Windowing Functions

Hitoshi Harada
(umi.tanuki@gmail.com)
Head of Engineering Dept.
FORCIA, Inc.
David Fetter
(david.fetter@pgexperts.com)

PGCon2009, 21-22 May 2009 Ottawa

What are Windowing Functions?

What are the Windowing Functions?

- Similar to classical aggregates but does more!
- Provides access to set of rows from the current row
- Introduced SQL:2003 and more detail in SQL:2008
  - SQL:2008 (http://wiscorp.com/sql200n.zip) 02: SQL/Foundation
    - 4.14.9 Window tables
    - 4.15.3 Window functions
    - 6.10 <window function>
    - 7.8 <window clause>
- Supported by Oracle, SQL Server, Sybase and DB2
  - No open source RDBMS so far except PostgreSQL (Firebird trying)
- Used in OLAP mainly but also useful in OLTP
  - Analysis and reporting by rankings, cumulative aggregates

How they work and what you get

How they work and what do you get

- Windowed table
  - Operates on a windowed table
  - Returns a value for each row
  - Returned value is calculated from the rows in the window

How they work and what do you get

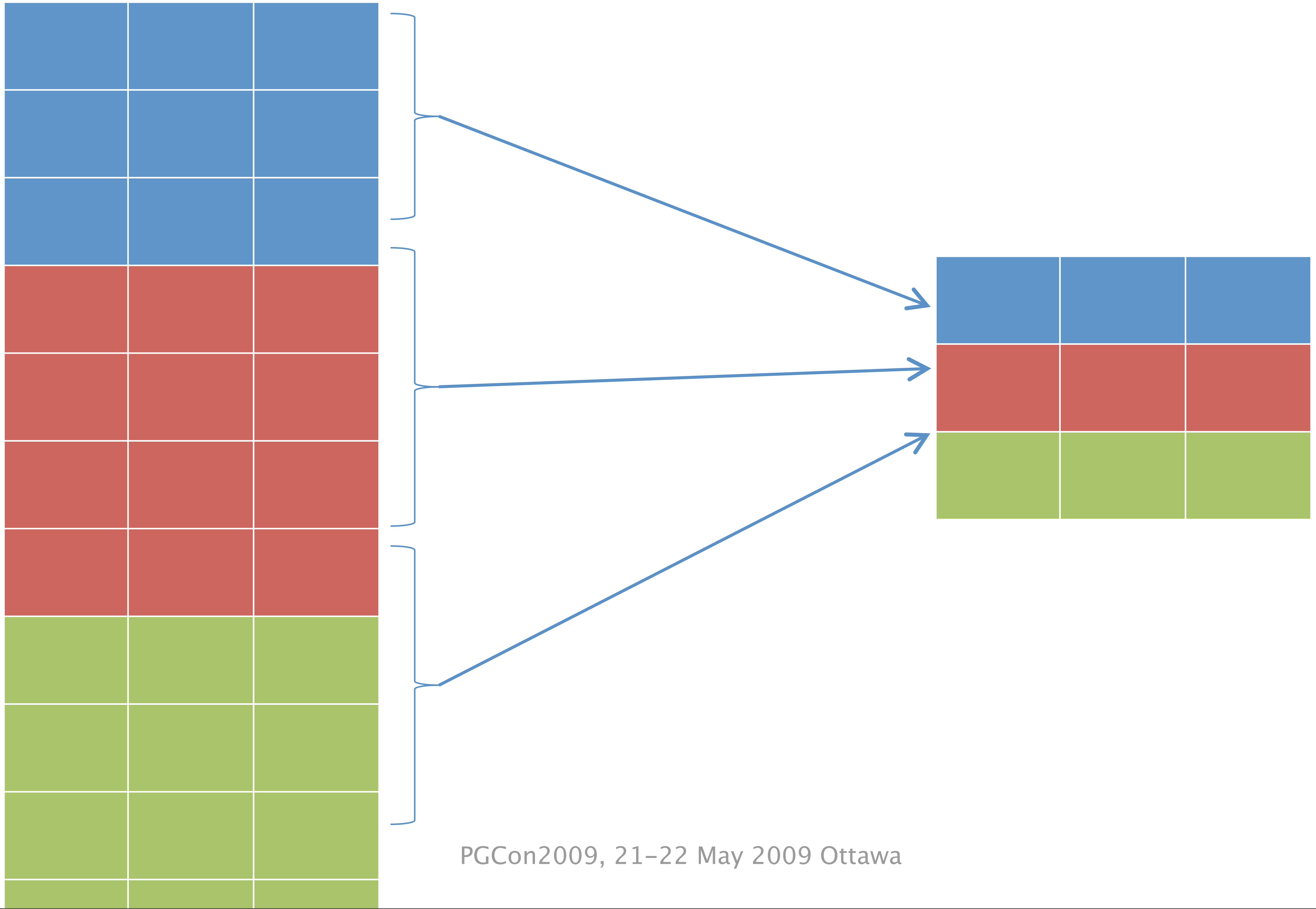
- You can use…
  - New window functions
  - Existing aggregate functions
  - User-defined window functions
  - User-defined aggregate functions

How they work and what do you get

- Completely new concept!
  - With Windowing Functions, you can reach outside the current row

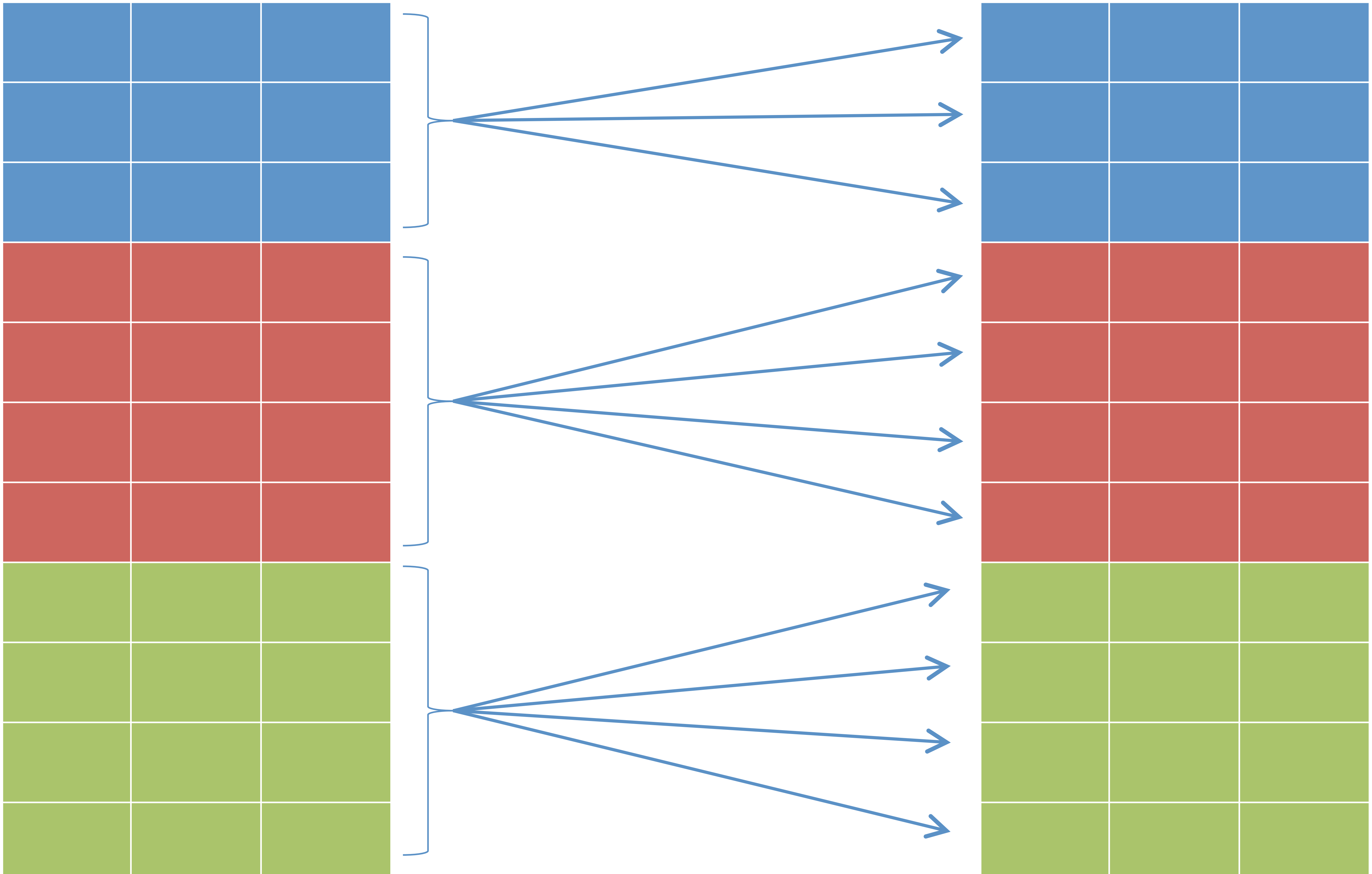
How they work and what you get

[Aggregates] SELECT key, SUM(val) FROM tbl GROUP BY key;



How they work and what you get

[Windowing Functions] SELECT key, SUM(val) OVER (PARTITION BY key) FROM tbl;



How they work and what you get

Who is the highest paid relatively compared with the department average?

depname	empno	salary
develop	10	5200
sales	1	5000
personnel	5	3500
sales	4	4800
sales	6	550
personnel	2	3900
develop	7	4200
develop	9	4500
sales	3	4800
develop	8	6000
develop	11	5200

How they work and what you get

```
SELECT depname, empno, salary, avg(salary) OVER (PARTITION BY depname)::int,
             salary - avg(salary) OVER (PARTITION BY depname)::int AS diff
FROM empsalary ORDER BY diff DESC
```

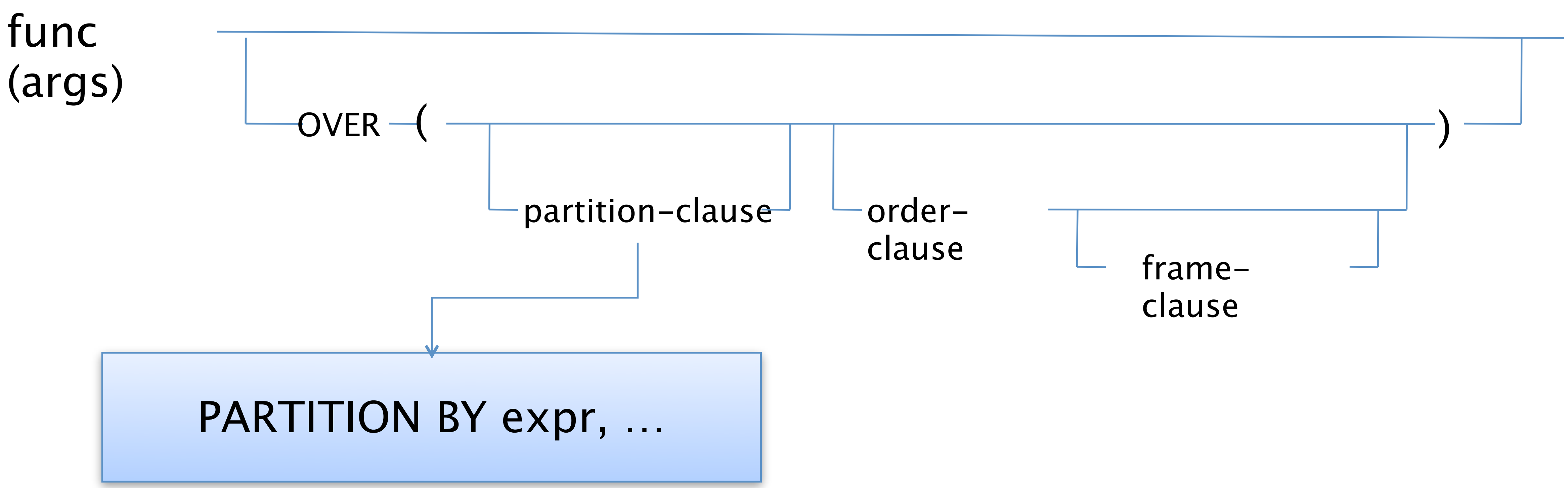
depname	empno	salary	avg	diff
develop	8	6000	5020	980
sales	6	5500	5025	475
personnel	2	3900	3700	200
develop	11	5200	5020	180
develop	10	5200	5020	180
sales	1	5000	5025	-25
personnel	5	3500	3700	-200
sales	4	4800	5025	-225
sales	3	4800	5025	-225
develop	9	4500	5020	-520
develop	7	4200	5020	-820

Anatomy of a Window

- Represents set of rows abstractly as:
- A partition
  - Specified by PARTITION BY clause
  - Never moves
  - Can contain:
- A frame
  - Specified by ORDER BY clause and frame clause
  - Defined in a partition
  - Moves within a partition
  - Never goes across two partitions
- Some functions take values from a partition. Others take them from a frame.

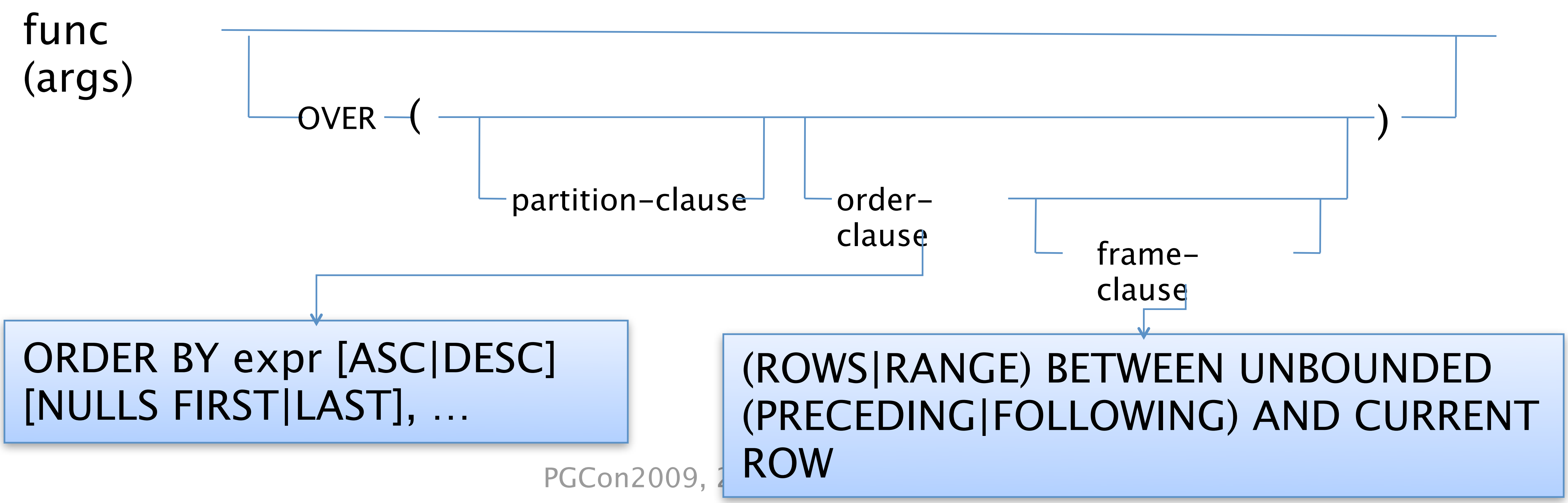
A partition

- Specified by PARTITION BY clause in OVER()
- Allows to subdivide the table, much like GROUP BY clause
- Without a PARTITION BY clause, the whole table is in a single partition



A frame

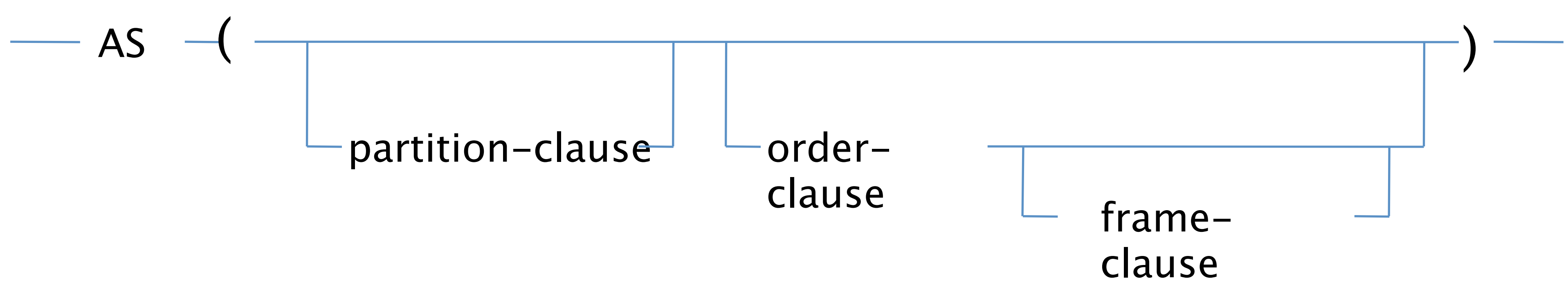
- Specified by ORDER BY clause and frame clause in OVER()
- Allows to tell how far the set is applied
- Also defines ordering of the set
- Without order and frame clauses, the whole of partition is a single frame



The WINDOW clause

- You can specify common window definitions in one place and name it, for convenience
- You can use the window at the same query level

SELECT target_list, … wfunc() OVER w
FROM table_list, …
WHERE qual_list, ….
GROUP BY groupkey_list, ….
HAVING groupqual_list, …
WINDOW w



Built-in Windowing Functions

List of built-in Windowing Functions

- row_number()
- rank()
- dense_rank()
- percent_rank()
- cume_dist()
- ntile()
- lag()
- lead()
- first_value()
- last_value()
- nth_value()

row_number()

- Returns number of the current row

SELECT val, row_number() OVER (ORDER BY val DESC) FROM tbl;

val	row_number()
5	1
5	2
3	3
1	4

Note: row_number() always incremented values independent of frame

rank()

- Returns rank of the current row with gap

SELECT val, rank() OVER (ORDER BY val DESC) FROM tbl;

val	rank()
5	1
5	1
3	3
1	4

gap

Note: rank() OVER(*empty*) returns 1 for all rows, since all rows are peers to each other

dense_rank()

- Returns rank of the current row without gap

```
SELECT val, dense_rank() OVER (ORDER BY val DESC) FROM tbl;
```

val	dense_rank()
5	1
5	1
3	2
1	3

no gap

Note: dense_rank() OVER(*empty*) returns 1 for all rows, since all rows are peers to each other

percent_rank()

- Returns relative rank; (rank() – 1) / (total row – 1)

SELECT val, percent_rank() OVER (ORDER BY val DESC) FROM tbl;

val	percent_rank()
5	0
5	0
3	0.666666666666667
1	1

values are always between 0 and 1 inclusive.

Note: percent_rank() OVER(*empty*) returns 0 for all rows, since all rows are peers to each other

cume_dist()

- Returns relative rank; (# of preced. or peers) / (total row)

SELECT val, cume_dist() OVER (ORDER BY val DESC) FROM tbl;

val	cume_dist()	
5	0.5	= 2 / 4
5	0.5	= 2 / 4
3	0.75	= 3 / 4
1	1	= 4 / 4

The result can be emulated by
"count(*) OVER (ORDER BY val DESC) / count(*) OVER ()"

Note: cume_dist() OVER(*empty*) returns 1 for all rows, since all rows are peers to each other

ntile()

- Returns dividing bucket number

SELECT val, ntile(3) OVER (ORDER BY val DESC) FROM tbl;

val	ntile(3)
5	1
5	1
3	2
1	3

4 % 3 = 1

The results are the divided positions, but if there's remainder add row from the head

Note: ntile() OVER (*empty*) returns same values as above, since ntile() doesn't care the frame but works against the partition

lag()

- Returns value of row above

```
SELECT val, lag(val) OVER (ORDER BY val DESC) FROM tbl;
```

val	lag(val)
5	NULL
5	5
3	5
1	3

Note: lag() only acts on a partition.

lead()

- Returns value of the row below

```
SELECT val, lead(val) OVER (ORDER BY val DESC) FROM tbl;
```

val	lead(val)
5	5
5	3
3	1
1	NULL

Note: lead() acts against a partition.

first_value()

- Returns the first value of the frame

```
SELECT val, first_value(val) OVER (ORDER BY val DESC) FROM tbl;
```

val	first_value(val)
5	5
5	5
3	5
1	5

last_value()

- Returns the last value of the frame

```
SELECT val, last_value(val) OVER
 (ORDER BY val DESC ROWS BETWEEN UNBOUNDED PRECEEDING
AND UNBOUNDED FOLLOWING) FROM tbl;
```

val	last_value(val)
5	1
5	1
3	1
1	1

Note: frame clause is necessary since you have a frame between the first row and the current row by only the order clause

nth_value()

- Returns the n-th value of the frame

```
SELECT val, nth_value(val, val) OVER
 (ORDER BY val DESC ROWS BETWEEN UNBOUNDED PRECEEDING
AND UNBOUNDED FOLLOWING) FROM tbl;
```

val	nth_value(val, val)
5	NULL
5	NULL
3	3
1	5

Note: frame clause is necessary since you have a frame between the first row and the current row by only the order clause

aggregates(all peers)

- Returns the same values along the frame

```
SELECT val, sum(val) OVER () FROM tbl;
```

val	sum(val)
5	14
5	14
3	14
1	14

Note: all rows are the peers to each other

cumulative aggregates

- Returns different values along the frame

```
SELECT val, sum(val) OVER (ORDER BY val DESC) FROM tbl;
```

val	sum(val)
5	10
5	10
3	13
1	14

Note: row#1 and row#2 return the same value since they are the peers.
the result of row#3 is sum(val of row#1…#3)

Inside the Implementation

Where are they?

- Windowing Functions are identified by a flag in pg_proc, which means they are very similar to plain functions
  - pg_proc.proiswindow : boolean
- All the aggregates including user-defined ones in any language can be called in WindowAgg, too
  - Your old code gets new behaviour!
- src/include/catalog/pg_proc.h
  - pg_proc catalog definitions
- src/backend/utils/adt/windowfuncs.c
  - implementations of built-in window functions

Added relevant nodes

- WindowFunc
  - primitive node for function itself “wfunc(avg1, …)”
- WindowDef
  - parse node for window definition “over (partition by … order by …)”
- WindowClause
  - parse node for window clause “window (partition by … order by …) as w”
- WindowAgg
  - plan node for window aggregate
- WindowAggState
  - executor node for window aggregate

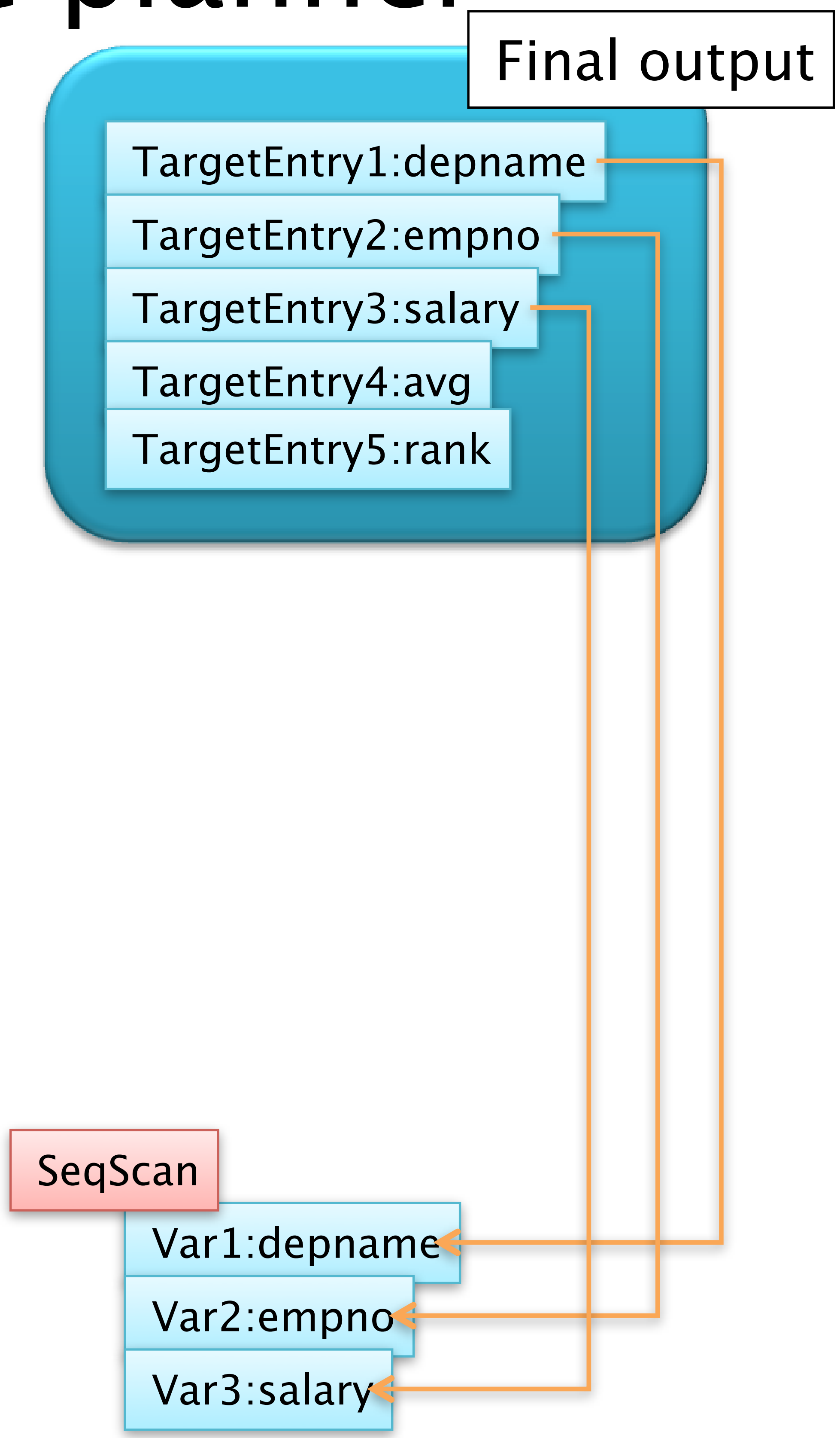
Hacking the planner

```
SELECT depname, empno, salary,
       avg(salary) over (partition by depname) AS a,
       rank() over (partition by depname order by salary desc) AS r
FROM empsalary ORDER BY r
```

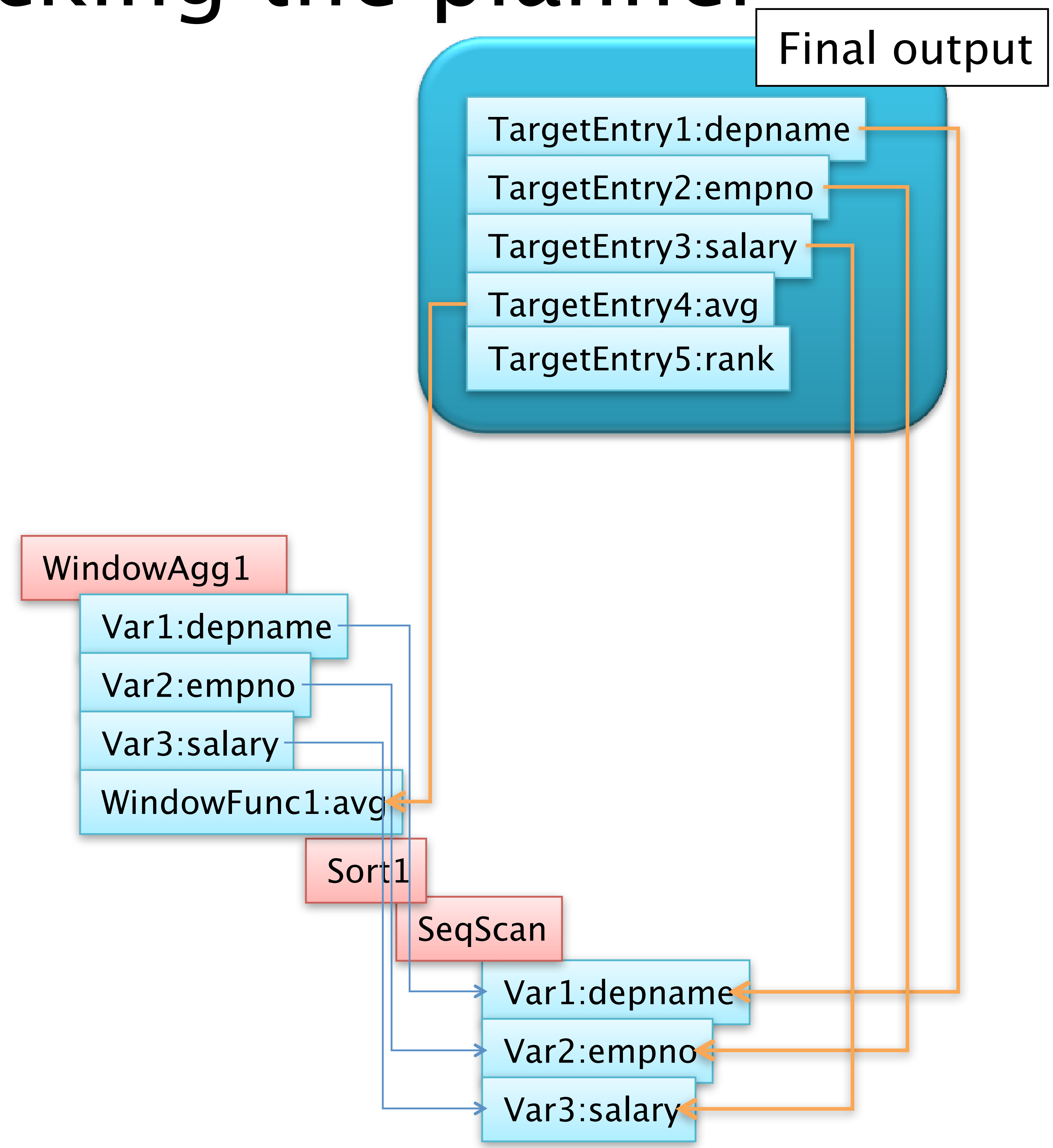
```
Sort  (cost=215.75..218.35 rows=1040 width=44)
  Output: depname, empno, salary, (avg(salary) OVER (?)), (rank() OVER (?))
  Sort Key: (rank() OVER (?))
  ->  WindowAgg  (cost=142.83..163.63 rows=1040 width=44)
        Output: depname, empno, salary, (avg(salary) OVER (?)), rank() OVER (?)
        ->  Sort  (cost=142.83..145.43 rows=1040 width=44)
              Output: depname, empno, salary, (avg(salary) OVER (?))
              Sort Key: depname, salary
              ->  WindowAgg  (cost=72.52..90.72 rows=1040 width=44)
                    Output: depname, empno, salary, avg(salary) OVER (?)
                    ->  Sort  (cost=72.52..75.12 rows=1040 width=44)
                          Output: depname, empno, salary
                          Sort Key: depname
                          ->  Seq Scan on empsalary  (cost=0.00..20.40 rows=1040 width=44)
                                Output: depname, empno, salary
```

Hacking the planner

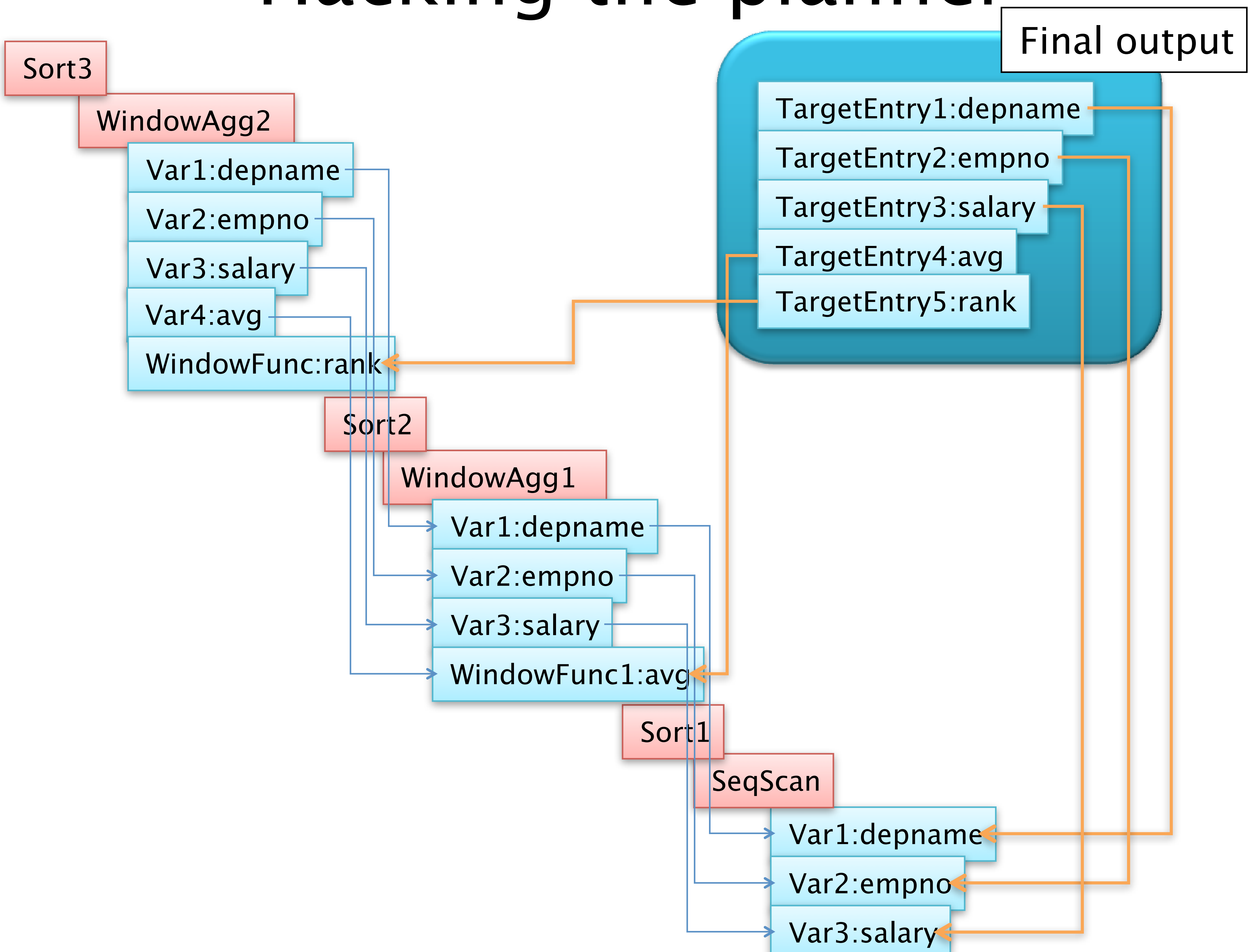
Hacking the planner



Hacking the planner



Hacking the planner



Hacking the planner

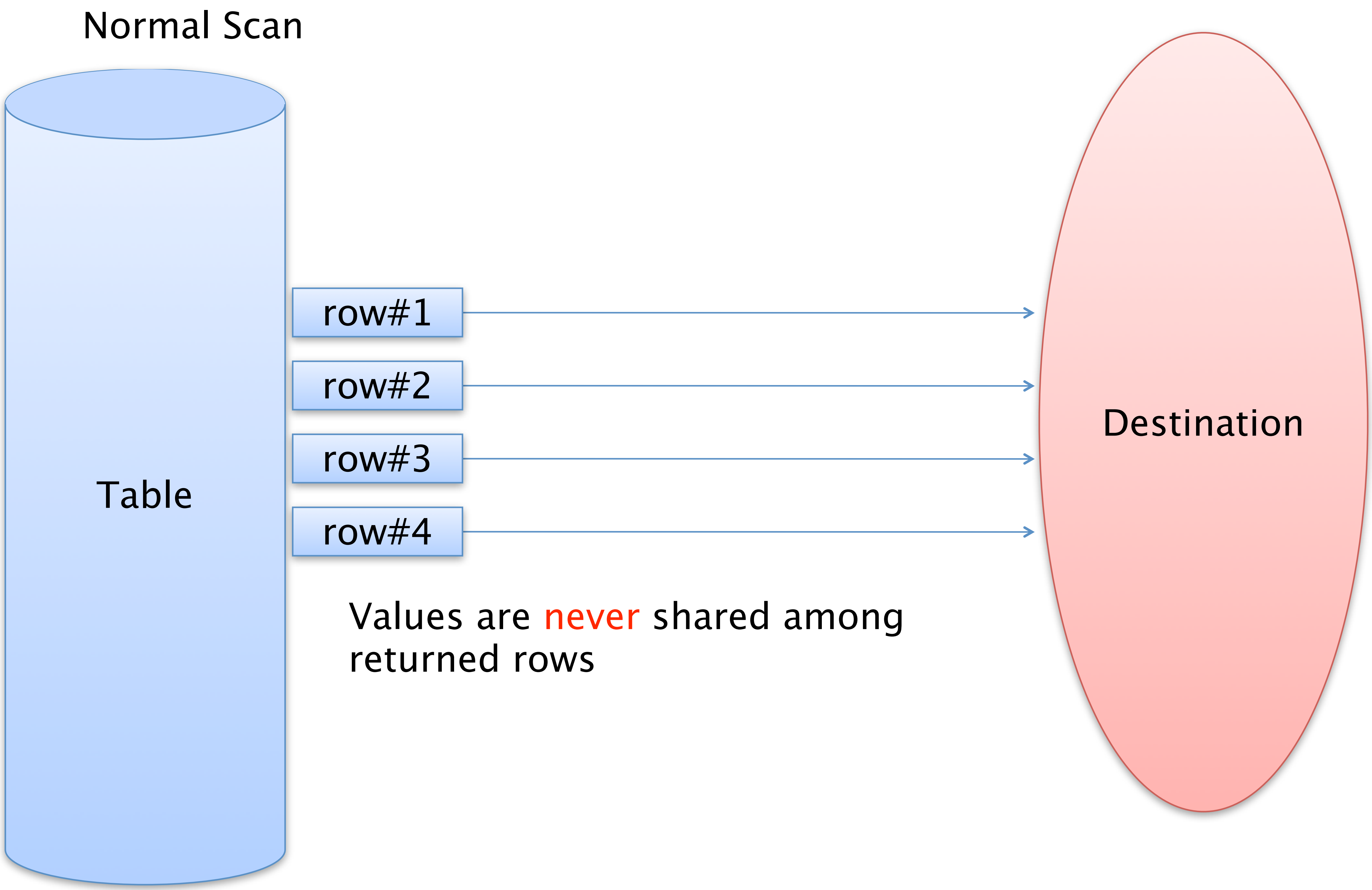
- We could optimize it by relocating WindowAgg

```
SELECT depname, empno, salary,
        rank() over (partition by depname order by salary desc) AS r,
        avg(salary) over (partition by depname) AS a
 FROM empsalary ORDER BY r
```

```
Sort  (cost=161.03..163.63 rows=1040 width=44)
  Output: depname, empno, salary, (rank() OVER (?)), (avg(salary) OVER (?))
  Sort Key: (rank() OVER (?))
  ->  WindowAgg  (cost=72.52..108.92 rows=1040 width=44)
        Output: depname, empno, salary, (rank() OVER (?)), avg(salary) OVER (?)
        ->  WindowAgg  (cost=72.52..93.32 rows=1040 width=44)
              Output: depname, empno, salary, rank() OVER (?)
              ->  Sort  (cost=72.52..75.12 rows=1040 width=44)
                    Output: depname, empno, salary
                    Sort Key: depname, salary
                    ->  Seq Scan on empsalary  (cost=0.00..20.40 rows=1040 width=44)
                          Output: depname, empno, salary
```

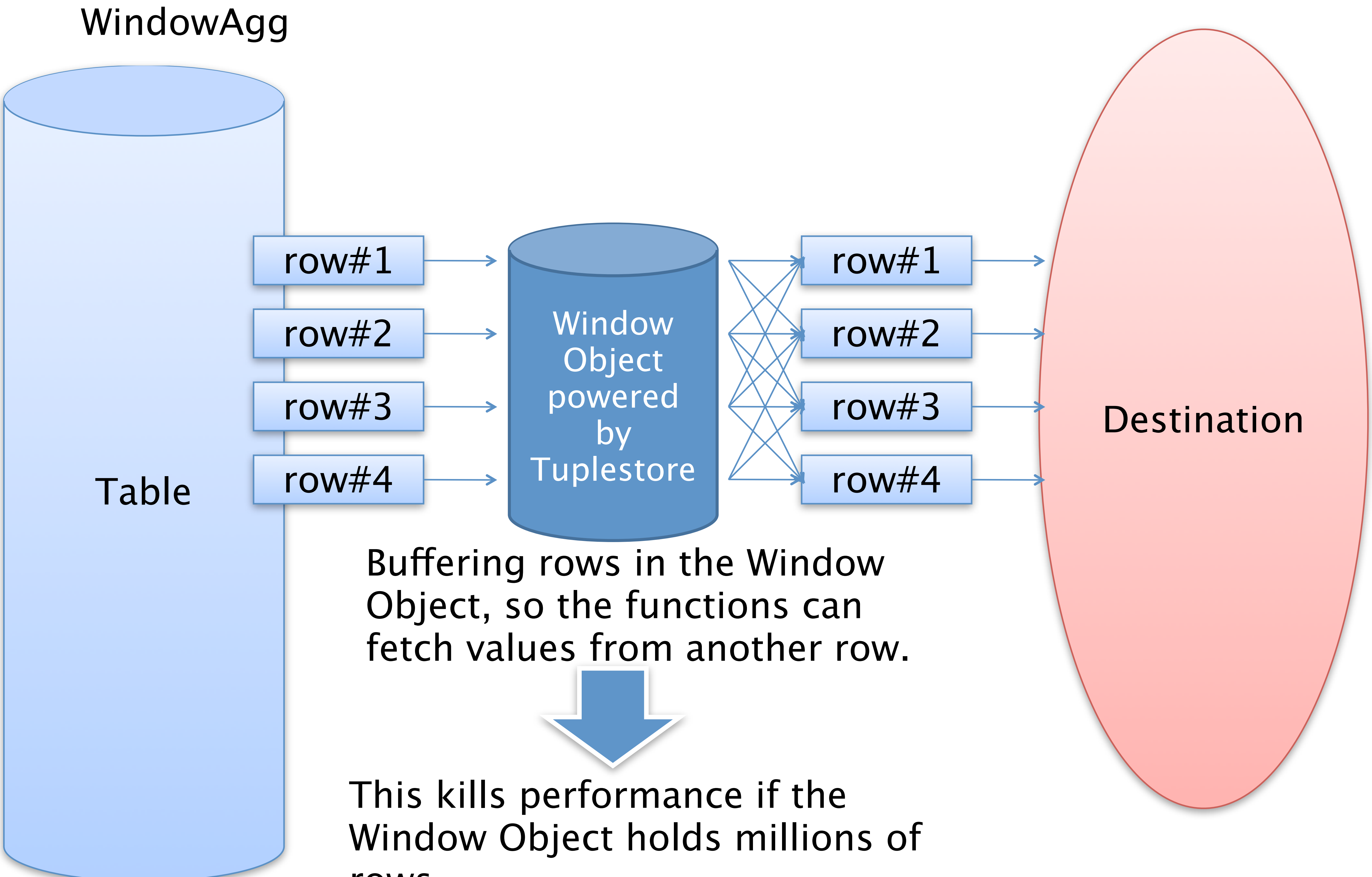
No Sort!

How the executor creates a window



Values are never shared among returned rows

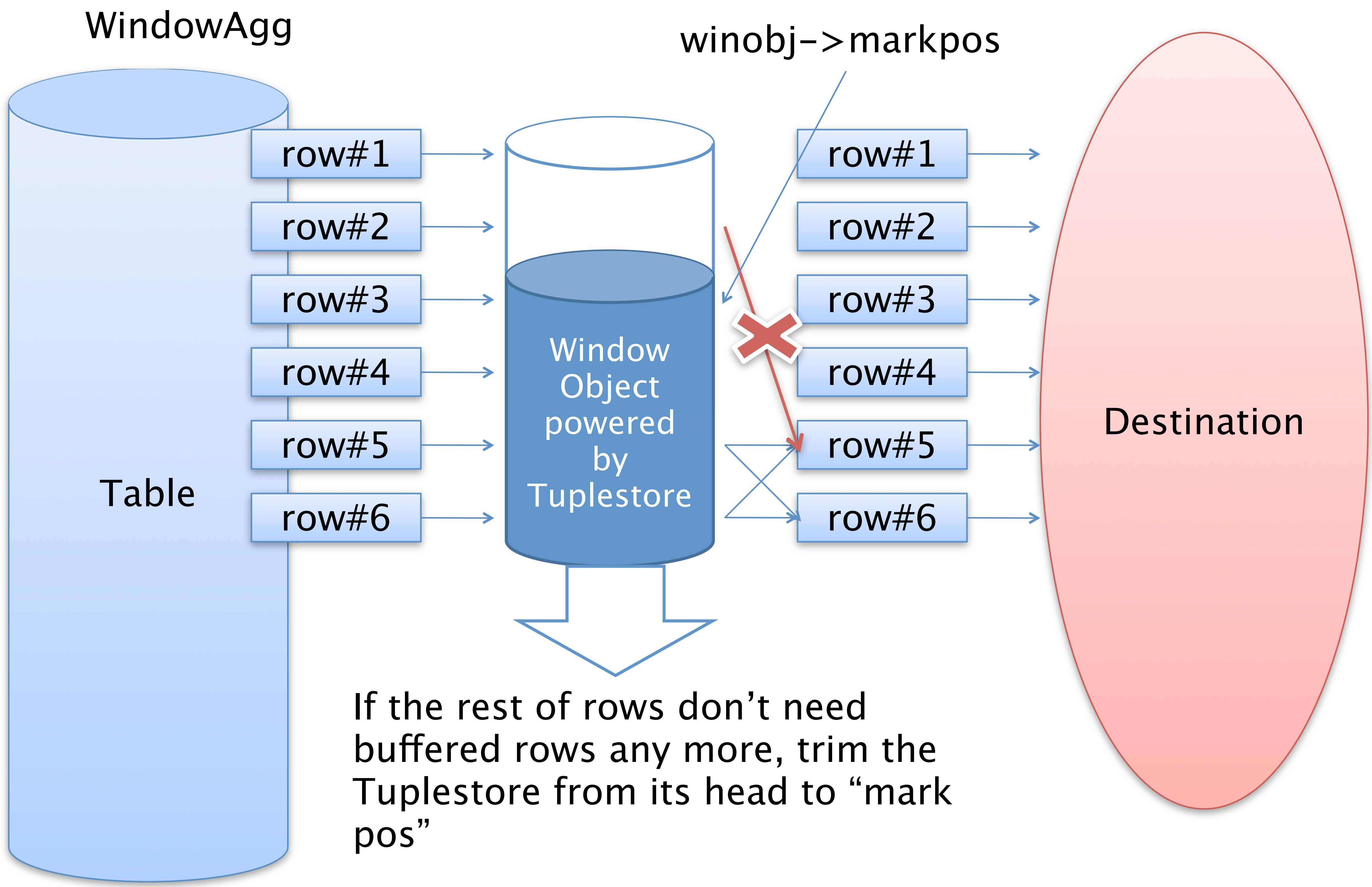
How the executor creates a window



Buffering rows in the Window Object, so the functions can fetch values from another row.

This kills performance if the Window Object holds millions of rows.

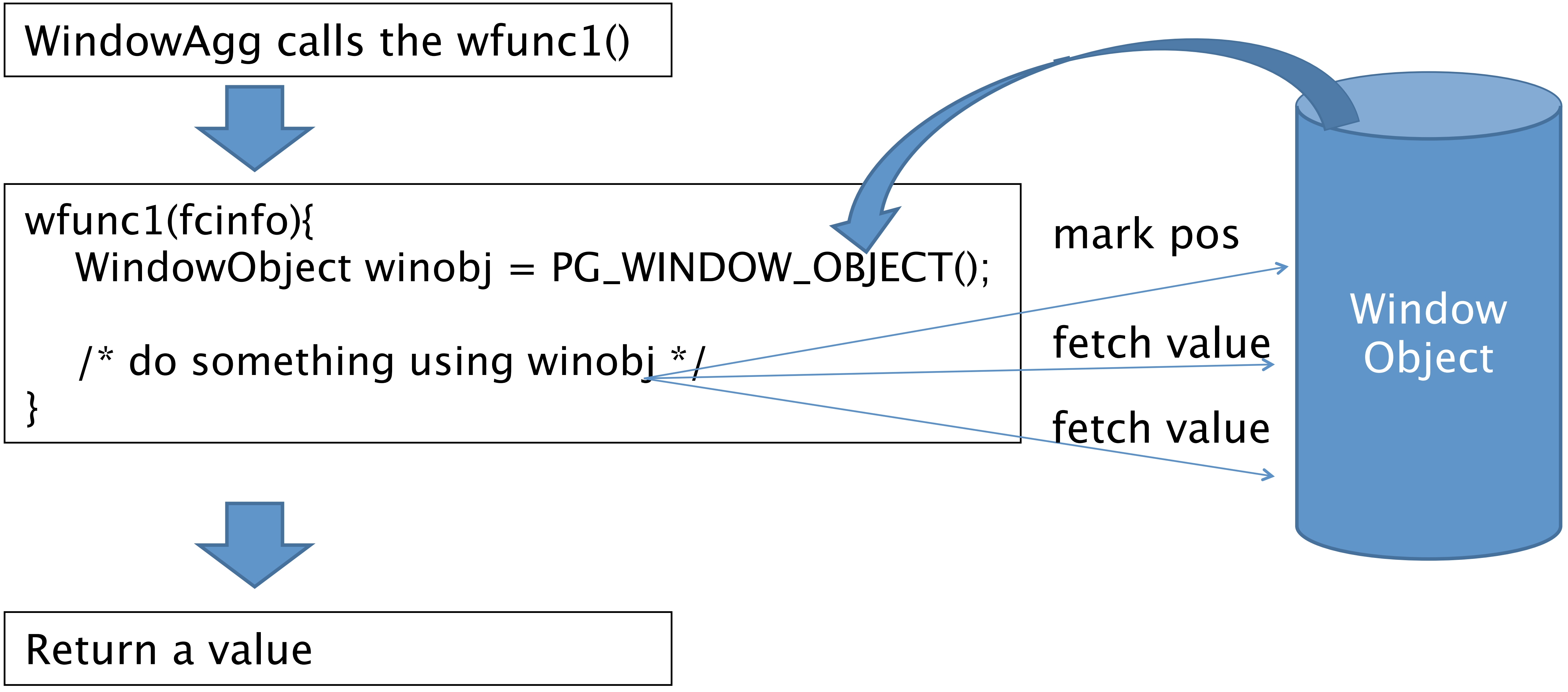
How the executor creates a window



If the rest of rows don't need buffered rows any more, trim the Tuplestore from its head to "mark pos"

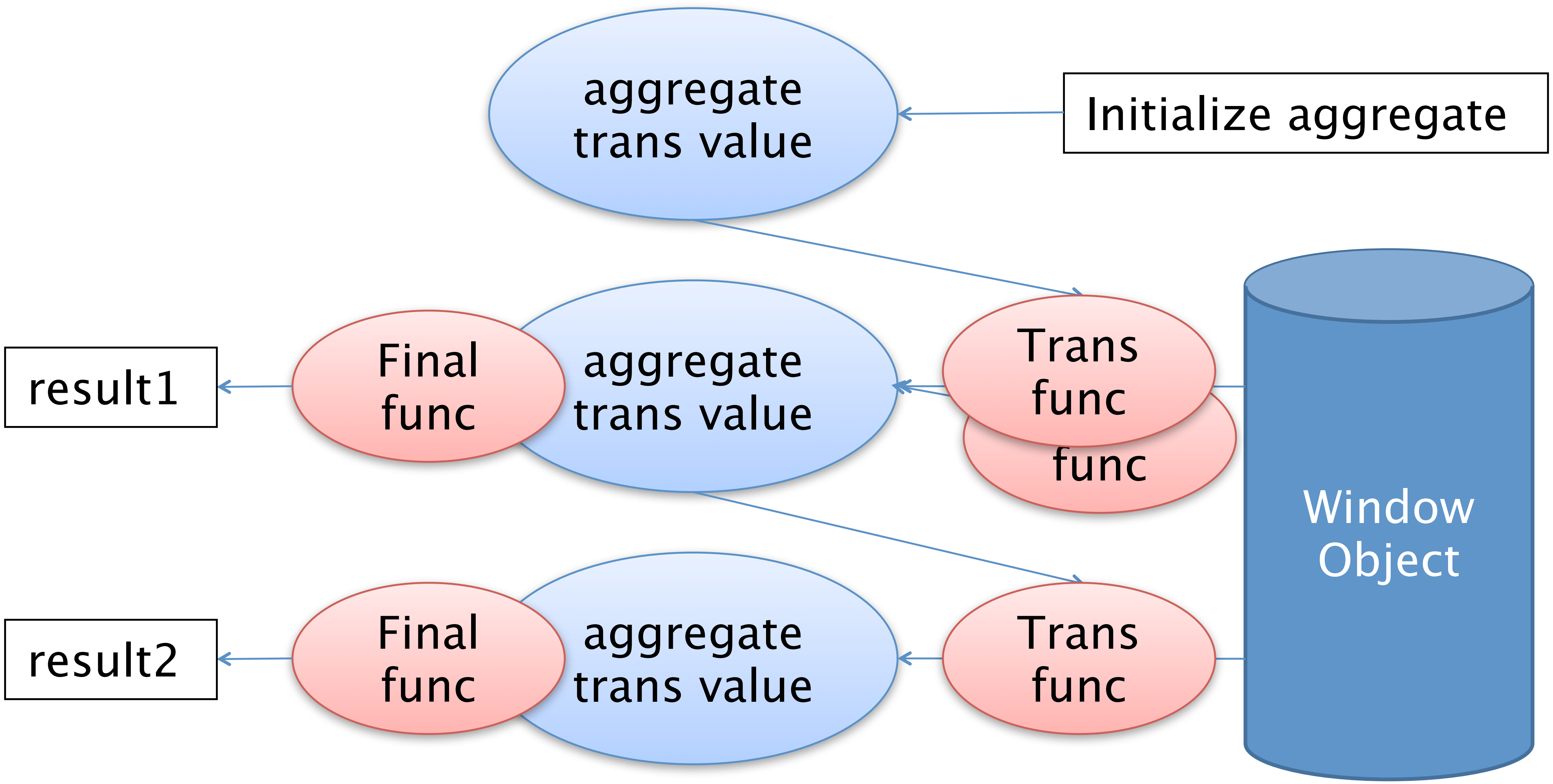
How the Windowing Functions work

- Functions workflow is very similar to plain functions
- Each function has its own seek position



How the Windowing Functions work

- What about aggregates?



Trans value is cached and reused after final function call.
-> BE CAREFUL of anything that will break if the final function is called more than once.
http://archives.postgresql.org//pgsql-hackers/2008-12/msg01820.php

Write Your Own Windowing Function

Write your own Windowing Function

- NOT documented for now, but you can do it
- The normal argument API like PG_GETARG_XXX() does not work for window functions
- STRICT-ness is meaningless
- The function must use the V1 calling convention
- Get WindowObject using PG_WINDOW_OBJECT()
- Check window-ness by WindowObjectIsValid()
- Call APIs to fetch values
- More details in src/include/windowapi.h, see also:
  - src/executor/nodeWindowAgg.c
  - src/utils/adt/windowfuncs.c
- These may be changed in the future releases

Windowing Function APIs

- PG_WINDOW_OBJECT(fcinfo)
  - Retrieve WindowObject, which is an interface of the window, for this call.
- WinGetPartitionLocalMemory(winobj, sz)
  - Store its own memory. It is used in rank() to save the current rank, for example.
- WinGetCurrentPosition(winobj)
  - Get current position in the partition (not in the frame). Same as row_number()
- WinGetPartitionRowCount(winobj)
  - Count up how many rows in the partition. Used in ntile() for example.

Windowing Function APIs

- WinSetMarkPosition(winobj, markpos)
  - Give to winobj a hint that you don't want rows preceding to markpos anymore
- WinRowsArePeers(winobj, pos1, pos2)
  - Rows at pos1 and at pos2 are peers? "Peers" means both rows have the same value in the meaning of ORDER BY columns

Windowing Function APIs

- WinGetFuncArgInPartition(winobj, argno, relpos, seektype, set_mark, isnull, isout)
  - Fetch argument from another row in the partition. “isout” will be set to true if “relpos” is out of range of the partition.
- WinGetFuncArgInFrame(winobj, argno, relpos, seektype, set_mark, isnull, isout)
  - Fetch argument from another row in the frame. “isout” will be set to true if “relpos” is out of range of the frame (may be within the partition.)
- WinGetFuncArgCurrent(winobj, argno, isnull)
  - Same as PG_GETARG_XXX(argno). Fetch argument from the current row.

Moving average example

- http://archives.postgresql.org/pgsql-hackers/2009-01/msg00562.php
- With standard SQL, you have to wait for extended frame clause support like ROWS n (PRECEDING|FOLLOWING) to calculate curve smoothing, but this sample does it now.

Future work

Future work for later versions

- More support for frame clause
  - ROWS n (PRECEDING|FOLLOWING)
  - RANGE val (PRECEDING|FOLLOWING)
  - EXCLUDE (CURRENT ROW|GROUP|TIES|NO OTHERS)
  - Not so difficult for window functions but integration with classical aggregates is hard
- Performance improvements
  - Reduce tuplestore overhead
  - Relocate WindowAgg node in the plan
- Support PLs for writing Windowing Functions
  - Let PLs call Windowing Function APIs

Finally...

Thanks all the hackers!!

- Simon Riggs
  - discussion over my first ugly and poor design
- Heikki Linnakangas
  - improvement of window object and its buffering strategy
- Tom Lane
  - code rework overall and deep reading in the SQL spec
- David Fetter
  - help miscellaneous things like git repository and this session
- David Rowley
  - many tests to catch early bugs, as well as spec investigations
- Takahiro Itagaki, Pavel Stehule, and All the Hackers
- I might have missed your name here, but really appreciate your help.