Эволюция векторного поиска в YDB: от базовых методов к масштабируемому глобальному индексу

Александр Зевайкин

Руководитель группы разработки,
кандидат технических наук, доцент

YDB

Saint HighLoad++

YDB: движение в сторону ML

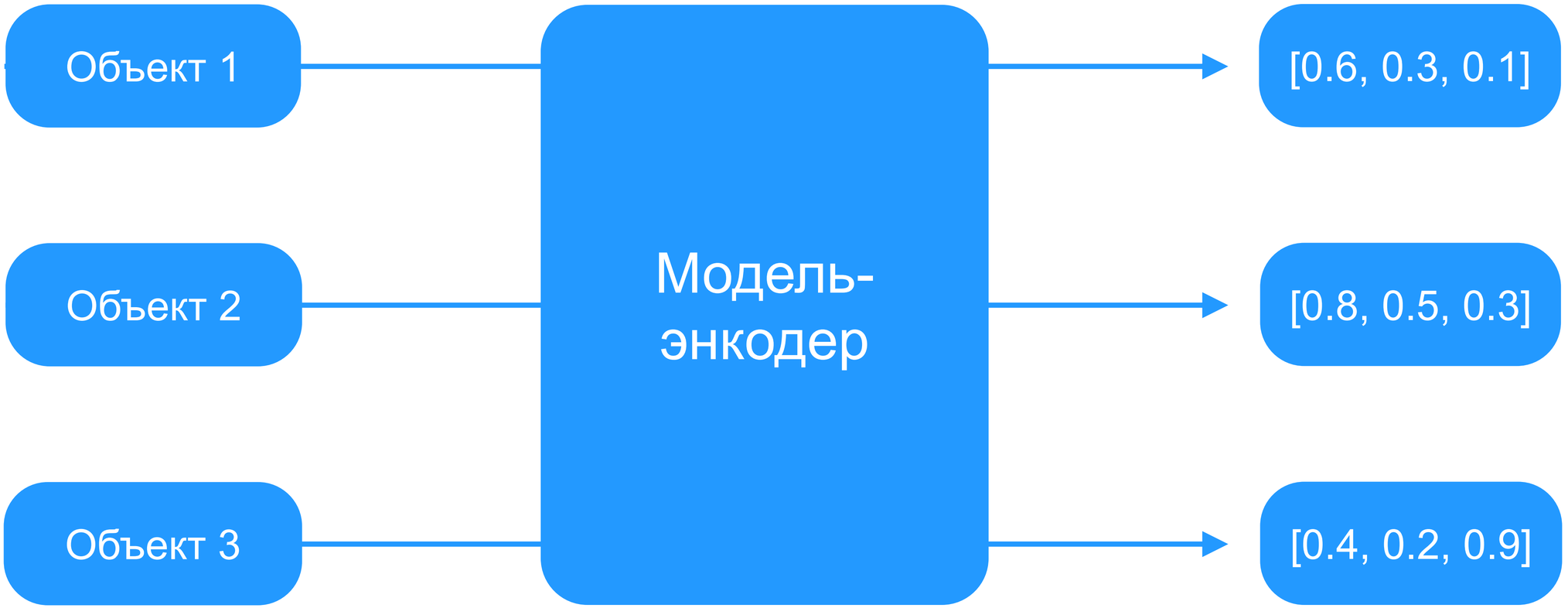
Часть 1

HL 2025 Saint HighLoad++

Векторы (Embeddings)

Текст, видео, аудио,
сырые данные

Векторы
фиксированной длины



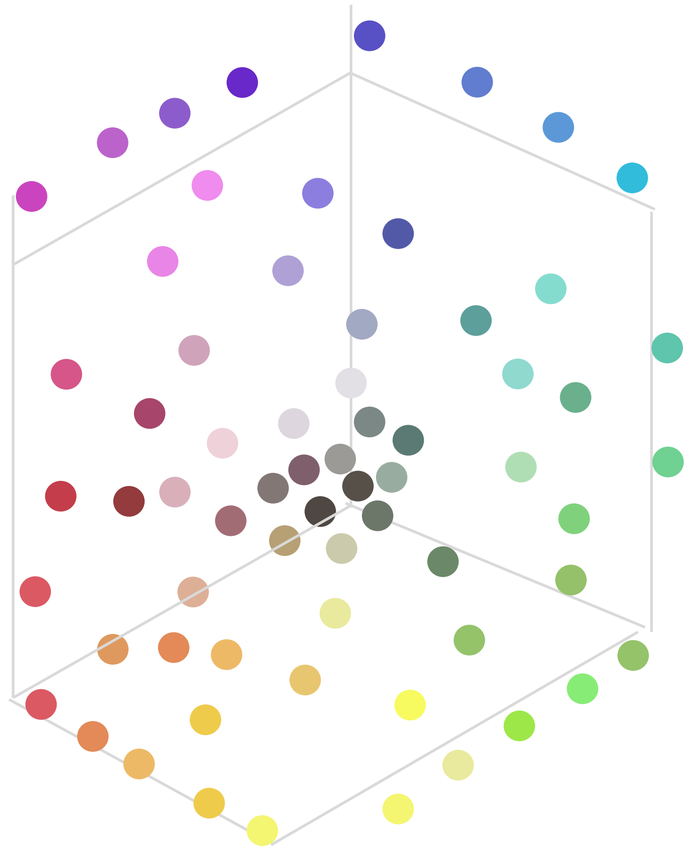
Векторное пространство

Чем ближе семантически, тем ближе
в векторном пространстве

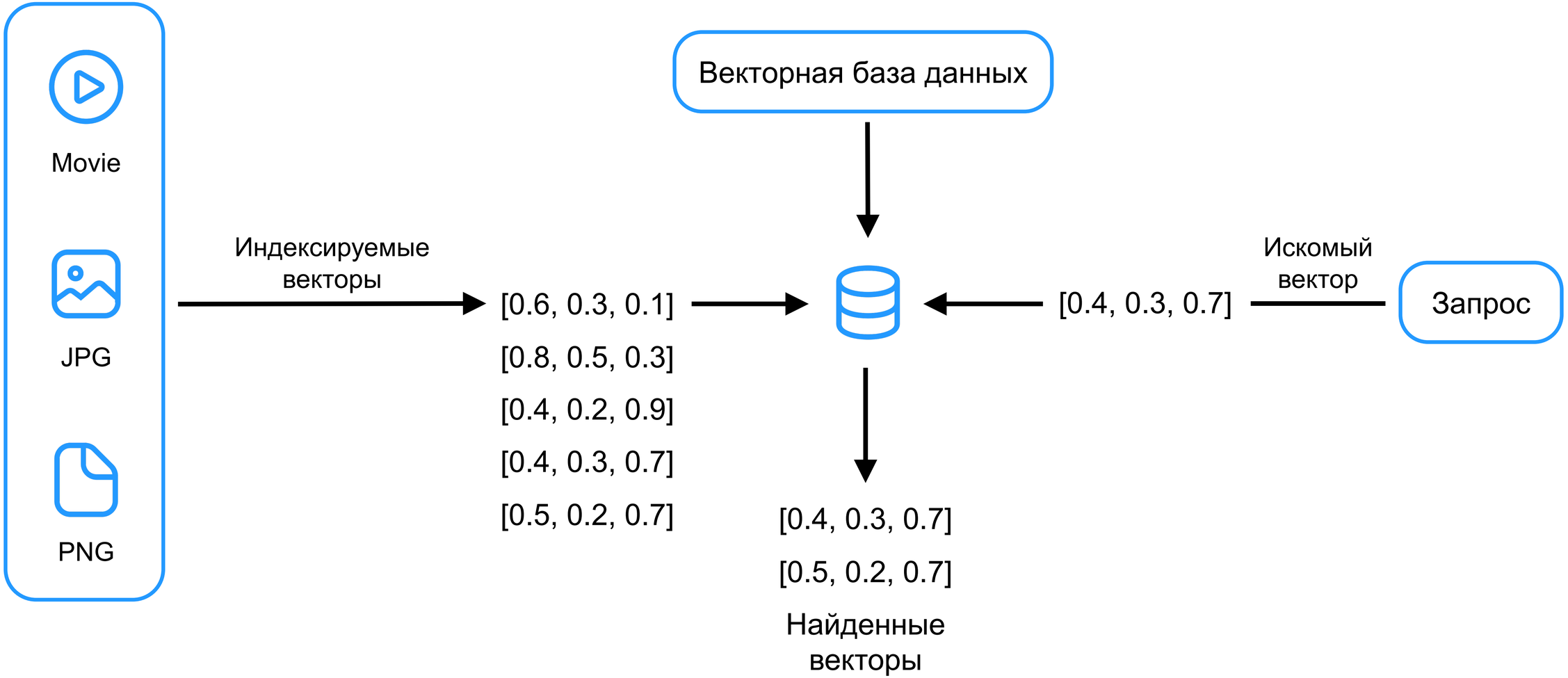
[0.6, 0.3, 0.1]

[0.8, 0.5, 0.3]

[0.4, 0.2, 0.9]



Векторная база данных



Недостатки больших языковых моделей (LLM)

1

Затраты на обучение

Порядок 100 тысяч GPU-дней

Тенденция к кратному росту

2

Устаревшие данные

Свежесть данных ограничена

Не может знать текущие факты

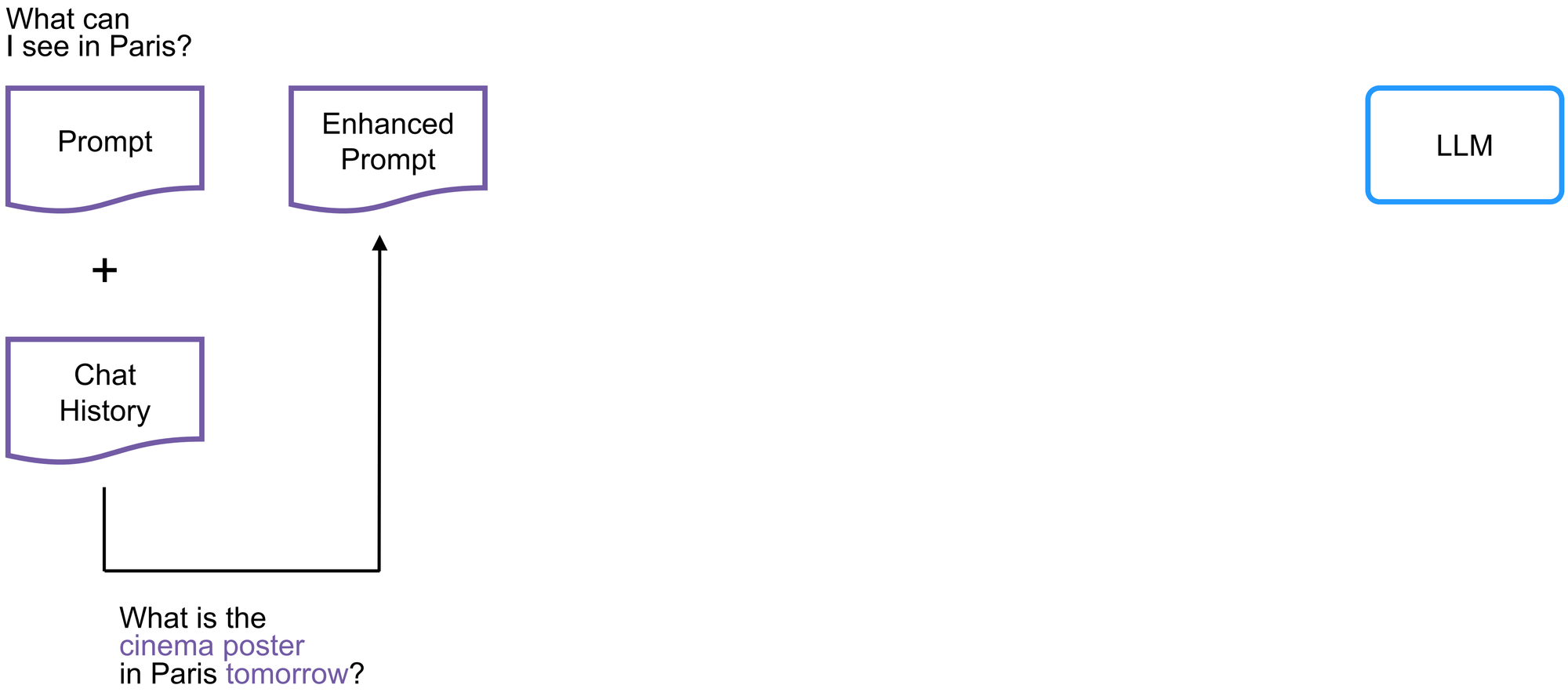
Что самое странное нашли на дне шахты лифта?

Источник: сгенерировано в Шедевруме.

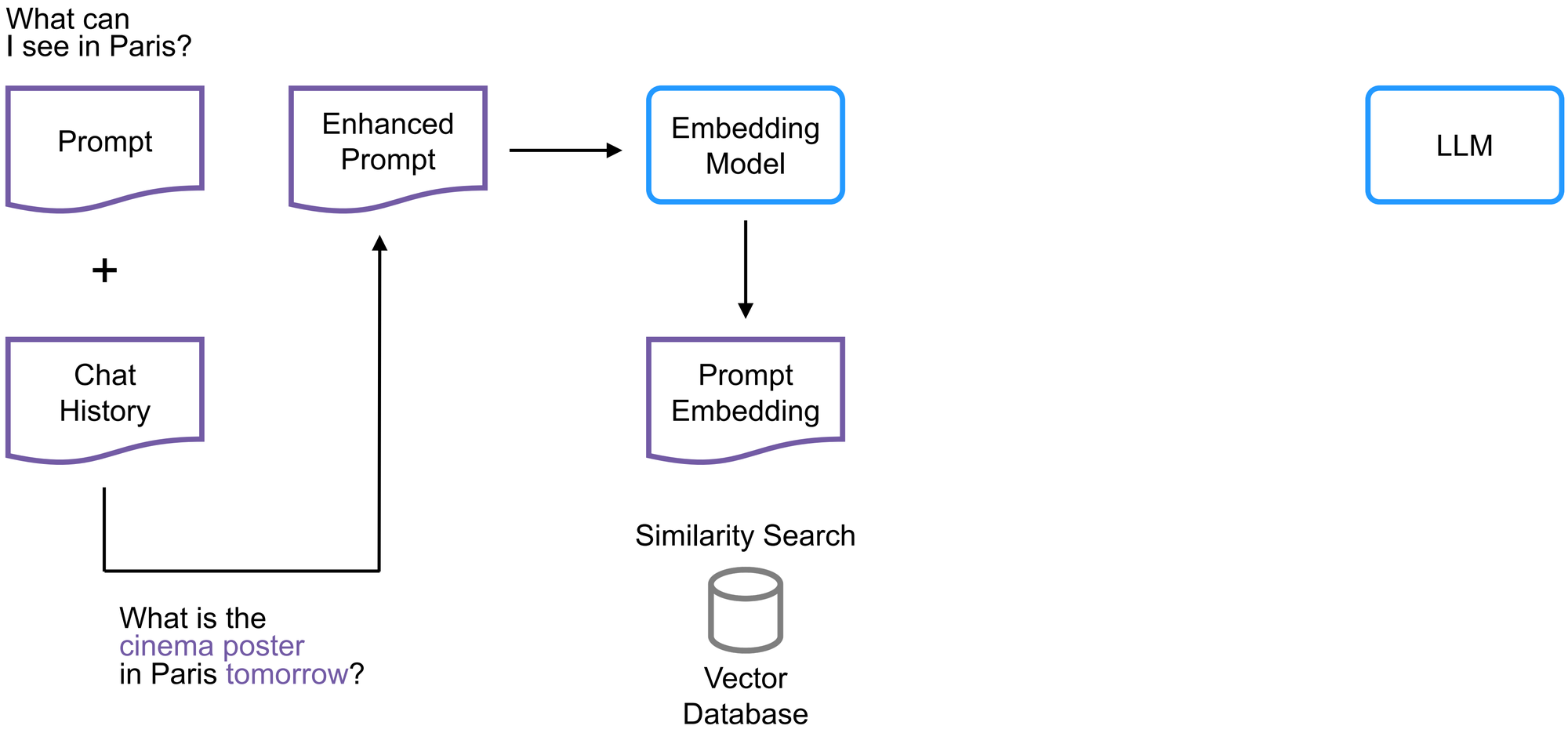
Retrieval-Augmented Generation

What can
I see in Paris?

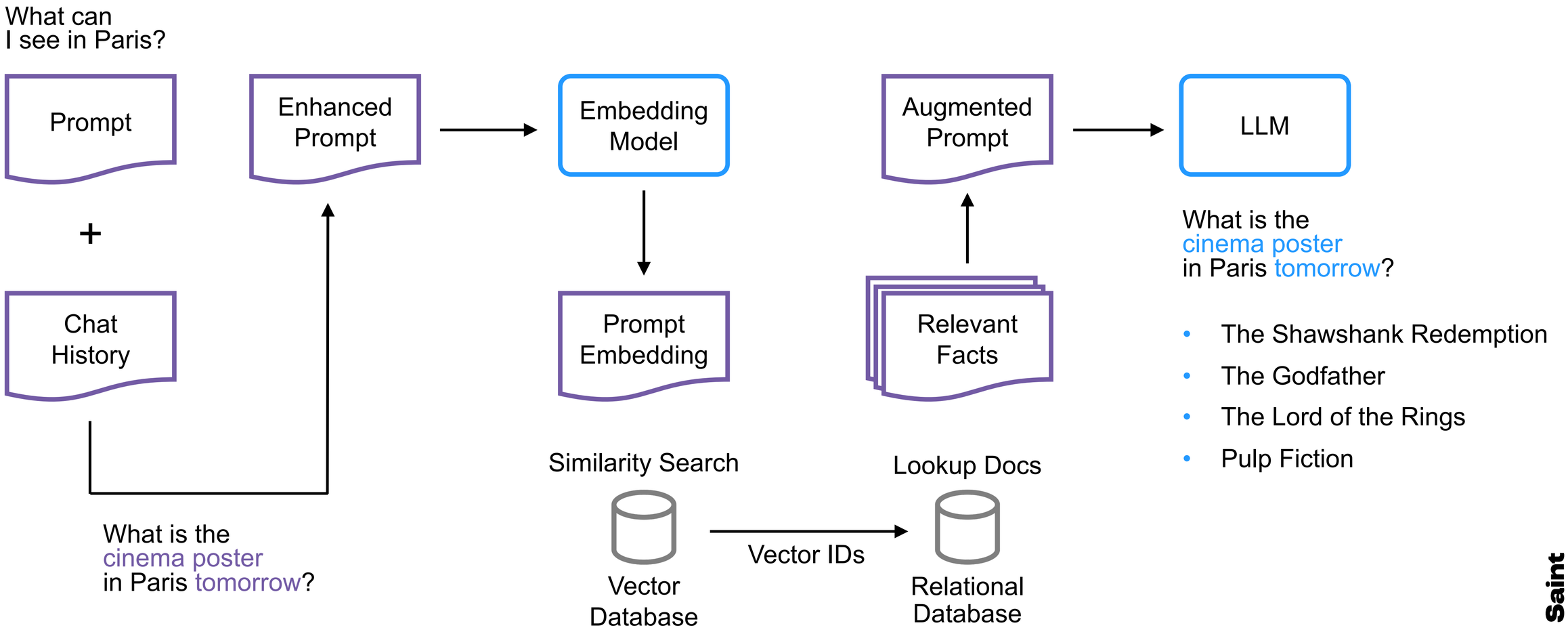
Retrieval-Augmented Generation



Retrieval-Augmented Generation



Retrieval-Augmented Generation



Пример пользовательского сценария

Голосовой помощник

- Помнит только ограниченный контекст диалога
- Не помнит факты пользователя

Источник: сгенерировано в Шедевруме.

Пример пользовательского сценария

Голосовой помощник

- Типичный пример RAG-технологии
- Сервис памяти хранит факты о пользователе
- Подгружает факты в момент диалога
- Обогащает пользовательские запросы

Требования

- Хранение миллиардов фактов
- Синхронный векторный индекс
- Приближённый поиск по векторам

Состояние рынка в области векторного поиска

Open Source DBs	Year
PostgreSQL	2021
Lucene™	2021
OpenSearch	2022
Redis™	2022
Cassandra®	**2023**
ClickHouse®	**2023**
MariaDB	**2023**
YDB	**2024**

Non-Open Source DBs	Year
Elasticsearch	2019
Oracle	**2023**
MongoDB	**2023**

Clouds	Year
Pinecone	2019
Amazon OpenSearch	2020
Google Cloud SQL	**2023**
Alibaba	**2023**
Microsoft Azure	**2023**
Amazon DocumentDB	**2023**
Cloudflare	**2023**
Google LangChain	**2024**

Поисковый индекс требует инфраструктуры

Масштабирование

- Шардирование
- Репликация
- Мультитенантность
- Пики нагрузок
- Несколько дата-центров

Эксплуатация

- Обновление версий
- Долговечность
- Согласованность
- Алерты
- Поддержка
- Мониторинг

Подвиги Геркулеса

Источник: сгенерировано в Шедевруме.

YDB как платформа

Распределённое хранилище

ACID-транзакции

Федеративные запросы

Очереди

Строковые таблицы

Колоночные таблицы

Векторный поиск

Направление развития

СУБД: распределённая отказоустойчивая транзакционная SQL с функциями векторного поиска

Мы **не фокусируемся** на:

- классических больших поисковых системах
- отдельных узконаправленных проектах
- встраиваемых библиотеках

YDB: точный векторный поиск

Часть 2

HL 2025 Saint HighLoad++

Создание таблицы

```
CREATE TABLE facts (
    id Uint64,
    text String,
    user_id Uint64,
    vector Bytes,
    PRIMARY KEY (id)
)
```

Наивный подход к поиску

```
$TargetVector — искомый вектор


SELECT id, text FROM facts
WHERE user_id = 1
ORDER BY Knn::CosineDistance(vector, $TargetVector)
LIMIT 10;
```

Подборка данных

Анонимизированные логи запросов

300 млн
число векторов

1,2 ТБ
объём данных

1024 float * 32 bit
размерность векторов

Время векторного поиска

Число строк	Время, с
1000	0,005
10 000	0,02
100 000	0,3
1 000 000	1,5
10 000 000	7,5
100 000 000	78

Наивный подход к поиску

Недостатки

- Необходимость полного перебора данных

Преимущества

- Нет дополнительных структур данных
- Полная поддержка транзакций
- Мгновенное применение операций модификации данных

YDB: приближённый векторный поиск

Часть 3

Saint HighLoad++

Статическое квантование

Тип данных	Число бит
Float	32
UInt8	8
Bit	1

Таблица для приближённого поиска

```
CREATE TABLE table (
    id Uint64,
    text String,
    BitEmbedding Bytes,
    FloatEmbedding Bytes,
    PRIMARY KEY (id)
)
```

Приближённый поиск

Шаг 1: грубый поиск по битовым векторам

```
$BitIds =
SELECT id
FROM table
ORDER BY Knn::ManhattanDistance(
      BitEmbedding
      $TargetBitEmbedding)
LIMIT 100
```

Шаг 2: уточняем результаты по вещественным векторам

```
SELECT id, text, embedding
FROM table
WHERE id IN $ BitIds
ORDER BY Knn::CosineDistance(
      FloatEmbedding,
      $TargetFloatEmbedding)
LIMIT 10
```

Приближённый поиск

Число векторов, млн	Время точного поиска, с	Время приближённого поиска, с
1	1,5	0,15
100	78	2

YDB: векторный индекс

Часть 4

Saint HighLoad++

Три популярных метода

	Annoy	Faiss*	HNSW
Тип	Random Projections	Inverted Index	Graph
Используется	ClickHouse®	• PostgreSQL • Oracle	• ClickHouse® • PostgreSQL • Oracle • MongoDB Atlas • Redis Stack™ • Lucene™ Elasticsearch OpenSearch Cassandra®

* Организация Meta признана экстремистской и запрещена на территории РФ.

Почему существующие алгоритмы не подошли

1

У нас распределённая система

2

Автоматическое масштабирование

3

Консистентная транзакционная вставка

Требования

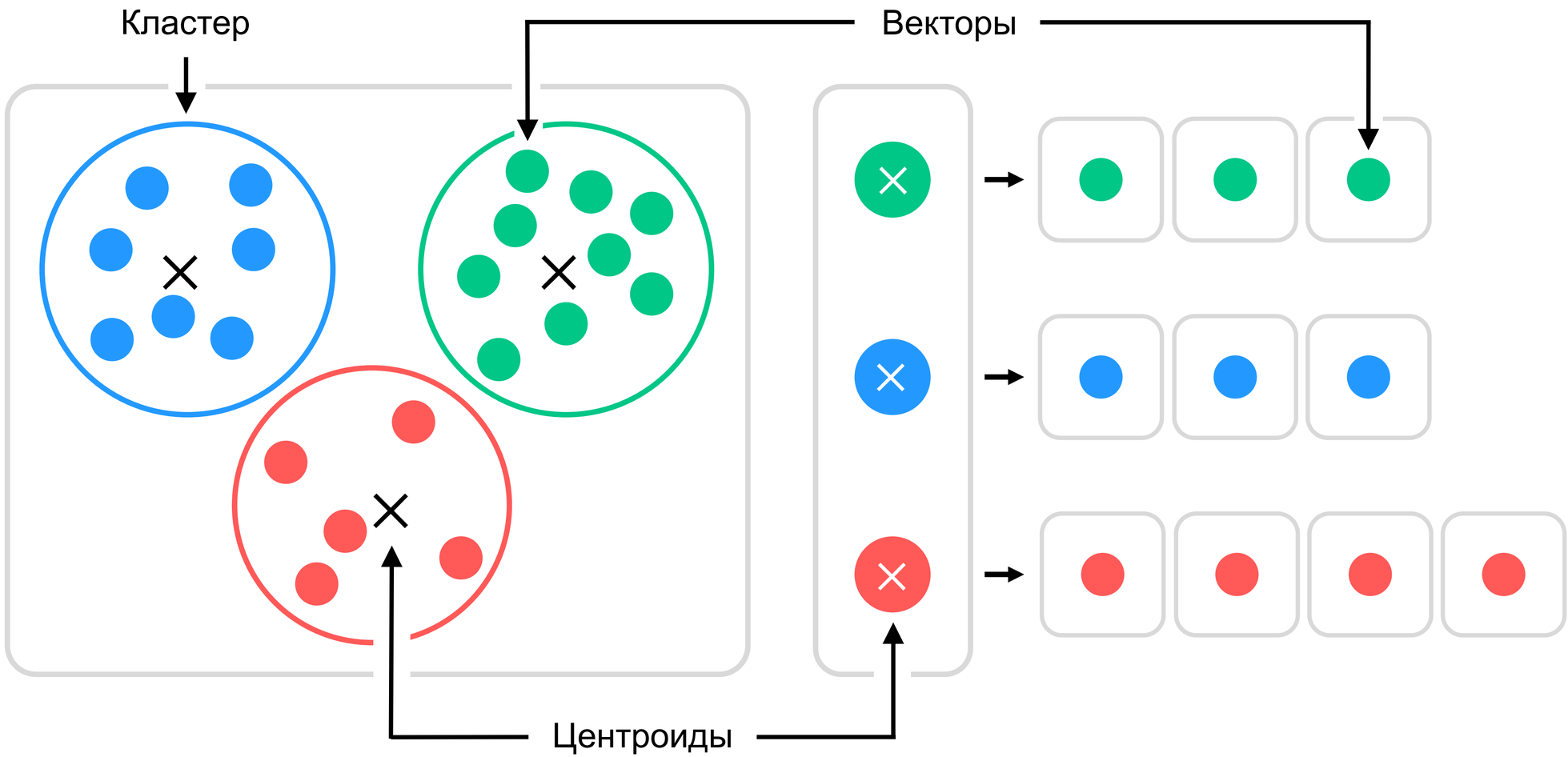
- Индекс глобальный
- Индекс обновляется синхронно с данными

- Число строк: млрд
- Запросы на выборку внутри одного ДЦ: десятки мс

- Время создания = O(размер таблицы)
- Занимаемое место = O(размер таблицы)

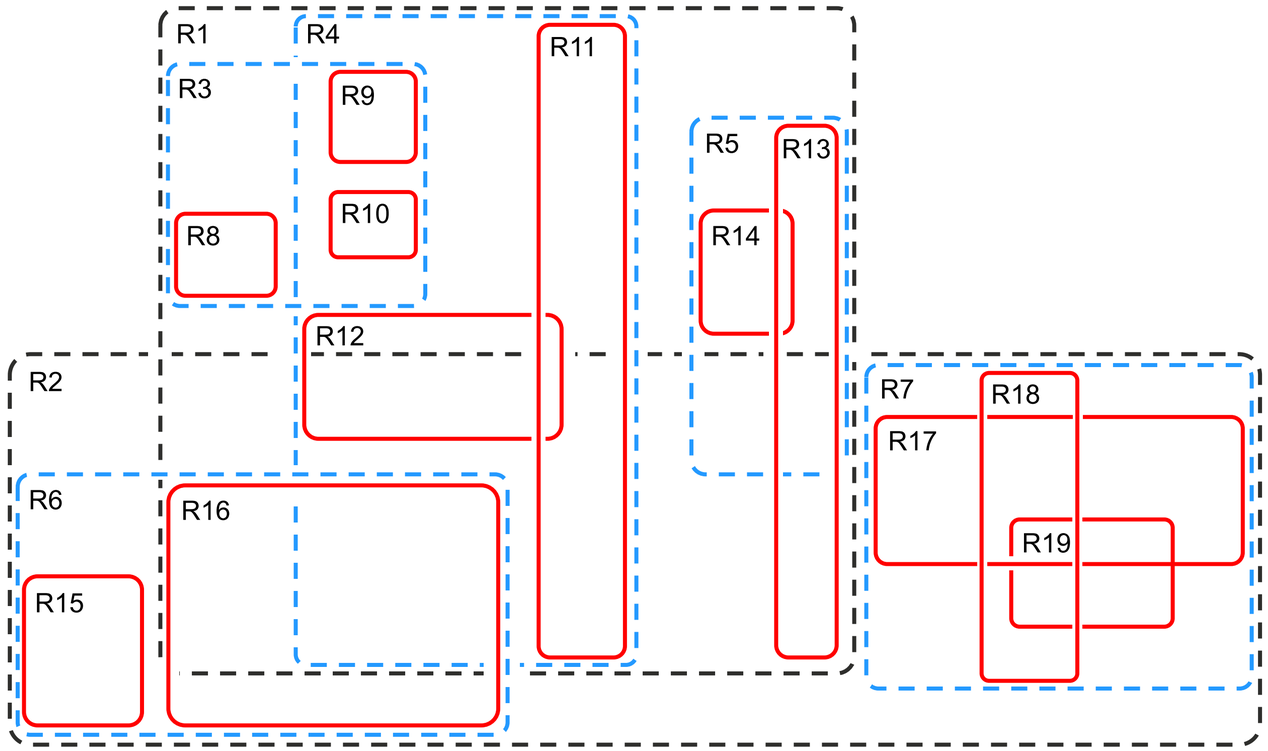
= Нужно линейно масштабироваться

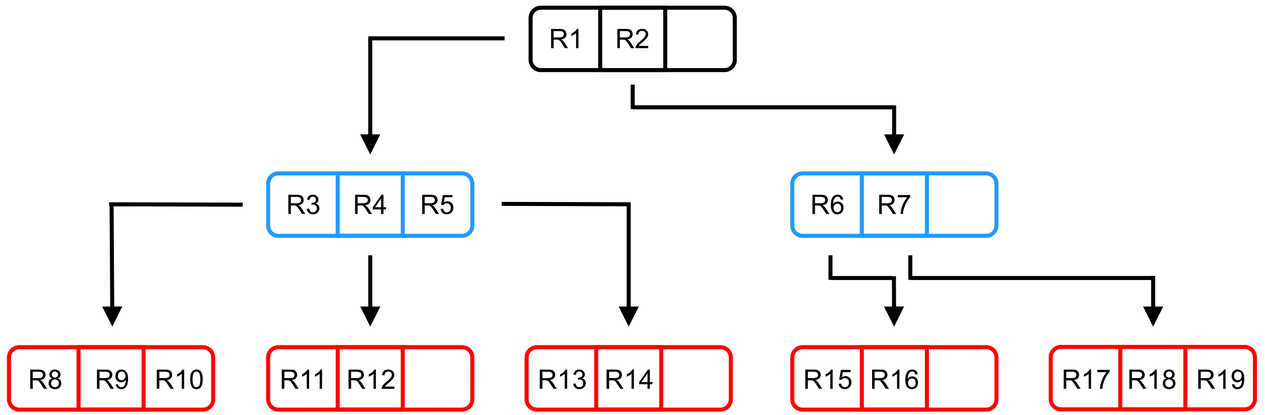
Векторный индекс как инвертированный список



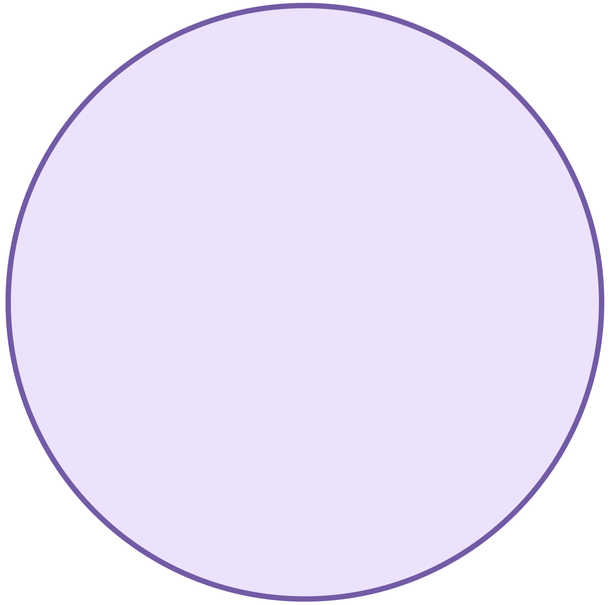
Сокращение пространства поиска

R-Tree



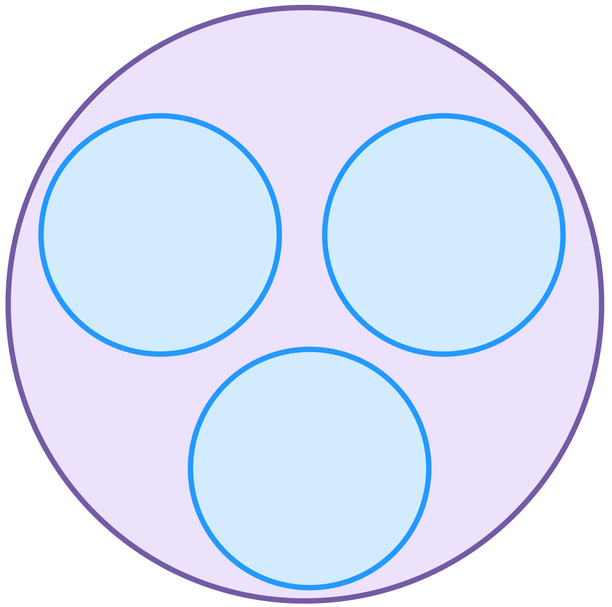


Иерархия кластеров векторного индекса



Main Table
(PK), Embedding, Covered

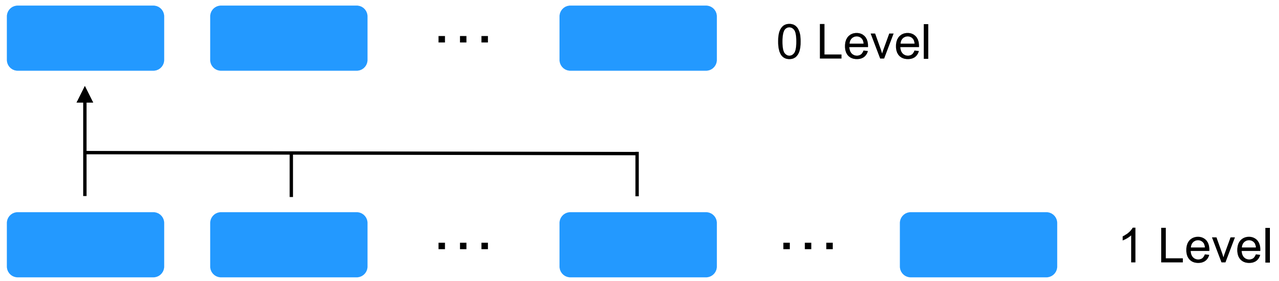
Иерархия кластеров векторного индекса

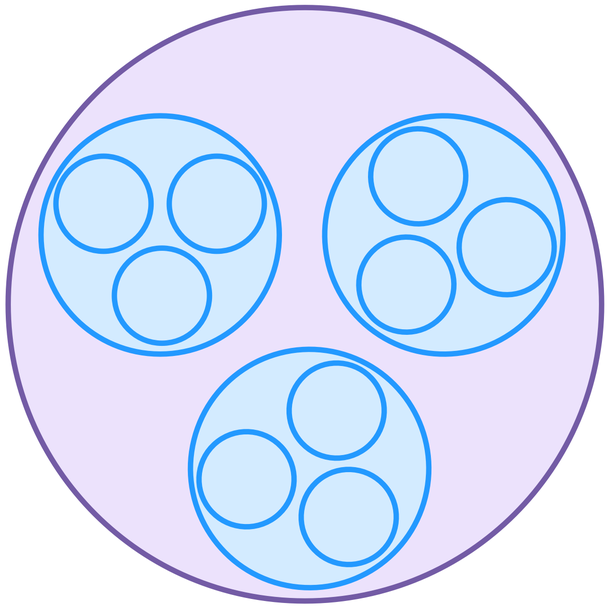


Level Table
(Parent, ID), Centroid

Main Table
(PK), Embedding, Covered

Иерархия кластеров векторного индекса

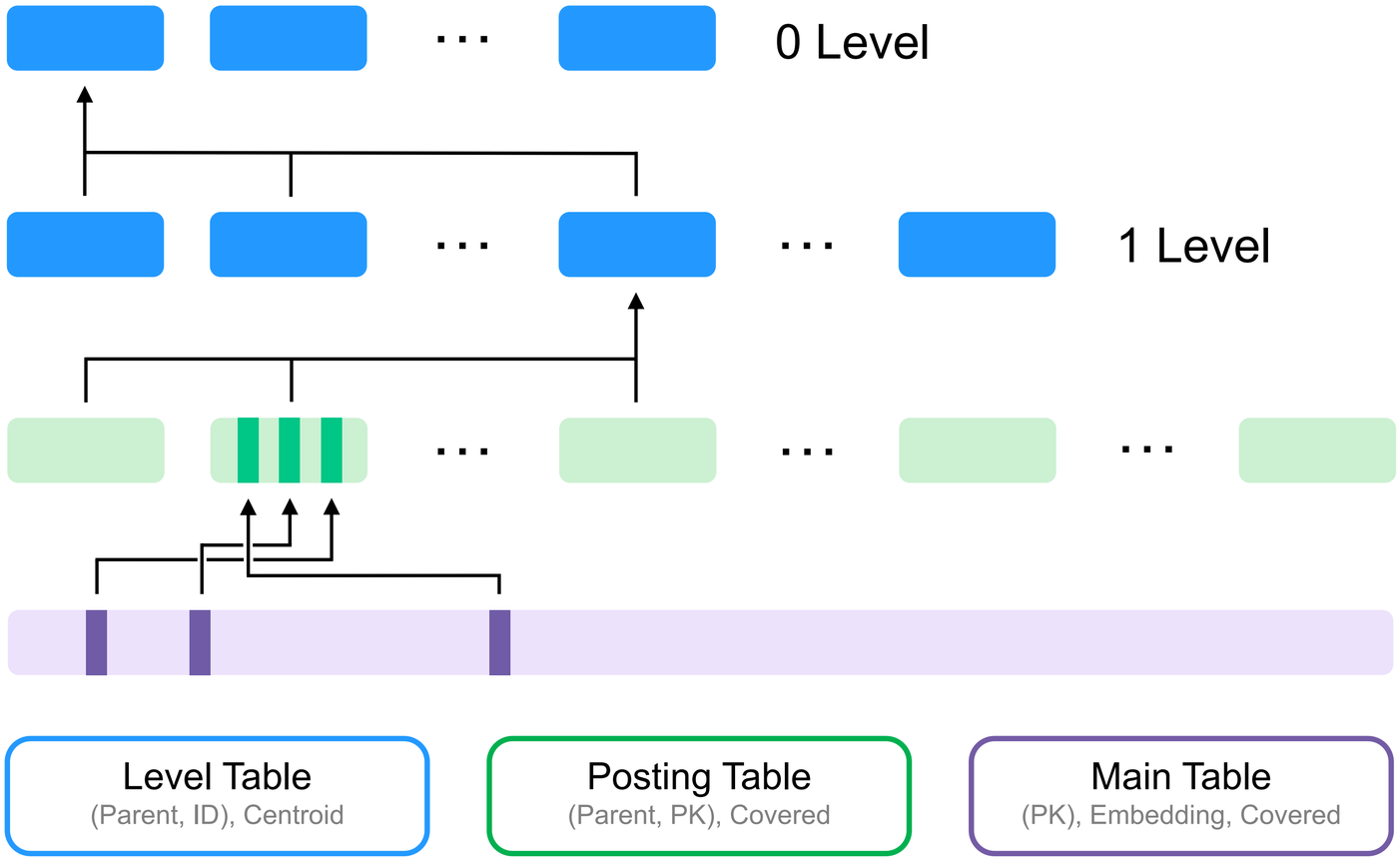


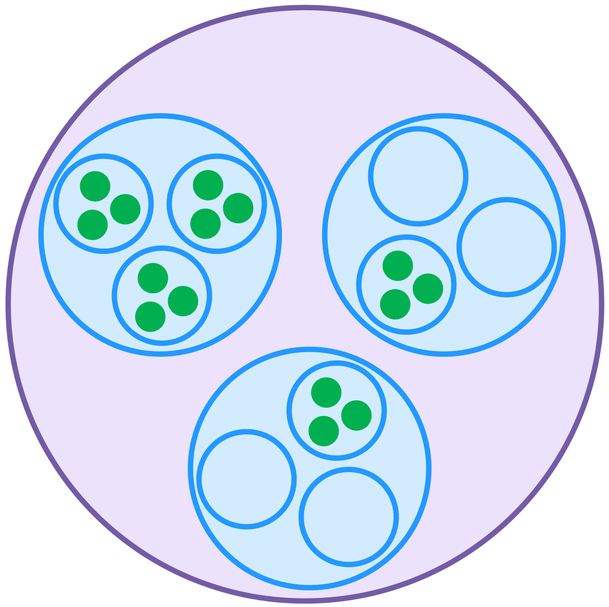


Level Table
(Parent, ID), Centroid

Main Table
(PK), Embedding, Covered

Иерархия кластеров векторного индекса





Создание векторного индекса

```
CREATE TABLE table (
        id Uint32,
        embedding Bytes,
        INDEX idx_vector
        GLOBAL USING vector_kmeans_tree
        ON (embedding)
        WITH (
                similarity = inner_product,
                vector_type = float,
                vector_dimension = 1024
        ),
        PRIMARY KEY (id)
)
```

SQL: поиск с использованием векторного индекса

```
SELECT * FROM table
VIEW idx_vector
ORDER BY Knn::CosineDistance(embedding, $Target)
LIMIT $k
```

Предварительные замеры производительности векторного индекса

Подборка	Размер таблицы	Время поиска топ-100
Wikipedia paragraphs	0,6 ГБ	13 мс
Query logs	1,2 ТБ	29 мс

Векторный индекс с фильтрацией

```
SELECT * FROM table VIEW idx_vector
WHERE user_id = $TargetUserId
ORDER BY Knn::CosineDistance(embedding, $TargetEmbedding)
LIMIT $K;
```

Векторный индекс с фильтрацией

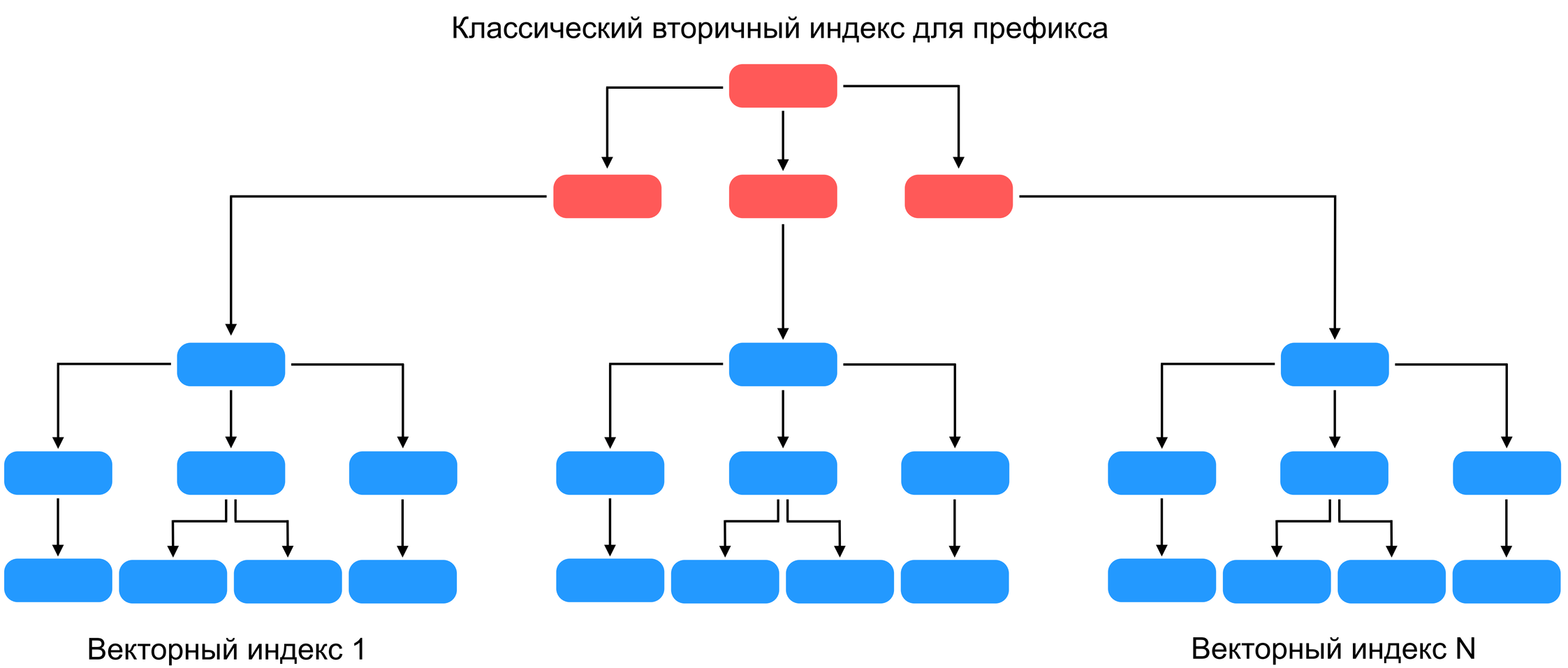
Нельзя фильтровать

- До векторного индекса — необходимость полного сканирования
- После векторного индекса — необходимость повторных запросов

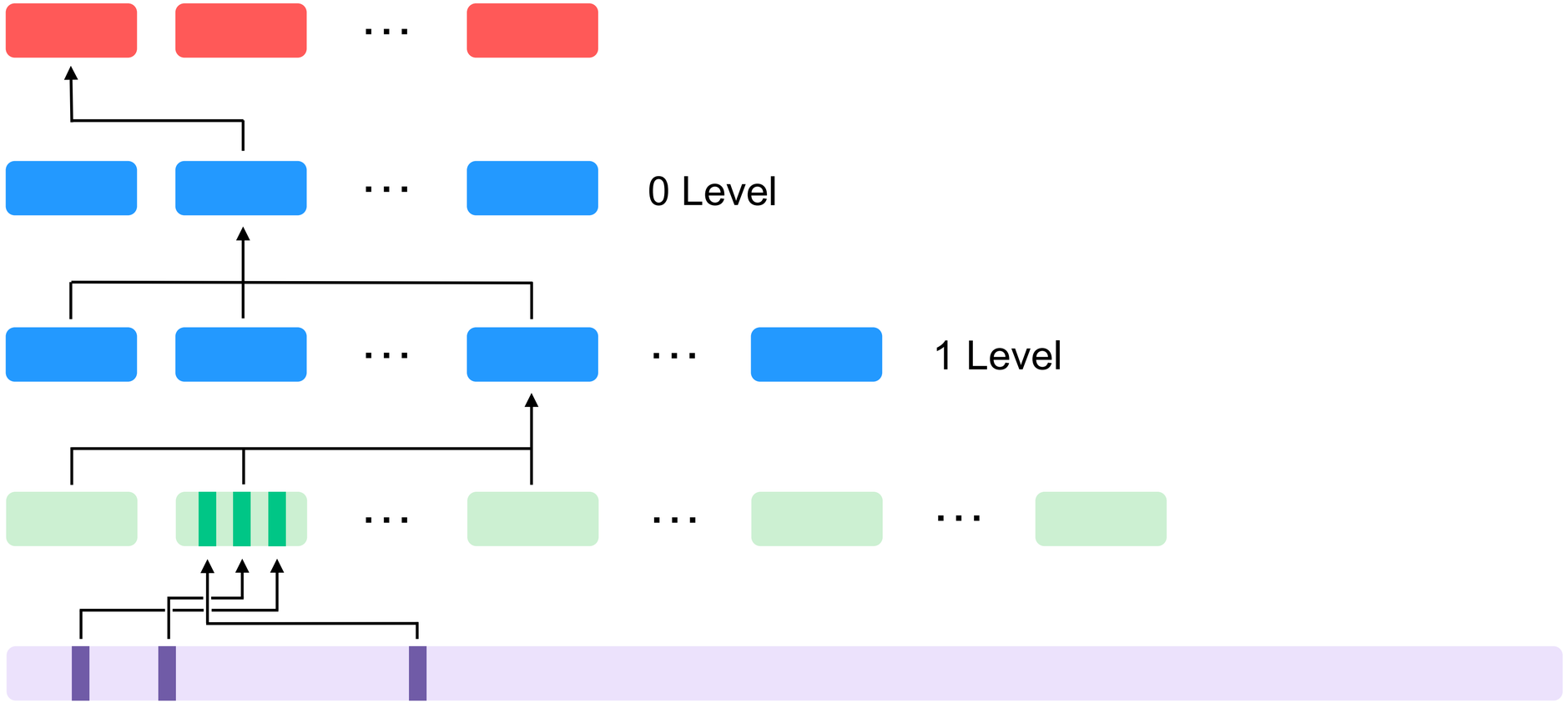
Нужно фильтровать

Внутри индекса

Векторный индекс с фильтрацией



Новый уровень с префиксной таблицей

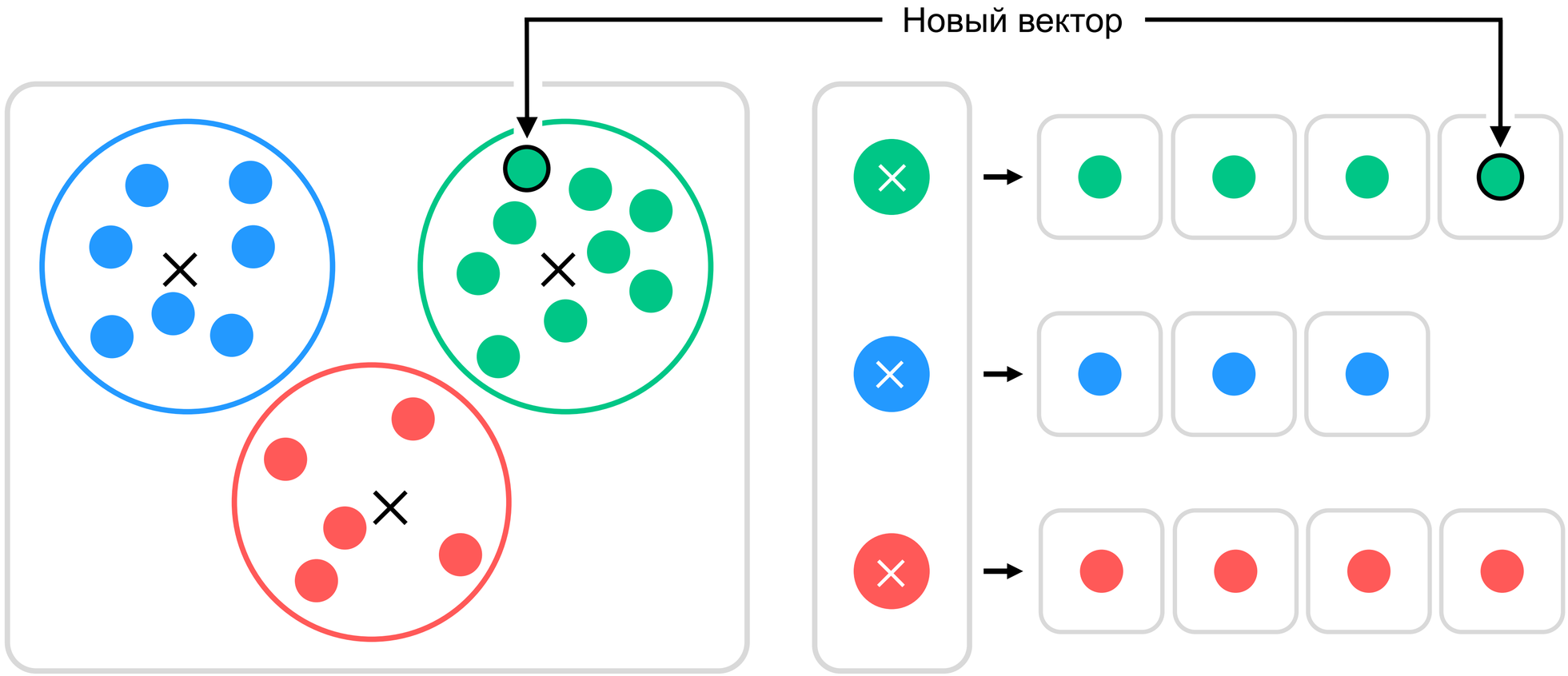


YDB: векторный индекс (в процессе)

Часть 5

Saint HighLoad++

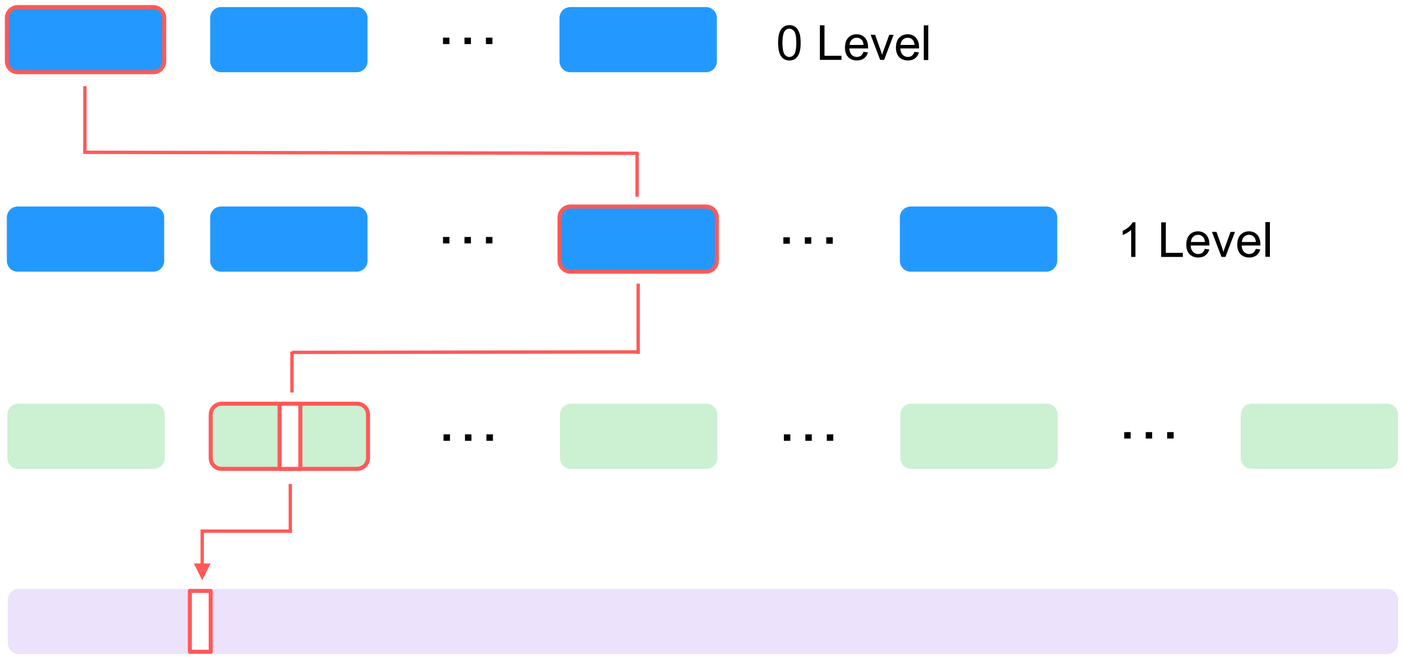
Добавление в векторный индекс



Только центроид одного кластера будет изменён = O(1)

Добавление в дерево векторного индекса

Изменение нужно протолкнуть
вверх по уровням = O(log)



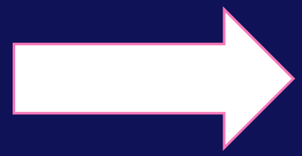
Level Table
(Parent, ID), Centroid

Posting Table
(Parent, PK), Covered

Main Table
(PK), Embedding, Covered

Итоги

- Точный векторный поиск работает неплохо
- Можно сделать его приближённым
- Глобальный векторный индекс
- Со встроенной фильтрацией по произвольным колонкам
- Доступно в Open source



Александр Зевайкин
Руководитель группы разработки, кандидат технических наук, доцент, YDB

HL 2025 Saint HighLoad++