AWS re:Invent

BDM402

Best Practices for Data Warehousing with Amazon Redshift

Eric Ferreira, Principal Engineer, AWS

Philipp Mohr, Sr. CRM Director , King.com

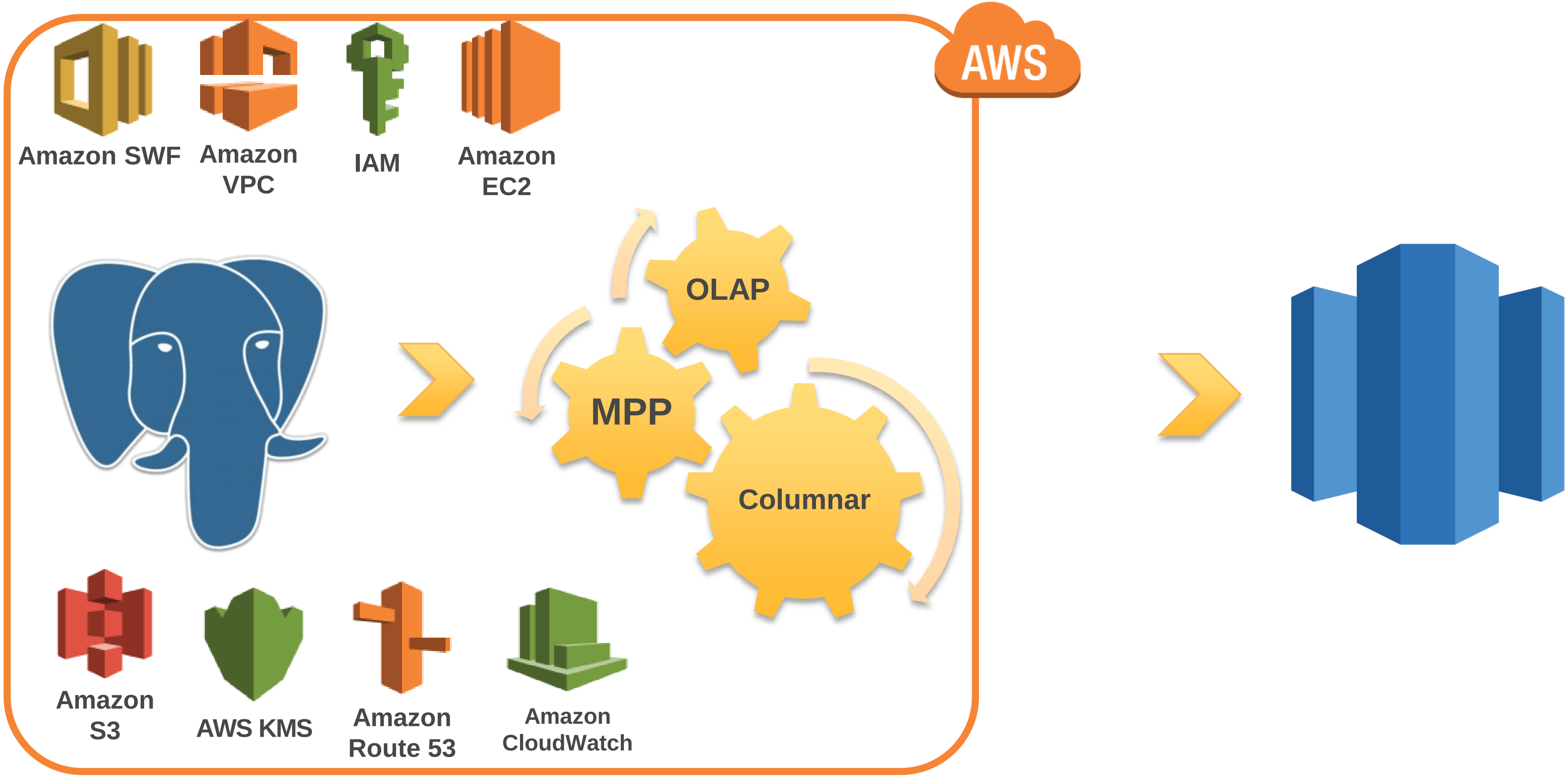
November 29, 2016

What to Expect from the Session

- Brief recap of Amazon Redshift service
- How King implemented their CRM
- Why their best practices work

What is Amazon Redshift ?

- Relational data warehouse
- Massively parallel; petabyte scale
- Fully managed
- HDD and SSD platforms
- $1,000/TB/year; starts at $0.25/hour



AWS

Amazon SWF

Amazon VPC

IAM

Amazon EC2

OLAP

MPP

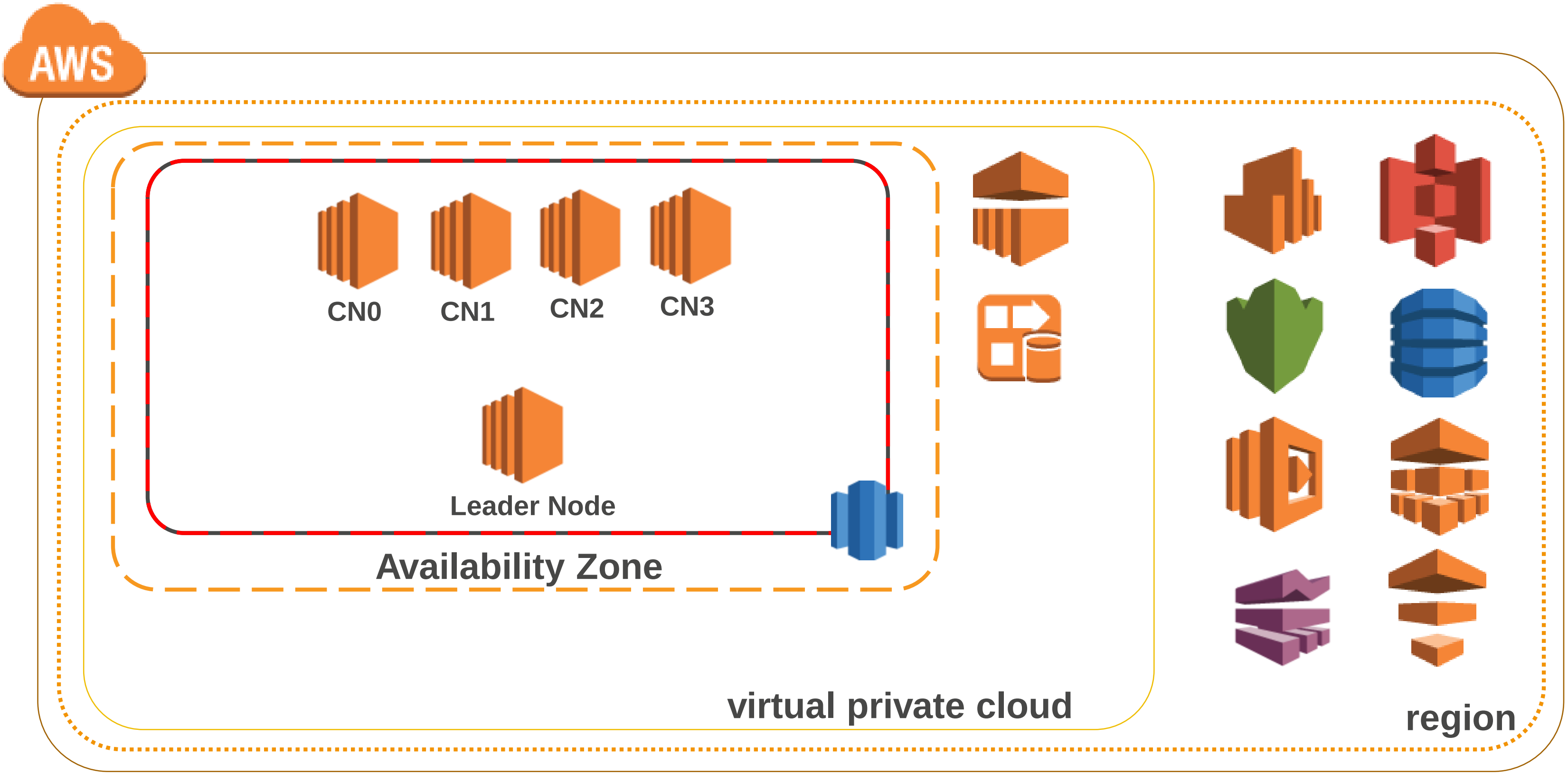
Columnar

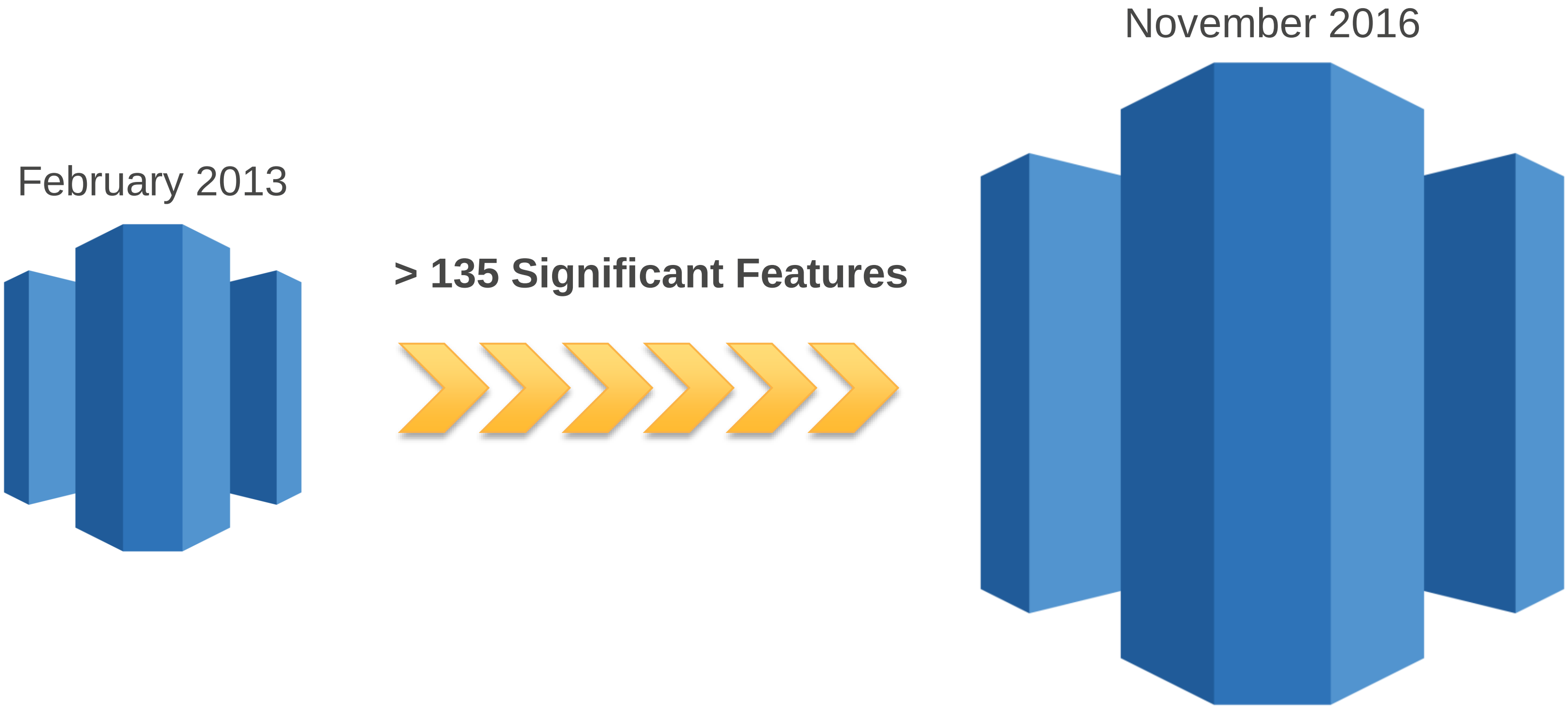
Amazon S3

AWS KMS

Amazon Route 53

Amazon CloudWatch


AWS
CN0
CN1
CN2
CN3
Leader Node
Availability Zone
virtual private cloud
region



February 2013

> 135 Significant Features

November 2016

Are you a user ?

King

Candy Crush
SAGA

Amazon **Redshift**

as an operational CRM database

@

King

Business challenges @ King CRM

Dynamic customer base

Very large scale

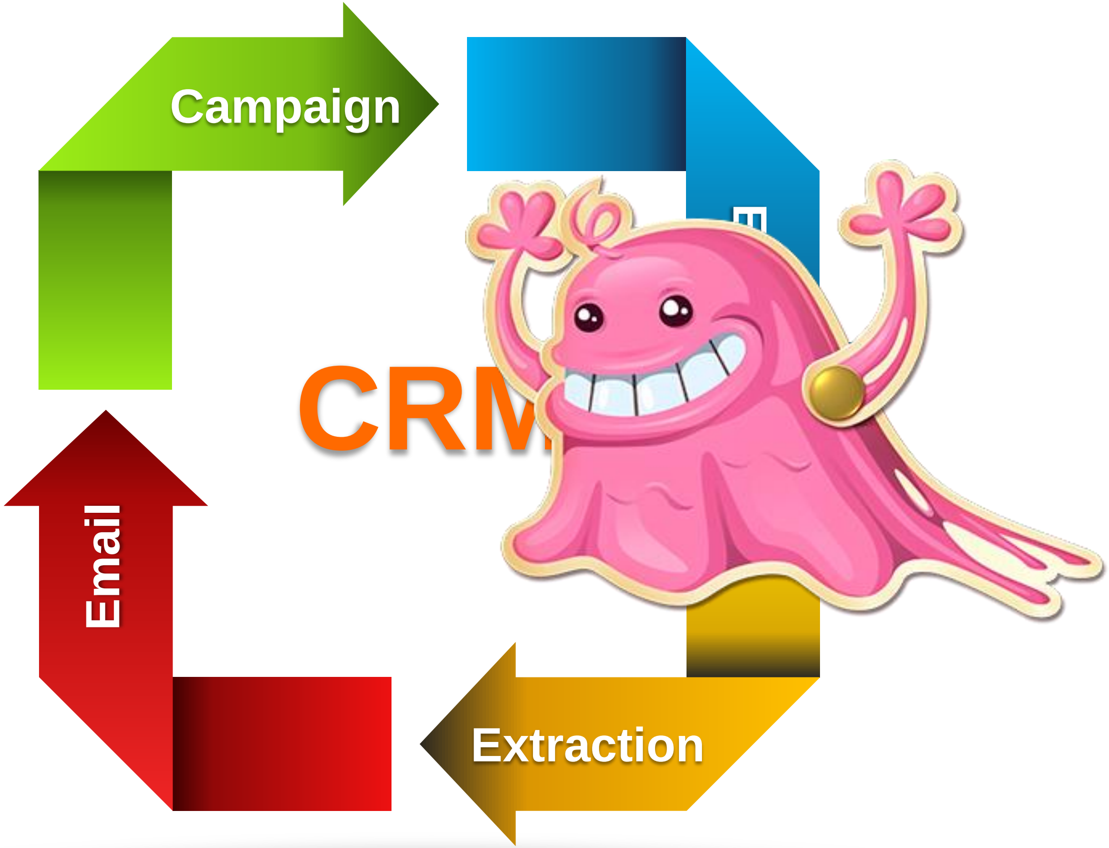
Limited DS resources

Previously in...

CRM

SAGA

The CRM Saga



# The scale we are talking about…

9.5K
campaigns executed / week

1.5B
messages sent / month

12
Games supported

8
promotions specialists

# The scale we are talking about…

9.5K
campaigns
executed / week

1.5B
messages
sent / month

12
Games
supported

8
promotions
specialists

Starting point

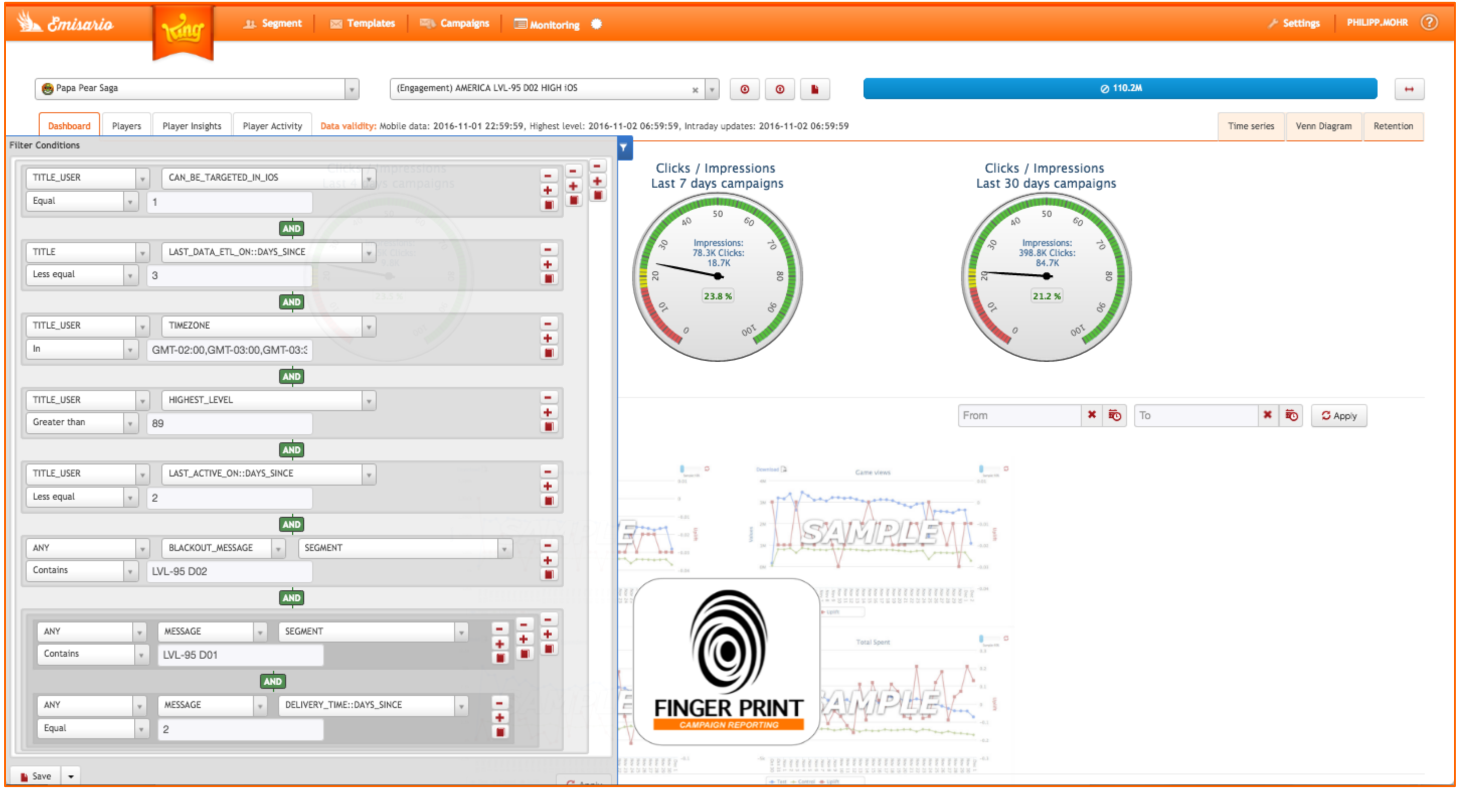
5
campaigns
executed / week

23k
messages
sent / month

5
Game
supported

10
Promotions specialist
DS and Dev support

Emisario: campaign manager



Papa Pear Saga

(Engagement) AMERICA LVL-95 D02 HIGH iOS

Data validity: Mobile data: 2016-11-01 22:59:59, Highest level: 2016-11-02 06:59:59, Intraday updates: 2016-11-02 06:59:59

Filter Conditions

TITLE_USER | CAN_BE_TARGETED_IN_IOS | Equal | 1

AND

TITLE | LAST_DATA_ETL_ON::DAYS_SINCE | Less equal | 3

AND

TITLE_USER | TIMEZONE | In | GMT-02:00,GMT-03:00,GMT-03:

AND

TITLE_USER | HIGHEST_LEVEL | Greater than | 89

AND

TITLE_USER | LAST_ACTIVE_ON::DAYS_SINCE | Less equal | 2

AND

ANY | BLACKOUT_MESSAGE | SEGMENT | Contains | LVL-95 D02

AND

ANY | MESSAGE | SEGMENT | Contains | LVL-95 D01

AND

ANY | MESSAGE | DELIVERY_TIME::DAYS_SINCE | Equal | 2

Why Amazon Redshift?

Quick time to market +

Scalability +

Value +

Bs customer records +

> 10k queries per day +

No dedicated admin team needed +

Peace of mind support +

Column based and massively parallel nature +

Execute analytical / marketing queries quickly +

Why Amazon Redshift?

Quick time to market +

Scalability +

Value +

Bs customer records +

> 10k queries per day +

No dedicated admin team needed +

Peace of mind support +

Column based and massively parallel nature +

Execute analytical / marketing queries quickly +

Performance

Scale of Amazon Redshift clusters

Development

2 x DC1.Large

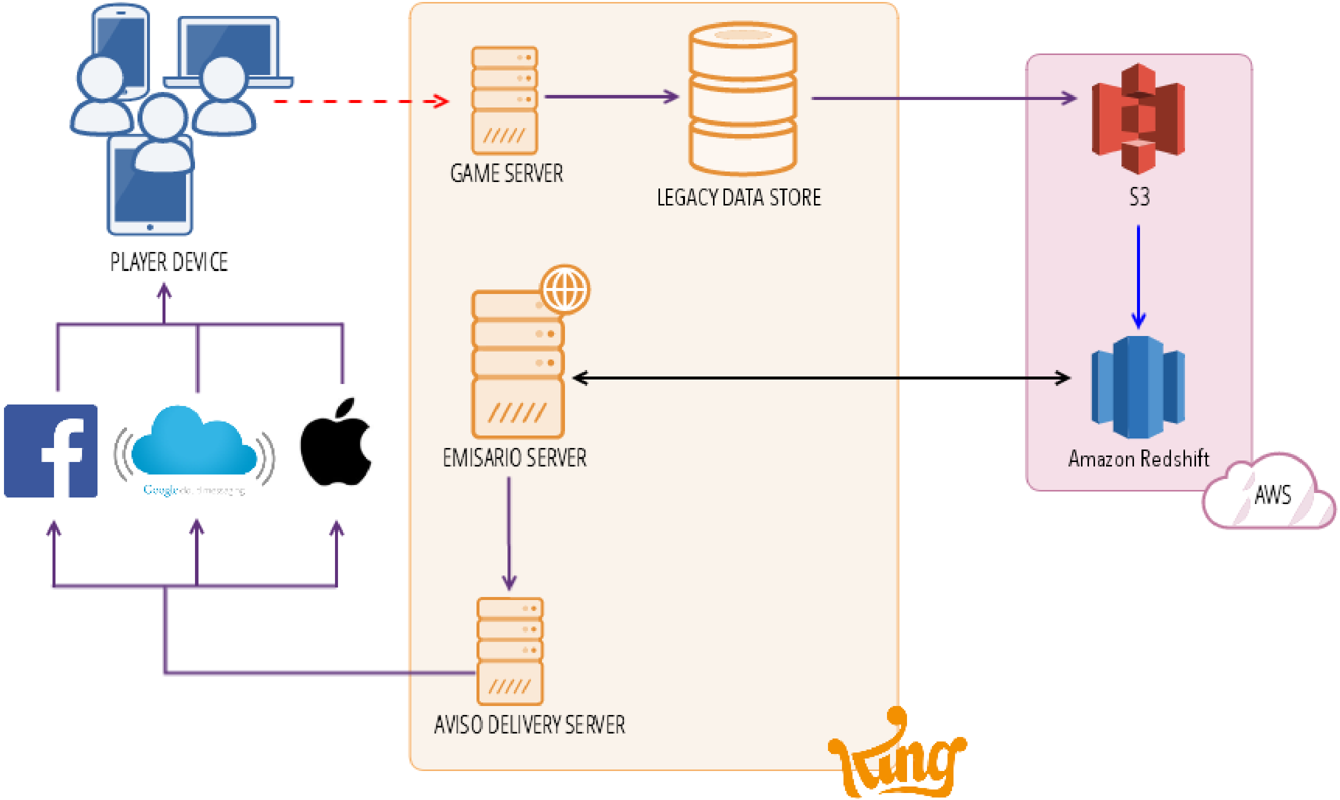
Staging

6 x DC1.Large

Production

24 x DC1.8Xlarge
EC2 Compute Units
(768 virtual cores) 60 TB

Technical architecture



Requirements for Amazon Redshift

- DB needs to be part of an **operational system**

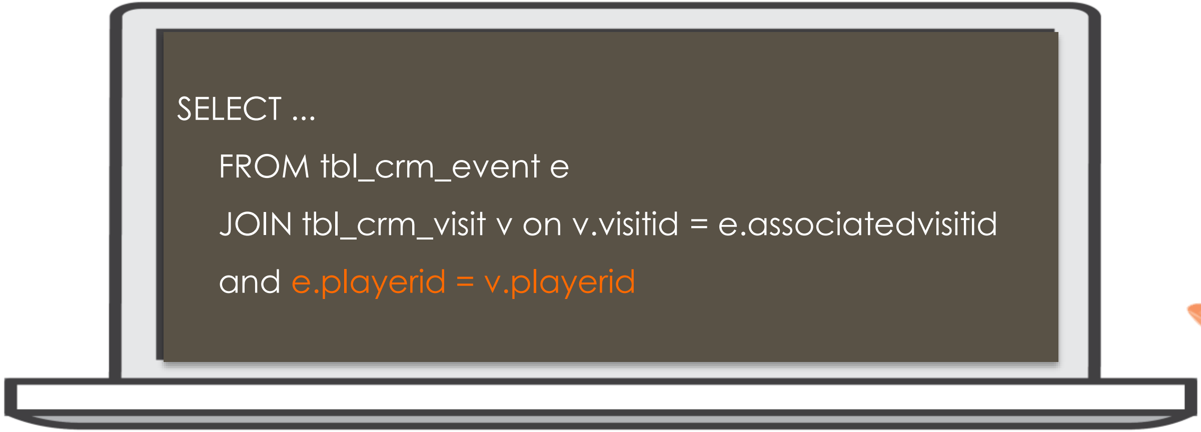
- Must be able to handle **parallel queries** on very large and dynamic data

- Must respond to queries within **15 seconds** in order not to disrupt user experience

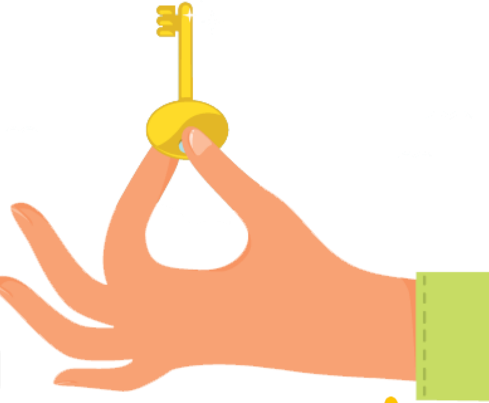
- Must be financially viable 💲

Use of distribution keys in all joins

- Segmentation and data merging queries require joining multiple tables with up to 4 billion rows each
- In such cases, anything other than a **Merge-join** was practically impossible
- Extra join condition added to all queries to join on the Distribution key even when they are semantically redundant
- Dramatic reduction of query times. In certain cases, up to 1,000% increase in performance



```
SELECT ...
    FROM tbl_crm_event e
    JOIN tbl_crm_visit v on v.visitid = e.associatedvisitid
    and e.playerid = v.playerid
```



Migrate to natural distribution keys

PROBLEM

- When scaling from a 100M rows to 3B+, the merge (upsert) process took over 24 hours to complete

SOLUTION

- Restructuring the data and switching to natural distribution keys reduced the average completion time to less than 30 minutes (quite often less than 5 minutes)

WHY

- Merge process can join existing and new data using the common distribution key
- Multiple processing steps of updating primary keys and related foreign keys were no longer necessary
- No operation required data re-distribution
  - update of their values requires moving data between nodes, which is costly

Data pre-processing outside the primary schema tables

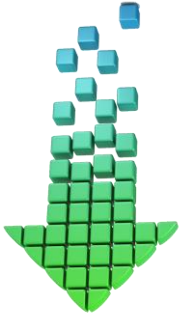
ACTIONS

- Merge (upsert) process performs all pre-processing on temporary tables
- If needed necessary primary tables are used by segmenting (read) queries
  - E.g. final insert/update of the pre-processed data

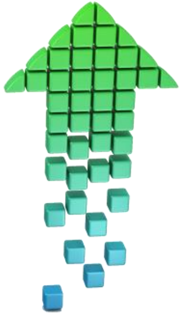
IMPACT

- Segmentation can run in parallel without affecting performance
  - (mostly) they do not access the same tables, and therefore, are not affected by locks

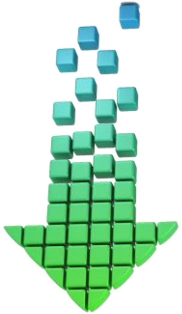
Thanks to column compression encoding…



- Heavy reduction of I/O



- Near 100% performance increase compared to raw uncompressed data



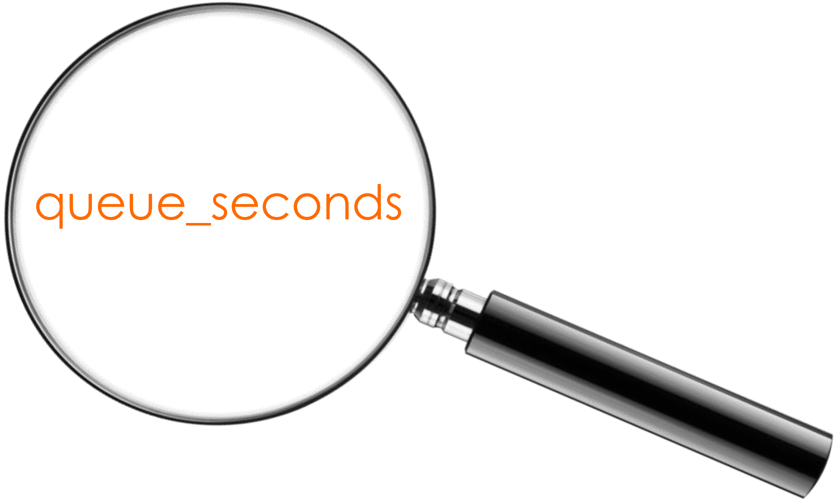
- Cluster size reduced from 48 X DC1.8XLarge to 24 nodes

Use Amazon Redshift column encoding utility to determine best encoding

Concurrency optimizations in WLM

- Amazon Redshift utils from GitHub https://github.com/awslabs/amazon-redshift-utils

```
/* query showing queries which are waiting on a WLM Query Slot */
SELECT w.query
      ,substring(q.querytxt,1,100) AS querytxt
      ,w.queue_start_time
      ,w.service_class AS class
      ,w.slot_count AS slots
      ,w.total_queue_time / 1000000 AS queue_seconds
      ,w.total_exec_time / 1000000 exec_seconds
      ,(w.total_queue_time + w.total_Exec_time) / 1000000 AS total_seconds
FROM stl_wlm_query w
 LEFT JOIN stl_query q
        ON q.query = w.query
       AND q.userid = w.userid
WHERE w.queue_start_Time >= dateadd(day,-7,CURRENT_DATE)
AND   w.total_queue_Time > 0
ORDER BY w.total_queue_time DESC
        ,w.queue_start_time DESC limit 35
```



Concurrency optimizations in WLM contd…

- Workload management (WLM) defines the number of query queues that are available and how queries are routed to those queues for processing

Default configuration

```
["query_concurrency":5,]
```

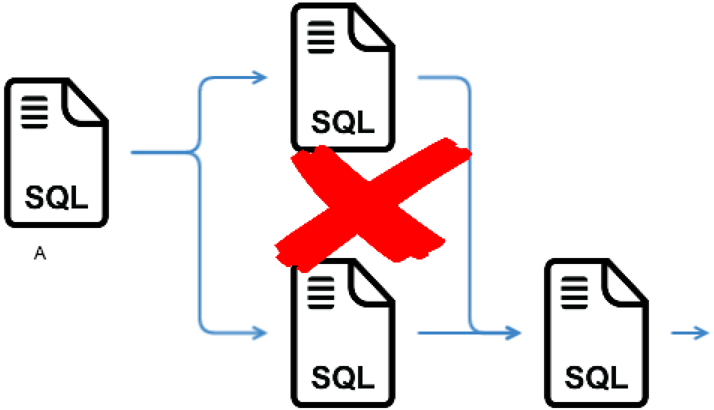
Current configuration

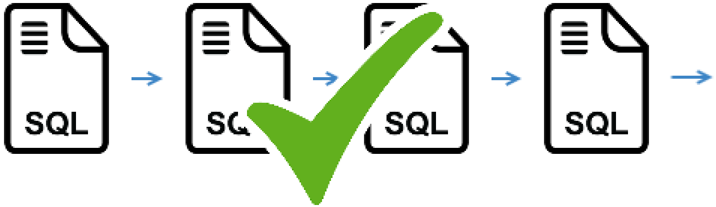
```
["query_concurrency":10,]
```

⚠ ***Extensive tests need to be done to ensure no query runs out of memory*** ⚠

Eliminate concurrent modification of data

- All data upserts are handled by a single process
- No concurrent writes
- Performance of sequential batch queries are better than parallel small queries





On demand vacuum based on the table state

PROBLEM

- Data merge → 10% of data can get updated
- Daily vacuum not sufficient as in 24 hours query performance is severely affected

SOLUTION

- Periodically monitor unsorted regions of tables + vacuum them when it's above threshold X
- Set threshold value per table
- SVV_TABLE_INFO system view used to diagnose and address table design issues that can influence query performance, including issues with compression encoding, distribution keys, sort style, data distribution skew, table size, and statistics.
- Less fluctuations, and therefore, predictable query performance

Reduce the number of selected columns

PROBLEM

- Segmentation queries are automatically generated
- Often requested more columns than necessary for the use-case

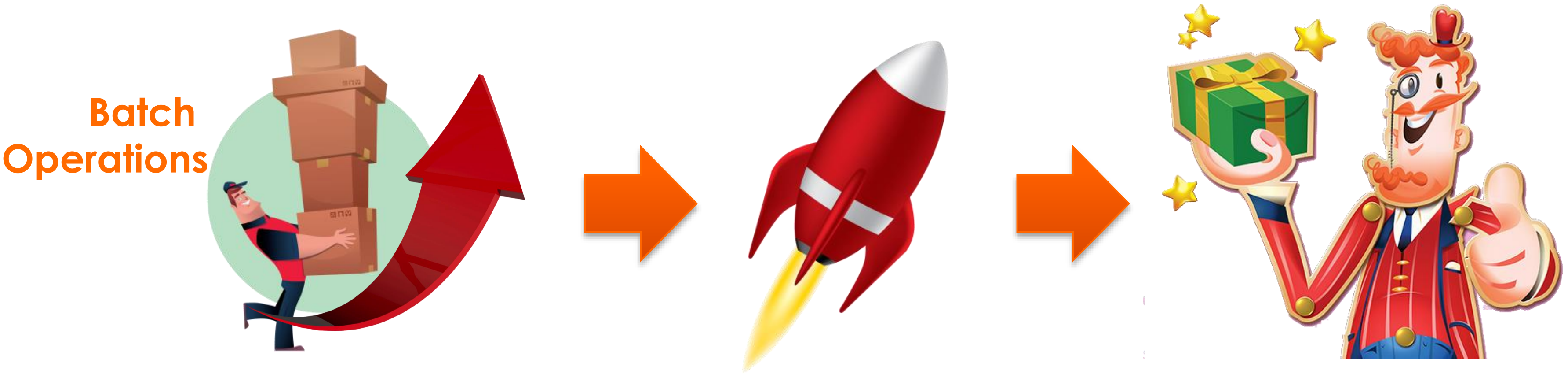
PERFORMANCE IMPACT

- Due to columnar model, extracting extra columns is more expensive compared to OLTP databases

SOLUTION

- Query generation process optimized to select ONLY the columns that are required for a certain use-case.

Increase the batch size as much as possible



- Increased performance: less selects performed
- We operate at 5 million batch size (up from 100K)
  - Upper limit set by memory constraints on operational servers
- But: Balance with data freshness requirements

Reduce use of leader node as much as possible

Problem

- Often, the leader node acts as a bottleneck
  - Extracting a large number of rows (Some segmentation queries return hundreds of millions of rows)
  - Aggregate calculations across distribution keys

Solution

- Ensure data is unloaded to S3 (or other AWS channels) which the individual nodes can communicate directly with
- Modify, if possible, queries to NOT span distribution keys. Each calculation can be performed in each node

Technical recommendations

- Use distribution keys that can be used in all joins
- Migrate to natural keys
- Reduce use of leader-node as much as possible
- Column compression encoding

- Data pre-processing outside the main tables
- WLM optimizations
- Increase batch-size as much as possible

- Prohibit concurrent modification of data
- Reduce selected columns
- On-demand vacuum based on the state of the database

King

Our vision:
Fast, Cheap and Easy-to-use

Think: Toaster

You submit your job

Choose a few options

It runs

Amazon Redshift Cluster Architecture

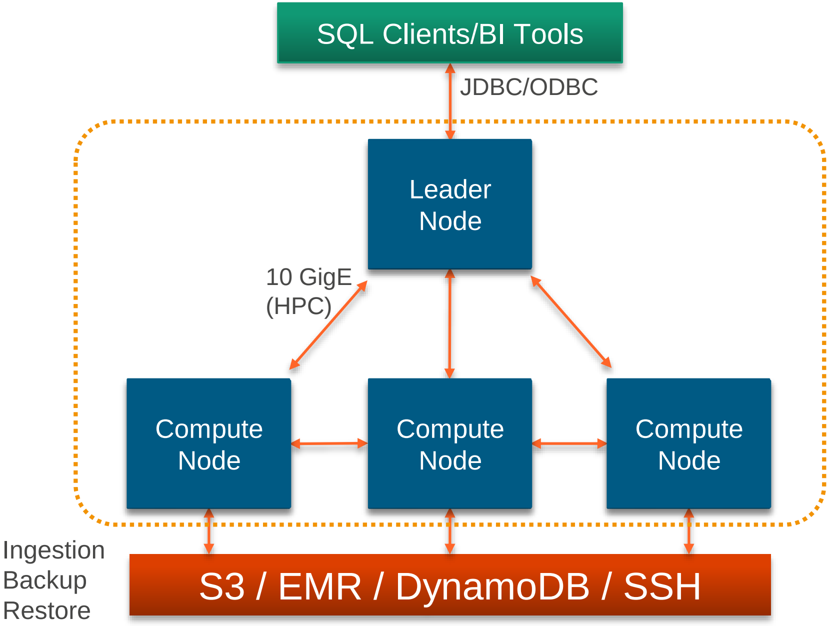
Massively parallel, shared nothing

Leader node

- SQL endpoint
- Stores metadata
- Coordinates parallel SQL processing

Compute nodes

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

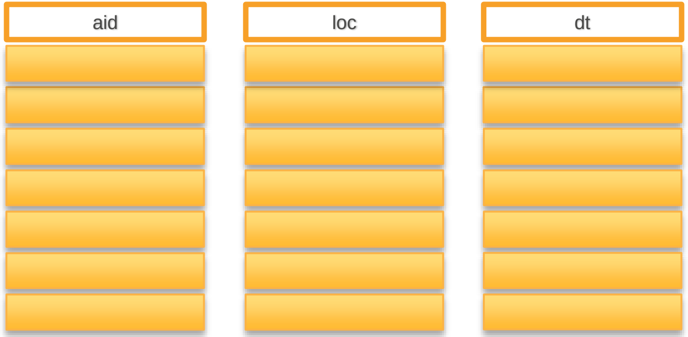


Designed for I/O Reduction

Columnar storage

Data compression

Zone maps



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

aid	loc	dt
1	SFO	2016-09-01
2	JFK	2016-09-14
3	SFO	2017-04-01
4	JFK	2017-05-14

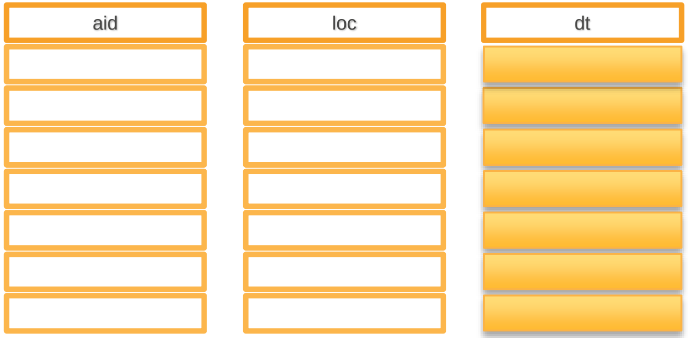
- Accessing dt with row storage:
  - Need to read everything
  - Unnecessary I/O

Designed for I/O Reduction

Columnar storage

Data compression

Zone maps



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

aid	loc	dt
1	SFO	2016-09-01
2	JFK	2016-09-14
3	SFO	2017-04-01
4	JFK	2017-05-14

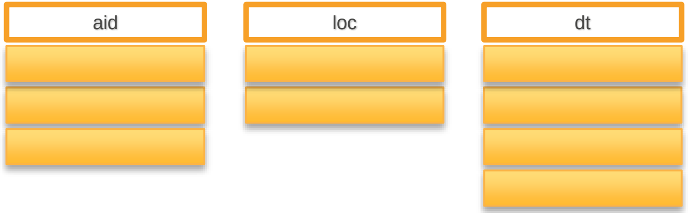
- Accessing dt with columnar storage:
  - Only scan blocks for relevant column

Designed for I/O Reduction

Columnar storage

Data compression

Zone maps



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

aid	loc	dt
1	SFO	2016-09-01
2	JFK	2016-09-14
3	SFO	2017-04-01
4	JFK	2017-05-14

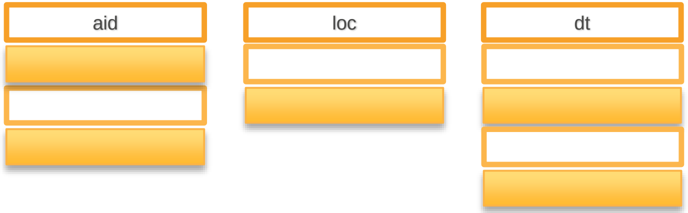
- Columns grow and shrink independently
- Effective compression ratios due to like data
- Reduces storage requirements
- Reduces I/O

Designed for I/O Reduction

Columnar storage

Data compression

Zone maps



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

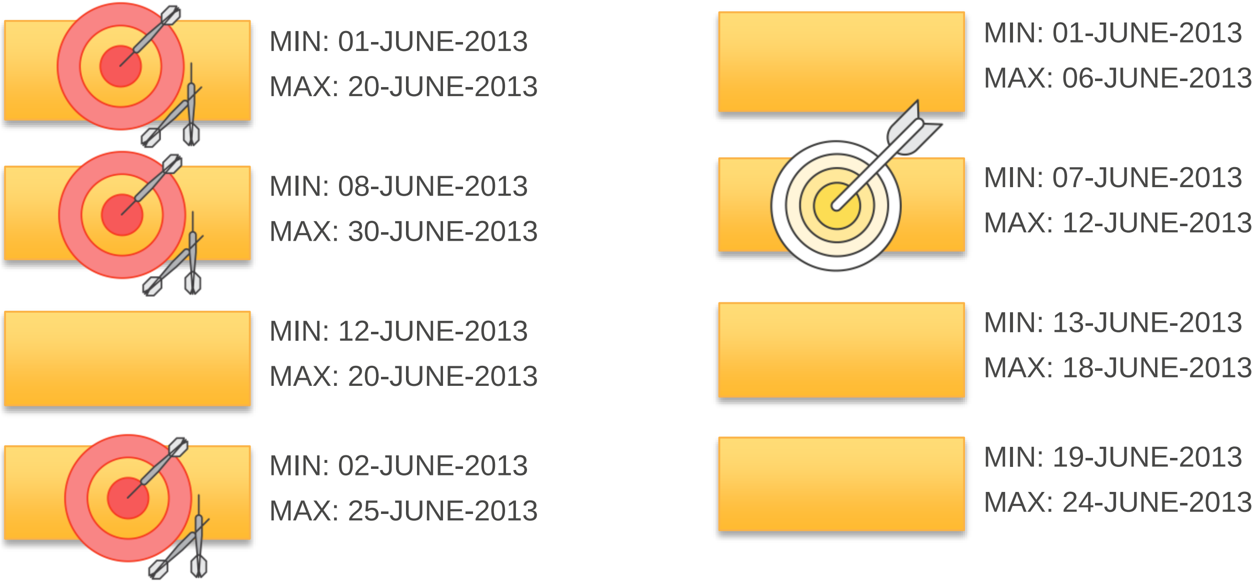
aid	loc	dt
1	SFO	2016-09-01
2	JFK	2016-09-14
3	SFO	2017-04-01
4	JFK	2017-05-14

- In-memory block metadata
- Contains per-block MIN and MAX value
- Effectively prunes blocks which cannot contain data for a given query
- Eliminates unnecessary I/O

Sort Keys

Zone Maps

SELECT COUNT(*) FROM reinvent_deep_dive WHERE DT = ‘09-JUNE-2013’



Unsorted Table

MIN: 01-JUNE-2013
MAX: 20-JUNE-2013

MIN: 08-JUNE-2013
MAX: 30-JUNE-2013

MIN: 12-JUNE-2013
MAX: 20-JUNE-2013

MIN: 02-JUNE-2013
MAX: 25-JUNE-2013

Sorted By Date

MIN: 01-JUNE-2013
MAX: 06-JUNE-2013

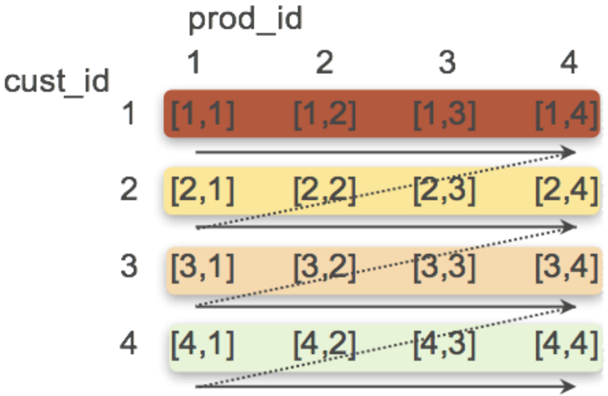
MIN: 07-JUNE-2013
MAX: 12-JUNE-2013

MIN: 13-JUNE-2013
MAX: 18-JUNE-2013

MIN: 19-JUNE-2013
MAX: 24-JUNE-2013

Compound Sort Keys

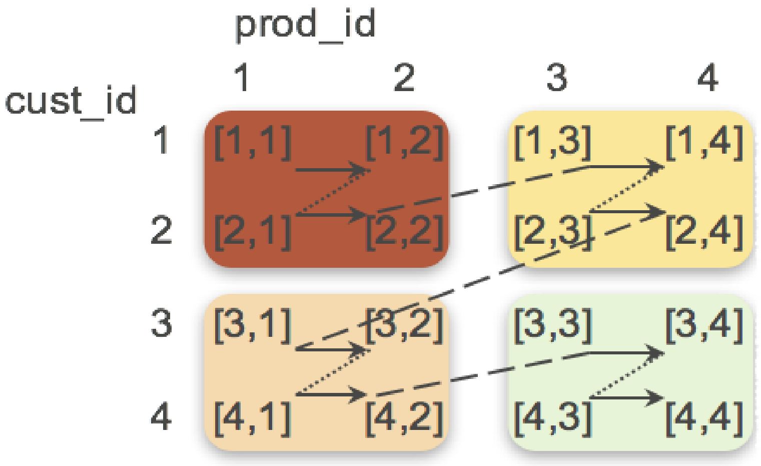
- Records in Amazon Redshift are stored in blocks
- For this illustration, let's assume that four records fill a block
- Records with a given cust_id are all in one block
- However, records with a given prod_id are spread across four blocks



	1	2	3	4
1	[1,1]	[1,2]	[1,3]	[1,4]
2	[2,1]	[2,2]	[2,3]	[2,4]
3	[3,1]	[3,2]	[3,3]	[3,4]
4	[4,1]	[4,2]	[4,3]	[4,4]

Interleaved Sort Keys

- Column values mapped in buckets and bits interleaved (order is maintained)
- Data is sorted in equal measures for both keys
- New values get assigned “others” bucket
- User has to re-map and re-write the whole table to incorporate new mappings
- Records with a given cust_id are spread across two blocks
- Records with a given prod_id are also spread across two blocks



cust_id \ prod_id	1	2	3	4
1	[1,1]	[1,2]	[1,3]	[1,4]
2	[2,1]	[2,2]	[2,3]	[2,4]
3	[3,1]	[3,2]	[3,3]	[3,4]
4	[4,1]	[4,2]	[4,3]	[4,4]

Interleaved Sort Key - Limitations

- Only makes sense on very large tables

Blocks Scanned

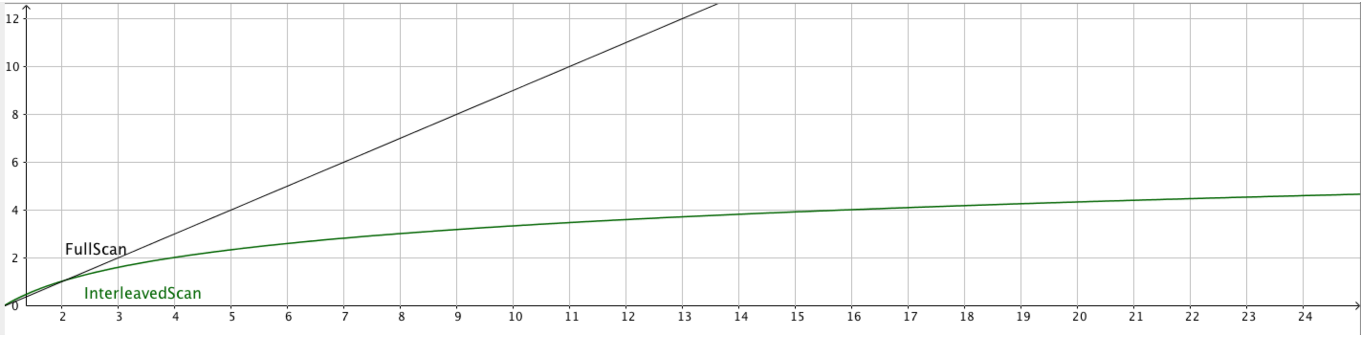


Table Size: Blocks per column per slice

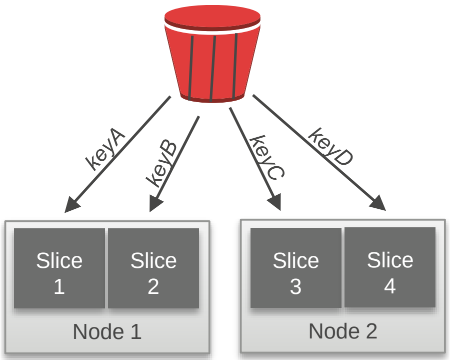
- Columns domain should be stable

Data Distribution

- Distribute data evenly for parallel processing
- Minimize data movement
  - Co-located joins
  - Localized aggregations

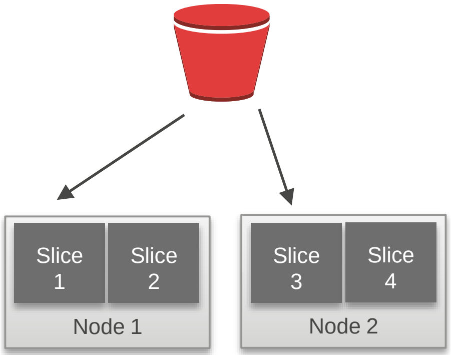
Distribution key

Same key to same location



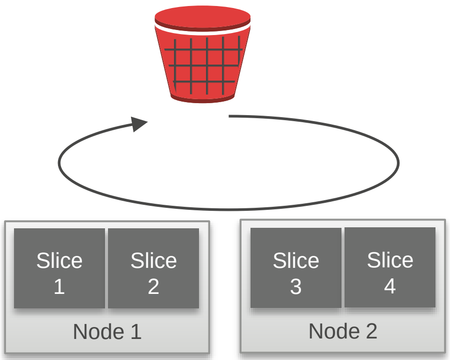
All

Full table data on first slice of every node



Even

Round robin distribution



We help you migrate your database…

AWS Schema Conversion Tool

Current Sources:

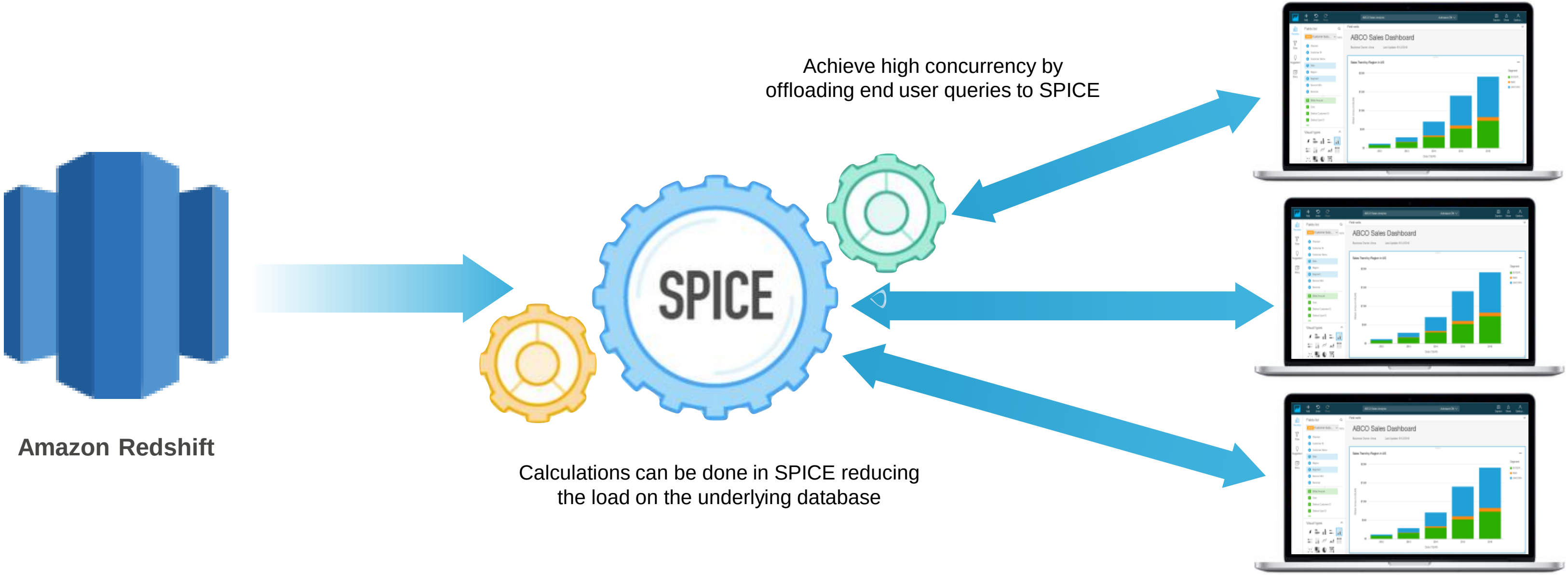
- Oracle
- Teradata
- Netezza
- Greenplum
- **Redshift**

Data migration available though partners today…

- Schema optimized for Amazon Redshift
- Convert SQL inside your code

QuickSight + Redshift

Redshift is one the fastest growing services in the AWS platform. QuickSight seamlessly connects to Redshift giving you native access to all of your instances, and tables.



Achieve high concurrency by offloading end user queries to SPICE

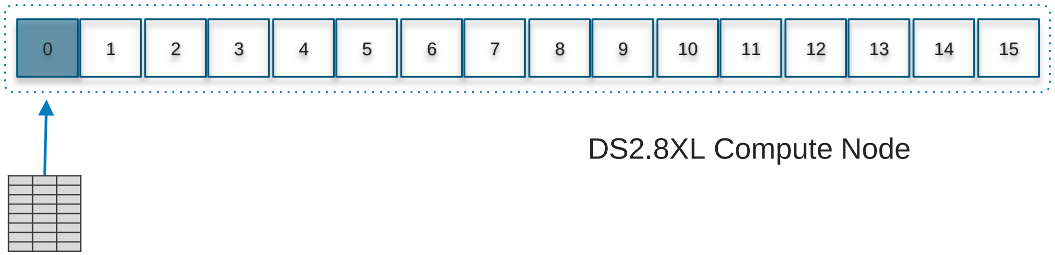
Amazon Redshift

Calculations can be done in SPICE reducing the load on the underlying database

Parallelism considerations with Amazon Redshift slices

Ingestion Throughput:

- Each slice's query processors can load one file at a time:
  - Streaming decompression
  - Parse
  - Distribute
  - Write



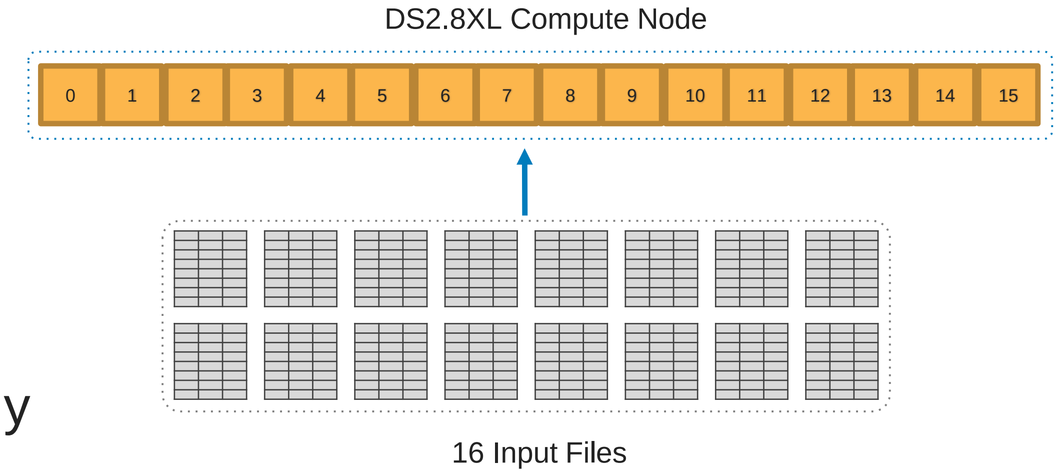
Realizing only partial node usage as 6.25% of slices are active

Design considerations for Amazon Redshift slices

Use at least as many input files as there are slices in the cluster

With 16 input files, all slices are working so you maximize throughput

COPY continues to scale linearly as you add nodes

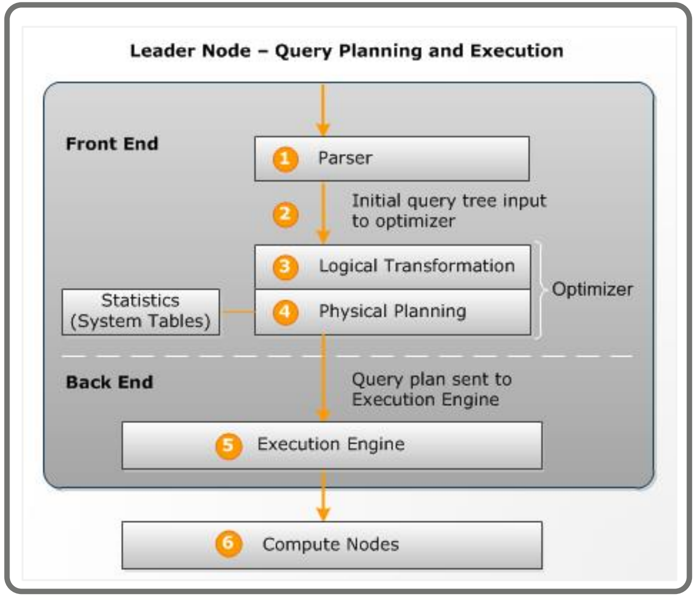


Optimizing a database for querying

- Periodically check your table status
- Vacuum and analyze regularly
    - SVV_TABLE_INFO
    - Missing statistics
    - Table skew
    - Uncompressed columns
    - Unsorted data
- Check your cluster status
    - WLM queuing
    - Commit queuing
    - Database locks

Missing statistics

- Amazon Redshift query optimizer relies on up-to-date statistics
- Statistics are necessary only for data that you are accessing
- Updated stats important on:
  - SORTKEY
  - DISTKEY
  - Columns in query predicates



Leader Node – Query Planning and Execution
Front End
1 Parser
2 Initial query tree input to optimizer
3 Logical Transformation
Statistics (System Tables)
4 Physical Planning
Optimizer
Back End
Query plan sent to Execution Engine
5 Execution Engine
6 Compute Nodes

Table maintenance and status

Table skew

- Unbalanced workload
- Query completes as fast as the slowest slice completes
- Can cause skew inflight:
    - Temp data fills a single node, resulting in query failure

Unsorted table

- Sortkey is just a guide, but data actually needs to be sorted
- VACUUM or DEEP COPY to sort
- Scans against unsorted tables continue to benefit from zone maps:
    - Load sequential blocks

Cluster status: commits and WLM

WLM queue

Identify short/long-running queries and prioritize them

Define multiple queues to route queries appropriately

Default concurrency of 5

Leverage wlm_apex_hourly to tune WLM based on peak concurrency requirements

Commit queue

How long is your commit queue?

- Identify needless transactions
- Group dependent statements within a single transaction
- Offload operational workloads
- STL_COMMIT_STATS

Open source tools

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, etc.

ColumnEncodingUtility

Gives you the ability to apply optimal column encoding to an established schema with data already loaded

What’s next ?

Don’t Miss…

BDA304 - What’s New with Amazon Redshift

DAT202-R - [REPEAT] Migrating Your Data Warehouse to Amazon Redshift

BDA203 - Billions of Rows Transformed in Record Time Using Matillion ETL for Amazon Redshift

BDM306-R - [REPEAT] Netflix: Using Amazon S3 as the fabric of our big data ecosystem

Oster
Frozen
Cancel
Bagel

AWS re:Invent

Thank you!

amazon web services

Remember to complete your evaluations!