Data Management
12 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 08, 2020

SCIENCE
PASSION
TECHNOLOGY

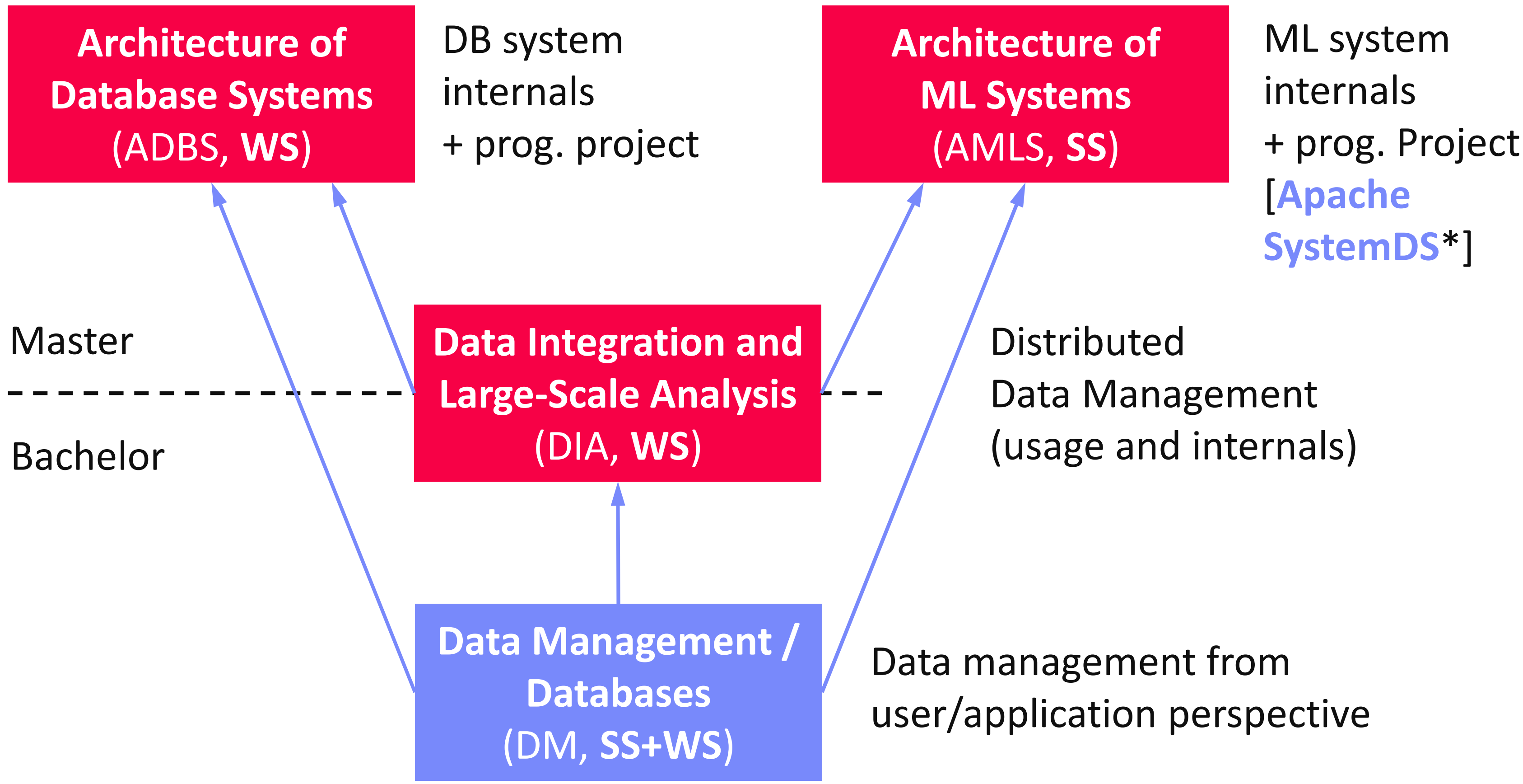
Announcements/Org

- **#1 Video Recording**
  - Link in **TeachCenter** & **TUbe** (lectures will be public)
  - **Live Streaming** Mo 4.10pm until end of semester (June 30)
  - **Office hours:** Mo 1pm-2pm (https://tugraz.webex.com/meet/m.boehm)
- **#2 Exercises**
  - **Exercise 1 graded**, Exercise 2 feedback soon
  - Exercise 4 deadline **June 16 11.59pm**
- **#3 Exam Dates** (VR Teaching Planning until June 27)
  - **June 22: 11am-1pm**, **2pm-4pm**, **5pm-7pm** (concurrently in i7, i11, i12, i13)
  - **Deregistration possible w/o failed attempt** for KU/VUs

DM 11am: **77/77**
DM 2pm: **77/77**
DM 5pm: **53/77**
DB 11am: **15+3/15**
DB 2pm: **15+3/15**
DB 5pm **15+3/15**
DM/DB Oral: **5**

- **#4 Course Evaluation**
  - Please participate; open period: **June 1 – July 15**

#5 Data Management Courses

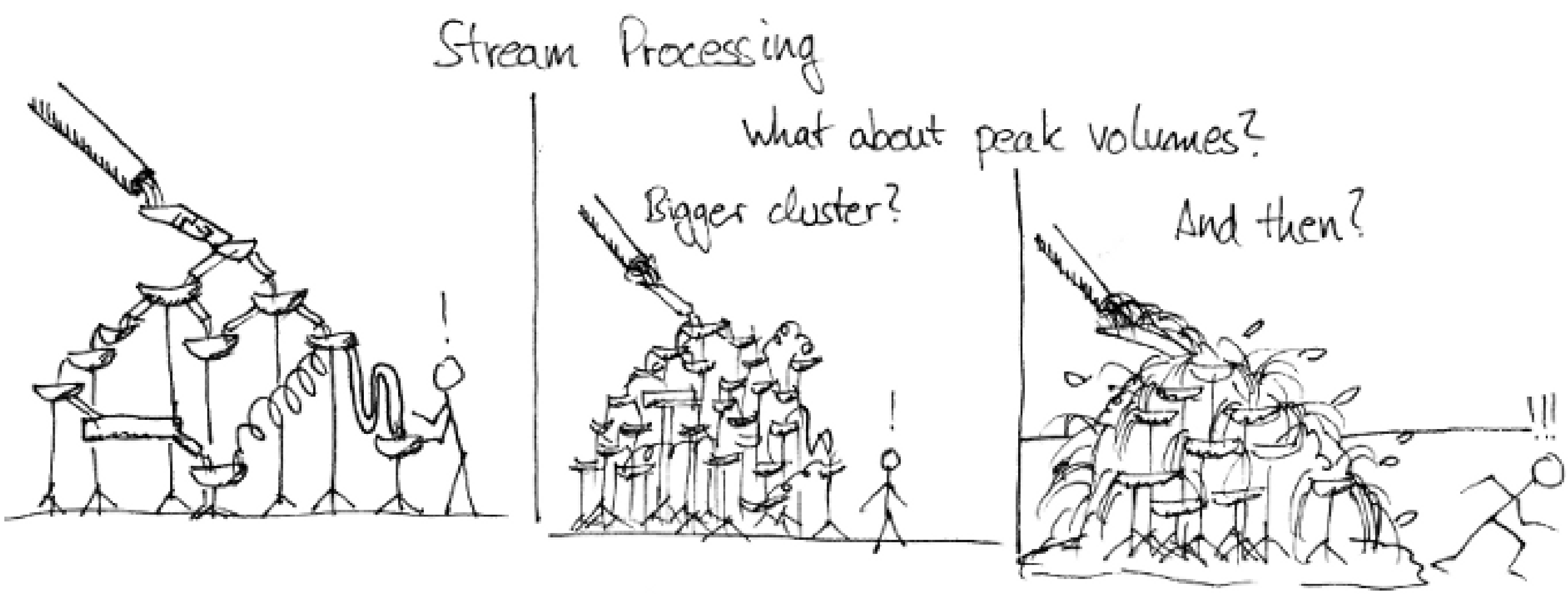


*[https://github.com/apache/systemml]

Agenda

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

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

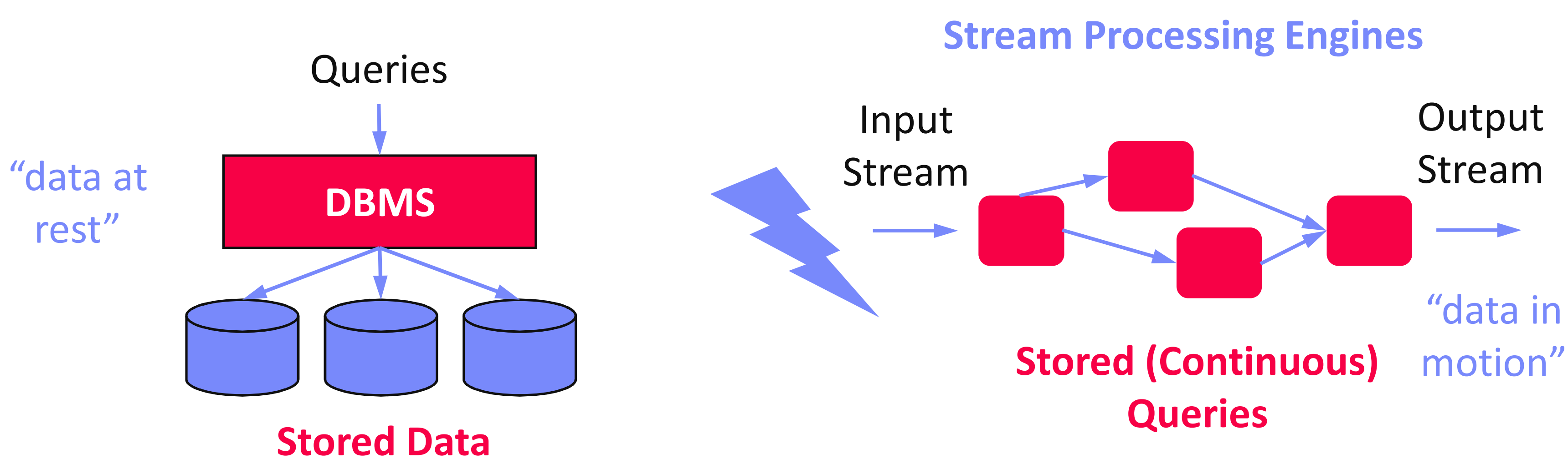


Data Stream Processing

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 **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

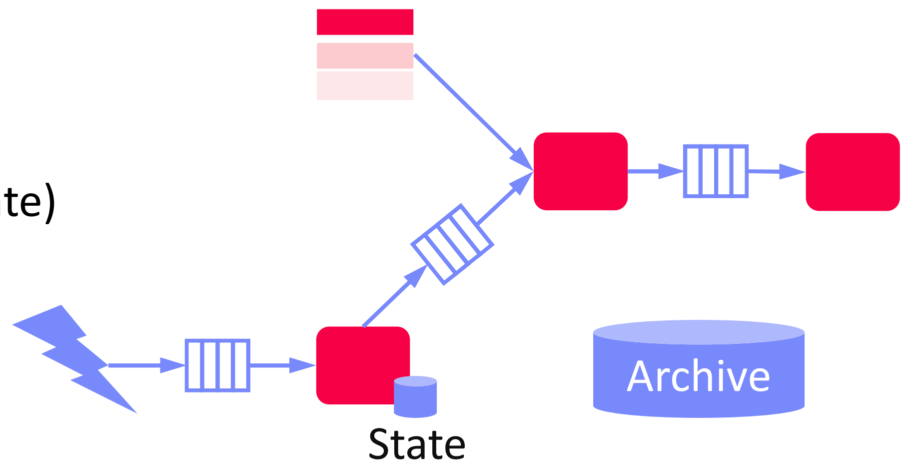
History of Stream Processing Systems

- **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

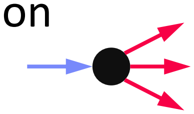
    ➔ but mostly unsuccessful in industry/practice
  - **Message-oriented middleware** and **Enterprise Application Integration** (EAI): IBM **Message Broker**, SAP **eXchange Infra.**, MS **Biztalk Server**, **TransConnect**
- **2010s**
  - **Distributed stream processing engines**, and "unified" batch/stream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



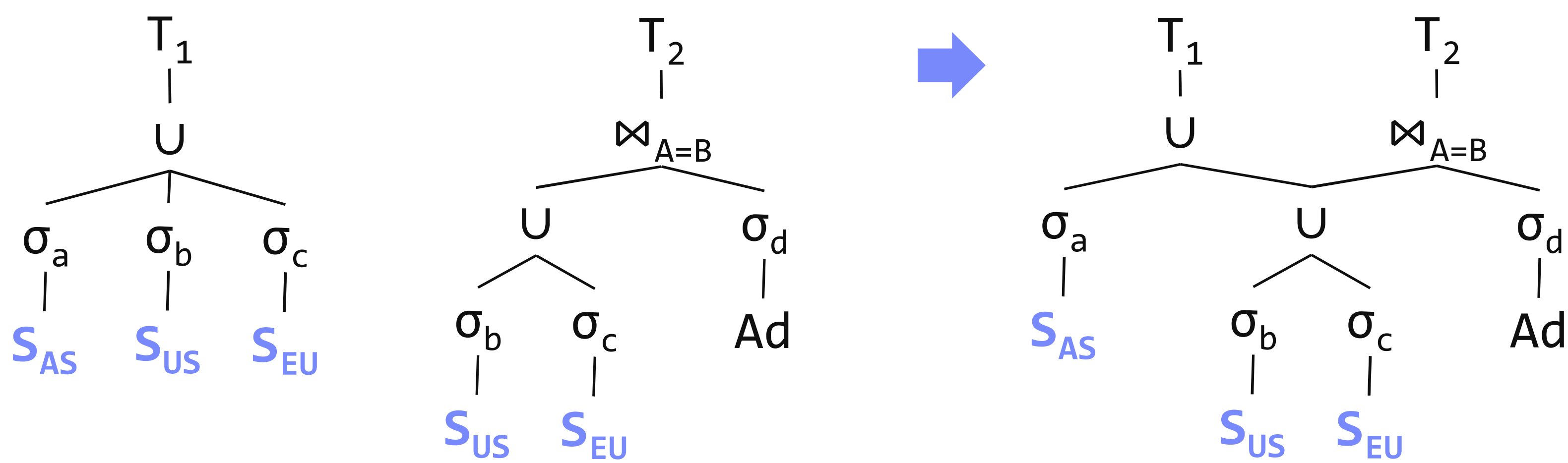
- **Operator Model**
  - Read from input queue
  - Write to potentially many output queues
  - Example Selection $\sigma_{A=7}$



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

System Architecture – Sharing

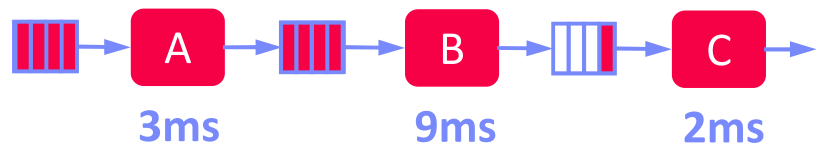
- **Multi-Query Optimization**
  - Given **set of continuous queries** (deployed), compile DAG w/o redundancy (see **08 Physical Design MV**) → **common subexpression elimination**



- **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

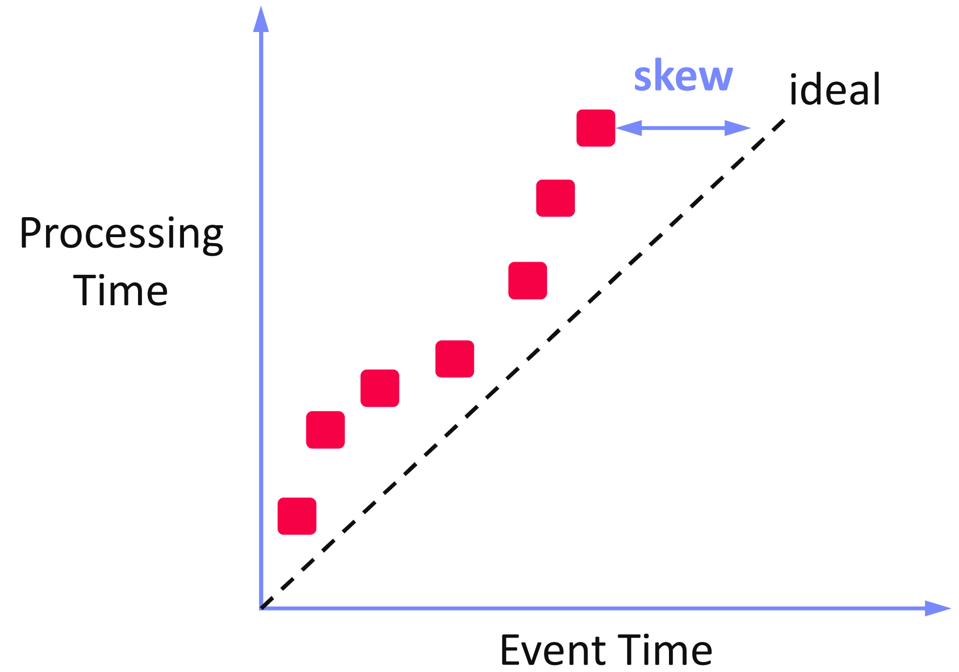
- **#2 Load Shedding**
  - #1 **Random-sampling**-based load shedding
  - #2 **Relevance-based** load shedding
  - #3 **Summary-based** load shedding (synopses)
  - Given SLA, select queries and shedding placement that minimize error and satisfy constraints

[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**)

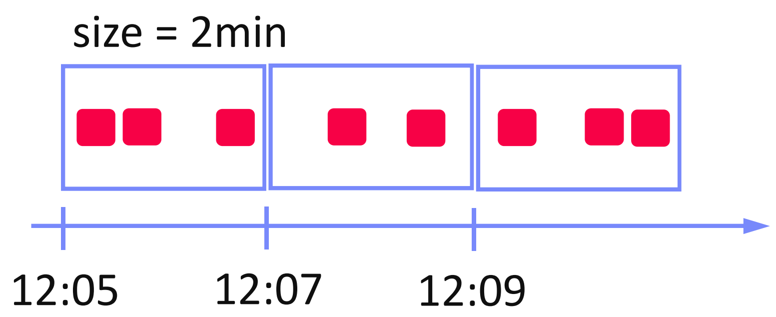
Durability and Consistency Guarantees

- **#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

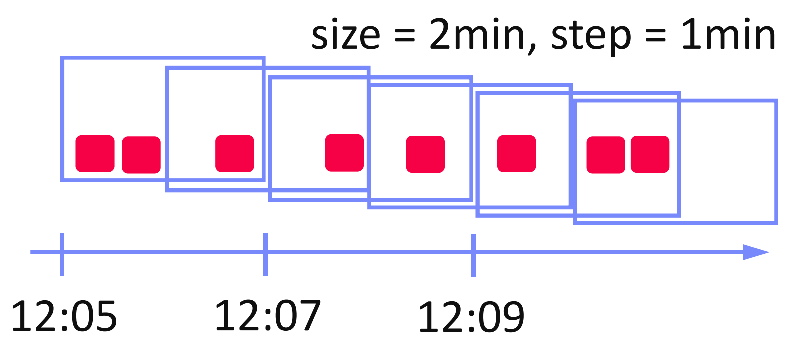
Window Semantics

- **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



Spark Streaming Example

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

```
// create spark context w/ batch interval 1s
sc = new JavaStreamingContext(conf, Durations.seconds(1));

// create DStream listening on socket (ip, port)
lines = sc.socketTextStream("localhost", 9999);

// traditional word count example on Dstream batches
JavaPairDStream<String, Integer> wordCounts = lines
   .flatMap(x -> Arrays.asList(x.split(" ")).iterator())
   .mapToPair(s -> new Tuple2<>(s, 1))
   .reduceByKey((i1, i2) -> i1 + i2);
wordCounts.print();
```

Tumbling
1s Window

```
// extended word count example on Dstream windows
JavaPairDStream<String, Integer> wordCounts2 = lines
   .flatMap(x -> Arrays.asList(x.split(" ")).iterator())
   .mapToPair(s -> new Tuple2<>(s, 1))
   .reduceByKeyAndWindow((i1, i2) -> i1 + i2,
      Durations.seconds(30), Durations.seconds(10));
```

Sliding
30s Window

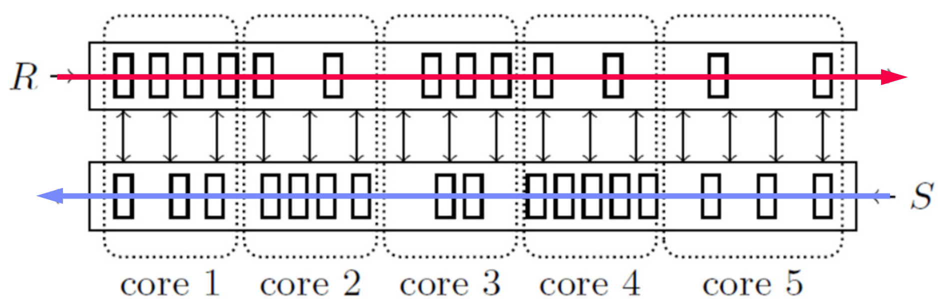
Window Length (Durations.seconds(30)) — **Sliding Interval** (Durations.seconds(10))

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

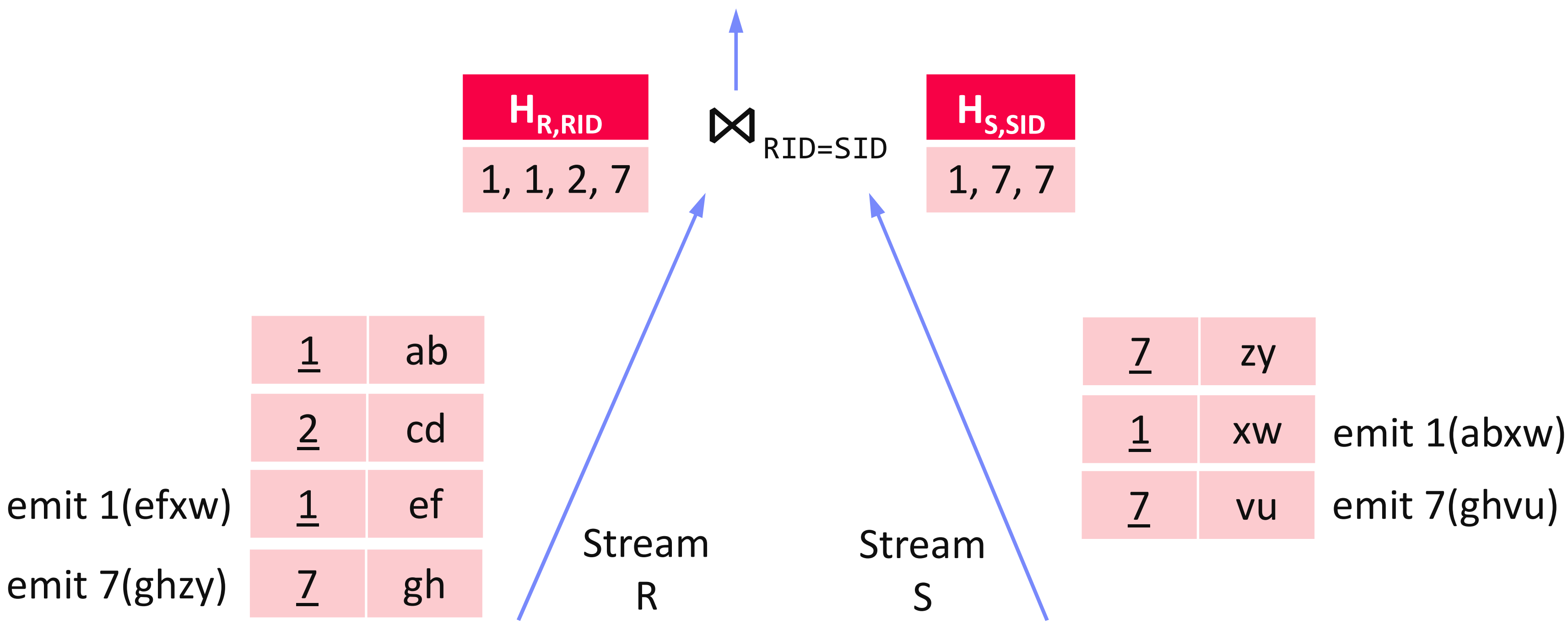


[Jens Teubner, René Müller: How soccer players would do stream joins. **SIGMOD 2011**]

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**]

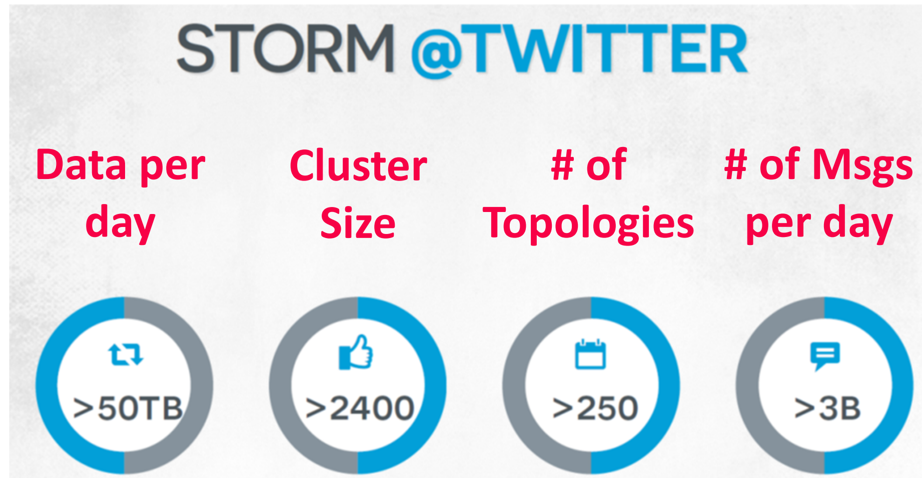
- **Double-Pipelined Hash Join**
  - Join of bounded streams (or unbounded w/ invalidation)
  - **Equi join predicate**, **symmetric and non-blocking**
  - For every incoming tuple (e.g. left): probe (right)+emit, and build (left)



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)**

Q&A and Exam Preparation

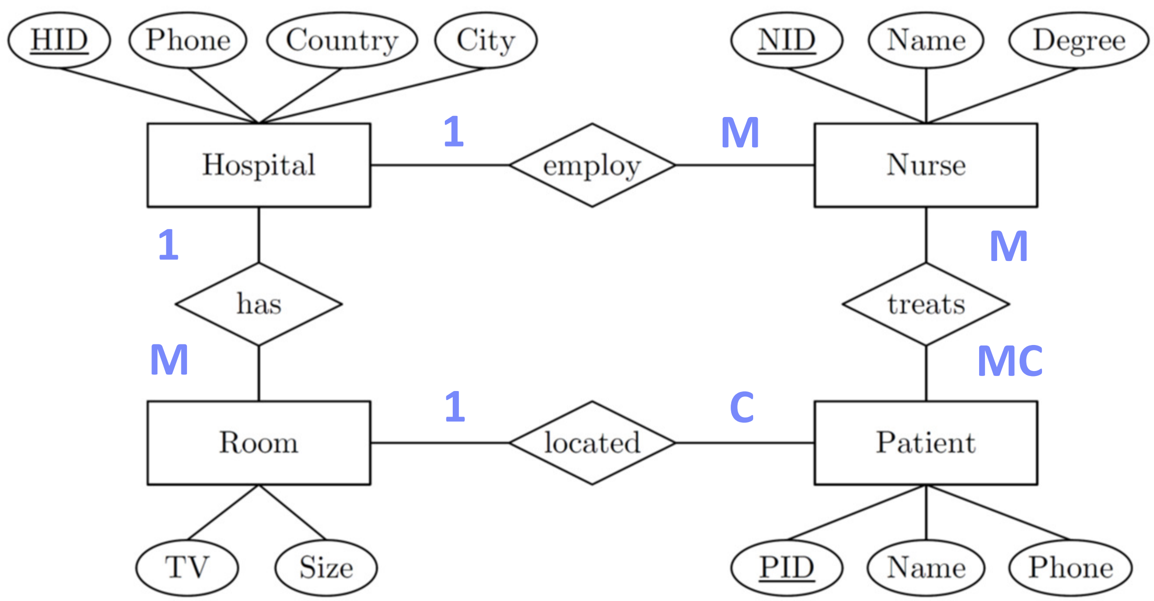
Basic focus: fundamental concepts and ability to apply learned techniques to given problems

Exam Logistics

- **Timing/Logistics**
  - [COVID-19]: Procedure and behavioral guide for on-campus exams (students) https://www.youtube.com/watch?v=dE4xoCjONRM&feature=youtu.be
  - Exam starts 10min after official start
  - 90min working time (plenty of time to think about answers)
  - **Write into the worksheet if possible**, additional paper allowed
  - Grading for June 22 will happen ~ June 27/28 → later exams as replacement
- **Covered Content**
  - **Must-have:** Data modeling/normalization, SQL query processing
  - Relational algebra, physical design, query and transaction processing
  - **DM only:** NoSQL, distributed storage and computation, streaming
- **Past Exams**
  - https://mboehm7.github.io/teaching/ws1920_dbs/index.htm (3+3)
  - https://mboehm7.github.io/teaching/ss19_dbs/index.htm (3+3)

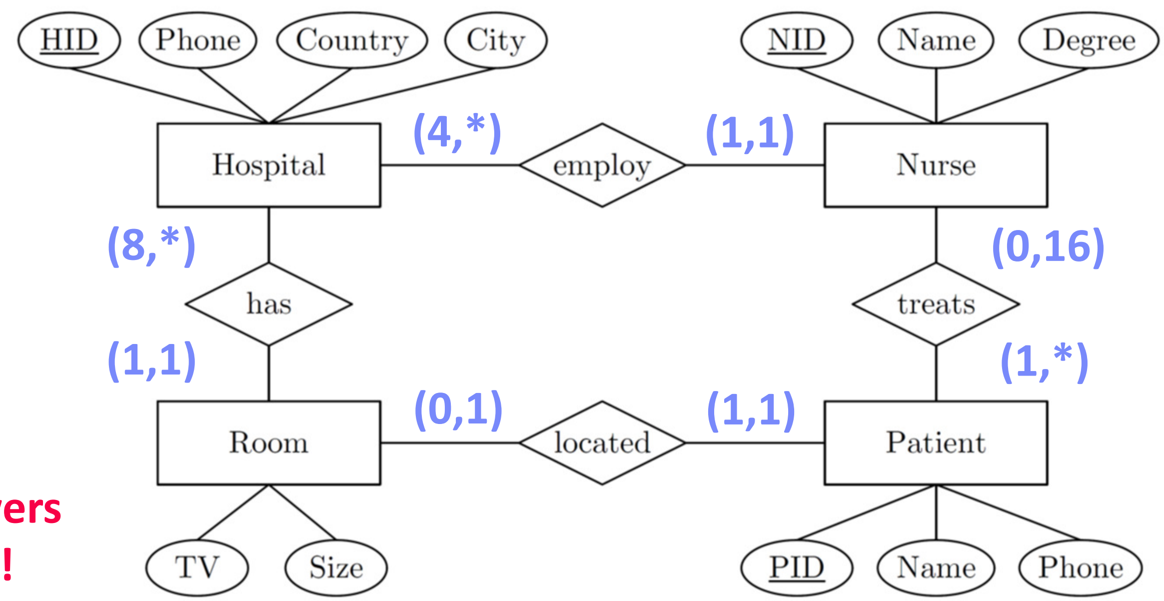
#1 Data Modeling

- **Task 1a: Specify the cardinalities in Modified Chen notation** (8 Points)
  - A hospital employs at least 4 nurses and has at least 8 patient rooms.
  - A nurse works in exactly one hospital and treats up to 16 patients.
  - A patient is treated by at least one but potentially many nurses.
  - Every patient has a room, a rooms belongs to exactly one hospital, and rooms are never shared by multiple patients.



#1 Data Modeling, cont.

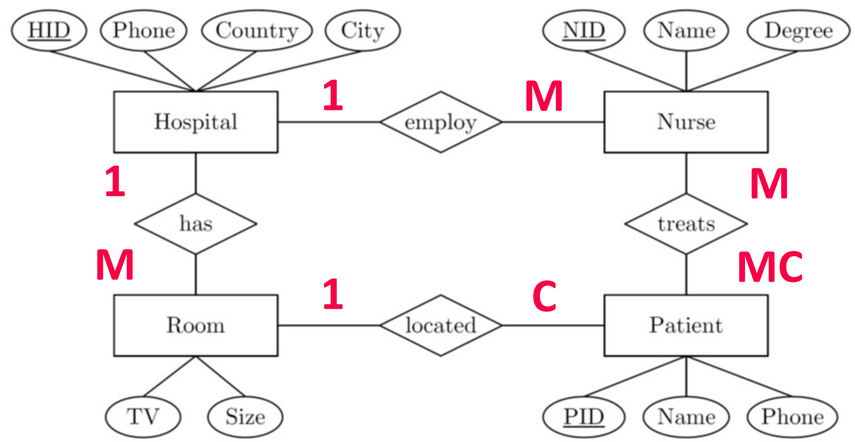
- **Task 1b: Specify the cardinalities in (min, max) notation (4 Points)**
  - A hospital employs at least 4 nurses and has at least 8 patient rooms.
  - A nurse works in exactly one hospital and treats up to 16 patients.
  - A patient is treated by at least one but potentially many nurses.
  - Every patient has a room, a rooms belongs to exactly one hospital, and rooms are never shared by multiple patients.



Only provide answers you're asked for!

#1 Data Modeling, cont.

- **Task 1c: Map the given ER diagram into a relational schema (10 points)**
  - Including data types, primary keys, and foreign keys



- **Solution**
  - Hospitals(
    HID:int, phone:char(16), Country:varchar(64), City:varchar(64))
  - Nurses(
    NID:int, Name:varchar(64), Degree:varchar(32), HID^{FK}:int)
  - Patient(
    PID:int, Name:varchar(64), Phone:char(16), RID^{FK}:int)
  - Room(
    RID:int, TV:boolean, Size:int, HID^{FK}:int)
  - Treated(
    NID^{FK}:int, PID^{FK}:int)

#1 Data Modeling, cont.

- **Task 1d: Bring your schema in 3rd normal form and explain why it is in 3NF (12 points)**
  - Let Hospital.Phone and Patient.Phone be multi-valued attributes
  - Assume the functional dependency City → Country

- **Solution**
  - `Phones(Number:char(16), HID^FK:int, PID^FK:int)` (primary key: Number)
  - `Cities(City:varchar(64), Country:varchar(64))` (primary key: City)
  - `Hospitals(HID:int, City^FK:varchar(64))` (primary key: HID)

  - **1st Normal Form:** no multi-valued attributes
  - **2nd Normal Form:** 1NF + all non-key attributes fully functional dependent on PK
  - **3rd Normal Form:** 2NF + no dependencies among non-key attributes

#2 Structured Query Language

- **Task 2a: Compute the results for the following queries (15 points)**

Orders

OID	Customer	Date	Quantity	PID
1	A	'2019-06-25'	3	2
2	B	'2019-06-25'	1	3
3	A	'2019-06-25'	1	4
4	C	'2019-06-26'	2	2
5	D	'2019-06-26'	1	4
6	C	'2019-06-26'	1	1

Products

PID	Name	Price
1	X	100
2	Y	15
4	Z	75
3	W	120

```
Q1: SELECT DISTINCT Customer, Date
      FROM Orders O, Products P
      WHERE O.PID = P.PID AND Name IN('Y','Z')
```

Customer	Date
A	'2019-06-25'
C	'2019-06-26'
D	'2019-06-26'

```
Q2: SELECT Customer, count(*) FROM Orders
      GROUP BY Customer
      ORDER BY count(*) DESC, Customer ASC
```

Customer	Count
A	2
C	2
B	1
D	1

```
Q3: SELECT Customer, sum(O.Quantity * P.Price)
      FROM Orders O, Products P
      WHERE O.PID = P.PID
      GROUP BY Customer
```

Customer	Sum
A	120
B	120
C	130
D	75

#2 Structured Query Language, cont.

- **Task 2b: Write SQL queries to answer the following Qs** (15 points)

Orders

OID	Customer	Date	Quantity	PID
1	A	‘2019-06-25’	3	2
2	B	‘2019-06-25’	1	3
3	A	‘2019-06-25’	1	4
4	C	‘2019-06-26’	2	2
5	D	‘2019-06-26’	1	4
6	C	‘2019-06-26’	1	1

Products

PID	Name	Price
1	X	100
2	Y	15
4	Z	75
3	W	120

Q4: Which products where bought on 2019-06-25 (return the distinct product names)?

```
SELECT DISTINCT P.Name
  FROM Orders O, Products P
  WHERE O.PID = P.PID
    AND Date = ‘2019-06-25’
```

Q5: Which customers placed only one order?

```
SELECT Customer FROM Orders
  GROUP BY Customer HAVING count(*) = 1
```

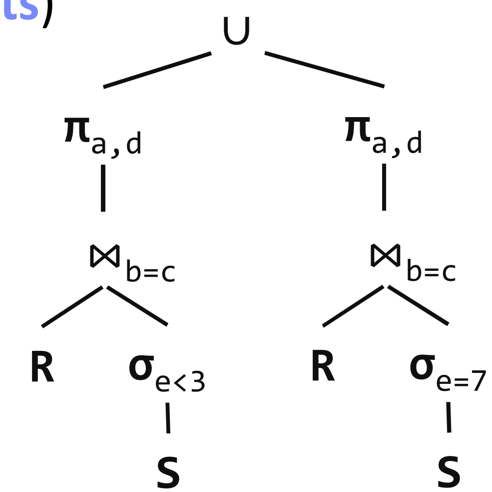
Q6: How much revenue (sum(O.Quantity * P.Price)) did products with a price less then 90 generate (return (product name, revenue))?

```
SELECT P.Name, sum(O.Quantity * P.Price)
  FROM Orders O, Products P
  WHERE O.PID = P.PID AND Price < 90
  GROUP BY P.Name
```

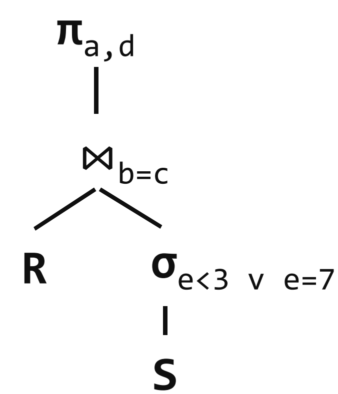
#3 Query Processing

- **Task 3a: Assume tables R(a,b), and S(c,d,e), draw a logical query tree in relational algebra for the following query: (5 points)**

```
Q7: SELECT R.a, S.d FROM R, S
      WHERE R.b = S.c AND S.e < 3
    UNION ALL
    SELECT R.a, S.d FROM R, S
      WHERE R.b = S.c AND S.e = 7
```

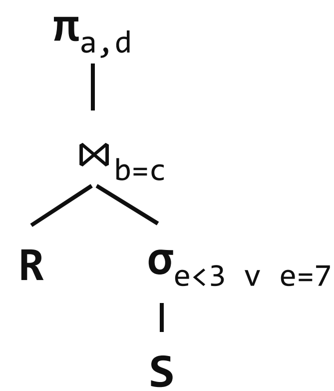


- **Task 3b: Draw an optimized logical query tree for the above query in relational algebra by eliminating the union operation (3 points)**

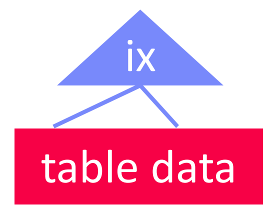


#3 Query Processing, cont.

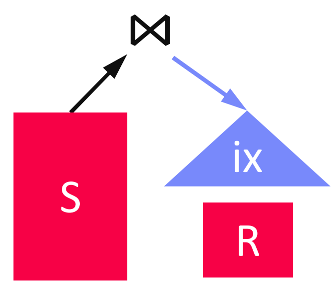
- **Task 3c: Given the schema and query above, which attribute or attributes are good candidates for secondary indexes and how could they be exploited during query processing? (4 points)**



- **Solution**
  - S.e → index scan
    (lookup e=7,
    lookup e=3 and scan DESC)



  - R.b (or S.c) → index nested loop join
    (for every S tuple s, loopup s.c in IX)



#3 Query Processing, cont.

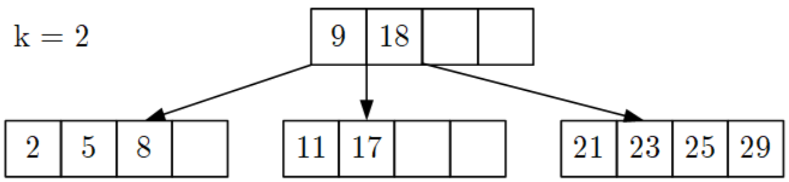
- **Task 3d: Describe the volcano (open-next-close) iterator model by example of a selection operator and discuss the space complexity of this selection operator.** (6 points)

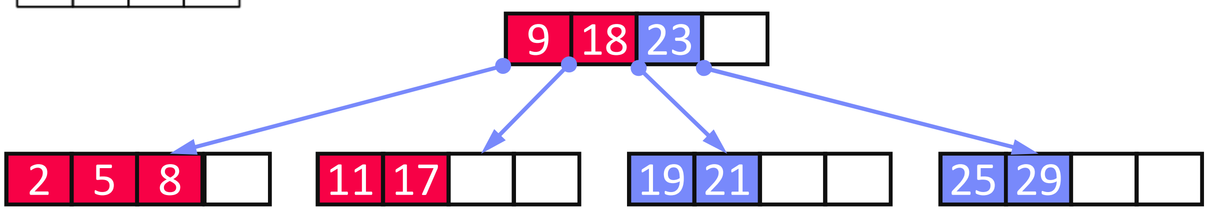
- **Solution**
  - Open, next, close calls propagate from root to leafs
  - **Open:** operator initialization
  - **Next:** compute next tuple (selection: call next of input until next qualifying tuple found)
  - **Close:** cleanup resources
  - **Space complexity:** O(1)

```
void open() { R.open(); }

void close() { R.close(); }

Record next() {
  while( (r = R.next()) != EOF )
    if( p(r) ) //A==7
      return r;
  return EOF;
}
```

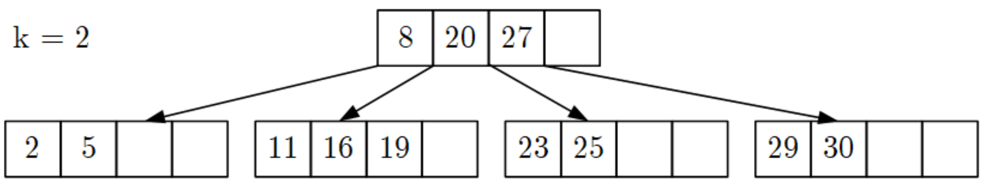
#4 Physical Design – B-Trees

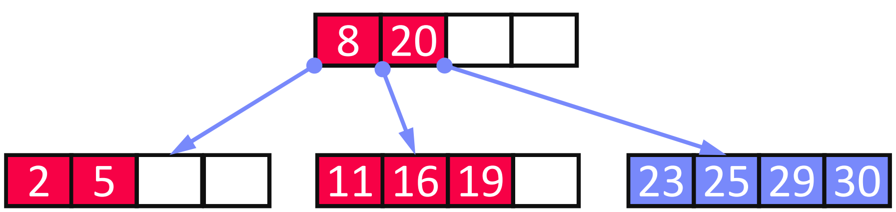
- **Task 4a: Given B-tree, insert key 19 and draw resulting B-tree (7 points)**





- **Task 4b: Given B-tree, delete key 27, and draw resulting B-tree (8 points)**

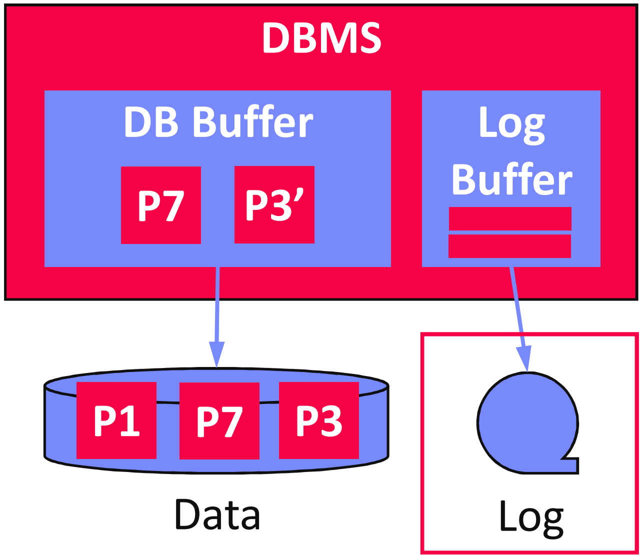




#5 Transaction Processing

- **Task 5a: Describe the concept of a database transaction log, and explain how it relates to the ACID properties Atomicity and Durability** (7 points)

- **Solution**
  - Log: append-only TX changes, often on separate devices
  - Write-ahead logging (log written before DB, forced-log on commit)
  - Recovery: forward (REDO) and backward (UNDO) processing

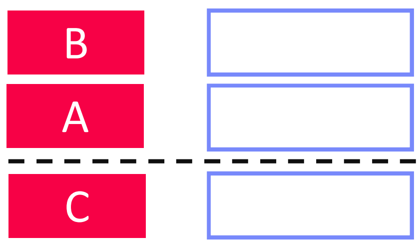


  - **#1 Atomicity:** A TX is executed atomically (completely or not at all); on failure/aborts no changes in DB (**UNDO**)
  - **#2 Durability: Guaranteed persistence** of changes of successful TXs; in case of system failures, the database is recoverable (**REDO**)

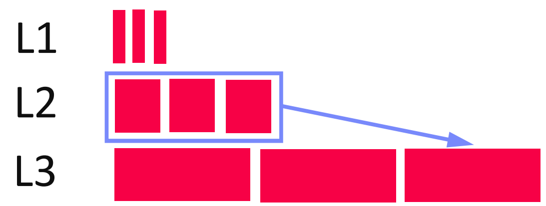
#6 NoSQL

- **Task 6a:** Describe the concept and system architecture of a **key-value store**, including techniques for achieving **high write throughput**, and **scale-out** in distributed environments. Please focus specifically on aspects of physical design such as **index structures**, and **distributed data storage**. (10 points)

- **Solution**
  - KV store: simple map of key-value pairs, w/ get/put interface, often distributed



  - Index structure for high write throughput: Log-structured merge trees (LSM)



  - Distributed data storage for scale-out:
    horizontal partitioning (sharding) via hash or range partitioning,
    partitioning via selection, reconstruction via union
    eventual consistency for high availability and partition tolerance

Conclusions and Q&A

- **12 Stream Processing Systems**
- **Q&A and Exam Preparation**

- **Misc**
  - Last Office Hour: **June 15, 1pm**
  - Exercise 4 Reminder: **June 16, 11.59pm** + [7+3 late days]

- **Exams**
  - **June 22: 11am-1pm**, **2pm-4pm**, **5pm-7pm** (i7, i11, i12, and i13)
  - **Oral exams** for international students (so far 5, to be scheduled)
  - 2nd Exam: July 1, 2pm (i7, i11, i12, and i13)
  - 3rd Exam: July 29, 2pm (i7, i11, i12, and i13)