High load RecSys

Vladislav Goncharenko

Materials by Vadim Kokhtev

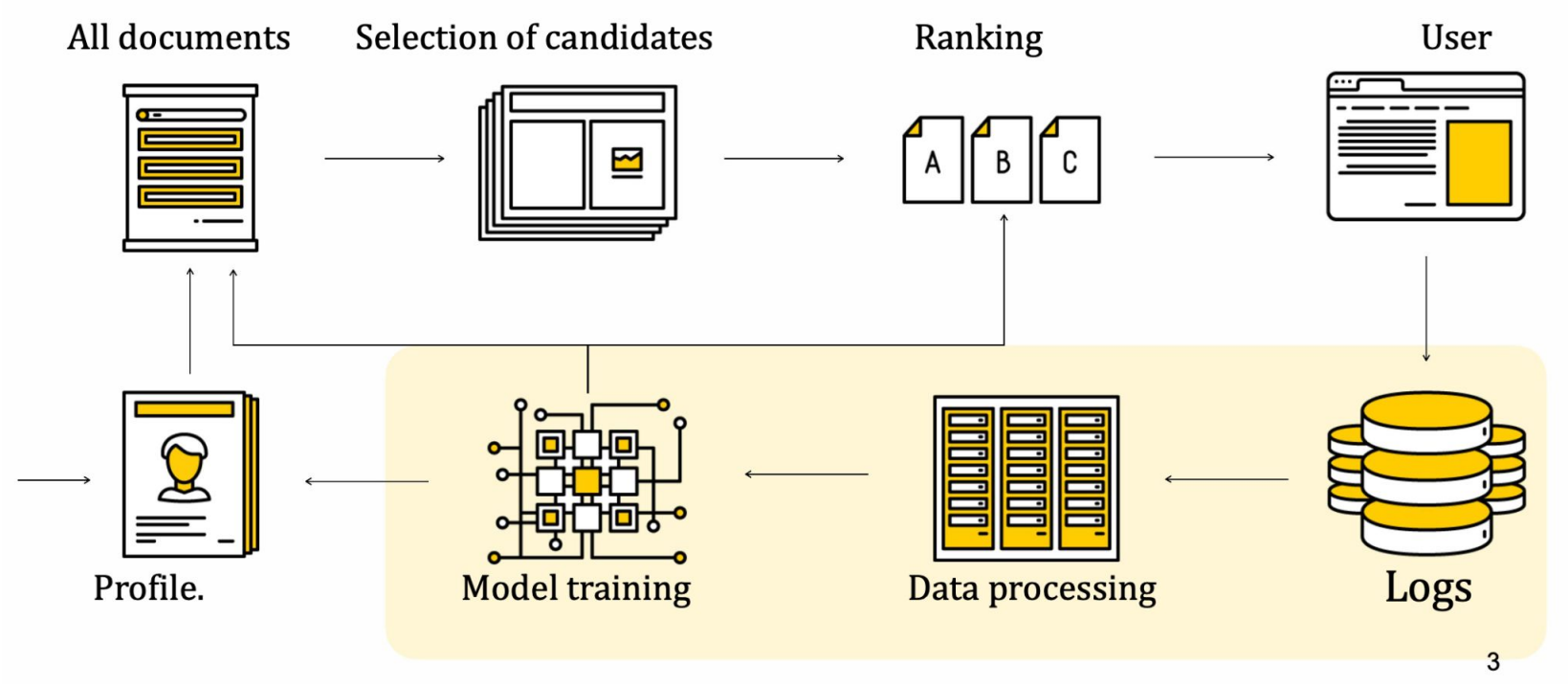
MIPT,
spring 2024

Typical service

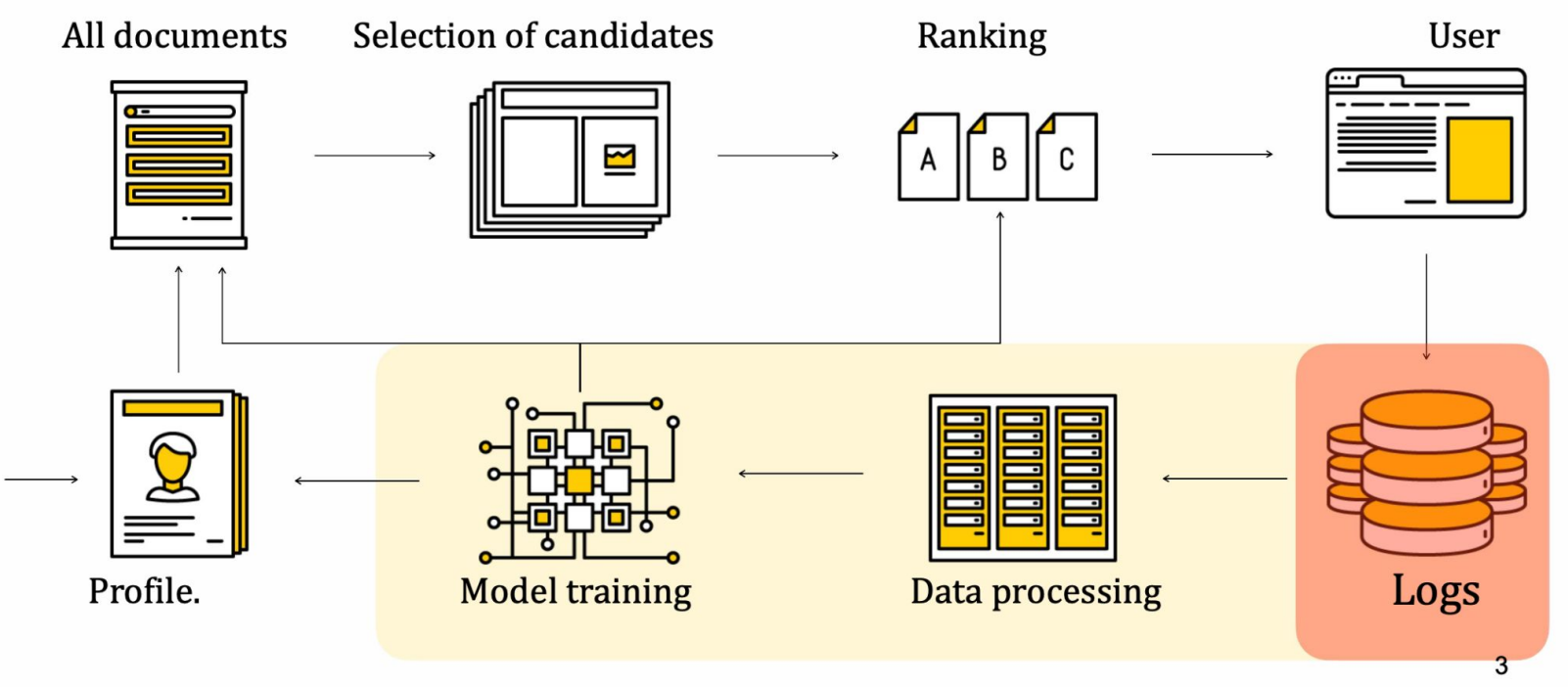
- 10^7 documents
- 10^6 users per day
- 1-10k requests per second (RPS) for recommendations

What could possibly go wrong? 😅

The real system



Logging


All documents
Selection of candidates
Ranking
User
A
B
C
Profile.
Model training
Data processing
Logs

Logging

- To train algorithms, information about user actions is stored in a structured form
- At a minimum, you will need a bundle of "user-item-rating-timestamp"
- Logs should be stored forever because this is unique information about your users

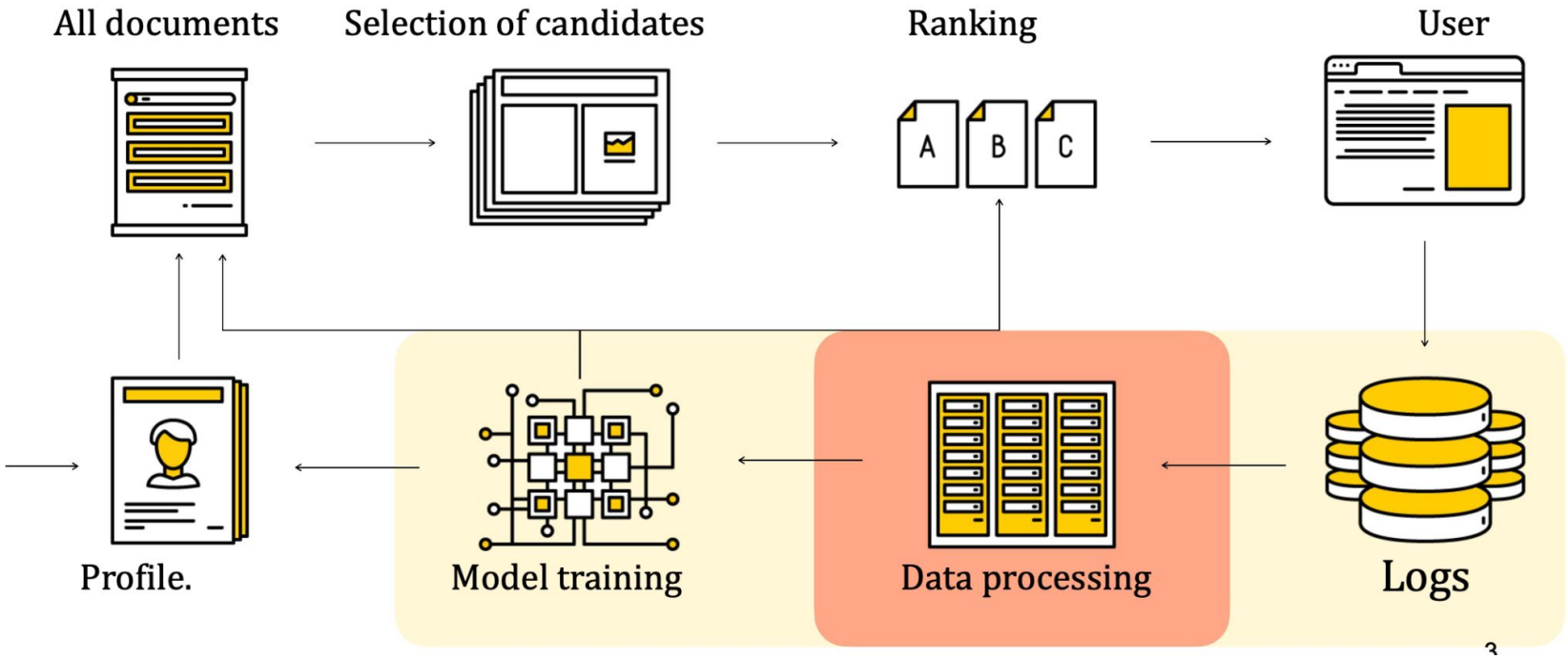
_version	String *optional*
client_info	**Struct** *optional* client_type : String client_version : String
event_id	String *optional*
event_type	String *optional*

Data processing

girafe
ai

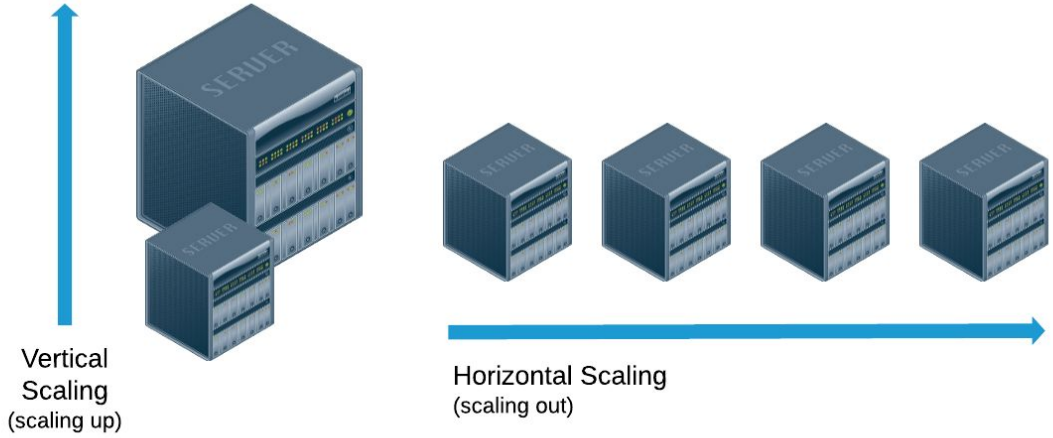
01

Data processing



Scalability

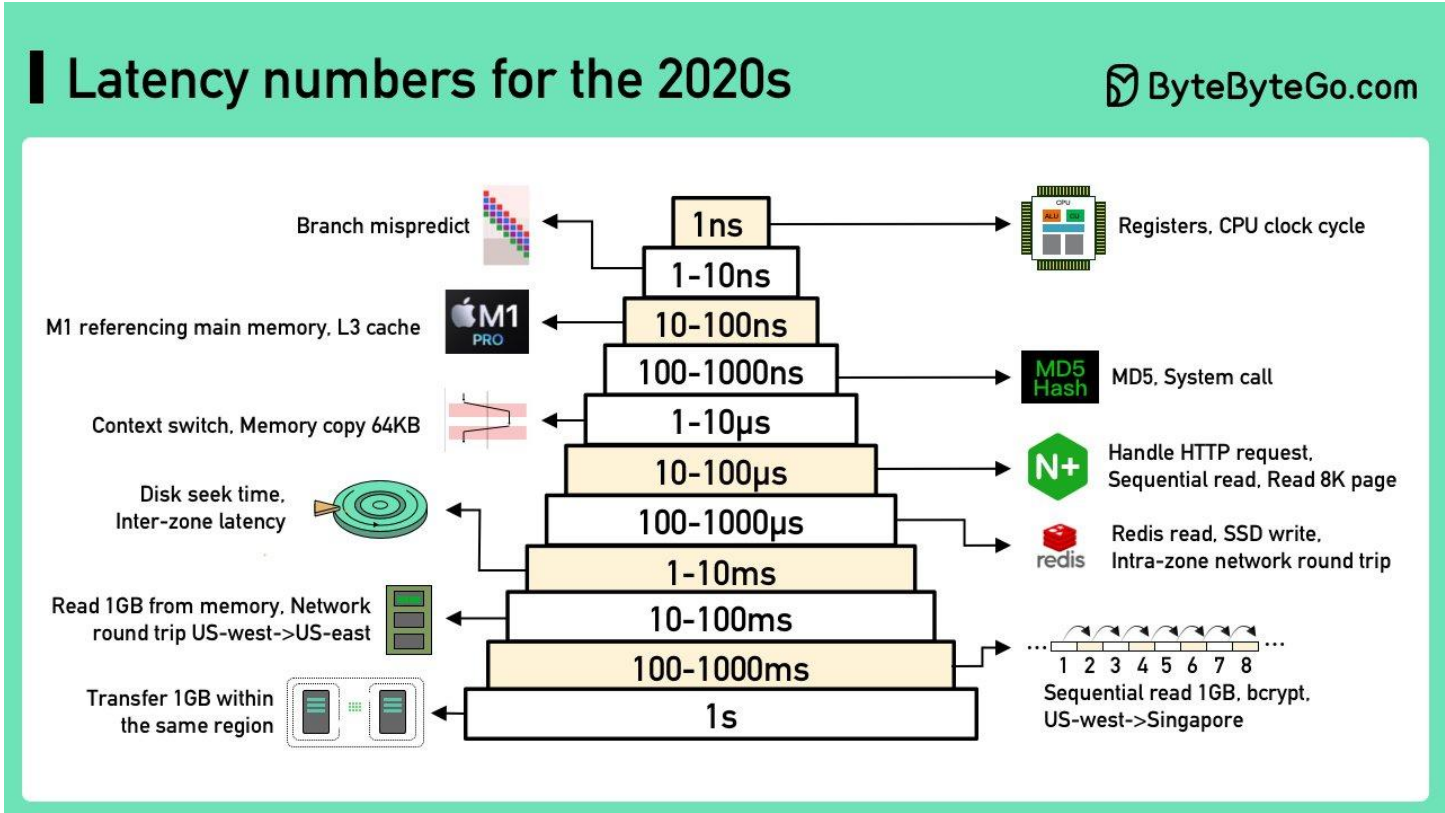
- When there was less data and they were more structured, they could be put into a relational database, having thought out the data schema and a list of possible queries to them in advance
- To speed up processing, we need a more powerful server - **vertical scalability**
- In the MapReduce paradigm, more average servers are needed to speed up processing (**horizontal scalability**), which is much cheaper



Data Locality

Ulrich Drepper, What every programmer should know about memory (2007)

Stackoverflow discussion



Data Locality

- Blocks of a huge file are stored on different machines
- We can process blocks locally on the machines where they are stored
  (fast read from local disk)
- If it is possible to process blocks independently in the task, then it is possible to achieve ideal scalability (embarrassingly parallel)
- For example, the task of filtering file lines scales perfectly. All lines are independently checked by the predicate, the more machines, the faster it will work

MapReduce

girafe ai

02

Word frequencies counting problem

Texts from all over the Internet are stored in a distributed system.

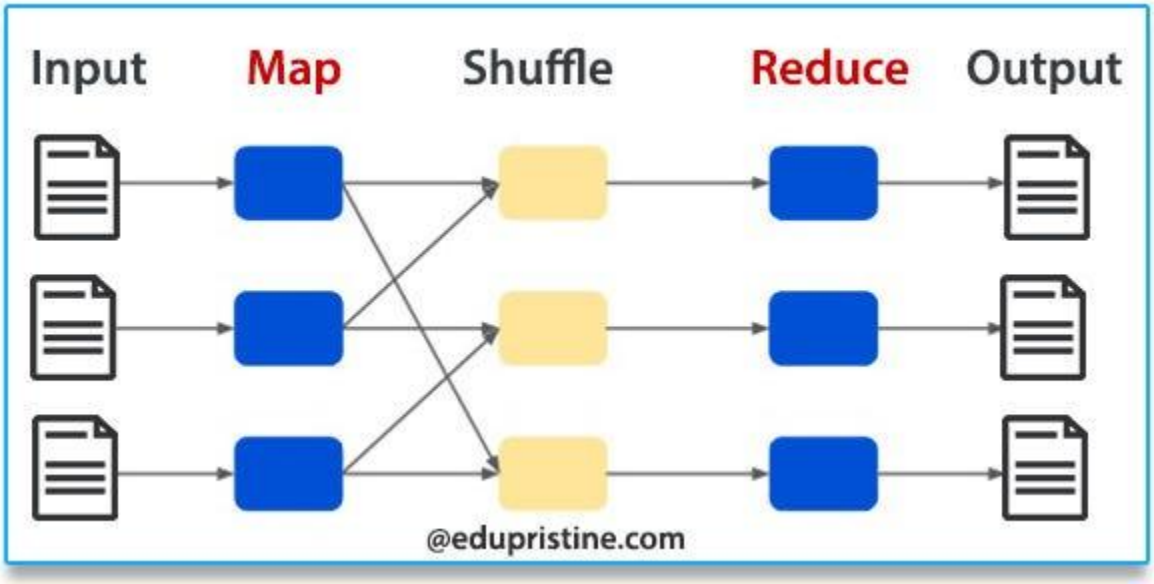
We are interested to know the frequencies of all words (counters for tf-idf).

Solution is MapReduce paradigm

How to solve:

- for each block, we calculate the frequency of words in it (ideally scaled)
- add up the frequencies for all blocks

MapReduce main idea



MapReduce solution

Map:

(K1, V1) ⇨ List(K2, V2)

(line number, “cat sleeps”) ⇨ [(“cat”, 1), (“sleep”, 1)]

Shuffle:

Keys are splitted by **hash(key) % N** on **N** parts

Each parts gets sorted by **key** (independently)

Reduce:

(K2, List(V2)) ⇨ List(K3, V3)

(“cat”, (1, 1, 1)) ⇨ [(“cat”, 3)]

Reliability of operations

- If the mapper is lost, we can restart the task only for its blocks
- If the reducer is lost, we re-collect data from all mappers only for its hash value

Popular track example

UserId	TrackId	AlbumId
11123	4521	842
14322	3593	957
...	...	...

Map: (K1, V1) ⇨ List(K2, V2)

Reduce: (K2, List(V2)) ⇨ List(K3, V3)

Let's take the logs of listening to music.
How do we find the most popular track for each album?

M: #, (user, track, album) ⇨ (album, track), 1

R: (album, track), (1,1,…) ⇨ (album, track), count

M: (album, track), count ⇨ album, (track, count)

R: album, tracks ⇨ album, most popular track

Example:
Item-based CF concurrency

girafe
ai

03

Example: Item-based CF concurrency

- ***n*** users, ***m*** items, ***n*** ~ 10^6, m ~ 10^7
- It is necessary to calculate in advance the Item-Item similarity

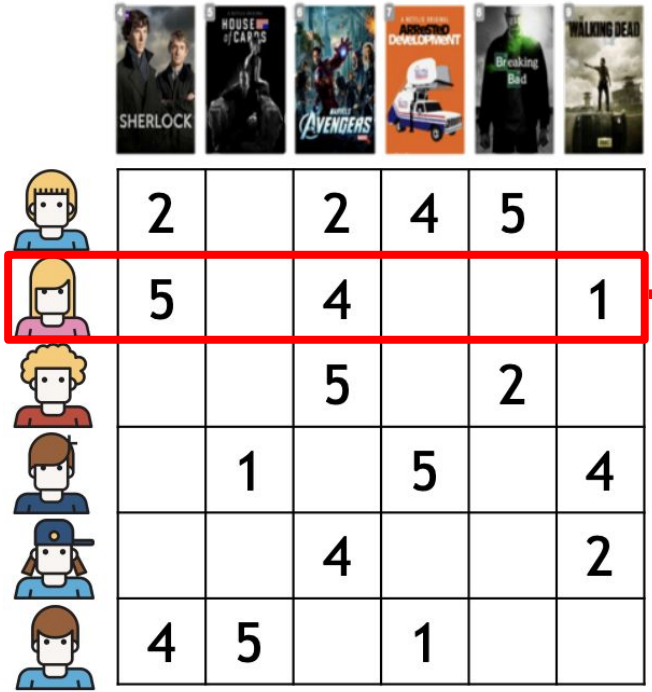
$$s(i,\ j) = \frac{\sum_{u \in U} (r_{ui} - r_u)(r_{uj} - r_u)}{\sqrt{\sum_{u \in U} (r_{ui} - r_u)^2} \sqrt{\sum_{u \in U} (r_{uj} - r_u)^2}}$$

adjusted cosine similarity

- Naive approach – ***O(m²n)***
- The evaluation matrix is very sparse, can it be better?
- The contribution to the similarity of the two products is non-zero only from users who rated both of these products

Inverted index

Update the calculations once a day using the inverted index



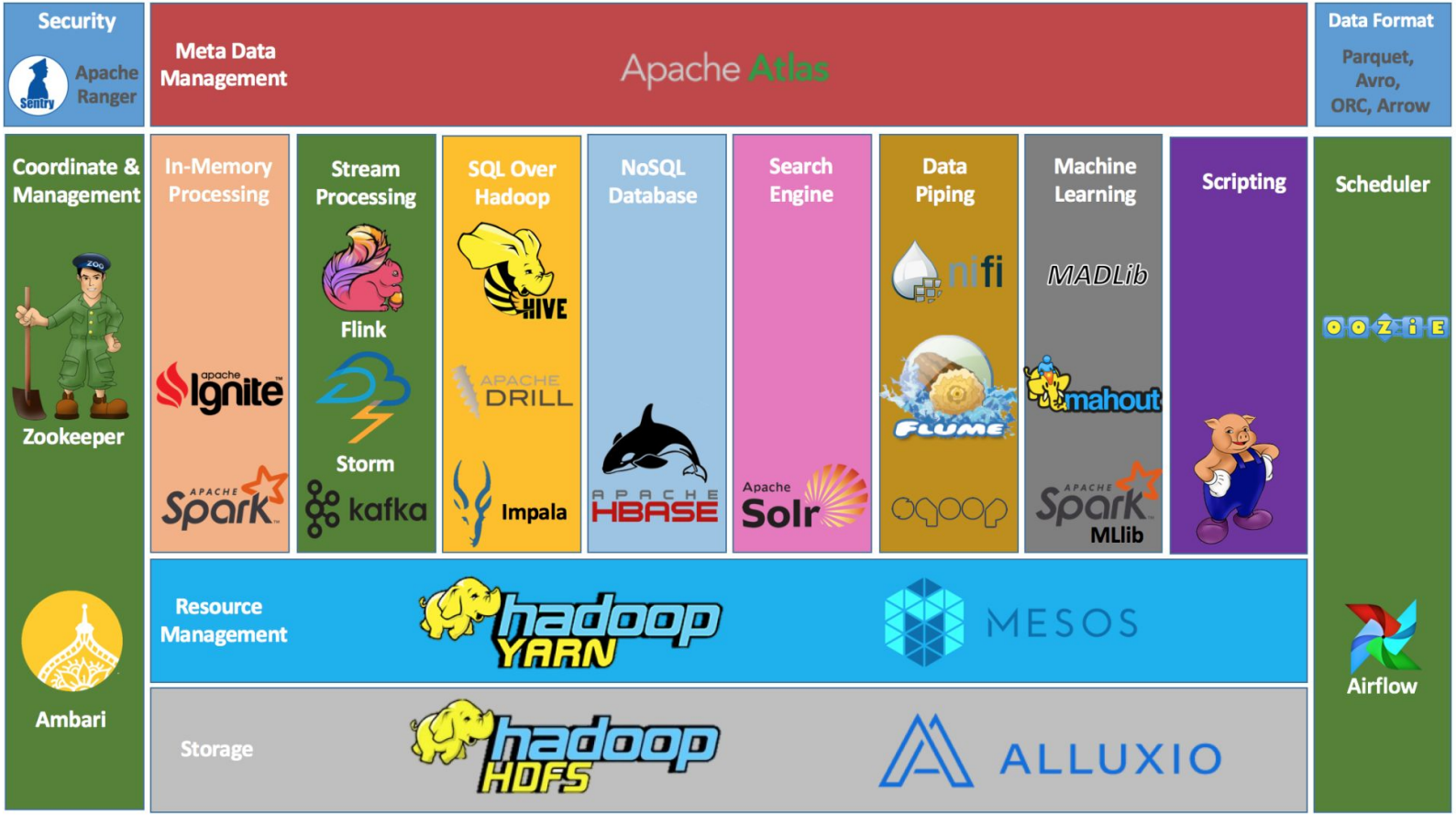
$$s(i,\ j) = \frac{\sum_{u \in U} (r_{ui} - r_u)(r_{uj} - r_u)}{\dots}$$

Map step:

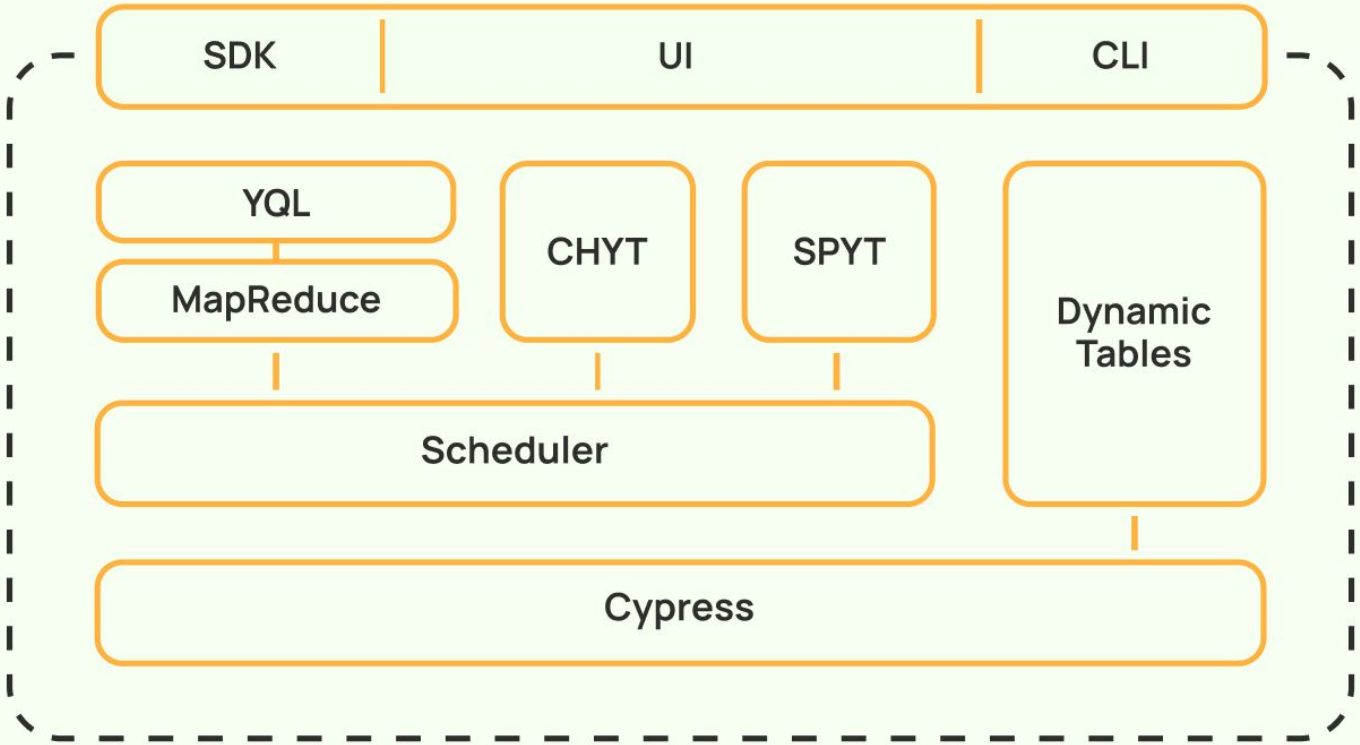
(1, 3) -> (5 - 3.3) * (4 - 3.3)
(1, 6) -> (5 - 3.3) * (1 - 3.3)
(3, 6) -> (4 - 3.3) * (1 - 3.3)

Reduce step: summing up by the key

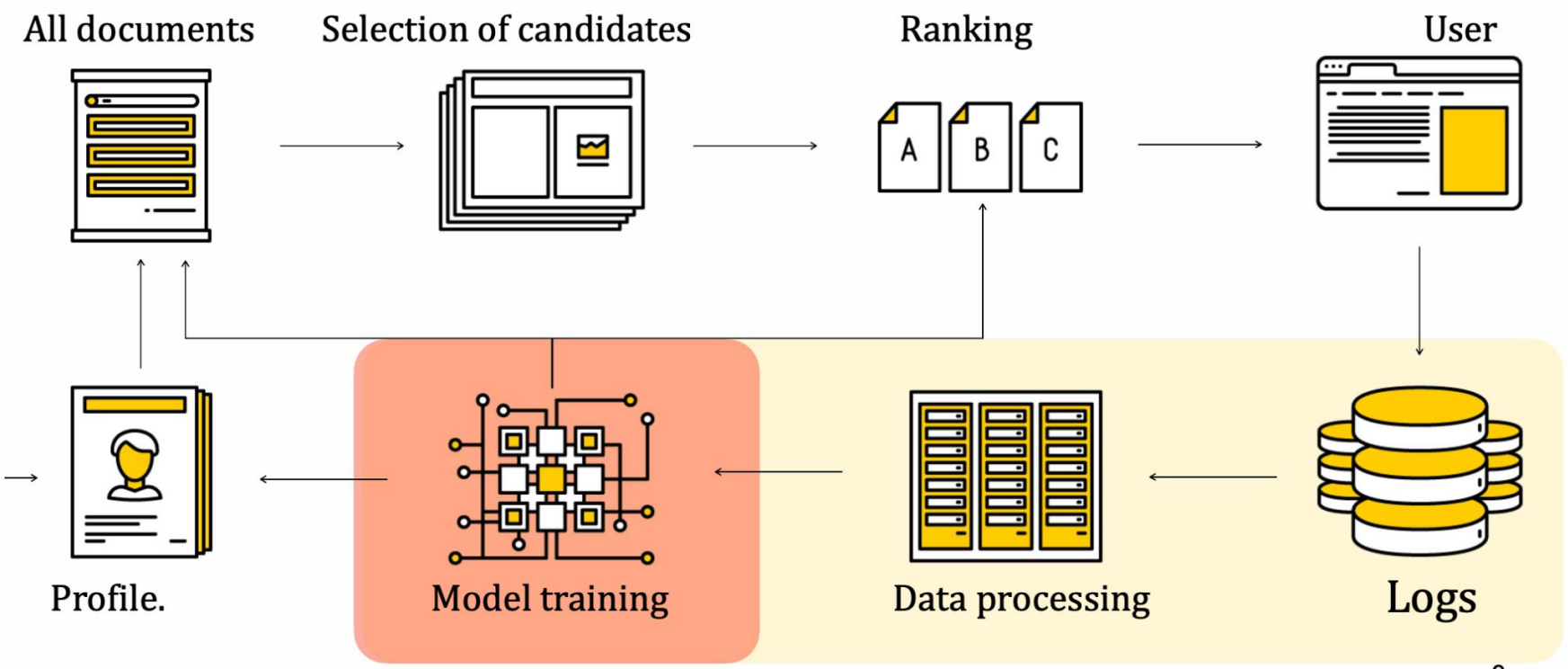
Hadoop: open source MapReduce



YTsaurus: one more MapReduce



Model training

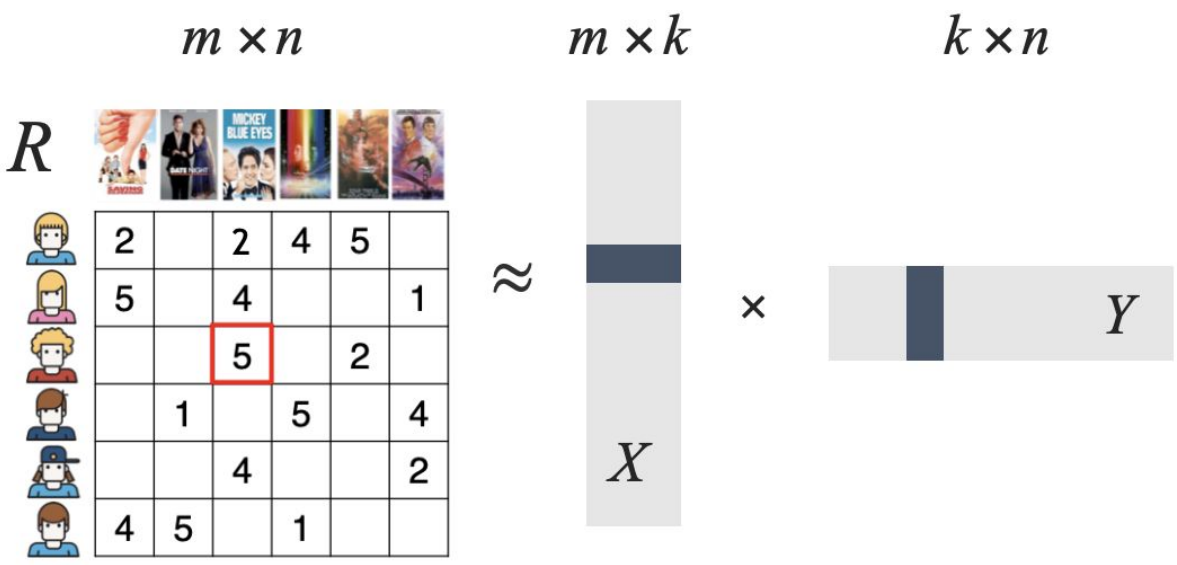


Alternating Least Squares (ALS)

girafe ai

04

Alternating Least Squares (ALS)



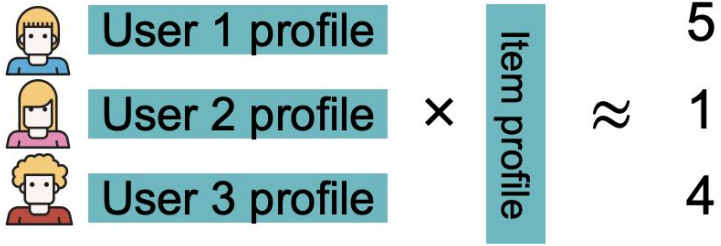
$$\min_{X,\,Y} \sum_{(u,\,i)\in R} \left(r_{ui} - x_u^T y_i\right)^2 + \lambda\left(\|x_u\|^2 + \|y_i\|^2\right)$$

Alternating Least Squares (ALS)

Initialize X and Y with random values

In the loop:

- We fix the matrix Y (products)
- Find the optimal matrix X (solve the ridge regression for each user)



$$\min_{X,Y} \sum_{(u,i)\in R} \left(r_{ui} - x_u^T y_i\right)^2 + \lambda\left(\|x_u\|^2 + \|y_i\|^2\right)$$

$$x_u = \left(\sum_{r_{ui}\in r_{u*}} y_i y_i^T + \lambda I_k\right)^{-1} \sum_{r_{ui}\in r_{u*}} r_{ui} y_i$$

- and vice versa

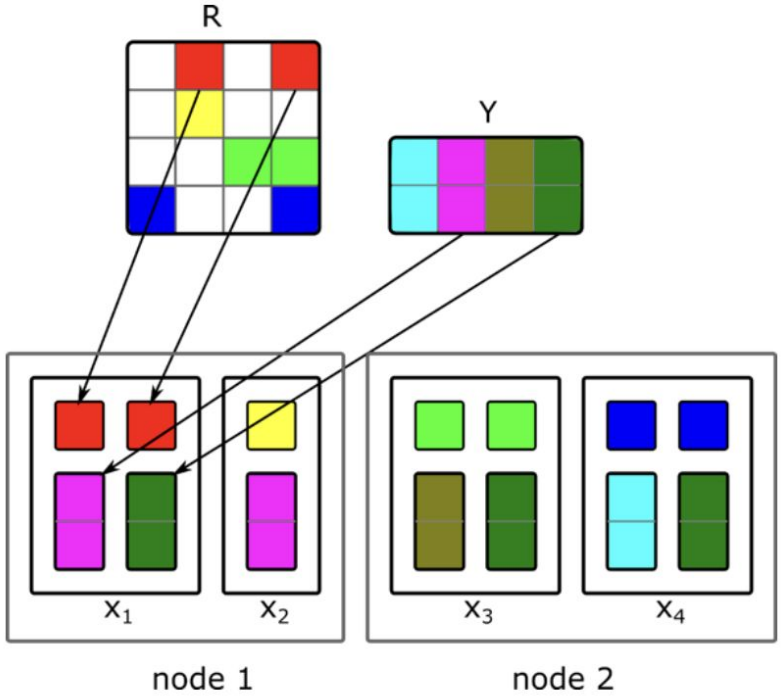
Naive Parallel ALS

- R = X@Y, update X

  Join R and Y by product → $(u,\ r_{ui},\ y_i)$

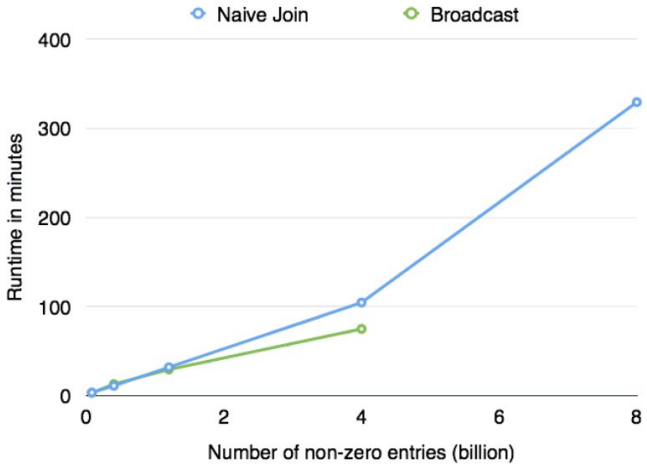
  Grouping by u

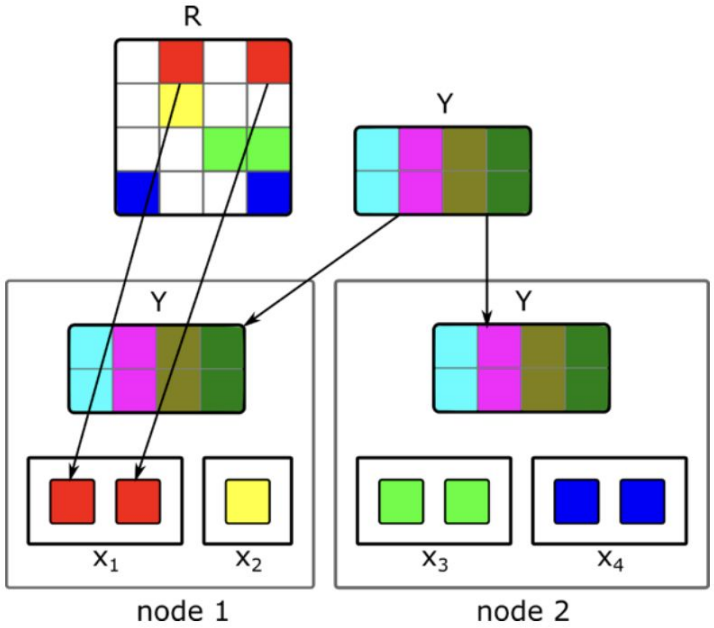
  On each machine we recalculate x_u
- The same vector y_i may be needed by several users on the machine, and will be sent several times



Implementation with broadcast Y

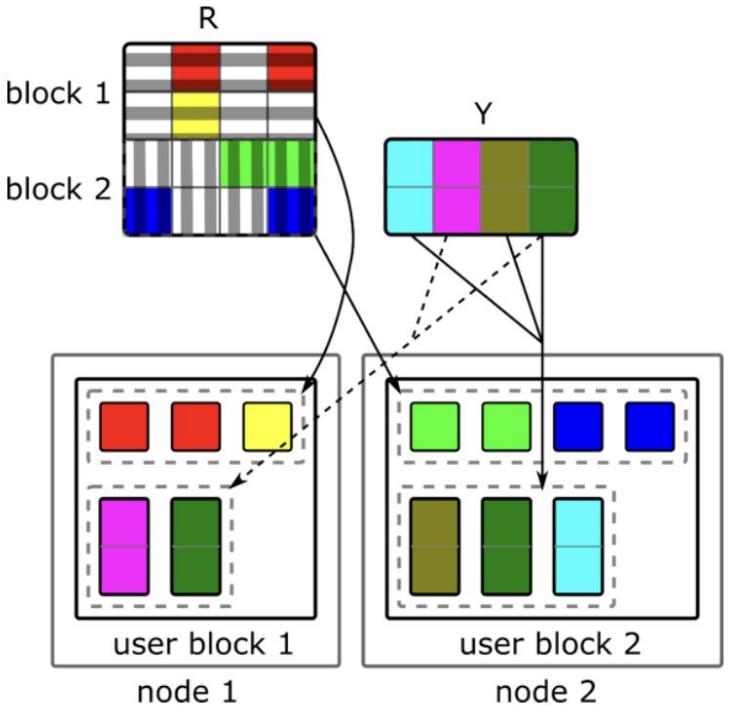
- Faster
- Has a scaling limit (Y must fit into memory)





Block ALS

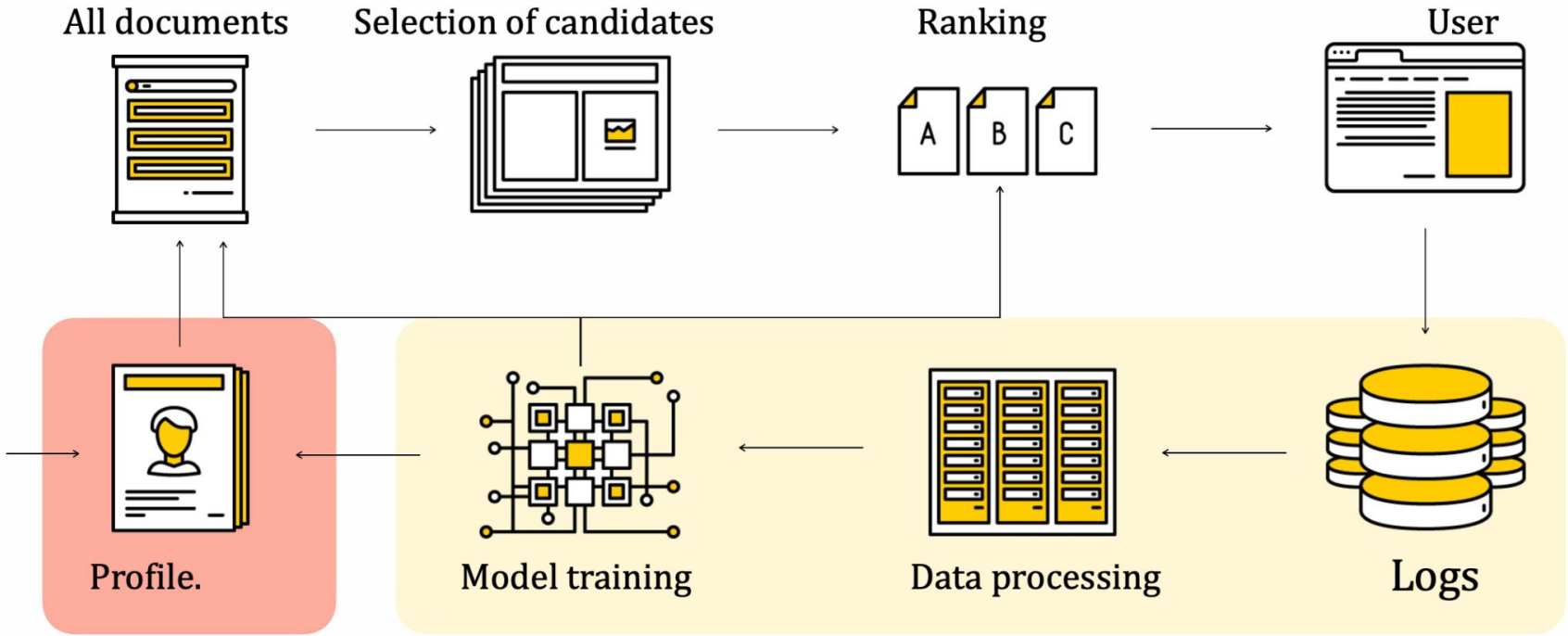
- Implemented in Spark ML
- Users are fighting for blocks, each block is processed on its own machine
- We calculate once what goods will be needed in each block (mapping is stored in memory)
- Only the necessary parts of the matrices in the machine memory are sent



Caching during training

- Recalculate statistics (but not models) of only updating users and objects
- We save the results of third-party services that are difficult to calculate

User Profile



Fast data storage

girafe
ai

05

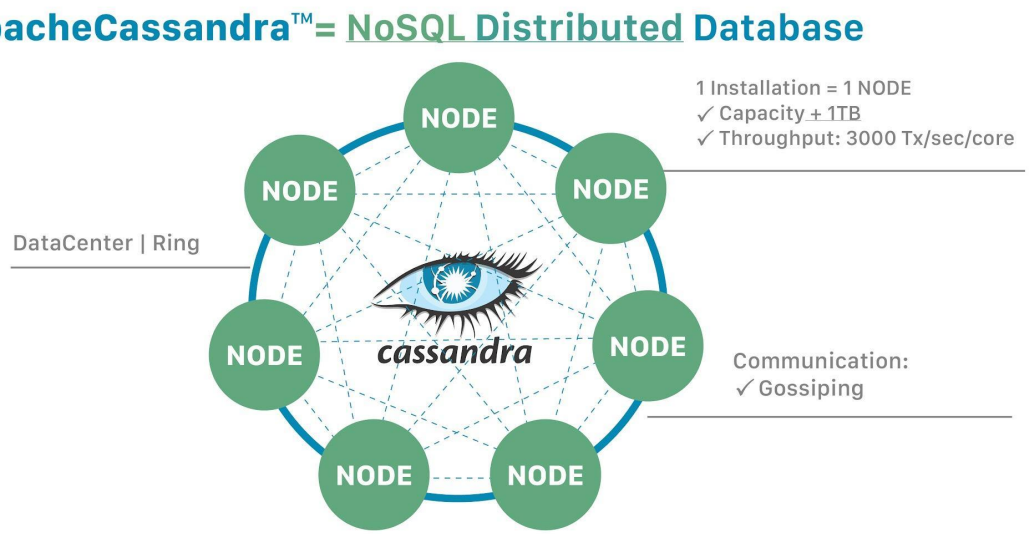
Fast data storage

Key-value

- Reading and writing for O(1)
- Eventual consistency

Examples:

- Apache Cassandra
- Amazon DynamoDB
- Redis

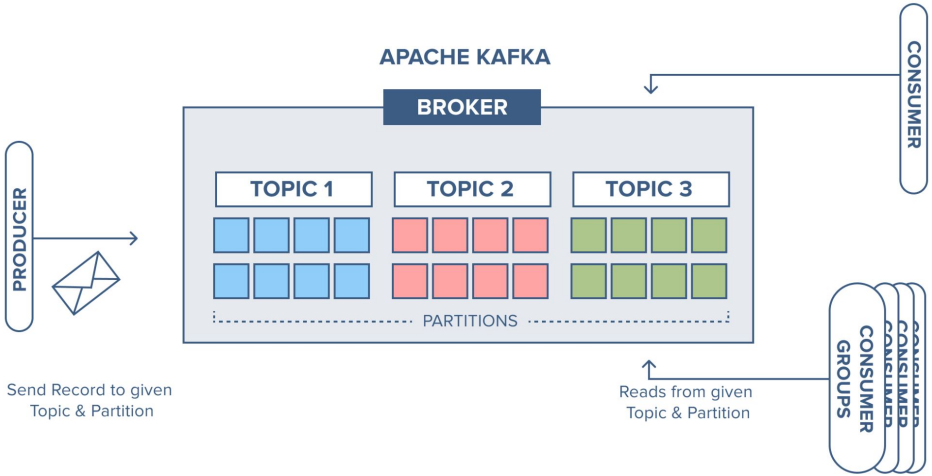


Near-realtime event handling

- We want to update user data as quickly as possible
- We update statistics, model results on the fly, reading updates from event queues

Example: writing user actions in Kafka

1. collect the user profile from the Key-Value response and events from Kafka
2. process statistics and do the ALS step every N minutes

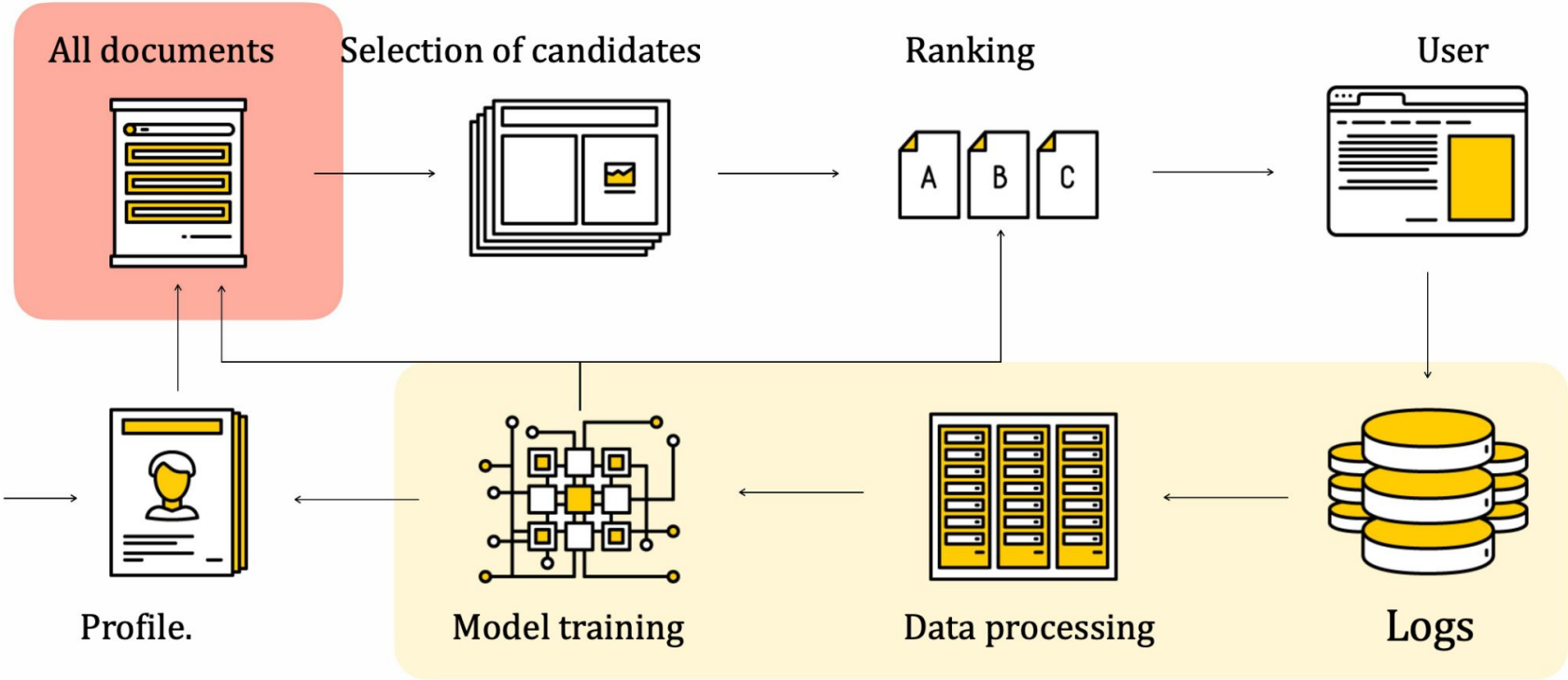


Document storage

girafe ai

06

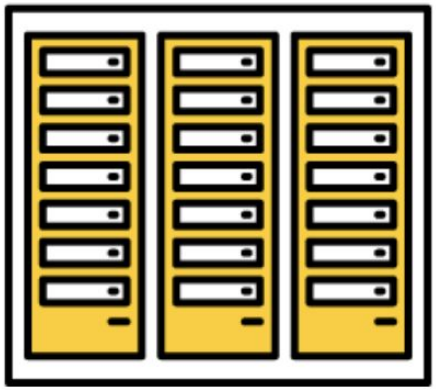
Document storage



Sharding

- One server has a limit on the number of documents stored
- Let's store disjoint sets of documents on different servers, and combine the results of responses

One shard

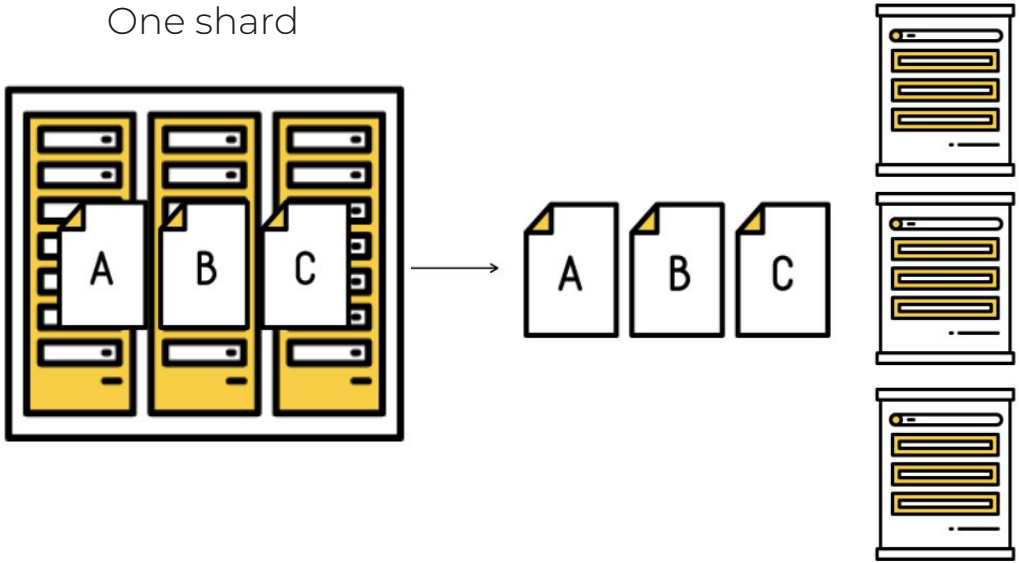


Sharding

- One server has a limit on the number of documents stored
- Let's store disjoint sets of documents on different servers, and combine the results of responses

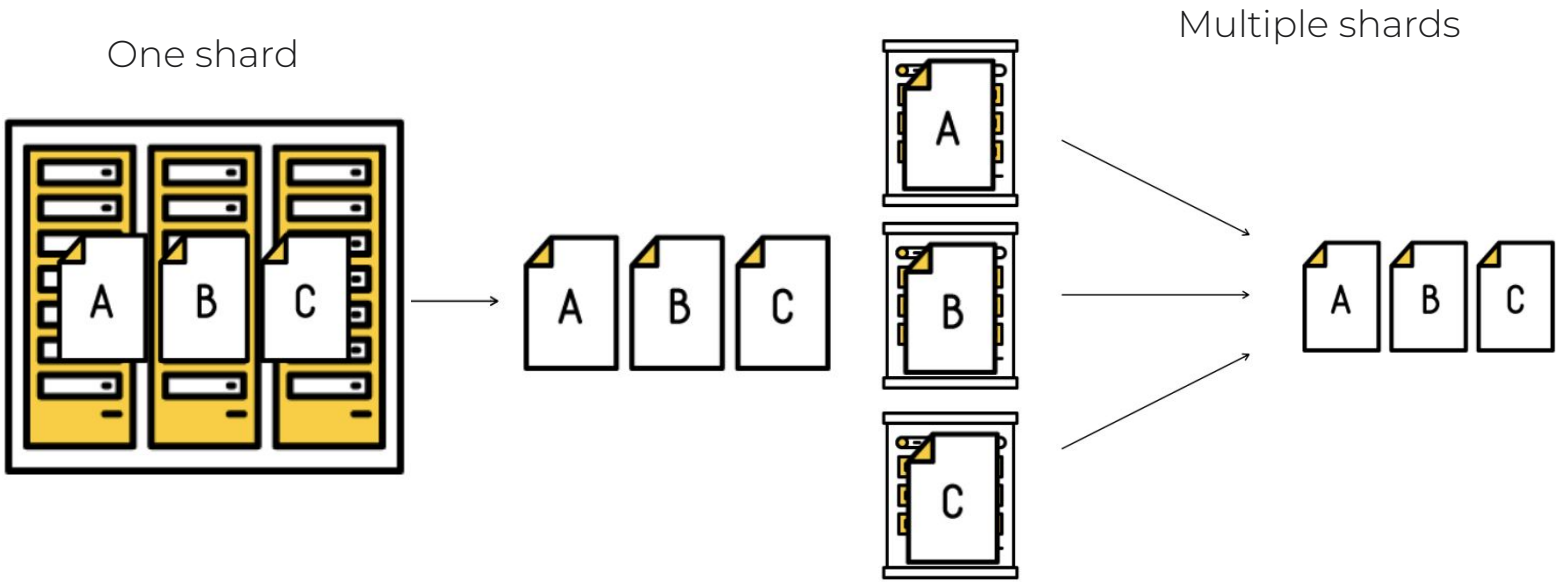
One shard

Multiple shards

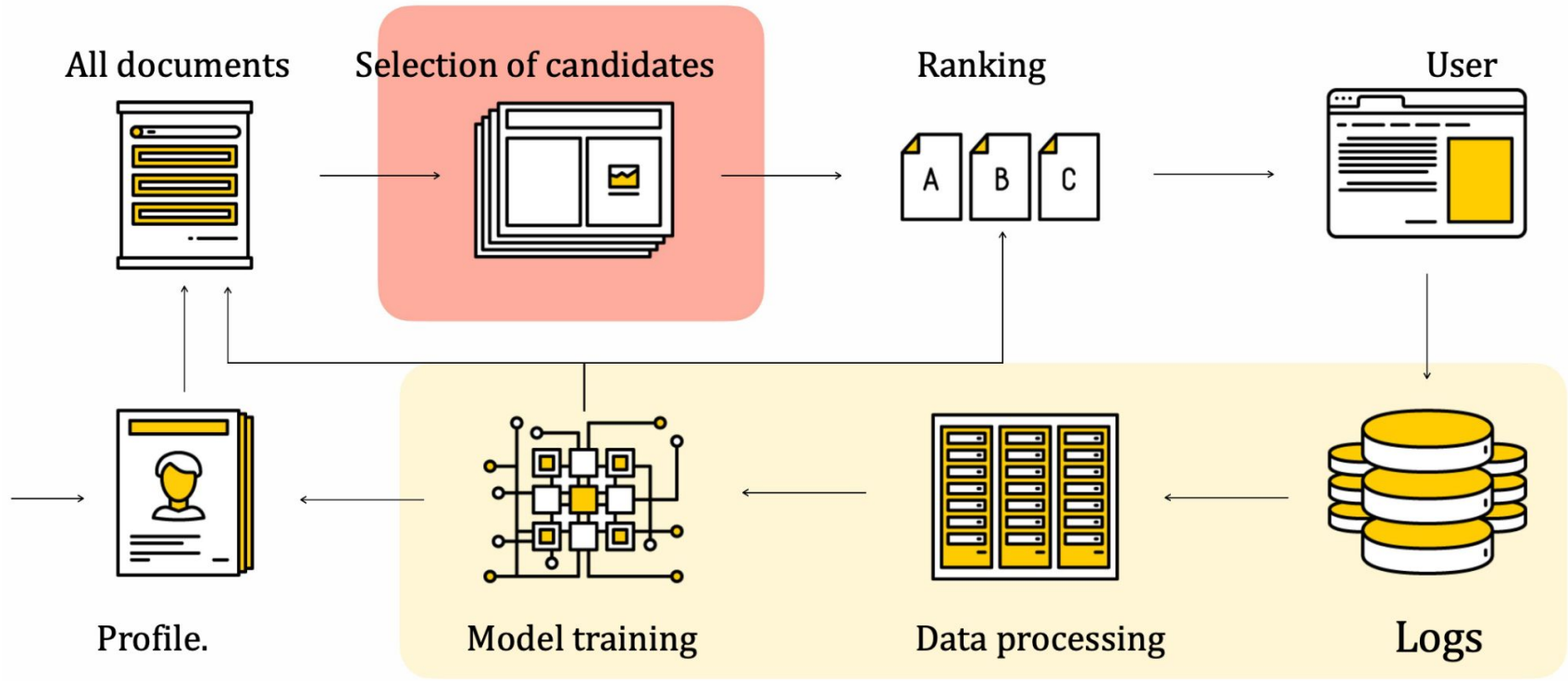


Sharding

- One server has a limit on the number of documents stored
- Let's store disjoint sets of documents on different servers, and combine the results of responses



Selection of candidates



Example: Item-based CF

- We will calculate for each product offline 1000 of the most similar to it, add it to the Key-Value storage (for example, Redis)
- Online, let's take the products evaluated by the user (the last 100) and go to Key-Value for similar ones for them

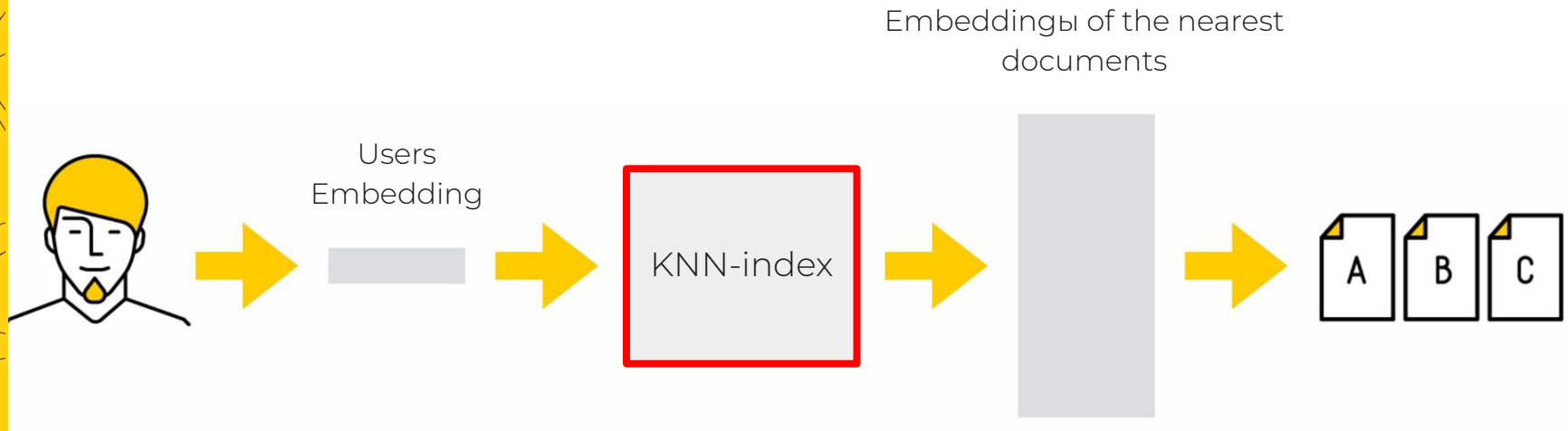
kNN indexes

girafe ai

07

kNN index

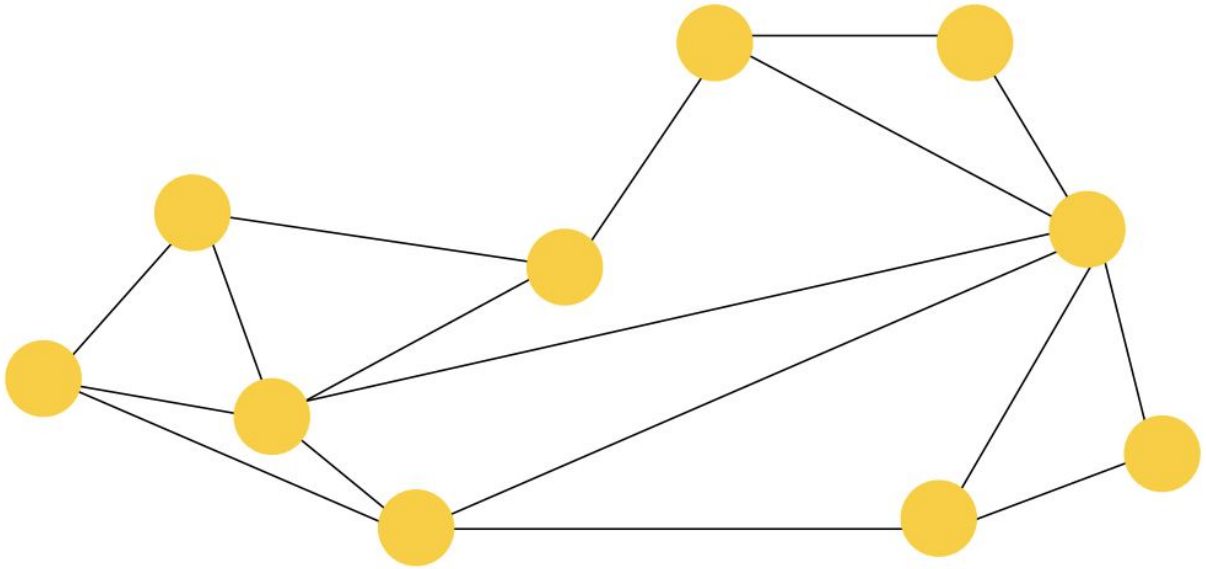
KNN index is a data structure
that allows you to quickly find the N nearest vectors for a given vector



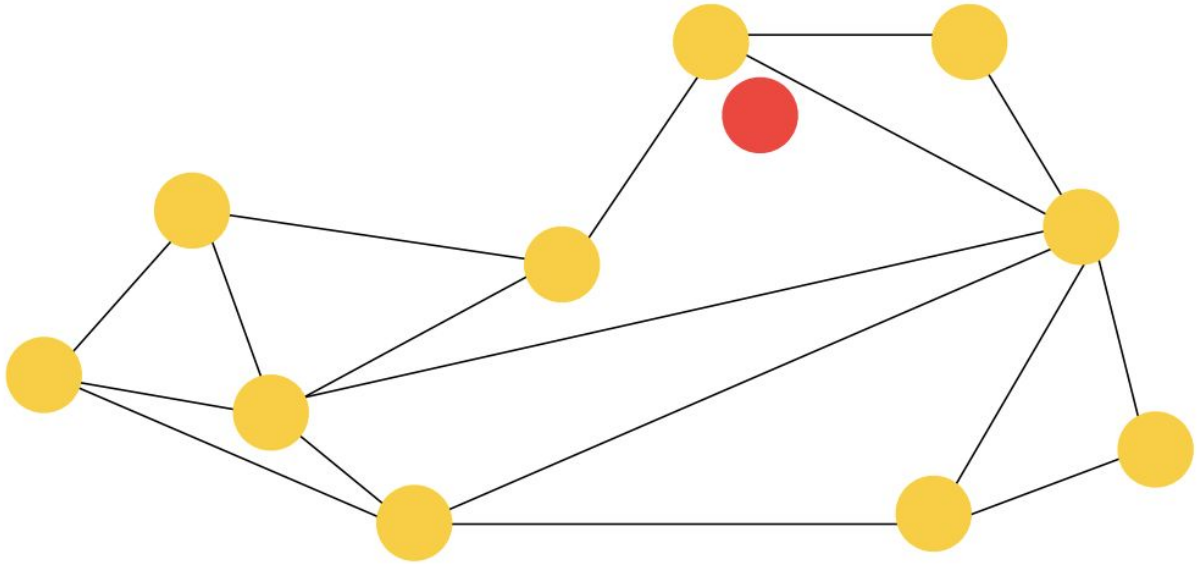
Navigable Small World

- A graph where vertices are embeddings, edges are distances
- Adding nearby and remote vertices as edges
- We start from a random vertex, count the distances and move along the graph closer to the query point

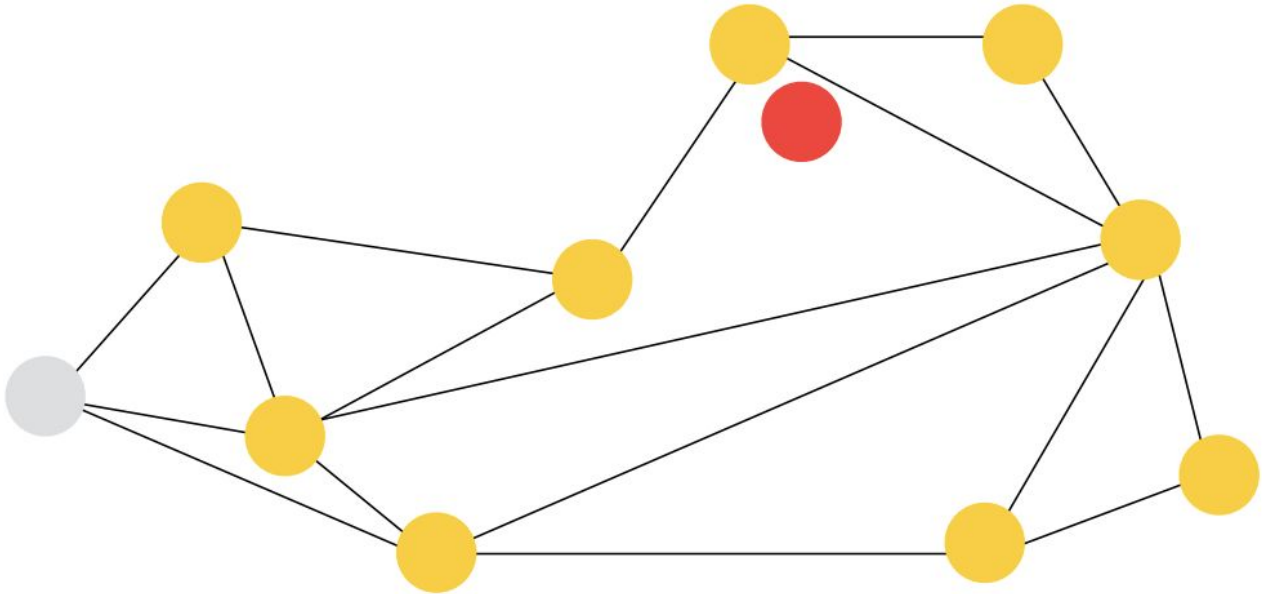
NSW (Navigable Small World)



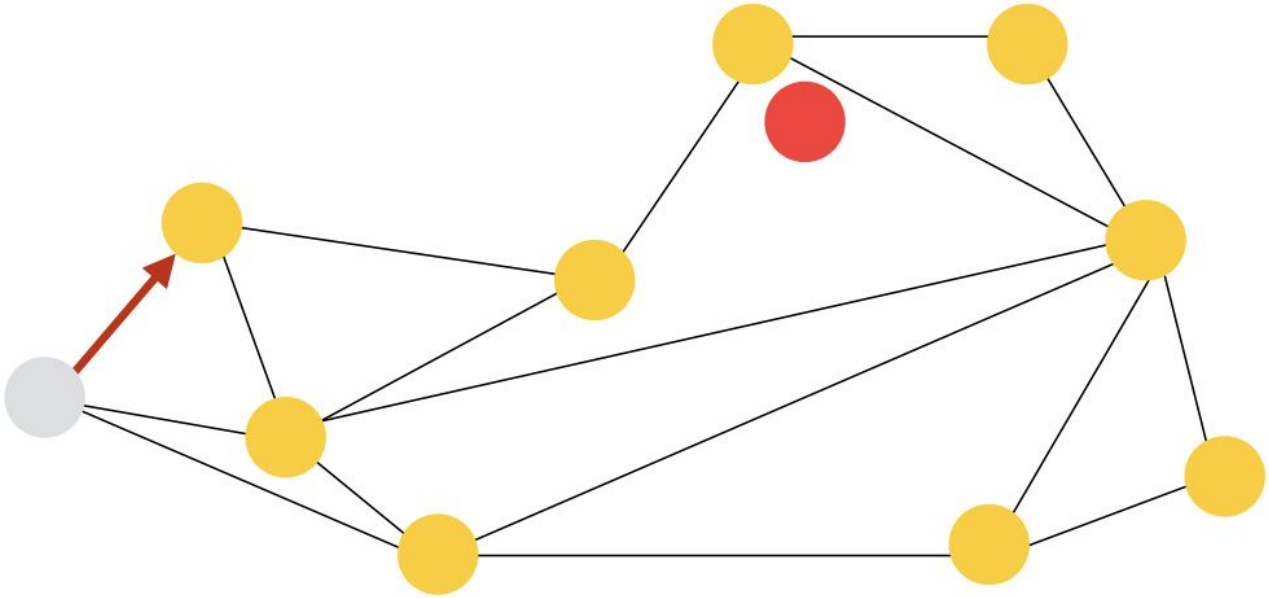
NSW (Navigable Small World)



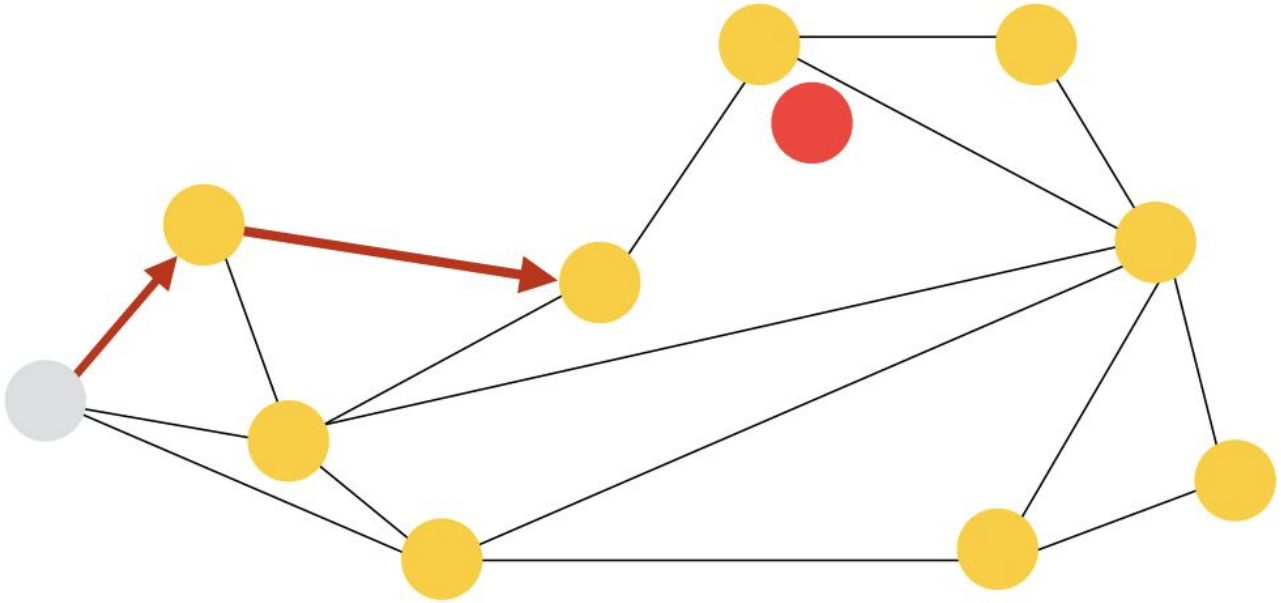
NSW (Navigable Small World)



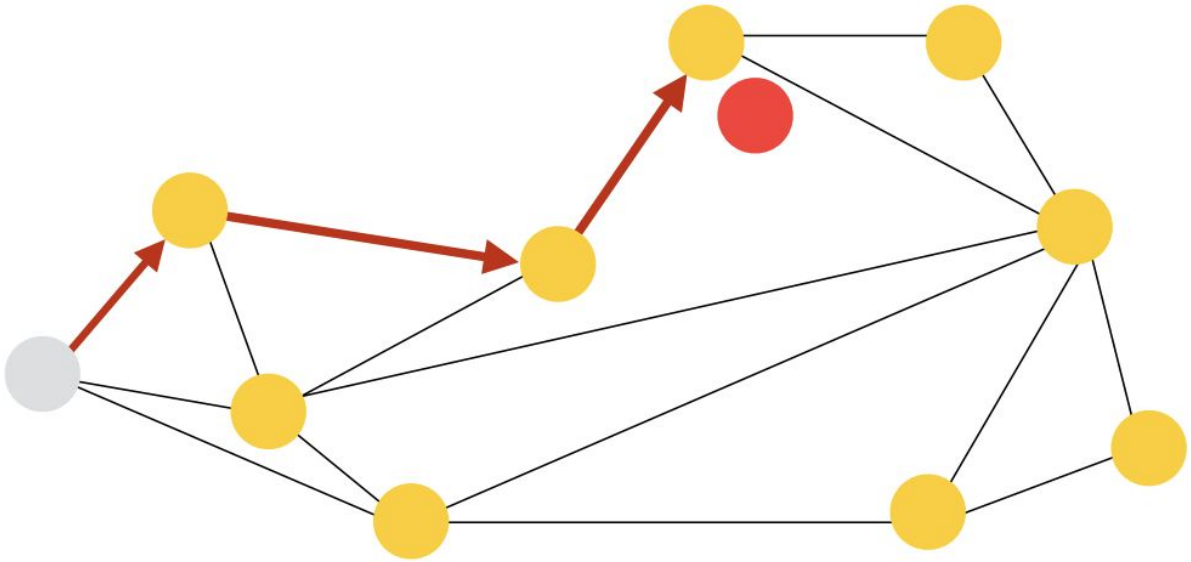
NSW (Navigable Small World)



NSW (Navigable Small World)



NSW (Navigable Small World)



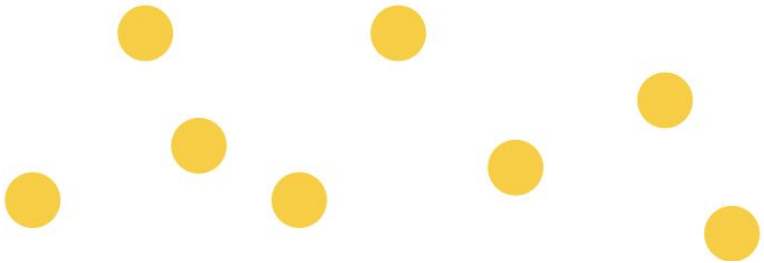
Hierarchical Navigable Small World (HNSW)

---

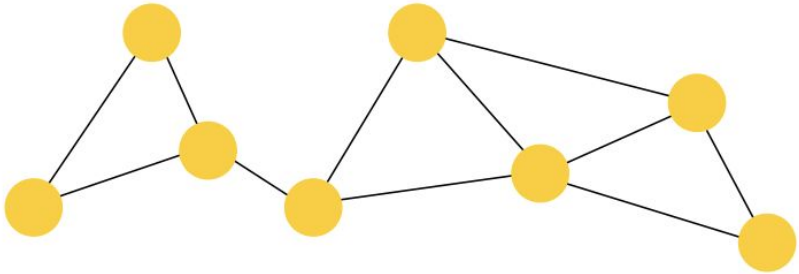
girafe ai

08

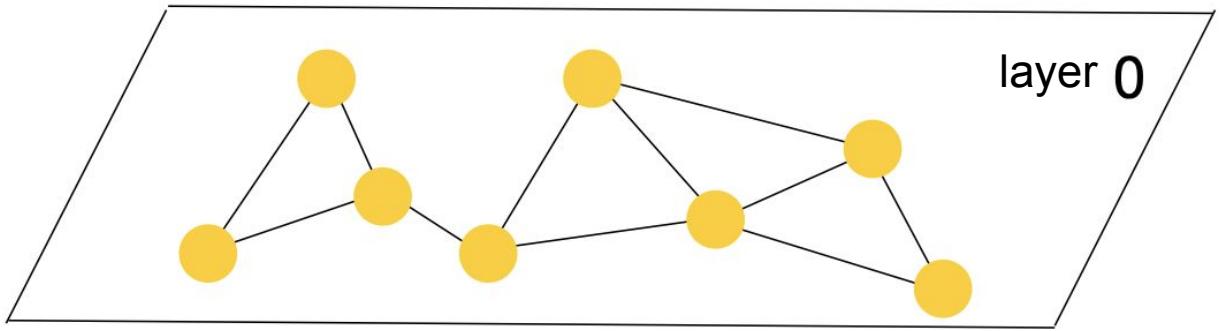
HNSW



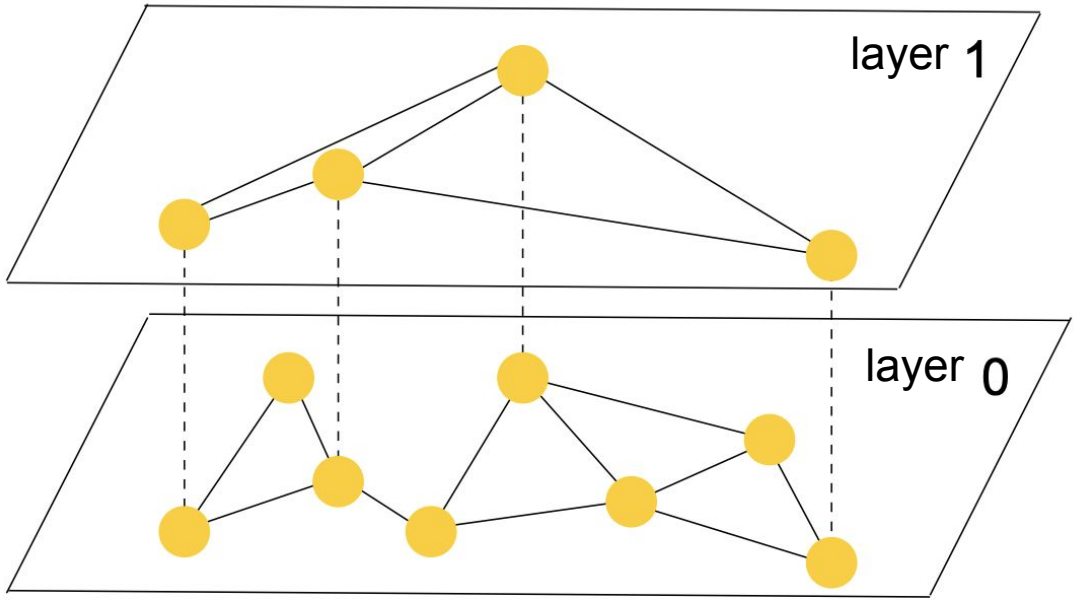
HNSW



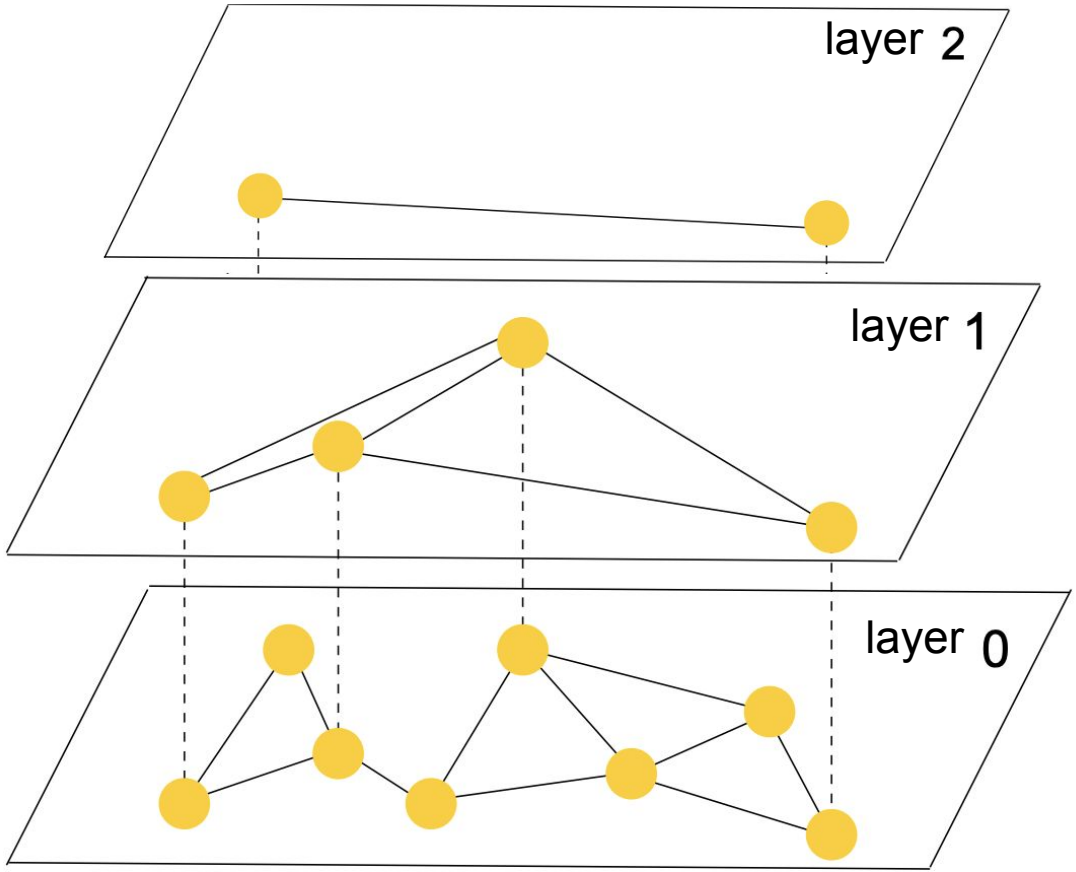
HNSW



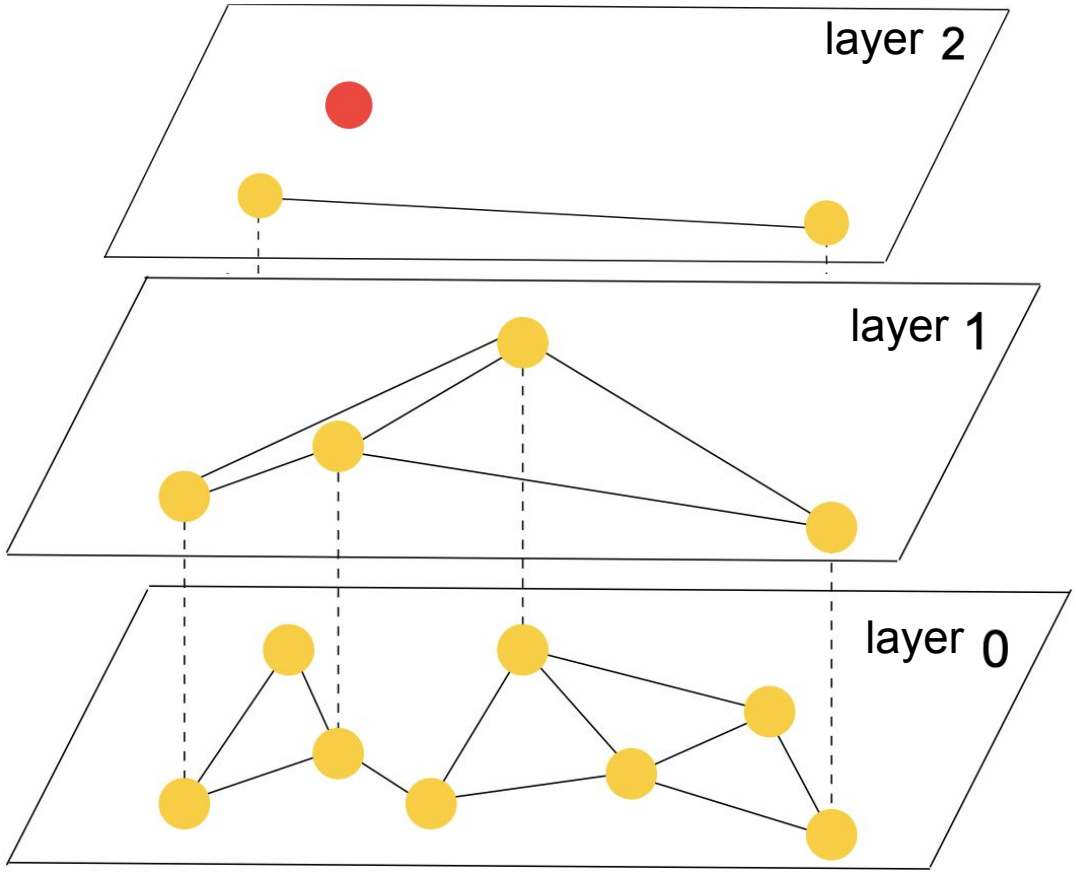
HNSW



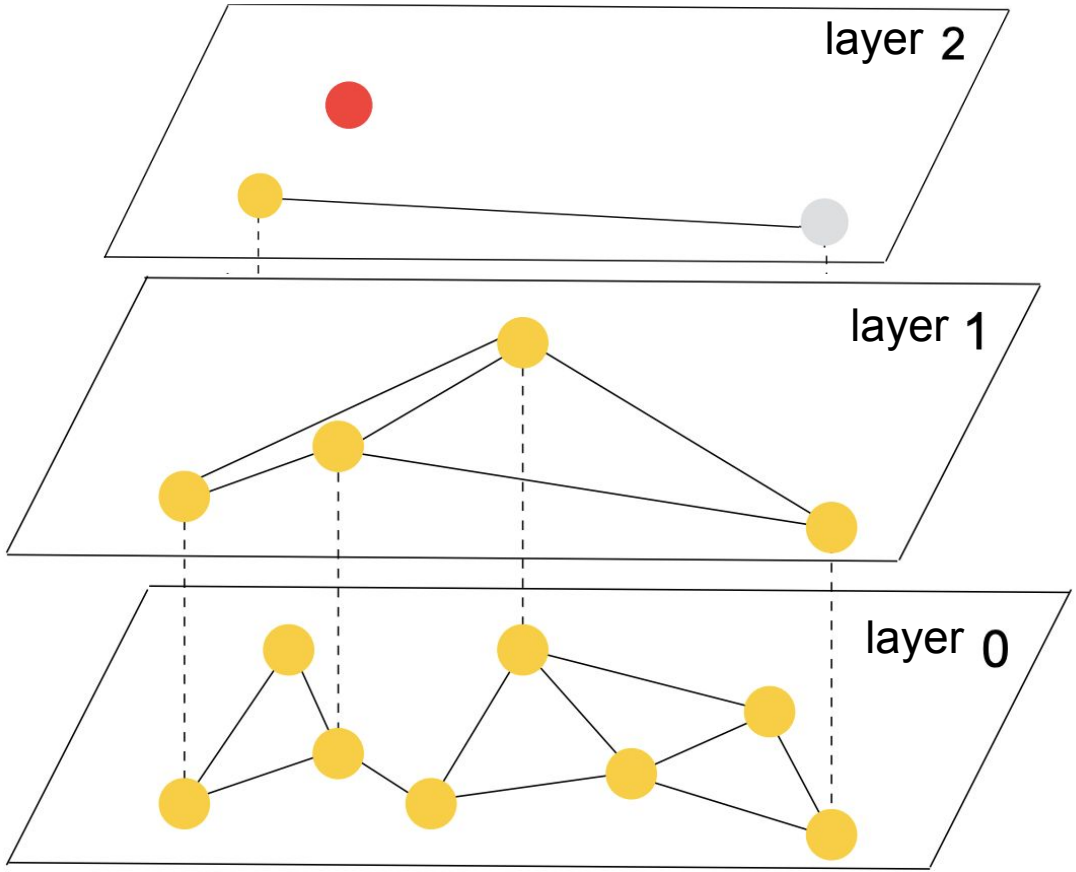
HNSW



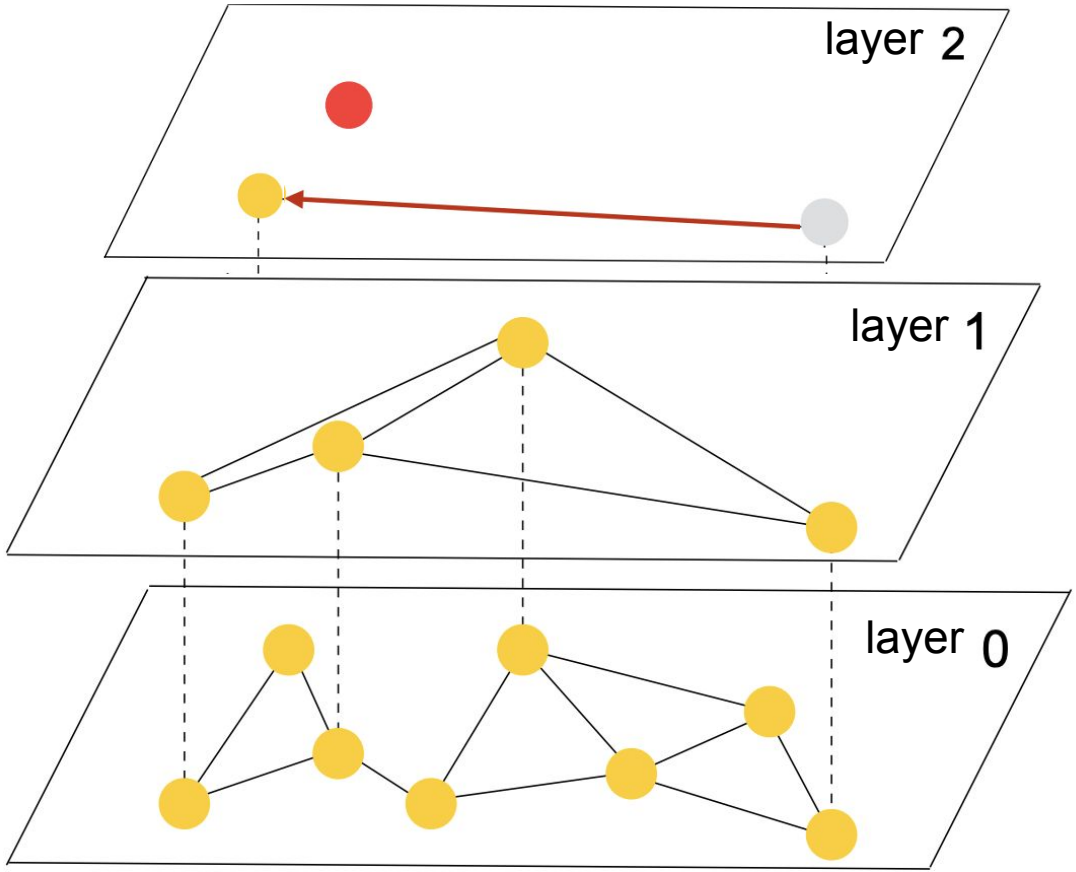
HNSW



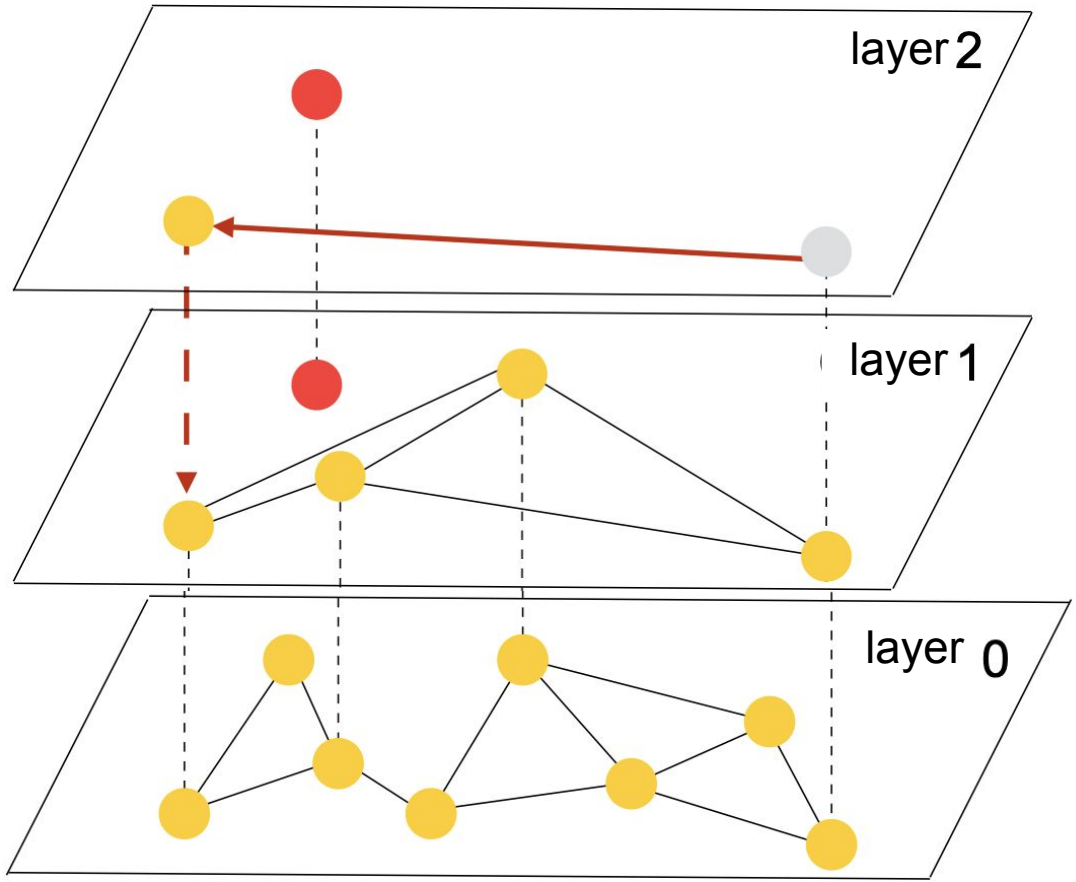
HNSW



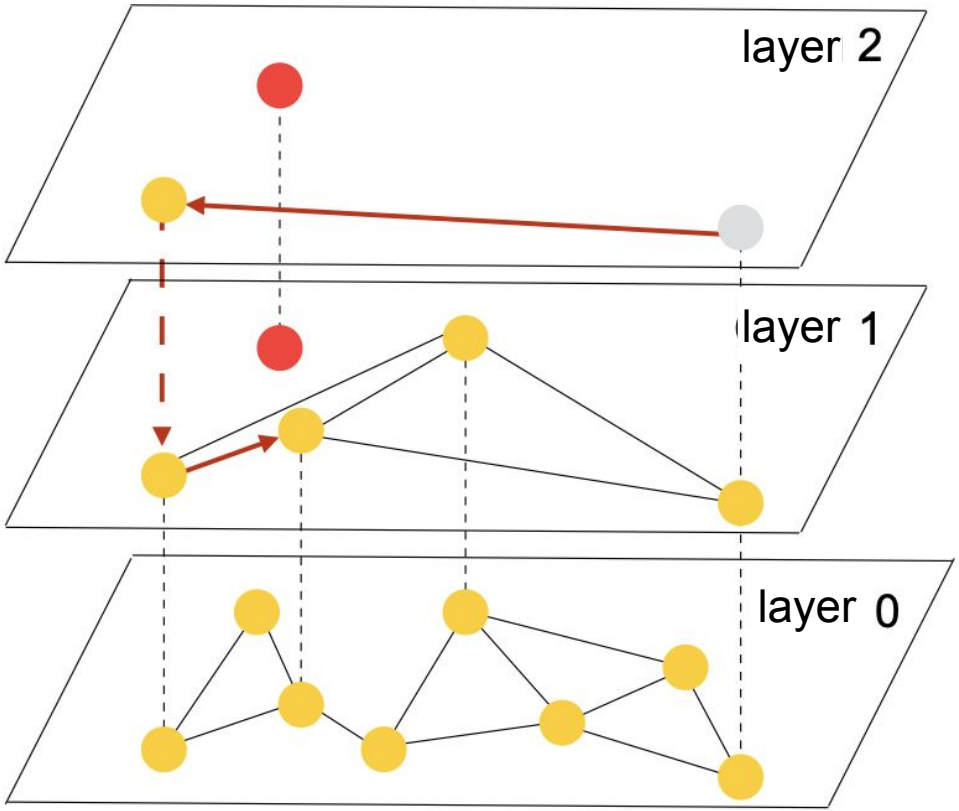
HNSW



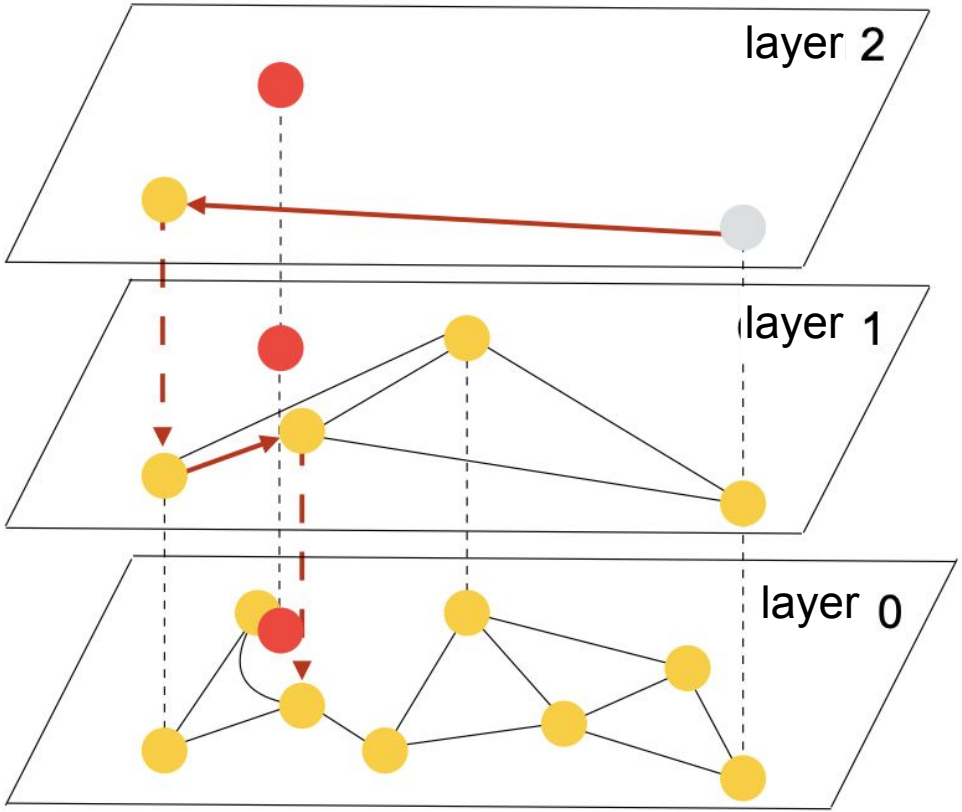
HNSW



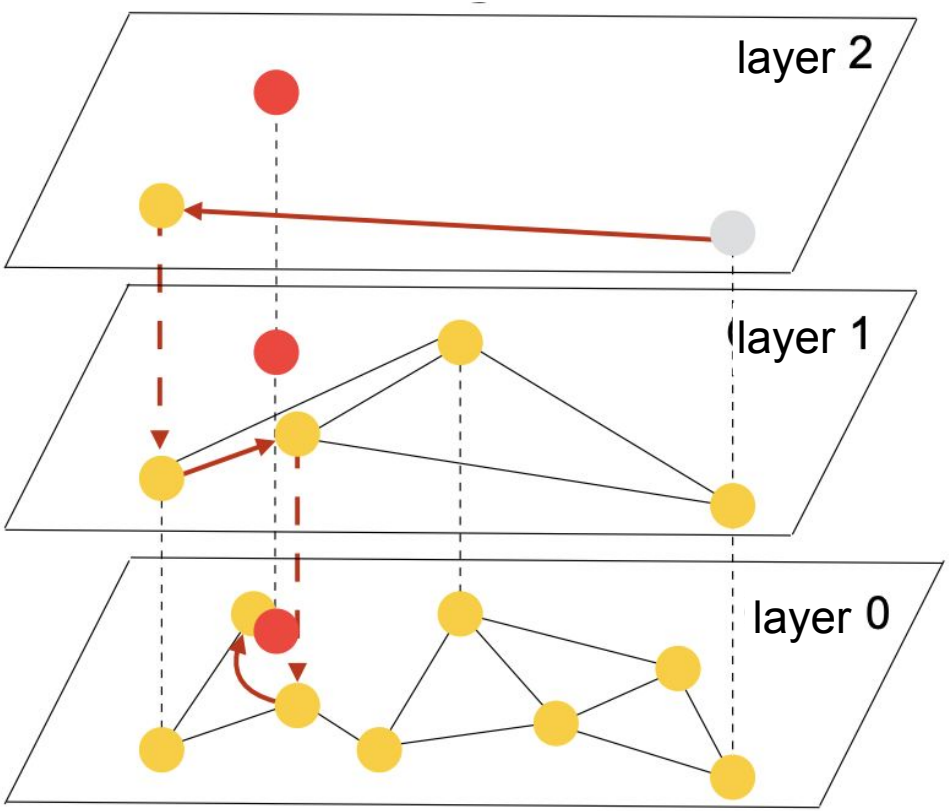
HNSW



HNSW

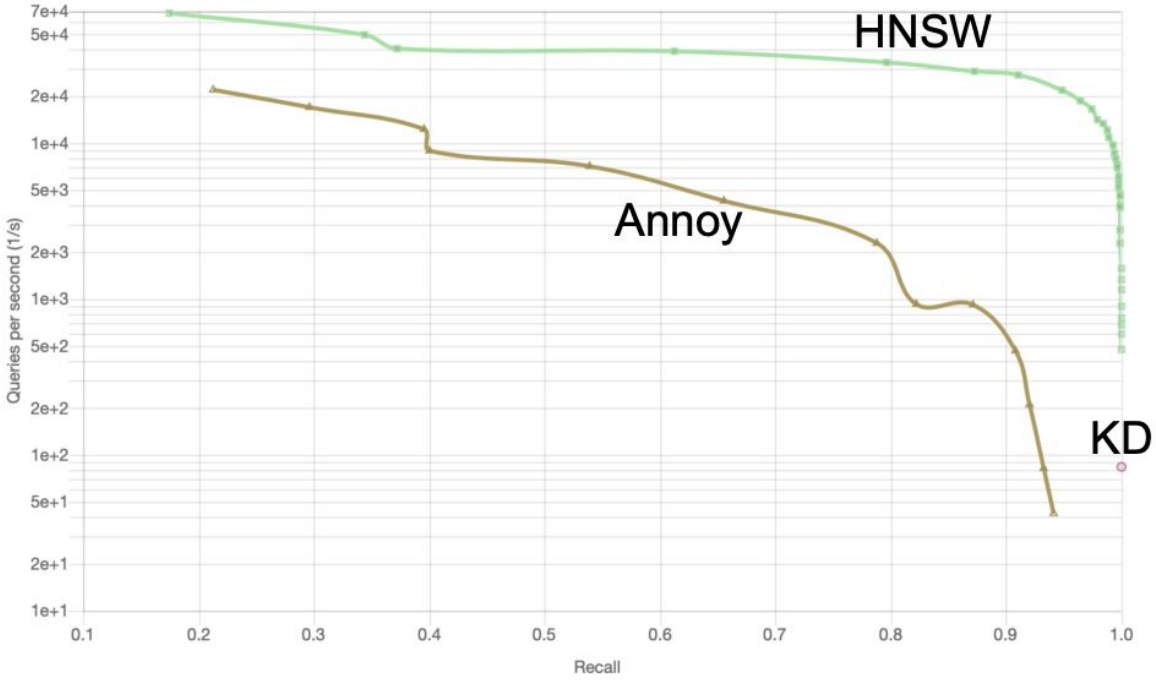


HNSW



Accuracy of approximate search

- In addition to speed, you need to find accurate nearby objects
- Comparison on ALS embedding last fm-64-dot, 360K



kNN index implemenataions

- https://github.com/facebookresearch/faiss
- https://github.com/nmslib/nmslib
- https://github.com/flann-lib/flann
- https://github.com/spotify/annoy
- https://lucene.apache.org/core/9_1_0/core/org/apache/lucene/util/hnsw/package-summary.html

Also list of vector databases from openAI
https://platform.openai.com/docs/guides/embeddings/how-can-i-retrieve-k-nearest-embedding-vectors-quickly

Ready for production solution https://github.com/qdrant/qdrant

Quantization

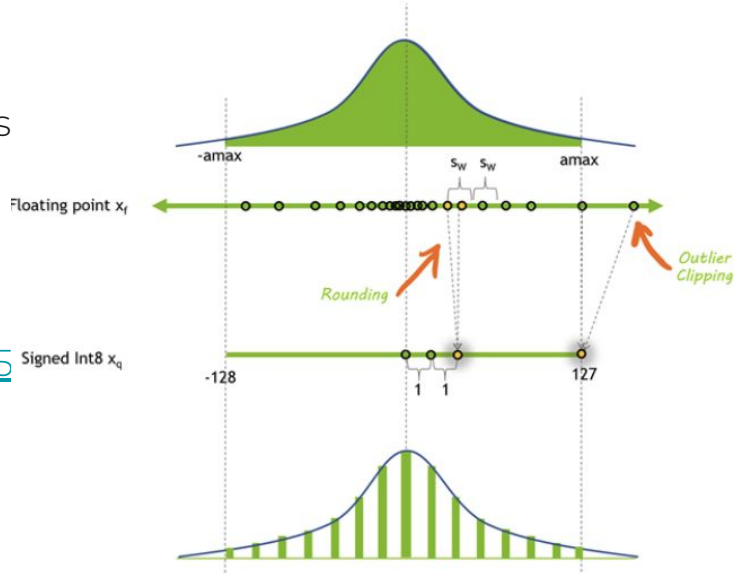
girafe ai

09

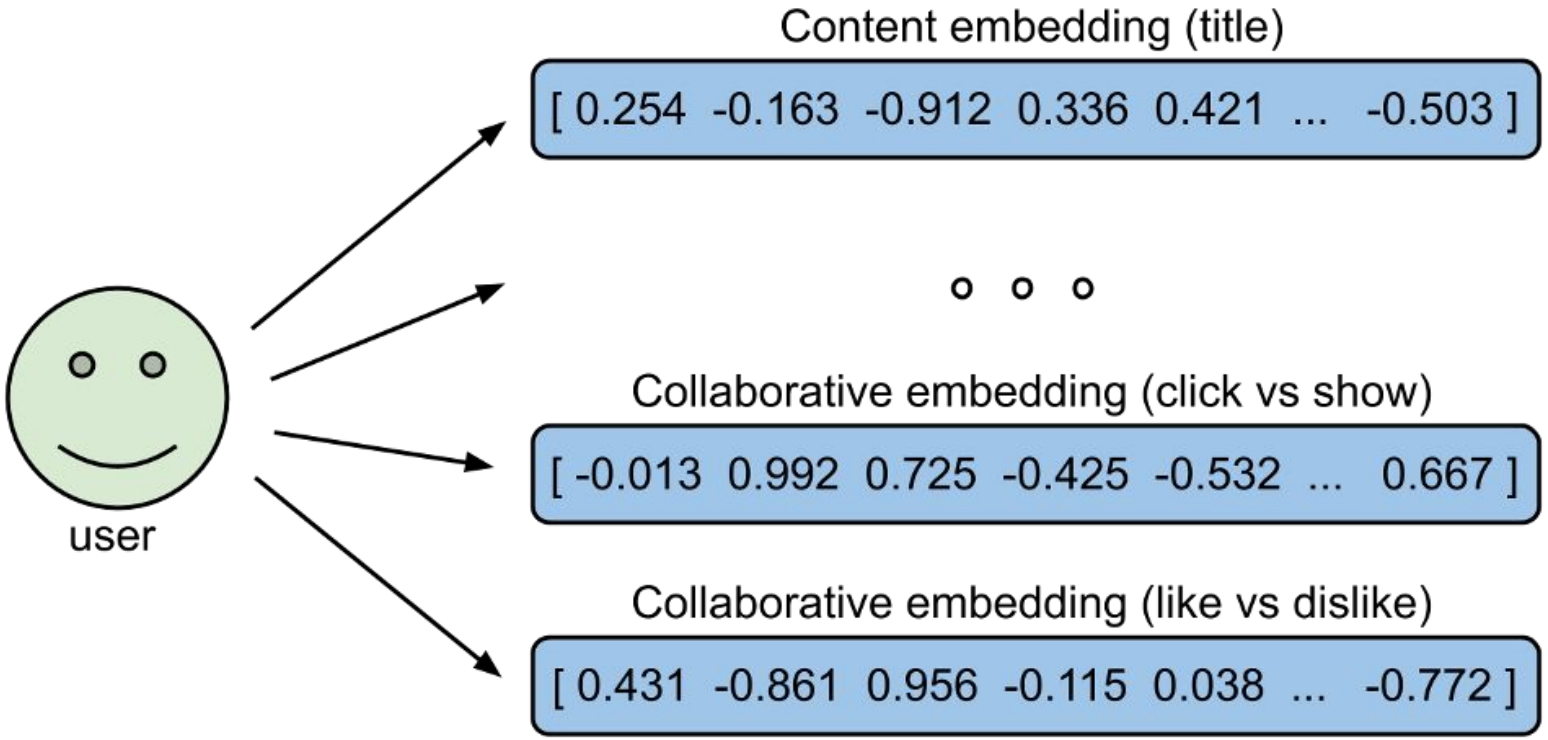
Quantization

- Embedding elements — float32
- How much will a 100-dimensional vector take in bytes?
  - 400 bites
- And if you replace it with int8?
  - 100 bites
- So let's break down the values by quantiles

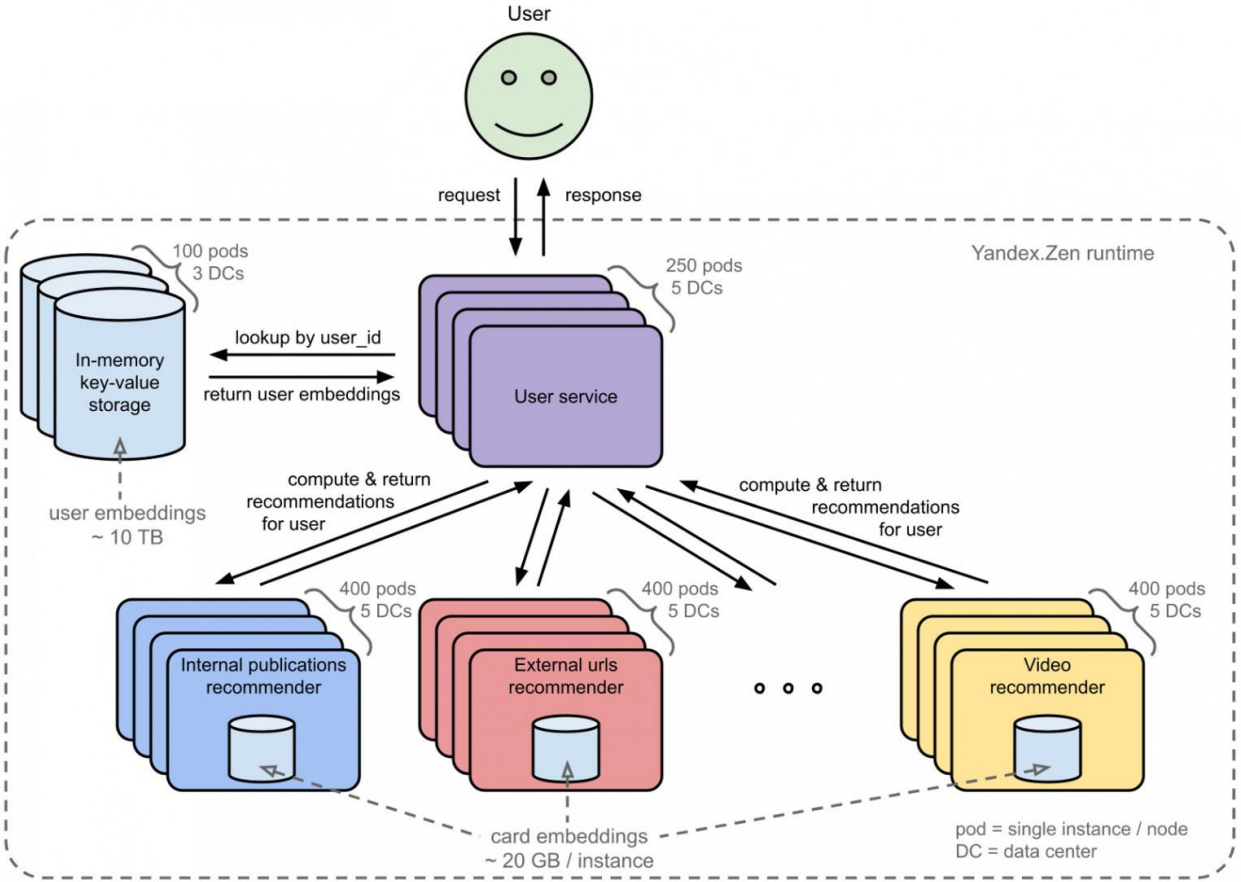
Dzen experience in quantization (ru)
https://habr.com/ru/companies/yandex/articles/5



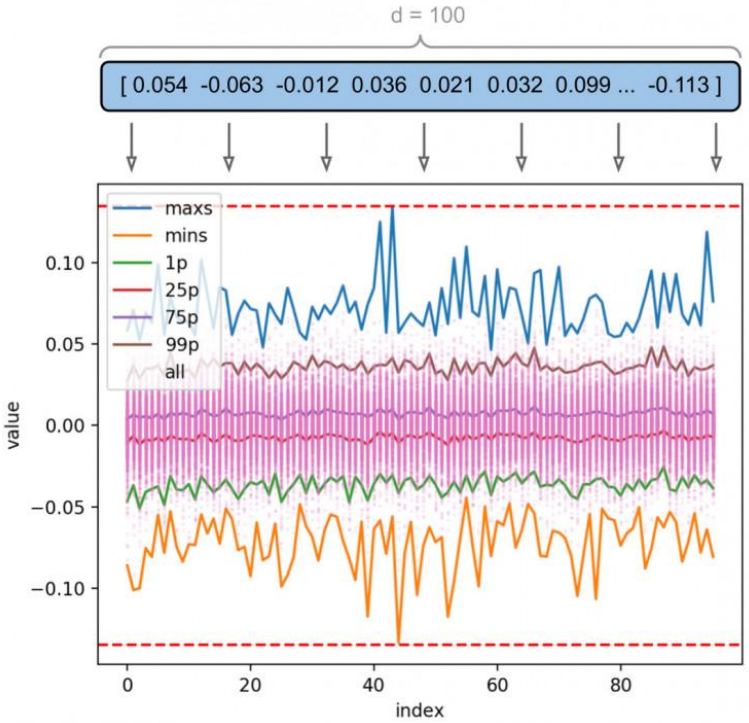
Quantization

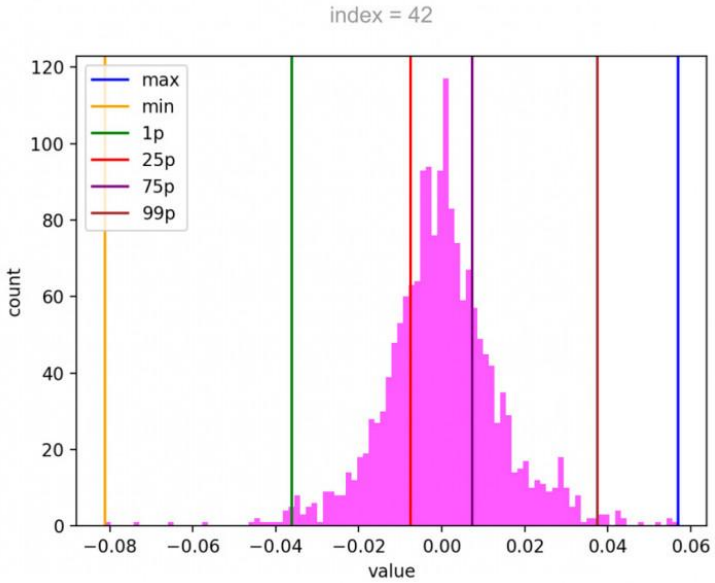


Quantization

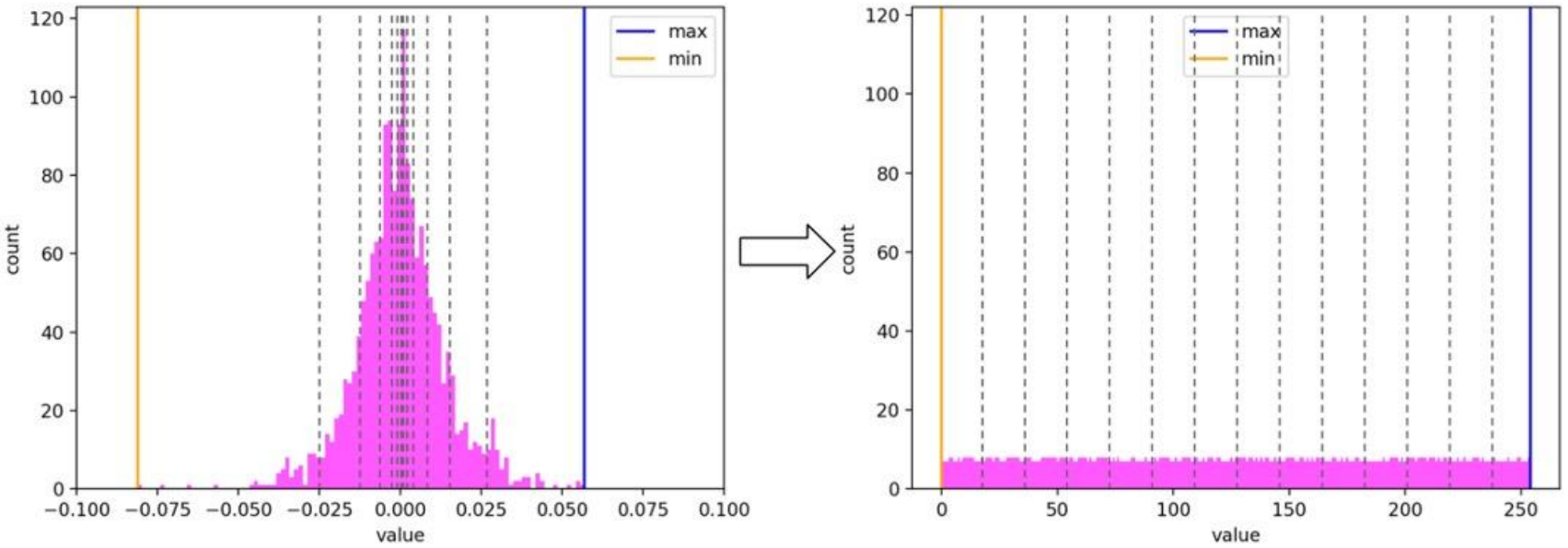


Quantization

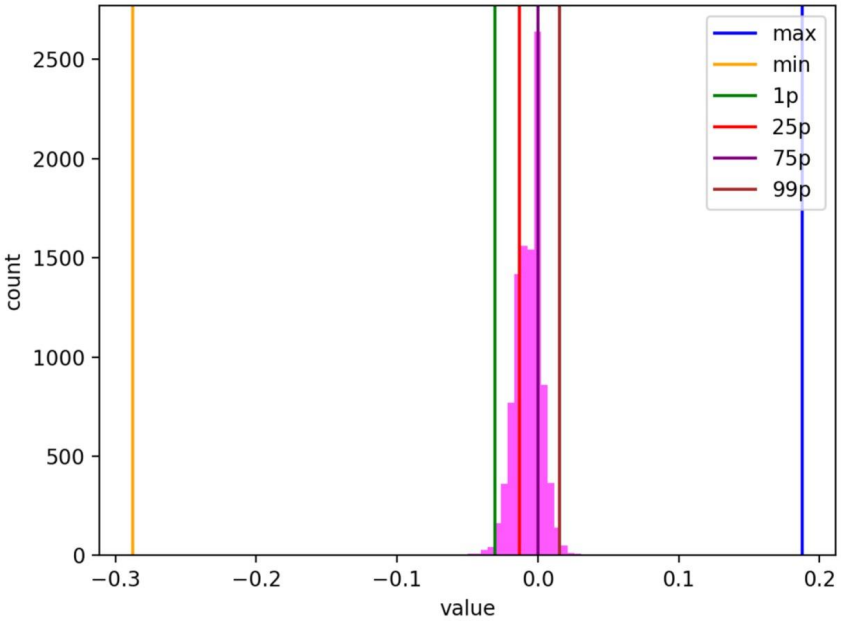




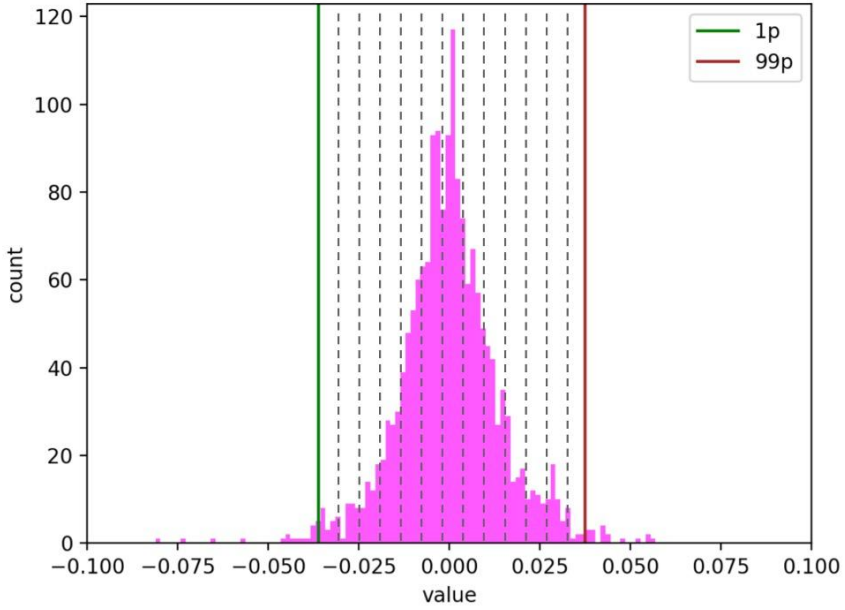
Quantization



Quantization

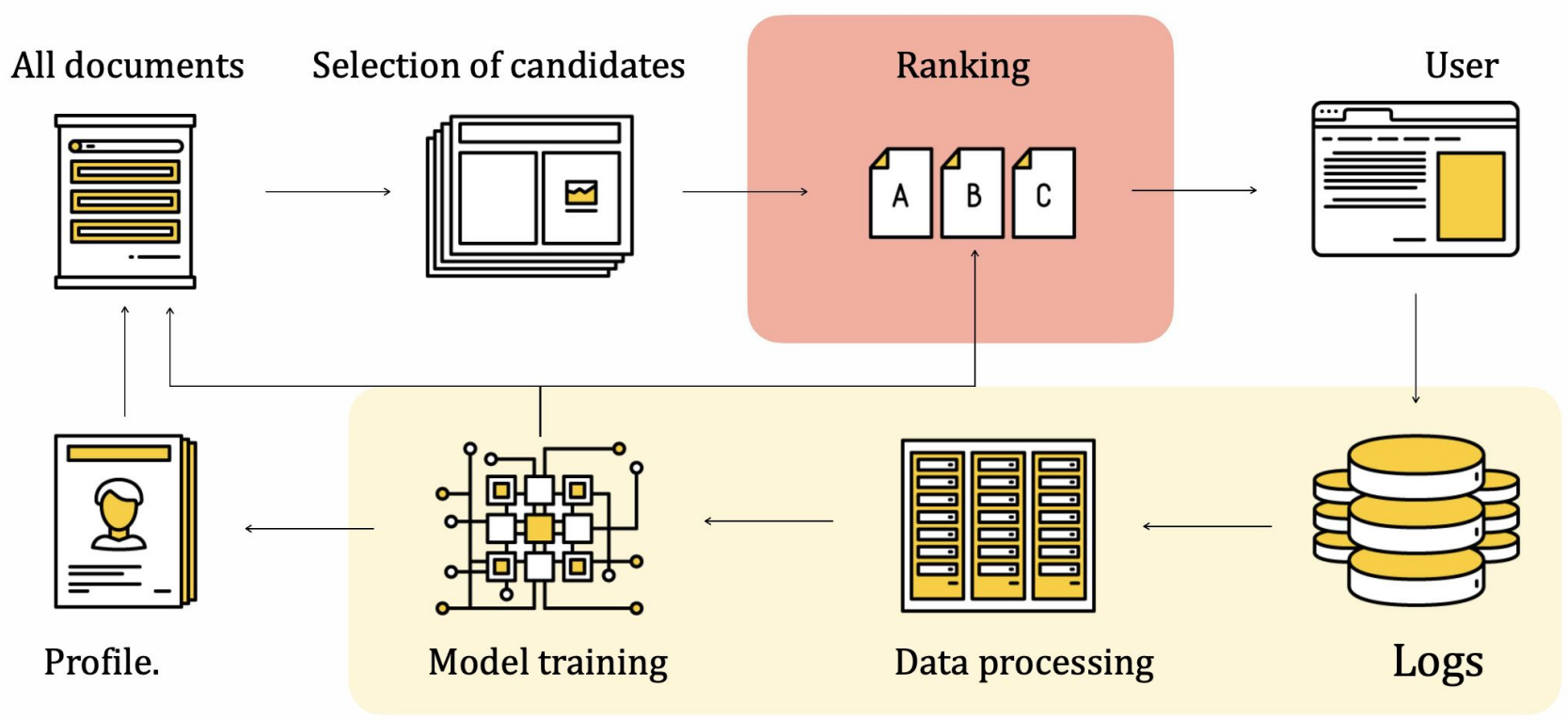


min-max distribution



1-99 percentiles distribution

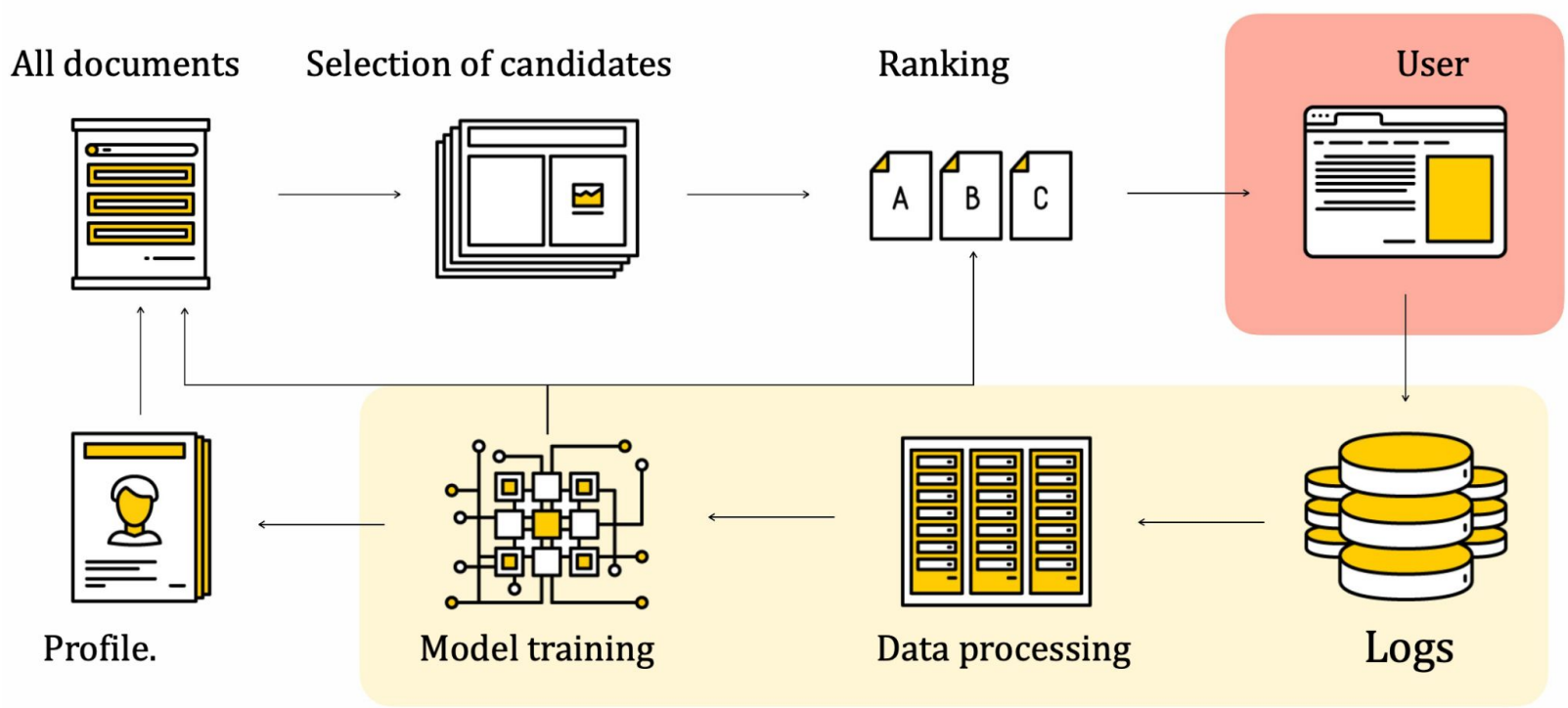
Ranking


All documents
Selection of candidates
Ranking
User
A
B
C
Profile.
Model training
Data processing
Logs

Ranking

In practice, several ranking models can be used, each of which is applied to the top of the documents selected by the previous one; gradient boosting is most often used as models. And almost always the response of the model is transformed using business rules: diversity, history of interaction with the document, freshness, time of day, etc.

Load from users



Caching during serving

- We get the results from the cache, reduce the load on the main service

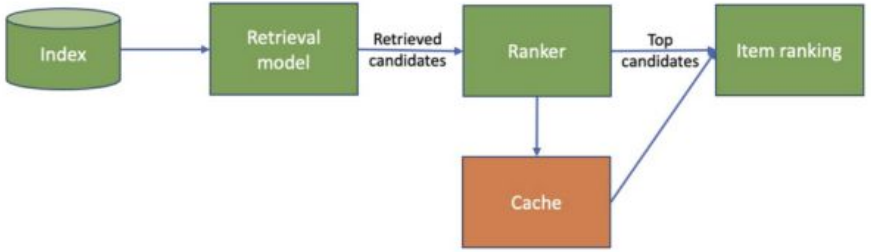


Figure 1: A Delivery Flow of Recommendation Systems with Simple Caching Strategy

- Adding a model that predicts cache usage

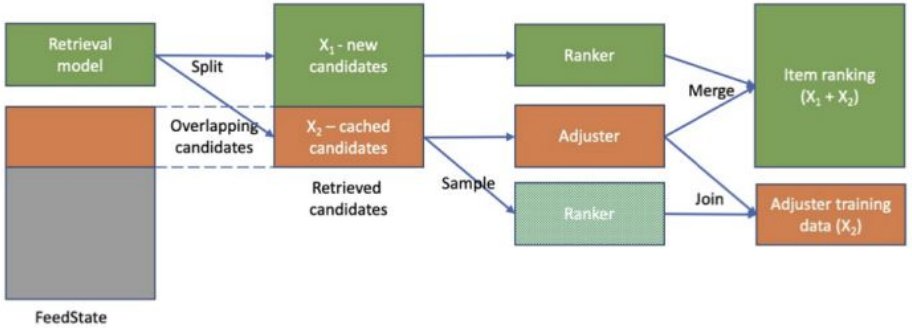


Figure 2: A Delivery Flow with Smart Cache

Recommendation system lifecycle

girafe ai

12

Features logging

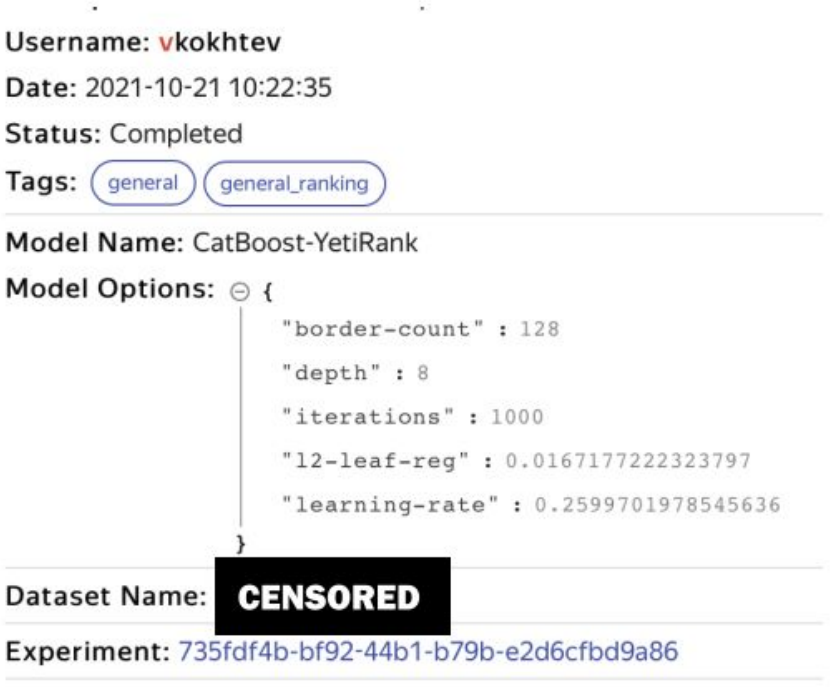
- It is necessary to train the model on the data recorded during the operation of the real system
- Collecting data on historical casts is guaranteed to lead to a discrepancy in values when training and applying models
- Errors in calculations will give rise to incorrect signs with leaks of the target variable ("face")

Reproducible learning

Always versioning your experiments

- Data
- Parameters of models and training
- Executable code

- And of course fix Random Seed (I advise 42)



Metrics monitoring

- Metrics from the last lecture must be constantly monitored
- Model quality metrics

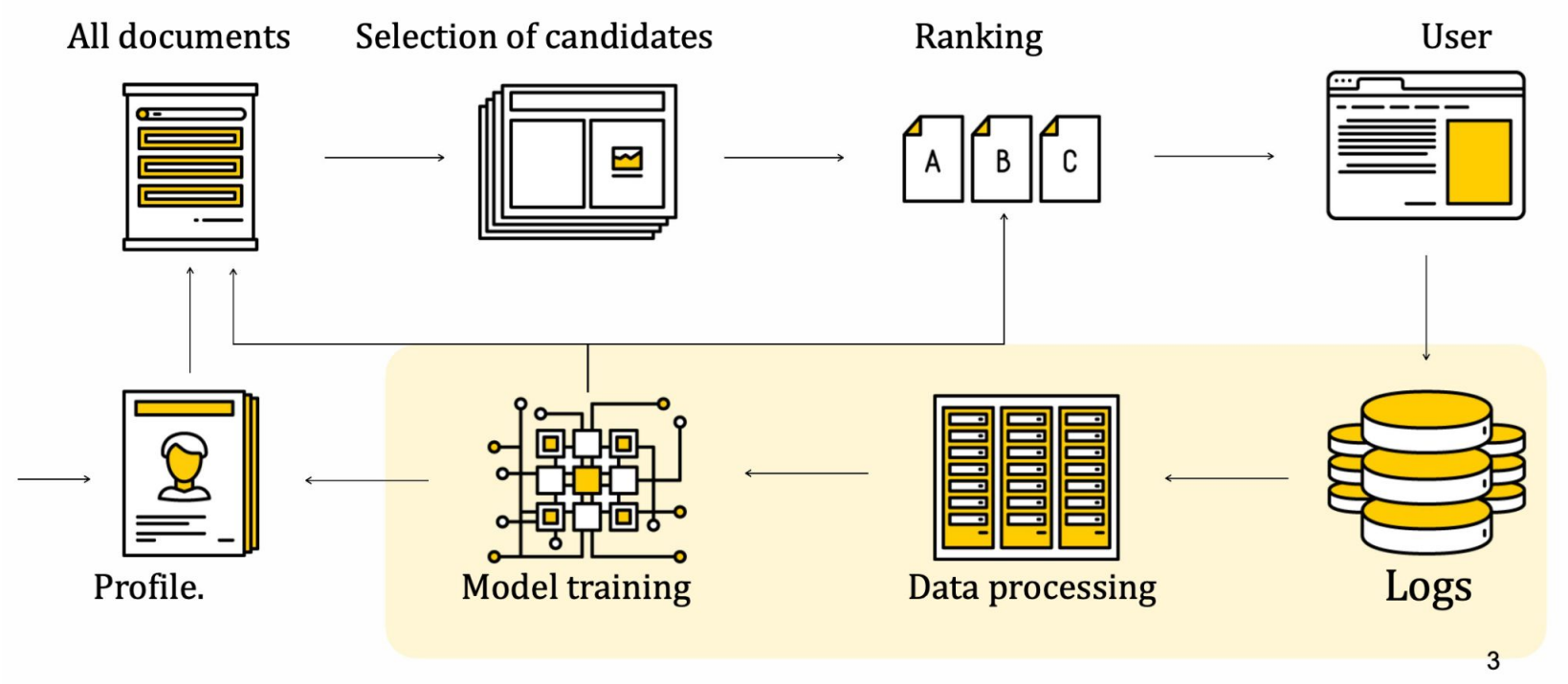
Technical

- Number of requests, response time
- Values of features and responses of models

Regular models retraining

- Even if you have received a good recommendation model once, it will become outdated over time. Therefore, it is necessary to periodically train/retrain models on fresh data
- You can trust automation and run each regular model in an A/B experiment, and if the main metrics are stable or improved, include them for all users

Resume


All documents
Selection of candidates
Ranking
User
A
B
C
Profile.
Model training
Data processing
Logs
3

Thanks for attention!

Questions?

---

girafe
ai