**Difference Between Using Watermark vs Without Watermark**

**Setup**

The core difference is **how Spark handles late data**:

* **Without watermark**: All late data is kept indefinitely → memory grows, re-processing continues.
* **With watermark**: Spark **drops data older than the watermark threshold**, which makes it more memory-efficient and avoids redundant aggregations.

**Code Comparison**

**Without Watermark**

python

# Aggregation without watermark

no\_watermark\_df = parking\_df \

.groupBy("terminal", "zone", "status", window("event\_time", "1 minute")) \

.agg(count("\*").alias("status\_count")) \

.filter("status = 'Occupied'")

**With Watermark**

python

# Aggregation with watermark of 2 minutes

with\_watermark\_df = parking\_df \

.withWatermark("event\_time", "2 minutes") \

.groupBy("terminal", "zone", "status", window("event\_time", "1 minute")) \

.agg(count("\*").alias("status\_count")) \

.filter("status = 'Occupied'")

**Simulated Output Difference**

Assume 5 sample events arrived (1 every second):

| **timestamp** | **value** | **terminal** | **zone** | **status** | **event\_time** |
| --- | --- | --- | --- | --- | --- |
| 2025-07-28 15:00:01 | 0 | A | P1 | Occupied | 2025-07-28 15:00:01 |
| 2025-07-28 15:00:02 | 1 | B | P2 | Available | 2025-07-28 15:00:02 |
| 2025-07-28 15:00:03 | 2 | C | P3 | Available | 2025-07-28 15:00:03 |
| 2025-07-28 15:00:04 | 3 | A | P4 | Occupied | 2025-07-28 15:00:04 |
| 2025-07-28 15:00:05 | 4 | B | P5 | Available | 2025-07-28 15:00:05 |

**Console Output (Streaming)**

**Without Watermark (after 15 seconds):**

+---------+-----+--------+------------------------------------------+

|terminal |zone |status\_count|window |

+---------+-----+--------+------------------------------------------+

|A |P1 |1 |{2025-07-28 15:00:00, 2025-07-28 15:01:00}|

|A |P4 |1 |{2025-07-28 15:00:00, 2025-07-28 15:01:00}|

+---------+-----+------------+--------------------------------------+

**With Watermark (2 minutes) — at first, output is same:**

+---------+-----+--------+------------------------------------------+

|terminal |zone |status\_count|window |

+---------+-----+--------+------------------------------------------+

|A |P1 |1 |{2025-07-28 15:00:00, 2025-07-28 15:01:00}|

|A |P4 |1 |{2025-07-28 15:00:00, 2025-07-28 15:01:00}|

+---------+-----+------------+--------------------------------------+

**Now Wait 3 Minutes and Send Late Data**

You insert a late record:

python

+---------+-----+--------+-----------------------------+

|timestamp | value | terminal | zone | status |

|2025-07-28 14:57:59 | 999 | C | P2 | Occupied |

**Without Watermark:**

* This late data will still be **aggregated and show up** in old windows, even if the event is >5 minutes old.

**With Watermark:**

* Spark will **drop this record** because it's older than max(event\_time seen) - 2 minutes.

**Summary Table**

| **Behavior** | **Without Watermark** | **With Watermark ("2 min")** |
| --- | --- | --- |
| Late event (3+ mins old) | Included in aggregation | Dropped (not processed) |
| Aggregation update window | Updates old windows forever | Drops data once window + watermark is passed |
| Memory use | Higher (stores all state) | Lower (cleans state after watermark) |
| Suitable for | Exact replay, debug | Production workloads with late data cutoff |

**Use Watermark When:**

* You expect late events but want to **limit how long you wait**.
* You care about **bounded memory usage**.
* You want reliable **event-time-based processing**.