Overtaking CPU DBMSes with a GPU in Whole-Query Analytic Processing

Adnan Agbaria - now at Intel
David Minor - now at GE Research
Natan Peterfreund
Eyal Rozenberg - now a post-doc at CWI Amsterdam
Ofer Rosenberg - now at QualComm

Motivation

- **We all want to put discrete GPUs to work on analytics.**
- **Lots(!) of proof-of-concept systems in recent years**

GPUDB	CoGaDB	MapD
OmniDB	Virginian	Ocelot
Galactica	Red Fox	GPL

- **None of these systems exhibits the combination of:**
  - significant performance boosts,
  - for complete queries of varying complexity,
  - relative to state-of-the-art analytic CPU-based DBMSes.

Our contributions

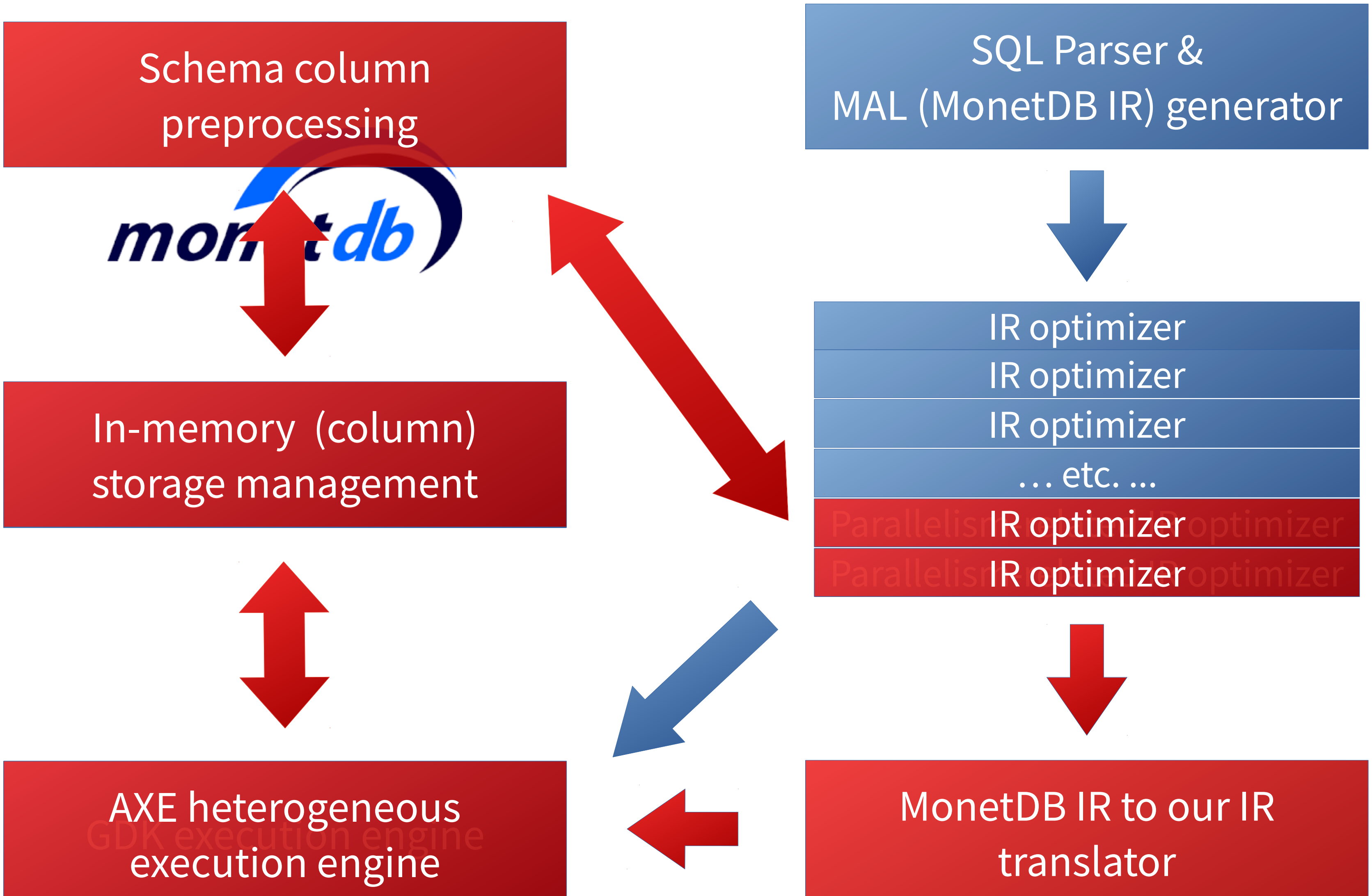
1. We present a first proof-of-concept GPU-based query processing framework exhibiting
   - significant performance boosts,
   - for a selection of TPC-H queries of varying complexity
   - relative to a FOSS state-of-the-art CPU DBMS.

2. We propose a different focus of the effort of squeezing performance out of discrete GPUs.

3. We indicate clearly-realizable potential for additional speedup.

Our system

System architecture



Execution Engine

- **Targeted at arbitrary data-processing-oriented GPU-utilizing applications.**
- **Not “domain-specific” knowledge of DBMS; not aware of concepts such as “tuple”, “table”, “column” etc.**
- **Supports, among others:**
  - CPU and GPU execution
  - task- and data-level parallelism
  - concurrent multi-device execution
- **Going into detail would require most of the remaining time we have.**
- **We even have some results on multi-GPU query execution, but those could not fit into the paper.**
- **It still has some “infancy issues”, such over-conservatism in synchronization.**

Schema preprocessing

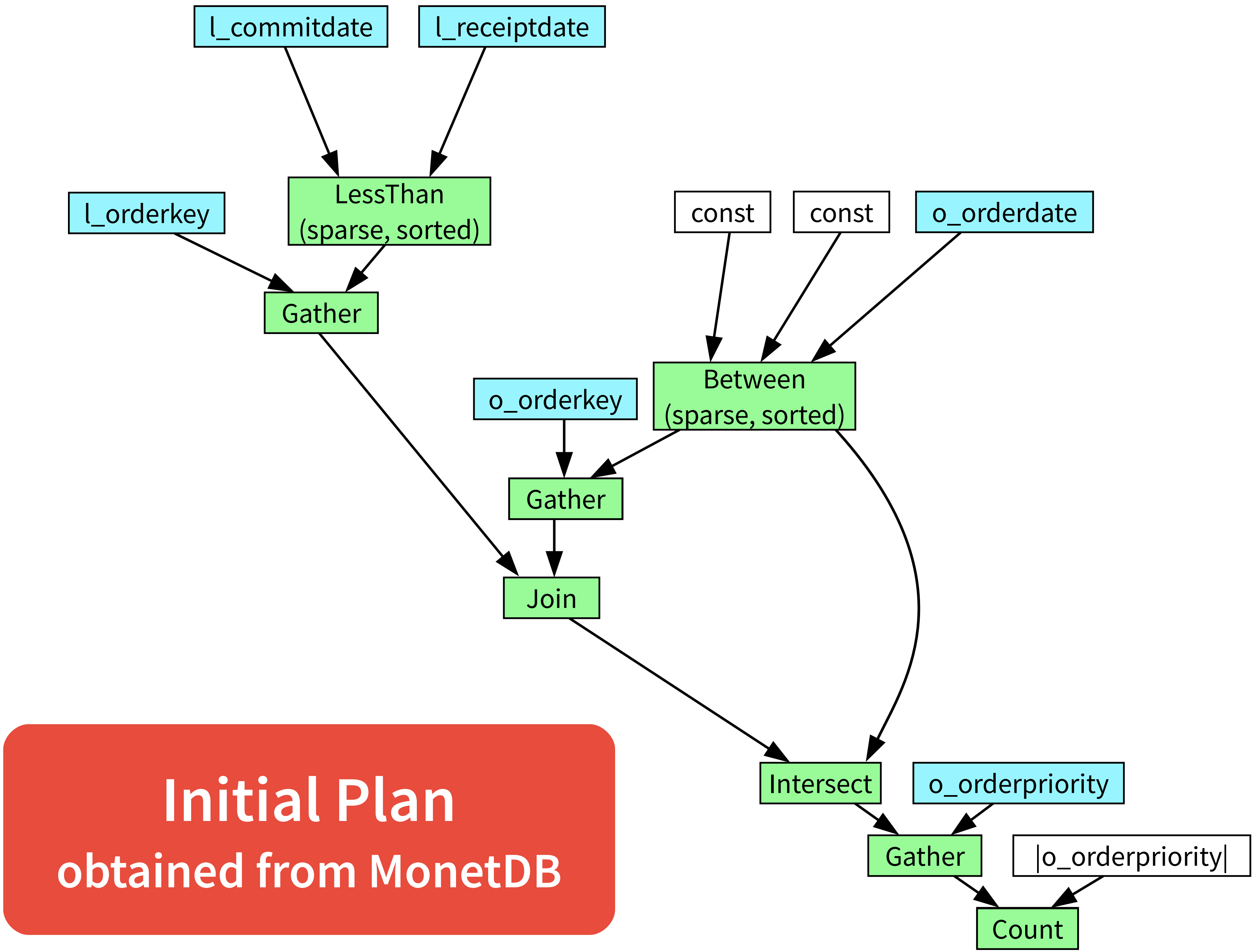
- **Generated at DB load time.**
- **No "cheating" – only producing what's allowed by TPC-H.**
- **Scalar precomputed data:**
  - min, max, mode, maximum multiplicity, support size etc.
- **Columnar precomputed data:**
  - Distinct values in order of appearance
  - First and last appearances of all distinct values, etc.
- **Not a free lunch: this has a cost in memory footprint.**

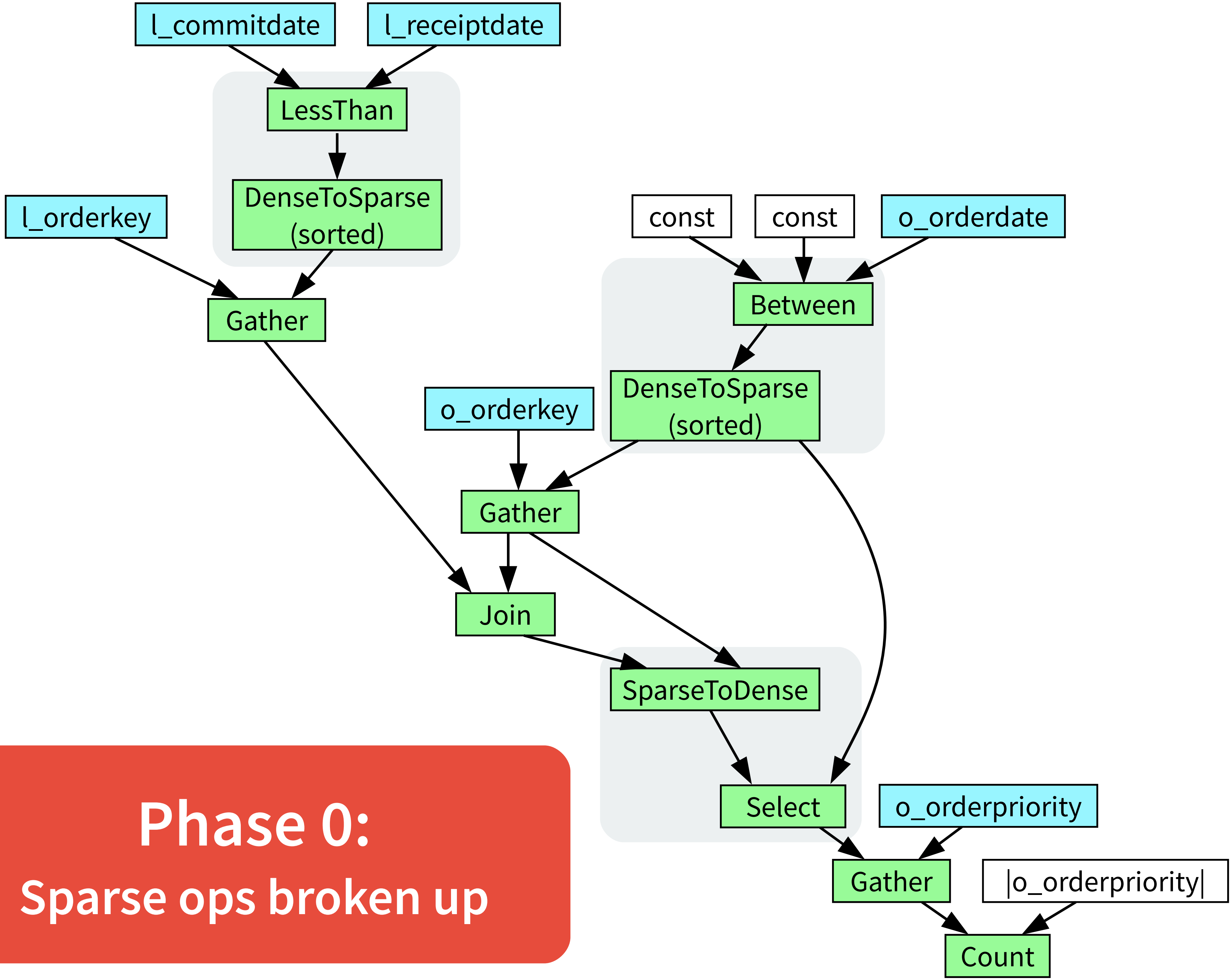
Let's make a GPU-friendly execution plan!

… for TPC-H Q4:

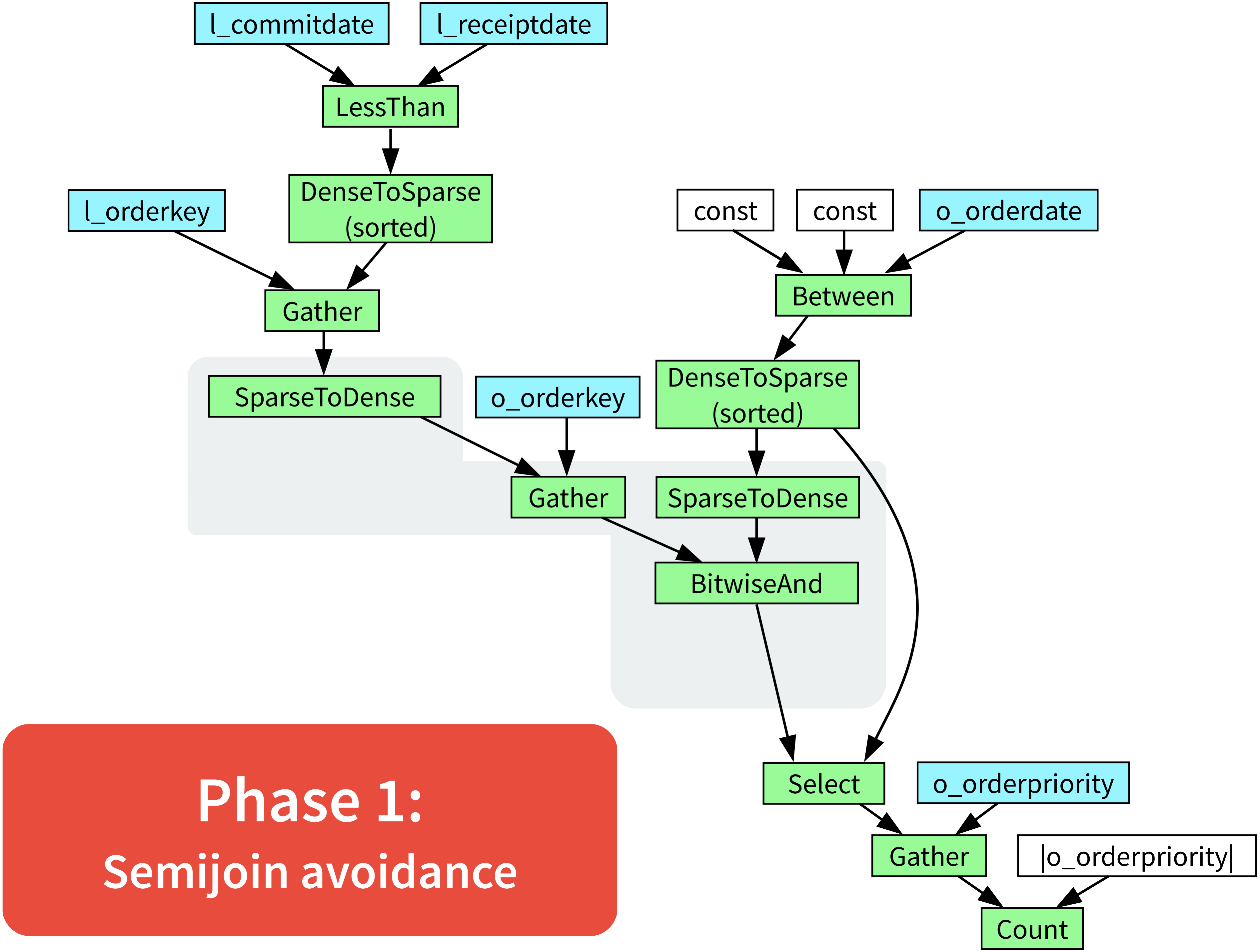
```
select count(*) as order_count
from orders
where o_orderdate >= date '1993-07-01'
and o_orderdate   <  date '1993-07-01' +
   interval '3' month
and exists (
    select * from lineitem
   where l_orderkey = o_orderkey
   and l_commitdate < l_receiptdate)
group by o_orderpriority
order by o_orderpriority;
```

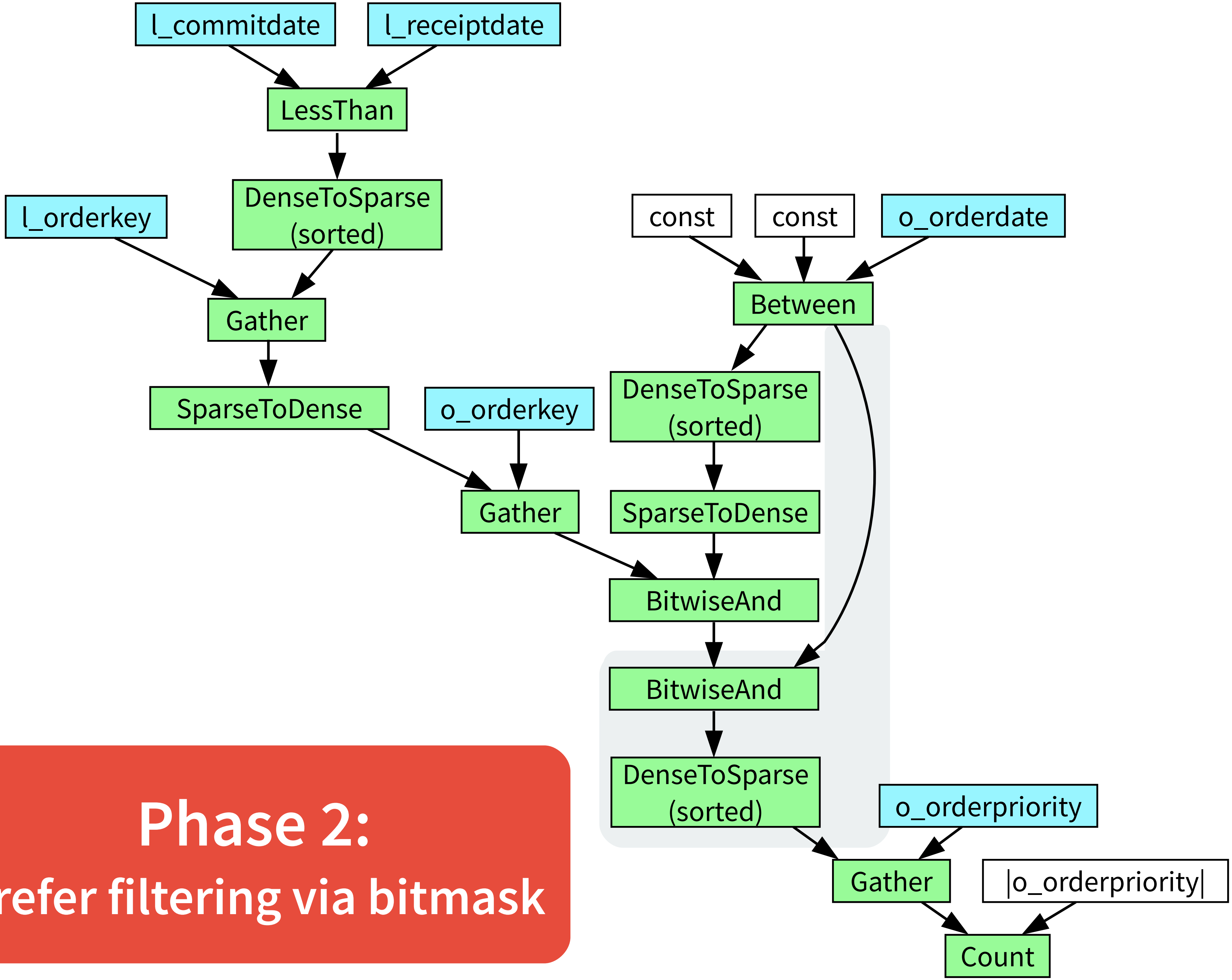
… without the string column (so that we fit on the slide)


l_commitdate
l_receiptdate
LessThan
(sparse, sorted)
l_orderkey
const
const
o_orderdate
Gather
Between
(sparse, sorted)
o_orderkey
Gather
Join
Intersect
o_orderpriority
Gather
|o_orderpriority|
Count
Initial Plan
obtained from MonetDB

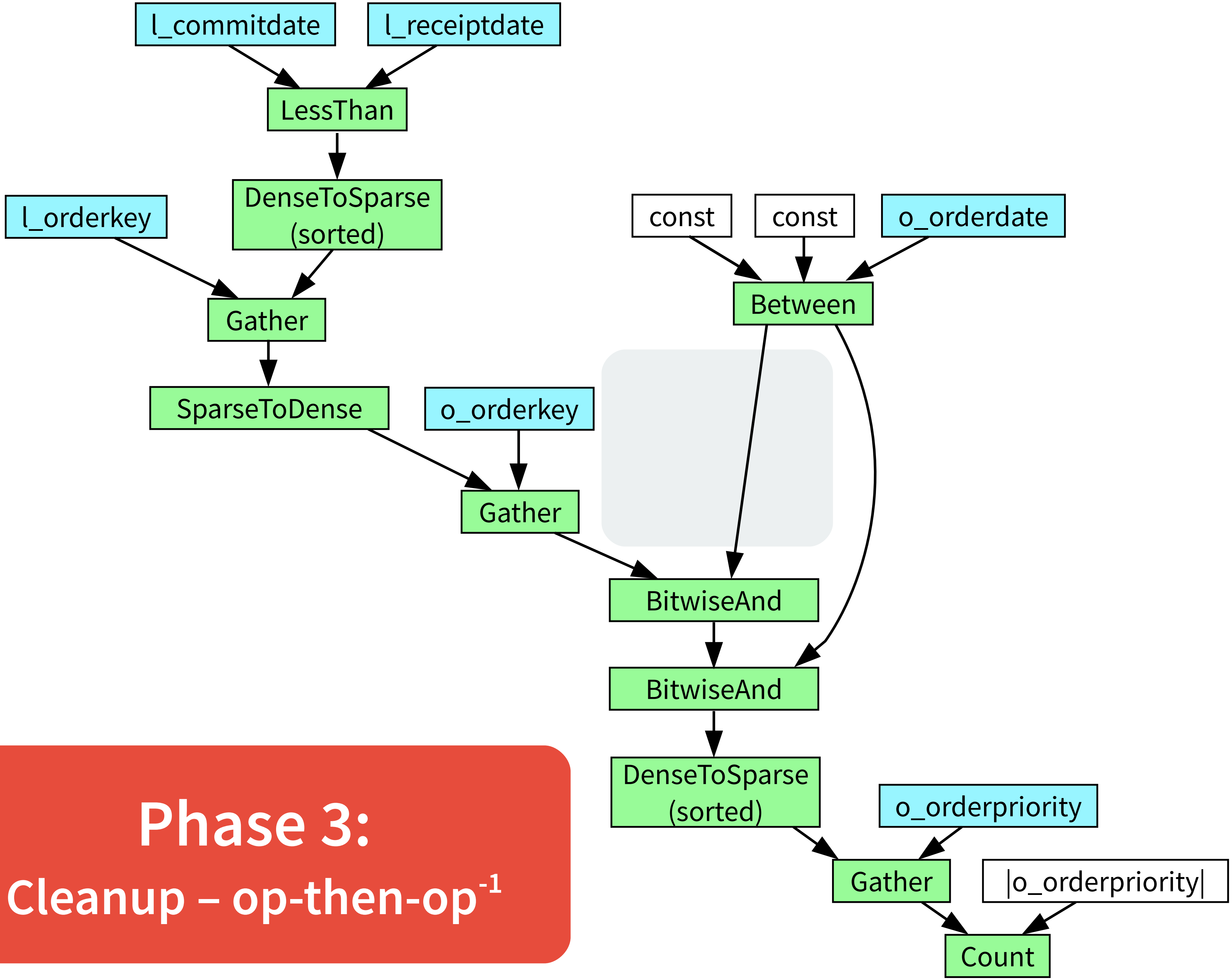


Phase 0:
Sparse ops broken up


l_commitdate
l_receiptdate
LessThan
DenseToSparse (sorted)
l_orderkey
Gather
const
const
o_orderdate
Between
SparseToDense
o_orderkey
DenseToSparse (sorted)
Gather
SparseToDense
BitwiseAnd
Select
o_orderpriority
Gather
|o_orderpriority|
Count
Phase 1:
Semijoin avoidance

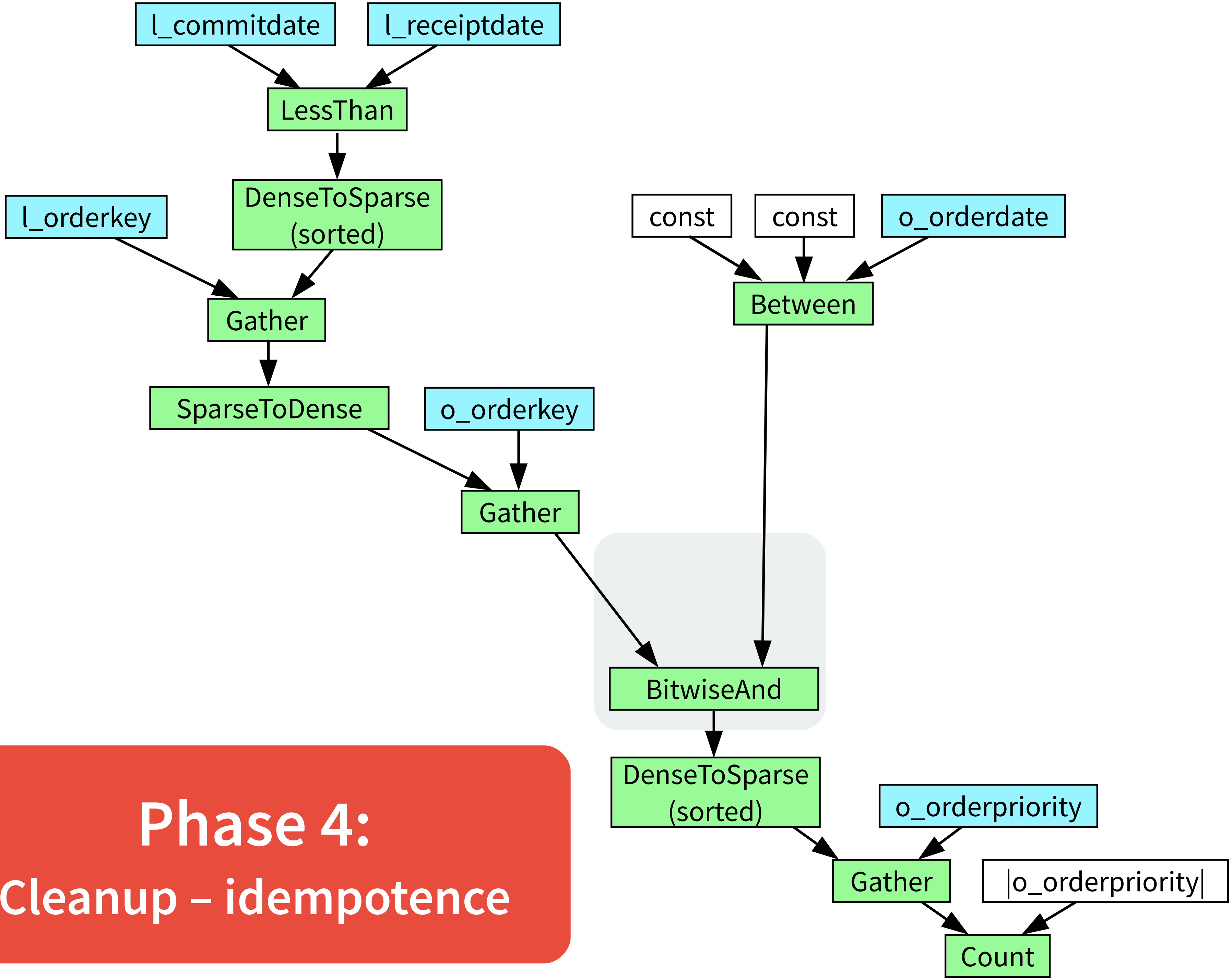


Phase 2:
Prefer filtering via bitmask



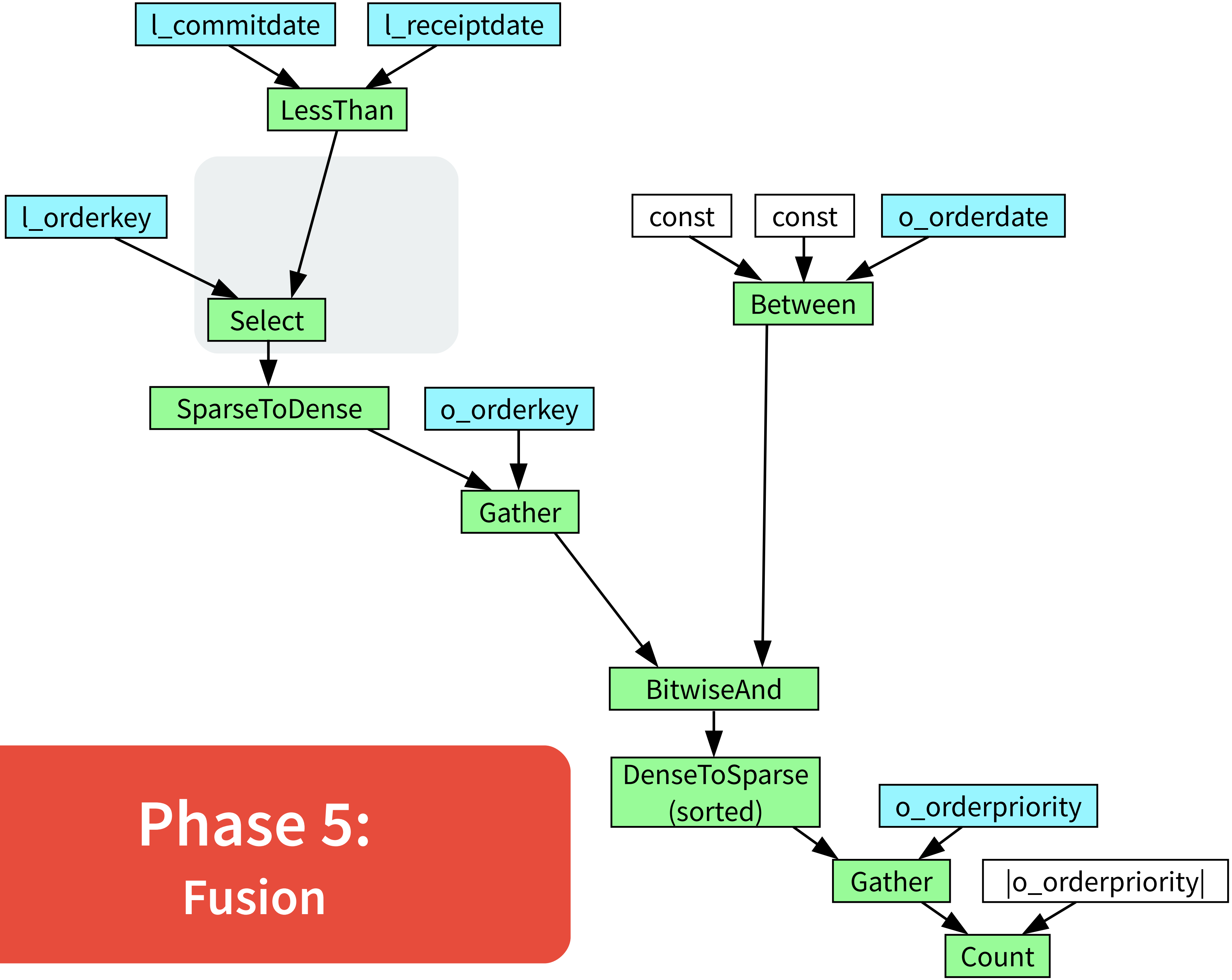
Phase 3:

Cleanup – op-then-op^{-1}



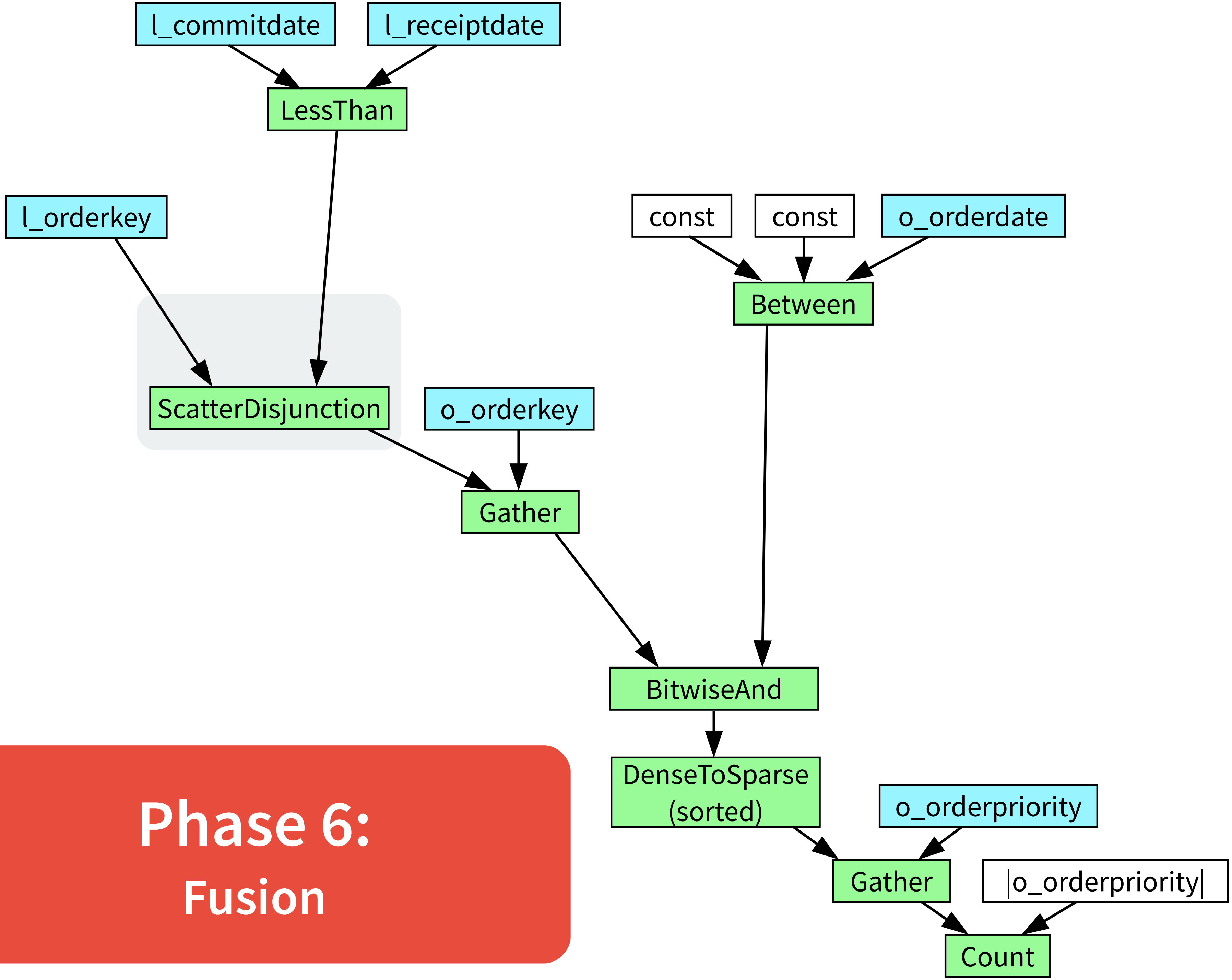
Phase 4:

Cleanup – idempotence

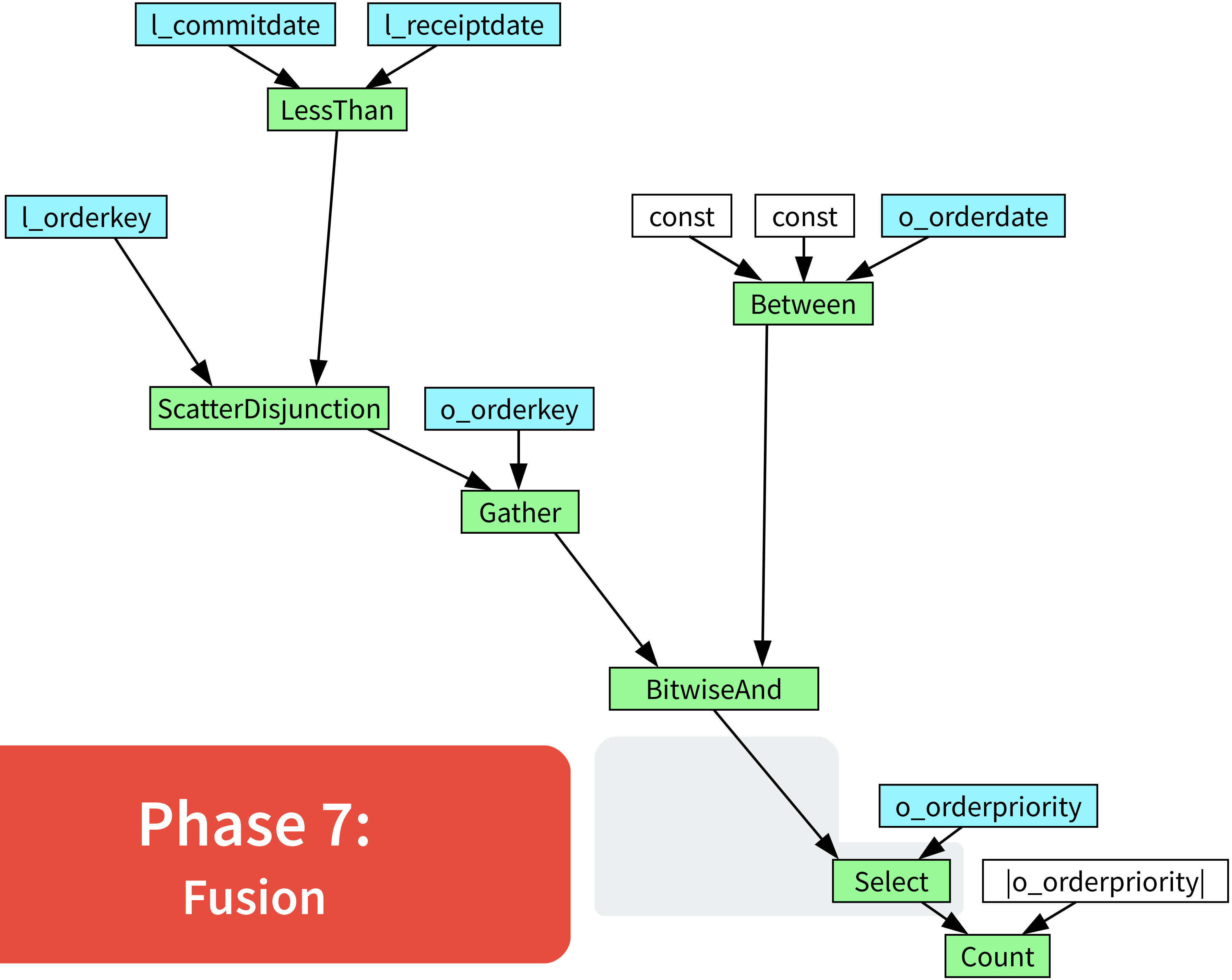


Phase 5:

Fusion

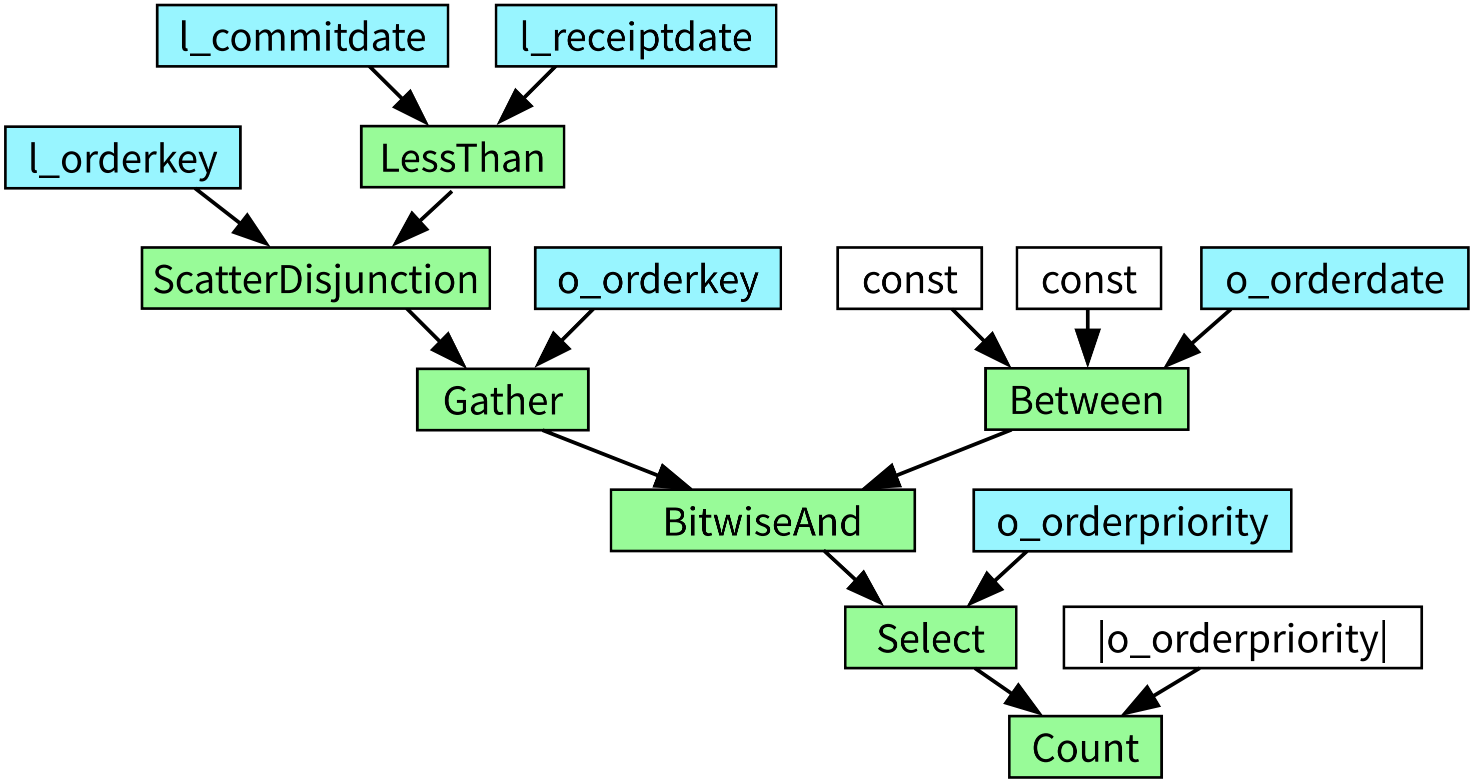


Phase 6:
Fusion



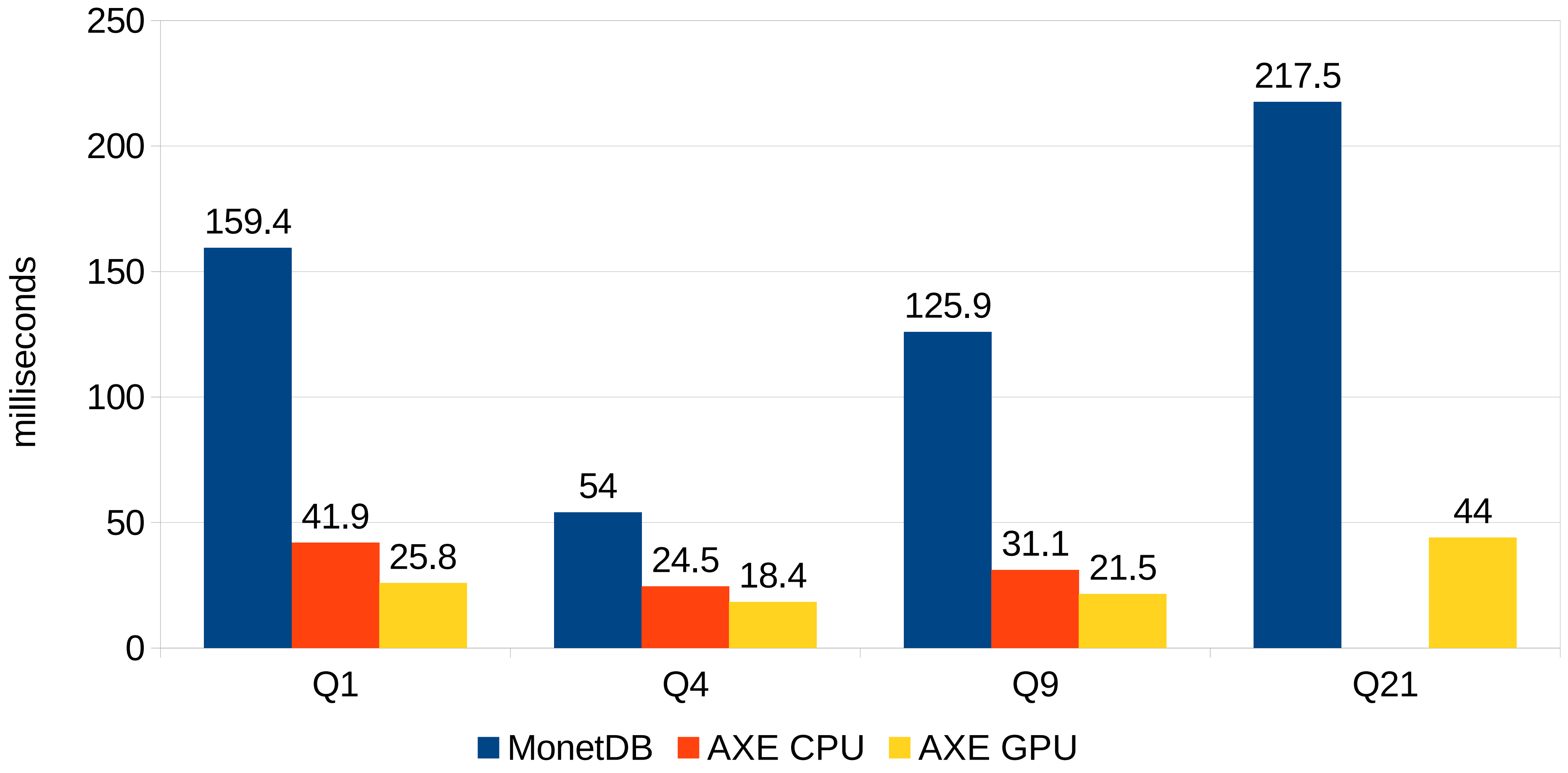
Phase 7:
Fusion

Let's make a GPU-friendly execution plan!



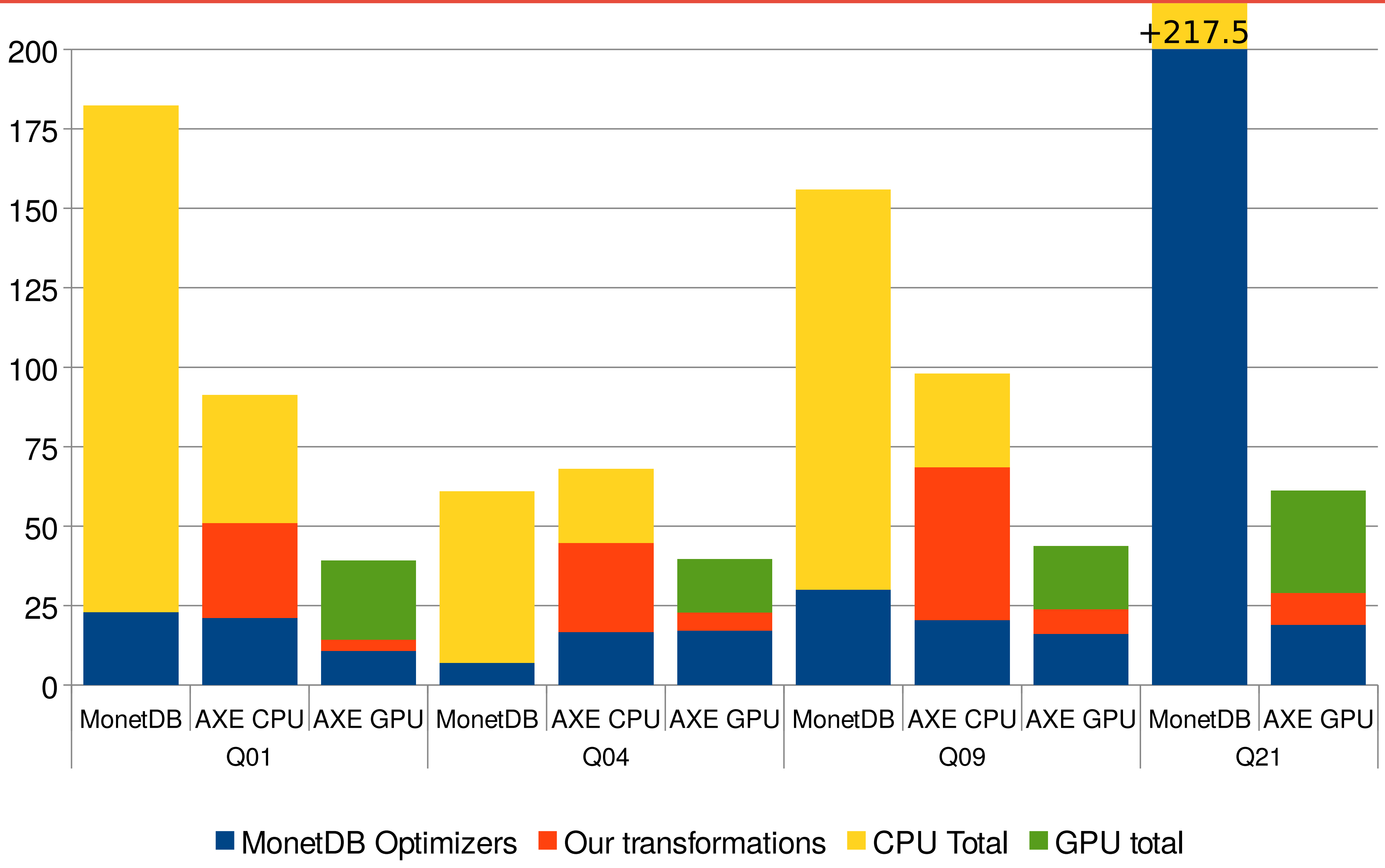
Experimental Results

The "Bottom Line" – Plan execution time

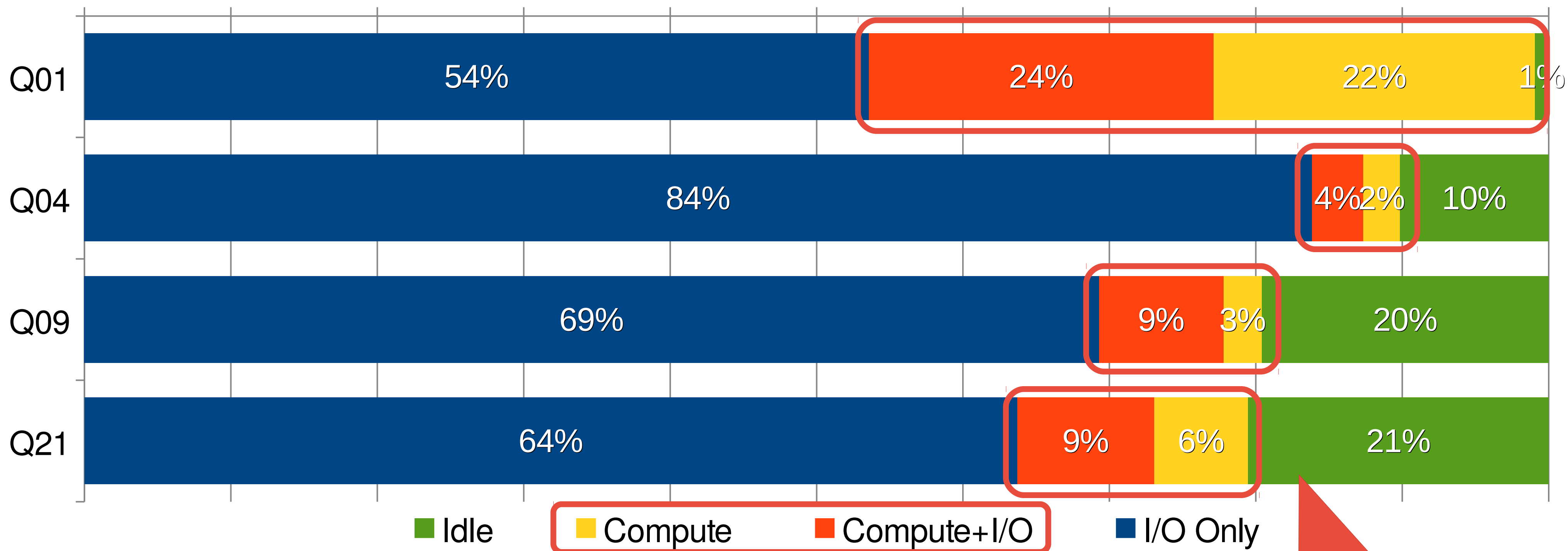


- Caveat: Not the latest MonetDB (v11.15.11 vs v11.23.7)
- These figures are somewhat misleading. Let's have a closer look...

Query processing time breakdown



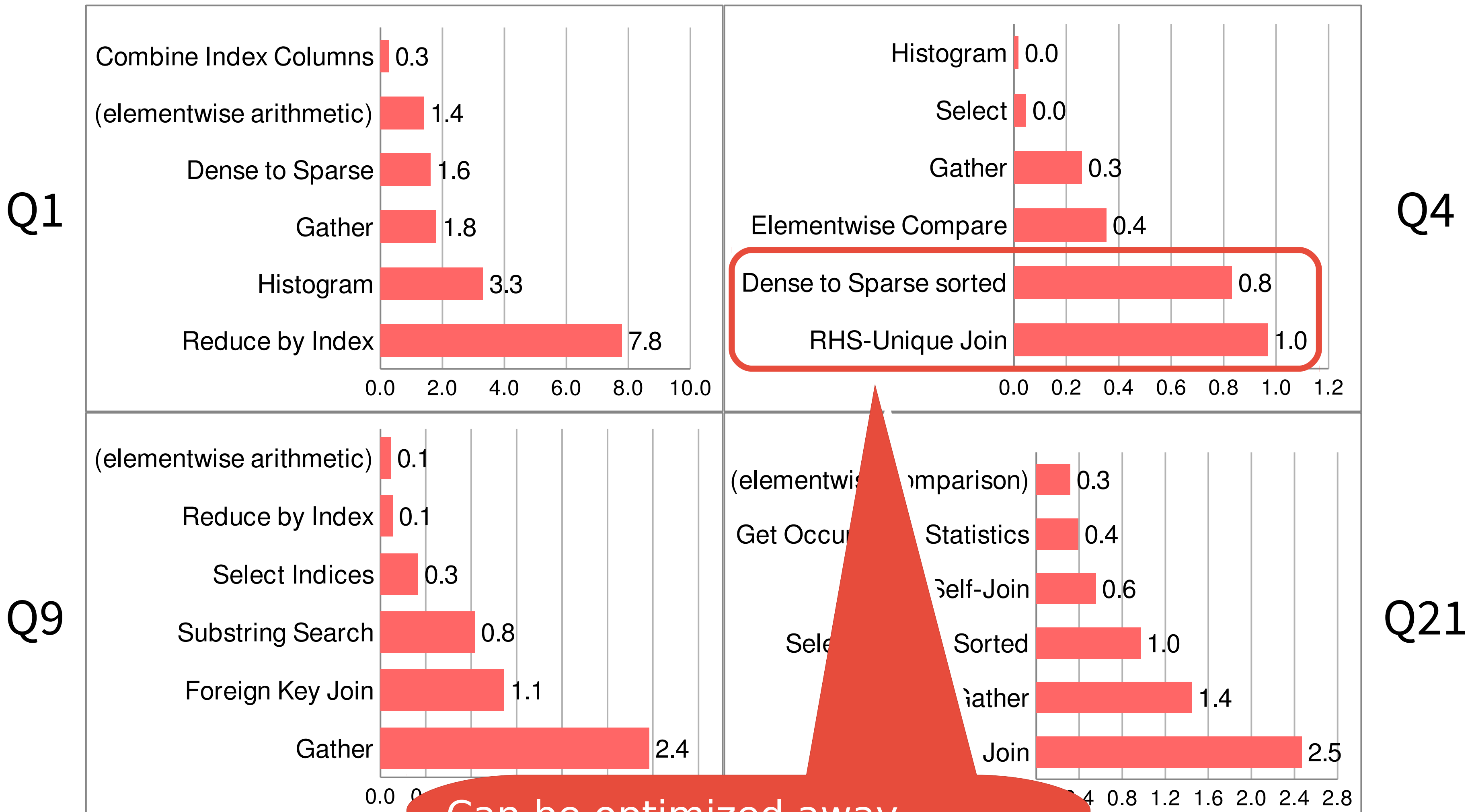
GPU Compute/IO activity breakdown



- The Dreaded PCIe bandwidth bottleneck rears its
- Pipelining/chunking initial operations would ofte
- Idle time - mostly artifacts of our implementation

Except for the "muscular" Q1, the GPU is doing work only for 6%-15% of the total time

GPU Compute Time Breakdown



Top 6 time-consuming c… …tions in msec)

Can be optimized away in favor of a Gather, a weird op (ScatterDisjunction) and elementwise logical ops.

Effect of scaling on query processing time breakdown

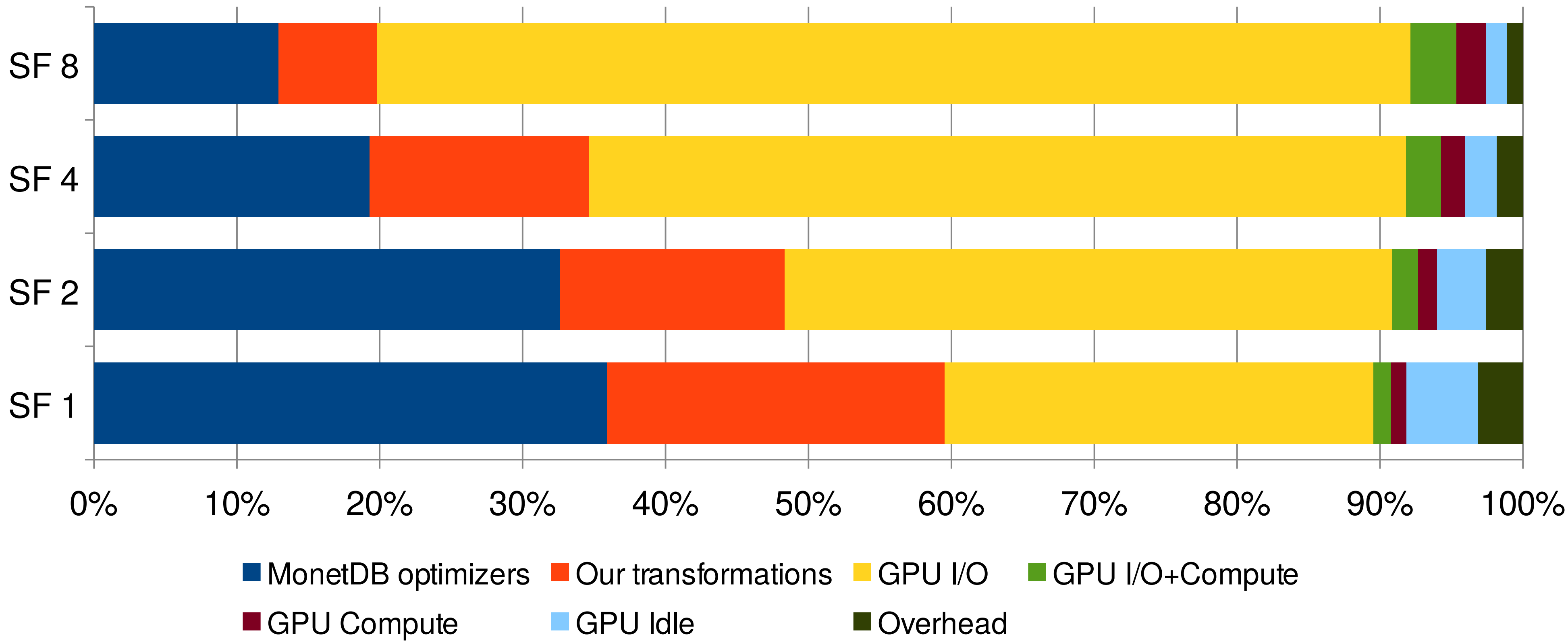


Chart regards TPC-H Q4.

Reflection, Analysis, Shortcomings etc.

Reflection, Analysis, Shortcomings, etc.

Q: Why only (these) four queries?

- Management decision re scope of our work.
- Sort-of-representative of the gamut of TPC-H queries.
- Our existing design would support all or almost-all of them.

Q: Why CUDA rather than OpenCL?

- Faster to develop with, more convenient and flexible.
- OpenCL had not yet caught up when we started (C++, templates).
- nVIDIA is impeding OpenCL adoption by holding back on 2.x support on their cards… despite being members of Khronos. For Shame.

Reflection, Analysis, Shortcomings, etc. (cont...)

Q: How come you spend so much time on I/O, and so little on Compute?

- Actually the result of a successful coding effort: at first, it was the other way around.
- A manifestation of the 'Yin-Yang principle' [LZ'13].
- The way to *really* address this issue is compression.

Q: But surely you could at least make the "Compute Only" regions overlap the "I/O only"?

- Not really, since these are based on intermediary results.
- Execution in chunks [BC'12, JHH'16] can help some.
- So can GPU-mapped memory.

Reflection, Analysis, Shortcomings, etc. (cont...)

Q: Why did you use low scale factors so much?

A combination of two shortcomings:

- No execution in chunks (so - materializing entire columns)
- Had not yet implemented a slab memory manager.

Q: Can I get the source code?

- **No :-(. In fact, it has probably been shelved forever, since our (former) group has been sort-of disbanded.**
- **But you can get *me*: I need collaboration to take this approach to the next level – with release-quality FOSS code. For now, I'm working at it alone – but that's too slow.**
- **Some source code from my FOSS efforts already available on request.**

Comments? Questions? Craving some ornate C++?

E.Rozenberg@cwi.nl