Data Management
13 Stream Processing

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMVIT endowed chair for Data Management

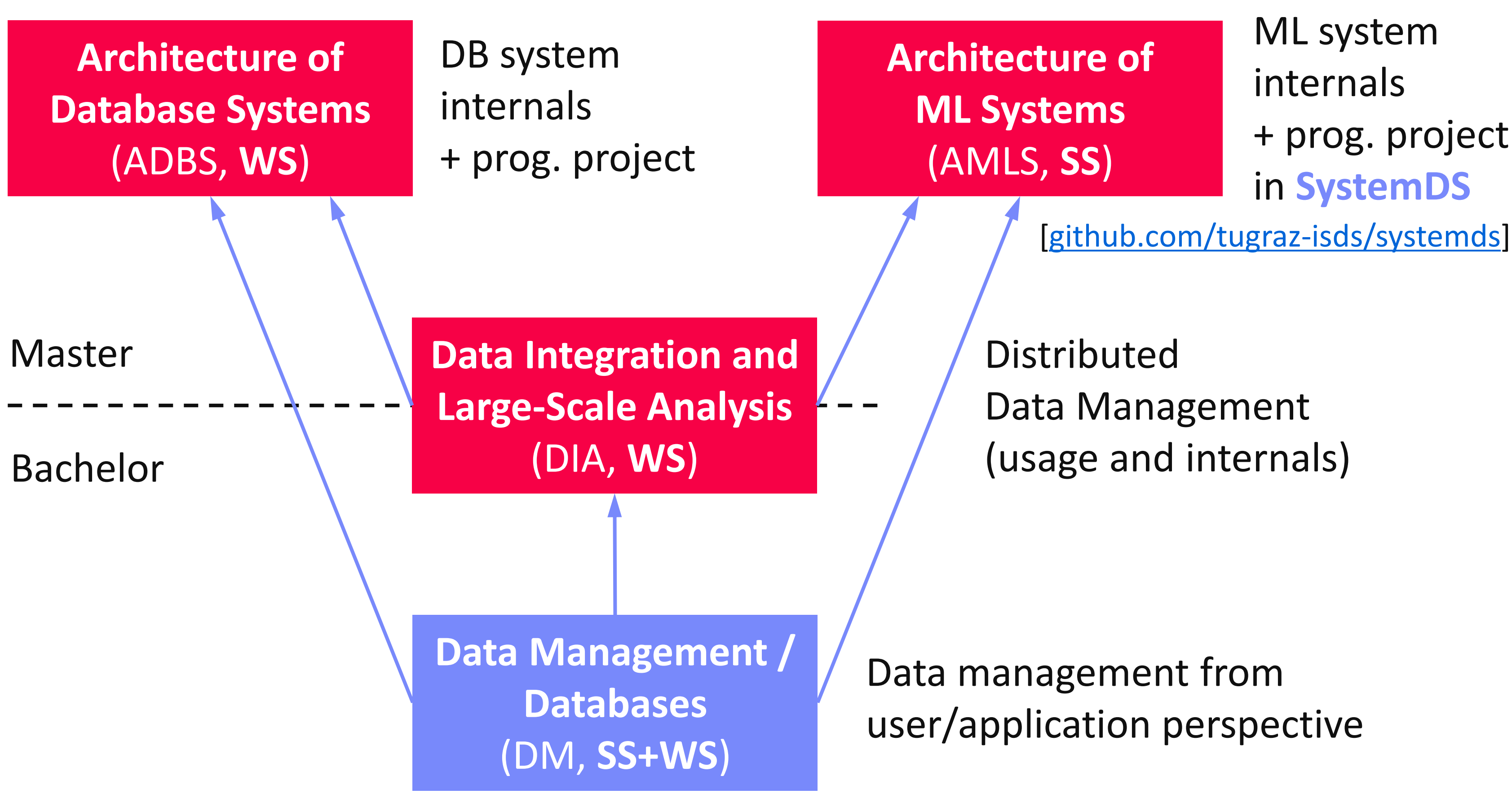
Last update: Jan 20, 2019

SCIENCE
PASSION
TECHNOLOGY

Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
- **#2 Exercises**
  - Exercise 1/2 graded, feedback in TC, office hours
  - **Exercise 3 in progress of being graded**
  - **Exercise 4 due Jan 21, 11.59pm**
- **#3 Course Evaluation**
  - Evaluation period: **Jan 14 – Feb 14**
  - Please, participate w/ honest feedback (pos/neg)
- **#4 Exam**
  - Dates: **Jan 30, 5.30pm**; **Jan 31, 5.30pm**; **Feb 6, 4pm**
  - Registration closes one day before exam
  - **Q&A and Exam Preparation** in today's lecture

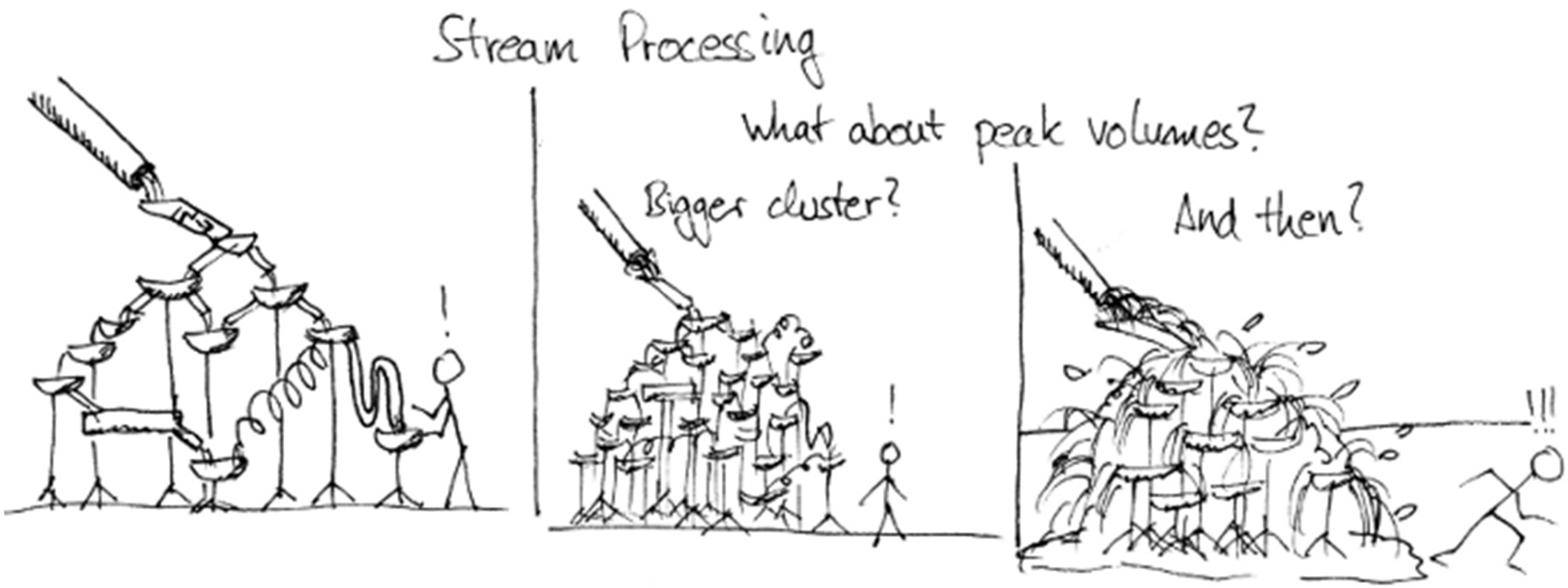
#5 Data Management Courses



Agenda

- **Data Stream Processing**
- ~~**Distributed Stream Processing**~~
- **Q&A and Exam Preparation**

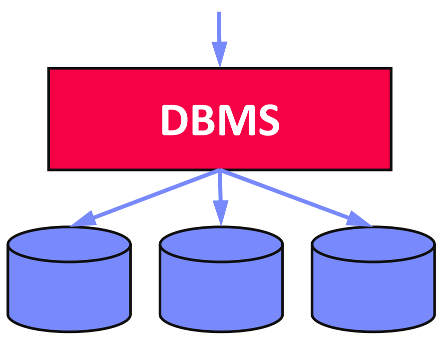
Data Integration and Large-Scale Analysis (DIA)
(bachelor/master)

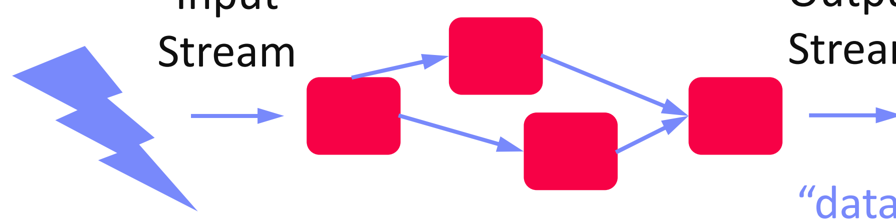


Data Stream Processing

Stream Processing Terminology

- **Ubiquitous Data Streams**
  - Event and message streams (e.g., click stream, twitter, etc)
  - Sensor networks, IoT, and monitoring (traffic, env, networks)
- **Stream Processing Architecture**
  - **Infinite input streams**, often with window semantics
  - Continuous (aka standing) queries





Stream Processing Terminology, cont.

- **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)

Continuously active

- **Data Stream**
  - Unbounded stream of data tuples $S = (s_1, s_2, \ldots)$ with $s_i = (t_i, d_i)$
  - See **08 NoSQL Systems** (time series)
- **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

History of Stream Processing Systems

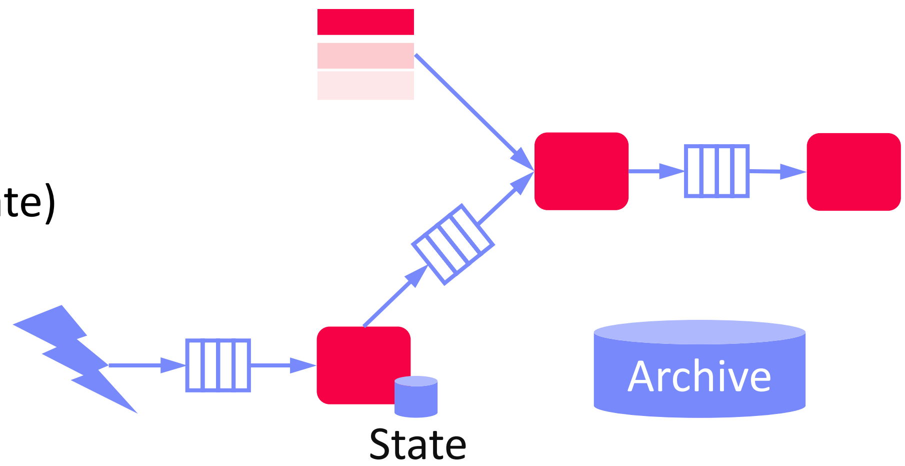
- **2000s**
  - **Data stream management systems** (DSMS, mostly academic prototypes): **STREAM** (Stanford'01), **Aurora** (Brown/MIT/Brandeis'02) → **Borealis** ('05), **NiagaraCQ** (Wisconsin), **TelegraphCQ** (Berkeley'03), and many others

    → but mostly unsuccessful in industry/practice
  - **Message-oriented middleware** and **Enterprise Application Integration** (EAI): IBM **Message Broker**, SAP **eXchange Infra.**, MS **Biztalk Server**, **TransConnect**

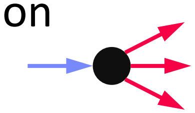
- **2010s**
  - **Distributed stream processing engines**, and "unified" batch/stream processing
  - **Proprietary systems:** Google Cloud Dataflow, MS StreamInsight / Azure Stream Analytics, IBM InfoSphere Streams / Streaming Analytics, AWS Kinesis
  - **Open-source systems:** **Apache Spark Streaming** (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



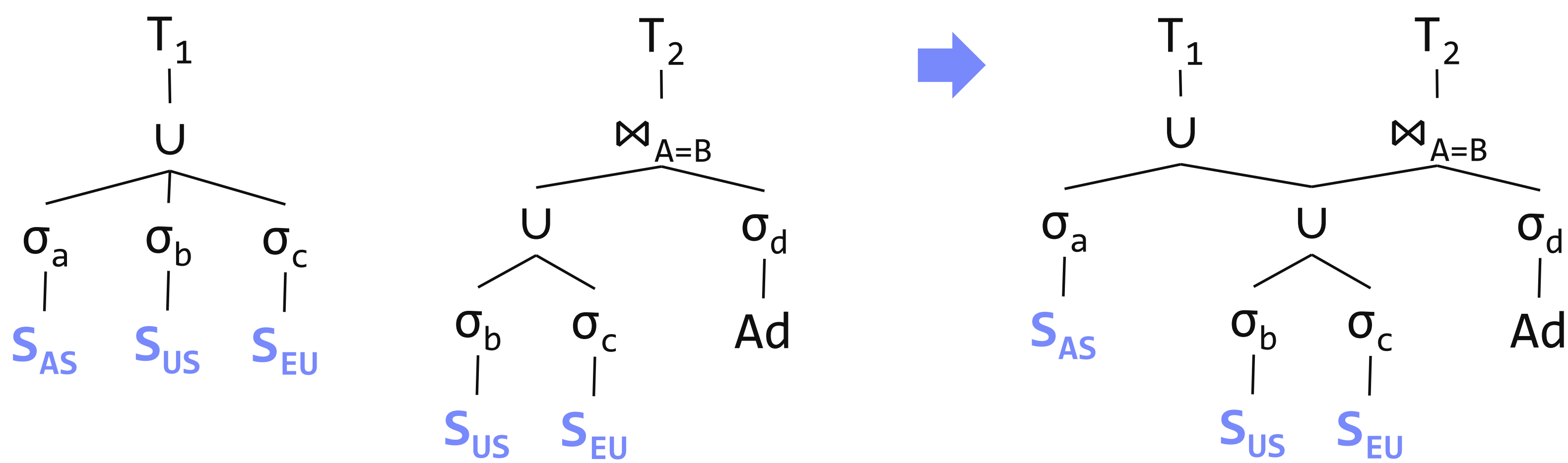
- **Operator Model**
  - Read from input queue
  - Write to potentially many output queues
  - Example Selection $\sigma_{A=7}$



```
while( !stopped ) {
   r = in.dequeue(); // blocking
   if( pred(r.A) )   // A==7
      for( Queue o : out )
         o.enqueue(r); // blocking
}
```

System Architecture – Sharing

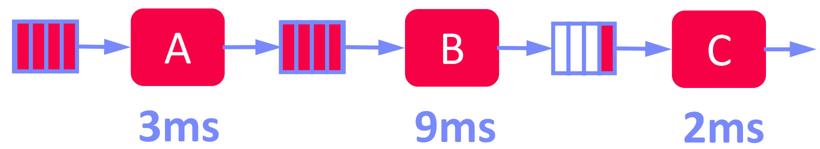
- **Multi-Query Optimization**
  - Given **set of continuous queries** (deployed), compile minimal DAG w/o redundancy (see **08 Physical Design MV**) → **subexpression elimination**



- **Operator and Queue Sharing**
  - **Operator sharing:** complex ops w/ multiple predicates for adaptive reordering
  - **Queue sharing:** avoid duplicates in output queues via masks

System Architecture – Handling Overload

- **#1 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

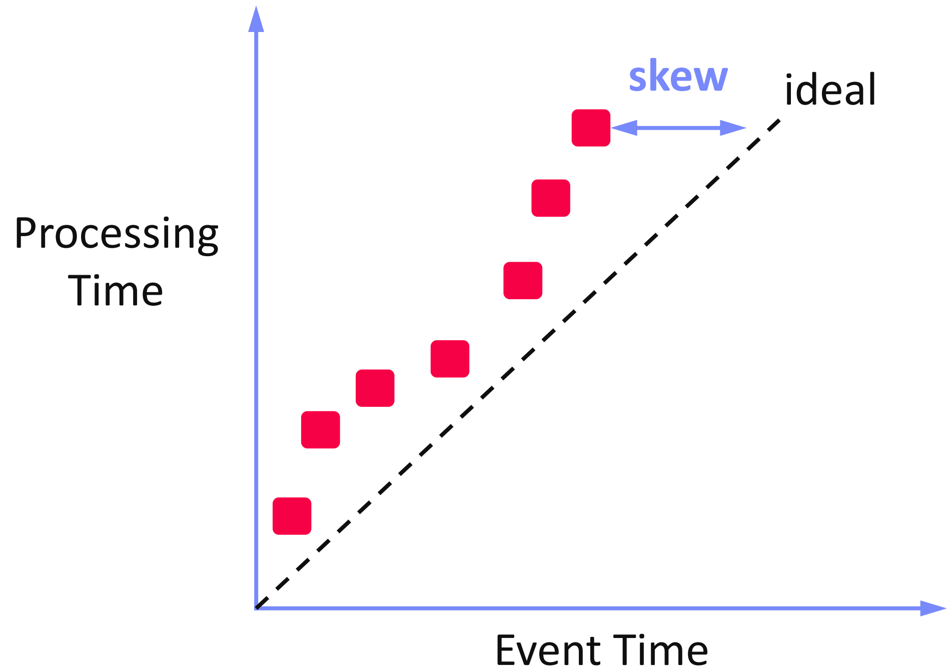
- **#2 Load Shedding**
  - #1 **Random-sampling**-based load shedding
  - #2 **Relevance-based** load shedding
  - #3 **Summary-based** load shedding (synopses)
  - Given SLA, select queries and shedding placement that minimize error and satisfy constraints

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

- **#3 Distributed Stream Processing** (see course DIA)
  - 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 Time**
  - 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., **water marks = delay threshold**)
  - Use of more complex triggers (**speculative and late results**)

Durability and Consistency Guarantees

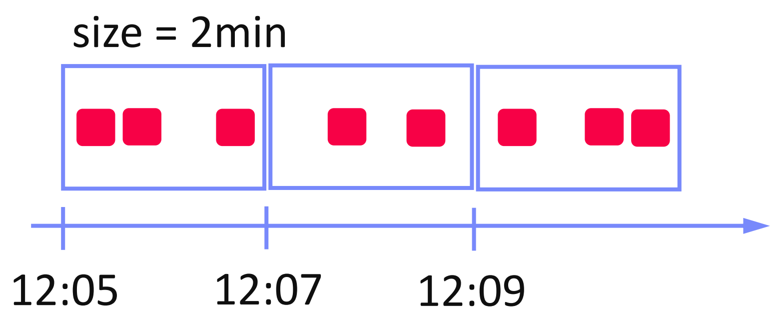
- **#1 At Most Once**
  - **“Send and forget”**, ensure data is never counted twice
  - Might cause data loss on failures

- **#2 At Least Once**
  - **“Store and forward”** or acknowledgements from receiver, replay stream from a checkpoint on failures
  - Might create incorrect state (processed multiple times)

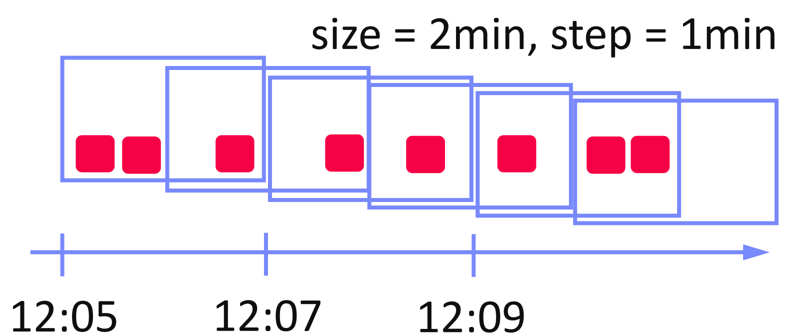
- **#3 Exactly Once**
  - **“Store and forward” w/ guarantees** regarding state updates and sent msgs
  - Often via dedicated transaction mechanisms

Window Semantics

- **Windowing Approach**
  - Many operations like joins/aggregation **undefined over unbounded streams**
  - Compute operations over **windows of time or elements**
- **#1 Tumbling Window**
  - Every data item is only part of a single window
  - Aka Jumping window



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

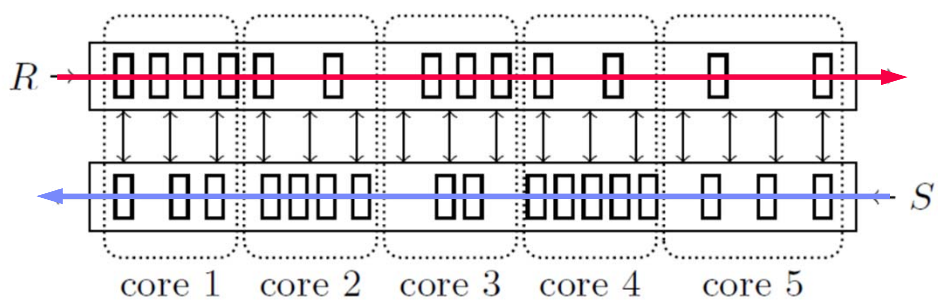


Stream Joins

- **Basic Stream Join**
  - **Tumbling window:** use classic join methods
  - **Sliding window** (symmetric for both R and S)
    - Applies to arbitrary join pred
    - See 08 Query Processing (NLJ)

```
For each new r in R:
   1. Scan window of stream S
      to find match tuples
   2. Insert new r into window
      of stream R
   3. Invalidate expired tuples
      in window of stream R
```

- **Excursus: How Soccer Players Would do Stream Joins**
  - Handshake-join w/ 2-phase forwarding

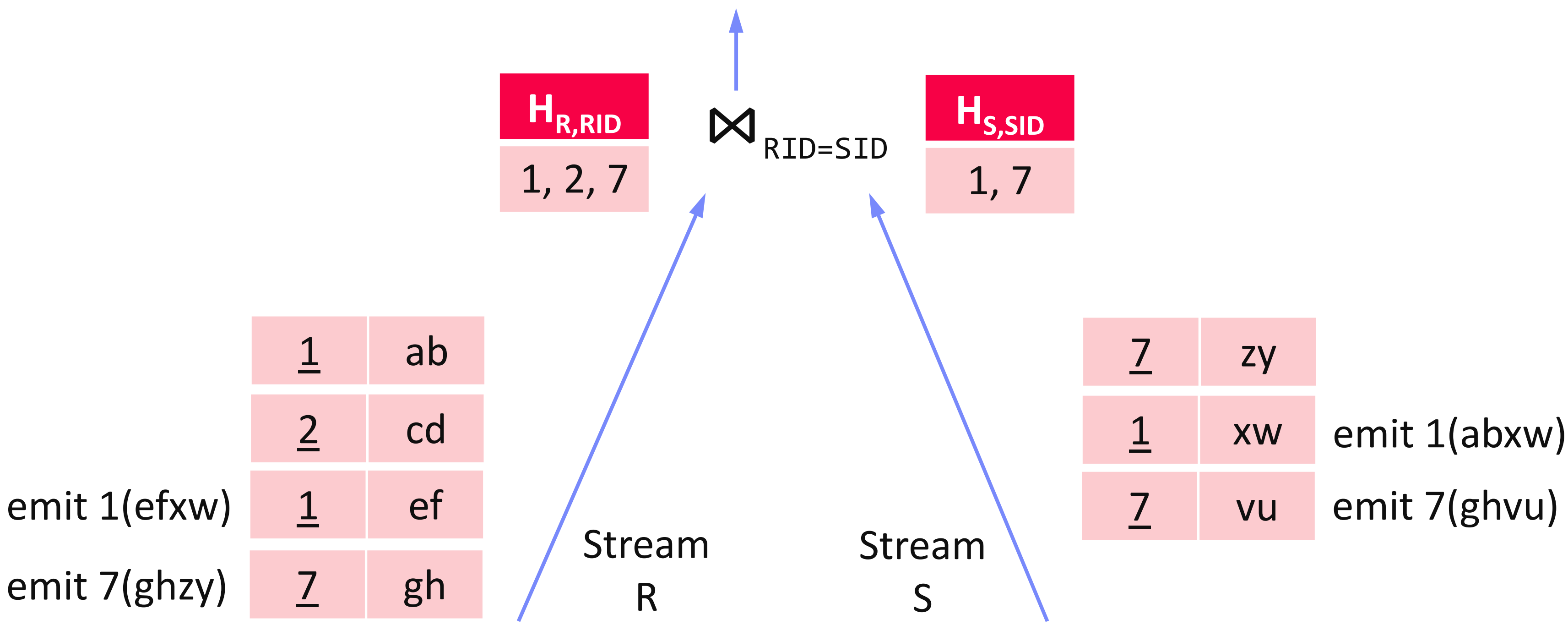


[Jens Teubner, René Müller: How soccer players would do stream joins. **SIGMOD 2011**]

Stream Joins, cont.

[Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Y. Levy, Daniel S. Weld: An Adaptive Query Execution System for Data Integration. **SIGMOD 1999**]

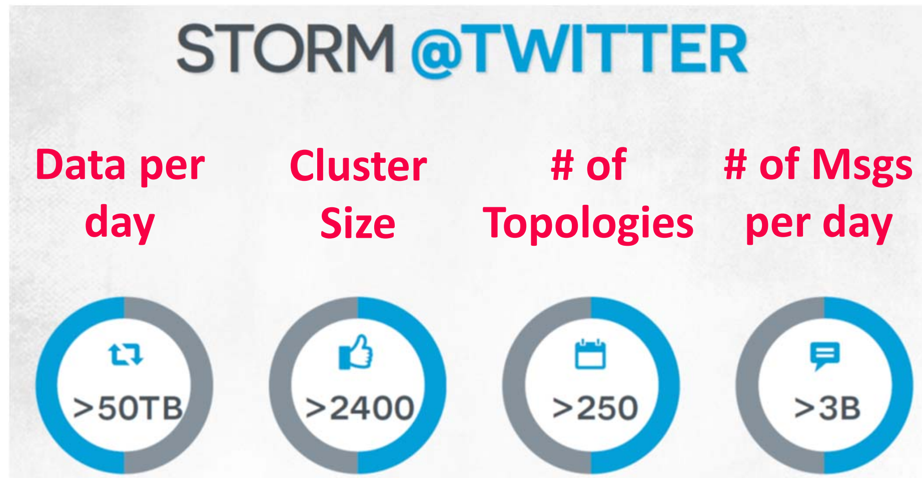
- **Double-Pipelined Hash Join**
  - Join of bounded streams (or unbounded w/ invalidation)
  - **Equi join predicate**, **symmetric and non-blocking**
  - For every incoming tuple (e.g. left): probe (right)+emit, and build (left)



Excursus: Example Twitter Heron

[**Credit:** Karthik Ramasamy]

- **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

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

- **Dhalion (Heron Extension)**
  - Automatically reconfigure Heron topologies to meet throughput SLO

[Avrilia Floratou et al: Dhalion: Self-Regulating Stream Processing in Heron. **PVLDB 2017**]

- **Now back pressure implemented in Apache Storm 2.0 (May 2019)**