IMEN891K / IDSC501 / AXGS-501 · DATA ENGINEERING · FALL 2026

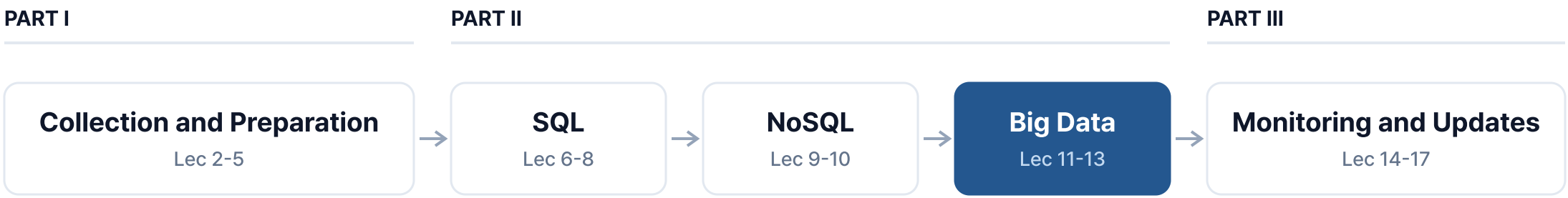
13.

Big Data III: Stream Processing

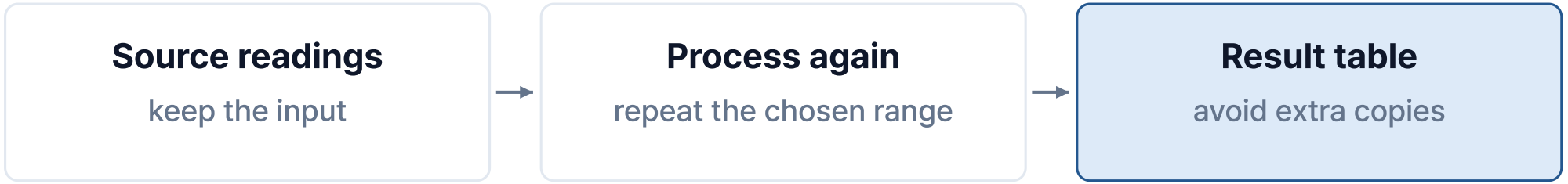
Sangwoo Mo · POSTECH

Industrial & Management Engineering

Where We Are


PART I
PART II
PART III
Collection and Preparation
Lec 2-5
SQL
Lec 6-8
NoSQL
Lec 9-10
Big Data
Lec 11-13
Monitoring and Updates
Lec 14-17

Recap: A Result That Can Be Rebuilt



Reading the same input again should preserve the intended result

Outline: Big Data III

01	**Recording Events**	why streams need a retained event log
02	**Reading and Restarting**	remember a position and handle repeated readings
03	**Handling Late Arrivals**	use the measurement time to update the right result

01 · Recording Events

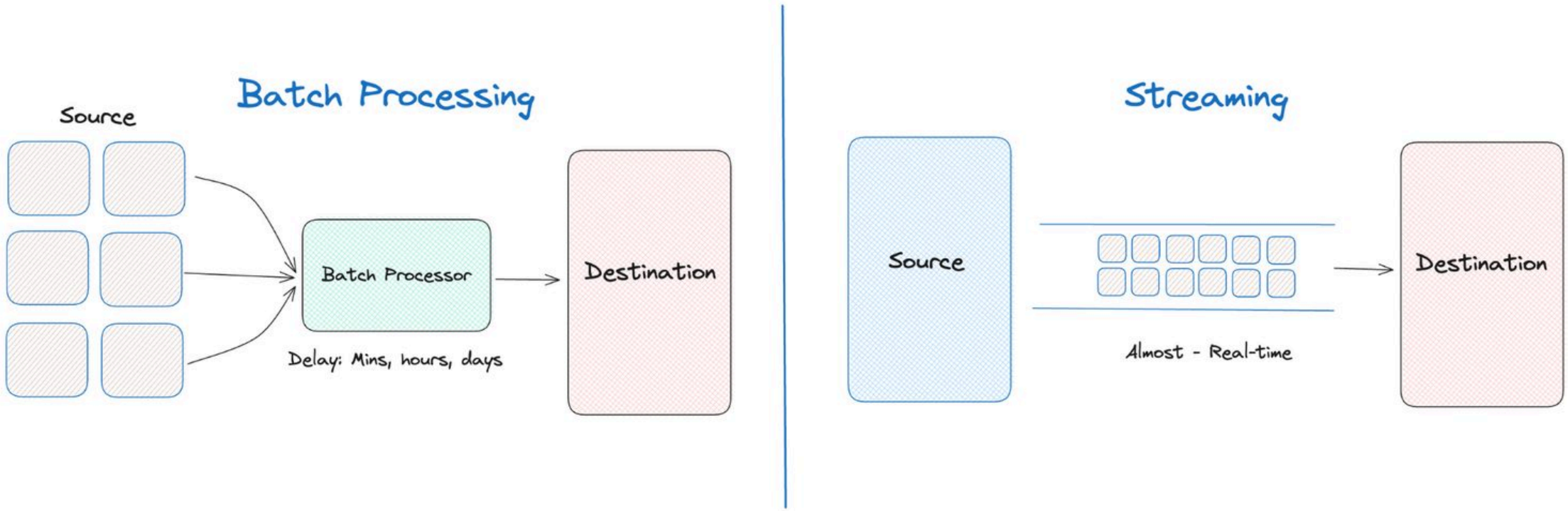
01 **Recording Events** — why streams need a retained event log

02 **Reading and Restarting** — remember a position and handle repeated readings

03 **Handling Late Arrivals** — use the measurement time to update the right result

Batch and Stream Processing

Batch processes a bounded set; streaming updates results as more events arrive

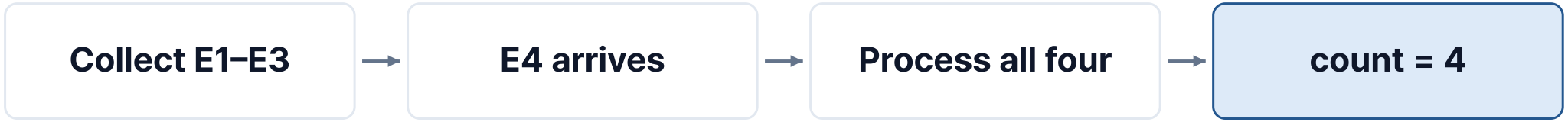


The difference is how ongoing input becomes an updated result

The Same Readings, Different Updates

The same count can be computed after a batch or updated during a stream

Batch: wait, then process the set



Stream: update as records arrive



Streaming makes a current result available before the input stops

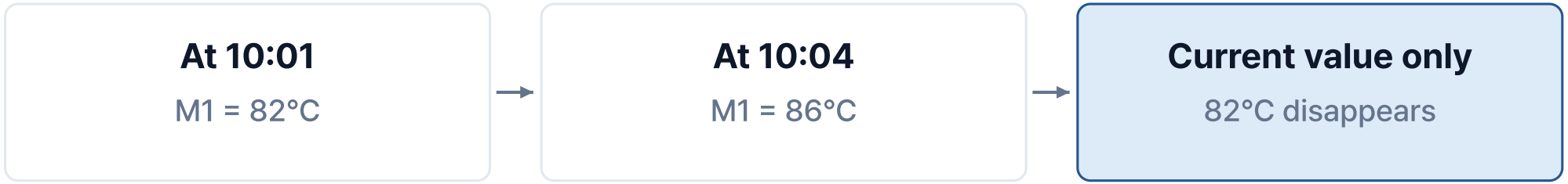
Four Readings from One Machine

Each row is one M1 reading; count readings at or above 80°C

Reading	Measured at	Arrived at	Temperature
E1	10:01	10:01	82°C
E2	10:04	10:04	86°C
E3	10:06	10:06	90°C
E4	10:03	10:07	84°C

E4 describes an earlier measurement that arrives last

A Current Value Loses the History



Overwriting one current value loses the earlier reading

One Event, One Reading

An event records one occurrence with its own identity

Field	E1 value	Meaning
event_id	E1	This measurement
machine_id	M1	The machine
measured_at	10:01	When it happened
temperature	82°C	The measured value

E1 identifies the reading; M1 identifies its machine

The First Record in a Log

A log appends incoming events in order

Stored M1 readings

E1 / M1
10:01 / 82°C
position 0

E1 occupies position 0 in the stored log

New Readings Append at the End

Stored M1 readings

E1 / M1	E2 / M1	E3 / M1
10:01 / 82°C	10:04 / 86°C	10:06 / 90°C
position 0	**position 1**	**position 2**

E1 remains stored when E2 and E3 append

A Late Reading Still Appends Last

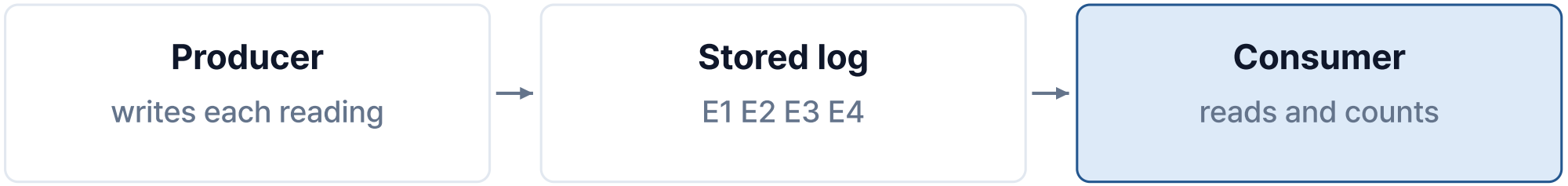
Stored M1 readings

E1 / M1	E2 / M1	E3 / M1	E4 / M1
10:01 / 82°C	10:04 / 86°C	10:06 / 90°C	10:03 / 84°C
position 0	**position 1**	**position 2**	**position 3**

E4 receives position 3 even though it happened before E3

Producer, Log and Consumer

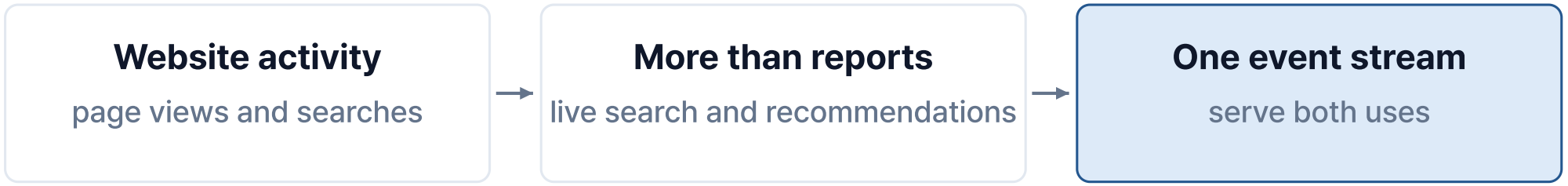
The producer writes events; the consumer reads them



Writing a reading and processing its result are separate steps

LinkedIn Needed Fresh Activity Data

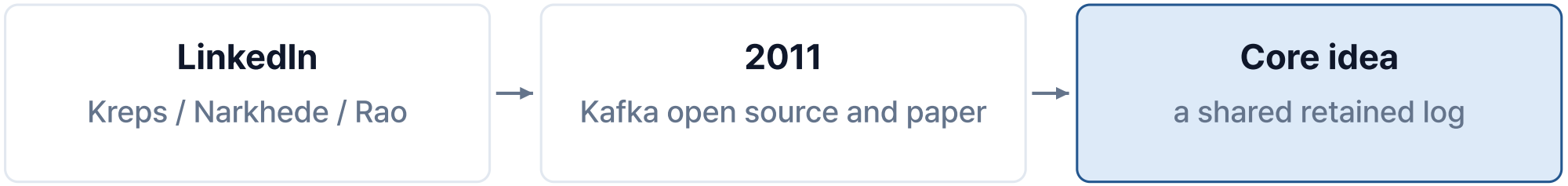
Activity logs became inputs to live services as well as later analysis



The same event data had to reach both live services and batch analysis

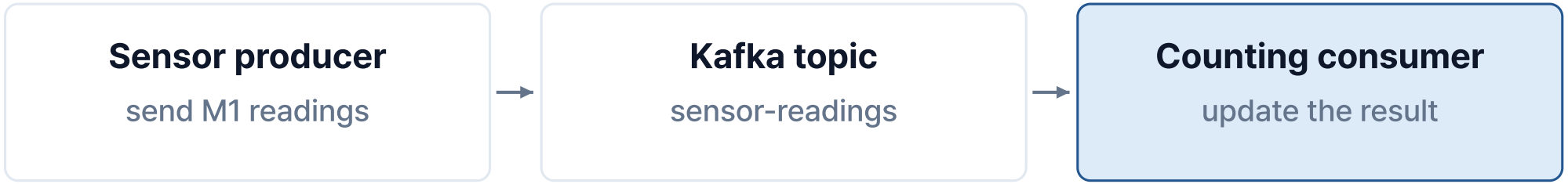
Kafka Begins at LinkedIn

Kafka was open-sourced in 2011 to deliver activity records to online and offline systems



A shared log connected the activity producers to different readers

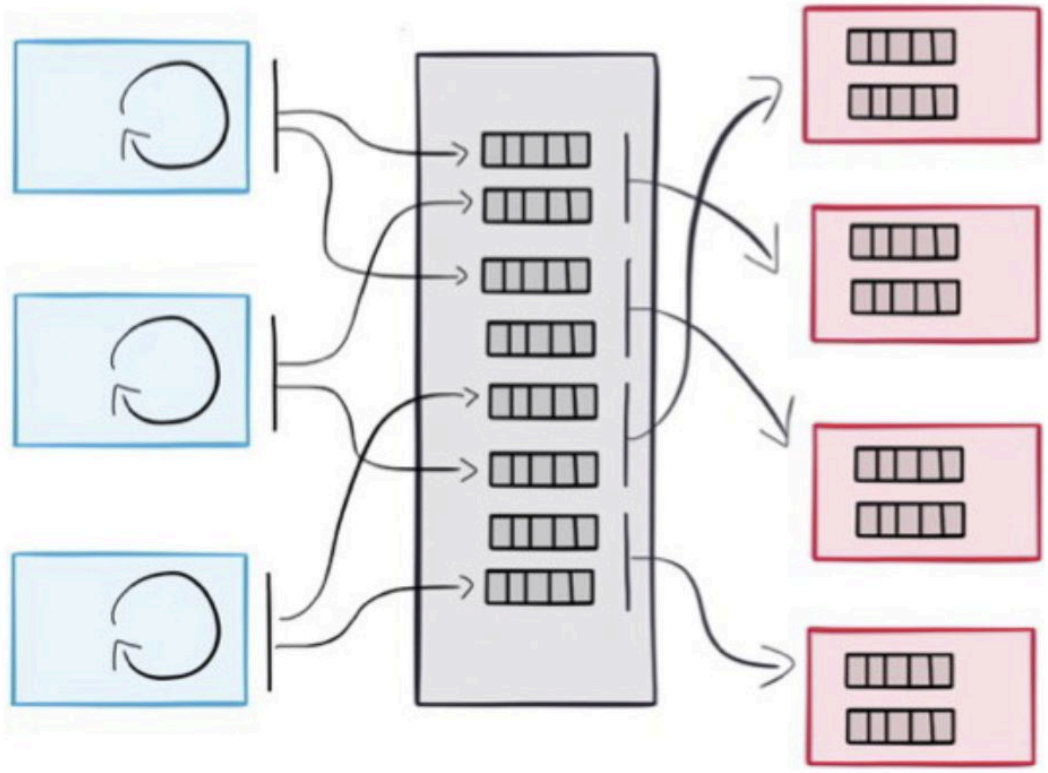
Kafka Holds the Event Log



The topic gives the event stream a name

Decoupling Writers and Readers

A retained log lets writers and readers work at different times



A new reader can use the log without adding a new producer connection

Two Readers, Two Positions

Independent readers can consume the same retained events at their own pace

The same retained log: E1 E2 E3 E4

Live display

E1 E2 E3 / count 3

Read next: 3

Daily report

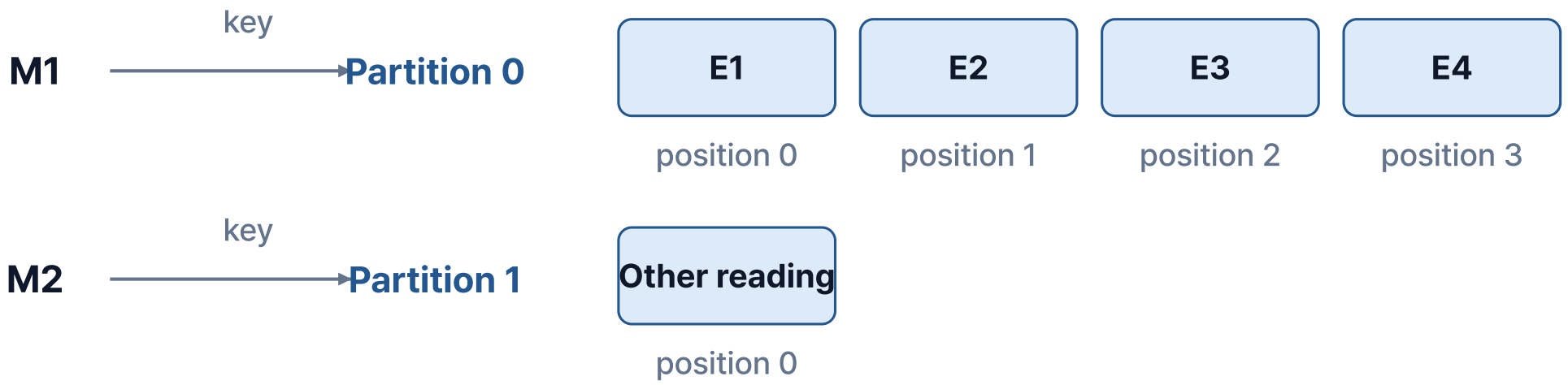
E1 / count 1

Read next: 1

The display resumes at E4; the report resumes at E2

Partitions Split a Topic into Logs

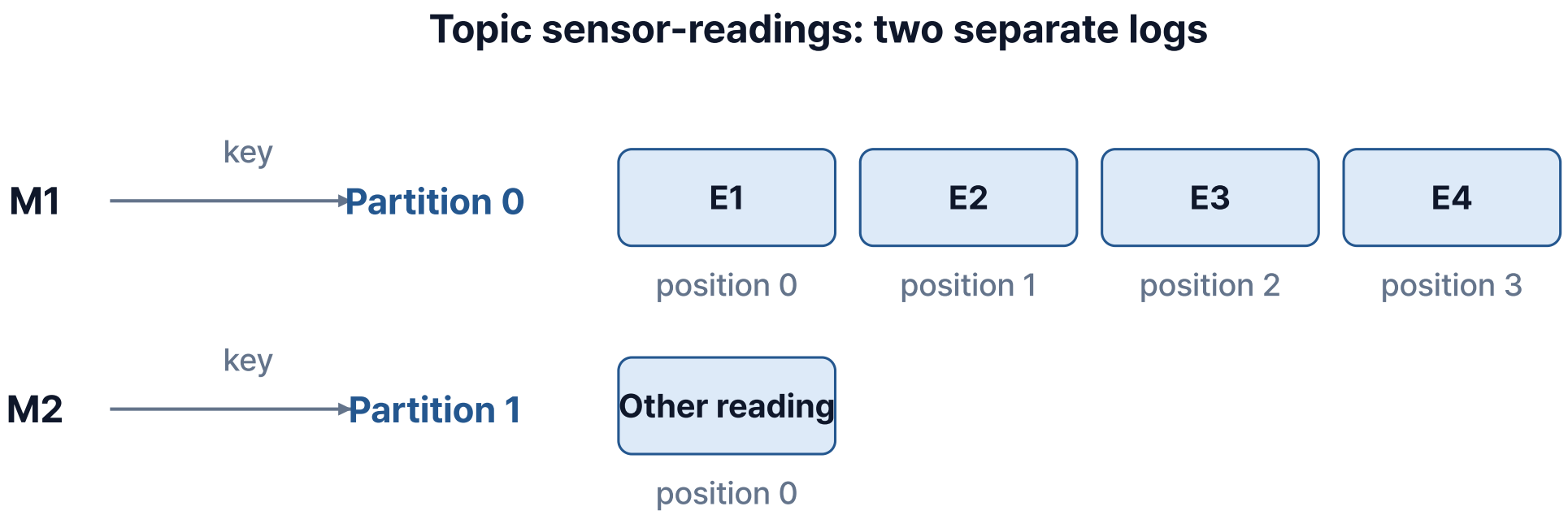
A partition is one ordered log inside a topic



M1 readings stay together in partition 0

A Key Keeps One Machine Together

Here the record key is the machine ID



The fixed mapping sends every M1 reading to the same log

An Offset Names a Position

Offsets belong to one partition

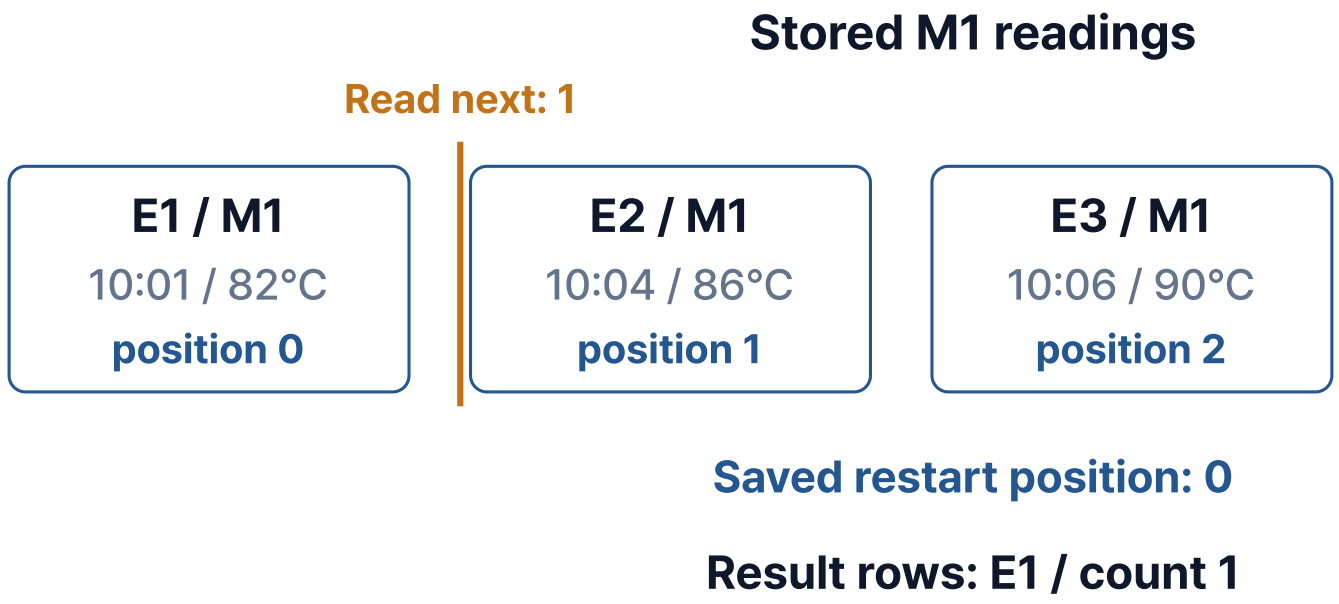
Partition	Offset	Stored reading
0	0	E1 from M1
0	1	E2 from M1
1	0	A reading from M2

Offset 0 in two partitions identifies two different records

02 · Reading and Restarting

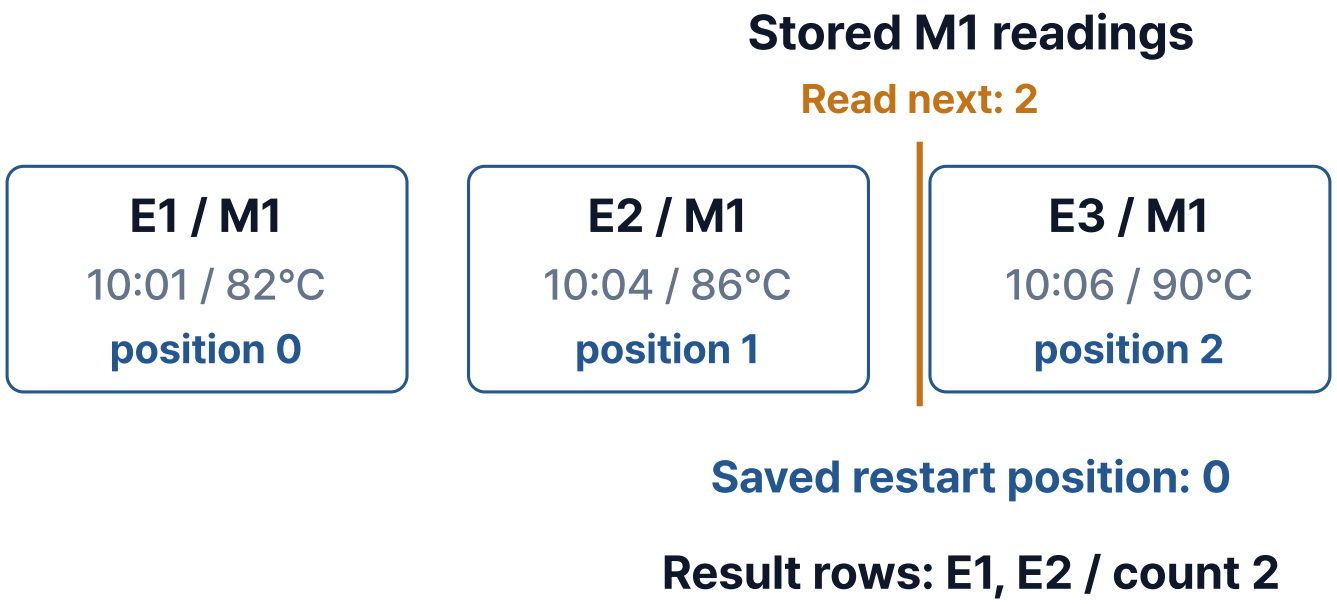
01	Recording Events	why streams need a retained event log
02	**Reading and Restarting**	remember a position and handle repeated readings
03	Handling Late Arrivals	use the measurement time to update the right result

Reading the First Record



After processing E1, the next read position becomes 1

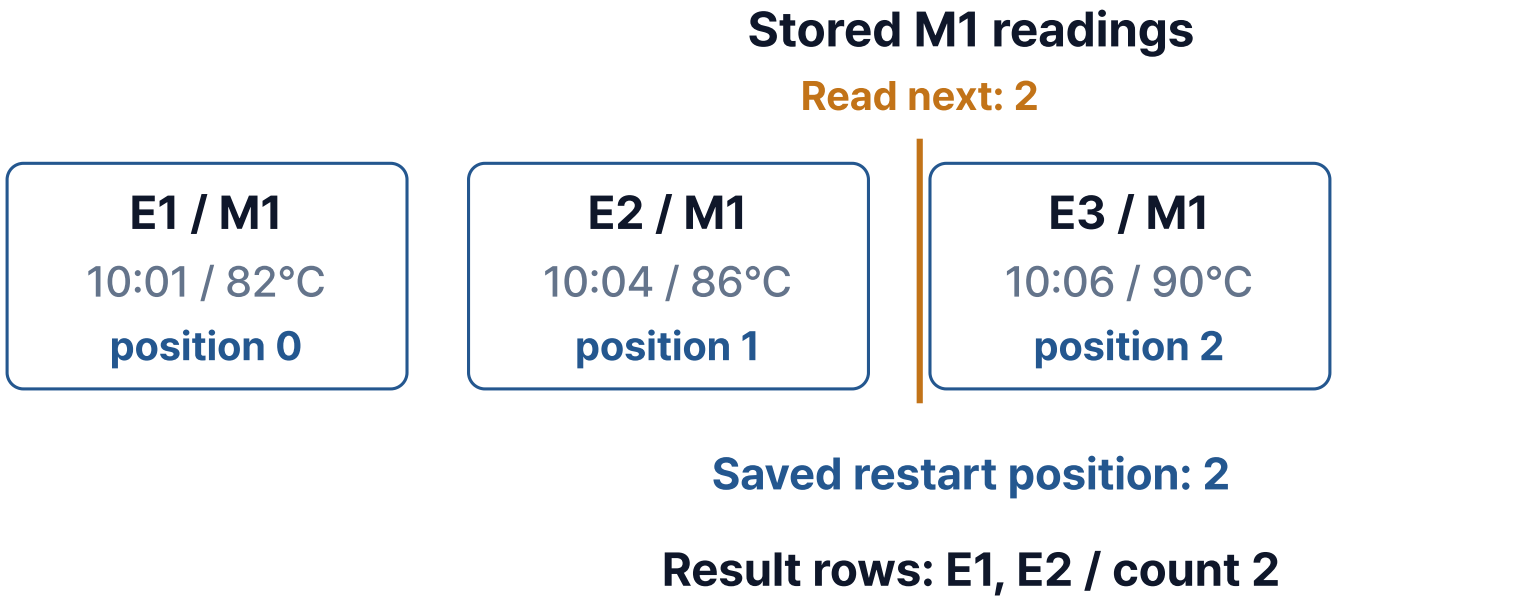
Reading the Second Record



Two processed readings produce two result rows

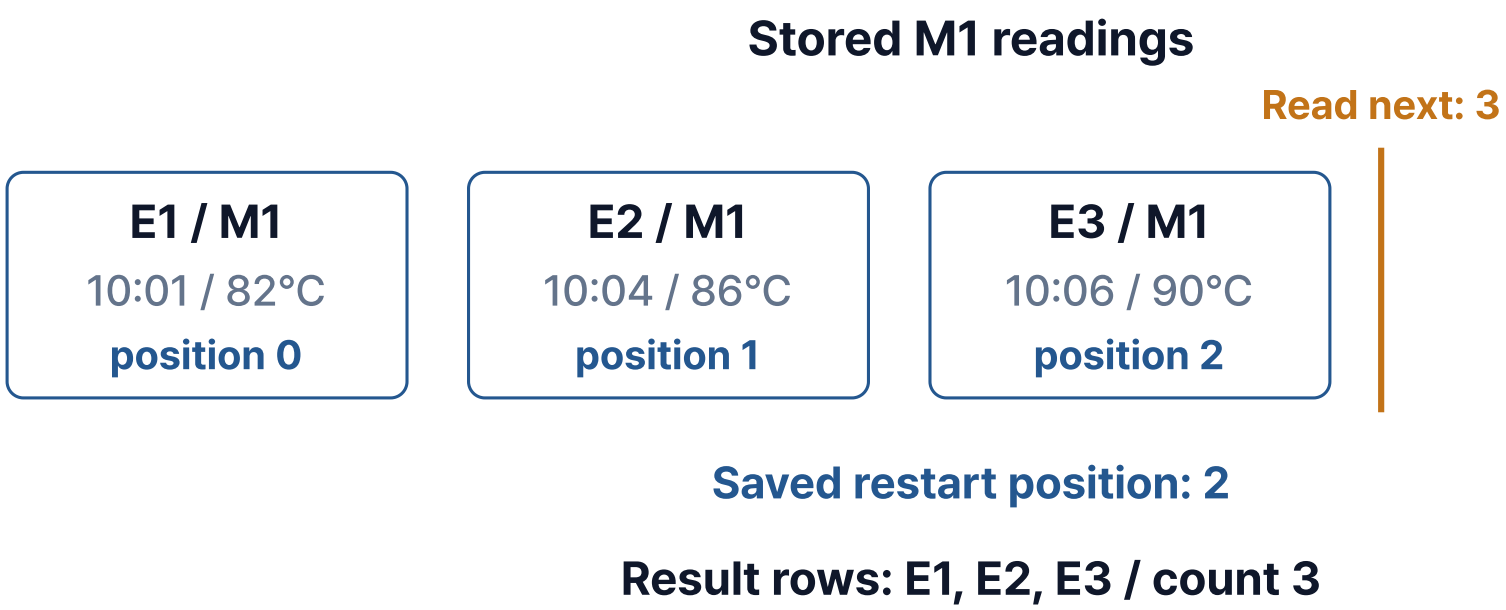
Saving the Restart Position

Committing an offset saves where a reader should resume



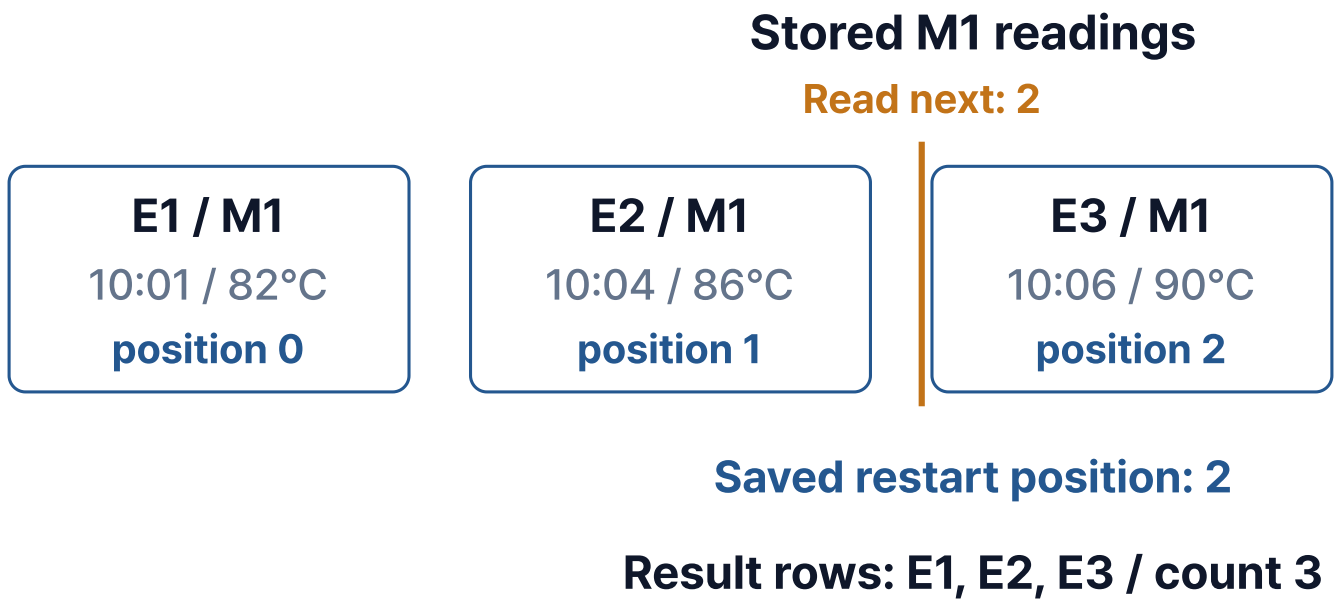
Saved position 2 means E3 is the next record after restart

A Failure Before Saving the Position



E3 reaches the result, but saved position remains 2

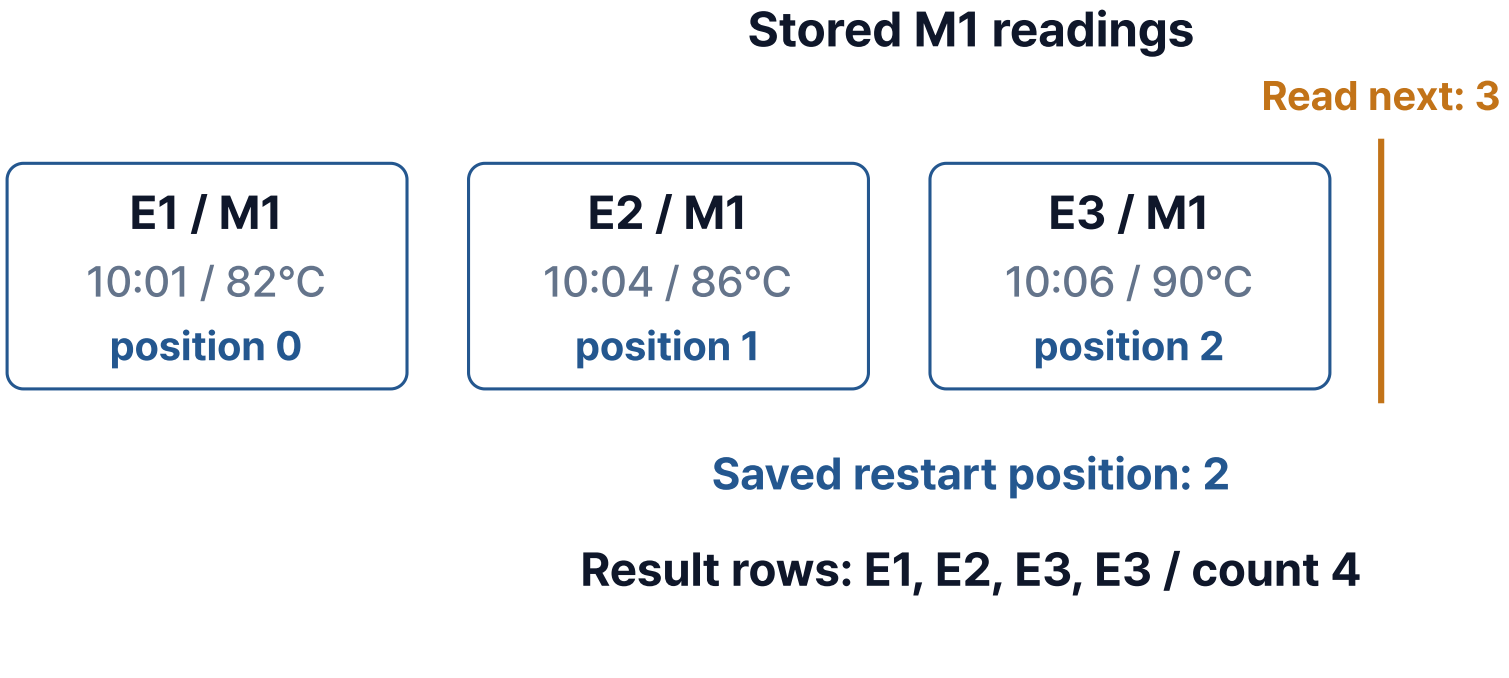
Restarting from the Saved Position



The restarted reader returns to E3 at offset 2

The Same Reading Can Appear Twice

Replay means reading stored records again



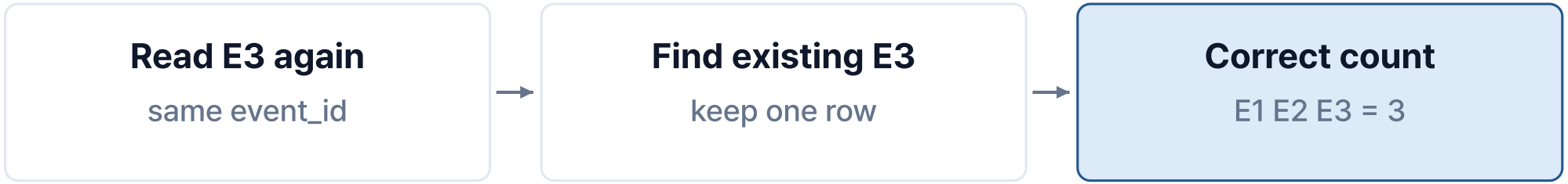
Three distinct readings can produce four rows after replay

Reading ID and Machine ID

Stored input	Machine	Event identity
First measurement	M1	E1
Second measurement	M1	E2
E3 replay	M1	E3 again

Different M1 events survive; a second copy of E3 does not

Keeping One Result Row per Event



A unique event ID keeps replay from adding a second row

Demo: Read, Save and Restart

Predict the result after restart, then repeat with unique event IDs

Read Next | Save Position | Restart | Reset | ☐ Ignore repeated event IDs

Simulation: read E1 and E2, save position, read E3, restart, then read E3 again

Read next: 0 / saved restart position: 0

E1 / M1	E2 / M1	E3 / M1	E4 / M1
offset 0	offset 1	offset 2	offset 3

`Result rows: none / count 0`

`Read E1 first`

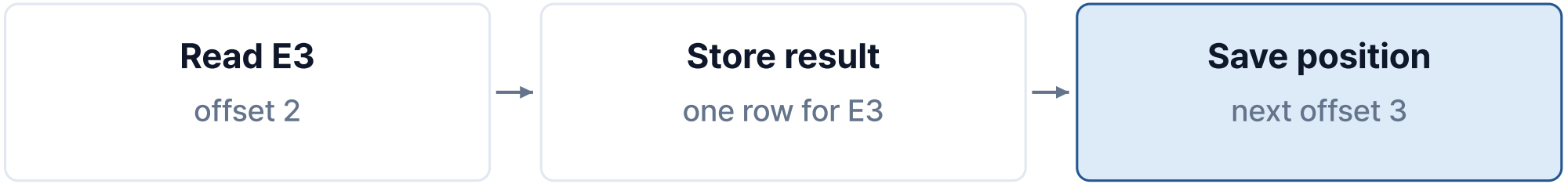
Compare result rows before and after reading E3 a second time

Answer: Read, Save and Restart

Action	Saved position	Result rows
Read E1 and E2; save	2	E1 E2
Read E3; restart	2	E1 E2 E3
Read E3 again	2	E1 E2 E3 E3
Repeat with unique event IDs	2	E1 E2 E3

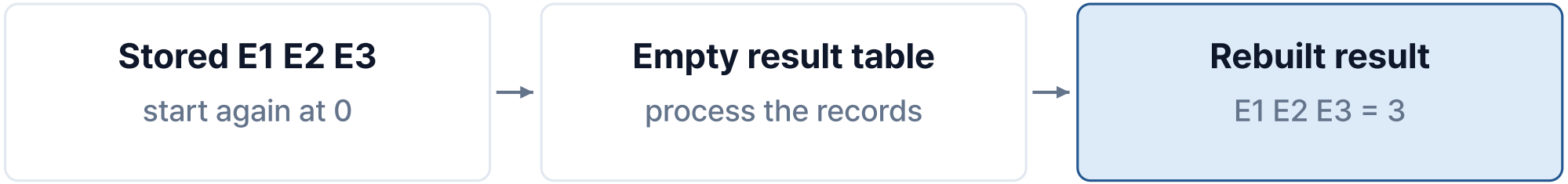
The saved position controls replay; event identity controls repeated result rows

Processing Before Saving Progress



Saving progress after the result leaves replay possible during a failure

Rebuilding a Result from the Log



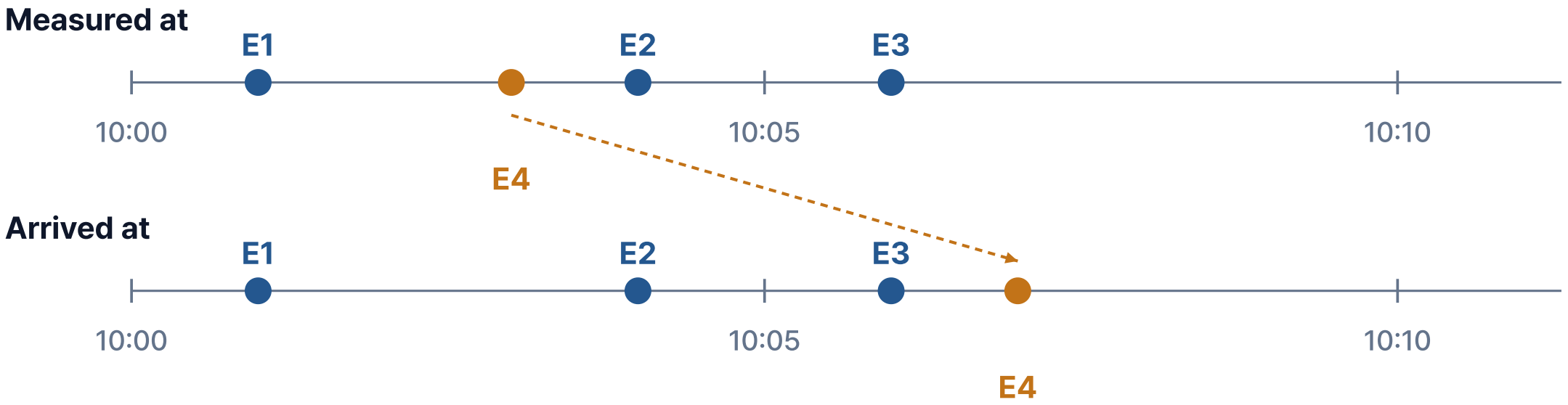
A retained log can rebuild a result from the same input

03 · Handling Late Arrivals

01	Recording Events	why streams need a retained event log
02	Reading and Restarting	remember a position and handle repeated readings
03	**Handling Late Arrivals**	use the measurement time to update the right result

One Reading Has Two Times

Event time is when the measurement happened



E4 happened at 10:03 and arrived at 10:07

Counting by Measurement Time

Count hot readings in each five-minute measurement interval

10:00 to before 10:05

E1, E2

count 2

10:05 to before 10:10

E3

count 1

Before E4 arrives, the earlier interval contains E1 and E2

A Late Arrival Updates the Earlier Interval

10:00 to before 10:05

E1, E2, E4

count 3

10:05 to before 10:10

E3

count 1

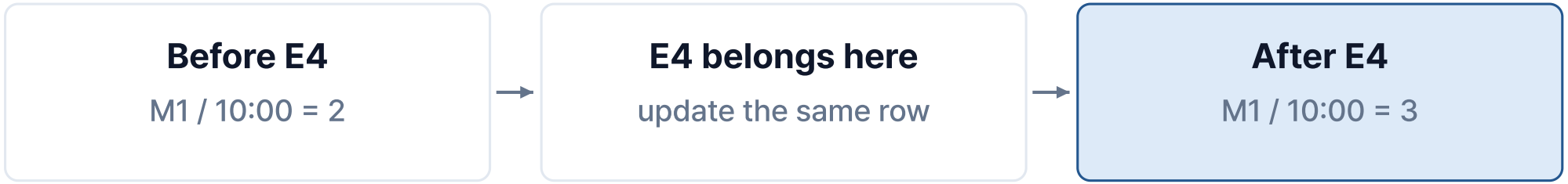
E4 changes the earlier count from 2 to 3

The Clock Changes the Answer

Time used	10:00 to before 10:05	10:05 to before 10:10
Measurement time	E1 E2 E4: 3	E3: 1
Arrival time	E1 E2: 2	E3 E4: 2

Measurement time and arrival time answer different questions

Replacing the Earlier Result



The revised count replaces 2 with 3 for the same interval

Takeaway: Stream Processing

1. **Streaming** updates ongoing input; a retained log can serve different readers
2. **Save the next position** to resume; use event identity to handle replay
3. **Measurement time** places a late reading in the right interval