Dr. Thorsten Papenbrock
Information Systems Group
Hasso Plattner Institute

July 26, 2021

Distributed Data Management – Exam

Summer Term 2021 **Musterlösung:**

Musterlösung

Matriculation Number: ______________________

0	1	2	3	4	5	6	7	8	9	10	Σ
1	6	6	4	13	6	8	5	7	9	5	70

Important Rules:

- The exam must be solved within **180 minutes** (13:00 – 16:00).
- Fill in your matriculation number (Matrikelnummer) above and **on every page**.
- Answers can be given in English or German.
- Any usage of external resources (such as scripts, prepared pages, electronic devices, or books) is not permitted.
- Make all **calculations transparent and reproducible**!
- Use the **free space under each task for your answers**. If you need more space, continue on the **back of the page**. Use the extra pages at the end of the exam only if necessary. **Provide a pointer to an extra page** if it should be considered for grading. The main purpose of the extra pages is for drafting.
- Please write clearly. **Do not use the color red or pencils**.
- If you have any questions, raise your hand.
- The exam consists of 28 pages including cover page and extra pages.
- For any multiple choice question, more or fewer than one answer might be correct.
- Good luck!

Task 0: Matriculation Number

Fill in your matriculation number (Matrikelnummer) on every page including the cover page and the extra pages (even if you don't use them).

Hint: Do it now! **1 point**

Task 1: Distributed Systems

1. Which of the following statements about distributed systems are true? Tick all true statements. **4 points**

   ☐ A distributed system is a group of independent compute nodes that communicate and collaborate to solve a common task.

   ☐ A reliable distributed system is one that expects faults and errors to happen, but prevents that these events escalate into failures.

   ☐ If a distributed system uses data parallelism, it needs to replicate data to different nodes.

   ☐ A distributed system that uses quorum-based leaderless replication has to compromise on CAP's "C" if failures occur, but it is therefore 100% "A" and "P".

   Musterlösung:

   ☒ That is exactly the definition of a distributed system.
   ☒ That is exactly the definition of reliability in distributed systems.
   ☐ Data parallelism implies data partitioning but not (necessarily) data replication.
   ☐ Leaderless replication might be more on the A side than on the C side, but it also drops A if quorums cannot be fulfilled, i.e., leaderless replicated systems are not 100% A. One can put the r- or w-part of the quorum to 1, but not both.

   Grading: 1P for each correctly ticked or non-ticked field

2. Alice was given the homework to write an algorithm that should achieve a speedup of at least 4 on 5 nodes. She wrote a 95% parallel solution and hands the homework in hoping that it will be sufficient. Considering Amdahl's law and assuming the problem size did not change with the distribution (and ignoring communication costs), what is the speedup that Alice will actually get? **2 points**

   Musterlösung:

   Solution and Grading:

- 1P Amdahls law: speedup(s) = 1/((1-p)+p/s)
- 1P correct calculation: speedup(s) = 1/((1-0.95)+0.95/5) = 4.16666

Task 2: Encoding

1. Encoding is the process of changing the representation of digital information, which is needed for remote communication. In the lecture, we discussed four data model layers: *representation*, *physical*, *conceptual*, and *logical*. Bring the layers in the correct order and assign each of the following concepts to the layer it belongs to: *XML documents*, *byte sequences*, *code objects*, *magnetic fields*, *hash maps*, *relational tables*, and *pulses of light*. **3 points**

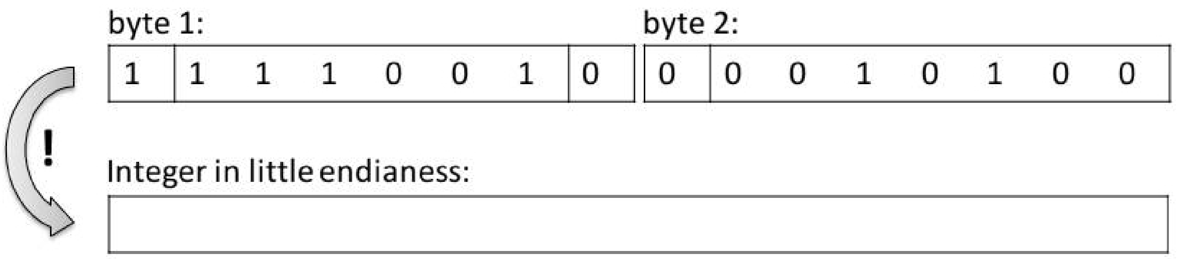
   Musterlösung:

   conceptual: *code objects*, *hash maps*
   logical: *XML documents*, *relational tables*
   representation: *byte sequences*
   physical: *magnetic fields*, *pulses of light*

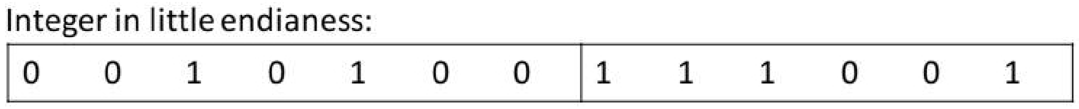
   Grading:

   - 1P for correct order of layer labels.
   - 0.5P for each layer with correct elements.

2. Variable-length integers are a central trick used in different binary encoding formats. Consider the following variable-length integer bit sequence. What bit sequence does it represent? **2 points**



   Musterlösung:



   Grading:

- 1P ignoring the prefix bits
- 1P inverting the byte-order

3. The binary encoding format Avro uses two types of schemata: the `readers schema` and the `writers schema`. When reading data, Avro performs on-the-fly schema mapping. By what criterion are the attributes in the two schemata matched?
1 points

Musterlösung:

Grading:

- 1P The attributes are matched by name (or attribute ID).

Task 3: Communication

1. The routing of messages is very important in message switched networks. Which layer of the OSI Model performs the routing? **1 points**

   Musterlösung:

   Grading:

   - 1P The **Network** layer (or: **layer 3**) performs the message routing.

2. Communication protocols have certain characteristic properties. How do we call a protocol with the following properties? **3 points**

   - If the receiver maintains some pre-allocated memory space for incoming messages, then the protocol is called:

   - If `send()` calls return directly, which is before the data is copied out of the local message structure, and `receive()` calls also return directly either with a message or status code, then the protocol is called:

   - If messages are copied directly from sender to receiver address space, for which both processes need to be active at the same time, then the protocol is called:

   Musterlösung:

   Grading:

   - 1P Buffering
   - 1P Asynchronous (or non-blocking)
   - 1P Transient

Task 4: Actor Programming

1. The Actor model uses *actors* as the universal primitive of concurrent programming. What three properties define an actor and how do actors communicate? **2 points**

   Musterlösung:

   0.5P state
   0.5P behavior
   0.5P mailbox
   0.5P (asynchronous, fire-and-forget) message passing

2. Which of the following statements on the actor model and actor programming are *true*? Tick the true ones. **3 points**

   ☐ An *actor system* in Akka organizes all created actor objects in a single hierarchy; for execution, the ActorSystem schedules these actor objects dynamically on threads.

   ☐ The *proxy pattern* helps an actor to, for instance, externalize functionality, handle resource intensive actions, or protect itself from other actors.

   ☐ The *cameo pattern* serves to encapsulate waiting behavior in a proxy that collects partial results from (potentially multiple) other actors.

   Musterlösung:

   ☒
   ☒
   ☒

   Grading: 1P for each correctly ticked or non-ticked field

3. A reoccurring problem in actor programming is that actors overload other actors with too many messages so that their mailboxes overflow. Given a `Producer` actor that needs to send lots of work messages to a `Consumer` actor, how can this problem be avoided? **1 points**

   Musterlösung:

   1P **work pulling**, i.e., **pull propagation**; or a protocol description that effectively is work pulling (example: producer buffers messages locally and sends out only a few messages; once these are acknowledged, it sends further messages; producer either buffers messages or produces them on-demand; producer sends individual messages or uses batching).

   Also ok are descriptions involving backpressure, proxy, or active waiting.

4. Another reoccurring problem in actor programming is that actors become unresponsive if they need to work on long running tasks. How can a `Worker` actor stay responsive and avoid extensively long thread-blocking even if it needs to work on a long task? Assume that the task cannot be broken down and/or offloaded to other actors. **2 points**

   Musterlösung:

   stop the work frequently, send a message to self to trigger continuing the work, consume messages from queue

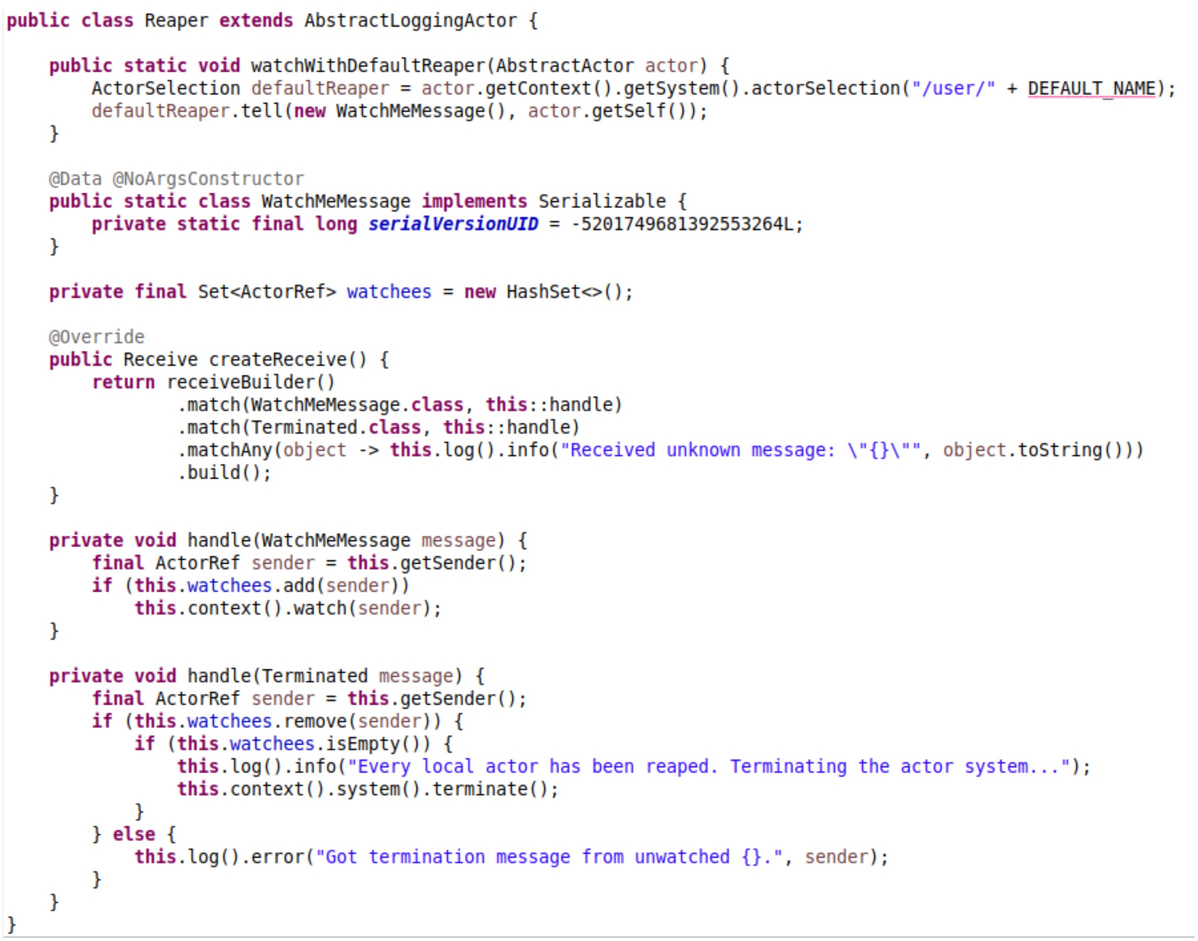
   Grading: 2P for this answer; 1P for other ideas that do not work perfectly; 0P for no answer or an answer that really does not solve the issue

   The use of a proxy to buffer incoming messages technically extends the message queue but does not solve the problem, as the worker is still unresponsive. But this answer still grands 0.5P, because it solves the mailbox overflows.

   Suggesting data parallelism might (in the specification of this task) solve the problem in some situations, but it is no general solution. Hence, also 0.5P

5. Shutting a distributed system down is a challenge, because some entity in each system needs to figure out that all work has been done. The *reaper pattern* in Akka serves exactly this purpose. Use the space on this sheet to implement a `Reaper` actor that watches (`this.context().watch(<ActorRef>)`) the actors of its ActorSystem and terminates (`this.context().system().terminate()`) the system when all actors are gone. **5 points**

Musterlösung:



```java
public class Reaper extends AbstractLoggingActor {

    public static void watchWithDefaultReaper(AbstractActor actor) {
        ActorSelection defaultReaper = actor.getContext().getSystem().actorSelection("/user/" + DEFAULT_NAME);
        defaultReaper.tell(new WatchMeMessage(), actor.getSelf());
    }

    @Data @NoArgsConstructor
    public static class WatchMeMessage implements Serializable {
        private static final long serialVersionUID = -5201749681392553264L;
    }

    private final Set<ActorRef> watchees = new HashSet<>();

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(WatchMeMessage.class, this::handle)
                .match(Terminated.class, this::handle)
                .matchAny(object -> this.log().info("Received unknown message: \"{}\"", object.toString()))
                .build();
    }

    private void handle(WatchMeMessage message) {
        final ActorRef sender = this.getSender();
        if (this.watchees.add(sender))
            this.context().watch(sender);
    }

    private void handle(Terminated message) {
        final ActorRef sender = this.getSender();
        if (this.watchees.remove(sender)) {
            if (this.watchees.isEmpty()) {
                this.log().info("Every local actor has been reaped. Terminating the actor system...");
                this.context().system().terminate();
            }
        } else {
            this.log().error("Got termination message from unwatched {}.", sender);
        }
    }
}
```

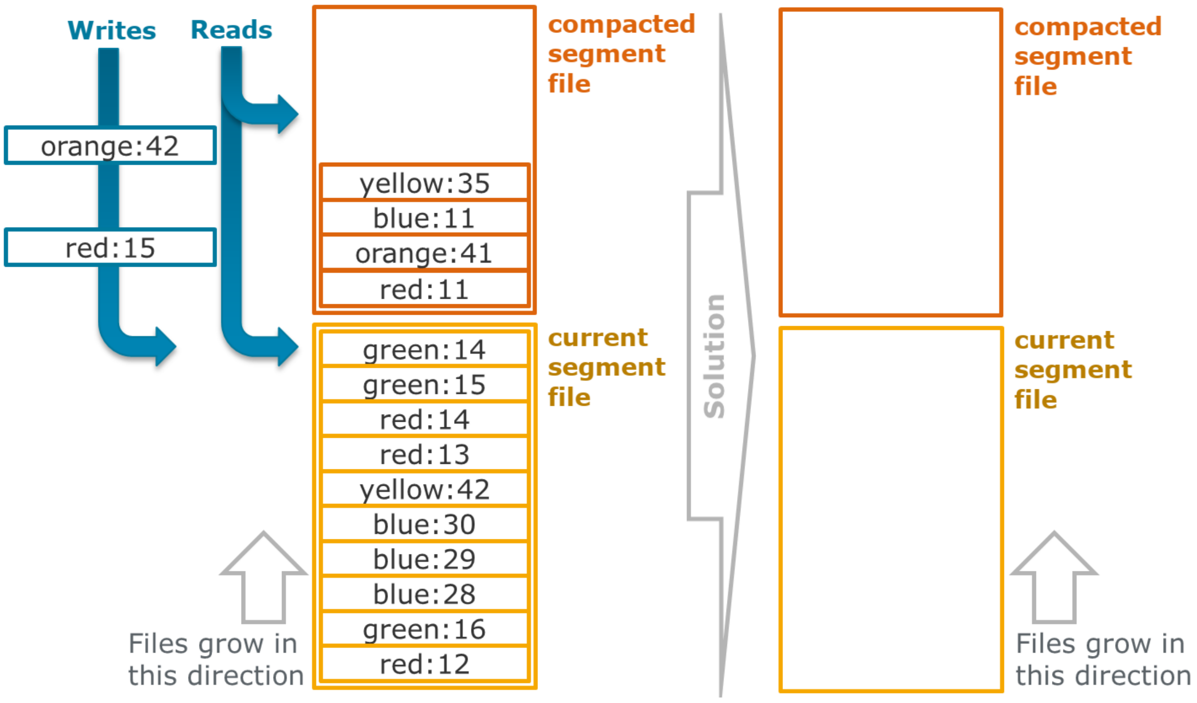
Grading:

- 1P Reaper actor class
- 1P extending some Actor superclass
- 0.5P matching on some WatchMeMessage
- 0.5P matching on Terminated message

- 1P storing watchees
- 1P calling terminate when list is empty
- By specifying the watchWithDefaultReaper()-function, one can get an extra point to compensate a minuspoint, but we do not grand more than 5 points here!

Task 5: Storage and Retrieval

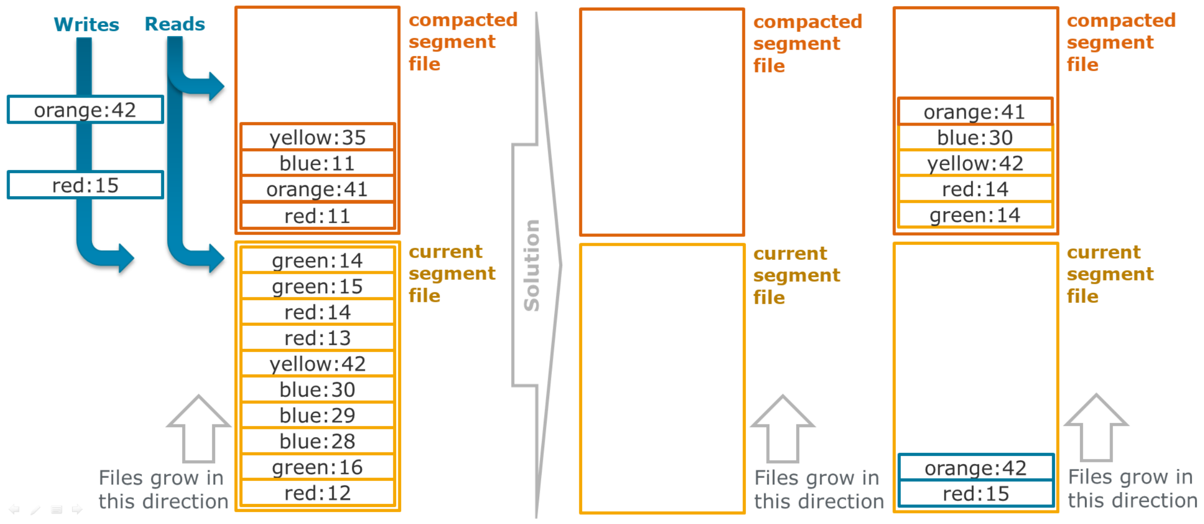
1. Consider the following *current segment file* and *compacted segment file* as well as the two entries that are currently in the process of being written. Because the current segment file is full, the two new entries do not fit into the current segment any more and we need to trigger a merge-and-compact operation. Write down were all the entries end up after the this operation by listing them in the correct order in the correct files. Use the two files on the right for your solution. **3 points**



Musterlösung:

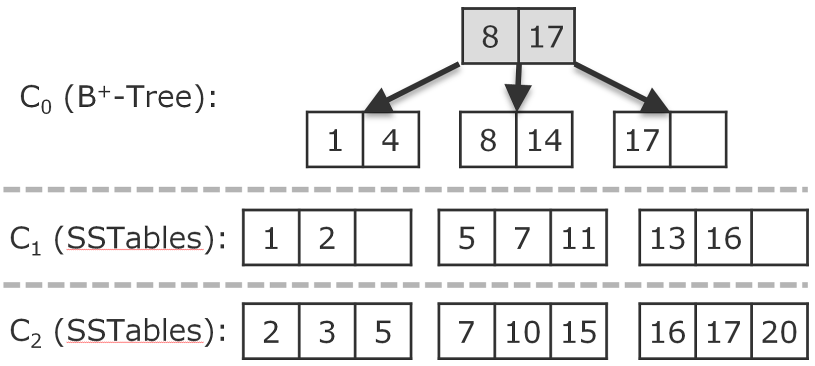
Grading: (idea)

- 1P if only the new entries (orange:42 and red:15) are in the current segment file / 0.5P if entries are reversed
- 1P if the order is correct
- 1P if entries in the compacted segment file are correct (keys and values)
- if the element values have been used for determining the newest entries (i.e., green:16 is in the result and not green 14), then -1P

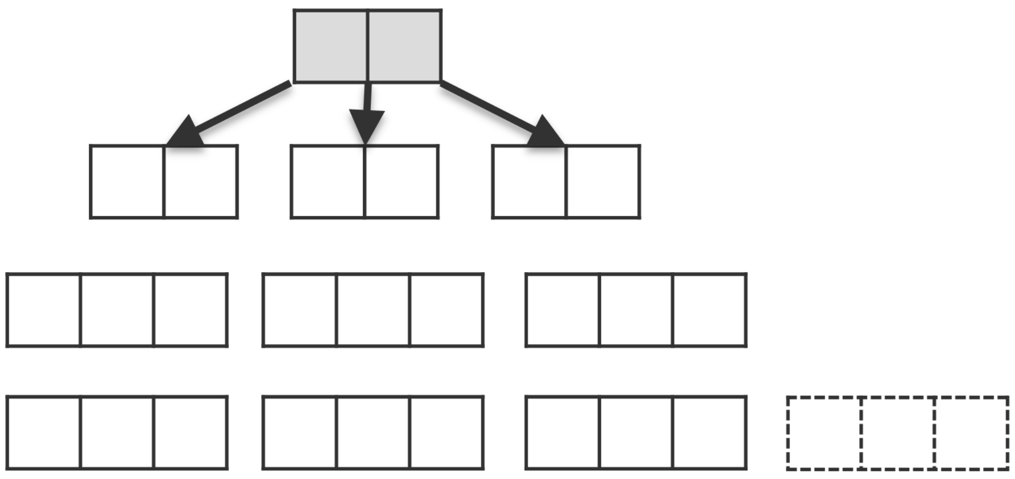


- if the element order was reversed and the solution assumes that files grew the other way around, then -0.5P if the solution is correct w.r.t. that ordering assumption)

2. The following figure depicts an LSM tree instances with three levels, which are C_0, C_1, and C_2. In the LSM tree, we show only the keys and not their values. Assume that all values have the same size, so that a B^+-tree leaf can hold exactly up to two key-value pairs and each SSTable can hold exactly up to three key-value pairs. The maximum depth of the B^+-tree shall be two, which means that the depicted tree cannot grow any further; the number of SSTables in C_1 is fix, but we can, if necessary, add SSTables in C_2.



We now insert a new element with key **9** into the LSM tree. Write the result of this insert operation into the template below. **3 points**

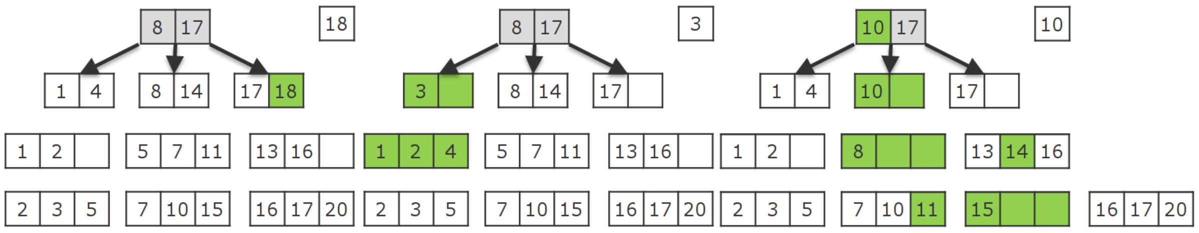


Musterlösung:

Same as for 10, but with 9 instead of ten in the B+-Tree

Grading:

- 1P for each correct layer



Task 6: Replication and Partitioning

1. Bob operates a leaderless replicated, distributed database on a cluster with 256 nodes. The database uses quorum reads and writes to ensure consistency and a gossip protocol to ensure that all updates will eventually spread to all nodes in the cluster. We assume that the gossip protocol has a perfect spreading technique, i.e., newly written data is always gossiped to nodes that do not already know it.

   What read (r) and write (w) values does Bob need to define for the database's quorum q(r,w) if writes should return as fast as possible (i.e., possibly small w) and every successfully written value should reach all cluster nodes in not more than 3 rounds of (perfect) gossiping? **4 points**

   Musterlösung:

   with 0 gossip rounds: 256
   with 1 gossip rounds: 256 / 2 = 128
   with 2 gossip rounds: 256 / 2 / 2 = 64
   with 3 gossip rounds: 256 / 2 / 2 / 2 = 32

   in general:
   $w * 2^{rounds} >= 256$
   $w >= 256/2^{rounds} = 256/2^3 = 256/8 = 32$

   consistency: r + w > n
   r > n - w = 256 - 32 = 224

   answer: q(225,32)

   Grading:

   - 1P w = 32 correctly calculated
   - 1P r = 225 correctly calculated
   - 1P gossip protocol correctly understood
   - 1P quorum consistency condition r + w > n

2. Consider a quorum q(r,w) with q(100,101) that was used in a leaderless replicated database system on a cluster with 200 nodes. Answer the four questions in the following matrix w.r.t. that quorum and briefly explain your answer: **4 points**

	... **adding** a node to the cluster?	... **removing** a node from the cluster?
Do **reads** still work consistently when ...		
Do **writes** still work consistently when ...		

Musterlösung:

Grading 0.5P correct answer and 0.5P correct explanation:

- 1P reads + adding: no, because a value might now be unknown to 99+1 nodes, which is exactly the r in the quorum.

- 1P reads + removing: yes, because at least 100 out of 199 nodes still know every successfully written value, so that reading 100 will still find any value.
- 1P writes + adding: no, because r+w>n does not hold any more, i.e., a value written to 101 out of 201 nodes is not guaranteed to be found with a read of 100 nodes.
- 1P writes + removing: yes, because the quorum still holds.

Task 7: Distributed Systems

1. The *Network Time Protocol* (NTP) can be used to calculate the time offset between a time server's reference clock and any client's local clock. Assume the round-trip time message has recorded the server times `16:42:00` and `16:42:02` and the client times `16:43:08` and `16:44:12` in the time format `hh:mm:ss`. What time offset would the NTP protocol calculate on the client w.r.t. this server? **2 points**

   Musterlösung:

   t0 = 16:43:08
   t1 = 16:42:00
   t2 = 16:42:02
   t3 = 16:44:12

   offset = ((t1-t0)+(t2-t3))/2

   offset = (16:42:00 - 16:43:08 + 16:42:02 - 16:44:12)/2
   = (- 68 - 130)/2 = - 198/2 = -99

   Grading:

   - 1P correct formula
   - 0.5P correct value interpretation
   - 0.5P correct calculation

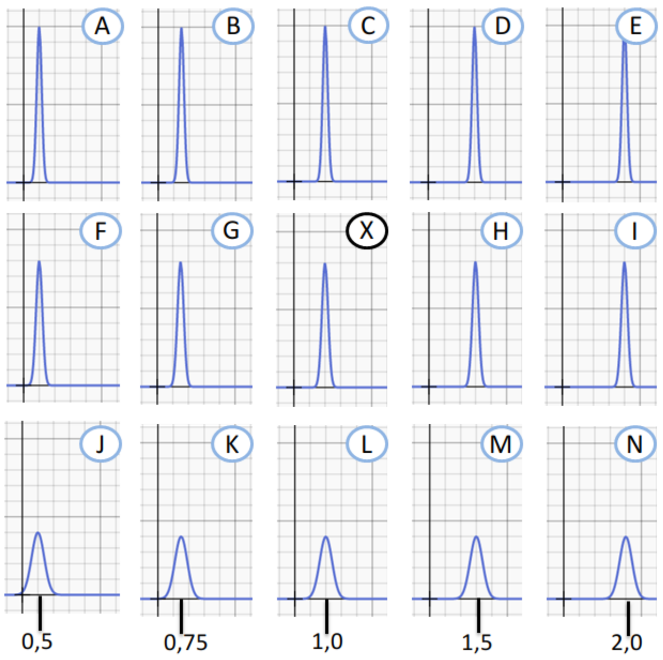
2. The ϕ accrual failure detector method uses the probabilistic density curve over the heartbeats of a process to calculate a continuous suspicion value $\phi(t)$. We now focus on this probabilistic density estimation. Assume a distribution as shown in chart (X) below. We now accelerate the heartbeats from 1 beat/sec to 2 beat/sec. What curve shape can we expect when the sampling window contains (for a short while) both 1 and 2 beat/sec heartbeats and what will the curve look like with only 2 beat/sec heartbeats? Enter the according chart labels in the solution sequence next to the charts! **3 points**

   Musterlösung:

   Correct answer: X -> K -> F

   Grading:

   - 1P middle is in J, K, L, M, N
   - 1P end is either F, J, A or I



Solution:

X → _____ → _____

- 1P K and F is exactly right

Task 8: Consistency and Consensus

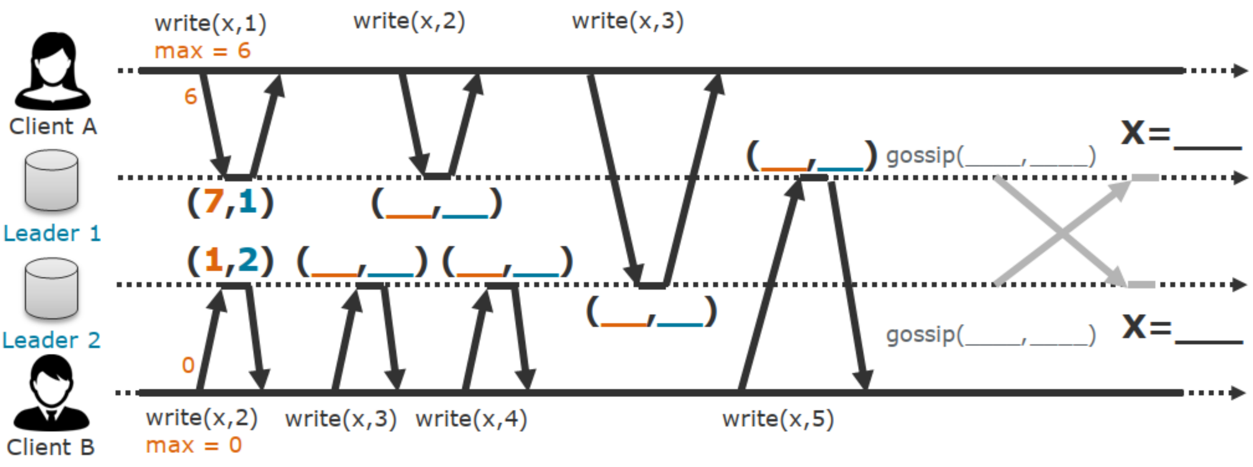
1. Linearizability is a consistency guarantee of eventual consistent databases stating that a read operation should always return the most recent value of an object although replicas might have older values. What three techniques are needed to make a field (or the entire database) linearizable in a leaderless replicated setup? Name the three techniques and very briefly note down, what each technique does (one or two sentences per technique). **3 points**

   Musterlösung:

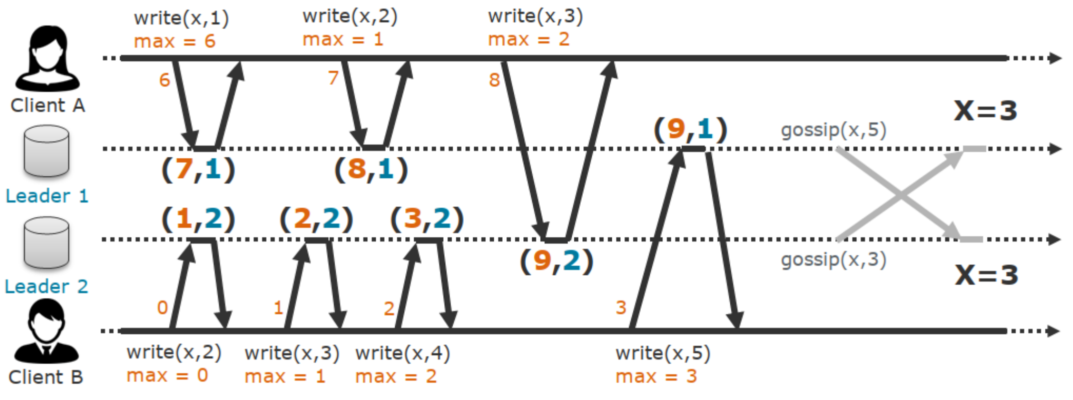
   Grading and solution:

   - 1P **Quorum reads and writes**: Quorum reads and writes specify the number of positive read- and write-responses needed for successfully written values. (They ensure the basic consistency for the leaderless replicated system.)
   - 1P **Read-repair**: Upon reading an outdated value, overwrite it with the known new value before returning the result of the read to the user.
   - 1P **Read before write**: Always issue a read before any write (helping any prior value to spread before it gets overwritten).

2. Given a database management system that implements *Lamport Timestamps* for causal write ordering and a gossip protocol for eventual consistency. A Lamport Timestamp is a pair (c, i) with a write counter c and a node identifier i. Every write operation is associated with such a timestamp. A gossip call $gossip(f, v)$ sends the value v for field f to another node; (f, v) implicitly carries the current Lamport Timestamp (c, i). Add the Lamport Timestamps and gossip values in the following example. What is the final value of field `x` on leader 1 and on leader 2? **4 points**



Musterlösung:



We can use these timestamps for write ordering, because lamport timestamps are comparable: $(c, i) > (c', i')$ iff $(c > c') \lor (c = c' \land i > i')$.

Grading:

- 1P first three writes correct
- 1P cross-writes correct
- 1P correct gossip (field and value correct w.r.t. last writes; not the timestamps in the call)
- 1P correct result (correct gossip w.r.t. last fields and timestamps)

Task 9: Batch Processing

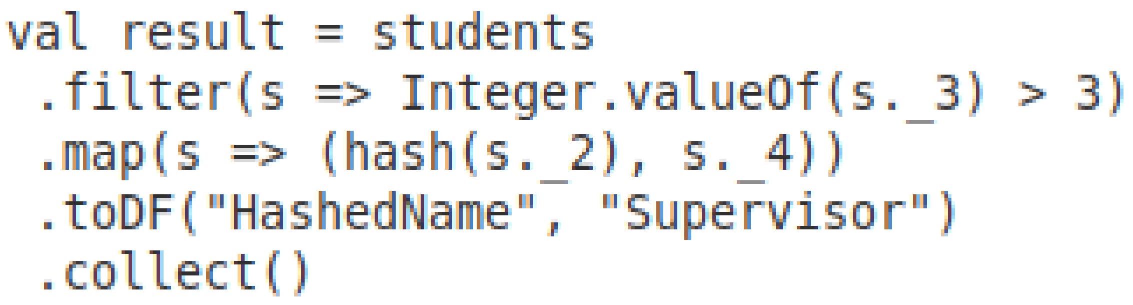
Suppose you are given three datasets: A `students` dataset that contains general information about students, a `courses` dataset that describes available courses for the students, and an `enrollment` dataset, which is basically a join table between `students` and their `courses`. The following code snippets read the three datasets into Spark `Datasets`:

```
val students = spark
  .read
  .option("quote", "\"")
  .option("delimiter", ",")
  .csv(s"data/students.csv")
  .toDF("ID", "Name", "Semester", "Supervisor")
  .as[(String, String, String, String)]

val enrollments = spark
  .read
  .option("quote", "\"")
  .option("delimiter", ",")
  .csv(s"data/enrollments.csv")
  .toDF("StudentID", "CourseID", "Credits")
  .as[(String, String, String)]

val courses = spark
  .read
  .option("quote", "\"")
  .option("delimiter", ",")
  .csv(s"data/courses.csv")
  .toDF("ID", "Title", "Teacher", "Topic")
  .as[(String, String, String, String)]
```

Use the three `Datasets` to solve the following tasks. You may use Spark's *Dataset* and/or *DataFrame* API but no SQL! Also have a look at the *Dataset* API documentation at the end of this task. If you are not sure about how a particular interface, call, or class works, make a good guess and provide a comment on how you *think* it works.

1. For administrative purposes, we need a list of all `students` that did not finish their studies in 10 semesters (value of `Semester` > 10) with their supervisors. Because this is very sensitive information, all student names need to be anonymized via `hash()` call on their `Name` values. The result should have the schema (`HashedName`, `Supervisor`) and be copied into a local variable in the driver program. Write a transformation pipeline that solves the request. **4 points**

   Musterlösung:



```
val result = students
  .filter(s => Integer.valueOf(s._3) > 3)
  .map(s => (hash(s._2), s._4))
  .toDF("HashedName", "Supervisor")
  .collect()
```

   Grading:

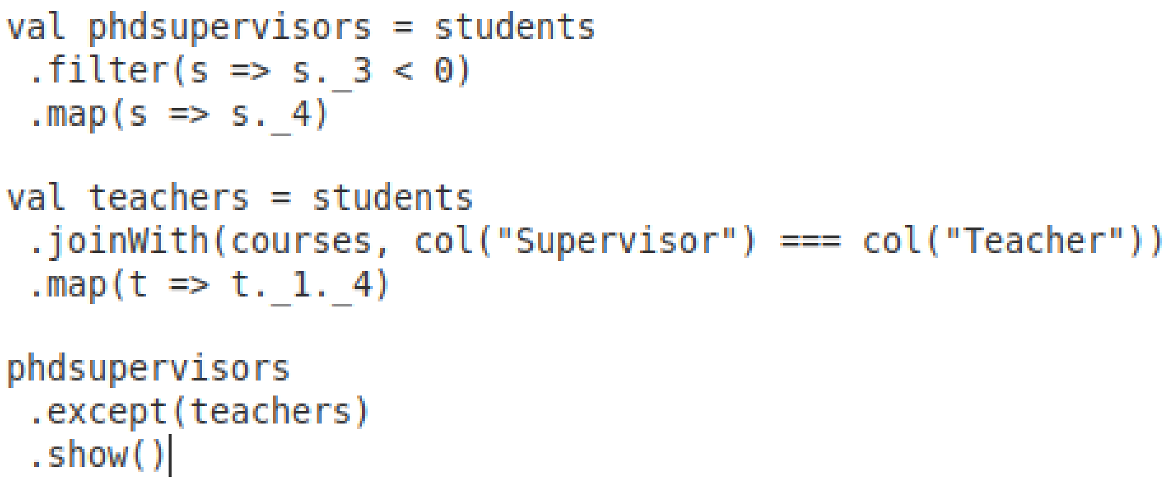
   - (>3 in the screenshot is a >10)
   - 1P filter; missing the integer conversion is ok
   - 1P map: 0.5P for hashing, 0.5P for projecting on 2 and 4 (1 and 3 is also ok, if students assumed 0-indexing)
   - 1P toDF renaming
   - 1P collect to val or var variable

2. The following SQL query searches for supervisors that supervise PhD students (`Semester` < 0), but never taught a course. Translate the SQL query into a Spark transformation pipeline that writes the result to the console of the driver. **5 points**

```
(SELECT Supervisor
 FROM students
 WHERE Semester < 0)
except
(SELECT Supervisor
 FROM students, courses
 WHERE Supervisor = Teacher)
```

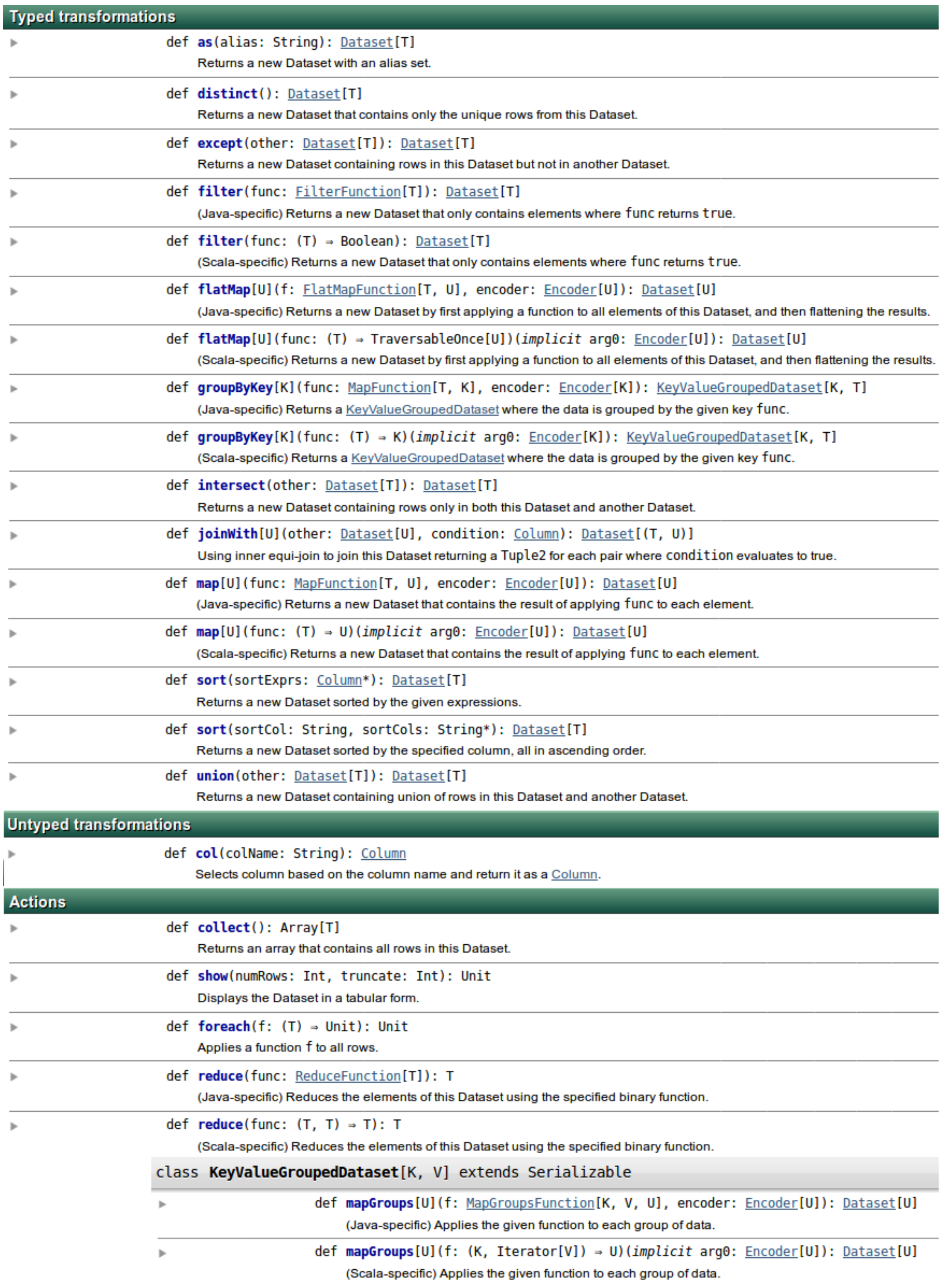
   Musterlösung:

   Grading:



```
val phdsupervisors = students
 .filter(s => s._3 < 0)
 .map(s => s._4)

val teachers = students
 .joinWith(courses, col("Supervisor") === col("Teacher"))
 .map(t => t._1._4)

phdsupervisors
 .except(teachers)
 .show()
```

- (both RDDs need to a .distinct() call, because EXCEPT in SQL is a set operation, but since we did not talk about set-semantics in the lecture, both having and not having distincts is fine)
- 1P filter
- (1P join is technically not needed; hence, solutions without it are also correct although they are no 1-to-1 translations)
- 1P bringing both relations to the same schema
- 1P except
- 1P show or print as action



Typed transformations

`def as(alias: String): Dataset[T]`
Returns a new Dataset with an alias set.

`def distinct(): Dataset[T]`
Returns a new Dataset that contains only the unique rows from this Dataset.

`def except(other: Dataset[T]): Dataset[T]`
Returns a new Dataset containing rows in this Dataset but not in another Dataset.

`def filter(func: FilterFunction[T]): Dataset[T]`
(Java-specific) Returns a new Dataset that only contains elements where `func` returns `true`.

`def filter(func: (T) ⇒ Boolean): Dataset[T]`
(Scala-specific) Returns a new Dataset that only contains elements where `func` returns `true`.

`def flatMap[U](f: FlatMapFunction[T, U], encoder: Encoder[U]): Dataset[U]`
(Java-specific) Returns a new Dataset by first applying a function to all elements of this Dataset, and then flattening the results.

`def flatMap[U](func: (T) ⇒ TraversableOnce[U])(implicit arg0: Encoder[U]): Dataset[U]`
(Scala-specific) Returns a new Dataset by first applying a function to all elements of this Dataset, and then flattening the results.

`def groupByKey[K](func: MapFunction[T, K], encoder: Encoder[K]): KeyValueGroupedDataset[K, T]`
(Java-specific) Returns a KeyValueGroupedDataset where the data is grouped by the given key `func`.

`def groupByKey[K](func: (T) ⇒ K)(implicit arg0: Encoder[K]): KeyValueGroupedDataset[K, T]`
(Scala-specific) Returns a KeyValueGroupedDataset where the data is grouped by the given key `func`.

`def intersect(other: Dataset[T]): Dataset[T]`
Returns a new Dataset containing rows only in both this Dataset and another Dataset.

`def joinWith[U](other: Dataset[U], condition: Column): Dataset[(T, U)]`
Using inner equi-join to join this Dataset returning a `Tuple2` for each pair where `condition` evaluates to true.

`def map[U](func: MapFunction[T, U], encoder: Encoder[U]): Dataset[U]`
(Java-specific) Returns a new Dataset that contains the result of applying `func` to each element.

`def map[U](func: (T) ⇒ U)(implicit arg0: Encoder[U]): Dataset[U]`
(Scala-specific) Returns a new Dataset that contains the result of applying `func` to each element.

`def sort(sortExprs: Column*): Dataset[T]`
Returns a new Dataset sorted by the given expressions.

`def sort(sortCol: String, sortCols: String*): Dataset[T]`
Returns a new Dataset sorted by the specified column, all in ascending order.

`def union(other: Dataset[T]): Dataset[T]`
Returns a new Dataset containing union of rows in this Dataset and another Dataset.

Untyped transformations

`def col(colName: String): Column`
Selects column based on the column name and return it as a Column.

Actions

`def collect(): Array[T]`
Returns an array that contains all rows in this Dataset.

`def show(numRows: Int, truncate: Int): Unit`
Displays the Dataset in a tabular form.

`def foreach(f: (T) ⇒ Unit): Unit`
Applies a function `f` to all rows.

`def reduce(func: ReduceFunction[T]): T`
(Java-specific) Reduces the elements of this Dataset using the specified binary function.

`def reduce(func: (T, T) ⇒ T): T`
(Scala-specific) Reduces the elements of this Dataset using the specified binary function.

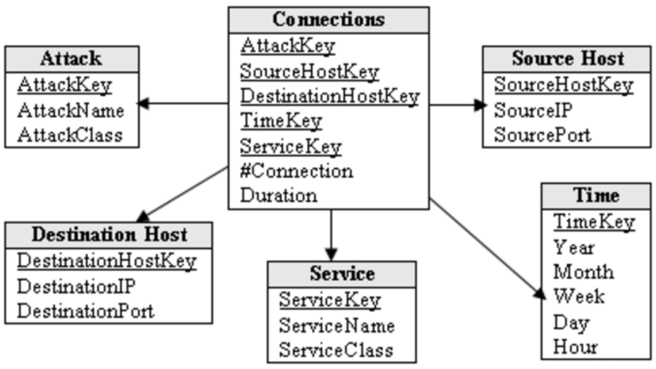
`class KeyValueGroupedDataset[K, V] extends Serializable`

`def mapGroups[U](f: MapGroupsFunction[K, V, U], encoder: Encoder[U]): Dataset[U]`
(Java-specific) Applies the given function to each group of data.

`def mapGroups[U](f: (K, Iterator[V]) ⇒ U)(implicit arg0: Encoder[U]): Dataset[U]`
(Scala-specific) Applies the given function to each group of data.

Task 10: Distributed Database Management Systems

1. Given the following data warehouse *star schema*, relation cardinalities and join query, which join algorithm, i.e., normal join or star join has the smaller intermediate results? Calculate the sizes for the two algorithms and, then, state the superior algorithm in this setting! **3 points**



$$((Connections \bowtie Attack) \bowtie Service) \bowtie Time$$

Connections: 10 000 tuples
Attack: 100 tuples
Service: 100 tuples
Time: 100 tuples

Musterlösung:

Intermediate normal join costs:
C+A: 10 000
C+A+S: 10 000
–> **20 000**

Intermediate star join costs:
A+S: 100 * 100 = 10 000
A+S+T: 10 000 * 100 = 1 000 000
–> **1 010 000**

The normal join works better, because the fact table is not that big, the dimension tables are relatively large and there are no filters in the query that might reduce the fact tables sizes.

Grading:

- 1P correct star join result
- 1P correct normal join result
- 1P correct statement (maybe the calculations are missing but the interpretation is still correct)
- If the intermediate size calculation also include the table reading costs (3x 100) and final result size (10 000), then that is also fine (it is the same for both variants).

2. Given the bitmap [1000000000010000]. Does it make sense to apply run-length encoding to this bitset before sending it over the network or not? Justify your answer! **2 points**

   Musterlösung:

   Grading and solution:

   - 1P answer: Does not make sense!
   - 1P justification: 2 bytes bitmap vs. 4 integers run-lengths encoding

Extra page 1

Extra page 2