**Windows and Watermark**

**Events in Our Example**

* **E1:** Event time = 9:00:10, arrival = 9:00:10
* **E2:** Event time = 9:00:40, arrival = 9:00:40
* **E3:** Event time = 9:00:20, arrival = 9:00:50 (**out-of-order**)
* **E4 (D):** Event time = 9:00:50, arrival = 9:01:20 (**delayed**)

Window = **[9:00–9:01)**.  
Watermark rule = **current max event time − 5s**.

**How the Watermark Works Step by Step**

1. **After E1 (9:00:10):**
   * Max event time seen = 9:00:10
   * Watermark = 9:00:05
   * [9:00–9:01) window is open
2. **After E2 (9:00:40):**
   * Max event time seen = 9:00:40
   * Watermark = 9:00:35
   * Flink is still waiting for late events up to 9:00:35
   * [9:00–9:01) window is open
3. **After E3 (9:00:20, late but still < watermark):**
   * Max event time still = 9:00:40
   * Watermark = 9:00:35
   * Since 9:00:20 < 9:00:35, this event is still accepted into [9:00–9:01)
4. **At 9:01:20, E4 (event time 9:00:50) arrives:**
   * Max event time now = 9:00:50
   * Watermark = 9:00:45
   * Window [9:00–9:01) only closes once watermark passes **9:01:00**
   * Since watermark (9:00:45) is still less than 9:01:00, the window is **still open**
   * So E4 is placed in [9:00–9:01)

**Why This Matters**

* **Without watermark:** Flink would just use arrival time (9:01:20) and wrongly put E4 into [9:01–9:02).
* **With watermark:** Flink waits for a while before closing [9:00–9:01).
  + That waiting (grace period) allows E4 to still be counted in the right window.

**Sequence of arrivals (showing the critical ordering where O9 arrives before O7)**

1. **Arrival: O1 (event\_time 09:01)**
   * max\_event\_time\_seen = 09:01
   * watermark = 09:01 − 5m = 08:56
   * Window B end = 09:30 → watermark (08:56) **< 09:30** → **Window B is OPEN**.
2. **Arrival: O2 (09:06)**
   * max\_event\_time\_seen = 09:06
   * watermark = 09:06 − 5m = 09:01
   * **Window B still OPEN**.
3. **Arrival: O3 (09:08)**
   * max\_event\_time\_seen = 09:08
   * watermark = 09:08 − 5m = 09:03
   * **Window B still OPEN**.
4. **Arrival: O4 (09:17)**
   * max\_event\_time\_seen = 09:17
   * watermark = 09:17 − 5m = 09:12
   * **Window B still OPEN** (because 09:12 < 09:30).
5. **Arrival: O5 (09:18)**
   * max\_event\_time\_seen = 09:18
   * watermark = 09:18 − 5m = 09:13
   * **Window B still OPEN**.
6. **Arrival: O6 (09:20)**
   * max\_event\_time\_seen = 09:20
   * watermark = 09:20 − 5m = 09:15
   * **Window B still OPEN** (watermark = 09:15 < 09:30).
   * At this point Window A (09:00–09:15) would have been closed (since watermark ≥ 09:15), but Window B remains open.
7. **Arrival: O8 (09:33)** *(notice O8 arrives before O9 and O7 in this scenario)*
   * max\_event\_time\_seen = 09:33
   * watermark = 09:33 − 5m = 09:28
   * **Window B still OPEN** (09:28 < 09:30).
8. **Arrival: O9 (09:36)** *(this is the critical event that advances watermark past Window B end)*
   * max\_event\_time\_seen = 09:36
   * watermark = 09:36 − 5m = 09:31
   * Now watermark = 09:31 **≥ 09:30** → **Flink closes Window B** (Window B is finalized/emitted at this moment).
   * Any subsequent event with event\_time < 09:30 will be considered **late for Window B**.
9. **Late Arrival: O7 (event\_time 09:29) arrives now (after O9)**
   * Event O7 belongs by *event\_time* to **Window B**.
   * But at its arrival the **current watermark = 09:31**, which is **after Window B end (09:30)**.
   * Therefore O7 is **late** for Window B.
   * What Flink does with O7 depends on configuration:
     + **Default behavior:** O7 is dropped (ignored) for Window B.
     + **If allowedLateness is configured:** O7 could still be accepted into Window B if it arrived within the allowed lateness period.
     + **If side-output for late events is configured:** O7 is sent to the late-events side output for special handling (e.g., logging, manual backfill, or separate aggregation).

**Window boundaries (15-minute tumbling windows)**

* Window A: **09:00 – 09:15** (end = 09:15)
* Window B: **09:15 – 09:30** (end = 09:30)
* Window C: **09:30 – 09:45** (end = 09:45)

Events (their *event\_time* from your sample):

* O1 → 09:01
* O2 → 09:06
* O3 → 09:08
* O4 → 09:17
* O5 → 09:18
* O6 → 09:20
* O7 → 09:29 ← this is the one that should belong to Window B
* O8 → 09:33

*(To show the cause of O7 being late, I’ll add one later event O9 with event\_time 09:36 — this models the situation where a later event makes the watermark advance past 09:30 before O7 arrives.)*

**Step 1: The Normal Join (Shuffle Join)**

Imagine you have **3 Flink tasks (workers)**, and data is split based on the join key (user\_id).

* **Clicks stream**: huge, millions of events.
* **Profiles table**: small, only a few rows.

**Example**

Clicks stream (big):

| **user\_id** | **click\_event** |
| --- | --- |
| u1 | click\_A |
| u1 | click\_B |
| u1 | click\_C |
| u2 | click\_D |

Profiles (small):

| **user\_id** | **user\_name** |
| --- | --- |
| u1 | Alice |
| u2 | Bob |

**How Shuffle Join works**

* Flink sends all rows with the same user\_id **to the same task**.

| **Task** | **Gets Clicks** | **Gets Profiles** | **Joins Produced** |
| --- | --- | --- | --- |
| Task 1 | u1 → click\_A, click\_B, click\_C | u1 → Alice | (u1, Alice, click\_A), (u1, Alice, click\_B), (u1, Alice, click\_C) |
| Task 2 | u2 → click\_D | u2 → Bob | (u2, Bob, click\_D) |
| Task 3 | nothing | nothing | nothing |

Problem: **Task 1 is overloaded** because most clicks are u1. This is **data skew**.

**Step 2: The Broadcast Join**

Instead of shuffling both sides, we tell Flink:  
 “The small table (profiles) should be **copied to every task**.”

**How Broadcast Join works**

* Flink **replicates the small side** (profiles) to **all tasks**.
* The big stream (clicks) is distributed normally (round-robin or partitioned by user).

| **Task** | **Gets Clicks** | **Gets Full Profiles Copy** | **Joins Produced** |
| --- | --- | --- | --- |
| Task 1 | u1 → click\_A | (u1 → Alice, u2 → Bob) | (u1, Alice, click\_A) |
| Task 2 | u1 → click\_B, u2 → click\_D | (u1 → Alice, u2 → Bob) | (u1, Alice, click\_B), (u2, Bob, click\_D) |
| Task 3 | u1 → click\_C | (u1 → Alice, u2 → Bob) | (u1, Alice, click\_C) |

**What Changed?**

* Before: all u1 clicks had to go to **Task 1** → overload.
* Now: u1 clicks are **split across tasks** (Task 1, Task 2, Task 3).
* Each task has a **full copy** of profiles, so they can still join correctly.