  - Filling
  - Modifying
  - Replacing
  - thrust::sequence
  - thrust::adjacent_difference
  - thrust::tabulate
  - thrust::adjacent_difference
  - thrust::sequence
  - thrust::transform_if
  - thrust::transform_if
  - thrust::sequence
  - thrust::transform
  - thrust::transform
  - thrust::generate_n
  - thrust::transform_if
  - thrust::sequence
  - thrust::transform_if
  - thrust::generate
  - thrust::transform_if
  - thrust::transform_if
  - thrust::tabulate
  - thrust::transform
  - thrust::transform
  - thrust::adjacent_difference
  - thrust::generate
  - thrust::sequence
  - thrust::adjacent_difference
  - thrust::generate_n
  - thrust::sequence
- Sorting
  - thrust::sort
  - thrust::sort_by_key
  - thrust::stable_sort
  - thrust::stable_sort_by_key
  - thrust::sort_by_key
  - thrust::stable_sort
  - thrust::stable_sort_by_key
  - thrust::sort_by_key
  - thrust::stable_sort
  - thrust::stable_sort
  - thrust::sort
  - thrust::sort
  - thrust::sort_by_key
  - thrust::stable_sort_by_key
  - thrust::sort
  - thrust::stable_sort_by_key

https://nvidia.github.io/cccl/thrust/api_docs/algorithms.html

Data-Parallel Primitives in CUB

- **Parallel primitives**
  - Warp-wide "collective" primitives
    - Cooperative warp-wide prefix scan, reduction, etc.
    - Safely specialized for each underlying CUDA architecture
  - Block-wide "collective" primitives
    - Cooperative I/O, sort, scan, reduction, histogram, etc.
    - Compatible with arbitrary thread block sizes and types
  - Device-wide primitives
    - Parallel sort, prefix scan, reduction, histogram, etc.
    - Compatible with CUDA dynamic parallelism

https://nvidia.github.io/cccl/cub/

Credits:

This lecture is primarily derived from:

- John Owens's course on Modern Parallel Computing (EEC 289Q, UC Davis, Winter 2018)
- Kayvon Fatahalian's course on Parallel Computing (CS149, Stanford, Fall 2023)
- Programming Massively Parallel Processors - A Hands-on Approach book, 4th edition by Wen-mei W. Hwu, David B. Kirk, and Izzat El Hajj, 2023

Questions?!