brytlyt

GPU Accelerated Data Processing

Speed of Thought Analytics at Scale

S9373 - TPC-H Benchmark on DGX-2
A New Paradigm for OLAP and Decision Support

Key pain points

Flexibility

43%

Of analysts say their analytics is not flexible enough to meet their needs

Performance

32%

Of analysts say they have to deal with slow query speeds

The reason data insights is so challenging is analytics solutions today simply do not have the speed, flexibility, and ease of use to answer the data questions people are asking.

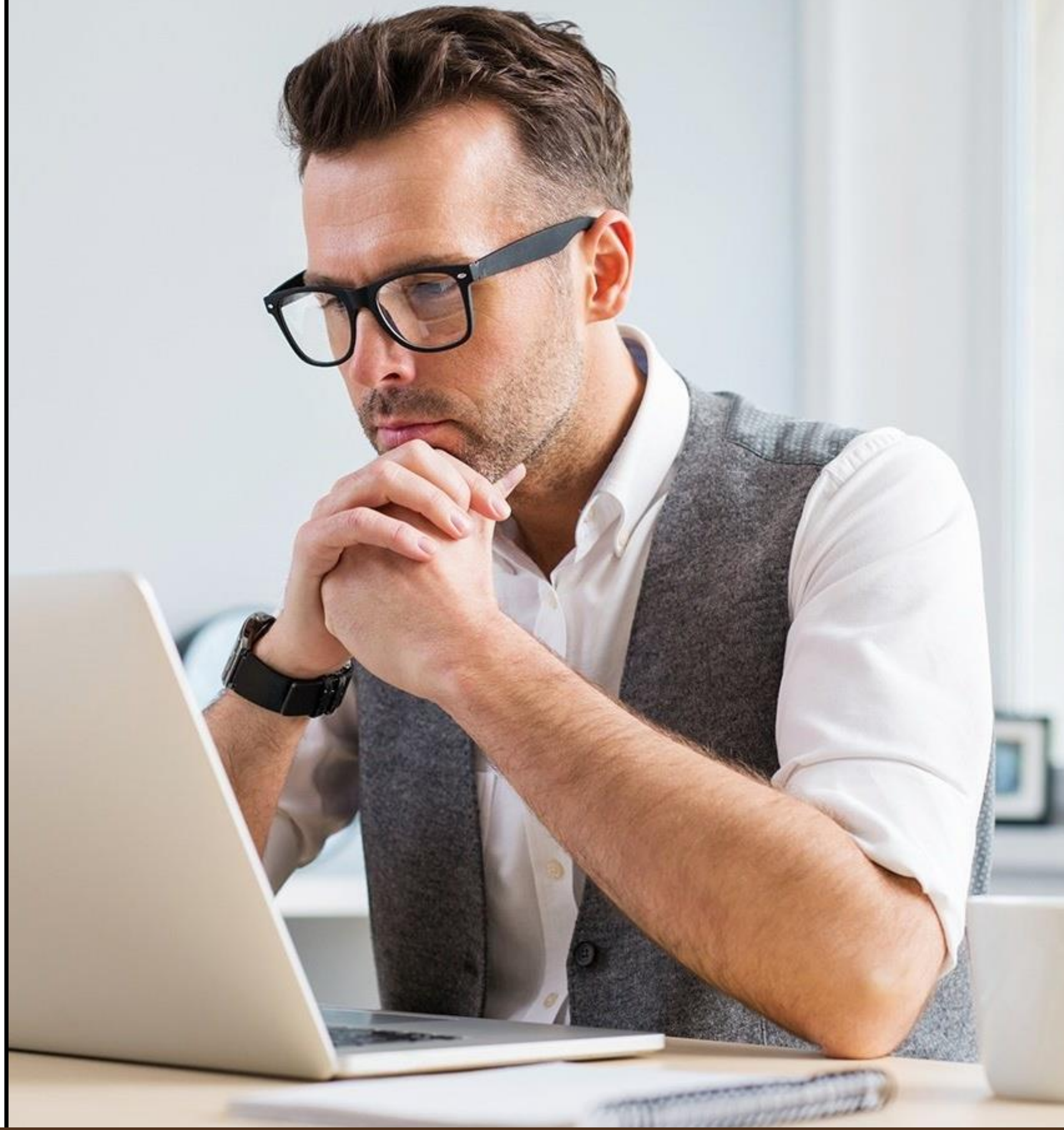
Where are analysts spending time?

64% of time is spent cleaning and organizing data

3 days per month is spent mining data for patterns or refining algorithms

37% of insight takes more than a week

SQL is the most common technology used ahead of Hadoop, Python and R



The fastest, most advanced GPU database on the market

Our mission is to empower organisations through **Speed of Thought Analytics.**

- The world’s fastest database according to independent benchmarking.
- Four years in research and development.
- Only vendor to have patent pending IP for JOINs.
- Fourth generation GpuManagner bridges the gap between SQL and AI.

The true value of Brytlyt lies in how this extreme performance is package for the end user.

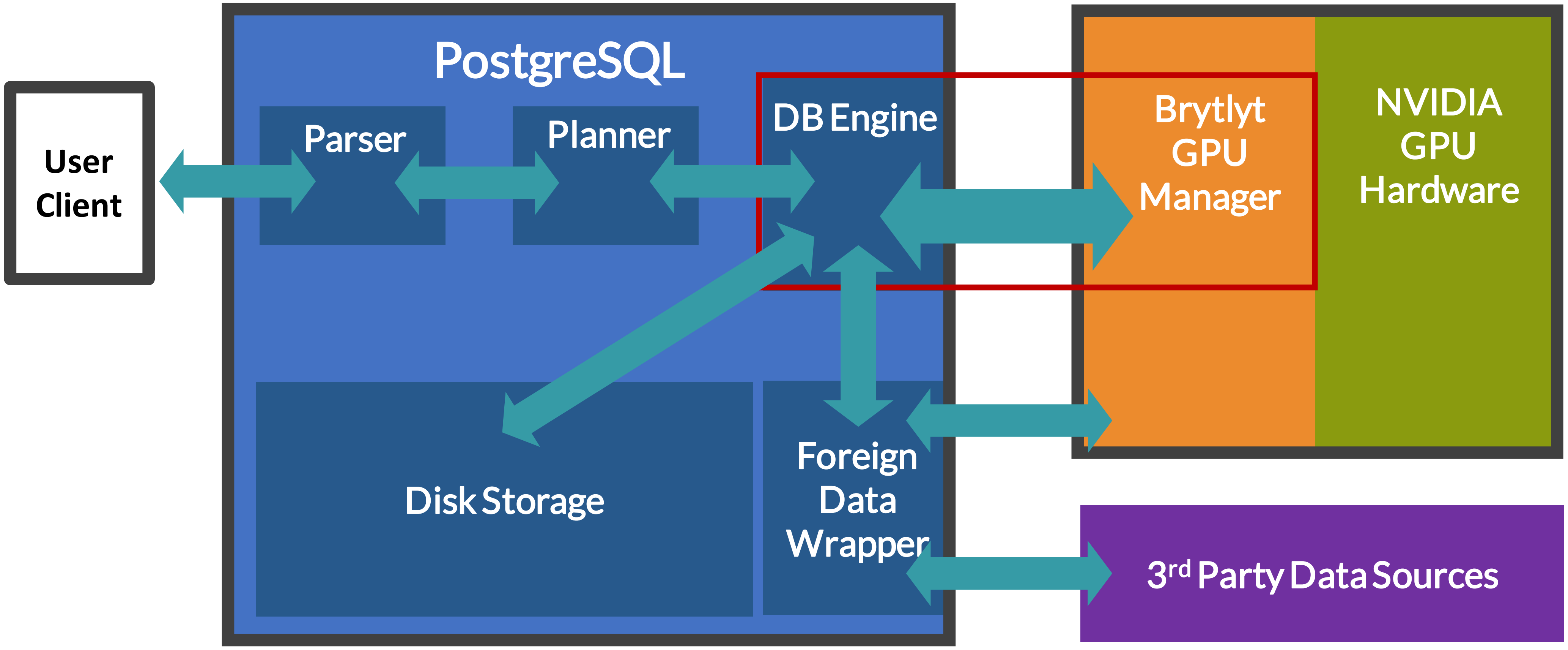
1.1 Billion Taxi Rides Benchmark

Mark Litwintschik Follow

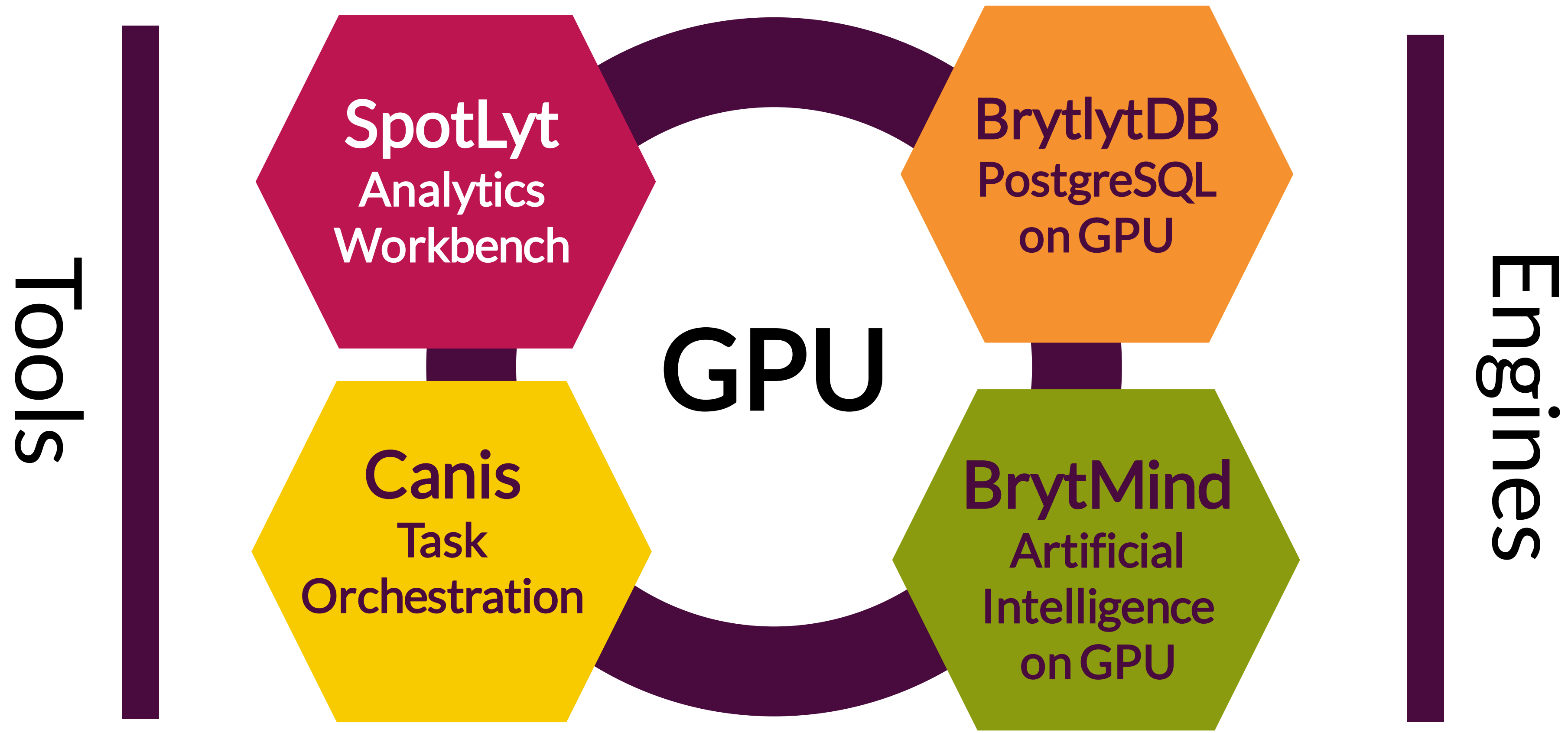
I have 15 years of consulting & hands-on build experience with clients in the UK, USA, Sweden, Ireland & Germany. Past clients include Bank of America Merrill Lynch, Blackberry, Bloomberg, British Telecom, Ford, Google, ITV, LeoVegas, News UK, Pizza Hut, Royal Bank of Scotland, Royal Mail, T-Mobile, TransferWise, Williams Formula 1 & UBS. I hold both a Canadian and a British passport. My CV & my LinkedIn profile.

Query 1	Query 2	Query 3	Query 4	Setup
0.005	0.011	0.103	0.188	BrytlytDB 2.1 & 5-node IBM Minsky cluster
0.009	0.027	0.287	0.428	BrytlytDB 2.0 & 2-node p2.16xlarge cluster
0.021	0.053	0.165	0.51	MapD & 8 Nvidia Pascal Titan Xs
0.027	0.083	0.163	0.891	MapD & 8 Nvidia Tesla K80s
0.028	0.2	0.237	0.578	MapD & 4-node g2.8xlarge cluster
0.034	0.061	0.178	0.498	MapD & 2-node p2.8xlarge cluster
0.036	0.131	0.439	0.964	MapD & 4 Nvidia Titan Xs
0.051	0.146	0.047	0.794	kdb+/q & 4 Intel Xeon Phi 7210 CPUs
0.762	2.472	4.131	6.041	BrytlytDB 1.0 & 2-node p2.16xlarge cluster
1.034	3.058	5.354	12.748	ClickHouse, Intel Core i5 4670K
1.56	1.25	2.25	2.97	Redshift, 6-node ds2.8xlarge cluster
2	2	1	3	BigQuery
4	4	10	21	Presto, 50-node n1-standard-4 cluster
4.88	11	12	15	Presto 0.188 & 21-node m3.xlarge cluster
6.41	6.19	6.09	6.63	Amazon Athena
8.1	18.18	n/a	n/a	Elasticsearch (heavily tuned)
10.19	8.134	19.624	85.942	Spark 2.1, 11 x m3.xlarge cluster w/ HDFS
11	10	21	31	Presto, 10-node n1-standard-4 cluster
11	14	16	22	Presto 0.188 & single i3.8xlarge w/ HDFS
14.389	32.148	33.448	67.312	Vertica, Intel Core i5 4670K
22	25	27	65	Spark 2.3.0 & single i3.8xlarge w/ HDFS
28	31	33	80	Spark 2.2.1 & 21-node m3.xlarge cluster

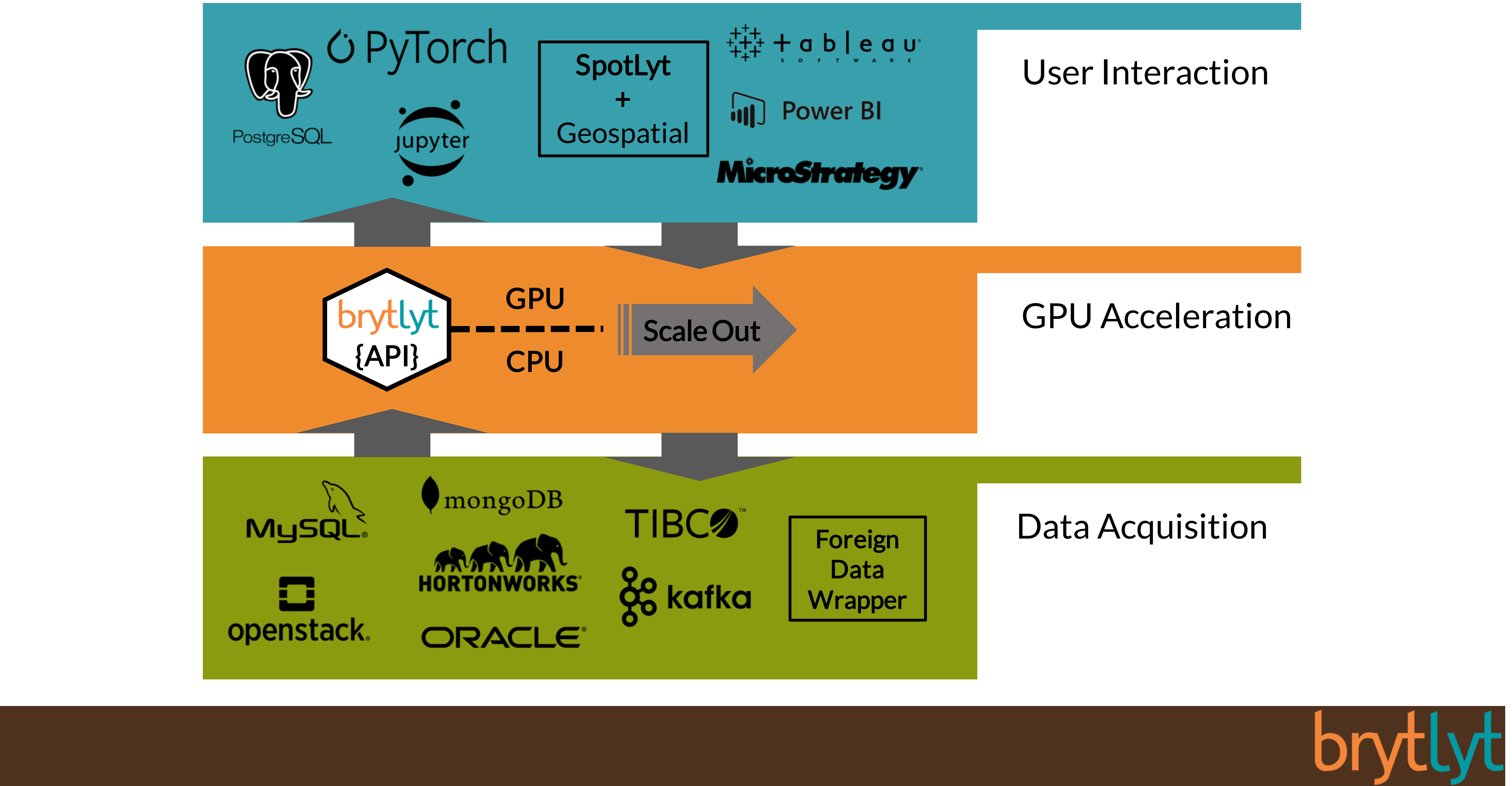
Brytlyt is a PostgreSQL fork



Brytlyt technology



brytlyt


PyTorch
PostgreSQL
jupyter
SpotLyt
+
Geospatial
tableau
SOFTWARE
Power BI
MicroStrategy
User Interaction
brytlyt
{API}
GPU
CPU
Scale Out
GPU Acceleration
MySQL
mongoDB
TIBCO
HORTONWORKS
openstack
ORACLE
kafka
Foreign
Data
Wrapper
Data Acquisition
brytlyt

TPC-H Benchmark

Why

- Measure of state of maturity of GPU database space.
- Performance comparisons of hardware and software.

What

- Examine **large volumes of data**, by executing queries with **high degree of complexity**, to give answers on **real-world business decisions.**

How

- Star schema, two large fact tables (88% of total row count) and six dimension tables
- Twenty two queries run as single user and concurrently.
- Based on typical retail use case.
- A data generator that goes up to and beyond 100TB

NVIDIA DGX-2

Why

- Step change in GPU footprint of a single server.
- Cluster of servers with network bottleneck less necessary.

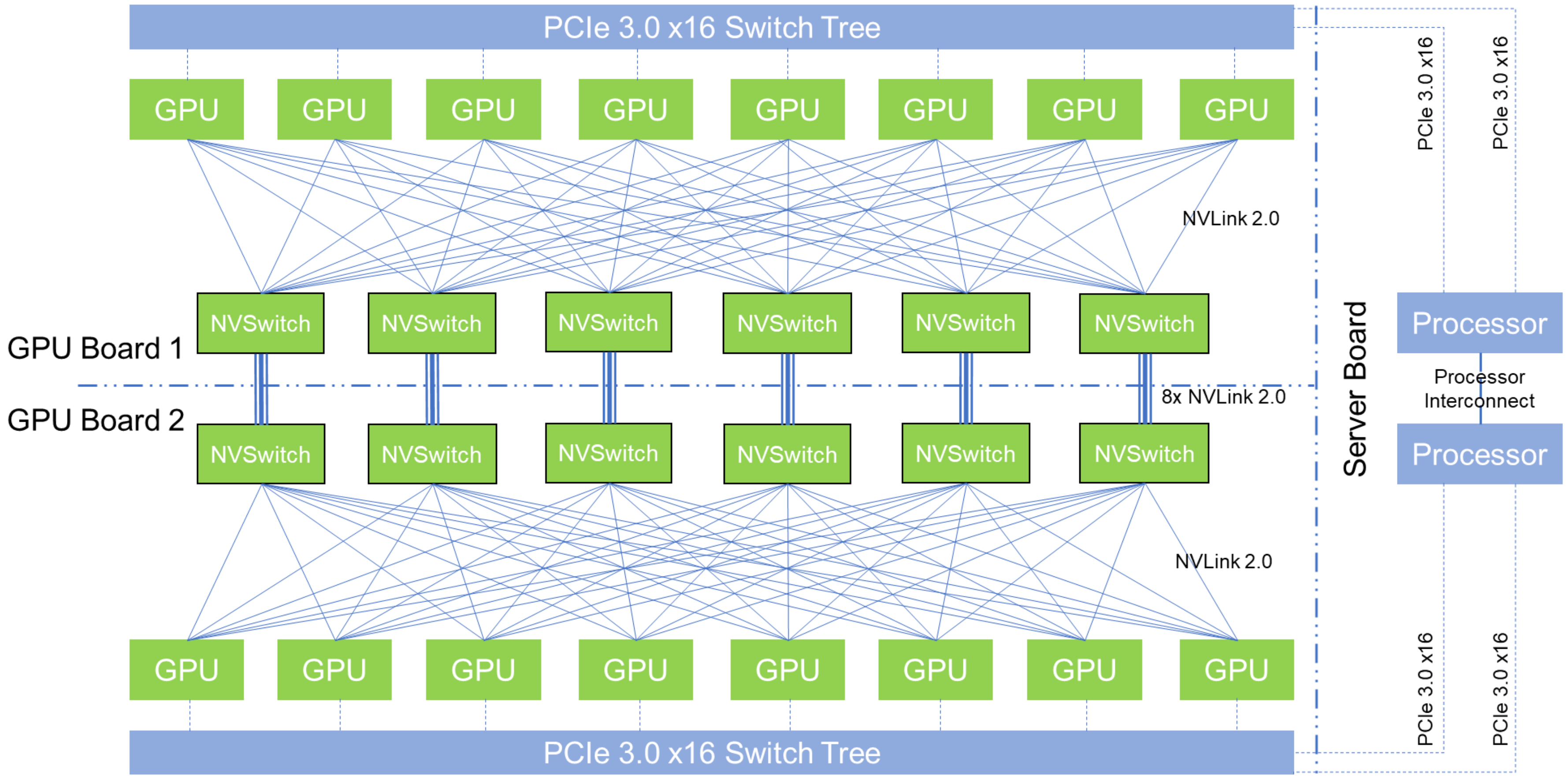
What

- Sixteen NVIDIA V100 GPUs with 32GB VRAM.
- Total of 512 GB VRAM and 2 petaFLOPs.

How

- NVSwitch provides 2.4 TB/s of GPU data transfer between GPUs.

NVIDIA DGX-2



brytlyt

TPC-H Summary

Aggregations

- Occur in all TPC-H queries and group-by performance is important.

Complex expressions

- Raw expressions in aggregations, complex expressions in joins and also string matching.

Nested queries and sub-queries

- Used to handle intermediate results in the real world.

JOINs

- All but two of the queries contain joins.

Correlated queries

- Special case of nested query where the subquery uses values from the outer query.

TPC-H – Set up and comparisons

Scale factor	1,000 GB (6 billion rows in the lineitem table)
Brytlyt	Year: 2019, DGX-2, Version 3.1 Alpha
Exasol	Year: 2014, twenty machines, TCO $719k
Microsoft	Year: 2017, one machine, TCO $472k

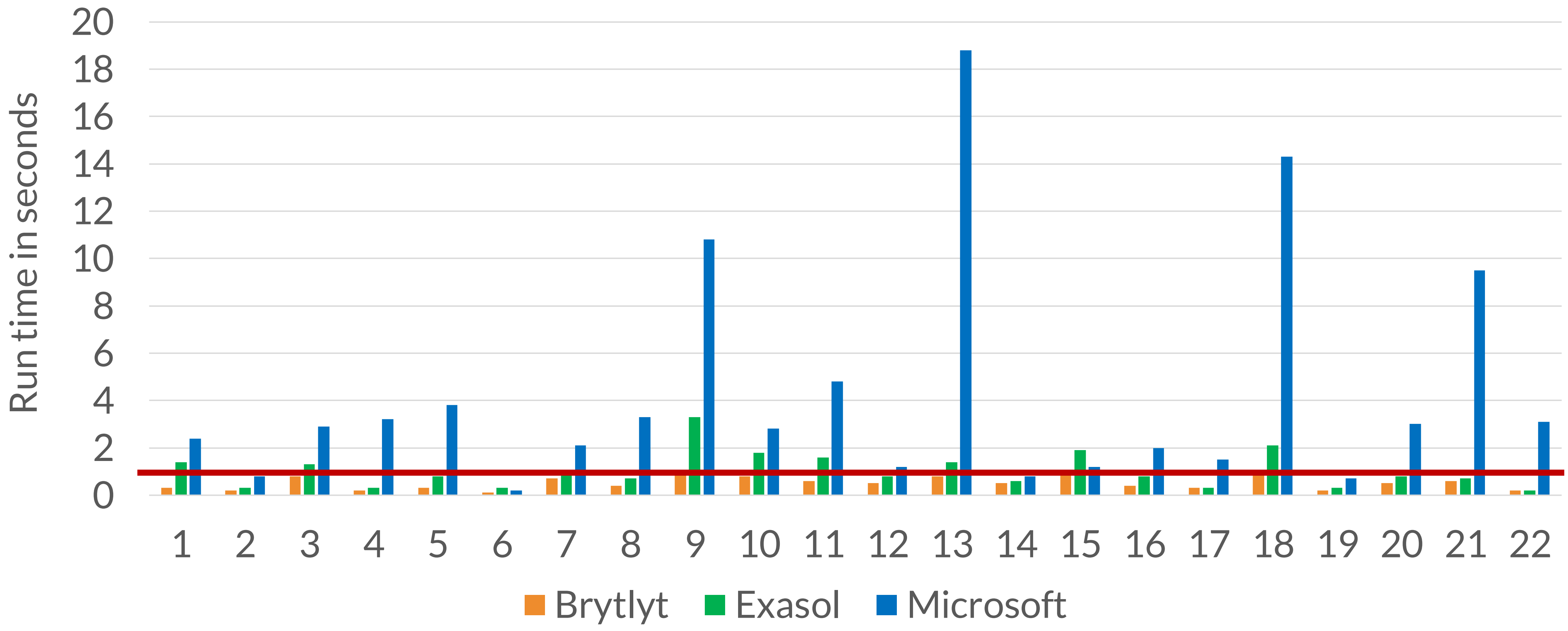
*No results of full benchmark by other GPU vendors in public domain.

Notes to benchmarking exercise

All queries run sub-second.

Redistributing lineitem table can be done sub-second (largest fact table, 70% of total data row count, 6 billion rows).

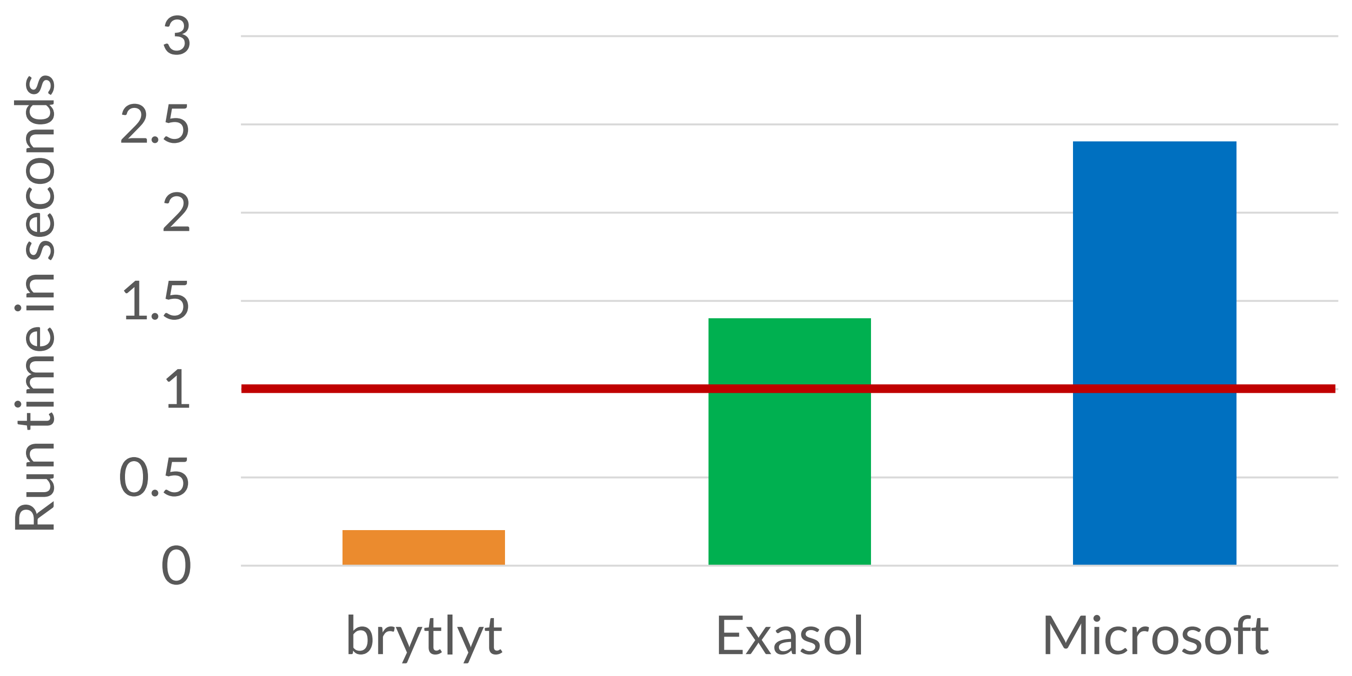
TPC-H Runtimes



Aggregations – Q1 scans 97% of lineitem table

```
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
FROM lineitem
WHERE  l_shipdate <= date '1998-12-01' - interval '90  day'
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

Runtime comparison – Q1



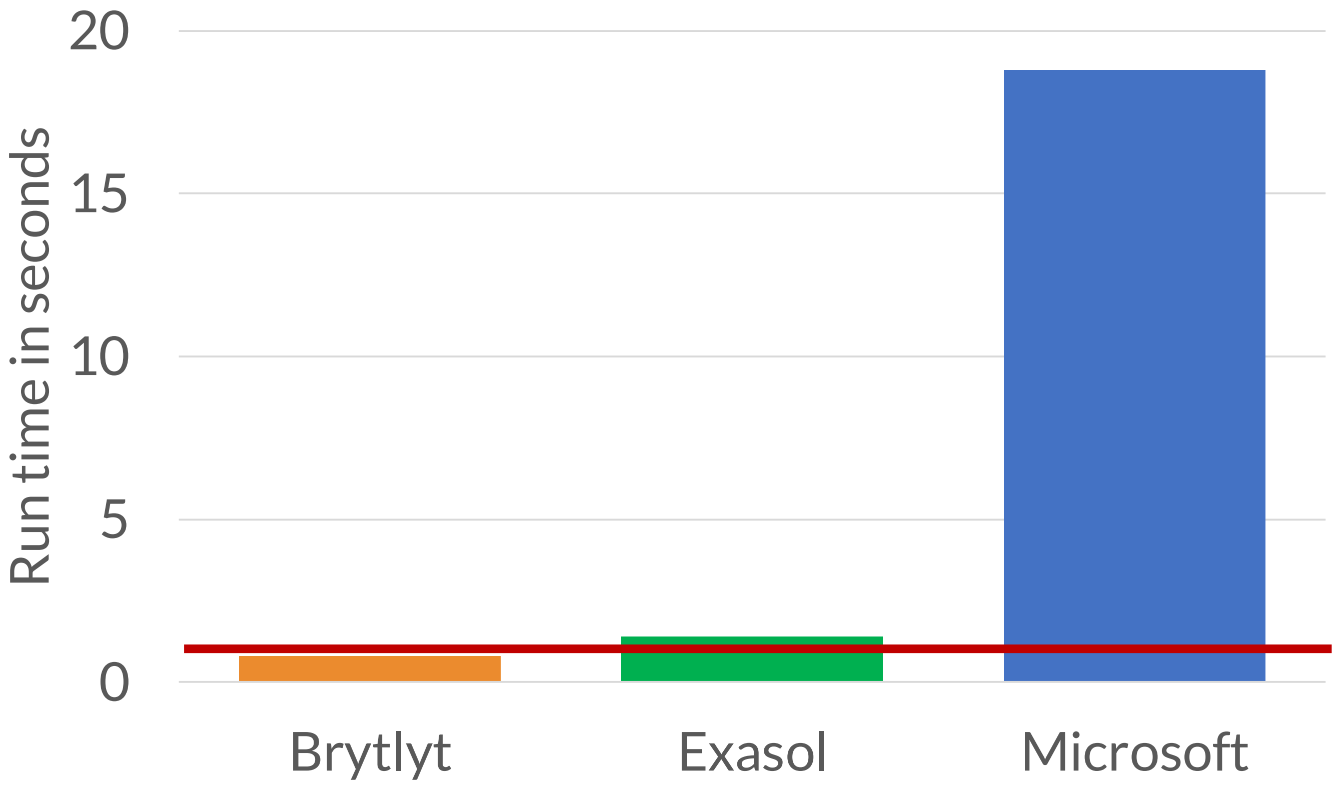
Nested queries and string expressions – Q13

```
SELECT c_count,
       count(*) AS custdist
FROM ( SELECT c_custkey,
              count(o_orderkey)
       FROM   customer
              LEFT OUTER JOIN orders ON
                     c_custkey = o_custkey
                     and o_comment NOT LIKE ‘%a%b%’
       GROUP BY c_custkey
       ) AS c_orders (c_custkey, c_count)
GROUP BY c_count
ORDER BY custdist desc, c_count desc;
```

Nested queries and string expressions – Q13

```
SELECT c_count,
       count(*) AS custdist
FROM  ( SELECT c_custkey,
               count(o_orderkey)
        FROM   customer
               LEFT OUTER JOIN orders ON
                   c_custkey = o_custkey
                   and o_comment NOT LIKE '%a%b%'
        GROUP BY c_custkey
      ) AS c_orders (c_custkey, c_count)
GROUP BY c_count
ORDER BY custdist desc, c_count desc;
```

Runtime comparison – Q13



brytlyt

JOINs – Q5 uses six tables

```
SELECT n_name,
       sum(l_extendedprice * (1 - l_discount)) as revenue
FROM   customer,
       JOIN orders ON c_custkey = o_custkey
       JOIN lineitem ON l_orderkey = o_orderkey
       JOIN supplier ON l_suppkey = s_suppkey
       JOIN nation ON s_nationkey = n_nationkey
       JOIN region ON n_regionkey = r_regionkey
WHERE   c_nationkey = s_nationkey
       r_name = '[REGION]'
       and o_orderdate >= date ‘1995-01-01'
       and o_orderdate < date '1995-01-01' + interval '1' year
GROUP BY n_name
ORDE BY revenue desc;
```

Recursive Interaction Probability (RIP)

Why

- JOINs are the most costly and useful of SQL operations.
- Better performance and flexibility than hash- and index-based methods.

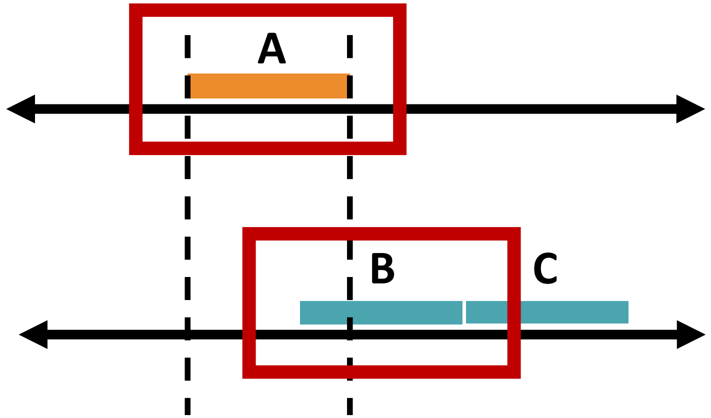
What

- Brytlyt’s patent pending intellectual property.
- Light weight pre-processing identifies tuples likely to fulfil JOIN predicate.
- Very efficient, Big O notation = O(n log n).

How

- Sorting JOIN columns.
- Recursively compare boundary elements of partitions of data.

Recursive Interaction Probability (RIP)

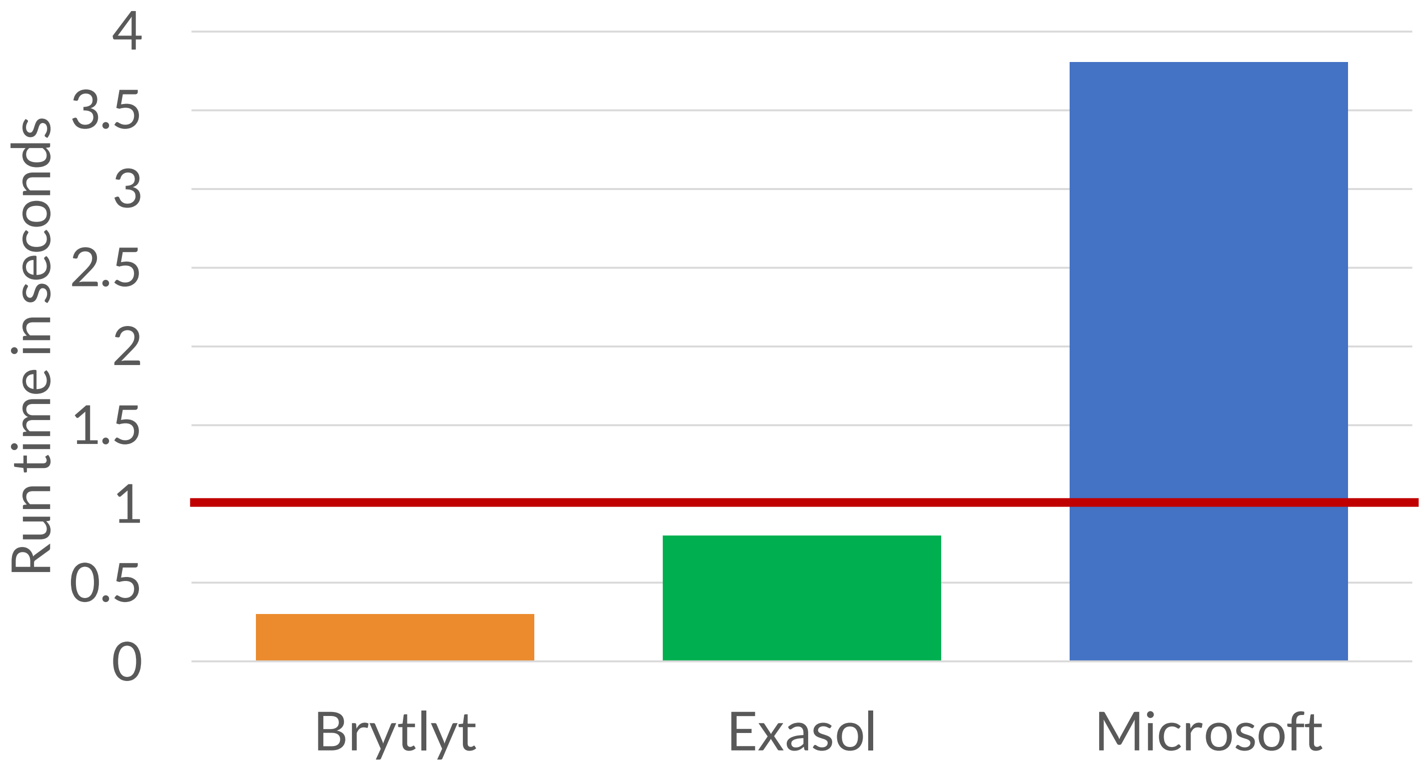


- Two number lines representing sorted JOIN columns.
- Using min and max values of sub-partition A.
- Comparing to min and max values of B and C.
- Determine there is zero probability of JOIN predicate. being fulfilled within sub-partitions A and C.

- For sub-partitions like A and B that "interact".
- Partition into smaller sub-partitions and repeat.

- Base case operation tests for JOIN.
- Incredibly efficient for "sparse" JOINs.

Runtime comparison – Q5



brytlyt

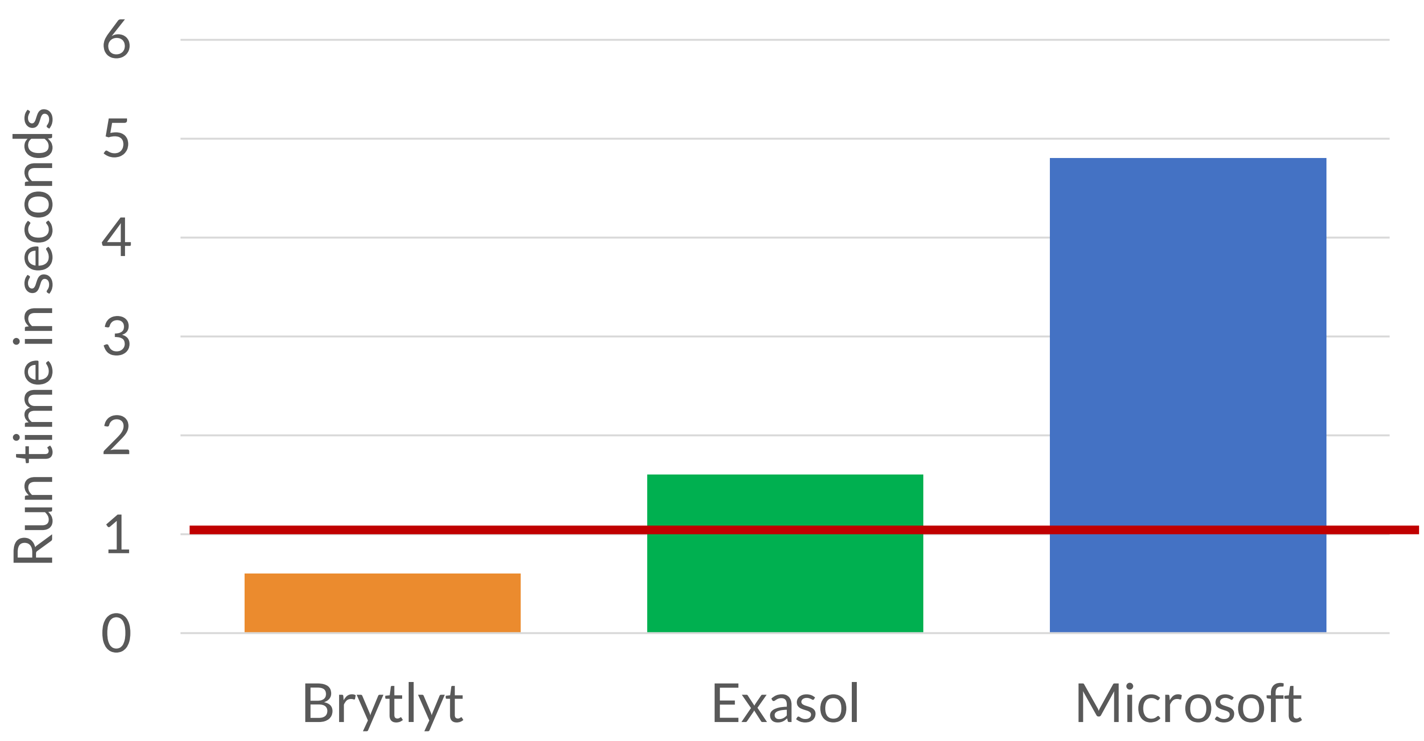
Correlated queries – Q11

```
SELECT  ps_partkey, SUM(ps_supplycost * ps_availqty) as value
FROM
    partsupp JOIN supplier ON ps_suppkey = s_suppkey
    JOIN nation ON s_nationkey = n_nationkey
WHERE
    n_name = 'ARGENTINA'
GROUP BY ps_partkey
HAVING  SUM(ps_supplycost * ps_availqty) >
      (    SELECT
               SUM(ps_supplycost * ps_availqty) * 0.015
           FROM
               partsupp JOIN supplier ON ps_suppkey = s_suppkey
               JOIN nation ON s_nationkey = n_nationkey
           WHERE
               n_name = 'PERU'
       )
ORDER BY value desc;
```

Correlated queries – Q11

```
SELECT  ps_partkey, SUM(ps_supplycost * ps_availqty) as value
FROM
     partsupp JOIN supplier ON ps_suppkey = s_suppkey
     JOIN nation ON s_nationkey = n_nationkey
WHERE
     n_name = 'ARGENTINA'
GROUP BY ps_partkey
HAVING  SUM(ps_supplycost * ps_availqty) >
        (     SELECT
                   SUM(ps_supplycost * ps_availqty) * 0.015
              FROM
                   partsupp JOIN supplier ON ps_suppkey = s_suppkey
                   JOIN nation ON s_nationkey = n_nationkey
              WHERE
                   n_name = 'PERU'
         )
ORDER BY value desc;
```

Runtime comparison – Q11



brytlyt

Brytlyt DB

GPU accelerated PostgreSQL

brytlyt

SpotLyt

Interactive analytics workbench for billion row datasets

brytlyt

BrytMind

SQL + AI + GPU

brytlyt

brytlyt

GPU Accelerated Data Processing

Speed of Thought Analytics at Scale

CEO	Richard Heyns
Email	Richard.Heyns@Brytlyt.com
URL	www.brytlyt.com
Twitter	@BrytlytDB