Towards an Efficient Linear Algebra Based OLAP Engine

João Afonso - 2017/2018

Useful links:

RPD

Dissertation

Tech Report (PI)

RPD presentation

PODS_PAPER

Incrementality

Paper Perform Increm VLDB14

The main challenges

- Plan and develop LA based OLAP engine, featuring:
  - Low latency (query time)
  - High throughput (queries/time)
  - Horizontal scalability
  - Schema free data access
- To ensure some of this topic we rely on:
  - Columnar data storage and processing
  - Well defined LA properties

Goals for the meeting

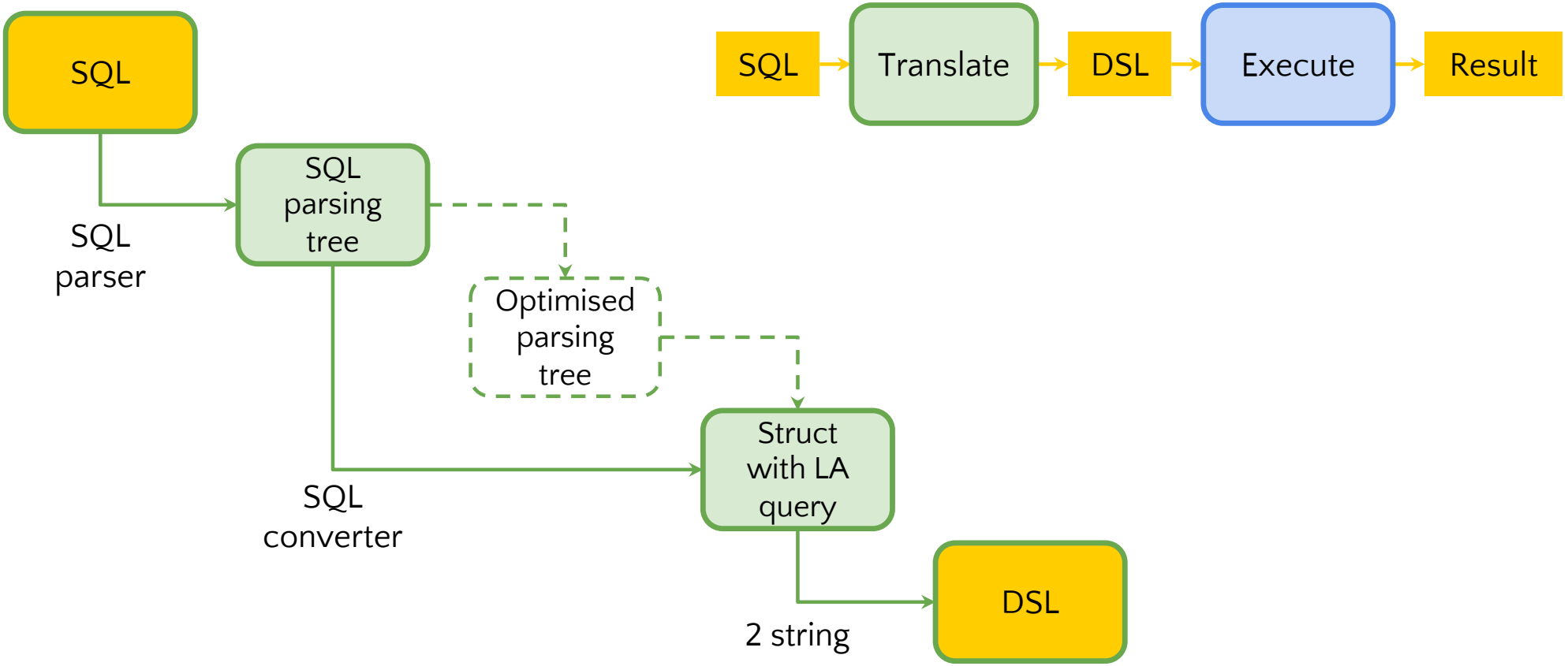
12 Oct

- Study implemented systems (Apache, Amazon, Google, ...)
- First considerations on the system architecture
- Study sparse matrix representations:
  - Read the paper suggested by Prof. Rui Ralha
- Optimise CSC/COO memory usage
- Start learning C++

Implemented systems

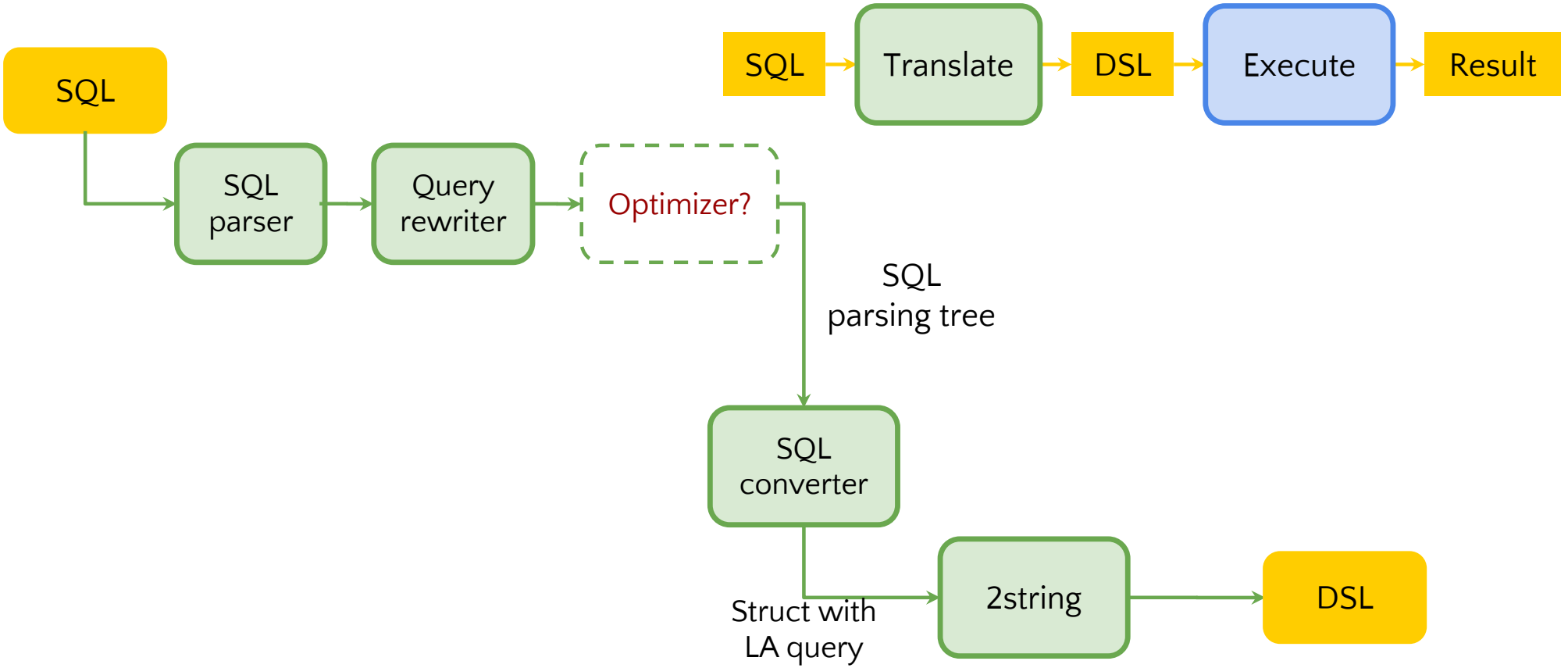
- Google provides database solutions considering both **usability**, **efficiency** and **scalability**
- Apache **HBase** is based on Google **BigTable**
- Apache **Drill** is based on Google **BigQuery**
- **EDIT: Impala, Hive + Tez/Pig**
- Both should be considered in the final system design!

Possible architecture

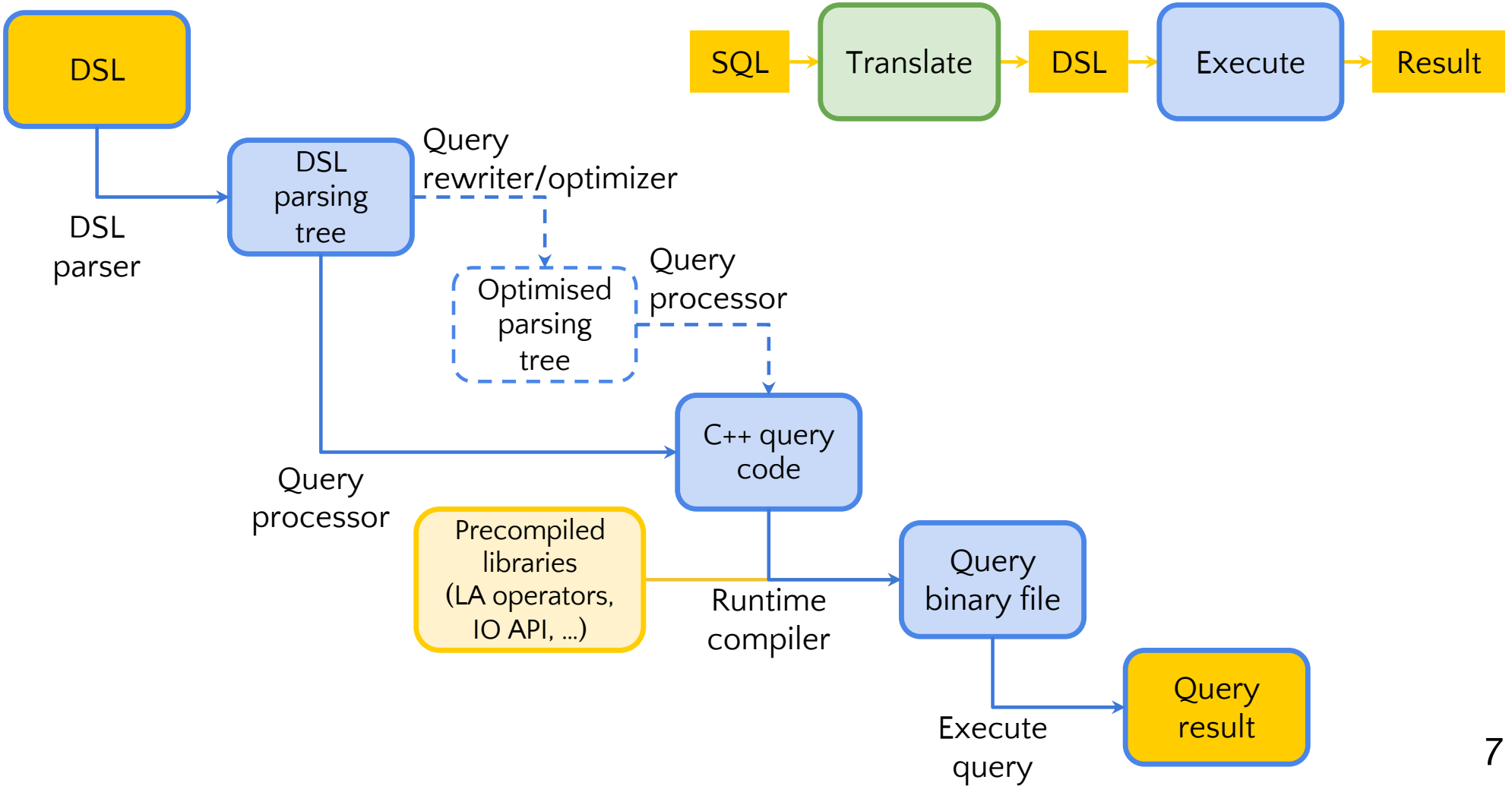


Too soon for a functional implementation!

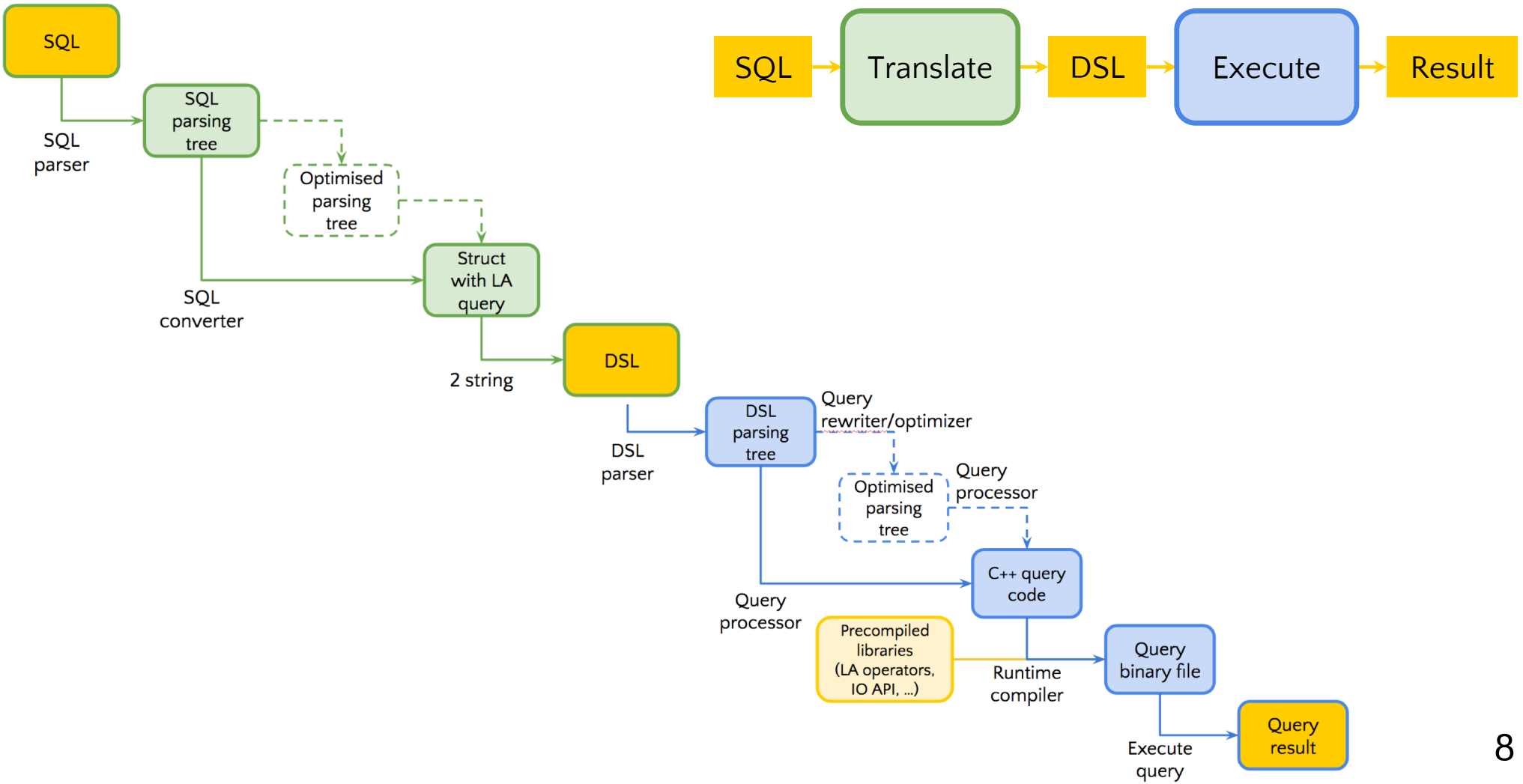
Possible architecture



Possible architecture



Possible architecture



Sparse matrix formats

- Previously studied formats: COO, LIL, CSR, CSC
- CSC can compress data when **nnz > n**
  - We have **nnz <= n** so we are wasting space
- CSB – Compressed Sparse Blocks:
  - Compressed representation for blocks
  - Discarded by having **dense outer matrix** and **binary search** inside blocks

CSC memory usage

- CSC format:
  - **Values** [nnz]
  - **Row_index** [nnz]
  - **Col_pointer** [n cols]

- Depending on the matrix to be represented, some of them may be redundant:

CSC/COO memory usage

	Values	Rows	Columns
Decimal Vector			
Bitmap			
Filtered Bit Vector			
Decimal Map			
Filtered Bitmap			
Filtered Decimal Vector			
Filtered Decimal Map			

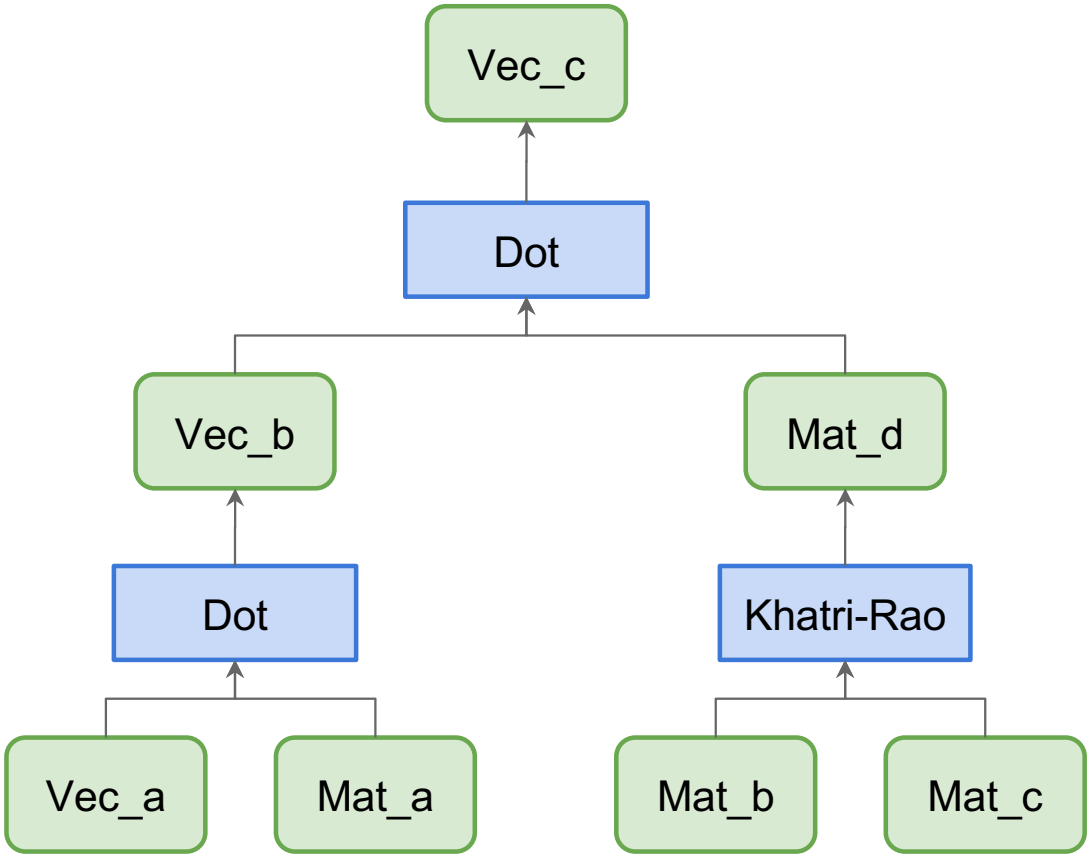
Meeting with professor Rui Ralha

16 Oct

- Considerations on sparse matrix representations:
  - Case of study
  - Matrix format (data)
  - CSC and COO
  - CSB

Case of study

- What are we trying to optimise?
  - A parallel flow of data (typically petabytes) through the operation tree (sequential or parallel)
  - Minimize the temporary data storage and waiting times.



Matrix format (data)

- Brands: Ford, BMW, Ford, VW, Audi, VW, Ford, Audi;
- Brand ≠ VW (matrix representation):

Ford	1		1				1	
BMW		1						
VW								
Audi					1			1

- *At most one* element per column (**partial** function)
- The distribution of the non-zero rows is unpredictable

CSC and COO

Ford	1		1				1	
BMW		1						
VW								
Audi					1			1

4x8

- CSC representation:

```
Values: [1, 1, 1, 1, 1, 1]
Rows:   [0, 1, 0, 3, 0, 3]
Cols:   [0, 1, 2, 3, 3, 4, 4, 5, 6]
```

- COO representation: 3x6

```
Values: [1, 1, 1, 1, 1, 1]
Rows:   [0, 1, 0, 3, 0, 3]
Cols:   [0, 1, 2, 4, 6, 7]
```

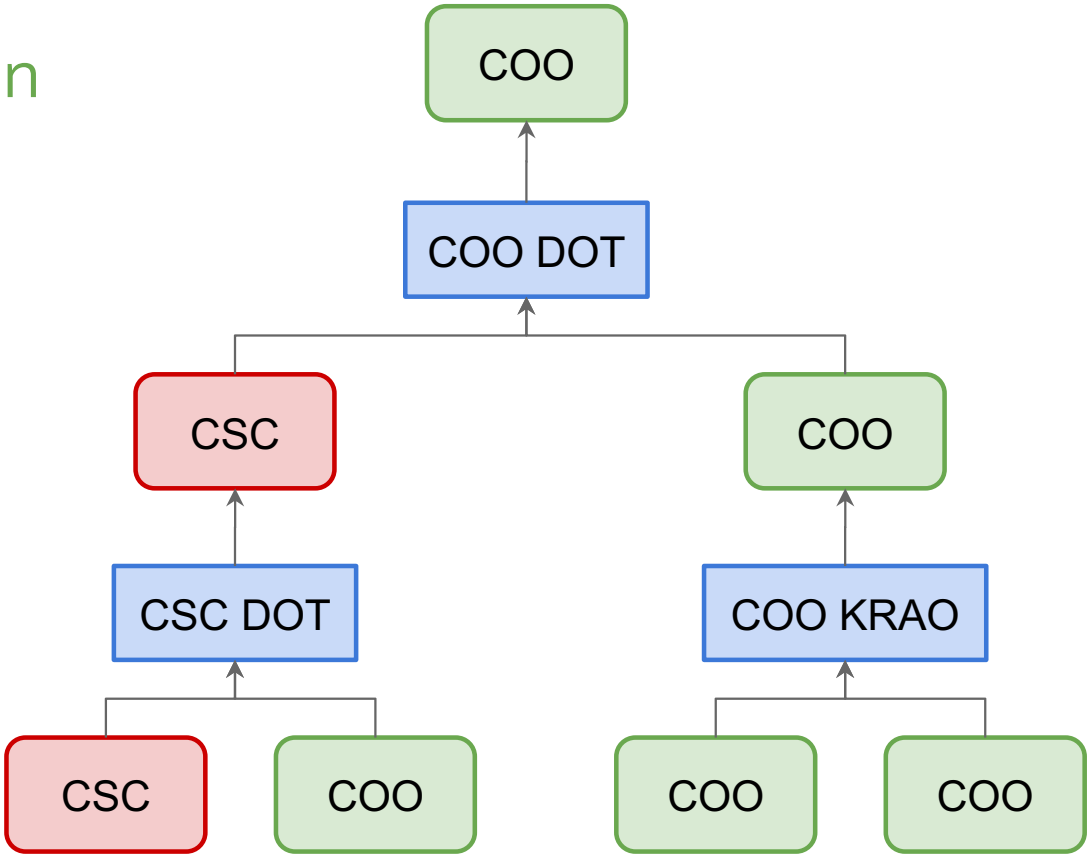
In this ex compacting saves 44% ...

CSC and COO

- CSC:
  - Can be iterated through the non-zero values
  - Can't be iterated row-wise
  - Θ(1) column access cost
- COO:
  - Can be iterated through the non-zero values
  - Can't be iterated row-wise
  - Θ(log(n)) column access cost (binary search)

CSC and COO

- Θ(1) left matrices column access cost
- Compressed column pointer array on COO matrices
- Two different matrix formats
- Two versions of each operator



CSB

- CSB:
  - Dense outside matrix pointing to COO blocks
- There is any safe way to define a maximum size (in bytes) for:
  - each individual block?
  - the outer matrix?
- How can the maximum number of columns can be calculated?
- LINK TO PAPER

Meeting remarks

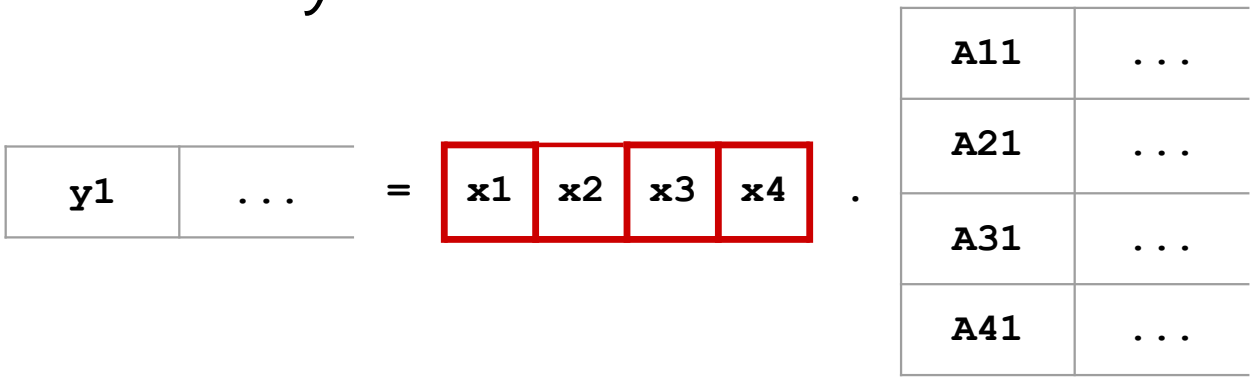
- Matrix format
- Data stream operations
  - Dot product dependencies
- Parallel dot product
  - Data ring
- Bibliography

Matrix format

- CSB is not suited to our study case
- CSC/COO as a possible approach
  - Khatri-Rao and Hadamard can use COO both sides
- Professor Rui Ralha will further investigate an adequate representation for functional sparse matrices

Stream dot product

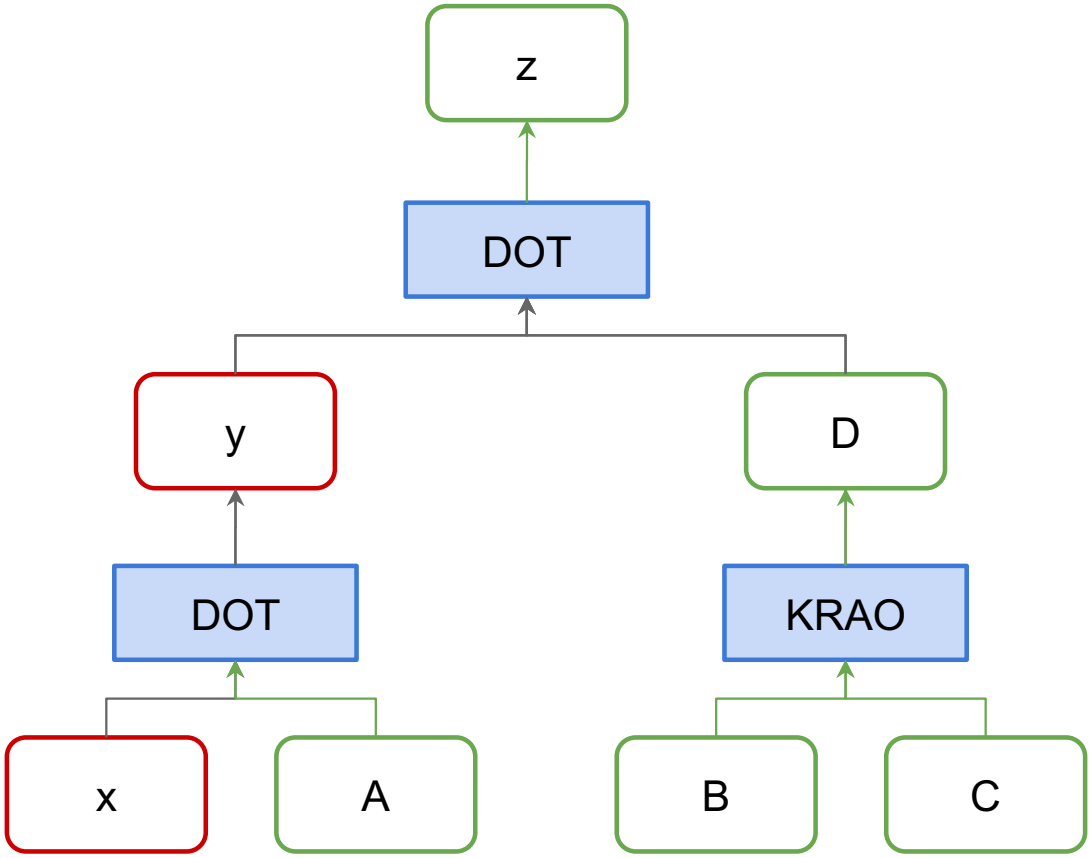
- Consider y = x . A
- Divide work by columns “i” of A:



  - y[i] = x . A[i] ; **x must be in memory!**
  - Same for C[i] = A . B[i] ; A must be in memory!
  - **EDIT: This is a broadcast JOIN**

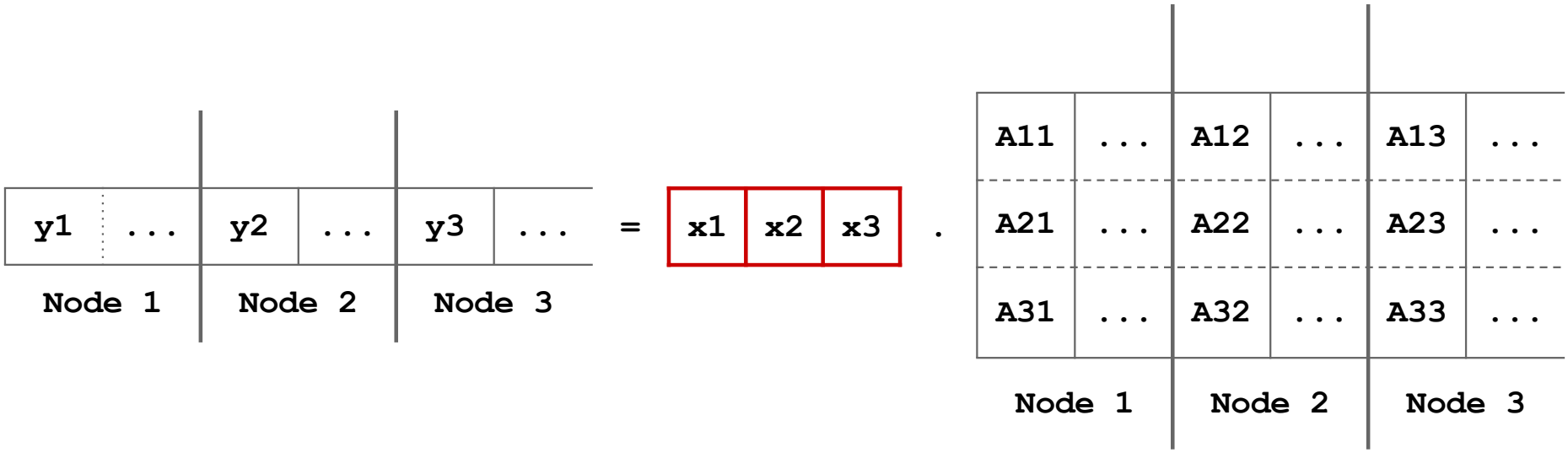
NB: Cf. fusion–+ law (8) of *“A Linear Algebra … to OLAP”* (JNO)

Stream operation tree



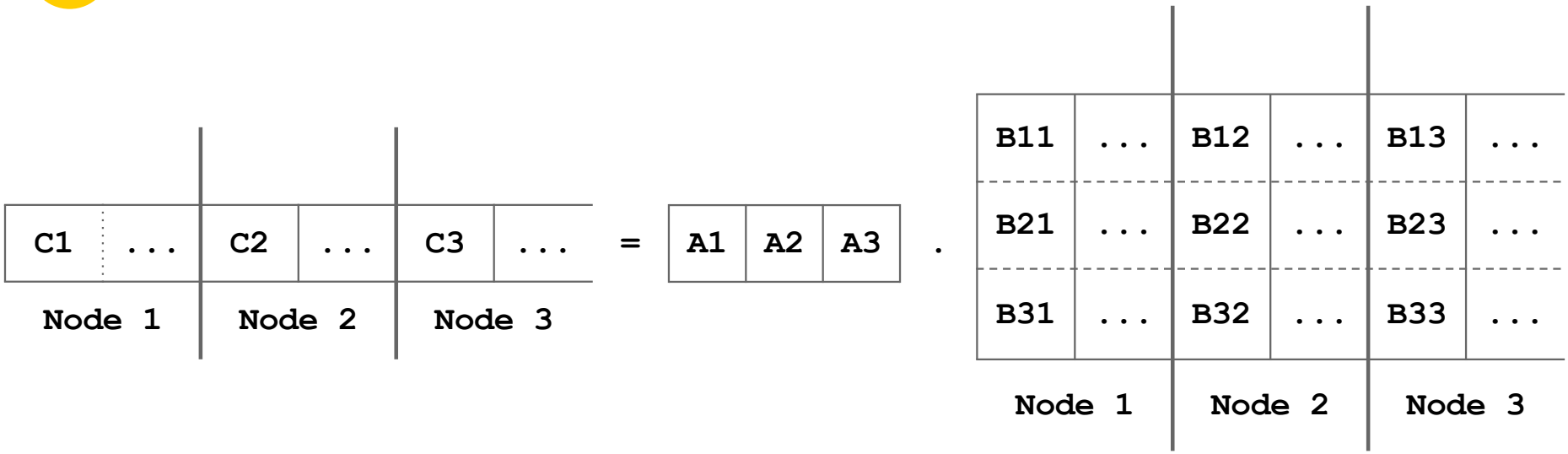
Parallel stream dot product

y: 1 ← ∞ **x:** 1 ← 3 **A:** 3 ← ∞



What if x does not fit in memory?

Parallel stream dot product



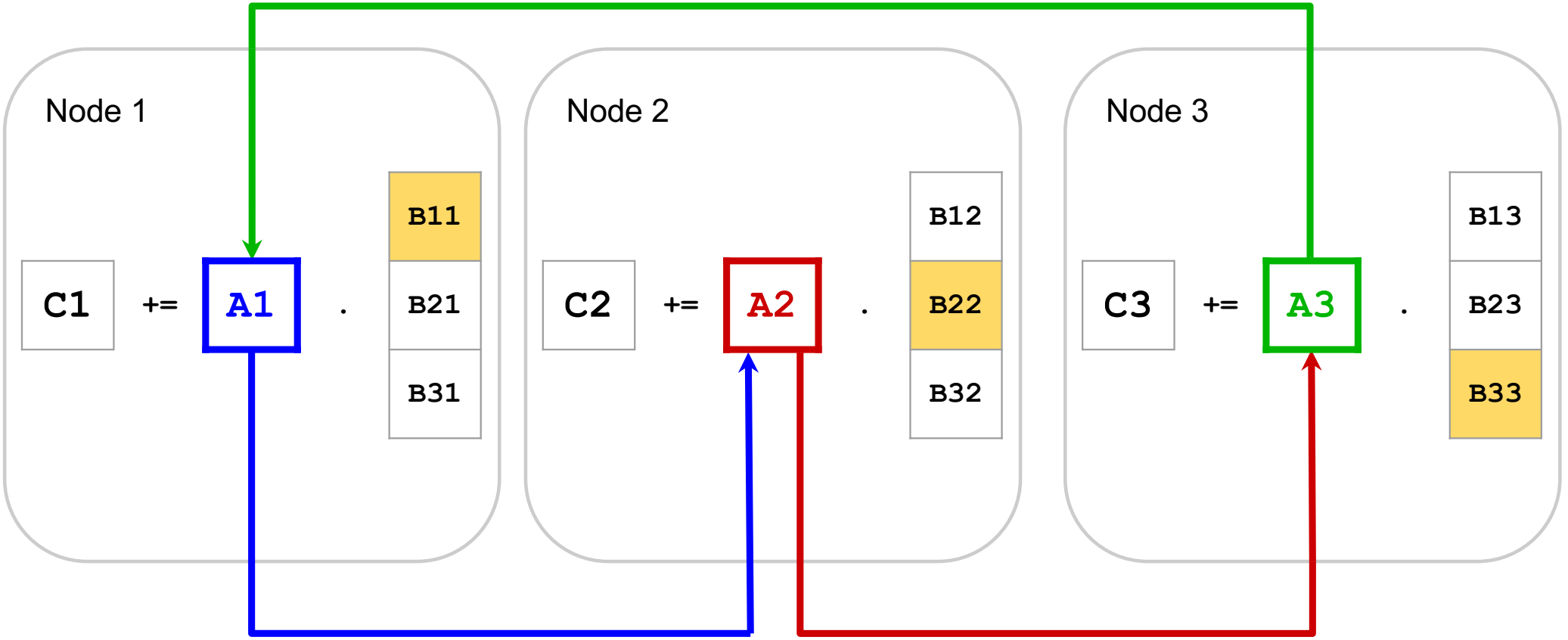
y1 = x1.A11 + x2.A21 + x3.A31

y2 = x1.A12 + x2.A22 + x3.A32

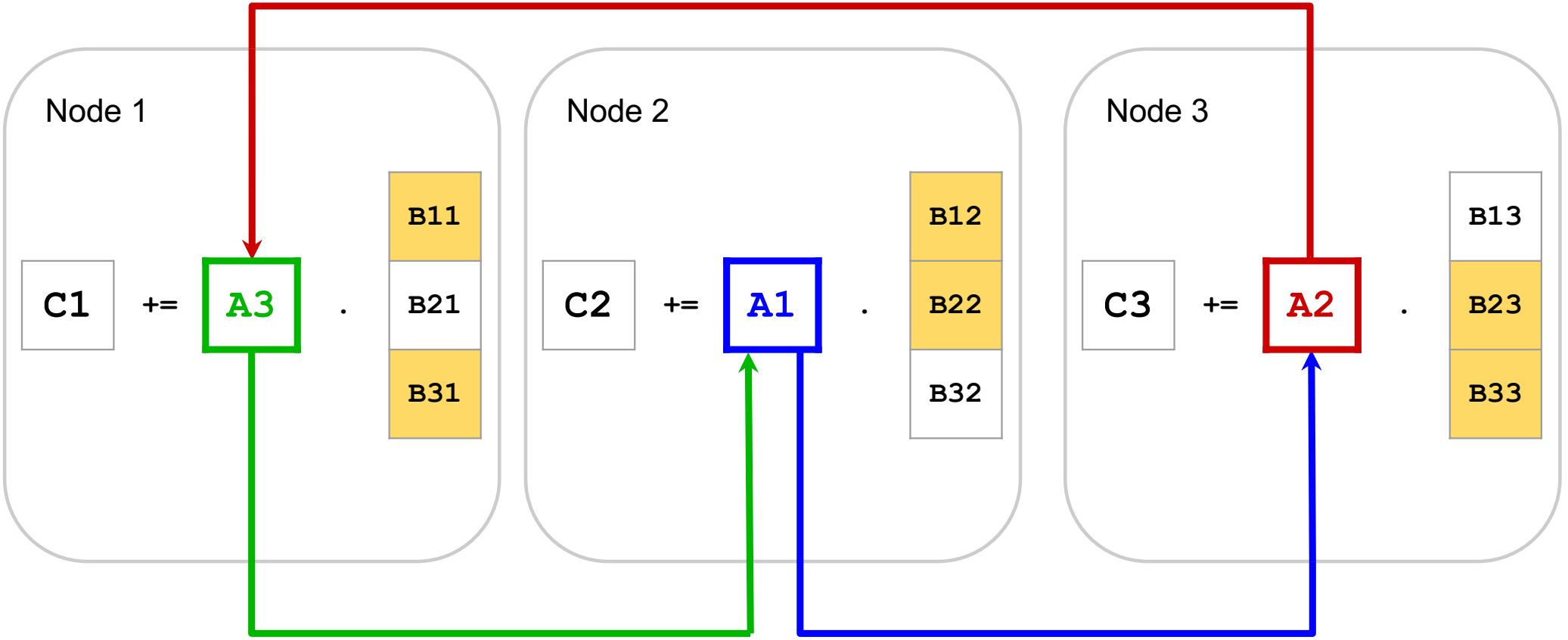
y3 = x1.A13 + x2.A23 + x3.A33

Scatter x across all nodes!

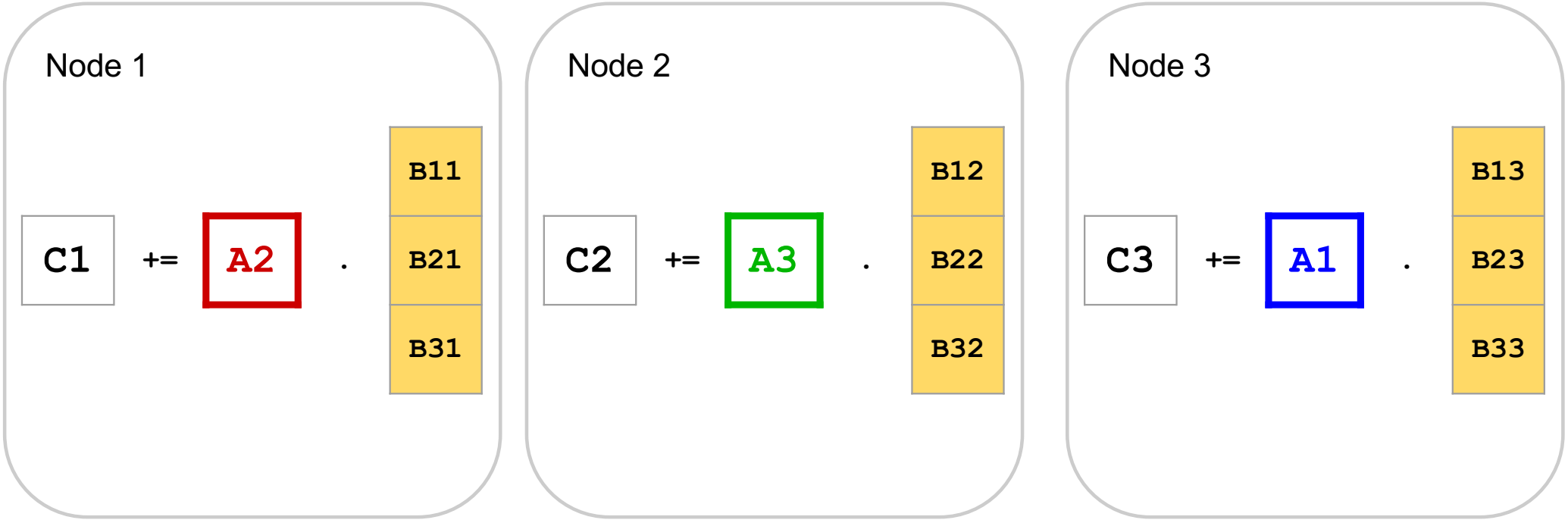
Parallel stream dot product



Parallel stream dot product



Parallel stream dot product



Parallel stream dot product

- Problem – we either have:
  - Both matrices in memory = large memory usage
  - Huge number of rotations = communication overhead.
- Possible solution:
  - Instead of rotating the left matrix, each node creates a thread to send data on demand (only the necessary values are passed).

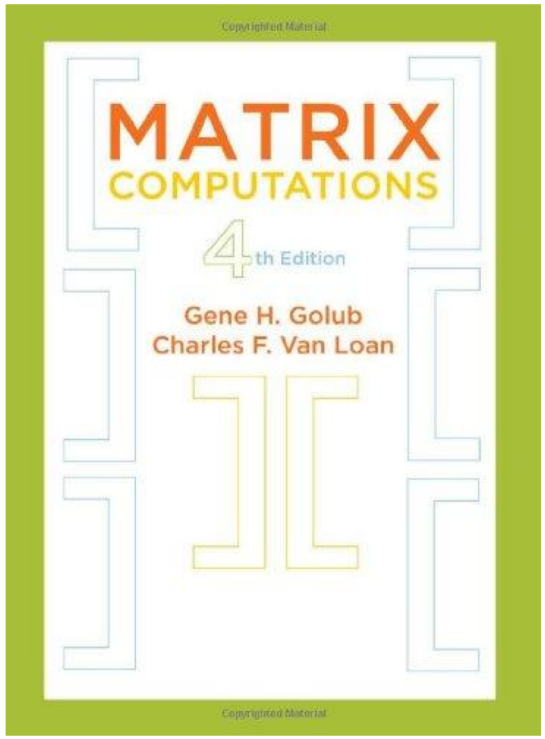
Parallel stream dot product

- Other possibility
  - Rotate small blocks / elements of the right side matrix instead of all the left one
  - Problems:
    - **C** must be reduced (MPI all to all + multi-merge)

Parallel stream dot product

- If the result of dot product is passed to a *bang()*, it can benefit from the horizontal partition!
  - Does it happen frequently**?**
- The *bang* also has dependencies that must be considered to build the operations tree
  - To be studied !!

Designing parallel solutions



Click on the image to open the book

Goals for the meeting

19 Oct

- Implement DSL parser
  - Name the DSL?
- Meeting with professor Rui Ralha
  - See slides above

DSL Parser

- Current state
  - Lex regular expressions are done
  - Grammar is written
  - Lost C functional version (C++ conversion is not trivial)
- To do
  - C++ version of the parser
  - Create list of operations
  - Create operations dependency tree based on the list
  - Reorder operations based on the previous concepts

DSL Name

- File extension website
- Suggestions:
  - LAQ – ‘LA Query’ language
  - LAS – ‘LA Scripting’ language

Goals for the meeting

26 Oct

- Problems of a block approach
- Matrix Transpose
- Matrix Vectorization
- Topics to cover in the dissertation's state of the art

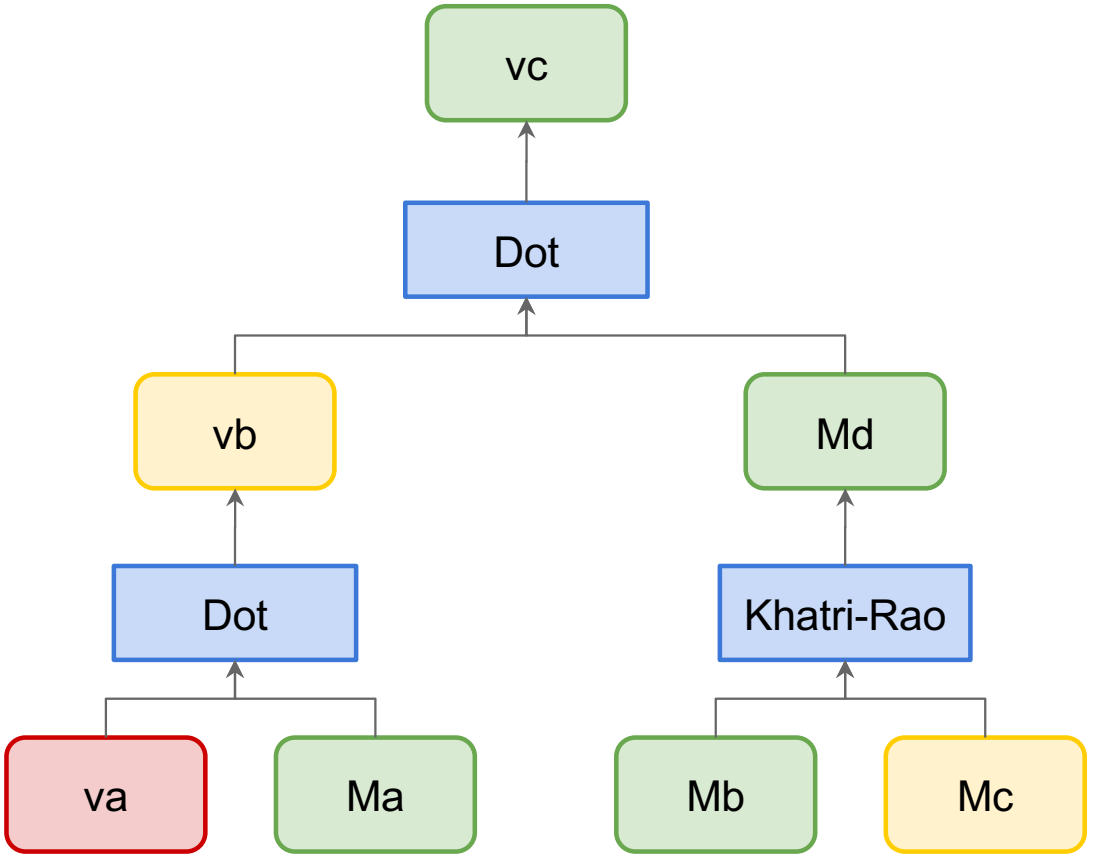
Problems of a block approach

- `va` in-memory
- `vb[i] = va . Ma[i]`

- How can we calculate `vc` streaming both `vb` and `Md`?

  `vc[i] += vb[j] . Md[i][j]`

  `Md` must be iterated row-wise



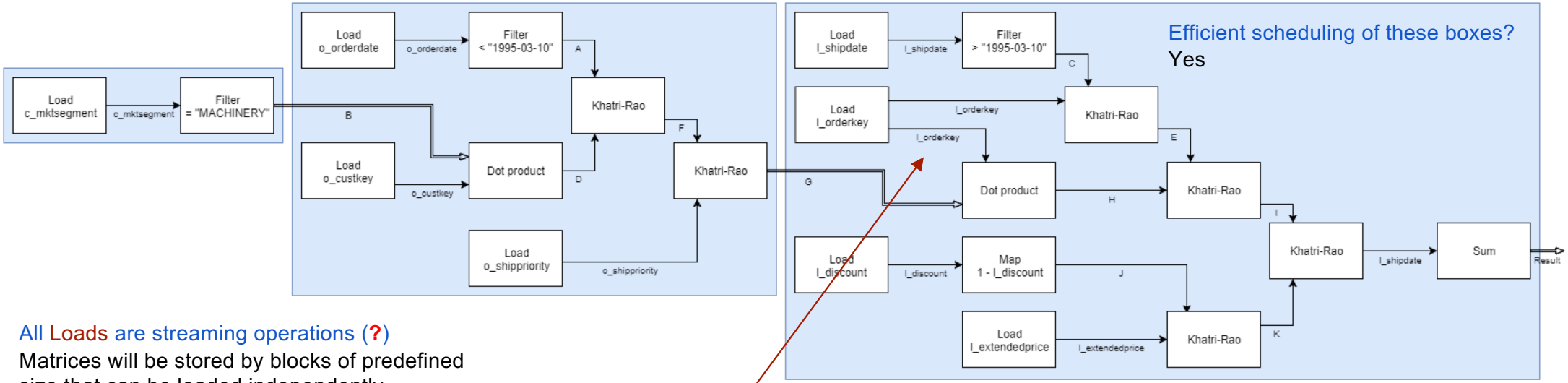
Let's calculate `Md` row-wise:

Vectorized image

```
A = filter_to_matrix( o_orderdate, <'1995-03-10' )
B = filter_to_vector( c_mktsegment, ='MACHINERY' )
C = filter_to_vector( l_shipdate, >'1995-03-10' )
D = dot( B, o_custkey )
E = krao( l_orderkey, C )
F = krao( A, D )
G = krao( F, o_shippriority )
H = dot( G, l_orderkey )
I = krao( E, H )
J = map( \x -> 1 - x, l_discount )
K = hadamard( l_extendedprice, J )
L = krao( I, K )
M = bang( sum, L )
```

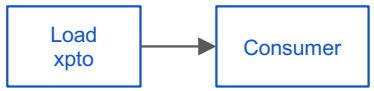
Full matrix data

Streaming data

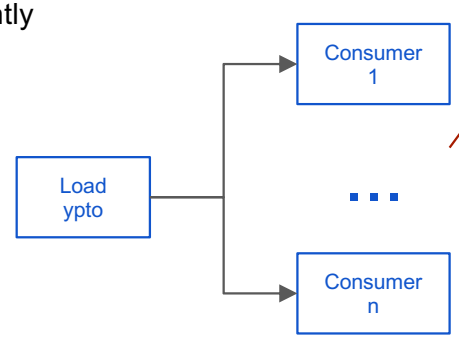


Efficient scheduling of these boxes?
Yes

All Loads are streaming operations (?)
Matrices will be stored by blocks of predefined size that can be loaded independently
(cf slide 52)



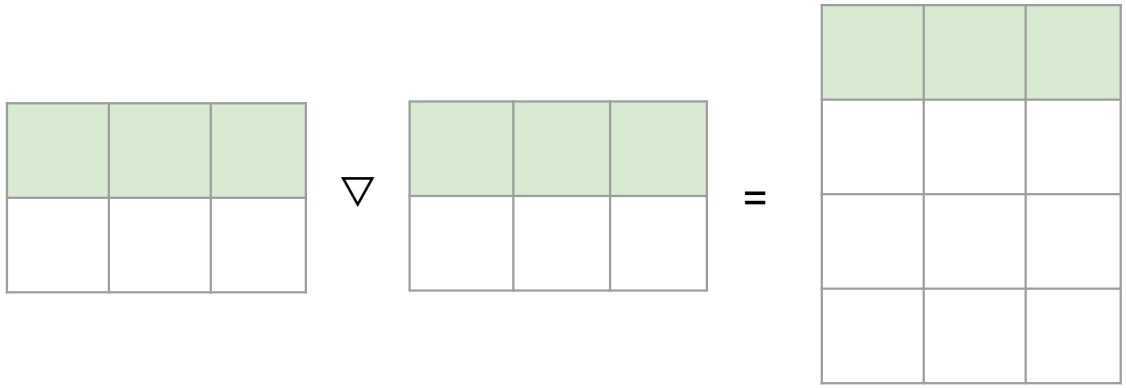
Consumer frees memory



Only last consumer frees memory
TRUE, and this forks also occur after other operations (not only loads)
E.g.: TPC-H Query 14 after map operation (G)

Problems of a block approach

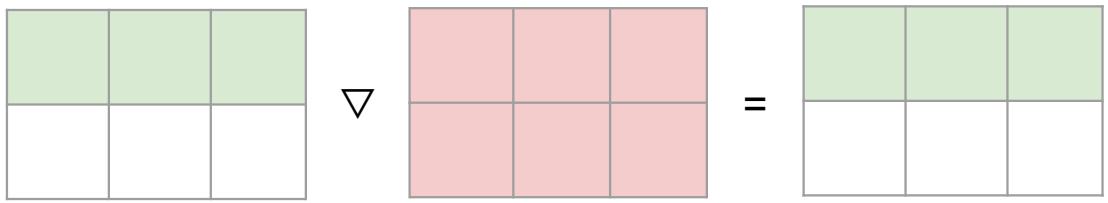
- Option 1) Dense outer matrix grows



- **After some KRs, dense matrix may not fit in memory (even on disk?)**

Problems of a block approach

- Option 2) Sparse inner blocks grow
  - Keeping the same NNZ



- We need the full matrix B to compute any row of C
  - **The memory optimization is lost!**

Matrix Transpose

1			1
	1		

COO:

c: 0 1 3

r: 0 1 0

1	
	1
1	

COO:

c: 0 0 1

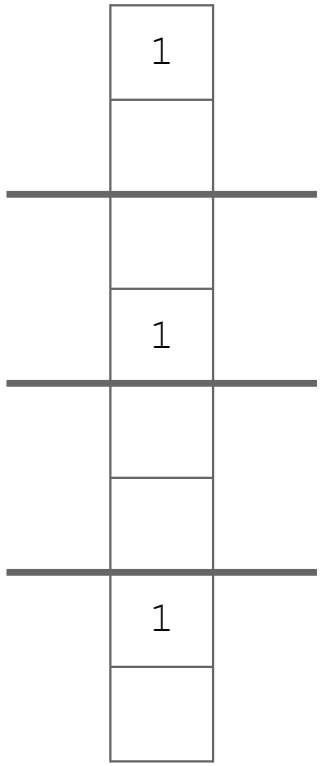
r: 0 3 1

- 1st – Swap column and row pointers: **Θ(1)**
- 2nd – Sort both arrays (based on columns):
  - **Θ(NNZ * log(NNZ))** – depending on sorting algorithm
- Comparing to the other operations – Θ(NNZ)
  - For a million records: **≈20 times slower**
  - For an american trillion records: **≈40 times slower**
  - Can be worst depending on the parallelization strategy

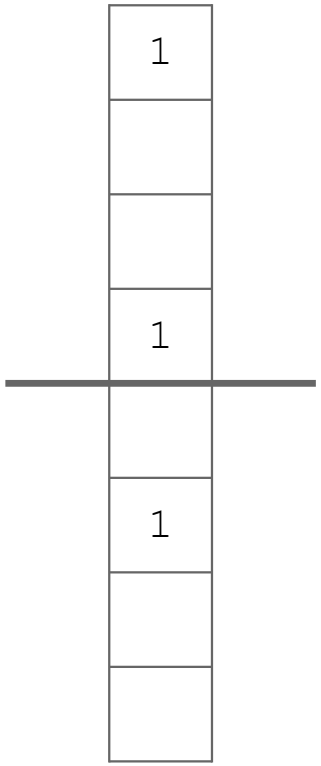
Matrix Vectorization

1			1
	1		

- COO:
  - c: 0 1 3
  - r: 0 1 0



- R←C ⇒ CxR←1
  - nr = c*R+r
  - nr: 0 3 6



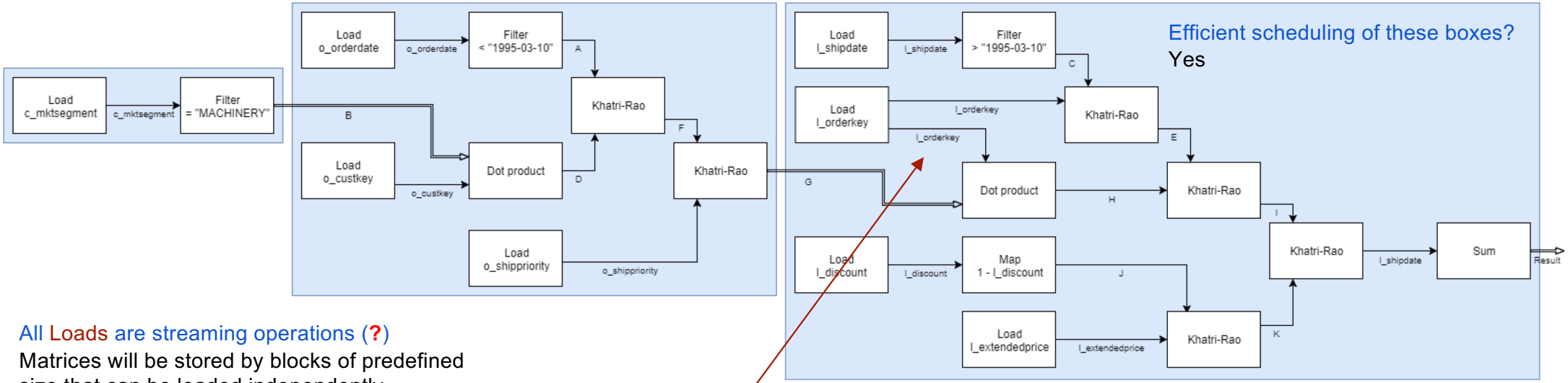
- R←C ⇒ RxC←1
  - nr = r*C+c
  - nr: **0 5 3**

Vectorized image

```
A = filter_to_matrix( o_orderdate, <'1995-03-10' )
B = filter_to_vector( c_mktsegment, ='MACHINERY' )
C = filter_to_vector( l_shipdate, >'1995-03-10' )
D = dot( B, o_custkey )
E = krao( l_orderkey, C )
F = krao( A, D )
G = krao( F, o_shippriority )
H = dot( G, l_orderkey )
I = krao( E, H )
J = map( \x -> 1 - x, l_discount )
K = hadamard( l_extendedprice, J )
L = krao( I, K )
M = bang( sum, L )
```

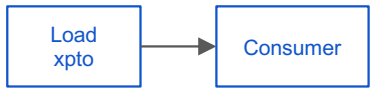
⟹ Full matrix data

⟶ Streaming data

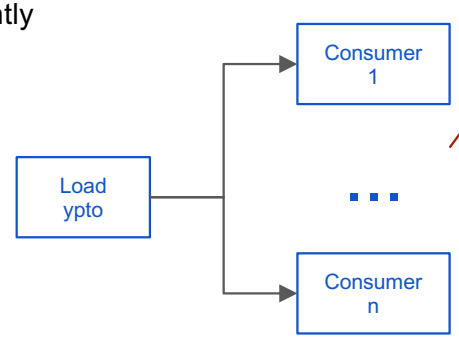


Efficient scheduling of these boxes?
Yes

All Loads are streaming operations (?)
Matrices will be stored by blocks of predefined size that can be loaded independently
(cf slide 52)



Consumer frees memory



Only last consumer frees memory
TRUE, and this forks also occur after other operations (not only loads)
E.g.: TPC-H Query 14 after map operation (G)

Matrix Vectorization

- Consider a generic COO matrix: `AxB←CxD`
  - The following vectorizations are trivial and cost **Θ(NNZ)**:
    - `B←CxDxA`
    - `1←CxDxAxB`
    - `DxAxB←C`
    - `CxDxAxB←1`
  - Any other needs sorting, thus costing **Θ(NNZ * log(NNZ))**

Vectorization analysis

From	To	Orientation	Read Cost	Write Cost	Memory Used
COO	COO	Vertical	Θ(NNZ)	Θ(NNZ)	NNZ
		Horizontal	Θ(NNZ)	Θ(NNZ)	NNZ
	CSC	Vertical	Θ(NNZ)	Θ(NNZ)	NNZ
		Horizontal	Θ(NNZ)	Θ(R x C)	R x C
CSC	COO	Vertical	Θ(C)	Θ(NNZ)	NNZ
		Horizontal	Θ(C)	Θ(NNZ)	NNZ
	CSC	Vertical	Θ(C)	Θ(NNZ)	NNZ
		Horizontal	Θ(C)	Θ(R x C)	R x C

State of the art

- **Relational DBMS**
- **Business Intelligence**
  - OLAP
- **Big Data / Business Analytics**
  - OLTP / Stream processing
- **Market solutions**
  - Relational: PostgreSQL
  - OLAP: Apache kylin, Hive?
  - Hybrid: Apache Drill, Impala?
  - Cloud: Google BigQuery, Amazon RedShift
- **The Typed LA approach**
  - Foundation of the previous report
- **Benchmarks**
  - TPC-H, papers

Goals for the meeting

2 Nov

- Matrix transpose via swapping & vectorization
- DSL
- DSL parser

Matrix transpose via swapping & vectorization

- T = swap . vec(M)
- vec(M) can be calculated in Θ(NNZ) [cf slide 40]
- swap = snd ▿ fst
  - `row = (col % R) * C + (col / R)`
  - `col = (row % C) * R + (row / C)`
- swap does not need to be created!
- `For each elem in vec(M)`
  - `col_swap = row_elem`
  - `row_swap  = (col_swap % R) * C + (col_swap / R)`
  - **`T.insert( col=row_swap )`** **Too costly: search+shift!**

DSL new operator

- Renamed from LAS to LAQ due to name conflict with LASSO (.las)
- Added **return (v1, v2, …, vn)** statement
  - The return is always the root node of the operators tree!
  - Otherwise multiple endpoints were allowed
    - E.g.: **`Select v1, v2, …, vn From …`**

DSL parser

- Slow development while learning C++
- Bison and flex completed!
- Operation tree constructed using adjacency lists (graphs)
  - $\Theta(\approx 1)$ search cost
  - Tree structure using “return” as root node
- What to do next?

1. Implement operators to convert the DSL in C++
2. Implement optimizations on the operators tree

Goals for the meeting

16 Nov

- Optimized Khatri-Rao between blocks
- From block to stream level
  - Who manages matrix blocks? EDIT: Considering doing it in the main function
- Data load
  - Concurrent queue? EDIT: Is protobuf thread-safe?

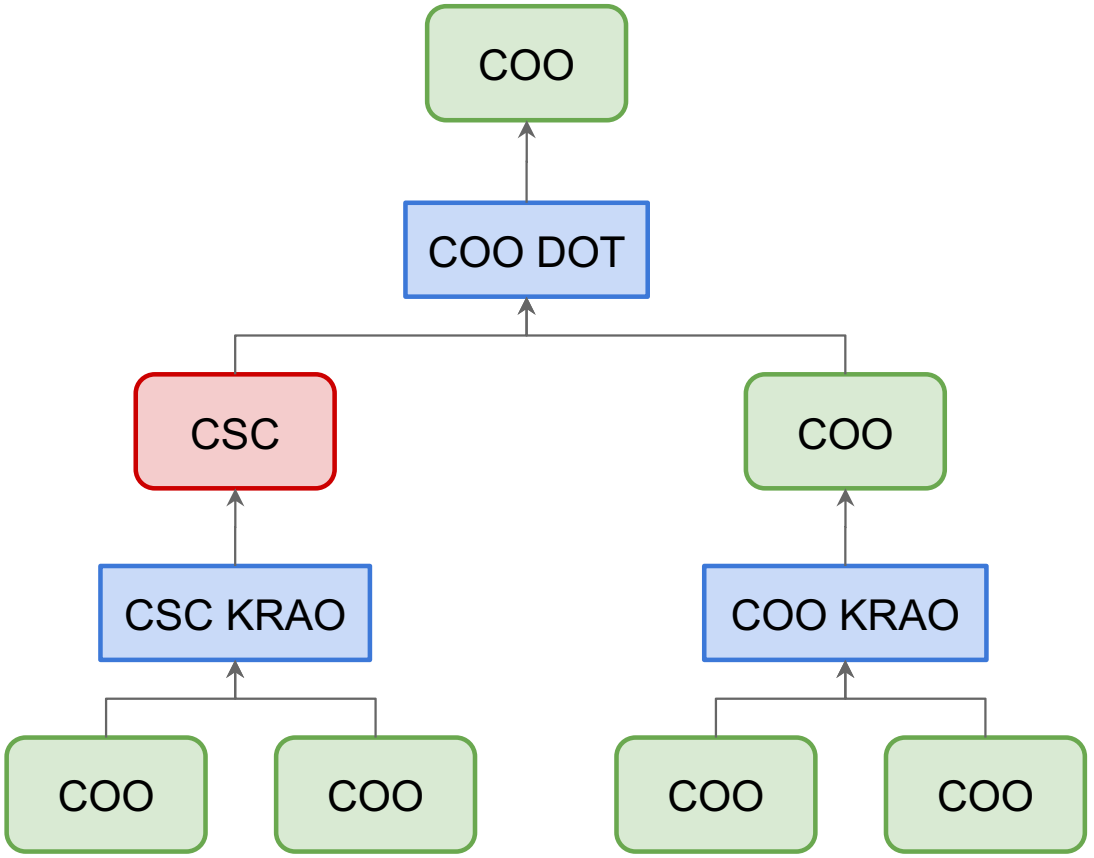
Block level Khatri-Rao

	Values	Rows	Columns
Decimal Vector			
Bitmap			
Filtered Bit Vector			
Decimal Map			
Filtered Bitmap			
Filtered Decimal Vector			
Filtered Decimal Map			

7 types of matrices: 7*7 = 49 possible operations

Block level Khatri-Rao

- CSC or COO output
- C = A ▿ B
  - **Possibly destroy A and/or B to optimize operations**
  - **EDIT: Too soon to do it with protobuf, wait for benchmark results**
- 49 * 2 * 2 = **196 combinations**
- Some functions can be reused:
  - **Total: 89**
  - **EDIT: maybe 61**



Goals for the meeting

23-11

- Write some example queries in C++ format
  - Look for a default syntax
- Matrix data serialization module
  - Choose a library/framework
  - Define the data structure
- Rewrite operations (Khatri-Rao) using the lib
- Define unit tests for the parser and the krao operator
- Design data io API
- Design bang: sum/count/avg functions

Data serialization

- Select a serialization API
  - Based on: C++ serializers benchmark, TensorFlow & Apache Calcite:
    - Google Protocol Buffer
- Code modified to use protobuf generated classes
- Functional version tested on local machine
  - Protobuf (and ALL dependencies) installed on SeARCH
    - Still with linking issues
    - Possibly replace Makefile with CMake
      - Short available time to learn CMake: **is it worth the time?**

Matrix protobuf specification

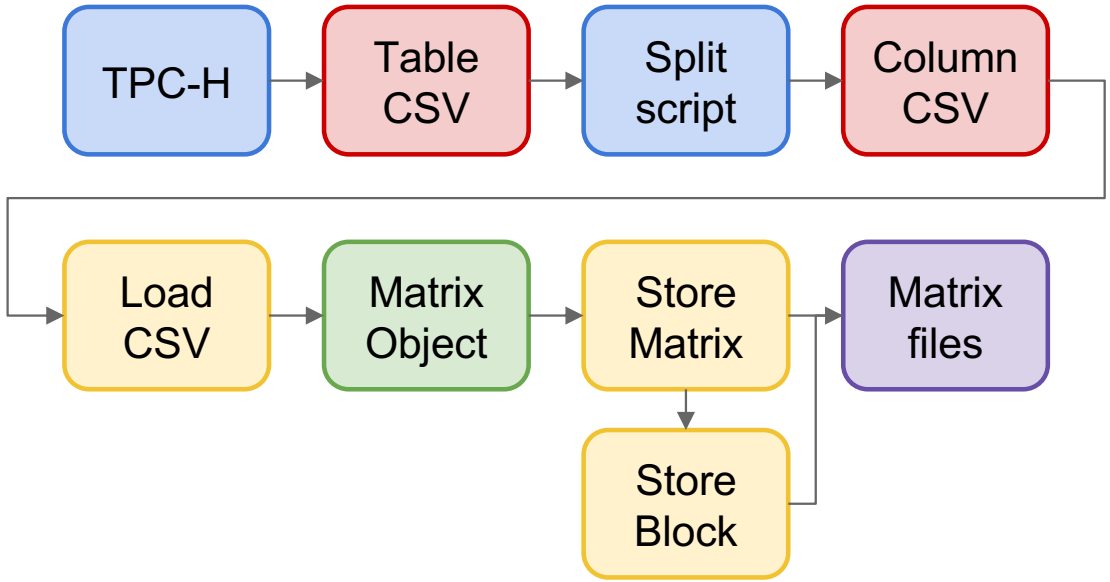
```
syntax = "proto3";

package engine;
import "src/block.proto";
import "src/label_block.proto";

option cc_enable_arenas = true;

message matrix {
  // matrix identifiers
  string database_path = 1;
  string table = 2;
  string column = 3;

  // matrix dimensions
  int32 nnz = 4;
  int32 nrows = 5;
  int32 ncols = 6;

  // matrix content
  map<int32, block> blocks = 7;

  // matrix labels
  map<int32, label_block> labels = 8;
}
```

Disk data structure

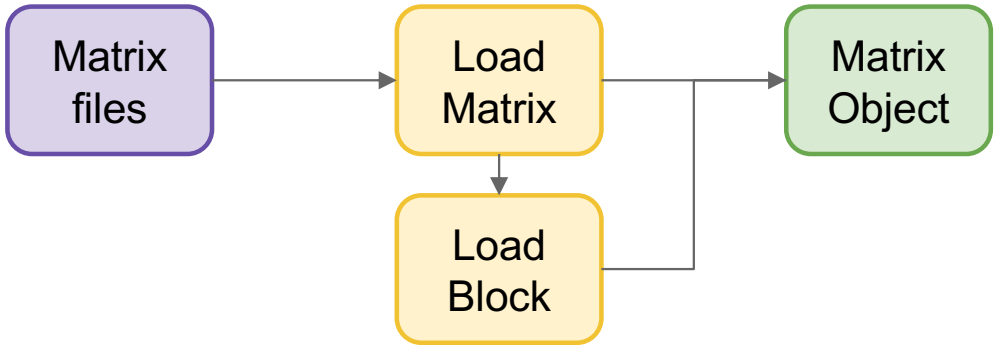
- database_path/
  - tableA/
    - column1/
      - blocks/
        - 1.dat, 2.dat, ...
      - matrix.dat
      - labels/
        - 1.dat, 2.dat, ...
    - column2/
    - ...
  - tableB/
  - ...

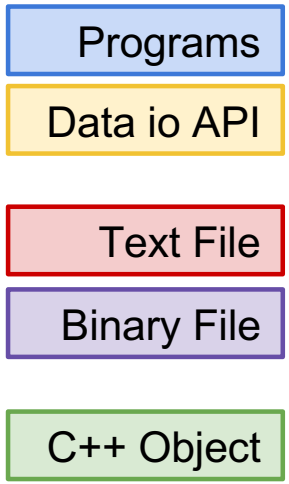
Data io API

Import CSV / store data:



Load Data:





Current status

	Design	Implementation	Test
Khatri-Rao (krao)			
Hadamard (krao)			
Load			
Bang			
Filter			
Map			
Dot product			

Needed to run TPC_H query 6

Goals for the meeting

07-12

- Queries 3, 6 execution plan (streaming)
- Query rewriter
- Label normalization
- Load API
  - Considerations on database schemas
- MySQL as example
  - More user friendly source code than PostgreSQL
- In-memory databases
  - Performance of non-volatile RAM? E.g. Intel Optane

TPC-H Query 3

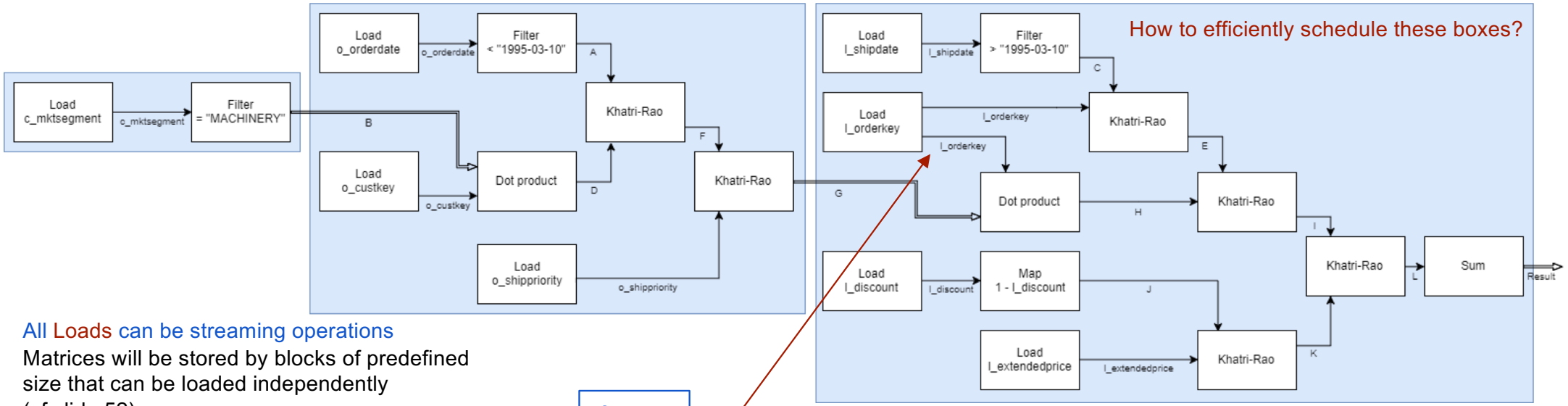
Streaming approach

```
A = filter_to_matrix( o_orderdate, <'1995-03-10' )
B = filter_to_vector( c_mktsegment, ='MACHINERY' )
C = filter_to_vector( l_shipdate, >'1995-03-10' )
D = dot( B, o_custkey )
E = krao( l_orderkey, C )
F = krao( A, D )
G = krao( F, o_shippriority )
H = dot( G, l_orderkey )
I = krao( E, H )
J = map( \x -> 1 - x, l_discount )
K = hadamard( l_extendedprice, J )
L = krao( I, K )
M = bang( sum, L )
```

SVG

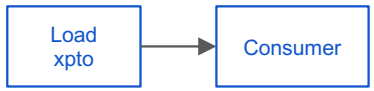
Full matrix data

Streaming data

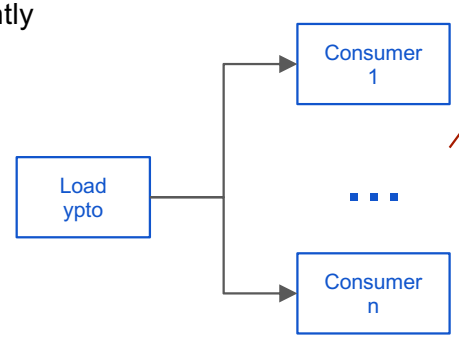


All Loads can be streaming operations

Matrices will be stored by blocks of predefined size that can be loaded independently (cf slide 52)



Consumer frees memory



Only last consumer frees memory

TRUE, and this forks also occur after other operations (not only loads)
E.g.: TPC-H Query 14 after map operation (G)

TPC-H Query 6

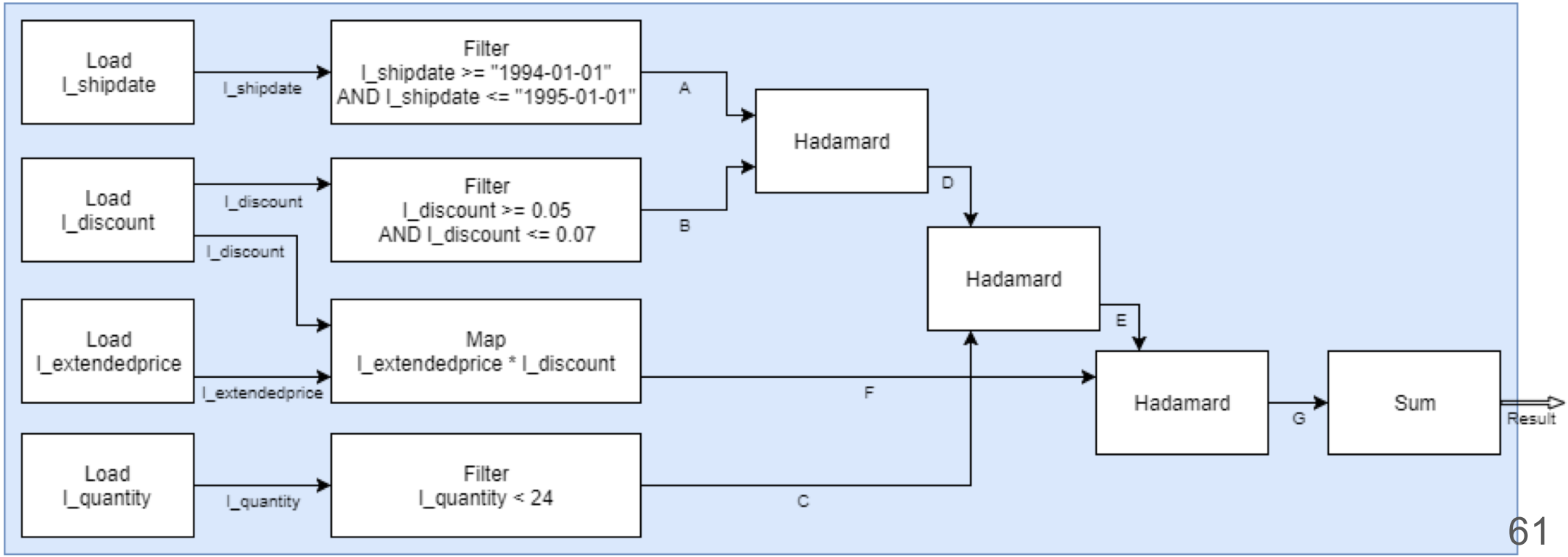
Streaming approach
(Unoptimized)

```
A = filter( l.shipdate >= "1994-01-01" AND l.shipdate <= "1995-01-01" )
B = filter( l.discount >= 0.05 AND l.discount <= 0.07 )
C = filter( l.quantity < 24 )
D = hadamard( A, B )
E = hadamard( C, D )
F = map( l.extendedprice * l.discount )
G = hadamard( E, F )
H = sum( G )
return ( H )
```

SVG

```
A = filter_to_vector( l_shipdate, >='1995-03-10', <'1995-03-10')
B = filter_to_vector( l_discount, >='0.49', <='0.51' )
C = filter_to_vector( l_quantity, <='3' )
D = hadamard( A , B )
E = hadamard( C , D )
F = hadamard( l_extendedprice , l_discount )
G = hadamard( E , F )
H = bang( sum, G )
```

Before...



TPC-H Query 6

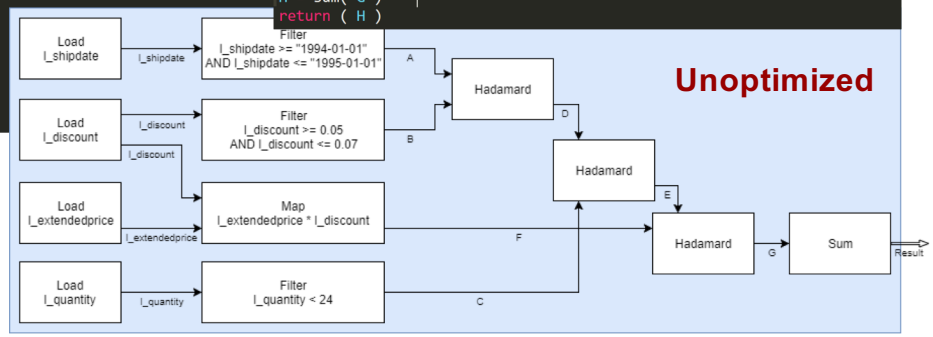
Streaming approach
(Optimized)

Optimized how? More parallelism, shorter pipeline
=> does not guarantee speedup!
Can this be automated? Yes, but LA properties must be considered
Which ones?
E.g. associative? in Hadamard, filter and map
(AxB)xC = Ax(BxC)

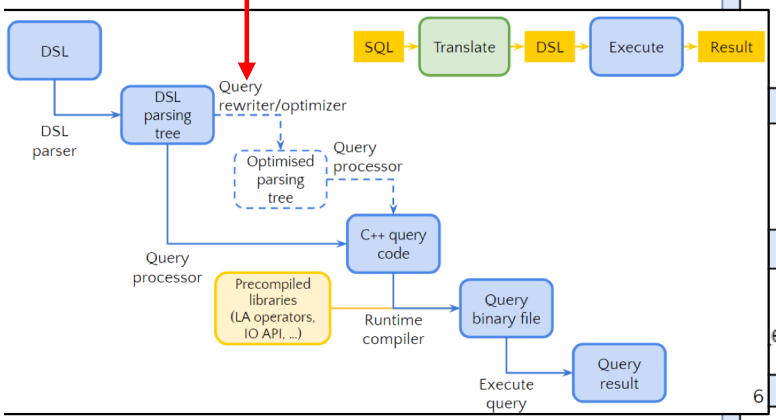
```
A = filter( l.shipdate >= "1994-01-01" AND l.shipdate <= "1995-01-01" )
B = filter( l.discount >= 0.05 AND l.discount <= 0.07 )
C = hadamard( A, B )
D = map( l.extendedprice * l.discount )
E = filter( l.quantity < 24 )
F = hadamard( D, E )
G = hadamard( C, F )
H = sum( G )
return ( H )
```

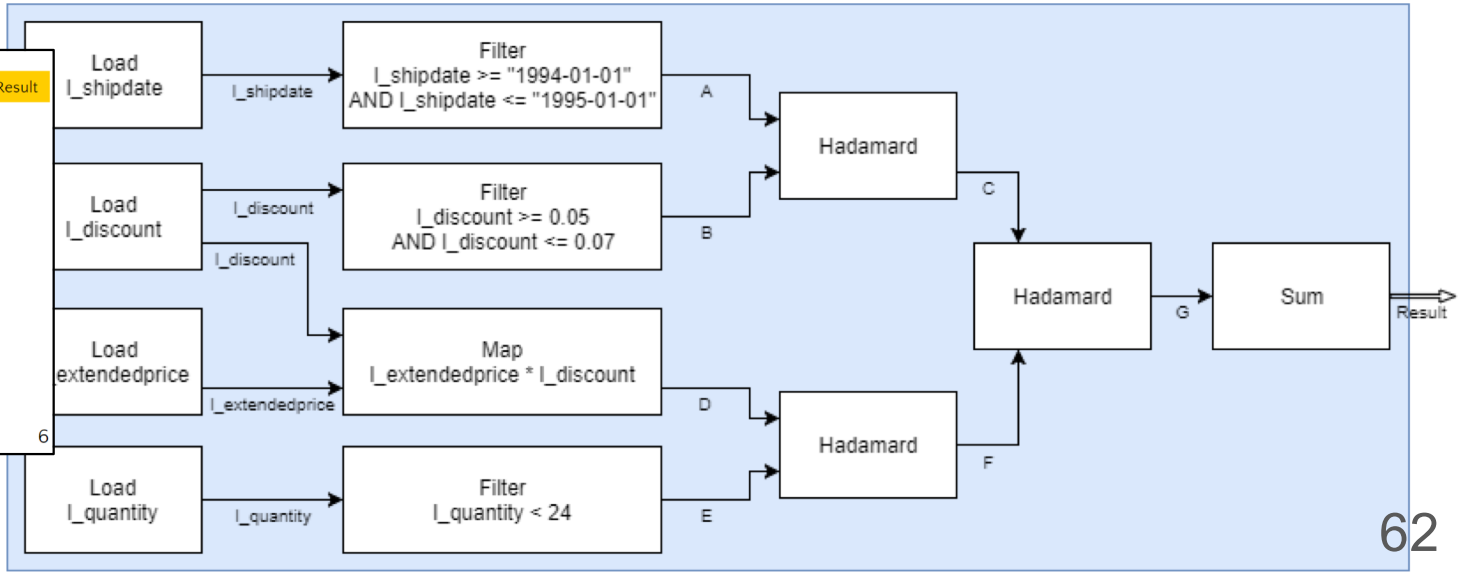
SVG

```
A = filter( l.shipdate >= "1994-01-01" AND l.shipdate <= "1995-01-01" )
B = filter( l.discount >= 0.05 AND l.discount <= 0.07 )
C = filter( l.quantity < 24 )
D = hadamard( A, B )
E = hadamard( C, D )
F = map( l.extendedprice * l.discount )
G = hadamard( E, F )
H = sum( G )
return ( H )
```



Unoptimized





Query rewriter

- Before proceeding to query translation, rewrite them to avoid
  - **SELECT FROM/WHERE/IN SELECT ?** see slides 63-64!
  - CASE statements
  - Complex aggregations
    - E.g. AVG + SUM => SUM + COUNT
  - (INNER / OUTER / LEFT / RIGHT / …) JOIN statements
    - Replace with WHERE clause(s)
- Example in the next slides…

Query rewriter (TPC-H query 1)

```
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
```

- L_quantity
  - Sum
  - Avg
  - Count
- L_extendedprice
  - Sum
  - Avg
  - Count
- L_discount
  - Sum
  - Avg
  - Count
- L_tax
  - Sum
  - Count

Query rewriter (TPC-H query 1)

- l_extendedprice * (1 - l_discount) is calculated multiple times

A

```
select
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
from
    lineitem
```

B

```
select
    sum(tmp) as sum_disc_price,
    sum(tmp * (1+l_tax)) as sum_charge
from
    lineitem,
    select
        l_extendedprice * (1 - l_discount) as tmp
    from
        lineitem
```

Considering the translation algorithm:

- is it easier to process a SELECT FROM SELECT (B)
- or to identify similar aggregations, creating tmp variables in the DSL to avoid repeated calculus (A) ?

- Let:
  - `count(*) = count(l_quantity)`
  - `Avg = sum / count`

A

```
select
    ...
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    ...
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
```

B

```
select
    ...
    sum_qty/count_order as avg_qty,
    sum_base_price/count_order as avg_price,
    sum(l_discount)/count_order as avg_disc
from
    lineitem,
    select
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        ...
        count(l_quantity) as count_order
```

Again:

- is it easier to process a SELECT FROM SELECT (B)
- or to dive AVG into SUM/COUNT avoiding repeated calculus (A) ?

Summary

- SQL must be processed before the translation takes place
- Current examples can become deprecated, depending on both **rewriting** and **translation** algorithms
  - Complete DSL engine before using SQL

Label normalization

```
PKey (A): a, b, c, d
```

A	#0	#1	#2	#3
a	1			
b		1		
c			1	
d				1

```
FKey (B): b, c, b, a
```

B	#0	#1	#2	#3
b	1		1	
c		1		
a				1

Transforming matrix B in conformity with the labels of A

```
new_row = labels_A(labels_B(old_row))
```

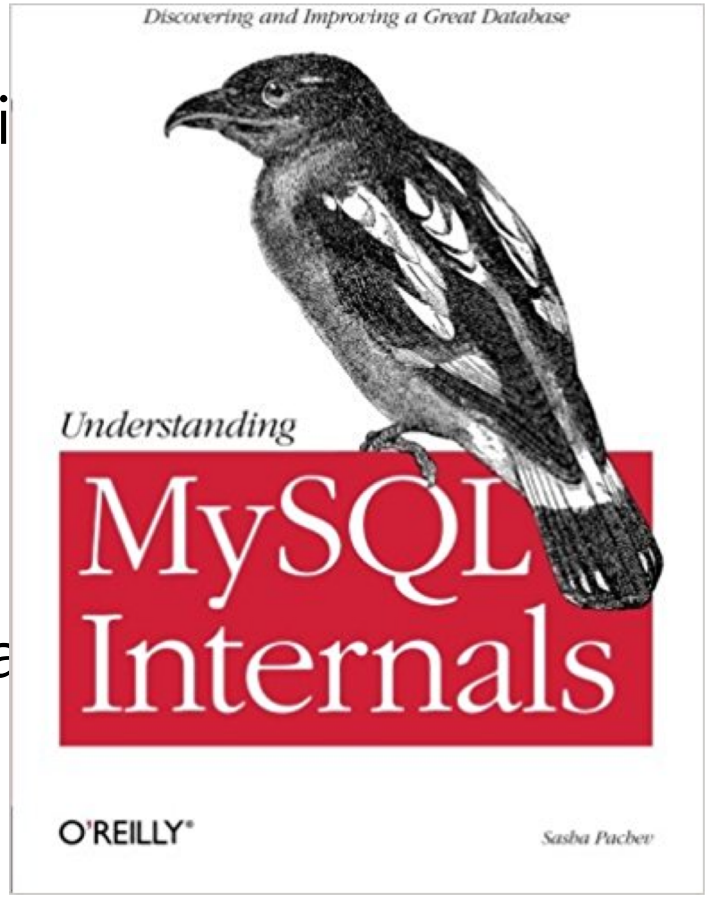
B	#0	#1	#2	#3
a				1
b	1		1	
c		1		
d				

Load API

- Import data from CSVs to bitmaps
  - **Database with a defined schema**
    - Load foreign keys respecting pk labels
      - **No overhead; drops functionalities**
  - **Schema free database**
    - Load the needed CSVs for each query execution
      - **High performance overhead**
    - Store matrices with labels extracted from their own data; map fk matrices into pk labels
      - **Limited overhead**

DBMS design patterns

- Reading about MySQL implementati
  - Trying to understand
    - Log based consistency
    - Storage mechanisms
      - Database schemas and rules
      - Table data
- **NOTE:** MySQL has no support for pa execution of a single query



In-memory databases

- In-memory databases keep all data on RAM
  - Not only during query processing
- Scalability in data size is ensured by the addition of computing nodes, each one storing part of the database
- Persistence is granted by non-volatile RAM
  - Intel Optane benchmark shows similar performance to SSD disks, although outscaling in read throughput of small files (4k)

Goals for the meeting

14-12

- Storage format
- Operations
  - Map
  - Filter
- Data creation path

Storage format

- Constraints:
  - Redundant data should not be stored
  - Values should only be loaded in filter/map/bang operatios
  - Allow streamed operations
- Solution:
  - Store Dimensions in bitmaps
  - Store Measures in dense arrays
    - Otherwise map can’t be streamed

Operations

- Map
  - Can only be applied to measures
  - Simply iterate over blocks
- Filter (Measure)
  - Same as map, but returns a sparse bit vector
- Filter (Dimension)
  - Iterate through the labels instead of values
  - **A Dot Product is needed to complete the operation**
    - Query 6 won't be fully streamed: 2 filters on dimensions

Current status

Needed to run TPC_H query 6

	Design	Implementation	Test
Khatri-Rao (krao)			
Hadamard (krao)			
Data creation			
Bang			
Filter			
Map			
Dot product			

Data creation path

- Create
  - Database
  - Table (with all columns)
- Insert
  - Value into column block
- Store
  - Block
  - Database metadata
- Load
  - Database metadata
  - Block

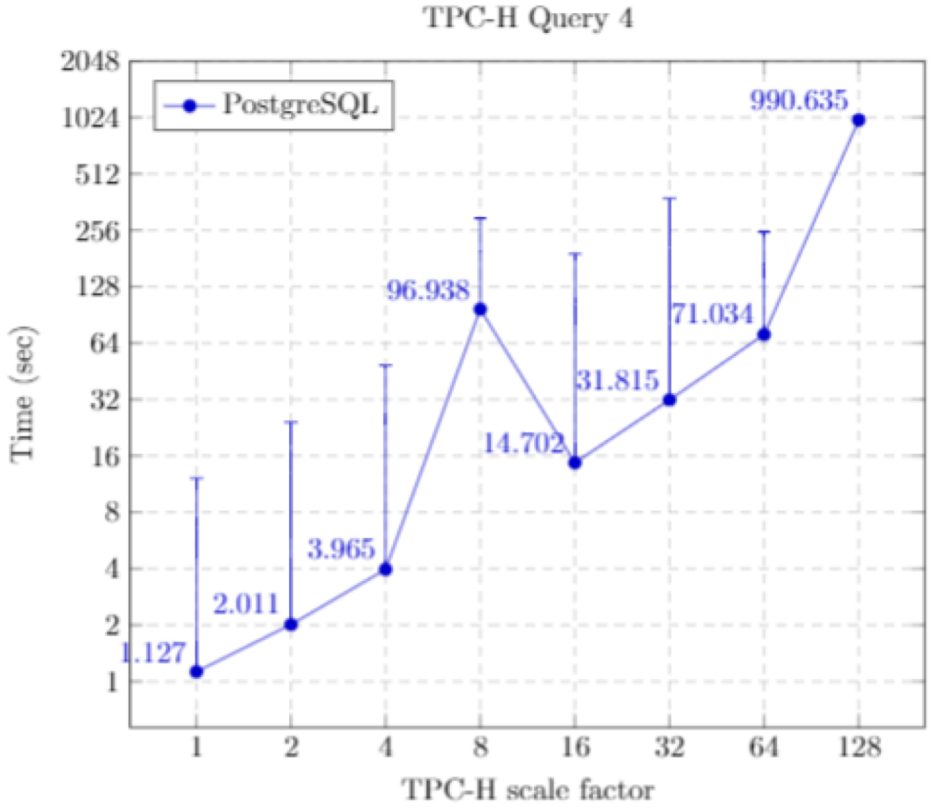
Goals for the meeting

23-01

- Benchmarks for the paper

Postgres

Variable times due to caching issues:



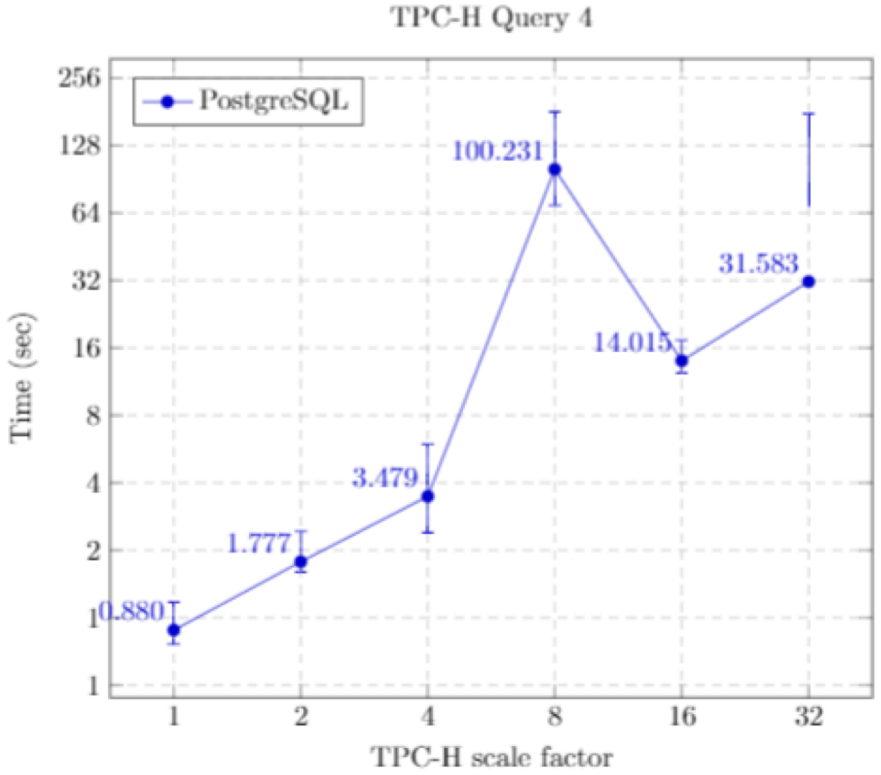
Mark: best of 2

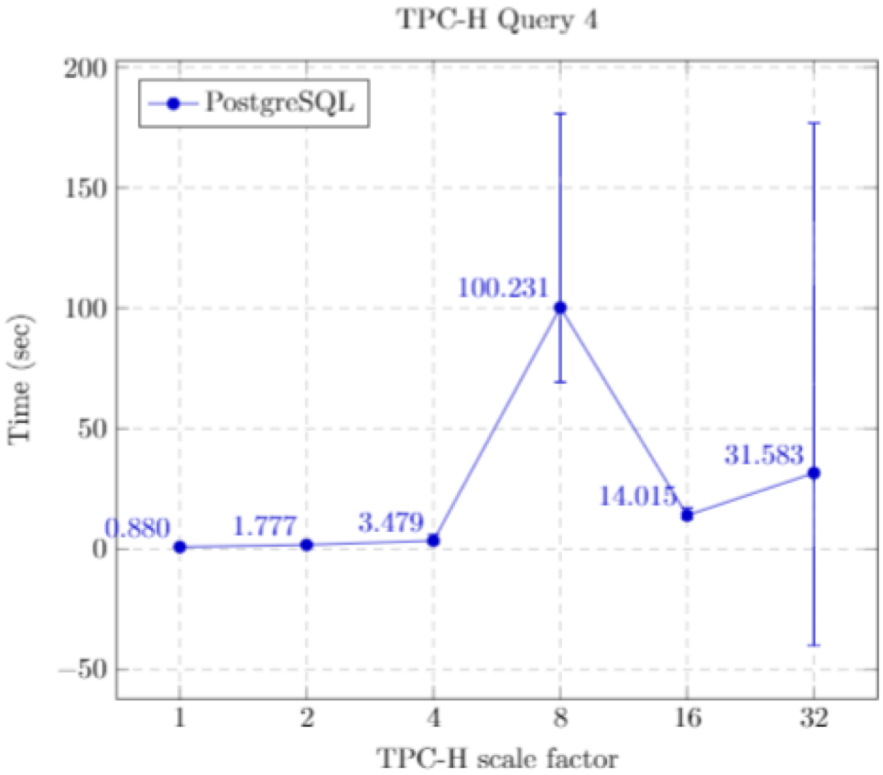
Bar center: Average

Bar size: Standard deviation

Postgres

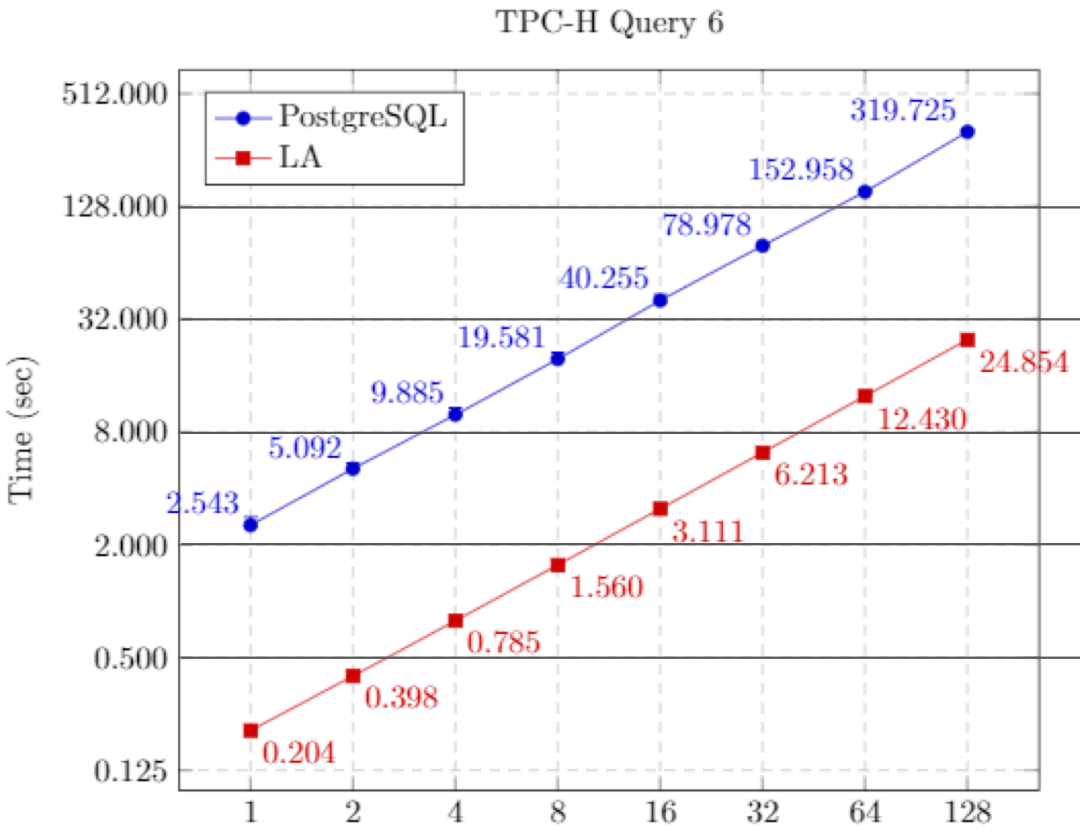
Sometimes SD>AVG, thus unplottable in logarithmic

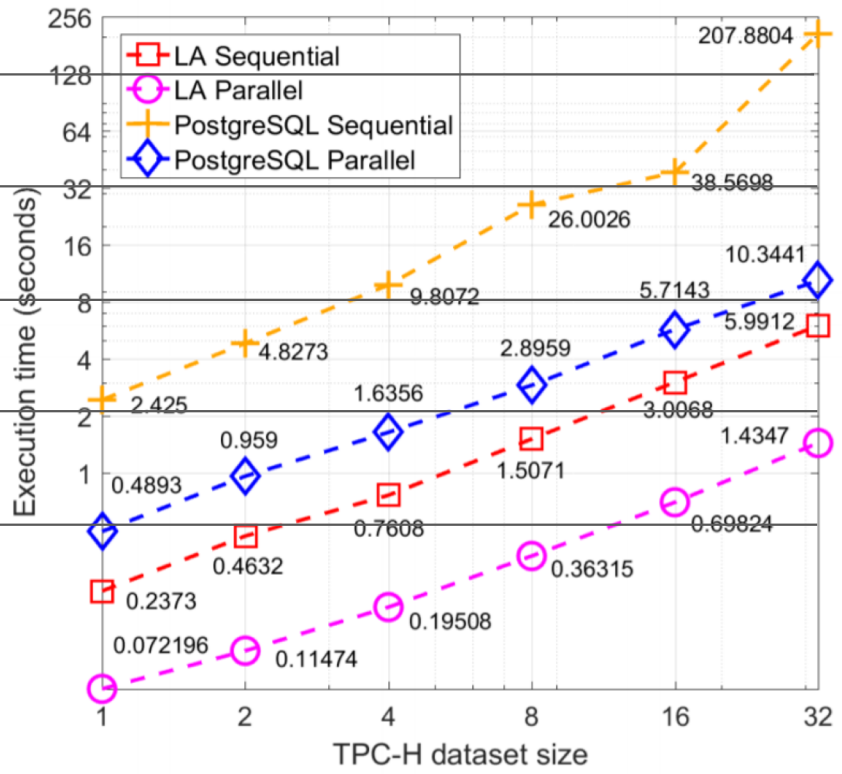




Mark: 3-best 10 (5%) Bar center: AVG Bar size: SD

Query 6


TPC-H Query 6
Time (sec)
512.000
128.000
32.000
8.000
2.000
0.500
0.125
PostgreSQL
LA
319.725
152.958
78.978
40.255
19.581
9.885
5.092
2.543
24.854
12.430
6.213
3.111
1.560
0.785
0.398
0.204
1
2
4
8
16
32
64
128


Execution time (seconds)
256
128
64
32
16
8
4
2
1
LA Sequential
LA Parallel
PostgreSQL Sequential
PostgreSQL Parallel
207.8804
38.5698
26.0026
9.8072
4.8273
2.425
10.3441
5.7143
2.8959
1.6356
0.959
0.4893
5.9912
3.0068
1.5071
0.7608
0.4632
0.2373
1.4347
0.69824
0.36315
0.19508
0.11474
0.072196
1
2
4
8
16
32
TPC-H dataset size

Goals for the meeting

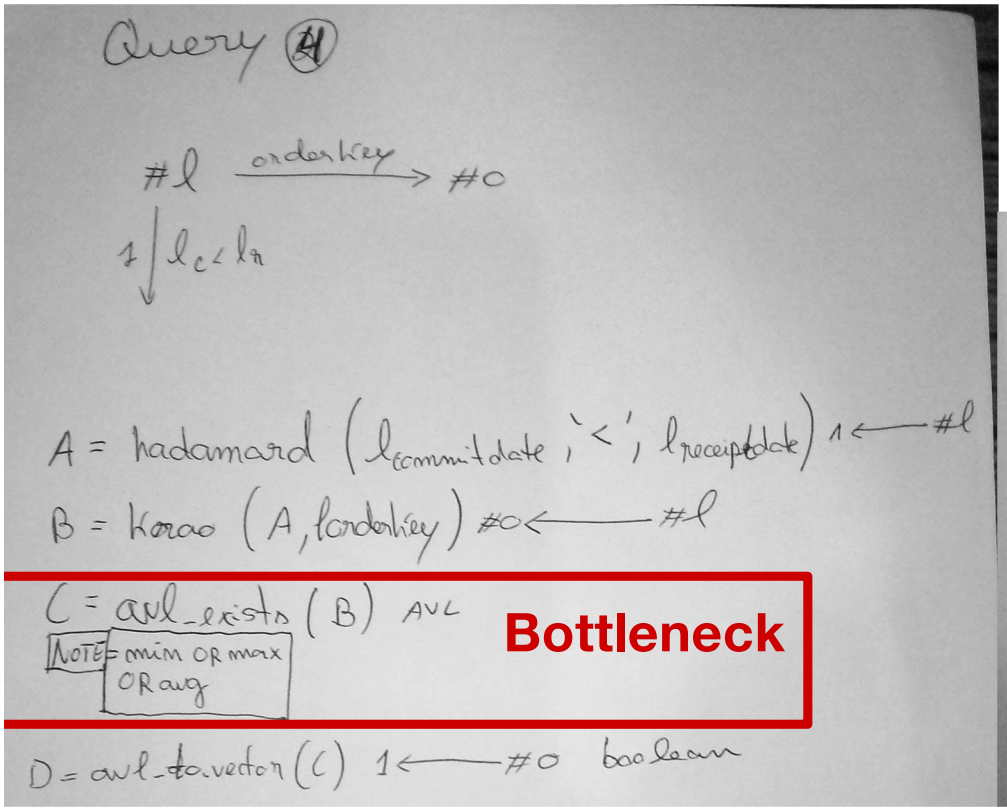
30-01

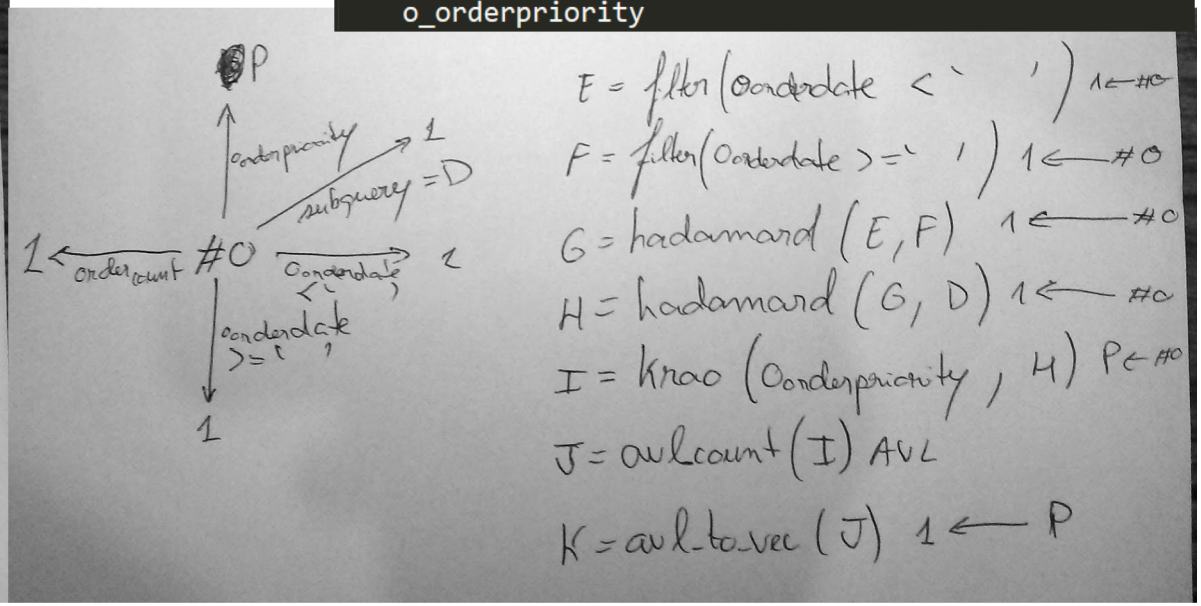
- Query 4: semi-join statement

Query 4

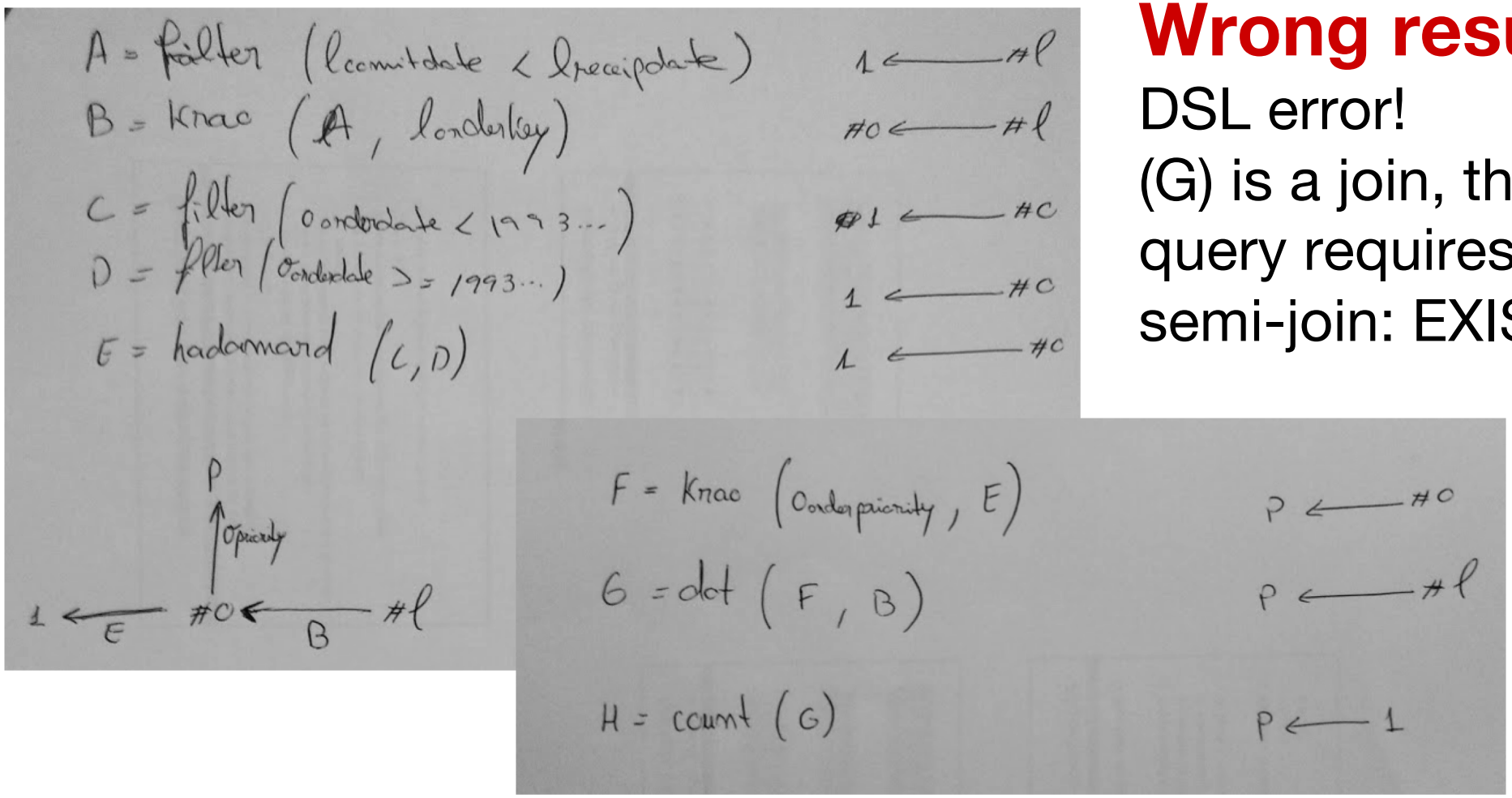
Slower than Postgres

```
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date ':1'
    and o_orderdate < date ':1' + interval '3' month
    and exists (
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
```





Query 4

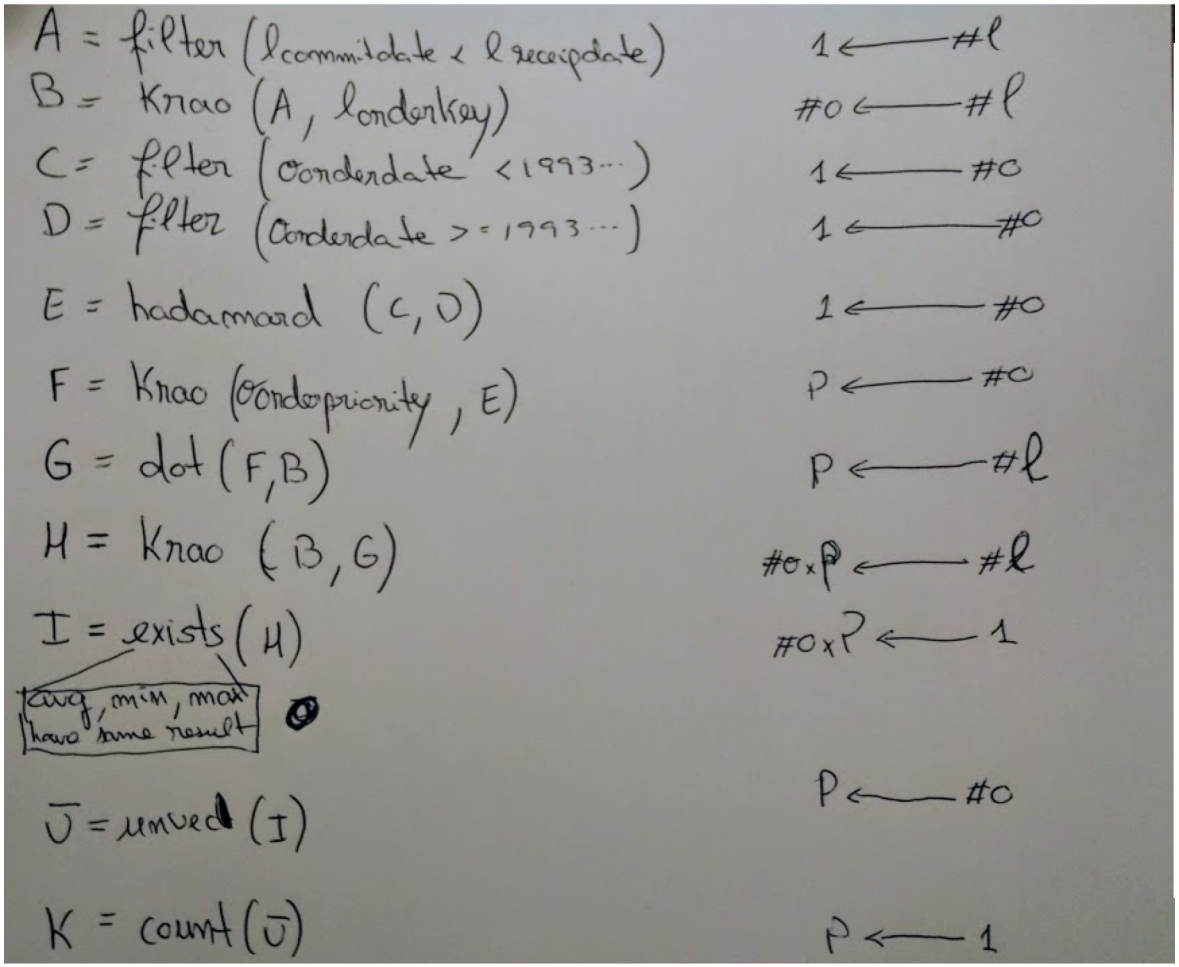


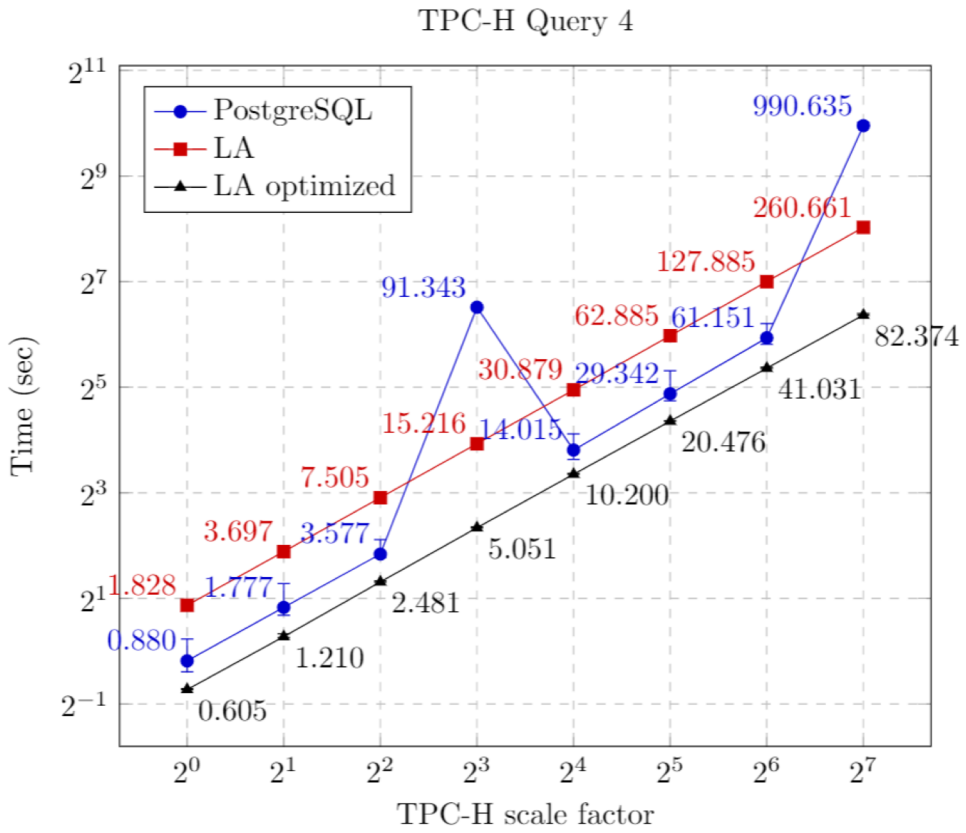
Wrong result!

DSL error!

(G) is a join, the query requires a semi-join: EXISTS

Query 4


A = filter (lcommitdate < lreceipdate)
B = Knao (A, lorderkey)
C = filter (orderdate <1993…)
D = filter (orderdate >=1993…)
E = hadamard (C, D)
F = Knao (orderpriority, E)
G = dot (F, B)
H = Knao (B, G)
I = exists (H)
avg, min, max have same result
J = unvec (I)
K = count (J)
1 ← #l
#O ← #l
1 ← #O
1 ← #O
1 ← #O
P ← #O
P ← #l
#O×P ← #l
#O×P ← 1
P ← #O
P ← 1


TPC-H Query 4
PostgreSQL
LA
LA optimized
Time (sec)
TPC-H scale factor
990.635
260.661
127.885
91.343
62.885
61.151
30.879
29.342
15.216
14.015
7.505
3.697
3.577
1.828
1.777
0.880
82.374
41.031
20.476
10.200
5.051
2.481
1.210
0.605

MySQL configuration file: my.cnf

```
[client]
port = 3306
socket = /home/laolap/new_libs/mysql/mysql/mysql.sock

[mysql]
no_auto_rehash
default_character_set = utf8

[mysqld_safe]
user = mysql
log-error = /home/laolap/new_libs/mysql/mysql/laolap_error.log

[mysqld]
port = 3306
max_open_files = 2048
table_open_cache = 1024
explicit_defaults_for_timestamp = 1
socket = /home/laolap/new_libs/mysql/mysql/mysql.sock
basedir = /home/laolap/new_libs/mysql/mysql
datadir = /home/laolap/new_libs/mysql/mysql/data
tmpdir = /home/laolap/new_libs/mysql/mysql/tmp

default_storage_engine = MEMORY
disable_partition_engine_check = true

query_cache_type = 1
query_cache_size = 1M
```

Goals for the meeting

08-03

- Redundant filters
  - Measures
  - Dimensions
  - Future further optimization
- Further optimize aggregations
- Future Khatri-Rao/Hadamard optimization

Redundant filters (measures)

Data

Facts	
Price	Quantity
5.1	9
2.3	8
3.5	9
4.7	7
4.2	5

SQL query (part of)

```
WHERE price < 2.3
  AND quant > 5
```

DSL query (1)

```
A = filter( f.price < 2.3 )
B = filter( f.quant > 5 )
C = hadamard( A, B )
```

A =	-	1	-	-	-
B =	1	1	1	1	-
C =	-	1	-	-	-

5+5+4 iterations

Desired: **5+1**

DSL query (2)

```
A = filter( f.price < 2.3
        AND f.quant > 5 )
```

Solutions:

- Iterate both matrices using C && clause (stops on false)
- Iterate **price** and then iterate the resultant vector, filtering by **quantity**

Redundant filters (dimensions)

Data

Car	
Color	Brand
Black	Ford
Blue	Ford
Black	Ford
White	Ford
White	Ferrari

SQL query (part of)

```
WHERE color = 'Blue'
  AND brand = 'Ford'
```

DSL query (1)

```
A = filter( c.color = 'Blue' )
B = filter( c.brand = 'Ford' )
C = hadamard( A, B )
```

DSL query (2)

```
A = filter( c.color = 'Blue'
        AND c.brand = 'Ford')
```

Black	Blue	White
-	1	-

•

1	-	1	-	-
-	1	-	-	-
-	-	-	1	1

A =

-	1	-	-	-

Ford	Ferrari
1	-

•

1	1	1	1	-
-	-	-	-	1

B =

1	1	1	1	-

C = AxB =

-	1	-	-	-

Next slide implements DSL (2)

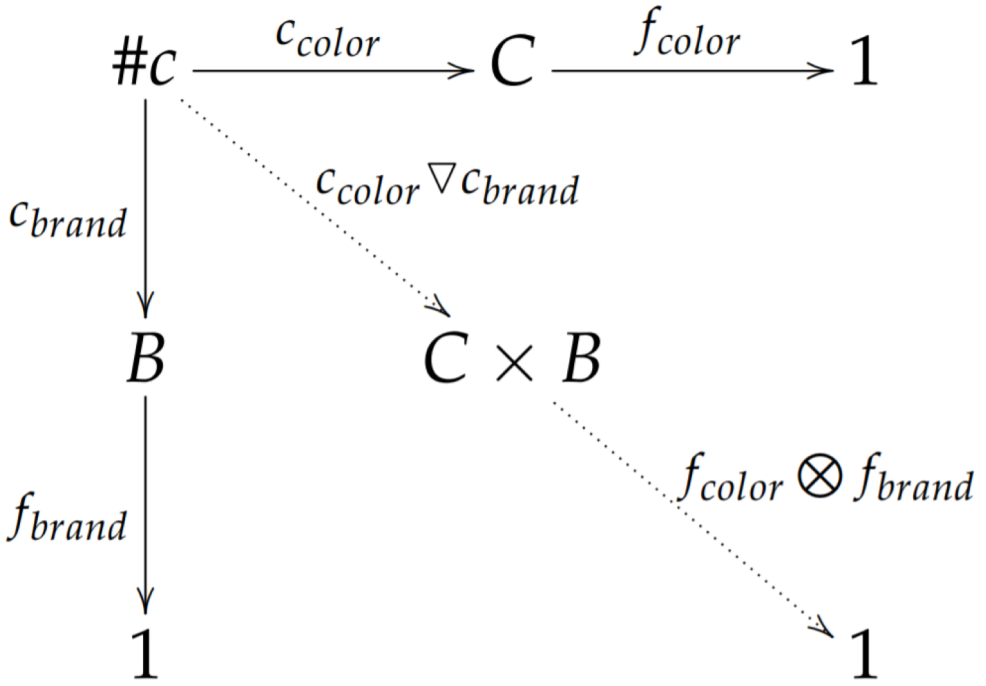
Redundant filters (dimensions)

DSL query (2)

```
A = filter( c.color = 'Blue'
        AND c.brand = 'Ford')
```

Is implemented using:

```
A = filter( c.color.labels = 'Blue' )
B = filter( c.brand.labels = 'Ford' )
C = kron( A, B )
D = krao ( c.color, c.brand )
E = dot ( C, D )
```



Problem: Khatri-Rao in (D) is always heavy (2 unfiltered matrices)
Solution: **Next Slide**

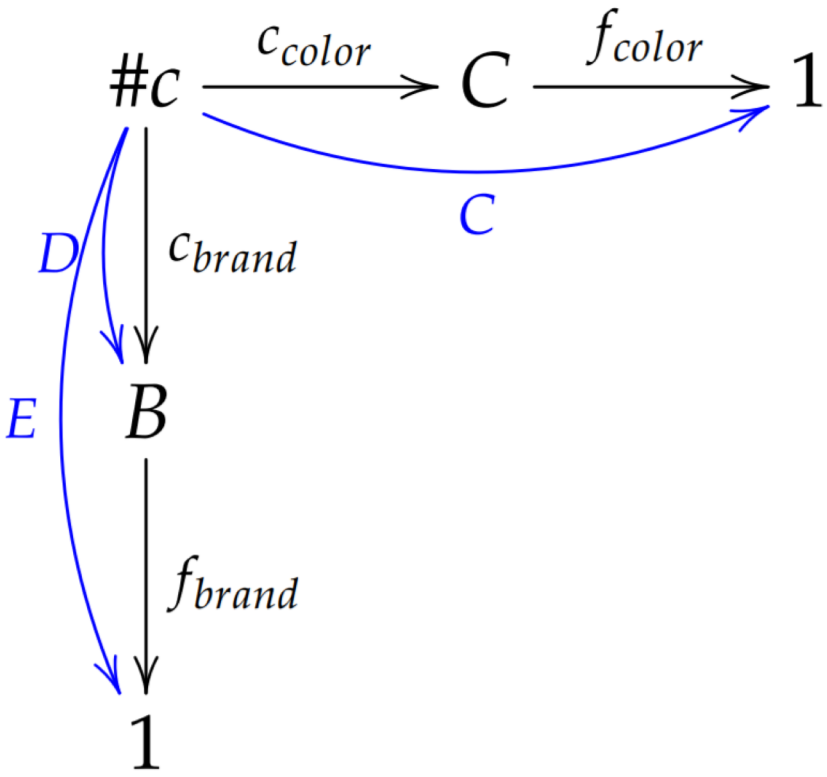
Redundant filters (dimensions)

DSL query (2)

```
E = filter( c.color = 'Blue'
        AND c.brand = 'Ford')
```

Is implemented using something like:

```
f_color = filter( c.color.labels = 'Blue' )
f_brand = filter( c.brand.labels = 'Ford' )
C = dot ( f_color, c.color )
D = krao ( C, c.brand )
E = dot ( f_brand, D )
```



Restriction: No filter can have attributes from distinct tables!
Problem: The SQL converter doesn't know which attributes are dimensions/measures
Solution: Expand the operation (as above) in DSL-runtime

Result: Highly complex filter operation, **must be studied** to check if is always solvable

Redundant filters (dimensions)

Future optimizations :

- `filter_B = predicate_B . B`
- **`predicate_A`** `.` `(` **`A`** `▿` **`filter_B`** `)`
  - Iterate the non-zeros of **`filter_B`** and get the column
  - Get the correspondent row in **`A`**
  - Check its existence in the columns of **`predicate_A`**
- 2 operations in a single run
  - Needs more loaded data! May not be faster...

Optimize aggregations

First approach by BFG:

Performance Results (2GB dataset)

The data below shows the execution time for each query step
≈ 95% of the time is occupied by the last step

Previous
performar
by Filipe

```
Execution Time: 33.830901
NNZ: 22767
0.152911 0.013123 0.597610 0.003269 0.083978 0.040801
0.121182 0.042822 0.090620 0.014628 0.009637 0.062796
0.049104 0.084366 0.111407 0.111351 32.228314
```

89

Aggregations were the bottleneck!

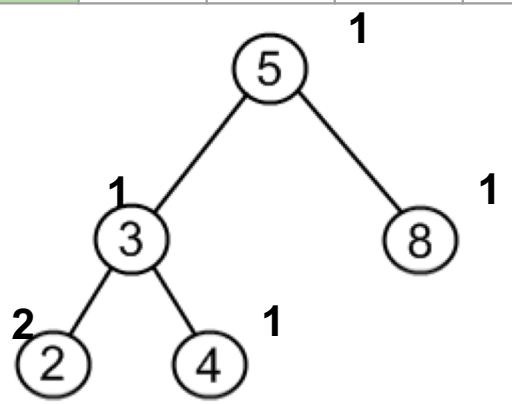
Sparse Matrix-Vector Multiplication

```
for each non-zero in vector B :
        col <- B_row_ind
        if A_col_ptr[col+1] > A_col_ptr[col] :
                temp[A_row_ind[A_col_ptr[col]]] +=
B_values[curr_nnz]

for each row in matrix A :
        if temp[i] != 0.0 :
                result_row_ind <- curr_row
                result_values <- temp[curr_r
                nnz++

col_ptr[0] = 0
col_ptr[1] = nnz
```

Basically F.m°

(using a complex algorithm...)

	#0	#1	#2	#3	#4	#5
1: A						
2: B		1		1		
3: C					1	
4: D			1			
5: D						1
6: E						
7: F						
8: G	1					



- Query 3
  - ~ **3 600 000 000** empty rows
  - ~ **0 000 011 000** rows with data
- Problem
  - `for(i=0;i<n_rows;i++)...`
  - Explained by the other group

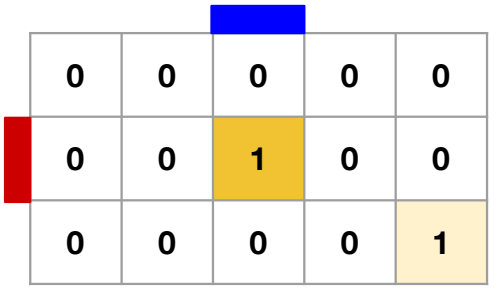
AVL avoids iterating rows

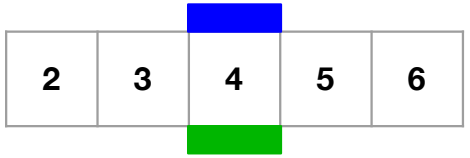
NNZ.log(NNZ) complexity!

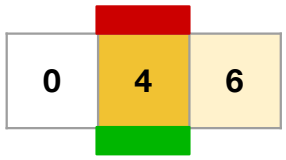
Optimize aggregations

```
Assuming: F is a (filtered) sparse Functional matrix
          m is an unfiltered measure vector
for i := 0 to f_nnz
        col = F_cols[i]
        row = F_rows[i]
        val = m_vals[col]
        res_vals[row] += val
```

Doesn't work: result vector too long (use AVL instead)
Still avoids a khatri-Rao!







Streamable since N is used transposed

$$M \cdot N = [\ A \mid B\ ] \cdot \left[\frac{C}{D}\right] = A \cdot C + B \cdot D$$

Khatri-Rao/Hadamard optimization

- COO used less memory, but [discovered during paper writing]
- Sometimes iterating the non-zeros is **NOT** the best case
  - Remember that this approach is similar to the merge on the merge sort...
- Eg.:
  - `M :: i <- k,` **`k non-zeros`**
  - `N :: j <- k,` **`1 non-zero`**
- Iterating M & N like in the merge, we have **~k+1 iterations**
- Solution:
  - Encode **M in CSC**: same memory for k non-zeros, slightly more for NNZ<k if NNZ~k
  - Iterate the few (single) non-zeros of **N**
  - Directly add Mrow[col] * Msize + Nrow[col] to the result: **1 iteration**

Goals for the meeting

15-03

- VLDB repository
- Coding
- DSL optimization

VLDB repository

- Completed (LINK)
  - Add in README: contact address
- Some parts verified
- Never tested from end to end
  - If it fails to run, at least serves as an installation and configuration guide

DSL optimization

```
A = filter( l.shipdate >= "1994-01-01" AND l.shipdate < "1995-01-01" )
B = filter( l.discount >= 0.05 AND l.discount <= 0.07 )
C = hadamard( A, B )
D = filter( l.quantity < 24 )
E = lift( l.extendedprice * l.discount )
F = hadamard( D, E )
G = hadamard( C, F )
H = sum( G )
return ( H )
```

DSL provided by the LCC students:

1 attribute per filter operation

Optimized DSL (operation A still has an hidden dot product):

```
A = filter( l.shipdate >= "1994-01-01" AND l.shipdate < "1995-01-01" )
B = filter( A, l.discount >= 0.05 AND l.discount <= 0.07 AND l.quantity < 24 )
C = lift( B, l.extendedprice * l.discount )
D = sum( C )
return ( D )
```

Coding

- Not all matrices can be encoded in COO
  - Reimplemented everything using CSC
  - COO may be explored later when fine tuning...
- Pbs using serialization of Google's *Protocol Buffers* (for disk accesses)
  - Must use the classes they generate, but not recommended...
  - Reimplemented using **Cereal** (faster, simpler, but with larger outputs)
- Current status:
  - Kernel implemented to run query 6
    - All possibilities of operators are easy to code, but development too long
  - Block load/store implemented

Goals for the meeting

05-04

- Query 6 streamed version
  - Profiling
  - GPU

Query 6 profiling (serial streaming)

```
[laolap@compute-431-7 dbms]$ perf stat -e cpu-clock,page-faults,context-switches,cpu-migrations engine/bin/q6
```

```
Performance counter stats for 'engine/bin/q6':

      3144.982266      cpu-clock (msec)
            24203      page-faults
            23778      context-switches
                0      cpu-migrations

      5.680358410 seconds time elapsed
```

Analysys developed as learned in *Engenharia de Sistemas da Computação*

Perf can introduce some overhead, but is a powerful tool to find hotspots (no changes to the source code are needed)

TBC: Ongoing issues with perf on Search...

NOTE: Consider compressing files on disk to make the engine less IO bound

Queries on GPU

Jose, 9:30 AM

Viva - Como disse, não poderei estar na reunião de amanhã (nem por ksype) pois estou de viagem. Mas, aqui, surgiu na conversa (quando falei da nossa LAOLAP a colegas daqui) surgiu a ideia de correr os scripts em GPUs, uma ideia antiga; qual é a vossa perspectiva sobre isto, agora que se tem muitos mais experiência nestas coisas?

Possible reasons to reject this suggestion:

1. The key bottleneck is the data access in memory, and we are talking about BIG DATA; the GPU memory is not shared with the multicore memory, and data transfers between memories is through PCIe. If we manage to overcome this...
2. …

João:

1. True, I'm working to complete query 6 and then make the profile of the streamed version. The GPU could make part of the pipeline if there are computation heavy LAQ(DSL) operators
2. Despite load time, COO operations cannot be vectorized and directly ported to GPU, but some operations over CSC can

Goals for the meeting

12-04
19-04

- VLDB paper review
  - Incremental querying
  - Testbed environment
- DSL to C++ conversion
- Parallel streaming simple test

VLDB paper review (1)

- Validate incremental querying
  - 1) Step by step incremental Q3 (paper & pen)
    - INSERT on **Lineitem**, **Orders** and **Customer**
  - 2) A possible controller must store:
    - The query result
    - DB Schema with counters
      - to get the delta
    - Query parsing tree (or some kind of key eg: MD5)
      - to compare with new queries
- Testbed environment: see testbed for dissertation?

VLDB paper review (2)

- Comparison with columnar format (Hyper, …)
- Compare SQL coverage in DB Toaster with ours
- Clearly show that other LA+OLAP works in the literature are not in similar areas
- What experimental results should be presented?
  - Data in memory *vs.* in disk (Sugg for the paper: only mem!)
  - Incremental querying

Querying in C++ (Q6)

```
 6  #include <iostream>
 7  #include <vector>
 8  #include "include/block.hpp"
 9  #include "include/database.hpp"
10  #include "include/dot.hpp"
11  #include "include/filter.hpp"
12  #include "include/fold.hpp"
13  #include "include/krao.hpp"
14  #include "include/lift.hpp"
15  #include "include/matrix.hpp"
16  #include "include/types.hpp"
```

1 - Load precompiled libraries

- Always include **vector** and **iostream**
- Always include **block**, **matrix**, **database** and **types**
- Include the operators as needed
  - **Simplification:** include all headers

DONE

```
18  inline bool filter_a(std::vector<engine::Literal> args) {
19    return args[0] >= "1994-01-01" && args[0] < "1995-01-01";
20  }
21
22  inline bool filter_b(std::vector<engine::Decimal> args) {
23    return args[0] >= 0.05 && args[0] <= 0.07;
24  }
25
26  inline bool filter_c(std::vector<engine::Decimal> args) {
27    return args[0] < 24;
28  }
29
30  inline engine::Decimal lift_f(std::vector<engine::Decimal> args) {
31    return args[0] * args[1];
32  }
```

2 - Define the necessary predicates **DONE**

- For each statement in the parsing tree:
  - If the operation is a **filter** or **lift**:
    - Check if the (first) attribute is a **measure** or **dimension**
    - Create the predicate using the parsed **expression**

```
34 int main() {
35   engine::Database db(
36     "data/la",
37     "TPCH_1");
```

3 - Select the database **DONE**

- The **LAQ driver** receives **engine::Database** as argument
  - get() the arguments from the provided DB

```
39   engine::Bitmap *lineitem_shipdate =
40     new engine::Bitmap(db.data_path,
41                        db.database_name,
42                        "lineitem",
43                        "shipdate");
44   engine::DecimalVector *lineitem_discount =
45     new engine::DecimalVector(db.data_path,
46                               db.database_name,
47                               "lineitem",
48                               "discount");
49   engine::DecimalVector *lineitem_quantity =
50     new engine::DecimalVector(db.data_path,
51                               db.database_name,
52                               "lineitem",
53                               "quantity");
54   engine::DecimalVector *lineitem_extendedprice =
55     new engine::DecimalVector(db.data_path,
56                               db.database_name,
57                               "lineitem",
58                               "extendedprice");
```

DONE

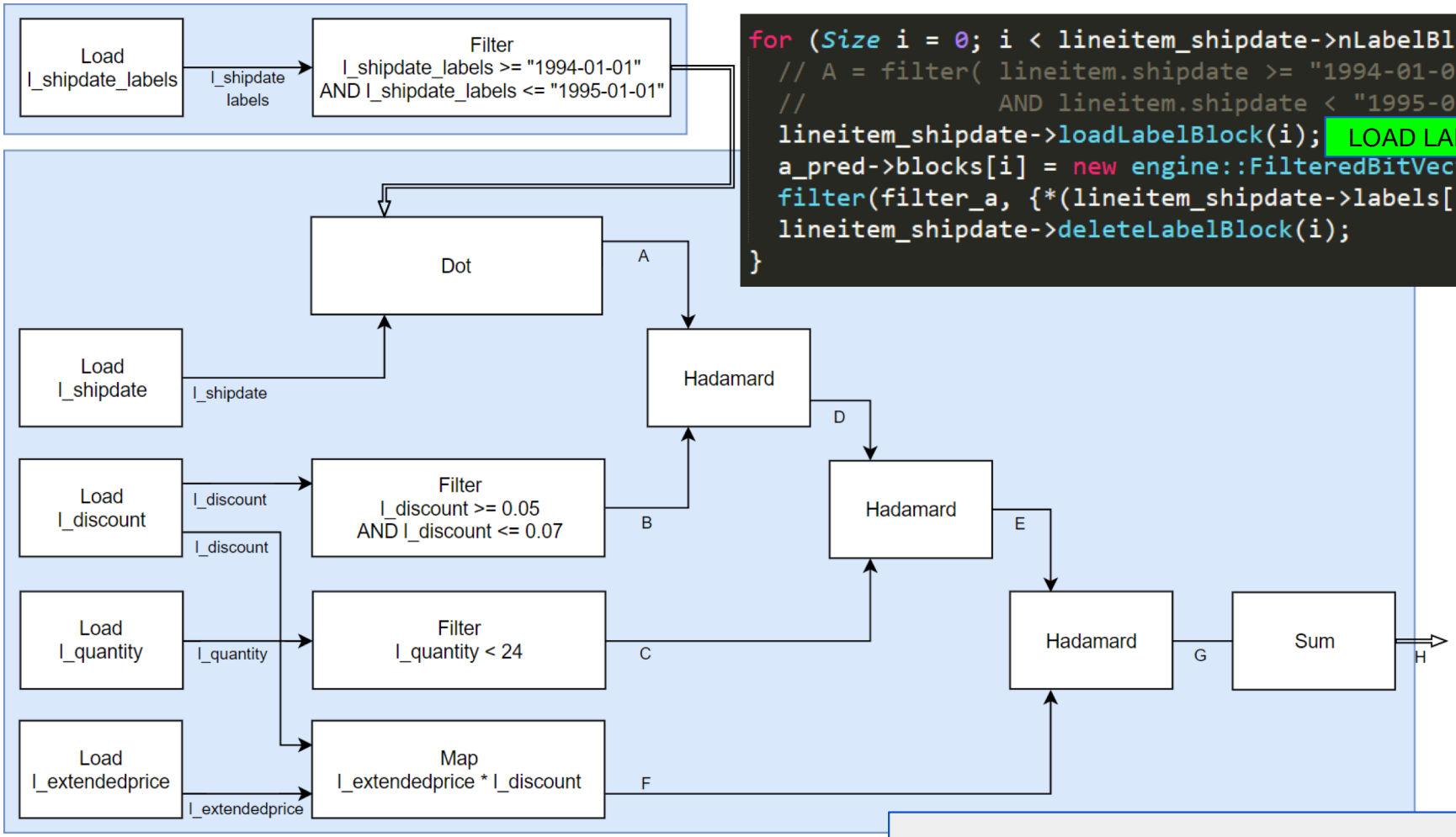
4 - Load the attributes metadata

- Iterate the list of **database vars**
  - Dimensions are Bitmaps
  - Measures are DecimalVectors

```
engine::FilteredBitVector *a_pred =
  new engine::FilteredBitVector(lineitem_shipdate->nLabelBlocks);
engine::FilteredBitVector *a =
  new engine::FilteredBitVector(lineitem_shipdate->nBlocks);
engine::FilteredBitVector *b =
  new engine::FilteredBitVector(lineitem_discount->nBlocks);
engine::FilteredBitVector *c =
  new engine::FilteredBitVector(lineitem_quantity->nBlocks);
engine::FilteredBitVector *d =
  new engine::FilteredBitVector(a->nBlocks);
engine::FilteredBitVector *e =
  new engine::FilteredBitVector(c->nBlocks);
engine::DecimalVector *f =
  new engine::DecimalVector(lineitem_extendedprice->nBlocks);
engine::FilteredDecimalVector *g =
  new engine::FilteredDecimalVector(e->nBlocks);
engine::Decimal *h =
  new engine::Decimal();
```

5 - Declare **DONE** temporary matrices

- Define a map that dictates the type of the output considering the operation and the input type(s) **INCOMPLETE**
- Recursively declare all nodes of the operation tree
- The argument is the number of blocks of the first argument in the operation




```
for (Size i = 0; i < lineitem_shipdate->nLabelBlocks; ++i) {
  // A = filter( lineitem.shipdate >= "1994-01-01"
  //             AND lineitem.shipdate < "1995-01-01" )
  lineitem_shipdate->loadLabelBlock(i);
  a_pred->blocks[i] = new engine::FilteredBitVectorBlock();
  filter(filter_a, {*(lineitem_shipdate->labels[i])}, a_pred->blocks[i]);
  lineitem_shipdate->deleteLabelBlock(i);
}
```

LOAD LABELS

FILTER

6 - Build the for loops

- 2 Blue boxes = 2 for loops
- Inside each loop all operations follow any sequence that respects dependencies

6 - some considerations

- The value in the for loop stop condition
  - Is the number of blocks of the first variable in the first operation
- Attributes may be reused, thus we need a list like this:

Variable	l_discount	l_shipdate	...	A	B	...
Type	DecimalVector	Bitmap	...	FilteredBitVector	FilteredBitVector	...
Total Uses	2	1	...	1	1	...
Actual Uses	<runtime>	<runtime>	...	<runtime>	<runtime>	...

- So if `Actual Uses = 0` and `Variable ∈ database attributes`
  - A variable block must be loaded
- If `Actual Uses = Total Uses` a variable block must be deleted

6 - some considerations

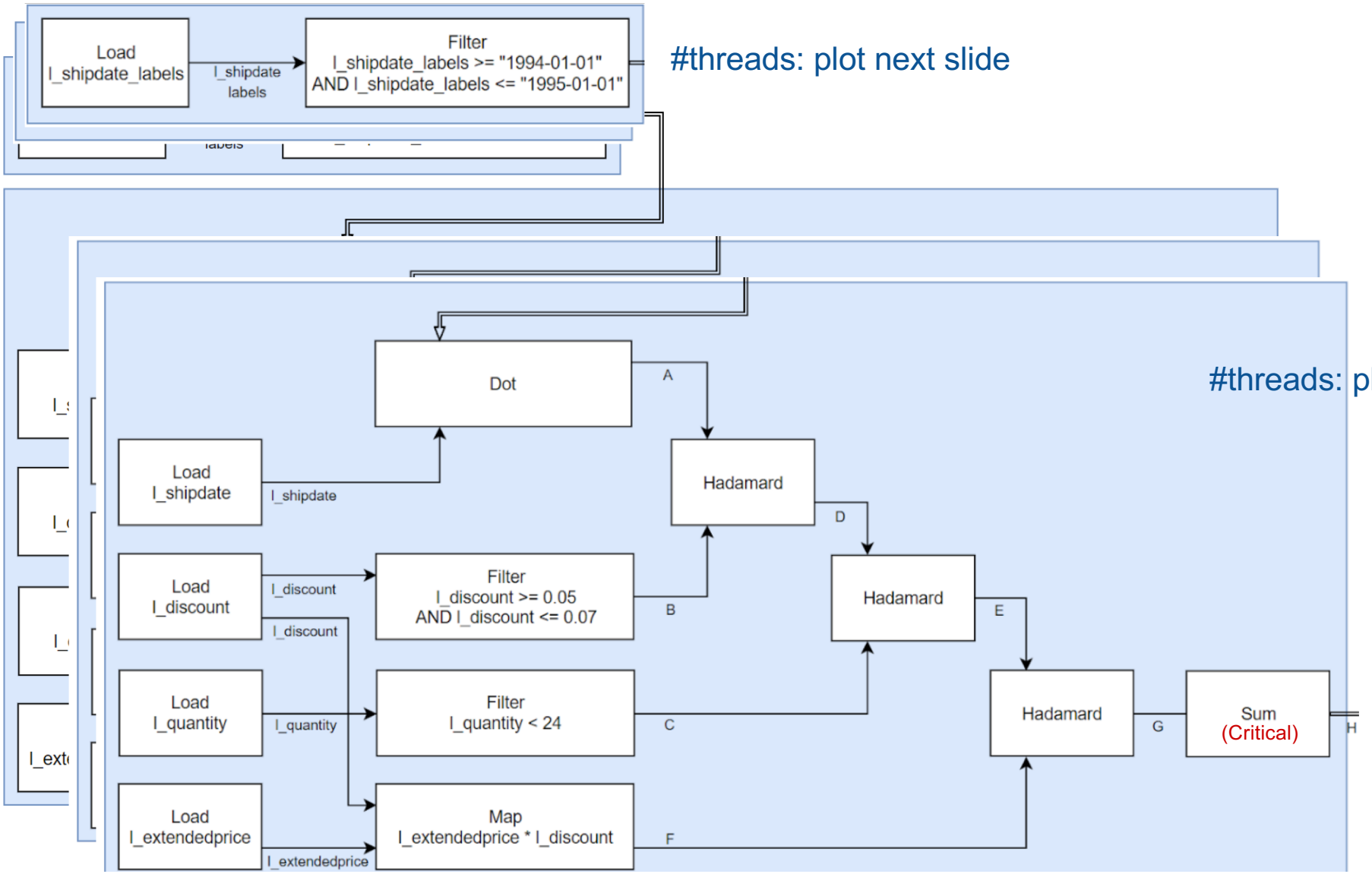
- When the operation is a Dot, Filter on dimensions or Fold
  - The previous loop must be closed and appended to a list
  - Some variables may have to be deleted **(strong arrows)**
    - Delete blocks inside streams and full matrices between for loops
  - A new loop must be created
- Special cases (unsolvable with left recursion -> solve in parent node)
  - Dot (orders_orderdate, lineitem_orderkey)
    - lineitem_orderkey loaded in stream
    - orders_orderdate loaded in a separate cycle (strong arrow)
  - FIlter on dimensions
    - Must be replaced by a filter on labels and a dot product

```
139     std::cout << (*h) << std::endl;
140
141     return 0;
142 }
```

7 - Print the result

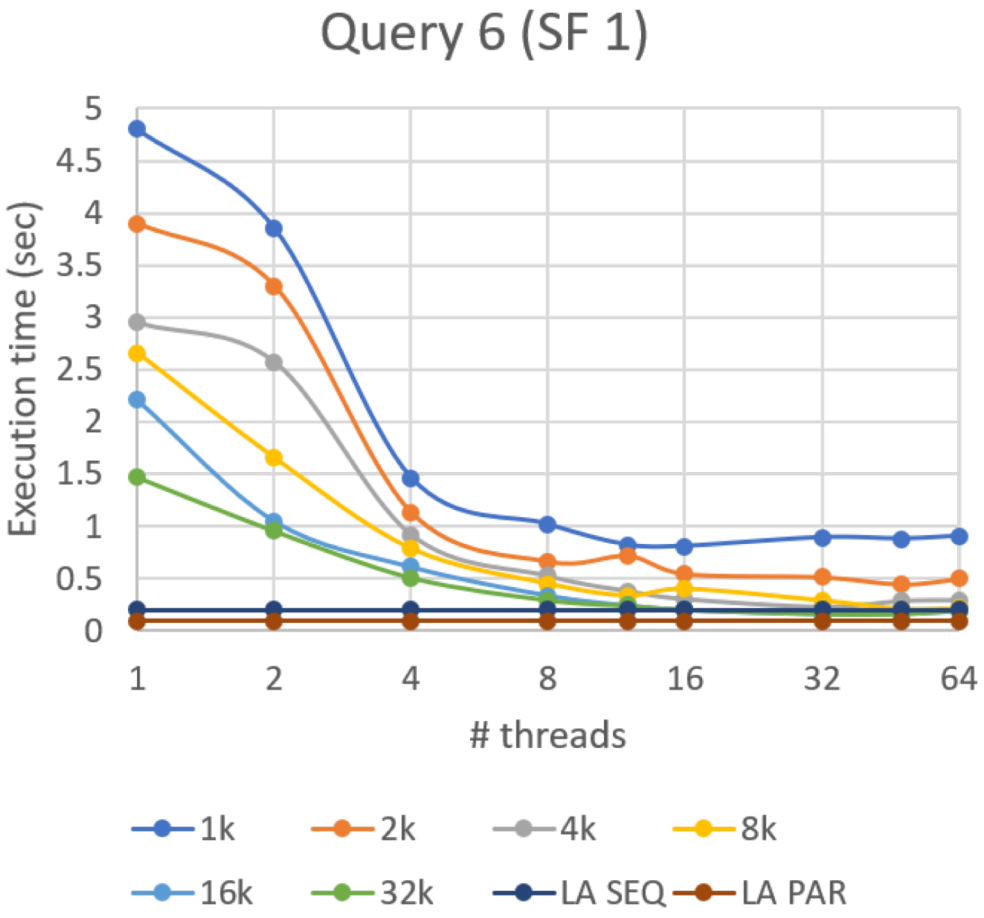
- **No algorithm (yet) for multiple outputs**
- **The final classes must implement <<**
  - In this case: double has << implemented by default

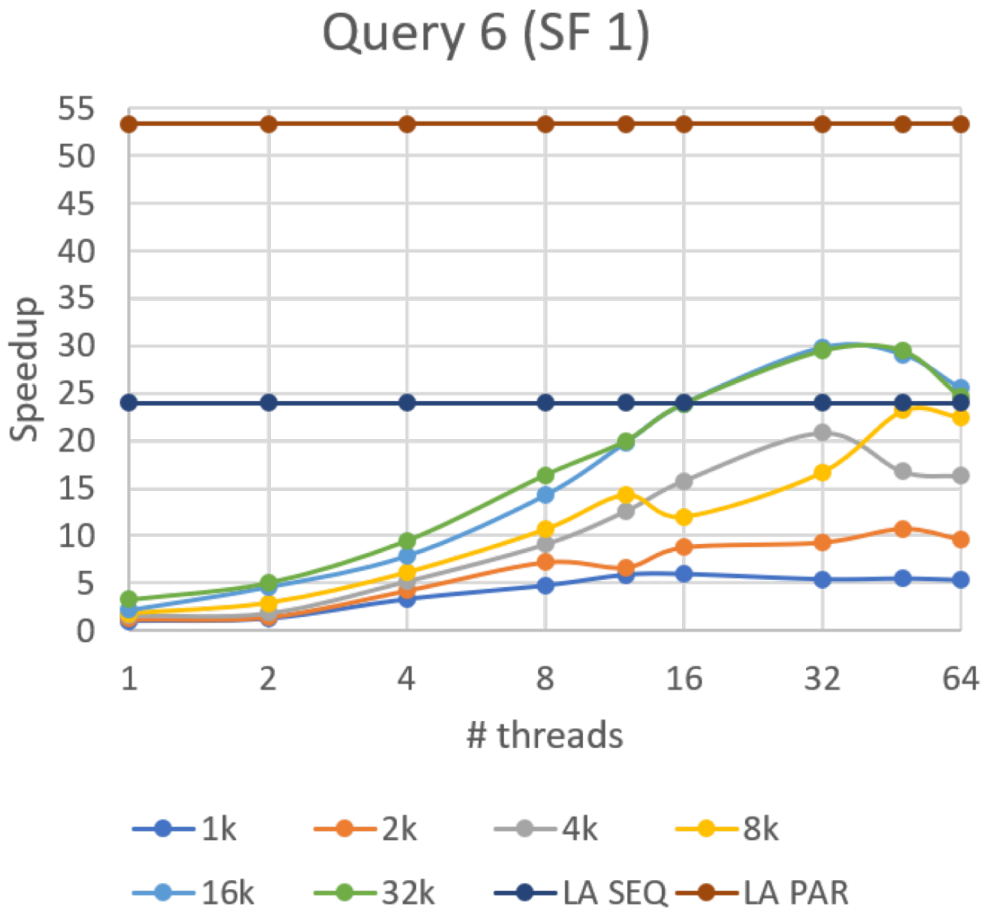
8 - Close main function **DONE**


Load
l_shipdate_labels
l_shipdate
labels
Filter
l_shipdate_labels >= "1994-01-01"
AND l_shipdate_labels <= "1995-01-01"
#threads: plot next slide
Dot
A
Load
l_shipdate
l_shipdate
Hadamard
D
Load
l_discount
l_discount
l_discount
Filter
l_discount >= 0.05
AND l_discount <= 0.07
B
Hadamard
E
Load
l_quantity
l_quantity
Filter
l_quantity < 24
C
Hadamard
G
Sum
(Critical)
H
Load
l_extendedprice
l_extendedprice
Map
l_extendedprice * l_discount
F
#threads: plot next slide

Parallel Streaming

- Testing block sizes (#elements); relative to worst case (longest, 1k_elem)





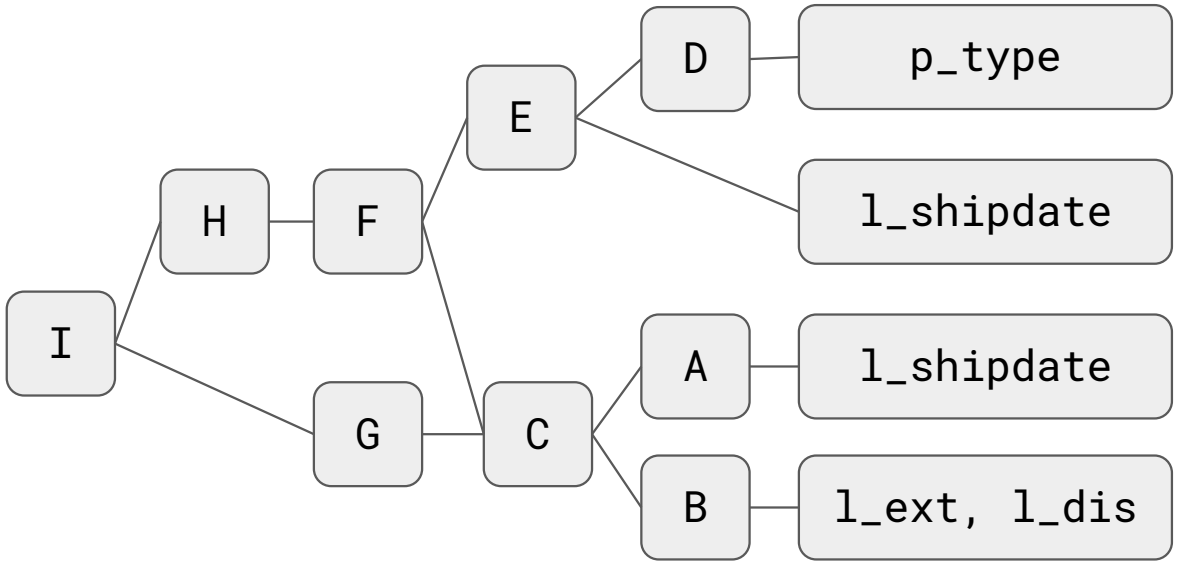
Goals for the meeting

10-05

- DSL to C++ conversion (also see 12 April)
- Dissertation topics
- Define work until July

Q14 LAQ

```
A = filter( lineitem.shipdate >= "1995-09-01" AND lineitem.shipdate < "1995-10-01") // 1 <-- #l
B = lift( lineitem.extendedprice * (1 - lineitem.discount) )                        // 1 <-- #l
C = hadamard( A, B )                                                                // 1 <-- #l
D = filter( match( part.type , ".+PROMO.+" ) )                                      // 1 <-- #p
E = dot( D, lineitem.partkey )                                                      // 1 <-- #l
F = hadamard( C, E )                                                                // 1 <-- #l
G = sum( C )                                                                        // 1 <-- 1
H = sum( F )                                                                        // 1 <-- 1
I = lift( 100.00 * H / G )                                                          // 1 <-- 1
return ( I )                                                                        // RESULT
```



Goals for the meeting

23-05

- Paper deadlines
- Time planning
  - Paper
  - Dissertation

Paper submission info (PODS' 19)

Submitted papers should be **at most twelve pa**
Additional details may be included in an append
at the submission time (online appendices are n
appendices will be read at the discretion of the p

Info source

Acm template

FIRST SUBMISSION CYCLE:
June 15, 2018: Abstract submission
June 22, 2018: Paper submission
August 31, 2018: First notification
September 28, 2018: Revised submission
November 2, 2018: Final notification

SECOND SUBMISSION CYCLE:
December 14, 2018: Abstract submission
December 21, 2018: Paper submission
March 8, 2019: Final notification

All deadlines end at 11:59pm AoE.

Another submission without appendix page limit that has the appendix before bibliography

Goals for the meeting

30-05

- Incremental Query 3 script
- Column store engines
  - MonetDB – install on Search

Incremental Q3 script

```
δA = dot( filter( [c_mktseg.labels | δc_mktseg.labels] ), δc_mktseg )
δB = dot( [A | δA], δo_custkey )
δC = dot( filter( [o_orddate.labels | δo_orddate.labels] ), δo_orddate)
δD = krao( δB, δC )
δE = krao( δD, δo_orderdate )
δF = krao( δE, δo_shippriority )
δG = dot( [F | δF], δl_orderkey )
δH = dot( filter( [l_shipdate.labels | δl_shipdate.labels] ), δl_shipdate )
δI = krao( δG, δH )
δJ = krao( δl_orderkey, δI )
δK = lift( δl_extendedprice * (1 - δl_discount) )
δL = krao( δJ, δK )
δM = sum( δL )
M' = add(M, δM)
return( M' )
```

MonetDB - install on Search

- MonetDB ./configure
  - Error on OpenSSL. **Too old version?**

```
checking for openssl... no
configure: error: OpenSSL library not found but required for MonetDB5
[laolap@compute-662-5 build]$ openssl version -v -b -o
OpenSSL 1.0.0-fips 29 Mar 2010
built on: Thu Aug 23 04:57:41 UTC 2012
options:  bn(64,64) md2(int) rc4(16x,int) des(idx,cisc,16,int) blowfish(idx)
```

When a component is disabled, it is usually due to a missing dependency. A dependency is often a library that the component in question relies on for its functionality. The following two dependencies are known trouble makers on most systems that aren't fairly recent: `openssl`, `libxml2` and `pcre` or `libpcre`. The former is usually available, but a not recent enough version. The latter two often lack the development headers on systems. You have to install missing dependencies either through the native package management system of your host system, or by compiling and installing them manually. Since this in general differs from package to package, we cannot provide generalized help on this topic. Make sure you install the `xxx-dev` versions on binary distributions of the necessary packages as well!

MonetDB - install on Search

- OpenSSL ./configure
  - Error on Perl modules

```
Can't locate Pod/Simple.pm in @INC (@INC contains: . /usr/local/lib64/perl5 /usr/l
sr/lib64/perl5 /usr/share/perl5 .) at /usr/share/perl5/Pod/Text.pm line 33.
BEGIN failed--compilation aborted at /usr/share/perl5/Pod/Text.pm line 33.
Compilation failed in require at /usr/share/perl5/Pod/Usage.pm line 452.
BEGIN failed--compilation aborted at /usr/share/perl5/Pod/Usage.pm line 459.
Compilation failed in require at configdata.pm line 17370.
BEGIN failed--compilation aborted at configdata.pm line 17370.
Compilation failed in require.
BEGIN failed--compilation aborted.
```

MonetDB - install on Search

- Install Perl 5.26.2 from source – **OK**
- Install OpenSSL 1.1 from source– **OK**
- Update system paths – **OK**

- Install MonetDB from source – **OpenSSL not found**

- `../configure --with-openssl=no` … – **OK**
  - Does it affect any important feature?

```
[laolap@compute-781-2 monetdb_opt]$ bin/mclient -d laolap
server requires unknown hash 'SHA512'
```

End of meetings' slides

$$(M \otimes N) \cdot (P \bigtriangledown Q) = (M \cdot P) \bigtriangledown (N \cdot Q)$$

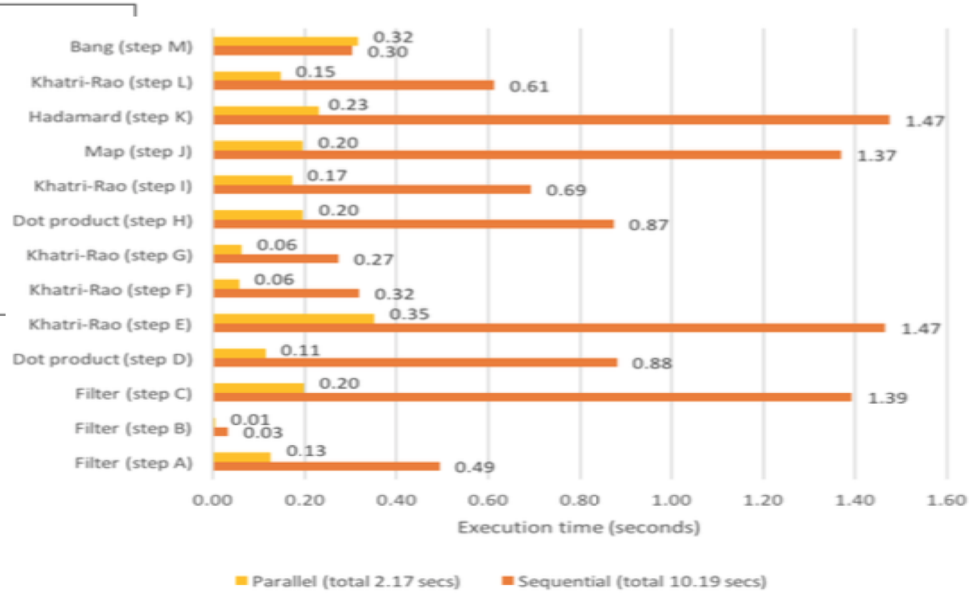
JNO: found this title while reviewing a paper,

Golub, G., Van Loan, C.: Matrix Computations. Johns Hopkins University Press, fourth edn. (2013),

Cf. state-of-the-art etc if not yet there.

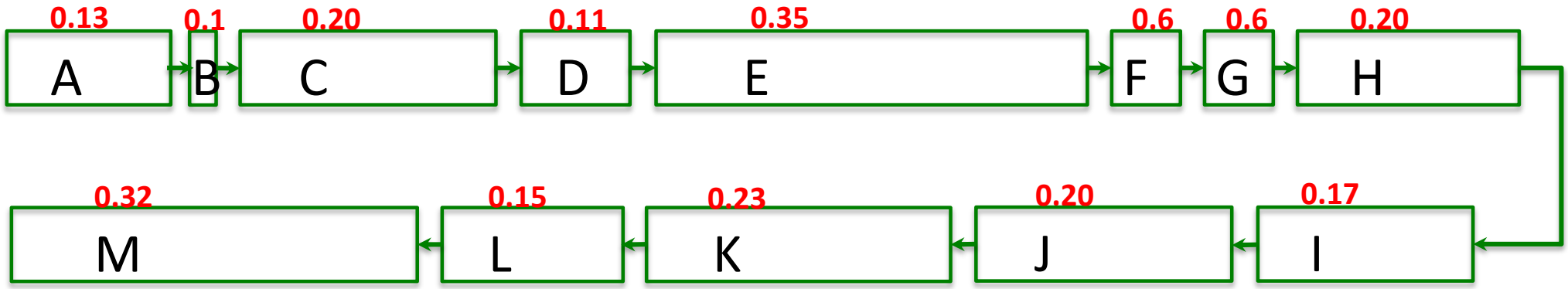
TPC-H Query 3

```
A = filter_to_matrix( o_orderdate, <'1995-03-10' )
B = filter_to_vector( c_mktsegment, ='MACHINERY' )
C = filter_to_vector( l_shipdate, >'1995-03-10' )
D = dot( B, o_custkey )
E = krao( l_orderkey, C )
F = krao( A, D )
G = krao( F, o_shippriority )
H = dot( G, l_orderkey )
I = krao( E, H )
J = map( \x -> 1 - x, l_discount )
K = hadamard( l_extendedprice, J )
L = krao( I, K )
M = bang( sum, L )
```

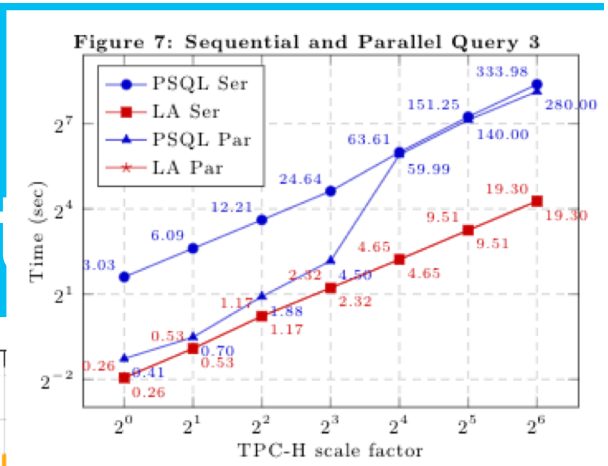


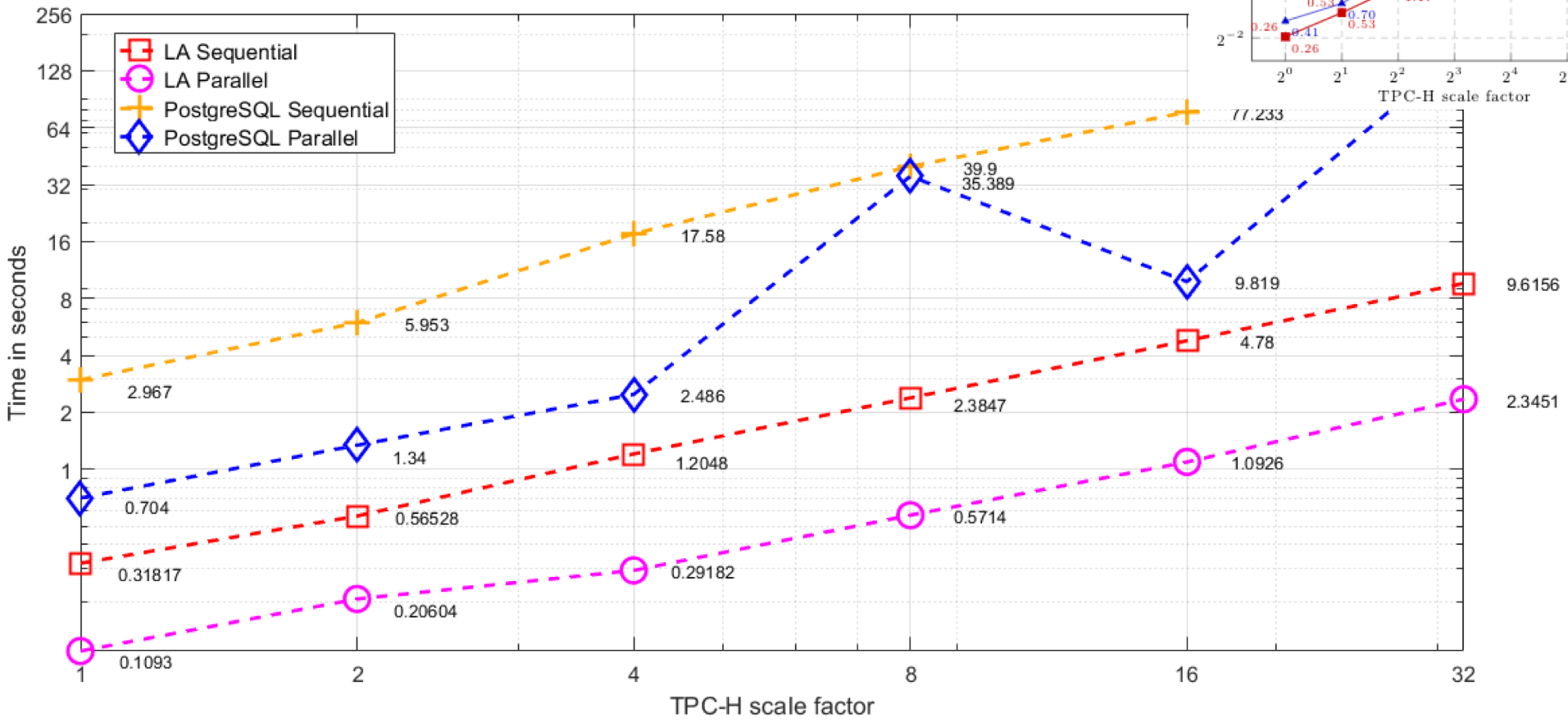
Proposed parallel implementation

Total = 2.17 sec



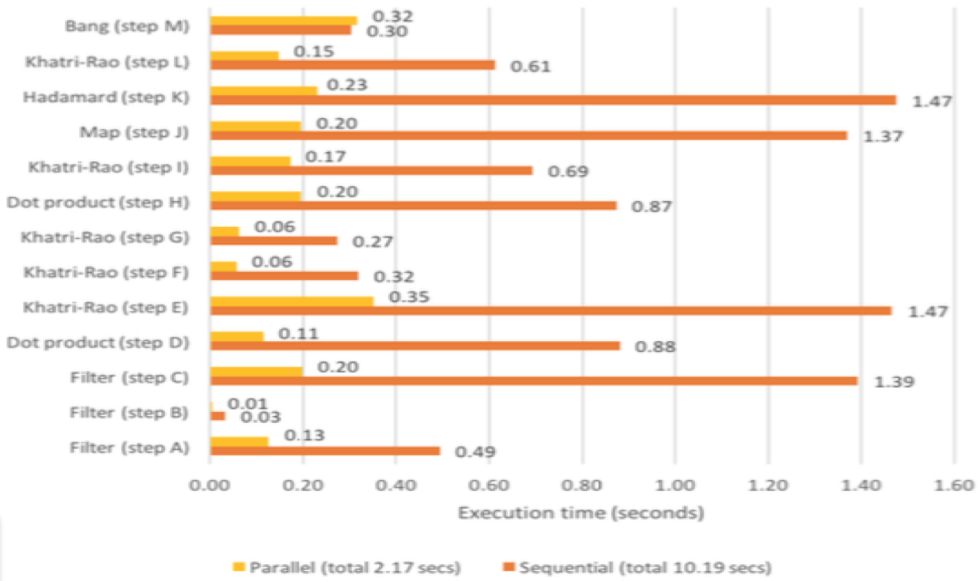
Query 3 performance result





TPC-H Query 3

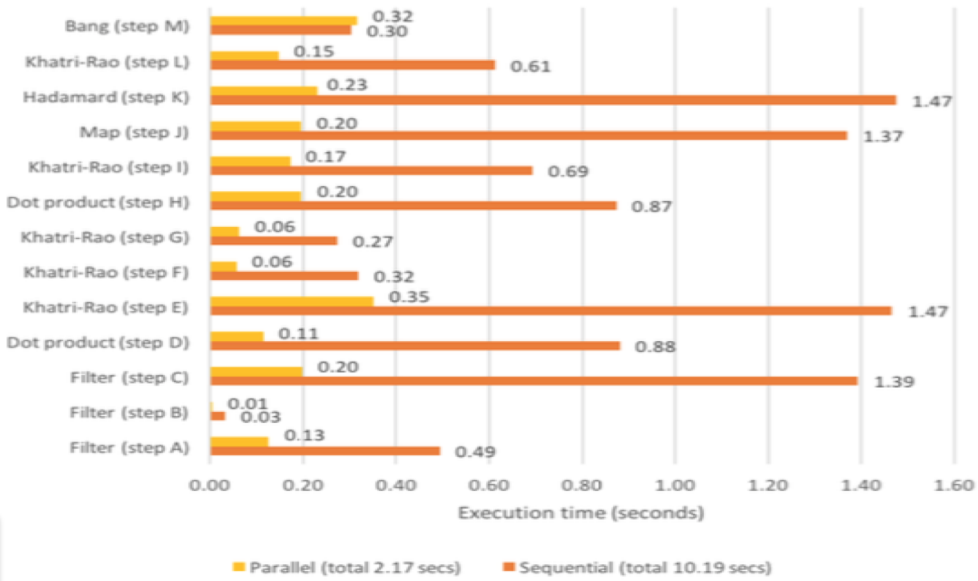
```
A = filter_to_matrix( o_orderdate, <'1995-03-10' )
B = filter_to_vector( c_mktsegment, ='MACHINERY' )
C = filter_to_vector( l_shipdate, >'1995-03-10' )
D = dot( B, o_custkey )
E = krao( l_orderkey, C )
F = krao( A, D )
G = krao( F, o_shippriority )
H = dot( G, l_orderkey )
I = krao( E, H )
J = map( \x -> 1 - x, l_discount )
K = hadamard( l_extendedprice, J )
L = krao( I, K )
M = bang( sum, L )
```



Using HEP-Frame (?): estimated improvement from 2.17 to 1.19 sec

TPC-H Query 3

```
A = filter_to_matrix( o_orderdate, <'1995-03-10' )
B = filter_to_vector( c_mktsegment, ='MACHINERY' )
C = filter_to_vector( l_shipdate, >'1995-03-10' )
D = dot( B, o_custkey )
E = krao( l_orderkey, C )
F = krao( A, D )
G = krao( F, o_shippriority )
H = dot( G, l_orderkey )
I = krao( E, H )
J = map( \x -> 1 - x, l_discount )
K = hadamard( l_extendedprice, J )
L = krao( I, K )
M = bang( sum, L )
```



Without streaming, using HEP-Frame (?): estimated improvement to 1.19 sec

TPC-H Query 3

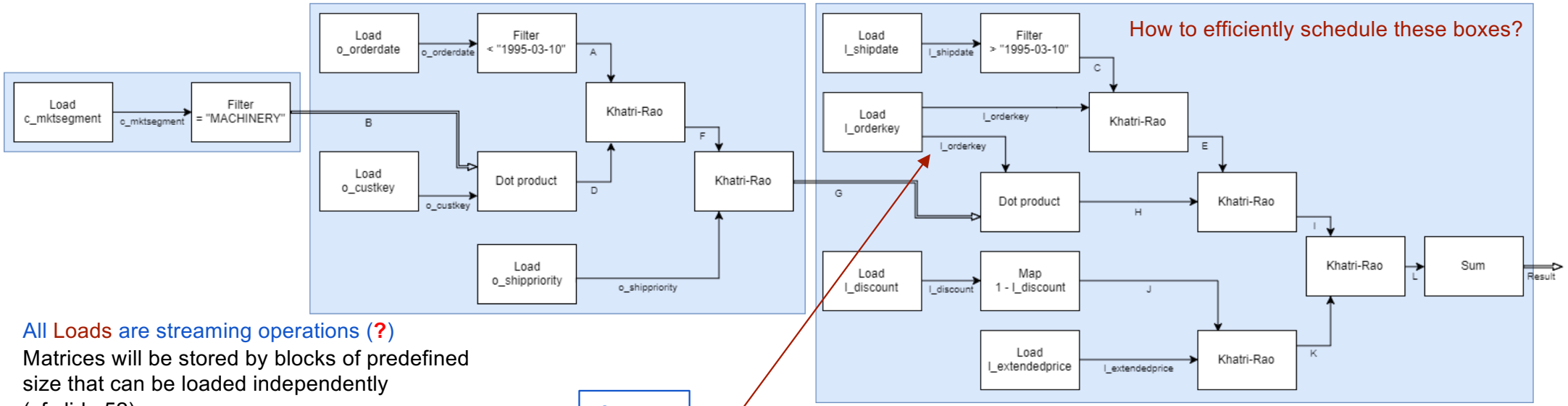
Streaming approach

```
A = filter_to_matrix( o_orderdate, <'1995-03-10' )
B = filter_to_vector( c_mktsegment, ='MACHINERY' )
C = filter_to_vector( l_shipdate, >'1995-03-10' )
D = dot( B, o_custkey )
E = krao( l_orderkey, C )
F = krao( A, D )
G = krao( F, o_shippriority )
H = dot( G, l_orderkey )
I = krao( E, H )
J = map( \x -> 1 - x, l_discount )
K = hadamard( l_extendedprice, J )
L = krao( I, K )
M = bang( sum, L )
```

Vectorized image

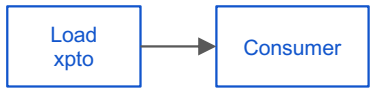
Full matrix data

Streaming data

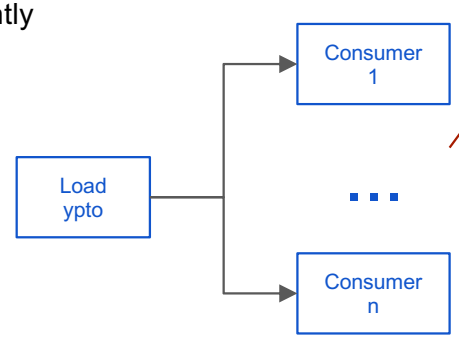


All Loads are streaming operations (?)

Matrices will be stored by blocks of predefined size that can be loaded independently (cf slide 52)



Consumer frees memory



Only last consumer frees memory

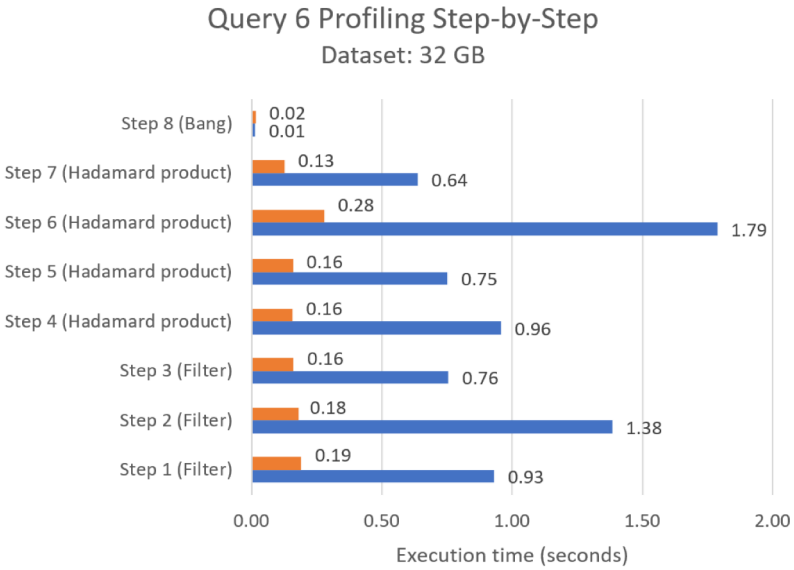
TRUE, and this forks also occur after other operations (not only loads)

E.g.: TPC-H Query 14 after map operation (G)

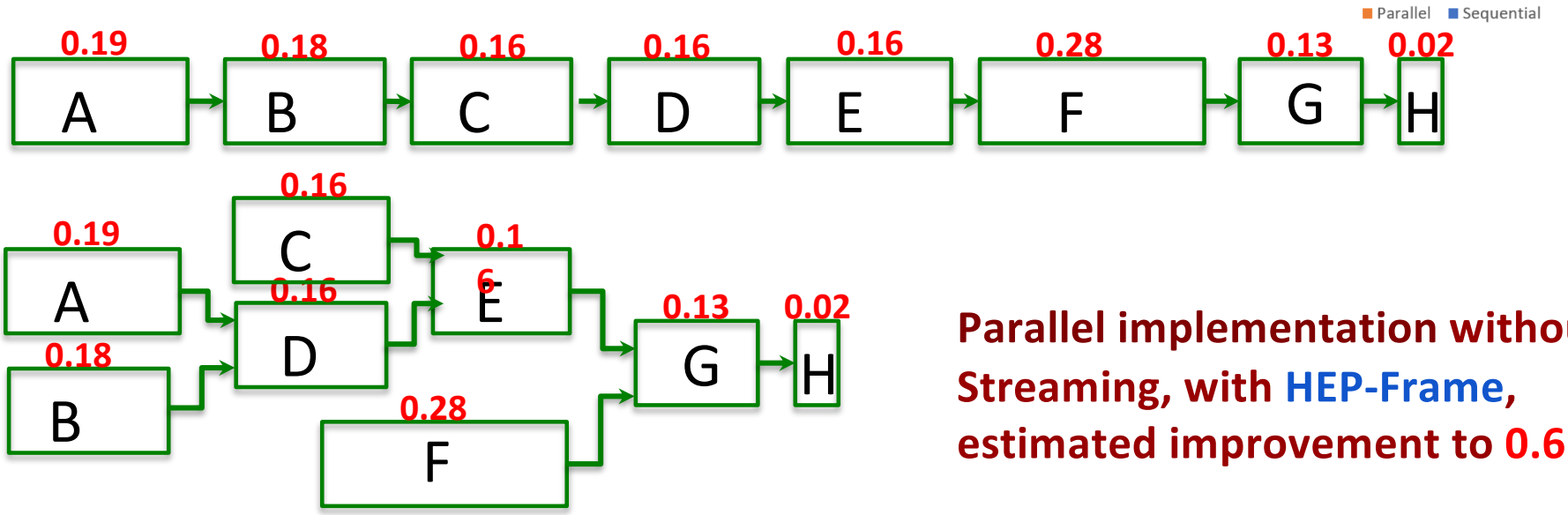
TPC-H Query 6

```
SELECT
        sum(l_extendedprice * l_discount) as revenue
FROM
        lineitem
WHERE
        l_shipdate >= date '1995-03-10'
        and l_shipdate < date '1995-03-10' + interval '1' year
        and l_discount between 0.05 - 0.01 and 0.05 + 0.01
        and l_quantity < 3;
```

```
A = filter_to_vector( l_shipdate, >='1995-03-10', <'1995-03-10')
B = filter_to_vector( l_discount, >='0.49', <='0.51' )
C = filter_to_vector( l_quantity, <='3' )
D = hadamard( A , B )
E = hadamard( C , D )
F = hadamard( l_extendedprice , l_discount )
G = hadamard( E , F )
H = bang( sum, G )
```



Parallel implementation w/out HEP-Frame => 1.28 sec



Parallel implementation without Streaming, with HEP-Frame, estimated improvement to 0.66 sec

TPC-H Query 6

```
SELECT
        sum(l_extendedprice * l_discount) as revenue
FROM
        lineitem
WHERE
        l_shipdate >= date '1995-03-10'
        and l_shipdate < date '1995-03-10' + interval '1' year
        and l_discount between 0.05 - 0.01 and 0.05 + 0.01
        and l_quantity < 3;
```

Cluster node 781.1

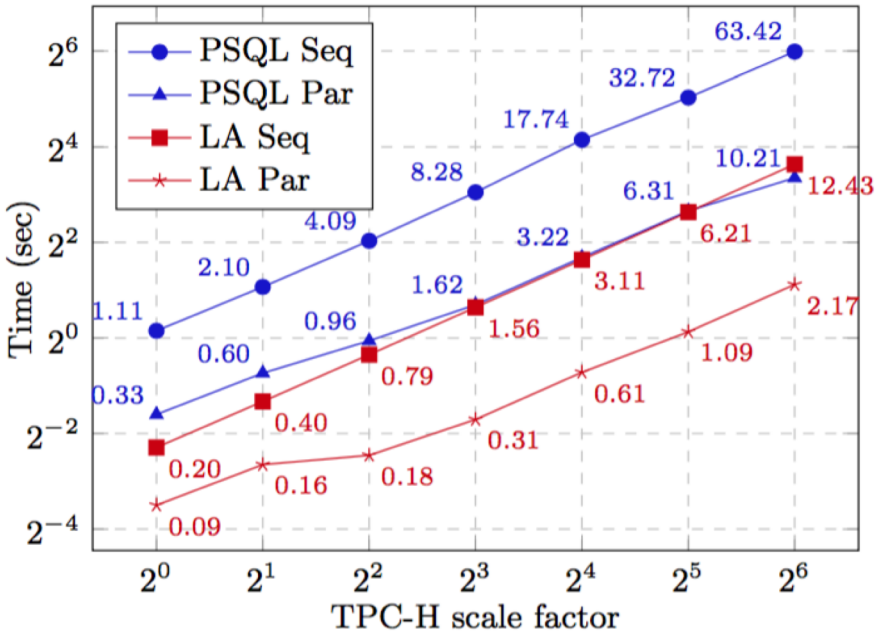
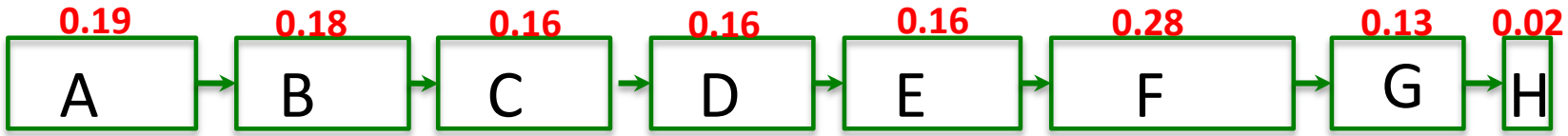


Figure 7: Sequential and Parallel Query 6

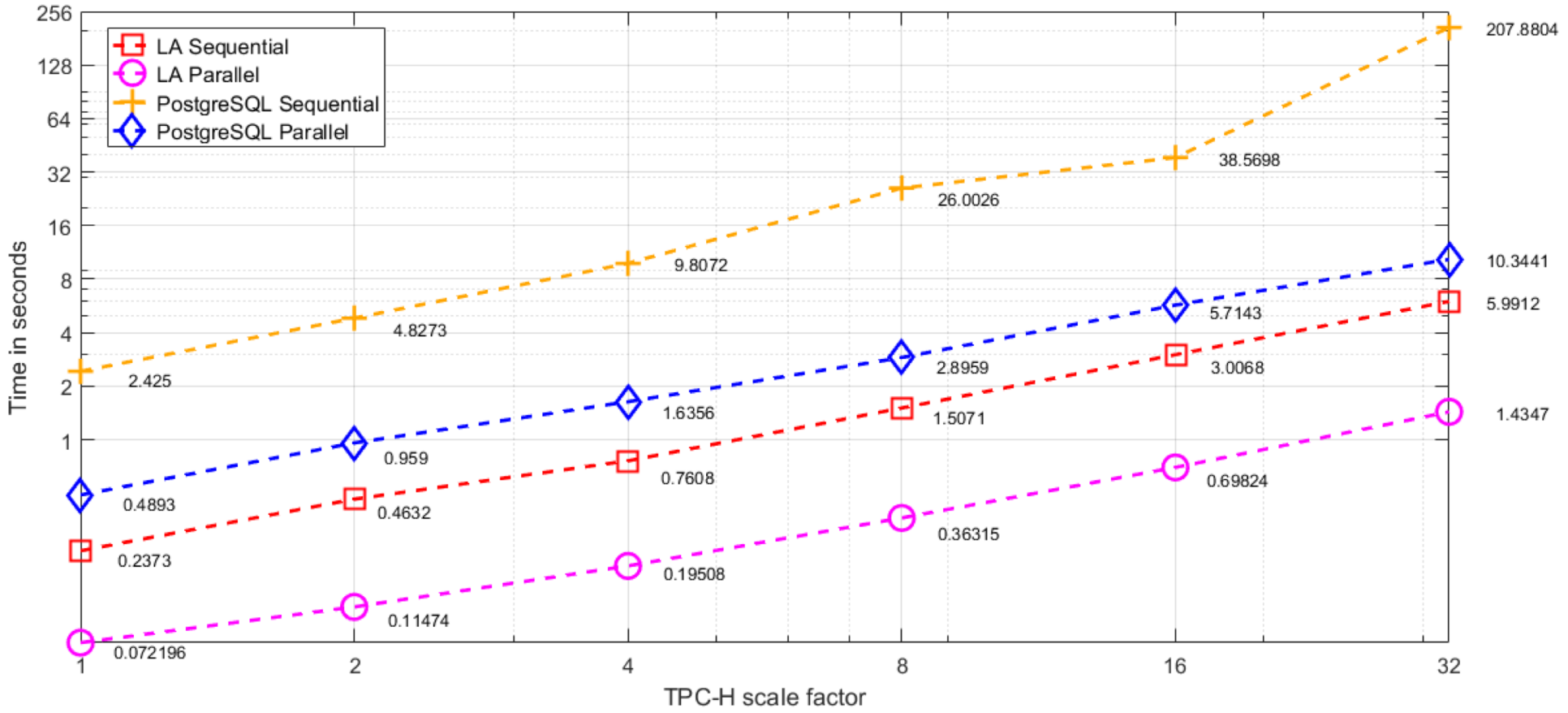
```
A = filter_to_vector( l_shipdate, >='1995-03-10', <'1995-03-10')
B = filter_to_vector( l_discount, >='0.49', <='0.51' )
C = filter_to_vector( l_quantity, <='3' )
D = hadamard( A , B )
E = hadamard( C , D )
F = hadamard( l_extendedprice , l_discount )
G = hadamard( E , F )
H = bang( sum, G )
```

Sequential implementation for scale 2^5 => 6.21 sec

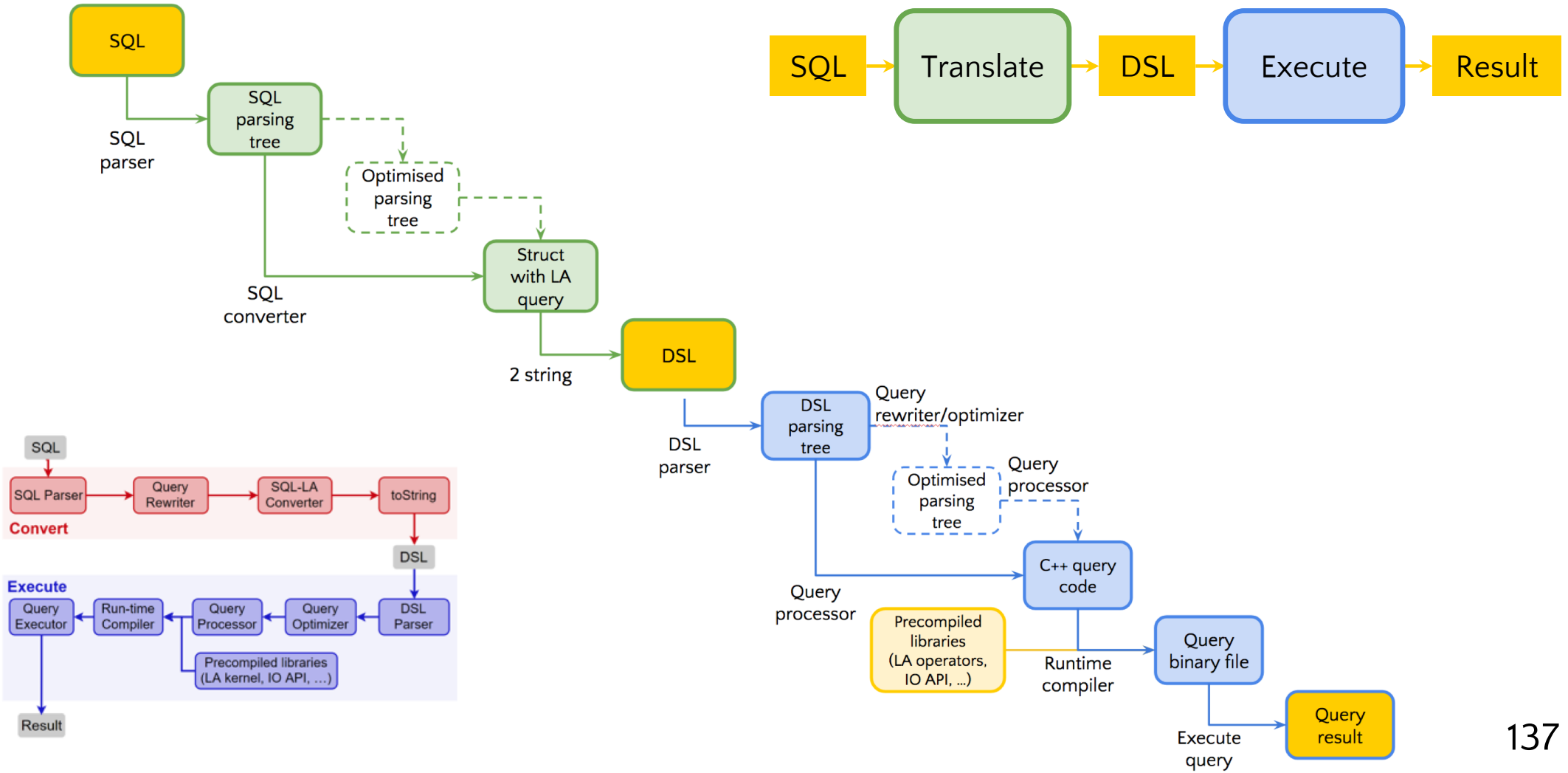
Needs to be updated...



Query 6 performance results



Possible architecture (initial proposal)



RPD: Contents

CONTENTS

Dissertation: Contents

1. Introduction
   a. Challenges & Goals
   b. Dissertation Outline
2. The relational model and OLAP
   a. Data Storage Origins
   b. The Relational Model
   c. Database System Implementation
   d. Data Warehousing
   e. OLAP
      i. ROLAP (current framework)
      ii. MOLAP (future work)
   f. Distributed Databases
   (to introduce distributed LA)
1. Typed linear algebra for OLAP
   a. Linear Algebraic Encoding of Data
   b. Type Diagrams
   c. Linear Algebra Query language
      i. Linear Algebra Operations
   d. From SQL to LAQ
   e. Cubing in Linear Algebra
   f. LA OLAP operations
   (dicing, slicing, roll up, drill down, ...)
4. A TLA DB engine for relational OLAP
   a. Matrix representation
   b. LAQ operations
   c. "Streaming" approach
   d. Controller
   e. SQL Driver
   f. LAQ Engine
5. Validation and performance results
   a. TPC Benchmark H
   b. In-memory
6. **Framework extension**
   a. Incremental querying
   (introduce this in the state of the art?)
   a. Distributed querying
   (already studied distributed LAQ operations)
   a. Future work
4. Conclusion

Sections to be deleted
Possible (new) sections
Comments

Dissertation topics - FUTURE WORK

- Refer data cubes theory in the state of the art ?
- Better thread-level scheduler? (HEP-Frame?)
- Better controller
  - When flush data to disk? ACID vs performance
  - Multi-user/query system
    - Study what should be cached
    - Study data share among queries executed in parallel
    - Study resource allocation
  - JDBC & console APIs
  - Multi-node system
    - Replicate / distribute data
    - Reduce communications on querying & minimize data transferences