Data Integration and Large Scale Analysis

11- Stream Processing

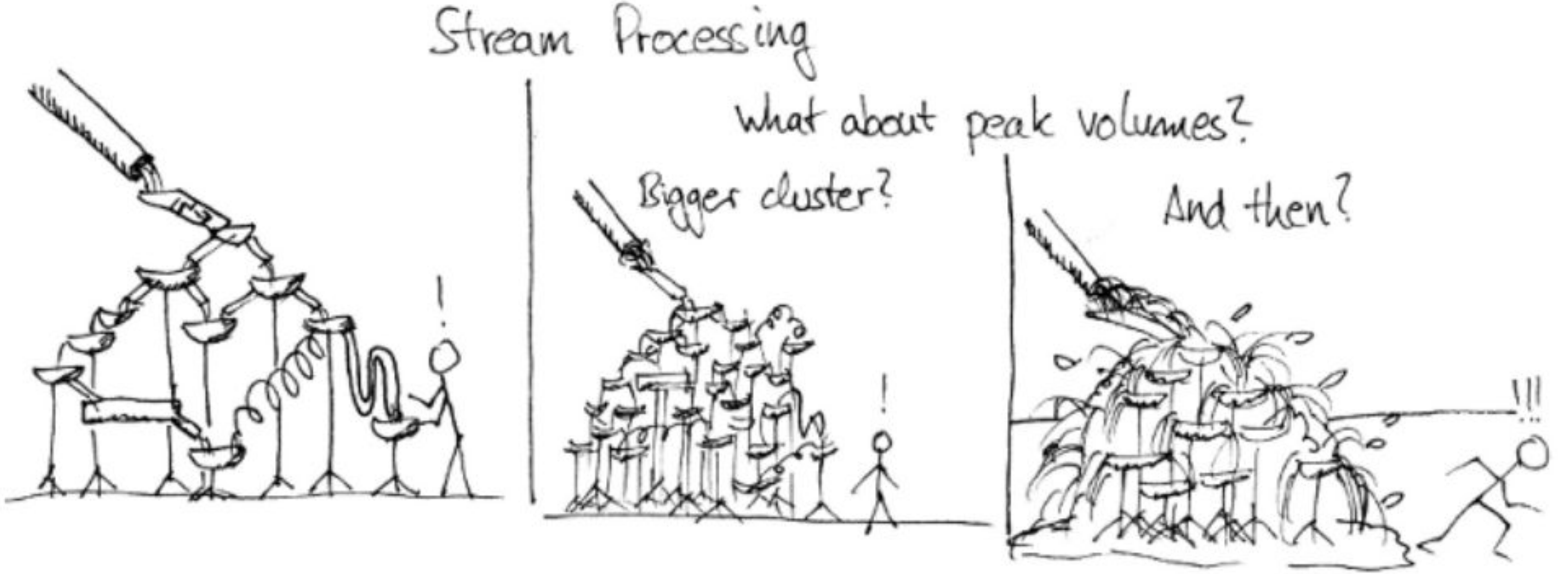
Lucas Iacono. PhD. - 2026

Institute of
HUMAN-CENTRED COMPUTING

Data Stream Processing

HCC
Institute of
HUMAN-CENTRED COMPUTING

Data Stream Processing



HCC
Institute of
HUMAN-CENTRED COMPUTING

Data Stream Processing: Terminology

Ubiquitous Data Streams

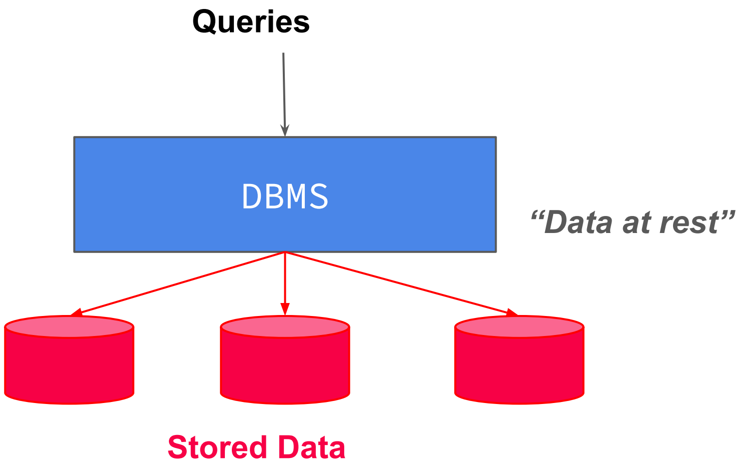
Event and message streams (e.g., clicks, X, IoT, Machines)

Characteristics	
Event Streams	**Message Streams**
• Time-oriented. • Immutable • Real-time processing	• System communication • Asynchronous • Structured

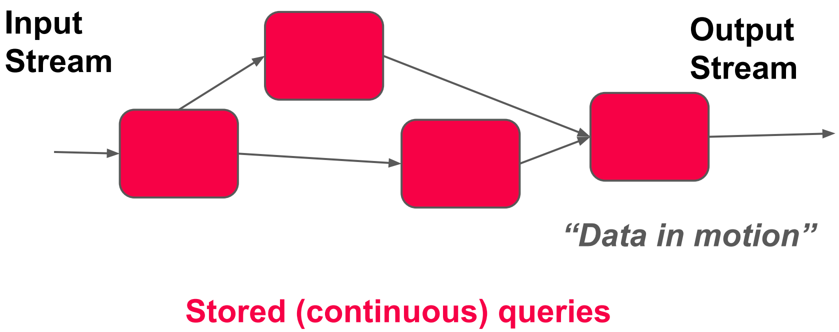
Data Stream Processing: Terminology

Stream Processing Architecture

- **Infinite input streams,** often with windows semantics
- Continuous queries



Stream Processing Engines



Data Stream Processing: Terminology

Use Cases

- **Monitoring and alerting** (notifications on events / patterns)
- **Real-time reporting** (aggregate statistics for dashboards)
- **Real-time ETL** and event-driven data updates
- Real-time decision making (fraud detection)
- **Data stream mining** (summary **statistics** w/ limited memory)

Data Stream

- Unbounded stream of data tuples $S = (s1, s2, \ldots sn)$. Each data tuple *(si)* → *si=(ti,di)*
- *ti* = timestamp, *di* = data
- *S={(10:30:00 09082025 ,22.5),(10:30:00 09082025 ,22.7),(10:30:00 09082025 ,22.8),…}*

Data Stream Processing: Terminology

Real-time Latency Requirements

- **Real-time:** guaranteed task completion by a given deadline (30 fps)
- **Near Real-time:** few milliseconds to seconds
- In practice → **used with much weaker meaning**

Challenges in Real-Time Systems:

- Resource Constraints (memory, computing, storage)
- Latency (e.g. data in a queue waiting for been processed)

Data Stream Processing: History of Stream Processing Systems

2000s

- **Data stream management systems: STREAM** (Stanford'01), **Aurora** (Brown/MIT/Brandeis'02), **TelegraphCQ** (Berkeley'03) → **mostly unsuccessful in industry/practice**
- **Message-oriented middleware** and **Enterprise Application Integration** (EAI): IBM Message Broker, SAP eXchange Infra

Data Stream Processing: History of Stream Processing Systems

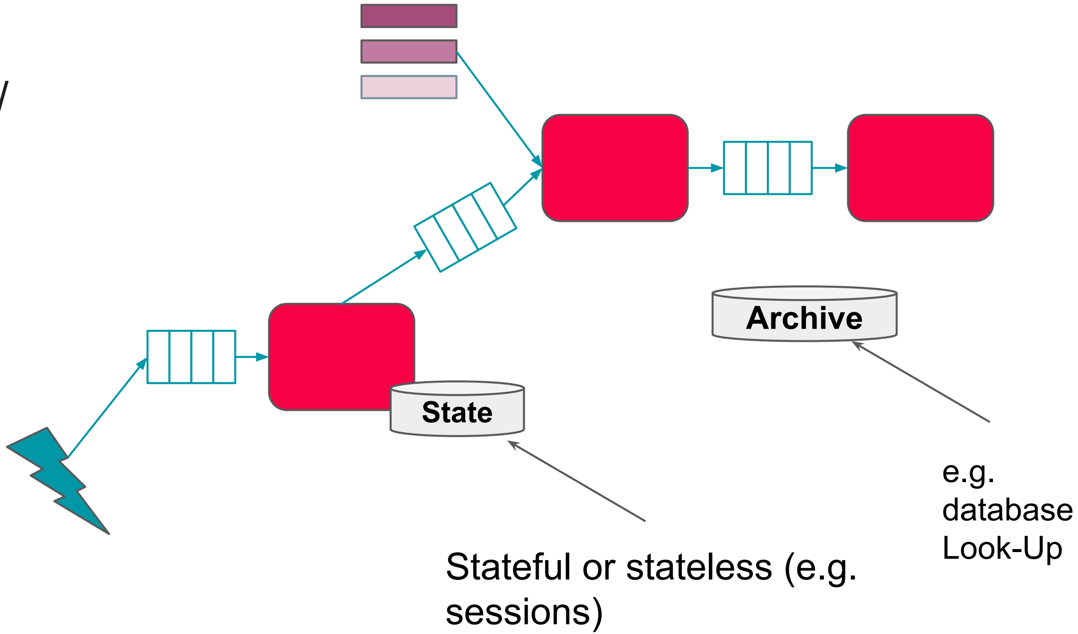
2010s

- **Distributed stream processing engines**, and “unified” batch/stream processing
- **Proprietary systems**: Google Cloud Dataflow, MS StreamInsight / Azure Stream Analytics, IBM InfoSphere Streams / Streaming Analytics, AWS Kinesis
- **Open-source systems:** Apache Spark Streaming (Databricks), **Apache Flink** (Data Artisans), **Apache Kafka** (Confluent), **Apache Storm**

System Architecture - Native Streaming

Basic System Architecture

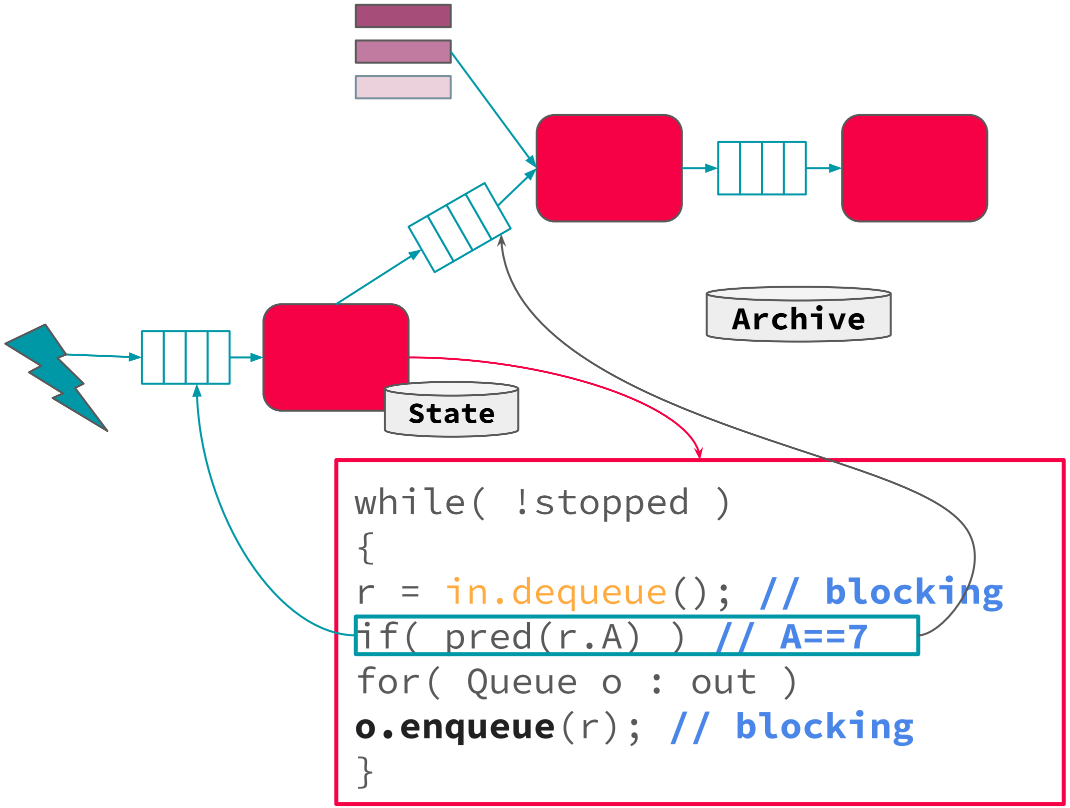
- Data flow graphs (potentially w/ multiple consumers)
- **Nodes** asynchronous ops (w/ state) (e.g., separate threads)
- **Edges** data dependencies (tuple/message streams)
- **Push model** data production controlled by source



System Architecture - Native Streaming

Operator Model

- Read from input queue
- Write to potentially many output queues



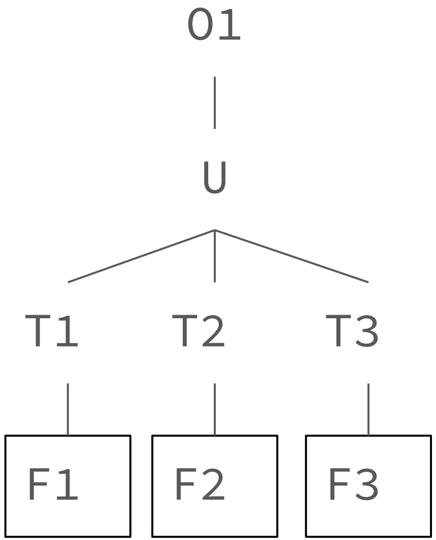
System Architecture - Sharing

Multi-Query Optimization

- Given **set of continuous queries** compile minimal DAG w/o redundancy **-> subexpression elimination -> avoid redundant operations and share intermediate results between queries to improve system efficiency**.

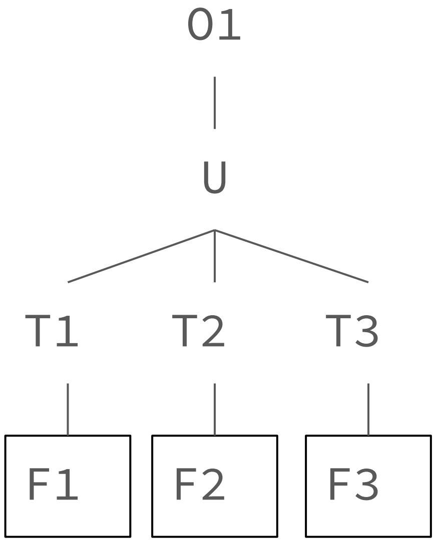
System Architecture - Sharing

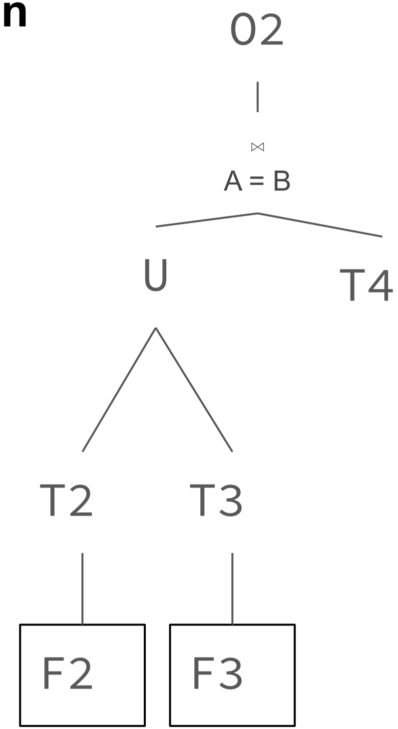
Multi-Query Optimization



System Architecture - Sharing

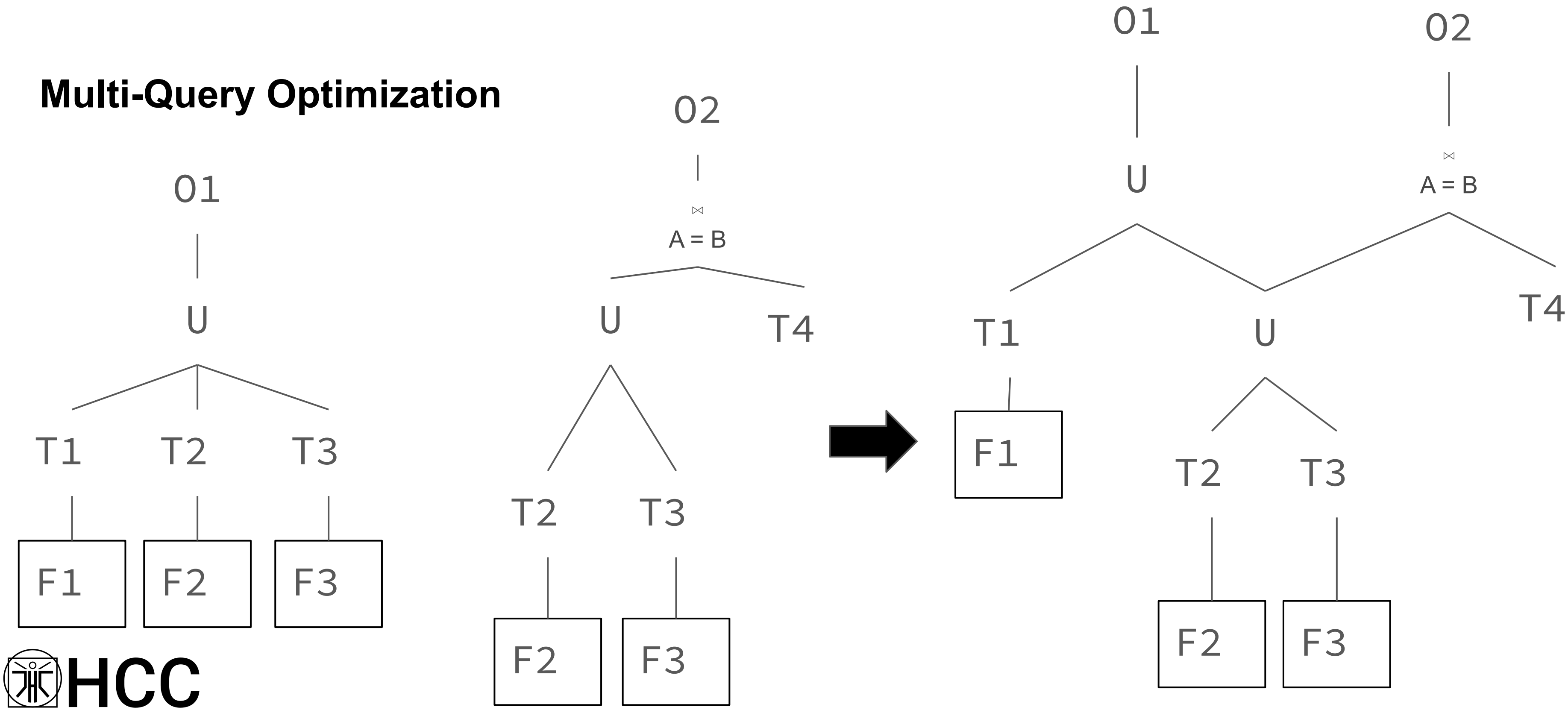
Multi-Query Optimization





System Architecture - Sharing

Multi-Query Optimization



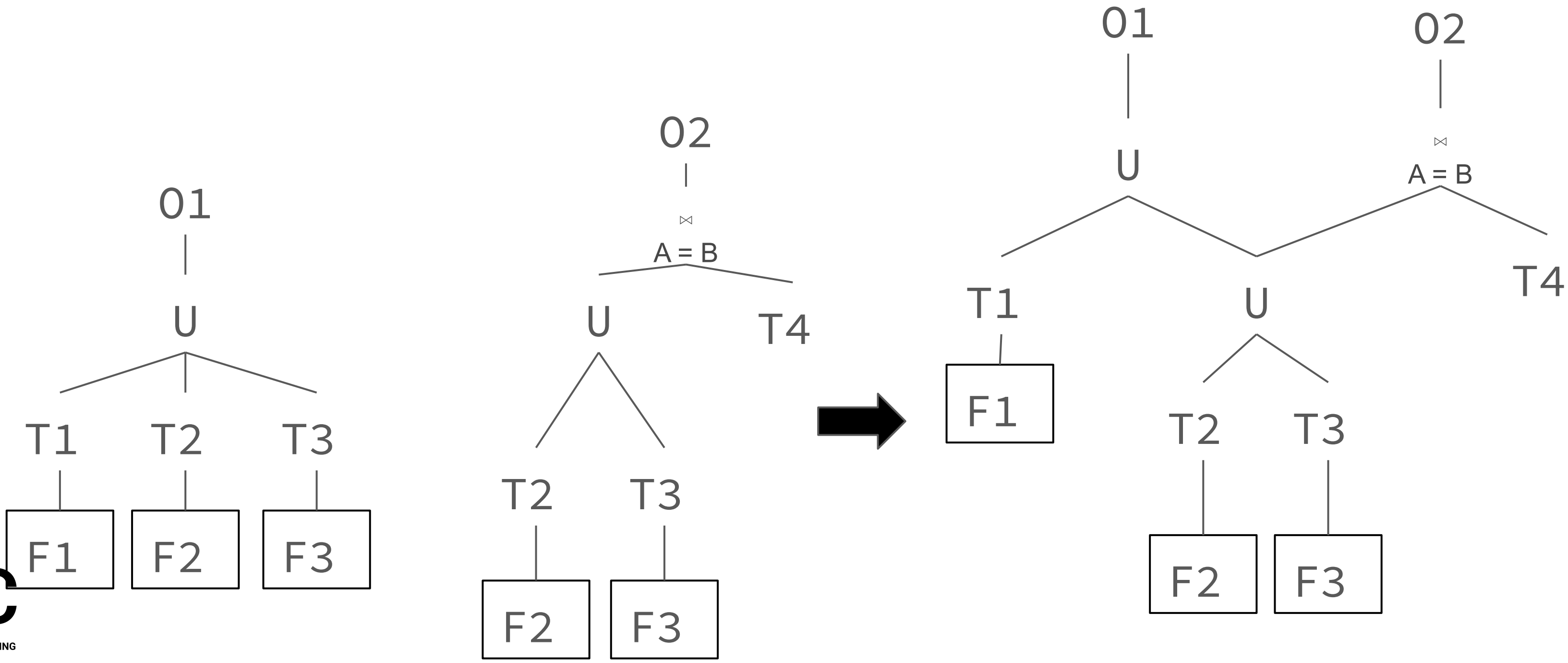
Institute of HUMAN-CENTRED COMPUTING

System Architecture - Sharing

Operator Sharing: complex ops w/ multiple predicates for adaptive reordering

Queue Sharing: share results with multiple queries

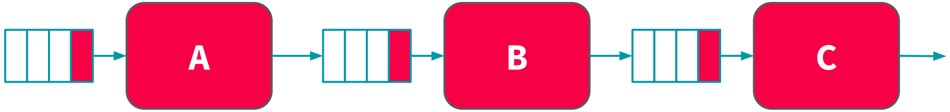
Multi-Query Optimization



Institute of HUMAN-CENTRED COMPUTING

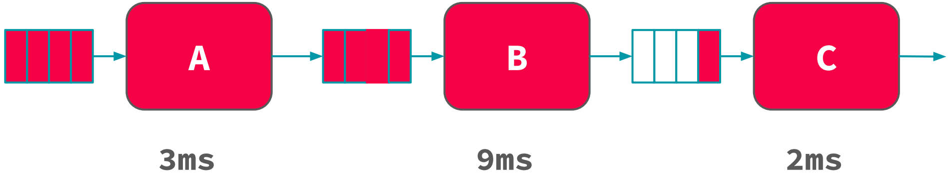
System Architecture - Handling Overload

- **Back Pressure**
  - Graceful handling of overload w/o data loss
  - **Slow down sources**
  - E.g. blocking queues



System Architecture - Handling Overload

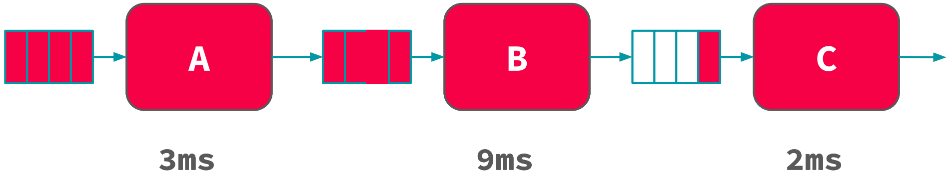
- **Back Pressure**
  - Graceful handling of overload w/o data loss
  - **Slow down sources**
  - E.g. blocking queues



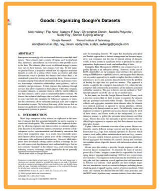
Self-adjusting operator scheduling
Pipeline runs at rate of **slowest op**

System Architecture - Handling Overload

- **Back Pressure**
  - Graceful handling of overload w/o data loss
  - **Slow down sources**
  - E.g. blocking queues
- **Load Shedding**
  - **Random-sampling-**based load shedding
  - **Relevance-based** load shedding



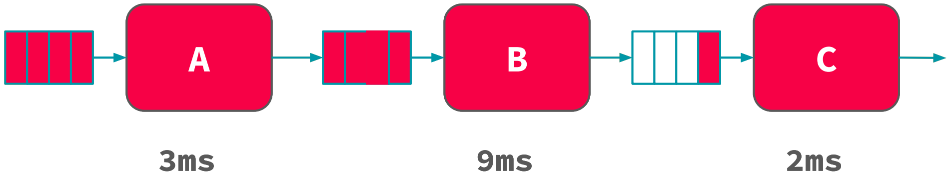
Self-adjusting operator scheduling
Pipeline runs at rate of **slowest op**


Goods: Organizing Google's Datasets

[Nesime Tatbul et al: Load Shedding in a Data Stream Manager. VLDB 2003]

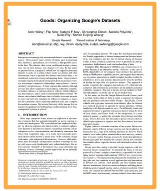
System Architecture - Handling Overload

- **Back Pressure**
  - Graceful handling of overload w/o data loss
  - **Slow down sources**
  - E.g. blocking queues



Self-adjusting operator scheduling
Pipeline runs at rate of **slowest op**

- **Load Shedding**
  - **Random-sampling-**based load shedding
  - **Relevance-based** load shedding
  - **Summary-based** load shedding (synopses)
  - Given SLA, select queries and shedding placement that minimize error and satisfy constraints

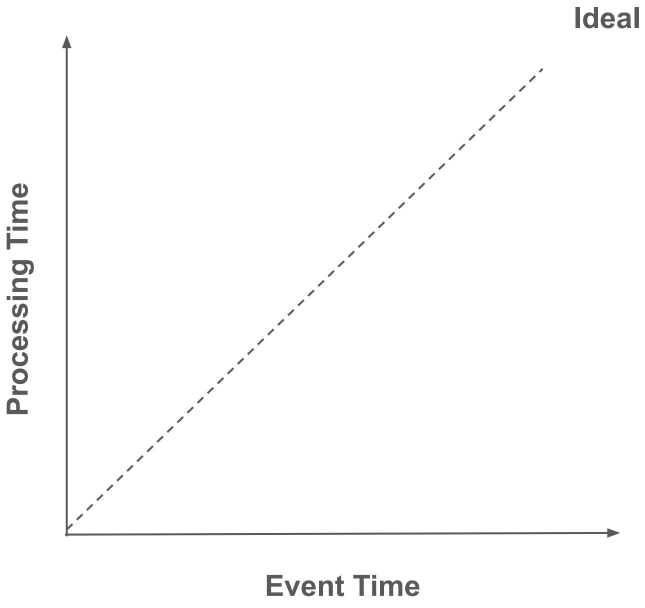


[Nesime Tatbul et al: Load Shedding in a Data Stream Manager. VLDB 2003]

- **Distributed Stream Processing**
  - Data flow partitioning (distribute the **query**)
  - Key range partitioning (distribute the **data** stream)

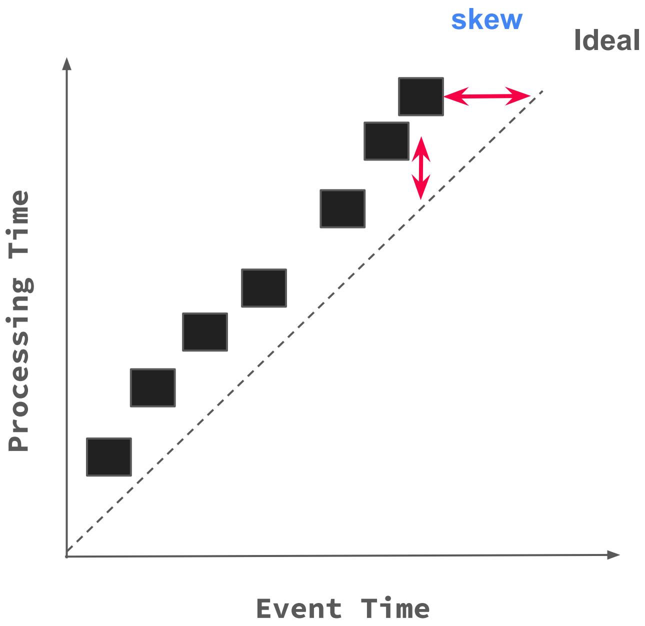
Time (Event, System, Processing)

- **Event Time**
  - Real time when the **event/data item was created**
- **Ingestion**
  - System time when the **data item was received**
- **Processing Time**
  - System time when the **data item is processed**



Time (Event, System, Processing)

- **Event Time**
  - Real time when the **event/data item was created**
- **Ingestion**
  - System time when the **data item was received**
- **Processing Time**
  - System time when the **data item is processed**
- **In practice**
  - **Delayed and unordered** data items
  - Use of heuristics (e.g **watermarks = delays threshold**)
  - Use of more complex triggers (late results)



HCC
Institute of
HUMAN-CENTRED COMPUTING

Durability and Consistency Guarantees

- **At Most Once**
  - **“Send and forget”**, ensure data is never counted twice
  - Might cause data loss on failures

Durability and Consistency Guarantees

- **At Most Once**
  - **"Send and forget"**, ensure data is never counted twice
  - Might cause data loss on failures
- **At Least Once**
  - **"Store and forward"** deliver the message until reception of the acknowledgements from receiver
  - Might create incorrect state (processed multiple times)

Durability and Consistency Guarantees

- **At Most Once**
  - **"Send and forget"**, ensure data is never counted twice
  - Might cause data loss on failures
- **At Least Once**
  - **"Store and forward"** deliver the message until reception of the acknowledgements from receiver
  - Might create incorrect state (processed multiple times)
- **Exactly Once**
  - **"Store and forward" w/ guarantees** regarding state updates and sent msgs
  - Often via dedicated transaction mechanisms (**hand-shaking protocols**)

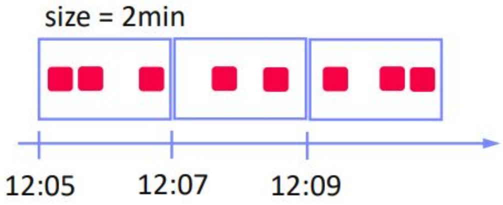
Window Semantics

- **Windowing Approach**
  - Many operations like joins/aggregation **undefined over unbounded streams**
  - Compute operations over windows of **(a)** **time** or **(b)** **elements counts**

Window Semantics

- **Windowing Approach**
  - Many operations like joins/aggregation **undefined over unbounded streams**
  - Compute operations over windows of **(a)** **time** or **(b)** **elements counts**

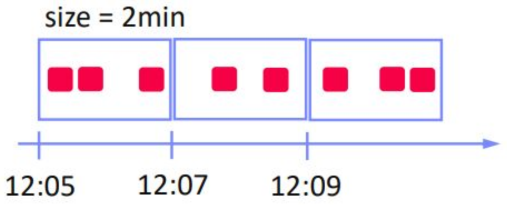
- **Tumbling Window**
  - Every data item is only part of a single window
  - **Aka Jumping window**



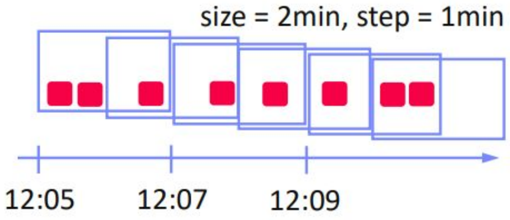
Window Semantics

- **Windowing Approach**
  - Many operations like joins/aggregation **undefined over unbounded streams**
  - Compute operations over windows of (a) time or (b) elements counts

- **Tumbling Window**
  - Every data item is only part of a single window
  - **Aka Jumping window**



- **Sliding Window**
  - Time- or tuple-based sliding windows
  - Insert new and expire old data items



Stream Joins I

Basic Stream Join

- **Tumbling window:** use classic join methods
- **Sliding window** (symmetric for both R and S) →Applies to arbitrary join pred

Stream Joins I

Basic Stream Join

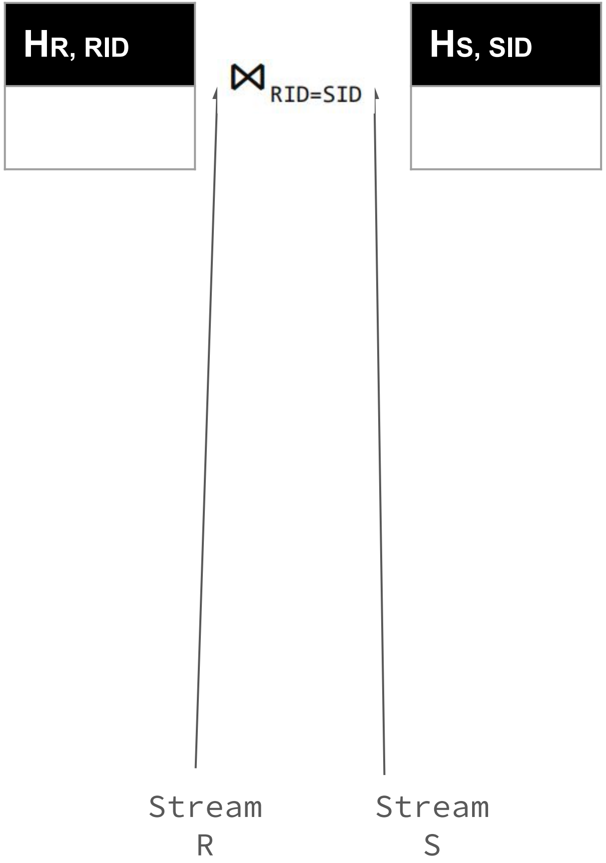
- **Tumbling window:** use classic join methods
- **Sliding window** (symmetric for both R and S) → Applies to arbitrary join pred

For each new r in R:

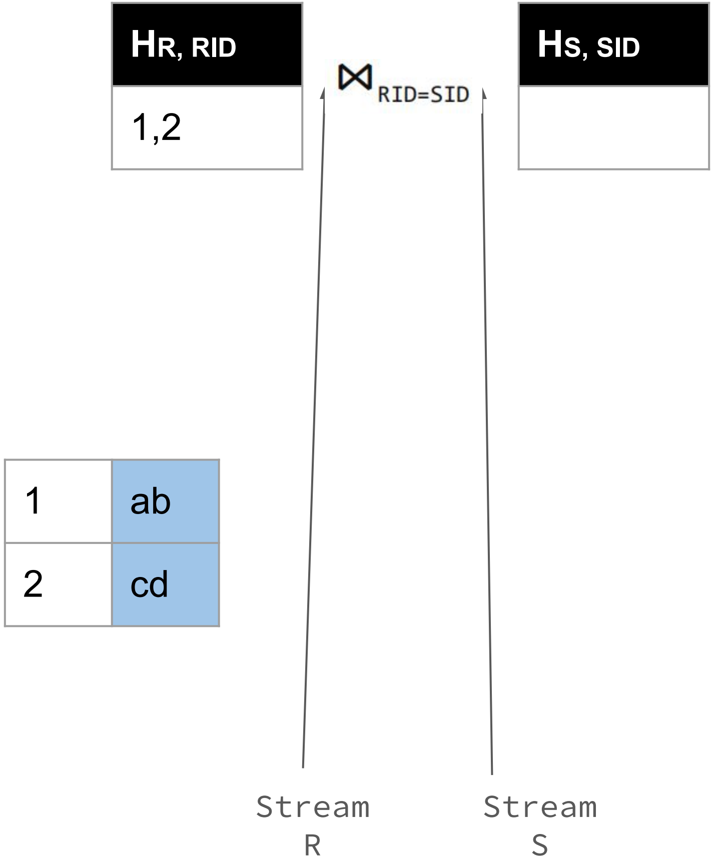
a. **Scan** window of stream S to find match tuples
b. **Insert** new r into window of stream R
c. **Invalidate** expired tuples in window of stream R

Stream Joins II

- **Double-Pipelined Hash Join**
  - Join of bounded streams (or unbounded w/ invalidation)
  - Equijoin predicate, symmetric and non-blocking
  - For every incoming tuple (e.g. left): probe (right)+emit, and build (left)

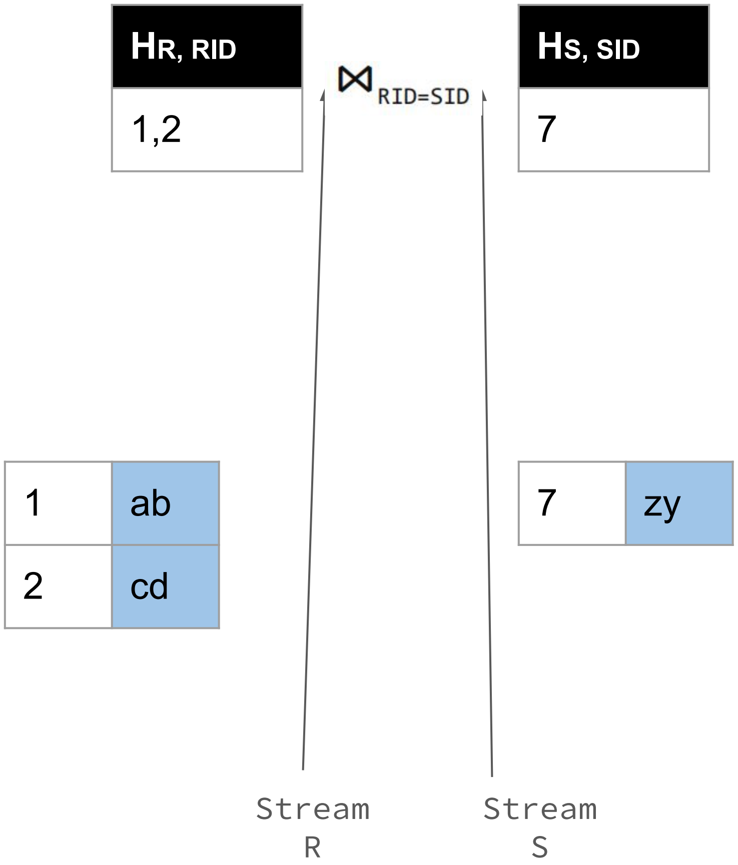


Stream Joins II



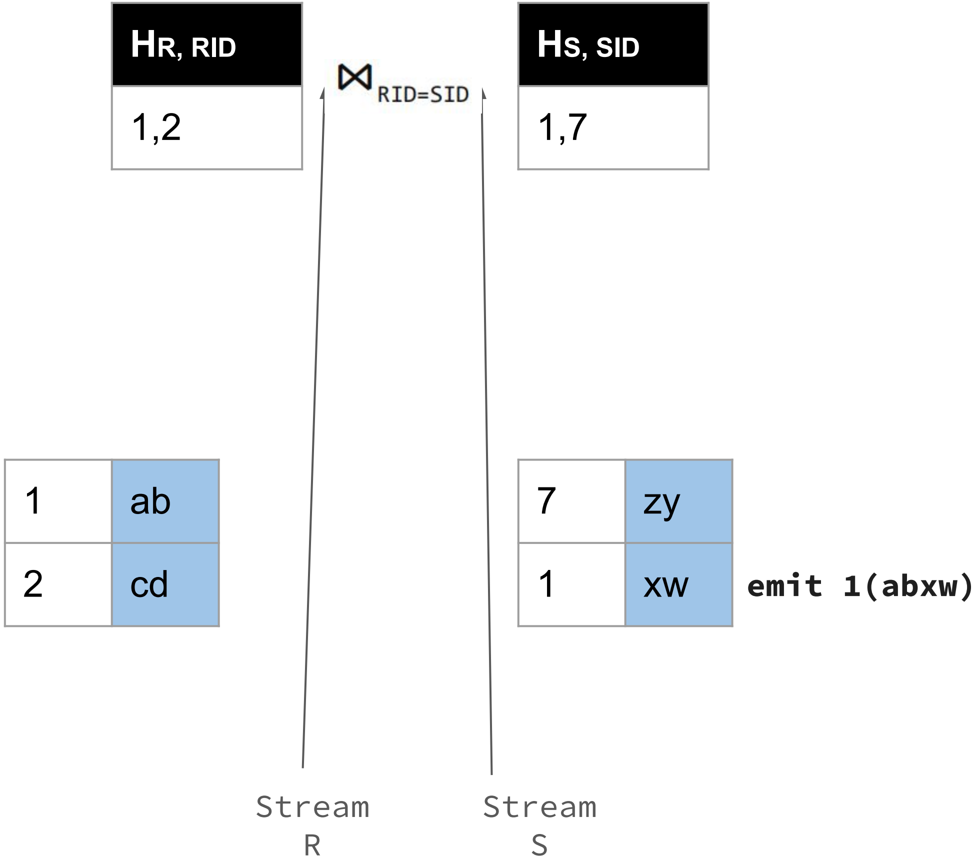
HCC
Institute of HUMAN-CENTRED COMPUTING

Stream Joins II



HCC Institute of HUMAN-CENTRED COMPUTING

Stream Joins II



Stream Joins II

$H_{R, RID}$
1,1,2

$\bowtie_{RID=SID}$

$H_{S, SID}$
1,7

1	ab
2	cd
1	ef

emit 1(efxw)

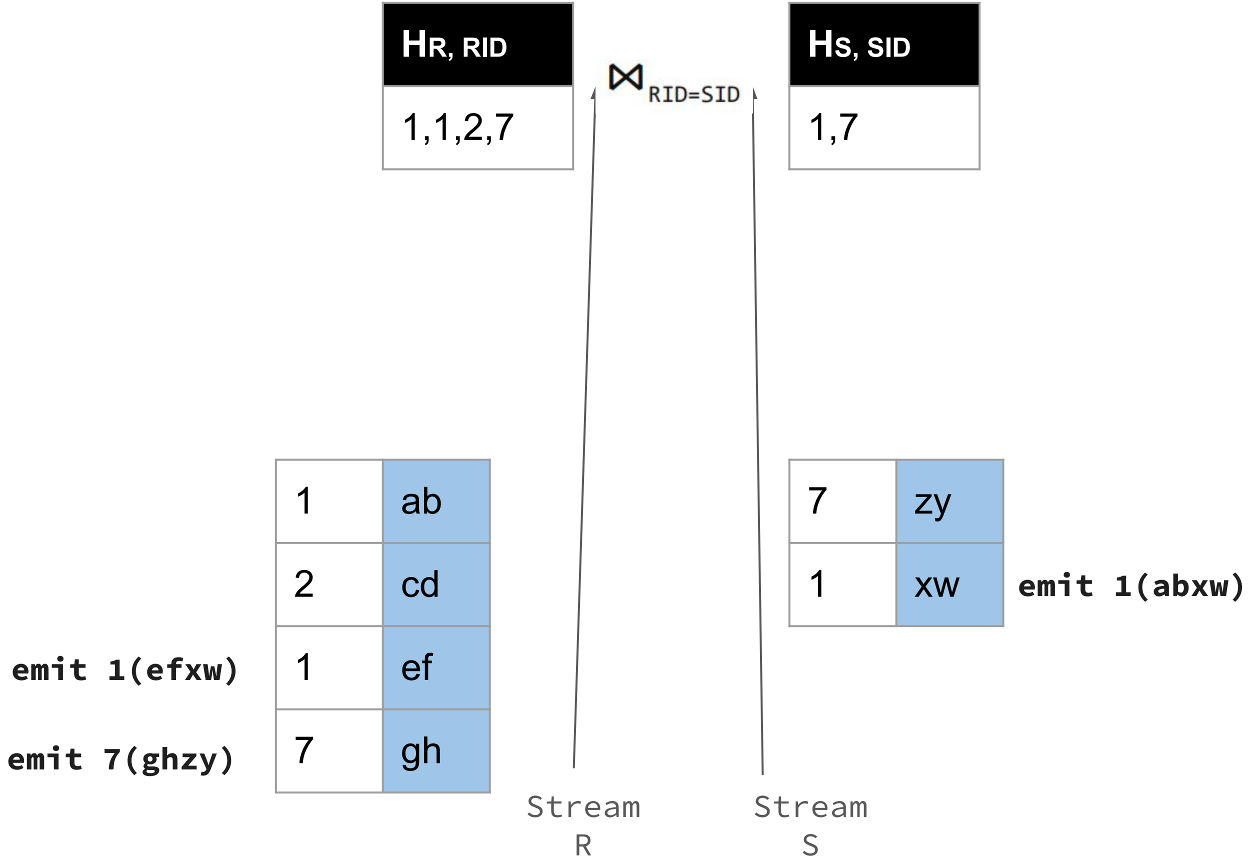
7	zy
1	xw

emit 1(abxw)

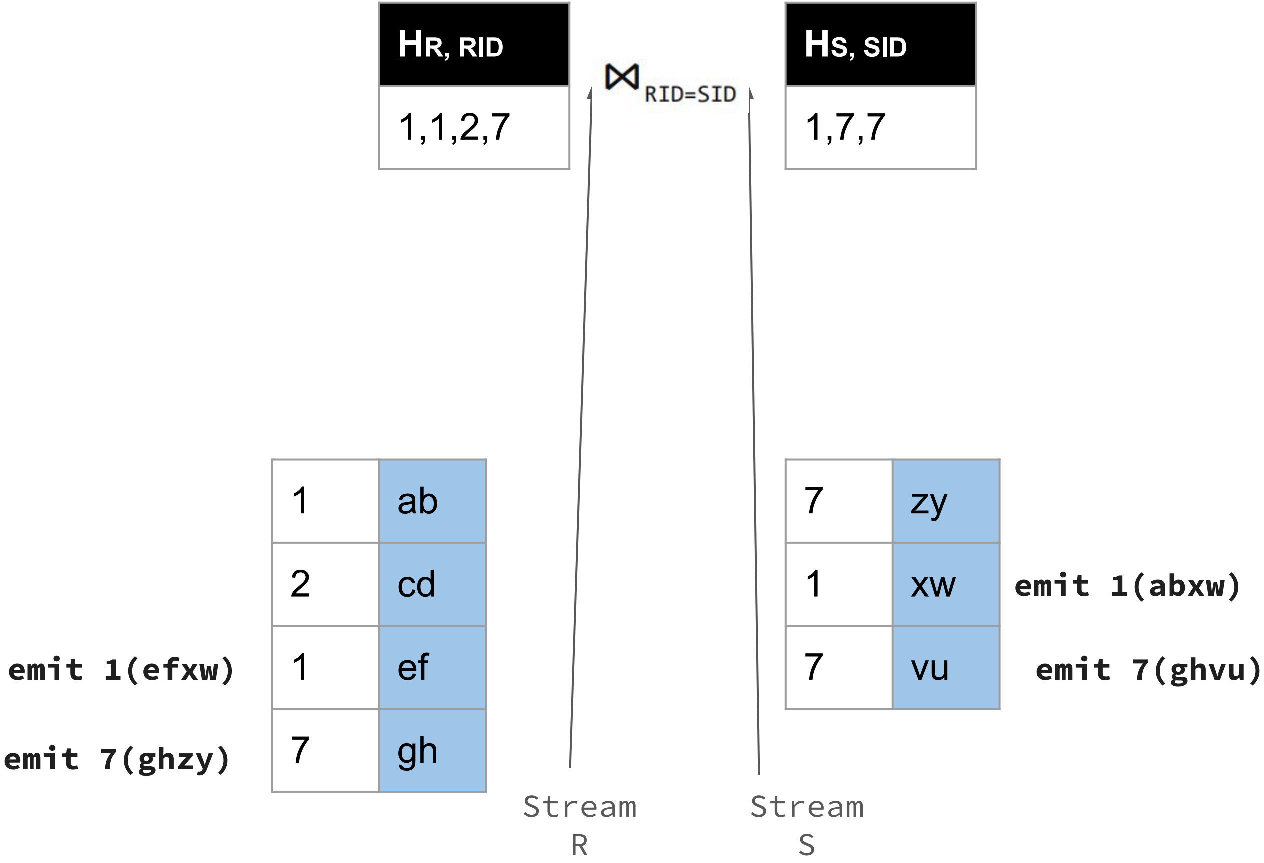
Stream R

Stream S

Stream Joins II



Stream Joins II



Distributed Stream Processing

Query-Aware Stream Partitioning

Example Use Case

- **AT&T** network monitoring
- **112M packets/s** → **26 cycles/tuple** on **3Ghz CPU**
- Complex query sets (apps w/ **~50 queries**) and massive data rates


Query-Aware Partitioning for Monitoring Massive Network Data Streams

Theodore Johnson, AT&T Labs-Research, johnsont@research.att.com
S. Muthukrishnan, Rutgers University, muthu@cs.rutgers.edu
Vladislav Shkapenyuk, AT&T Labs-Research, vshkap@research.att.com
Oliver Spatscheck, AT&T Labs-Research, spatsch@research.att.com

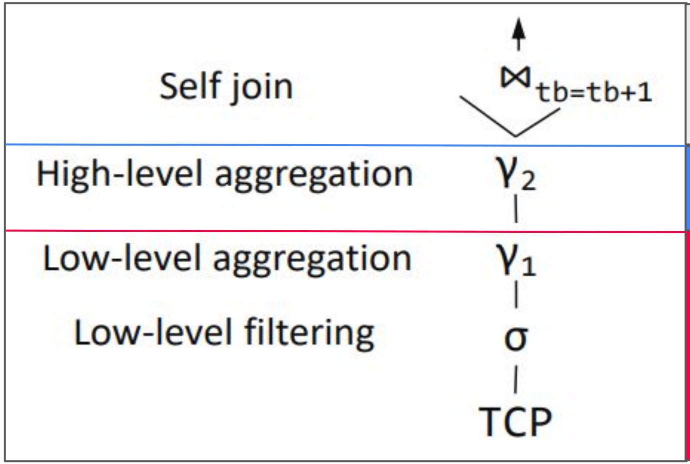
ABSTRACT

T. Johnson et.al, Query-aware partitioning for monitoring massive network data streams. **SIGMOD 2008**

Query-Aware Stream Partitioning

T. Johnson et.al, Query-aware partitioning for monitoring massive network data streams. **SIGMOD 2008**

Baseline Query Execution Plan



Query FLOW PAIRS: users generating heavy-loads during longer periods (no peaks)

Query HEAVY FLOWS: heaviest flows for each IP

Query FLOWS: how many requests are generated per each IP in the network (per minute)

HCC Institute of HUMAN-CENTRED COMPUTING

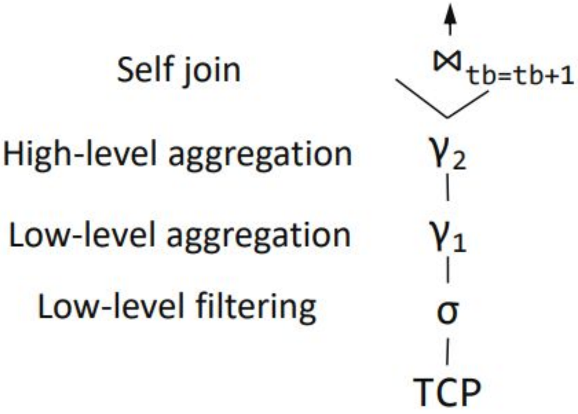
Query-Aware Stream Partitioning

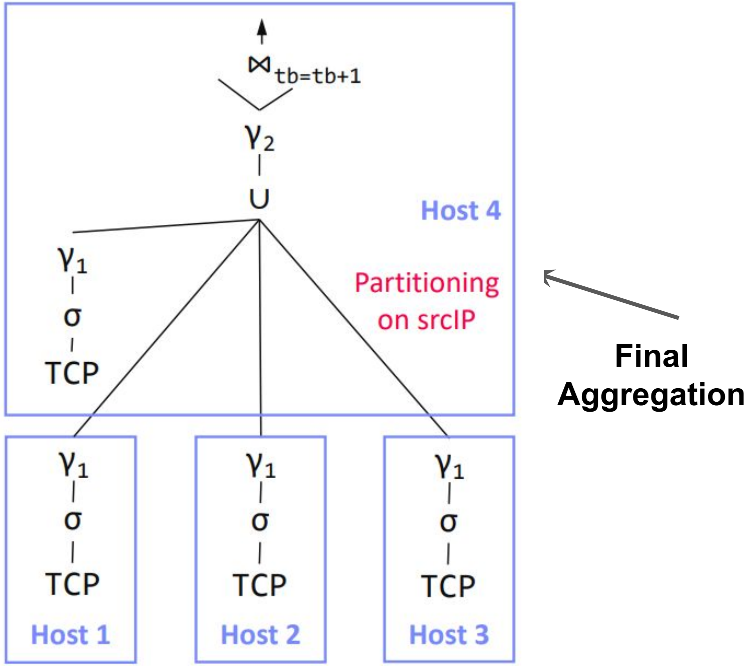
T. Johnson et.al, Query-aware partitioning for monitoring massive network data streams. **SIGMOD 2008**

Solution divide in sub-queries and distribute

Optimized Plan:

- Distributed Plan operators
- Pipeline and task parallelism
- **Not always enough**

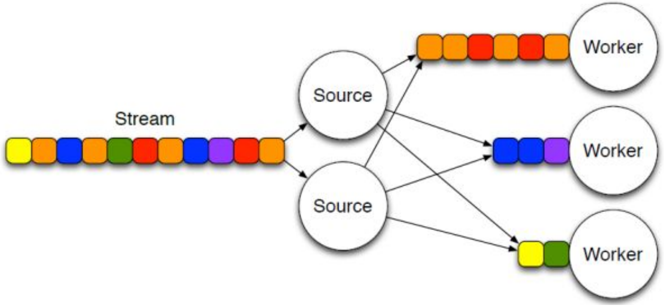




Stream Group Partitioning

Large-Scale Stream Processing

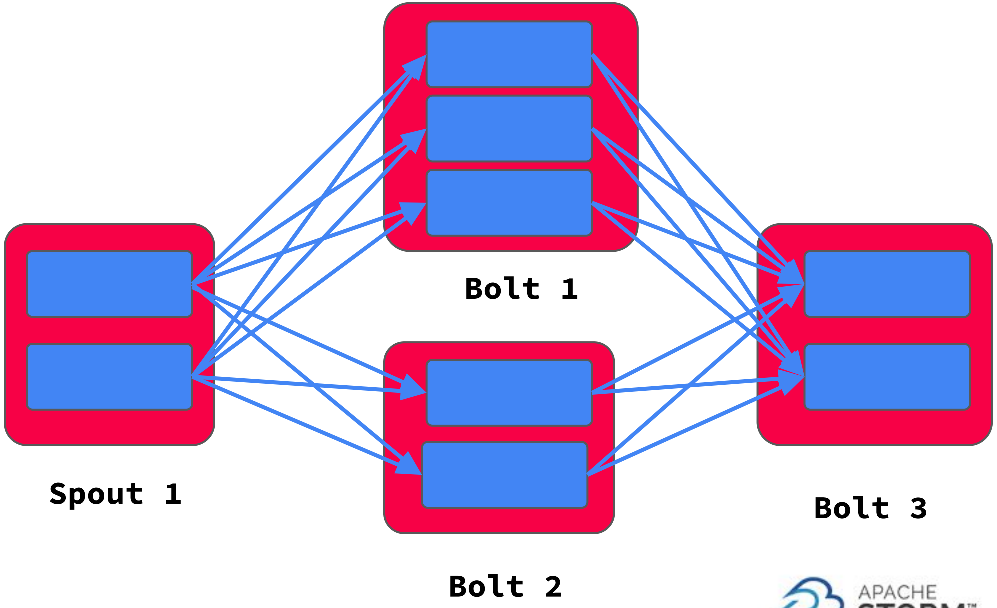
- Limited pipeline parallelism and task parallelism (independent subqueries)
- **Combine with data-parallelism over stream groups**
  - **Shuffle Grouping**
    - Tuples are randomly distributed across consumer tasks
    - Good load balance
  - **Fields Grouping**
    - Tuples partitioned by grouping attributes (keys)
    - Guarantees order within keys, but load imbalance if skew



HCC
Institute of
HUMAN-CENTRED COMPUTING

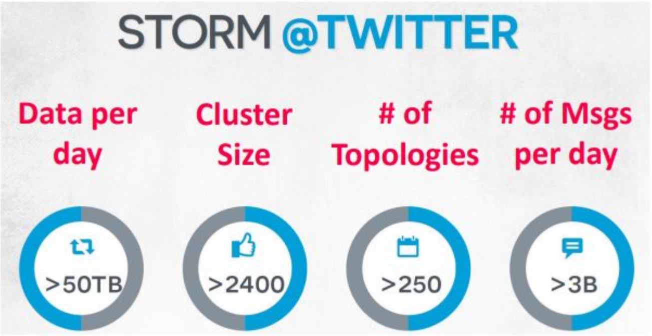
Example Apache Storm

- **Example Topology DAG**
  - **Spouts:** sources of streams
  - **Bolts:** UDF compute ops
  - Tasks mapped to worker processes and executors (threads)




```
Config conf = new Config();
conf.setNumWorkers(3);
topBuilder.setSpout("Spout1", new FooS1(), 2);
topBuilder.setBolt("Bolt1", new FooB1(), 3).shuffleGrouping("Spout1");
topBuilder.setBolt("Bolt2", new FooB2(), 2).shuffleGrouping("Spout1");
topBuilder.setBolt("Bolt3", new FooB3(), 2)
.shuffleGrouping("Bolt1").shuffleGrouping("Bolt2");
StormSubmitter.submitTopology(..., topBuilder.createTopology());
```

Example Twitter Heron



- **Motivation**
  - **Heavy use of Apache Storm** at Twitter
  - Issues: **debugging, performance**, shared **cluster resources**, **back pressure mechanism**
- **Twitter Heron**
  - API-compatible distributed streaming engine
  - **De-facto streaming engine at Twitter** since 2014
- **Dhalion (Heron Extension)**
  - Automatically reconfigure Heron topologies to meet throughput SLO
- **Now back pressure implemented in Apache Storm 2.0 (May 2019)**


Twitter Heron: Stream Processing at Scale

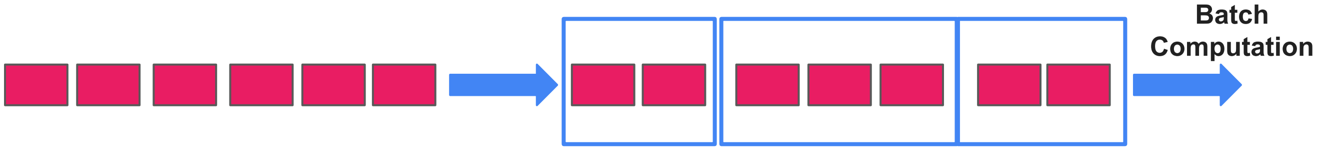
Sanjeev Kulkarni et al:
Twitter Heron: Stream Processing at Scale.
SIGMOD 2015

HCC
Institute of
HUMAN-CENTRED COMPUTING

Discretized Stream (Batch) Computation

- **Motivation**
  - **Fault tolerance** (low overhead, fast recovery)
  - Combination w/ **distributed batch analytics**
- **Discretized Streams (DStream)**
  - Batching of input tuples (100ms – 1s) based on ingest time.
  - Periodically run distributed jobs of stateless, deterministic tasks → **DStreams**
  - State of all tasks materialized as RDDs, recovery via lineage
- **Criticism: High latency, required for batching**

Matei Zaharia et al: Discretized streams: fault-tolerant streaming computation at scale. SOSP 2013



Sequence of Immutable Partitioned Datasets (RDDs)

Unified Batch/Streaming Engines

- **Apache Spark Streaming (Databricks)**
  - Micro-batch computation with exactly-once guarantee
  - Back-pressure and water mark mechanisms
  - Structured streaming via SQL (2.0), continuous streaming (2.3)
- **Apache Flink (Data Artisans, now Alibaba)**
  - Tuple-at-a-time with exactly-once guarantee
  - Back-pressure and water mark mechanisms
  - Batch processing viewed as special case of streaming

Summary and Q&A

- **Summary and Q&A**
  - Data Stream Processing
  - Distributed Stream Processing
- **Next Lectures**
  - Distributed Machine Learning **[Jan 16]**
  - **Written Exam [Jan 30]**

Vielen Dank!