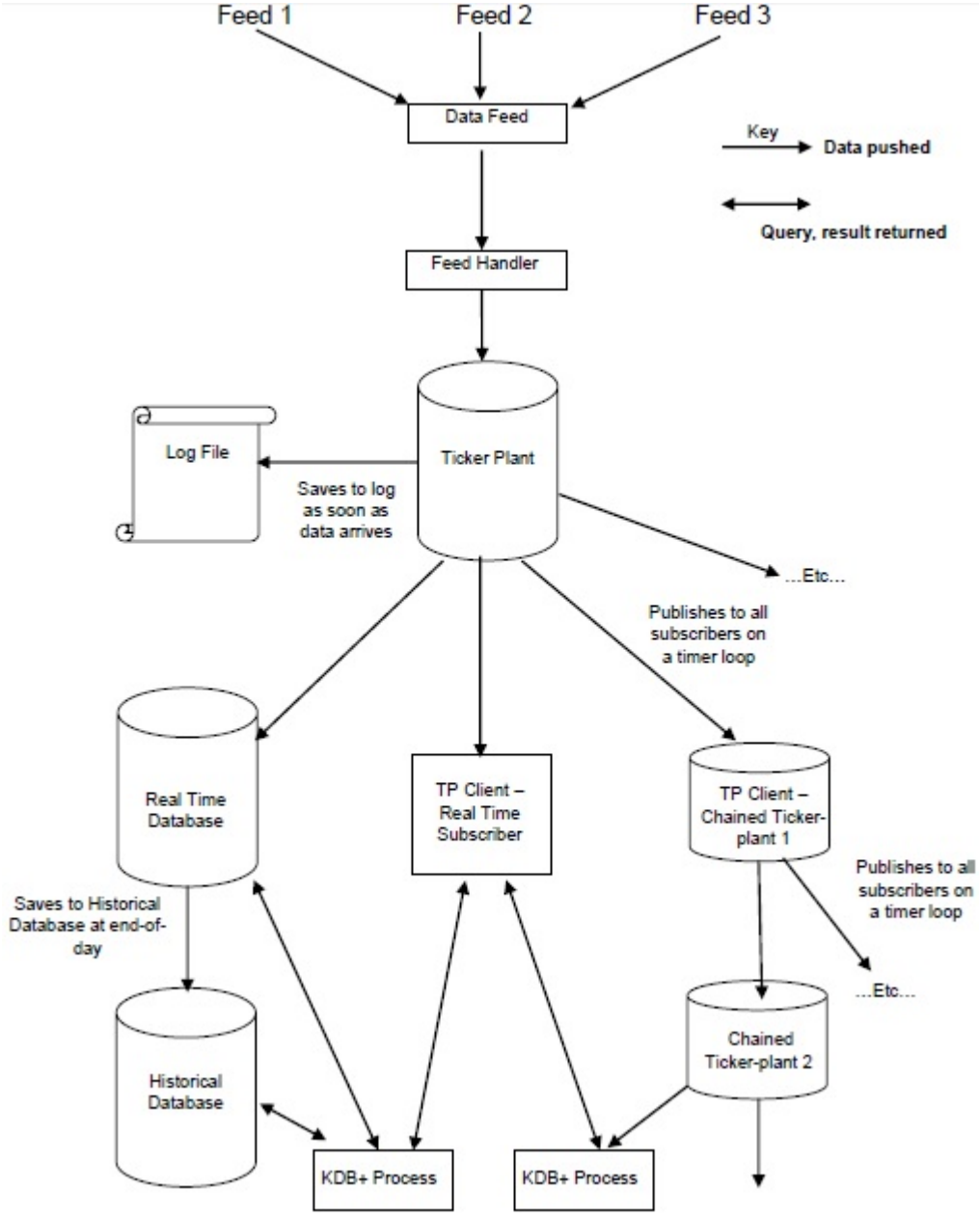
KDB+ ARCHITECTURE

Kdb+ is a high-performance, high-volume database designed from the outset to handle tremendous volumes of data. It is fully 64-bit, and has built-in multi-core processing and multi-threading. The same architecture is used for real-time and historical data. The database incorporates its own powerful query language, **q,** so analytics can be run directly on the data.

kdb+tick is an architecture which allows the capture, processing, and querying of real-time and historical data.

Kdb+/ tick Architecture

The following illustration provides a generalized outline of a typical Kdb+/tick architecture, followed by a brief explanation of the various components and the through-flow of data.



- The **Data Feeds** are a time series data that are mostly provided by the data feed providers like Reuters, Bloomberg or directly from exchanges.
- To get the relevant data, the data from the data feed is parsed by the **feed handler**.
- Once the data is parsed by the feed handler, it goes to the **ticker-plant**.

- To recover data from any failure, the ticker-plant first updates/stores the new data to the log file and then updates its own tables.
- After updating the internal tables and the log files, the on-time loop data is continuously sent/published to the real-time database and all the chained subscribers who requested for data.
- At the end of a business day, the log file is deleted, a new one created and the real-time database is saved onto the historical database. Once all the data is saved onto the historical database, the real-time database purges its tables.

Components of Kdb+ Tick Architecture

Data Feeds

Data Feeds can be any market or other time series data. Consider data feeds as the raw input to the feed-handler. Feeds can be directly from the exchange *live − streamingdata*, from the news/data providers like Thomson-Reuters, Bloomberg, or any other external agencies.

Feed Handler

A feed handler converts the data stream into a format suitable for writing to kdb+. It is connected to the data feed and it retrieves and converts the data from the feed-specific format into a Kdb+ message which is published to the ticker-plant process. Generally a feed handler is used to perform the following operations −

- Capture data according to a set of rules.
- Translate */enrich* that data from one format to another.
- Catch the most recent values.

Ticker Plant

Ticker Plant is the most important component of KDB+ architecture. It is the ticker plant with which the real-time database or directly subscribers *clients* are connected to access the financial data. It operates in **publish and subscribe** mechanism. Once you obtain a subscription *license*, a tick *routinely* publication from the publisher *tickerplant* is defined. It performs the following operations −

- Receives the data from the feed handler.
- Immediately after the ticker plant receives the data, it stores a copy as a log file and updates it once the ticker plant gets any update so that in case of any failure, we should not have any data loss.
- The clients *real − timesubscriber* can directly subscribe to the ticker-plant.
- At the end of each business day, i.e., once the real-time database receives the last message, it stores all of today's data onto the historical database and pushes the same to all the subscribers who have subscribed for today's data. Then it resets all its tables. The log file is also deleted once the data is stored in the historical database or other directly linked subscriber to real time database *rtdb*.
- As a result, the ticker-plant, the real-time database, and the historical database are operational on a 24/7 basis.

Since the ticker-plant is a Kdb+ application, its tables can be queried using **q** like any other Kdb+ database. All ticker-plant clients should only have access to the database as subscribers.

Real-Time Database

A real-time database *rdb* stores today's data. It is directly connected to the ticker plant. Typically it would be stored in memory during market hours *aday* and written out to the historical database *hdb* at the end of day. As the data *rdbdata* is stored in memory, processing is extremely fast.

As kdb+ recommends to have a RAM size that is four or more times the expected size of data per

day, the query that runs on rdb is very fast and provides superior performance. Since a real-time database contains only today's data, the date column *parameter* is not required.

For example, we can have rdb queries like,

```
select from trade where sym = `ibm

OR

select from trade where sym = `ibm, price > 100
```

Historical Database

If we have to calculate the estimates of a company, we need to have its historical data available. A historical database *hdb* holds data of transactions done in the past. Each new day's record would be added to the hdb at the end of day. Large tables in the hdb are either stored splayed *eachcolumnisstoredinitsownfile* or they are stored partitioned by temporal data. Also some very large databases may be further partitioned using **par.txt** *file*.

These storage strategies *splayed*, *partitioned*, *etc.* are efficient while searching or accessing the data from a large table.

A historical database can also be used for internal and external reporting purposes, i.e., for analytics. For example, suppose we want to get the company trades of IBM for a particular day from the trade *orany* table name, we need to write a query as follows −

```
thisday: 2014.10.12

select from trade where date = thisday, sym =`ibm
```

Note − We will write all such queries once we get some overview of the **q** language.