TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Termal Defect Detection

Systems and Architectures for Big Data

A.Y. 2024/2025

Martorelli Luca
0357263

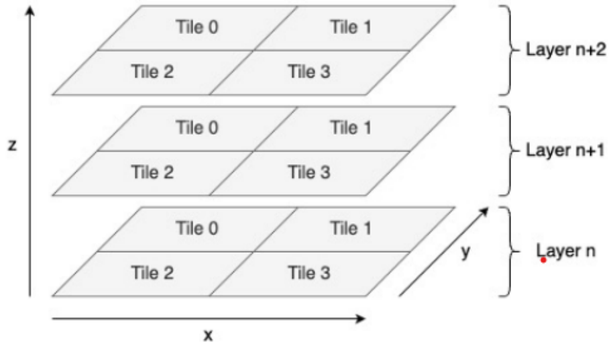
Iurato Chiara
0341741

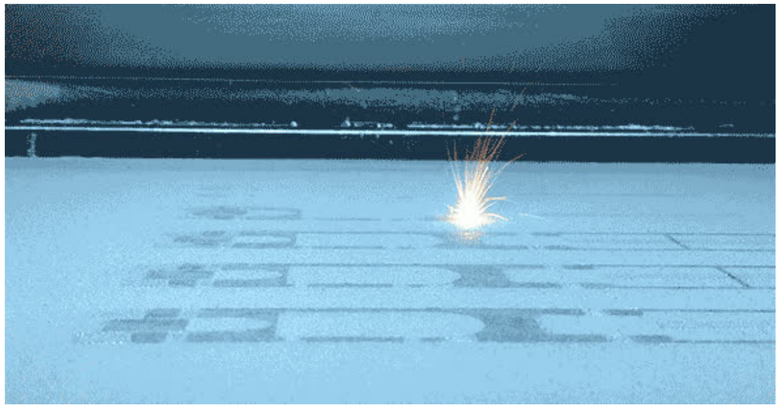
Cortese Alessandro
0350035

Agenda

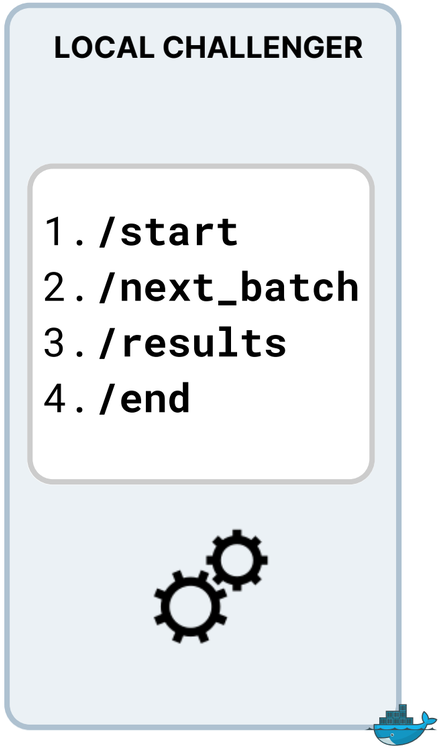
Introduction

- L-PBF (Laser Powder Bed Fusion) is a metal 3D printing technique that fuses powder layer by layer using a laser, enabling design flexibility and minimizing material waste.
- However, it can produce critical defects (e.g., porosity) often detected only after production, leading to wasted time and resources.
- To address this, Optical Tomography (OT) enables real-time monitoring by capturing thermal images layer by layer during printing.
- OT images reveal temperature distributions, allowing early detection of thermal anomalies and timely interventions.
- Each data batch is a stream of high-resolution 16-bit TIFF OT images, divided into tiles representing local blocks of each printed layer.

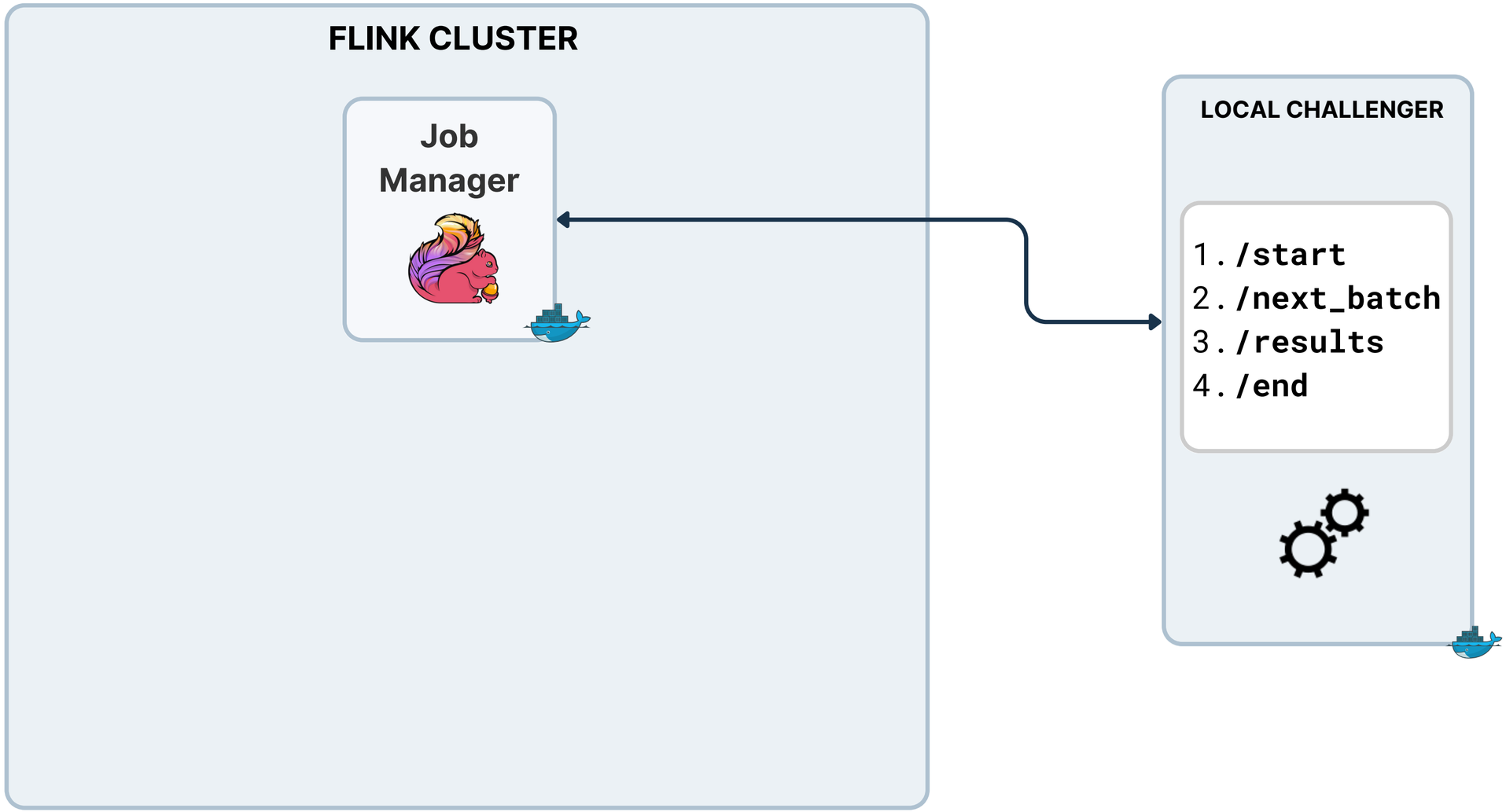




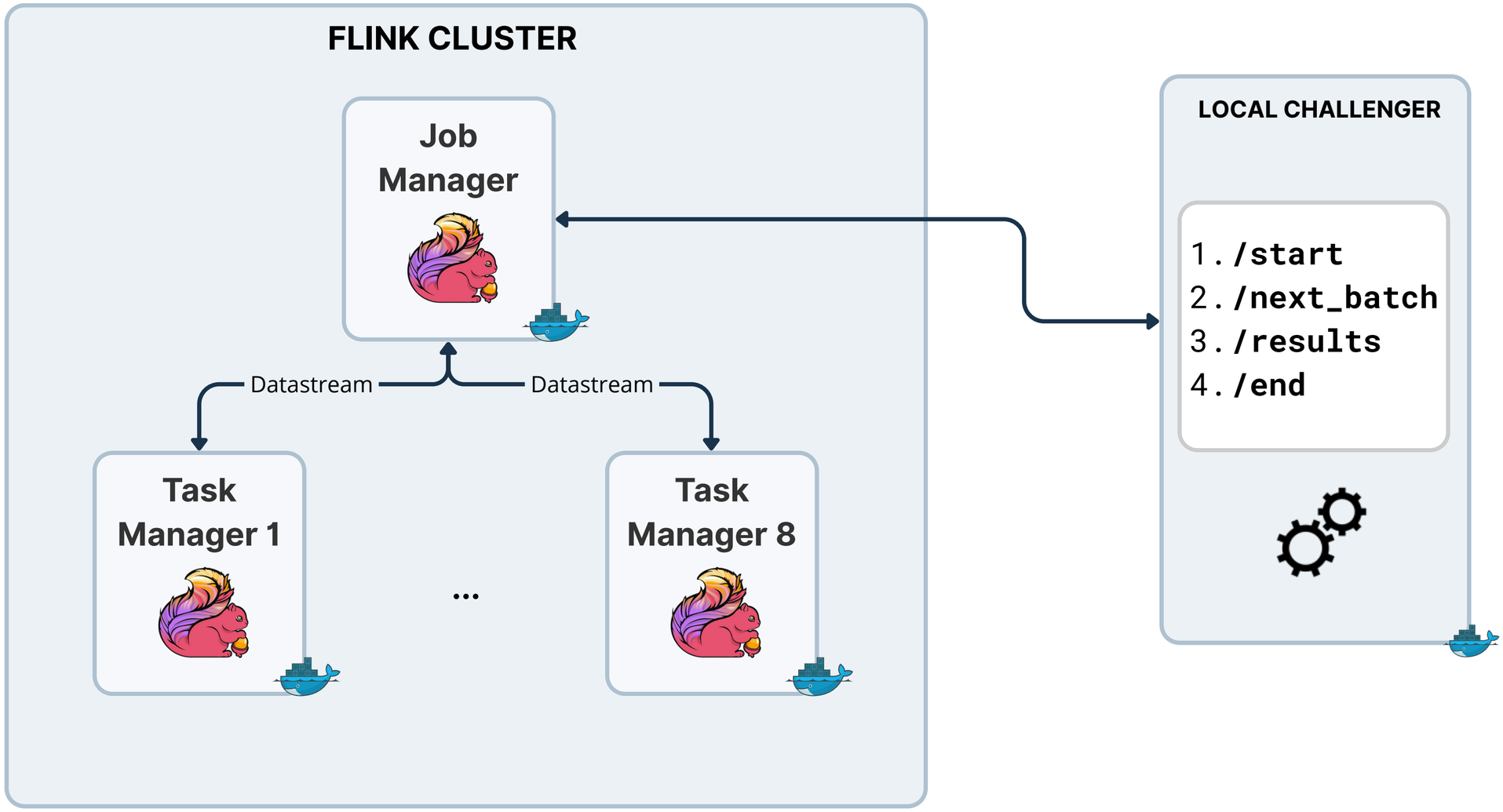
Flink System Architecture



Flink System Architecture

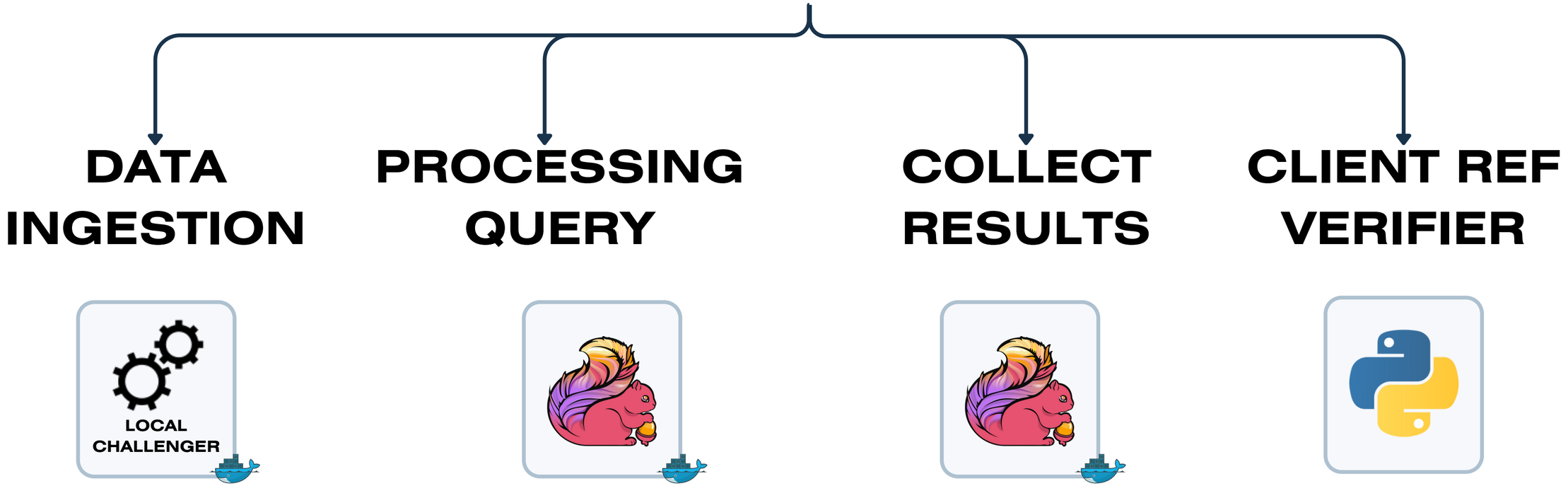


Flink System Architecture

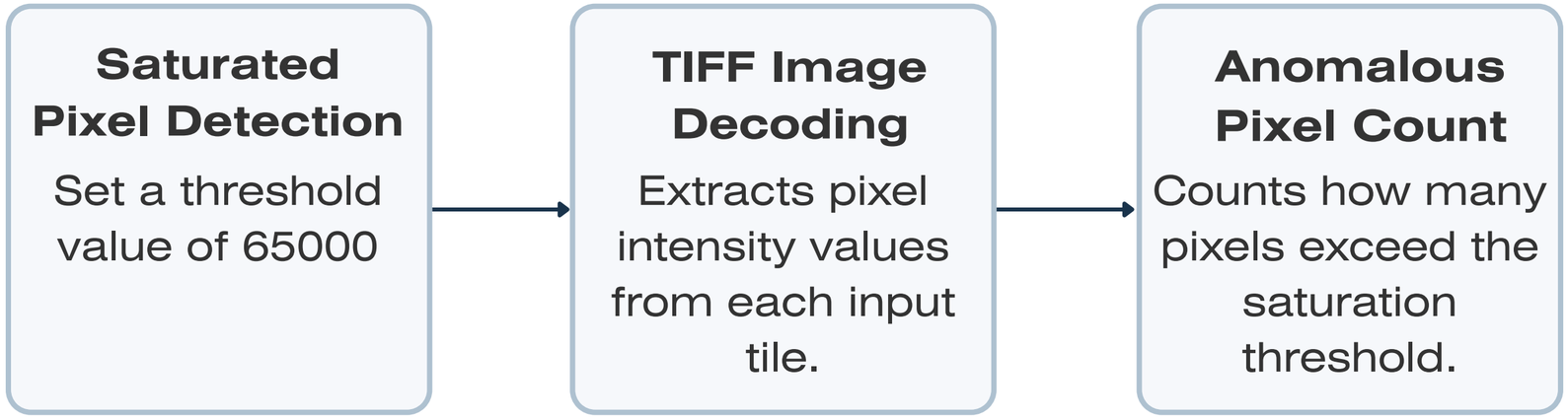


Flink Pipeline

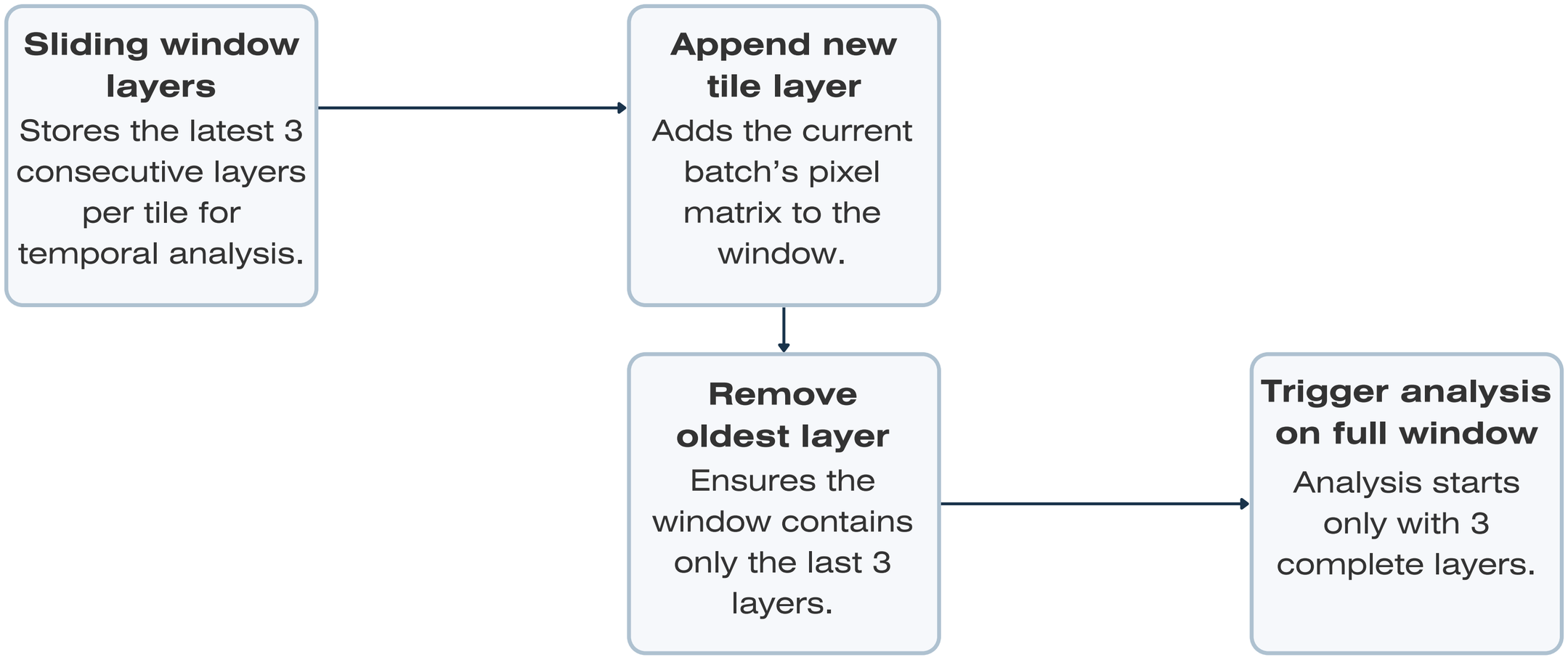
The data processing pipeline consists of the following steps:



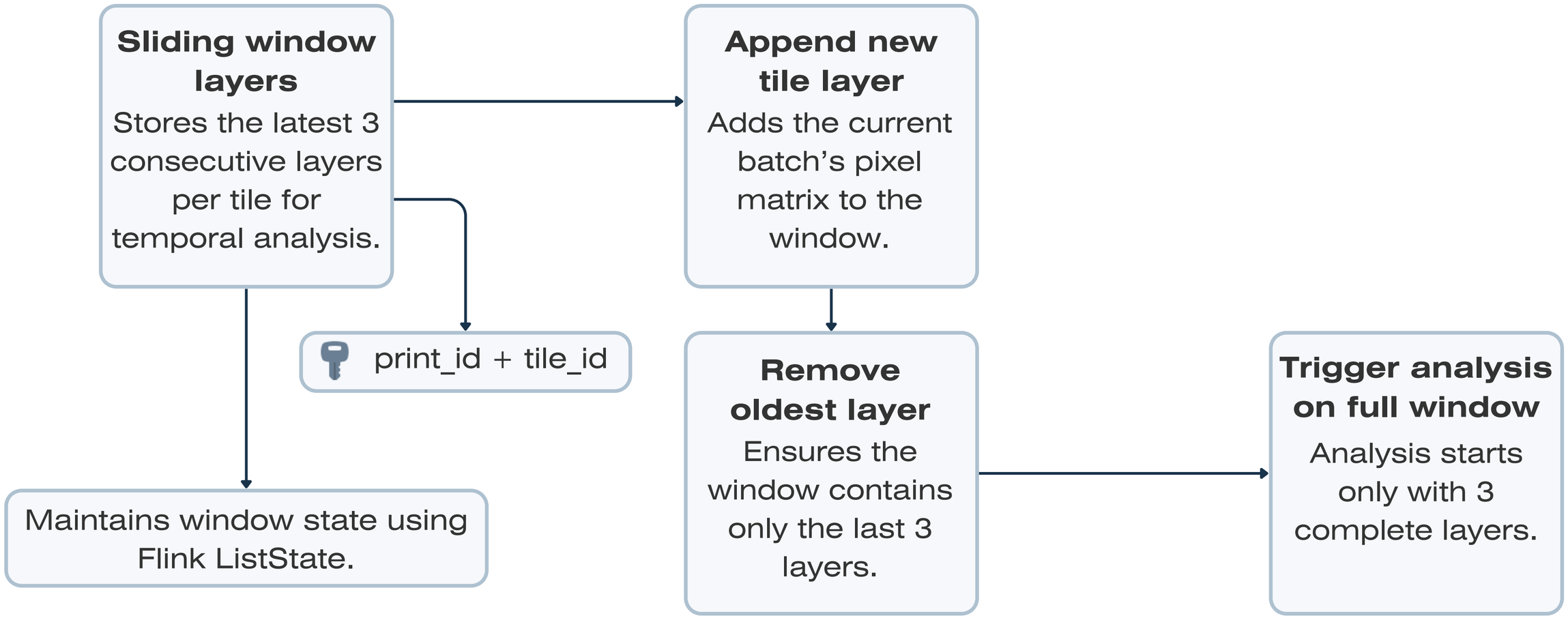
Query 1 - Flink



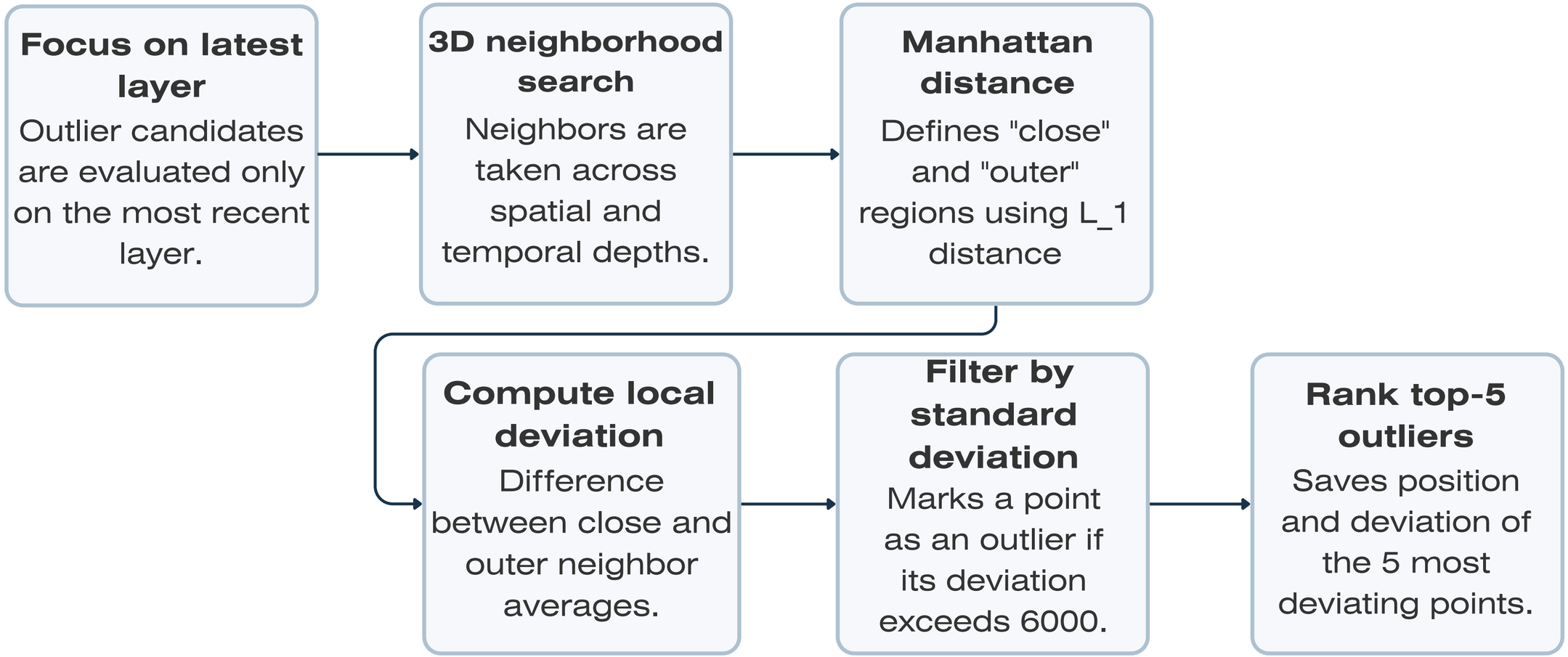
Query 2 - Flink (Windowing)



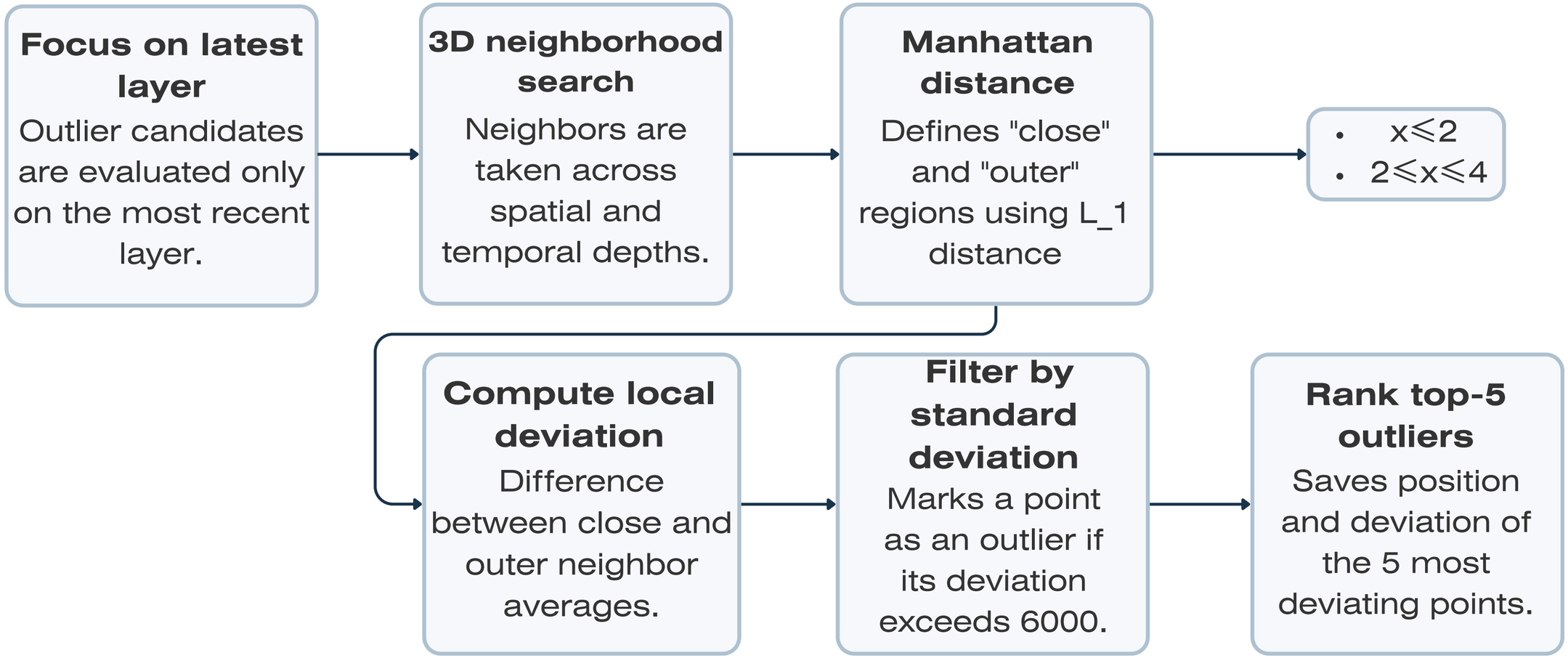
Query 2 - Flink (Windowing)



Query 2 - Flink (Outlier analysis)



Query 2 - Flink (Outlier analysis)



Query 2 Optimized - Flink

Exploit spatial symmetry

Reduce the total number of iterations performed in the calculation

$$C_p = \{q \mid \delta(p,q) \leq 2\}$$
$$O_p = \{q \mid \delta(p,q) \leq 4\}$$
$$d_p = \left| \frac{1}{||C_p||} \sum_{q \in C_p} T_q - \frac{1}{||O_p||} \sum_{q \in O_p} T_q \right|$$

Build Convolutional Kernel

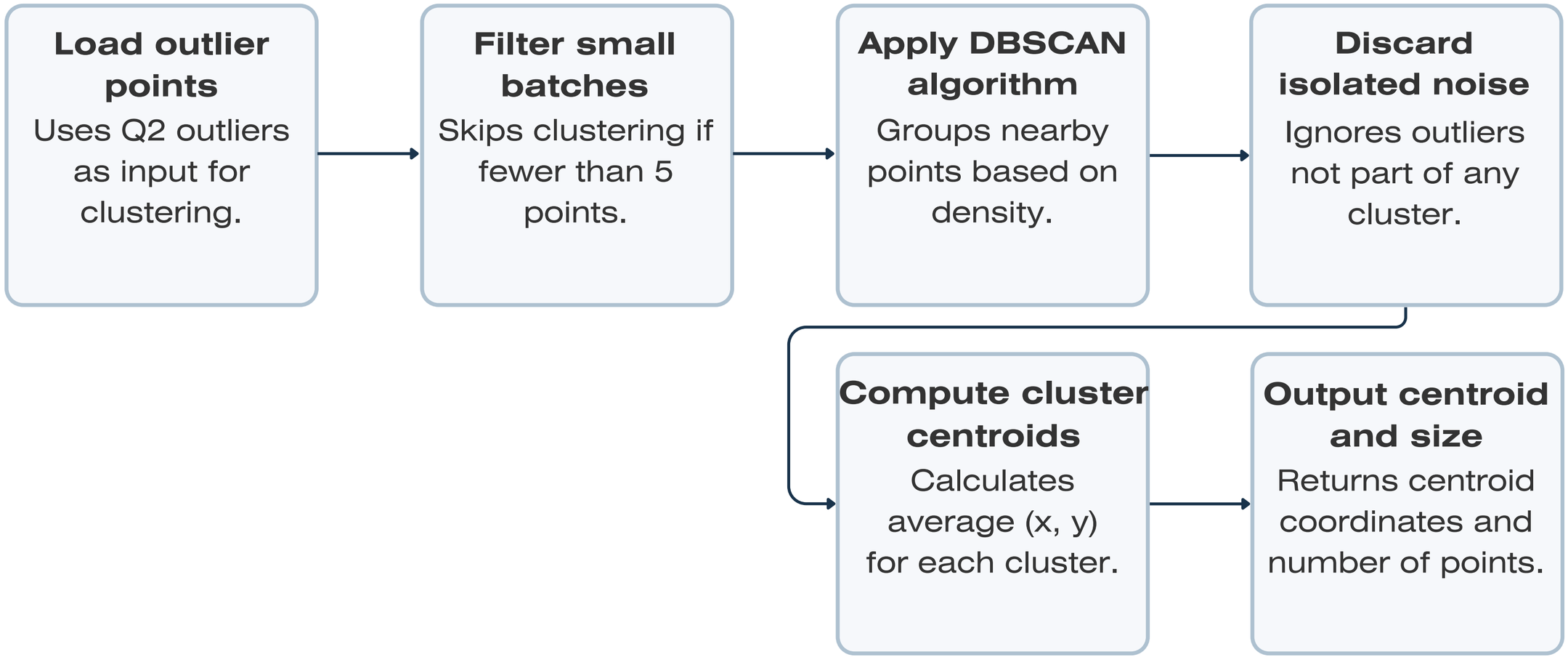
The 3D convolution is applied to the window tensor W

$$K(i,j,k) = \begin{cases} \frac{1}{||C_p||}, & \text{if } \delta((0,0,0),(i,j,k)) \leq 2 \\ \frac{-1}{||O_p||}, & \text{if } 2 < \delta((0,0,0),(i,j,k)) \leq 4 \\ 0, & \text{altrimenti} \end{cases}$$

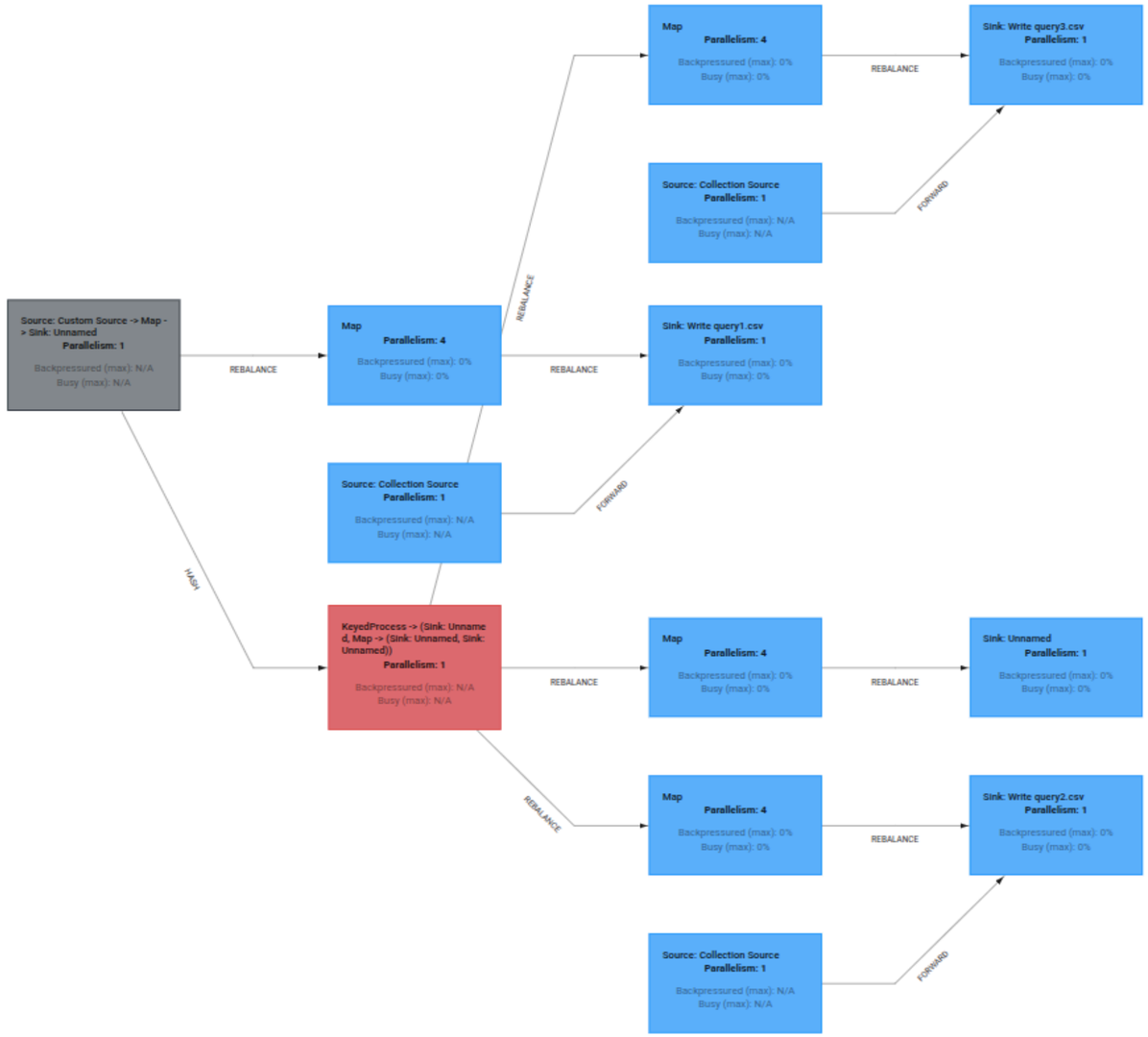
Calculate Deviation Map

Each value represents the local deviation δp at pixel (x, y)

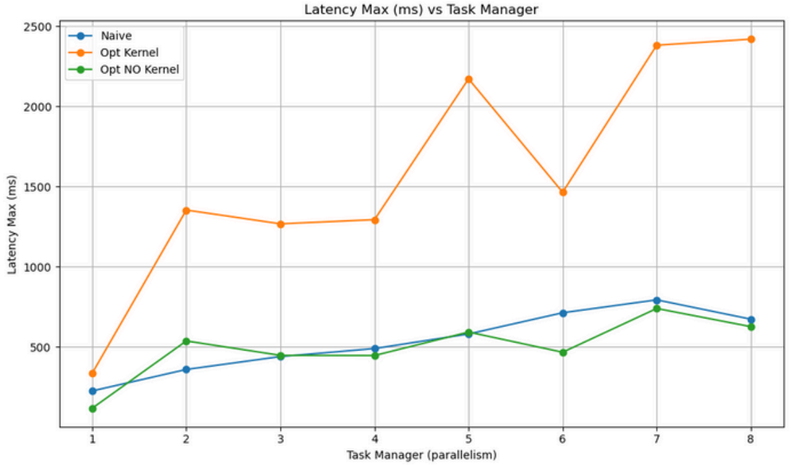
Query 3 - Flink

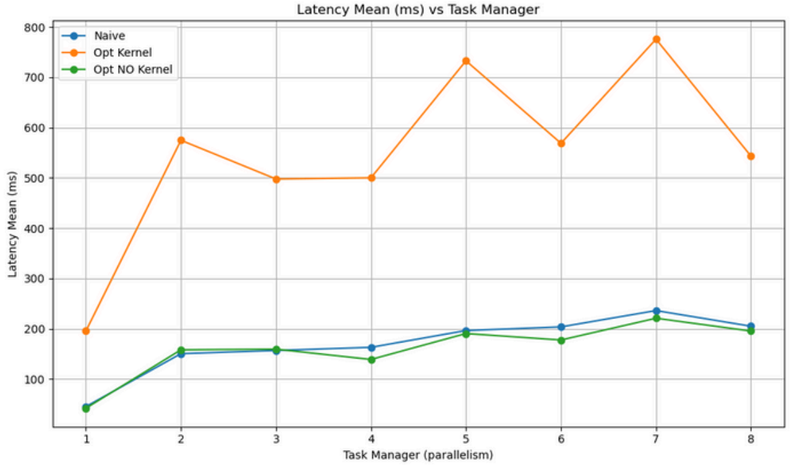


Flink - DAG


Source: Custom Source -> Map -> Sink: Unnamed
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
REBALANCE
Map
Parallelism: 4
Backpressured (max): 0%
Busy (max): 0%
REBALANCE
Map
Parallelism: 4
Backpressured (max): 0%
Busy (max): 0%
REBALANCE
Sink: Write query3.csv
Parallelism: 1
Backpressured (max): 0%
Busy (max): 0%
Source: Collection Source
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
Sink: Write query1.csv
Parallelism: 1
Backpressured (max): 0%
Busy (max): 0%
Source: Collection Source
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD
HASH
KeyedProcess -> (Sink: Unnamed, Map -> (Sink: Unnamed, Sink: Unnamed))
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
REBALANCE
Map
Parallelism: 4
Backpressured (max): 0%
Busy (max): 0%
REBALANCE
Sink: Unnamed
Parallelism: 1
Backpressured (max): 0%
Busy (max): 0%
REBALANCE
Map
Parallelism: 4
Backpressured (max): 0%
Busy (max): 0%
REBALANCE
Sink: Write query2.csv
Parallelism: 1
Backpressured (max): 0%
Busy (max): 0%
Source: Collection Source
Parallelism: 1
Backpressured (max): N/A
Busy (max): N/A
FORWARD

Results - Flink Version



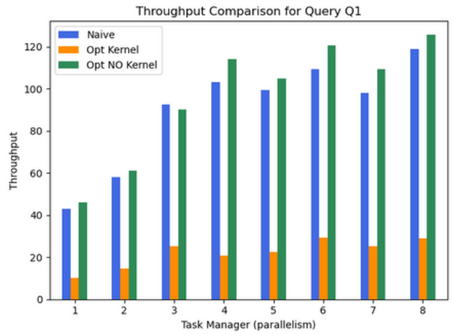


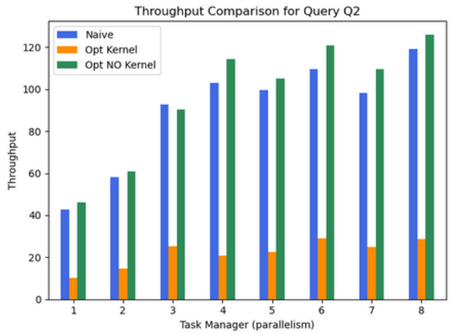
Results - Flink Version

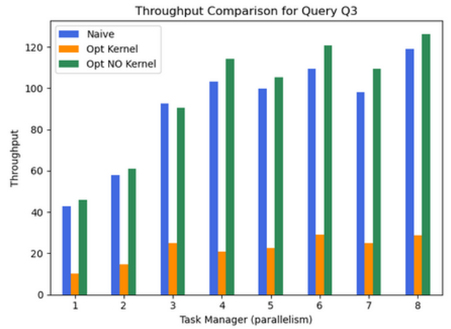
	parallelism	bench_id	throughput	latency_mean	latency_max
1	1	01JZFCS4SBDP2TKRR47KN65S7Y	42.80	45ms429µs638ns	226ms26µs220ns
2	2	01JZFCVQT4BWXMNRZTN530P1XZ	57.92	150ms608µs245ns	359ms985µs929ns
3	3	01JZFCXNESZPABDJ22EHNMRWCQ	92.33	156ms928µs791ns	440ms297µs384ns
4	4	01JZFCYXEVNMKKQ741WAB1R3VA	102.62	163ms190µs819ns	490ms156µs78ns
5	5	01JZFD01VNMFA2YKASDVM6TNJS	99.14	196ms580µs312ns	581ms91µs858ns
6	6	01JZFD15ZDC99MQ7A0N3PBZG8Z	109.31	203ms833µs977ns	713ms358µs40ns
7	7	01JZFD28978RGHDQZ0YHWP2B38	97.81	236ms257µs818ns	793ms698µs202ns
8	8	01JZFD3EVTAKB0FF1CJYCZDMD4	118.48	205ms153µs826ns	673ms430µs230ns
9	1	01JZMS1NPQ9D87D4TKY67ZV84X	10.11	195ms810µs931ns	336ms526µs330ns
10	2	01JZMSCK3F70WW0WDNW48VAEN2	14.55	574ms802µs902ns	1s353ms600µs274ns
11	3	01JZMSM6H6M1XQMSZKPKV6BR8C	25.07	497ms869µs679ns	1s267ms758µs257ns
12	4	01JZMSRM9BY0REQE45Q787FMYX	20.71	500ms178µs369ns	1s293ms715µs701ns
13	5	01JZMSXZC3FBCPSJ62F63PF54T	22.59	732ms853µs132ns	2s171ms416µs910ns
14	6	01JZMT2WSGCPHPBC368A0ZJJ6N	29.17	568ms810µs971ns	1s464ms497µs679ns
15	7	01JZMT6PSKPZRM0V0ZB9G5GZJD	25.14	775ms841µs351ns	2s381ms154µs852ns
16	8	01JZMTB4JMQW3NC7Z9NPC4ZESX	29.01	543ms312µs256ns	2s418ms807µs851ns
17	1	01JZFD4ED571JZNNR6PN1G1EF2	45.96	42ms171µs852ns	118ms387µs586ns
18	2	01JZFD6VSTJ2TBRYQJZ9Z84SHP	60.96	158ms307µs79ns	537ms606µs234ns
19	3	01JZFD8QGMXTKKP1ZF7E3H1WYQ	90.22	159ms555µs576ns	447ms823µs150ns
20	4	01JZFD9ZJ4H3VDT8DEM3Q4TKZ4	113.68	139ms63µs312ns	446ms736µs44ns
21	5	01JZFDAZYAXMZ47PG8K2HWW0V8	104.75	190ms587µs520ns	593ms58µs31ns
22	6	01JZFDC26DFTCC3NGQGTMRGQCW	121.01	177ms768µs746ns	467ms259µs783ns
23	7	01JZFDD0K5RRY86ZH10RJY5EA2	109.12	221ms23µs61ns	740ms50µs76ns
24	8	01JZFDE2YBVMQ3SWBB1VDHJ0Z9	125.76	195ms537µs562ns	626ms853µs155ns

Results Query - Flink Version

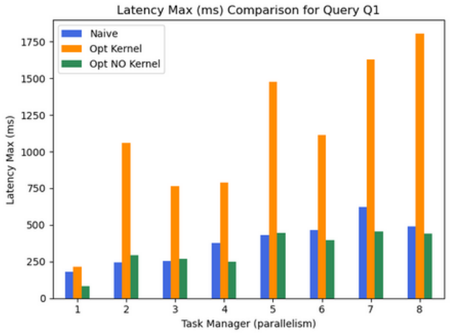
Throughput

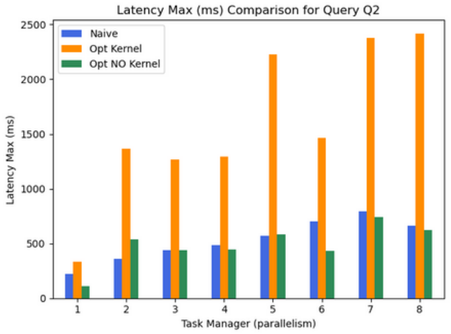


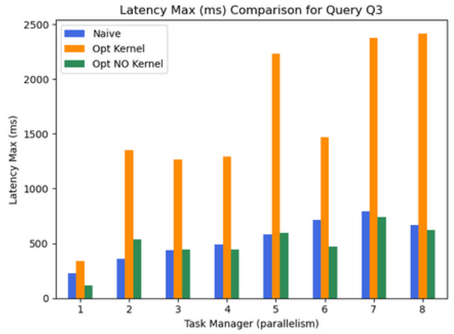




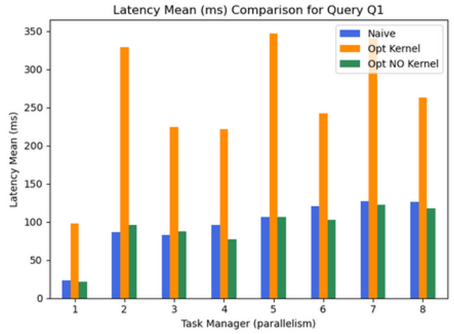
Max Latency

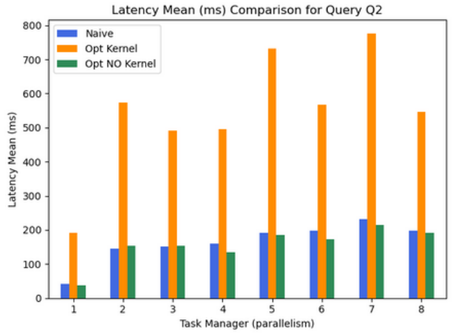


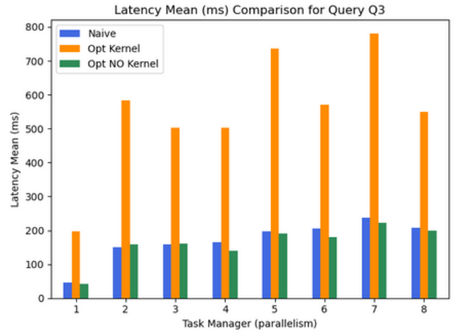




Avg Latency

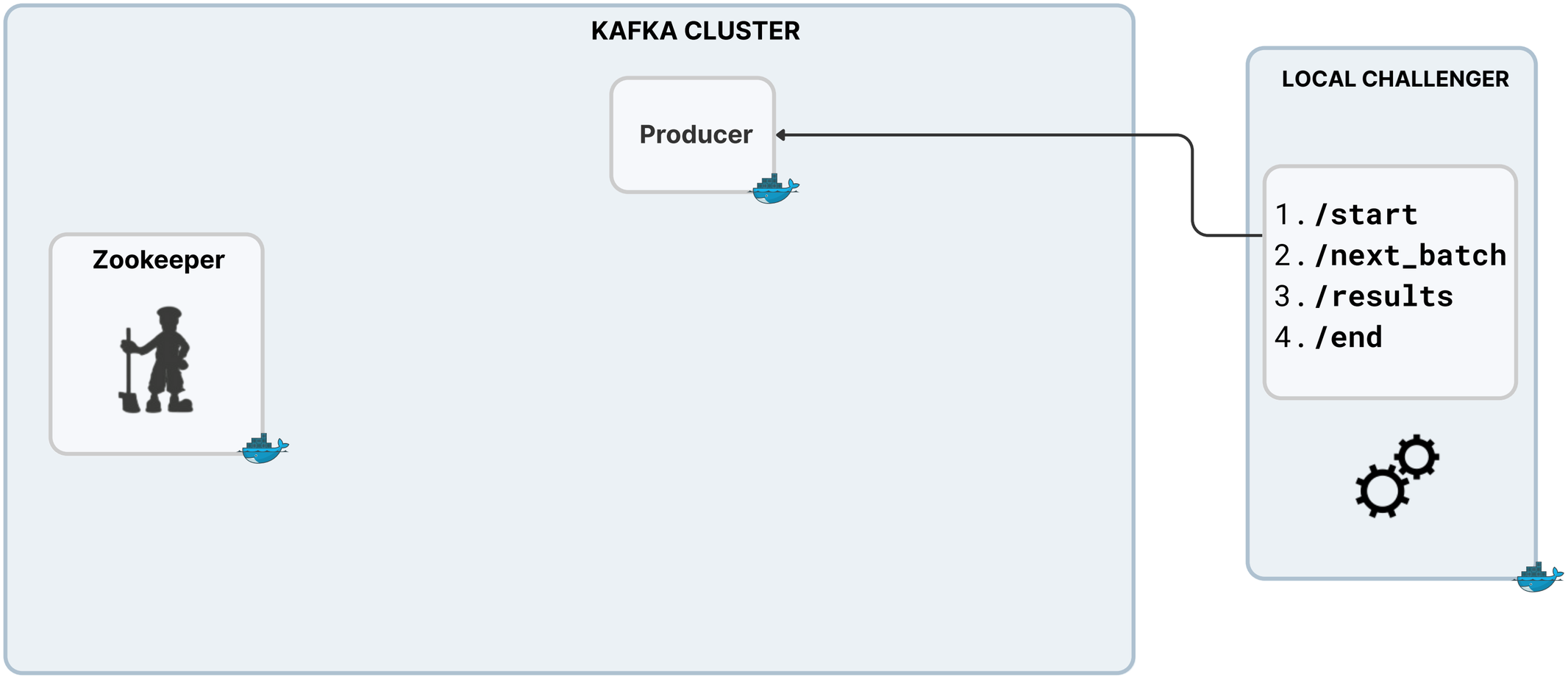




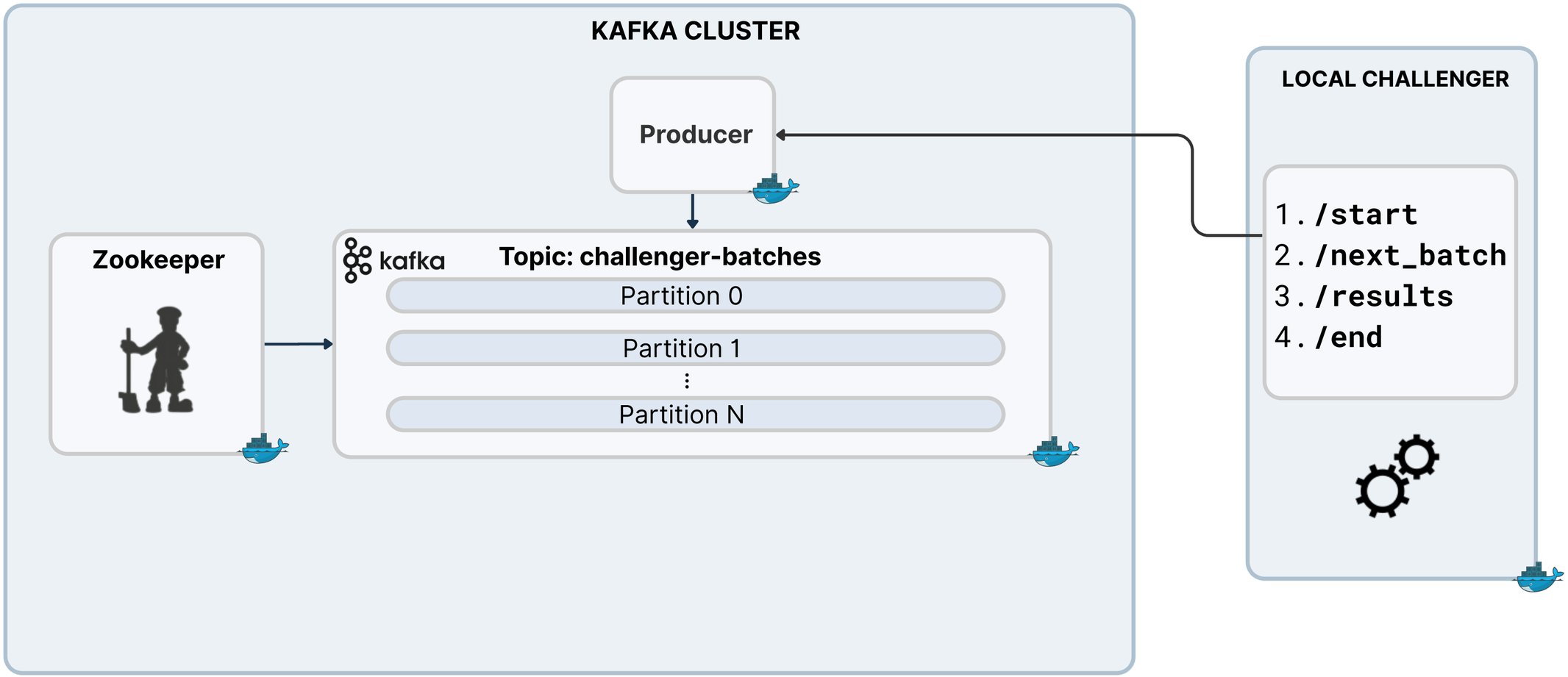


Kafka System Architecture

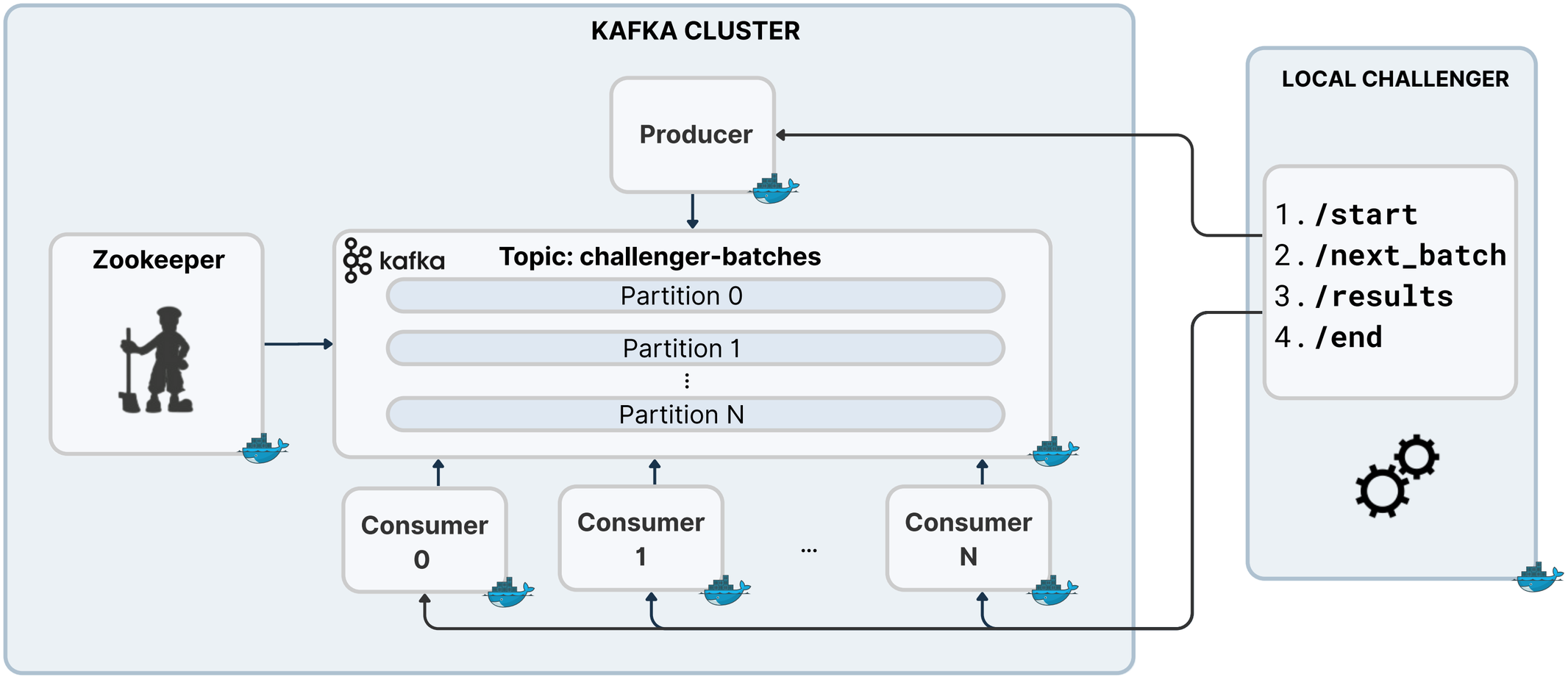
Kafka System Architecture



Kafka System Architecture

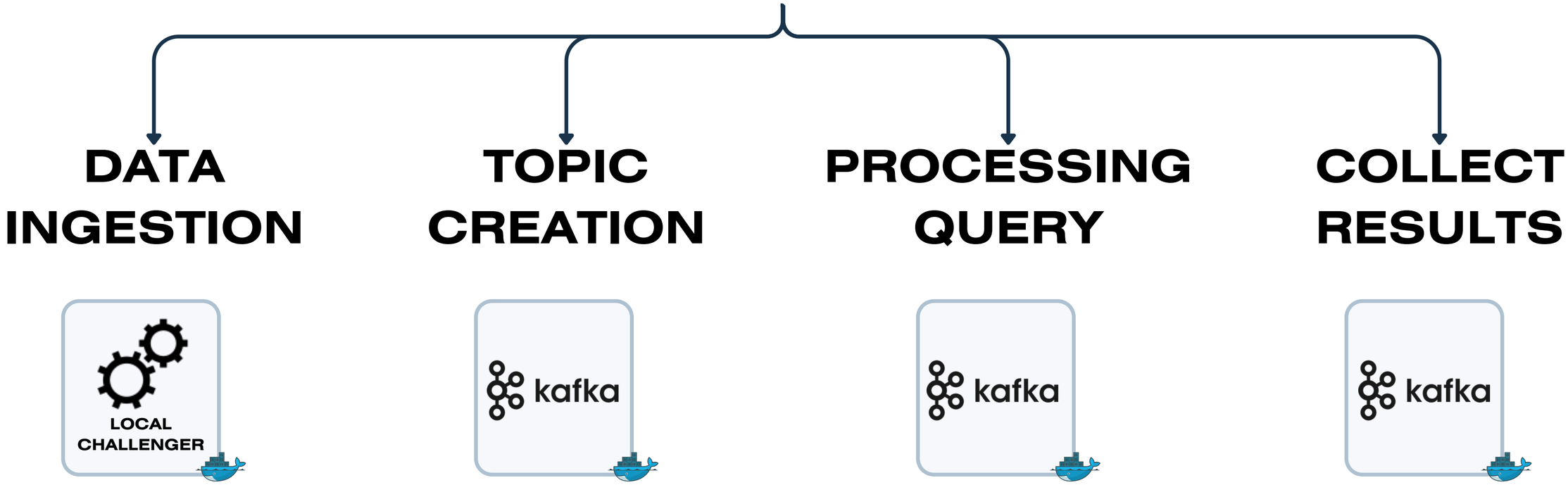


Kafka System Architecture

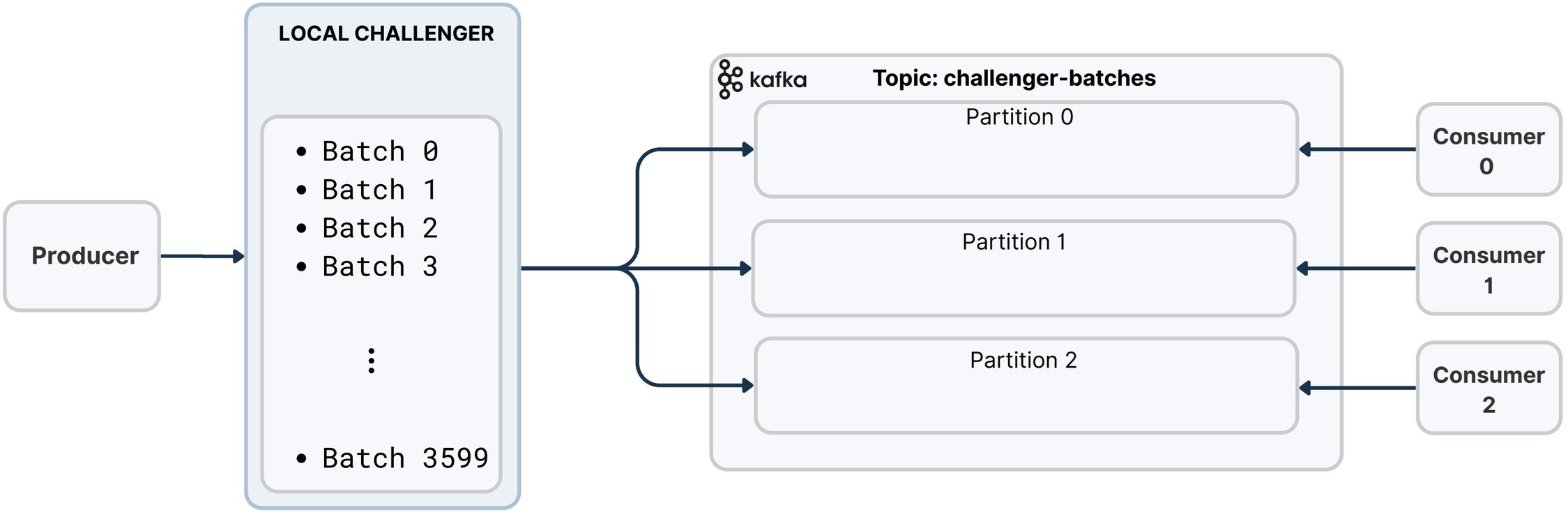


Kafka Pipeline

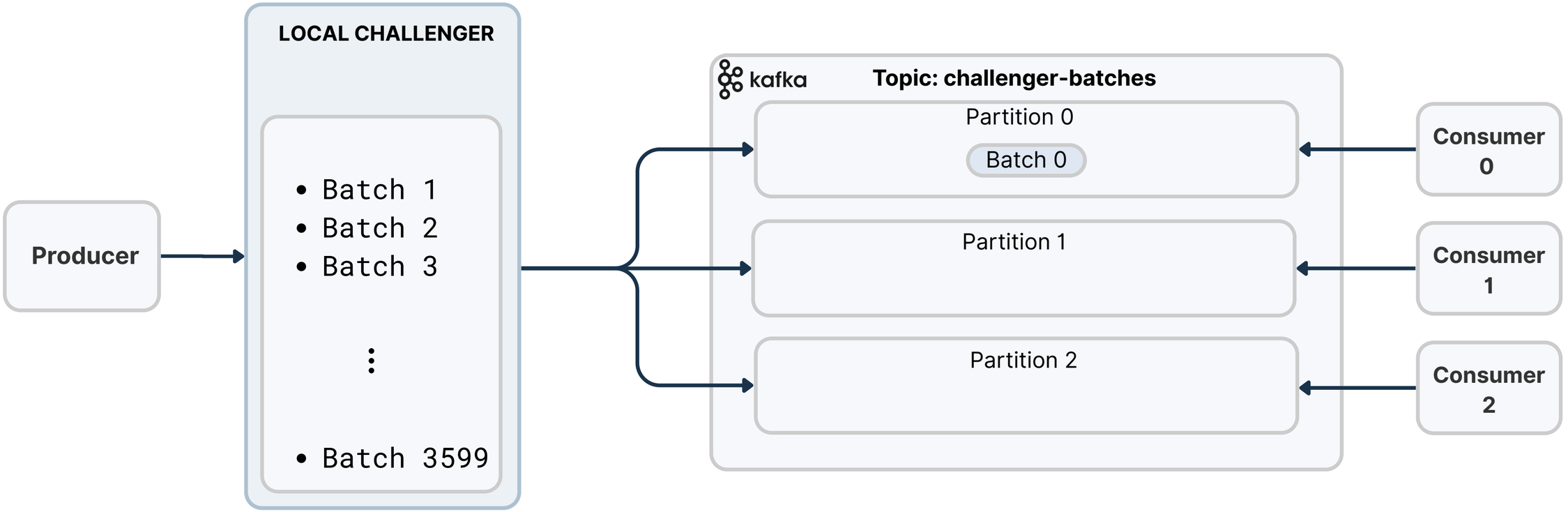
The data processing pipeline consists of the following steps:



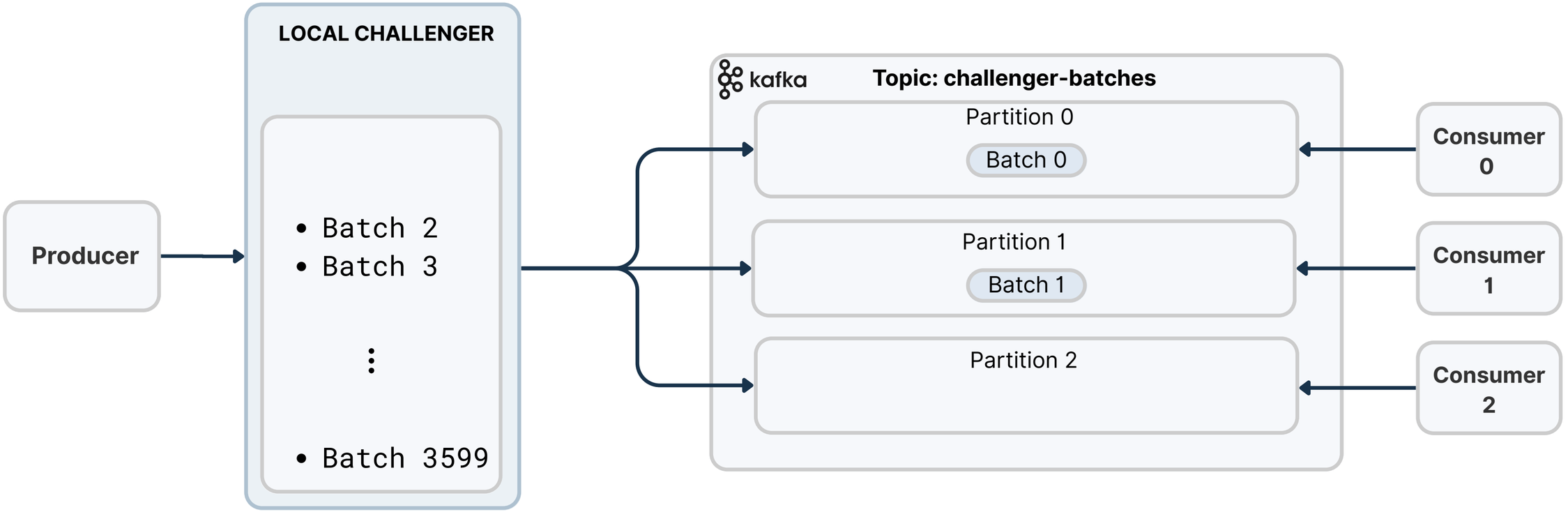
Kafka Producer



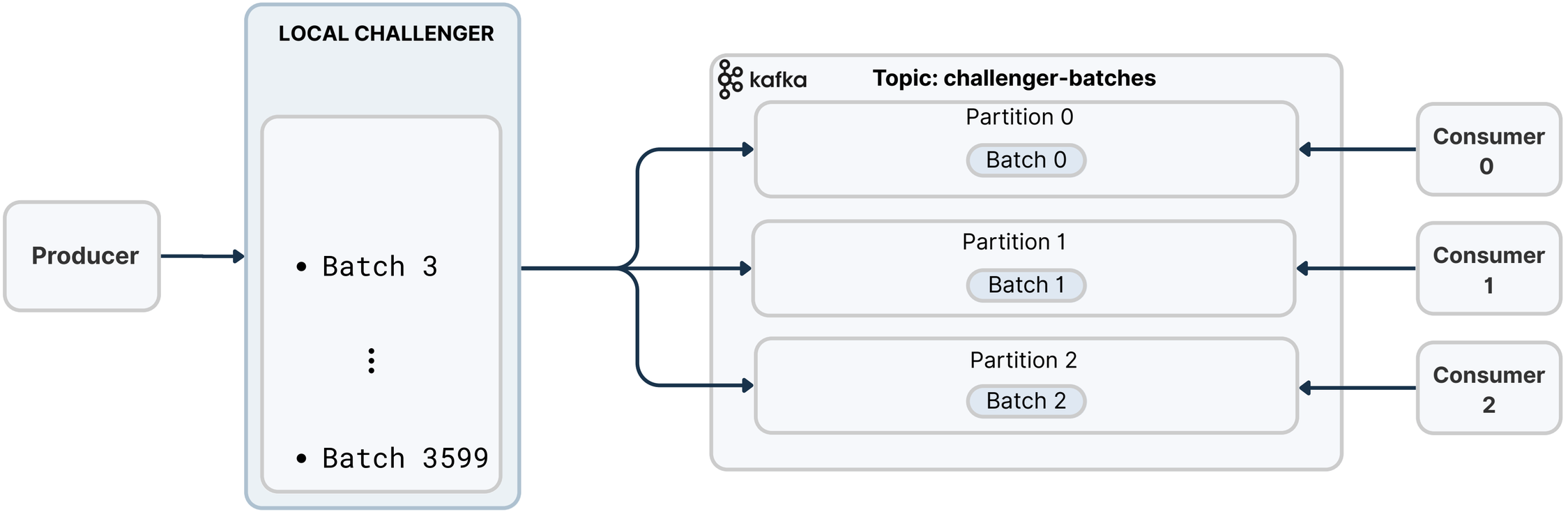
Kafka Producer



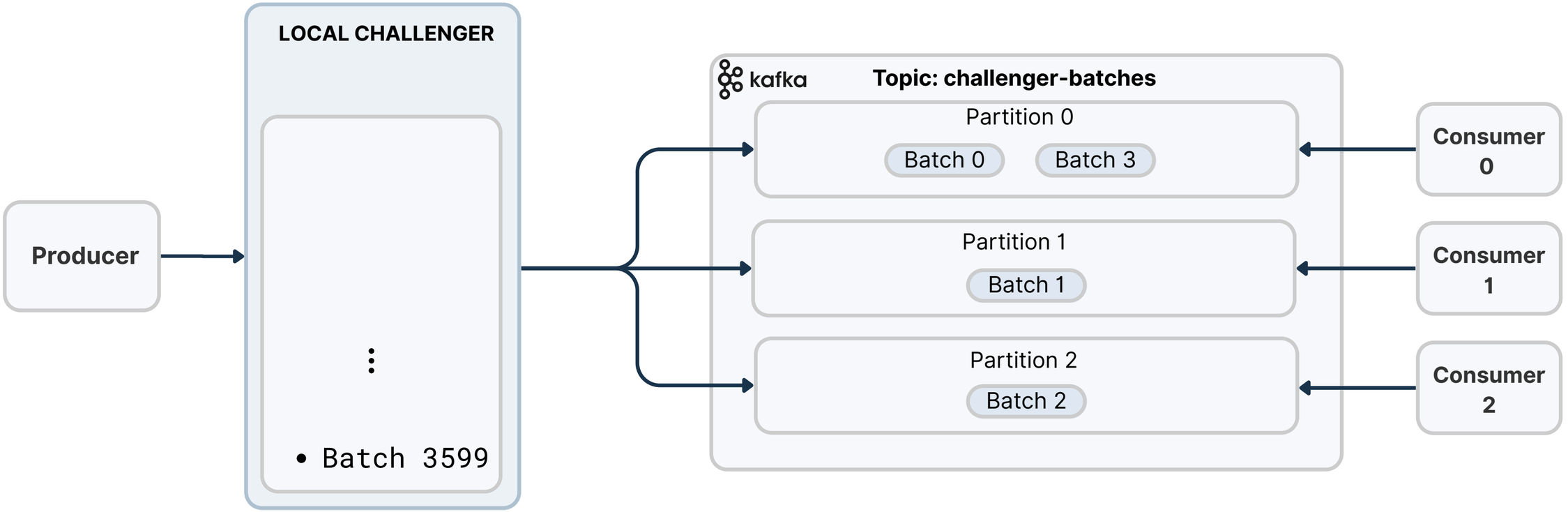
Kafka Producer



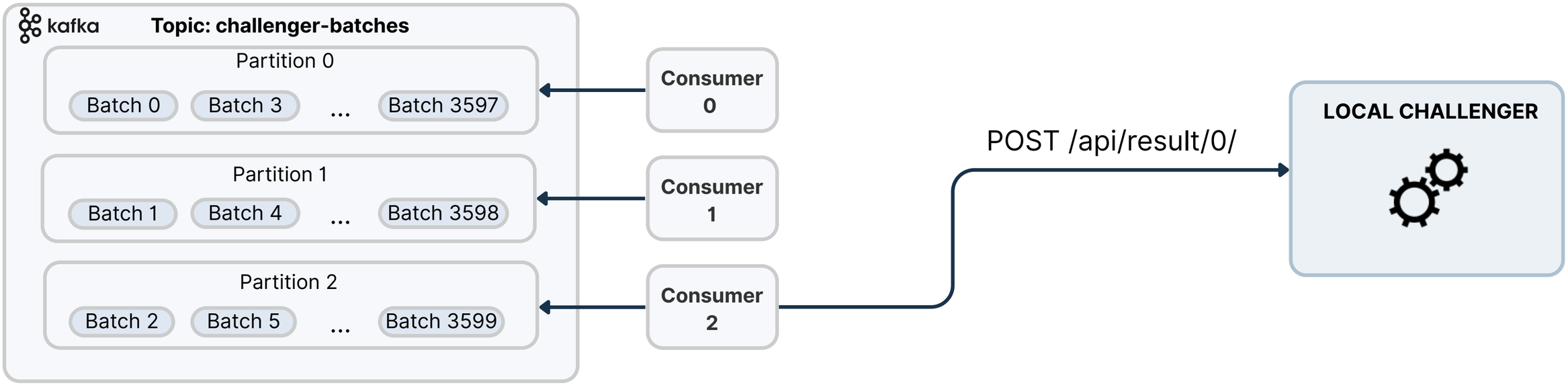
Kafka Producer



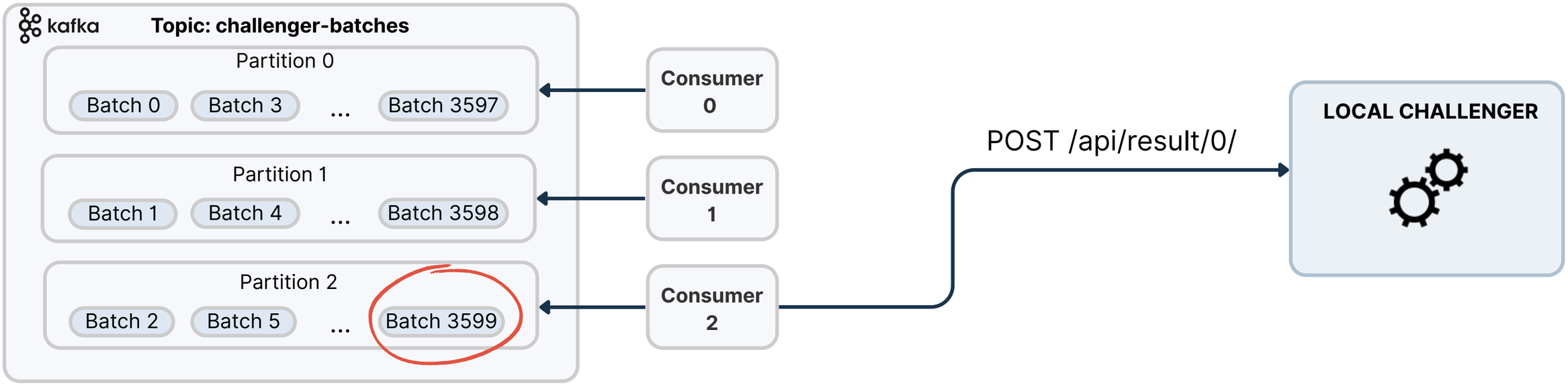
Kafka Producer



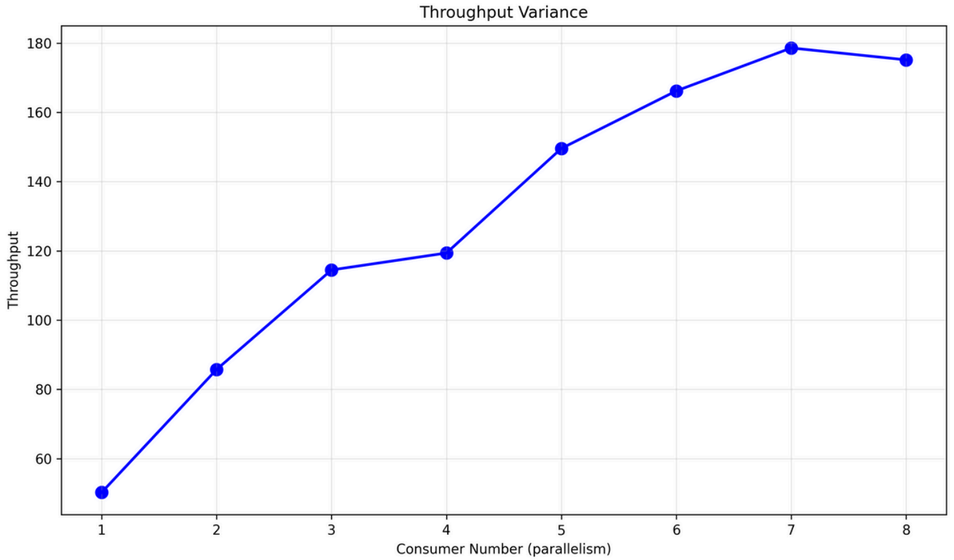
Kafka Consumer

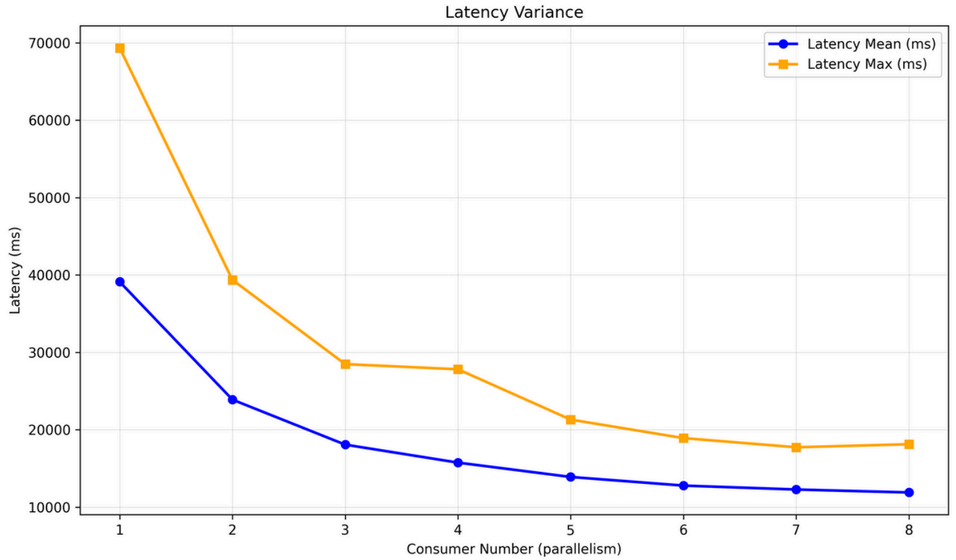


Kafka Consumer



Results - Kafka Version



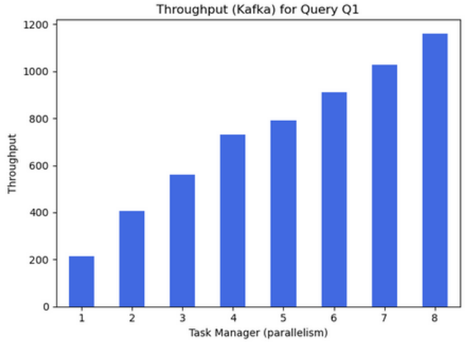


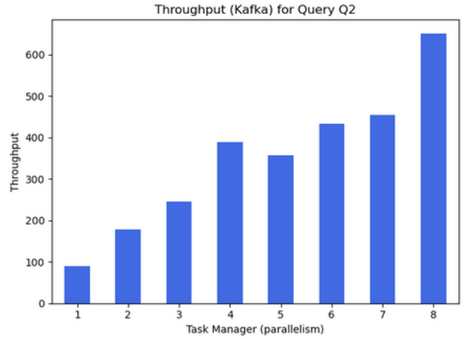
Results - Kafka Version

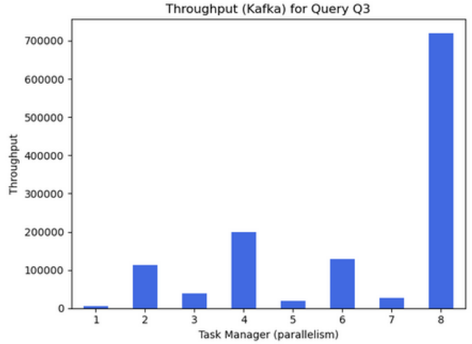
	parallelism	bench_id	throughput	latency_mean	latency_max
1	1	01JZMFGQ81SDJ8G2HEMABQ4T6N	50.22098903979123	39s107ms178µs730ns	1m9s320ms873µs598ns
2	2	01JZMFN465CZ738SRTNVA0ZX06	85.74736328028007	23s879ms521µs806ns	39s353ms927µs902ns
3	3	01JZMFS2Q4HEMZTANZTACPDE91	114.48882427842976	18s67ms809µs109ns	28s472ms43µs119ns
4	4	01JZMFVF1RKGCD4398GS0J6GN7	119.3848778296372	15s743ms100µs850ns	27s804ms834µs136ns
5	5	01JZMFXW1JKZNNT6MKXYRYNNMJ	149.59246634095877	13s889ms424µs369ns	21s303ms15µs344ns
6	6	01JZMG4MMMFJFSKHVFHVBT6JFT	166.21358244595234	12s775ms733µs853ns	18s912ms902µs556ns
7	7	01JZMGFHTE26P8B5P8FB4WV0S7	178.61263541897438	12s268ms572µs75ns	17s726ms551µs491ns
8	8	01JZMGQ7AV1BK50KVZVACMQ6TH	175.17665642473474	11s889ms744µs639ns	18s120ms278µs394ns

Results Query - Kafka Version

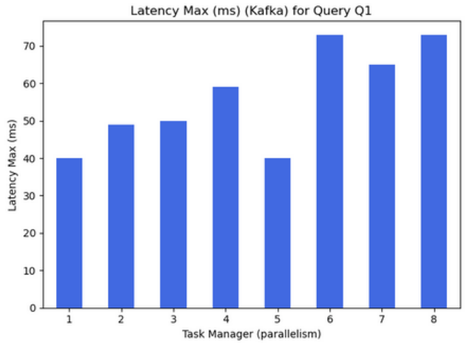
Throughput

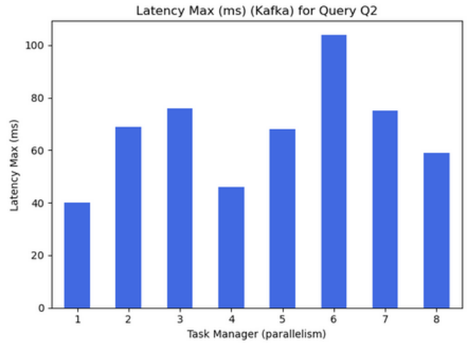


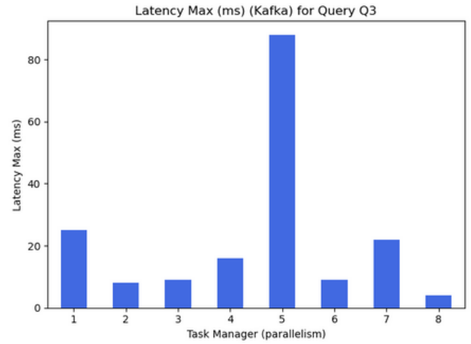




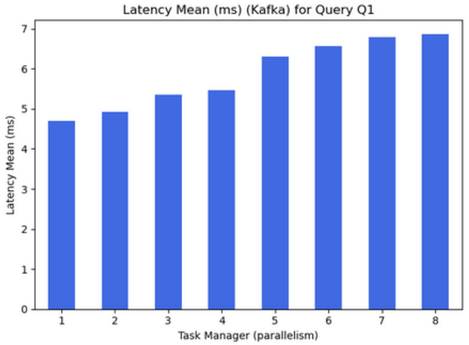
Max Latency

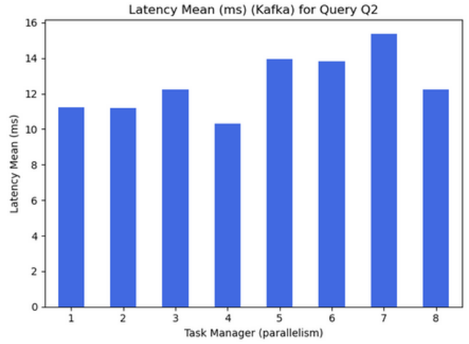


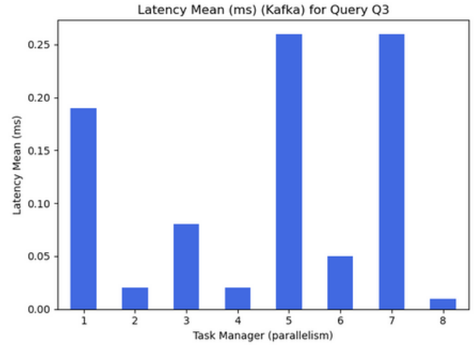




Avg Latency







So, who has the best perfomance?

kafka

So, who has the best perfomance?

?

kafka

Conclusions

It should be noted that Flink uses the proprietary Backpressure mechanism, which ensures that the performance of individual queries is the same.

With Kafka, we don’t have this mechanism natively and this determines the difference in performance of individual queries.

The difference in performance between the two versions is justified by the intrinsic differences of the two systems.

It should be noted that the collected performance was obtained on a machine with an Intel i9-13980HX and 16 GB RAM in a fully containerised environment, without the use of GPU acceleration, where only the implementation code was executed.

Thanks for the Attention!