SL01：8.6 迟到数据处理

8.6.1 事件时间

8.6.2 迟到数据

8.6.3 水印

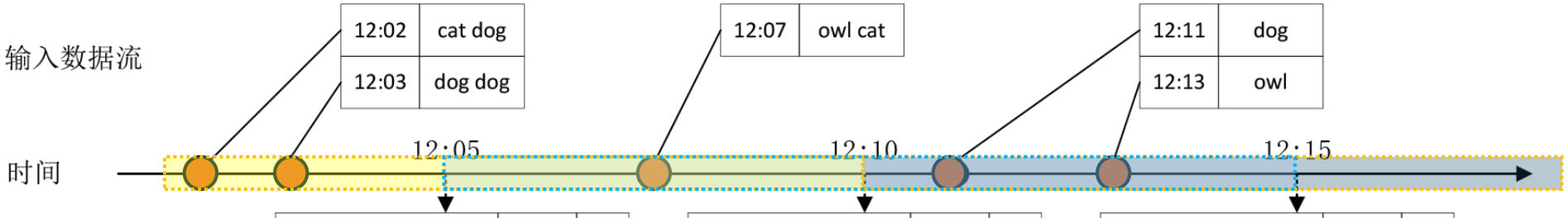
8.6.4 多水印规则

8.6.5 处理迟到数据的实例

SL02：8.6.1 事件时间



假设某个数据流中包含数据生成的时间，计算每10分钟内接收到的数据流内的单词个数，并每5分钟更新一次。



12:00 —12:10	cat	1
12:00 —12:10	dog	3

每5分钟触发周期后的结果表

窗口大小为10分钟，滑动时间间隔为5分钟，计算滑动窗口内的聚合结果

12:00 —12:10	cat	2
12:00 —12:10	dog	3
12:00 —12:10	owl	1
12:05 —12:15	cat	1
12:05 —12:15	owl	1

12:00—12:10和12:05—12:15时间窗口内的词频都会增加

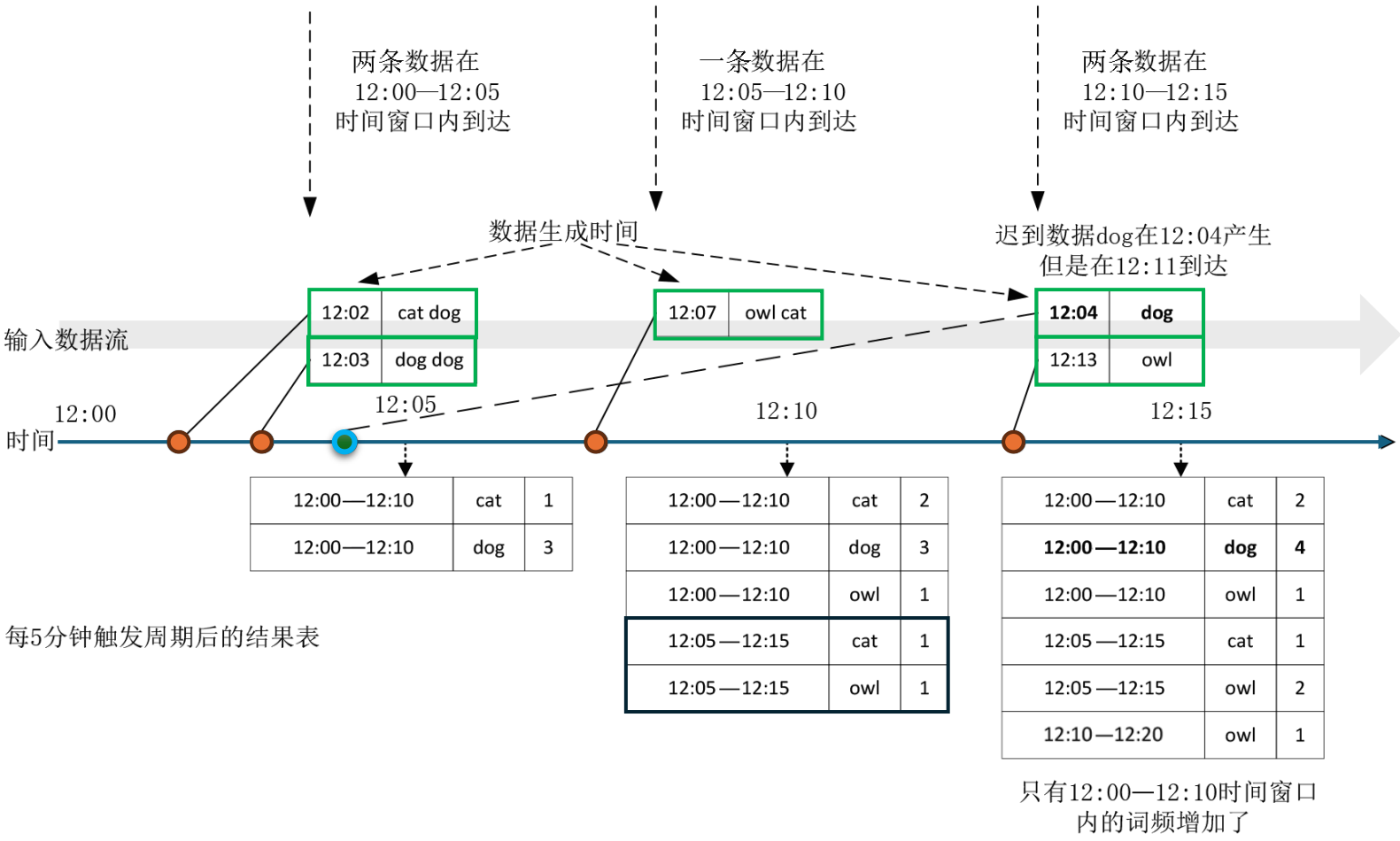
12:00—12:10	cat	2
12:00—12:10	dog	3
12:00—12:10	owl	1
12:05—12:15	cat	1
12:05—12:15	owl	2
12:05—12:15	dog	1
12:10—12:20	dog	1
12:10—12:20	owl	1

12:05—12:15和12:10—12:20时间窗口内的词频都会增加

SL03：8.6.2 迟到数据



现在考虑这样一种情况，如果一个事件迟到，应用程序会发生什么。



SL04：8.6.3 水印



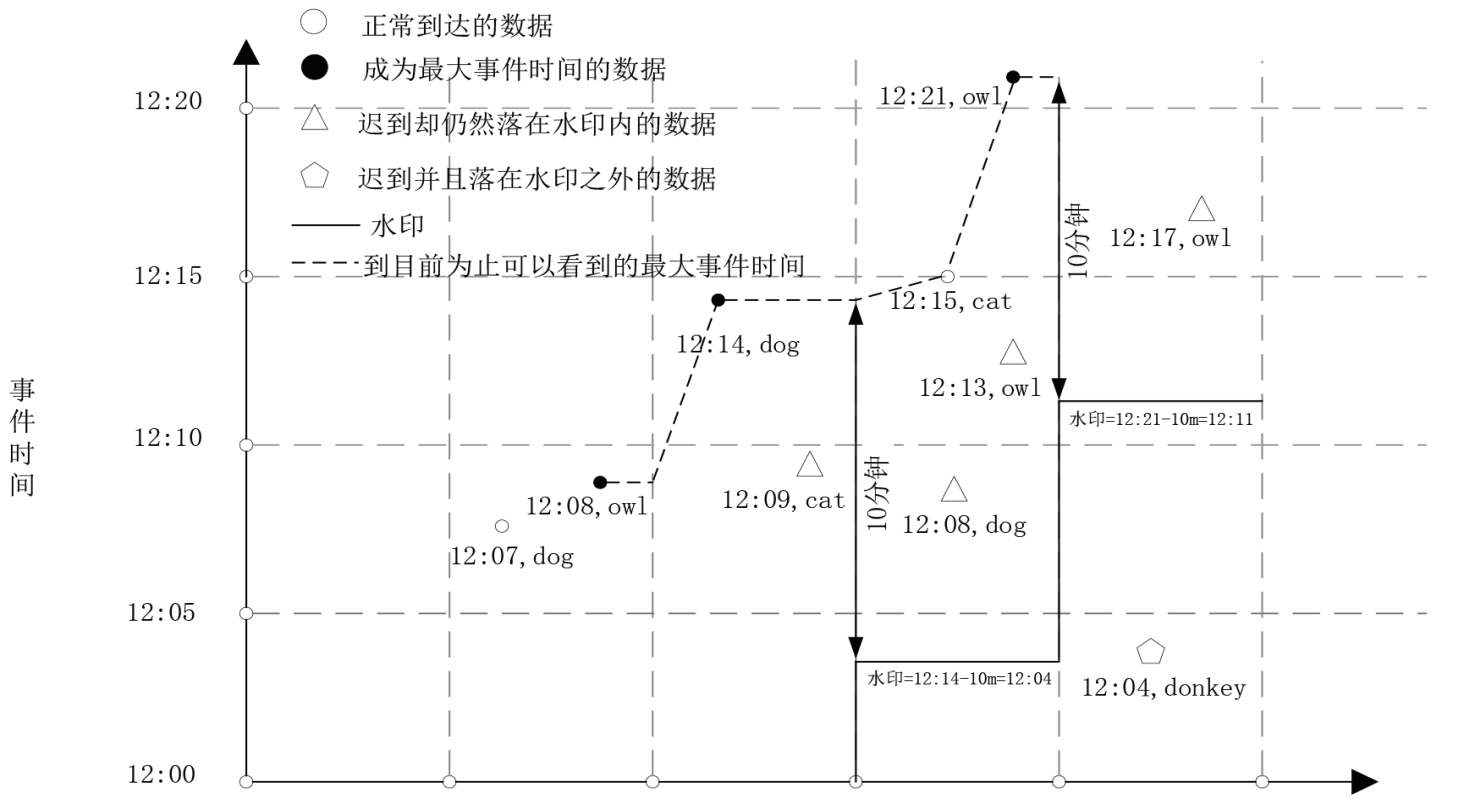
定义水印可以使用withWatermark()方法，下面是一个代码片段：

```
words = ...  # DataFrame流的结构为 { timestamp: Timestamp, word: String }

# 对窗口和字母进行聚合并计算计数
windowedCounts = words \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
        .count()
```

SL05：8.6.3 水印





处理时间，每5分钟触发一次

12:05

12:00—12:10	owl	1
12:00—12:10	dog	1
12:05—12:15	owl	1
12:05—12:15	dog	1

12:10

12:00—12:10	owl	1
12:00—12:10	dog	1
12:00—12:10	cat	1
12:05—12:15	owl	1
12:05—12:15	dog	2
12:05—12:15	cat	1
12:10—12:20	dog	1

12:15

12:00—12:10	owl	1
12:00—12:10	dog	2
12:00—12:10	cat	1
12:05—12:15	owl	2
12:05—12:15	dog	3
12:05—12:15	cat	2
12:10—12:20	dog	1
12:10—12:20	cat	1
12:10—12:20	owl	1
...		

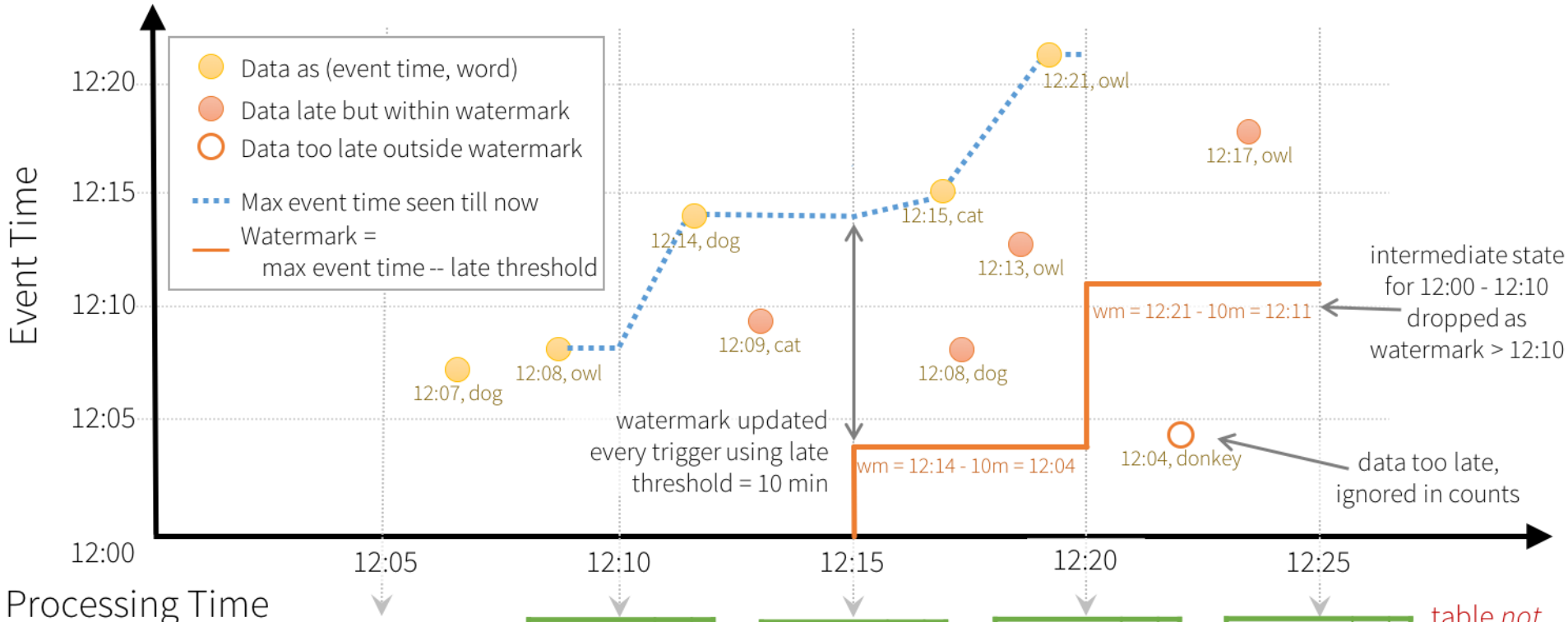
12:20

12:00—12:10	owl	1
12:00—12:10	dog	2
12:00—12:10	cat	1
12:05—12:15	owl	2
12:05—12:15	dog	3
12:05—12:15	cat	2
12:10—12:20	dog	1
12:10—12:20	cat	1
12:10—12:20	owl	2
...		

12:25

12:00—12:10	owl	1
12:00—12:10	dog	2
12:00—12:10	cat	1
12:05—12:15	owl	2
12:05—12:15	dog	3
12:05—12:15	cat	2
12:10—12:20	dog	1
12:10—12:20	cat	1
12:10—12:20	owl	2
...		

每5分钟触发后的结果表





12:00 - 12:10	owl	1
12:00 - 12:10	dog	1
12:05 - 12:15	owl	1
12:05 - 12:15	dog	1

12:00 - 12:10	owl	1
12:00 - 12:10	dog	1
12:00 - 12:10	cat	1
12:05 - 12:15	owl	1
12:05 - 12:15	dog	2
12:05 - 12:15	cat	1
12:10 - 12:20	dog	1

12:00 - 12:10	owl	1
12:00 - 12:10	dog	2
12:00 - 12:10	cat	1
12:05 - 12:15	owl	2
12:05 - 12:15	dog	3
12:05 - 12:15	cat	2
12:10 - 12:20	dog	1
12:10 - 12:20	cat	1
12:10 - 12:20	owl	1
...		

12:00 - 12:10	owl	1
12:00 - 12:10	dog	2
12:00 - 12:10	cat	1
12:05 - 12:15	owl	2
12:05 - 12:15	dog	3
12:05 - 12:15	cat	2
12:10 - 12:20	dog	1
12:10 - 12:20	cat	1
12:10 - 12:20	owl	2
...		

Result Tables after each trigger

table *not* updated with too late data (12:04, donkey)

table updated with late data (12:17, owl)

purple rows are updated rows that are written to the sink as output

Watermarking in **Windowed Grouped Aggregation** with **Update Mode**



正常到达的数据

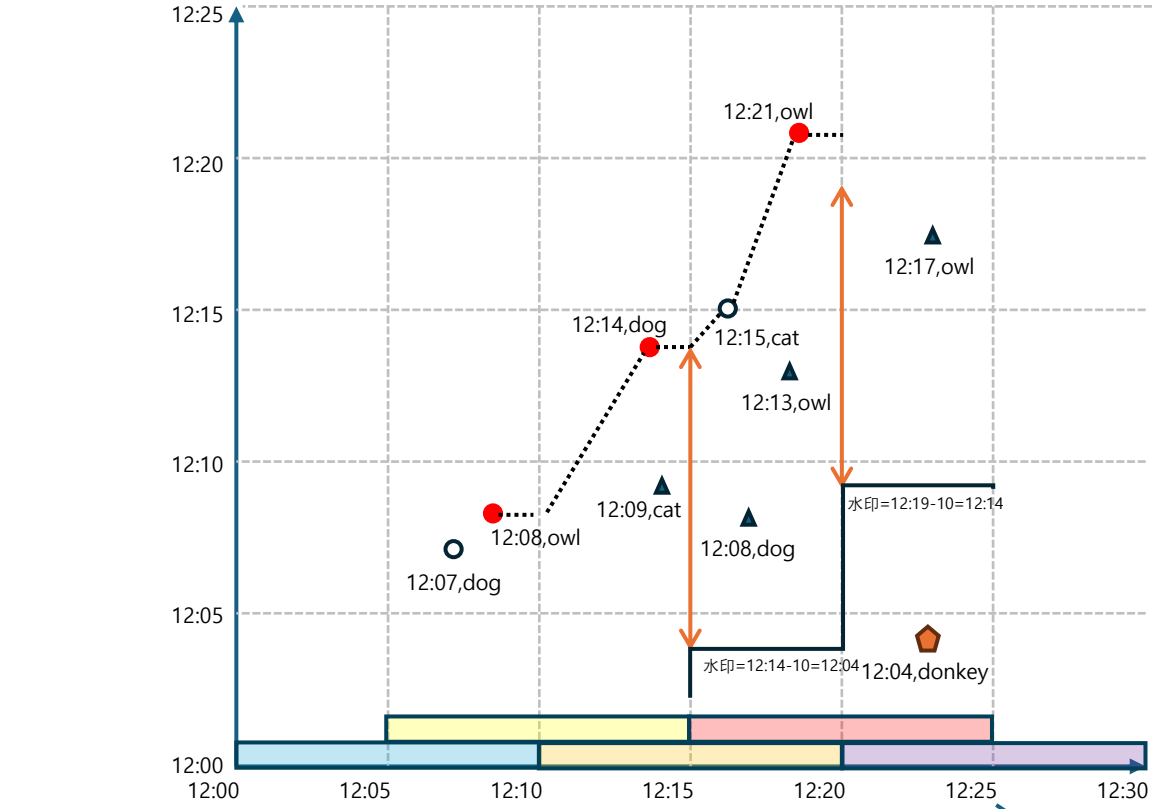
成为最大事件时间的数据

迟到却仍然落在水印内的数据

迟到并落在水看到的最大事件时间

水印（watermark）

到目前为止可以看到的最大事件事件



12:00-12:10	dog	1
12:00-12:10	owl	1
12:05-12:15	dog	1
12:05-12:15	owl	1

12:00-12:10	dog	1
12:00-12:10	owl	1
12:05-12:15	dog	2
12:05-12:15	owl	1
12:05-12:15	cat	1
12:10-12:20	dog	1

12:00-12:10	dog	2
12:00-12:10	owl	1
12:05-12:15	dog	3
12:05-12:15	owl	2
12:05-12:15	cat	1
12:10-12:20	dog	1
12:10-12:20	owl	2
---	--	-

12:00-12:10	dog	2
12:00-12:10	owl	1
12:05-12:15	dog	3
12:05-12:15	owl	2
12:05-12:15	cat	1
12:10-12:20	dog	1
12:10-12:20	owl	2
---	--	-

SL0 8 ：8.6.4 多水印规则



- 如果一个查询来自多个输入源的联合或者连接，那么，每个输入源都可以自定义一个单独的水印来跟踪中间状态。单个输入源定义的水印，不会影响其他输入源。
- 多个输入源可以定义不同的水印，Structured Streaming会独立跟踪每个数据流中的事件时间，并分别计算延迟。
- 此外，如果联合或连接查询存在有状态的操作，Structured Streaming会选择一个全局的水印，来跟踪所有流计算的中间状态。
- 在默认情况下，为了保证所有流的迟到数据都会被处理，Structured Streaming会选择一个最长的迟到的水印时间。
- 也就是说，如果一个流允许数据迟到10分钟，而另外一个流允许数据迟到20分钟，那么，全局的水印就会允许保留20分钟的中间状态。

SL09 ：8.6.5 处理迟到数据的实例



```
# 导入需要用到的模块
import os
import shutil
from functools import partial

from pyspark.sql import SparkSession
from pyspark.sql.functions import window
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import TimestampType, StringType
```

SL10：8.6.5 处理迟到数据的实例



```
# 定义CSV文件的路径常量
TEST_DATA_DIR = '/tmp/testdata/'
TEST_DATA_DIR_SPARK = 'file:///tmp/testdata/'


# 搭建测试环境，判断CSV文件夹是否存在，如果存在则删除旧数据，并建立文件夹
def test_setUp():
    if os.path.exists(TEST_DATA_DIR):
        shutil.rmtree(TEST_DATA_DIR, ignore_errors=True)
    os.mkdir(TEST_DATA_DIR)


# 恢复测试环境，对CSV文件夹进行清理
def test_tearDown():
    if os.path.exists(TEST_DATA_DIR):
        shutil.rmtree(TEST_DATA_DIR, ignore_errors=True)
```

SL11: 8.6.5 处理迟到数据的实例



```
# 编写模拟输入的函数，传入CSV文件名和数据。注意写入应当是原子性的
# 如果写入时间较长，应当先写入临时文件再移动到CSV目录内
# 这里采取直接写入的方式
def write_to_csv(filename, data):
    with open(TEST_DATA_DIR + filename, "wt", encoding="utf-8") as f:
        f.write(data)
```

SL12：8.6.5 处理迟到数据的实例



```
if __name__ == "__main__":
    test_setUp()

    # 定义模式，该模式为字符串类型的word和时间戳类型的eventTime两个列组成
    schema = StructType([
        StructField("word", StringType(), True),
        StructField("eventTime", TimestampType(), True)])

    spark = SparkSession \
        .builder \
        .appName("StructuredNetworkWordCountWindowedDelay") \
        .getOrCreate()

    spark.sparkContext.setLogLevel('WARN')
```

SL13：8.6.5 处理迟到数据的实例



```
lines = spark \
    .readStream \
    .format('csv') \
    .schema(schema) \
    .option("sep", ";") \
    .option("header", "false") \
    .load(TEST_DATA_DIR_SPARK)
```

```
# 定义窗口
windowDuration = '1 hour'

windowedCounts = lines \
    .withWatermark("eventTime", "1 hour") \
    .groupBy('word', window('eventTime', windowDuration)) \
    .count()
```

SL14：8.6.5 处理迟到数据的实例



```
    query = windowedCounts \
        .writeStream \
        .outputMode("update") \
        .format("console") \
        .option('truncate', 'false') \
        .trigger(processingTime="8 seconds") \
        .start()

    # 写入测试文件file1.csv
    write_to_csv('file1.csv', """
正常;2023-10-01 08:00:00
正常;2023-10-01 08:10:00
正常;2023-10-01 08:20:00
""")
```

SL15: 8.6.5 处理迟到数据的实例



```
    # 处理当前数据
    query.processAllAvailable()

    # 此时事件时间更新到上次看到的最大的2023-10-01 08:20:00

    write_to_csv('file2.csv', """
正常;2023-10-01 20:00:00
一小时以内延迟到达;2023-10-01 10:00:00
一小时以内延迟到达;2023-10-01 10:50:00
""")
```

SL16: 8.6.5 处理迟到数据的实例



```
    # 处理当前数据
    query.processAllAvailable()

    # 此时事件时间更新到上次看到的最大的2023-10-01 20:00:00

    write_to_csv('file3.csv', """
正常;2023-10-01 20:00:00
一小时以外延迟到达;2023-10-01 10:00:00
一小时以外延迟到达;2023-10-01 10:50:00
一小时以内延迟到达;2023-10-01 19:00:00
""")
```

SL17: 8.6.5 处理迟到数据的实例



```
    # 处理当前数据
    query.processAllAvailable()

    query.stop()

    test_tearDown()
```

SL1 2 ：8.6.5 处理迟到数据的实例



```
/home/hadoop/anaconda3/envs/nanyang/bin/python /home/hadoop/PycharmProject
-------------------------------------------
Batch: 0
-------------------------------------------
+----+--------------------------------------------+-----+
|word|window                                      |count|
+----+--------------------------------------------+-----+
|正常|{2023-10-01 08:00:00, 2023-10-01 09:00:00}|3    |
+----+--------------------------------------------+-----+


-------------------------------------------
Batch: 1
-------------------------------------------
+----+------+-----+
|word|window|count|
+----+------+-----+
+----+------+-----+
```

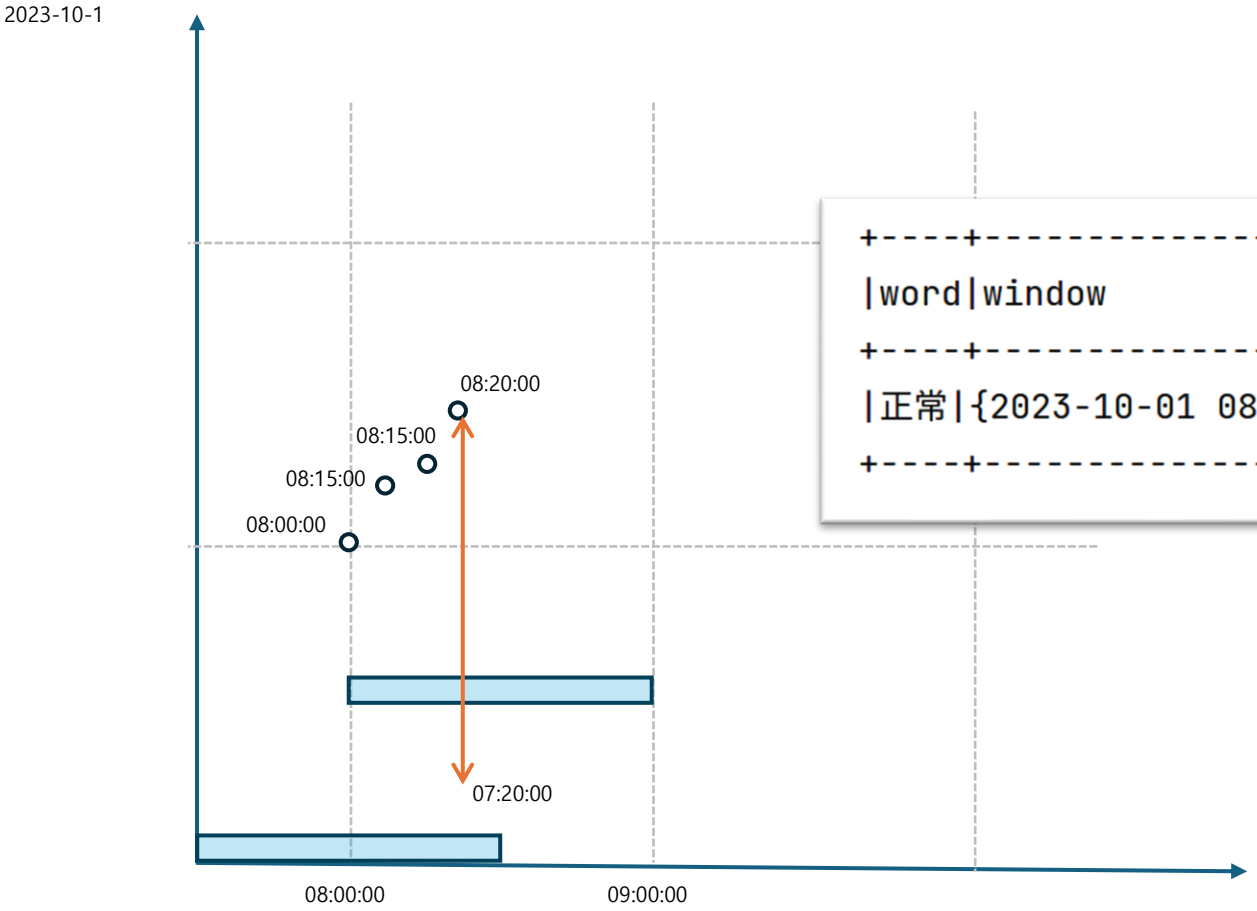
```
-------------------------------------------
Batch: 2
-------------------------------------------
+------------------+--------------------------------------------+-----+
|word              |window                                      |count|
+------------------+--------------------------------------------+-----+
|正常              |{2023-10-01 20:00:00, 2023-10-01 21:00:00}|1    |
|一小时以内延迟到达|{2023-10-01 10:00:00, 2023-10-01 11:00:00}|2    |
+------------------+--------------------------------------------+-----+


-------------------------------------------
Batch: 3
-------------------------------------------
+----+------+-----+
|word|window|count|
+----+------+-----+
+----+------+-----+


-------------------------------------------
Batch: 4
-------------------------------------------
+------------------+--------------------------------------------+-----+
|word              |window                                      |count|
+------------------+--------------------------------------------+-----+
|正常              |{2023-10-01 20:00:00, 2023-10-01 21:00:00}|2    |
|一小时以内延迟到达|{2023-10-01 19:00:00, 2023-10-01 20:00:00}|1    |
+------------------+--------------------------------------------+-----+
```

Batch 0



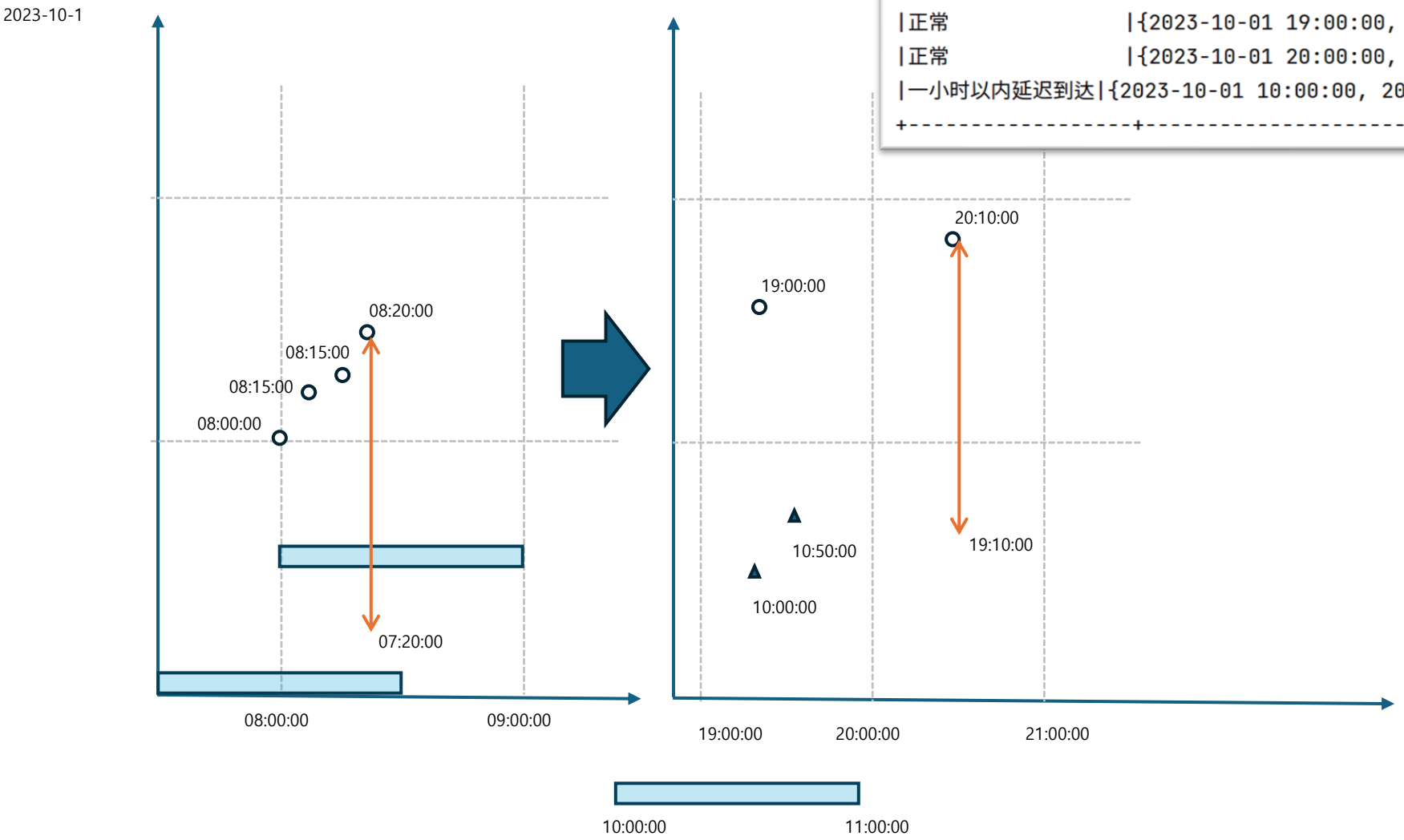



```
+----+-----------------------------------------+-----+
|word|window                                   |count|
+----+-----------------------------------------+-----+
|正常|{2023-10-01 08:00:00, 2023-10-01 09:00:00}|4    |
+----+-----------------------------------------+-----+
```

Batch 1



```
+------------------+-----------------------------------------+-----+
|word              |window                                   |count|
+------------------+-----------------------------------------+-----+
|正常              |{2023-10-01 19:00:00, 2023-10-01 20:00:00}|1    |
|正常              |{2023-10-01 20:00:00, 2023-10-01 21:00:00}|1    |
|一小时以内延迟到达|{2023-10-01 10:00:00, 2023-10-01 11:00:00}|2    |
+------------------+-----------------------------------------+-----+
```



Batch 2



```
-------------------------------------------
Batch: 4
-------------------------------------------
+------------------+------------------------------------------+-----+
|word              |window                                    |count|
+------------------+------------------------------------------+-----+
|正常              |{2023-10-01 19:00:00, 2023-10-01 20:00:00}|2    |
|正常              |{2023-10-01 20:00:00, 2023-10-01 21:00:00}|3    |
|一小时以内延迟到达|{2023-10-01 19:00:00, 2023-10-01 20:00:00}|2    |
+------------------+------------------------------------------+-----+
```

