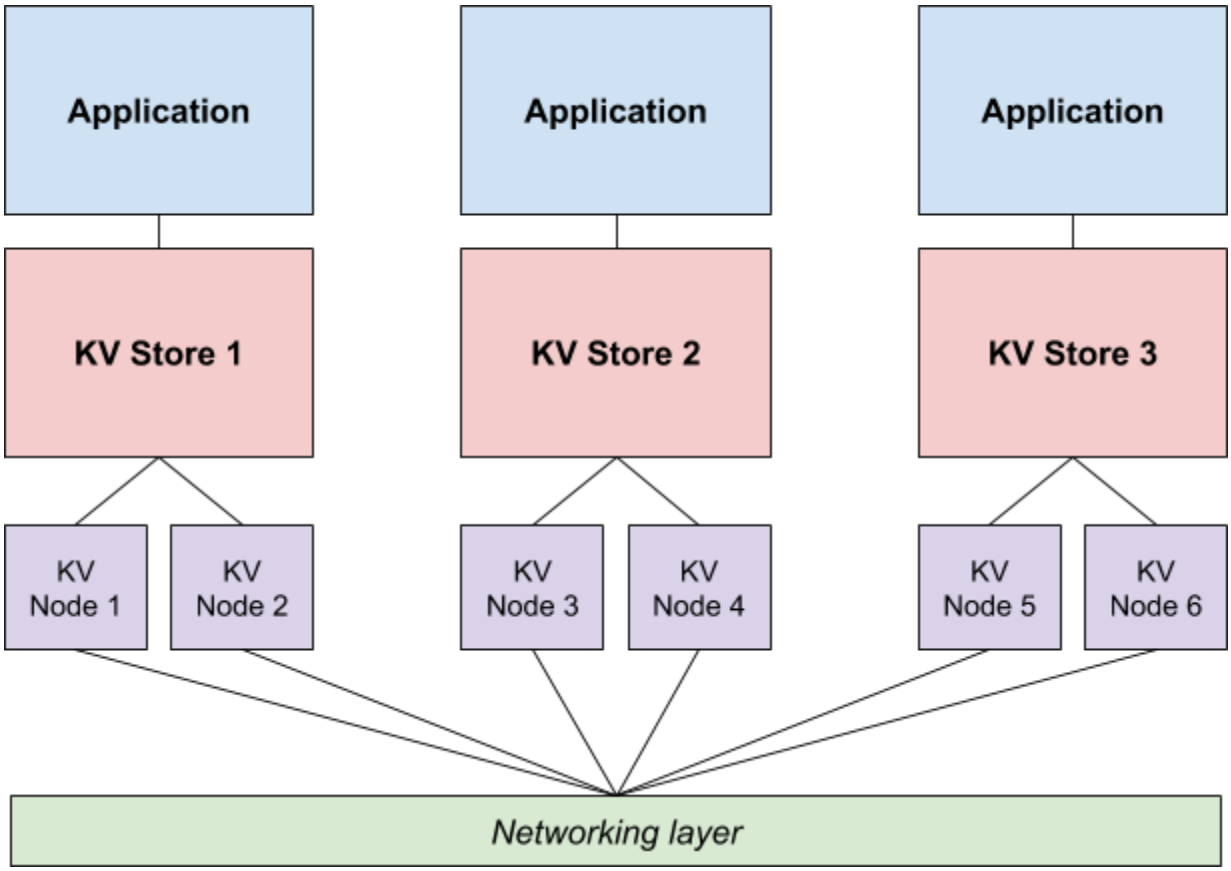
- *Introduction:*
  - The eau2 system is a layered system which allows access to and manipulation of 'big data' over a distributed network of nodes. Nodes are able to talk with each other to accomplish large tasks. Dataframes are automatically chunked into various pieces on creation and are shared amongst the network. All data is read-only.
- *Architecture:*
  - As previously mentioned, eau2 is a layered system: three to be exact. The foundation layer consists of a KV store running on each system. These KV stores hold one or more KVStore_Node objects depending on storage requirements. A KVStore_Node will hold pieces of data and are able to network with other nodes to request/provide data if necessary. The data layer rests above/depends on the foundation. It allows for overarching abstractions of the data in the form of Dataframes and distributed arrays, which allow for easier data manipulation. The top ("application") layer of the system is where the user will live and interact. Here, users will provide custom queries which our system will process efficiently and accurately, handling all the complex computations and networked data sharing occurring beneath the surface.
- *Implementation:*
  - A singular KV Store object is created for each individual machine in the cluster. These KV Stores will create one or more nodes, depending on client needs, to distribute the storage of large data.
  - Data is loaded into a KV Store with a Key object. This Key object consists of a characterized name, and an index indicating the node hosting the value linked to this specific key. A Key's hashed value is derived from the individual characters of the key's name.
  - All data put into the KV Store is serialized before its storage as a generic Value object, which holds the serialized value. This allows for easier shipping around the cluster when external data is needed by a different node.
  - Networking code running on each node will allow for direct peer to peer communication once registered with a master server (ideally, the first KV Store created will act as the master server). This is largely done using built-in POSIX library functions with little abstractions sprinkled in. We support several different message types, each serving a unique function (i.e. sending data, shutdown, register, etc.)
  - Dataframes will serve as a collection of keys so that a singular dataframe can refer to data possibly stored on several various nodes.
  - A distributed column pertains to a specific Dataframe and will automatically chunk its data into PrimitiveArrayChunks as generated. A column maintains an instance of a current cached chunk which is used for quickly adding new data before it becomes full.
- *Use cases:*



- *Open questions:*
  -
- *Status:*
  - We currently have implementations of a networked, distributed KV Store. We can successfully store and retrieve data stored both on singular nodes and distributed across multiple nodes. Our networking implementation sends byte-serialized data directly between nodes once registration has occurred. Our chunked, distributed column data solution is complete, and we are in progress of implementing similar chunked-architecture for maintaining column keys in a Dataframe. We have the “Demo” application running successfully, but are unable to run Linus or the WordCount application.