Clickhouse

Darwinium

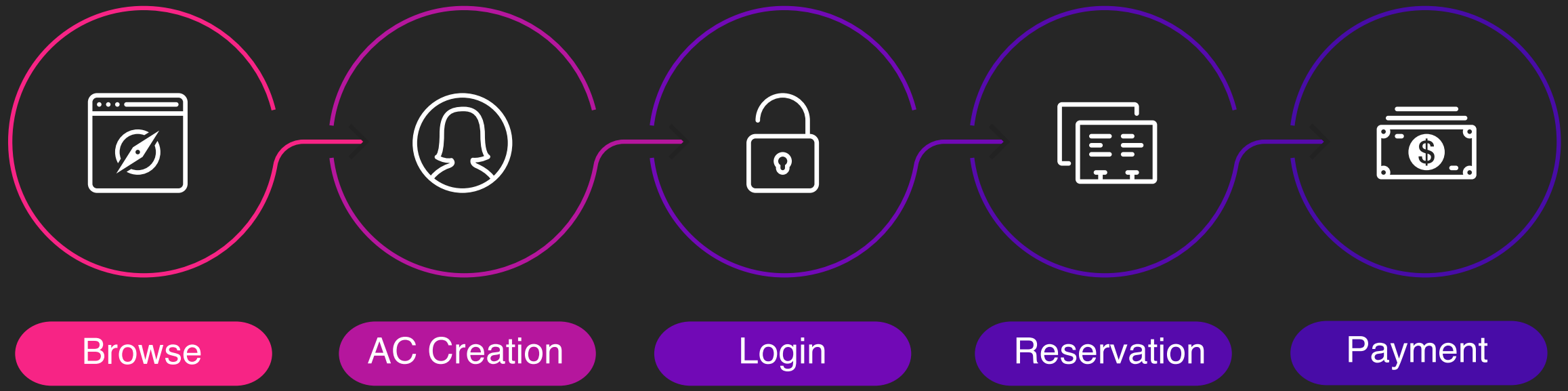
Building an Analytical System of Record to Unify Digital Security with Fraud Prevention

Ananth Gundabattula
Co-Founder Darwinium

Darwinium

Introduction

Darwinium - Full Visibility, Context and Control Across the Customer Journey



Darwinium Continuous Customer Protection – at the edge

Darwinium

Darwinium - Core Philosophy & How Clickhouse played a crucial role

Time to Value

Rapid Deployment
Insights from Day One
Leverage Existing Infrastructure

Agility

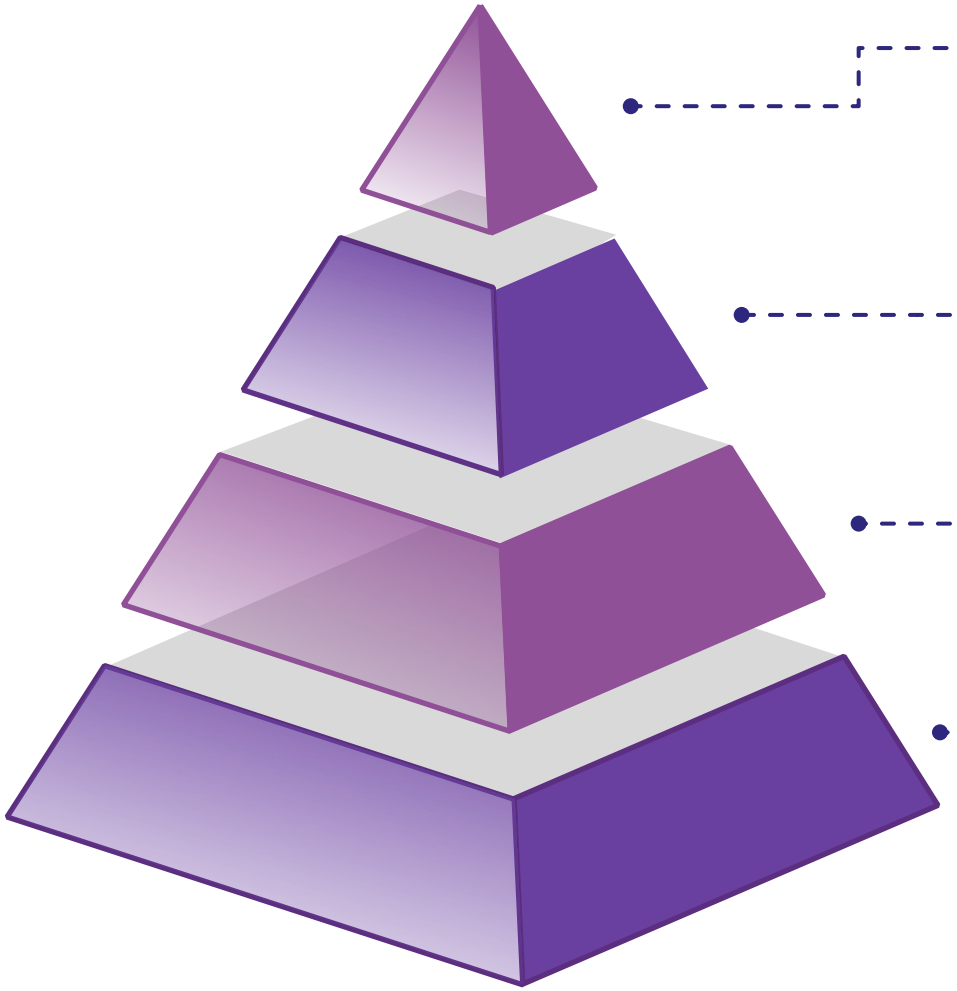
Ultimate Configuration
Cross Platform and Device
Minimal Engineering

Cost Efficiency

Edge Compute Efficiencies
Reduced Operational Cost
Cross-Organizational Relevance

Platform Summary: Hierarchy of Risk Evolution

Darwinium has a complete solution across the customer adoption buying lifecycle with full visibility being the foundational layer



ACTION

Tailor user journeys in real time according to user behavior, trust and risk

DECISIONING

Deploy policies and machine learning models without risk

ENRICHMENT

Conditionally enrich risk decisions based on risk, cost and efficacy

COMPLETE VISIBILITY

Real-time and historic journey intelligence across entire digital estate

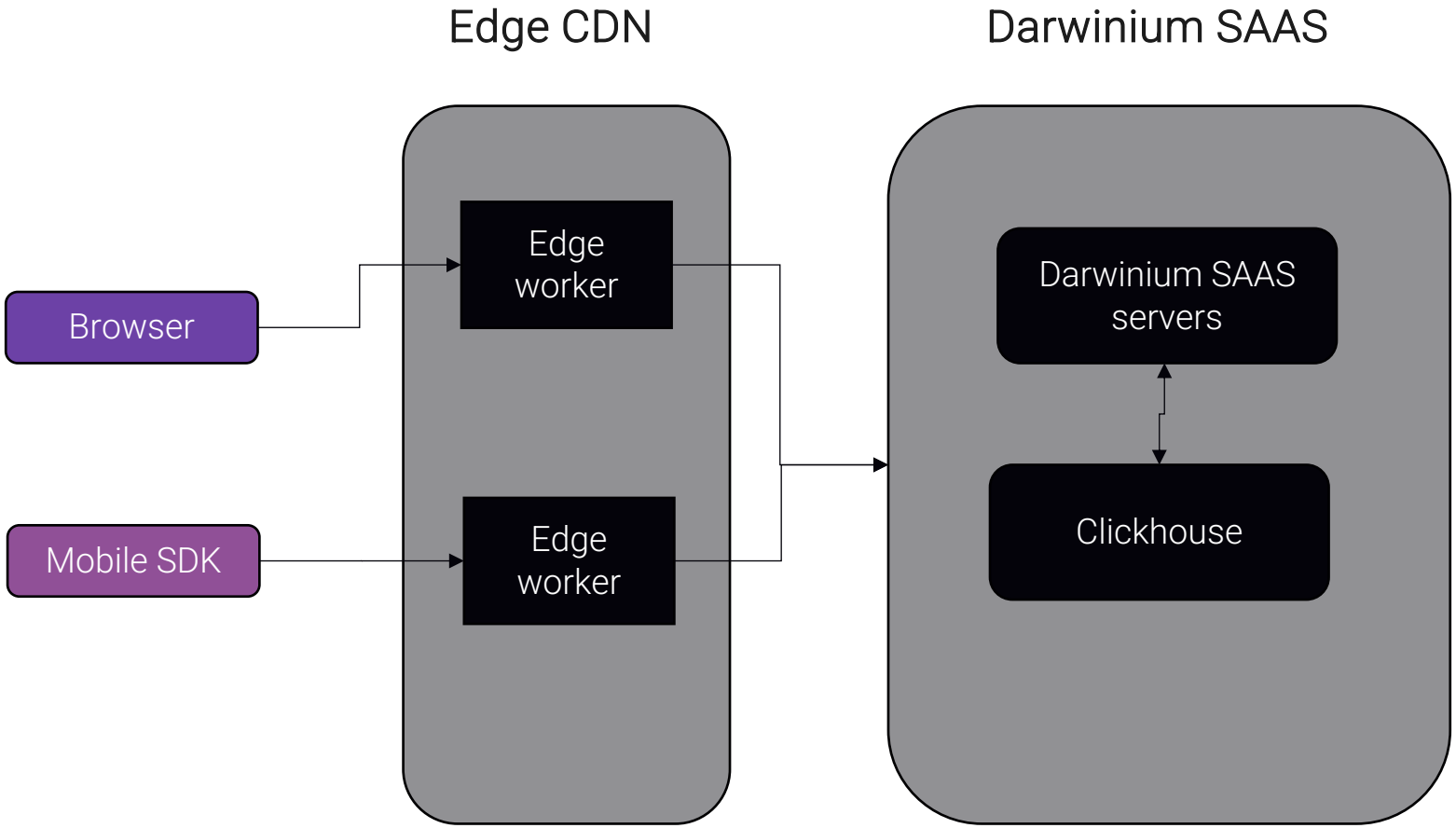
Darwinium

Key Features built using Clickhouse

Unifying Intelligence Across Use Cases: Key Challenges

- Unifying security and fraud:
  - Very Large datasets – Billions of rows
  - Lowest interactive query response times

- Partitioning for multi-tenancy
- Distributed Tables

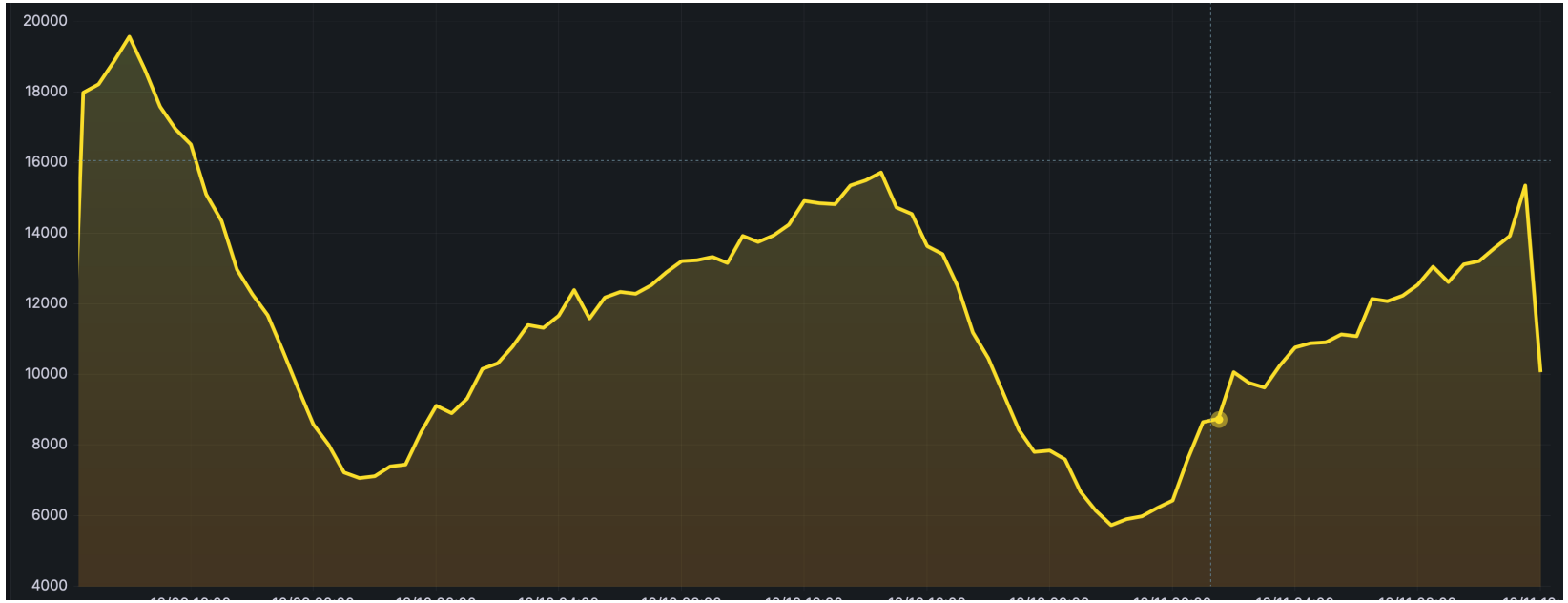


Unifying Intelligence Across Use Cases: Key Challenges

- Thousands of columns – Wide denormalized tables
  - Ex: Postgres could not support > 1600 columns
- Rich Data types
  - Maps for features
  - Tuples – Table within a Table
- Codecs
- Rich catalogue of functions
- Location analytics
  - H3 Indexes
  - Polygons

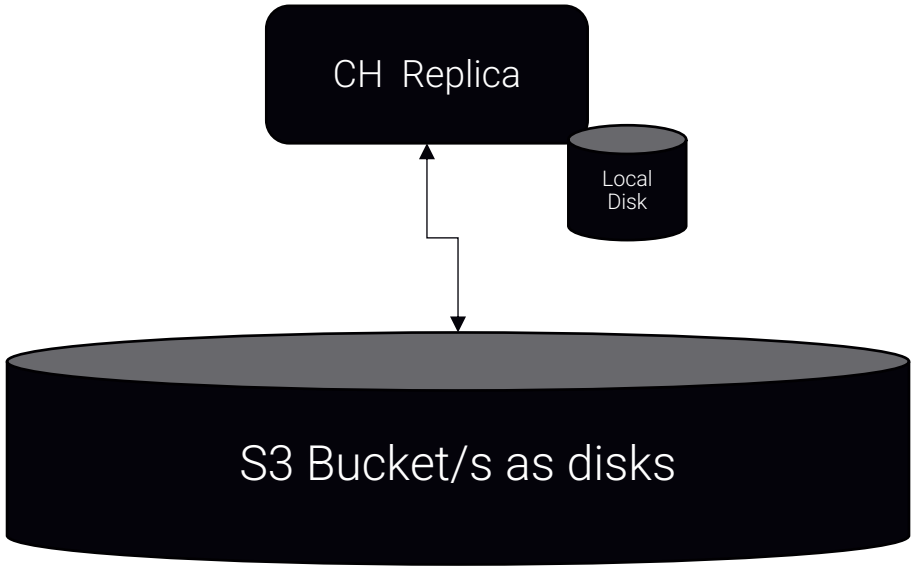
Unifying Intelligence Across Use Cases

- Real time dashboards over very large datasets. Ticker graphs use:
  - Sketches
    - Compact data structures
    - Partitioned by write threads avoids complex data pipelines
  - Multiple and flexible time windows possible
    - Last N minutes
    - Wall clock intervals
  - Heavy hitters
    - Most common Signals
  - Quantiles
    - Distribution metrics



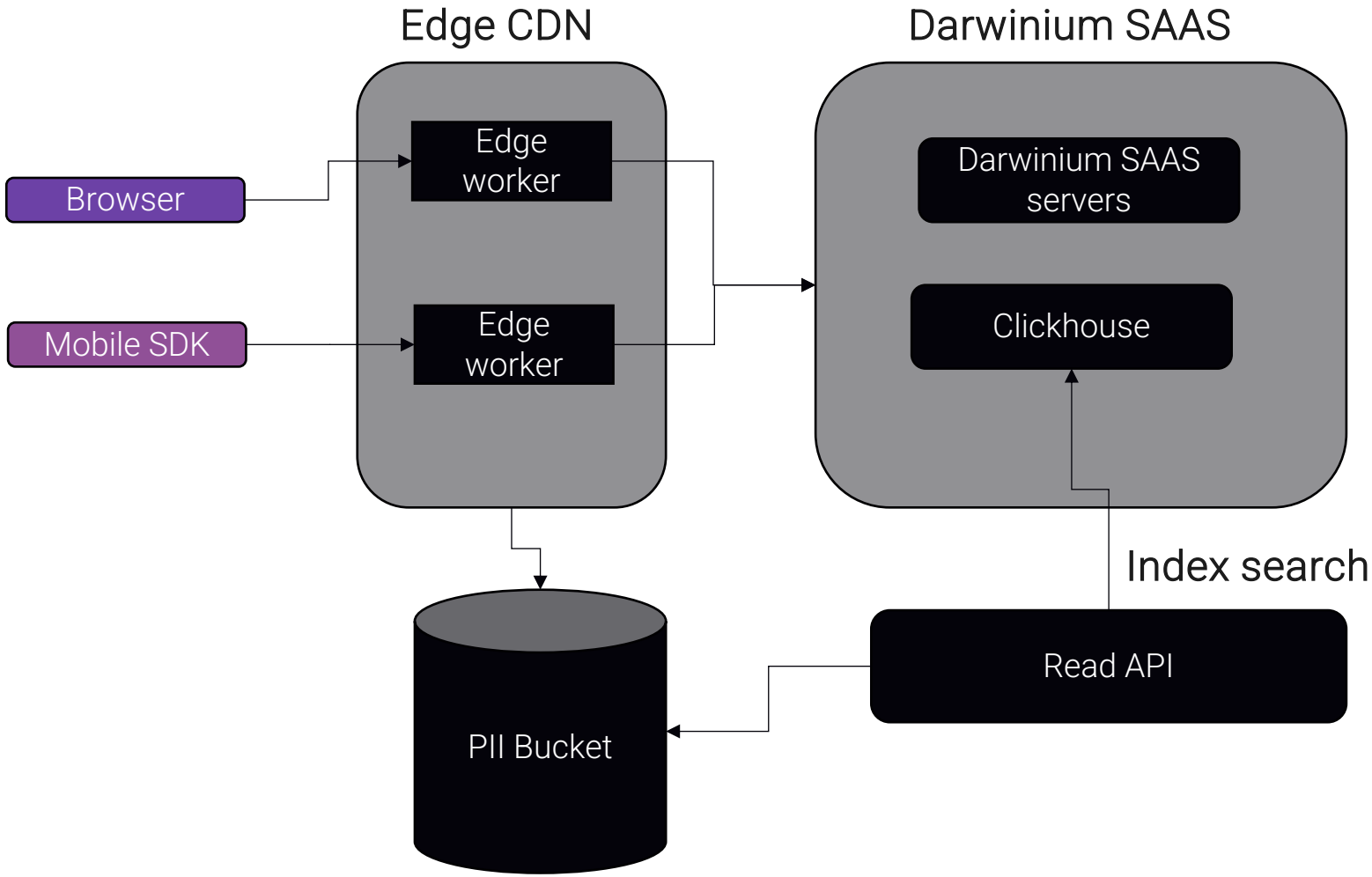
Cloud Native - AWS

- Compute and Storage Decoupled
  - Define disks
- S3 disks for tiered storage
  - Server with ”unlimited” storage
- S3 for backups
  - Local disk sizes no longer a limitation
  - Pick your instance types that suit your read/write load combinations
- Clickhouse cloud the way to go for more optimal storage and compute patterns.



Cloud Native – With Columnar Databases

- All columns fetch – an anti pattern for Columnar databases.
  - Typically needed for better analysis for fraud and security use cases
  - A few thousand columns for a single row.
- Clickhouse as an indexing/search system.
  - Full row fetched from S3
  - Better privacy positioning for our customers.



Cost-Optimized implementation using Clickhouse constructs

Space gains using Codecs

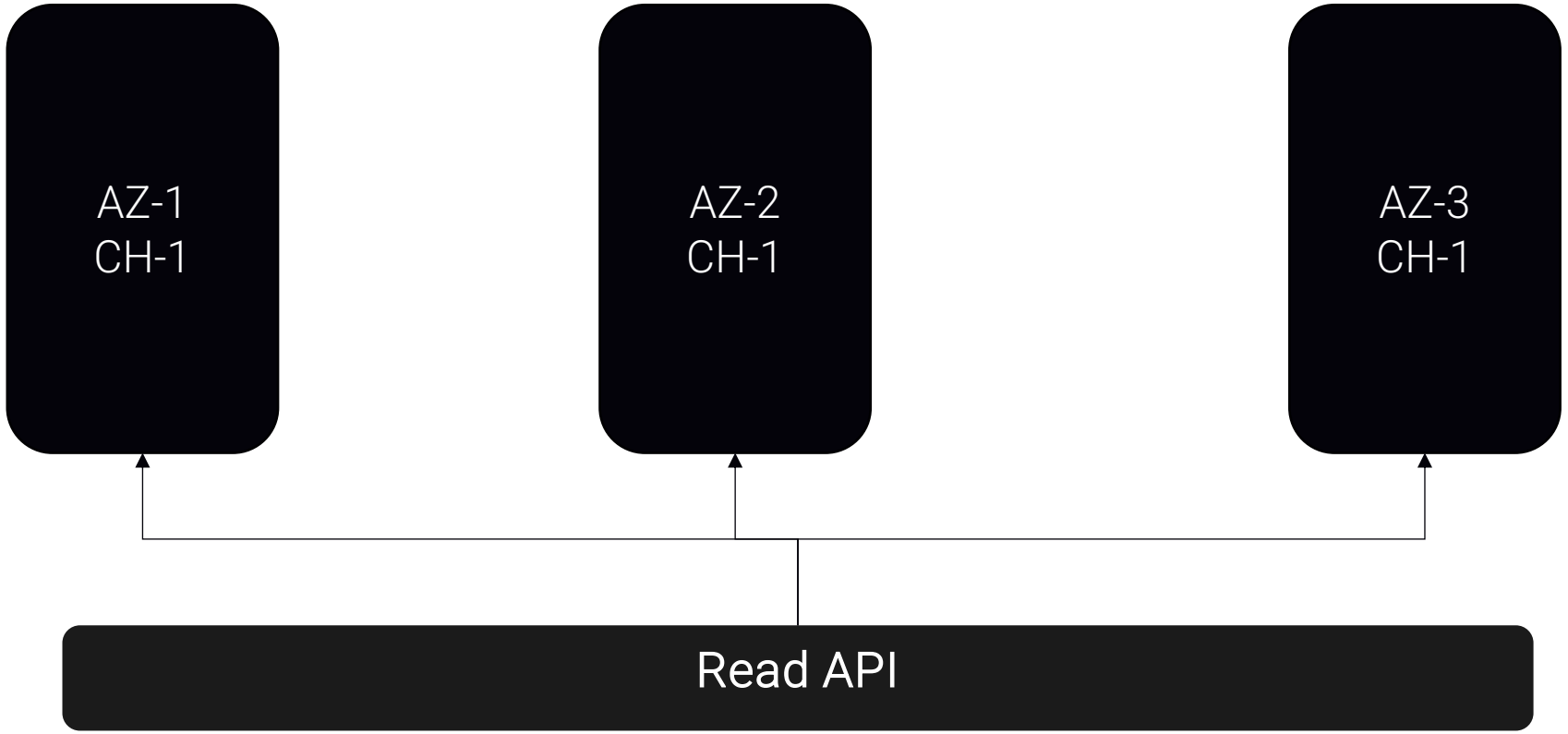
TTL for easier data management

Hot versus cold data

Caching for latencies

AWS Marketplace Certification

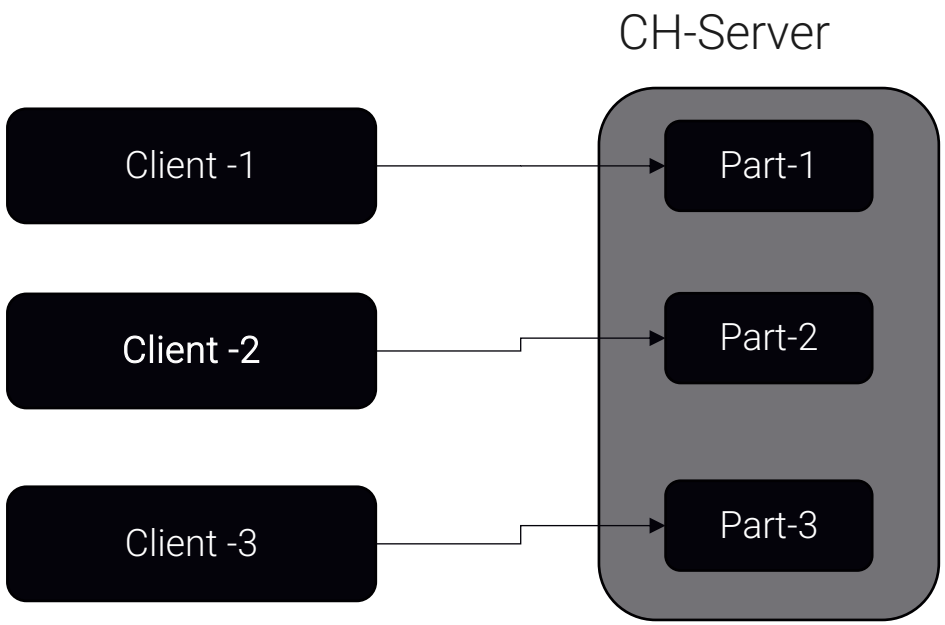
- Replicated engines across availability Zones
  - Single Shard (CH-1) replicated across 3 availability zones

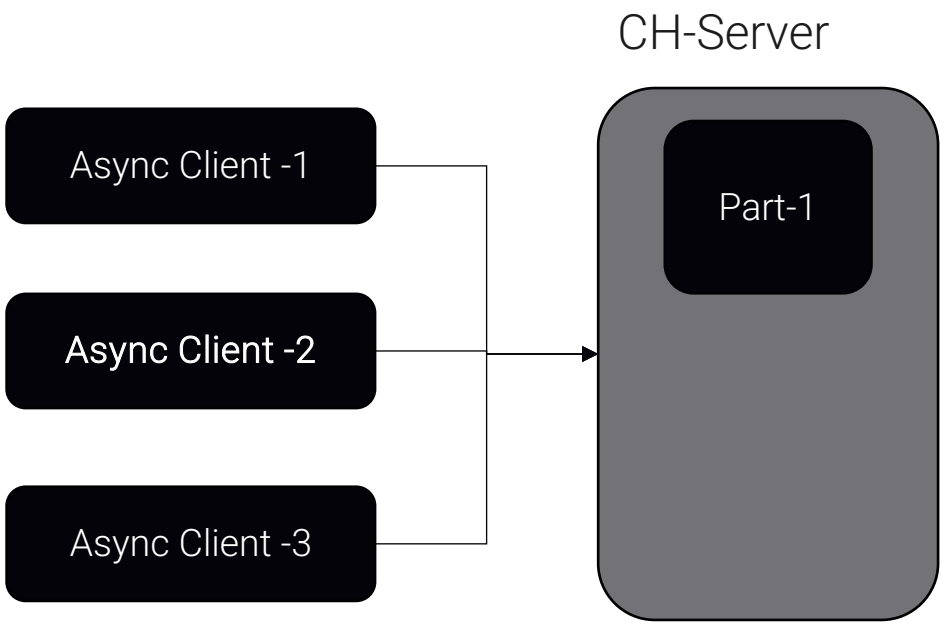


Data Pipelines

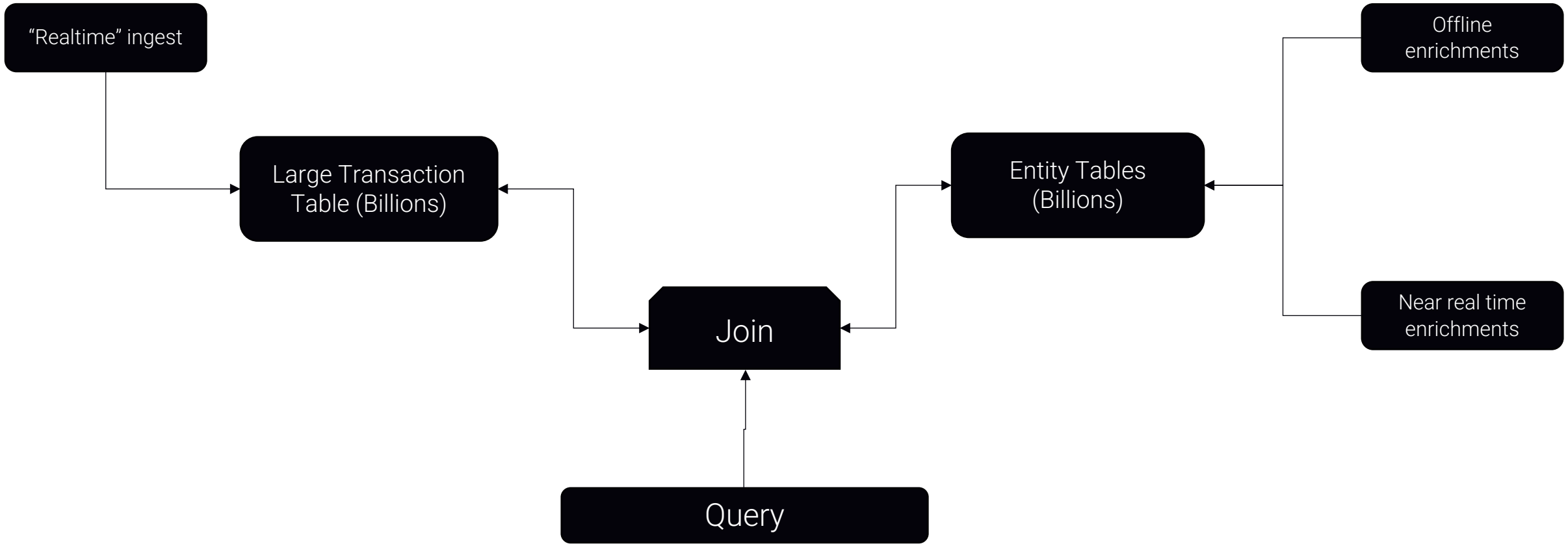
- Data de-duplication
  - Limit 1 BY ..
- GO client
  - Rich features made dynamic schema management possible
  - Batch Appends on a per column basis
- CH Keeper - A ZK alternative for your data pipeline coordination

Async Inserts – High Throughput with Lesser Parts





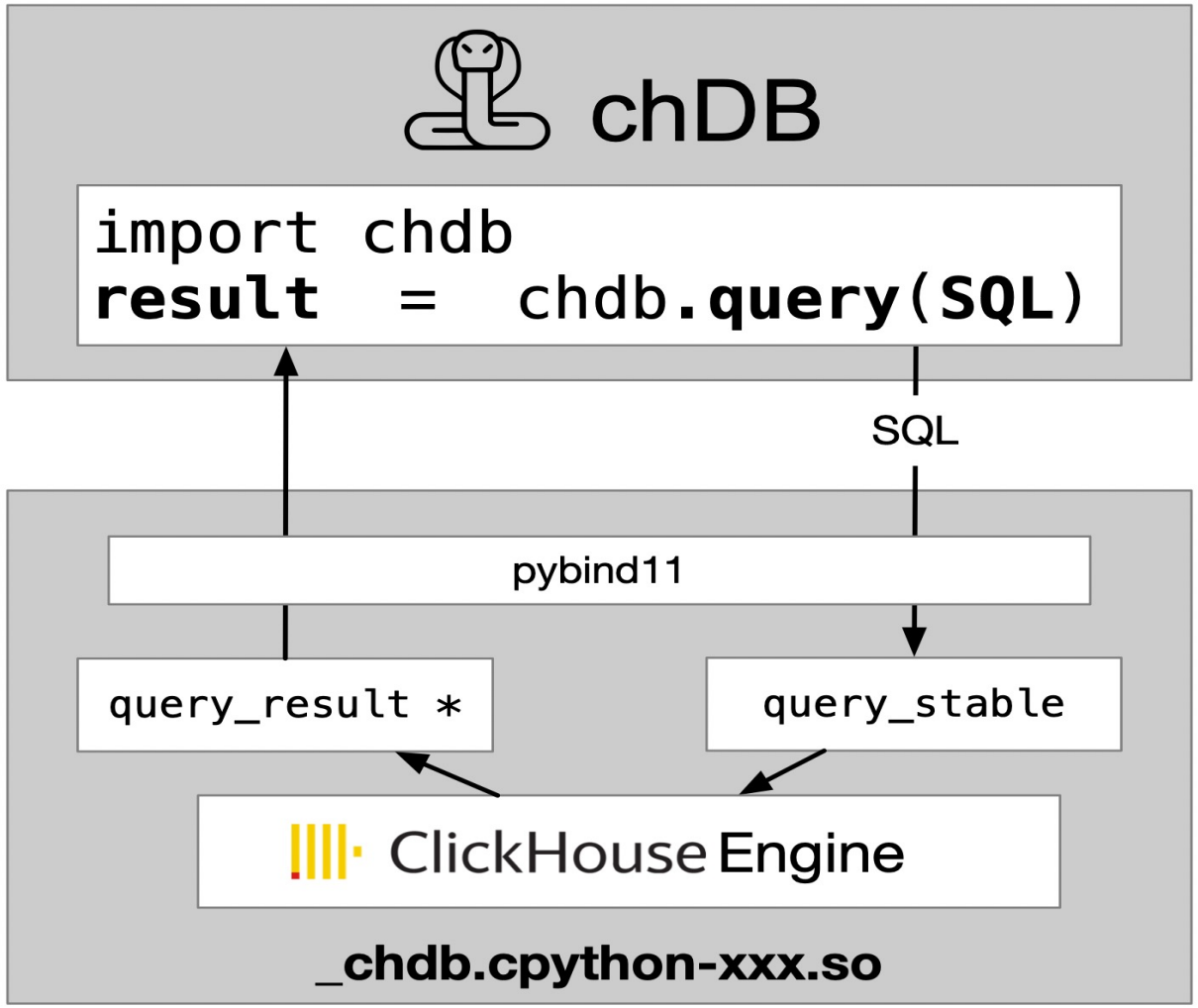
Fast Point Lookups with Dictionaries - Merge Clickhouse with KV stores



Darwinium

Clickhouse: Python & Feature Engineering

CHDB: Embedded CH for Python



Query on S3/HTTP/File/Another ClickHouse

```
# Run chDB on HTTP (Parquet, CSV, JSON ...)
!wget 'https://datasets.clickhouse.com/hits_compatible/athena_partitioned/hits_0.parquet' -q -O hits_0.parquet
import chdb

data = "url('https://datasets.clickhouse.com/hits_compatible/athena_partitioned/hits_0.parquet')"
# data = "file('hits_0.parquet', Parquet)"
# data = "s3('xxx')"

sql = f"""SELECT RegionID, SUM(AdvEngineID), COUNT(*) AS c, AVG(ResolutionWidth), COUNT(DISTINCT UserID)
                    FROM {data} GROUP BY RegionID ORDER BY c DESC LIMIT 10"""
ret = chdb.query(sql, 'dataframe')
```

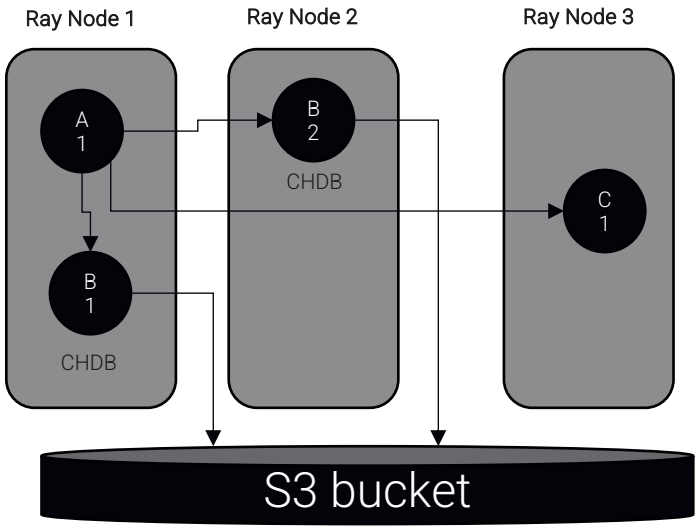
Feature Engineering Pipelines with Ray and CH

- Ray
  - An open-source unified compute framework that makes it easy to scale AI and Python workloads
- Ray Actor
  - A stateful worker that can carry out Ray tasks

- SQL
  - Perhaps worlds most adopted language (After English) among data scientists and engineers

Ray with Clickhouse

- A flexible framework
- Pure pythonic
- Supports SQL based feature generation in the same pipeline definition
- Rich catalogue of functions
- Build upon Ray Ecosystem



Darwinium

Q&A