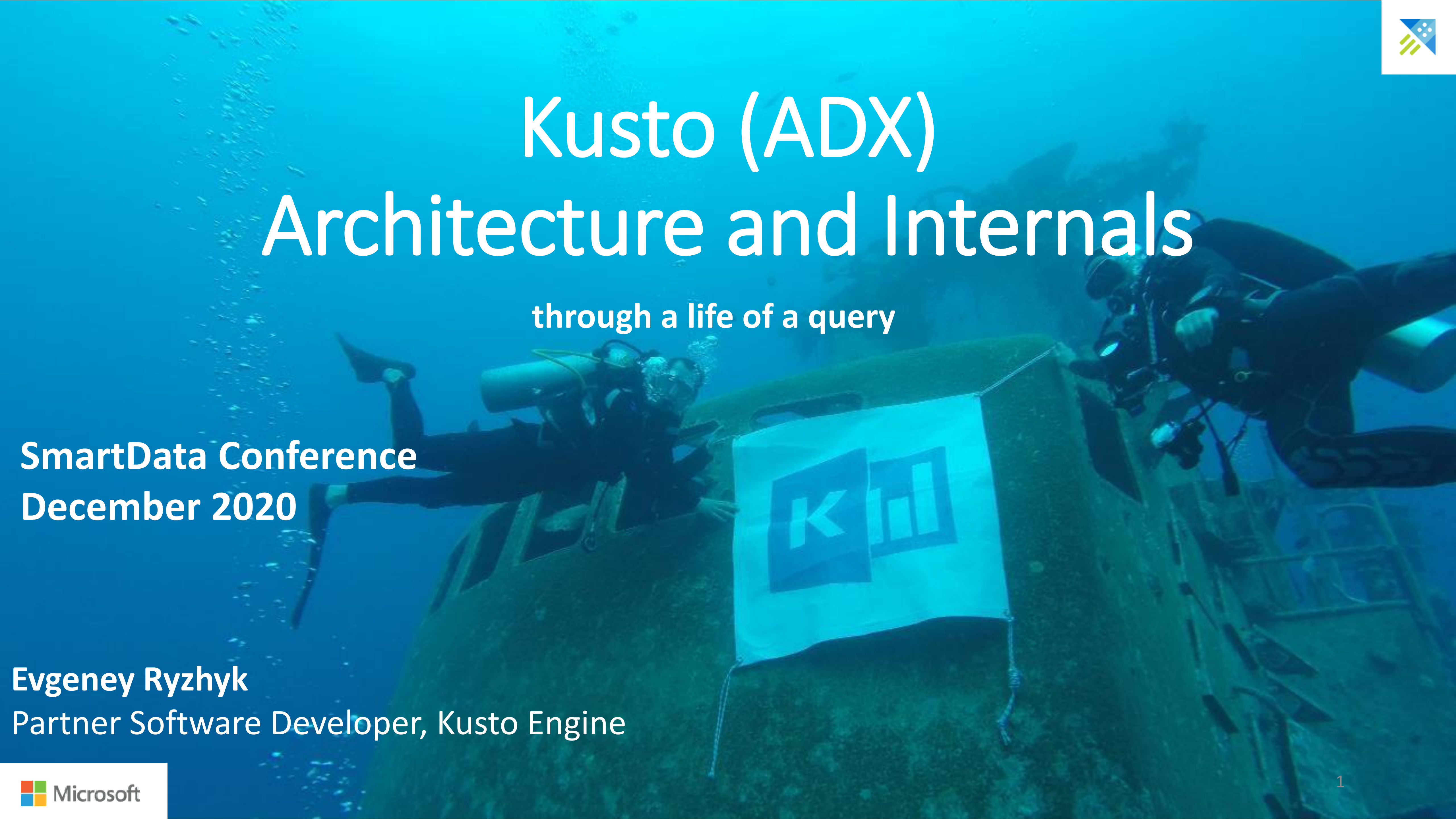


Kusto (ADX) Architecture and Internals

through a life of a query

SmartData Conference
December 2020

Evgeney Ryzhyk
Partner Software Developer, Kusto Engine

Recap: what is Kusto

- BigData Database / Platform
- Managed in Azure Cloud
- Fully distributed
- Optimized for append-only log and telemetry workloads
- Structured, semi-structured and unstructured data

Azure Data Explorer (Kusto)
Workload size / Nov 2020

Microsoft

Все
Регионы Azure

> 1M
CPU Cores

> 2 EB
Общий объем данных

+35 PB
Добавляется каждый день

30 Billion
Запросов в месяц

Kusto: один из самых масштабных сервисов Microsoft

Agenda

- Instead of the dry architecture diagrams…
- …understand Kusto by looking at the “life of a query”
- We will review:
  - Query planning and optimization
  - Kusto cluster: roles and responsibilities
  - Query distribution
  - Storage format and low-level query engine

Sample query

Count error log messages per machine during last week:

```
KustoLogs
| where Timestamp > ago(7d)
| where EventText has "error"
| summarize count() by Machine
```

1-minute demo (Live!)

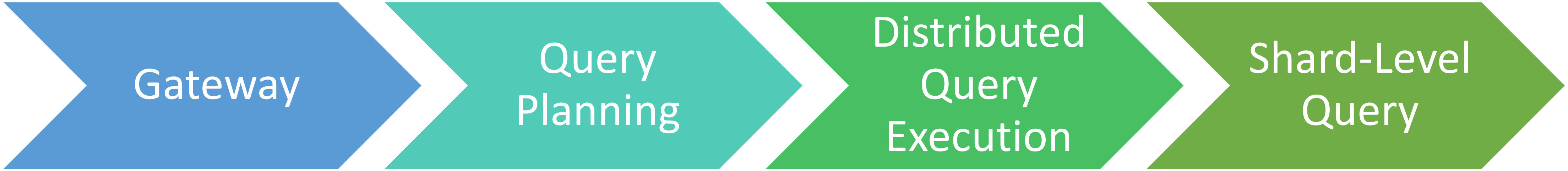
- 7 Trillion records after Timestamp filter
- Down to 14 billion by full-text term search
- Aggregated by Machine

How fast is it?

In dedicated log analytics benchmark (100TB):

- **x40** faster than BigQuery for the same cost (single user)
- **x200** faster that BigQuery for the same cost (50-concurrent users)
- Workload isn’t feasible on comparable HW with Elasticsearch

Query Lifetime



Gateway

Query Planning

Distributed Query Execution

Shard-Level Query

Step 0: Gateway

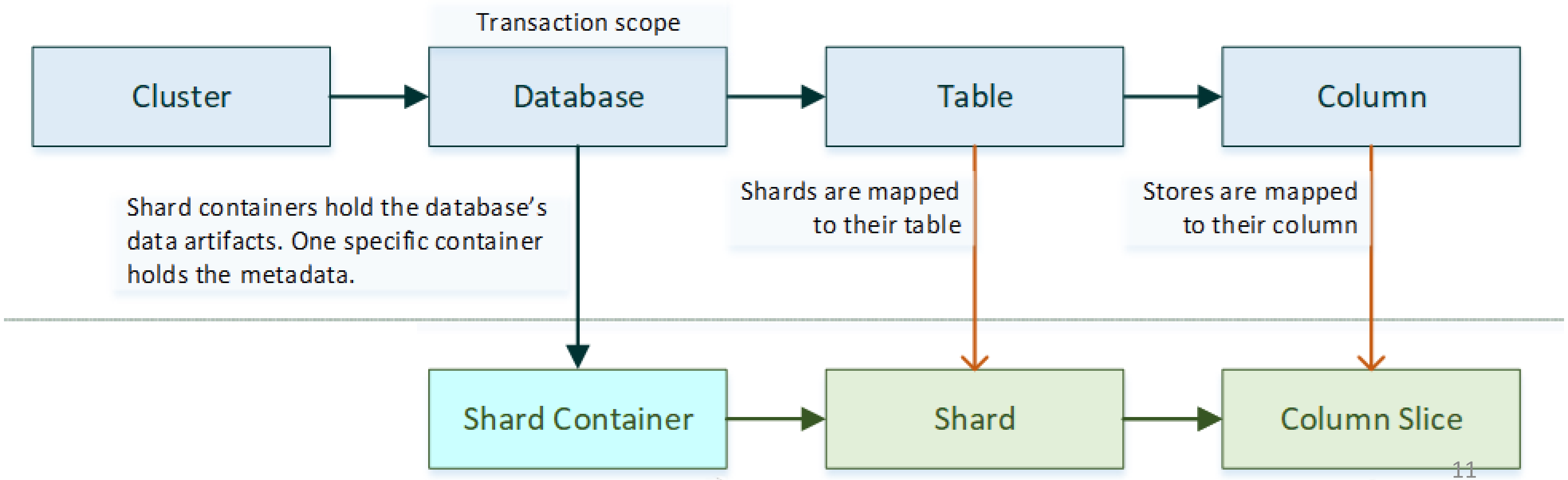
- Query arrives at the Gateway HTTP API endpoint as GET or POST request
- Authentication (AAD)
- Ask the cluster fabric to find a suitable “query head” node
- Route to the query head for planning and further execution

Step 1: Query Head/Admin

- Parse the text into AST
- Understand what we're querying: resolve database and table names (fetch metadata for the latest DB and table definitions and schema)
- Authorization (ensure current user can view the DB)
- Check semantics and data types

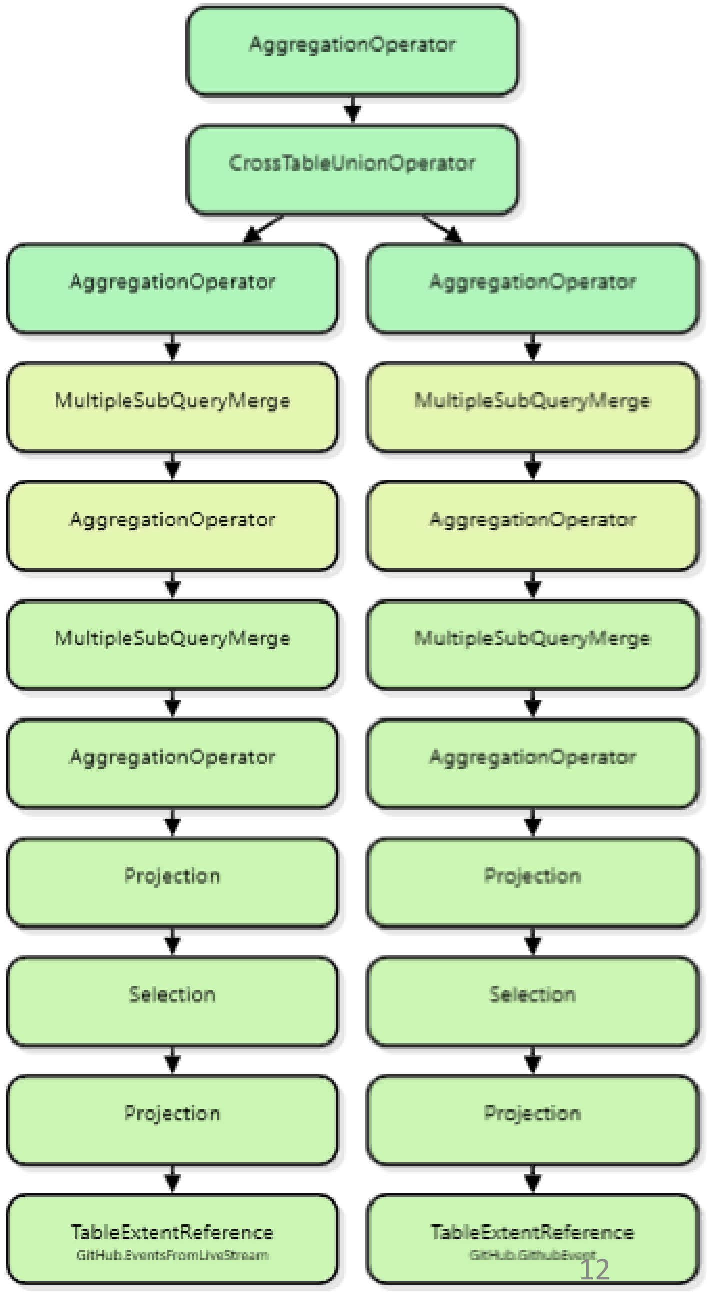
Detour: metadata

- Structure:
  - Kusto cluster: list of DB's
  - DB: list of tables
  - Table: schema + list of shards
- Updated transactionally; query sees a consistent snapshot


Transaction scope
Cluster
Database
Table
Column
Shard containers hold the database's data artifacts. One specific container holds the metadata.
Shards are mapped to their table
Stores are mapped to their column
Shard Container
Shard
Column Slice

Step 1.1: initial logical plan

- Build an **initial tree** of relational operators
- **Generic optimizations**:
  - push down predicates and projections
  - fold constant expressions
  - rewrite complex operators into simpler ones
  - …
- There’s a **query optimizer** framework: recognize and rewrite fragments of the RelOp tree
  - Global transformations are achieved by iterative application of local rewrites



Step 1.2: query distribution

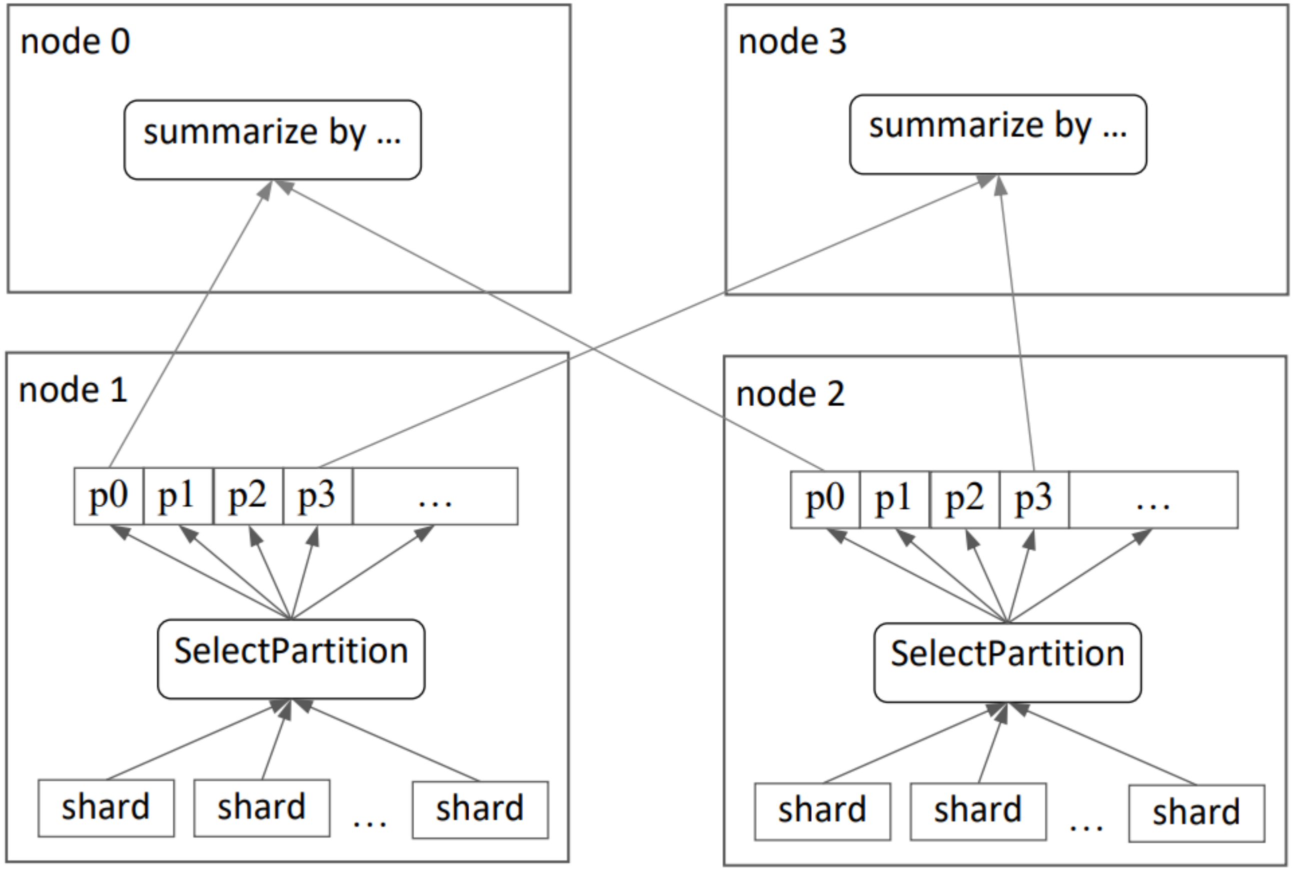
- Table may have thousands of horizontal shards
- Shards are assigned to cluster nodes (using consistent hash)
- Machines cache the frequently accessed data fragments, on SSD and in memory
- Need to turn an abstract, logical query plan into a parallel and distributed one, such that:
  - “Heavy lifting” is done on each node, close to the cached data
  - All cluster resources are utilized
  - Non-trivial strategies for large joins and aggregations are possible

Step 1.2: query distribution

- Represent the table as a **hierarchy of shards**
- Make important **strategy decisions** (shuffled vs broadcast join, shuffled vs non-shuffled aggregation, etc.)
- Semantic-preserving transformations of the query plan:
  - E.g. logical `count() by Key` becomes
  - distributed `count() by key` → `merge` → `sum(counts) by Key`
- Hundreds of other transformations, rule-based and cost-based

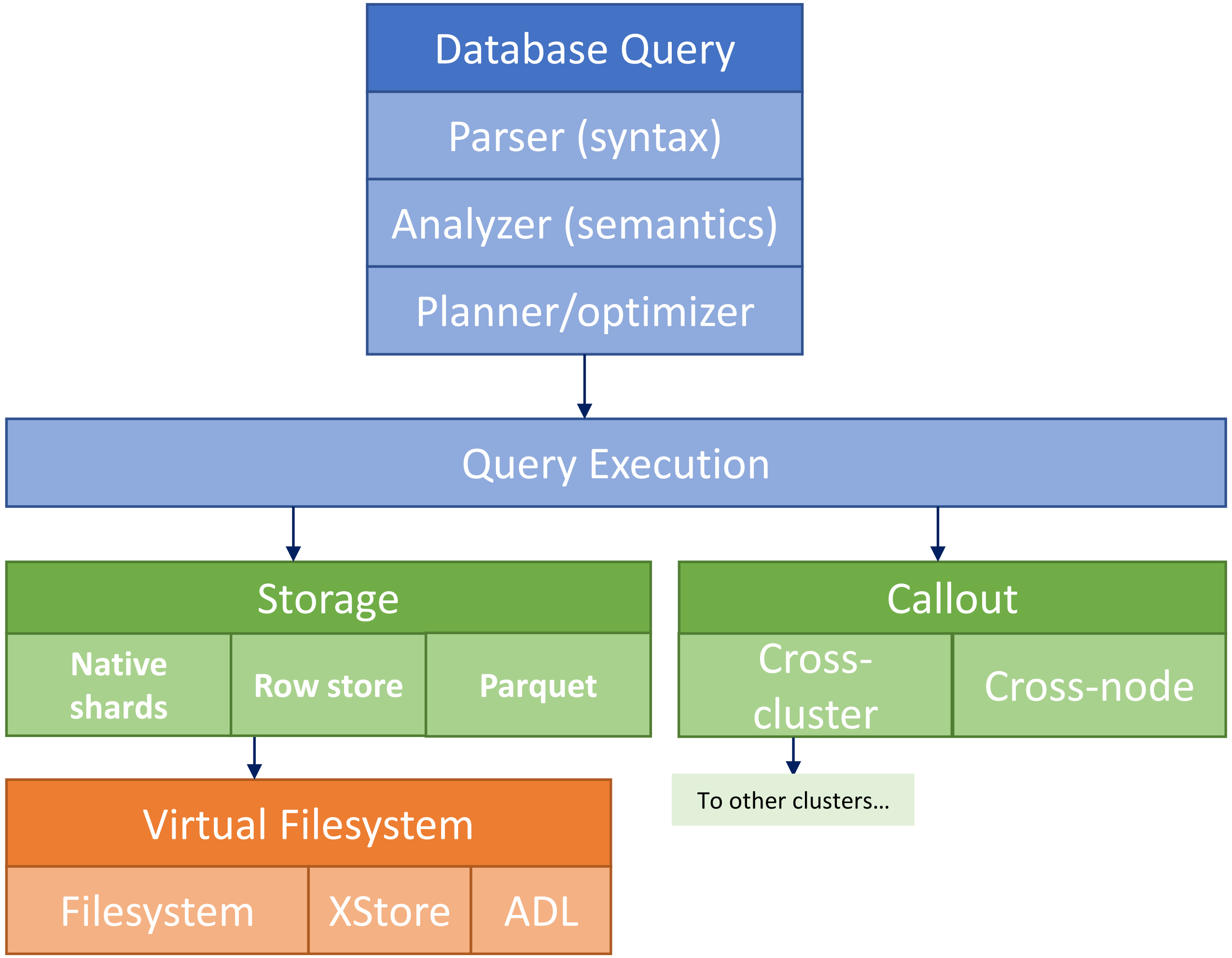
Detour: shuffled and broadcast join

- Query distribution is not just tree-merge style
- Shuffled joins and aggregations create a DAG of producers and consumers:



Extendable storage model

- Shards (column-store) are created from regular batch ingestion
- We also have RowStore, for trickling NRT ingestion
- And external tables (e.g. Parquet files)
- And other clusters (cross-cluster queries)
- There are handled by the same query distribution/federation mechanisms:
  - Represent logical table as union of shards, row stores, external tables
  - Push down and distribute the operators to other "engines“



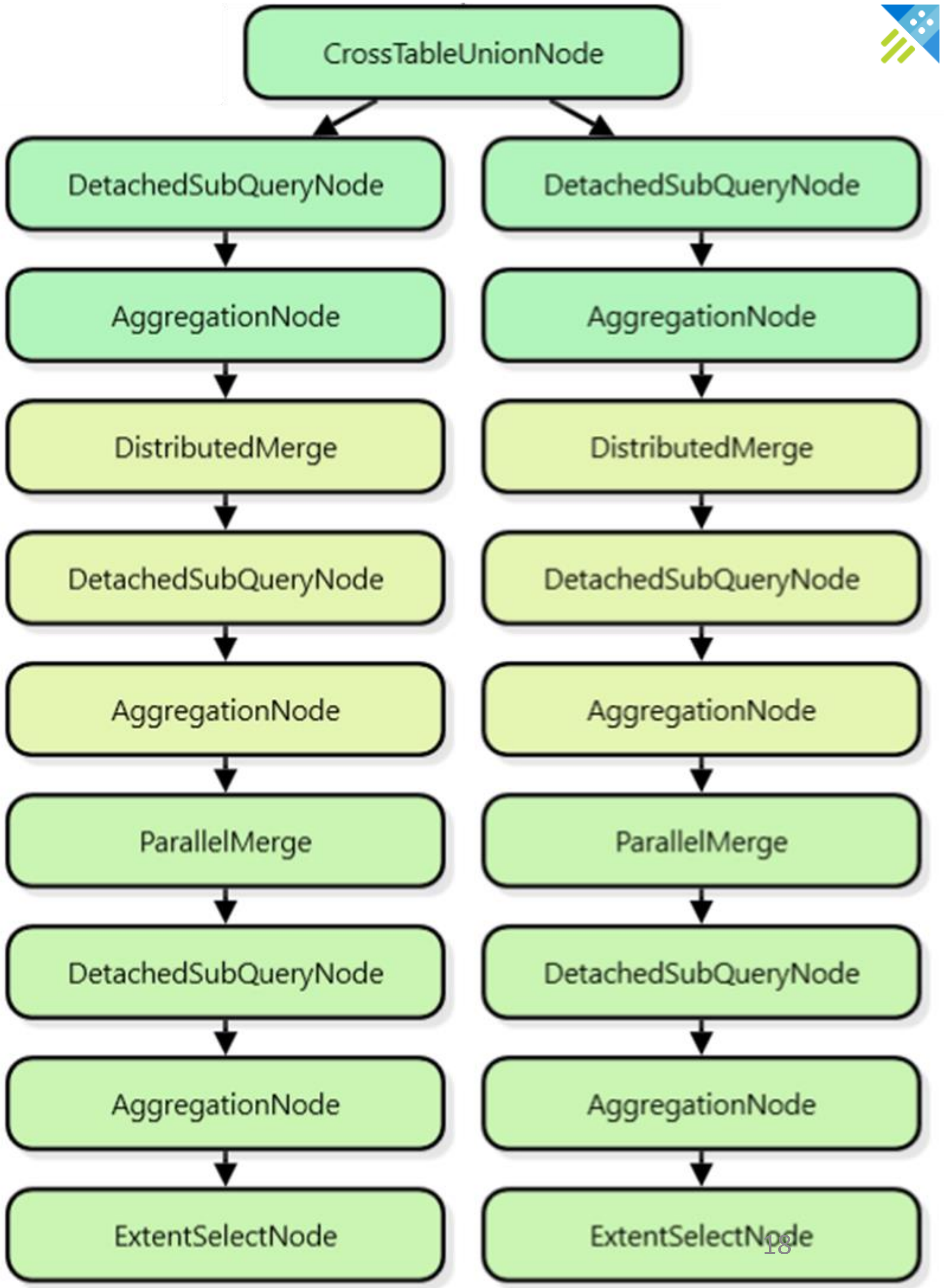
Step 1.3: shard pruning

- Query head maintains a **cache** of high-level **shard stats** (min/max values, Bloom filters, etc.)
- Apply the coarse-grained query filters to prune irrelevant shards early
  - Reduction in query plan size and inter-node traffic
- In our example: eliminate shards that fall out of the last 7 days range based on the per-shard min/max stats of the Timestamp field.

Step 1.4: final query plan

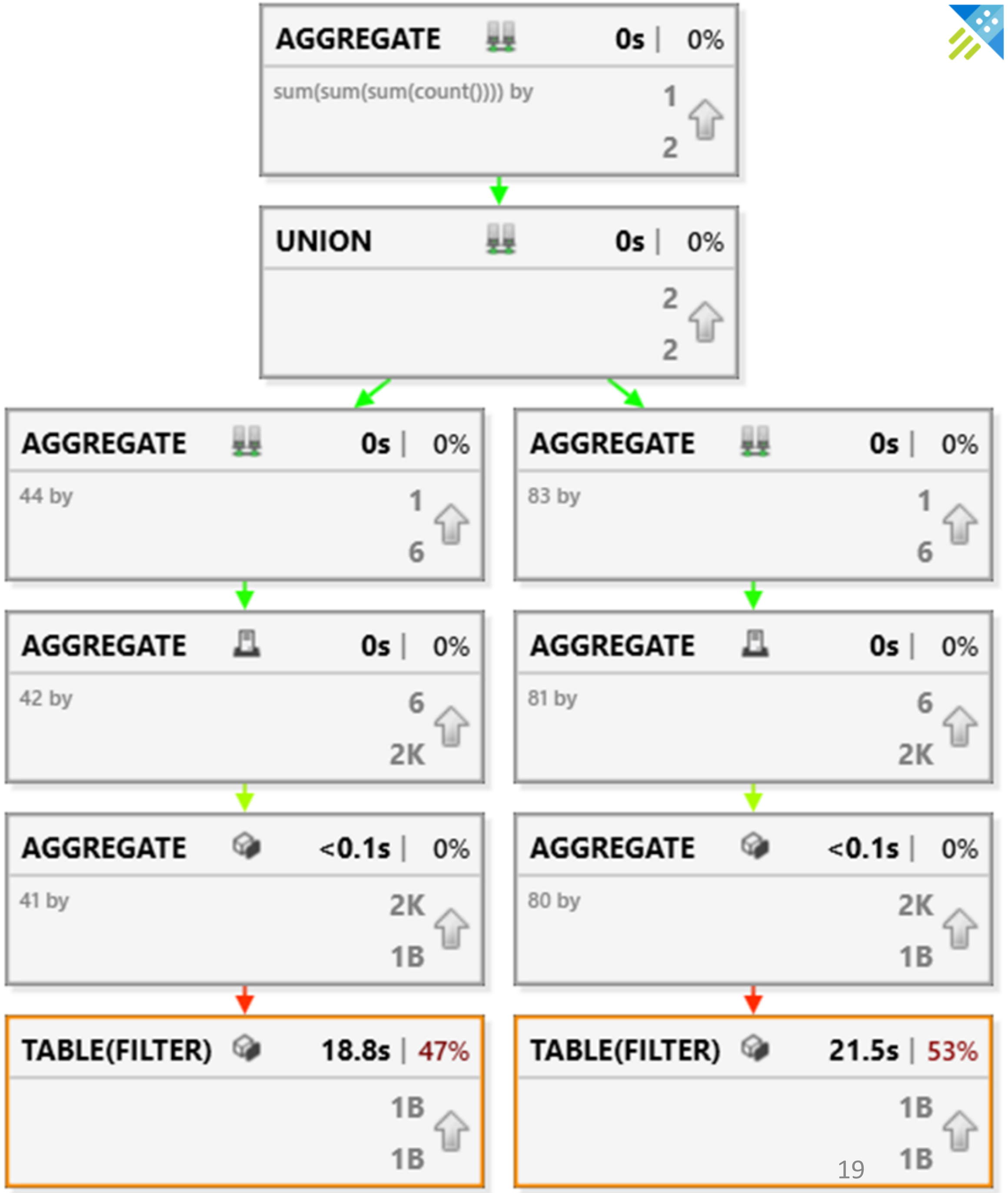
- A large tree of operator nodes

- 3 types of operators:
  - Functional (actual query logic)
  - Parallelization (run sub-trees in parallel)
  - Remoting (run sub-tree on another node and stream results back)



Step 2: query execution

- Distributed query plan is passed to the “root” engine.
- Each node in the query plan represents a pull-mode, block-level iterator
- Leaf nodes are special: they know and exploit the intricacies of the storage format and the indexes
- Control flow: top-down
- Data flow: bottom-up



Step 2.1

- Parallelization and remoting operators quickly bring us to shards on the leaf nodes
- Leaf shard queries run in parallel on all nodes and cores
- Shard queries are executed by the **storage engine**

Step 3: storage (shard) query

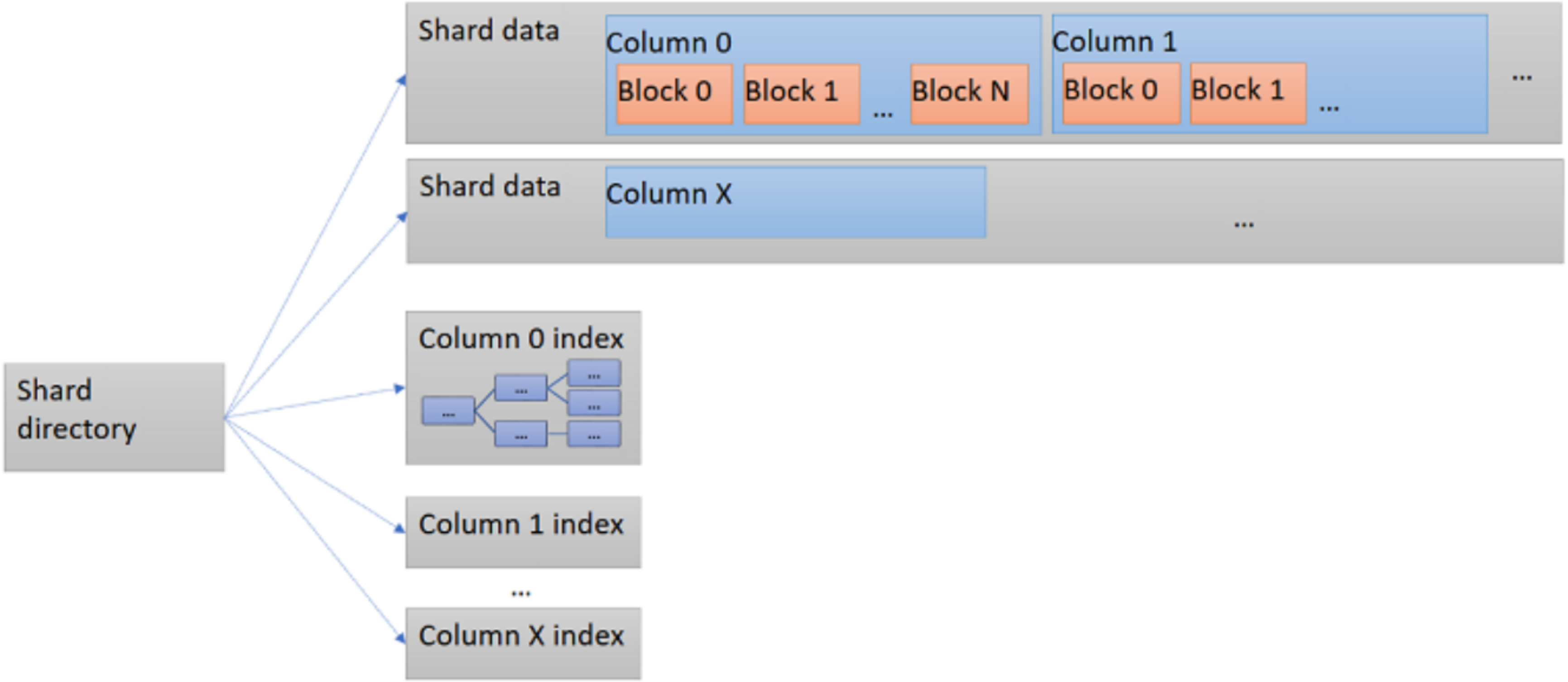
- As part of the query planning, identify the bottom part of the query that runs “close enough” to the shard:

```
Shard
| where Timestamp > ago(7d)
| where Message has 'error’
| summarize count() by Machine
```

- Generate (LLVM) a fused, efficient machine code that:
  - Calls out to the storage runtime to probe indexes and read slices of the shard columns’ data
  - Performs the vectorized and tuple-level filtering/calculations/aggregations
  - Takes advantage of the specific shard encoding (dictionaries, non-nullability)

Detour: shard format

- Compressed column store with free text support and full-text inverted index



Generated shard query

Assume that “Machine” field is low-cardinality and dictionary-encoded in this shard: query can benefit from it.

```
time_index_res = probe_time_index(field=“Timestamp”, range=-7d..now);
txt_index_res = probe_text_index(field=“Message”, term=“error”);
positions = intersect(time_index_res, txt_index_res);
for each pos in positions:
   key = fetch_field_key_at(“Machine”, pos);
   grid[key] += 1; // count() by Machine
for each key in grid: // decode dict keys as text values
   decode_value(“Machine”, key)
propagate aggregated grid to the parent operator
```

Beyond query

- Update policy (built-in lightweight ETL)
- Materialized Views
- Bulk and Streaming ingestion
- Continuous Data Export
- External tables and Data Lake integration
- Follower databases (aka “virtual data-warehouse”)
- Time-series analytics
- Geo-spatial queries
- ML: embed python in query

Recap / Summary

- This was not a full design of Kusto (see whitepaper), but we touched upon key elements.
- Kusto architecture brings together classical and well-known DB design patterns, along with recent research and domain-specific innovation.
    - Relational model at the core
    - Shared-nothing distributed architecture
    - Tiered storage model (RAM/SSD/Blobs)
    - Purpose-built compressed column store
    - Automatic full-text index
    - Special support for JSON-like data type

Azure Data Explorer (Kusto)
Product links

Product

- Product Page: http://aka.ms/AzureDataExplorer
- Docs: https://aka.ms/adx.docs
- Pricing Page: https://azure.microsoft.com/en-us/pricing/details/data-explorer/
- Cost Estimator: http://aka.ms/adx.cost

Demos

- Scott Guthrie's Keynote (ADX Announcement/Demo)
- Rohan Kumar's Keynote (ADX Announcement/Demo)
- Scott Guthrie's in Techorama (Scott's demo)

Blogs

- Announcement: https://azure.microsoft.com/en-in/blog/introducing-azure-data-explorer/
- Whitepaper: https://azure.microsoft.com/en-us/resources/azure-data-explorer/en-us/
- 101 blog: https://azure.microsoft.com/en-us/blog/azure-data-explorer-technology-101/

Social and Community

- Twitter: @AzDataExplorer
- Stack overflow: https://stackoverflow.com/search?q=Azure+Data+Explorer
- Tech Community: https://techcommunity.microsoft.com/t5/Azure-Data-Explorer/bd-p/Kusto