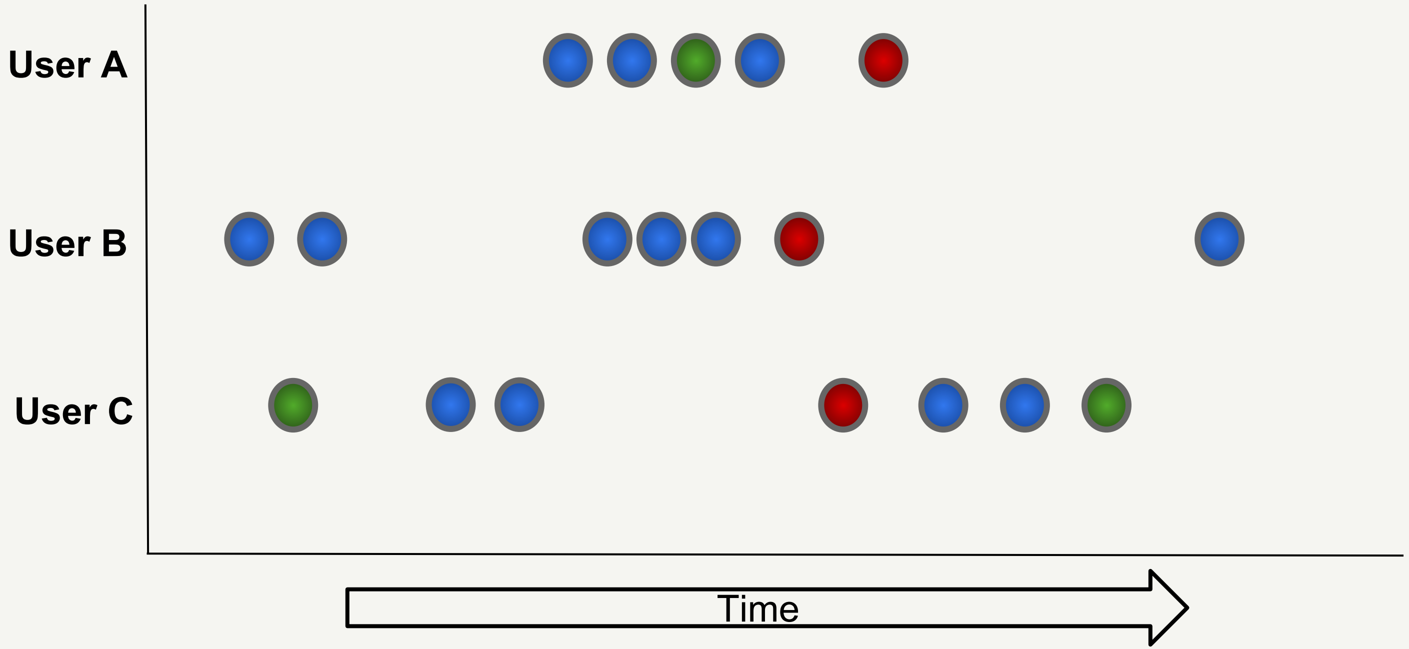
Custom, Complex Windows at Scale Using Apache Flink

Matt Zimmer
QCon San Francisco
14 November | 2017

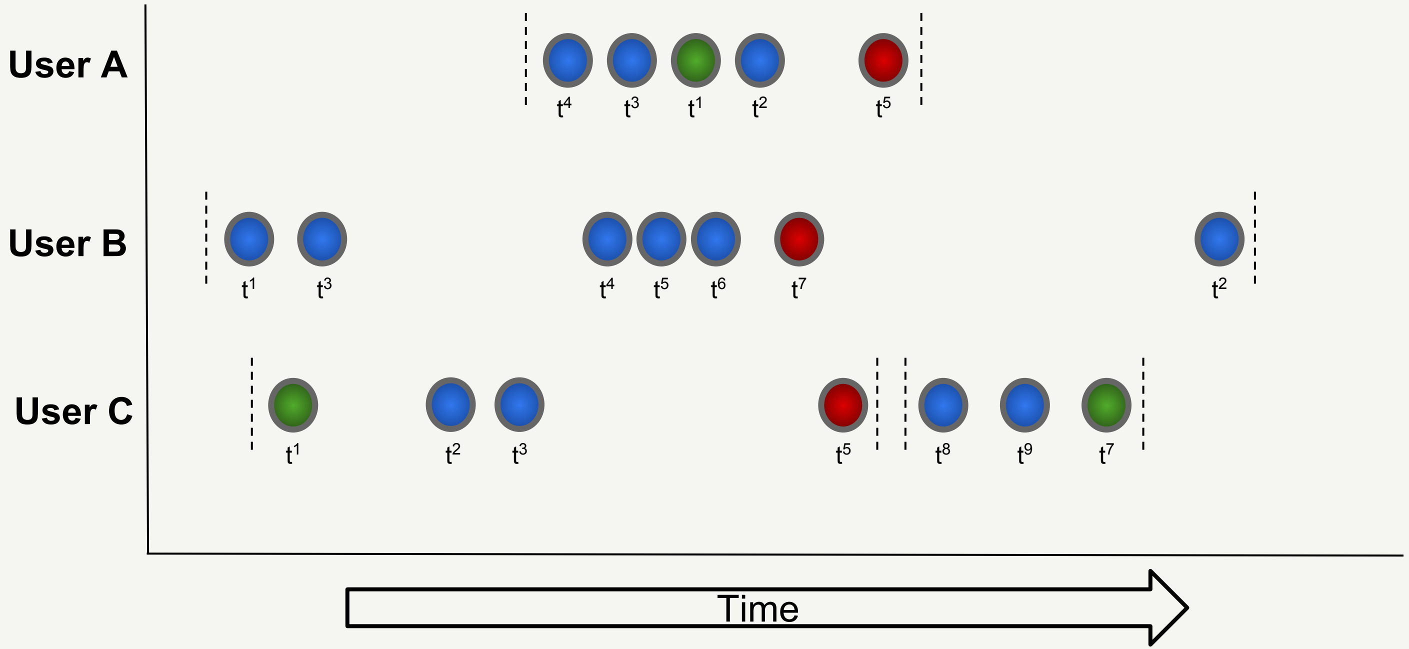
@zimmermatt

Agenda.

- Motivating Use Cases.
- Window Requirements.
- The Solution (Conceptual).
- Event Processing Flow.
- Apache Flink Window API Walk-Through.
- The Solution (Detail).
- Pitfalls to Watch Out For.
- Alternative Implementations.
- Questions.

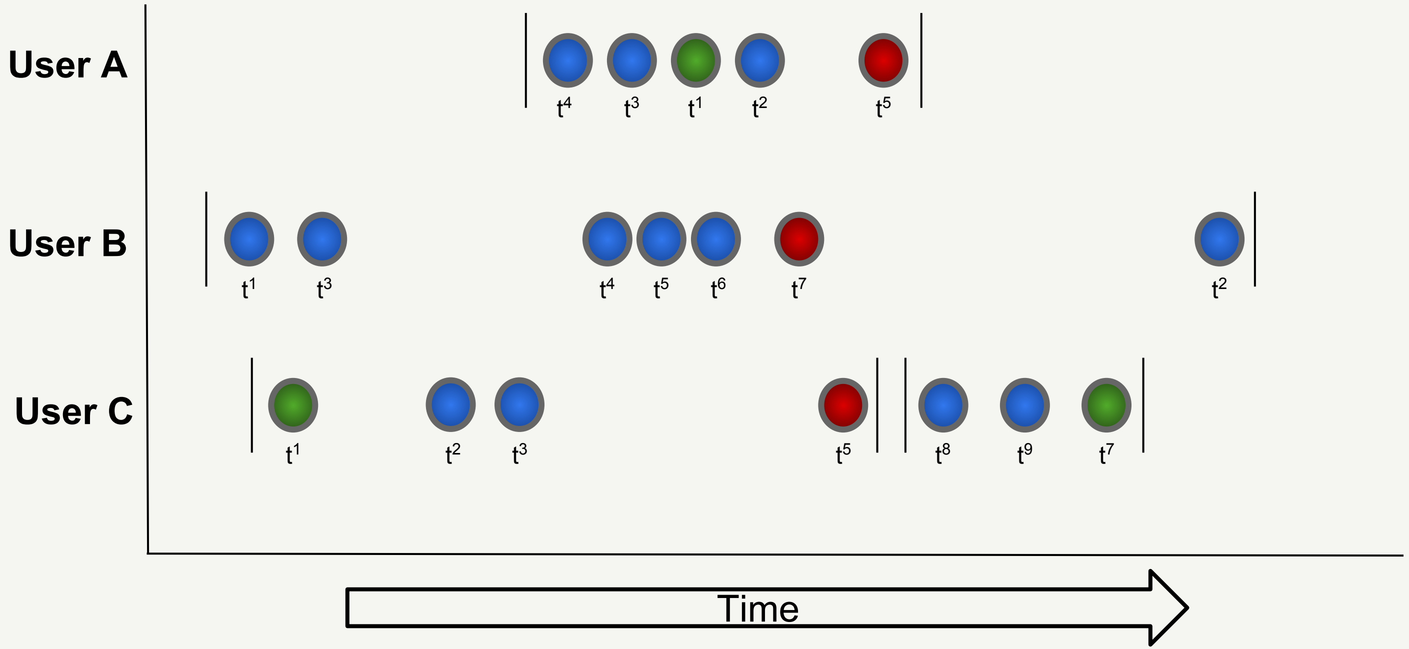

User A
User B
User C
Time

Use Case: Most Simple.



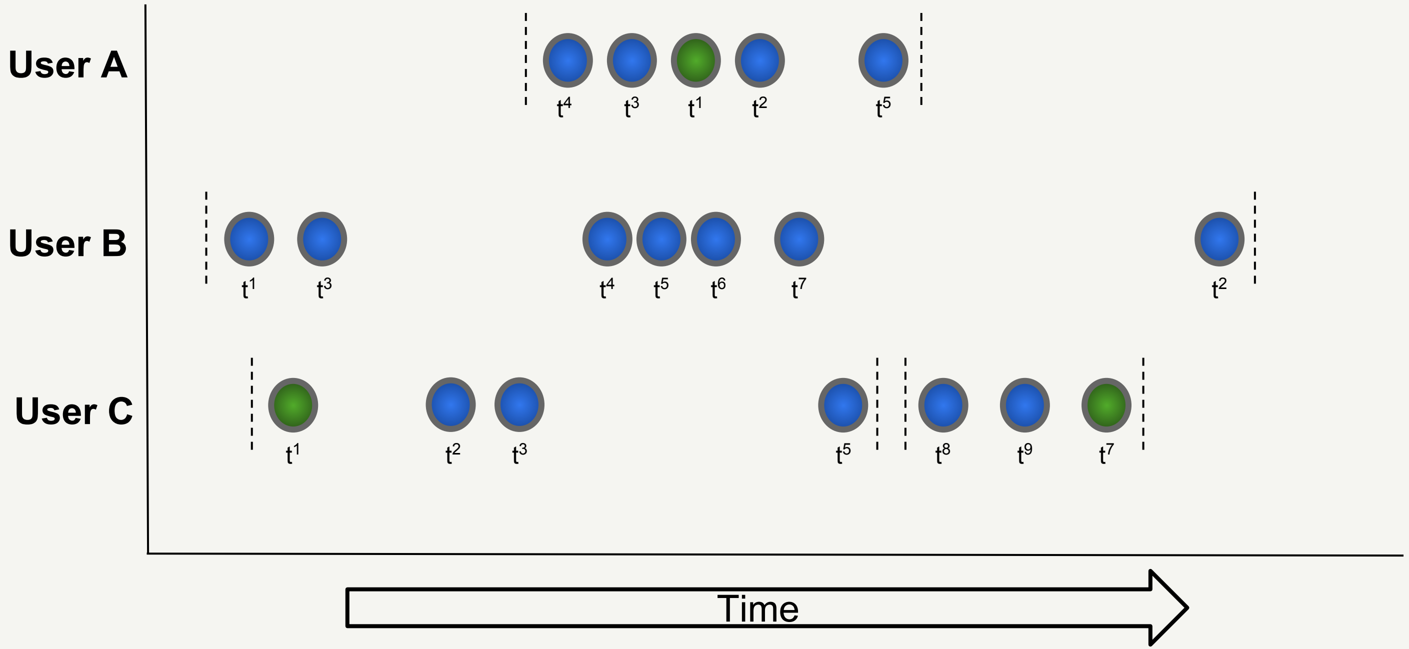
@zimmermatt

Use Case: Most Simple.



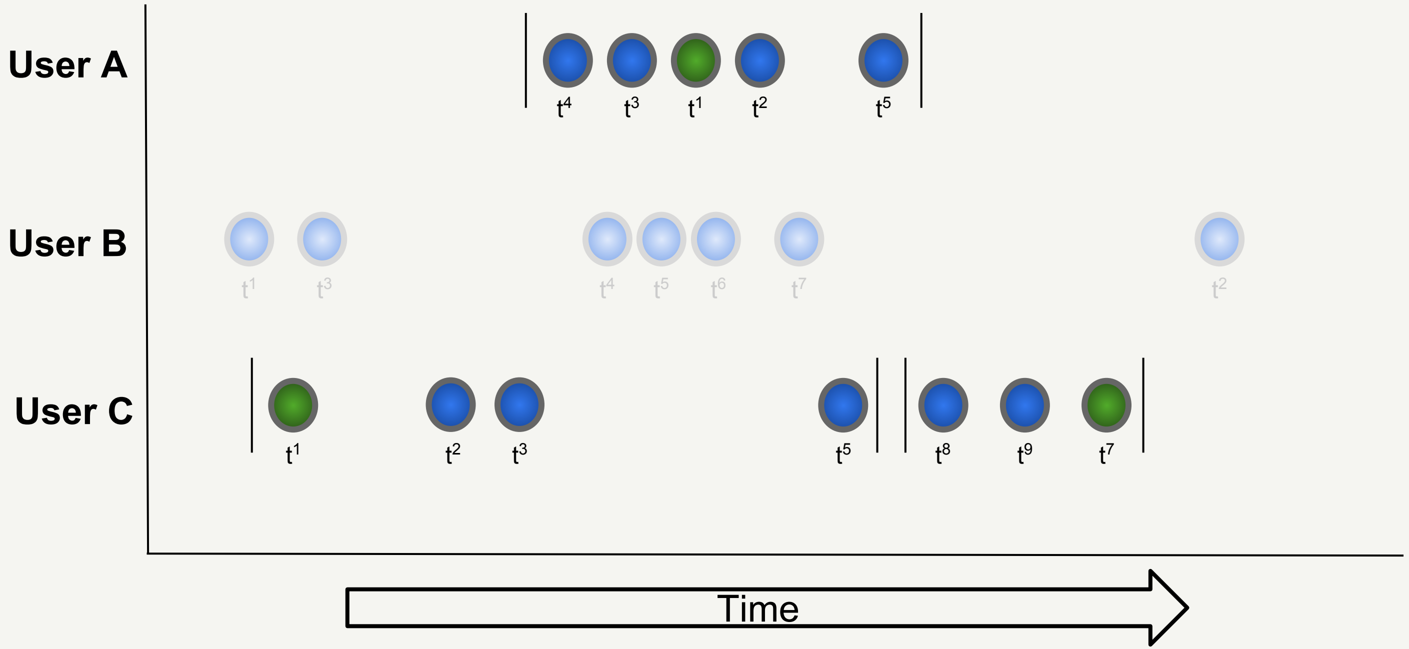
@zimmermatt

Use Case: More Complex.



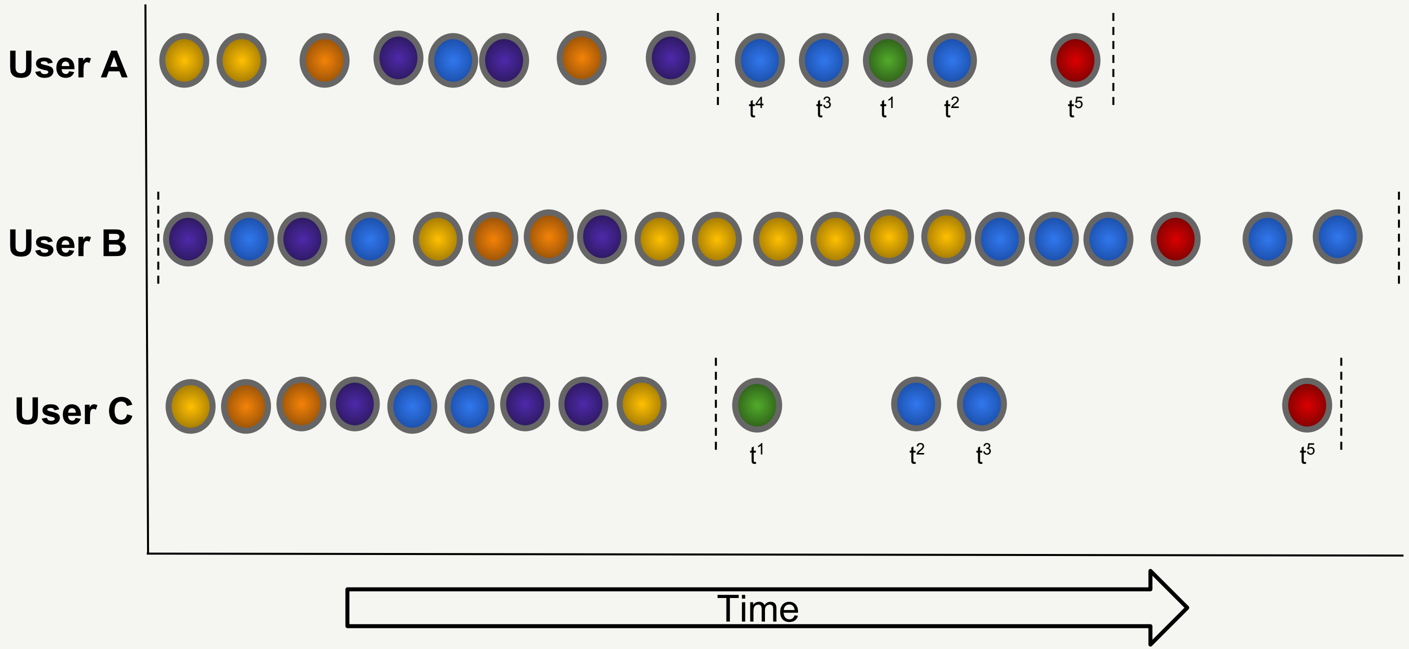
@zimmermatt

Use Case: More Complex.



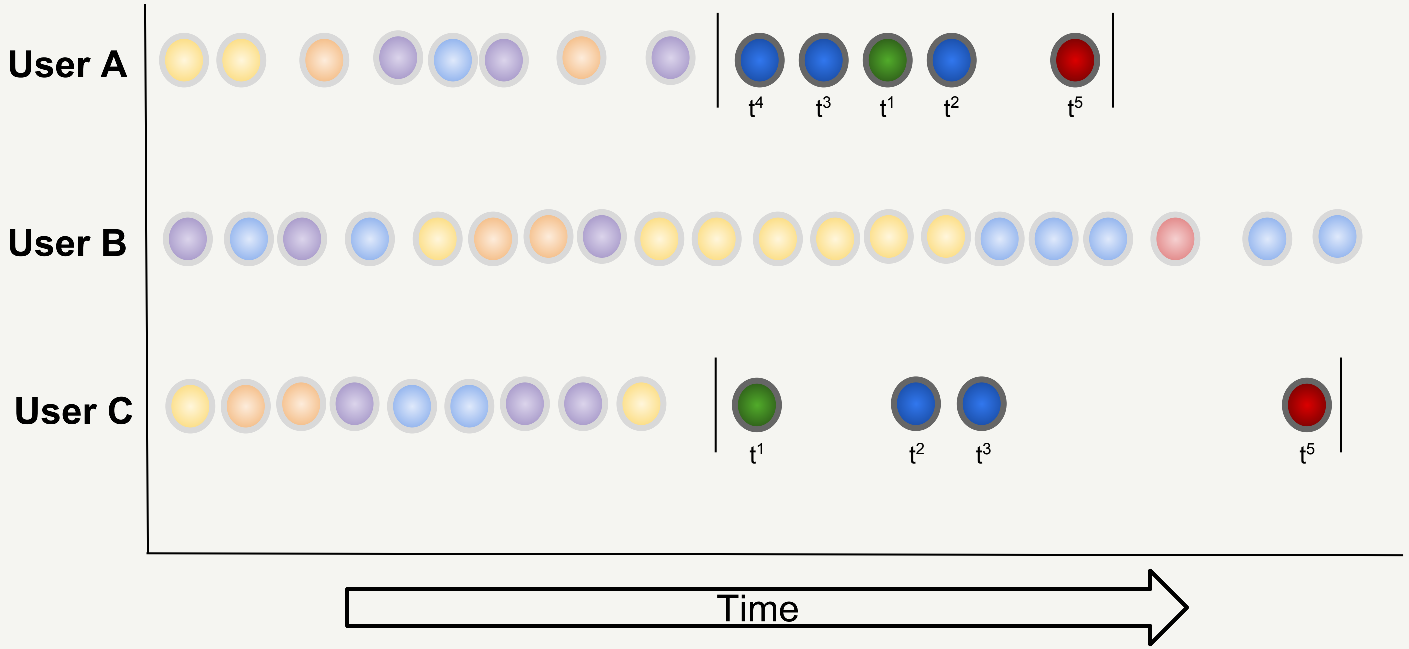
@zimmermatt

Use Case: Most Complex.



@zimmermatt

Use Case: Most Complex.



@zimmermatt

Targeted Scale.

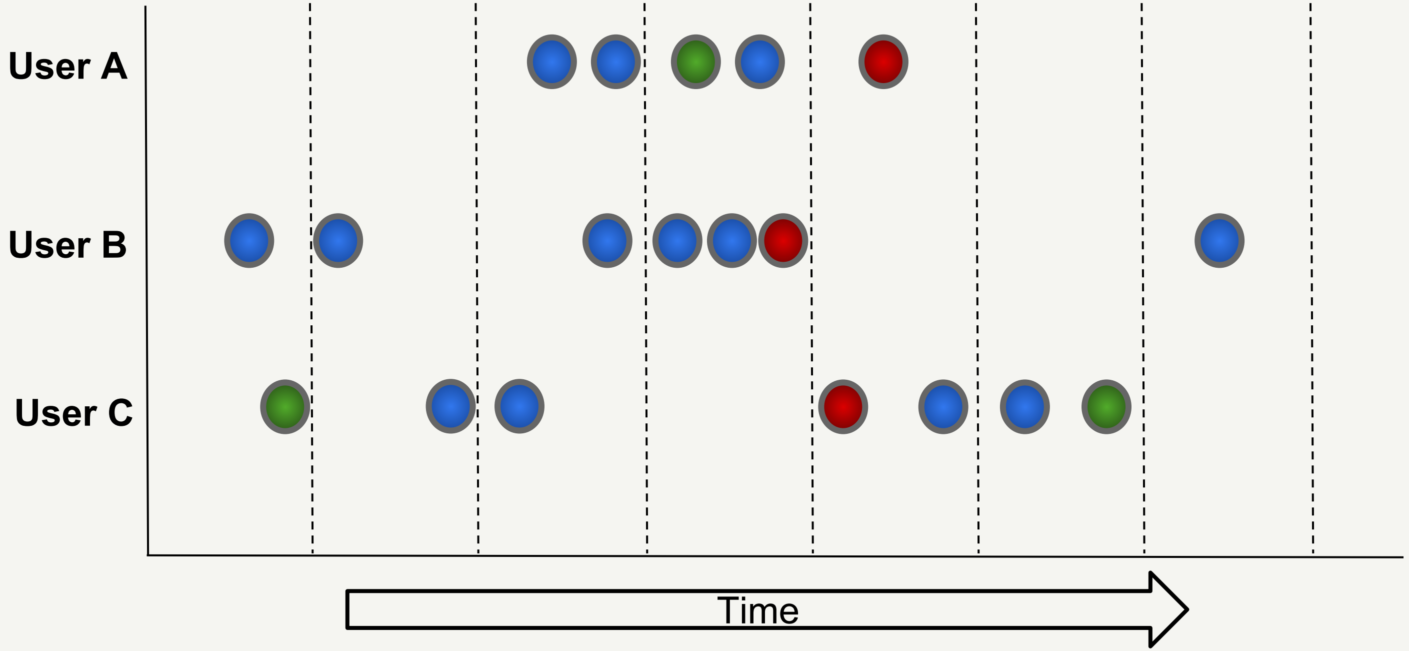
- Events
  - Millions per second
  - 100s billions per day
- Data Flowing In
  - 10s of GiB per second
  - Low (single digit) PiB per day
- State
  - 10s of TiB

Window Requirements.

- Unaligned windows
- Bounded by event type
- Handle out of order events
- Emit early results
- Capture relevant events; ignore the rest

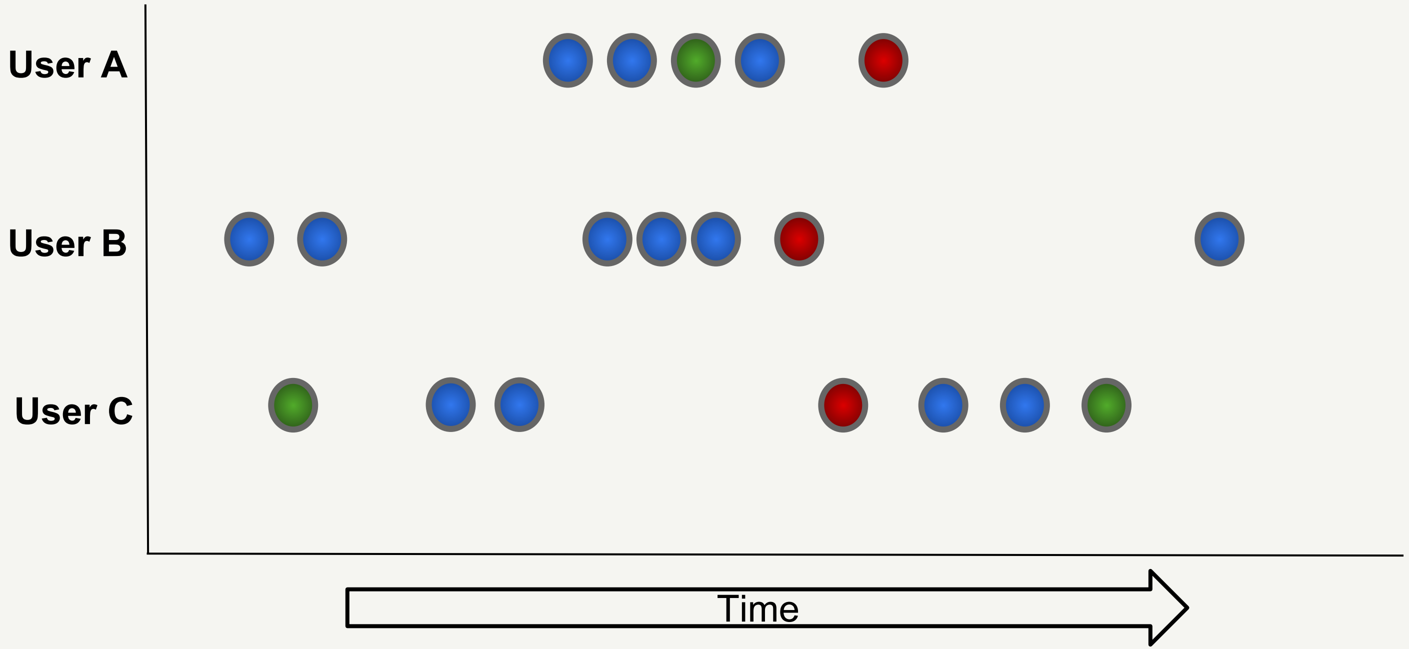
Can we use a standard window type?

Tumbling Window?

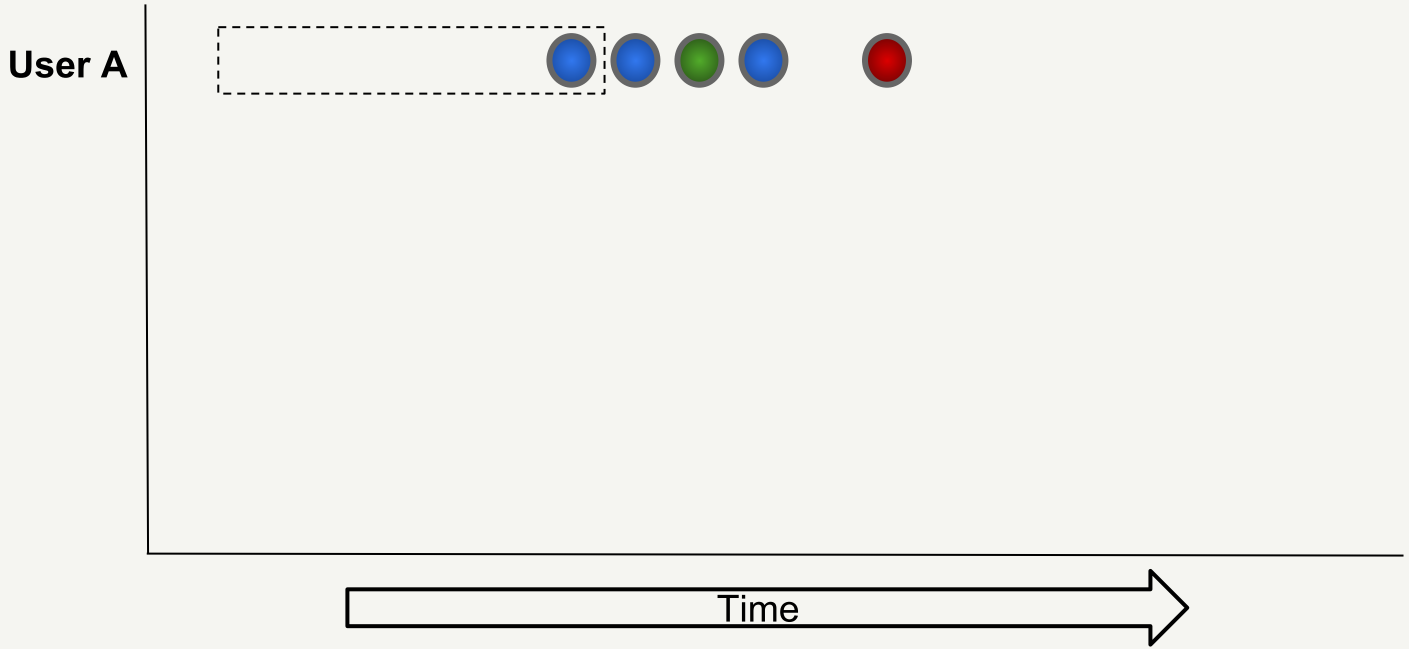


@zimmermatt

Sliding Window?

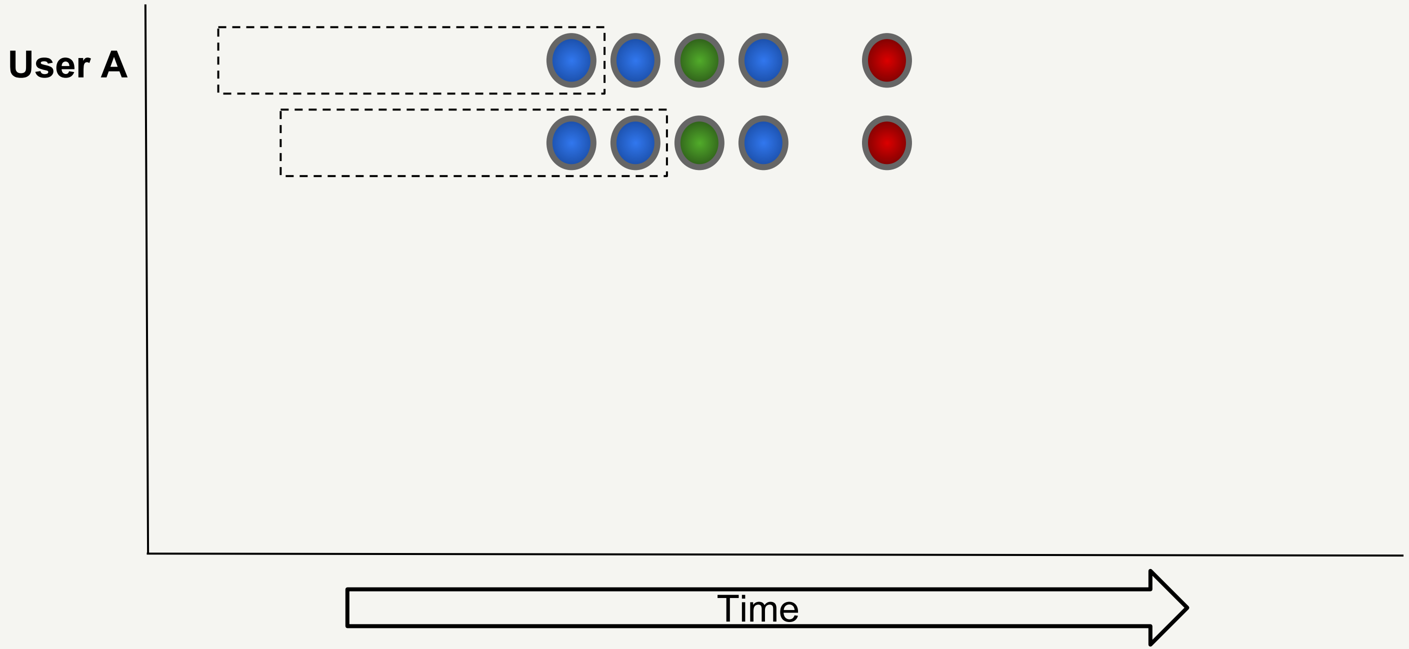


Sliding Window?

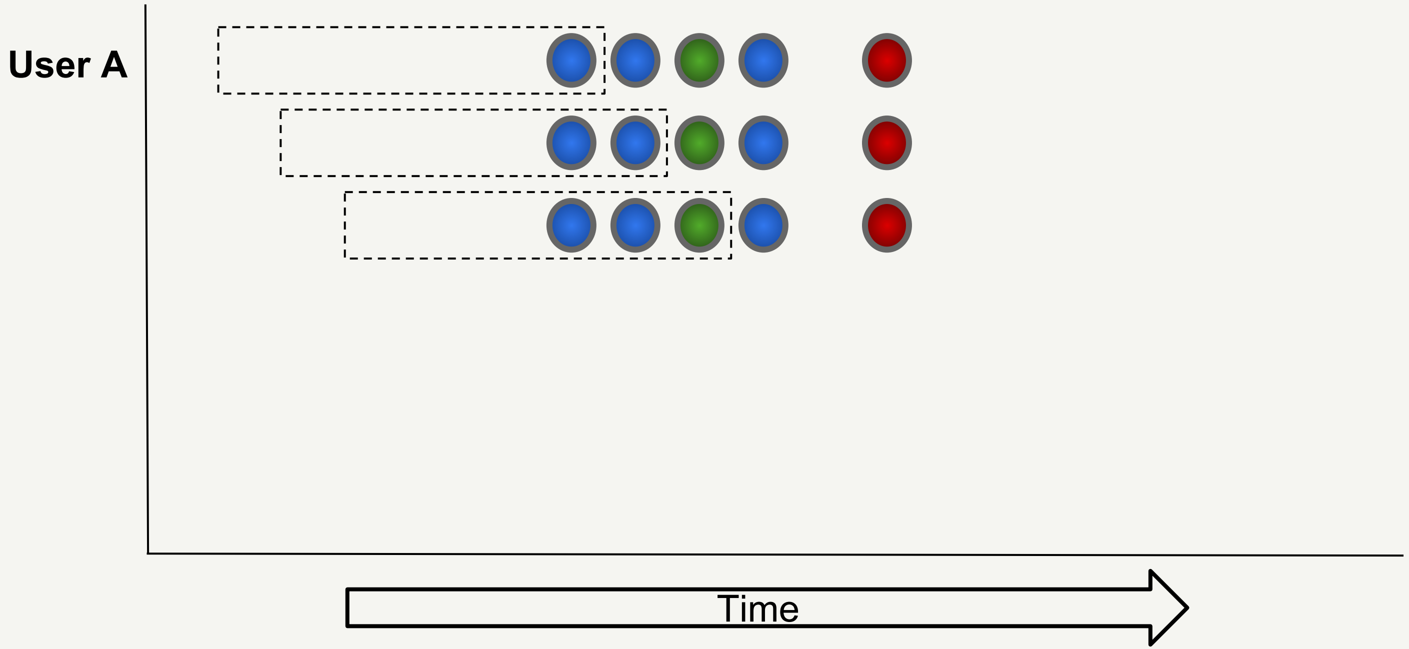


@zimmermatt

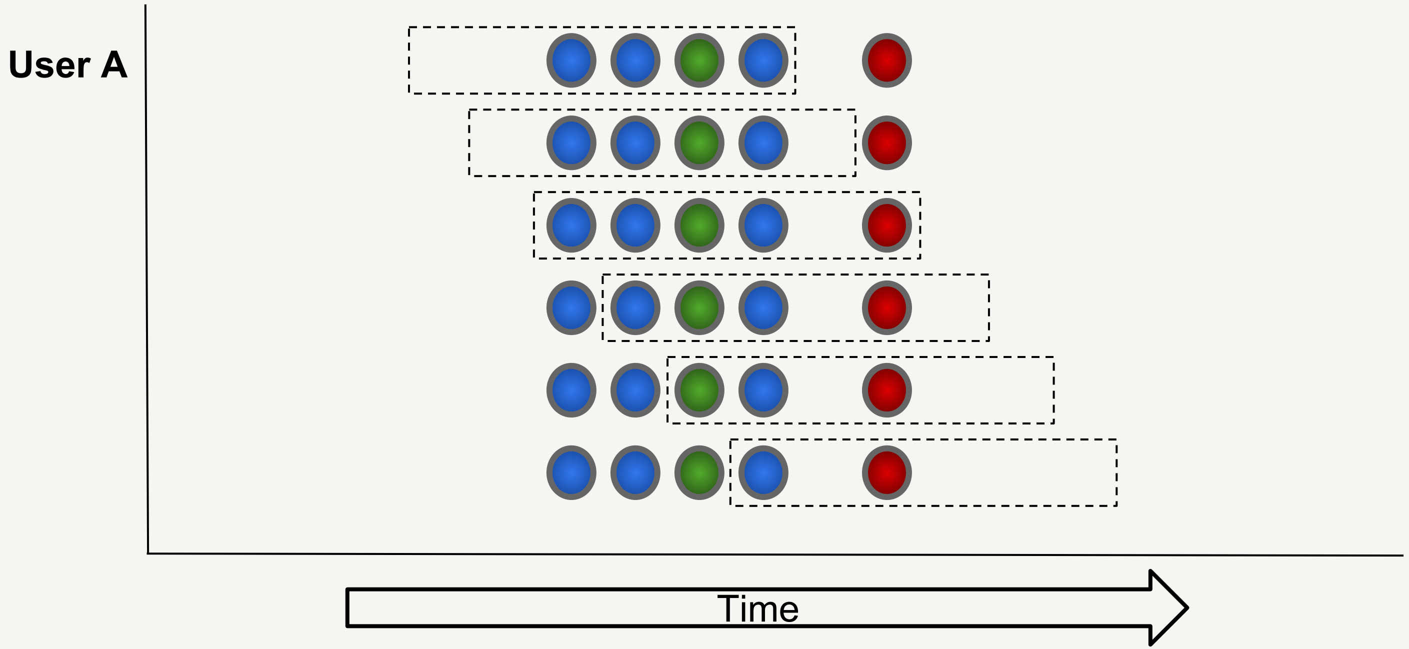
Sliding Window?



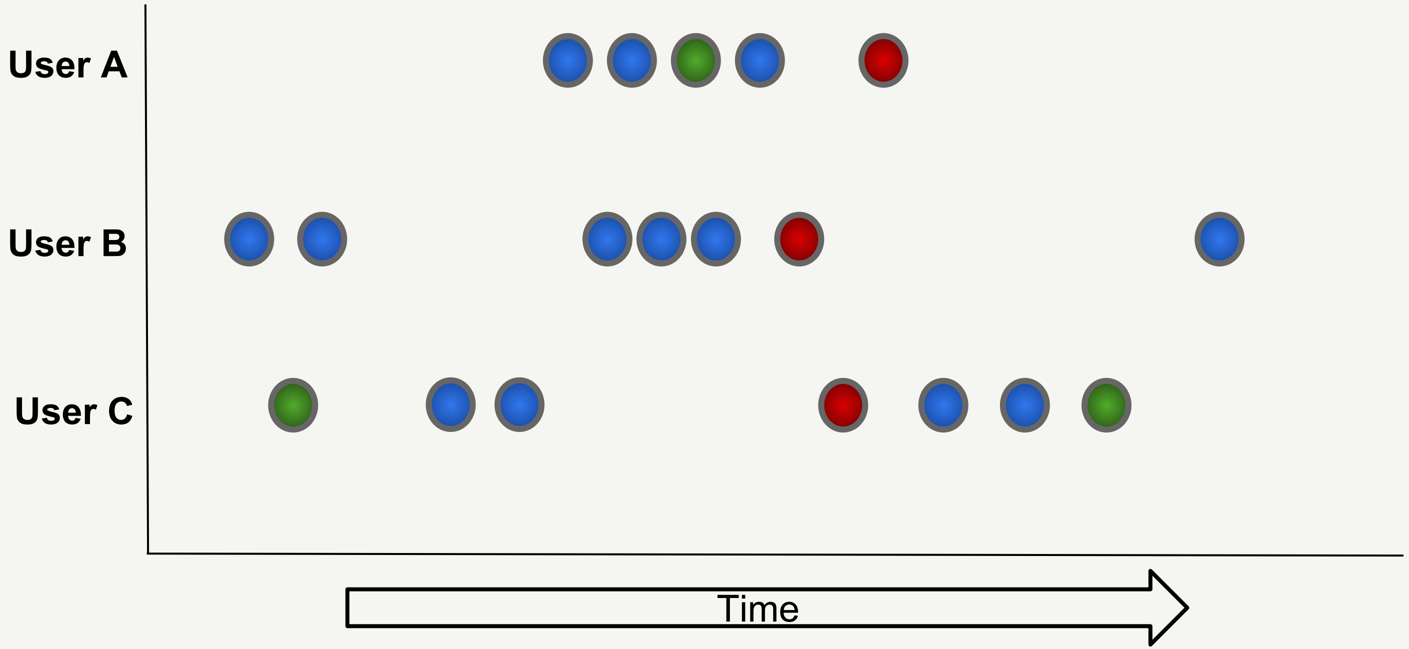
Sliding Window?



Sliding Window?

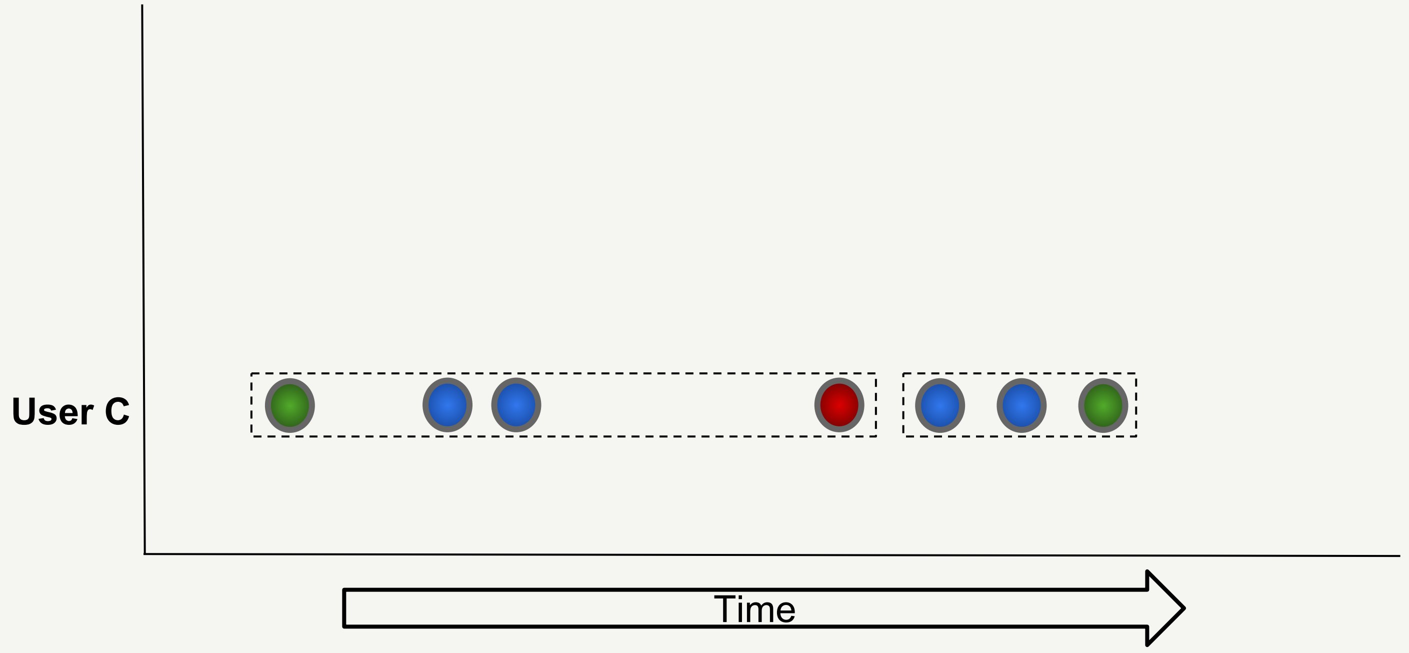


Apache Beam Session Window?

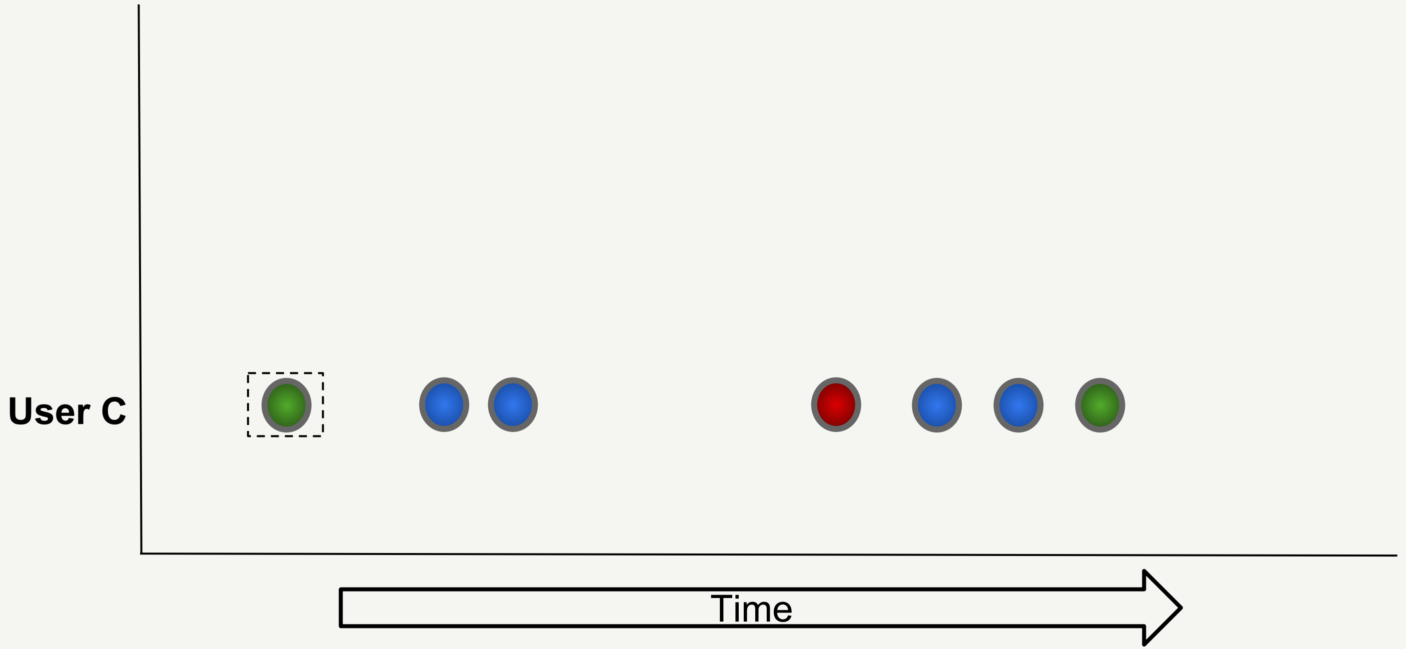


@zimmermatt

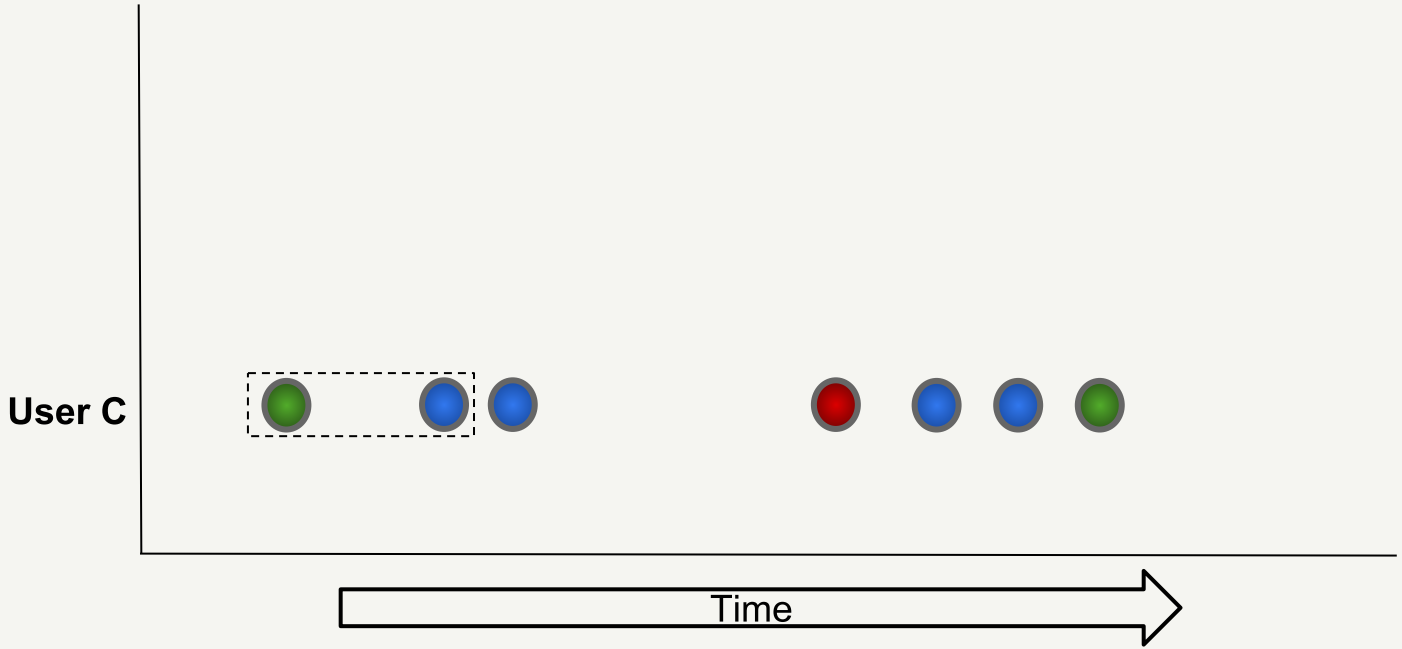
Apache Beam Session Window?



Apache Beam Session Window?

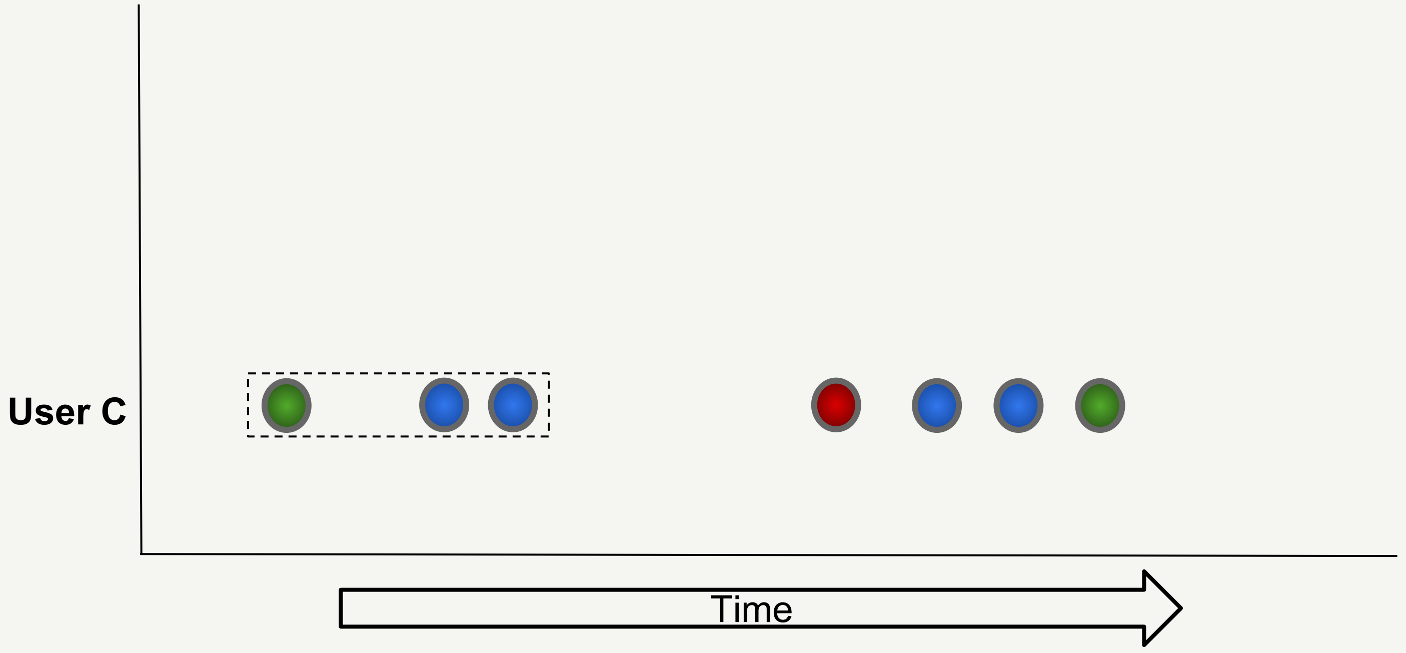


Apache Beam Session Window?

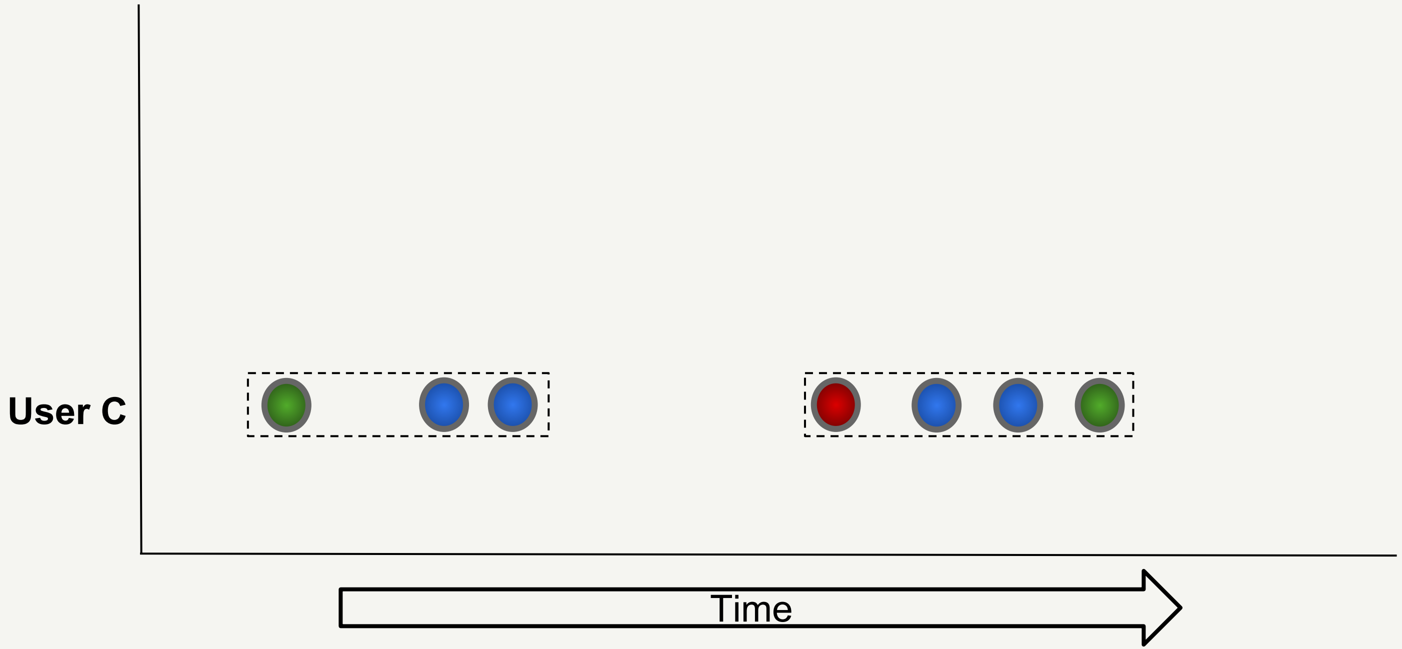


@zimmermatt

Apache Beam Session Window?

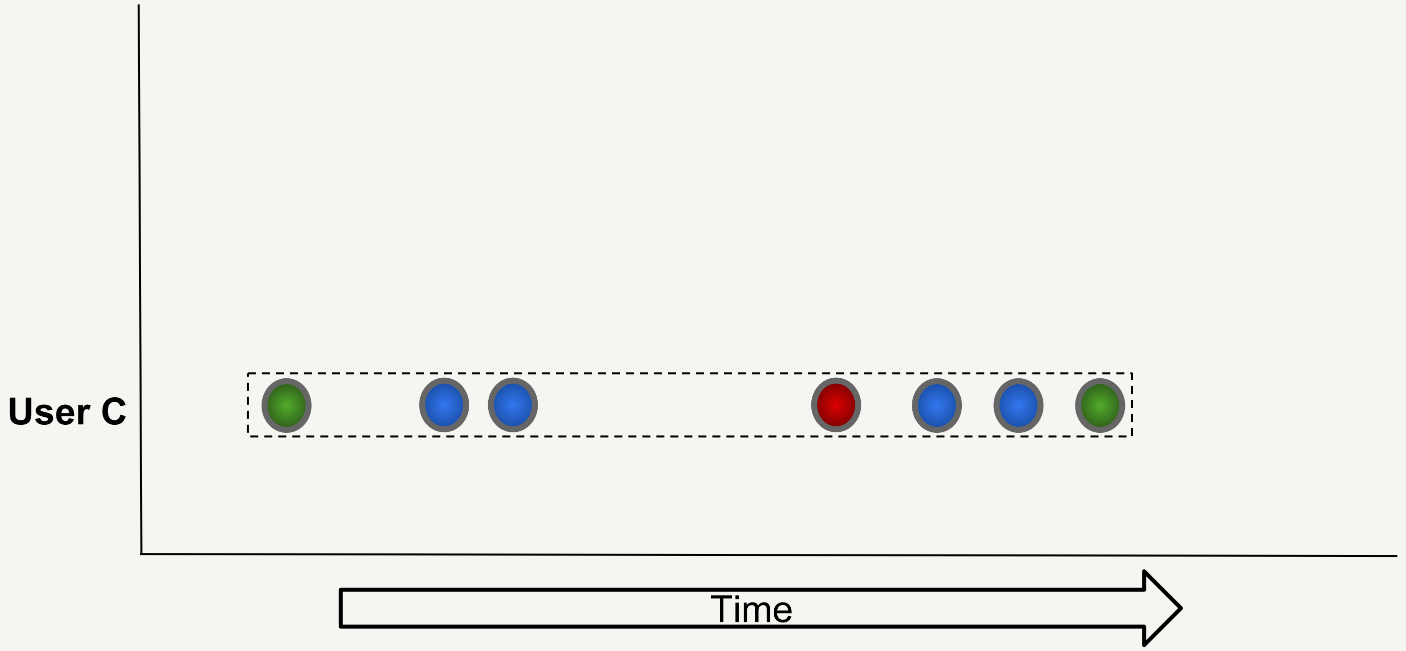


Apache Beam Session Window?



@zimmermatt

Apache Beam Session Window?



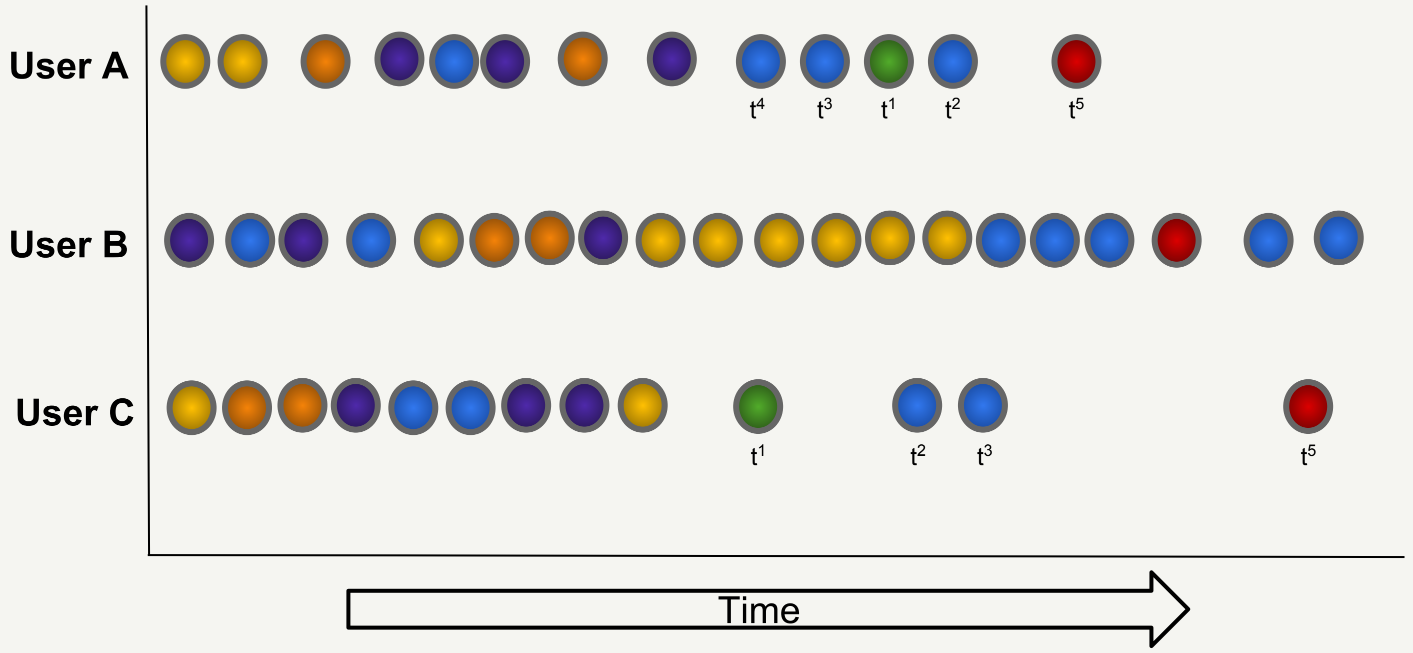
Window Requirements Redux.

- Unaligned windows
- Bounded by event type
- Handle out of order events
- Emit early results
- Capture relevant events; ignore the rest

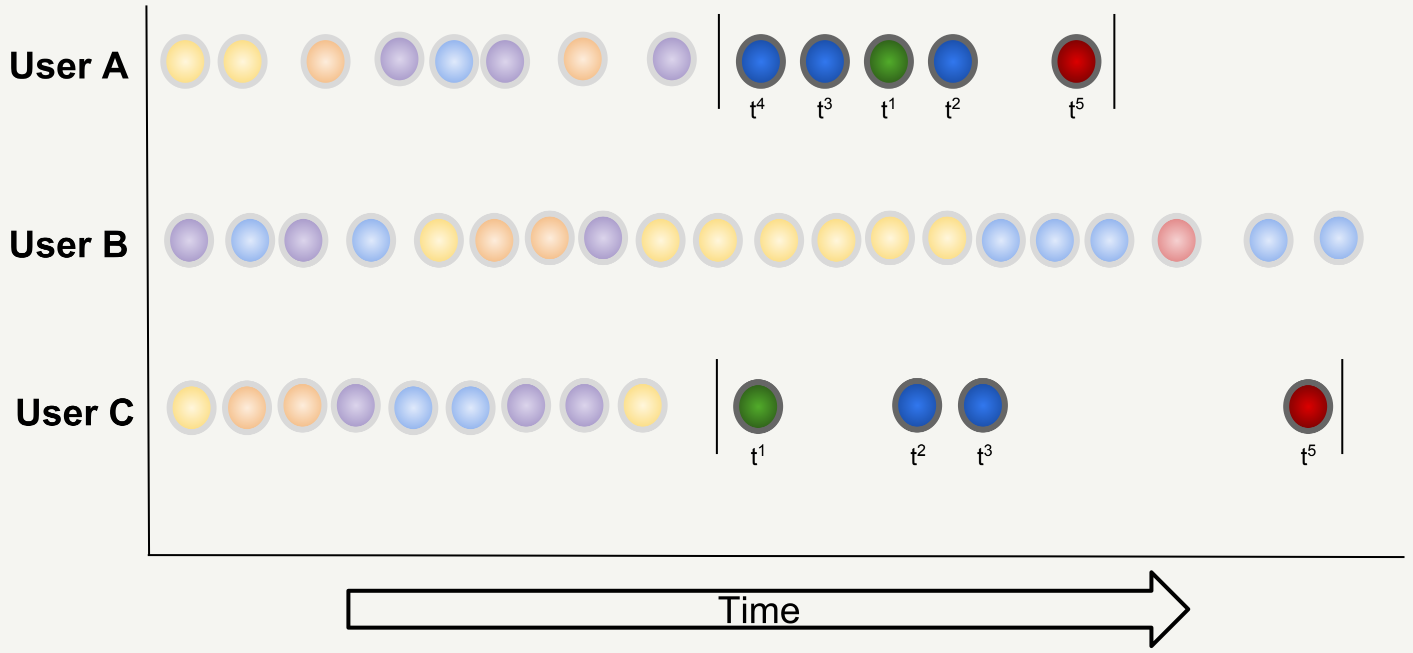
The solution

at 10,000 feet.

The Solution (Conceptual).

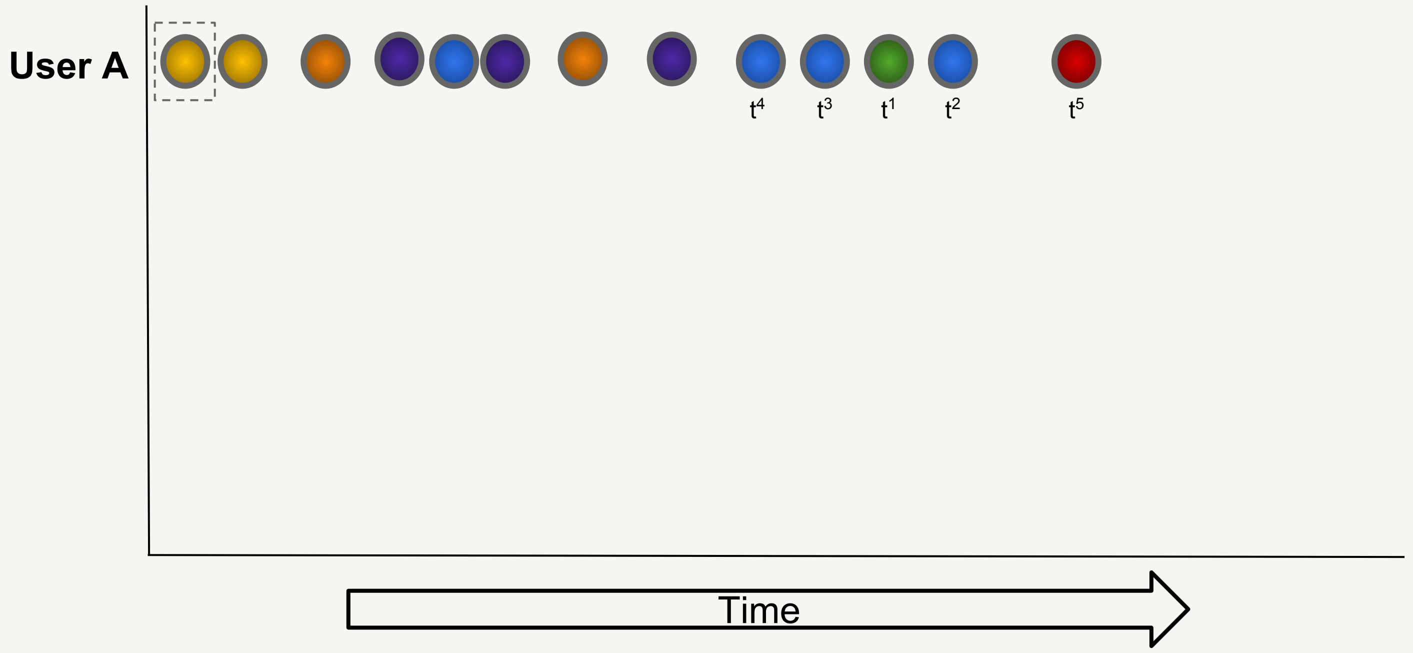


The Solution (Conceptual).

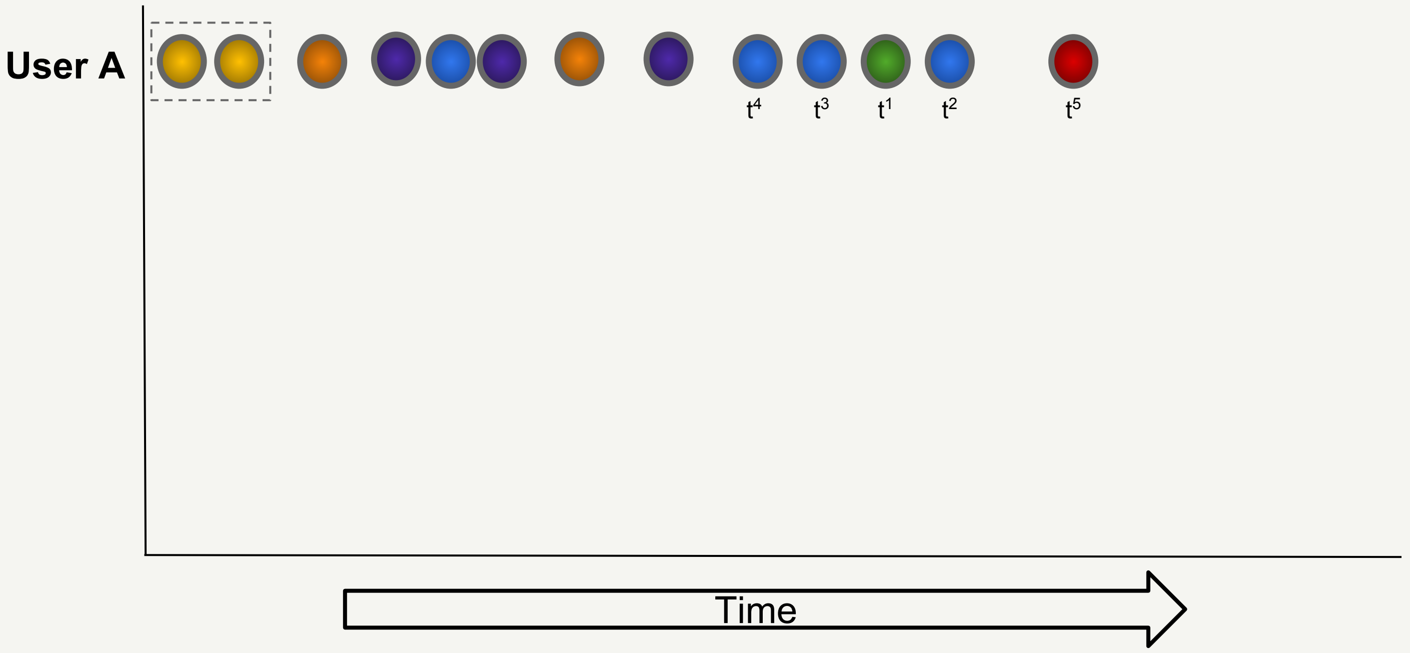


@zimmermatt

The Solution (Conceptual).

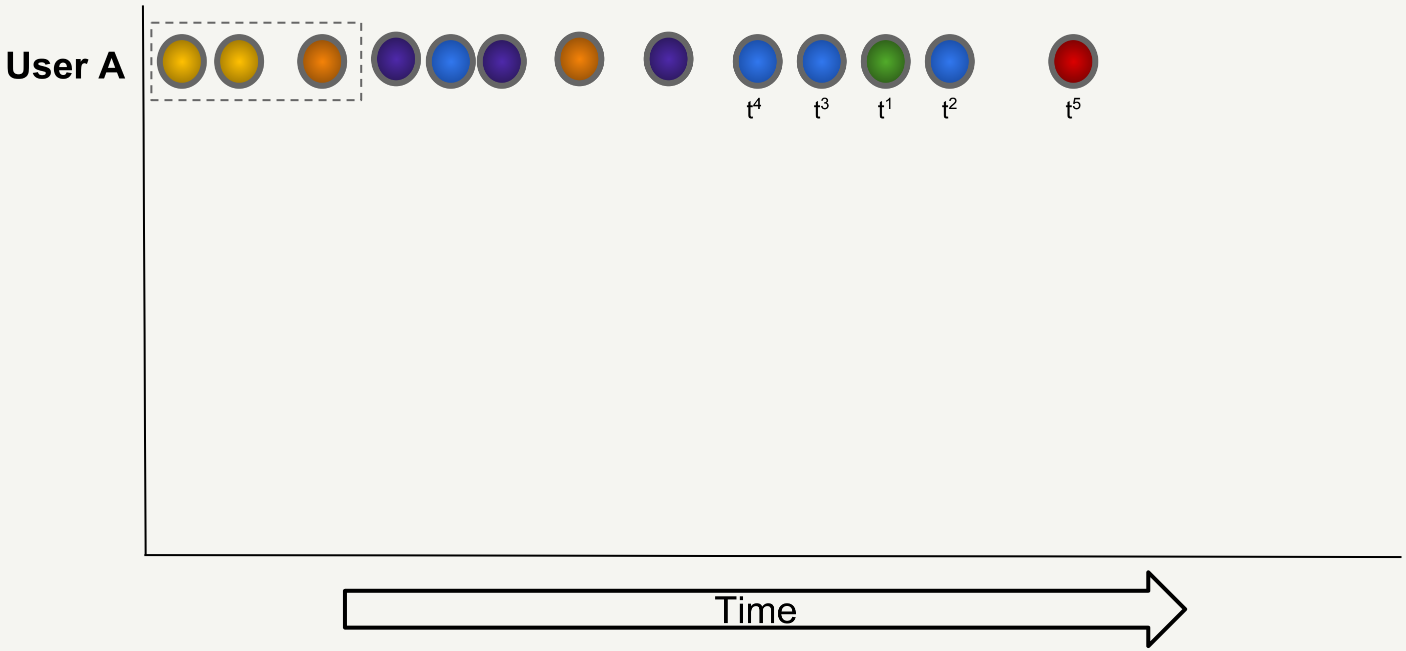


The Solution (Conceptual).

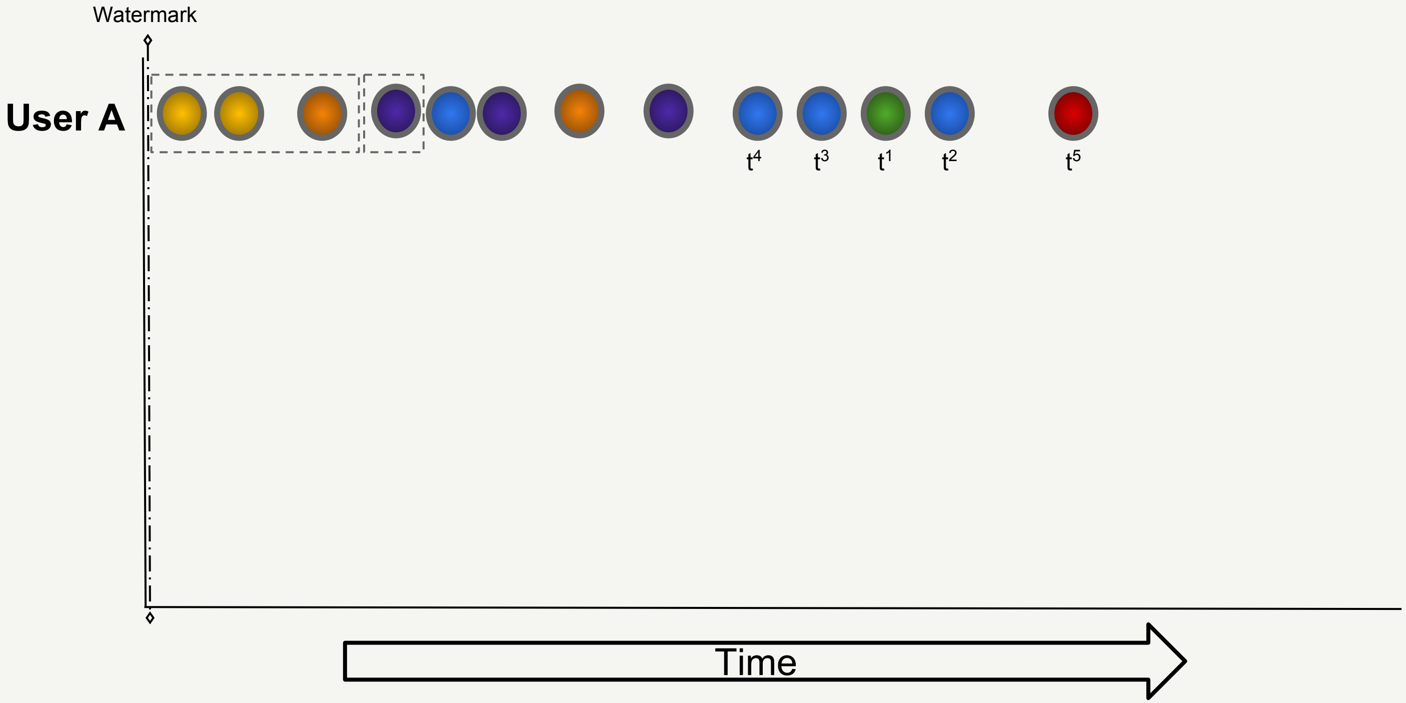


@zimmermatt

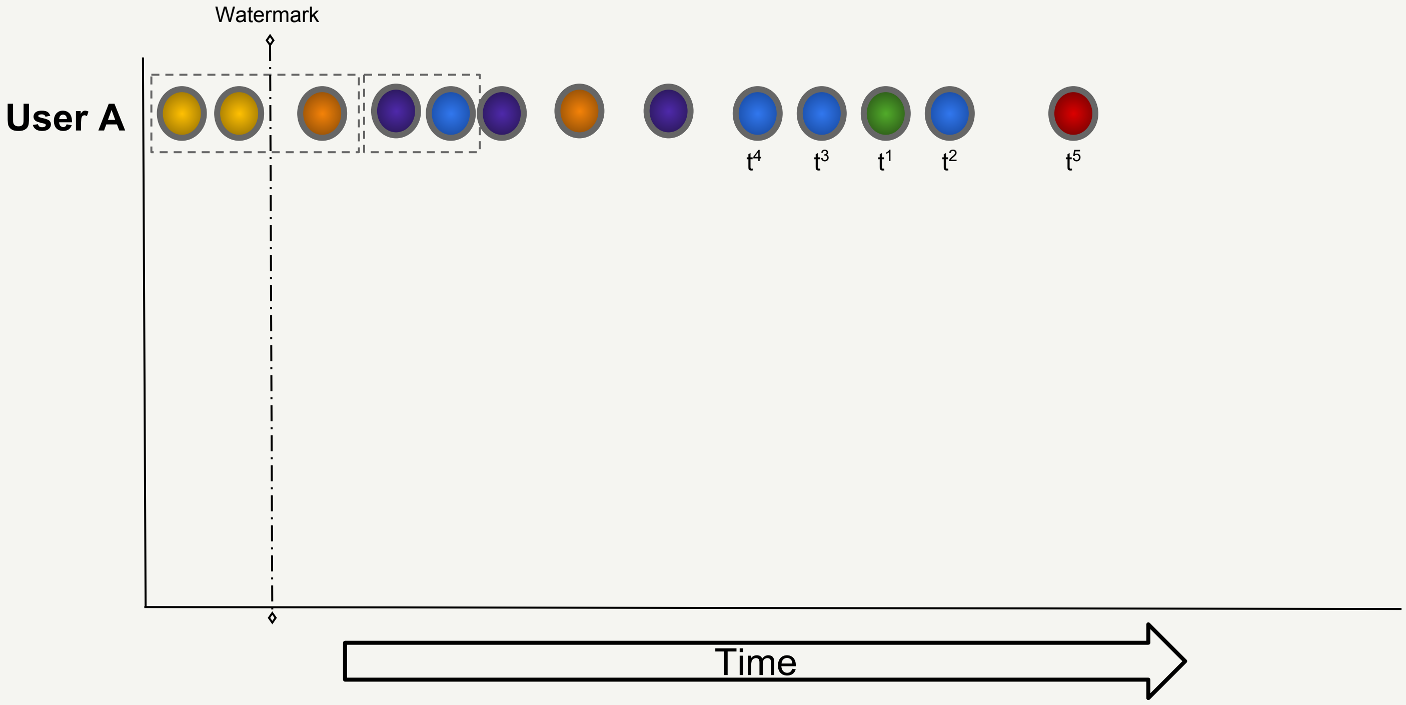
The Solution (Conceptual).



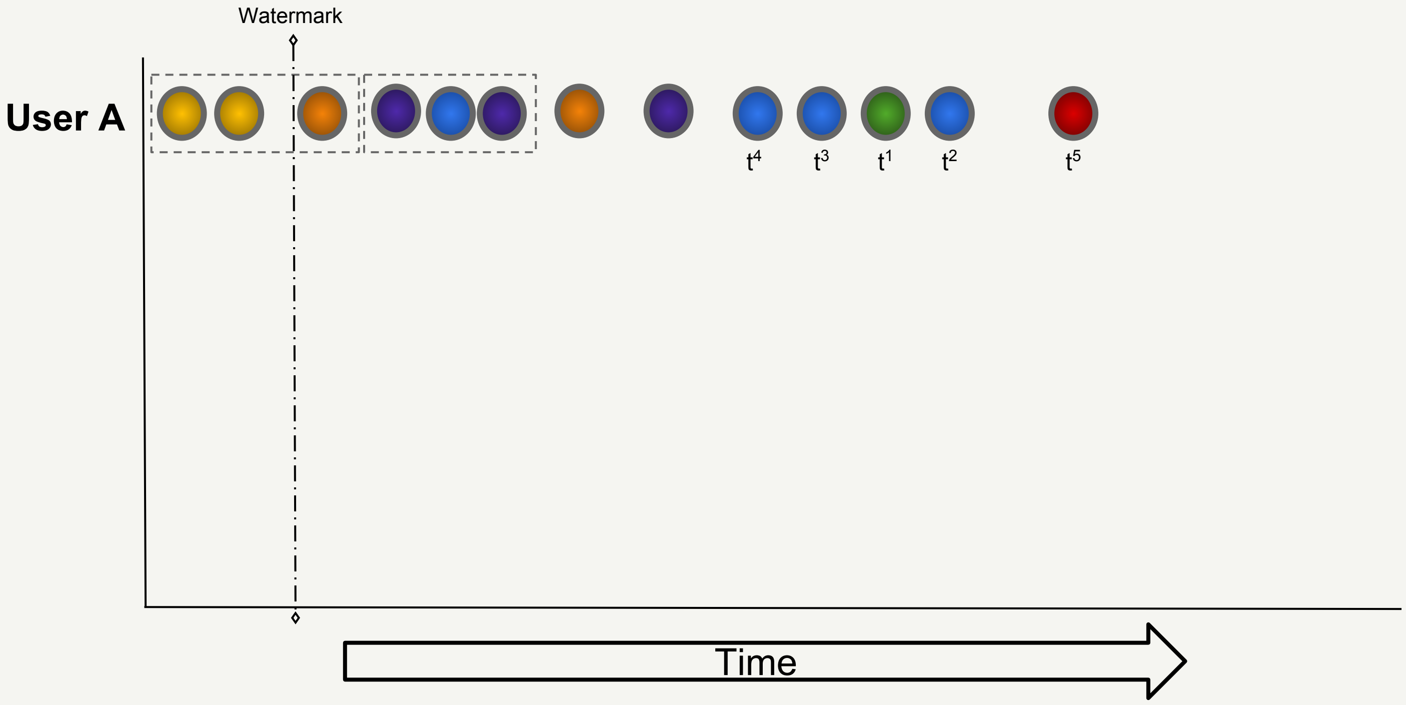
The Solution (Conceptual).



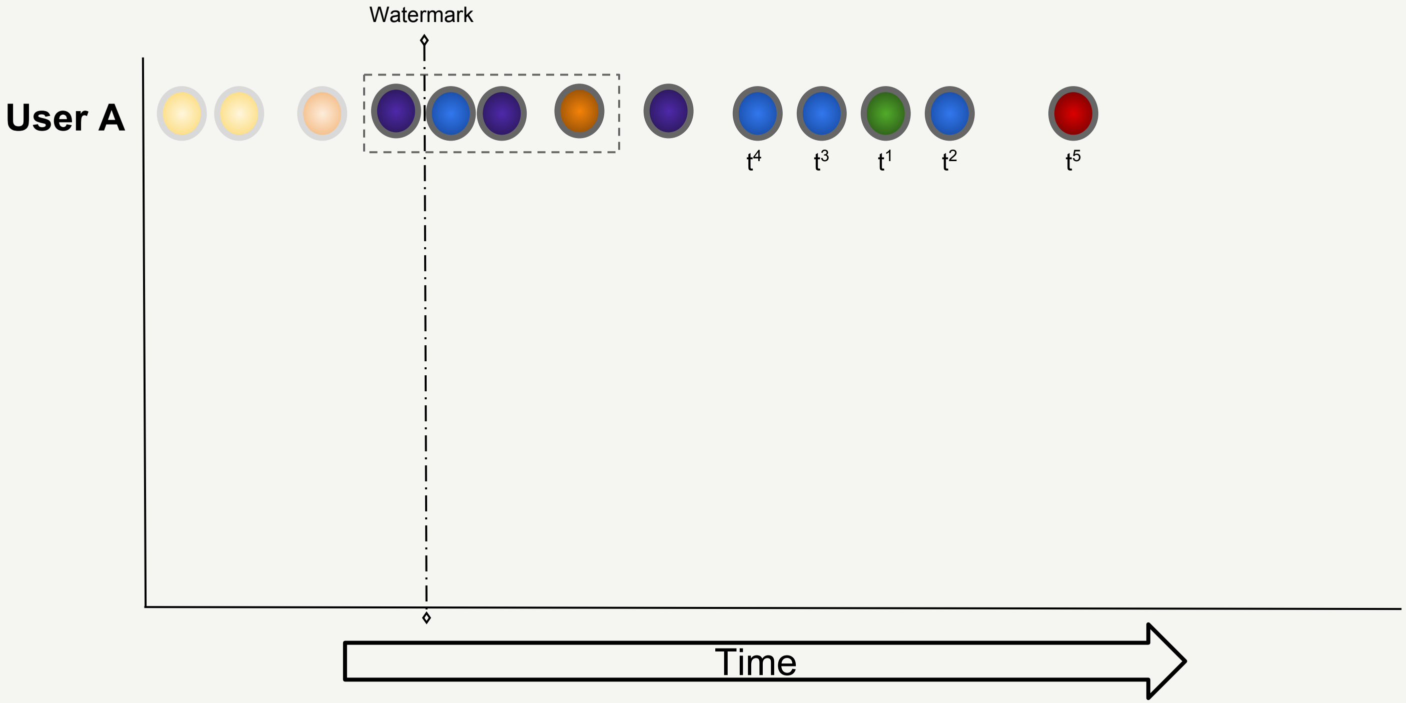
The Solution (Conceptual).



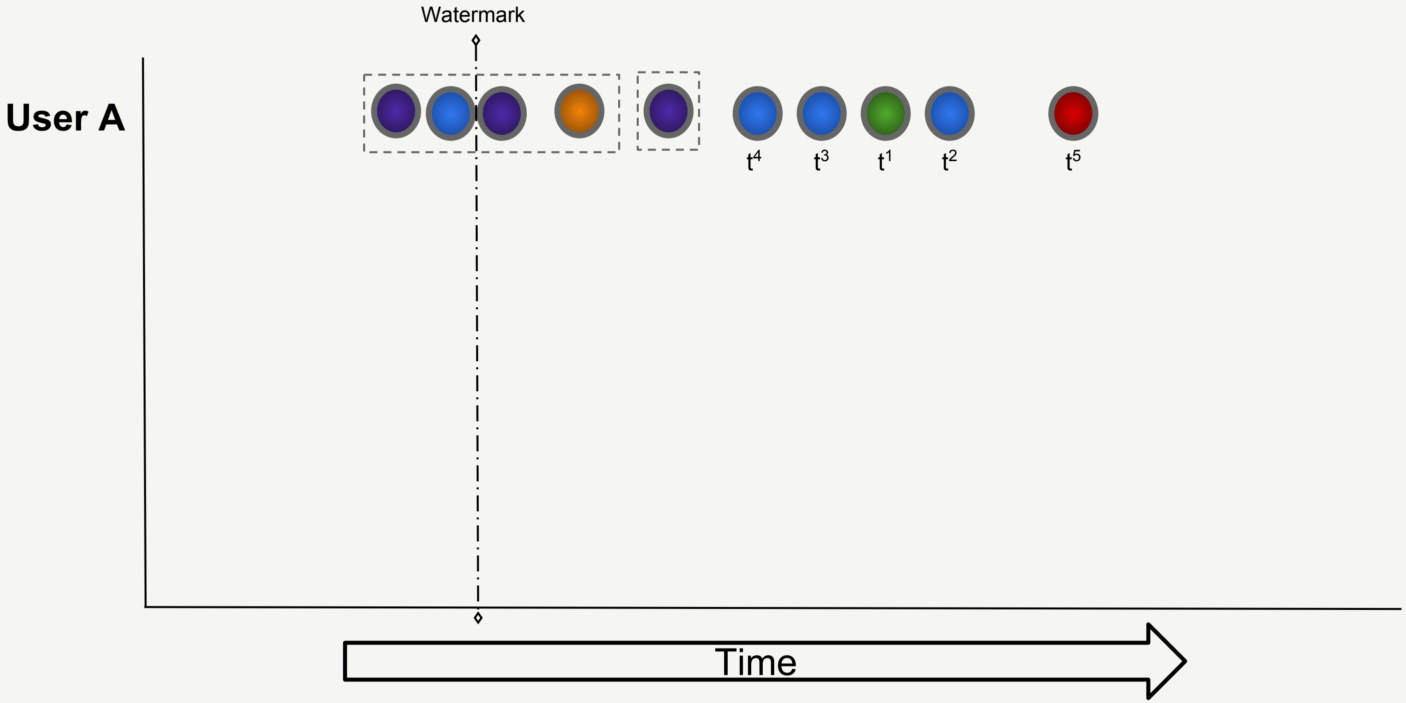
The Solution (Conceptual).



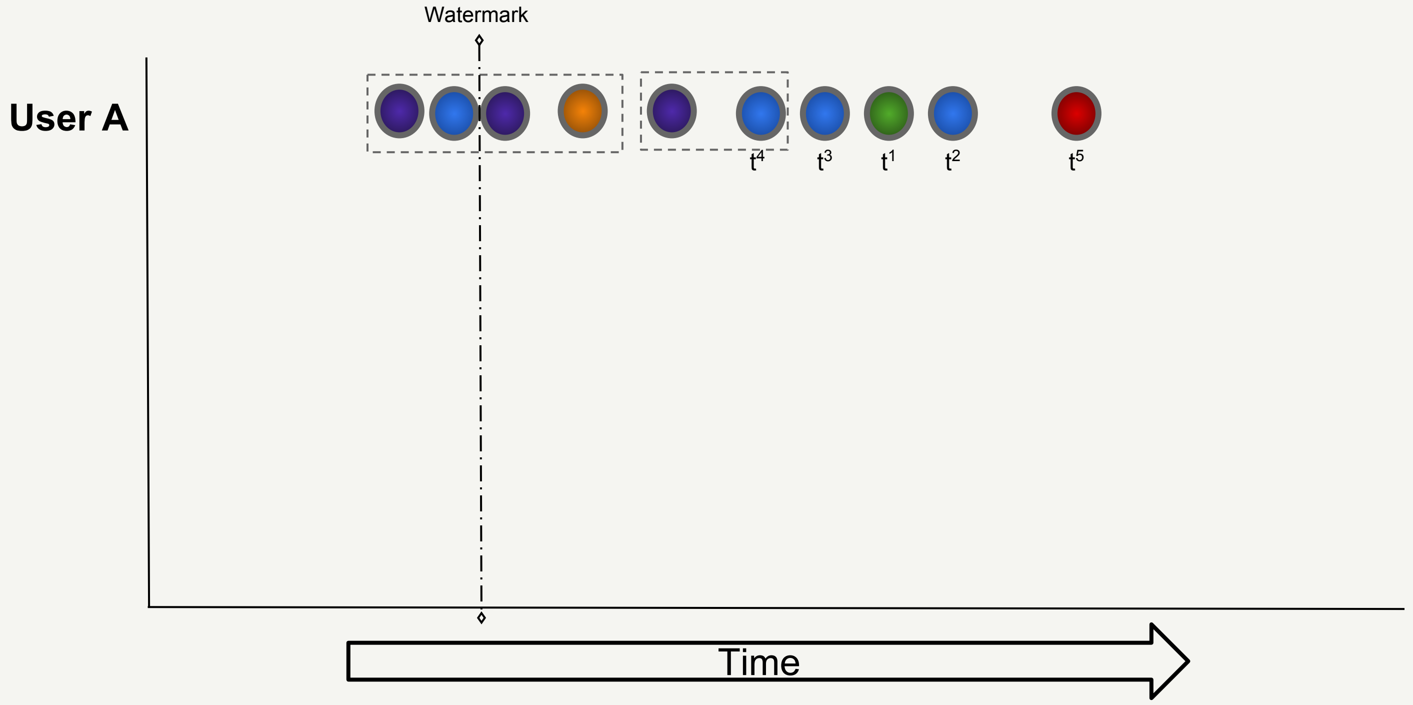
The Solution (Conceptual).



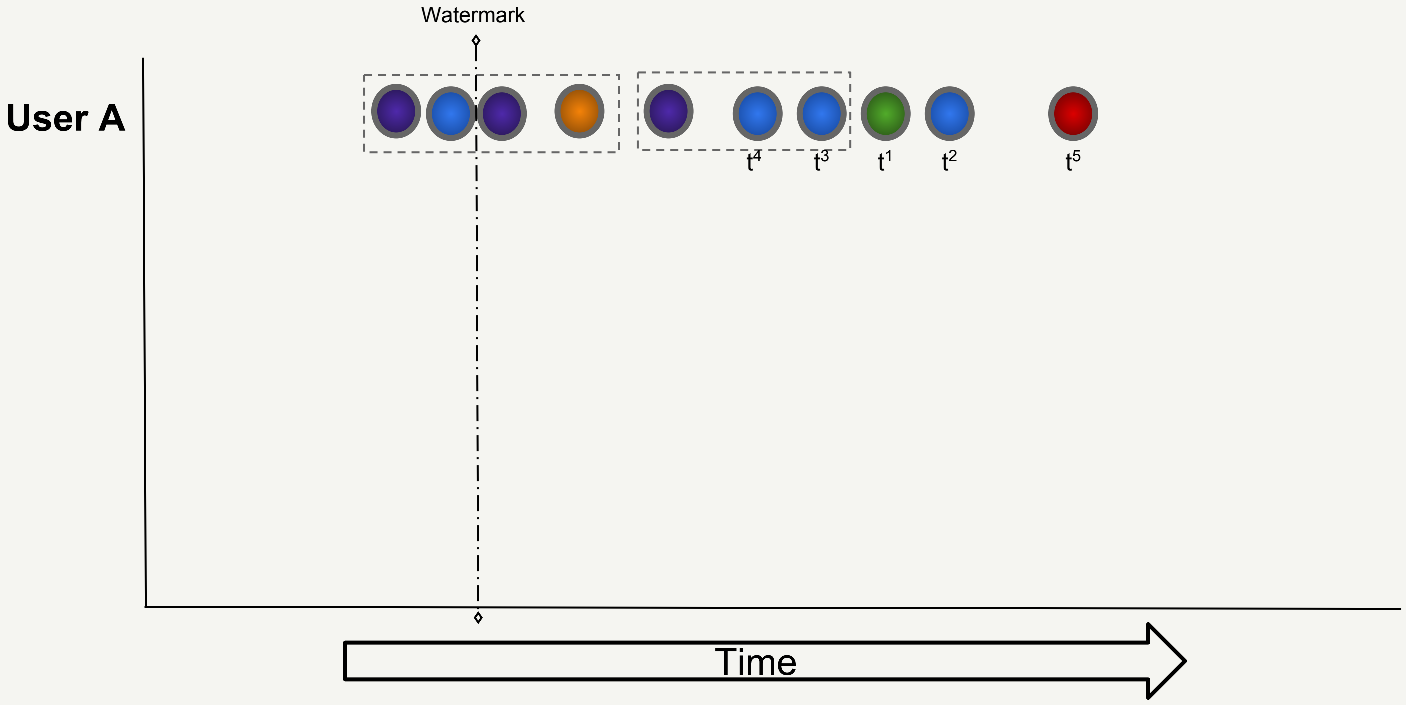
The Solution (Conceptual).



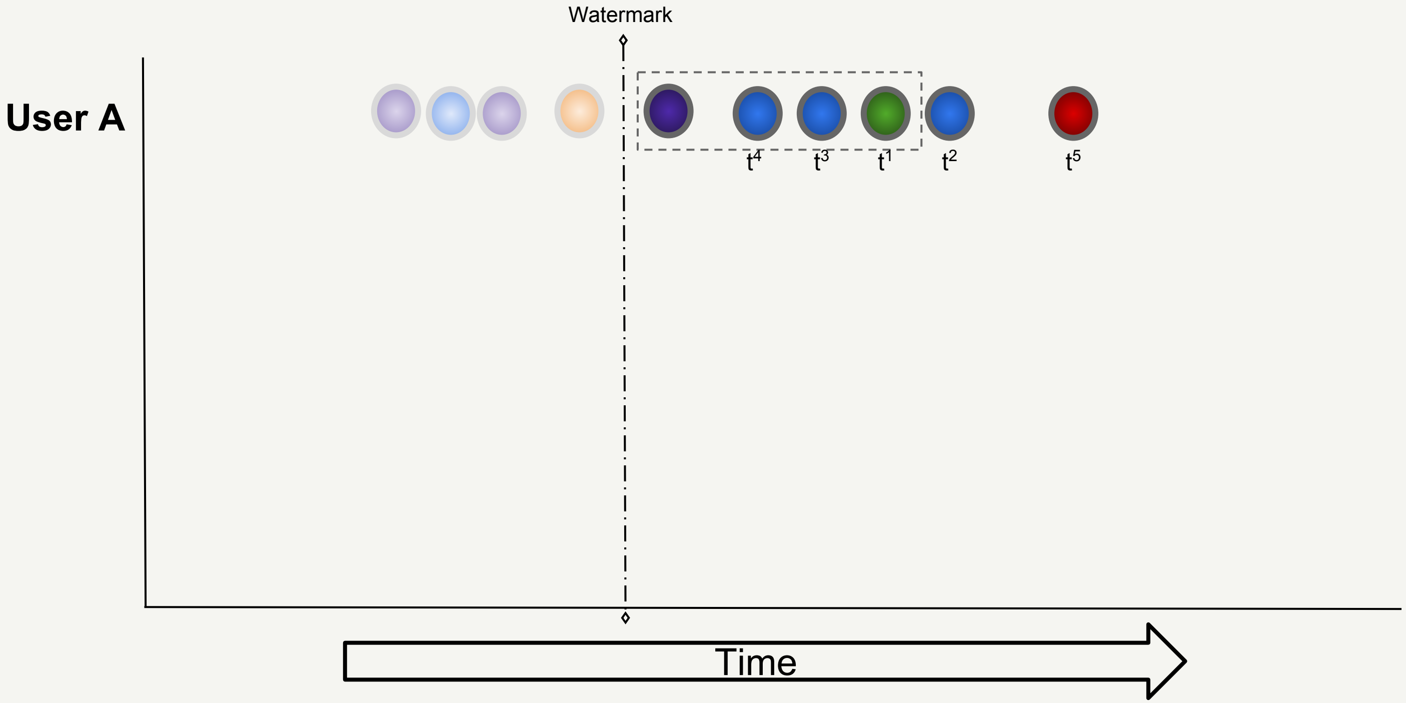
The Solution (Conceptual).



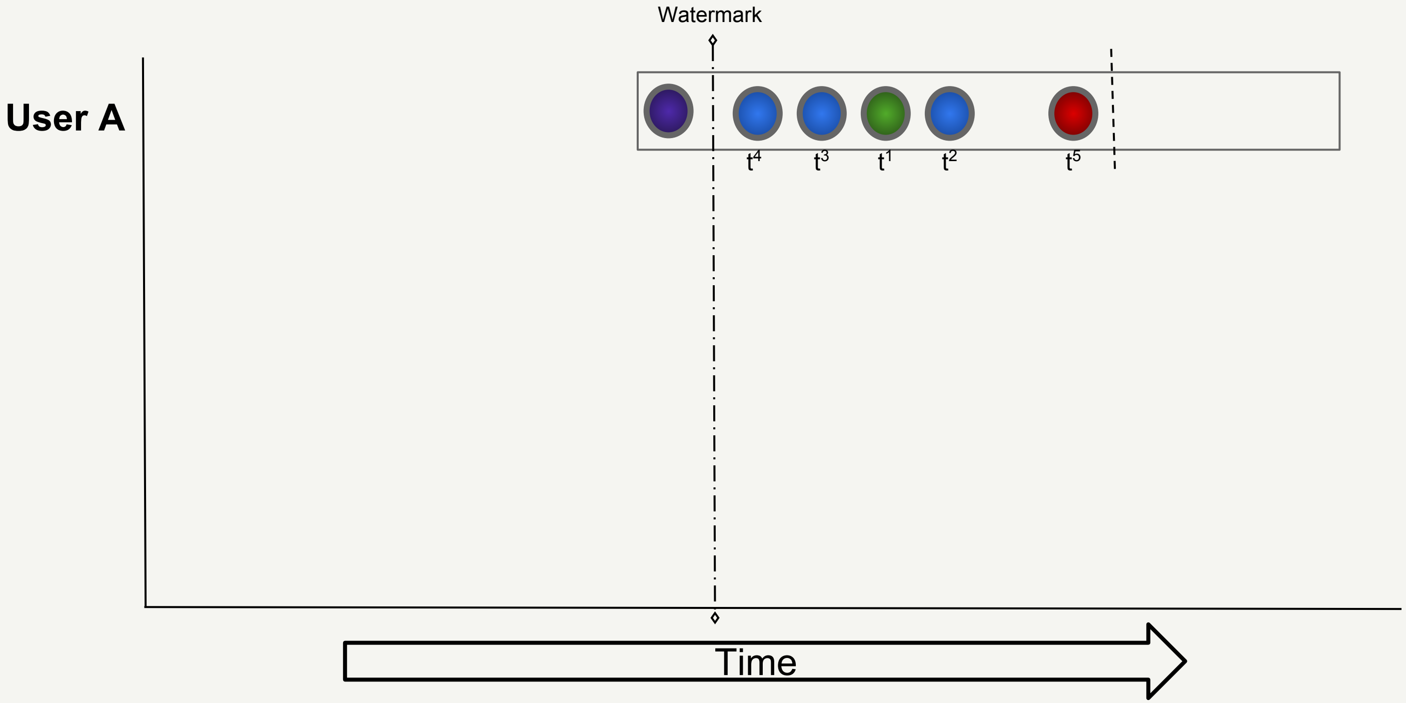
The Solution (Conceptual).



The Solution (Conceptual).



The Solution (Conceptual).



@zimmermatt

Event processing flow.

1. Window assigner.

Event processing flow.

1. Window assigner.
   a. Assign Event to Window(s) (`assignWindows`).

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).
    b. Merge (`onMerge`).

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).
    b. Merge (`onMerge`).
3. Trigger Timers.

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).
    b. Merge (`onMerge`).
3. Trigger Timers.
    a. Processing Time (`onProcessingTime`).

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).
    b. Merge (`onMerge`).
3. Trigger Timers.
    a. Processing Time (`onProcessingTime`).
    b. Event Time (`onEventTime`).

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).
    b. Merge (`onMerge`).
3. Trigger Timers.
    a. Processing Time (`onProcessingTime`).
    b. Event Time (`onEventTime`).
4. Evictor.

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).
    b. Merge (`onMerge`).
3. Trigger Timers.
    a. Processing Time (`onProcessingTime`).
    b. Event Time (`onEventTime`).
4. Evictor.
    a. Before (`evictBefore`).

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).
    b. Merge (`onMerge`).
3. Trigger Timers.
    a. Processing Time (`onProcessingTime`).
    b. Event Time (`onEventTime`).
4. Evictor.
    a. Before (`evictBefore`).
    b. Evaluate Window (`WindowFunction#apply`).

Event processing flow.

1. Window assigner.
    a. Assign Event to Window(s) (`assignWindows`).
    b. Merge Windows (`mergeWindows`).
2. Trigger Handlers.
    a. Element (`onElement`).
    b. Merge (`onMerge`).
3. Trigger Timers.
    a. Processing Time (`onProcessingTime`).
    b. Event Time (`onEventTime`).
4. Evictor.
    a. Before (`evictBefore`).
    b. Evaluate Window (`WindowFunction#apply`).
    c. After (`evictAfter`).

- Motivating Use Cases.
- Window Requirements.
- The Solution (Conceptual).
- Event Processing Flow.
- Apache Flink Window API Walk-Through.
- The Solution in (Detail).
- Pitfalls to Watch Out For.
- Alternative Implementations.
- Questions.

Window API: `Window`.

```
package org.apache.flink.streaming.api.windowing.windows;

public abstract class Window {
      public abstract long maxTimestamp();
}
```

* If you implement Window, you'll need to provide a `TypeSerializer` implementation for it.

Window API: `WindowAssigner`.

```
package org.apache.flink.streaming.api.windowing.assigners;

public abstract class WindowAssigner<T, W extends Window> implements Serializable {
     public abstract Collection<W> assignWindows(T element,
                                                 long timestamp,
                                                 WindowAssignerContext context);
    public abstract Trigger<T, W> getDefaultTrigger(StreamExecutionEnvironment env);
    public abstract TypeSerializer<W> getWindowSerializer(ExecutionConfig executionConfig);
    public abstract boolean isEventTime();
    public abstract static class WindowAssignerContext {
        public abstract long getCurrentProcessingTime();
    }
}
```

Window API: `MergingWindowAssigner`.

```
package org.apache.flink.streaming.api.windowing.assigners;

public abstract class MergingWindowAssigner<T, W extends Window>
  extends WindowAssigner<T, W> {
     public abstract void mergeWindows(Collection<W> windows,
                                       MergeCallback<W> callback);
    public interface MergeCallback<W> {
        void merge(Collection<W> toBeMerged, W mergeResult);
    }
}
```

Window API: Trigger.

```
package org.apache.flink.streaming.api.windowing.triggers;

public abstract class Trigger<T, W extends Window> implements Serializable {
    ...
     public abstract TriggerResult onElement(T element,
                                             long timestamp,
                                             W window,
                                             TriggerContext ctx) throws Exception;

     public boolean canMerge() { return false; }
     public void onMerge(W window,
                         OnMergeContext ctx) throws Exception { throws by default }

    ...
}
```

@zimmermatt

Window API: Trigger.

```
package org.apache.flink.streaming.api.windowing.triggers;

public abstract class Trigger<T, W extends Window> implements Serializable {
    ...
     public abstract TriggerResult onProcessingTime(long time,
                                                    W window,
                                                    TriggerContext ctx) throws Exception;

     public abstract TriggerResult onEventTime(long time,
                                               W window,
                                               TriggerContext ctx) throws Exception;
    ...
}
```

Window API: Trigger.

```
package org.apache.flink.streaming.api.windowing.triggers;

public abstract class Trigger<T, W extends Window> implements Serializable {
    ...
    public abstract void clear(W window, TriggerContext ctx) throws Exception;
    public interface TriggerContext { ... }
    public interface OnMergeContext extends TriggerContext { ... }
    ...
}
```

Window API: Trigger.

```
package org.apache.flink.streaming.api.windowing.triggers;

public abstract class Trigger<T, W extends Window> implements Serializable {
    ...
    public interface TriggerContext {
        long getCurrentProcessingTime();
        MetricGroup getMetricGroup();
        long getCurrentWatermark();
        void registerProcessingTimeTimer(long time);
        void registerEventTimeTimer(long time);
        void deleteProcessingTimeTimer(long time);
        void deleteEventTimeTimer(long time);
        <S extends State> S getPartitionedState(StateDescriptor<S, ?> stateDescriptor);
    }

    public interface OnMergeContext extends TriggerContext {
        <S extends MergingState<?, ?>> void mergePartitionedState(StateDescriptor<S, ?> stateDescriptor);
    }
}
```

Window API: Evictor.

```
package org.apache.flink.streaming.api.windowing.evictors;

public interface Evictor<T, W extends Window> extends Serializable {
    void evictBefore(Iterable<TimestampedValue<T>> elements,
                     int size,
                     W window,
                     EvictorContext evictorContext);
    void evictAfter(Iterable<TimestampedValue<T>> elements,
                    int size,
                    W window,
                    EvictorContext evictorContext);
    ...
}
```

Window API: Evictor.

```
package org.apache.flink.streaming.api.windowing.evictors;

public interface Evictor<T, W extends Window> extends Serializable {
    ...
    interface EvictorContext {
        long getCurrentProcessingTime();
        MetricGroup getMetricGroup();
        long getCurrentWatermark();
    }
}
```

The solution

in detail.

Custom Window: `WindowAssigner`.

```
public class CustomWindowAssigner<E extends CustomEvent> extends MergingWindowAssigner<E, CustomWindow<E>> {
    ...
    @Override
    public Collection<CustomWindow<E>> assignWindows(E element,
                                                       long timestamp,
                                                       WindowAssignerContext context) {
        return Collections.singletonList(new CustomWindow<>(element, timeoutDuration));
    }
    ...
}
```

@zimmermatt

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    @Override
    public long maxTimestamp() {
            return maxTimestamp;
    }
    ...
}
```

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
     @Override
     public boolean equals(Object o) {
         // important: equals implementation must compare using “value” semantics
     }

     @Override
     public int hashCode() {
         // important: same for hashCode implementation
     }
    ...
}
```

@zimmermatt

Custom Window: `Window.`

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent>
                                    extends TypeSerializer<CustomWindow<T>> {
        ...
    }
    ...
}
```

@zimmermatt

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent>
                                 extends TypeSerializer<CustomWindow<T>> {
        @Override
        public boolean isImmutableType() { return true; }
        ...
    }
    ...
}
```

@zimmermatt

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent> extends TypeSerializer<CustomWindow<T>> {
        ...
         @Override
         public TypeSerializer<CustomWindow<T>> duplicate() { return this; }
         @Override
         public CustomWindow<T> createInstance() { return null; }
         @Override
         public CustomWindow<T> copy(CustomWindow<T> from) { return from; }
         @Override
         public CustomWindow<T> copy(CustomWindow<T> from, CustomWindow<T> reuse) { return from; }
         @Override
         public int getLength() { return -1; }
    }
    ...
}
```

@zimmermatt

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent> extends TypeSerializer<CustomWindow<T>> {
        ...
        public void serialize(CustomWindow<T> record, DataOutputView target)
          throws IOException {
            serializeStartEvent(record, target);
            target.writeLong(record.getDuration().toMillis());
            target.writeBoolean(record.evaluate());
            final boolean hasEndEventData = record.getEndEventData() != null;
            target.writeBoolean(hasEndEventData);
            if (hasEndEventData) serializeEndEvent(record, target);
        }
    }
    ...
}
```

@zimmermatt

Custom Window: `Window.`

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent> extends TypeSerializer<CustomWindow<T>> {
        ...
        @Override
        public CustomWindow<T> deserialize(DataInputView source) throws IOException {
            final T startEvent = deserializeStartEvent(source);
            final Duration duration = Duration.ofMillis(source.readLong());
            final boolean evaluate = source.readBoolean();
            final boolean hasEndEventData = source.readBoolean();
            final T endEvent = hasEndEventData ? deserializeEndEvent(source) : null;
            return new CustomWindow<>(startEvent, duration, endEvent, evaluate);
        }
    }
    ...
}
```

@zimmermatt

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent> extends TypeSerializer<CustomWindow<T>> {
        ...
        @Override
        public CustomWindow<T> deserialize(CustomWindow<T> reuse,
                                           DataInputView source) throws IOException {
            return reuse != null ? reuse : deserialize(source);
        }
    }
    ...
}
```

@zimmermatt

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent> extends TypeSerializer<CustomWindow<T>> {
        ...
        @Override
        public void copy(DataInputView source, DataOutputView target) throws IOException {
            // slightly less efficient, but more maintainable
            CustomWindow<T> deserializedWindow = deserialize(source);
            serialize(deserializedWindow, target);
        }
    }
    ...
}
```

@zimmermatt

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent> extends TypeSerializer<CustomWindow<T>> {
        ...
            @Override
            public boolean equals(Object obj) { return obj instanceof Serializer; }
            @Override
            public boolean canEqual(Object obj) { return obj instanceof Serializer; }
            @Override
            public int hashCode() { return 0; }
    }
    ...
}
```

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    ...
    public static class Serializer<T extends CustomEvent> extends TypeSerializer<CustomWindow<T>> {
        ...
        @Override
        public TypeSerializerConfigSnapshot snapshotConfiguration() { ... }
        @Override
        public CompatibilityResult<CustomWindow<T>> ensureCompatibility(
                TypeSerializerConfigSnapshot configSnapshot) {
            return CompatibilityResult.requiresMigration();
        }
        private static class CustomWindowSerializerConfigSnapshot extends TypeSerializerConfigSnapshot {
            ...
        }
    }
    ...
}
```

Custom Window: `Window`.

```
public class CustomWindow<E extends CustomEvent> extends Window {
    …
    public CustomWindow(@Nonnull D primaryEventData,
                        @Nonnull Duration timeoutDuration,
                        D endEventData,
                        boolean evaluate) {
        ...
          this.endTimestamp = endEventData != null ?
                                     endEventData.getTimestamp() : maxTimestamp;
        ...
    }
    ...
     public boolean evaluate() { return evaluate; }
     public Instant startTimestamp() { return primaryEventData.getTimestamp(); }
     public Instant endTimestamp() { return endTimestamp; }
    }
    ...
}
```

@zimmermatt

Custom Window: `WindowAssigner`.

```
public class CustomWindowAssigner<E extends CustomEvent> extends MergingWindowAssigner<E, CustomWindow<E>> {
    ...
    @Override
    public void mergeWindows(Collection<CustomWindow<E>> mergeCandidates,
                             MergeCallback<CustomWindow<E>> mergeCallback) {

        final CustomWindow<E> sessionWindow = calculateSessionWindow(mergeCandidates);

        final Collection<CustomWindow<E>> inWindow = filterWithinWindow(mergeCandidates);

        // MergeCallback#merge implementation expects 2 or more.
        if (inWindow.size() > 1) {
            mergeCallback.merge(inWindow, sessionWindow);
        }
    }
    ...
}
```

@zimmermatt

Custom Window: `WindowAssigner`.

```
public class CustomWindowAssigner<E extends CustomEvent> extends MergingWindowAssigner<E, CustomWindow<E>> {
    ...
    private CustomWindow<E> calculateSessionWindow(Collection<CustomWindow<E>> mergeCandidates) {

        CustomWindow<E> startEventWindow = findStartEventWindow(mergeCandidates);

        if (startEventWindow != null) {  // valid window
            ...
        } else {  // exploratory window
            ...
        }
    }
    ...
}
```

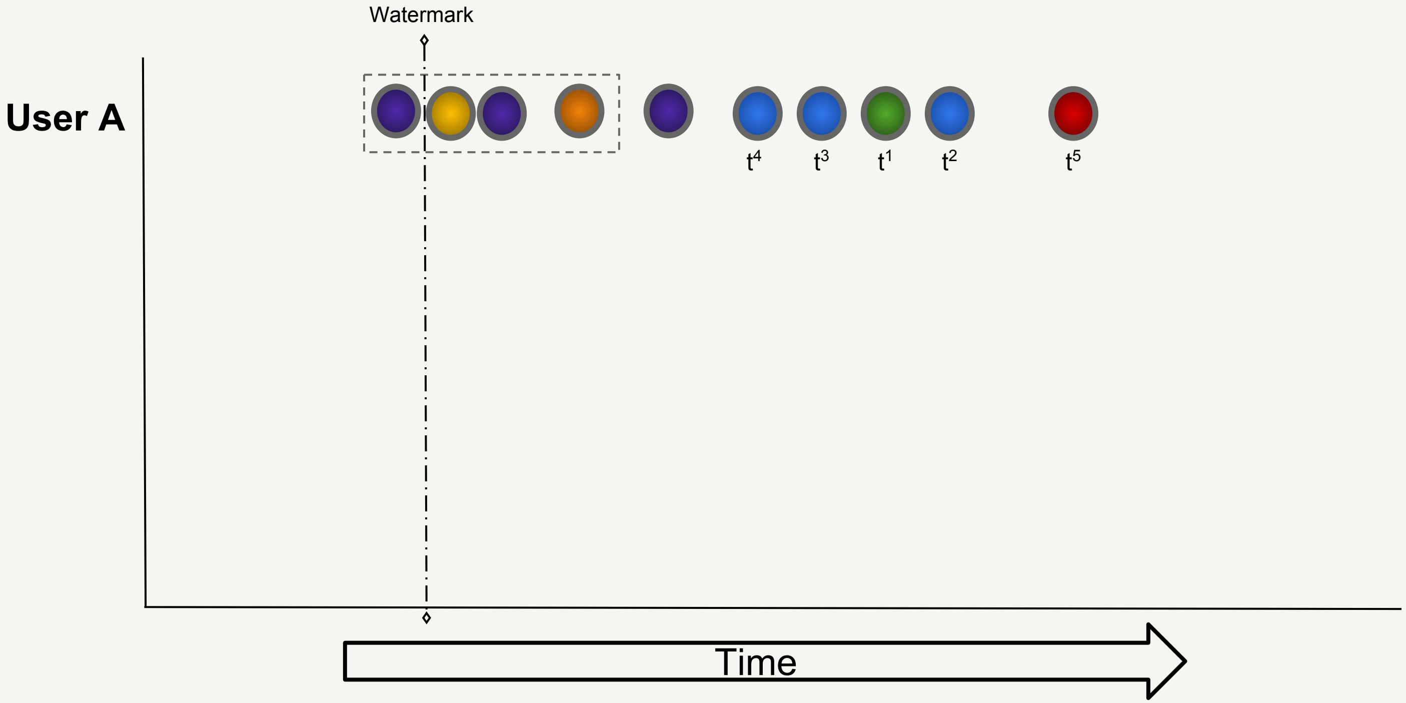
@zimmermatt

Custom Window: `WindowAssigner`.

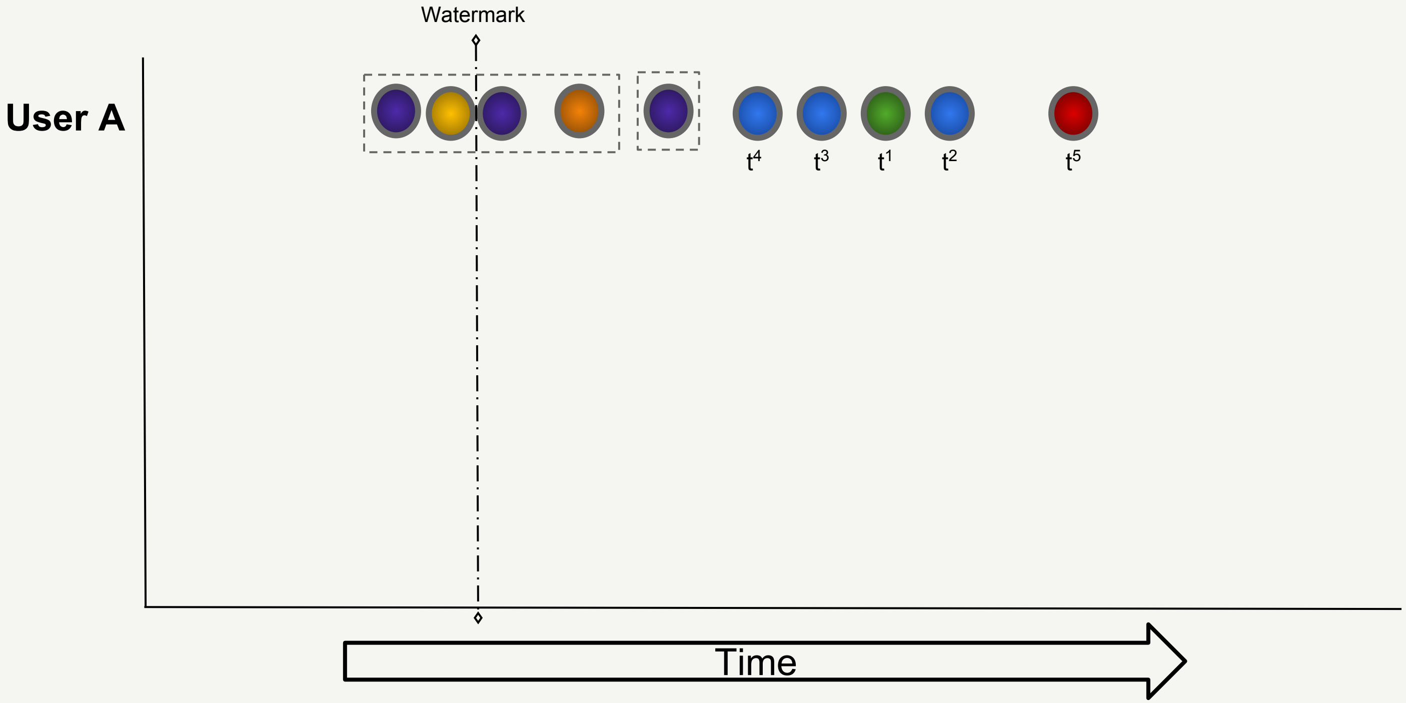
```
if (startEventWindow != null) {  // valid window
    CustomWindow<E> endEvent = findEndEventWindow(mergeCandidates); // can return null

    return new CustomWindow<>(startEventWindow.getEvent, timeoutDuration, endEvent,
                              true); // fire (send this one to the WindowFunction)
} else {  // exploratory window
    ...
}
```

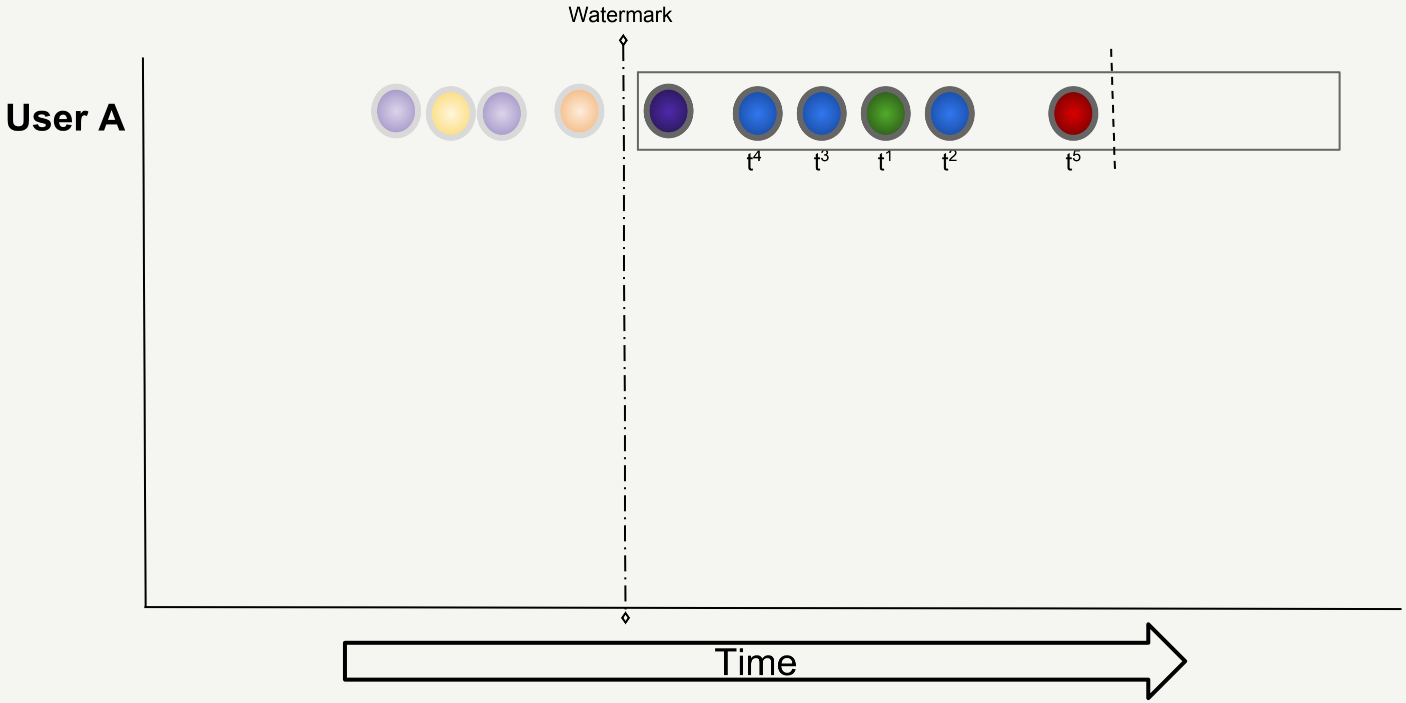
Custom Window: `WindowAssigner`.

```
if (startEventWindow != null) {  // valid window
    ...
} else {  // exploratory window
    CustomWindow<E> window = findClosestToMidpointByStartTime(mergeCandidates);
    return new CustomWindow(window.getEvent, explorat oryDuration,
                                        false) // just purge without firing
}
```



@zimmermatt


Watermark
User A
t^4
t^3
t^1
t^2
t^5
Time



@zimmermatt

Custom Window: Trigger.

```
public class CustomWindowTrigger<E extends CustomEvent> extends Trigger<E, CustomWindow<E>> {
    ...
     @Override
     public boolean canMerge() { return true; }

     @Override
     public void onMerge(CustomWindow<E> window, OnMergeContext onMergeContext)
       throws Exception {
         onMergeContext.registerEventTimeTimer(window.endTimestamp().toEpochMilli());
     }
    ...
}
```

@zimmermatt

Custom Window: Trigger.

```
public class CustomWindowTrigger<E extends CustomEvent> extends Trigger<E, CustomWindow<E>> {
    ...
    @Override
    public TriggerResult onElement(E element, long timestamp, CustomWindow<E> window,
                                   TriggerContext triggerContext) throws Exception {
        final TriggerResult triggerResult;
        final ValueState<Boolean> windowClosedState =
                triggerContext.getPartitionedState(windowClosedDescriptor);
        final long endTimestamp = window.endTimestamp().toEpochMilli();
        if (triggerContext.getCurrentWatermark() >= endTimestamp) {
            triggerResult = windowClosedState.value() ? TriggerResult.CONTINUE
                    : triggerWindow(triggerContext, windowClosedState, window);
        } else {
            ...
        }
        return triggerResult;
    }
    ...
}
```

@zimmermatt

Custom Window: Trigger.

```
public class CustomWindowTrigger<E extends CustomEvent> extends Trigger<E, CustomWindow<E>> {
    ...
    private TriggerResult triggerWindow(TriggerContext triggerContext,
                                        ValueState<Boolean> windowClosedState,
                                        CustomWindow<E> window) throws IOException {
        windowClosedState.update(Boolean.TRUE);
        removeEarlyFiringTimer(triggerContext);
        return window.evaluate() ? TriggerResult.FIRE_AND_PURGE : TriggerResult.PURGE;
    }
    private void removeEarlyFiringTimer(TriggerContext triggerContext) throws IOException {
        final ValueState<Long> earlyFiringState =
                triggerContext.getPartitionedState(earlyFiringDescriptor);
        if (earlyFiringState.value() > 0) {
            triggerContext.deleteProcessingTimeTimer(earlyFiringState.value());
            // set to -1L to differentiate from the default value
            earlyFiringState.update(-1L);
        }
    }
    ...
}
```

Custom Window: `Trigger`.

```
public class CustomWindowTrigger<E extends CustomEvent> extends Trigger<E, CustomWindow<E>> {
    ...
    @Override
    public TriggerResult onElement(E element, long timestamp, CustomWindow<E> window,
                                   TriggerContext triggerContext) throws Exception {
        final TriggerResult triggerResult;
        final long endTimestamp = window.endTimestamp().toEpochMilli();
        final ValueState<Boolean> windowClosedState =
                triggerContext.getPartitionedState(windowClosedDescriptor);
        if ...
        } else {
            windowClosedState.update(Boolean.FALSE);
            triggerResult = TriggerResult.CONTINUE;
            triggerContext.registerEventTimeTimer(endTimestamp);
            registerEarlyFiringTimerIfNecessary(window, triggerContext);
        }
        return triggerResult;
    }
    ...
}
```

Custom Window: Trigger.

```
public class CustomWindowTrigger<E extends CustomEvent> extends Trigger<E, CustomWindow<E>> {
    ...
    private void registerEarlyFiringTimerIfNecessary(CustomWindow<E> window,
                                                     TriggerContext triggerContext)
      throws IOException {
        if (!window.evaluate() || earlyFiringInterval.toMillis() < 1) return;

        final ValueState<Long> earlyFiringState = triggerContext.getPartitionedState(earlyFiringDescriptor);

        if (earlyFiringState.value() == Long.MIN_VALUE) {
            final Long newEarlyFiringTimestamp = System.currentTimeMillis() + earlyFiringInterval.toMillis();
            if (newEarlyFiringTimestamp < window.endTimestamp().toEpochMilli()) {
                triggerContext.registerProcessingTimeTimer(newEarlyFiringTimestamp);
                earlyFiringState.update(newEarlyFiringTimestamp);
            }
        }
    }
    ...
}
```

@zimmermatt

Custom Window: Trigger.

```
private void registerEarlyFiringTimerIfNecessary(CustomWindow<E> window,
                                                 TriggerContext triggerContext)
  throws IOException {
    if (!window.evaluate() || earlyFiringInterval.toMillis() < 1) return;

  final ValueState<Long> earlyFiringState =
          triggerContext.getPartitionedState(earlyFiringDescriptor);

  if (earlyFiringState.value() == Long.MIN_VALUE) {
  final Long newEarlyFiringTimestamp =
          System.currentTimeMillis() + earlyFiringInterval.toMillis();

    if (newEarlyFiringTimestamp < window.endTimestamp().toEpochMilli()) {
        triggerContext.registerProcessingTimeTimer(newEarlyFiringTimestamp);
        earlyFiringState.update(newEarlyFiringTimestamp);
    }
}
```

Custom Window: Trigger.

```
public class CustomWindowTrigger<E extends CustomEvent> extends Trigger<E, CustomWindow<E>> {
    ...
    @Override
    public TriggerResult onEventTime(long time,
                                     CustomWindow<E> window,
                                     TriggerContext triggerContext) throws Exception {
        if (time != window.endTimestamp().toEpochMilli()) {
            return TriggerResult.CONTINUE;
        }

        final ValueState<Boolean> windowClosedState =
                triggerContext.getPartitionedState(windowClosedDescriptor);
        if (windowClosedState.value()) {
            return TriggerResult.CONTINUE;
        }

        return triggerWindow(triggerContext, windowClosedState, window);
    }
    ...
}
```

@zimmermatt

Custom Window: Trigger.

```
public class CustomWindowTrigger<E extends CustomEvent> extends Trigger<E, CustomWindow<E>> {
    ...
    @Override
    public TriggerResult onEventTime(long time,
                                     CustomWindow<E> window,
                                     TriggerContext triggerContext) throws Exception {

        if (time != window.endTimestamp().toEpochMilli()) {
            return TriggerResult.CONTINUE;
        }

        final ValueState<Boolean> windowClosedState =
                triggerContext.getPartitionedState(windowClosedDescriptor);
        if (windowClosedState.value()) {
            return TriggerResult.CONTINUE;
        }

        return triggerWindow(triggerContext, windowClosedState, window);
    }
    ...
}
```

@zimmermatt

Custom Window: Trigger.

```
public class CustomWindowTrigger<E extends CustomEvent> extends Trigger<E, CustomWindow<E>> {
    ...
     @Override
     public TriggerResult onProcessingTime(long time,
                                          CustomWindow<E> window,
                                          TriggerContext triggerContext) throws Exception {
         TriggerResult triggerResult = TriggerResult.CONTINUE;
         if (window.evaluate()) {
             ...
         }
         return triggerResult;
      }
    ...
}
```

@zimmermatt

Custom Window: Trigger.

```
if (window.evaluate()) { // Update early firing
    final ValueState<Long> earlyFiringState =
            triggerContext.getPartitionedState(earlyFiringDescriptor);

    final Long newEarlyFiringTimestamp =
            earlyFiringState.value() + earlyFiringInterval.toMillis();

    if (newEarlyFiringTimestamp < window.endTimestamp().toEpochMilli()) {
        triggerContext.registerProcessingTimeTimer(newEarlyFiringTimestamp);
        earlyFiringState.update(newEarlyFiringTimestamp);
    }

    triggerResult = TriggerResult.FIRE;
}
return triggerResult;
```

Pitfall:

Window `equals` / `hashCode`.

Pitfall:

Metrics and Logs.

Pitfall:

Event Design.

Pitfall:

Large State.

Alternative:

```
ProcessFunction
```

Alternative:

`CEP Library`

- Motivating Use Cases.
- Window Requirements.
- The Solution (Conceptual).
- Event Processing Flow.
- Apache Flink Window API Walk-Through.
- The Solution (Detail).
- Pitfalls to Watch Out For.
- Alternative Implementations.
- Questions.

Thank You!

@zimmermatt