Choices for Implementing a Data Warehouse:

DWH

- Cloud:
  - Low barrier to entry
  - Easy add on requirements after scalability
  - Scalability & elasticity out of the box.
  - Cloud Managed
    Amazon S3, Amazon RDS
    Know expertise, less IT Staff
  - Self-Managed
    EC2 + PostGre, EC2 + Cassandra
    Total control of the infrastructure
- On-Premise:
  - Heterogeneity, scalability, elasticity of the tools
  - New IT staff skills & multiple locations
  - Cost of ownership

Amazon Redshift:

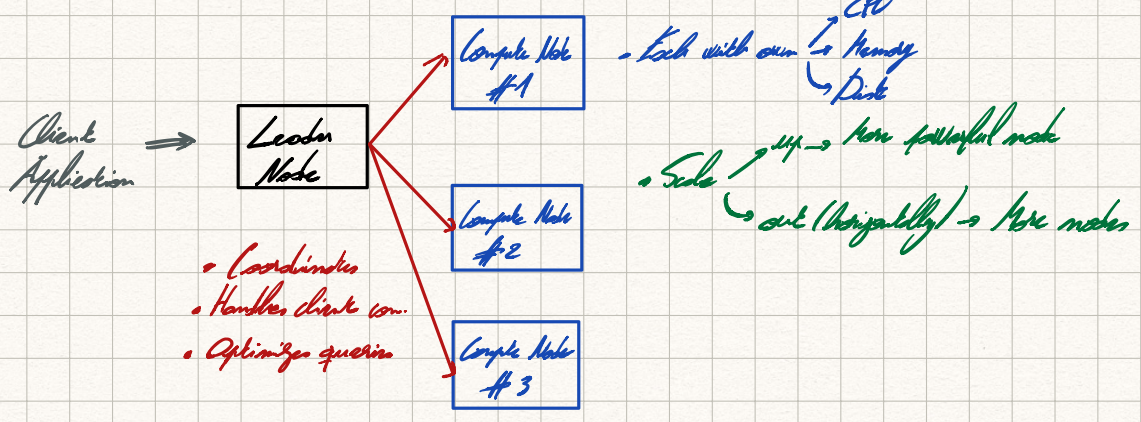
Relational Databases:

- Every query is always executed on a single CPU of a single machine
- Acceptable for OLTP, mostly updates and few and not retrieved.

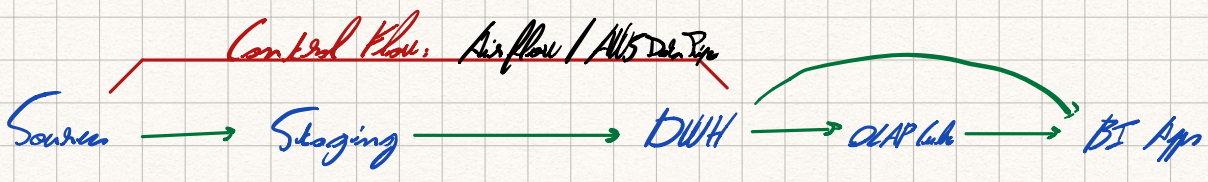
Massive Parallel Processing (MPP) databases

- Parallelize the execution of one query on multiple CPU/machines.
- Table partition (distribution across node 'cluster') becomes critical
  - Partitions are processed in parallel.

Architecture:



Redshift & ETL Context:



Optimizing Table Design:

- When a table is partitioned up into many pieces and distributed across slices in different machines, this is done blindly.
- If one has an idea about the frequent access pattern of a table, one can choose a more clever strategy
  - Two possible strategies: a Distribution style
    - Sorting style

Distribution Style:

Types: EVEN distribution
ALL distribution
AUTO distribution
KEY distribution

→ sliced → slices evenly distributed.

EVEN distribution:
- Round-robin over all slices to achieve load balancing
- Good if tables won't be joined → High cost of shuffling operations

ALL distribution:
- Small tables could be replicated on all slices to speed up joins
- Used frequently for dimensional tables
- Also "Broadcasting" → Prevents "massive shuffling" of dimensional tables while joining.

AUTO distribution: → Leave decision to REDshift

Table Size ↑
- → "Small Enough" tables are distributed with an "ALL Strategy"
- → Large tables are distributed with "EVEN Strategy"

KEY distribution: → Rows with similar values are placed in the same slice.

↓ This can lead to a skewed distribution if some values of the dist key are more frequent than others

↓ Very useful when a dimension table is too big to be distributed with ALL strategy. In that case, we distribute both the fact table and the dimension table using the same dist key

If 2 tables are distributed on the joining keys, redshift collocates the rows from both tables on the same slices.

Sorting Key:

- One can define columns as sort key
- Upon loading, rows are sorted before distribution to slices
- Minimizes the query time since each node already has contiguous ranges of rows based on the sorting key
- Useful for columns that are used frequently in sorting like the date dimension and its corresponding foreign key in the fact table.