Altinity WEBINAR

INTRODUCTION TO HIGH VELOCITY ANALYTICS

USING CLICKHOUSE ARRAYS

Altinity Engineering

Presenter Bios and Altinity Introduction

Robert Hodges - Altinity CEO

30+ years on DBMS plus virtualization and security. ClickHouse is DBMS #20

Alexander Zaitsev - Altinity CTO

Altinity founder with decades of expertise on petabyte-scale analytic systems

The #1 enterprise ClickHouse provider. Now offering Altinity.Cloud

Major committer and community sponsor for ClickHouse in US/EU

Key features of ClickHouse

Single C++ binary

SQL language

Shared nothing architecture

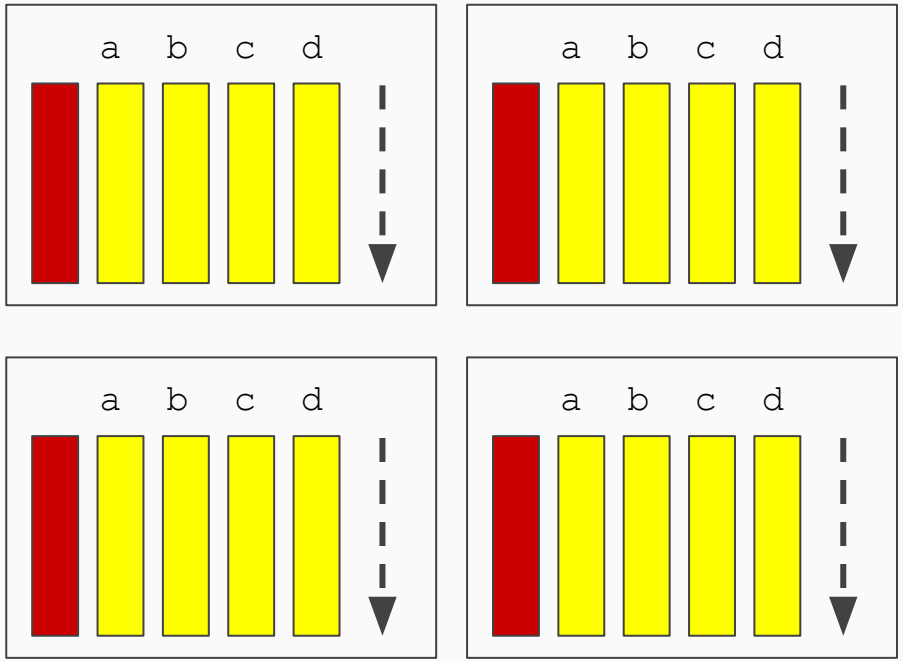
Column storage

Vectorized query execution

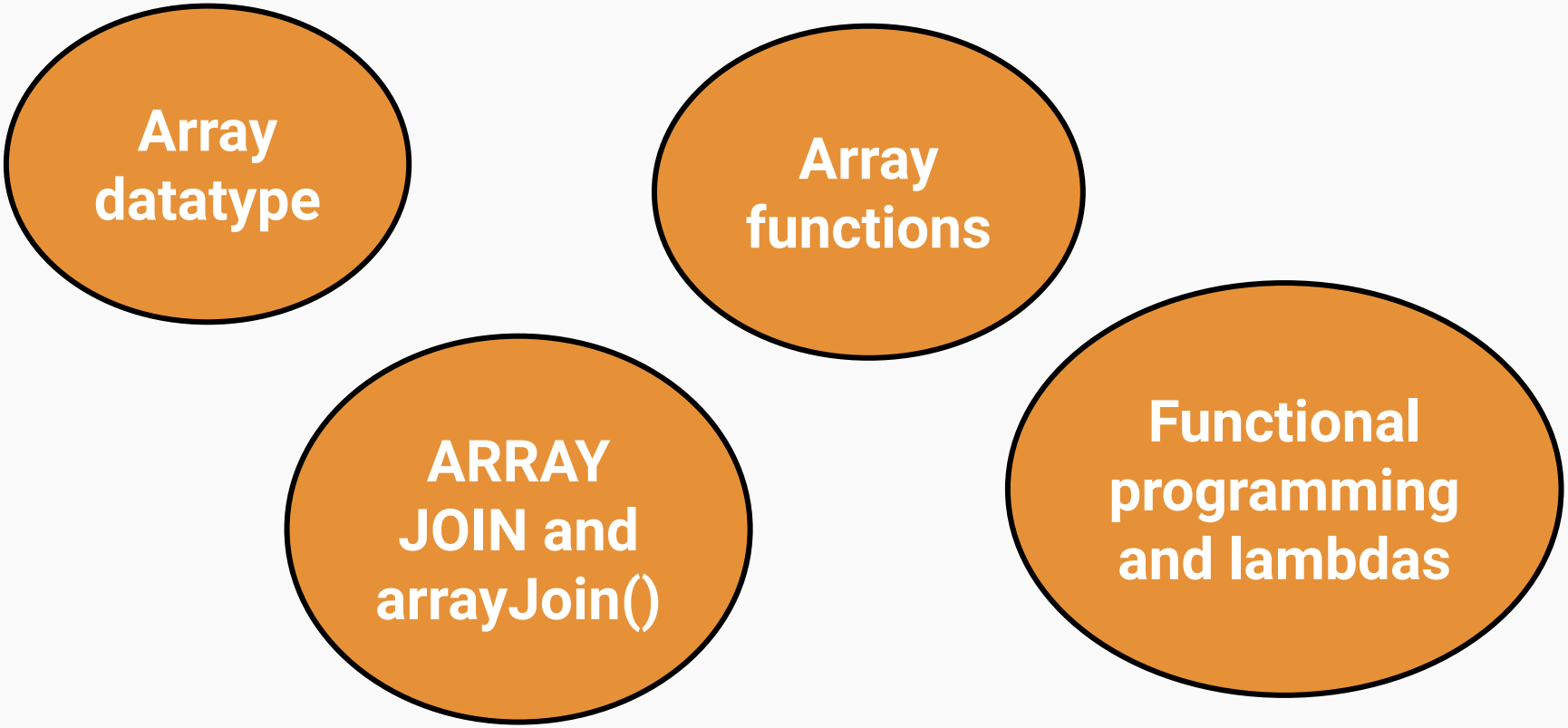
MPP-enabled (shards/replicas)

Apache 2.0 license

Really fast



Arrays are a ClickHouse SQL extension



Modeling structured data using arrays

Use case: Monitoring VMs with differing attributes

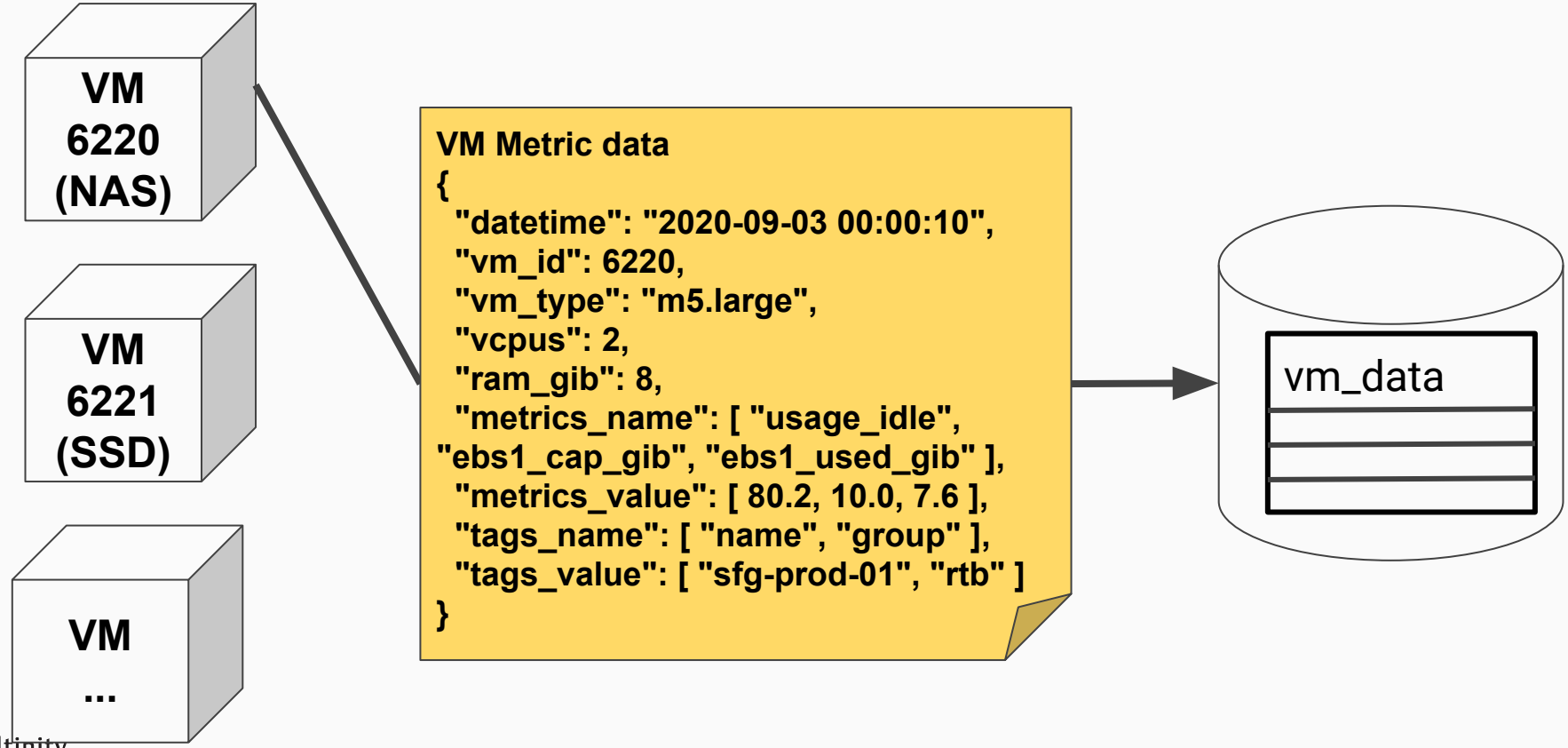
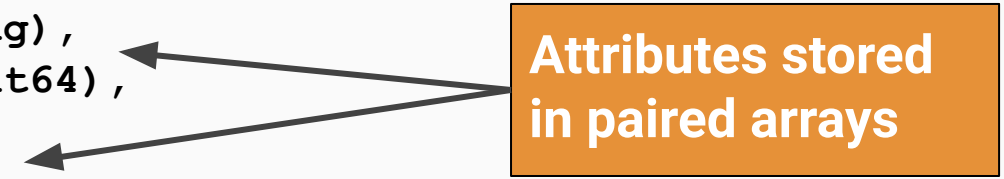


Table definition for VM data

```
CREATE TABLE vm_data (
  datetime DateTime,
  date Date default toDate(datetime),
  vm_id UInt32,
  vm_type LowCardinality(String),
  vcpus UInt16,
  ram_gib Float32,
  metrics_name Array(String),
  metrics_value Array(Float64),
  tags_name Array(String),
  tags_value Array(String)
)
ENGINE = MergeTree()
PARTITION BY date
ORDER BY (vm_type, vm_id, datetime)
```



Arrays work similar to Python, etc.

```
WITH [1, 2, 4] AS array
SELECT
    array[1] AS First,
    array[2] AS Second,
    array[3] AS Third,
    array[-1] AS Last,
    length(array) AS Length
```

OK, this is SQL so we have to start with 1

```
┌─First─┬─Second─┬─Third─┬─Last─┬─Length─┐
│     1 │      2 │     4 │    4 │      3 │
└───────┴────────┴───────┴──────┴────────┘
```

ARRAY JOIN “unrolls” arrays to rows

```
SELECT date, vm_id, vm_type, name, value
FROM vm_data
ARRAY JOIN tags_name AS name, tags_value AS value
ORDER BY vm_id, name
```

date	vm_id	vm_type	name	value
2020-09-03	6220	m5.large	group	rtb
2020-09-03	6220	m5.large	name	sfg-prod-01
2020-09-03	6221	m5ad.xlarge	group	marketing
2020-09-03	6221	m5ad.xlarge	name	mt-prod-65
2020-09-03	6221	m5ad.xlarge	owner	casey

Put this in a subquery to answer questions using normal SQL

ClickHouse has other ways to join arrays

```
SELECT 1, 2,
  arrayJoin(['a', 'b']) a1
```

```
┌─1─┬─2─┬─a1─┐
│ 1 │ 2 │ a  │
│ 1 │ 2 │ b  │
└───┴───┴────┘
```

```
SELECT 1, 2,
  arrayJoin(['a', 'b']) a1,
  arrayJoin(['i', 'ii']) a2
```

```
┌─1─┬─2─┬─a1─┬─a2─┐
│ 1 │ 2 │ a  │ i  │
│ 1 │ 2 │ a  │ ii │
│ 1 │ 2 │ b  │ i  │
│ 1 │ 2 │ b  │ ii │
└───┴───┴────┴────┘
```

Cartesian join between arrays!

Fun with groupArray and sortArray

Reporting on airline flights

Q: What is the highest number of hops per day for a single aircraft using the same flight number? What does the itinerary look like?

Find the maximum hops on a single day

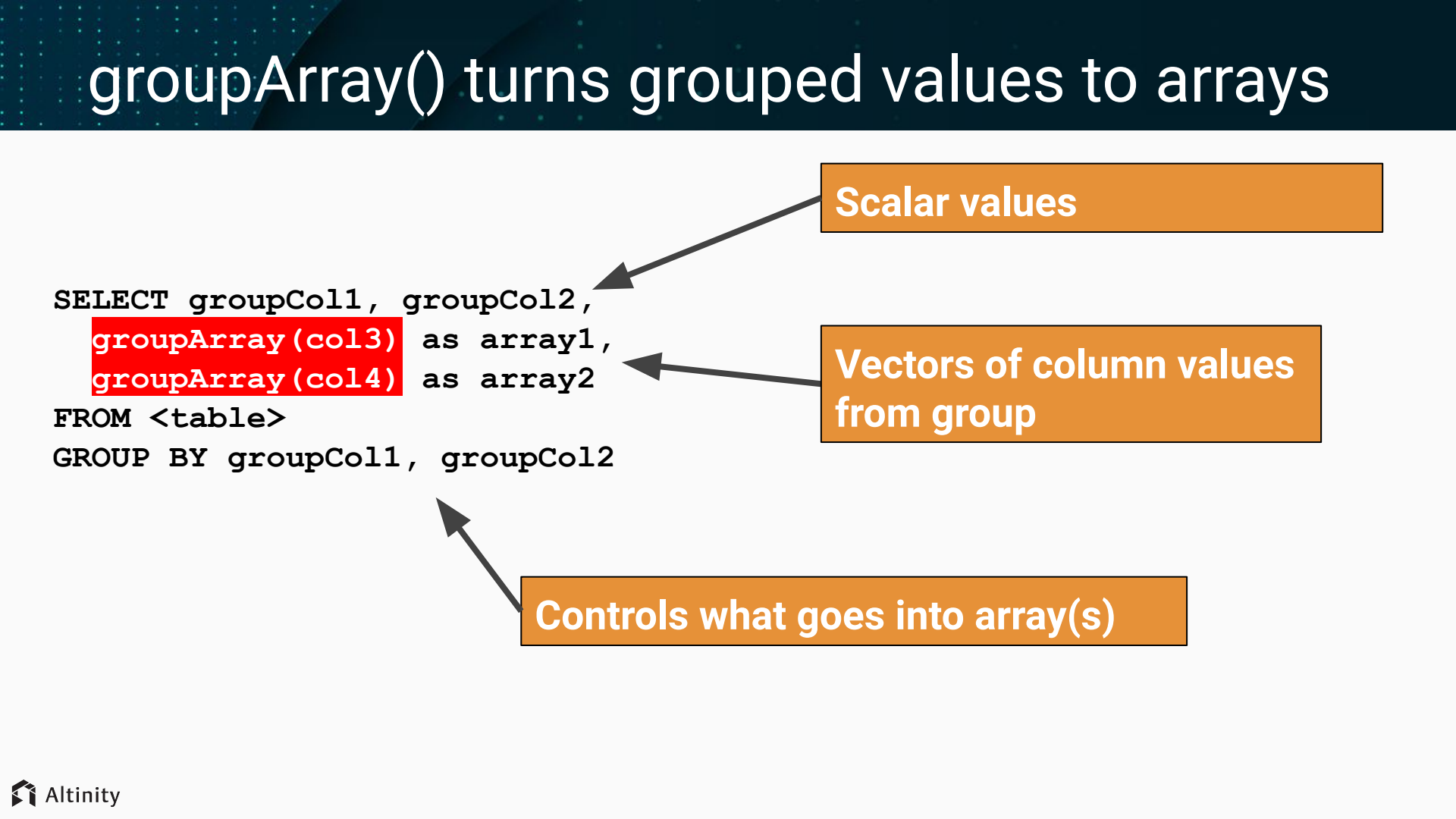
```
SELECT Carrier, TailNum, FlightNum, count(*) AS Hops
FROM ontime
WHERE (FlightDate = toDate('2017-01-15')) AND (DepTime < ArrTime)
GROUP BY Carrier, TailNum, FlightNum
ORDER BY Hops DESC, Carrier, TailNum, FlightNum
LIMIT 5
```

Carrier	TailNum	FlightNum	Hops
WN	N963WN	5663	7
WN	N223WN	4315	6
WN	N438WN	4415	6
WN	N445WN	3462	6
WN	N451WN	4362	6

Look Ma! No arrays!

groupArray() turns grouped values to arrays

```
SELECT groupCol1, groupCol2,
   groupArray(col3) as array1,
   groupArray(col4) as array2
FROM <table>
GROUP BY groupCol1, groupCol2
```



Let's build up a list of stops

```
SELECT Carrier, TailNum, FlightNum,
  groupArray(DepTime) as Departures,
  length(groupArray(DepTime)) as Hops
FROM ontime
WHERE (FlightDate = toDate('2017-01-15')) AND (DepTime < ArrTime)
GROUP BY Carrier, TailNum, FlightNum
ORDER BY Hops DESC, Carrier, TailNum, FlightNum
LIMIT 5
```

All values in group

How many values

Not sorted!

Carrier	TailNum	FlightNum	Departures	Hops
WN	N963WN	5663	[1224,1923,1003,745,1616,2140,1439]	7
WN	N223WN	4315	[1529,2105,1237,1041,1920,1722]	6
WN	N438WN	4415	[1922,1748,629,2127,1200,1436]	6
WN	N445WN	3462	[1512,1334,1635,1008,651,1827]	6
WN	N451WN	4362	[843,1724,1240,1058,1945,600]	6

Introducing the arraySort function

```
SELECT arraySort([1224, 1923, 1003, 745]) AS s

┌─s──────────────────────┐
│ [745,1003,1224,1923]   │
└────────────────────────┘

SELECT arraySort((x) -> -x,
         [1224, 1923, 1003, 745]) AS s
-- (OR)
SELECT arraySort((x, y) -> y,
         [1224, 1923, 1003, 745],
         [2,1,3,4]) AS s

┌─s──────────────────────┐
│ [1923,1224,1003,745]   │
└────────────────────────┘
```

Sort array in ascending order

Lambda reverses order

Lambda applies new keys

We can now coordinate array ordering

```
SELECT Carrier, TailNum, FlightNum,
  arraySort(groupArray(DepTime)) as Departures,
  arraySort((x,y) -> y, groupArray(Origin), groupArray(DepTime)) as Origins,
  length(groupArray(DepTime)) as Hops
FROM ontime
WHERE (FlightDate = toDate('2017-01-15')) AND (DepTime < ArrTime)
GROUP BY Carrier, TailNum, FlightNum
ORDER BY Hops DESC, Carrier, TailNum, FlightNum
LIMIT 5
```

Carrier	TailNum	FlightNum	Departures	Origins	Hops
WN	N963WN	5663	[745,1003,1224,1439,1616,1923,2140]	['OMA','MDW','GRR','BWI','PIT','MCO','ATL']	7
WN	N223WN	4315	[1041,1237,1529,1722,1920,2105]	['CLE','BWI','OKC','DAL','MSY','BNA']	6
WN	N438WN	4415	[629,1200,1436,1748,1922,2127]	['SAN','AUS','BNA','LAX','LAS','SMF']	6
WN	N445WN	3462	[651,1008,1334,1512,1635,1827]	['MSY','MCO','GRR','DEN','LAS','SJC']	6
WN	N451WN	4362	[600,843,1058,1240,1724,1945]	['CLT','HOU','SNA','LAS','ICT','STL']	6

Let’s put the routes into a nice tabular form

```
SELECT Carrier, TailNum, FlightNum, Depart, Arr, Origin, Dest
FROM
(
  SELECT Carrier, TailNum, FlightNum,
    arraySort(groupArray(DepTime)) AS Departures,
    arraySort(groupArray(ArrTime)) AS Arrivals,
    arraySort((x, y) -> y, groupArray(Origin), groupArray(ArrTime)) AS Origins,
    arraySort((x, y) -> y, groupArray(Dest), groupArray(ArrTime)) AS Destinations,
    length(groupArray(DepTime)) AS Hops
  FROM ontime
  WHERE (FlightDate = toDate('2017-01-15')) AND (DepTime < ArrTime)
  GROUP BY Carrier, TailNum, FlightNum
  ORDER BY Hops DESC, Carrier, TailNum, FlightNum
)
ARRAY JOIN Departures AS Depart, Arrivals AS Arr,
    Origins AS Origin, Destinations AS Dest
```

And here is our winner!

Carrier	TailNum	FlightNum	Depart	Arr	Origin	Dest
WN	N963WN	5663	745	909	OMA	MDW
WN	N963WN	5663	1003	1145	MDW	GRR
WN	N963WN	5663	1224	1346	GRR	BWI
WN	N963WN	5663	1439	1535	BWI	PIT
WN	N963WN	5663	1616	1821	PIT	MCO
WN	N963WN	5663	1923	2037	MCO	ATL
WN	N963WN	5663	2140	2333	ATL	DTW
WN	N223WN	4315	1041	1150	CLE	BWI
WN	N223WN	4315	1237	1451	BWI	OKC
WN	N223WN	4315	1529	1618	OKC	DAL
WN	N223WN	4315	1722	1844	DAL	MSY
WN	N223WN	4315	1920	2040	MSY	BNA
WN	N223WN	4315	2105	2334	BNA	DCA
...						

Building funnels with arrays

More questions about airplanes

How many planes fly each day?

How many of those fly to Chicago (MDW)?

How many of those then fly to Atlanta (ATL) the same day?

See talk by Mariya Mansurova from Yandex:
https://www.youtube.com/watch?v=YpurT78U2qA

Start with a base query

```
SELECT FlightDate, Carrier, TailNum, FlightNum,
    arraySort(groupArray(ArrTime)) AS Arrivals,
    arraySort((x, y) -> y, groupArray(Dest), groupArray(ArrTime)) AS Dests
  FROM ontime
  WHERE toYYYYMM(FlightDate) = toYYYYMM(toDate('2017-01-01'))
    AND (DepTime < ArrTime)
  GROUP BY FlightDate, Carrier, TailNum, FlightNum
  ORDER BY FlightDate, Carrier, TailNum, FlightNum
LIMIT 15
```

FlightDate	Carrier	TailNum	FlightNum	Arrivals	Dests
2017-01-01	AA	N001AA	1446	[2213]	['ATL']
2017-01-01	AA	N003AA	2528	[1514,1914]	['HDN','DFW']
2017-01-01	AA	N005AA	2569	[1238,1757]	['JAC','ORD']
2017-01-01	AA	N006AA	1287	[850]	['ORD']
2017-01-01	AA	N006AA	2193	[2230]	['MIA']
2017-01-01	AA	N006AA	2635	[1209,1730]	['EGE','ORD']

. . .

We can filter values using arrayFilter

```
WITH ['a', 'bc', 'def', 'g'] AS array
SELECT arrayFilter(v -> (length(v) > 1), array) AS filtered
```

```
┌─filtered──────┐
│ ['bc','def'] │
└───────────────┘
```

```
WITH
    ['name', 'owner', 'account'] AS tags,
    ['joe', 'susie', 'website'] AS values
SELECT arrayFilter((v, t) -> (t = 'name'), values, tags)[1] AS name
```

```
┌─name─┐
│ joe  │
└──────┘
```

Input arrays

Lambda

Input arrays

Filter the aircraft that fly via MDW, ATL

```
SELECT FlightDate, Carrier, TailNum, FlightNum,
    arraySort(groupArray(ArrTime)) AS Arrivals,
    arraySort((x, y) -> y, groupArray(Dest), groupArray(ArrTime)) AS Dests,
    arrayFilter((arr, dest) -> (dest = 'MDW'), Arrivals, Dests)[1] as MDW_Arr,
    arrayFilter((arr, dest) -> (dest = 'ATL' AND arr > MDW_Arr And MDW_Arr !=
0), Arrivals, Dests)[1] as ATL_arr,
    length(Arrivals) as Hops
  FROM ontime
  WHERE toYYYYMM(FlightDate) = toYYYYMM(toDate('2017-01-15'))
    AND (DepTime < ArrTime)
  GROUP BY FlightDate, Carrier, TailNum, FlightNum
  HAVING has(Dests, 'MDW') AND has(Dests, 'ATL') AND Hops > 3
  ORDER BY FlightDate, Hops DESC, Carrier, TailNum, FlightNum
LIMIT 15
```

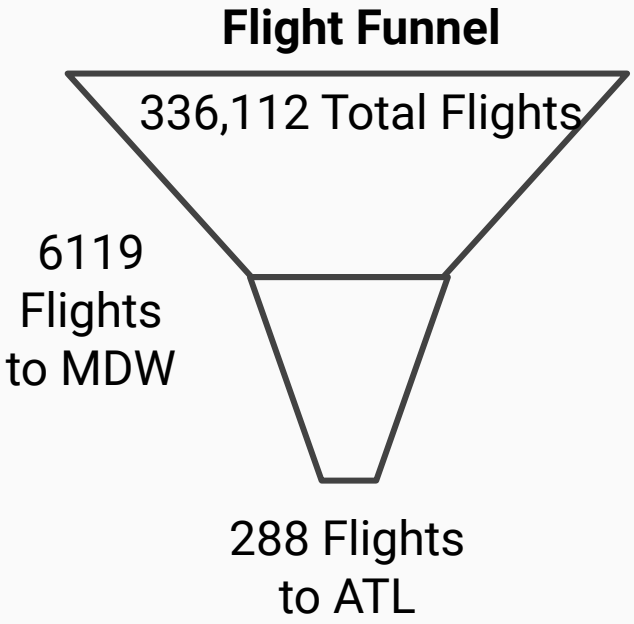
Now we see the daily pattern for all flights

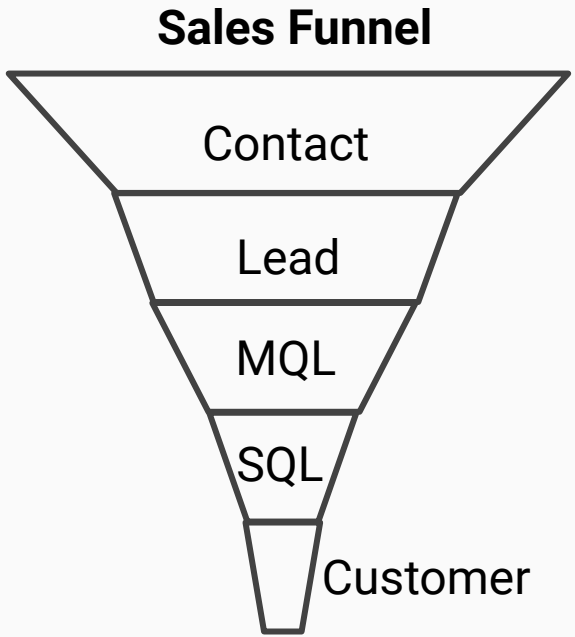
```
SELECT
  FlightDate,
  count(*) AS Total,
  sum(MDW_Arr != 0) AS MDW_tot,
  sum(ATL_Arr != 0) AS ATL_tot
FROM (
  -- (Query from previous slide)
) GROUP BY FlightDate ORDER By FlightDate
```

FlightDate	Total	MDW_tot	ATL_tot
2017-01-01	10636	202	6
2017-01-02	11378	192	7
2017-01-03	11243	213	8
2017-01-04	10881	218	8
2017-01-05	11035	211	13

. . .

The result looks like a funnel!





ClickHouse Functional Programming

Higher-Order Functions

Classic set:

- arrayFilter: array(T) -> array(T)
- arrayMap: array(T1) -> array(T2)
- arrayReduce: array(T) -> N

Some extras:

- arrayExists
- arrayFirst/arrayFirstIndex
- arraySort/arrayReverseSort
- arraySum/arrayCumSum

```
function( T -> V, array(T))

function( (T1, ... , Tn) -> V,
    array(T1), ... ,array(Tn))

e.g.

arrayFilter(x -> x>0, col1)
```

Introducing the versatile arrayMap function

```
WITH ['tiny', 'longer', 'longest'] AS array
SELECT arrayMap(v -> substr(v, 1, 4), array) AS map
```

```
┌─map──────────────────────┐
│ ['tiny','long','long'] │
└──────────────────────────┘
```

```
WITH
    ['tiny', 'longer', 'longest'] AS array,
    [1, 2, 3] AS length
SELECT arrayMap((v, l) -> substr(v, 1, l), array, length) AS map
```

```
┌─map──────────────────┐
│ ['t','lo','lon'] │
└──────────────────────┘
```

Plus some other handy array functions

```
WITH ['tiny', 'longer', 'longest'] AS array
SELECT
    arrayReverse(array) AS reversed,
    arrayEnumerate(array) AS indexes,
    arrayFirstIndex(s -> (length(s) > 5), array) AS longest,
    length(array) AS length
```

reversed	indexes	longest	length
['longest','longer','tiny']	[1,2,3]	2	3

Weighted sum? Lambdas can do it!

```
WITH
    [1, 2, 3, 4, 5] AS arr_v,
    [0.5, 0.5, 0.3, 0.2, 0.2] AS arr_w
SELECT arraySum(arrayMap((v, w) -> (v * w), arr_v, arr_w))

┌─arraySum(arrayMap(lambda(tuple(v, w), multiply(v, w)), arr_v, arr_w))─┐
│                                                                   4.2 │
└───────────────────────────────────────────────────────────────────────┘
```

```
WITH
    [1, 2, 3, 4, 5] AS arr_v,
    [0.5, 0.5, 0.3, 0.2, 0.2] AS arr_w
SELECT arraySum((v, w) -> (v * w), arr_v, arr_w)

┌─arraySum(lambda(tuple(v, w), multiply(v, w)), arr_v, arr_w)─┐
│                                                         4.2 │
└─────────────────────────────────────────────────────────────┘
```

arrayReduce offers processing that is flexible

```
WITH [1, 2, 3, 4, 5] AS arr
SELECT arraySum(arr)

┌─arraySum(arr)─┐
│             15 │
└────────────────┘
```

```
WITH [1, 2, 3, 4, 5] AS arr
SELECT arrayReduce('sum', arr)

┌─arrayReduce('sum', arr)─┐
│                      15 │
└─────────────────────────┘
```

arrayReduce applies aggregation function to fold the array

...and fast, as arrayReduce is vectorised

```
SELECT arraySum(range(1, 100000000))

┌─arraySum(range(1, 100000000))─┐
│               4999999950000000 │
└────────────────────────────────┘

1 rows in set. Elapsed: 0.991 sec.
```

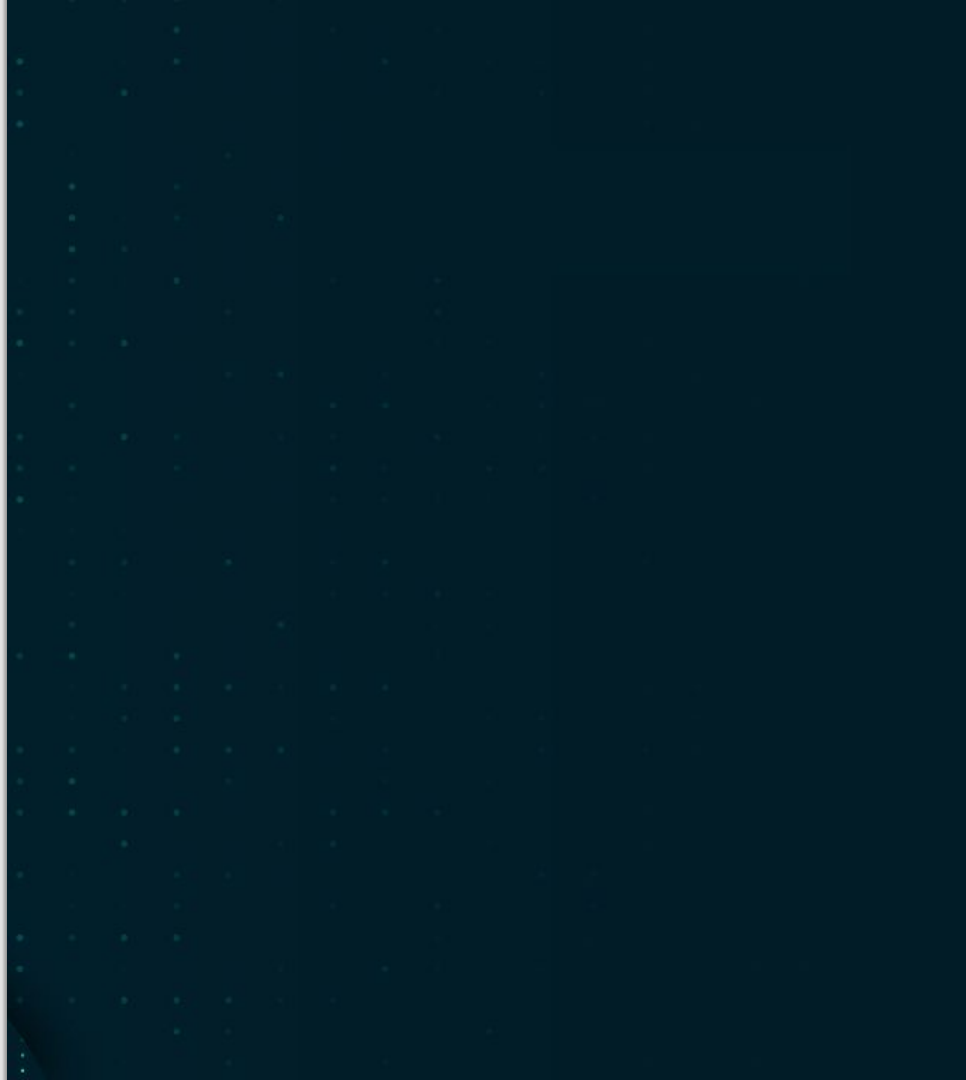
```
SELECT arrayReduce('sum', range(1, 100000000))

┌─arrayReduce('sum', range(1, 100000000))─┐
│                         4999999950000000 │
└──────────────────────────────────────────┘

1 rows in set. Elapsed: 0.305 sec.
```

Interpolation with arrayMap

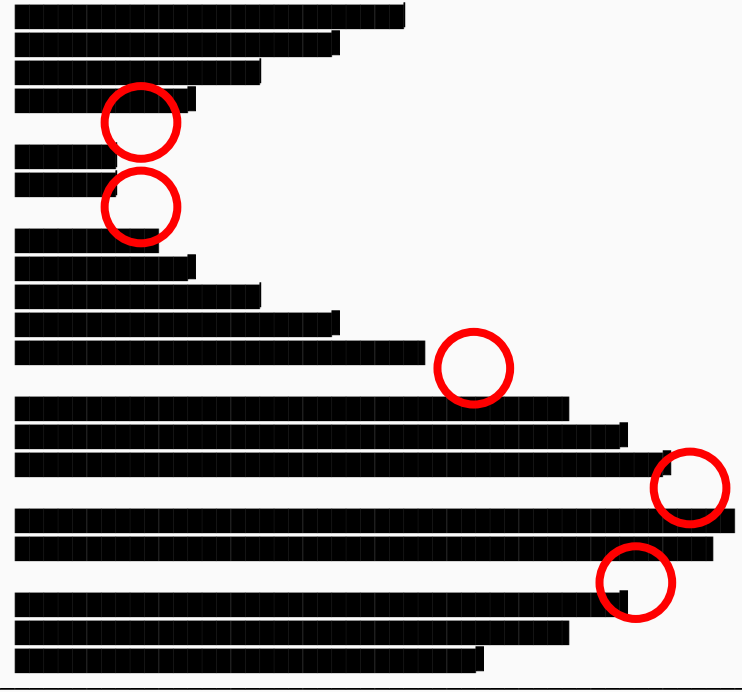
(Some people don’t know how to quit when they are ahead)



How do you deal with broken data?

```
SELECT datetime, temp, bar(temp, 30., 65., 50) FROM readings
```

datetime	temp	bar(temp, 30., 65., 50)
2020-01-01 00:00:00	49.99996	
2020-01-01 01:00:00	46.11768	
2020-01-01 02:00:00	42.49997	
2020-01-01 03:00:00	39.393375	
2020-01-01 04:00:00	0	
2020-01-01 05:00:00	35.51111	
2020-01-01 06:00:00	35	
2020-01-01 07:00:00	0	
2020-01-01 08:00:00	37.009624	
2020-01-01 09:00:00	39.393406	
2020-01-01 10:00:00	42.500008	
2020-01-01 11:00:00	46.117718	
2020-01-01 12:00:00	50	
2020-01-01 13:00:00	0	
2020-01-01 14:00:00	57.499992	
2020-01-01 15:00:00	60.606594	
2020-01-01 16:00:00	62.990376	
2020-01-01 17:00:00	0	
2020-01-01 18:00:00	65	
2020-01-01 19:00:00	64.48889	
2020-01-01 20:00:00	0	
2020-01-01 21:00:00	60.606625	
2020-01-01 22:00:00	57.50003	
2020-01-01 23:00:00	53.88232	



Let’s fix up the data using interpolation

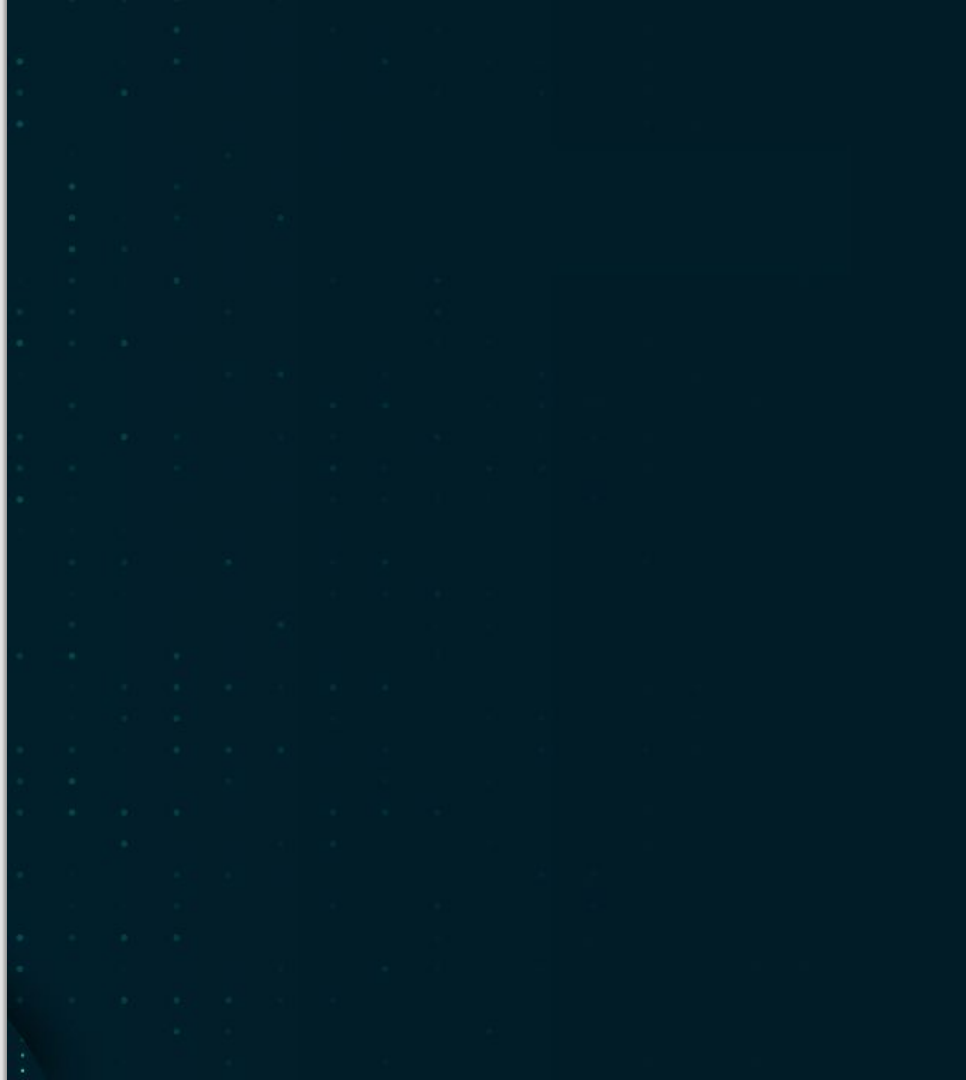
```
SELECT toDate(datetime) as day,
  groupArray(datetime) as datetimeA, groupArray(temp) as tempA,
  length(tempA) as length, arrayReverse(tempA) as tempAReverse,
  /* Indexes of arrays. */
  arrayEnumerate(datetimeA) as indexes,
  /* For each tempA[i], find first k such that k >= i and tempA[k] > 0 */
  arrayMap(i ->
    arrayFirstIndex(s-> s >= i AND tempA[s] > 0, indexes), indexes) as aA,
  /* For each tempA[i], find last l such that l <= i and tempA[l] > 0 */
  arrayReverse(arrayMap(i -> length + 1 - arrayFirstIndex(s-> s >= i AND
tempAReverse[s] > 0, indexes), indexes)) as bA,
  /* Finally...put an interpolated value in any temperature that is 0. */
  arrayMap(i ->
    if(tempA[i] > 0,
       tempA[i],
       tempA[bA[i]] + (tempA[aA[i]] - tempA[bA[i]])/(aA[i] - bA[i])),
    indexes) AS tempAFixed
FROM readings GROUP BY day order by day FORMAT Vertical
```

Now our data looks better...

```
SELECT datetime, fixed_temp, bar(fixed_temp, 30., 65., 50) FROM
 ( -- ) ARRAY JOIN datetimeA AS datetime, tempAFixed as fixed_temp
```

datetime	fixed_temp	bar(fixed_temp, 30., 65., 50)
2020-01-01 00:00:00	49.999961853027344	
2020-01-01 01:00:00	46.117679595947266	
2020-01-01 02:00:00	42.499969482421875	
2020-01-01 03:00:00	39.393375396728516	
2020-01-01 04:00:00	37.45224189758301	
2020-01-01 05:00:00	35.5111083984375	
2020-01-01 06:00:00	35	
2020-01-01 07:00:00	36.004812240600586	
2020-01-01 08:00:00	37.00962448120117	
2020-01-01 09:00:00	39.39340591430664	
2020-01-01 10:00:00	42.50000762939453	
2020-01-01 11:00:00	46.11771774291992	
2020-01-01 12:00:00	50	
2020-01-01 13:00:00	53.749996185302734	
2020-01-01 14:00:00	57.49999237060547	
2020-01-01 15:00:00	60.60659408569336	
2020-01-01 16:00:00	62.99037551879883	
2020-01-01 17:00:00	63.995187759399414	
2020-01-01 18:00:00	65	
2020-01-01 19:00:00	64.4888916015625	
2020-01-01 20:00:00	62.54775810241699	
2020-01-01 21:00:00	60.606624603271484	
2020-01-01 22:00:00	57.500030517578125	
2020-01-01 23:00:00	53.882320404052734	

Almost done...



Wrap-up

- Use arrays to model vectors or matrices
  - Including paired key/value pairs
- ARRAY JOIN unrolls arrays into rows
- groupArray creates arrays from GROUPed values
- Rich library of array functions in ClickHouse
- Higher or functions with lambdas allow versatile behavior

ClickHouse arrays have many more capabilities!

More information

- ClickHouse documentation: https://clickhouse.tech/
- Talk by Mariya Mansurova from Yandex at 2017 ClickHouse Meetup in Berlin: https://www.youtube.com/watch?v=YpurT78U2qA
- Altinity blog: https://altinity.com/blog/clickhouse-vs-redshift-performance-for-fintech-risk-management

Thank you!

We’re hiring

www.altinity.com

Email:
info@altinity.com

ClickHouse:
https://github.com/ClickHouse/ClickHouse

Altinity.Cloud:
https://altinity.com/cloud-database/