

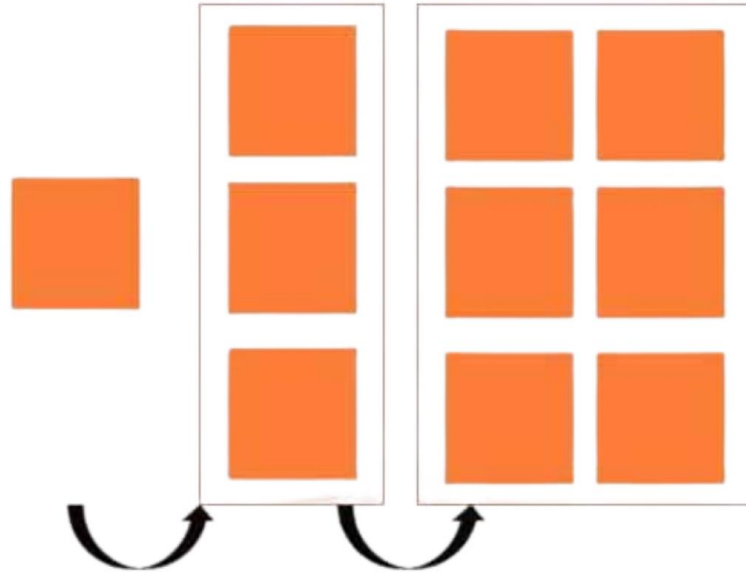
# Realtime Big Data ist doch keine Kunst - wirklich?

Ursula Deriu / [jugs.ch](http://jugs.ch)

## Definition Big Data

Der Begriff '**Big Data**' bezeichnet Methoden zur Speicherung, Abfrage, Verarbeitung und Analyse von Daten mit Hilfe von **verteilten und horizontal skalierbaren Systemen**.

# Horizontal skalierbar



# Big Data RZ



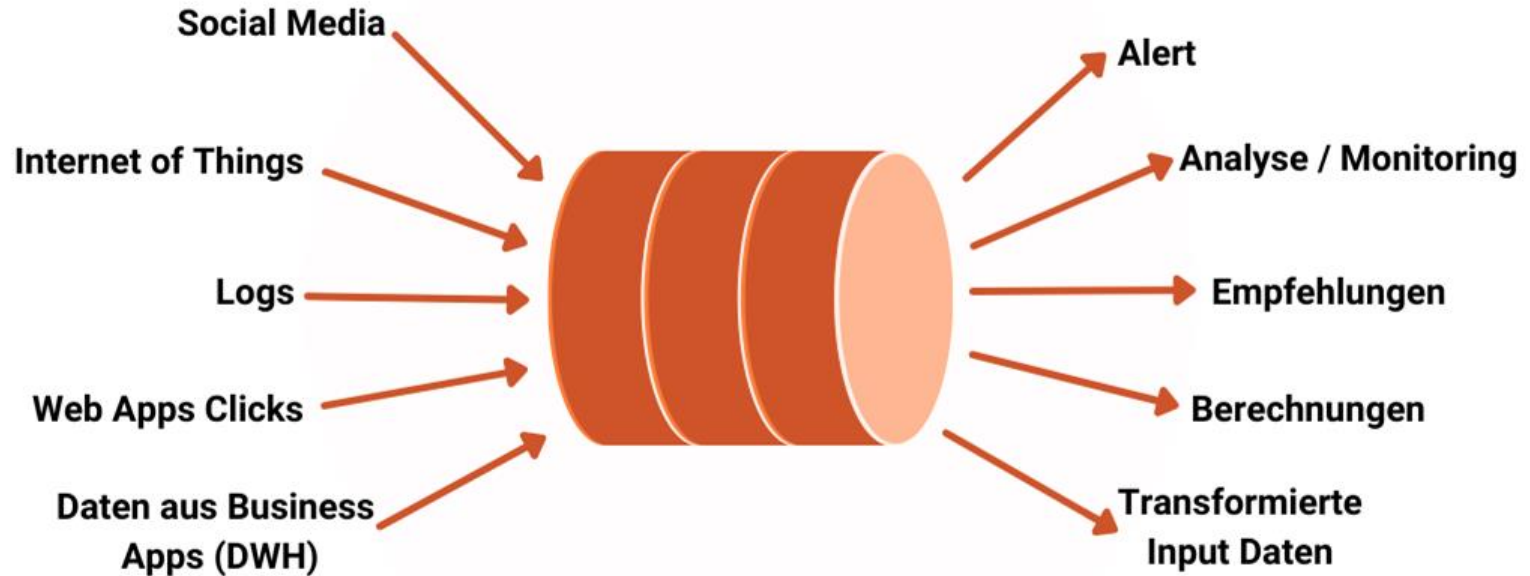
Quelle Bilder: Google



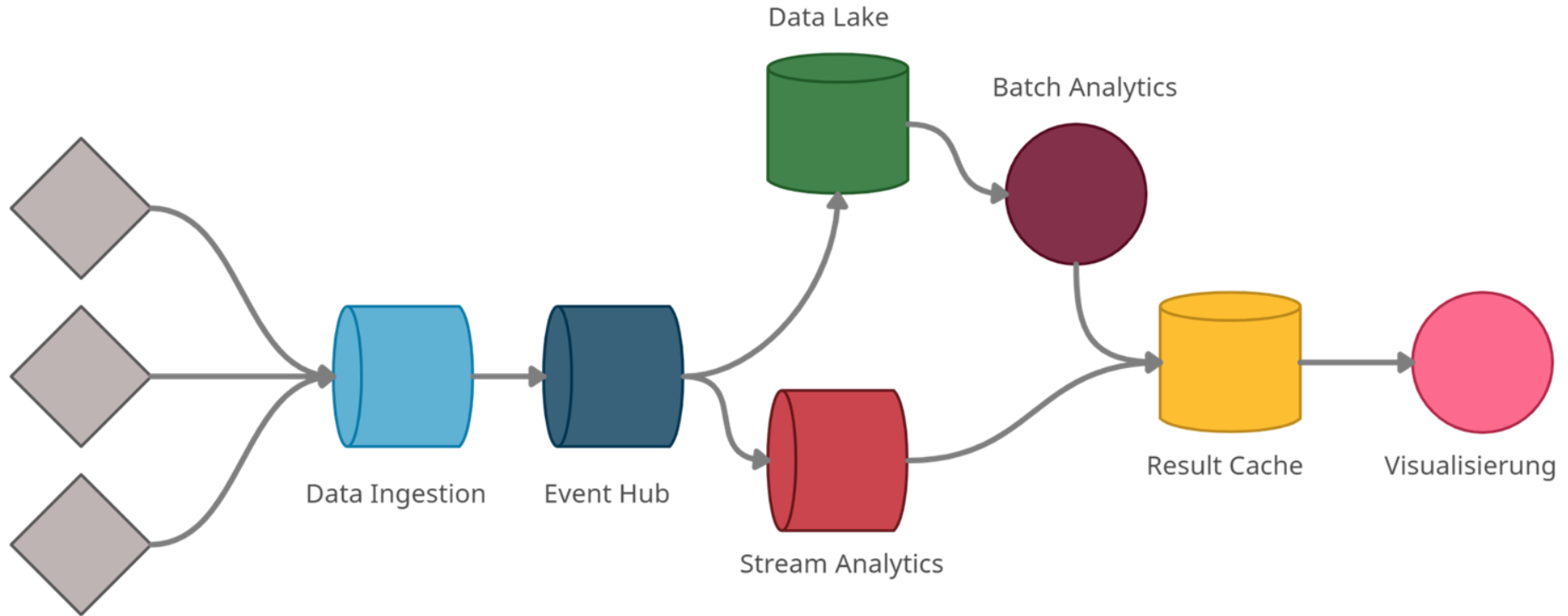
# Braucht Big Data auch Big Infrastructure?



# Einsatzmöglichkeiten



# Anatomie einer Big-Data-Streaming-Pipeline

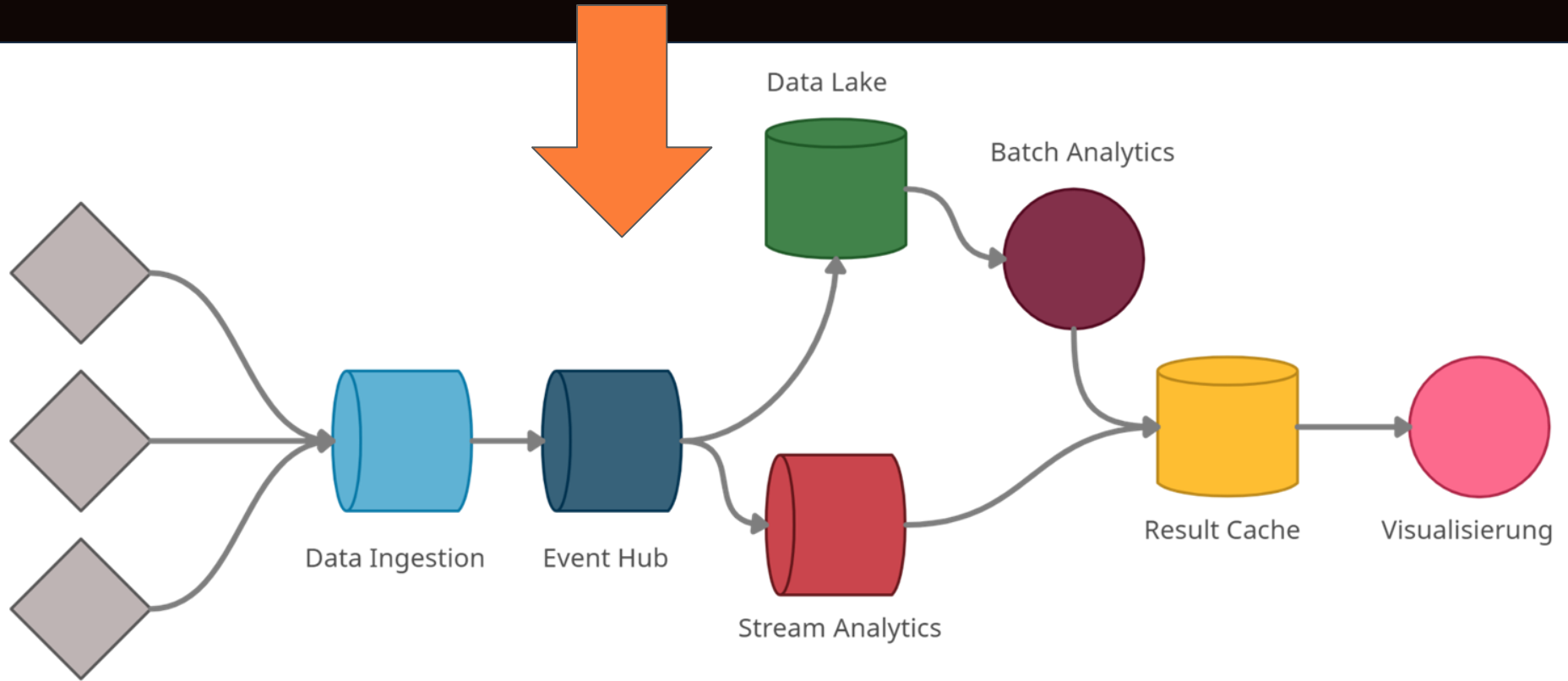


Datenquellen

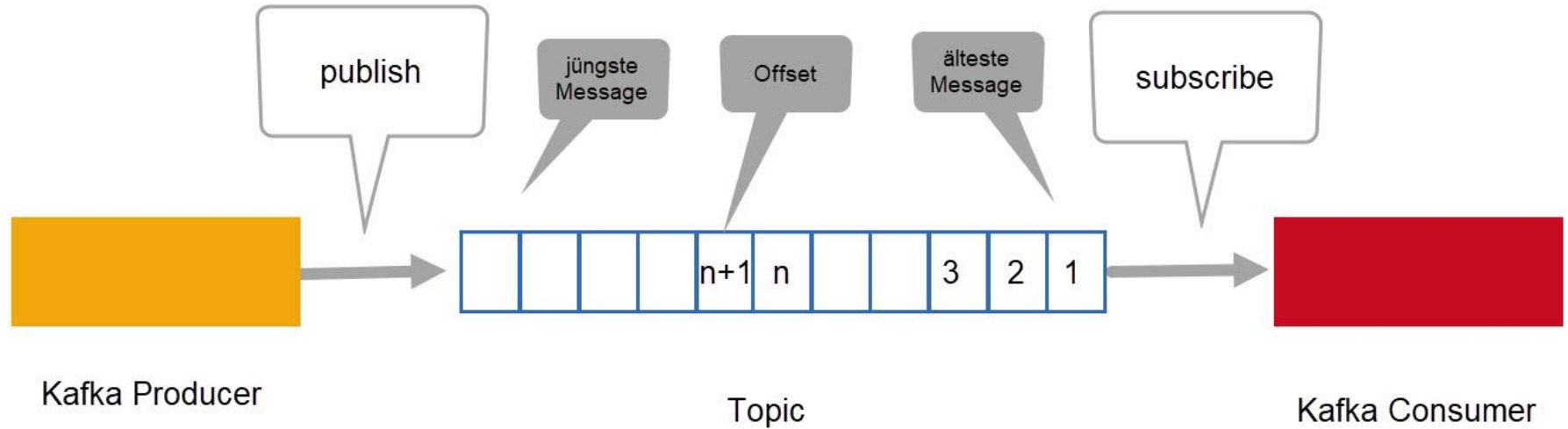
# Apache Kafka als Event Hub



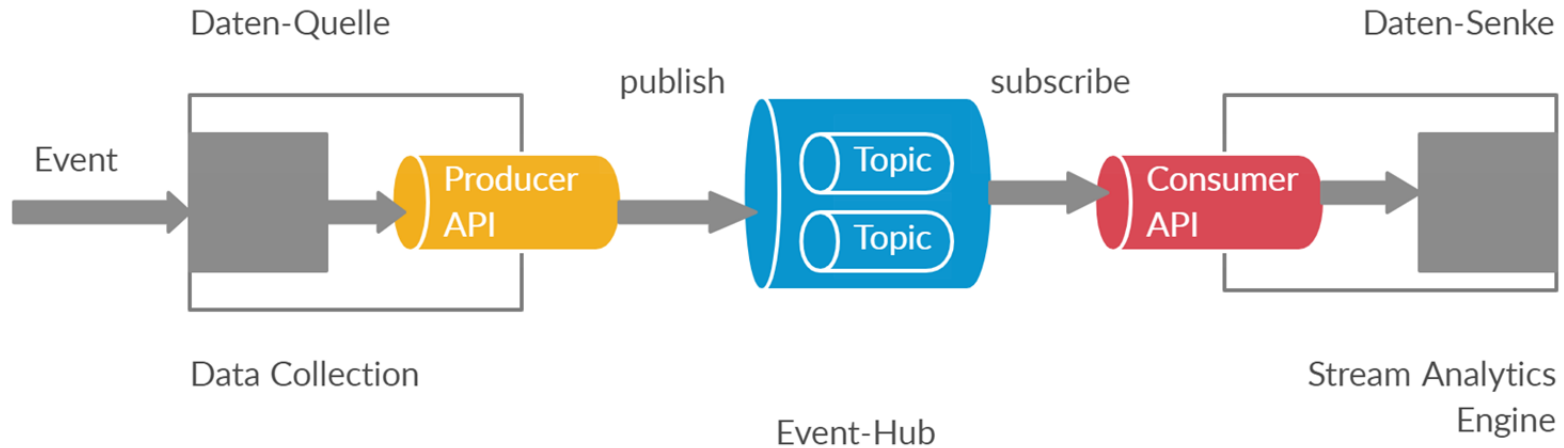
# Apache Kafka als Event Hub



# Warteschlange



# Verteilter Event Hub



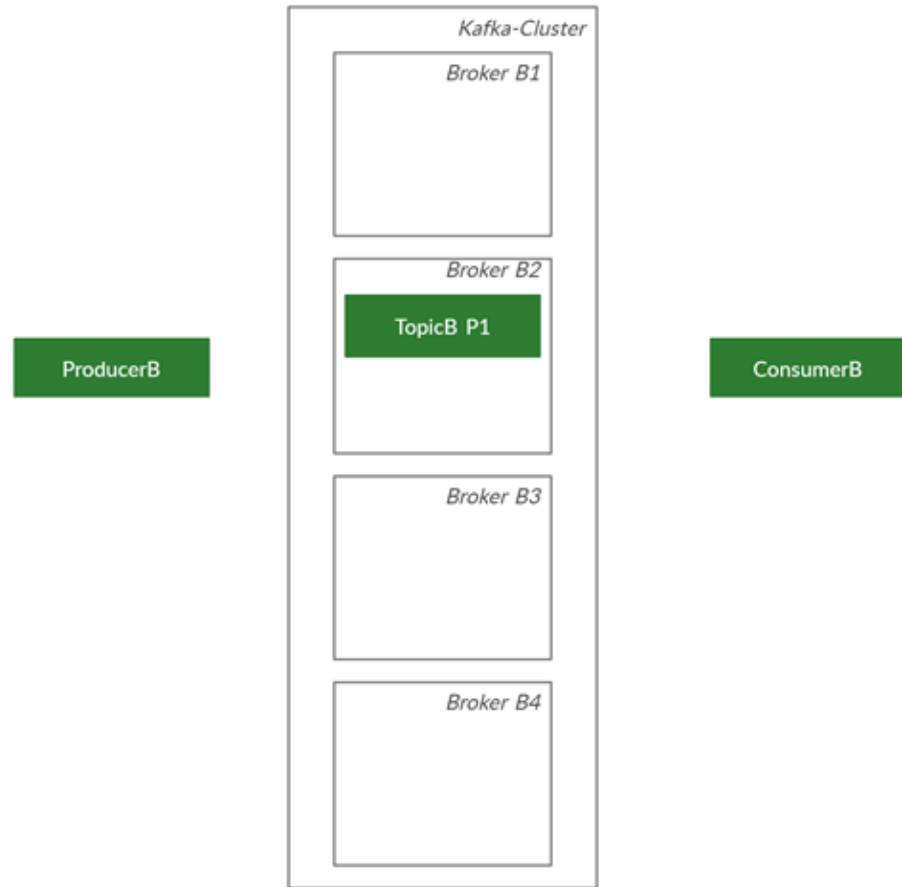
# Ordering Guarantee in Apache Kafka

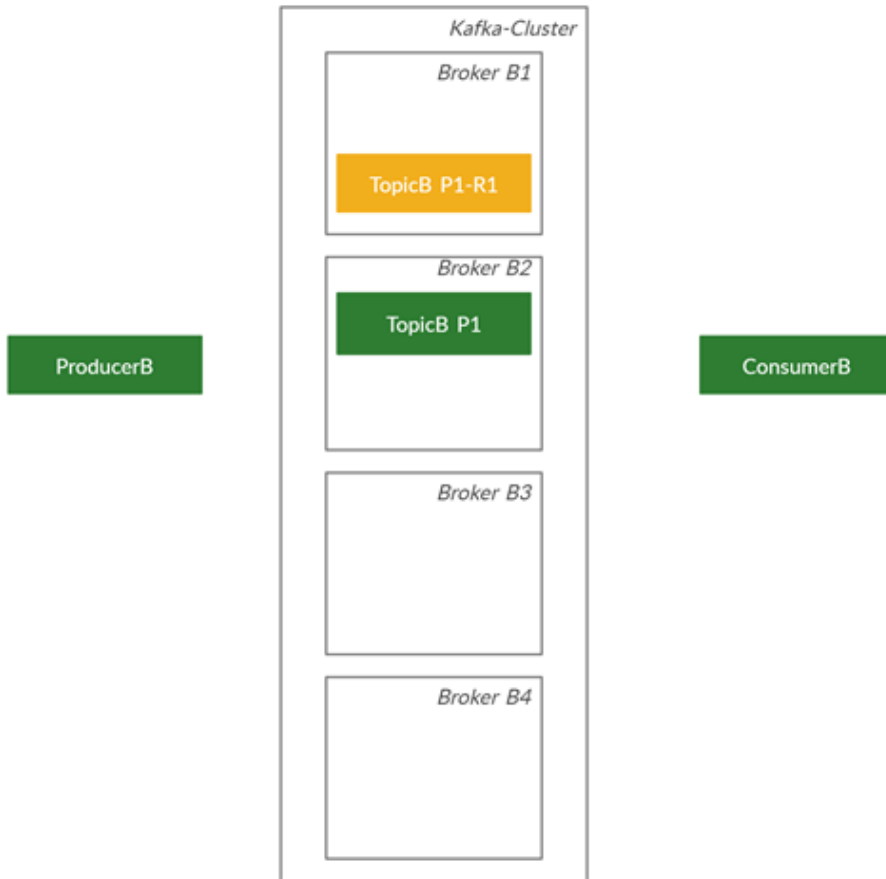


Video hier: <https://tirsus.com/big-data/realtime-streaming/ordering-guarantee-in-apache-kafka/>

# Blick hinter die Kulissen von Apache Kafka

# TopicB mir einer Partition P1

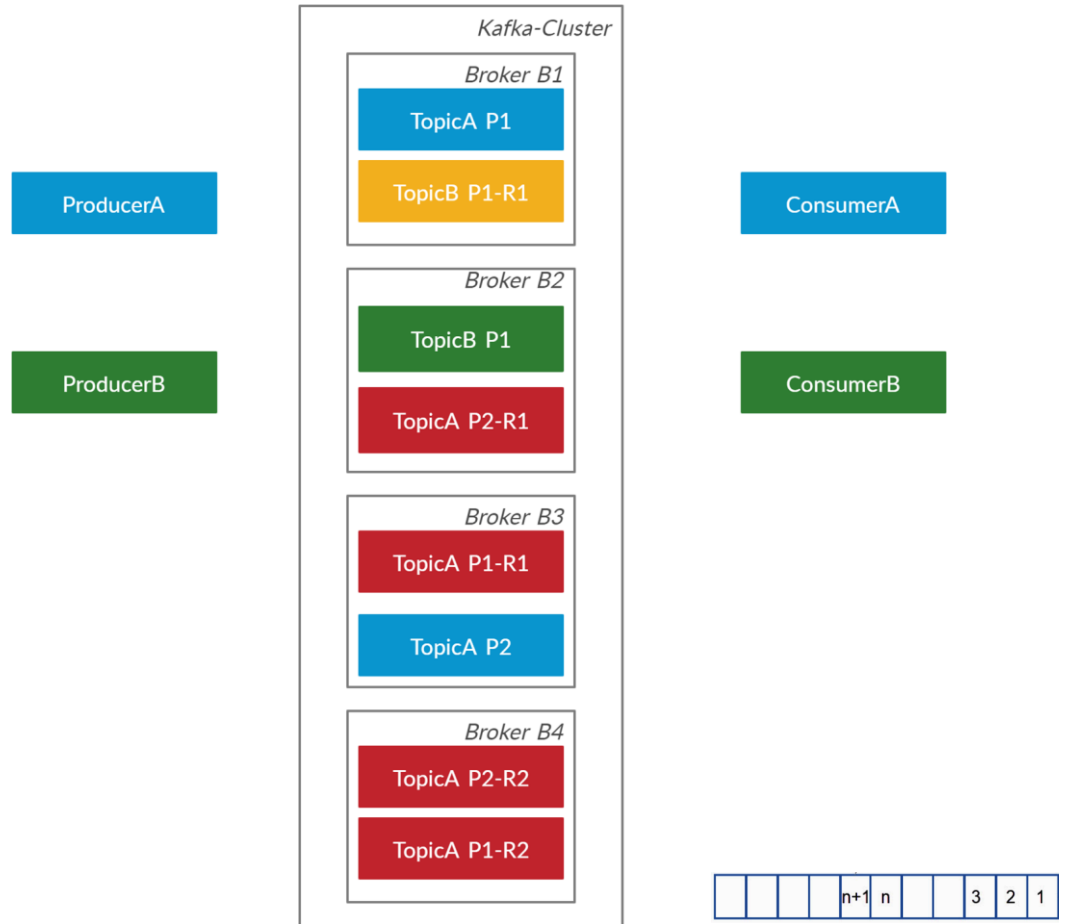




**TopicB mit einer  
Partition und  
zwei Replikaten  
P1, P1-R1**

# TopicA mit 2 Partitionen und Replikationsfaktor 3

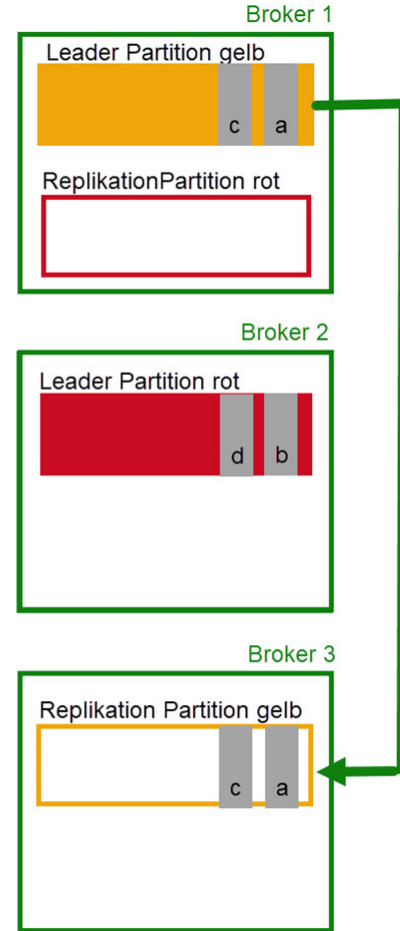
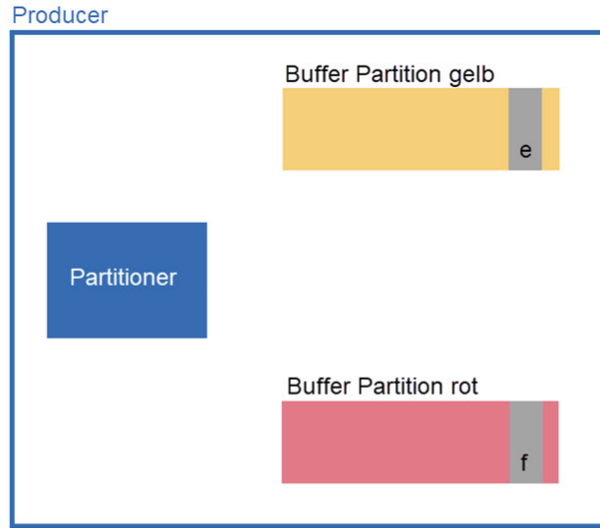
TopicB mit einer Partition und zwei Replikaten P1, P1-R1





# Auf ein Topic schreiben

g



# Brokerausfall?

# Wegweisend für Big Data: Vor 20 Jahren bewiesen

# Das CAP Theorem

Ein verteiltes Datensystem kann nur zwei der drei folgenden Garantien geben:

## Consistency

Jeder Lesebefehl erhält das Ergebnis des jüngsten Schreibbefehls oder eine Fehlermeldung.

## Availability

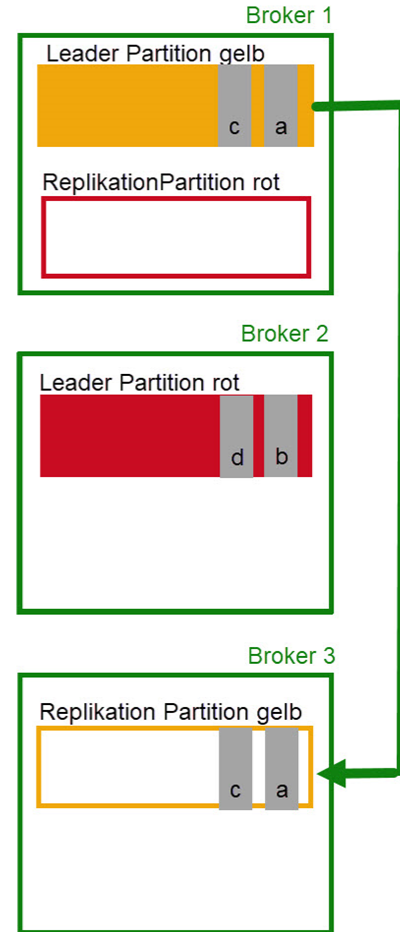
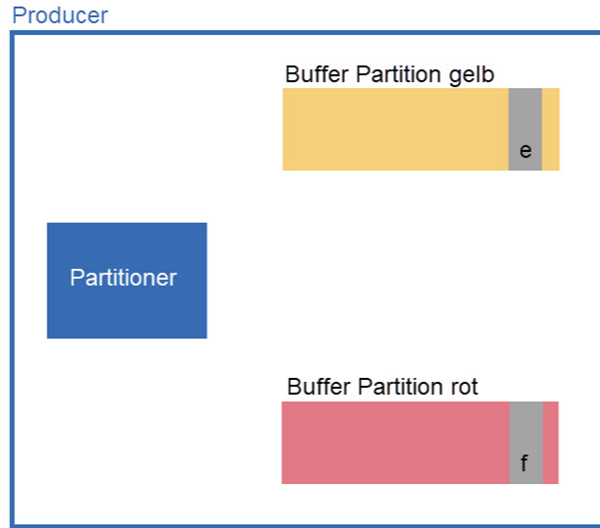
Jede Anfrage erhält eine fehlerfreie Antwort, jedoch ohne Garantie, dass die aktuellsten Daten berücksichtigt werden.

## Partition Tolerance

Das Gesamtsystem funktioniert weiter auch, wenn nicht alle Nachrichten, die zwischen den Systemkomponenten getauscht werden, eine zeitnahe Antwort erhalten, oder auch wenn Nachrichten verloren gehen.



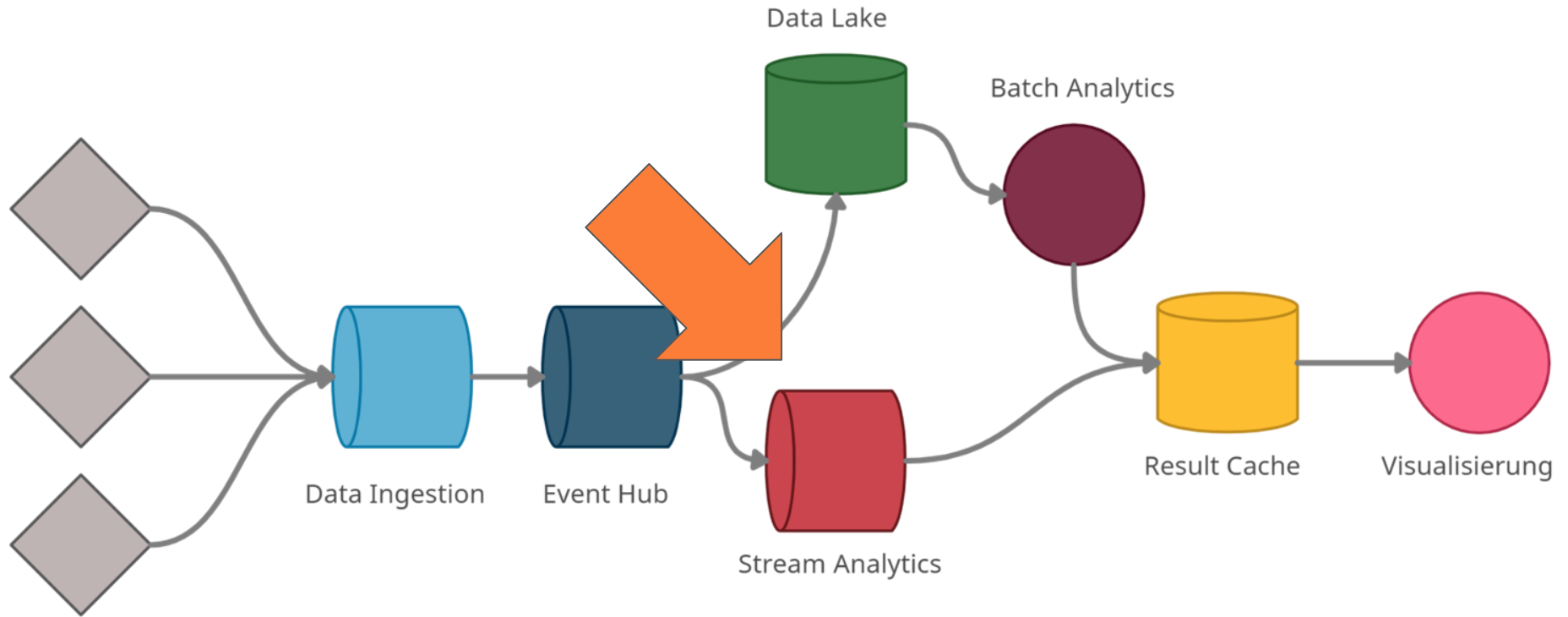
g



# Brokerausfall?

# Apache Spark als Stream Analytics Engine

# Apache Spark als Stream Analytics Engine



Datenquellen

# Daten von Apache Kafka lesen

```
messungen = spark\  
  .readStream\  
  .format('kafka')\  
  .option('kafka.bootstrap.servers', 'kafka-1:9092')\  
  .option('subscribe', 'sunus')\  
  .option('startingOffsets', 'earliest')\  
  .option('includeTimestamp', 'true')\  
  .load()\  
  .selectExpr("cast(value as string) as value","timestamp")\  
  .selectExpr("cast(split(value,',')[0] as double) as x",  
    "cast(split(value,',')[1] as double) as y",  
    "cast(split(value,',')[2] as timestamp) as event_time",  
    "timestamp",  
    "substring(split(value,',')[2], 1, 16) as minute"  
  )
```

# Daten analysieren und auf Result Cache schreiben

```
windowedCounts = messungen.groupBy(  
    window(messungen.timestamp, '3 seconds'),  
    messungen.minute  
)\  
    .avg()  
data = windowedCounts.select('window.start', 'window.end', 'avg(y)', 'minute')  
query = data\  
    .writeStream\  
    .outputMode("update")\  
    .foreachBatch(writeToResultCache)\  
    .start()\  
    .awaitTermination()
```



# Spark IDE – ein neues Beispiel

The screenshot displays the Spark IDE interface for a workflow named "financial\_integration\_workflow\_10 (11-21)". The interface includes a top navigation bar with categories: Input / Output, Transform, Join / Split, Machine Learning, Script / Reformat, and Visualize. A left sidebar contains icons for navigation and settings. The main workspace shows a complex data pipeline with the following components:

- Input Stage:** Three parallel "FileInput" nodes.
- Transformation Stage:** Each "FileInput" is followed by a "Router" and then a "Select" node.
- Aggregation Stage:** The outputs from the "Select" nodes are combined into two "UnionAll" nodes.
- Preparation Stage:** Each "UnionAll" node is followed by a "Prepare" node.
- Final Processing Stage:** The two "Prepare" nodes feed into a central "Router", which then branches into two parallel paths. Each path consists of a "Prepare" node, an "Aggregate" node, an "OrderByPartit..." node, and a "FileOutput" node.

Additional UI elements include a "SAVE" button, an "Editor" toggle, a "Select pool for cluster" dropdown, and a play button in the bottom right corner.

# Spark IDE – ein neues Beispiel

The screenshot displays the Spark IDE interface. At the top, there are navigation tabs: "Input / Output", "Transform", "Join / Split", "Machine Learning", "Script / Reformat", and "Visualize". The main workspace is divided into two panes. The left pane, titled "main.scala", shows a list of data frames and their sources, such as "df\_B9\_Process\_Router" (RowDistributor1) and "df\_c232" (FileInput). The right pane, titled "graph/B9\_Process\_Prepare.scala", contains Scala code for a Spark job. The code includes imports for Spark SQL functions and a visual definition for "B9\_Process\_Prepare". The visual definition specifies the input data frame and the output columns, including a complex filter for transaction IDs and directions.

```
main.scala
19 df_B9_Process_Router: RowDistributor1 = B9_Process_Router(spark, df,
20 df_c232: FileInput = c232(spark)
21 df_Bucket4: FileInput = Bucket4(spark)
22 df_Bucket8: FileInput = Bucket8(spark)
23 df_B8_Process_Router: RowDistributor1 = B8_Process_Router(spark, df,
24 df_Bucket5: FileInput = Bucket5(spark)
25 df_B5_Process_Router: RowDistributor1 = B5_Process_Router(spark, df,
26 df_Bucket3: FileInput = Bucket3(spark)
27 df_B3_Process_Router: RowDistributor1 = B3_Process_Router(spark, df,
28 df_c133: FileInput = c133(spark)
29 df_b27_File: FileInput = b27_File(spark)
30 df_B9_Process_Prepare: Prepare = B9_Process_Prepare(spark, df,
31 df_Bucket2: FileInput = Bucket2(spark)
32 df_B2_Process_Router: RowDistributor1 = B2_Process_Router(spark, df,
33 df_B3_Process_Prepare: Prepare = B3_Process_Prepare(spark, df,
34 df_All_day_annotated: FileInput = All_day_annotated(spark)
35 df_Rollup_ann: Aggregate = Rollup_ann(spark, df_All_day_annotated,
36 df_Sort_ann: OrderByPartition = Sort_ann(spark, df_Rollup_ann,
37 df_B2_Process_Prepare: Prepare = B2_Process_Prepare(spark, df,
38 df_B4_Process_Router_Log: Select = B4_Process_Router_Log(spark, df,
39 df_B8_Process_Prepare: Prepare = B8_Process_Prepare(spark, df,
40 df_B5_Process_Prepare: Prepare = B5_Process_Prepare(spark, df,
41 df_c10_Prepare: Prepare = c10_Prepare(spark, df_c133)
42 df_c6_Prepare: Prepare = c6_Prepare(spark, df_b27_File,
43 day_ann_processed(spark, df_Sort_ann)
44 df_B9_Process_Router_Log: Select = B9_Process_Router_Log(spark, df_B9_Process_Router, df_c10_Prepare, df_c6_Prepare, df_B9_Process_Prepare)
```

```
graph/B9_Process_Prepare.scala
13 import org.apache.spark.sql.functions._
14 import graph._
15
16 @Visual(id = "n31", label = "Tran_Process.B9_Process_Prepare", x = 100, y = 100)
17 object B9_Process_Prepare {
18
19   def apply(spark: SparkSession, in: DataFrame): Prepare = {
20     import spark.implicits._
21
22     val out = in.select(
23       string_substring(col("full_date"), lit(7), lit(4)).as("year"),
24       string_substring(col("full_date"), lit(5), lit(2)).as("month"),
25       string_substring(col("inserted_date"), lit(3), lit(2)).as("day"),
26       // transaction operations
27       concat(
28         col("transaction_id"),
29         when(
30           (col("transaction_id") == lit("05"))
31             .or(col("transaction_id") == lit("06"))
32             .or(col("transaction_id") == lit("07")),
33           when(
34             col("trasaction_direction") == lit("SS"),
35             when(col("post_code_type") != lit("13"), lit("1")).otherwise(
36               ).otherwise(when(col("usage_type") == lit("1"), lit("1")).otherwise(
37                 lit(" ")
38             ),
39           when(col("settlement_claim_id").cast(DecimalType(4, 0)) == lit(1
```

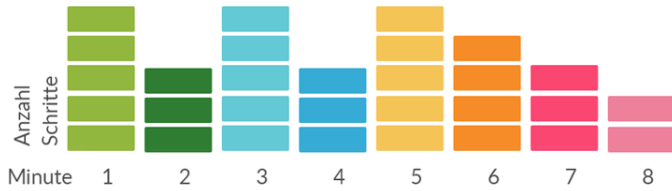
# Blick hinter die Kulissen von Apache Spark



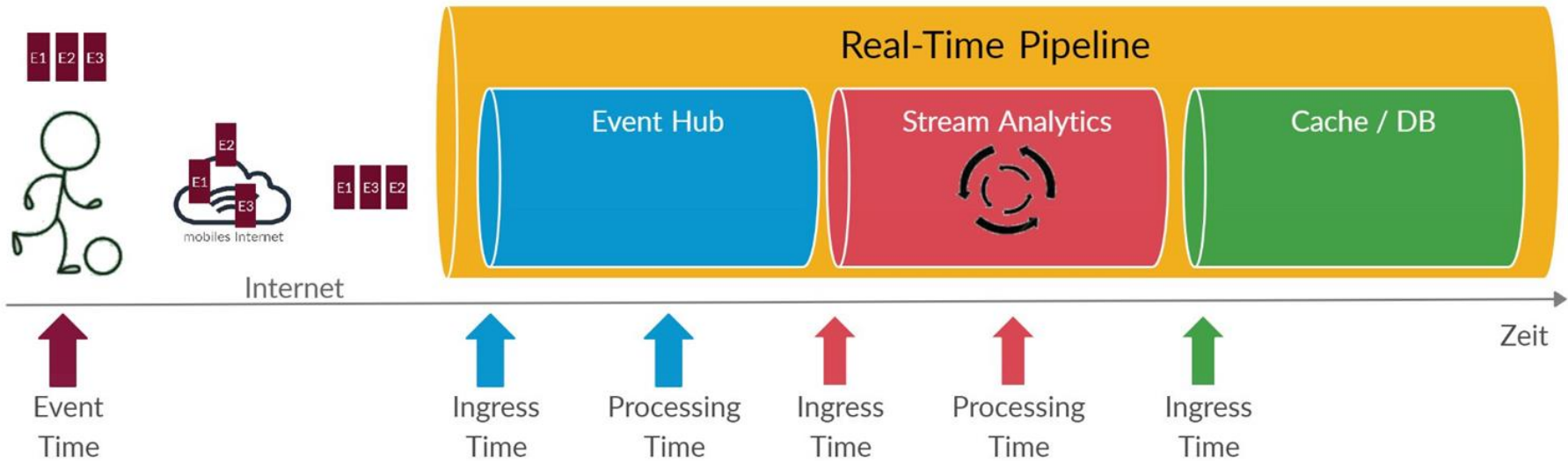
Messung

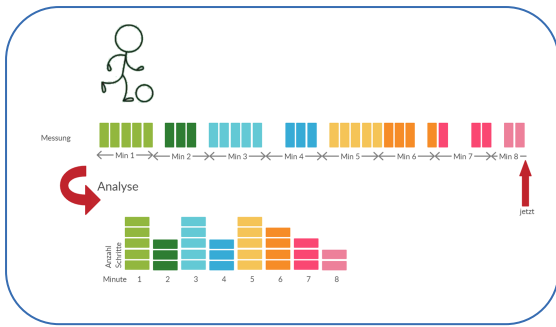


Analyse



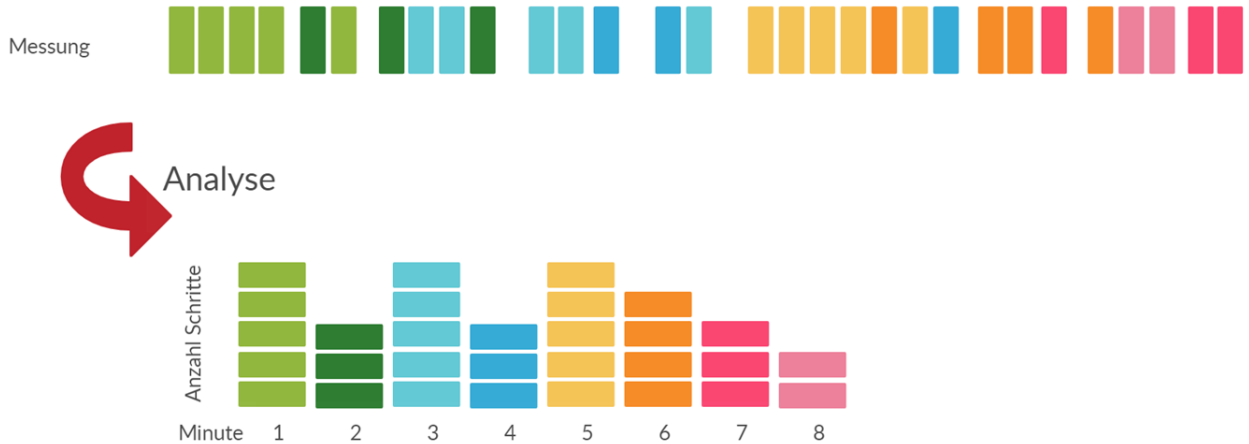
jetzt





Ziel: Exactly Once

```
schritteProMinute = schrittstream \
.withWatermark("timestamp", "2 minutes") \
.groupBy(
  window (schrittstream.timestamp, "1 minute"),
  schrittstream.spieler) \
.count()
```



# Dive Deeper

## Exactly Once Semantik

**Exactly Once** Jedes Event wird genau ein Mal verarbeitet.

**At Most Once** Wir nehmen in Kauf, dass im Fehlerfall Events verloren gehen.

**At Least Once** Wir nehmen in Kauf, dass im Fehlerfall Events mehrfach verarbeitet werden.

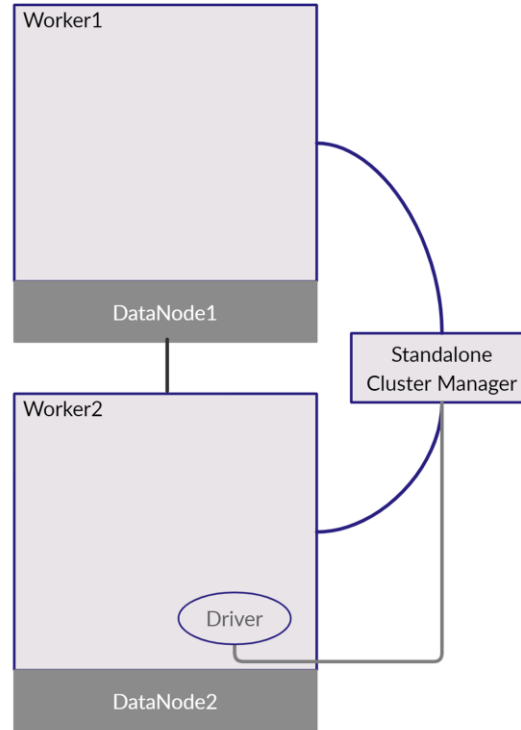
**Exactly Once -> Transaktionen**



# Cluster Manager

## Workers

## HDFS



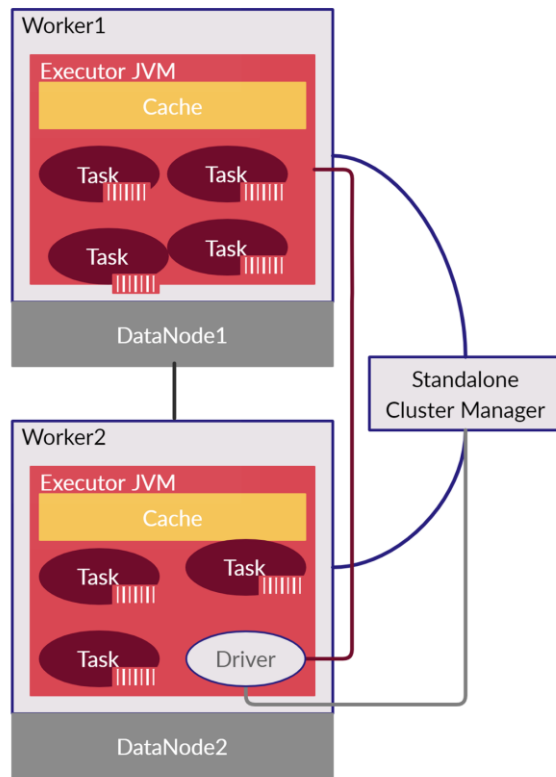
**Standalone Cluster Manager**

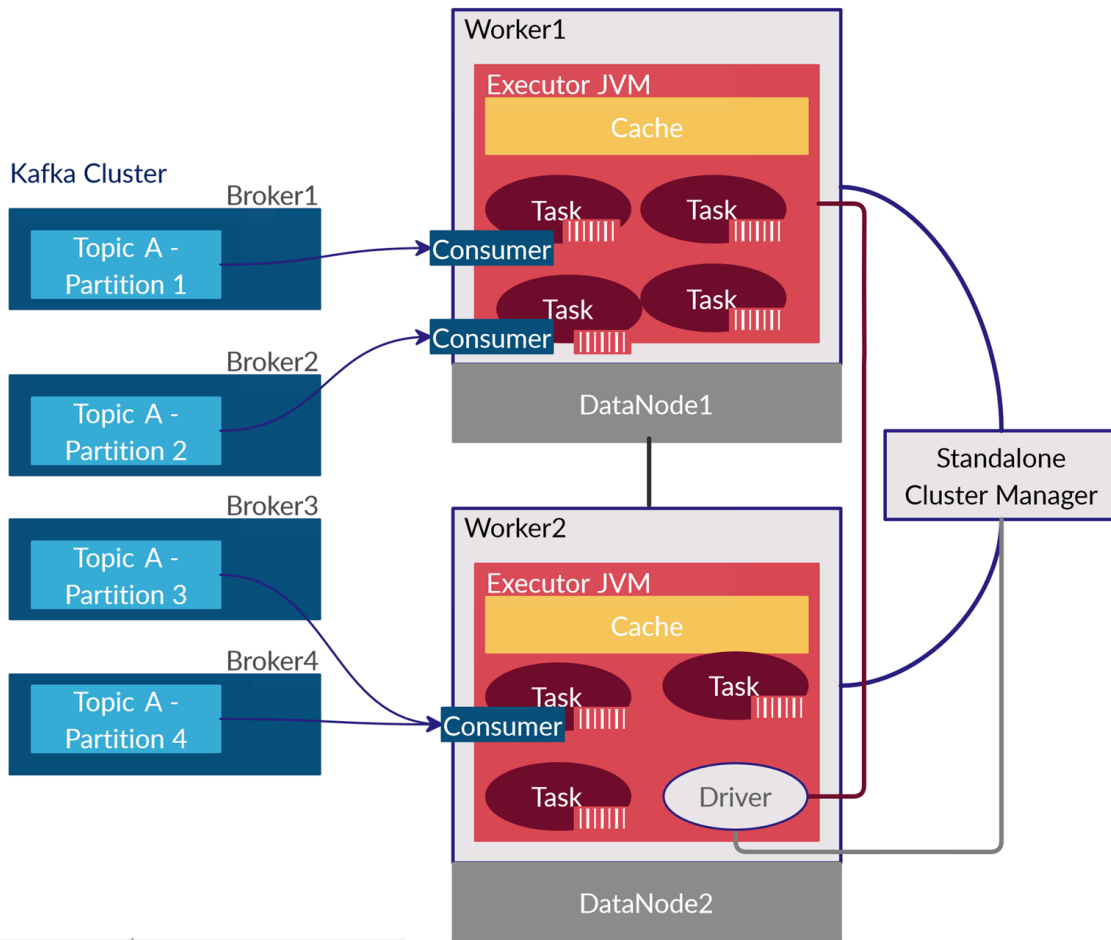
**Apache YARN**

**Apache Mesos**

**Kubernetes**

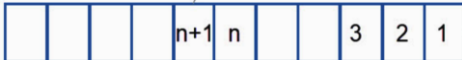
# Driver - Executor





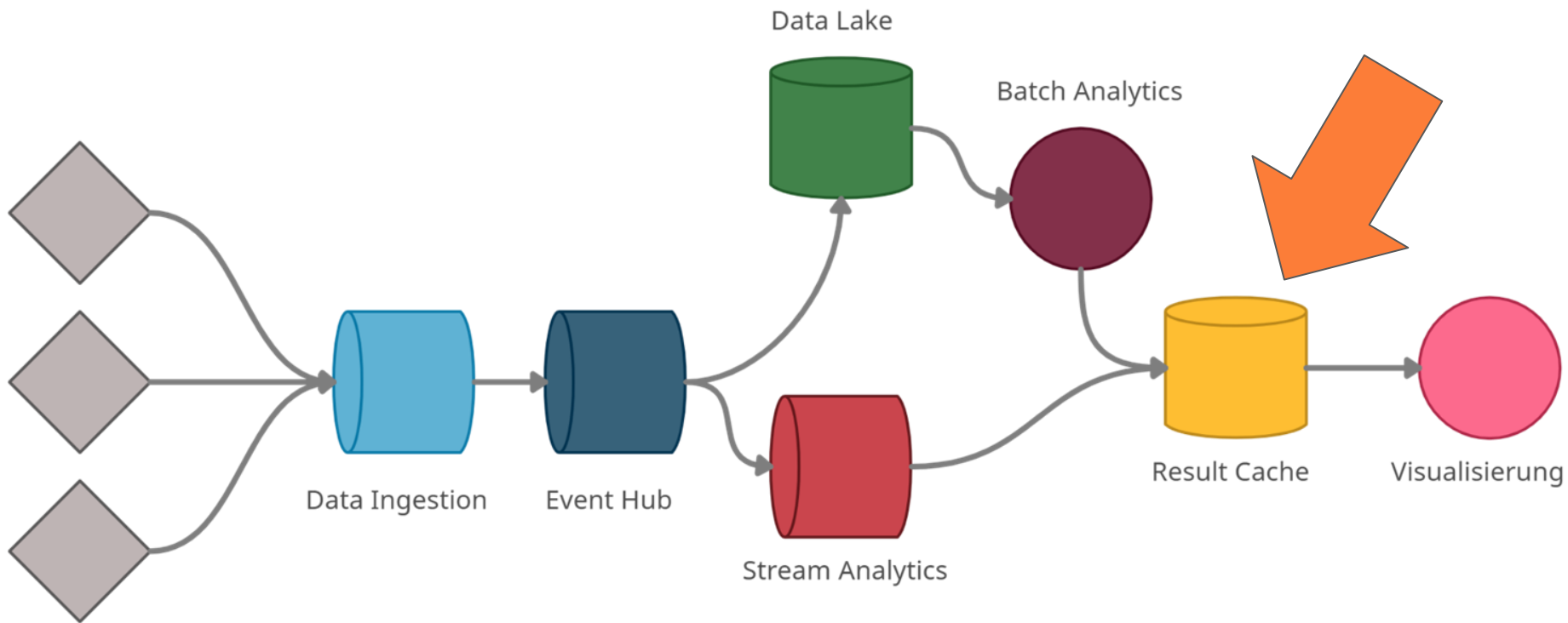
Spark liest von Kafka

Workerausfall?



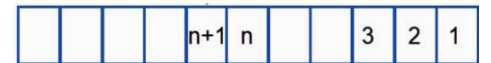
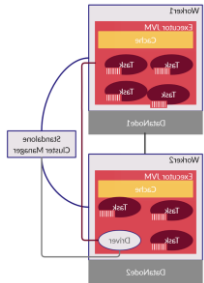
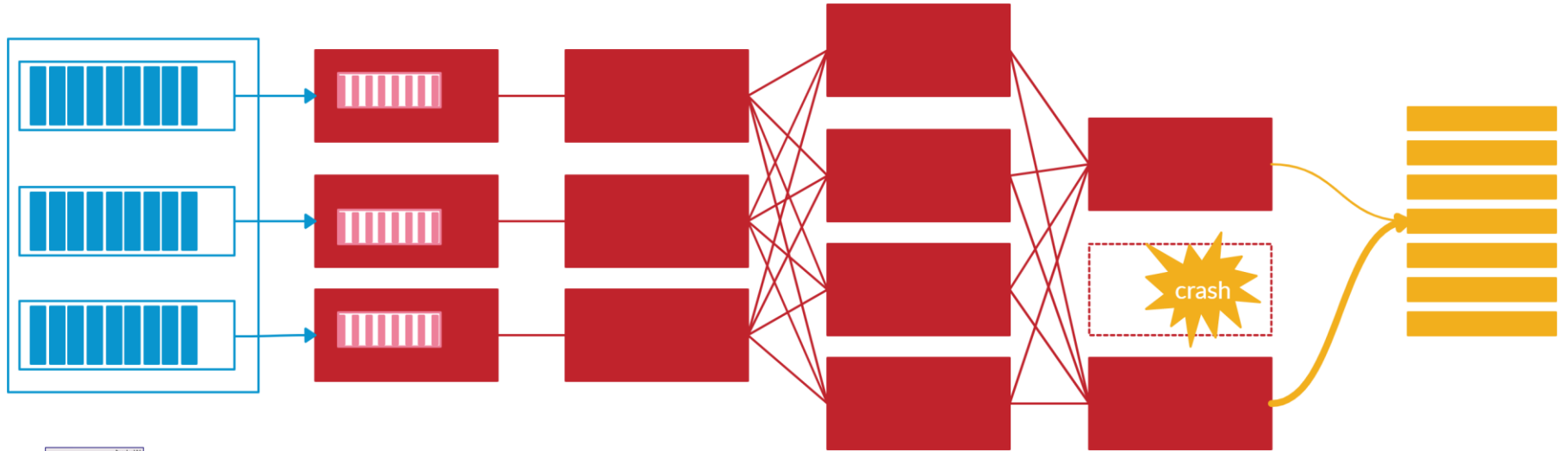
# Der Result Cache

# Der Result Cache



Datenquellen

# Idempotente Schreiboperationen



# Trade-Offs der Real-Time Analyse großer Datenströme



## Free E-Mail-Training



<https://tirsus.com/pipeline/>

**Ursula Deriu**  
ursula.deriu@tirsus.com  
  
 <https://www.linkedin.com/in/ursuladeriu>  
**tirsus.com/big-data**

# Q & A