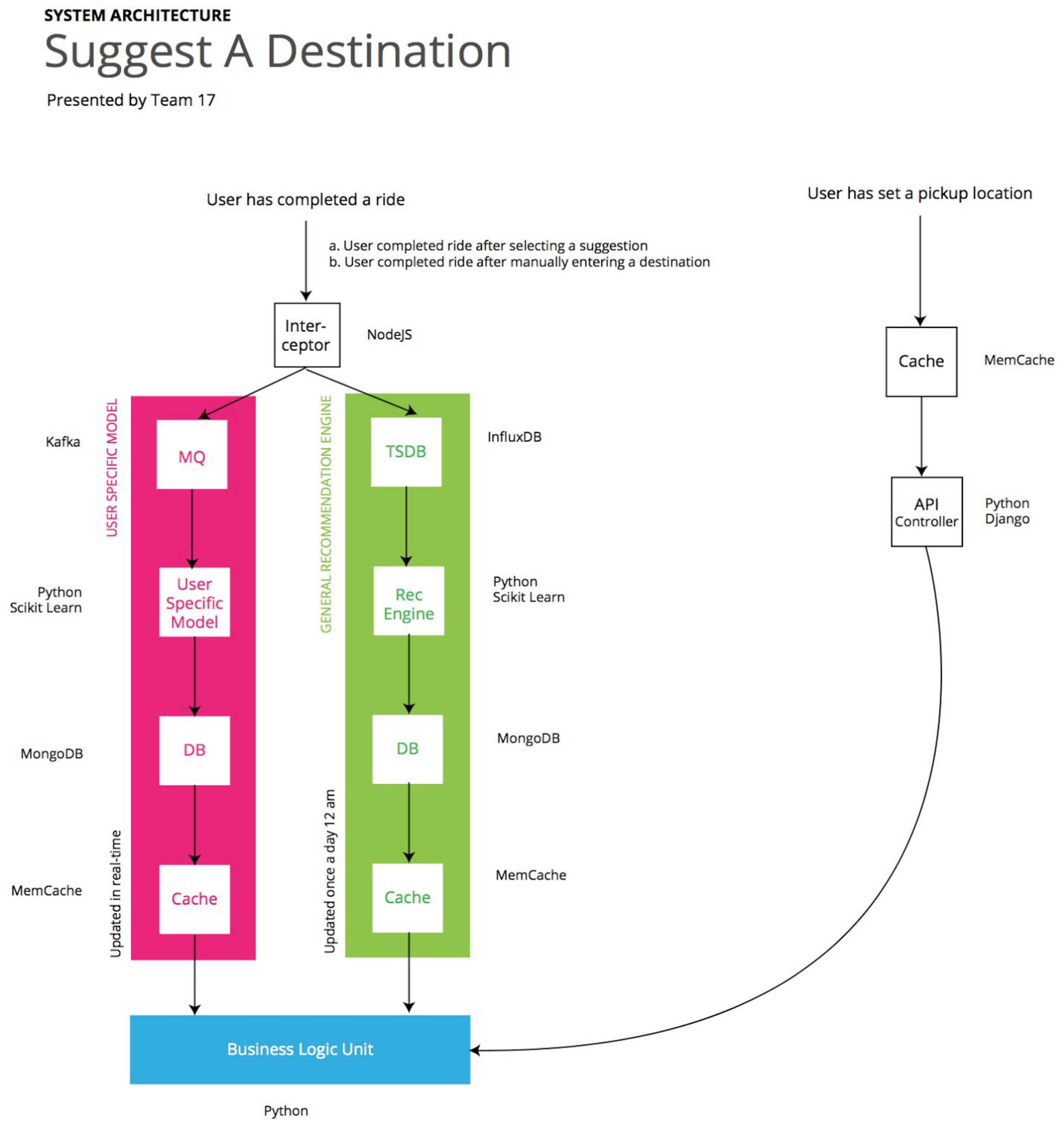

Building a recommendation engine

As soon as a ride completes, an event is sent to NodeJS based interceptor. Reason of choosing NodeJS is relative ease in scaling it for handling high traffic rates using a load balancer on front and horizontally scaling the server nodes. The message then ends up in a message queue for further processing. Purpose of placing a message queue between event emitter and the consumer is to provide scalability. This message informs the recommendation service that it needs to update the trained user recommendation model based on the newly received data. Any stable message queuing system like Kafka or Rabbit MQ can be utilized for this purpose.

One (or more as required) worker nodes continuously polls the message queue(s). On finding a new message it fetches the existing model of associated user and update it based on newly received data.

The user specific recommendation engine, the complete pink box, can be scaled horizontally according to need as each box is independent of each other's operation.

Machine Learning Model

At this point it's hard to say exactly what ML algorithm would be suitable here. Choice of ML algorithm to use depends upon following factors

1. Accuracy
2. Time for training and prediction
3. Does it need retraining after each data addition?
4. Available features

We believe neural networks can be a good choice here because of the fact that, unlike Random Forest and other similar algorithms, we do not need to retrain complete model once new data comes.

Showing Recommendations

When a user selects a pickup location from Careem app, app sends a REST request in background with following parameters, {user_id, timestamp, pickup_location} to fetch recommendations. Server responds with suggestions. Note that it's not necessary that server always responds with some recommendation. If confidence of the prediction is not high enough server can send an empty response or server can use the General Recommendation Engine, as shown in the architecture diagram. Data returned from the server would be shown to user as soon as user goes onto Drop Off page. Calling REST service in the background in an async manner would help in giving user wait-free user experience.

Purpose of adding a caching layer between mobile app and API layer is to improve performance. This can be a memcache server. Idea is to avoid calling backend api controller and business logic unit if user is trying the same pickup location.