TU Graz
SCIENCE
PASSION
TECHNOLOGY

Database Systems
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: Jun 03, 2019

ISDS

Announcements/Org

- **#1 Video Recording**
  - Since lecture 03, video/audio recording
  - Link in TeachCenter & TUbe

- **#2 Exercises**
  - Exercise 1 graded, feedback in TC, office hours
  - **Exercise 2 in progress of being graded**
  - **Exercise 3 due Jun 04, 11.59pm**

77.4%

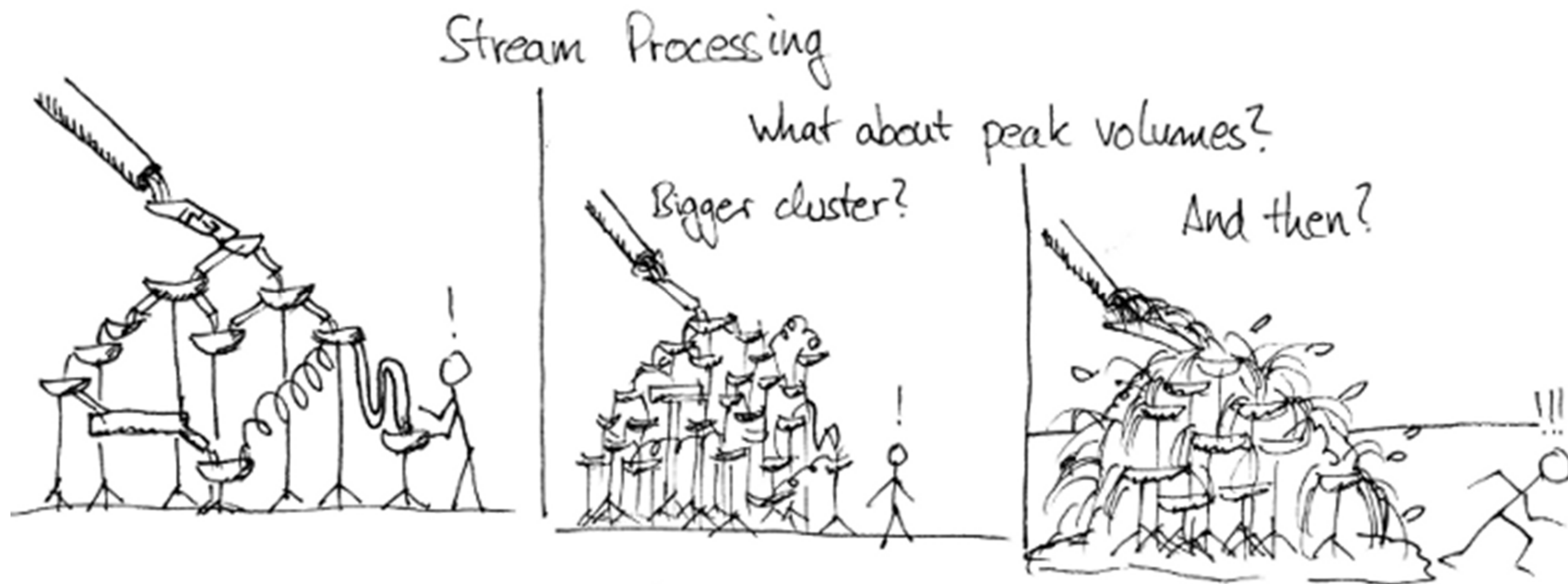
60.4%

- **#3 Course Evaluation**
  - Evaluation period: **Jun 18 – Aug 13**
  - Please, participate w/ honest feedback (pos/neg)
- **#4 Open Positions**
  - **ExDRa: Exploratory Data Science over Raw Data**
  - 2x PhDs / **student assistants** → m.boehm@tugraz.at

Agenda

- **Data Stream Processing**
- **Distributed Stream Processing**
- **Exercise 4: Large-Scale Data Analysis**

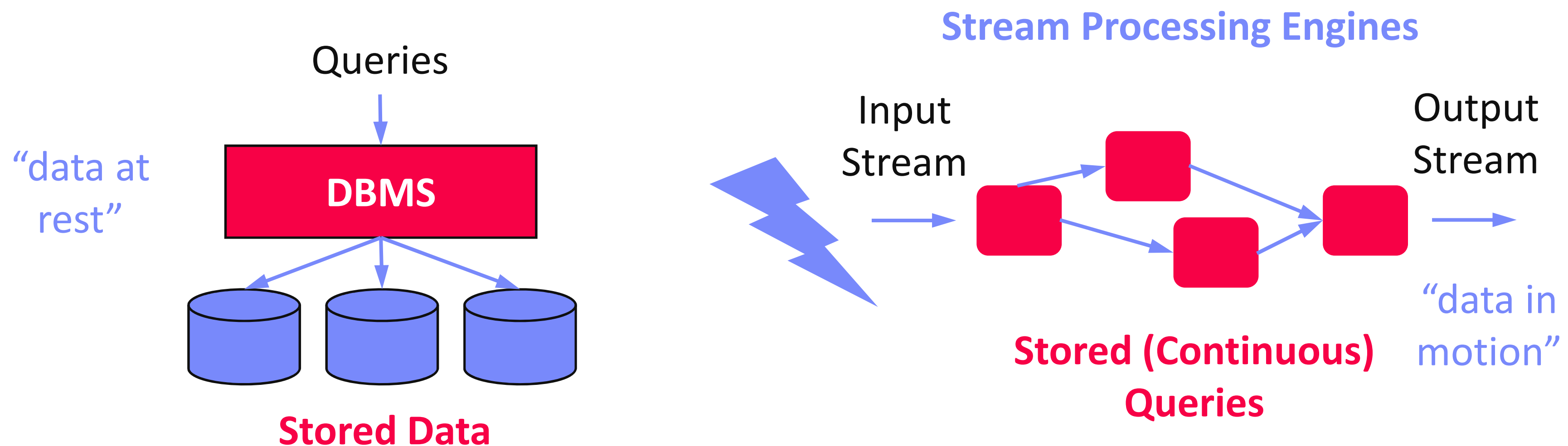
Data Integration and Large-Scale Analysis (DIA)
(bachelor/master)



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

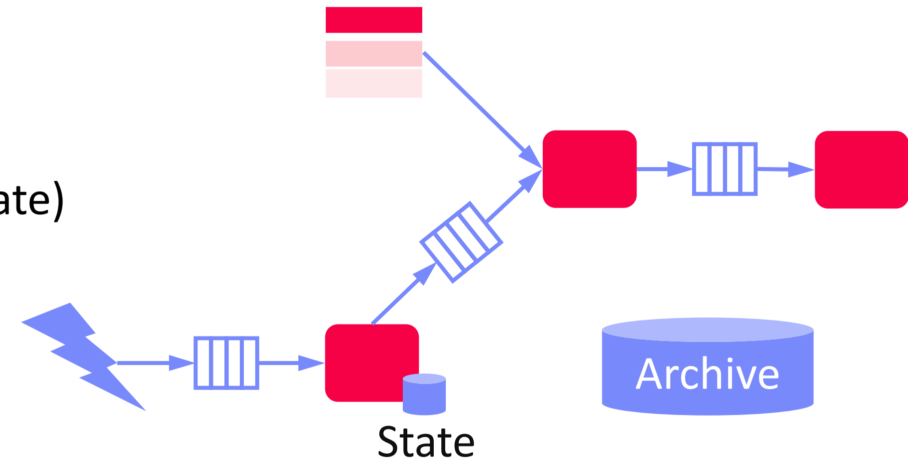
History of Stream Processing Systems

- **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

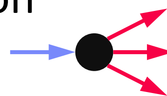
    → but mostly unsuccessful in industry/practice
  - **Message-oriented middleware** and **Enterprise Application Integration** (EAI): IBM **Message Broker**, SAP **eXchange Infra.**, MS **Biztalk Server**, **TransConnect**
- **2010s**
  - **Distributed stream processing engines**, and "unified" batch/stream 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



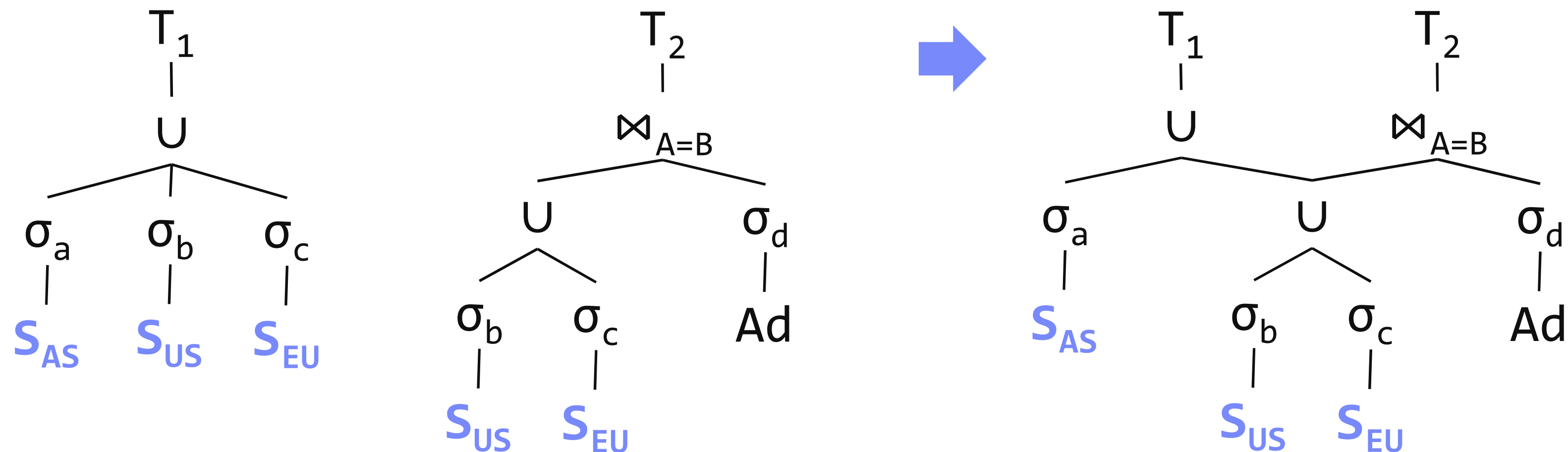
- **Operator Model**
  - Read from input queue
  - Write to potentially many output queues
  - Example Selection $\sigma_{A=7}$



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

System Architecture – Sharing

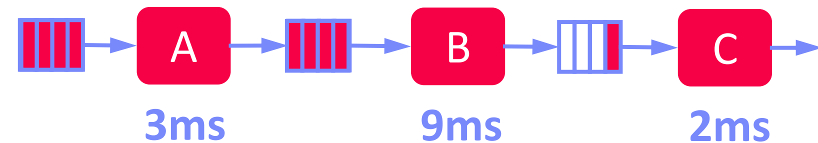
- **Multi-Query Optimization**
  - Given **set of continuous queries** (deployed), compile minimal DAG w/o redundancy (see **08 Physical Design MV**) ➔ **subexpression elimination**



- **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

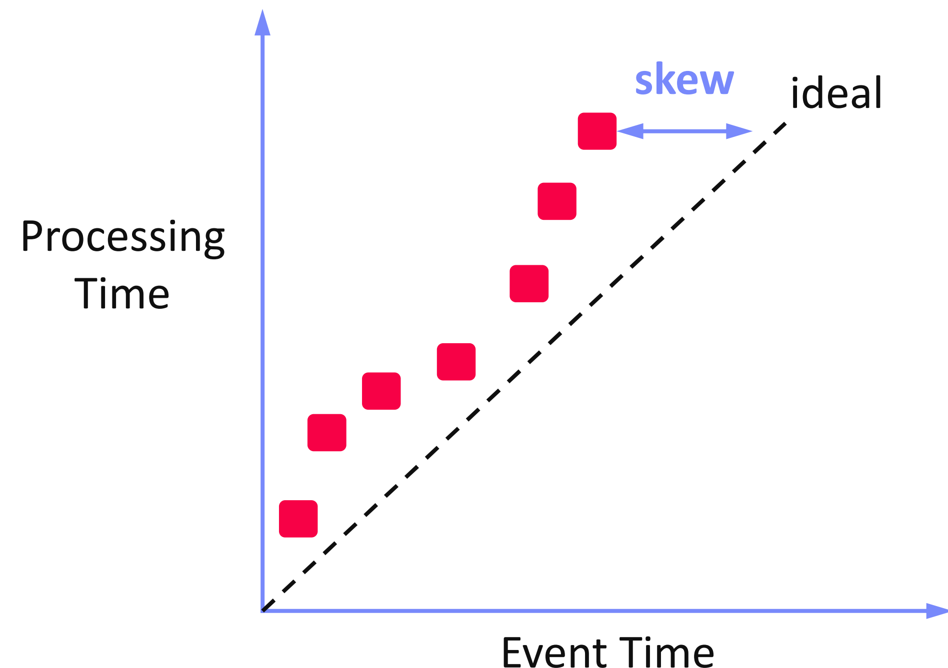
- **#2 Load Shedding**
  - #1 **Random-sampling**-based load shedding
  - #2 **Relevance-based** load shedding
  - #3 **Summary-based** load shedding (synopses)
  - Given SLA, select queries and shedding placement that minimize error and satisfy constraints

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

- **#3 Distributed Stream Processing** (part of today's lecture)
  - 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

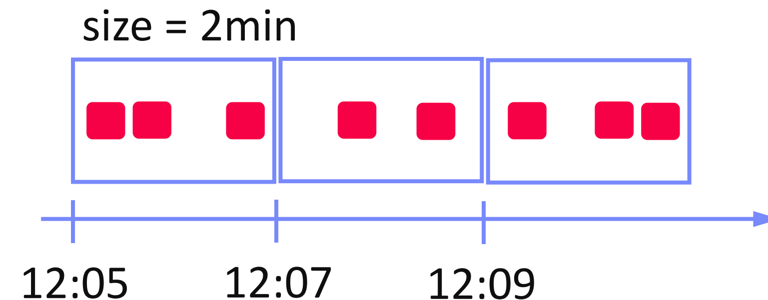
- **#2 At Least Once**
  - **"Store and forward"** or acknowledgements from receiver, replay stream from a checkpoint on failures
  - Might create incorrect state (processed multiple times)

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

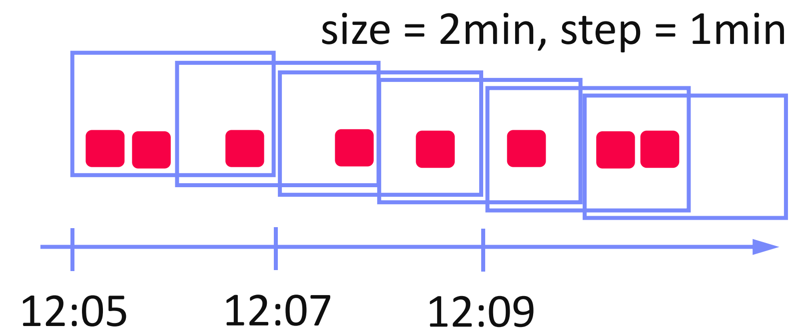
Window Semantics

- **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

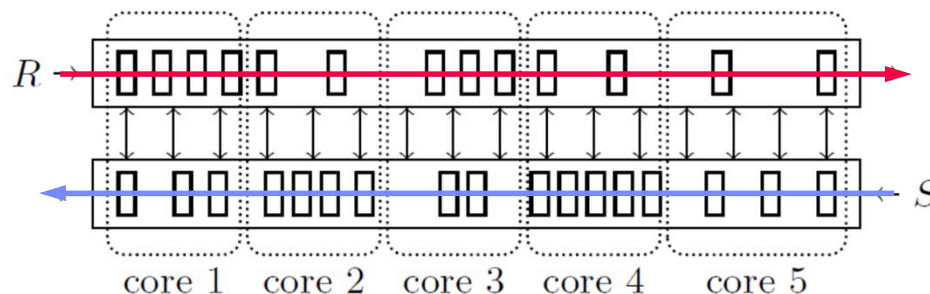


Stream Joins

- **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

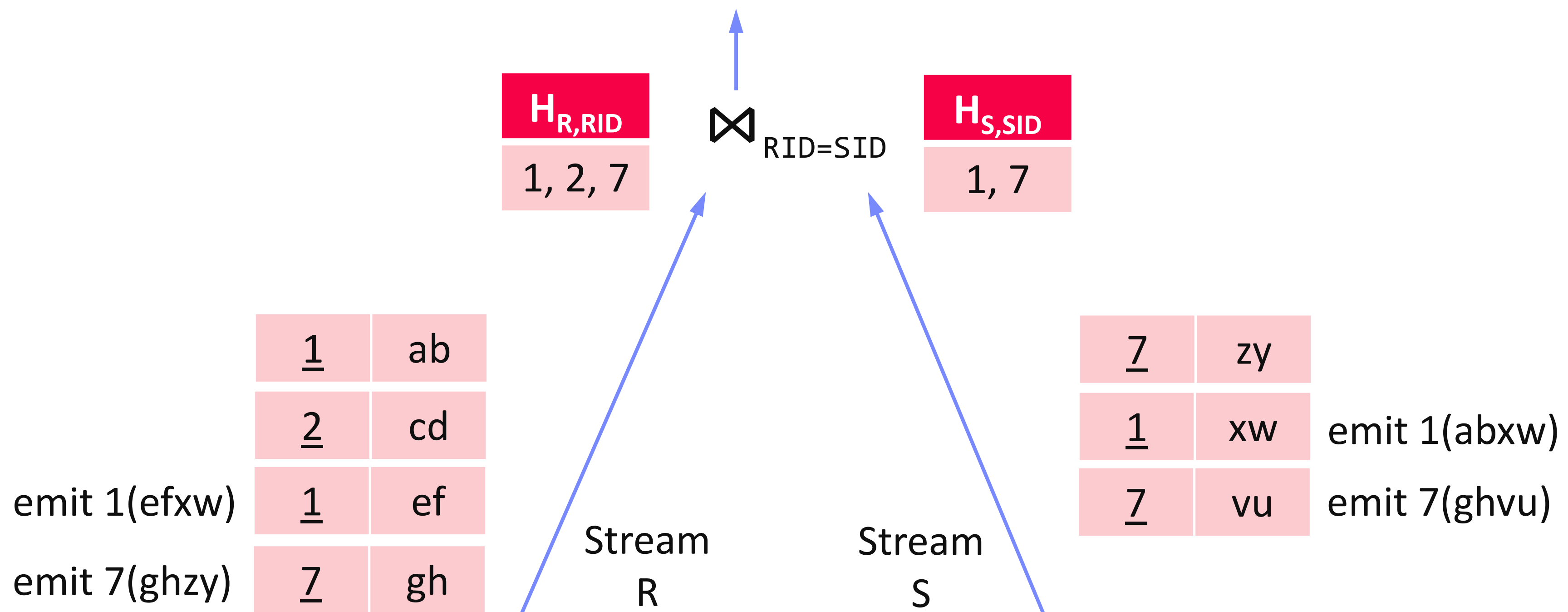


[Jens Teubner, René Müller: How soccer players would do stream joins. **SIGMOD 2011**]

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)



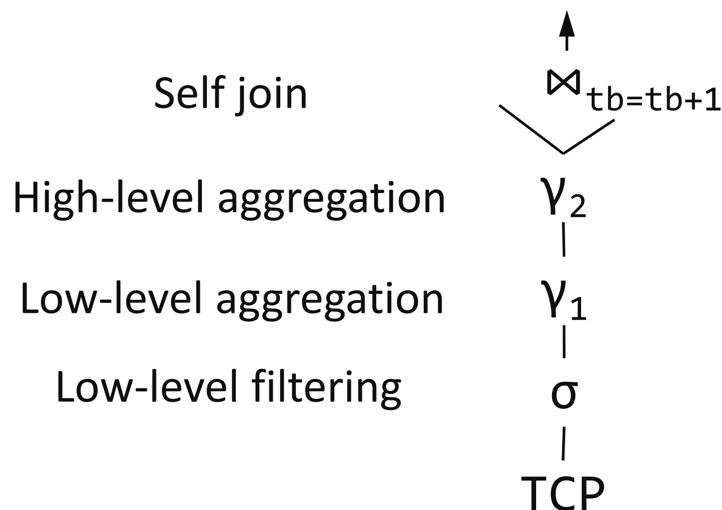
Distributed Stream Processing

Query-Aware Stream Partitioning

- **Example Use Case**
  - AT&T network monitoring with Gigascope (e.g., OC768 network)
  - 2x40 Gbit/s traffic → 112M packets/s → **26 cycles/tuple** on 3Ghz CPU
  - Complex query sets (apps w/ **~50 queries**) and massive data rates

[Theodore Johnson, S. Muthu Muthukrishnan, Vladislav Shkapenyuk, Oliver Spatscheck: Query-aware partitioning for monitoring massive network data streams. **SIGMOD 2008**]

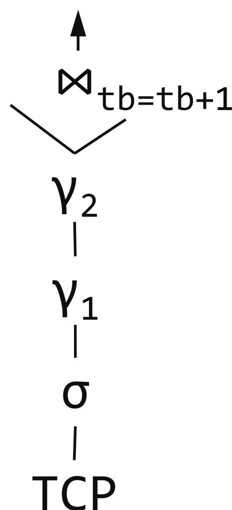
- **Baseline Query Execution Plan**

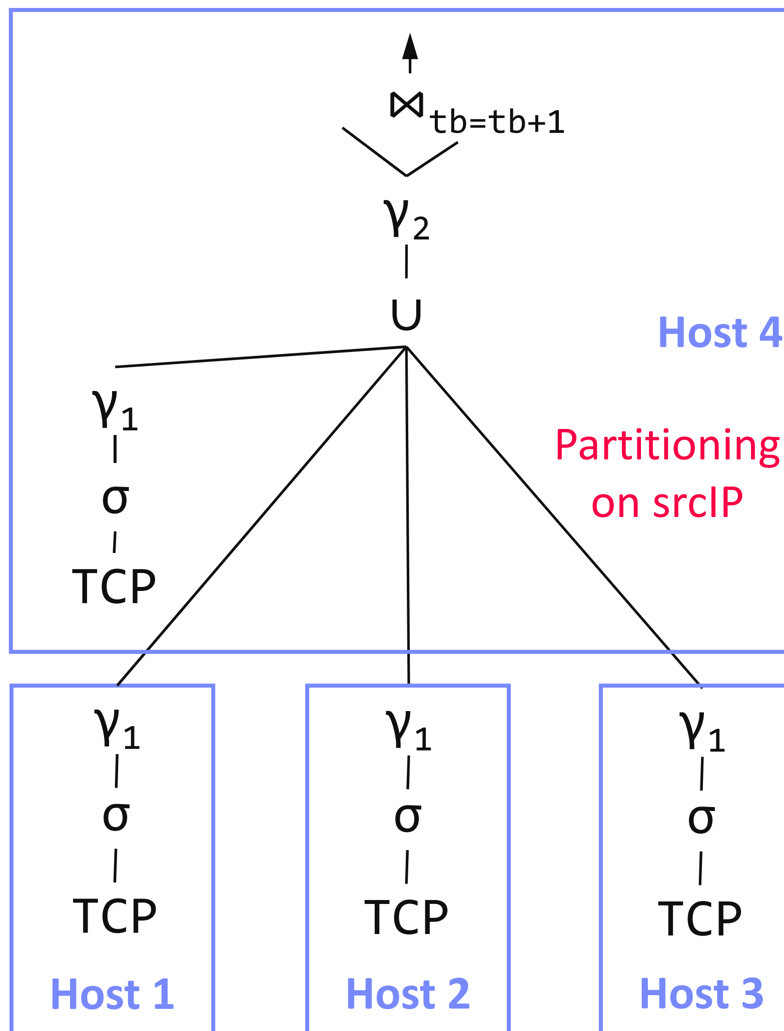



```
Query flow_pairs:
SELECT S1.tb, S1.srcIP, S1.max, S2.max
   FROM heavy_flows S1, heavy_flows S2
   WHERE S1.srcIP = S2.srcIP
     and S1.tb = S2.tb+1

Query heavy_flows:
SELECT tb,srcIP,max(cnt) as max_cnt
   FROM flows
   GROUP BY tb, srcIP

Query flows:
SELECT tb, srcIP, destIP, COUNT(*) AS cnt
   FROM TCP WHERE ...
   GROUP BY time/60 AS tb,srcIP,destIP
```

Query-Aware Stream Partitioning, cont.

- **Optimized Query Execution Plan**
  - Distributed plan operators
  - Pipeline and task parallelism

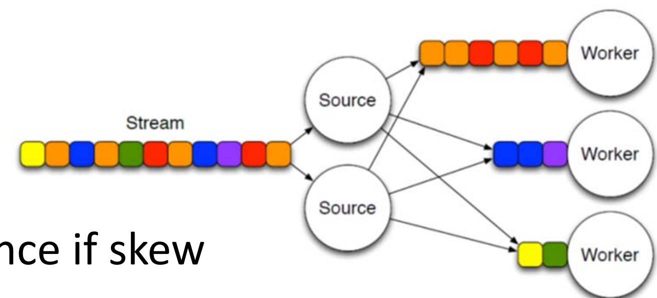




Stream Group Partitioning

- **Large-Scale Stream Processing**
  - Limited pipeline parallelism and task parallelism (independent subqueries)
  - Combine with **data-parallelism over stream groups**
- **#1 Shuffle Grouping**
  - Tuples are randomly distributed across consumer tasks
  - Good load balance
- **#2 Fields Grouping**
  - Tuples partitioned by grouping attributes
  - Guarantees order within keys, but load imbalance if skew
- **#3 Partial Key Grouping**
  - Apply **"power of two choices"** to streaming
  - **Key splitting:** select among 2 candidates per key (works for all associative aggregation functions)
- **#4 Others: Global, None, Direct, Local**

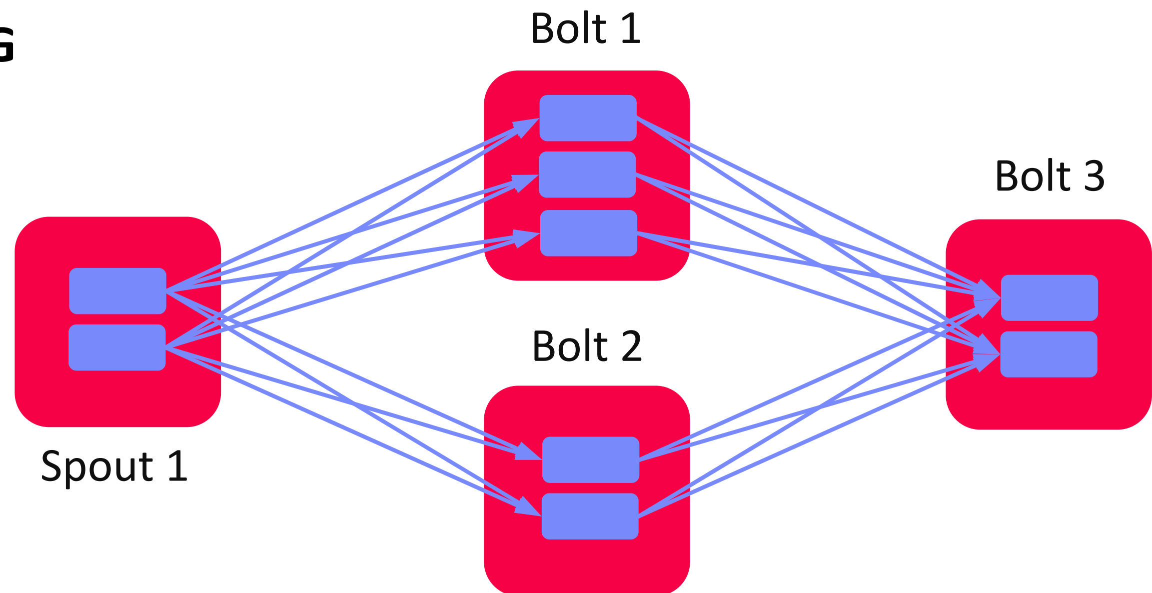
APACHE STORM™



[Md Anis Uddin Nasir et al: The power of both choices: Practical load balancing for distributed stream processing engines. **ICDE 2015**]

Example Apache Storm

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

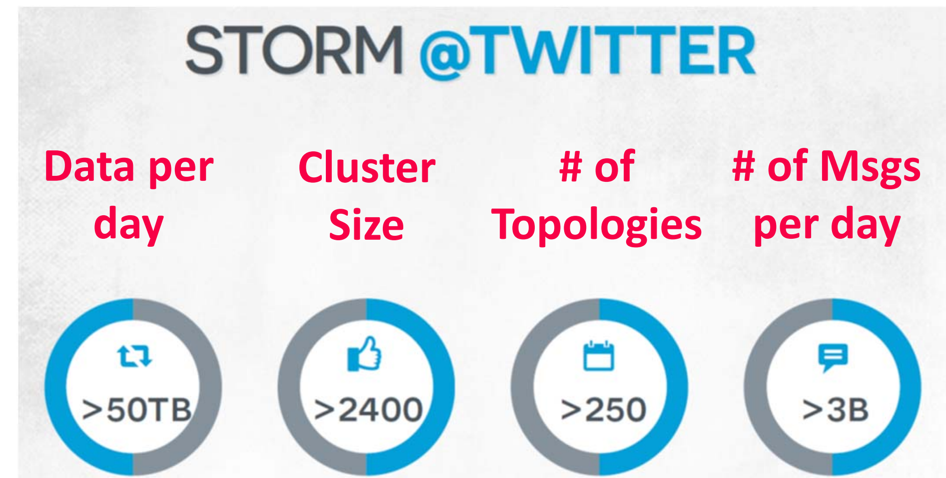



```
Config conf = new Config();
conf.setNumWorkers(3);

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

[**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**]

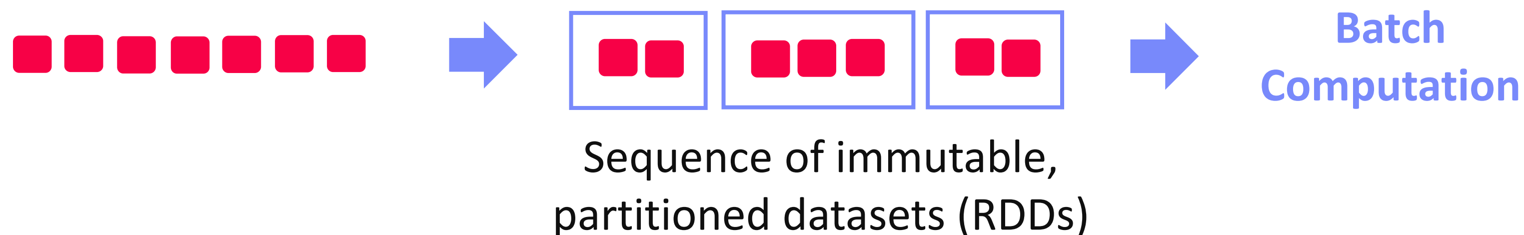
- **Now back pressure implemented in Apache Storm 2.0 (May 2019)**

Discretized Stream (Batch) Computation

- **Motivation**
  - **Fault tolerance** (low overhead, fast recovery)
  - Combination w/ **distributed batch analytics**

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

- **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**

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

[https://flink.apache.org/news/2019/02/13/unified-batch-streaming-blink.html]

- **Google Cloud Dataflow**
  - **Tuple-at-a-time** with exactly-once guarantee
  - MR → FlumeJava → MillWheel → Dataflow
  - Google's fully managed batch and stream service

[T. Akidau et al.: The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. **PVLDB 2015**]

→ **Apache Beam (API+SDK from Dataflow)**
  - **Abstraction for Spark, Flink, Dataflow** w/ common API, etc
  - Individual runners for the different runtime frameworks

Exercise 4:
Large-Scale Data Analysis

Published: Jun 03

Deadline: Jun 25

Task 4.1 Apache Spark Setup

4/25 points

- **#1 Pick your Spark language binding**
  - Java, Scala, Python
- **#2 Install Dependencies**
  - Java: Maven
    spark-core, spark-sql
  - Python:
    pip install pyspark

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.11</artifactId>
  <version>2.4.3</version>
 </dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.4.3</version>
</dependency>
```

- **(#3 Win Environment)**
  - Download https://github.com/steveloughran/winutils/tree/master/hadoop-2.7.1/bin/winutils.exe
  - Create environment variable HADOOP_HOME="<some-path>/hadoop"

Task 4.2 SQL Query Processing

5/25 points

- **Q11: Clubs of German Players**
  - Distinct clubs of players from team Germany 2014
  - Return (Club Name, Number of Players)
  - Sorted in descending order of the number of players

- **Q12: Length of World Cups**
  - World cup tournament lengths (difference first and last match)
  - Return (TYear, Host Name, Length)
  - Sorted by (Length, Year) in ascending order
  - Tournaments with multiple hosts → multiple tuples

Task 4.2 SQL Query Processing, cont.

- **Expected Results**

Q11: Clubs of German Players in World Cup 2014

	name character varying (256)	count bigint
1	Bayern Munich	7
2	Borussia Dortmund	5
3	Arsenal	3
4	Schalke 04	2
5	Chelsea	1
6	Hannover 96	1
7	Lazio	1
8	Real Madrid	1
9	SC Freiburg	1
10	Borussia Mönchengladbach	1

Q12: Length of World Cups

	tyear smallint	name character varying (256)	length integer
1	1954	Switzerland	18
2	1962	Chile	18
3	1966	England	19
4	1958	Sweden	21
5	1970	Mexico	21
6	1974	West Germany	24
7	1978	Argentina	24
8	1982	Spain	28
9	1986	Mexico	29
10	1990	Italy	30
11	1994	United States	30
12	2002	South Korea	30
13	2002	Japan	30
14	2006	Germany	30
15	2010	South Africa	30
16	2014	Brazil	31
17	1998	France	32

Task 4.3 Query Processing via Spark RDDs

10/25 points

- **#1 Spark Context Creation**
  - Create a spark context sc w/ local master (`local[*]`)
- **#2 Implement Q11 via RDD Operations**
  - Implement Q11 self-contained in executeQ11RDD()
  - All reads should use `sc.textFile(fname)`
  - RDD operations only → stdout
- **#3 Implement Q12 via RDD Operations**
  - Implement Q12 self-contained in executeQ12RDD()
  - All reads should use `sc.textFile(fname)`
  - RDD operations only → stdout

See Spark online documentation for details

Query Processing via Spark SQL

- **#1 Spark Session Creation**
  - Create a spark session via a spark session builder and w. local master (`local[*]`)

6/25 points

- **#2 Implement Q11 via Dataset Operations**
  - Implement Q11 self-contained in executeQ11Dataset()
  - All reads should use `sc.read().format("csv")`
  - SQL or Dataset operations only → JSON

See Spark online documentation for details

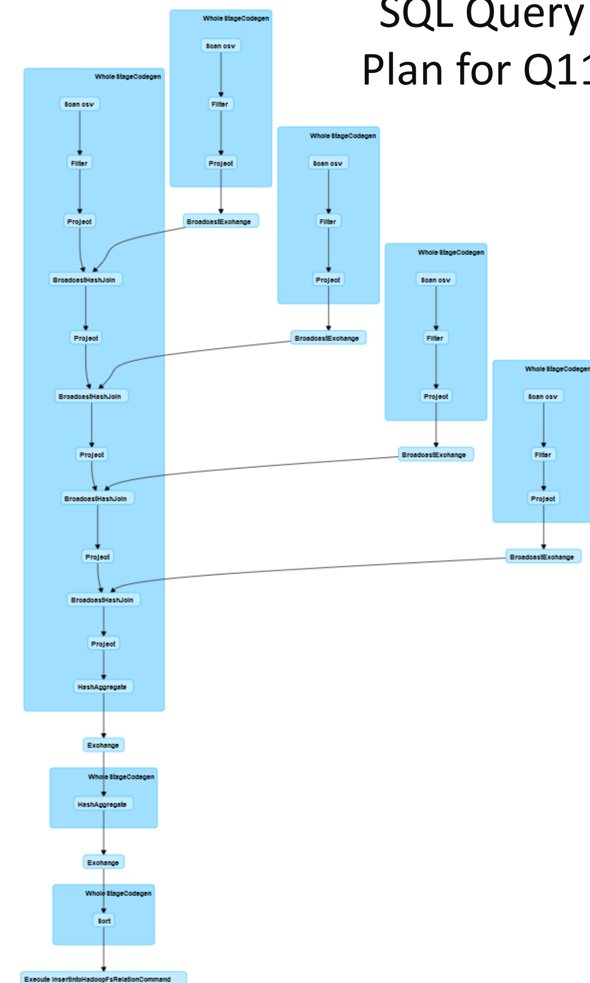
- **#3 Implement Q12 via Dataset Operations**
  - Implement Q12 self-contained in executeQ12Dataset()
  - All reads should use `sc.read().format("csv")`
  - SQL or Dataset operations only → JSON

→ SQL processing of high importance in modern data management

Query Processing via Spark SQL, cont.

- **Optional: Explore Spark Web UI**
  - Web UI started even in local mode
  - Explore distributed jobs and stages
  - Explore effects of caching on repeated query processing
  - Explore statistics

```
INFO Utils: Successfully started
   service 'SparkUI' on port 4040.
INFO SparkUI: Bound SparkUI to 0.0.0.0, and
   started at http://192.168.108.220:4040
```



SQL Query Plan for Q11

Conclusions and Q&A

- **Summary** **13 Data stream processing systems**
  - Data Stream Processing
  - Distributed Stream Processing
  - Exercise 4: Large-Scale Data Analysis

- **Next Lectures/Exams**
  - **Jun 17: Q&A and exam preparation**
  - **Jun 24, 4pm Exam** DB / DB1, HS i13
  - **Jun 27, 4pm Exam** DB / DB1, HS i13
  - **Jun 27, 7.30pm Exam** DB / DB1, HS i13