SUMMIT ONLINE

Modernize your data warehouse

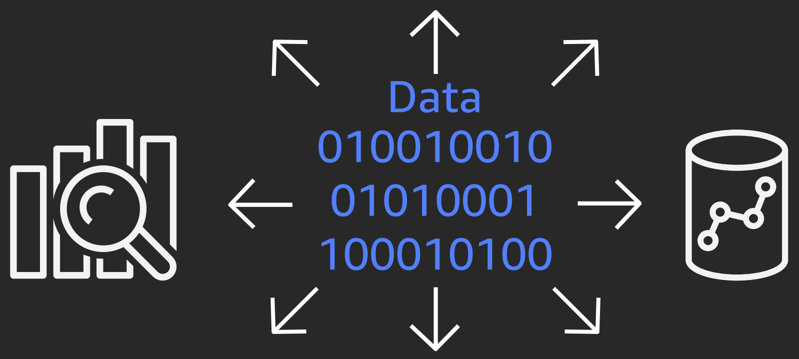
Aneesh Chandra PN
Data & Analytics Specialist Solutions Architect
Amazon Web Services

AWS SUMMIT ONLINE

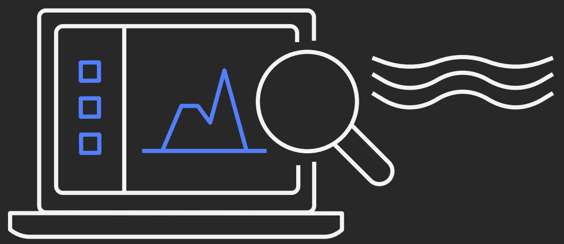
Agenda

- Modern analytics and Amazon Redshift
- Architecture and concepts
- Accelerating your data warehouse migration
- Additional resources
- Q&A

Data warehousing trends



Exponential growth
of event data



End-to-end insights from
analyzing all of your data

Migrations
to the cloud

Why Amazon Redshift

Tens of thousands of customers use Amazon Redshift and process over 2 EB of data per day

Data lake & AWS integrated

Lake formation catalog & security

Exabyte querying & AWS integrated
(e.g., AWS DMS, Amazon CloudWatch)

Best performance

3x faster than other
cloud data warehouses

Best value

Usage-based, RIs

Predictable costs

Most scalable

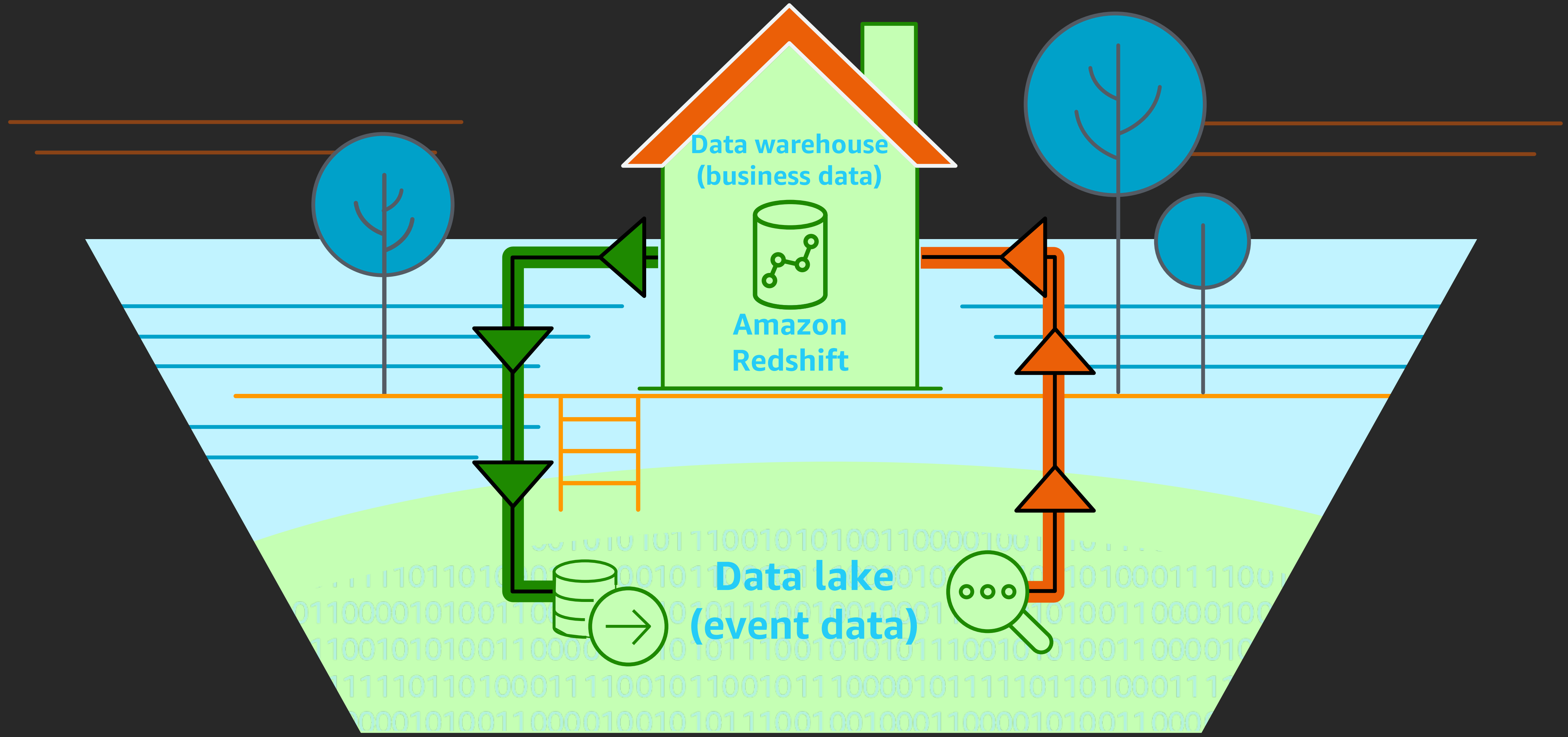
Virtually unlimited
elastic linear scaling

Most secure & compliant

AWS-grade security (e.g., VPC, encryption
with AWS KMS, AWS CloudTrail)

Easy to manage

Easy to provision & manage, automated
backups, AWS support, and 99.9% SLAs



Customers moving to data lake architectures

Amazon Redshift enables you to have a lake house approach

EQUINOX

"Moving to Amazon Redshift has helped us reduce our overall maintenance costs by nearly 80% compared with our legacy data warehouse. By leveraging Amazon Redshift Spectrum's ability to query data directly in our Amazon S3 data lake, we have been able to easily integrate new data sources in hours, not days or weeks. This has not only reduced our time to insight, but it helped us control our infrastructure costs. Amazon Redshift requires very little maintenance, to the point where we don't even have a dedicated administrator, and we spend less than an hour a month on maintenance and administration."

Elliott Cordo
VP of Data Analytics
Equinox

Tens of thousands of customers use Amazon Redshift

23andMe | 3M Science. Applied to Life. | aetna | amazon | ANA | ancestry.com | AstraZeneca | Bristol-Myers Squibb | Chick-fil-A | CIVIS

CIVITAS LEARNING | coinbase | coursera | EA | EQUINOX | FINANCIAL TIMES | FOURSQUARE | Gainsight | GP Georgia-Pacific | HEARST corporation | HelloFRESH

hudl | INNOVID | intuit | Jack in the box | J.D. POWER | KICKSTARTER | Kimberly-Clark | Klarna. | Liberty Mutual | lyft

MANTHAN | Nasdaq | NTT docomo | NU SKIN | The Ohio State University | OLX GROUP | PADDYPOWER betfair | Periscope Data | Pfizer | Pinterest

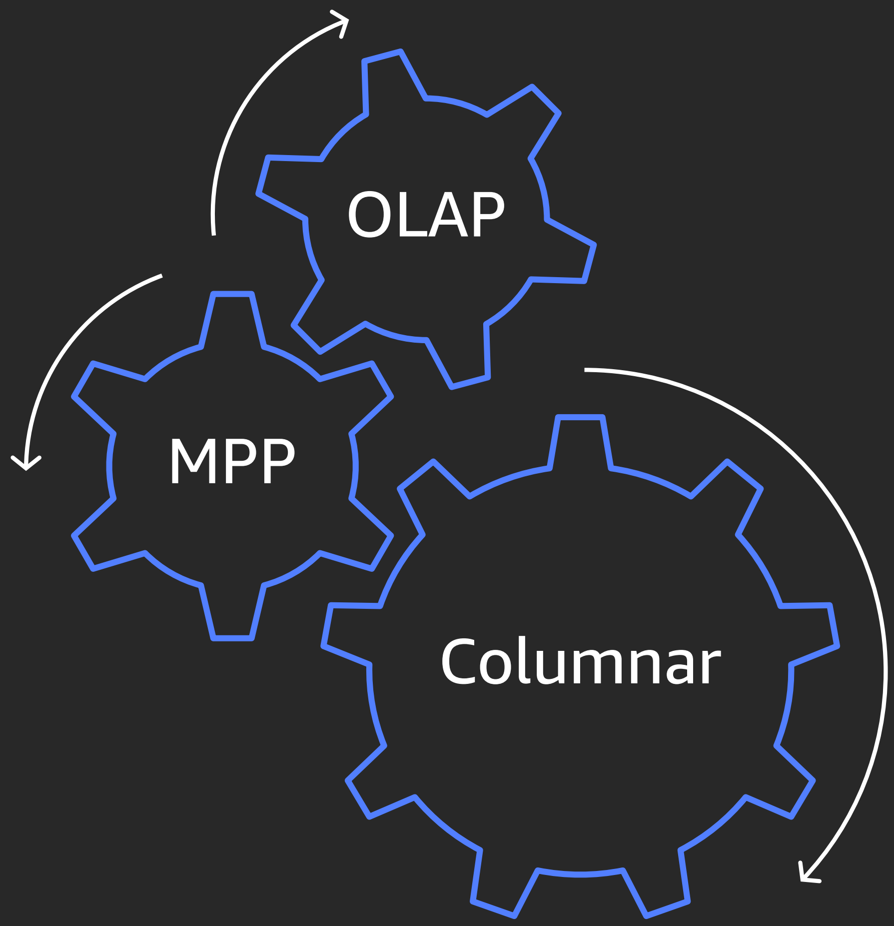
Prezi | REDFIN | robinhood | SCOPELY | SOUNDCLOUD | SUBWAY | Sysco | tinder | volaris | yelp

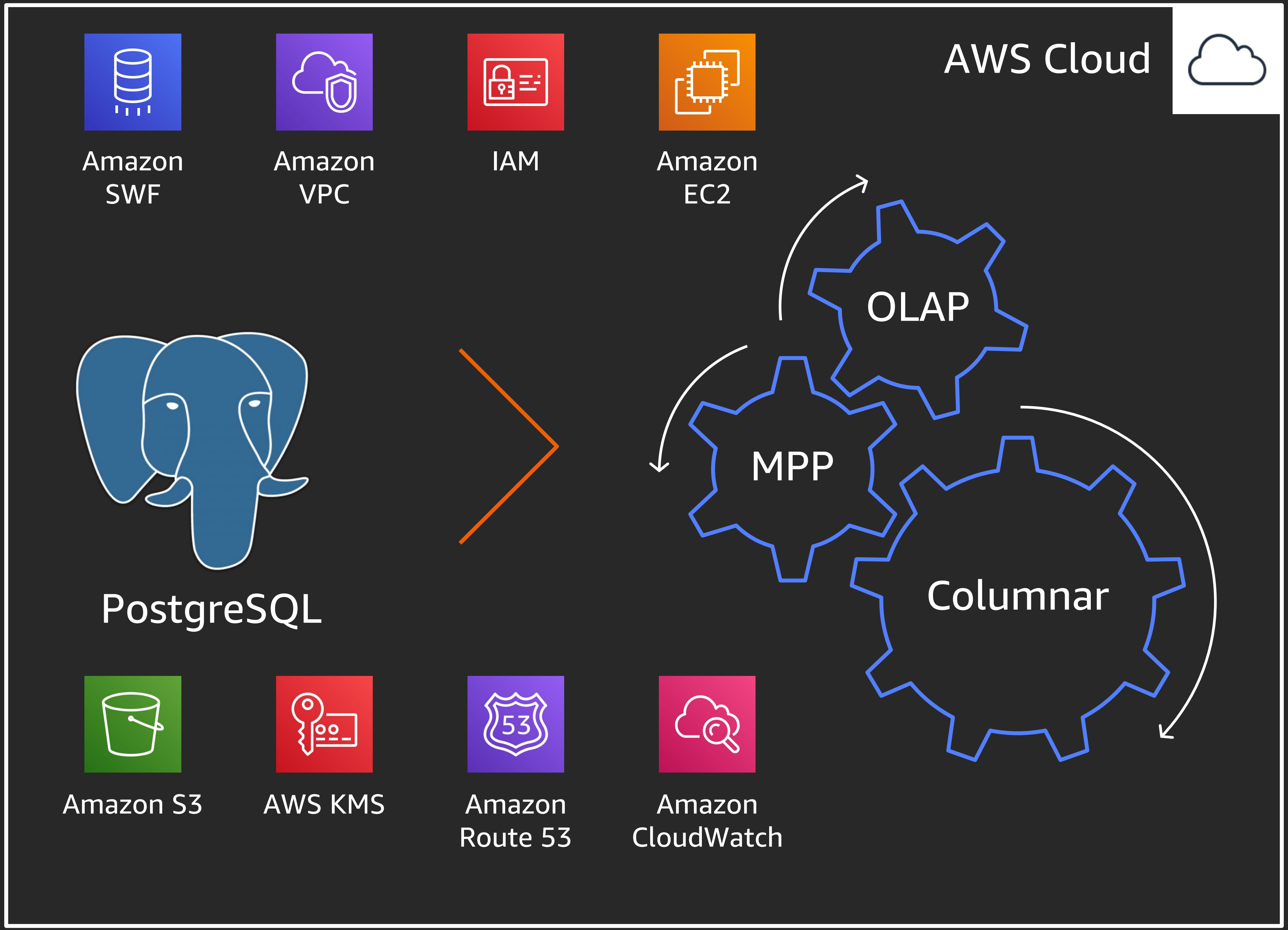
Amazon Redshift – evolving architecture

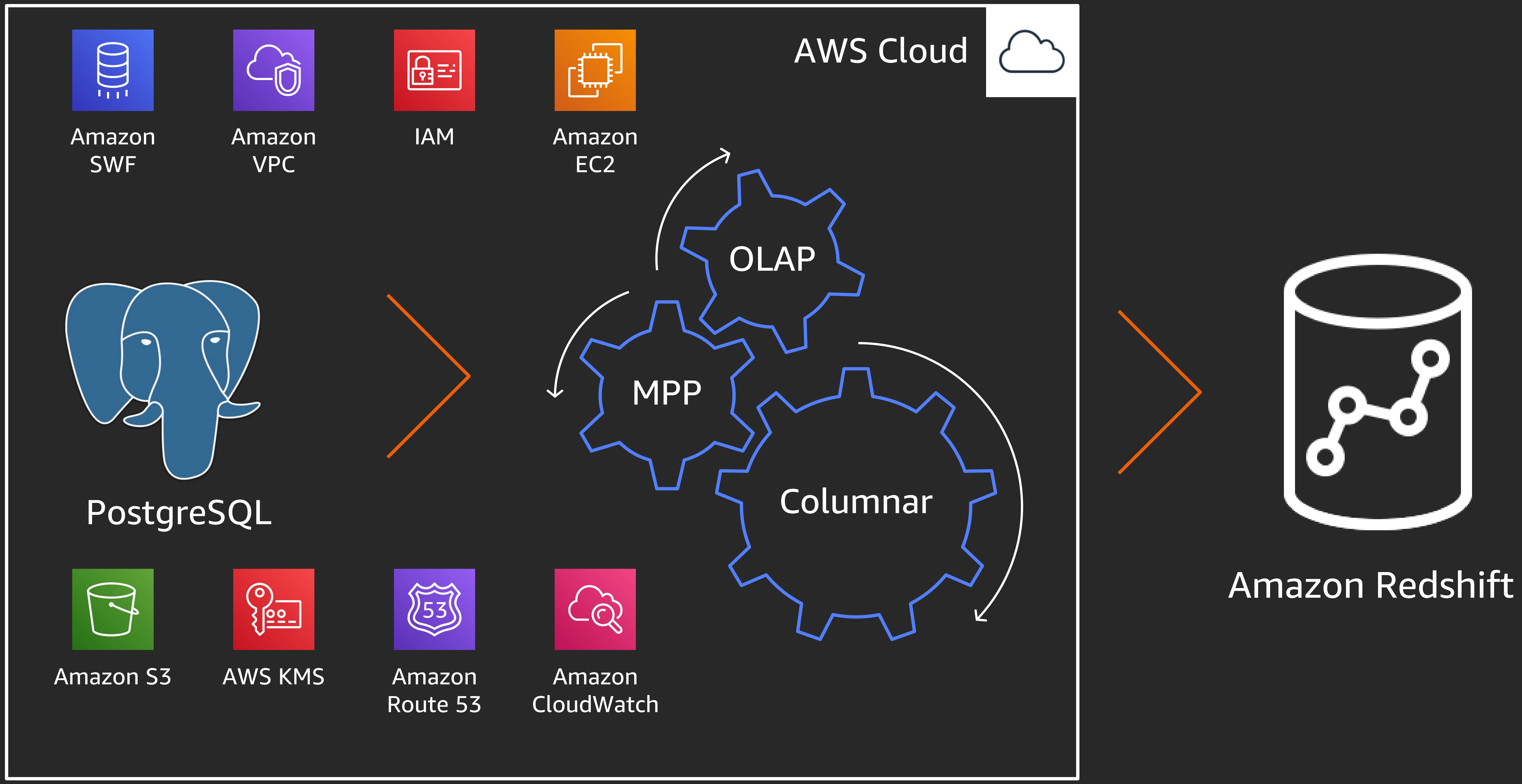
AWS SUMMIT ONLINE

PostgreSQL

PostgreSQL




AWS Cloud
Amazon SWF
Amazon VPC
IAM
Amazon EC2
PostgreSQL
OLAP
MPP
Columnar
Amazon S3
AWS KMS
Amazon Route 53
Amazon CloudWatch


AWS Cloud
Amazon SWF
Amazon VPC
IAM
Amazon EC2
PostgreSQL
OLAP
MPP
Columnar
Amazon S3
AWS KMS
Amazon Route 53
Amazon CloudWatch
Amazon Redshift

Features delivered to meet customer needs

Robust result set caching

Large # of tables support ~20,000

Copy command support for ORC, Parquet

IAM role chaining

Elastic resize

Groups

Amazon Redshift Spectrum: date formats, scalar JSON and ION file format support, region expansion, predicate filtering

Auto-analyze

Health and performance monitoring with Amazon CloudWatch

Automatic table distribution style

CloudWatch support for WLM queues

Performance enhancements: hash join, vacuum, window functions, resize ops, aggregations, console, union all, efficient compile code cache

Unload to CSV

Auto WLM

~25 query monitoring rules (QMR) support

AQUA (Advanced Query Accelerator)

Concurrency scaling

200+

new features in the past 18 months

DC1 migration to DC2

Resiliency of ROLLBACK processing

Manage multi-part query in AWS console

Autoanalyze for incremental changes on table

Spectrum Request Accelerator

Apply new distribution key

Amazon Redshift Spectrum: Row group filtering in Parquet and ORC, nested data support, enhanced VPC routing, multiple partitions

Faster classic resize with optimized data transfer protocol

Performance: Bloom filters in joins, complex queries that create internal table, communication layer

Amazon Redshift Spectrum: Concurrency scaling

Integration with AWS Lake Formation

Autovacuum sort, autoanalyze, and autotable sort

Auto WLM with query priorities

Snapshot scheduler

Stored procedures

Performance: Join pushdowns to subquery, mixed workload temporary tables, rank functions, null handling in join, single row insert

Advisor recommendations for distribution keys

AZ64 compression encoding

Console redesign

Spatial processing

Column-level access control with AWS Lake Formation

RA3

Performance of inter-region snapshot transfers

Federated query

Materialized views

Manual pause and resume

Amazon Redshift architecture

Massively parallel, shared-nothing columnar architecture

Leader node

SQL endpoint

Stores metadata

Coordinates parallel SQL processing

Compute nodes

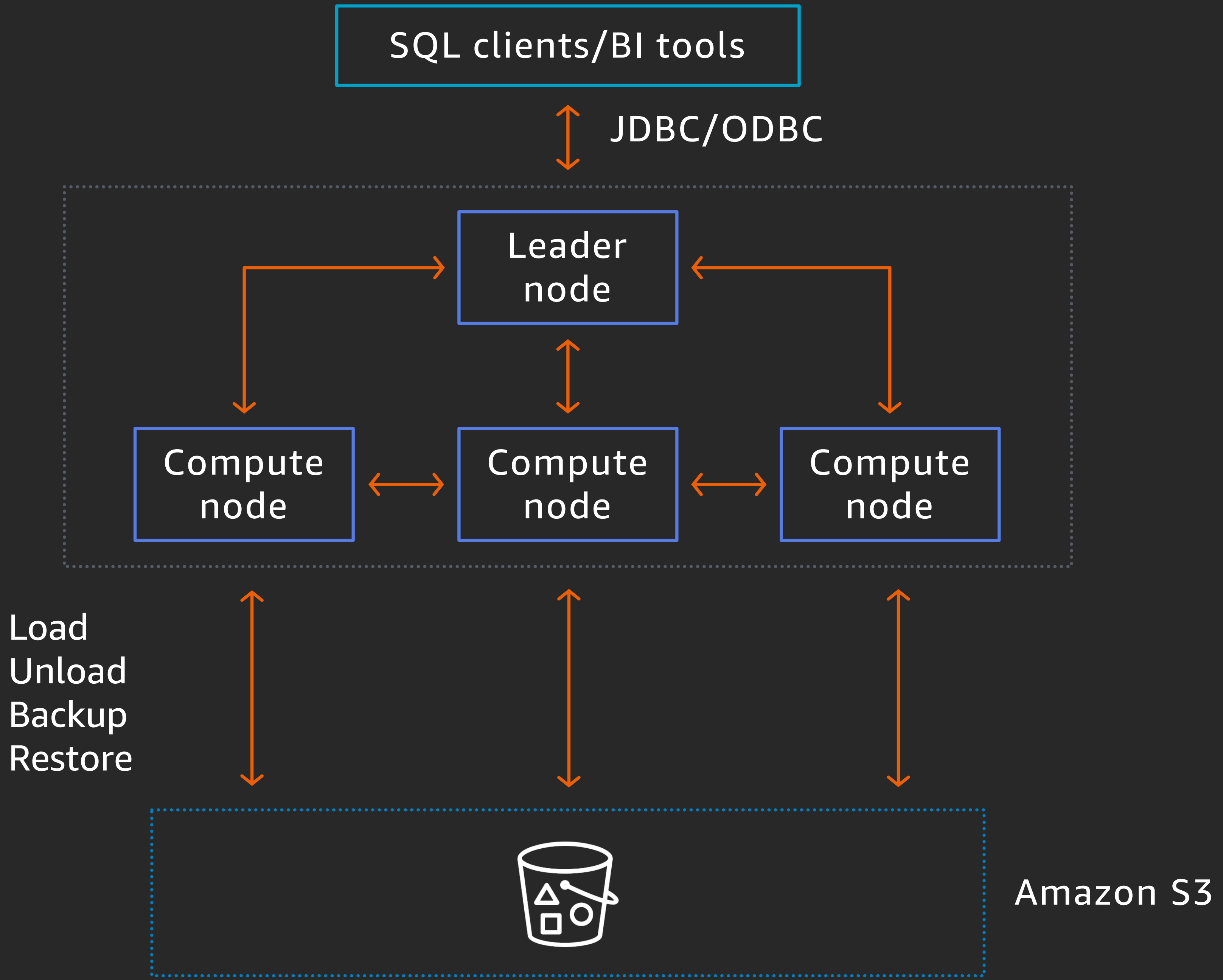
Local, columnar storage

Executes queries in parallel

Load, unload, backup, restore

Amazon Redshift Spectrum nodes

Execute queries directly against Amazon S3



Amazon Redshift architecture

Massively parallel, shared-nothing columnar architecture

Leader node

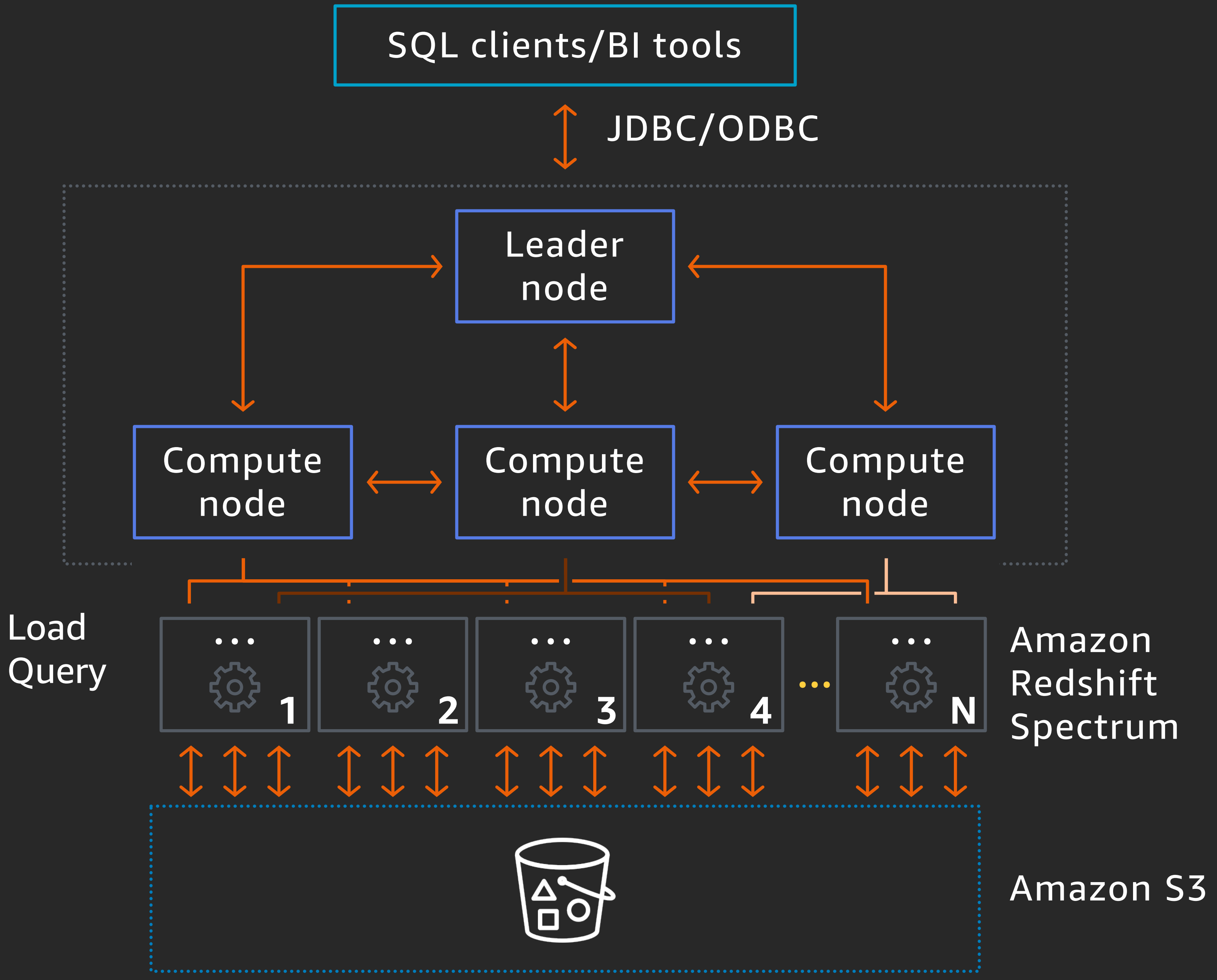
- SQL endpoint
- Stores metadata
- Coordinates parallel SQL processing

Compute nodes

- Local, columnar storage
- Executes queries in parallel
- Load, unload, backup, restore

Amazon Redshift Spectrum nodes

- Execute queries directly against Amazon S3



Amazon Redshift evolving architecture

Massively parallel, shared-nothing columnar architecture

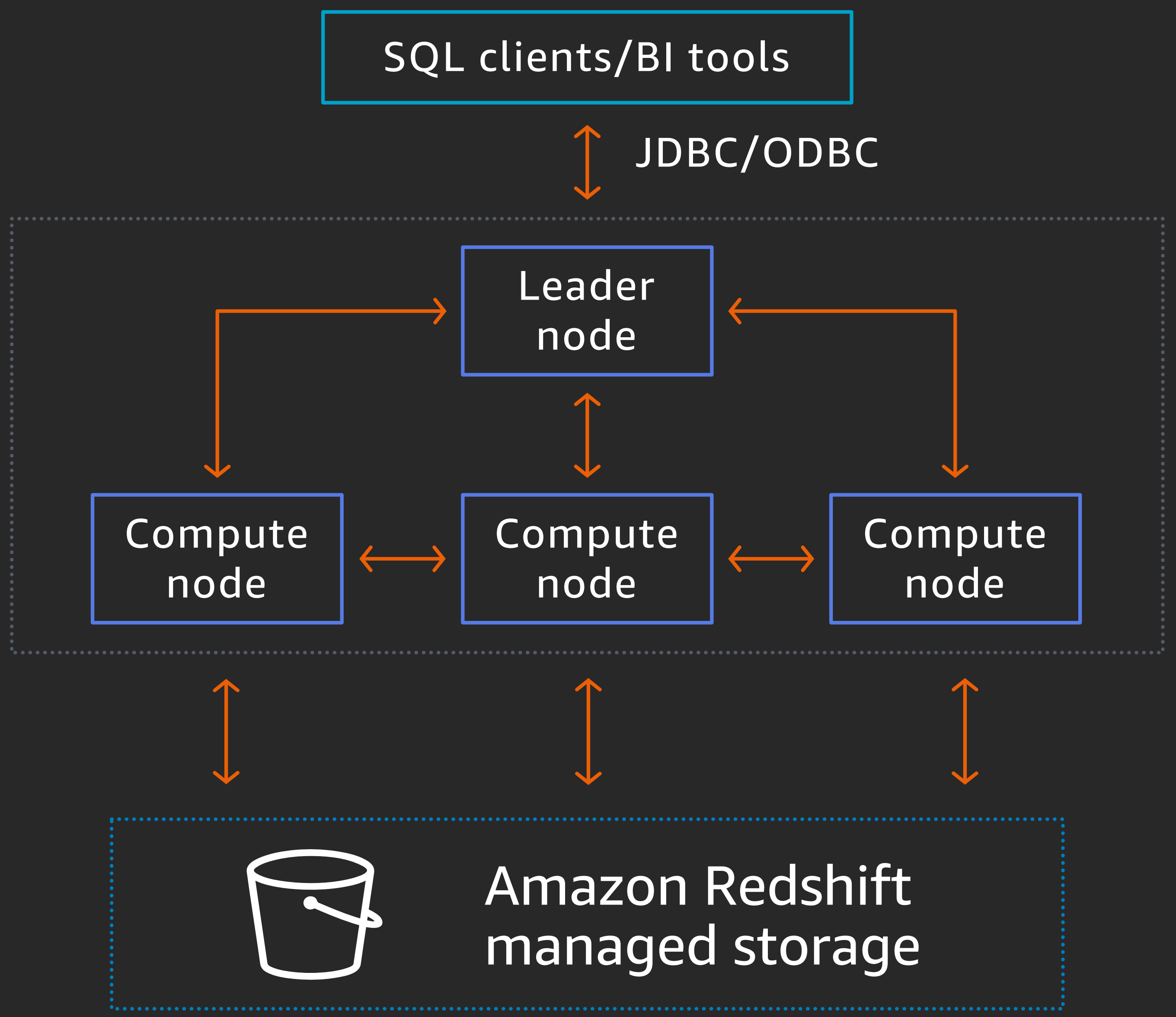
Leader node

Compute nodes

Amazon Redshift Spectrum nodes

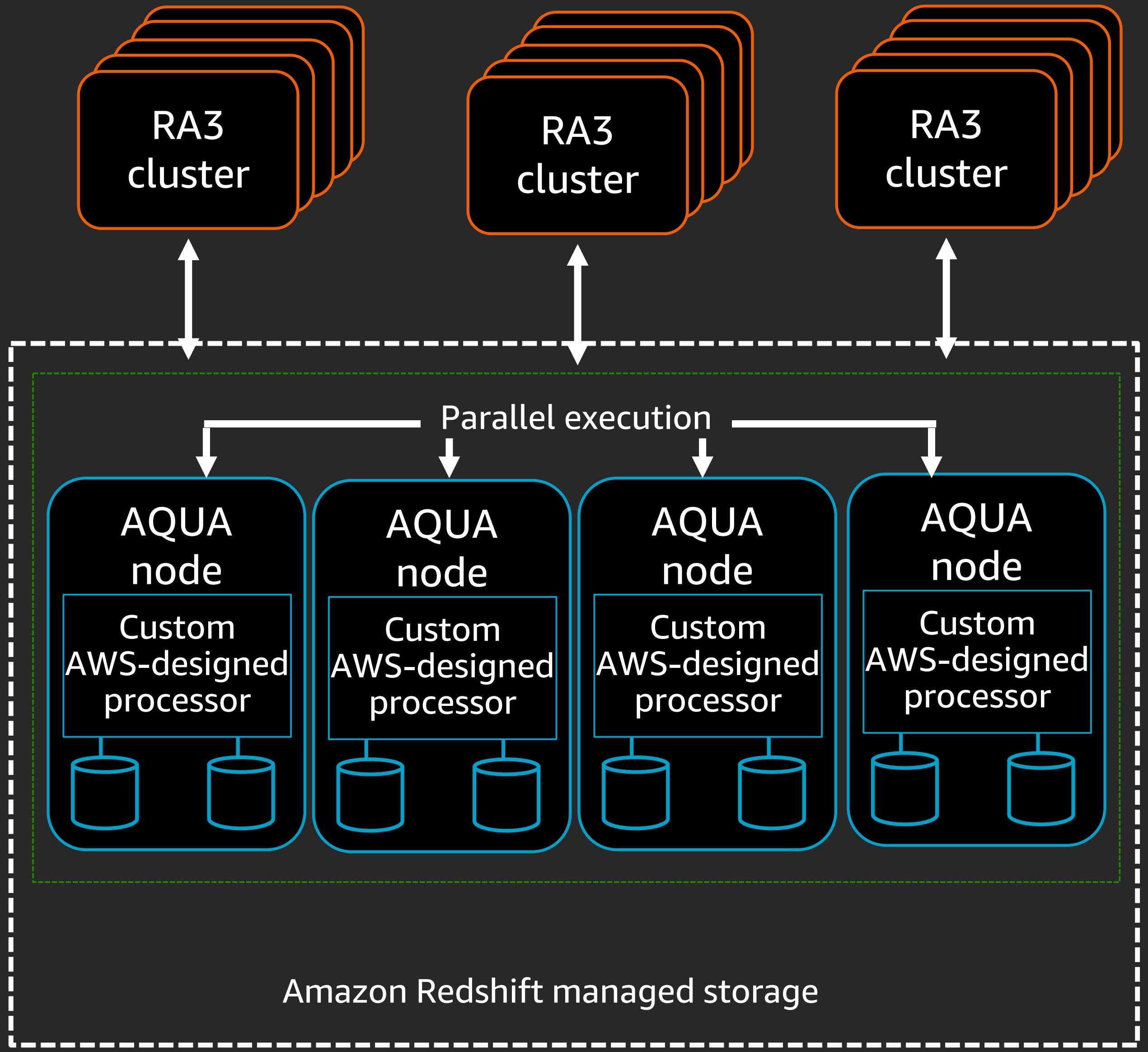
Amazon Redshift managed storage

- Pay separately for storage and compute
- Large high-speed SSD-backed cache
- Automatic scaling (up to 64 TB/instance)
- Supports up to 8.2 PB of cluster storage



AQUA: Advanced Query Accelerator (preview)

A new distributed and hardware-accelerated processing layer that will make Amazon Redshift 10x faster than any other cloud data warehouse without increasing cost



Minimize data movement over the network by pushing down operations to AQUA nodes

AQUA nodes with custom AWS-designed analytics processors to make operations (compression, encryption, filtering, and aggregations) faster than traditional CPUs

Available in preview only with RA3 – no code changes required

Terminology and concepts

Terminology and concepts: Node types

Amazon Redshift analytics – RA3 (new)

Amazon Redshift managed storage – solid-state disks + Amazon S3

Dense compute – DC2

Solid-state disks

Dense storage – DS2

Magnetic disks

Instance type	Disk type	Size	Memory	CPUs	Slices
RA3 4xlarge (new)	RMS	Scales to 16 TB	96 GB	12	4
RA3 16xlarge (new)	RMS	Scales to 64 TB	384 GB	48	16
DC2 large	SSD	160 GB	16 GB	2	2
DC2 8xlarge	SSD	2.56 TB	244 GB	32	16
DS2 xlarge	Magnetic	2 TB	32 GB	4	2
DS2 8xlarge	Magnetic	16 TB	244 GB	36	16

Terminology and concepts: Columnar

Amazon Redshift uses a columnar architecture for storing data on disk

---

Goal: Reduce I/O for analytics queries

Physically store data on disk by column rather than row

Read only the column data that is required

Example: Columnar architecture

```
CREATE TABLE deep_dive (
     aid  INT         --audience_id
    ,loc  CHAR(3)     --location
    ,dt   DATE        --date
);
```

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14


```
SELECT min(dt) FROM deep_dive;
```

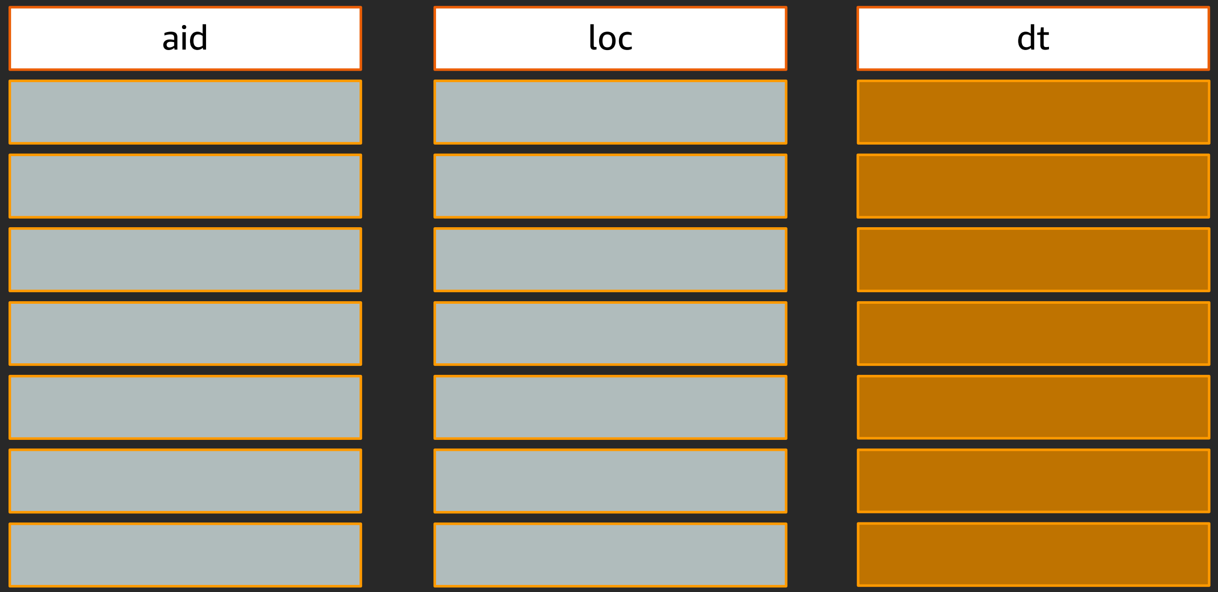
Row-based storage

- Need to read everything
- Unnecessary I/O

Example: Columnar architecture

```sql
CREATE TABLE deep_dive (
     aid  INT         --audience_id
    ,loc  CHAR(3)     --location
    ,dt   DATE        --date
);
```

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14




```sql
SELECT min(dt) FROM deep_dive;
```

Column-based storage

Only scan blocks for relevant column

Terminology and concepts: Compression

Goals

Allow more data to be stored within an Amazon Redshift cluster

Improve query performance by decreasing I/O

Impact

Allows 2–4x more data to be stored within the cluster

By default, COPY automatically analyzes and compresses data on first load into an empty table

ANALYZE COMPRESSION is a built-in command that finds the optimal compression for each column on an existing table

Compression example

```
CREATE TABLE deep_dive (
     aid  INT        --audience_id
    ,loc  CHAR(3)    --location
    ,dt   DATE       --date
);
```

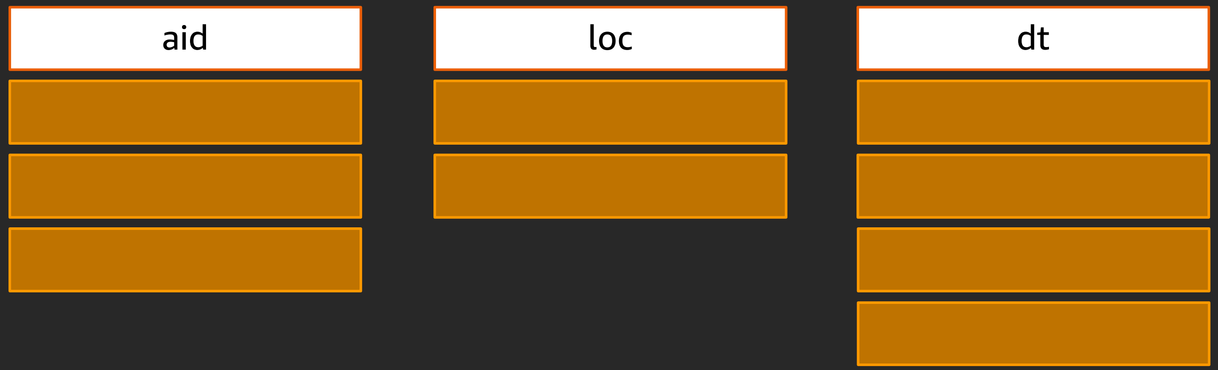
aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14

Add 1 of 13 different encodings to each column

Compression example

```
CREATE TABLE deep_dive (
     aid  INT        ENCODE AZ64
    ,loc  CHAR(3)    ENCODE BYTEDICT
    ,dt   DATE       ENCODE RUNLENGTH
);
```

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14



- More efficient compression by storing the same data type in the columnar architecture
- Columns grow and shrink independently
- Reduces storage requirements
- Reduces I/O

Terminology and concepts: Blocks

Column data is persisted to 1 MB immutable blocks

Blocks are individually encoded with 1 of 13 encodings

A full block can contain millions of values

Terminology and concepts: Zone maps

Goal

Eliminates unnecessary I/O

In-memory block metadata

- Contains per-block minimum and maximum values
- All blocks automatically have zone maps
- Effectively prunes blocks that cannot contain data for a given query

Terminology and concepts: Data sorting

Goal

Make queries run faster by increasing the effectiveness of zone maps and reducing I/O

Impact

Enables range-restricted scans to prune blocks by leveraging zone maps

Achieved with the table property SORTKEY defined on one or more columns

Optimal sort key is dependent on:

- Query patterns
- Business requirements
- Data profile

Example: Sort key

```
CREATE TABLE deep_dive (
     aid   INT         --audience_id
    ,loc   CHAR(3)     --location
    ,dt    DATE        --date
) SORTKEY (dt, loc);
```

Add a sort key to one or more columns to physically sort the data on disk

deep_dive		
aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14

Example: Sort key

```
CREATE TABLE deep_dive (
     aid   INT         --audience_id
    ,loc   CHAR(3)     --location
    ,dt    DATE        --date
) SORTKEY (dt, loc);
```

Add a sort key to one or more columns to physically sort the data on disk

deep_dive		
aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14

deep_dive (sorted)		
aid	loc	dt
3	SFO	2017-04-01
4	JFK	2017-05-14
2	JFK	2017-10-20
1	SFO	2017-10-20

Example: Zone maps and sorting

```
SELECT count(*) FROM deep_dive WHERE dt = '06-09-2017';
```

Unsorted table

MIN: 01-JUNE-2017
MAX: 20-JUNE-2017

MIN: 08-JUNE-2017
MAX: 30-JUNE-2017

MIN: 12-JUNE-2017
MAX: 20-JUNE-2017

MIN: 02-JUNE-2017
MAX: 25-JUNE-2017

Example: Zone maps and sorting

```
SELECT count(*) FROM deep_dive WHERE dt = '06-09-2017';
```

Unsorted table

MIN: 01-JUNE-2017
MAX: 20-JUNE-2017

MIN: 08-JUNE-2017
MAX: 30-JUNE-2017

MIN: 12-JUNE-2017
MAX: 20-JUNE-2017

MIN: 02-JUNE-2017
MAX: 25-JUNE-2017

Example: Zone maps and sorting

```
SELECT count(*) FROM deep_dive WHERE dt = '06-09-2017';
```

Unsorted table

MIN: 01-JUNE-2017
MAX: 20-JUNE-2017

MIN: 08-JUNE-2017
MAX: 30-JUNE-2017

MIN: 12-JUNE-2017
MAX: 20-JUNE-2017

MIN: 02-JUNE-2017
MAX: 25-JUNE-2017

Sorted by date

MIN: 01-JUNE-2017
MAX: 06-JUNE-2017

MIN: 07-JUNE-2017
MAX: 12-JUNE-2017

MIN: 13-JUNE-2017
MAX: 21-JUNE-2017

MIN: 21-JUNE-2017
MAX: 30-JUNE-2017

Example: Zone maps and sorting

```
SELECT count(*) FROM deep_dive WHERE dt = '06-09-2017';
```

Unsorted table	Sorted by date
MIN: 01-JUNE-2017 MAX: 20-JUNE-2017	MIN: 01-JUNE-2017 MAX: 06-JUNE-2017
MIN: 08-JUNE-2017 MAX: 30-JUNE-2017	MIN: 07-JUNE-2017 MAX: 12-JUNE-2017
MIN: 12-JUNE-2017 MAX: 20-JUNE-2017	MIN: 13-JUNE-2017 MAX: 21-JUNE-2017
MIN: 02-JUNE-2017 MAX: 25-JUNE-2017	MIN: 21-JUNE-2017 MAX: 30-JUNE-2017

Terminology and concepts: Slices

A slice can be thought of as a virtual compute node

Unit of data partitioning

Parallel query processing

Facts about slices

Each compute node is initialized with either 2 or 16 slices

Table rows are distributed to slices

A slice processes only its own data

Data distribution

Distribution style is a table property that dictates how that table's data is distributed throughout the cluster

KEY: Value is hashed, same value goes to same location (slice)

ALL: Full table data goes to the first slice of every node

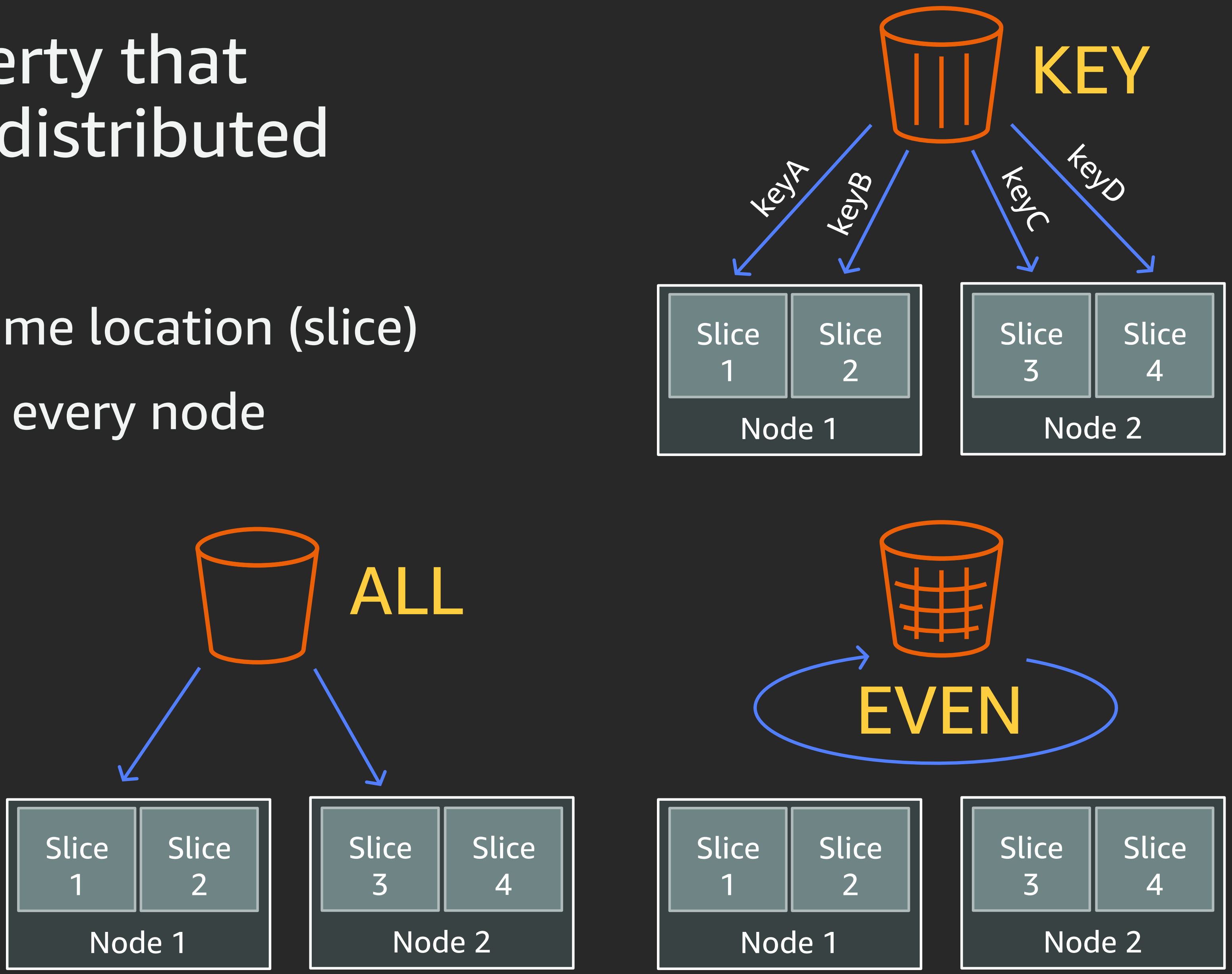
EVEN: Round robin

AUTO: Combines EVEN and ALL

Goals

Distribute data evenly for parallel processing

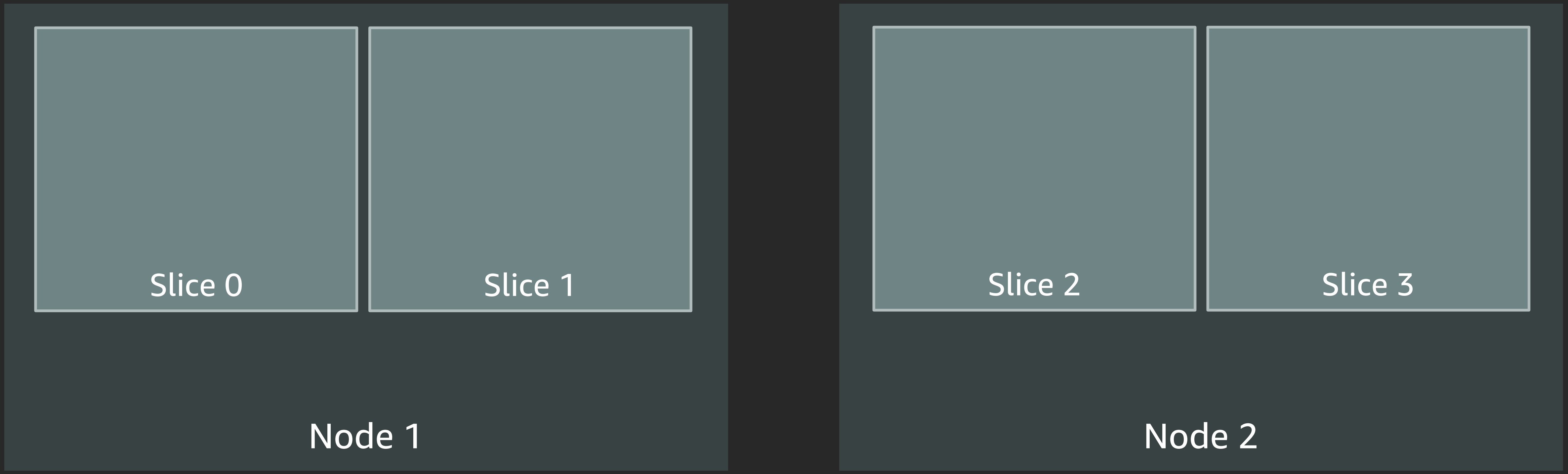
Minimize data movement during query processing



Data distribution example

```
CREATE TABLE deep_dive (
    aid    INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
) (EVEN|KEY|ALL|AUTO);
```

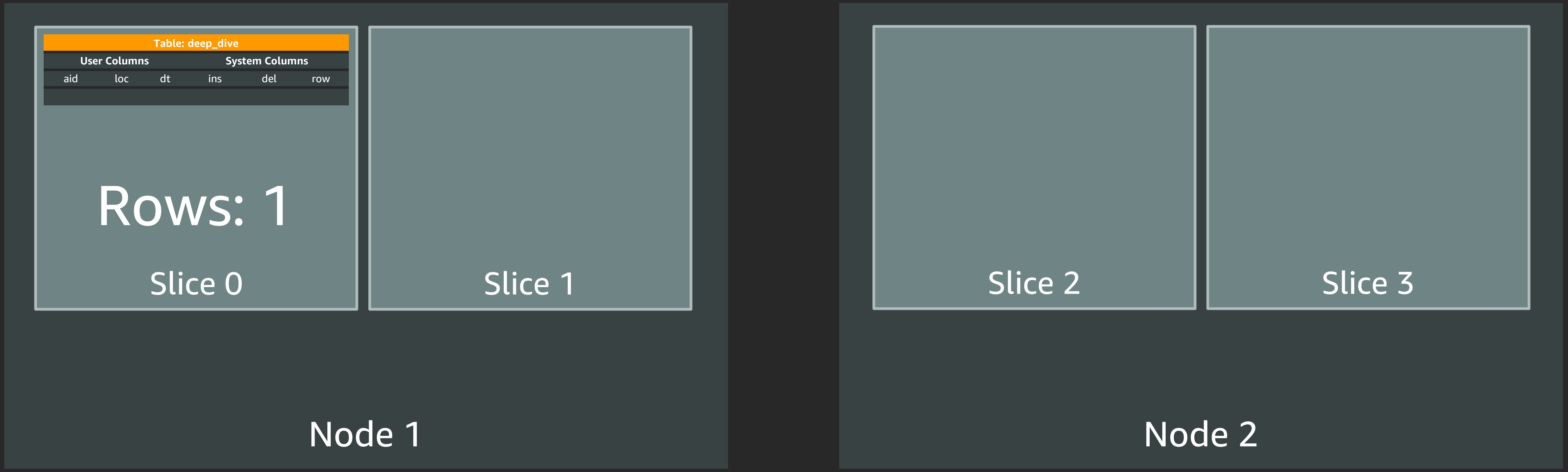
Table: deep_dive					
User columns			System columns		
aid	loc	dt	ins	del	row



Data distribution, EVEN example

```
CREATE TABLE deep_dive (
    aid    INT         --audience_id
    ,loc   CHAR(3)     --location
    ,dt    DATE        --date
) DISTSTYLE EVEN;
```

```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, EVEN example

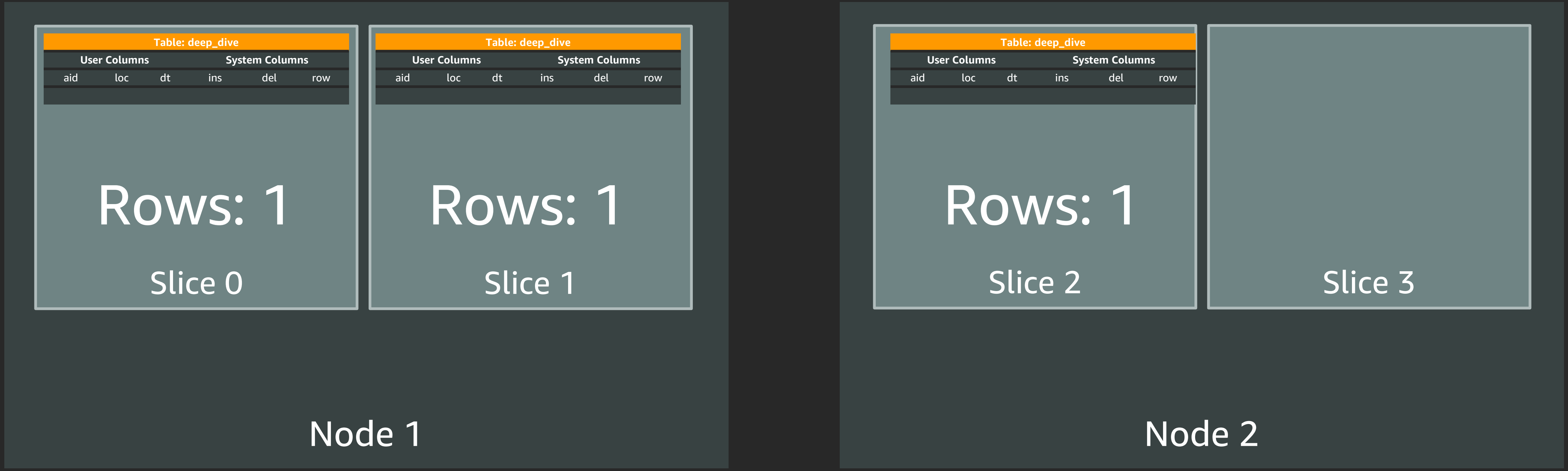
```
CREATE TABLE deep_dive (
    aid   INT        --audience_id
    ,loc  CHAR(3)    --location
    ,dt   DATE       --date
) DISTSTYLE EVEN;
```

```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```

Data distribution, EVEN example

```
CREATE TABLE deep_dive (
    aid    INT          --audience_id
    ,loc   CHAR(3)      --location
    ,dt    DATE         --date
) DISTSTYLE EVEN;
```

```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, EVEN example

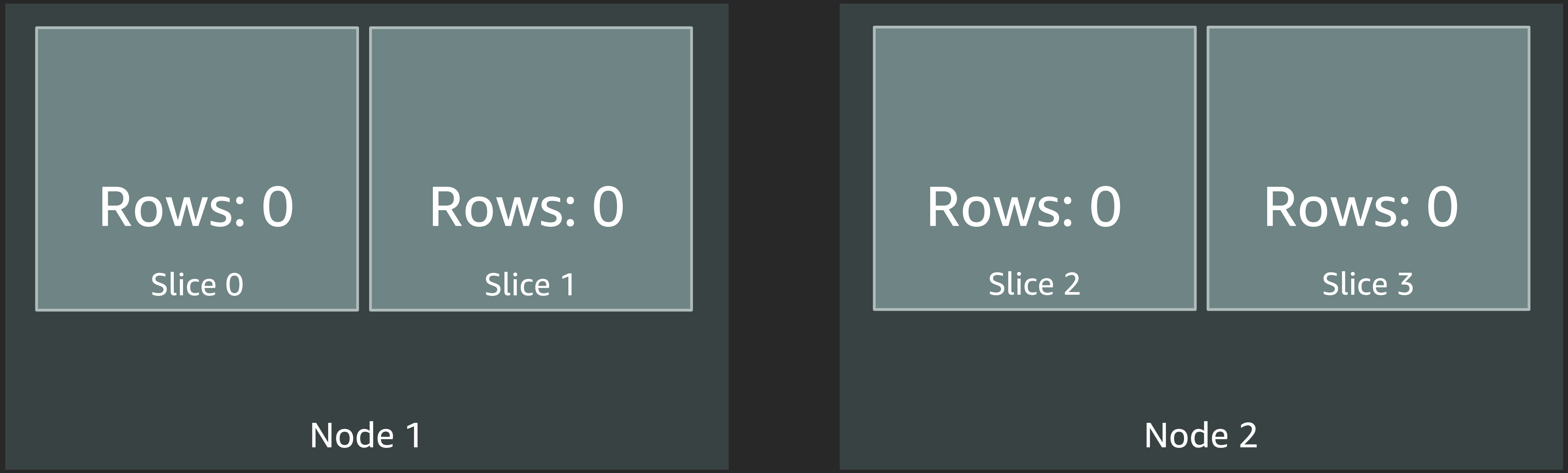
```
CREATE TABLE deep_dive (
    aid    INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
) DISTSTYLE EVEN;
```

```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```

Data distribution, KEY Example #1

```
CREATE TABLE deep_dive (
    aid    INT          --audience_id
    ,loc   CHAR(3)      --location
    ,dt    DATE         --date
) DISTSTYLE KEY DISTKEY (loc);
```

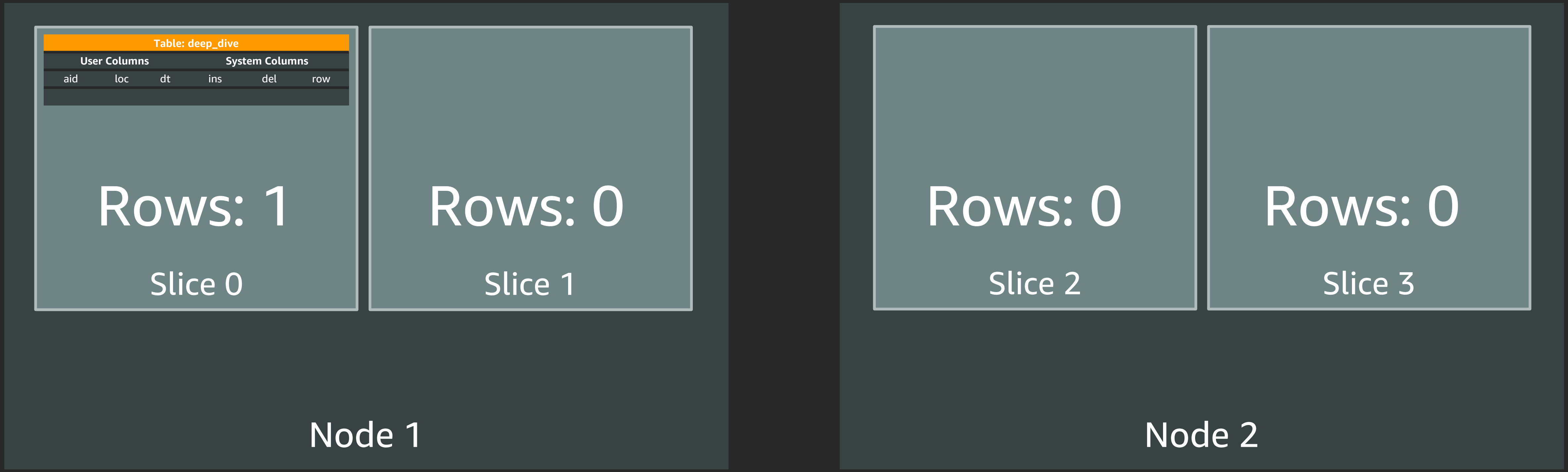
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, KEY Example #1

```
CREATE TABLE deep_dive (
    aid    INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
) DISTSTYLE KEY DISTKEY (loc);
```

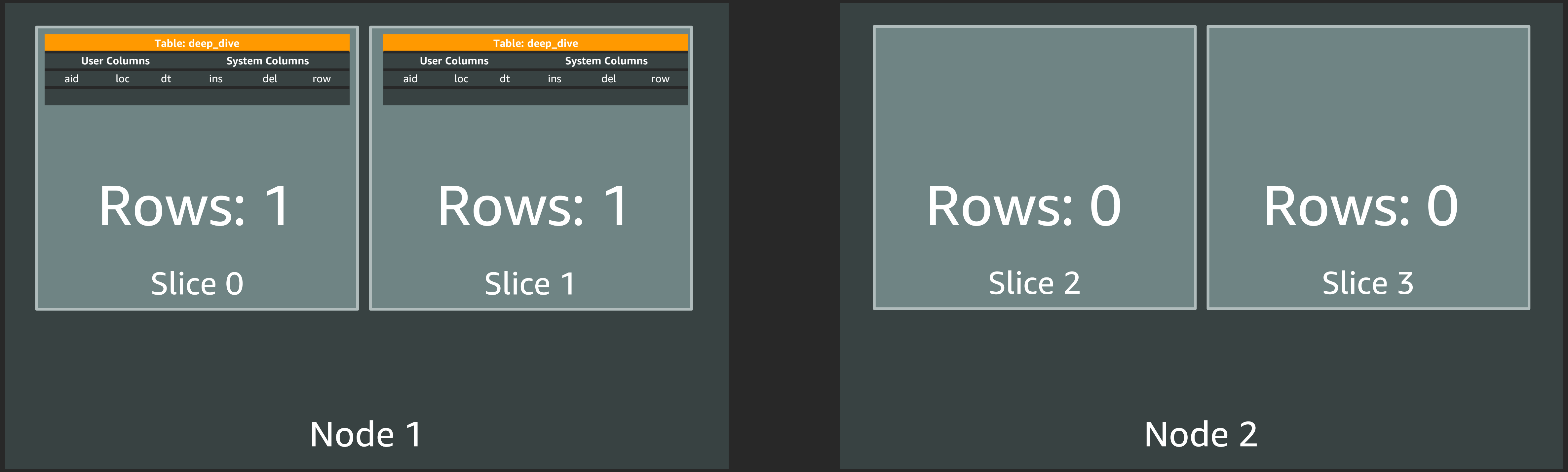
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, KEY Example #1

```
CREATE TABLE deep_dive (
    aid   INT        --audience_id
    ,loc  CHAR(3)    --location
    ,dt   DATE       --date
) DISTSTYLE KEY DISTKEY (loc);
```

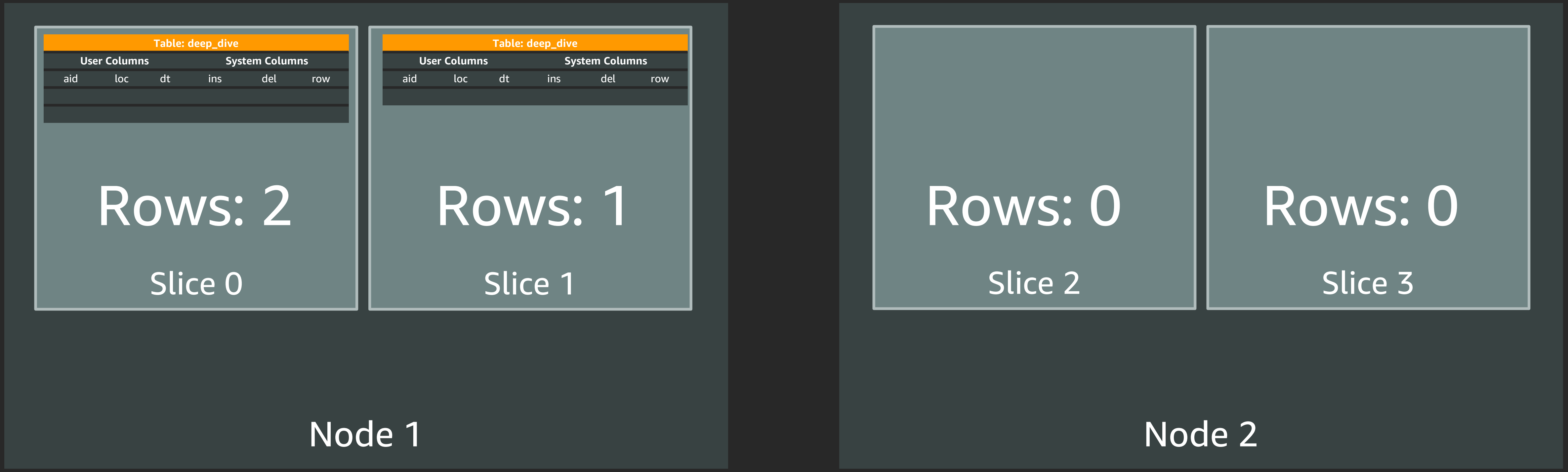
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, KEY Example #1

```
CREATE TABLE deep_dive (
    aid    INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
) DISTSTYLE KEY DISTKEY (loc);
```

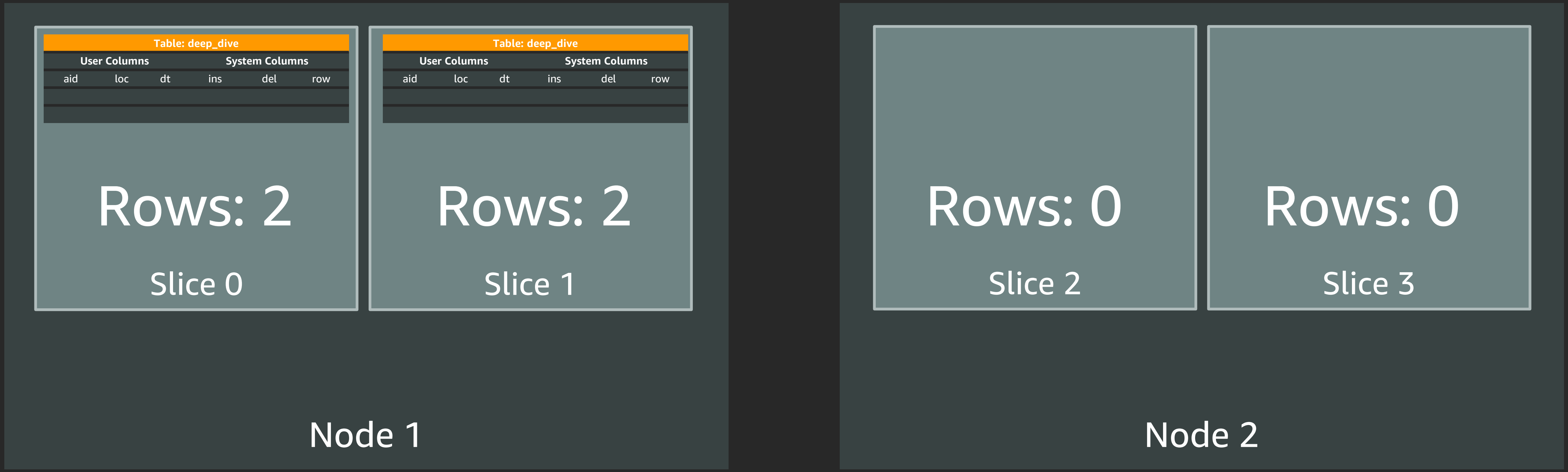
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, KEY Example #1

```
CREATE TABLE deep_dive (
    aid    INT          --audience_id
    ,loc   CHAR(3)      --location
    ,dt    DATE         --date
) DISTSTYLE KEY DISTKEY (loc);
```

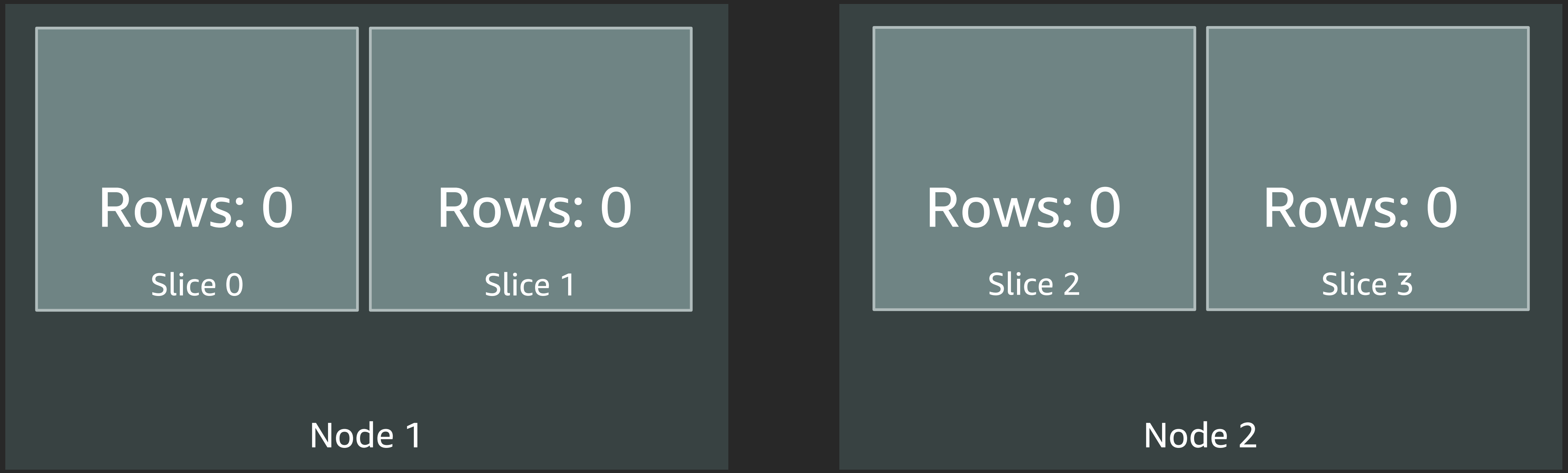
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, KEY Example #2

```
CREATE TABLE deep_dive (
    aid    INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
) DISTSTYLE KEY DISTKEY (aid);
```

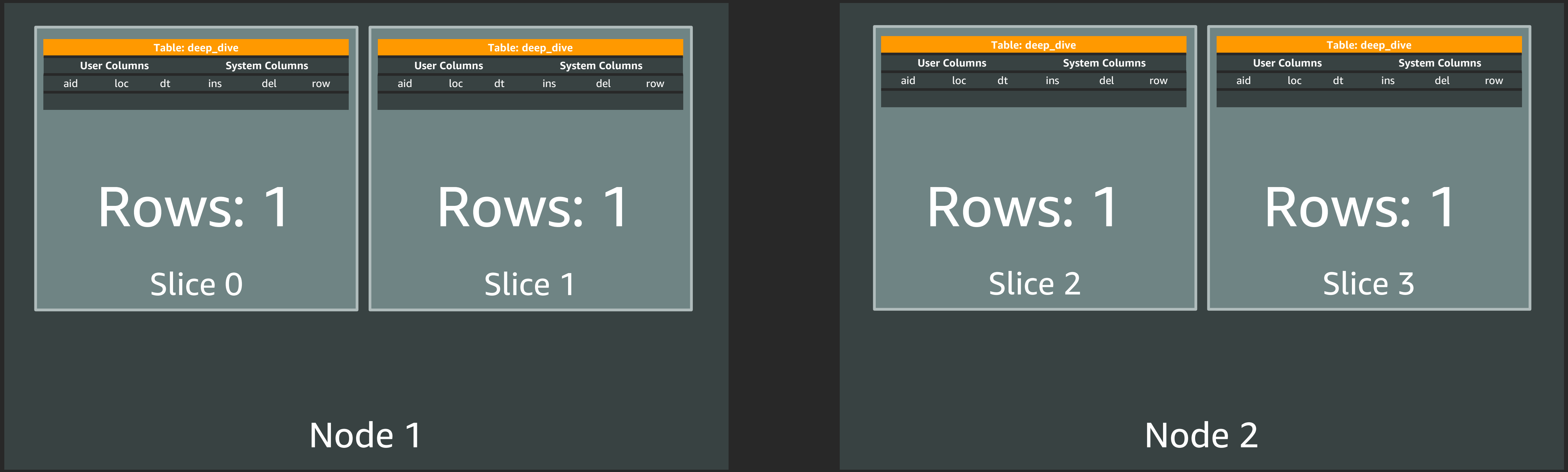
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, KEY Example #2

```
CREATE TABLE deep_dive (
    aid    INT          --audience_id
    ,loc   CHAR(3)      --location
    ,dt    DATE         --date
) DISTSTYLE KEY DISTKEY (aid);
```

```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution, ALL example

```
CREATE TABLE deep_dive (
    aid    INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
) DISTSTYLE ALL;
```

```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```

Data distribution summary

DISTSTYLE KEY

Goals

- Optimize **JOIN** performance between large tables by distributing on columns used in the **ON** clause
- Optimize **INSERT INTO SELECT** performance
- Optimize **GROUP BY** performance

The column that is being distributed on should have a high cardinality and not cause row skew

```
SELECT diststyle, skew_rows
FROM svv_table_info WHERE "table" = 'deep_dive';
 diststyle | skew_rows
-----------+-----------
 KEY(aid)  |      1.07
```

← Ratio of the slice with the most and least number of rows

DISTSTYLE ALL

Goals

- Optimize **JOIN** performance with dimension tables
- Reduces disk usage on small tables

Small- and medium-size dimension tables (<3M rows)

DISTSTYLE EVEN

If neither **KEY** or **ALL** apply

DISTSTYLE AUTO

Default distribution combines DISTSTYLE **ALL** and **EVEN**

Accelerating your data warehouse migration

AWS SUMMIT ONLINE

AWS migration tooling

AWS Schema Conversion Tool (AWS SCT) converts your commercial database and data warehouse schemas to open-source engines or AWS native services, such as Amazon Aurora and Amazon Redshift

---

AWS Database Migration Service (AWS DMS) easily and securely migrates and/or replicates your databases and data warehouses to AWS

AWS SCT

AWS SCT helps automate database schema and code conversion tasks when migrating from source to target database engines

Features

Create assessment reports for homogeneous/heterogeneous migrations

Convert database schema

Convert data warehouse schema

Convert embedded application code

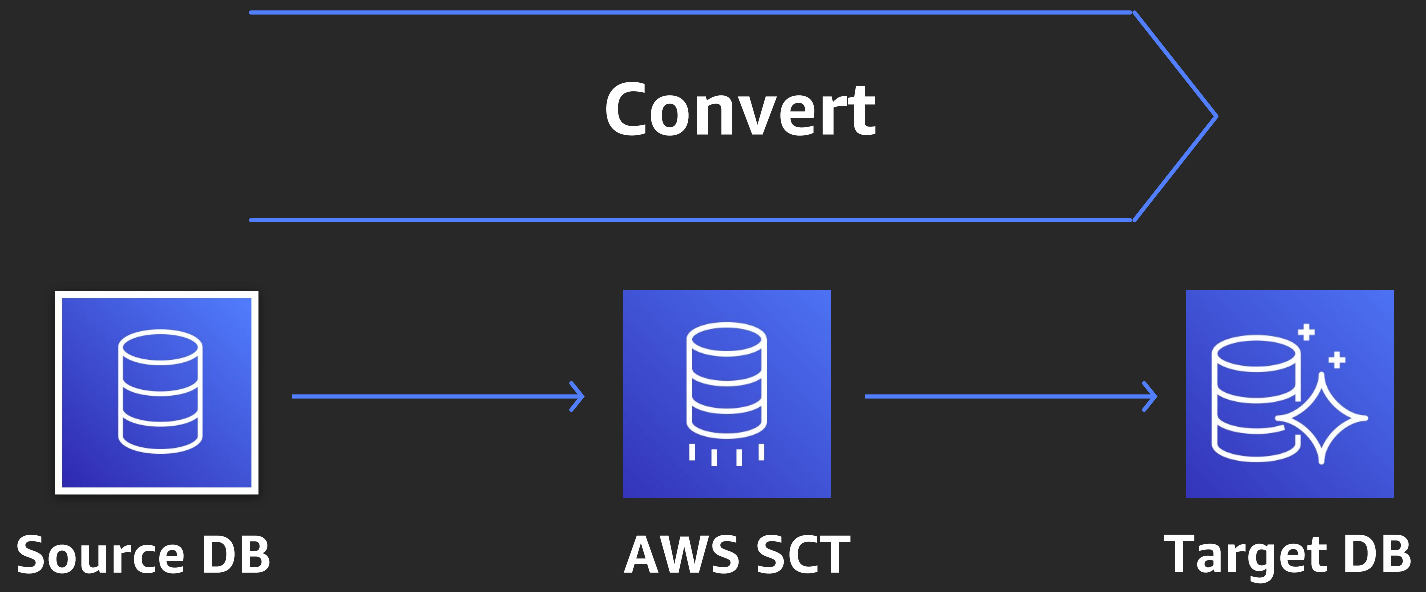
Code browser that highlights places where manual edits are required

Secure connections to your databases with SSL

Service substitutions/ETL modernization to AWS Glue

Migrate data to data warehouses using SCT data extractors

Optimize schemas in Amazon Redshift



AWS SCT data extractors

Extract data from your data warehouse and migrate to Amazon Redshift

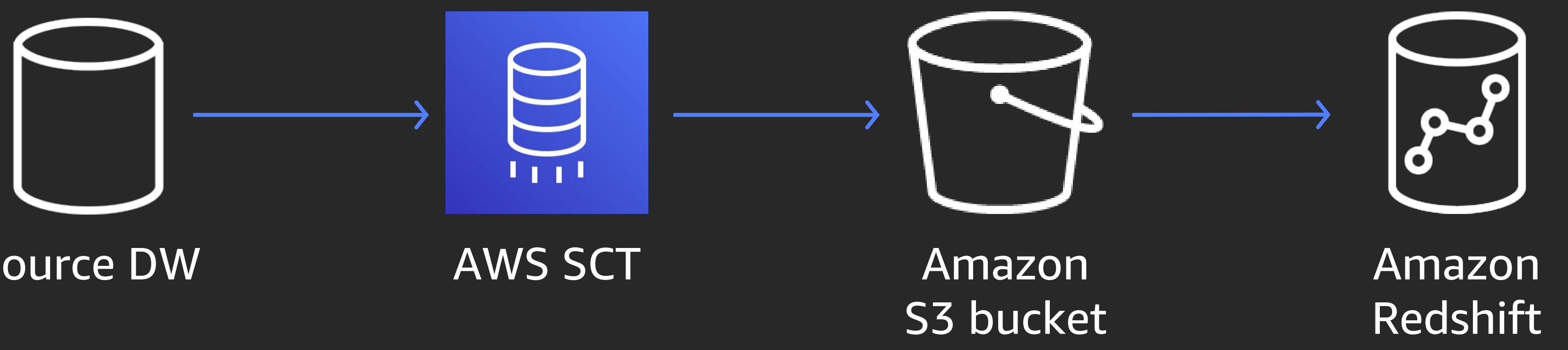
- Extracts data through local migration agents
- Data is optimized for Amazon Redshift and saved in local files
- Files are loaded to an Amazon S3 bucket (through network or AWS Snowball Edge) and then to Amazon Redshift

VERTICA ORACLE

Microsoft SQL Server

Greenplum

teradata.



AWS DMS

Migrating databases to AWS

Migrate between on-premises and AWS

Migrate between databases

Automated schema conversion

Data replication for zero downtime migration

Legacy data warehouse migration tips

When moving from legacy row-based data warehouses

- Denormalize tables where it makes sense (predicate columns in the fact table)
- Avoid date dimension tables
- Amazon Redshift is efficient with wide tables because of columnar storage and compression

When moving from SMP legacy data warehouses

- Colocation of tables is required for fast JOINS
  - Leverage DIST STYLE ALL/AUTO or KEY
- Amazon Redshift is designed for big data (>100 GB to PB scale)
  - Smaller datasets consider Amazon Aurora PostgreSQL
- Wrap workflows in explicit transactions

Leverage Amazon Redshift stored procedures for faster migrations

PL/pgSQL stored procedures were added to make porting legacy procedures easier

Additional resources

aws SUMMIT ONLINE

AWSLabs on GitHub – Amazon Redshift

https://github.com/awslabs/amazon-redshift-utils

https://github.com/awslabs/amazon-redshift-monitoring

https://github.com/awslabs/amazon-redshift-udfs

Admin scripts

Collection of utilities for running diagnostics on your cluster

Admin views

Collection of utilities for managing your cluster, generating schema DDL, and so on

Analyze Vacuum utility

Utility that you can schedule to vacuum and analyze the tables within your Amazon Redshift cluster

Column Encoding utility

Utility that applies optimal column encoding to an established schema with data already loaded

AWS Big Data Blog – Amazon Redshift

Amazon Redshift Engineering's Advanced Table Design Playbook

https://aws.amazon.com/blogs/big-data/amazon-redshift-engineerings-advanced-table-design-playbook-preamble-prerequisites-and-prioritization/

–Zach Christopherson

Top 10 Performance Tuning Techniques for Amazon Redshift

https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-techniques-for-amazon-redshift/

–Ian Meyers and Zach Christopherson

Twelve Best Practices for Amazon Redshift Spectrum

https://aws.amazon.com/blogs/big-data/10-best-practices-for-amazon-redshift-spectrum/

–Po Hong and Peter Dalton

AWS Training and Certification

Training for the whole team

Explore tailored learning paths for customers and partners

Flexibility to learn your way

Build cloud skills with 550+ free digital training courses, or dive deep with classroom training

Validate skills with AWS Certification

Demonstrate expertise with an industry-recognized credential

Education Programs

Find entry-level cloud talent with AWS Academy and AWS re/Start

aws.amazon.com/training

Thank you!

Aneesh Chandra PN
nanhyama@amazon.com