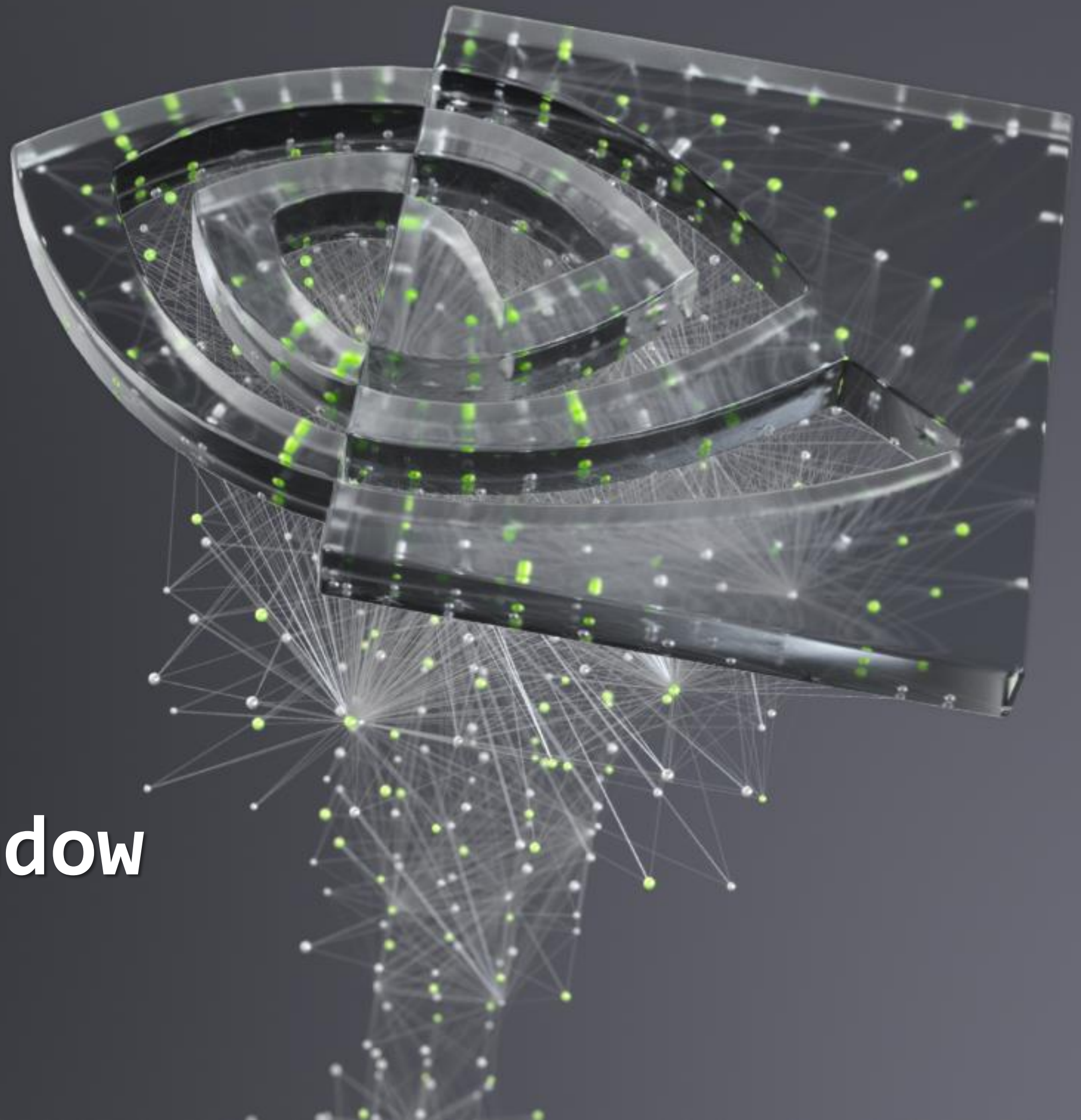




# cudf::rolling\_window

Conor Hoekstra

January 6, 2021





**HAPPY NEW YEARS!**

**RAPIDS**



AGENDA

HISTORY

---

`cudf::rolling_window`

---

DISCUSSION

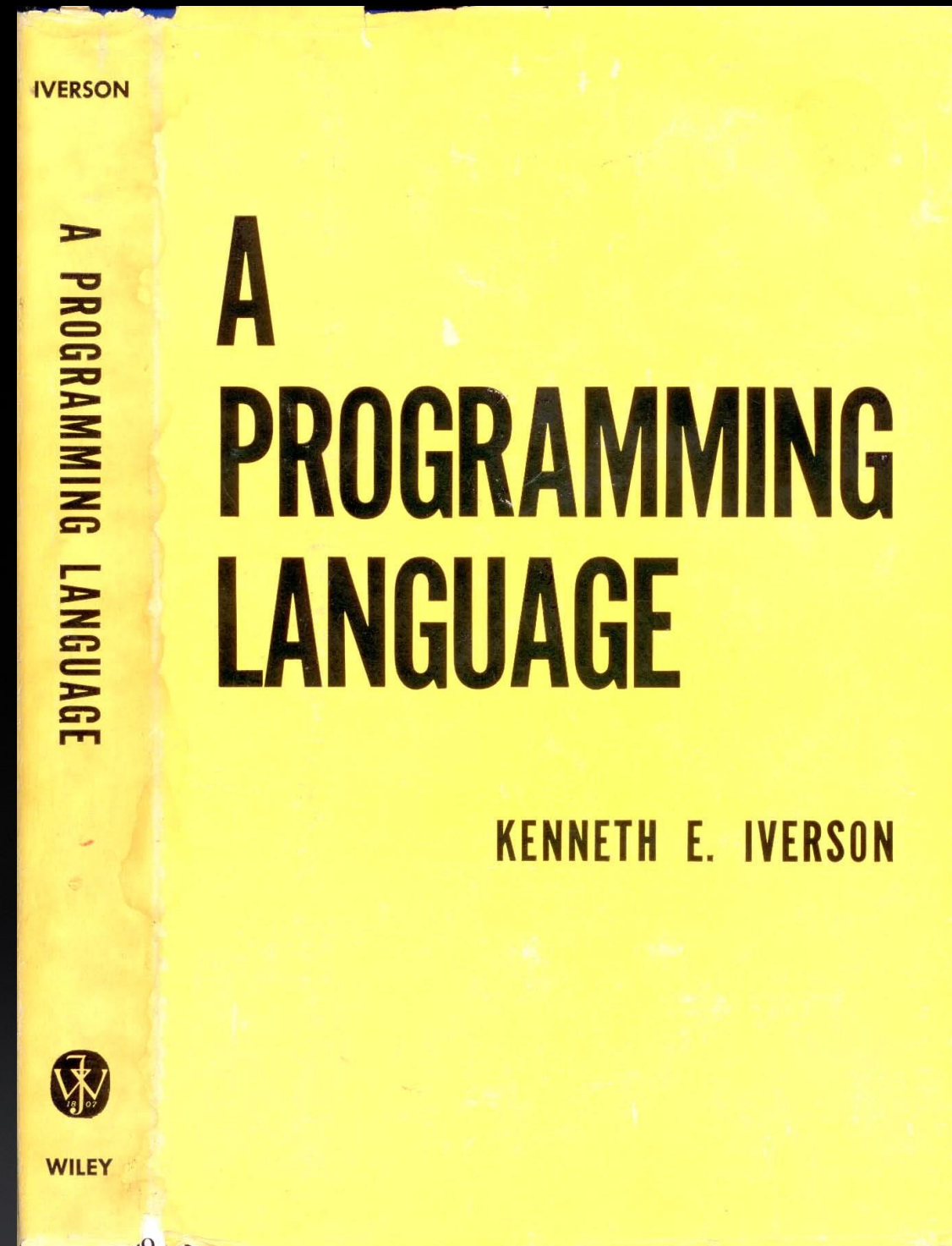
---





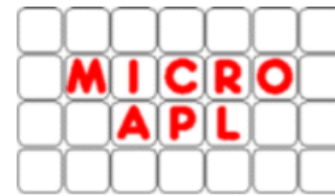
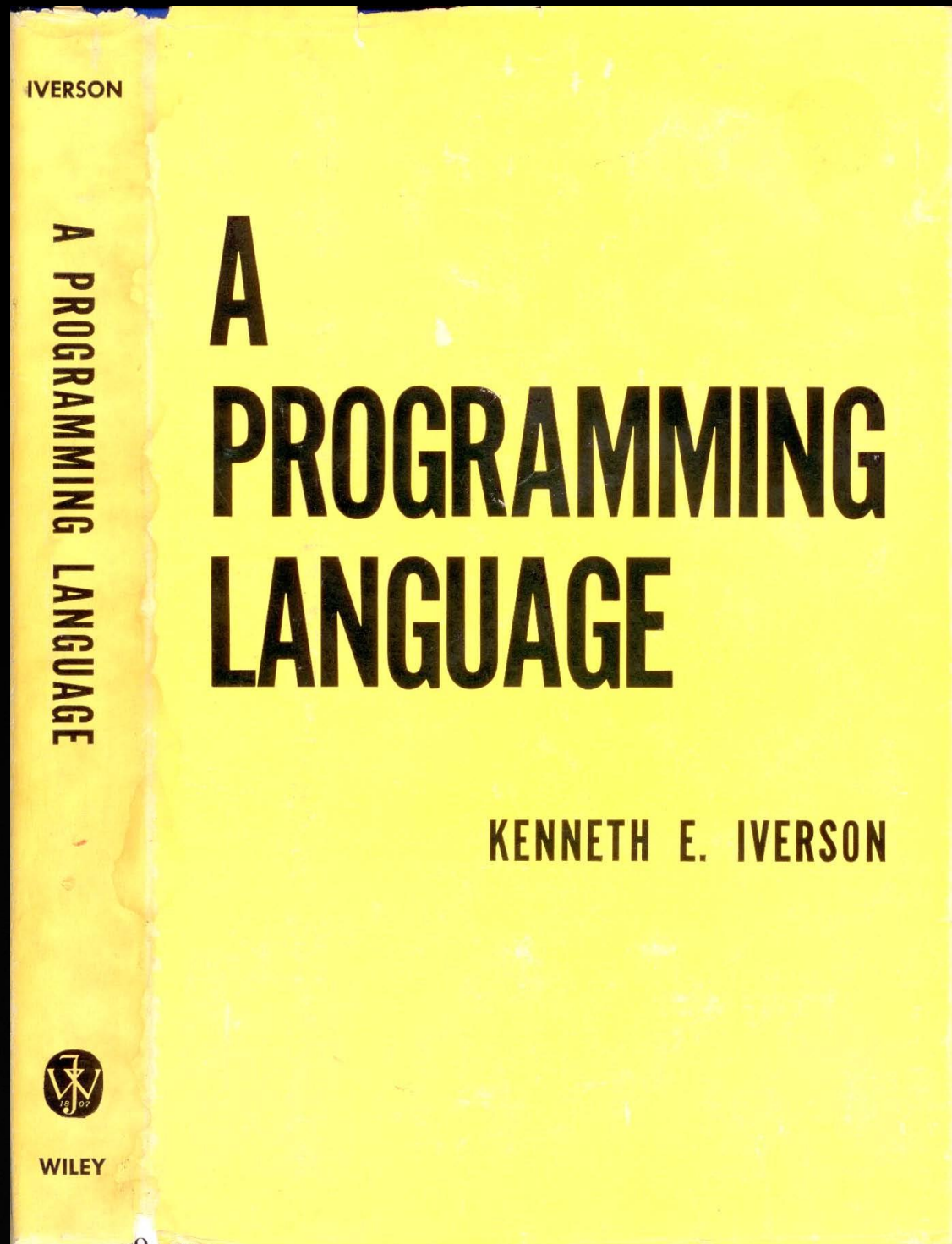
HISTORY

# HISTORY





# HISTORY

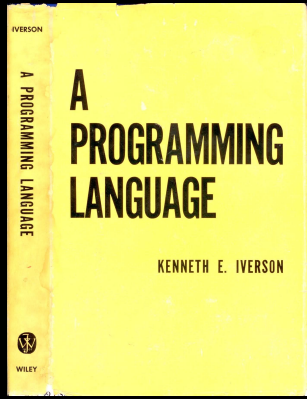


## / Reduction

When used with a function operand the `/` operator is known as Reduction (see the entry for Compression for the other functions derived from `/`). The context in which the `/` is used should make clear the operation being carried out. `/` can be applied to any dyadic function, including user defined functions. When used with a scalar or one-element vector integer left argument, the `/` operator is used to perform 'N-wise reduction'.

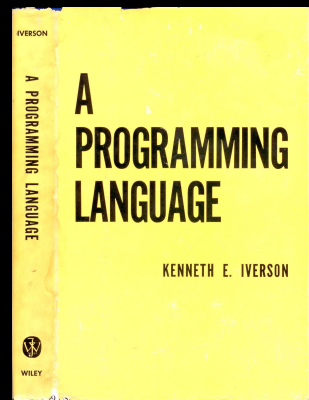
The left operand of `/` is inserted between all elements of the array right argument. In the absence of an axis specification, the operand is inserted between items along the last axis (see also the entry for `[]`, the [Axis](#) operator).

	<code>+/ 2 4 6</code>	(This is the same as <code>2+4+6</code> )
12	<code>SALES←25 5.7 8 50 101 74 19</code> <code>+/SALES</code>	
282.7		(The sum of the numbers in SALES)
	<code>[]/82 66 93 13</code>	(The same as <code>82 [] 66 [] 93 [] 13</code> .
93		The result of <code>93[]13</code> is compared with 66; the result of this comparison is compared with 82; the result of the last comparison is the largest)
	<code>∨/0 1 1 0 0</code>	(The same as <code>0 ∨ 1 ∨ 1 ∨ 0 ∨ 0</code> )
1		(Used to test if there are any 1s)



# HISTORY

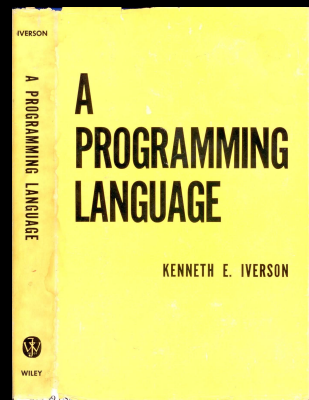
```
12      +/ 2 4 6      (This is the same as 2+4+6)
```



# HISTORY

	$\lceil / 82 \ 66 \ 93 \ 13$	(The same as $82 \lceil 66 \lceil 93 \lceil 13$ .
93		The result of $93 \lceil 13$ is compared with 66; the result of this comparison is compared with 82; the result of the last comparison is the largest)
	$\vee / 0 \ 1 \ 1 \ 0 \ 0$	(The same as $0 \vee 1 \vee 1 \vee 0 \vee 0$ )
1		(Used to test if there are any 1s)





# HISTORY

## N-Wise Reduction

The definition of N-wise Reduction is very similar to the definition of Reduction. The left argument, an integer scalar or length one vector, is used to specify the length of successive subsets of the right argument on which the Reduction operation is performed. If the left argument is negative, each subset is reversed before the reduction operation is carried out.

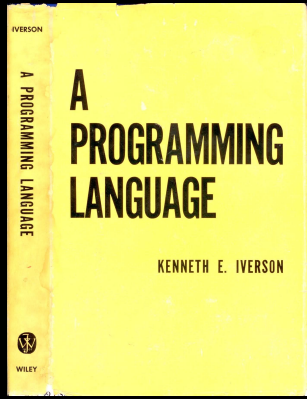
For a left argument of absolute value  $n$  and the selected axis of the right argument of length  $m$ , the number of subsets to which the reduction can be applied are:

$$1 + m - n$$

and thus the limiting case is where the sample size is 1 greater than the length of the selected axis, giving a empty result.

```
2+/\10
3 5 7 9 11 13 15 17 19
5+/\10
15 20 25 30 35 40
```

(Add up the numbers 2 at a time, starting  
at the beginning of the vector)  
(5 at a time)



# HISTORY

$\iota 5$

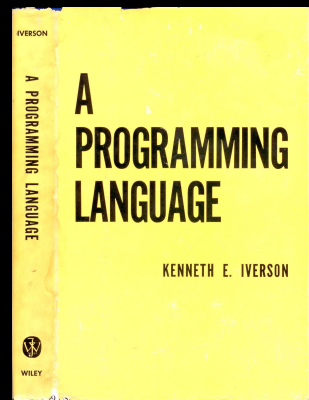
1 2 3 4 5

$2 + / \iota 5$

3 5 7 9

$2, / \iota 5$

1	2	2	3	3	4	4	5
---	---	---	---	---	---	---	---



# HISTORY

A adjacent\_difference

2- / 5

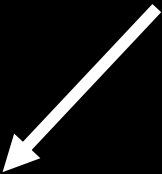
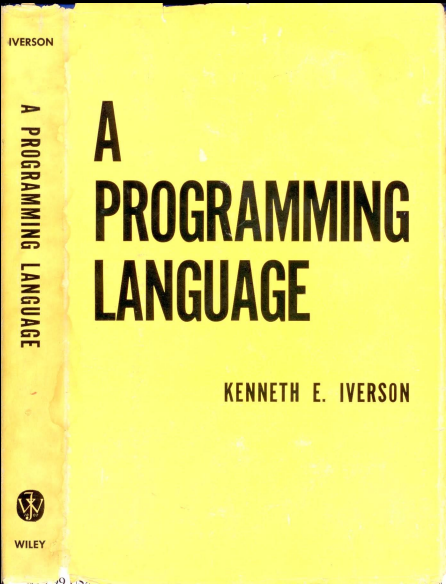
-1 -1 -1 -1

2- ~ / 5

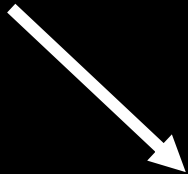
1 1 1 1



# HISTORY



J



K



q)

# HISTORY

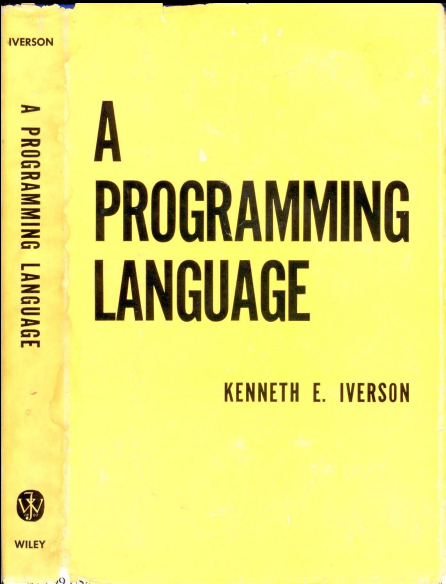
$m \leftarrow 3 \quad 3\rho \wr 9$

1	2	3
4	5	6
7	8	9

$$\{\subset \omega\} \boxtimes 2 \quad 2 \vdash m$$

1	2	2	3
4	5	5	6
4	5	5	6
7	8	8	9

# HISTORY



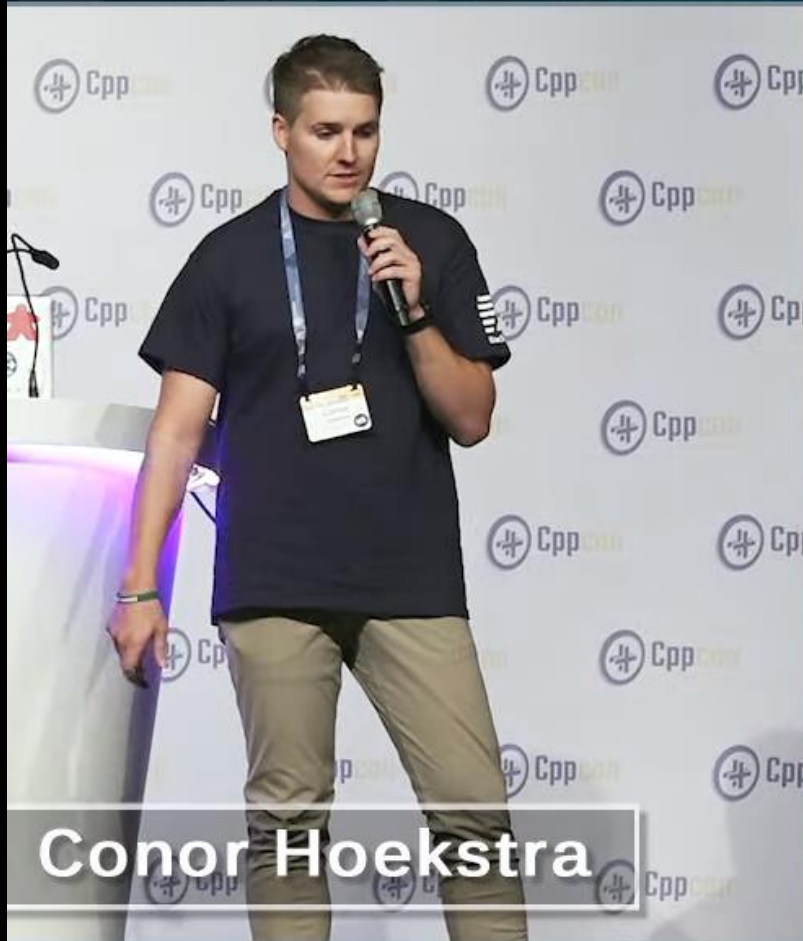
J

K

q)







Conor Hoekstra

23 Ranges:  
slide & stride

Video Sponsorship Provided By:

ansatz



windowed



sliding



slide



divvy



partition













code\_report



Conor Hoekstra

## 23 Ranges: slide & stride

	Language	step, size = ?	step = size	step = 1	size = 1
	C++	-	chunk	sliding	stride
	Haskell	divvy	chunksOf	-	-
	Elixir	chunk_every/4	chunk_every/2	-	take_every
	Ruby	-	each_slice	each_cons	each_with_index
	D	slide	chunks	slide*	stride
	Rust	-	chunks	windows	-
	F#	-	chunkBySize	windowed	-
	Clojure	partition	partition	-	take-nth
	Kotlin	windowed	chunked	windowed*	-
	Scala	sliding	grouped	sliding**	-

Video Sponsorship Provided By:





# cudf::rolling\_window

```
std::unique_ptr<column> rolling_window(  
    column_view const& input,  
    size_type preceding_window,  
    size_type following_window,  
    size_type min_periods,  
    std::unique_ptr<aggregation> const& agg,  
    rmm::mr::device_memory_resource* mr);
```



The following is a support matrix for aggregations operations vs aggregations algorithms:

Agg Ops	reduce	scan	rolling	group_by	grouped_rolling
SUM	✓	✓	✓	✓	
PRODUCT	✓	✓			
MIN	✓	✓	✓	✓	
MAX	✓	✓	✓	✓	
COUNT_VALID	✗	✗	✓	✓	
COUNT_ALL	✗	✗	✓	✓	
ANY	✓	✗			
ALL	✓	✗			
SUM_OF_SQUARES	✓	✗			
MEAN	✓	✗	✓	✓	
VARIANCE	✓	✗		✓	
STD	✓	✗		✓	

# cudf::rolling\_window

```
std::unique_ptr<column> rolling_window(  
    column_view const& input,  
    size_type preceding_window,  
    size_type following_window,  
    size_type min_periods,  
    std::unique_ptr<aggregation> const& agg,  
    rmm::mr::device_memory_resource* mr);
```

```
std::unique_ptr<column> rolling_window(
    column_view const& input,
    size_type preceding_window,
    size_type following_window,
    size_type min_periods,
    std::unique_ptr<aggregation> const& agg,
    rmm::mr::device_memory_resource* mr);
```

# cudf::rolling\_window

```
// auto col = { 1, 2, 3, 2, 1 };
```

```
auto res = cudf::rolling_window(col, 2, 1, 1, make_min_aggregation());
auto res = cudf::rolling_window(col, 2, 1, 1, make_max_aggregation());
auto res = cudf::rolling_window(col, 2, 1, 1, make_sum_aggregation());
```

```
// res = { 1, 1, 2, 1, 1 }
// res = { 2, 3, 3, 3, 2 }
// res = { 3, 6, 7, 6, 3 }
```



```
std::unique_ptr<column> rolling_window(
    column_view const& input,
    size_type preceding_window,
    size_type following_window,
    size_type min_periods,
    std::unique_ptr<aggregation> const& agg,
    rmm::mr::device_memory_resource* mr);
```

# cudf::rolling\_window

```
// auto col = { 1, 2, 3, 2, 1 };
```

```
auto res = cudf::rolling_window(col, 2, 1, 1, make_min_aggregation());
auto res = cudf::rolling_window(col, 2, 1, 1, make_max_aggregation());
auto res = cudf::rolling_window(col, 2, 1, 1, make_sum_aggregation());
```

```
// res = { 1, 1, 2, 1, 1 }
// res = { 2, 3, 3, 3, 2 }
// res = { 3, 6, 7, 6, 3 }
```

# cudf::rolling\_window

```
std::unique_ptr<column> rolling_window(  
    column_view const& input,  
    size_type preceding_window,  
    size_type following_window,  
    size_type min_periods,  
    std::unique_ptr<aggregation> const& agg,  
    rmm::mr::device_memory_resource* mr);
```

# cudf::rolling\_window

```
<ROW or RANGE clause> ::=  
{ ROWS | RANGE } <window frame extent>  
  
<window frame extent> ::=  
{ <window frame preceding>  
  | <window frame between>  
}  
<window frame between> ::=  
  BETWEEN <window frame bound> AND <window frame bound>  
  
<window frame bound> ::=  
{ <window frame preceding>  
  | <window frame following>  
}  
  
<window frame preceding> ::=  
{  
  UNBOUNDED PRECEDING  
  | <unsigned_value_specification> PRECEDING  
  | CURRENT ROW  
}
```



# cudf::rolling\_window

SQL

 Copy

```
SELECT BusinessEntityID, TerritoryID
, CONVERT(VARCHAR(20), SalesYTD, 1) AS SalesYTD
, DATEPART(yy, ModifiedDate) AS SalesYear
, CONVERT(VARCHAR(20), SUM(SalesYTD) OVER (PARTITION BY TerritoryID
                                         ORDER BY DATEPART(yy, ModifiedDate)
                                         ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING ), 1) AS Cumulati
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5;
```

# cudf::rolling\_window

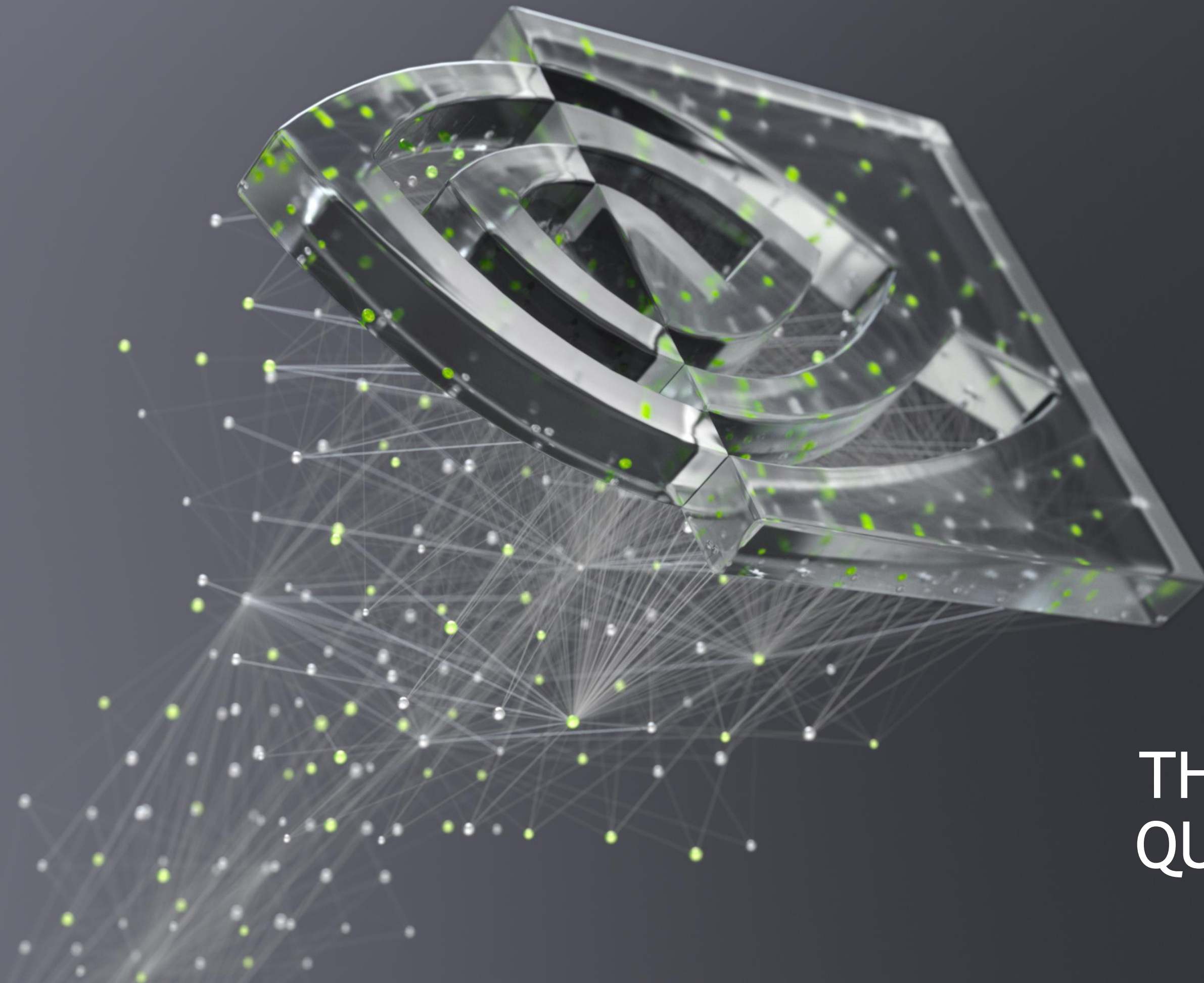
SQL

 Copy

```
select
  object_id
, [preceding]    = count(*) over(order by object_id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
, [central]     = count(*) over(order by object_id ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING )
, [following]    = count(*) over(order by object_id ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
from sys.objects
order by object_id asc
```

# DISCUSSION





THANK YOU  
QUESTIONS?

