TECHNISCHE UNIVERSITÄT BERLIN

Data Integration and Large-scale Analysis (DIA)
12 Distributed Stream Processing

Prof. Dr. Matthias Boehm
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

Last update: Jan 10, 2025

BIFOLD

Announcements / Administrative Items

- **#1 Video Recording**
  - Hybrid lectures: in-person H 0107, zoom live streaming, video recording
  - https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09

- **#2 Exercise/Project Submission**
  - Submission deadline: **Jan 30, 11.59pm**
  - Pull-requests submitted (not necessarily merged) by deadline

- **#3 Course Evaluation**
  - Lecture: https://befragung.tu-berlin.de/evasys/online.php?pswd=JNXRG
  - Exercise: https://befragung.tu-berlin.de/evasys/online.php?pswd=1X7EN

Lecture

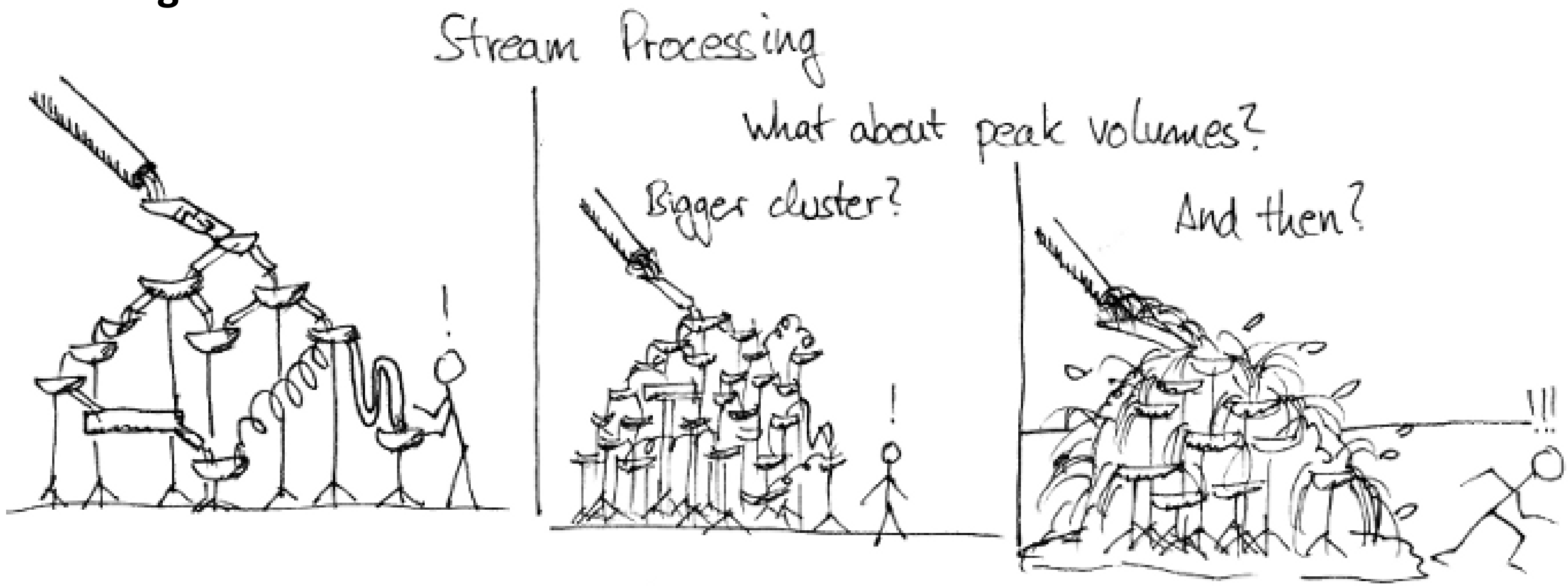
Exercise

Course Outline Part B: Large-Scale Data Management and Analysis

	12 Distributed Stream Processing	13 Distributed Machine Learning Systems
Compute/ Storage	11 Distributed Data-Parallel Computation	
	10 Distributed Data Storage	
Infra	09 Cloud Resource Management and Scheduling	
	08 Cloud Computing Fundamentals	

Agenda

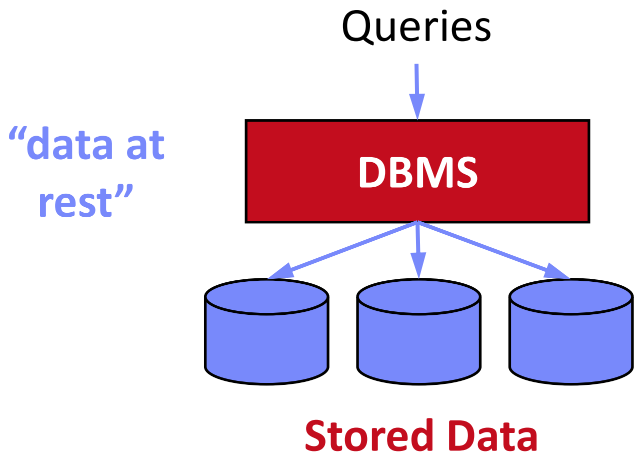
- **Data Stream Processing**
- **Distributed Stream Processing**
- **Data Stream Mining**

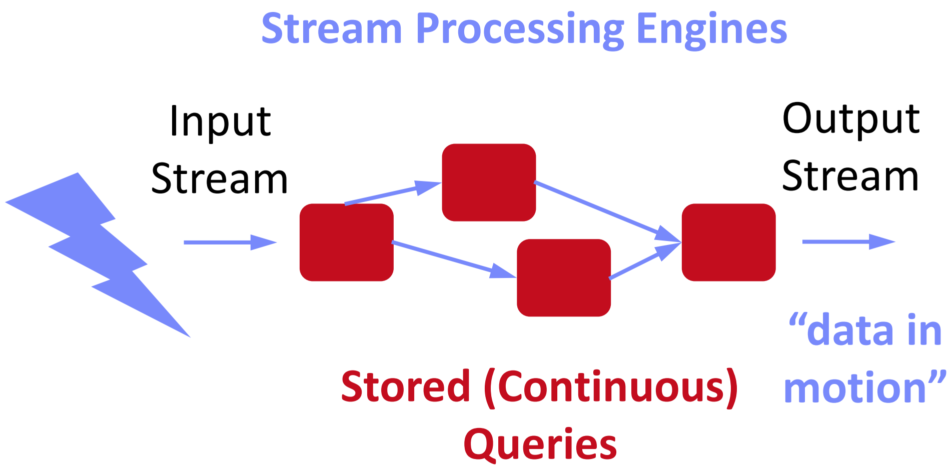


Data Stream Processing

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 queries (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 **DM 10 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

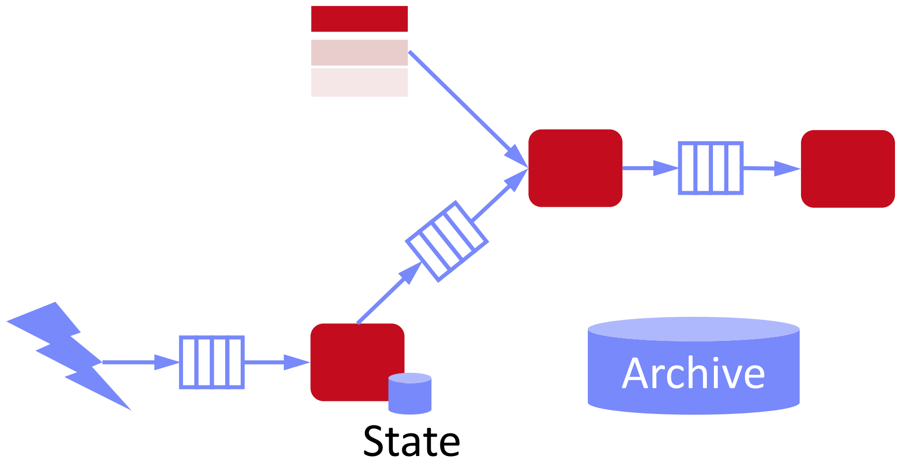
History of Stream Processing Systems

- **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

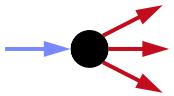
- **2010s**
  - **Distributed stream processing engines**, and "unified" batch/stream processing
  - **Proprietary systems:** Google Cloud Dataflow, MS StreamInsight / Azure Stream Analytics, IBM InfoSphere Streams / Streaming Analytics, AWS Kinesis
  - **Open-source systems:** Apache Spark Streaming (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 operations w/ state (e.g., separate threads)
  - **Edges:** data dependencies (tuple/message streams)
  - **Push model:** data production controlled by source



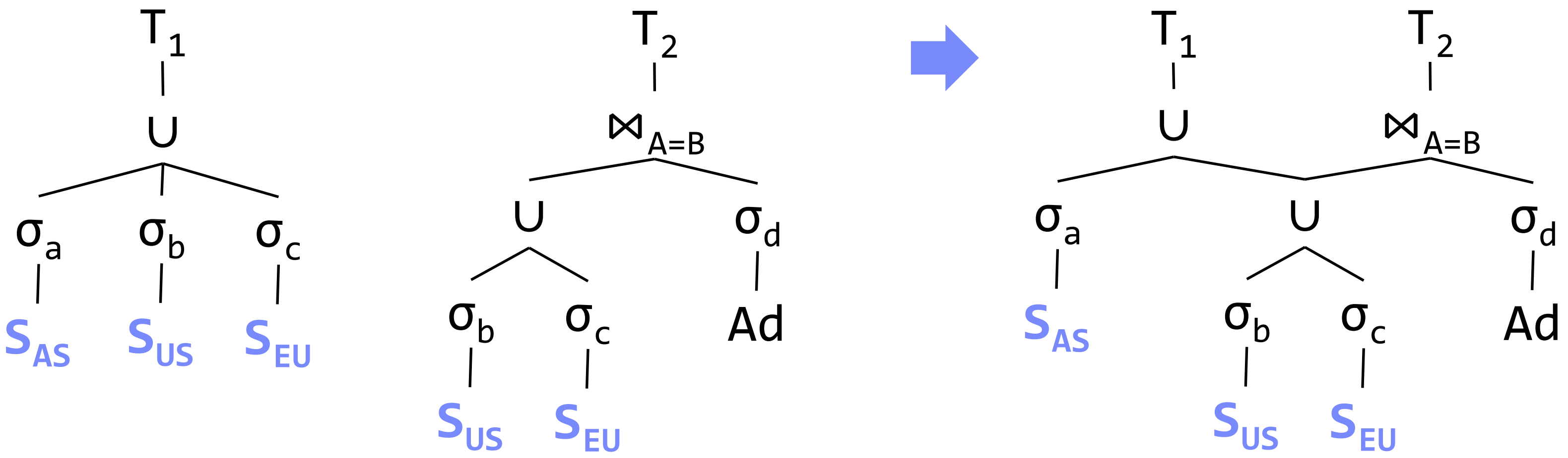
- **Operator Model**
  - Read from input queue
  - Write to potentially many output queues
  - Example Selection $\sigma_{A=7}$



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

System Architecture – Sharing

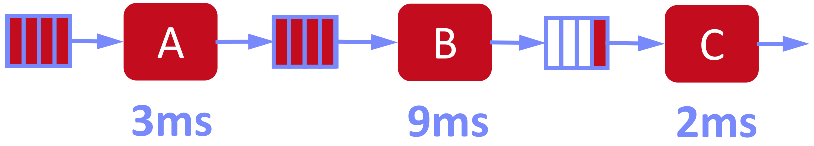
- **Multi-Query Optimization**
  - Given **set of continuous queries** (deployed), compile minimal DAG w/o redundancy (see **DM 08 Physical Design MV**) ➔ **subexpression elimination**



- **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

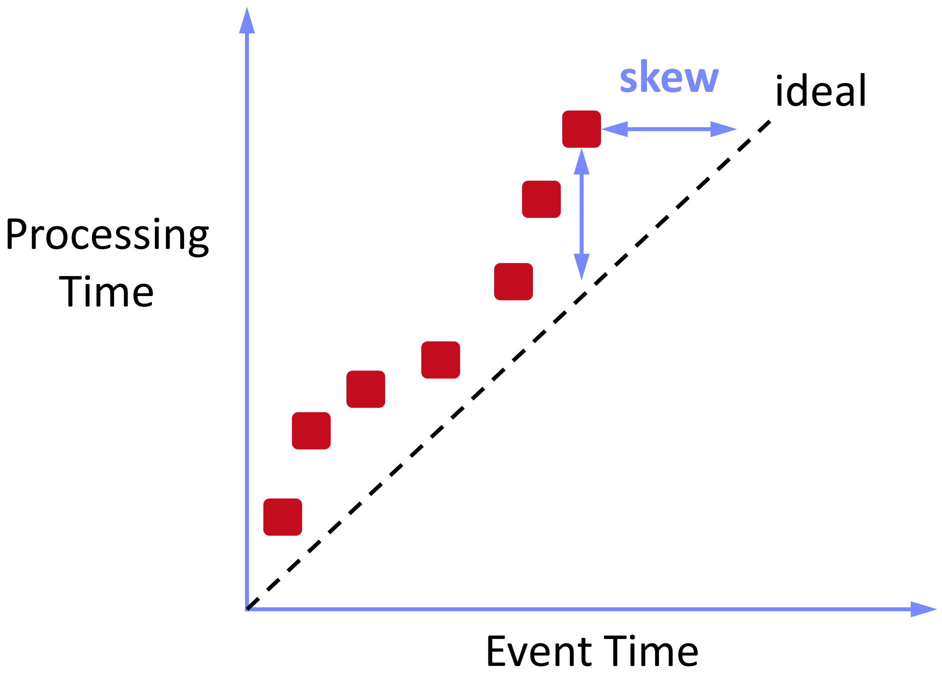
[Nesime Tatbul et al: Load Shedding in a Data Stream Manager. **VLDB 2003**]

- **#3 Distributed Stream Processing** (see next part)
  - 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 Delivery Guarantees

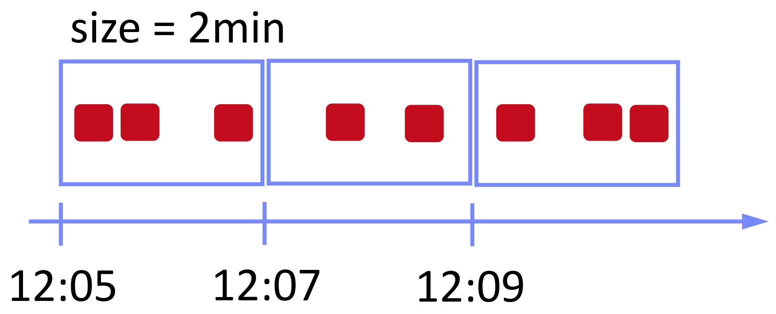
- **#1 At Most Once**
  - **“Send and forget”**, ensure data is never counted twice
  - Might cause data loss on failures
- **#2 At Least Once**
  - **“Store and forward”** or acknowledgements from receiver, replay stream from a checkpoint on failures
  - Might create incorrect state (processed multiple times)
- **#3 Exactly Once**
  - **“Store and forward” w/ guarantees** regarding state updates and sent msgs
  - Often via dedicated transaction mechanisms

03 Message-oriented Middleware, EAI, and Replication

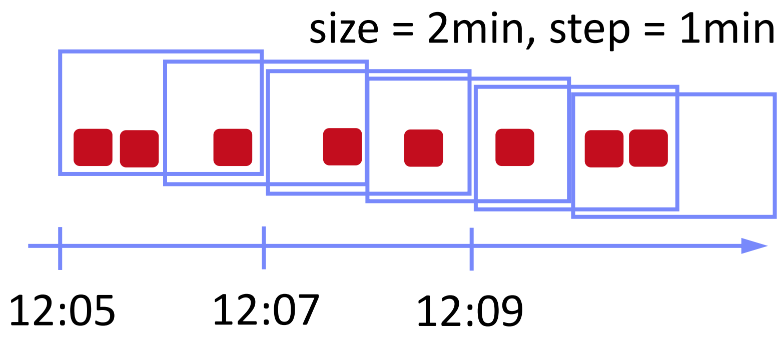
Window Semantics

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

- **#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



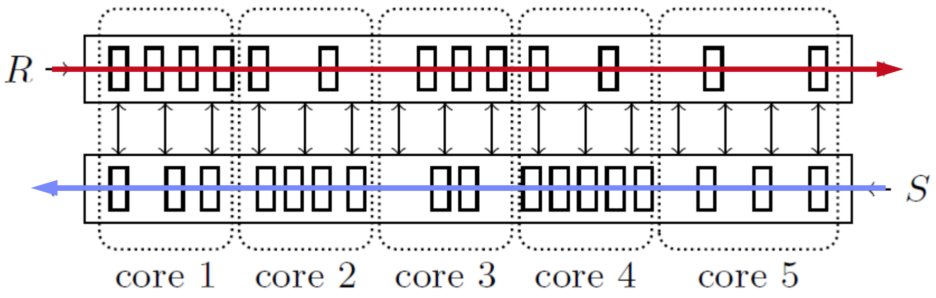
Stream Joins

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

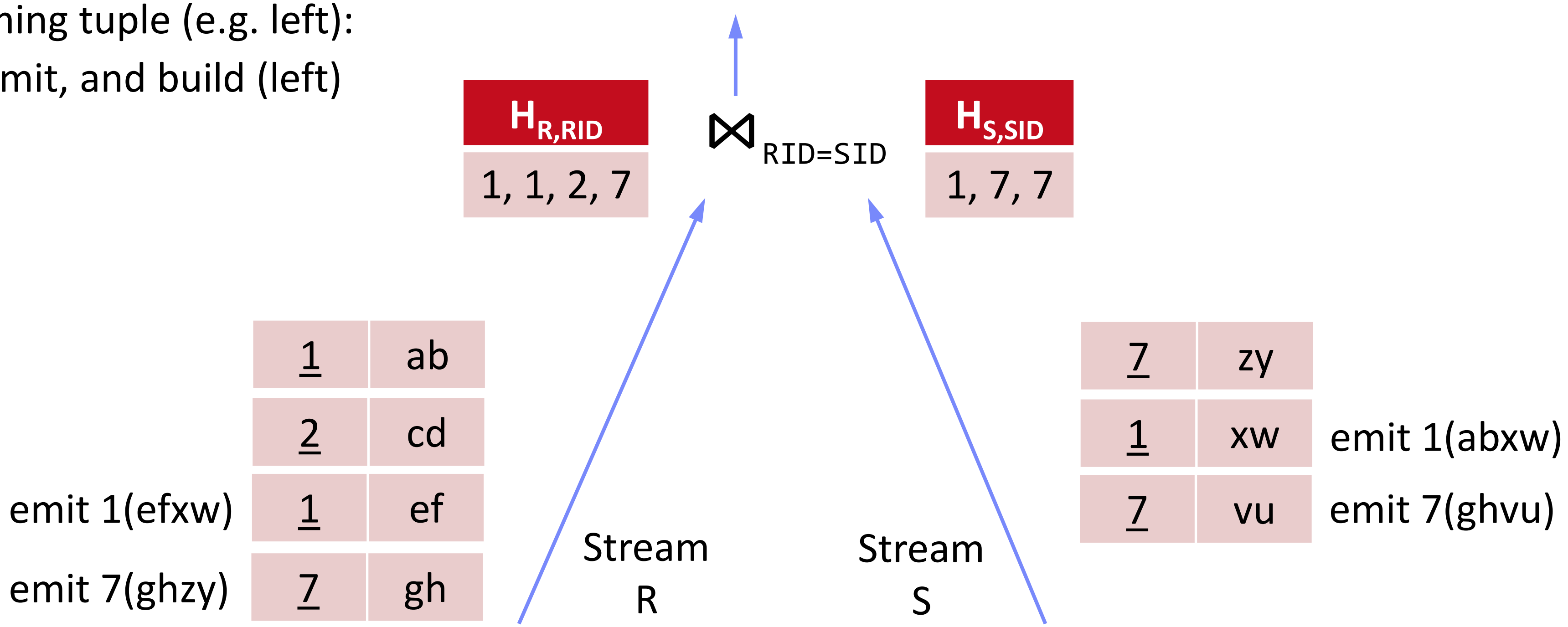
[Jens Teubner, René Müller: How soccer players would do stream joins. **SIGMOD 2011**]



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

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

- **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

- **Baseline Query Execution Plan**

Self join	$\bowtie_{tb=tb+1}$
High-level aggregation	γ_2
Low-level aggregation	γ_1
Low-level filtering	σ
	TCP

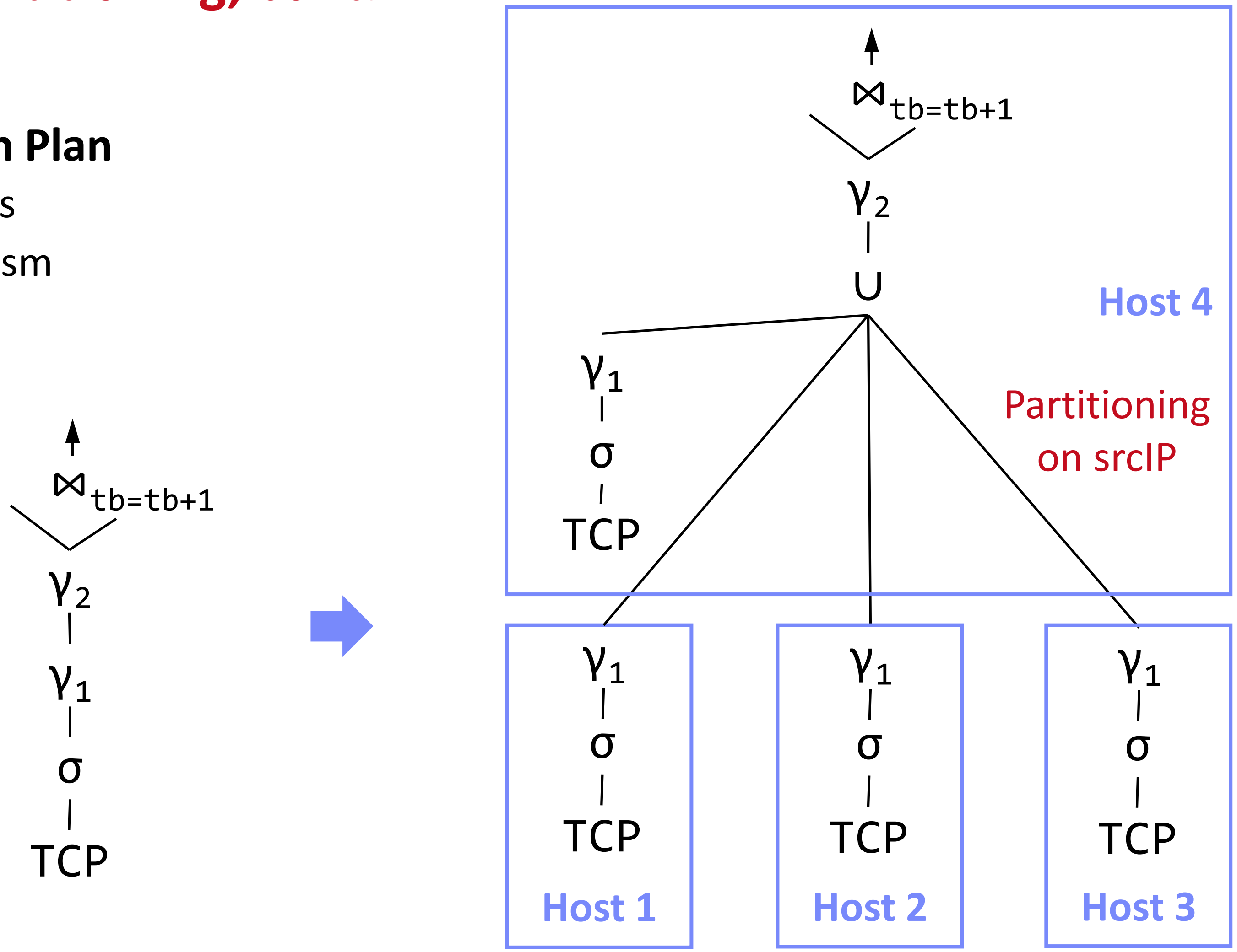
```
Query flow_pairs:
SELECT S1.tb, S1.srcIP, S1.max, S2.max
   FROM heavy_flows S1, heavy_flows S2
   WHERE S1.srcIP = S2.srcIP and S1.tb = S2.tb+1

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
```

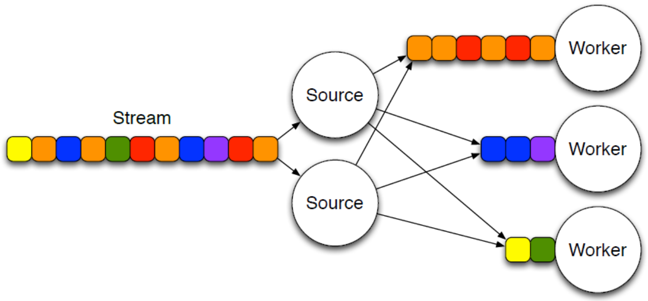
Query-Aware Stream Partitioning, cont.

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



Stream Group Partitioning

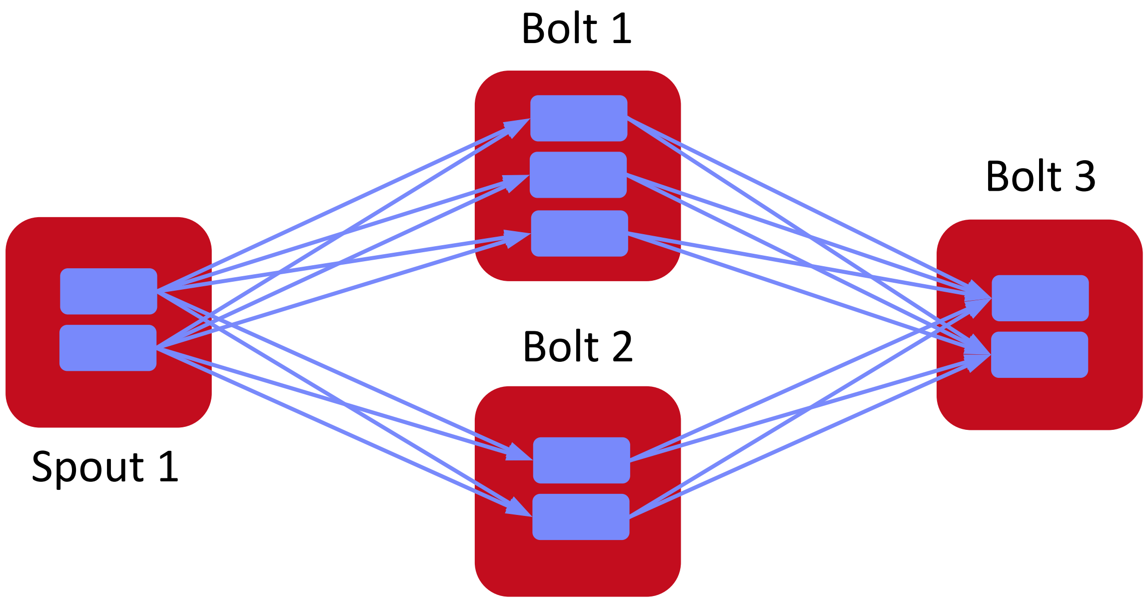
- **Large-Scale Stream Processing**
  - Limited pipeline parallelism and task parallelism (independent subqueries)
  - Combine with **data-parallelism over stream groups**
- **#1 Shuffle Grouping**
  - Tuples are randomly distributed across consumer tasks
  - Good load balance
- **#2 Fields Grouping**
  - 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 (associative agg)
- **#4 Others: Global, None, Direct, Local**



[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)



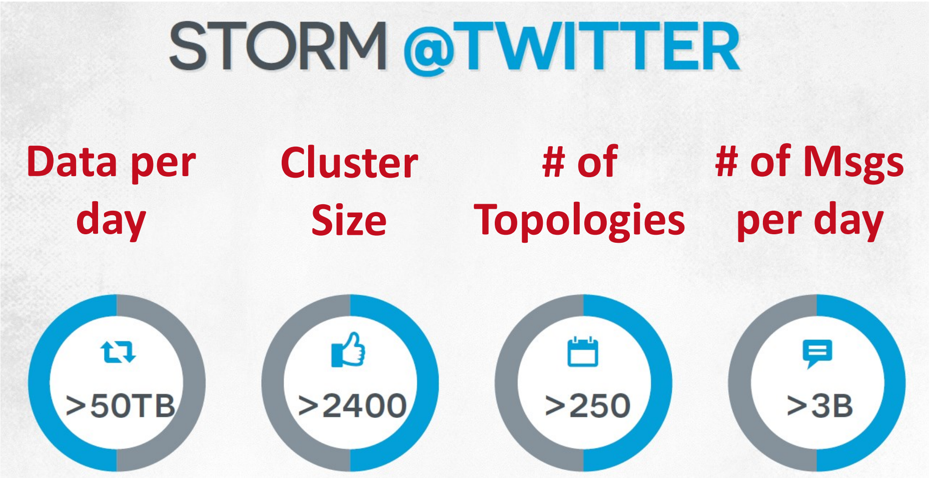

```
Config conf = new Config();
conf.setNumWorkers(3);

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

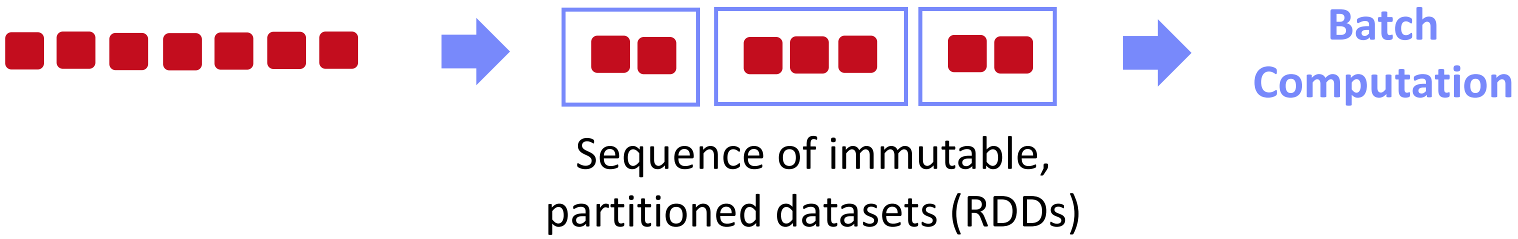
[Avrilia Floratou et al: Dhalion: Self-Regulating Stream Processing in Heron. **PVLDB 2017**]

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

Discretized Stream (Batch) Computation

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

- **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**

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
- **Google Cloud Dataflow**
  - **Tuple-at-a-time** with exactly-once guarantee
  - MR → FlumeJava → MillWheel → Dataflow (managed batch/stream service)

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

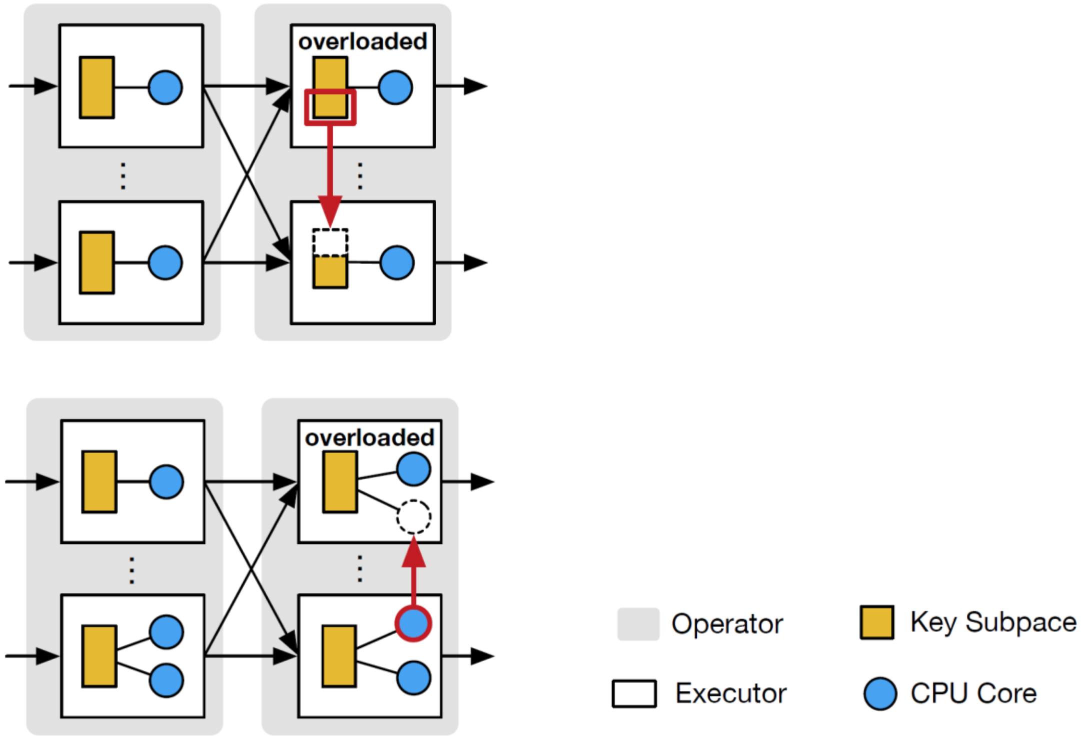
[https://flink.apache.org/news/2019/02/13/unified-batch-streaming-blink.html]

[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**]

Resource Elasticity

[Li Wang, Tom Z. J. Fu, Richard T. B. Ma, Marianne Winslett, Zhenjie Zhang: Elasticutor: Rapid Elasticity for Realtime Stateful Stream Processing. **SIGMOD 2019**]

- **#1 Static**
  - Static, operator-level key partitioning
- **#2 Resource-Centric**
  - Dynamic, operator-level key partitioning
  - **Global synchronization** for key repartitioning and state migration
- **#3 Executor-Centric**
  - Static, operator-level key partitioning
  - **CPU core reassignments** via local and remote tasks



Data Stream Mining

Selected Example Algorithms

Overview Stream Mining

- **Streaming Analysis Model**
  - Independent of actual storage model and processing system
  - Unbounded stream of data item S = (s_1, s_2, …)
  - Evaluate function f(S) as aggregate over stream or window of stream
  - Standing vs ad-hoc queries
- **Recap: Classification of Aggregates**
  - **Additive** aggregation functions (SUM, COUNT)
  - **Semi-additive** aggregation functions (MIN, MAX)
  - **Additively computable** aggregation functions (AVG, STDDEV, VAR)
  - ~~Aggregation functions (MEDIAN, QUANTILES)~~ → approximations

02 Data Warehousing, ETL, and SQL/OLAP

→ Selected Algorithms
  - Higher-Order Statistics (e.g., STDDEV)
  - Approximate # Distinct Items (e.g., KMV, HyperLogLog)
  - Approximate Heavy Hitters (e.g. CountMin-Sketch)

Higher-Order Statistics

- **Overview Order Statistics**
  - Many order statistics computable via p^{th} central moment
  - **Examples:** Variance σ^2, skewness, kurtosis

$$m_p = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^p$$

- **Incremental Computation of Variance**
  - **#1** Default 2-pass algorithm (mean, and squared diffs)
  - **#2** Textbook 1-pass algorithm (incrementally maintainable)
    ➔ **numerically instable**
  - **#3** Incremental update rules for m_p
    with Kahan addition (variance since 1979)

$$\sigma^2 = \frac{n}{n-1} m_2$$

$$\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i\right)^2$$

[Yuanyuan Tian, Shirish Tatikonda, Berthold Reinwald: Scalable and Numerically Stable Descriptive Statistics in SystemML. **ICDE 2012**]

$$n = n_a + n_b, \quad \delta = \mu_b - \mu_a, \quad \mu = \mu_a \oplus n_b \frac{\delta}{n}$$

$$M_p = M_{p,a} \oplus M_{p,b} \oplus \{\sum_{j=1}^{p-2} \binom{p}{j} [(-\frac{n_b}{n})^j M_{p-j,a} + (\frac{n_a}{n})^j M_{p-j,b}]\delta^j + (\frac{n_a n_b}{n}\delta)^p [\frac{1}{n_b^{p-1}} - (\frac{-1}{n_a})^{p-1}]\}$$

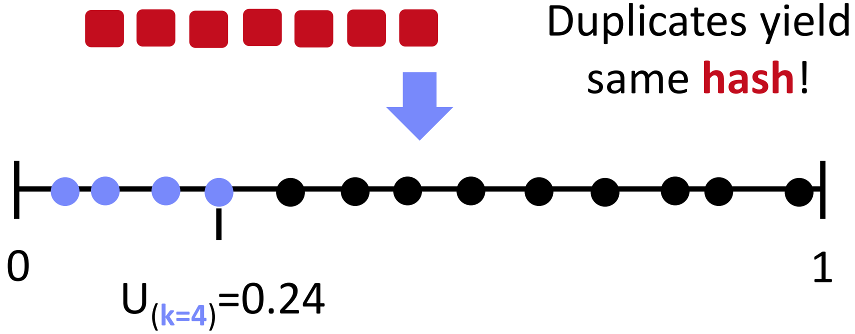
11 Distributed, Data-Parallel Computation

Number of Distinct Items

[Kevin S. Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, Rainer Gemulla: On synopses for distinct-value estimation under multiset operations. **SIGMOD 2007**]

- **Problem**
  - **Estimate # distinct items** in a dataset / data stream w/ limited memory
  - Support for set operations (union, intersect, difference)

- **K-Minimum Values (KMV)**
  - Hash values d_i to $h_i \in [0, M]$
  - Domain $M = O(D^2)$ to avoid collisions → $\mathbf{O}(\boldsymbol{k \log D})$ **space**
  - **Store k minimum hash values** (e.g., via priority queue) in normalized form $h_i \in [0,1]$
  - Basic estimator:
  - **Unbiased estimator:**



$$\hat{D}_k^{BE} = k/U_{(k)}$$

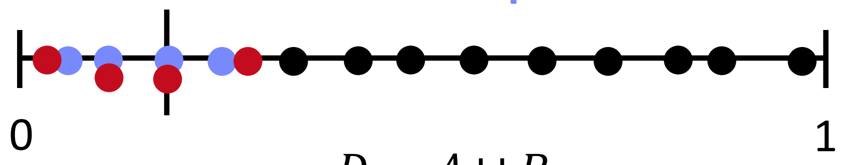
$$\hat{D}_k^{UB} = (k-1)/U_{(k)}$$

Example:
16.67 vs 12.5

Number of Distinct Items, cont.

- **KMV** **Set Operations**
  - Union and intersection directly on partition synopses
  - Difference via **Augmented KMV** (AKMV) that include counters of multiplicities of k-minimum values

11 Distributed, Data-Parallel Computation



$$D = A \cup B$$
$$KMV(D_\cup) \equiv KMV(A) \oplus KMV(B)$$

- **HyperLogLog**
  - Hash values and maintain maximum **# of leading zeros** p → $\hat{D} = 2^p$
  - Stochastic averaging over m sub-streams (p maintain in registers M)
  - **HyperLogLog++**

[P. Flajolet, Éric Fusy, O. Gandouet, and F. Meunier: Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. **AOFA 2007**]

[Stefan Heule, Marc Nunkesser, Alexander Hall: HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. **EDBT 2013**]

Stream Summarization

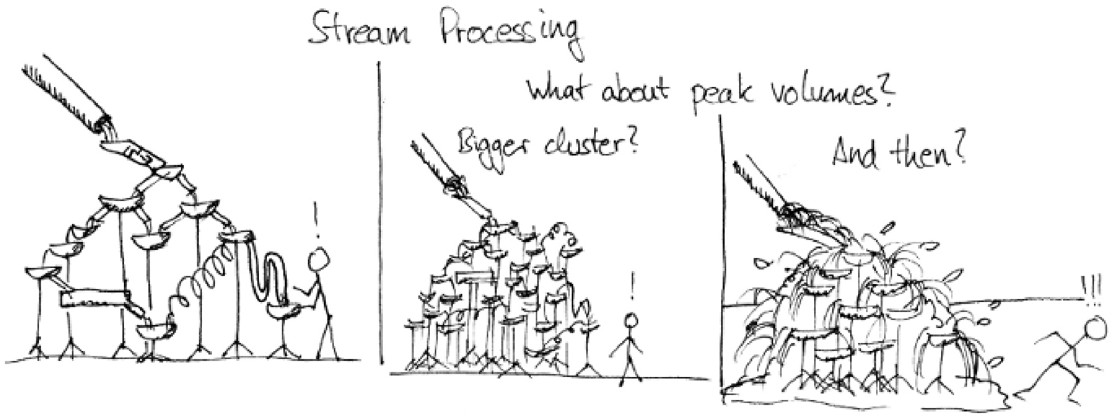
[Graham Cormode, S. Muthukrishnan: An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. LATIN 2004]

- **Problem**
  - **Summarize stream in sketch**/synopsis w/ limited memory
  - Finding quantiles, frequent items (heavy hitters), etc
- **Count-Min (CM) Sketch**
  - Two-dimensional count array of width w and depth d
  - d hash functions map {1 … n} → {1 … w}
  - **Update (s_i,c_i):** compute d hashes for s_i and increase counts of all locations
  - **Point query (s_i):** compute d hashes for s_i and estimate frequency as $\min(count[j,h_j(s_i)])$

Unlikely similar hash collisions

h_1		6			2	1
h_2	1		3	5		
h_3	3		4		1	1
h_4		1	2	1	5	
h_d		7	1	1		

Summary and Q&A



- **Data Stream Processing**
- **Distributed Stream Processing**
- **Data Stream Mining**

- **Next Lectures (Large-scale Data Management and Analysis)**
  - **13 Distributed Machine Learning Systems** [Jan 23, 4pm]
  - **14 Exam Preparation** [Jan 23, 6pm]