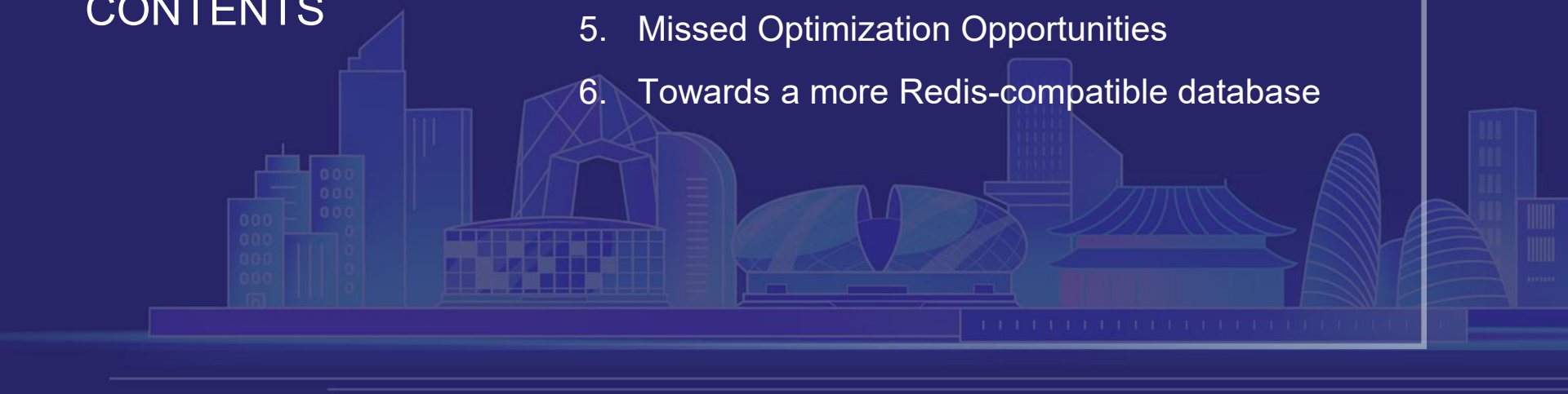
COMMUNITY
THE ASF CONFERENCE
CODE

The Evolution of **Apache Kvrocks™**: Search, Vector, and Beyond

Mingyang Liu: ASF Member, PMC Chair of Apache Kvrocks

COMMUNITY
THE ASF CONFERENCE
CODE

CONTENTS

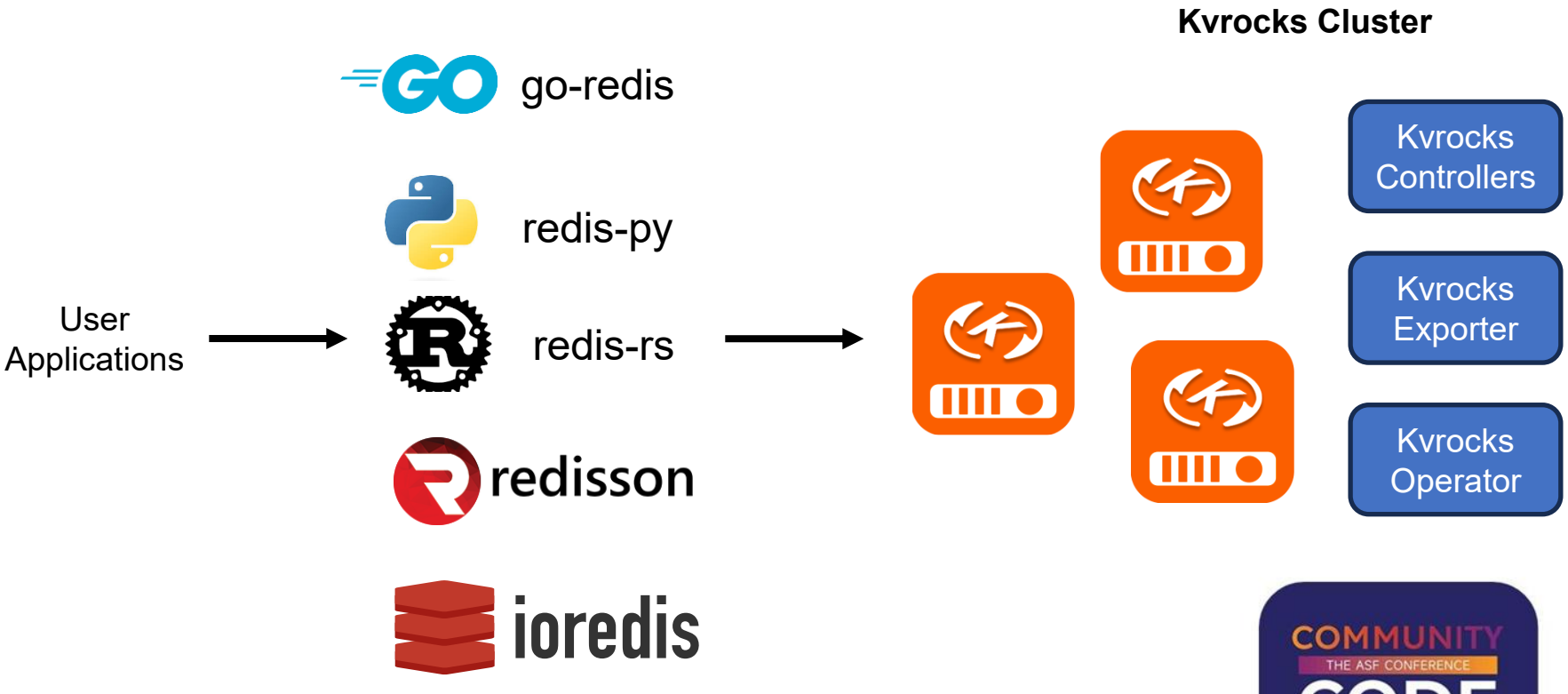


Kvrocks ↔ Redis

A **RocksDB**-based **Redis**-compatible database:

- accessible via **any existing Redis client** (via RESP 2/3)
- support **various Redis data structures** (String, Hash, List, ZSet, Stream …)
- architected with **flash storage** instead of memory-centric via RocksDB
- partially Redis-compatible **centralized cluster solution**

Kvrocks ↔ Redis



Expressive Queries

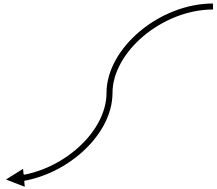
- Redis commands are NOT **composable**
- Data is NOT stored in a **structured** form
- Different data structures are NOT **interoperable**
- Hard to do **filtering**, **union/intersection** or **aggregation**
- No **general-purpose** expressions to encoding user-defined conditions
- Programmability is achieved by scripting, not in a **declarative** way

Expressive Queries

```
SELECT
    email
FROM
    users
WHERE
    membership_level = 'Premium'
    AND age > 30;
```

Expressive Queries

```
premium_users_ids = redis.SMEMBERS('premium_member_ids')

filtered_users_ids = []
for user_id in premium_users_ids:
    age_str = redis.HGET('user:ages', user_id)
    if int(age_str) > 30:
        filtered_users_ids.append(user_id)

result_emails = []
if filtered_users_ids:
    email_values = redis.HMGET('user:emails', *filtered_users_ids)
    for email in email_values:
        result_emails.append(email)
```



```
SELECT
    email
FROM
    users
WHERE
    membership_level = 'Premium'
    AND age > 30;
```

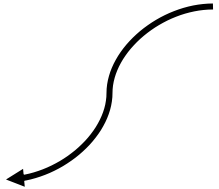
RediSearch

```
FT.CREATE users_idx ON HASH PREFIX user: SCHEMA
    email TAG
    age NUMERIC
    membership_level TAG

HSET user:101 email "a@example.com" age 25 membership_level "Basic"
HSET user:105 email "b@example.com" age 32 membership_level "Premium"
HSET ...

FT.SEARCH users_idx
    "@membership_level:{Premium} @age:[31 +inf]"
    RETURN 1 email
```

RediSearch: Schema

```
FT.CREATE users_idx ON HASH PREFIX user: SCHEMA
    email TAG
    age NUMERIC
    membership_level TAG
```

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(255),
    age INT,
    membership_level VARCHAR(50)
);

CREATE INDEX idx_membership_level ON users (membership_level);
CREATE INDEX idx_age ON users (age);
```

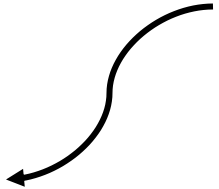


RediSearch: Schema

- TAG: exactly matching (no full-text indexing)
- NUMERIC: numeric comparison (on fp64)

```
FT.CREATE users_idx ON HASH PREFIX user: SCHEMA
    email TAG
    age NUMERIC
    membership_level TAG
```

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(255),
    age INT,
    membership_level VARCHAR(50)
);

CREATE INDEX idx_membership_level ON users (membership_level);
CREATE INDEX idx_age ON users (age);
```

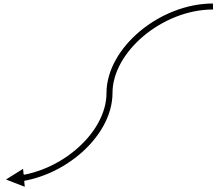


RediSearch: Schema

- one HASH or JSON key → one record
- multiple keys (prefixed by `user:*`) → a table

```
FT.CREATE users_idx ON HASH PREFIX user: SCHEMA
    email TAG
    age NUMERIC
    membership_level TAG
```

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(255),
    age INT,
    membership_level VARCHAR(50)
);

CREATE INDEX idx_membership_level ON users (membership_level);
CREATE INDEX idx_age ON users (age);
```

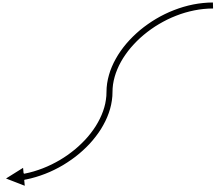


RediSearch: Schema

- indexing each field by default
- add NOINDEX to disable

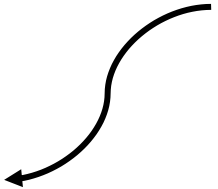
```
FT.CREATE users_idx ON HASH PREFIX user: SCHEMA
    email TAG
    age NUMERIC
    membership_level TAG
```

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(255),
    age INT,
    membership_level VARCHAR(50)
);

CREATE INDEX idx_membership_level ON users (membership_level);
CREATE INDEX idx_age ON users (age);
```



RediSearch: Inserting

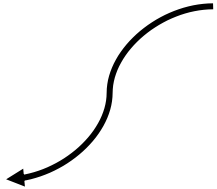
```
HSET user:101 email "a@example.com" age 25 membership_level "Basic"
HSET user:105 email "b@example.com" age 32 membership_level "Premium"
HSET ...
```



```
INSERT INTO users (user_id, email, age, membership_level) VALUES
(101, 'a@example.com', 25, 'Basic'),
(105, 'b@example.com', 32, 'Premium');
-- ...
```

RediSearch: Query

```
FT.SEARCH users_idx
    "@membership_level:{Premium} @age:[31 +inf]"
    RETURN 1 email
```

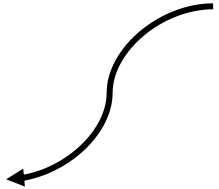


```
SELECT
    email
FROM
    users
WHERE
    membership_level = 'Premium'
    AND age > 30;
```

RediSearch: Query

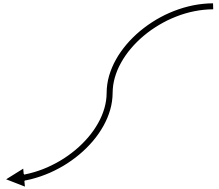
```
FT.SEARCH users_idx
    "@membership_level:{Premium} @age:[31 +inf]"
    RETURN 1 email
```

```
SELECT
    email
FROM
    users
WHERE
    membership_level = 'Premium'
    AND age > 30;
```



RediSearch: Query

```
FT.SEARCH users_idx
    "@membership_level:{Premium} @age:[31 +inf]"
    RETURN 1 email
```

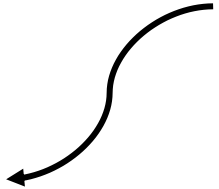


```
SELECT
    email
FROM
    users
WHERE
    membership_level = 'Premium'
    AND age > 30;
```

RediSearch: Query

```
FT.SEARCH users idx
    "@membership_level:{Premium} @age:[31 +inf]"
    RETURN 1 email
```

```
SELECT
    email
FROM
    users
WHERE
    membership_level = 'Premium'
    AND age > 30;
```



RediSearch: Query Language

- `@field:{ tag1 | tag2 | … }`
  - Exactly match if `field` is `tag1`, `tag2` or …
  - `field` must be typed TAG
- `@field:[min max]`
  - `field` is in the range between `min` and `max`; `±inf` as infinity
  - closed interval unless `(` is prefixed, e.g. `[(1 10]` means $1 < v \leq 10$
  - `field` must be typed NUMERIC
- AND: `@field:.. @field:..`; OR: `@field:.. | @field:..`
- NOT: `-@field:..`

Story of Kvrocks Search: The Capability to Query

Redis is mostly used as a **cache**, but Kvrocks may NOT.

Hence capability to do **complex queries** is more vital.

Among relational databases and NoSQL databases:

trade-offs between **performance** and **expressiveness**.

Kvrocks want to explore more than a key-value database.

Story of Kvrocks Search: Redis Query and SQL

Redis is moving closer to SQL via RediSearch!

- Complex queries on **semi-structured** data
- A **unique** query syntax than SQL
- A **growing** ecosystem: client library supporting, RedisVL
- Utilized in **LangChain** for Retrieval-Augmented Generation

Support its syntax and inherit its ecosystem!

Story of Kvrocks Search: Redis Query and SQL

But wait, why not SQL?

- Already **familiar** to everyone
- The syntax is more **intuitive** and generic
- Aligns semantically with RediSearch queries
- Supports more features in a **consistent** form
- Redis query syntax is NOT well-designed, e.g. precedence of AND and OR changes across different version

So, we want them both!

Kvrocks Search

```
FT.CREATE users_idx ON HASH PREFIX user: SCHEMA
    email TAG
    age NUMERIC
    membership_level TAG

HSET ...

FT.SEARCH users_idx
    "@membership_level:{Premium} @age:[31 +inf]"
    RETURN 1 email

FT.SEARCHSQL "SELECT email FROM users_idx
    WHERE membership_level TAGGED 'Premium' AND age > 30"
```

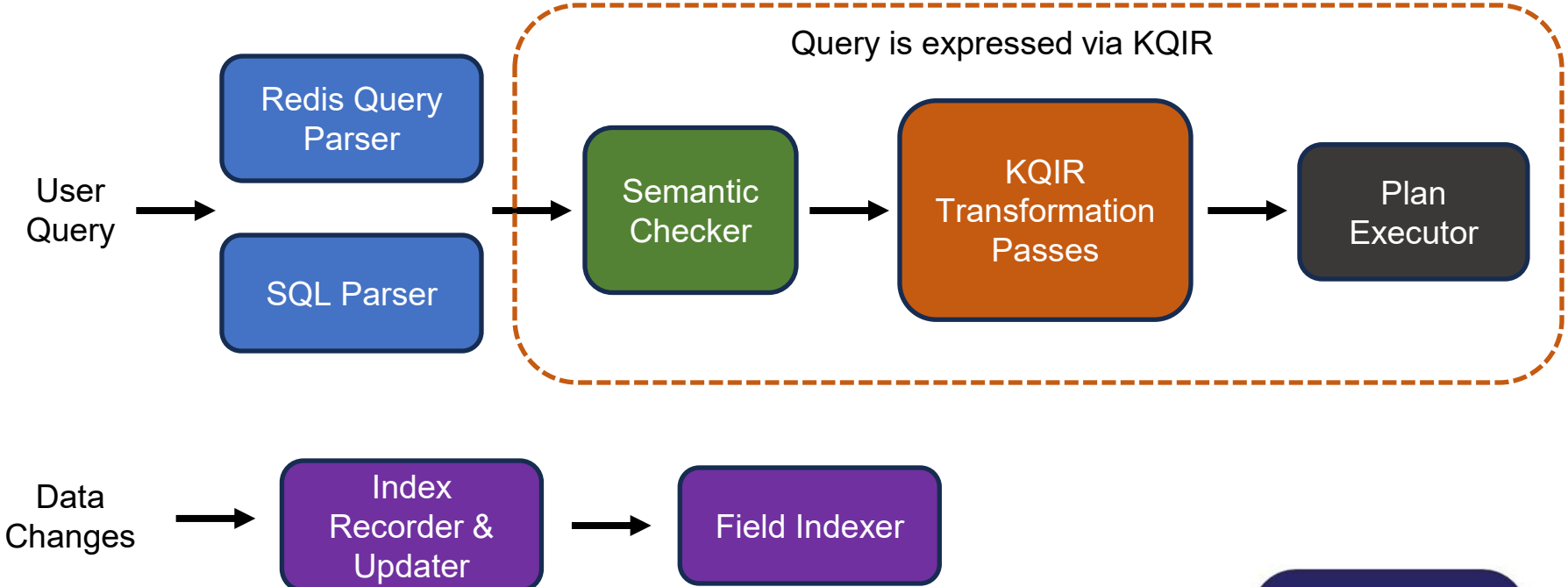
Kvrocks Search

```
FT.CREATE users_idx ON HASH PREFIX user: SCHEMA
    email TAG
    age NUMERIC
    membership_level TAG

HSET ...

FT.SEARCH users_idx
    "@membership_level:{Premium} @age:[31 +inf]"
    RETURN 1 email

FT.SEARCHSQL "SELECT email FROM users_idx
    WHERE membership_level TAGGED 'Premium' AND age > 30"
```

Query via Redis-compatible syntax

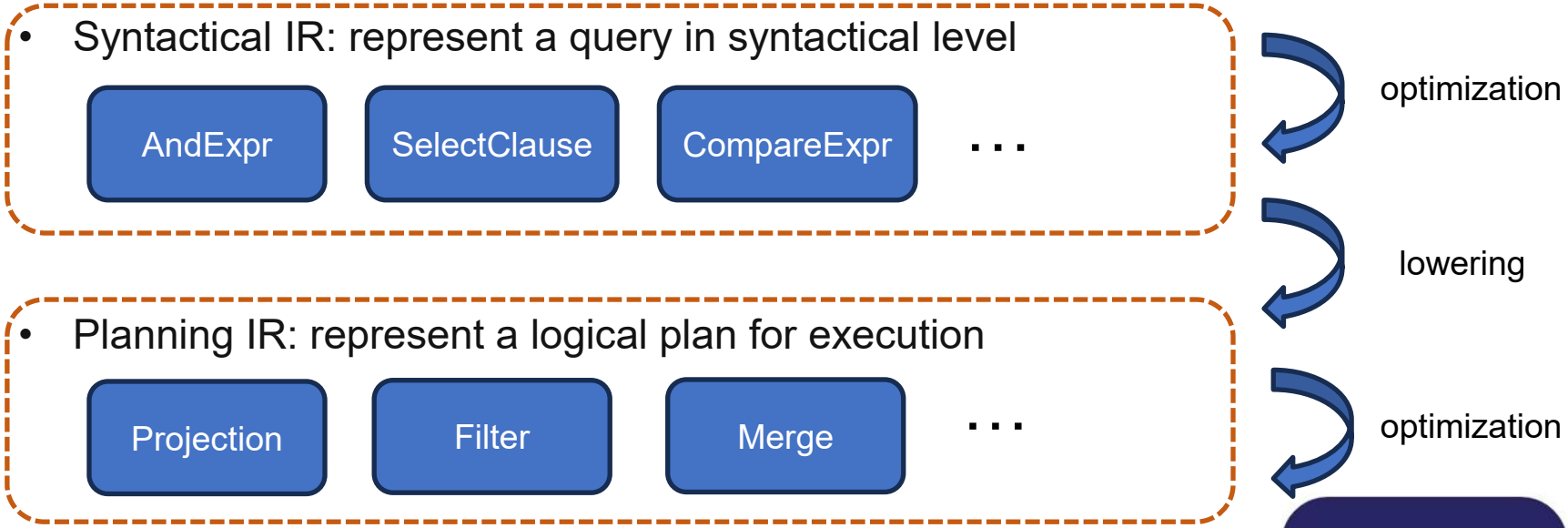
Query via SQL!

Design of Kvrocks Search



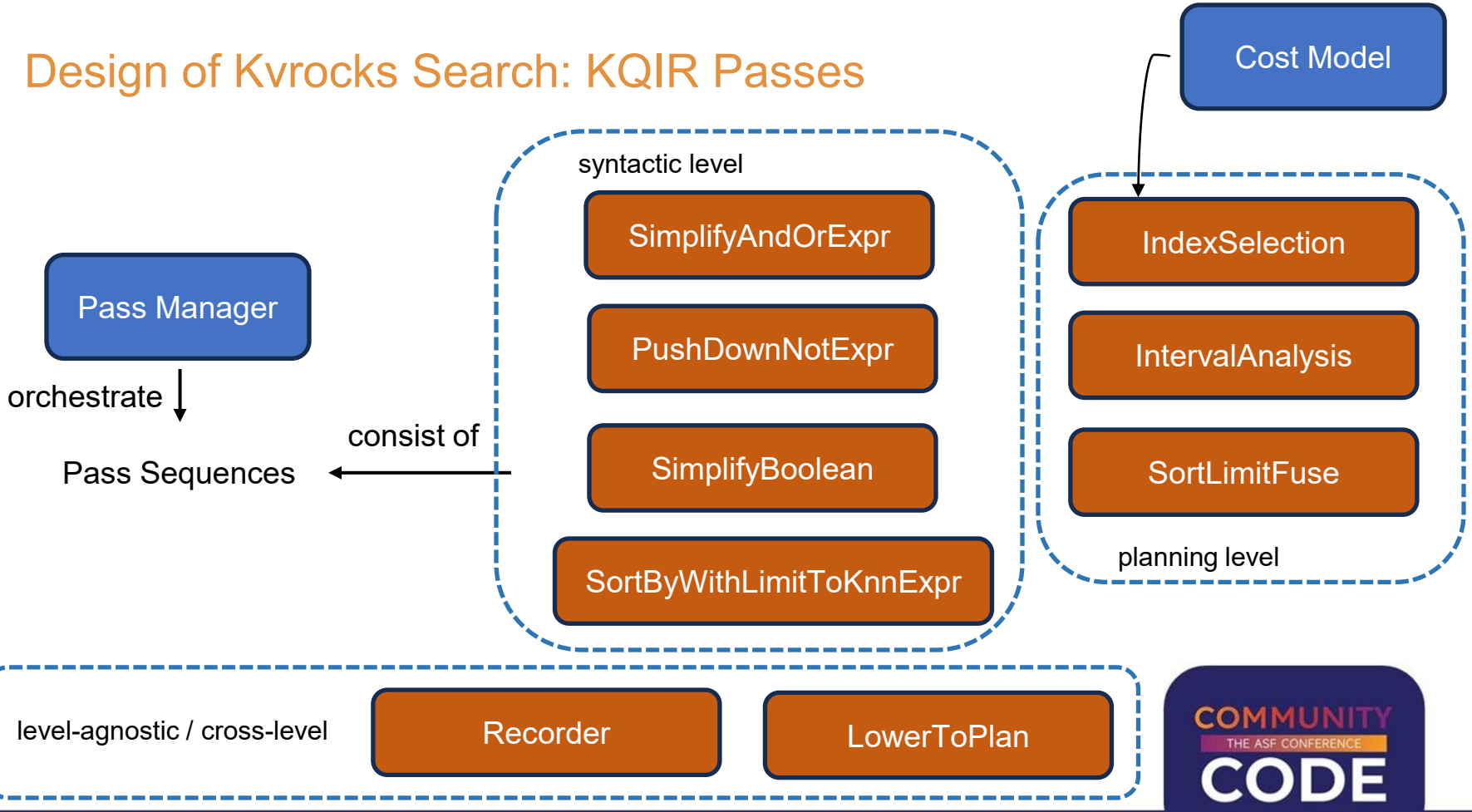
Design of Kvrocks Search: KQIR

KQIR is a multiple-level **I**ntermediate **R**epresentation for **Q**ueries in **K**vrocks:



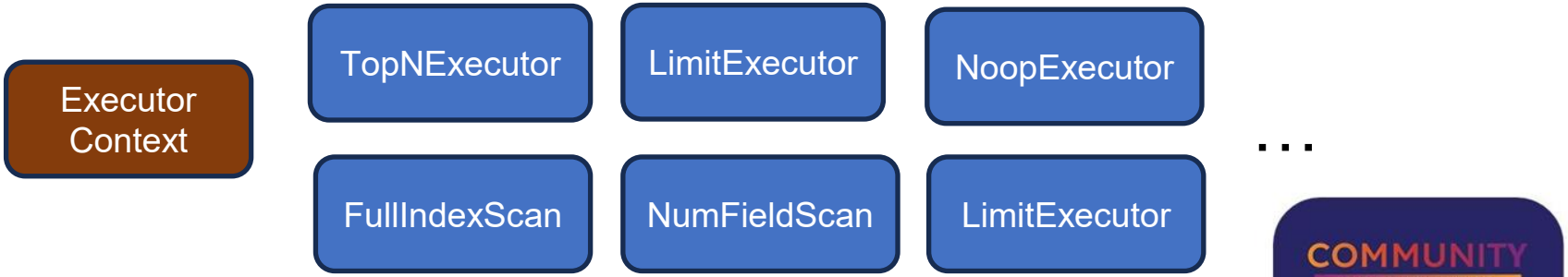
KQIR is transformed by optimization/lowering passes between levels.

Design of Kvrocks Search: KQIR Passes



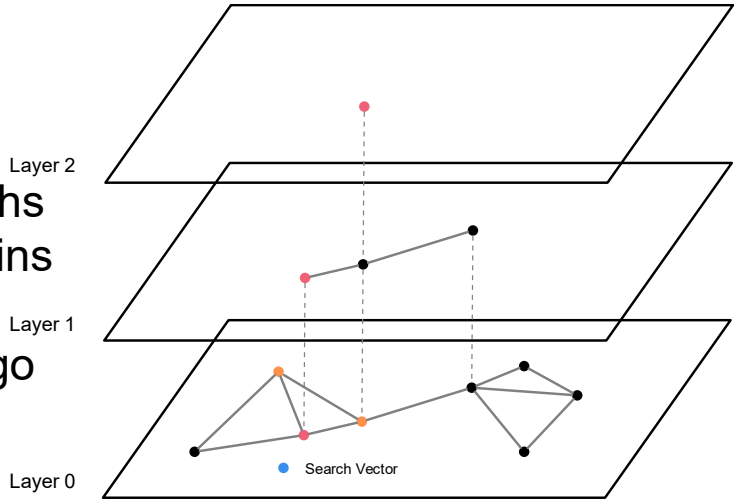
Design of Kvrocks Search: Indexer and Executor

- A new column family and encoding design for index data
- Data changes is captured before and after command exection for index updating
- Plan executor is built on the Volcano model (nothing fancy)



Vector Search via HNSW

- HNSW is a layered structure of neighbor graphs
- Layer 0 contains all nodes; higher layer contains fewer nodes, i.e. a “skiplist” built upon layer 0
- Search from the top layer, and progressively go down to lower layers
- Encoded into RocksDB key-values



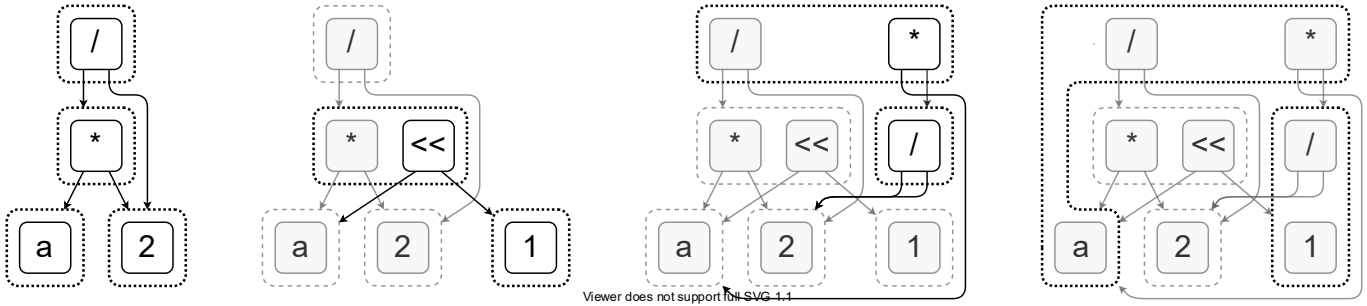

```
.. -> | field flag | vector type | dimension | distance metric | initial cap | m |
      | ef construction | ef runtime | epsilon | number of levels |

.. | level | NODE | user key | -> | num of neighbours | vector dimension | vector data |
.. | level | EDGE | source key | target key | -> null
```

Current Status

- Supported field types: TAG, NUMERIC and VECTOR; no TEXT support
- Transaction guarantee is experimental and can be enabled by an config option
- Work well with namespace feature, but disabled in cluster mode currently
- Vector inserting performance is limited due to an index-level lock
- Currently no runtime query statistics (e.g. histogram) available for cost analysis
- Still quite experimental

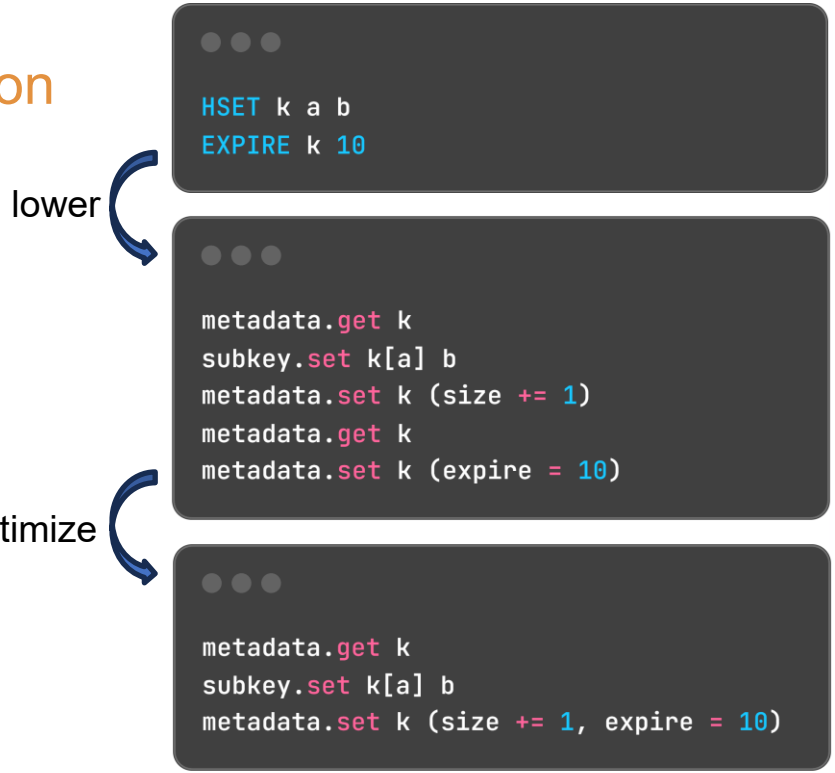
Missed Opportunities: Equality Saturation

- Trivial term rewriting in passes faces the *phase ordering* problem
- E-classes of expression DAGs (e-graphs) to achieve “global optimum”
- Works well for rewriting rules of arithmetic and relational algebra



Missed Opportunities: Low-level Fusion

- Fusion opportunities may appear while lowering IR to lower levels
- IO operations can be reduced by such low-level fusions
- Requires looking ahead



Towards a more Redis-compatible database

Enhanced compatibility on basic features:

- BITFIELD commands (since 2.7.0)
- RDB dump/restore support (since 2.9.0)
- Cluster hint commands: READONLY, READWRITE, ASKING (since 2.9.0)
- Key-related commands: SORT, MOVE, COPY, RENAME (since 2.9.0)
- Stream group support (since 2.10.0)

refer to https://kvrocks.apache.org/docs/supported-commands

Towards a more Redis-compatible database

Advanced Redis-compatible features:

- Bloom filter (since 2.6.0)
- JSON (since 2.7.0)
- Functions (since 2.7.0)
- Search (since 2.11.0)
- t-digest (since 2.12.0?)
- Time series (WIP)

refer to https://kvrocks.apache.org/docs/supported-commands

Try it yourself!

```
$ docker run -p 6666:6666 apache/kvrocks

$ redis-cli -p 6666
> echo "Do whatever you want!"
"Do whatever you want!"
>
```

Join the community (and welcome to contribute!)

Website: https://kvrocks.apache.org

Zulip Chat: https://kvrocks.zulipchat.com

Kvrocks: https://github.com/apache/kvrocks

Controller: https://github.com/apache/kvrocks-controller

Kvrocks

COMMUNITY
THE ASF CONFERENCE
CODE

Thanks

Mingyang Liu