CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Big Data Processing

L20-21: Spark Structured Streaming

Dr. Ignacio Castineiras
Department of Computer Science

Outline

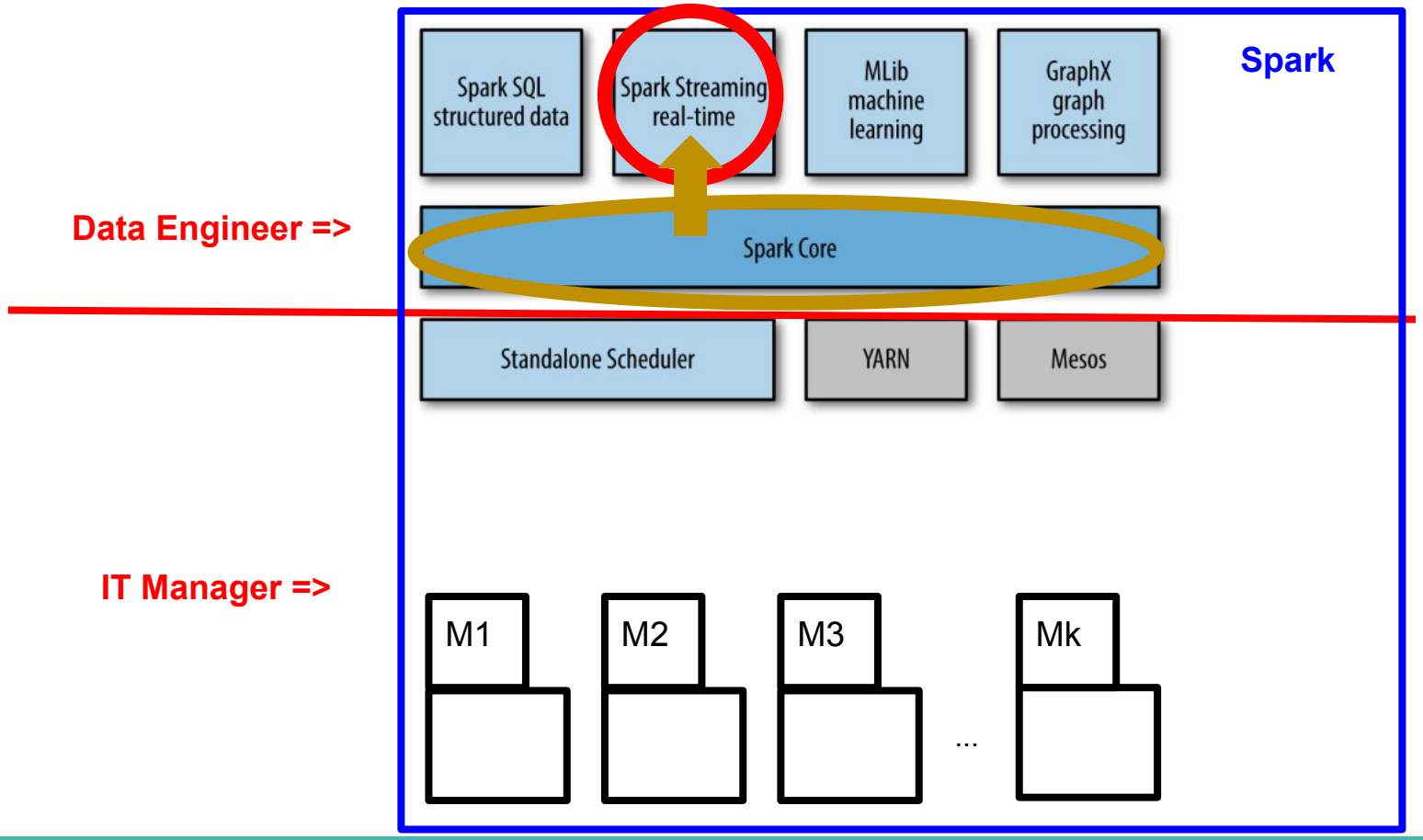
1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.

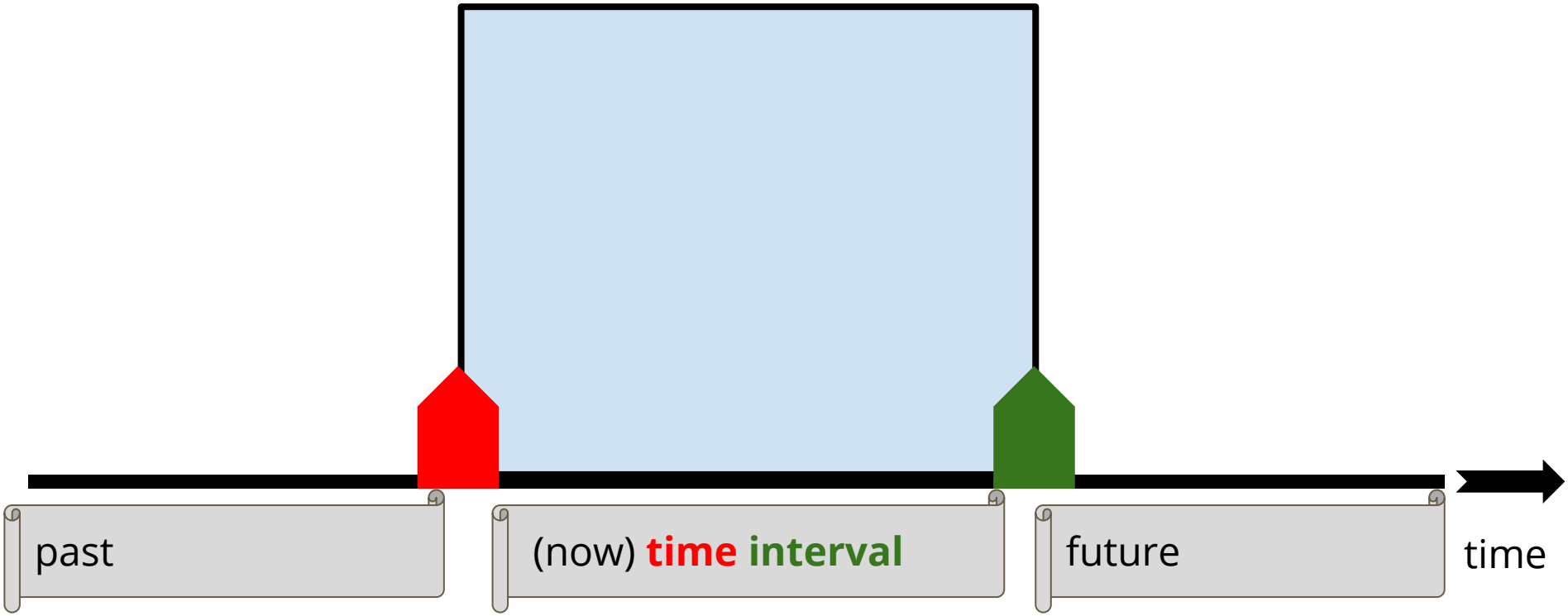
Setting Up the Context

1. We have seen that Spark Core functionality can be extended to work in a Streaming fashion.



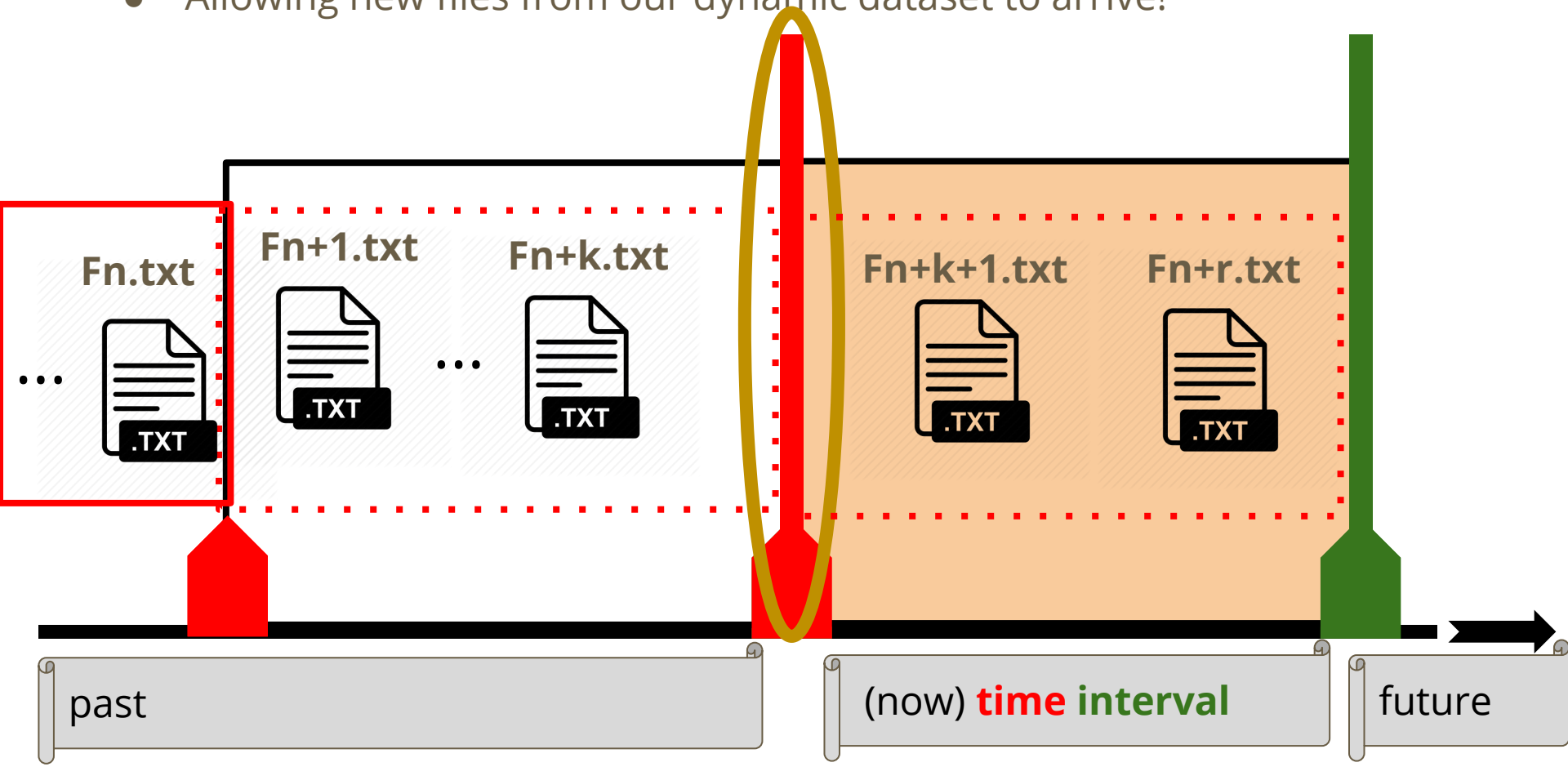
Setting Up the Context

2. For it we need the notion of **time interval**...



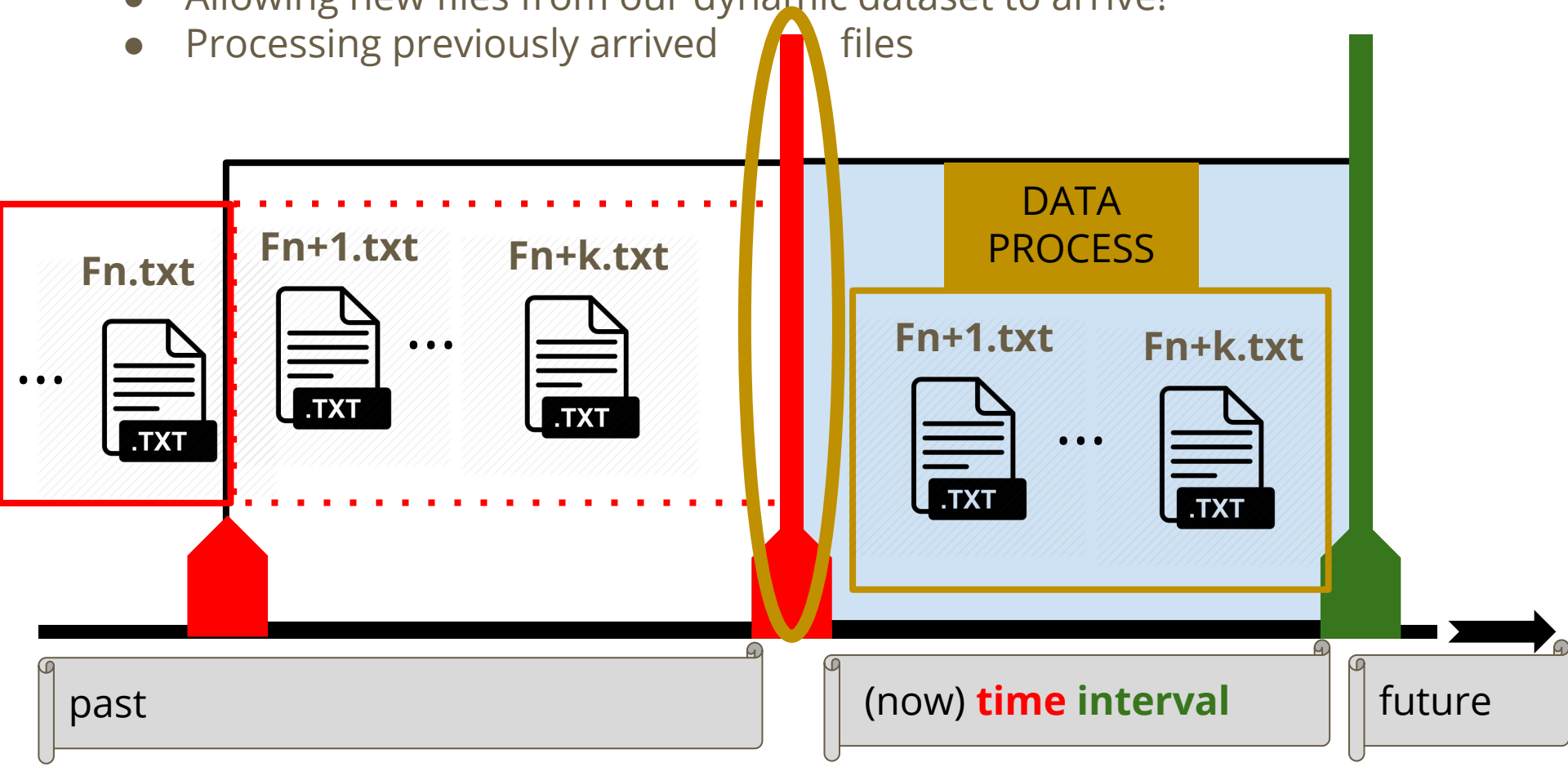
Setting Up the Context

2. For it we need the notion of **time interval**...
   - Allowing new files from our dynamic dataset to arrive!



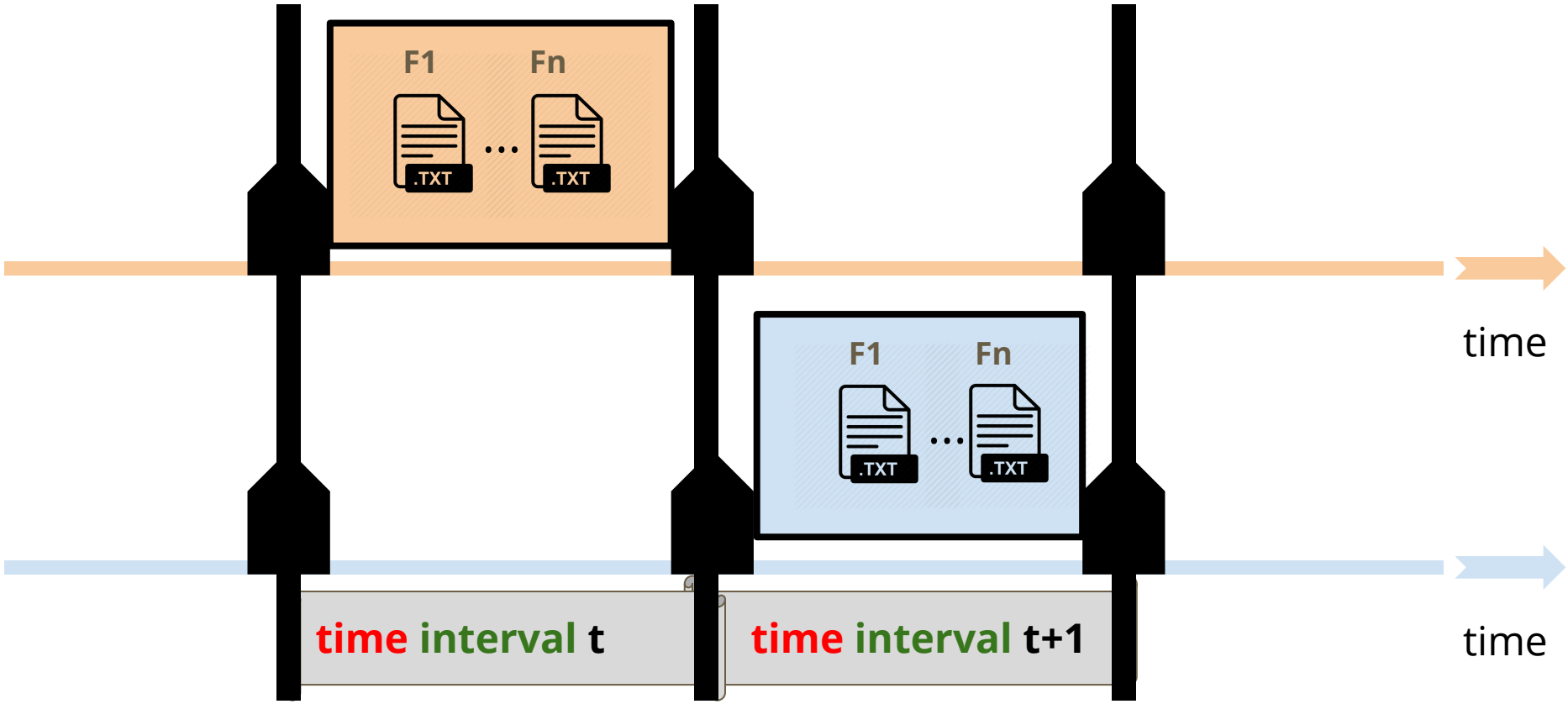
Setting Up the Context

2. For it we need the notion of **time interval**...
   - Allowing new files from our dynamic dataset to arrive!
   - Processing previously arrived files



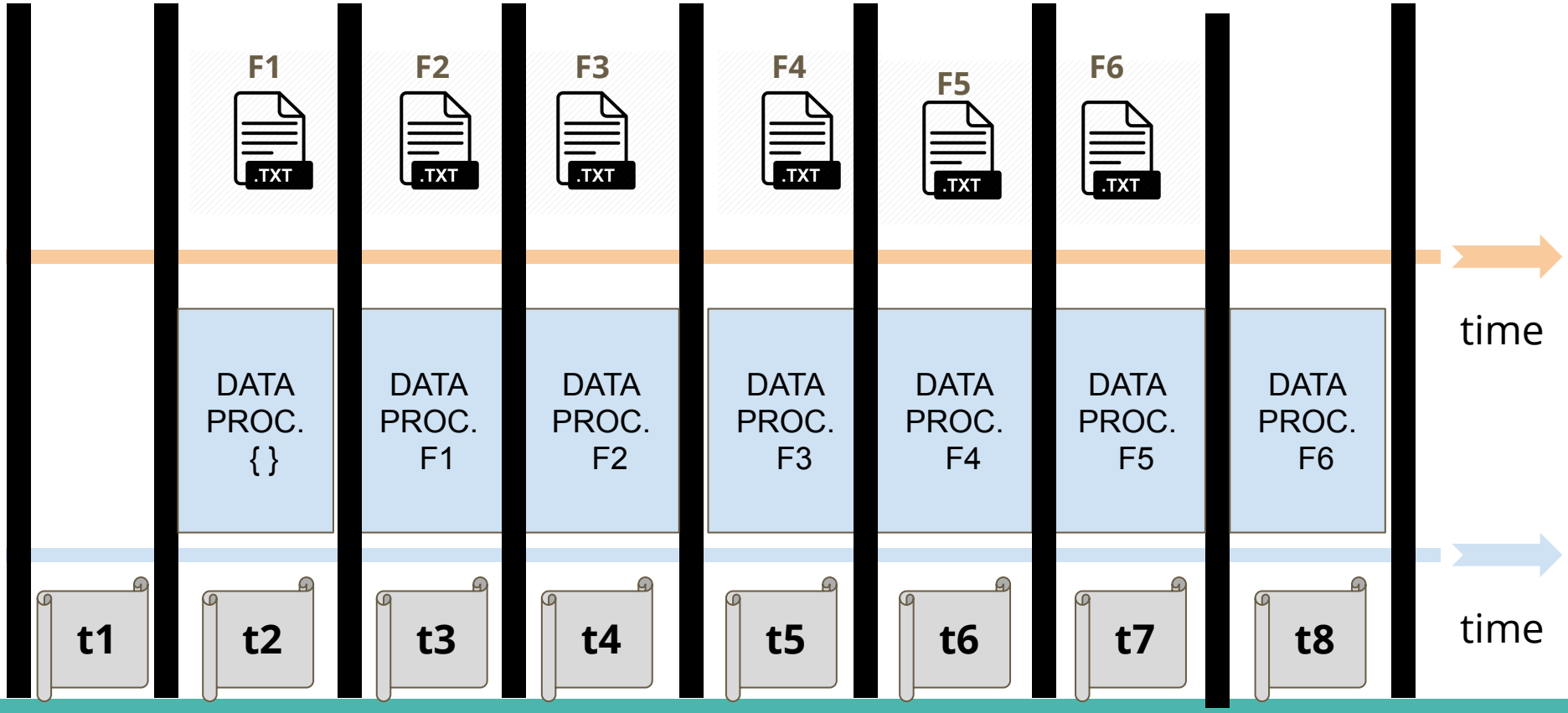
Setting Up the Context

2. For it we need the notion of **time interval**...
   - ➢ In a process that is repeated over and over.



Setting Up the Context

2. For it we need the notion of **time interval**...
   - ➢ In a process that is repeated over and over.

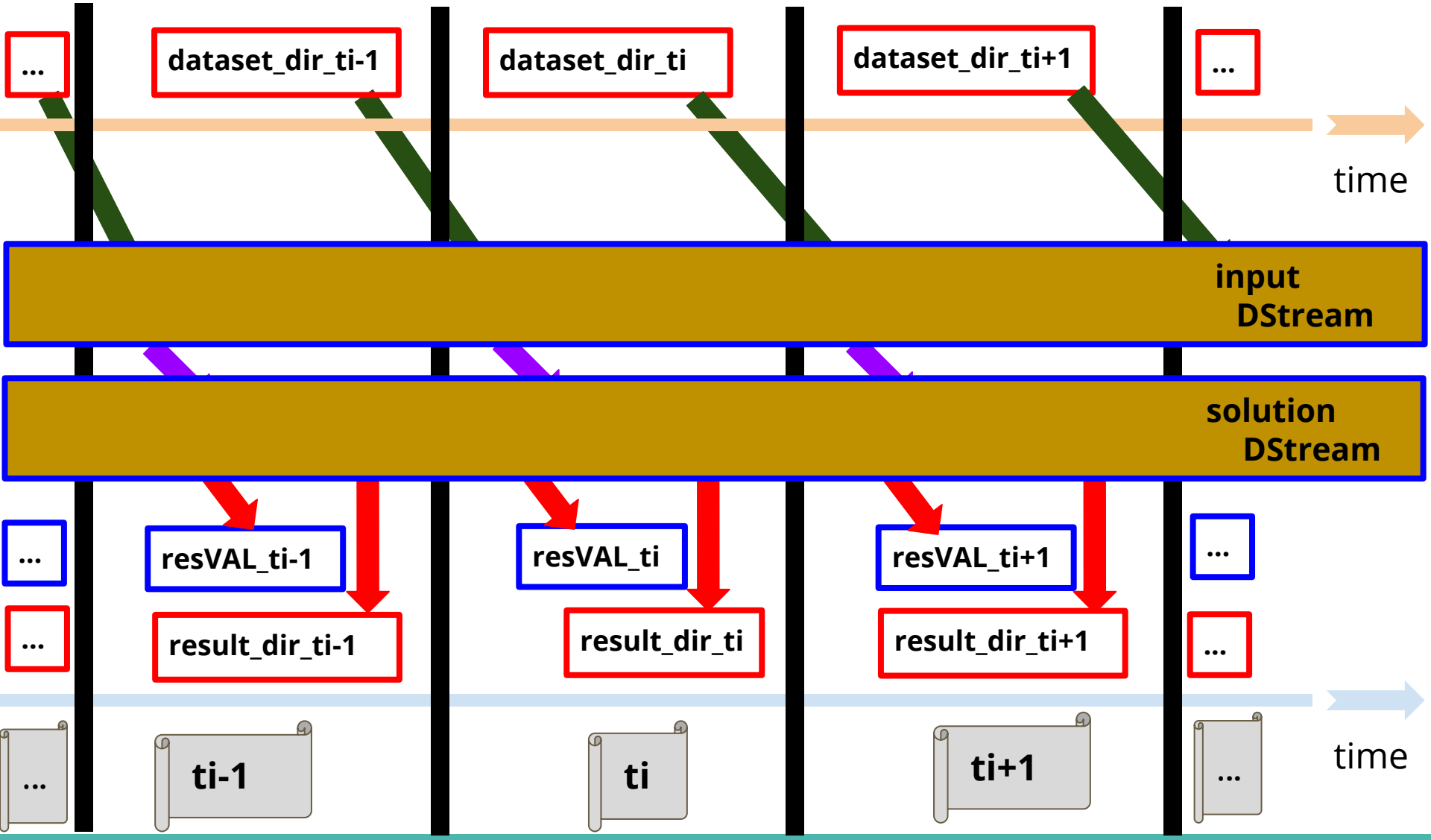


Setting Up the Context

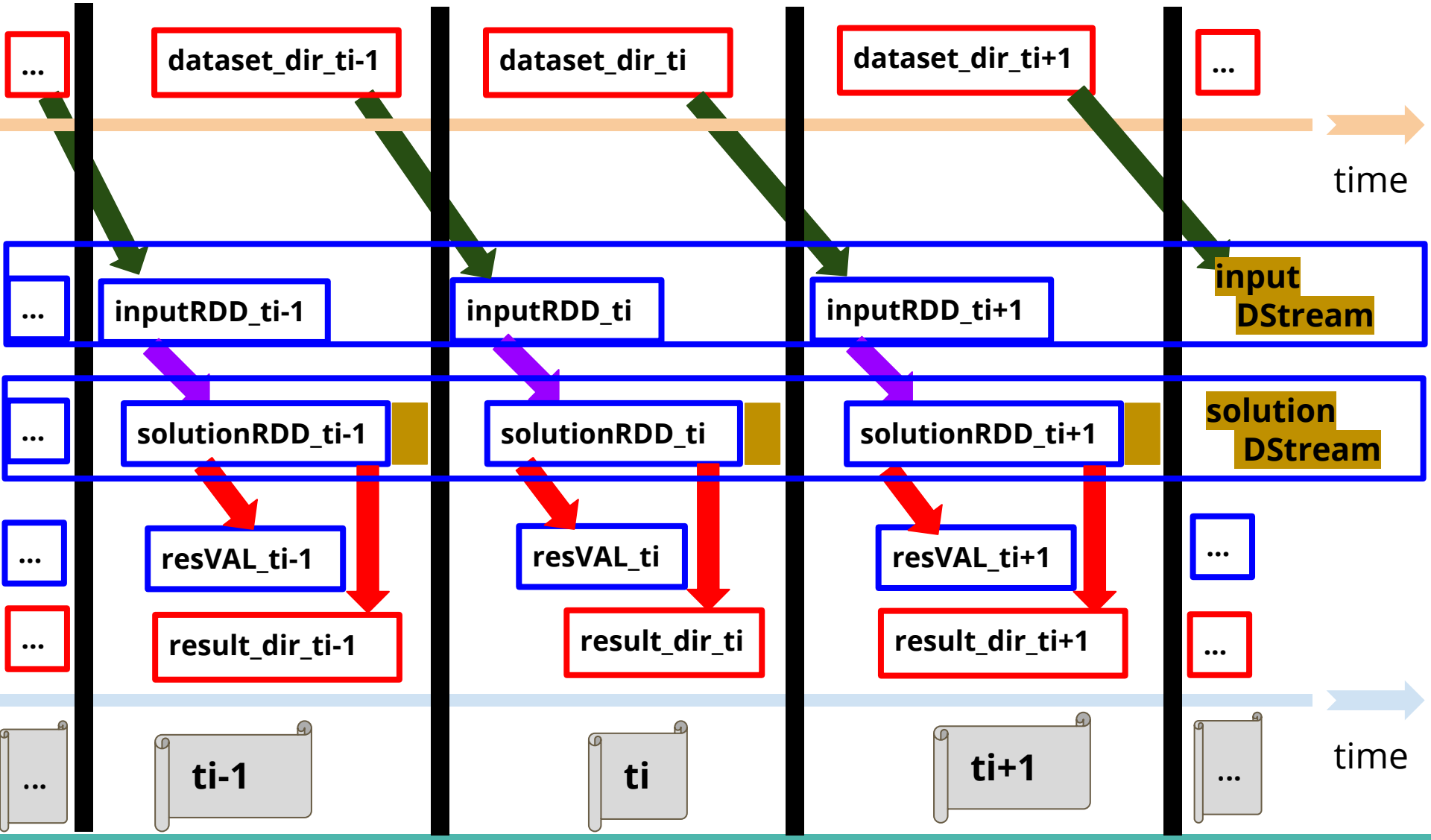
3. The data abstraction of Spark Streaming is a **DStream**, which can be seen as a **train** where each **wagon** represents the underlying **RDD** being processed during a **time interval**.

...  ...

Setting Up the Context

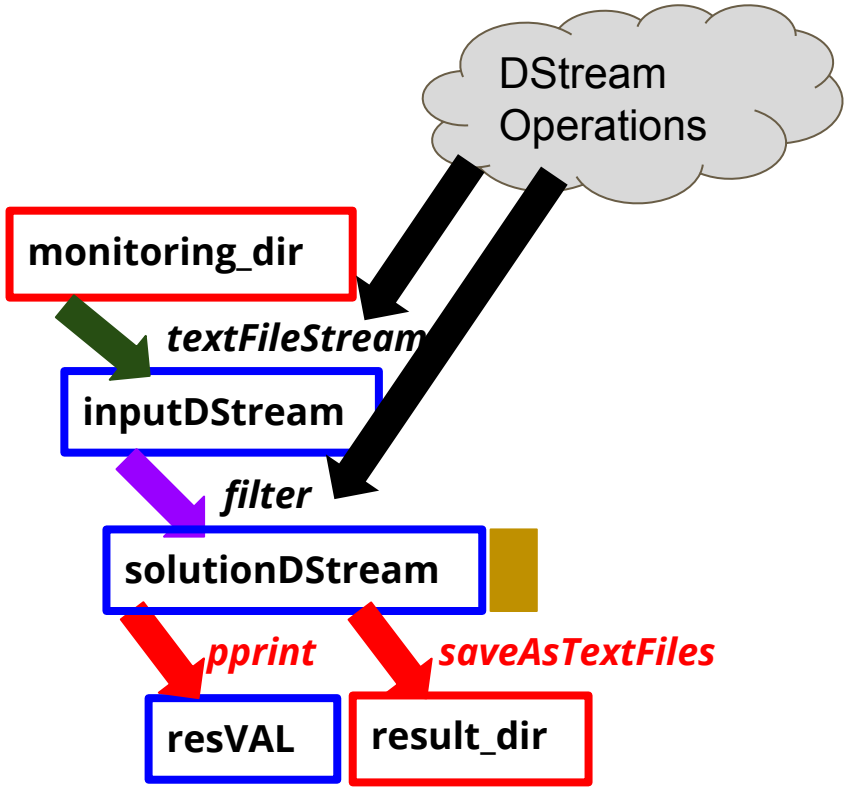


Setting Up the Context

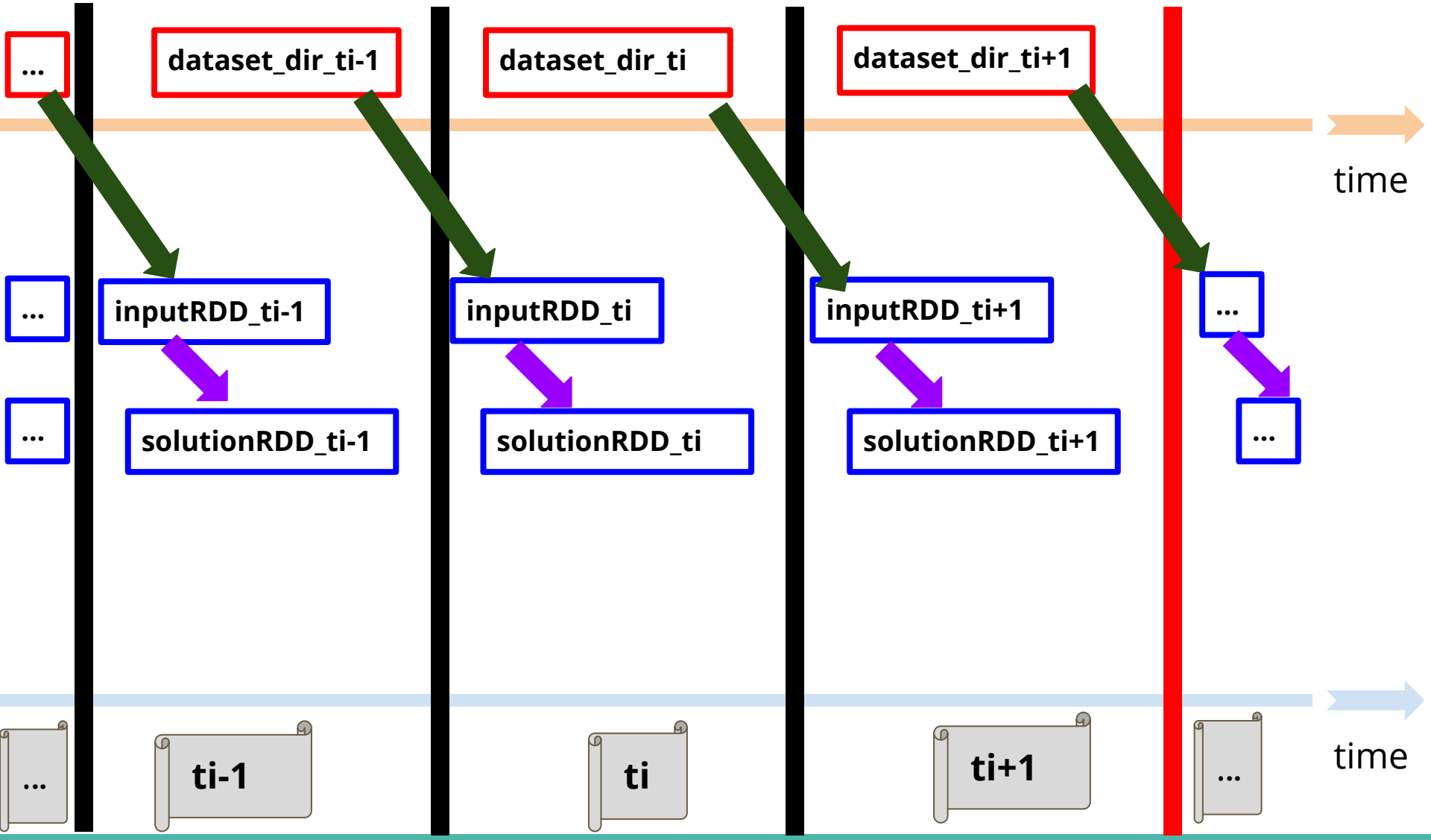


Setting Up the Context

4. Regular **DStream** operations are applied to the underlying **RDDs** over time.



Setting Up the Context



Setting Up the Context

5. Window-based Operations can be used to group **wagons** together.

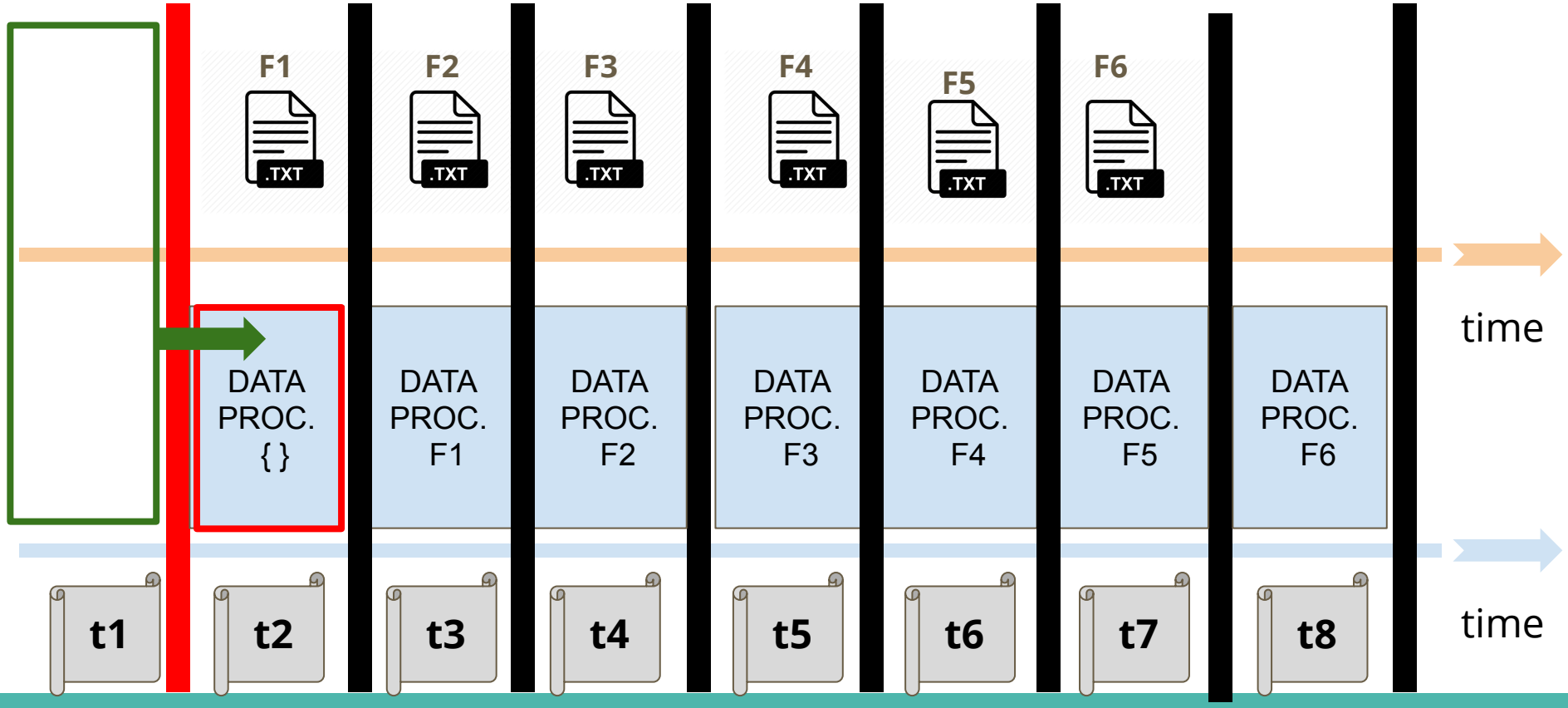
Setting Up the Context

5. Window-based Operations can be used to group **wagons** together.

Setting Up the Context

5. Window-based Operations can be used to group wagons together.

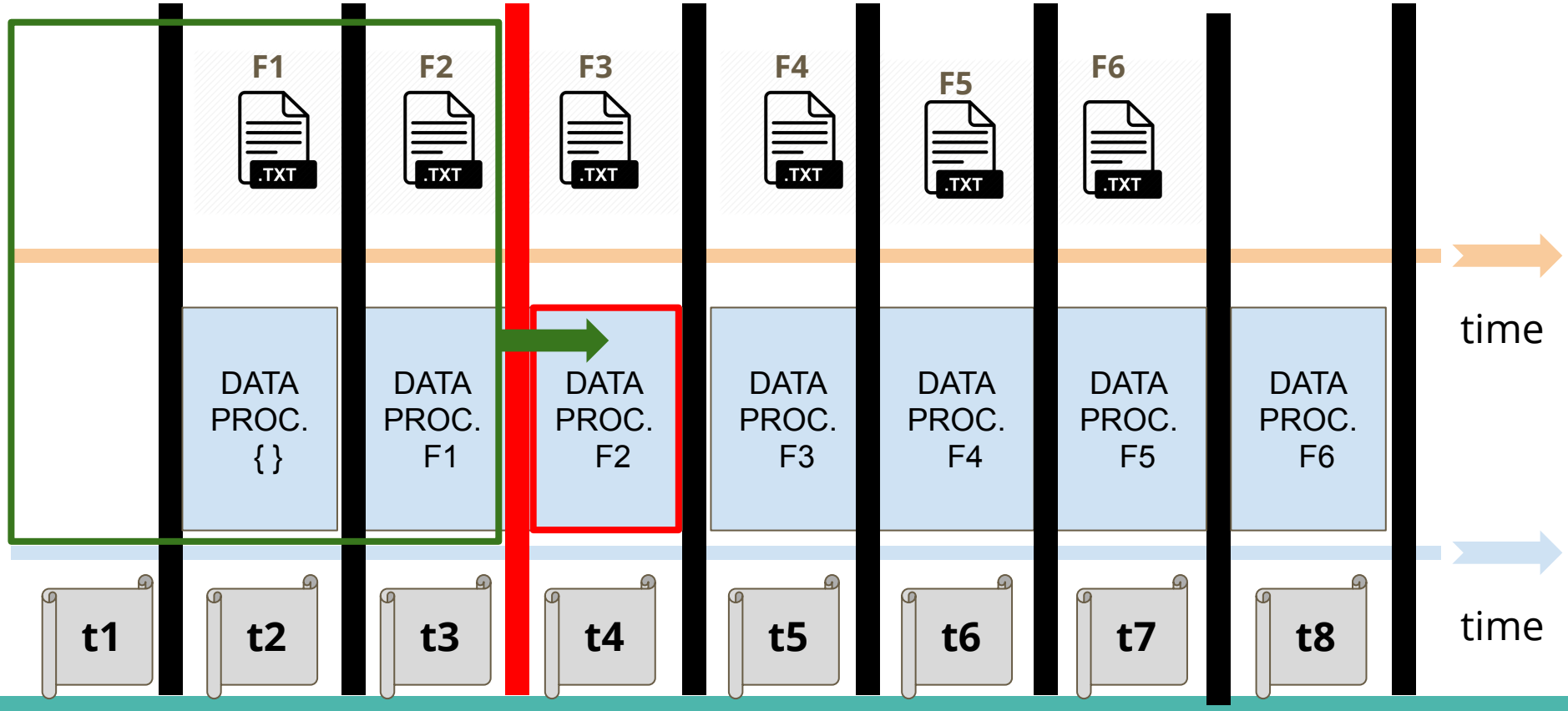
Sliding Duration = 2 and **Window Duration = 3**
we create the following windows at the following times



Setting Up the Context

5. Window-based Operations can be used to group wagons together.

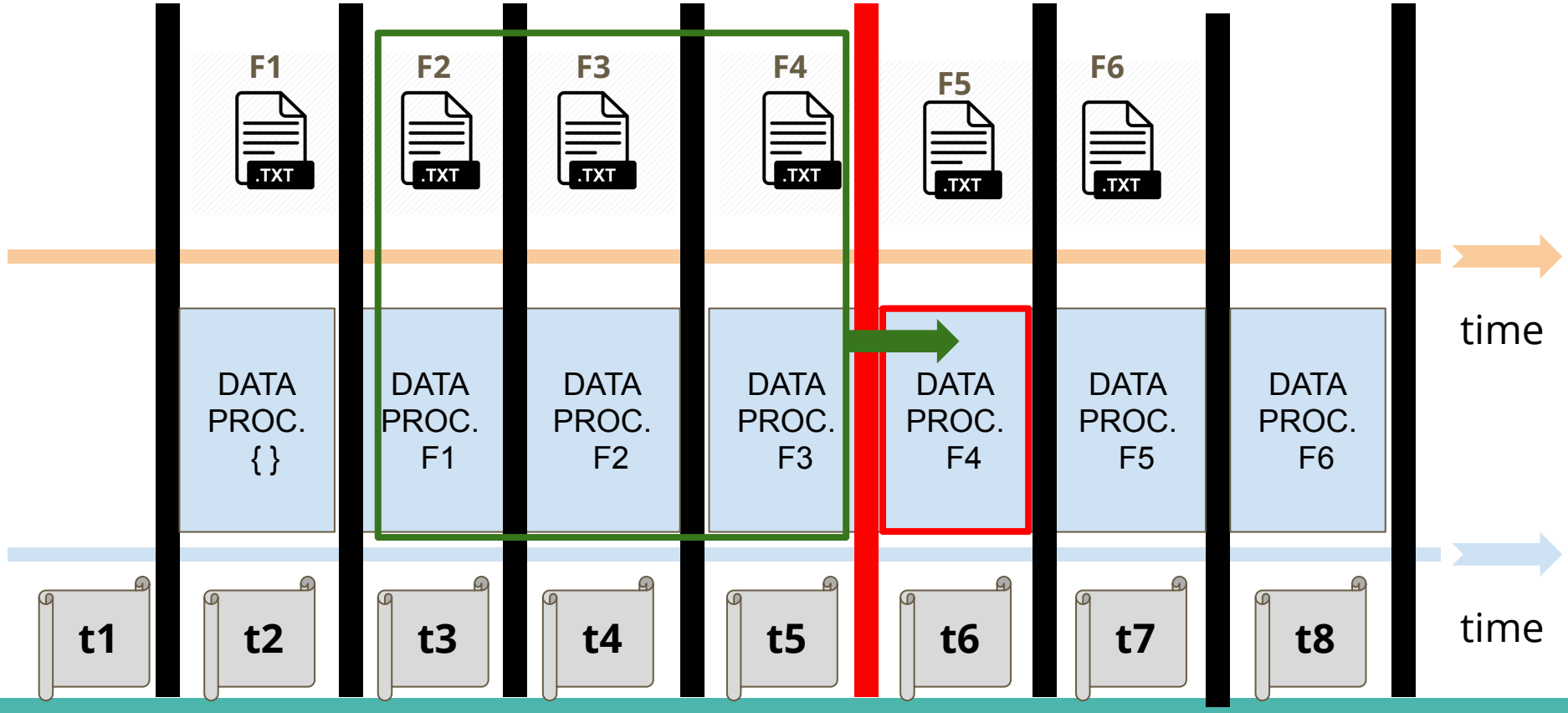
Sliding Duration = 2 and **Window Duration = 3**
we create the following windows at the following times



Setting Up the Context

5. Window-based Operations can be used to group wagons together.

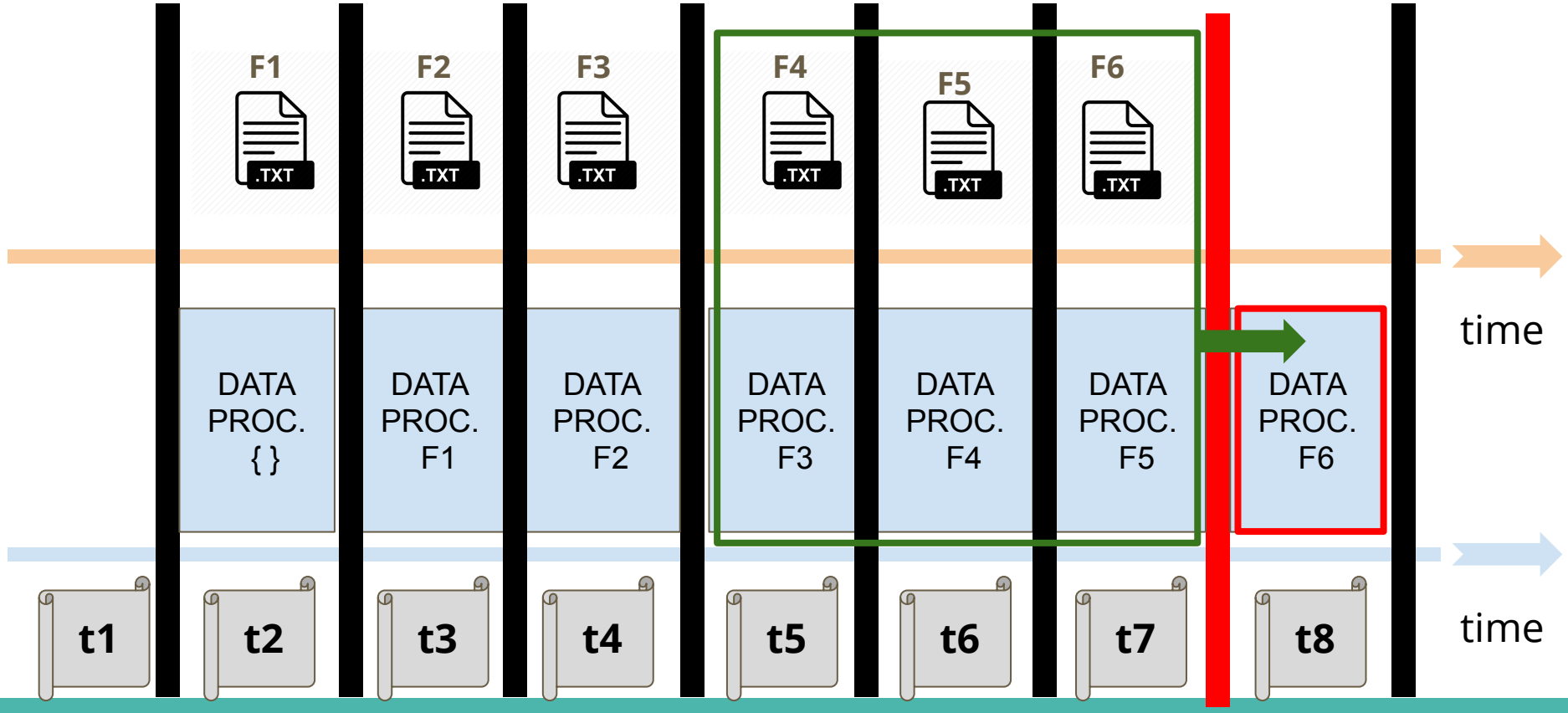
Sliding Duration = 2 and **Window Duration = 3**
we create the following windows at the following times



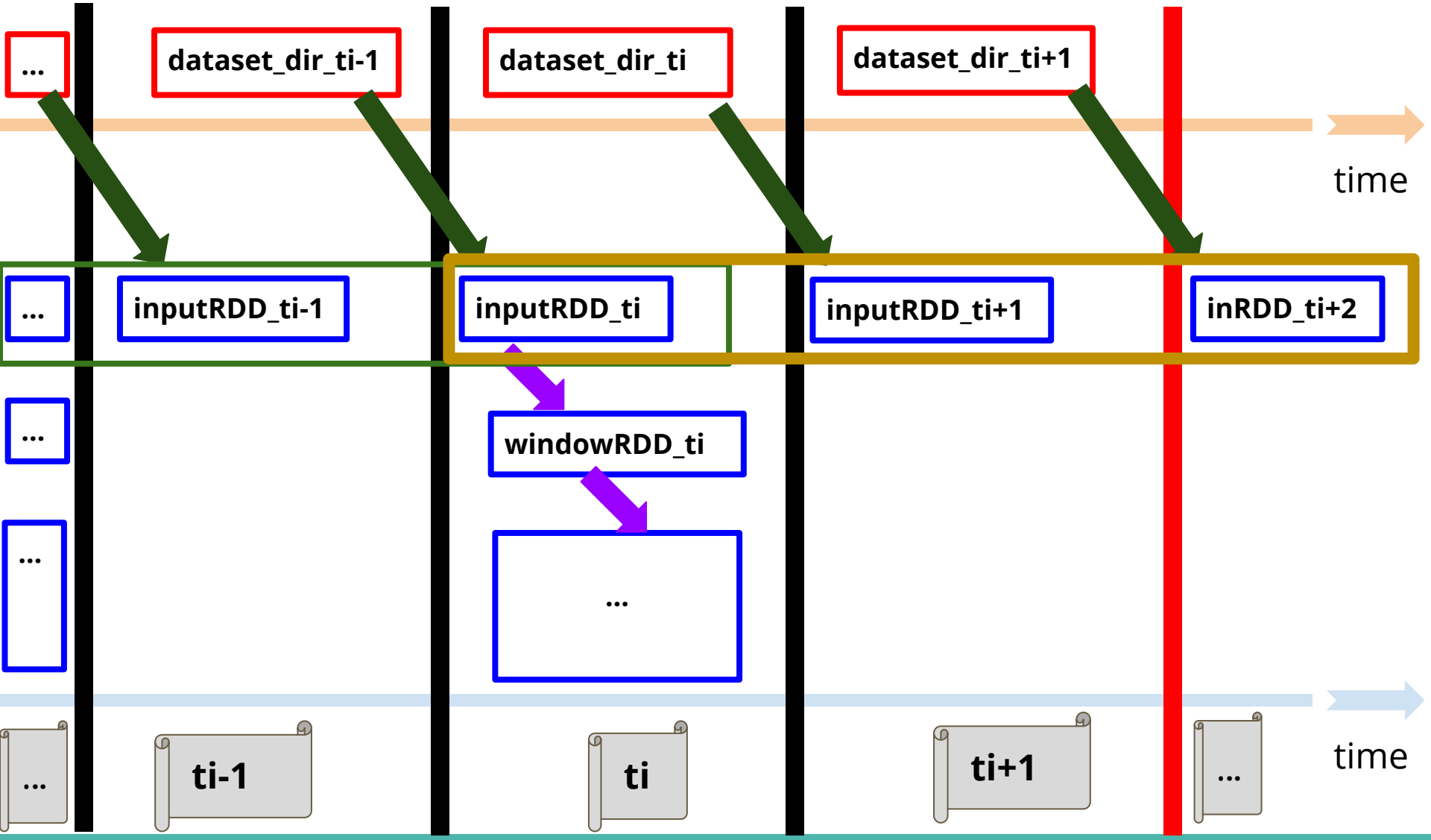
Setting Up the Context

5. Window-based Operations can be used to group wagons together.

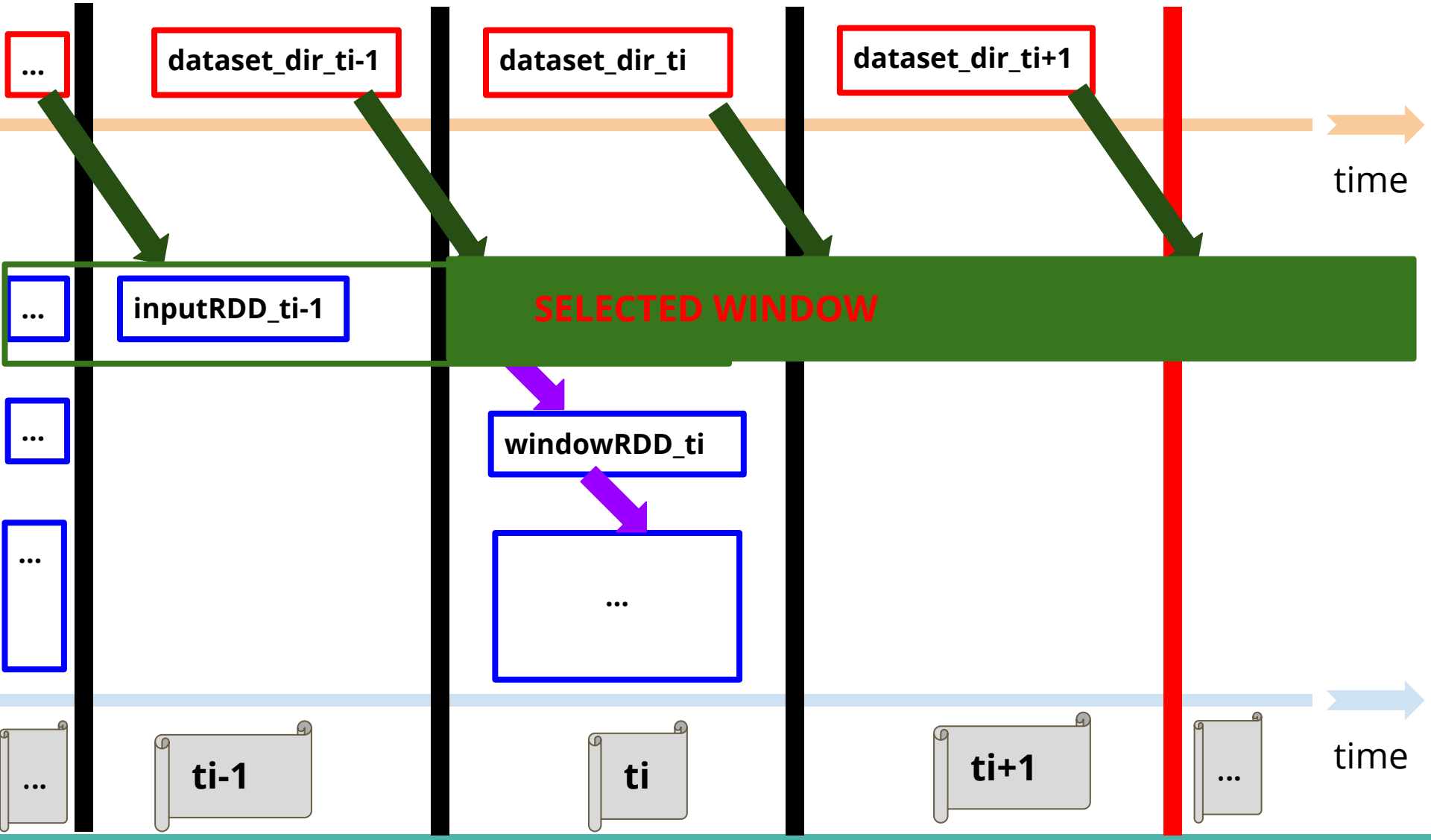
Sliding Duration = 2 and **Window Duration = 3**
we create the following windows at the following times



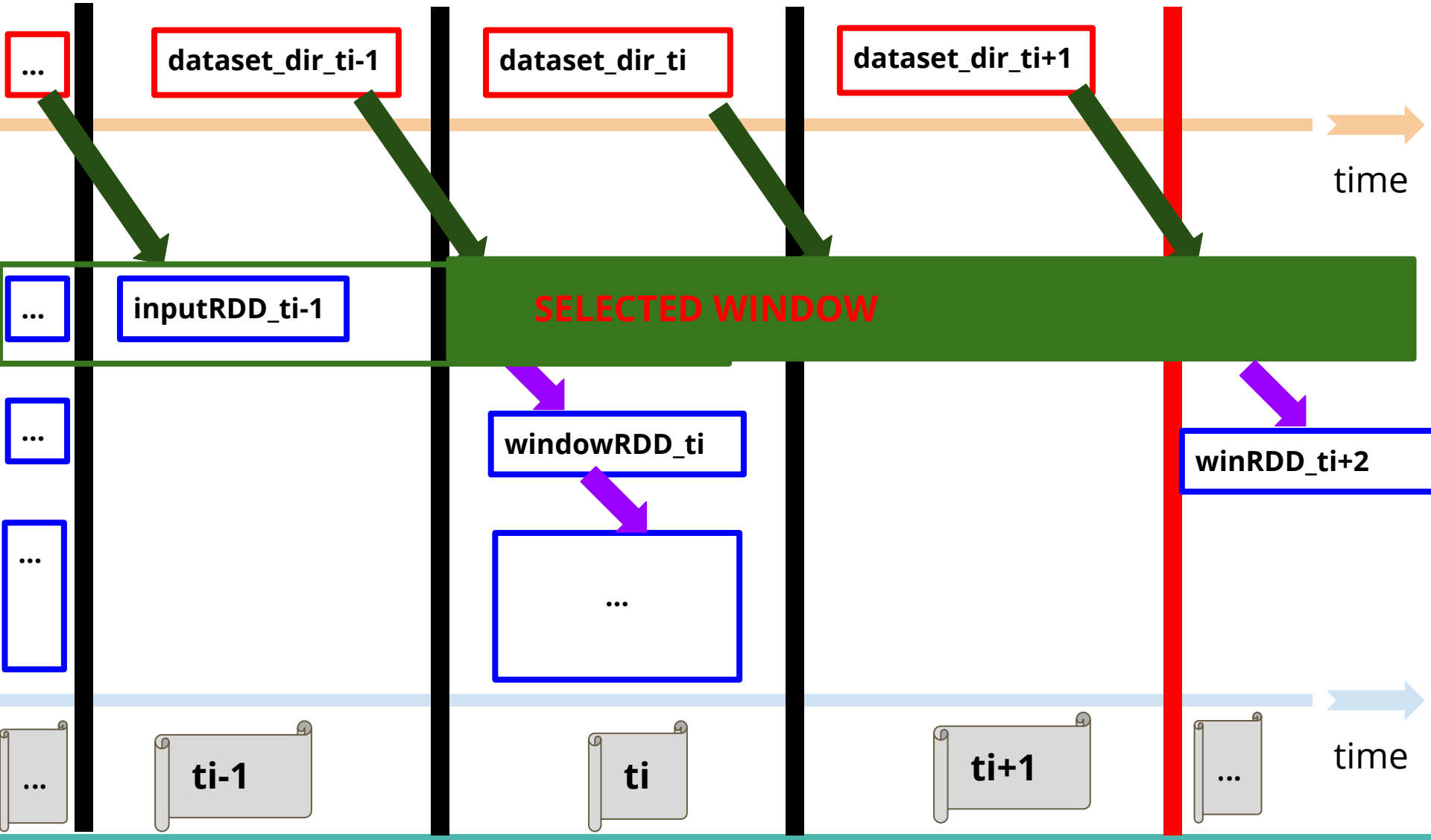
Setting Up the Context



Setting Up the Context



Setting Up the Context



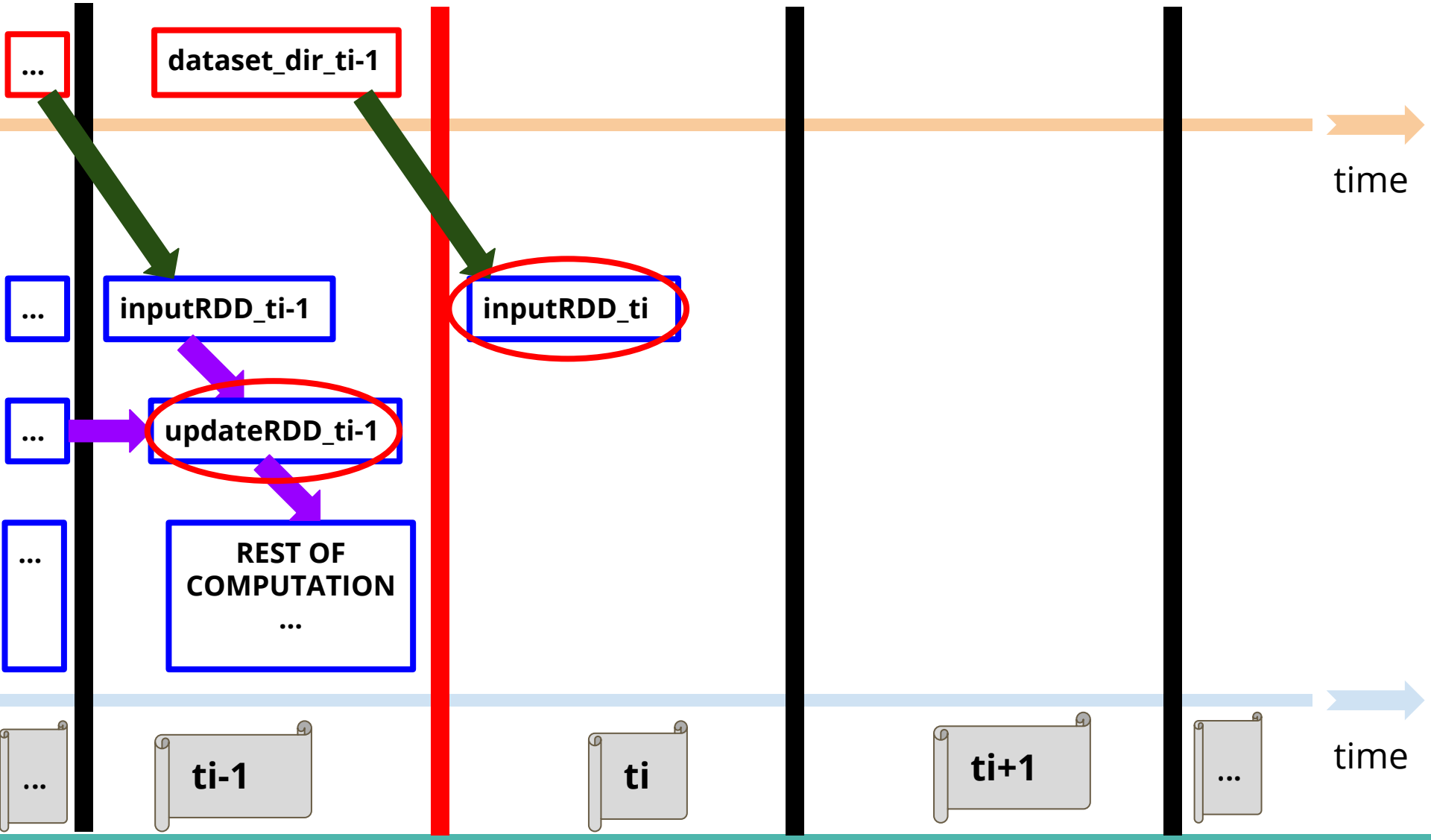
Setting Up the Context

6. Update-based operations aggregate results for all the **wagons** processed so far.

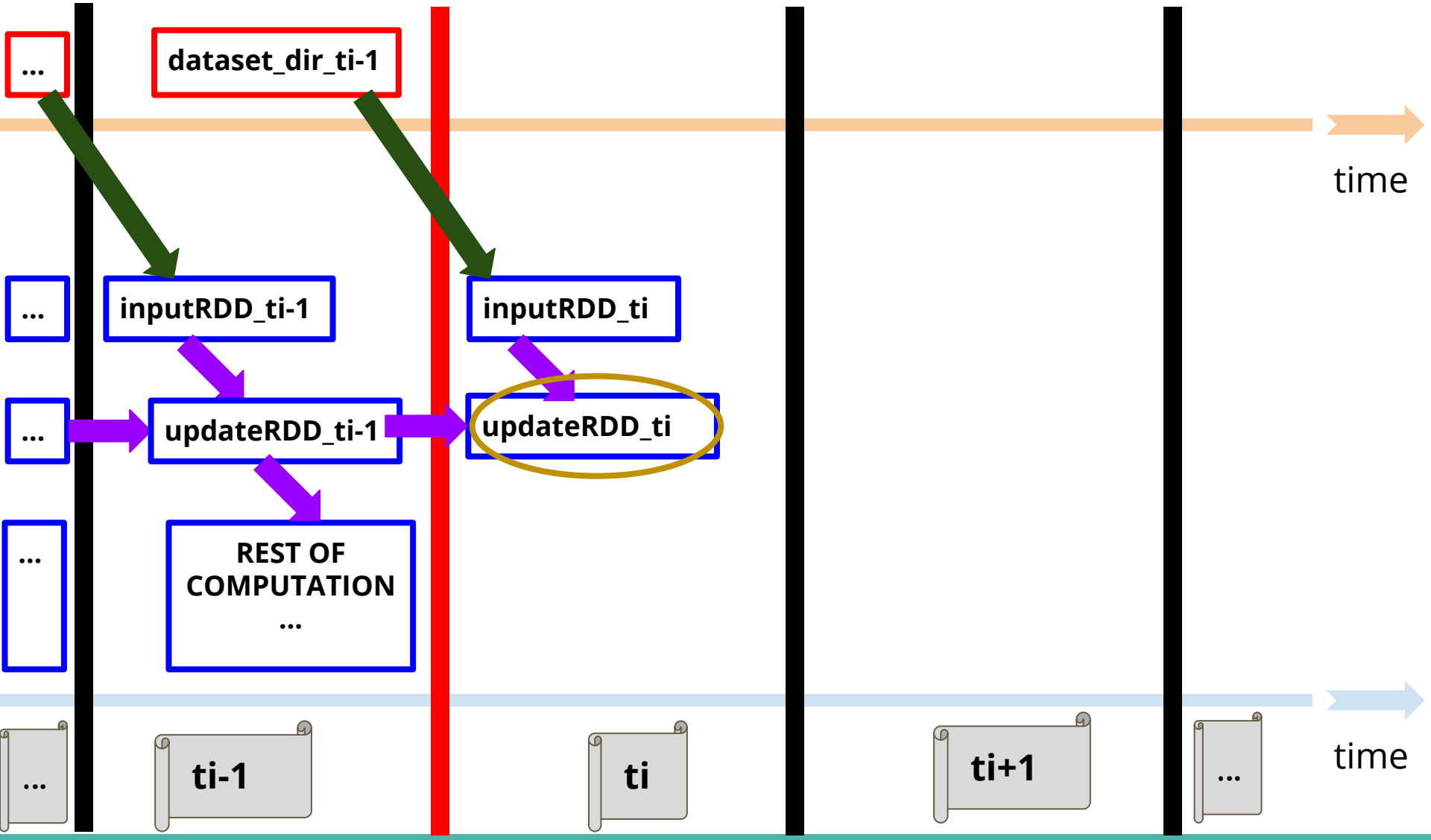
Setting Up the Context

6. Update-based operations aggregate results for all the **wagons** processed so far.

Setting Up the Context



Setting Up the Context



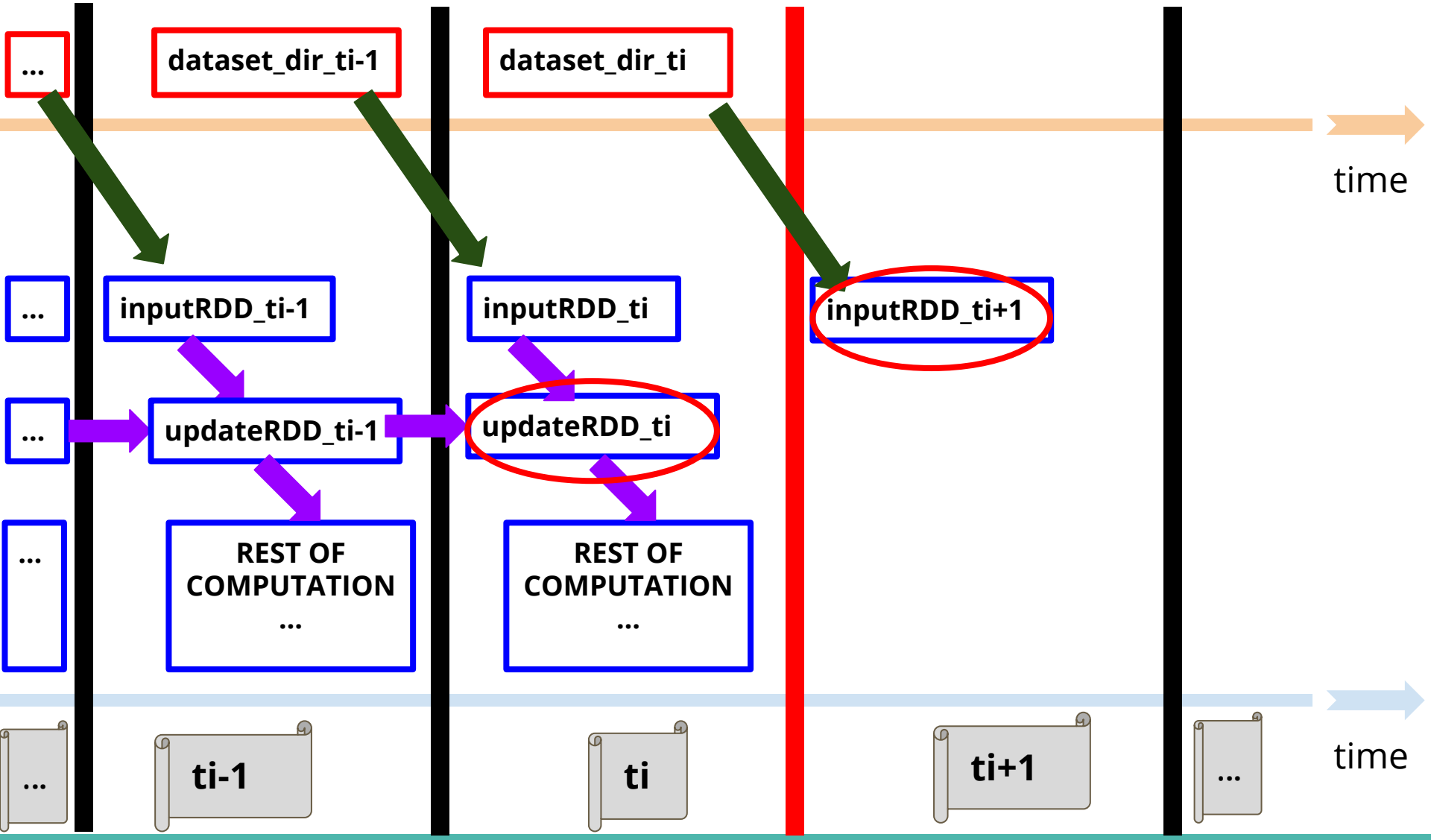
Setting Up the Context

6. Update-based operations aggregate results for all the **wagons** processed so far.

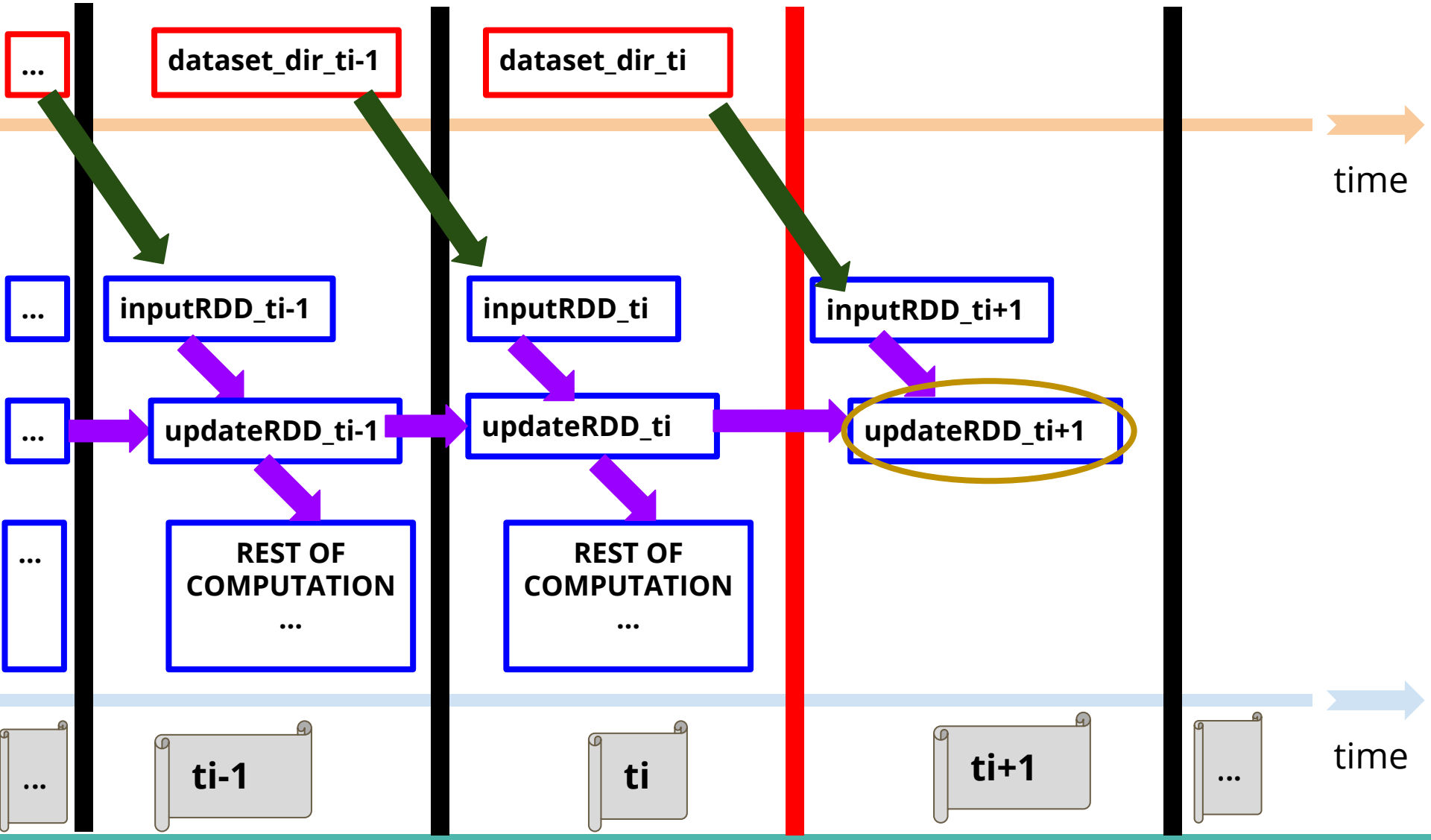
Setting Up the Context

6. Update-based operations aggregate results for all the **wagons** processed so far.

Setting Up the Context



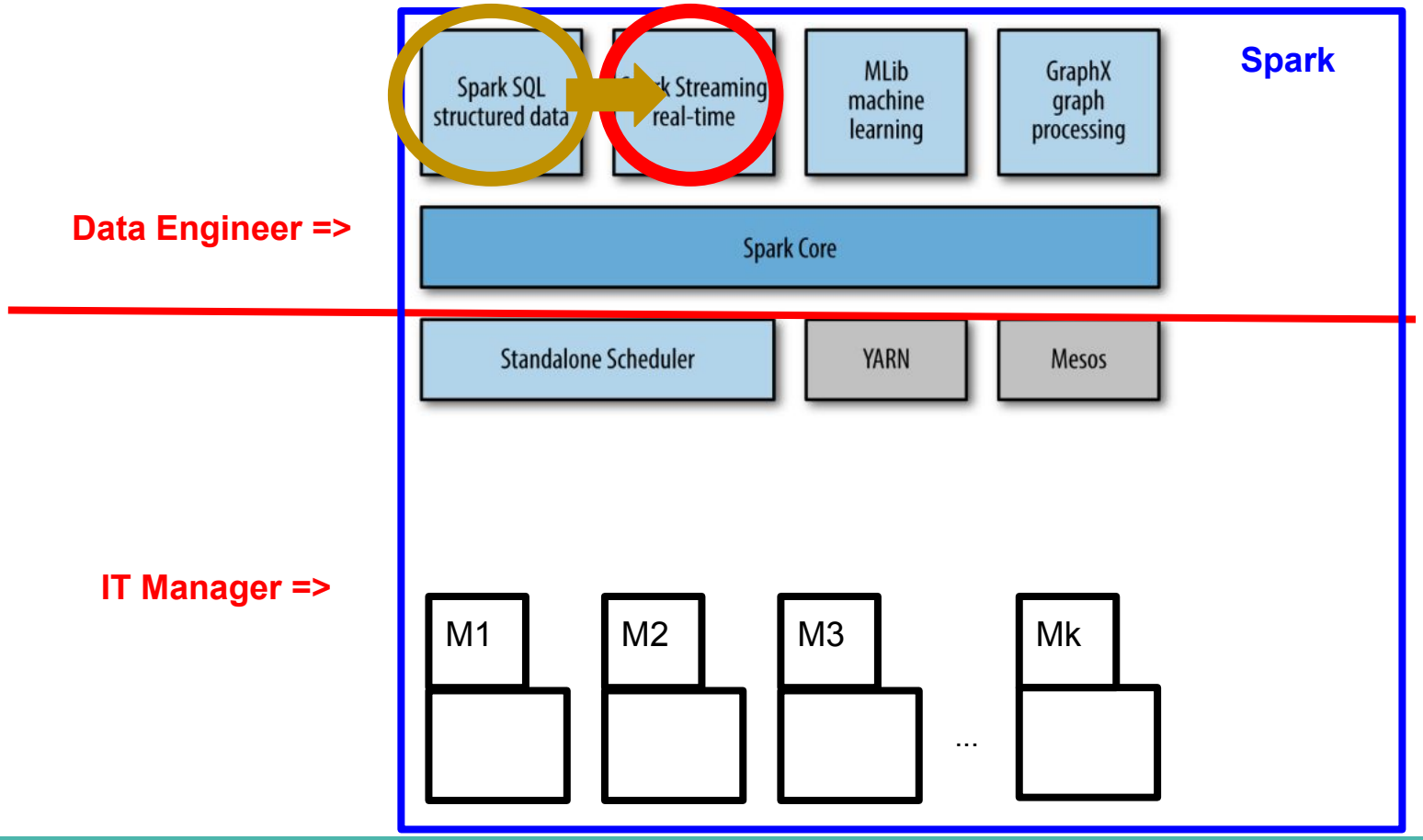
Setting Up the Context



Setting the Context

In this lecture we focus on using the library
Spark Structured Streaming
to extend Spark SQL
with streaming functionality.

Setting the Context



Setting the Context

One of my best professors stated once...
"When learning a new topic, let's approach the unknown concepts via known concepts".

So let's follow this approach and learn
Spark Structured Streaming
via **Spark Streaming**.

Setting the Context

In particular, we revise the functionality of
Spark Streaming
and evaluate whether and how is it still applicable to
Spark Structured Streaming.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.

From DStream to SDF

There are 5 main concepts we want to present!

Outline

1. Setting Up the Context.
2. From DStream to SDF.
   a. Concept1: SDF.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
3. Practising with the Concepts.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
   a. Concept1: SDF.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
3. Practising with the Concepts.

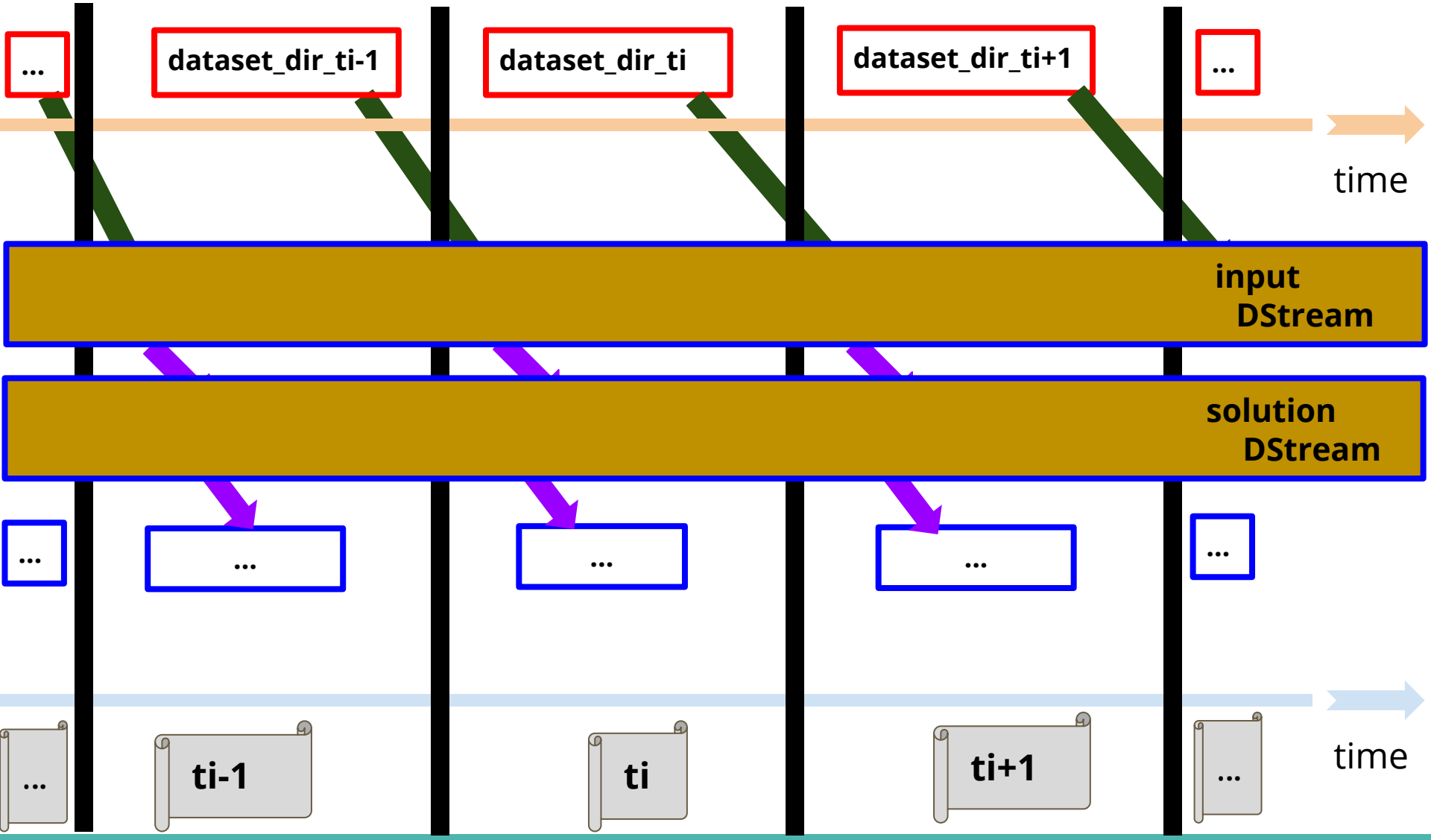
Concept1: SDF

A **DStream** represents an amalgamation of **RDDs** over time...

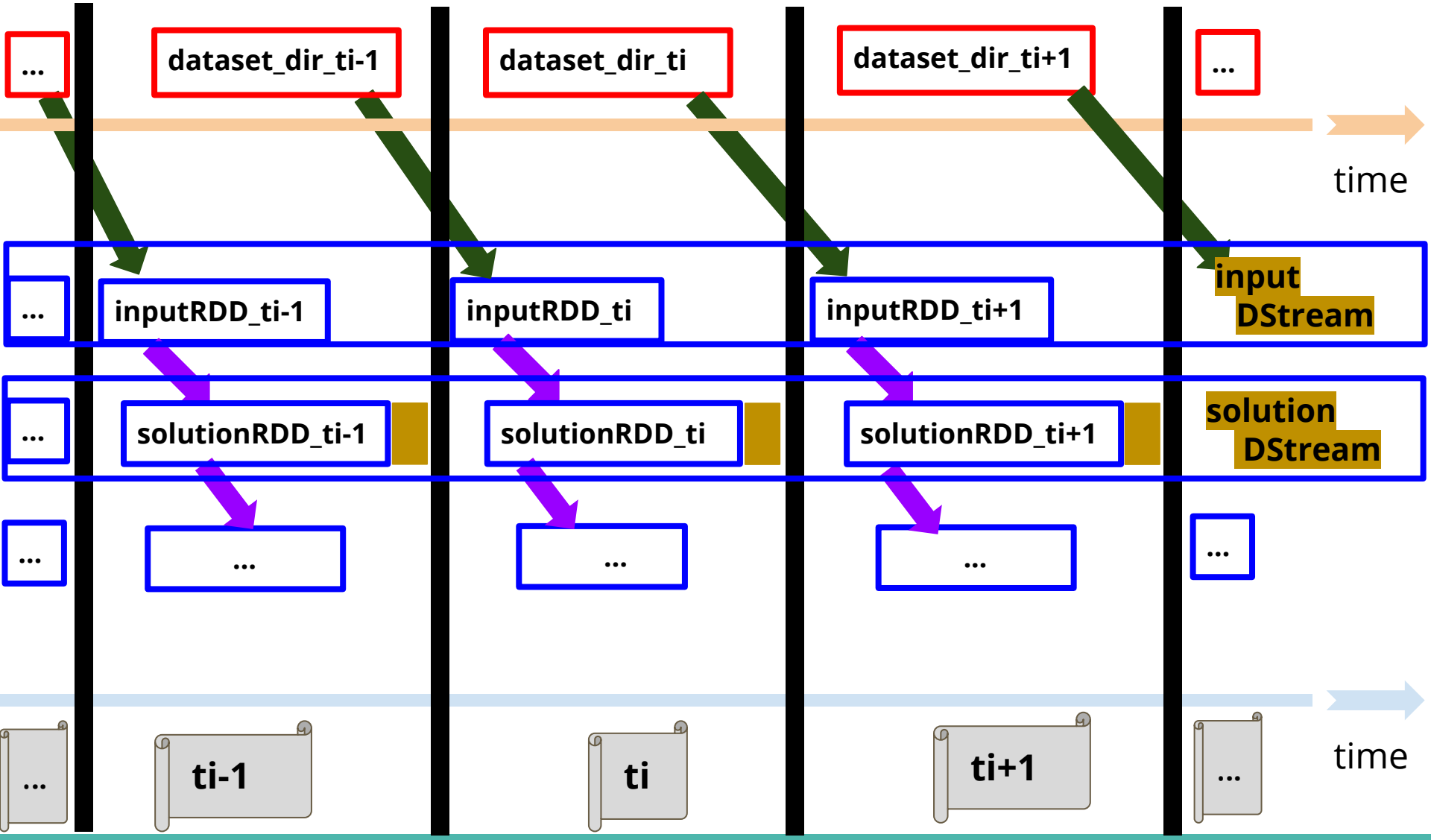
...

...

Concept1: SDF



Concept1: SDF



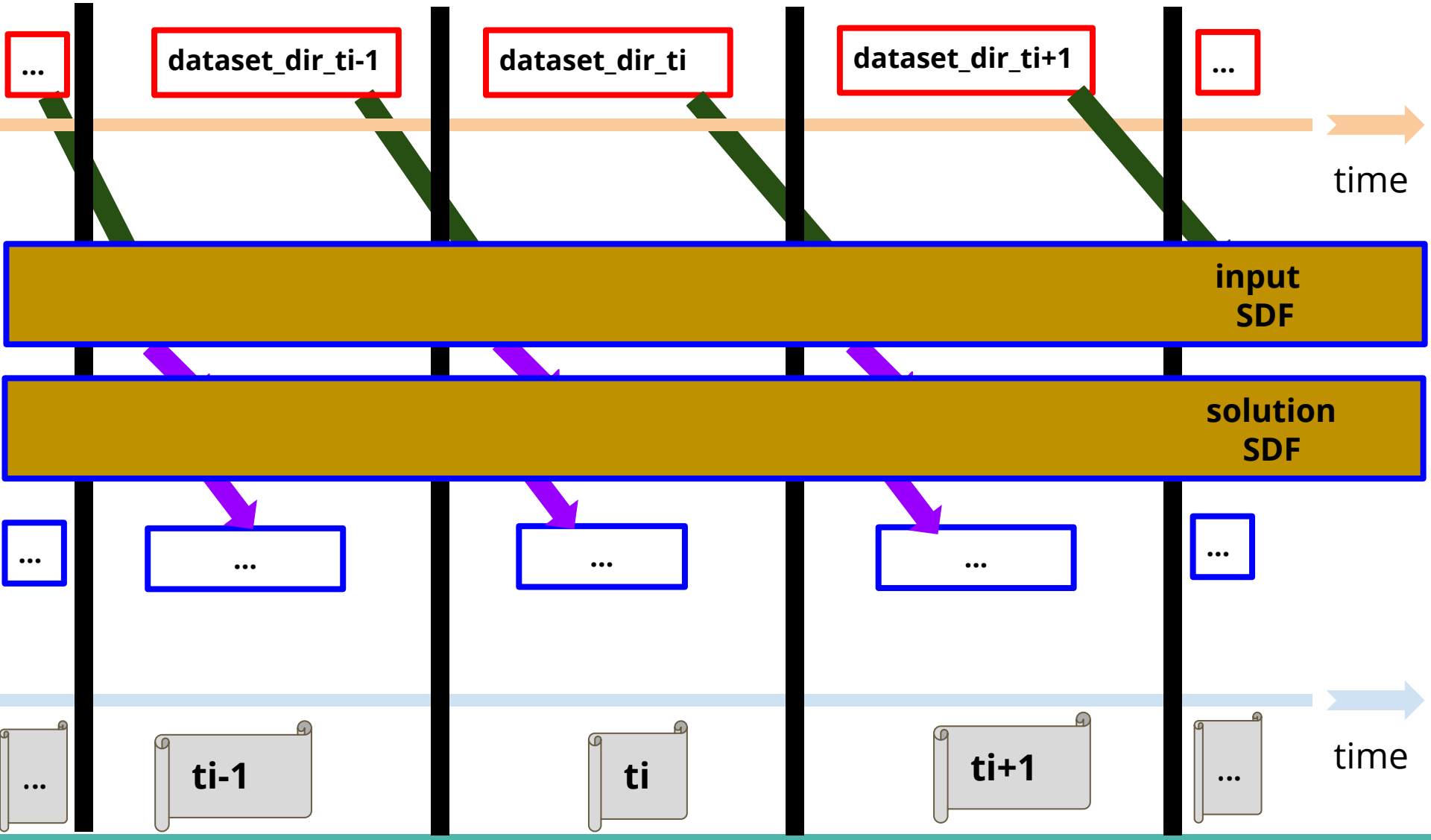
Concept1: SDF

The same behaviour applies to **SDF**!

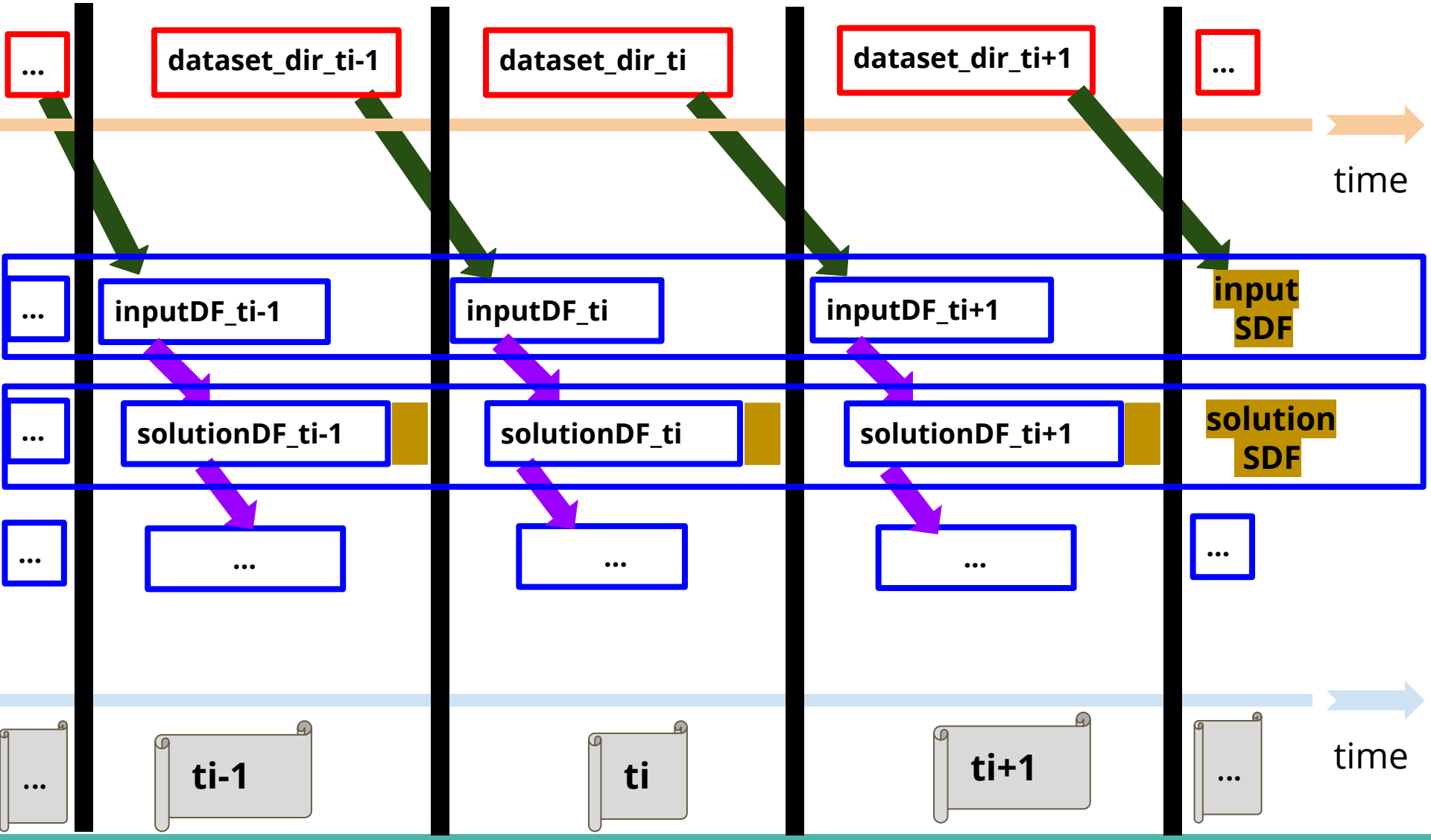
Concept1: SDF

A **Streaming DataFrame (SDF)** also represents an amalgamation of **DataFrames (DF)** over time.

Concept1: SDF



Concept1: SDF



Concept1: SDF

The SDF can also be visualised as an Unbound table, which keeps receiving Rows over time.

inputSDF

...

	Name	City	Eyes
Row1	...	...	...
Row2	...	...	...

inputDF_ti-1

	Name	City	Eyes
Row3	...	...	...

inputDF_ti

	Name	City	Eyes
Row4	...	...	...

inputDF_ti+1

...

Concept1: SDF

In any case, whether you use:

1. An horizontal viewing based on a train of wagons (DFs)
2. A vertical viewing based on an unbound data structure of Rows

the behaviour of **DStream** and **SDF** is equivalent!

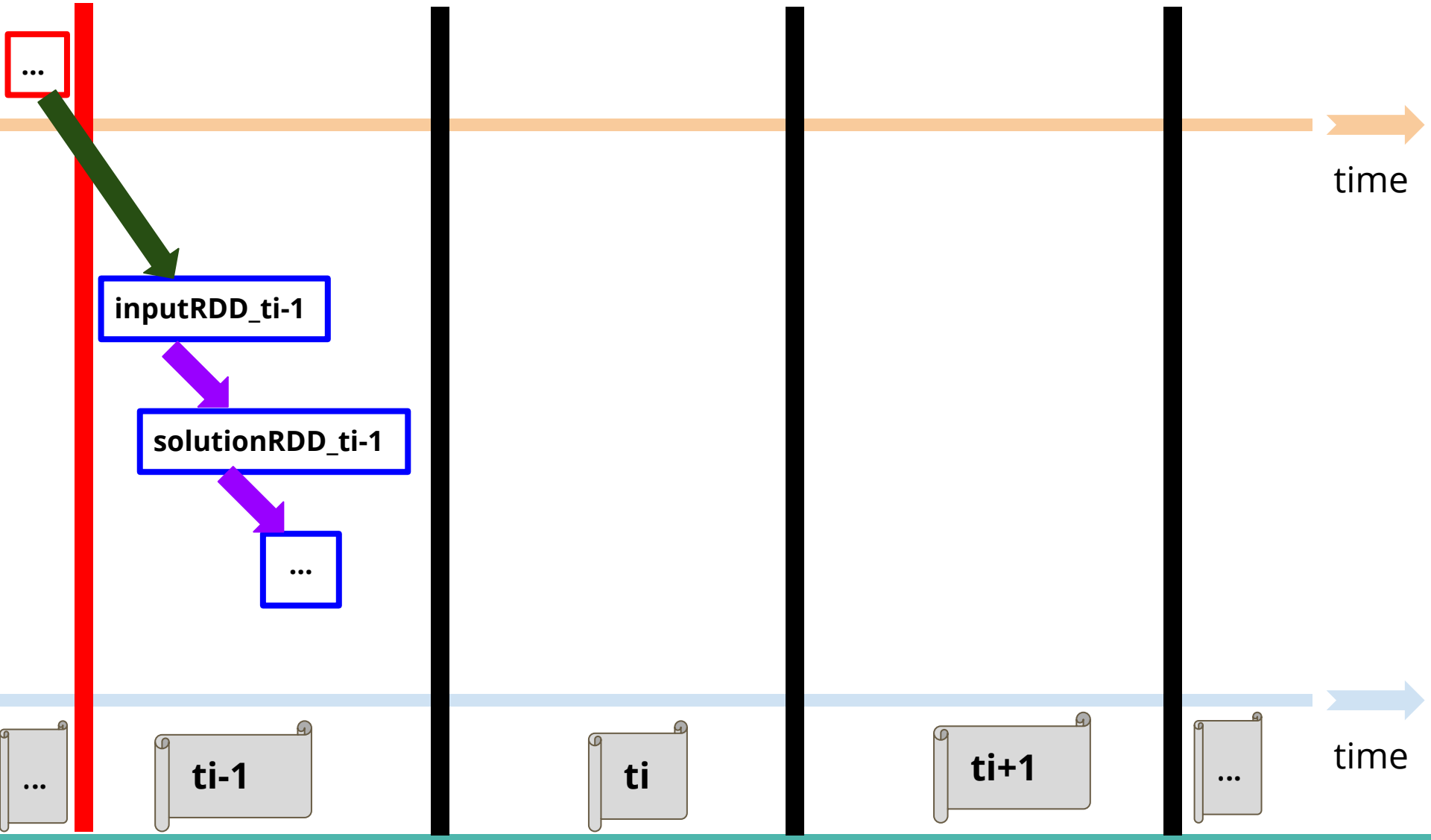
Outline

1. Setting Up the Context.
2. From DStream to SDF.
   a. Concept1: SDF.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
3. Practising with the Concepts.

Concept2: Append Mode

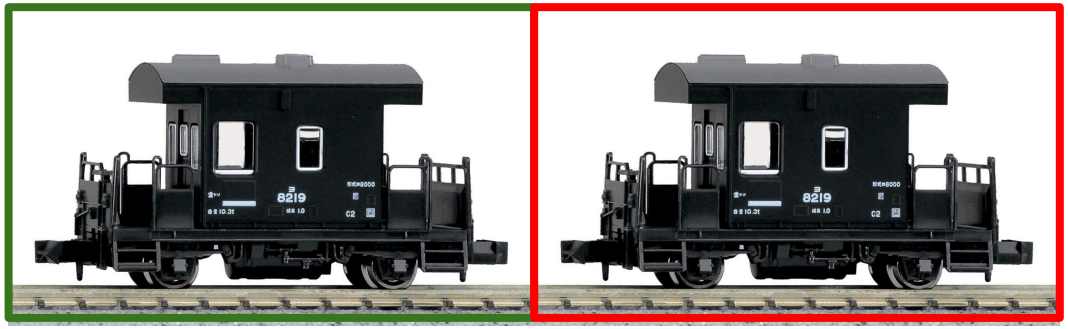
A **DStream** does not increase its number of wagons (**RDDs)** over time...

Concept2: Append Mode

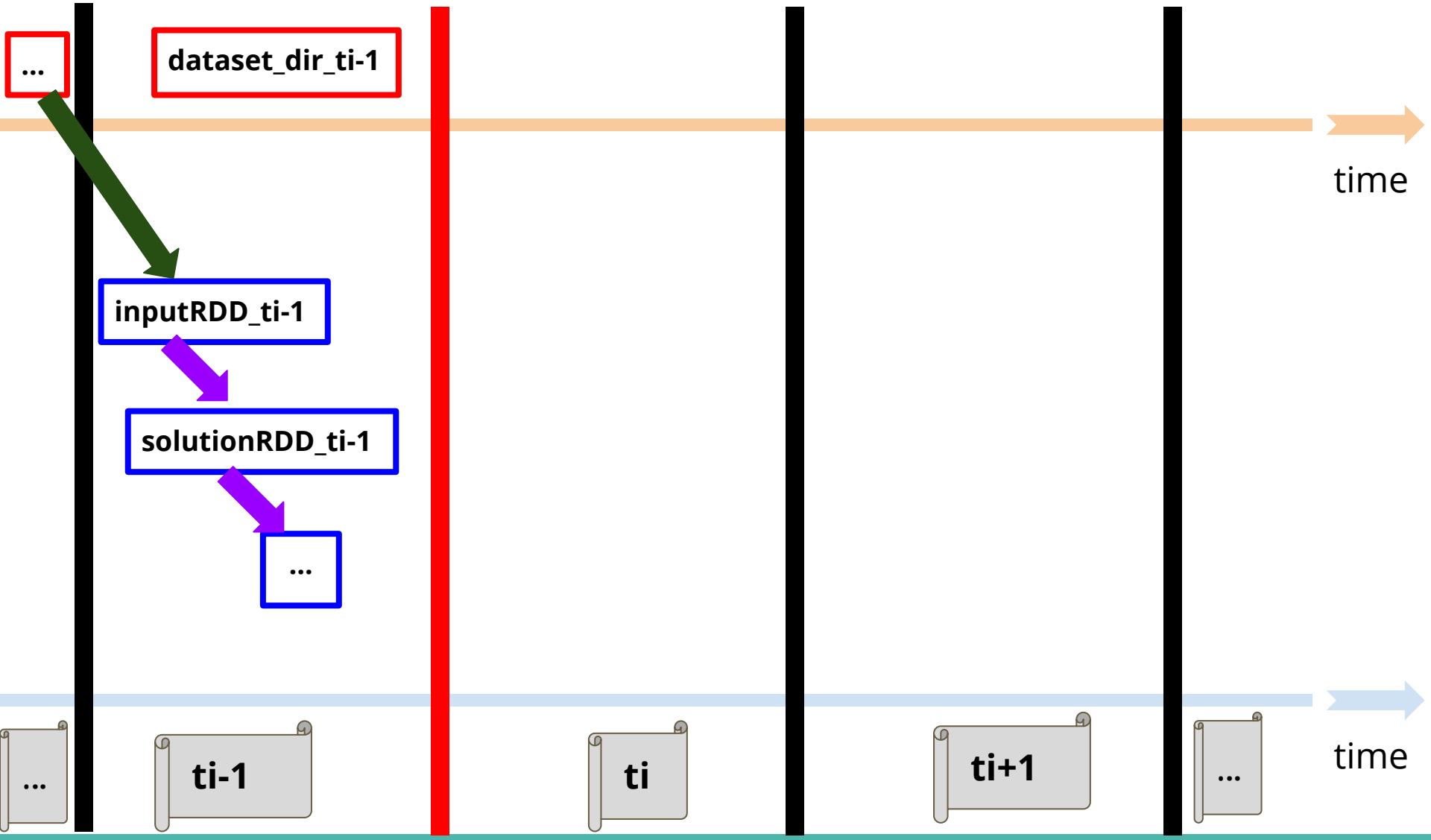


Concept2: Append Mode

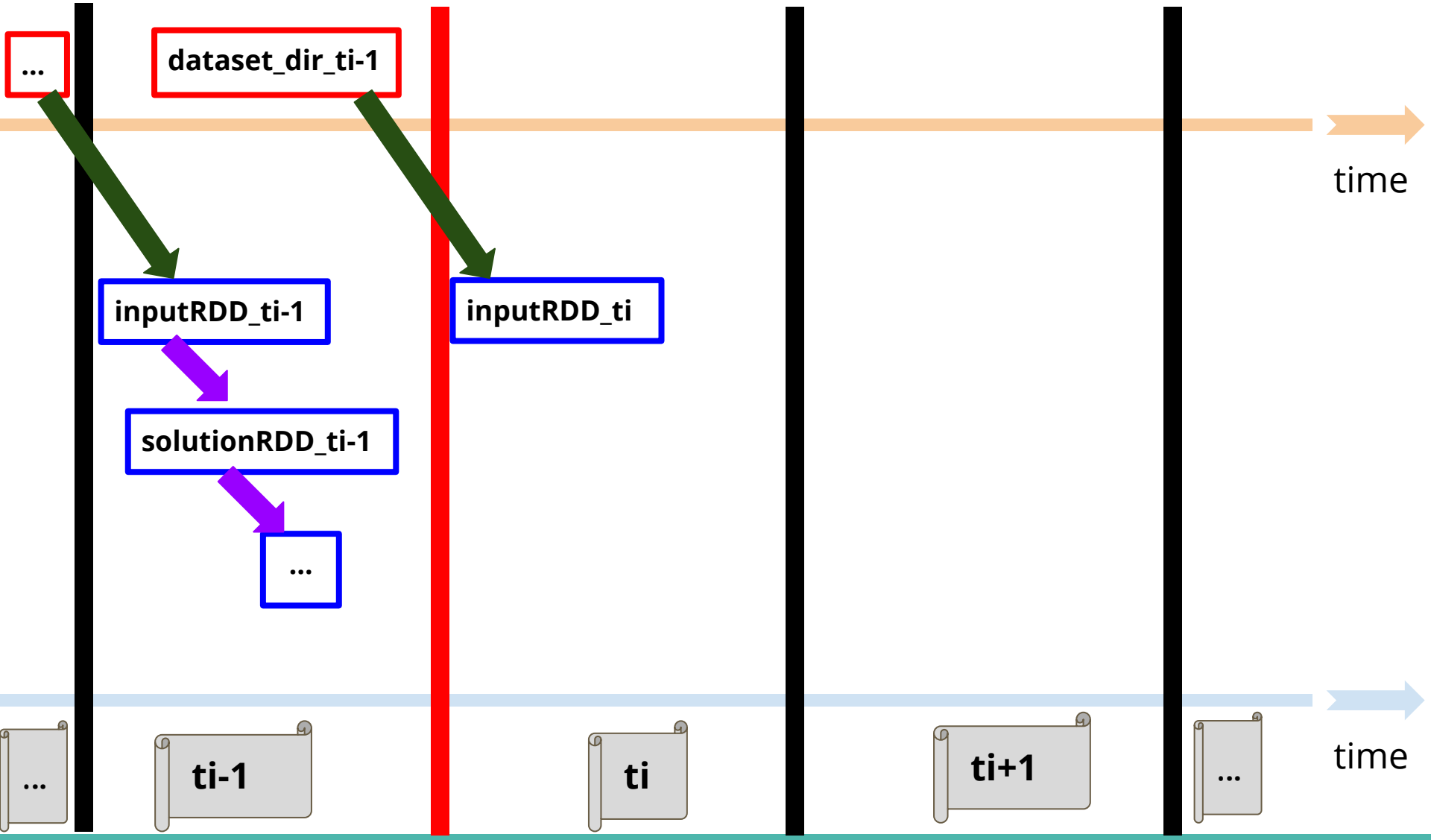
A **DStream** <u>does not increase</u> its number of wagons (**RDDs)** over time...
...as each time a new wagon comes in...



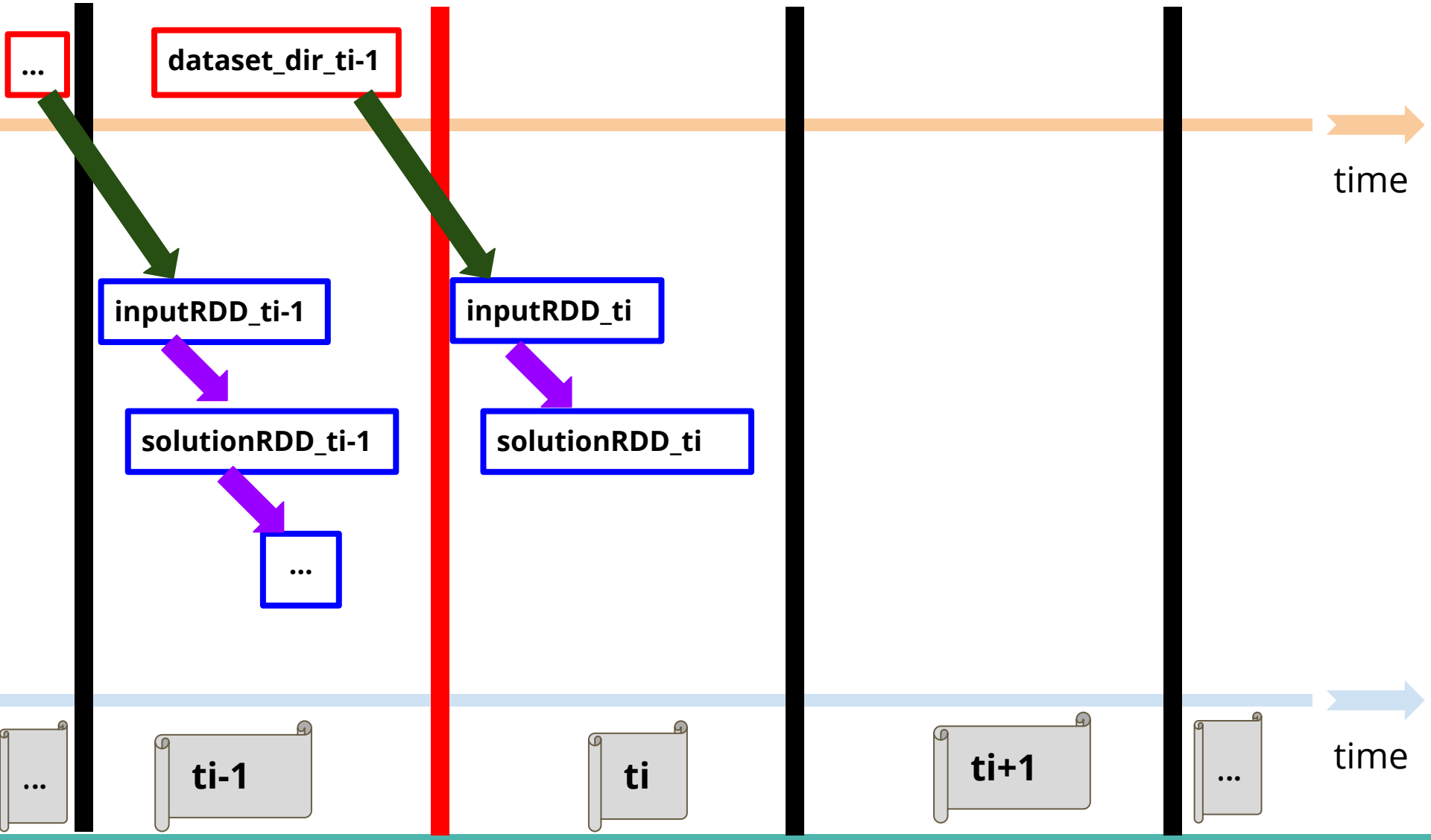
Concept2: Append Mode



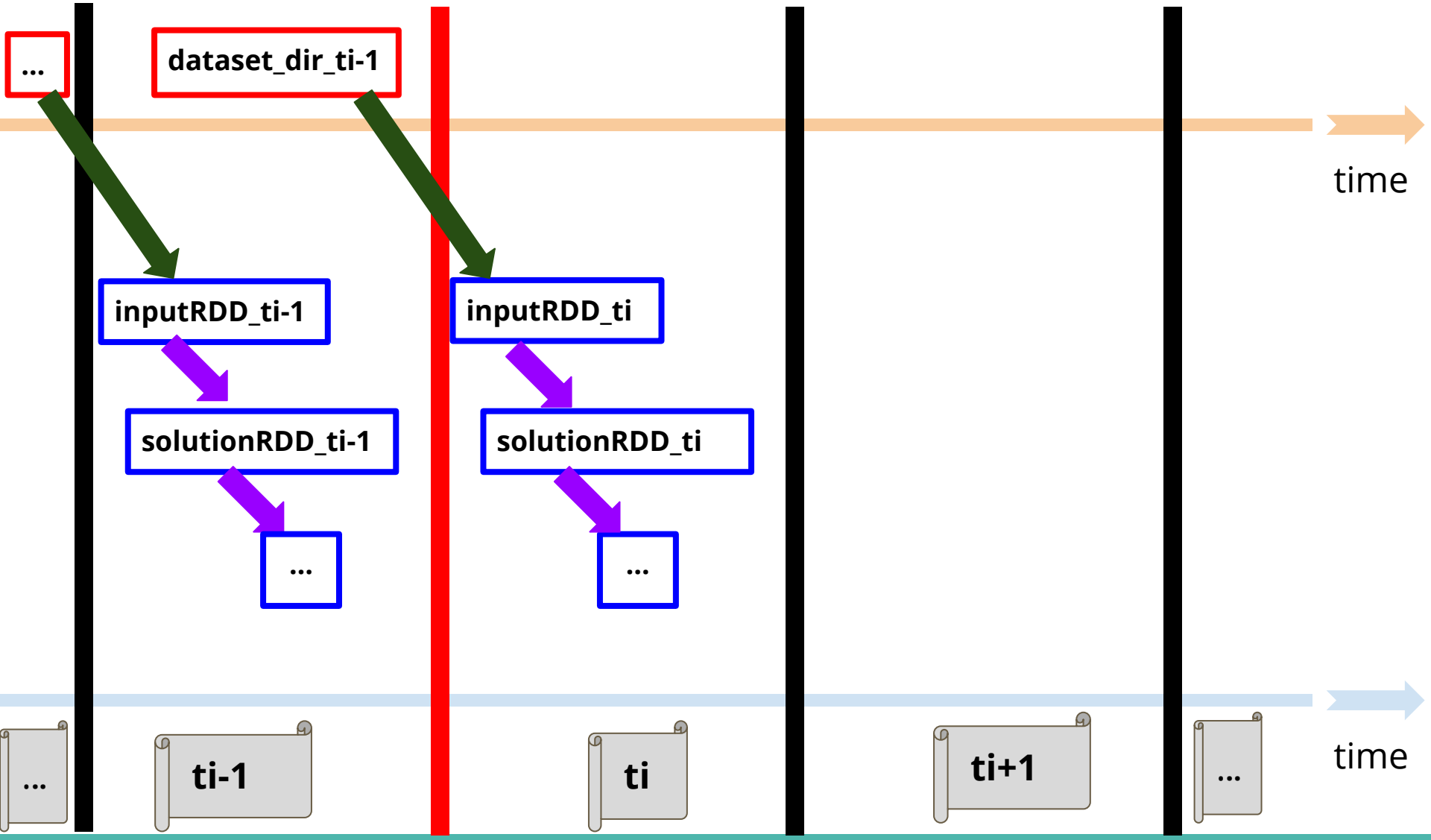
Concept2: Append Mode



Concept2: Append Mode



Concept2: Append Mode



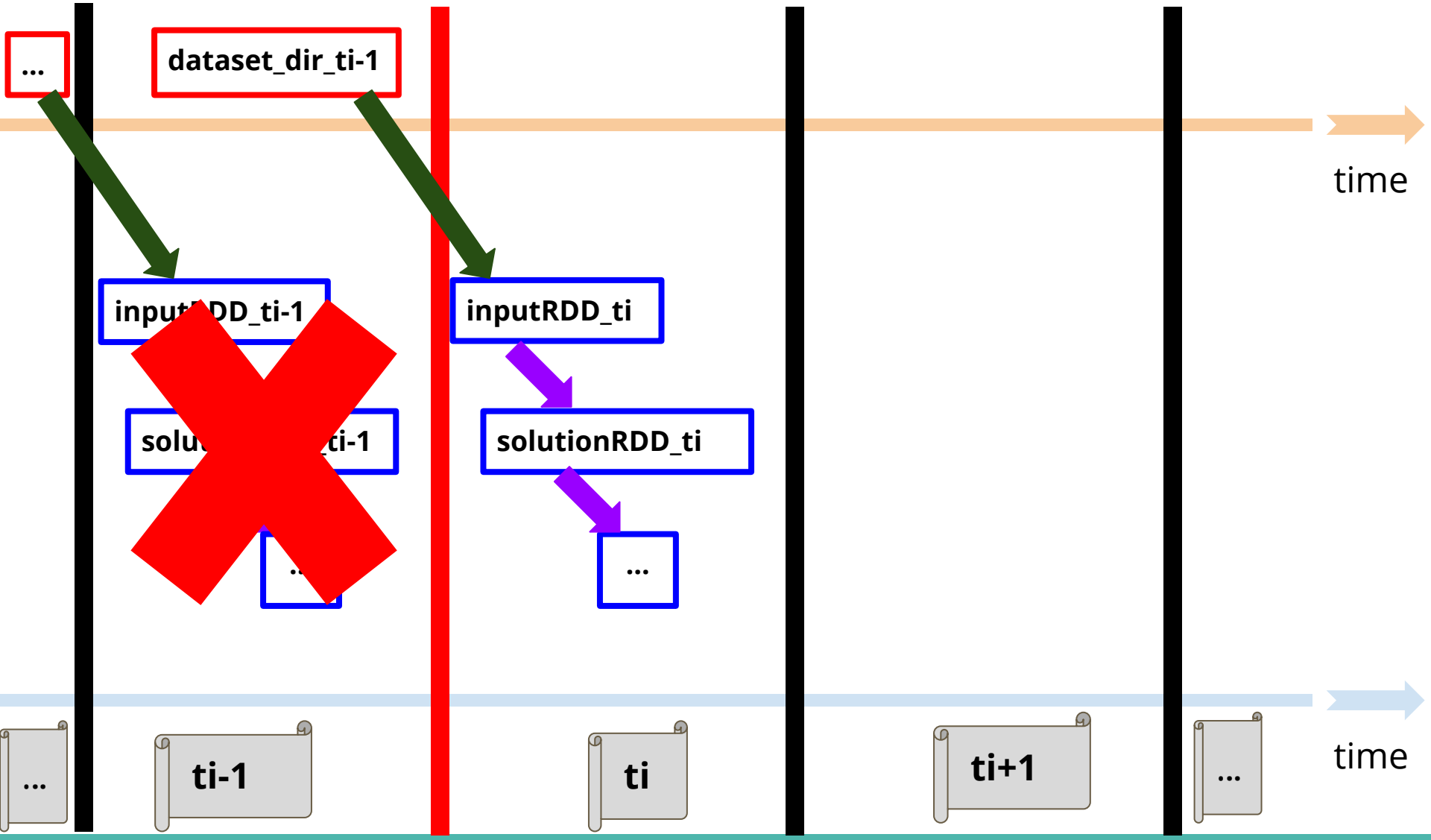
Concept2: Append Mode

A **DStream** does not increase its number of wagons (**RDDs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!

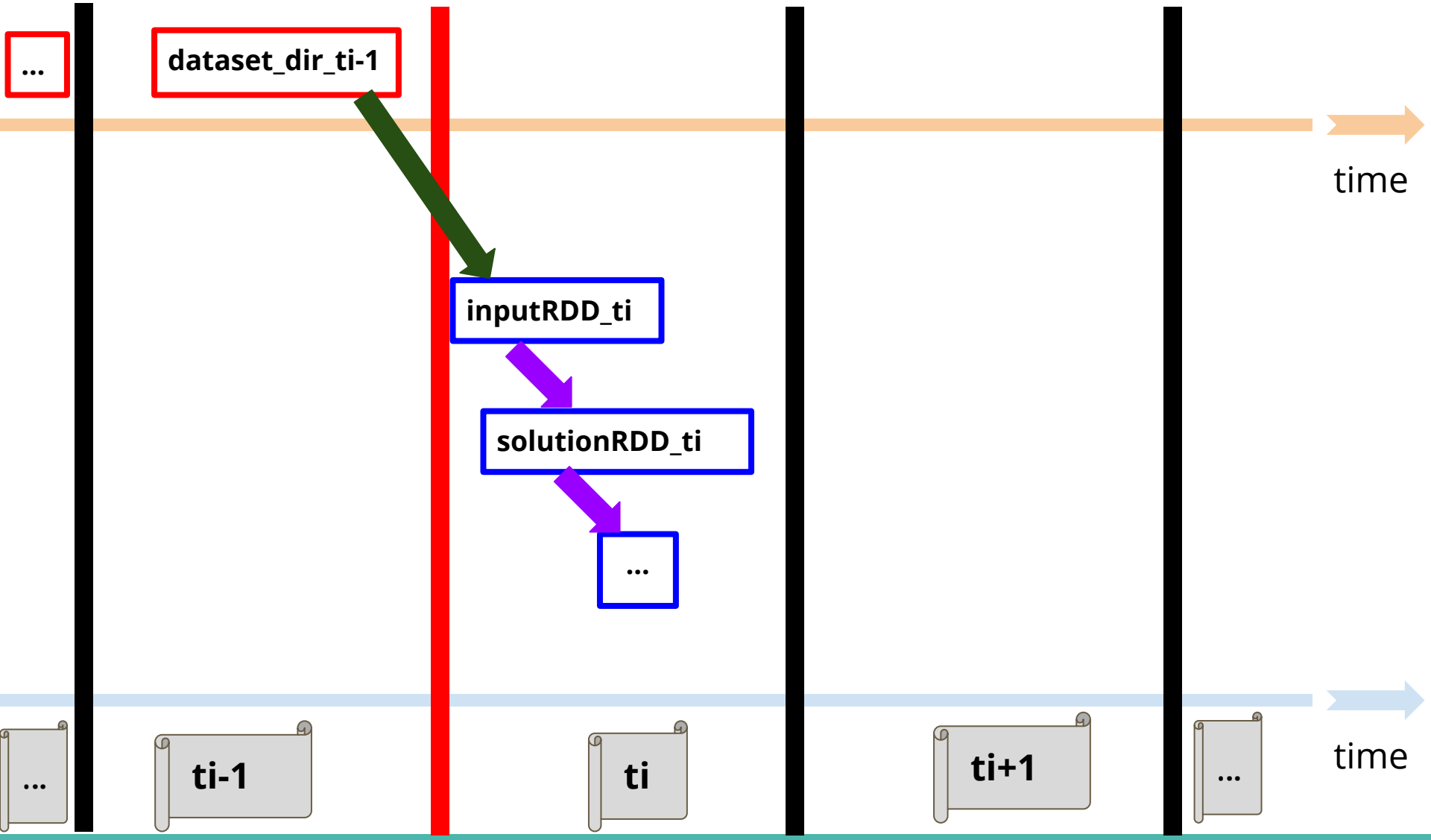
Concept2: Append Mode

A **DStream** does not increase its number of wagons (**RDDs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!

Concept2: Append Mode

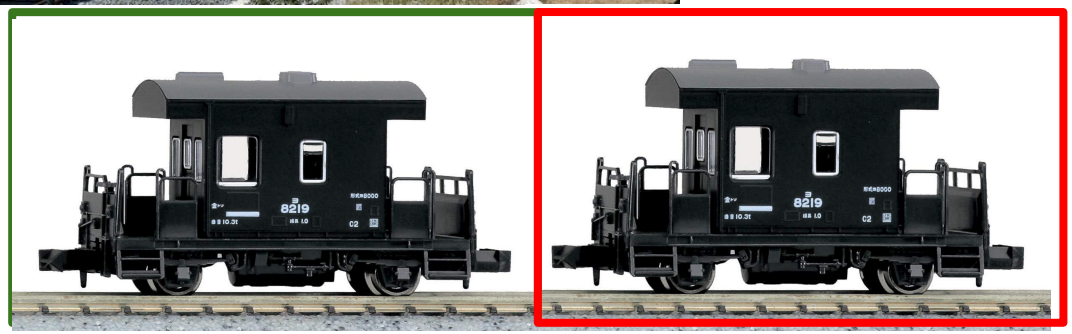


Concept2: Append Mode

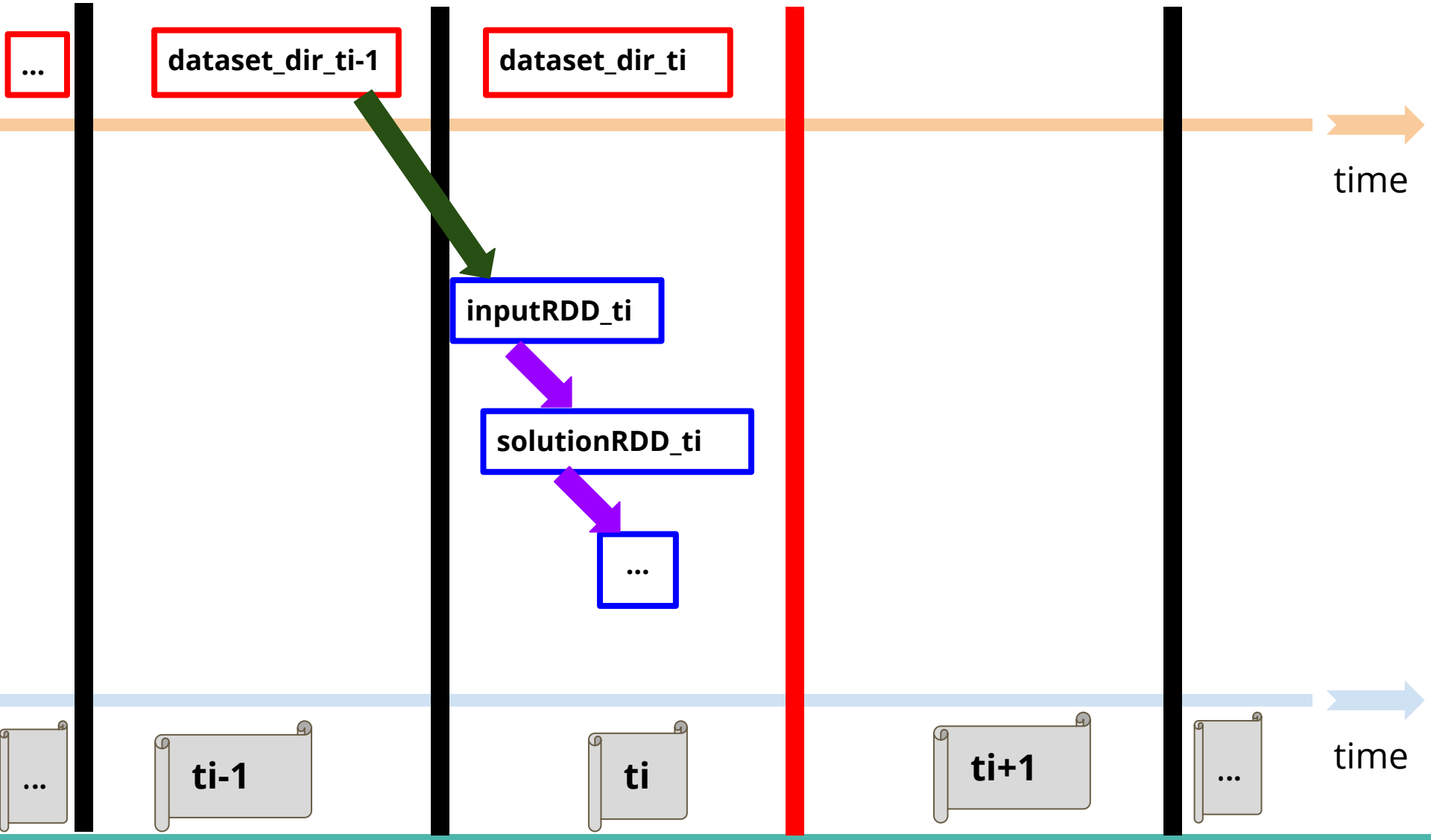


Concept2: Append Mode

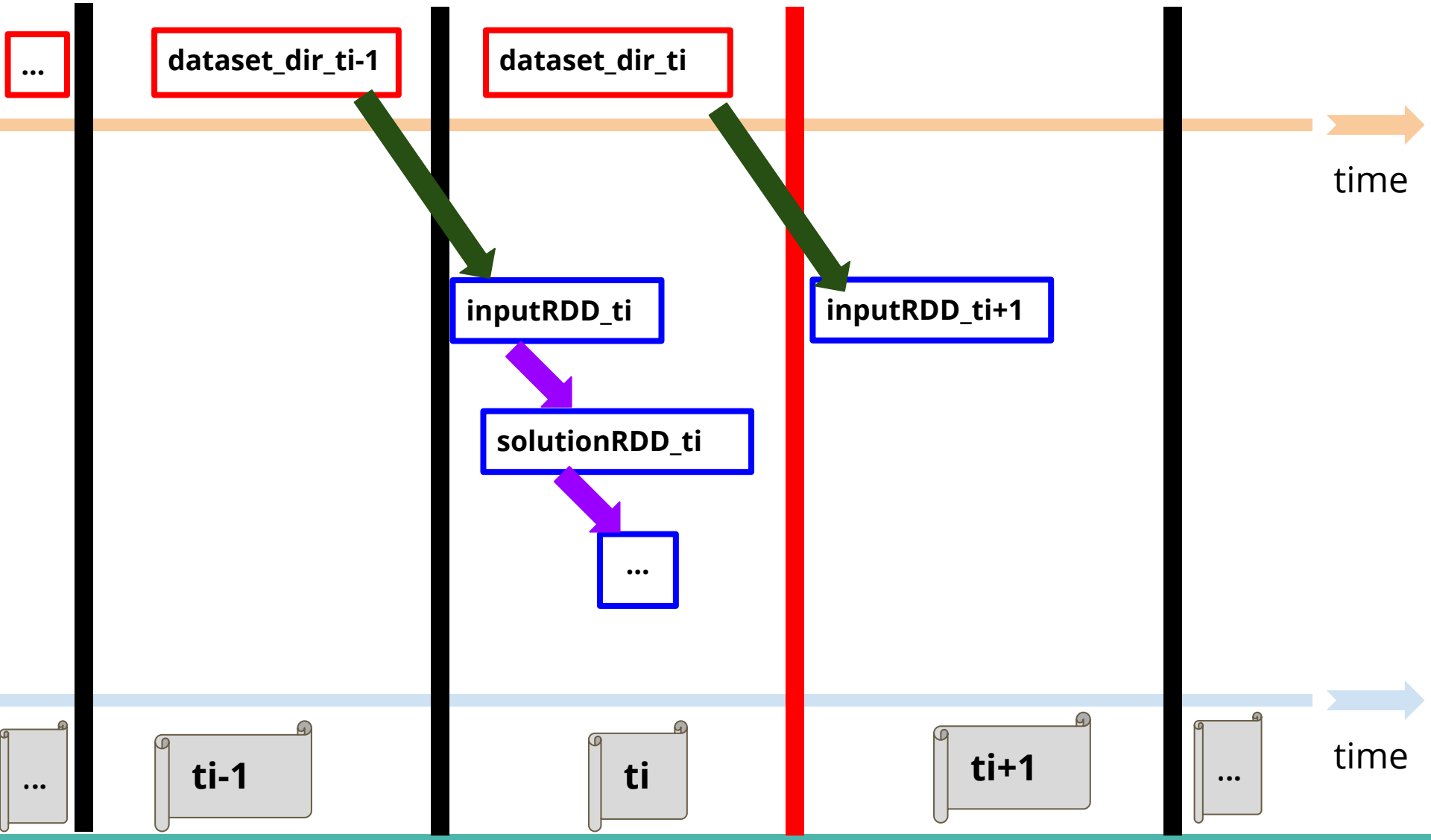
A **DStream** does not increase its number of wagons (**RDDs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!



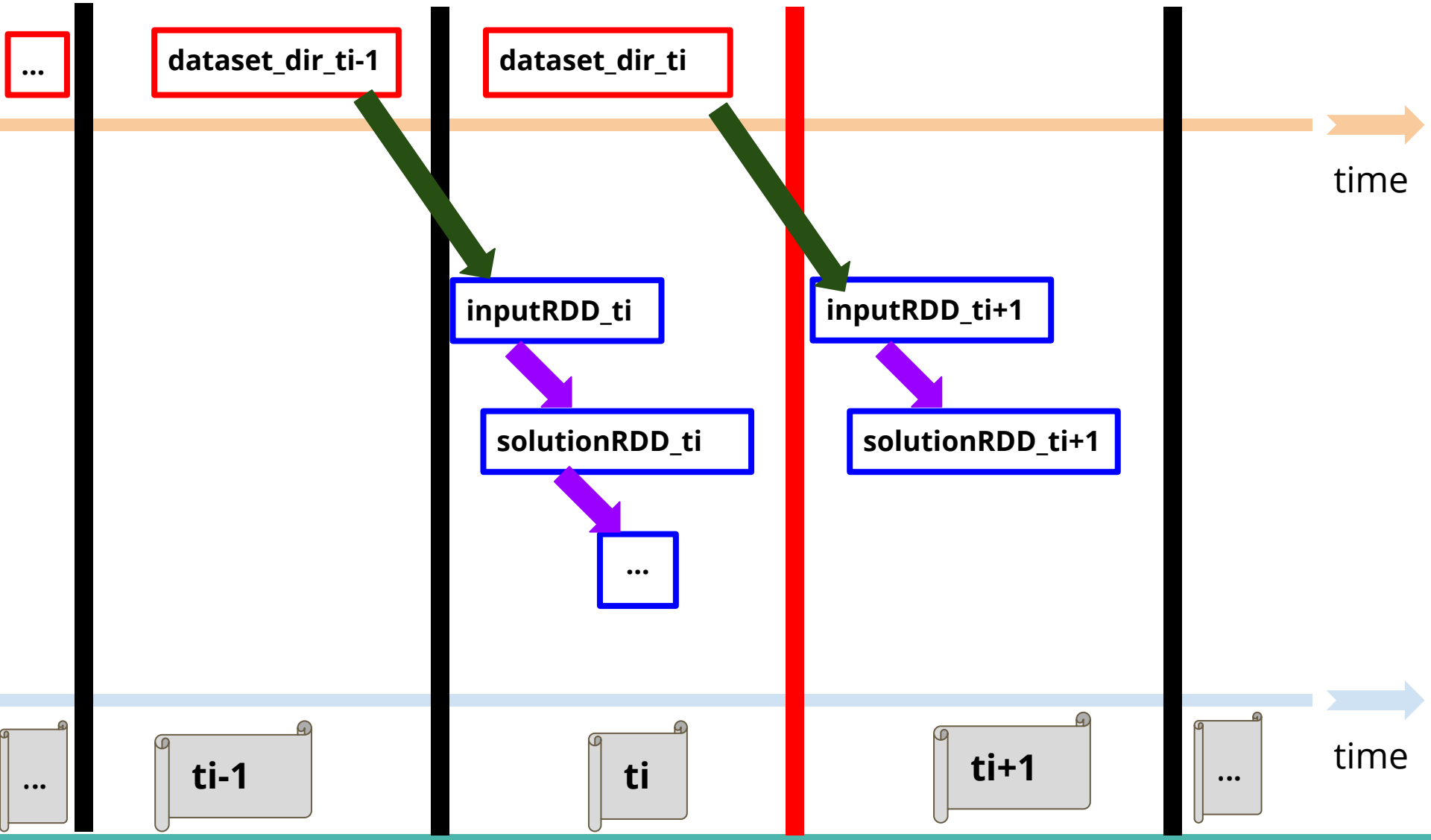
Concept2: Append Mode



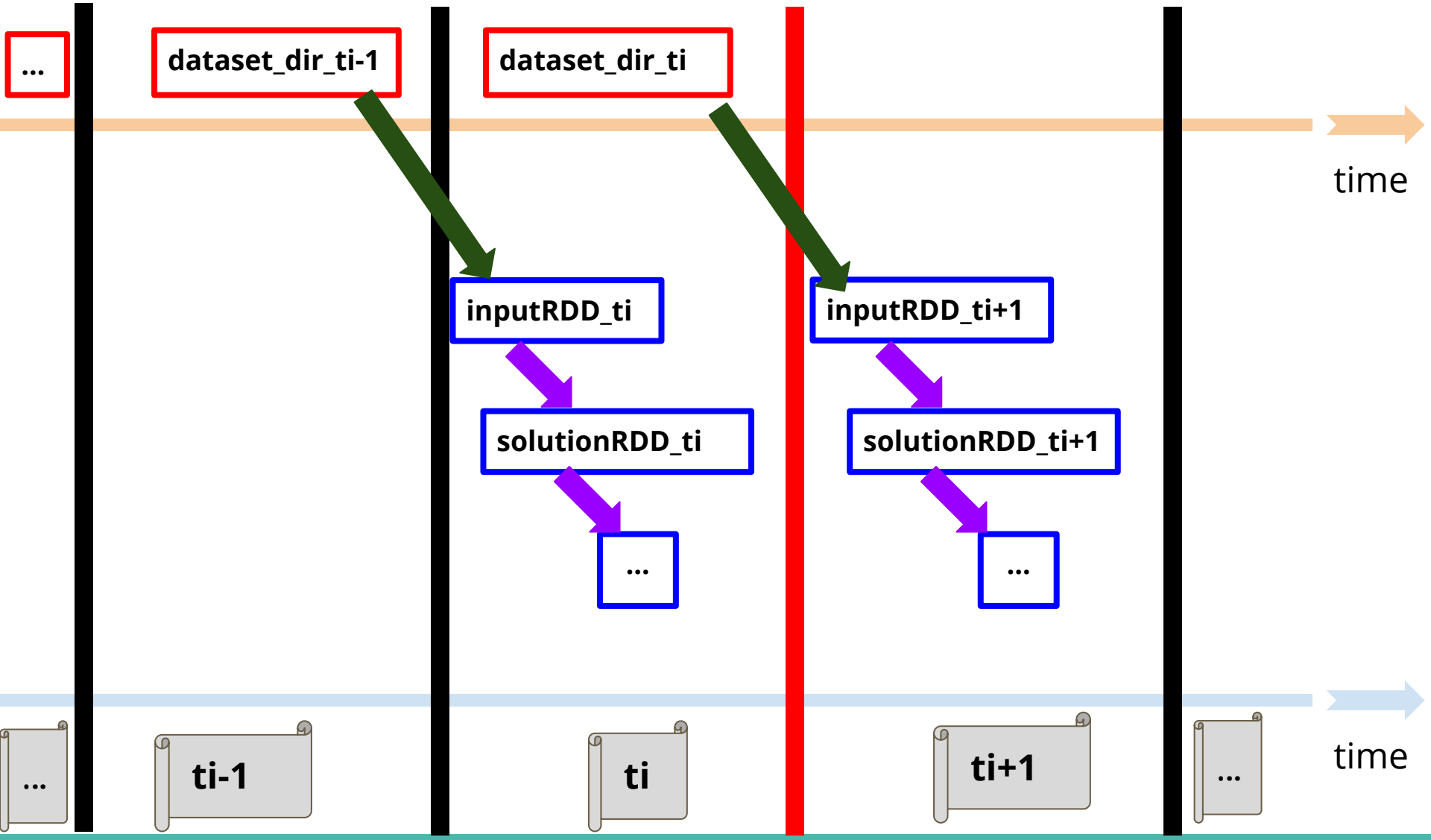
Concept2: Append Mode



Concept2: Append Mode



Concept2: Append Mode



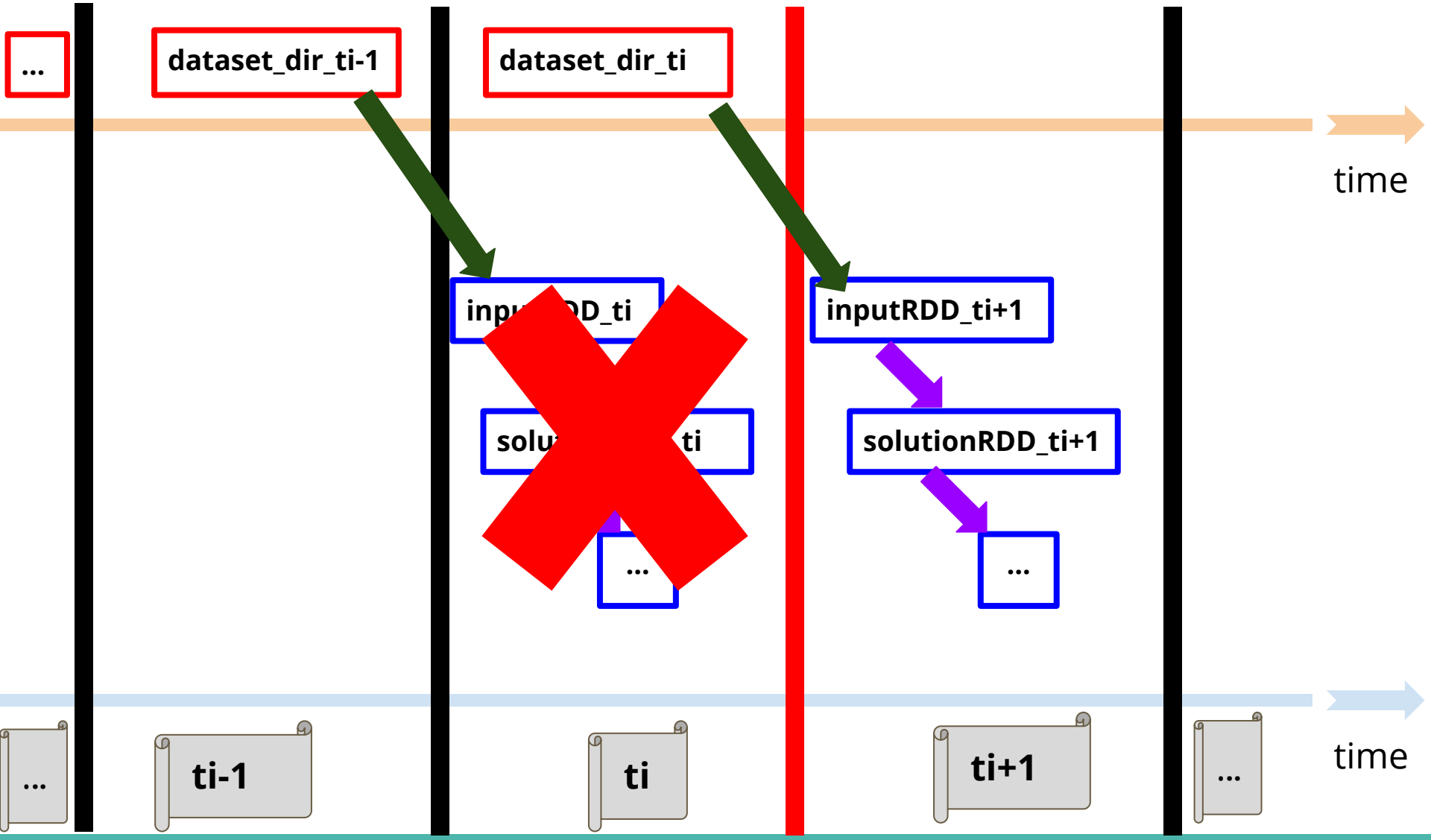
Concept2: Append Mode

A **DStream** does not increase its number of wagons (**RDDs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!

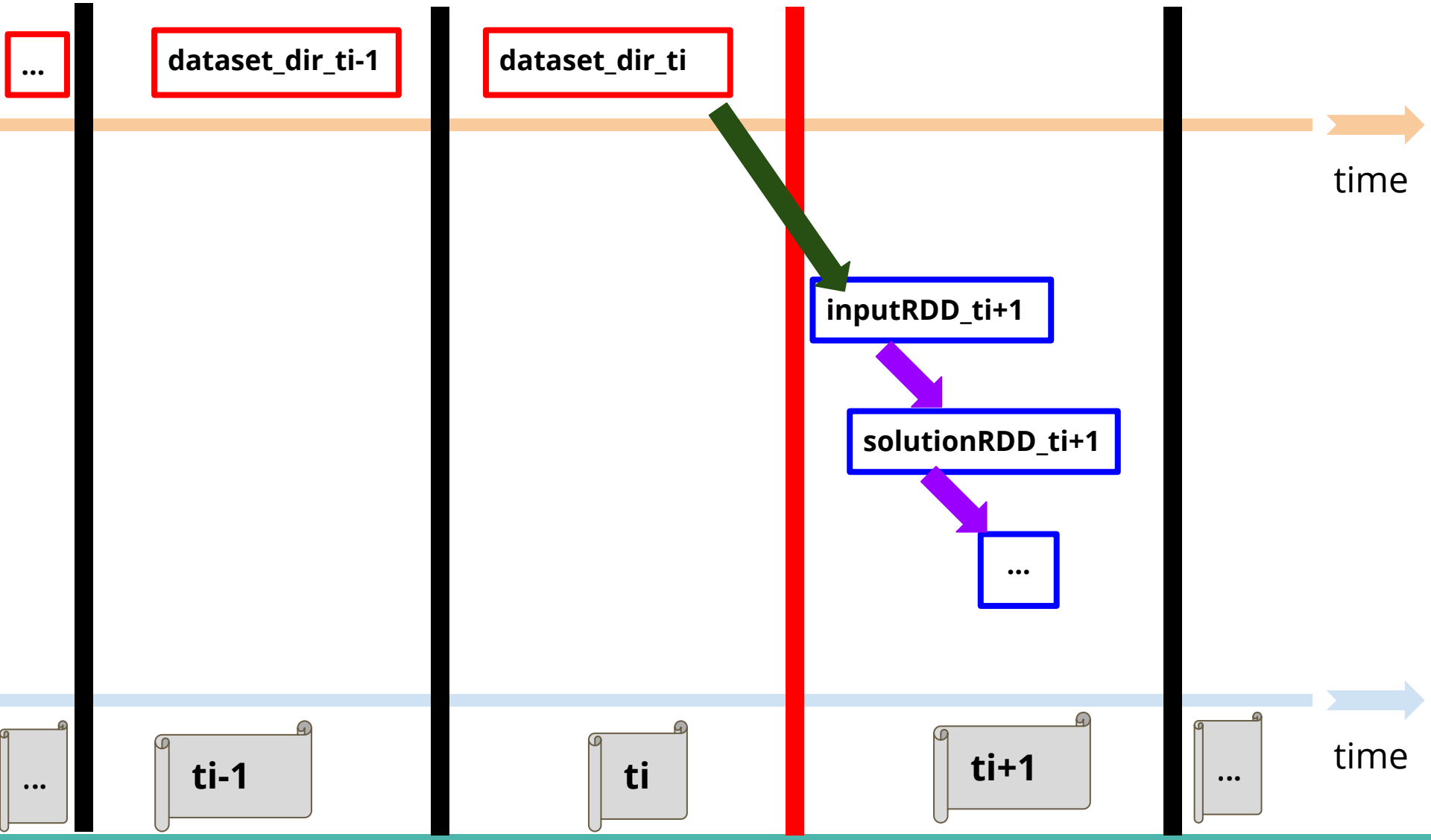
Concept2: Append Mode

A **DStream** <u>does not increase</u> its number of wagons (**RDDs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!

Concept2: Append Mode



Concept2: Append Mode



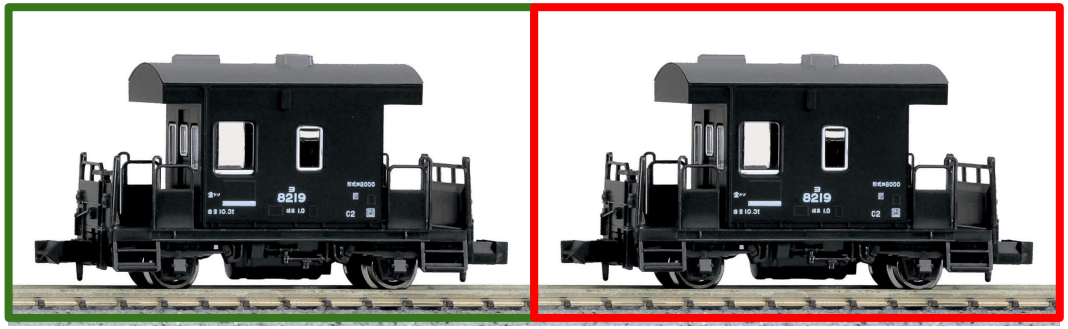
Concept2: Append Mode

The same behaviour applies to **SDF**!

Concept2: Append Mode

The same behaviour applies to SDF!

It does not increase its number of wagons (**DFs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!



Concept2: Append Mode

The same behaviour applies to SDF!

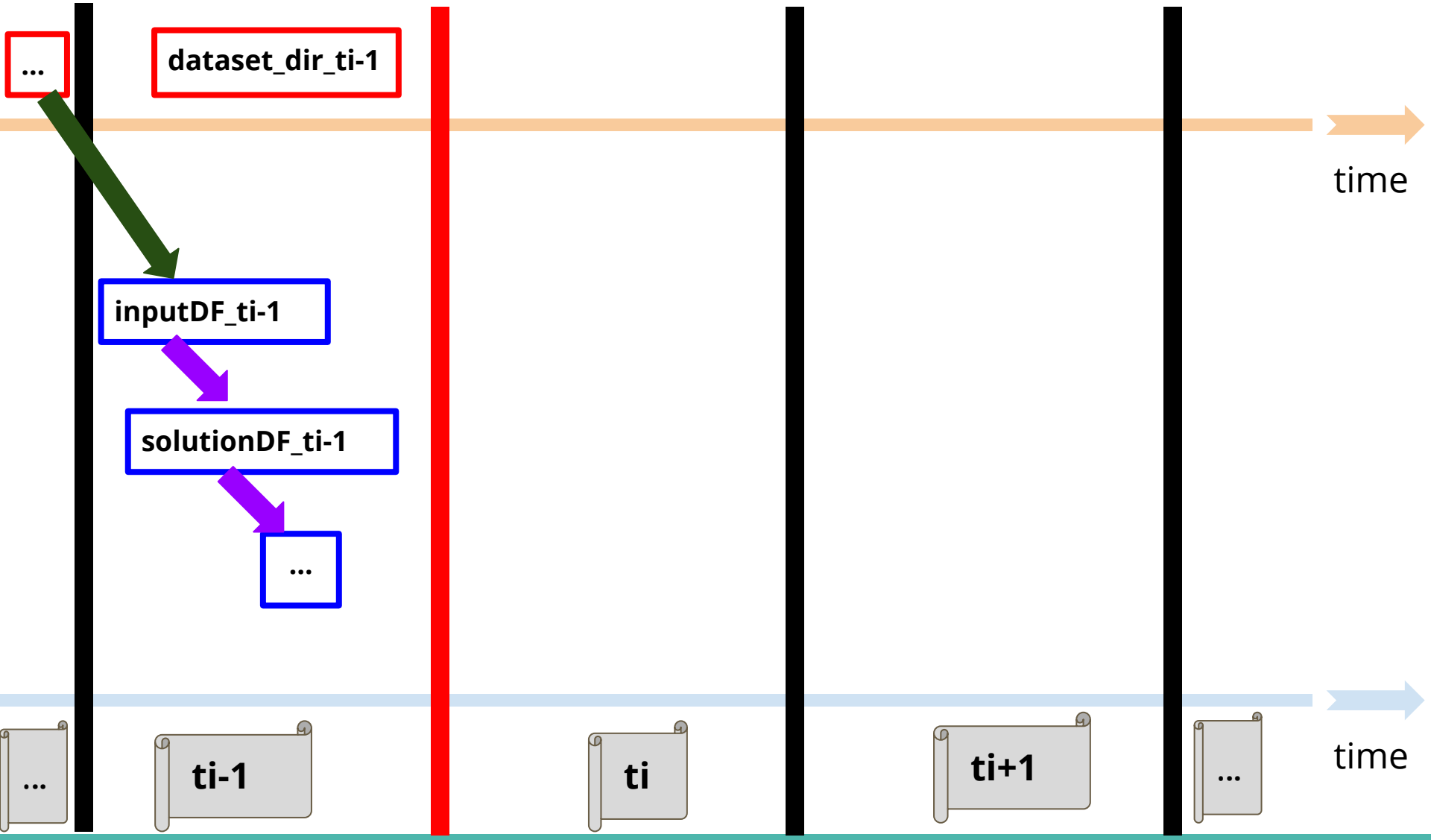
It does not increase its number of wagons (**DFs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!

Concept2: Append Mode

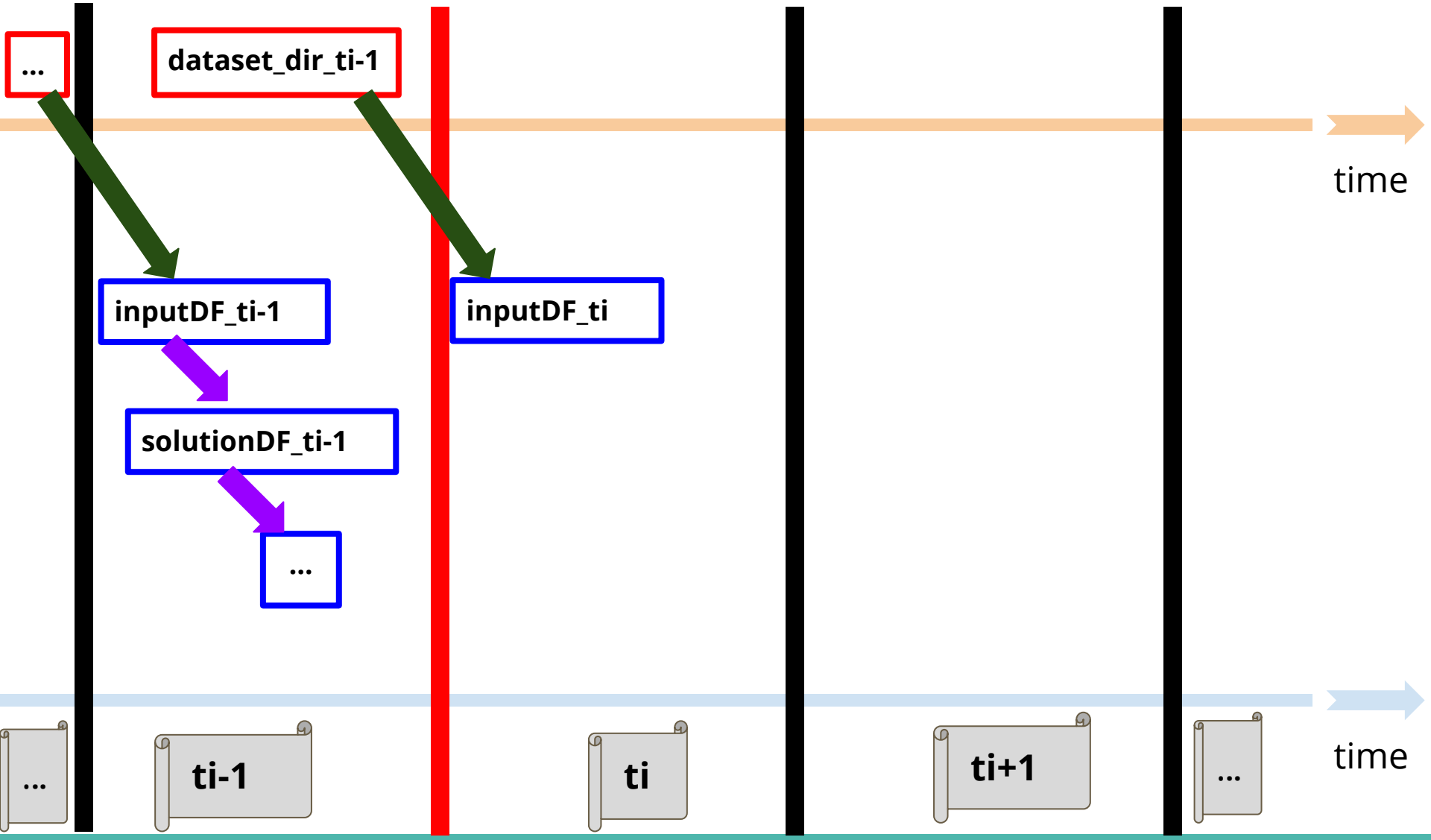
The same behaviour applies to SDF!

It does not increase its number of wagons (**DFs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!

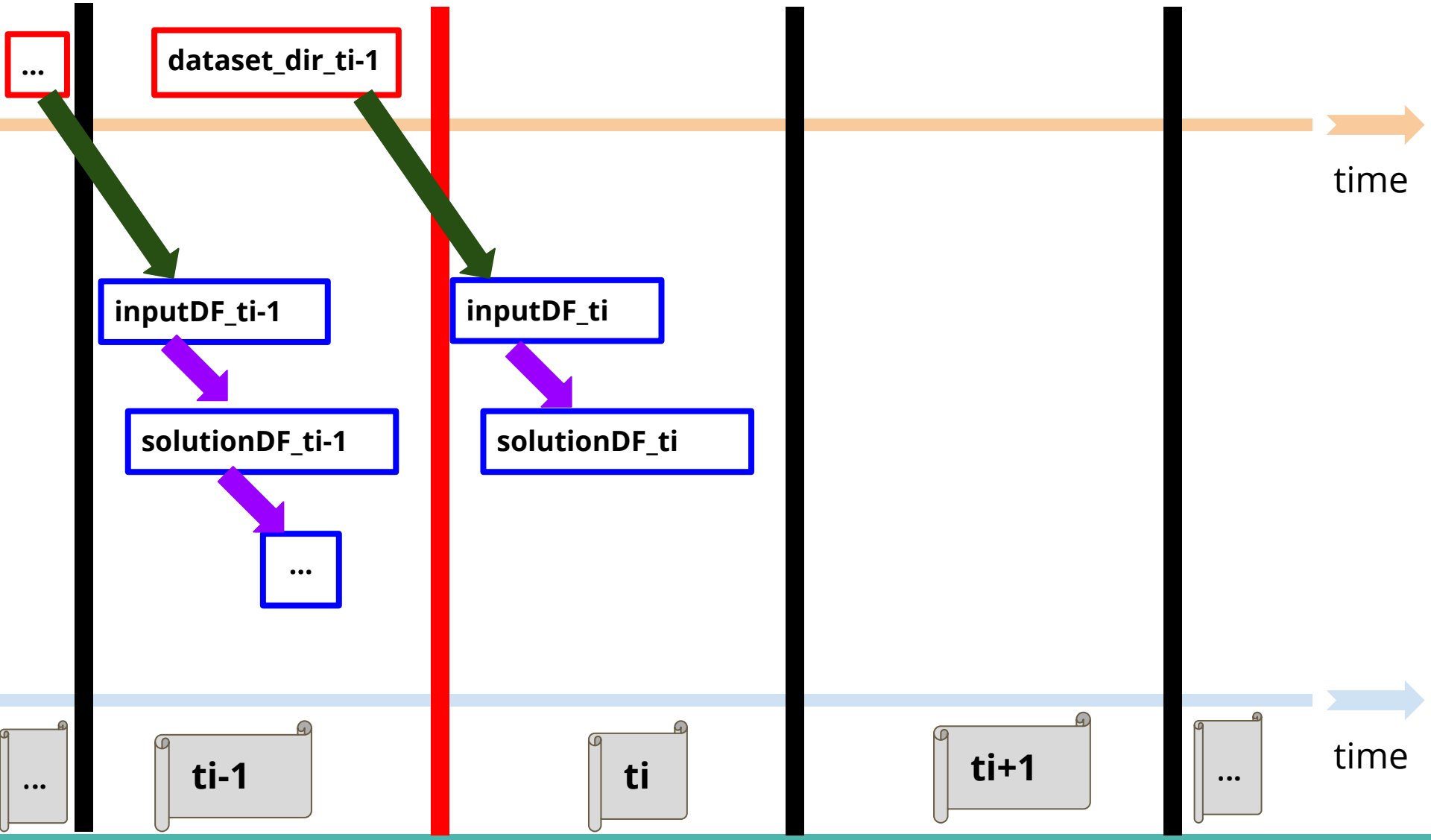
Concept2: Append Mode



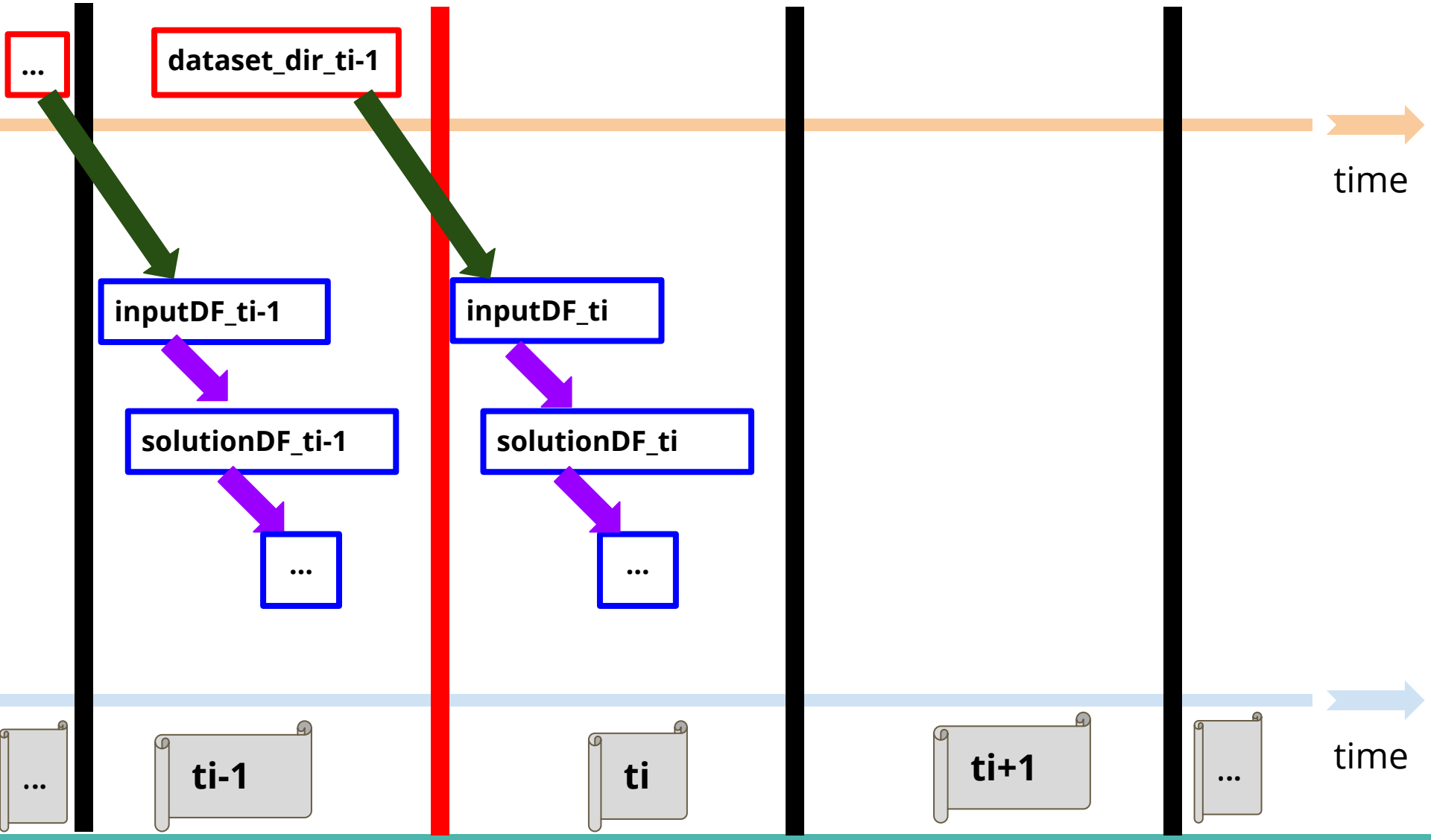
Concept2: Append Mode



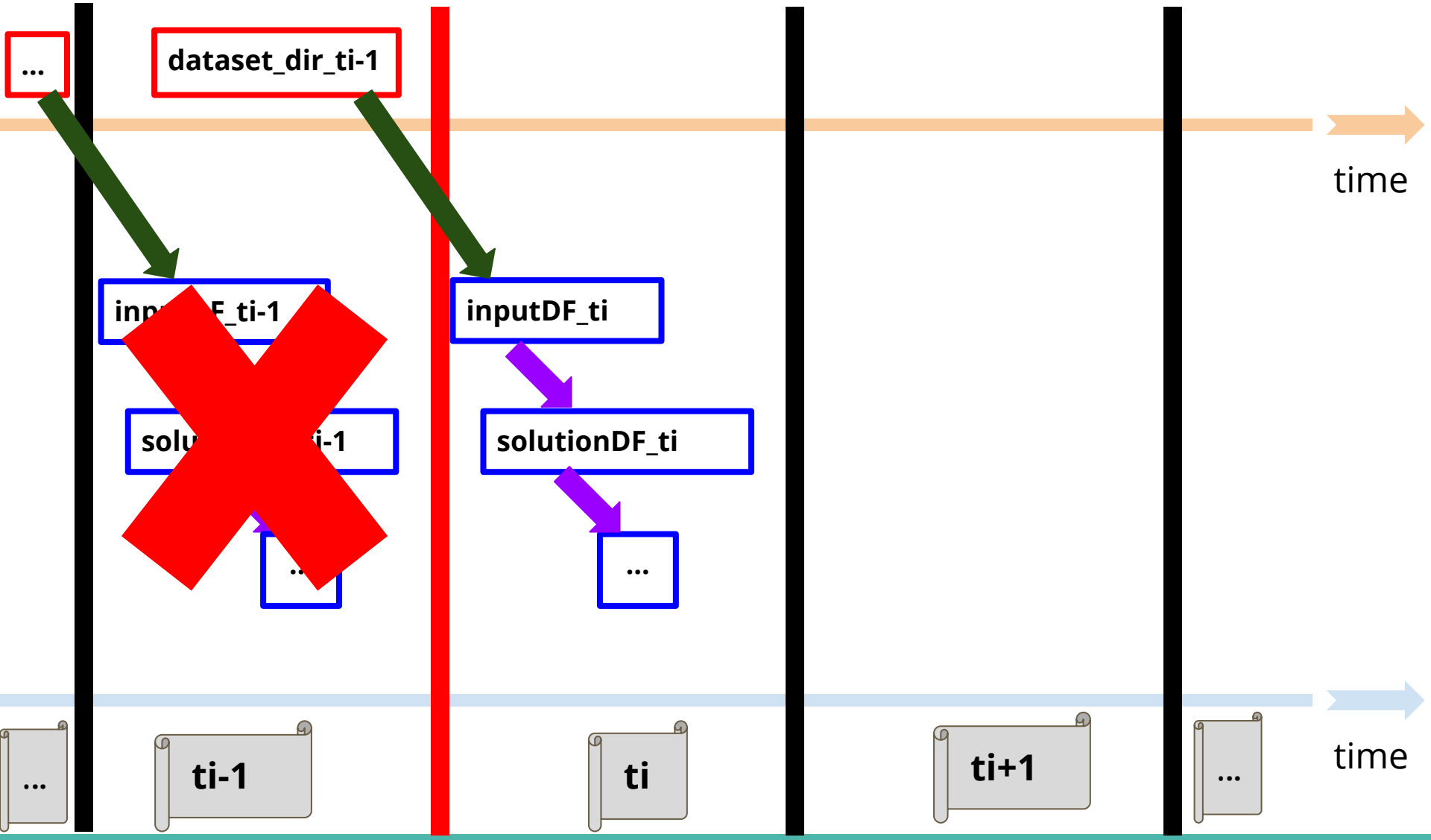
Concept2: Append Mode



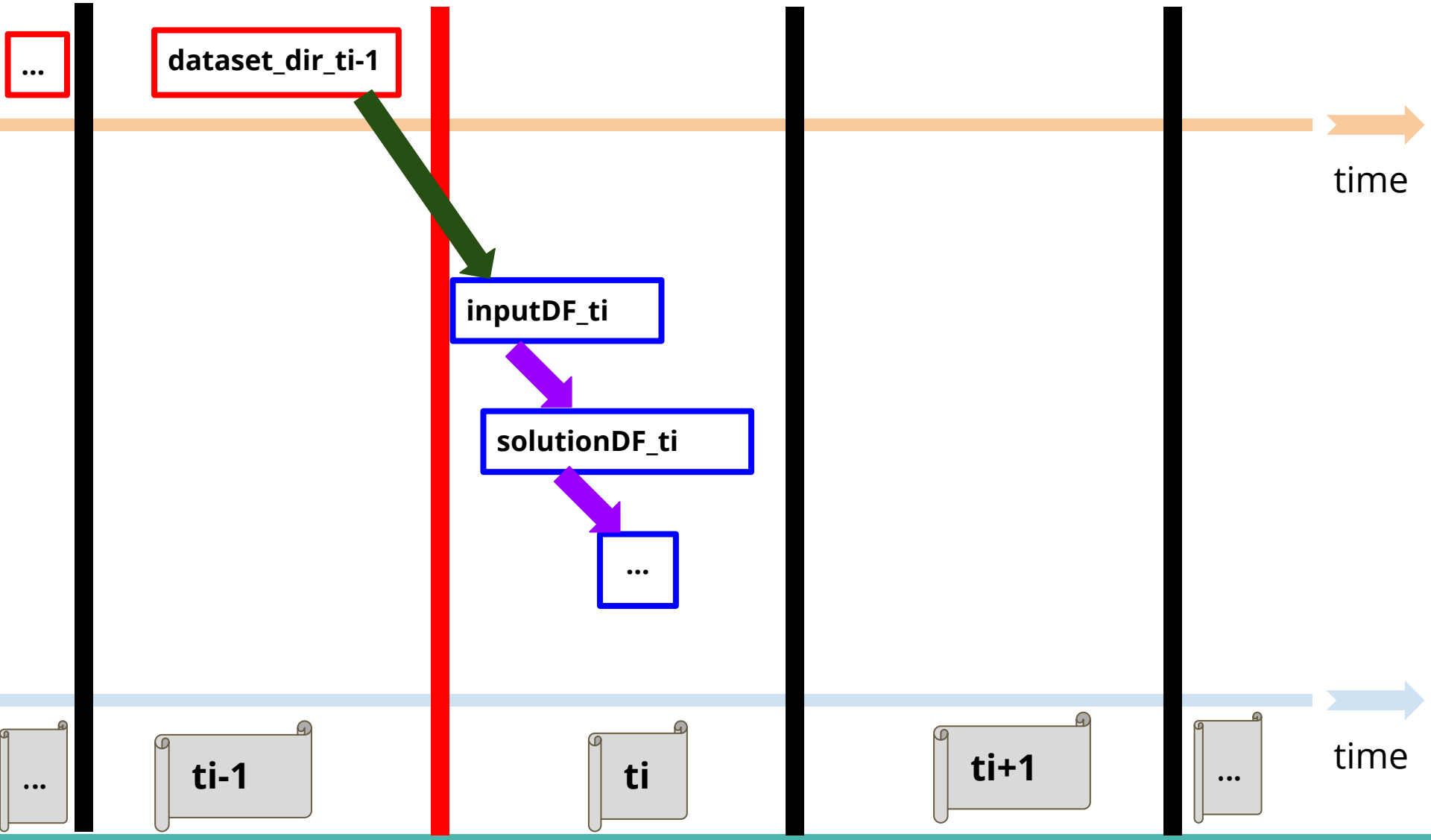
Concept2: Append Mode



Concept2: Append Mode



Concept2: Append Mode



Concept2: Append Mode

And with the unbound table representation of the SDF...

inputSDF

	Name	City	Eyes
Row1	...	...	...
Row2	...	...	...

inputDF_ti-1

Concept2: Append Mode

And with the unbound table representation of the SDF...

inputSDF

	Name	City	Eyes
Row1	...	...	...
Row2	...	...	...

inputDF_ti-1

	Name	City	Eyes
Row3	...	...	...

inputDF_ti

Concept2: Append Mode

And with the unbound table representation of the SDF...

inputSDF

	Name	City	Eyes
Row1	...	...	...
Row2	...	...	...

inputDF_ti-1

	Name	City	Eyes
Row3	...	...	...

inputDF_ti

Concept2: Append Mode

And with the unbound table representation of the SDF...

inputSDF

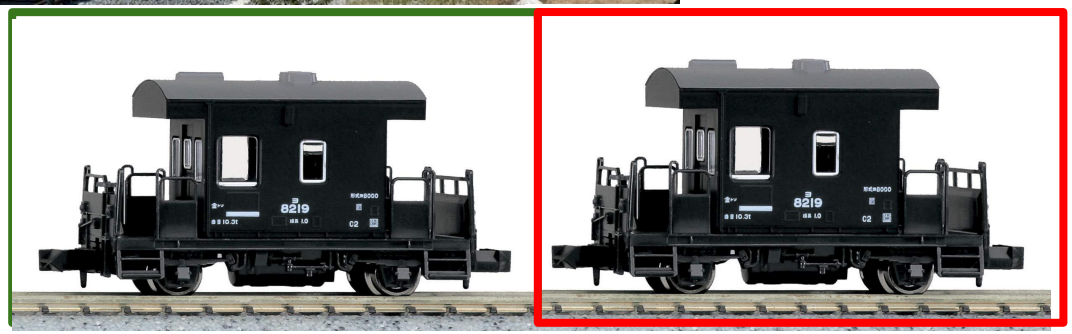
	Name	City	Eyes
Row3	...	...	...

inputDF_ti

Concept2: Append Mode

The same behaviour applies to SDF!

It does not increase its number of wagons (**DFs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!



Concept2: Append Mode

The same behaviour applies to SDF!

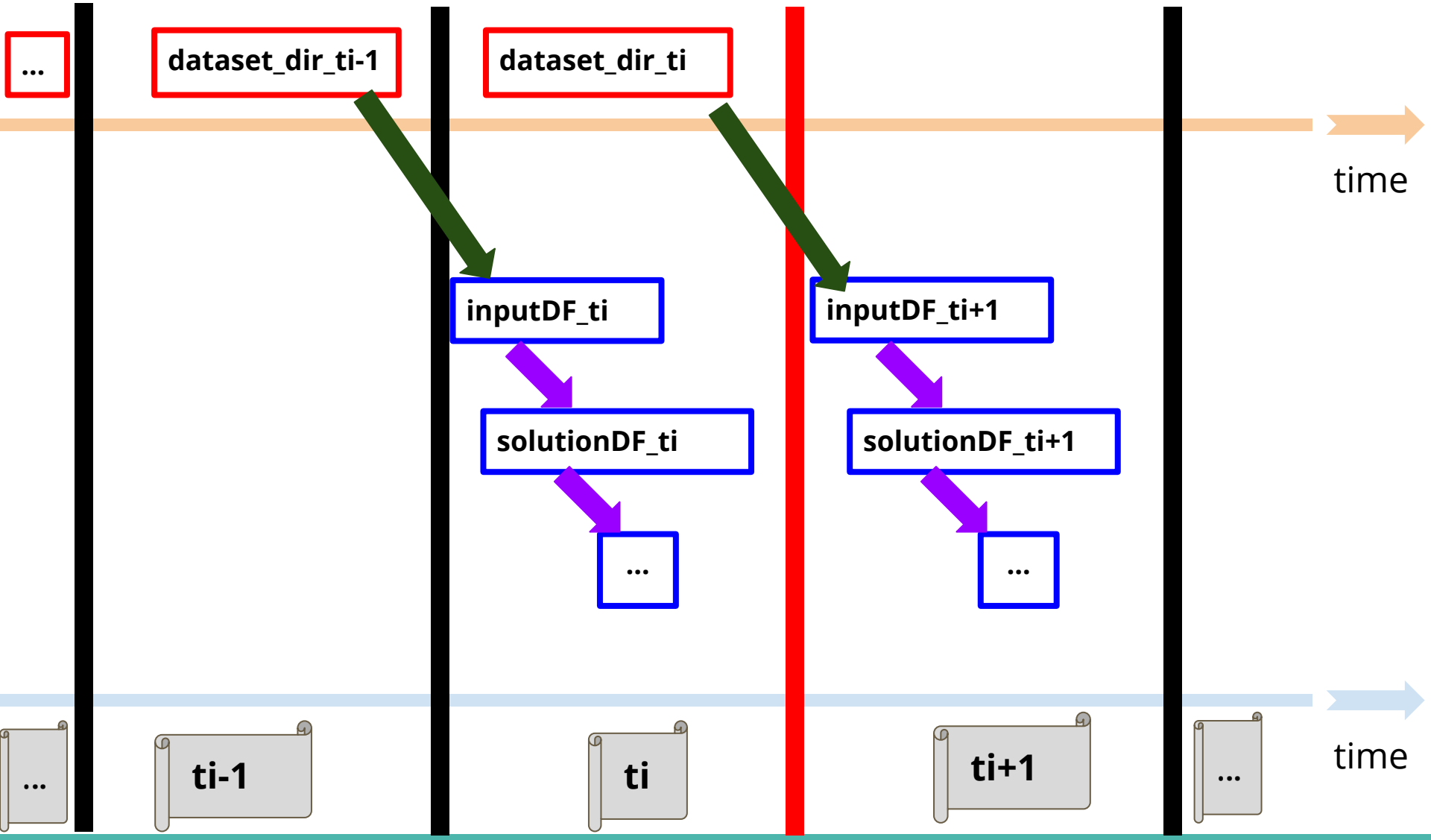
It does not increase its number of wagons (**DFs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!

Concept2: Append Mode

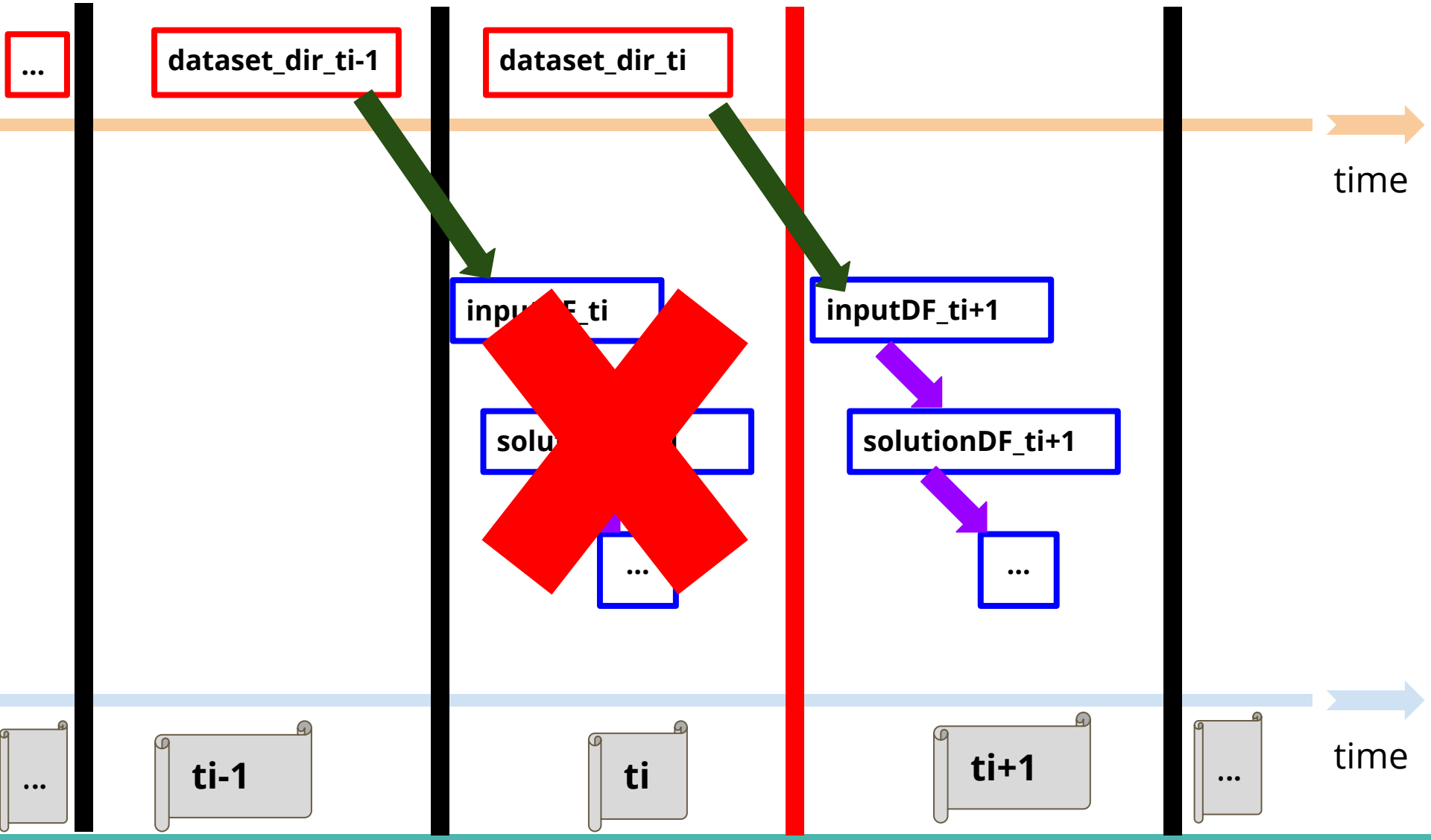
The same behaviour applies to SDF!

It does not increase its number of wagons (**DFs)** over time...
...as each time a new wagon comes in...
...the current last wagon leaves!

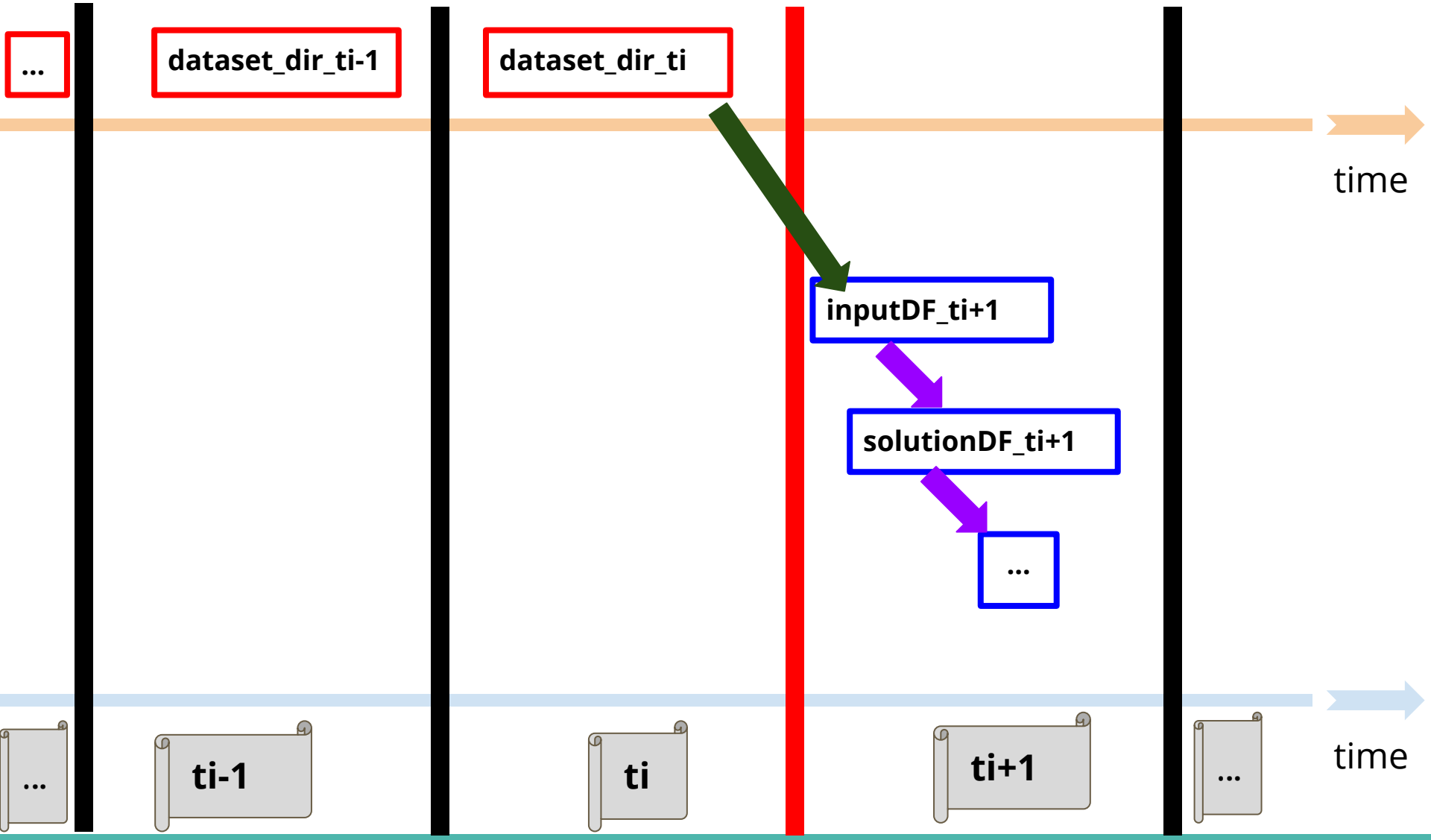
Concept2: Append Mode



Concept2: Append Mode



Concept2: Append Mode



Concept2: Append Mode

And with the unbound table representation of the SDF...

inputSDF

	Name	City	Eyes
Row3	...	...	...

inputDF_ti

Concept2: Append Mode

And with the unbound table representation of the SDF...

inputSDF

	Name	City	Eyes
Row3	...	...	...

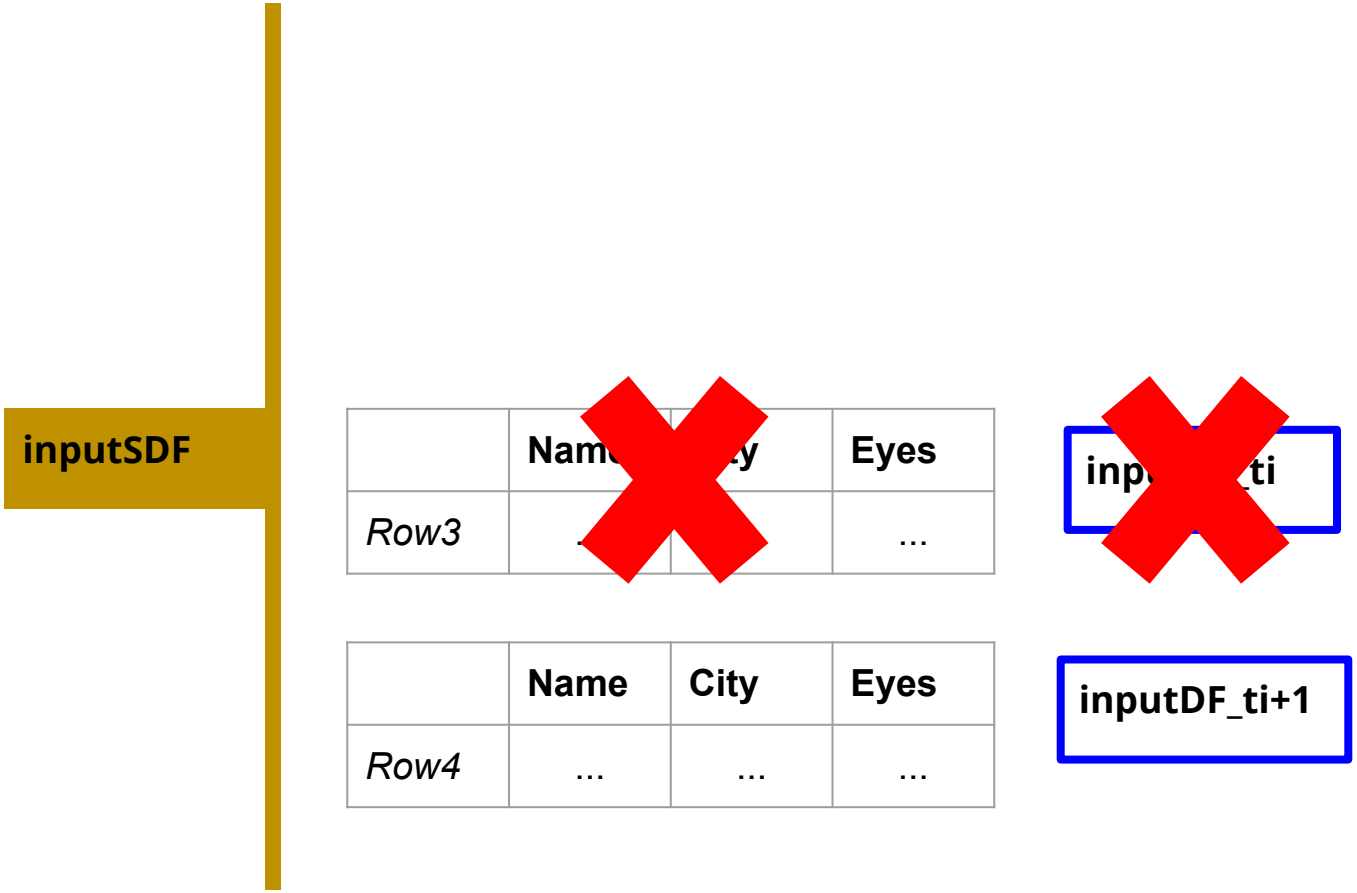
inputDF_ti

	Name	City	Eyes
Row4	...	...	...

inputDF_ti+1

Concept2: Append Mode

And with the unbound table representation of the SDF...



Concept2: Append Mode

And with the unbound table representation of the SDF...

inputSDF

	Name	City	Eyes
Row4	...	...	...

inputDF_ti+1

Concept2: Append Mode

Concept 2:

This behaviour is called
a **SDF** in **Append Mode**!

Outline

1. Setting Up the Context.
2. From DStream to SDF.
   a. Concept1: SDF.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
3. Practising with the Concepts.

Concept3: Append Mode with Windows

A **DStream** uses window-based operations to group **wagons** together.
This automatically keeps in memory all wagons being relevant to a further window.

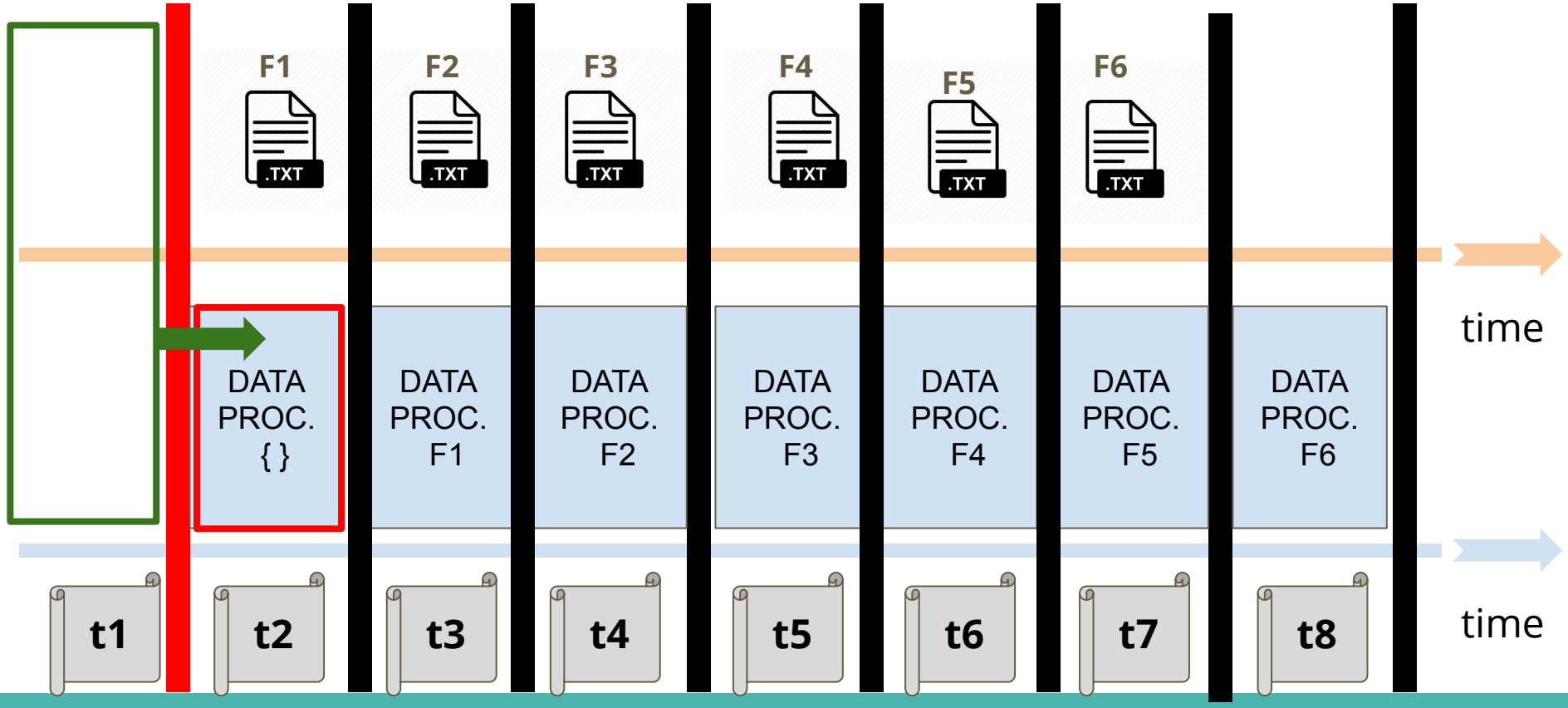
Concept3: Append Mode with Windows

A **DStream** uses window-based operations to group **wagons** together.
This automatically keeps in memory all wagons being relevant to a further window.

Concept3: Append Mode with Windows

Let's come back to our classical example.

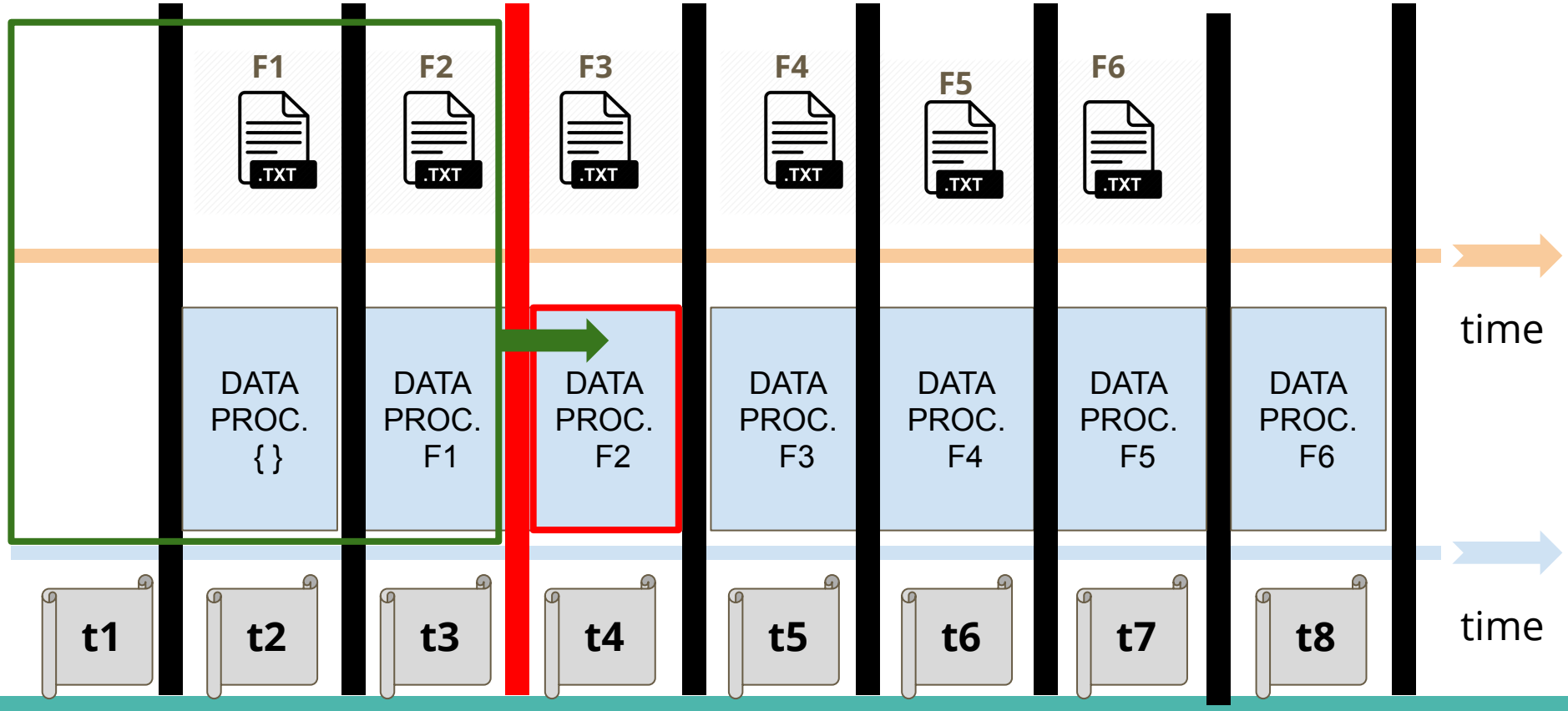
Sliding Duration = 2 and **Window Duration = 3**
we create the following windows at the following times



Concept3: Append Mode with Windows

Let's come back to our classical example.

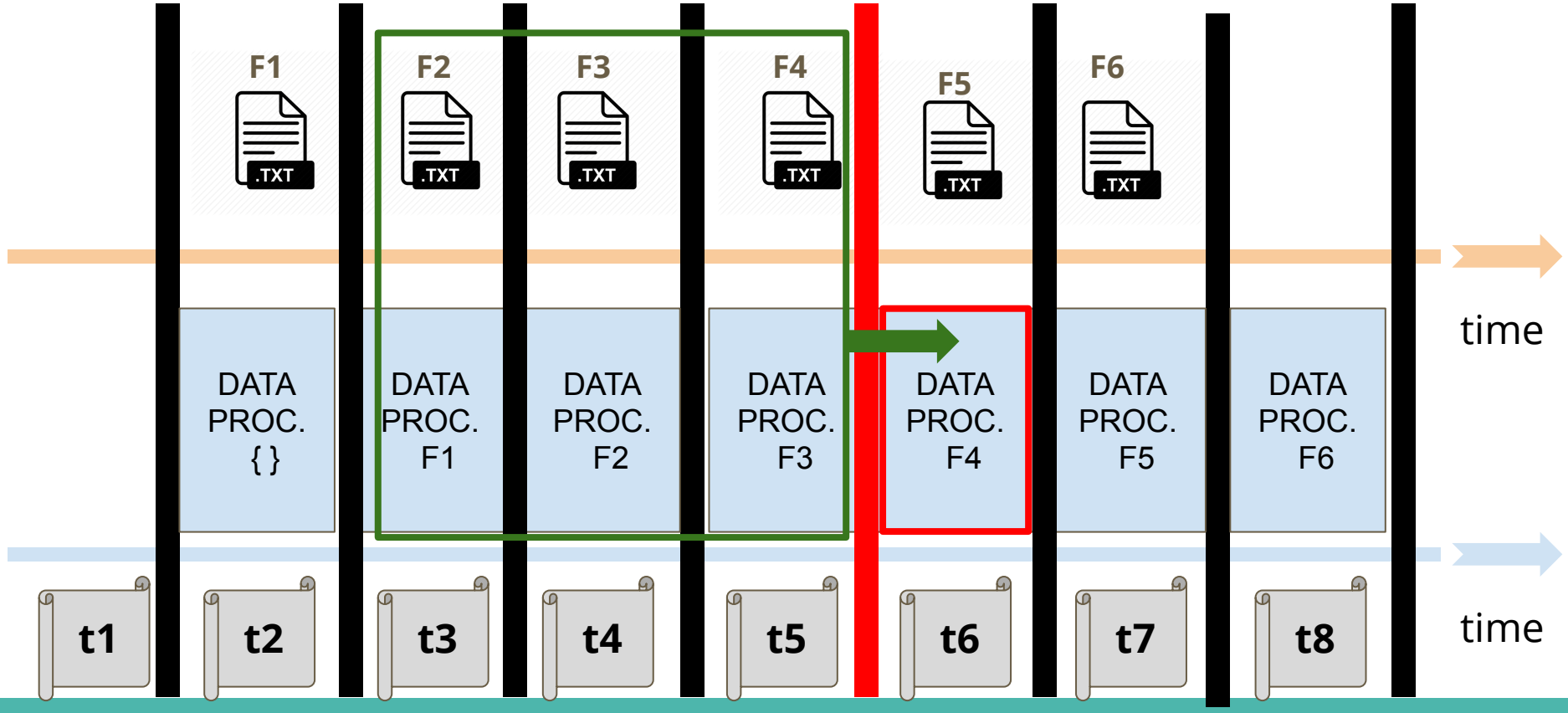
Sliding Duration = 2 and **Window Duration = 3**
we create the following windows at the following times



Concept3: Append Mode with Windows

Let's come back to our classical example.

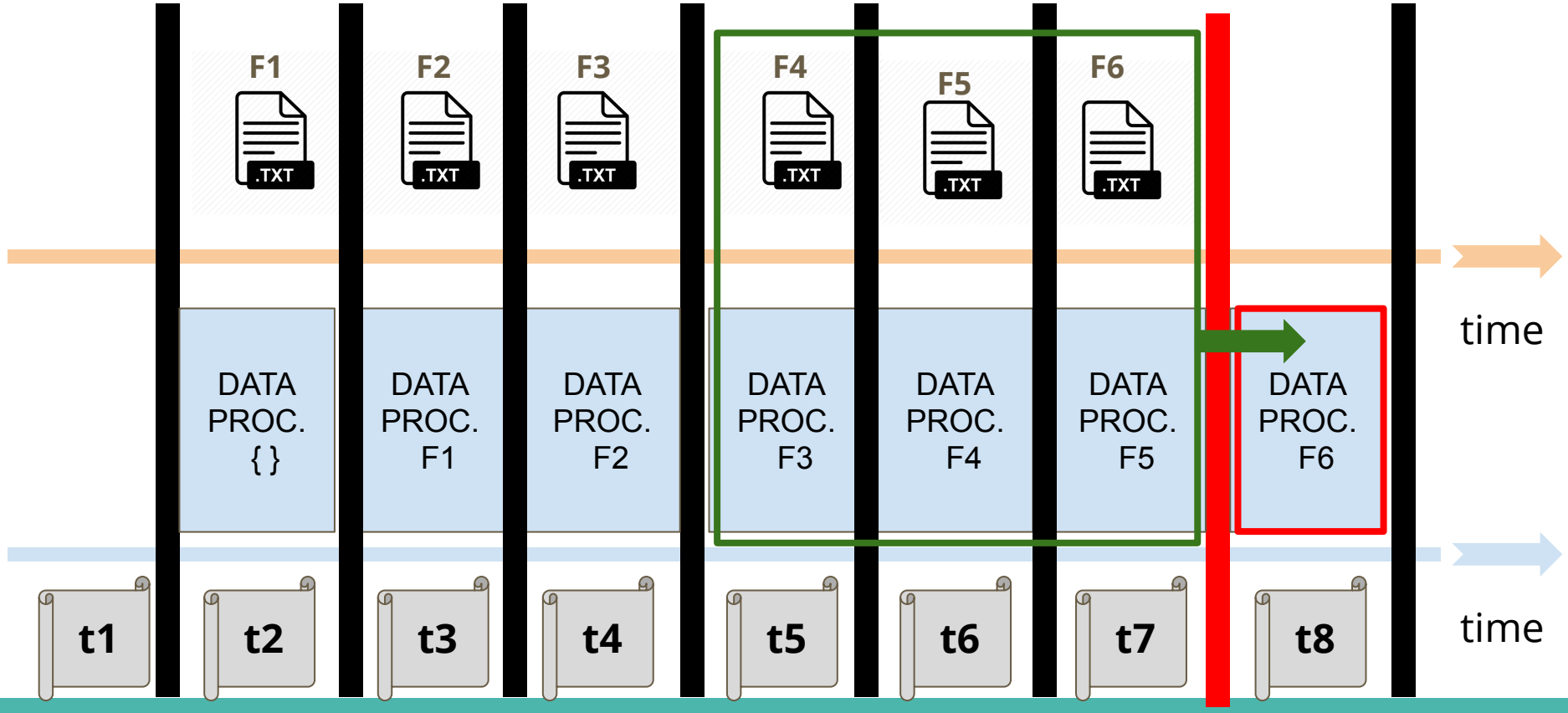
Sliding Duration = 2 and **Window Duration = 3**
we create the following windows at the following times



Concept3: Append Mode with Windows

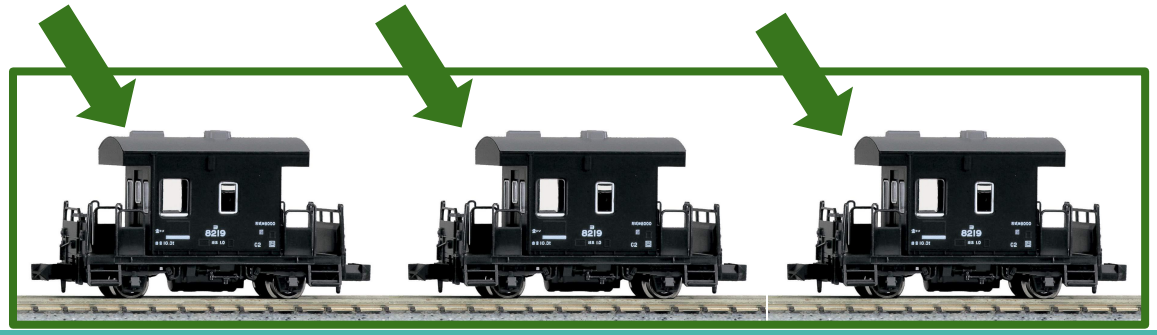
Let's come back to our classical example.

Sliding Duration = 2 and **Window Duration = 3**
we create the following windows at the following times

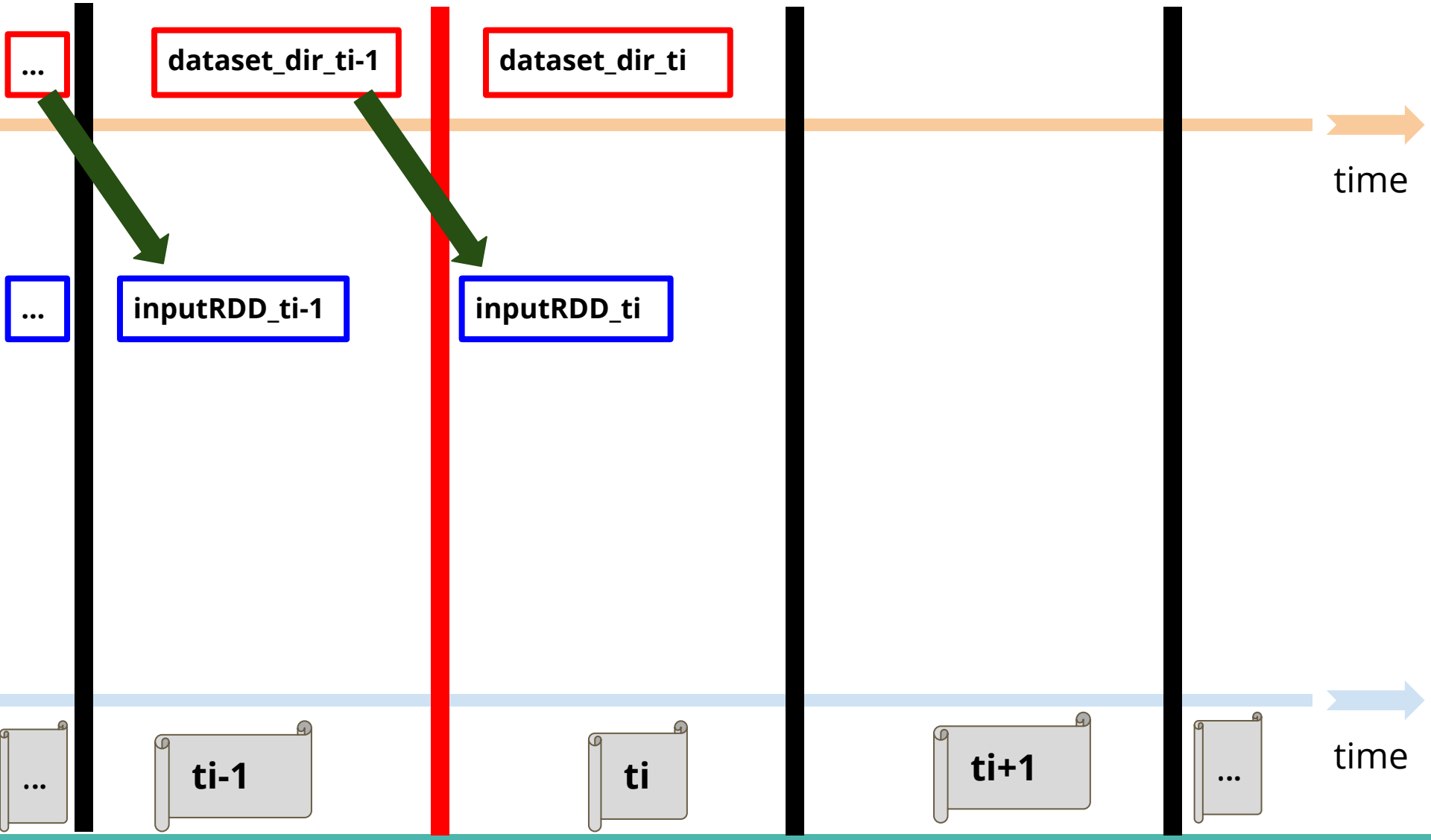


Concept3: Append Mode with Windows

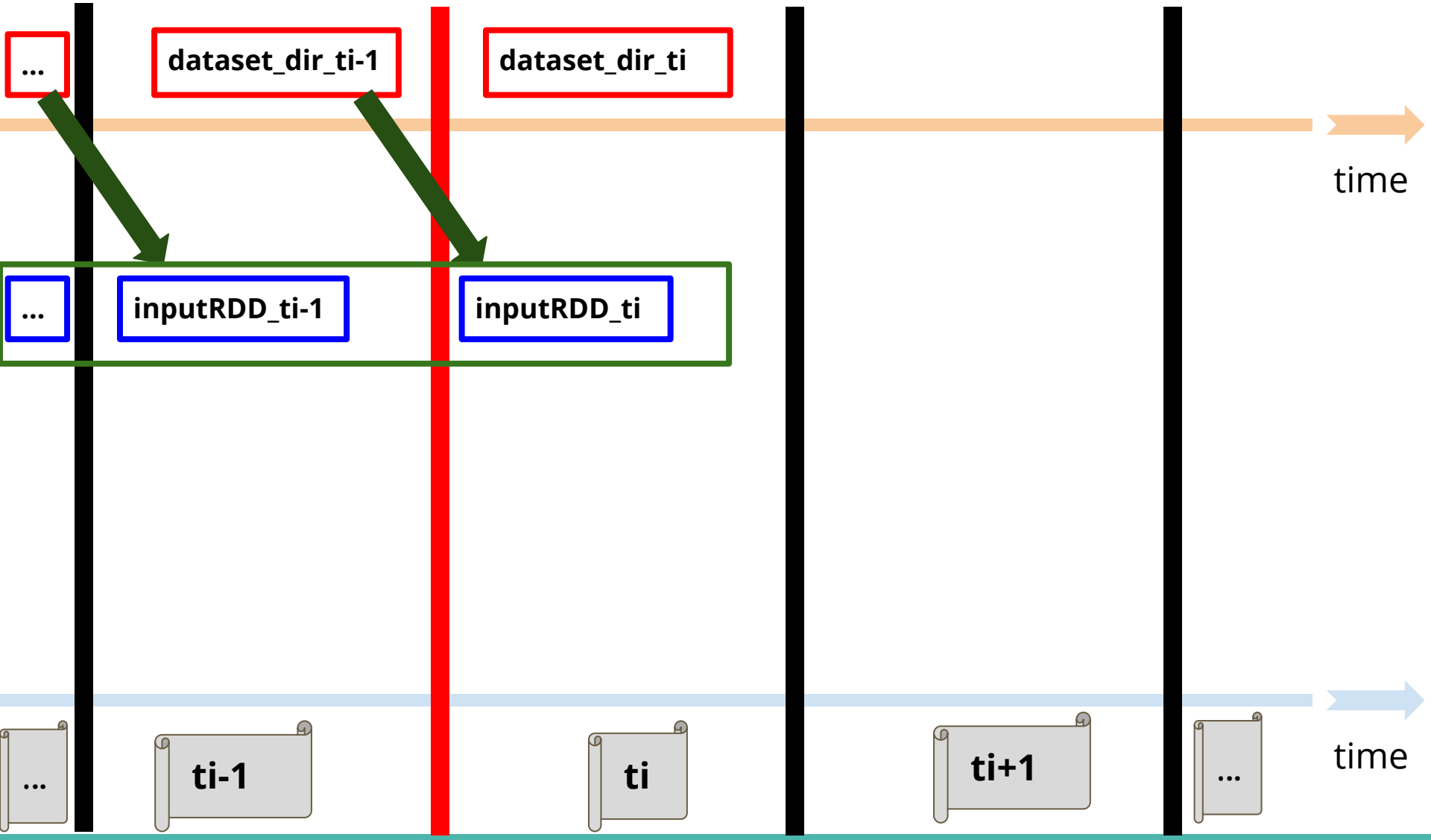
Let's come back to our classical example.



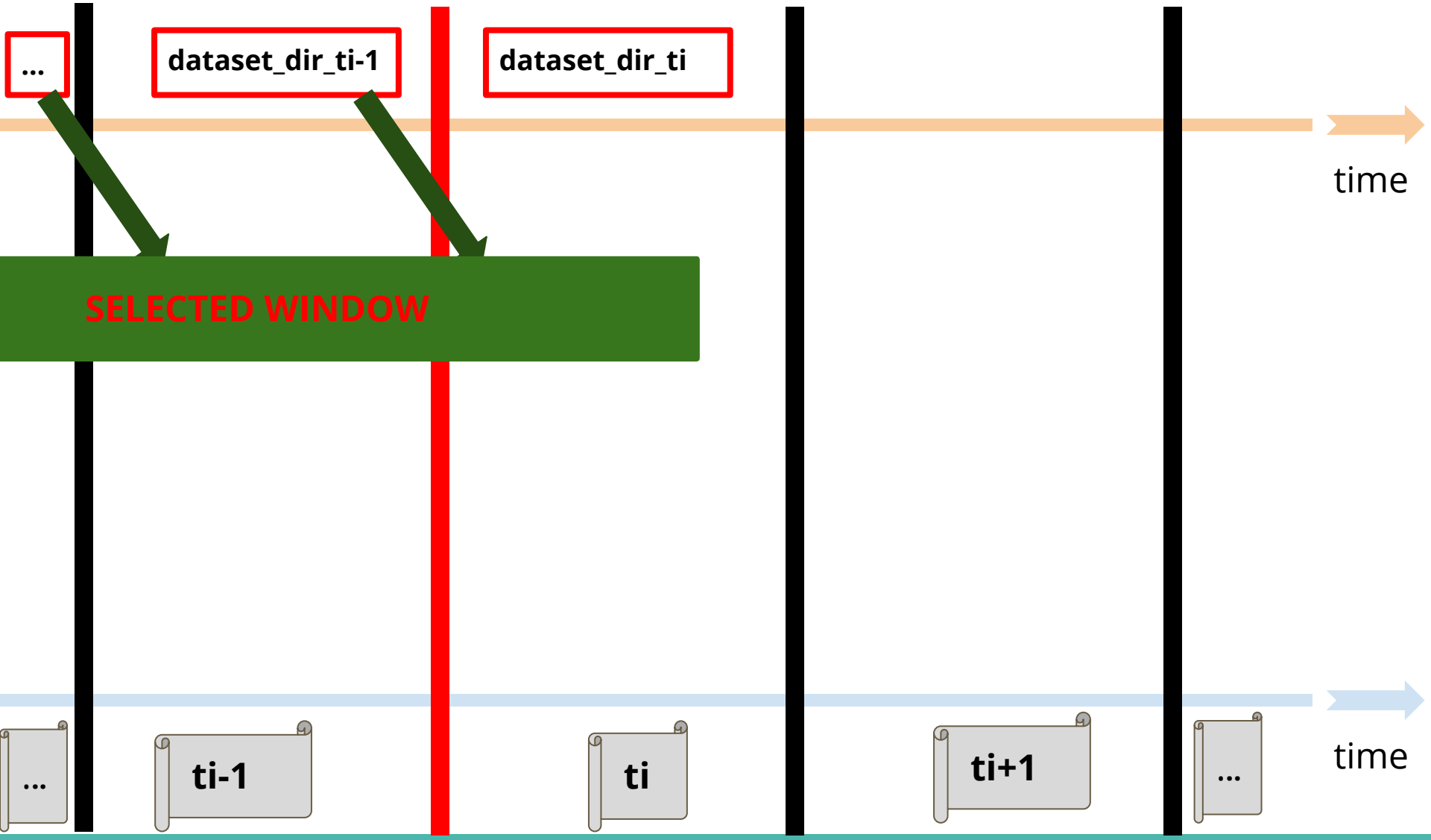
Concept3: Append Mode with Windows



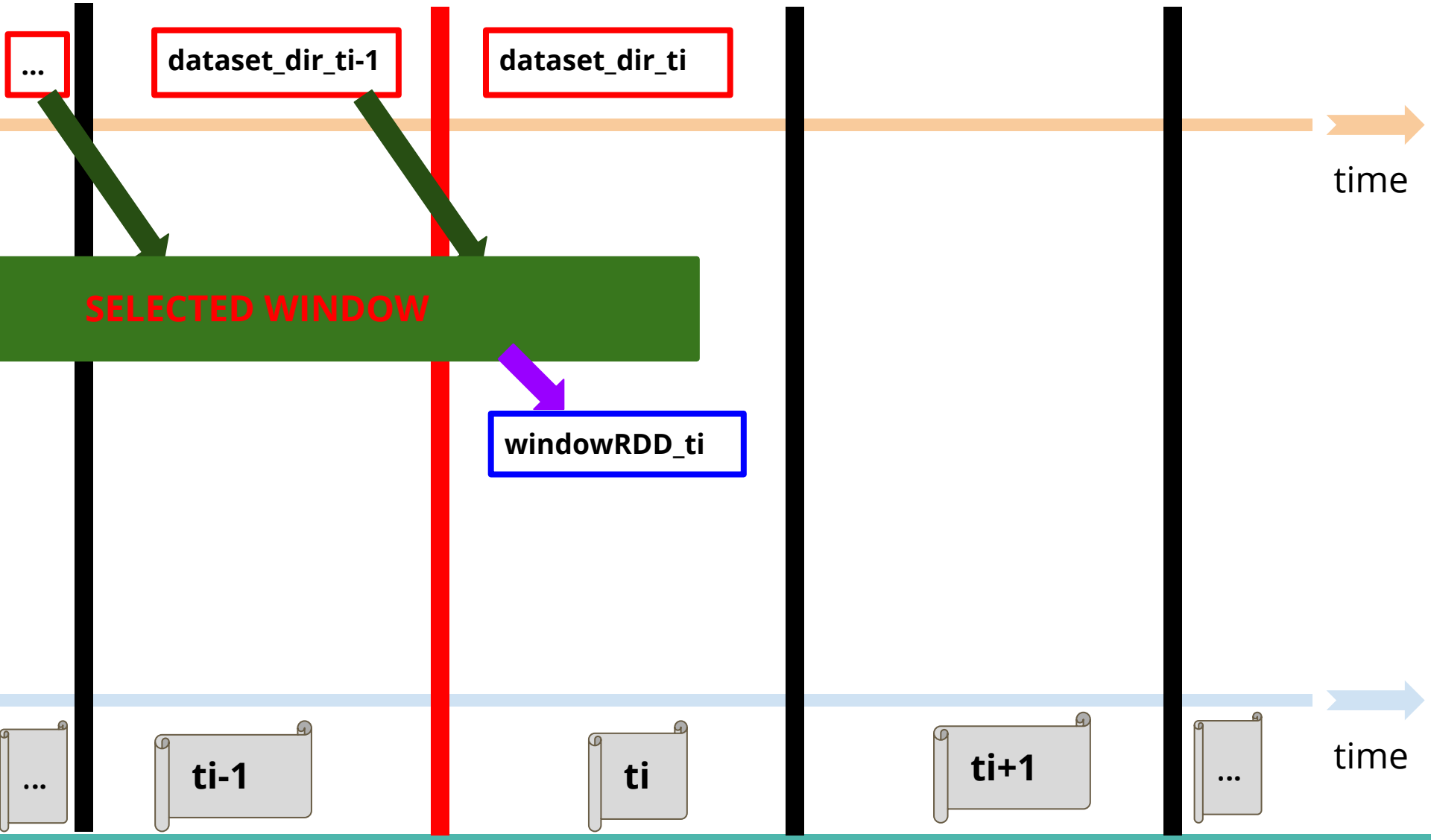
Concept3: Append Mode with Windows



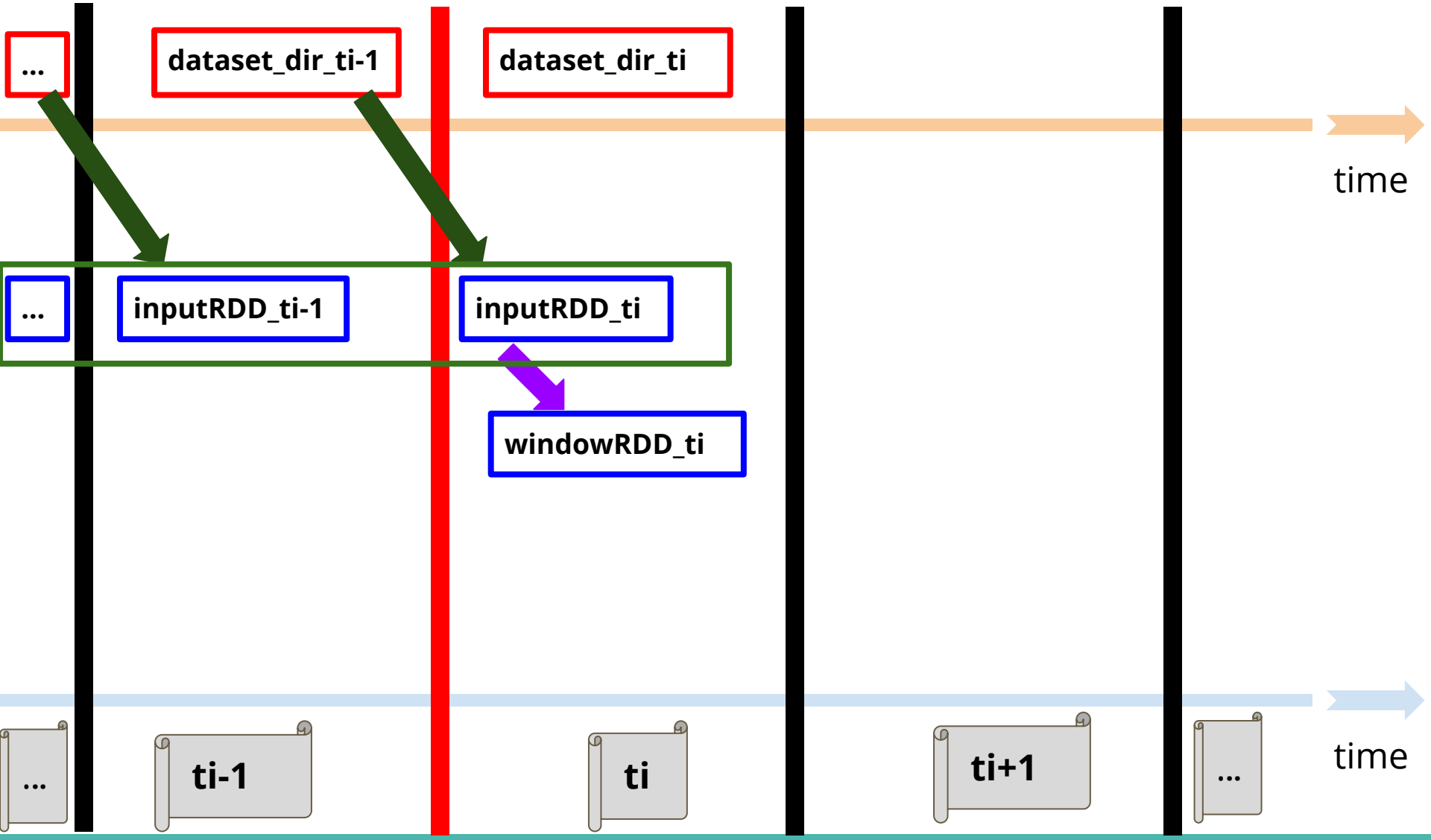
Concept3: Append Mode with Windows



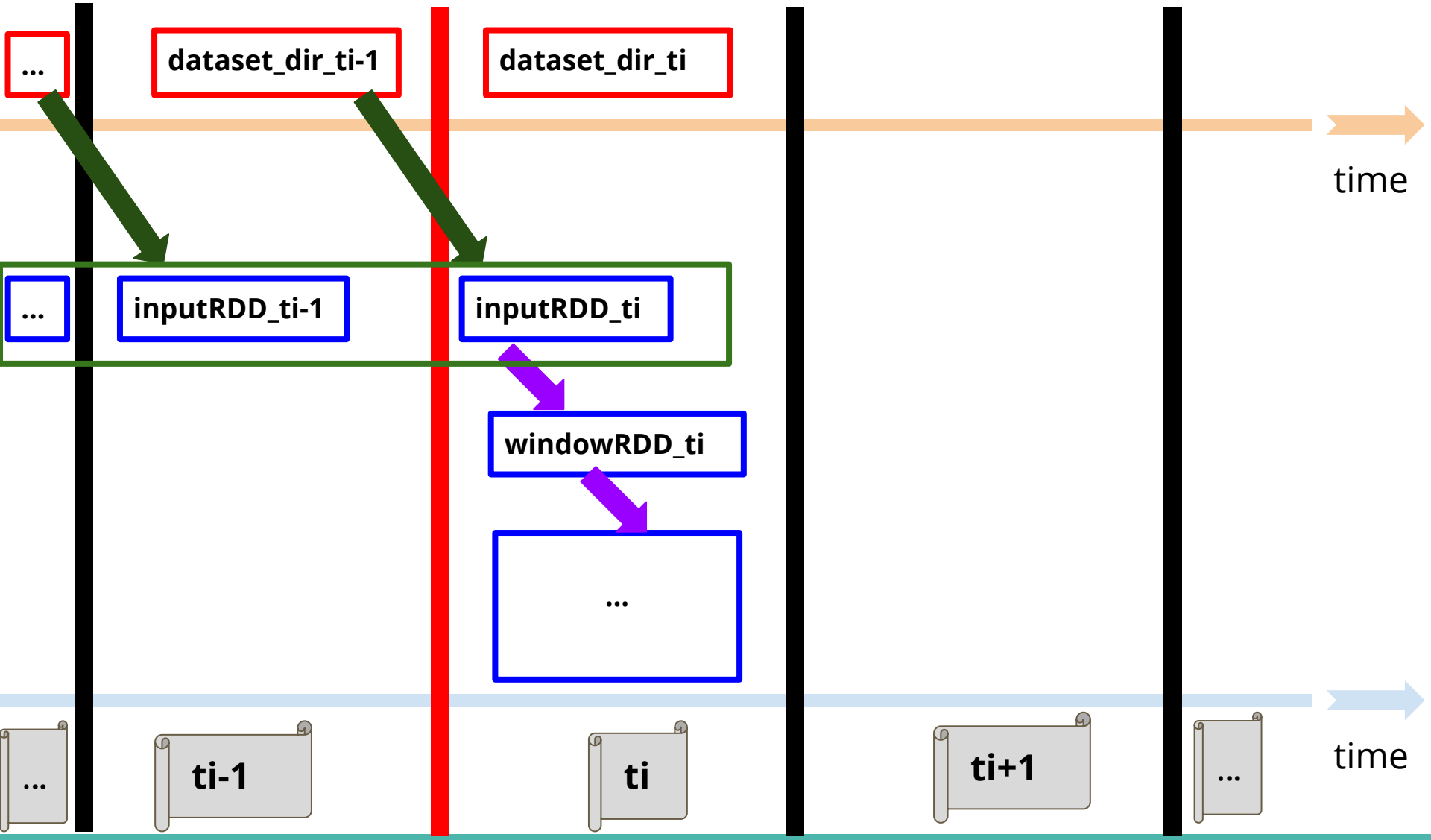
Concept3: Append Mode with Windows



Concept3: Append Mode with Windows



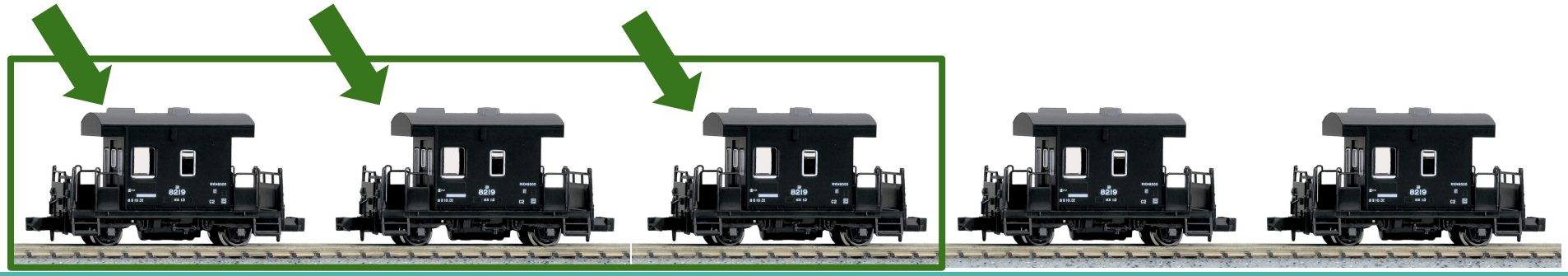
Concept3: Append Mode with Windows



Concept3: Append Mode with Windows

Let's come back to our classical example.

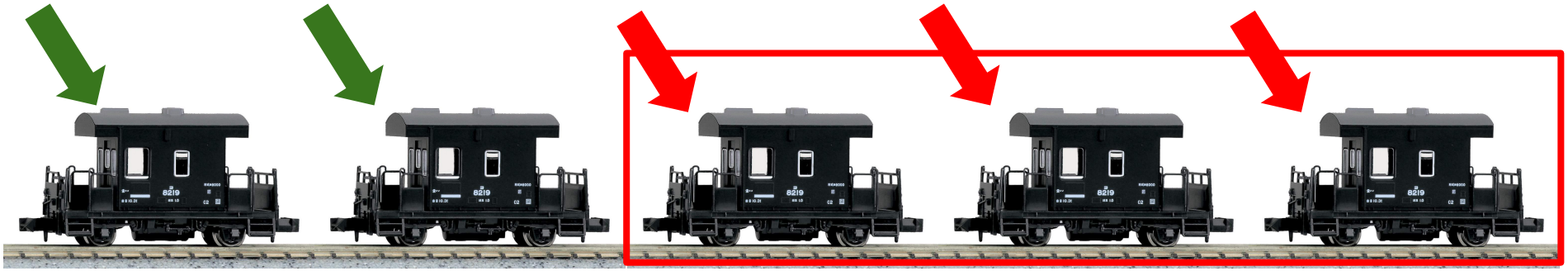
Let's see the transition to the next window:



Concept3: Append Mode with Windows

Let's come back to our classical example.

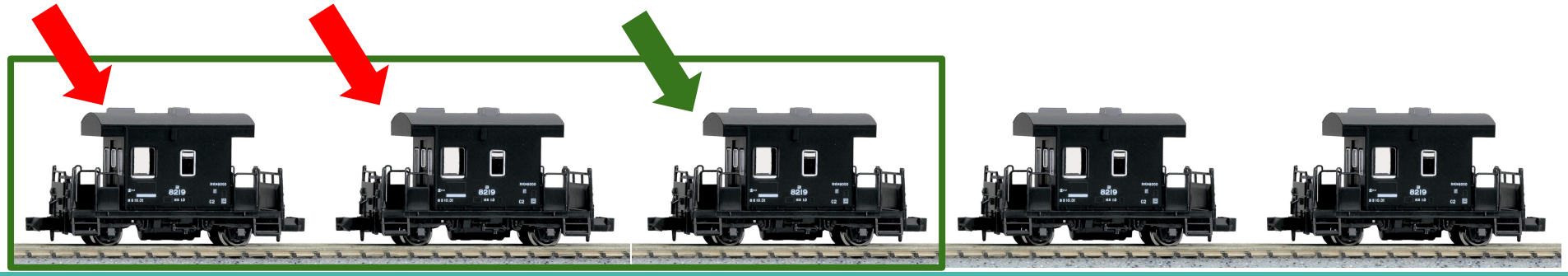
Let's see the transition to the next window:



Concept3: Append Mode with Windows

Let's come back to our classical example.

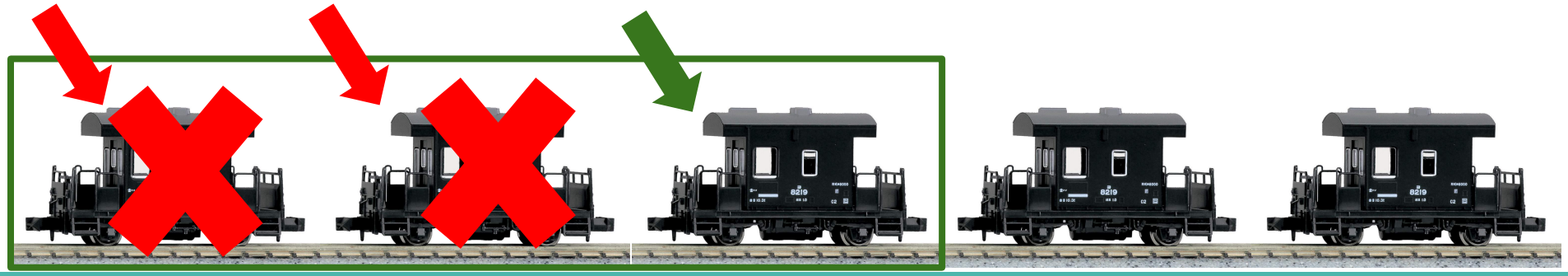
As we can see, the first two wagons are no longer needed for the next window, and thus they can be removed!



Concept3: Append Mode with Windows

Let's come back to our classical example.

As we can see, the first two wagons are no longer needed for the next window, and thus they can be removed!

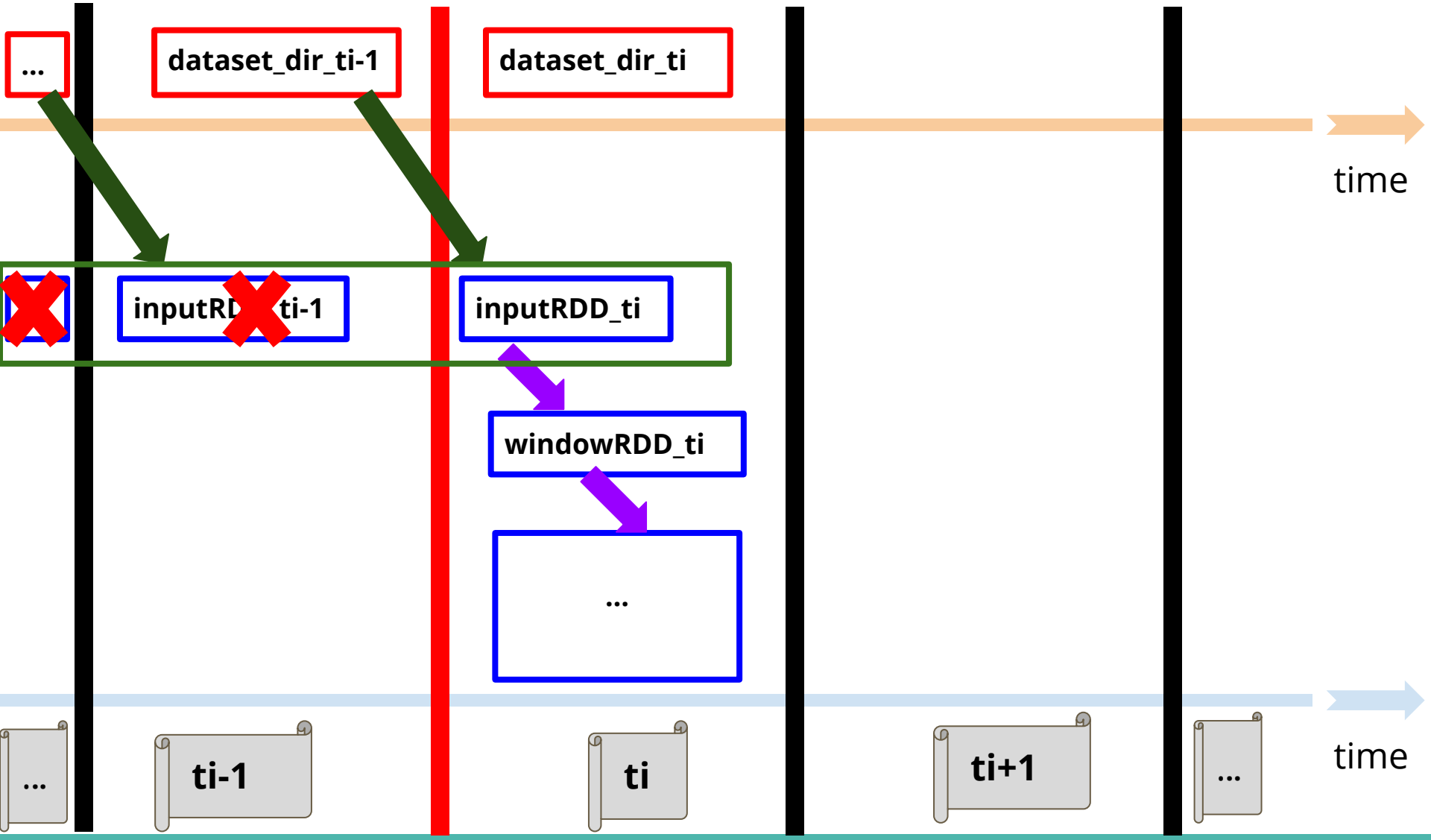


Concept3: Append Mode with Windows

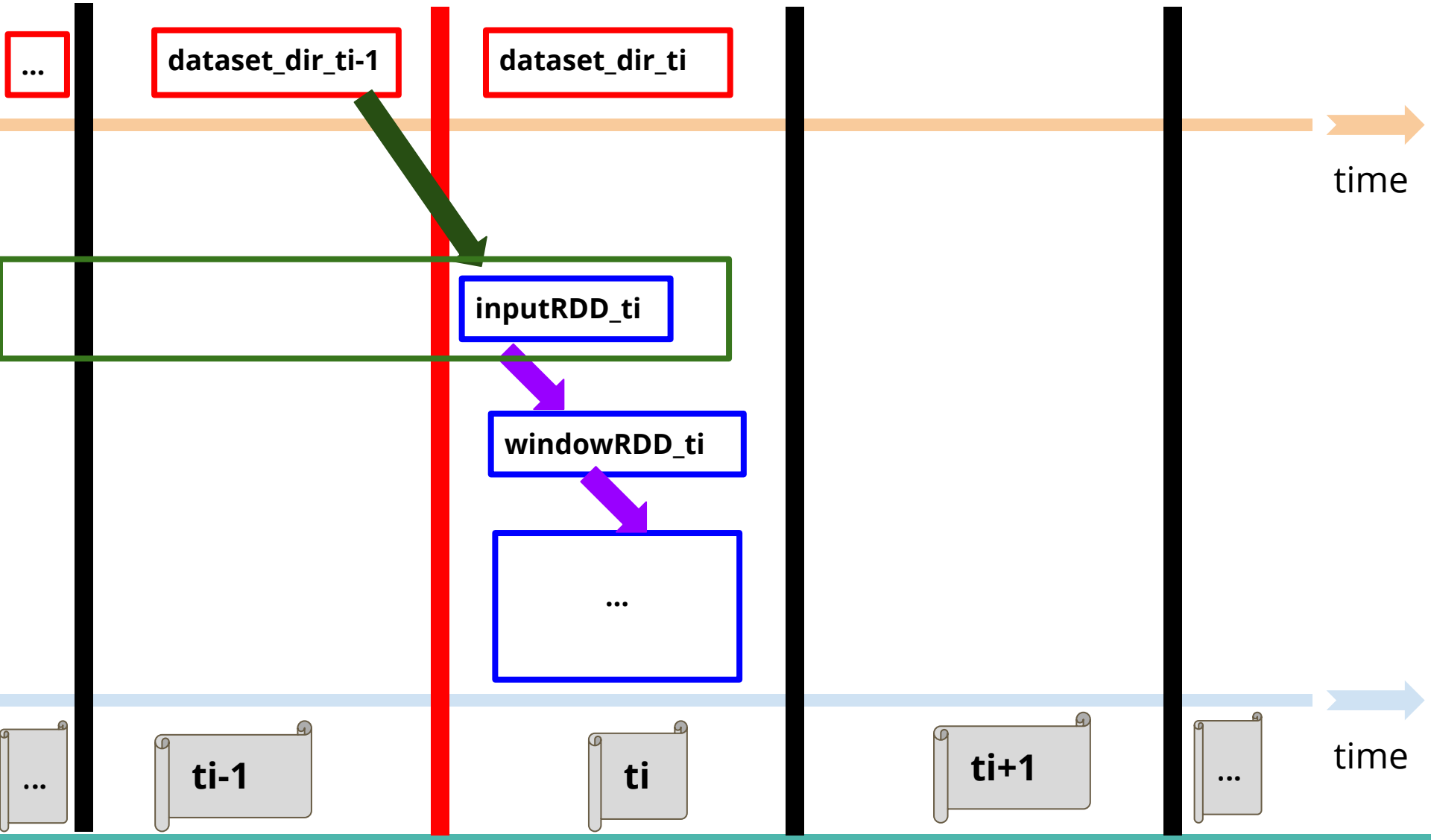
Let's come back to our classical example.

As we can see, the first two wagons are no longer needed for the next window, and thus they can be removed!

Concept3: Append Mode with Windows



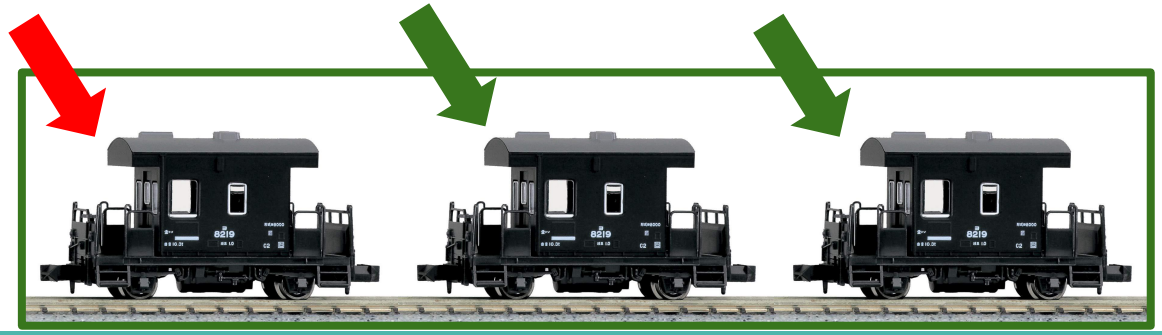
Concept3: Append Mode with Windows



Concept3: Append Mode with Windows

Let's come back to our classical example.

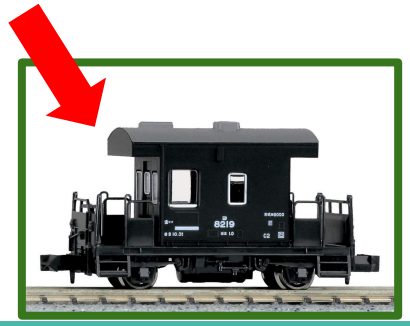
As we can see, the last wagon is indeed needed for the next window, and thus it is automatically kept in memory!



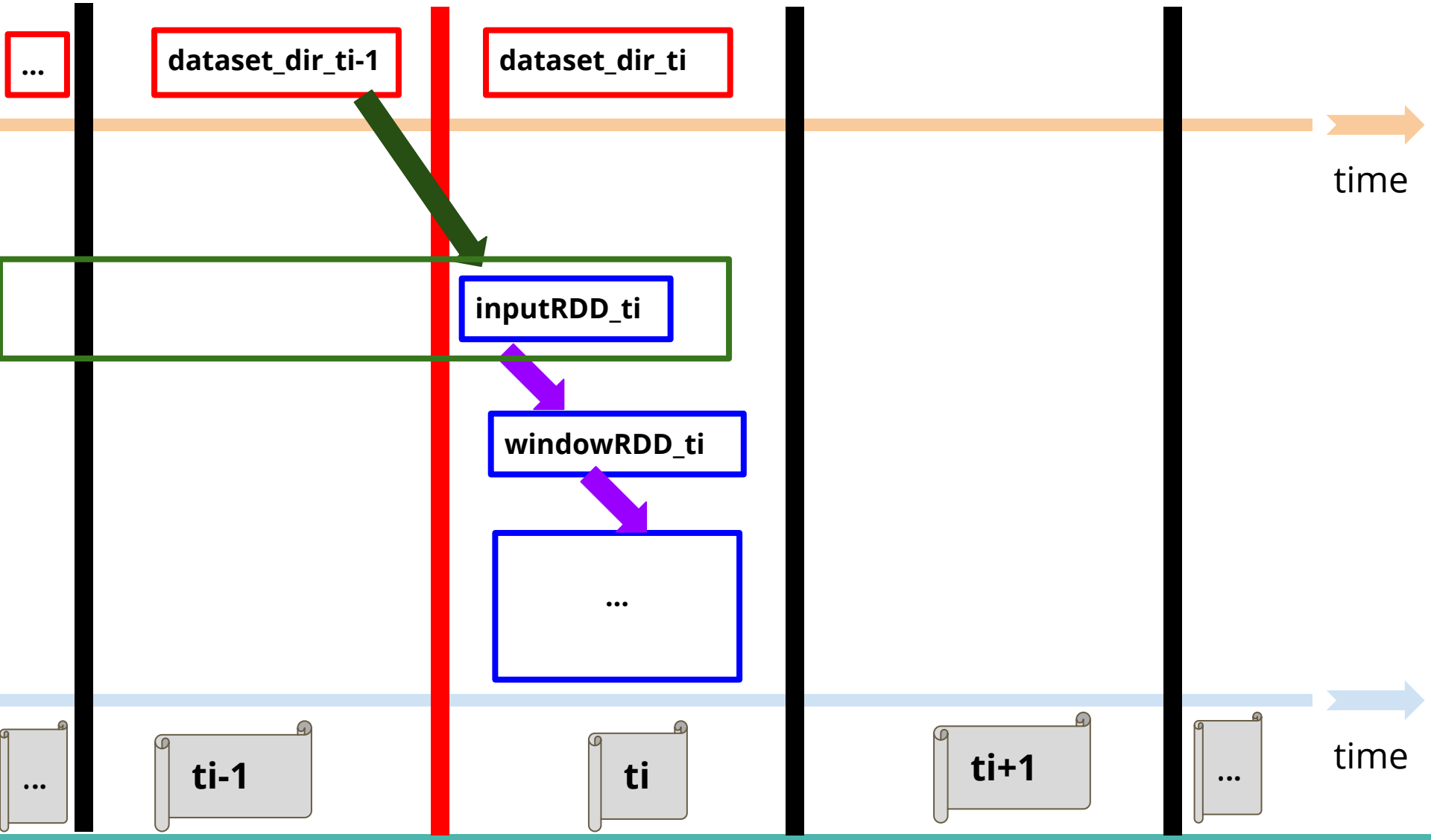
Concept3: Append Mode with Windows

Let's come back to our classical example.

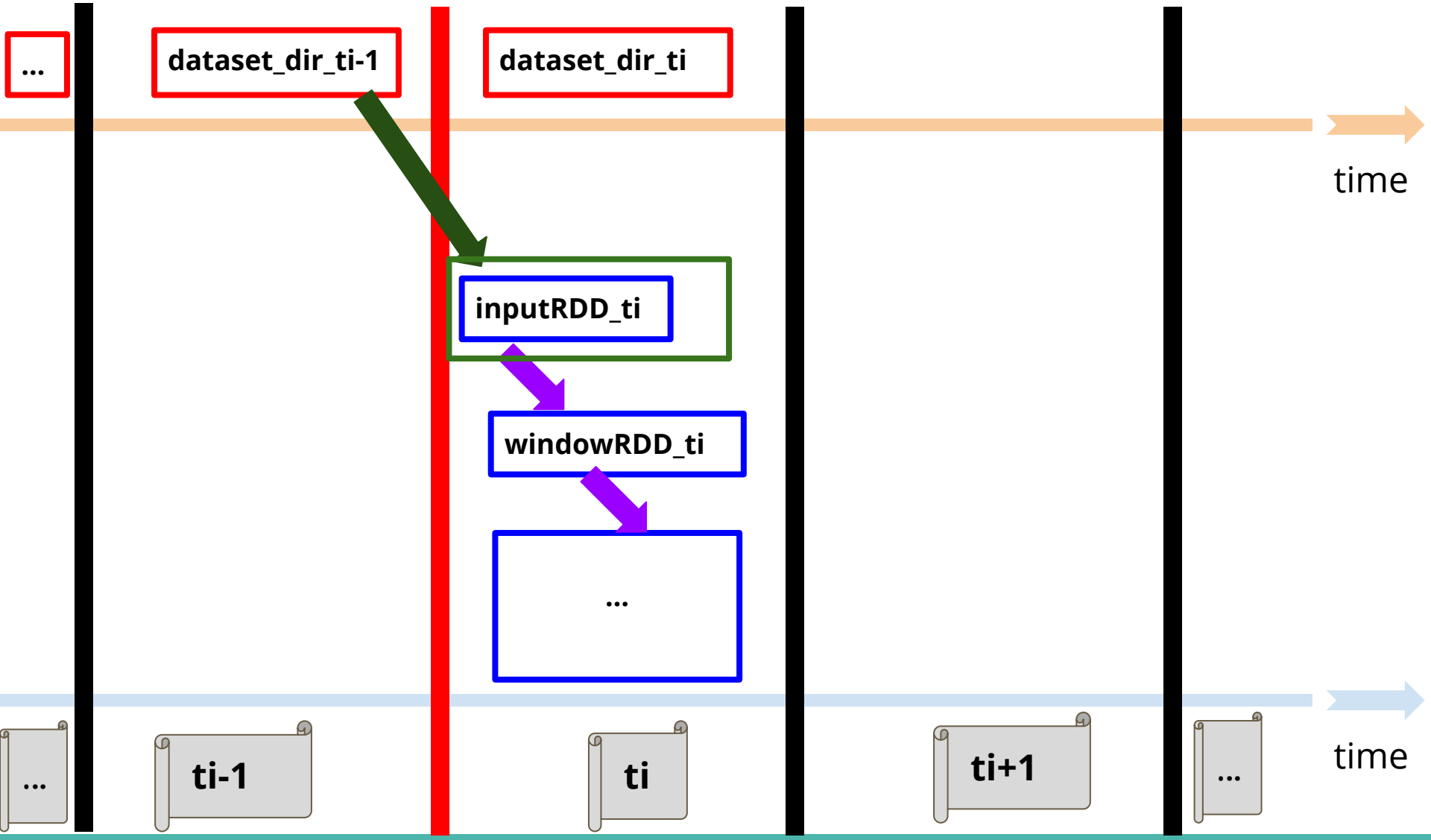
As we can see, the last wagon is indeed needed for the next window, and thus it is automatically kept in memory!



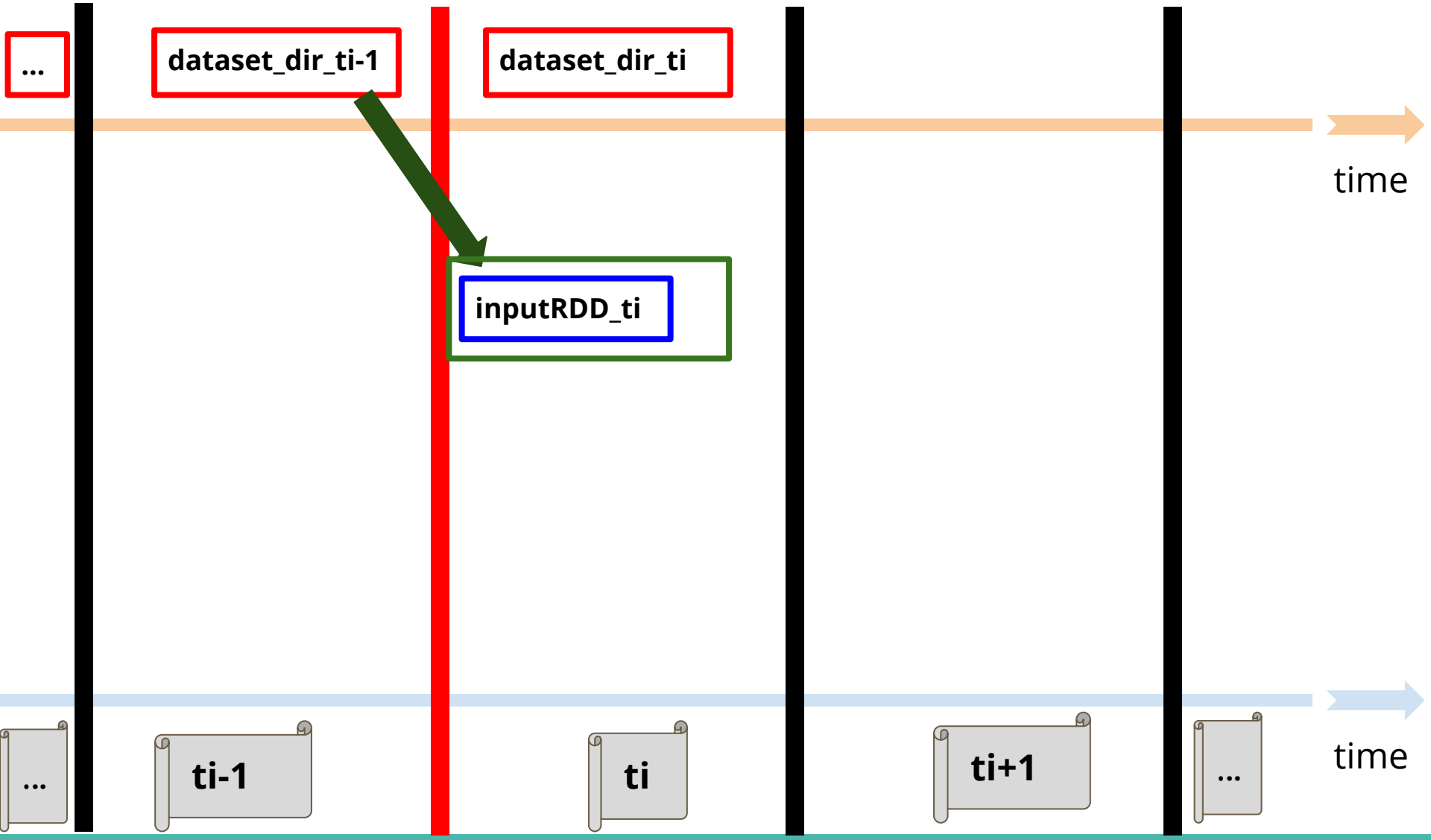
Concept3: Append Mode with Windows



Concept3: Append Mode with Windows



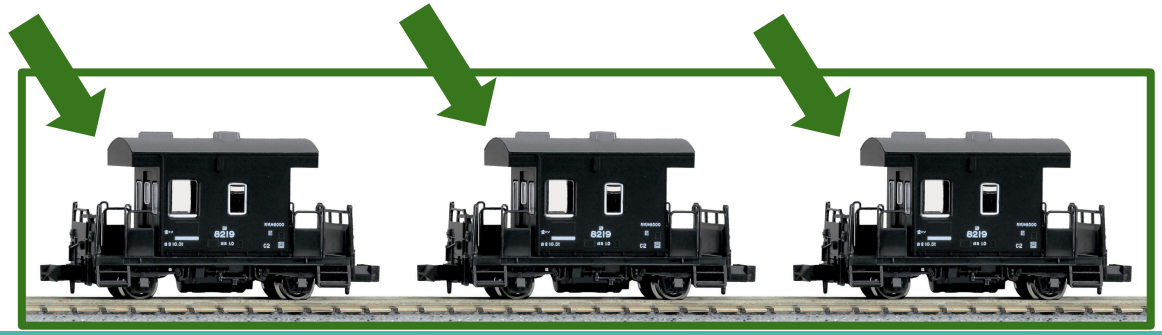
Concept3: Append Mode with Windows



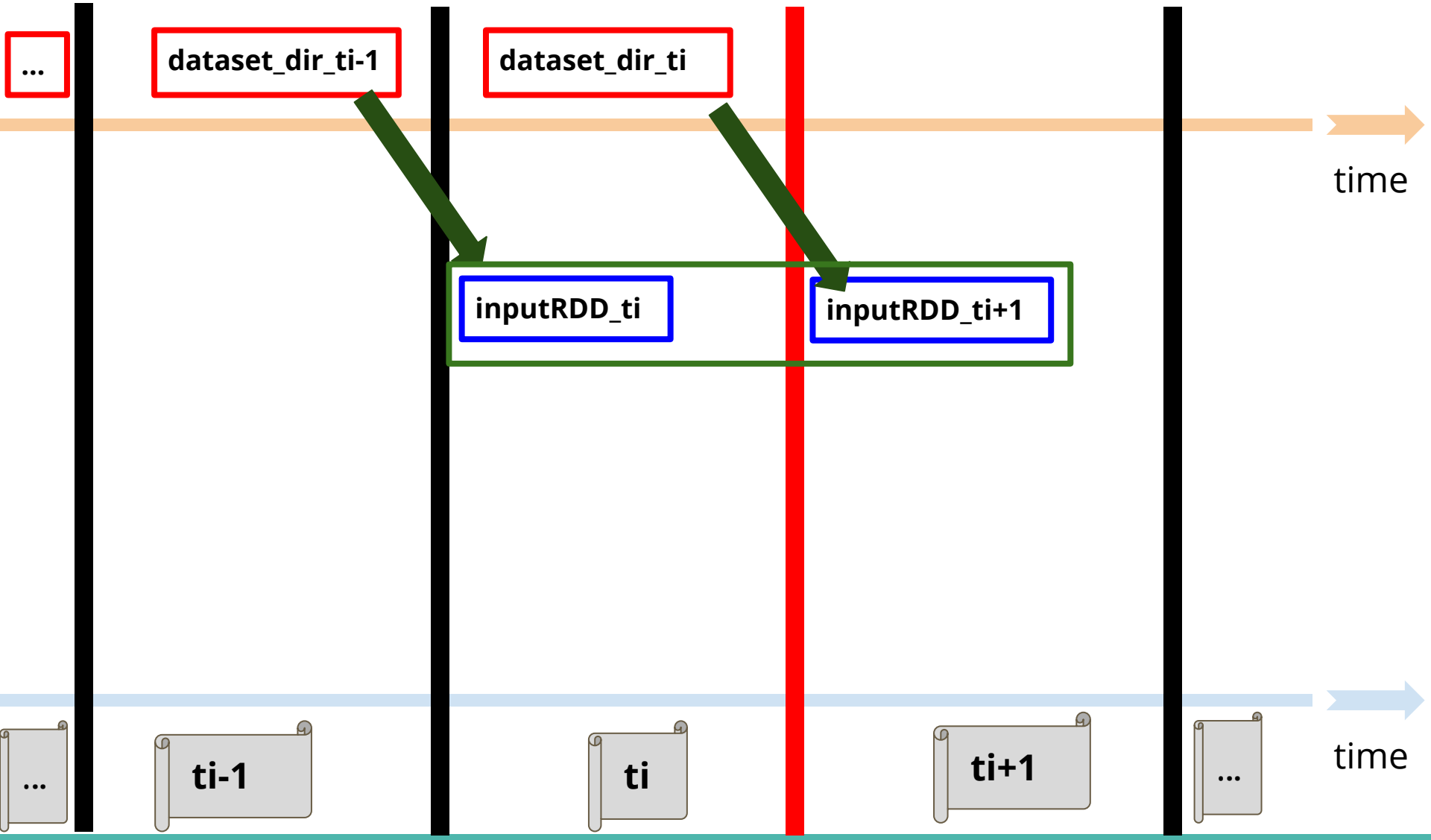
Concept3: Append Mode with Windows

Let's come back to our classical example.

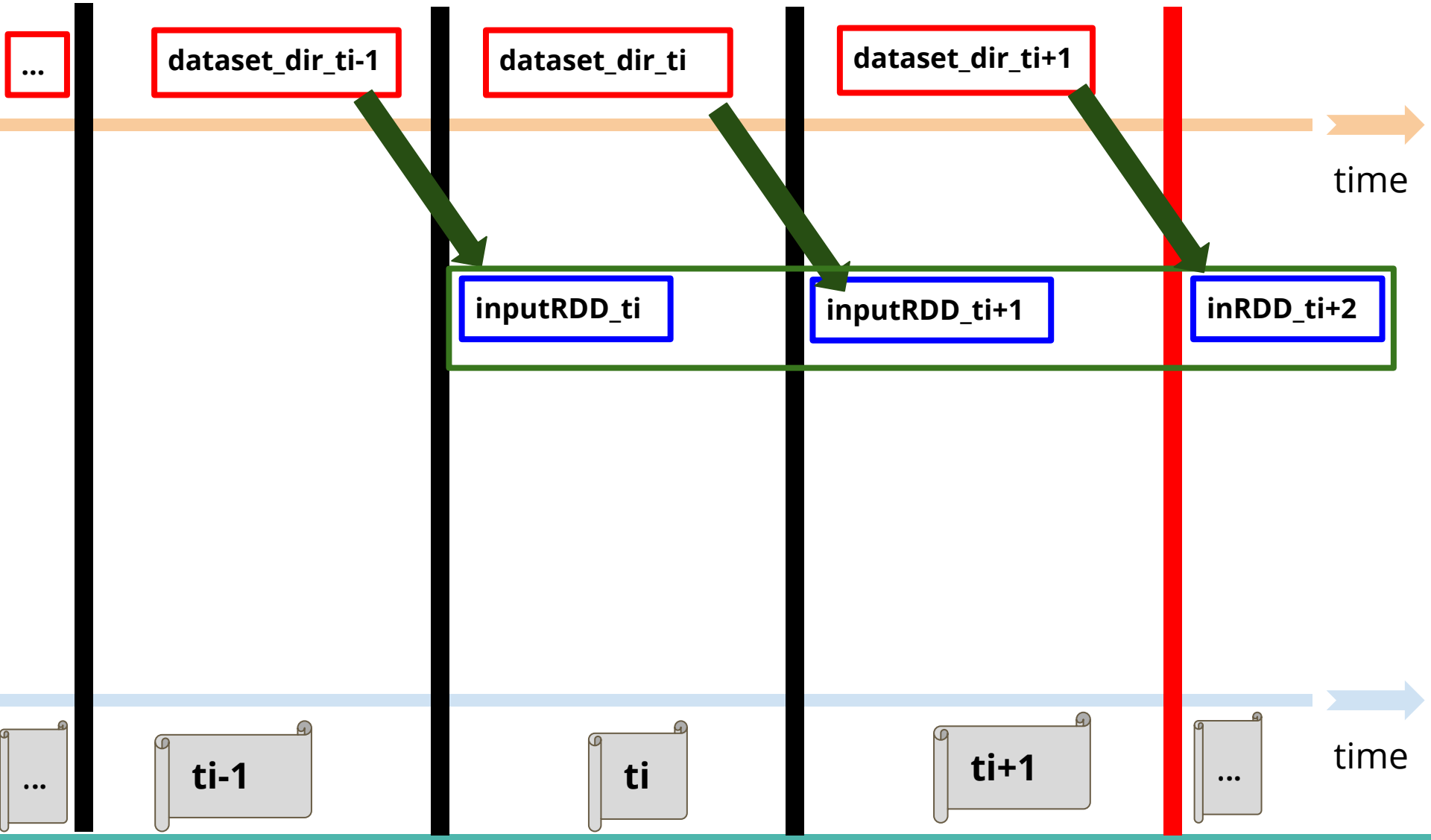
Now the next window happens!



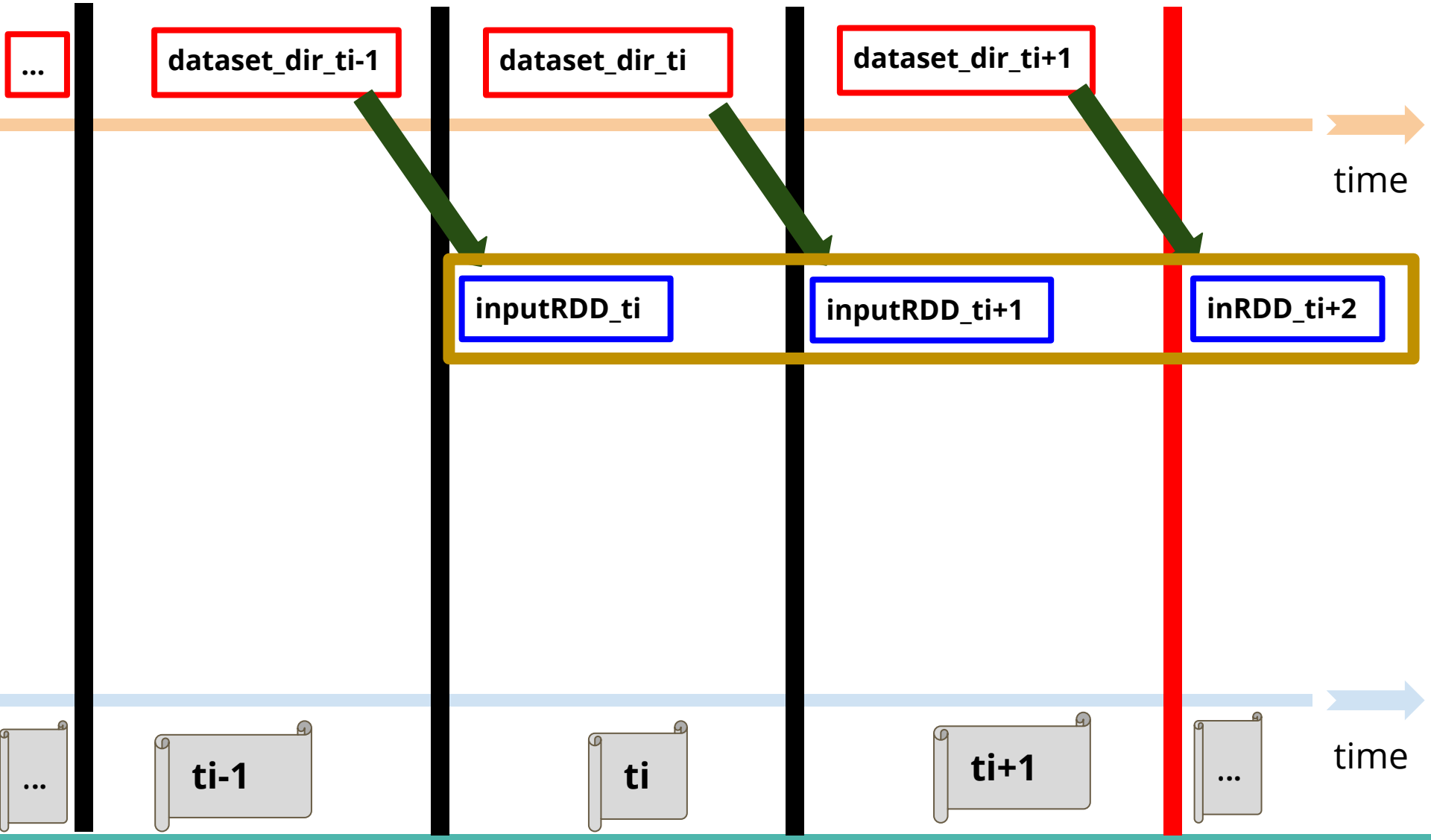
Concept3: Append Mode with Windows



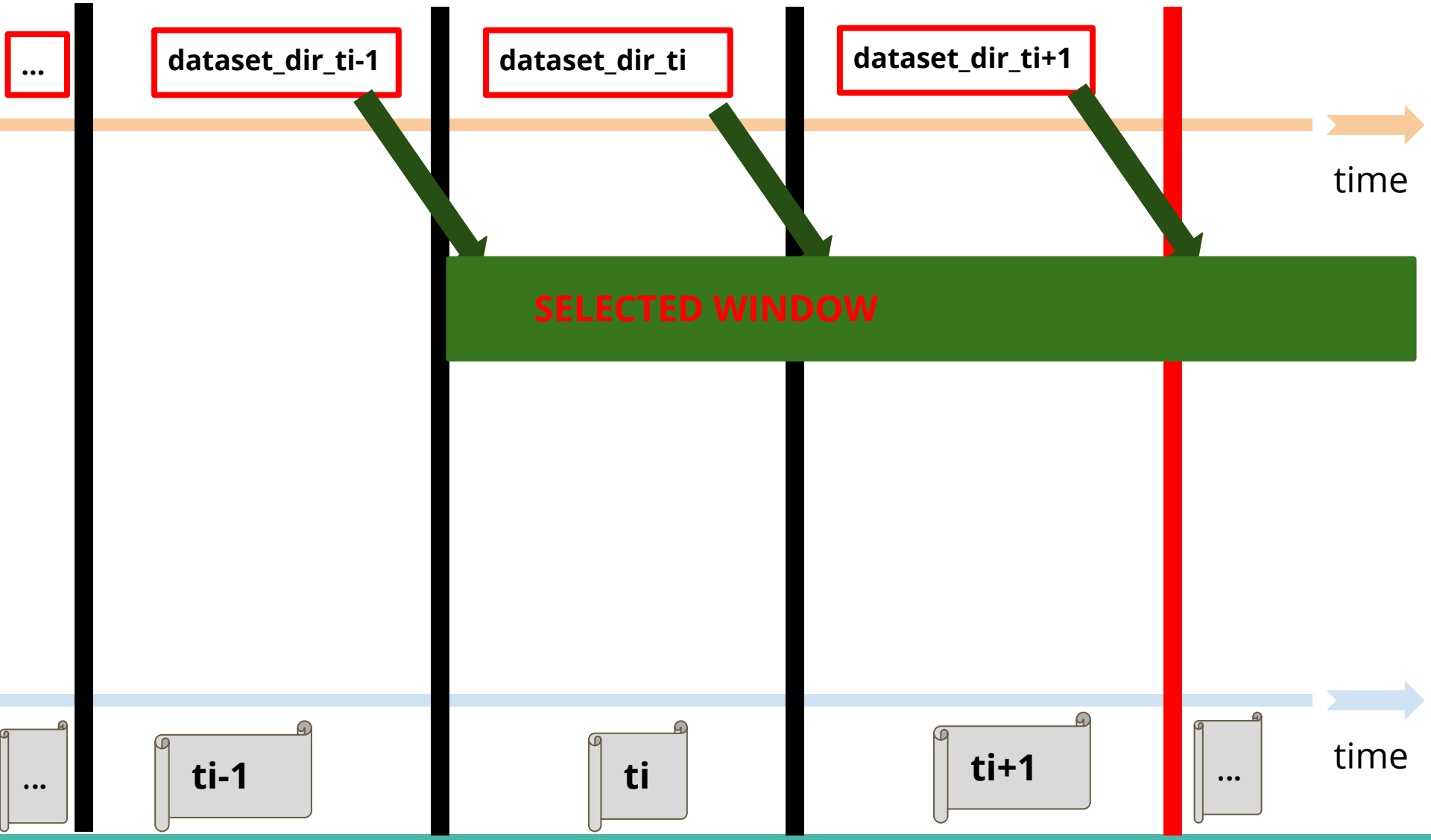
Concept3: Append Mode with Windows



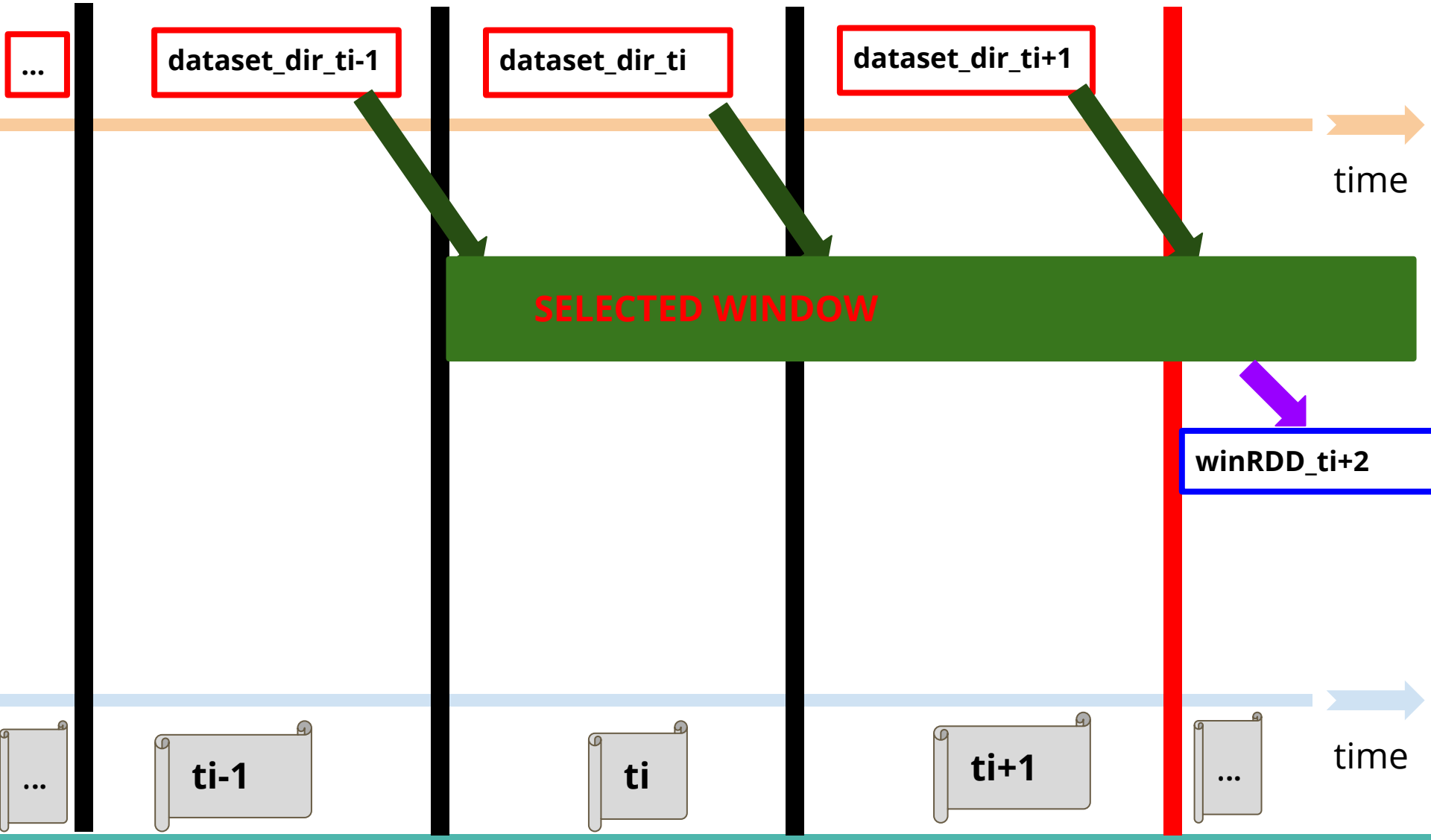
Concept3: Append Mode with Windows



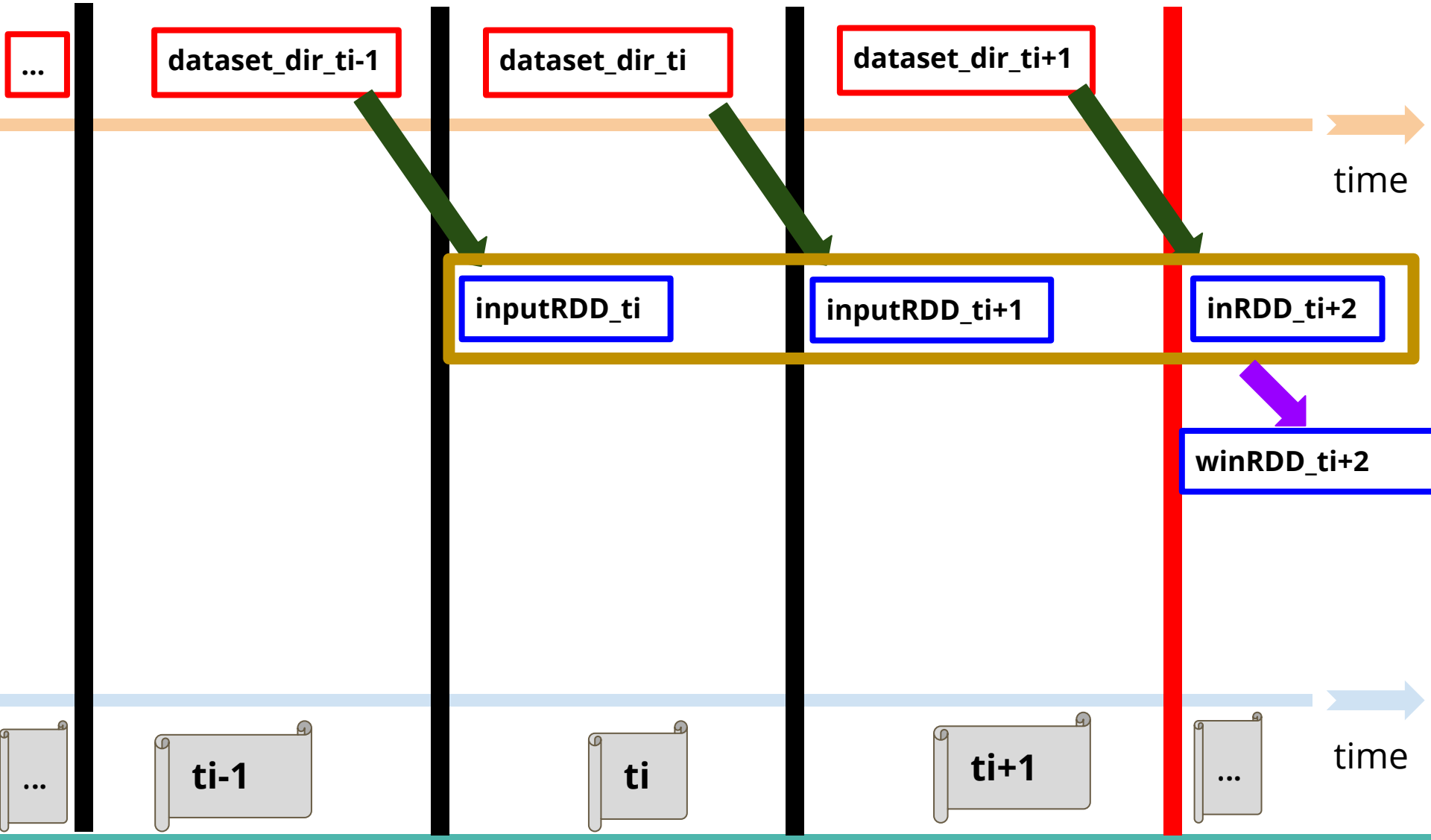
Concept3: Append Mode with Windows



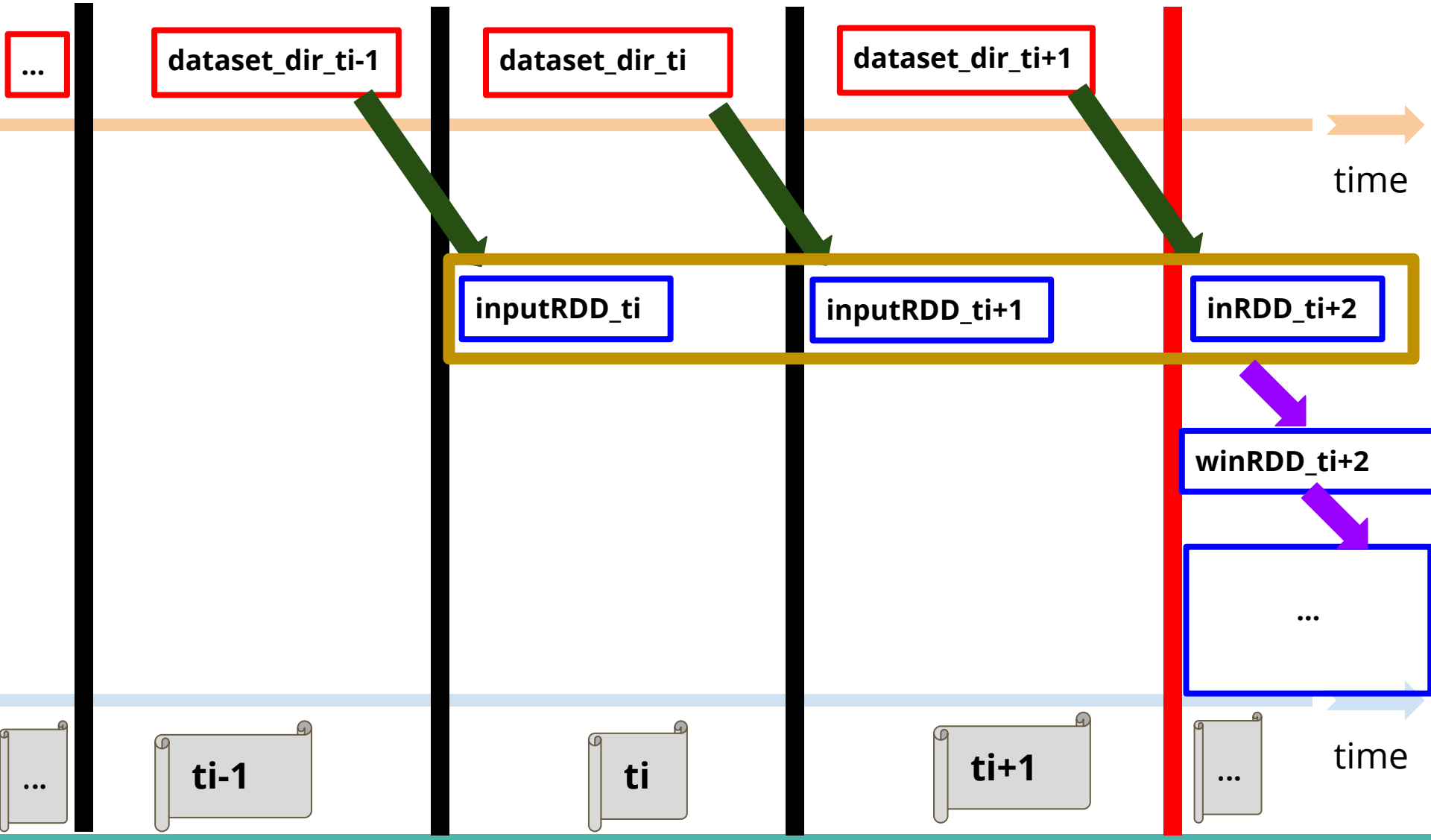
Concept3: Append Mode with Windows



Concept3: Append Mode with Windows



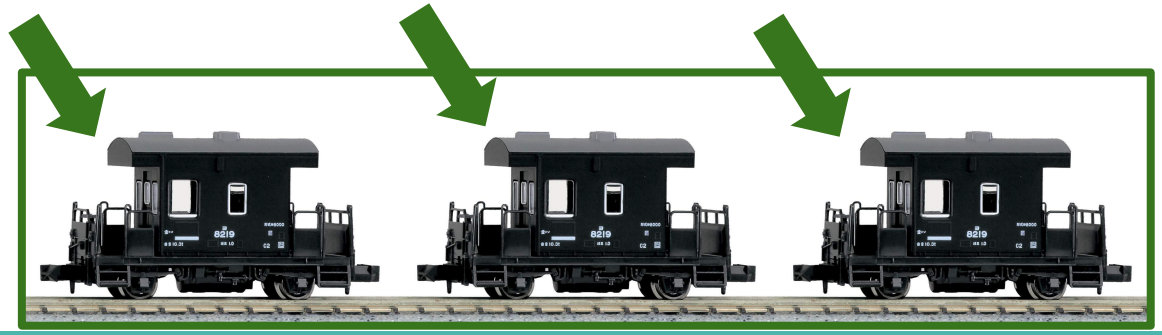
Concept3: Append Mode with Windows



Concept3: Append Mode with Windows

Let's come back to our classical example.

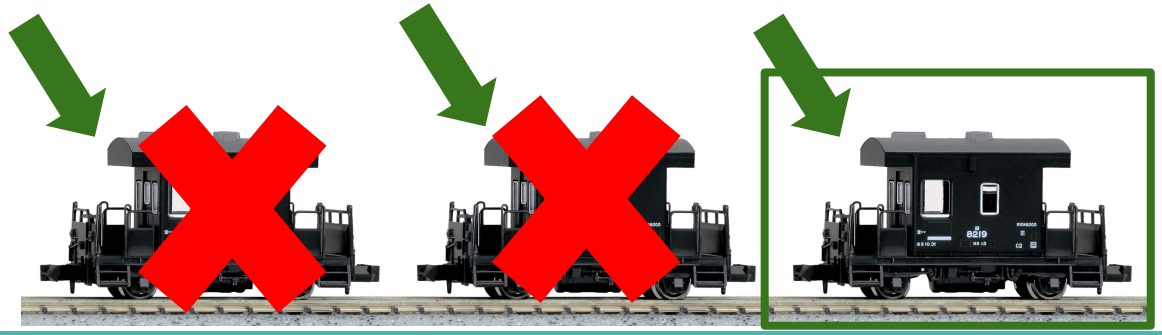
And, again, whereas last wagon is kept in memory for any further window the previous two wagons are removed.



Concept3: Append Mode with Windows

Let's come back to our classical example.

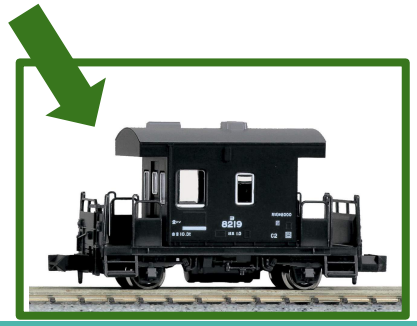
And, again, last wagon is kept in memory for any further window.



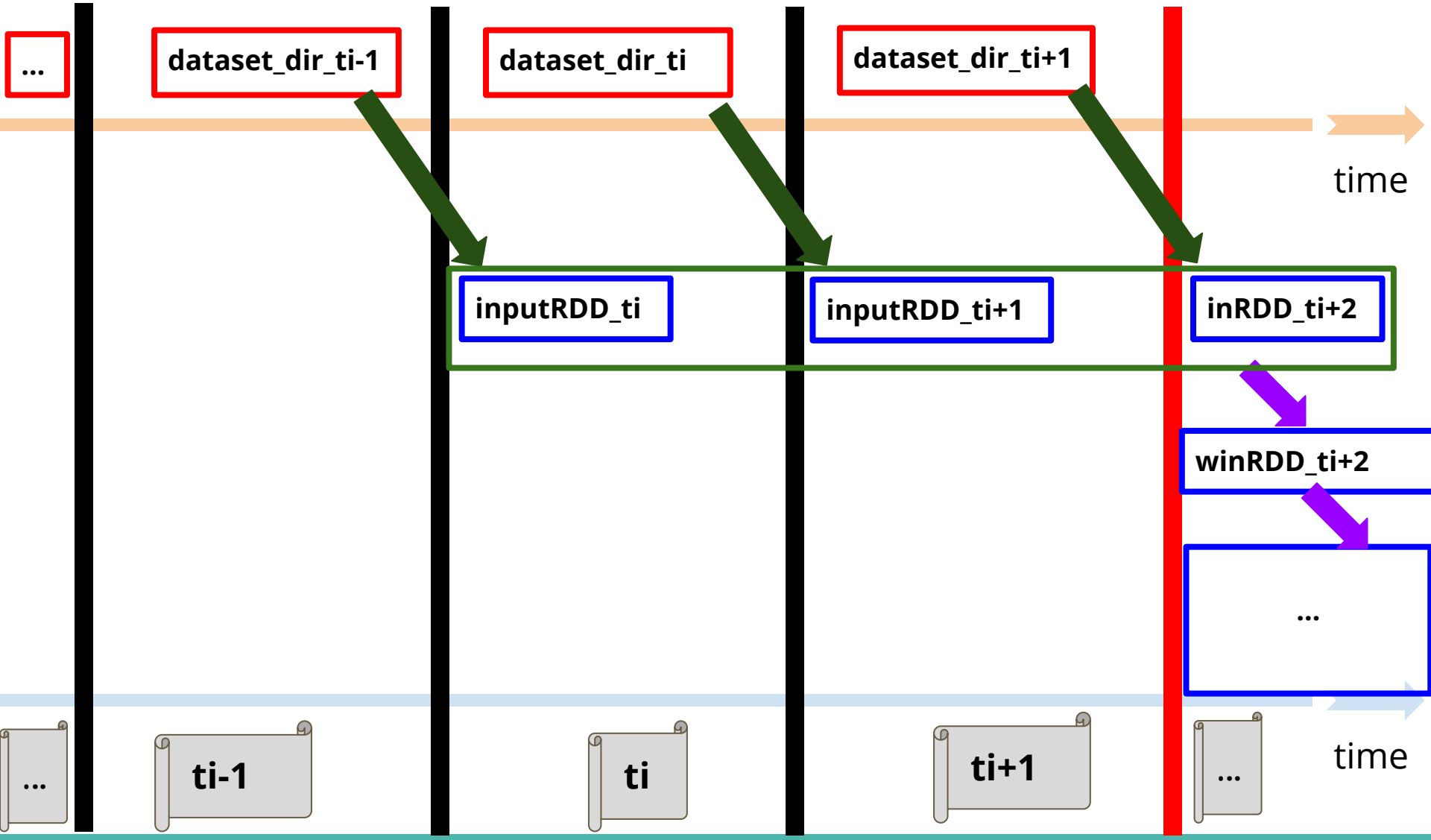
Concept3: Append Mode with Windows

Let's come back to our classical example.

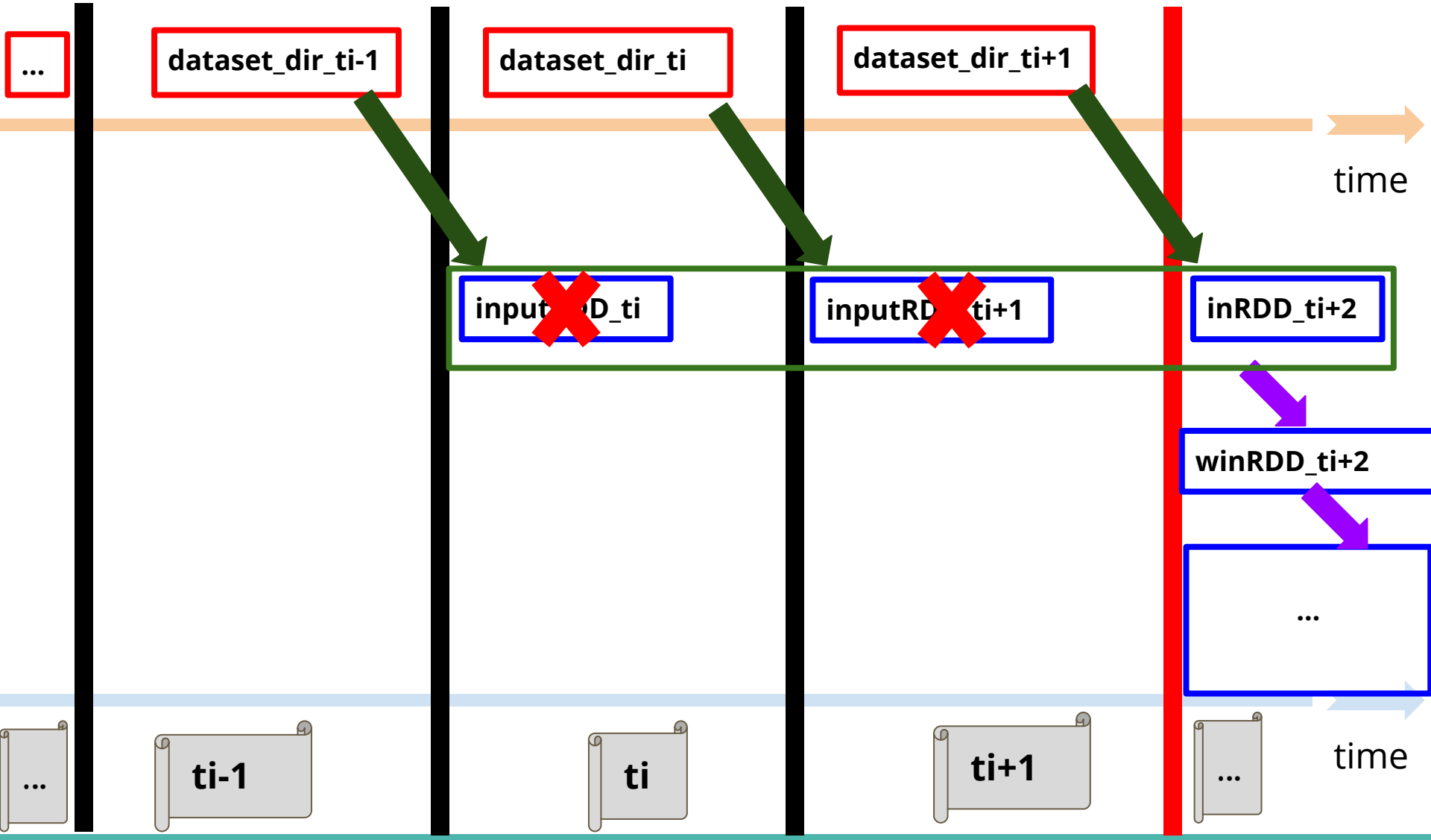
And, again, last wagon is kept in memory for any further window.



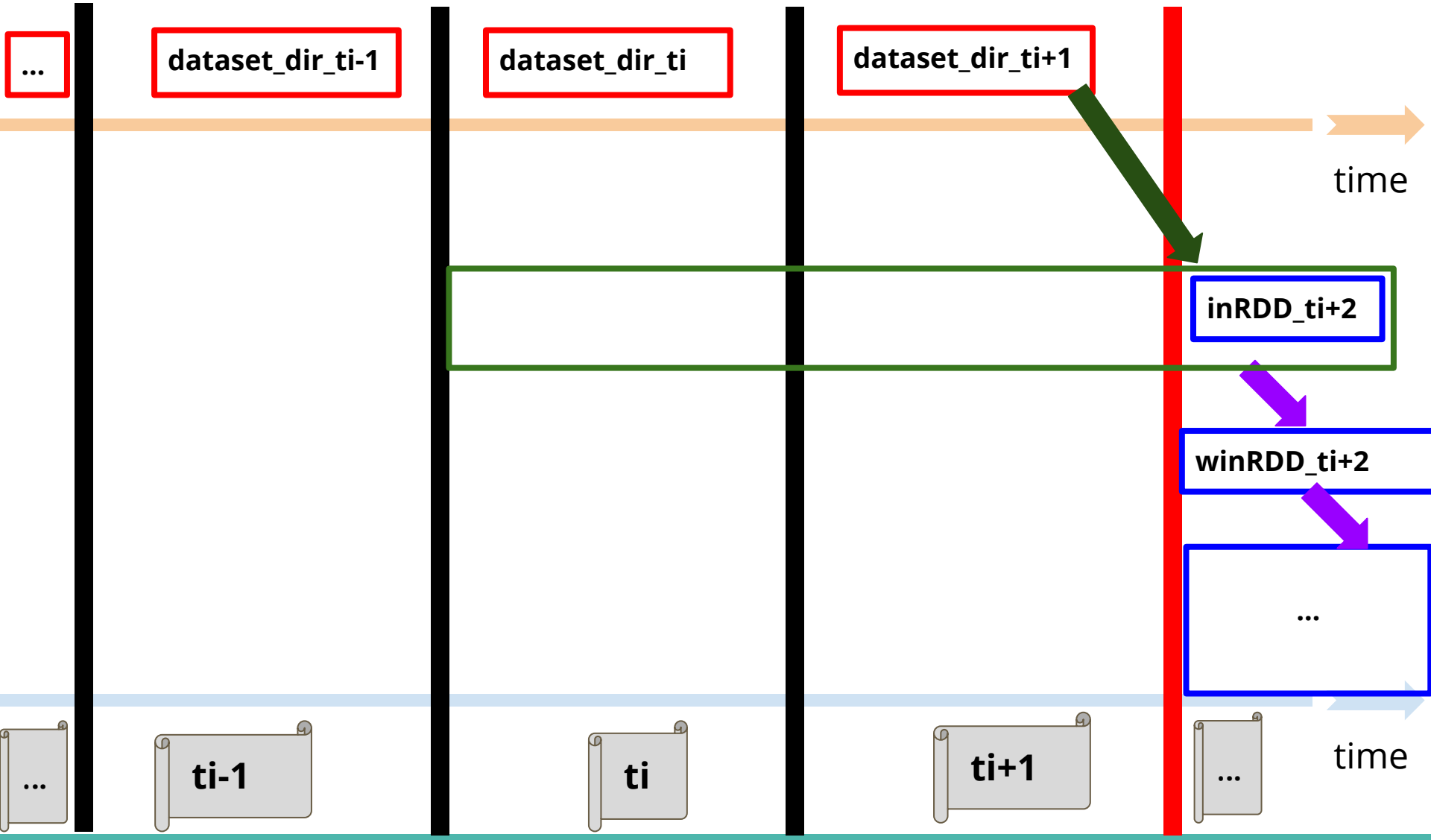
Concept3: Append Mode with Windows



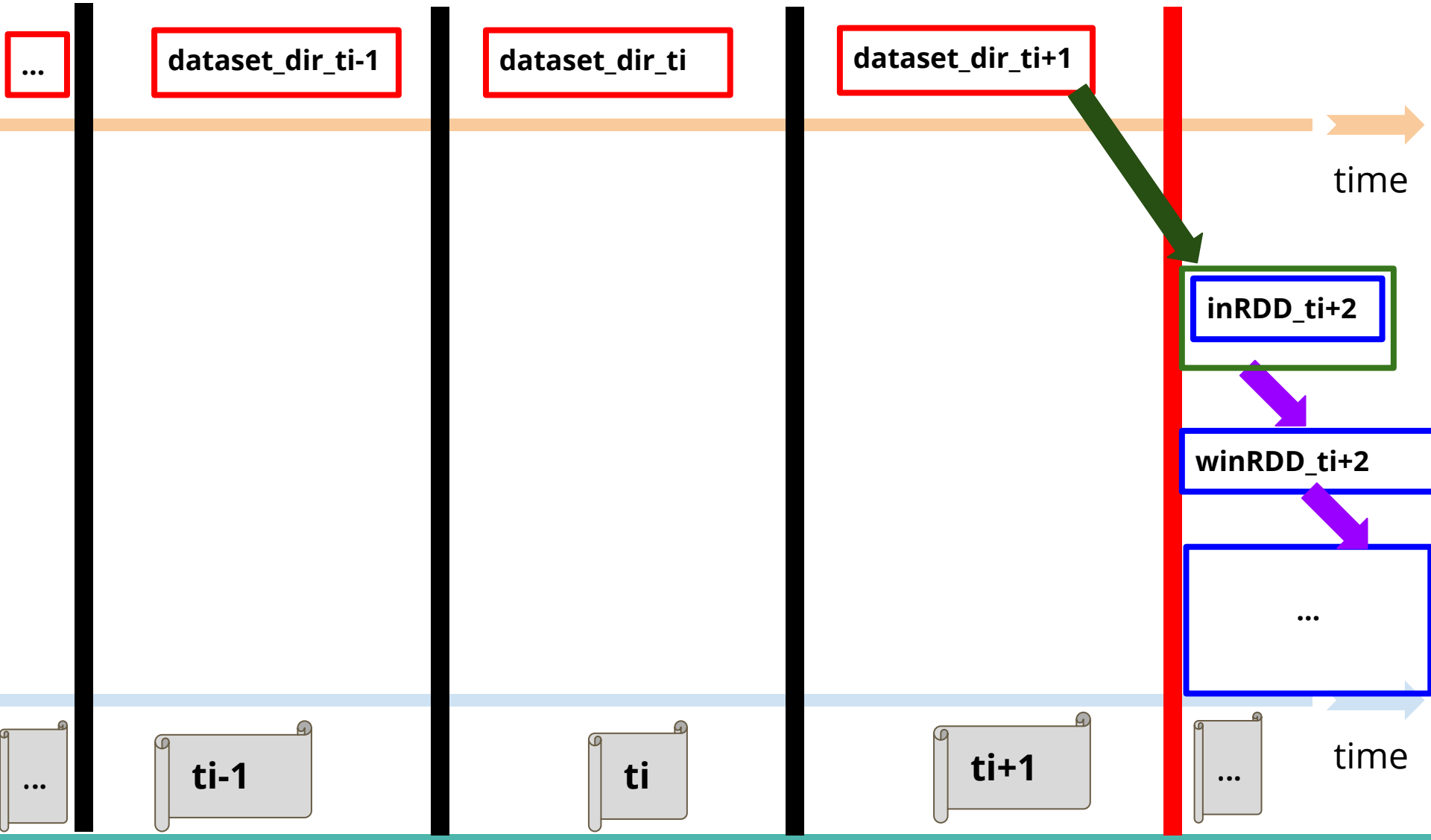
Concept3: Append Mode with Windows



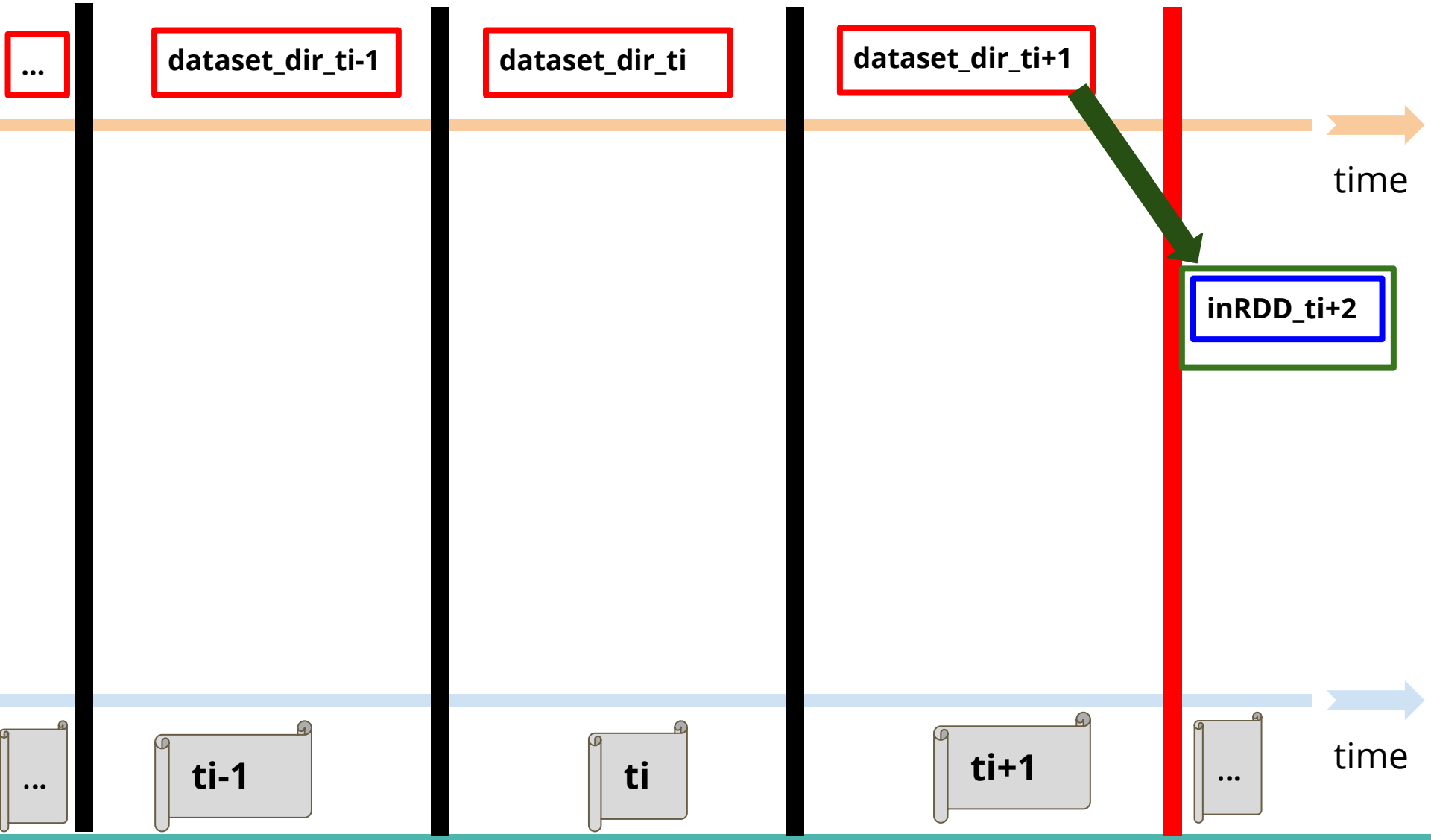
Concept3: Append Mode with Windows



Concept3: Append Mode with Windows



Concept3: Append Mode with Windows



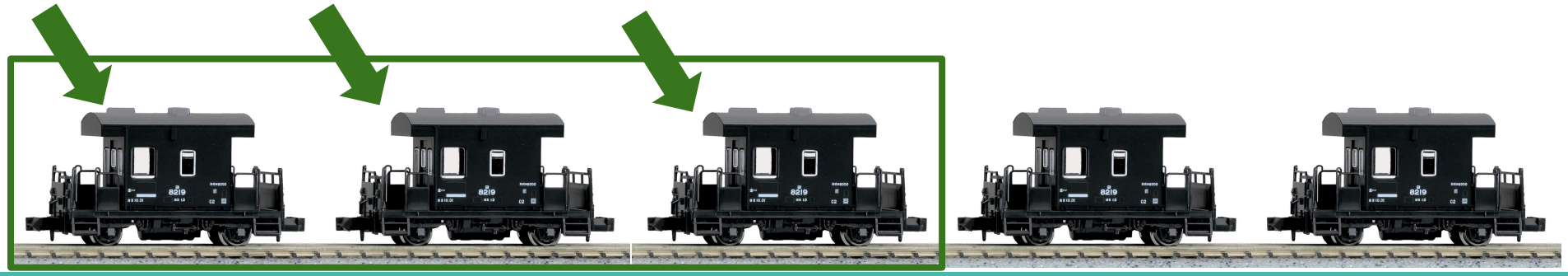
Concept3: Append Mode with Windows

The same behaviour applies to **SDF**!

Concept3: Append Mode with Windows

Let's come back to our classical example.

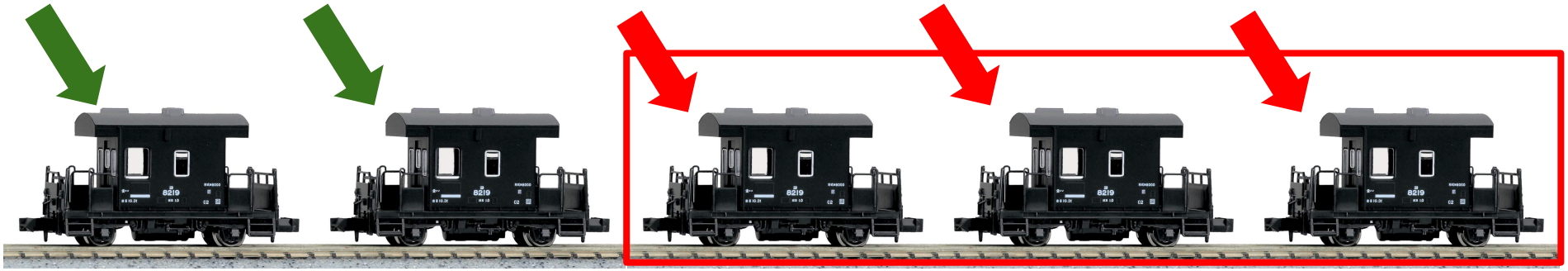
Let's see the transition to the next window:



Concept3: Append Mode with Windows

Let's come back to our classical example.

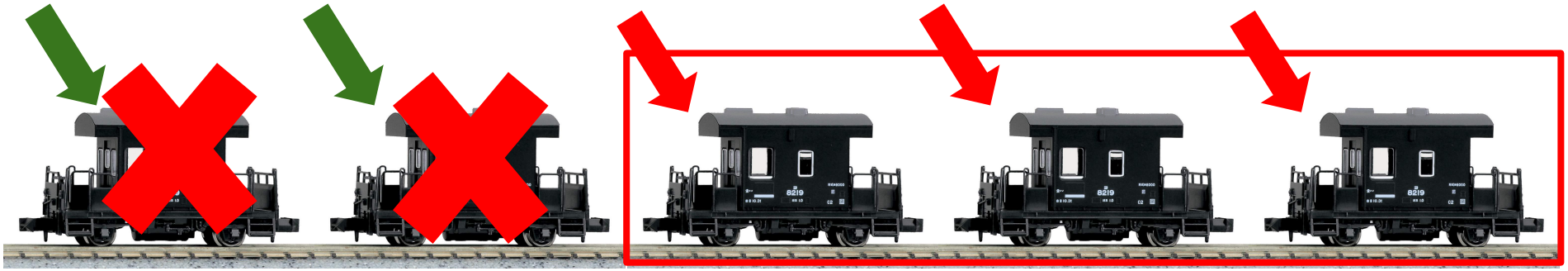
Let's see the transition to the next window:



Concept3: Append Mode with Windows

Let's come back to our classical example.

Let's see the transition to the next window:

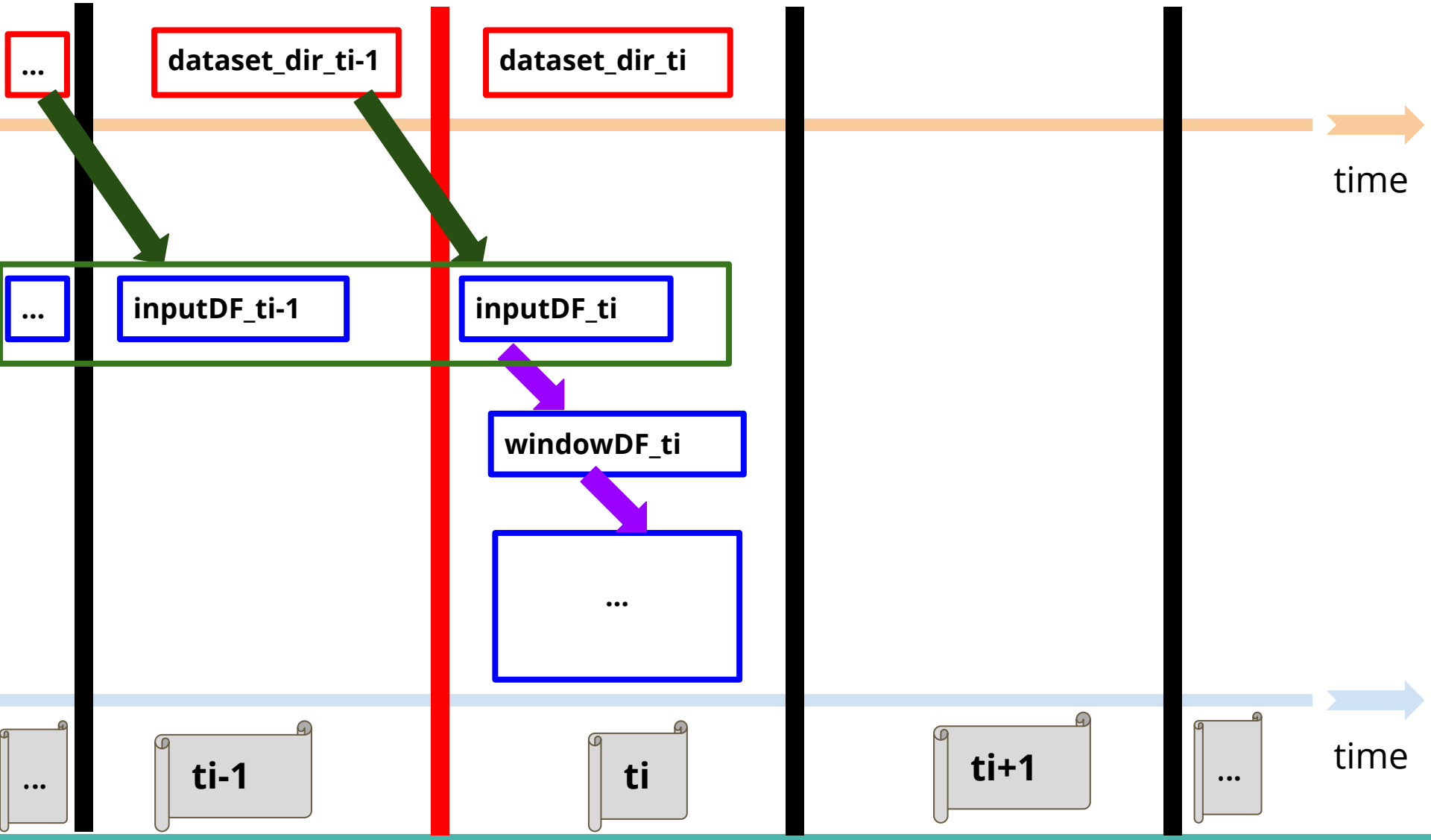


Concept3: Append Mode with Windows

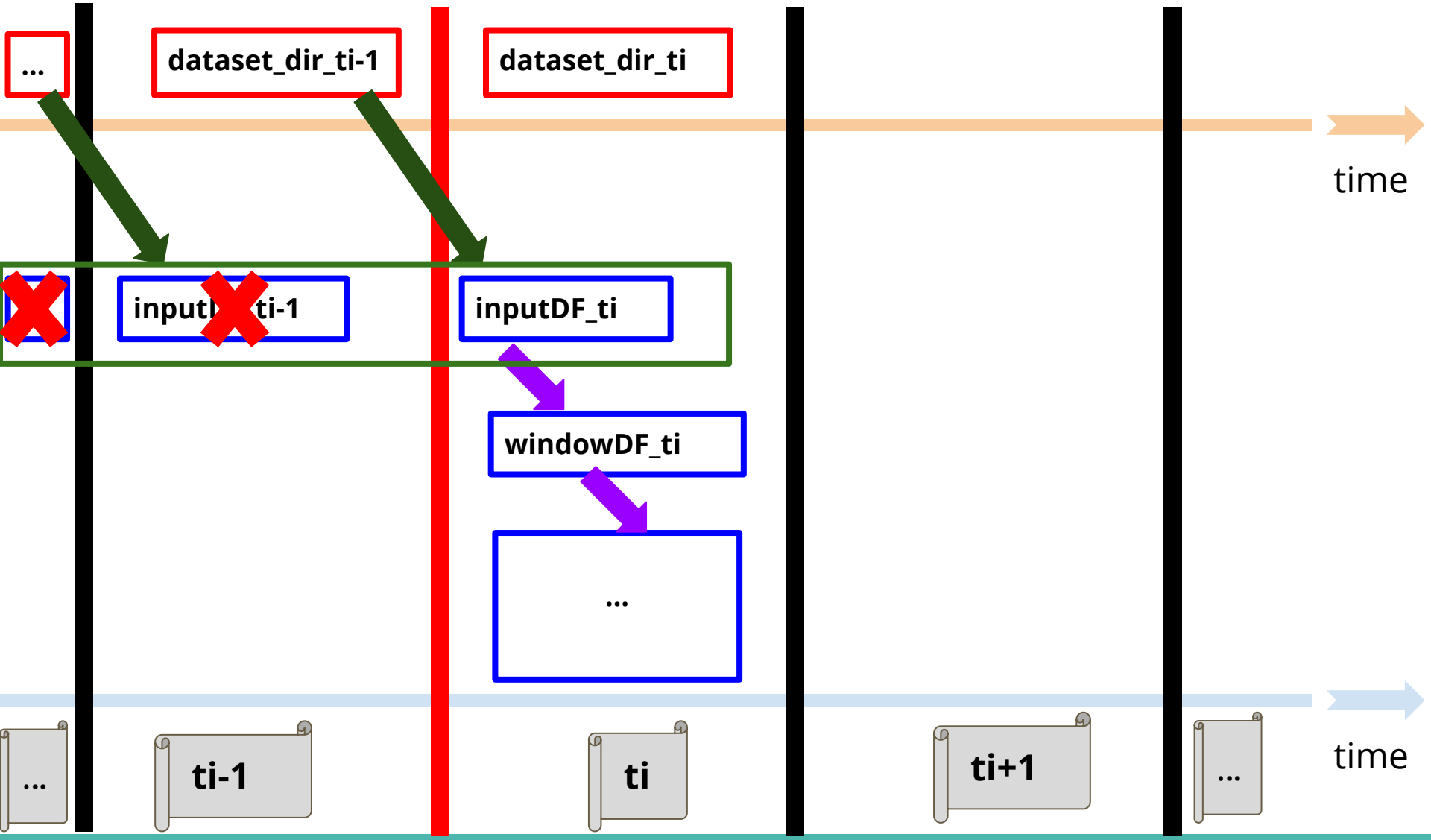
Let's come back to our classical example.

Let's see the transition to the next window:

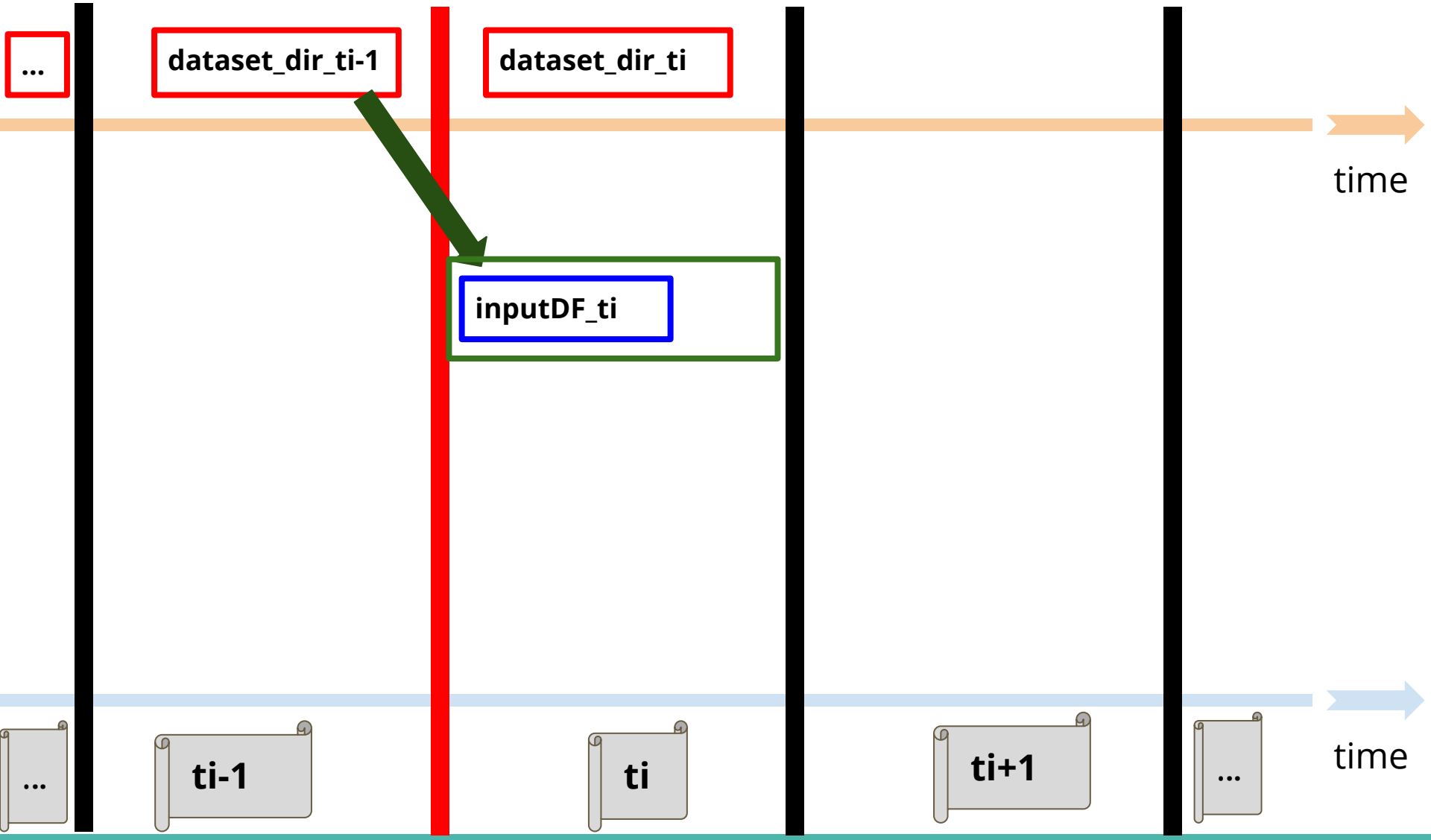
Concept3: Append Mode with Windows



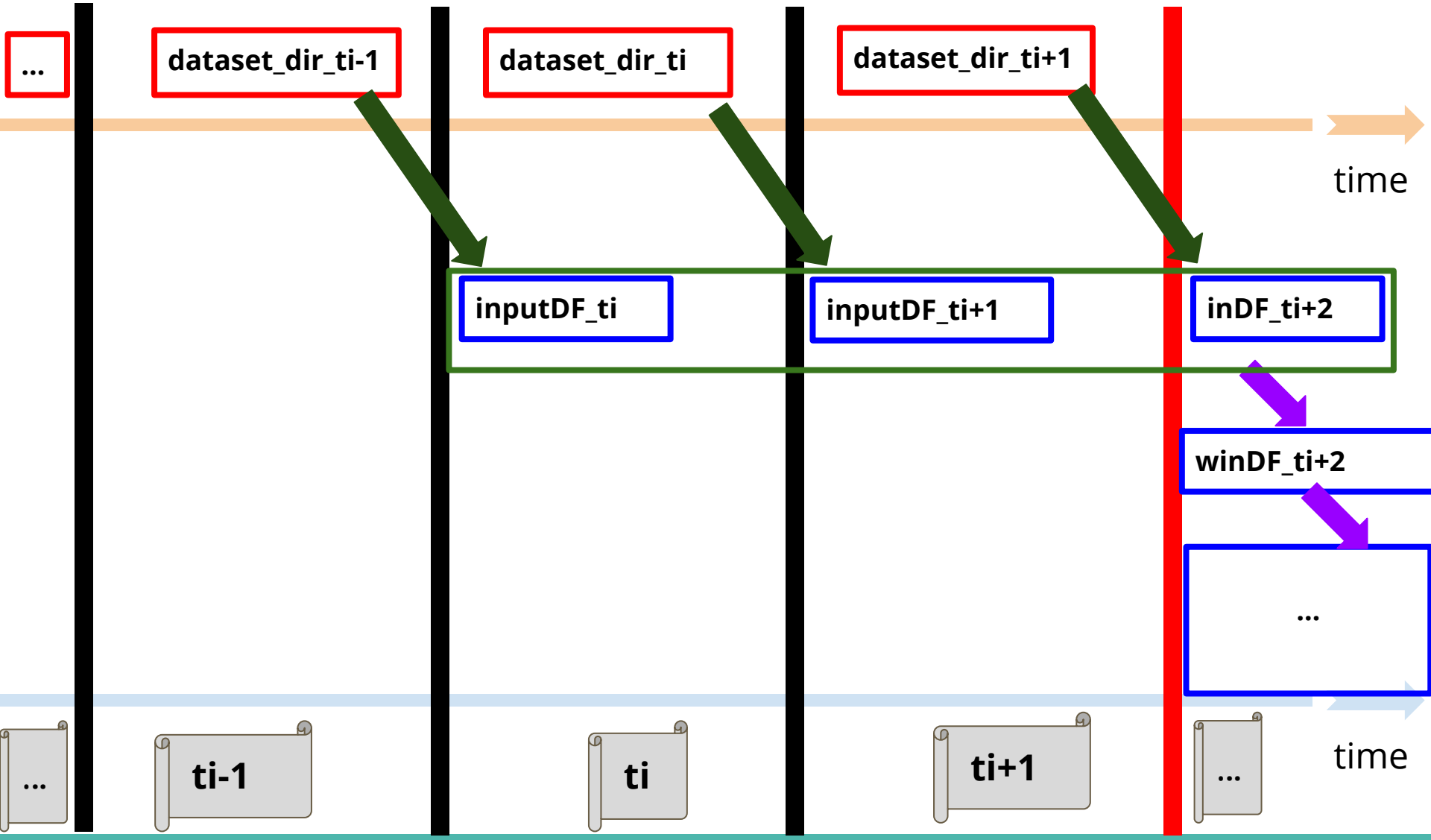
Concept3: Append Mode with Windows



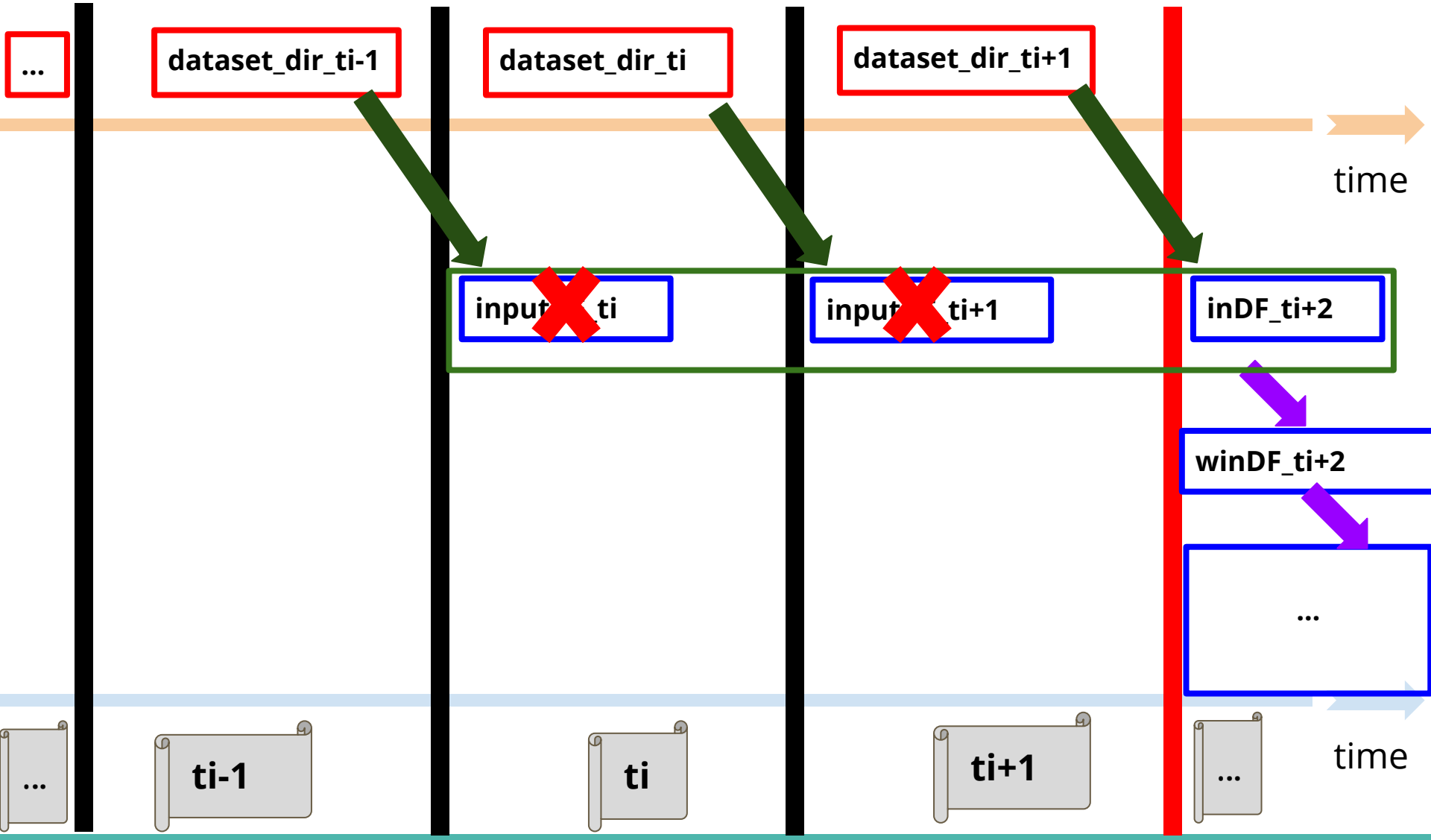
Concept3: Append Mode with Windows



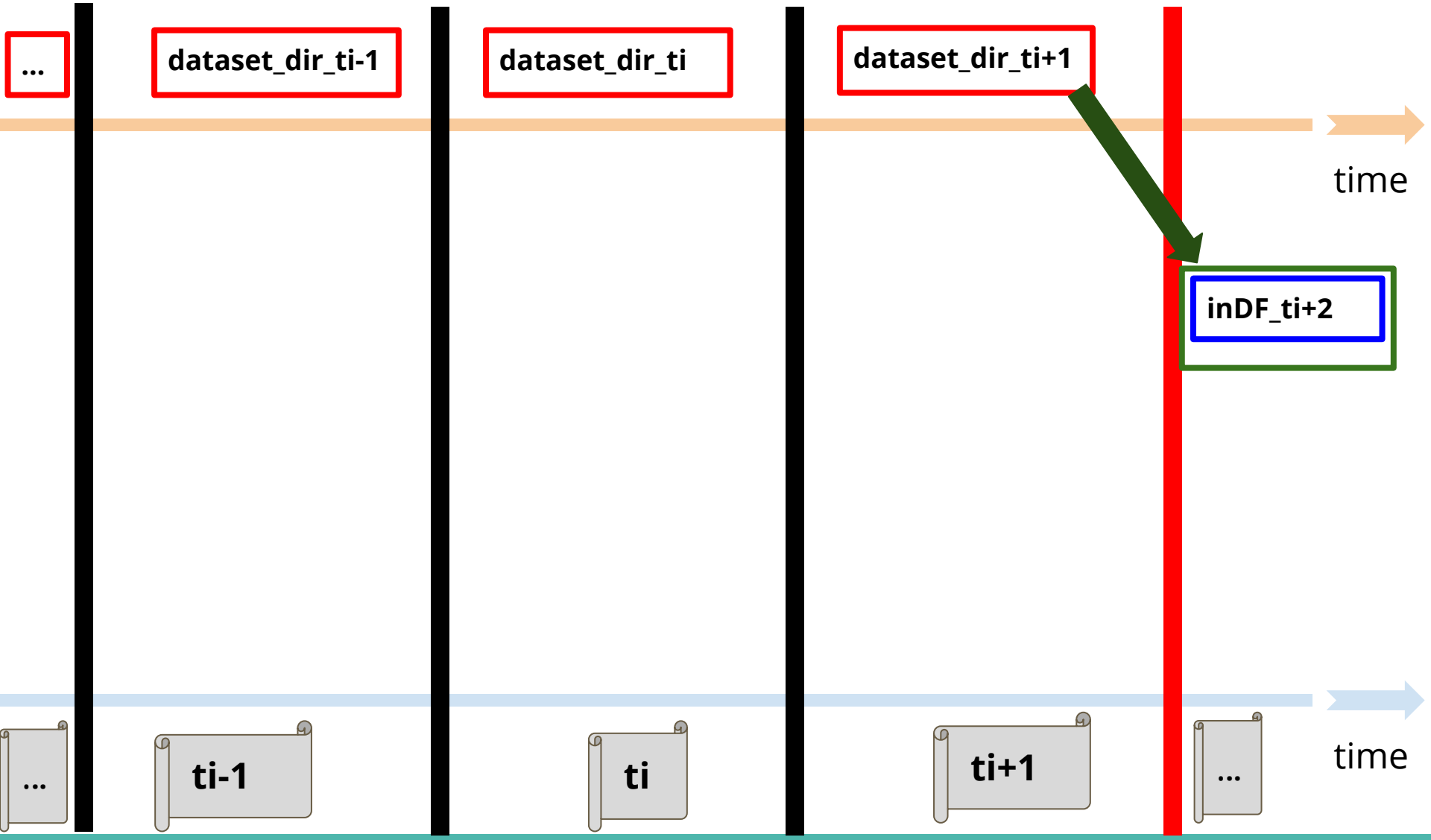
Concept3: Append Mode with Windows



Concept3: Append Mode with Windows

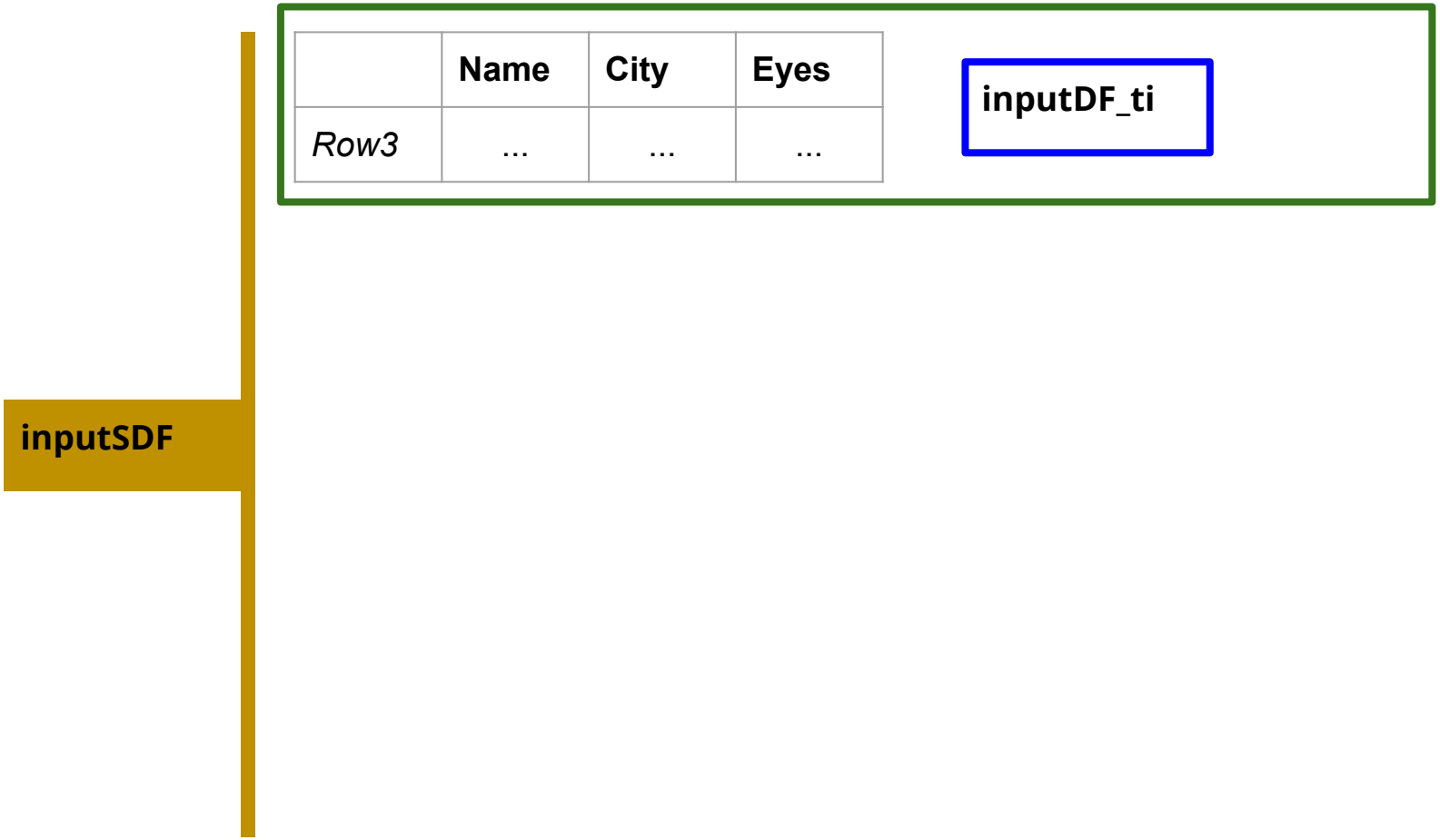


Concept3: Append Mode with Windows



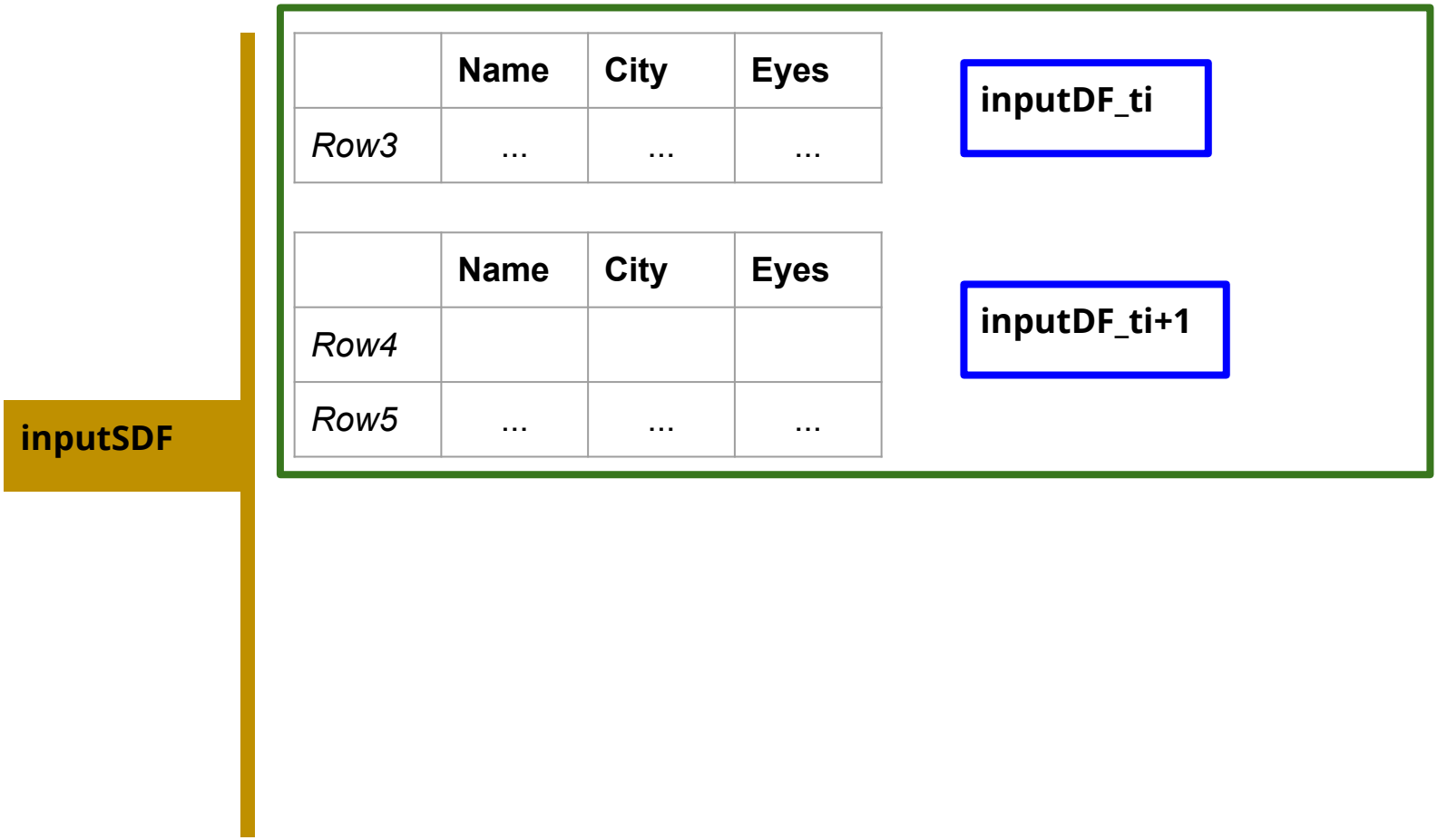
Concept3: Append Mode with Windows

And with the unbound table representation of the SDF...



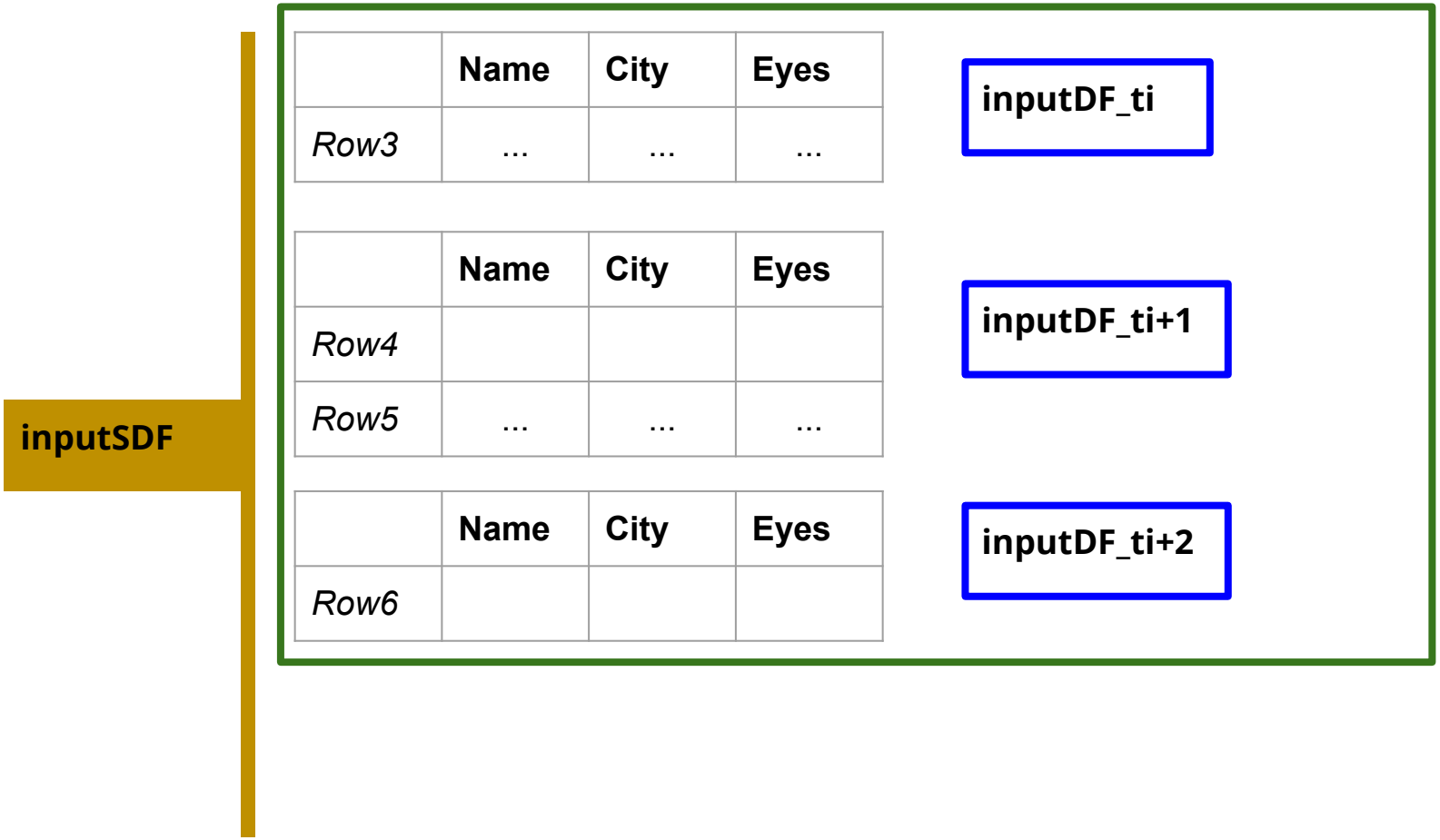
Concept3: Append Mode with Windows

And with the unbound table representation of the SDF...



Concept3: Append Mode with Windows

And with the unbound table representation of the SDF...

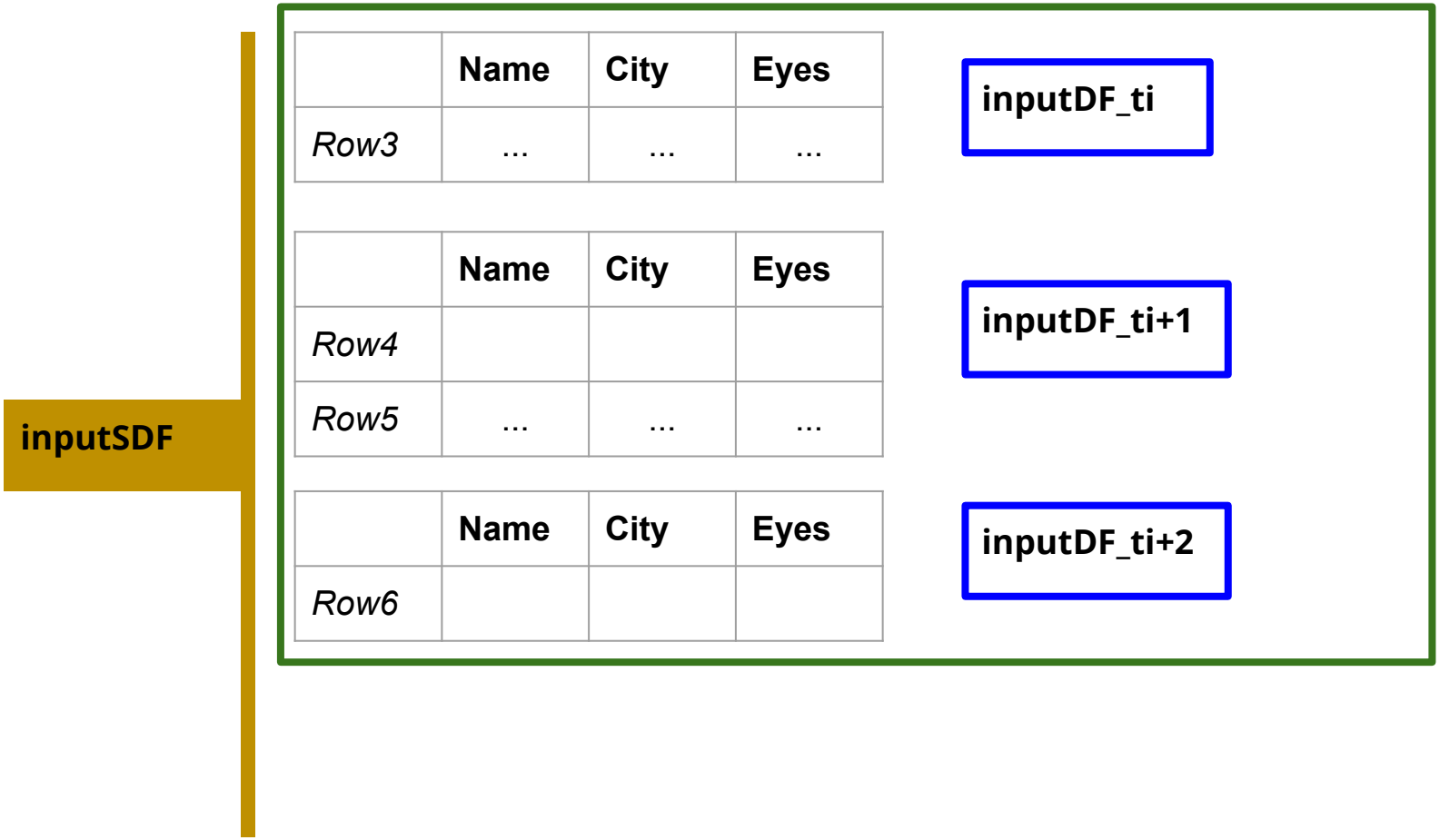


Concept3: Append Mode with Windows

And with the unbound table representation of the SDF...

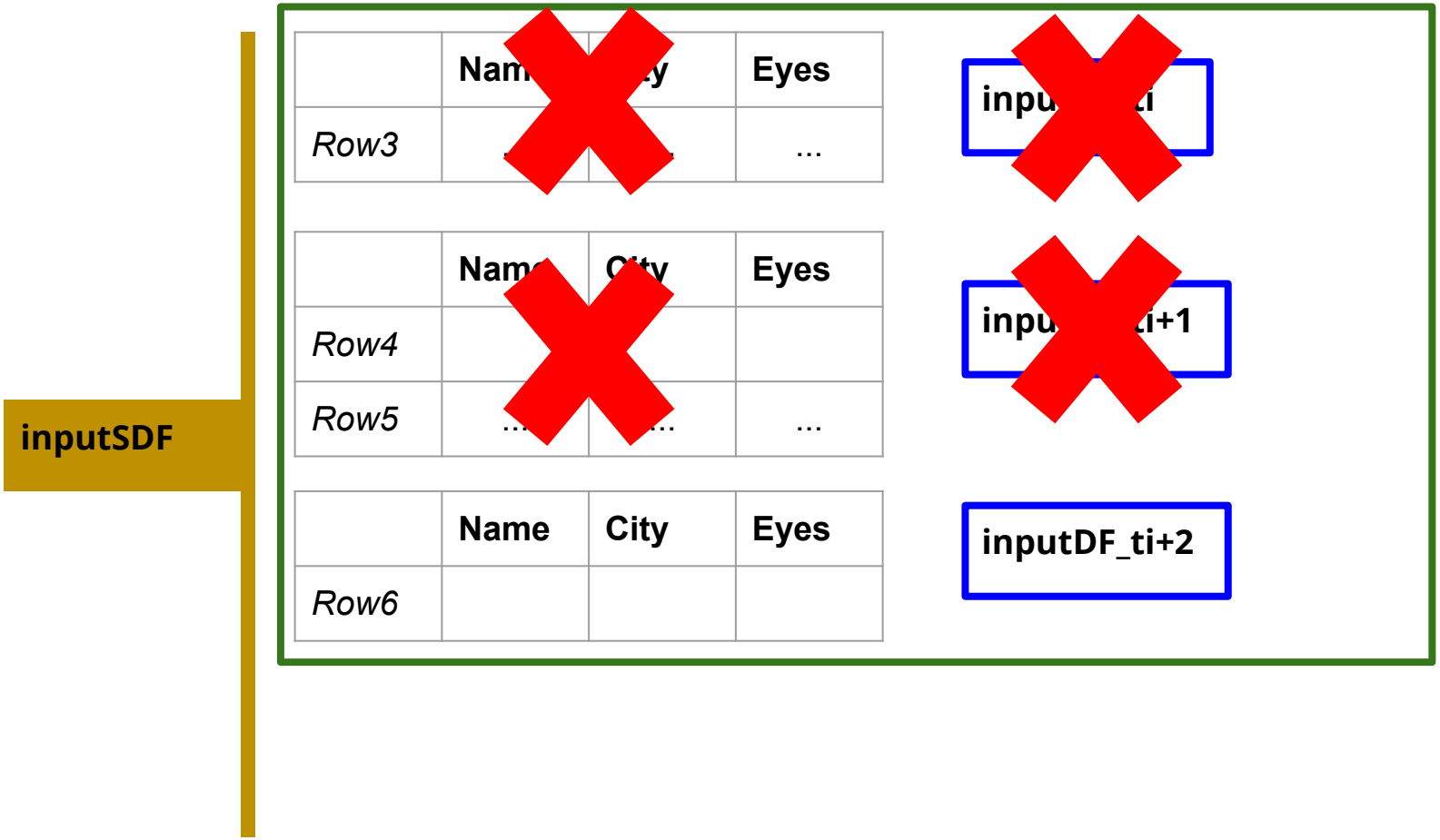
Concept3: Append Mode with Windows

And with the unbound table representation of the SDF...



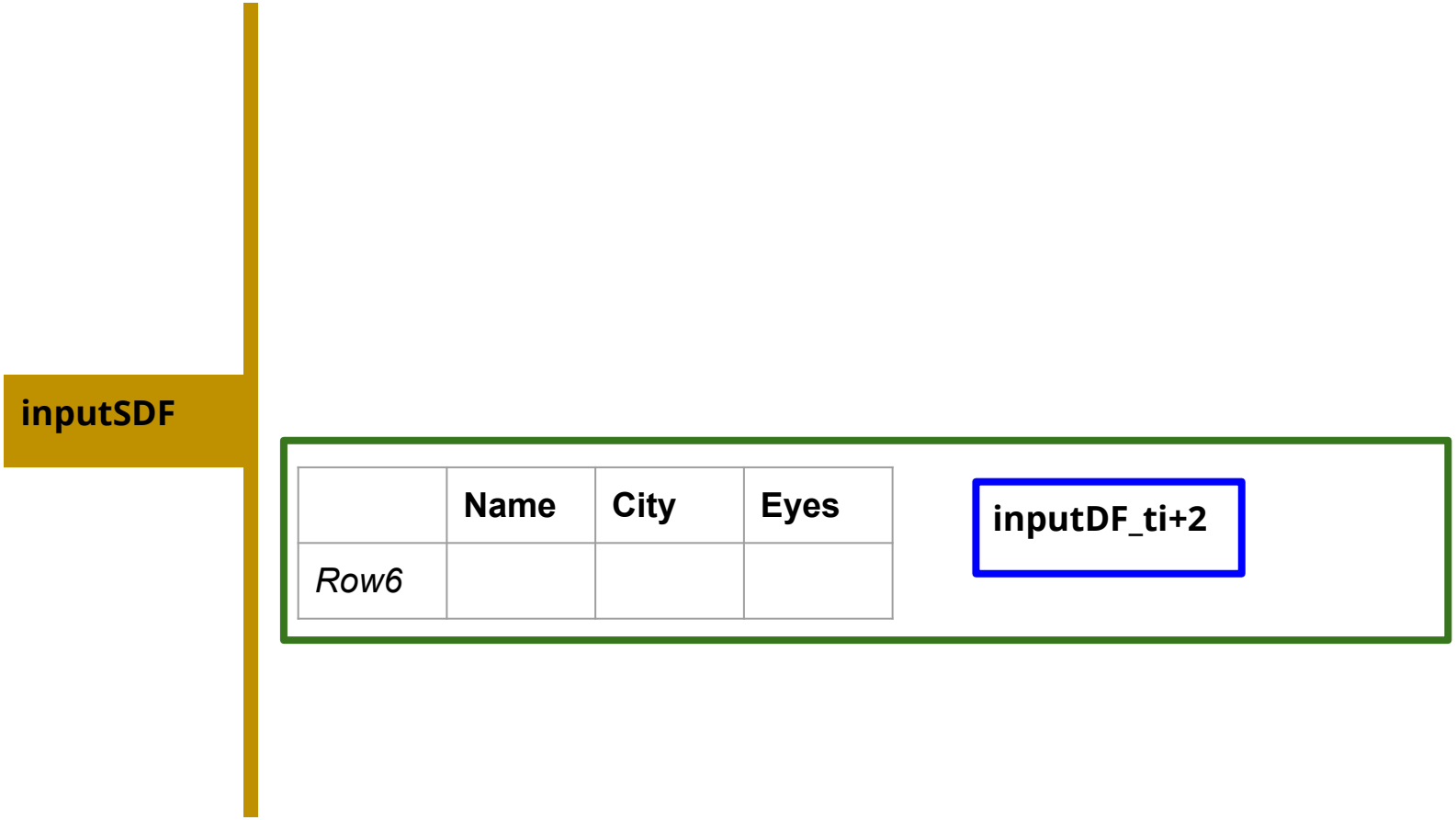
Concept3: Append Mode with Windows

And with the unbound table representation of the SDF...



Concept3: Append Mode with Windows

And with the unbound table representation of the SDF...



Concept3: Append Mode with Windows

Concept 3:

This behaviour is called
a **SDF** in **Append Mode**
reasoning over time window!

Outline

1. Setting Up the Context.
2. From DStream to SDF.
   a. Concept1: SDF.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
3. Practising with the Concepts.

Concept4: Complete Mode

A **DStream** uses update-based operations to aggregate results for all the **wagons** processed so far.

Concept4: Complete Mode

A **DStream** uses update-based operations to aggregate results for all the **wagons** processed so far.

Concept4: Complete Mode

A **DStream** uses update-based operations to aggregate results for all the **wagons** processed so far.

Concept4: Complete Mode

A **DStream** uses update-based operations to aggregate results for all the **wagons** processed so far.

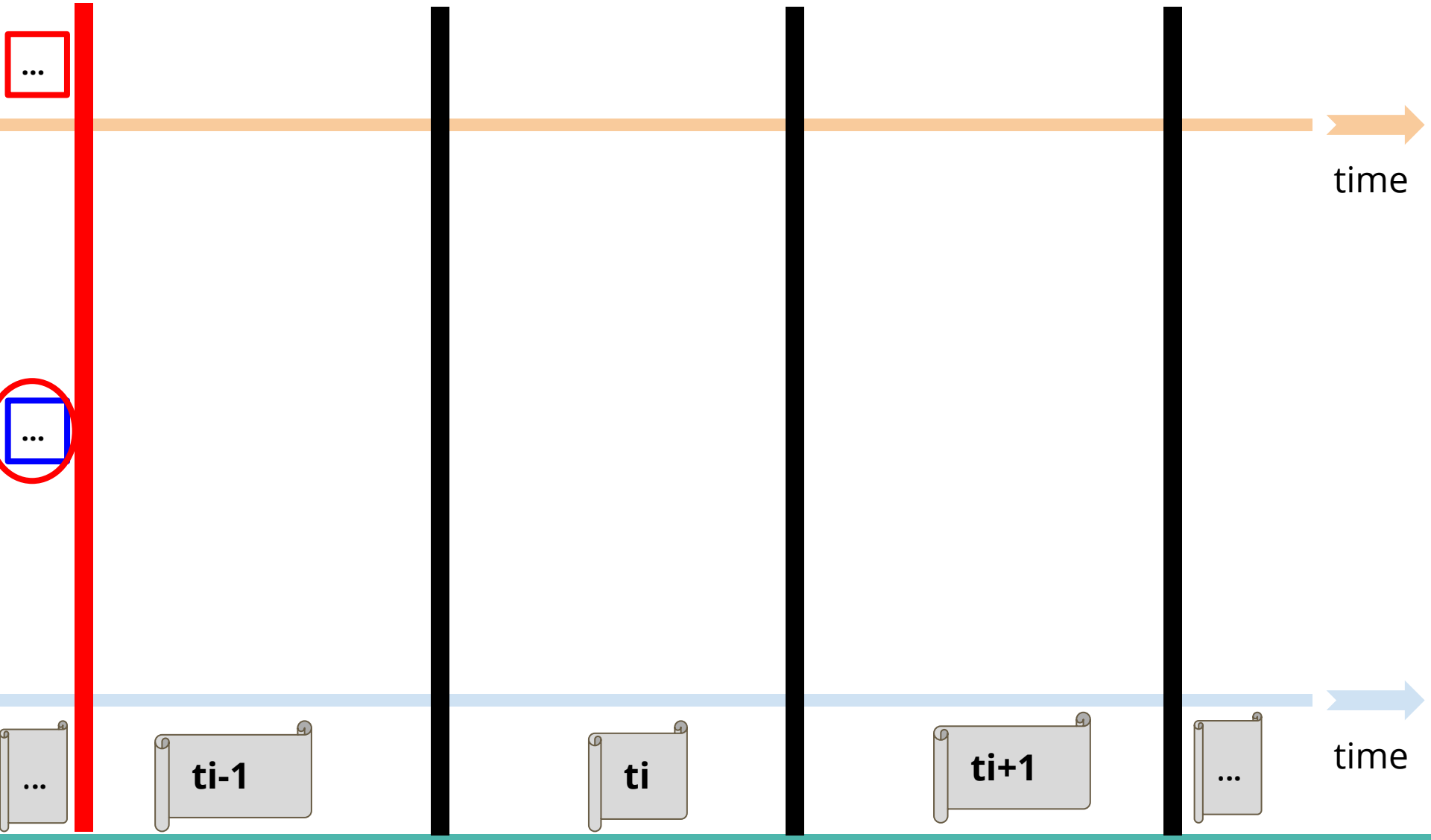
Concept4: Complete Mode

A **DStream** uses update-based operations to aggregate results for all the **wagons** processed so far.

Concept4: Complete Mode

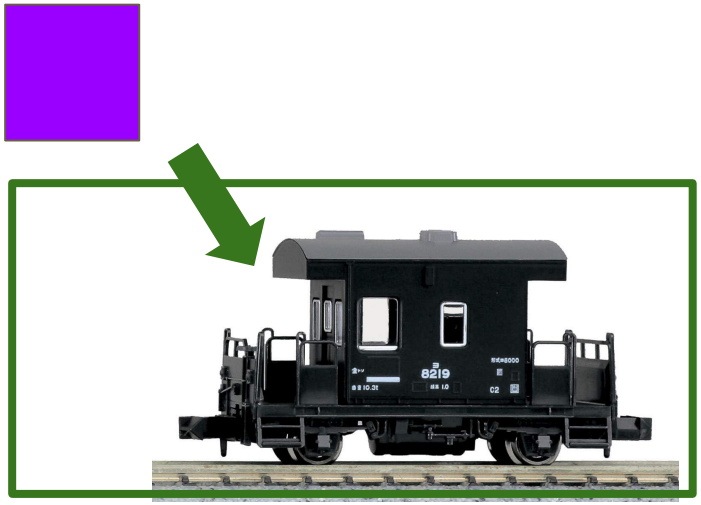
On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.

Concept4: Complete Mode



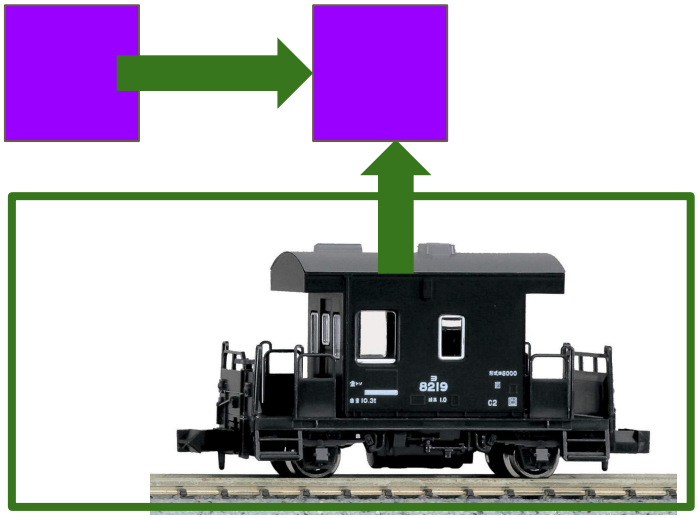
Concept4: Complete Mode

On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.



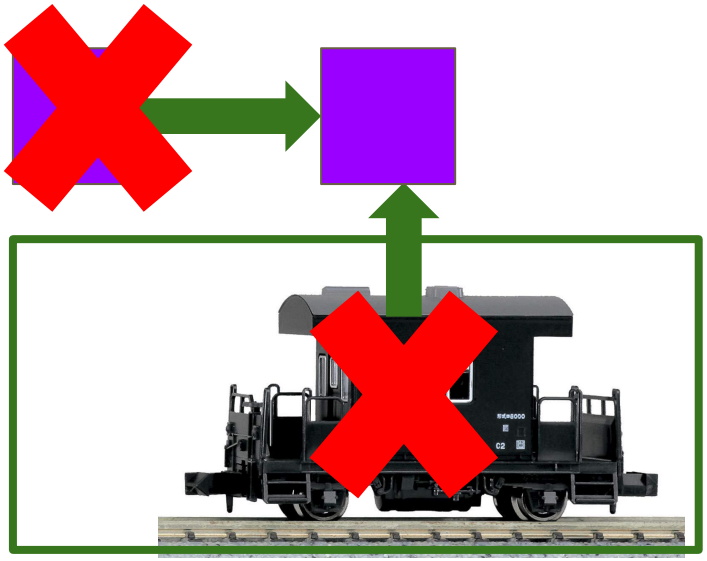
Concept4: Complete Mode

On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.



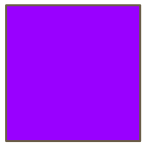
Concept4: Complete Mode

On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.

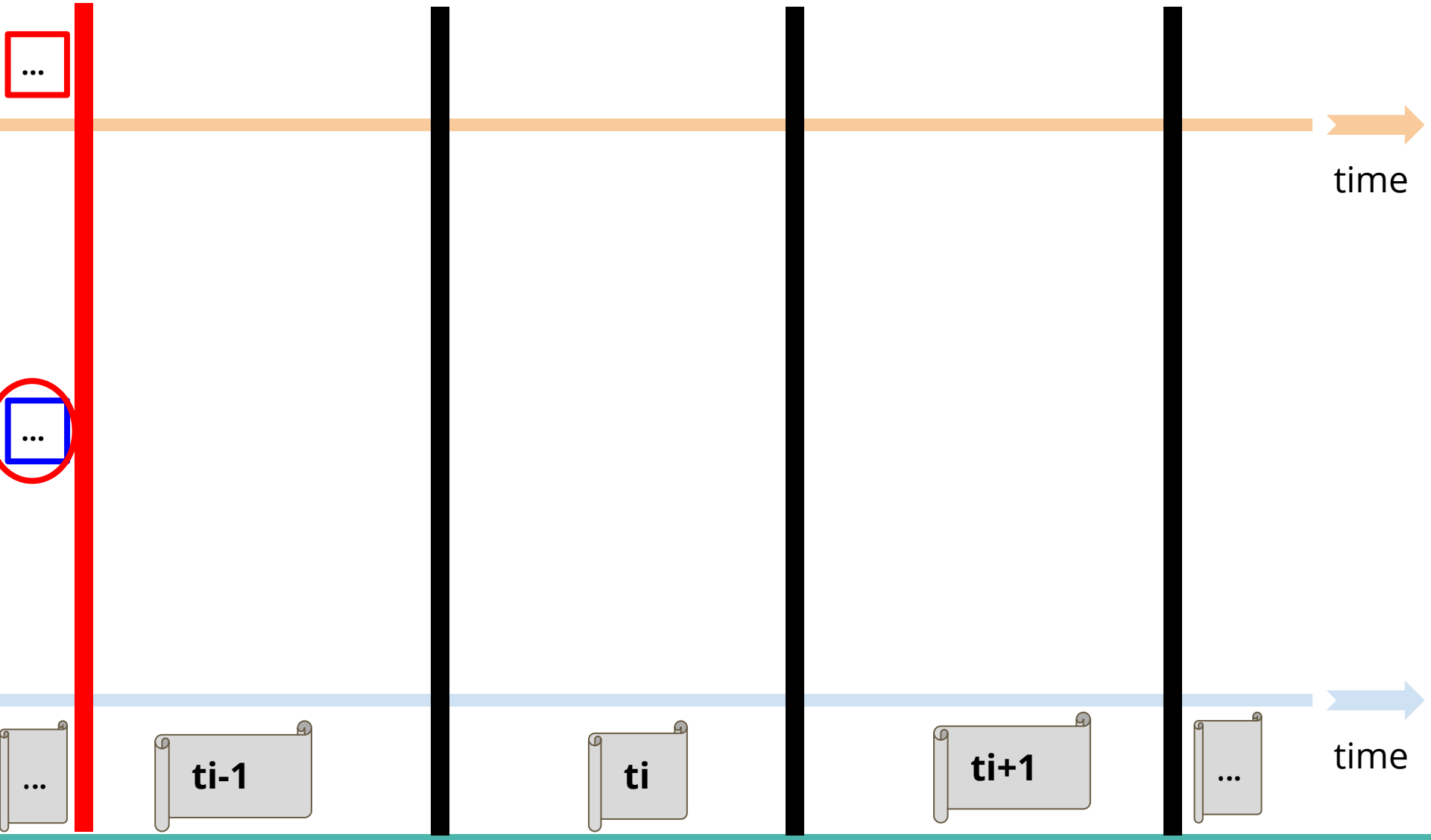


Concept4: Complete Mode

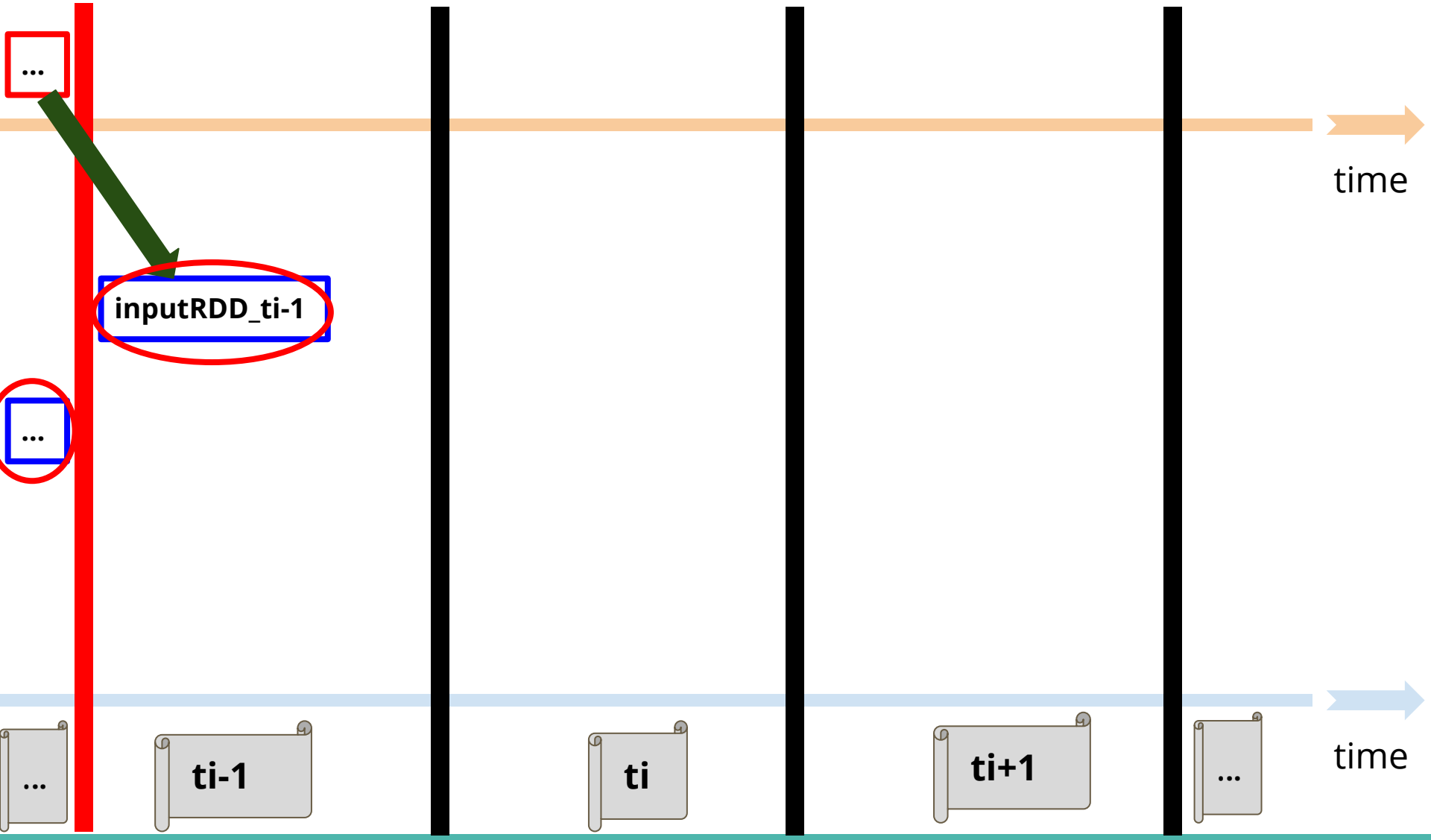
On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.



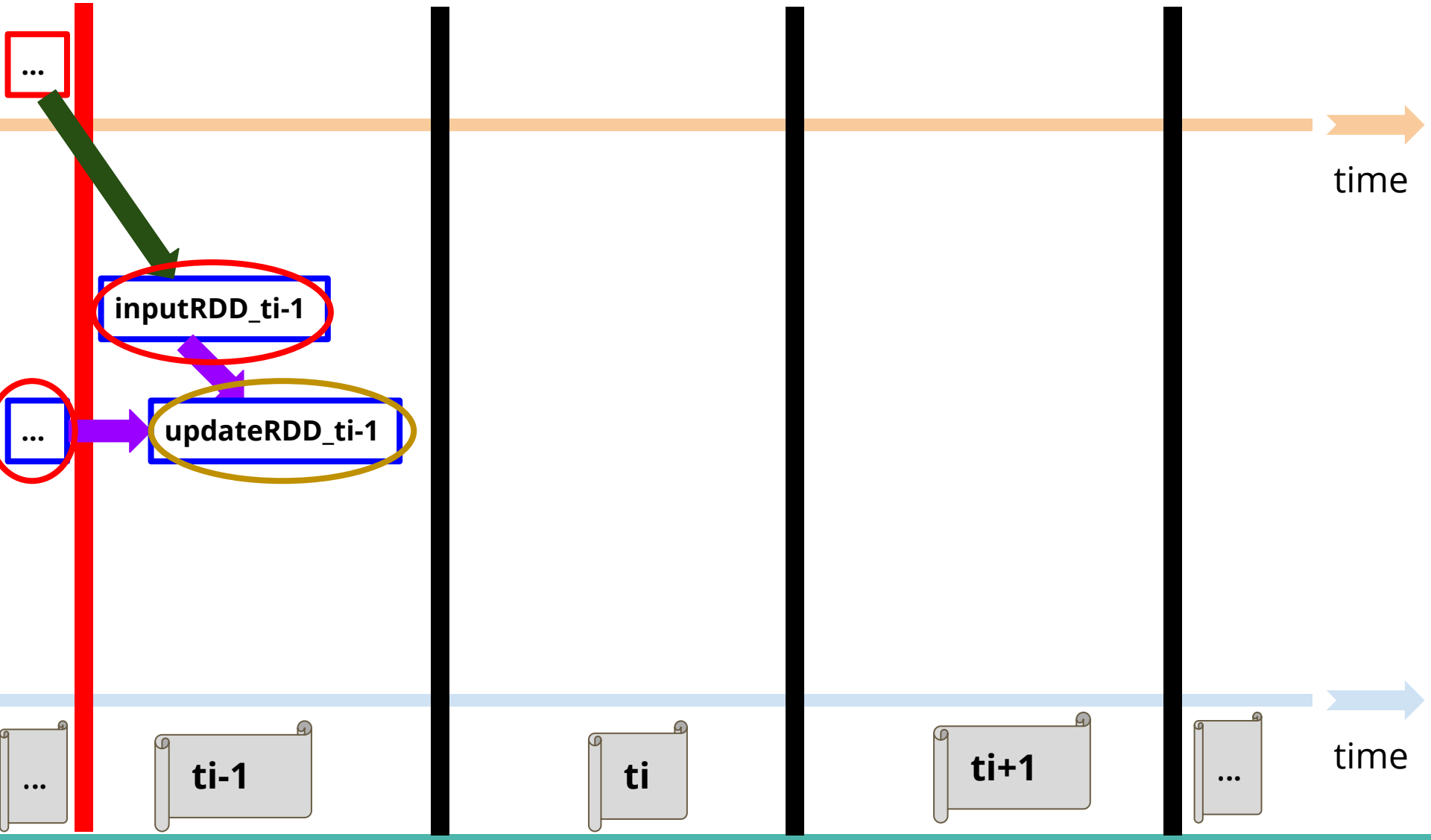
Concept4: Complete Mode



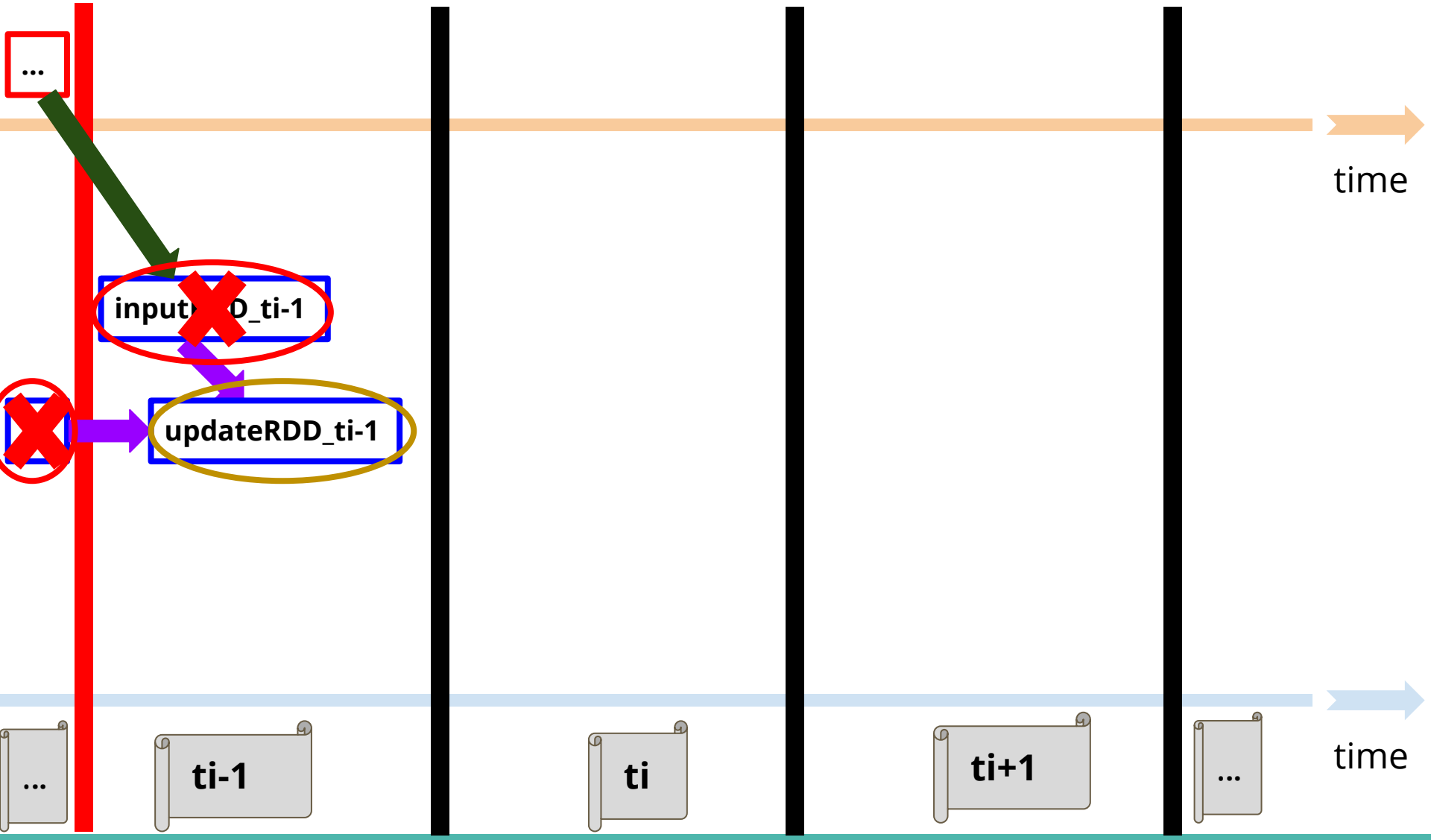
Concept4: Complete Mode



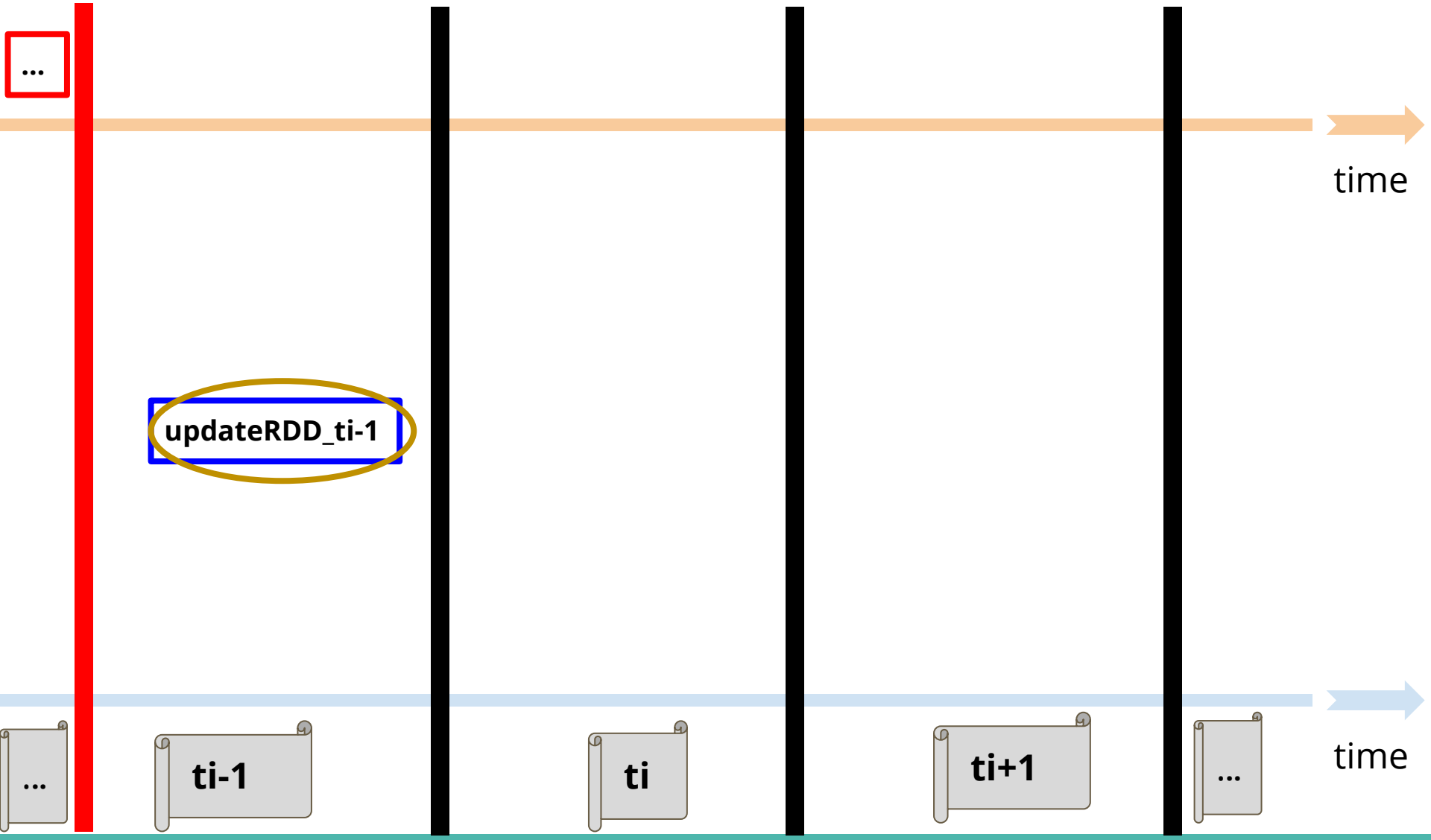
Concept4: Complete Mode



Concept4: Complete Mode

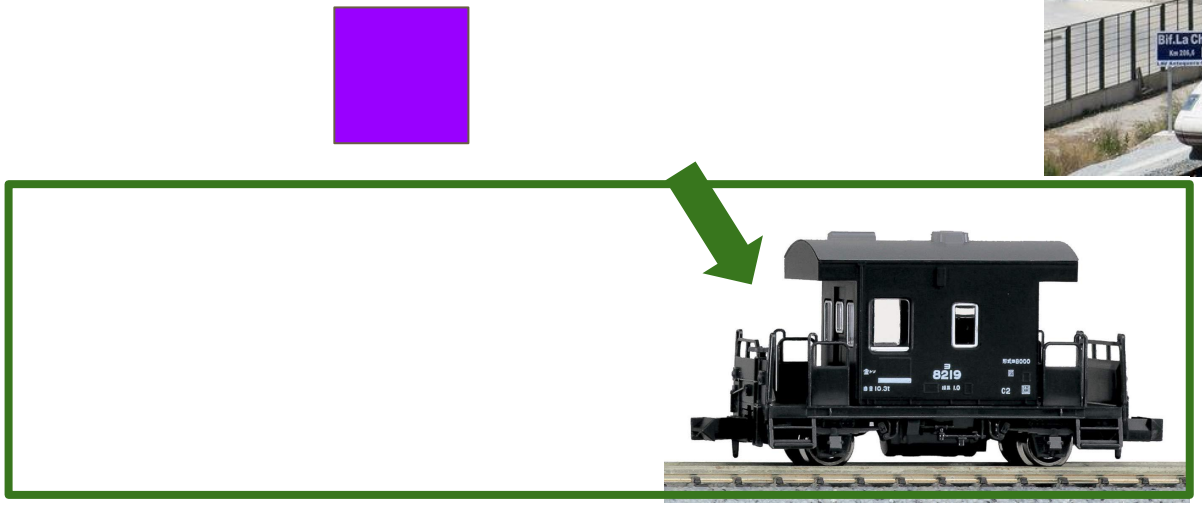


Concept4: Complete Mode



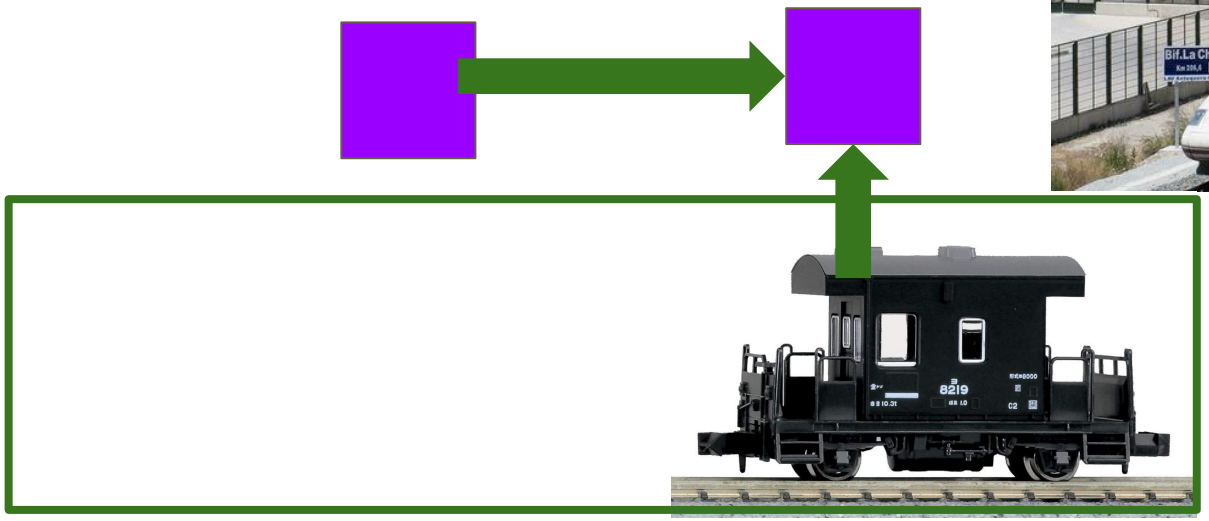
Concept4: Complete Mode

On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.



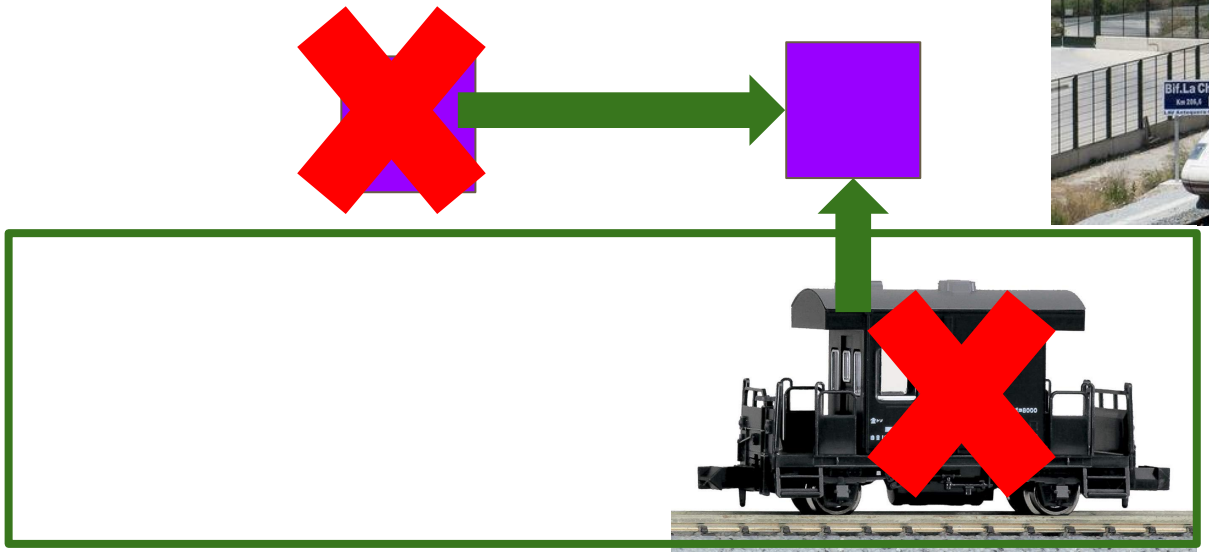
Concept4: Complete Mode

On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.



Concept4: Complete Mode

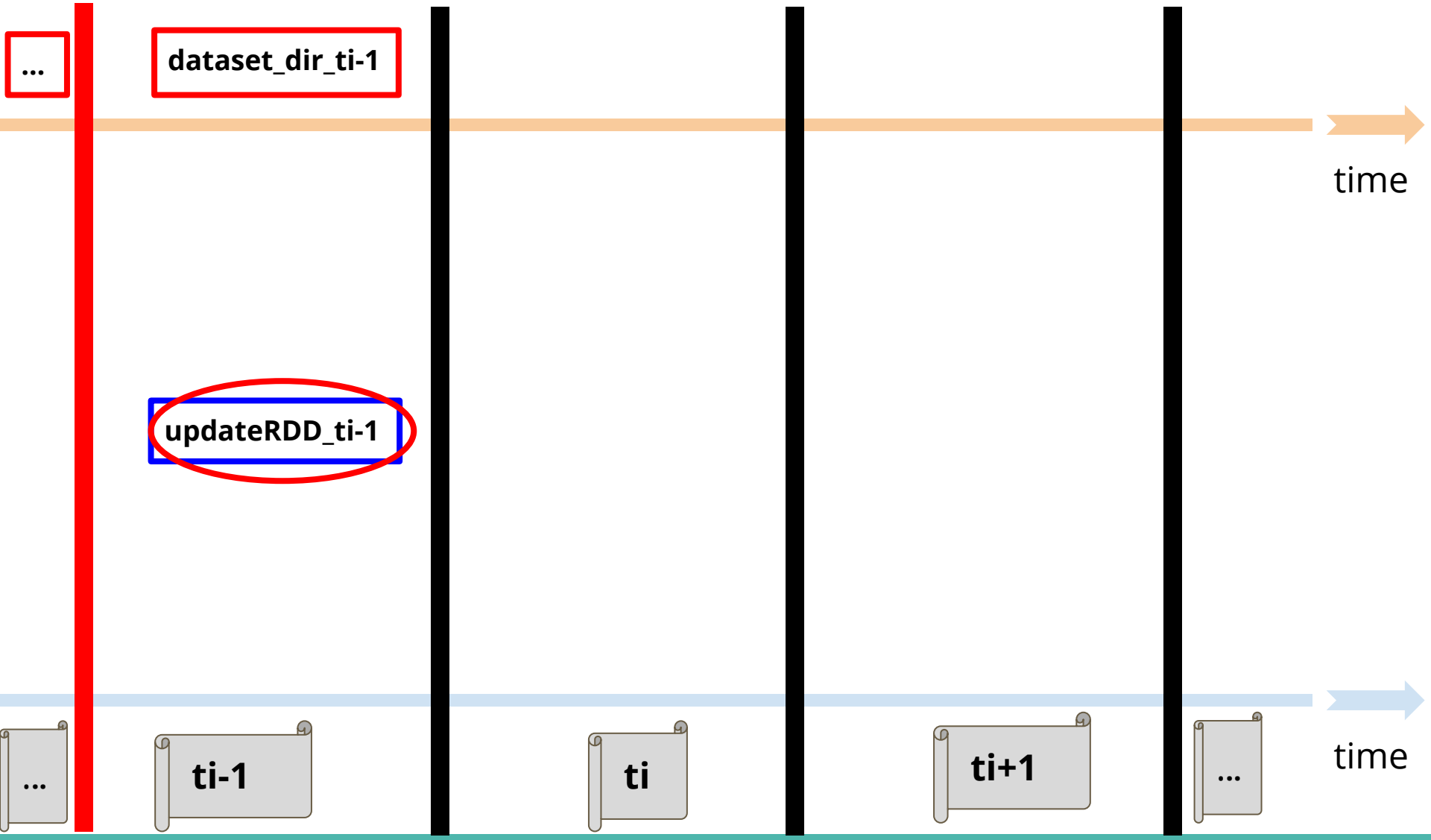
On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.



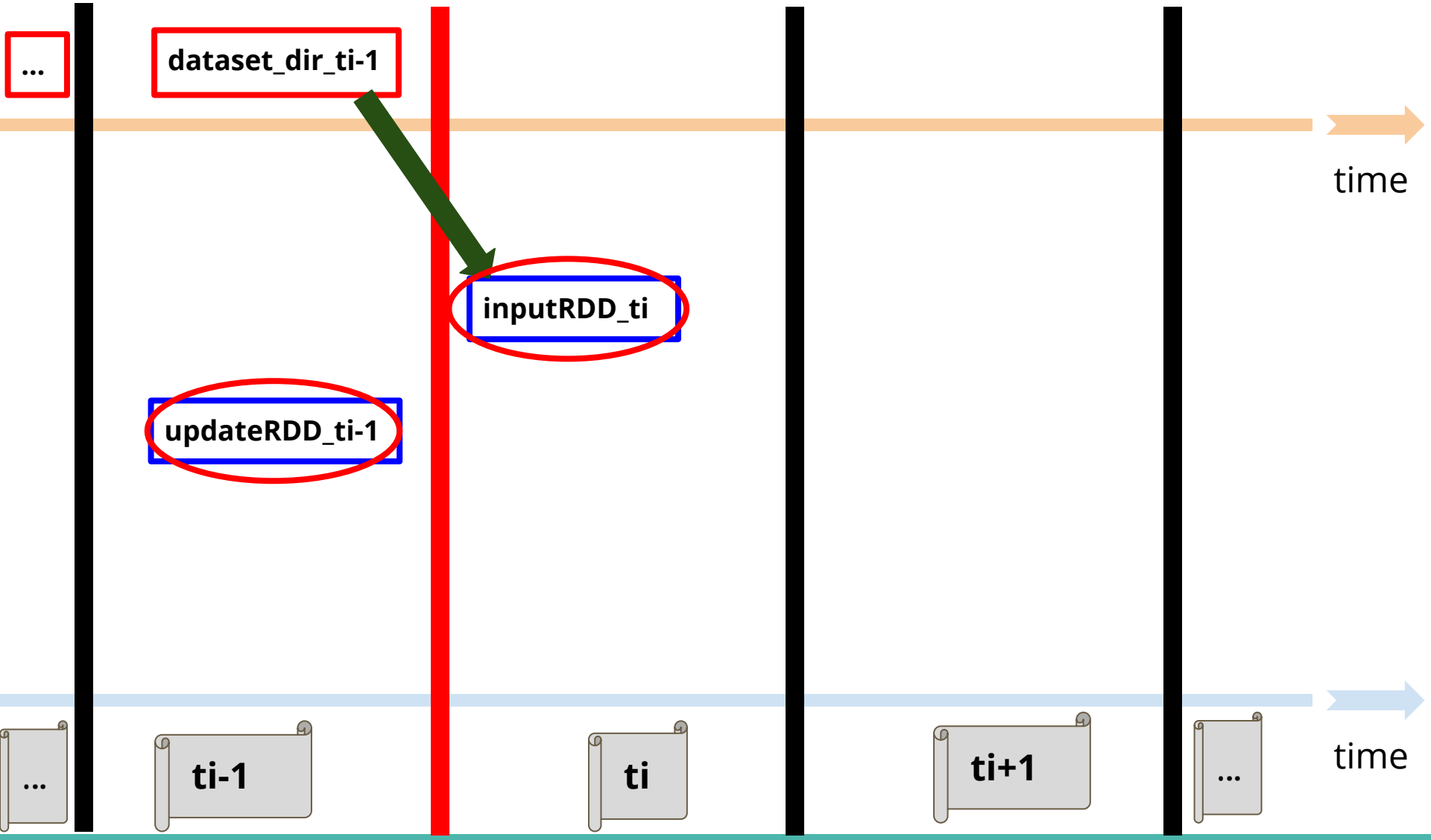
Concept4: Complete Mode

On doing so, the DStream does not keep all its wagons.
Instead, it keeps in memory the current aggregated value,
which is then aggregated with each new wagon arriving over time.

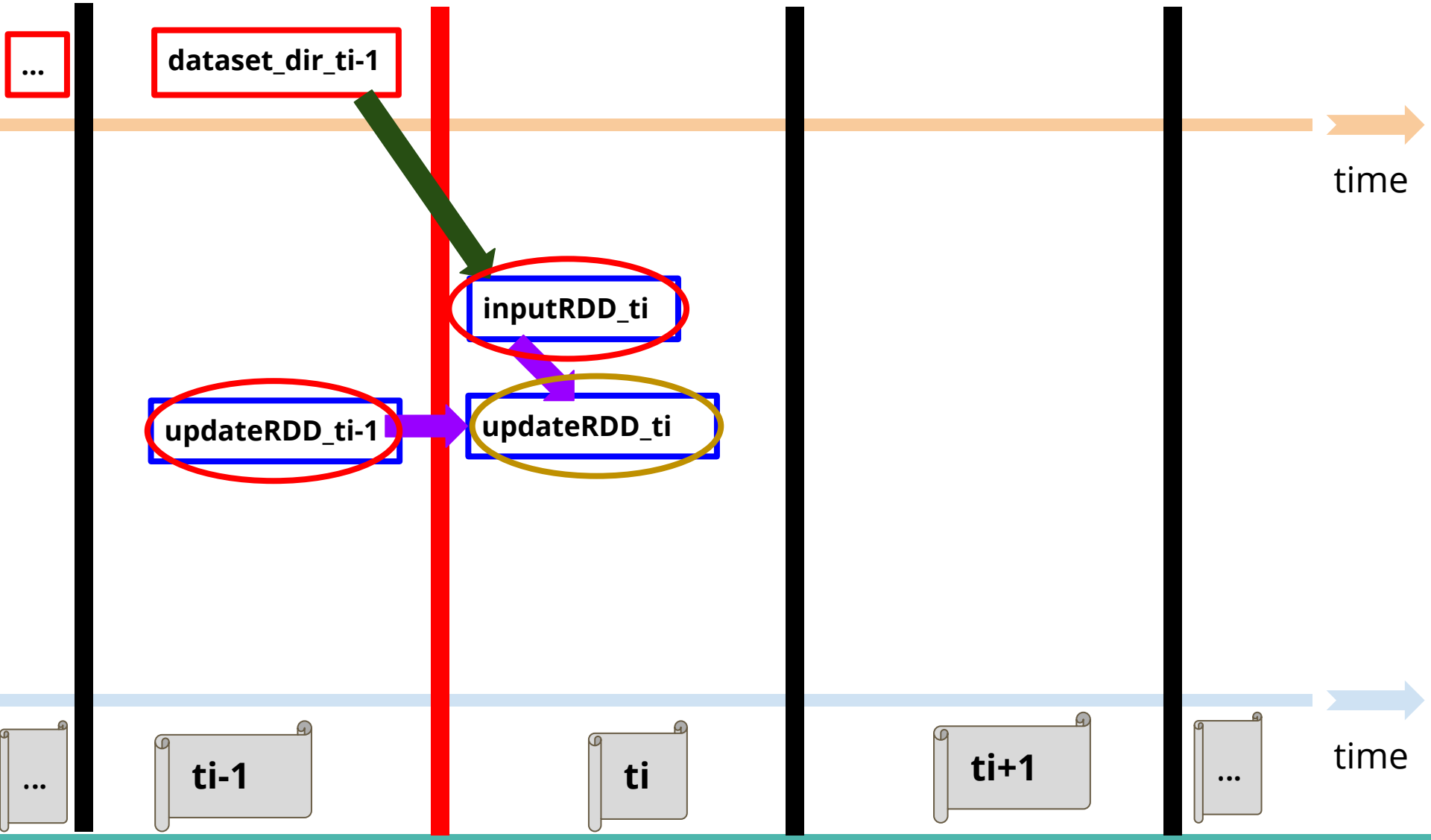
Concept4: Complete Mode



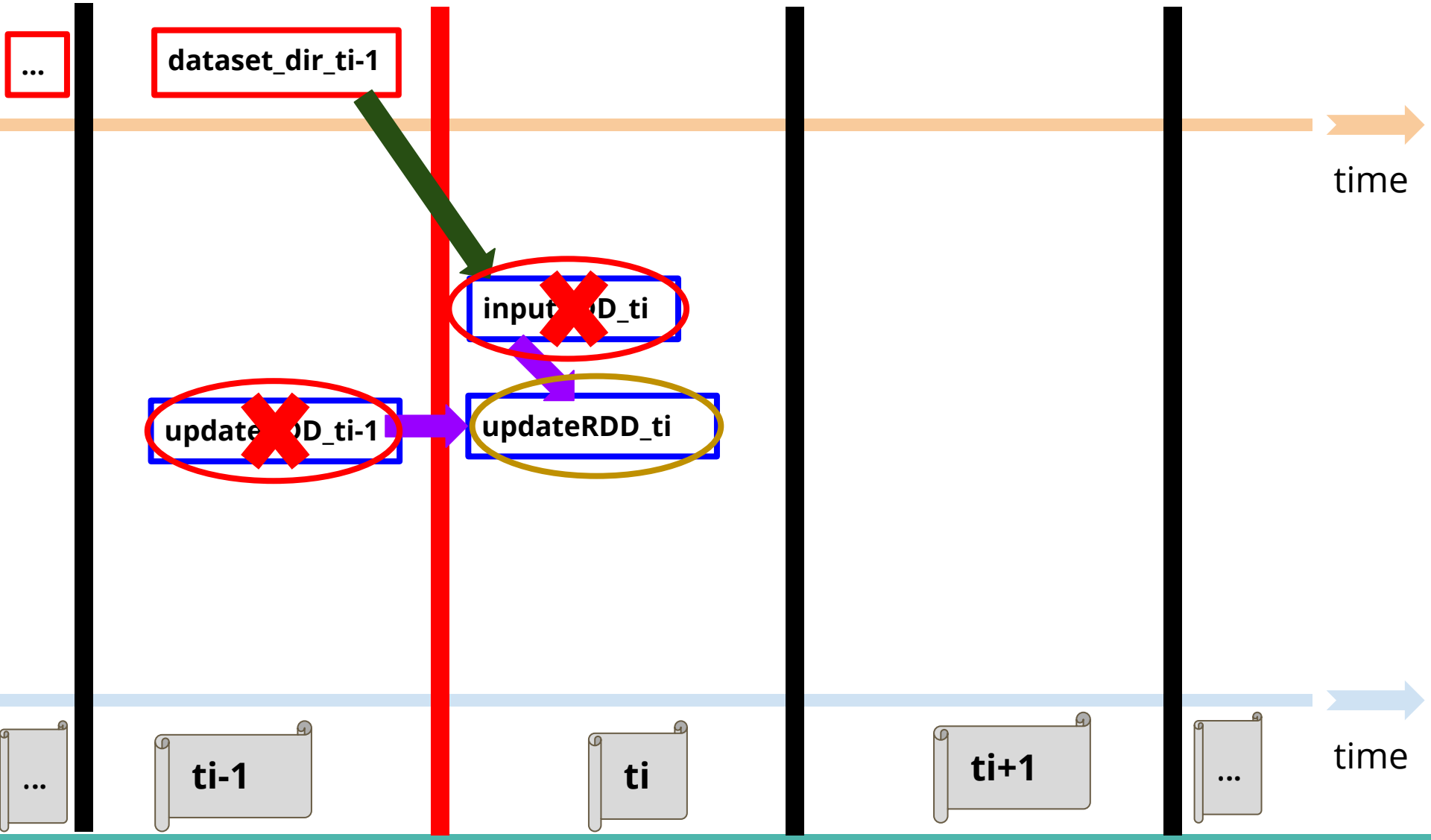
Concept4: Complete Mode



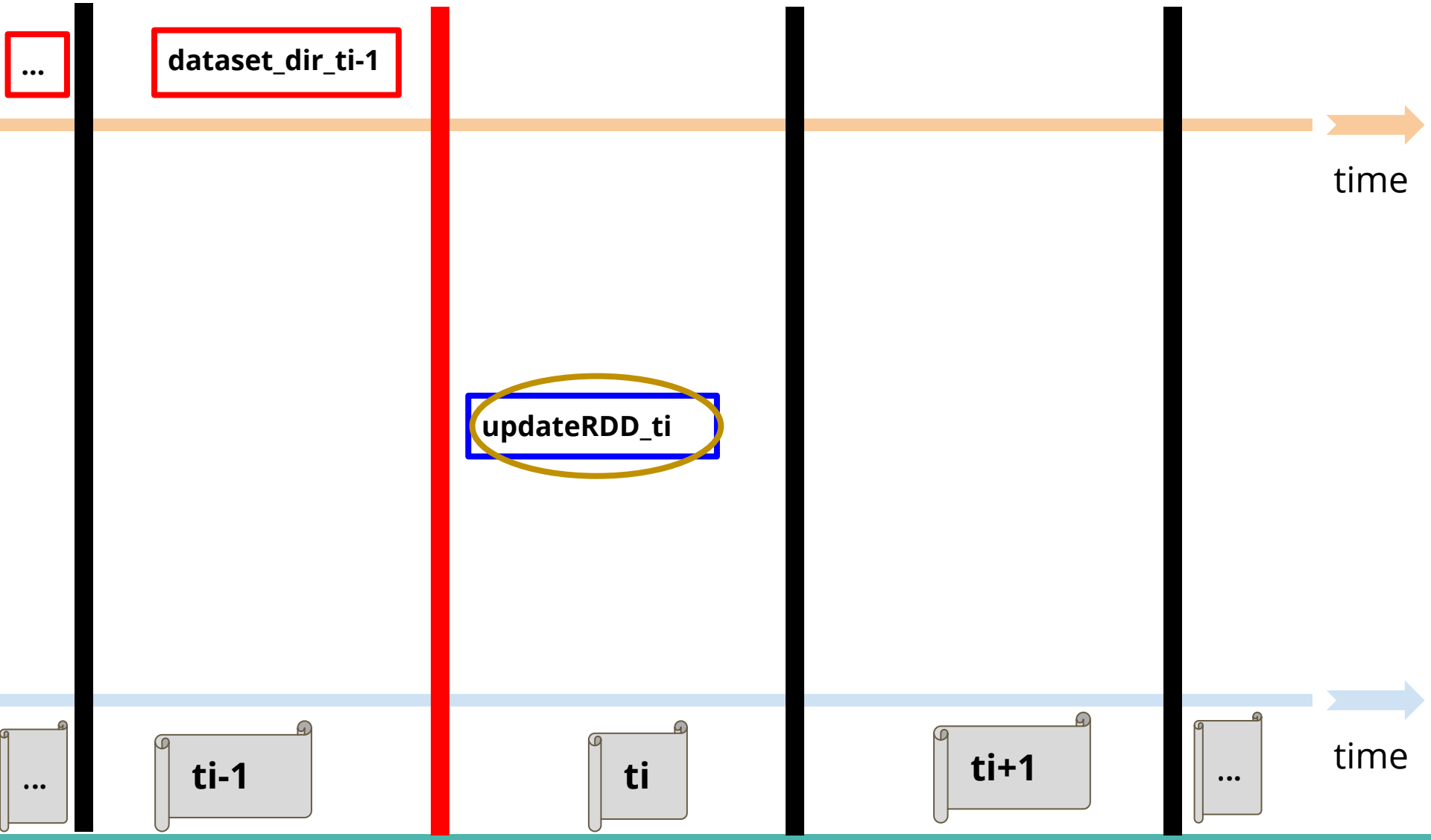
Concept4: Complete Mode



Concept4: Complete Mode



Concept4: Complete Mode

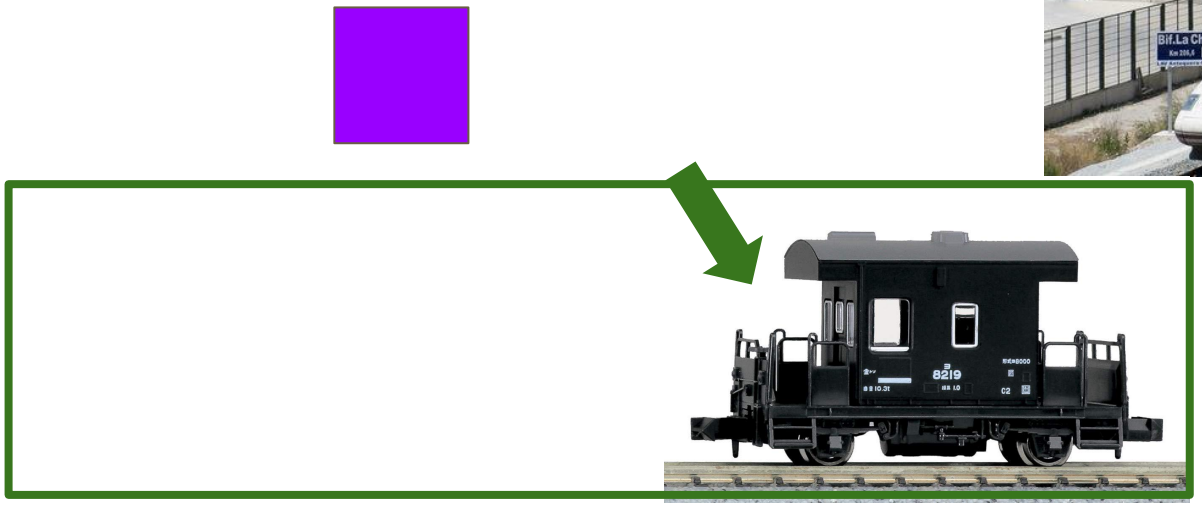


Concept4: Complete Mode

The same behaviour applies to **SDF**!

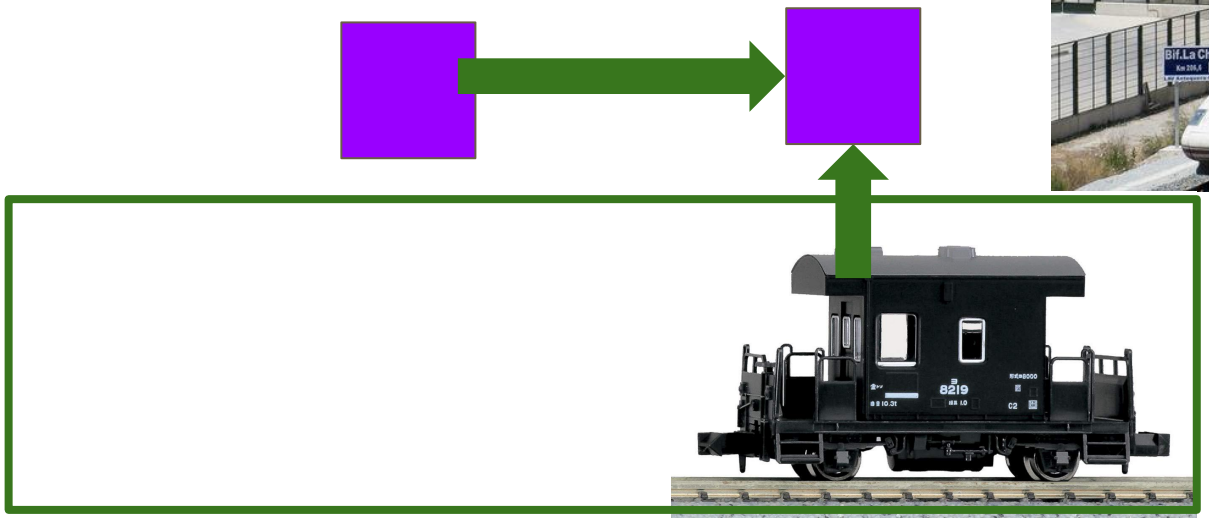
Concept4: Complete Mode

Coming back to our example



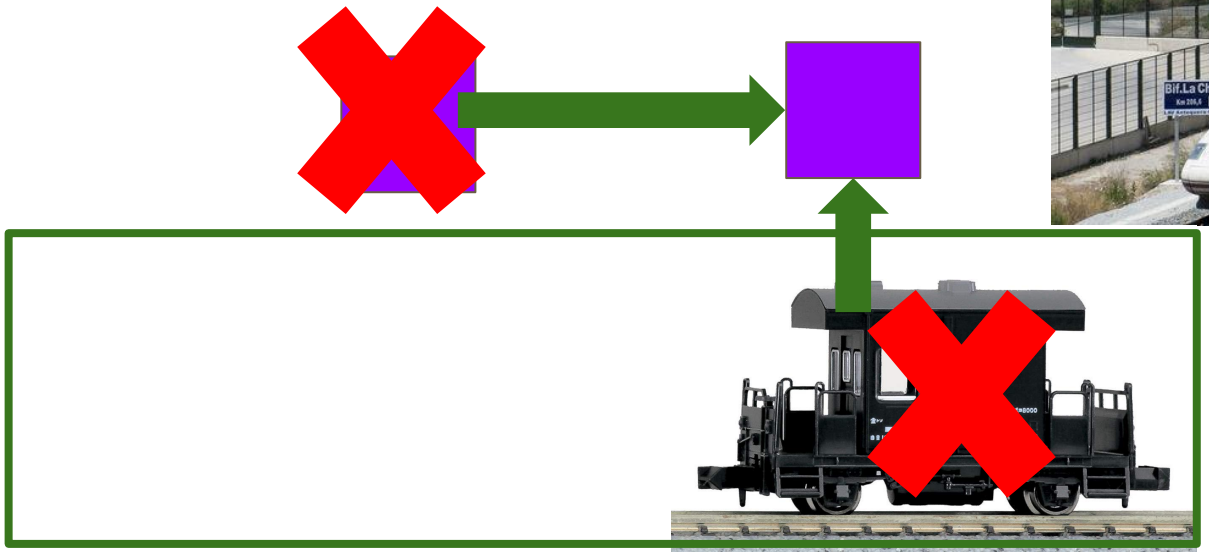
Concept4: Complete Mode

Coming back to our example



Concept4: Complete Mode

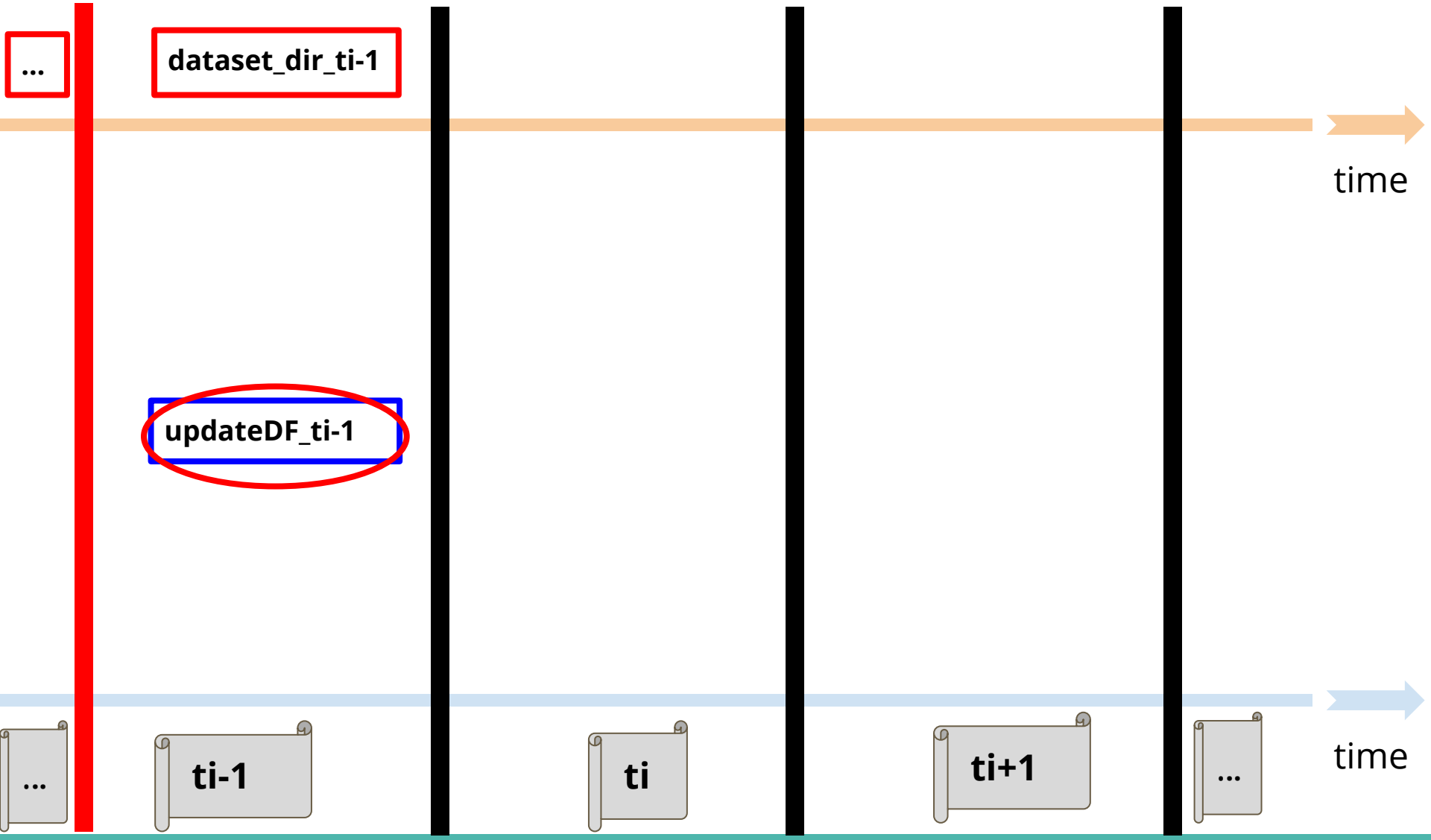
Coming back to our example



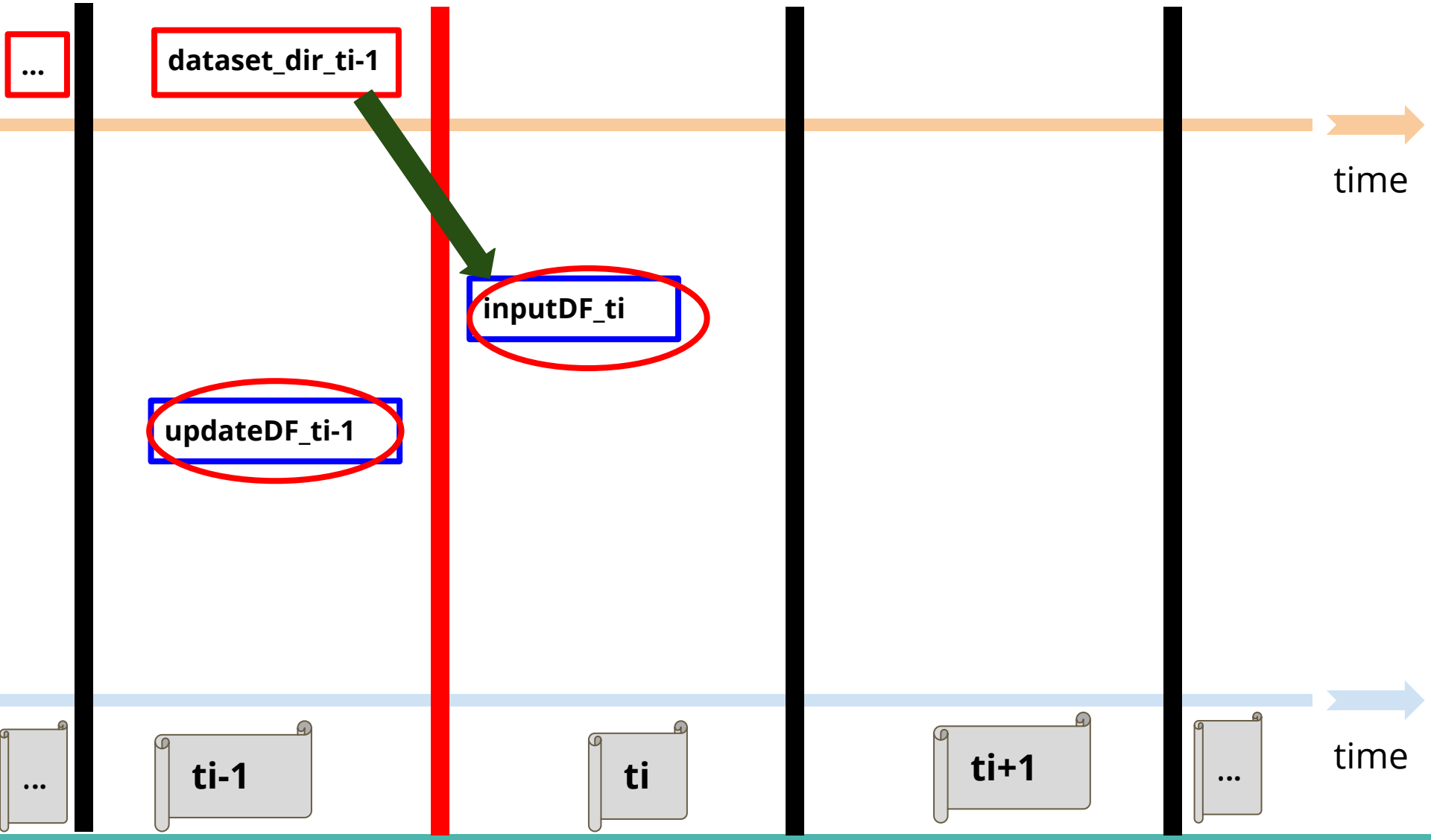
Concept4: Complete Mode

Coming back to our example

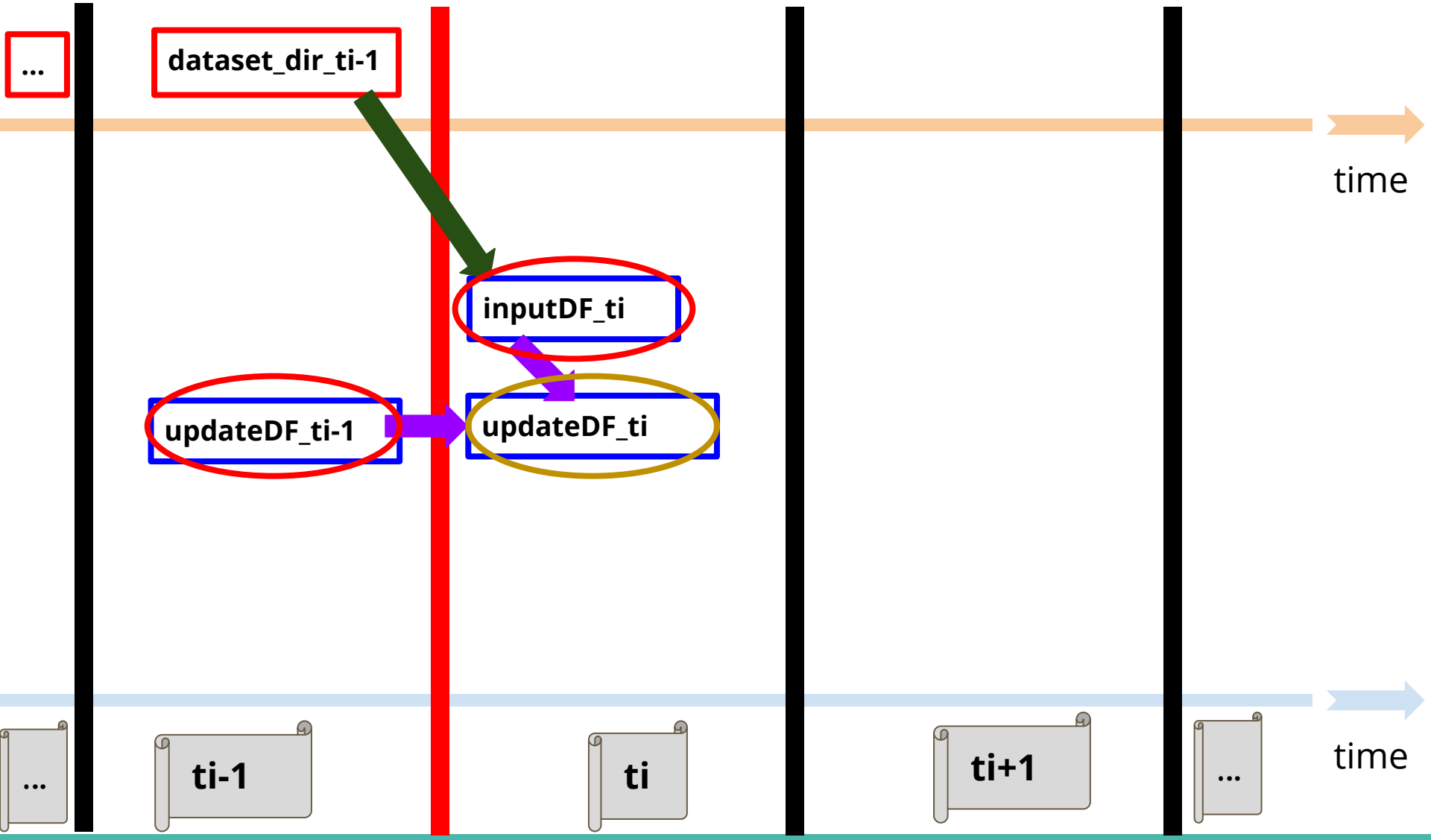
Concept4: Complete Mode



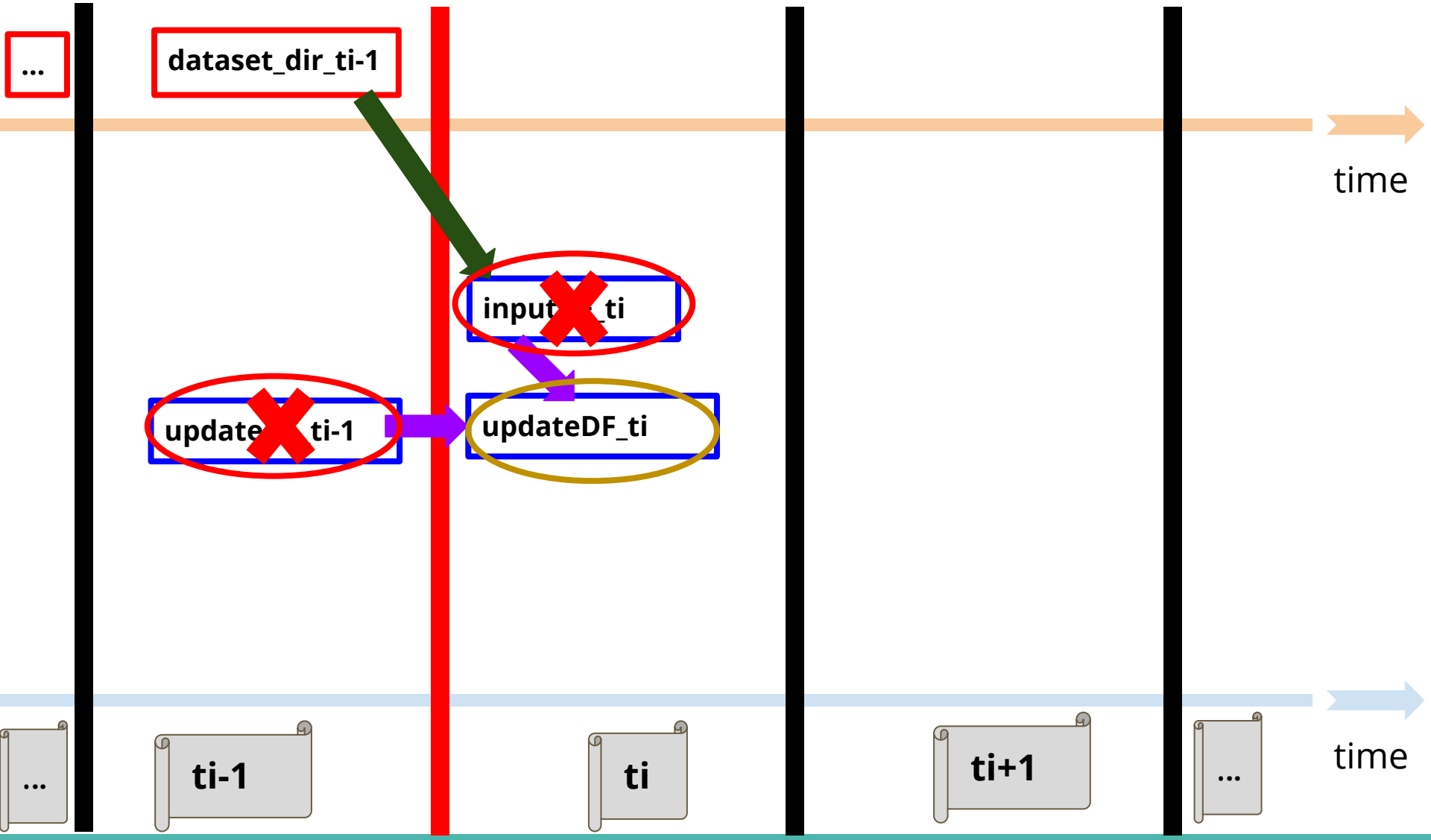
Concept4: Complete Mode



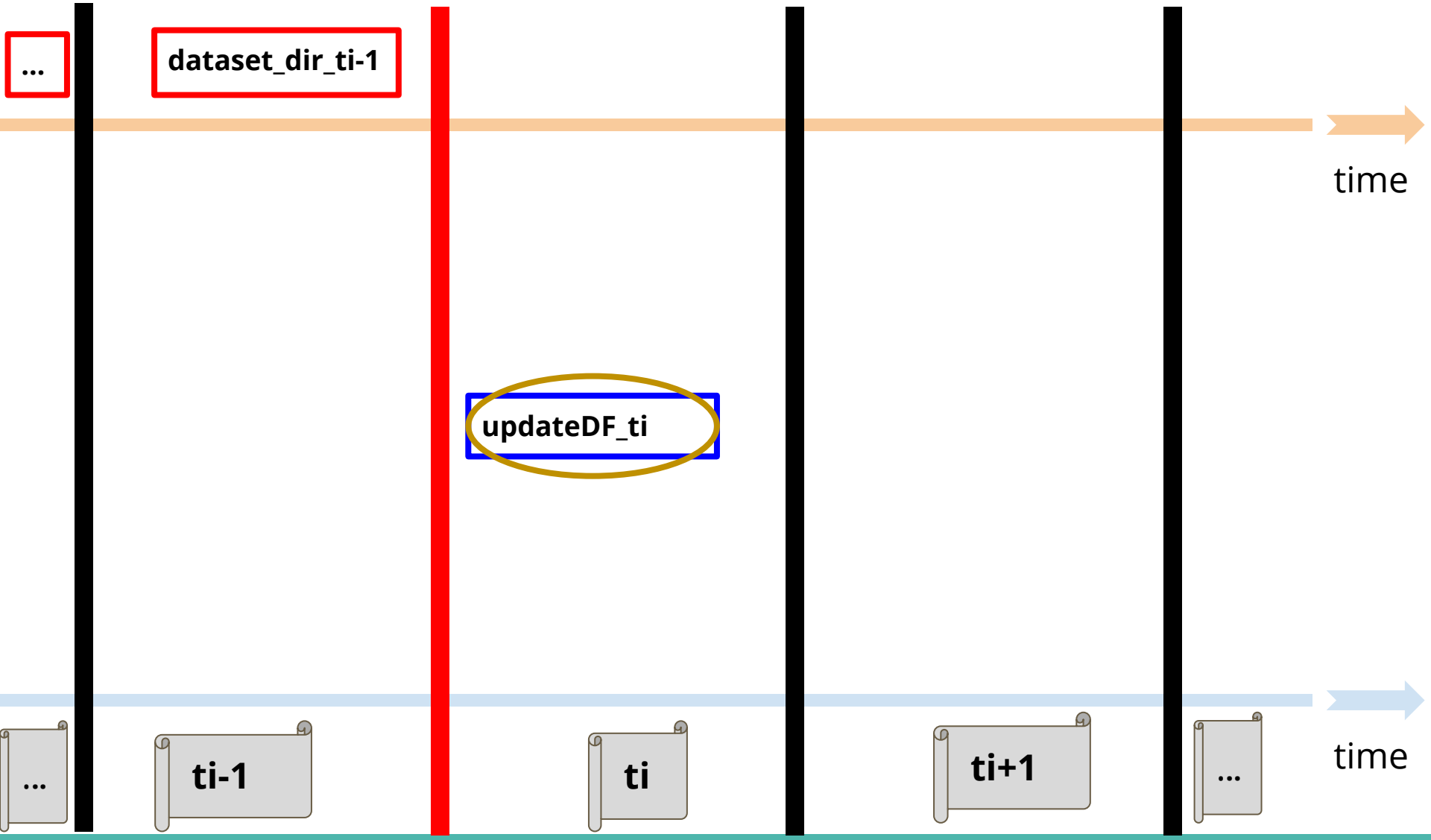
Concept4: Complete Mode



Concept4: Complete Mode



Concept4: Complete Mode



Concept4: Complete Mode

And with the unbound table representation of the SDF...

inputSDF

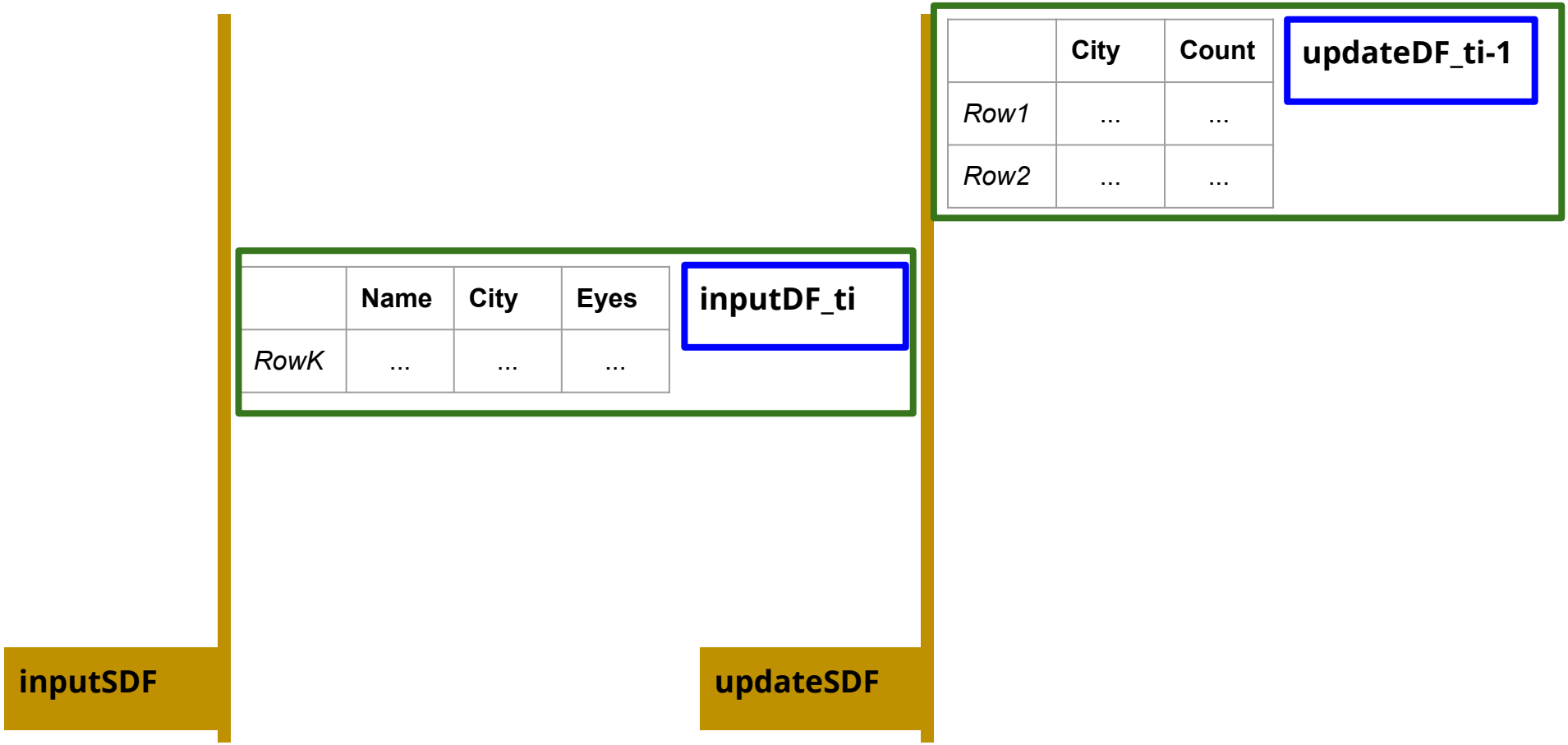
updateSDF

	City	Count
Row1	...	...
Row2	...	...

updateDF_ti-1

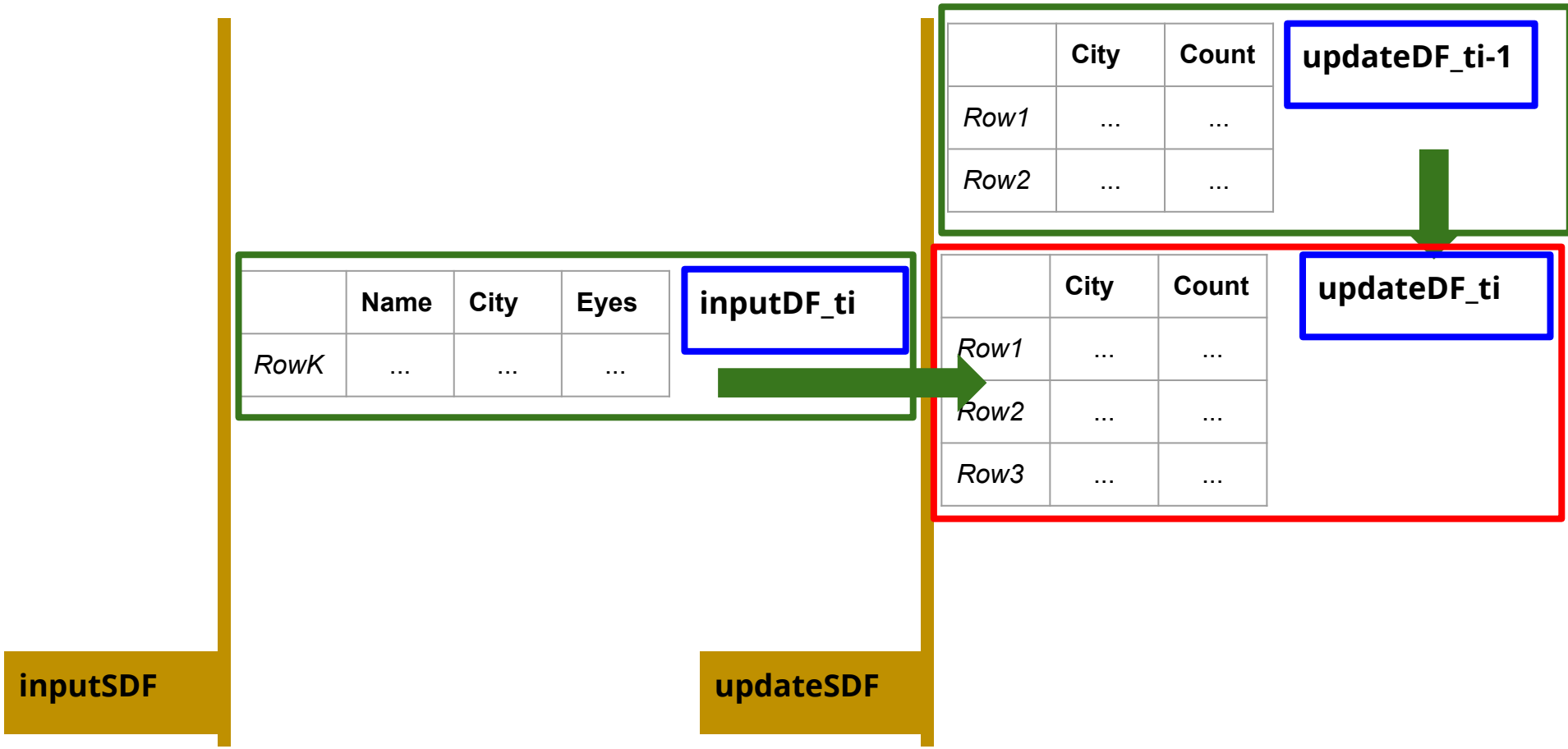
Concept4: Complete Mode

And with the unbound table representation of the SDF...



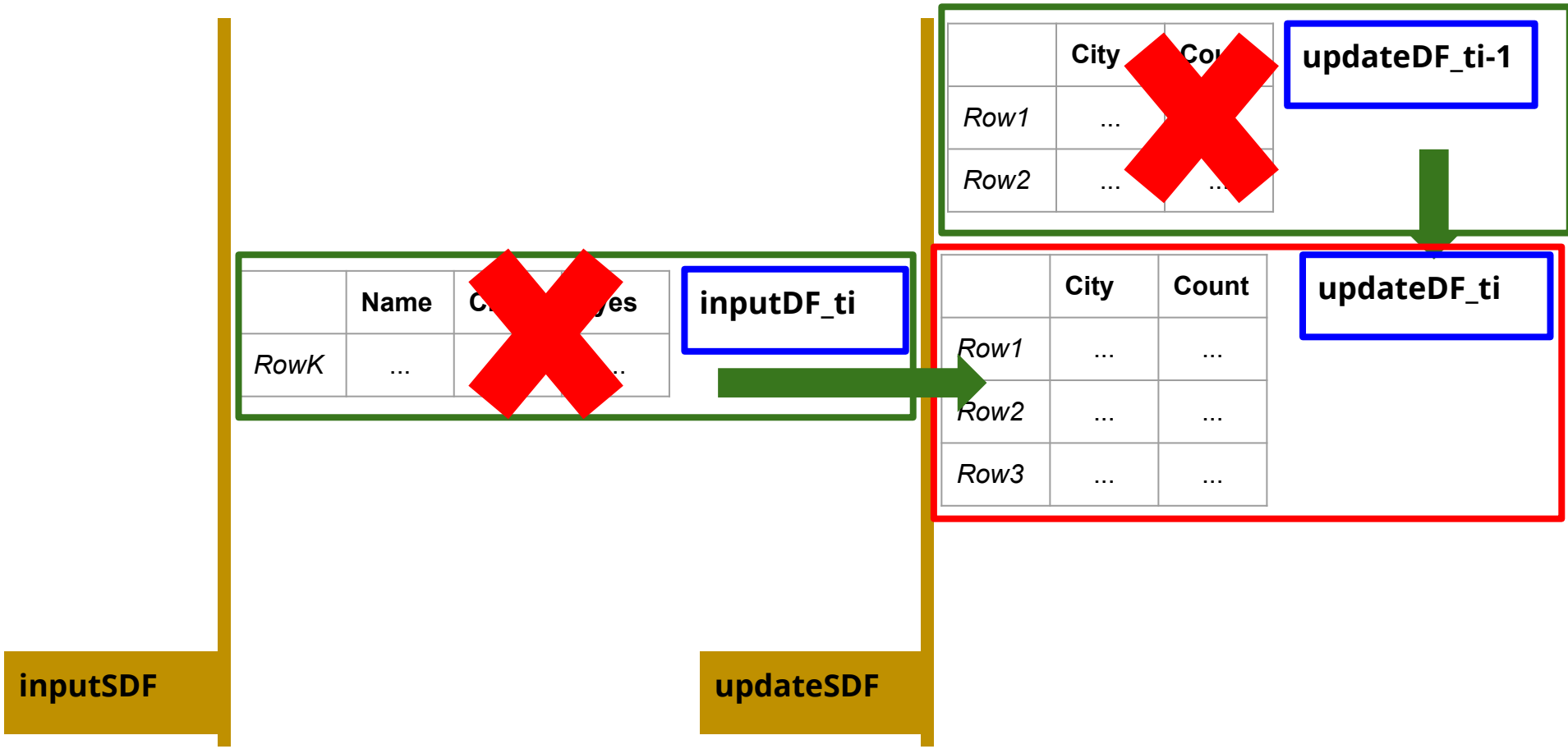
Concept4: Complete Mode

And with the unbound table representation of the SDF...



Concept4: Complete Mode

And with the unbound table representation of the SDF...



Concept4: Complete Mode

And with the unbound table representation of the SDF...

inputSDF

updateSDF

	City	Count
Row1	...	...
Row2	...	...
Row3	...	...

updateDF_ti

Concept4: Complete Mode

Concept 4:

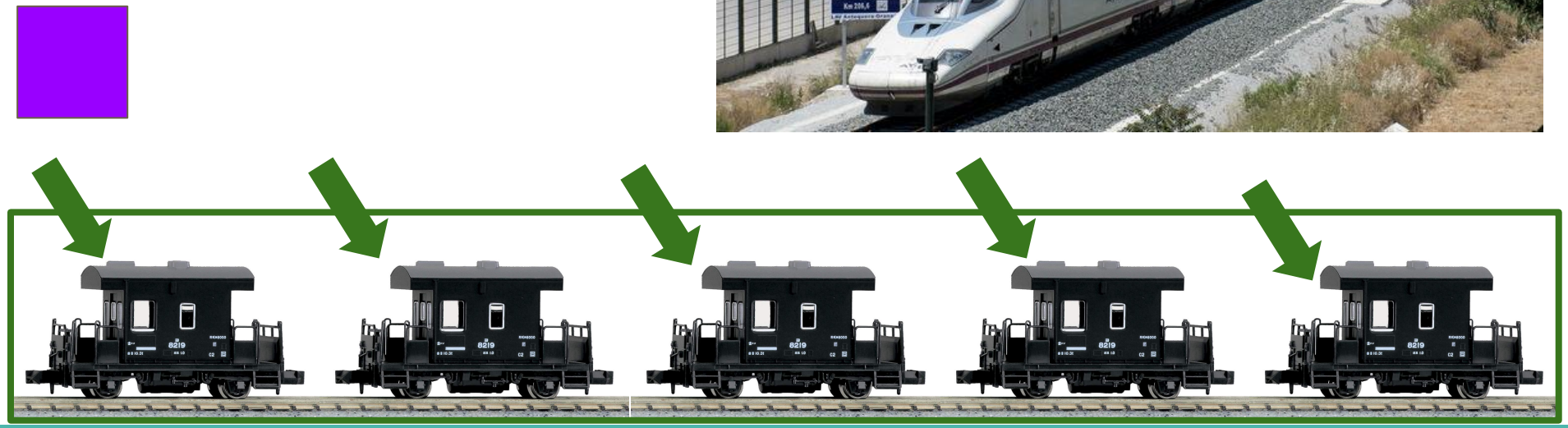
This behaviour is called
a **SDF** in **Complete Mode**!

Outline

1. Setting Up the Context.
2. From DStream to SDF.
   a. Concept1: SDF.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
3. Practising with the Concepts.

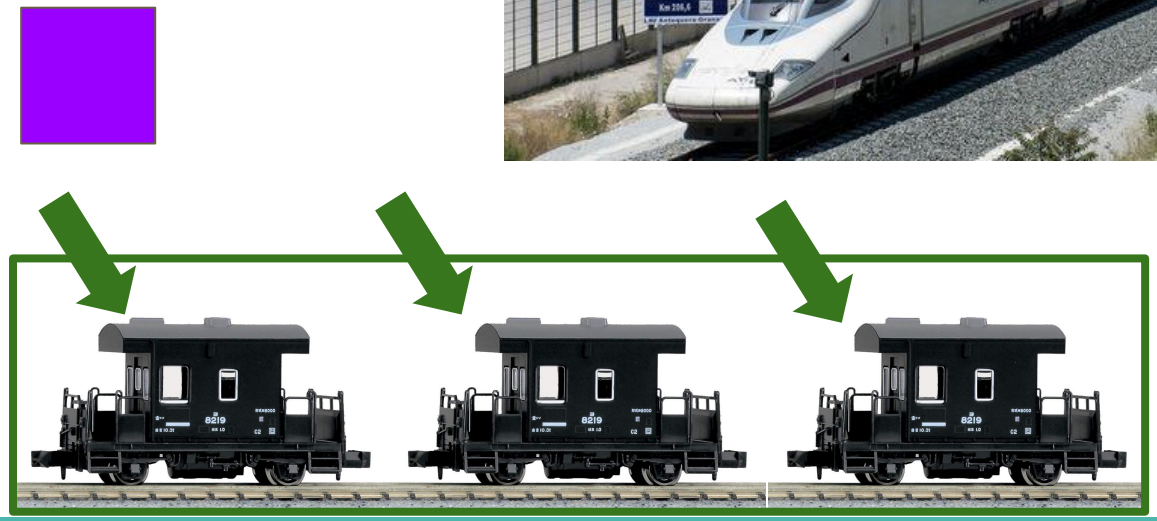
Concept5: Complete Mode with Windows

A **DStream** uses update-based operations combined with window-based operations to group **wagons** together before aggregating their results.



Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**



Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**

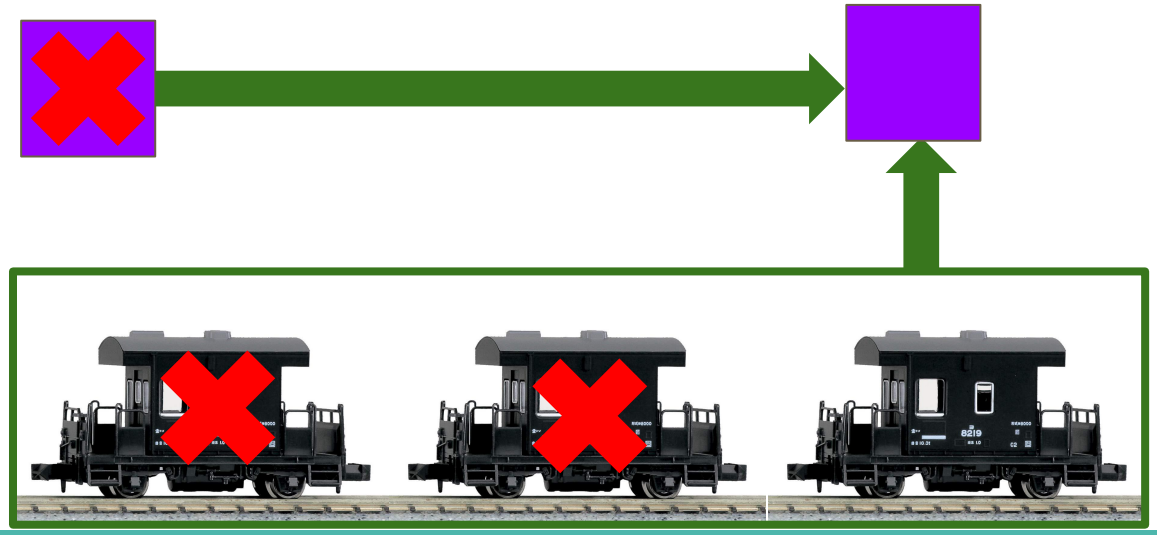
SELECTED WINDOW

Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**

Concept5: Complete Mode with Windows

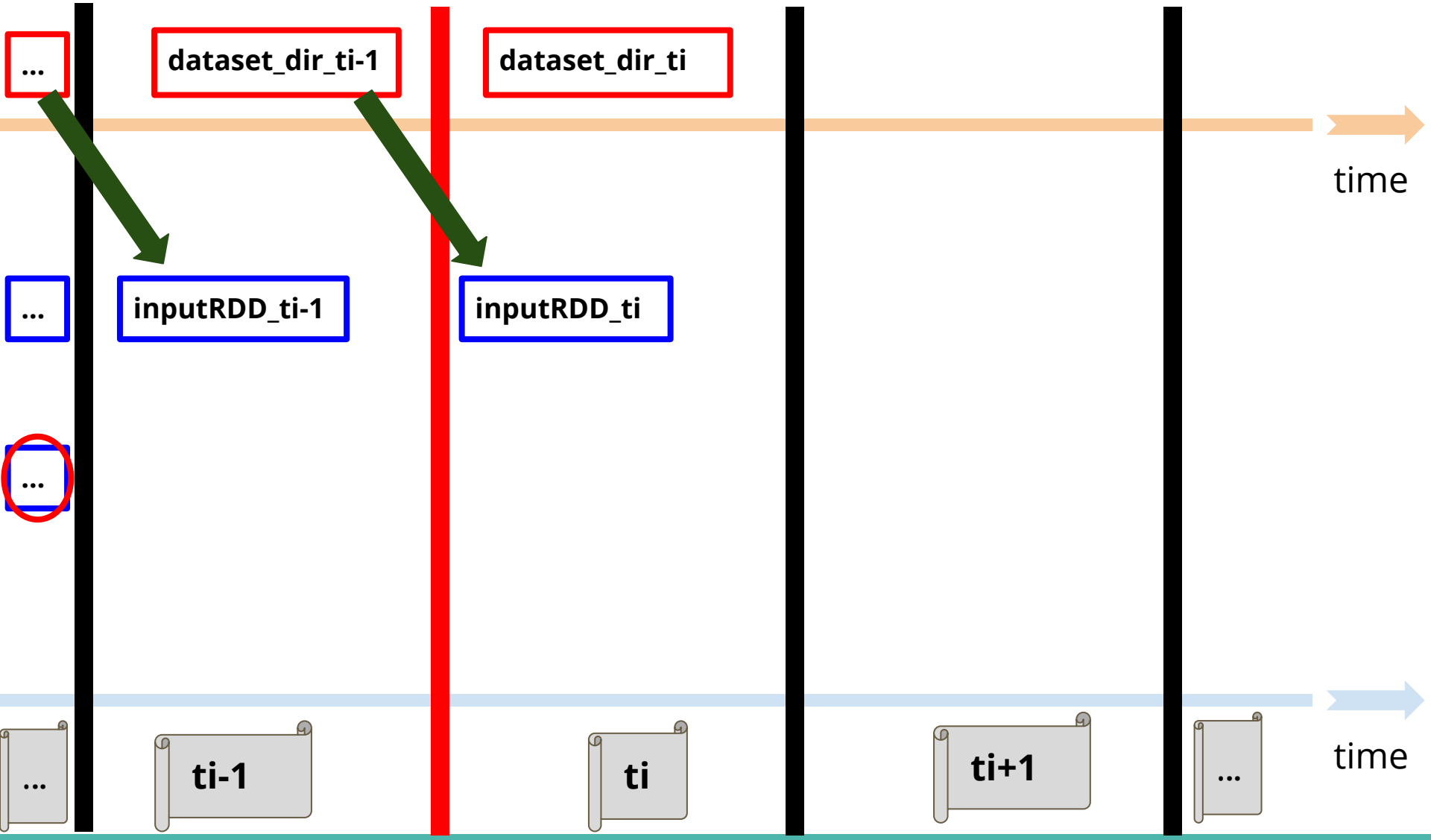
Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**



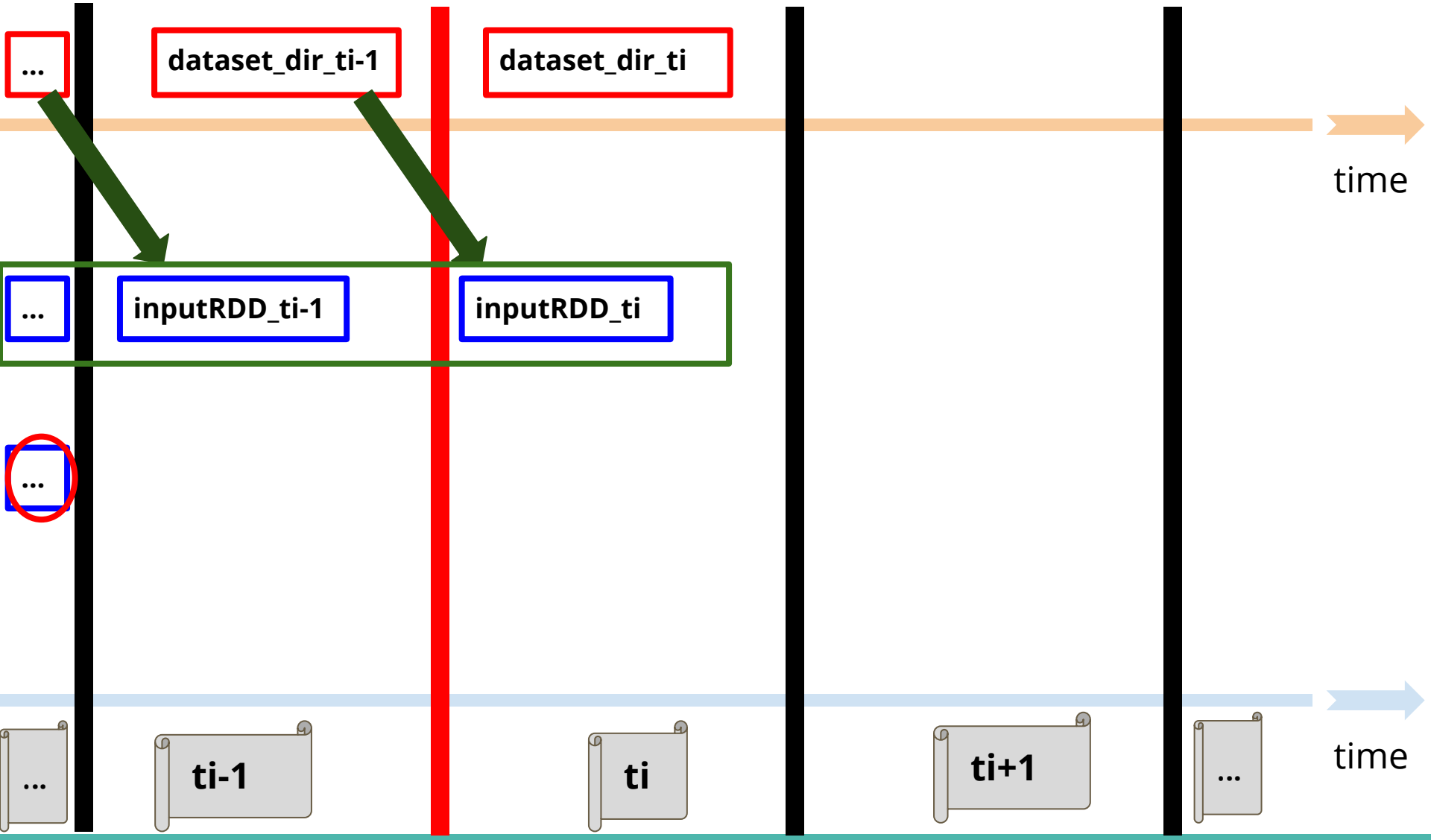
Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**

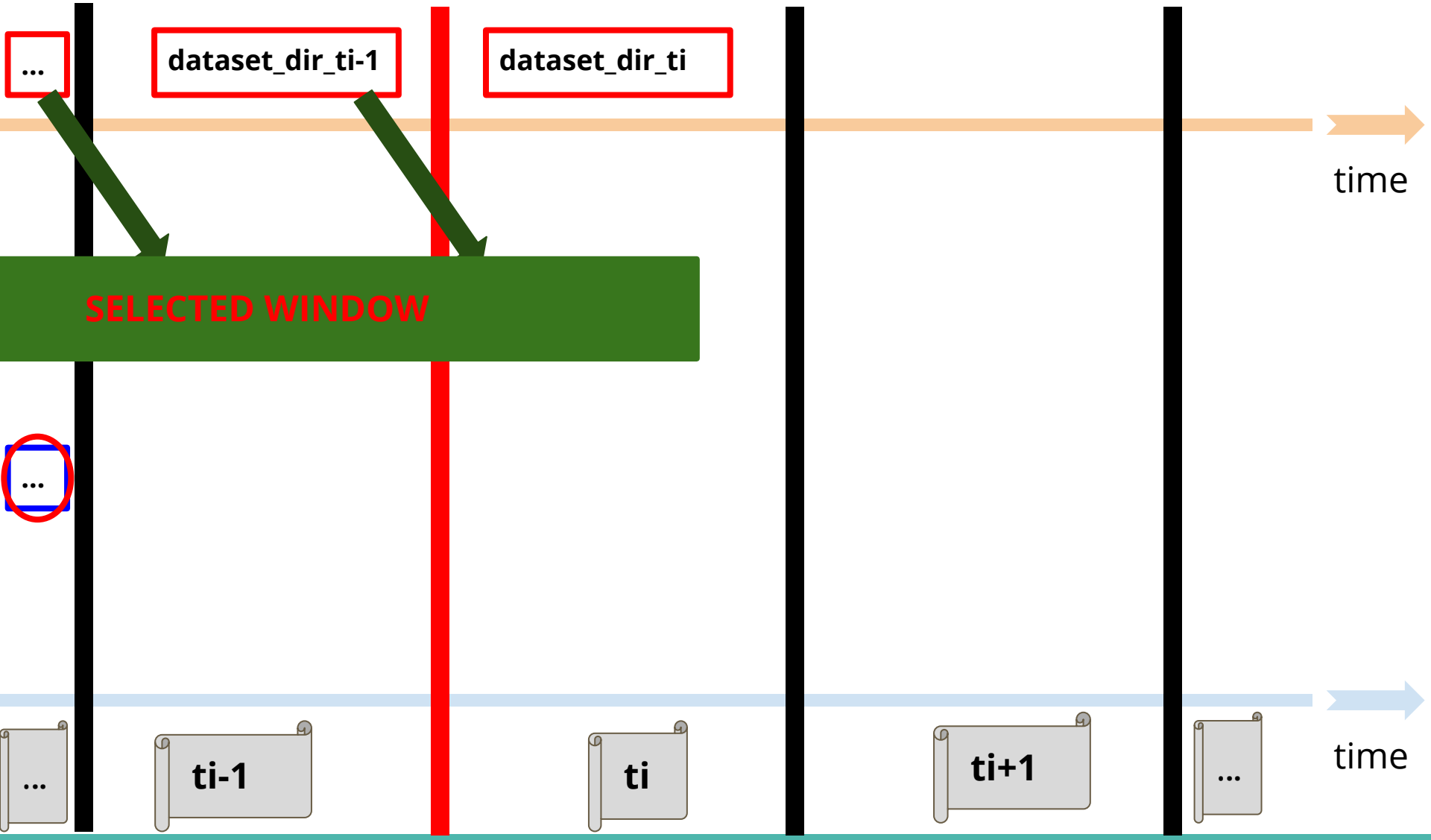
Concept5: Complete Mode with Windows



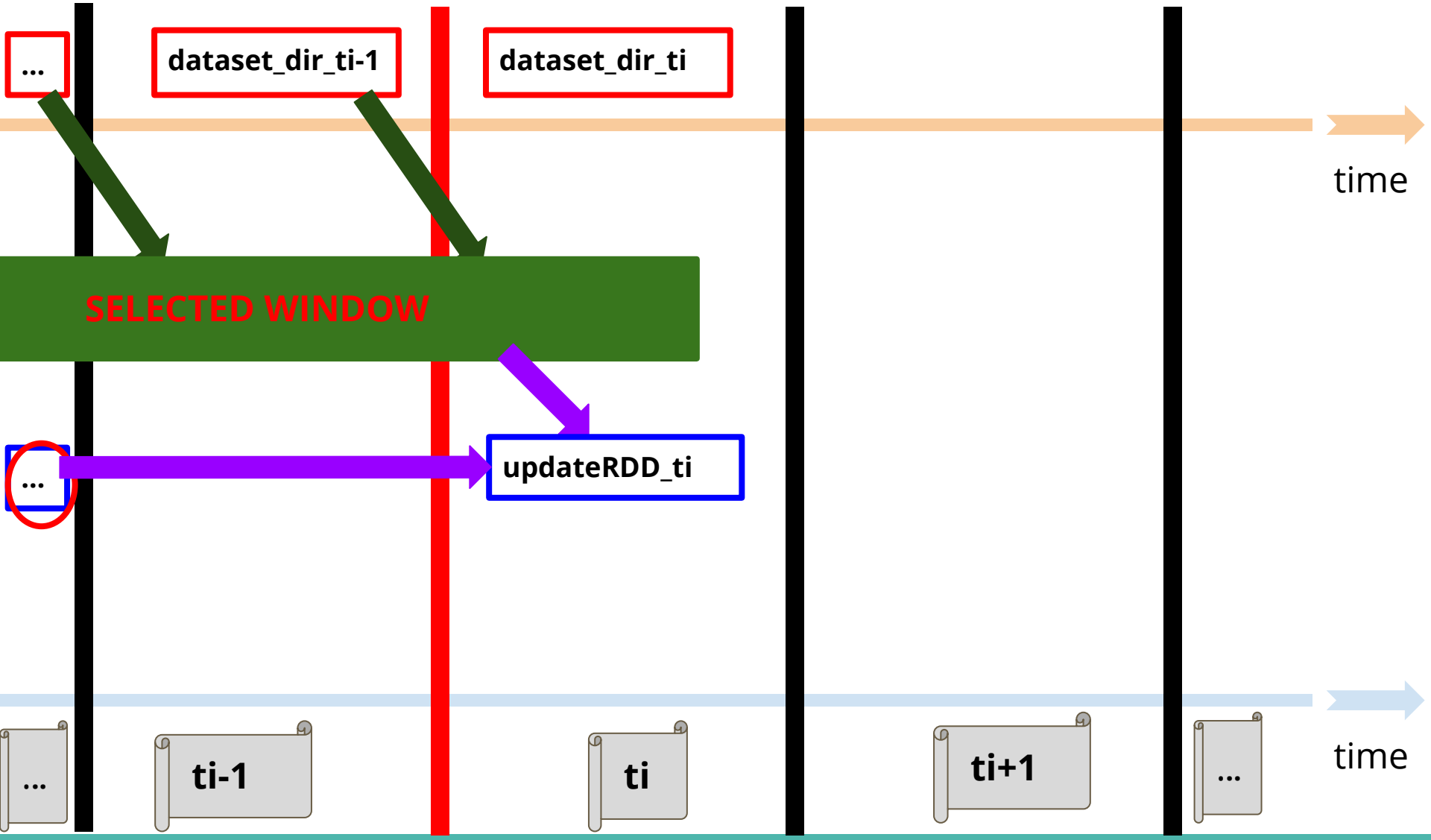
Concept5: Complete Mode with Windows



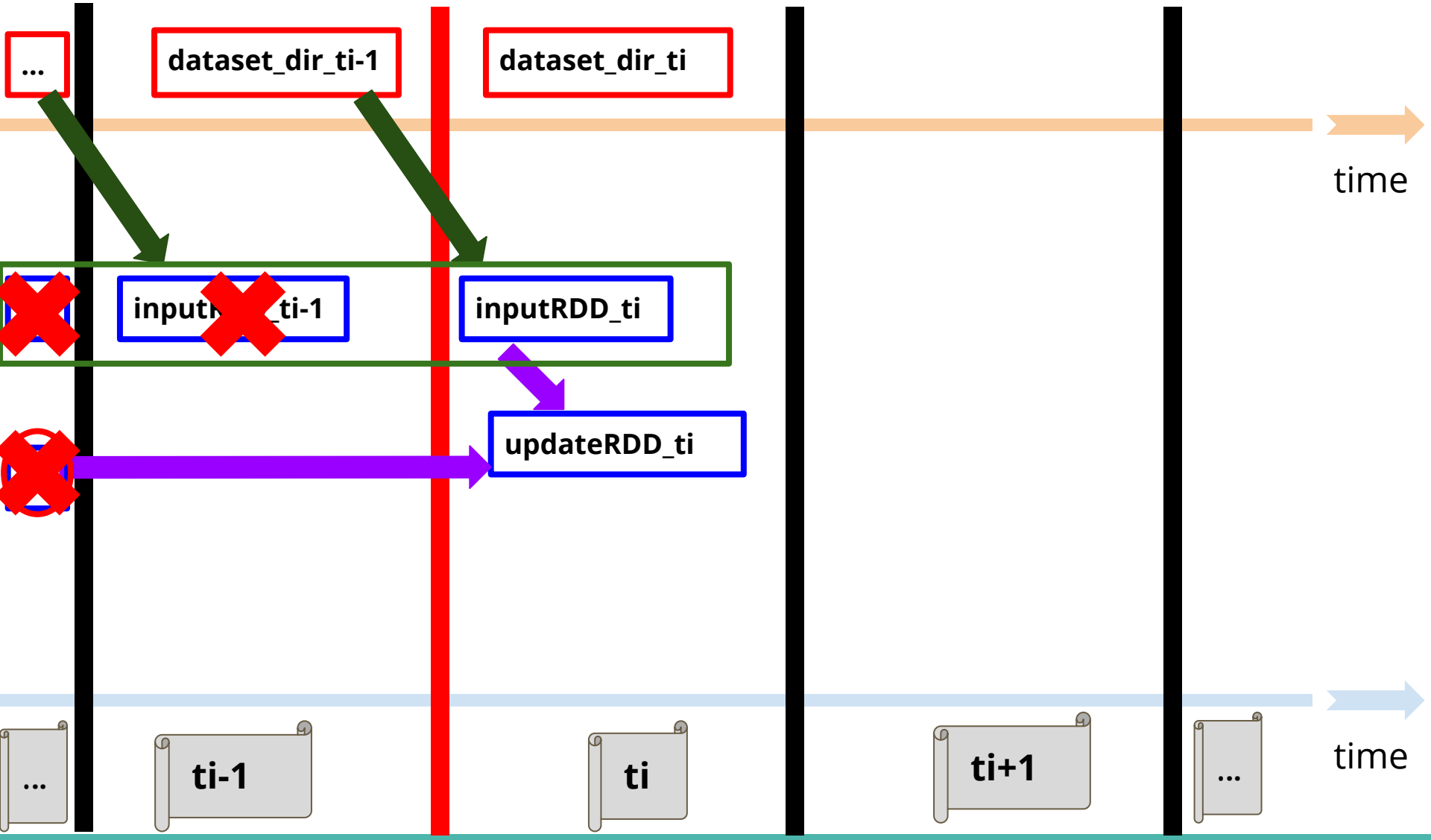
Concept5: Complete Mode with Windows



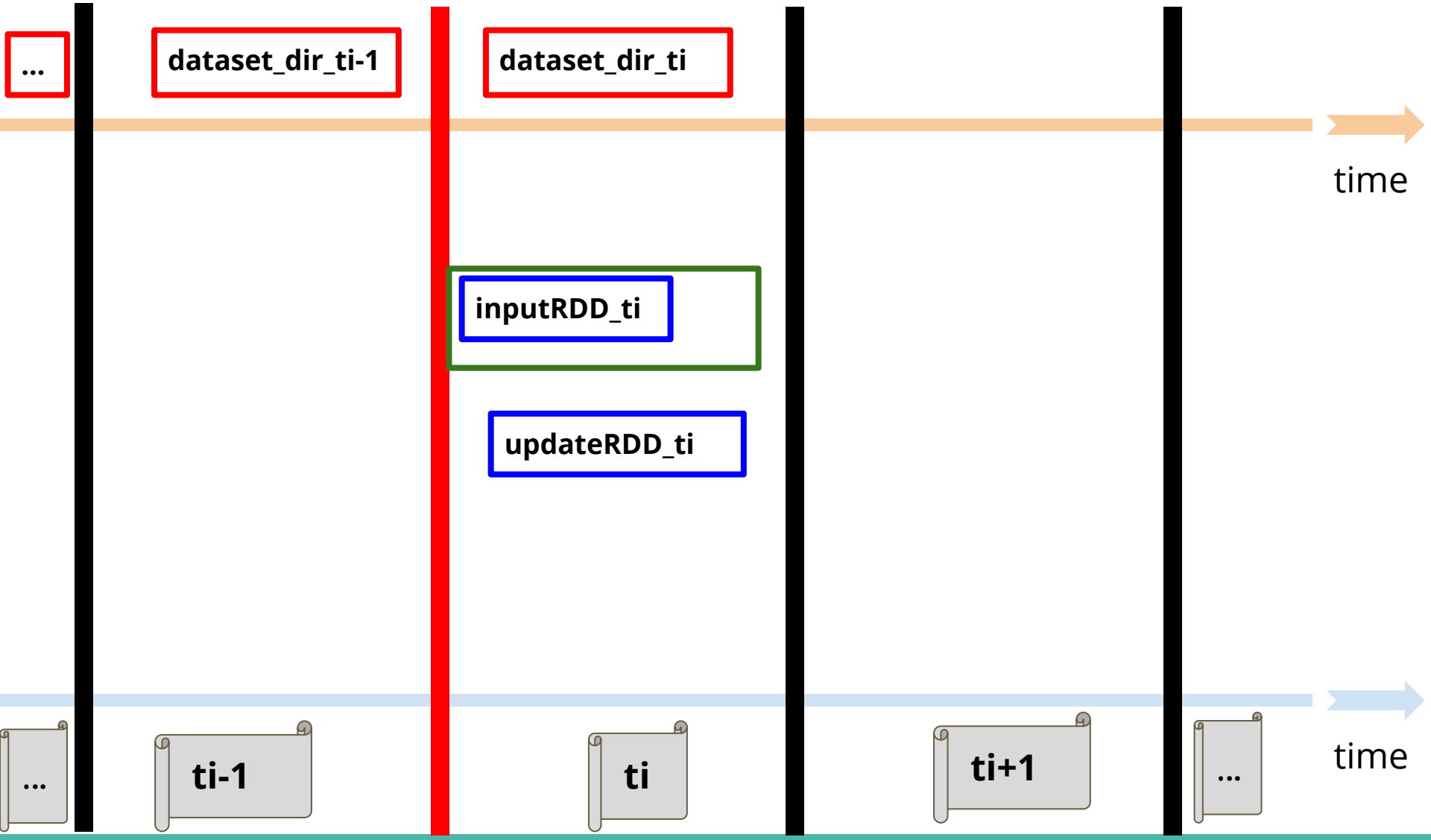
Concept5: Complete Mode with Windows



Concept5: Complete Mode with Windows



Concept5: Complete Mode with Windows



Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**

Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**

Concept5: Complete Mode with Windows

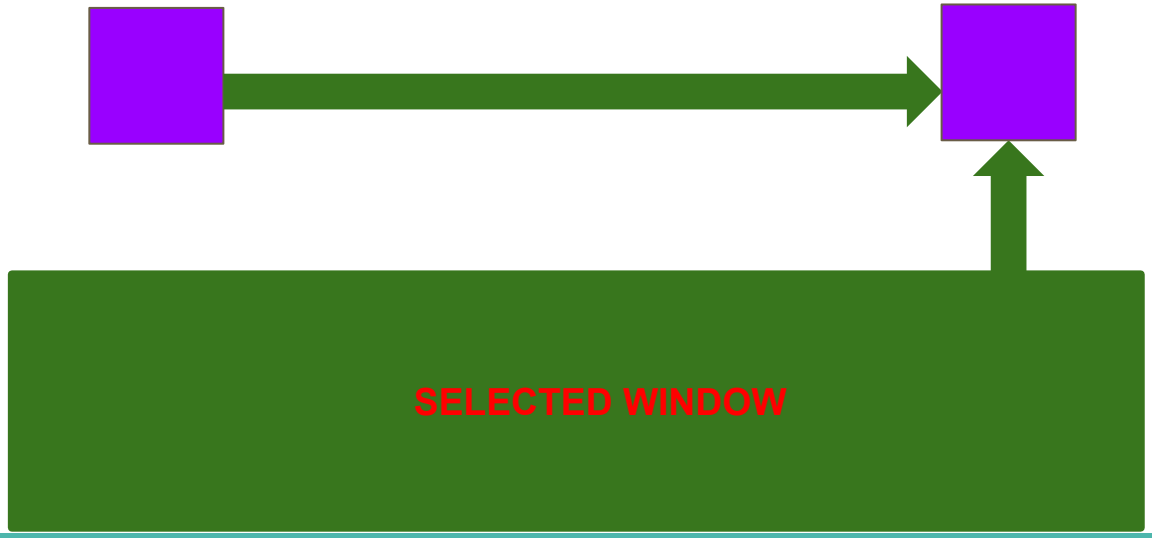
Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**

Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**

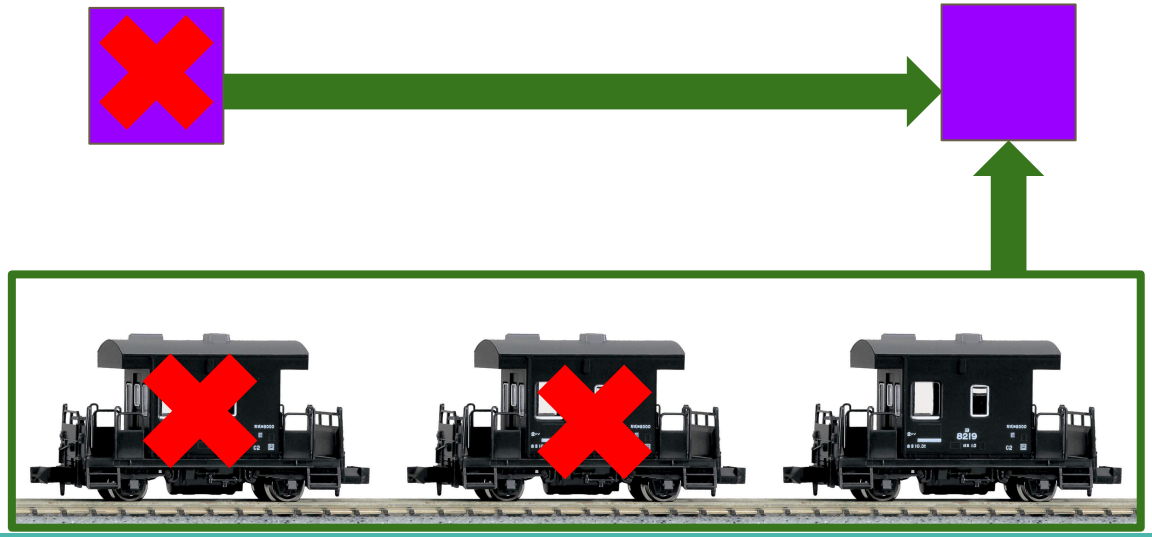
Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**



Concept5: Complete Mode with Windows

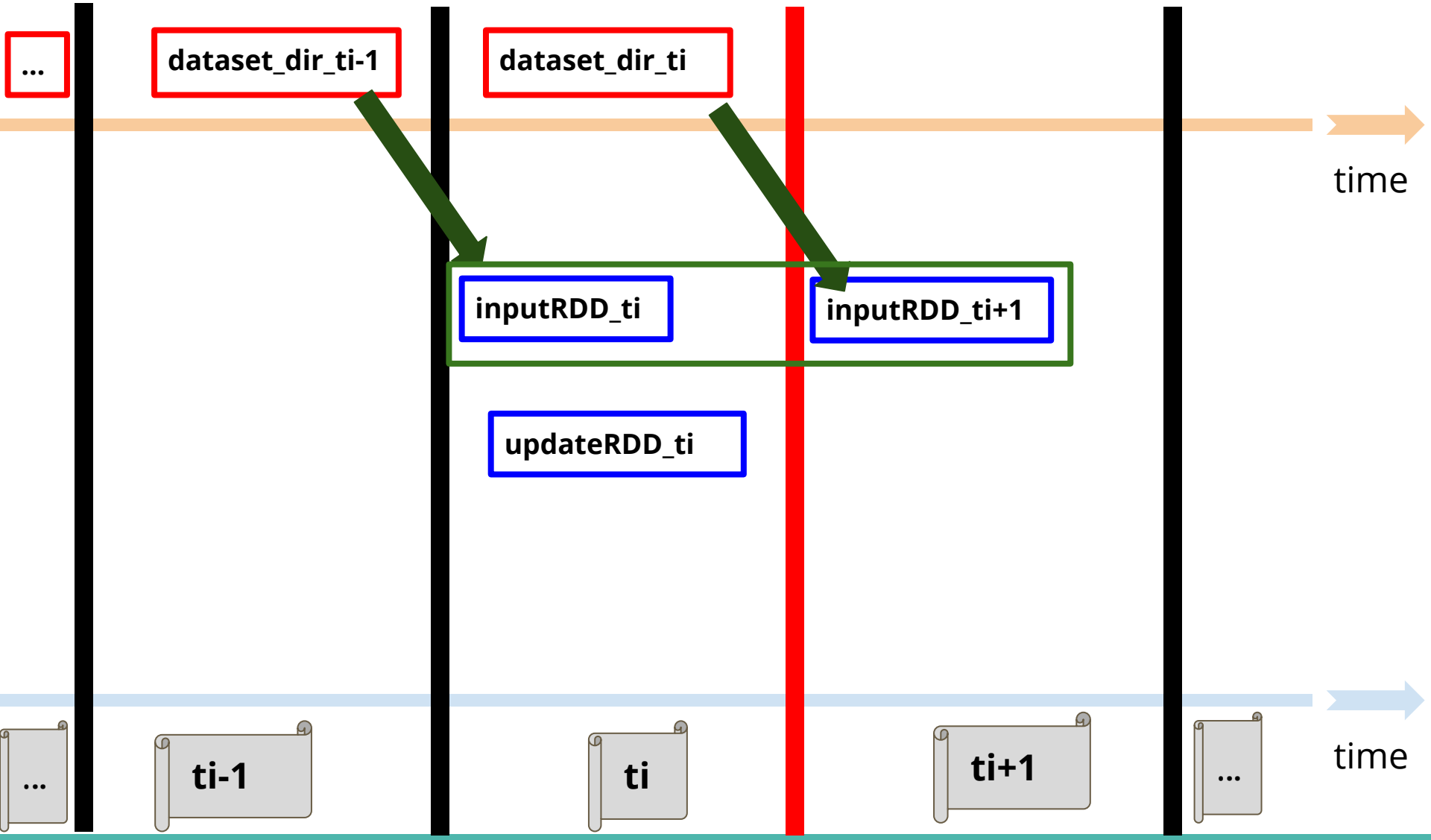
Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**



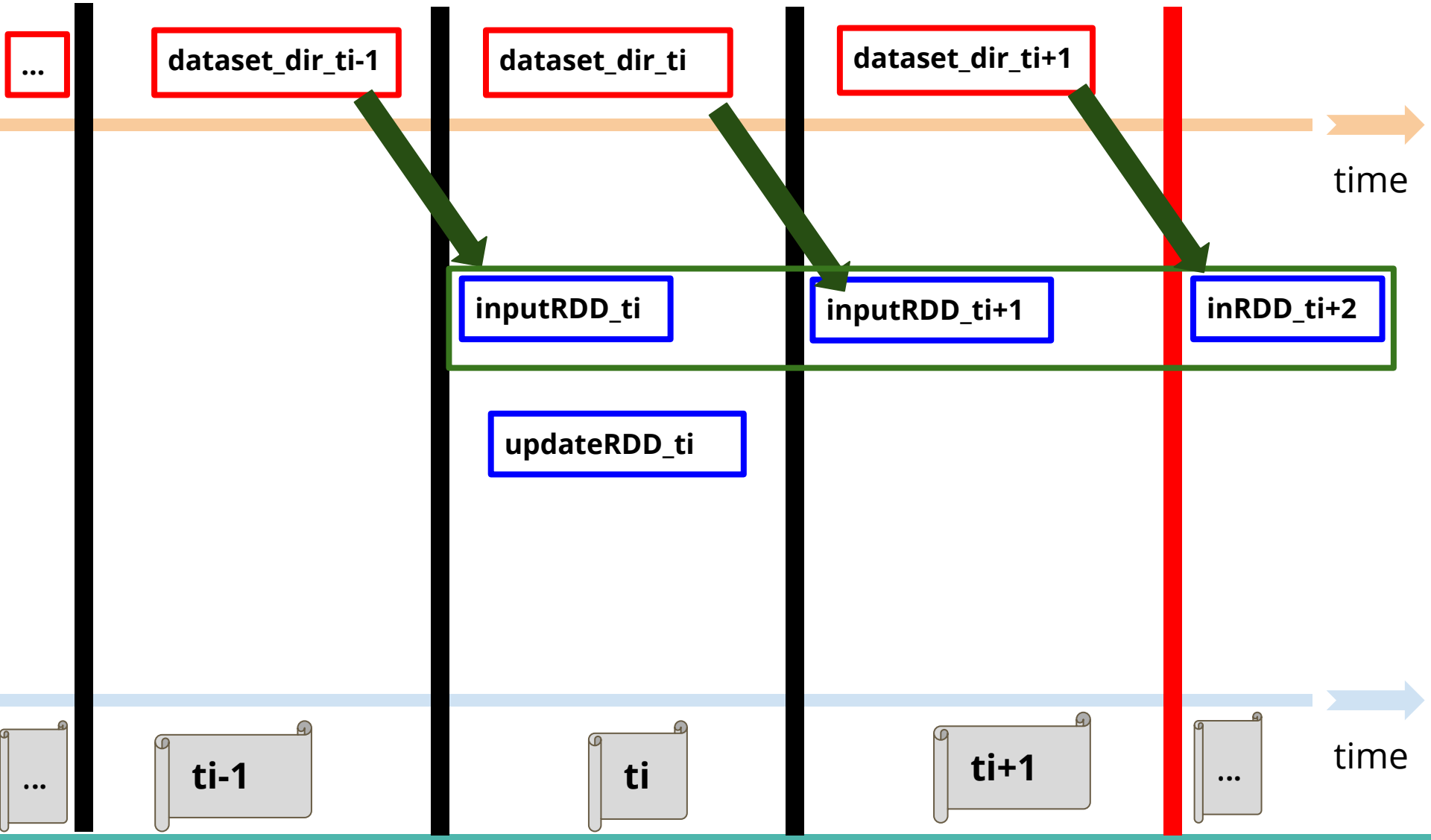
Concept5: Complete Mode with Windows

Let's come back to our classical example.
Sliding Duration = 2 and **Window Duration = 3**

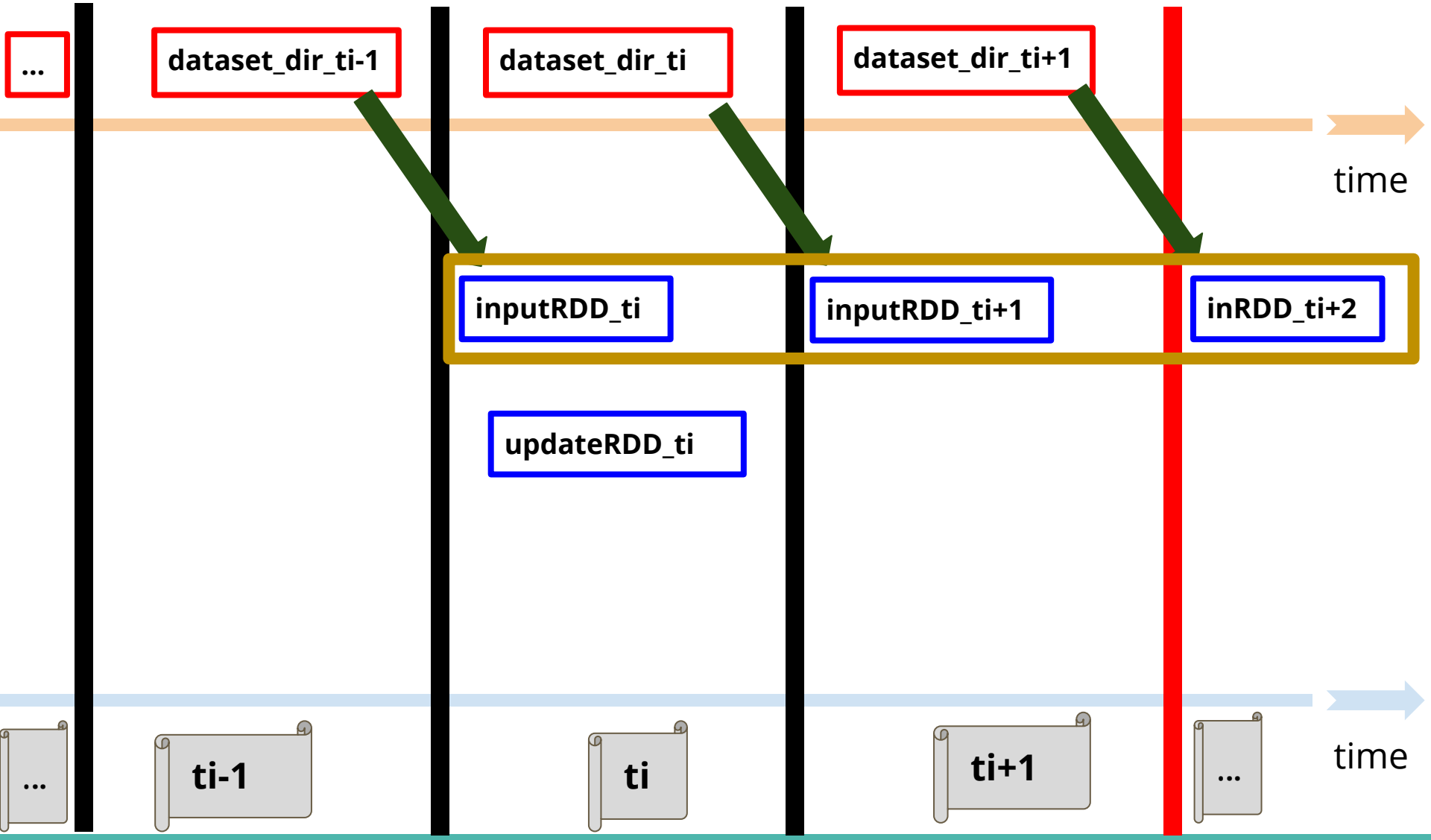
Concept5: Complete Mode with Windows



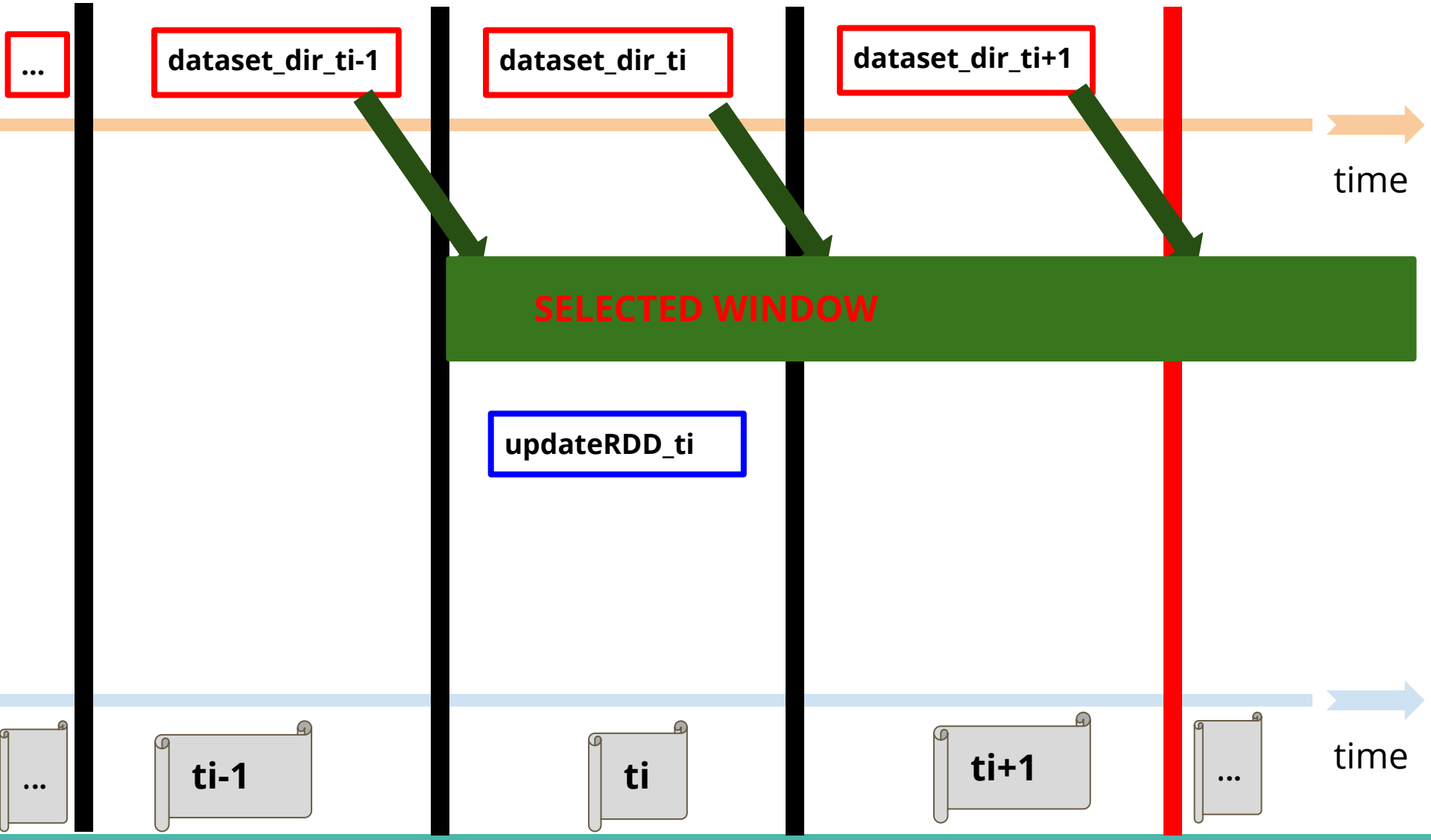
Concept5: Complete Mode with Windows



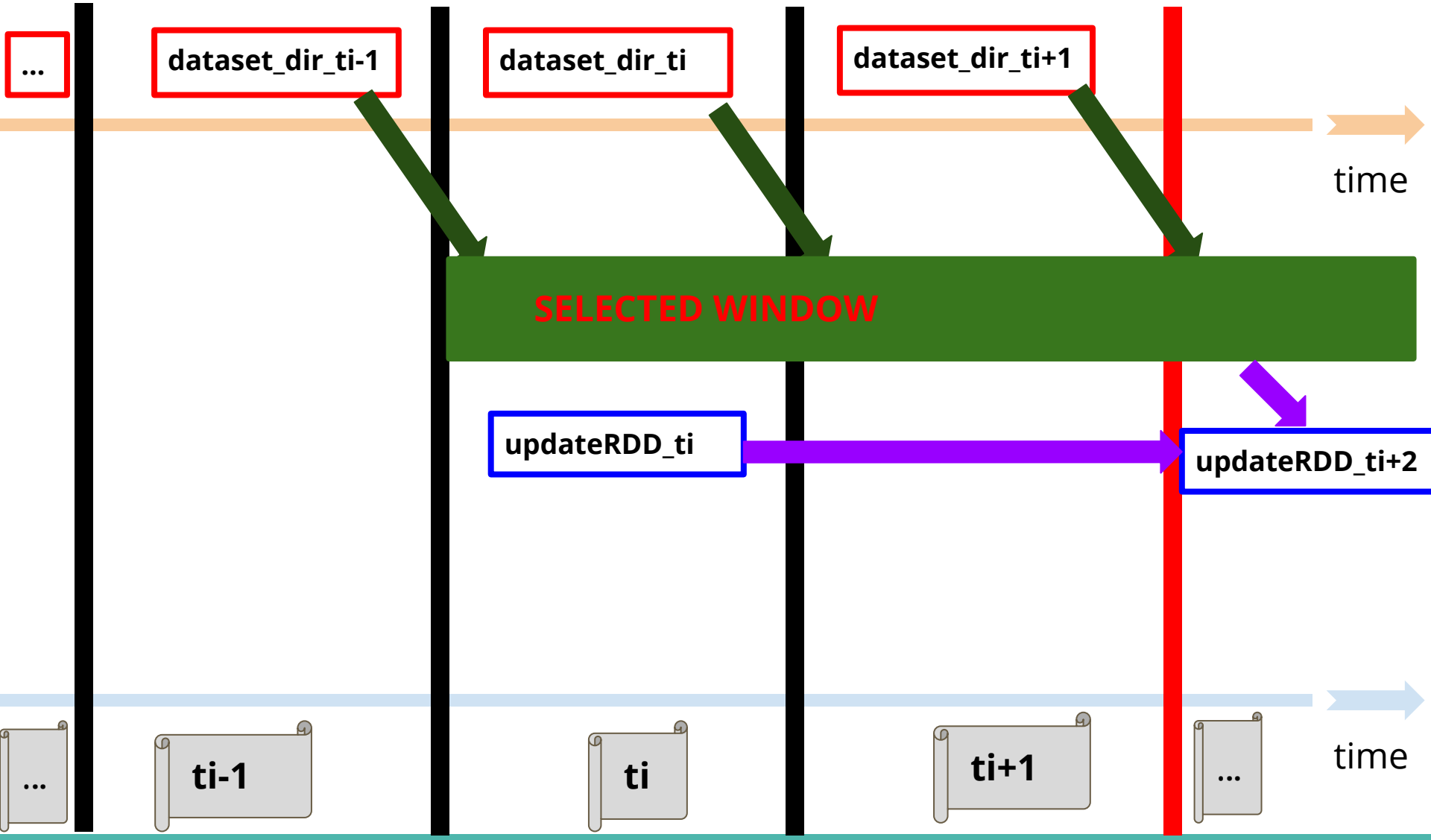
Concept5: Complete Mode with Windows



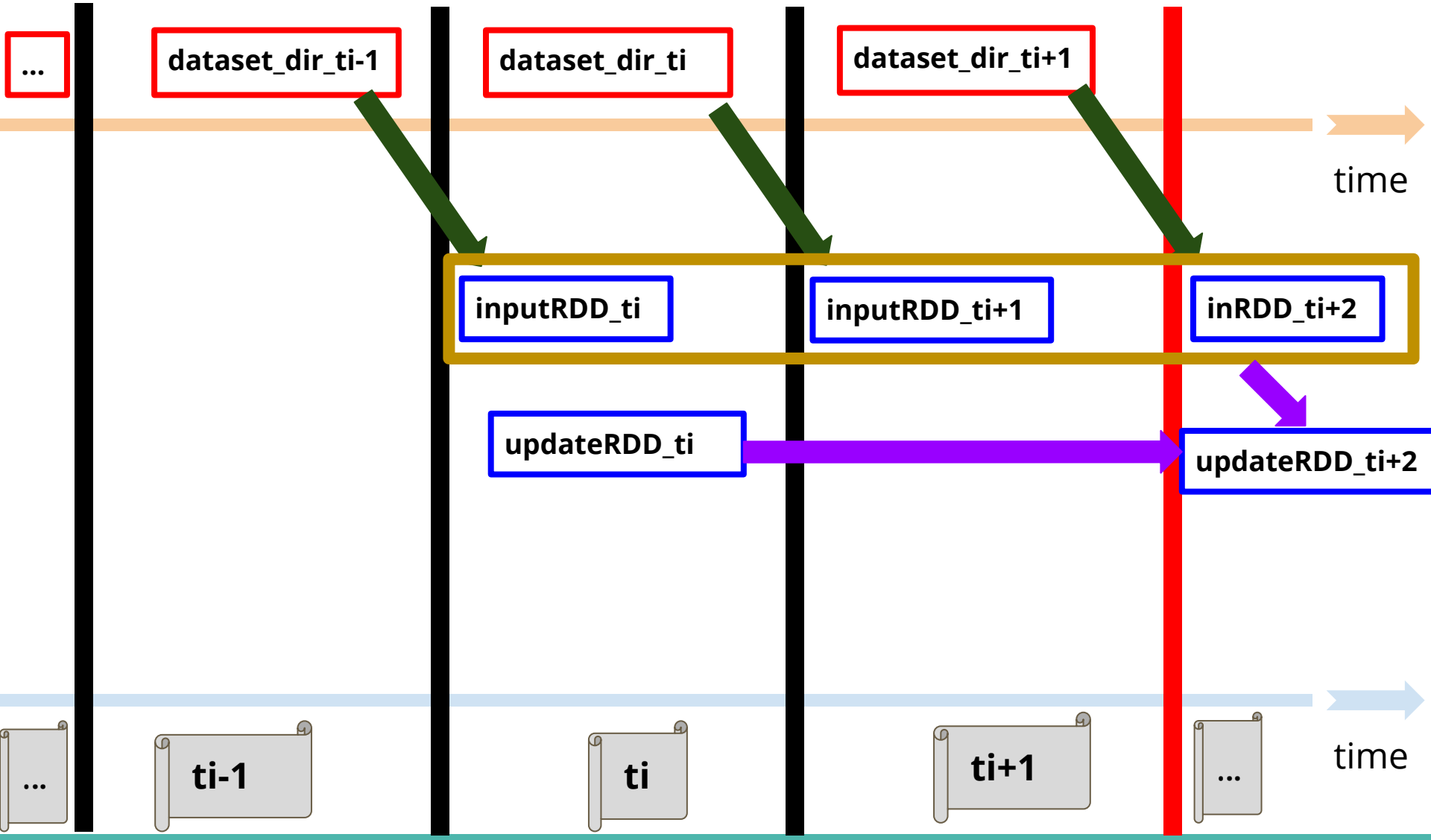
Concept5: Complete Mode with Windows



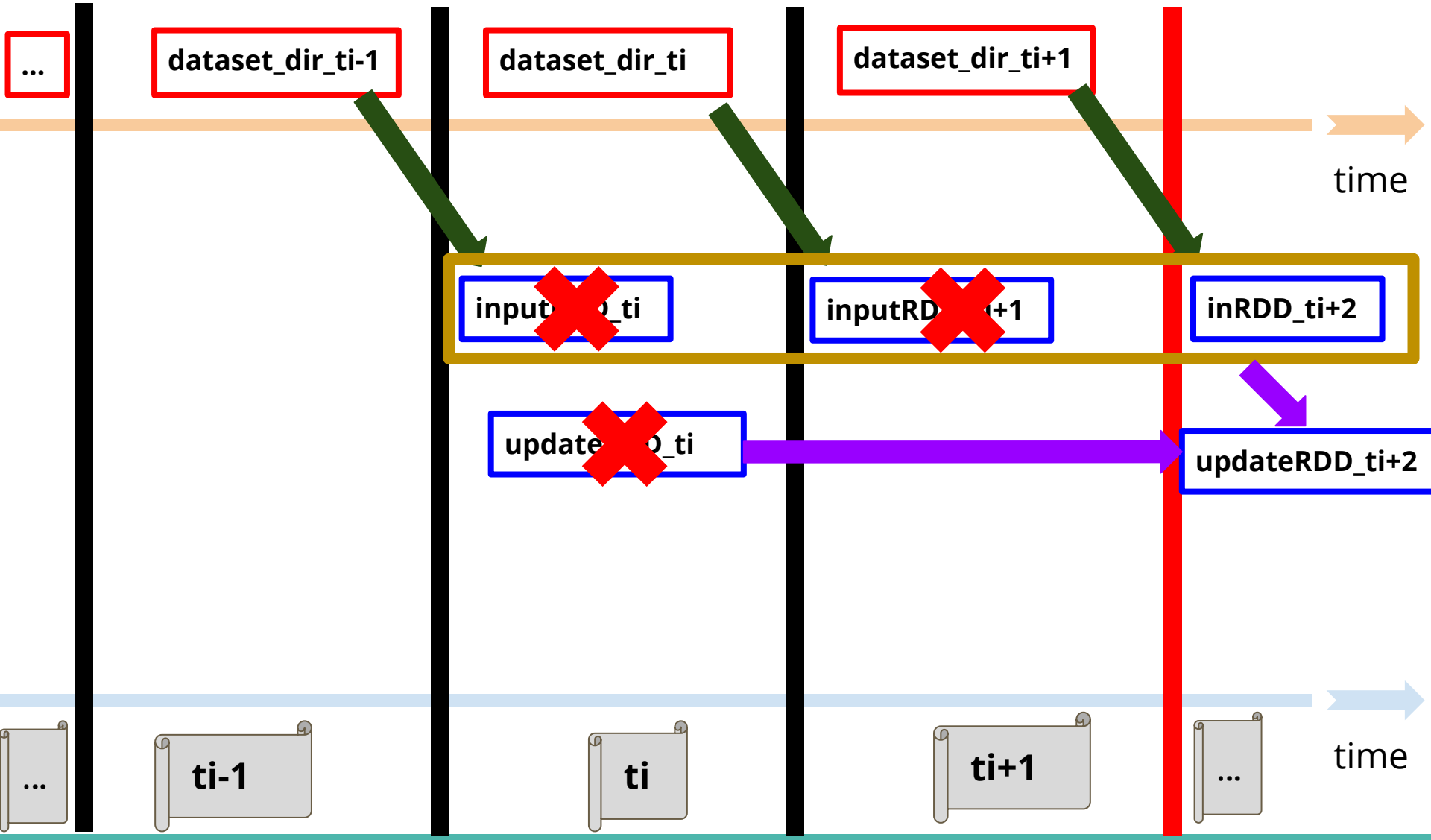
Concept5: Complete Mode with Windows



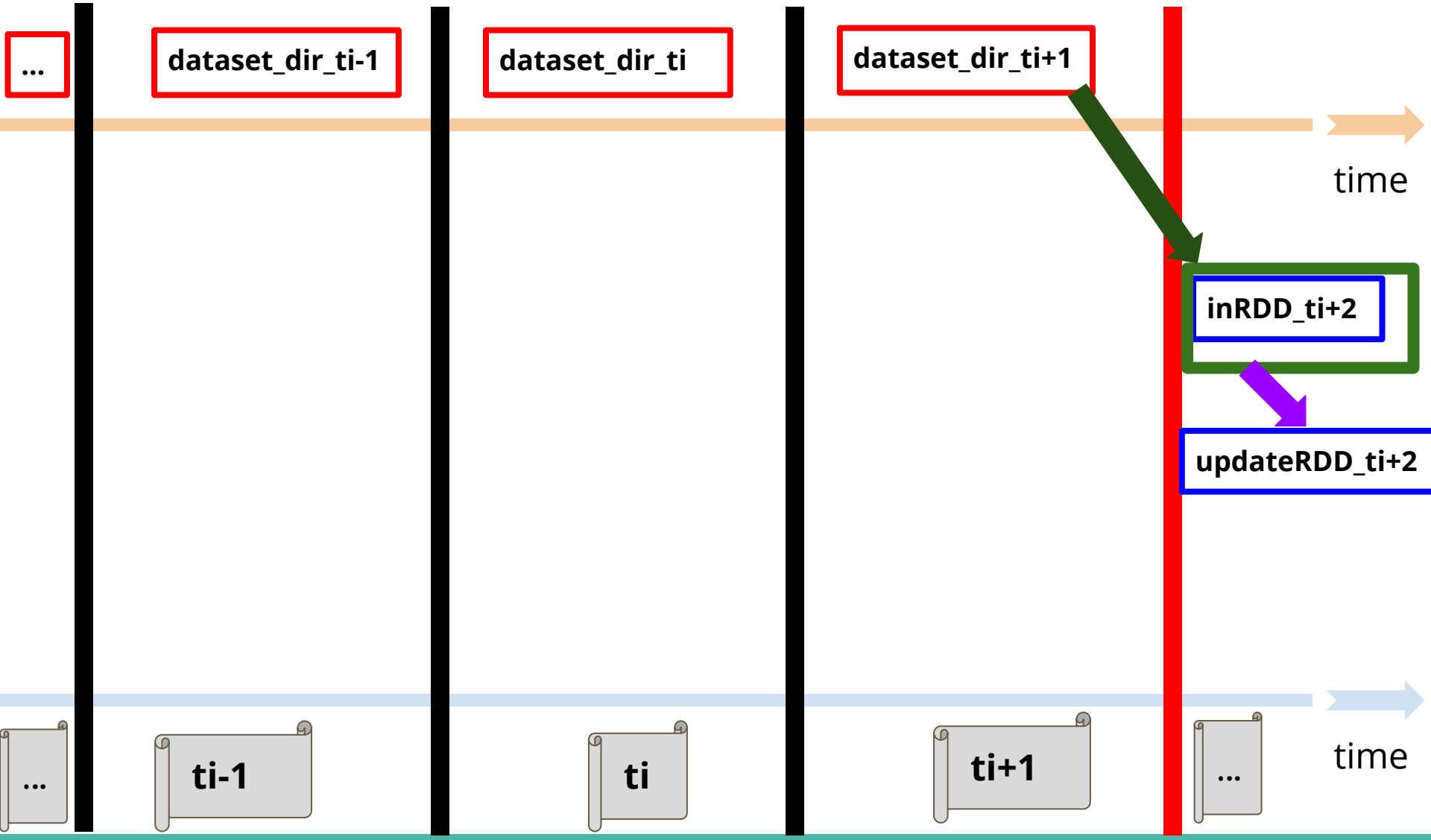
Concept5: Complete Mode with Windows



Concept5: Complete Mode with Windows



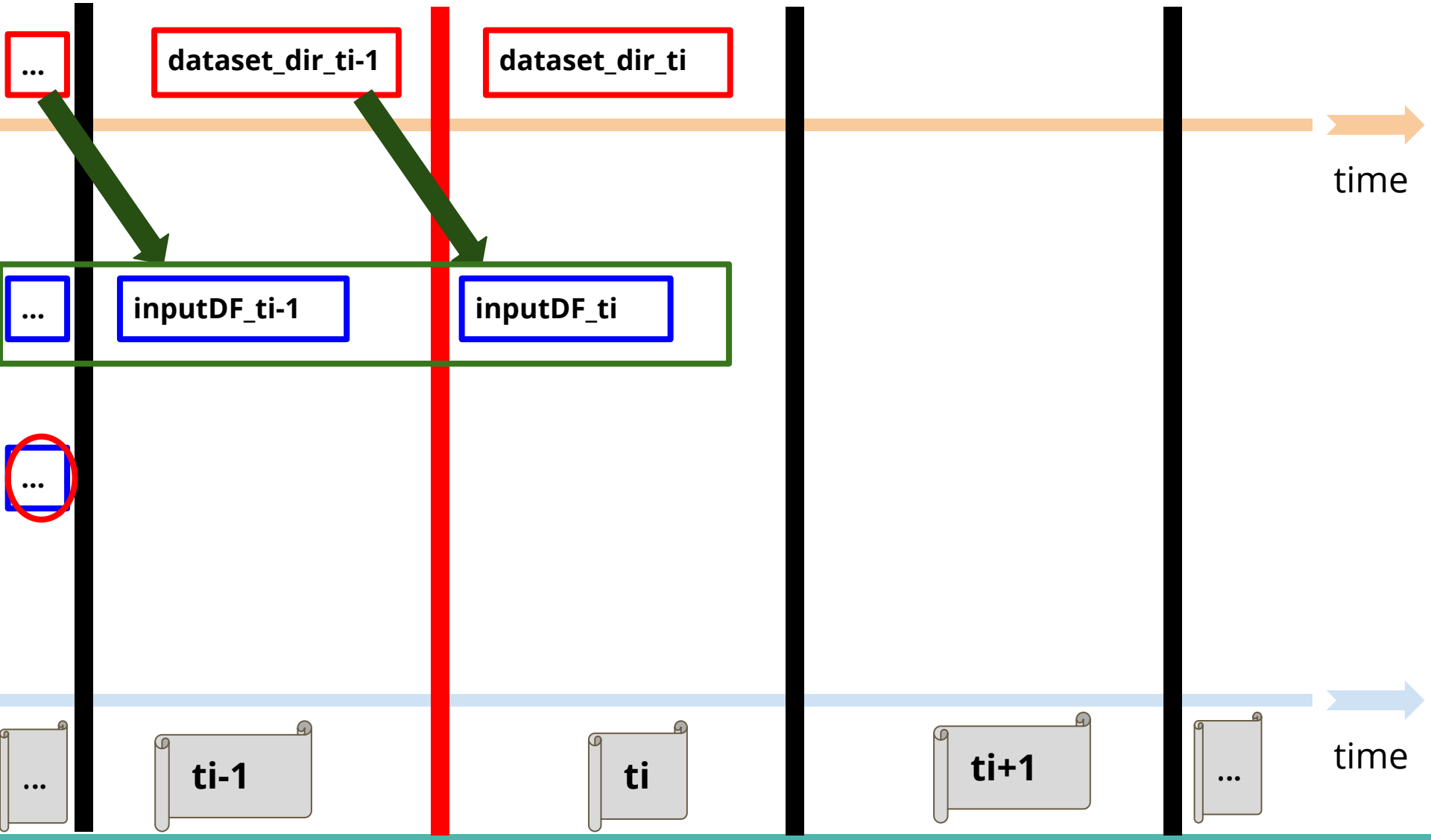
Concept5: Complete Mode with Windows



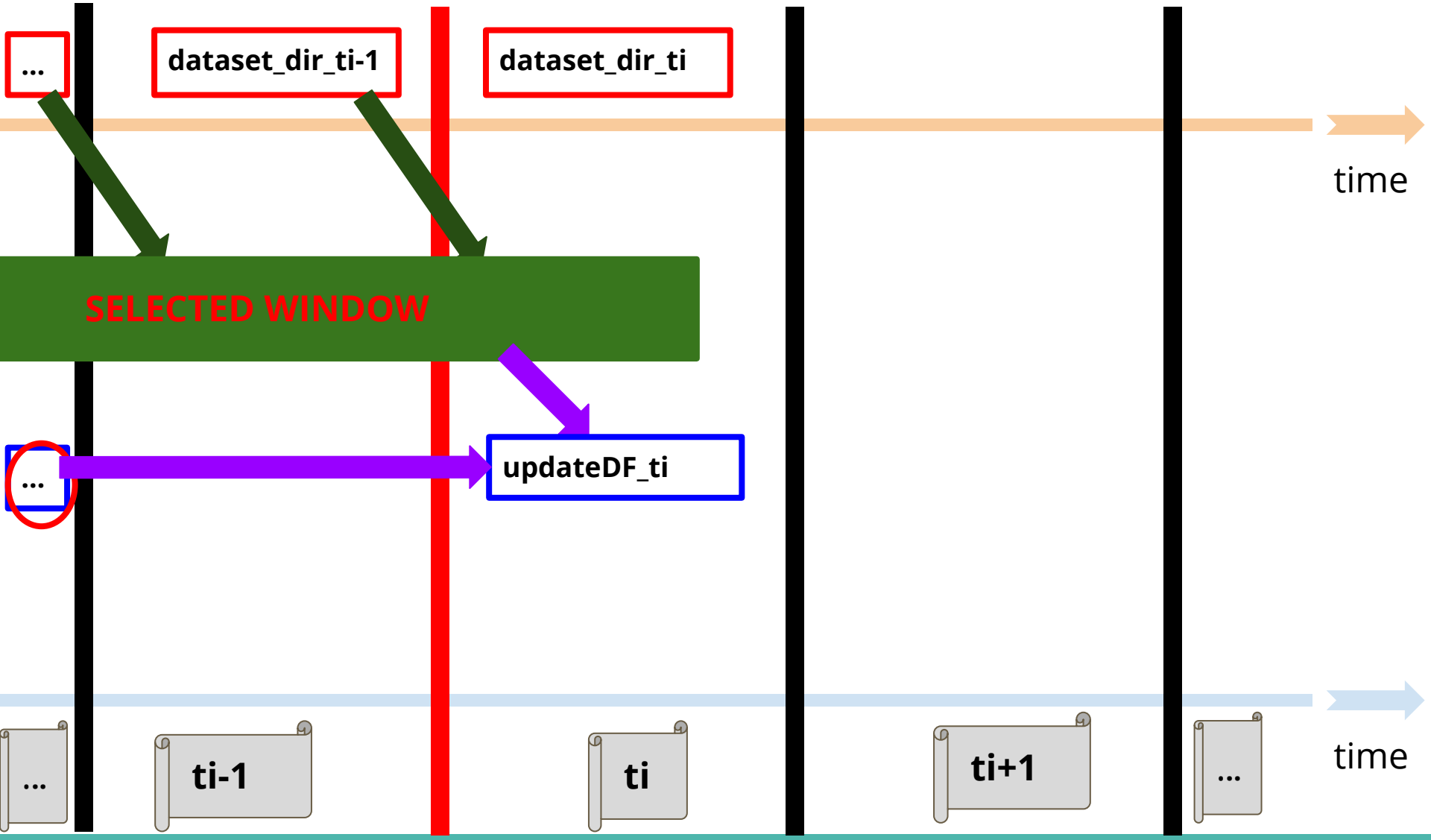
Concept5: Complete Mode with Windows

The same behaviour applies to **SDF**!

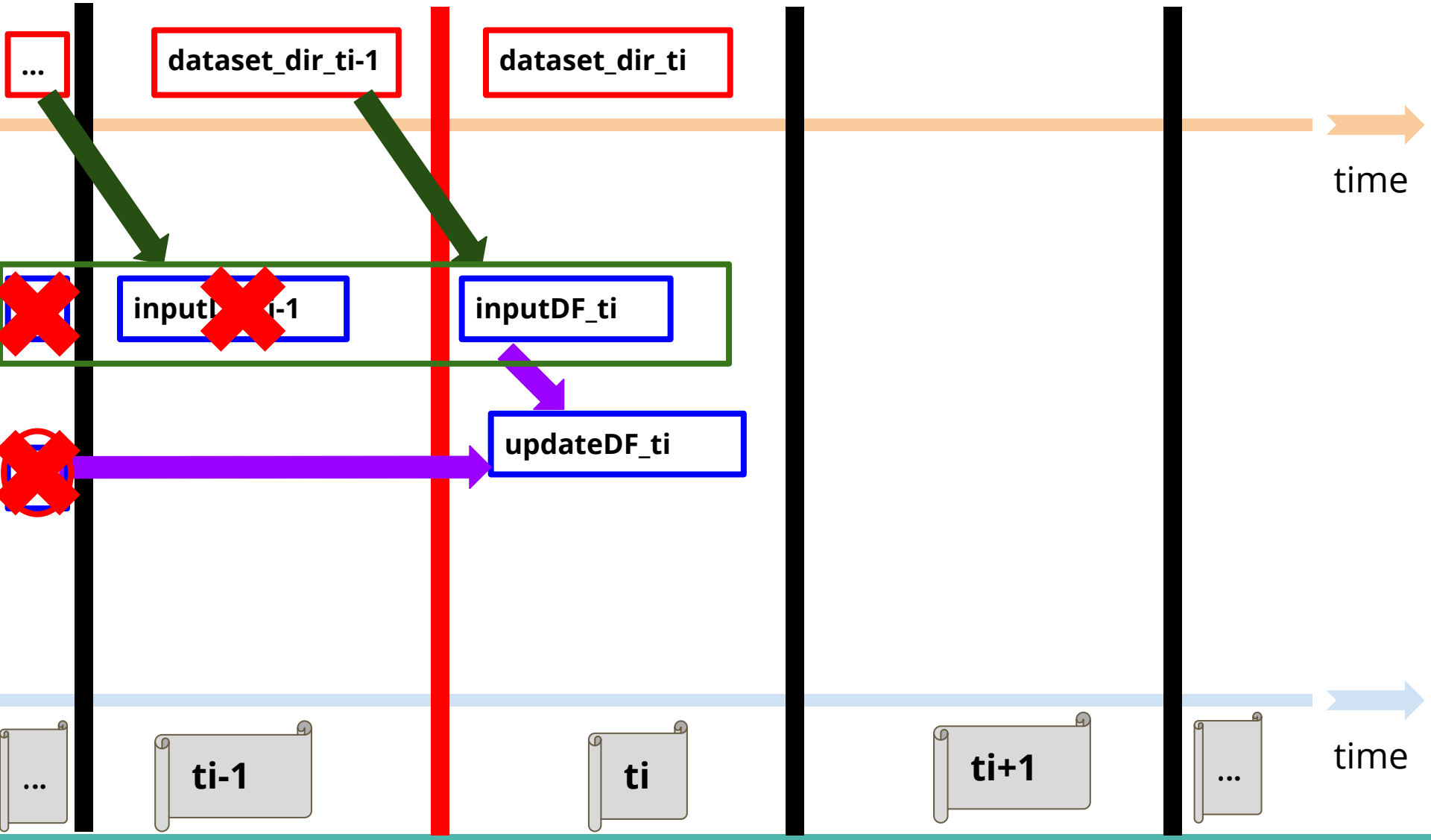
Concept5: Complete Mode with Windows



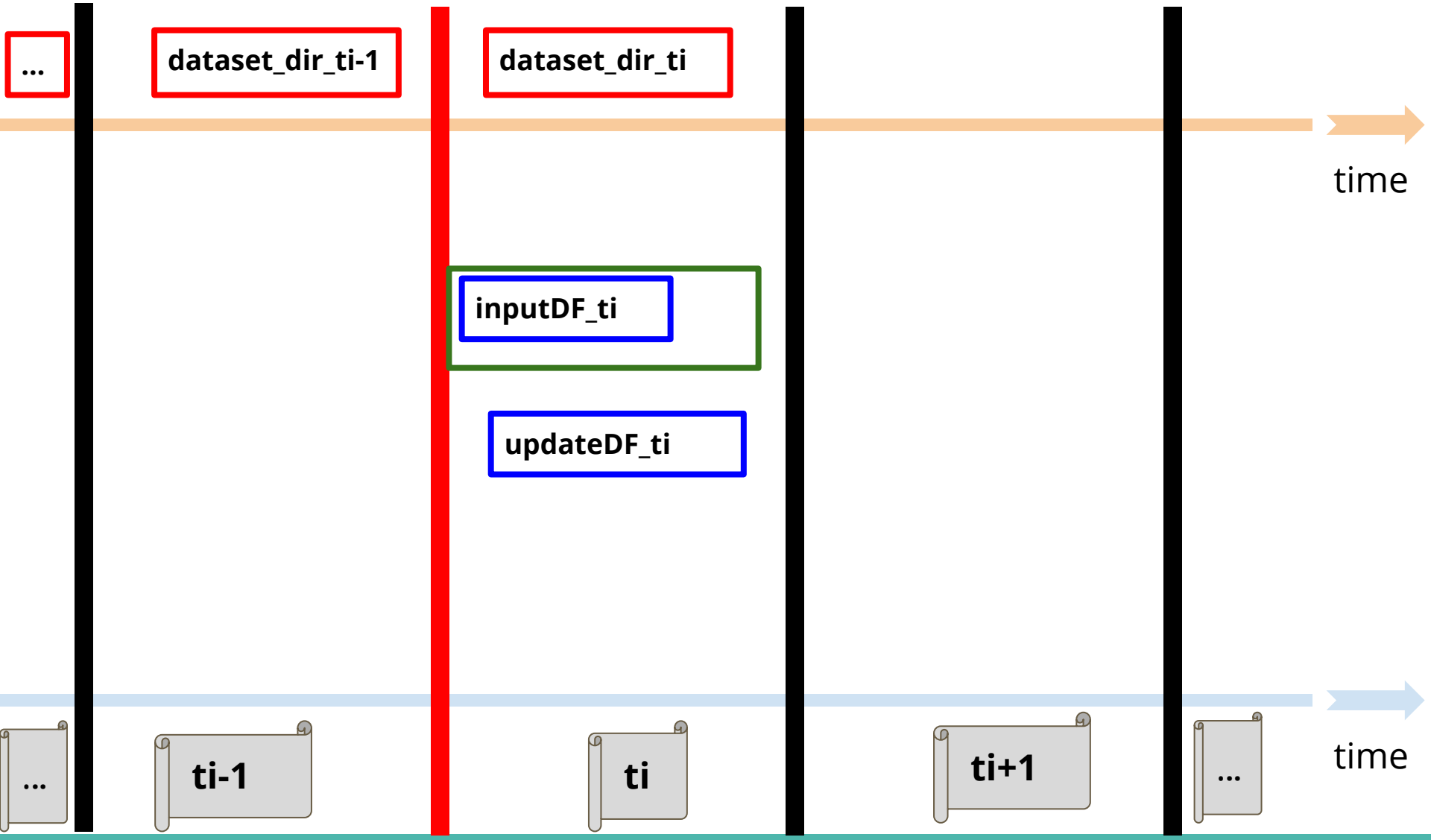
Concept5: Complete Mode with Windows



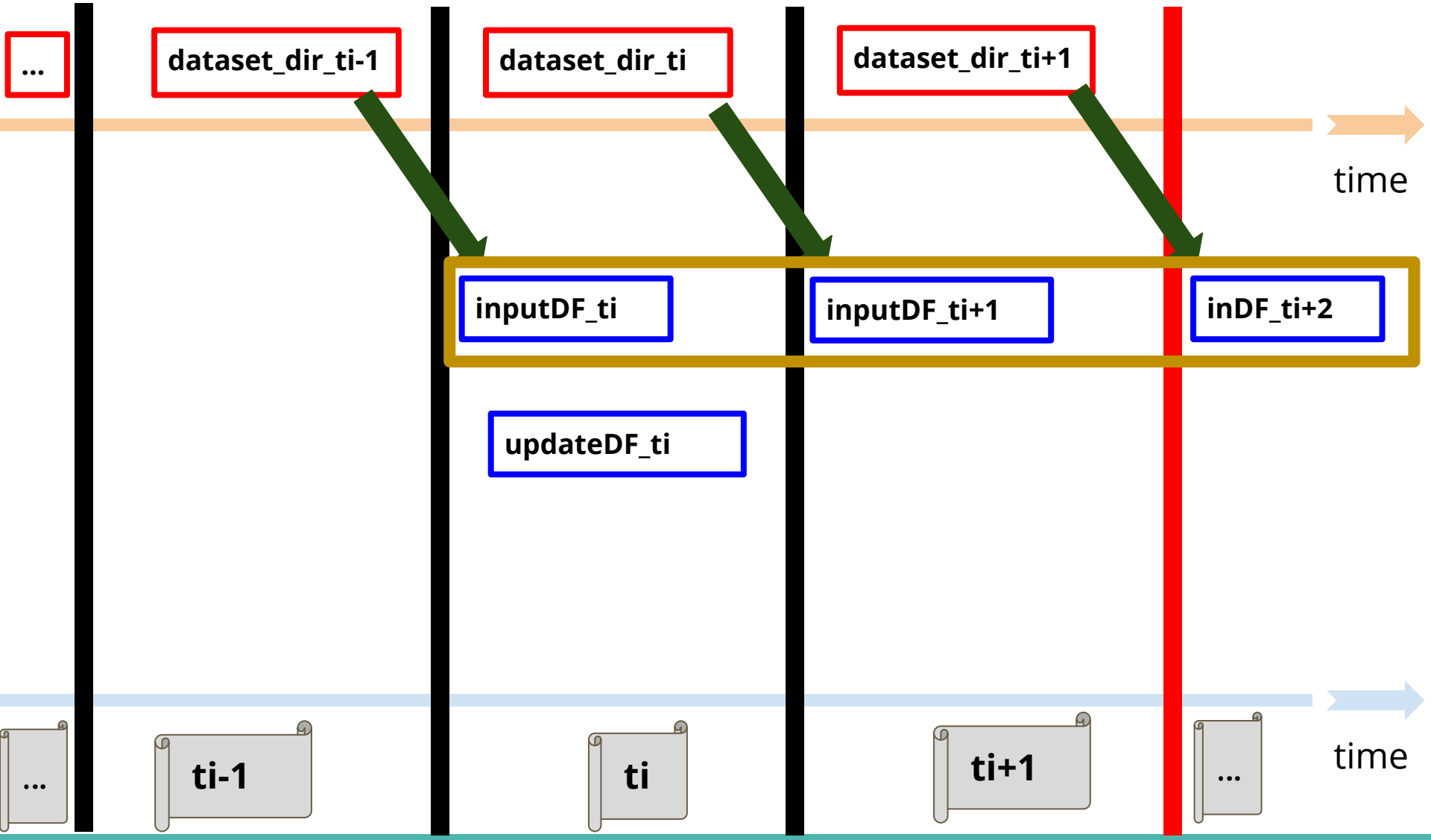
Concept5: Complete Mode with Windows



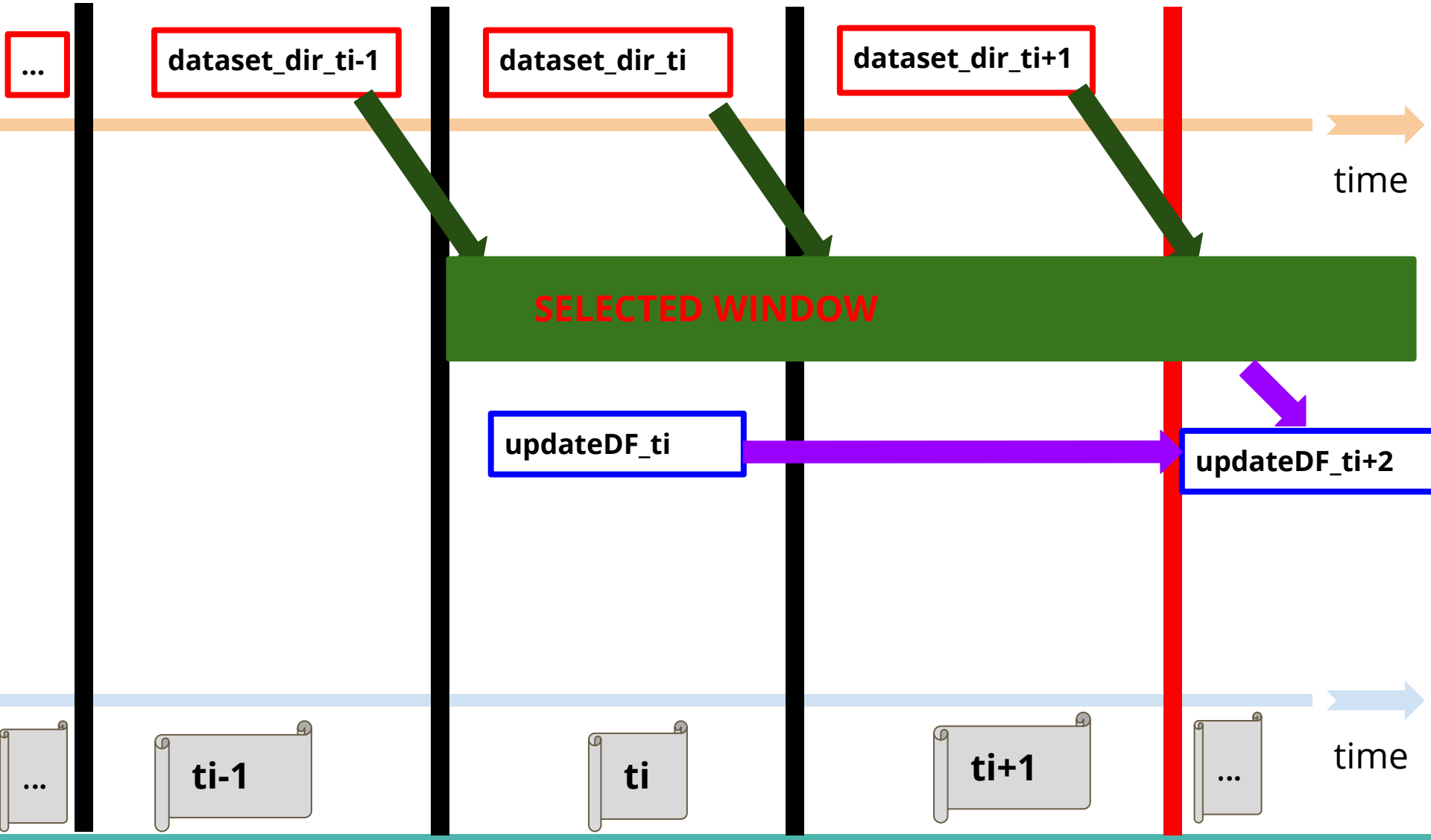
Concept5: Complete Mode with Windows



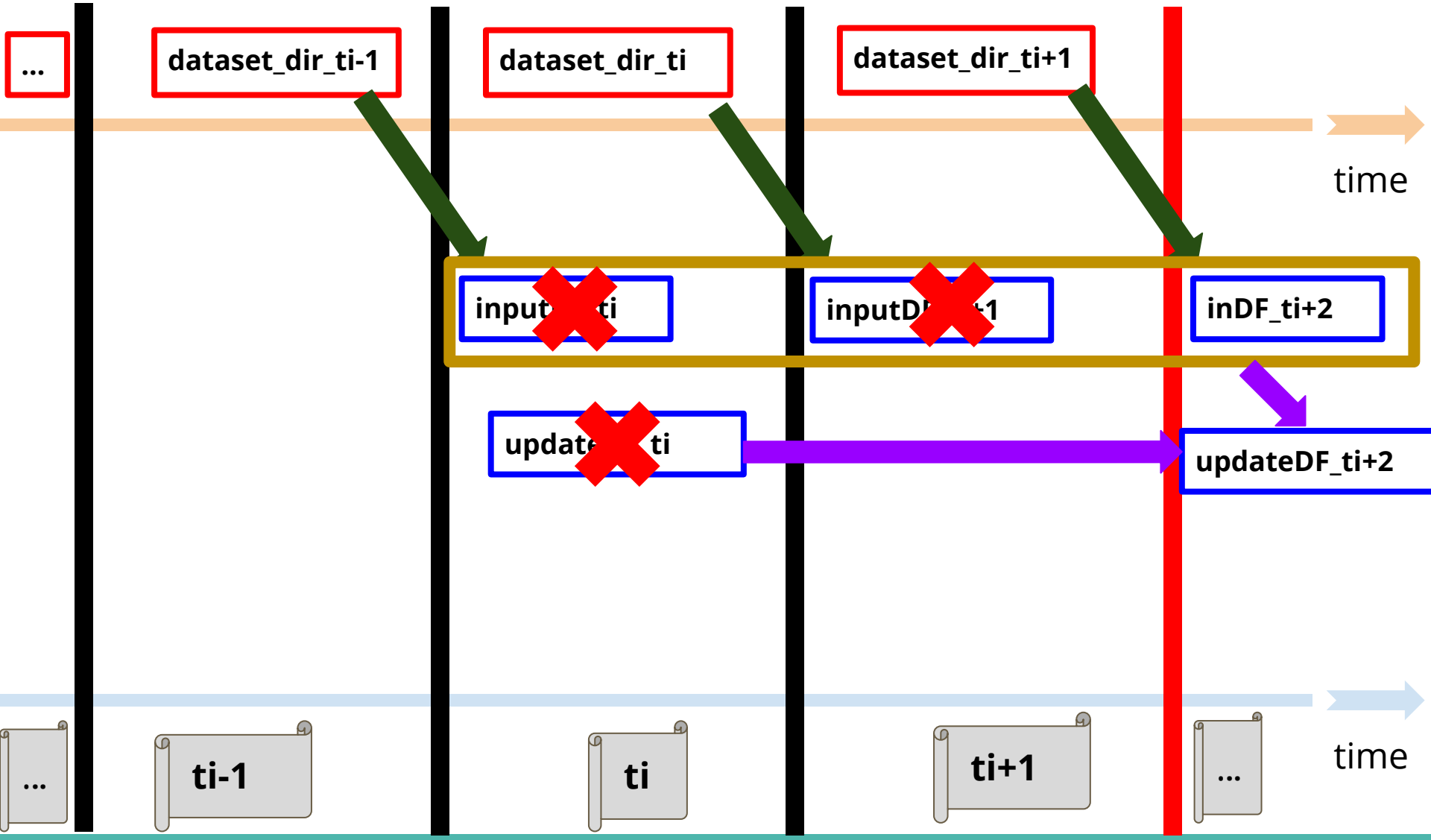
Concept5: Complete Mode with Windows



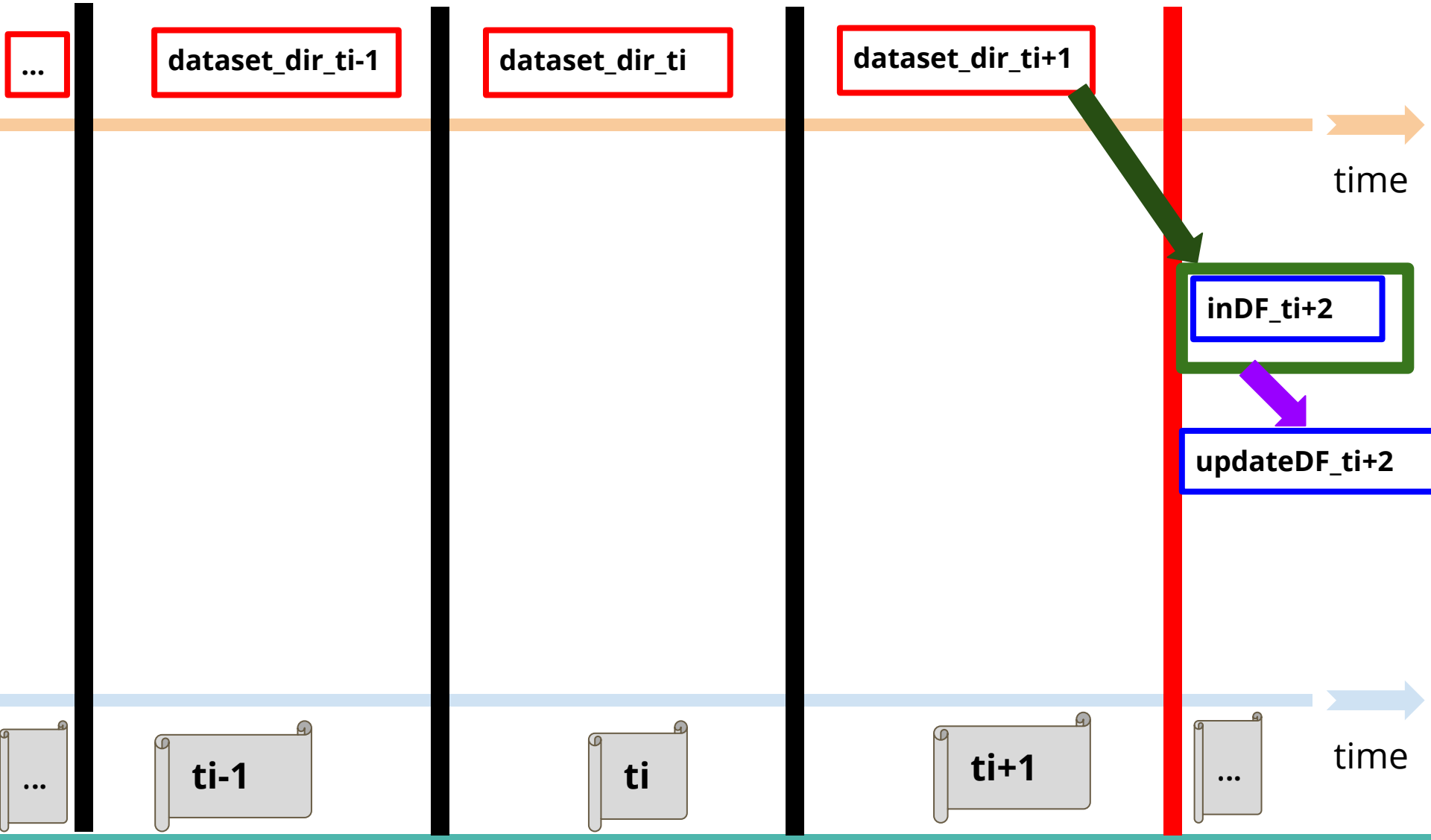
Concept5: Complete Mode with Windows



Concept5: Complete Mode with Windows



Concept5: Complete Mode with Windows



Concept5: Complete Mode with Windows

And with the unbound table representation of the SDF...

inputDF_ti

	Name	City	Eyes
RowK	...	...	...

updateDF_ti

	City	Count
Row1	...	...
Row2	...	...

inputSDF

updateSDF

Concept5: Complete Mode with Windows

And with the unbound table representation of the SDF...

inputDF_ti

	Name	City	Eyes
RowA	...	...	...

inputDF_ti+1

	Name	City	Eyes
RowB	...	...	...

updateDF_ti

	City	Count
Row1	...	...
Row2	...	...

inputSDF

updateSDF

Concept5: Complete Mode with Windows

And with the unbound table representation of the SDF…

inputDF_ti

	Name	City	Eyes
RowA	…	…	…

inputDF_ti+1

	Name	City	Eyes
RowB	…	…	…

inputDF_ti+2

	Name	City	Eyes
RowB	…	…	…

inputSDF

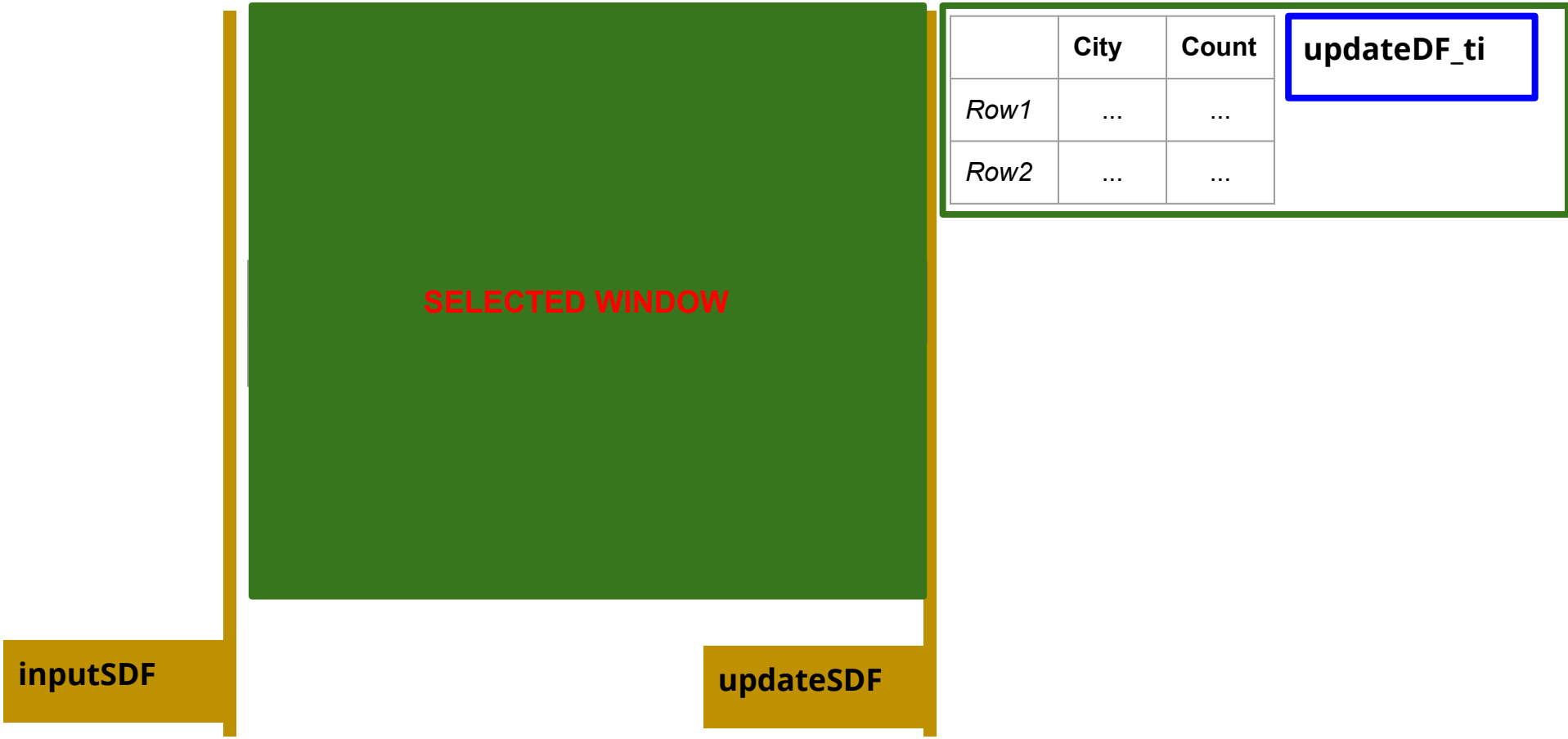
updateDF_ti

	City	Count
Row1	…	…
Row2	…	…

updateSDF

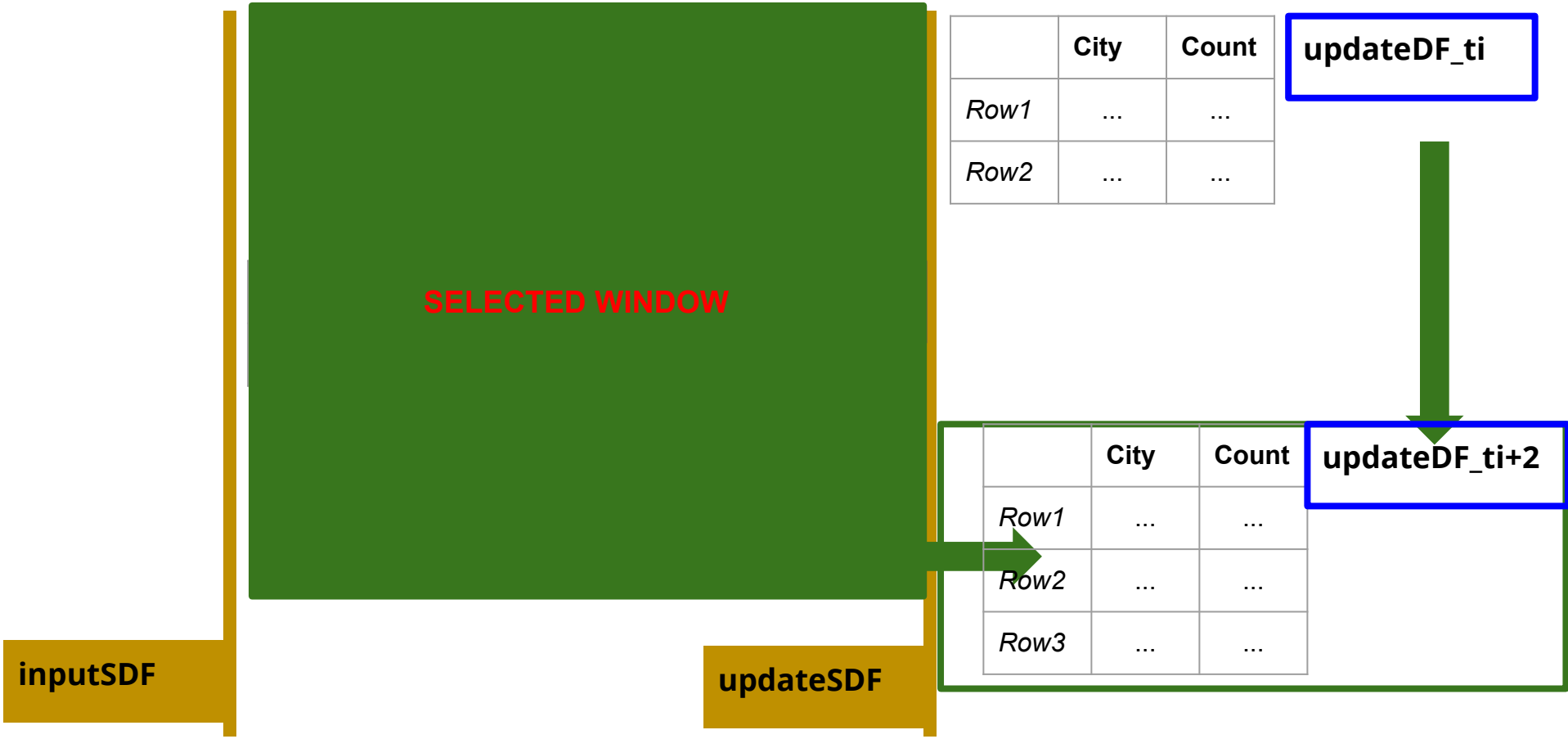
Concept5: Complete Mode with Windows

And with the unbound table representation of the SDF...



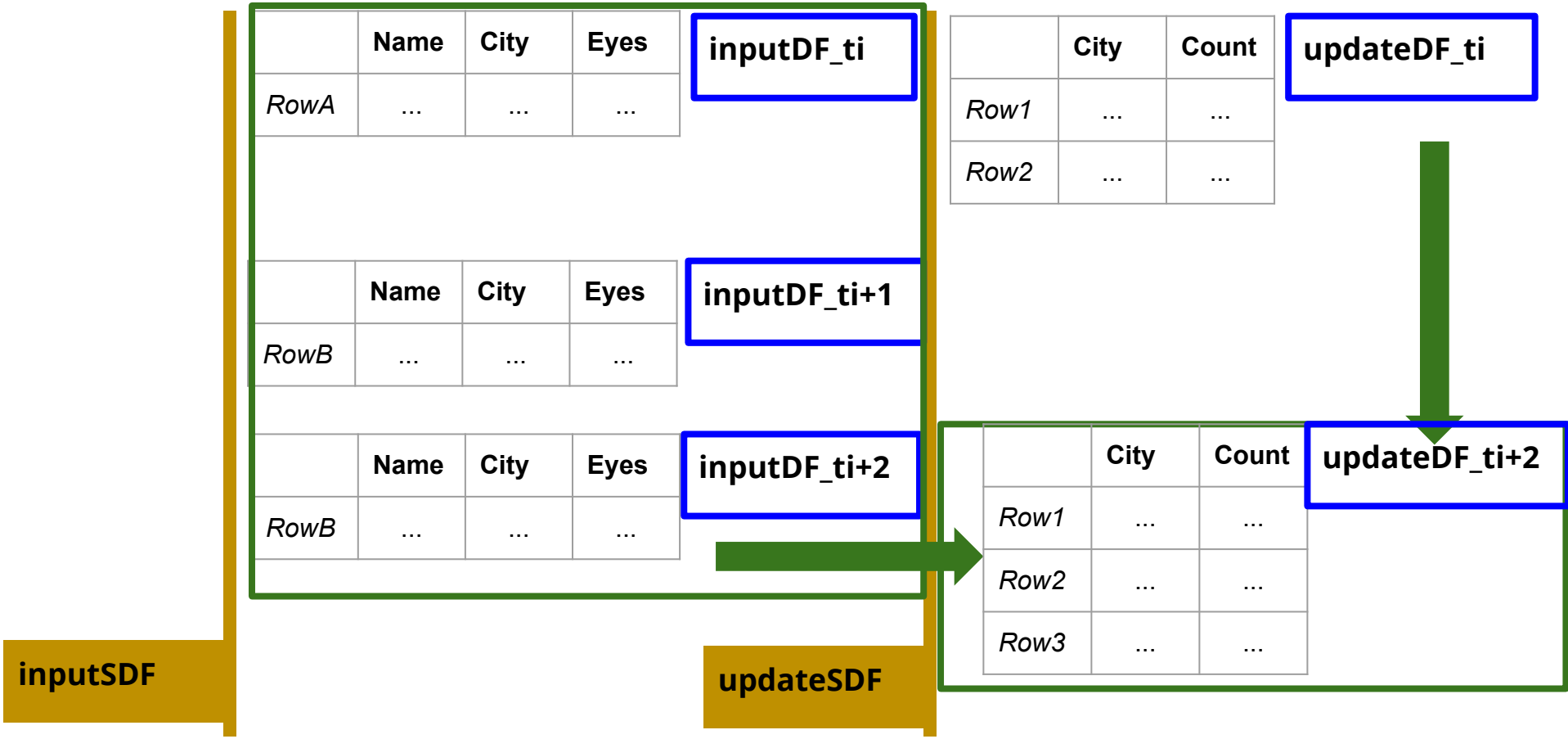
Concept5: Complete Mode with Windows

And with the unbound table representation of the SDF...



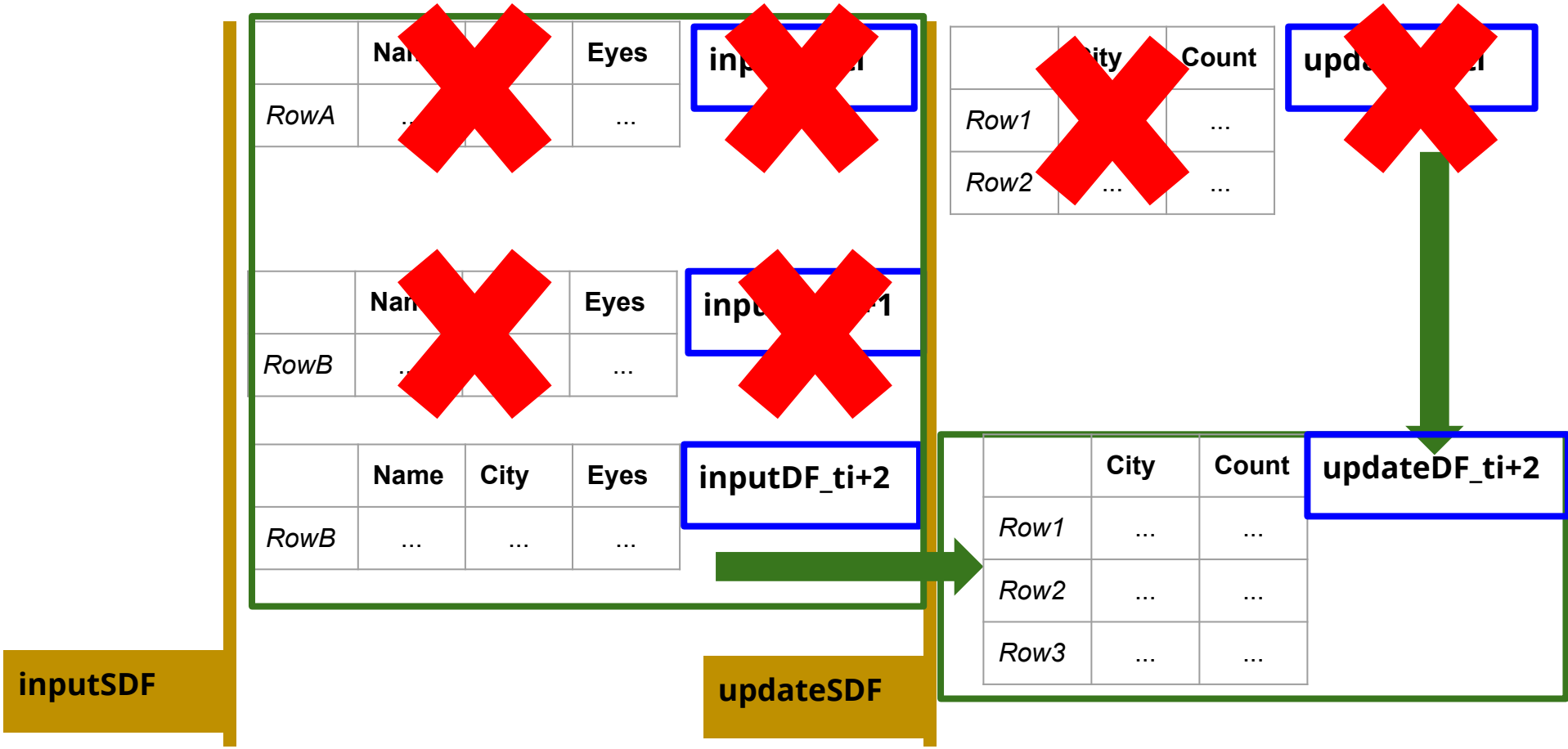
Concept5: Complete Mode with Windows

And with the unbound table representation of the SDF...



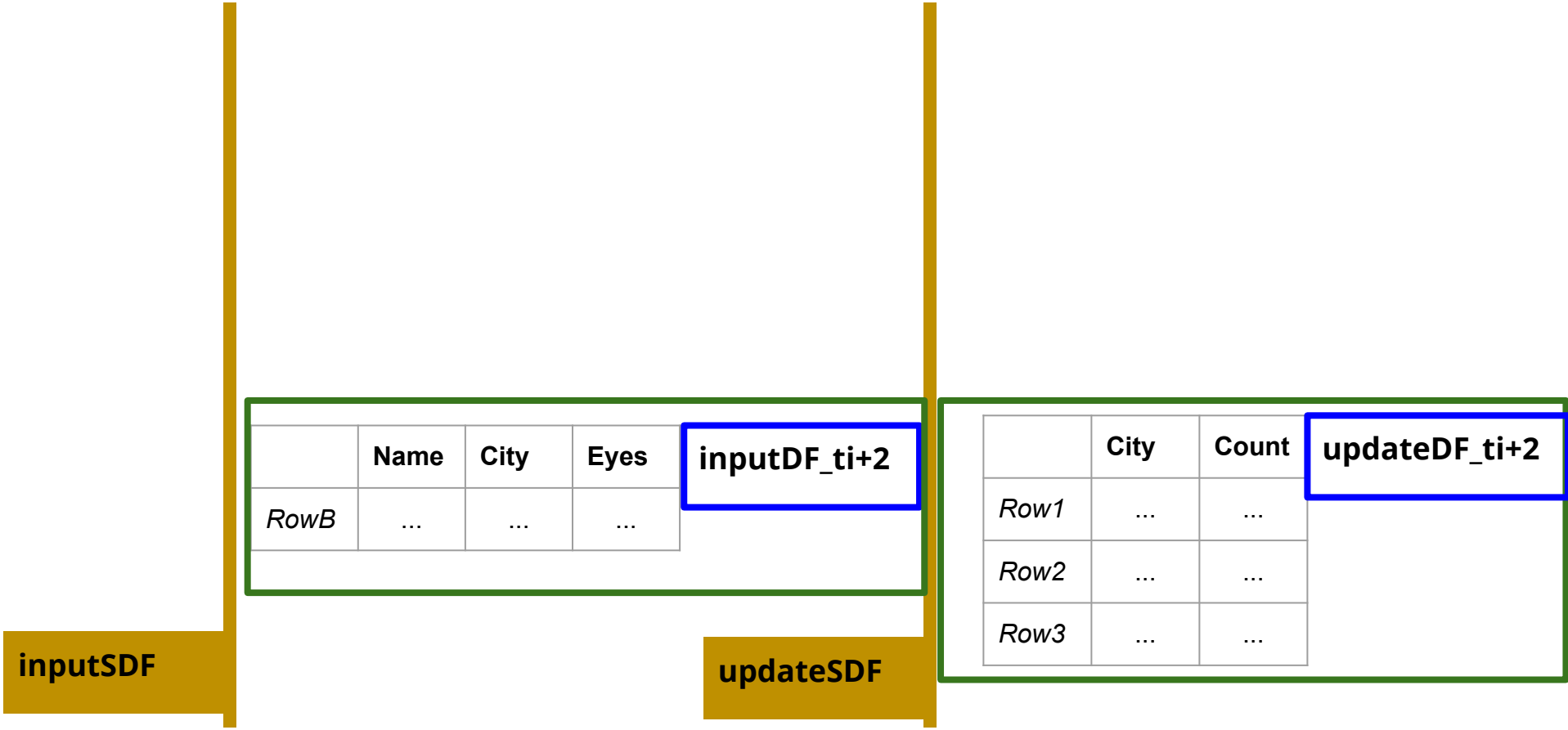
Concept5: Complete Mode with Windows

And with the unbound table representation of the SDF…



Concept5: Complete Mode with Windows

And with the unbound table representation of the SDF...



Concept5: Complete Mode with Windows

Concept 5:

This behaviour is called
a **SDF** in **Complete Mode**
reasoning over time window!

Outline

1. Setting Up the Context.
2. From DStream to SDF.
   a. Concept1: SDF.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
3. Practising with the Concepts.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.
   a. A Program-Driven Example.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
   f. Console Sink vs. File Sink.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.
   a. A Program-Driven Example.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
   f. Console Sink vs. File Sink.

A Program-Driven Example

Let's suppose we have the following simple dataset

Dataset

F1.txt

Argentina Brazil Colombia\n

F2.txt

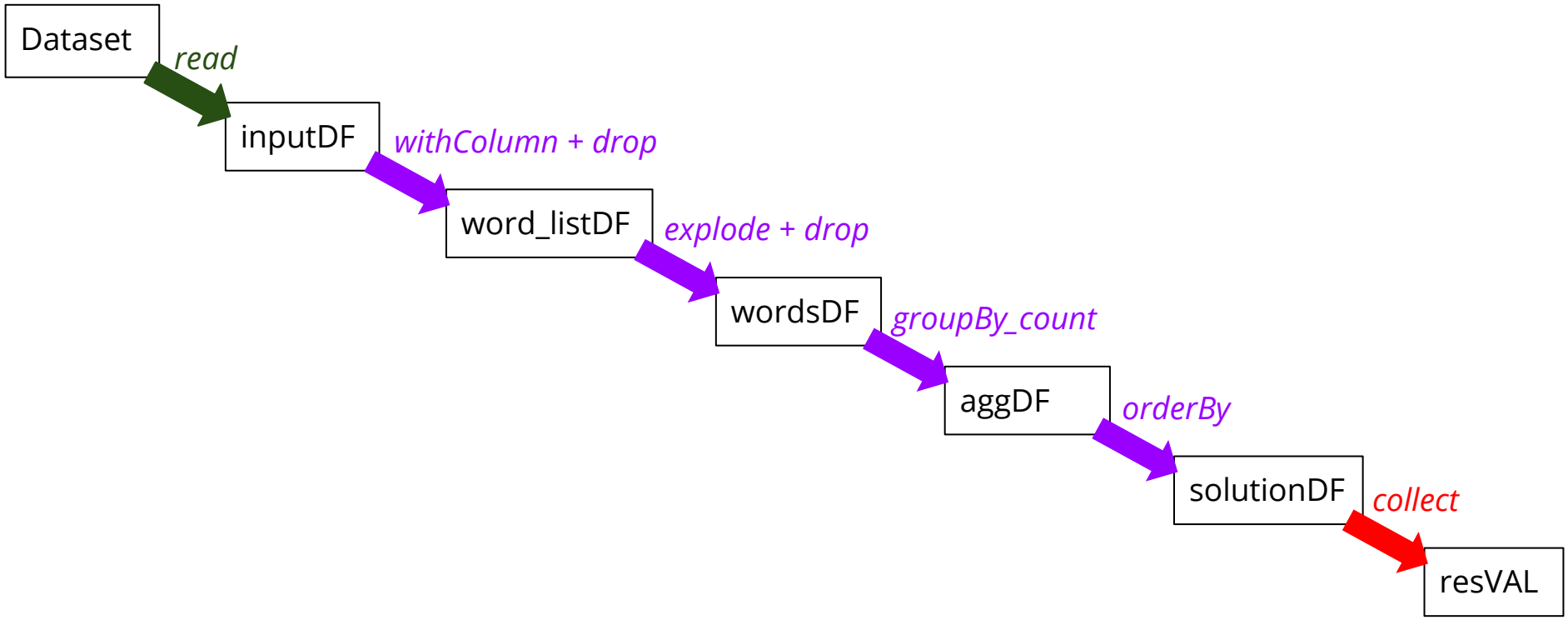
Argentina Brazil\n

F3.txt

Argentina Colombia Colombia Colombia\n

A Program-Driven Example

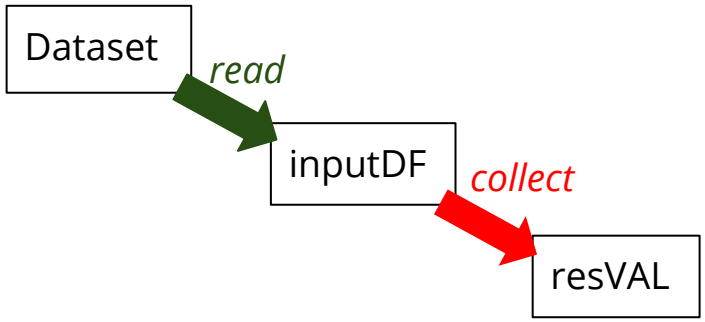
Let's suppose we have the following Spark SQL program
p01_intro_DF.py



A Program-Driven Example

Let's analyse the program operation by operation

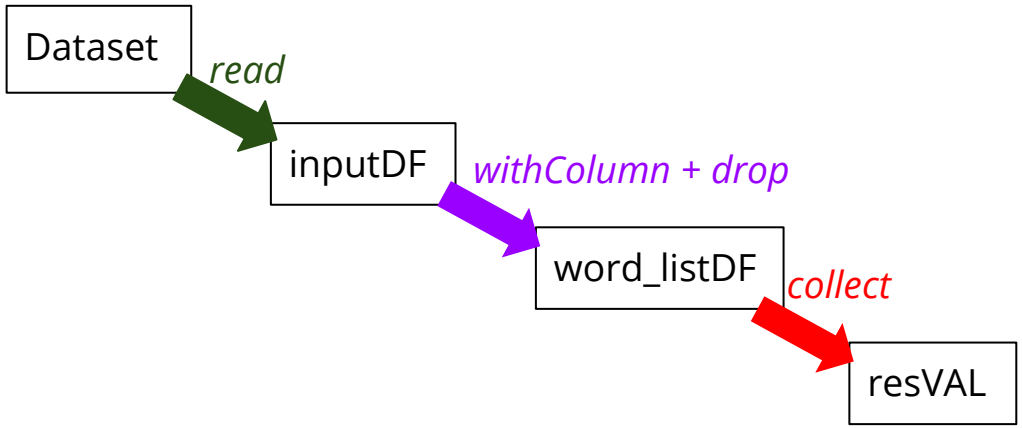
p01_intro_DF.py




```
Row(value='Argentina Brazil Colombia')
Row(value='Argentina Brazil')
Row(value='Argentina Colombia Colombia Colombia')
```

A Program-Driven Example

Let's analyse the program operation by operation
p01_intro_DF.py

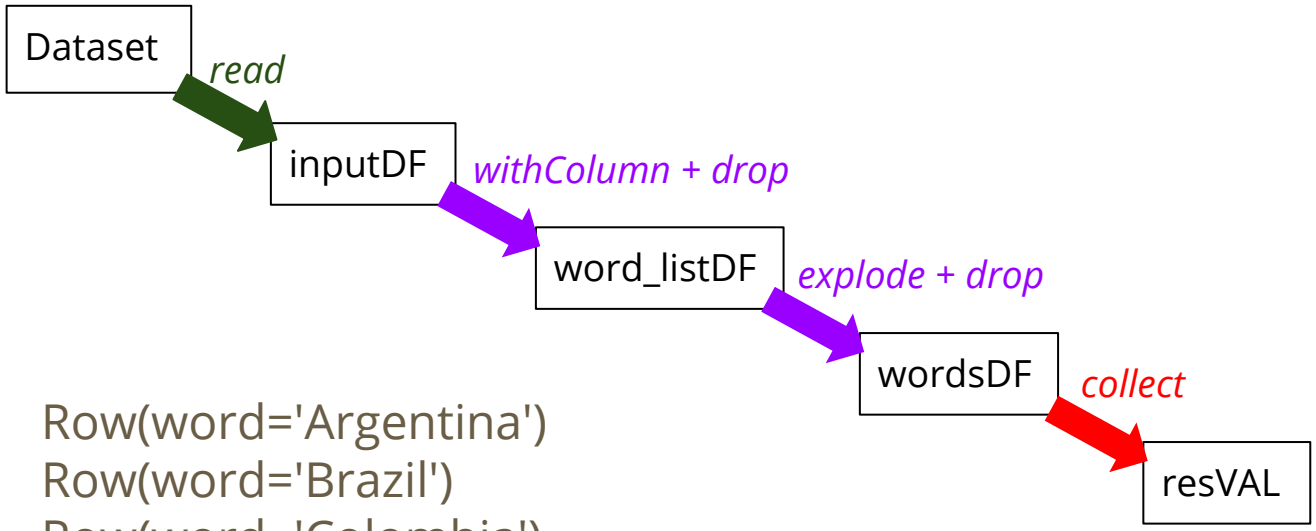



```
Row(wordList=['Argentina', 'Brazil', 'Colombia'])
Row(wordList=['Argentina', 'Brazil'])
Row(wordList=['Argentina', 'Colombia', 'Colombia', 'Colombia'])
```

A Program-Driven Example

Let's analyse the program operation by operation

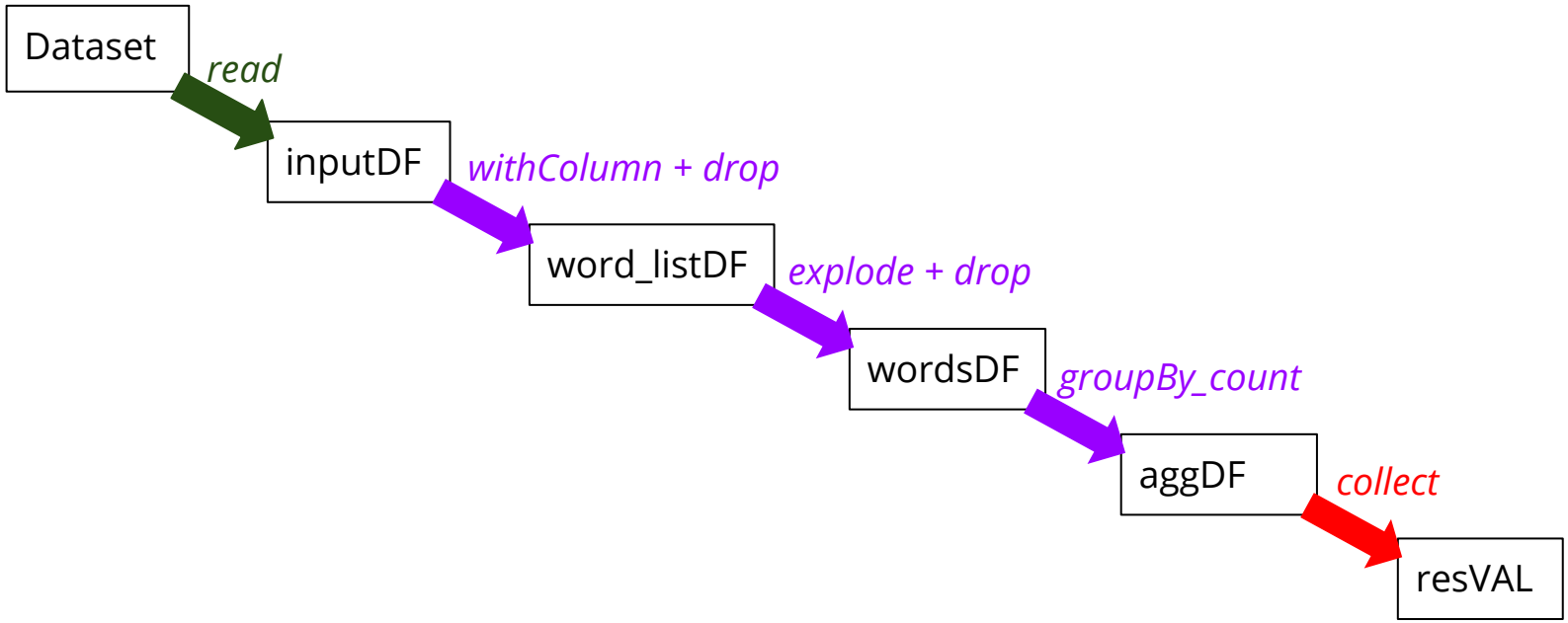
p01_intro_DF.py




```
Row(word='Argentina')
Row(word='Brazil')
Row(word='Colombia')
Row(word='Argentina')
Row(word='Brazil')
Row(word='Argentina')
Row(word='Colombia')
Row(word='Colombia')
Row(word='Colombia')
```

A Program-Driven Example

Let's analyse the program operation by operation
p01_intro_DF.py

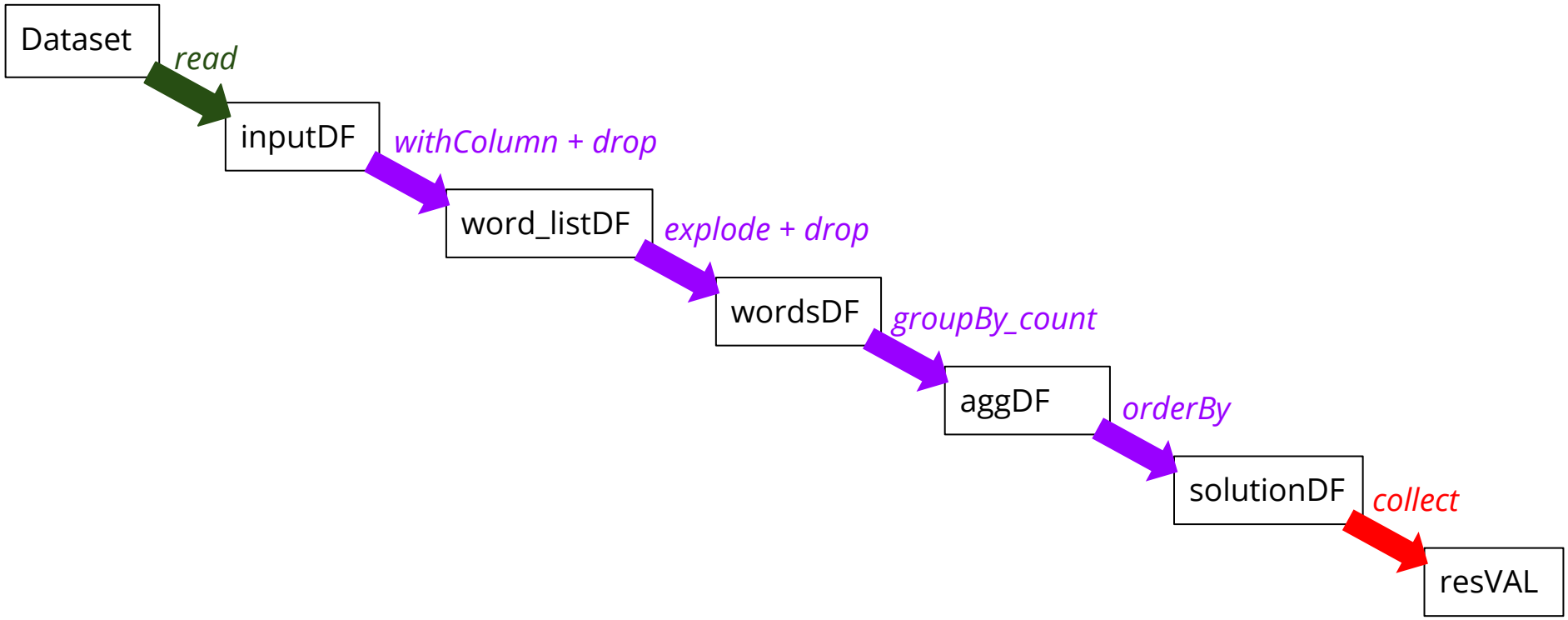


Row(word='Argentina', count=3)
Row(word='Brazil', count=2)
Row(word='Colombia', count=4)

A Program-Driven Example

Let's analyse the program operation by operation

p01_intro_DF.py



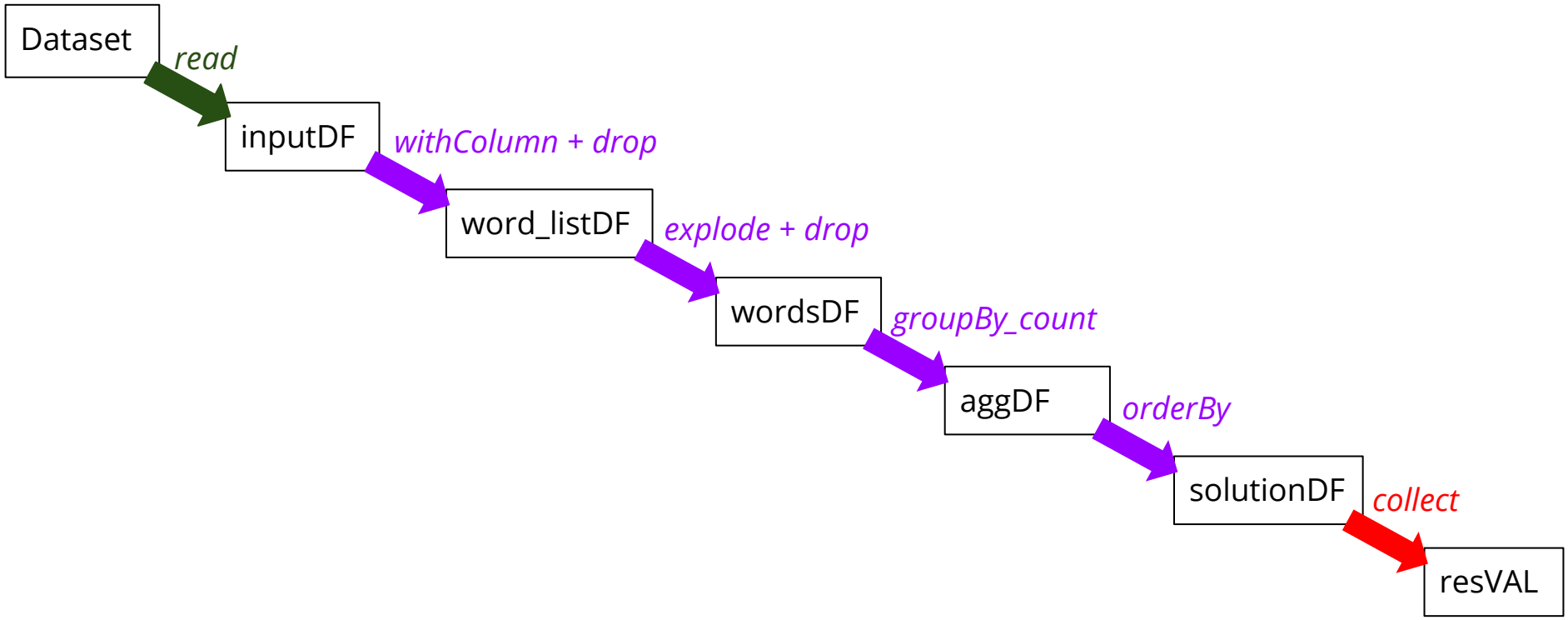
Row(word='Colombia', count=4)
Row(word='Argentina', count=3)
Row(word='Brazil', count=2)

A Program-Driven Example

Now let's suppose the streaming of this dataset, with one new file each 15 seconds

A Program-Driven Example

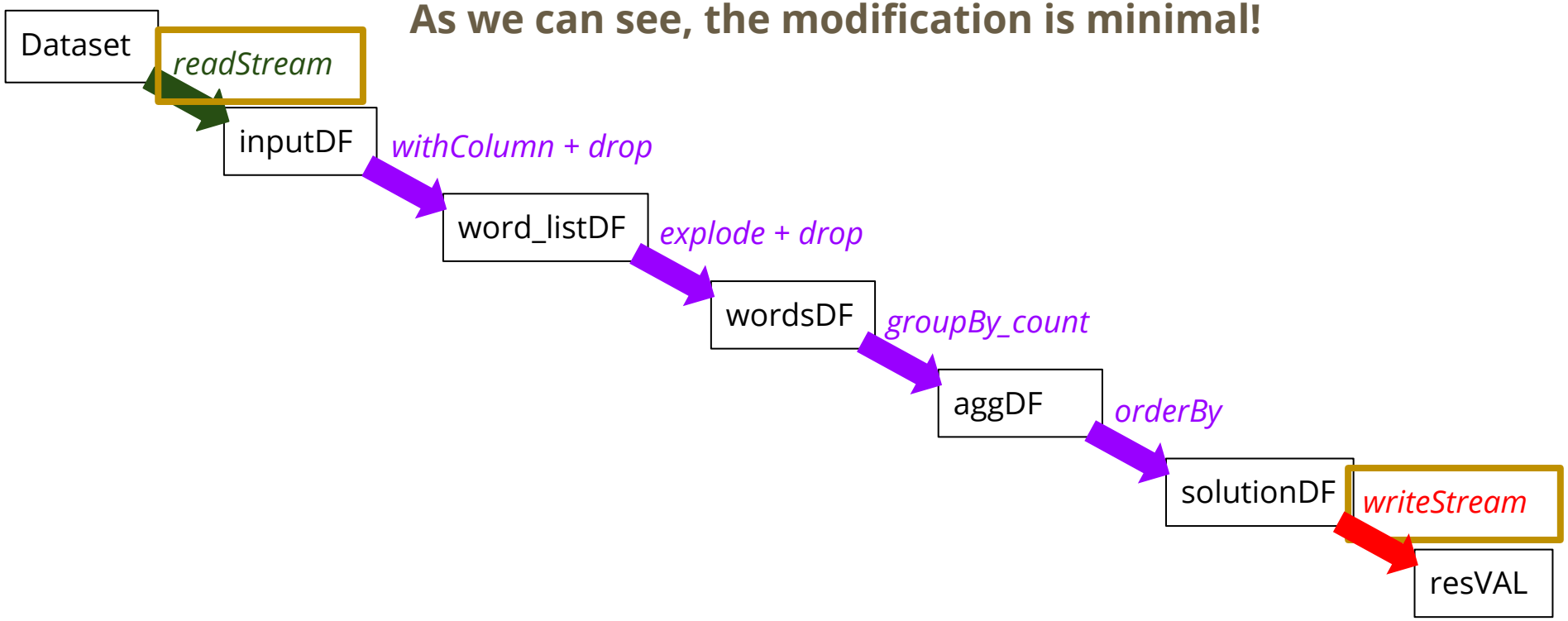
Let's modify our Spark SQL program to work in Spark Structured Streaming



A Program-Driven Example

Let's modify our Spark SQL program to work in Spark Structured Streaming

As we can see, the modification is minimal!



A Program-Driven Example

Finally, let's use this program to reason about the 4 concepts of last section:

A Program-Driven Example

Finally, let's use this program to reason about the concepts of last section:

- Concept 1: SDF is an amalgamation of DFs over time *(nothing to show here)*.

A Program-Driven Example

Finally, let's use this program to reason about the concepts of last section:

- Concept 1: SDF is an amalgamation of DFs over time *(nothing to show here)*.
- Concept 2: SDF in Append Mode.

A Program-Driven Example

Finally, let's use this program to reason about the concepts of last section:

- Concept 1: SDF is an amalgamation of DFs over time *(nothing to show here)*.
- Concept 2: SDF in Append Mode.
- Concept 3: SDF in Append Mode reasoning over time window.

A Program-Driven Example

Finally, let's use this program to reason about the concepts of last section:

- Concept 1: SDF is an amalgamation of DFs over time *(nothing to show here)*.
- Concept 2: SDF in Append Mode.
- Concept 3: SDF in Append Mode reasoning over time window.
- Concept 4: SDF in Complete Mode.

A Program-Driven Example

Finally, let's use this program to reason about the concepts of last section:

- Concept 1: SDF is an amalgamation of DFs over time *(nothing to show here)*.
- Concept 2: SDF in Append Mode.
- Concept 3: SDF in Append Mode reasoning over time window.
- Concept 4: SDF in Complete Mode.
- Concept 5: SDF in Complete Mode reasoning over time window.

A Program-Driven Example

For each concept:

A Program-Driven Example

For each concept:

Let's first guess what result do we expect to obtain...

A Program-Driven Example

For each concept:

Let's first guess what result do we expect to obtain...

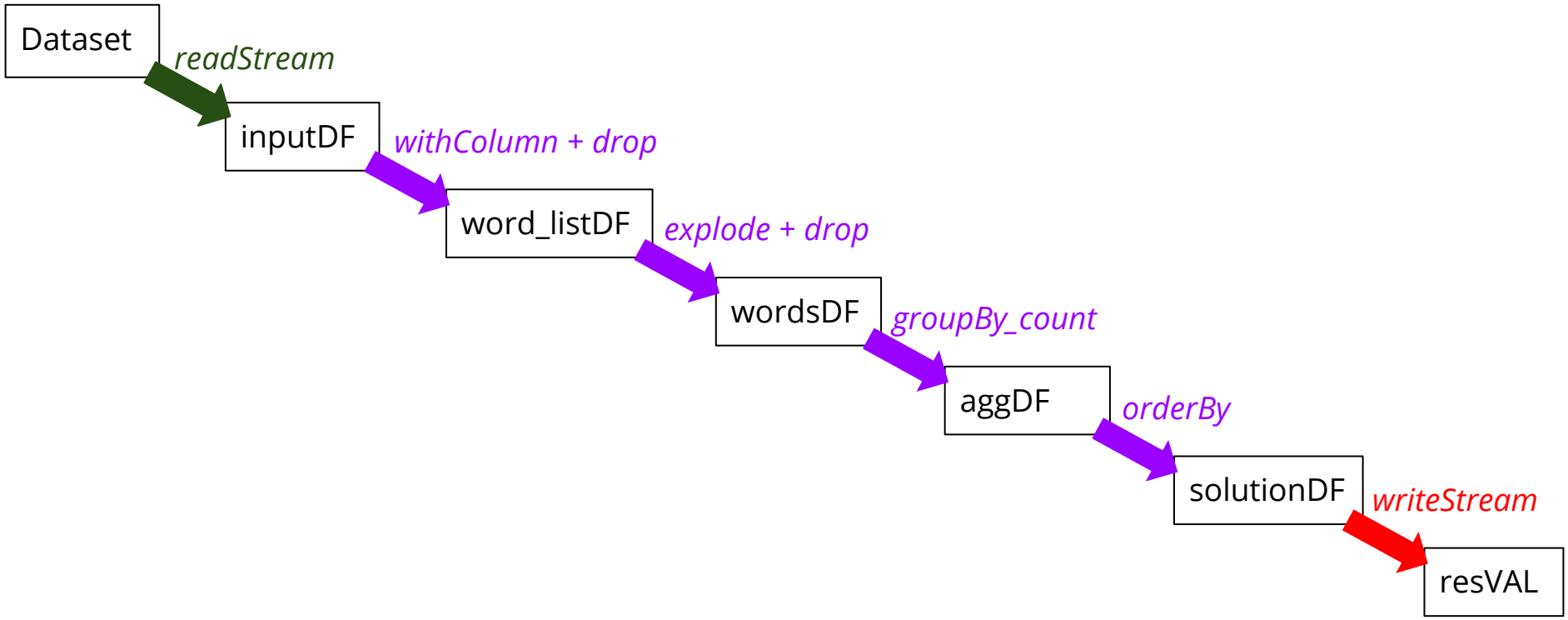
...to then see what do we actually obtain (and how do we need to modify our original program in order to obtain it).

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.
   a. A Program-Driven Example.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
   f. Console Sink vs. File Sink.

Concept2: Append Mode

We have our program



Concept2: Append Mode

We have our dataset

Concept2: Append Mode

What do we expect to obtain?

Concept2: Append Mode

Time Step 0:

word	count
Argentina	1
Brazil	1
Colombia	1

Concept2: Append Mode

Time Step 1:

```
+--------------+--------------+
|     word     | count        |
+--------------+--------------+
|Argentina     |      1       |
|   Brazil     |      1       |
+--------------+--------------+
```

Concept2: Append Mode

Time Step 2:

```
+--------------+--------------+
|     word     | count        |
+--------------+--------------+
|Colombia      |      3       |
| Argentina    |      1       |
+--------------+--------------+
```

Concept2: Append Mode

What do we actually obtain?

Concept2: Append Mode

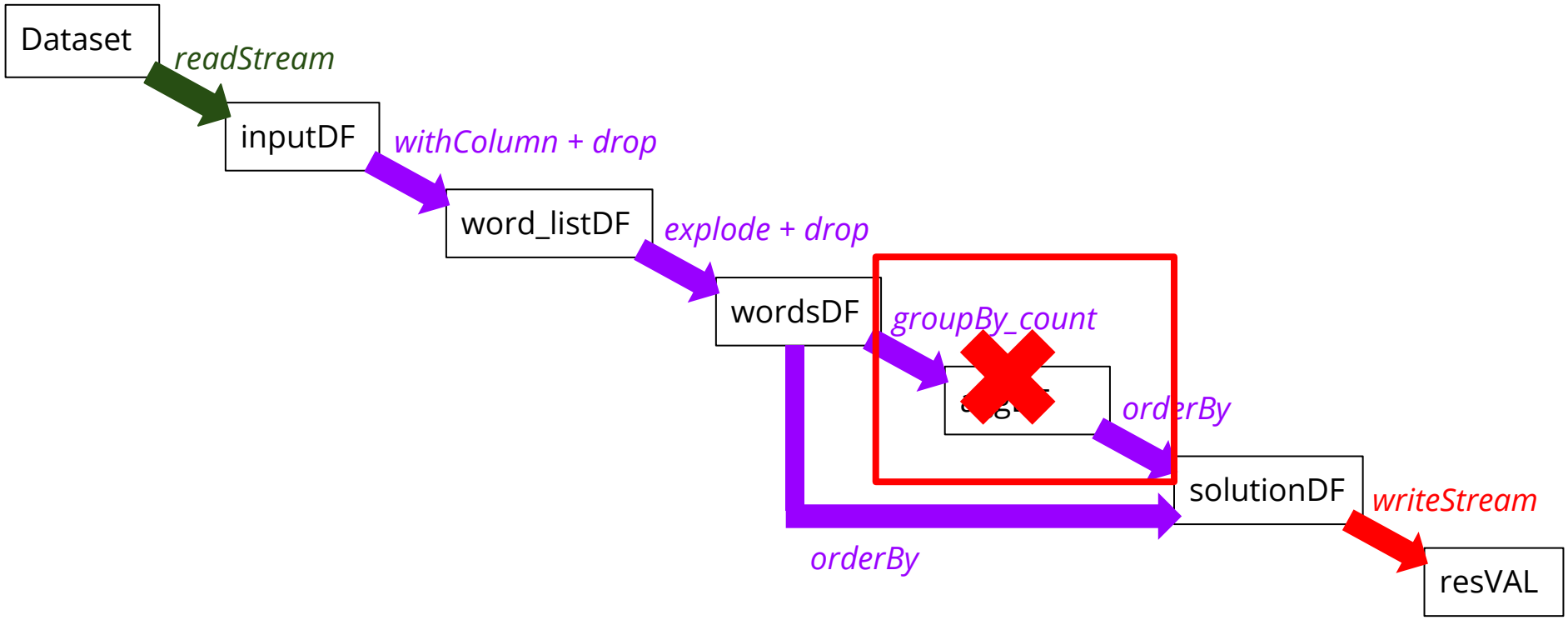
Program **p02_ERROR_groupBy.py** fails:

ERROR: Append output mode not supported when there are streaming aggregations on streaming DataFrames/DataSets without watermark.

Concept2: Append Mode

We modify our program **p02_ERROR_groupBy.py** into **p03_ERROR_orderBy.py** by:

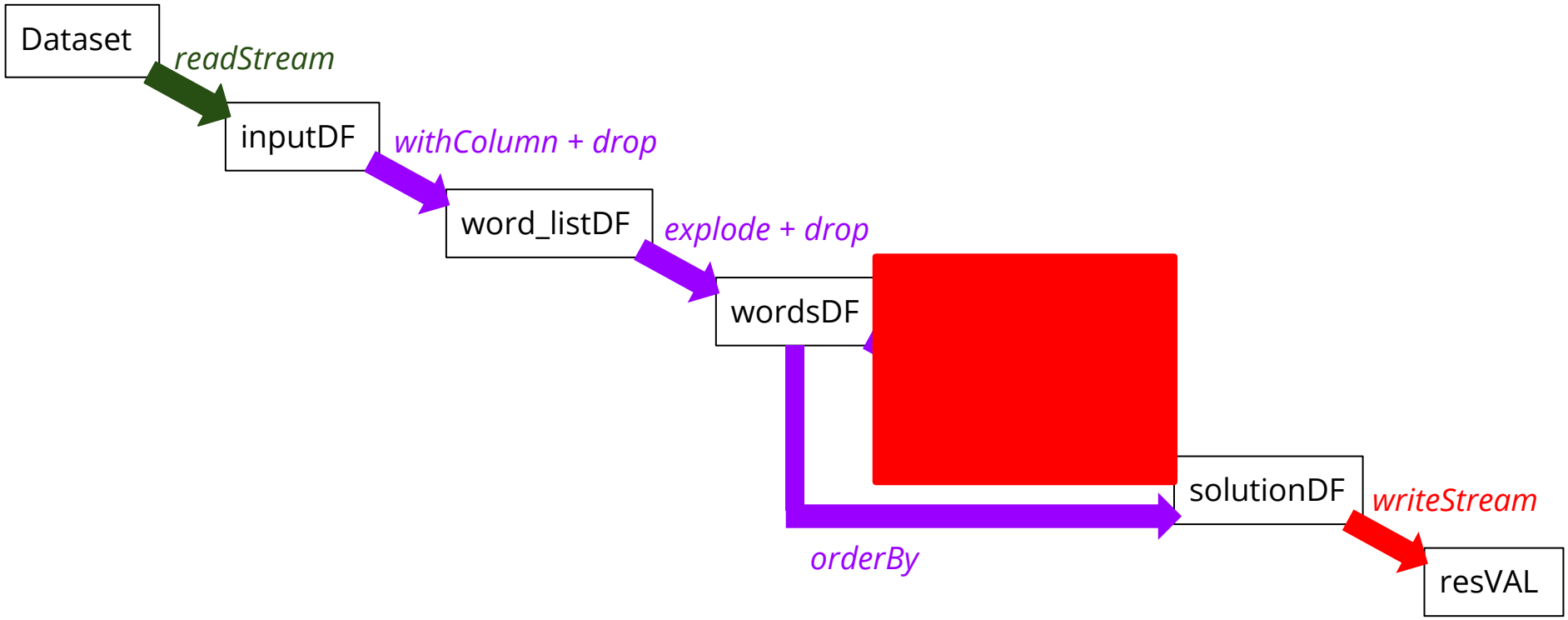
1. Not to do any aggregation.



Concept2: Append Mode

We modify our program **p02_ERROR_groupBy.py** into **p03_ERROR_orderBy.py** by:

1. Not to do any aggregation.



Concept2: Append Mode

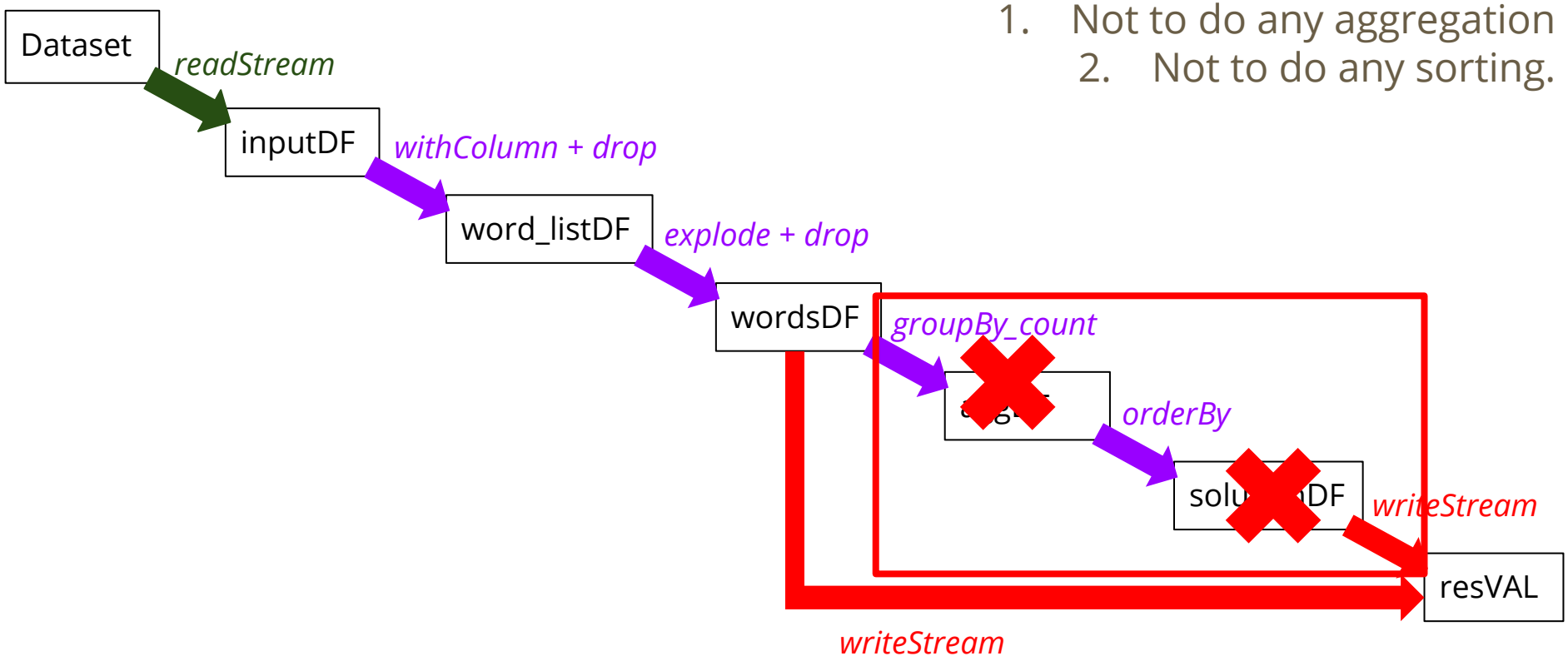
Program **p03_ERROR_orderBy.py** fails:

ERROR: Sorting is not supported on streaming DataFrames/Datasets,
unless it is on aggregated DataFrame/Dataset in Complete output mode.

Concept2: Append Mode

We modify our program **p03_ERROR_orderBy.py** into **p04_ok_no_groupBy_no_orderBy.py** by:

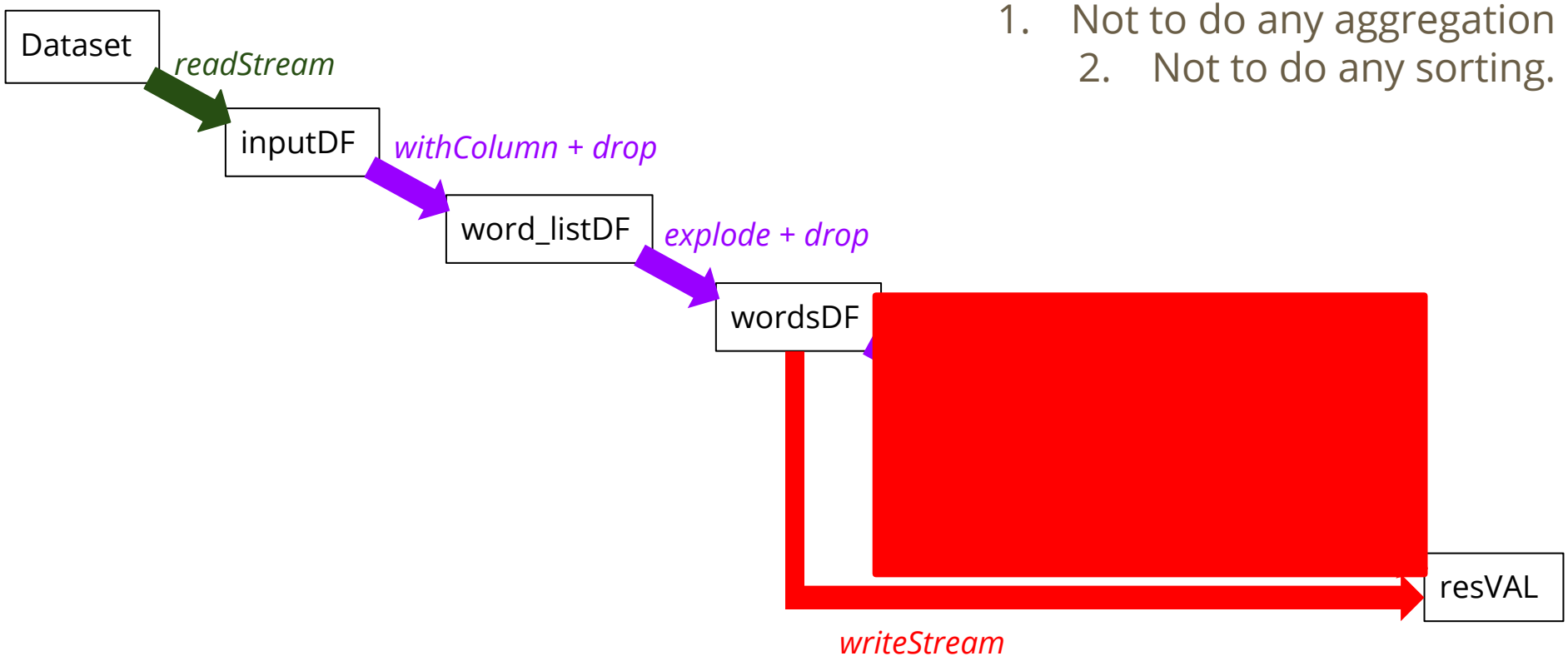
1. Not to do any aggregation
2. Not to do any sorting.



Concept2: Append Mode

We modify our program **p03_ERROR_orderBy.py** into **p04_ok_no_groupBy_no_orderBy.py** by:

1. Not to do any aggregation
2. Not to do any sorting.



Concept2: Append Mode

Program **p04_ok_no_groupBy_no_orderBy.py** executes without error.

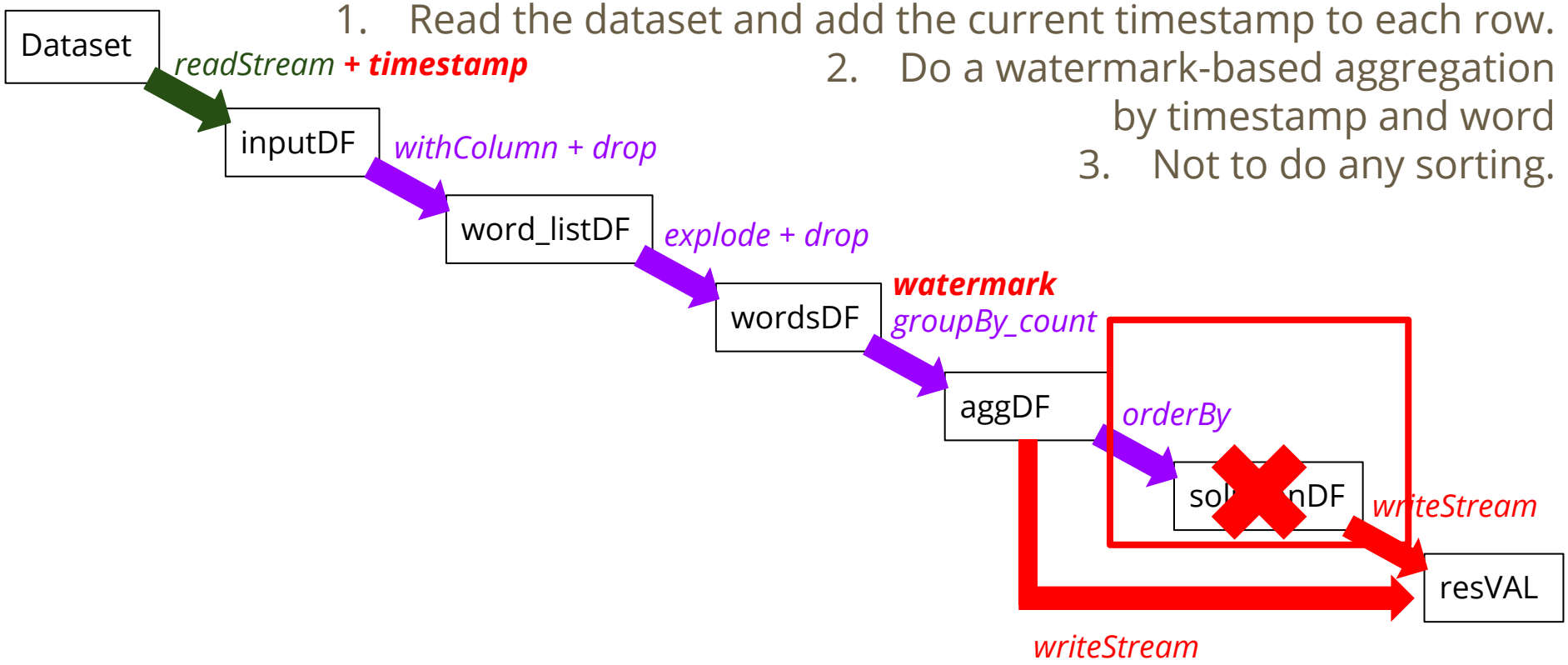
Its results are presented in the file **p04_ok_no_groupBy_no_orderBy.txt**

Obviously, its output does not aggregate nor sort,
so it differs from what we were originally expecting.

Concept2: Append Mode

We modify our program **p04_ok_no_groupBy_no_orderBy.py** into **p05_ok_but_misses_time_interval.py** by:

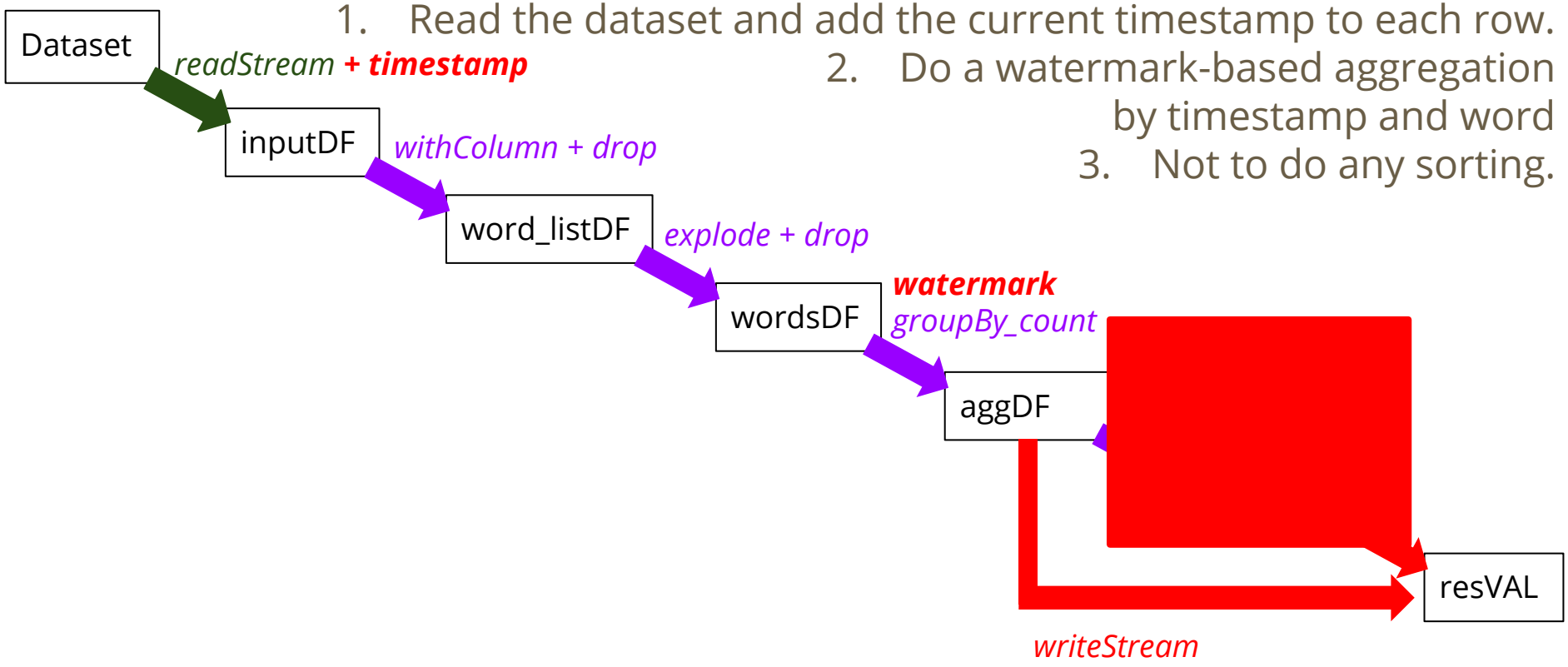
1. Read the dataset and add the current timestamp to each row.
2. Do a watermark-based aggregation by timestamp and word
3. Not to do any sorting.



Concept2: Append Mode

We modify our program **p04_ok_no_groupBy_no_orderBy.py** into **p05_ok_but_misses_time_interval.py** by:

1. Read the dataset and add the current timestamp to each row.
2. Do a watermark-based aggregation by timestamp and word
3. Not to do any sorting.



Concept2: Append Mode

Program **p05_ok_but_misses_time_interval.py** executes without error.

Its results are presented in the file **p05_ok_but_misses_time_interval.txt**

The results are now as we originally expecting.
However, for a reason we can't understand, the results for the last time step are never computed.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.
   a. A Program-Driven Example.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
   f. Console Sink vs. File Sink.

Concept3: Append Mode with Windows

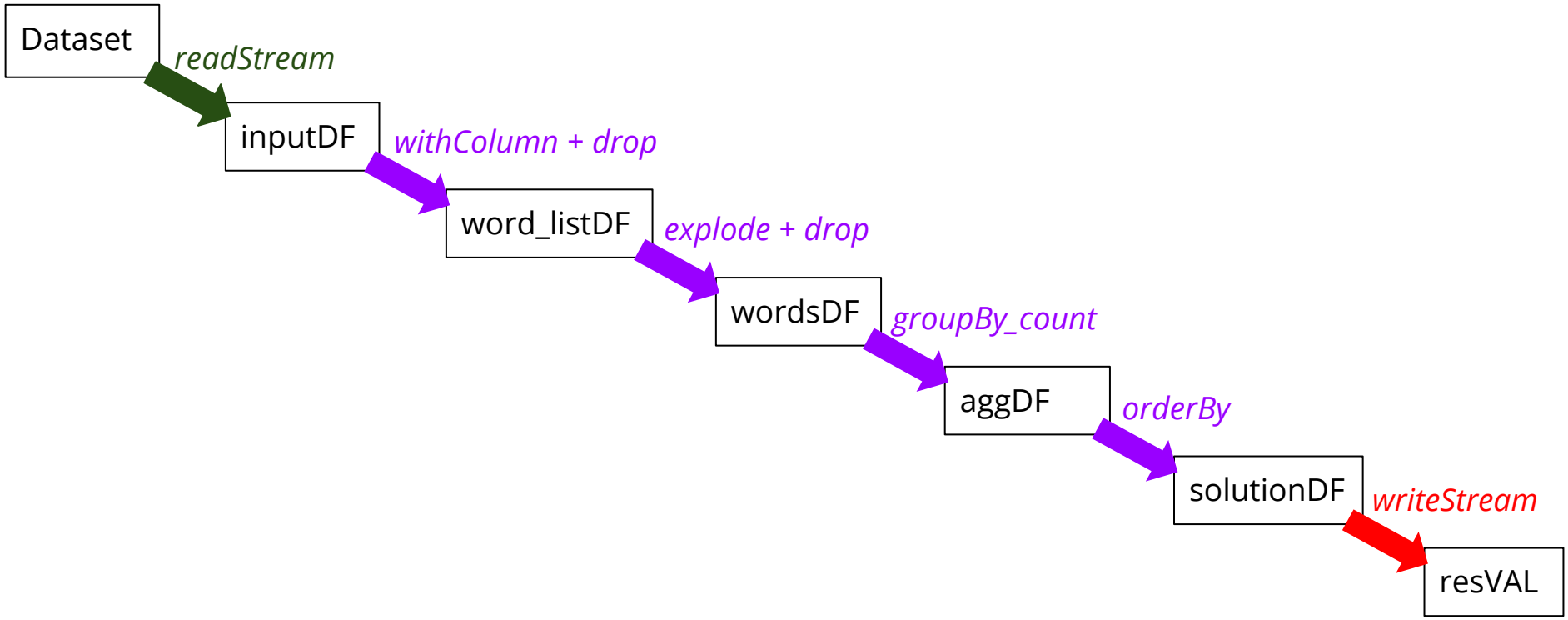
We use

sliding_duration = 1

window_duration = 2

Concept3: Append Mode with Windows

We have our program



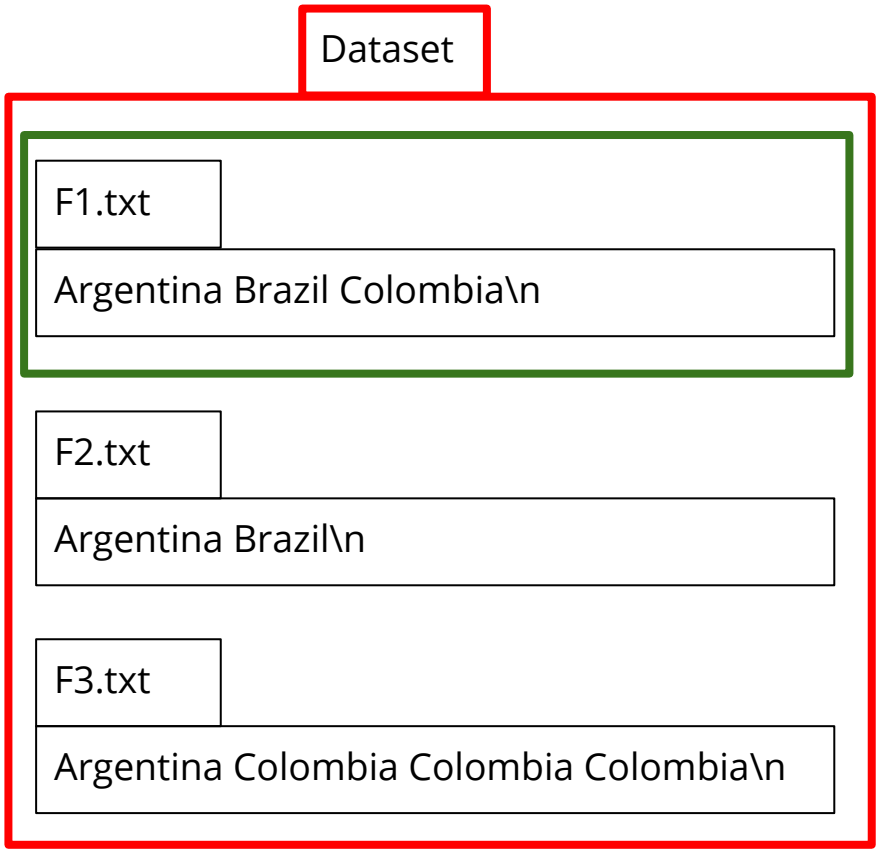
Concept3: Append Mode with Windows

We have our dataset

Concept3: Append Mode with Windows

What do we expect to obtain?

Concept3: Append Mode with Windows

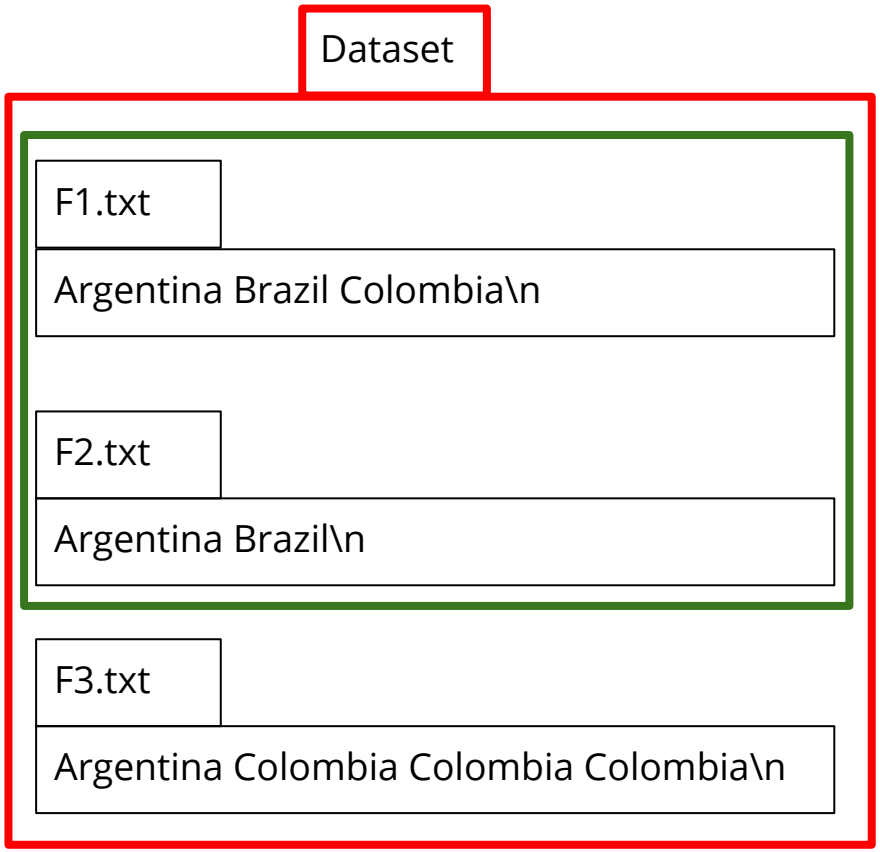


Concept3: Append Mode with Windows

Time Step 0:

```
+-------------+-------------+
|    word     | count       |
+-------------+-------------+
|Argentina    |     1       |
|  Brazil     |     1       |
| Colombia    |     1       |
+-------------+-------------+
```

Concept3: Append Mode with Windows

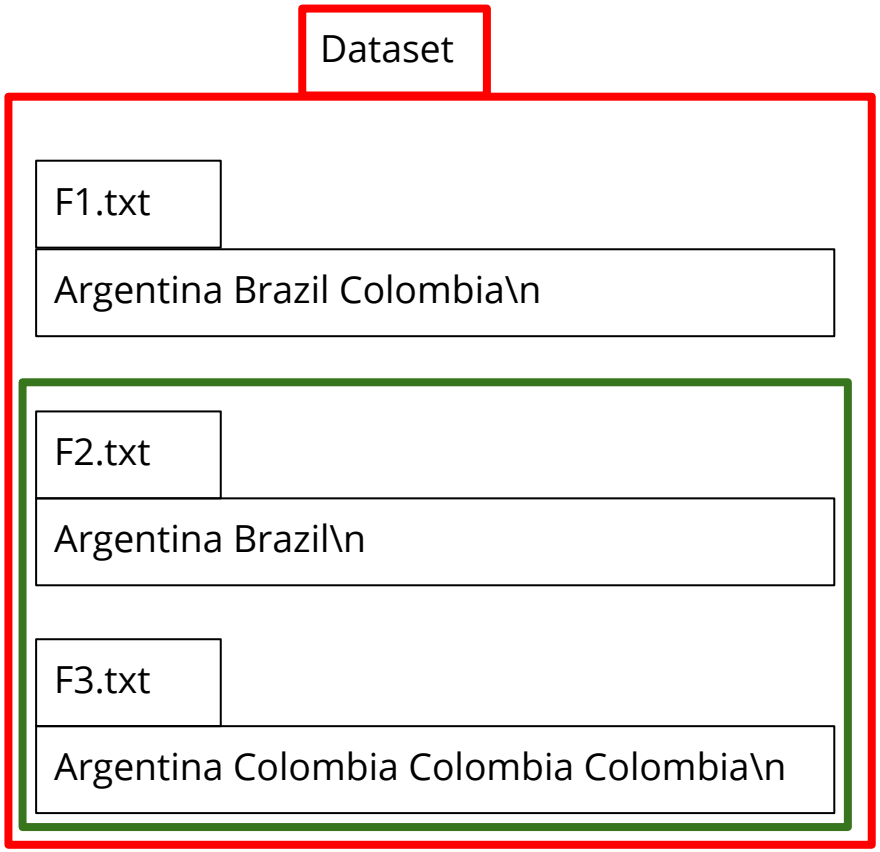


Concept3: Append Mode with Windows

Time Step 1:

```
+--------------+--------------+
|     word     | count        |
+--------------+--------------+
|Argentina     |      2       |
|   Brazil     |       2      |
|   Colombia   |       1      |
+--------------+--------------+
```

Concept3: Append Mode with Windows



Concept3: Append Mode with Windows

Time Step 2:

```
+-------------+-------------+
|    word     | count       |
+-------------+-------------+
|Colombia     |     3       |
| Argentina   |     2       |
|  Brazil     |     1       |
+-------------+-------------+
```

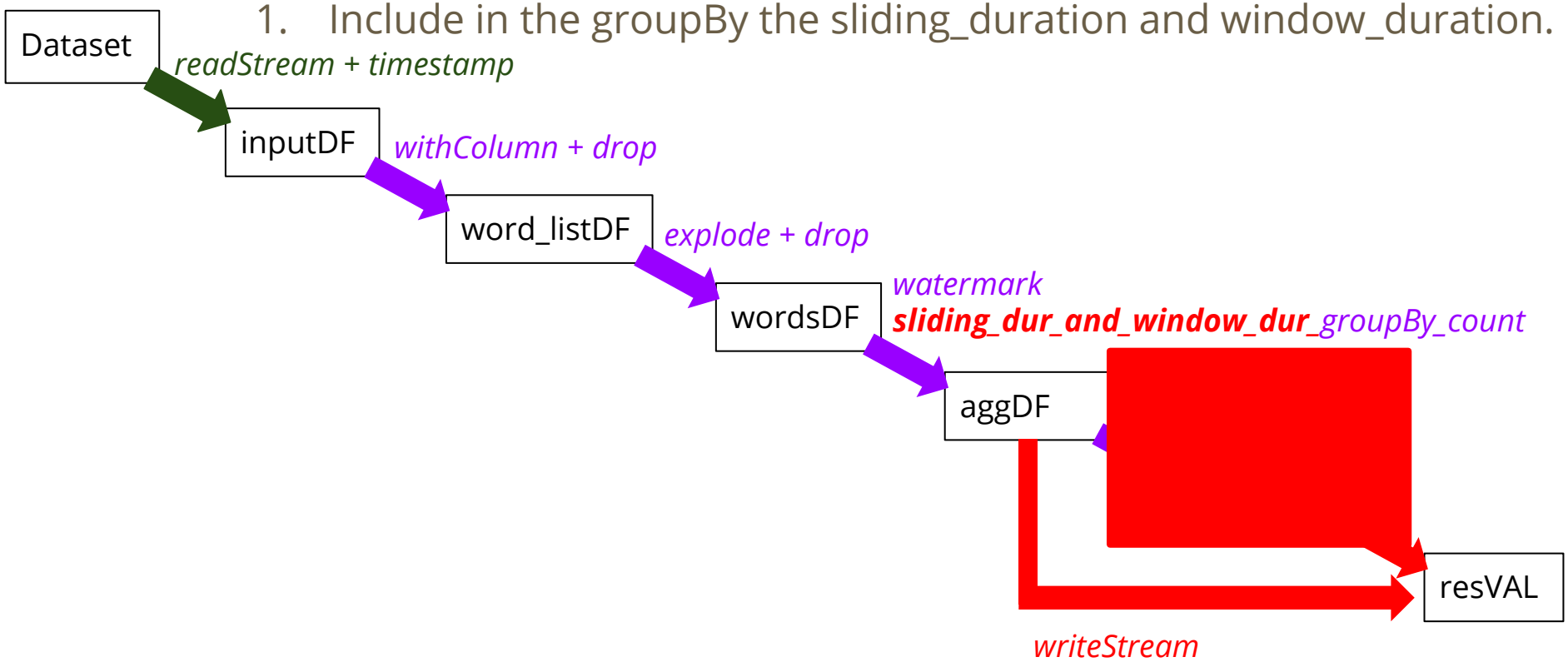
Concept3: Append Mode with Windows

What do we actually obtain?

Concept3: Append Mode with Windows

We modify our previous program **p05_ok_but_misses_time_interval.py** into **p06_ok_but_misses_time_interval.py** by:

1. Include in the groupBy the sliding_duration and window_duration.



Concept3: Append Mode with Windows

Program **p06_ok_but_misses_time_interval.py** executes without error.

Its results are presented in the file **p06_ok_but_misses_time_interval.txt**

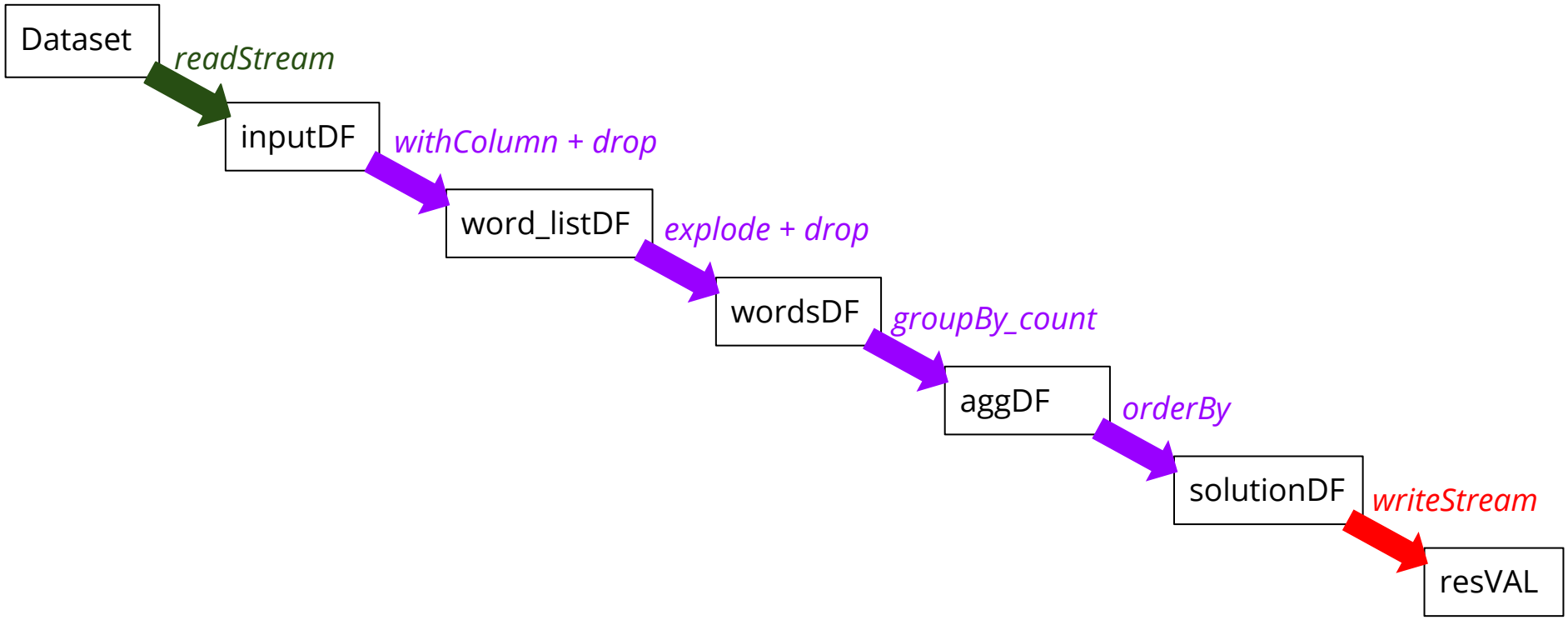
The results are as we originally expecting. However, for a reason we can't understand, the results for the last time step are never computed.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.
   a. A Program-Driven Example.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
   f. Console Sink vs. File Sink.

Concept4: Complete Mode

We have our program



Concept4: Complete Mode

We have our dataset

Concept4: Complete Mode

What do we expect to obtain?

Concept4: Complete Mode

Time Step 0:

word	count
Argentina	1
Brazil	1
Colombia	1

Concept4: Complete Mode

Time Step 1:

word	count
Argentina	2
Brazil	2
Colombia	1

Concept4: Complete Mode

Time Step 2:

word	count
Colombia	4
Argentina	3
Brazil	2

Concept4: Complete Mode

What do we actually obtain?

Concept4: Complete Mode

Program **p07_ok.py** executes without error.

|

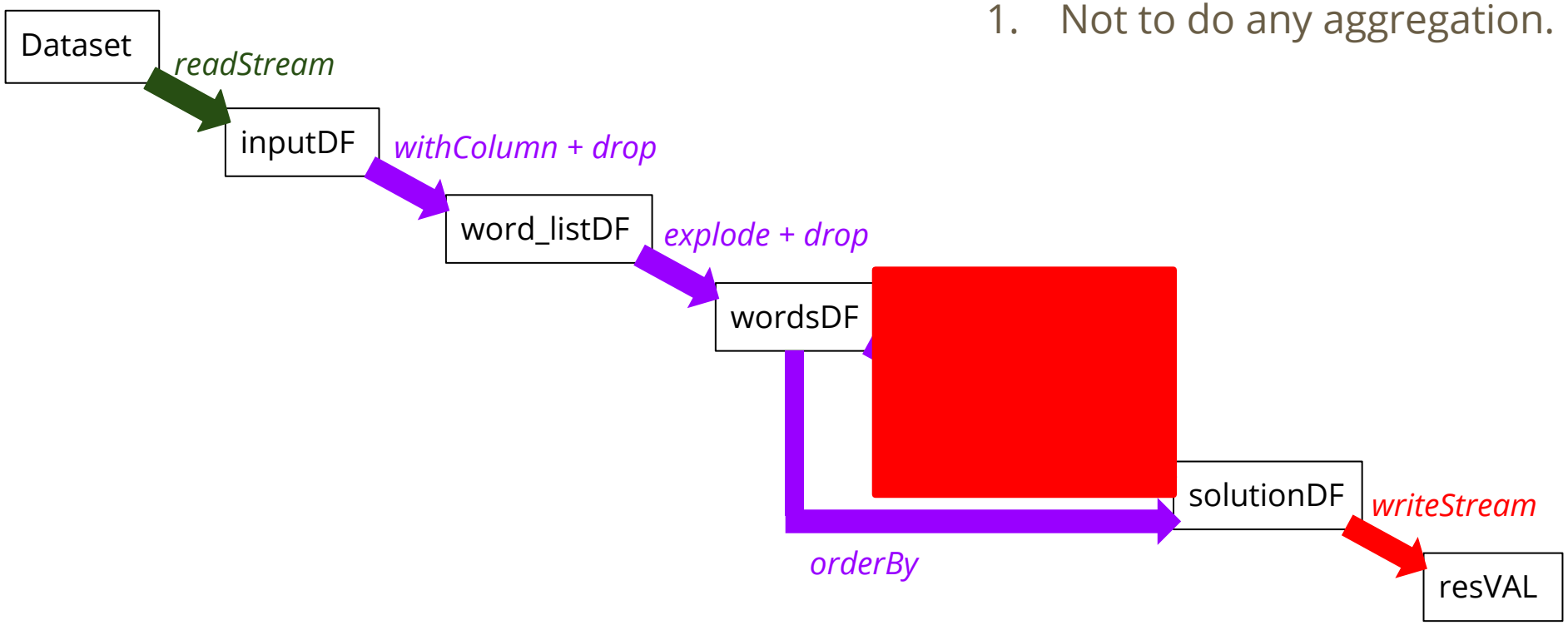
Its results are presented in the file **p07_ok.txt**

The results are as we originally expecting.

Concept4: Complete Mode

Just out of curiosity we modify our **program p07_ok.py** into **p08_ERROR_no_groupBy.py** by:

1. Not to do any aggregation.



Concept4: Complete Mode

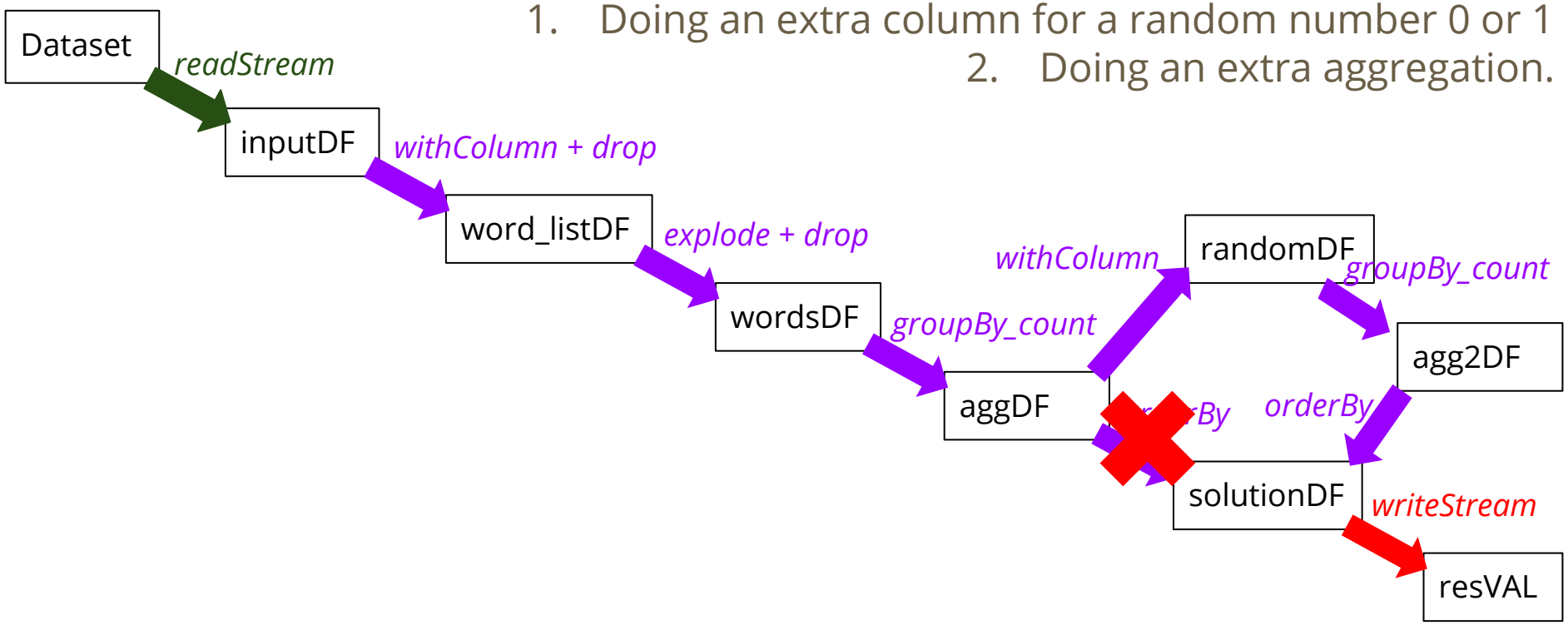
Program **p08_ERROR_no_groupBy.py** fails:

ERROR: Complete output mode not supported when there are no streaming aggregations on streaming DataFrames/Datasets.

Concept4: Complete Mode

Just out of curiosity we modify our **program p07_ok.py** into **p09_ERROR_two_groupBy.py** by:

1. Doing an extra column for a random number 0 or 1
2. Doing an extra aggregation.



Concept4: Complete Mode

Program **p09_ERROR_two_groupBy.py** fails:

ERROR: Multiple streaming aggregations are not supported with streaming DataFrames/Datasets.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.
   a. A Program-Driven Example.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
   f. Console Sink vs. File Sink.

Concept5: Complete Mode with Windows

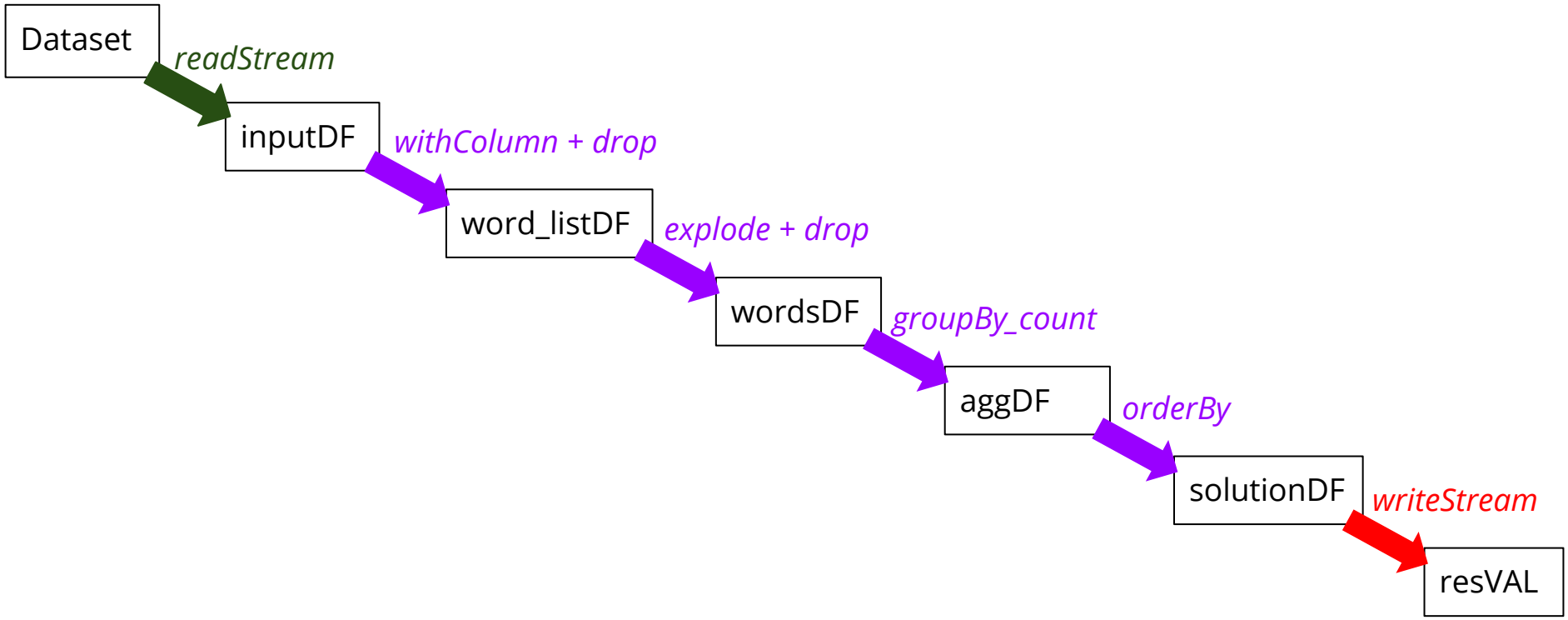
We use

sliding_duration = 1

window_duration = 2

Concept5: Complete Mode with Windows

We have our program



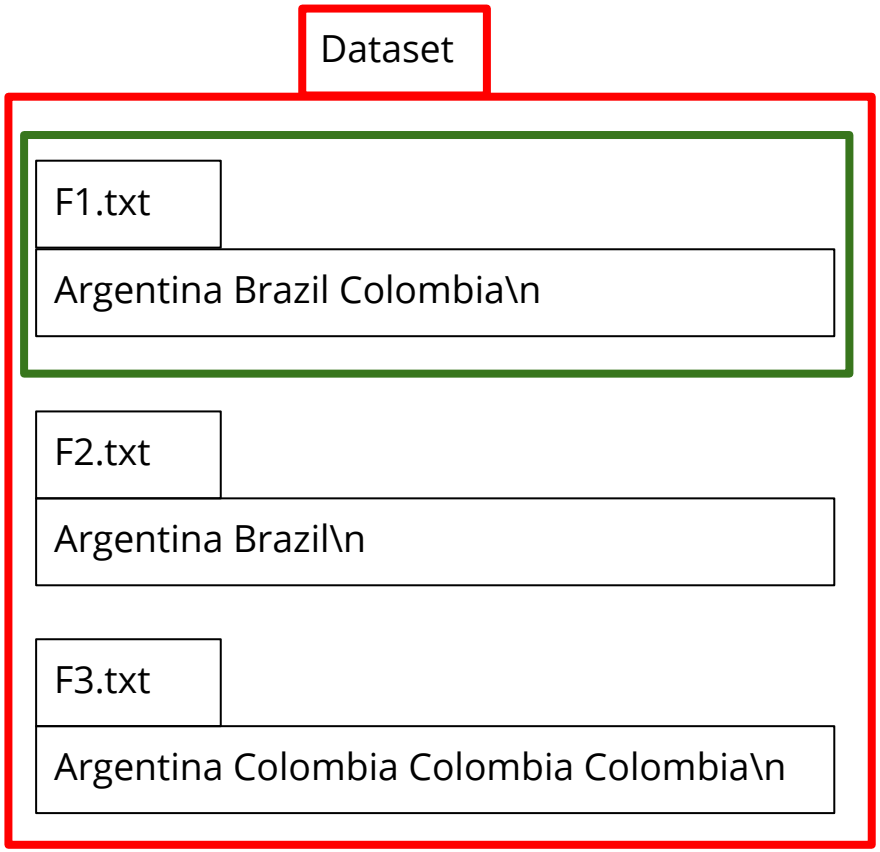
Concept5: Complete Mode with Windows

We have our dataset

Concept5: Complete Mode with Windows

What do we expect to obtain?

Concept5: Complete Mode with Windows

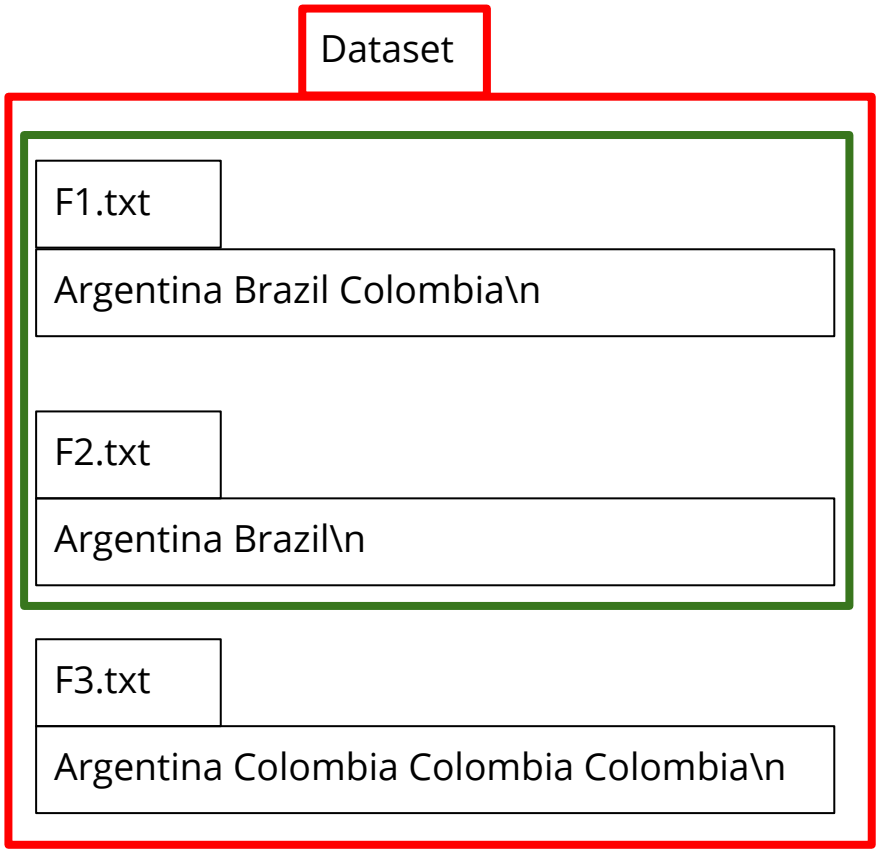


Concept5: Complete Mode with Windows

Time Step 0:

word	count
Argentina	1
Brazil	1
Colombia	1

Concept5: Complete Mode with Windows

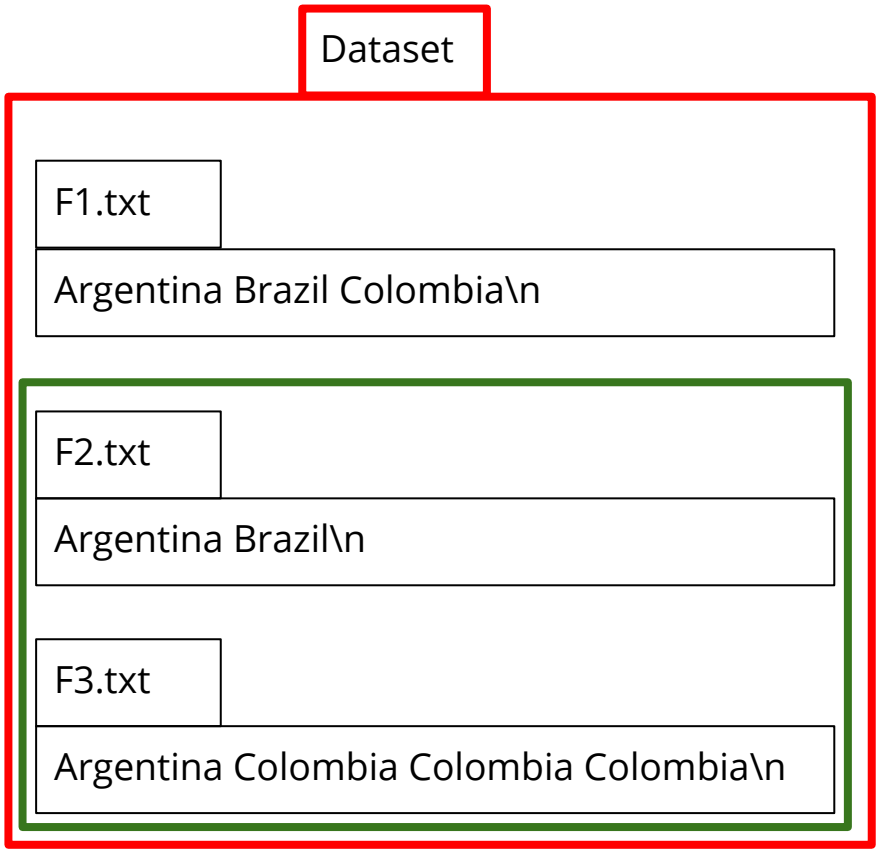


Concept5: Complete Mode with Windows

Time Step 1:

```
+--------------+--------------+
|     word     | count        |
+--------------+--------------+
|Argentina     |      3       |
|   Brazil     |       3      |
|   Colombia   |       2      |
+--------------+--------------+
```

Concept5: Complete Mode with Windows



Concept5: Complete Mode with Windows

Time Step 2:

word	count
Argentina	5
Colombia	5
Brazil	4

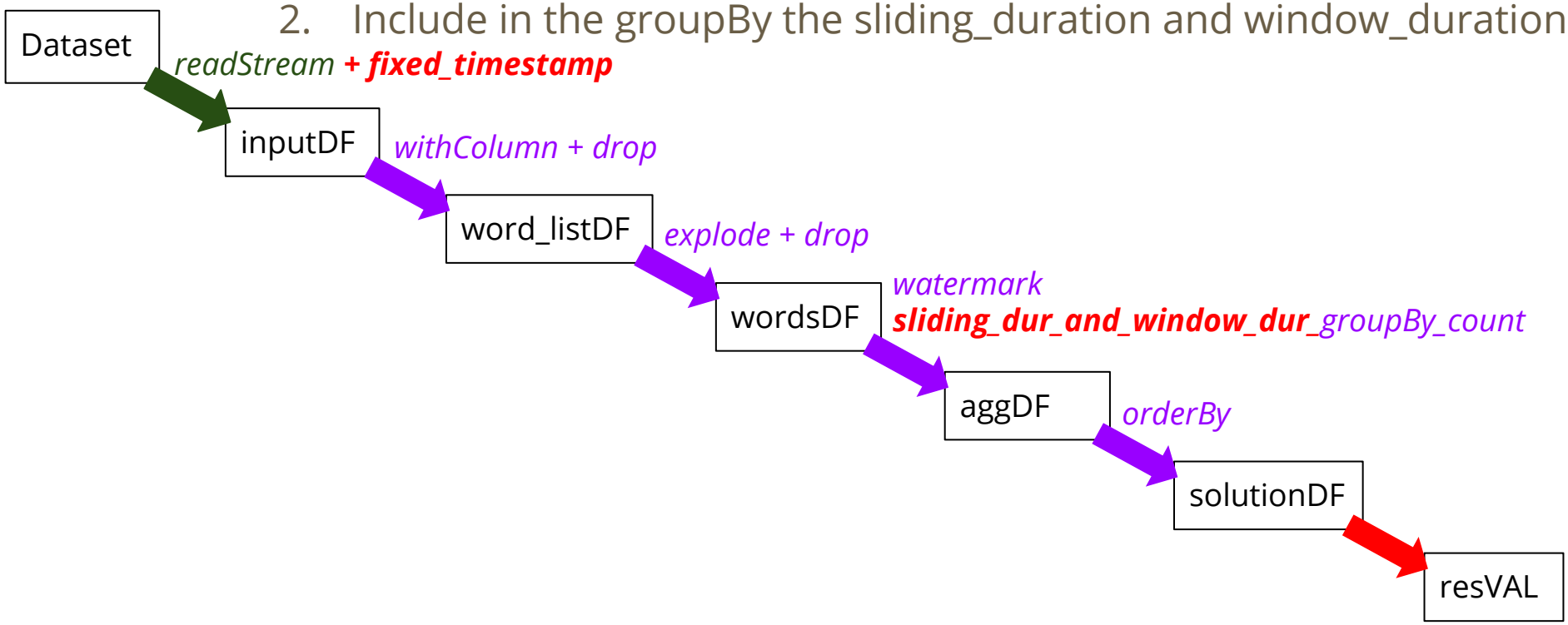
Concept5: Complete Mode with Windows

What do we actually obtain?

Concept5: Complete Mode with Windows

We modify our previous program **p07_ok.py** into **p10_ok_different_results.py** by:

1. Create a column with a fixed timestamp
2. Include in the groupBy the sliding_duration and window_duration.



Concept5: Complete Mode with Windows

Program **p10_ok_different_results.py** executes without error.

Its results are presented in the file **p10_ok_different_results.txt**

The results are not as we originally expecting.
It seems to be computing the complete mode twice, instead of aggregating the results of previous windows to the ones of current window.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. Practising with the Concepts.
   a. A Program-Driven Example.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
   f. Console Sink vs. File Sink.

Console Sink vs. File Sink

We have seen all the concepts
by using Console as our output sink.

Console Sink vs. File Sink

We have seen all the concepts
by using Console as our output sink.

Let's re-assess them now
by using File (ResultDir) as our output sink.

Console Sink vs. File Sink

Although the expected and obtained results should be the same as for Console sink, we see there are extra considerations to take.

Console Sink vs. File Sink

In particular, the way
Spark Structured Streaming
presents the results when writing to a
File sink is...

Console Sink vs. File Sink

In particular, the way
Spark Structured Streaming
presents the results when writing to a
File sink is...

complicated :(

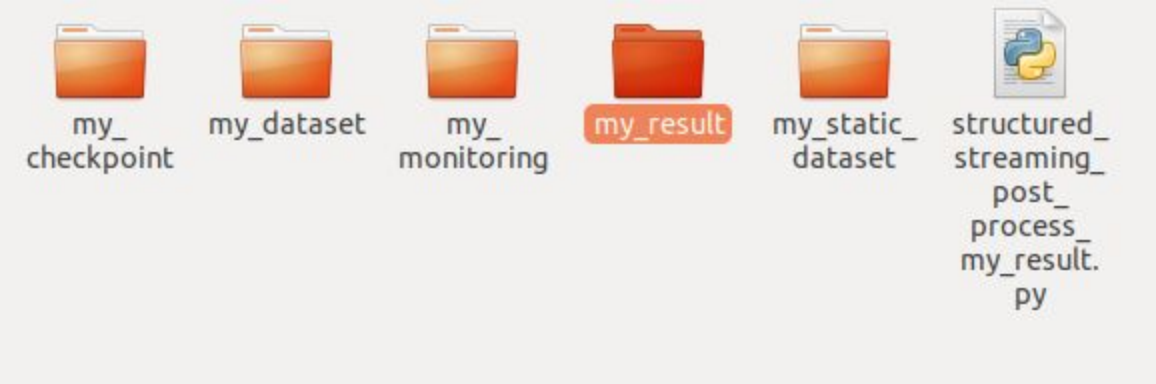
Console Sink vs. File Sink

Let's take a look at **my_result** directory after having run a program

```
Run:  p11_ok_but_misses_time_interval
/usr/bin/python3.6 /home/nacho/CIT/Tools/MyCode/Spark/
2019-12-05 11:08:48 WARN  Utils:66 - Your hostname, cc
2019-12-05 11:08:48 WARN  Utils:66 - Set SPARK_LOCAL_I
2019-12-05 11:08:49 WARN  NativeCodeLoader:62 - Unable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).



2019-12-05 11:09:14 WARN  ProcessingTimeExecutor:66 -

Process finished with exit code 0
```

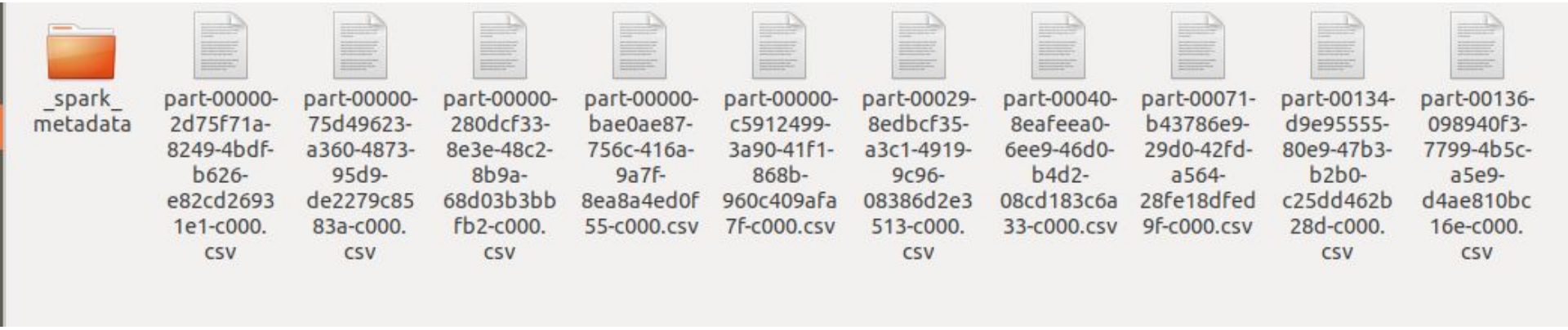
Console Sink vs. File Sink

Let's take a look at **my_result** directory after having run a program



Console Sink vs. File Sink

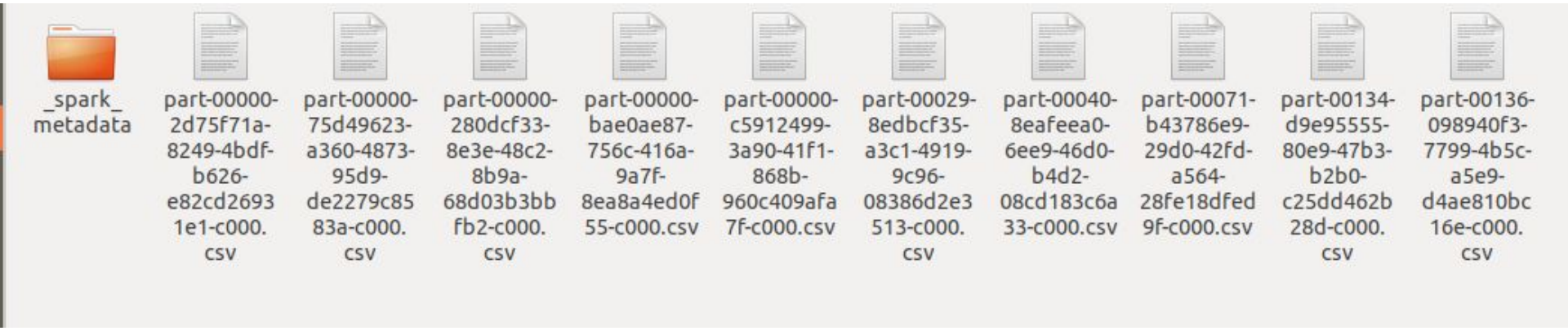
As we can see, the results are far from being presented in a human-legible way.



Console Sink vs. File Sink

As we can see, the results are far from being presented in a human-legible way.

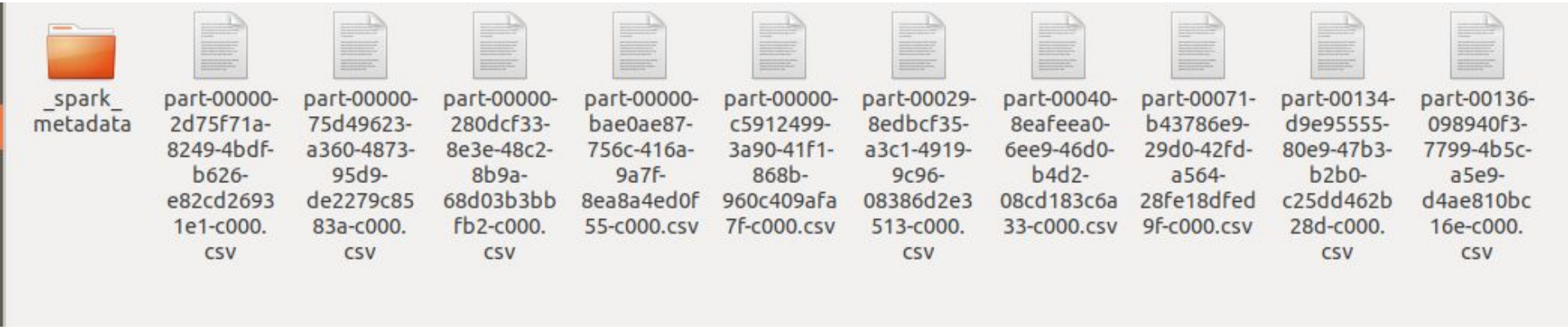
- We have a bunch of files, that are the results of some partitions, but their name do not have any *human-legible* relationship with the time step in which they were run.



Console Sink vs. File Sink

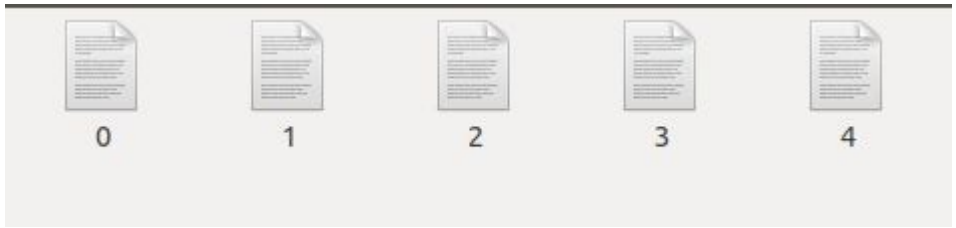
As we can see, the results are far from being presented in a human-legible way.

- We have a bunch of files, that are the results of some partitions, but their name do not have any *human-legible* relationship with the time step in which they were run.
- We also have an extra folder **_spark_metadata**. So let's start by exploring it.



Console Sink vs. File Sink

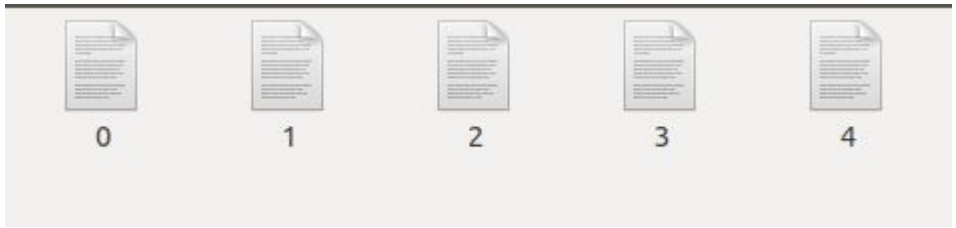
As we can see, **_spark_metadata** has some files [0, 1, 2, ...]
that we *guess* are related to the time step they were computing results for.



Console Sink vs. File Sink

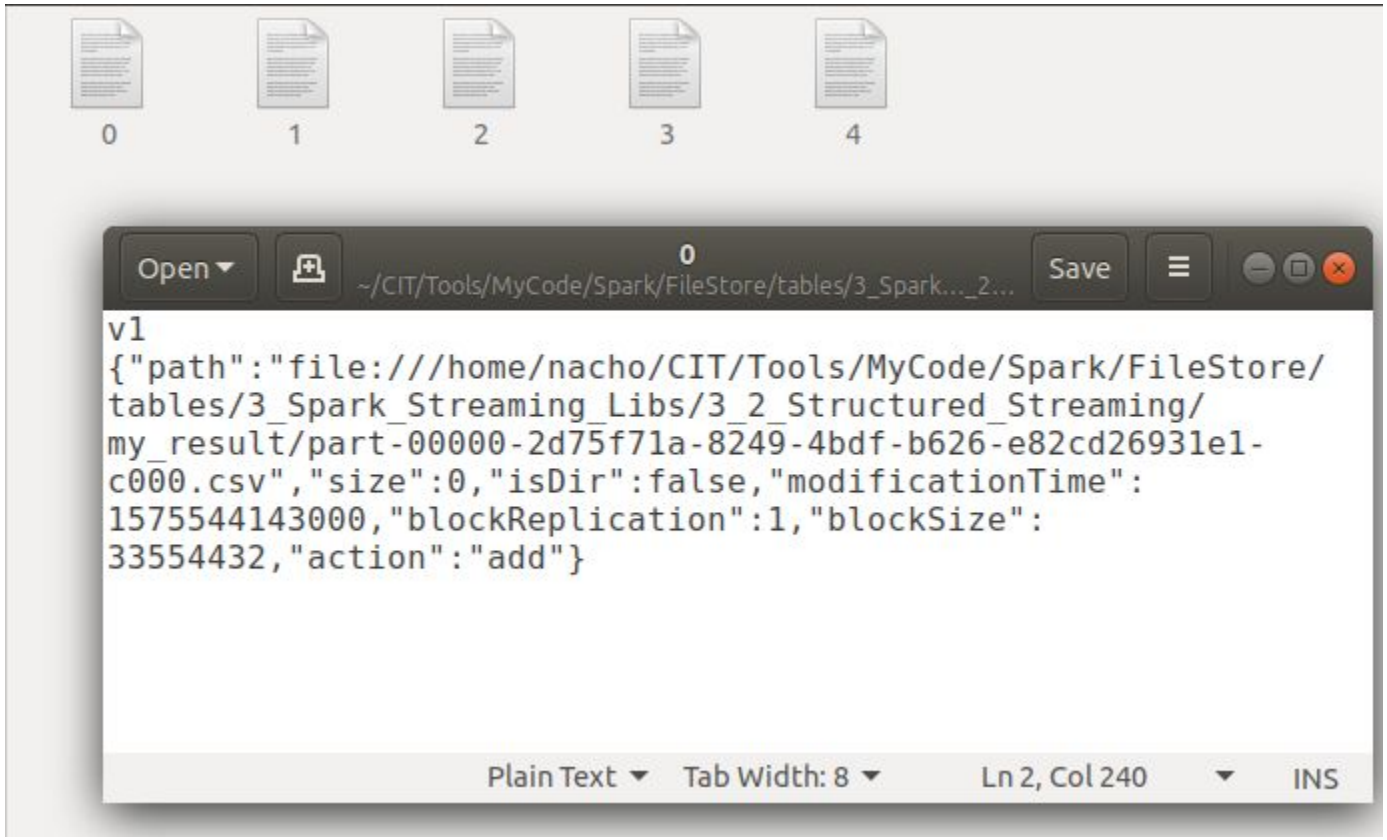
As we can see, **_spark_metadata** has some files [0, 1, 2, ...]
that we *guess* are related to the time step they were computing results for.

Let's explore the first of these files: **0**



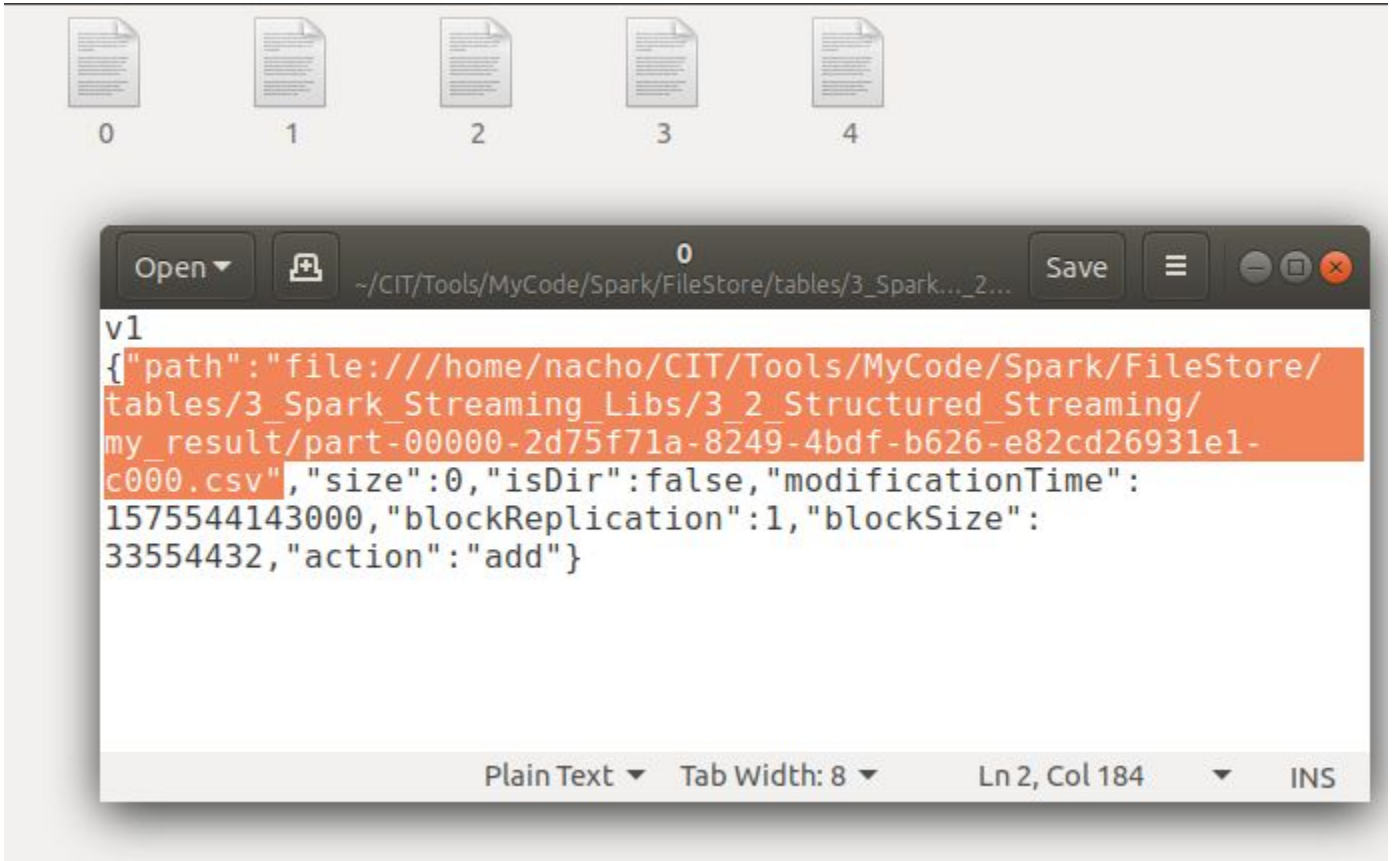
Console Sink vs. File Sink

Ok, things are not getting much clearer.
But let's try to *guess* what does this content mean.



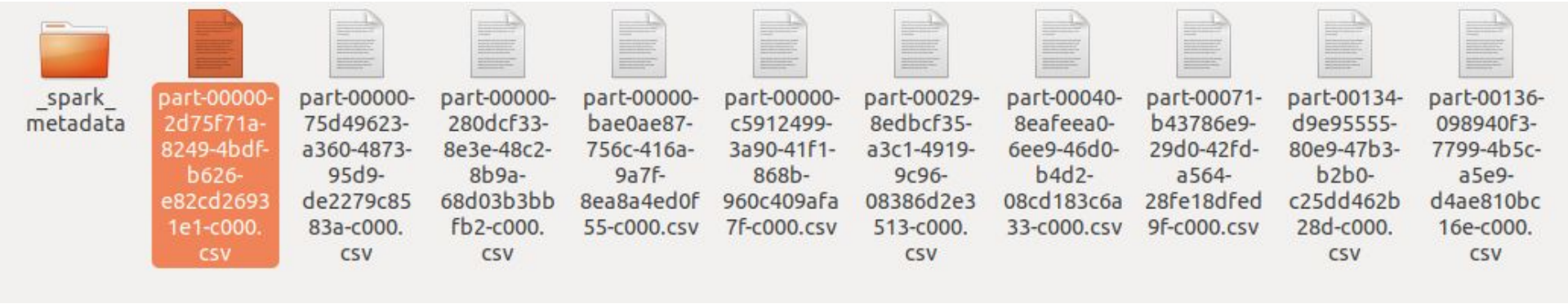
Console Sink vs. File Sink

There is a key **path**, that relates to one of the file **part-00000-2d75f71a-8249-4bdf-b626-e82cd26931e1-c000.csv** of my_result directory.



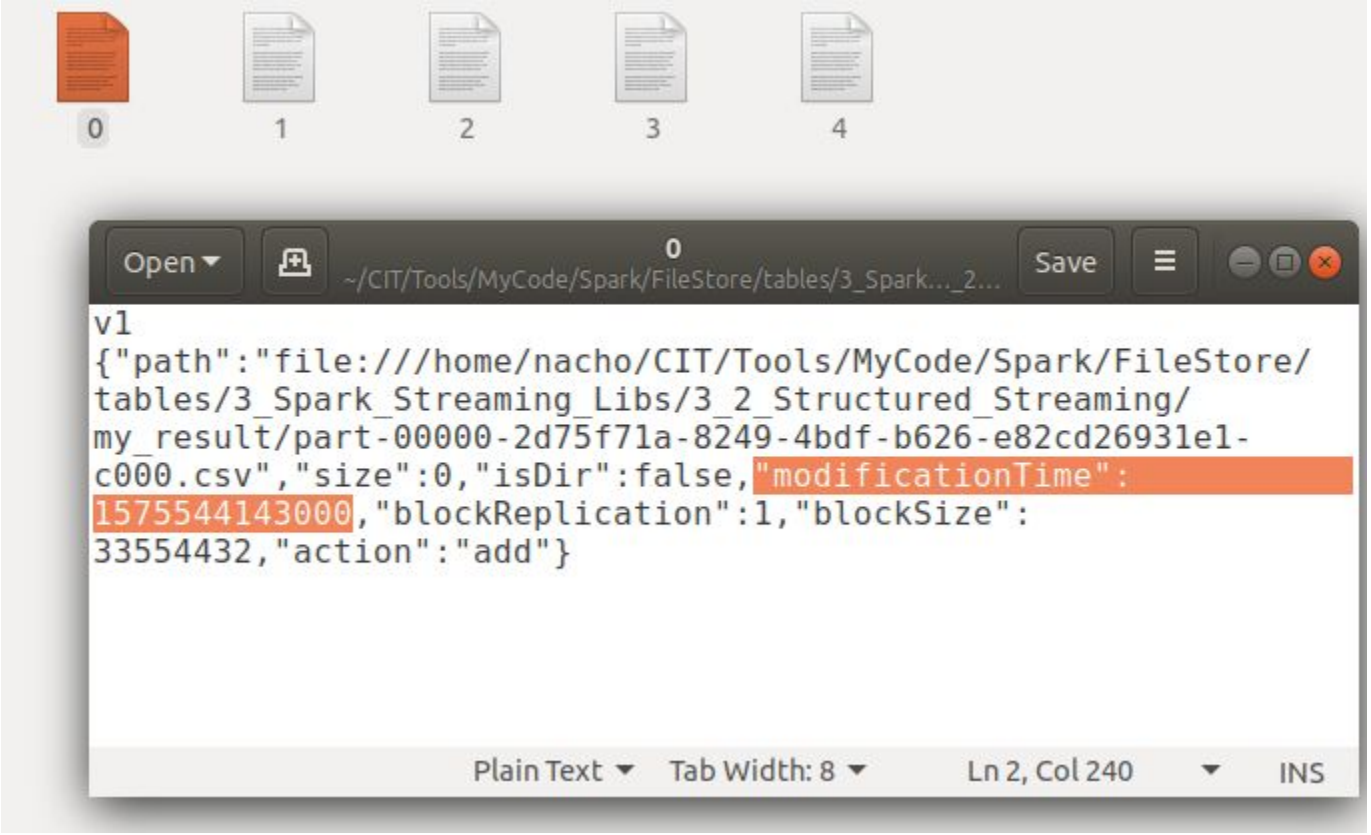
Console Sink vs. File Sink

There is a key **path**, that relates to one of the file
part-00000-2d75f71a-8249-4bdf-b626-e82cd26931e1-c000.csv
of my_result directory.



Console Sink vs. File Sink

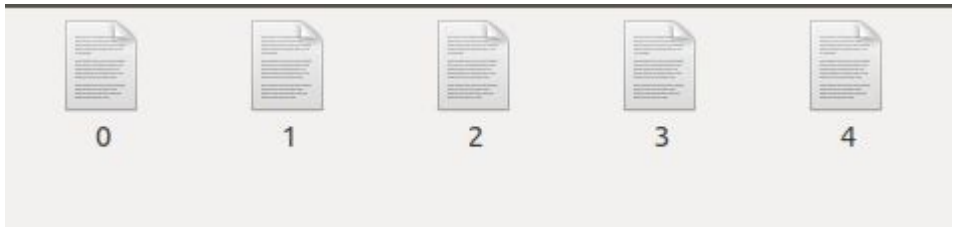
There is a key **modificationTime**,
that we *guess* it relates to the creation timestamp of the file
part-00000-2d75f71a-8249-4bdf-b626-e82cd26931e1-c000.csv



Console Sink vs. File Sink

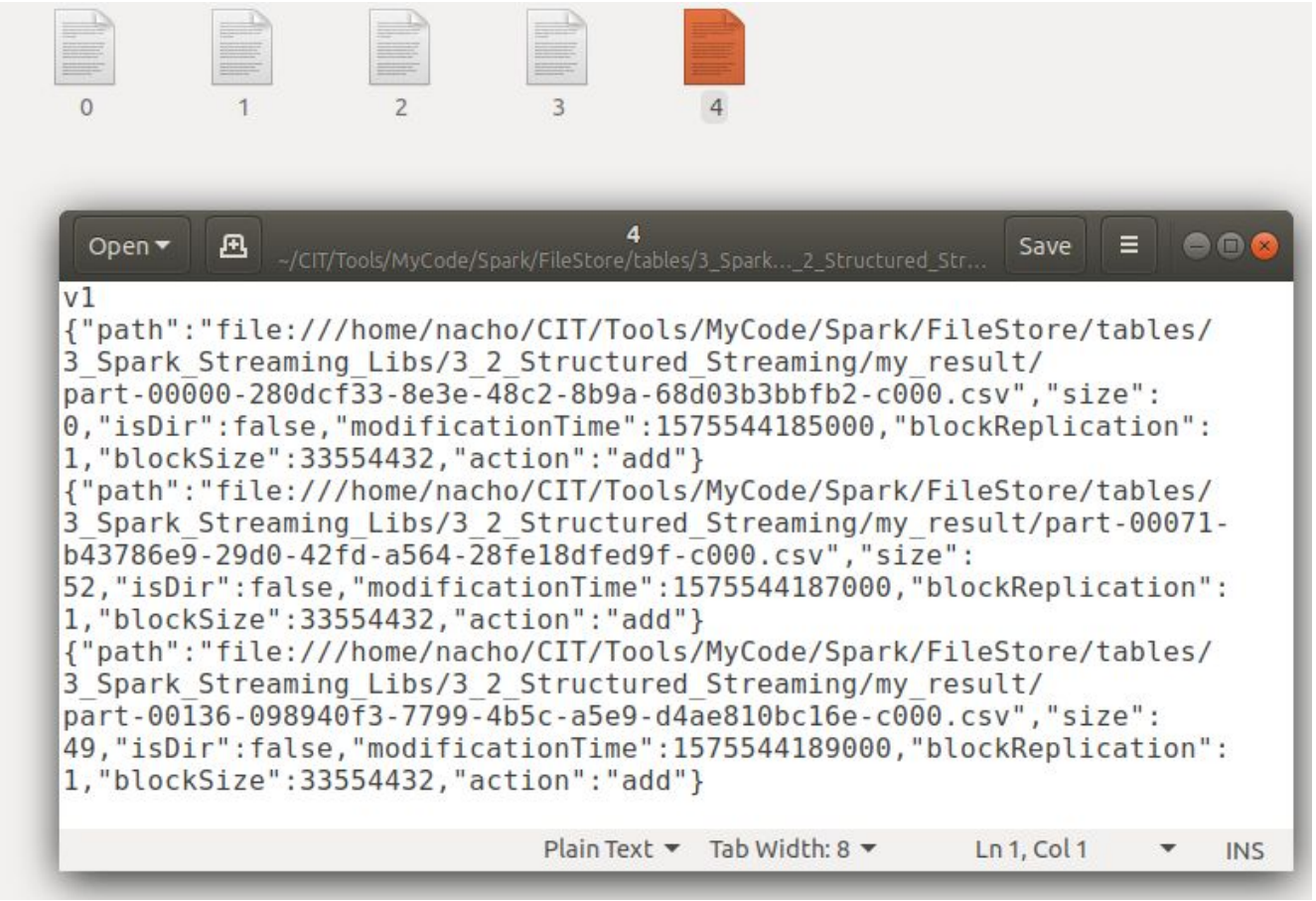
As we can see, **_spark_metadata** has some files [0, 1, 2, ...]
that we *guess* are related to the time step they were computing results for.

Let's explore the last of these files: **4**



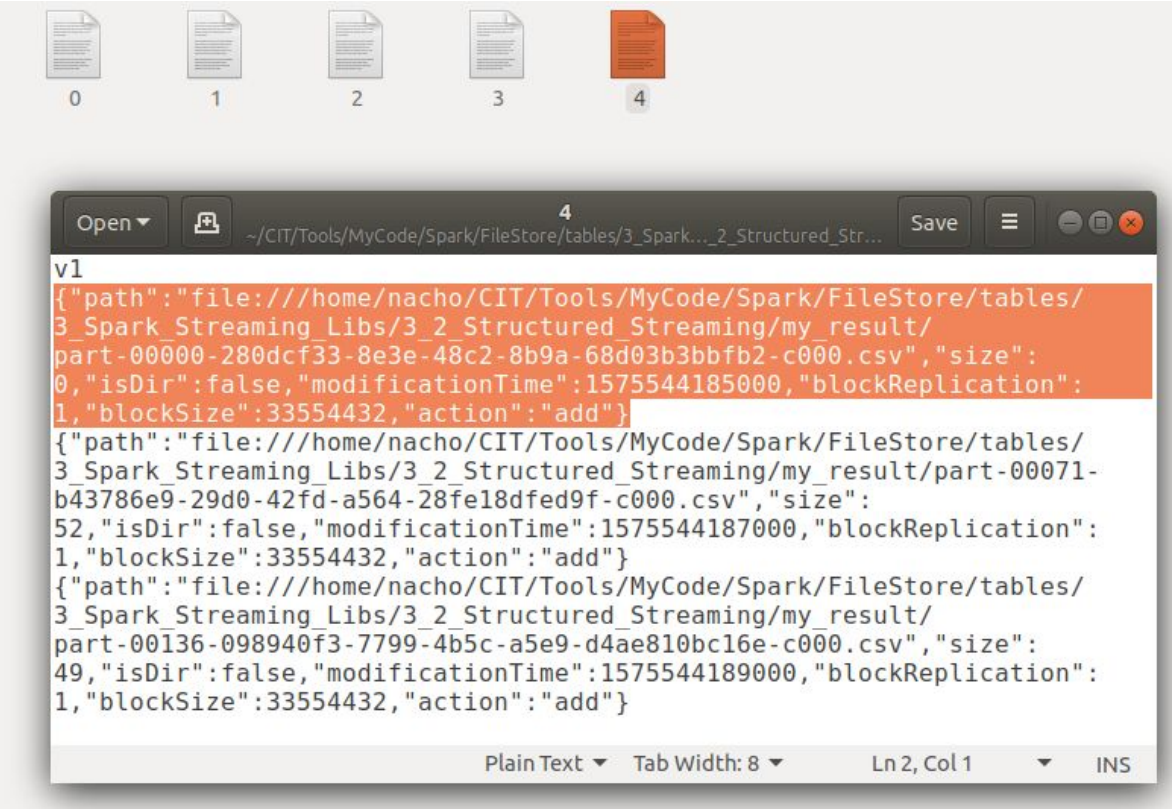
Console Sink vs. File Sink

As we can see, **4** relates to <u>three files</u> from my_result directory.



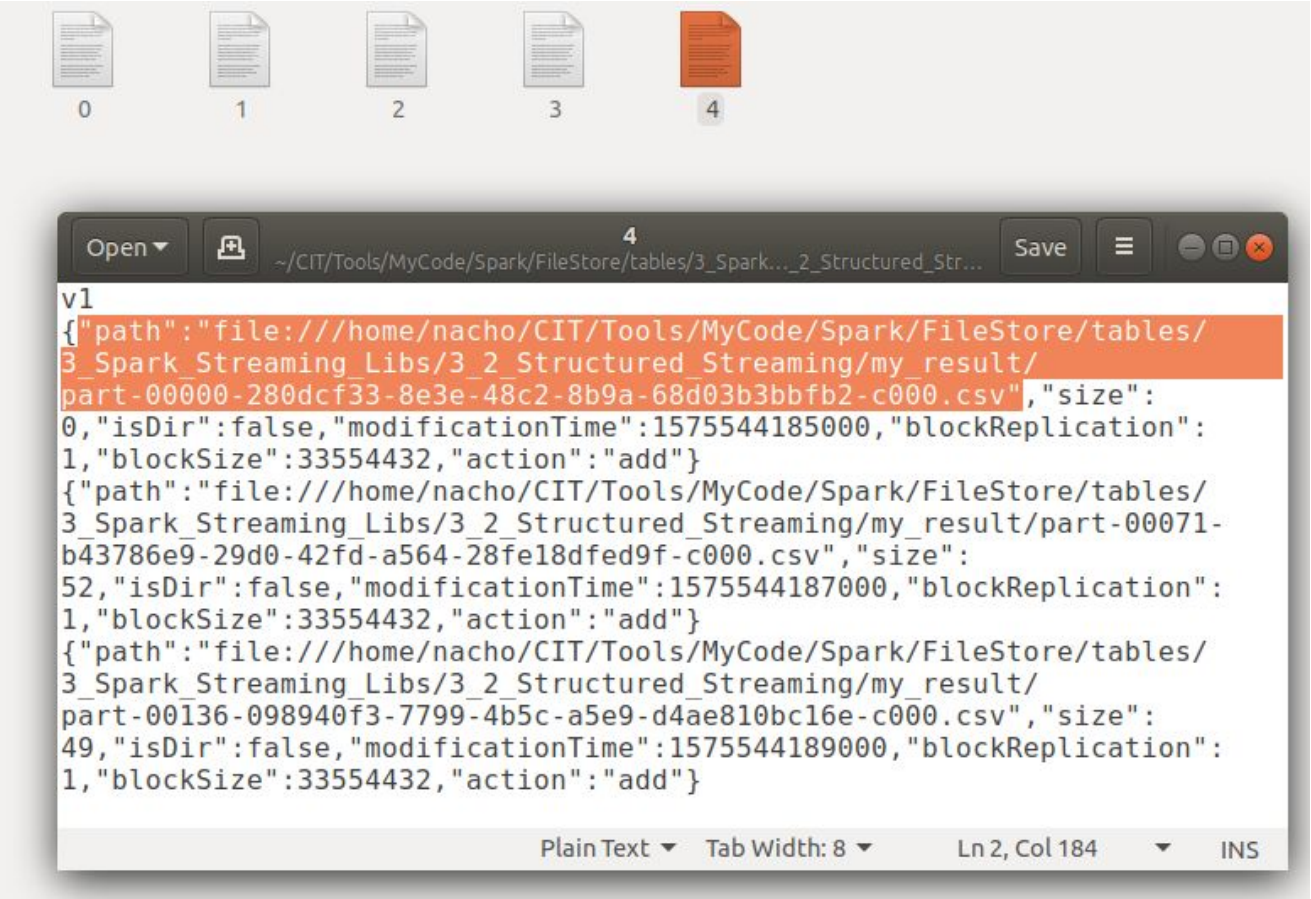
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
First block:



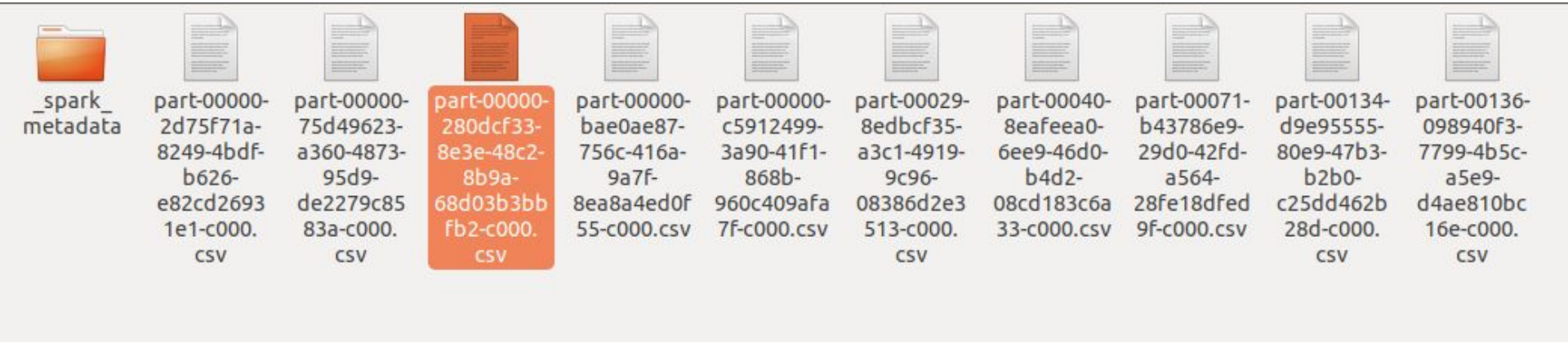
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
First block:



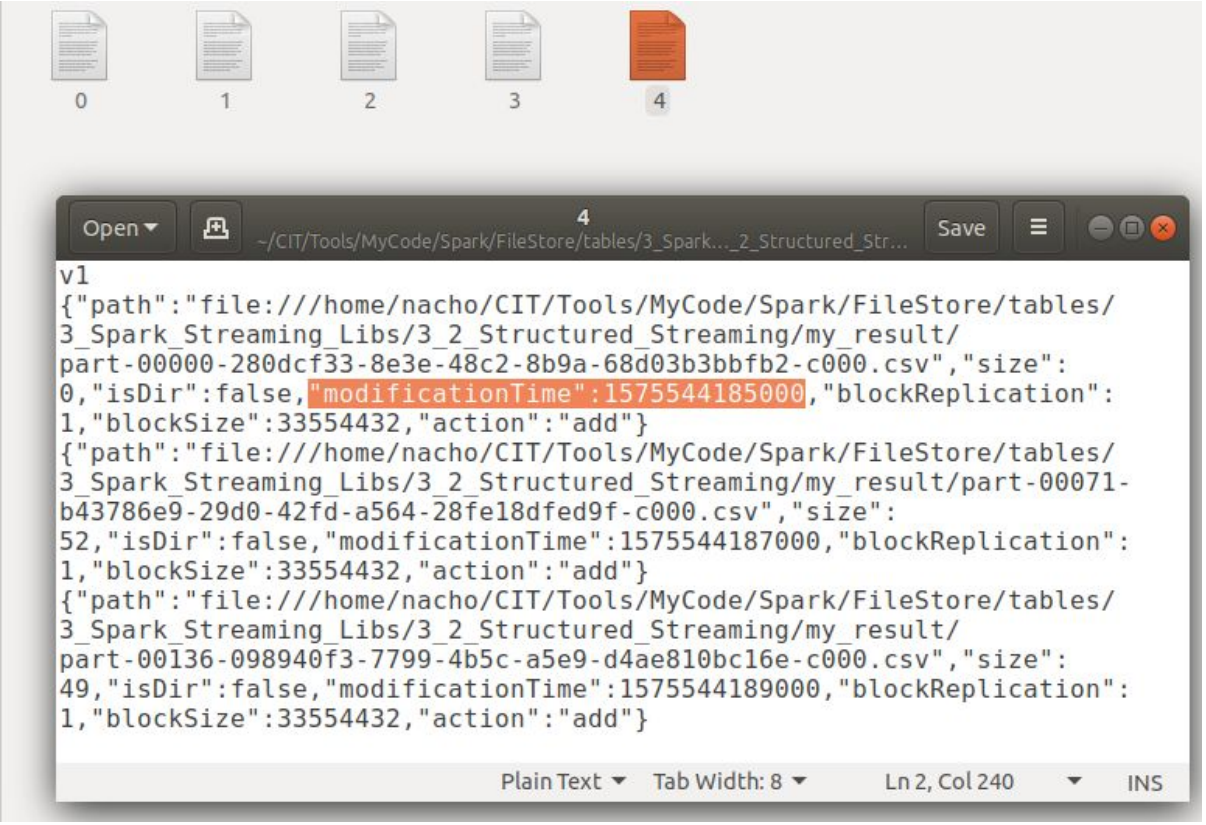
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
First block:



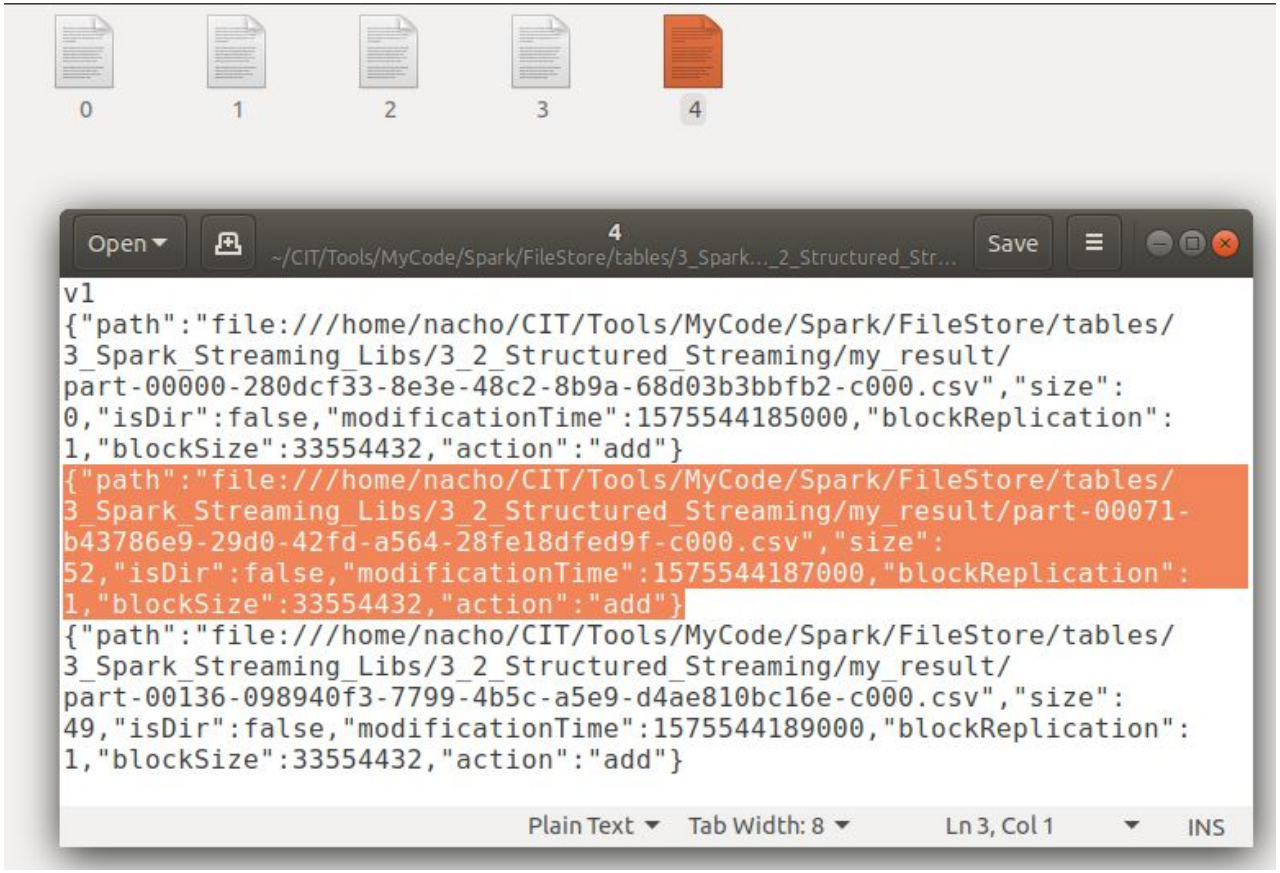
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
First block:



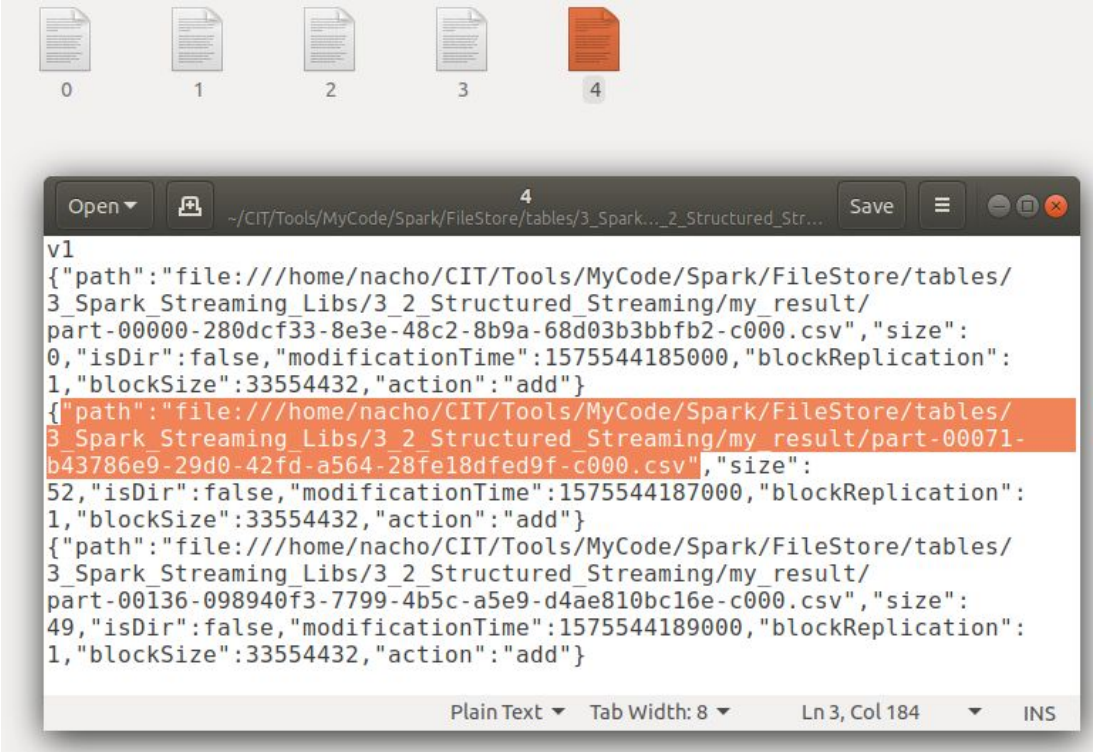
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
Second block:



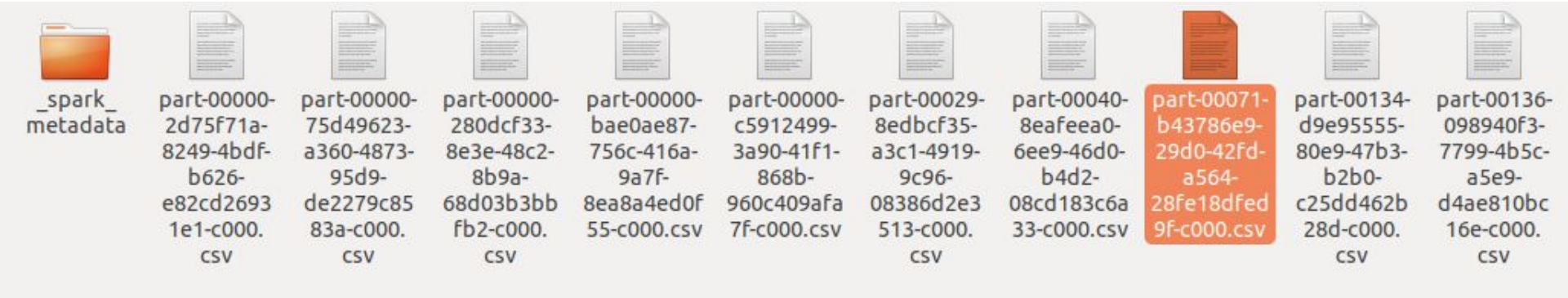
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
Second block:



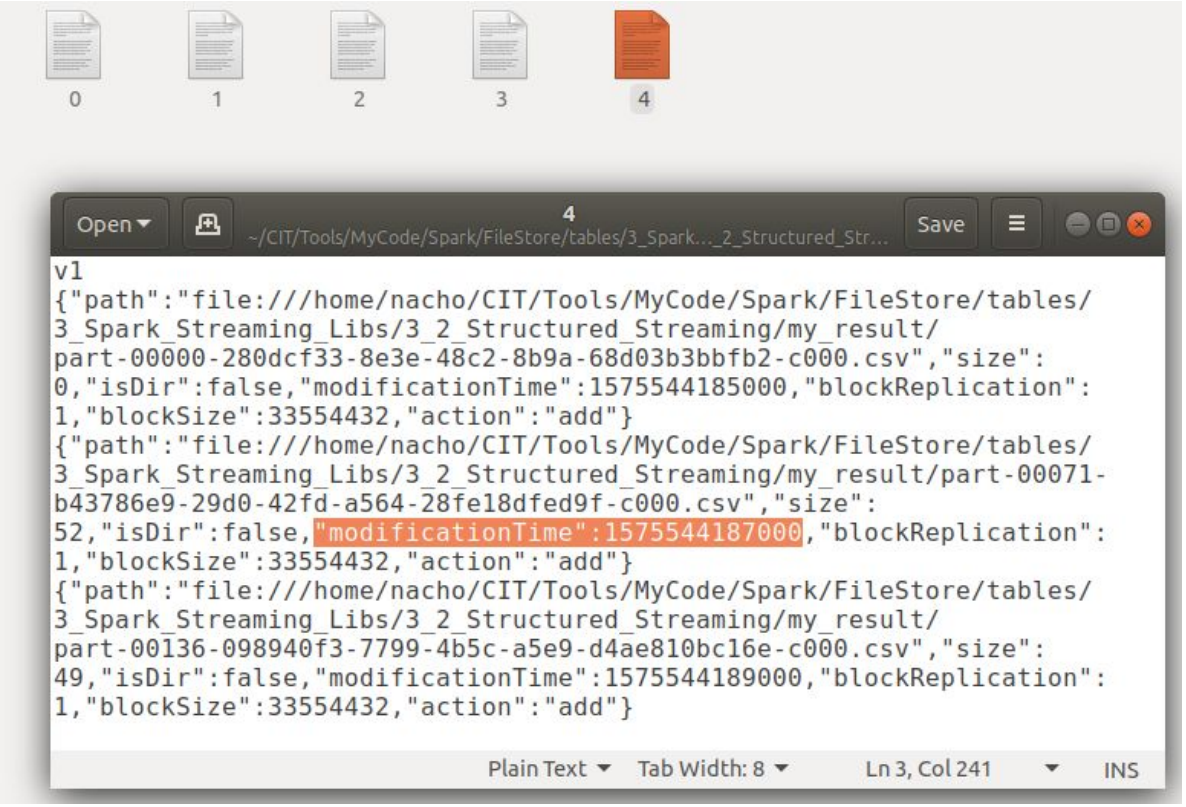
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
Second block:



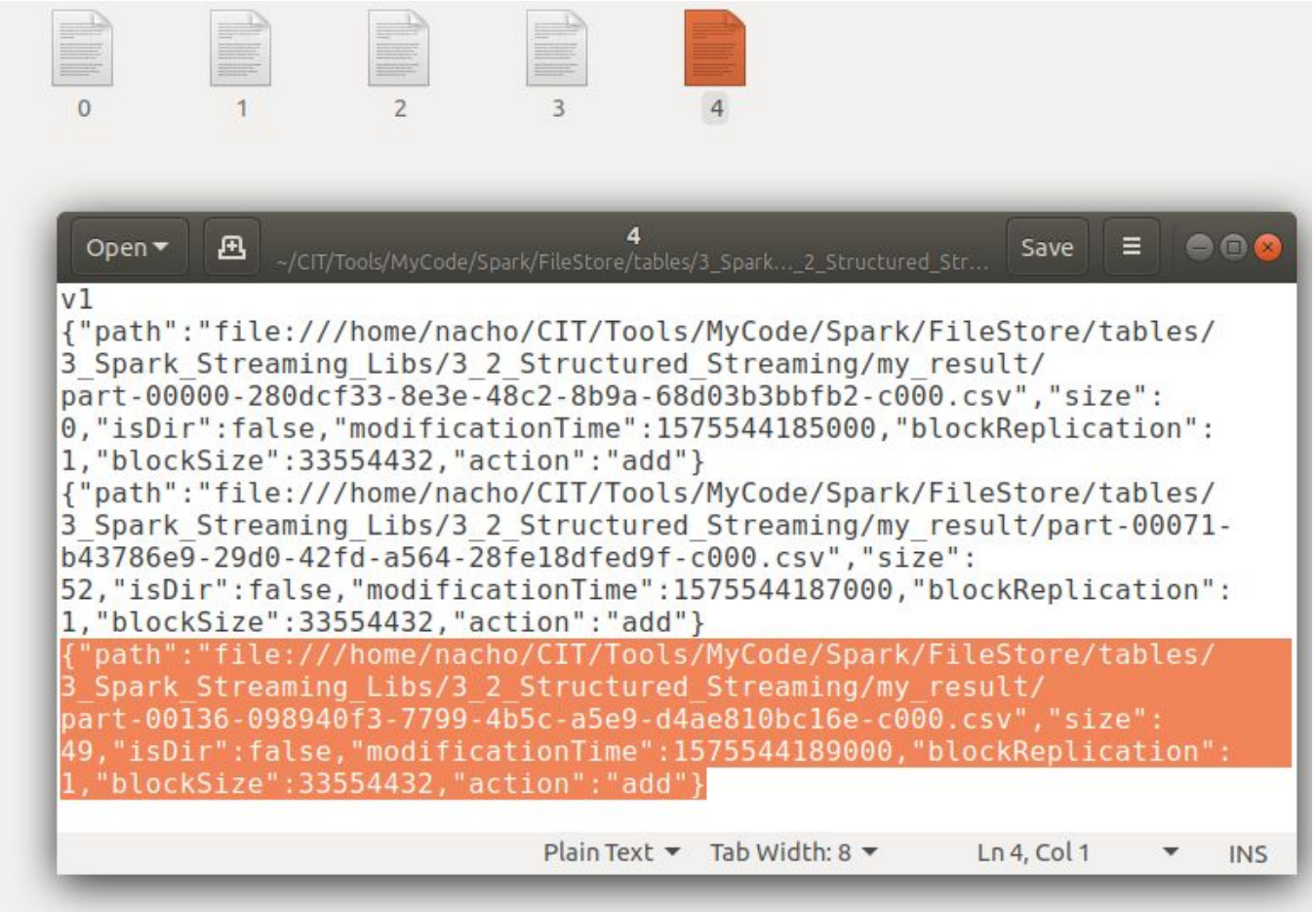
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
Second block:



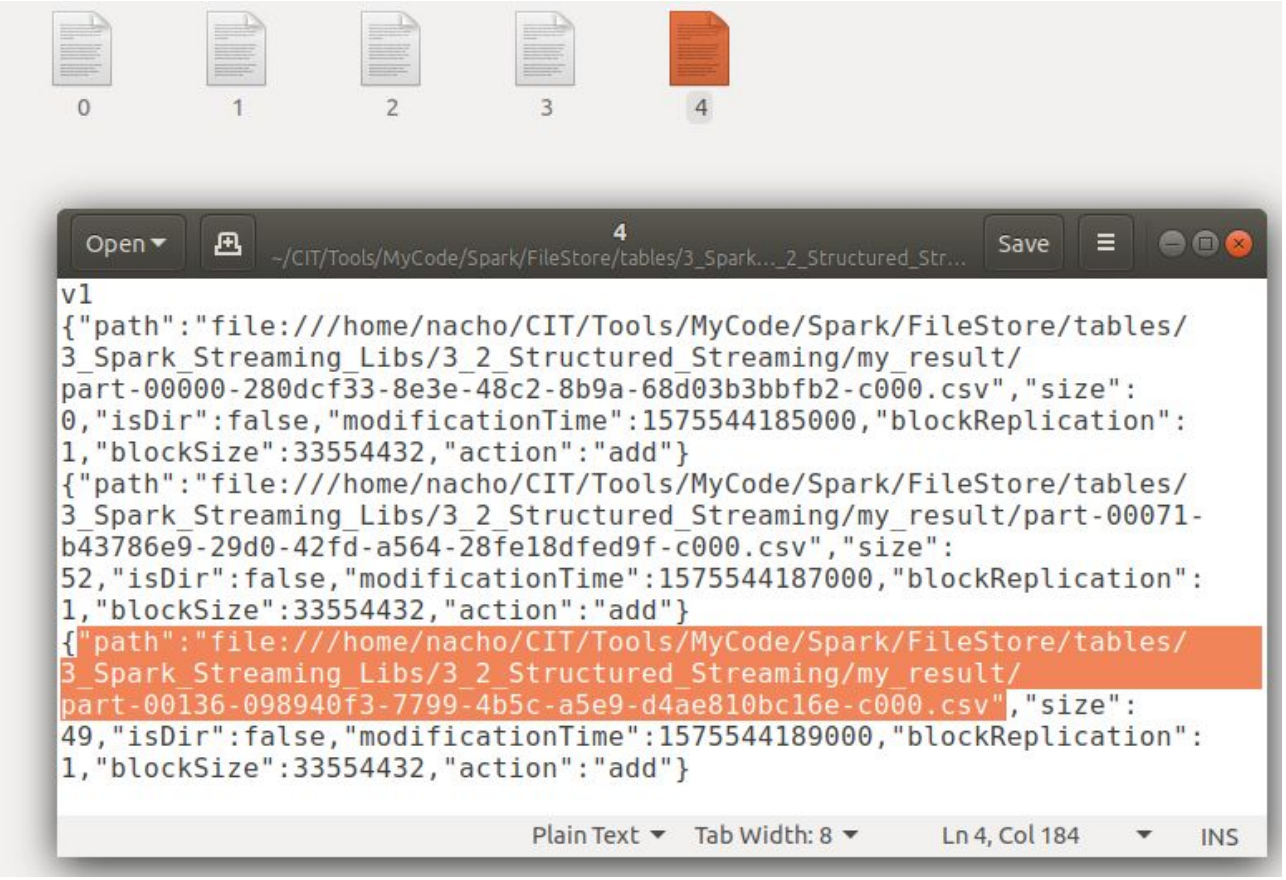
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
Third block:



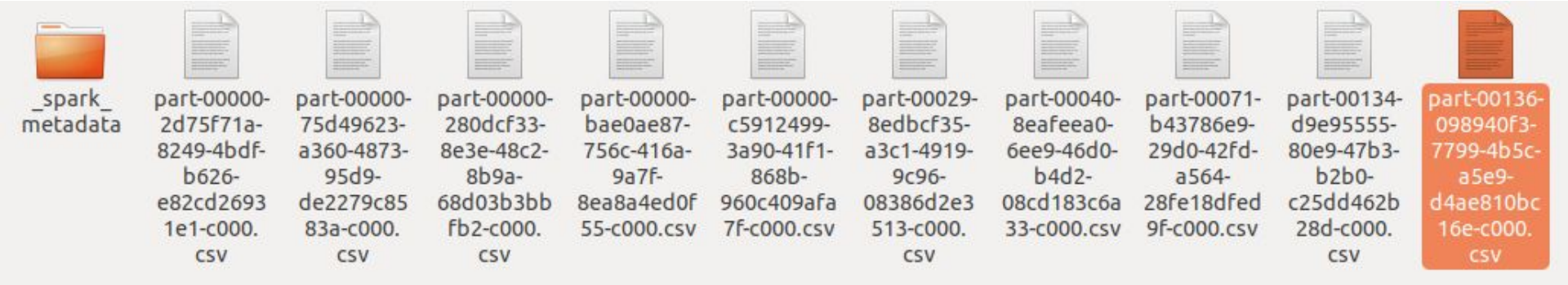
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
Third block:



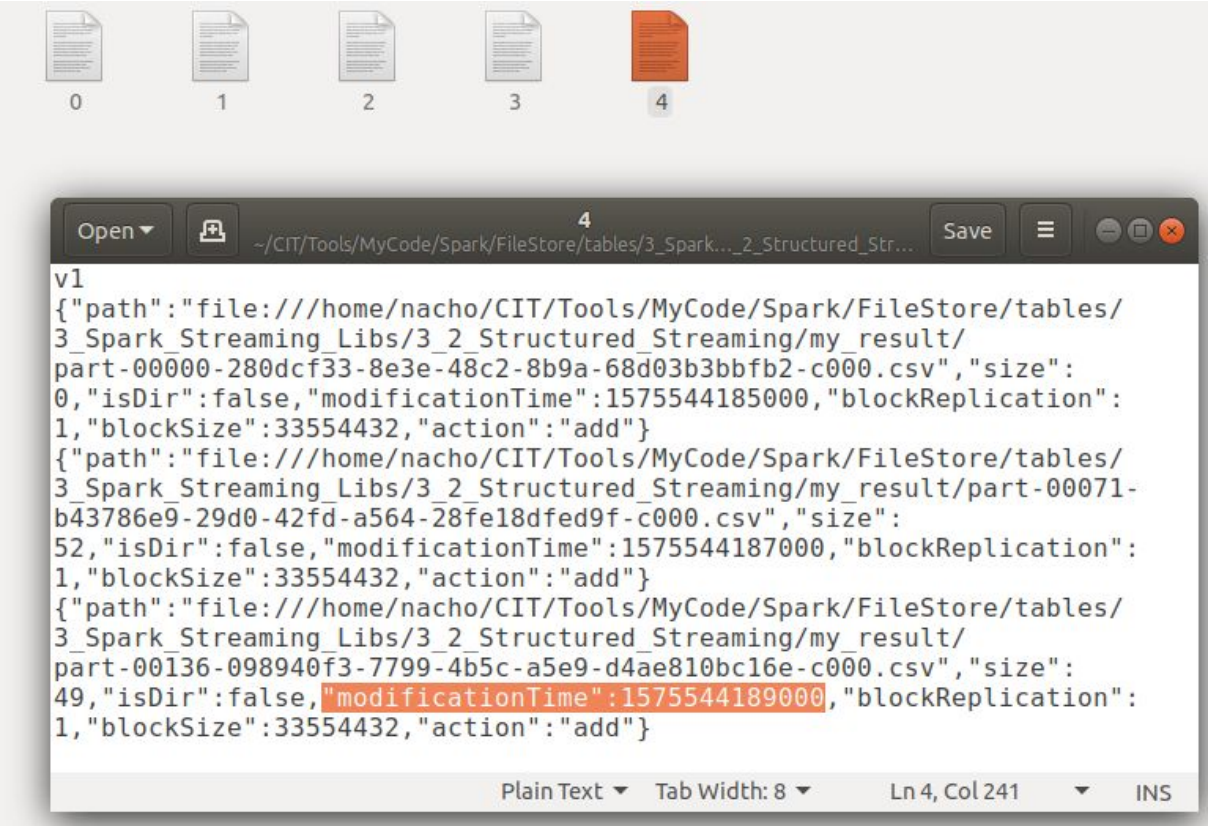
Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
Third block:



Console Sink vs. File Sink

As we can see, **4** relates to three files from my_result directory.
Third block:



Console Sink vs. File Sink

We cannot understand the results this way.

Console Sink vs. File Sink

We cannot understand the results this way.
Thus, let's parse them...
...based on the guesses we were making.

Console Sink vs. File Sink

The python file:
structured_streaming_post_process_my_result.py
does this parsing, turning my_result content into *human-legible*

Console Sink vs. File Sink

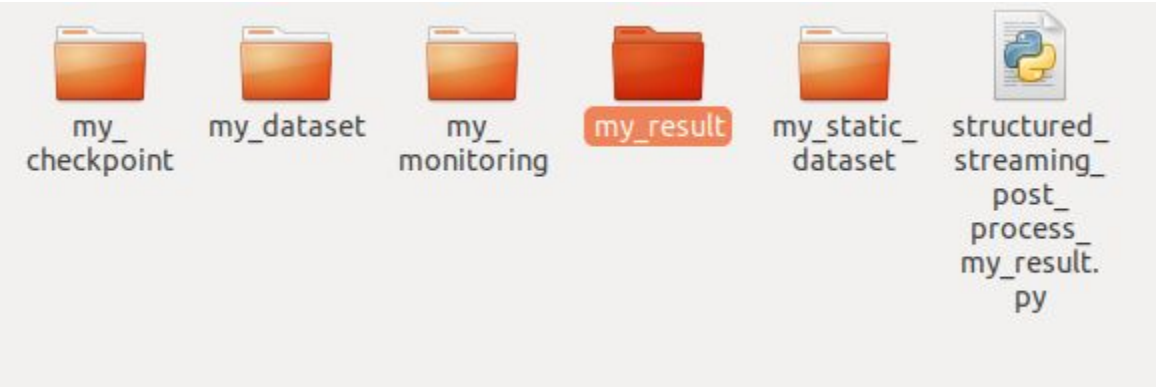
The python file:
structured_streaming_post_process_my_result.py
does this parsing, turning my_result content into *human-legible*

Console Sink vs. File Sink

The python file:

structured_streaming_post_process_my_result.py

does this parsing, turning my_result content into *human-legible*

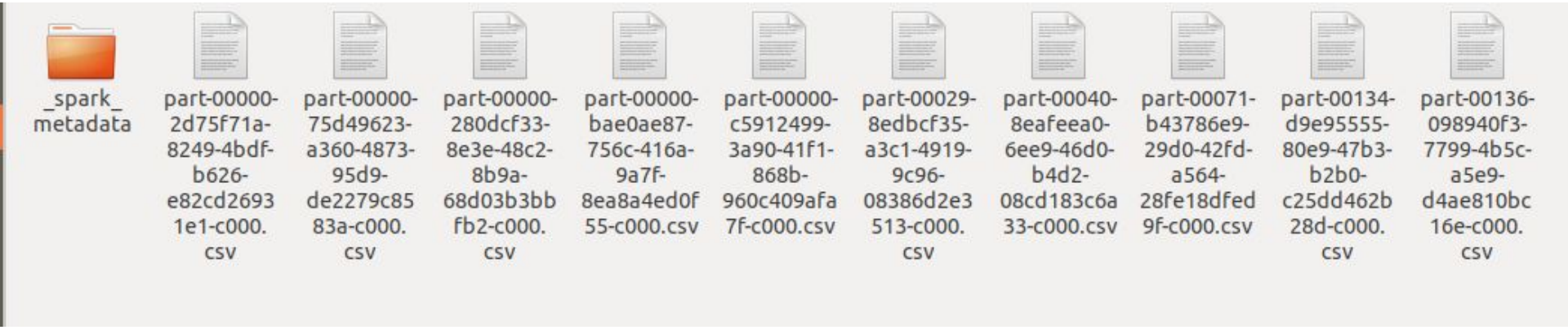


Console Sink vs. File Sink

The python file:

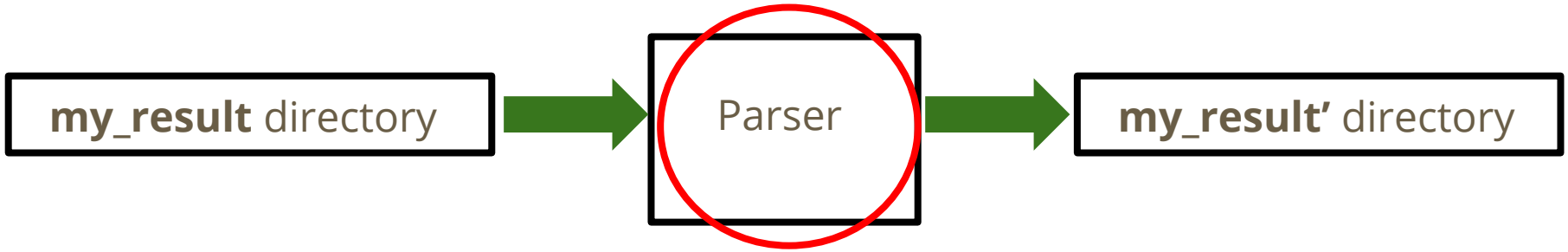
structured_streaming_post_process_my_result.py

does this parsing, turning my_result content into *human-legible*



Console Sink vs. File Sink

The python file:
structured_streaming_post_process_my_result.py
does this parsing, turning my_result content into *human-legible*

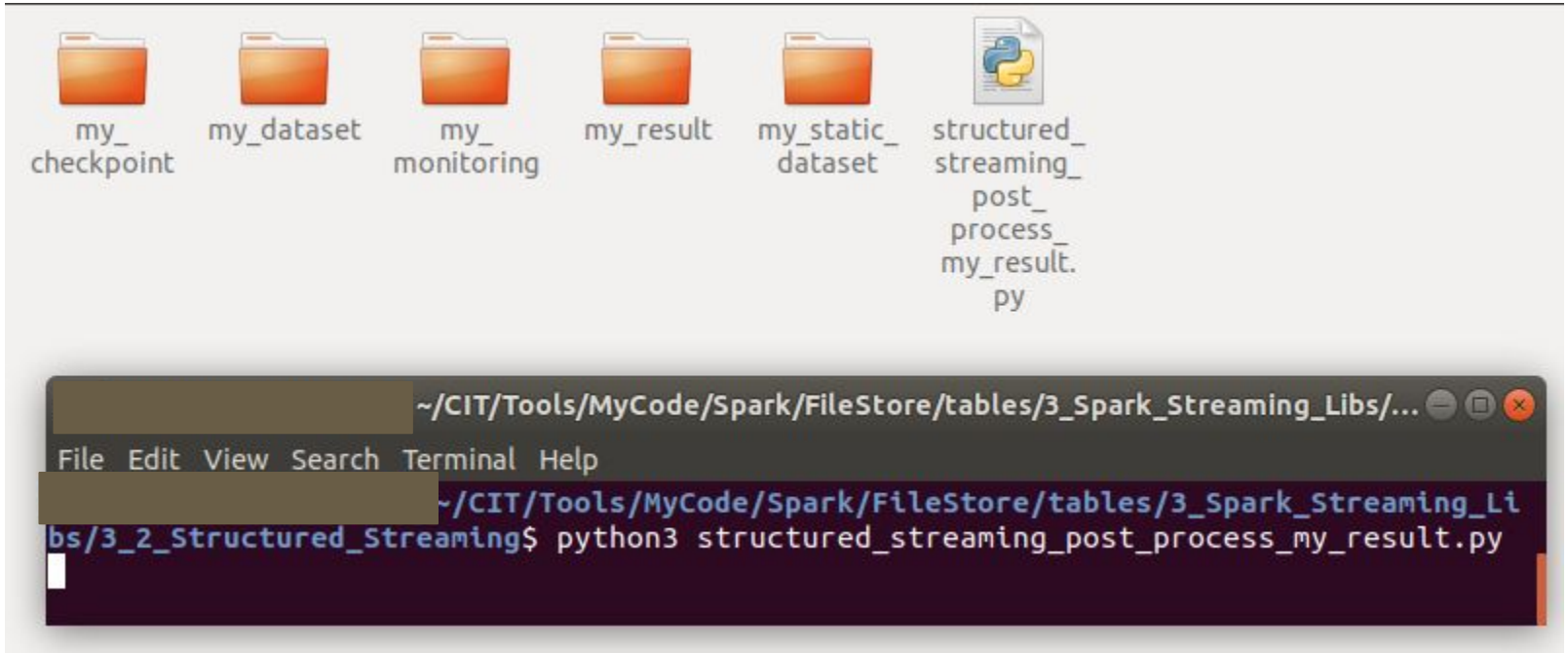


Console Sink vs. File Sink

The python file:

structured_streaming_post_process_my_result.py

does this parsing, turning my_result content into *human-legible*



Console Sink vs. File Sink

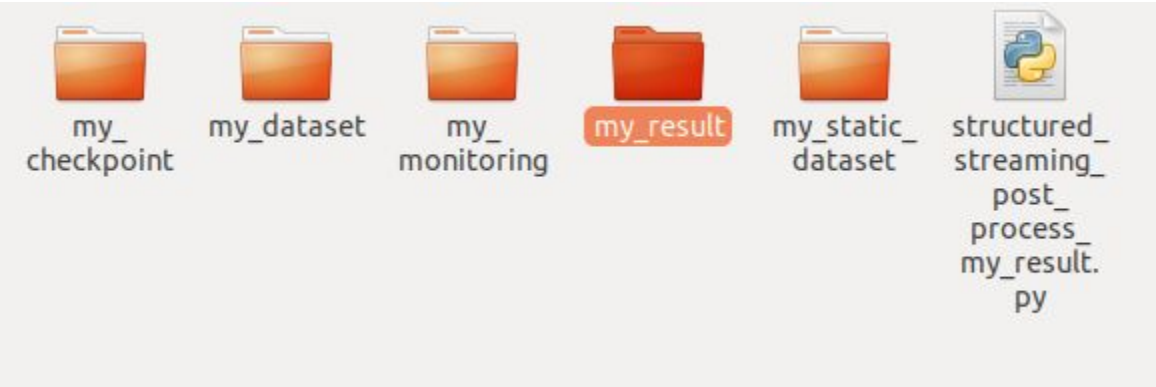
The python file:
structured_streaming_post_process_my_result.py
does this parsing, turning my_result content into *human-legible*

Console Sink vs. File Sink

The python file:

structured_streaming_post_process_my_result.py

does this parsing, turning my_result content into *human-legible*

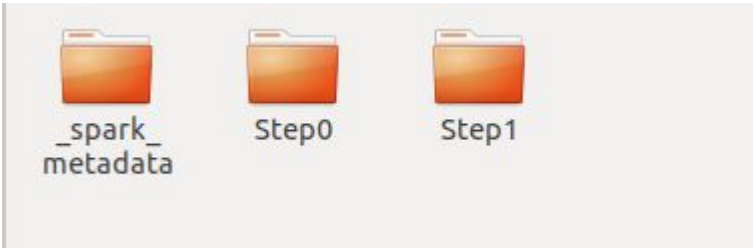


Console Sink vs. File Sink

The python file:

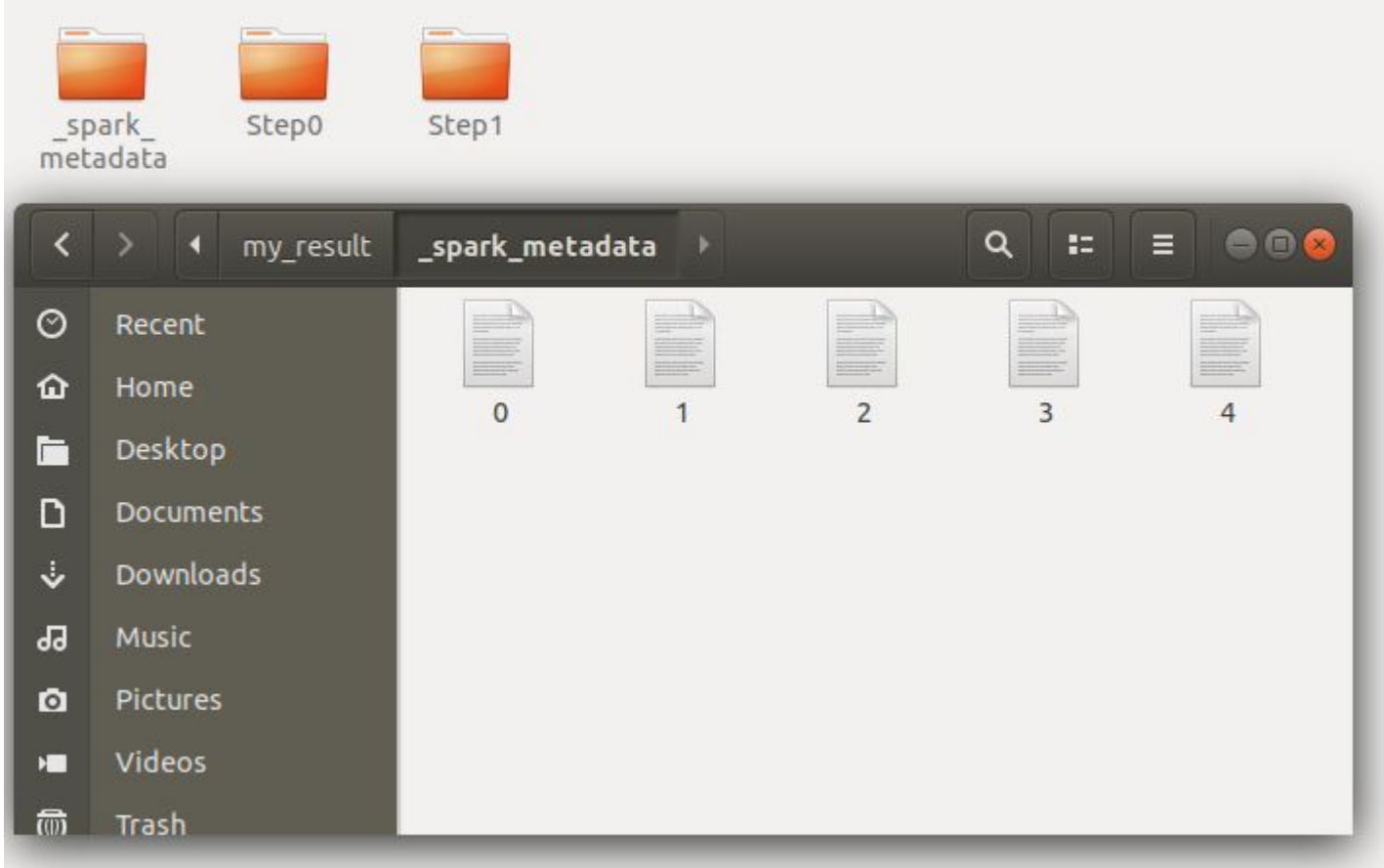
structured_streaming_post_process_my_result.py

does this parsing, turning my_result content into *human-legible*



Console Sink vs. File Sink

As we can see, the **_spark_metadata** folder remains the same.



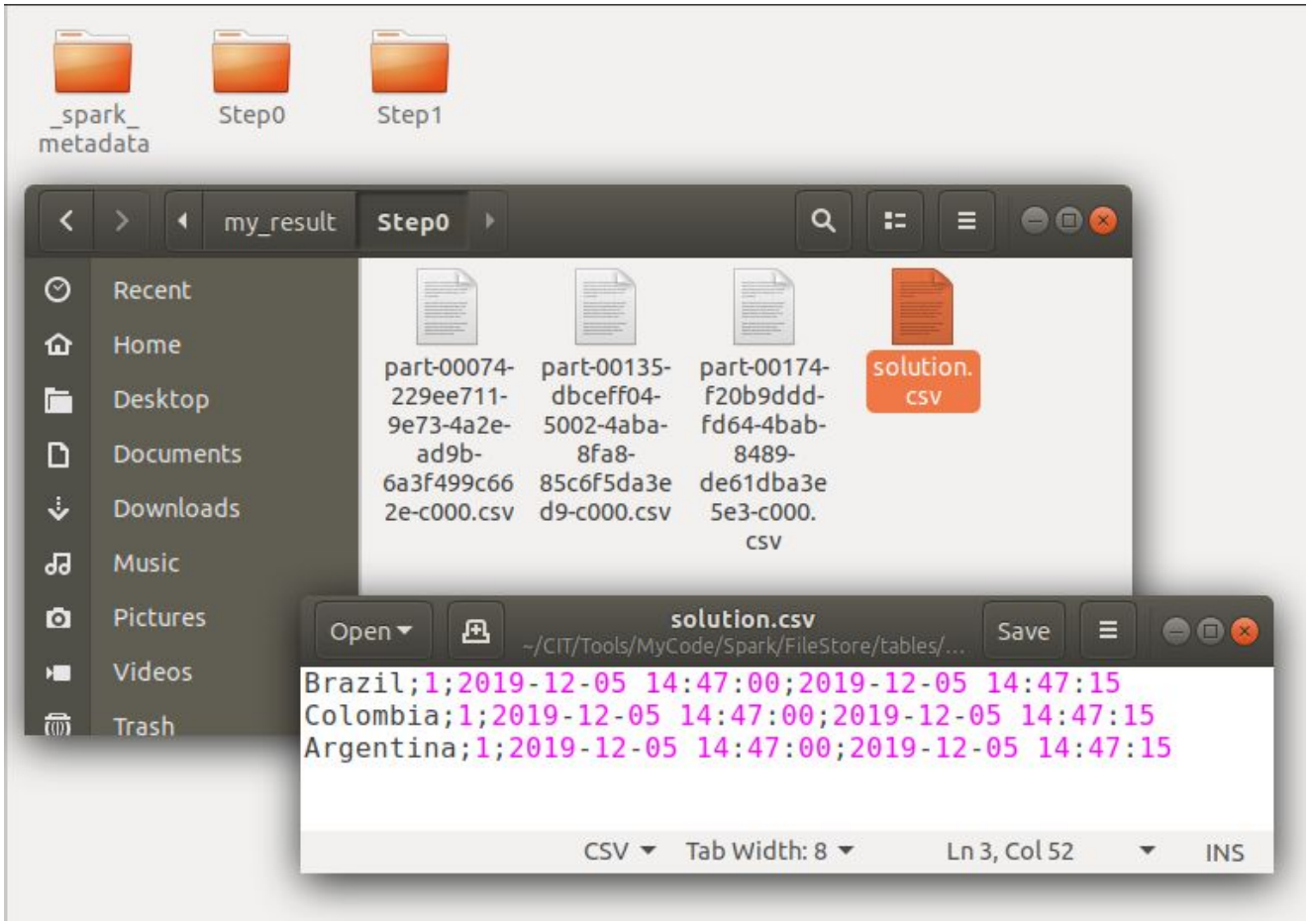
Console Sink vs. File Sink

The remaining folders contain the results of each time interval.

Console Sink vs. File Sink

The remaining folders contain the results of each time interval.

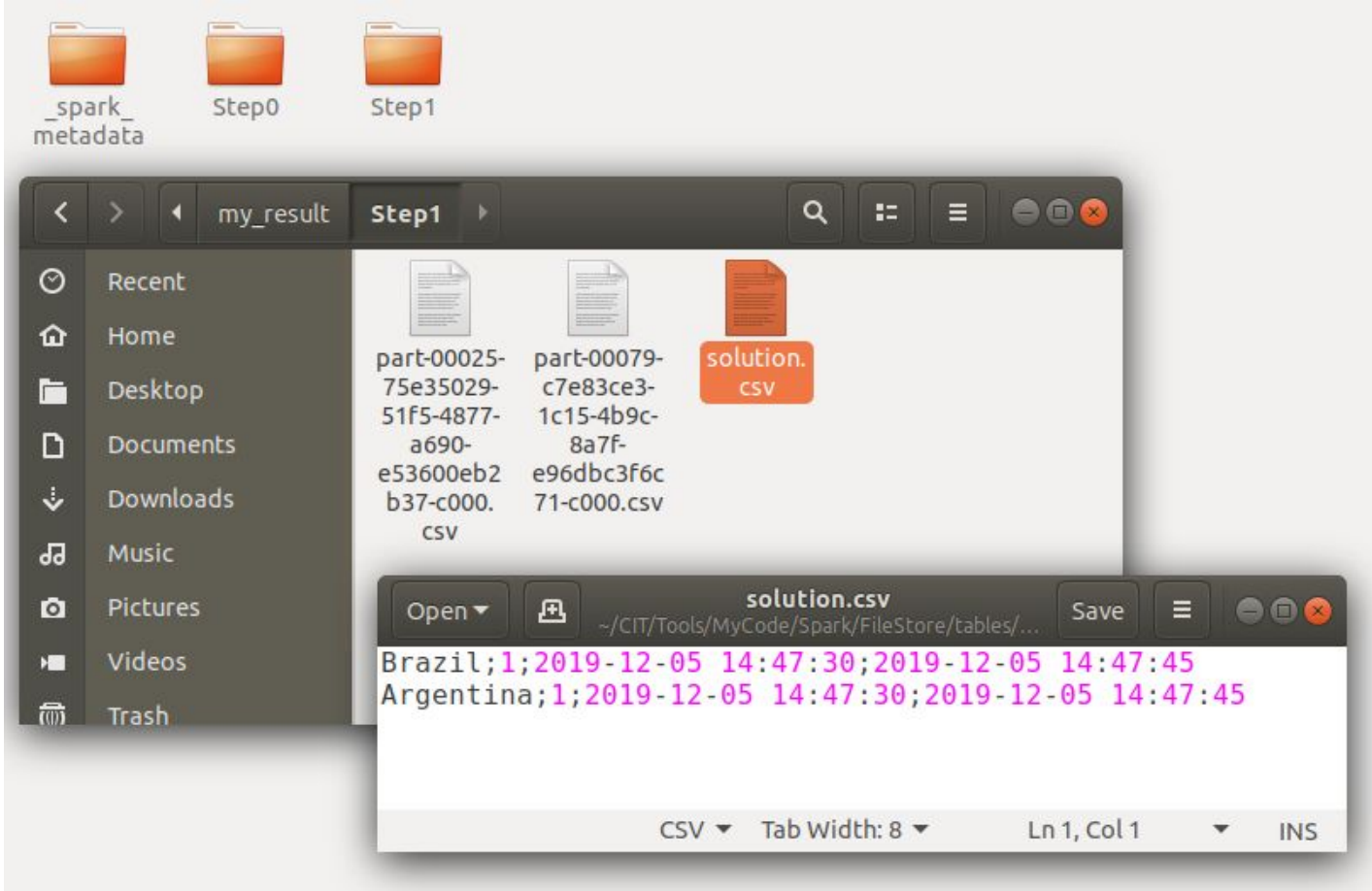
Step0:



Console Sink vs. File Sink

The remaining folders contain the results of each time interval.

Step1:



Console Sink vs. File Sink

As we see, our parser allows us to get the results in human-legible format.

Console Sink vs. File Sink

But, again, this parser is based in what we *guess* the result information produced by Spark Structured Streaming means!

Console Sink vs. File Sink

But, again, this parser is based in what we *guess* the result information produced by Spark Structured Streaming means!

We have no guarantee whatsoever that the parser will work for other code examples than the ones proposed in this module.

Console Sink vs. File Sink

But, again, this parser is based in what we *guess* the result information produced by Spark Structured Streaming means!

We have no guarantee whatsoever that the parser will work for other code examples than the ones proposed in this module.

Neither we do of the parser being compatible with future versions of Spark Structured Streaming.

Console Sink vs. File Sink

So, all in all, the parser is just
a walkaround solution to
the messy way of Spark Structured Streaming
of producing File sink results!

Console Sink vs. File Sink

That being said, let's now use the parser to assess our 4 concepts with File sink:

- Concept2: Append Mode.
- Concept3: Append Mode with window.
- Concept4: Complete Mode.
- Concept5: Complete Mode with window.

Console Sink vs. File Sink

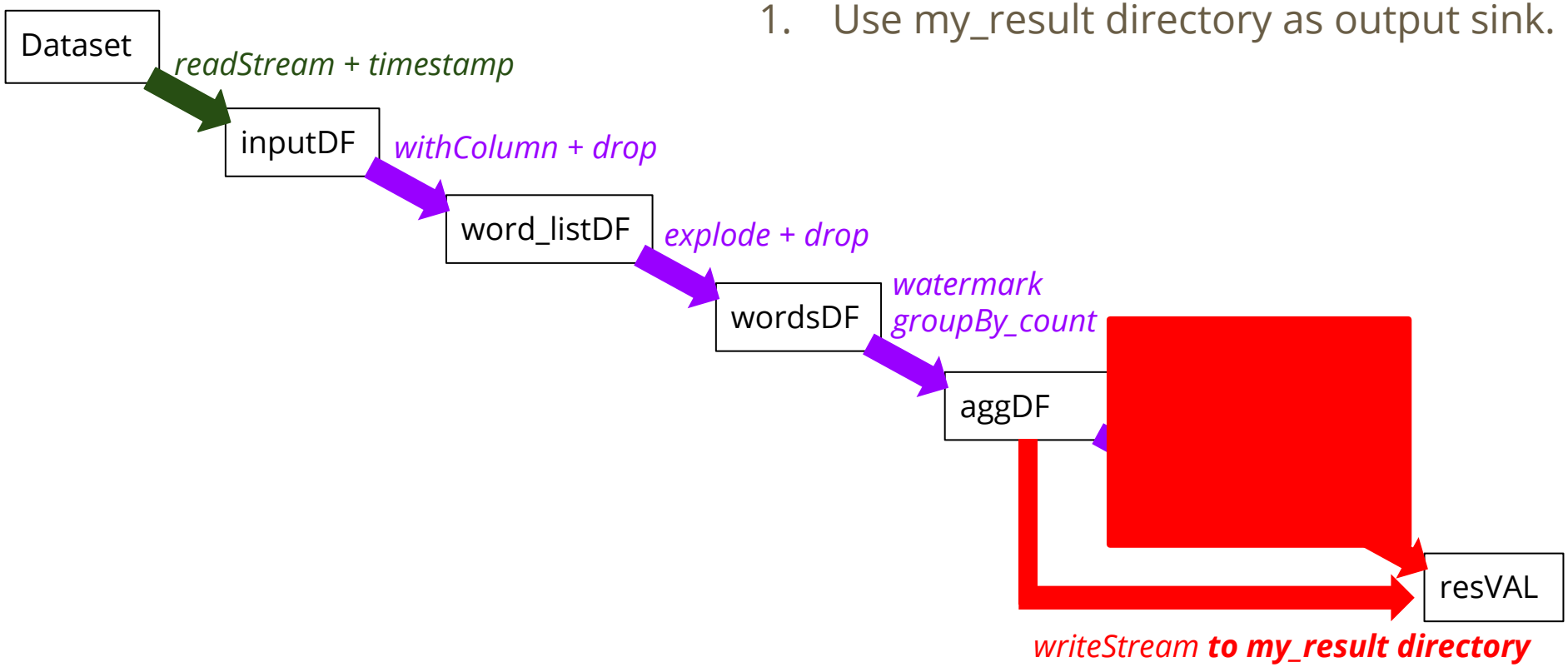
That being said, let's now use the parser to assess our 4 concepts with File sink:

- Concept2: Append Mode.
- Concept3: Append Mode with window.
- Concept4: Complete Mode.
- Concept5: Complete Mode with window.

Console Sink vs. File Sink

We modify our program **p05_ok_but_misses_time_interval.py** into **p11_ok_but_misses_time_interval.py** by:

1. Use my_result directory as output sink.



Console Sink vs. File Sink

Program **p11_ok_but_misses_time_interval.py** executes without error.

Its results are presented in the file **p11_ok_but_misses_time_interval.txt**

The results are now as we originally expecting.
However, for a reason we can't understand, the results for the last time step are never computed.

Console Sink vs. File Sink

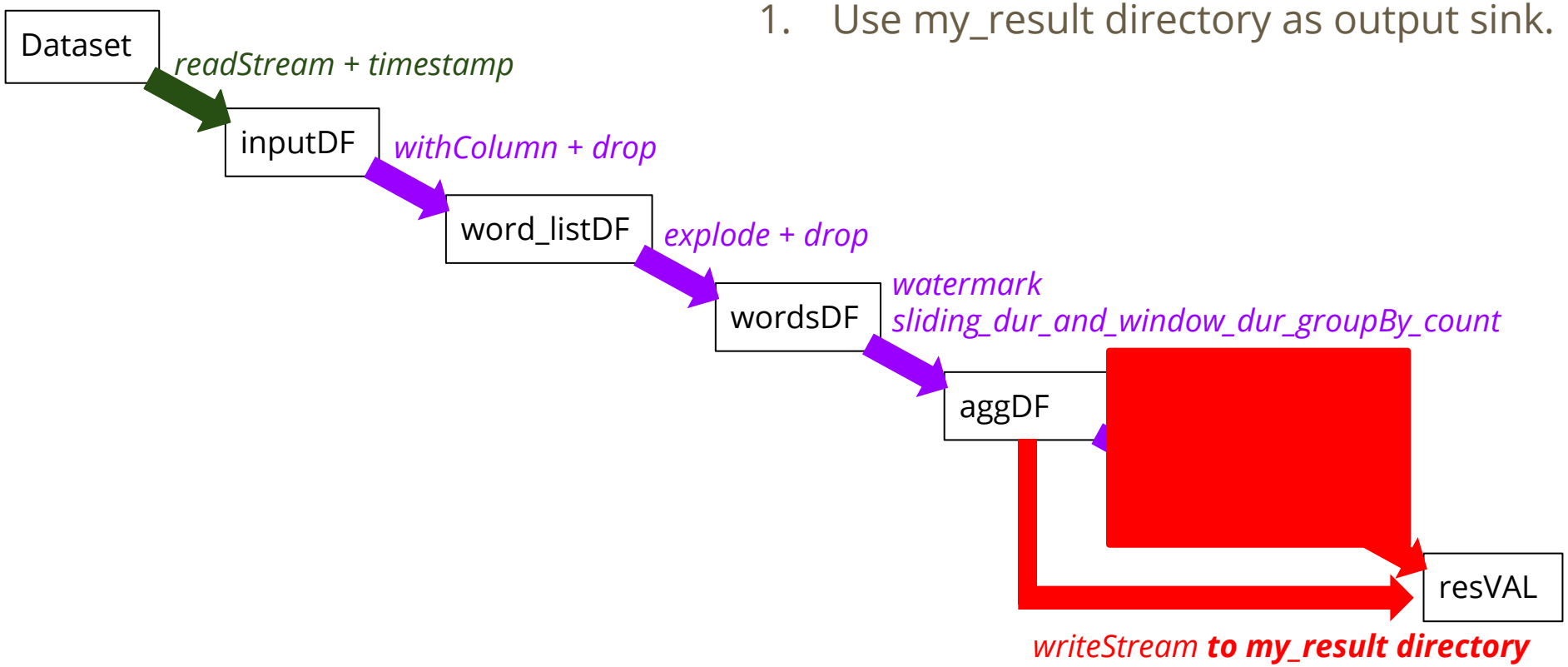
That being said, let's now use the parser to assess our 4 concepts with File sink:

- Concept2: Append Mode.
- Concept3: Append Mode with window.
- Concept4: Complete Mode.
- Concept5: Complete Mode with window.

Console Sink vs. File Sink

We modify our previous program **p06_ok_but_misses_time_interval.py** into **p12_ok_different_results.py** by:

1. Use my_result directory as output sink.



Console Sink vs. File Sink

Program **p12_ok_different_results.py** executes without error.

Its results are presented in the file **p12_ok_different_results.txt**

The results are different from what we got for
p06_ok_but_misses_time_interval.py.
Really strange.

Console Sink vs. File Sink

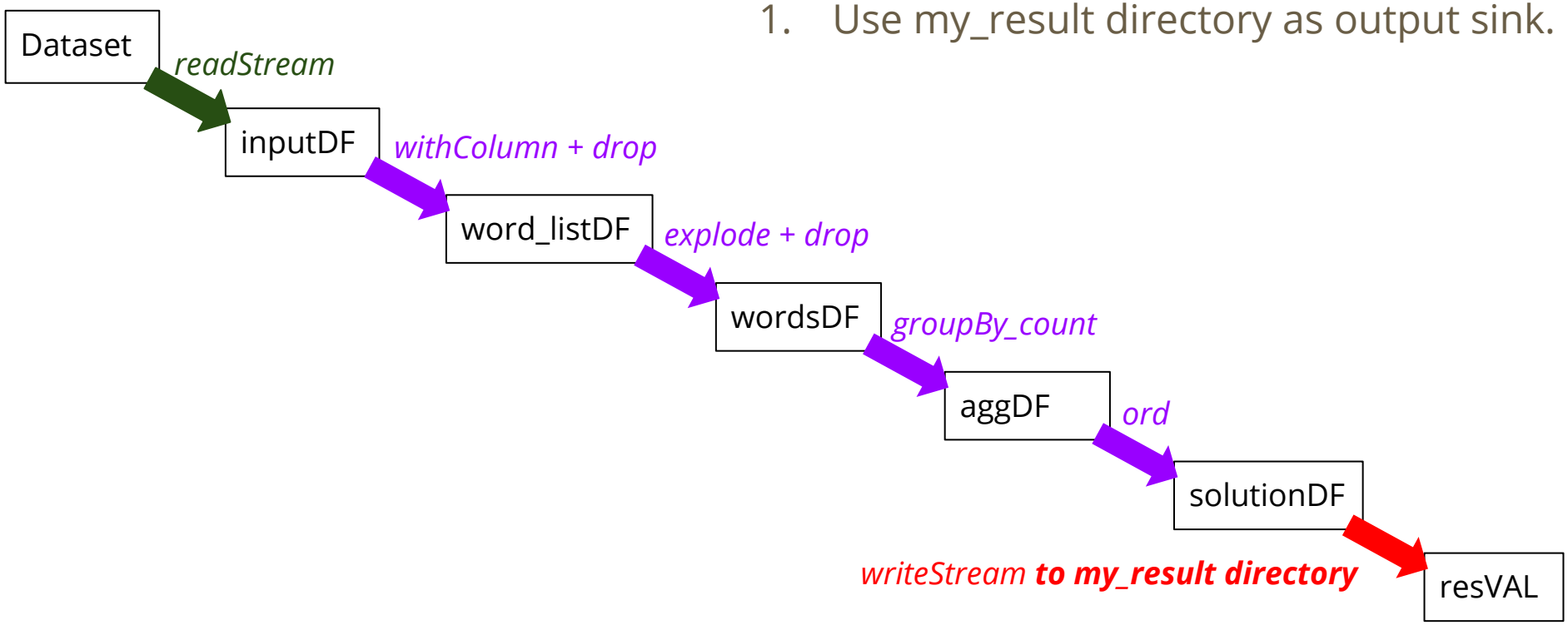
That being said, let's now use the parser to assess our 4 concepts with File sink:

- Concept2: Append Mode.
- Concept3: Append Mode with window.
- Concept4: Complete Mode.
- Concept5: Complete Mode with window.

Console Sink vs. File Sink

We modify our previous program **program p07_ok.py** into **p13_ERROR_File_not_supported.py** by:

1. Use my_result directory as output sink.



Console Sink vs. File Sink

Program **p13_ERROR_File_not_supported** fails:

ERROR: Data source csv does not support Complete output mode.

Console Sink vs. File Sink

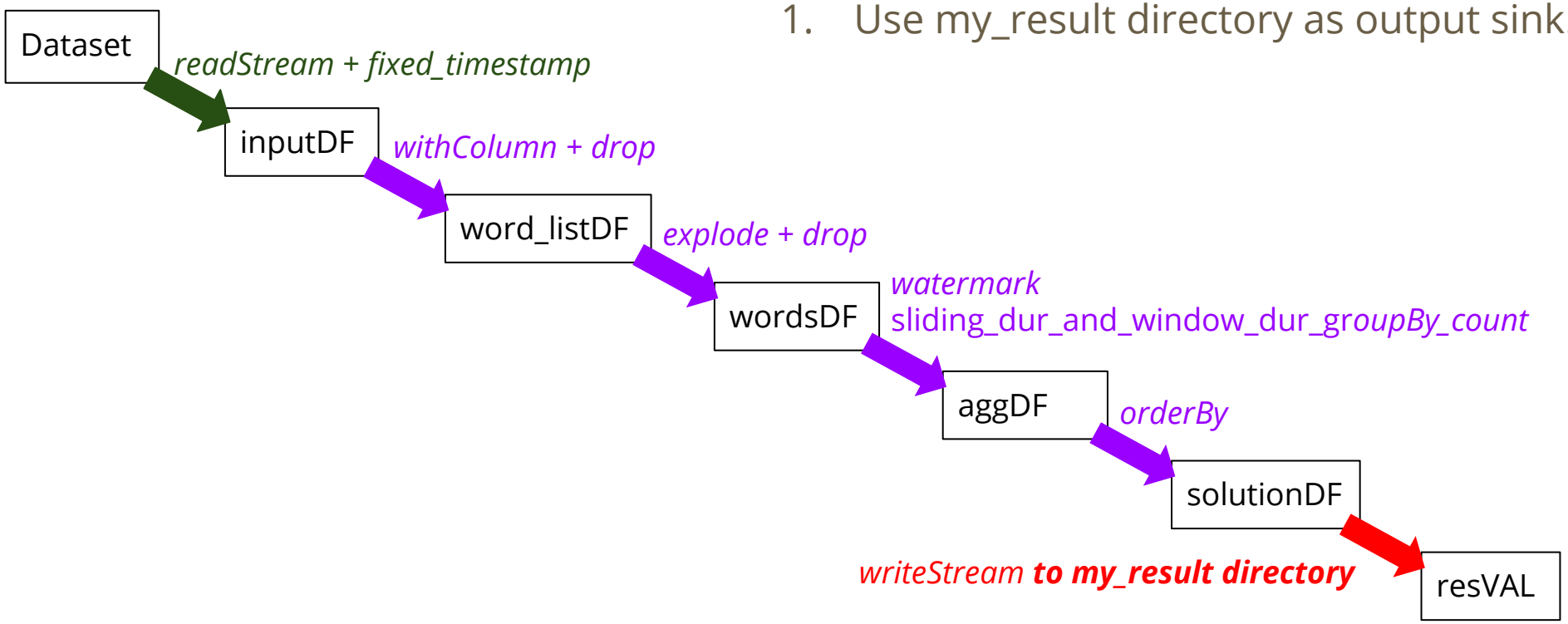
That being said, let's now use the parser to assess our 4 concepts with File sink:

- Concept2: Append Mode.
- Concept3: Append Mode with window.
- Concept4: Complete Mode.
- Concept5: Complete Mode with window.

Console Sink vs. File Sink

We modify our previous program **p10_ok_different_results.py** into **p14_ERROR_File_not_supported.py** by:

1. Use my_result directory as output sink.



Console Sink vs. File Sink

Program **p14_ERROR_File_not_supported** fails:

ERROR: Data source csv does not support Complete output mode.

From DF to SDF: A Program-Driven Example

All in all, we have seen that we can apply some of the functionality of **Spark Streaming** to **Spark Structured Streaming** as well.

From DF to SDF: A Program-Driven Example

All in all, we have seen that we can apply some of the functionality of **Spark Streaming** to **Spark Structured Streaming** as well.

However, there are quite some differences and limitations making **Spark Structured Streaming** not as appealing as it should be.

From DF to SDF: A Program-Driven Example

It that was not enough, Console sink does not work at the moment on Databricks. Programs are run, but no output is shown.

The bug is reported, but still no fix to it: https://forums.databricks.com/questions/15729/

From DF to SDF: A Program-Driven Example

Thus, only File sink works in Databricks at the moment, also requiring the use of the parser to see the results in human-legible format.

From DF to SDF: A Program-Driven Example

More specifically...

From DF to SDF: A Program-Driven Example

More specifically…

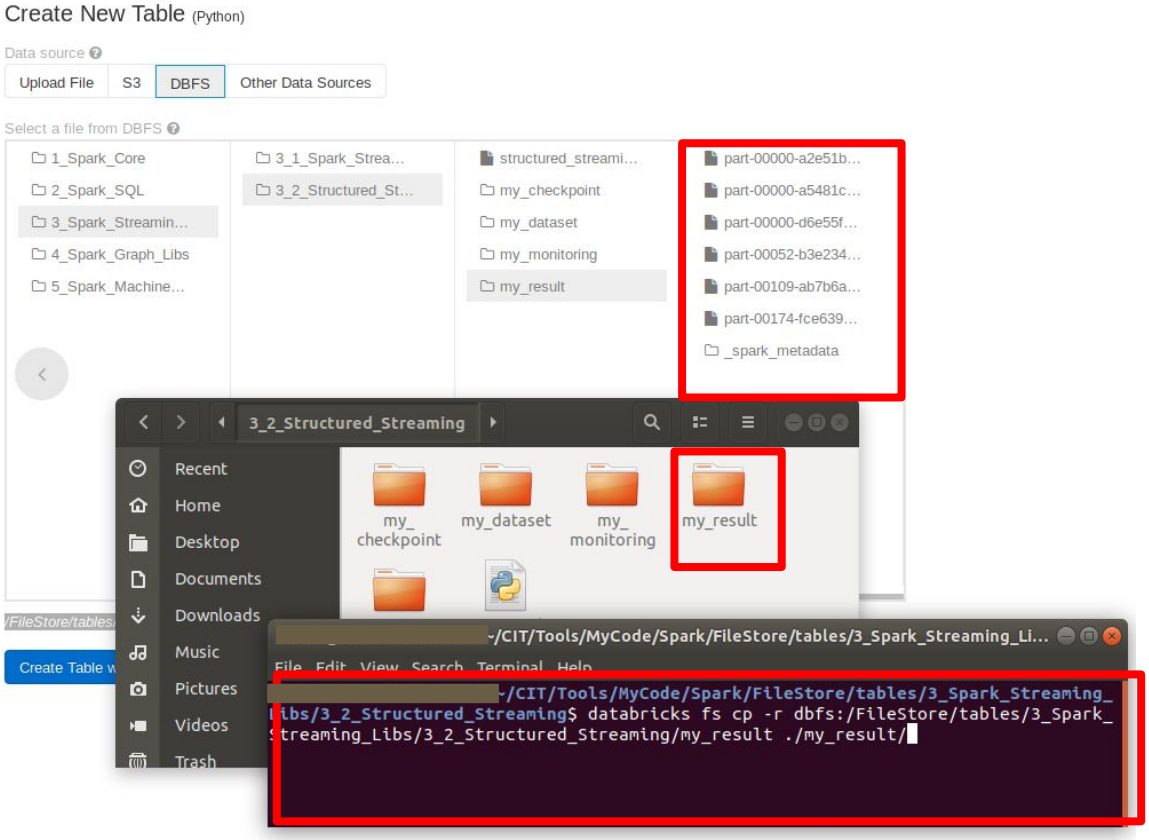
1. A program can be run in Databricks.

databricks

From DF to SDF: A Program-Driven Example

More specifically…

2. Its result can be brought back to the local hard drive of our computer by using the databricks-cli.



From DF to SDF: A Program-Driven Example

More specifically…

3. And, finally, the parser can be run in our local machine, so as to get the results in human-legible format.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. From DF to SDF: A Program-Driven Example.
   a. A Program-Driven Example.
   b. Concept2: Append Mode.
   c. Concept3: Append Mode with Windows.
   d. Concept4: Complete Mode.
   e. Concept5: Complete Mode with Windows.
   f. Console Sink vs. File Sink.

Outline

1. Setting Up the Context.
2. From DStream to SDF.
3. From DF to SDF: A Program-Driven Example.

Thank you for your attention!