Data Stream Management

Big Data Management

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

DTIM
www.essi.upc.edu/dtim

Knowledge objectives

1. Define a data stream
2. Distinguish the two kinds of stream management systems
3. Recognize the relevance of stream management
4. Enumerate the most relevant characteristics of streams
5. Explain to which extent a DBMS can manage streams
6. Name 10 differences between DBMS and SPE
7. Characterize the kinds of queries in an SPE
8. Explain the two parameters of a sliding window
9. Explain the three architectural patterns
10. Explain the goals of Spark streaming architecture
11. Draw the architecture of Spark streaming

Understanding Objectives

1. Identify the need of a stream ingestion pattern
2. Identify the need of a near-real time processing pattern
3. Identify the kind of message exchange pattern
4. Simulate the mesh-join algorithm
5. Estimate the cost of the mesh-join algorithm
6. Use windowing transformations in Spark streaming

Basics

Tens of thousands of elements/events per second

- Internet traffic analysis
- Trading on Wall Street
- Fraud detection (i.e., credit cards)
- Highway traffic monitoring
- Surveillance cameras
- Command and control in military environments
- Log monitoring
  - Google receives several hundred million search queries per day
- Click analysis
  - Yahoo! Accepts billions of clicks per day
- RFID monitoring
  - Venture Development Corporation predicted in 2006 that RFID can generate in Walmart up to 7TB/day (≈ 292GB/hour ≈ 5GB/minute ≈ 80MB/second)
- Scientific data processing (i.e., sensor data)
  - One million sensors reporting at a rate of ten per second would generate 3.5TB/day (only 4 bytes per message)
  - Large Hadron Collider (LHC) at CERN
    - Collisions are produced at 40MHz, which generates approximately 40TB/second
      - Cluster of 39 nodes with a total memory of 18TB and 1658 cores, containing 32 HBase region servers

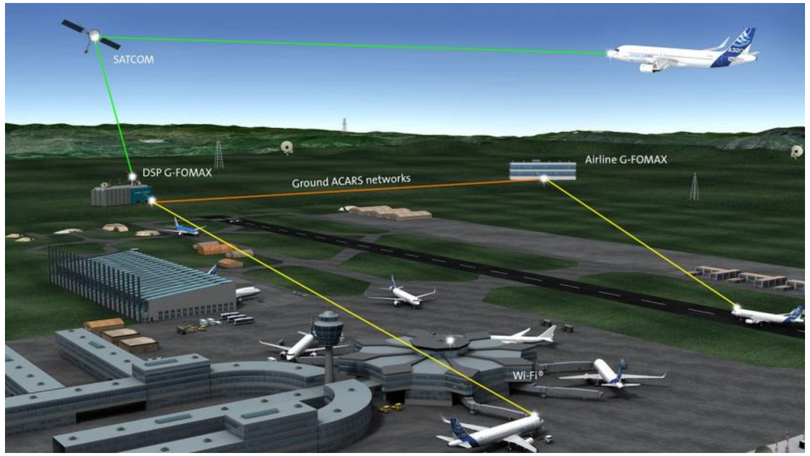
Danish wind turbines

- One park:
  - 100+ turbines
- One turbine:
  - 500 sensors
  - More than 2500 derived data streams
- One sensor:
  - 8 bytes sampled at 100+Hz

100 turbines*2500 streams*100 samples/sec = $25 \cdot 10^6$ samples/second
8bytes*$25 \cdot 10^6$ samples/second*3600seconds/hour*24hours/day = 17.5TB/day
17.5TB/day*365 days/year = 6+ PB/year/park

Having thousands of parks and storing 20+ years of history …

Aerospace corporation



Collins Corp.

- One (not big) airline
  - 125 planes
- One plane:
  - 24.000 sensors (Flight Operations & Maintenance Exchanger, FOMAX)
  - 10 hours/day
- One sensor
  - 8 bytes sampled at 20+Hz

125 planes*24.000 sensors*20 samples/sec = $60 \cdot 10^6$ samples/second
8 bytes*$60 \cdot 10^6$ samples/second*3600seconds/hour = 1.73TB/hour
1.73TB/hour*10 hours/day*365 days/year = 6+ PB/year

Having tens of airlines and storing 10+ years of history …

Streaming use cases

- Triggers
  - Rise alerts
- Data enrichment
  - Join static data
- Continuous learning
  - Create ML models online
- Streaming ETL
  - Pre-processing (filter, aggregate, etc.) before storage

https://www.datanami.com/2015/11/30/spark-streaming-what-is-it-and-whos-using-it

Stream characterization

1. Arrival rate not under the control of the system
   - Faster than processing time -- algorithms must work with only one pass of the data
2. Unbounded memory requirements -- Some drastic reduction is needed
3. Keep the data moving -- Only volatile storage
4. Support for near-real time application -- Latency of 1 second is unacceptable
   - Need to scale and parallelize
5. Arrival order not guaranteed -- Some data will be delayed
6. Imperfections must be assumed -- Some data will be missing
7. There is temporal locality -- Data (characteristics) evolve over time
8. **Approximate** (not accurate) **answers** are acceptable
   - Deterministic outputs

Data Stream Management System (DSMS)

"Class of software systems that deals with processing streams of high volume messages with very low latency"

M. Stonebraker

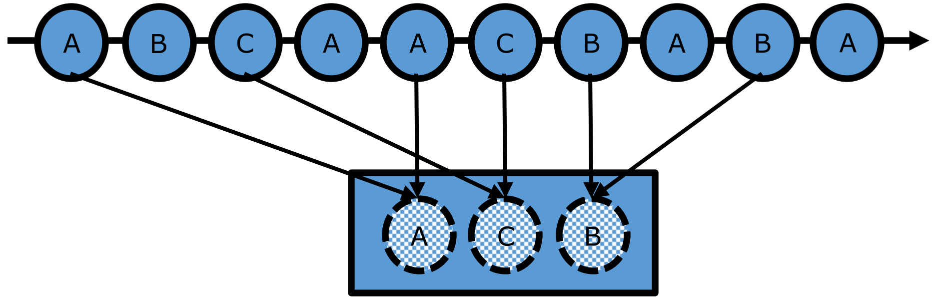
- Concept introduced in 1992
- Messages constantly arrive at a high pace
- Sub-second latency

D. B. Terry et al. *Continuous Queries over Append-Only Databases*. SIGMOD Conference 1992

Kinds of systems

- Stream Processing Engine
  - Focus on
    - Near-real time processing and scalability
    - Offering windowing operations to define aggregates
  - Tools
    - Spark streaming
    - Flink
    - Storm
    - S4

- Complex Event Processing/Pattern Matching Engine
  - Focus on
    - Offering windowing operations to define indicators (based on thresholds)
    - Express complex temporal correlations
  - Tools
    - Esper
    - Aleri
    - StreamBase
    - T-Rex
    - Huawey PME
    - Orange CRS network monitoring



Pattern

SPE vs CEP

Stream Processing Engine	Complex Event Processing
Keep data moving	***Pattern*** *identification*
Window aggregates definition	***Pattern*** *expressions*
Handle stream imperfections	
Integrate stored and streamed data	***State*** *management*
High availability of ***processing***	*High availability of* ***patterns***
Process *distribution*	***States*** *distribution*

Challenges of SPE

1) Unify batch and live parallel processing model

2) Out-of-order processing
   - *Low watermark*

A "low watermark" is a timestamp that signals that all events with a timestamp less than or equal to the watermark value have likely arrived. The ones arriving "late" are dropped …

3) State management
   - Custom user-defined state with stateful operators
     - Provide consistency guarantees in front of failures

P. Carbone et al. *Beyond Analytics: The Evolution of Stream Processing Systems*. SIGMOD Conference, 2020

Challenges of SPE

4) Fault tolerance and high availability
   - Provide *at-least-once* and *exactly-once* semantics
     - Maintaining snapshots of states, migrating states and scaling out

At-Least-Once: Guarantees an event will be processed at least once. It might be processed multiple times if a failure occurs (leading to duplicates, which might be acceptable for some use cases).

Exactly-Once: The gold standard, guaranteeing each event is processed exactly once, even in the face of failures, preventing both data loss and duplication.

5) Load management and elasticity
   - *Load shedding* and *back-pressure*

Load shedding: drop less critical data to prevent a complete collapse
Back-pressure: apply a signal upstream to slow down the data ingestion

P. Carbone et al. *Beyond Analytics: The Evolution of Stream Processing Systems*. SIGMOD Conference, 2020

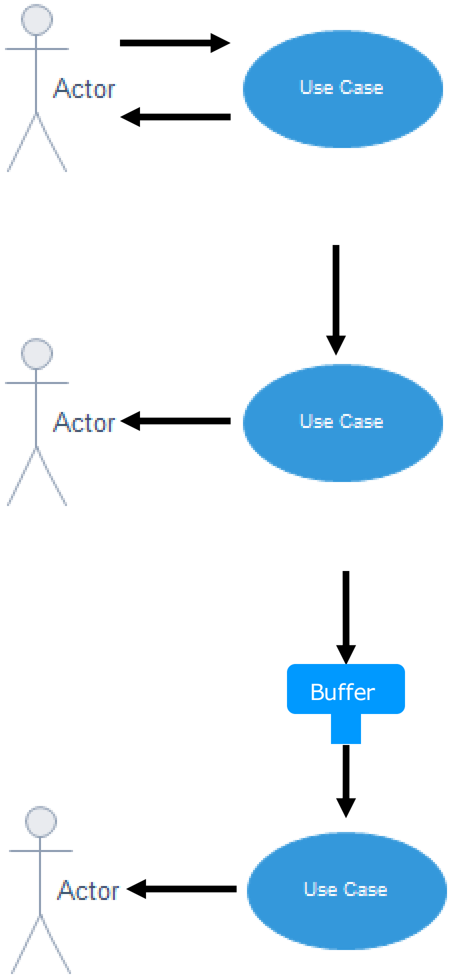
Characterization of operations in SPE

Kinds of system interaction

CRUD

Stream

Micro-batch



Kinds of queries

- Depending on the trigger
  - Standing
  - Ad-hoc
- Depending on the output
  - Alerts
  - Result set
- Depending on the inputs
  - Based on a summary of history
    - Synopsis/Sketches
  - Based on the *X* last elements
    - Based on the last element (*X*=1)
    - Sliding window (*X*>1)

Tumbling & Sliding window examples

Window Duration = 5
Sliding Duration = 5

Fixed-size, non-overlapping, and contiguous segments
e.g., Calculating the total sales for every hour

20.6 20.5 20.6 20.5 20.5 | 20.5 20.4 20.4 20.3 20.2 | 20.1 20.0 20.1 20.1 20.2 | 20.1 20.0 19.9 20.0 20.1

Window Duration = 5
Sliding Duration = 3

Fixed-size, but they overlap
e.g., "sliding window" of 10 minutes, sliding every 1 minute: every minute, you get a new window containing the temperature readings from the past 10 minutes.

19.5 19.6 19.7 19.8 19.9 20.0 20.1 20.0 20.1 20.1 20.1 20.2 20.2 20.3 20.4 20.5 20.5 20.4 20.5 20.5

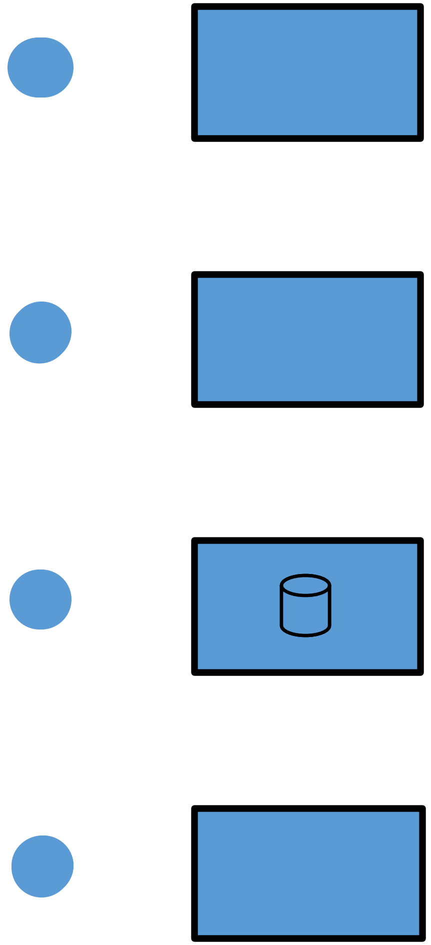
Kinds of operations

Filter

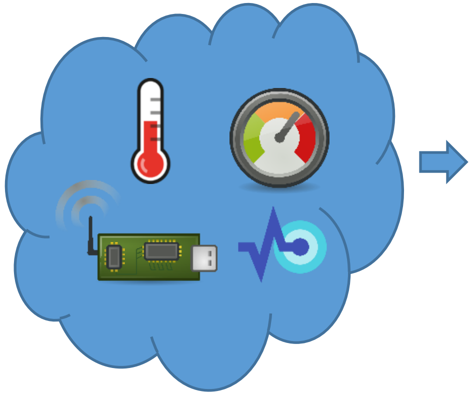
Project

Lookup

Aggregation



Processing example



Filter
kindOfSensor="Termometer"

Aggregate
windowDuration=5min
slidingDuration=5min
difference=max-min

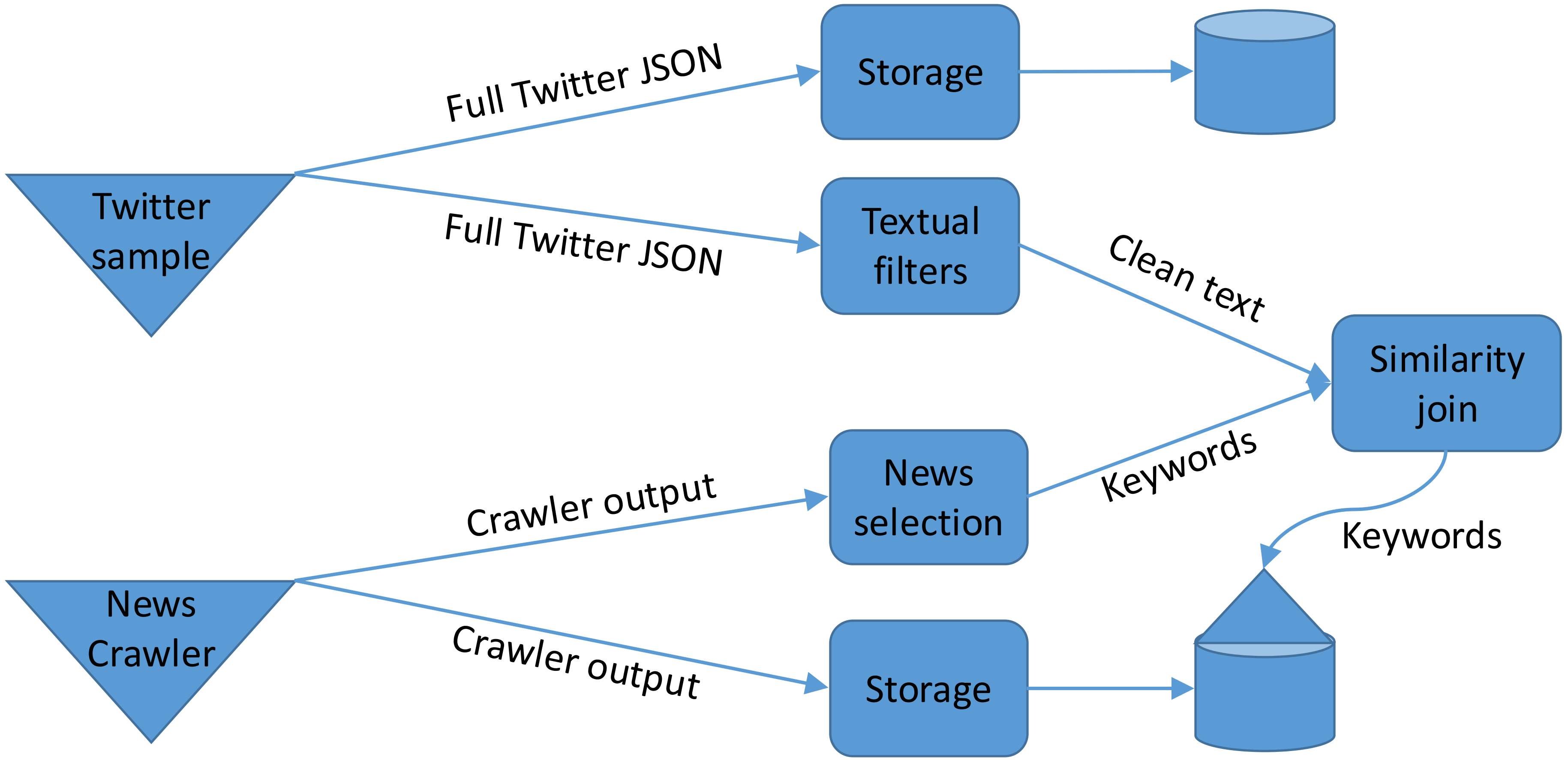
Filter
difference>0.1

Aggregate
windowDuration=60min
slidingDuration=60min
count

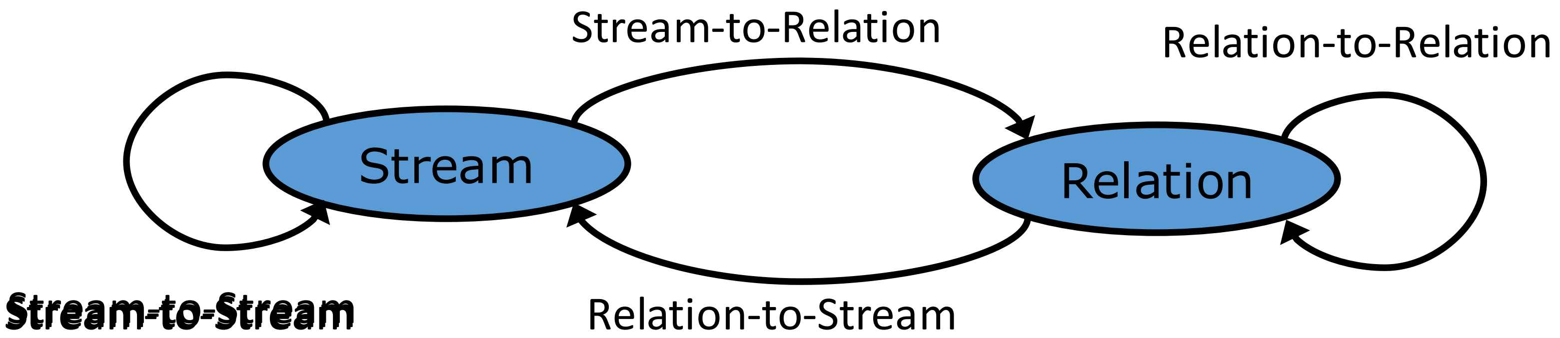
Filter
count>0

Binary operations

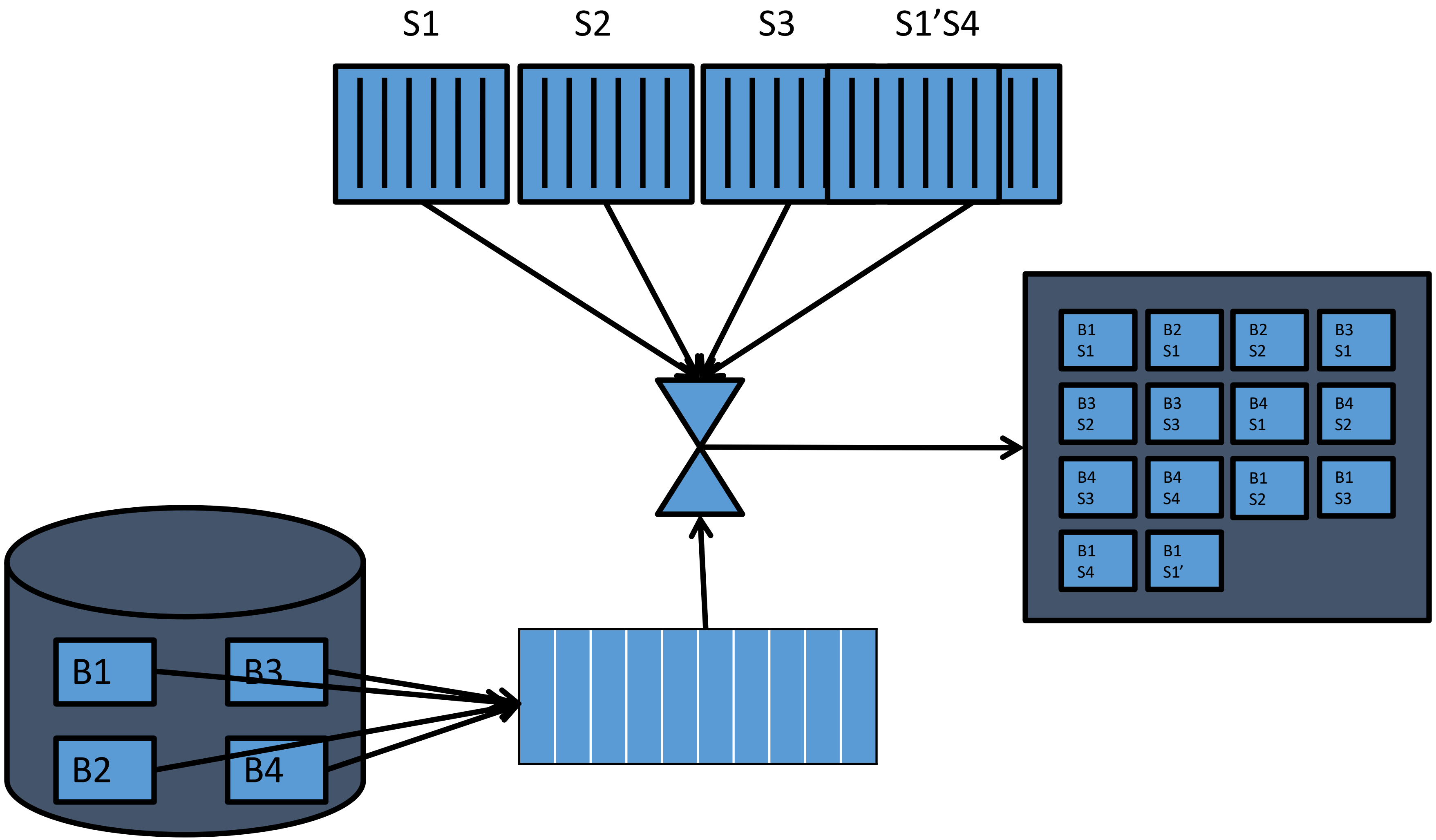
Stream-to-Stream



Kinds of binary operations



Meshjoin algorithm example



Meshjoin algorithm

- Algorithm
  - Performs a cyclic scan of the table and keeps a sliding window of the stream in memory

```
while true do
    read next block of R into a memory page/buffer
    if memory is full then
            dequeue w messages
    endif
    foreach m in memory do
            generate m join R (for the current block)
    endforeach
endwhile
```

D = Time to retrieve one block
C = CPU time to process one message
B = Blocks of R
R_S= Stream messages per page
w = Stream messages removed per loop
λ = Arrival-stream rate (messages/sec)
μ = Service-join rate (messages/sec)

1 for the buffer, 1 for the output (before writing to disk)

- Cost (in time) of one loop (assuming M+2 memory pages)
  - $D+M \cdot R_S \cdot C = D+(B \cdot w/R_S) \cdot R_S \cdot C = D+w \cdot B \cdot C$
- Considerations (aiming at $\lambda \leq \mu$)
  - We try to maximize μ given M
    - $w = M \cdot R_S/B \Rightarrow M = B \cdot w/R_S$
    - $\mu = w/(D+w \cdot B \cdot C) = 1/(D/w+B \cdot C)$
  - This would almost always be faster than Row Nested Loops
    - $\mu = 1/(h \cdot D+D)$

Stream processing

Relational temporary tables

CREATE GLOBAL TEMPORARY TABLE <tablename> (…)
[ON COMMIT {DELETE ROWS|PRESERVE ROWS}];

- Relational mapping
  - Each element is a tuple
  - The sliding window is a relation
- Data is not persistent
  a) Transaction specific
  b) Session specific
- Does **not** support:
  - Foreign keys
  - Cluster
  - Partitions
  - Parallelism

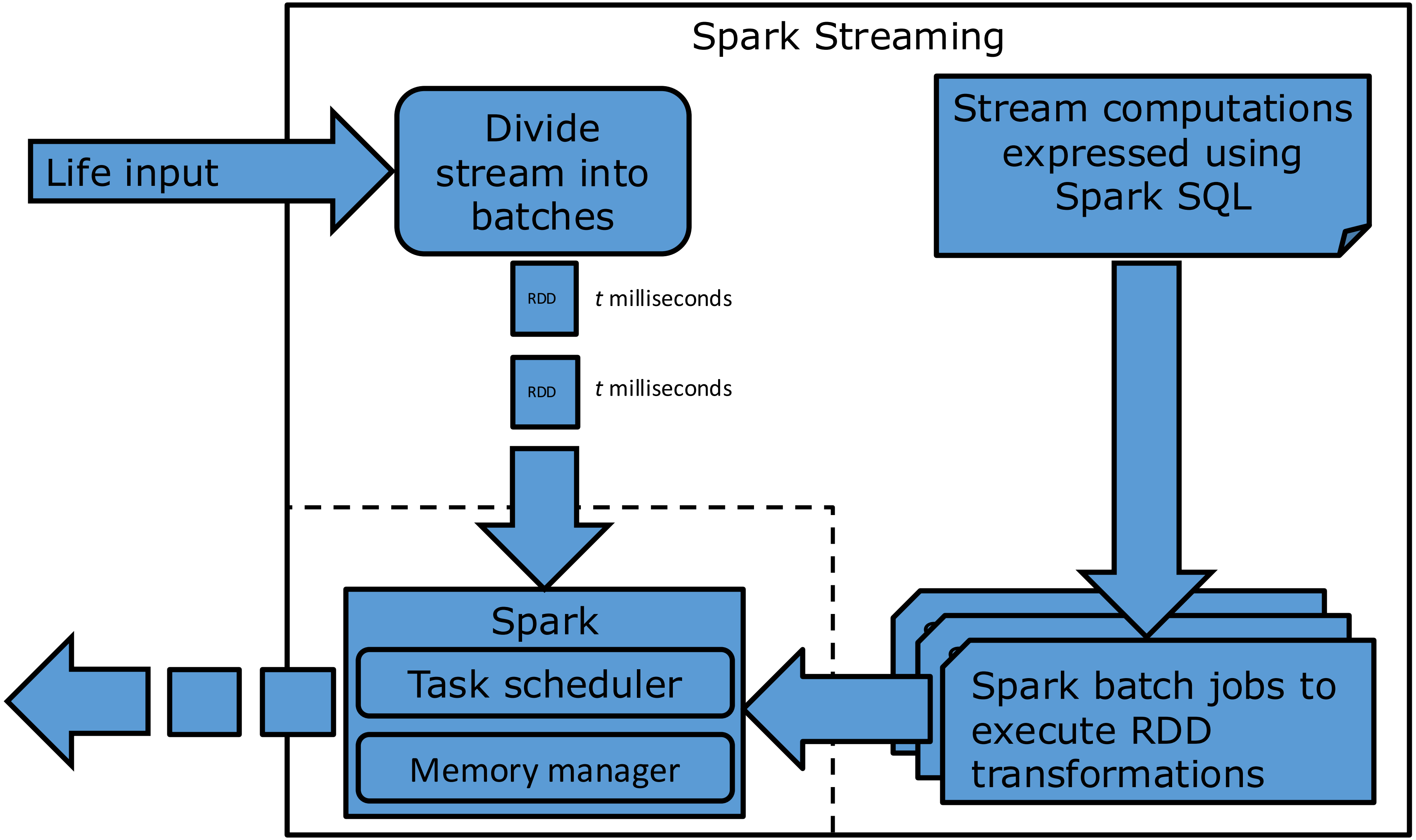
Databases vs Streams

	Database management	Stream management
Data	Persistent	Volatile
Access	Random	Sequential
Queries	One-time	Continuous
Support	Unlimited disk	Limited RAM
Order	Current state	Sorted
Ingestion rate	Relatively low	Extremely high
Temporal requirements	Little	Near-real time
Accuracy	Exact data	Imprecise data
Heterogeneity	Structured data	Imperfections
Algorithms	Multiple passes	One pass

Spark streaming goals

- Scalability to hundreds of nodes
- Minimal overhead
  - Sub-second latency
    - End-to-end: ~100milliseconds
- Recovery from faults and stragglers
  - On reception, data is replicated to a second executor in another worker
  - State (i.e., summary) is periodically (e.g., every 10 RDDs) saved to a reliable file system

Micro-batch processing engine



Structured Stream

- Based on Dataframes
  - Different output modes generate …
    - Complete: … the whole result
    - Append: … only new rows
    - Update: … only the rows that changed (e.g., the latest aggregate for each key)
- Any number of stream queries can be started in a single Spark session
- Incremental processing
  - Maintains intermediate state for partial aggregates (implemented as watermarking)
    - Late data can update aggregates of old windows correctly
    - Supports stream-stream joins
- Two execution modes
  1. Micro-batch processing
     - Exactly-once and fault-tolerance guarantees
  2. Continuous processing
     - At-least-once guarantees
       - Only for map-like operations

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Input/Output

- Input sources
  - Parquet files (in a directory)
  - Kafka
  - Socket (in the driver)
  - Rate (key-value pairs self-generation for testing/benchmarking purposes)
- Triggers
  - End of processing previous batch (default)
  - Fixed Interval (establishes max frequency)
  - One-time/Available-now
  - Continuous
- Output sinks
  - Parquet files (in a directory)
  - Kafka
  - Foreach/ForeachBatch
  - Console (for debugging)
  - Memory (for debugging)

https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Structured Streaming example

```
# Create DataFrame representing the stream of input lines from connection to host:port
lines = spark.readStream.format('socket').option('host', host).option('port', port).load()

# Split the lines into words, an unbounded dataframe with a word column
words = lines.select(
    # explode turns each item in an array into a separate row
    explode(split(lines.value, ' ')).alias('word'))

# Generate running word count, unbounded dataframe that will continuously update
wordCounts = words.groupBy('word').count()

# Start running the query that prints the running counts to the console
query = wordCounts.writeStream.outputMode('complete').format('console').start()

query.awaitTermination()
```

https://github.com/apache/spark/blob/v3.3.1/examples/src/main/python/sql/streaming/structured_network_wordcount.py

Unsupported dataframe operations

- Multiple chained aggregations
- Limit/Take
- Sorting
- Distinct
- Outer joins
- Count (requires grouping)
- Foreach (requires writing the stream first)
- Show (use console)

Architectural patterns for stream/event processing

Architectural patterns

A. Stream ingestion

B. **Near**-real time
- Non-partitioned
  - Get profile information (lookup) needed for decisions
  - Requires nearly no coding beyond the application-specific logic
- Partitioned
  - Define a key to partition data
    - Match incoming data to the subset of the context data that is relevant to it

C. Pattern matching

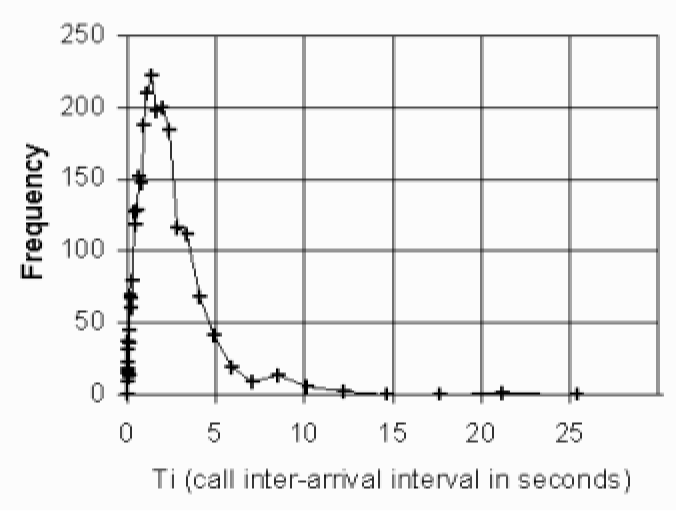
D. Complex topology
- Aggregation
- Machine learning

https://blog.cloudera.com/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop

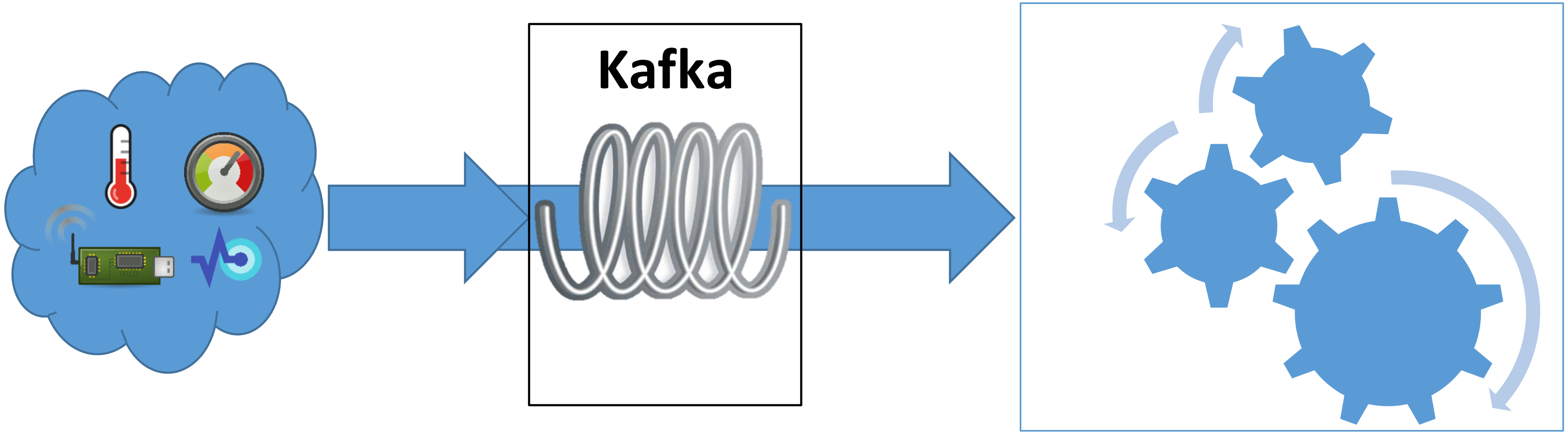
A. Stream ingestion

- The objective is to not lose any event

Trunked Radio System of E-Comm 9-1-1

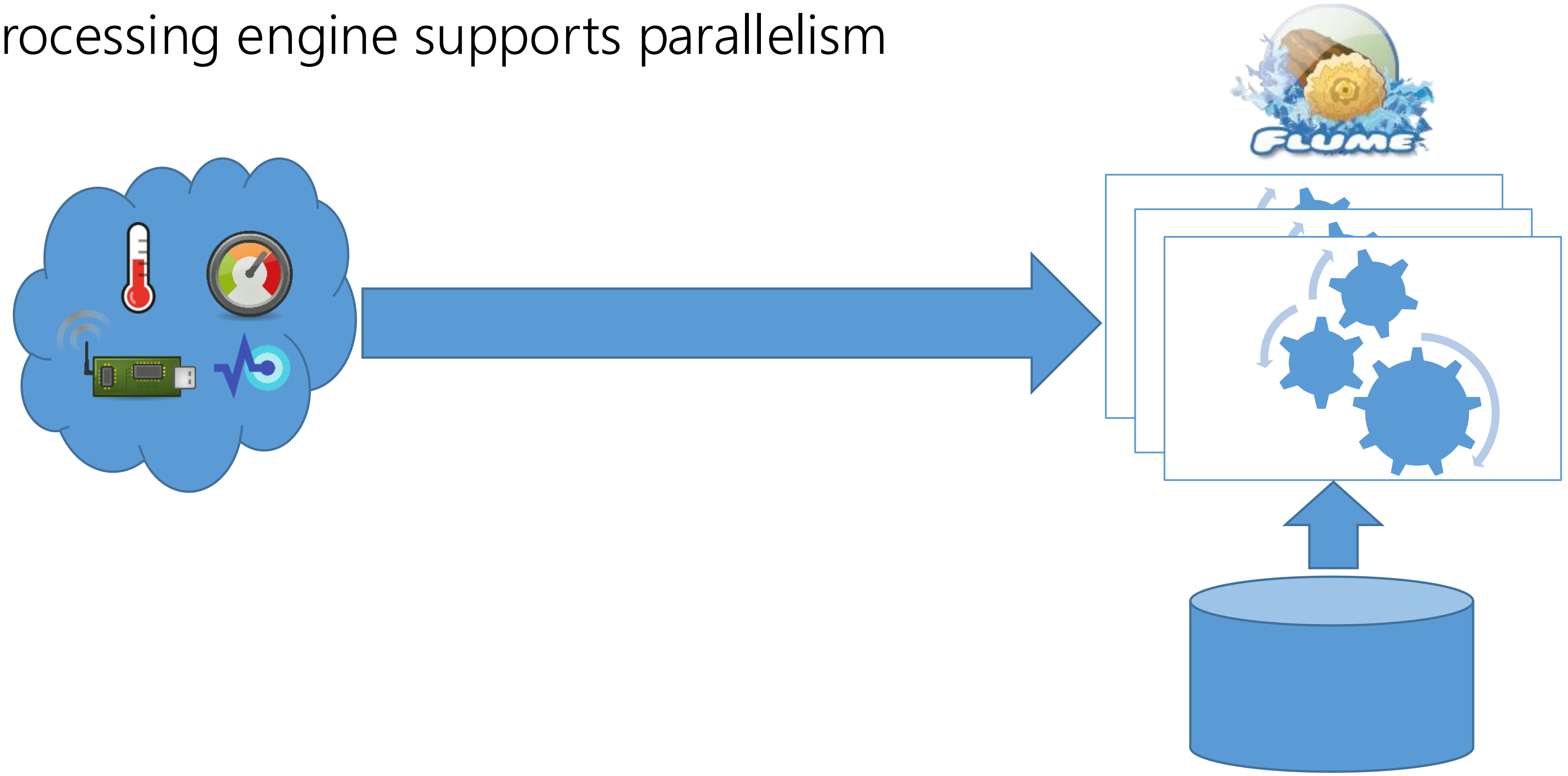


By D. Sharp



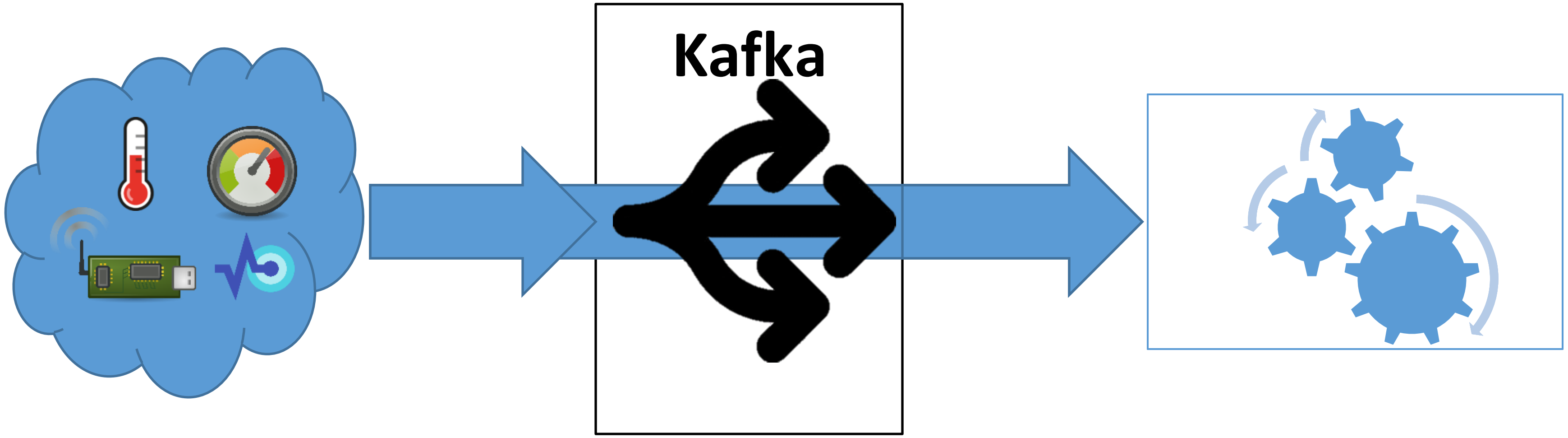
B. Near-real time event processing (I)

- The objective is to react as soon as possible
  - Processing engine supports parallelism



B. Near-real time event processing (II)

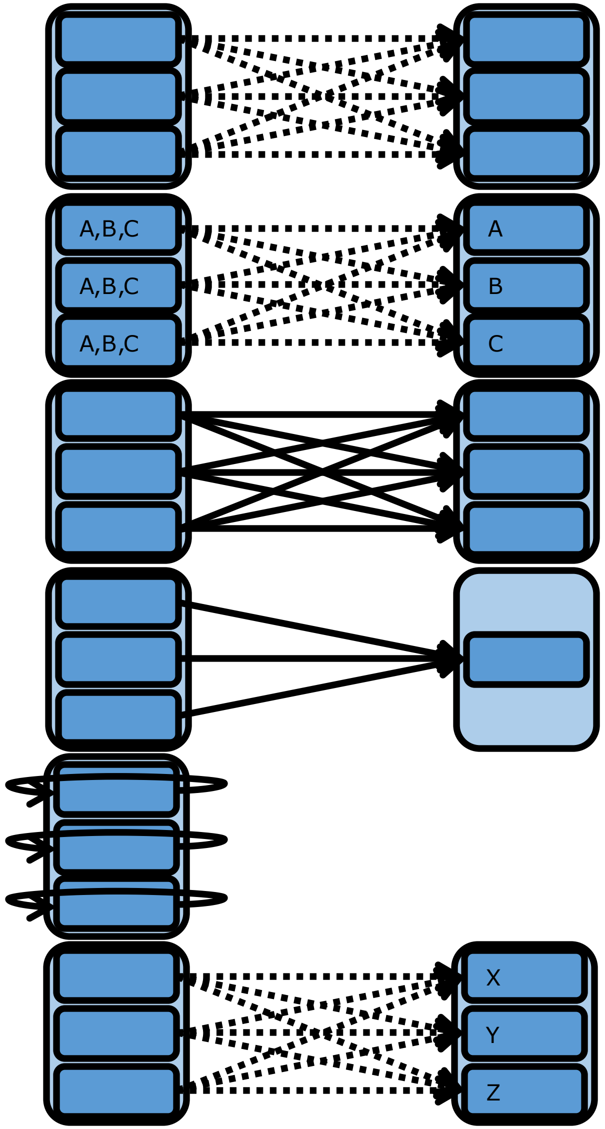
- The objective is to react as soon as possible
  - Processing engine does not support parallelism



C. Complex Event Processing

- Pattern matching
  - State keeps all potential matches
    - Tree
    - NFA (Non-deterministic Finite Automata)
- Hard to distribute
- Consider
  - Time constraints
  - Absence of events
  - Re-emitting complex events

D. Complex topology



- Shuffle grouping (good for balancing the workload)
  - Random
- Fields grouping
  - Same value, same task
- All grouping
  - Broadcast to all task
- Global grouping
  - All data converges to one task
- None grouping
  - Execution stays in the same thread (if possible)
- Direct grouping (not depending on the fields …)
  - Producers direct the output to a concrete task

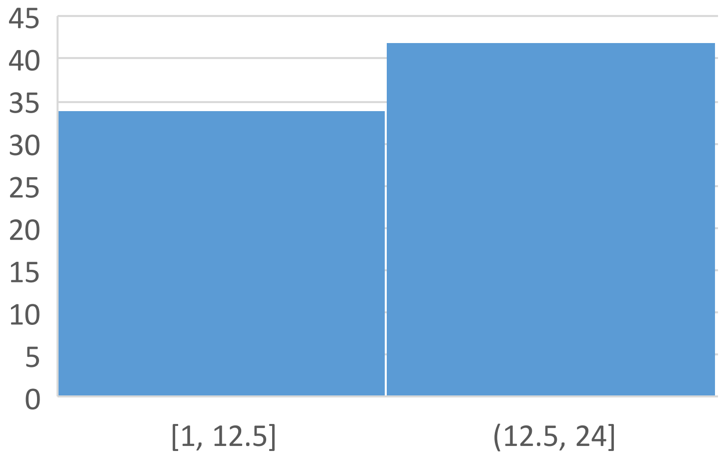
Algorithmic challenges and approaches

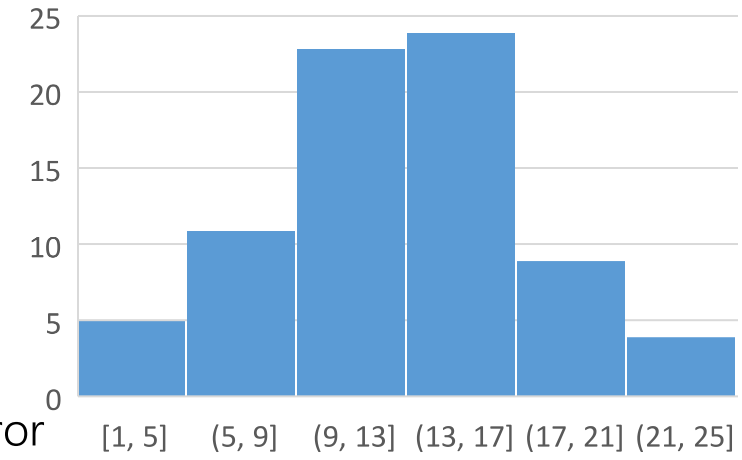
Constraints

- Data cannot be stored
  - One-pass algorithms with
    - Bounded processing time
    - Bounded resources (i.e., memory)
      - At most, logarithmic on the size of the stream
    - Answer available at any time
- Processing must be on-line
  - Bounded response time for both
    a) Summary update
    b) Response retrieval

Challenges and approaches

- Limited computation capacity
  - Sampling (i.e., Load shedding)
    - Probabilistically drop stream elements
  - Filtering (i.e., Bloom filters)
- Limited memory capacity
  - Sliding window -> Discard elements
    - Aging (use only most recent data)
  - Exponentially decaying window -> Weight elements
  - Synopsis -> Approximate solutions
    - Examples:
      - Histograms - Works under uniform distribution of values in a bucket
      - Concise sampling - Works under a limited number of distinct values
      - Heavy hitters - Uses logarithmic memory space
      - Sketching - Space needed depends on error and probability of that error





Closing

Summary

- Stream definition and characterization
  - Complex event processing
- Streaming architectural patterns
- Streaming operations
  - Sliding windows
  - Binary operations
- Spark streaming
  - Architecture

References

- M. Stonebraker et al. *The 8 Requirements of Real-Time Stream Processing*. SIGMOD Record 34(4), 2005
- N. Polyzotis et al. *Meshing Streaming Updates with Persistent Data in an Active Data Warehouse*. IEEE Trans. Knowl. Data Eng. 20(7), 2008
- L. Liu and M.T. Özsu (Eds.). Encyclopedia of Database Systems. Springer, 2009
- I. Botan et al. *Flexible and Scalable Storage Management for Data-intensive Stream Processing*. EDBT, 2009
- T. Akidau et al. *The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in MassiveScale, Unbounded, OutofOrder Data Processing*. VLDB, 2015
- I. Flouris. *Issues in complex event processing: Status and prospects in the Big Data era*. J. of Systems and Software 127, 2017
- P. Carbone et al. *Beyond Analytics: The Evolution of Stream Processing Systems*. SIGMOD Conference, 2020