Execution Engine Team #2

Christos Laspias
Hyoungjoo Kim
Yash Kothari

Goals

75%	100%	125%
Pull-Based	Push-Based	Multithreading
Basic Operators	More Operators	DB Integration
Vectorized Execution	TPCH-Support	Optimizations
Unit Tests	End-to-End Correctness	Performance

What we did

Implemented our own Operators

Push-Based, Vectorized Execution

Execute Multiple pipelines (end-to-end queries)

Translate Datafusion Physical Plans to our internal format (visitor)

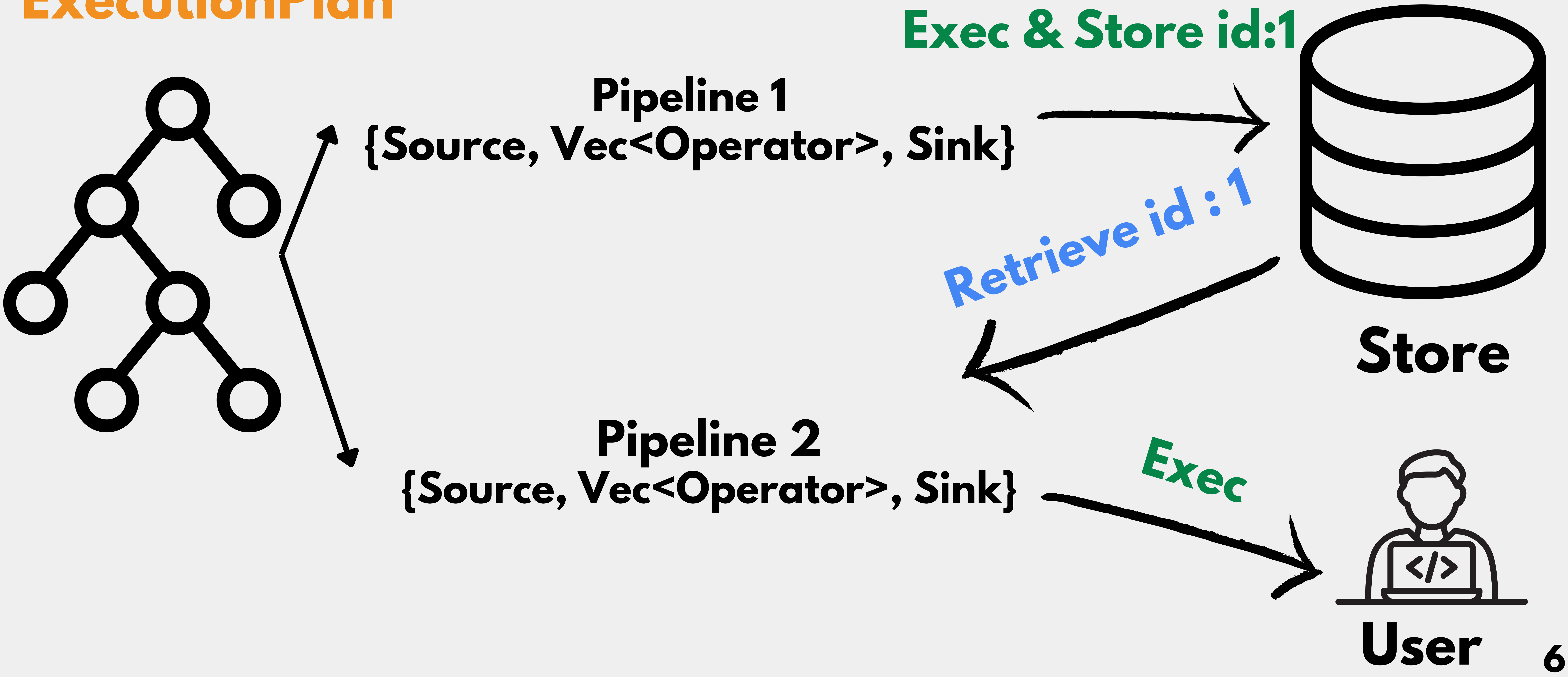
Push-Based Pipelined Execution

- How do we create pipelines ?
- Where do pipelines store data ?
- How does the next pipeline find the data produced by previous pipelines ?

Push-Based Pipelined Execution

1. Break the plan into pipelines
2. Execute pipelines in order
3. Each sink in each pipeline stores the intermediate results in a hash map
4. Next pipeline retrieves intermediate results from hash map and continues execution.

System Design



Datafusion ExecutionPlan

Pipeline 1
{Source, Vec<Operator>, Sink}

Exec & Store id:1

Retrieve id : 1

Store

Pipeline 2
{Source, Vec<Operator>, Sink}

Exec

User

Operators Supported

- ParquetScan
- Projection
- Filter
- Hash Join Build/Probe
- Hash Aggregation
- Limit
- Sort

Operator Implementation

- Utilize Datafusion library
    - ParquetScan + filter/projection pushdown
    - Filter, Sort
- Straightforward
    - Projection, Limit
- Custom Hashmap
    - HashAggregate, HashJoinBuild/Probe

Custom Hashmap

batch 0

ID	Name	Grade
7	foo	A
8	bar	B
9	baz	A

batch 1

ID	Name	Grade
10	fux	C
11	rab	A
12	zab	B

Key	List <(batch_id, row_idx)>
A	(0, 0), (0, 2), (1, 1)
B	(0, 1), (1, 2)
C	(1, 0)

Alternatives?

1. Concat_batches & List<row_idx>
   - Memcpy required
2. List<List<row_idx>>
   - If each key exists at 1-2 rows, wastes memory

Testing

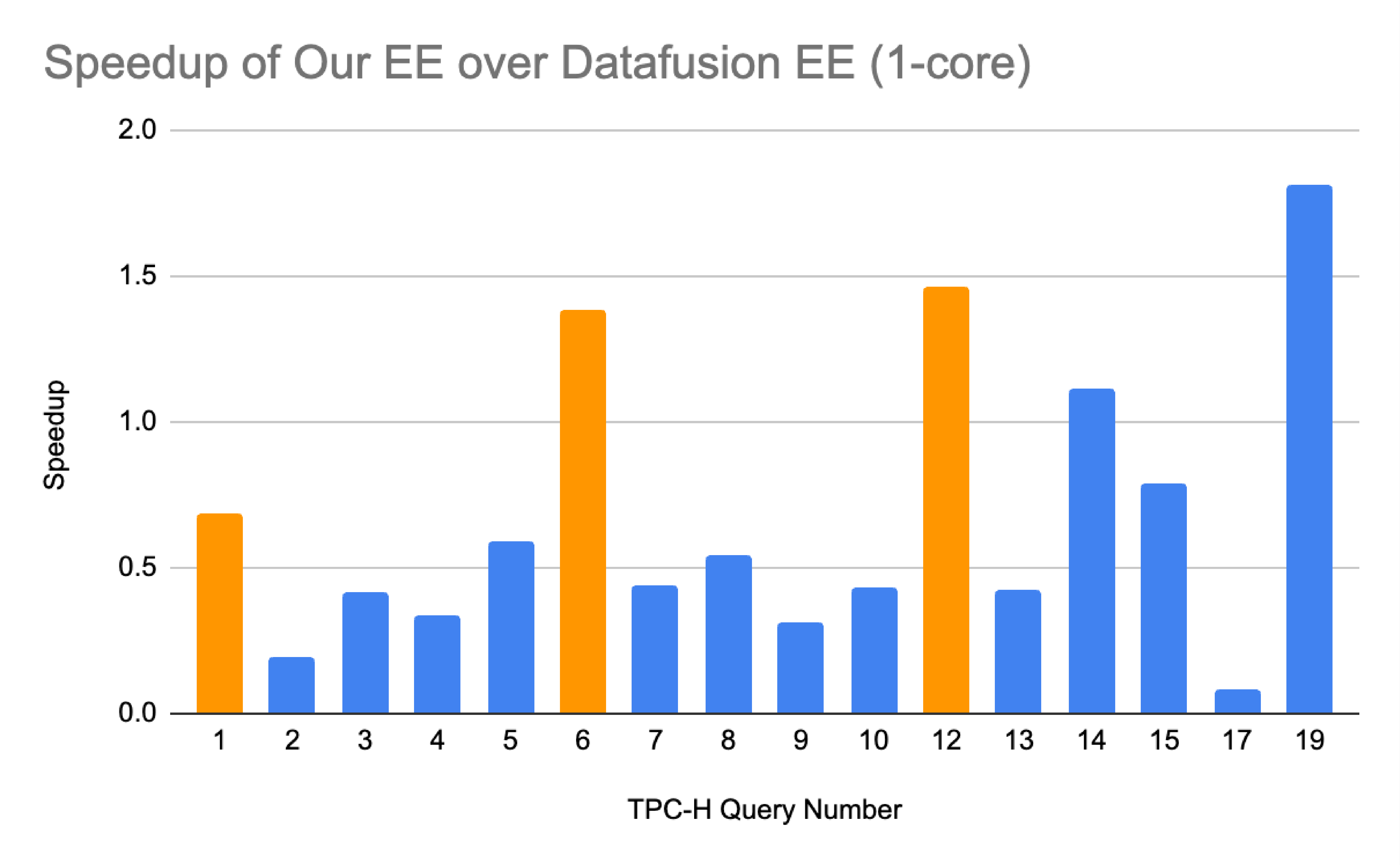
1. Unit tests for individual opreators
2. End-to-end test
   a. SQL → Datafusion frontend + optimizer
   b. Execute the plan on “Our EE” & “Datafusion EE”
   c. Compare the result & runtime

TPC-H Result

Out of 22 TPC-H queries,

- **16 runs without panic**
- **3 produces correct result**

TPC-H Performance



Correct Result

Different Result

Strengths / Weaknesses

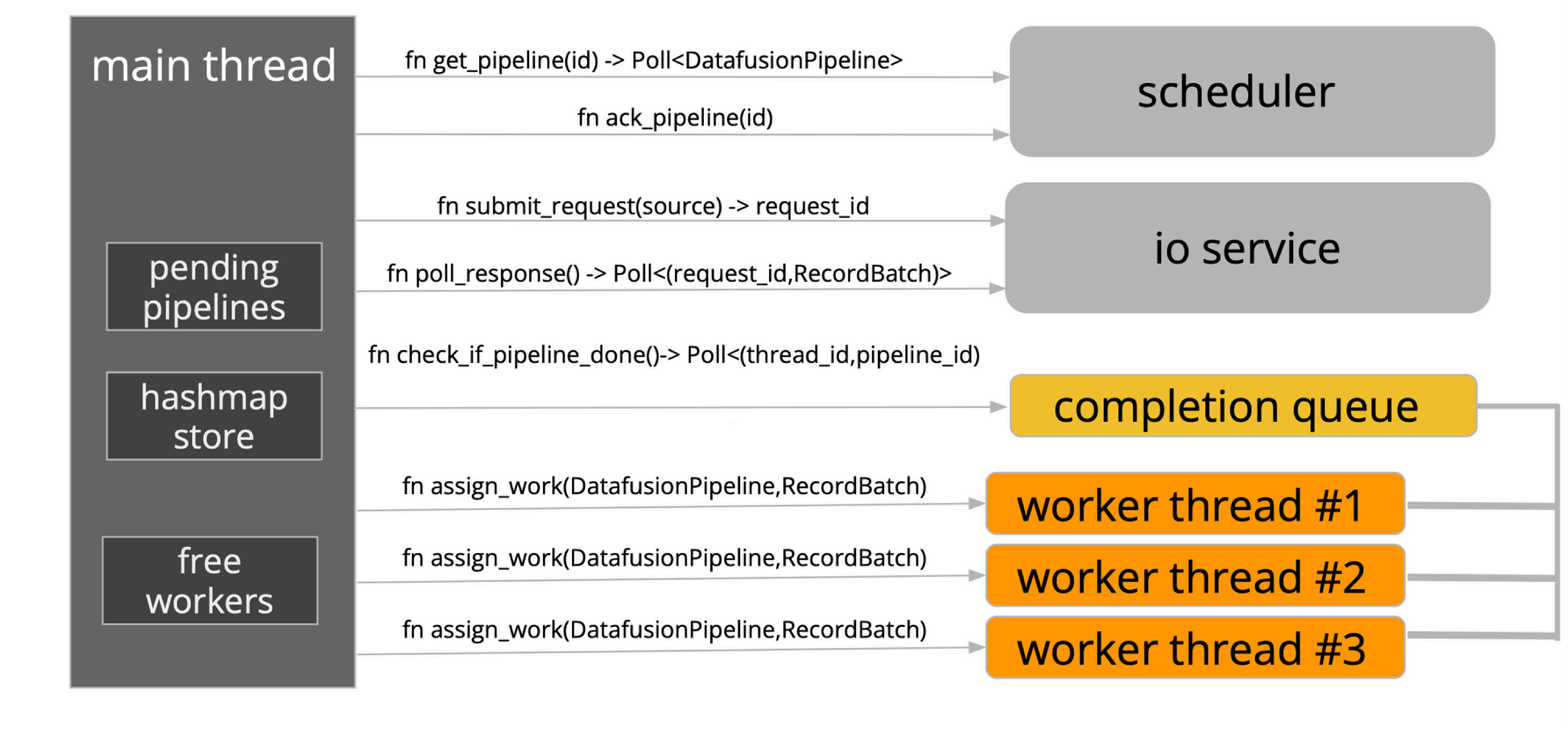
Strengths

- Efficient push-based model
- Our own specialized operators

Weaknesses

- Not fully compatible with Datafusion Execution Plan (e.g. unsupported nodes & options)
- Some of our operations on Arrow are not optimal

Intra Query Parallelism (In Progress)



main thread

pending pipelines

hashmap store

free workers

fn get_pipeline(id) -> Poll<DatafusionPipeline>

fn ack_pipeline(id)

scheduler

fn submit_request(source) -> request_id

fn poll_response() -> Poll<(request_id,RecordBatch)>

io service

fn check_if_pipeline_done()-> Poll<(thread_id,pipeline_id)

completion queue

fn assign_work(DatafusionPipeline,RecordBatch)

worker thread #1

fn assign_work(DatafusionPipeline,RecordBatch)

worker thread #2

fn assign_work(DatafusionPipeline,RecordBatch)

worker thread #3

Future Works

1. Compatibility with other microservices
2. More plan nodes, More tests
3. Intra query parallelism
4. Faster kernels on Arrow RecordBatch
5. Profile & Optimize

Thank you!