TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

SABD Project #2

Real-Time Defect Detection in L-PBF Manufacturing Using Stream Processing with Apache Flink

Corso di Sistemi e Architetture per Big Data
Laurea Magistrale in Ingegneria Informatica

A.A. 2024/25
Matteo Basili, Adriano Trani

Outline

- Goals
- Introduction & Background
- Overview
- Data Acquisition and (Kafka) Ingestion
- Implementation of Queries (Flink)
- Results
- Performance Analysis and Discussion
- Demo

Goals

- Analyze real-time thermal images from L-PBF additive manufacturing using DEBS 2025 dataset
- Develop a distributed stream processing pipeline to:
  - Retrieve, process and analyze data
  - Answer specific queries
  - Export results
- Measure latency and throughput for each query
- Present and discuss the results and system performance

Introduction & Background

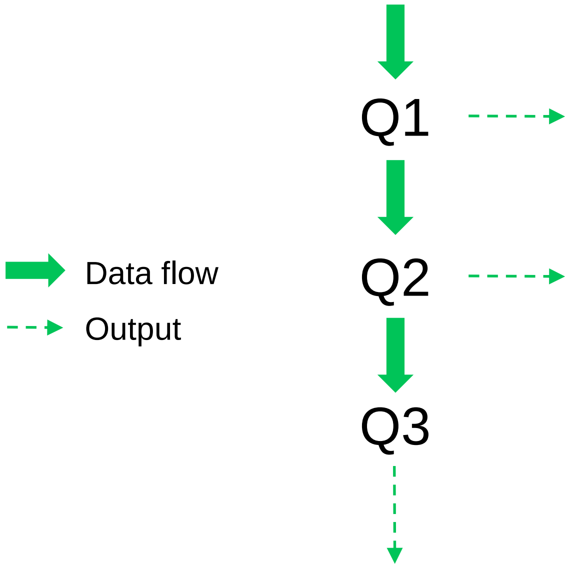
- Laser Powder Bed Fusion (L-PBF) is a 3D printing technology that builds metal parts layer by layer using a high-power laser
- Enables complex geometries and reduces material waste compared to subtractive methods
- Quality issues (e.g., porosity, thermal instabilities) can affect part integrity and are traditionally detected post-production
- Real-time defect detection can reduce waste and improve reliability
- Optical Tomography (OT) captures high-resolution thermal images during the printing process
- These images represent temperature distribution across the powder bed for each layer

Dataset

- Provided by ACM DEBS 2025 Grand Challenge as a continuous stream via a REST-based LOCAL-CHALLENGER
- Organized by layers (z-axis)
- Each layer divided into tiles (sub-images)
- Each tile is a temperature map: pixels represent temperature values (0–65,535)
- Each element of the data stream includes:
    - *seq_id*: Sequence number
    - *print_id*: Printed object ID
    - *tile_id*: Tile identifier
    - *layer*: Z-coordinate
    - *tiff*: Binary temperature image

Queries

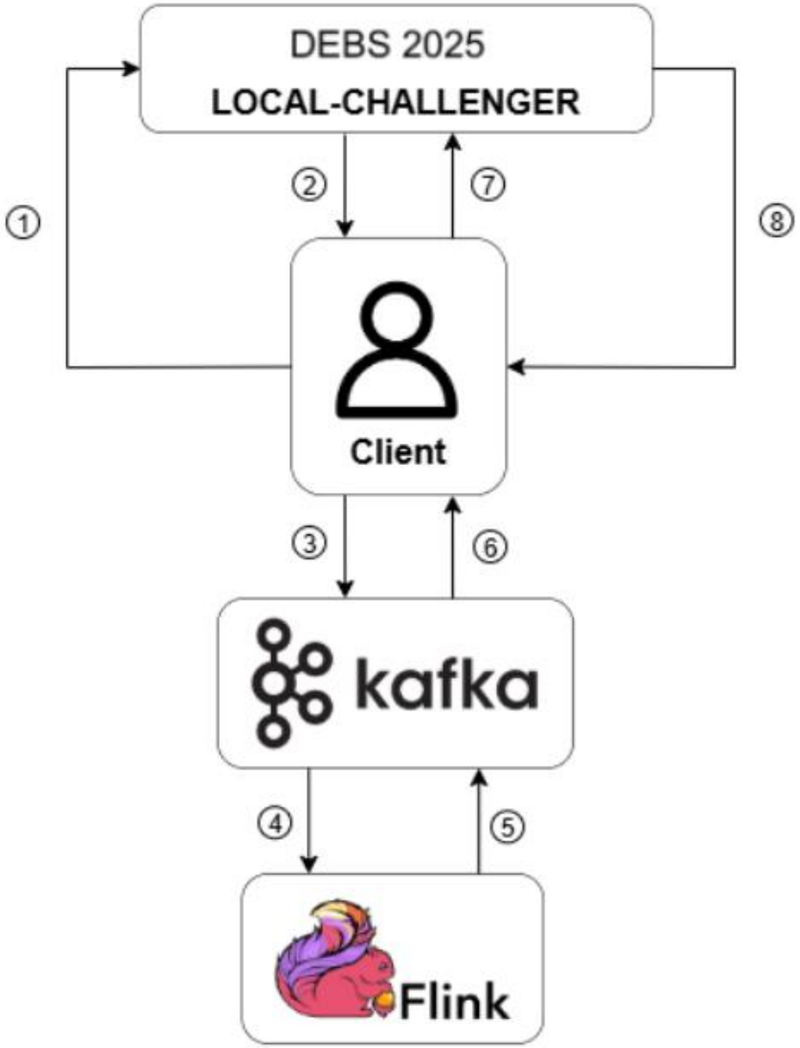
- **Q1** – Saturation Analysis
  - Identify pixels in each tile with:
    - Tp < 5,000 → void areas (excluded from further analysis)
    - Tp > 65,000 → saturated pixels (potential defects)
  - Count the number of saturated points
- **Q2** – Outlier Detection
  - Sliding window over last 3 layers
  - For each pixel in the newest layer:
    - Compute local temperature deviation =
      | avg(inner neighbors) - avg(outer neighbors) |
    - Mark as outlier if deviation > 6,000
  - Output top-5 points with highest local temperature deviation per tile
- **Q3** – Outlier Clustering
  - Apply DBSCAN clustering on Q2 outliers (Euclidean distance) to compute cluster centroids and sizes
  - Output sent to Local-Challenger for performance evaluation



Overview - Technology Stack

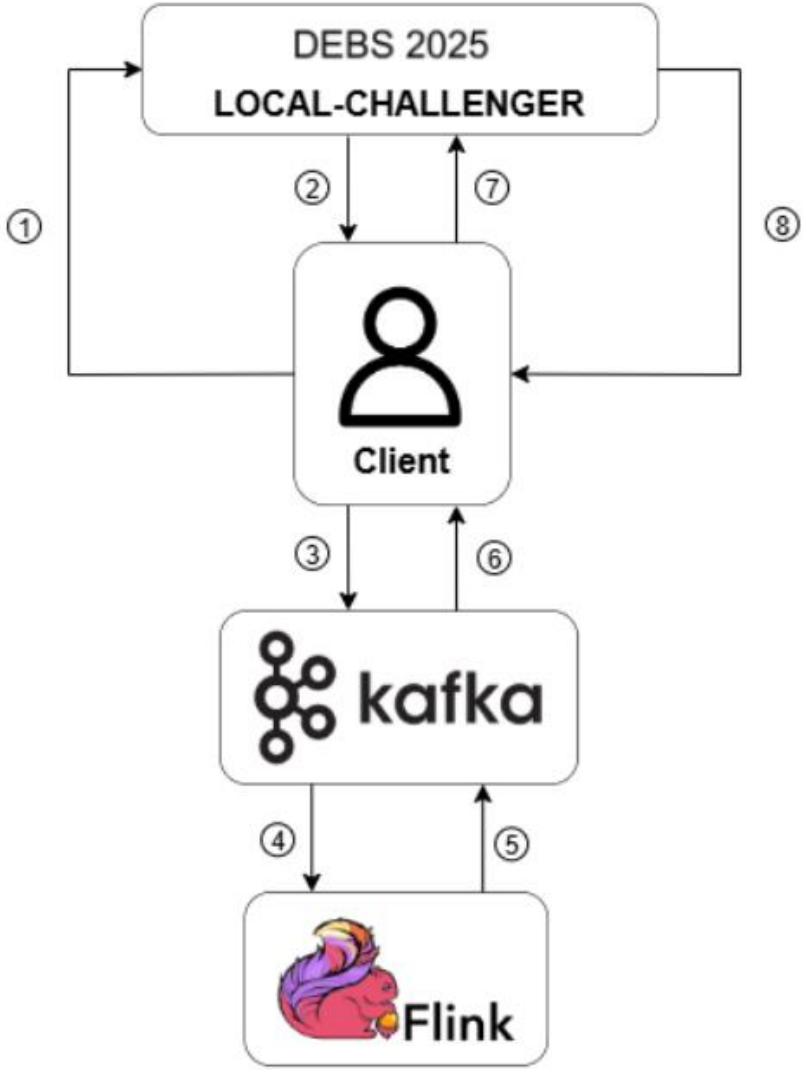
- Docker & Docker Compose: Containerization of the entire system
- Apache Flink: Processing the data
- Apache Kafka: Message brokering between components
- Python: Main programming language

Overview - Architecture



Overview - DataFlow

1) Client establishes a session with the LOCAL-CHALLENGER and starts the benchmark
2) OT image tiles are fetched in real time from the LOCAL-CHALLENGER stream
3) Retrieved tiles are published to a Kafka topic for processing
4) Flink consumes the tiles and executes the multi-stage analysis pipeline
5) Query results are written to dedicated Kafka output topics
6) Client consumes Q3 outputs from Kafka
7) Q3 results are sent to the LOCAL-CHALLENGER to complete the benchmark
8) The LOCAL-CHALLENGER evaluates and returns performance metrics (latency, throughput)



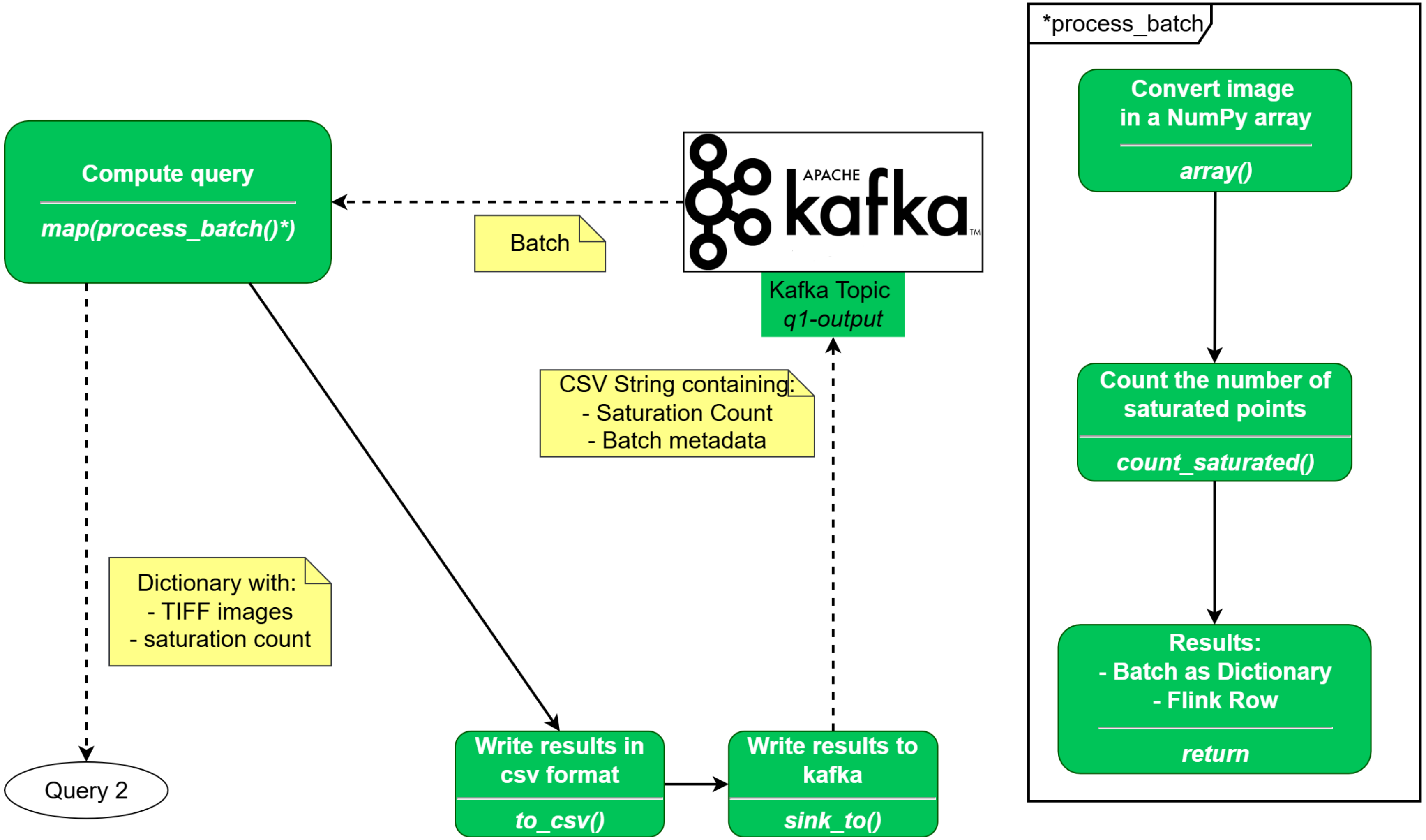
Overview - Docker Containers

- The system is fully containerized with Docker and orchestrated using Docker Compose
  - LOCAL-CHALLENGER: local-challenger
  - Flink: jobmanager, taskmanager1, taskmanager2
  - Kafka: zookeeper, kafka1, kafka2, topic-init
  - Client: l-pbf-client
  - Results exporter: csv-writer

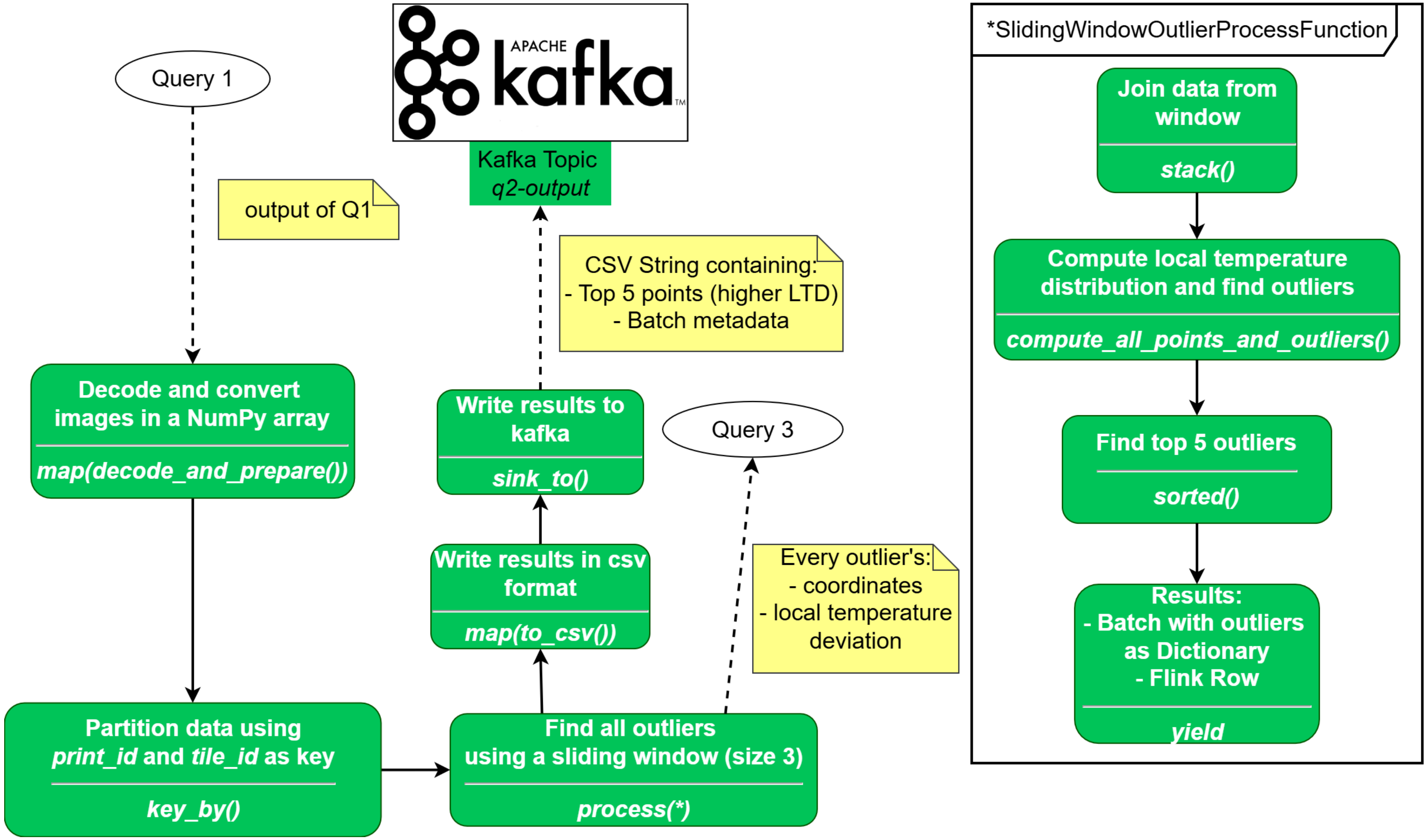
Client – Data Acquisition and Ingestion

- Client continuously requests image batches from the LOCAL-CHALLENGER platform via REST API
- The data is published to the Kafka topic *tiff-batches*, which serves as the input stream for the Flink pipeline
- A dedicated Kafka producer thread handles batch publishing

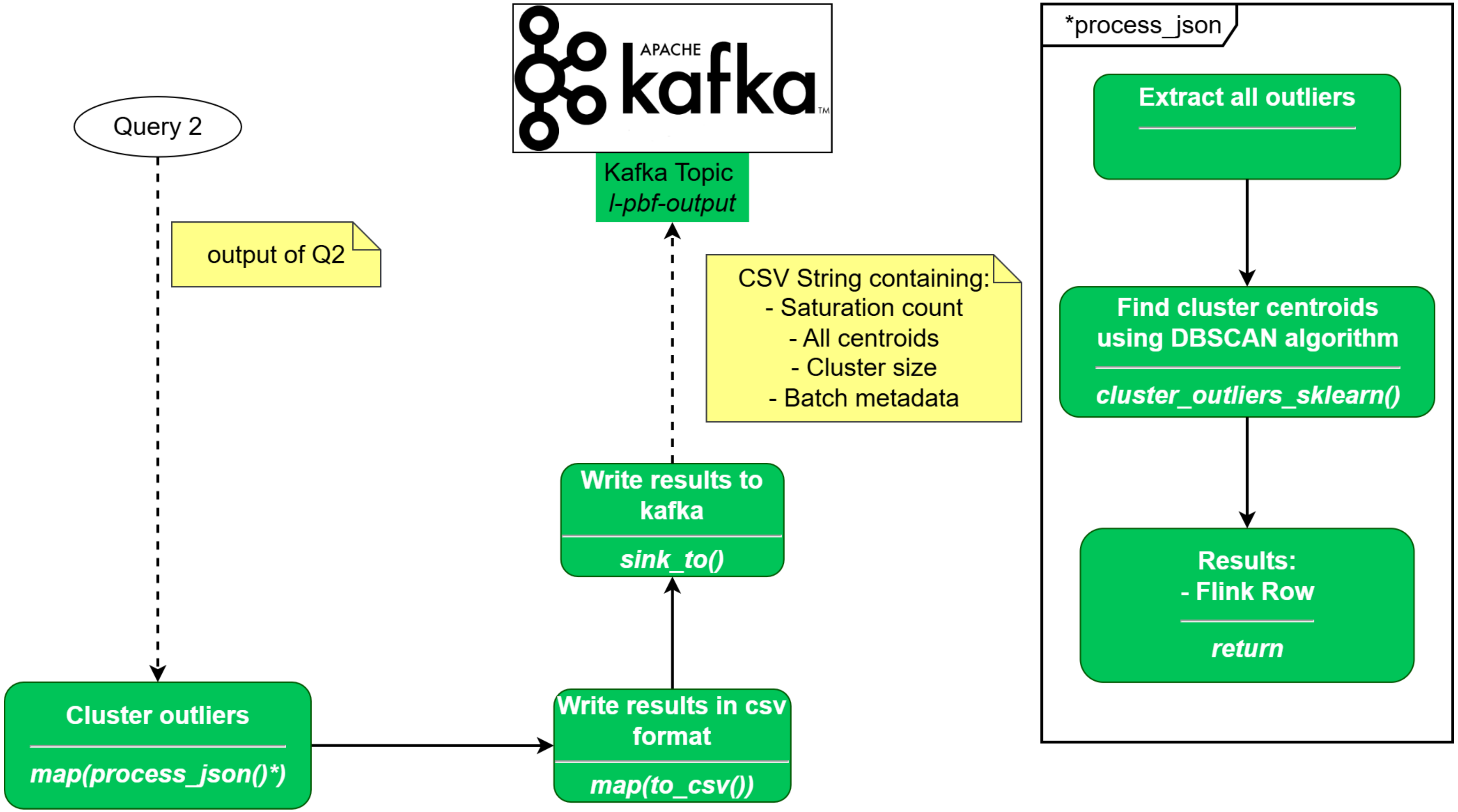
Implementation – Q1



Implementation – Q2



Implementation – Q3



Client – Results collection

- A Kafka consumer runs in a separate thread, continuously listening to the Q3 output topic (*l-pbf-output*)
- Each message is sent via REST to LOCAL-CHALLENGER for validation
- When all batches are received, the client:
  - Finalizes the benchmark
  - Triggers LOCAL-CHALLENGER to return application performance metrics (latency, throughput, etc.)
- ***l-pbf_client.py*** script

Results – Export

- The CSV Writer ingests streaming data from multiple Kafka topics and writes them to corresponding CSV files in real time
- Uses Kafka consumers, spawning one thread per topic for concurrent consumption
- Each message is parsed as a CSV string and appended to a query-specific file
- Output:
  - Q1: *seq_id, print_id, tile_id, saturated*
  - Q2: *seq_id, print_id, tile_id, p_1, dp_1, p_2, dp_2, p_3, dp_3, p_4, dp_4, p_5, dp_5*
  - Q3: *seq_id, print_id, tile_id, saturated, centroids*
- ***kafka_to_csv_stream_writer.py*** script

Results - Example

- Q3 output

seq_id	print_id	tile_id	saturated	centroids
98	SI266220200309225433	2	0	[{"x":381.2463768115942,"y":183.92753623188406,"count":69},{"x":337.22222222222223,"y":176.05555555555554,"count":54},{"x":498.0,"y":239.7,"count":10}]
99	SI266220200309225433	3	7571	[{"x":470.26517571884983,"y":141.38658146964858,"count":313},{"x":353.4415584415584,"y":134.48701298701297,"count":308}]
100	SI266220200309225433	4	0	[]
101	SI266220200309225433	5	0	[]
102	SI266220200309225433	6	0	[{"x":1.0,"y":240.0,"count":12},{"x":273.2083333333333,"y":378.7083333333333,"count":24},{"x":436.2,"y":378.0,"count":5}]
103	SI266220200309225433	7	3084	[{"x":28.02061855670103,"y":152.73711340206185,"count":194},{"x":270.6111111111111,"y":95.0,"count":18},{"x":1.1842105263157894,"y":141.0,"count":38}]
104	SI266220200309225433	8	0	[]
105	SI266220200309225433	9	6	[{"x":156.5321100917431,"y":449.88990825688074,"count":109},{"x":327.59375,"y":447.1145833333333,"count":96},{"x":146.48979591836735,"y":498.14285714285717,"count":49}]
106	SI266220200309225433	10	0	[{"x":143.9512195121951,"y":0.9512195121951219,"count":41},{"x":251.83333333333334,"y":35.833333333333336,"count":6}]
107	SI266220200309225433	11	0	[]
108	SI266220200309225433	12	0	[{"x":238.15384615384616,"y":297.61538461538464,"count":13}]

Performance Analysis – Setup

- Apache Flink (default) configuration
  - Parallelism: set to 1
    → Each operator runs as a single task
- Kafka Topics:
  - Single-partition
    → One Flink subtask per topic → *no parallel reads*
  - Replication factor = 2
    → Basic fault tolerance
- Rationale:
  - Simplified setup for baseline performance evaluation
  - Avoids noise from parallelism and tuning overhead

Performance Analysis – Execution Logic

- Custom log lines emitted by each query with batch ID and latency:

  METRICS|Q1|batch=123|latency_ms=17.46
- Implemented via manual timestamp logging in each query function
- ***analyze_latency.py*** script
  - Log parsing
  - Latency stats
    - Mean, max, p50/p90/p99, batch counts
  - Throughput estimation
    - Batches processed / total run time
  - Excel export
- 3 runs
  - 100, 200 and 300 batches

Performance Analysis – Results

Q1	Mean Latency (ms)	Max Latency (ms)	Latency Perc 50 (ms)	Latency Perc 90 (ms)	Latency Perc 99 (ms)	Throughput (batch/s)
100 batch	58.7562	942.16	22.14	80.521	711.49	0.352939
200 batch	87.36195	2535.01	27.085	83.67	1506.5952	0.233359
300 batch	100.7782	2081.97	31.23	128.08	1606.7608	0.244792

Q2	Mean Latency (ms)	Max Latency (ms)	Latency Perc 50 (ms)	Latency Perc 90 (ms)	Latency Perc 99 (ms)	Throughput (batch/s)
100 batch	3390.3514	10696.52	3212.92	8266.299	9991.9865	0.352274
200 batch	4280.8253	11439.79	3743.695	8620.857	10433.9657	0.232865
300 batch	4711.088267	12825.29	4095.37	8836.136	11893.4476	0.244775

Q3	Mean Latency (ms)	Max Latency (ms)	Latency Perc 50 (ms)	Latency Perc 90 (ms)	Latency Perc 99 (ms)	Throughput (batch/s)
100 batch	9.129	592.46	0.06	9.117	76.8383	0.269552
200 batch	5.09805	440.77	1.3	7.424	17.7009	0.225976
300 batch	5.2814	493.82	2.52	8.68	29.8744	0.207425

References

- https://github.com/MatteoBasili/sabd-progetto2-2024_25

Demo

- Open terminal
- Enter the root directory of the project
- Launch containers
  - $ docker compose up --build –d
- Execute the full pipeline
  - $ python3 ./scripts/run_all.py [--limit N]
- Check the results (CSV files) in the directory Results/csv