Aggregation details for KIP-450

When a new record comes in, there are two new windows created and an unknown number of existing windows updated. For in order records, one new window will contain the record while the other will not contain the record, shown in **Figure 1.** With in order records, in this example before **c** comes in, the second (right) window will be empty while the first (left) window will need to contain records that have already been processed. Any already existing windows that **b** falls into need to be updated with **b**'s value in their aggregation

Figure 1

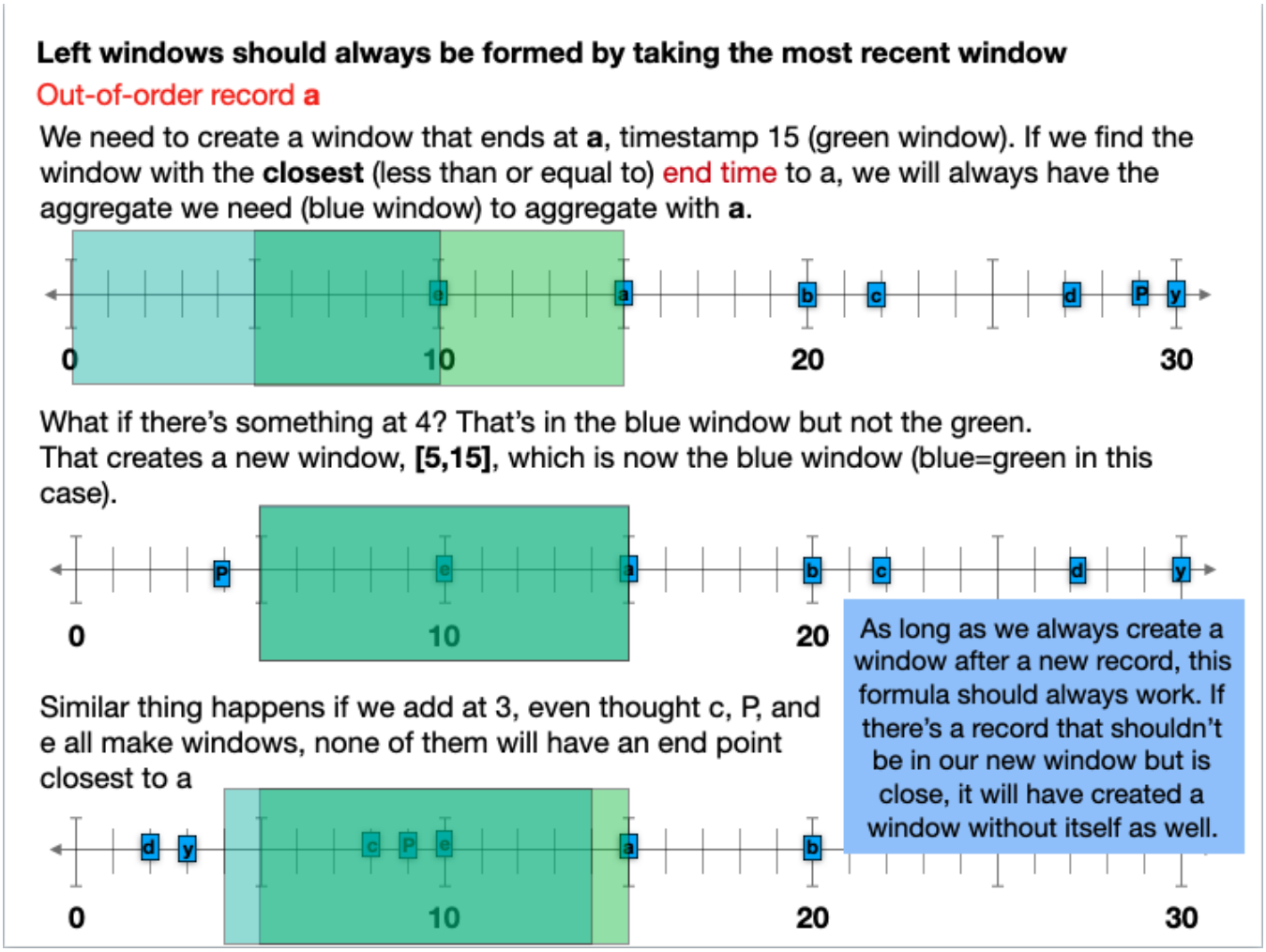
blocked URL

Aggregating for New Windows

A new record for SlidingWindows will always be associated with two windows. If either of those windows already exist in the windows store, their aggregation will simply be updated to include the new record and no duplicate window will be added to the WindowStore. If the window does not exist, the correct aggregation will be calculated and it will be put in the window store. For in-order records, the right window will always be empty and will be created by the first record that comes after the current record and falls within that record's right window. **Note:** in order for this to be possible, we need to know the timestamp of the latest record within a window, and therefore we must use a TimestampedWindowStore.

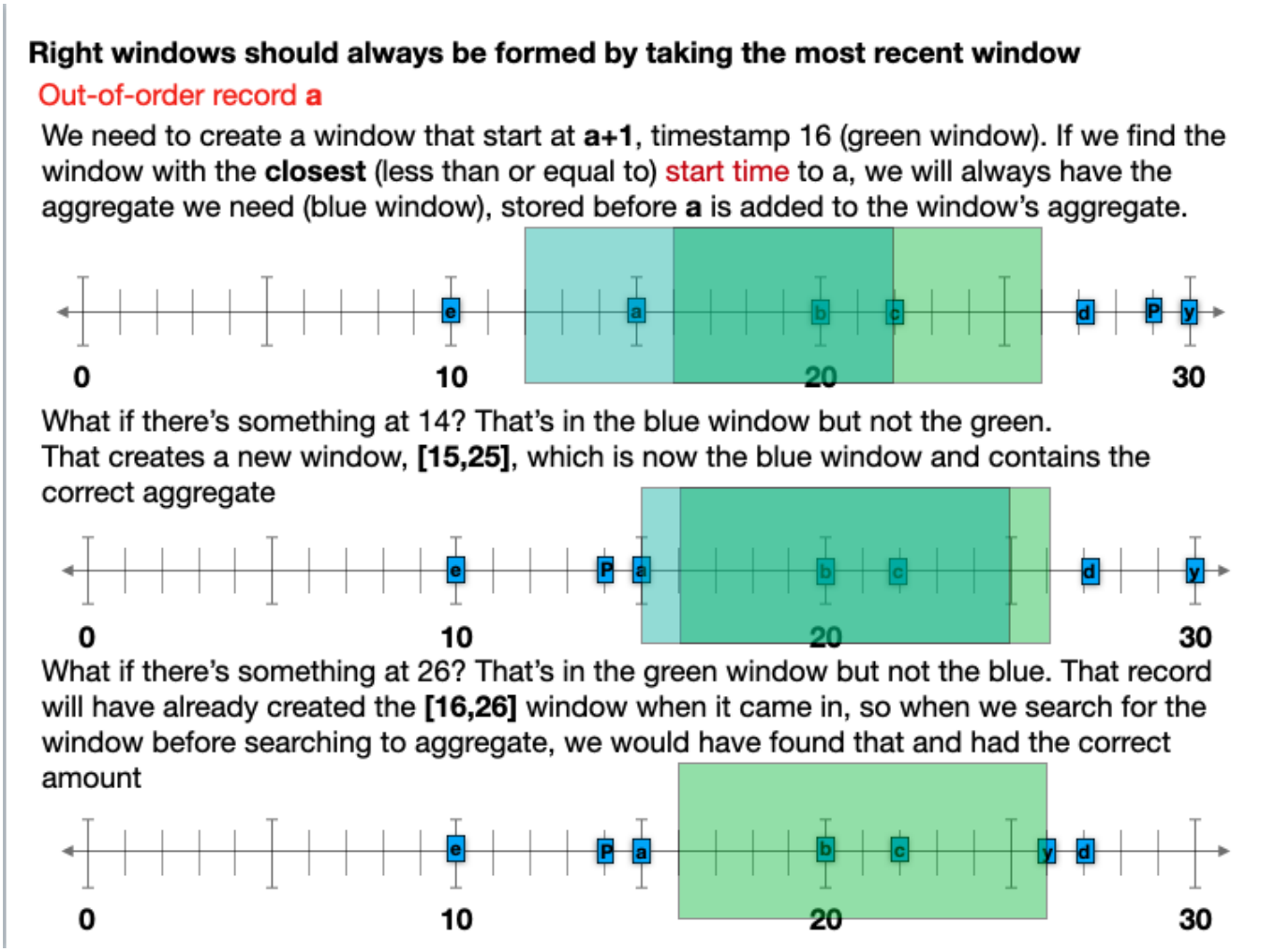
The left window of in-order records and both windows for out-of-order records need to be updated with the values of records that have already been processed, assuming there are existing records that fall within these windows and that these windows do not already exist. Because each record creates or contributes to one window that includes itself and one window that does not, we have the set of all possible windows stored in our WindowStore. Therefore, when we need to find an aggregation to create a new window, that aggregation is already stored somewhere in the WindowStore. **Figure 2** illustrates how we will find the aggregation value for the new left window.

Figure 2



Finding the aggregate for the new right window (for out-of-order records) is similarly predictable, as shown in **figure 3.**

Figure 3.



Aggregating for Existing Windows

When a new record falls within existing windows, a scan is performed in the WindowStore to find windows that have a starting time of <= recordTime - windowSize. These windows will have their aggregations updated with the new record's value and will emit the new result.

Handling Empty Windows

The above theorems work for all cases, unless the right window is meant to be empty, or the left window is meant to only have the current record's aggregate in it.

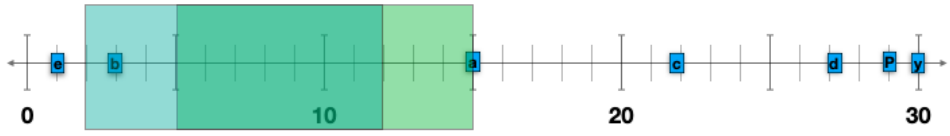
If a left window is meant to be empty aside from the record's value, then the window with the closest end time to our record's timestamp will give us an aggregate that we do not want stored in our new record's left window. To detect whether a left window (for either an in-order or out-of-order record) is supposed to be empty aside from the record's value, we can see if it creates the previous record's right window. If a record either creates the previous record's right window, or falls within the previous record's right window (even if the window is already in the window store) we know the aggregate we found is correct. If it does not, then the left window is meant to be empty aside from the current record's value.

If a new record is *able to create the right window of the previous record then the record's left window is not empty**

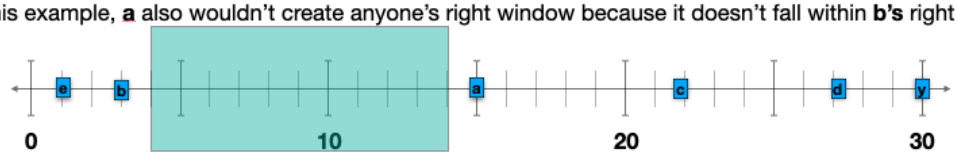
In-order record **a**

We know from earlier examples that the window with the closest end time to a record will hold the aggregate that needs to go in that record's left window. This fails if the left window is supposed to be empty aside from the current record's value.

See record **a. b's** right window isn't created yet (per the new algorithm), so the next closest end time is **e's** right window (blue), created after **b** entered, from **[2,12]**. This holds the aggregate **b**, which is not what we want for **a's** left window (green).



However, in this example, **a** also wouldn't create anyone's right window because it doesn't fall within **b's** right window (blue).



This is in contrast to when **a** does create the previous record's right window (blue) which also indicates that **a's** left window (green) contains an aggregate other than just **a**. This gives us the go ahead to use the aggregate we found and stored earlier

*able means the new record falls within the previous record's right window

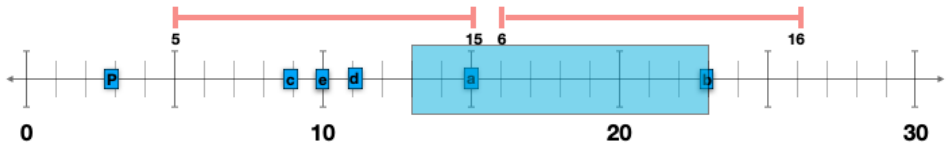
If b was located at 4, a would still fall within its window but it would not fall within a's left window. We can handle this case by checking if a's left window and b's right window are the same

If a right window is meant to be empty, then the window with the closest start time to our record's timestamp will give us an aggregate that we do not want stored in our new record's right window. To detect whether a right window (for an out-of-order record) is supposed to be empty, can check 2 things. If the window with the closest start time to our record is a left window, or if it is a right window but the previous record's right window is already created. If either of these is true, then the aggregate we found is correct. If it is a right window and the previous record's right window is not created, then the aggregate is wrong and the new right window is meant to stay empty.

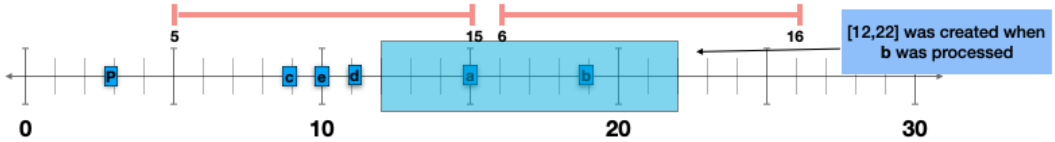
Rules: Out-of-order record **a** (right and left window in pink)

1. If the first window we come upon during our window scan backwards from **a** is a left window, then its aggregate is needed for **a's** right window and **a's** right window is not empty
2. If the first window we come upon is a right window **and** the previous record's right window is already created, then we have the correct aggregate for **a's** right window
3. Else, **a's** right window is empty.

Case 1
First window: [13,23]
Blue window, contains **b**
b's left window
Holds the correct agg



Case 2
First window: [12,22]
Blue window, contains **b**
d's right window
Holds the correct agg



[12,22] was created when b was processed

Case 3
First window: [11,21]
Blue window, contains **d**
e's left window
Holds the wrong agg
D's right window hasn't been created