State and Timers

Israel Herraiz

Strategic Cloud Engineer,
Google Cloud

Agenda

Course Intro

Beam Concepts Review

Windows, Watermarks, and Triggers

Sources and Sinks

Schemas

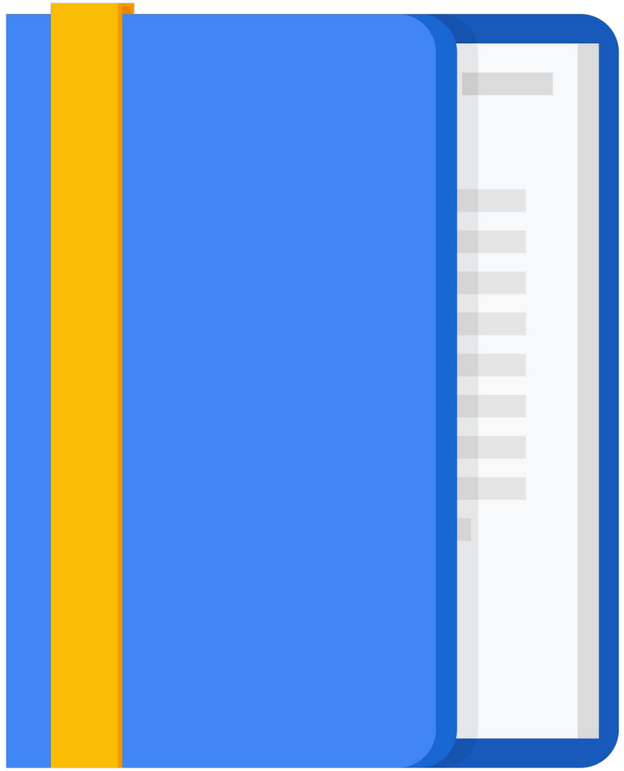
State and Timers

Best Practices

Dataflow SQL and DataFrames

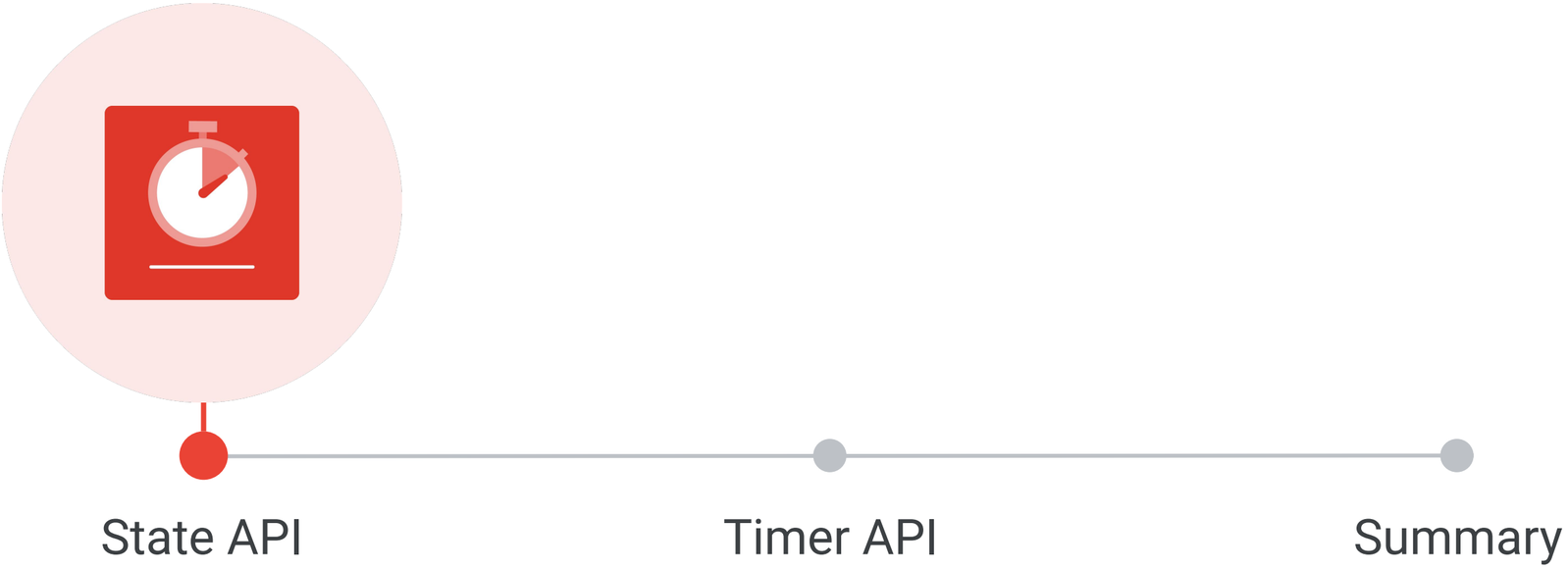
Beam Notebooks

Summary

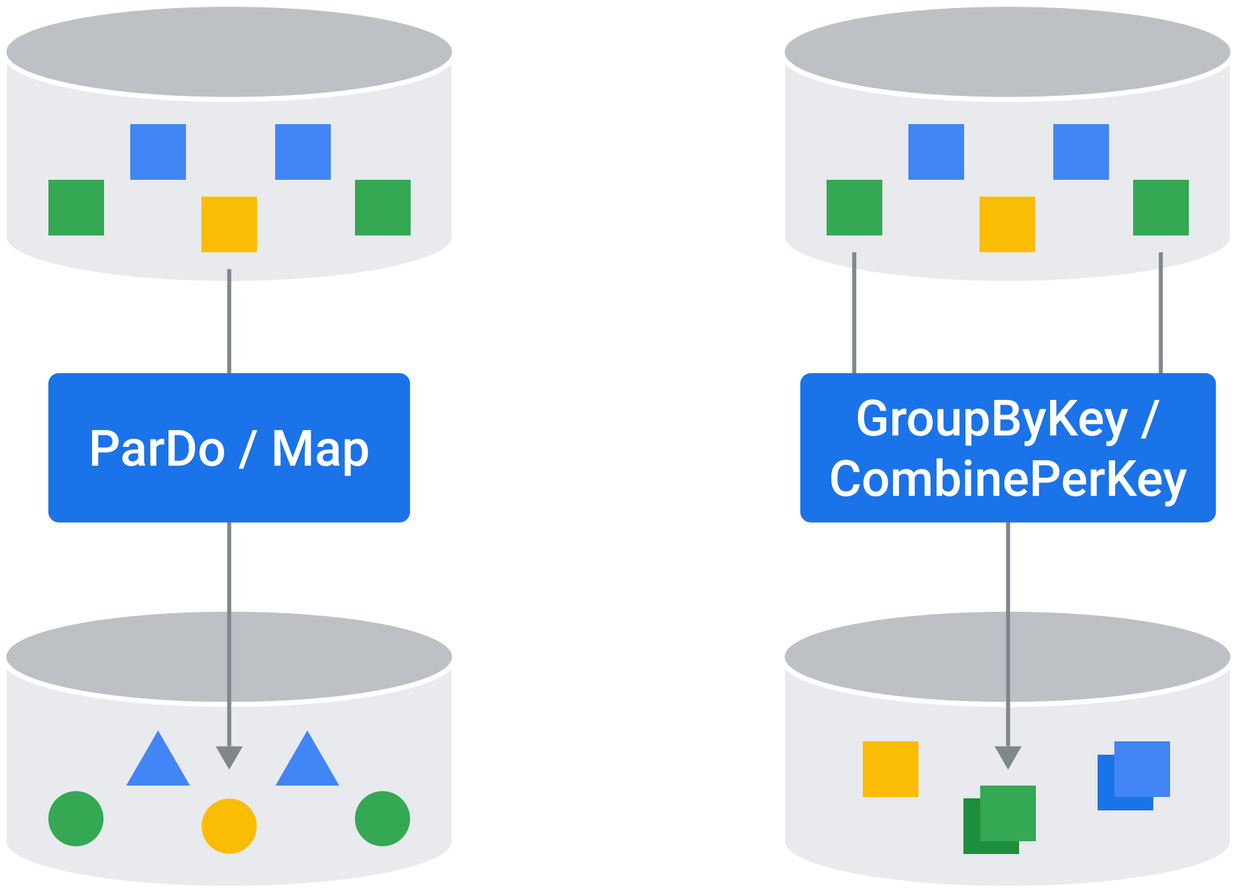


State and Timers

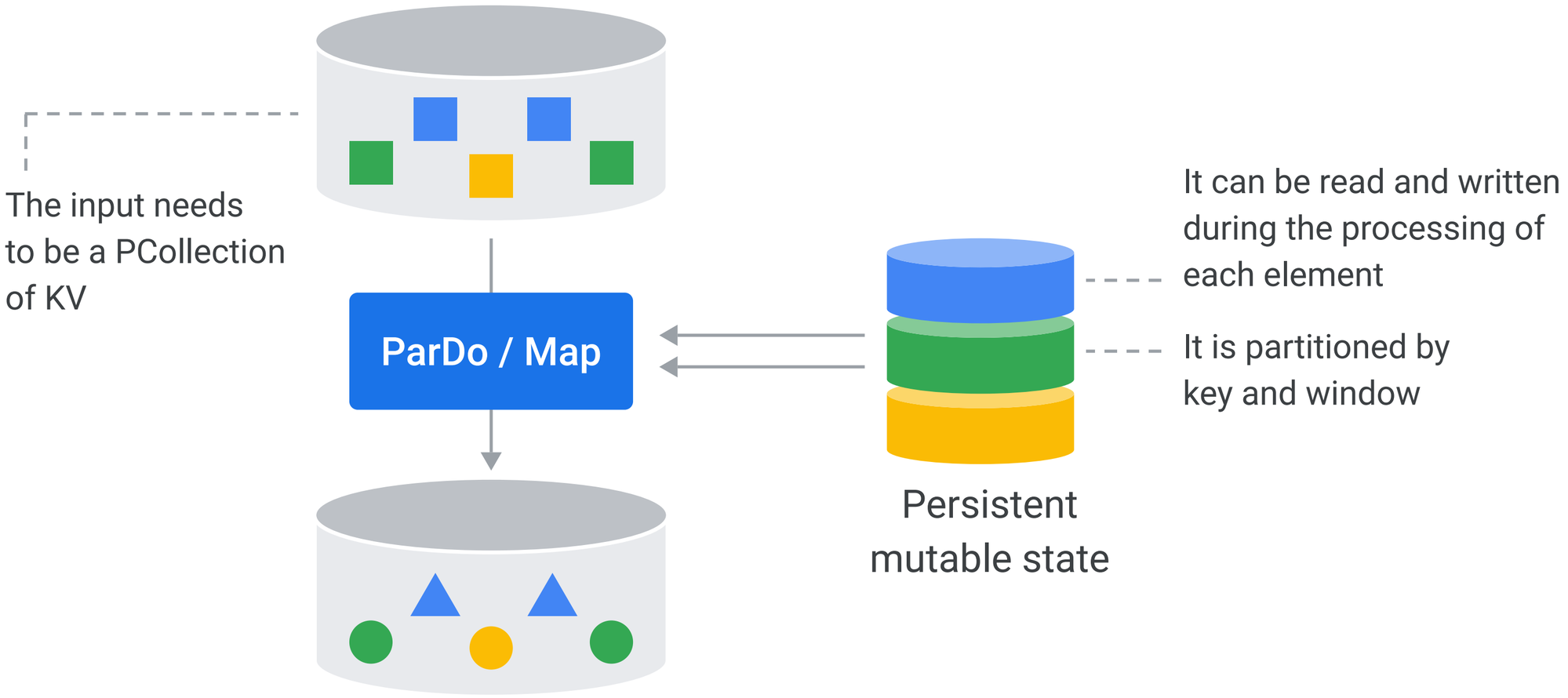
Agenda



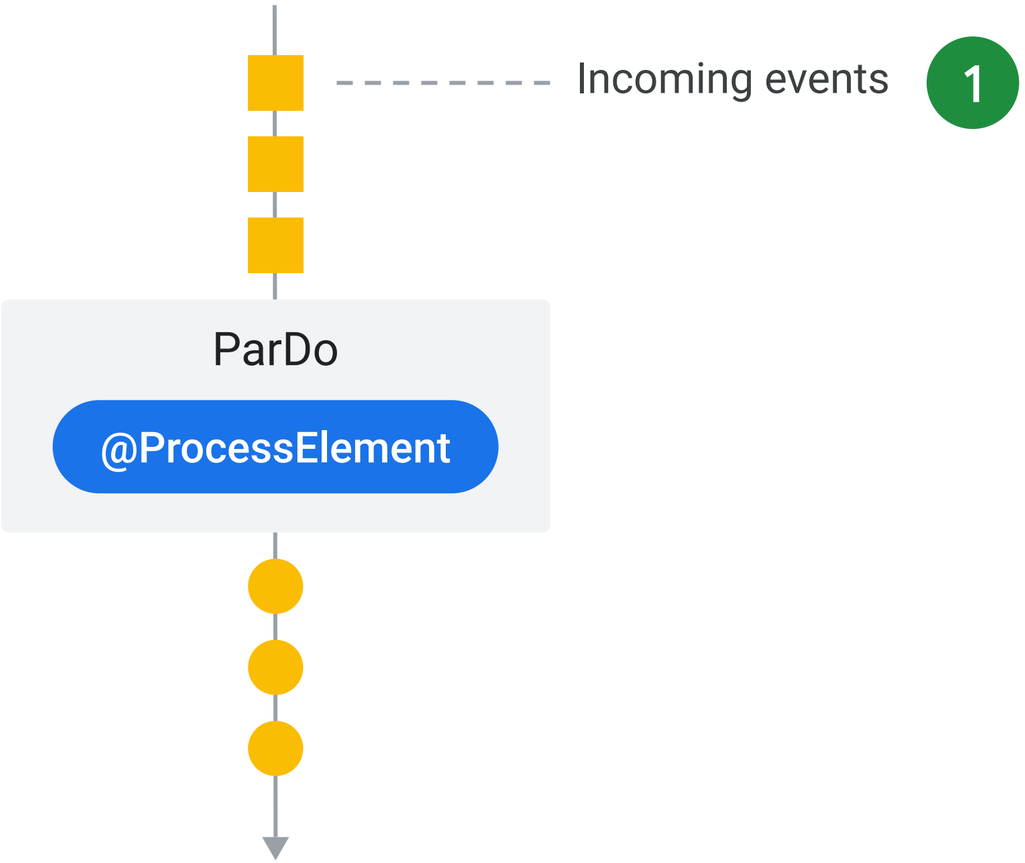
How to aggregate data in Apache Beam



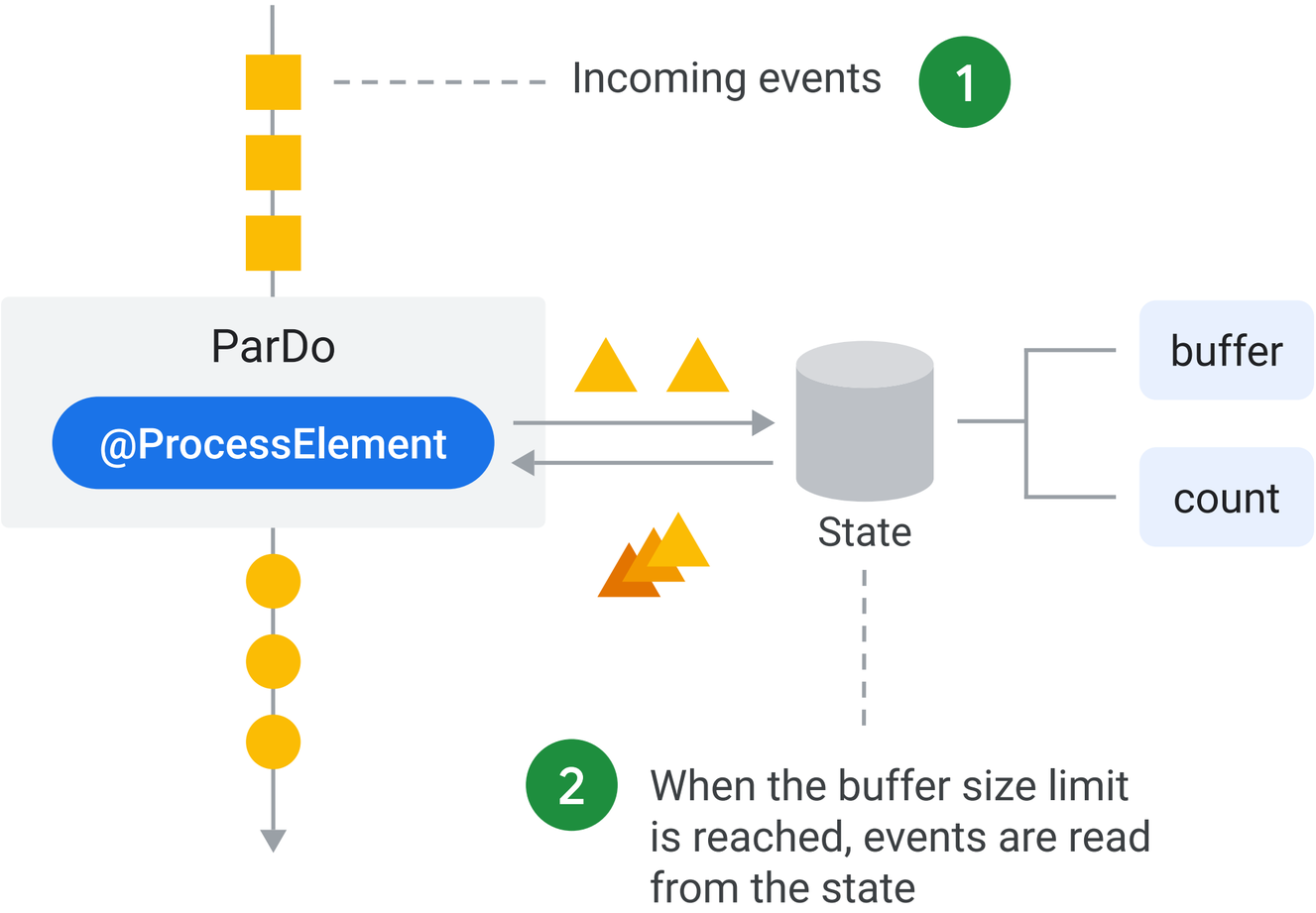
Introducing stateful ParDo



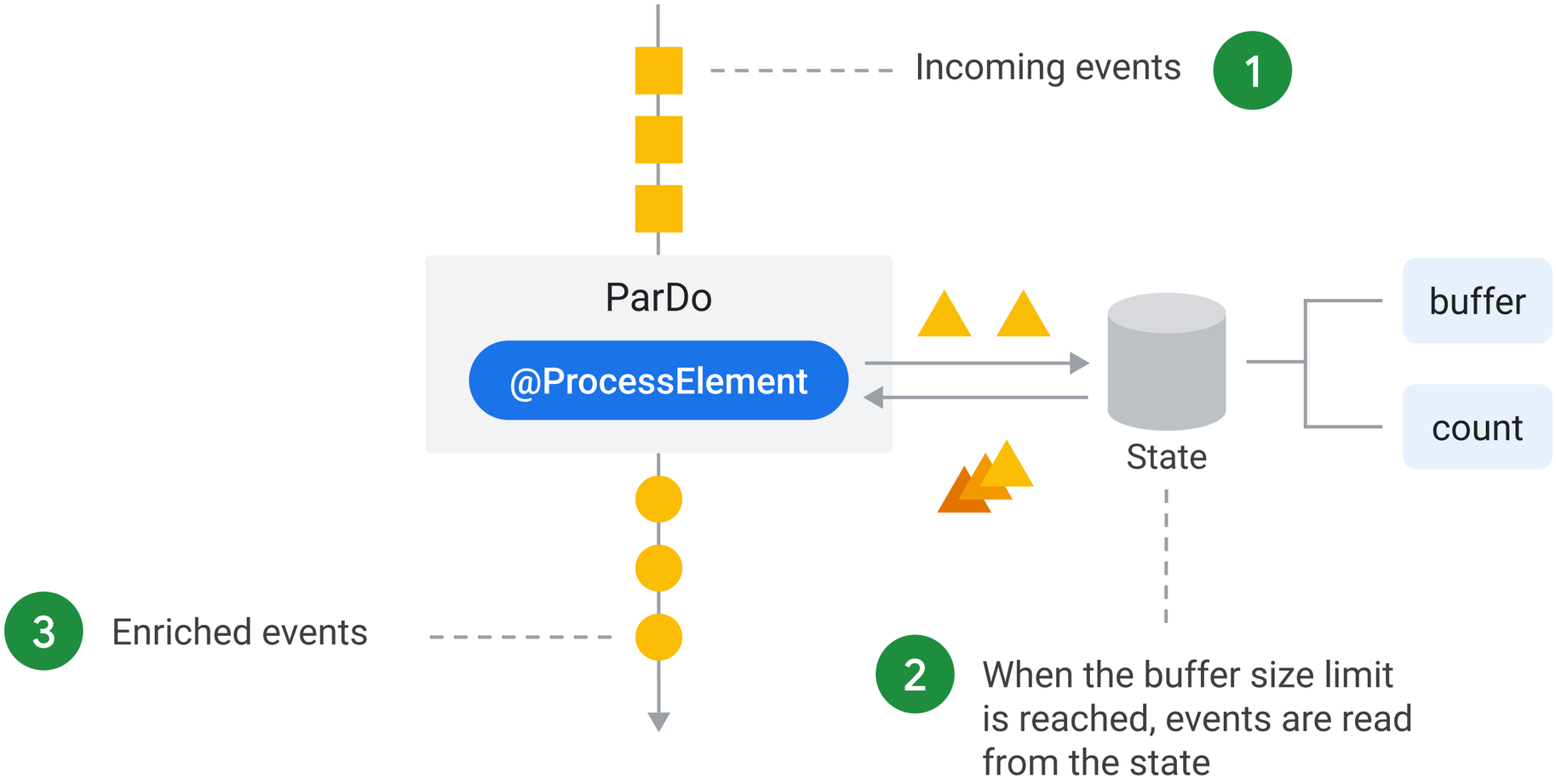
How does it work?



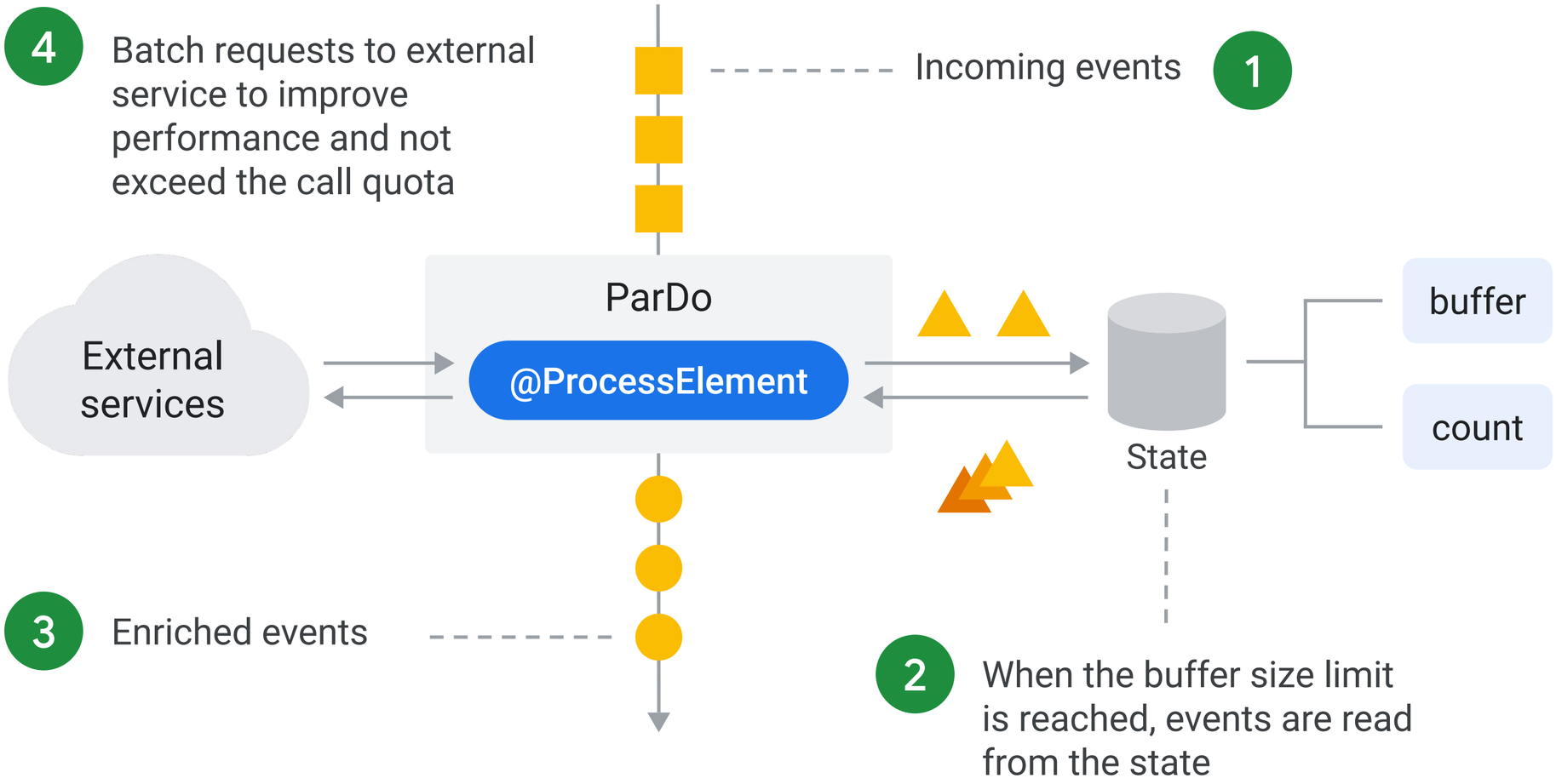
How does it work?



How does it work?



What is it useful for?



Python

```
class StatefulBufferingFn(beam.DoFn):
  MAX_BUFFER_SIZE = 500;
  BUFFER_STATE = BagStateSpec('buffer', EventCoder())
  COUNT_STATE = CombiningValueStateSpec('count',
                                        VarIntCoder(),
                                        combiners.SumCombineFn())
  def process(self, element,
              buffer_state=beam.DoFn.StateParam(BUFFER_STATE),
              count_state=beam.DoFn.StateParam(COUNT_STATE)):
    buffer_state.add(element)
    count_state.add(1)
    count = count_state.read()
    if count >= MAX_BUFFER_SIZE:
      for event in buffer_state.read():
        yield event
      count_state.clear()
      buffer_state.clear()
```

Increment count and add element to buffer

When buffer size limit is reached, a request is sent to the external service

Java

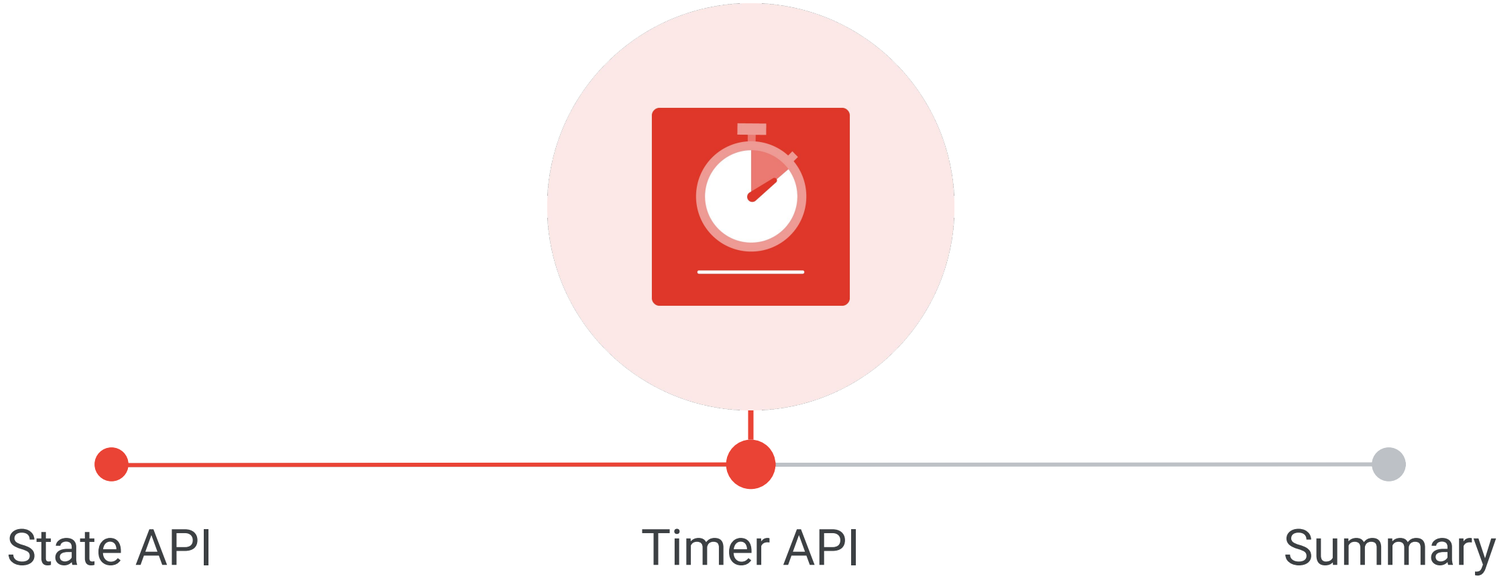
```java
new DoFn<Event, EnrichedEvent>() {
  private static final int MAX_BUFFER_SIZE = 500;
  @StateId("buffer") private final StateSpec<BagState<Event>> bufferedEvents = StateSpecs.bag();
  @StateId("count")  private final StateSpec<ValueState<Integer>> countState = StateSpecs.value();

  @ProcessElement
  public void process(ProcessContext context,
      @StateId("buffer") BagState<Event> bufferState,
      @StateId("count") ValueState<Integer> countState) {
    int count = firstNonNull(countState.read(), 0);
    count = count + 1; countState.write(count);
    bufferState.add(context.element());
    if (count >= MAX_BUFFER_SIZE) {
      for (EnrichedEvent enrichedEvent : enrichEvents(bufferState.read())) {
        context.output(enrichedEvent);
      }
      bufferState.clear();
      countState.clear();
    }
  }
  ...
}
```

Increment count and add element to buffer

When buffer size limit is reached a request is sent to the external service

State and Timers

Agenda



Timers

Event-time timers

Callback when the watermark reaches some threshold.

Processing-time timers

Callback after a certain amount of time has elapsed.

Python

```
class StatefulBufferingFn(beam.DoFn):
  …
  EXPIRY_TIMER = TimerSpec('expiry', TimeDomain.WATERMARK)
  def process(self, element,
              w=beam.DoFn.WindowParam,
              …
              expiry_timer=beam.DoFn.TimerParam(EXPIRY_TIMER)):
    expiry_timer.set(w.end + ALLOWED_LATENESS)
    … same logic as on previous code slide…

  @on_timer(EXPIRY_TIMER)
  def expiry(self,
             buffer_state=beam.DoFn.StateParam(BUFFER_STATE),
             count_state=beam.DoFn.StateParam(COUNT_STATE)):
    events = buffer_state.read()
    for event in events:
      yield event
    buffer_state.clear()
    count_state.clear()
```

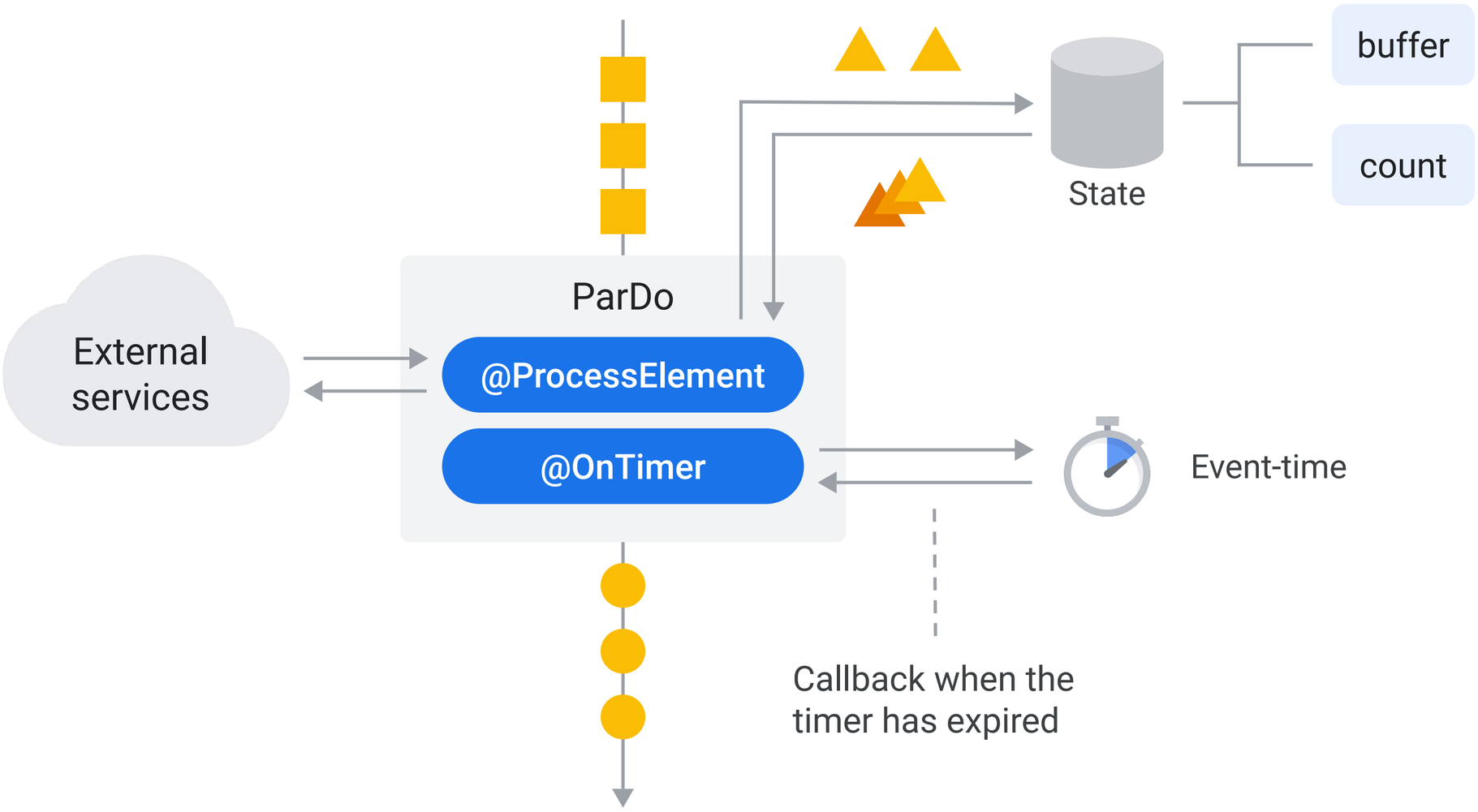
Added an event time timer so that when the window expires, any events remaining in the buffer are processed.

Java

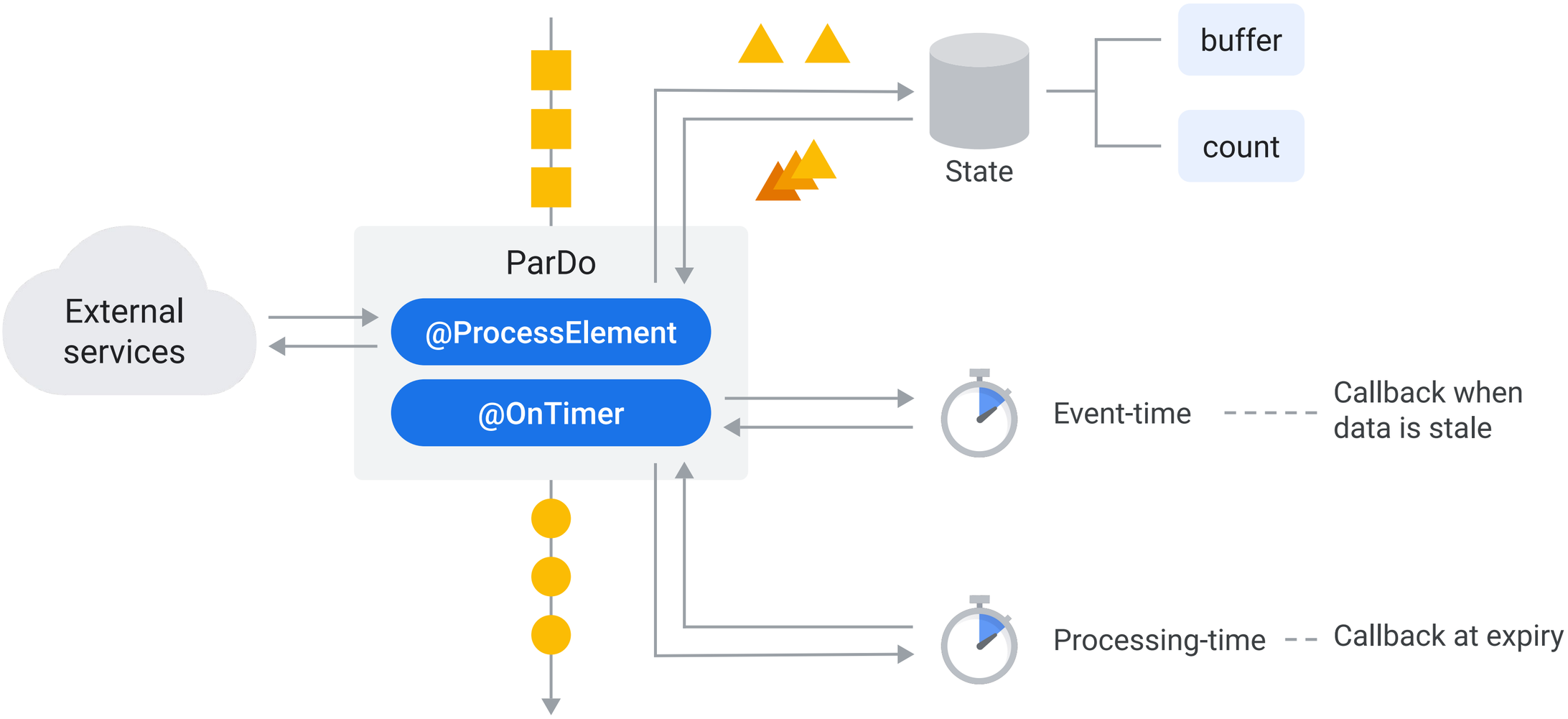
```
new DoFn<Event, EnrichedEvent>() {
  …
  @TimerId("expiry") private final TimerSpec expirySpec = TimerSpecs.timer(TimeDomain.EVENT_TIME);
  …
  @ProcessElement
  public void process(ProcessContext context, BoundedWindow window, …
   @TimerId("expiry") Timer expiryTimer) {
    expiryTimer.set(window.maxTimestamp().plus(allowedLateness));
    … same logic as on previous code slide…
 }

 @OnTimer("expiry")
 public void onExpiry(OnTimerContext context,
  @StateId("buffer") BagState<Event> bufferState) {
    if (!bufferState.isEmpty().read()) {
      for (EnrichedEvent enrichedEvent : enrichEvents(bufferState.read())) {
        context.output(enrichedEvent);
      }
      bufferState.clear();
    }
  }
}
```

Added an event-time timer so that when the window expires, any events remaining in the buffer are processed.

Stateful DoFn with state and timers

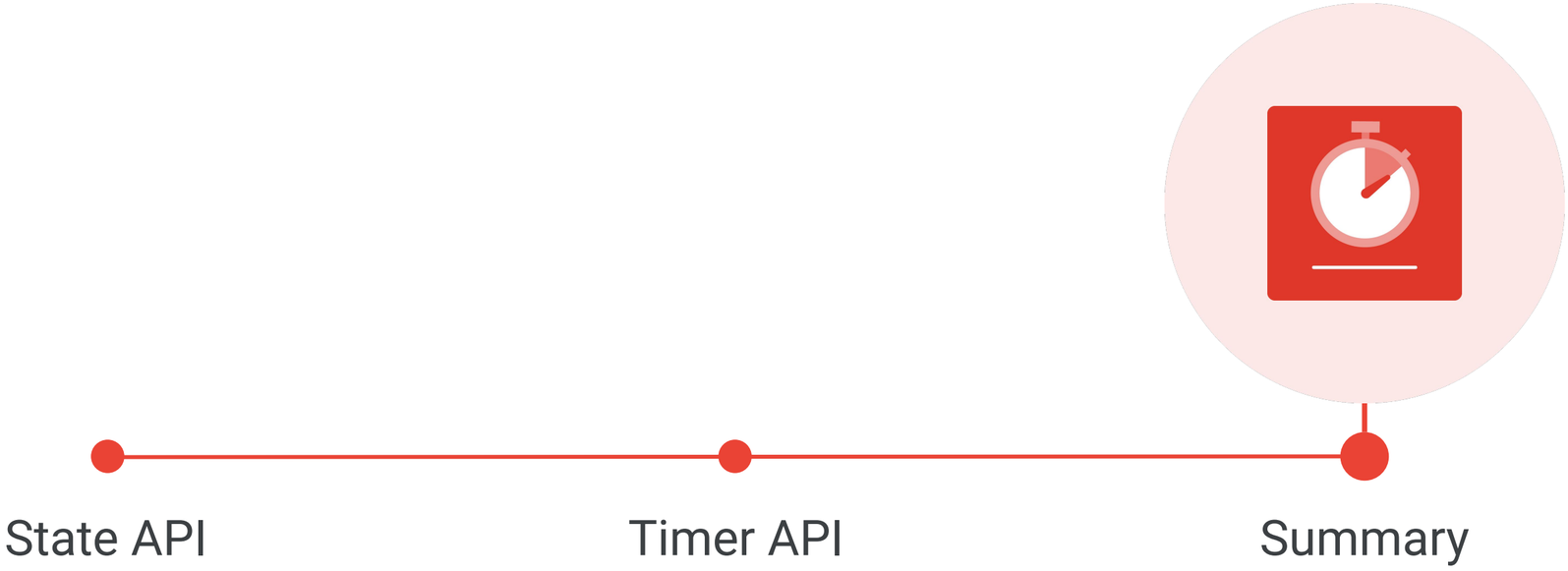


Stateful DoFn with state and timers



State and Timers

Agenda



Summary: Types of timers

Processing time

Event time

Summary: Types of timers

Processing time

- Timeouts
- Relative times ("in 5 minutes")
- Periodic output based on state

Event time

Summary: Types of timers

Processing time

- Timeouts
- Relative times ("in 5 minutes")
- Periodic output based on state

Event time

- Output based on completeness of input data
- Absolute times ("when the data is complete up to 5:00am")
- Final/authoritative outputs
- Don't leave data behind in state!

Summary: Types of state variables

Summary: Types of state variables

Type	Strength	Dataflow runner
Value	Read/write any value (but always the whole value)	✓

Summary: Types of state variables

Type	Strength	Dataflow runner
Value	Read/write any value (but always the whole value)	✓
Bag	Cheap append. No ordering on read	✓

Summary: Types of state variables

Type	Strength	Dataflow runner
Value	Read/write any value (but always the whole value)	✓
Bag	Cheap append No ordering on read	✓
Combining	**Associative/commutative compaction**	✓

Summary: Types of state variables

Type	Strength	Dataflow runner
Value	Read/write any value (but always the whole value)	✓
Bag	Cheap append No ordering on read	✓
Combining	Associative/commutative compaction	✓
Map	**Read/write just keys you specify**	✓

Summary: Types of state variables

Type	Strength	Dataflow runner
Value	Read/write any value (but always the whole value)	✓
Bag	Cheap append No ordering on read	✓
Combining	Associative/commutative compaction	✓
Map	Read/write just keys you specify	✓
Set	Membership checking	✗

What can you do with state and timers?

- **Domain-specific triggering** ("output when five people who live in Seattle have checked in")
- **Slowly changing dimensions** ("update FX rates for currency ABC")
- **Stream joins** ("join-matrix" / "join-biclique")
- **Fine-grained aggregation** ("add odd elements to accumulator A and event elements to accumulator B")
- **Per-key workflows** (like user sign up flow w/ reminders & expiration)