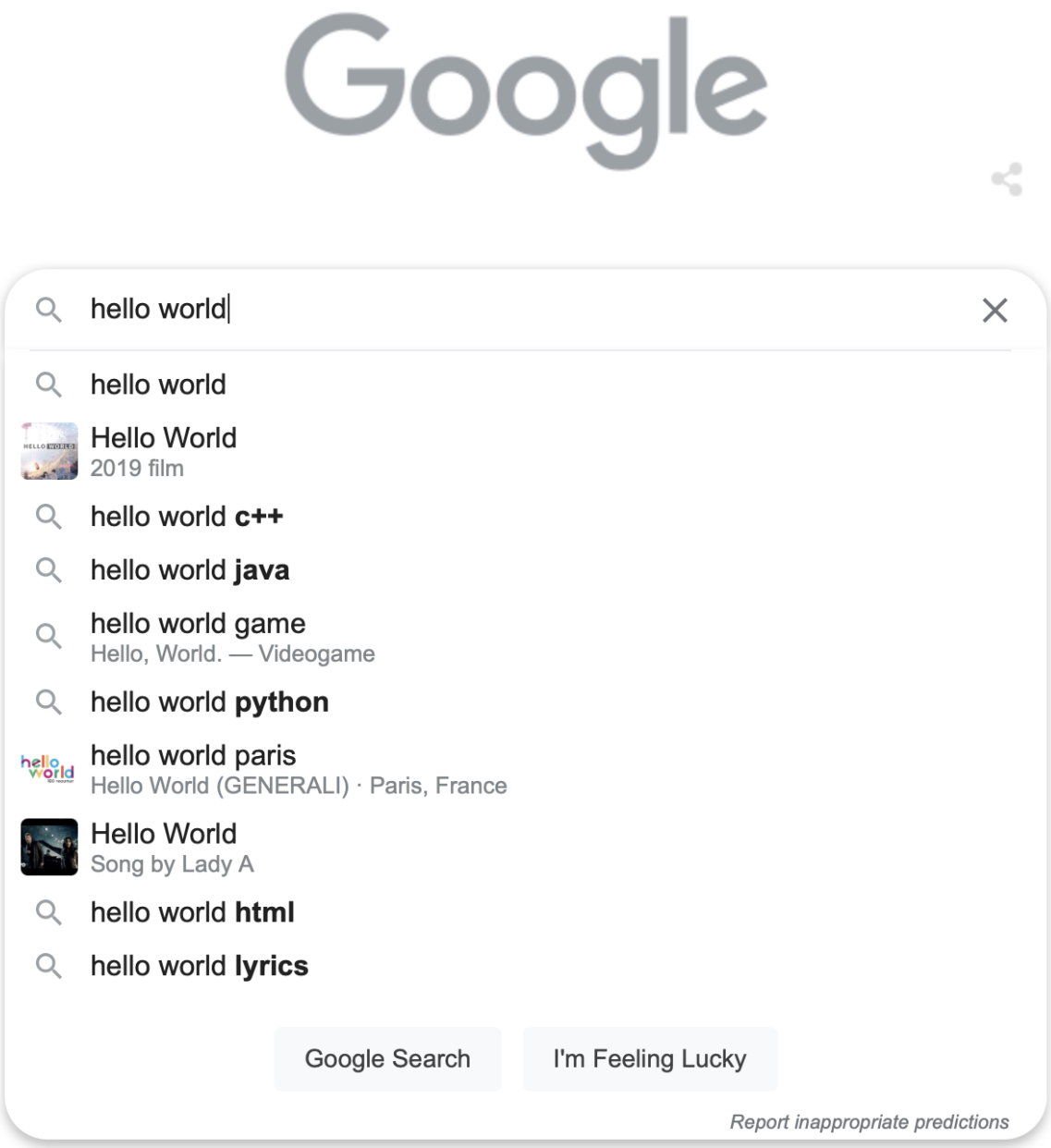
#systemDesign #programming

Search Autocomplete



High Level Features

- Real time suggestions as user is typing
- Return 10 suggestions per prefix
- Suggestions based on frequency

- Response time < 100ms
- Search engine, rather than real-time apps like Twitter

API Endpoints

GET `/suggestions?q=<prefix>`

Example

Let's say user types: `dog`

API requests made:

GET `/suggestions?q=d`
GET `/suggestions?q=do`
GET `/suggestions?q=dog`

For every request, 10 suggestions are returned as the user is typing.

Naive Solution

We have a Relational Table with frequency of every search string.

query_string	frequency
burger	1500
cheese	700
cheque	600
burlington	300

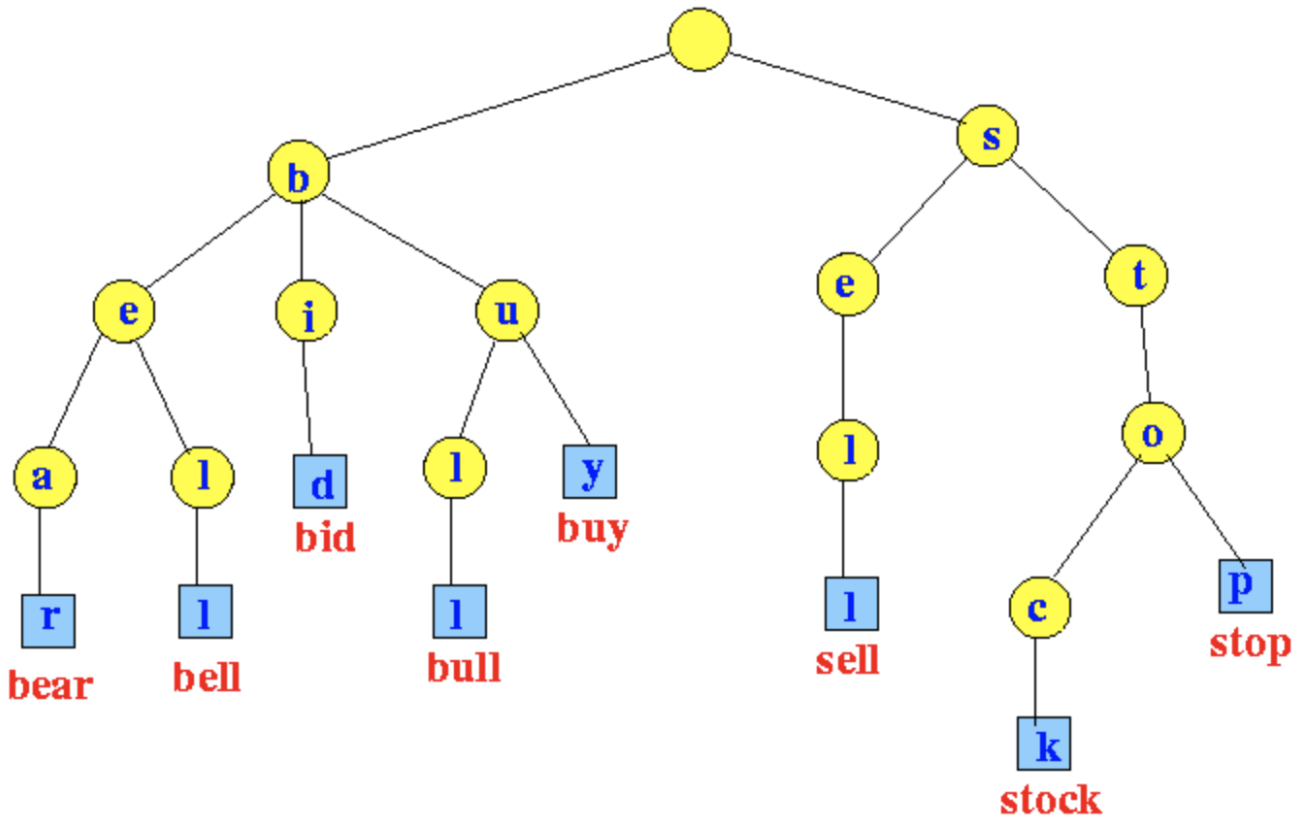
To get suggestions, we use a simple SQL query:

```
SELECT *
FROM search_history
WHERE query_string LIKE "<prefix>%";
```

This is fine for **small datasets** and **low volume** traffic.
However, it **won't scale**.

Let's work on the more optimized solution now:

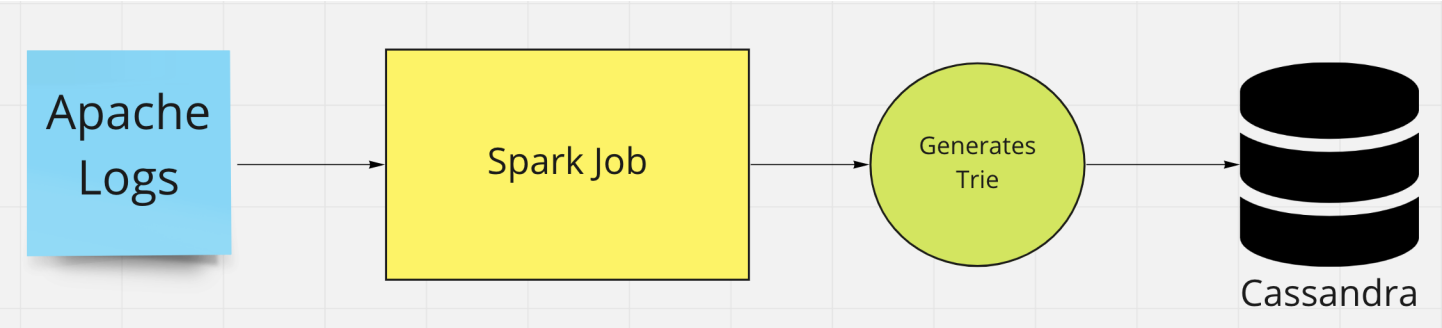
Trie Data Structure



- Generate Trie from existing data periodically
- Each node also contains all the suggestions
    - At Node "e"
        - bear
        - bell
    - At Node "sto"
        - stock
        - stop
- For smaller dataset, you can store Trie in-memory
- Store Trie in a database
    - Create a hashmap from it and store it in
        - MongoDB

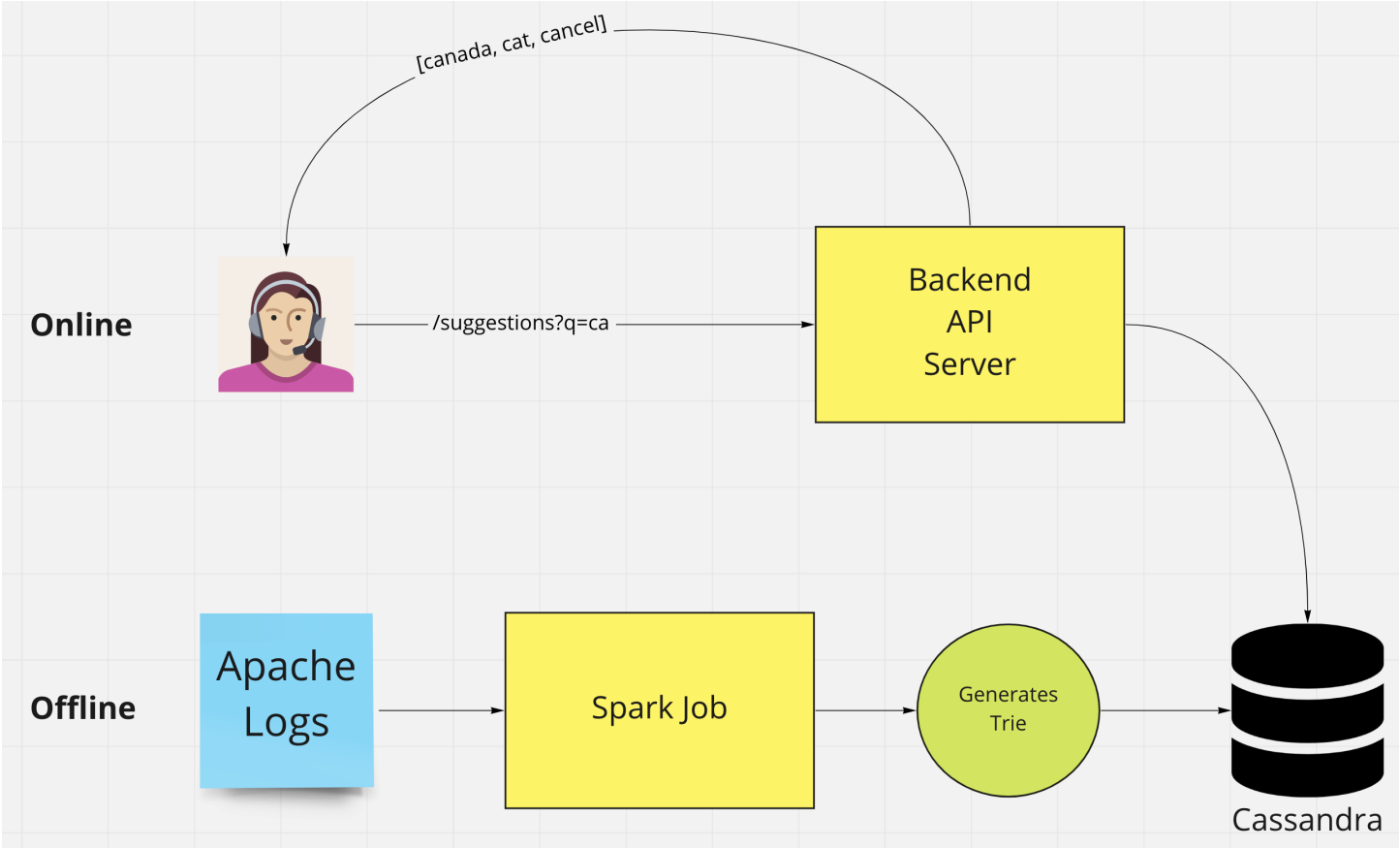
    - Cassandra
- Cache most common search strings if required

When Do You Generate the Trie?



- Run pipeline weekly
- Parses all Apache logs
- Computes frequency of different search strings
- Computes Trie
- Read Trie
- Write to Cassandra as key-value

Full Architecture


[canada, cat, cancel]
Online
/suggestions?q=ca
Backend
API
Server
Offline
Apache
Logs
Spark Job
Generates
Trie
Cassandra