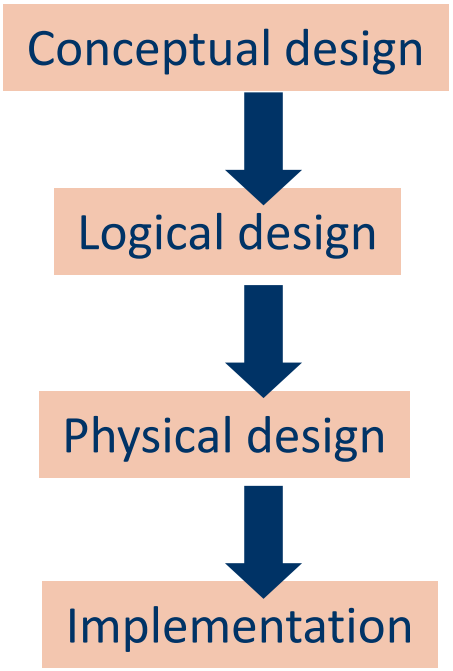
Business intelligence

Unit 2 – Datawarehouse and OLAP
S2-4 – Physical design

DW design



- Efficiency in storage
- Efficiency in queries
  - Storage model (ROLAP, MOLAP, HOLAP)

Physical design

- Objective: improve performance
- Architecture: ROLAP, MOLAP, HOLAP
- Storage strategy and query optimization
  - Denormalization
  - Partitioning
  - Index
  - Compression
  - Materialized Views
- Big data is different?

Physical design

- Different physical storage and query structures
  - Multidimensional Systems (MOLAP): DBMS built specifically for data analysis
  - Relational systems (ROLAP) Relational DBMS with certain extensions
  - Hybrid systems: HOLAP

ROLAP

- ROLAP:
  - The DW is built on top of a relational DBMS.
  - The relational DBMS vendors offer extensions and tools to use the RDBMS as a DW. Eg. OLAP extensions in SQL, functions, mining tools, …

ROLAP

- ROLAP:
  - Advantages:
    - There is no limit on the amount of data.
    - Functionality already available.
    - Cheap??
  - disadvantages:
    - Performance: queries on normalized db
    - Limited functionality for SQL. For example, to perform complex calculations.

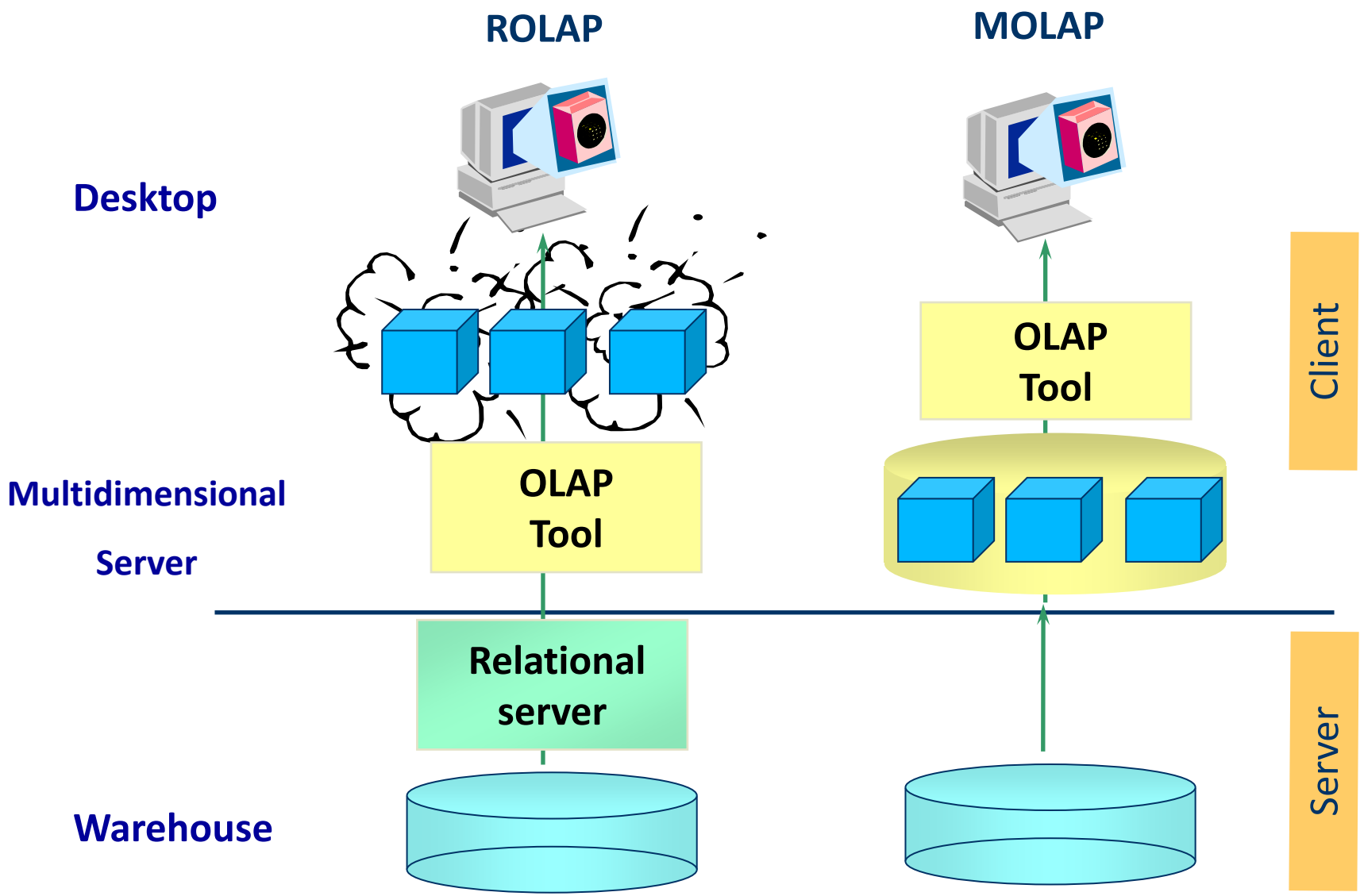
MOLAP

- Consist of physically storing data in multidimensional structures so that the external representation *matches* the internal representation.
- BD complexity is hidden from the users
- analysis is done on aggregate data and precalculated metrics or indicators.
- Specific functionality:
  - Array data structures (proprietary formats)
  - Query Optimization
  - Data Compaction

MOLAP

- MOLAP:
  - advantages:
    - Performance: slide & dice specific operations.
    - Complex calculations are pre-generated.
  - disadvantages:
    - Limited size
    - New investment: this technology is not usually present in enterprises. Also usually proprietary.

ROLAP and MOLAP

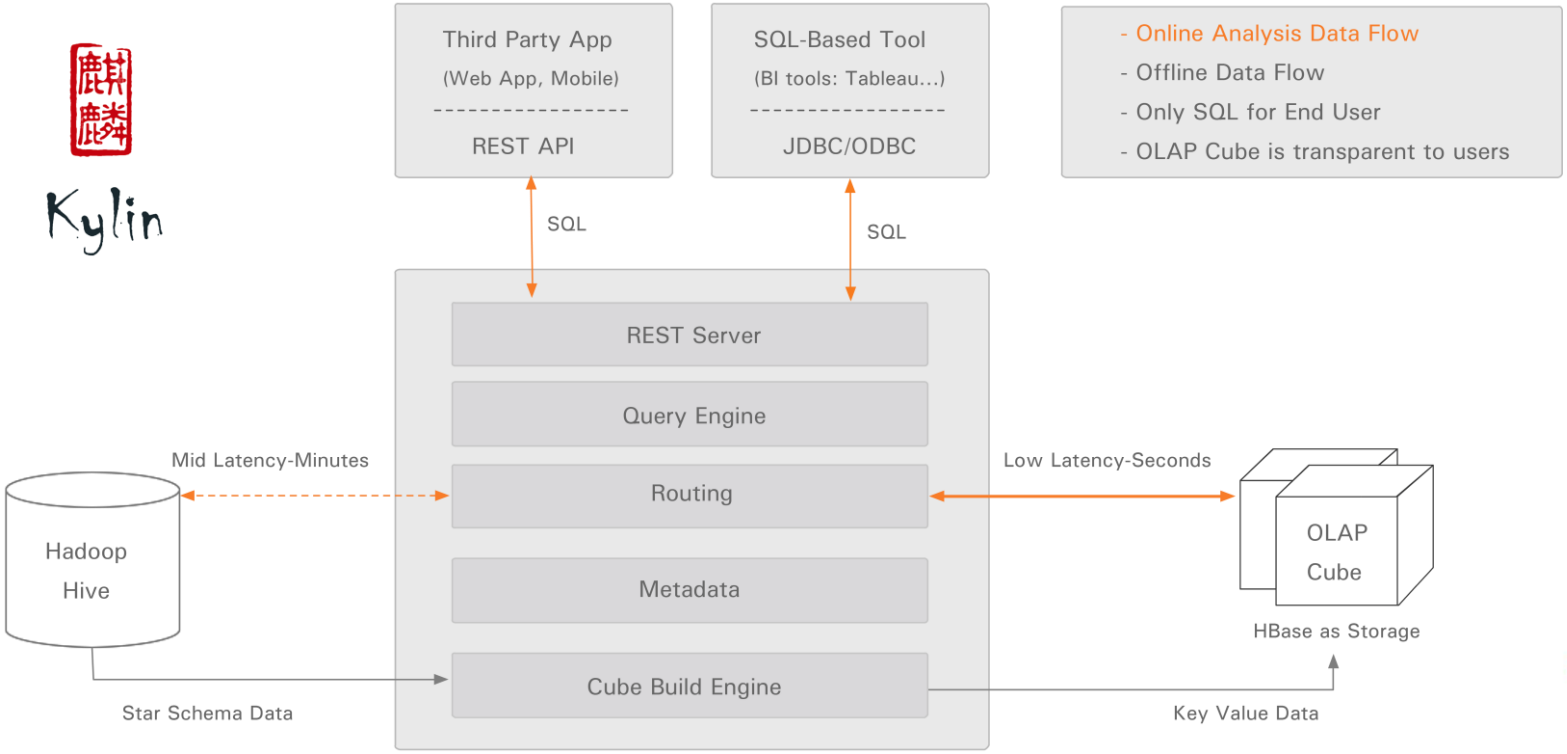


Apache Kylin

- Apache Kylin: http://kylin.apache.org/
- Open source Distributed Analytics Engine designed to provide SQL interface and multi-dimensional analysis (**MOLAP**) on Hadoop supporting extremely large datasets, original contributed from eBay Inc.
- Extremely Fast OLAP Engine at designed to reduce query latency on Hadoop
- Features
  - ANSI SQL Interface supports most ANSI SQL query functions
  - Interactive Query Capability: low latency
  - MOLAP Cube: User can define a data model and pre-build in Kylin
  - Seamless Integration with BI Tools
  - Other Highlights:
    - Compression and Encoding Support
    - Incremental Refresh of Cubes
    - Approximate Query Capability for distinct Count (HyperLogLog)

Apache Kylin

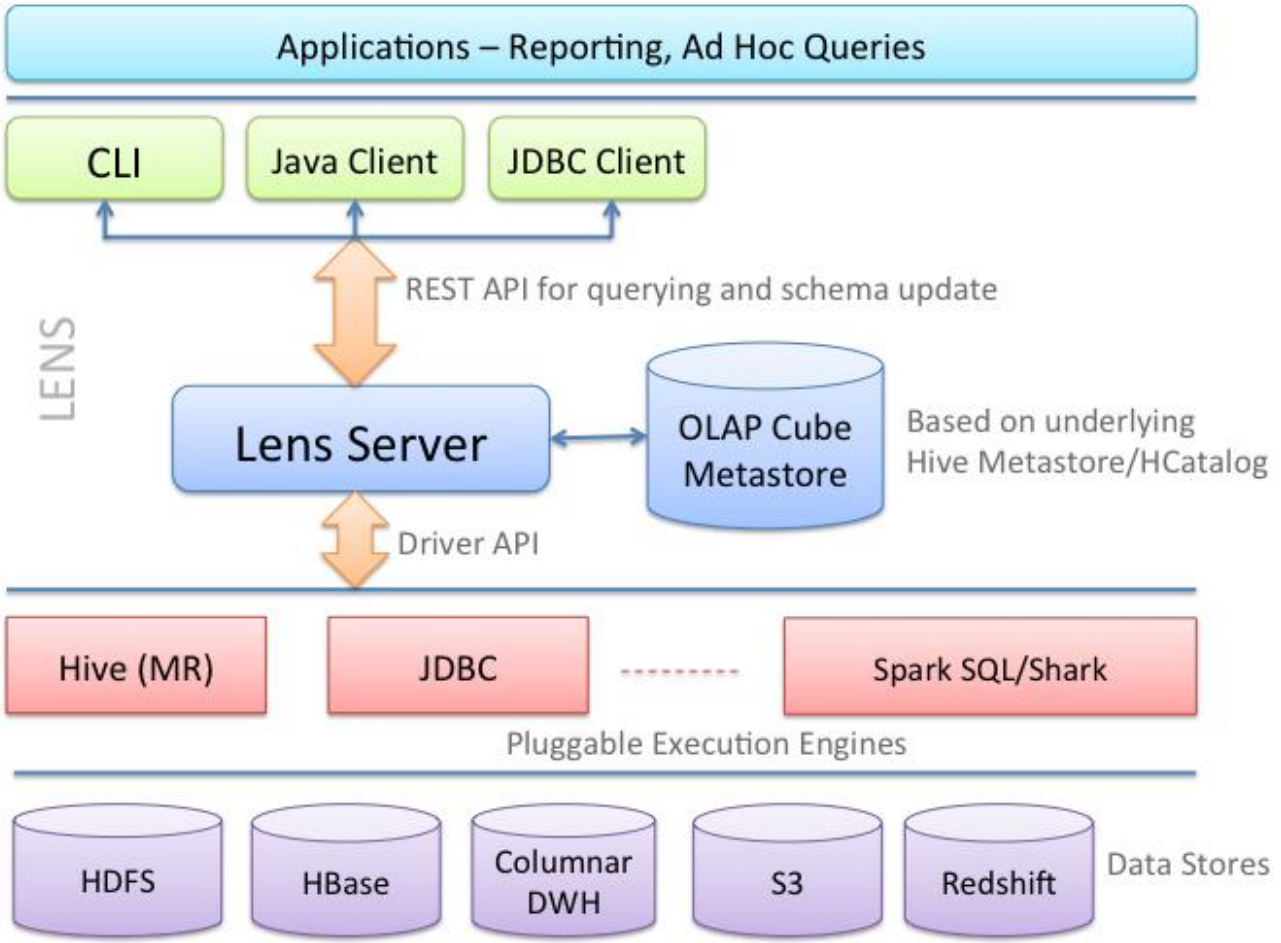
- Identify a Star Schema on Hadoop. Kylin only supports the star schema. You are limited to a single fact table for each cube.
- Build Cube: precomputing the various dimensional combinations and the measure aggregates via Hive queries and populating HBase with the results.
- Query with ANSI-SQL and get results in sub-second, via ODBC, JDBC or RESTful API.



Apache Lens

- APACHE LENS: https://lens.apache.org/
- Lens provides an Unified Analytics interface: a single view of data across multiple tiered data stores and optimal execution environment for the analytical query.
- It seamlessly integrates Hadoop with traditional data warehouses to appear like one: ROLAP architecture.
- At a high level the project provides these features:
  - Simple metadata layer which provides an abstract view over tiered data stores
  - Single shared schema server based on the Hive Metastore - This schema is shared by data pipelines (HCatalog) and analytics applications.
  - OLAP Cube QL which is a high level SQL like language to query and describe data sets organized in data cubes.
  - A JDBC driver and Java client libraries to issue queries, and a CLI for ad hoc queries.
  - Lens application server - a REST server which allows users to query data, make schema changes, scheduling queries and enforcing quota limits on queries.
  - Driver based architecture allows plugging in reporting systems like Hive, Columnar data warehouses, Redshift etc.
  - Cost based engine selection - allows optimal use of resources by selecting the best execution engine for a given query based on the query cost.

Apache Lens

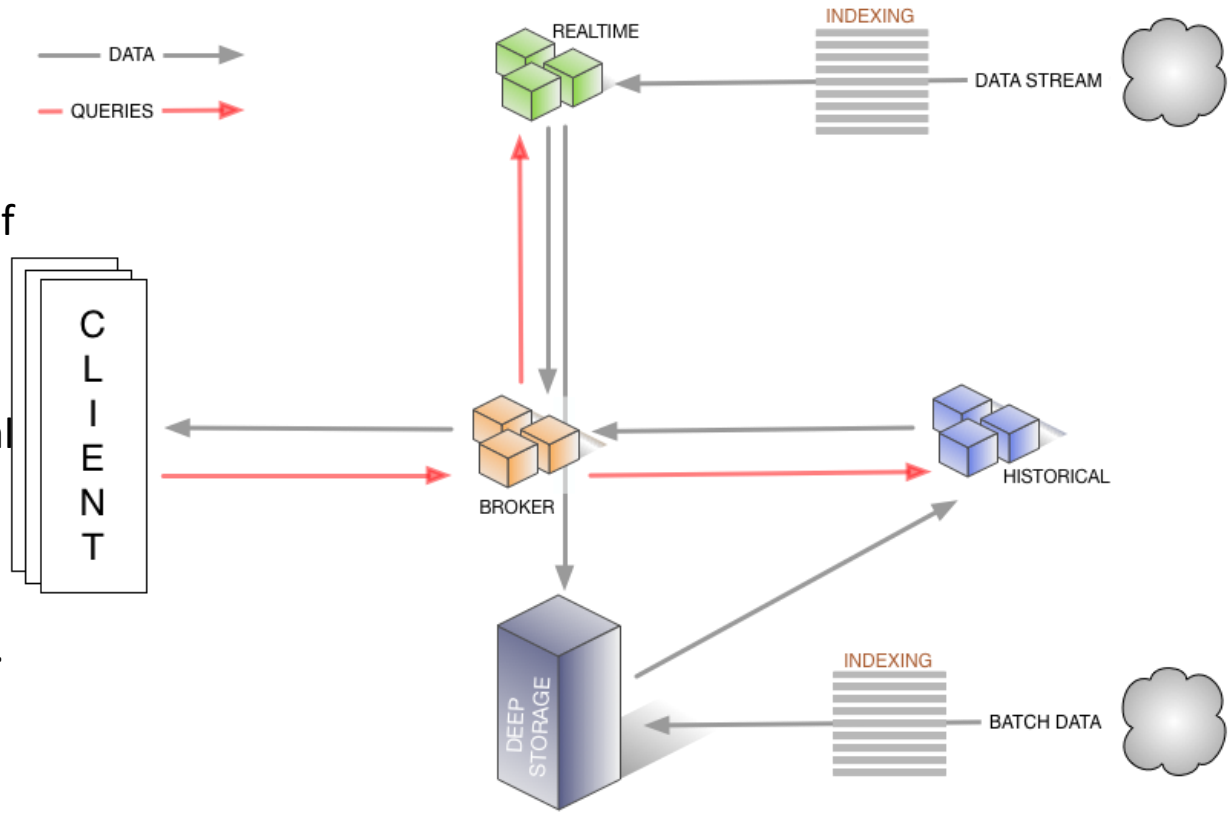

Applications – Reporting, Ad Hoc Queries
CLI
Java Client
JDBC Client
REST API for querying and schema update
LENS
Lens Server
OLAP Cube Metastore
Based on underlying Hive Metastore/HCatalog
Driver API
Hive (MR)
JDBC
Spark SQL/Shark
Pluggable Execution Engines
HDFS
HBase
Columnar DWH
S3
Redshift
Data Stores

Apache Druid

- Druid http://druid.io/druid.html
- Druid is an open source data store designed for OLAP queries on event data.
- Neither MOLAP nor ROLAP. Pure Big Data architecture.
- Features:
  - columnar storage format for partially nested data structures
  - Indexed with bitmap indexes
  - Compressed using various algorithms
  - hierarchical query distribution with intermediate pruning
  - indexing for quick filtering
  - realtime ingestion (ingested data is immediately available for querying)
  - fault-tolerant distributed architecture that doesn't lose data

Druid

- **Historical nodes** handle storage and querying on "historical" data (non-realtime). HN download segments from deep storage, respond to the queries from broker nodes about these segments, and return results to the broker nodes.
- **Coordinator nodes** monitor the grouping of historical nodes to ensure that data is available, replicated and in a generally "optimal" configuration.
- **Broker nodes** receive queries from external clients and forward those queries to Realtime and Historical nodes. When Broker nodes receive results, they merge these results and return them to the caller.
- **Indexing Service nodes** form a cluster of workers to load batch and real-time data into the system as well as allow for alterations to the data stored in the system.
- **Realtime nodes** also load real-time data into the system. More limited but faster.



Database partitioning

- Managing large tables with small subtables
  - Allows parallel and distributed execution
  - Partition prunning
  - Partition wise joins
  - Partition data processing: ETL, refresing, backup
- Horizontal
  - Rows of a table are separated
- Vertical
  - Columns of a table are separated

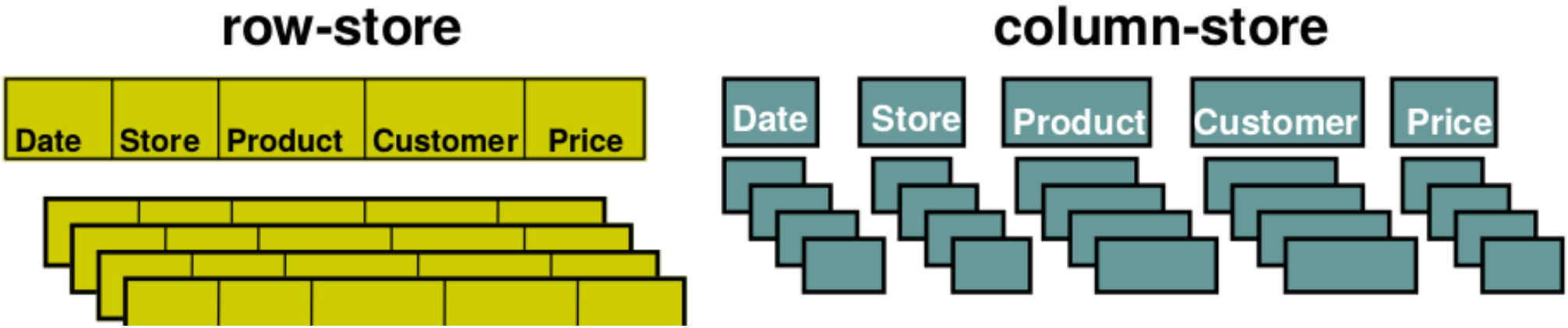
Horizontal partitioning

- Rows of the same table are stored in different locations according to some criteria
- Remember that in DW the load is incremental (usually there are not update).
  - New inserts do not produce table movements, can be just as simple are adding a partition.
  - Easy to delete by “time”.
- Criteria
  - Range, List, Hash, Partitioned Index
  - Composed: range + hash, list+hash

Vertical partitioning

- A new paradigm for database: NoSQL Columnar DB
  - E.g. Hbase, Apache Cassandra
- Not only for multidimensional db
- For DW:
  - A number of operations are Aggregations by columns
  - Only columns affected by operation are retrieved
  - Compression of columns is very efficient
    - Text columns without predefined size
  - Some columns may change more than others

Columnar databases



Pros:
-easy to add/modify a record
Cons
-might read in unnecessary data

Pros:
-only need to read in relevant data
- Column compression easy and efficient (light algorithms)
- Aggregated queries very fast.
Cons:
- tuple writes require multiple accesses

Note: Not any more here because you see them in other course.

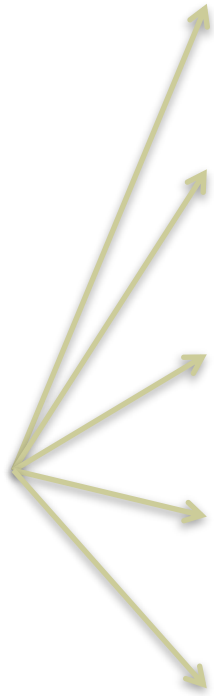
[http://cs-www.cs.yale.edu/homes/dna/talks/Column_Store_Tutorial_VLDB09.pdf]

Indexing

- Index: secondary data structure for arbitrary access and efficient
- 3 characteristics:
  - Content:
    - Value of the attribute indexed
    - Address to the storage
  - Sorted entries according to the value for Efficient searches
  - Small size: usually fits in one data block
- Some types: B-Trees, Hash
  - Specific for OLAP: Bitmap, Join Index
- Used only for retrieving few values

Index: General concept

codA	Bloque
00	B4
11	B1
22	B2
33	B2
44	B4
55	B3
77	B3
88	B1
99	B5

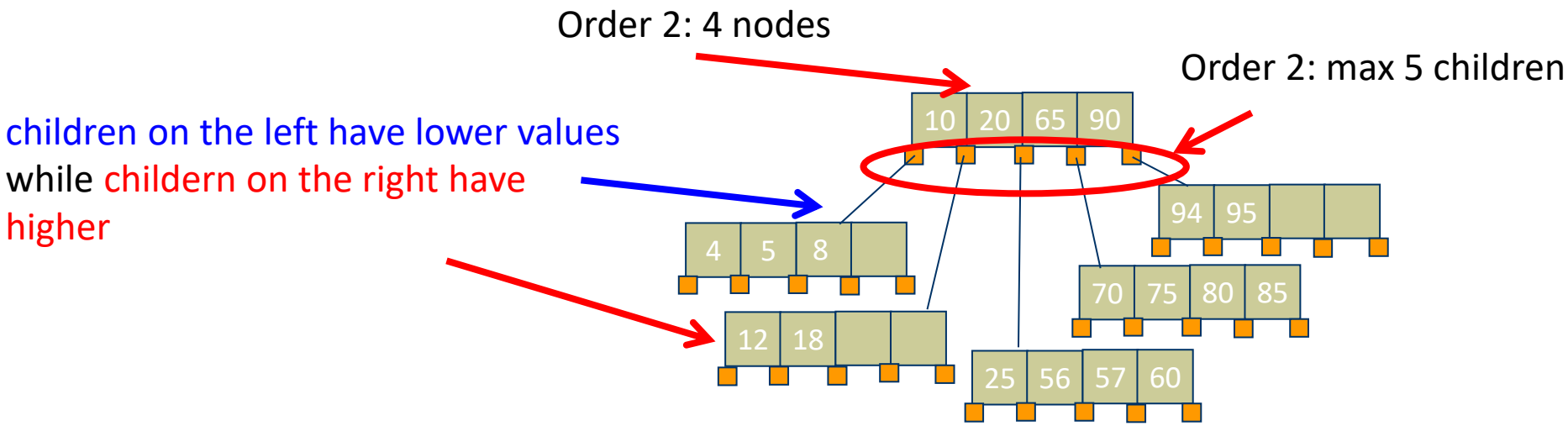


Bloque	codA	nombre	agencia	fechaNacim
B1	88	Fele Martínez	Glam	22/02/75
	11	Najwa Nimri	Actors	14/02/72
B2	33	Nancho Novo	Rol	17/09/78
	22	Santiago Segura	Amiguetes	17/07/65
B3	77	Luis Tosar	BCN	13/10/71
	55	Candela Peña	Actors	14/07/73
B4	00	Maribel Verdú	Glam	02/10/70
	44	Penélope Cruz	BCN	28/04/74
B5	99	Javier Bardem	BCN	01/03/69

Just one block

B-Tree Index

- Deep balanced tree: size at same depth
- N-order:
  - Each node has between n and 2n elements
  - Each node with “n” elements has “n+1” children
- Variations



Bitmap Index

- Index on a particular column
- Each value in the column has a bit vector: bit-op is fast
- The length of the bit vector: # of records in the base table
- The i-th bit is set if the i-th row of the base table has the value for the indexed column

Cust	Region	Type
C1	Asia	Retail
C2	Europe	Dealer
C3	Asia	Dealer
C4	America	Retail
C5	Europe	Dealer

RecID	Asia	Europe	America
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

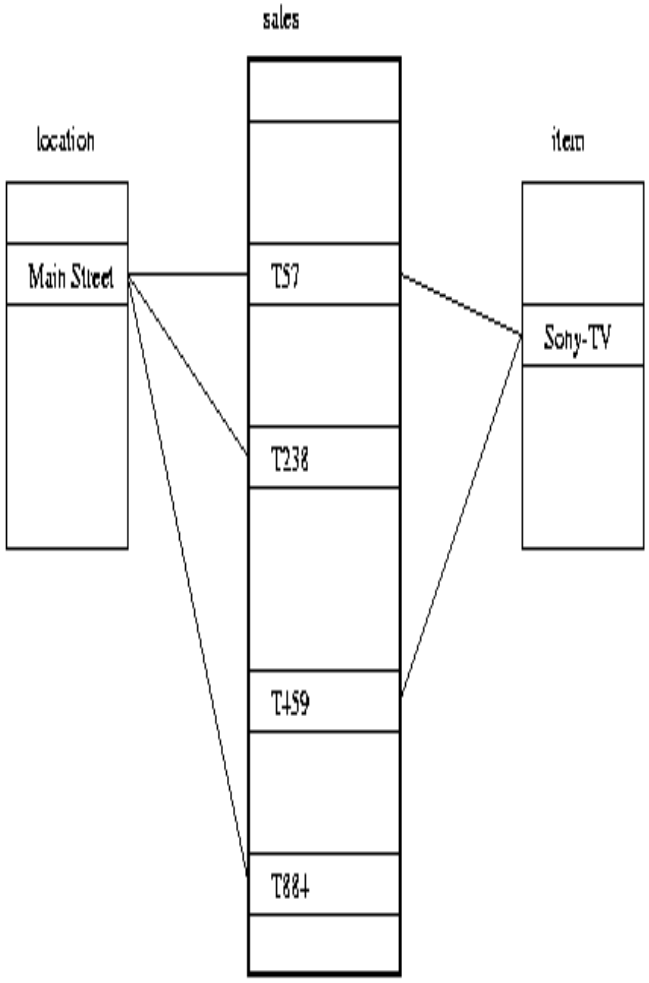
RecID	Retail	Dealer
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Bitmap Index

- Reduced response time for large classes of ad hoc queries: Flexible WHERE.
- Reduced storage requirements compared to other indexing techniques.
  - Fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of disk space because the indexes can be several times larger than the data in the table.
  - Bitmap indexes are typically only a fraction of the size of the indexed data in the table.
- Efficient maintenance during parallel DML and loads.
- The advantages of using bitmap indexes are greatest for columns in which the ratio of the number of distinct values to the number of rows in the table is small (1:100).
  - A table with one million rows, a column with 10,000 distinct values is a candidate
  - Comparison, union and aggregation use bit arithmetic.
- Not suitable for high cardinality domains: too many columns (better B-Tree)
  - Compression

[Bitmap] Join Index

- It materializes relational join in JI file and speeds up relational join
  - Join index: JI(R-id, S-id) where R (R-id, …) JOIN S (S-id, …)
- In DW: join between facts and dimensions.
  - Equi-inner join between PK and FK
  - E.g. fact table: Sales and two dimensions city and product
    - A join index on city maintains for each distinct city a list of R-IDs of the tuples recording the Sales in the city
  - Join indices can span multiple dimensions
- NOTE: Related to denormalization



Compression

- Used in multidimensional DBMS.
- Saving disk space, increasing memory efficiency, and improving query performance by compressing heap-organized tables.
- Overhead for updating, deleting, and processing in general.
- Hybrid Columnar Compression: groups of rows are stored in columnar format, with the values for a given column stored and compressed together.
- Storing column data together, with the same data type and similar characteristics, drastically increases the storage savings achieved from compression.

Materialized Views

- Materialized views are query results that have been stored in advance so long-running calculations are not necessary when you actually execute your SQL statements.
- Are tables or partitioned tables and behave like indexes in that they are used transparently and improve performance.
- Usefull for aggregates.
- Problem:
  - Resources consumption
  - Refreshing policies: on demand, on commit.

Efficient Data Cube Computation

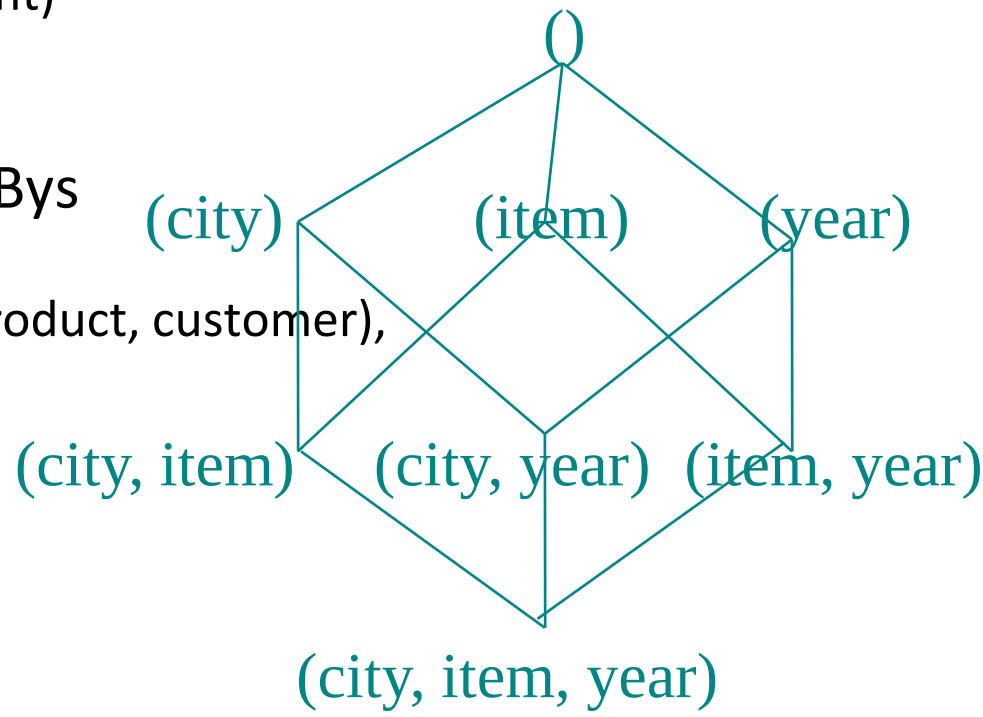
- Data cube can be viewed as a lattice of cuboids
  - The bottom-most cuboid is the base cuboid
  - The top-most cuboid (apex) contains only one cell
  - How many cuboids in an n-dimensional cube with L levels?

$$T = \prod_{i=1}^{n} (L_i + 1)$$

- Materialization of data cube
  - Materialize every (cuboid) (full materialization), none (no materialization), or some (partial materialization)
  - Selection of which cuboids to materialize
    - Based on size, sharing, access frequency, etc.

The “Compute Cube” Operator

- Cube definition and computation in DMQL
  - define cube sales [item, city, year]: sum (sales_in_dollars)
  - compute cube sales
- Transform it into a SQL-like language (with a new operator cube by, introduced by Gray et al.’96)
  - SELECT item, city, year, SUM (amount)
  - FROM SALES
  - CUBE BY item, city, year
- Need compute the following Group-Bys
  - (date, product, customer),
  - (date,product),(date, customer), (product, customer),
  - (date), (product), (customer)
  - ()



Efficient Processing OLAP Queries

- Determine which operations should be performed on the available cuboids
  - Transform drill, roll, etc. into corresponding SQL and/or OLAP operations, e.g., dice = selection + projection
- Determine which materialized cuboid(s) should be selected for OLAP op.
  - Let the query to be processed be on {brand, province_or_state} with the condition “year = 2004”, and there are 4 materialized cuboids available:
    - 1) {year, item_name, city}
    - 2) {year, brand, country}
    - 3) {year, brand, province_or_state}
    - 4) {item_name, province_or_state} where year = 2004
    - Which should be selected to process the query?
- Explore indexing structures and compressed vs. dense array structs in MOLAP