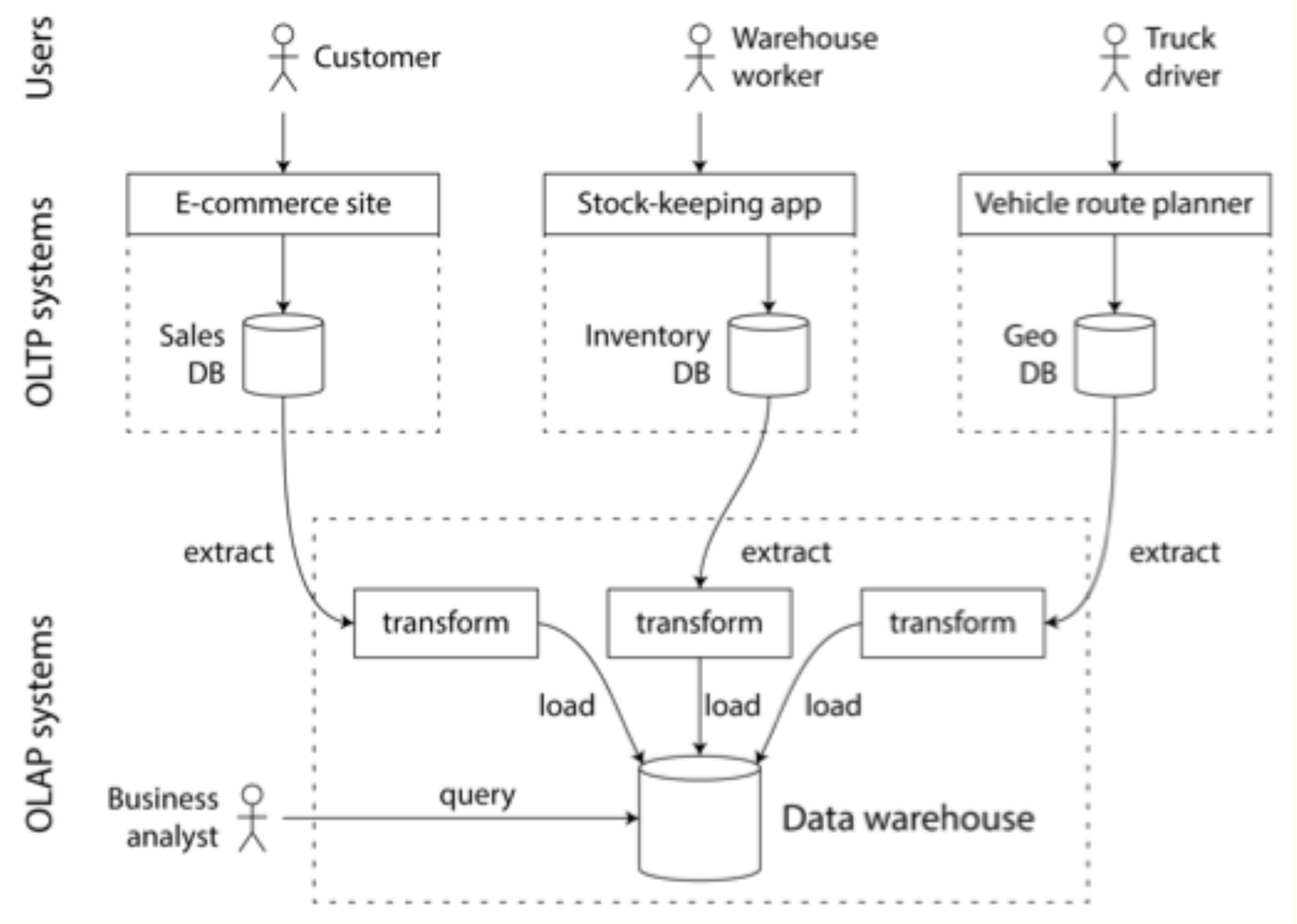
SQL for Data Scientists: 2

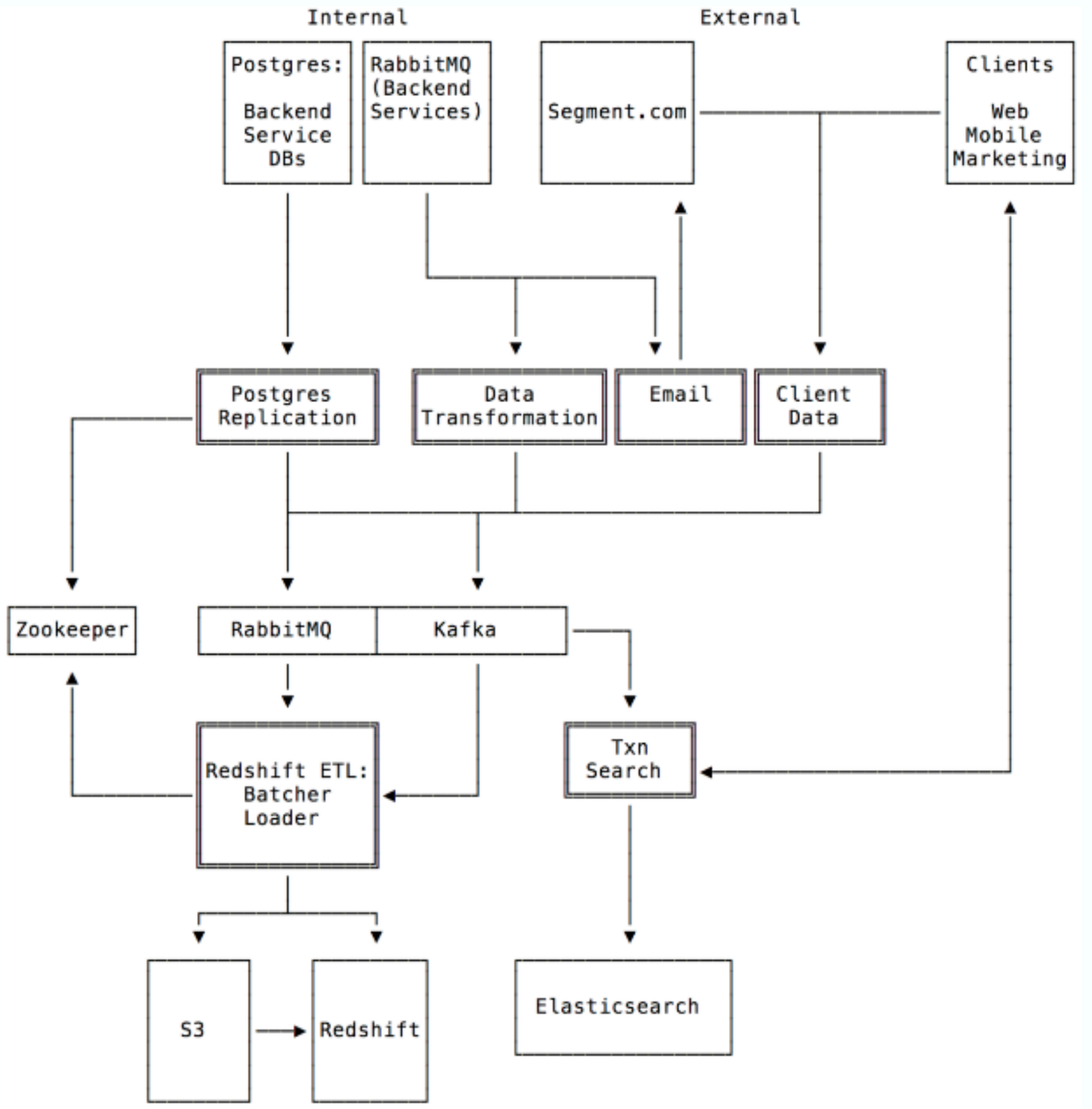
Rahul Dave, Univ.Ai

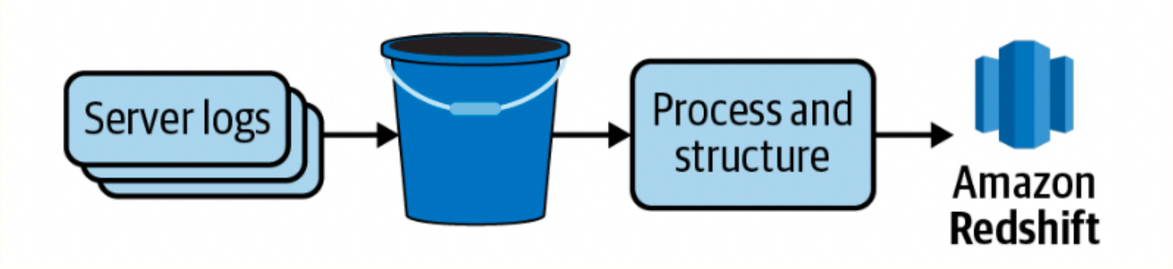
Univ.AI

Where are the databases used?



Data Pipelines write data into a OLAP system





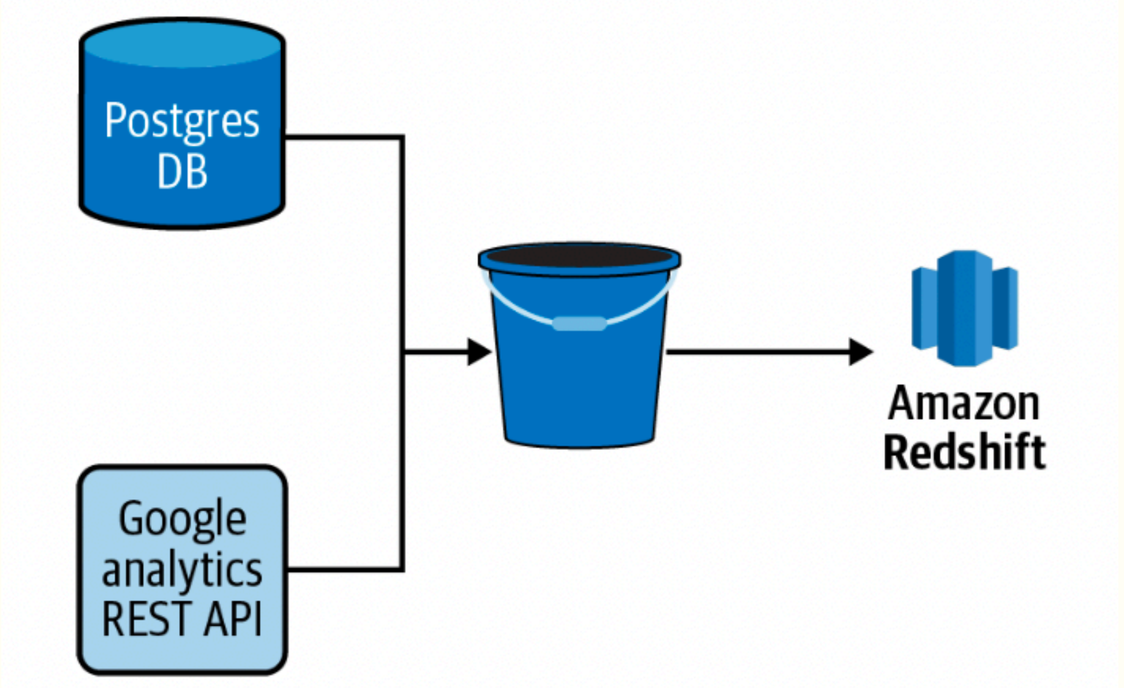
Where does data come from?

E-commerce databases

Logs

Customer tracking

Other sites

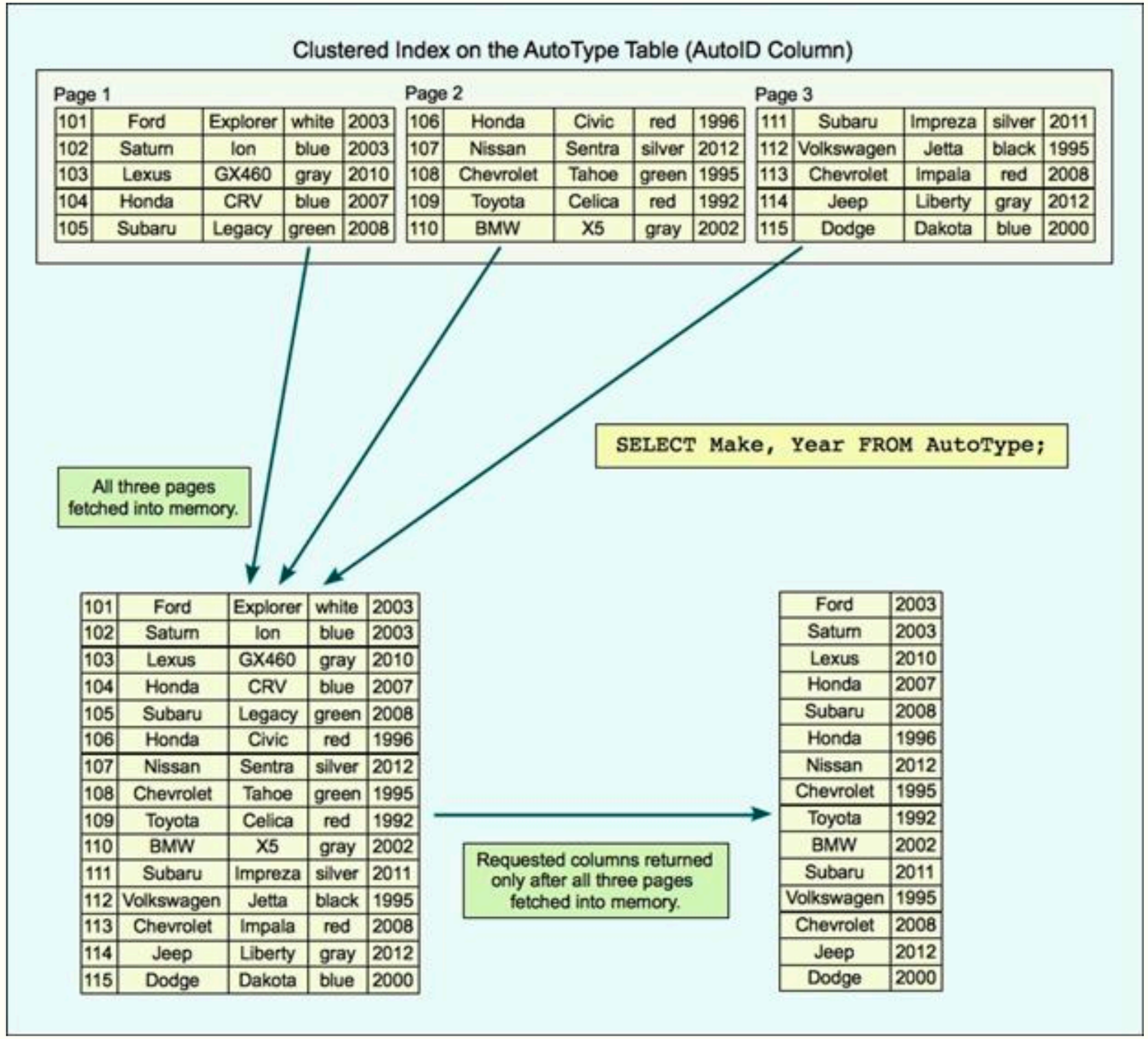


Storage Components of a RDBMS

- the heap file: this is where the rows or columns are stored
- regular relational databases use row oriented heap files
- an index file(s): this is where the index for a particular attribute is stored
- sometimes you have a clustered index (all data stored in index) or covering index (some data is stored in index)
- the WAL or write-ahead log: this is used to handle transactions

Row-oriented Storage

- heapfile or clustered index is a set of rows
- index could be a tree with appropriate pointer to heapfile offset

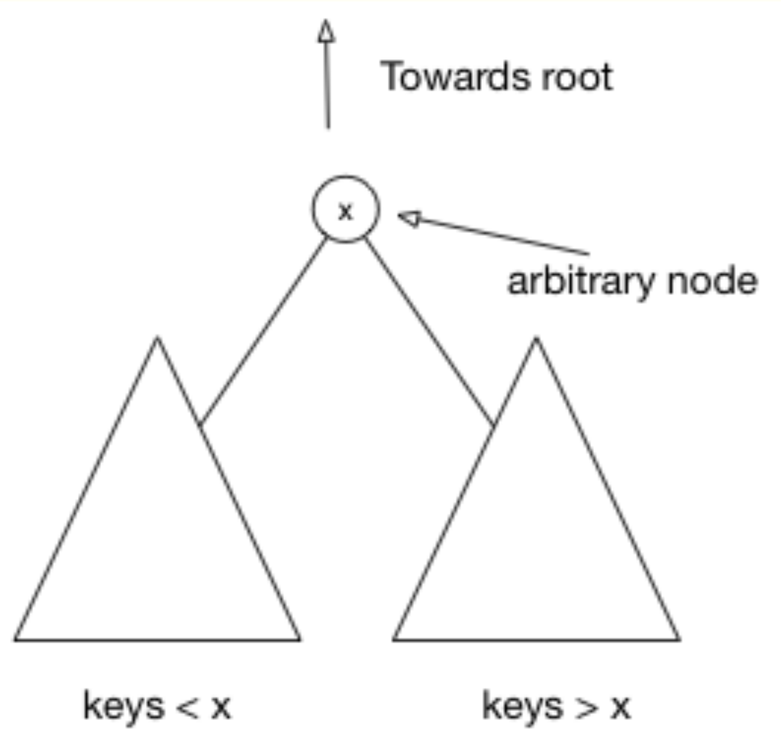


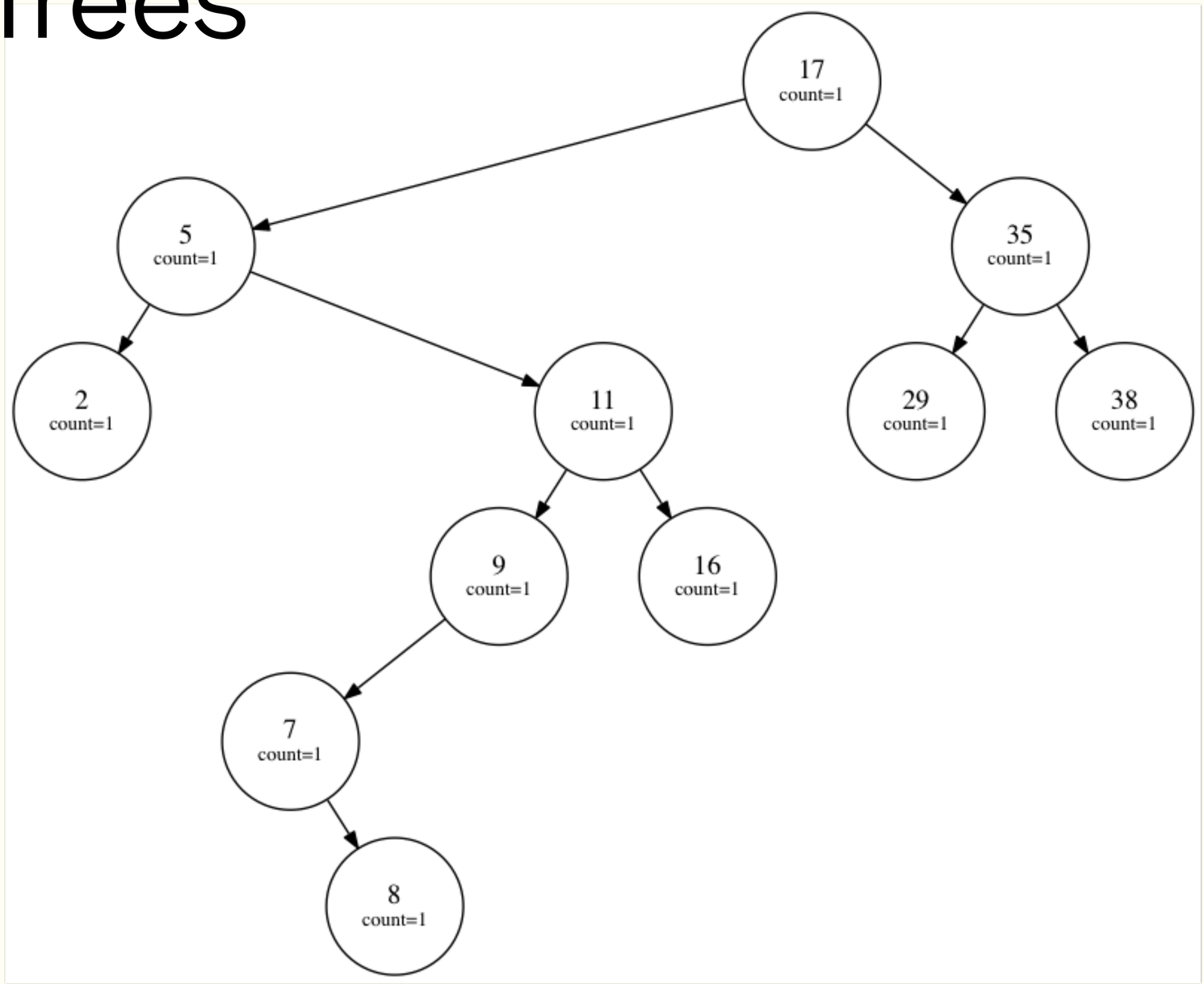
Index

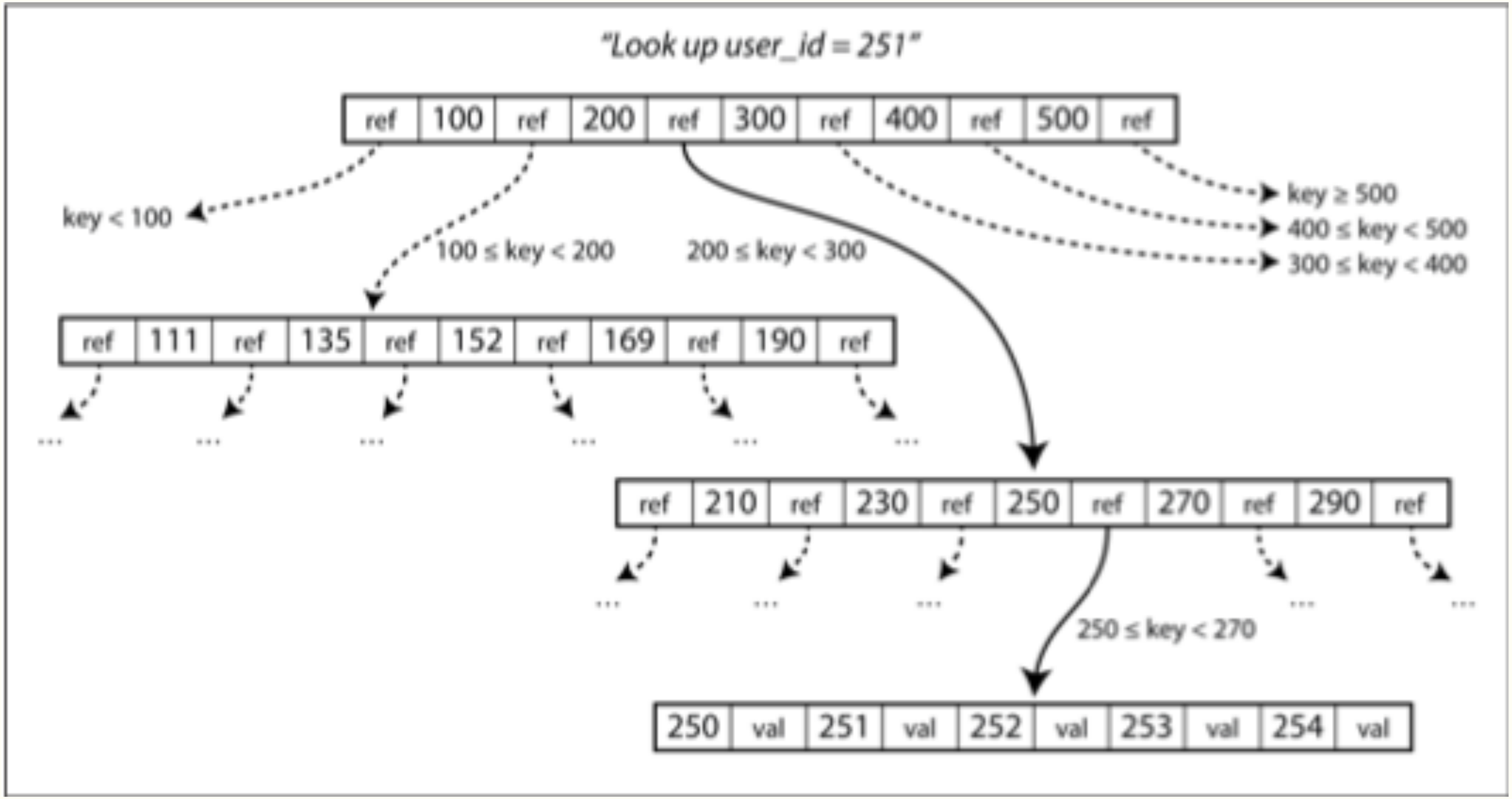
- An index is an additional structure derived from the primary data
- There is overhead on writes: indexes speed up queries but slow down writes
- Whenever we want to maintain our search dataset in memory, sorted, we use something like a Binary Search Tree instead.
- They perform well with dynamic data where insertions and deletions are frequent, because of the so called O(height) guarantees. If we can balance the height out guarantees will be good.

Binary Search Trees

- These have the property that for any value x, numbers less go to the left and numbers right go to the right
- Consider a list: [17,5,35,2,11,29,38,9,16,7,8]
- Then the binary tree you get is:







Btrees

Univ.AI

BTrees

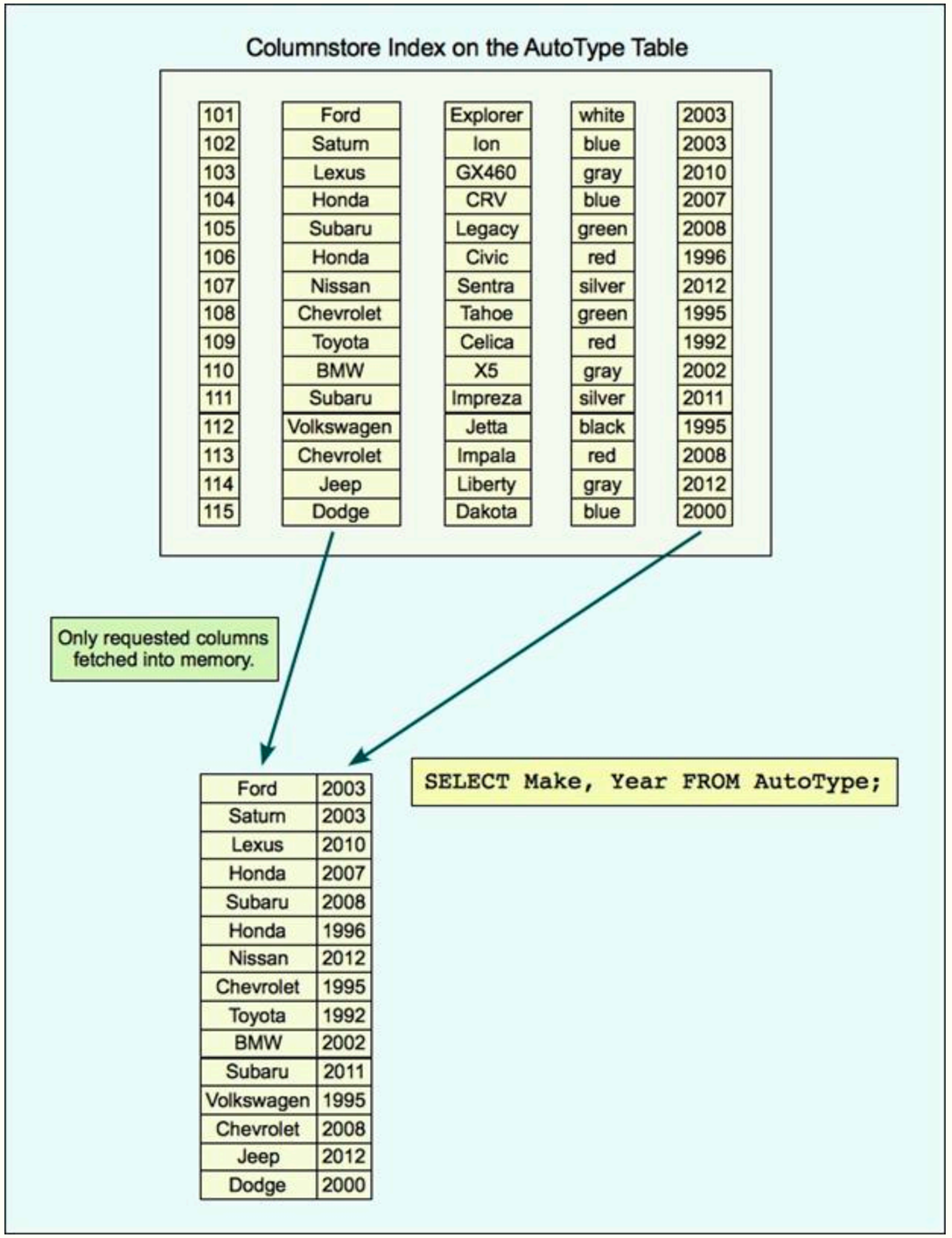
- (from https://loveforprogramming.quora.com/Memory-locality-the-magic-of-B-Trees): ”A linked sorted distributed range array with predefined sub array size which allows searches, sequential access, insertions, and deletions in logarithmic time. "
- It is a generalization of a binary tree, but the branching factor is much higher, and the depth thus smaller
- btrees break database into pages, and read-or-write one page at a time. A page is about 4k in size (see https://www.tutorialspoint.com/operating_system/os_virtual_memory.htm)
- Leaf pages contain all the values and may represent a clustered index
- The pointers in a btree are disk based pointers

From Normalization to De-Normalization

- Normalization is important in OLTP databases for reducing the surface area of transactions: less the surface area, less the locking, less the rollback.
- But pointers are not useful in datasets you want to feed to BI software such as Tableau or machine learning software such as sklearn
- There you want to de-normalize datasets by joining multiple tables
- Usually you will want to do an inner join. However outer joins are important when you need your full data to be represented, even if they lead to more nulls.
- Think of your problem and ask: what join do I need?

Column-oriented Storage

- store values from each column together in separate storage
- lends itself to compression with bitmap indexes
- compressed indexes can fit into cache and are usable by iterators
- several different sort orders can be redundantly stored
- writing is harder: updating a row touches many column files
- but you can write an in-memory front sorted store (row or column), and eventually merge onto the disk



Bitmap Indexes

- lends itself to compression with bitmap indexes and run-length encoding. This involves choosing an appropriate sort order. The index then can be the data (great for IN and AND queries): there is no pointers to “elsewhere"
- bitwise AND/OR can be done with vector processing

Column values:

product_sk:	69	69	69	69	74	31	31	31	31	29	30	30	31	31	31	68	69	69

Bitmap for each possible value:

product_sk = 29:	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
product_sk = 30:	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
product_sk = 31:	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0
product_sk = 68:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
product_sk = 69:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
product_sk = 74:	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Run-length encoding:

product_sk = 29:	9, 1	(9 zeros, 1 one, rest zeros)
product_sk = 30:	10, 2	(10 zeros, 2 ones, rest zeros)
product_sk = 31:	5, 4, 3, 3	(5 zeros, 4 ones, 3 zeros, 3 ones, rest zeros)
product_sk = 68:	15, 1	(15 zeros, 1 one, rest zeros)
product_sk = 69:	0, 4, 12, 2	(0 zeros, 4 ones, 12 zeros, 2 ones)
product_sk = 74:	4, 1	(4 zeros, 1 one, rest zeros)

Column Storage in Data Science

- DuckDB is an example of a columnar OLAP database, like SQLITE but with data stored as columns.
- Parquet files on python, which pandas can save into and read from, are column oriented storage as well. They are optimized for space.
- The Apache arrow format is a column format as well, designed as a in-memory format, requiring 0 serialization. (For differences see https://stackoverflow.com/questions/56472727/difference-between-apache-parquet-and-arrow).
- Today, any data supporting software: redshift, snowflake, dremio, presto, pandas, duckdb support loading parquet files and use the arrow in-memory format.
- SQL has gone from SQL on row or columnar databases to SQL over distributed heterogeneous database files such as parquet files. It has made SQL even more important.

Where is your data?

Does not matter if you can get to it via SQL.