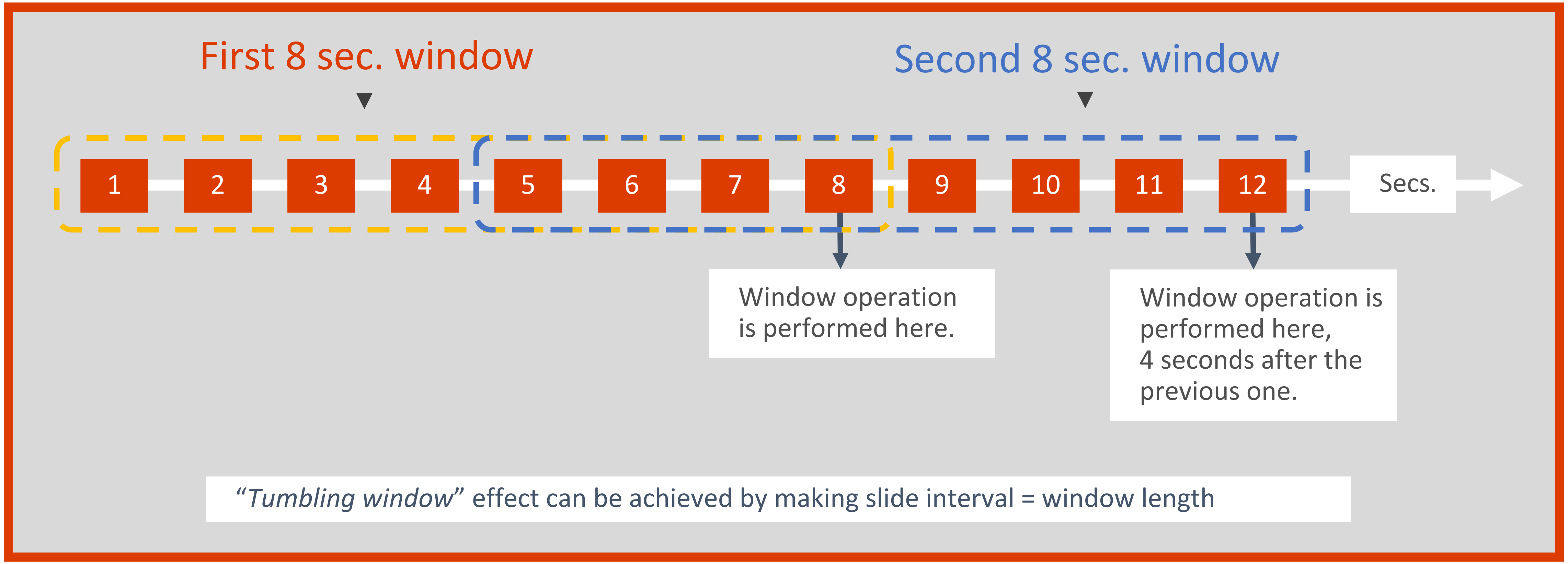
Windowing

Windowing in Spark

Spark provides a set of transformations that apply to a *sliding window* of data.

A window is defined by two parameters: window length and slide interval.



First 8 sec. window
Second 8 sec. window
1
2
3
4
5
6
7
8
9
10
11
12
Secs.
Window operation is performed here.
Window operation is performed here, 4 seconds after the previous one.

"Tumbling window" effect can be achieved by making slide interval = window length

Spark: Window transformation operations

Transformation	Meaning
`Window` *(windowLength, slideInterval)*	Returns a new DStream that is computed based on windowed batches of the source DStream.
`countByWindow` *(windowLength, slideInterval)*	Returns a sliding window count of elements in the stream.
`reduceByWindow` *(func, windowLength, slideInterval)*	Returns a new single-element stream, created by aggregating elements in the stream over a sliding interval using func. The function should be associative so that it can be computed correctly in parallel.
`reduceByKeyAndWindow` *(func, windowLength, slideInterval, [numTasks])*	When called on, a DStream of (K, V) pairs returns a new DStream of (K, V) pairs, where the values for each key are aggregated using the given reduce function (func) over batches in a sliding window.
`reduceByKeyAndWindow` *(func, invFunc, windowLength, slideInterval, [numTasks])*	A more efficient version of the above reduceByKeyAndWindow(), where the reduce value of each window is calculated incrementally, using the reduce values of the previous window. This is done by reducing the new data that enters the sliding window, and "inverse reducing" the old data that leaves the window. An example would be that of "adding" and "subtracting" counts of keys as the window slides. However, it is applicable to only "invertible reduce functions"—that is, those reduce functions that have a corresponding "inverse reduce" function (taken as parameter invFunc).
`countByValueAndWindow` *(windowLength, slideInterval, [numTasks])*	When called on, a DStream of (K, V) pairs returns a new DStream of (K, Long) pairs where the value of each key is its frequency within a sliding window.

Windowing sample code

Count hashtags in a Twitter stream over the last 10 mins., and update every 2 secs.

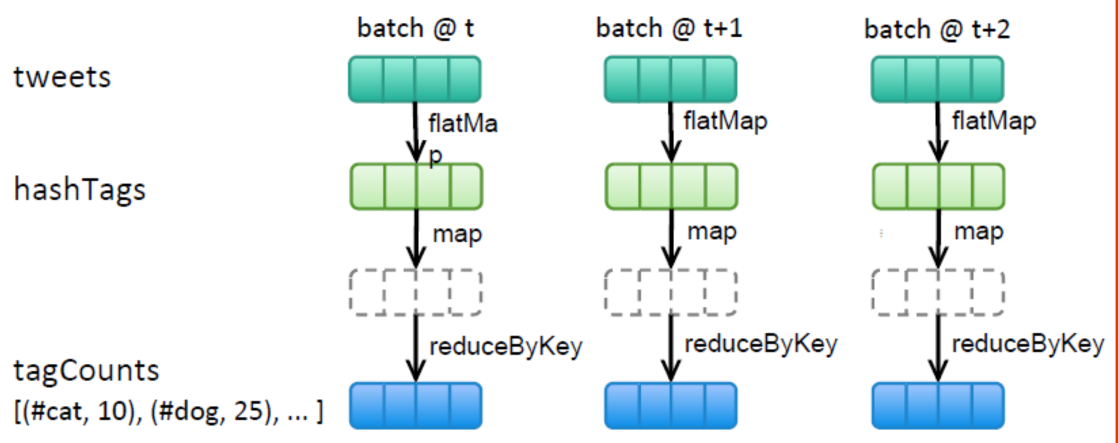
```
val tweets = ssc.twitterStream<Twitter username>, <Twitter password>)

val hashTags = tweets.flatMap (status => getTags(status))

val tagCounts = hashTags.window(Minutes(10), Second(2)) countByValue()
```

Window operation | Window length | Sliding interval | Callback operation

countByValue() is invoked every 2 secs. All RDDs accumulated in the previous 10 minutes are passed to the callback function.


batch @ t
batch @ t+1
batch @ t+2
tweets
flatMa p
flatMap
flatMap
hashTags
map
map
map
reduceByKey
reduceByKey
reduceByKey
tagCounts
[(#cat, 10), (#dog, 25), ...]