THOUGHTS ON DATA PROCESSING AND DATA ANALYSIS

Giulio Eulisse (CERN)

DISCLAIMER

*This is to **trigger discussions**, not a plan.*

*What follows is **not approved** by the O2 Technical Board and just started discussions in various groups, including WP4 (Framework), WP1 (Data Model) and WP14 (Analysis Framework & Facilities).*

*Different technologies shown will of course need to go under **accurate evaluation** of features / performance.*

***Nevertheless**, I personally think there is some merit in what I am about to show.*

DATA PROCESSING LAYER: A REMINDER

Idea: encode a strict (O2 specific) data oriented pipelines on top of ALFA.

As trade-off, exchange:

- *Flexibility when building topologies.*

- *Requirement on users to specify inputs, outputs and parameters.*

to gain:

- *Simplified, homogeneous programming environment, with limited distributed system hiccups.*

- *Automatic topology generation, optimisation and deployment.*

- *Debug tools.*

DATA PROCESSING LAYER: LATEST DEVELOPMENTS

Improved ROOT support

Thanks to Matthias and Mikolaj efforts, exchanging ROOT messages is now more flexible, with the ability to exchange generic (non-TObject derived) objects and to separate schema information from the actual payload.

TPC Reconstruction chain

Mathias is adapting David's TPC reconstruction to integrate with the DPL.

Improved support for O2 simulation

Following feedback from Sandro, the DPL now provides support for ***gsl::span*** *which is widely used in his Simulation efforts.*

Integration Data Sampling

Thanks to the efforts of Piotr, we are now adapting the QC prototype on top of the DPL.

Ongoing efforts to integrate the Monitoring package

Thanks to the efforts of Adam, the Monitoring package is being made available as a "Service" of the DPL.

LET'S TRY TO DO THE SAME EXERCISE WITH ANALYSIS

Can we have a similar approach for data analysis?

What tradeoffs are important there?

How can we imagine doing it, concretely?

- What's the memory layout?
- How do we do disk persistency?
- How do we interface with ROOT and other analysis packages

CONSTRAINTS ON THE USER

Similarly to what happens in the Data Processing Layer, we should put some rules on what the analyser can do:

Simplified schema description: *Move away from generic C++ objects to adopt more "Dremel" (https://research.google.com/pubs/pub36632.html) like schema primitives: allow only simple types, repeated records (i.e. vectors), nested records (i.e. dictionaries), mostly organised in column-wise tables or nested records.*

Input filter declarations: *analysers get data by defining **filters** on the input collections, e.g. "Give me all the Vertices with $z > 0$".*

Processing on whole objects: *analysers still would get the Vertex "iterator", not just one "Vertex.z" column. This should be derivable automatically from the filter. Data will be however chunked.*

Output declarations: *analysers will have to declare their output columns or book histograms from an "histogram service".*

No pointers: *references through index tables or bitmaps. Of course this should be made transparent to the user.*

CONSTRAINTS: SIMPLIFIED SCHEMA

♣	♠	♥
1	0.1	TRUE
2	0.2	FALSE
3	0.3	TRUE
4	0.3	FALSE

Let's say we allow for tables of basic types, stored as columns...

CONSTRAINTS: SIMPLIFIED SCHEMA

♣	♠	♥	♦
1	0.1	TRUE	"One"
2	0.2	FALSE	"Two"
3	0.3	TRUE	"Three"
4	0.4	FALSE	"Four"

…and we actually allow for repeated basic types in a column, e.g. strings…

CONSTRAINTS: SIMPLIFIED SCHEMA

❖

♣	♠	♥	♦
1	0.1	TRUE	[1]
2	0.2	FALSE	[1,2]
3	0.3	TRUE	[1,2,3]
4	0.4	FALSE	[1,2,3,4]

…or vectors…

CONSTRAINTS: SIMPLIFIED SCHEMA

❖

♣	♠	♥	♦begin	♦end
1	0.1	TRUE	0	1
2	0.2	FALSE	1	3
3	0.3	TRUE	3	6
4	0.4	FALSE	6	10

	♦
0	1
1	1
2	2
3	1
4	2
5	3
6	1
7	2
8	3
9	4

... which are actually (transparently) stored as ranges in a separate column...

CONSTRAINTS: SIMPLIFIED SCHEMA

♣	♠	♥	♦
1	0.1	TRUE	"One"
2	0.2	FALSE	NIL
3	0.3	NIL	"Three"
4	0.4	FALSE	"Four"

...let's allow "empty" values...

CONSTRAINTS: SIMPLIFIED SCHEMA

❖

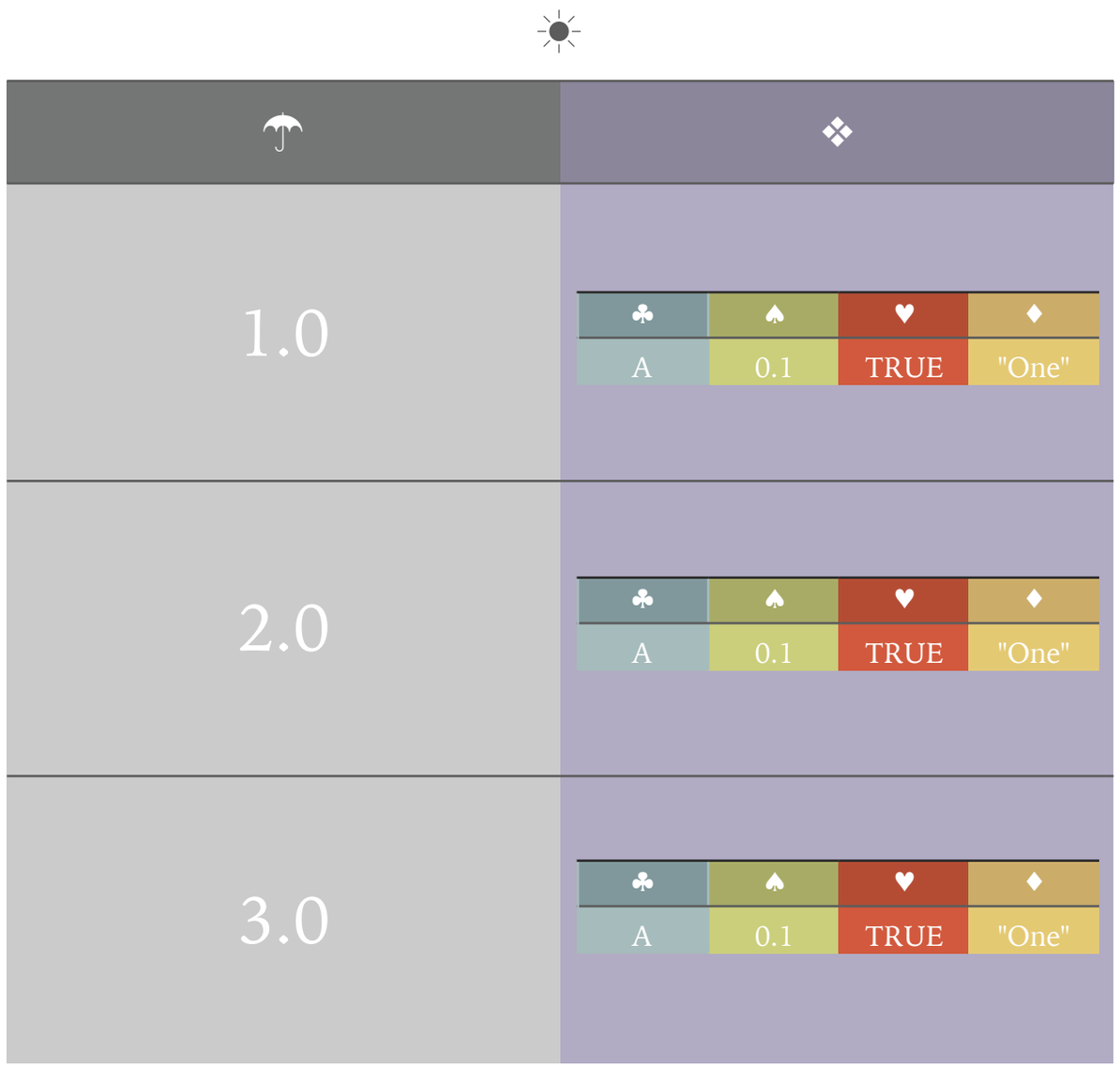
♣	♠	♥	♦
1	0.1	TRUE	"One"
2	0.2	FALSE	NIL
3	0.3	NIL	"Three"
4	0.4	FALSE	"Four"

❖ ok

♥	♦
1	1
1	0
0	1
1	1

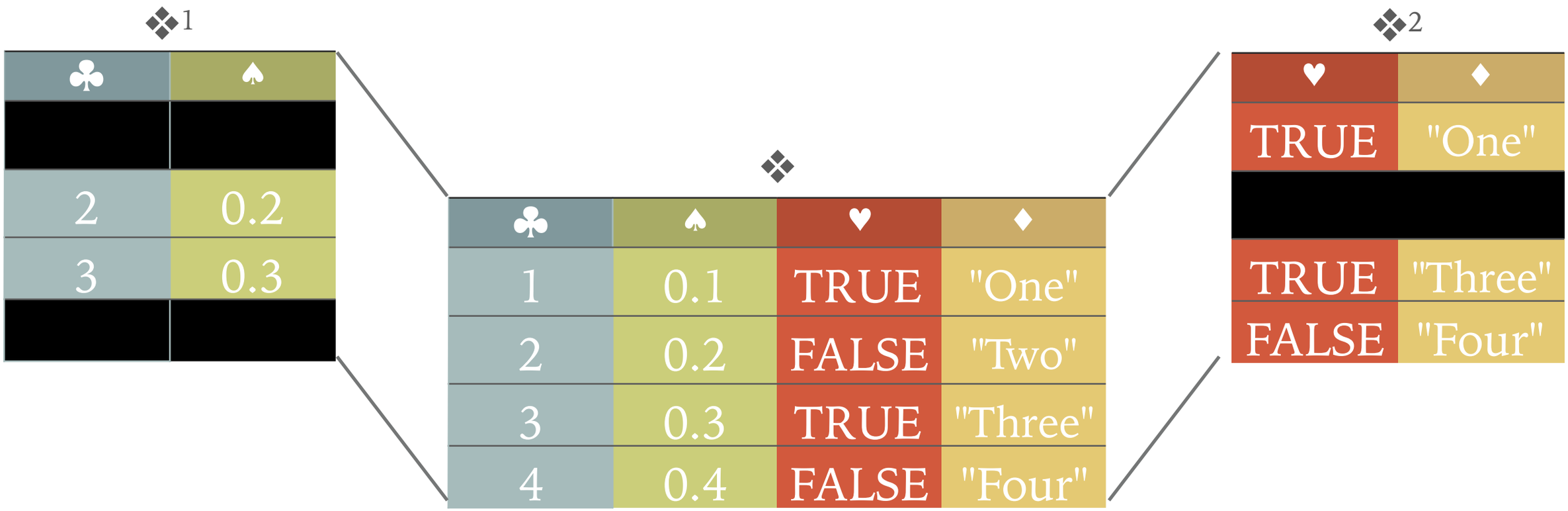
…(transparently) stored as bitmaps…

CONSTRAINTS ON THE SCHEMA



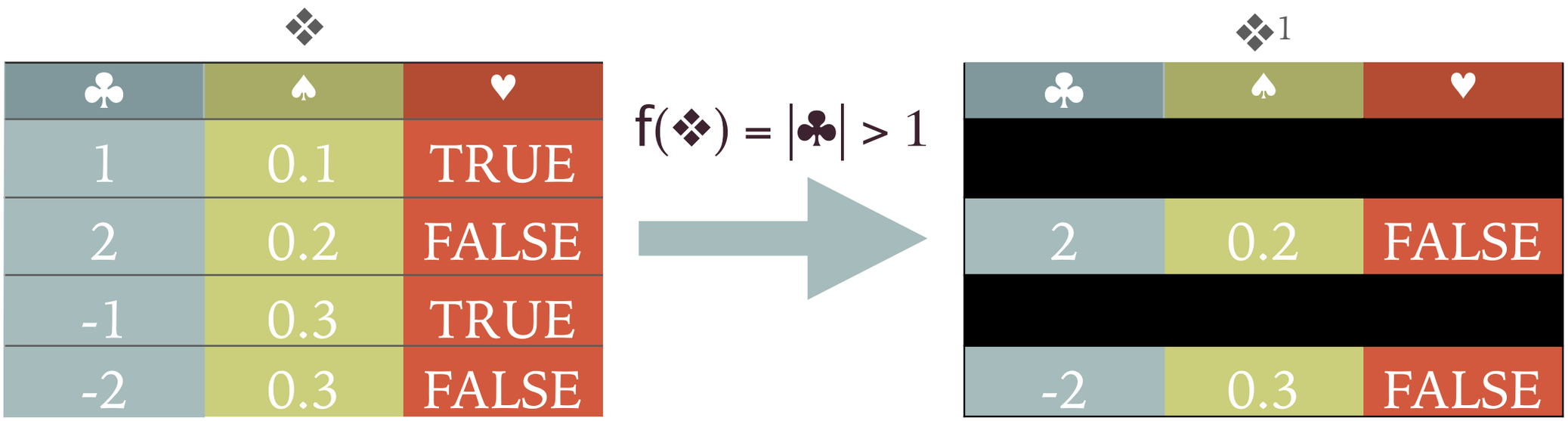
...let's allow nested records...

CONSTRAINTS: SIMPLIFIED SCHEMA



...let's have zero-copy "views", where maybe some rows have been removed...

CONSTRAINTS: INPUT FILTERS



...let's have the user define filters...

CONSTRAINTS: INPUT FILTERS

❖

♣	♠	♥
1	0.1	TRUE
2	0.2	FALSE
-1	0.3	TRUE
-2	0.3	FALSE

$f(❖) = |♣| > 1$

❖

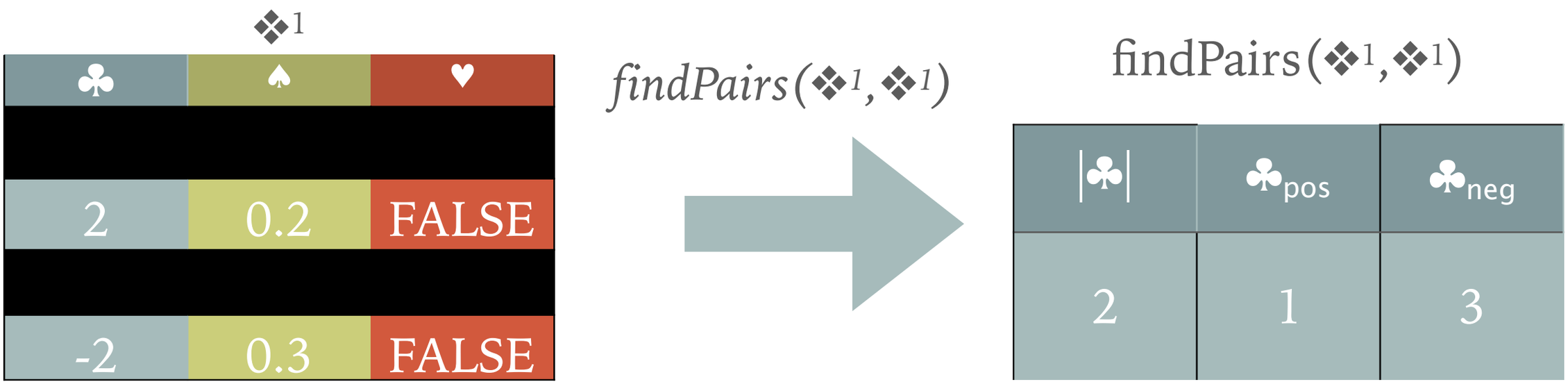
♣	♠	♥
1	0.1	TRUE
2	0.2	FALSE
-1	0.3	TRUE
-2	0.3	FALSE

+

$♣_{ok}$
0
1
0
1

...which are nothing else but views on the original data...

CONSTRAINTS: DECLARE OUTPUTS

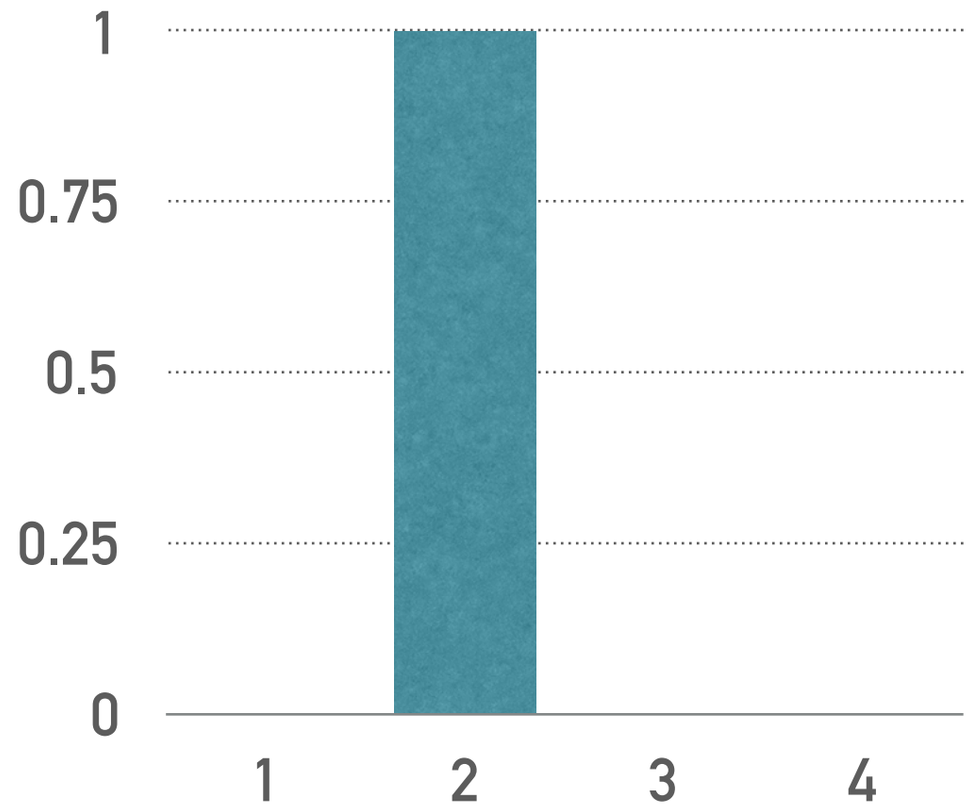


...let users define associations...

CONSTRAINTS: DECLARE OUTPUTS

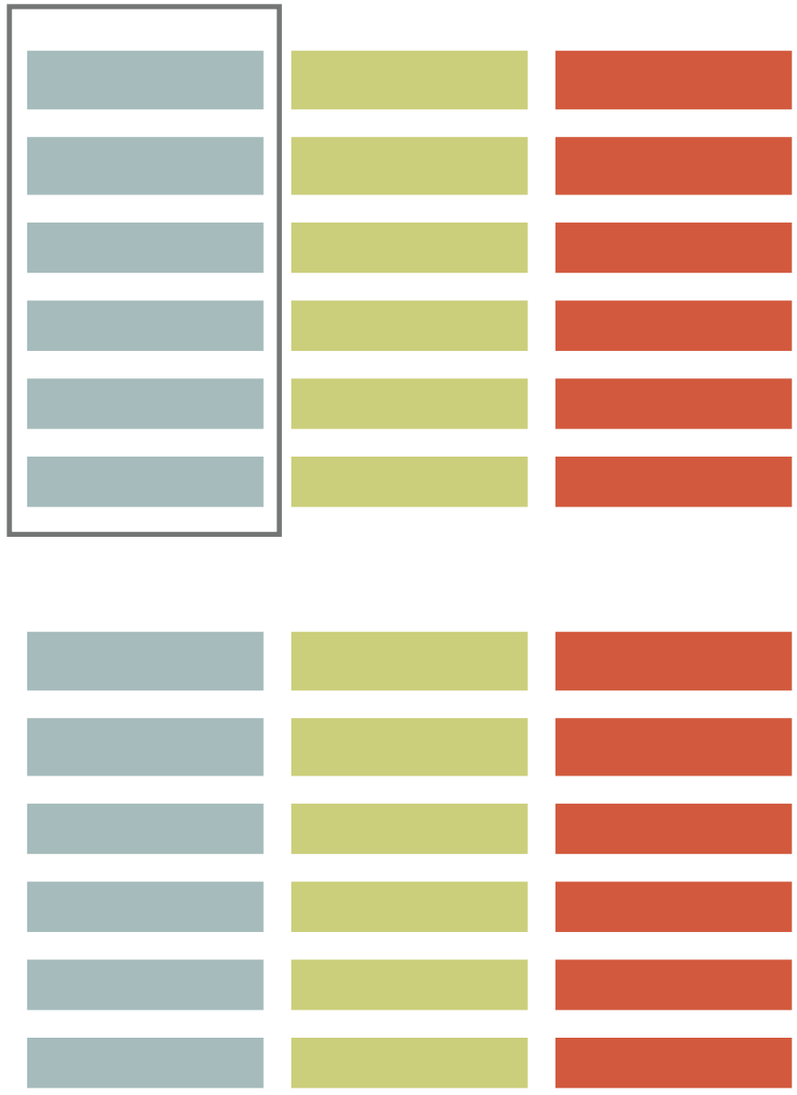
findPairs(❖[1],❖[1])

|♣|	♣pos	♣neg
2	1	3



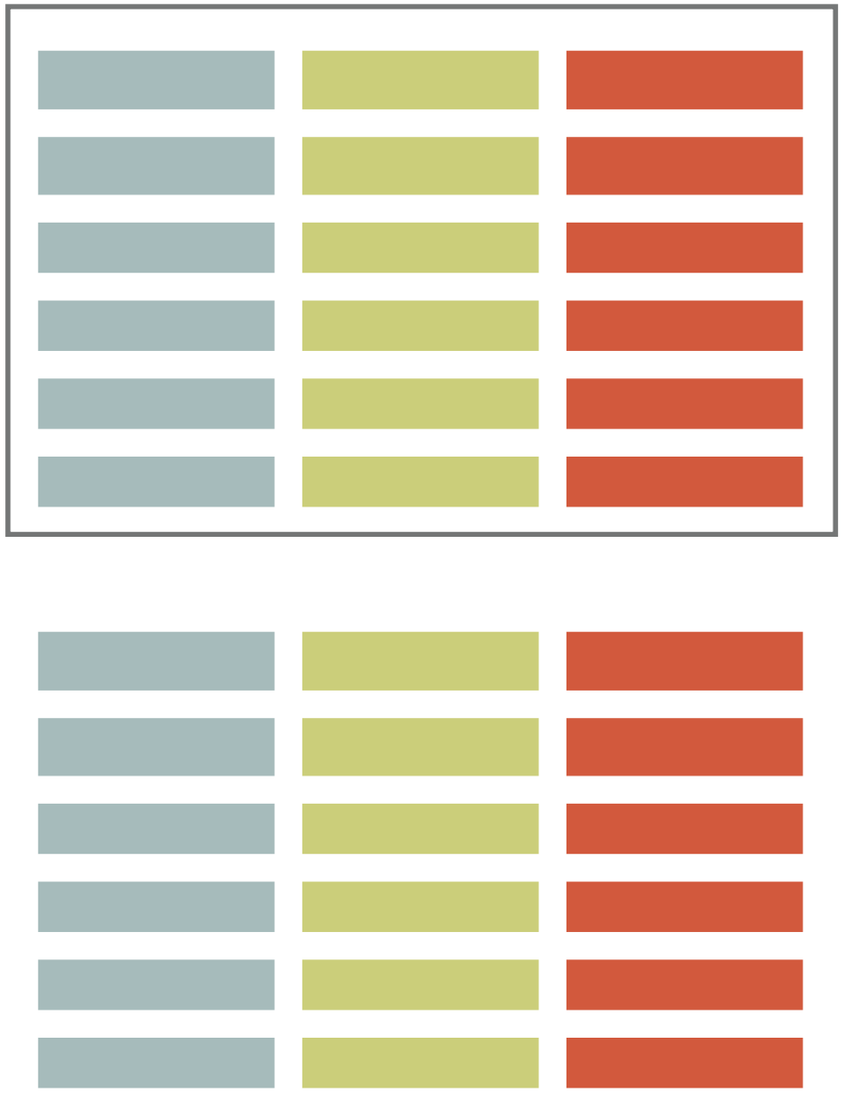
…fill some histogram and publish a paper…

IN MEMORY LAYOUT: BATCHES OF COLUMN WISE DATA



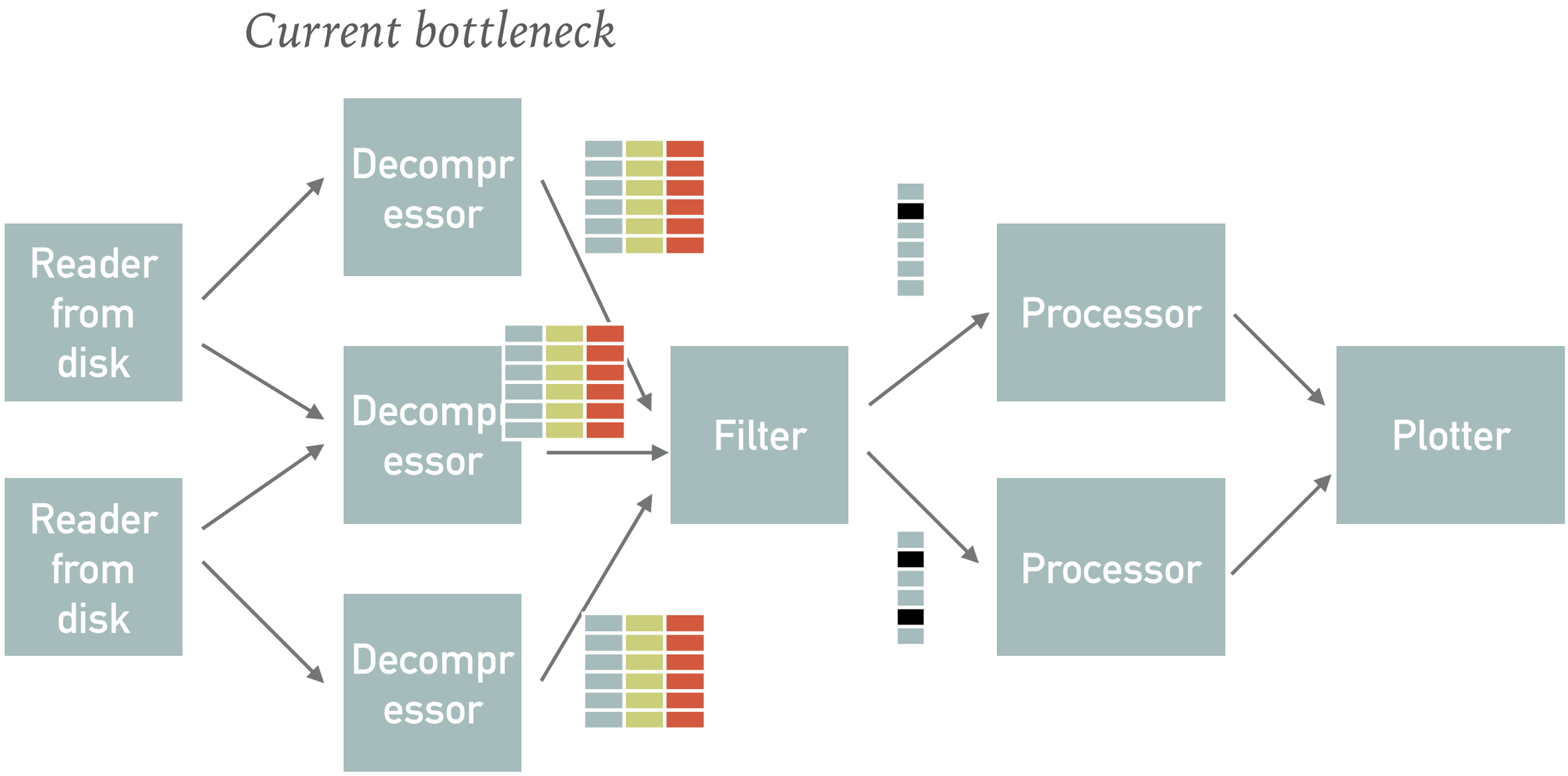
Column-wise storage facilitates vectorisation, improves compression ratios.

IN MEMORY LAYOUT: BATCHES OF COLUMN WISE DATA



chunking data which is read in any case by one of the analysers makes smaller working set, allows incremental processing.

THE FAIRMQ BIT: A POSSIBLE TOPOLOGY



… can be scaled out by adding extra devices (assuming multiple cores)…

OK, BUT HOW DO WE DO IT?

"Isn't that a lot of code?"

"Does it work with shared memory?"

"Do we need to invent our own memory layout?"

"How do we integrate our system with pandas?"

"I want to do machine learning with Tensorflow!"

"How do we integrate with ROOT?"

"Can we try with Snappy compression library?"

"Do we need to invent our persistent storage format?"

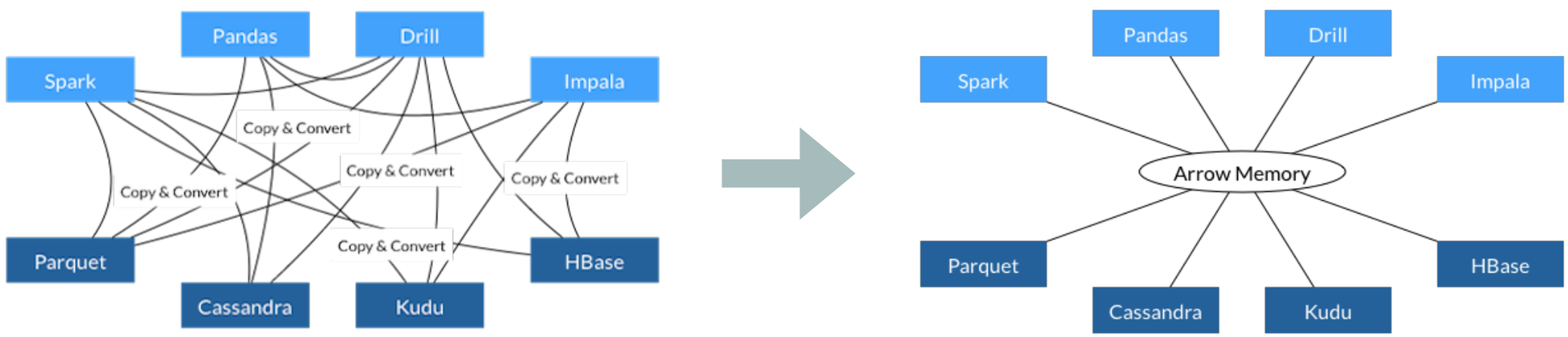
"Can we do zero-copy message passing?"

"Good artists copy, great artists steal."

Pablo Picasso

APACHE ARROW

"Cross-language development platform for in-memory columnar data."



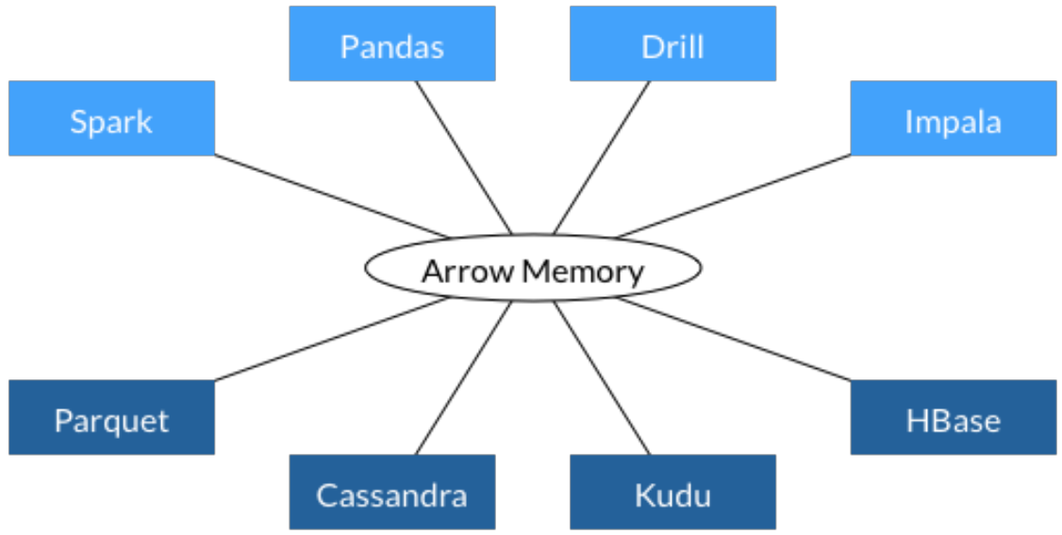
Well established. *Top-Level Apache project backed by key developers of a number of opensource projects:* ***Calcite****, Cassandra, Drill, Hadoop, HBase, Ibis, Impala, Kudu,* ***Pandas****,* ***Parquet****, Phoenix,* ***Spark****, and Storm.*

Very active. *119 contributors, https://github.com/apache/arrow*

O2 design friendly. *message passing / shared memory friendly. Support for zero-copy slicing, filtering.*

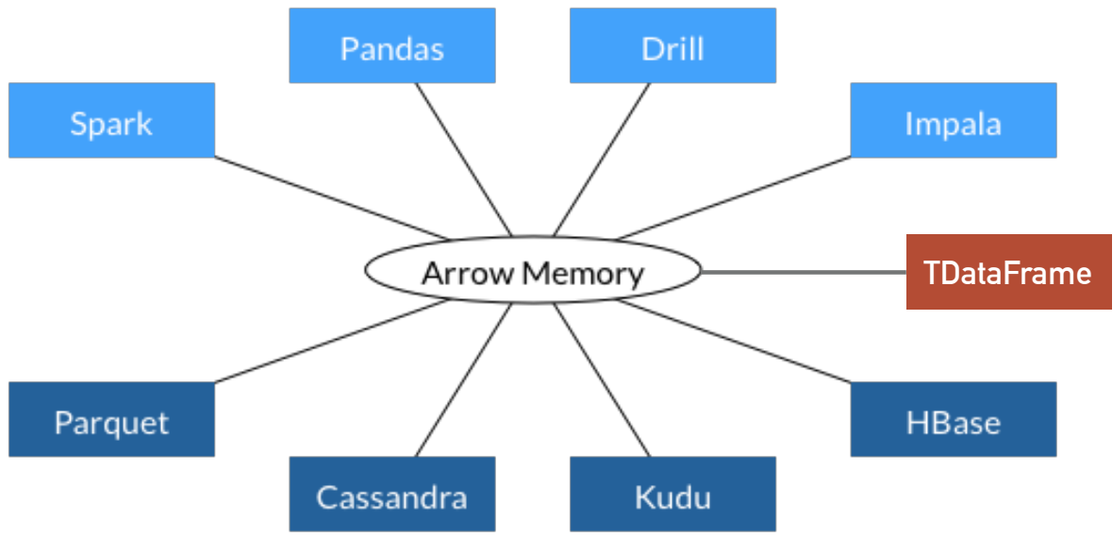
APACHE ARROW: INTEGRATION WITH ROOT

The main concern here is of course "how do I use this from ROOT"?



APACHE ARROW: INTEGRATION WITH ROOT

The main concern here is of course "how do I use this from ROOT"?



TDataFrame: *nice introduction earlier today by Danilo. Arrow fits naturally as a "TDataSource".*

Show me the code! *https://github.com/root-project/root/pull/1712*

Bonus: *ROOT gets seamless integration with many OpenSource projects which you can mention to impress your friends and that make your CV look good to head-hunters.*

WHAT IS TO BE GAINED? WHAT ARE THE GOALS?

Query optimisation: *by writing an analysis in form of a filter - emit - reduce query, the system has the ability to optimise it's execution. "All vertices with Vertex.z > 1" should run on "All vertices with Vertex.z > 0".*

Remove / minimise serialisation, scale decompression.

Allow vectorised processing. *Not a big deal at the moment (bottleneck is deserialisation). Might become relevant in the future?*

Homogeneity with the rest of the system: *analysis wagons become yet another device / data processor in the O2 framework.*
Integration with other system: *using established OpenSource components add complications, but potentially simplifies long term sustainability and integration with other tools.*

BUILDING A D(A)PL: NEXT STEPS

Embrace Apache Arrow *in the Data (Analysis) Processing Layer as top-level message format.*

- *Zero Copy, row-wise POD ⇒ gsl::span API*
- *Histograms and serialisable objects ⇒ ROOT*
- *Zero Copy, (chunked) column-wise data ⇒ (future) Apache Arrow API*

Prototype AOD schema *in collaboration with WP1 and WP14, also profiting from the work done by Roel and Michele in the last few years.*

Write a source device *which reads RUN1 / RUN2 AOD data to the above format.*

Evaluate the solution *with trivial and non-trivial analysis adapting it to use TDataFrame while doing it.*

Iterate.