2024.10.09.Wed.11-12-08 - Converstations - Studying the Energy Consumption of Stream Processing Systems

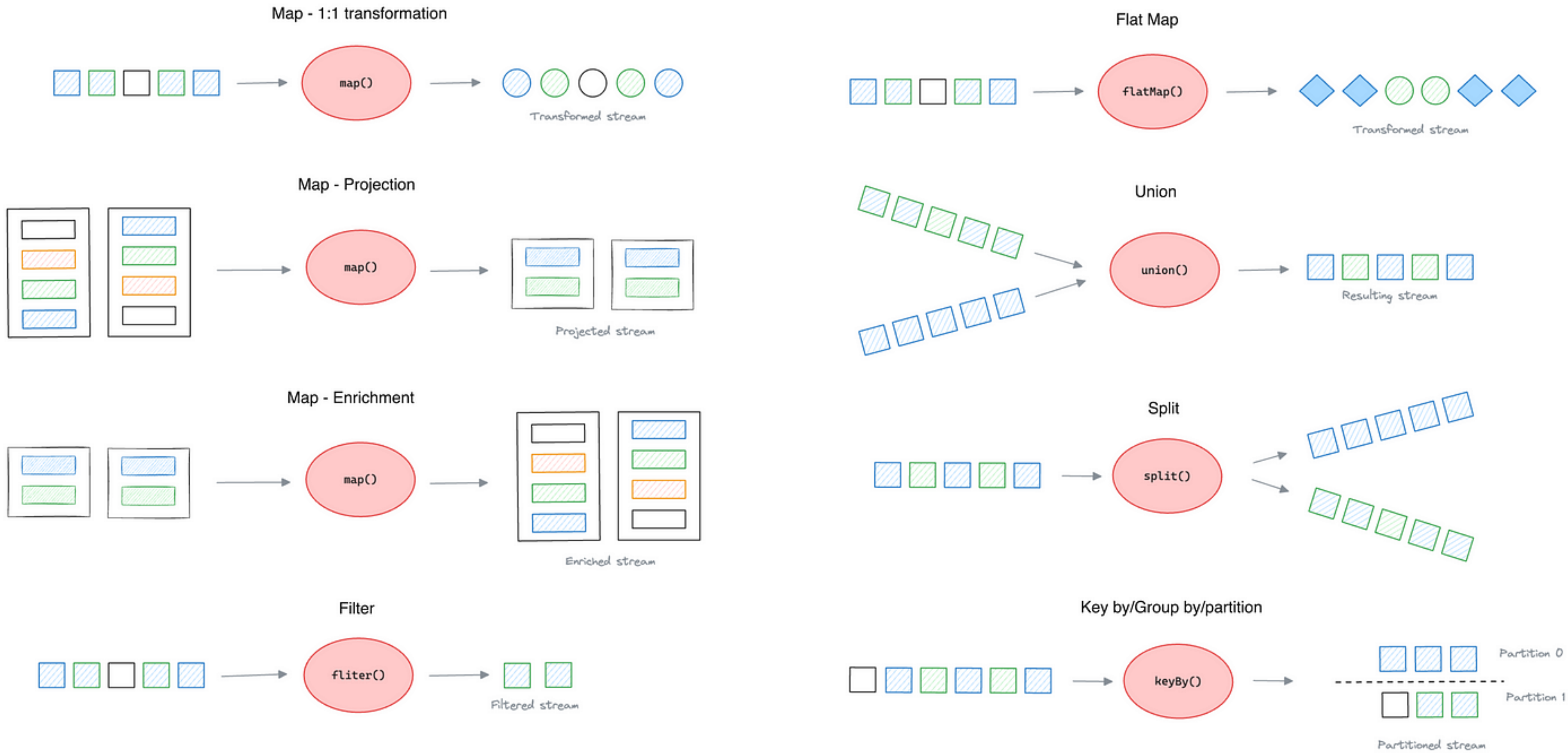
2024.10.09.Wed.11-12-08 - Converstations - Studying the Energy Consumption of Stream Processing Systems

Text Stream

Converstations - Studying the Energy Consumption of Stream Processing Systems in the cloud

1. Introduction
   - Background
     - What is the cloud?
     - Energy consumption in data centers is a growing concern
     - UN climate goals require significant reductions in emissions
     - Software optimization is crucial for further energy savings

- Data Stream Processing (DSP) systems



- Increasingly popular for handling big data

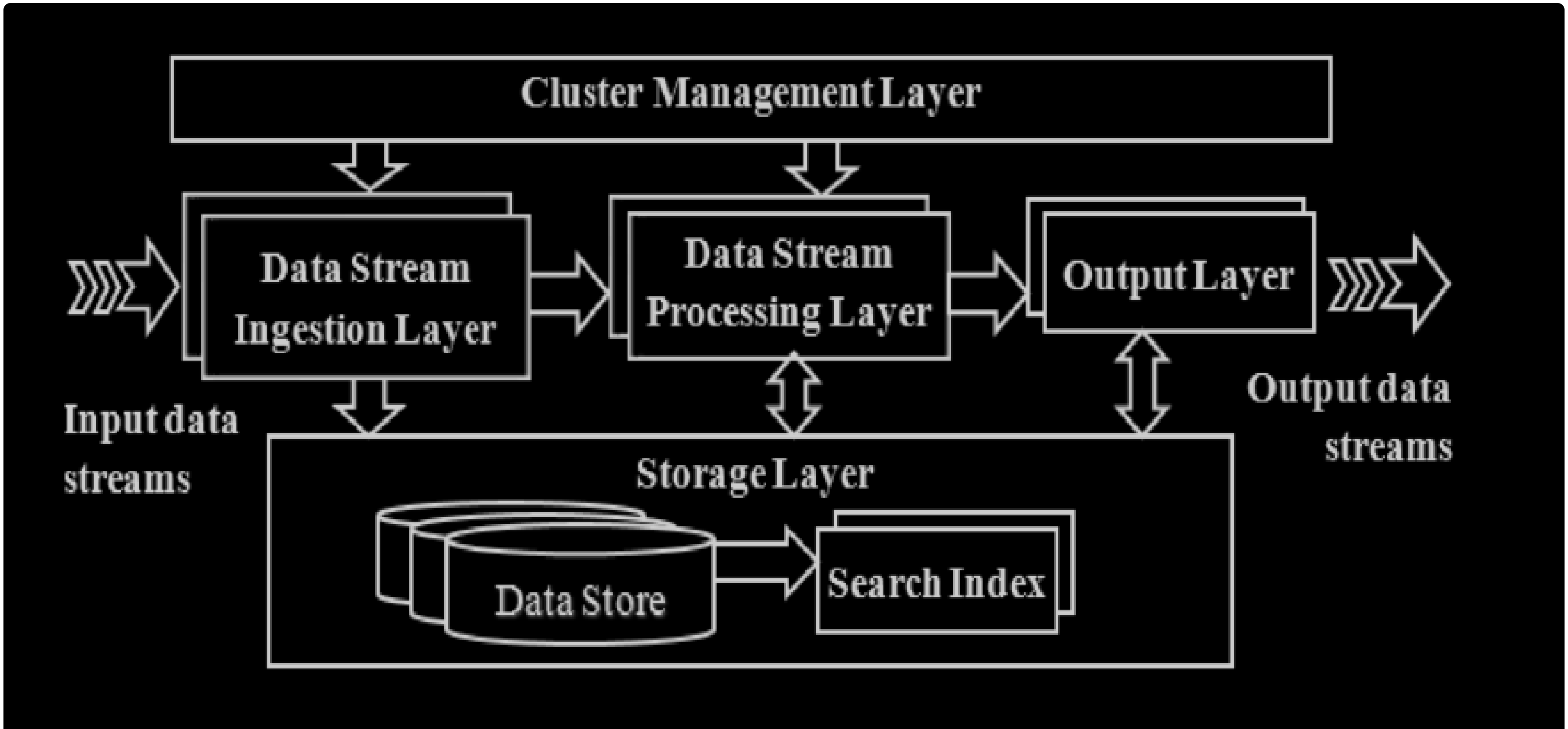


Fig. 1. Architecture of a typical data stream processing system (adapted from [21]).

- Process continuous streams of data in real-time
- Used by major companies to process billions of events daily

- Can we improve these systems?
  - Challenges in studying DSP energy consumption
    - Distributed across multiple servers
    - Complex systems with many interdependent components
    - Large parameter space
    - Highly dynamic and adaptive nature

- Need for standardized benchmarks and metrics
- Proposal: GreenFlow - Tool to study energy consumption of systems

1. Background
   - DSP systems
     - Key components: ingestion, processing, output, storage, cluster management
     - Popular engines: Apache Flink, Kafka Streams, Apache Spark
     - Typically deployed using VMs or container orchestration platforms
     - Wide parameter space (e.g., parallelism, buffer sizes, window sizes)
   - Energy Metrology
     - Wattmeter reading vs. software power meters
     - RAPL (Running Average Power Limit)
     - Scaphandre software power meter
2. Related Works
   - Horizontal studies (comparing different implementations)
     - Limited due to complexity and lack of standardization
   - Vertical studies (specific optimization approaches)
     - More common, but often not focused on cloud-native deployments
   - Theodolite: cloud-native Kubernetes implementation for benchmarking
   - Gap in research for cloud-native deployment settings
3. Methodology: GreenFlow
   - Design principles
   1. Declarative configurable deployment
   2. Cloud native design
   3. Best practices for reproducibility
   - System Design
   1. Configuration management

    - Gin Config framework
    - TinyDB for metadata storage
  2. Energy measurement
    - Scaphandre software power meter
    - Kubernetes pod-level attribution
  3. Metrics management
    - Prometheus for metric collection
    - PromQL for querying
    - Grafana for visualization
  4. Experiment lifecycle
    - Resource provisioning (Grid'5000)
    - Kubernetes deployment
    - Theodolite framework for workload generation
  5. Interface
    - Real-time monitoring and configuration
    - Historical data exploration

4. Evaluation
  - Experimental Setup
    - Grid'5000 testbed
    - UC-3 workload from Theodolite