PRESENTED BY SACHIN TRIPATHI

ADOPTING CLICKHOUSE @ YOUR WORK

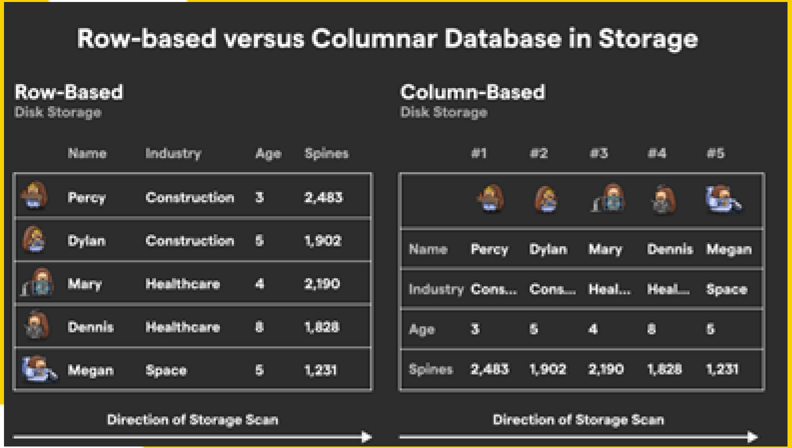
MASTER THE FUNDAMENTALS AND BEYOND

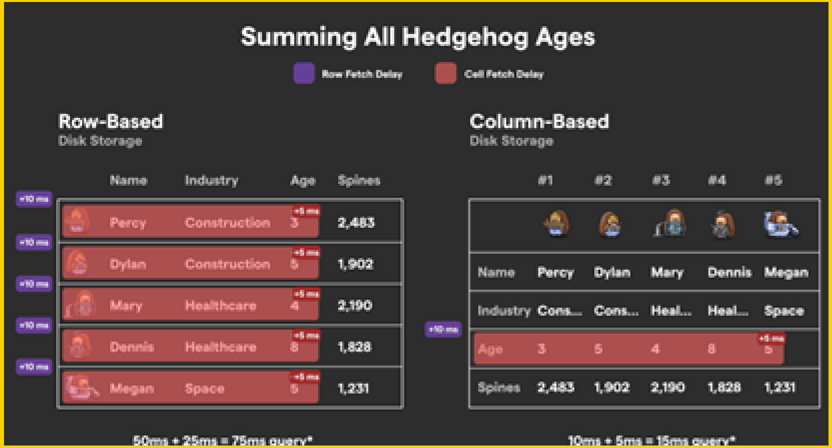
INTRODUCTION

Senior Data Engineer.
I have worked in constructing real-time data pipelines, implementing MLOPS architectures, and ensuring data security at scale.

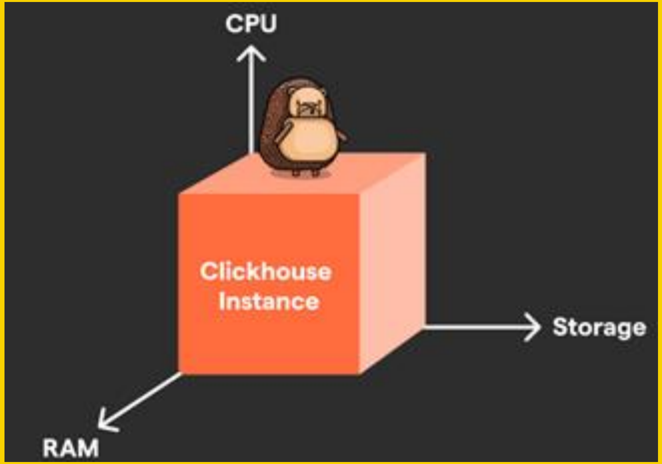
Currently at Earnin, I focus on empowering teams to turn complex data into actionable insights by leveraging modern lakehouse architectures and distributed computing frameworks.

OLAP VS OLTP

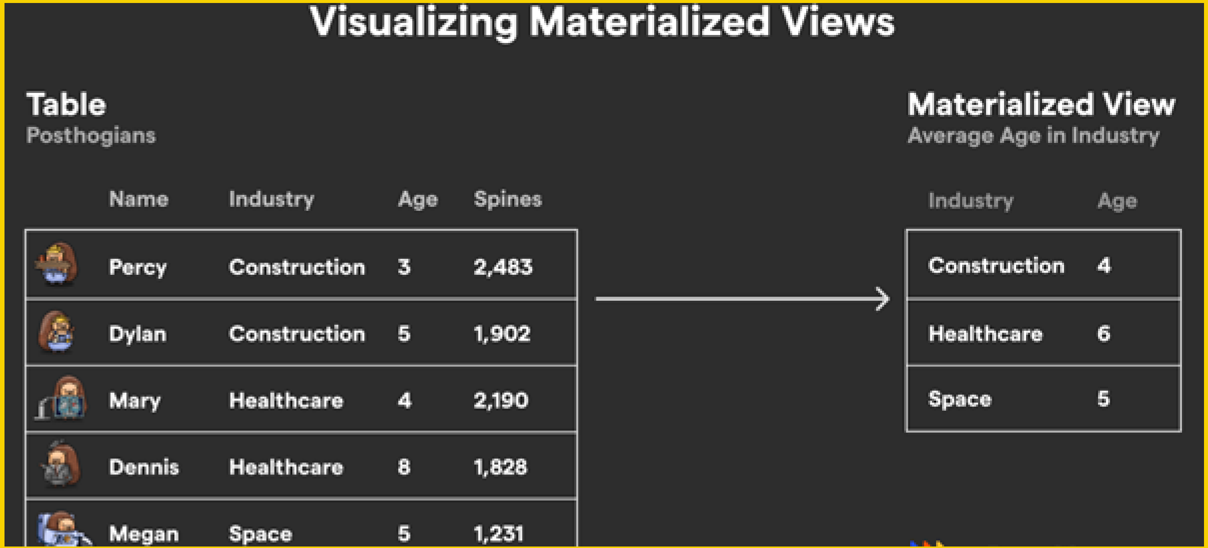

Row-based versus Columnar Database in Storage
Row-Based
Disk Storage
Name Industry Age Spines
Percy Construction 3 2,483
Dylan Construction 5 1,902
Mary Healthcare 4 2,190
Dennis Healthcare 8 1,828
Megan Space 5 1,231
Direction of Storage Scan
Column-Based
Disk Storage
#1 #2 #3 #4 #5
Name Percy Dylan Mary Dennis Megan
Industry Cons... Cons... Heal... Heal... Space
Age 3 5 4 8 5
Spines 2,483 1,902 2,190 1,828 1,231
Direction of Storage Scan


Summing All Hedgehog Ages
Row Fetch Delay
Cell Fetch Delay
Row-Based
Disk Storage
Name Industry Age Spines
Percy Construction 3 2,483
Dylan Construction 5 1,902
Mary Healthcare 4 2,190
Dennis Healthcare 8 1,828
Megan Space 5 1,231
50ms + 25ms = 75ms query*
Column-Based
Disk Storage
#1 #2 #3 #4 #5
Name Percy Dylan Mary Dennis Megan
Industry Cons... Cons... Heal... Heal... Space
Age 3 5 4 8 5
Spines 2,483 1,902 2,190 1,828 1,231
10ms + 5ms = 15ms query*

CH VS POSTGRES



CH VS POSTGRES



Visualizing Materialized Views

Table
Posthogians

Name	Industry	Age	Spines
Percy	Construction	3	2,483
Dylan	Construction	5	1,902
Mary	Healthcare	4	2,190
Dennis	Healthcare	8	1,828
Megan	Space	5	1,231

Materialized View
Average Age in Industry

Industry	Age
Construction	4
Healthcare	6
Space	5

MERGETREE

Parts:

- Data is stored in immutable directories
- sorted by the ORDER BY clause.
- Each part contains compressed column files and a sparse primary index.

Granules: Rows are grouped into granules (size set by index_granularity, default 8192). The sparse index stores marks pointing to the start of each granule.

Merging: Small parts are merged into larger ones (like merge sort) to optimize storage and query efficiency.

MERGETREE

SQL

```sql
SELECT
    toStartOfDay(timestamp),
    event,
    sum(metric_value) as total_metric_value
FROM sensor_values
WHERE site_id = 233 AND timestamp > '2010-01-01' and timestamp < '2023-0
GROUP BY toStartOfDay(timestamp), event
ORDER BY total_metric_value DESC
LIMIT 20
```

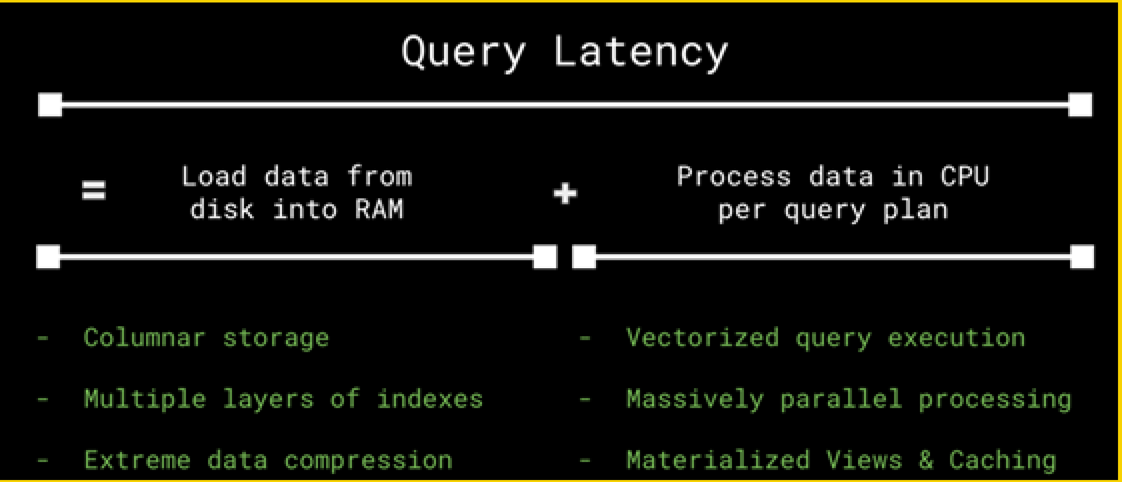
```
ReadFromMergeTree
Header: and(greater(timestamp, '2010-01-01'), less(timestamp, '2023-01-0
        timestamp DateTime
        site_id UInt32
        event String
        metric_value Int32
Indexes:
    PrimaryKey
    Keys:
        site_id
        toStartOfDay(timestamp)
    Condition: and(and((toStartOfDay(timestamp) in (-Inf, 1672531200]),
    Parts: 2/2
    Granules: 11/24415
```

Data is expensive to update

- Scan all the data to find what parts contain the relevant data. This isn't often covered by ORDER BY and thus quite expensive.
- Rewrite the whole part (including any columns)

- Ways to handle this :
  - Writing duplicated rows for new data, using other table engines (e.g. ReplacingMergeTree) and accounting for this duplication in our queries.

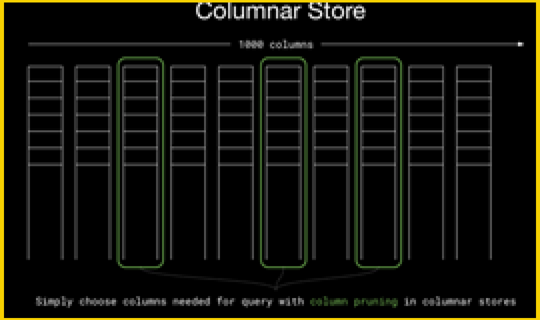
Why CH is so Fast?

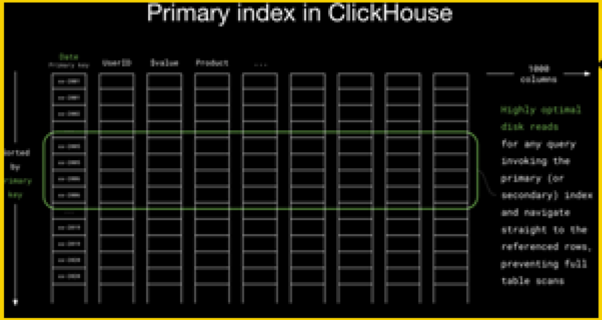


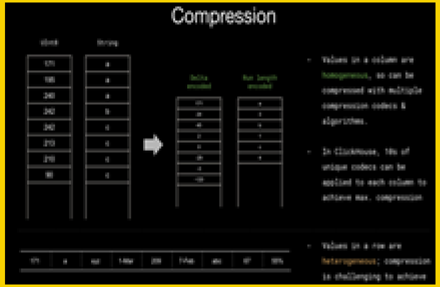

```
SELECT Column_a, SUM (Column_b) AS
Column_b_sum, MEDIAN (Column_3) AS
Column_c_median
FROM Table_1
WHERE  Column_x >= XX AND Column_x
<= YY
GROUP BY (Column_a)
ORDER BY (Column_b)
LIMIT 1000
```

- Selected Columns
- aggregations
- WHERE to select requisite rows
- groupings

Load data from disk into RAM

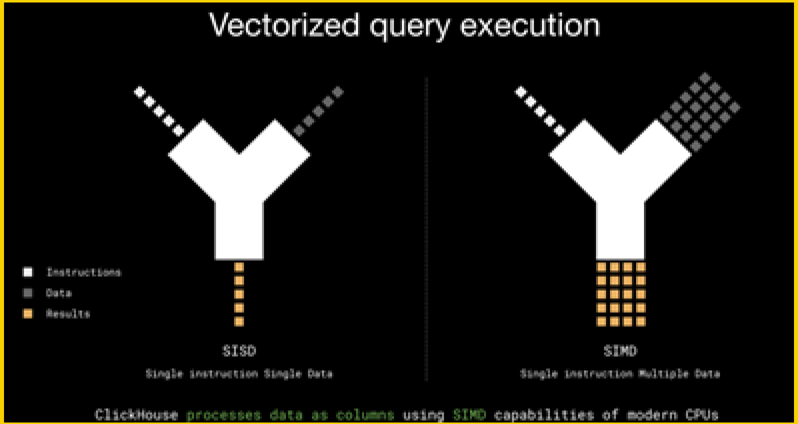


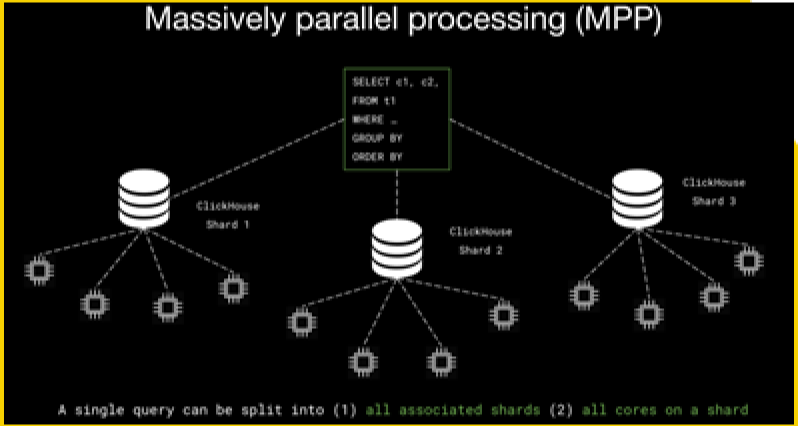




- ClickHouse uses columnar storage to identify precisely the query-relevant columns,
- multiple layers of indexes to identify precisely the query-relevant rows,
- advanced compression techniques on columnar storage format to minimize size of the data read from disk.

Process Data in CPU





- Vectorized query execution reduces CPU cache miss rates, utilizes the SIMD (single instruction multiple data) capabilities of modern CPUs.
- Not only can a single large query be split into the various CPU cores of a single node, but a single query can also leverage all the other CPU cores and disks corresponding to all the other shards in the cluster.
- Materialised views

USE CASE 1: WORKING WITH JSON IN CLICKHOUSE

HIGHLIGHTS

- **True Columnar Storage:** JSON paths are stored in a columnar format, enabling optimal compression and vectorized operations.
- **Dynamic Data Types:** Supports mixed and dynamically changing data types for the same JSON path .
- **Dense Storage:** Avoids redundant storage of NULL/default values, ensuring scalability for PB-scale datasets.

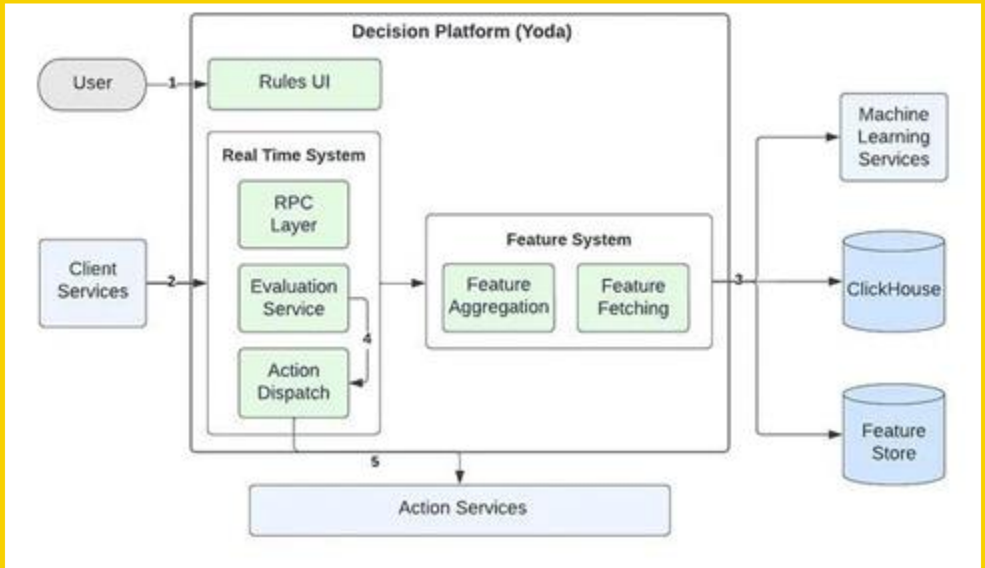
QUERY PATTERNS

- **Extracting All Paths and Types:** Extract metadata about JSON paths and their types efficiently
- **Structured Querying and filtering nested fields:** Query specific JSON paths with ease and filter rows based on specific JSON values
- **JSON Type Declaration:**

```
<column_name> JSON(
   max_dynamic_paths=N,
   max_dynamic_types=M,
   some.path TypeName,
   SKIP path.to.skip,
   SKIP REGEXP 'paths_regexp'
)
```

USE CASE 2: REAL-TIME RULE ENGINES

FLOW



WORKFLOW

- **Data Ingestion:** Real-time data from PostgreSQL and event streams is ingested using Kafka and Flink.
- **Feature Engineering:** Analysts define dynamic SQL-based features using YAML configurations.
- **Rule Evaluation:** Features are evaluated against rules, triggering actions such as warnings, suspensions, or account blocks.
- If pos_amt exceeds a threshold, flag the transaction.
- Actions include warnings, suspensions, or account blocks

Example YAML Configuration:

```
Features:
  - Name: pos_amt
    Type: FLOAT
Query: |
    SELECT
      order_id,
      SUM(amount) AS pos_amt
  FROM transactions
  WHERE created_at >= NOW() - INTERVAL 1 HOUR;
```

COMPARING THE ALTERNATIVES

CLICKHOUSE VS LSM-BASED SYSTEMS

Aspect	ClickHouse (MergeTree)	LSM-Based Trees(Cassendra)
Write Path	Direct to disk; sorting by PIDX (batching). Lighter on memory	WAL → in-memory sorting → disk. Memory Intensive
Ingestion Pattern	Prefers larger batches (e.g., 1k rows).Think of this as delayed data freshness	Prefers smaller, frequent writes.
Indexing	Sparse primary_idx and column.mrk.	Bloom filters and indexes for SSTables.
Point Queries	Scans excess data; less optimal.	Minimal data scans; highly efficient.
Analytical Queries	Best for aggregations and range scans.	Less optimized compared to columnar storage.
Storage Format	True columnar; efficient compression/vectorization.	Row-oriented; less compression-friendly.

CLICKHOUSE VS APACHE PINOT

Aspect	Apache Pinot	ClickHouse
Data Format	Row-oriented for real-time, columnar for offline.	True columnar format always.
Ingestion Latency	Near real-time with consuming segments.Memory Intensive	Batch ingestion; less optimal for frequent small updates.
Use Case Examples	Real-time dashboards, ad-tech, streaming event analytics.	Batch analytics, log aggregation,

READY TO USE?

PRESENTED BY SACHIN TRIPATHI