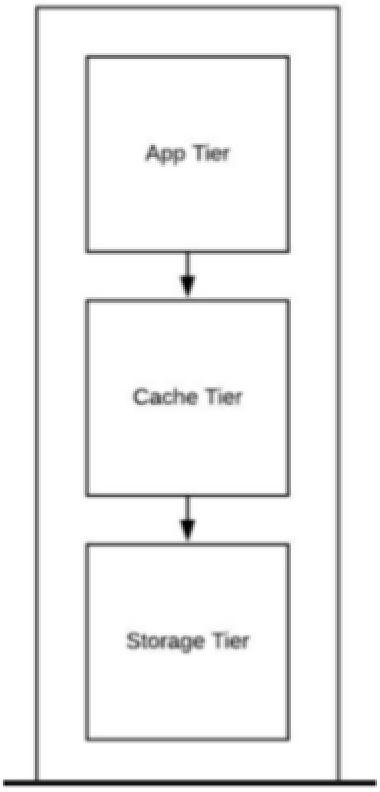
Step 1: Gather the requirements
Functional Requirements:
1. Real-time Notification to the billionth search query (incognito search)
2. Notify on WebSocket for session

Design Constraints:
1. 70k search/sec
2. Search country redirection
3. Geo-distributed

Step 2: Define microservices

There will be only one microservice. Counter Service
API: notify_billionth_user();
Data Center: USA(8), Asia(2), Europe(4), South America(1)
Source: https://www.google.com/about/datacenters/locations/

Step 3: Draw the logical architecture



Step 4: Deep dive into the microservice
Algorithm/Approach:

1. CRDT Counter:
   Counter: {DC_USA_0_Val_0, DC_USA_0_VAL_1,...,DC_ASIA_0_VAL_0}
   - Row Oriented
   - Each search app service assigned a valve to be updated in cache/db by the Cluster Manager.
   - Grow-only counter, use max for sum
   - Global broadcast needed, eventual consistent.
   - Use a CRDT DB (Redis, COSMODB) to store the {Search Query Number, VALVE} list
   - DC Aggregator app server uses its cache
   - A queue and queue consumer between the search DB and Aggregator cache ensures neartime processing
2. COLUMNAR:

DC_USA_0_VAL_0: VALVE| DC_USA_0_VAL_1: VALVE
DC_USA_0_VAL_2: VALVE| DC_ASIA_0_VAL_1: VALVE

- Each seach app server is assigned a value to be updated by the the Cluster Manager.
- After updating the Valve in the cache {DC_Counter_#_Val#: Valve} is updated in a db replicated within a data center.
- Updates are faster and less chatty.
- DC Aggregator aggregates at DC level.
- Global aggregator aggregates from DC Aggregator and notify if billionth number.

Step 5: Identify the need for scale
Single Node: one atomic uint64_t. To keep count has data hotspot.

Scale for:

1. Storage: No
2. Throughput: No
3. Hotspot: Yes
4. API Parallelisation: No
5. Availability: Yes
6. Geo-Distribution: Yes

Capacity Estimates: 70k counter increment per second
Sharded by 1 counter per data center.
70K/15 counters = 5k counter increment/second (Still a hotspot)

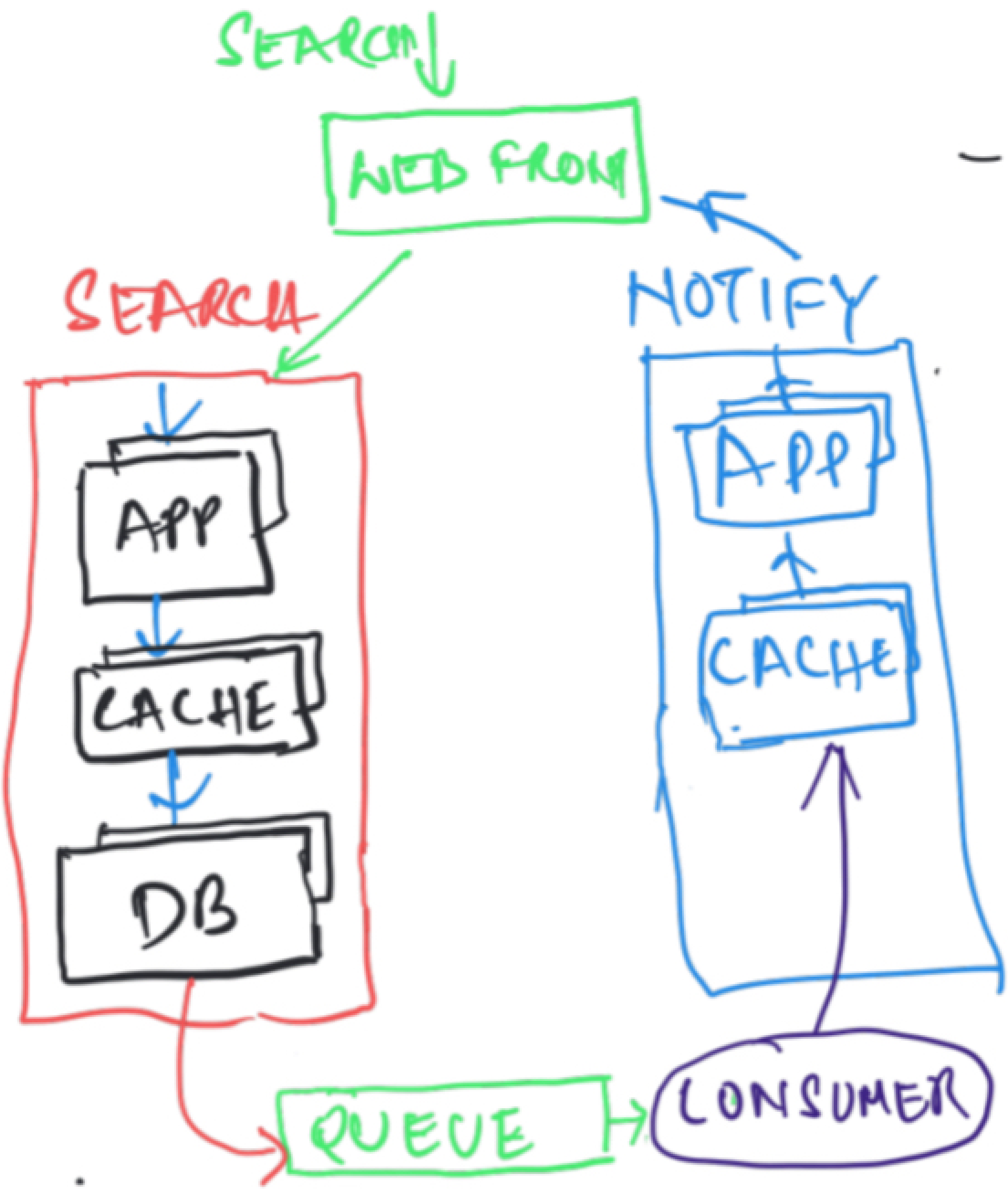
Key	Value per Data Center
Search	DC_USA_0, DC_USA_1,...,DC_USA_7,DC_Asia_0,..
Query Number	DC_Asia_1,DC_Europe_0, DC_Europe_3,...

Shard the counter within the data center:
5k counter updates per second => Shard each counter into 10 valves. So, 500 counter updates/sec.

Each Valve = uint64_t

Key	Value per Data Center Sharded
Search	DC_USA_0_Val_0, DC_USA_1_Val_1,...,DC_USA_7,DC_Asia_0,..
Query Number	DC_Asia_1_val_9,DC_Europe_0_Val_1, DC_Europe_3_Val_5,...

15 Data Centers * 10 Valves each = 150 unit64_t => 1200 Bytes
Step 6: Propose the distributed architecture



Architecture:

- Write-Back cache for Search Service
- Aggregator always sums from cache
- Search Service persists to DB to the updated counter values.
- A queue and queue consumer between Seacrh DB and Aggregator cache ensures nearline aggregator cache updates

- Global Aggregator uses a queue and queue consumer to gather count values from individual DC Aggregators and calculate billionth number.
- Notification is neartime, so if user exists browser before aggregator notifies he/she may miss the notification.
- When we reach the last 1% of the billionth request, all search queries are redirected to the global aggregator so that we can do strictly consistent counting for the billionth and notify the end user that he/she is the billionth user.

Storage: Row-Oriented: Only a few columns

Moving towards CP:

1. Quorum-based writes or master-slave
2. CP preferred since nearline accurate notification to the billionth query needed.
3. Persistent writes neeed for recovery
4. DB writes are needed for recovery
5. With columnar shareded counters, DB writes are around 500QPS. Typical DB standard server gives around 200QPS. So may need more shards for counter.
6. CRDT may not be able to provide the required DB write QPS. However, in CRDT data center specific aggregator can find the billionth and global aggregator may not be needed.