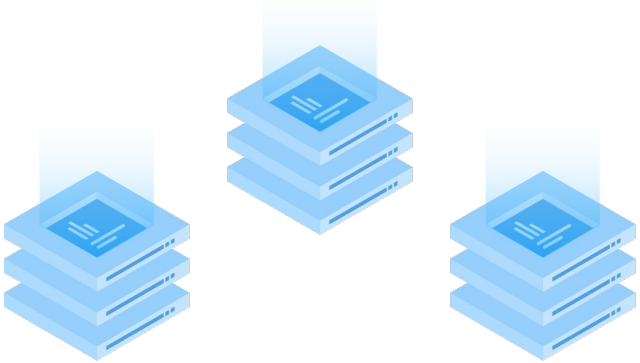
Recent work on Vectorized exectuor

Presented by Feng Liyuan

PingCAP TiDB



About me

- Feng Liyuan (冯立元)
- Was: Experienced Engineer, Cloud Storage, Qiniu
- Now: Engineer, TiDB SQL Engine Team
- Focus on: TiDB Runtime

PingCAP

Agenda

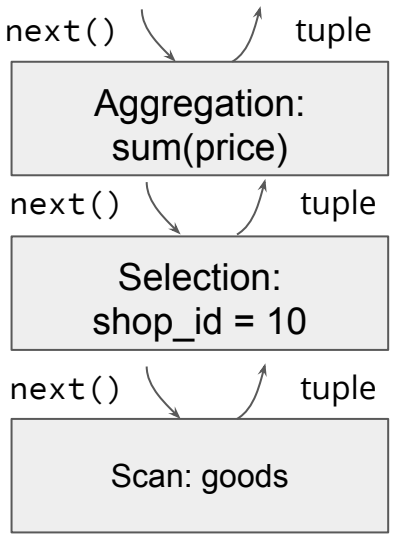
- Backgroud
  - What is executor?
  - What is vectorized executor on TiDB
- Recent work
  - Hash Join
  - Stream Aggregation
  - Vectorized expression evaluation
- Future work & Call for participation

PingCAP

Part I - Backgroud

Row-based executor
Volcano Iterator Model

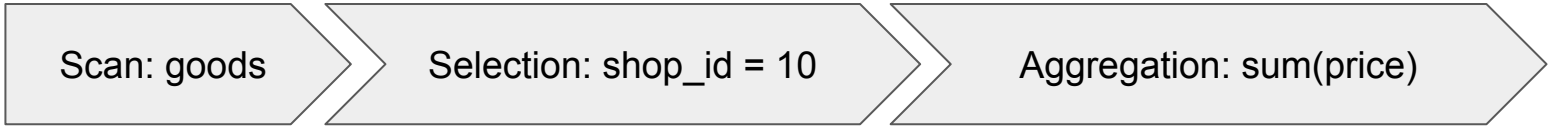
```
SELECT SUM(price)
FROM goods
WHERE shop_id = 10
```



Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

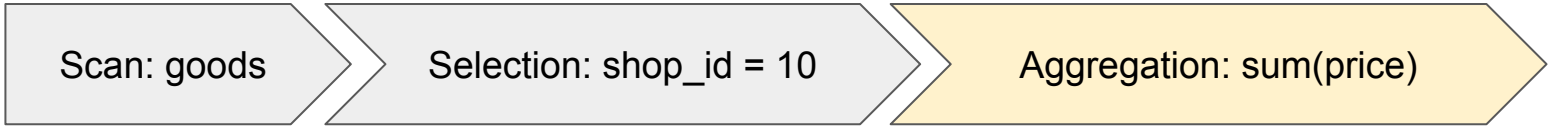


Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

next()

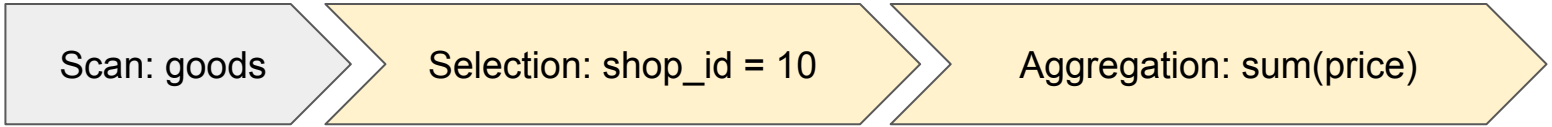


Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

next()

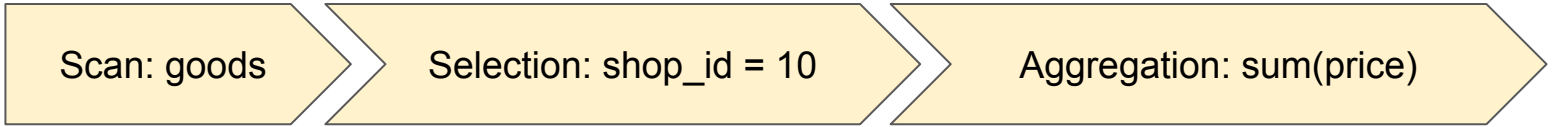


Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

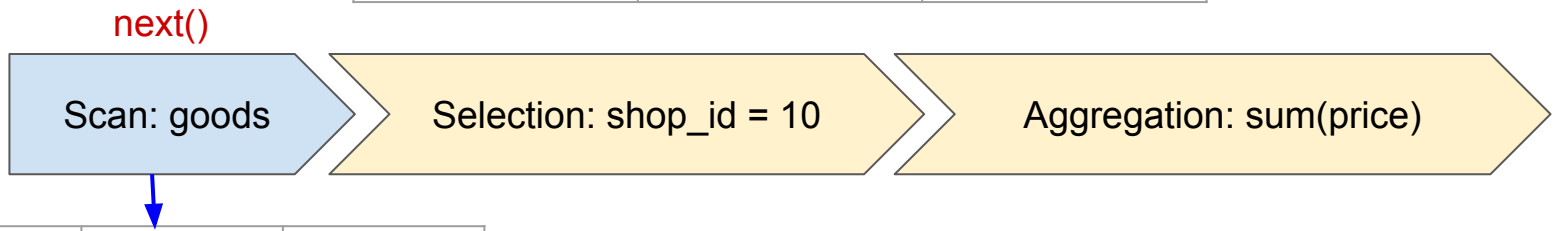
next()



Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

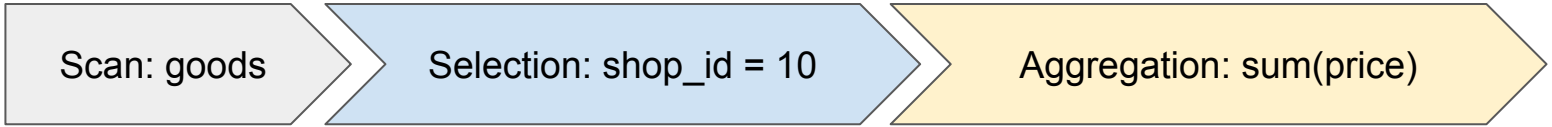


id	shop_id	price
1	5	10.5

Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

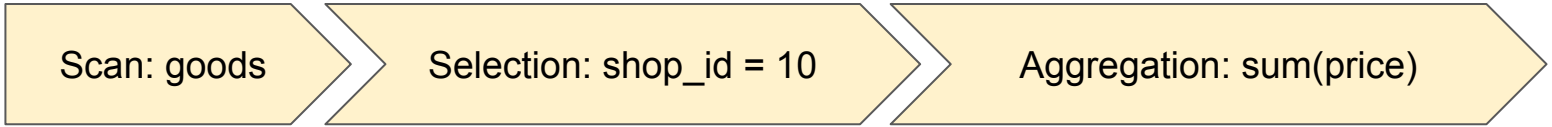


Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

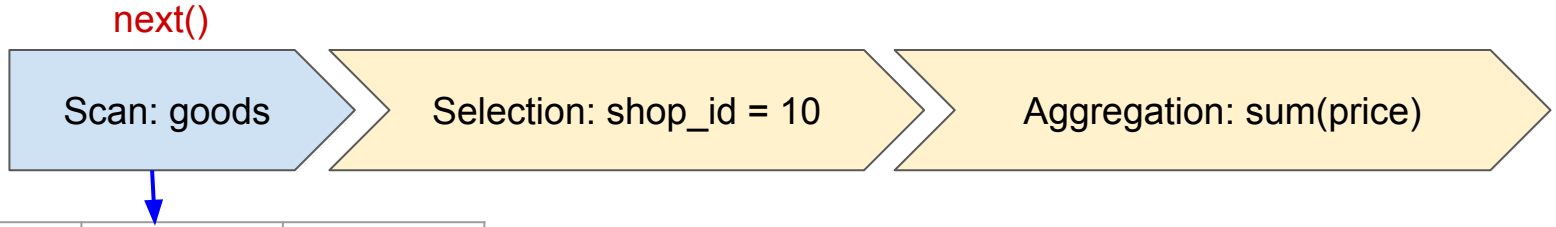
next()



Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7



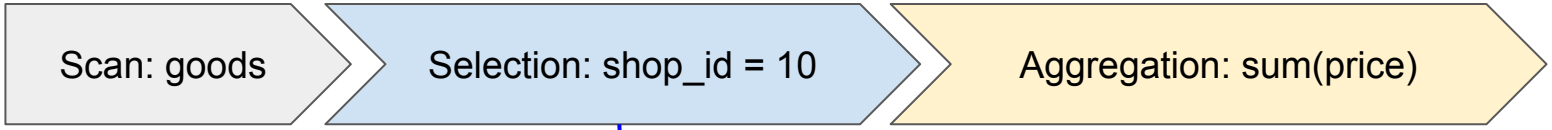
id	shop_id	price
2	10	1.2

Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

next()



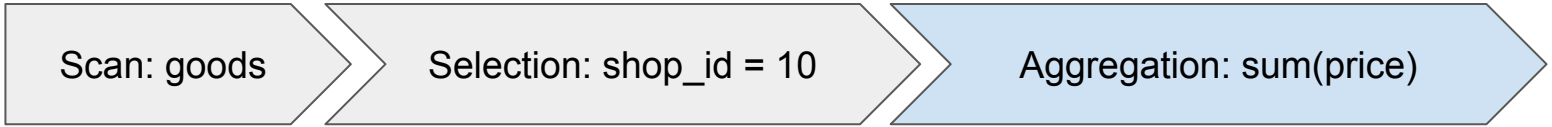
id	shop_id	price
2	10	1.2

PingCAP

Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

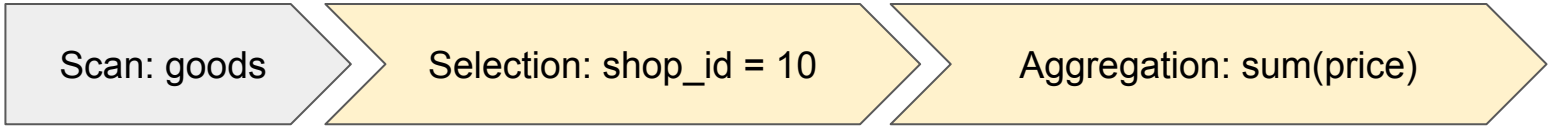


Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

next()

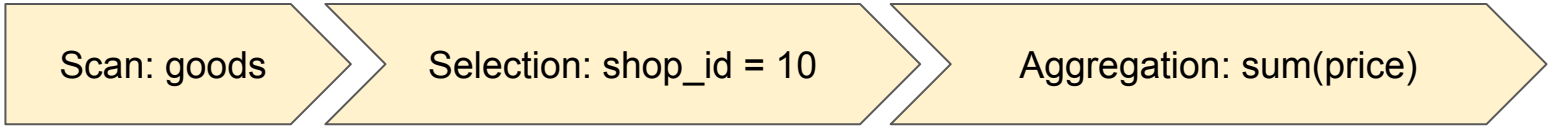


Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

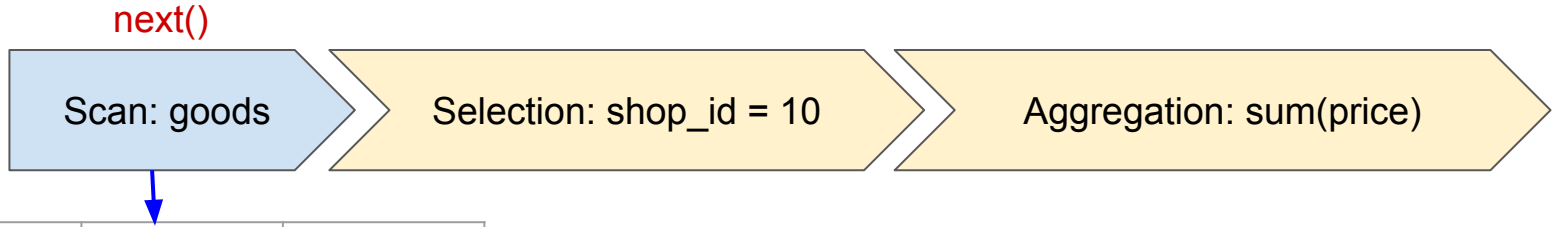
next()



Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7



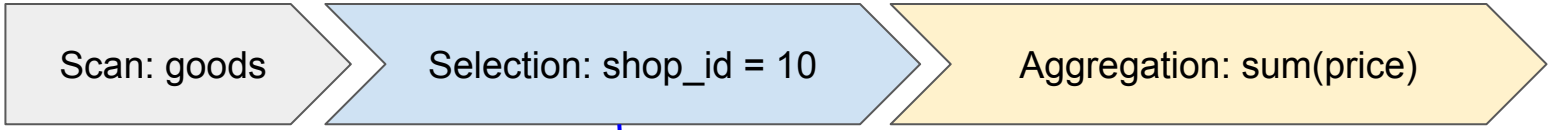
id	shop_id	price
3	10	13.7

Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

next()



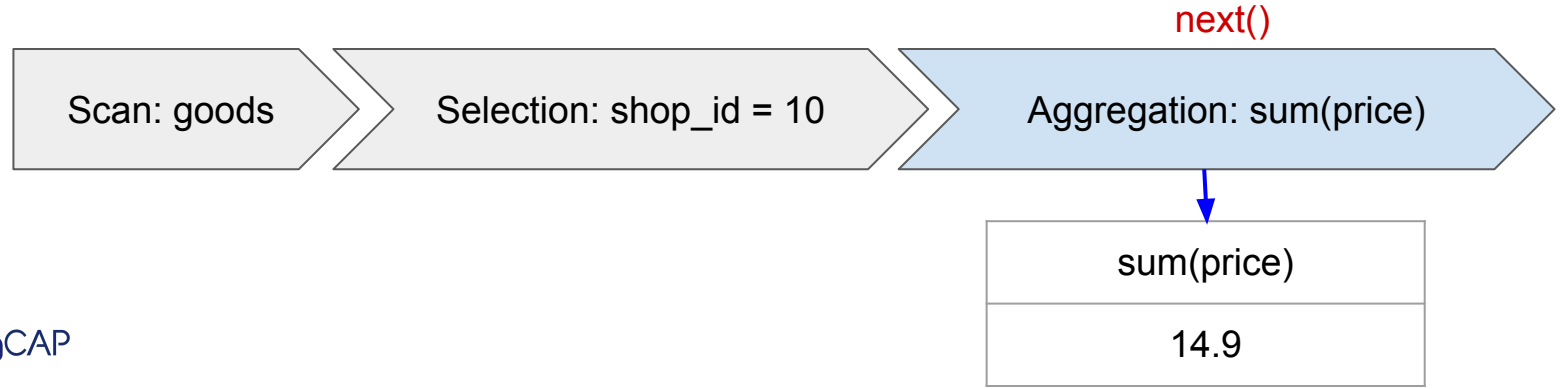
id	shop_id	price
3	10	13.7

PingCAP

Volcano Iterator Model

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

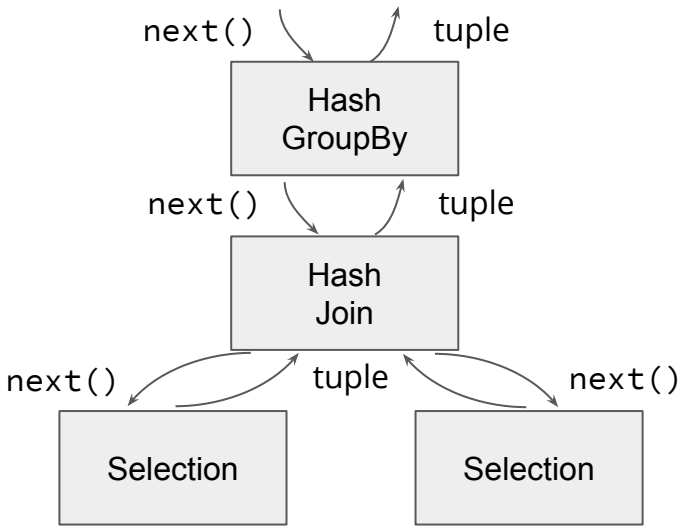
id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7



PingCAP

Volcano Iterator Model

- A.k.a Tuple at a time
- Elegant, flexible, extensible and powerful[2]
- Efficient at that time
  - disk I/O is the main overhead
- Low memory cost
- Very natural for row stores



PingCAP

[2] Graefe, Goetz. "Volcano/spl minus/an extensible and parallel query evaluation system." *IEEE Transactions on Knowledge and Data Engineering* 6.1 (1994): 120-135.

Vectorized executor

Row-based

We call it row

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

Vectorized

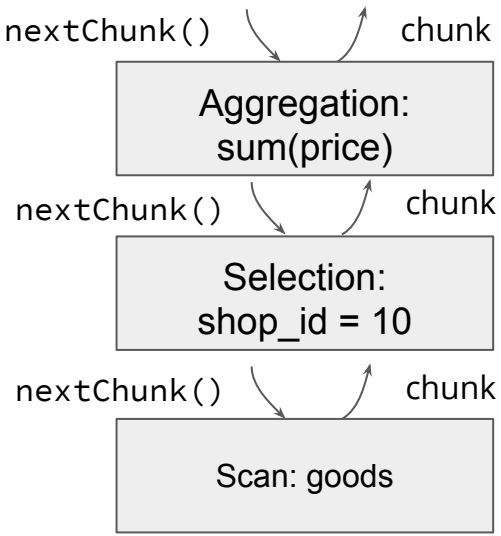
We call it column

We call it chunk

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
...	...	...
...	...	...
1024	10	15.3
1025		
...		
2048		

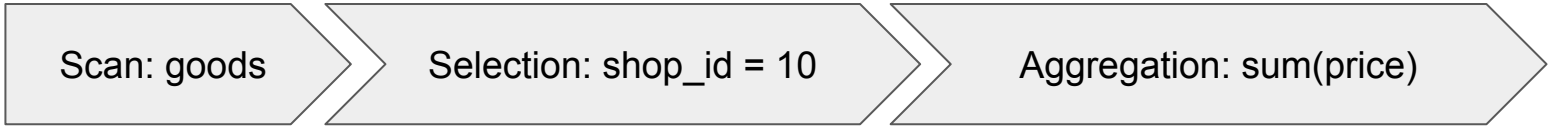
Vectorized executor

```
SELECT SUM(price)
FROM goods
WHERE shop_id = 10
```



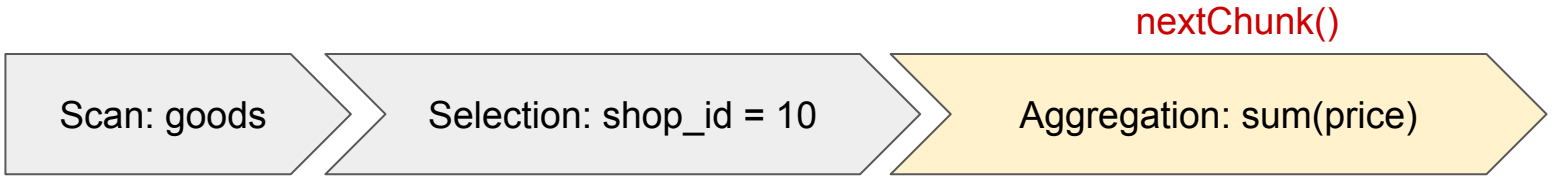
```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
...	...	...
...	...	...
1024	10	15.3



```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

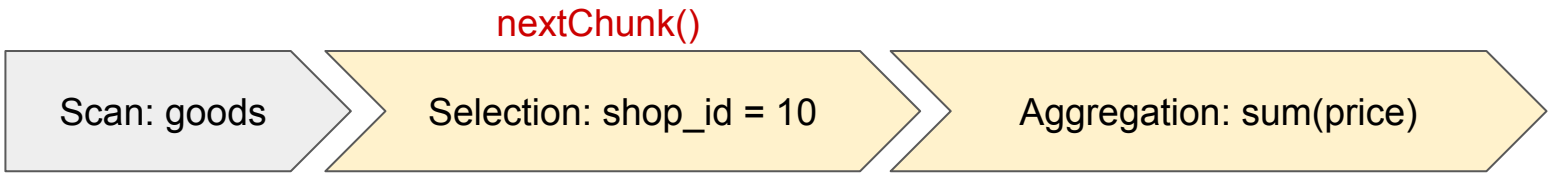
id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
...	...	...
...	...	...
1024	10	15.3



PingCAP

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

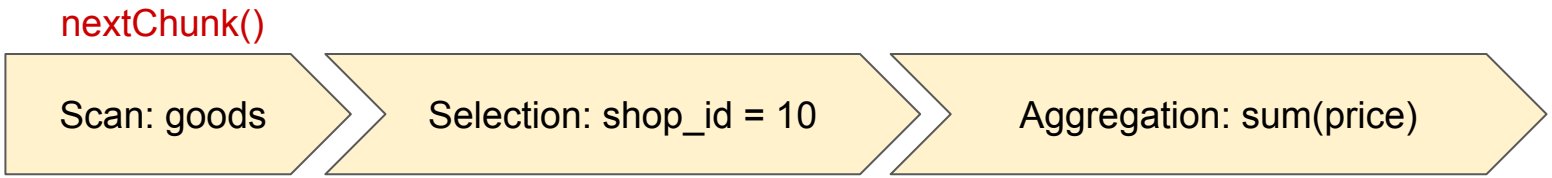
id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
...	...	...
...	...	...
1024	10	15.3



PingCAP

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

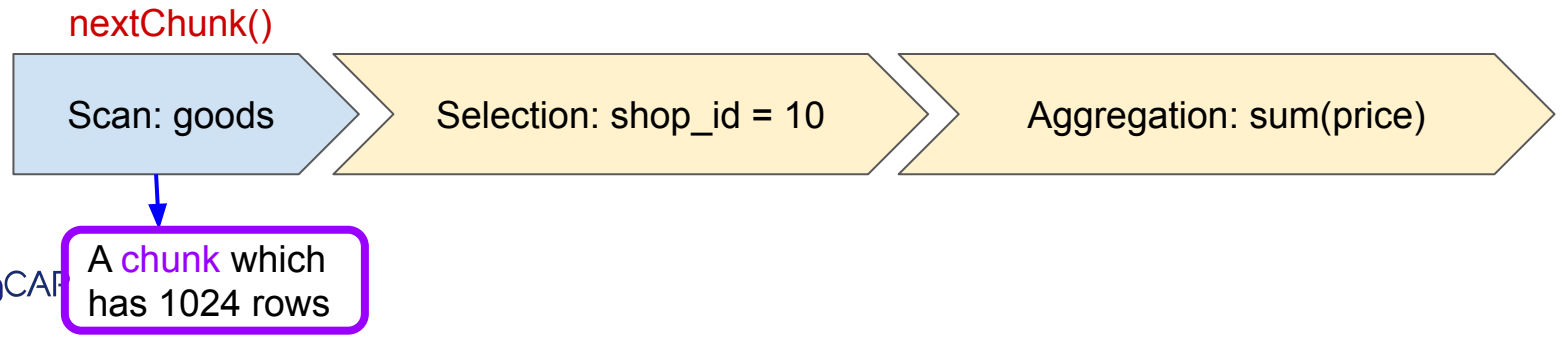
id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
...	...	...
...	...	...
1024	10	15.3



PingCAP

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

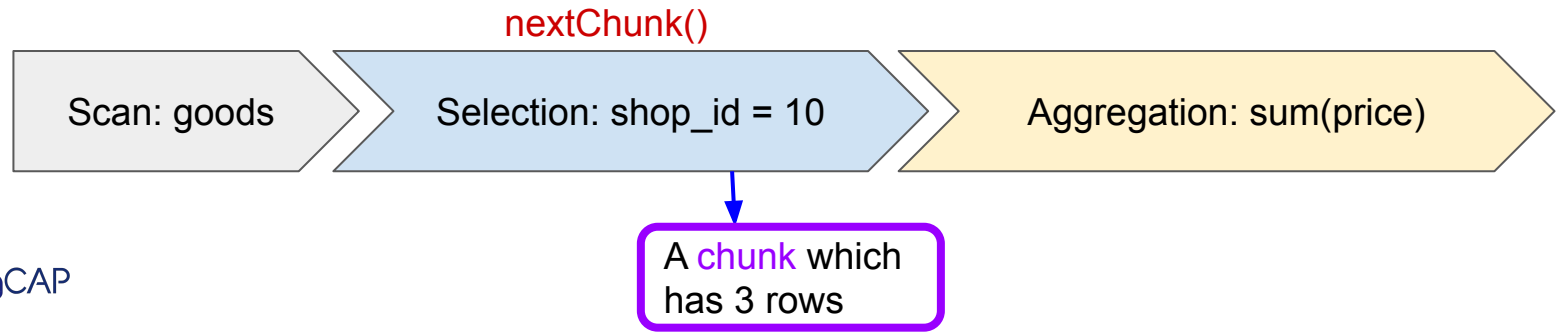
id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
...	...	...
...	...	...
1024	10	15.3



PingCAP

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

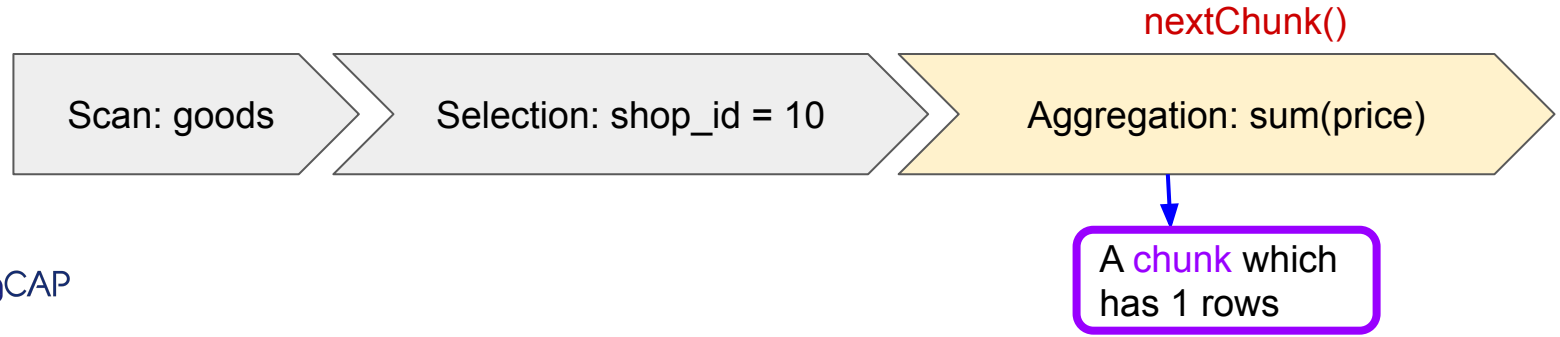
id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
...	...	...
...	...	...
1024	10	15.3



PingCAP

```
SELECT SUM(price) FROM goods WHERE shop_id = 10
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
...	...	...
...	...	...
1024	10	15.3

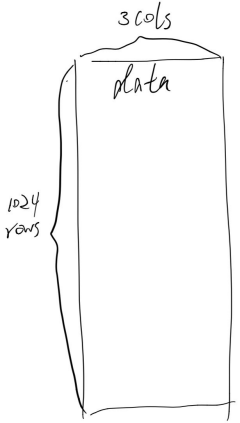


PingCAP

Why is it fast

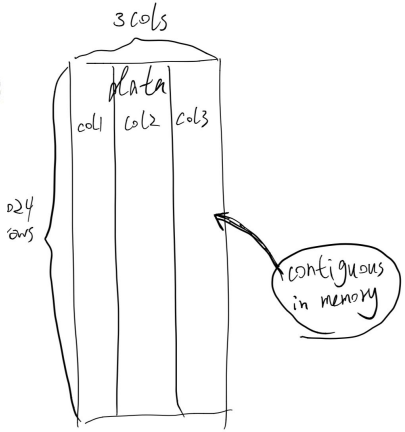
Advantages compared with traditional Vocalno-Model:

- lower interpretation overhead,
- higher cache hit-ratio.




```
bool sel_eq_row(r row) {
    return r[0].String() == "green" && r[1].Int() == 4;
}

vec<row> sel_eq_rows(vec<row> rows) {
    vec<row> res;
    for (r : rows) {
        if sel_eq_row(r) {
            res.append(r);
        }
    }
    return res;
}
```



Why is it fast

The instruction throughput of a CPU depending on:

- the amount of independent instructions the CPU can detect,
- the number of branches can be predicated,
- the cache hit-ratio of the memory loads and stores.

out-of-order execution + instruction pipelining

```
F(A[0]),G(A[0]), F(A[1]),G(A[1]),..  F(A[n]),G(A[n])
```

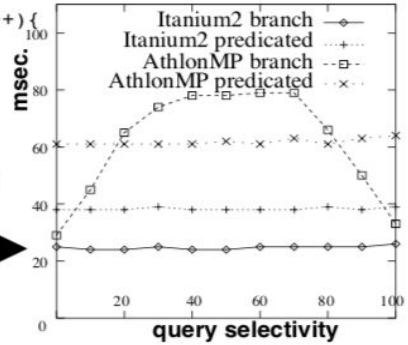
into:

```
F(A[0]),F(A[1]),F(A[2]), G(A[0]),G(A[1]),G(A[2]), F(A[3]),..
```

Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

branch predication

```
int sel_lt_int_col_int_val(int n, int* res, int* in, int V) {
  for(int i=0,j=0; i<n; i++){
    /* branch version */
    if (src[i] < V)
      out[j++] = i;
    /* predicated version */
    bool b = (src[i] < V);
    out[j] = i;
    j += b;
  }
  return j;
}
```



References:

- MonetDB/X100: Hyper-Pipelining Query Execution
- Everything You Always Wanted to Know About Compiled and Vectorized Queries But Were Afraid to Ask

PingCAP

Part II - Recent work

Vectorized Hash Join

- Build
- Probe
- Filter
- Materialize

Vectorized Hash Join

```
SELECT price, custom
FROM goods, deal
WHERE
    goods.shop_id = deal.shop_id
and goods.id = deal.goods_id
```

Table: goods

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

Table: deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Hash Join - Build

We call it
row

Table: deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Hash Join - Build

HashFunc([1, 5]) -> hash

Table: deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Hash Join - Build

HashFunc([1, 5]) -> hash

Table: deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Hash Join - Build

HashFunc([3, 10]) -> hash

Table: deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Vectorized Hash Join - Build

We name it a column

Table Deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Vectorized Hash Join - Build

```
var h []hash.Hash
h[0].Write(1)
h[1].Write(1)
h[2].Write(3)
```

Table: deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Vectorized Hash Join - Build

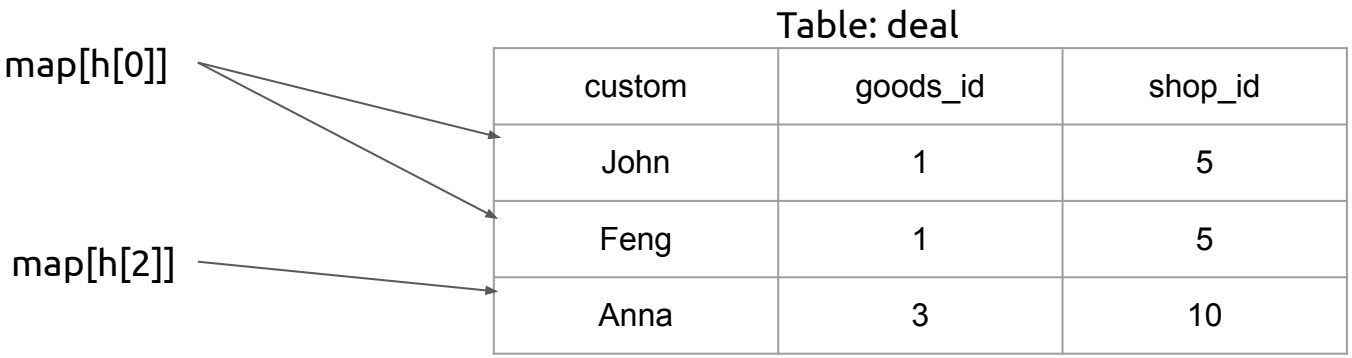
```
var h []hash.Hash
h[0].Write(5)
h[1].Write(5)
h[2].Write(10)
```

Table: deal

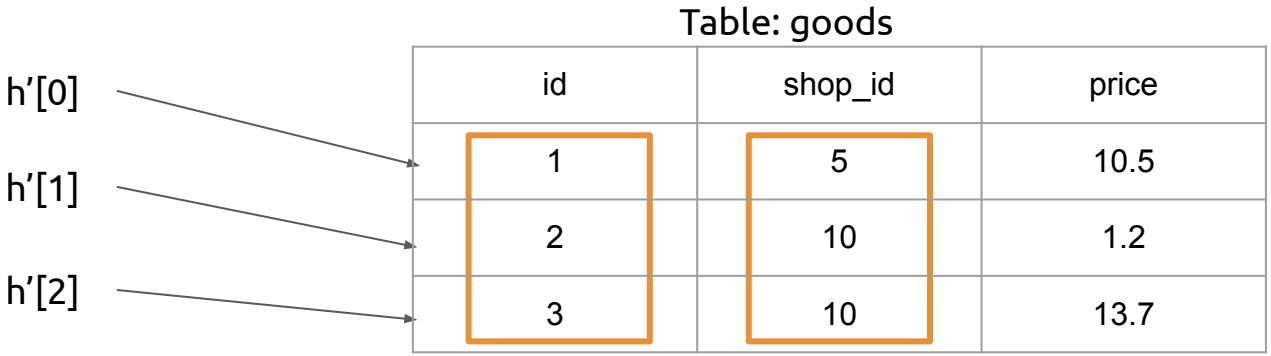
custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Vectorized Hash Join - Build

We’ve got a hash map!



Vectorized Hash Join - Probe



Vectorized Hash Join - Probe

Table: goods

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

map[h[0]]

map[h[2]]

Table: deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

Vectorized Hash Join - Materialize

help wanted

Table: goods

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7

We call it chunk

Table: deal

custom	goods_id	shop_id
John	1	5
Feng	1	5
Anna	3	10

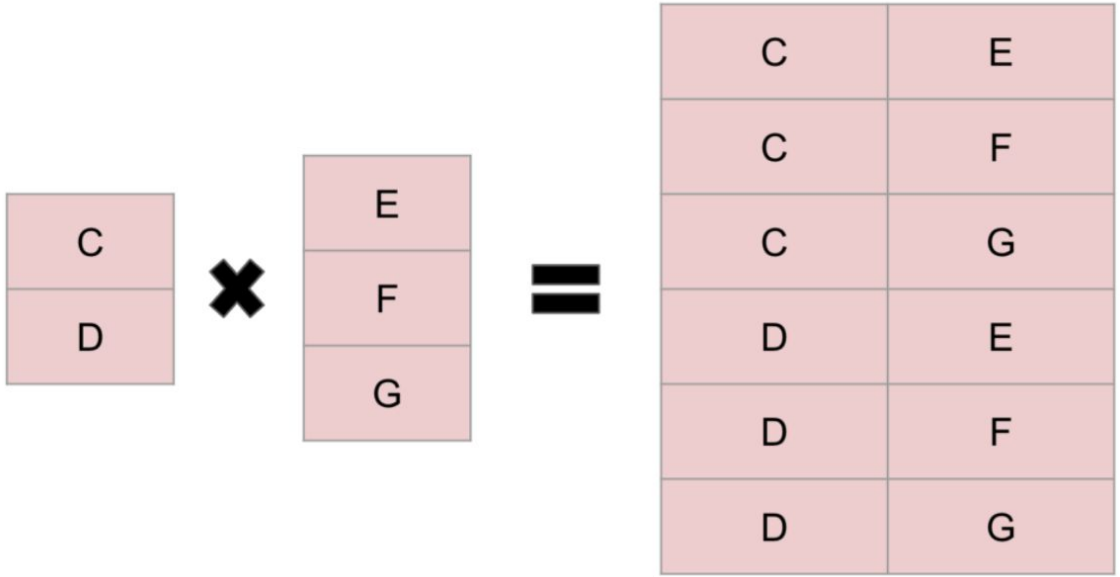
map[h[0]]

map[h[2]]

Vectorized Hash Join - Materialize

help wanted

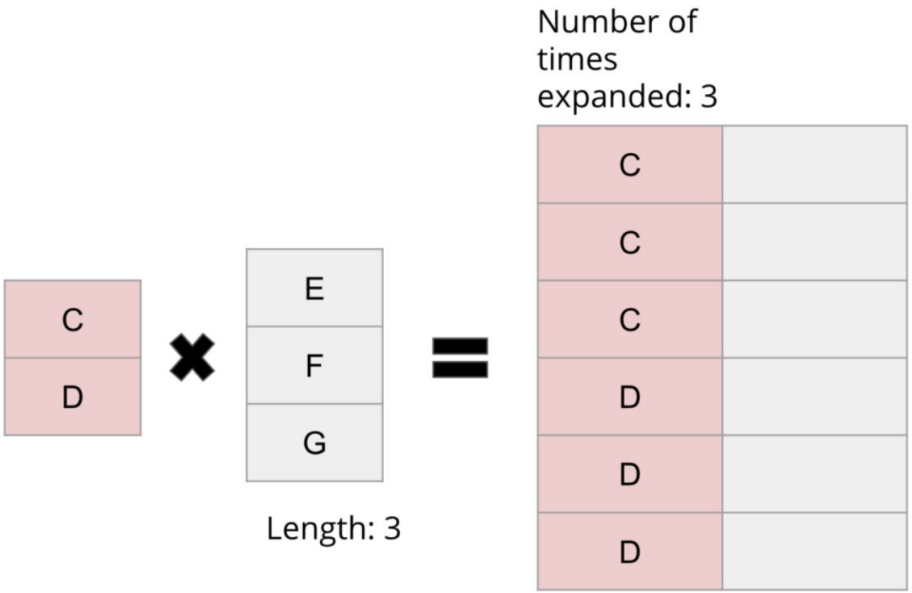
The vectorized matrializing phase



Vectorized Hash Join - Materialize

help wanted

The vectorized matrializing phase

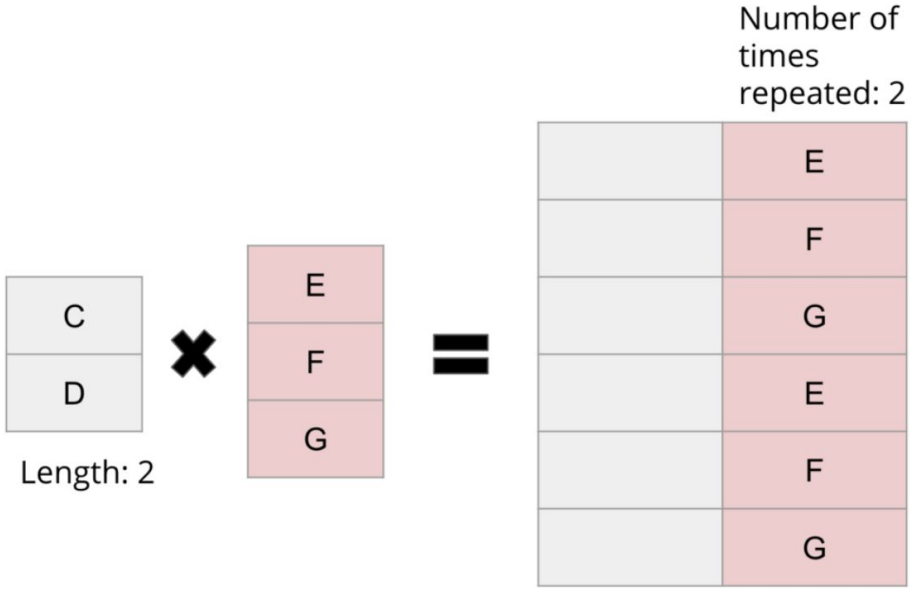


Left cross product of join

Vectorized Hash Join - Materialize

help wanted

The vectorized matrializing phase



Right cross product of join

Vectorized Hash Join - Materialize

help wanted

- Vectorized materializing has not implemented currently
- help wanted

Vectorized Stream Aggregation

```
SELECT SUM(price) FROM goods WHERE shop_id = 10 GROUP BY shop_id
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
4	10	13.0
5	11	1.5
6	11	1.0
7	11	3.7
8	11	3.9

Vectorized Stream Aggregation

```
SELECT SUM(price) FROM goods GROUP BY shop_id
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
4	10	13.0
5	11	1.5
6	11	1.0
7	11	3.7
8	11	3.9

Vectorized Stream Aggregation

```
SELECT SUM(price) FROM goods GROUP BY shop_id
```

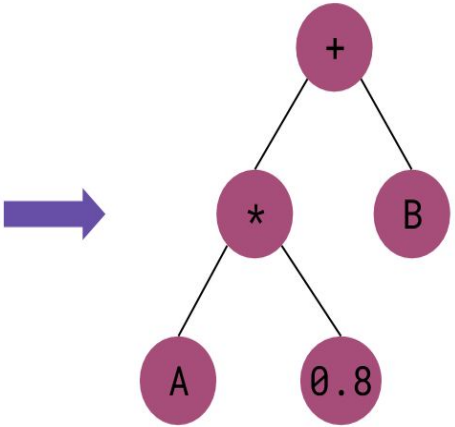
help wanted

Use binary search to find bondary

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
4	10	13.0
5	11	1.5
6	11	1.0
7	11	3.7
8	11	3.9

Vectorized function evaluation

colA * 0.8 + colB

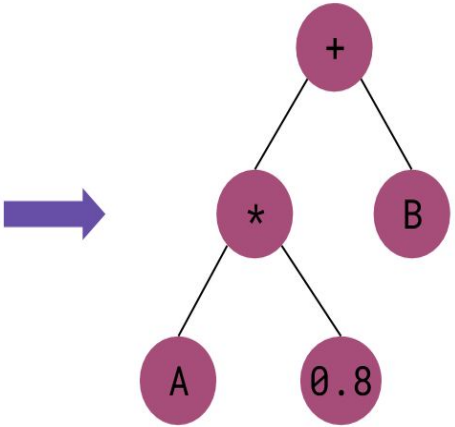


colA
1.0
3.1
4.0
-1.5
5.0
9.9

colB
3.0
77.1
14.1
1.1
-39.4
-5.3

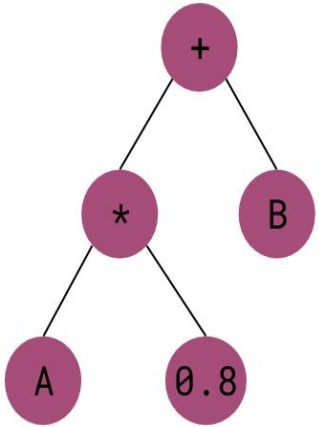
Vectorized function evaluation

colA * 0.8 + colB



colA		colB	
1.0	✕ 0.8 +	3.0	=
3.1		77.1	
4.0		14.1	
-1.5		1.1	
5.0		-39.4	
9.9		-5.3	

Vectorized function evaluation

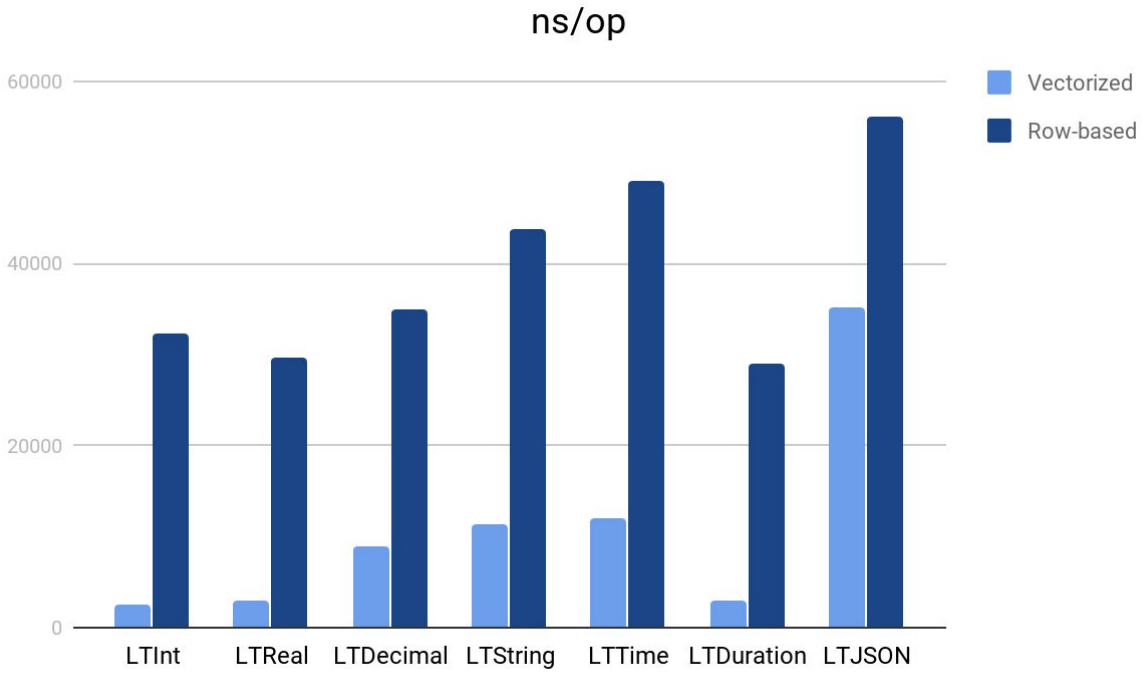


colA	Const	❌	colB	➕
1.0	0.8	0.8	3.0	3.8
3.1	0.8	2.48	77.1	79.58
4.0	0.8	3.2	14.1	17.3
-1.5	0.8	1.2	1.1	2.3
5.0	0.8	4.0	-39.4	-35.4
9.9	0.8	8.91	-5.3	3.61

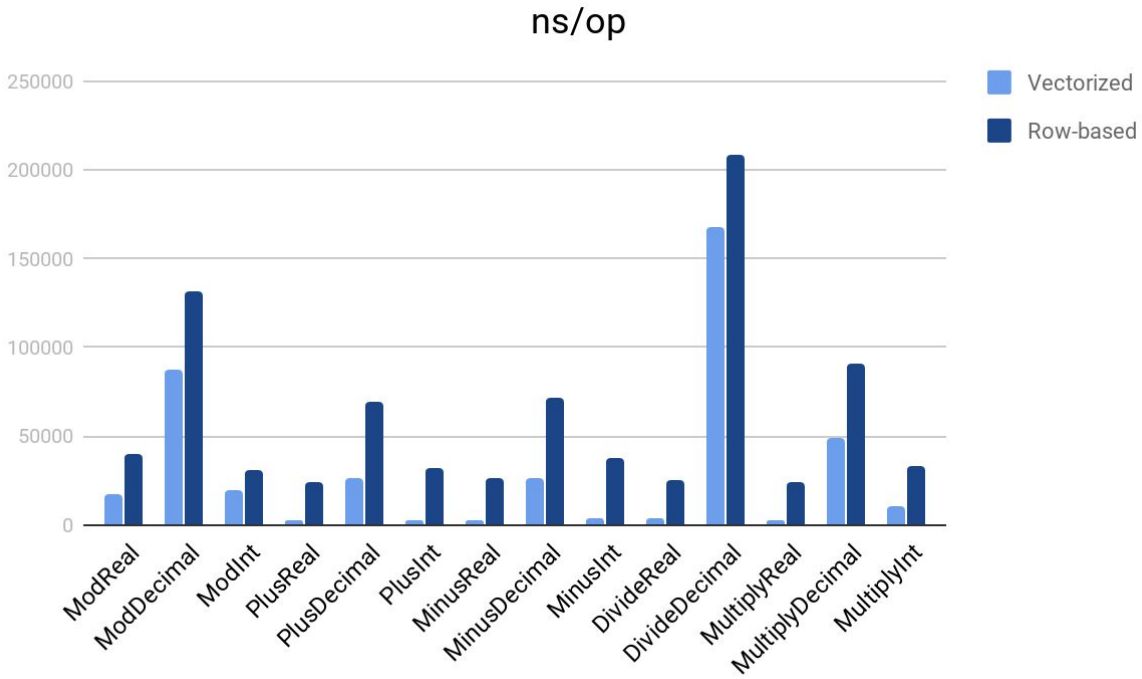
Community’s help

- 360 functions in just two months
- 256 Pull Requests filed by community
- 60+ unique contributor involved
- 9 Active Contributors
  - (who merged at least eight PRs in one year)
- 2 Reviewers
  - (who merge at least 30 PRs in one year).

Vectorized function evaluation



Vectorized function evaluation



PingCAP

Part III - Furture work

Vectorized Aggregation Functions

help wanted

```
SELECT SUM(price) FROM goods
```

id	shop_id	price
1	5	10.5
2	10	1.2
3	10	13.7
4	10	13.0
5	11	1.5
6	11	1.0
7	11	3.7
8	11	3.9

Vectorized Merge Join

help wanted

SELECT t1.k, t1.v, t2.v FROM t1 INNER MERGE JOIN t2 ON t1.k = t2.k

t1

k	v
1	A
2	C
2	D
4	H
5	J
6	L

t2

k	v
1	B
2	E
2	F
2	G
3	I
5	K

Output

1	A	B
2	C	E
2	C	F
2	C	G
2	D	E
2	D	F
2	D	G
5	J	K

Vectorized Merge Join

help wanted

The traditional merge join algorithm

t1

k	v
1	A
2	C
2	D
4	H
5	J
6	L

t2

k	v
1	B
2	E
2	F
2	G
3	I
5	K

Output

1	A	B

Vectorized Merge Join

help wanted

The traditional merge join algorithm

t1

k	v
1	A
2	C
2	D
4	H
5	J
6	L

t2

k	v
1	B
2	E
2	F
2	G
3	I
5	K

Output

1	A	B
2	C	E
2	C	F
2	C	G
2	D	E
2	D	F
2	D	G

Vectorized Merge Join

help wanted

The traditional merge join algorithm

t1

k	v
1	A
2	C
2	D
4	H
5	J
6	L

t2

k	v
1	B
2	E
2	F
2	G
3	I
5	K

Output

1	A	B
2	C	E
2	C	F
2	C	G
2	D	E
2	D	F
2	D	G
5	J	K

Vectorized Merge Join

help wanted

The vectorized probing phase

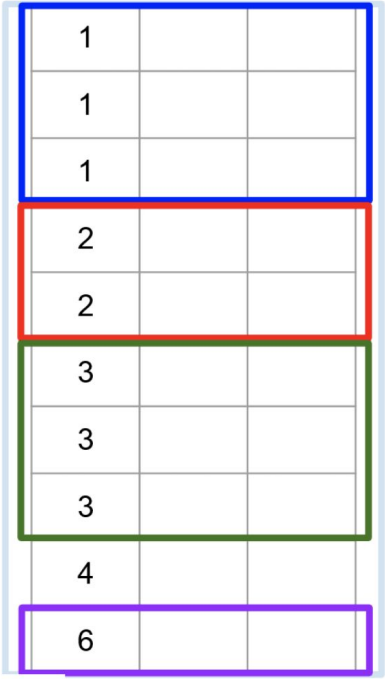
1	2	
1	4	
1	4	
2	3	
2	4	
3	5	
3	10	
3	11	
4	6	
6	7	

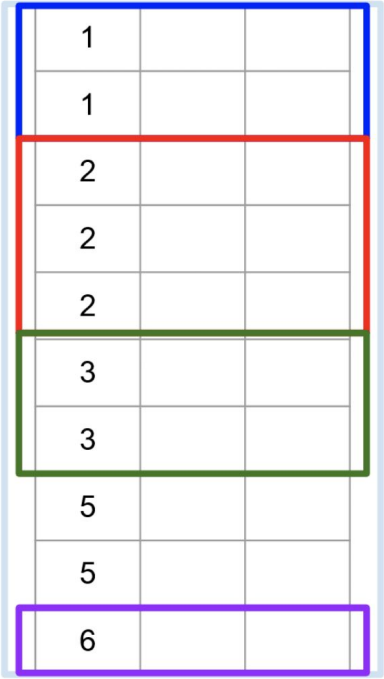
1	2	
1	3	
2	3	
2	5	
2	9	
3	6	
3	7	
5	6	
5	7	
6	7	

Vectorized Merge Join

help wanted

The vectorized probing phase

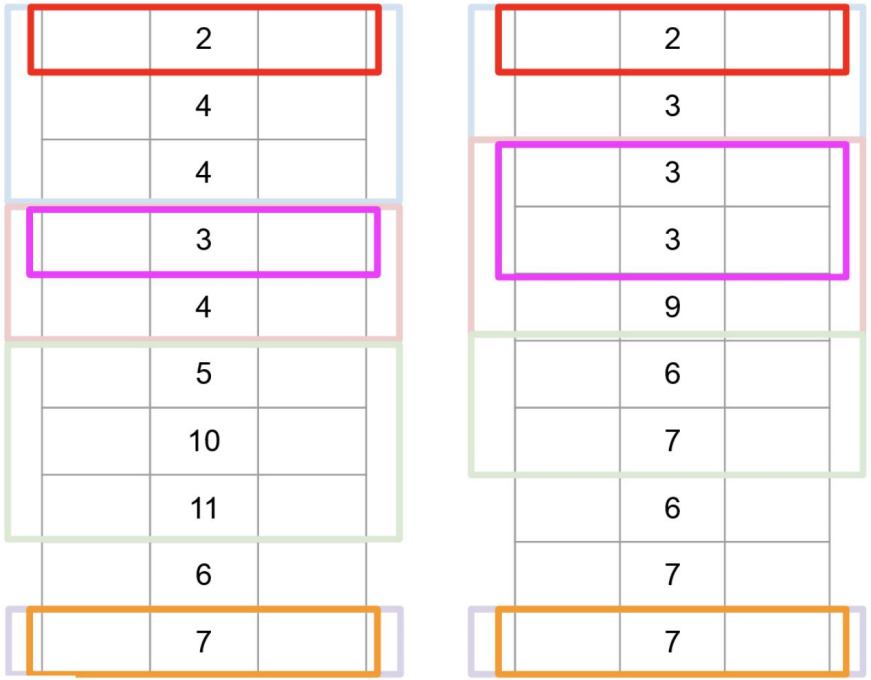




Vectorized Merge Join

help wanted

The vectorized probing phase



Vectorized Merge Join

help wanted

The vectorized probing phase

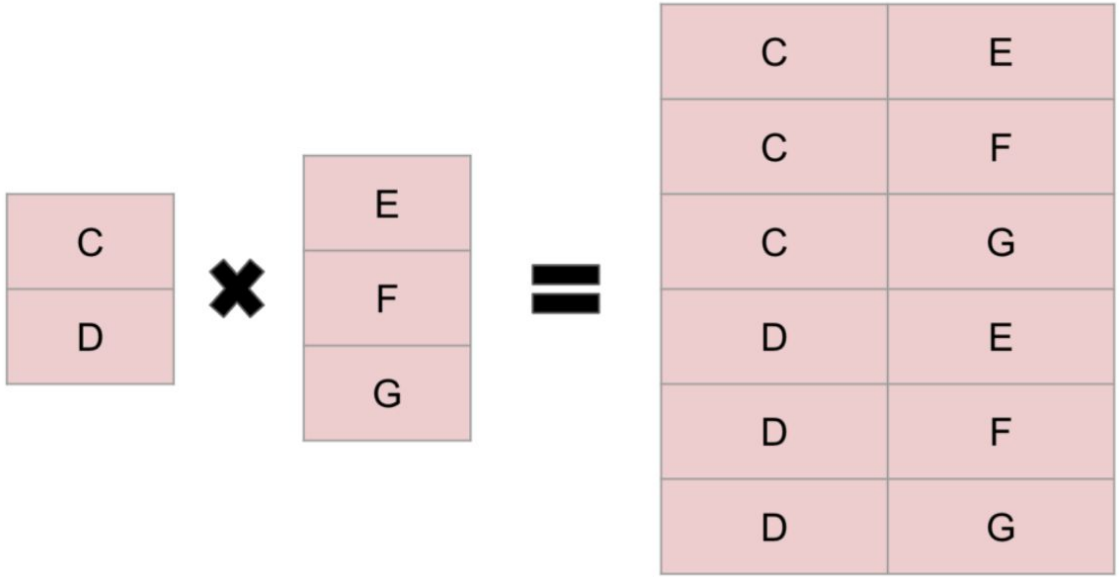
1	2	
	4	
	4	
2	3	
	4	
	5	
	10	
	11	
	6	
6	7	

1	2	
	3	
2	3	
2	3	
	9	
	6	
	7	
	6	
	7	
6	7	

Vectorized Merge Join

help wanted

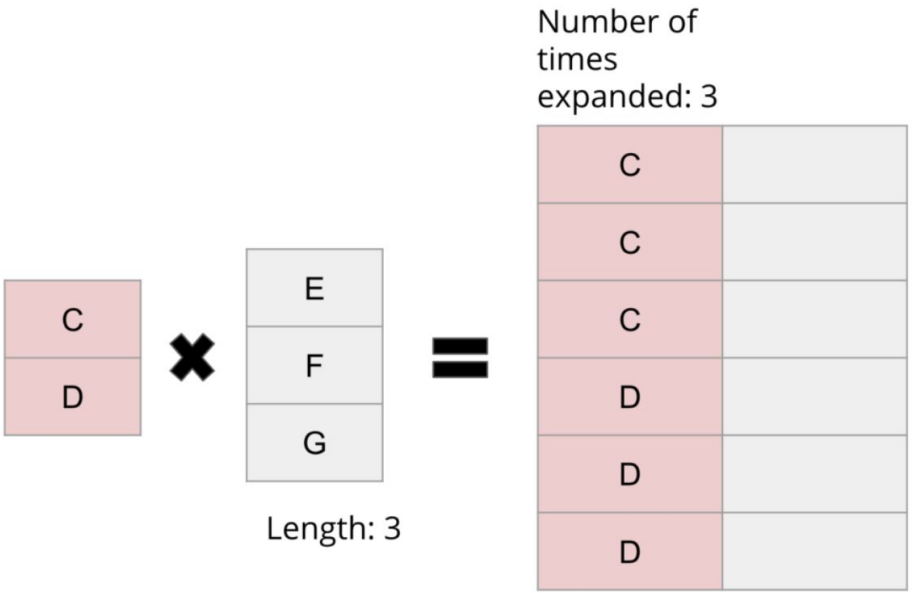
The vectorized matrializing phase



Vectorized Merge Join

help wanted

The vectorized matrializing phase

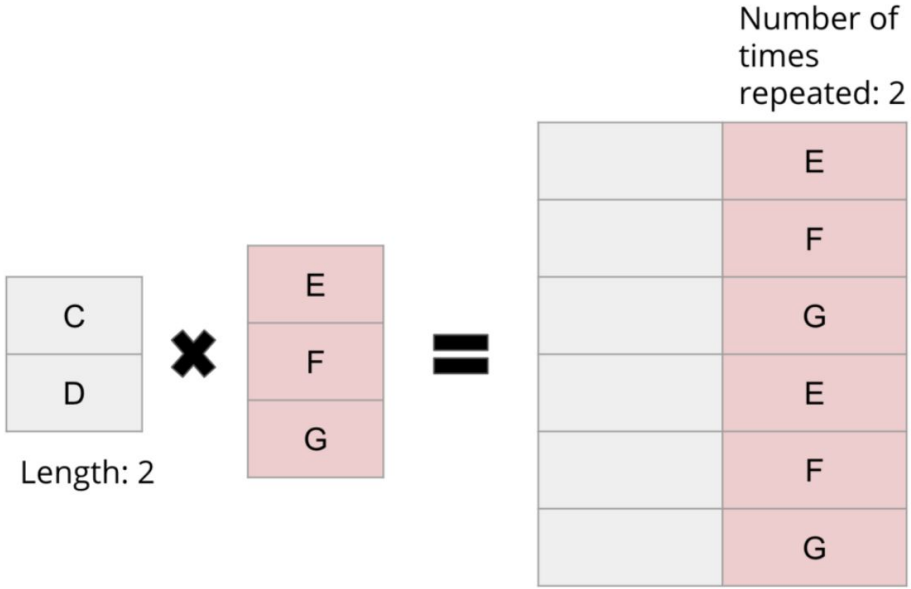


Left cross product of join

Vectorized Merge Join

help wanted

The vectorized matrializing phase

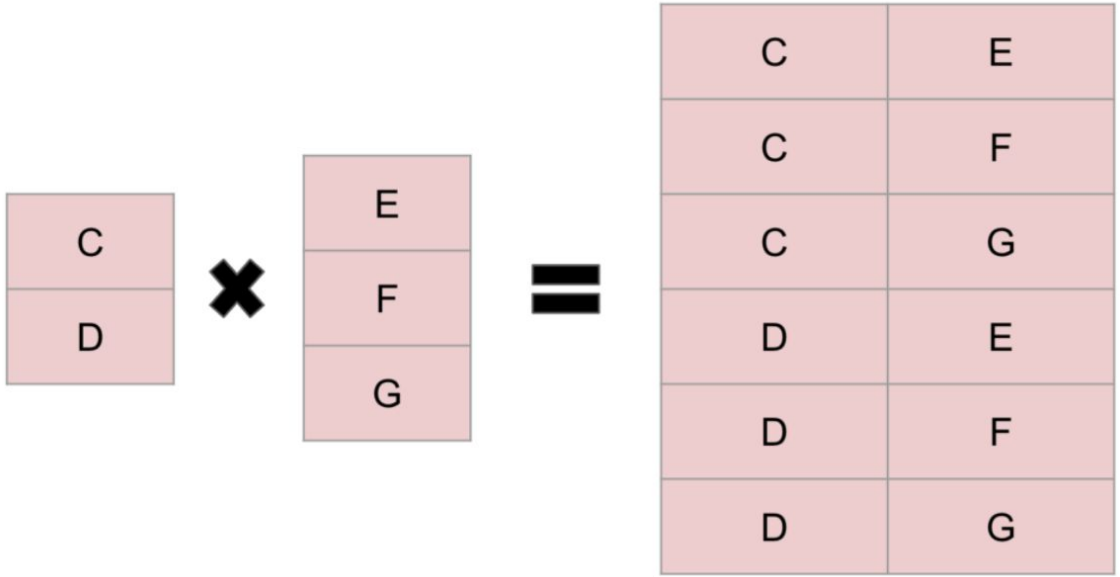


Right cross product of join

Vectorized Merge Join

help wanted

The final result



PingCAP

Thank You !