Detecting Change-Points in Apache Beam

By Devon Peticolas - Oden Technologies

https://github.com/x/slides/tree/master/beam-summit-2022

BEAM SUMMIT Austin, 2022

Devon Peticolas

Principal Engineer @
Oden Technologies
“Beam Guy”

BEAM
SUMMIT

In This Talk

- What is Change-Point Detection?
- Why and how does Oden use Change-Point Detection to deliver features?
- Methods of doing Change-Point Detection in Beam.
- Methods of doing Change-Point Detection with Smoothing.
- Impacts of event sparsity, lateness, and order.

A little about Oden

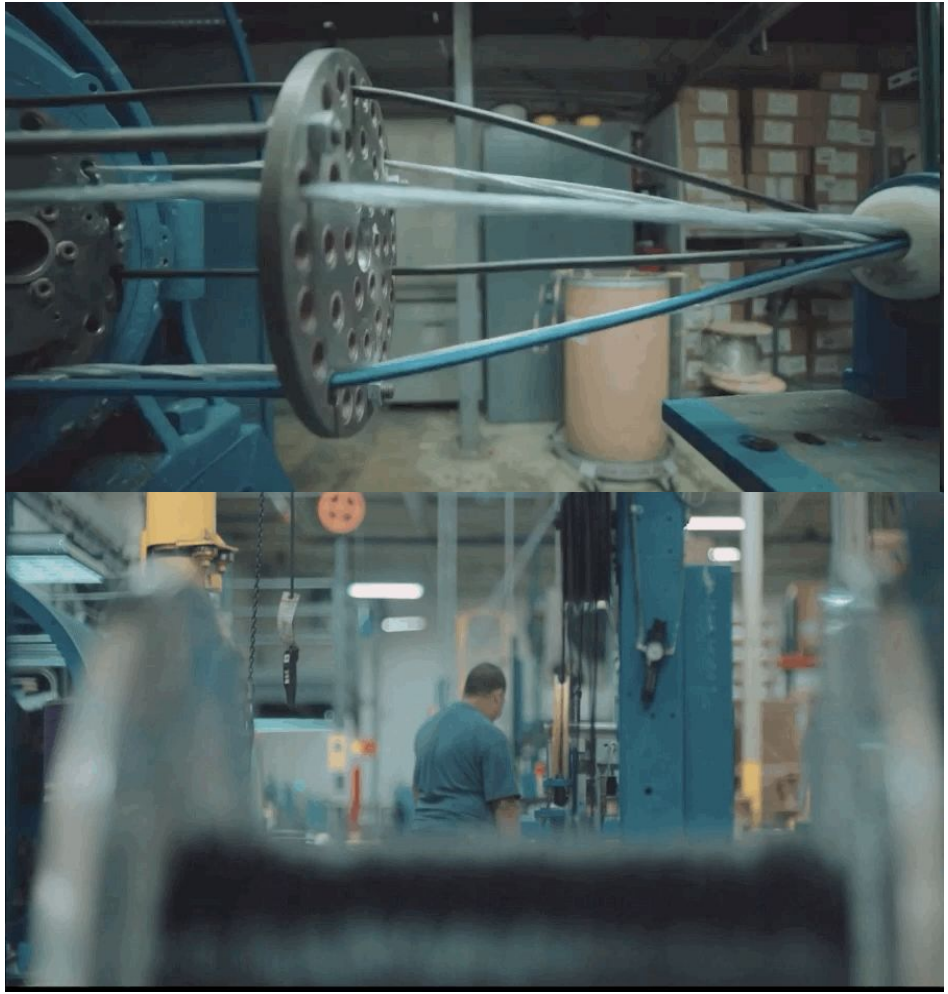
BEAM SUMMIT Austin, 2022

Oden's Customers

Medium to large manufacturers in plastics extrusion, injection molding, and pipes, chemical, paper and pulp.

Process and Quality Engineers looking to centralize, analyze, and act on their data.

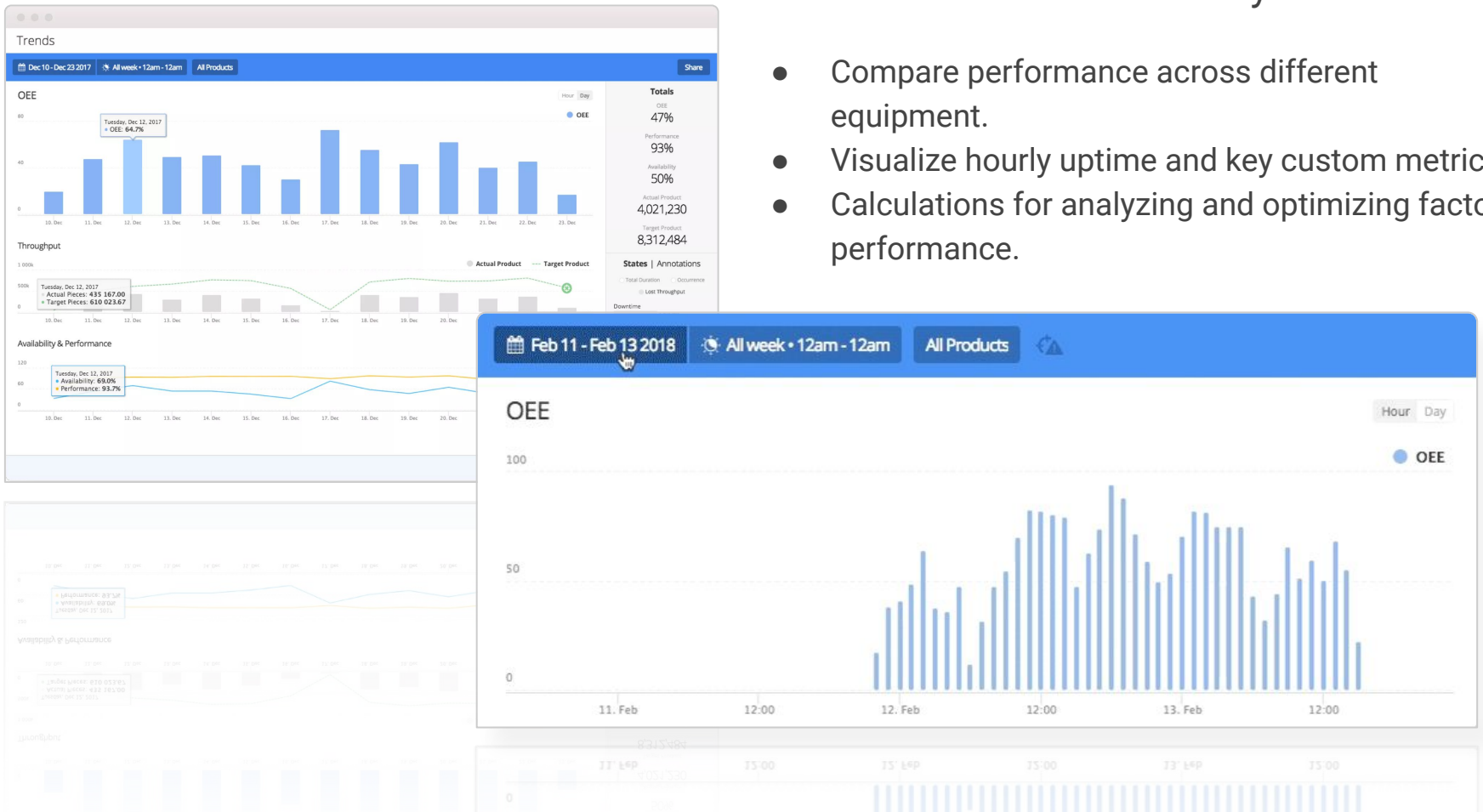
Plant managers who are looking to optimize logistics, output, and cost.

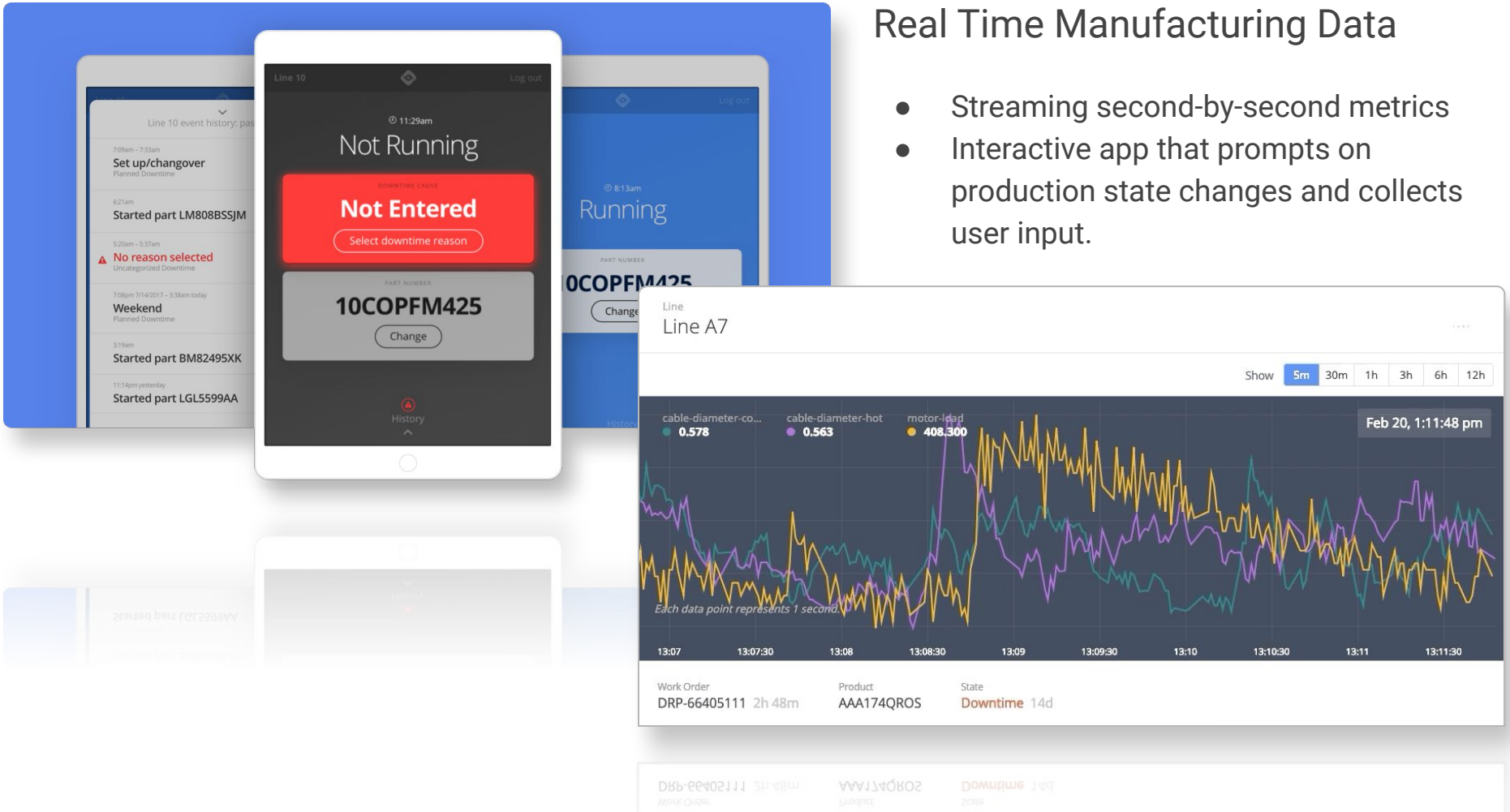


BEAM SUMMIT

Interactive Time-series Analysis

- Compare performance across different equipment.
- Visualize hourly uptime and key custom metrics.
- Calculations for analyzing and optimizing factory performance.





Real Time Manufacturing Data

- Streaming second-by-second metrics
- Interactive app that prompts on production state changes and collects user input.

Background:
How Oden Uses Beam

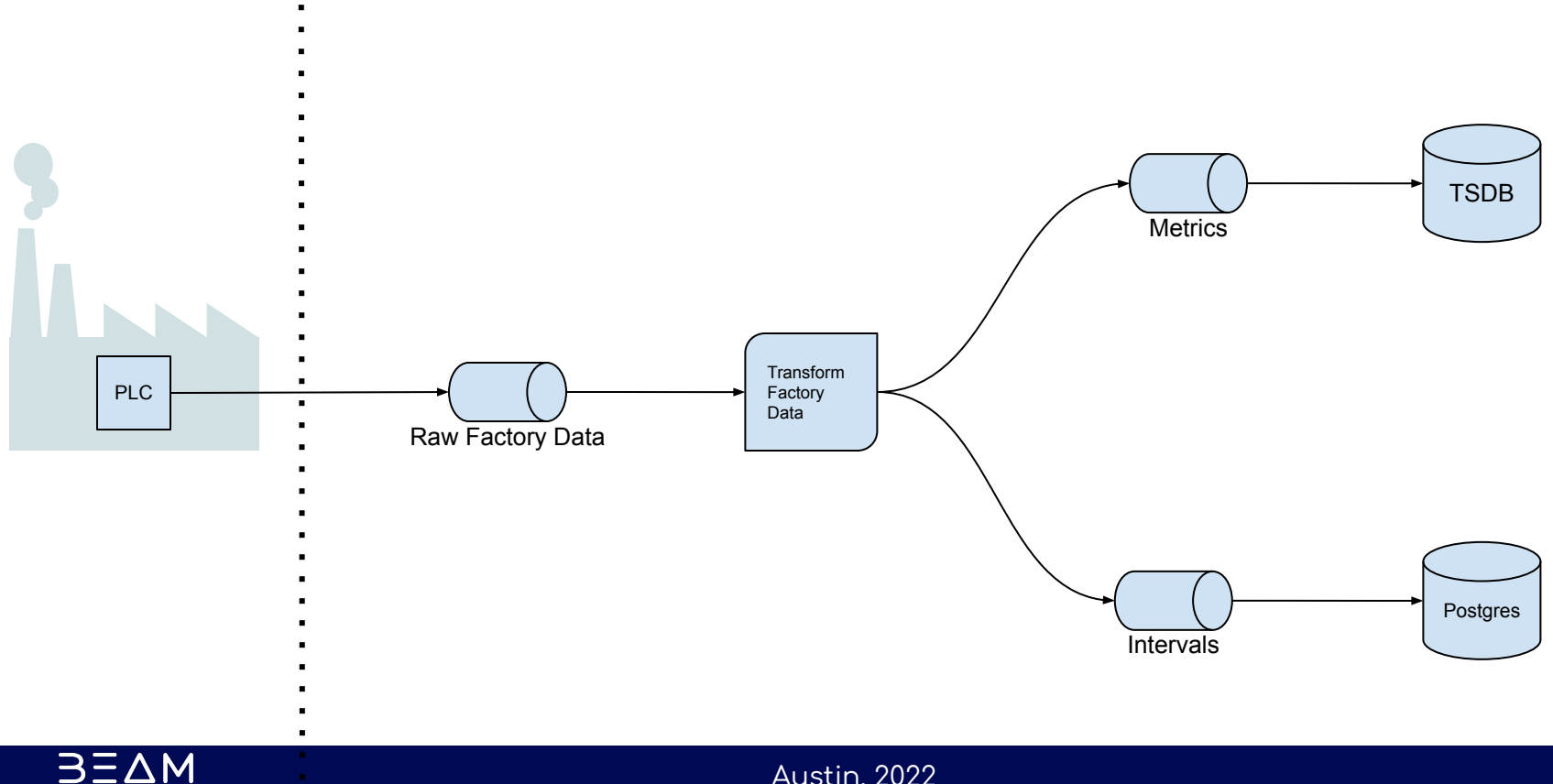
BEAM SUMMIT Austin, 2022

How Oden Uses Beam

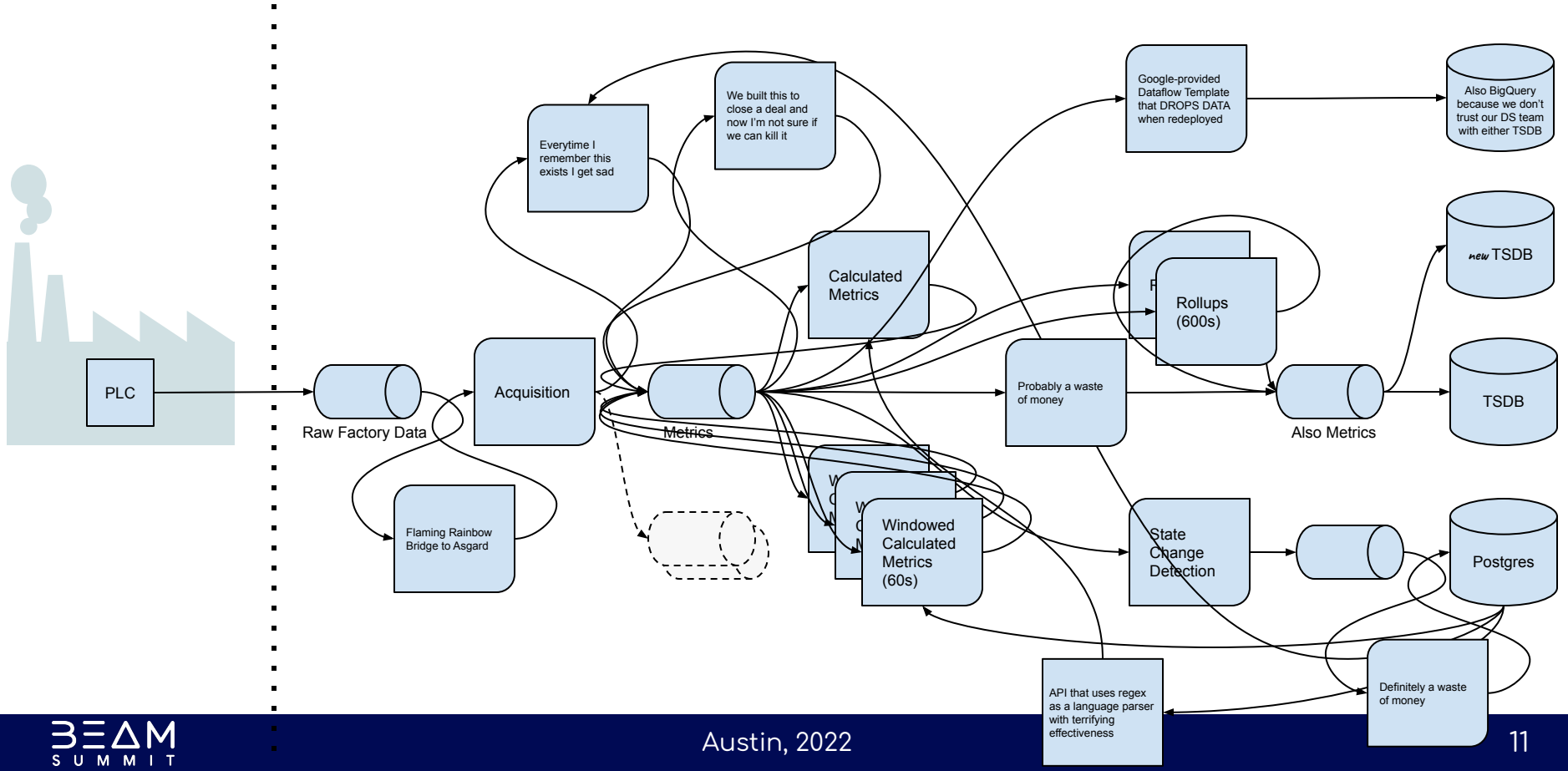
- Ingesting “raw” manufacturing data and mapping it into Oden “events”
- Combining events using streaming joins
- Making customer-configured transformations to events
- Transforming metric events into contextual interval events

*Lots of Side-Input Joining
*Lots of Complex Windowing
*Lots of Performance Concerns

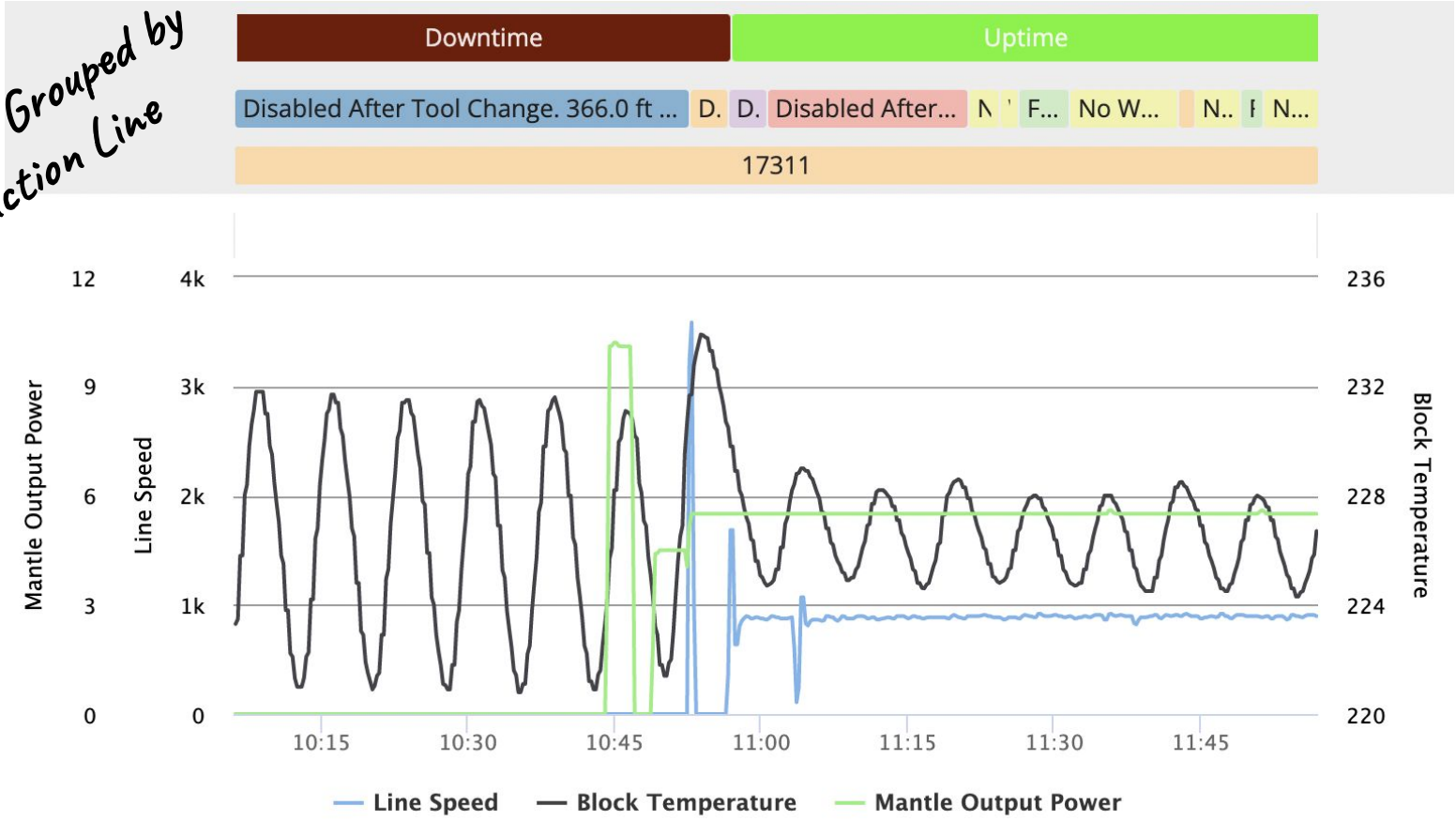
Streaming Factory Data - In Summary

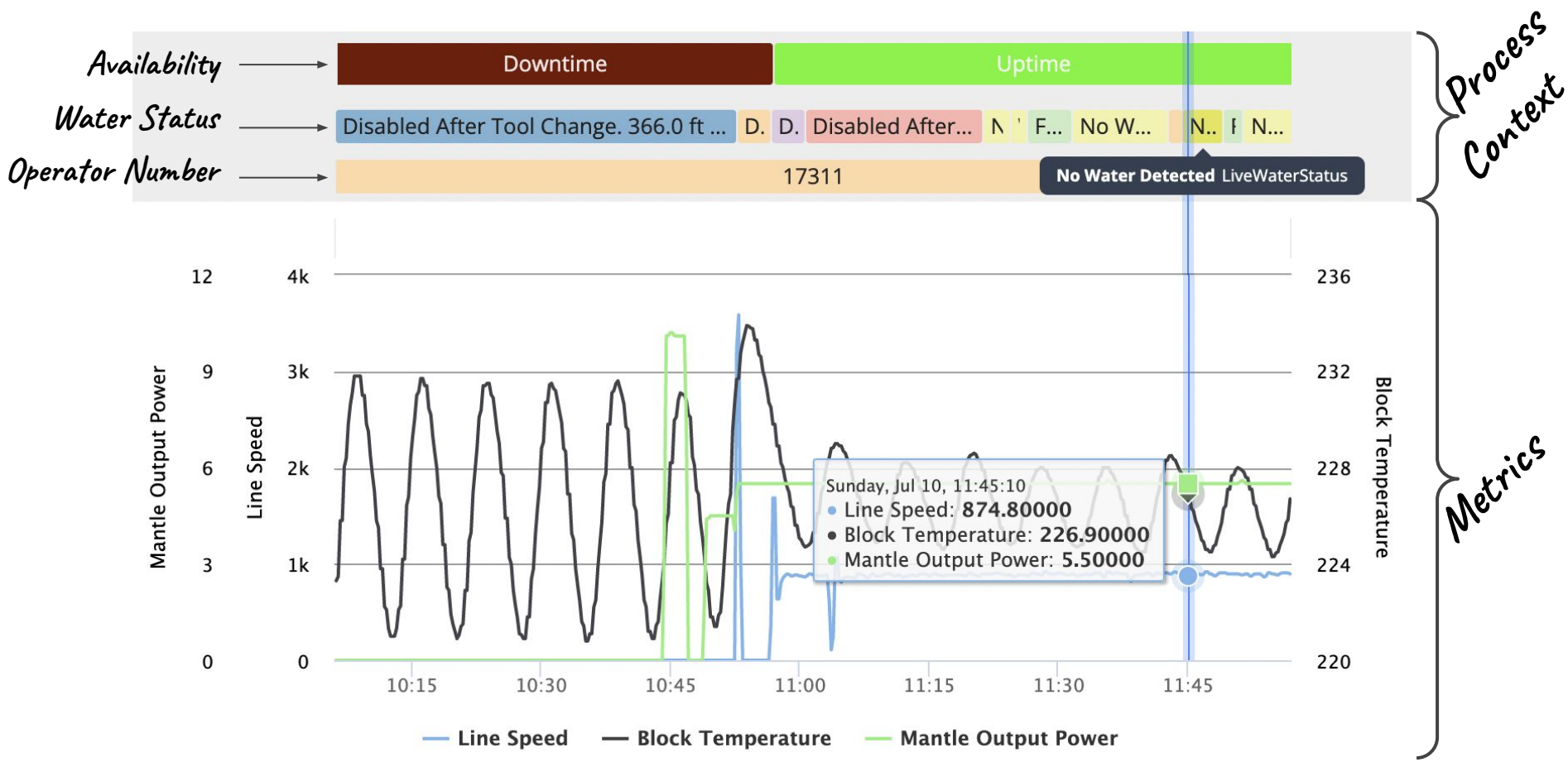


Streaming Factory Data - In ~~Summary~~ Reality

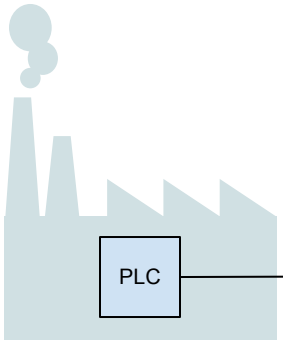


Data is Grouped by Production Line




Availability
Water Status
Operator Number
Downtime
Uptime
Disabled After Tool Change. 366.0 ft ...
D.
D.
Disabled After...
F...
No W...
N..
N...
17311
No Water Detected LiveWaterStatus
Process Context
Metrics
Mantle Output Power
Line Speed
Block Temperature
12
9
6
3
0
4k
3k
2k
1k
0
236
232
228
224
220
10:15
10:30
10:45
11:00
11:15
11:30
11:45
Sunday, Jul 10, 11:45:10
Line Speed: 874.80000
Block Temperature: 226.90000
Mantle Output Power: 5.50000
Line Speed
Block Temperature
Mantle Output Power

Metrics



PLC

Raw Factory Data

Transform Factory Data

Metrics

TSDB

Each metric describes a sensor

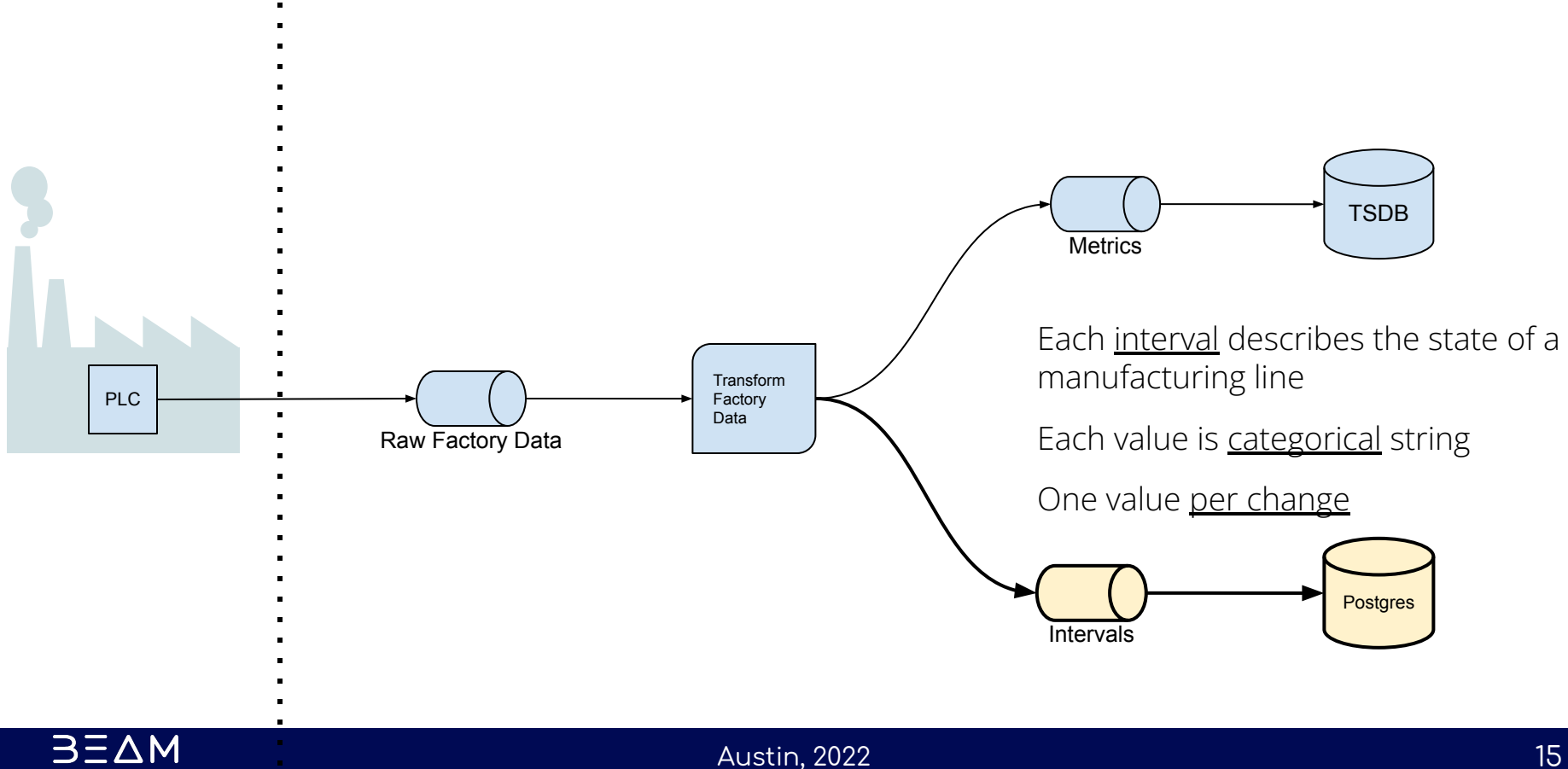
Each value is a float

One value per second per metric

Intervals

Postgres

Intervals

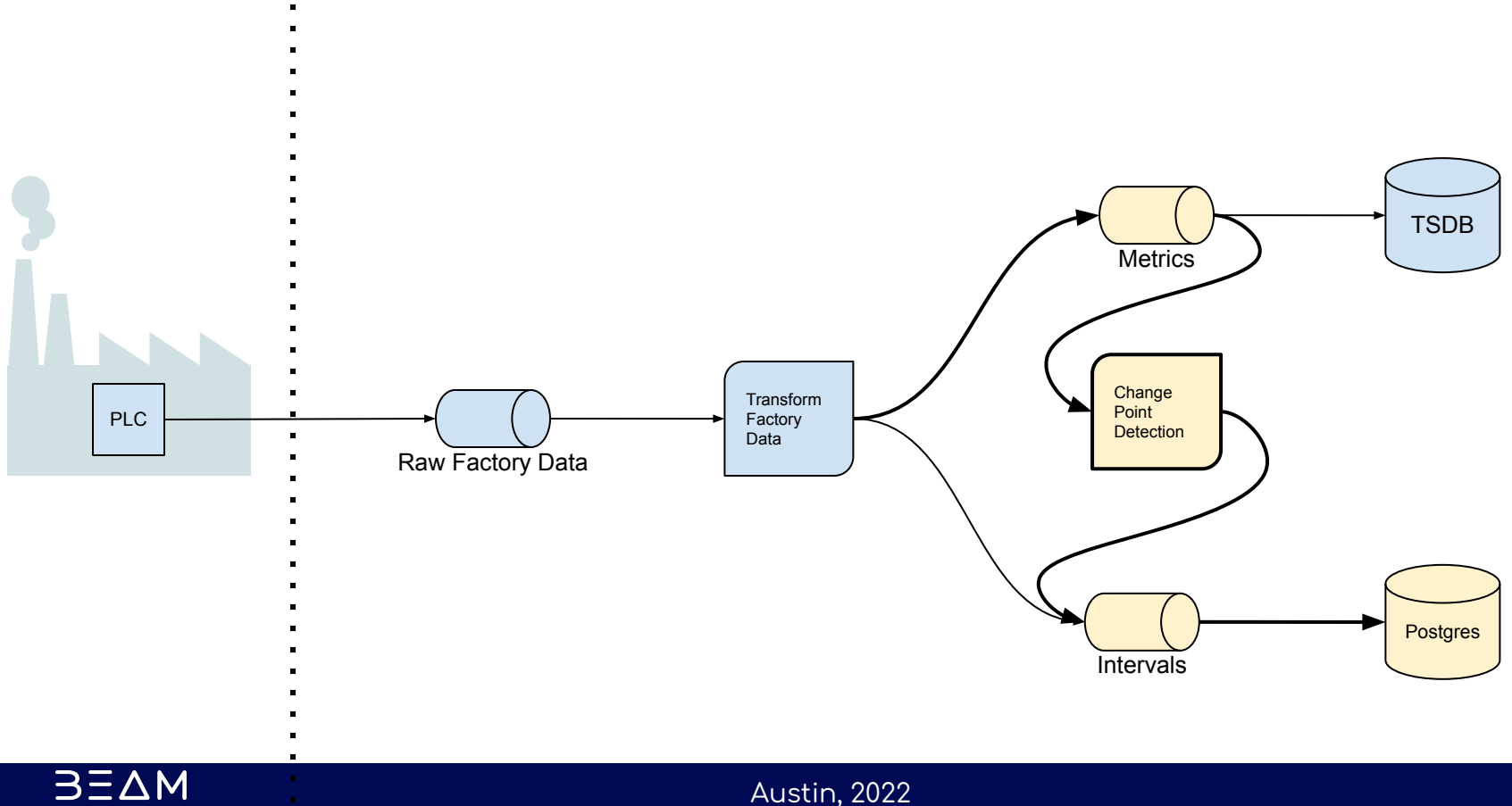


Each interval describes the state of a manufacturing line

Each value is categorical string

One value per change

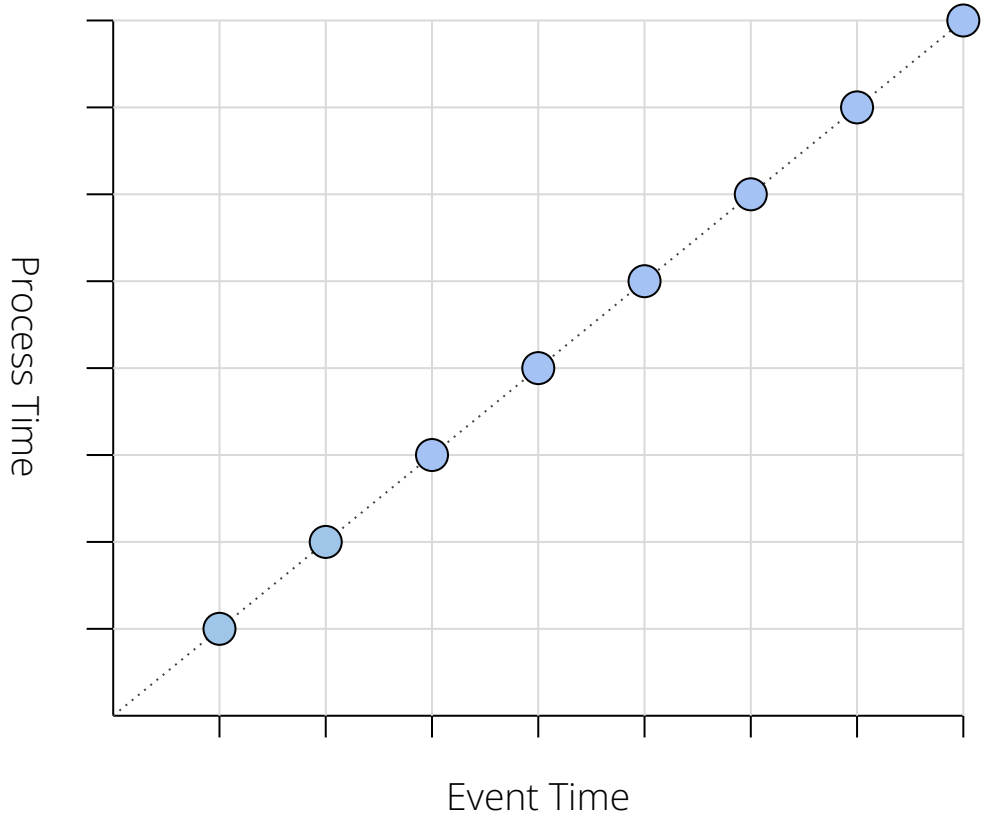
Creating Intervals from Metrics



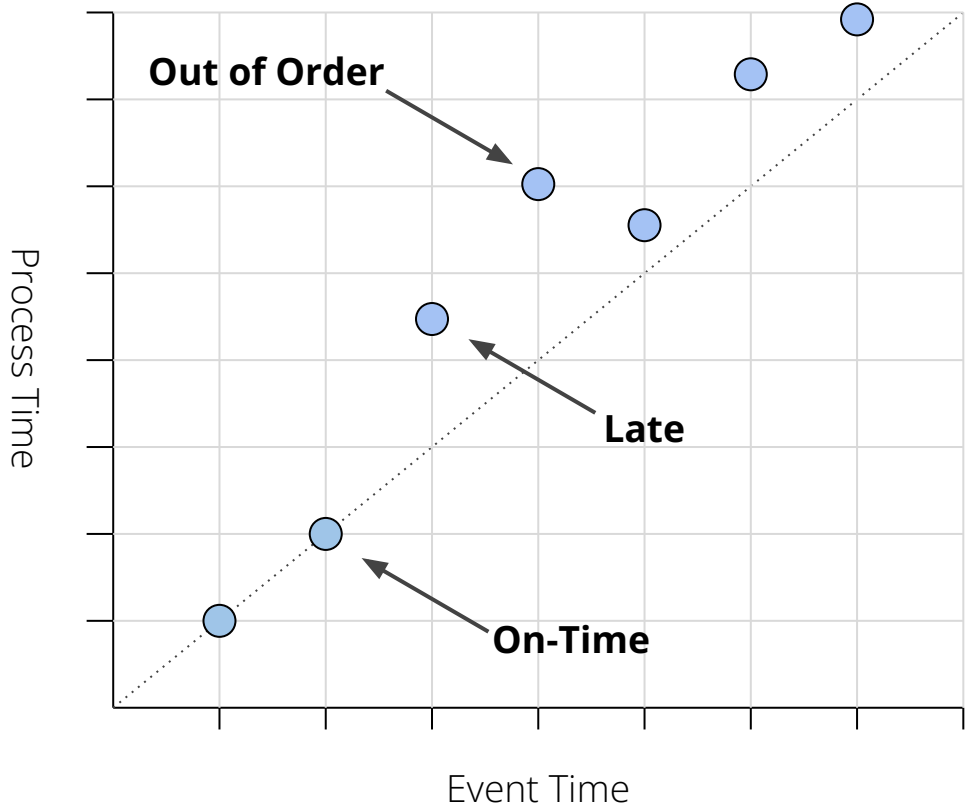
Use Case:
Creating Intervals from Metrics

BEAM SUMMIT Austin, 2022

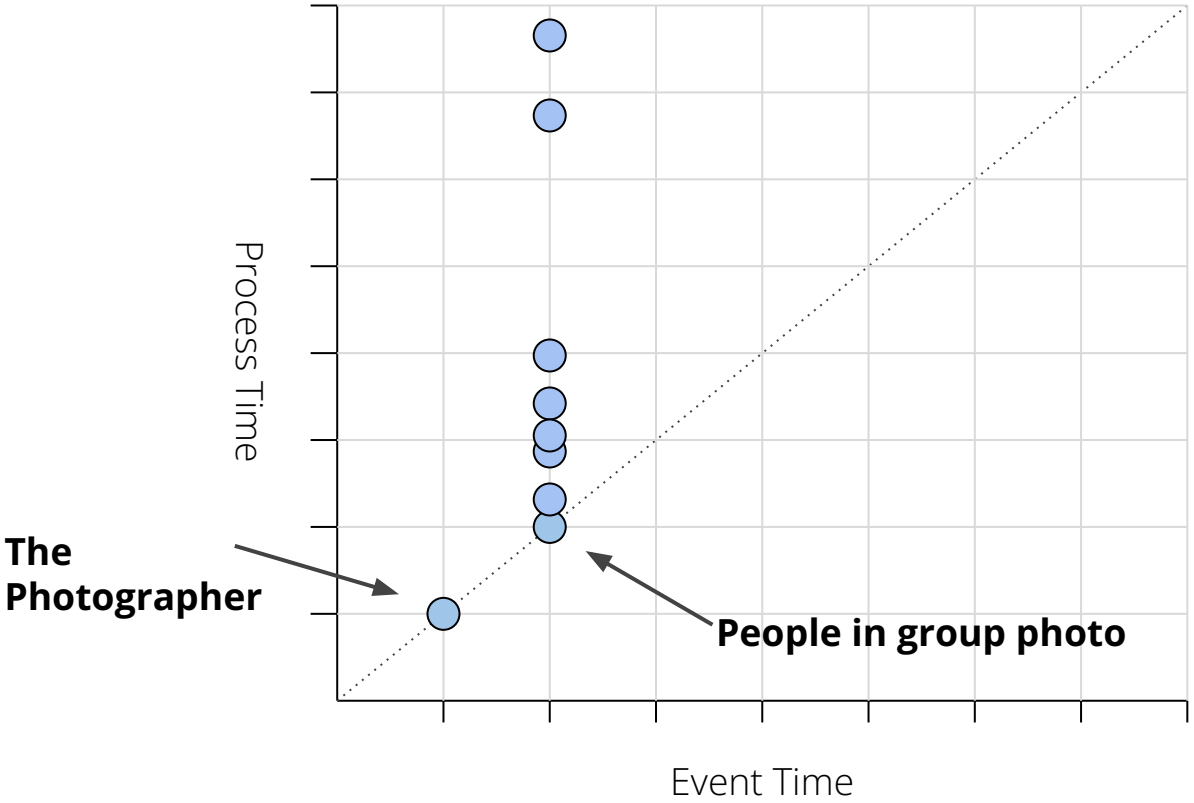
Visualizing Events



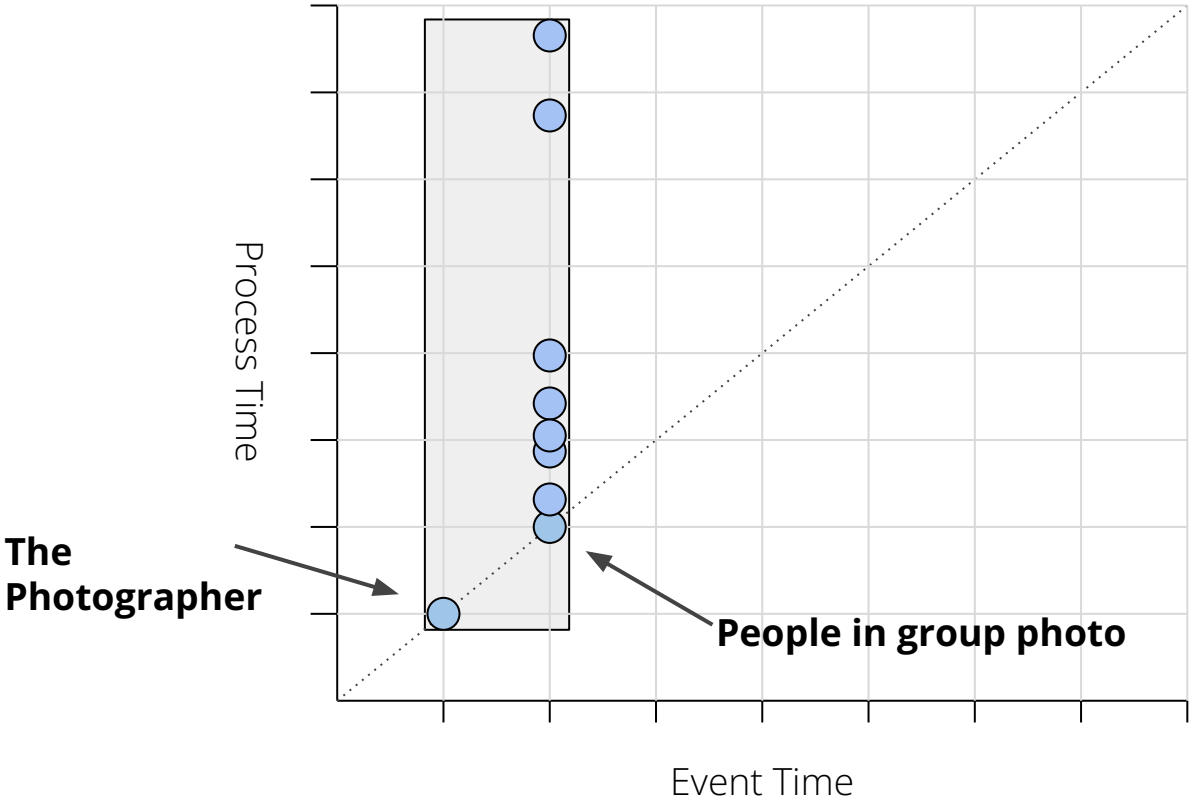
Visualizing Events



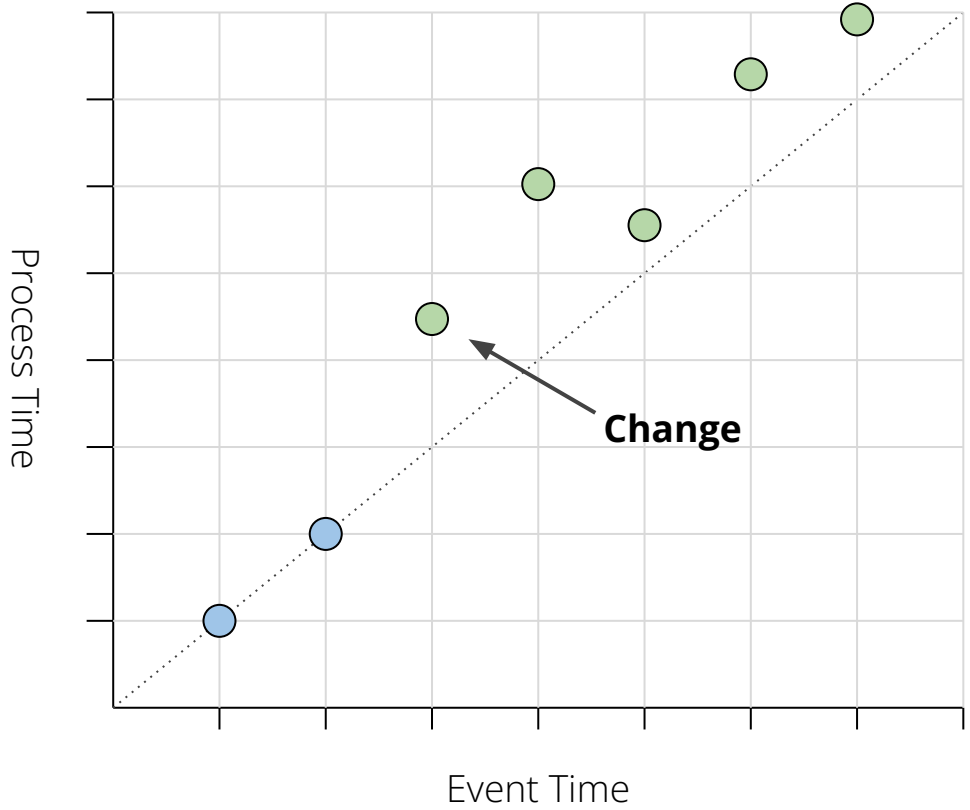
Visualizing Events

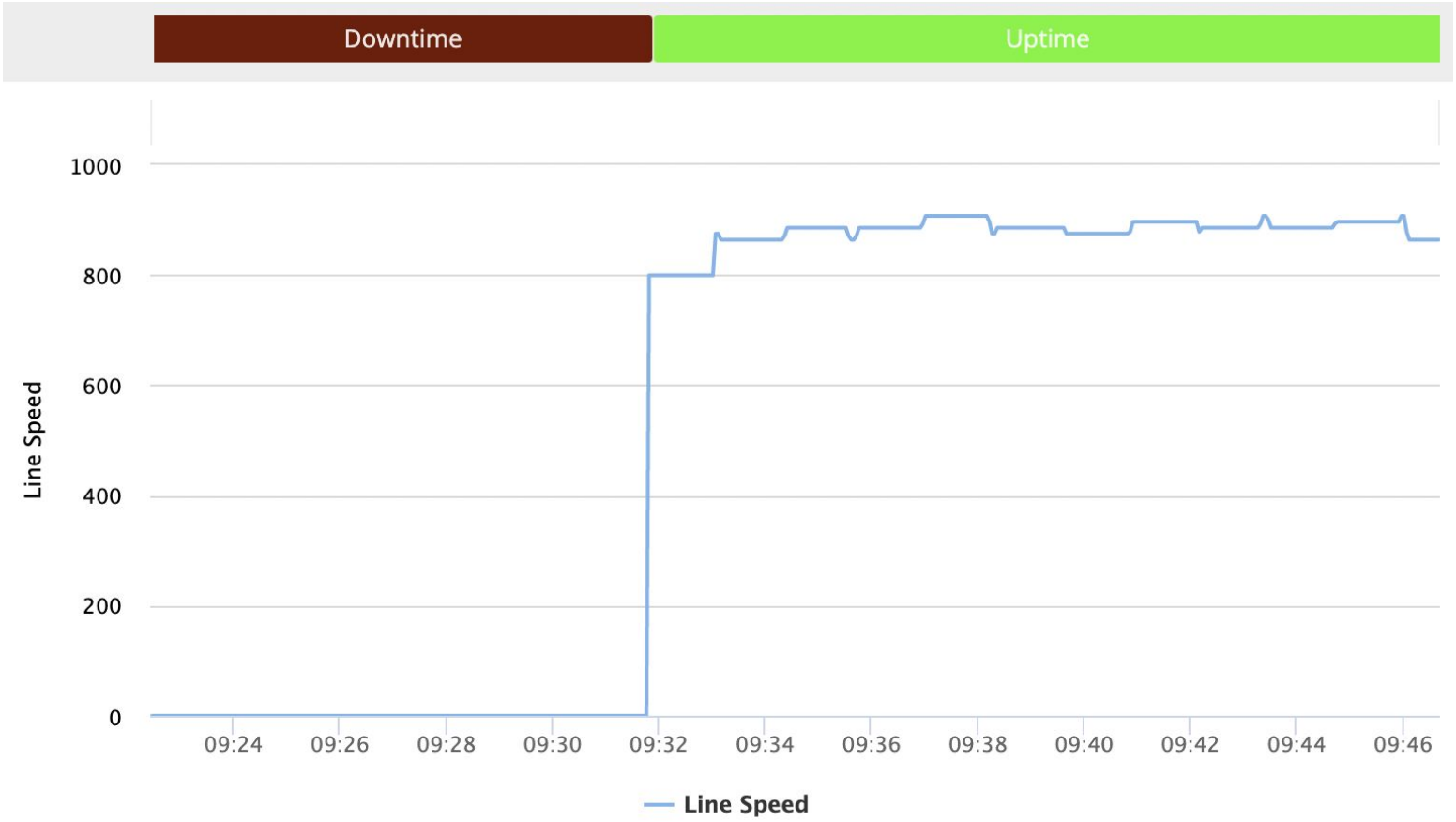


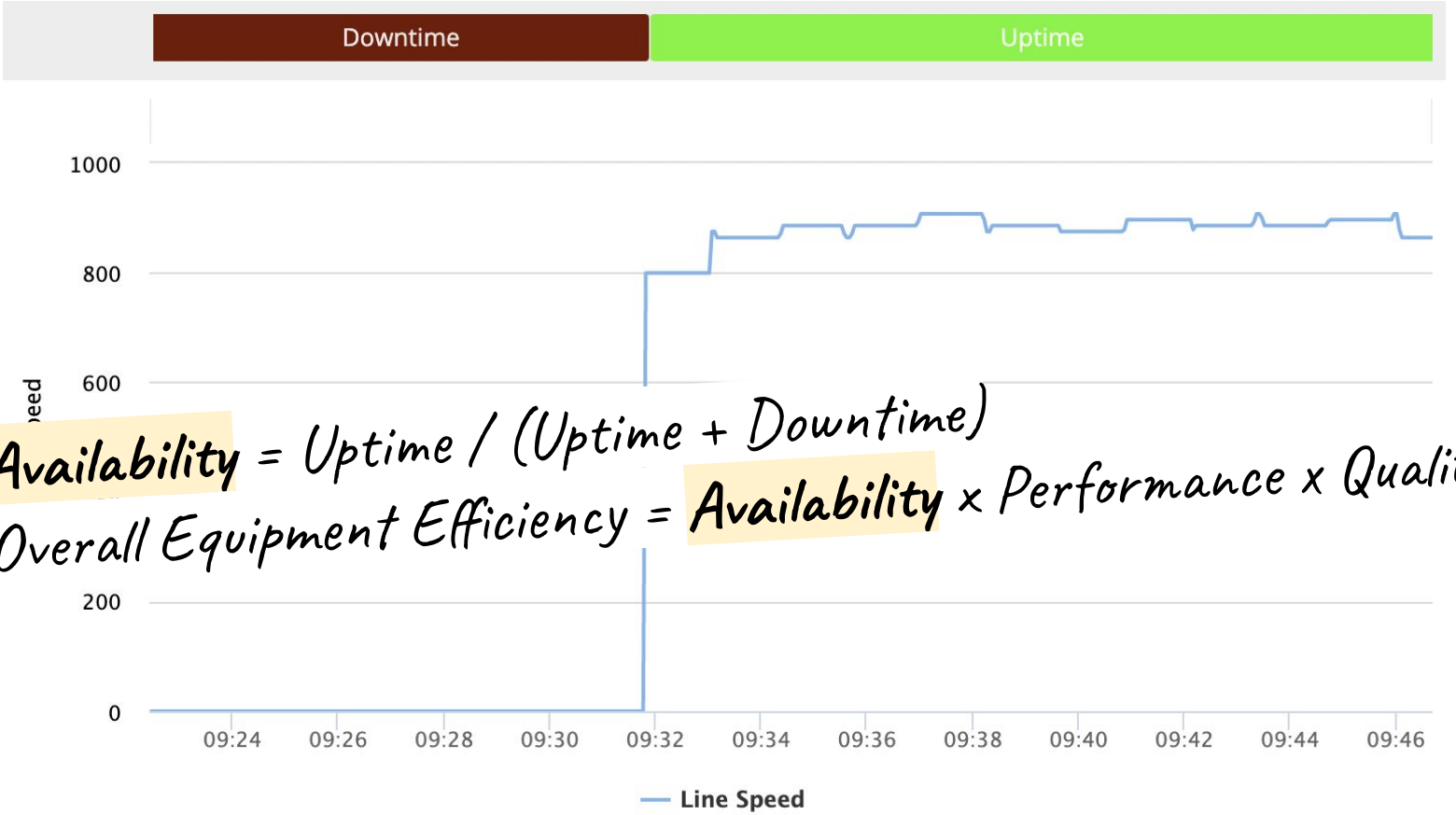
Visualizing Events



Visualizing Events w/ Change




Downtime
Uptime
1000
800
600
400
200
0
Line Speed
09:24
09:26
09:28
09:30
09:32
09:34
09:36
09:38
09:40
09:42
09:44
09:46
Line Speed



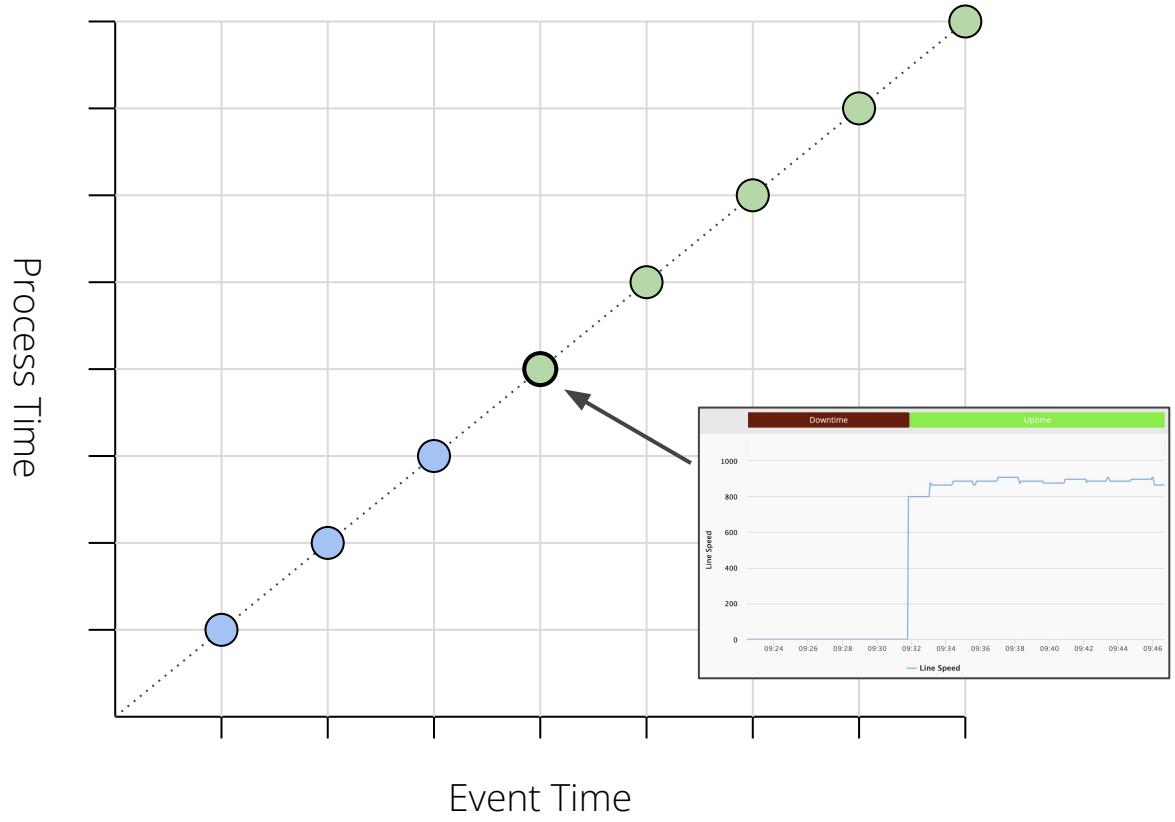
Availability = Uptime / (Uptime + Downtime)

Overall Equipment Efficiency = **Availability** x Performance x Quality

Metrics into Categorical Values

```
// Load configuration every 5-minutes.
PCollectionView<Config> configView = p
  .apply(
    GenerateSequence
      .from(0)
      .withRate(
        1, Duration.standardMinutes(5)))
  .apply(MapElements(...)) // API call
  .apply(View.asSingleton());

// Map metric values to categorical
// values using config side-input.
p.apply(ParDo
  .of(new DoFn<Metric, String>() {
    public void processElement(
      Metric m, ProcessContext c
    ) {
    Config config = c.sideInput(configView)
    if (m.value > config.forMetric(m)) {
      c.output("up");
    } else {
      c.output("down");
    }
  })
  .withSideInputs(configView))
```



Factory A

Last 2 hours | 5 lines | Print | List

Line	LINE AGGREGATE State History	Util	Perf	Production	CURRENT RUN ON LINE Work Order	Current Speed	Current Production
Compounding 1	Uptime 4h 50m	100%	0%	15,000 lb	UHK-71120000 25h 13m Product 6D50D9B2 Batch 3C64E95D	7,500	174,222 lb
Extruder 1	Downtime 2m 55s	81%	181%	36,407 ft	Test Run 347d 1h Product 109810-P-A	0 207 +3.56 / -17	12,385,545 ft
Extruder 2	Uptime 49m 2s	95%	79%	5,066 ft	XYZ-123 656d 9h Product 85EB50E1 Batch 22CD2E30	50 56 +4 / -6	1,753,394 ft
Extruder 3	Uptime 3m 44s	63%	78%	12,468 ft	UHL-64957000 2h 56m Product D48199B2	183 210 ± 15	17,830 ft
Line 01	Uptime 53m 33s	97%	0%	0 ft	line_demo_in_PROD 53d 12h Product 2AFBCA5E Batch D824D563	330 320 +10 / -20	0 ft

OEE	Utilization	Performance	Quality
0%	87.3%	80.6%	0%

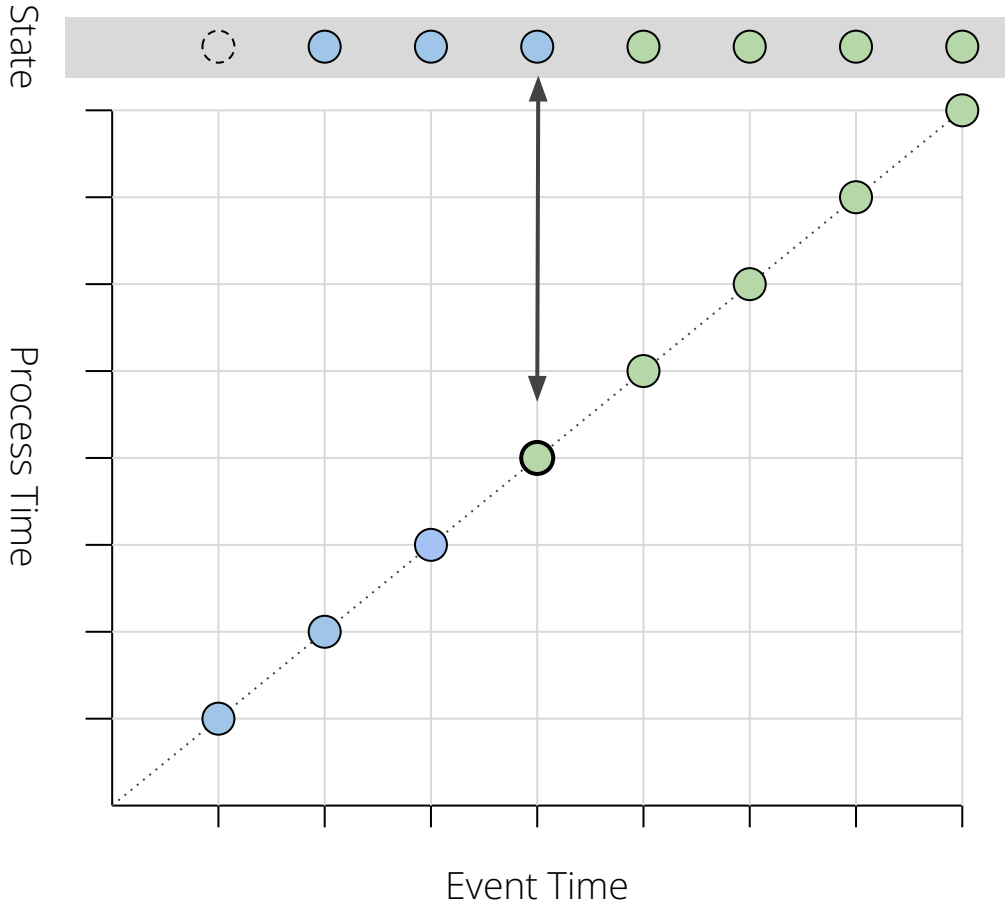
Solution:
Using Beam State

BEAM SUMMIT Austin, 2022

Beam State to Detect Change-Points

```
DoFn<KV<String, T>, T> {
  StateSpec<ValueSpec<T>>
prevSpec =
    StateSpecs.value(...);

  public void processElement(
    ProcessContext c,
    ValueState<T> prev) {
    T curr =
c.element().getValue();
    T last = prev.read();
    if ( curr != last) {
      c.output(curr);
    }
  }
}
```



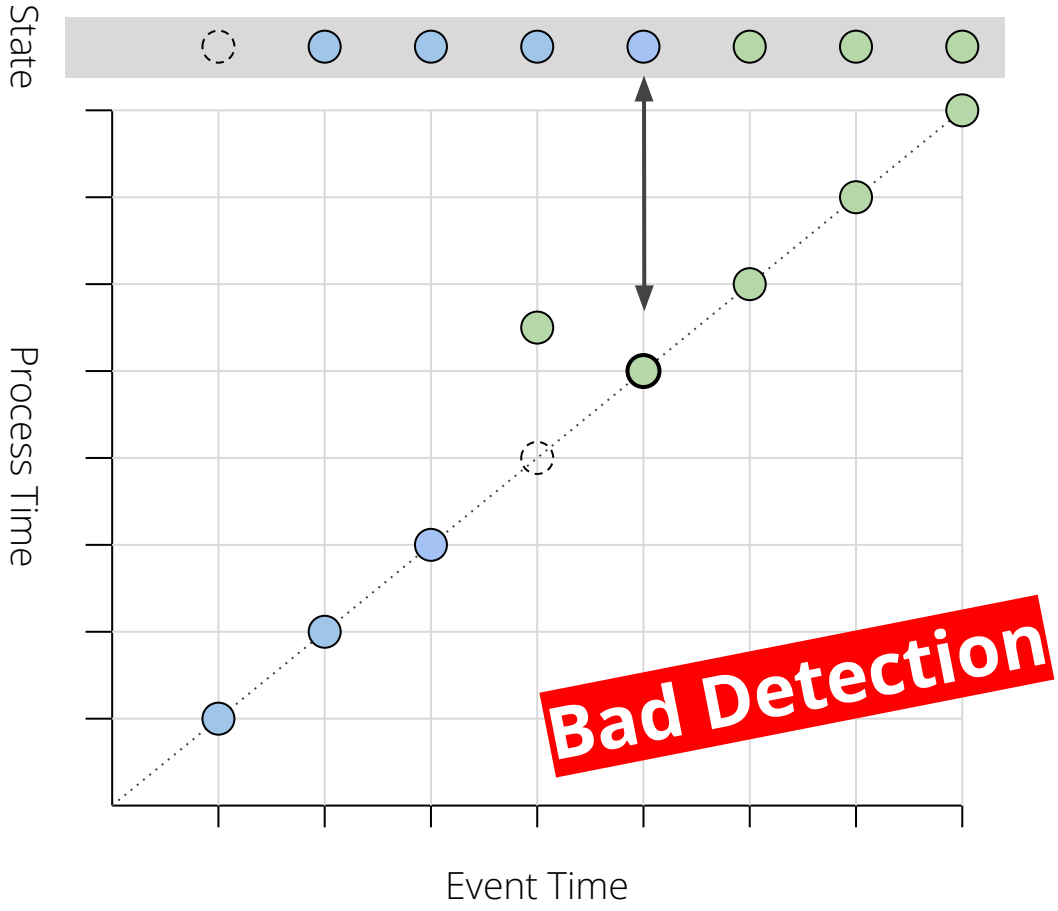
Issues:
Using Beam State

BEAM SUMMIT Austin, 2022

Beam State to Detect Change-Points

```
DoFn<KV<String, T>, T> {
  StateSpec<ValueSpec<T>>
prevSpec =
    StateSpecs.value(...);

  public void processElement(
    ProcessContext c,
    ValueState<T> prev) {
    T curr =
c.element().getValue();
    T last = prev.read();
    if ( curr != last) {
      c.output(curr);
    }
  }
}
```

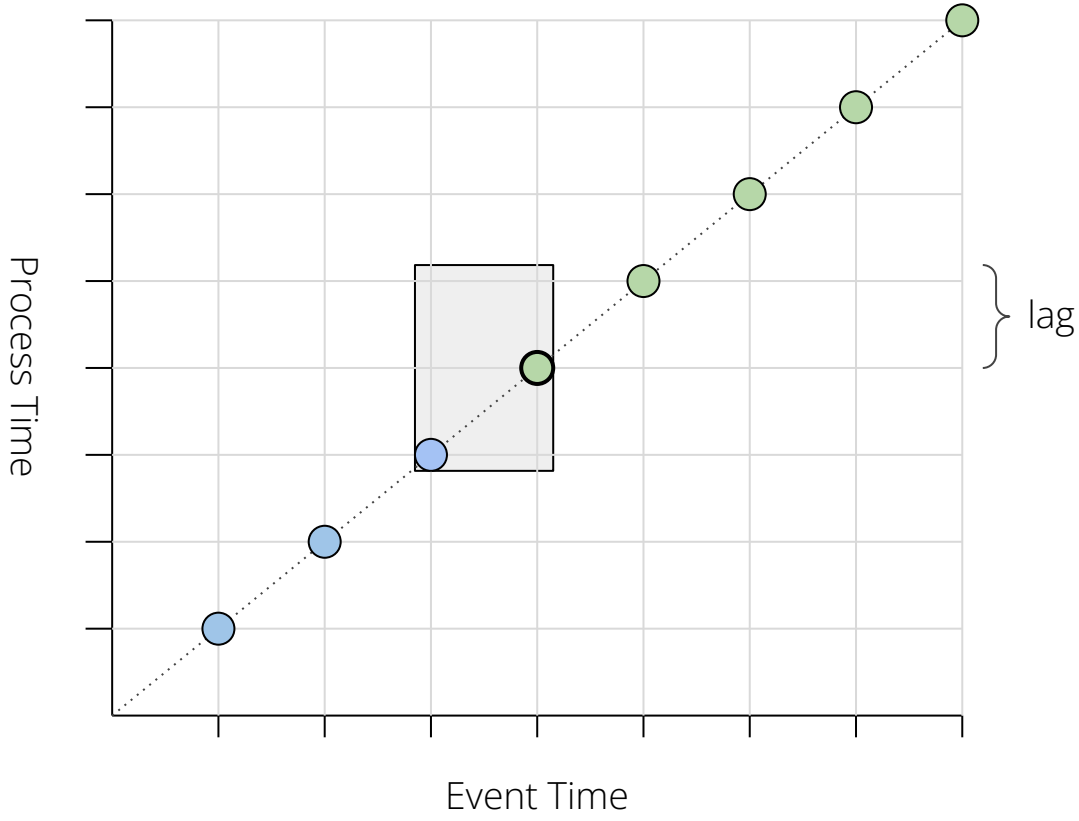


Solution:
Watermark-Triggered Windows

BEAM SUMMIT Austin, 2022

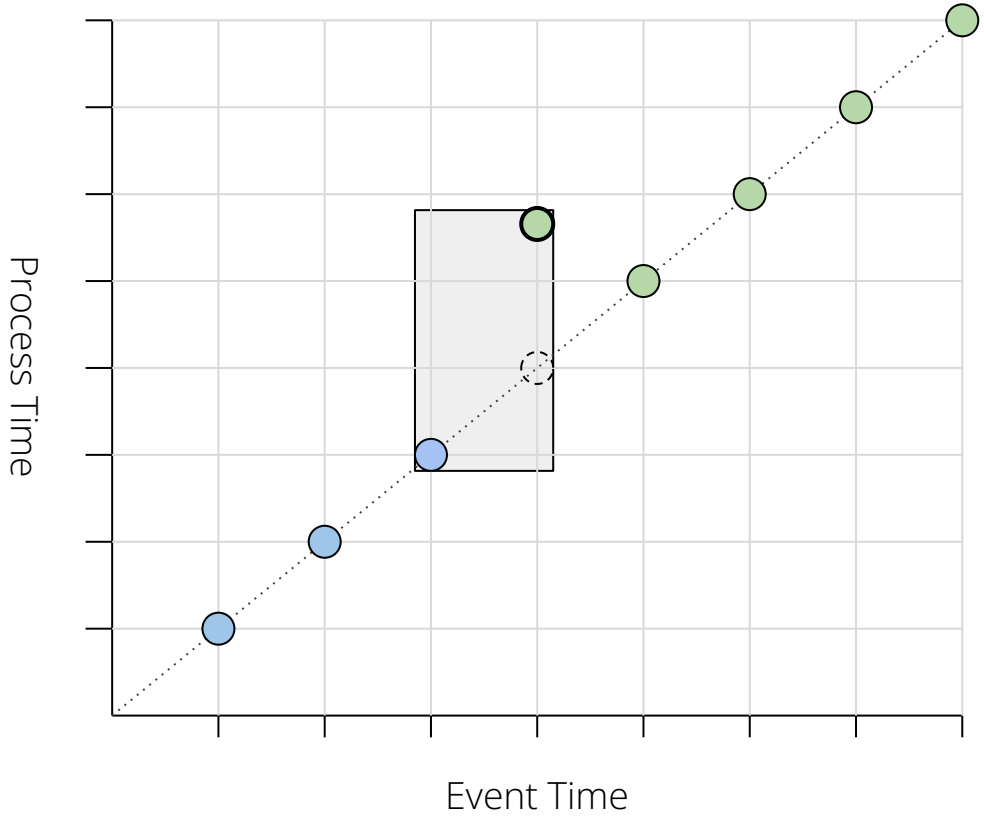
Watermark-Triggered Windows

```
Window
  .<T>into(
    SlidingWindows
      .of(TWO_SECONDS)
      .every(ONE_SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterWatermark
        .pastEndOfWindow()))
```



Watermark-Triggered Windows and Out-of-order Data

```
Window
  .<T>into(
    SlidingWindows
      .of(TWO_SECONDS)
      .every(ONE_SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterWatermark
        .pastEndOfWindow()))
```



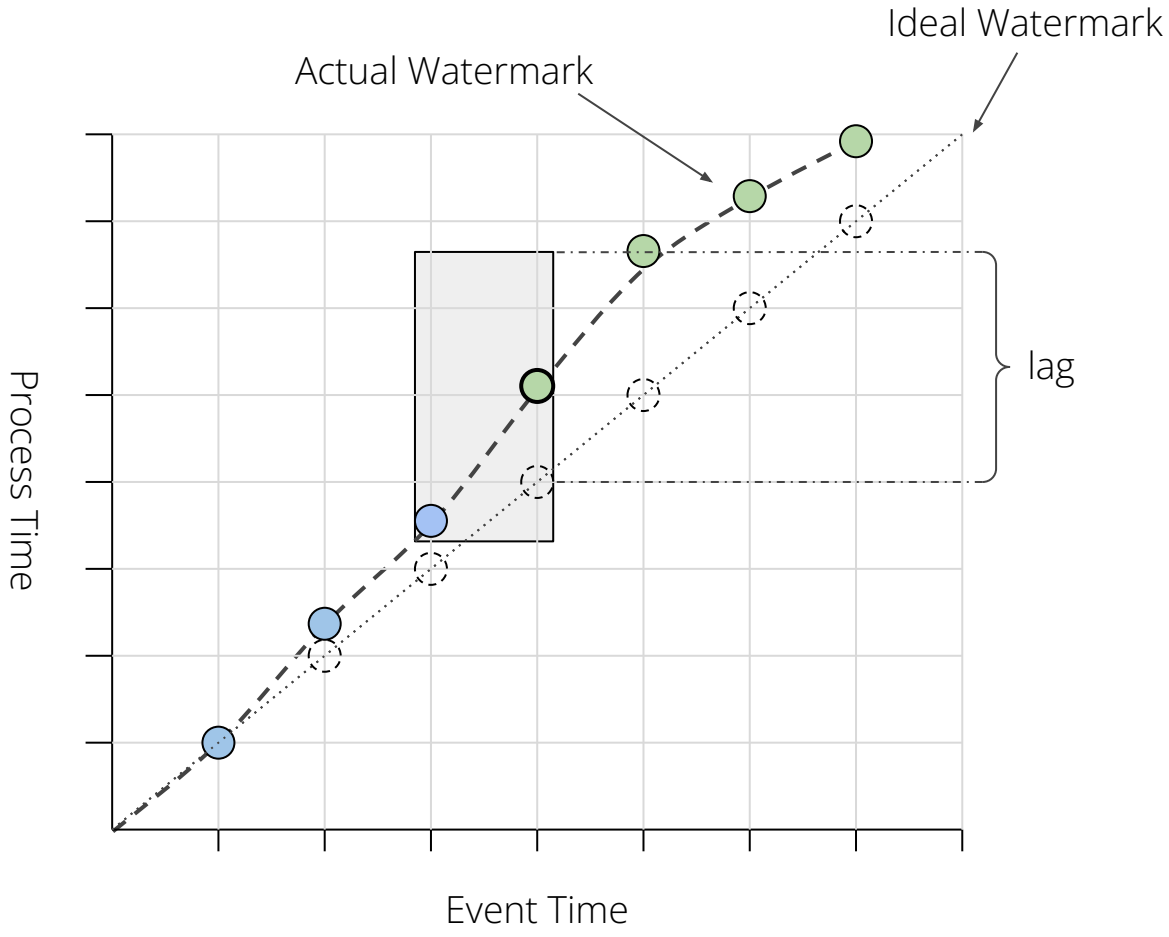
BEAM SUMMIT

Issues:
Watermark-Triggered Windows and Lag

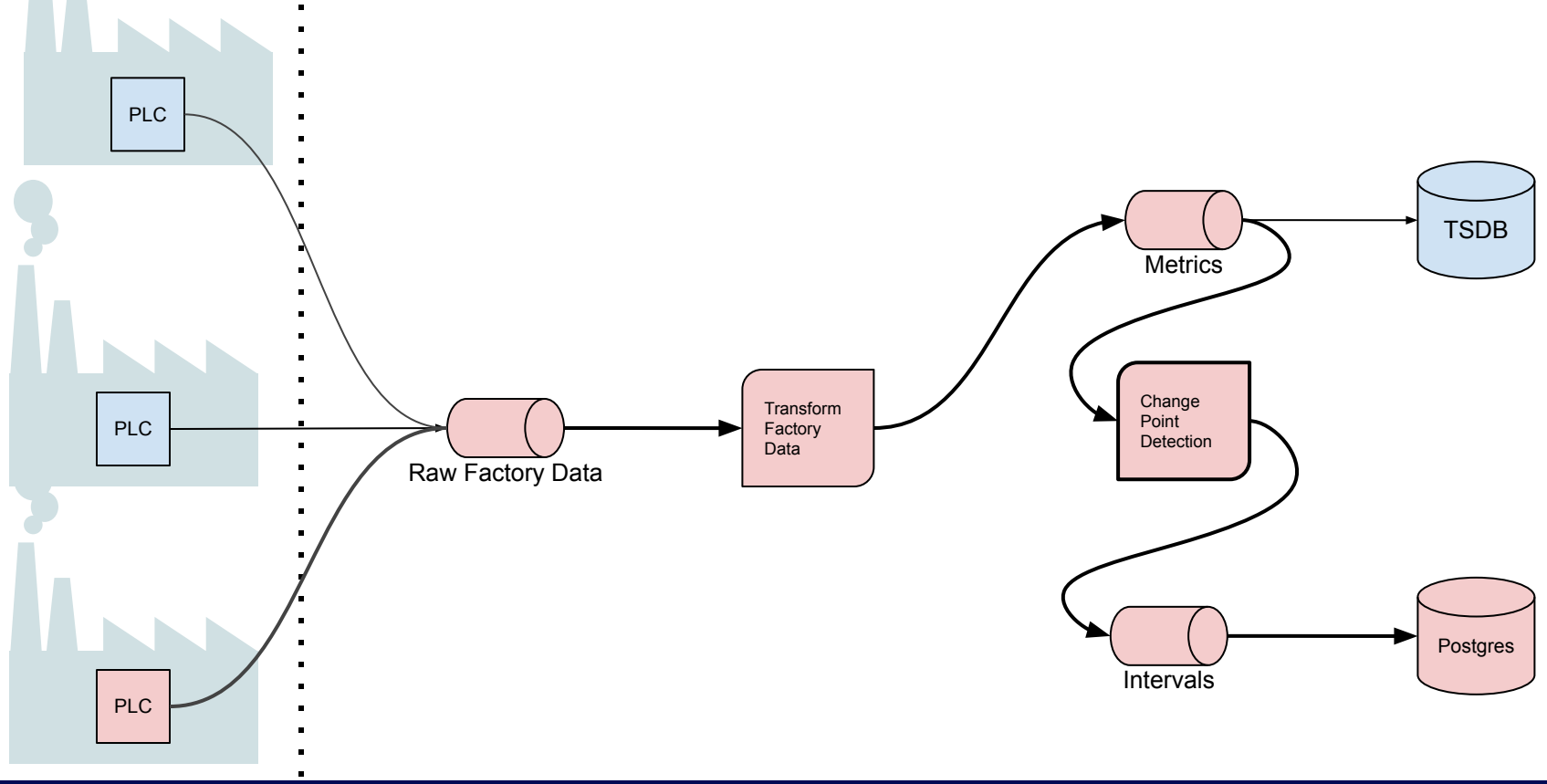
BEAM SUMMIT Austin, 2022

Watermark-Triggered Windows and Lagging Data

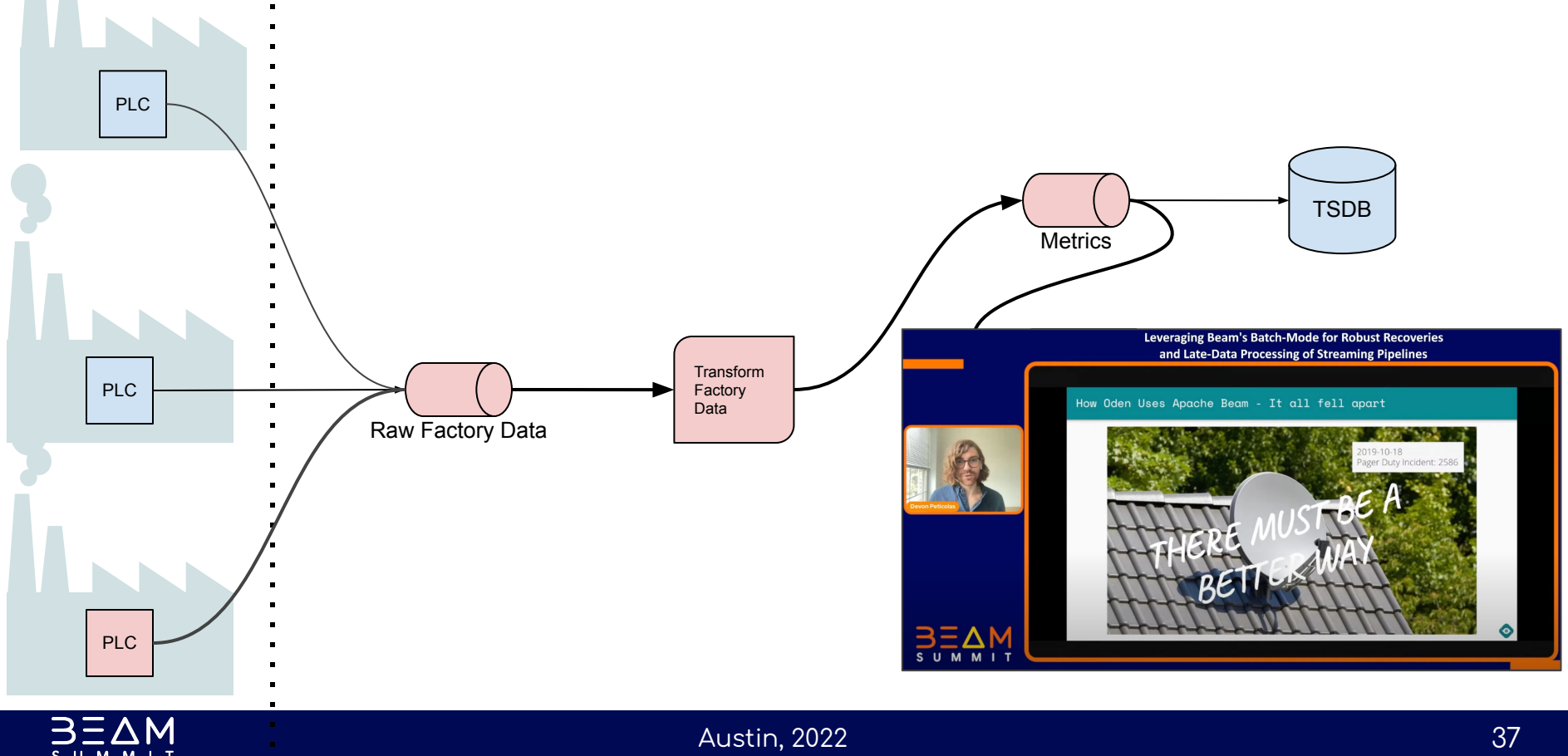
```
Window
  .<T>into(
    SlidingWindows
      .of(TWO_SECONDS)
      .every(ONE_SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterWatermark
        .pastEndOfWindow()))
```



Non-homogeneous Lag



Non-homogeneous Lag



Watermark-Triggered Windows and Lagging Data

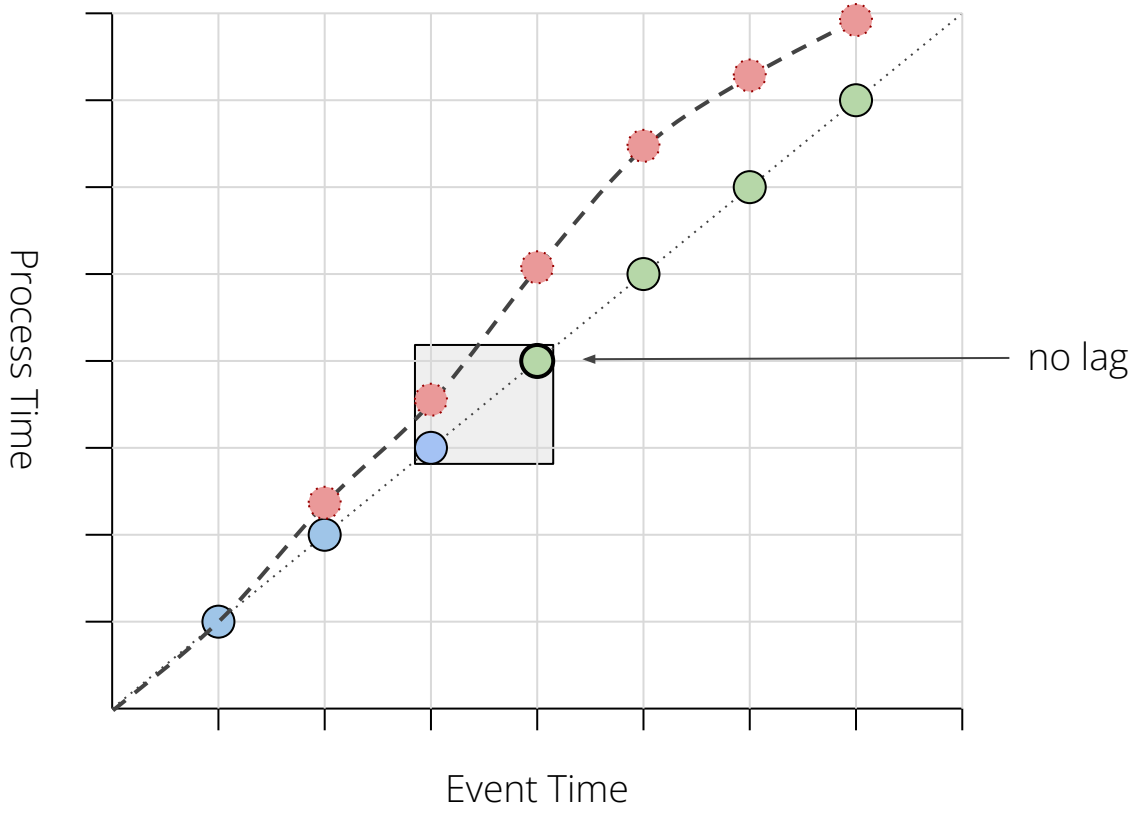
```
Window
  .<T>into(
    SlidingWindows
      .of(TWO_SECONDS)
      .every(ONE_SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterWatermark
        .pastEndOfWindow()))
```

Solution:
Data-Triggered Windows

BEAM SUMMIT Austin, 2022

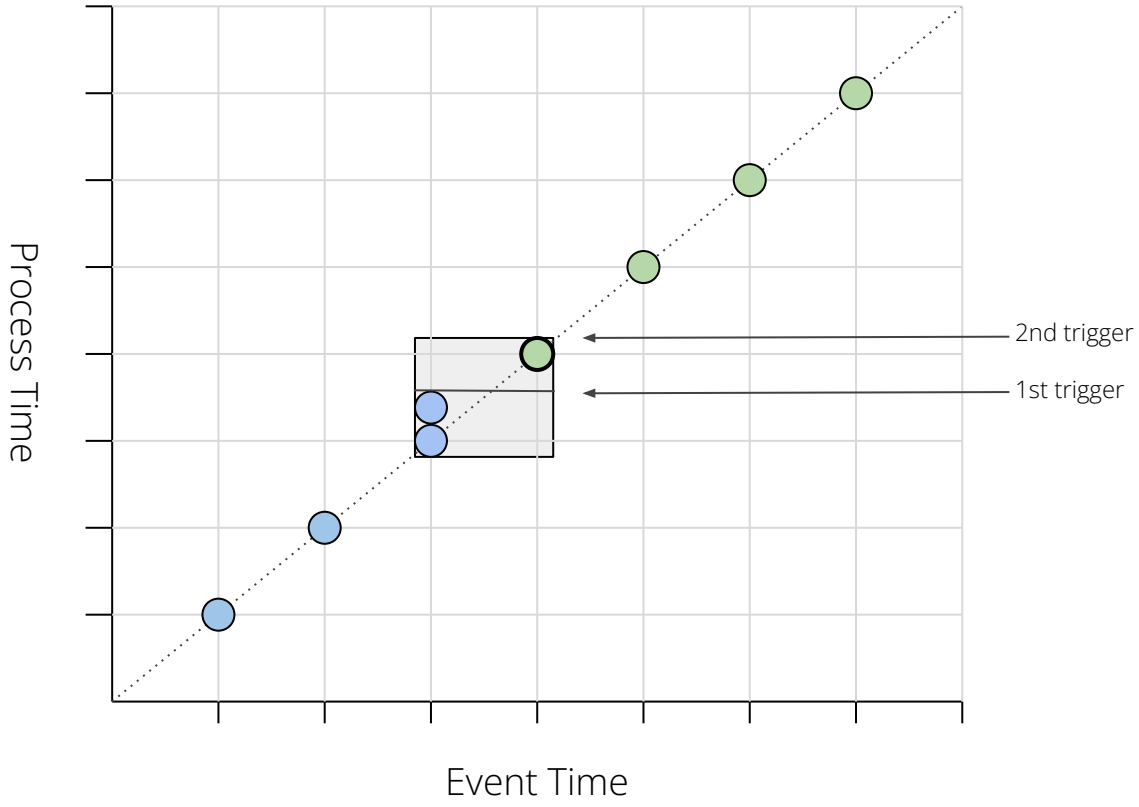
Data-Triggered Windows

```
Window
  .<T>into(
    SlidingWindows
      .of(TWO_SECONDS)
      .every(ONE_SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterPane
        .elementCountAtLeast(2)))
```



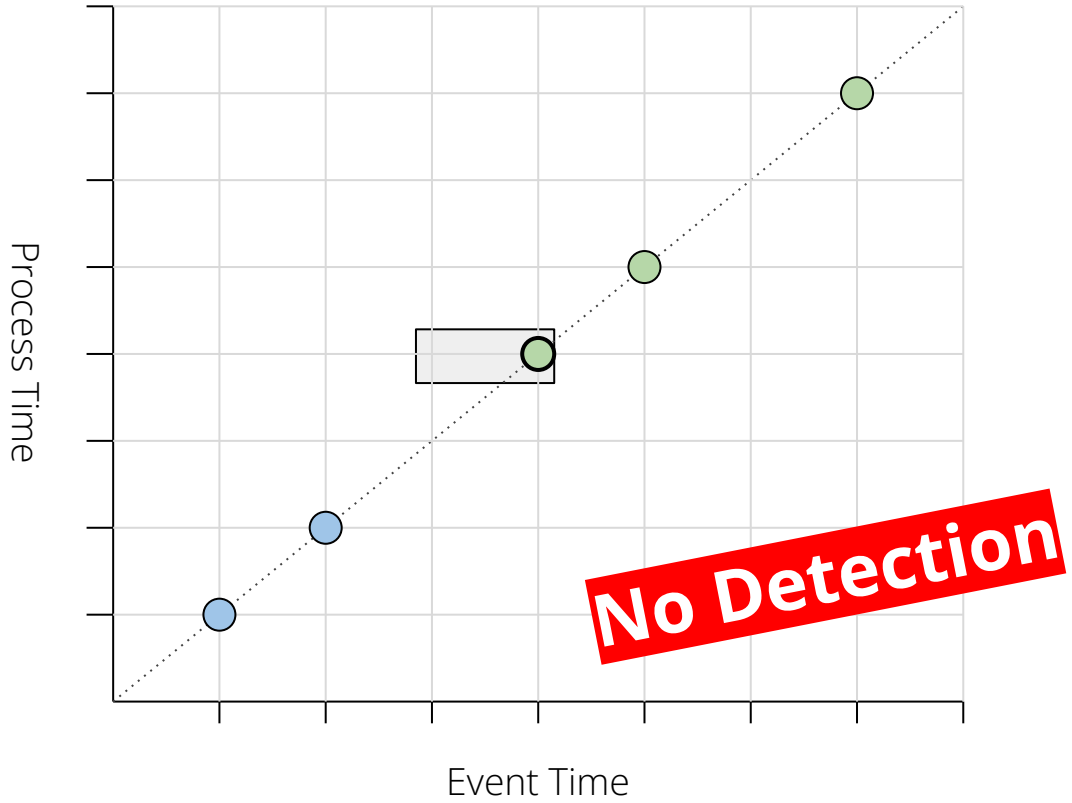
Data-Triggered Windows

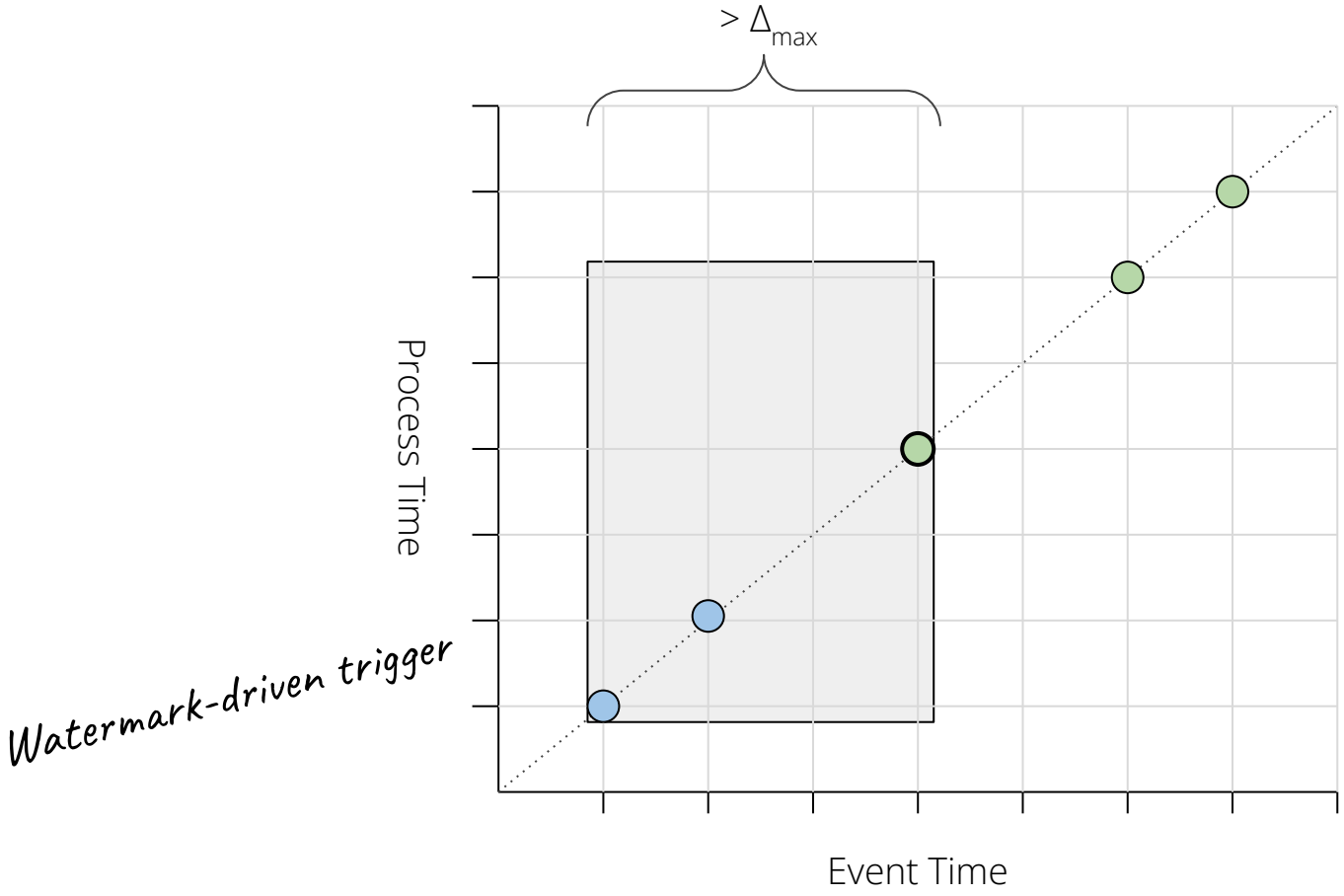
```
Window
  .<T>into(
    SlidingWindows
      .of(TWO_SECONDS)
      .every(ONE_SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterPane
        .elementCountAtLeast(2)))
```

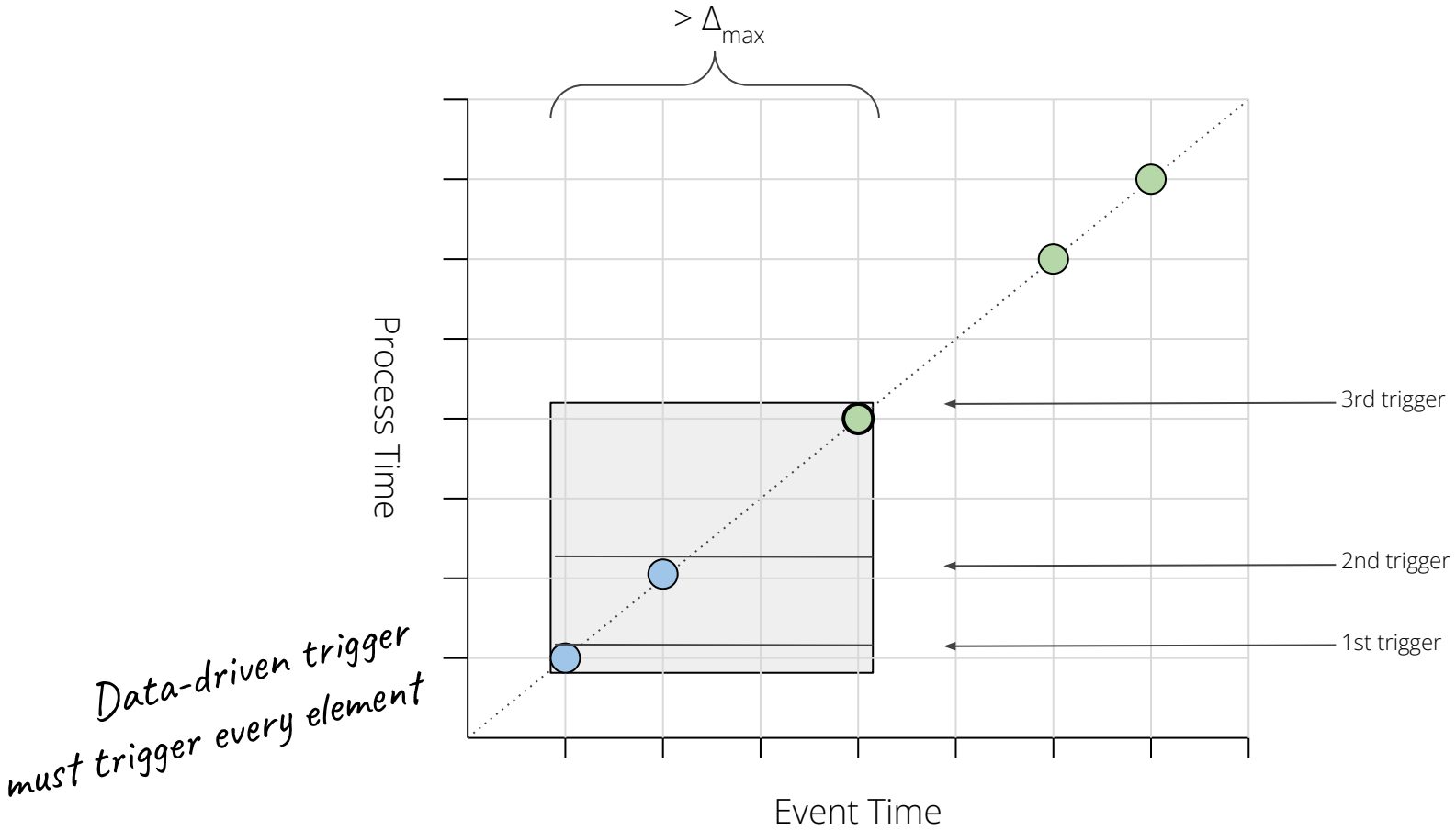


Issues:
Using Windows but Sparse Data

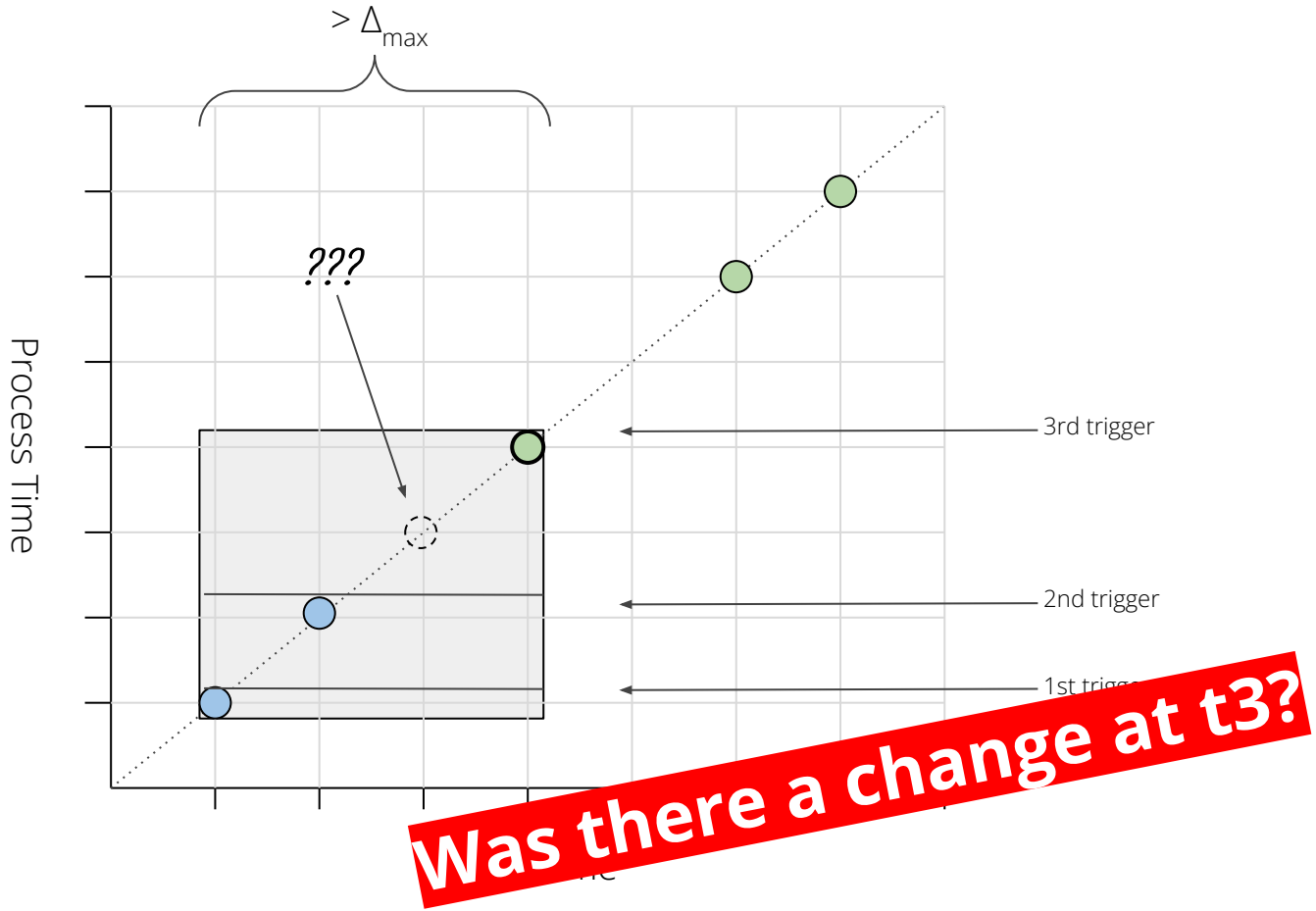
BEAM SUMMIT Austin, 2022


Process Time
Event Time
No Detection


> Δmax
Process Time
Event Time
Watermark-driven trigger

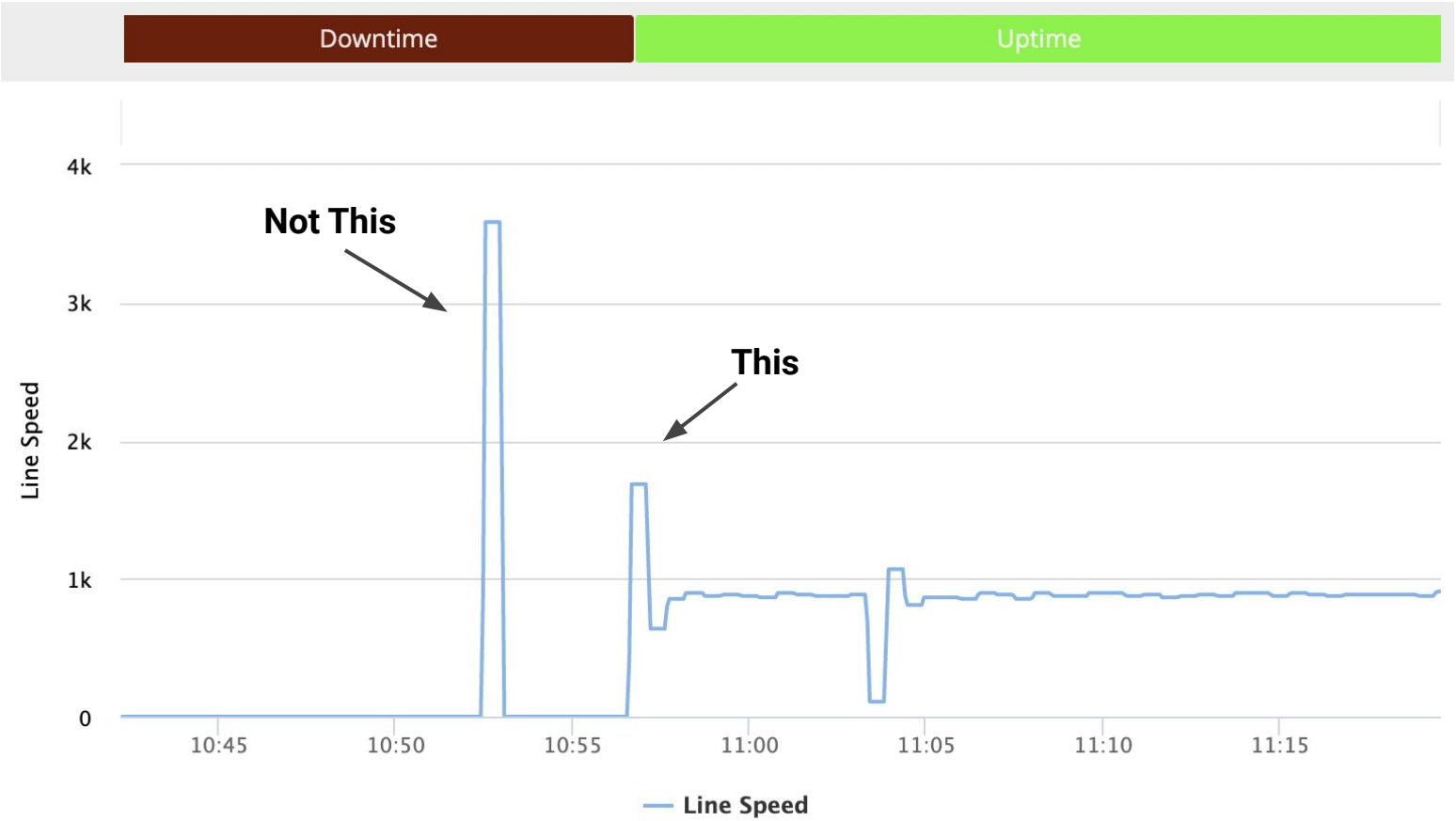


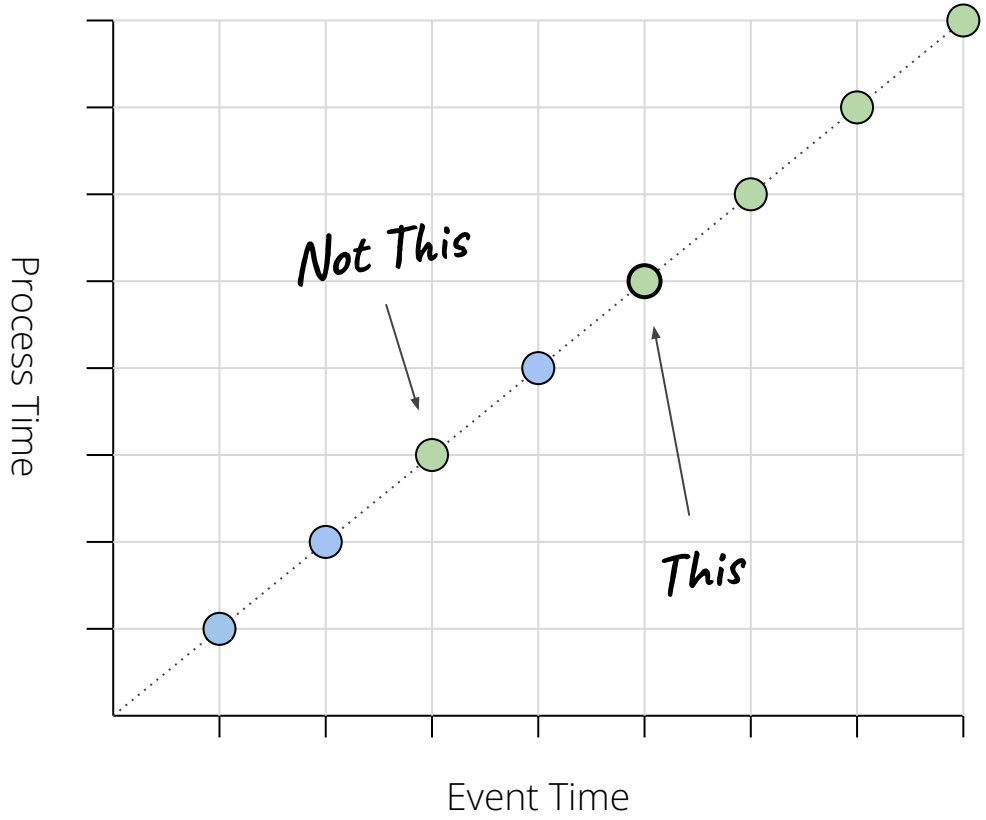
Data-driven trigger must trigger every element

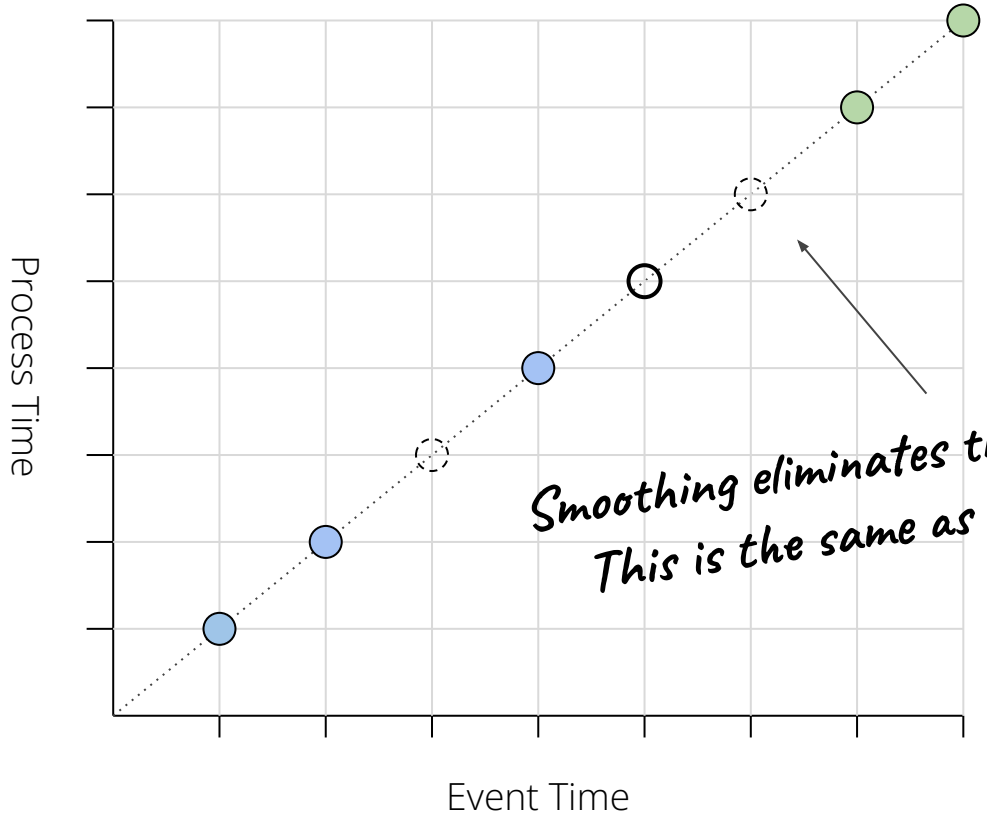

> Δmax
???
Process Time
3rd trigger
2nd trigger
1st trigg
Was there a change at t3?

Use Case:
Creating “Smoothed” Intervals from Metrics

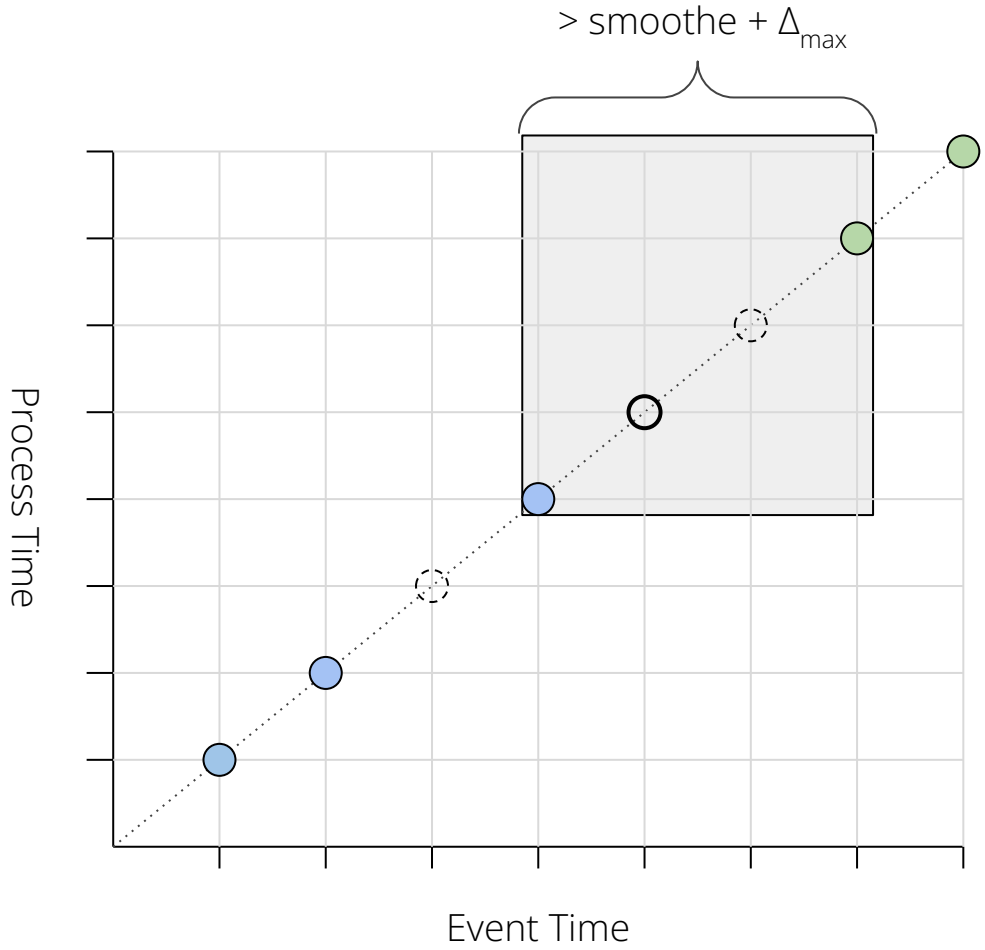
BEAM SUMMIT Austin, 2022

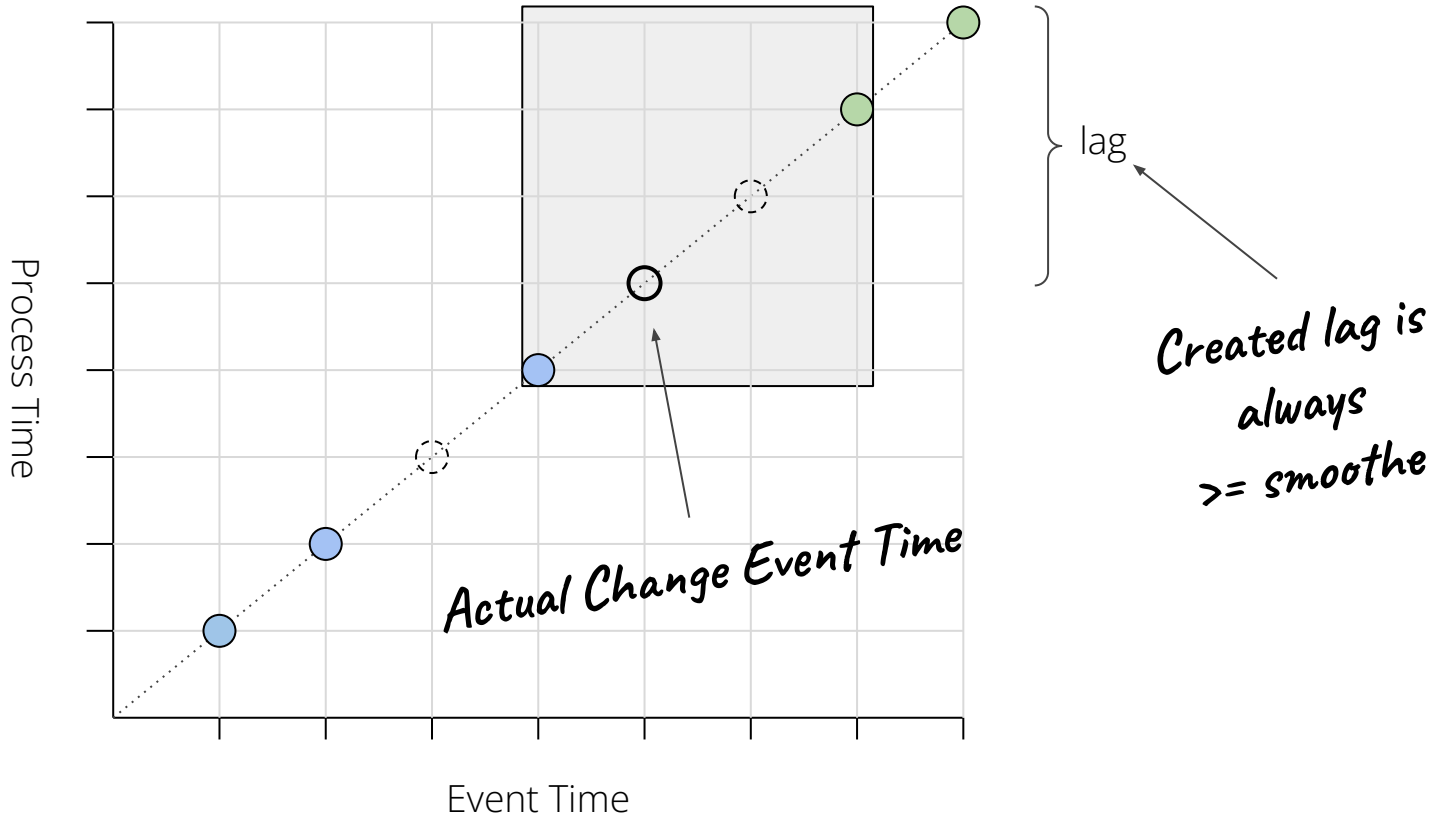

Downtime
Uptime
4k
3k
2k
1k
0
Line Speed
Not This
This
10:45
10:50
10:55
11:00
11:05
11:10
11:15
Line Speed


Not This
This
Process Time
Event Time



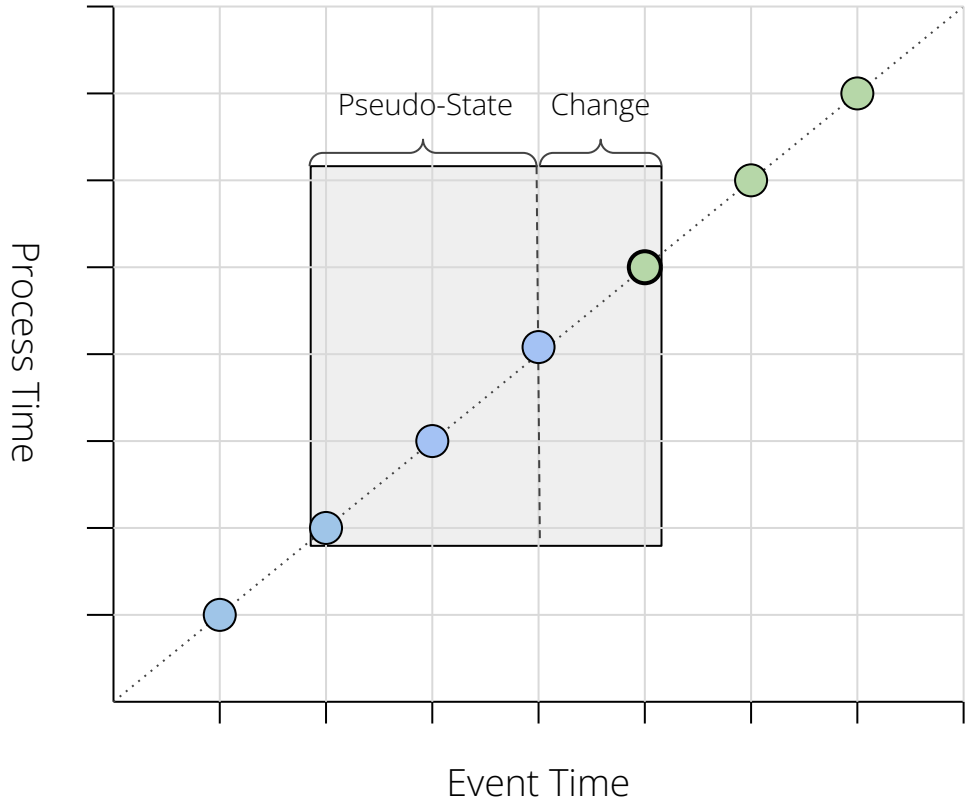
Smoothing eliminates the first n-seconds.
This is the same as our sparse data.

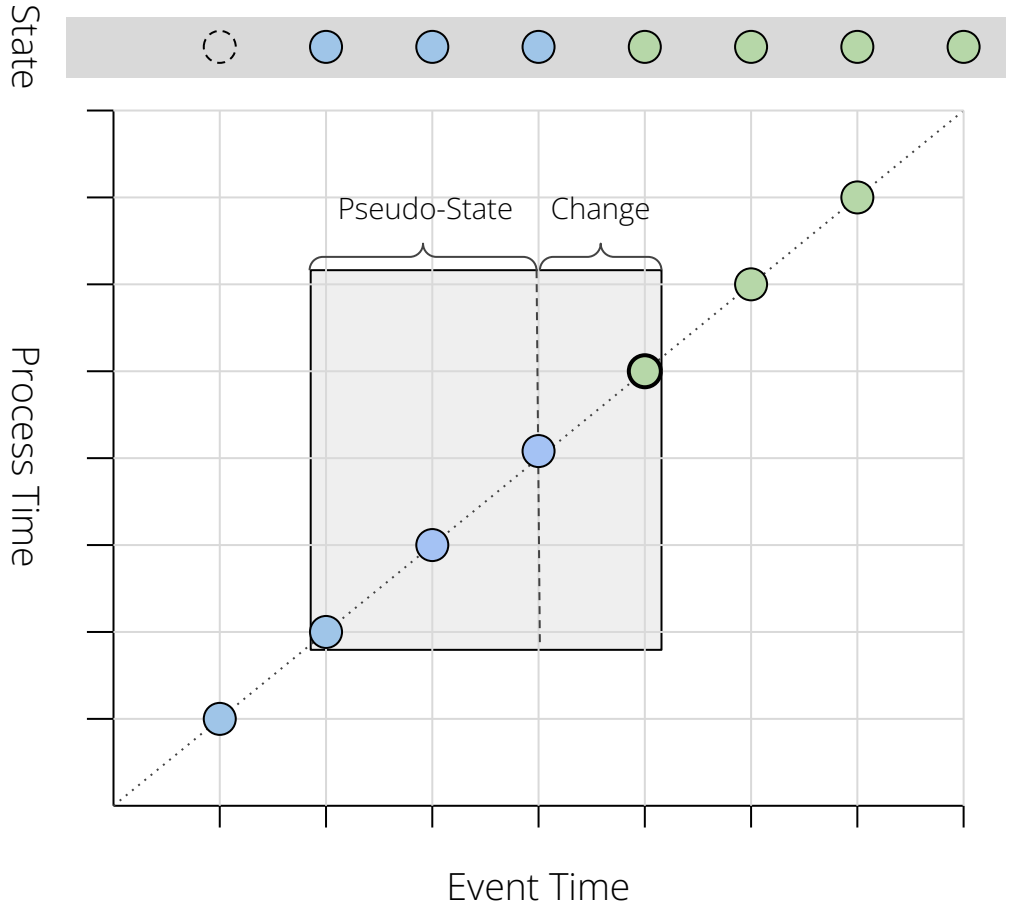

> smoothe + Δmax
Process Time
Event Time

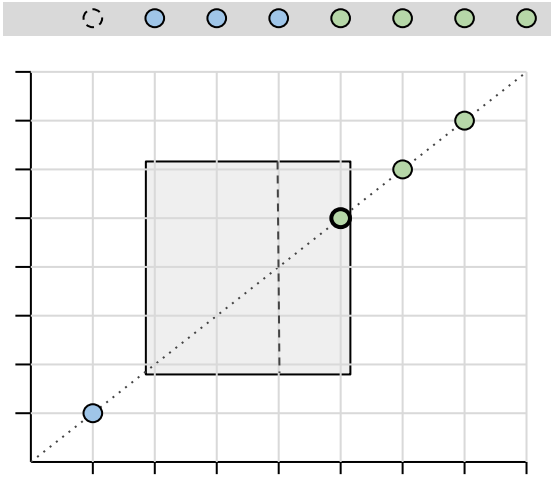

Process Time
Event Time
lag
Actual Change Event Time
Created lag is always >= smoothe

Solution:
State + Sliding Windows

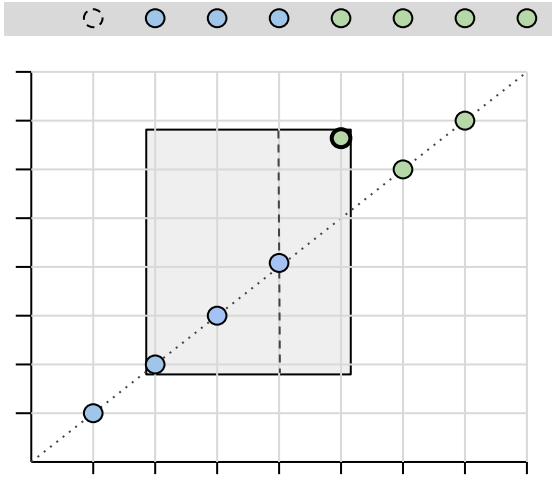
BEAM SUMMIT Austin, 2022


Pseudo-State
Change
Process Time
Event Time

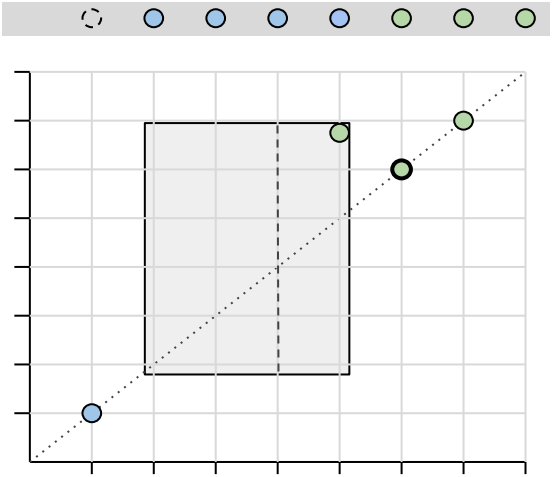

State
Process Time
Pseudo-State
Change
Event Time



✅

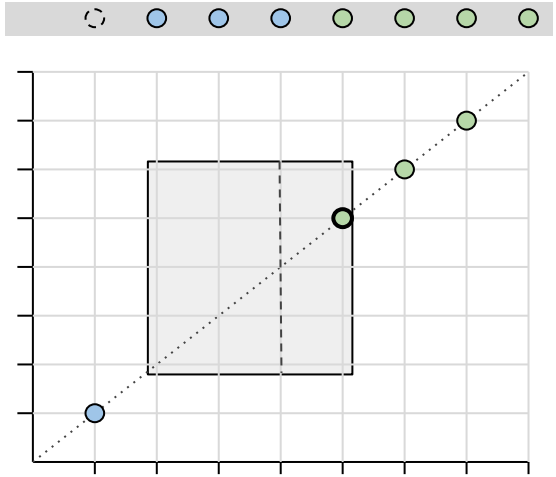


✅

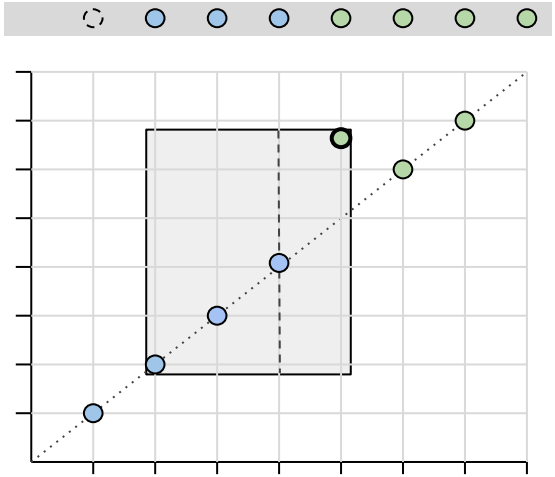


❌

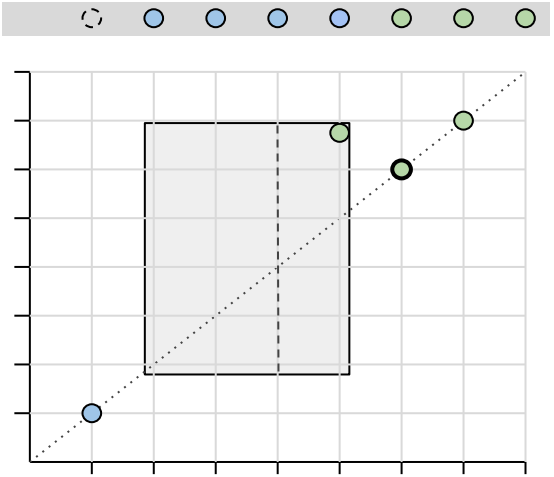
I think this is wrong but...



✅



✅



❌

$$smoothe\text{-}time + event\text{-}time\ \Delta_{max} + out\text{-}of\text{-}order_{max} < process\text{-}time\ \Delta_{min}$$

Key Takeaways

- Oden Uses Change-Point Detection to transform Metrics into Intervals
- Beam State is fast and good at sparsity, but bad at out-of-order
- Windowing is slower and good at out-of-order, but bad at sparsity
- Combining Beam State and Windowing is good at out-of-order and sparsity
- “Smoothed” Change-Point Detection is just a sparsity problem

Thank You

And a special thanks to Jie Zhang, Jake Skelcy, and Deepak Turaga

Questions?

Email: devon@petiocol.as
Github: github.com/x

BEAM SUMMIT Austin, 2022