Exercise Session 11 : FLWOR Expressions on Spark.

Agenda :

① Compound index in MongoDB (Session 10)

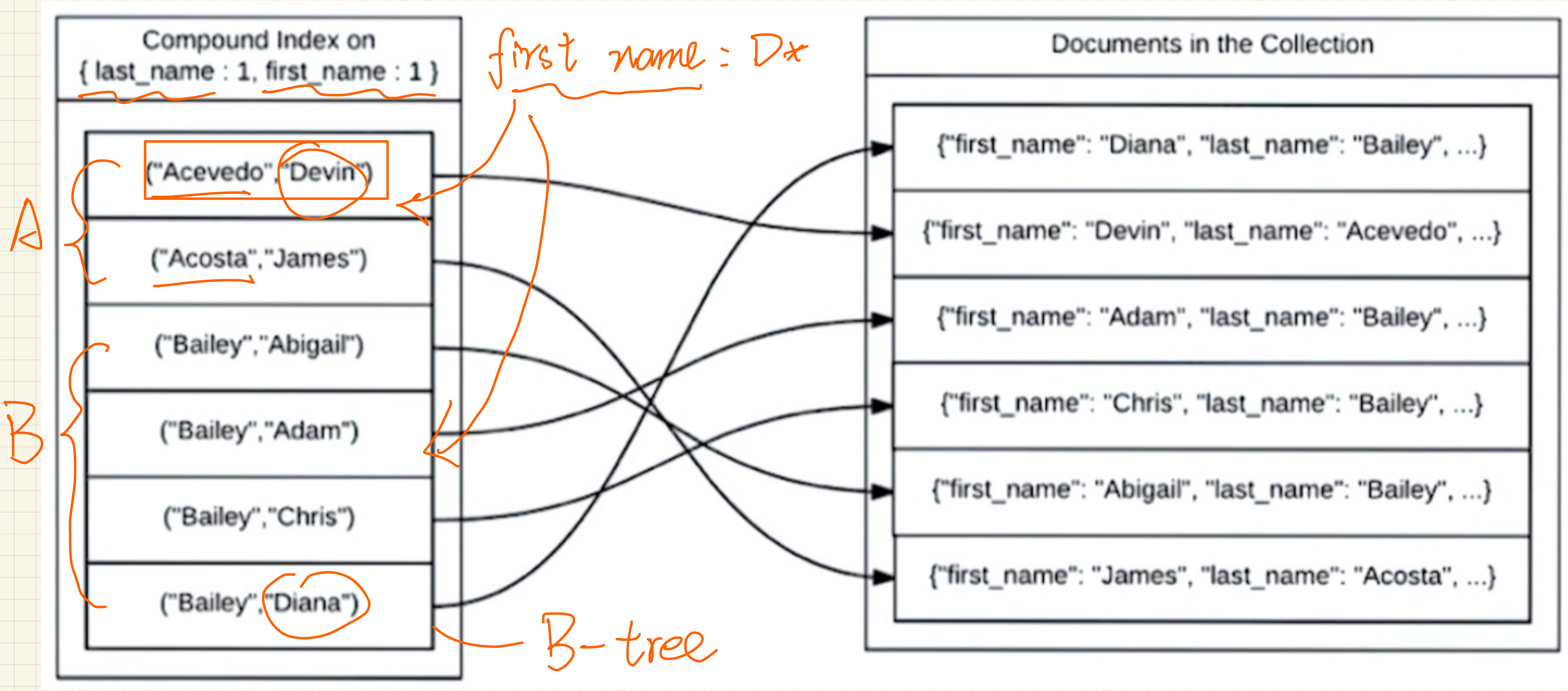
② Inner workings of the FLOWR expressions

- Tuple streams
- Execution and implementation on Spark.

③ Tips on JSONiq.

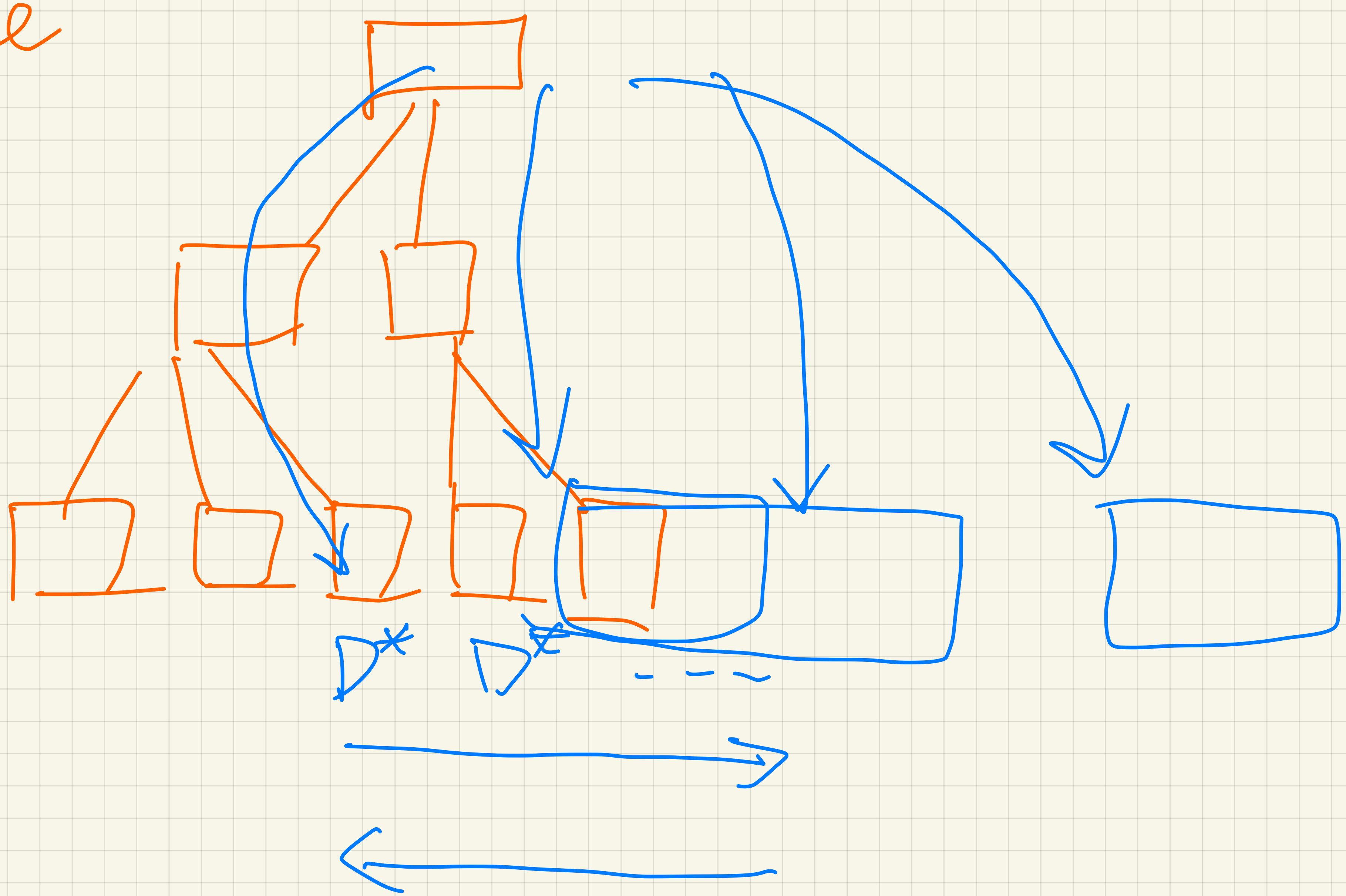
MongoDB Compound Indices (session 10)

- Compound keys are "flat" => 1 D regardless of # fields
- Non-prefix queries can only use the index as a "filter"

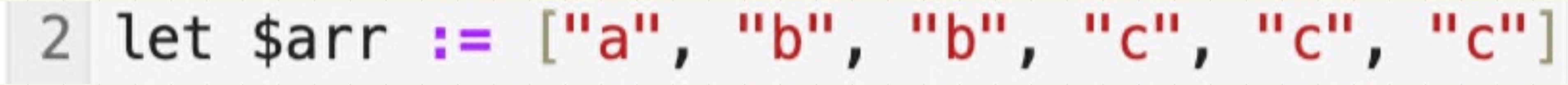


Compound Index on
{ last_name : 1, first_name : 1 }
first name : D*
("Acevedo","Devin")
("Acosta","James")
("Bailey","Abigail")
("Bailey","Adam")
("Bailey","Chris")
("Bailey","Diana")
A
B
B-tree
Documents in the Collection
{"first_name": "Diana", "last_name": "Bailey", ...}
{"first_name": "Devin", "last_name": "Acevedo", ...}
{"first_name": "Adam", "last_name": "Bailey", ...}
{"first_name": "Chris", "last_name": "Bailey", ...}
{"first_name": "Abigail", "last_name": "Bailey", ...}
{"first_name": "James", "last_name": "Acosta", ...}

B- tree

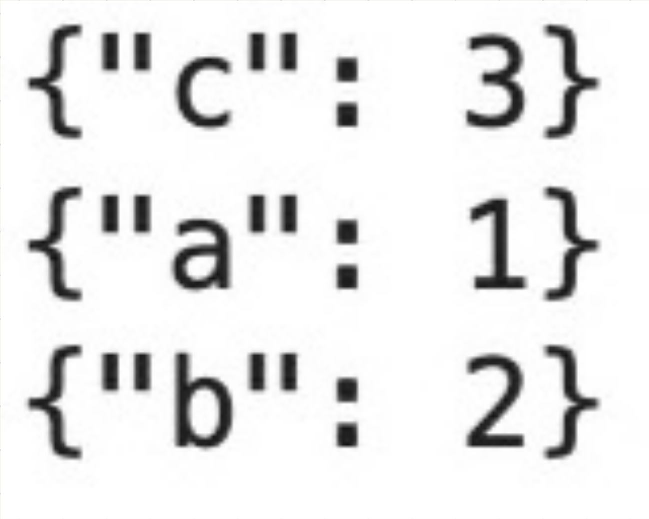


Motivating Example of JSONiq



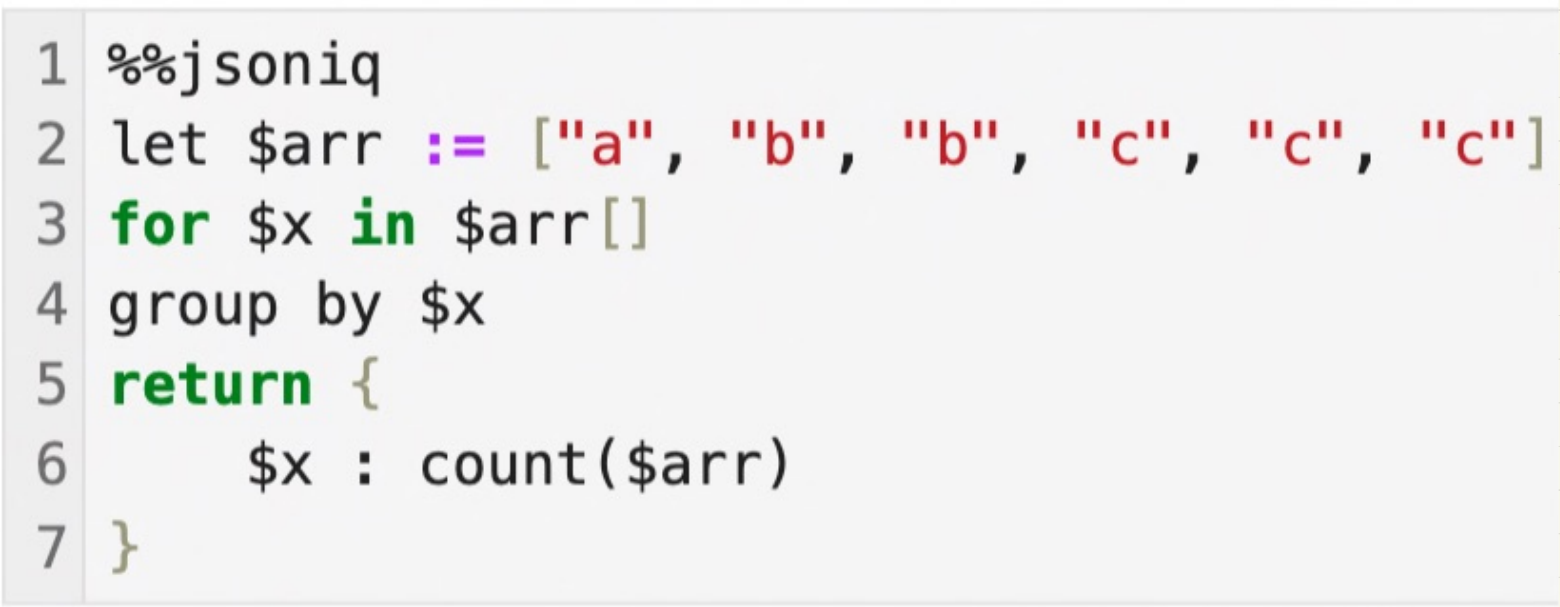
```
let $arr := ["a", "b", "b", "c", "c", "c"]
```

⟹



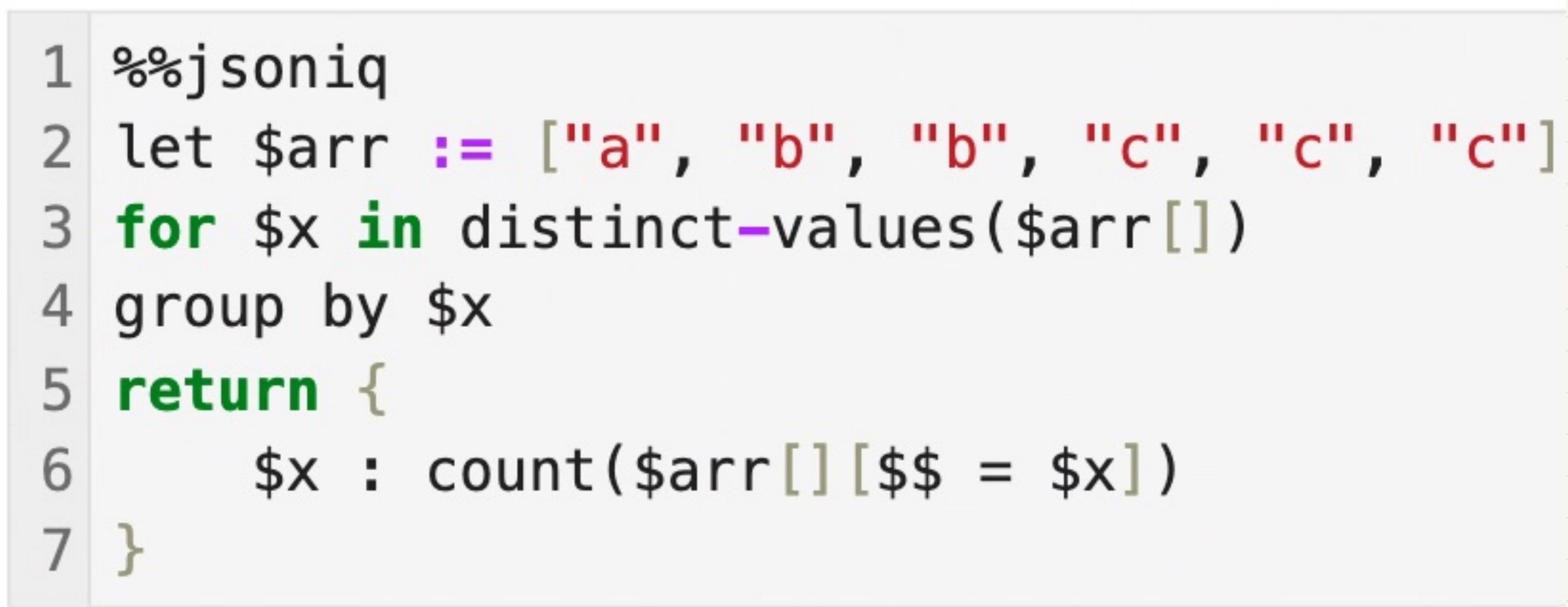
```
{"c": 3}
{"a": 1}
{"b": 2}
```

A:



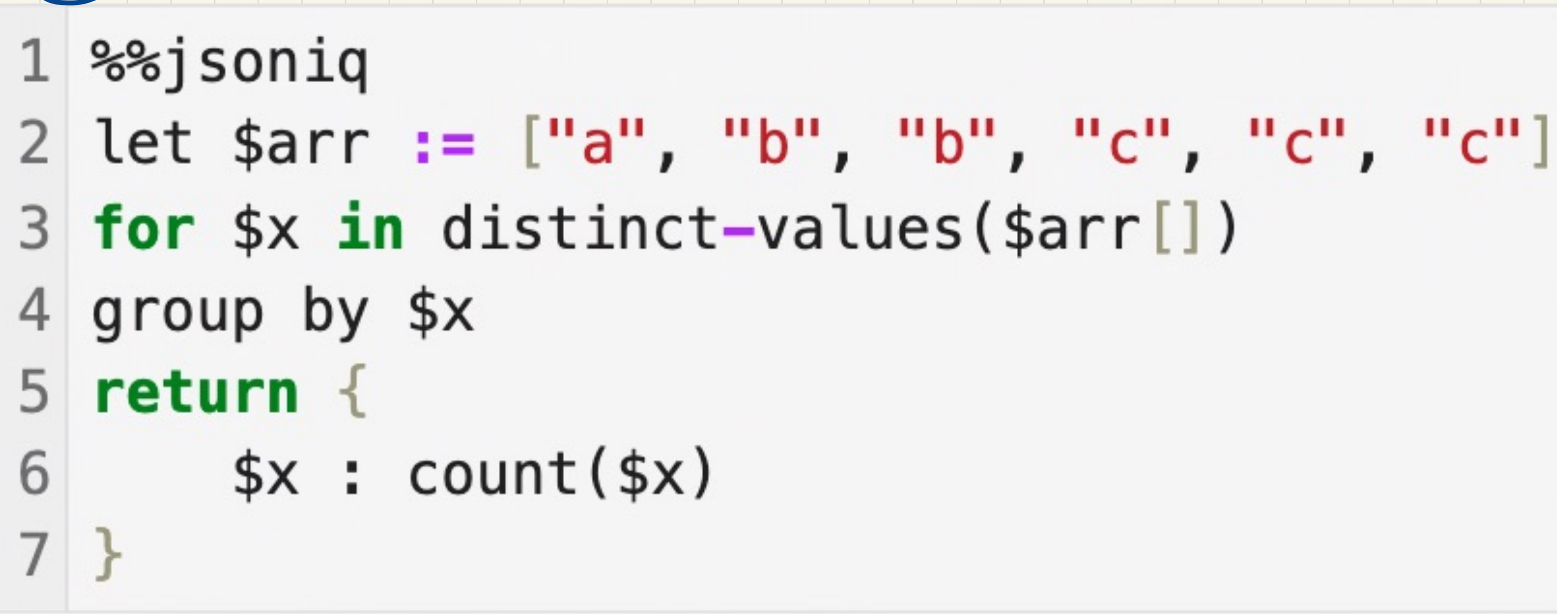
```
%%jsoniq
let $arr := ["a", "b", "b", "c", "c", "c"]
for $x in $arr[]
group by $x
return {
    $x : count($arr)
}
```

B:



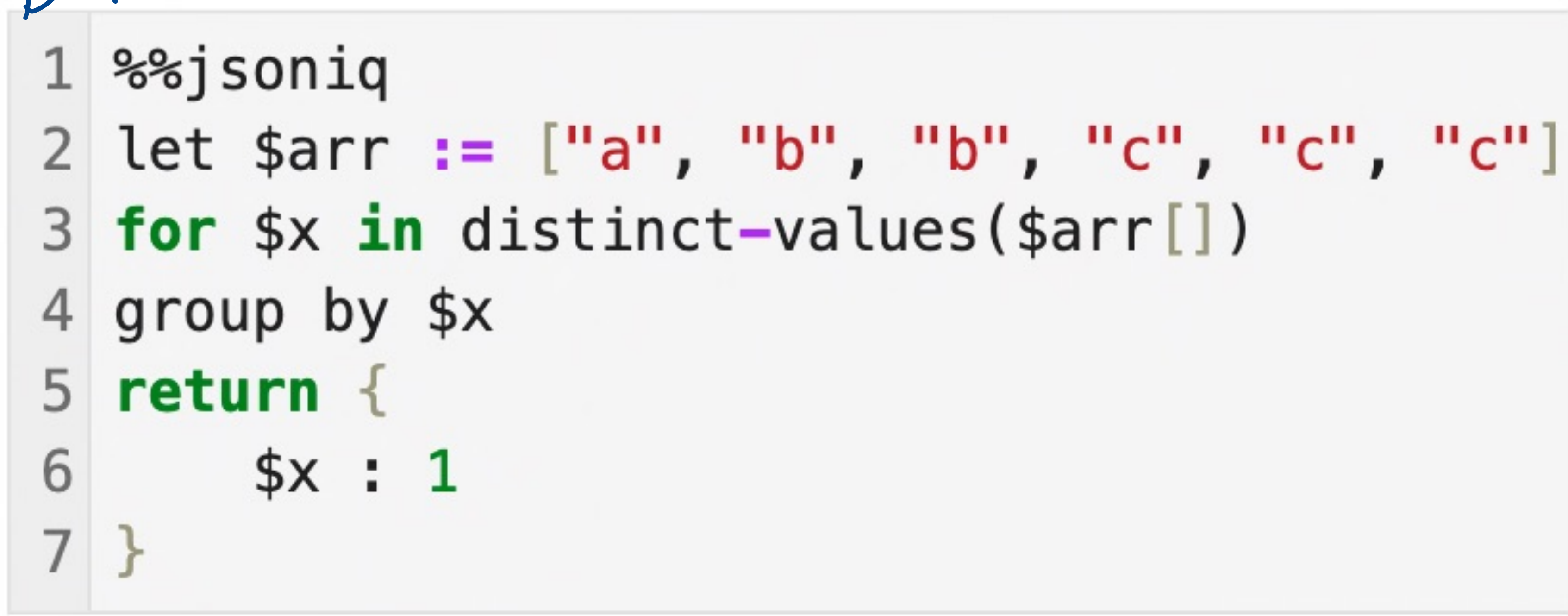
```
%%jsoniq
let $arr := ["a", "b", "b", "c", "c", "c"]
for $x in distinct-values($arr[])
group by $x
return {
    $x : count($arr[][$$ = $x])
}
```

C:



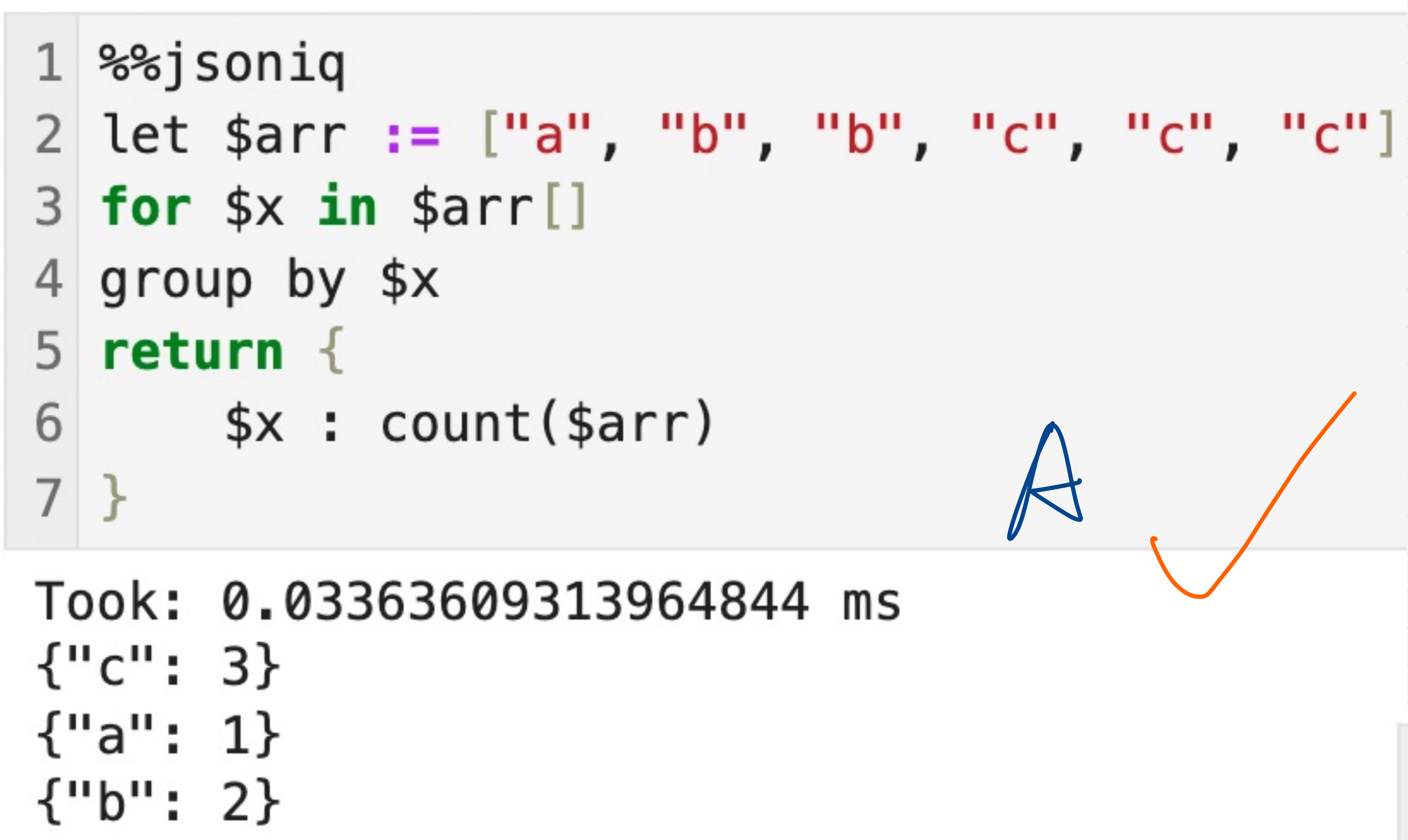
```
%%jsoniq
let $arr := ["a", "b", "b", "c", "c", "c"]
for $x in distinct-values($arr[])
group by $x
return {
    $x : count($x)
}
```

D:



```
%%jsoniq
let $arr := ["a", "b", "b", "c", "c", "c"]
for $x in distinct-values($arr[])
group by $x
return {
    $x : 1
}
```

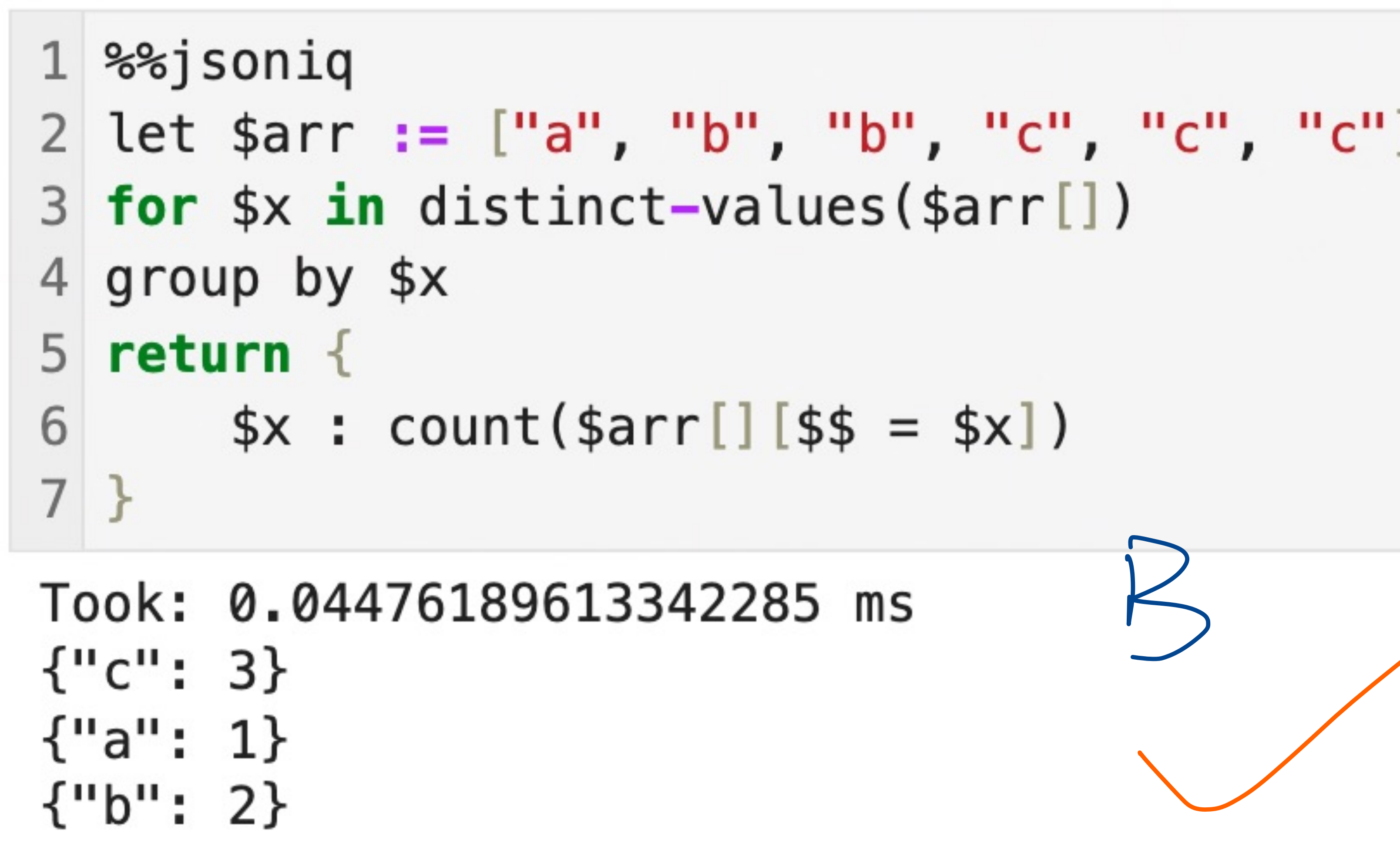
Outputs :



```
%%jsoniq
let $arr := ["a", "b", "b", "c", "c", "c"]
for $x in $arr[]
group by $x
return {
     $x : count($arr)
}
```

A ✓

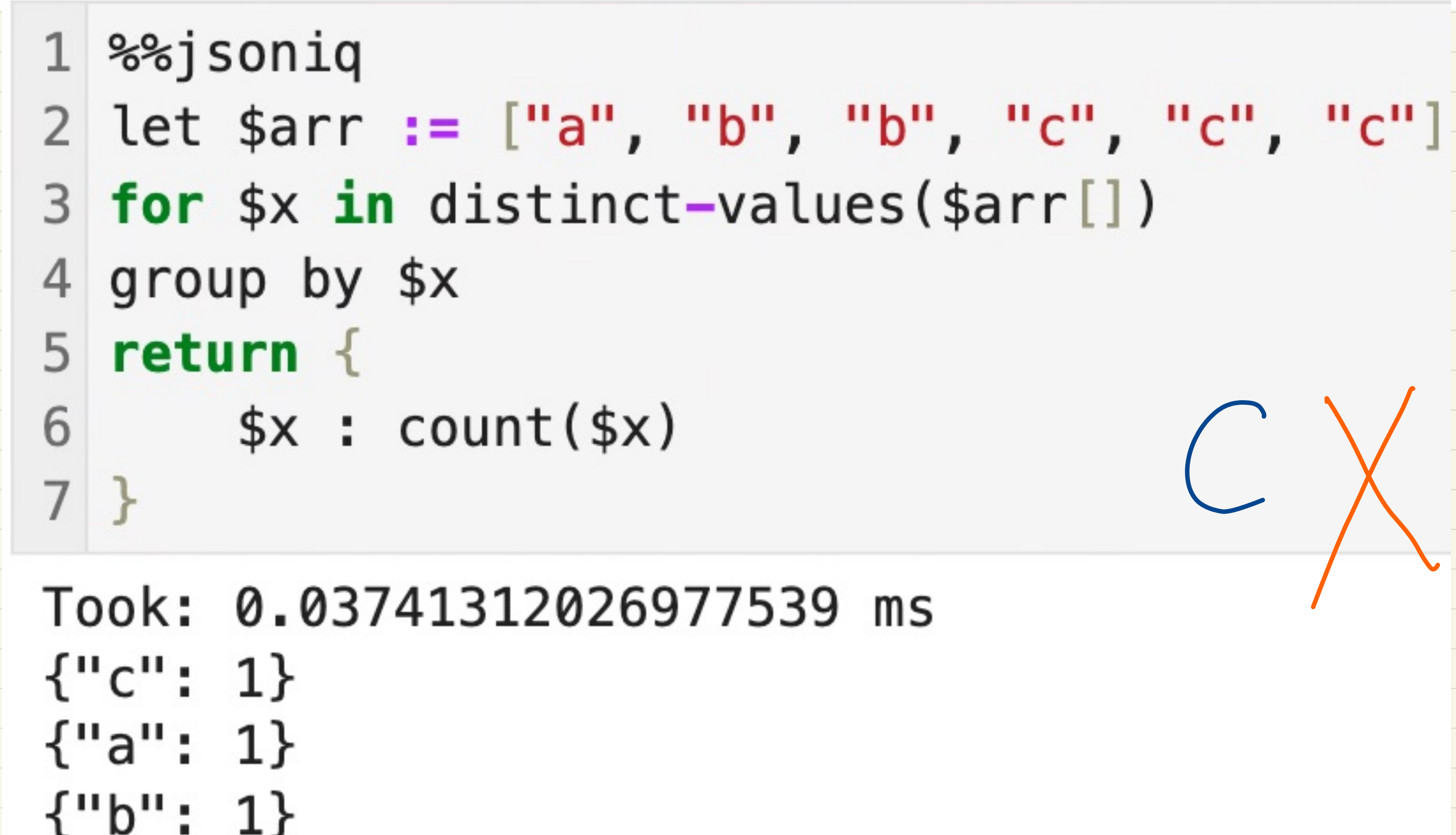
```
Took: 0.03363609313964844 ms
{"c": 3}
{"a": 1}
{"b": 2}
```



```
%%jsoniq
let $arr := ["a", "b", "b", "c", "c", "c"
for $x in distinct-values($arr[])
group by $x
return {
     $x : count($arr[][$$ = $x])
}
```

B ✓

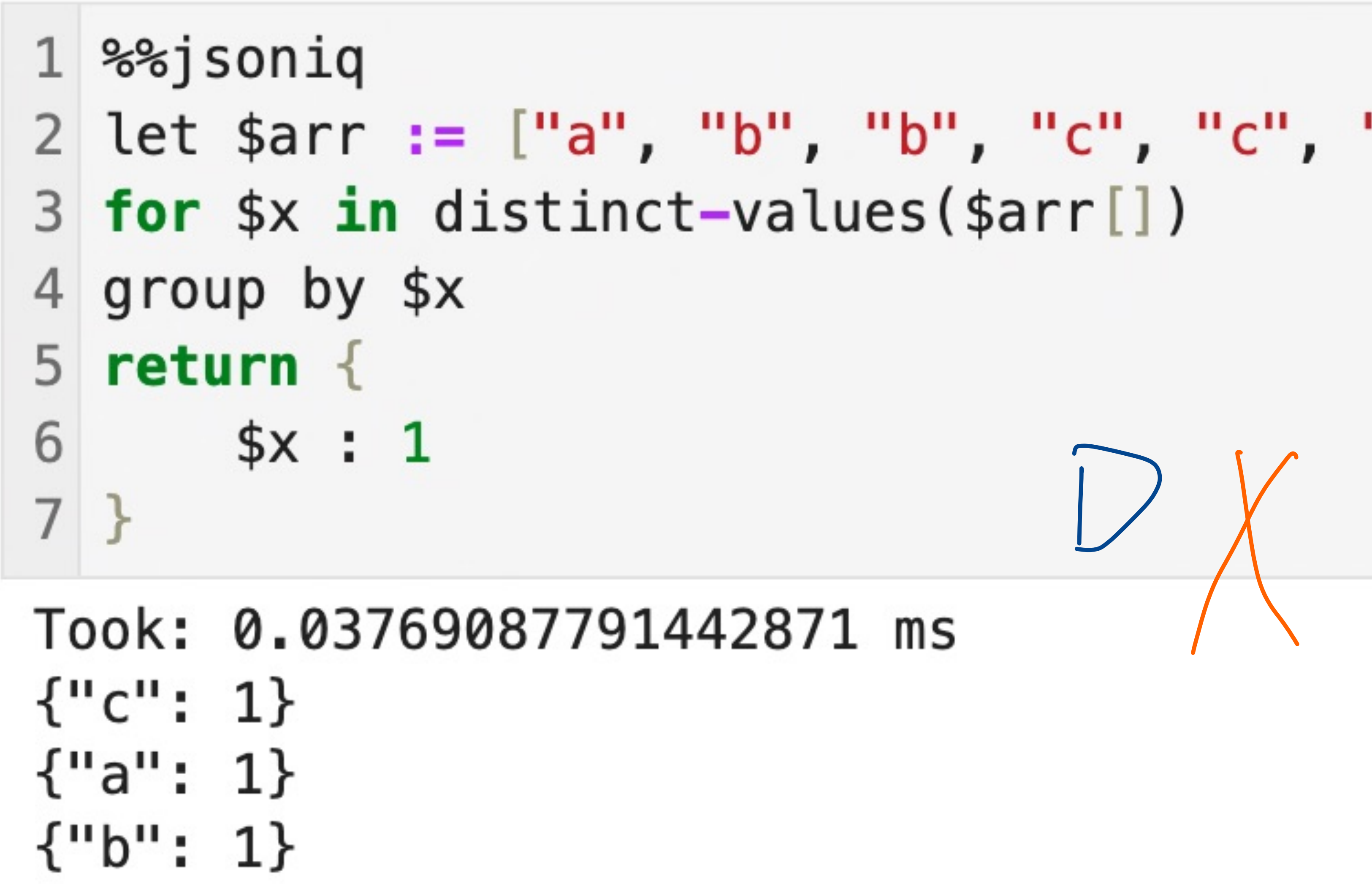
```
Took: 0.04476189613342285 ms
{"c": 3}
{"a": 1}
{"b": 2}
```



```
%%jsoniq
let $arr := ["a", "b", "b", "c", "c", "c"]
for $x in distinct-values($arr[])
group by $x
return {
     $x : count($x)
}
```

C ✗

```
Took: 0.03741312026977539 ms
{"c": 1}
{"a": 1}
{"b": 1}
```



```
%%jsoniq
let $arr := ["a", "b", "b", "c", "c", "
for $x in distinct-values($arr[])
group by $x
return {
     $x : 1
}
```

D ✗

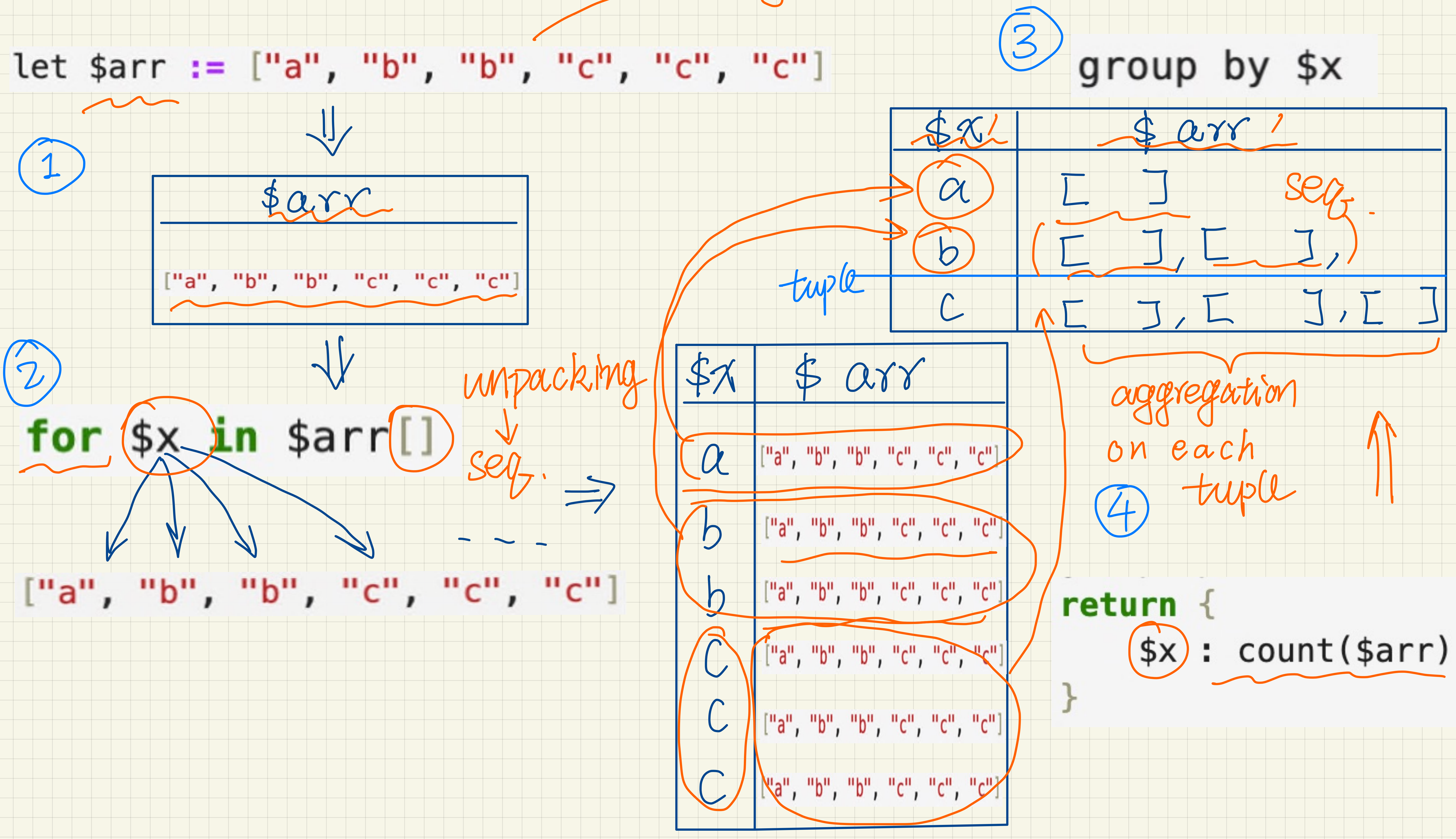
```
Took: 0.03769087791442871 ms
{"c": 1}
{"a": 1}
{"b": 1}
```

FLOWR ~~Semantics~~ Effects

- `let` — Creates a new column
- `for` — Reshapes a list of values into a column
- `where` — Fliters out tuples given a criterion
- `group by` — Collects tuples of the same key to a sequence
- `return` — Emits streams of tuples one by one

SELECT

Example Breakdown



```
let $arr := ["a", "b", "b", "c", "c", "c"]
```

$arr
["a", "b", "b", "c", "c", "c"]

```
for $x in $arr[]
```

```
["a", "b", "b", "c", "c", "c"]
```

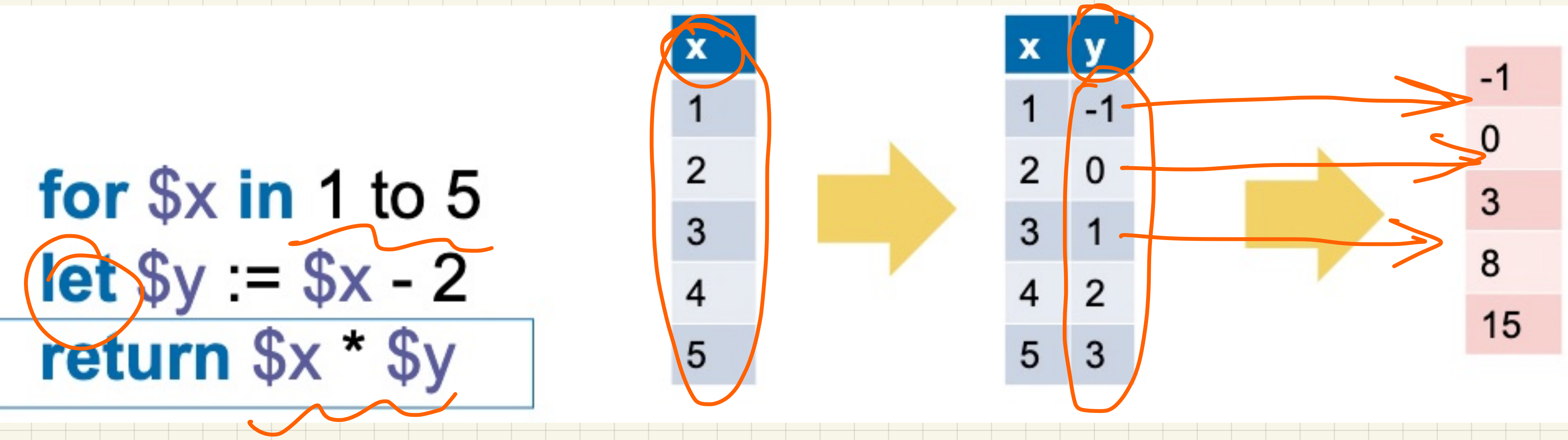
$x	$ arr
a	["a", "b", "b", "c", "c", "c"]
b	["a", "b", "b", "c", "c", "c"]
b	["a", "b", "b", "c", "c", "c"]
c	["a", "b", "b", "c", "c", "c"]
c	["a", "b", "b", "c", "c", "c"]
c	["a", "b", "b", "c", "c", "c"]

```
group by $x
```

$x'	$ arr'
a	[ ]
b	([ ], [ ],)
c	[ ], [ ], [ ]

```
return {
    $x : count($arr)
}
```

More Examples from Lectures

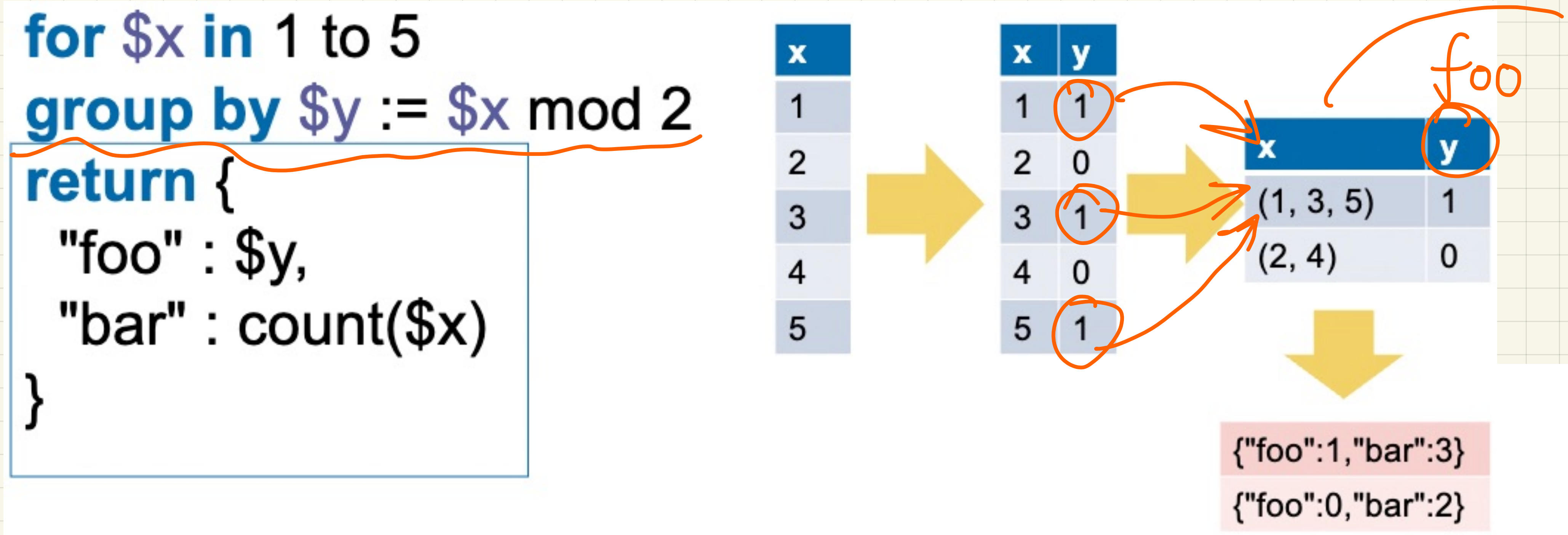


```
for $x in 1 to 5
let $y := $x - 2
return $x * $y
```

x
1
2
3
4
5

x	y
1	-1
2	0
3	1
4	2
5	3

-1
0
3
8
15



```
for $x in 1 to 5
group by $y := $x mod 2
return {
  "foo" : $y,
  "bar" : count($x)
}
```

x
1
2
3
4
5

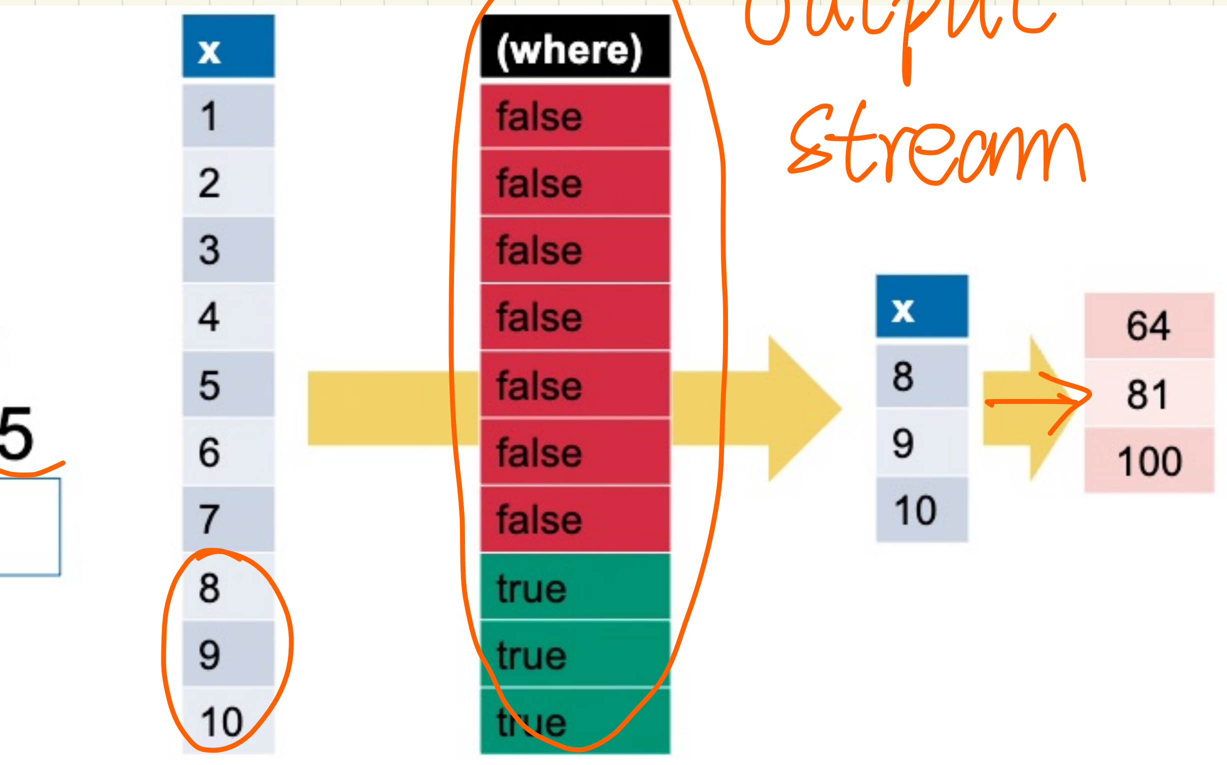
x	y
1	1
2	0
3	1
4	0
5	1

a sequence

foo

x	y
(1, 3, 5)	1
(2, 4)	0

{"foo":1,"bar":3}
{"foo":0,"bar":2}



```
for $x in 1 to 10
where $x - 2 gt 5
return $x * $x
```

x	(where)
1	false
2	false
3	false
4	false
5	false
6	false
7	false
8	true
9	true
10	true

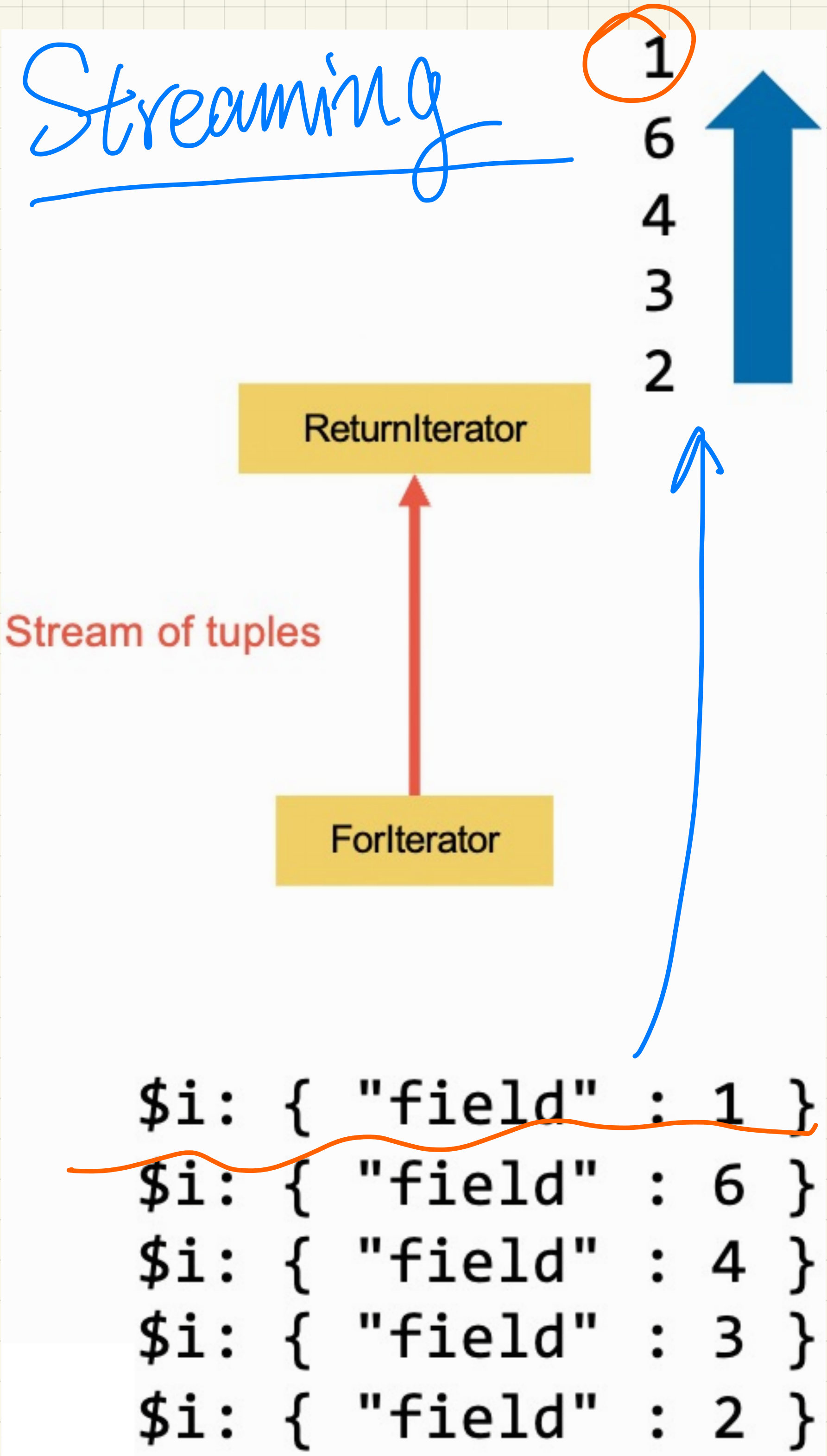
output stream

x
8
9
10

64
81
100

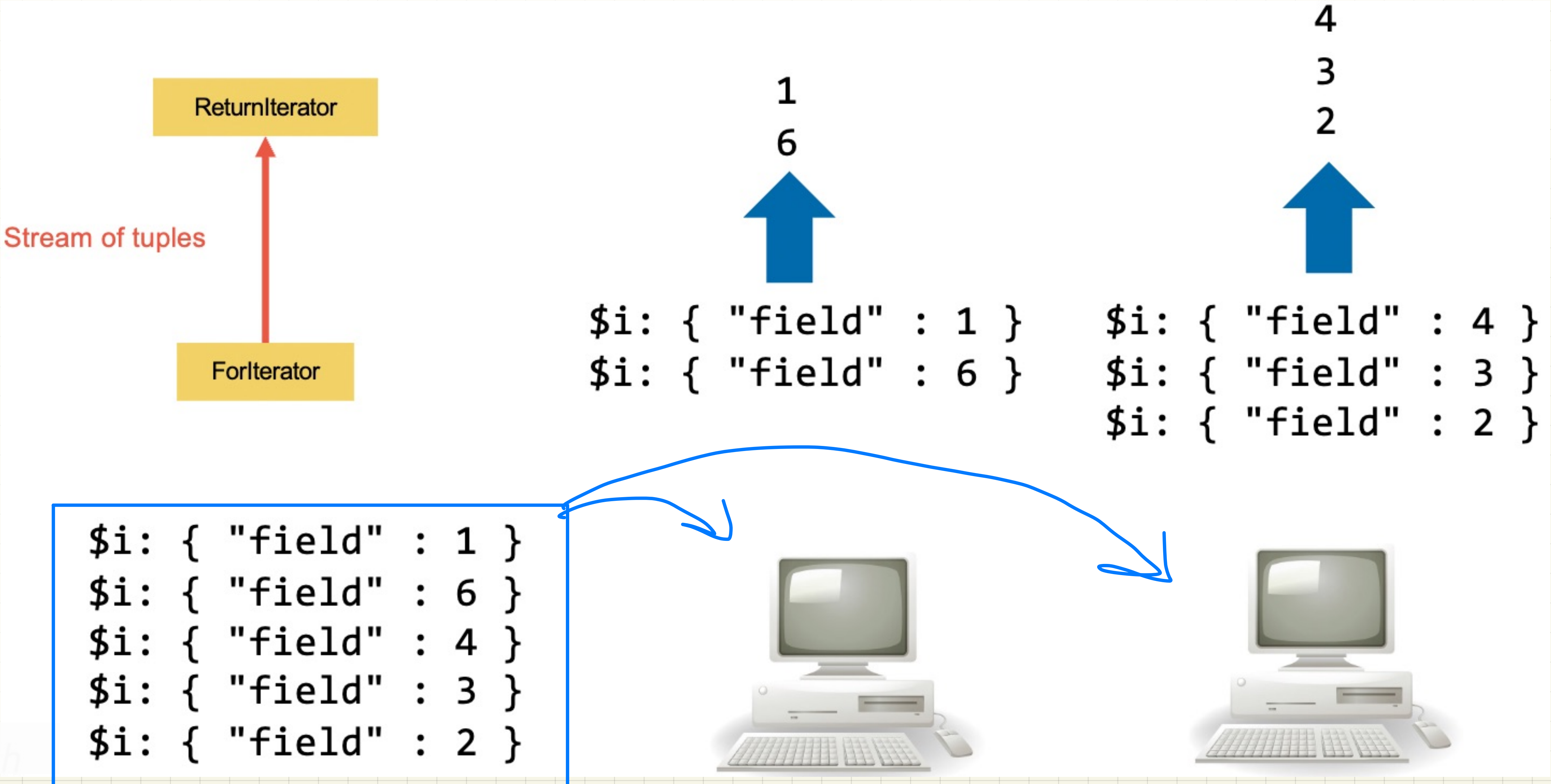
Execution

Streaming



```
$i: { "field" : 1 }
$i: { "field" : 6 }
$i: { "field" : 4 }
$i: { "field" : 3 }
$i: { "field" : 2 }
```

Parallel



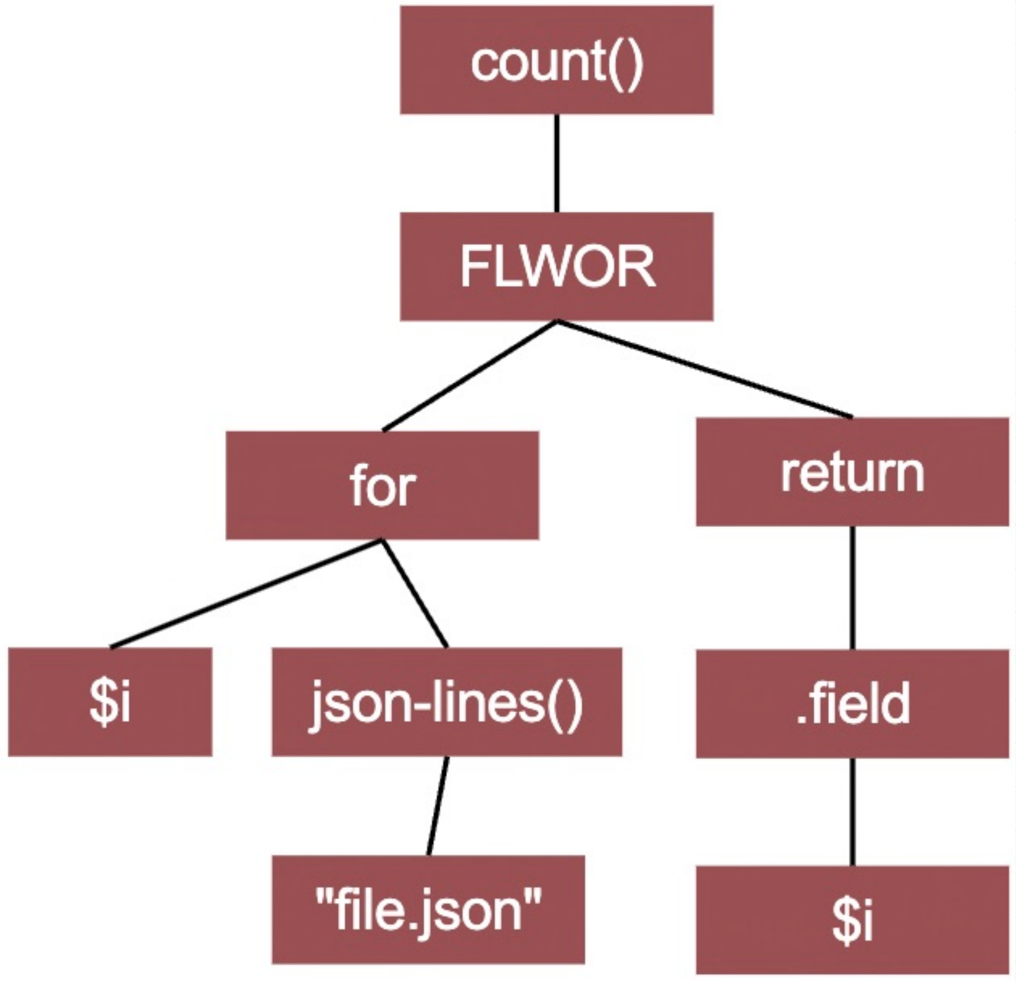
```
$i: { "field" : 1 }
$i: { "field" : 6 }
```

```
$i: { "field" : 4 }
$i: { "field" : 3 }
$i: { "field" : 2 }
```

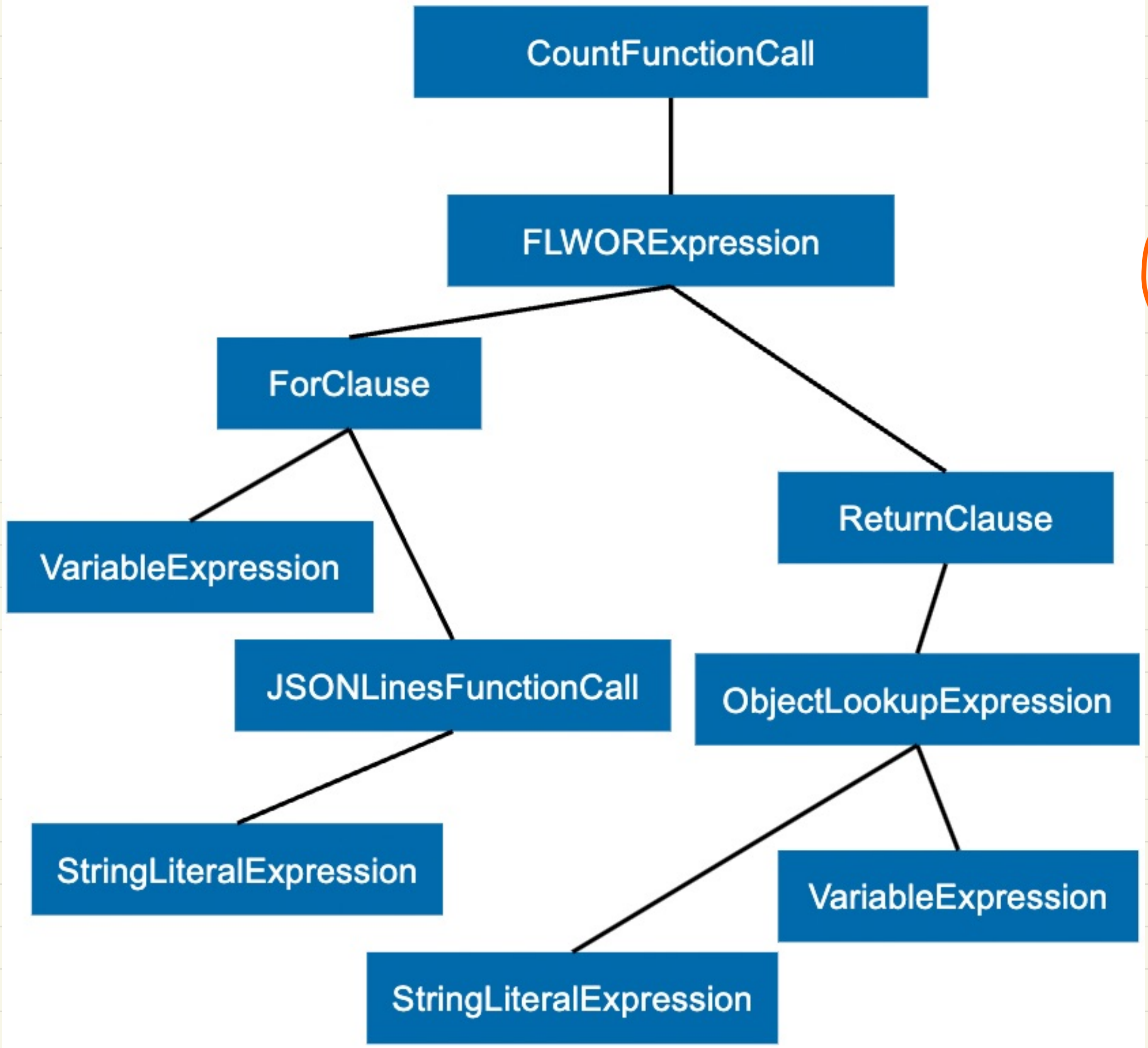
```
$i: { "field" : 1 }
$i: { "field" : 6 }
$i: { "field" : 4 }
$i: { "field" : 3 }
$i: { "field" : 2 }
```

RDD

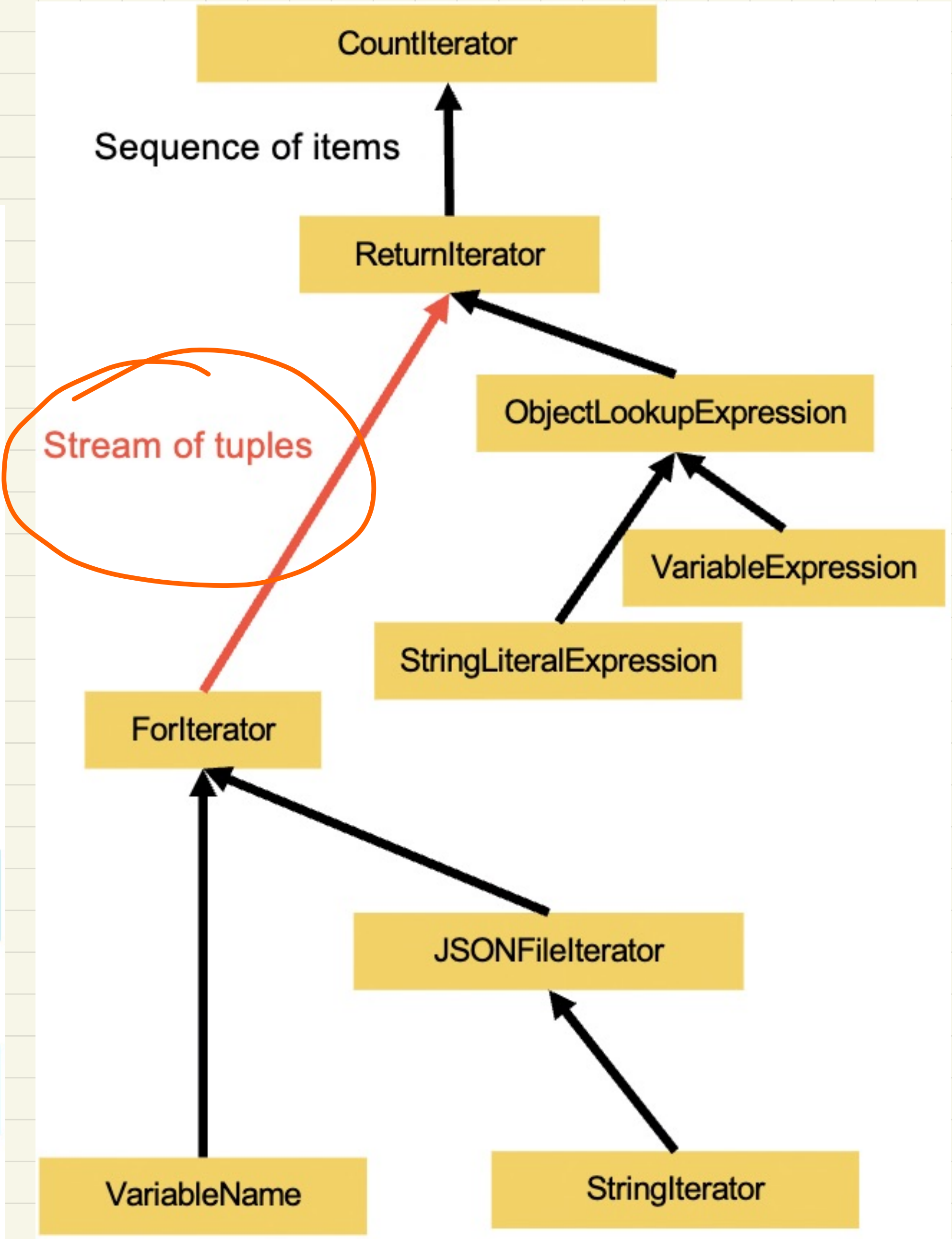
Implimentation atop Spark



AST



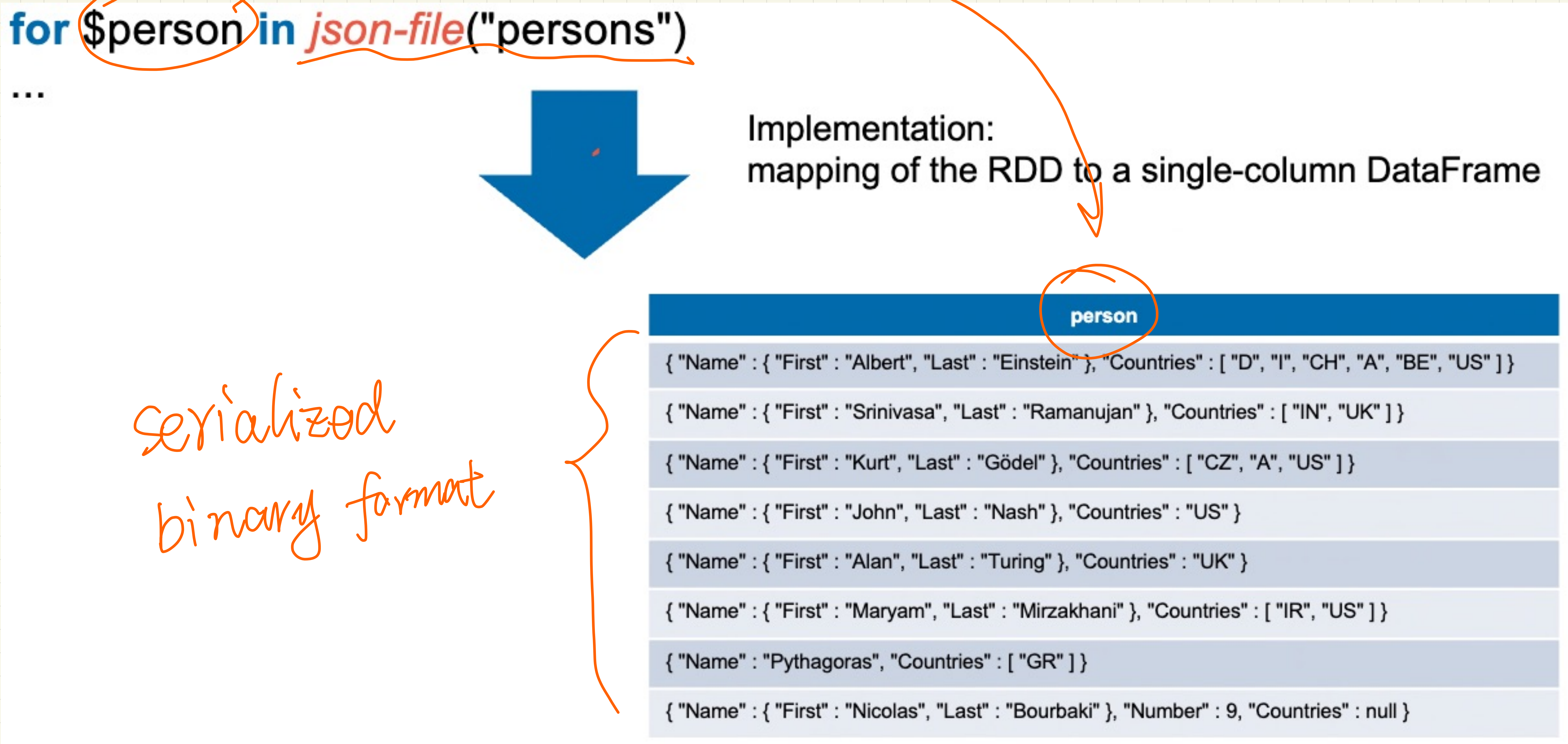
Expression Tree



Iterator Tree

Spark Implementation

new col

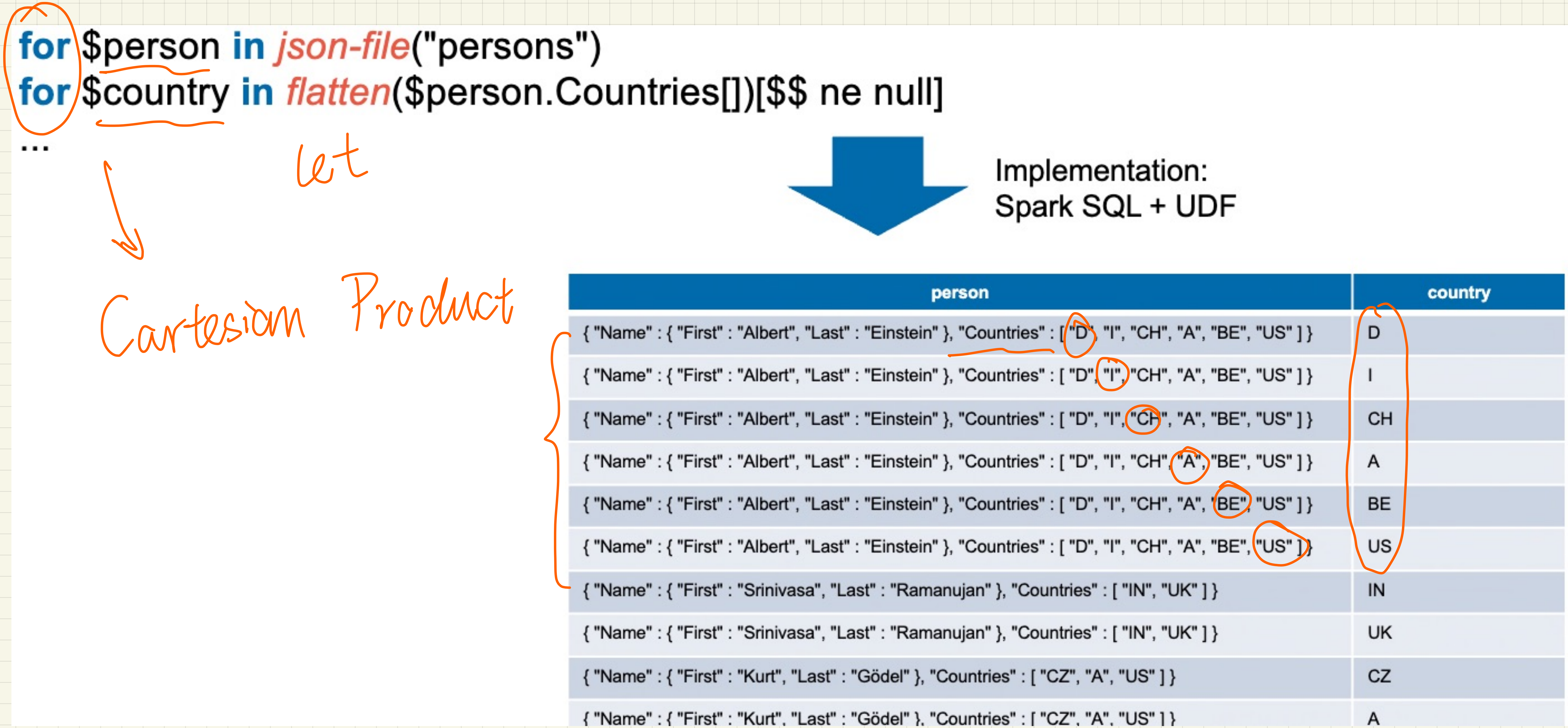


```
for $person in json-file("persons")
...
```

Implementation:
mapping of the RDD to a single-column DataFrame

person
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }
{ "Name" : { "First" : "Srinivasa", "Last" : "Ramanujan" }, "Countries" : ["IN", "UK"] }
{ "Name" : { "First" : "Kurt", "Last" : "Gödel" }, "Countries" : ["CZ", "A", "US"] }
{ "Name" : { "First" : "John", "Last" : "Nash" }, "Countries" : "US" }
{ "Name" : { "First" : "Alan", "Last" : "Turing" }, "Countries" : "UK" }
{ "Name" : { "First" : "Maryam", "Last" : "Mirzakhani" }, "Countries" : ["IR", "US"] }
{ "Name" : "Pythagoras", "Countries" : ["GR"] }
{ "Name" : { "First" : "Nicolas", "Last" : "Bourbaki" }, "Number" : 9, "Countries" : null }

serialized
binary format



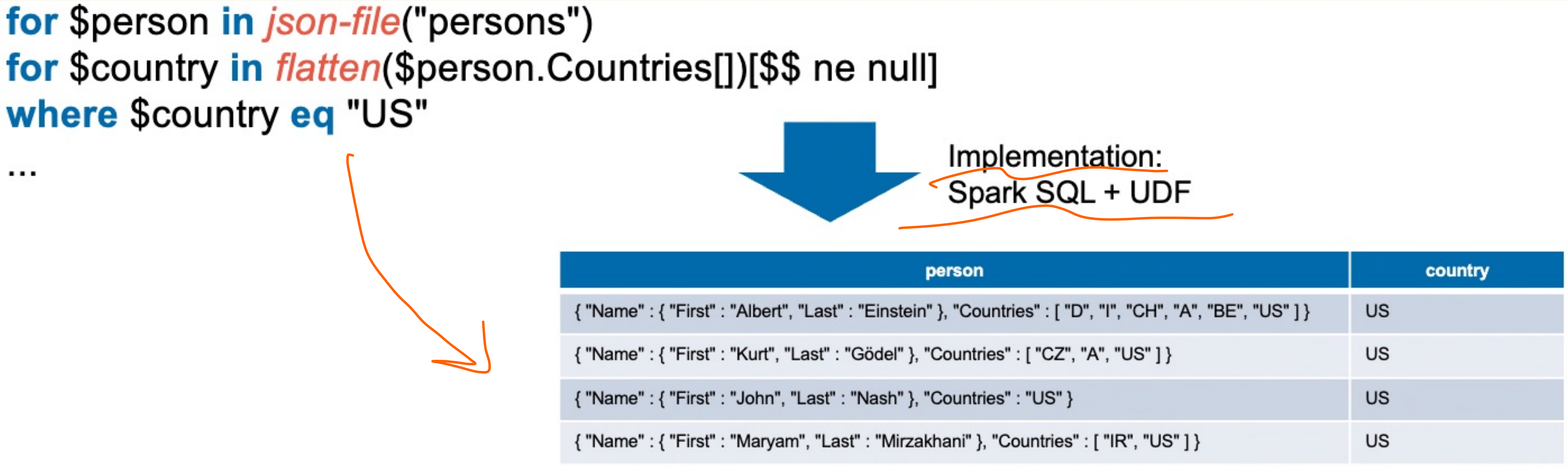
```
for $person in json-file("persons")
for $country in flatten($person.Countries[])[$$ ne null]
...
```

let

Cartesian Product

Implementation:
Spark SQL + UDF

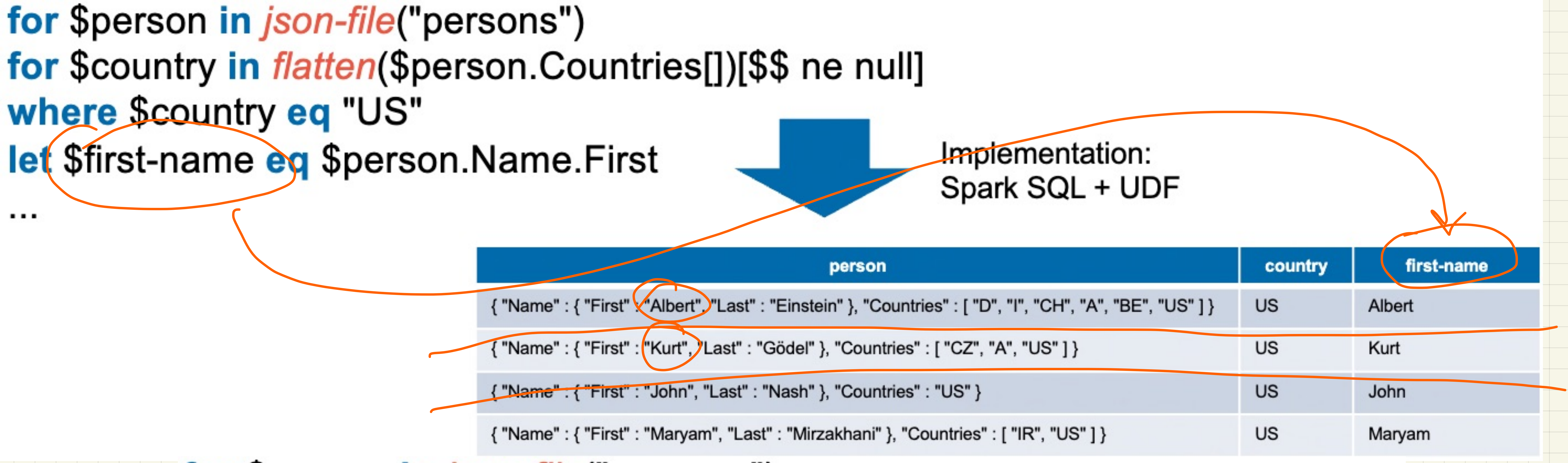
person	country
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }	D
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }	I
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }	CH
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }	A
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }	BE
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }	US
{ "Name" : { "First" : "Srinivasa", "Last" : "Ramanujan" }, "Countries" : ["IN", "UK"] }	IN
{ "Name" : { "First" : "Srinivasa", "Last" : "Ramanujan" }, "Countries" : ["IN", "UK"] }	UK
{ "Name" : { "First" : "Kurt", "Last" : "Gödel" }, "Countries" : ["CZ", "A", "US"] }	CZ
{ "Name" : { "First" : "Kurt", "Last" : "Gödel" }, "Countries" : ["CZ", "A", "US"] }	A



```
for $person in json-file("persons")
for $country in flatten($person.Countries[])[$$ ne null]
where $country eq "US"
...
```

Implementation:
Spark SQL + UDF

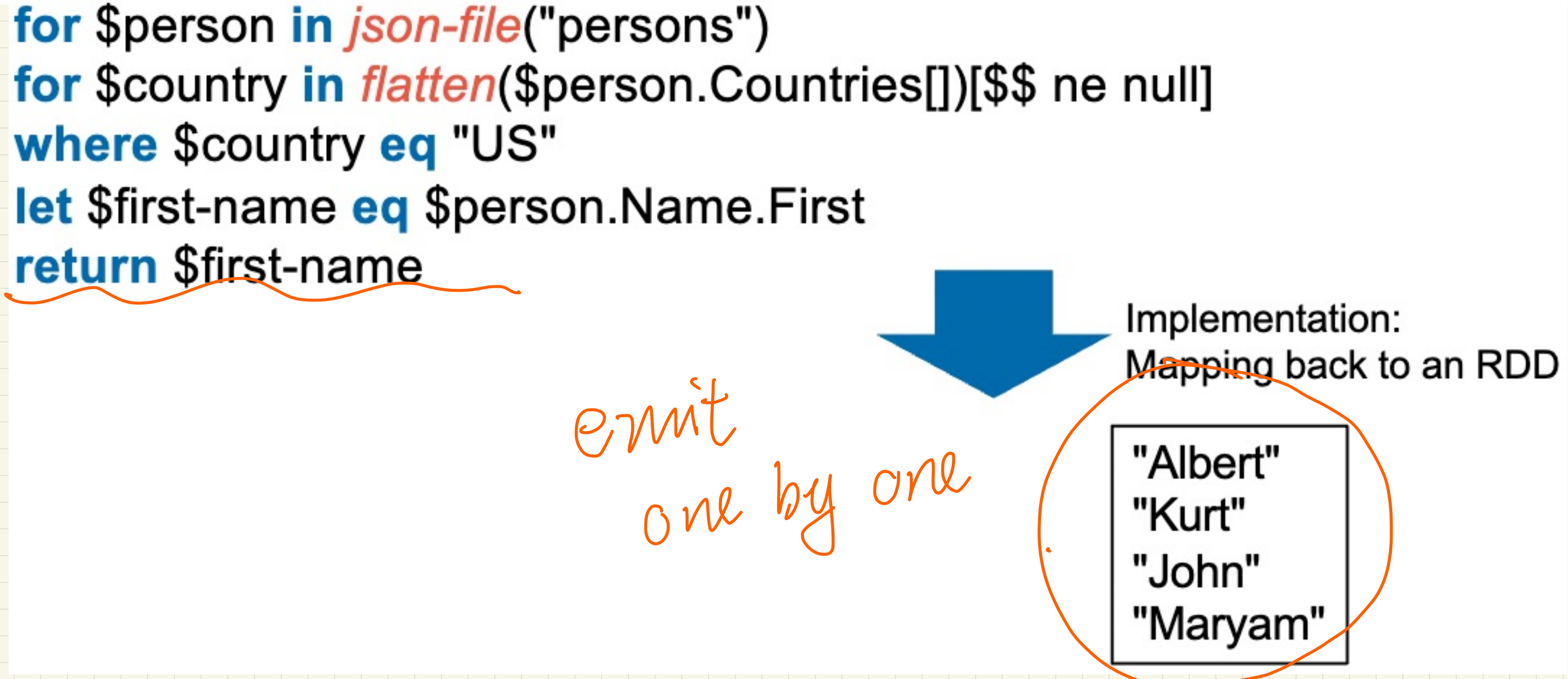
person	country
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }	US
{ "Name" : { "First" : "Kurt", "Last" : "Gödel" }, "Countries" : ["CZ", "A", "US"] }	US
{ "Name" : { "First" : "John", "Last" : "Nash" }, "Countries" : "US" }	US
{ "Name" : { "First" : "Maryam", "Last" : "Mirzakhani" }, "Countries" : ["IR", "US"] }	US



```
for $person in json-file("persons")
for $country in flatten($person.Countries[])[$$ ne null]
where $country eq "US"
let $first-name eq $person.Name.First
...
```

Implementation:
Spark SQL + UDF

person	country	first-name
{ "Name" : { "First" : "Albert", "Last" : "Einstein" }, "Countries" : ["D", "I", "CH", "A", "BE", "US"] }	US	Albert
{ "Name" : { "First" : "Kurt", "Last" : "Gödel" }, "Countries" : ["CZ", "A", "US"] }	US	Kurt
{ "Name" : { "First" : "John", "Last" : "Nash" }, "Countries" : "US" }	US	John
{ "Name" : { "First" : "Maryam", "Last" : "Mirzakhani" }, "Countries" : ["IR", "US"] }	US	Maryam



```
for $person in json-file("persons")
for $country in flatten($person.Countries[])[$$ ne null]
where $country eq "US"
let $first-name eq $person.Name.First
return $first-name
```

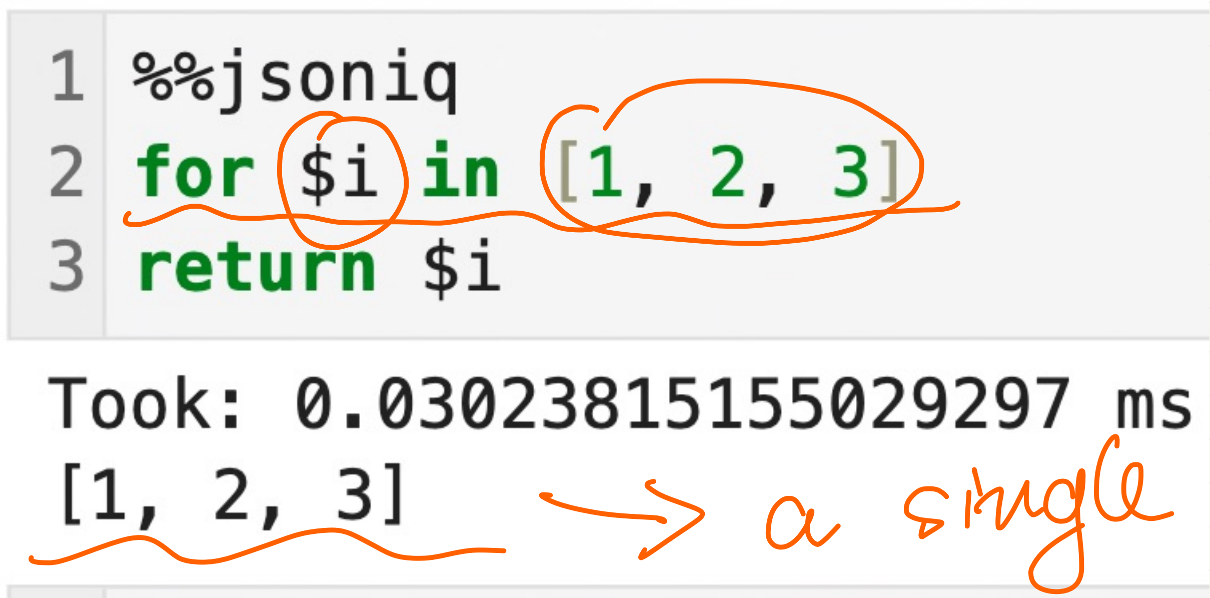
Implementation:
Mapping back to an RDD

```
"Albert"
"Kurt"
"John"
"Maryam"
```

emit
one by one

Final Tips

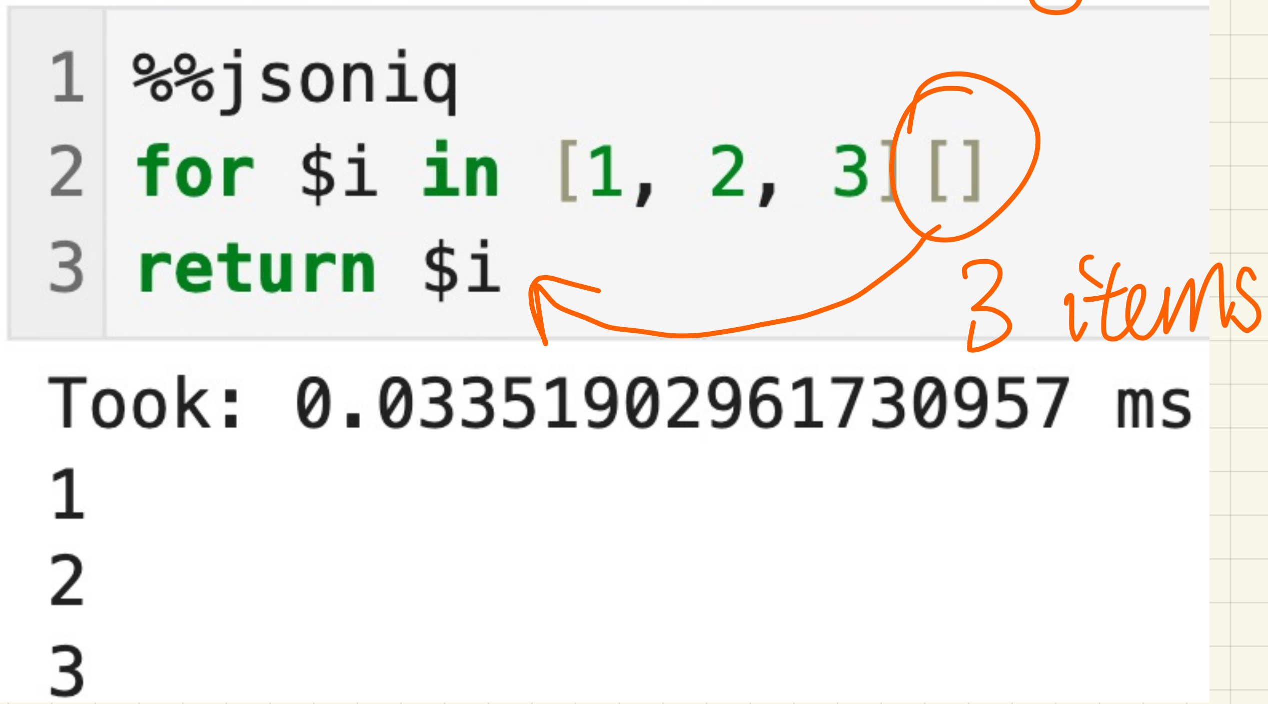
① Iterate over arrays :



```
%%jsoniq
for $i in [1, 2, 3]
return $i
```

```
Took: 0.03023815155029297 ms
[1, 2, 3]
```

→ a single item

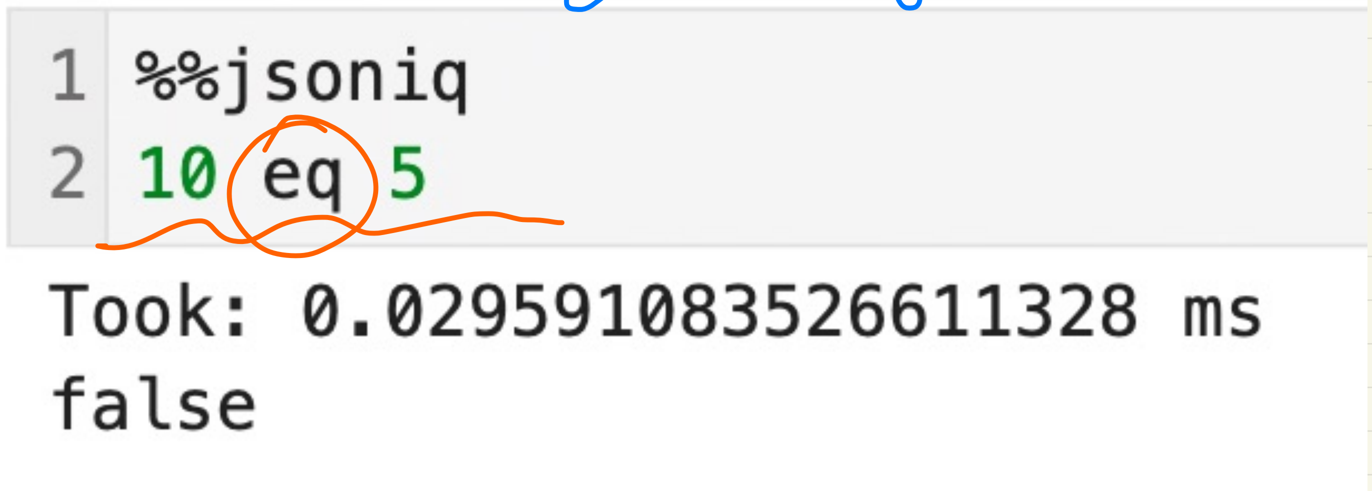


```
%%jsoniq
for $i in [1, 2, 3][]
return $i
```

```
Took: 0.03351902961730957 ms
1
2
3
```

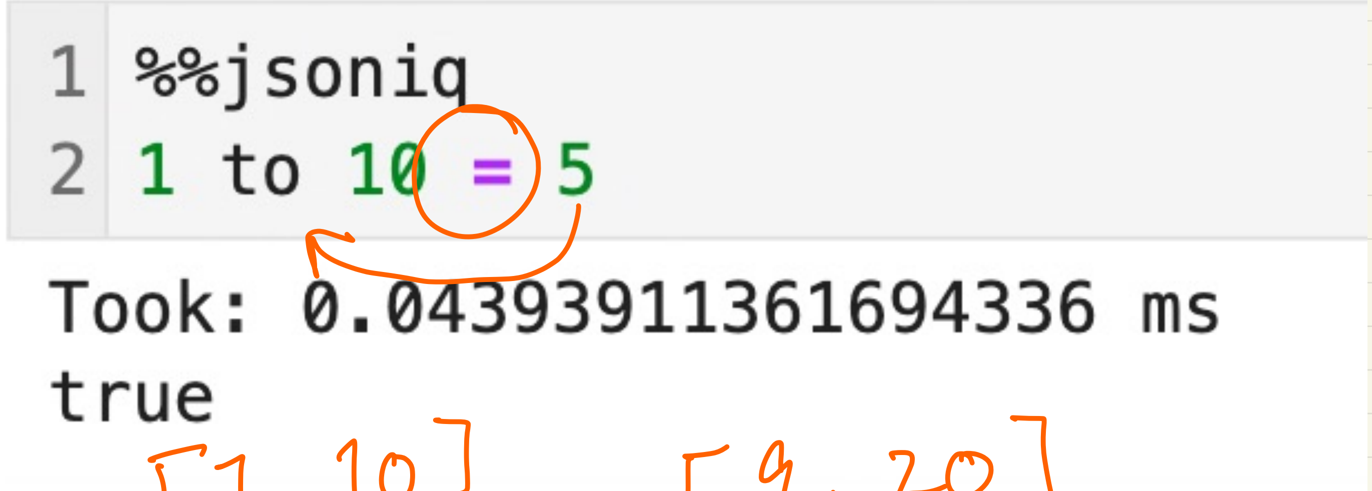
3 items (ints)

② Implicit existential quantification :



```
%%jsoniq
10 eq 5
```

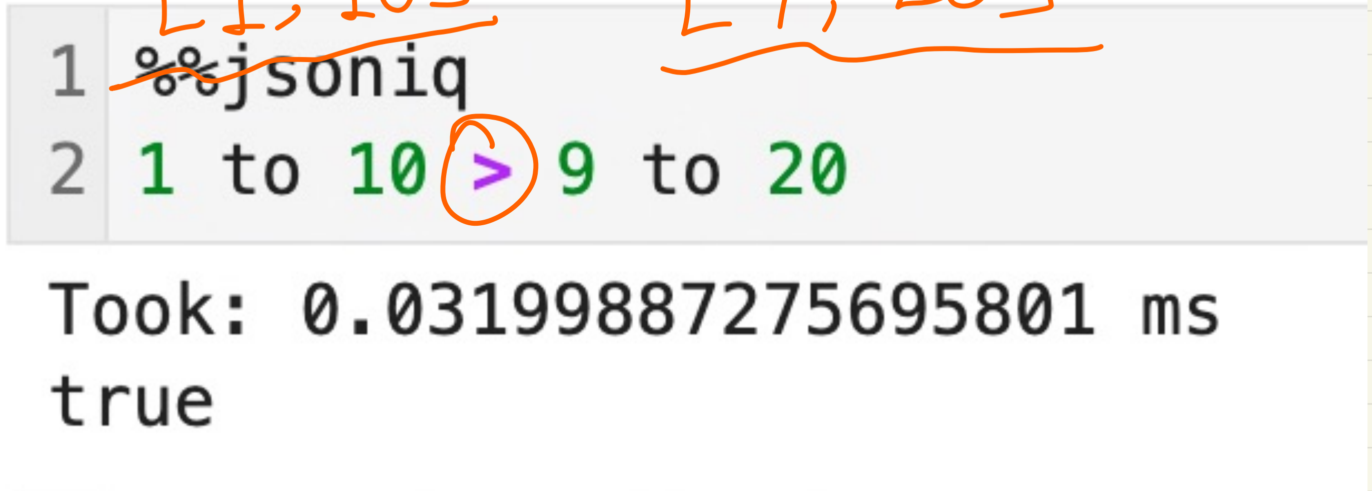
```
Took: 0.029591083526611328 ms
false
```



```
%%jsoniq
1 to 10 = 5
```

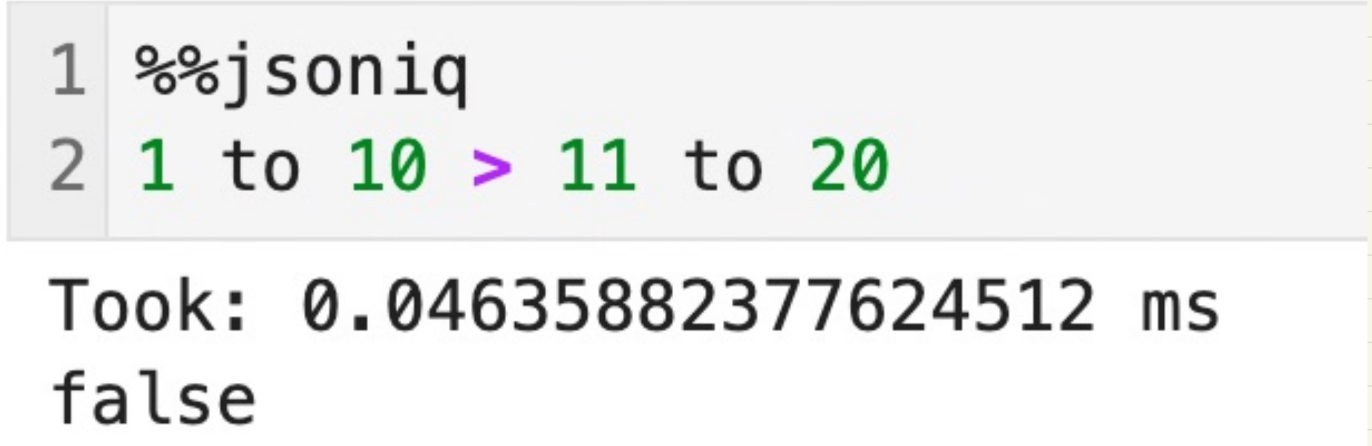
```
Took: 0.043931136169433594 ms
true
```

[1, 10] [9, 20]



```
%%jsoniq
1 to 10 > 9 to 20
```

```
Took: 0.031998872756958008 ms
true
```



```
%%jsoniq
1 to 10 > 11 to 20
```

```
Took: 0.04635882377624512 ms
false
```