CORK INSTITUTE OF TECHNOLOGY

INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Big Data Processing

L15-16: Spark Streaming

Dr. Ignacio Castineiras

Department of Computer Science

Outline

1. Setting Up the Context.
2. Measurement Unit: Time Interval & Data Batch.
3. From RDDs to a DStream.
4. File Transfer Process.
5. Spark Streaming Context Process.
6. Stateless and Stateful Operations.

Outline

1. Setting Up the Context.
2. Measurement Unit: Time Interval & Data Batch.
3. From RDDs to a DStream.
4. File Transfer Process.
5. Spark Streaming Context Process.
6. Stateless and Stateful Operations.

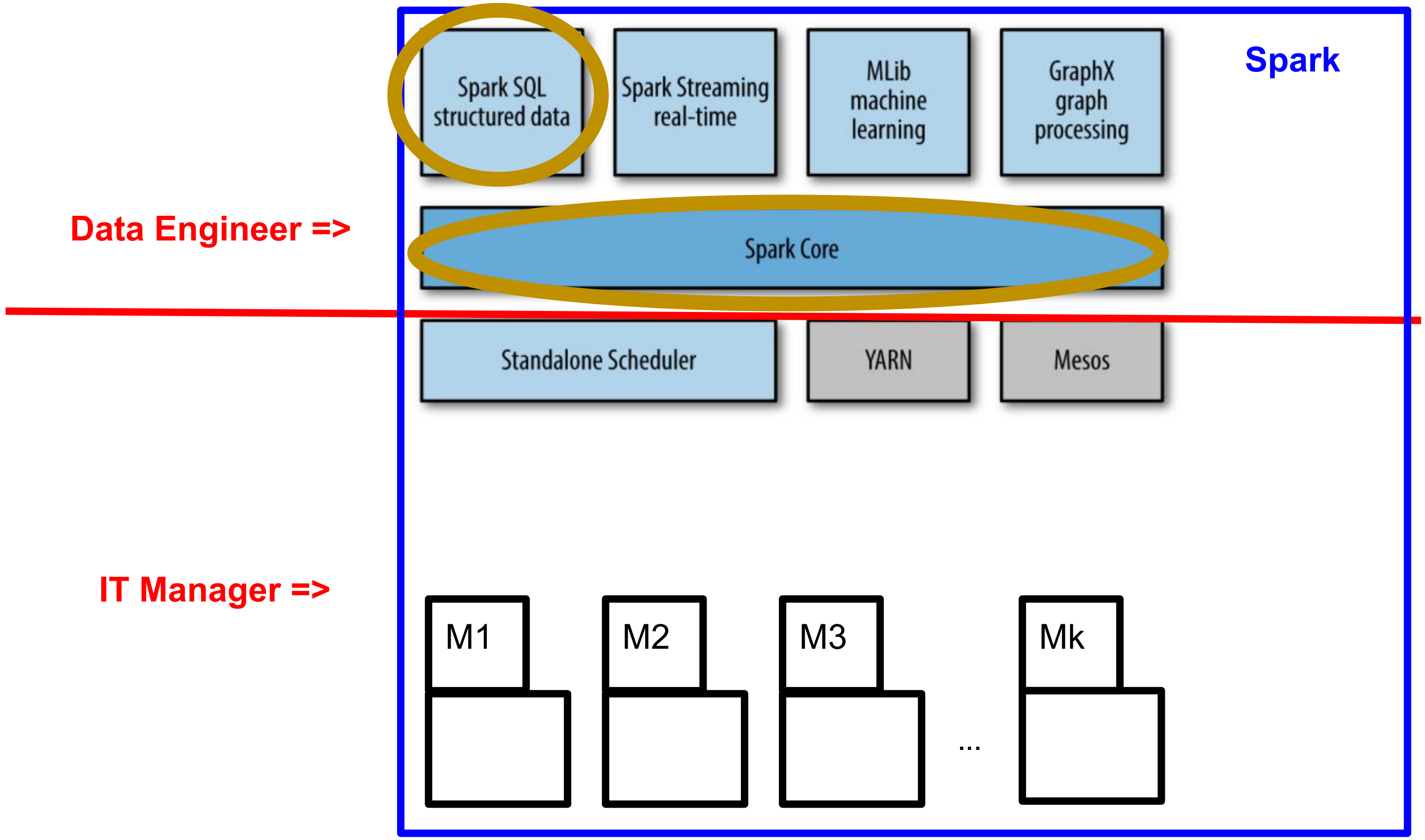
Setting the Context

1. At this stage we are fully familiar with Spark, our:
   - open-source
   - distributed
   - general-purpose
   - cluster-computing

   framework.

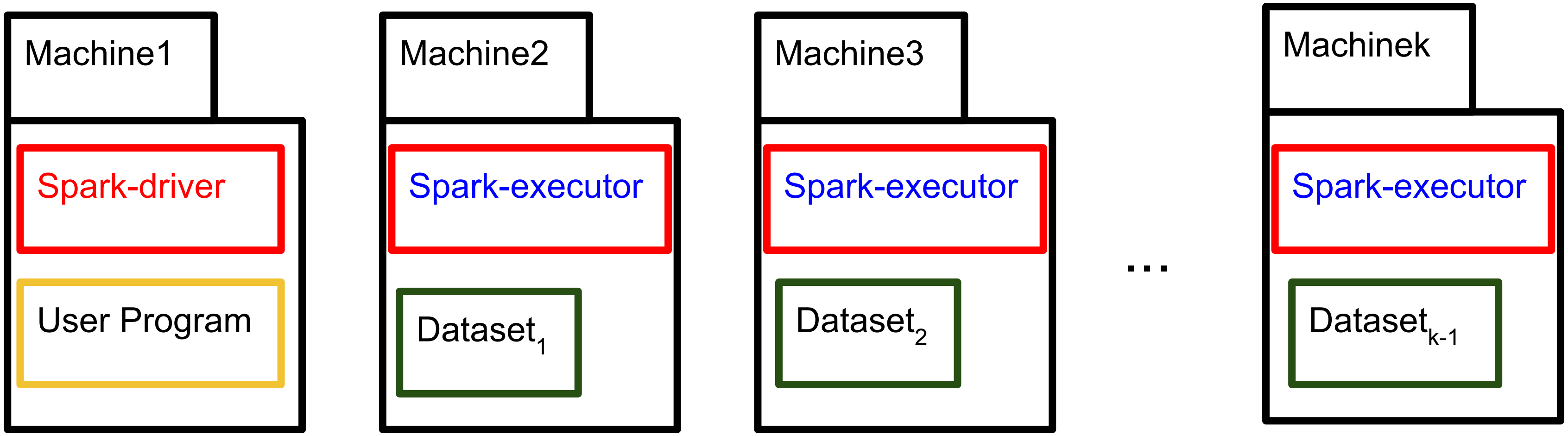
Setting the Context

1. In particular, we are fully familiar with the Data Engineer role after having explored in detail the Spark Core and Spark SQL components of Spark:



Setting the Context

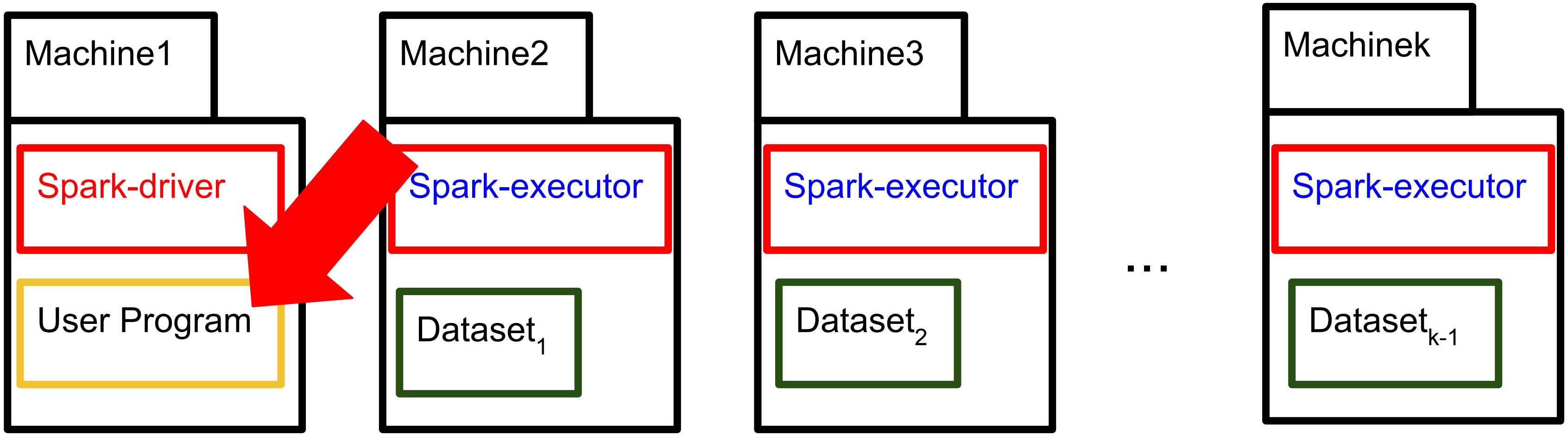
2. We have seen that a Spark program runs in a **cluster of computers**, connected among them so as to support the distributed computation.
   - The Spark driver coordinates the execution of the User program.
   - The Spark executor provide their CPU and memory for the execution of such program.



Setting the Context

3. We know by now that a Spark User Application has a life cycle based on:

   a. **Creation** operations to bring the dataset in.
   b. **Transformation** operations to manipulate the data.
   c. **Persist** operations to help lazy evaluation.
   d. **Action** operations to trigger the computation.

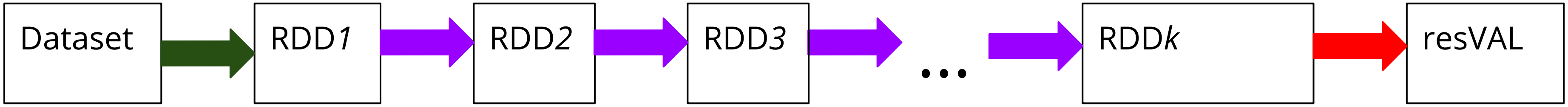


Setting the Context

4. We know by now that a Spark User Application has a life cycle based on:

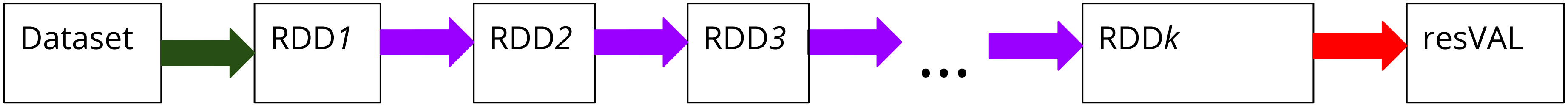
   a. **Creation** operations to bring the dataset in.
   b. **Transformation** operations to manipulate the data.
   c. **Persist** operations to help lazy evaluation.
   d. **Action** operations to trigger the computation.

If the User Application is based in Spark Core then the data abstraction are Resilient Distributed Datasets (RDDs).



Setting the Context

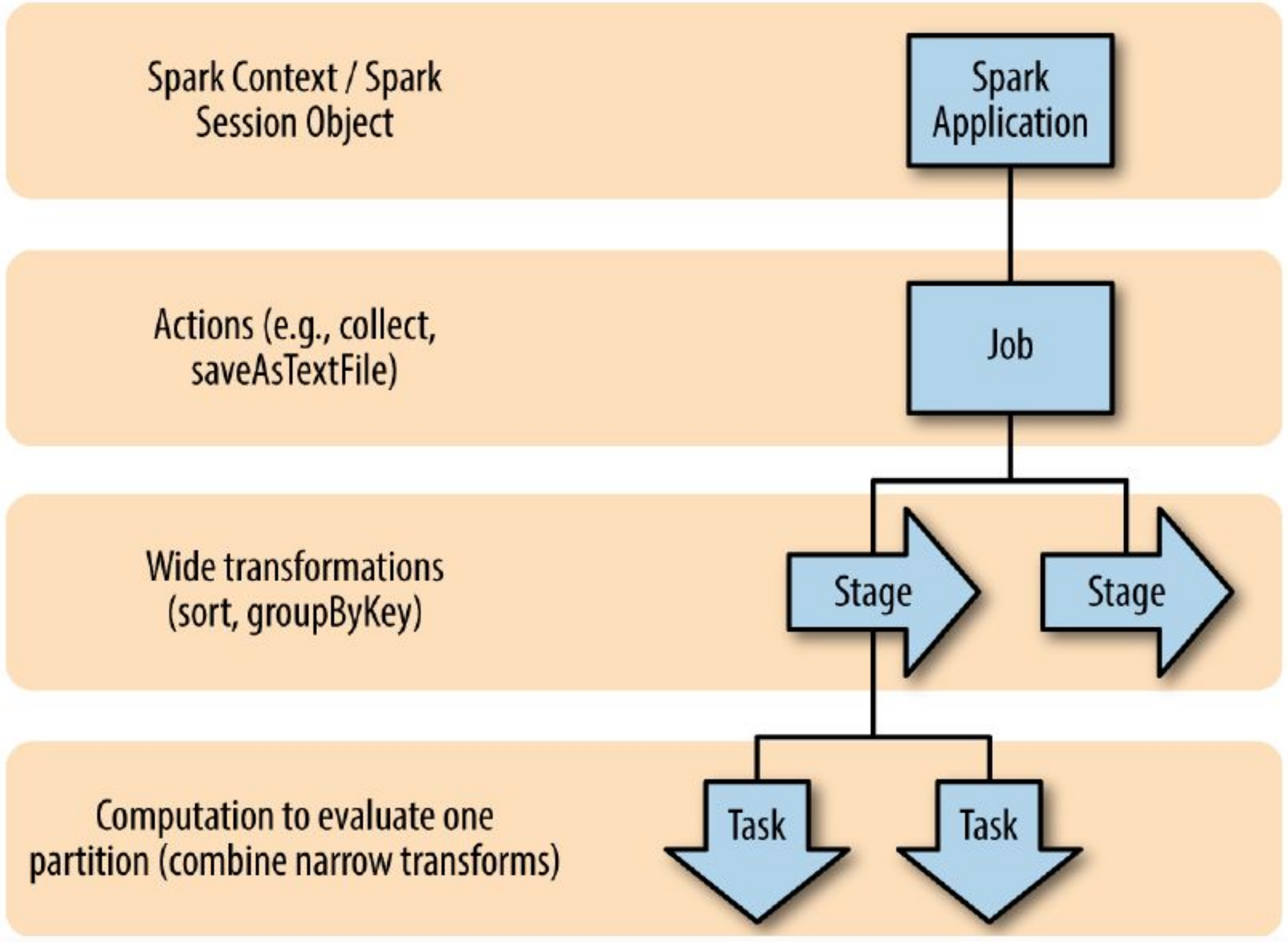
4. RDDs provide a small set of primitives,
each of them supporting the application of very general functions.



```
newRDD = inputRDD.map( f ) where f: a -> b

resVAL = inputRDD.reduce( f ) where f: a -> a -> a

resVAL = inputRDD.aggregate( accum, f1, f2 )
          where accum:: b, f1: b -> a -> b, f2: b -> b -> b

newRDD = inputRDD.reduceByKey( f ) where f: a -> a

newRDD = inputRDD.combineByKey( f1, f2, f3 )
          where f1: a -> b, f2: b -> a -> b and f3: b -> b -> b
```

Setting the Context

4. The execution of the Application is directly split into Jobs, Stages and Tasks.

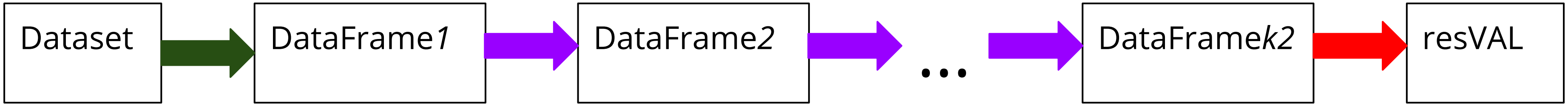

Spark Context / Spark Session Object
Spark Application
Actions (e.g., collect, saveAsTextFile)
Job
Wide transformations (sort, groupByKey)
Stage
Stage
Computation to evaluate one partition (combine narrow transforms)
Task
Task

Setting the Context

5. We know by now that a Spark User Application has a life cycle based on:

   a. **Creation** operations to bring the dataset in.
   b. **Transformation** operations to manipulate the data.
   c. **Persist** operations to help lazy evaluation.
   d. **Action** operations to trigger the computation.

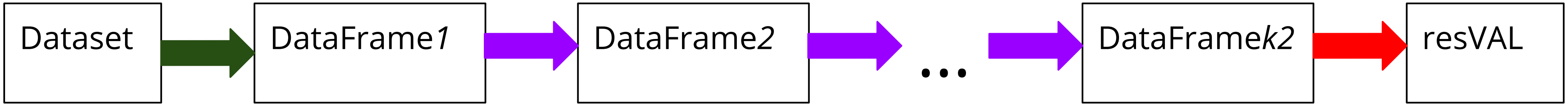
If the User Application is based in Spark SQL then the data abstraction are DataFrames (DF).



Dataset → DataFrame*1* → DataFrame*2* → … → DataFrame*k2* → resVAL

Setting the Context

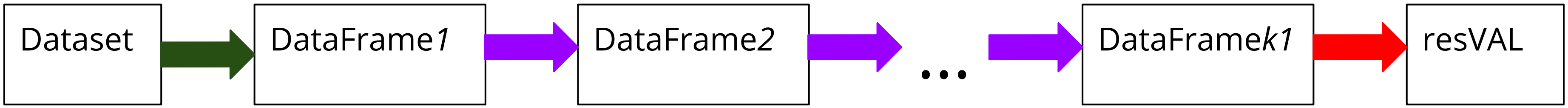
5. DFs provide an extensive set of Domain Specific Language (DSL) operators:
    - ○ Very restrictive with the expressions to be applied to.
    - ○ Making possible leverage optimisations in their application.



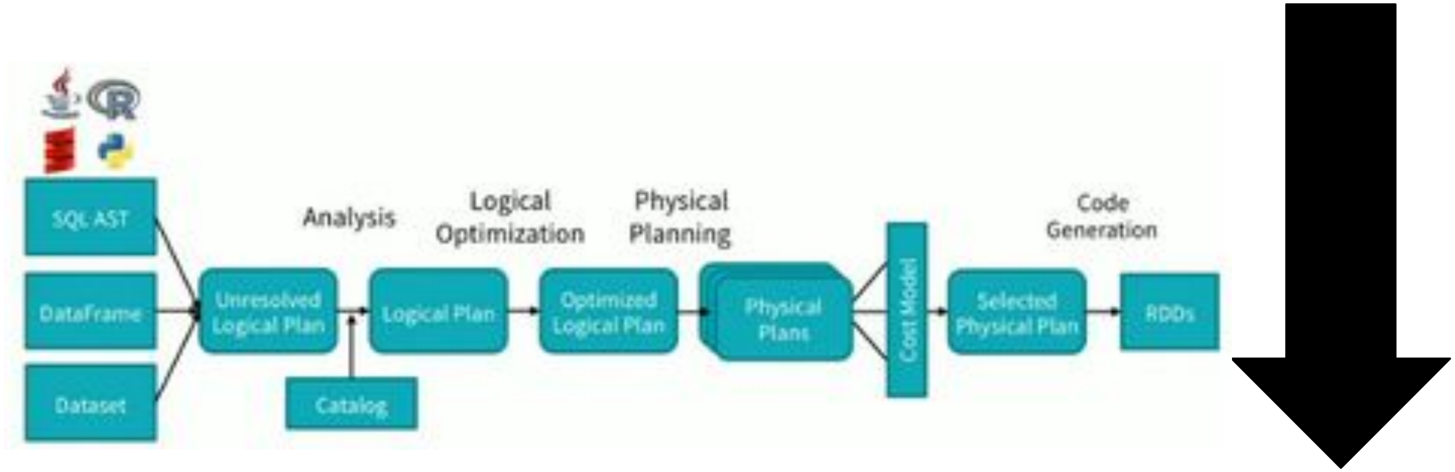
Dataset → DataFrame*1* → DataFrame2 → … → DataFrame*k*2 → resVAL

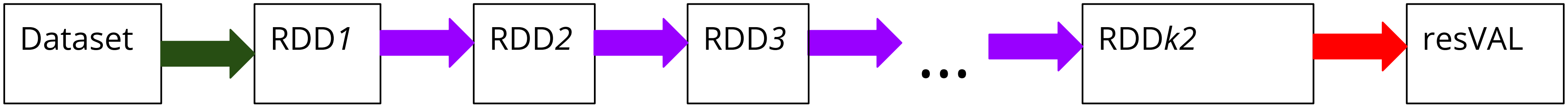
Setting the Context

5. For it to be executed, the Spark SQL DF -based program has to be firstly translated into an equivalent Spark Core RDD-based program.



Dataset → DataFrame*1* → DataFrame2 → … → DataFrame*k1* → resVAL





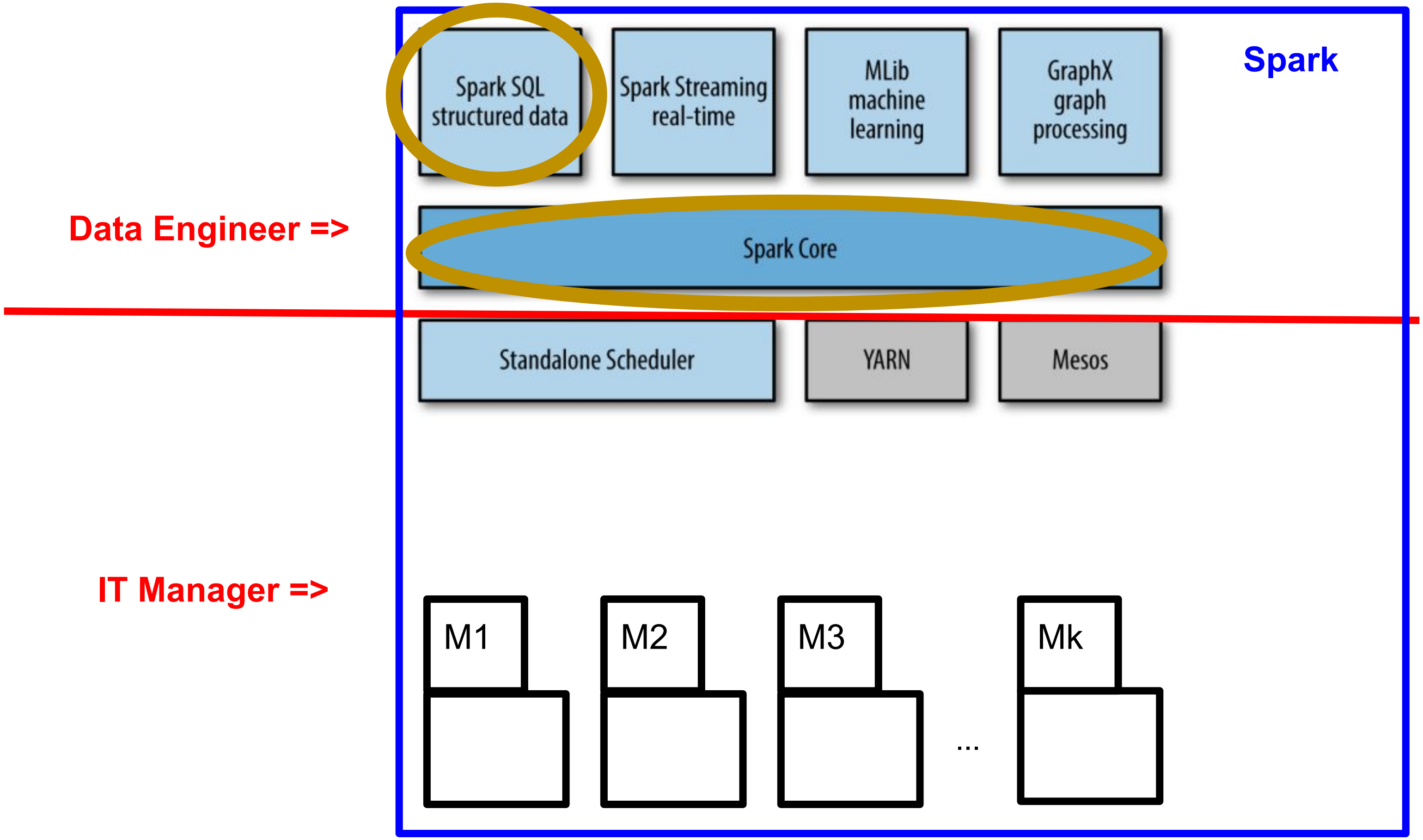
Dataset → RDD*1* → RDD2 → RDD*3* → … → RDD*k2* → resVAL

Setting the Context

However…

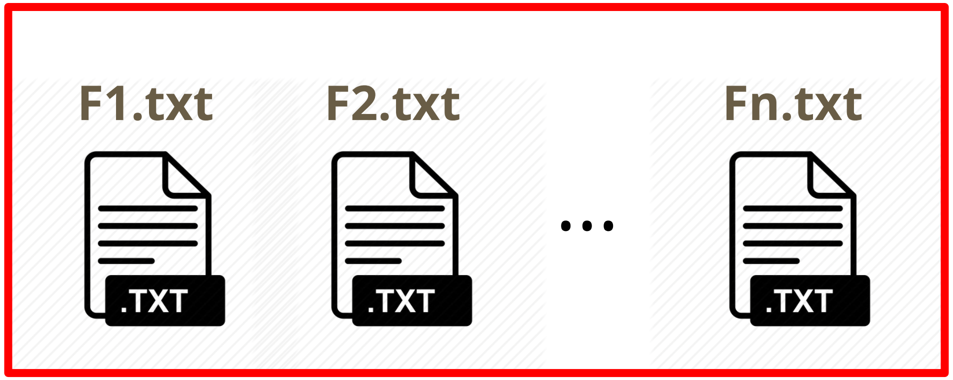
Setting the Context

All the examples we have seen for Spark Core and Spark SQL have shared a common assumption…



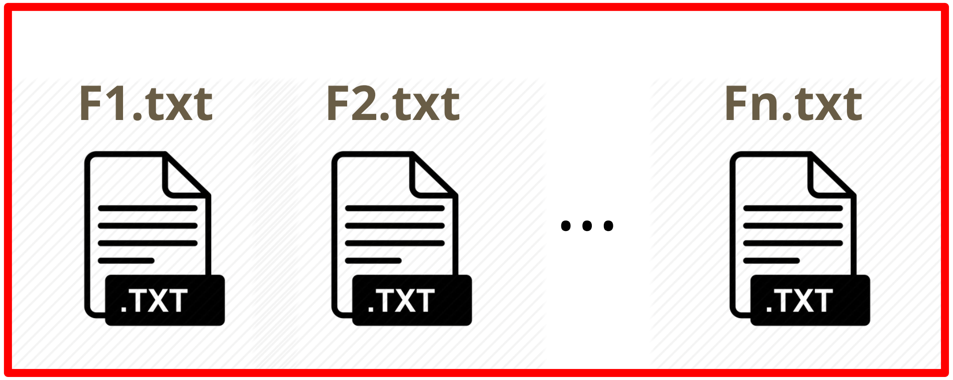
Setting the Context

The dataset being analysed was static!



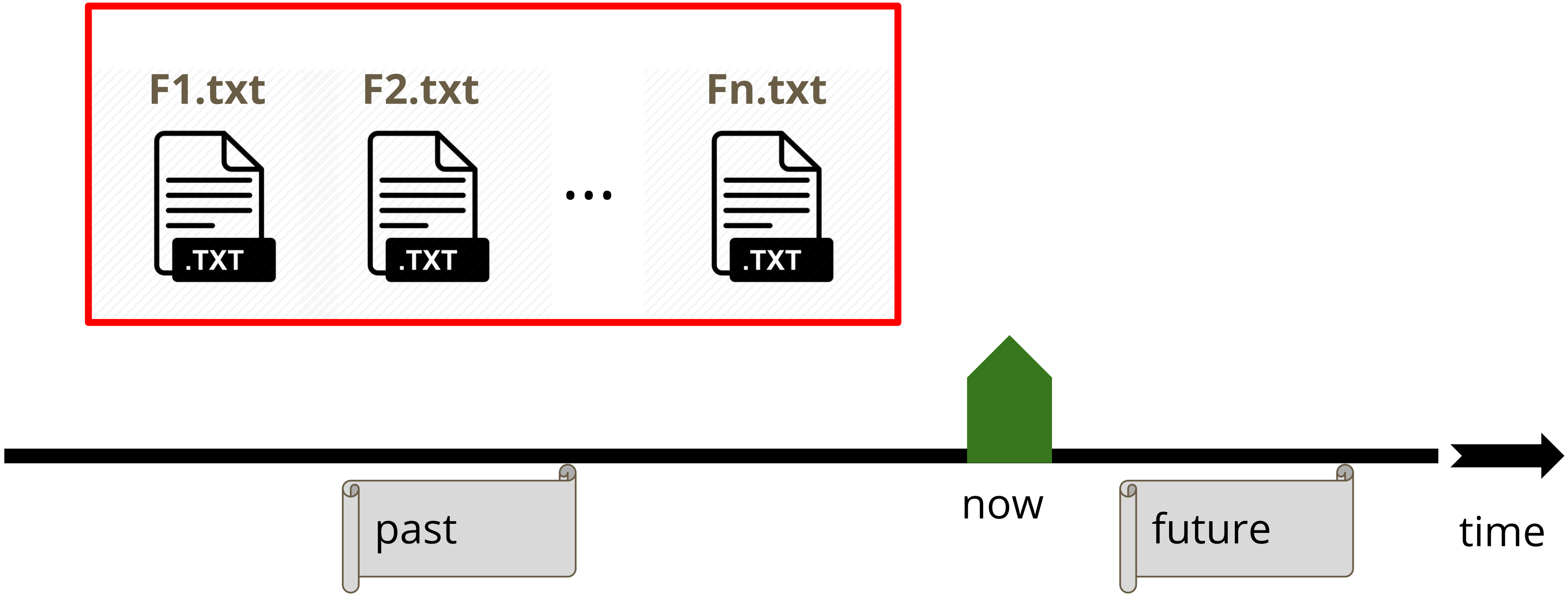
Setting the Context

If you want to see it more graphically, with a temporal line…



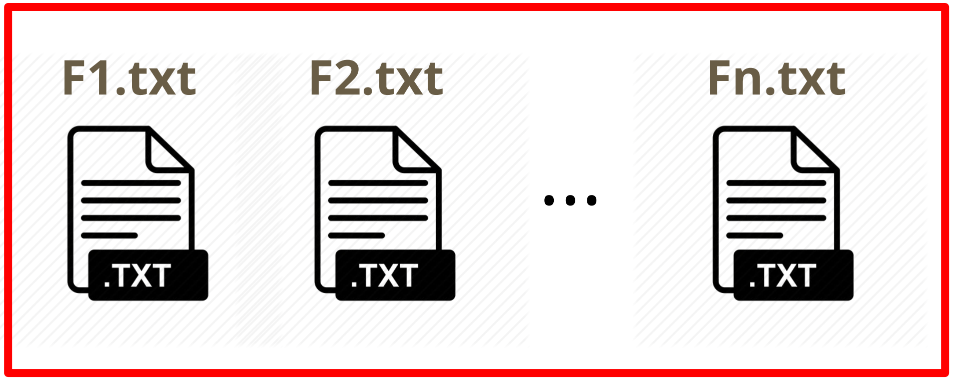
Setting the Context

We can say the dataset belonged to the past!



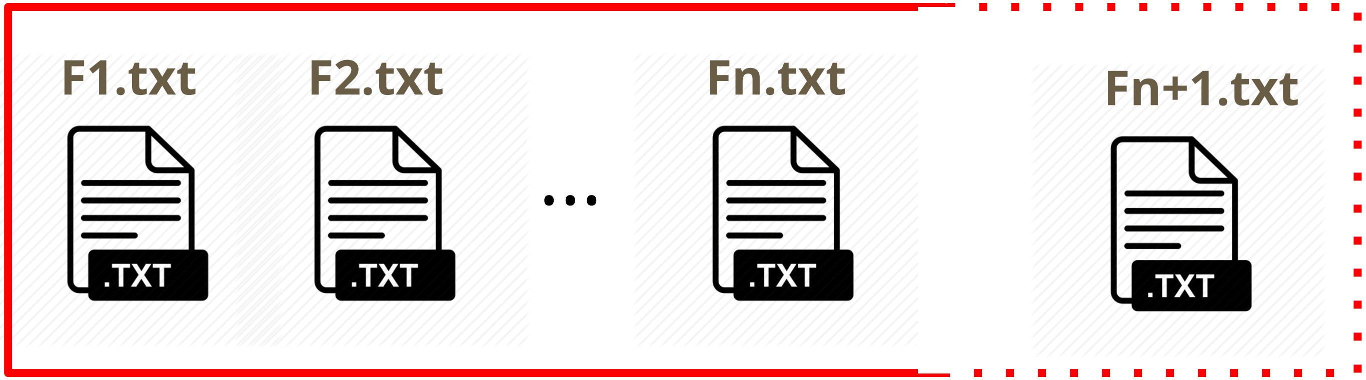
Setting the Context

But,
what does it happen if the dataset is dynamic?



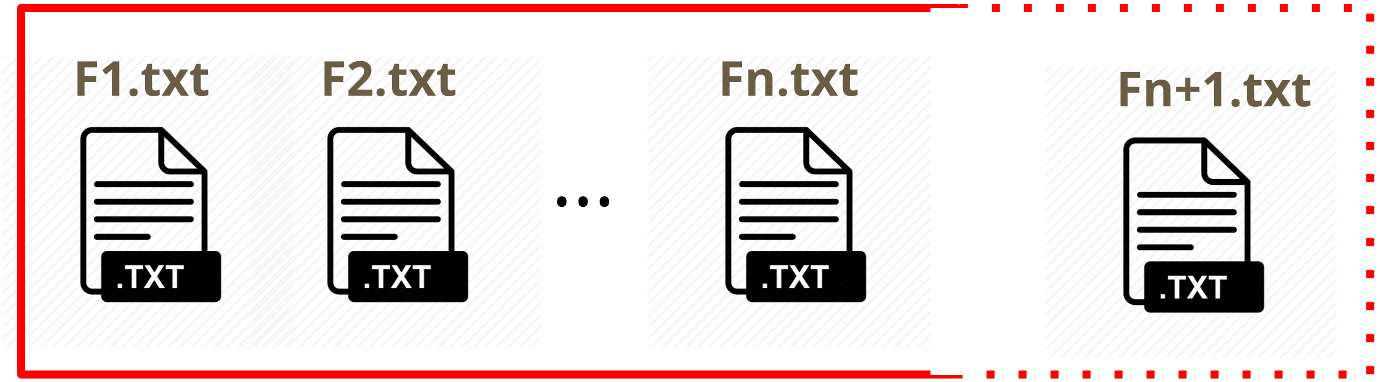
Setting the Context

But,
what does it happen if the dataset is dynamic?



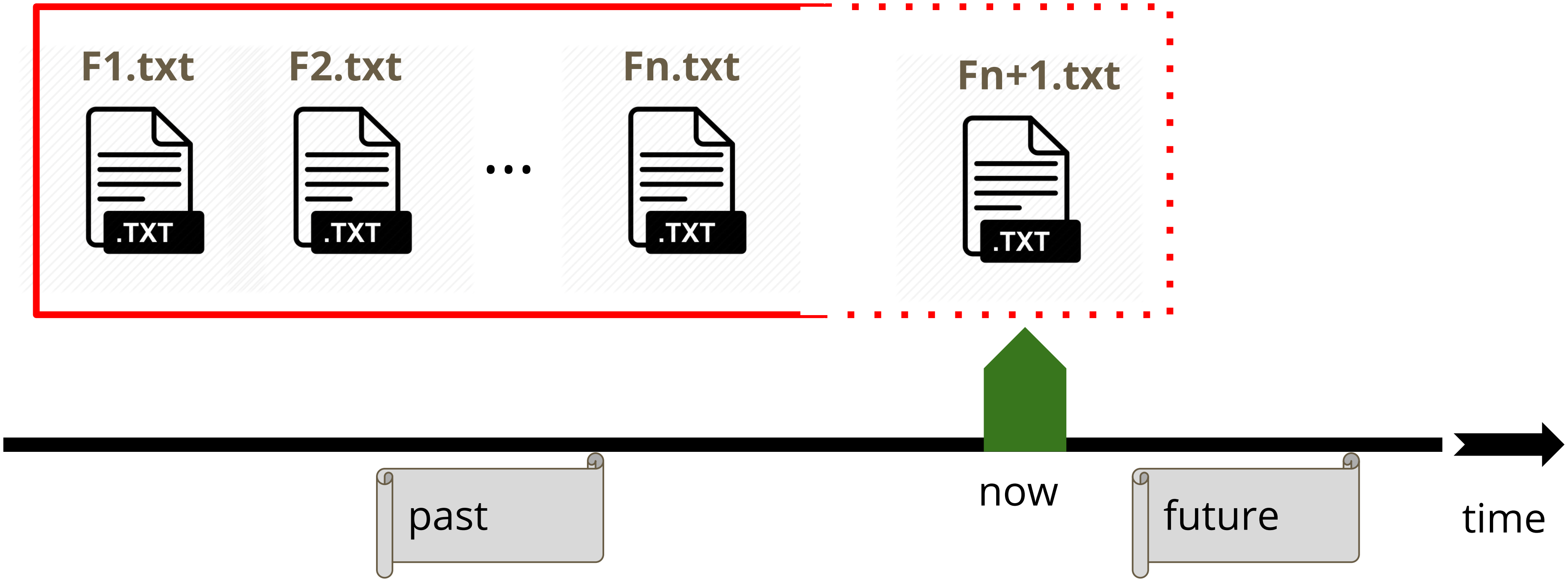
Setting the Context

If you want to see it more graphically,
with a temporal line...



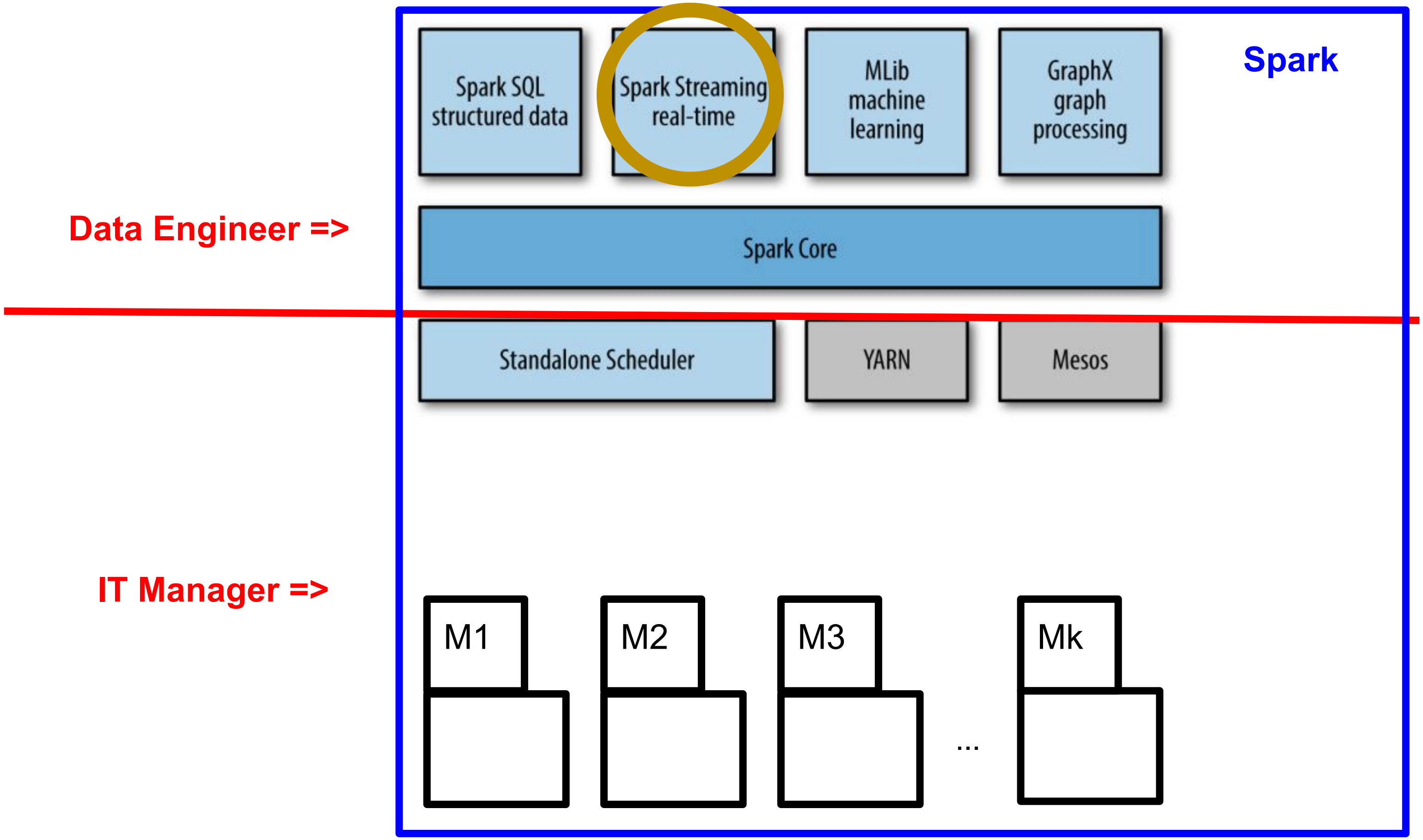
Setting the Context

What if the dataset is still active in the present? *now!*



Setting the Context

In this case we need to use streaming functionality!

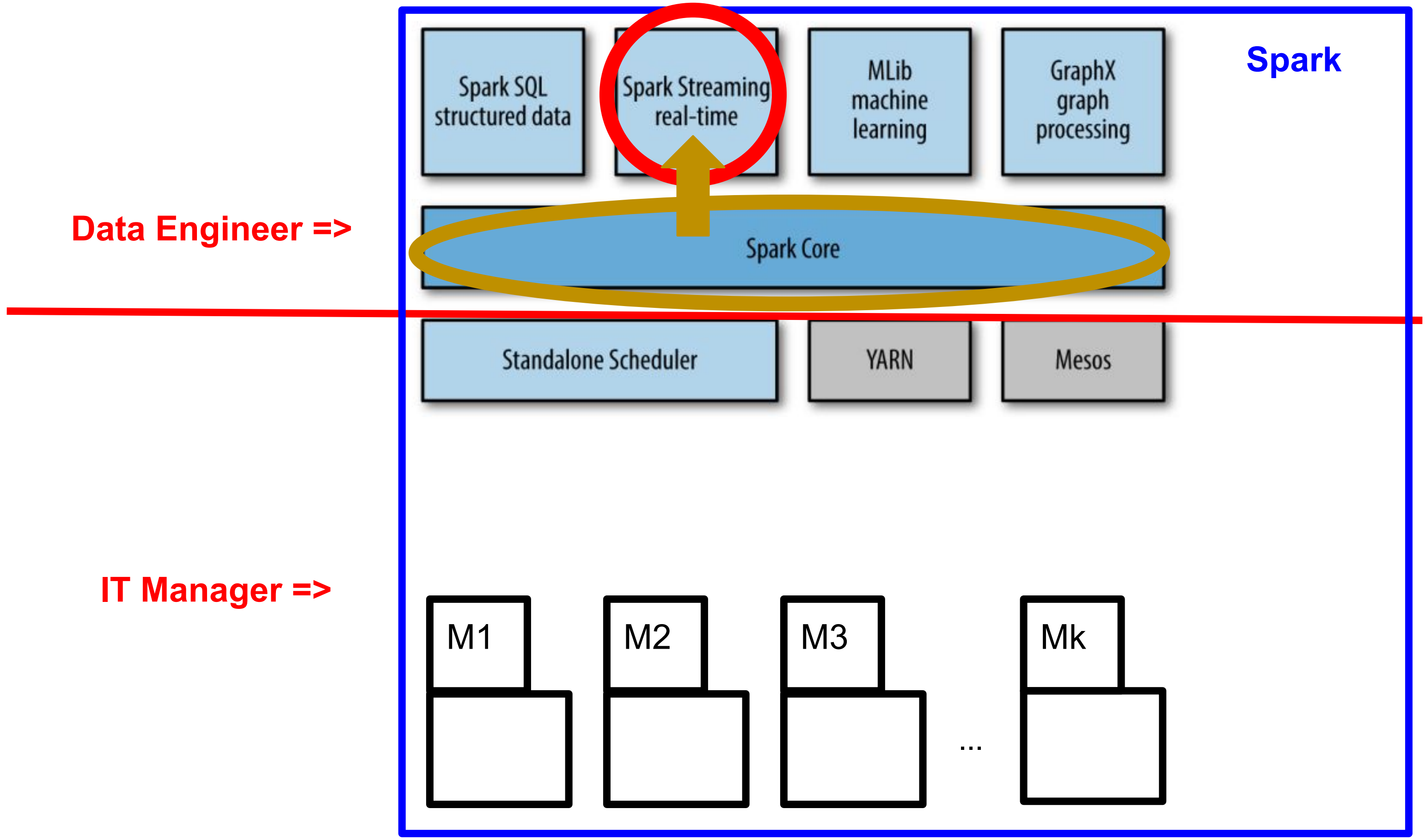


Setting the Context

And that’s the goal of this lecture!

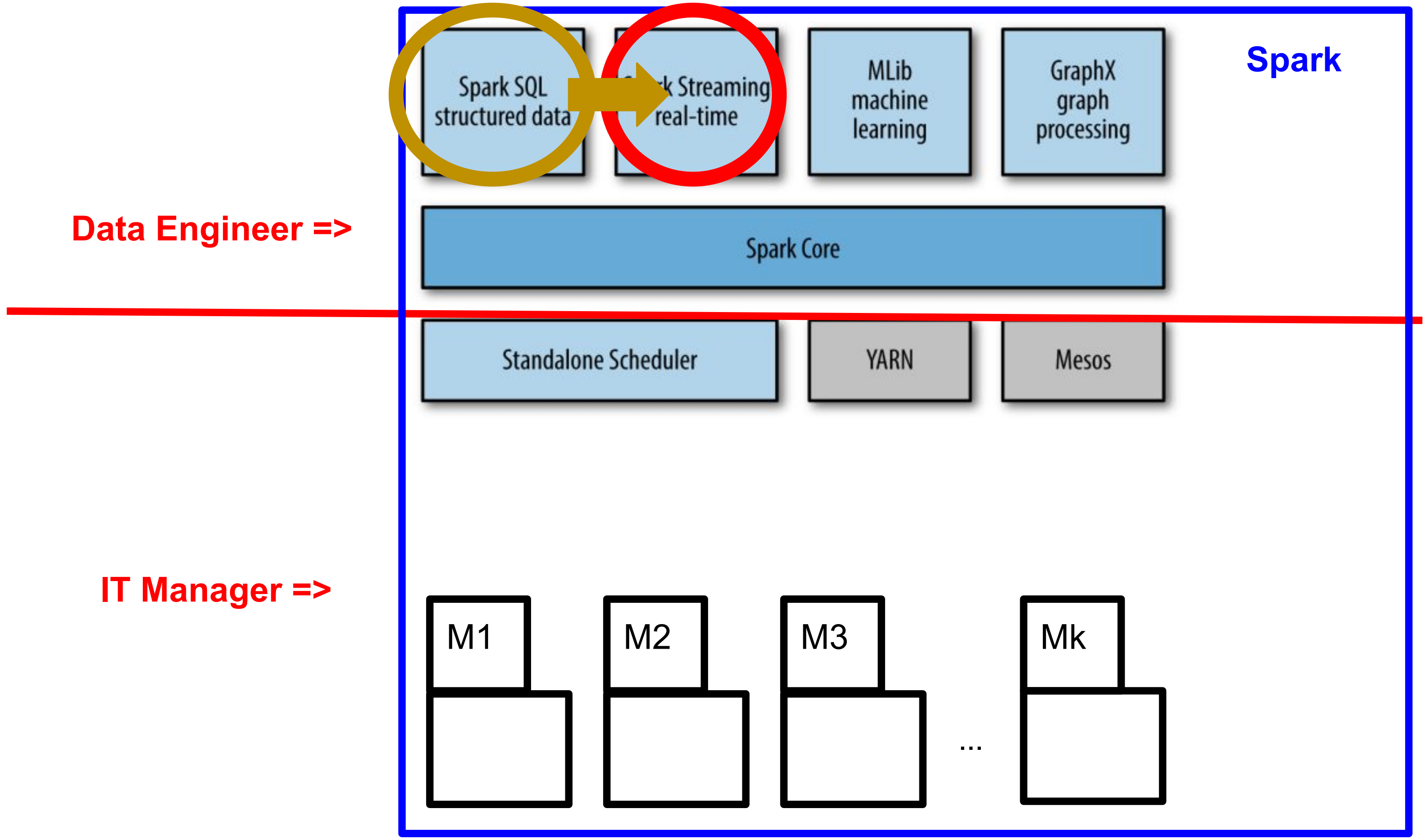
Setting the Context

1. **Enhance our Spark Core techniques with novel streaming functionality (this lecture)!**



Setting the Context

2. **Enhance our Spark SQL techniques with novel streaming functionality (next lecture)!**



Spark SQL structured data

k Streaming real-time

MLib machine learning

GraphX graph processing

Spark

Data Engineer =>

Spark Core

Standalone Scheduler

YARN

Mesos

IT Manager =>

M1

M2

M3

...

Mk

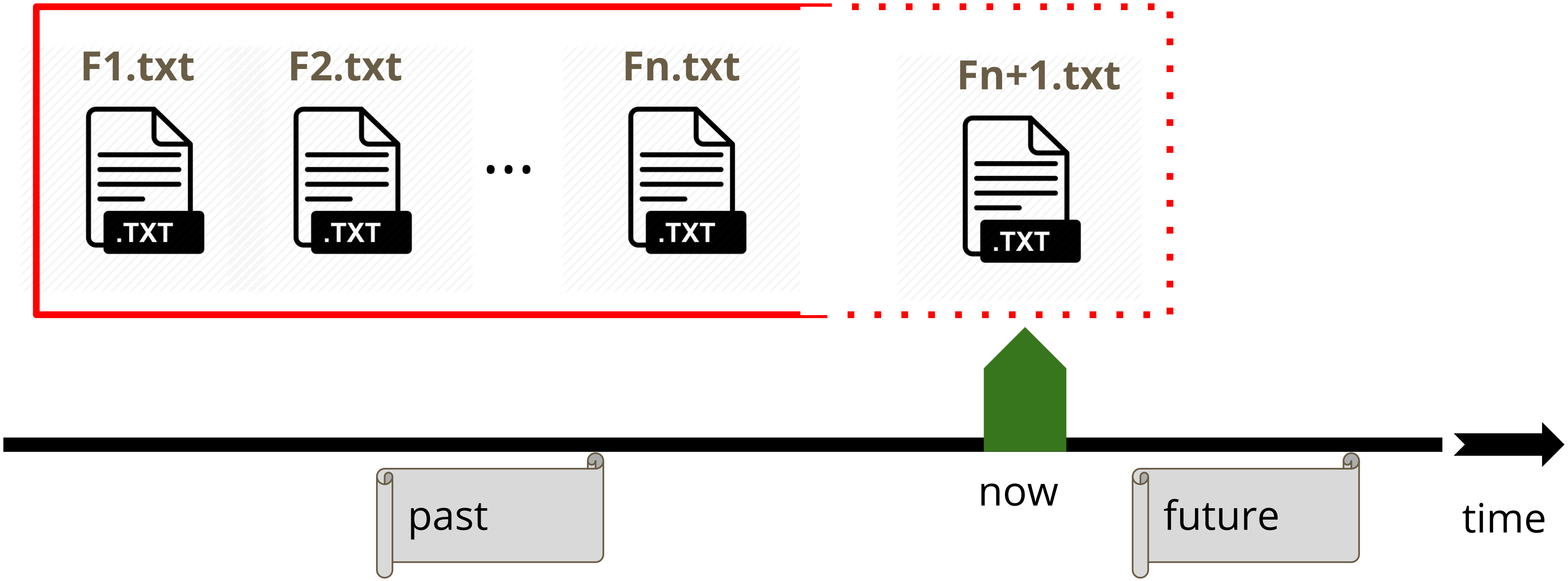
Outline

1. Setting Up the Context.
2. Measurement Unit: Time Interval & Data Batch.
3. From RDDs to a DStream.
4. File Transfer Process.
5. Spark Streaming Context Process.
6. Stateless and Stateful Operations.

Measurement Unit: Time Interval & Data Batch

- When we pose the question…
  - ❖ What if the dataset is still active in the present? *now!*

  we need to be a bit more precise about what do we actually mean.



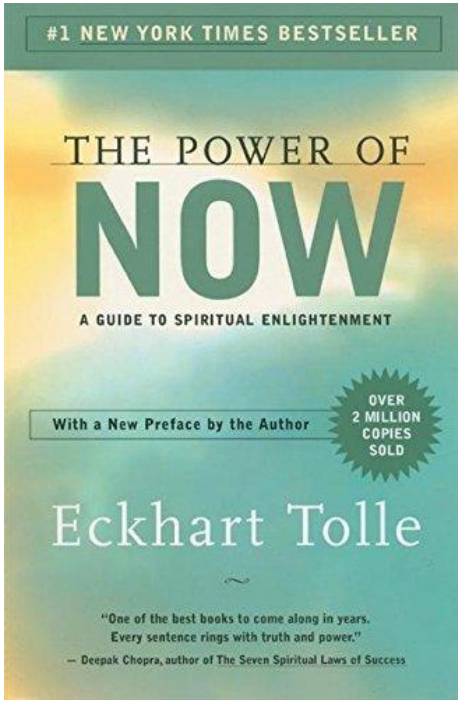
Measurement Unit: Time Interval & Data Batch

Uff, the very word ***now***…
…might be complicated to define :)

Measurement Unit: Time Interval & Data Batch

- For a, let's say, more **spiritual** definition, Eckhart Tolle defines ***now*** as the only thing we have in our life, associating the creation of time (for remembering the past and anticipating the future) to the ego of the mind.

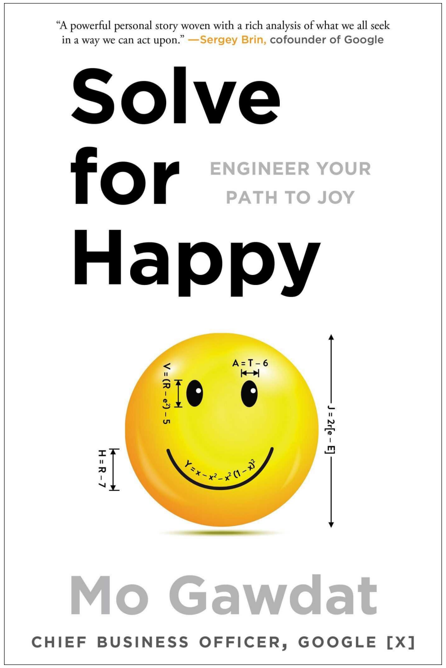
  "Nothing has happened in the past; it happened in the Now. Nothing will ever happen in the future; it will happen in the Now."


#1 NEW YORK TIMES BESTSELLER
THE POWER OF
NOW
A GUIDE TO SPIRITUAL ENLIGHTENMENT
With a New Preface by the Author
OVER 2 MILLION COPIES SOLD
Eckhart Tolle
"One of the best books to come along in years. Every sentence rings with truth and power."
— Deepak Chopra, author of The Seven Spiritual Laws of Success

Measurement Unit: Time Interval & Data Batch

- For a, let’s say, more **scientific** definition, Mo Gawdat revises the Newtonian physics and Einstein relativity theory of space-time to present a timeless experiment, in order to conclude that mechanical time is just a human construct, an illusion.

  “Time isn’t moving; you’re the one who is moving through time.”



“A powerful personal story woven with a rich analysis of what we all seek in a way we can act upon.” —Sergey Brin, cofounder of Google

Solve for Happy

ENGINEER YOUR PATH TO JOY

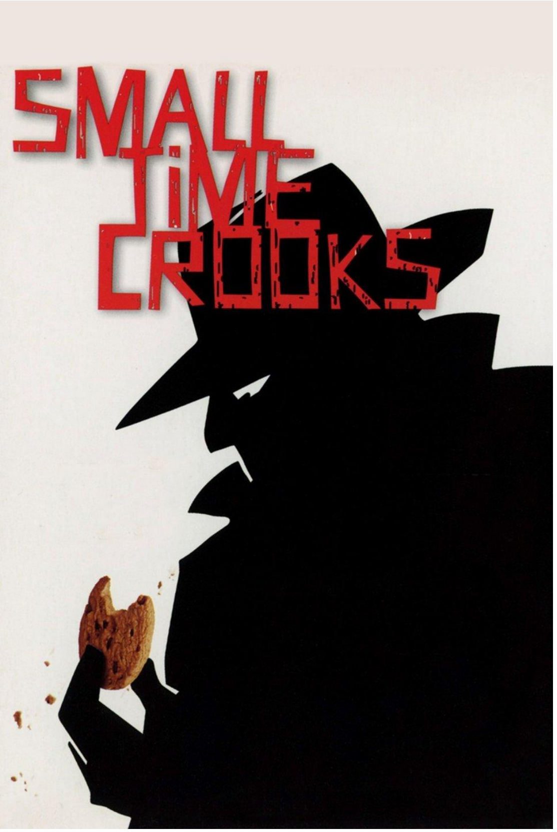
Mo Gawdat

CHIEF BUSINESS OFFICER, GOOGLE [X]

Measurement Unit: Time Interval & Data Batch

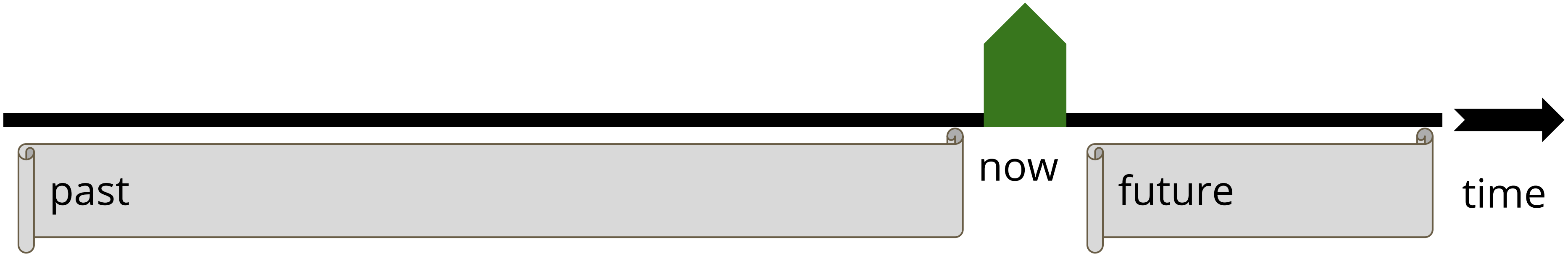
- For a, let’s say, more **artistic** definition, Woody Allen relates being present in the moment with the confidence we humans can develop in ourselves.

  “80% of success in life is just showing up.”



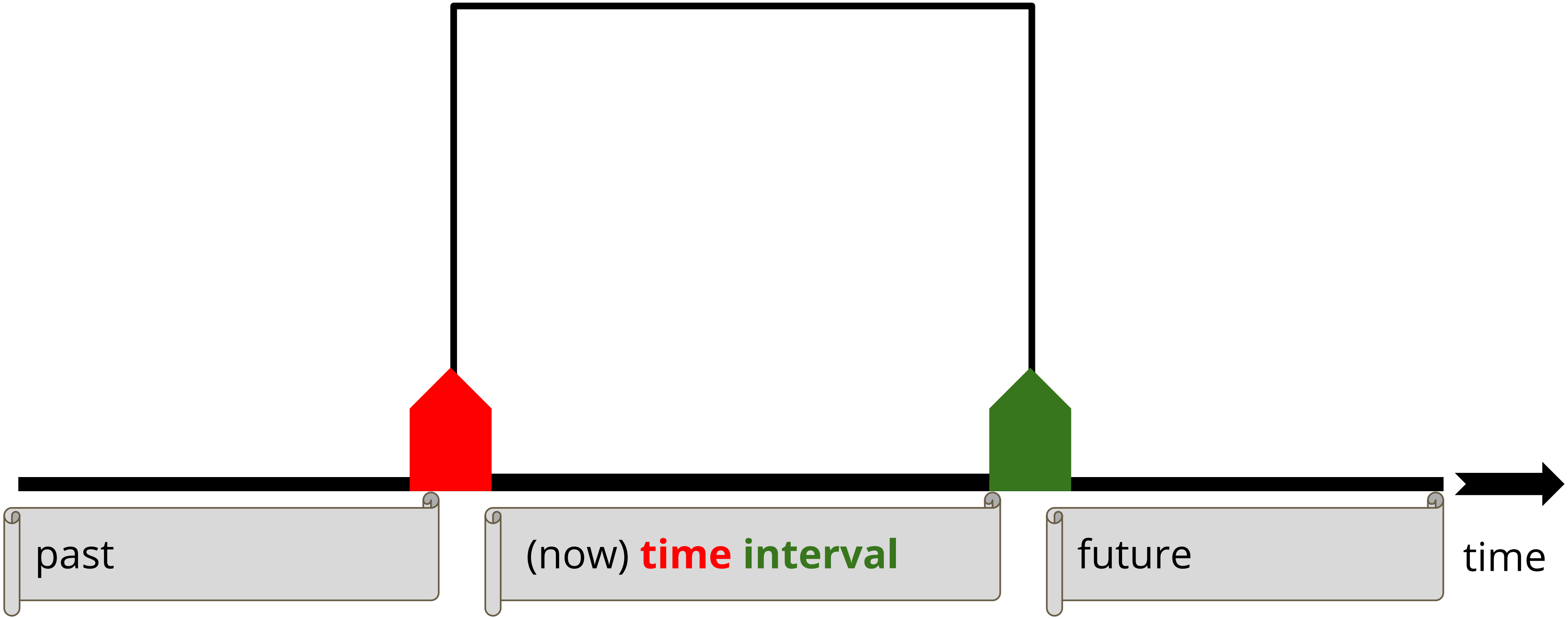
Measurement Unit: Time Interval & Data Batch

- In the case of Spark Streaming, when we talk about ***now***...



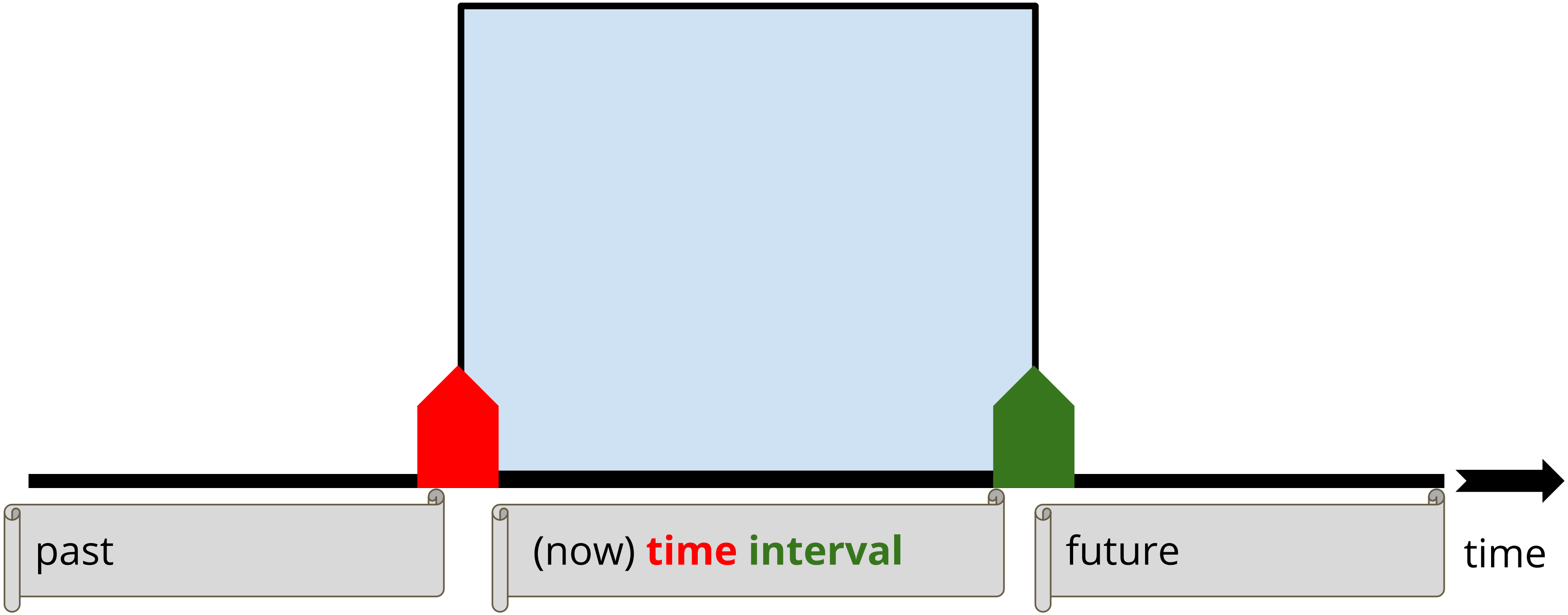
Measurement Unit: Time Interval & Data Batch

- In the case of Spark Streaming, when we talk about ***now***…
  …we really mean a **time interval**, measured in the mathematical way:
  - ❖ Starting at time t_{lb}
  - ❖ Finishing at time t_{ub}



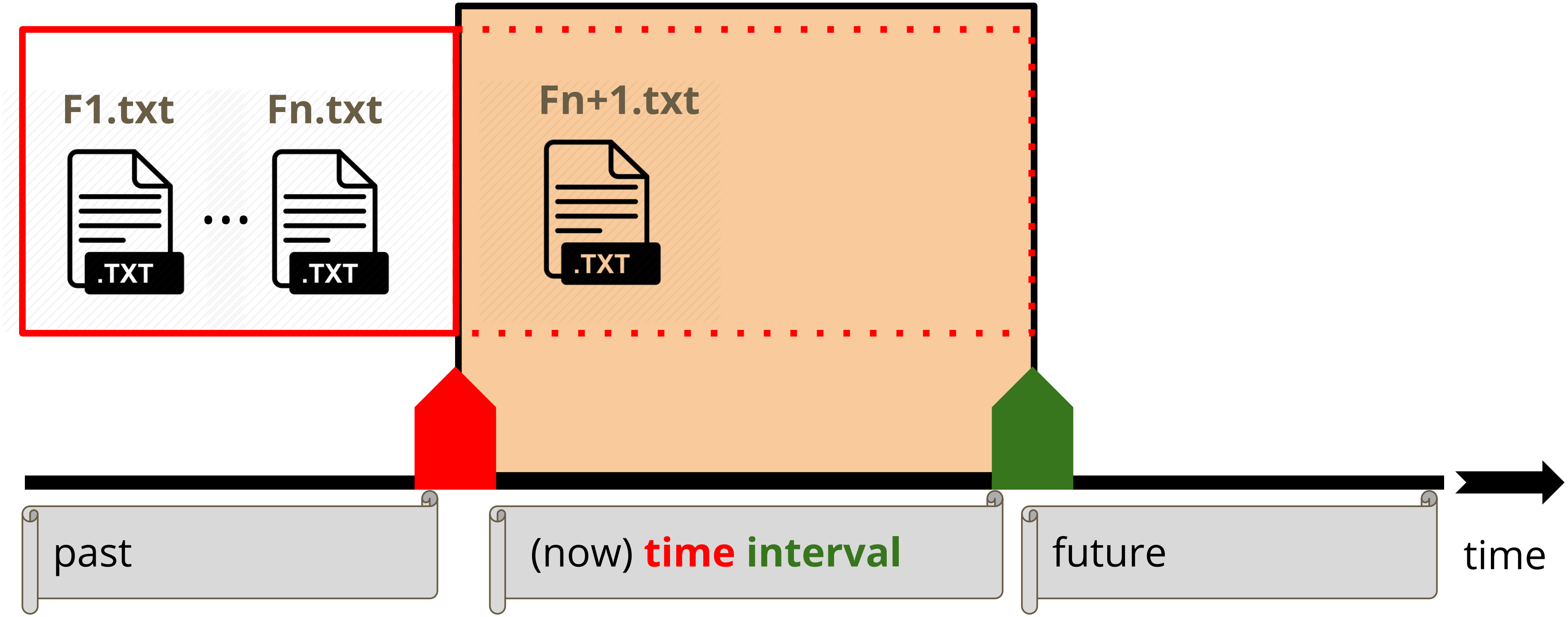
Measurement Unit: Time Interval & Data Batch

- This **time interval** creates room for things to happen!



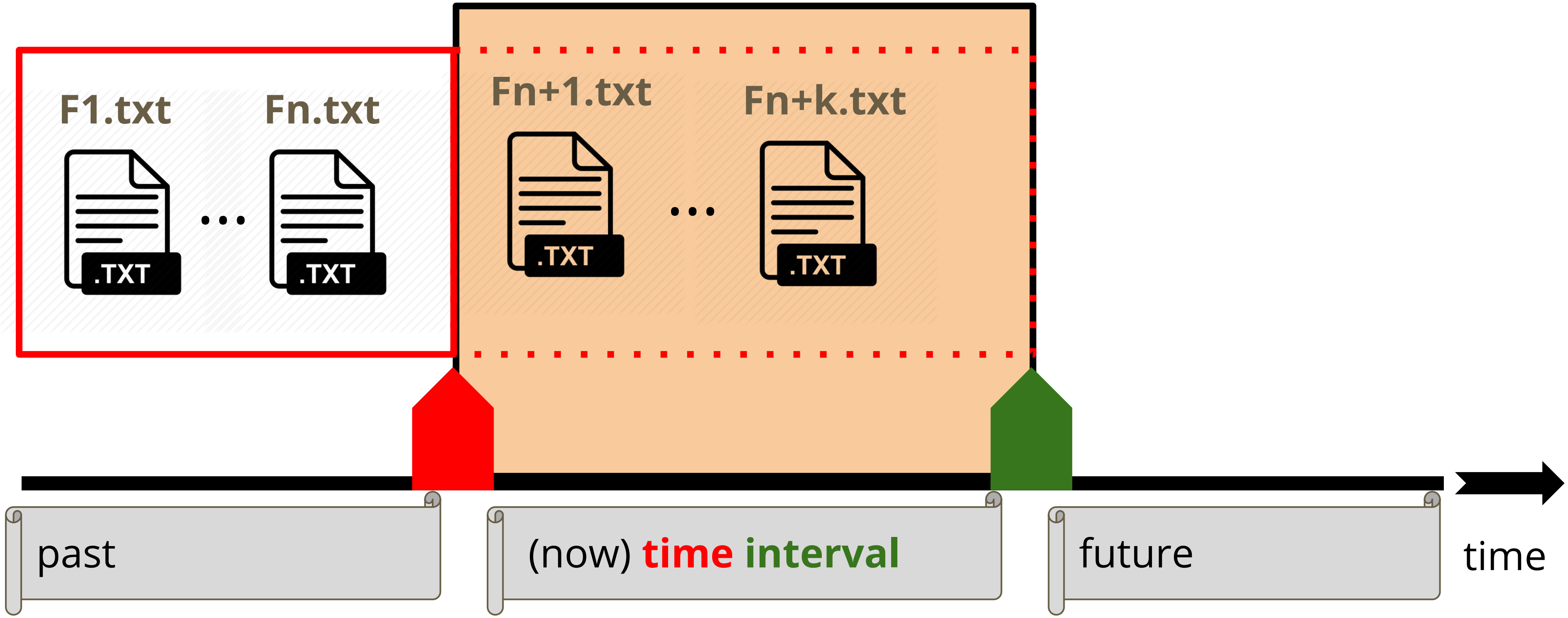
Measurement Unit: Time Interval & Data Batch

- This **time interval** creates room for things to happen!
  - For example, one new text file of our dynamic dataset to arrive!



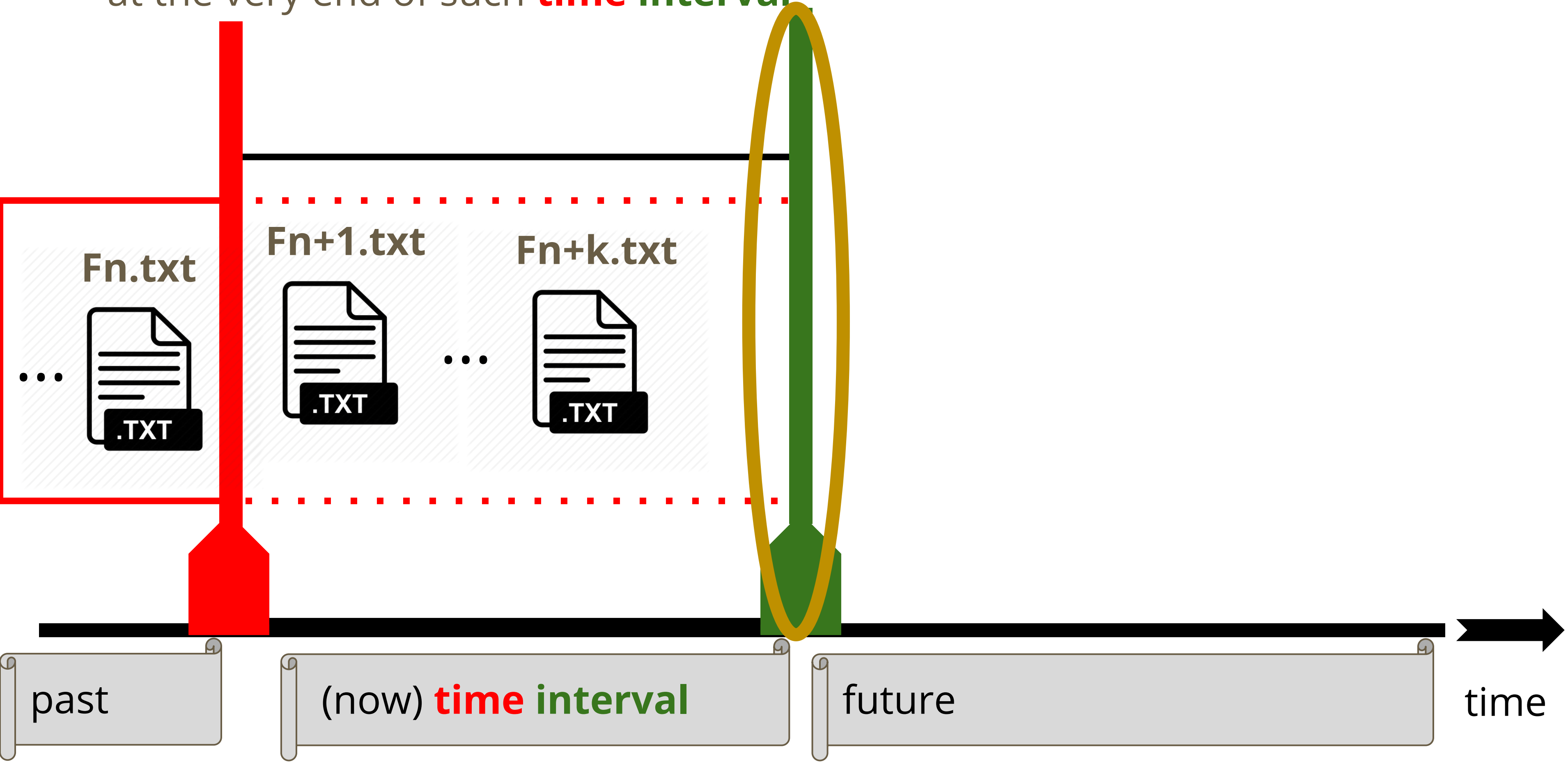
Measurement Unit: Time Interval & Data Batch

- This **time interval** creates room for things to happen!
  - Or, perhaps multiple new files to arrive!



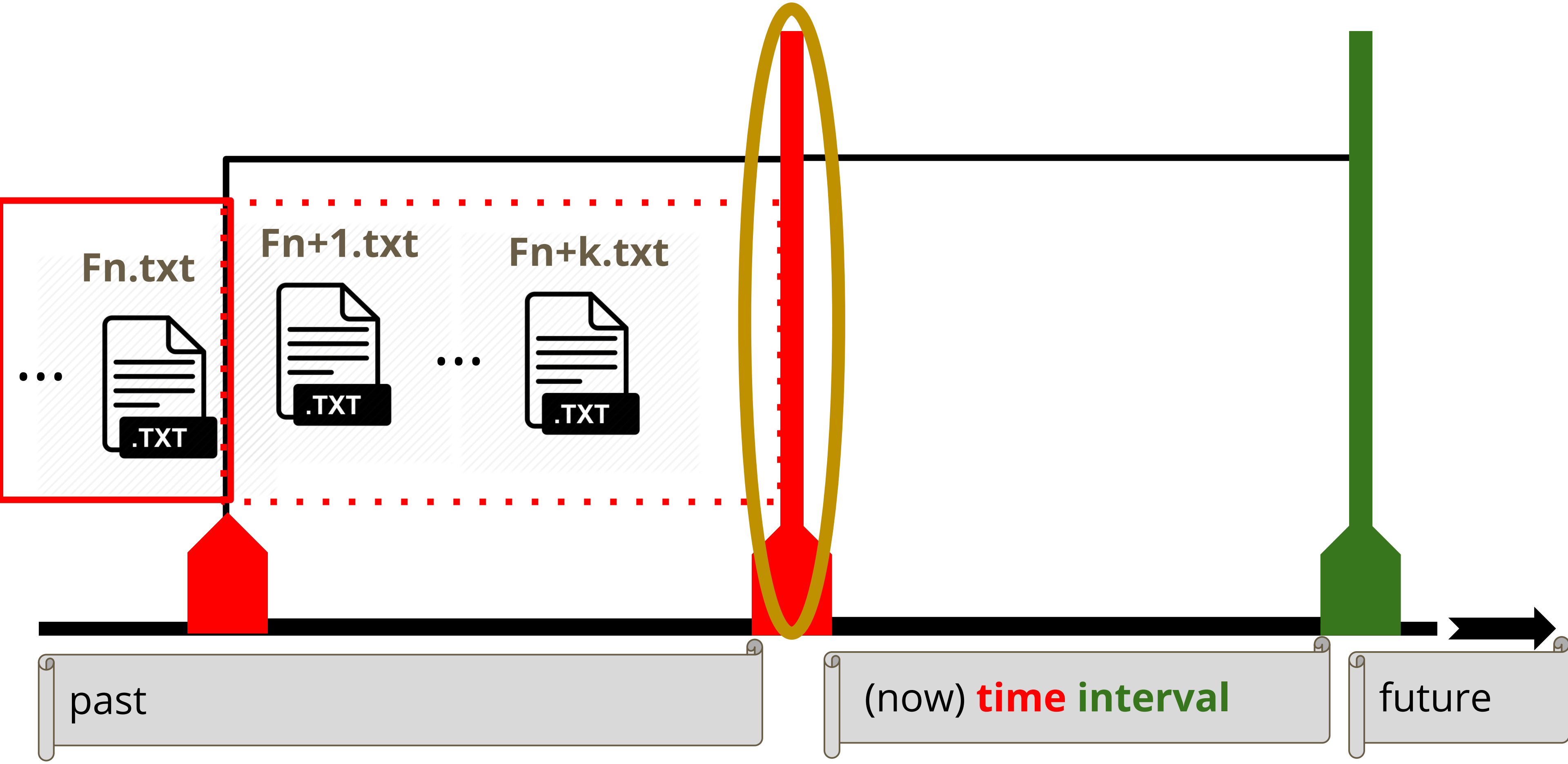
Measurement Unit: Time Interval & Data Batch

- Spark will only check for new files arriving during our current **time interval** at the very end of such **time interval**.



Measurement Unit: Time Interval & Data Batch

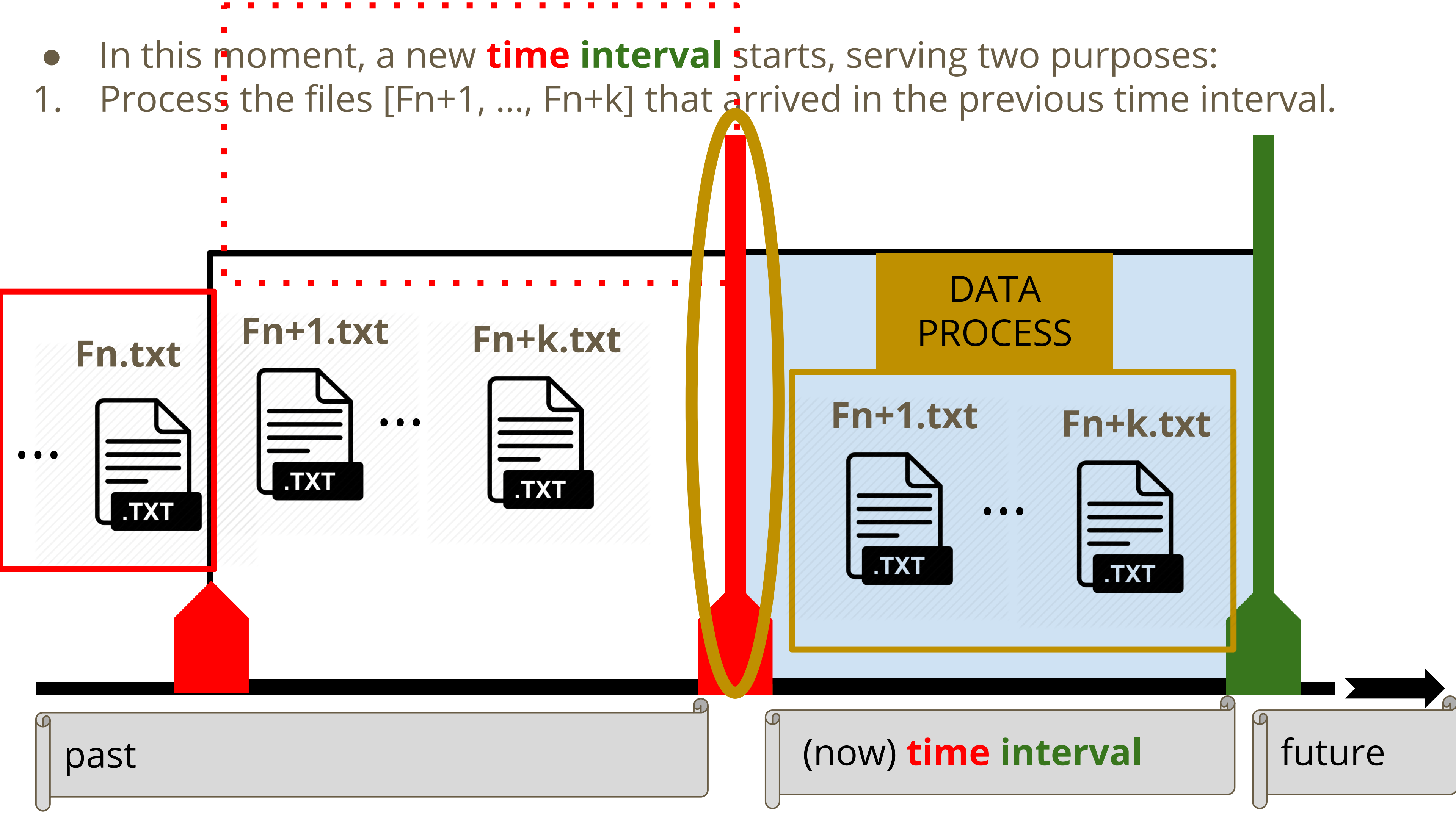
- In this moment, a new **time interval** starts, serving two purposes:



Measurement Unit: Time Interval & Data Batch

- In this moment, a new **time interval** starts, serving two purposes:

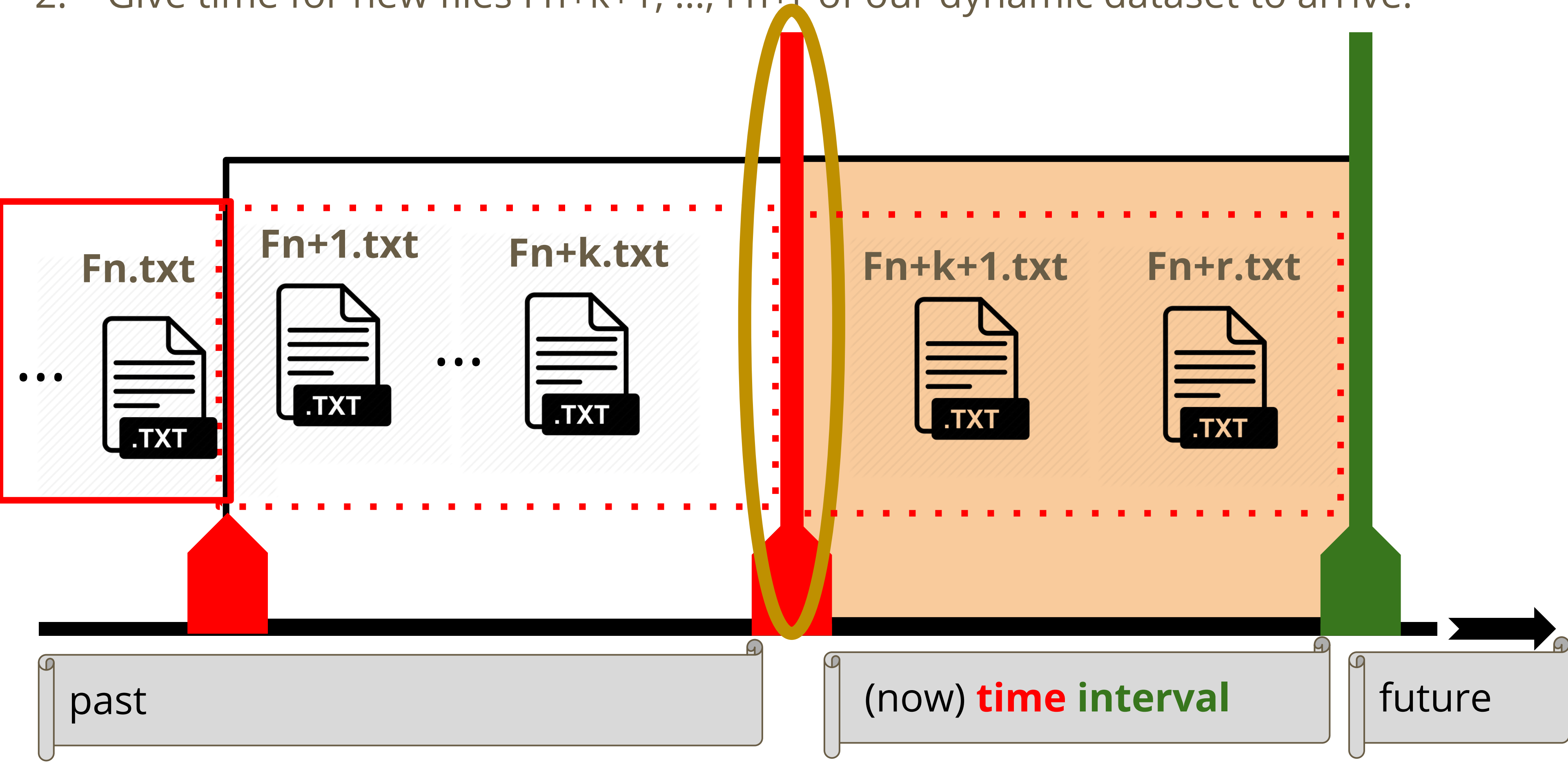
1. Process the files [Fn+1, ..., Fn+k] that arrived in the previous time interval.



Measurement Unit: Time Interval & Data Batch

- In this moment, a new **time interval** starts, serving two purposes:

2. Give time for new files Fn+k+1, ..., Fn+r of our dynamic dataset to arrive!

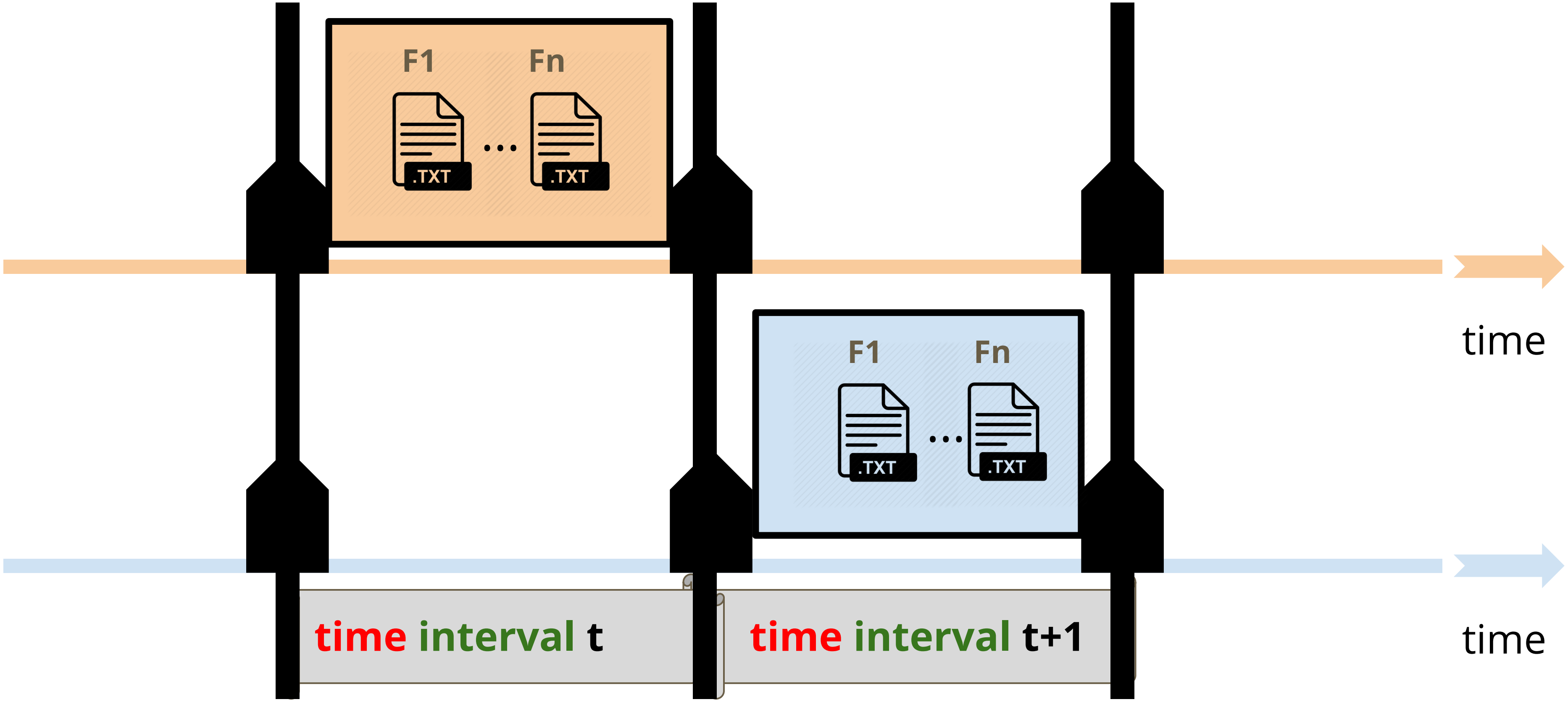


Measurement Unit: Time Interval & Data Batch

This process making:

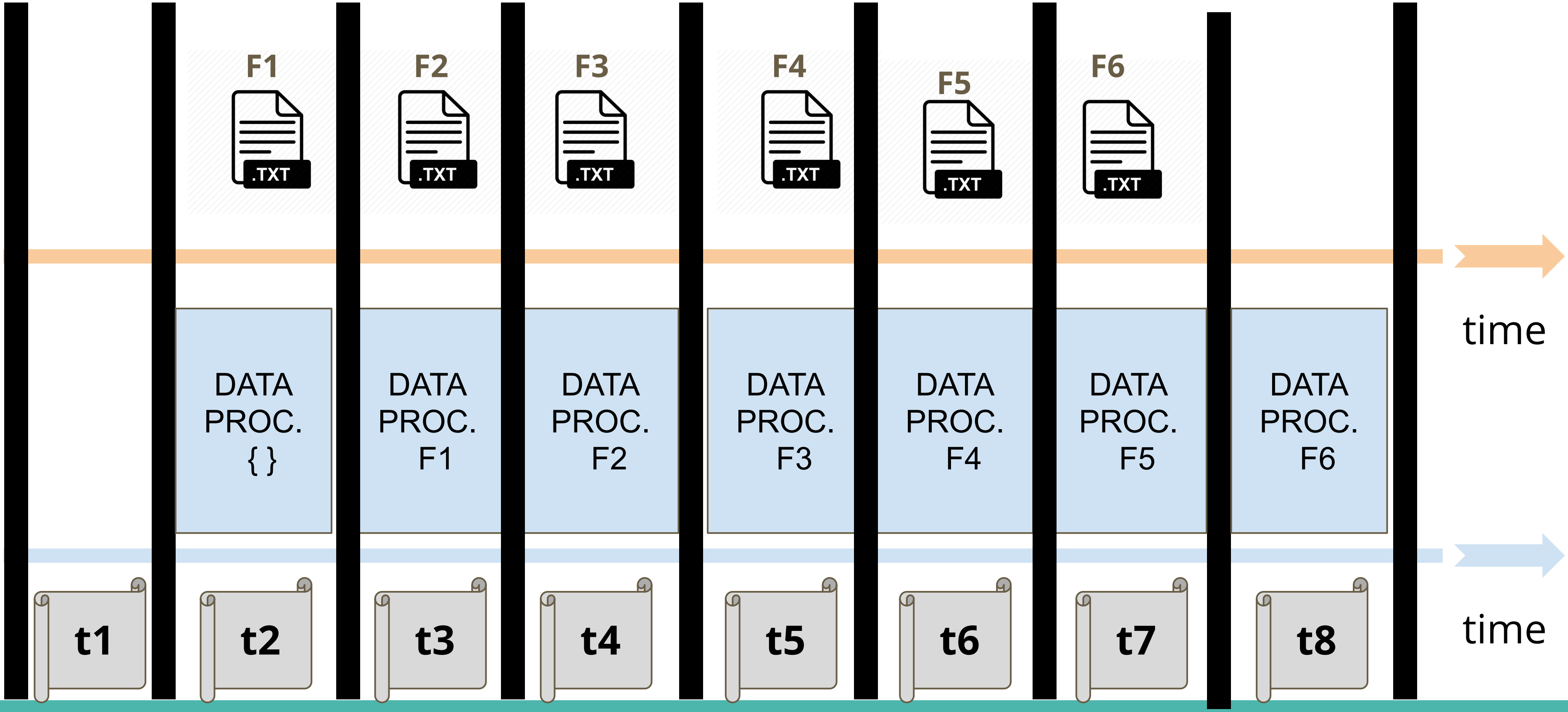
Files arriving at **time interval t**

To be processed at **time interval t+1**



Measurement Unit: Time Interval & Data Batch

Is repeated over and over during the entire execution of our Spark Streaming program!



Outline

1. Setting Up the Context.
2. Measurement Unit: Time Interval & Data Batch.
3. From RDDs to a DStream.
4. File Transfer Process.
5. Spark Streaming Context Process.
6. Stateless and Stateful Operations.

From RDDs to a DStream

One of my best professors stated once…

“When learning a new topic, let’s approach the unknown concepts via known concepts”.

From RDDs to a DStream

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 Streaming** via **Spark Core**.

From RDDs to a DStream

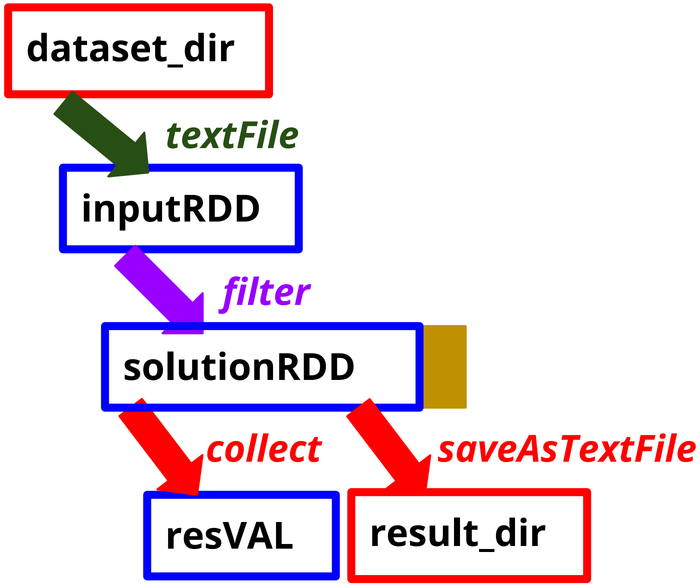
The example **p01_introRDD.py** is based in Spark Core.

From RDDs to a DStream

p01_introRDD.py

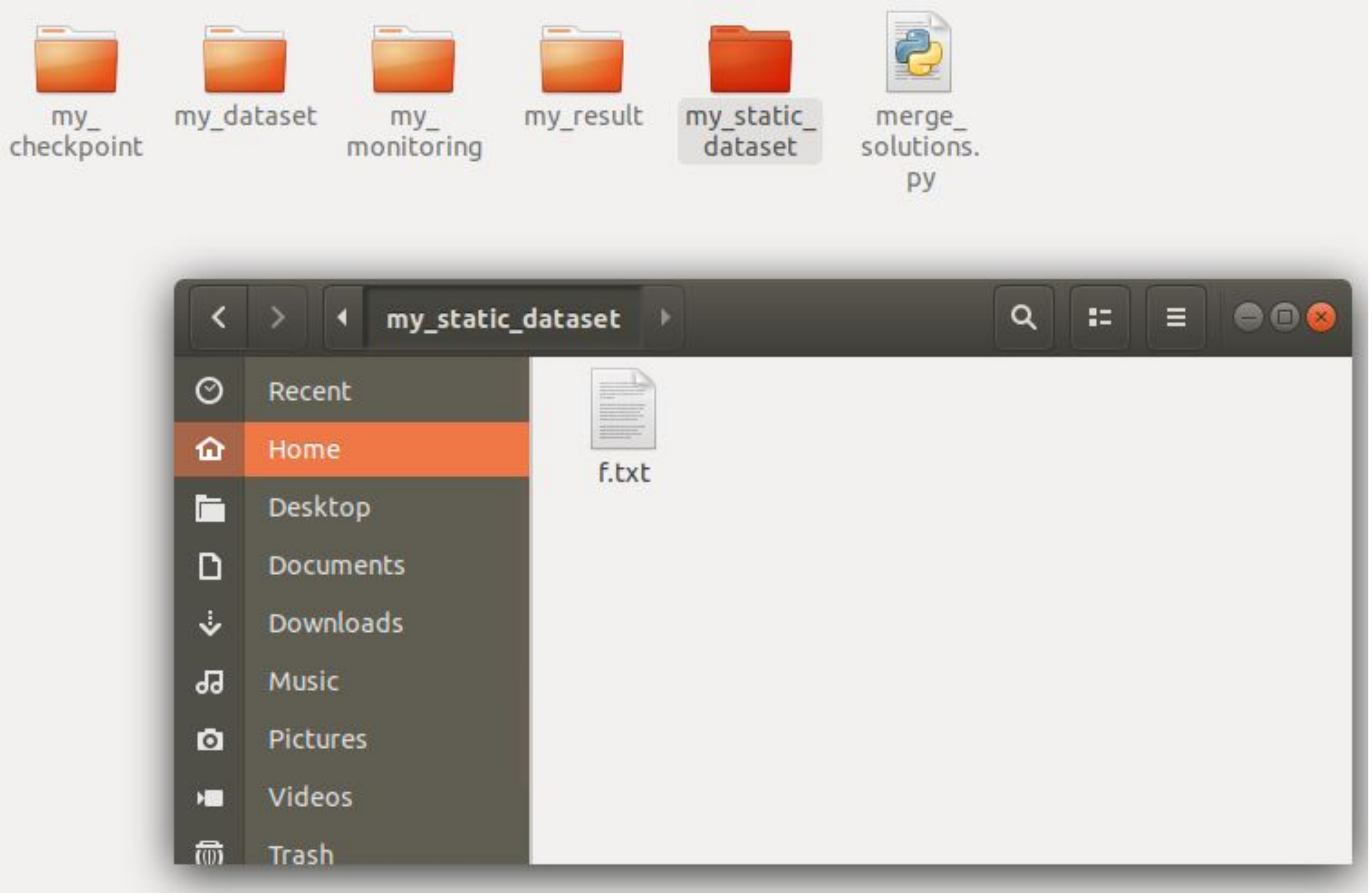
1. Read in all the lines of my_dataset_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Collect them and print them by the screen.
5. Save them to the new directory my_result_dir.

A high level view of its operations is presented next:



From RDDs to a DStream

Let's assume dataset_dir contains just 1 file [F.txt] with 18 lines



From RDDs to a DStream

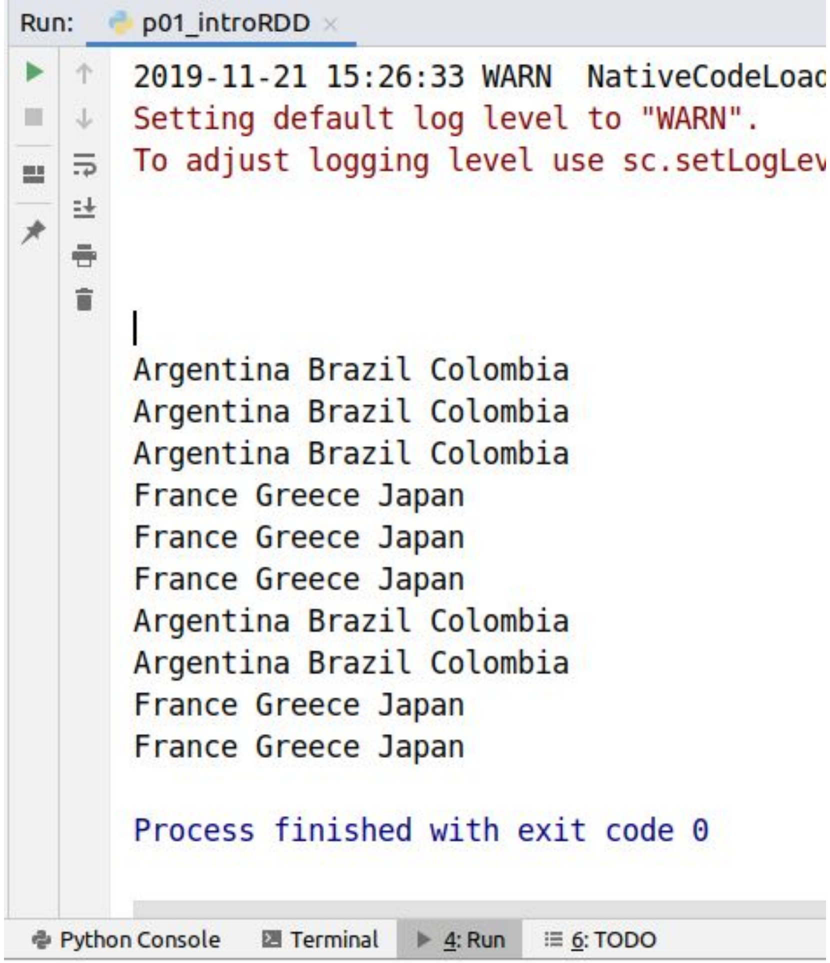
Let's assume dataset_dir contains just 1 file [F.txt] with 18 lines

(Line 01)	Argentina Brazil Colombia\n
(Line 02)	Argentina Brazil Colombia\n
(Line 03)	Argentina Brazil Colombia\n
(Line 04)	Denmark Egypt\n
(Line 05)	Denmark Egypt\n
(Line 06)	Denmark Egypt\n
(Line 07)	France Greece Japan\n
(Line 08)	France Greece Japan\n
(Line 09)	France Greece Japan\n
(Line 10)	Argentina Brazil Colombia\n
(Line 11)	\n
(Line 12)	Argentina Brazil Colombia\n
(Line 13)	Denmark Egypt\n
(Line 14)	\n
(Line 15)	Denmark Egypt\n
(Line 16)	France Greece Japan\n
(Line 17)	\n
(Line 18)	France Greece Japan\n

F.txt

From RDDs to a DStream

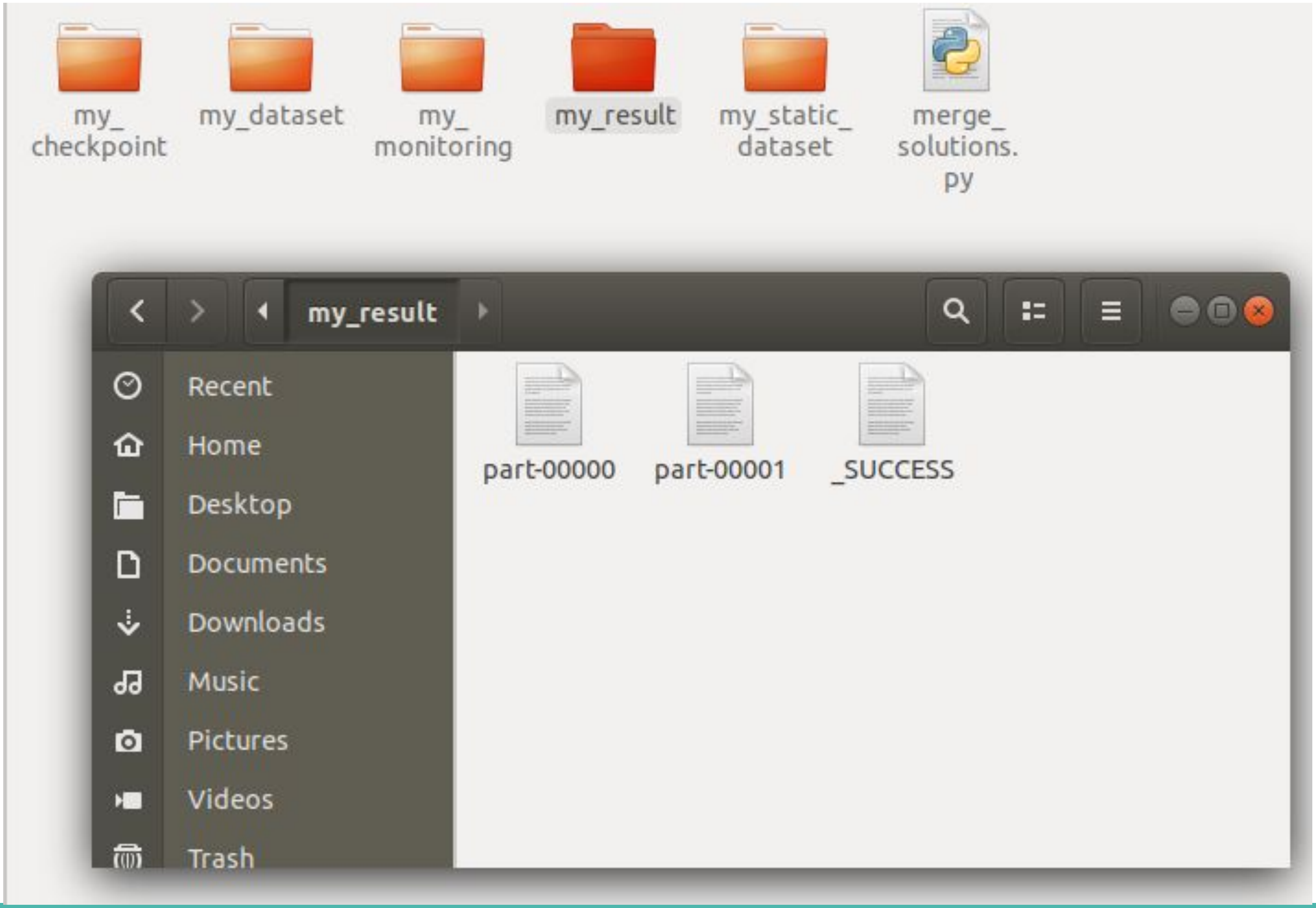
If we run the Spark Core Application filtering the lines of at least 15 characters then the following lines will be collected and printed:



```
2019-11-21 15:26:33 WARN  NativeCodeLoac
Setting default log level to "WARN".
To adjust logging level use sc.setLogLev



Argentina Brazil Colombia
Argentina Brazil Colombia
Argentina Brazil Colombia
France Greece Japan
France Greece Japan
France Greece Japan
Argentina Brazil Colombia
Argentina Brazil Colombia
France Greece Japan
France Greece Japan

Process finished with exit code 0
```

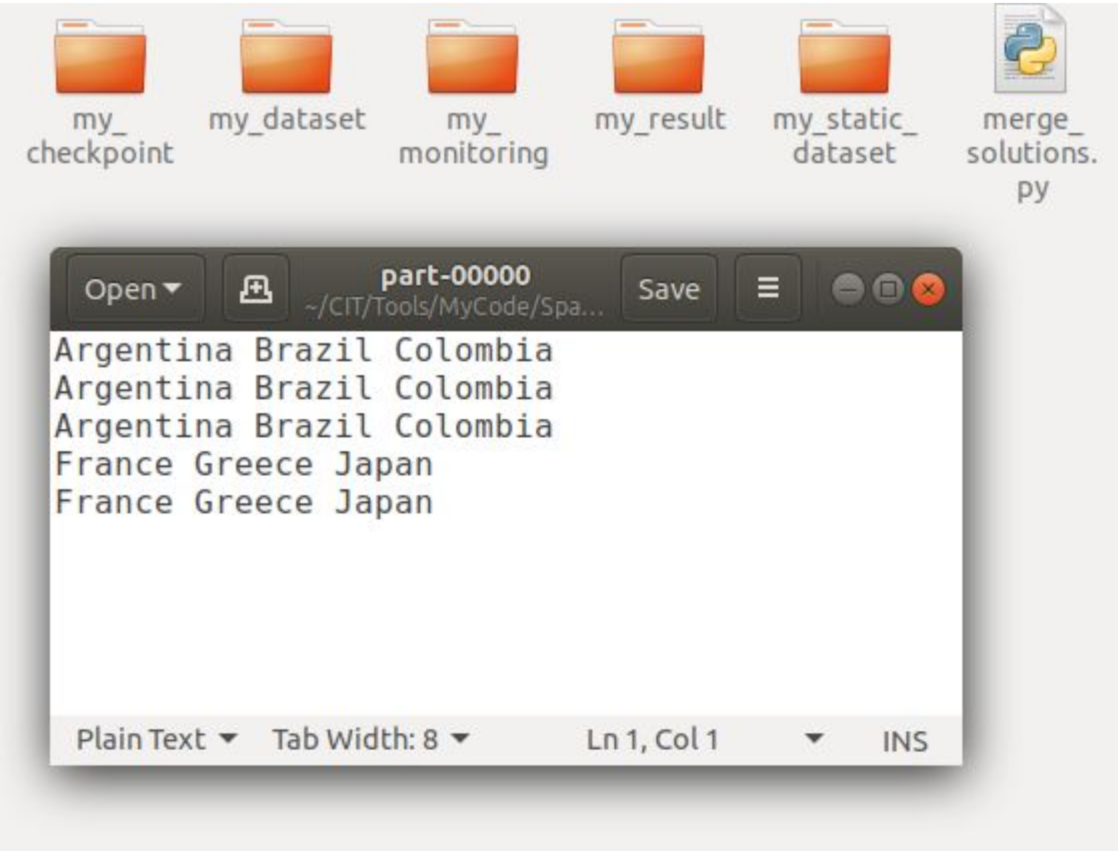
From RDDs to a DStream

If we run the Spark Core Application filtering the lines of at least 15 characters then the following result_dir will be generated:



From RDDs to a DStream

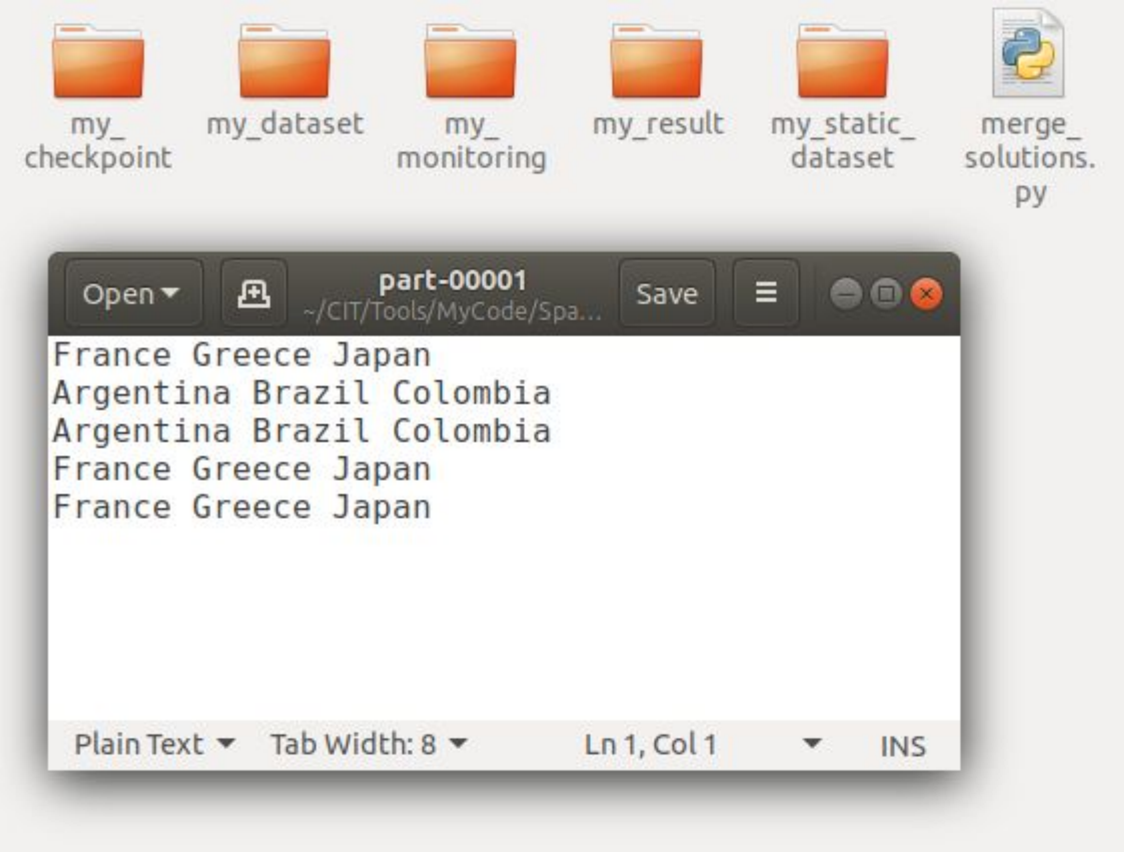
If we run the Spark Core Application filtering the lines of at least 15 characters then the following result_dir will be generated:



```
Argentina Brazil Colombia
Argentina Brazil Colombia
Argentina Brazil Colombia
France Greece Japan
France Greece Japan
```

From RDDs to a DStream

If we run the Spark Core Application filtering the lines of at least 15 characters then the following result_dir will be generated:



```
France Greece Japan
Argentina Brazil Colombia
Argentina Brazil Colombia
France Greece Japan
France Greece Japan
```

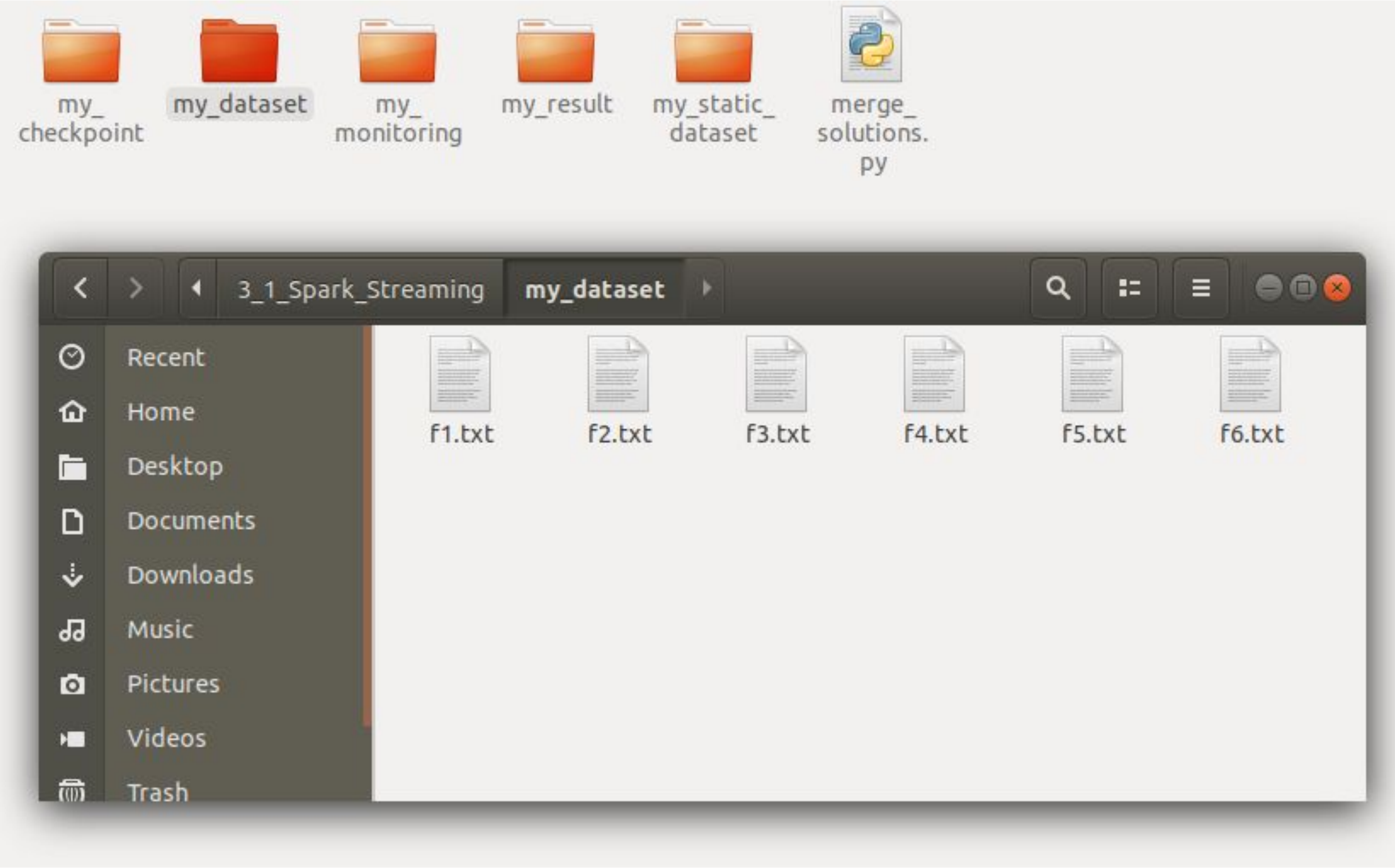
From RDDs to a DStream

Now…

the example **p02_introDStream.py** is based in Spark Streaming.

From RDDs to a DStream

Let's assume we split **dataset_dir** into 6 files [F1.txt, ..., F6.txt] with 3 lines each:



From RDDs to a DStream

Let's assume we split dataset_dir into 6 files [F1.txt, …, F6.txt] with 3 lines each:

(Line 01)	Argentina Brazil Colombia\n
(Line 02)	Argentina Brazil Colombia\n
(Line 03)	Argentina Brazil Colombia\n
(Line 01)	Denmark Egypt\n
(Line 02)	Denmark Egypt\n
(Line 03)	Denmark Egypt\n
(Line 01)	France Greece Japan\n
(Line 02)	France Greece Japan\n
(Line 03)	France Greece Japan\n
(Line 01)	Argentina Brazil Colombia\n
(Line 02)	\n
(Line 03)	Argentina Brazil Colombia\n
(Line 01)	Denmark Egypt\n
(Line 02)	\n
(Line 03)	Denmark Egypt\n
(Line 01)	France Greece Japan\n
(Line 02)	\n
(Line 03)	France Greece Japan\n

F1.txt

.TXT

F2.txt

.TXT

F3.txt

.TXT

F4.txt

.TXT

F5.txt

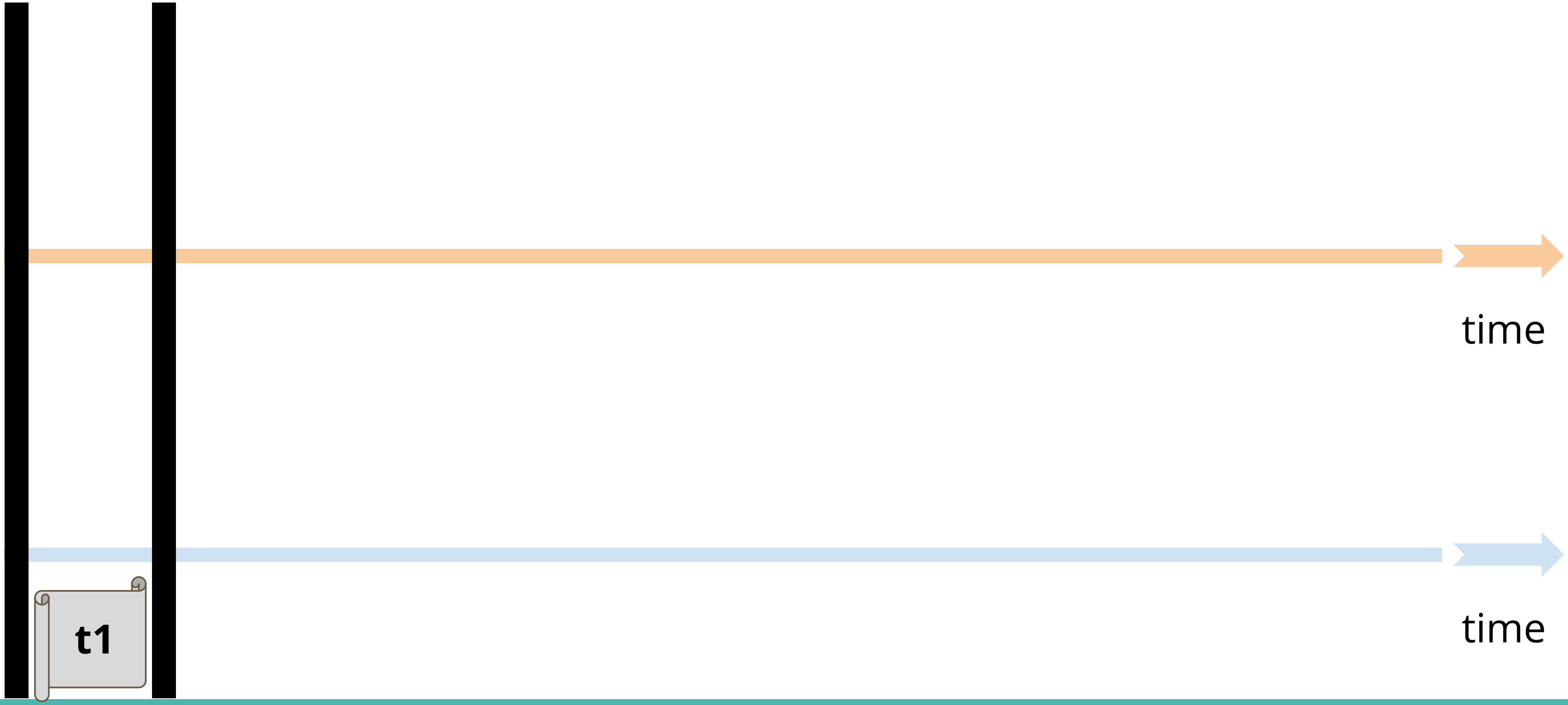
.TXT

F6.txt

.TXT

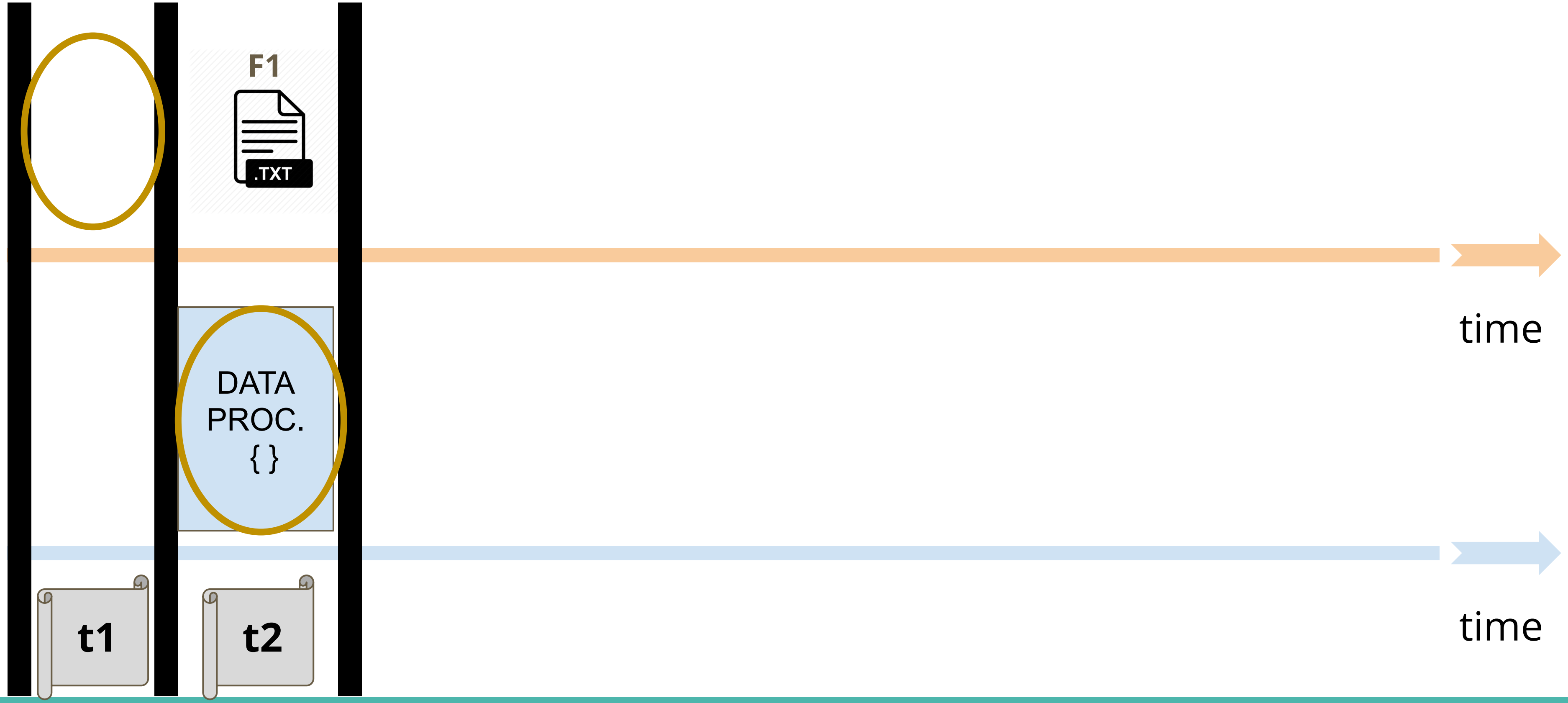
From RDDs to a DStream

Let's assume we simulate the files [F1.txt, ..., F6.txt] to arrive in streaming, setting the time intervals to be of 3 seconds.



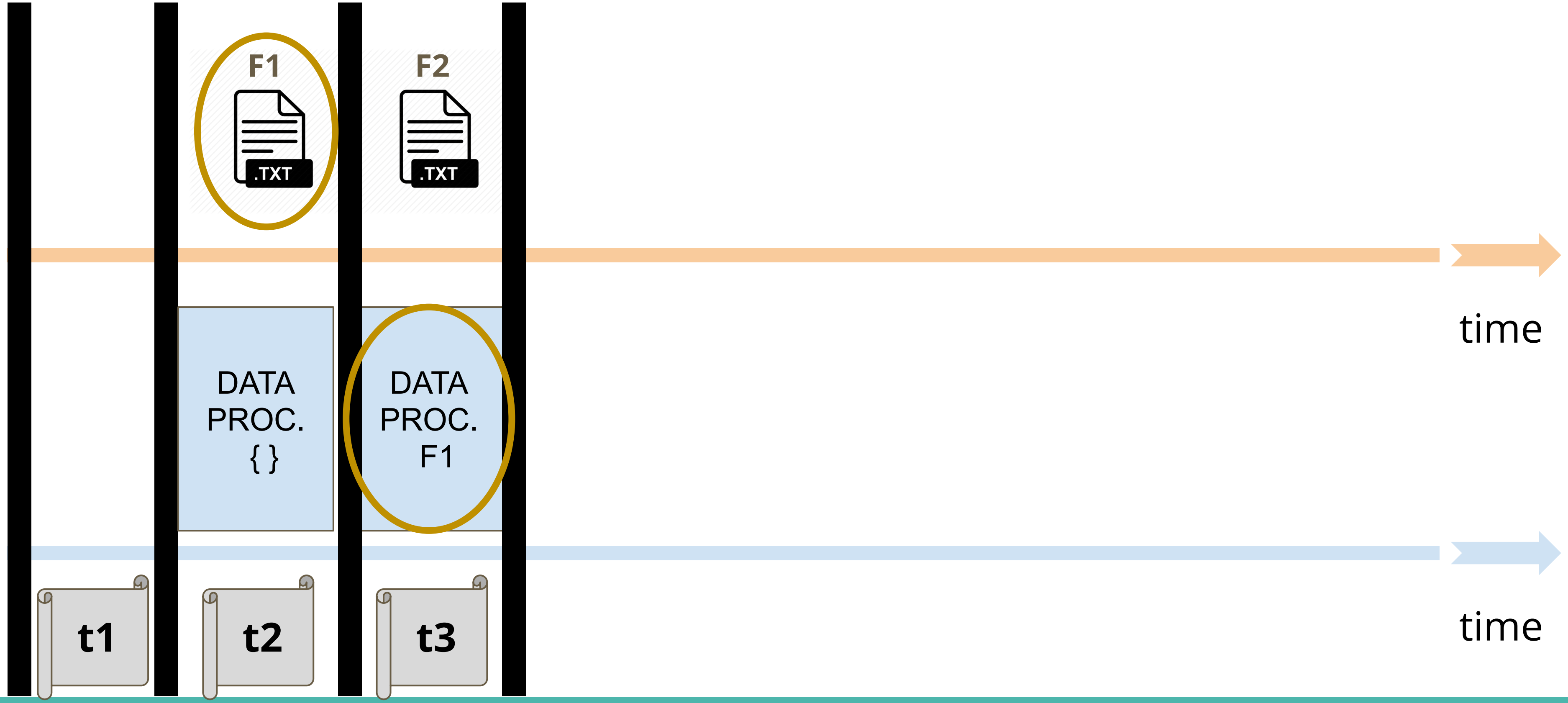
From RDDs to a DStream

Let’s assume we simulate the files [F1.txt, …, F6.txt] to arrive in streaming, setting the time intervals to be of 3 seconds.



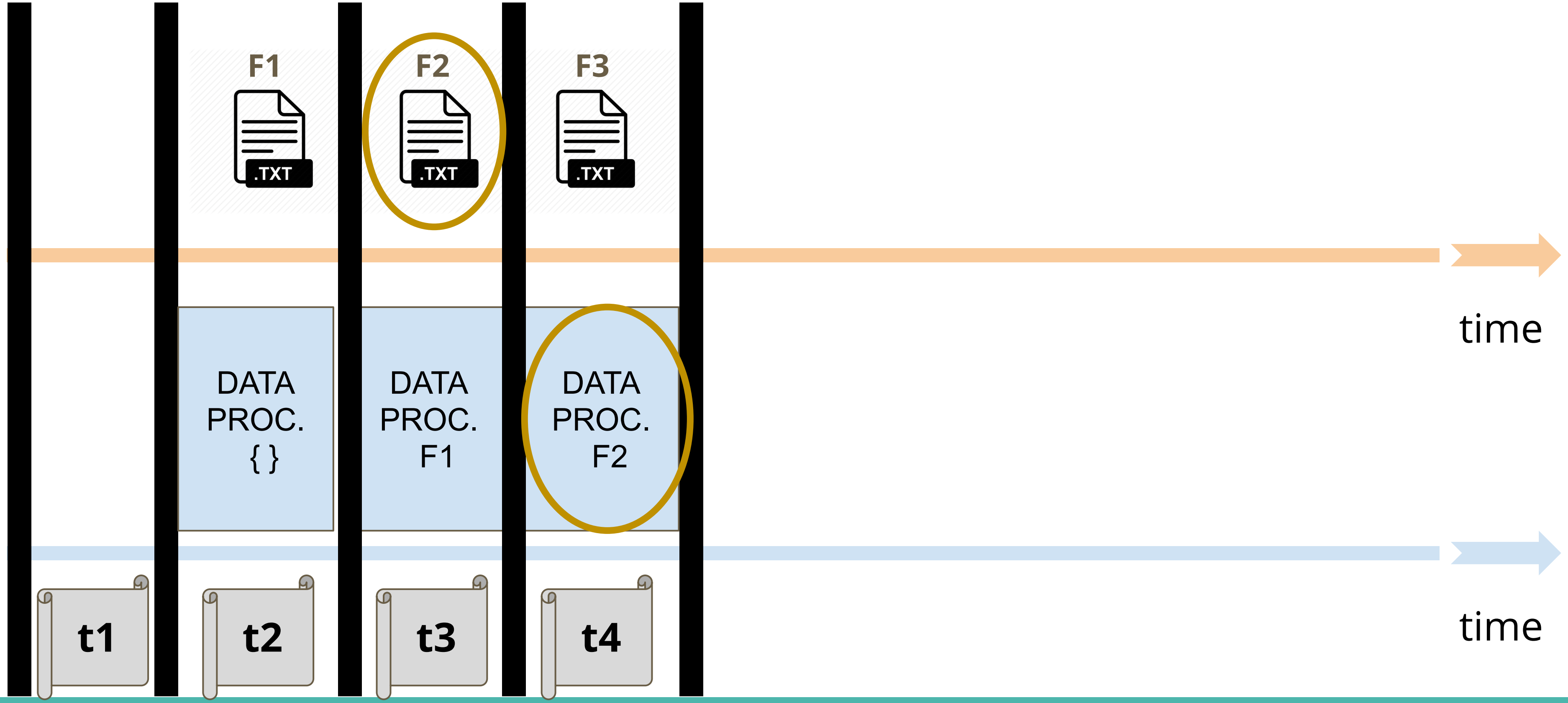
From RDDs to a DStream

Let’s assume we simulate the files [F1.txt, ..., F6.txt] to arrive in streaming, setting the time intervals to be of 3 seconds.



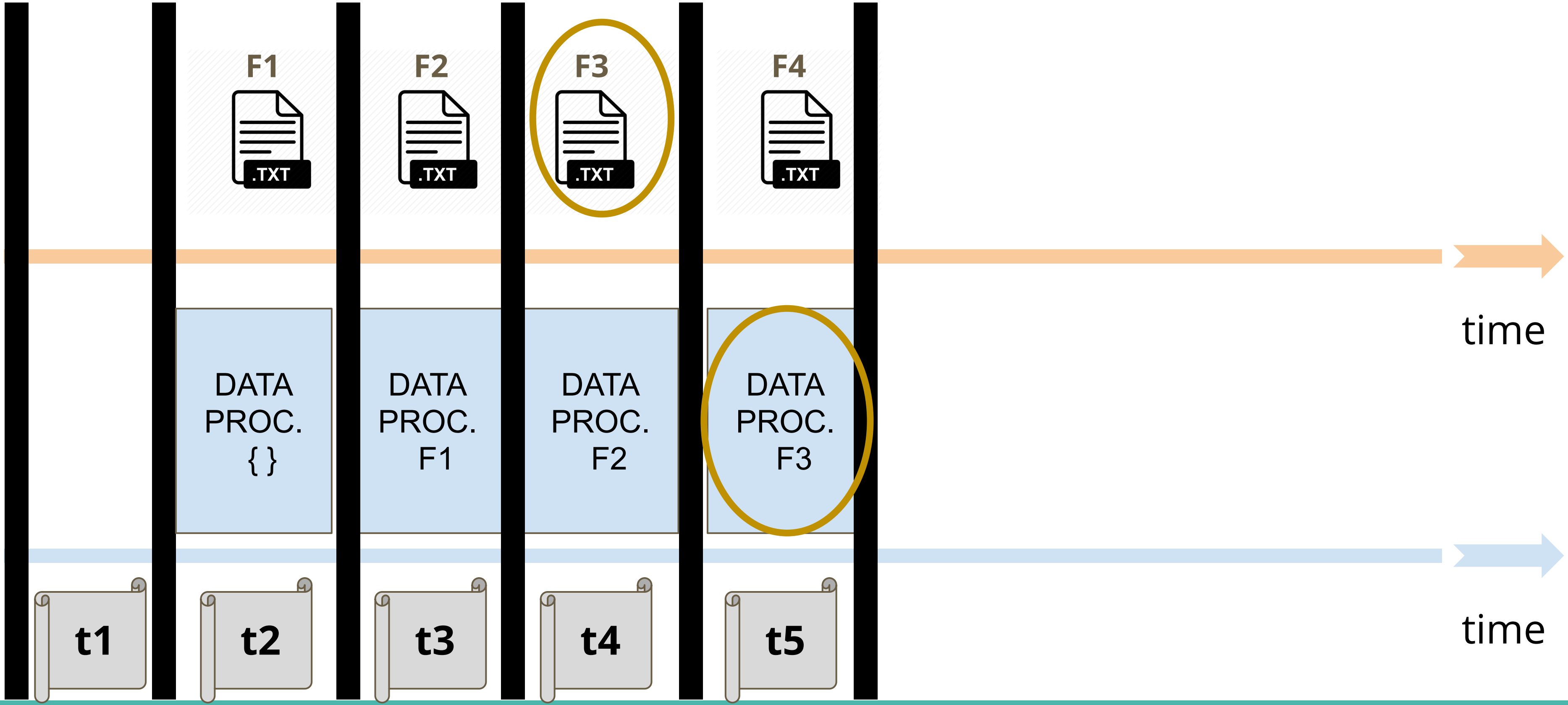
From RDDs to a DStream

Let's assume we simulate the files [F1.txt, …, F6.txt] to arrive in streaming, setting the time intervals to be of 3 seconds.



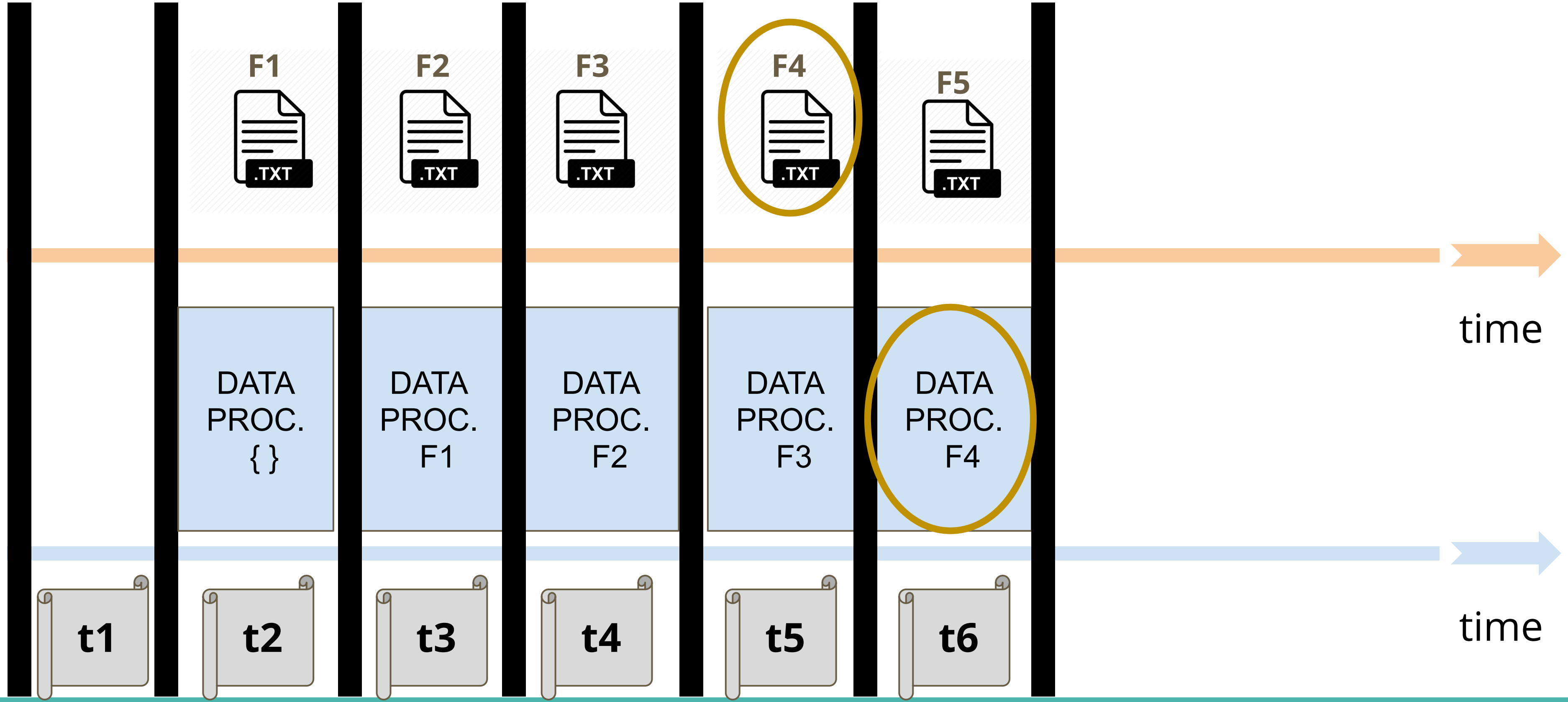
From RDDs to a DStream

Let's assume we simulate the files [F1.txt, ..., F6.txt] to arrive in streaming, setting the time intervals to be of 3 seconds.



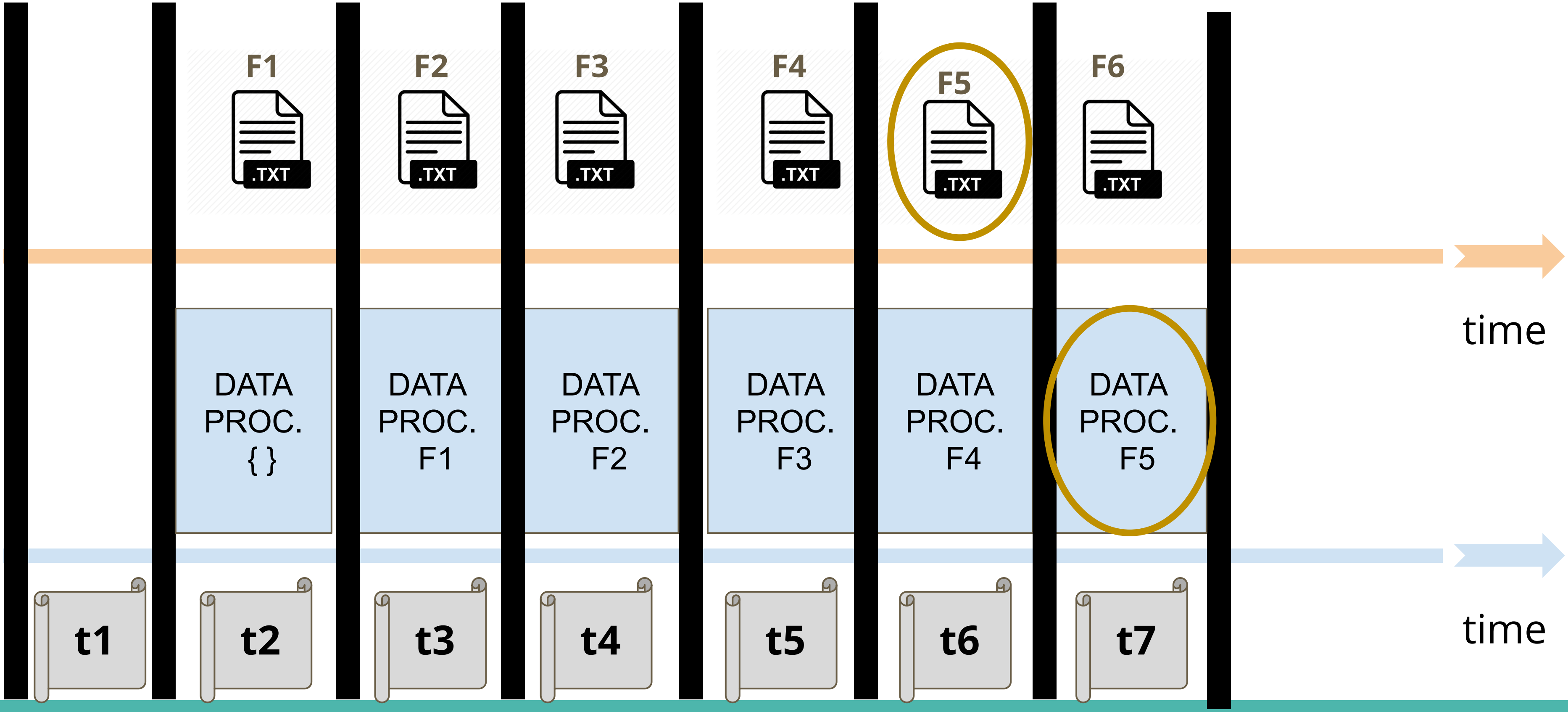
From RDDs to a DStream

Let's assume we simulate the files [F1.txt, ..., F6.txt] to arrive in streaming, setting the time intervals to be of 3 seconds.



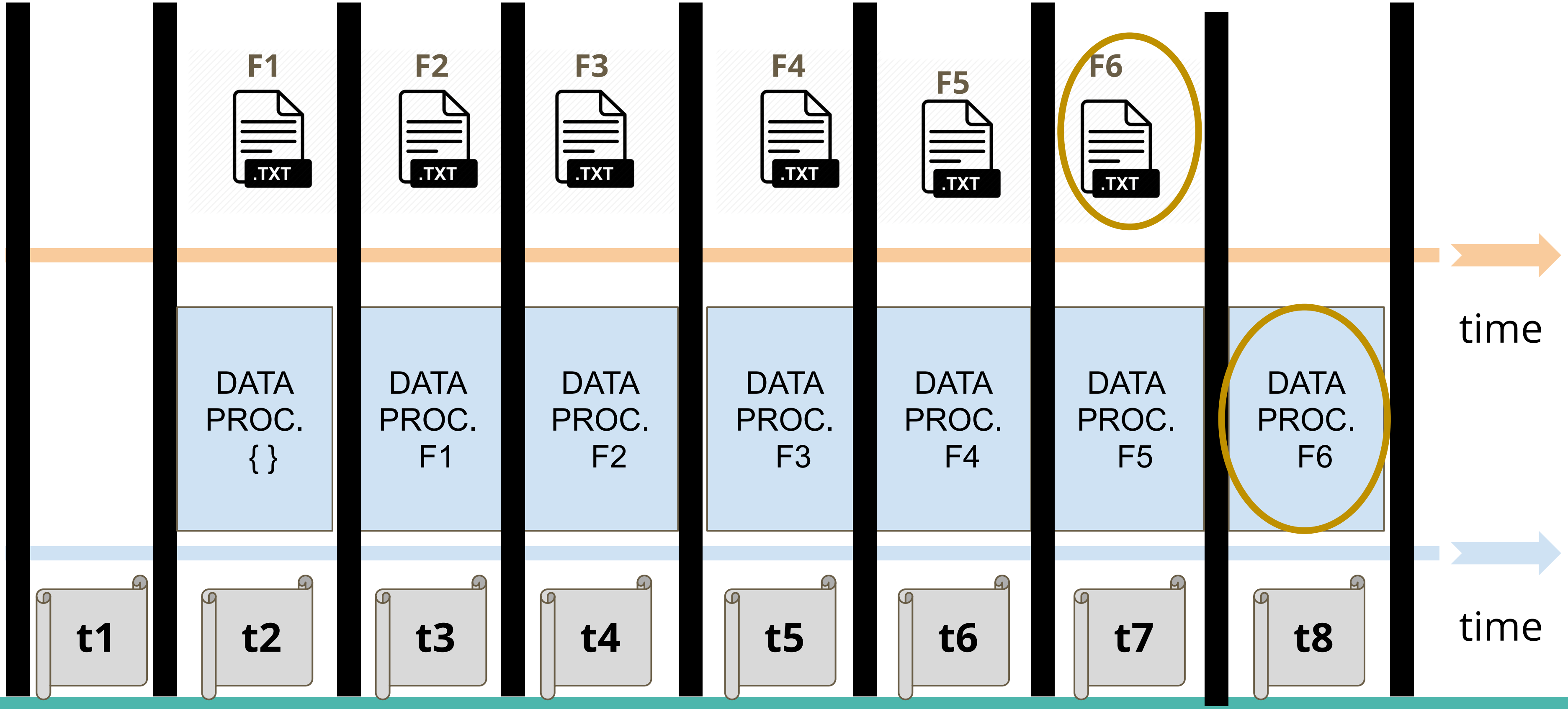
From RDDs to a DStream

Let's assume we simulate the files [F1.txt, ..., F6.txt] to arrive in streaming, setting the time intervals to be of 3 seconds.



From RDDs to a DStream

Let's assume we simulate the files [F1.txt, ..., F6.txt] to arrive in streaming, setting the time intervals to be of 3 seconds.

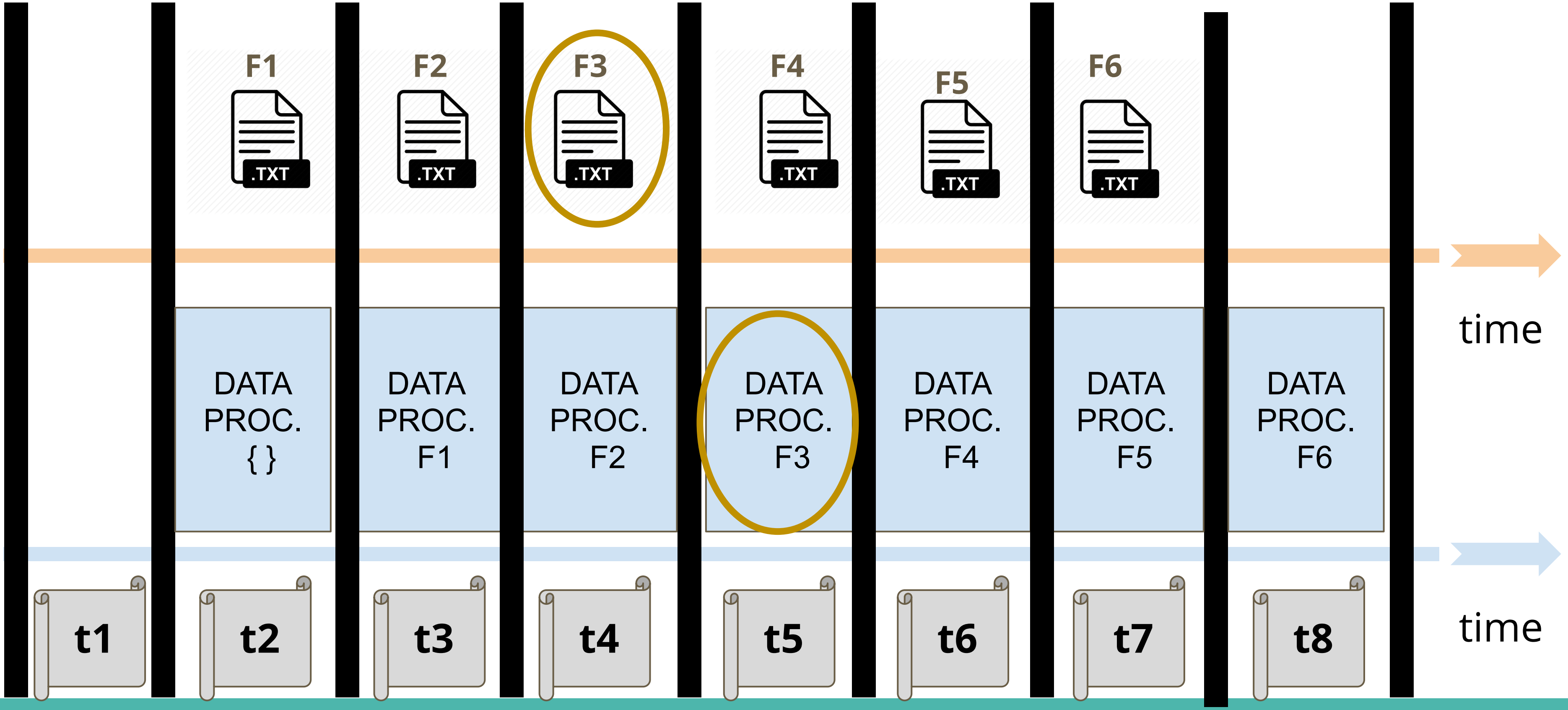


From RDDs to a DStream

So…

From RDDs to a DStream

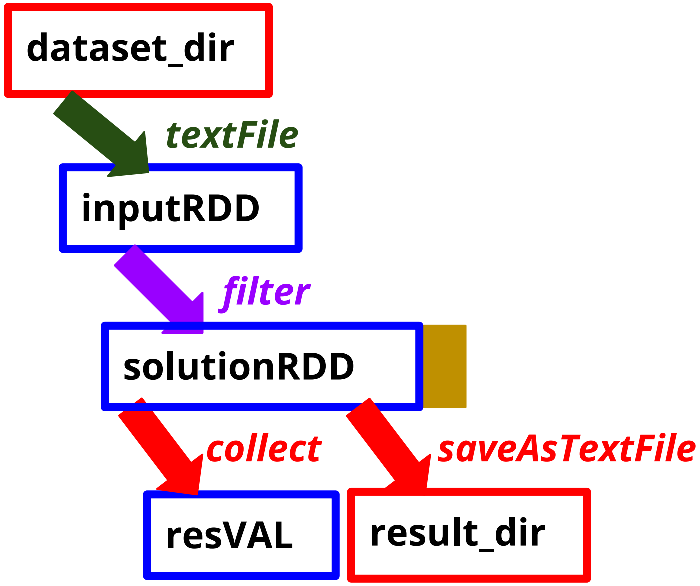
How does the processing of a concrete file (e.g., F3.txt) look like?



From RDDs to a DStream

Exactly the same as the processing of F.txt for the Spark Core example!

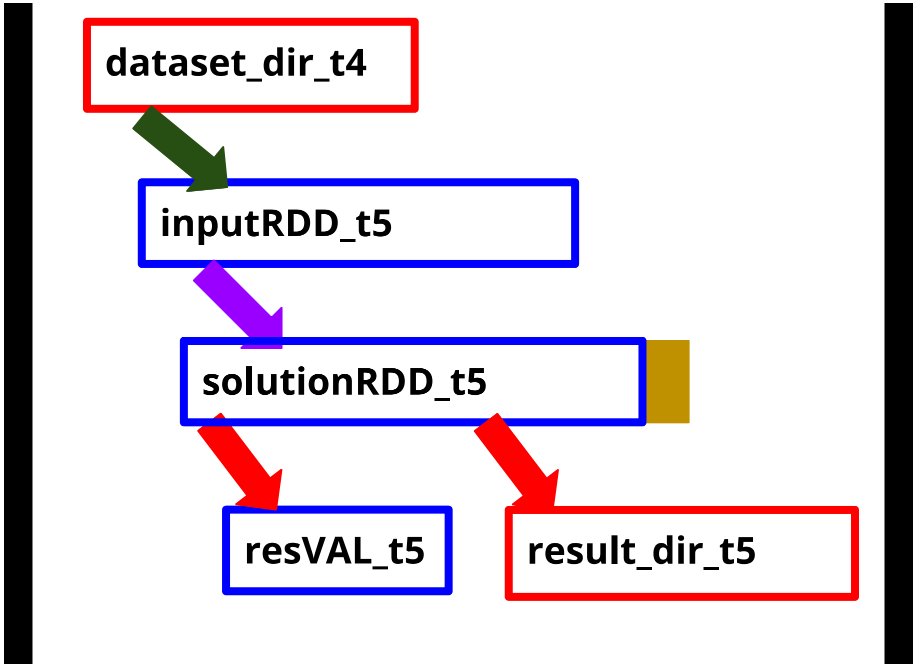
Processing of F.txt in Spark Core:



From RDDs to a DStream

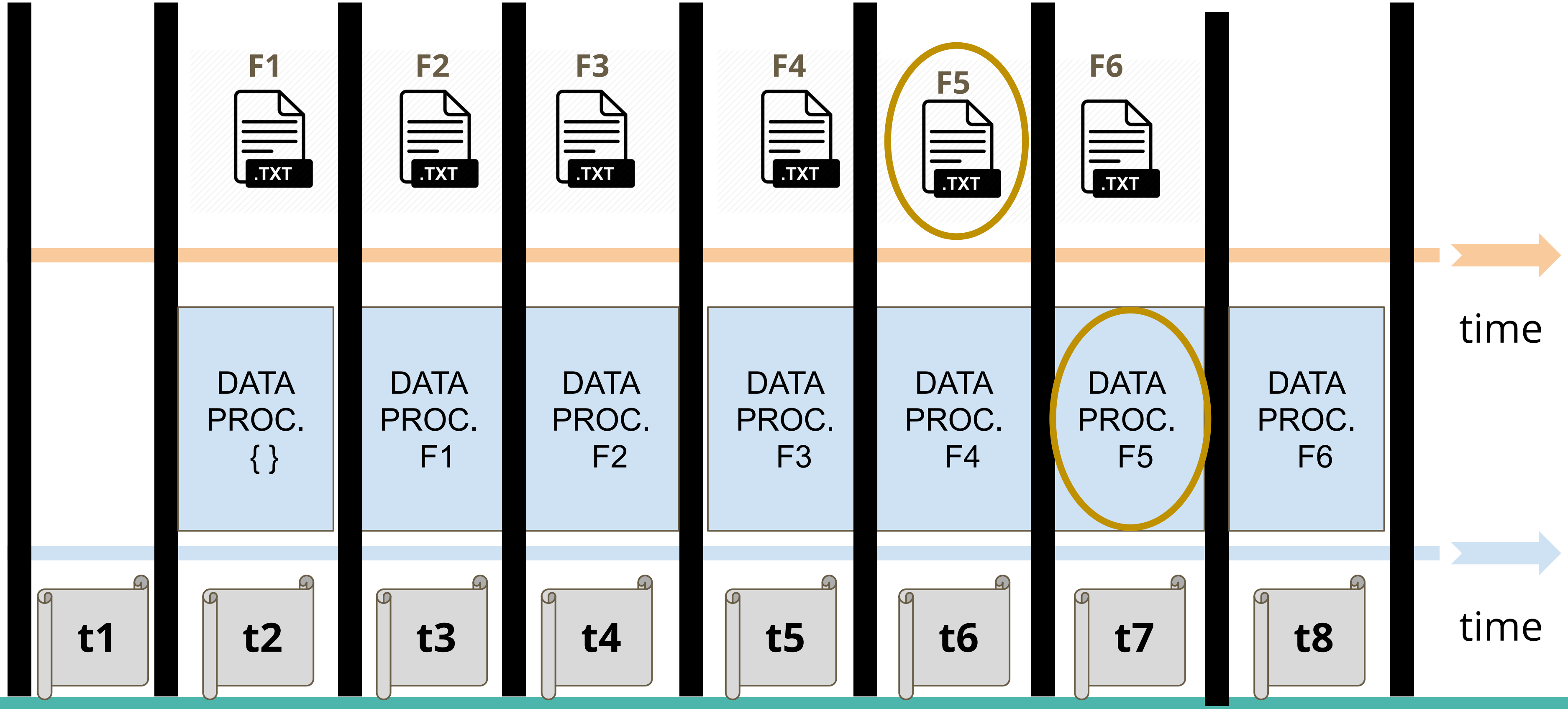
Exactly the same as the processing of F.txt for the Spark Core example!

Processing of F3.txt in Spark Streaming:



From RDDs to a DStream

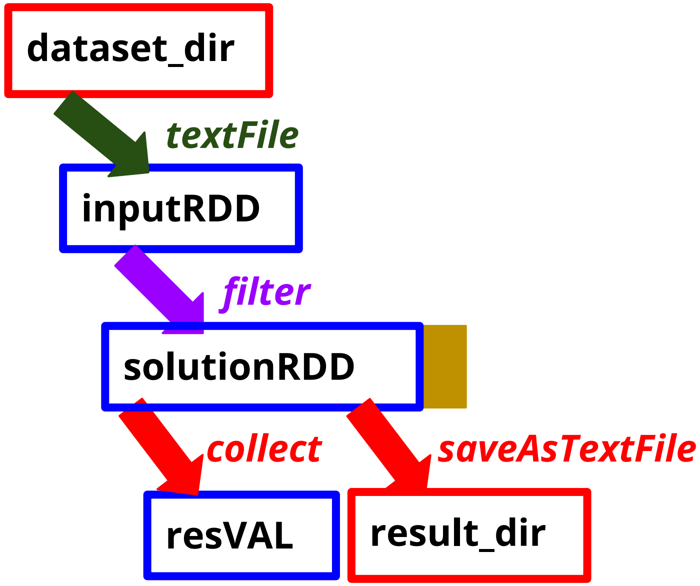
And how does the processing of another concrete file (e.g., F5.txt) look like?



From RDDs to a DStream

Exactly the same as processing of F.txt for the Spark Core example!

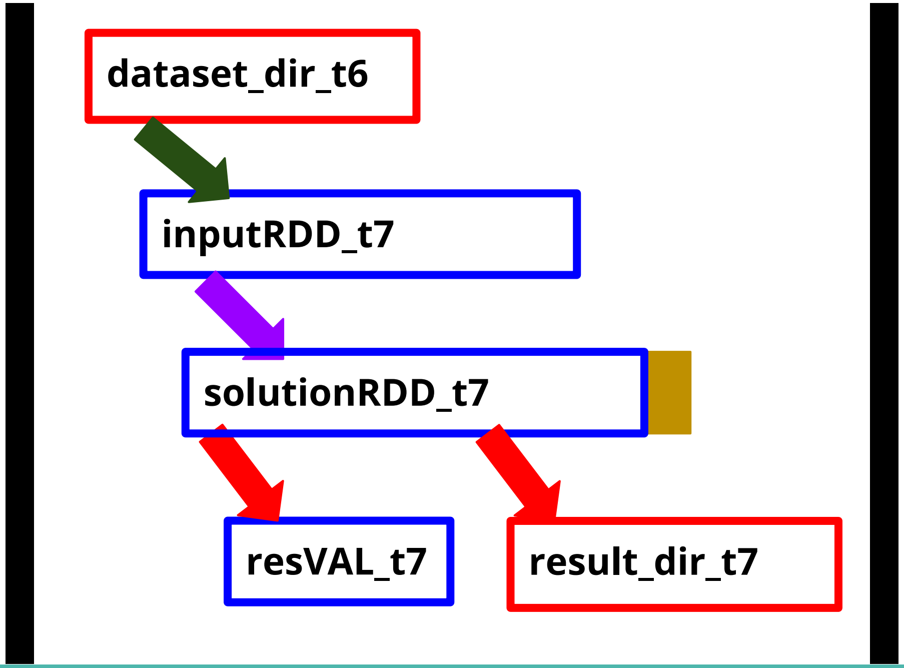
Processing of F.txt in Spark Core:



From RDDs to a DStream

Exactly the same as processing of F.txt for the Spark Core example!

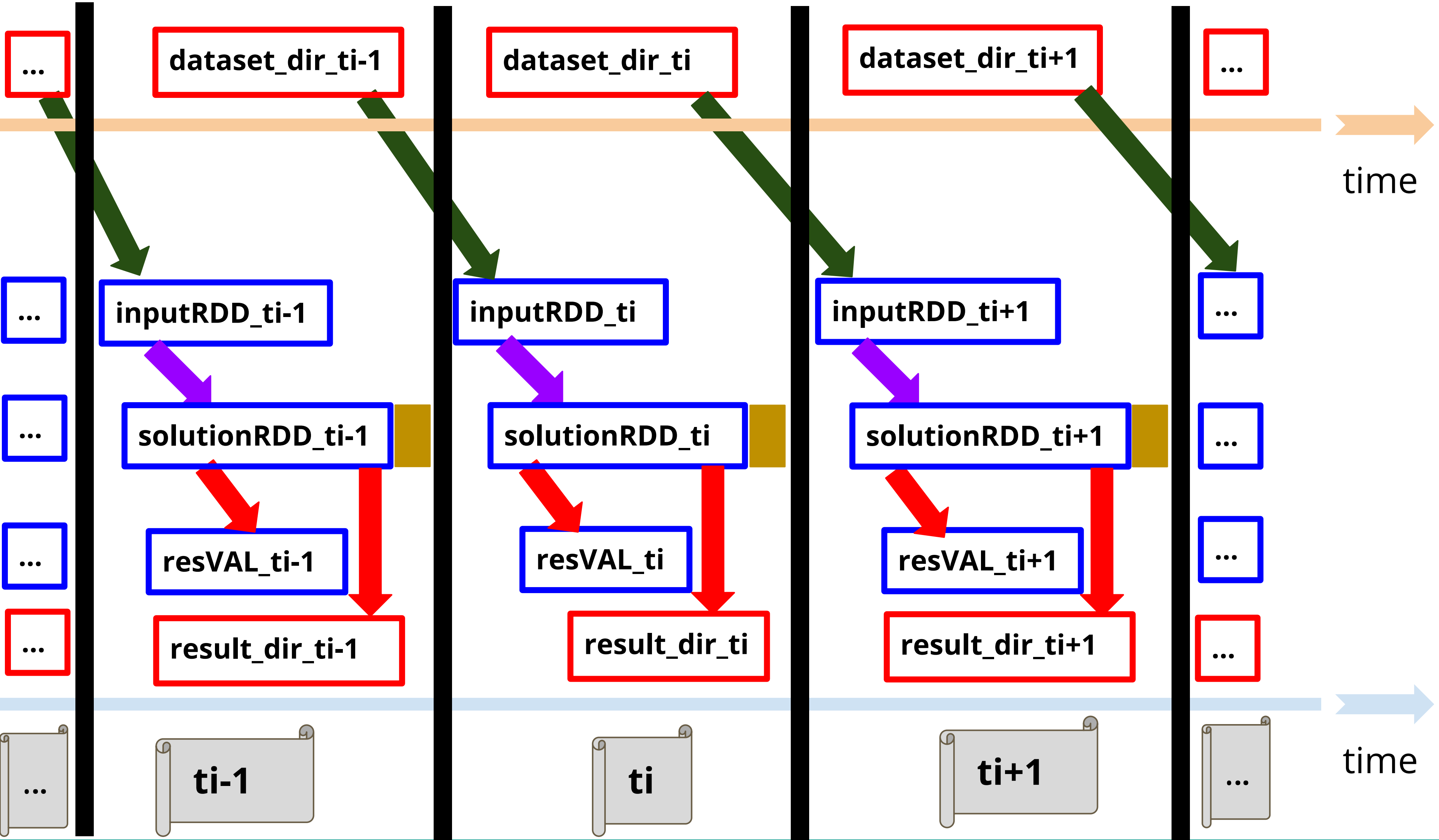
Processing of F5.txt in Spark Streaming:



From RDDs to a DStream

So, all in all,
What we are computing is nothing but
an RDD per **time interval t_i**

From RDDs to a DStream



From RDDs to a DStream

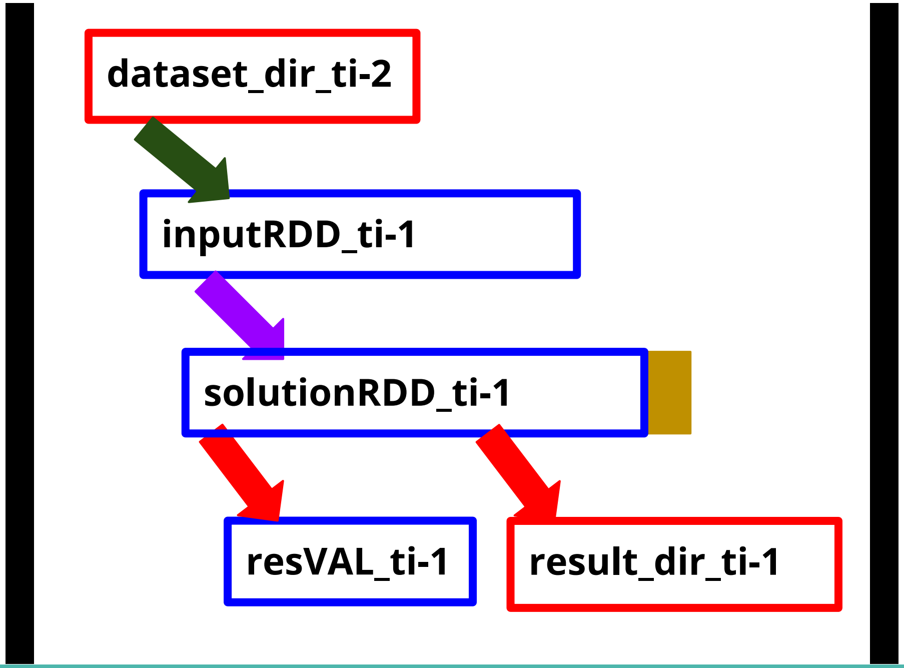
Looking at it as the different time intervals arise…

From RDDs to a DStream

Time Interval t_{i-1}

From RDDs to a DStream

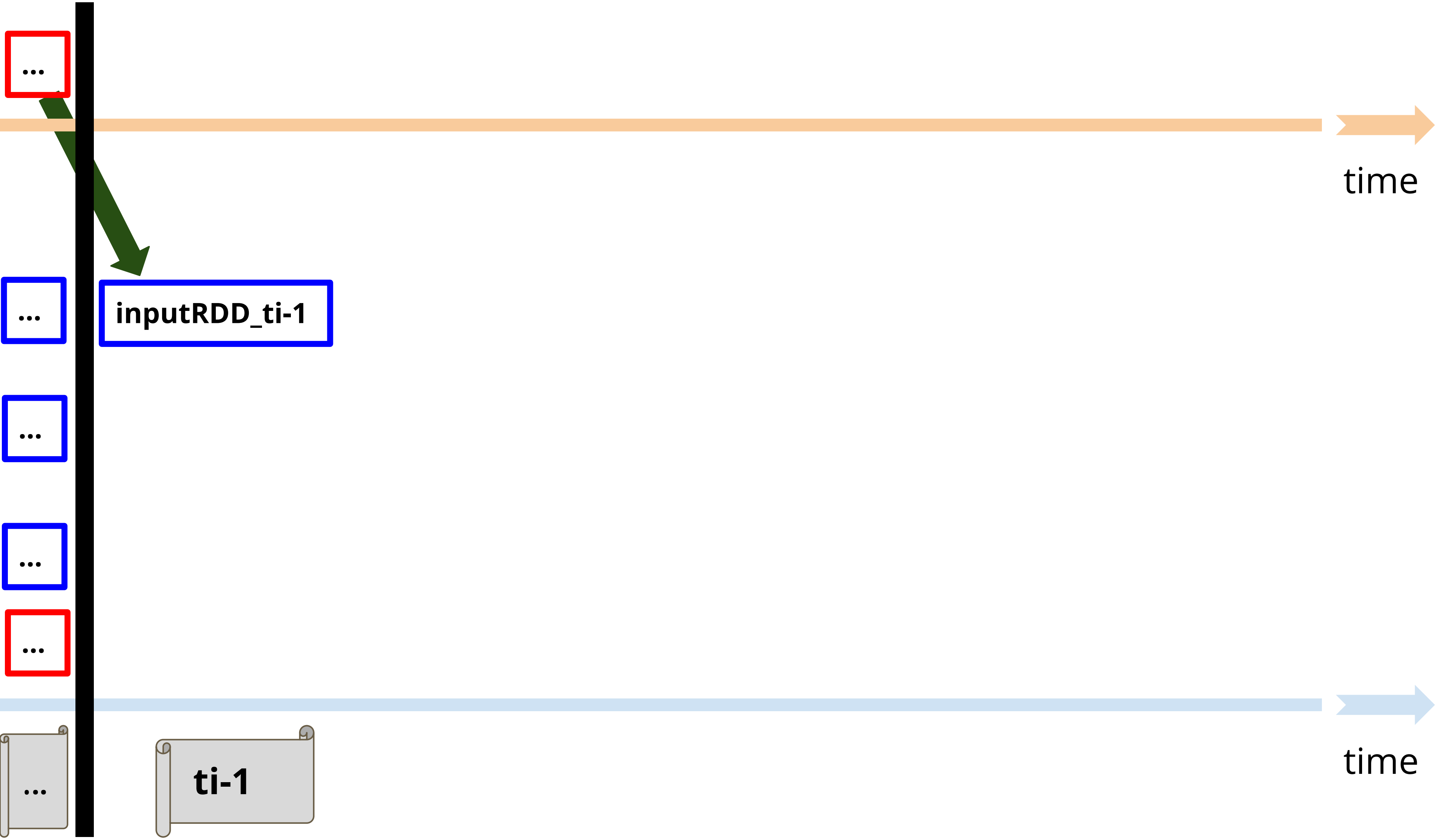
Processing of Fi-3.txt in Spark Streaming:



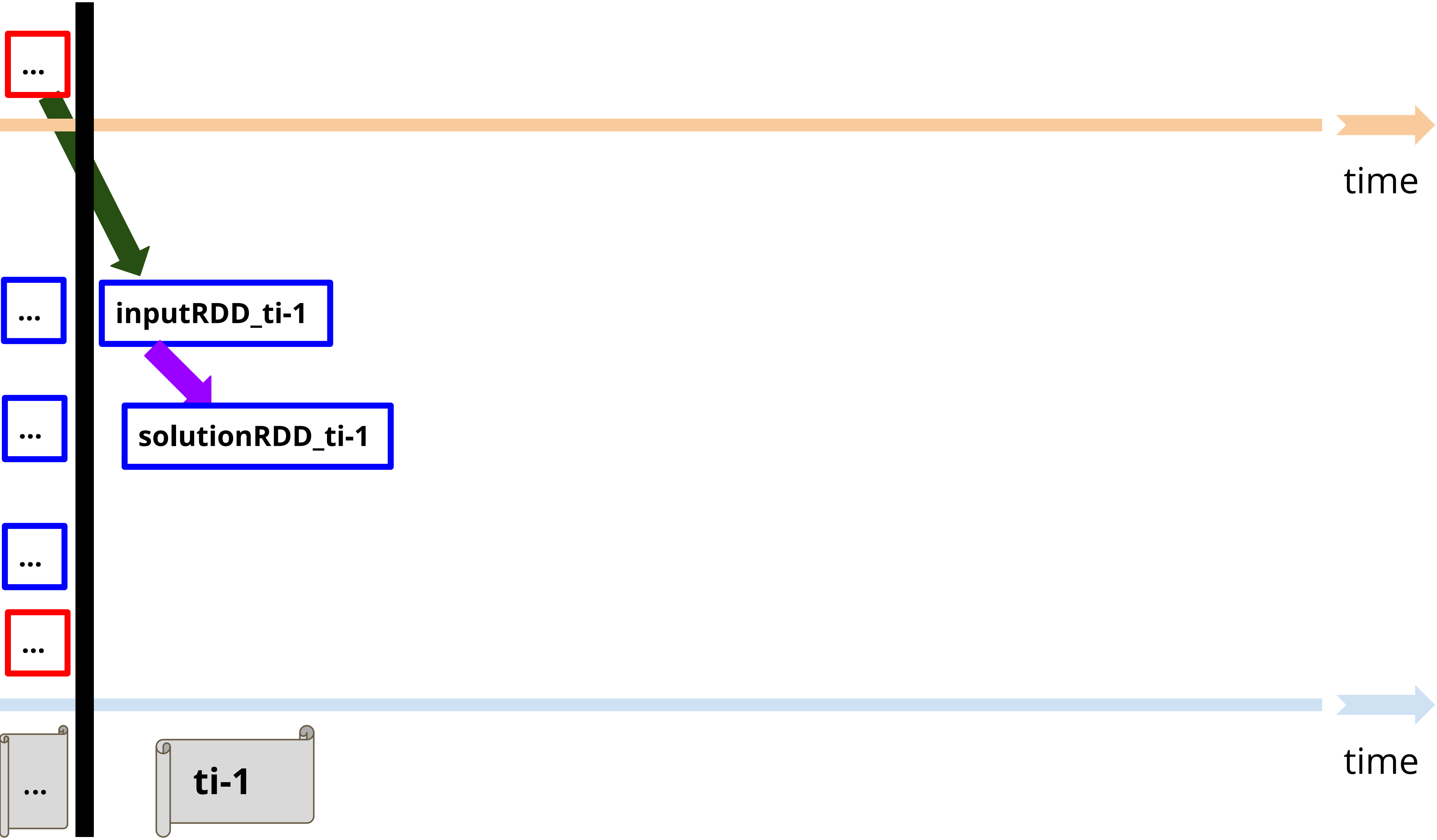
From RDDs to a DStream

...
time
...
...
...
...
time
...
ti-1

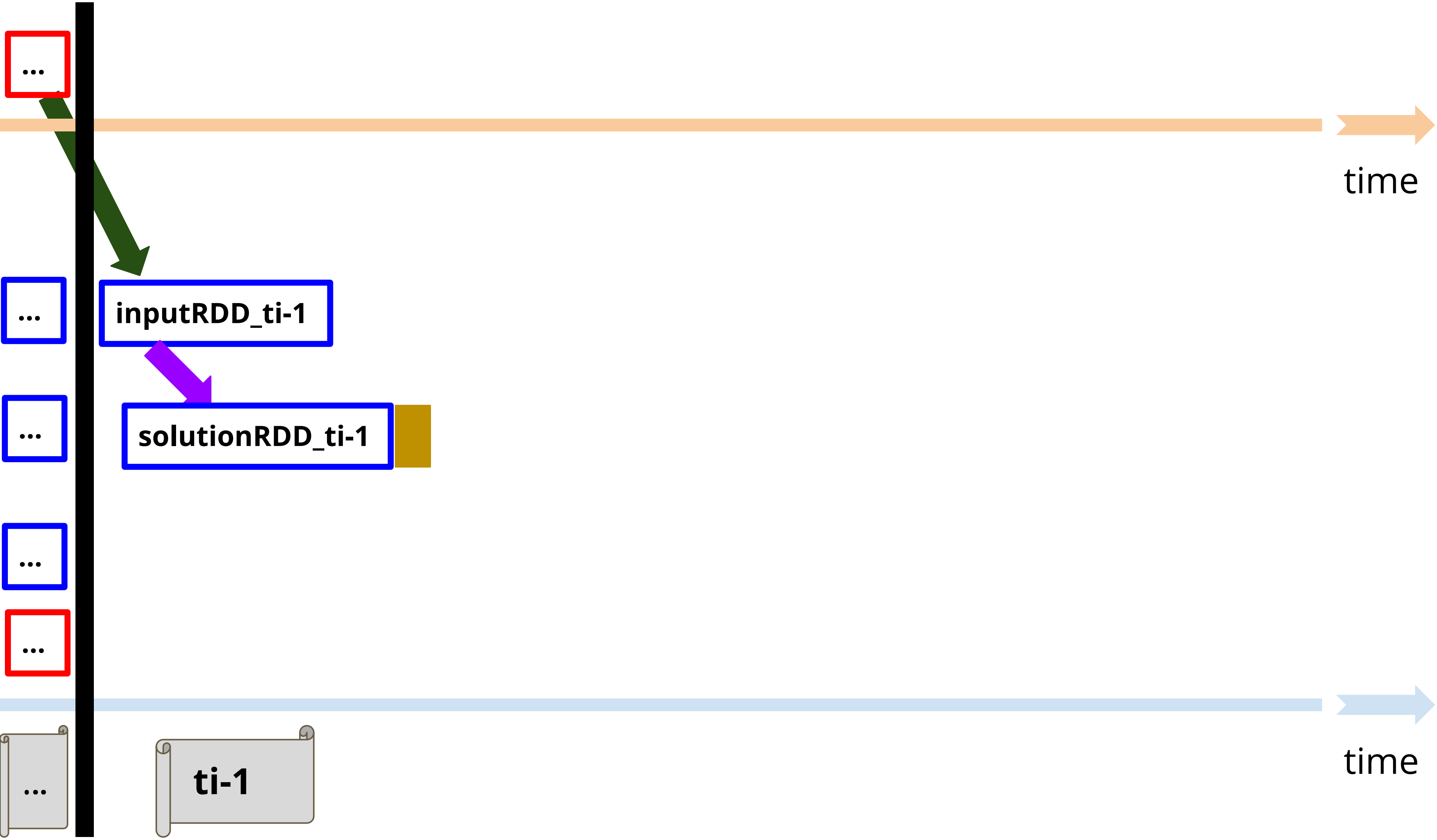
From RDDs to a DStream



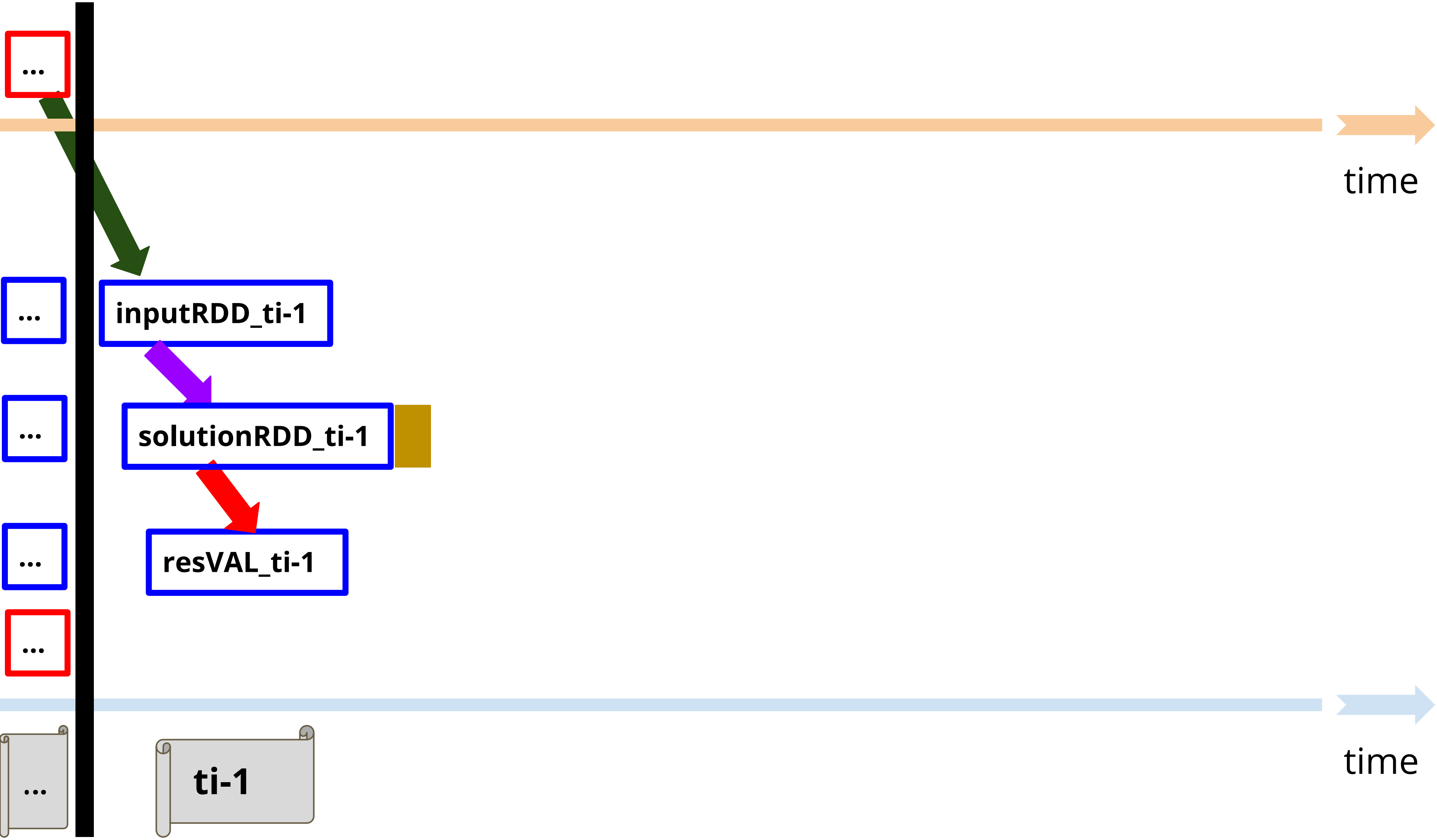
From RDDs to a DStream



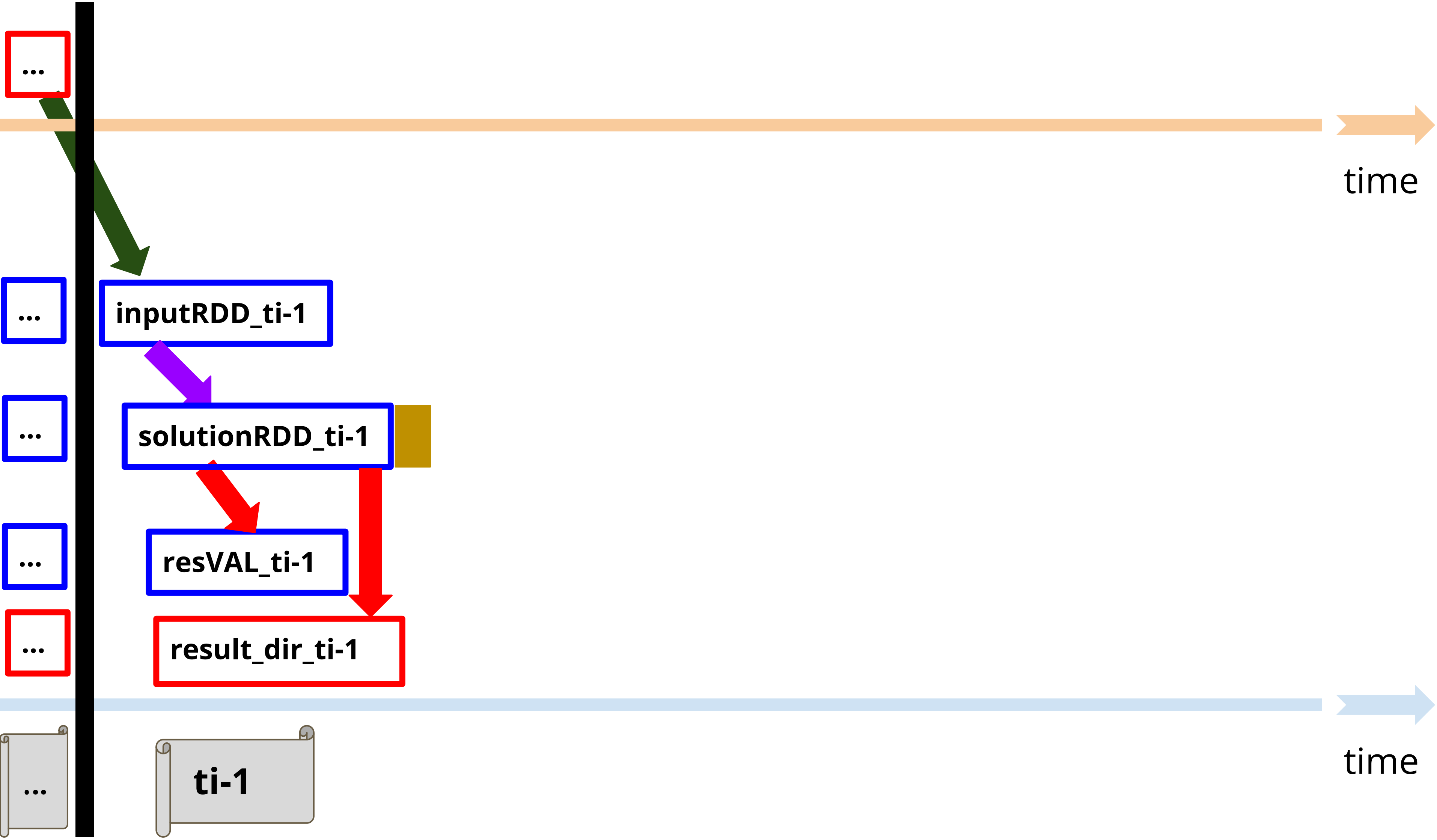
From RDDs to a DStream



From RDDs to a DStream



From RDDs to a DStream

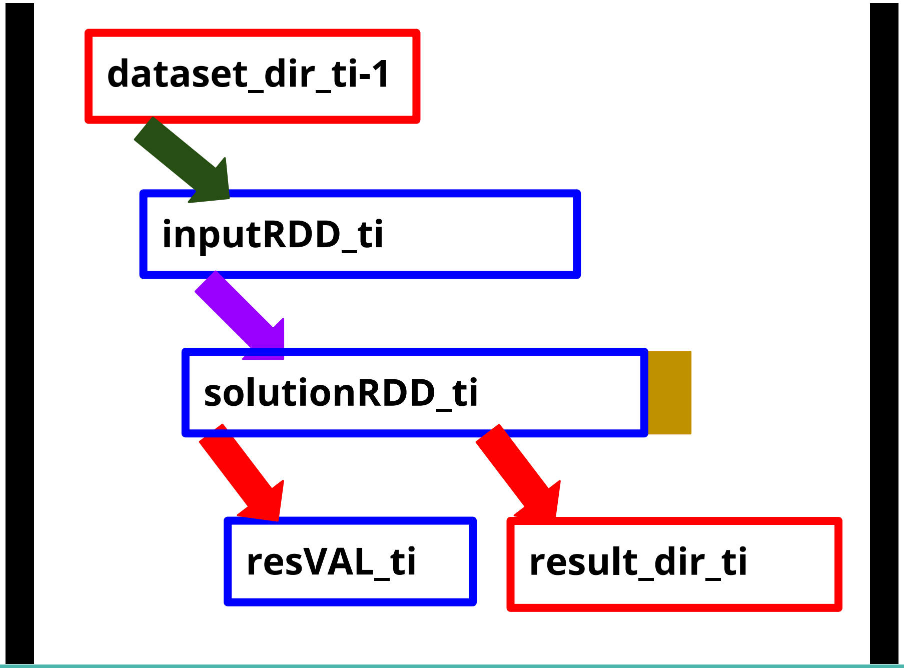


From RDDs to a DStream

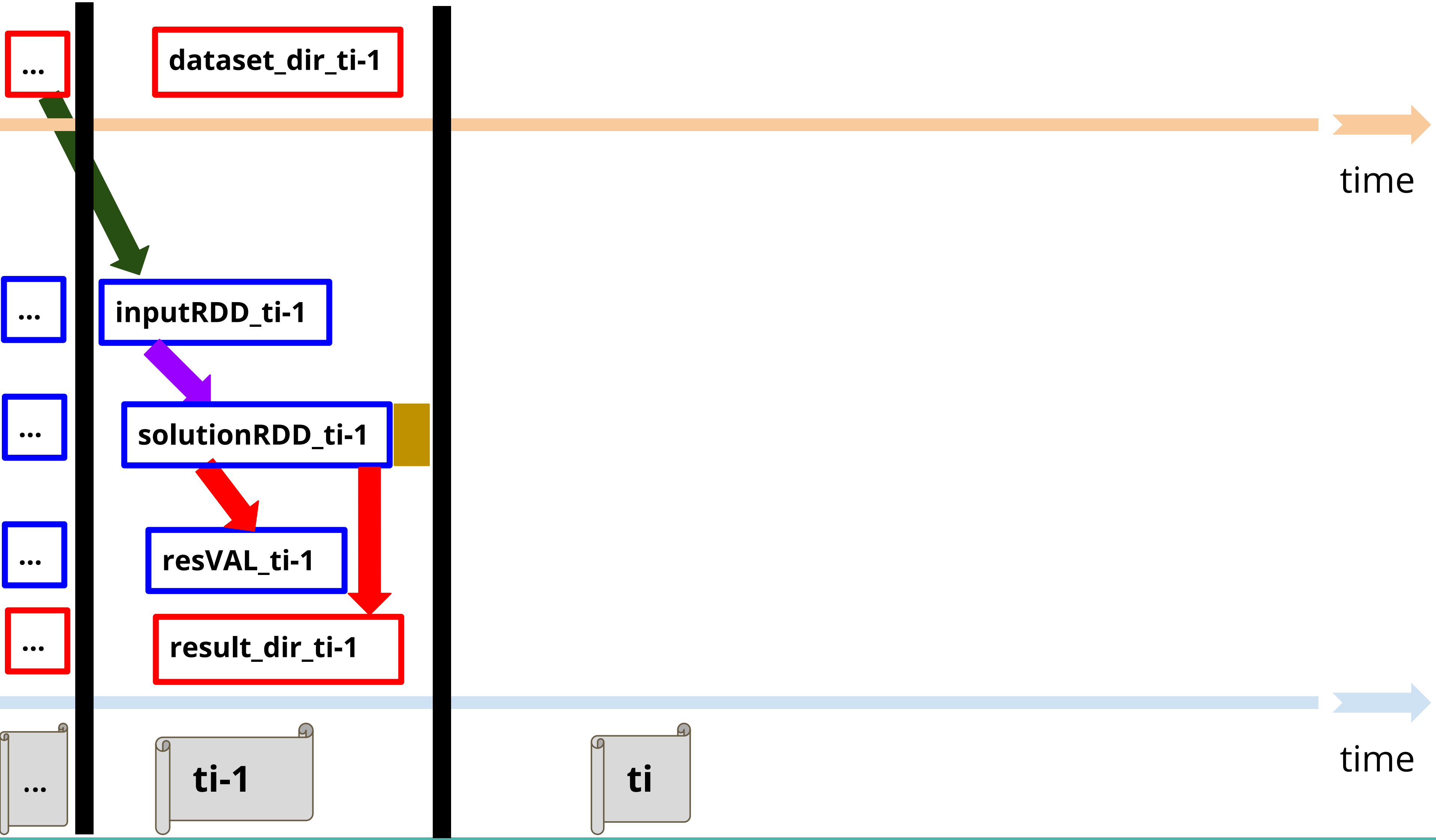
Time Interval t_i

From RDDs to a DStream

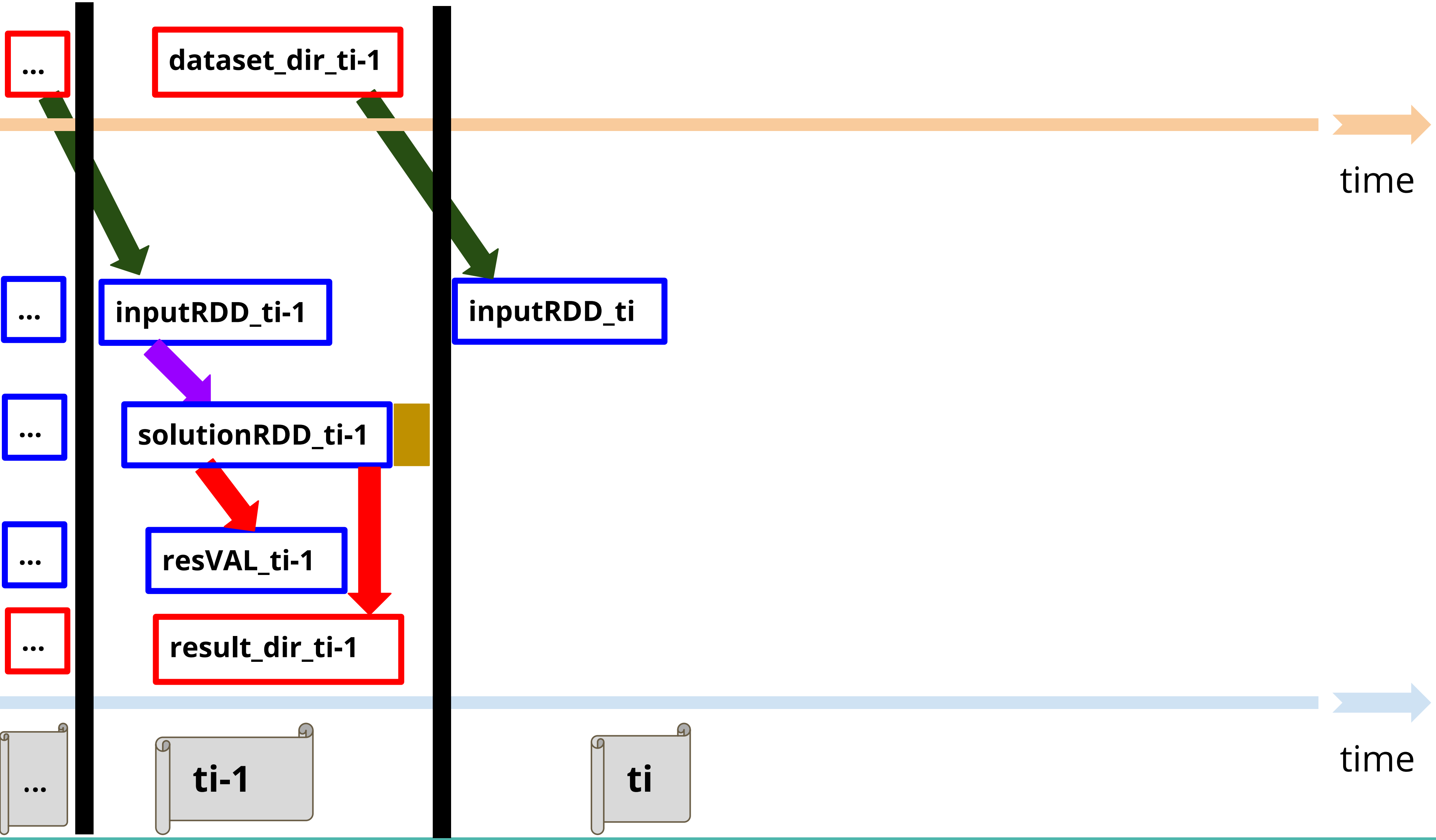
Processing of Fi-2.txt in Spark Streaming:



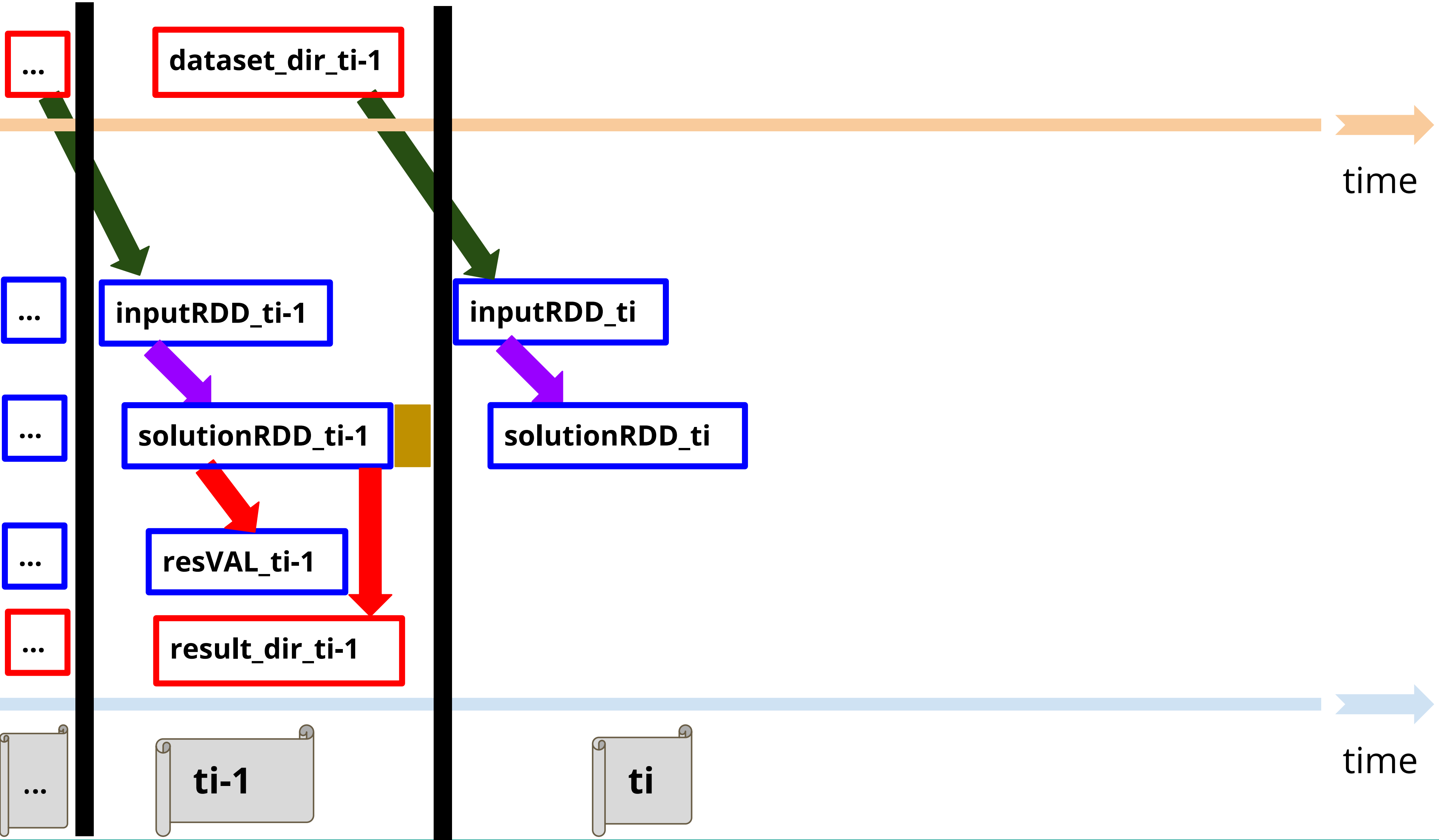
From RDDs to a DStream



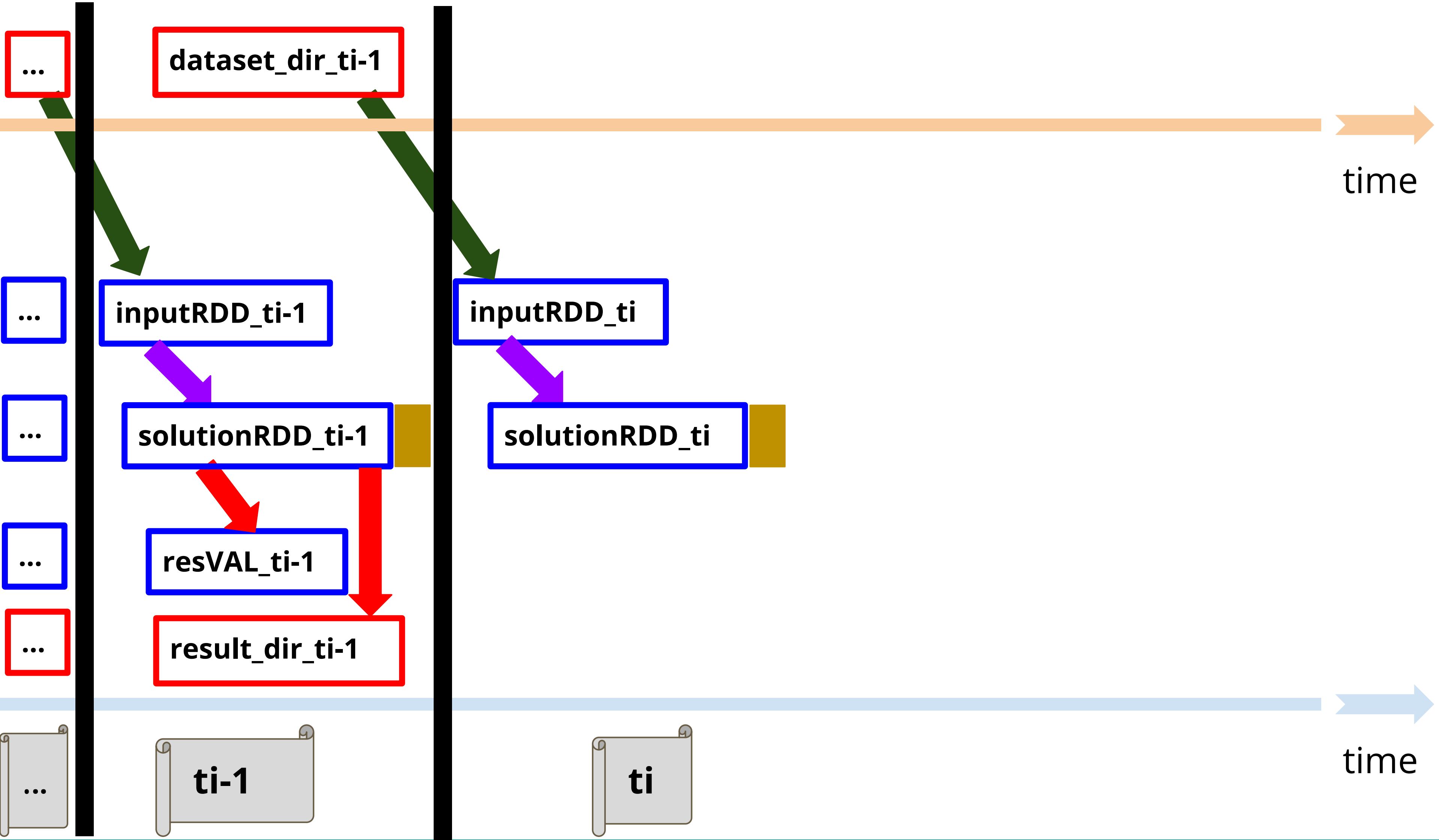
From RDDs to a DStream



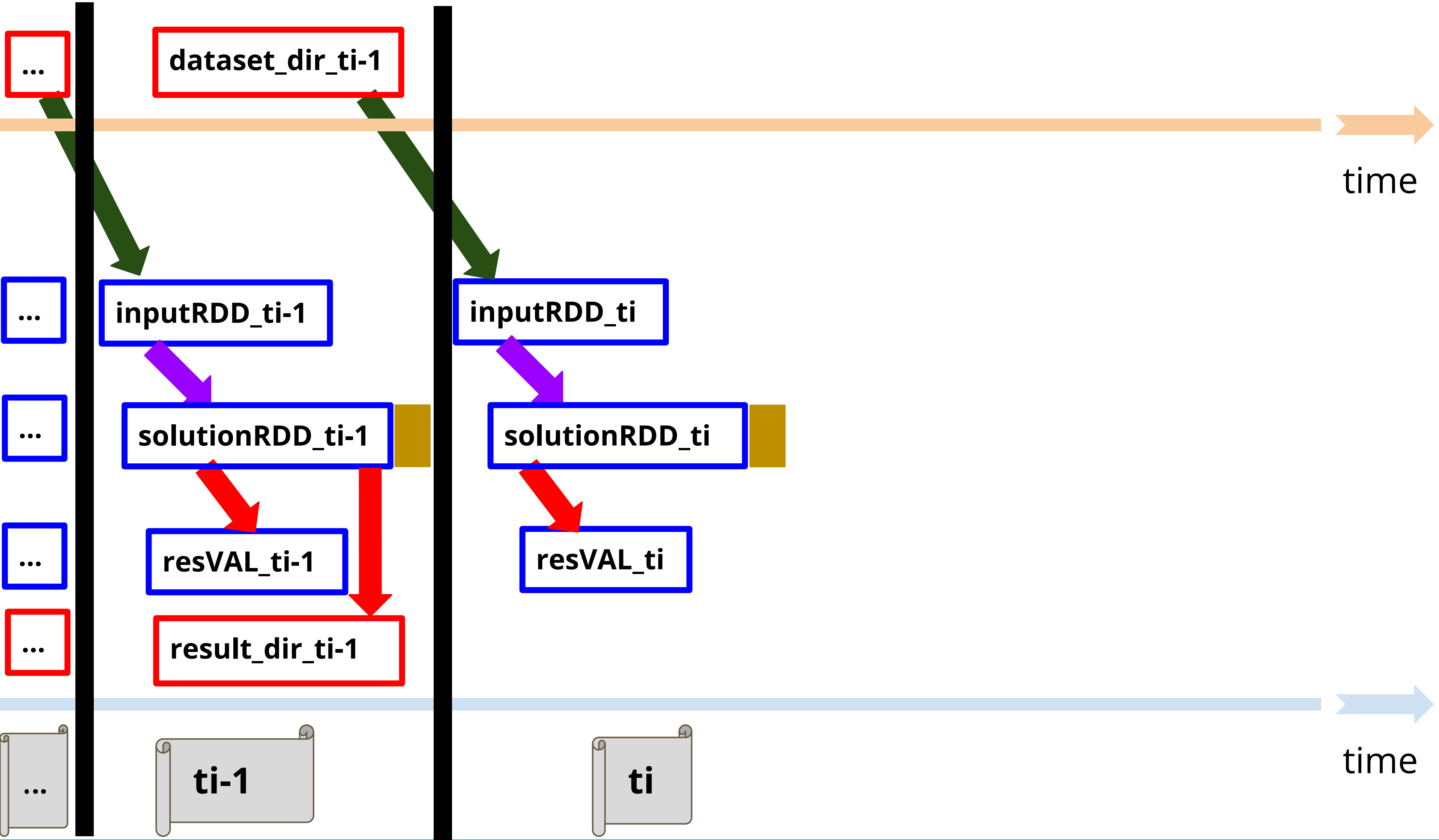
From RDDs to a DStream



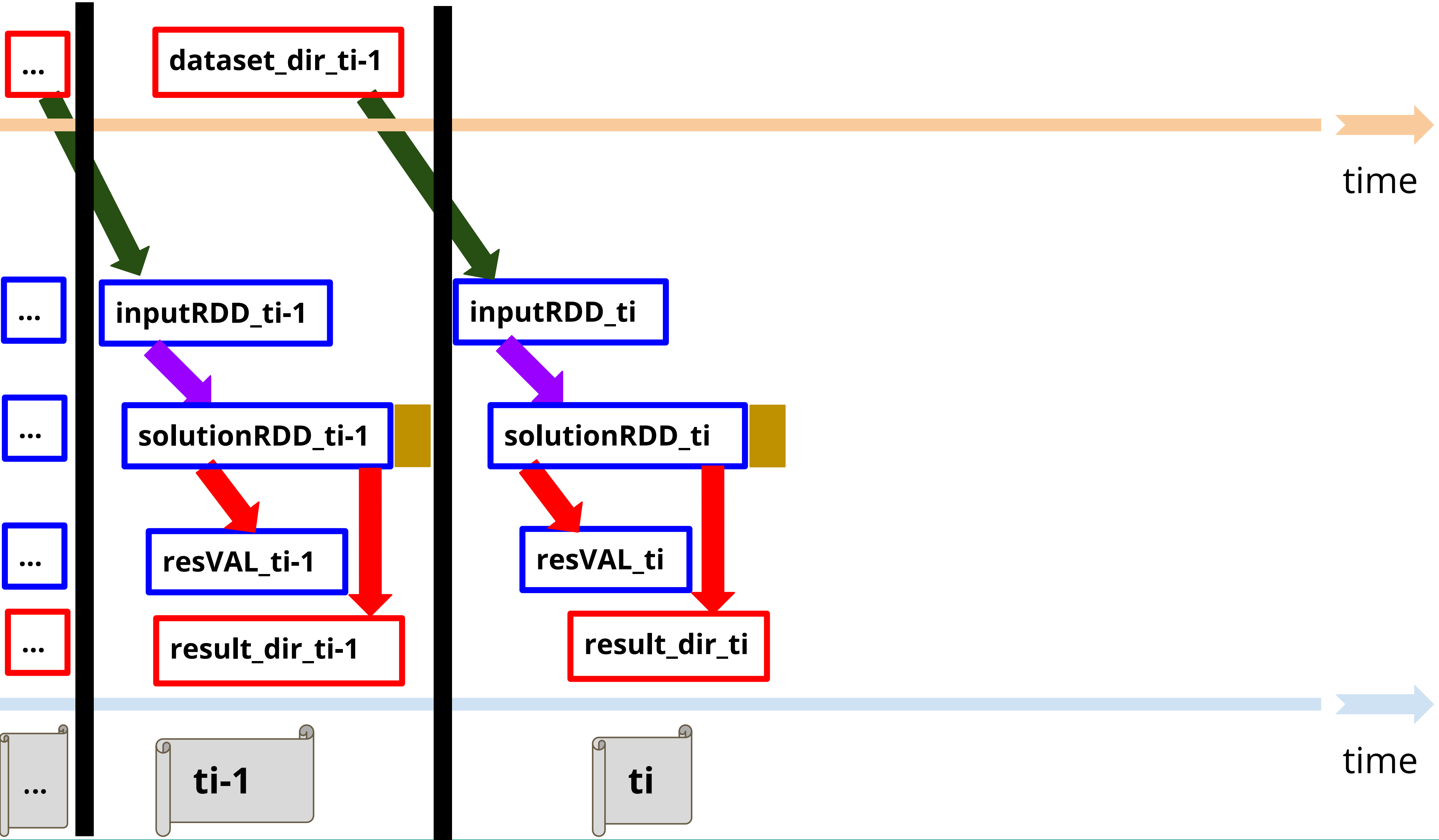
From RDDs to a DStream



From RDDs to a DStream



From RDDs to a DStream

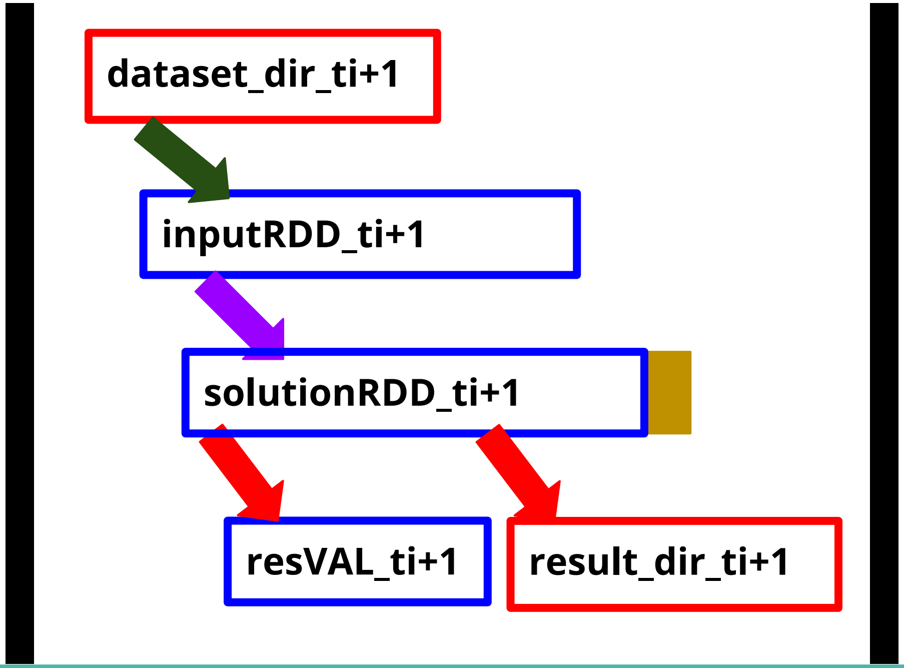


From RDDs to a DStream

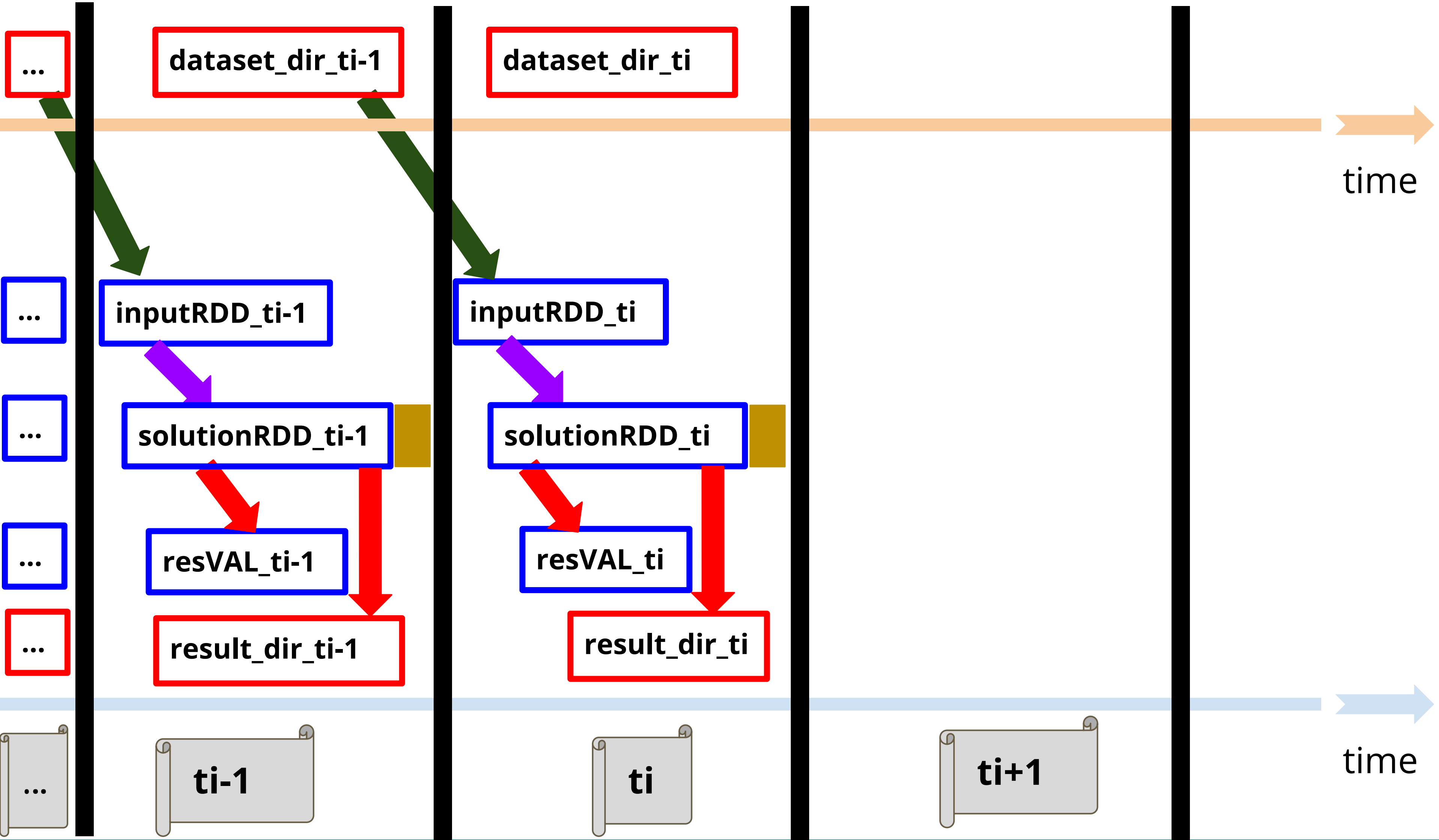
Time Interval t_{i+1}

From RDDs to a DStream

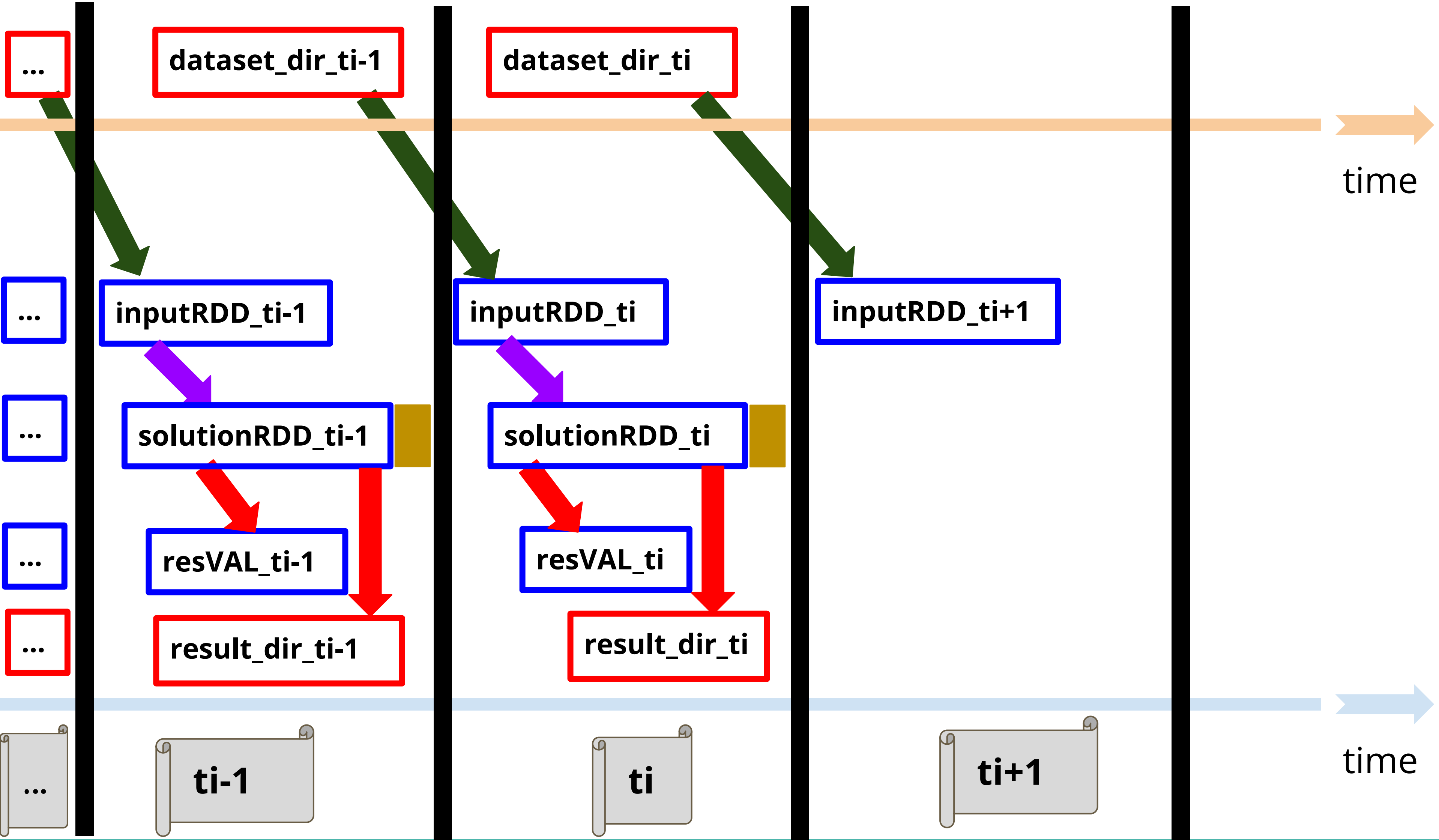
Processing of Fi-1.txt in Spark Streaming:



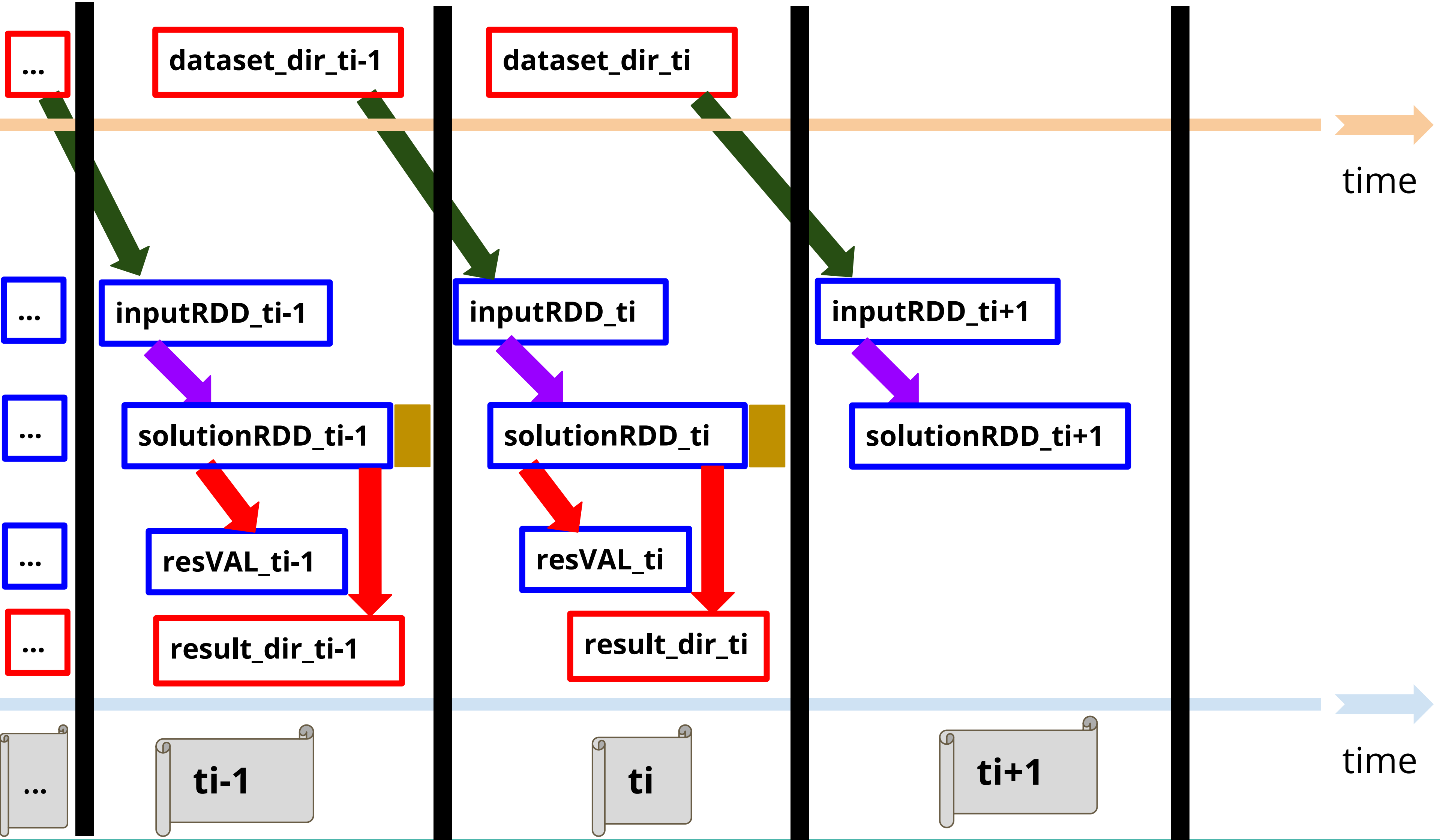
From RDDs to a DStream



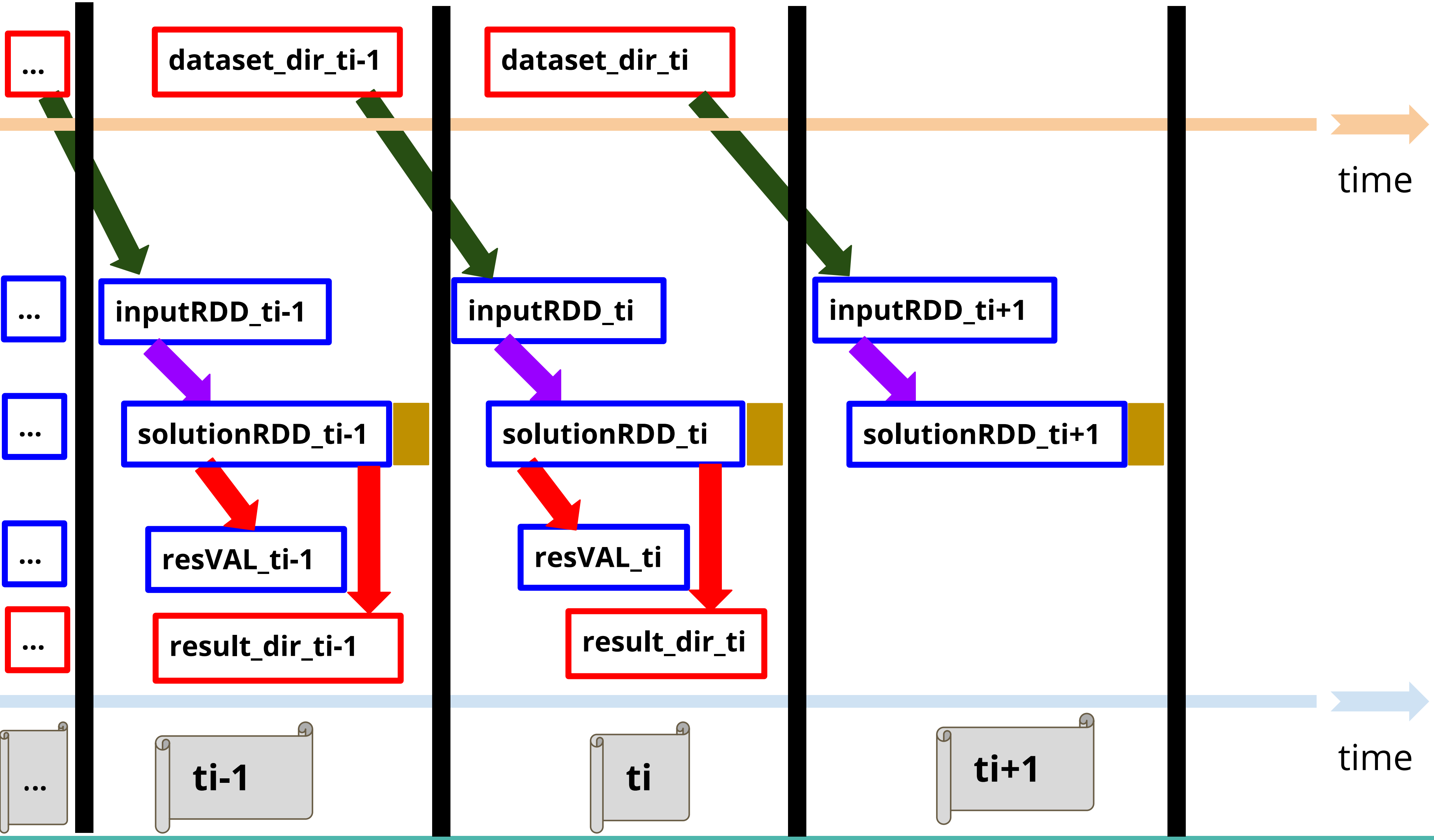
From RDDs to a DStream



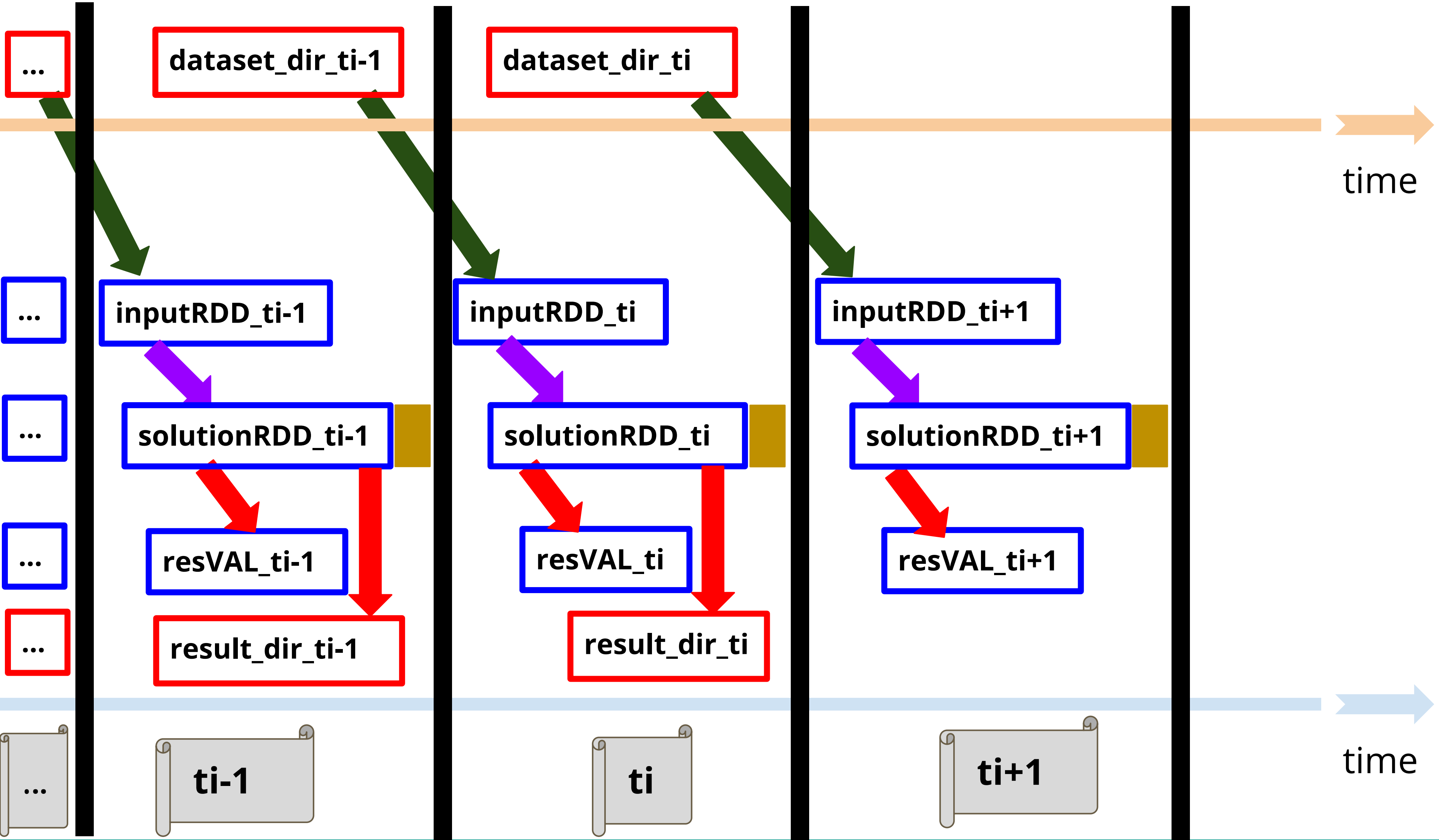
From RDDs to a DStream



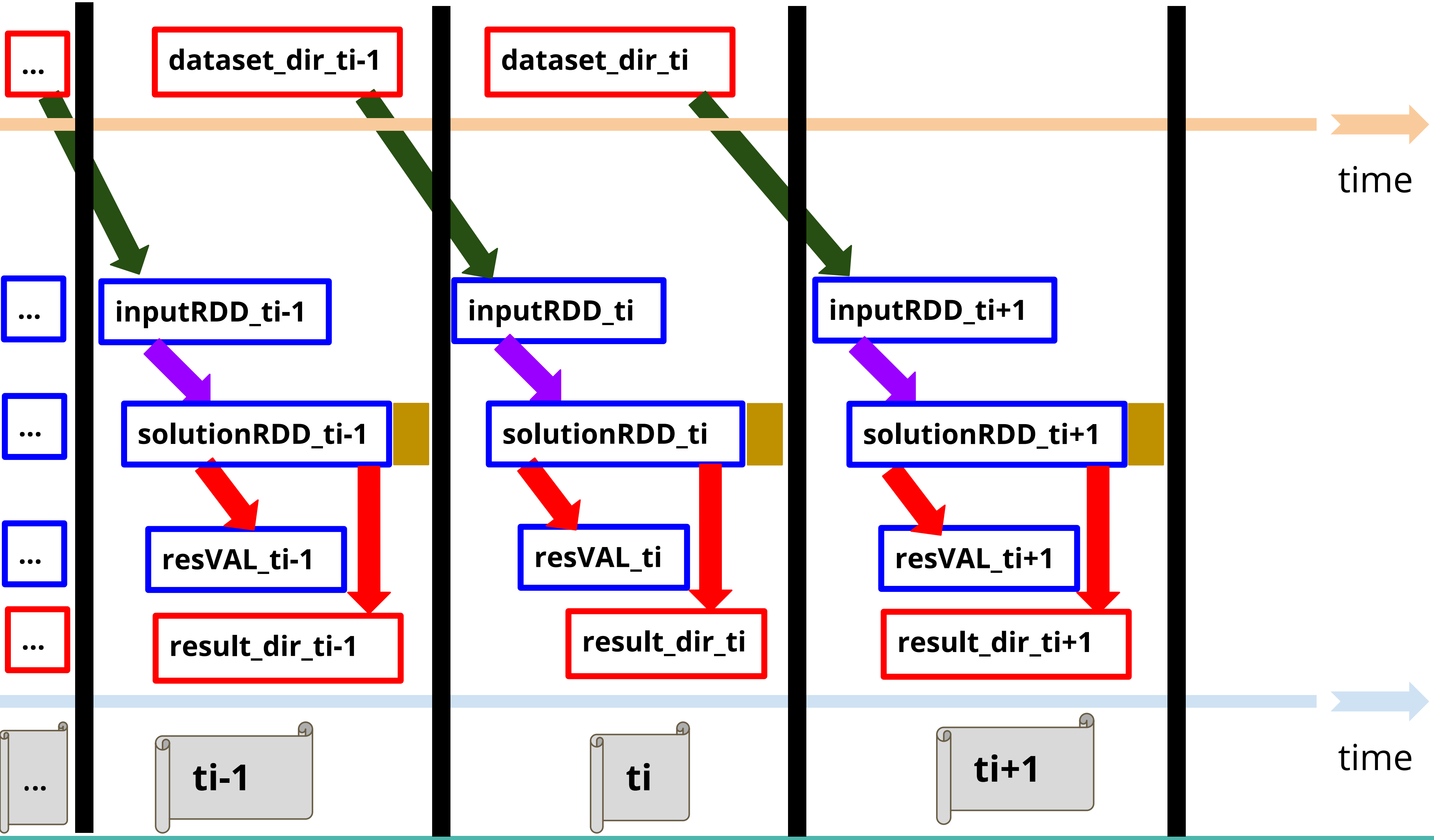
From RDDs to a DStream



From RDDs to a DStream



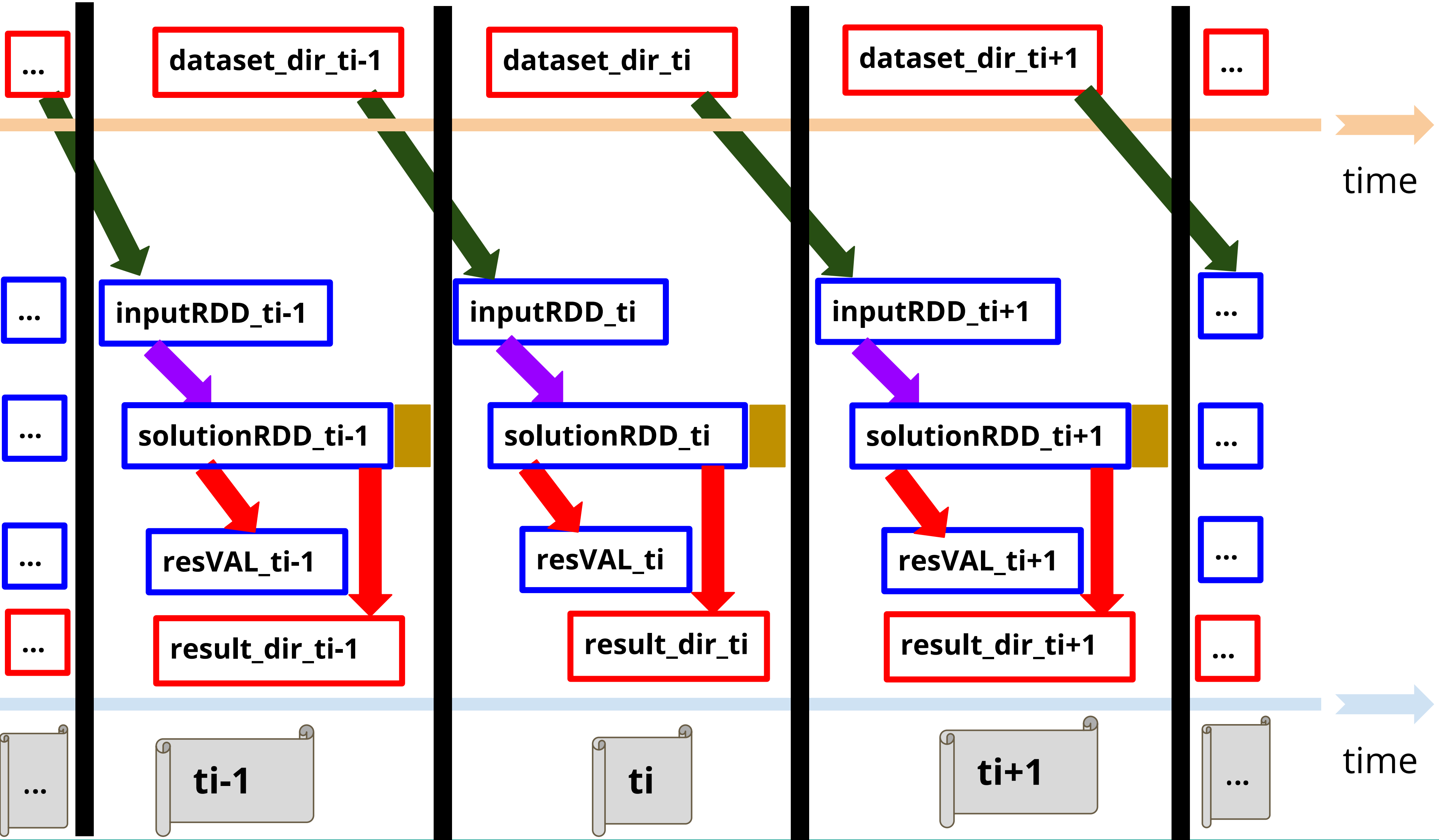
From RDDs to a DStream



From RDDs to a DStream

and so on...

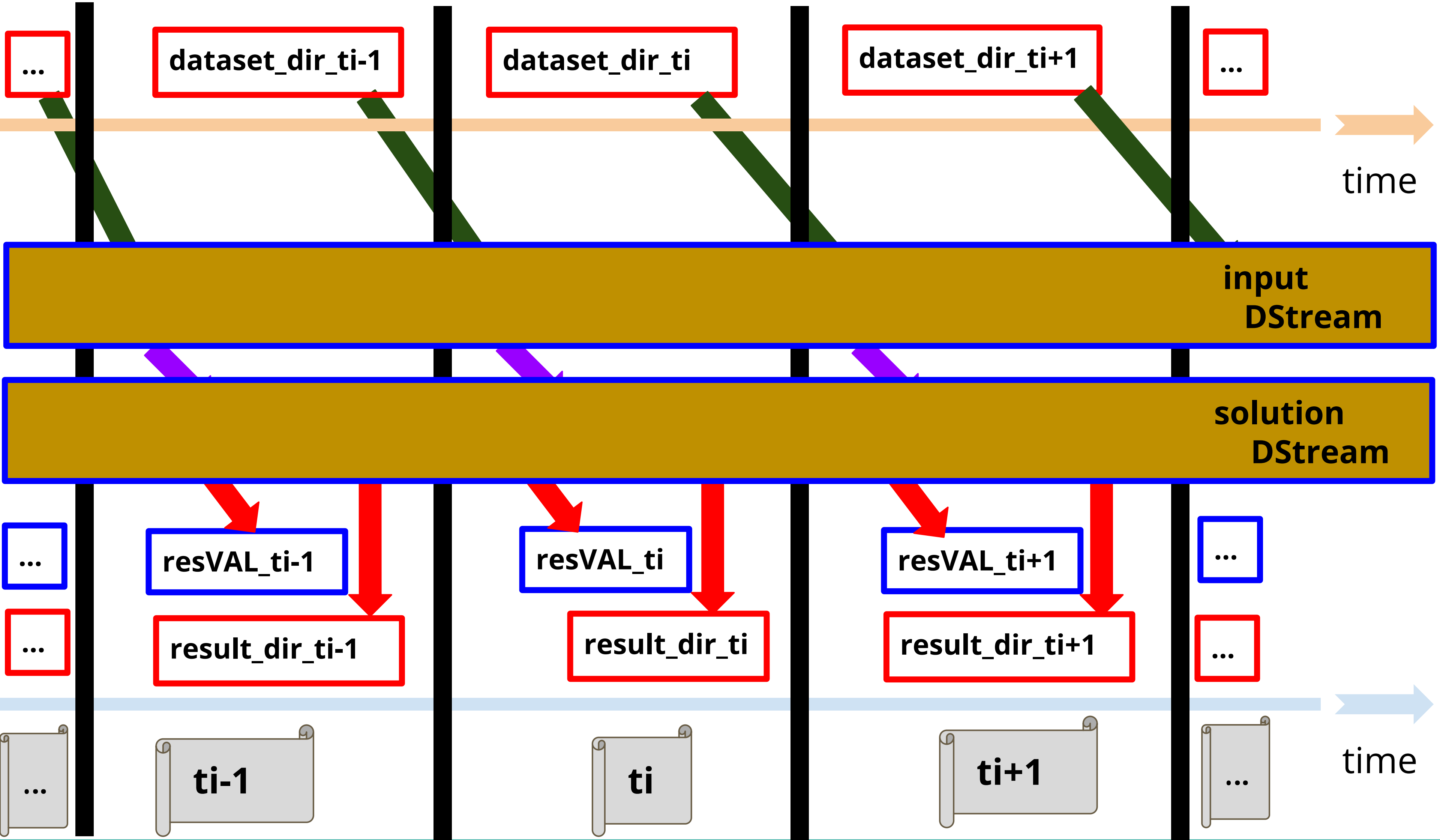
From RDDs to a DStream



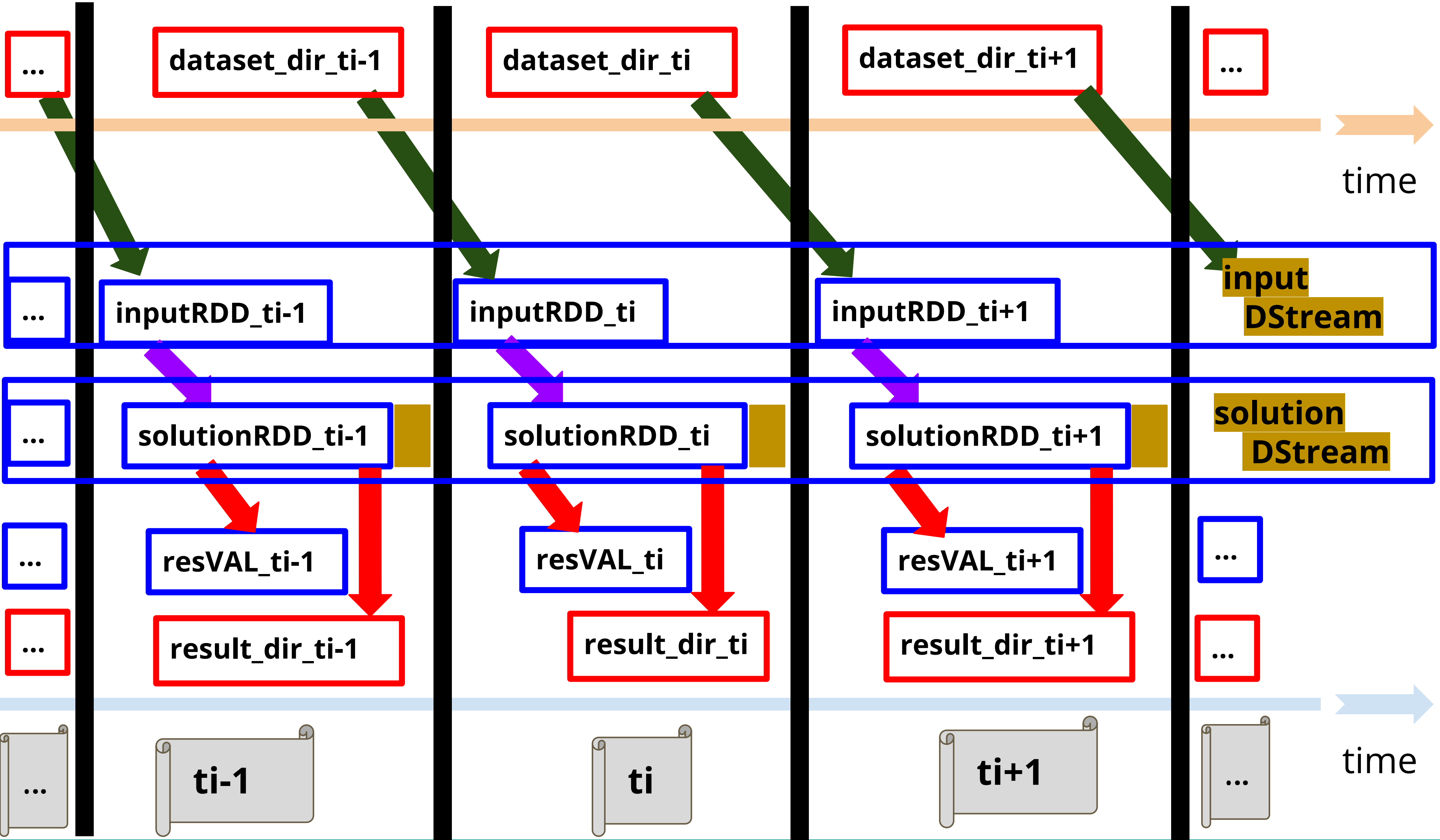
From RDDs to a DStream

And these **RDDs** computed over time
are abstracted in Spark Streaming
as a single **Discretized Stream (DStream)**

From RDDs to a DStream

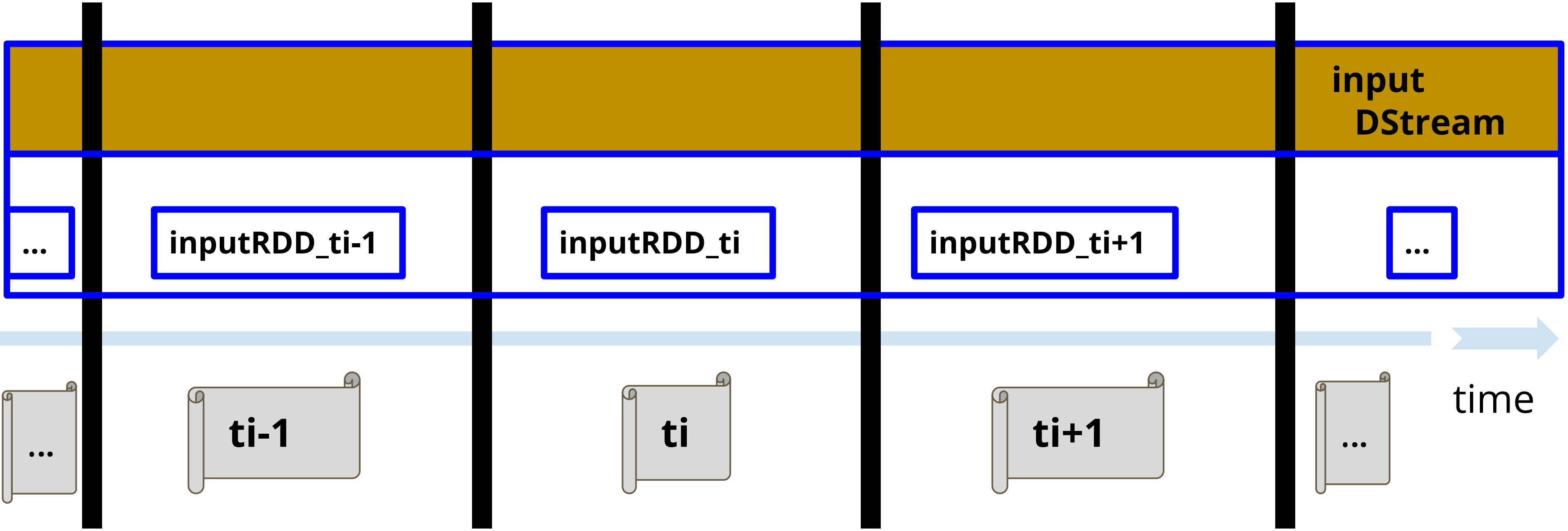


From RDDs to a DStream



From RDDs to a DStream

The best metaphor we can think of for explaining
the relationship between a **DStream** and its internal **RDDs**...



From RDDs to a DStream

...is a **train** and its **wagons**.

...

...

From RDDs to a DStream

Now, when we look back at our example
p02_introDStream.py
in Spark Streaming

From RDDs to a DStream

Now, when we look back at our example
p02_introDStream.py
in Spark Streaming

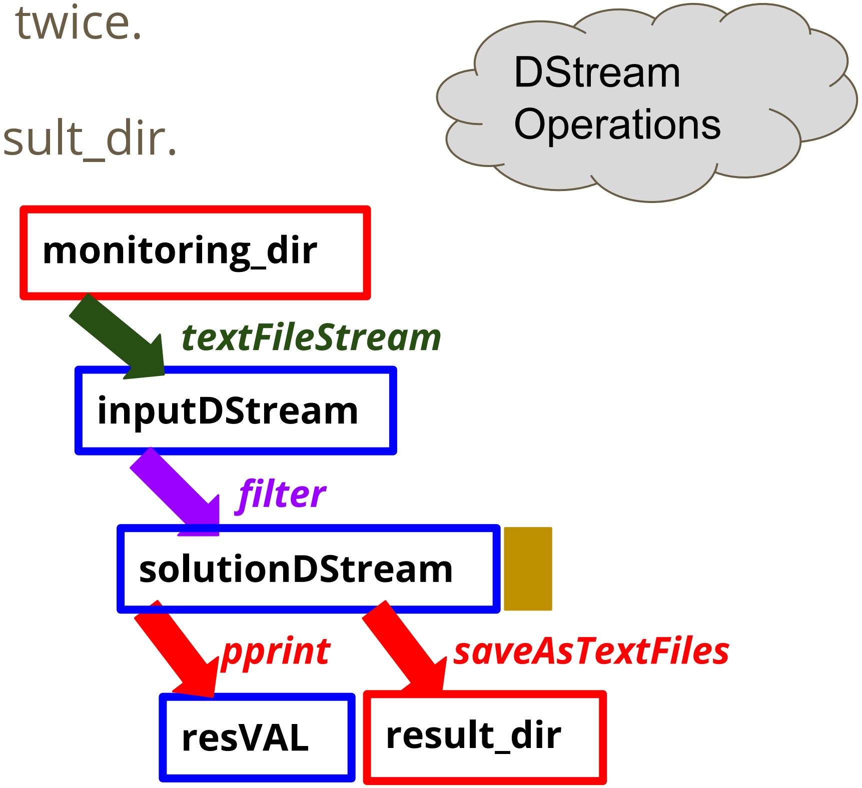
we observe that the reasoning is done at the level of **DStreams** (the train)
and not of its **internal RDDs** (its wagons).

From RDDs to a DStream

p02_introDStream.py

1. Read in all the lines of the files arriving in streaming to my_monitoring_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Print them by the screen.
5. Save them to the new directory my_result_dir.

A high level view of its operations is presented next:

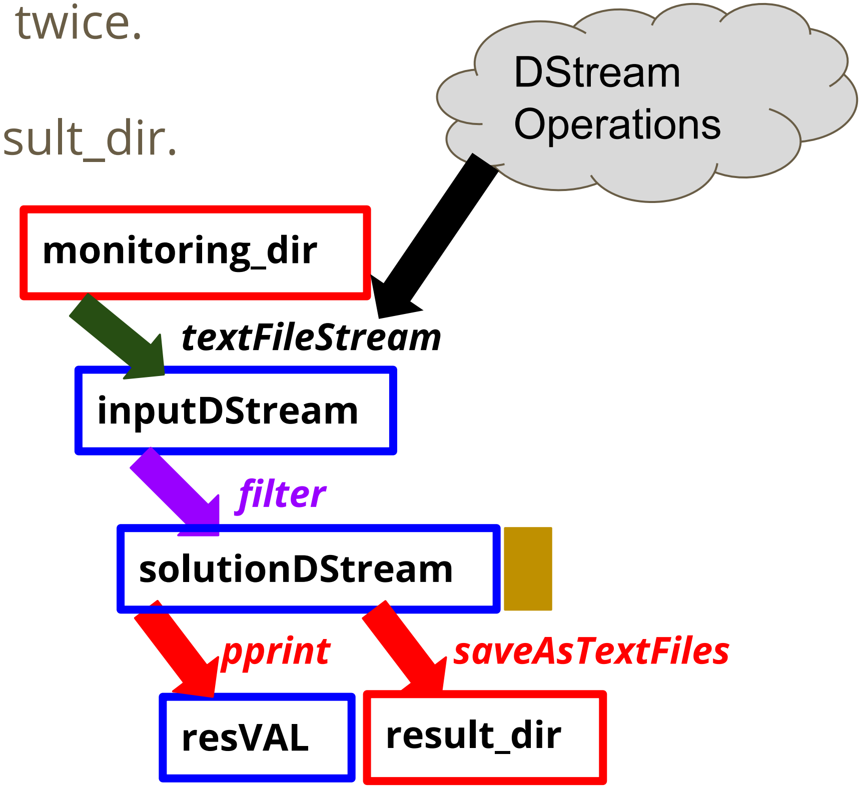


From RDDs to a DStream

p02_introDStream.py

1. Read in all the lines of the files arriving in streaming to my_monitoring_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Print them by the screen.
5. Save them to the new directory my_result_dir.

A high level view of its operations is presented next:

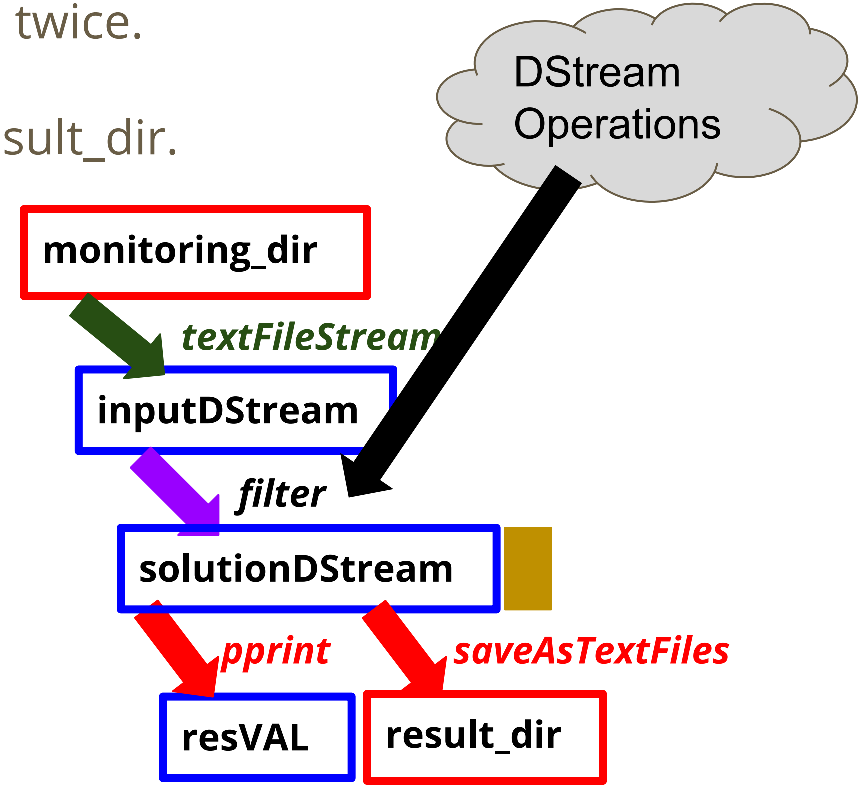


From RDDs to a DStream

p02_introDStream.py

1. Read in all the lines of the files arriving in streaming to my_monitoring_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Print them by the screen.
5. Save them to the new directory my_result_dir.

A high level view of its operations is presented next:

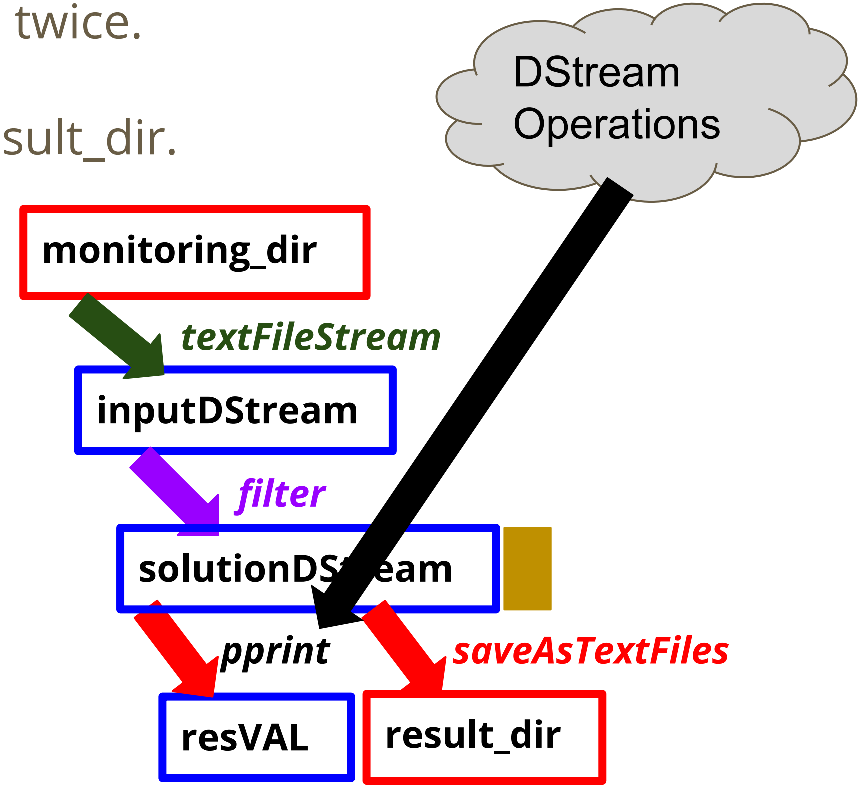


From RDDs to a DStream

p02_introDStream.py

1. Read in all the lines of the files arriving in streaming to my_monitoring_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Print them by the screen.
5. Save them to the new directory my_result_dir.

A high level view of its operations is presented next:

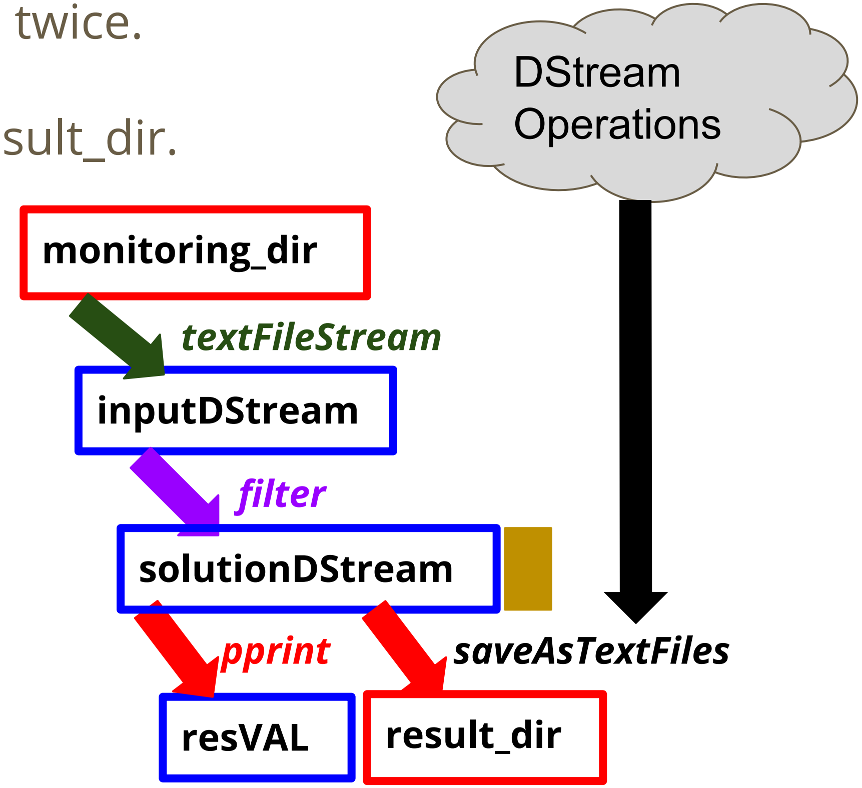


From RDDs to a DStream

p02_introDStream.py

1. Read in all the lines of the files arriving in streaming to my_monitoring_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Print them by the screen.
5. Save them to the new directory my_result_dir.

A high level view of its operations is presented next:



From RDDs to a DStream

But, when we run the application…

From RDDs to a DStream

But, when we run the application...

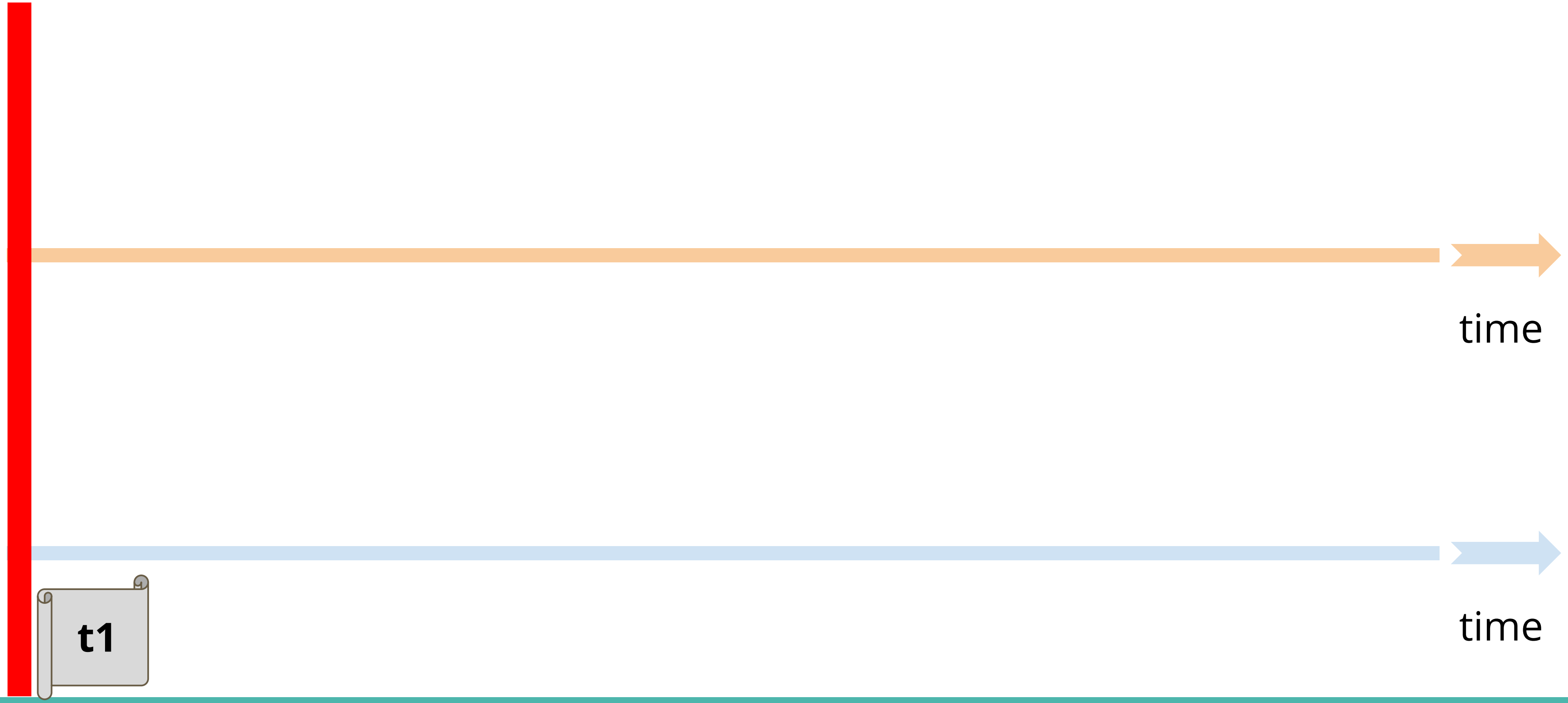
we see the **DStream** reasoning (train)
is indeed applied to each **RDD** (wagon)
as they arrive on each time interval!

From RDDs to a DStream

Time Interval t_1

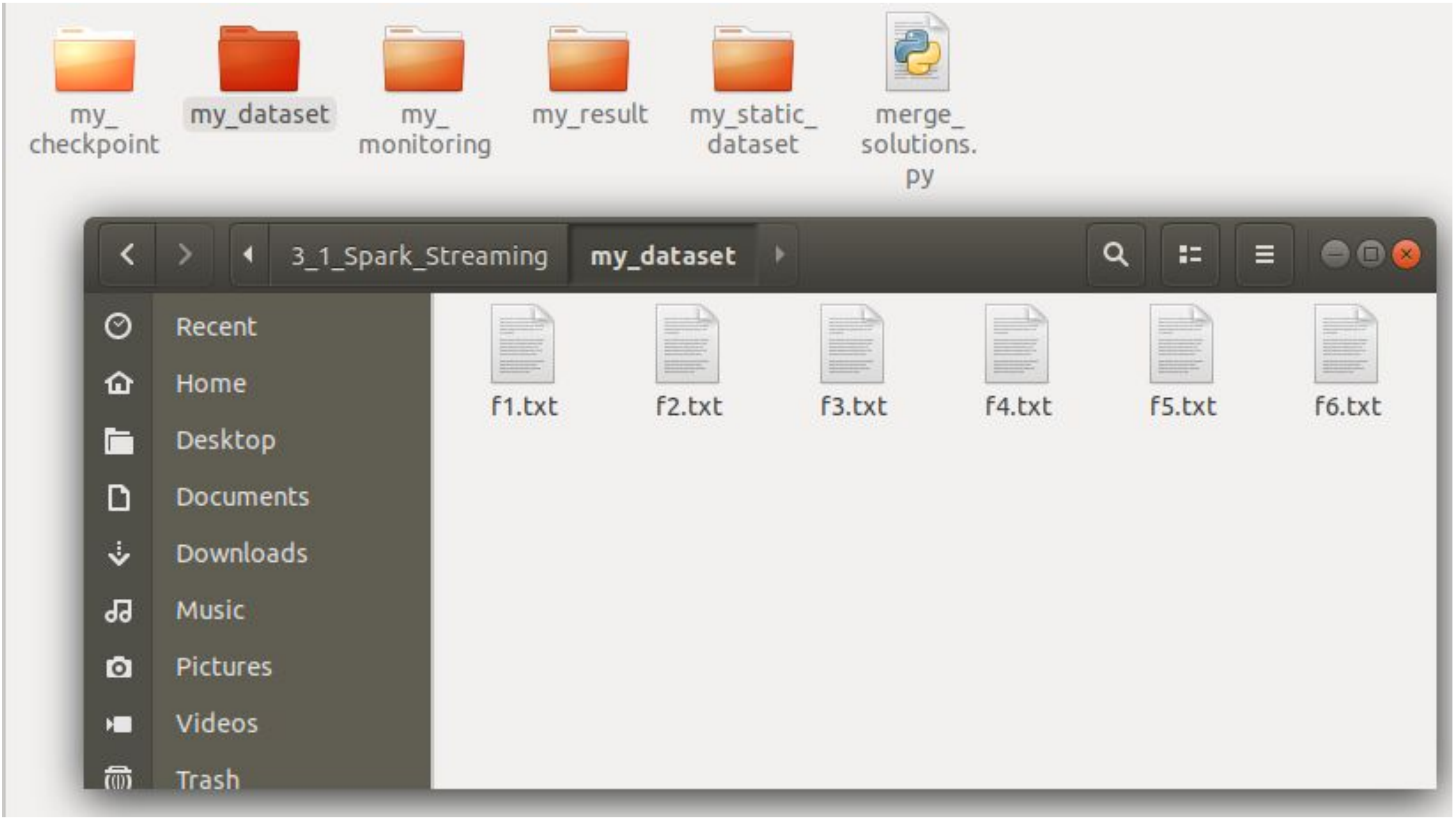
From RDDs to a DStream

The Spark Streaming Context is started.
But we set it to wait for an entire time interval before we start working.



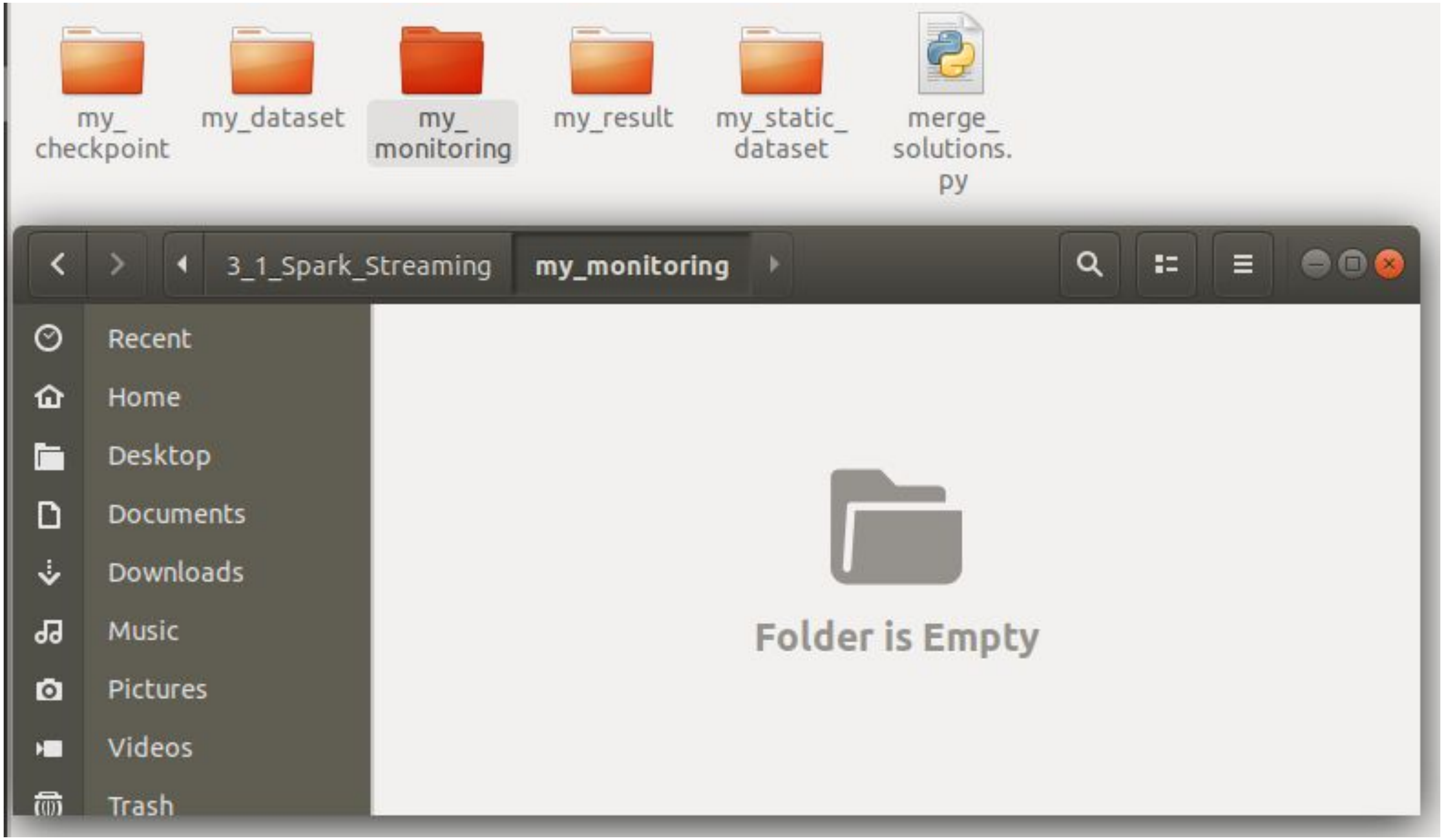
From RDDs to a DStream

As we can see, dataset_dir contains the set of files to be transferred.



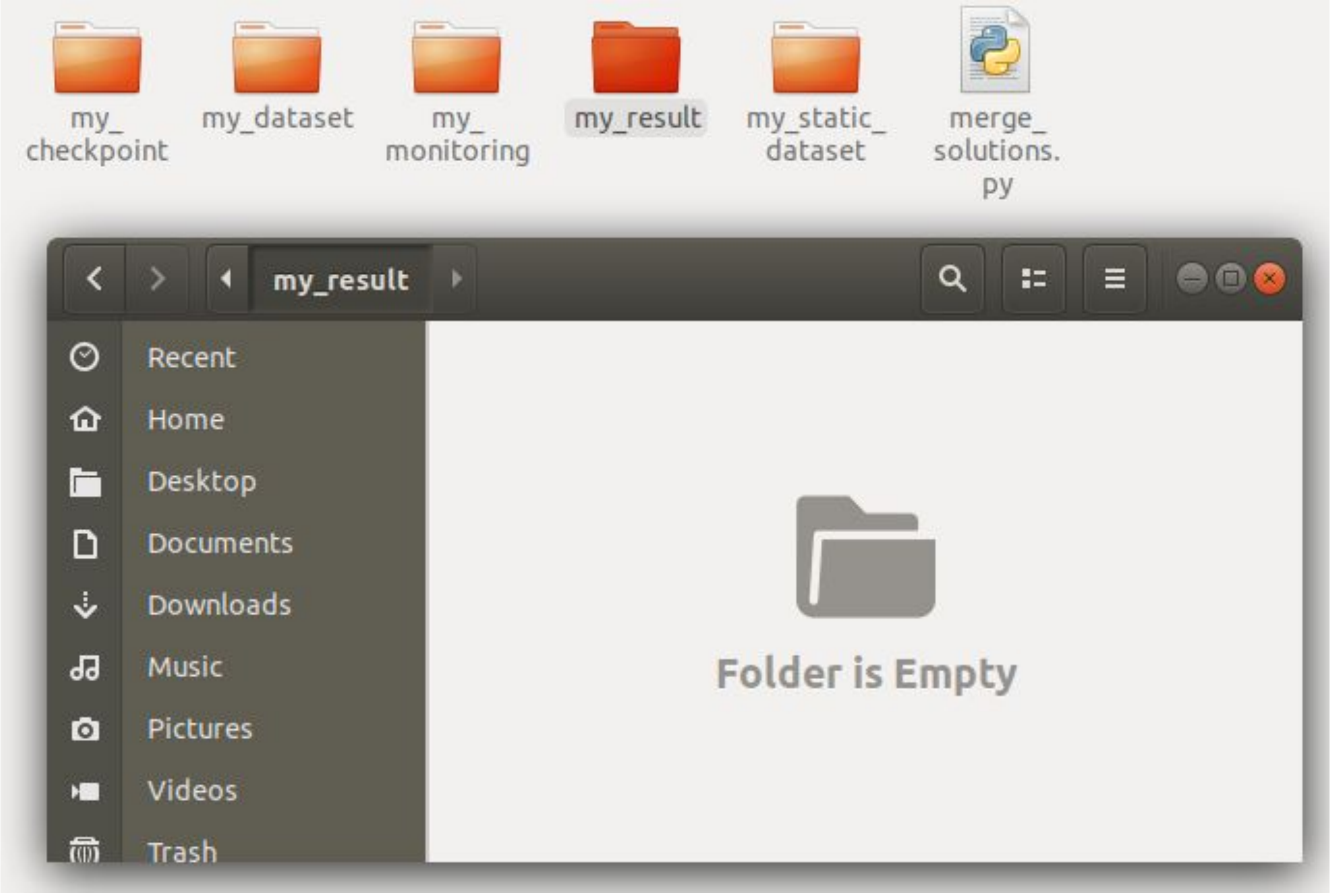
From RDDs to a DStream

As we can see, monitoring_dir is empty.



From RDDs to a DStream

As we can see, result_dir is empty.

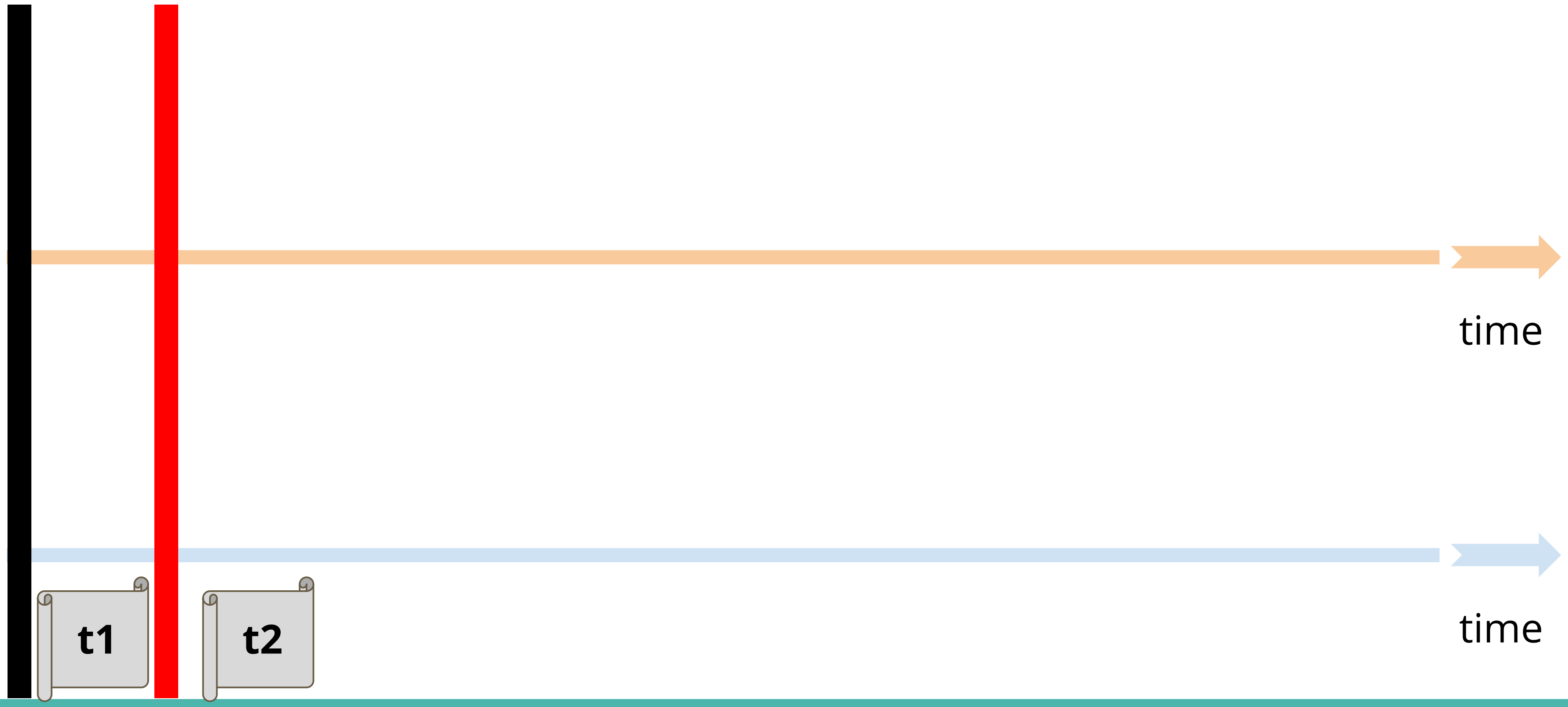


From RDDs to a DStream

Time Interval t_2

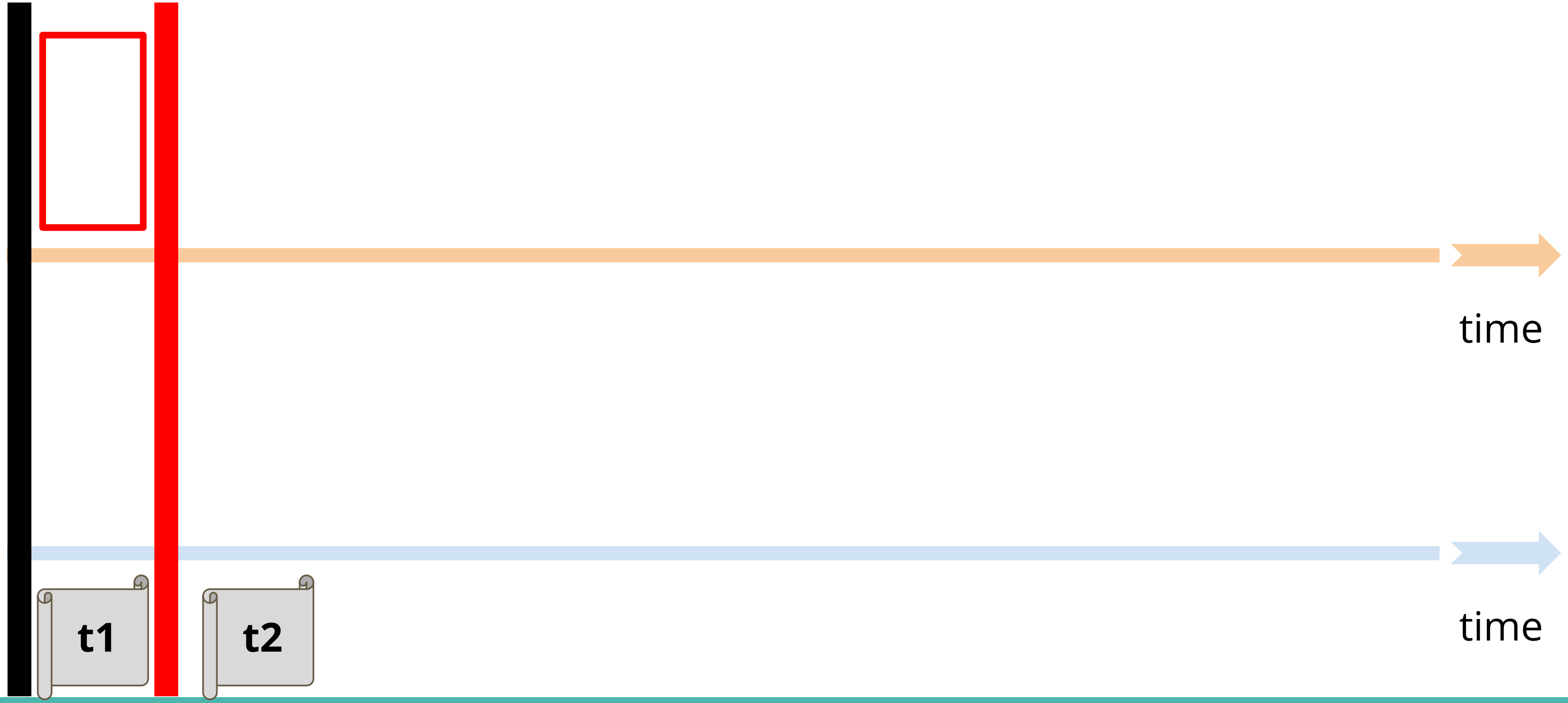
From RDDs to a DStream

Next time interval is checked by the Spark Streaming Context.



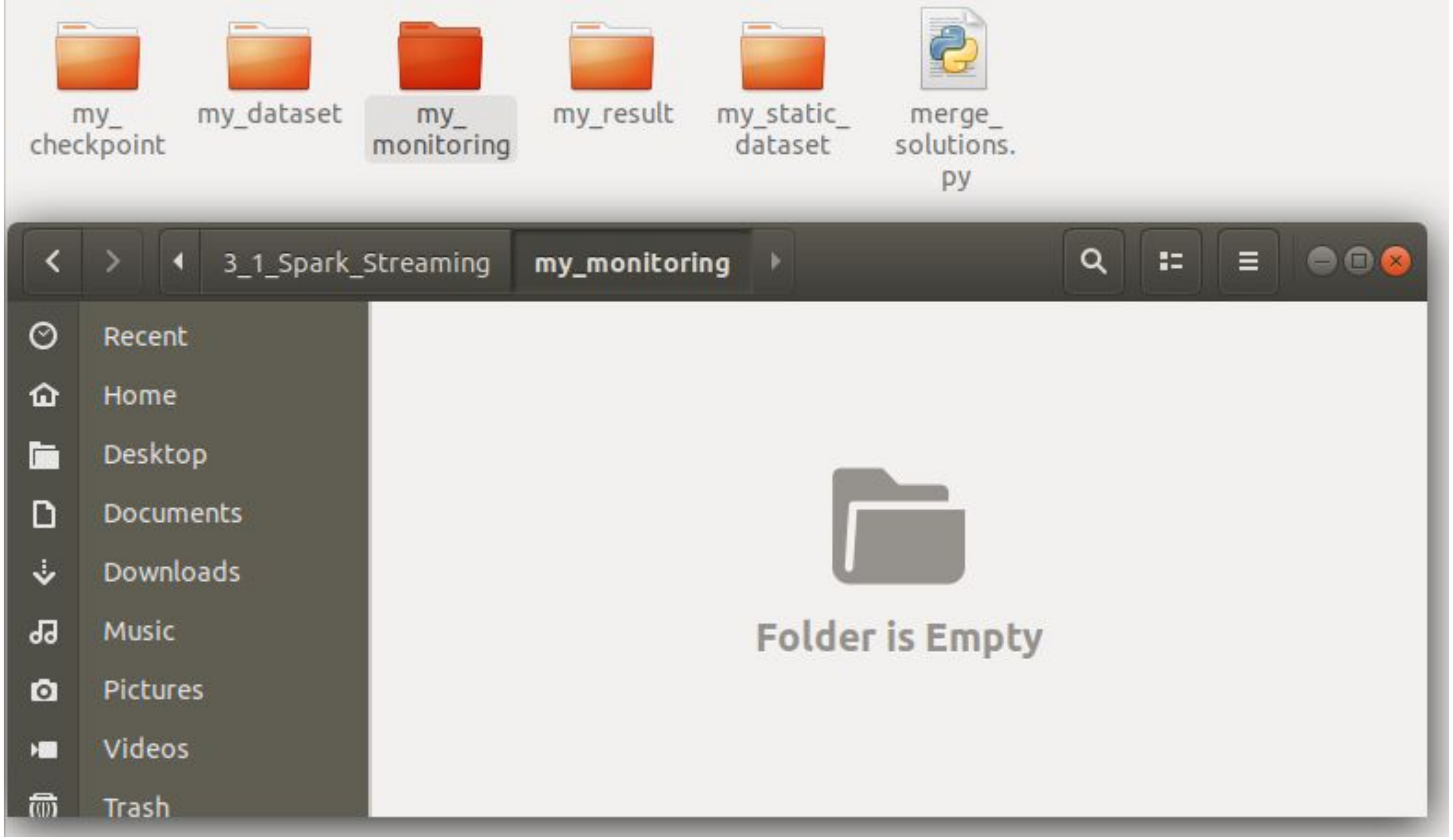
From RDDs to a DStream

It does not find any new file in **monitoring_dir**



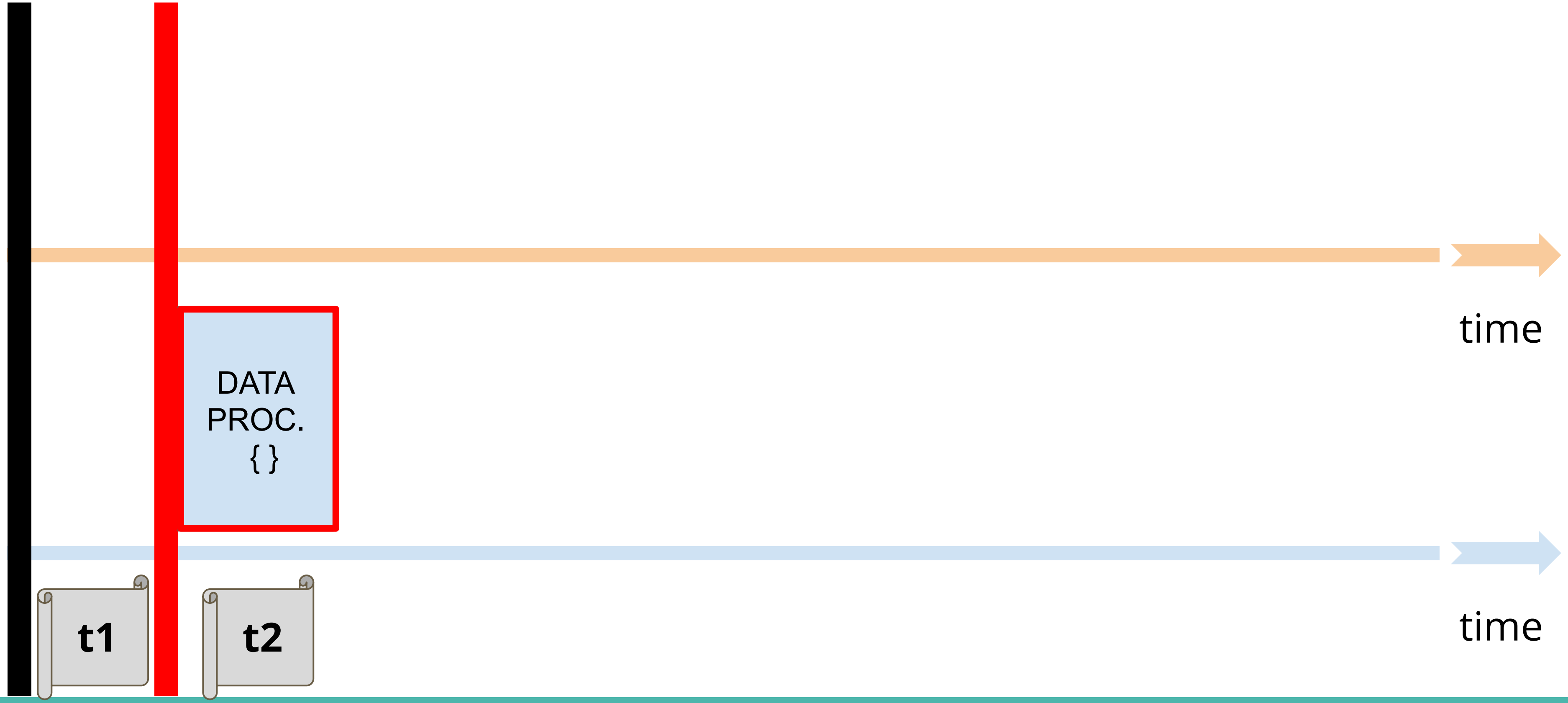
From RDDs to a DStream

It does not find any new file in **monitoring_dir**



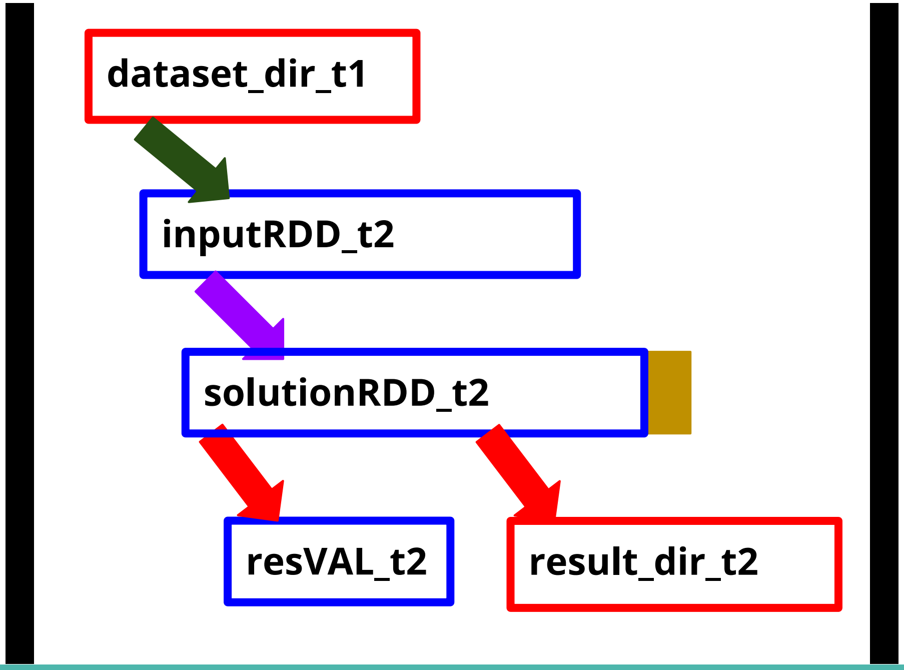
From RDDs to a DStream

As there is no new content, there is nothing to reason with in **RDD_t2** (wagon 2) of the **DStream** (train).



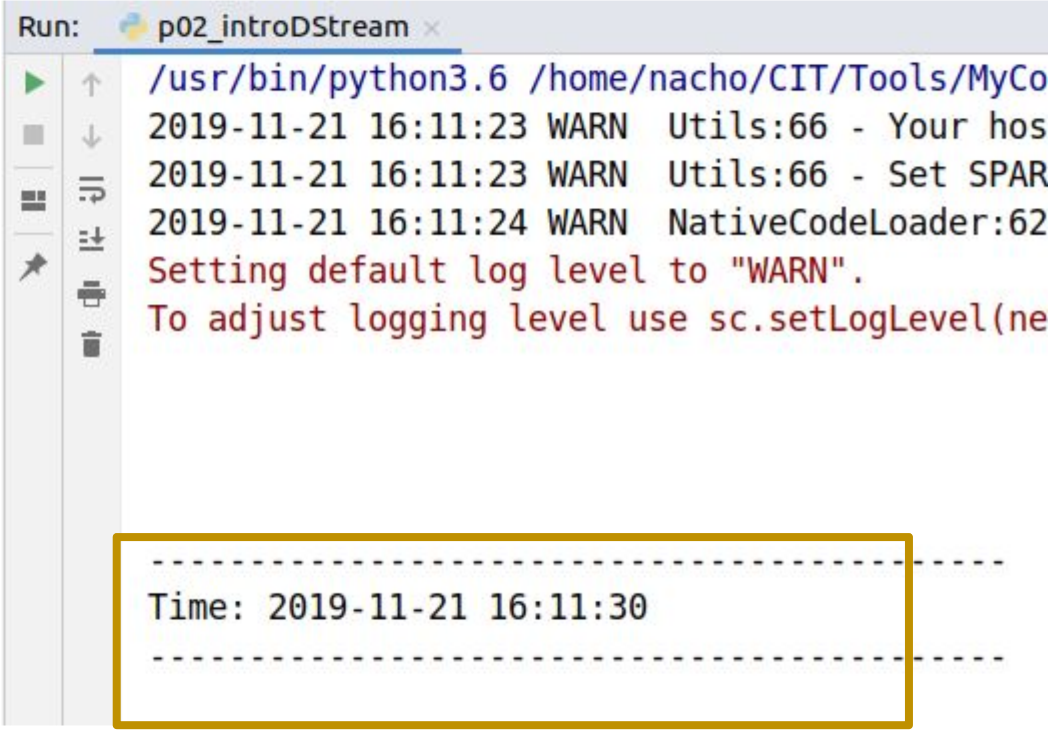
From RDDs to a DStream

As there is no new content, there is nothing to reason with in **RDD_t2** (wagon 2) of the **DStream** (train).



From RDDs to a DStream

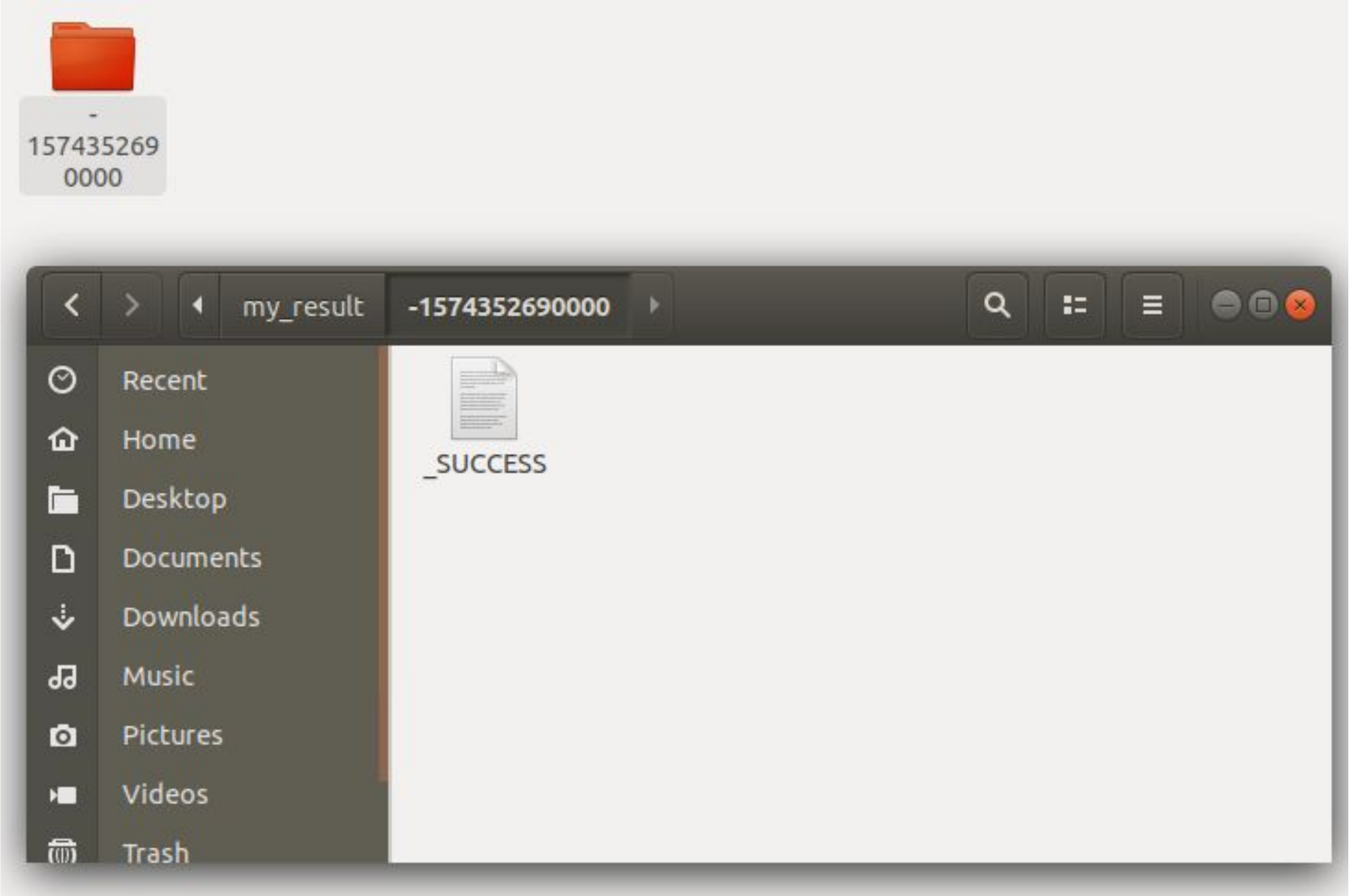
It **pprints** the results by the screen



```
/usr/bin/python3.6 /home/nacho/CIT/Tools/MyCo
2019-11-21 16:11:23 WARN  Utils:66 - Your hos
2019-11-21 16:11:23 WARN  Utils:66 - Set SPAR
2019-11-21 16:11:24 WARN  NativeCodeLoader:62
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(ne



-------------------------------------------
Time: 2019-11-21 16:11:30
-------------------------------------------
```

From RDDs to a DStream

It **saveAsTextFiles** by creating a new sub-folder in **result_dir**.

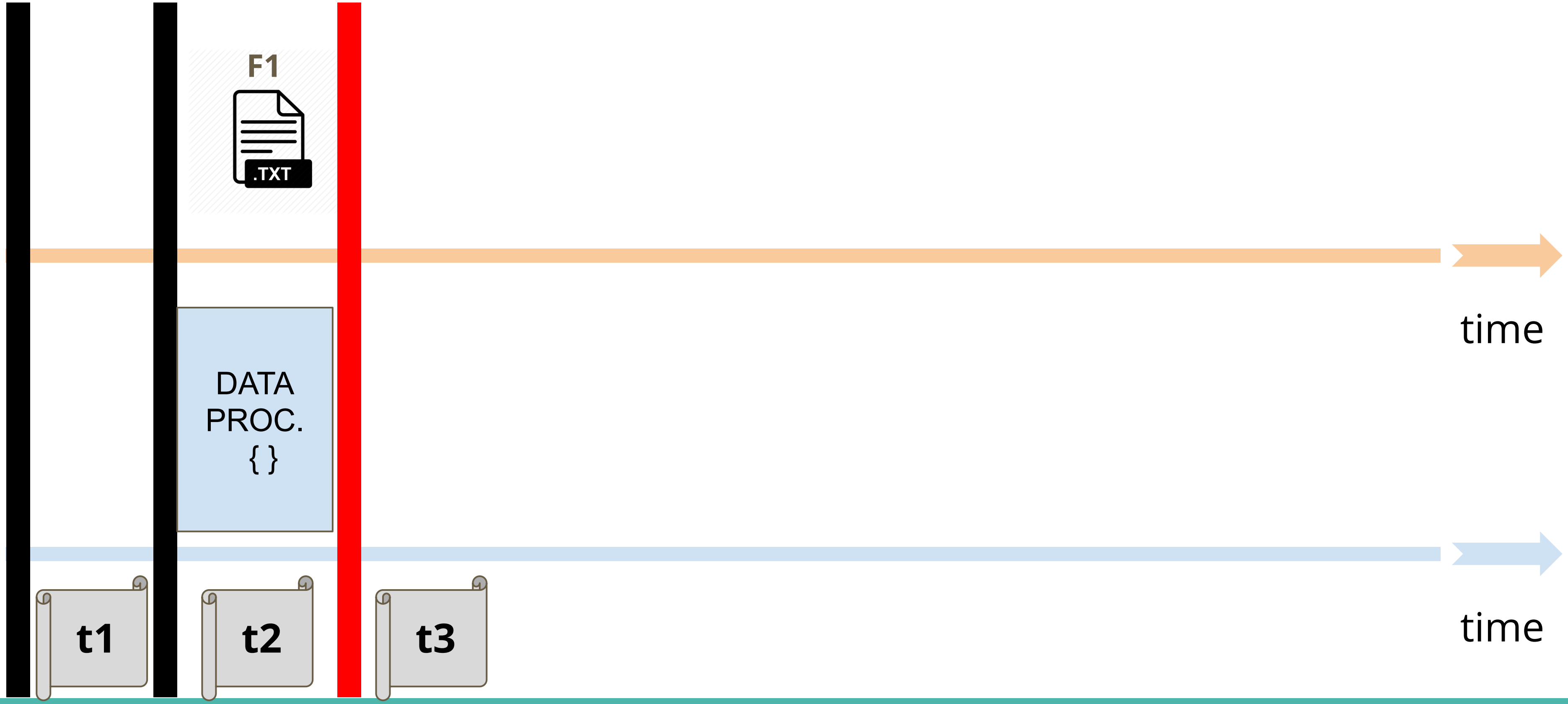


From RDDs to a DStream

Time Interval t_3

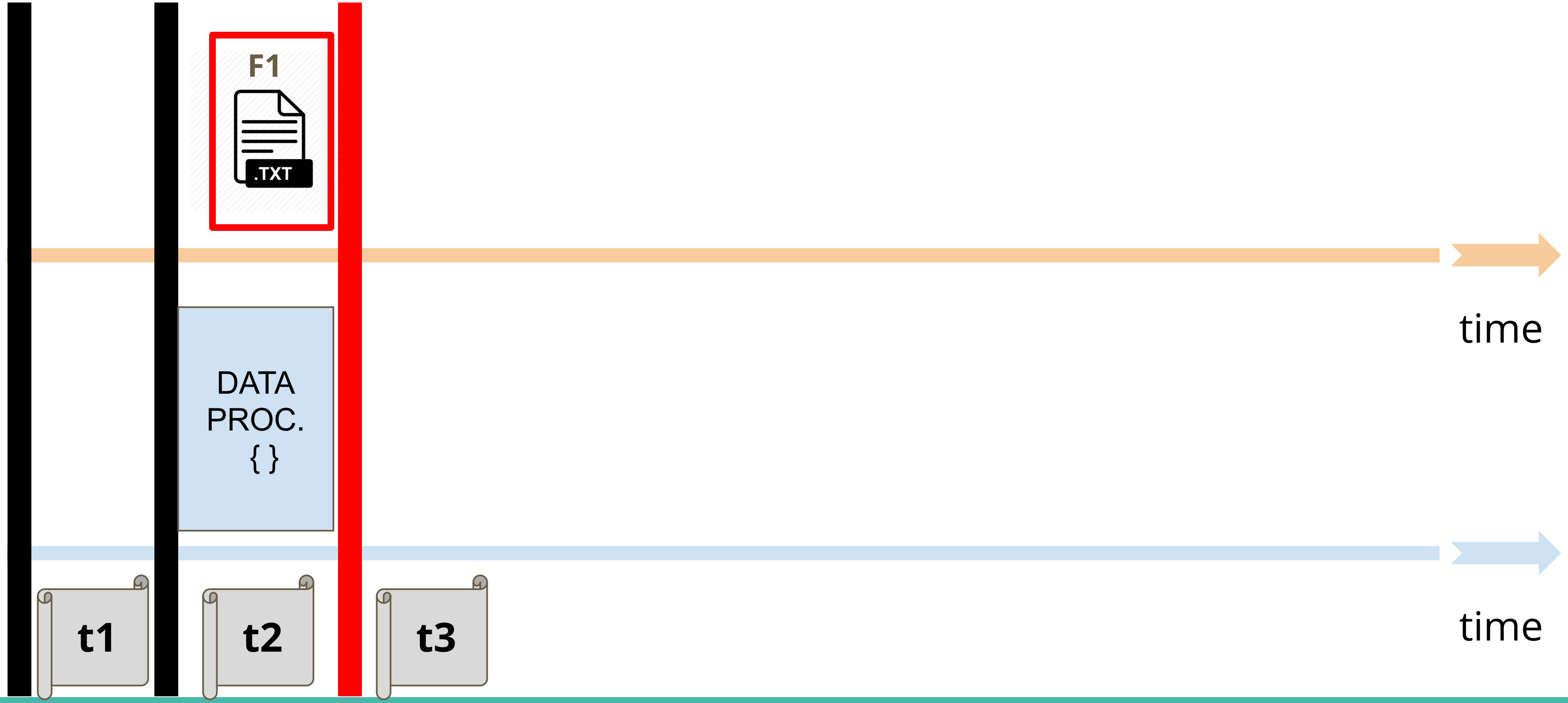
From RDDs to a DStream

Next time interval is checked by the Spark Streaming Context.



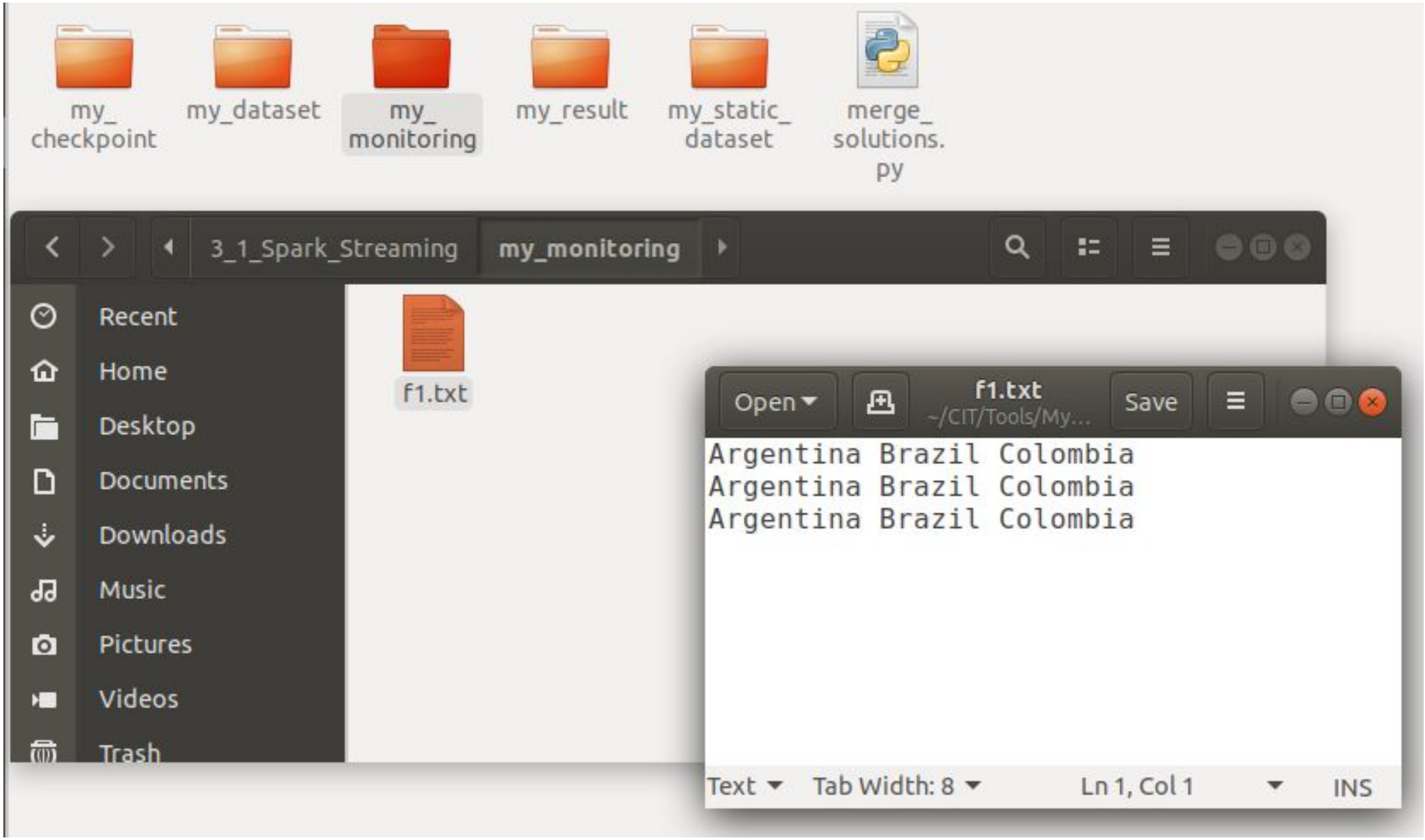
From RDDs to a DStream

It finds the new file **f1.txt** in **monitoring_dir**



From RDDs to a DStream

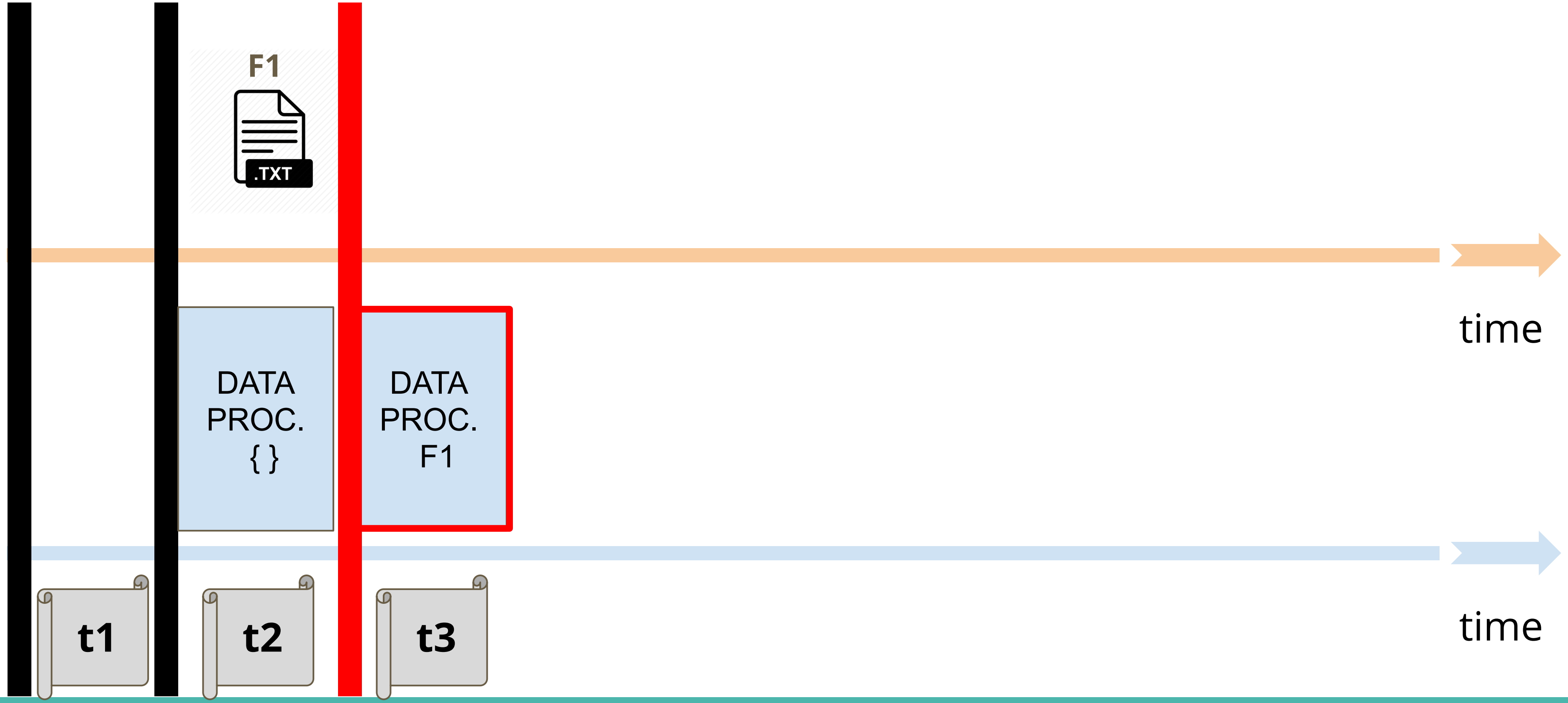
File **f1.txt** is detected as a new file in **monitoring_dir**



```
Argentina Brazil Colombia
Argentina Brazil Colombia
Argentina Brazil Colombia
```

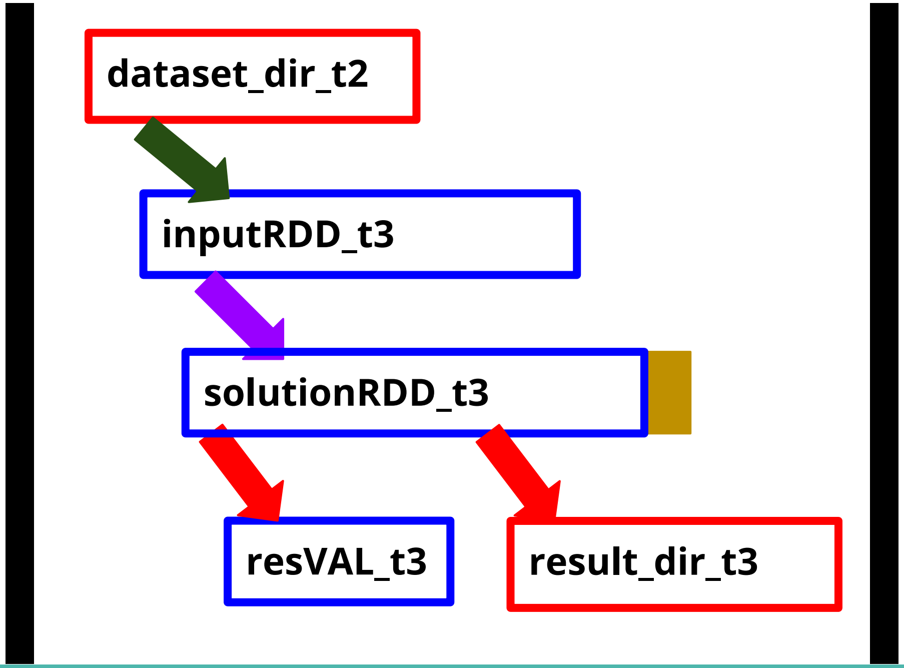
From RDDs to a DStream

It processes the file by reasoning with **RDD_t3** (wagon 3) of the **DStream** (train).



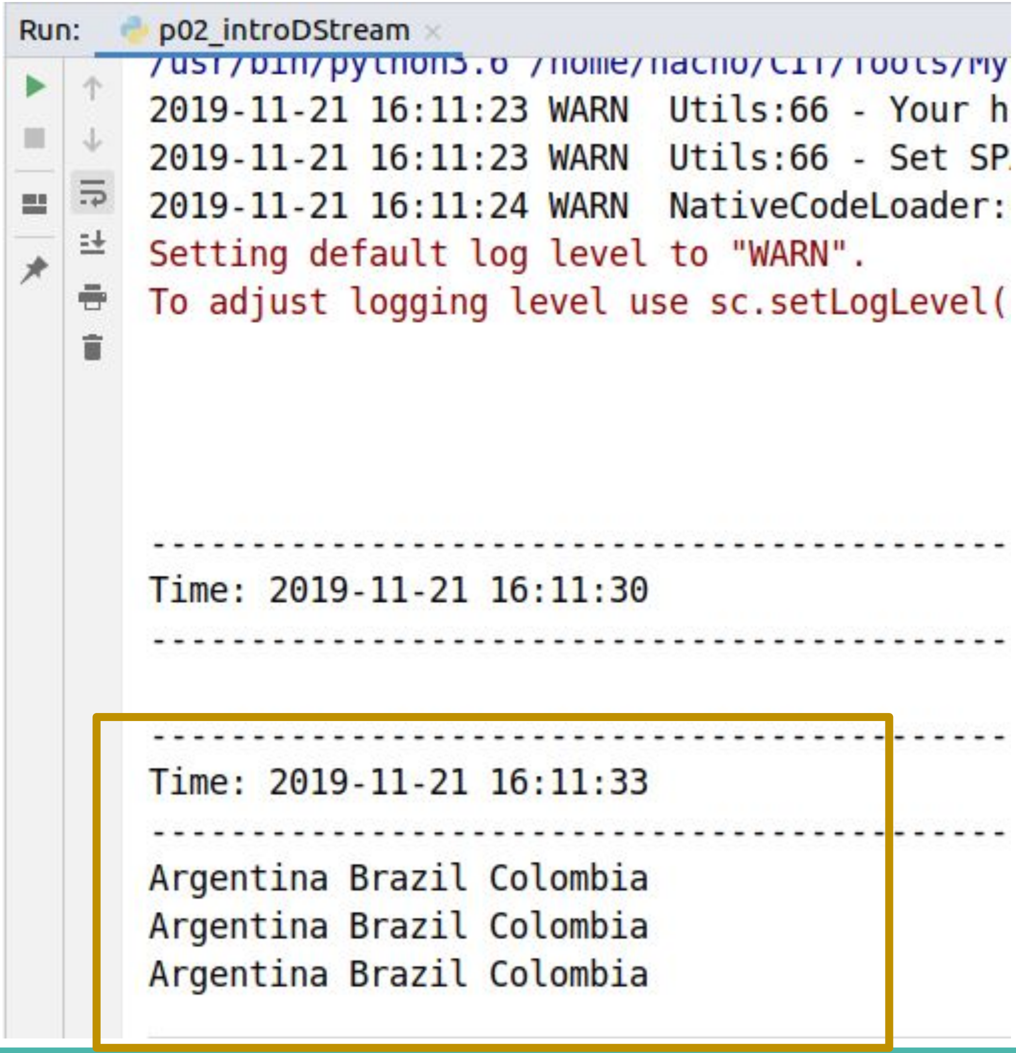
From RDDs to a DStream

It processes the file by reasoning with **RDD_t3** (wagon 3) of the **DStream** (train).



From RDDs to a DStream

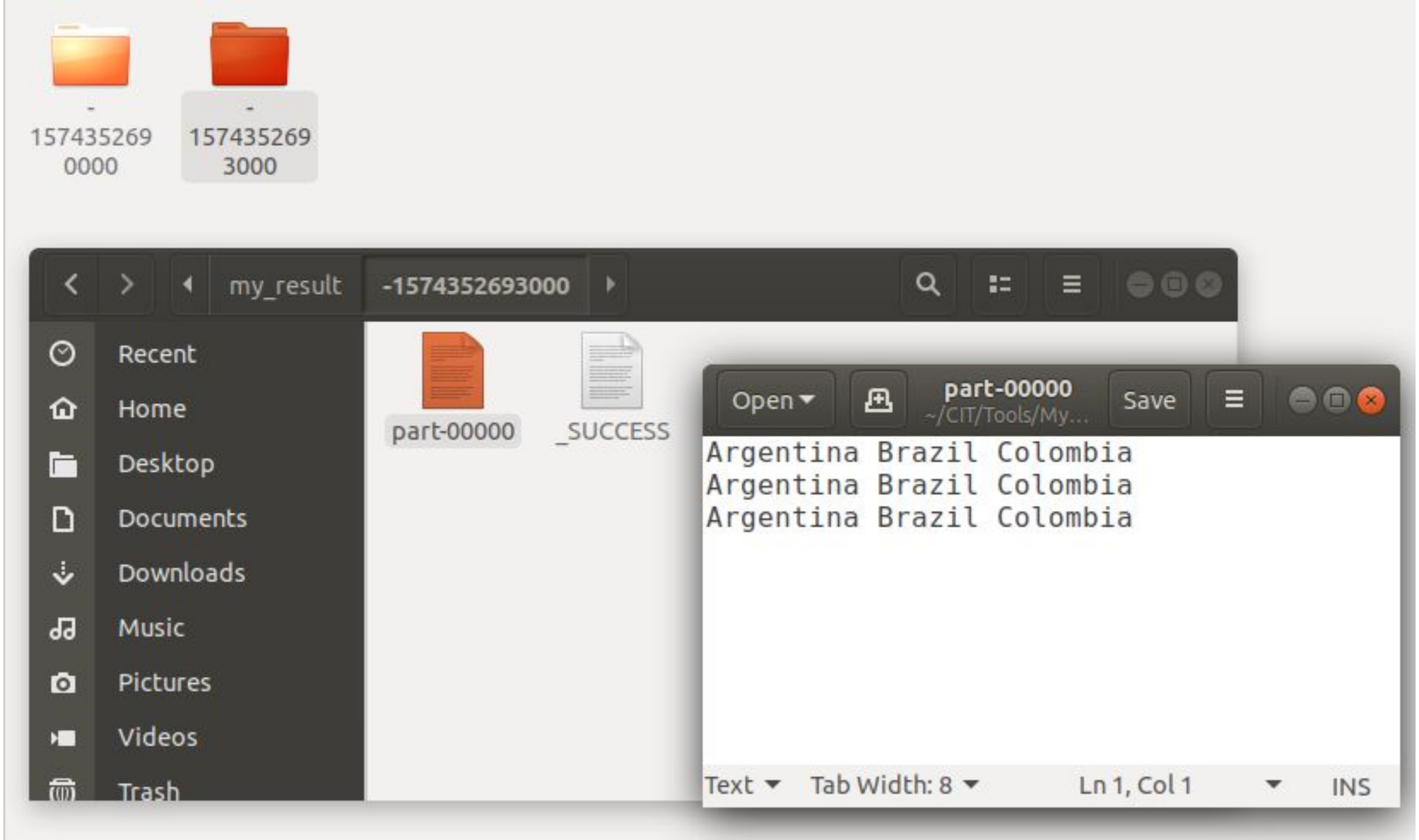
It **pprints** the results by the screen



```
/usr/bin/python3.6 /home/nacho/CIT/Tools/My
2019-11-21 16:11:23 WARN  Utils:66 - Your h
2019-11-21 16:11:23 WARN  Utils:66 - Set SP
2019-11-21 16:11:24 WARN  NativeCodeLoader:
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(



-------------------------------------------
Time: 2019-11-21 16:11:30
-------------------------------------------

-------------------------------------------
Time: 2019-11-21 16:11:33
-------------------------------------------
Argentina Brazil Colombia
Argentina Brazil Colombia
Argentina Brazil Colombia
```

From RDDs to a DStream

It **saveAsTextFiles** by creating a new sub-folder in **result_dir**.



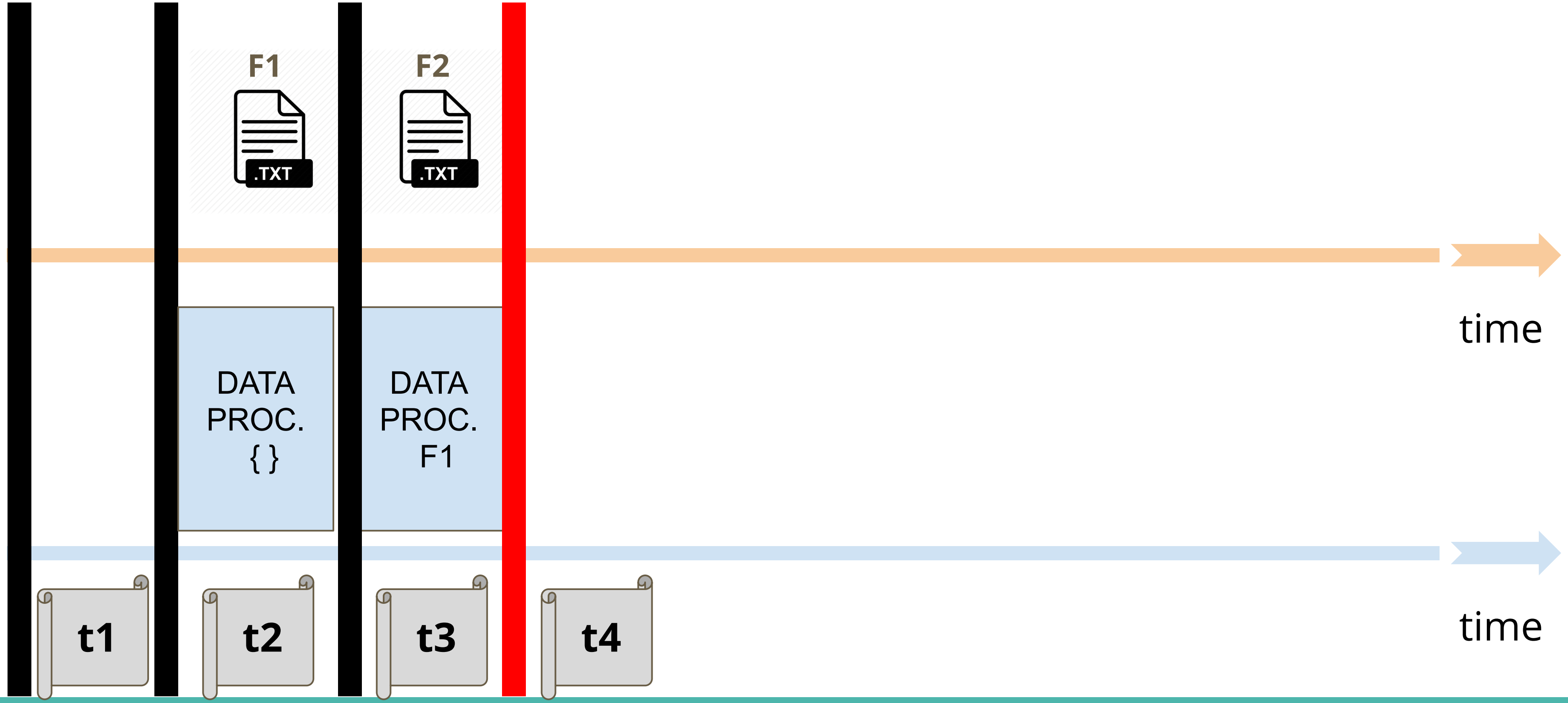
```
Argentina Brazil Colombia
Argentina Brazil Colombia
Argentina Brazil Colombia
```

From RDDs to a DStream

Time Interval t_4

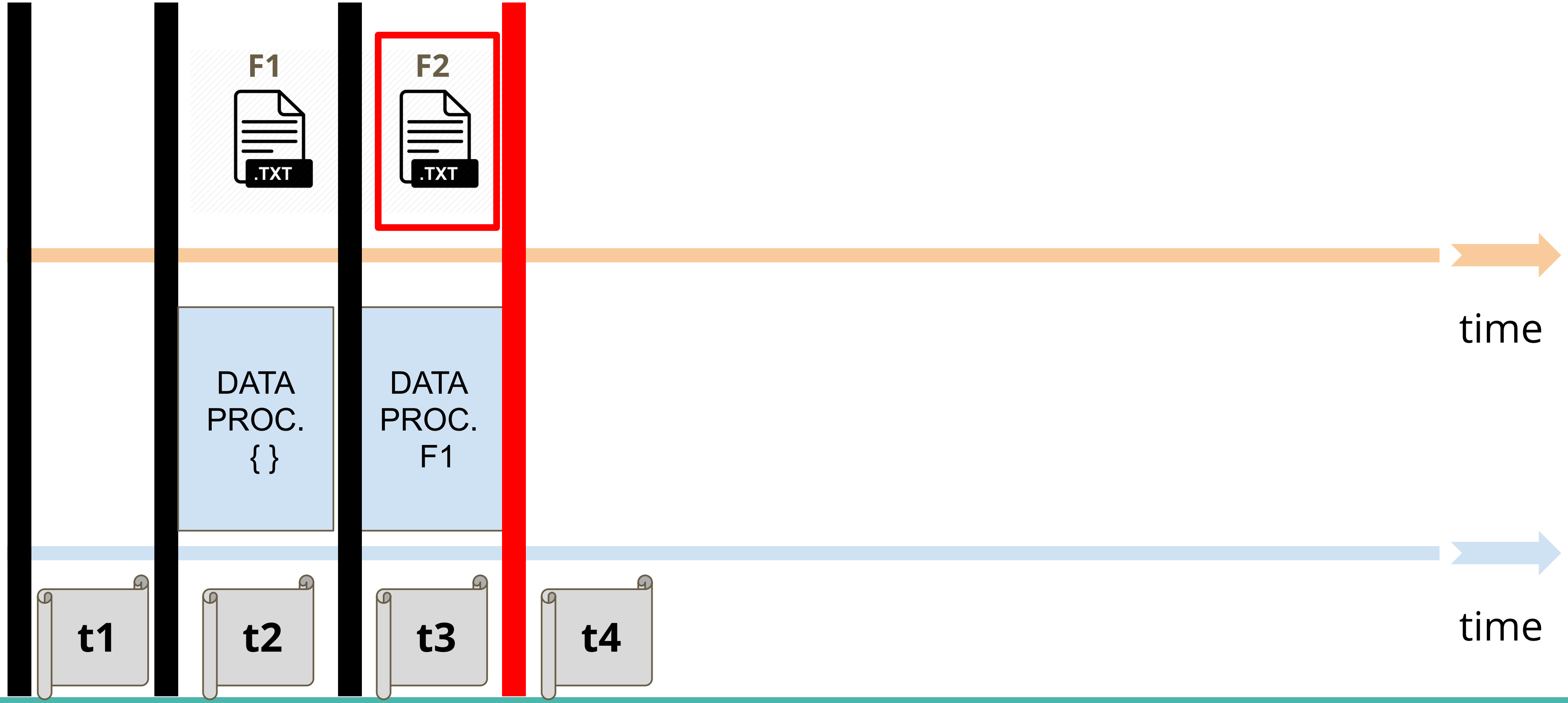
From RDDs to a DStream

Next time interval is checked by the Spark Streaming Context.



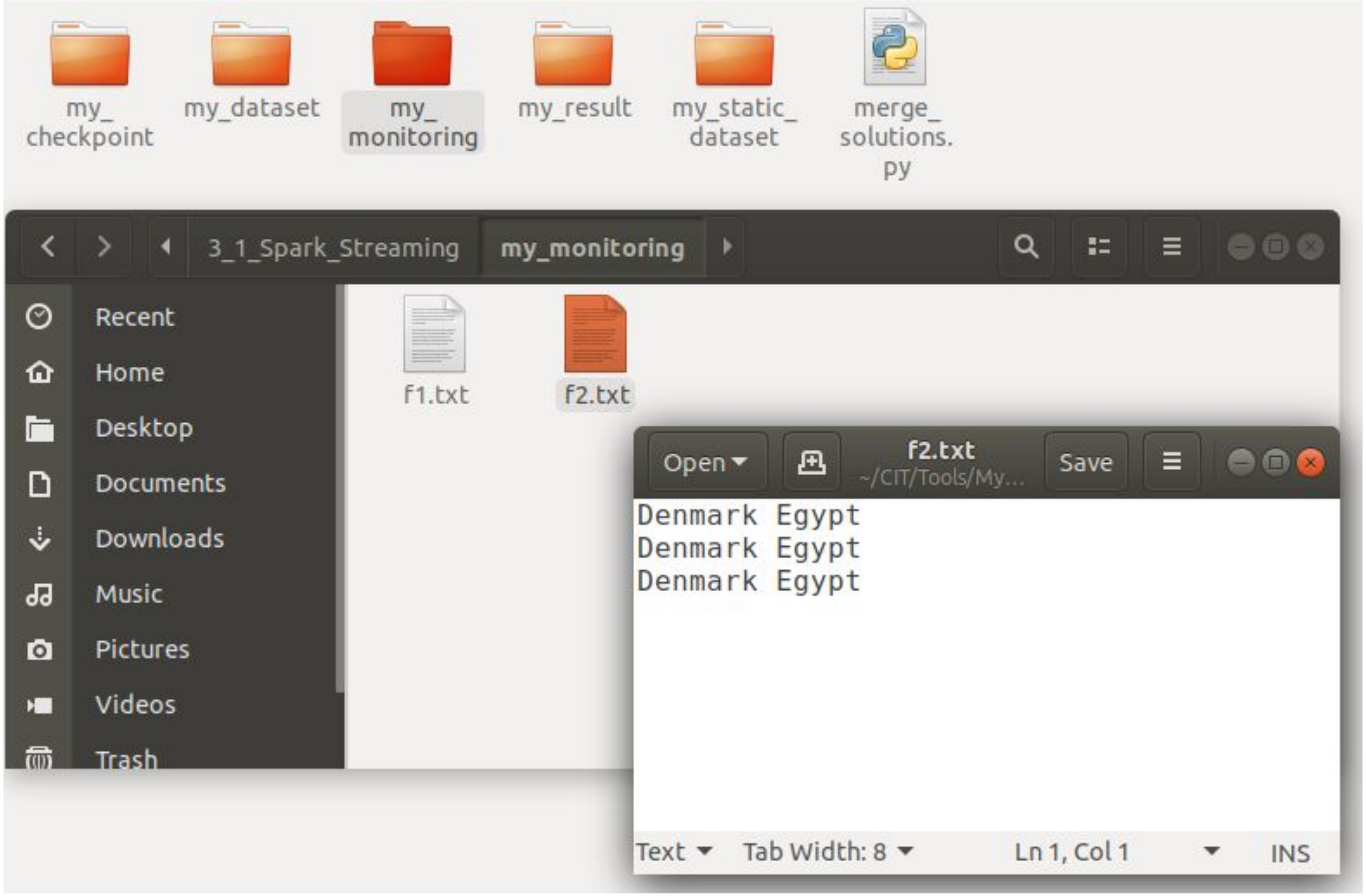
From RDDs to a DStream

It finds the new file **f2.txt** in **monitoring_dir**



From RDDs to a DStream

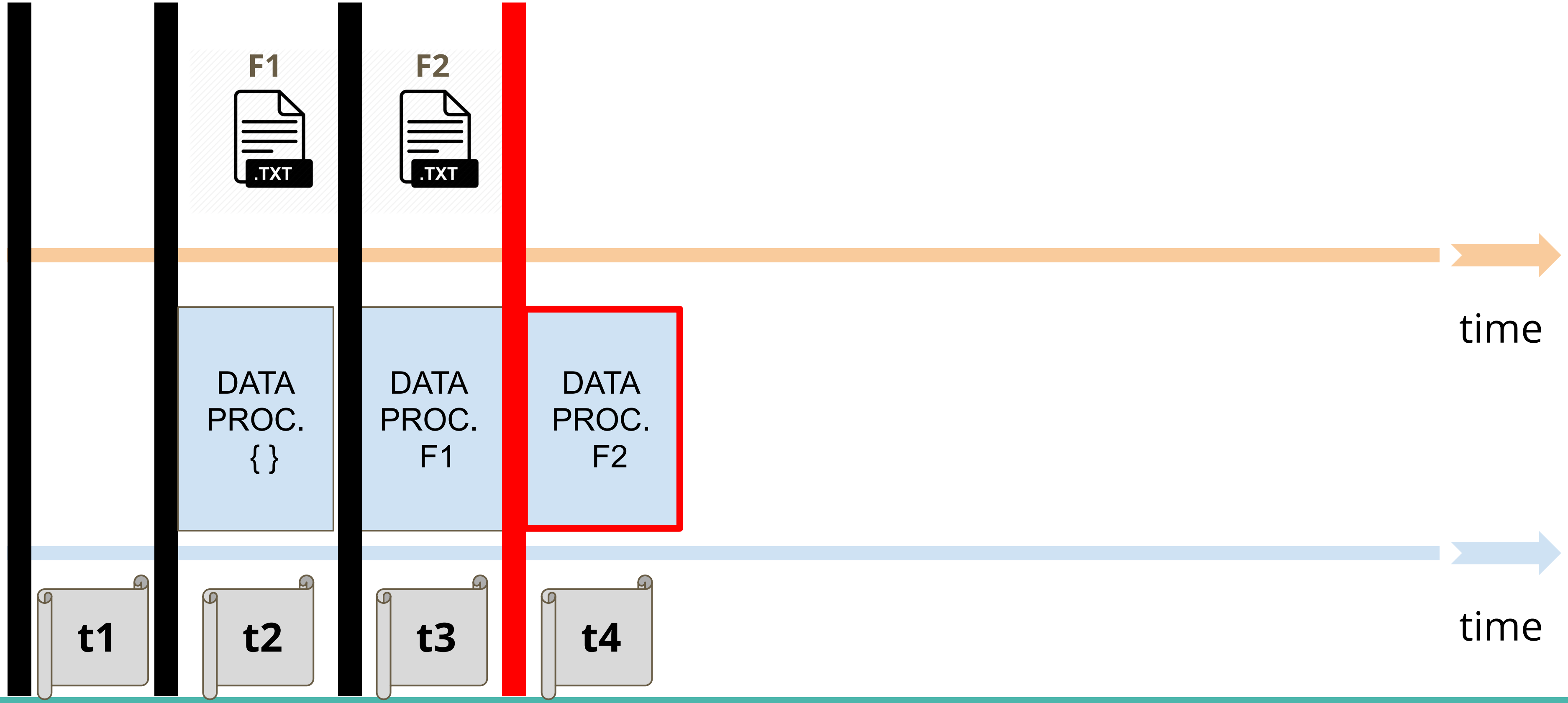
File **f2.txt** is detected as a new file in **monitoring_dir**



```
Denmark Egypt
Denmark Egypt
Denmark Egypt
```

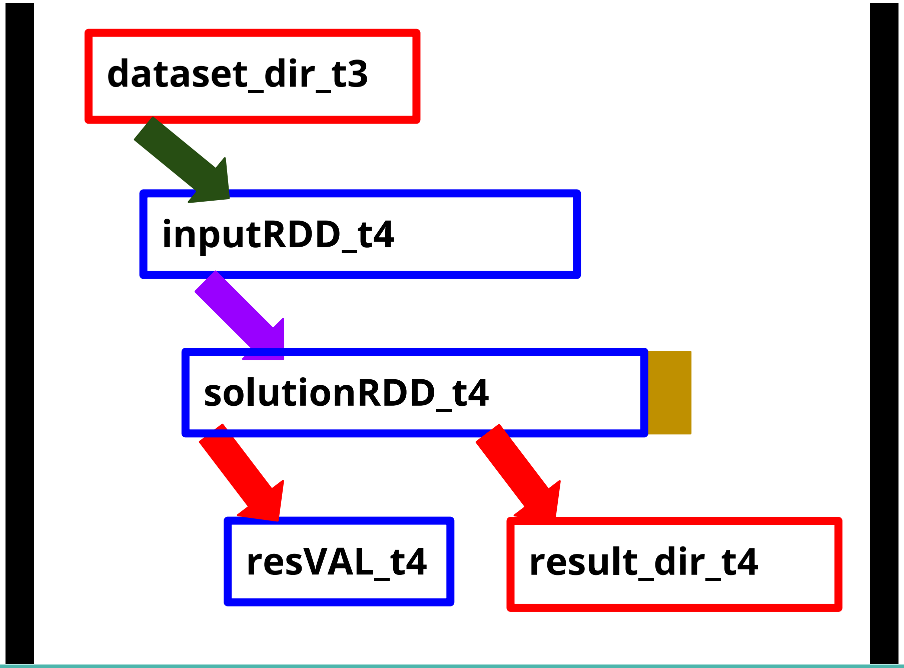
From RDDs to a DStream

It processes the file by reasoning with **RDD_t4** (wagon 4) of the **DStream** (train).



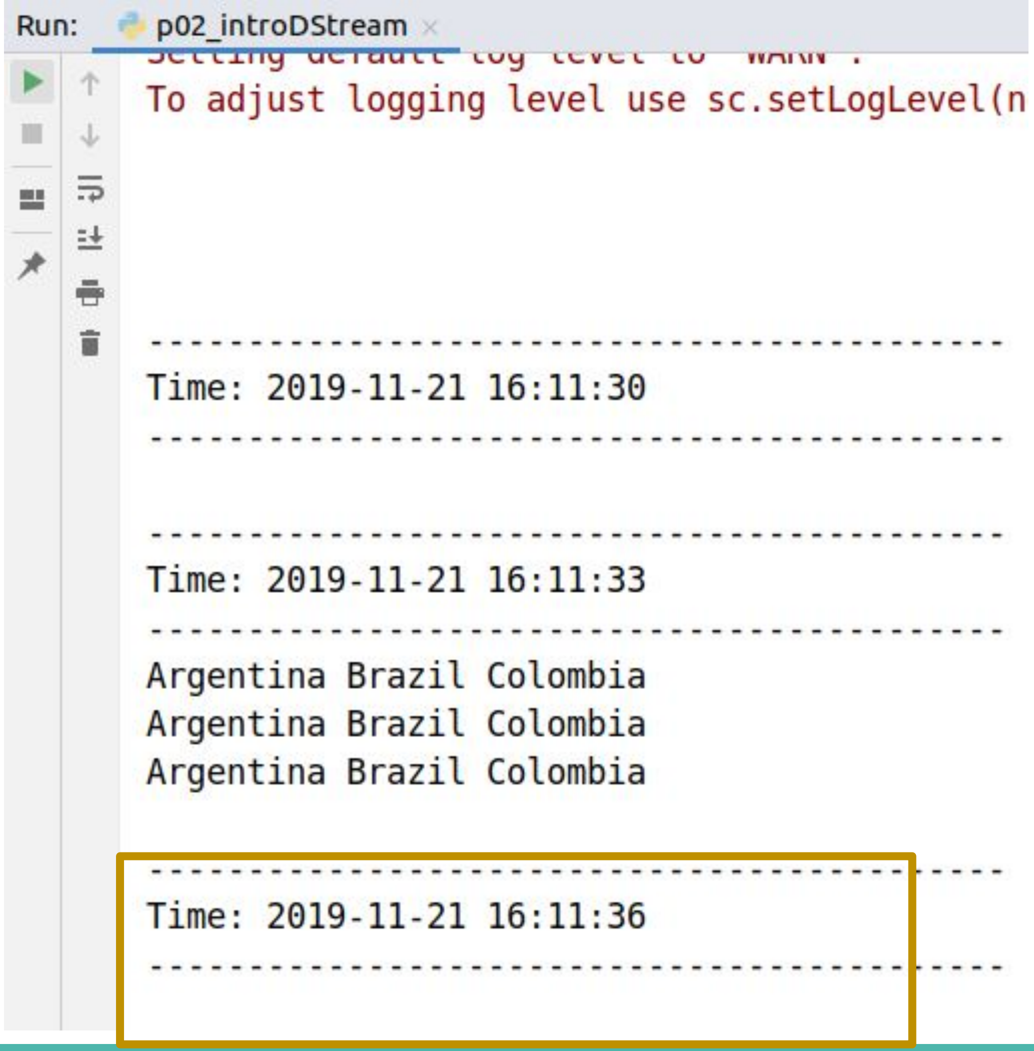
From RDDs to a DStream

It processes the file by reasoning with **RDD_t4** (wagon 4) of the **DStream** (train).



From RDDs to a DStream

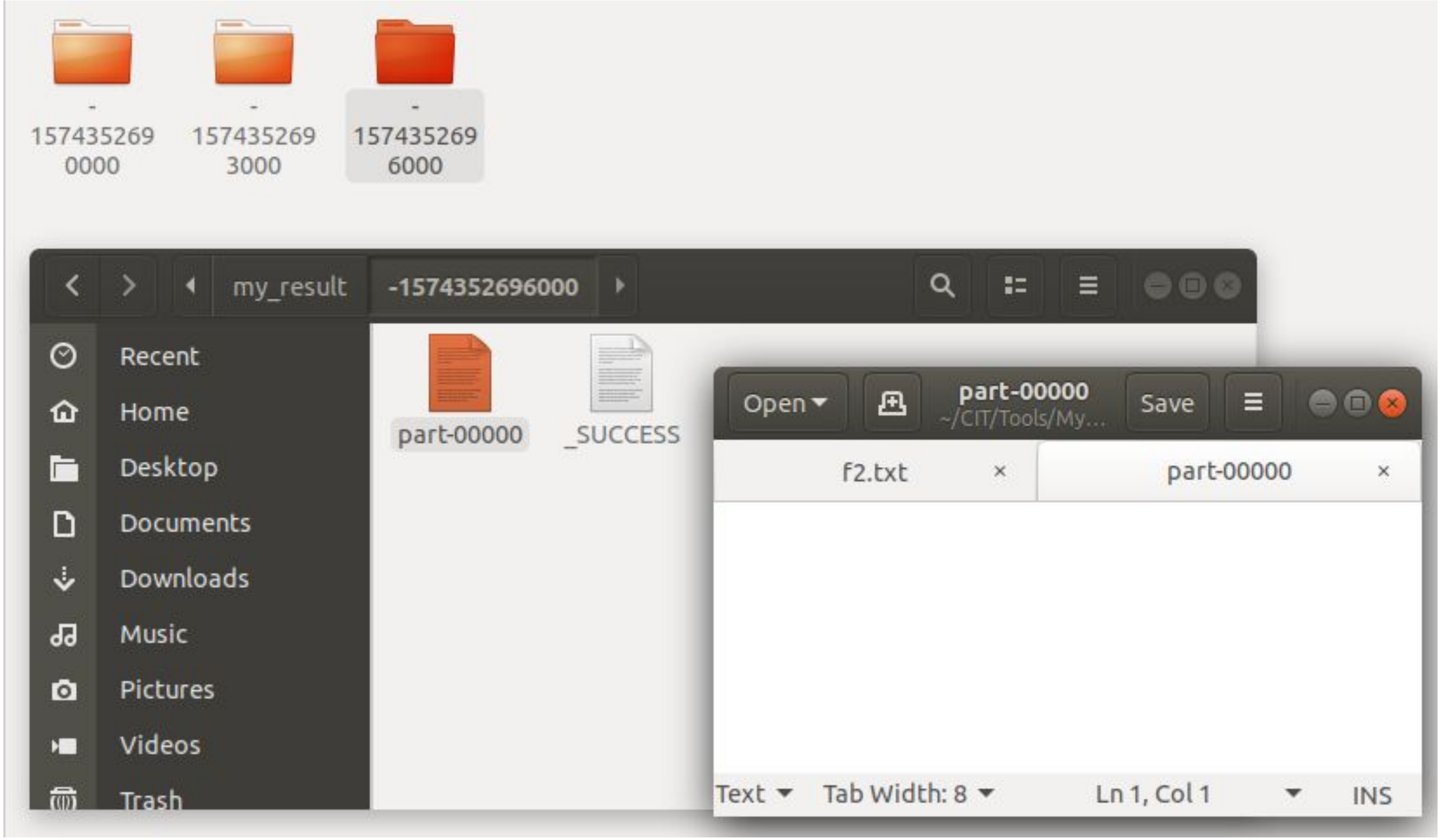
It **pprints** the results by the screen



```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(n



-------------------------------------------
Time: 2019-11-21 16:11:30
-------------------------------------------

-------------------------------------------
Time: 2019-11-21 16:11:33
-------------------------------------------
Argentina Brazil Colombia
Argentina Brazil Colombia
Argentina Brazil Colombia

-------------------------------------------
Time: 2019-11-21 16:11:36
-------------------------------------------
```

From RDDs to a DStream

It **saveAsTextFiles** by creating a new sub-folder in **result_dir**.

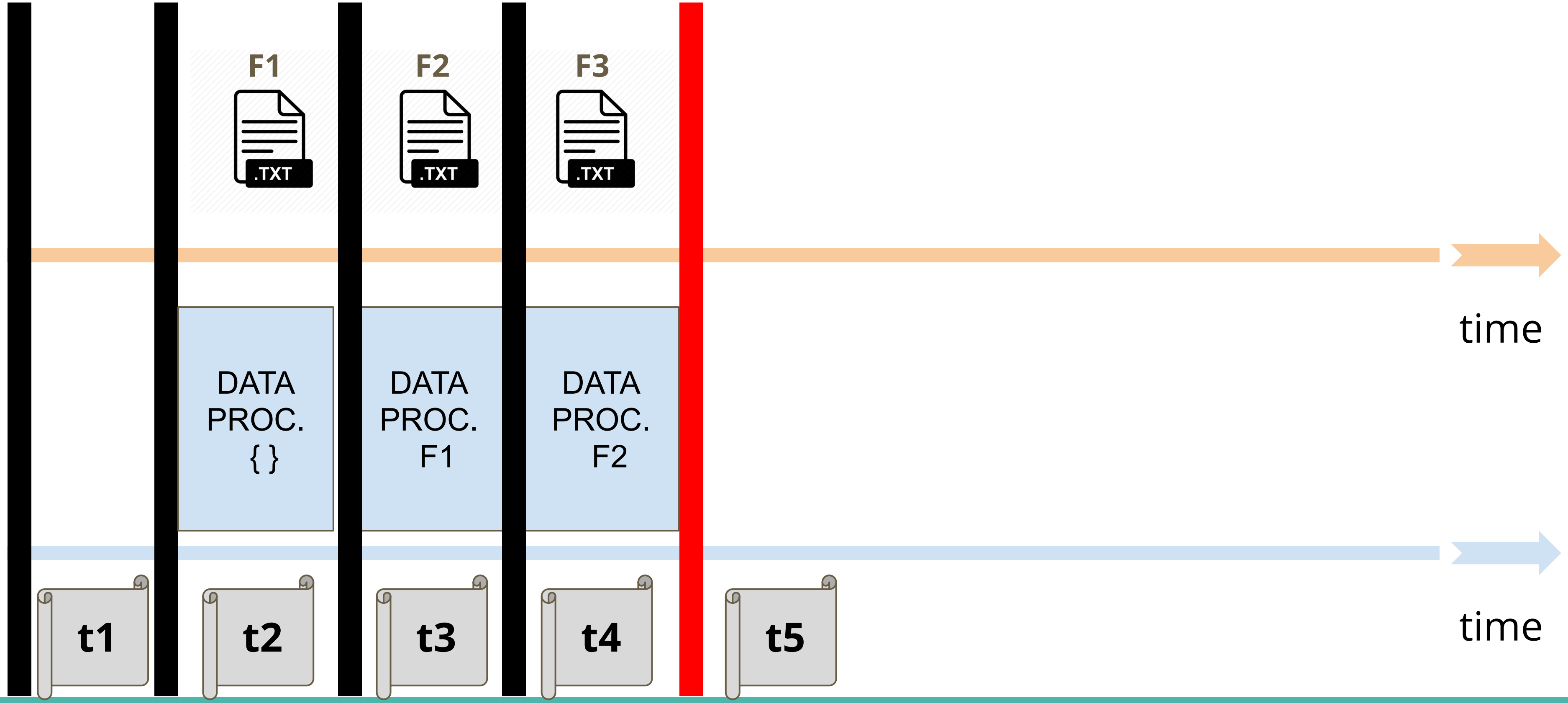


From RDDs to a DStream

Time Interval t_5

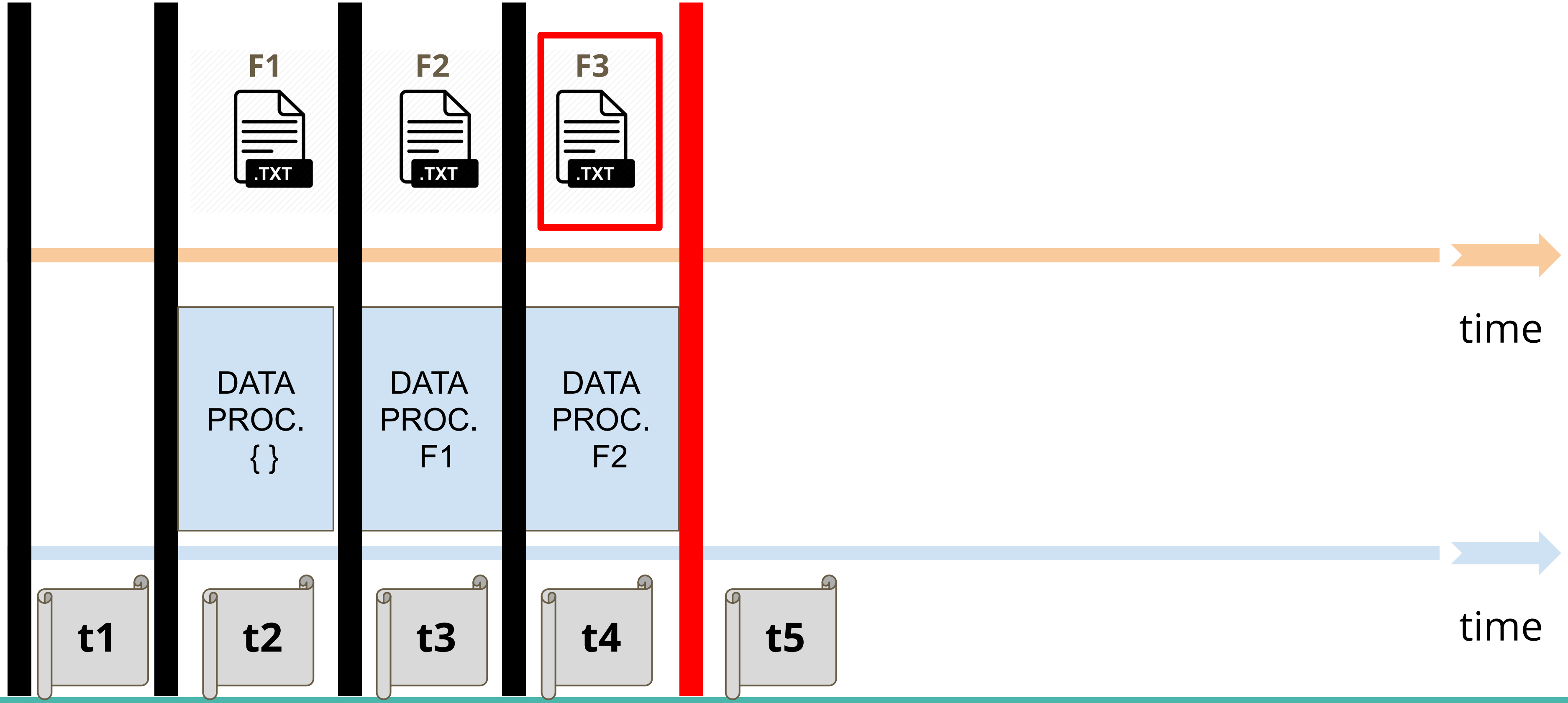
From RDDs to a DStream

Next time interval is checked by the Spark Streaming Context.



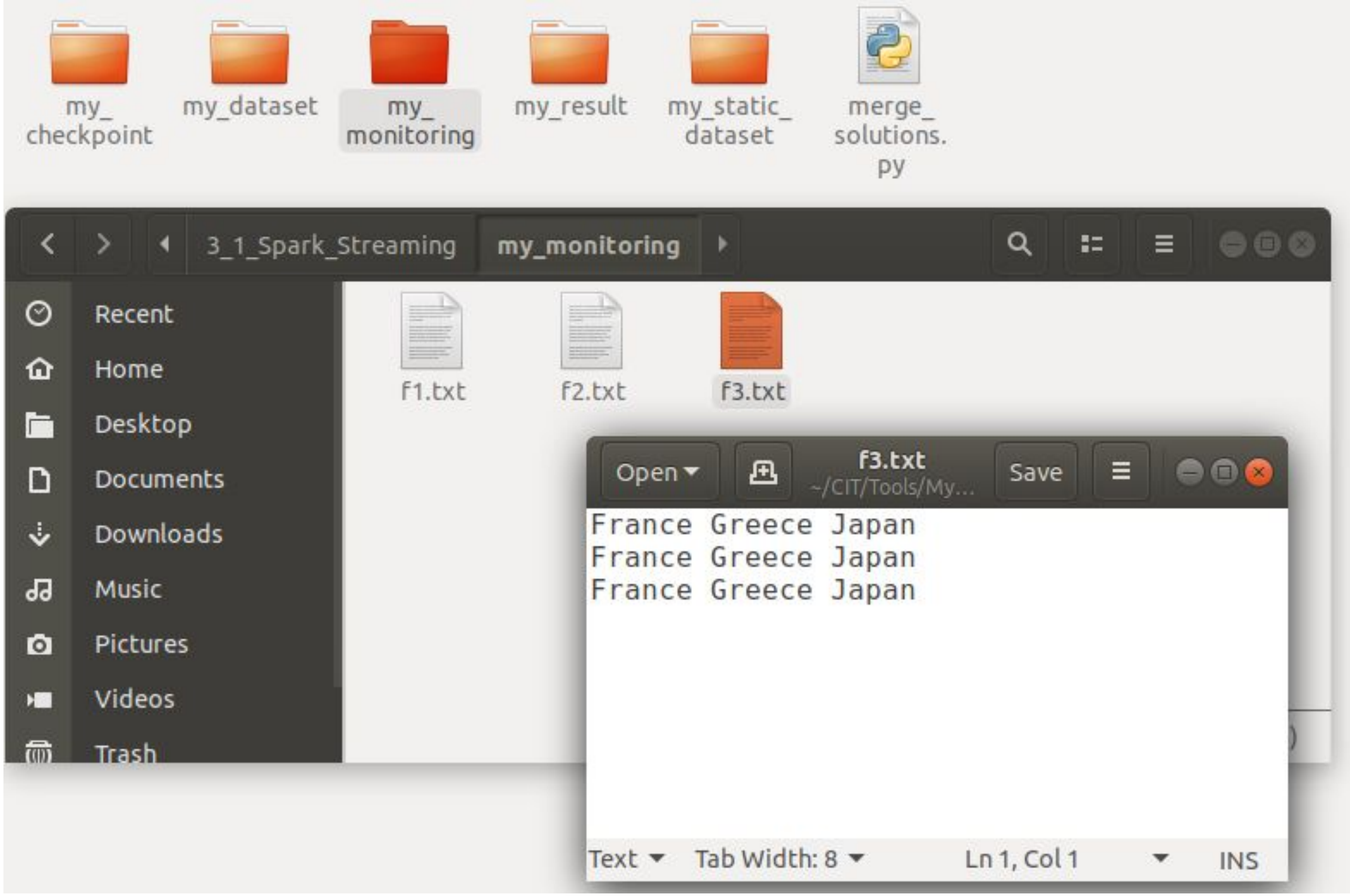
From RDDs to a DStream

It finds the new file **f3.txt** in **monitoring_dir**



From RDDs to a DStream

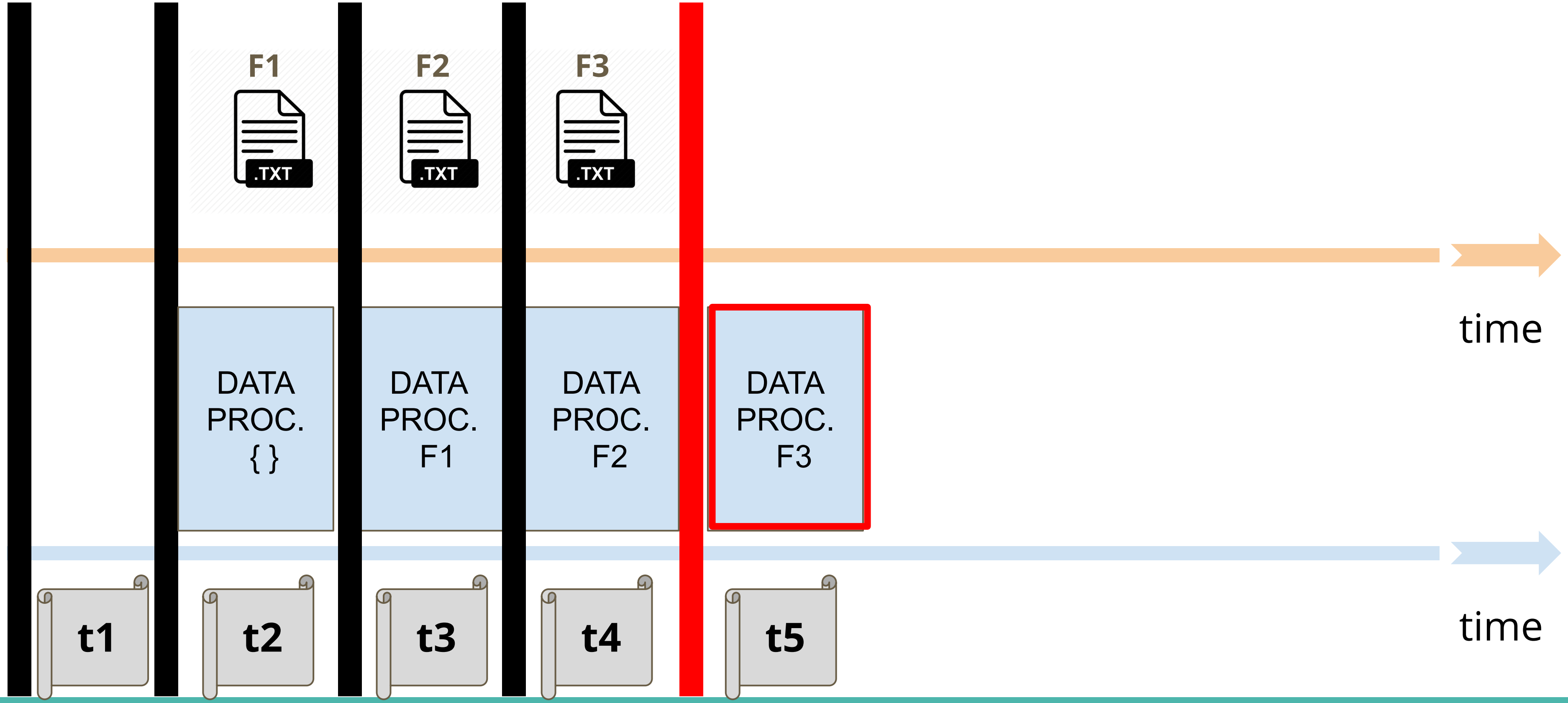
File **f3.txt** is detected as a new file in **monitoring_dir**



```
France Greece Japan
France Greece Japan
France Greece Japan
```

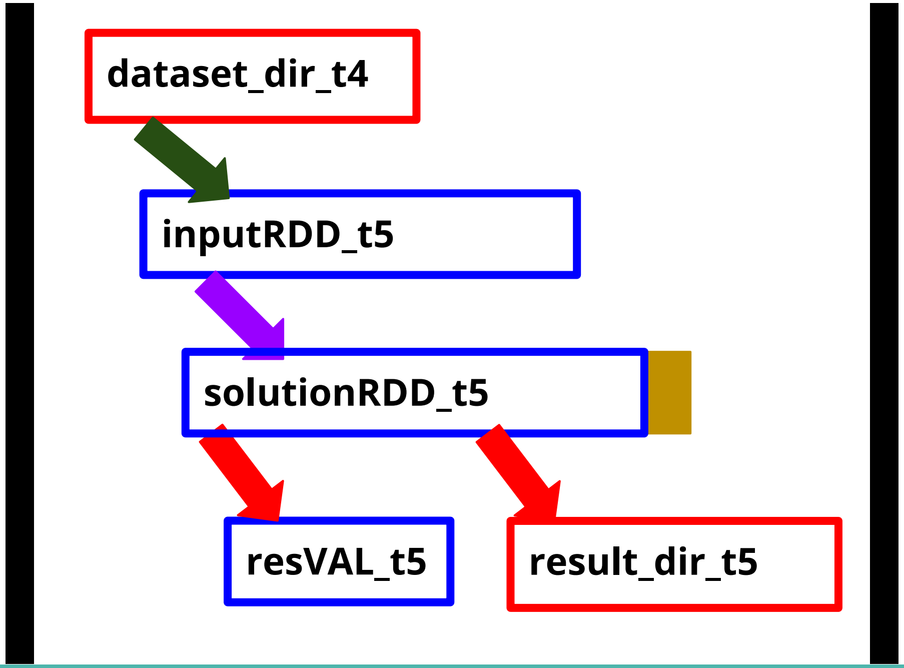
From RDDs to a DStream

It processes the file by reasoning with **RDD_t5** (wagon 5) of the **DStream** (train).



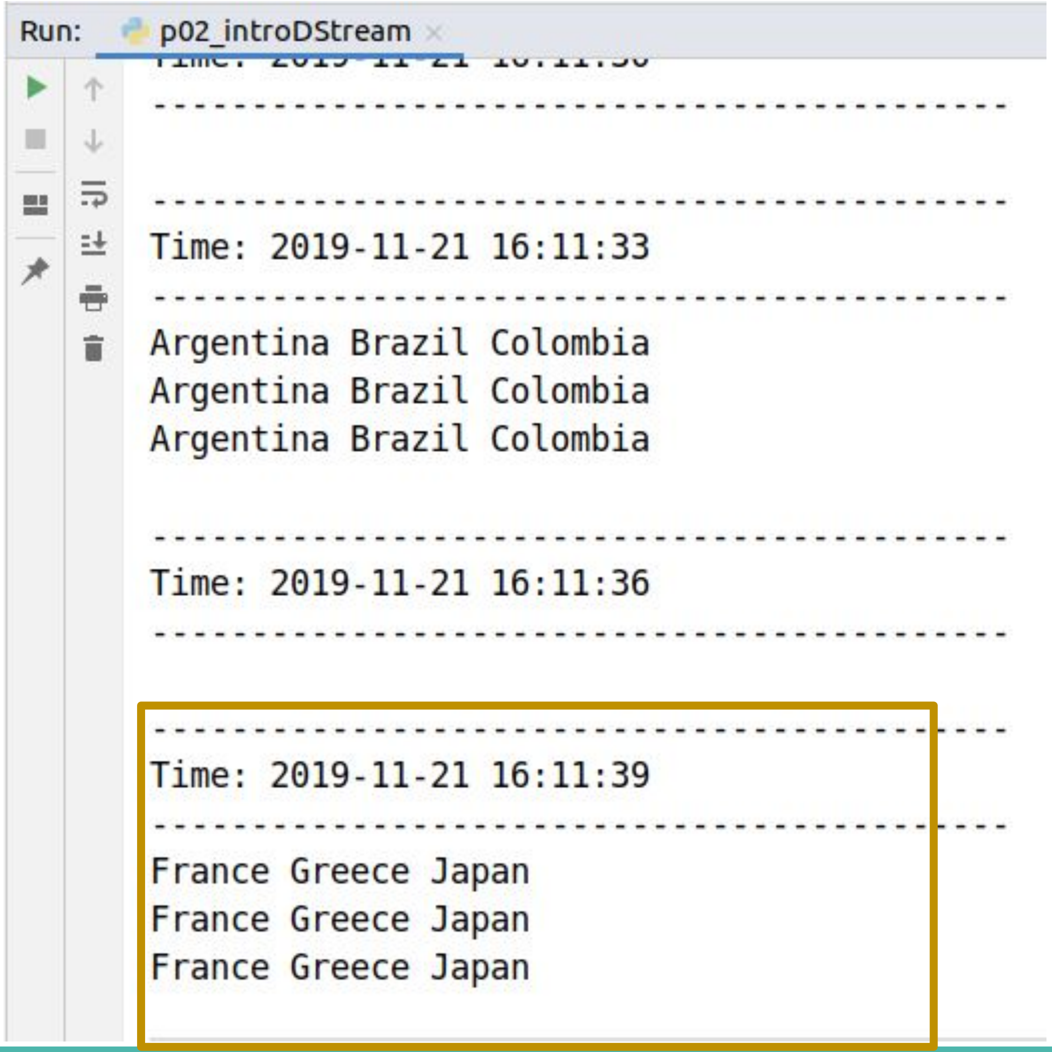
From RDDs to a DStream

It processes the file by reasoning with **RDD_t5** (wagon 5) of the **DStream** (train).



From RDDs to a DStream

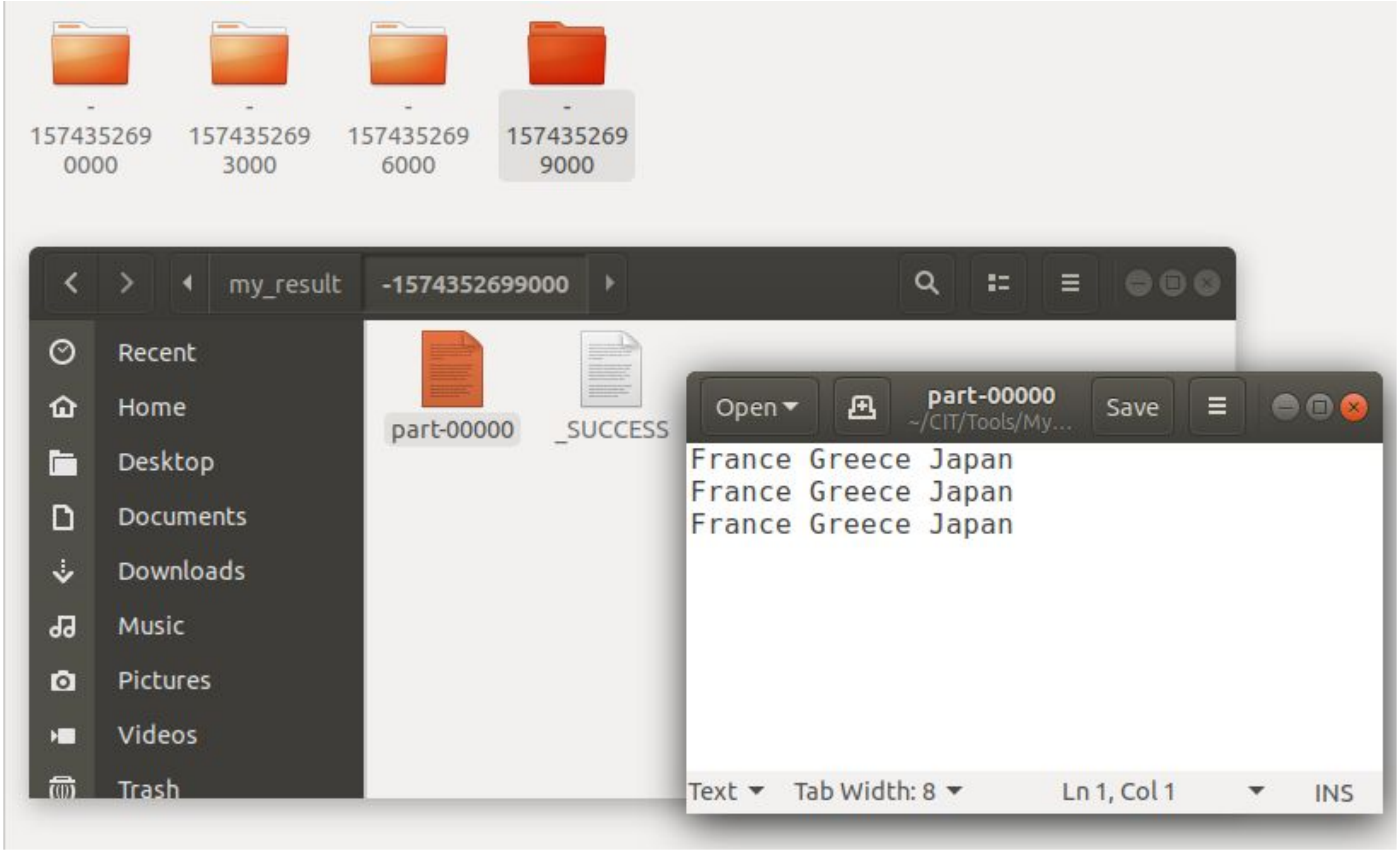
It **pprints** the results by the screen



```
-------------------------------------------


-------------------------------------------
Time: 2019-11-21 16:11:33
-------------------------------------------
Argentina Brazil Colombia
Argentina Brazil Colombia
Argentina Brazil Colombia


-------------------------------------------
Time: 2019-11-21 16:11:36
-------------------------------------------


-------------------------------------------
Time: 2019-11-21 16:11:39
-------------------------------------------
France Greece Japan
France Greece Japan
France Greece Japan
```

From RDDs to a DStream

It **saveAsTextFiles** by creating a new sub-folder in **result_dir**.



-1574352690000

-1574352693000

-1574352696000

-1574352699000

my_result -1574352699000

part-00000

_SUCCESS

part-00000 ~/CIT/Tools/My...

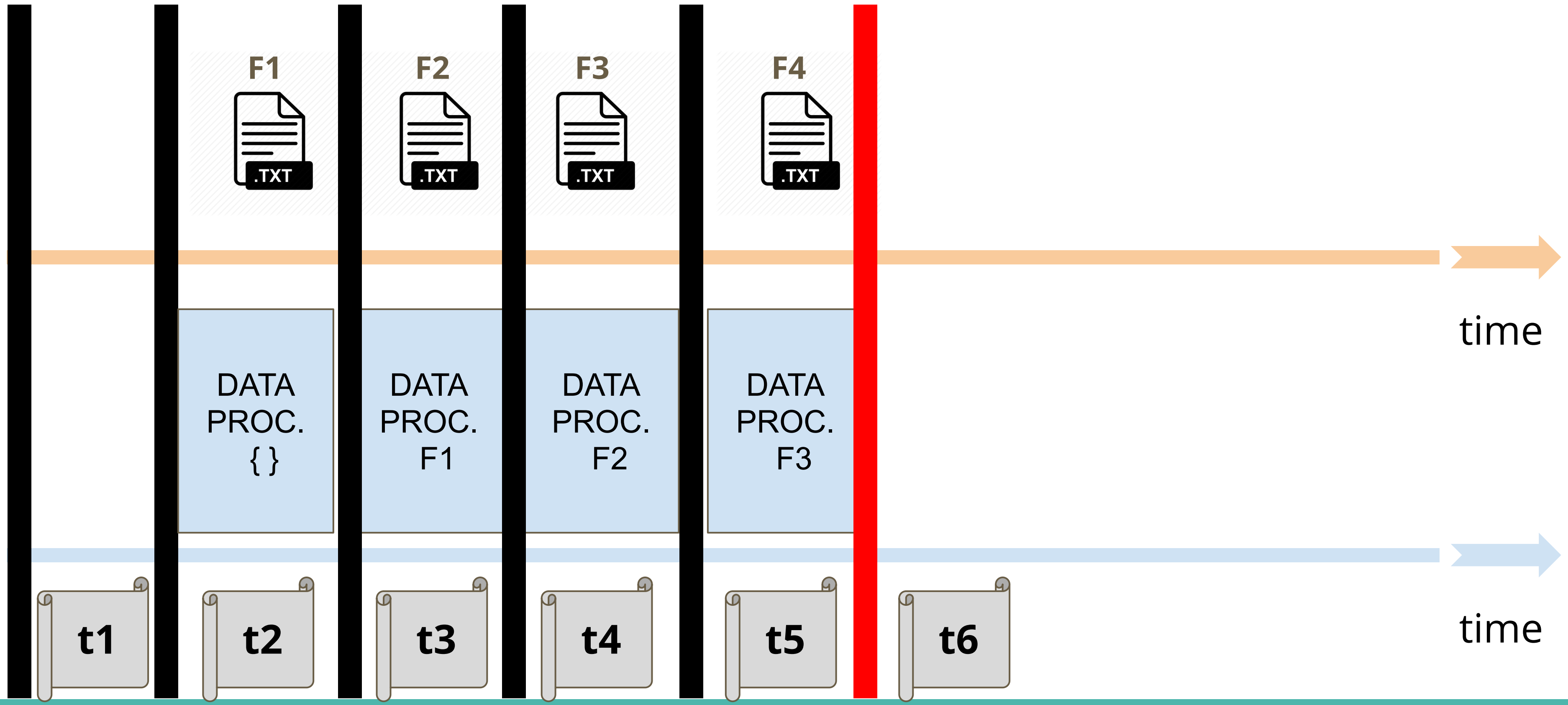
```
France Greece Japan
France Greece Japan
France Greece Japan
```

From RDDs to a DStream

Time Interval t_6

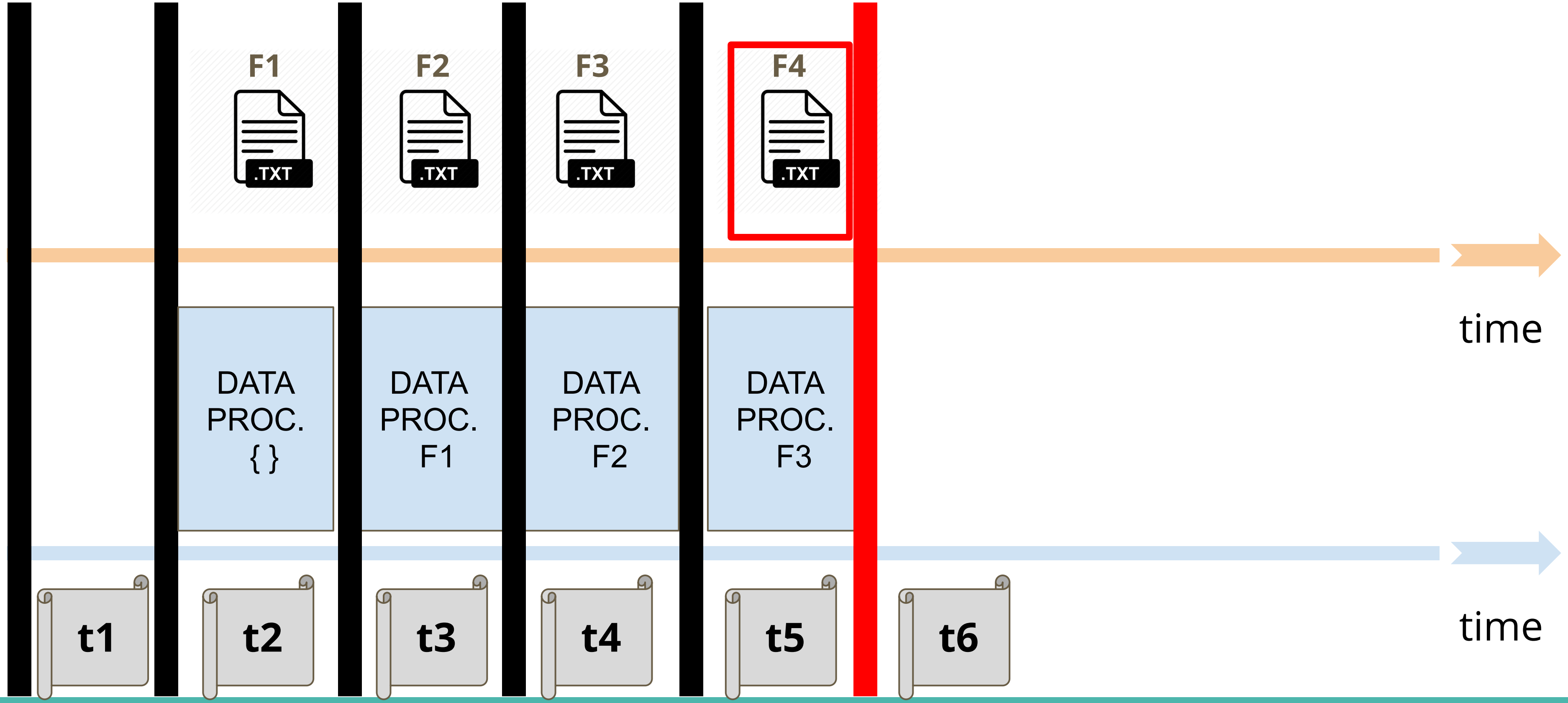
From RDDs to a DStream

Next time interval is checked by the Spark Streaming Context.



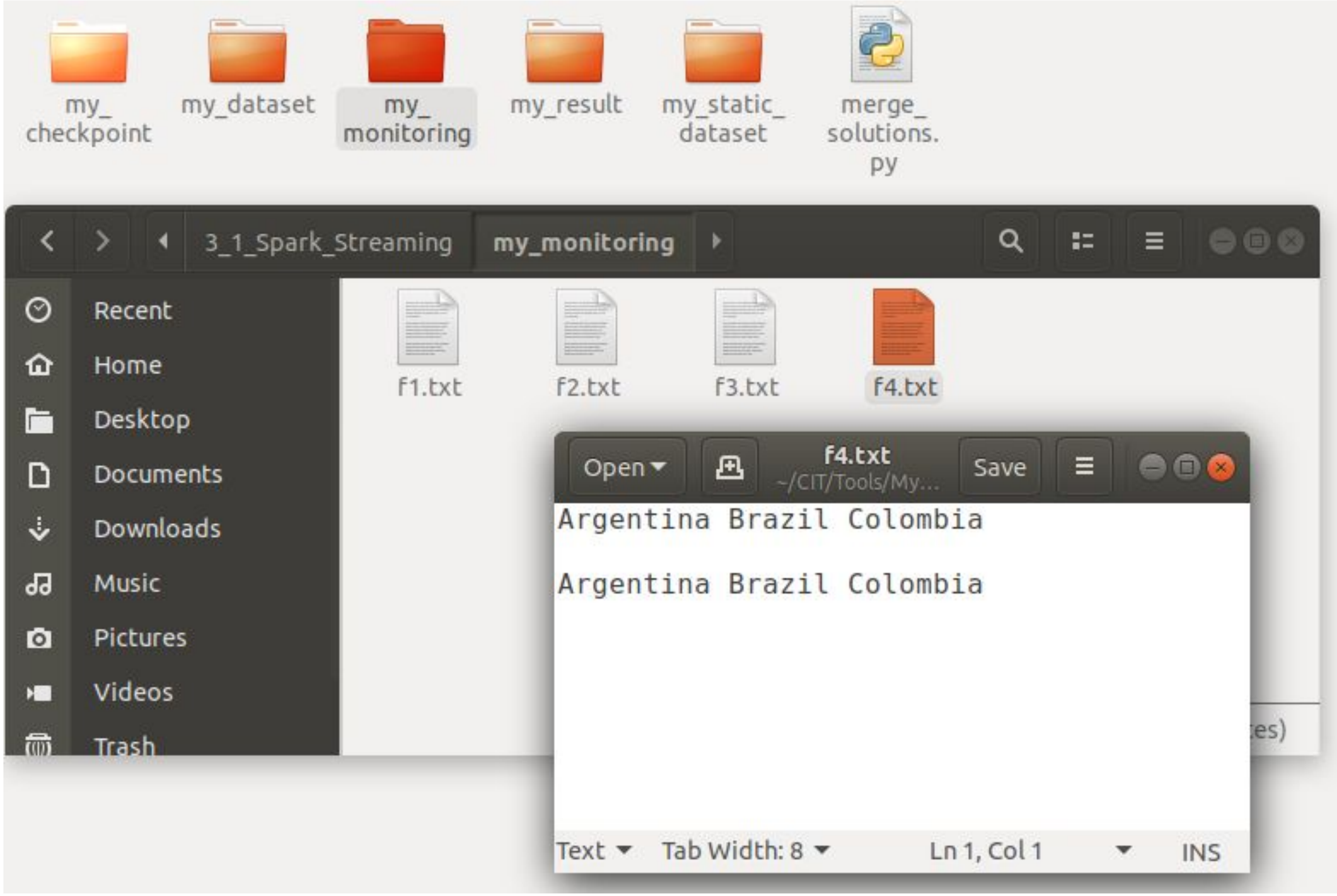
From RDDs to a DStream

It finds the new file **f4.txt** in **monitoring_dir**



From RDDs to a DStream

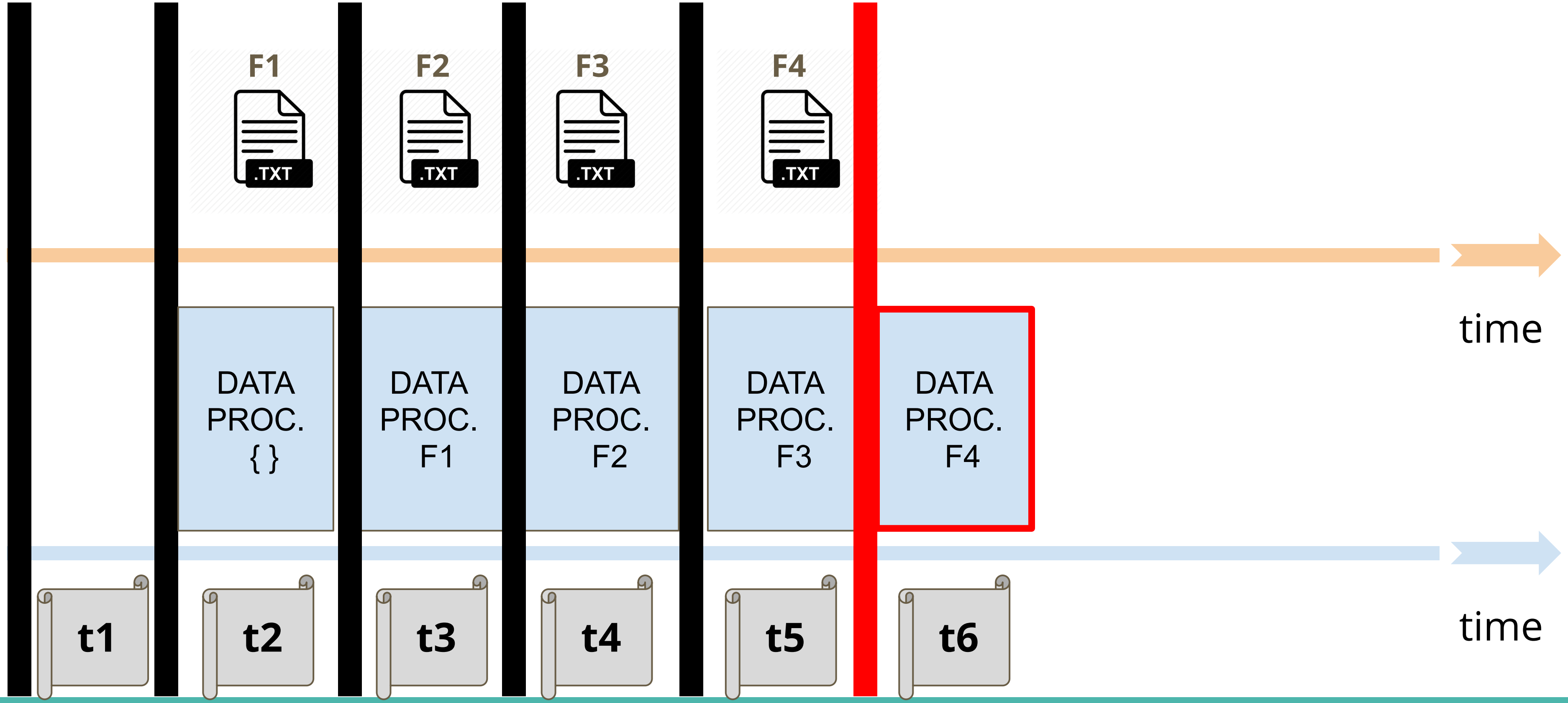
File **f4.txt** is detected as a new file in **monitoring_dir**



```
Argentina Brazil Colombia

Argentina Brazil Colombia
```

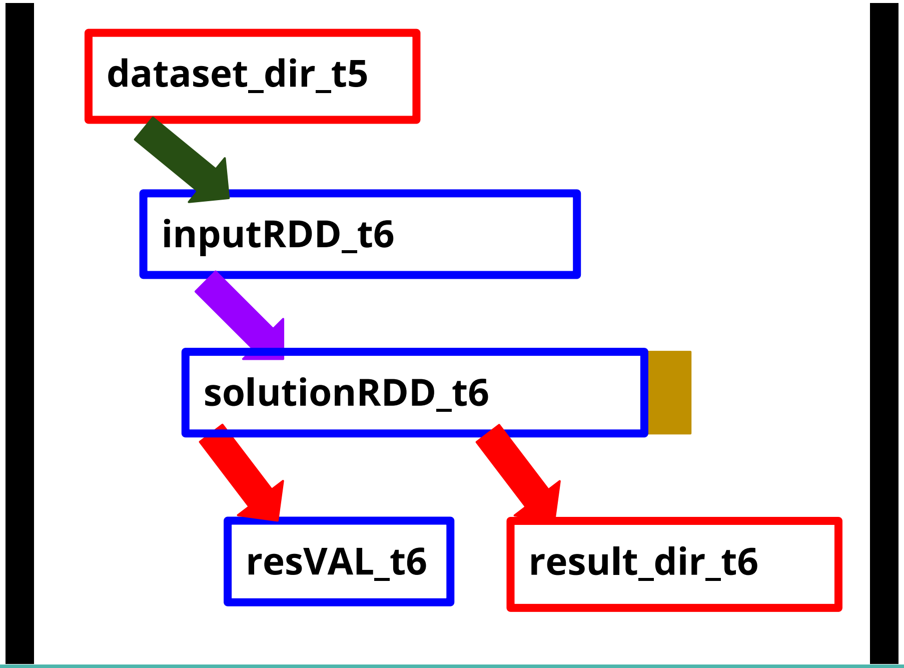
From RDDs to a DStream

It processes the file by reasoning with **RDD_t6** (wagon 6) of the **DStream** (train).



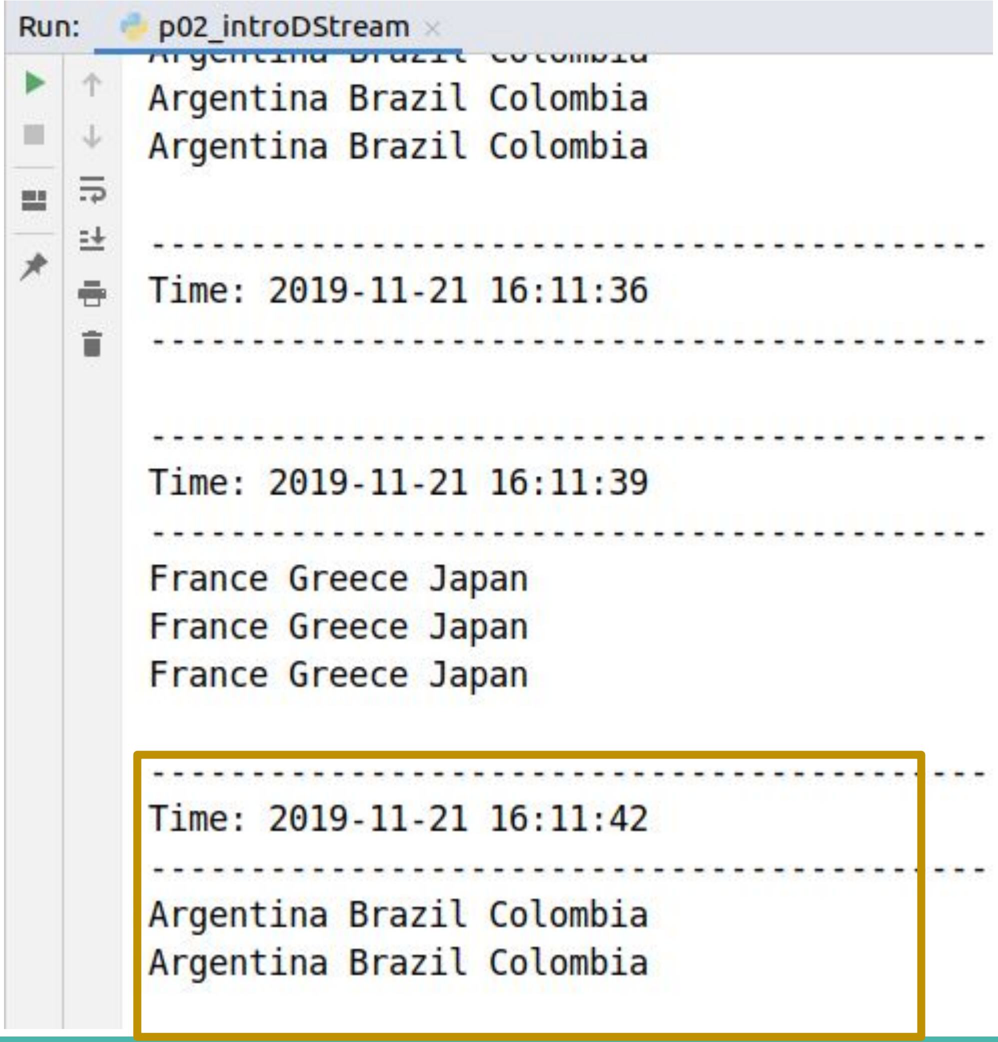
From RDDs to a DStream

It processes the file by reasoning with **RDD_t6** (wagon 6) of the **DStream** (train).



From RDDs to a DStream

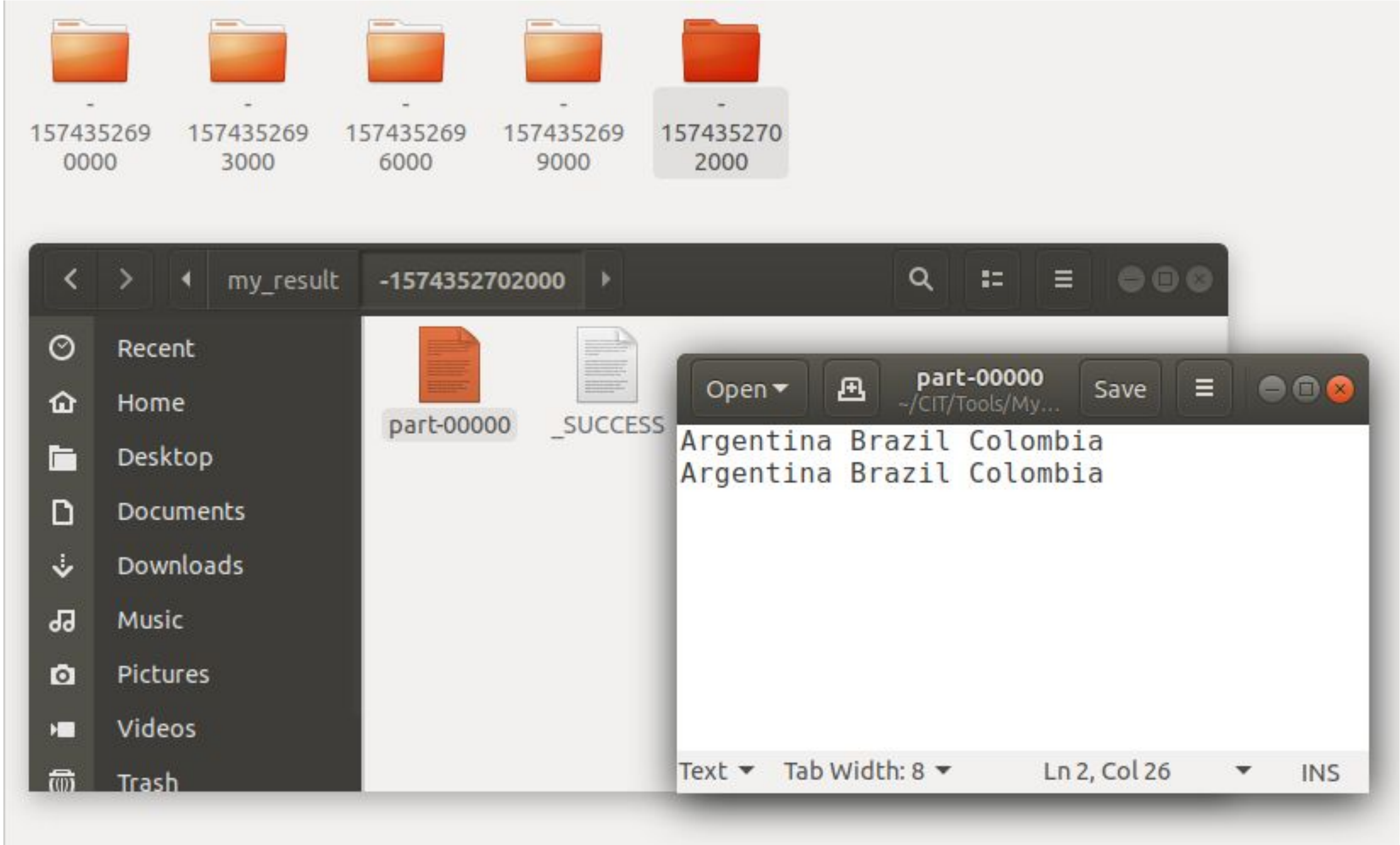
It **pprints** the results by the screen



```
Argentina Brazil Colombia
Argentina Brazil Colombia


-------------------------------------------
Time: 2019-11-21 16:11:36
-------------------------------------------


-------------------------------------------
Time: 2019-11-21 16:11:39
-------------------------------------------
France Greece Japan
France Greece Japan
France Greece Japan

-------------------------------------------
Time: 2019-11-21 16:11:42
-------------------------------------------
Argentina Brazil Colombia
Argentina Brazil Colombia
```

From RDDs to a DStream

It **saveAsTextFiles** by creating a new sub-folder in **result_dir**.



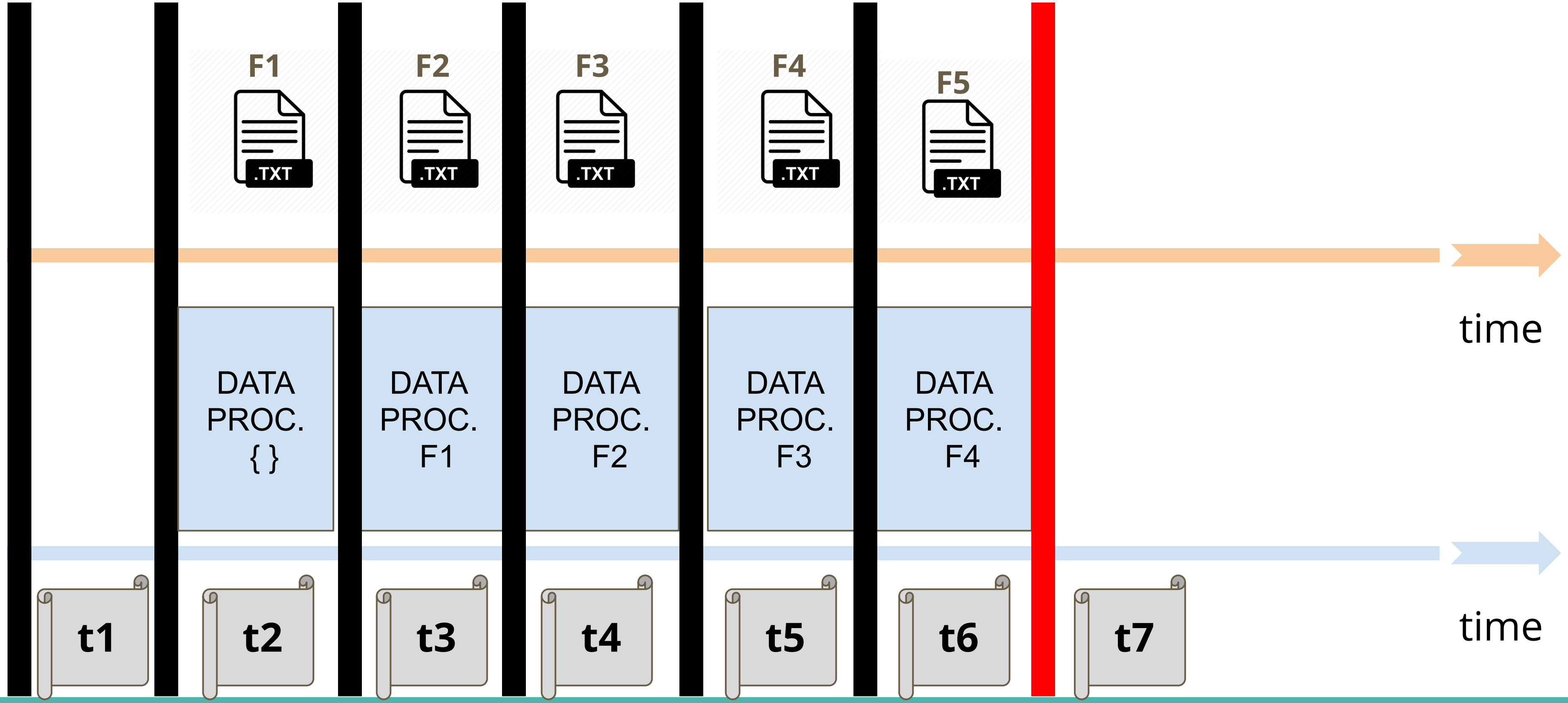
```
Argentina Brazil Colombia
Argentina Brazil Colombia
```

From RDDs to a DStream

Time Interval t_7

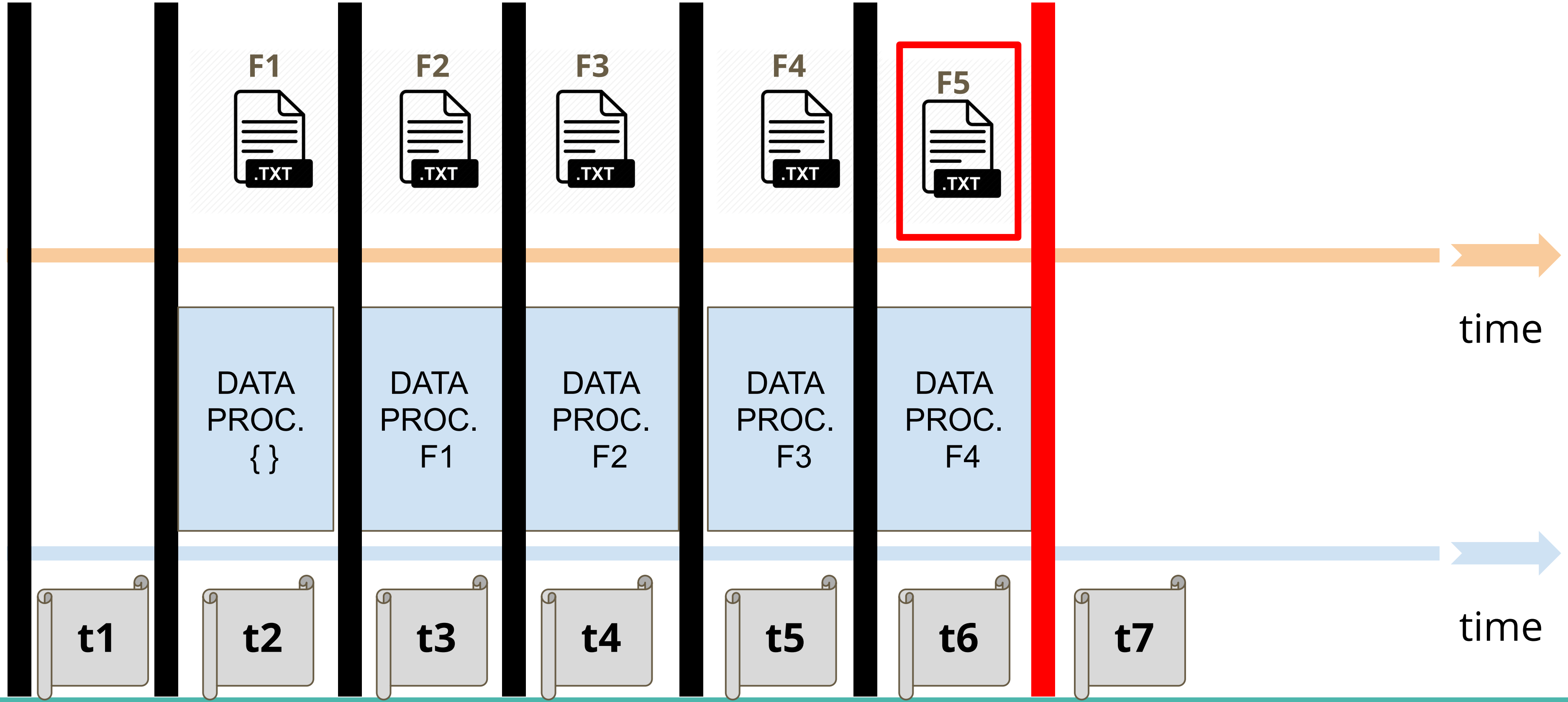
From RDDs to a DStream

Next time interval is checked by the Spark Streaming Context.



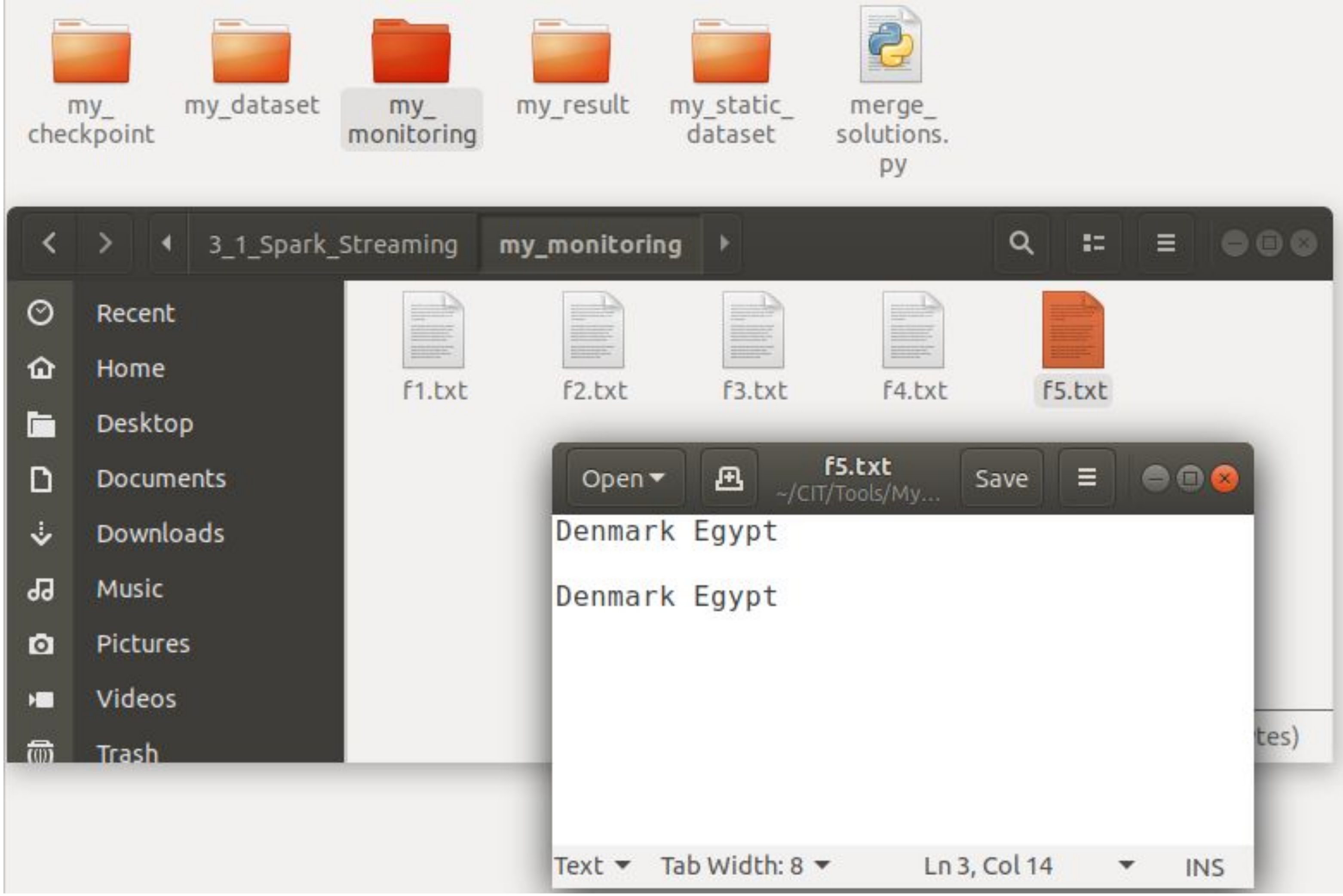
From RDDs to a DStream

It finds the new file **f5.txt** in **monitoring_dir**



From RDDs to a DStream

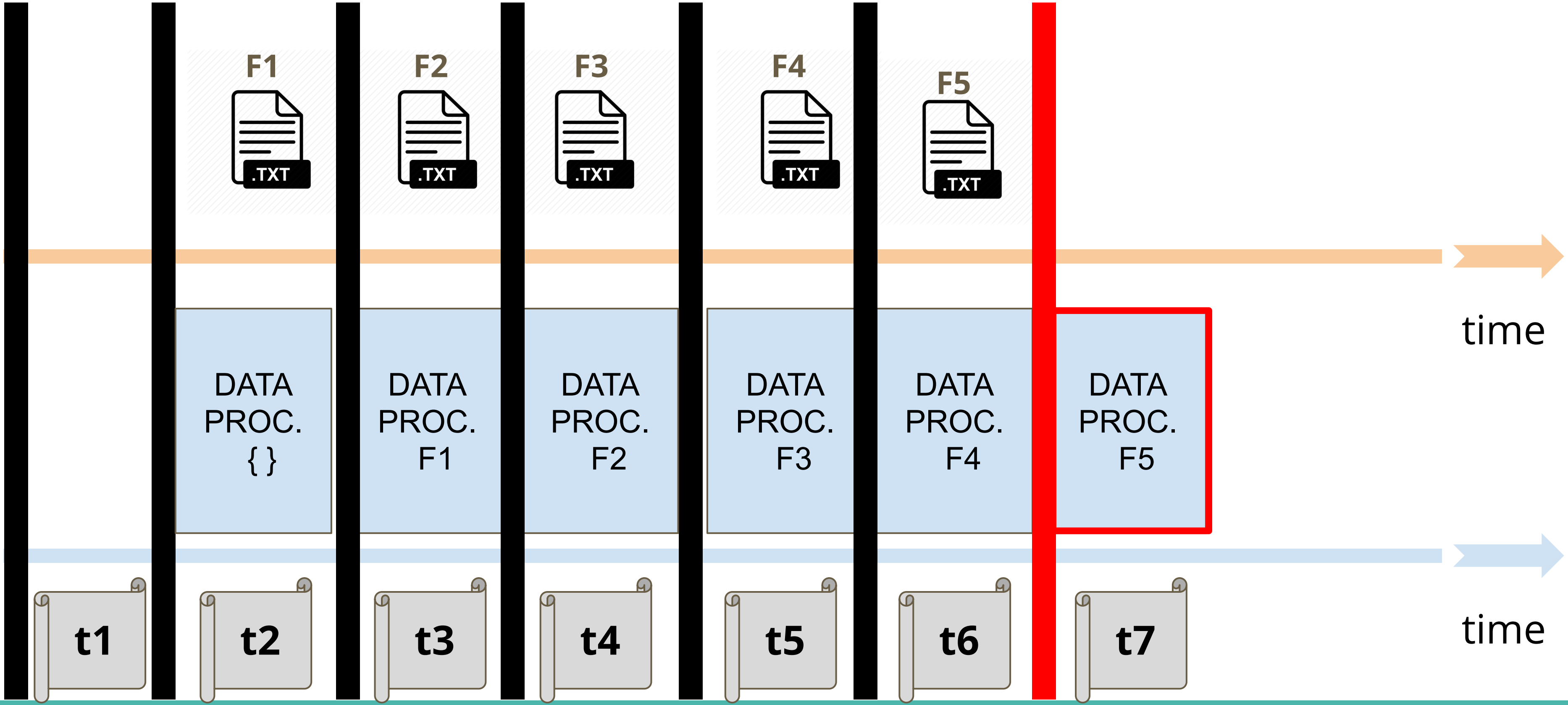
File **f5.txt** is detected as a new file in **monitoring_dir**



```
Denmark Egypt

Denmark Egypt
```

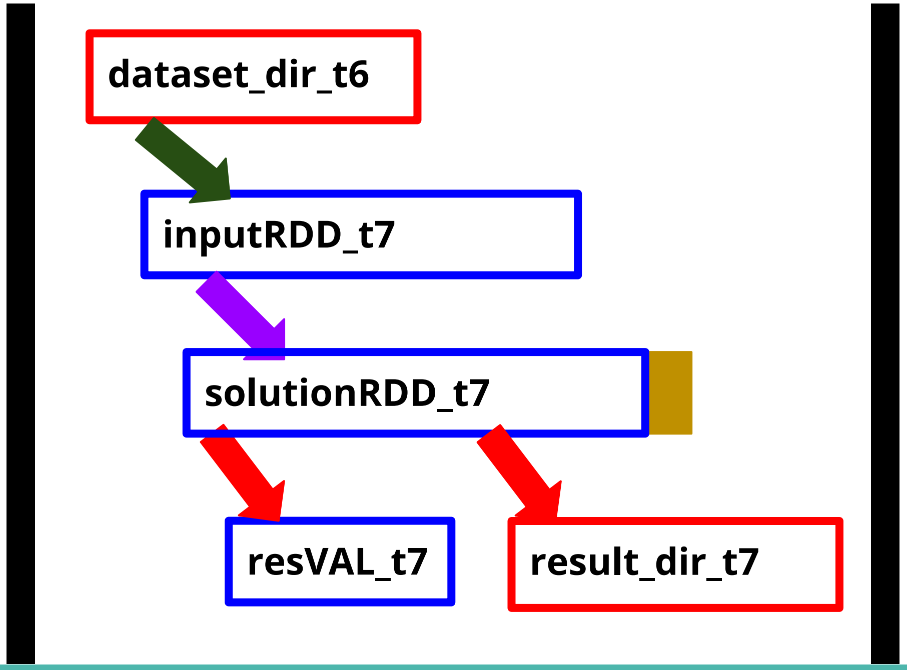
From RDDs to a DStream

It processes the file by reasoning with **RDD_t7** (wagon 7) of the **DStream** (train).



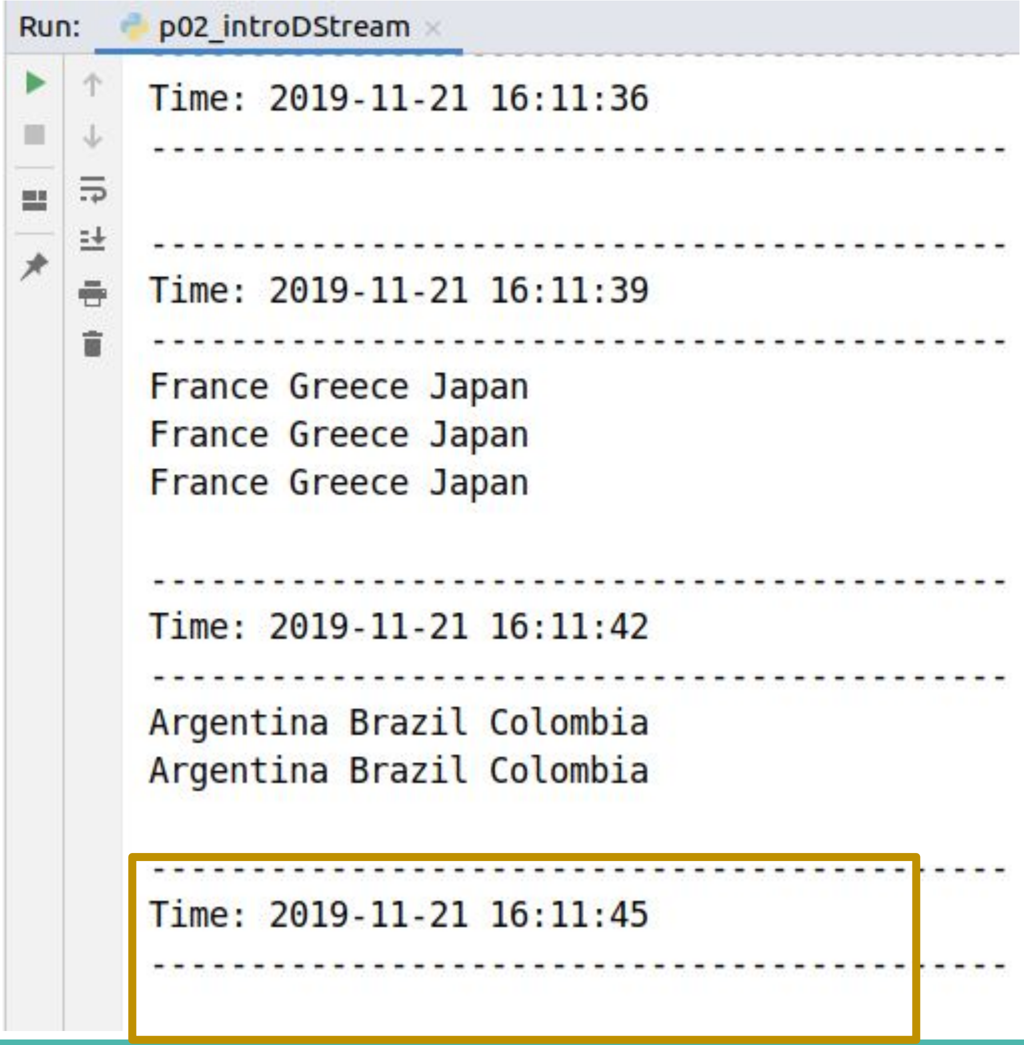
From RDDs to a DStream

It processes the file by reasoning with **RDD_t7** (wagon 7) of the **DStream** (train).



From RDDs to a DStream

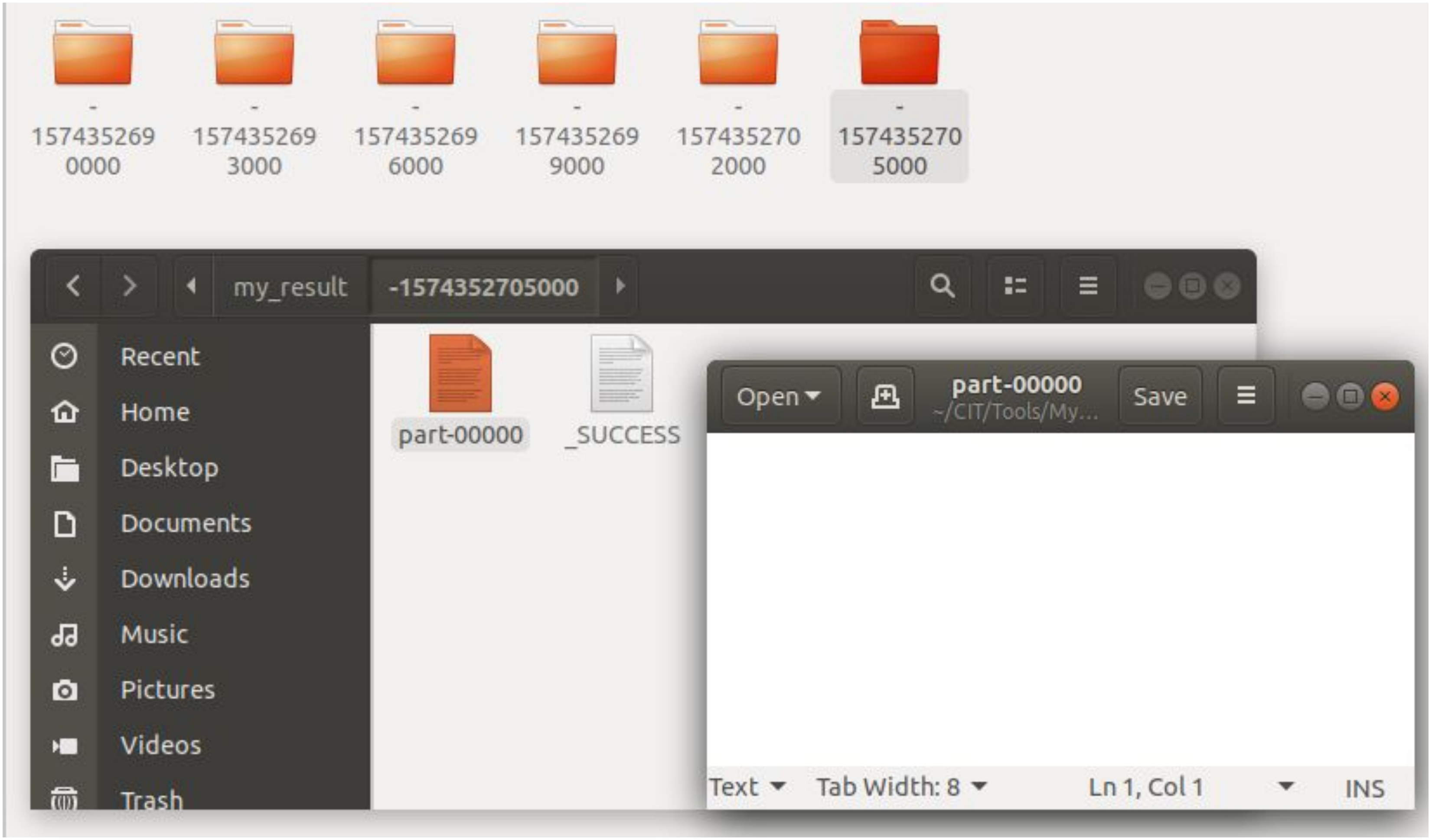
It **pprints** the results by the screen



```
-------------------------------------------
Time: 2019-11-21 16:11:36
-------------------------------------------


-------------------------------------------
Time: 2019-11-21 16:11:39
-------------------------------------------
France Greece Japan
France Greece Japan
France Greece Japan


-------------------------------------------
Time: 2019-11-21 16:11:42
-------------------------------------------
Argentina Brazil Colombia
Argentina Brazil Colombia


-------------------------------------------
Time: 2019-11-21 16:11:45
-------------------------------------------
```

From RDDs to a DStream

It **saveAsTextFiles** by creating a new sub-folder in **result_dir**.

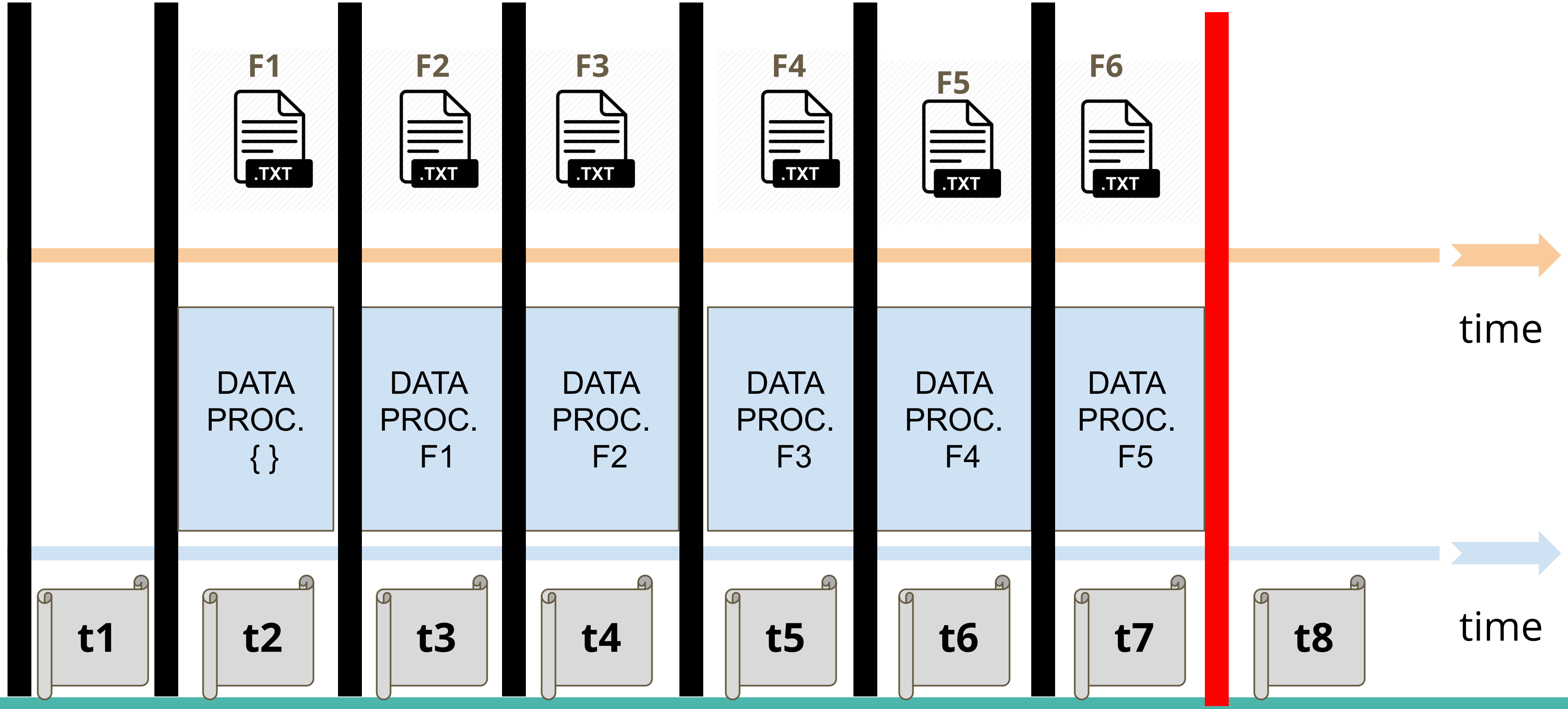


From RDDs to a DStream

Time Interval t_8

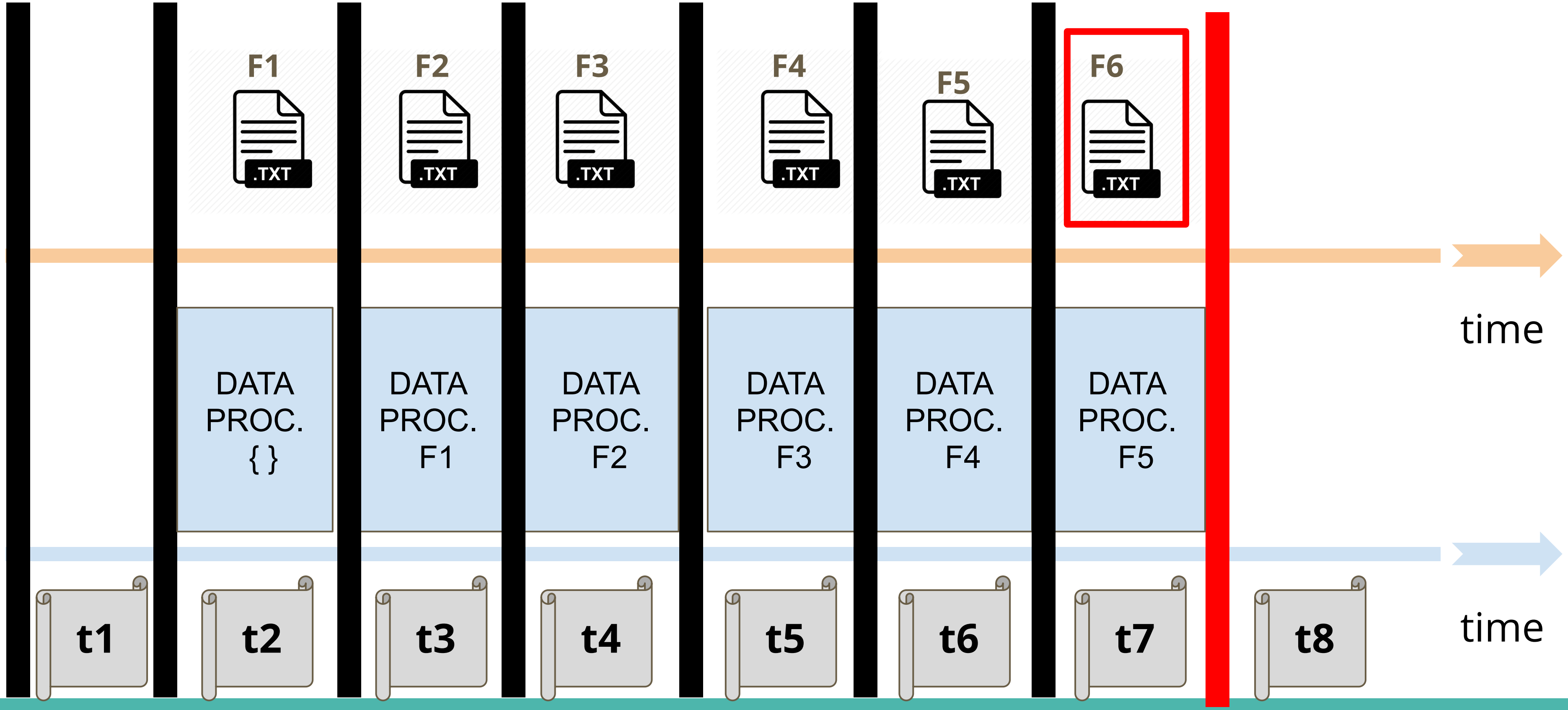
From RDDs to a DStream

Next time interval is checked by the Spark Streaming Context.



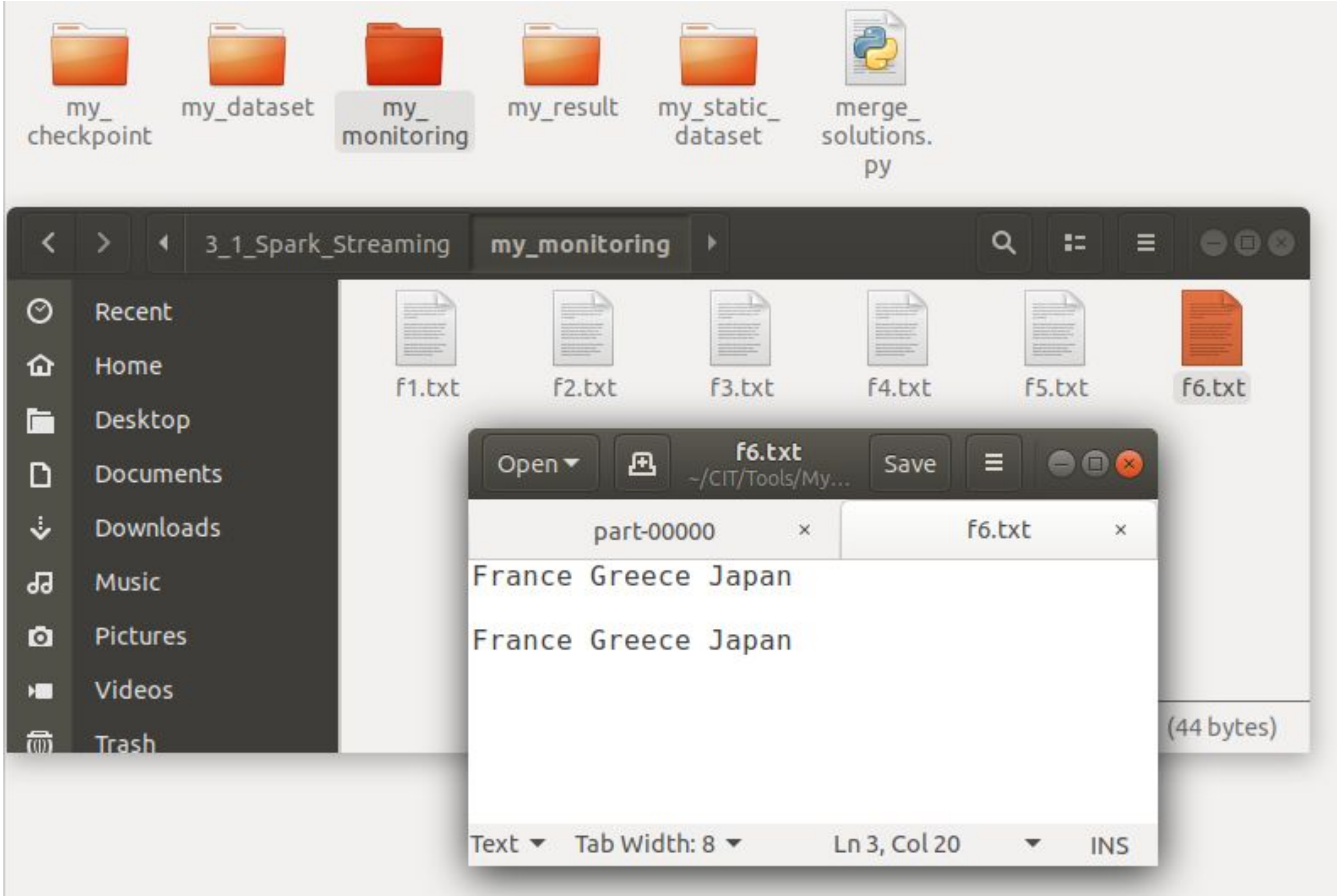
From RDDs to a DStream

It finds the new file **f6.txt** in **monitoring_dir**



From RDDs to a DStream

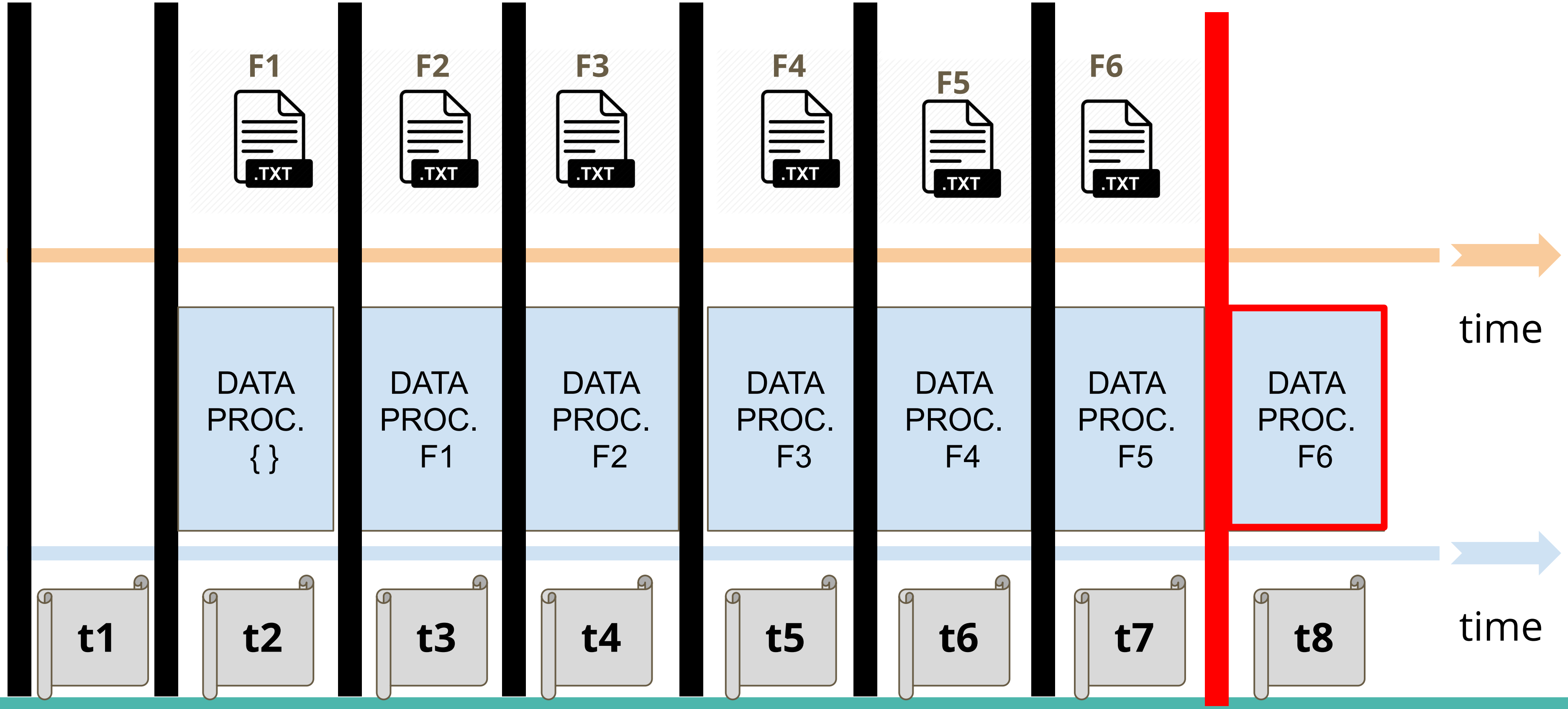
File **f6.txt** is detected as a new file in **monitoring_dir**



```
France Greece Japan

France Greece Japan
```

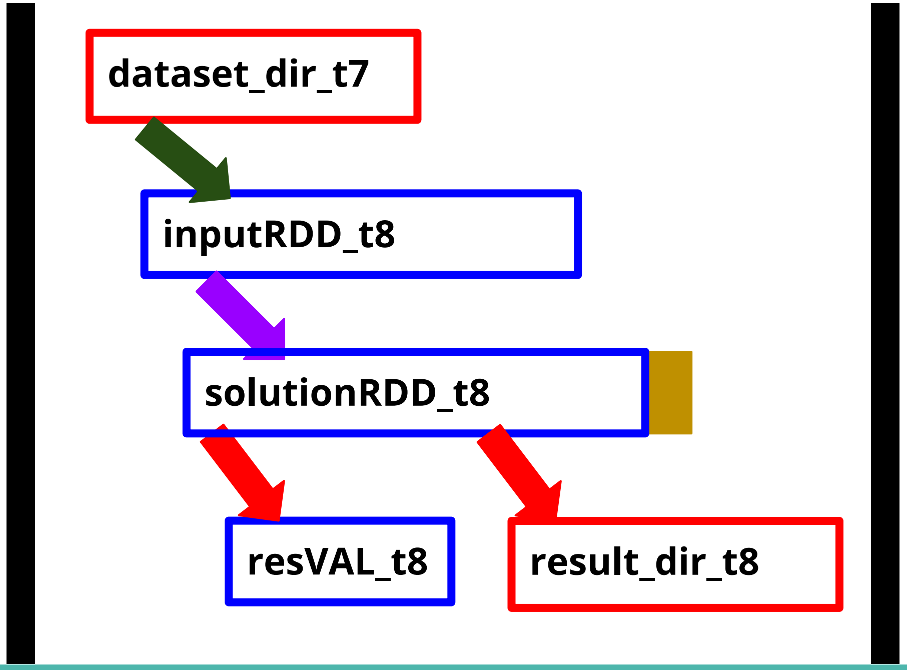
From RDDs to a DStream

It processes the file by reasoning with **RDD_t8** (wagon 8) of the **DStream** (train).



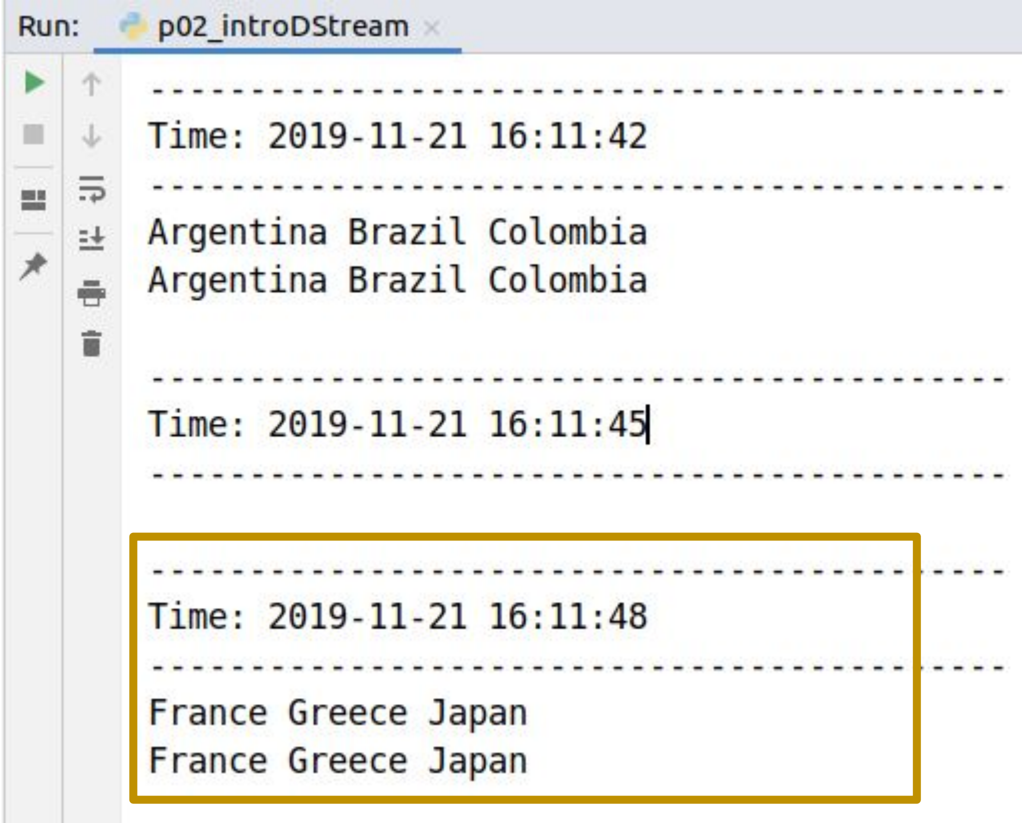
From RDDs to a DStream

It processes the file by reasoning with **RDD_t8** (wagon 8) of the **DStream** (train).



From RDDs to a DStream

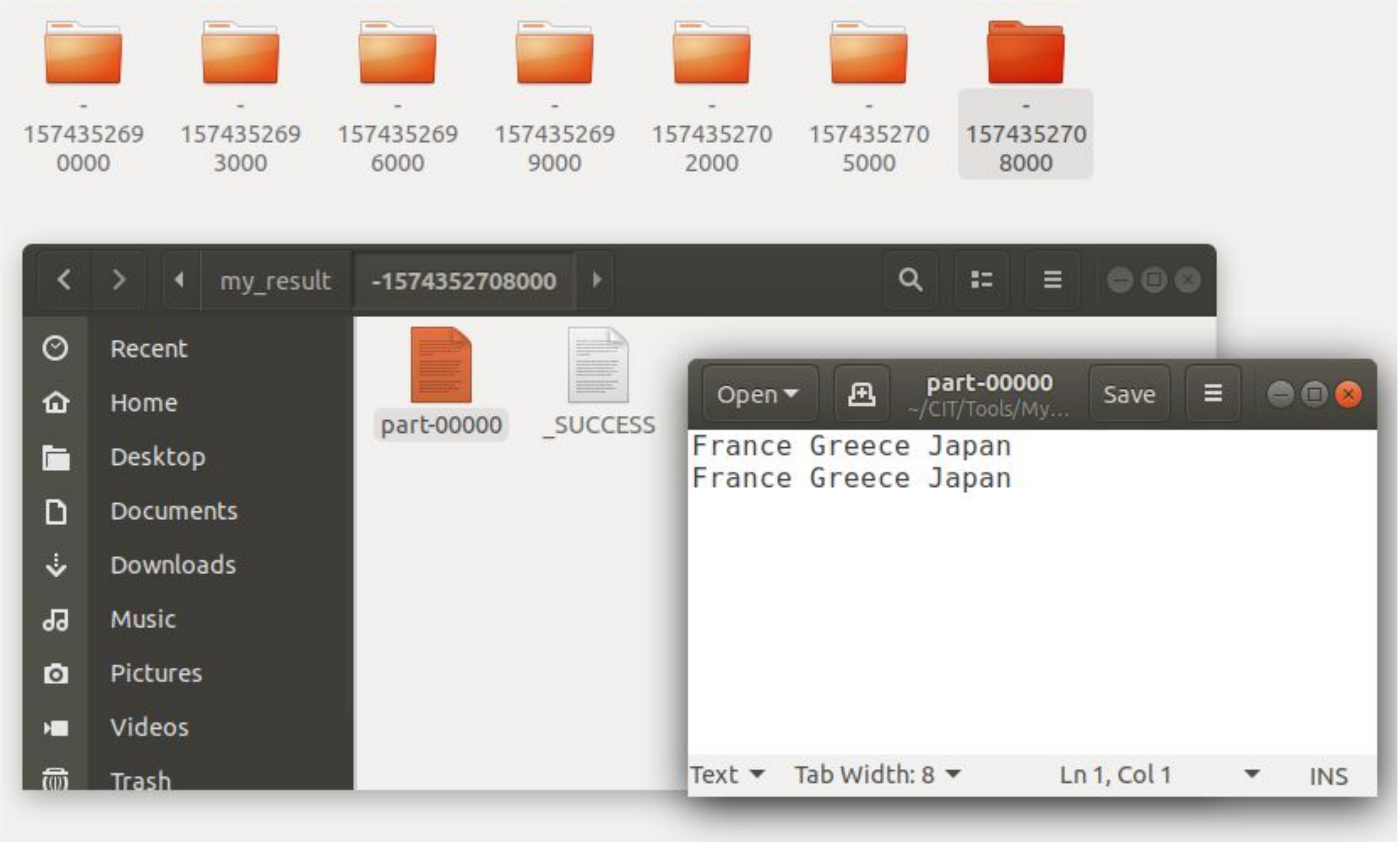
It **pprints** the results by the screen



```
-------------------------------------------
Time: 2019-11-21 16:11:42
-------------------------------------------
Argentina Brazil Colombia
Argentina Brazil Colombia

-------------------------------------------
Time: 2019-11-21 16:11:45
-------------------------------------------

-------------------------------------------
Time: 2019-11-21 16:11:48
-------------------------------------------
France Greece Japan
France Greece Japan
```

From RDDs to a DStream

It **saveAsTextFiles** by creating a new sub-folder in **result_dir**.



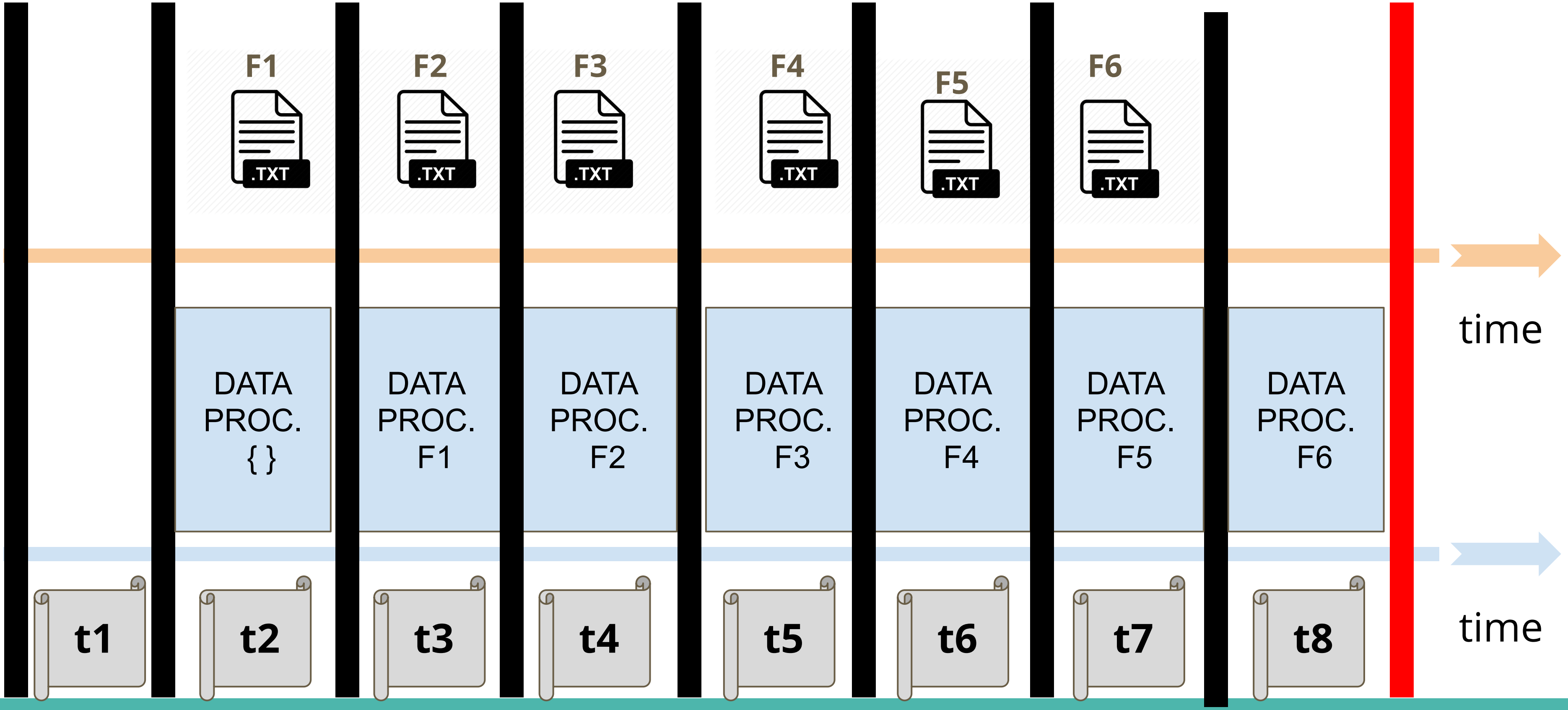
```
France Greece Japan
France Greece Japan
```

From RDDs to a DStream

There is no next
Time Interval t_9

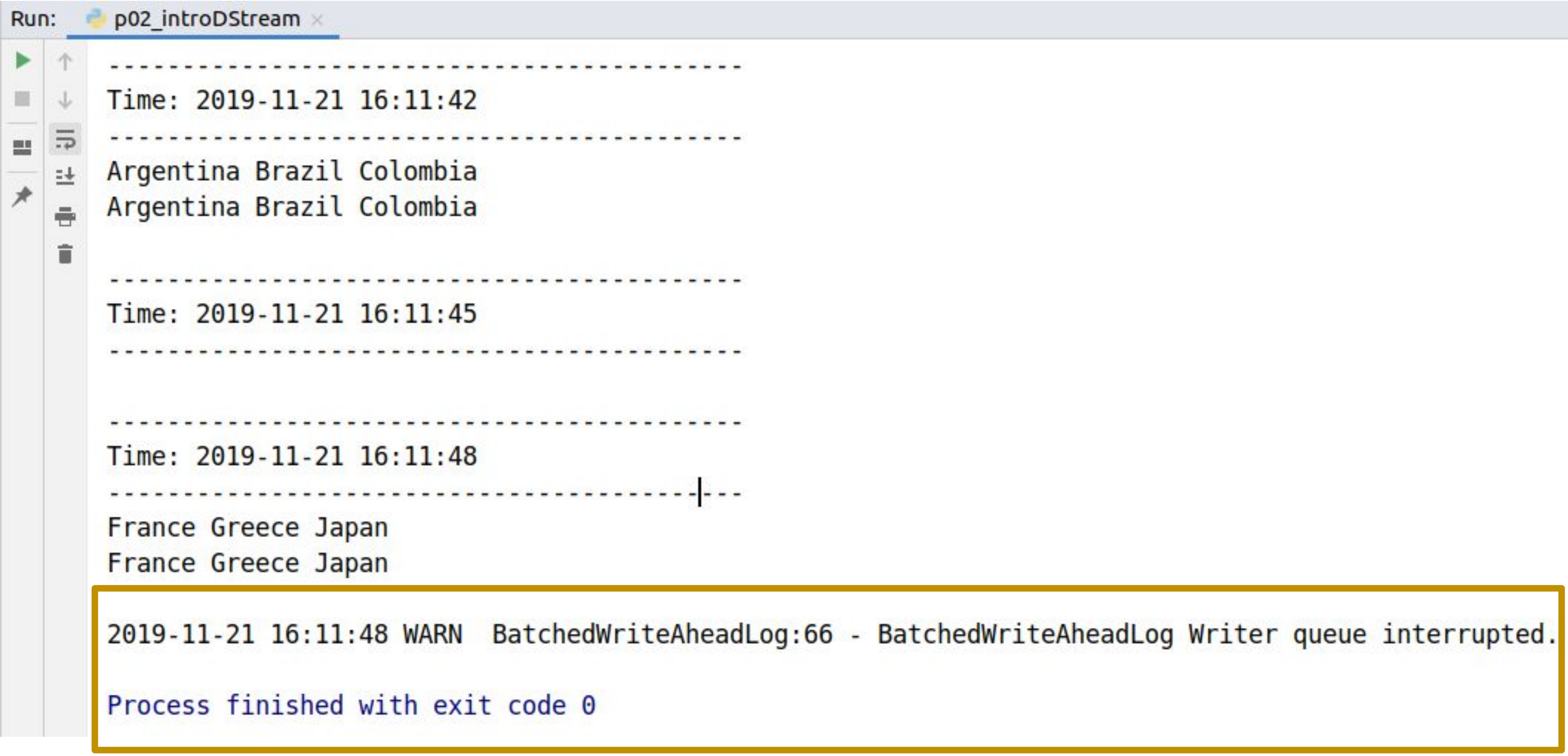
From RDDs to a DStream

After processing this last file the Spark Streaming Context is asked to stop. The entire application finishes.



From RDDs to a DStream

After processing this last file the Spark Streaming Context is asked to stop. The entire application finishes.



```
-------------------------------------------
Time: 2019-11-21 16:11:42
-------------------------------------------
Argentina Brazil Colombia
Argentina Brazil Colombia

-------------------------------------------
Time: 2019-11-21 16:11:45
-------------------------------------------


-------------------------------------------
Time: 2019-11-21 16:11:48
-------------------------------------------
France Greece Japan
France Greece Japan

2019-11-21 16:11:48 WARN  BatchedWriteAheadLog:66 - BatchedWriteAheadLog Writer queue interrupted.

Process finished with exit code 0
```

Outline

1. Setting Up the Context.
2. Measurement Unit: Time Interval & Data Batch.
3. From RDDs to a DStream.
4. File Transfer Process.
5. Spark Streaming Context Process.
6. Stateless and Stateful Operations.

File Transfer Process

We have seen our first
Spark Streaming Application,
p02_introDStream.py

File Transfer Process

We have seen our first
Spark Streaming Application,
p02_introDStream.py

However,
two questions remain still unclear...

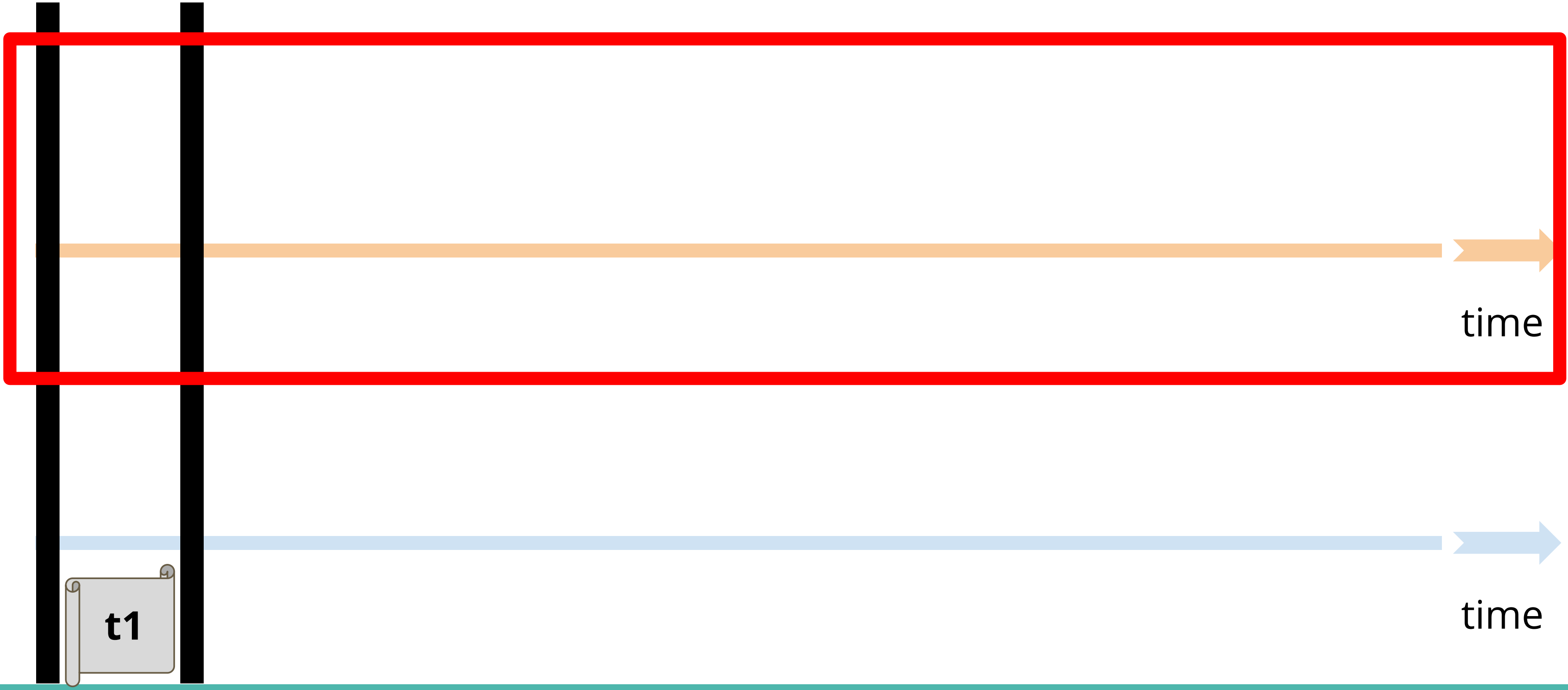
File Transfer Process

Question 1

How are the files [F1.txt, …, F6.txt] transferred from **dataset_dir** to **monitoring_dir** at a file per time interval pace?

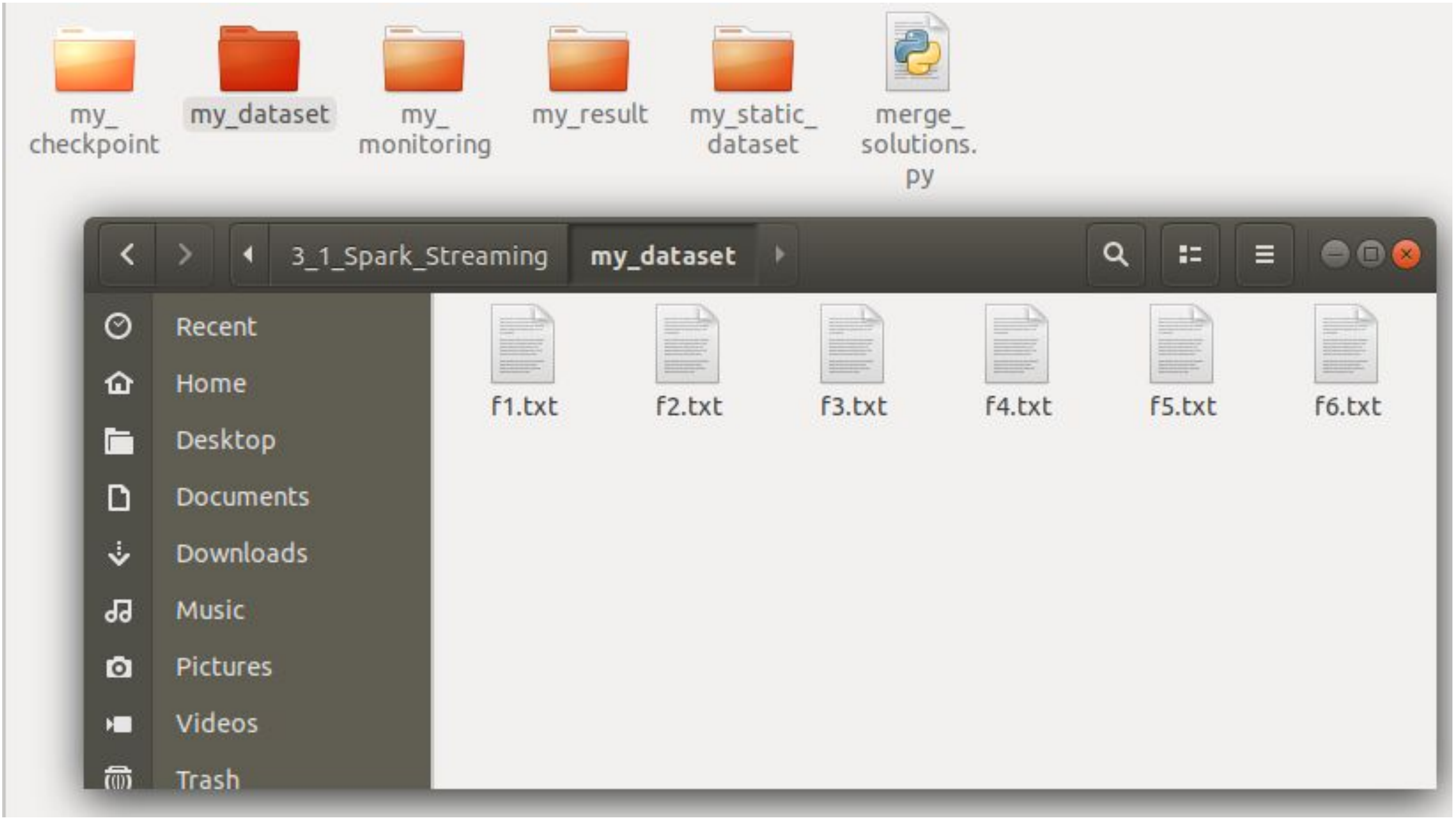
File Transfer Process

<u>Note:</u>
In our diagram, the File transfer process is the one on top of the picture.



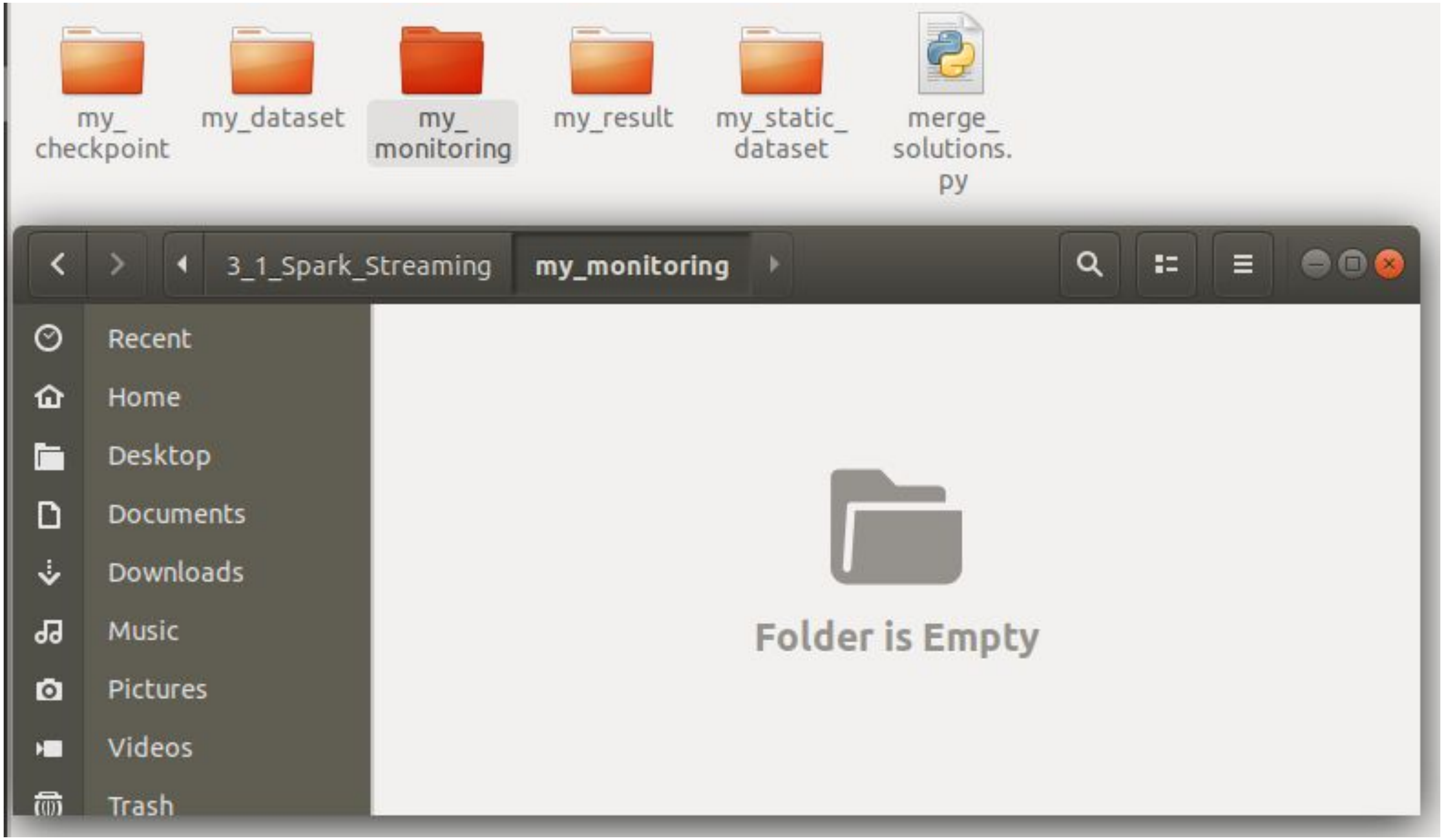
File Transfer Process

As we can see, dataset_dir contains the set of files to be transferred.

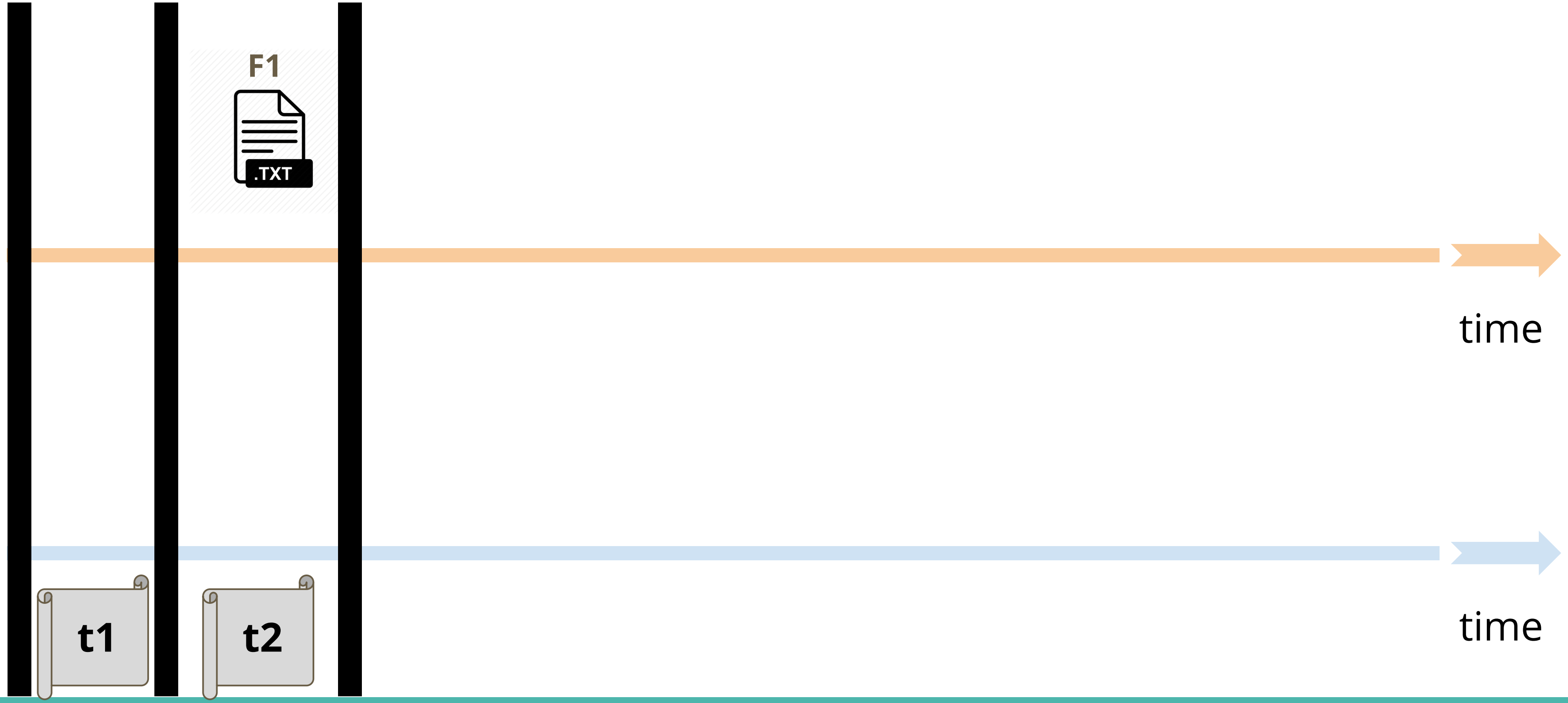


File Transfer Process

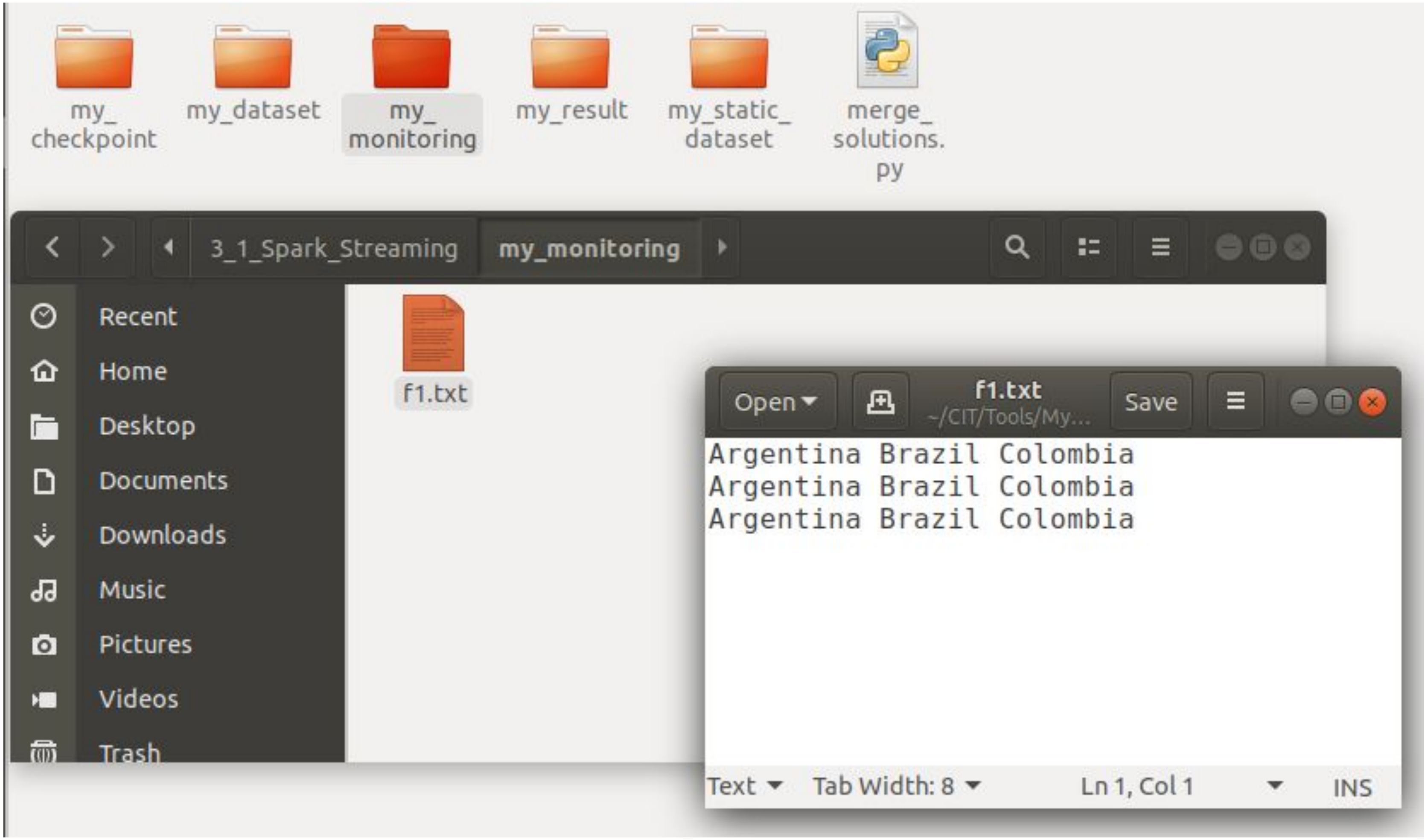
As we can see, monitoring_dir is empty.



File Transfer Process

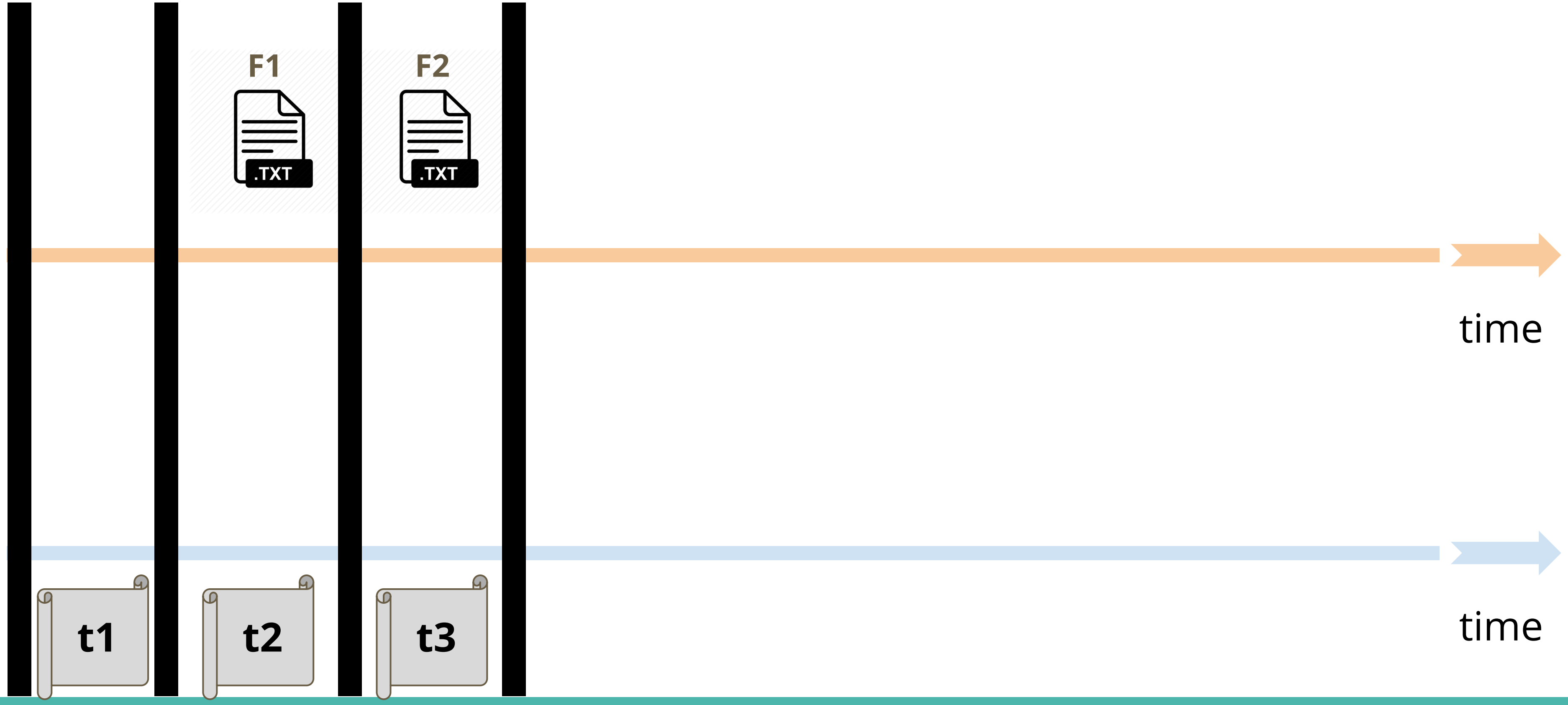


File Transfer Process

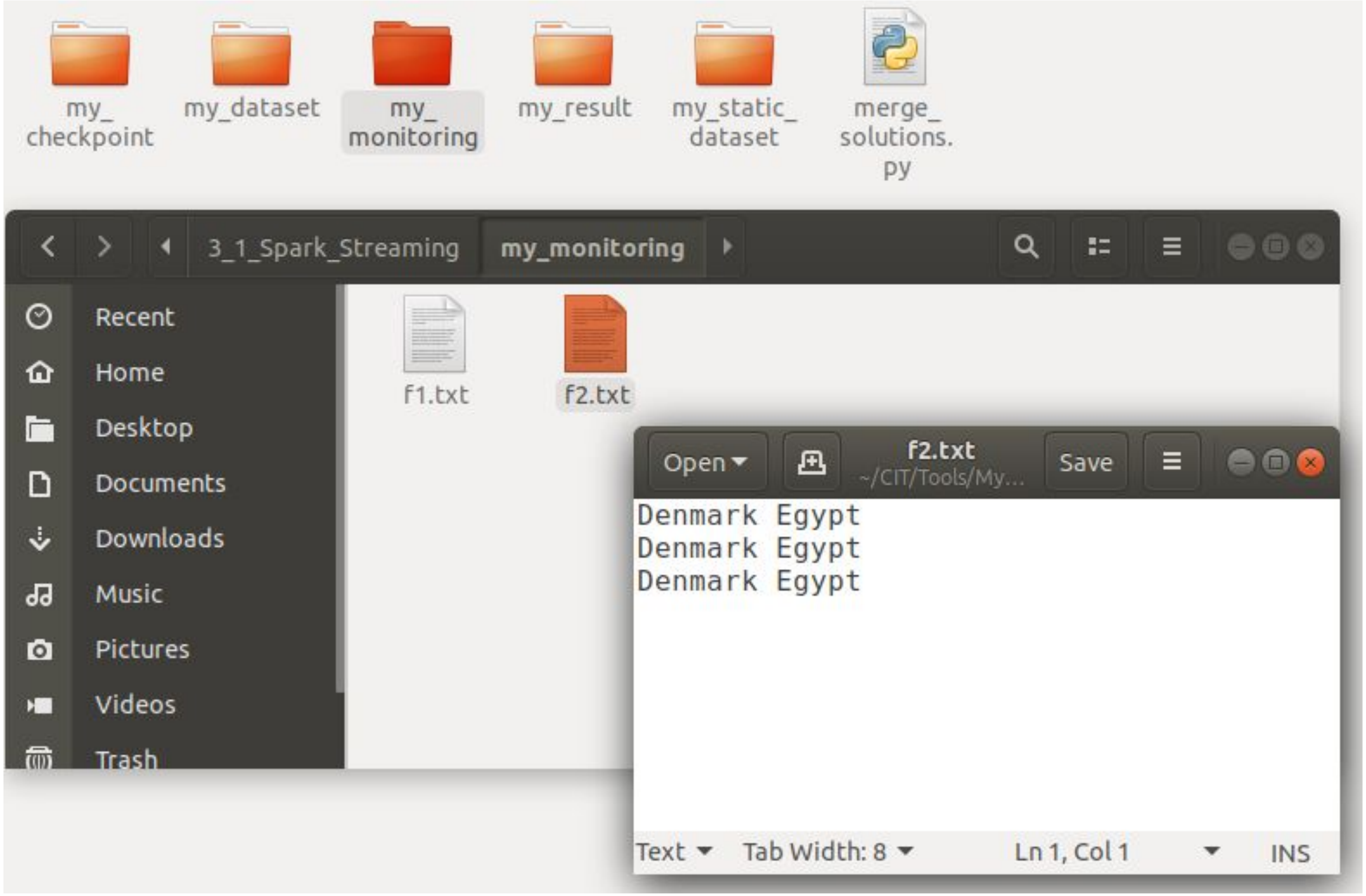


```
Argentina Brazil Colombia
Argentina Brazil Colombia
Argentina Brazil Colombia
```

File Transfer Process

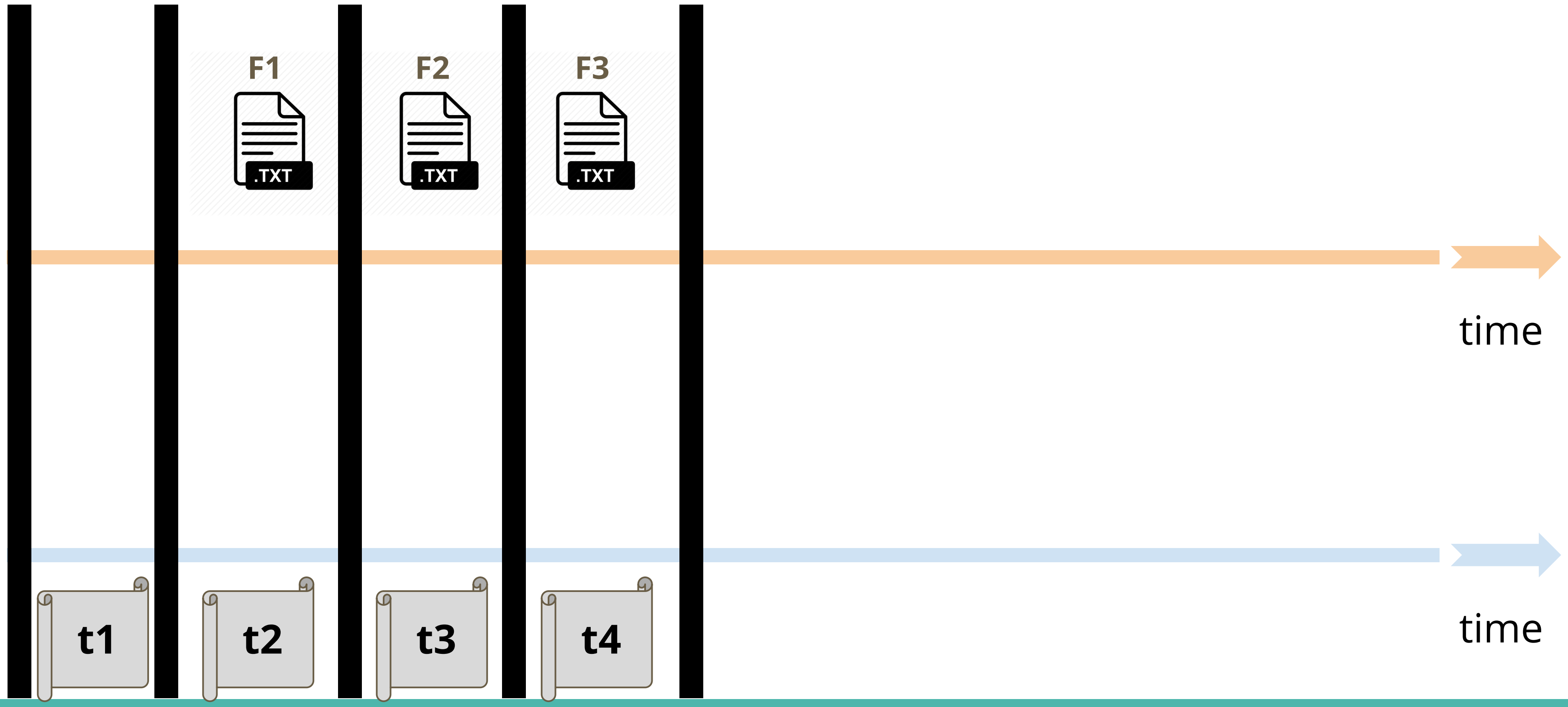


File Transfer Process

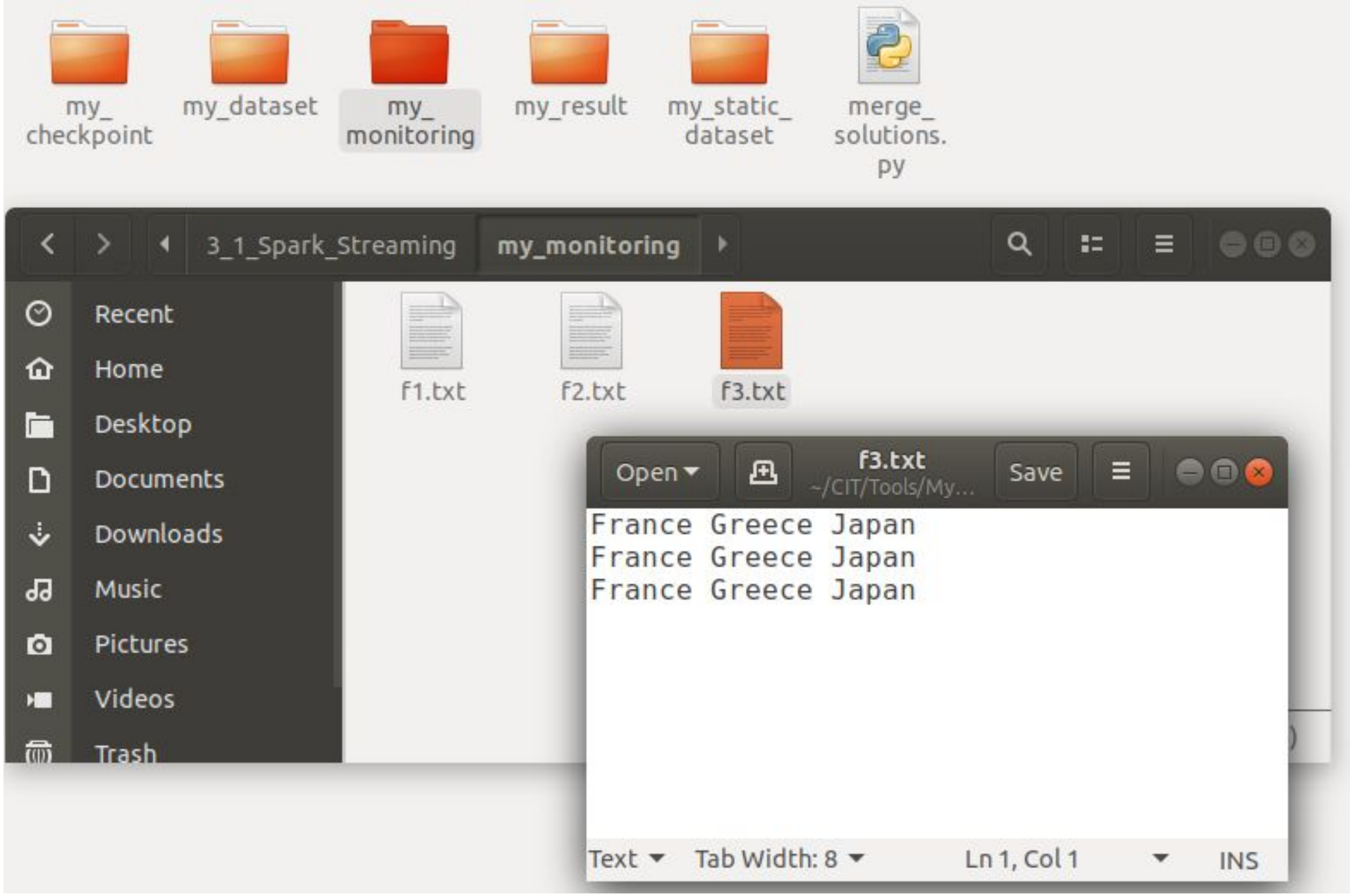


```
Denmark Egypt
Denmark Egypt
Denmark Egypt
```

File Transfer Process

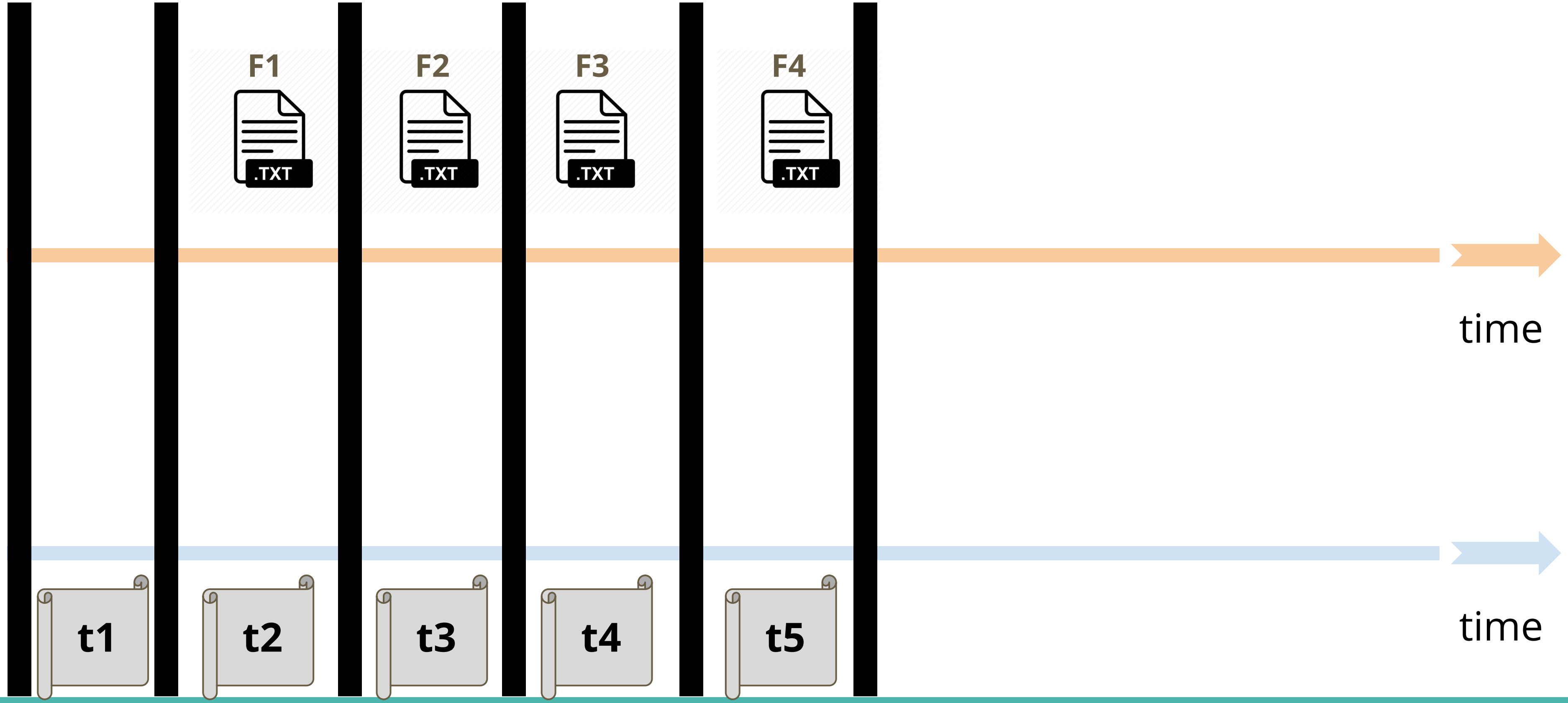


File Transfer Process

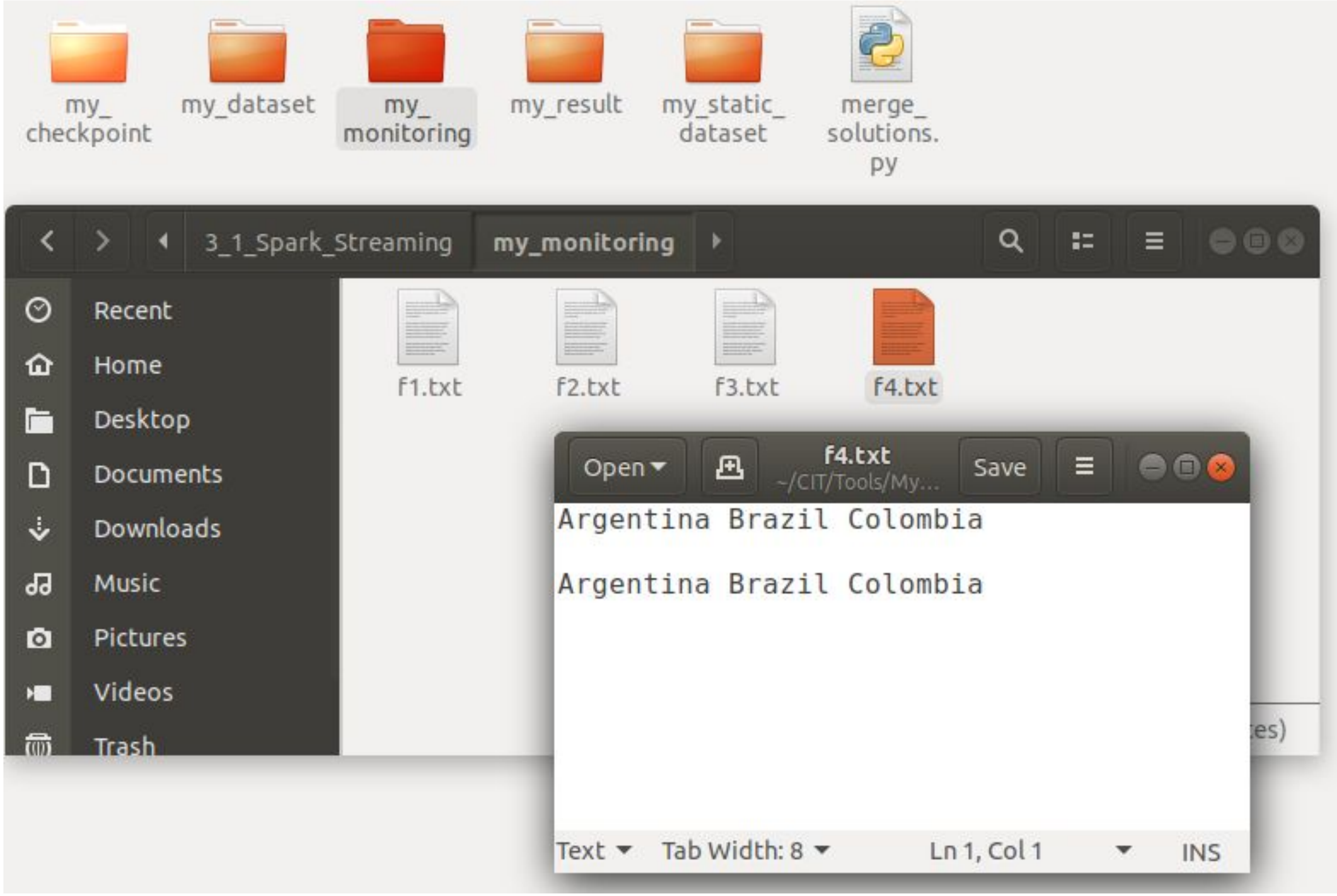


```
France Greece Japan
France Greece Japan
France Greece Japan
```

File Transfer Process

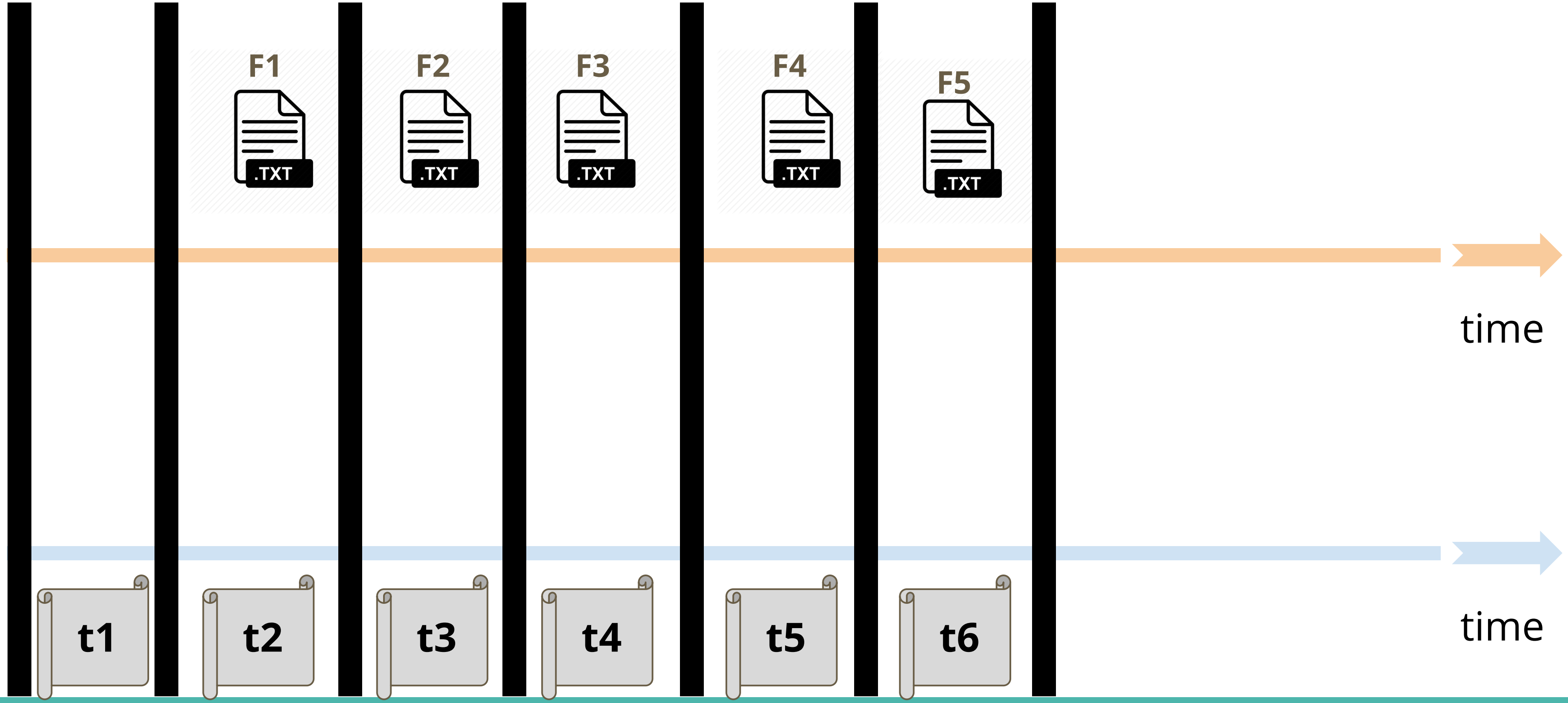


File Transfer Process

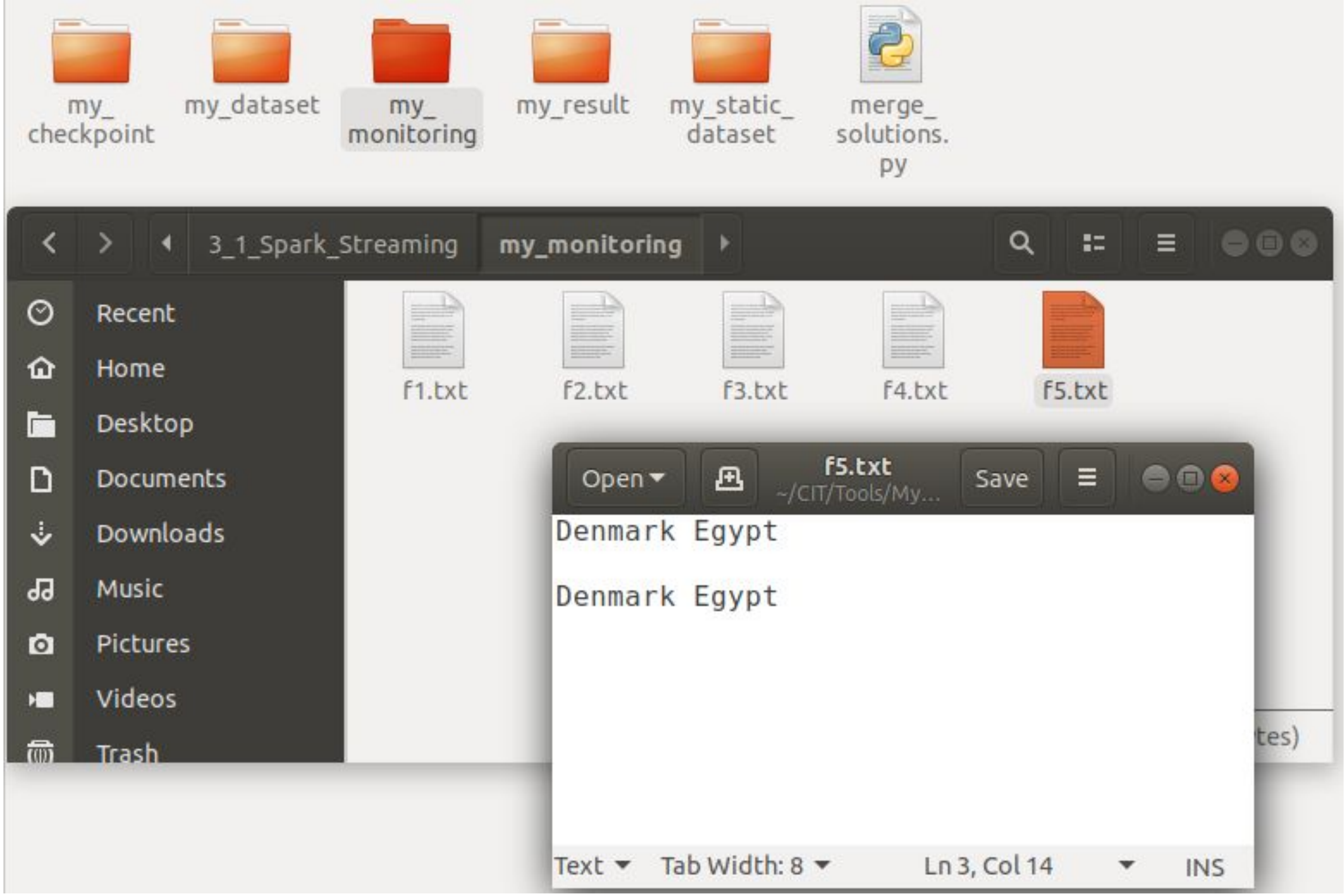


```
Argentina Brazil Colombia

Argentina Brazil Colombia
```

File Transfer Process

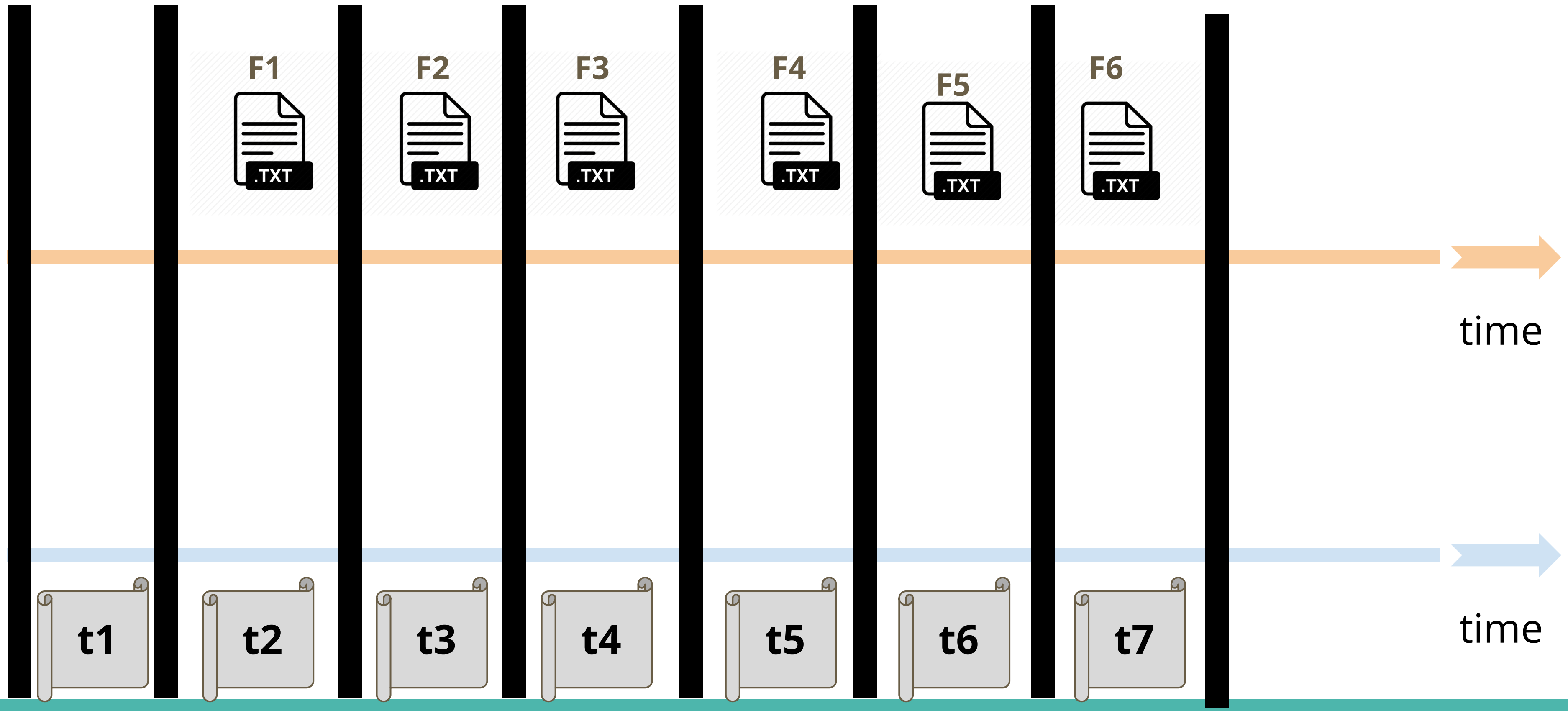


File Transfer Process

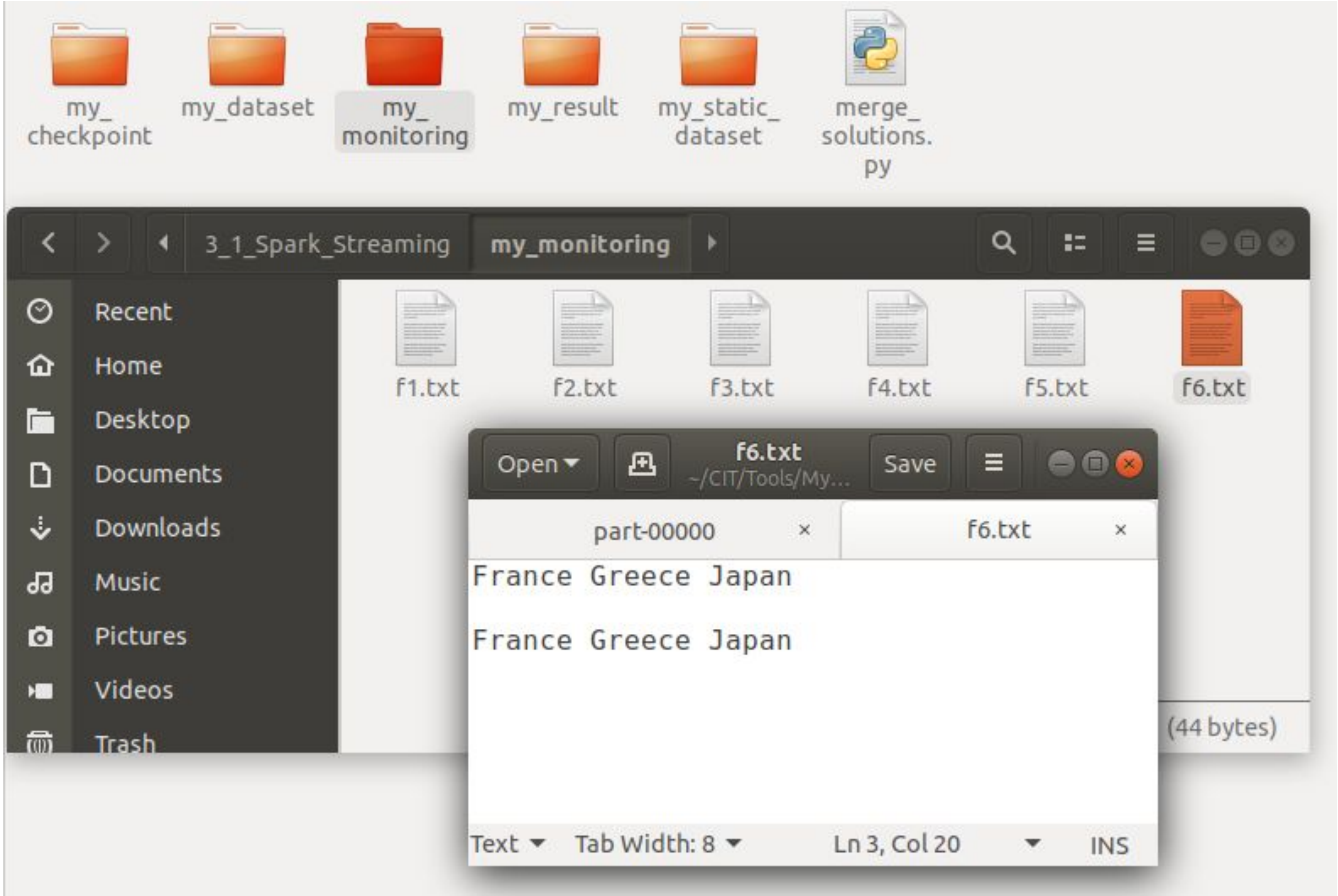


```
Denmark Egypt

Denmark Egypt
```

File Transfer Process

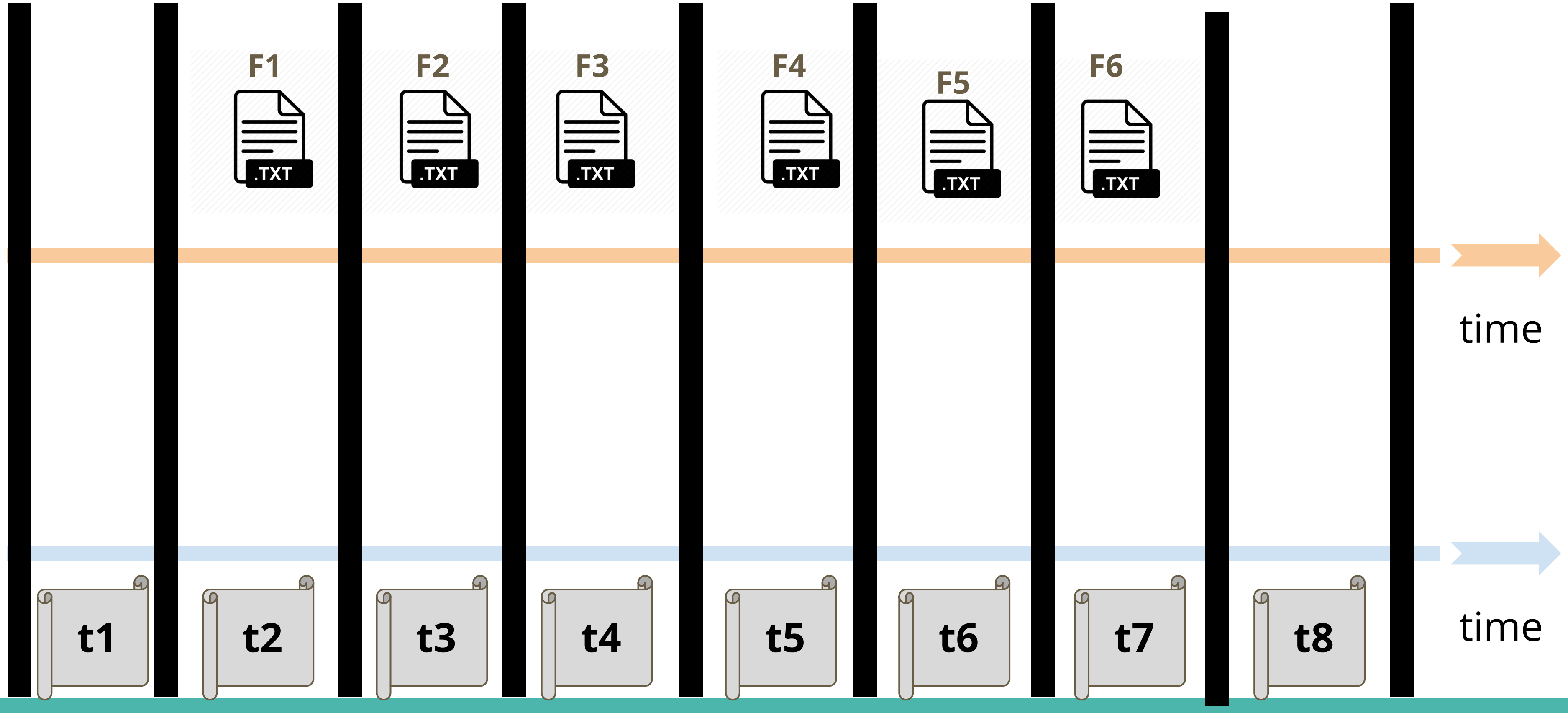


File Transfer Process



```
France Greece Japan

France Greece Japan
```

File Transfer Process



File Transfer Process

The functions `get_source_dir_file_names` and `streaming_simulation` are in charge of such this file transference.

File Transfer Process

The functions `get_source_dir_file_names` and `streaming_simulation` are in charge of such this file transference.

- The function `get_source_dir_file_names` returns a list with the names of all files present in **dataset_dir**.

File Transfer Process

The functions `get_source_dir_file_names` and `streaming_simulation` are in charge of such this file transference.

- The function `get_source_dir_file_names` returns a list with the names of all files present in **dataset_dir**.
- The function `streaming_simulation` transfers the files, one by one, every time_interval seconds.

File Transfer Process

The functions `get_source_dir_file_names` and `streaming_simulation` are in charge of such this file transference.

- The function `get_source_dir_file_names` returns a list with the names of all files present in **dataset_dir**.
- The function `streaming_simulation` transfers the files, one by one, every time_interval seconds.

These functions are present in all our code examples.

Outline

1. Setting Up the Context.
2. Measurement Unit: Time Interval & Data Batch.
3. From RDDs to a DStream.
4. File Transfer Process.
5. Spark Streaming Context Process.
6. Stateless and Stateful Operations.

Spark Streaming Context Process

We have seen our first
Spark Streaming Application,
p02_introDStream.py

Spark Streaming Context Process

We have seen our first
Spark Streaming Application,
p02_introDStream.py

However,
two questions remain still unclear…

Spark Streaming Context Process

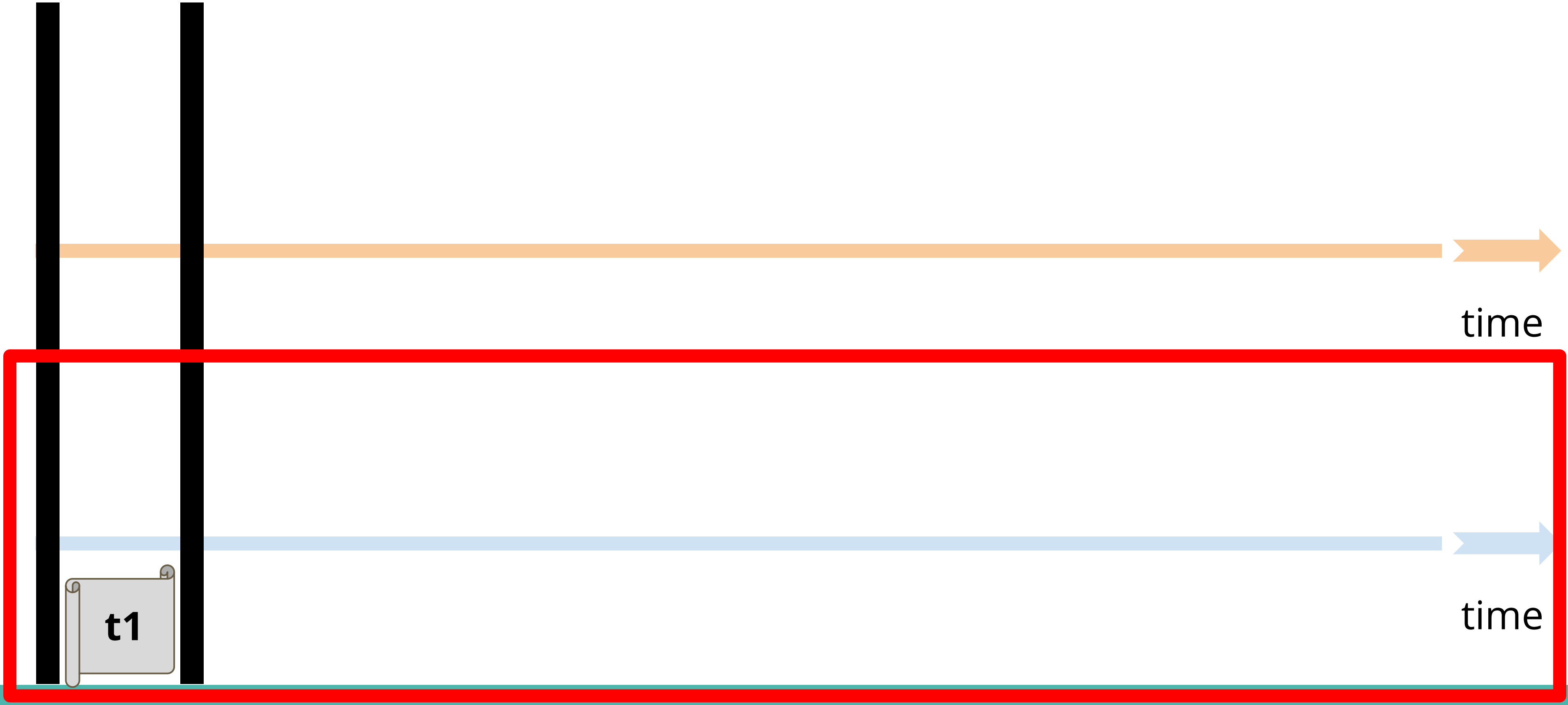
Question 2

How do we configure
the Spark Streaming Context **ssc** on…

Spark Streaming Context Process

Note:

In our diagram, the Spark Streaming Context process is the one on the bottom of the picture.



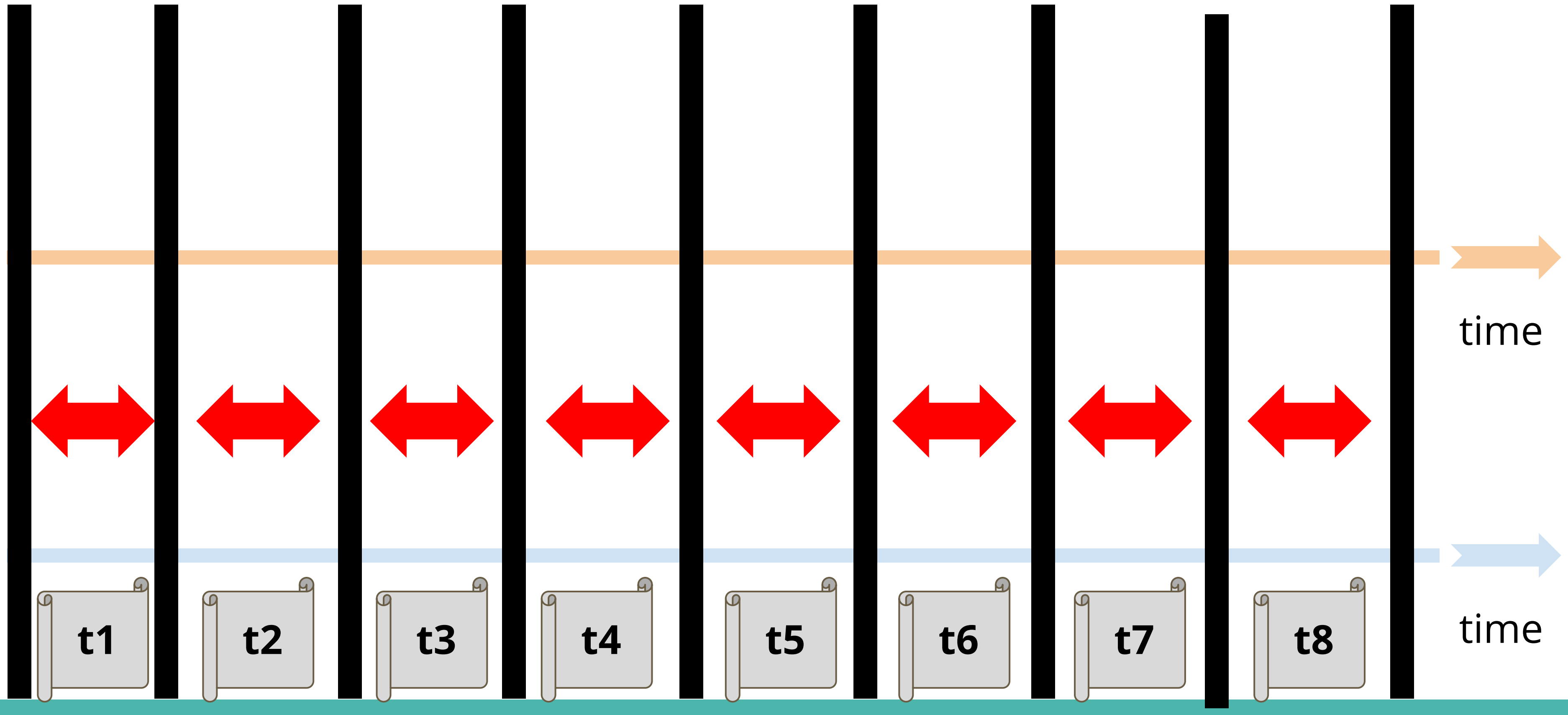
Spark Streaming Context Process

Question 2

How do we configure
the Spark Streaming Context **ssc** on…

- How often to work?
(i.e., length of a time interval?)

Spark Streaming Context Process



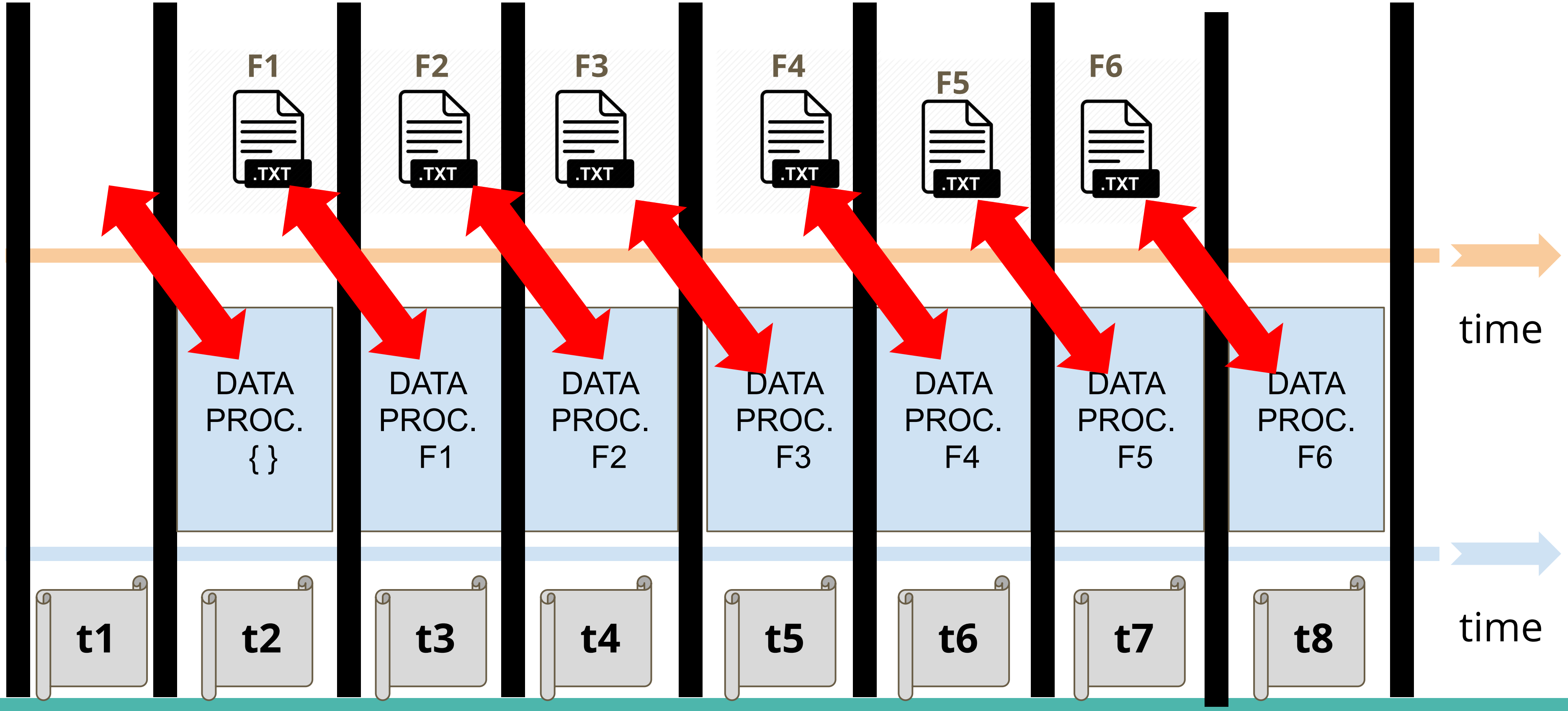
Spark Streaming Context Process

Question 2

How do we configure
the Spark Streaming Context **ssc** on…

- How often to work?
  (i.e., length of a time interval?)
- What to do each time it has to work?
  (i.e., how to process data each time interval?)

Spark Streaming Context Process



Spark Streaming Context Process

- The function `create_ssc` is in charge of these 2 things.
- In particular, for answering the question of what to do on each time interval, `create_ssc` calls to the function `my_model`, which is the one we (as Spark Streaming users) have to program.

Spark Streaming Context Process

- The function `create_ssc` is in charge of these 2 things.
- In particular, for answering the question of what to do on each time interval, `create_ssc` calls to the function `my_model`, which is the one we (as Spark Streaming users) have to program.

These functions are present in all our code examples.
In particular, once again, the function `my_model` will be different on each code example, as it represents the functionality (actual work to do) we want to demonstrate.

Spark Streaming Context Process

Last (but not least),
the function `my_main` puts all the pieces together by starting and stopping the Spark Streaming Context process and the File Transfer process.

Outline

1. Setting Up the Context.
2. Measurement Unit: Time Interval & Data Batch.
3. From RDDs to a DStream.
4. File Transfer Process.
5. Spark Streaming Context Process.
6. Stateless and Stateful Operations.

Stateless and Stateful Operations

All the aforementioned explanations
for p02_introDStream.py
have an important assumption:

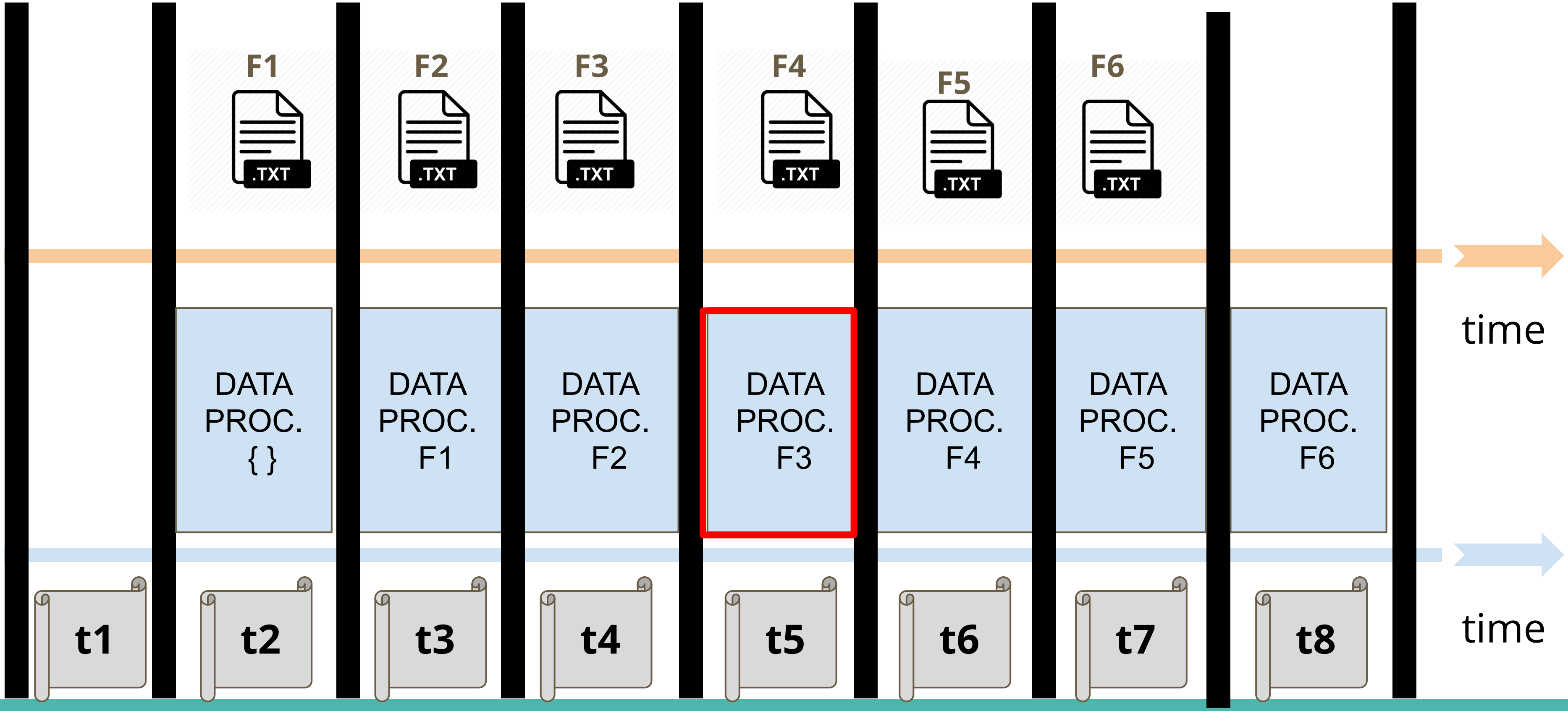
Stateless and Stateful Operations

All the aforementioned explanations for p02_introDStream.py have an important assumption:

The Data Processing being done in **time interval t_i** is completely independent!

Stateless and Stateful Operations

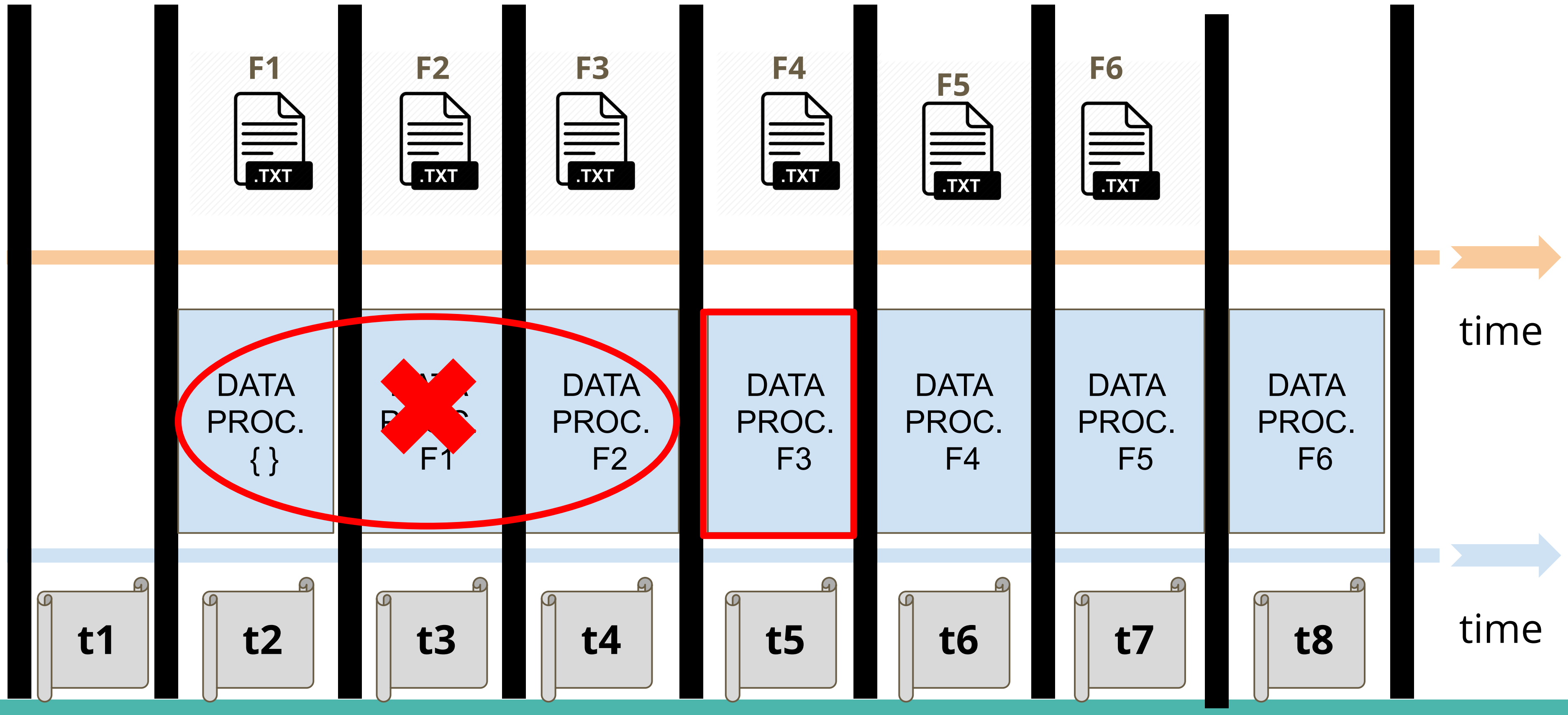
The data processing of p02_introDStream.py being done in **time interval t_i**



Stateless and Stateful Operations

The data processing of p02_introDStream.py being done in **time interval t_i**

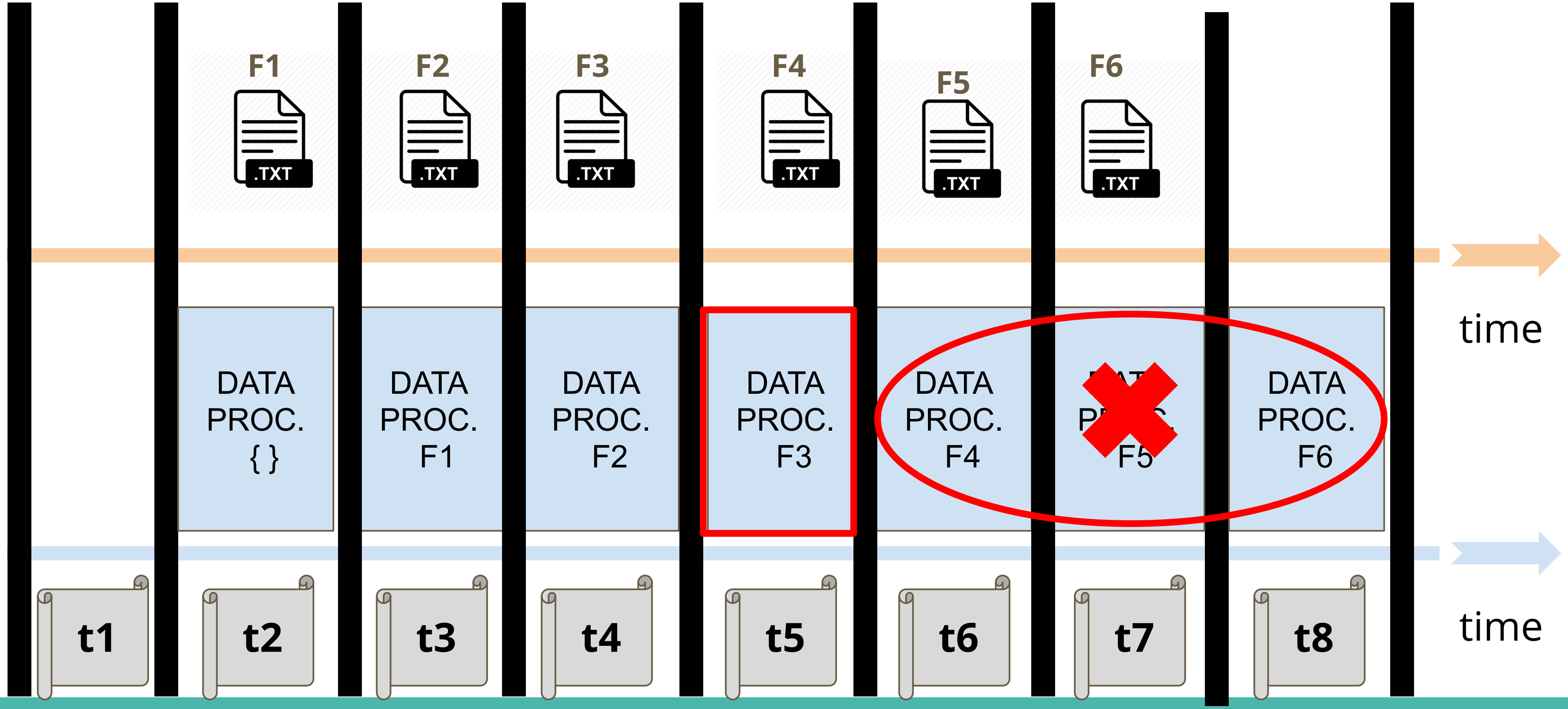
- Does not depend on any previous processing of time intervals t_{i-k}



Stateless and Stateful Operations

The data processing of p02_introDStream.py being done in **time interval t_i**

- Does not depend on any previous processing of time intervals t_{i-k}
- Does not create dependencies on further processing of time intervals t_{i+k}

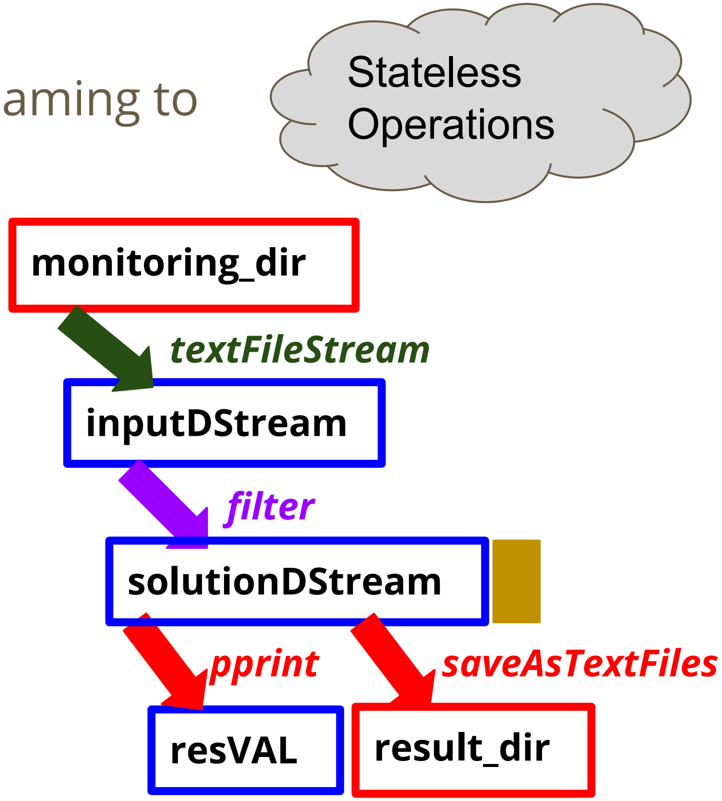


Stateless and Stateful Operations

This is because all the creation and transformation operations used in `my_model` in p02_introDStream.py are Stateless Operations!

p02_introDStream.py

1. Read in all the lines of the files arriving in streaming to my_monitoring_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Print them by the screen.
5. Save them to the new directory my_result_dir.



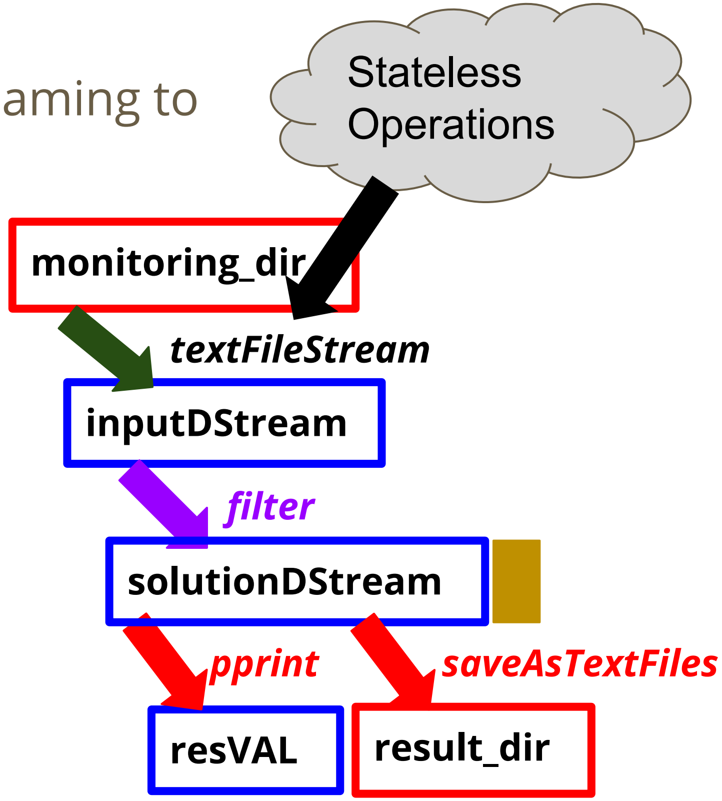
A high level view of its operations is presented next:

Stateless and Stateful Operations

This is because all the creation and transformation operations used in `my_model` in p02_introDStream.py are Stateless Operations!

p02_introDStream.py

1. Read in all the lines of the files arriving in streaming to my_monitoring_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Print them by the screen.
5. Save them to the new directory my_result_dir.



A high level view of its operations is presented next:

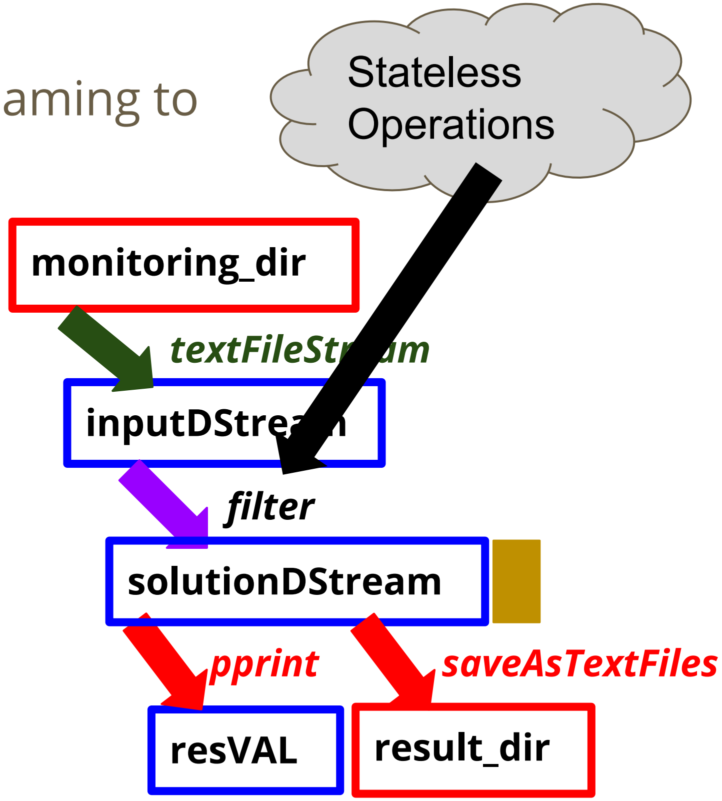
Stateless and Stateful Operations

This is because all the creation and transformation operations used in `my_model` in p02_introDStream.py are Stateless Operations!

p02_introDStream.py

1. Read in all the lines of the files arriving in streaming to my_monitoring_dir.
2. Filter the ones with enough length.
3. Persist the results as they will be used twice.
4. Print them by the screen.
5. Save them to the new directory my_result_dir.

A high level view of its operations is presented next:



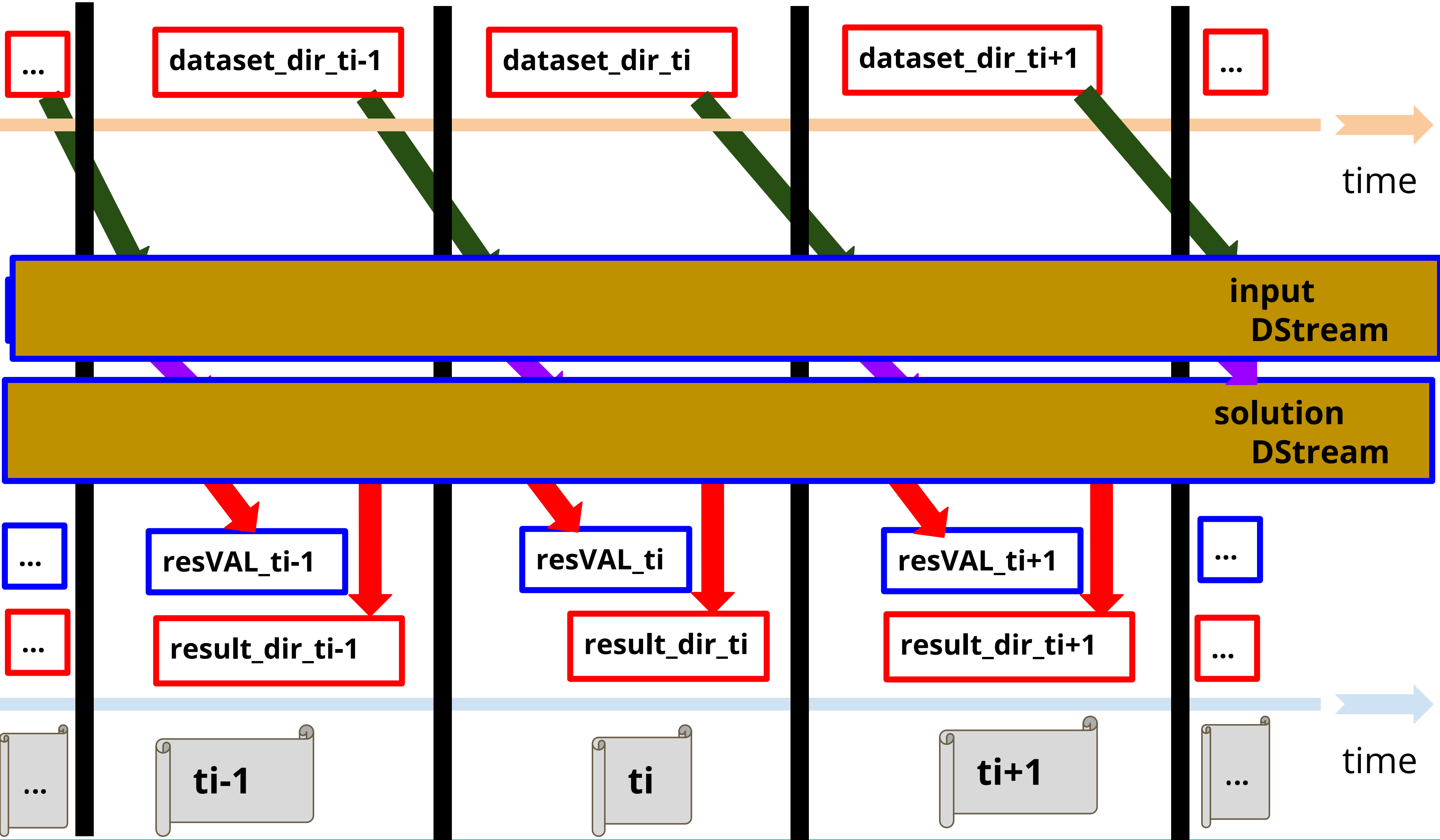
Stateless and Stateful Operations

Coming back to our **train** and **wagons** metaphor, all wagons are independent. Or, in other words, the **DStream** reasons about each **RDD** separately.

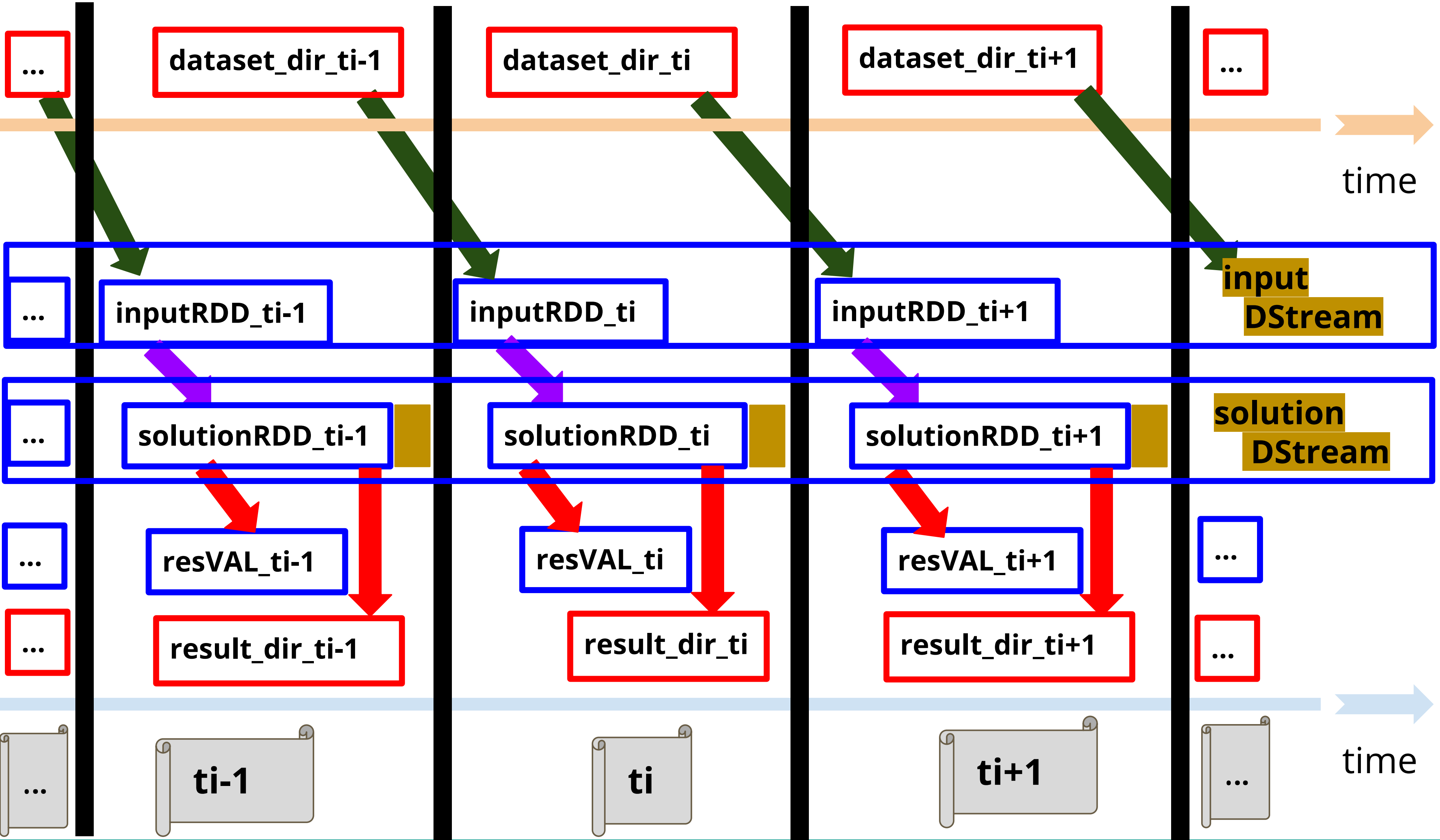
...

...

Stateless and Stateful Operations



Stateless and Stateful Operations



...

dataset_dir_ti-1

dataset_dir_ti

dataset_dir_ti+1

...

time

...

inputRDD_ti-1

inputRDD_ti

inputRDD_ti+1

input DStream

...

solutionRDD_ti-1

solutionRDD_ti

solutionRDD_ti+1

solution DStream

...

resVAL_ti-1

resVAL_ti

resVAL_ti+1

...

...

result_dir_ti-1

result_dir_ti

result_dir_ti+1

...

...

ti-1

ti

ti+1

...

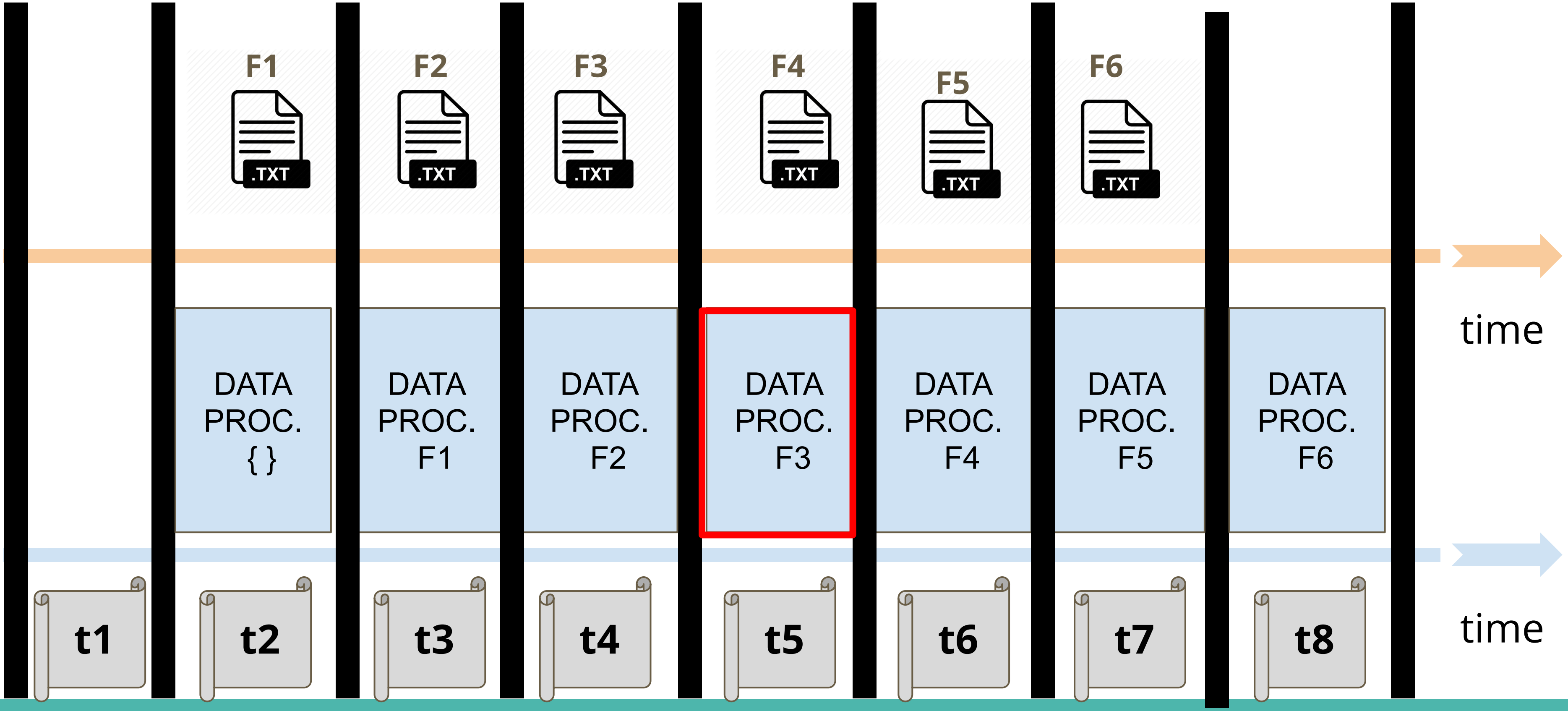
time

Stateless and Stateful Operations

However, Spark Streaming also provides Stateful Operations!

Stateless and Stateful Operations

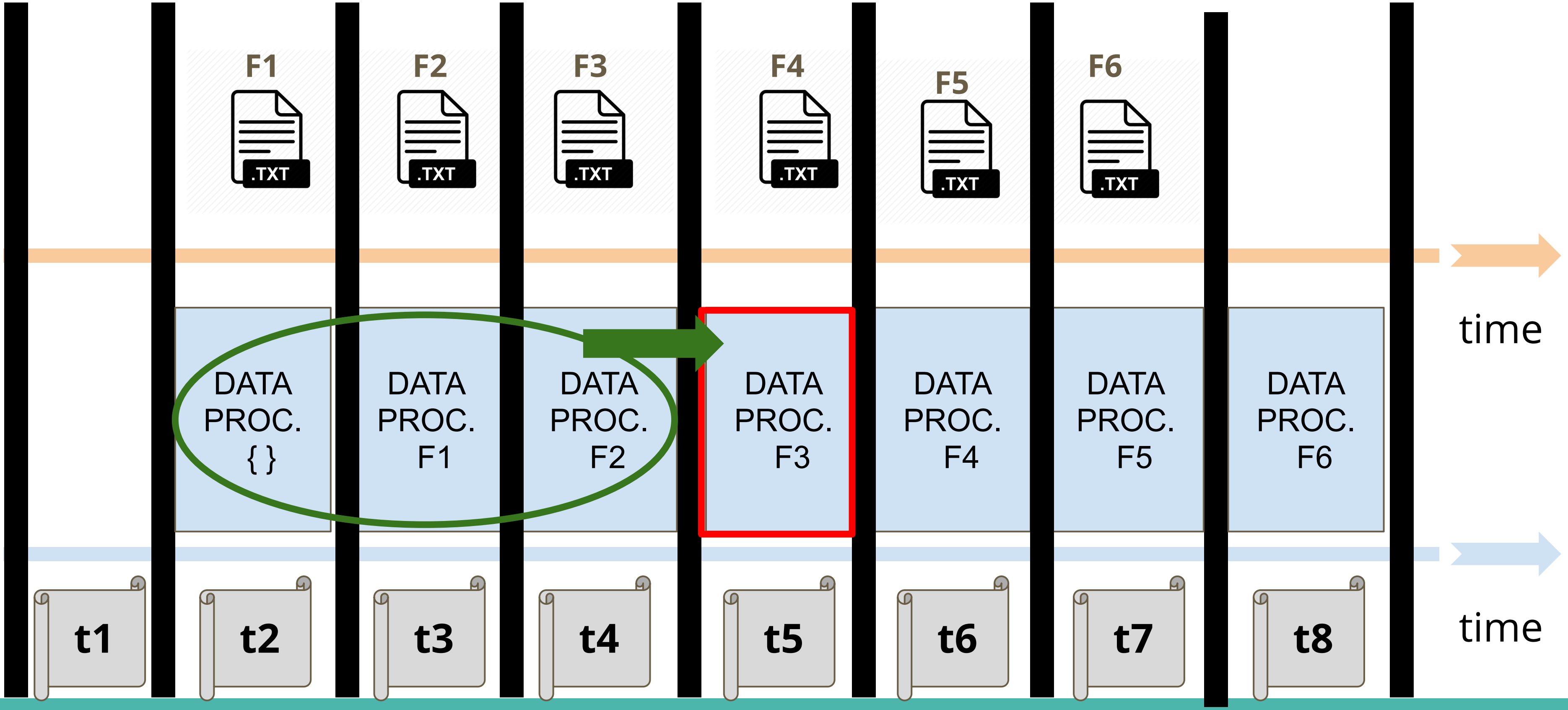
That is, operations where the data processing of **time interval t_i**



Stateless and Stateful Operations

That is, operations where the data processing of **time interval t_i**

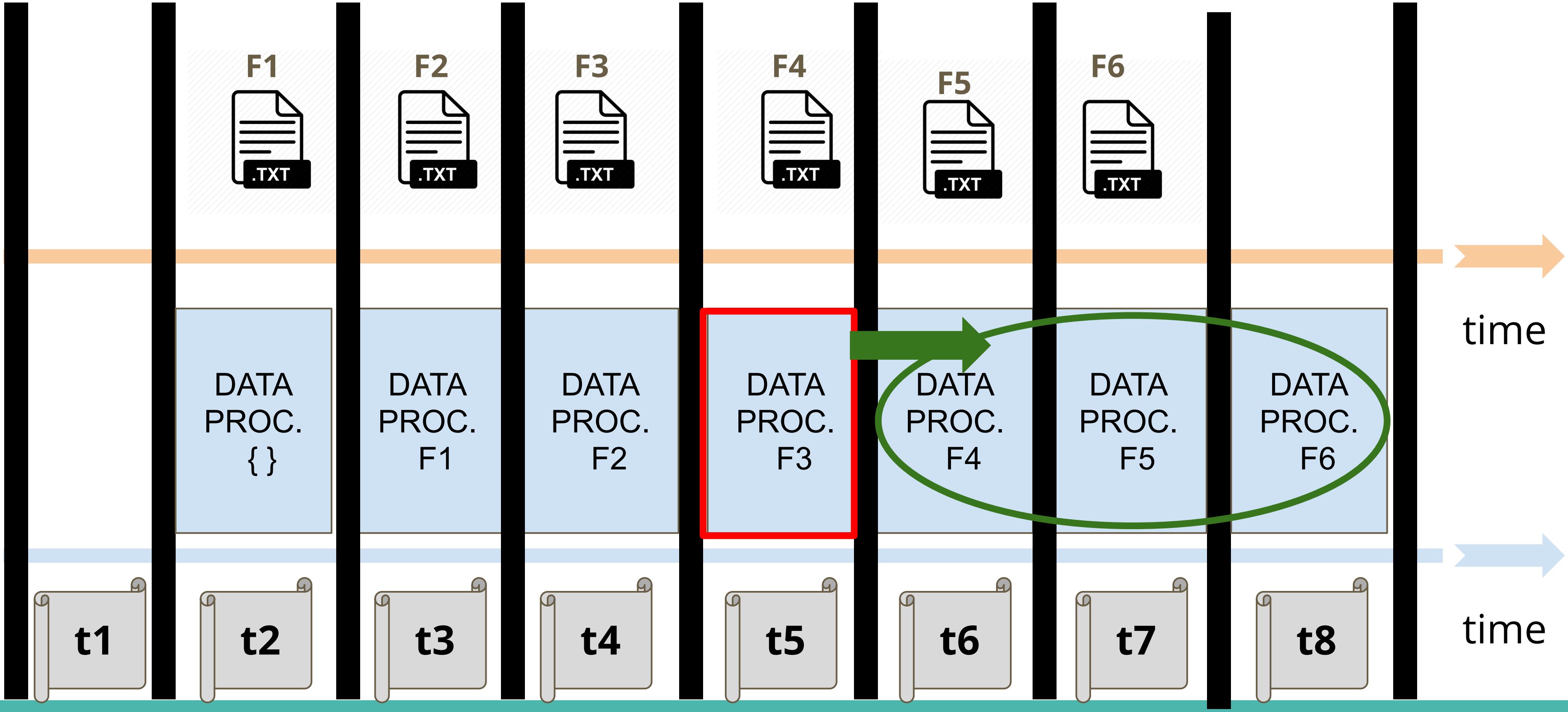
- Depends on any previous processing of time intervals t_{i-k}



Stateless and Stateful Operations

That is, operations where the data processing of **time interval t_i**

- Depends on any previous processing of time intervals t_{i-k}
- Create dependencies on further processing of time intervals t_{i+k}



Stateless and Stateful Operations

Coming back to our **train** and **wagons** metaphor, now wagons have dependencies. Or, in other words, the **DStream** reasons by **putting together groups of RDDs**.

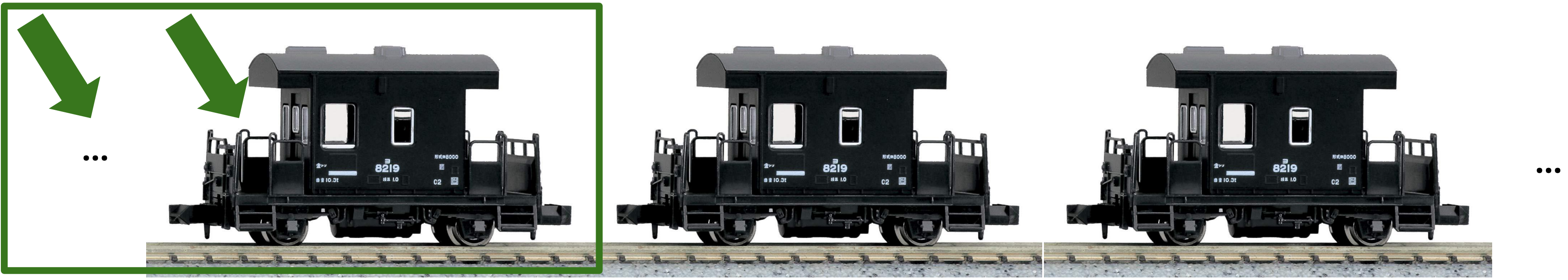
Stateless and Stateful Operations

Let’s see the first category of Stateful Operations:

Window-based Operations

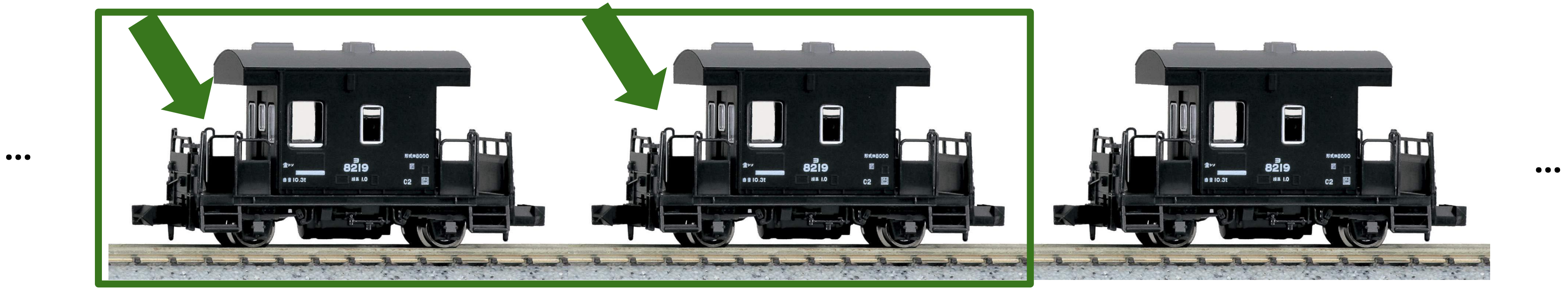
Stateless and Stateful Operations

On Window-based operations, the **DStream** **puts together groups of RDDs** over time.



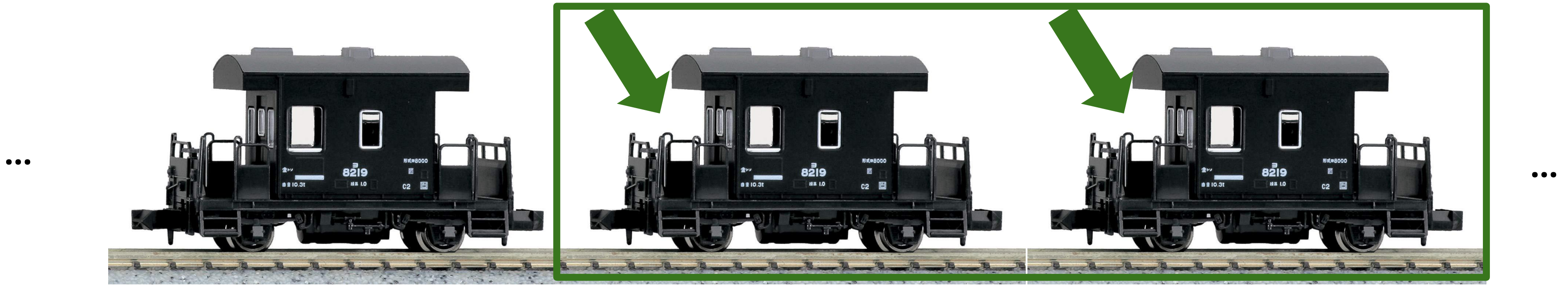
Stateless and Stateful Operations

On Window-based operations, the **DStream puts together groups of RDDs** over time.



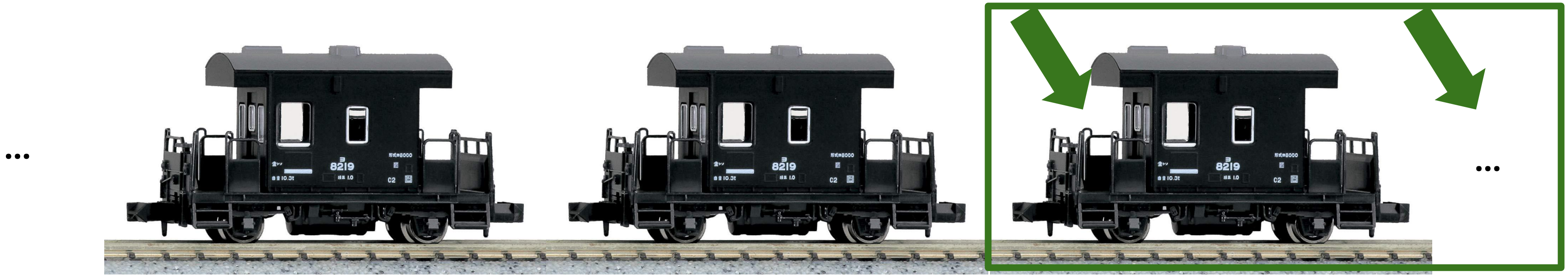
Stateless and Stateful Operations

On Window-based operations, the **DStream** **puts together groups of RDDs** over time.



Stateless and Stateful Operations

On Window-based operations, the **DStream** **puts together groups of RDDs** over time.



Stateless and Stateful Operations

Each window-based operation requires two parameters:

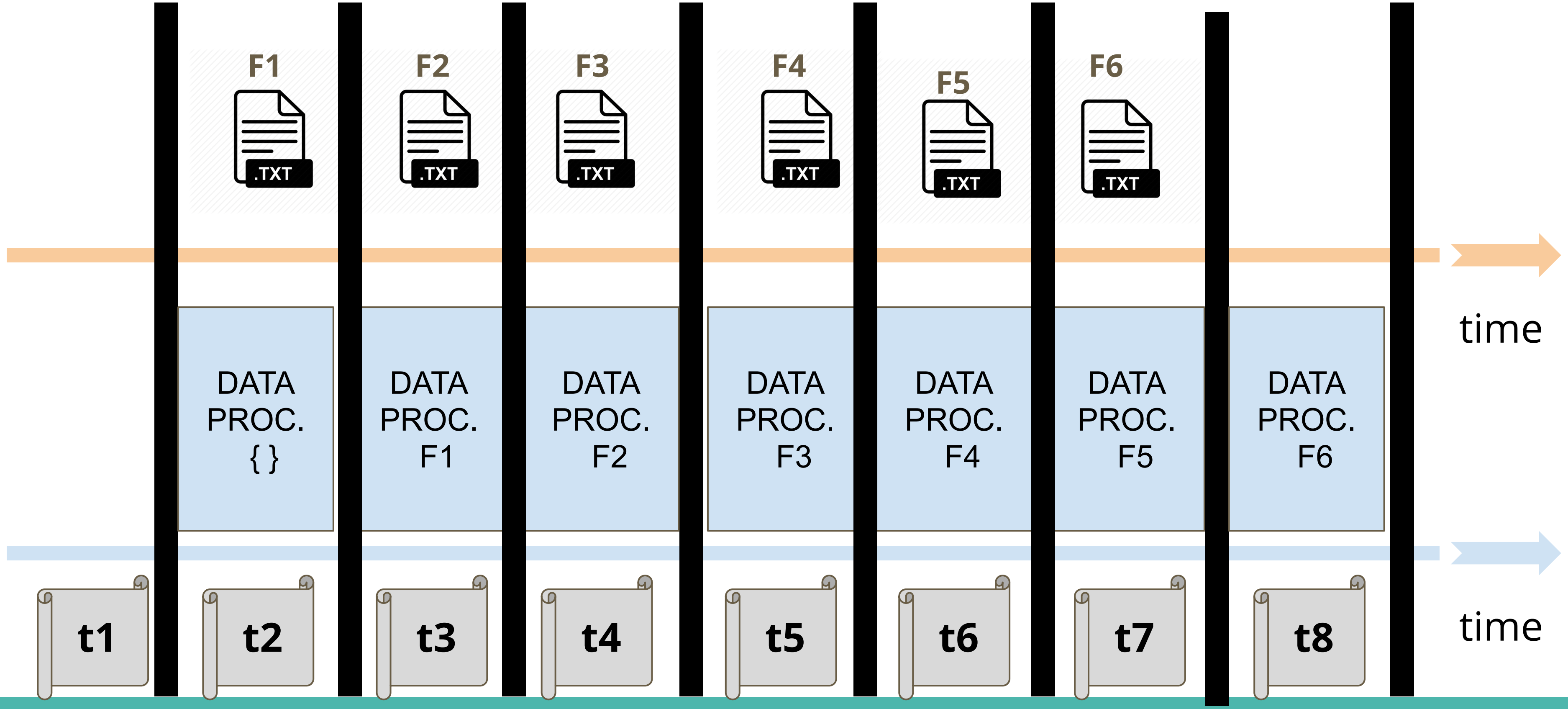
Stateless and Stateful Operations

Each window-based operation requires two parameters:

1. **Sliding Duration:** How often do you want to create a new window?

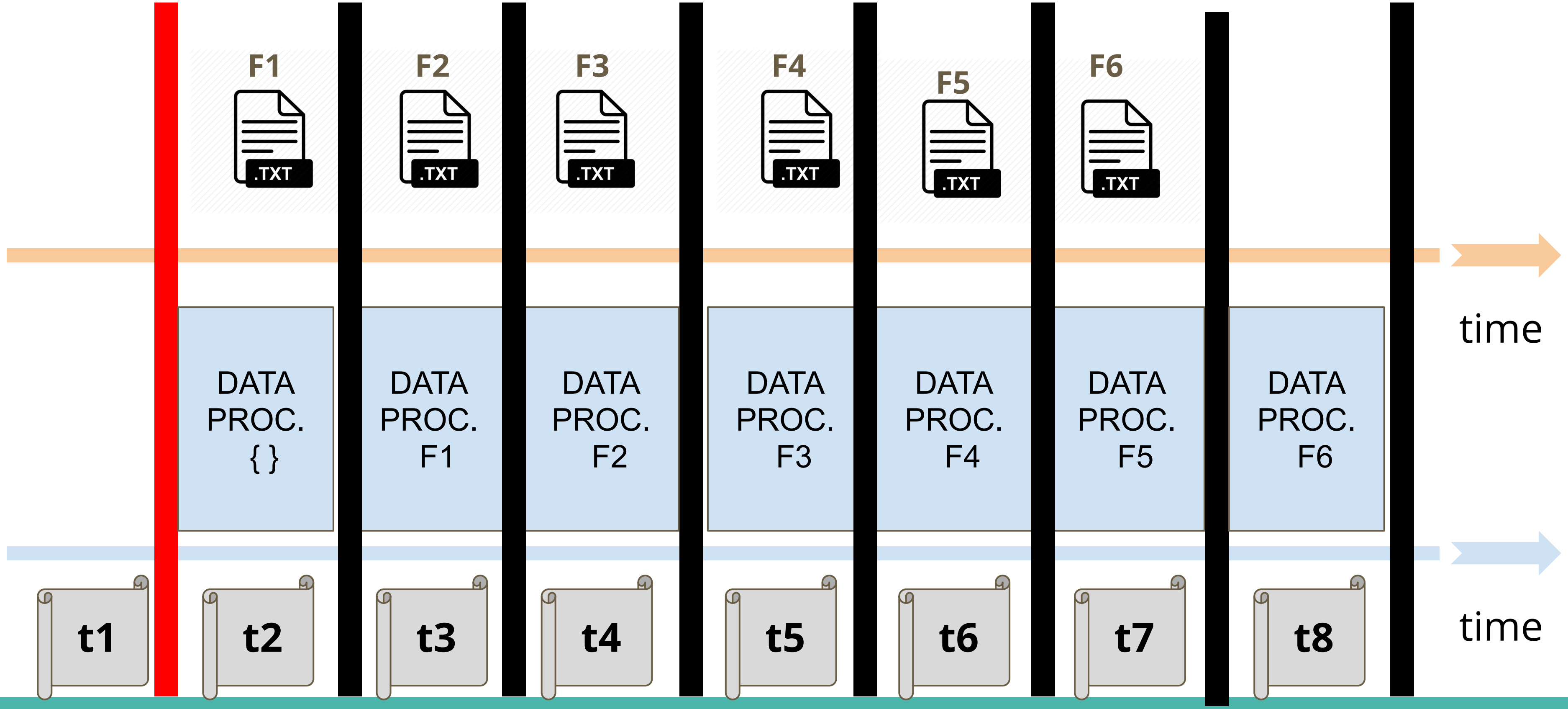
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2**,
we create a new window at the following times



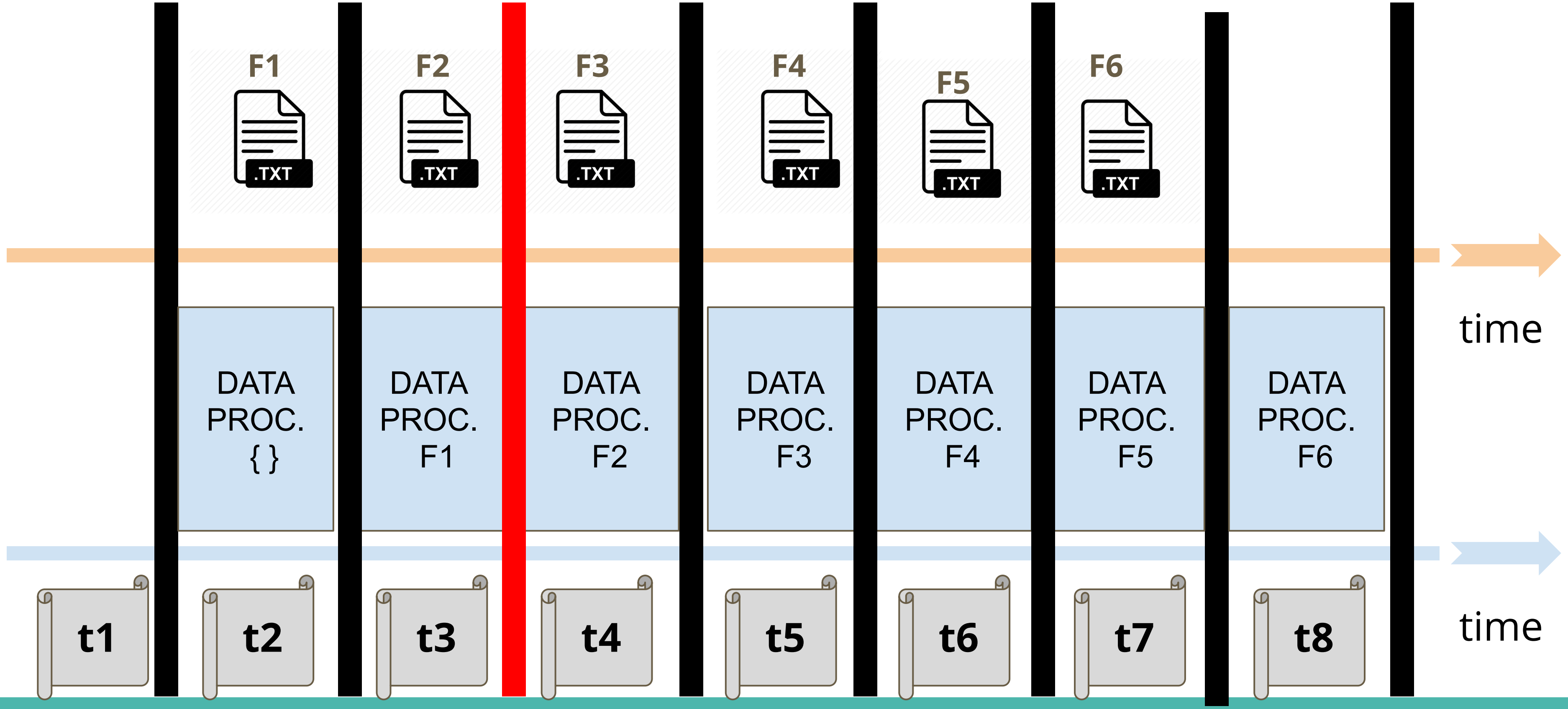
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2**,
we create a new window at the following times



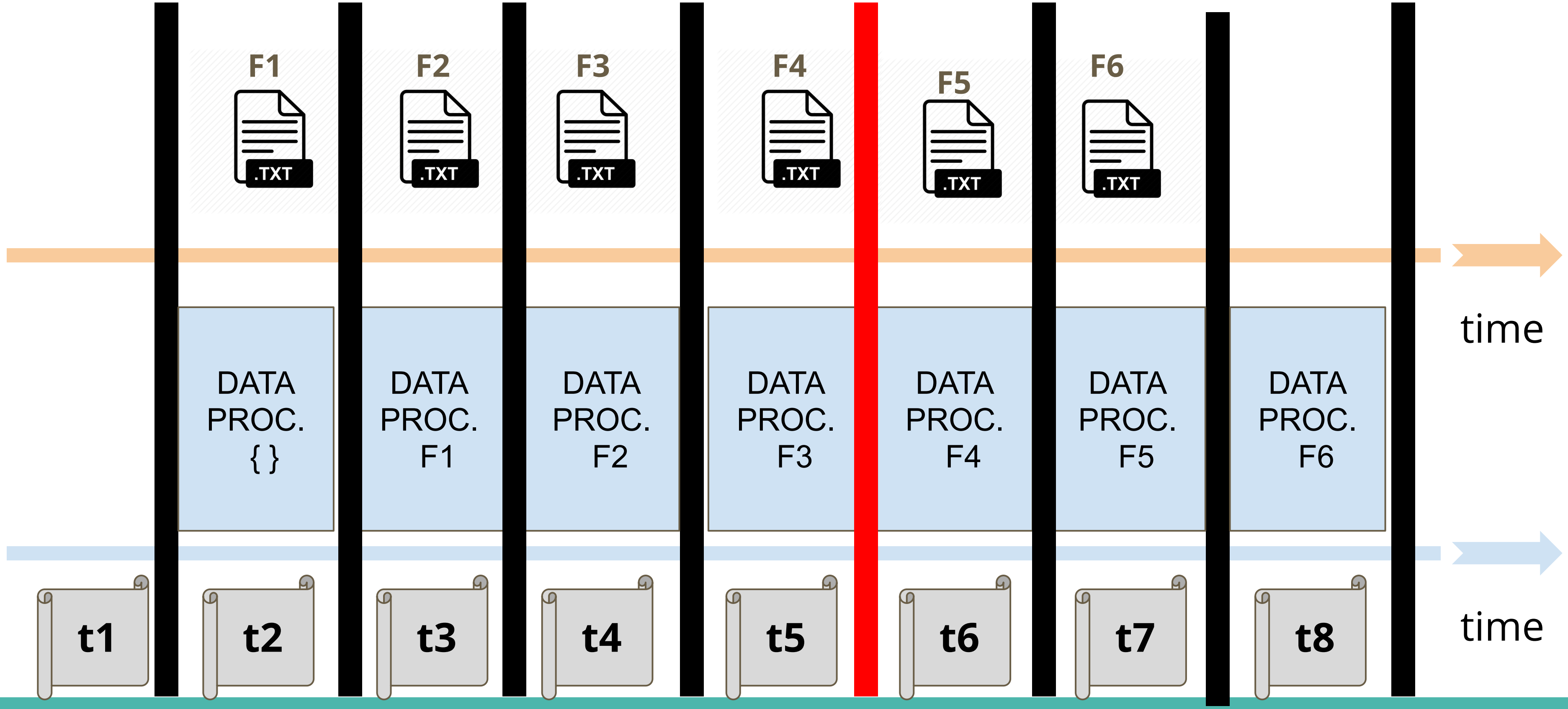
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2**,
we create a new window at the following times



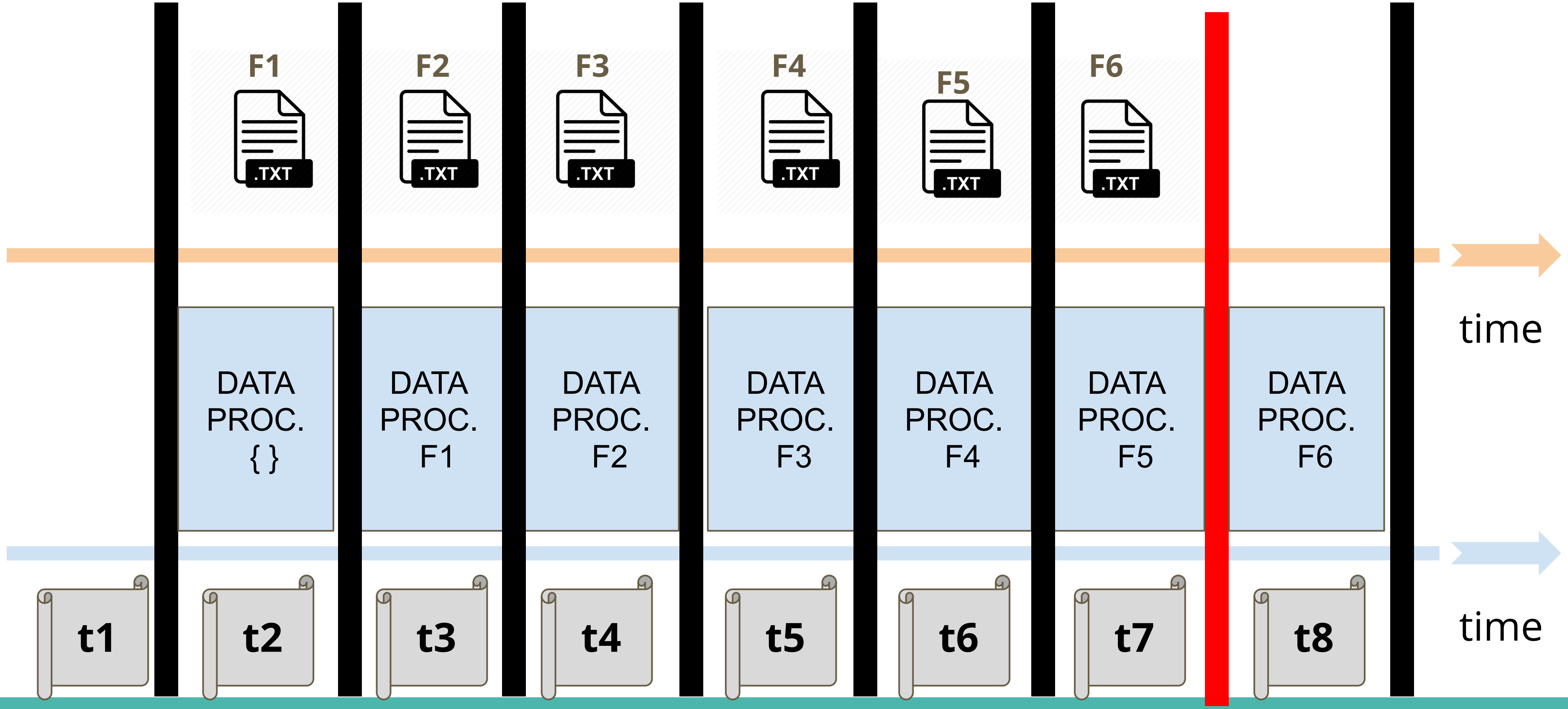
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2**,
we create a new window at the following times



Stateless and Stateful Operations

For example, with a **Sliding Duration = 2**,
we create a new window at the following times



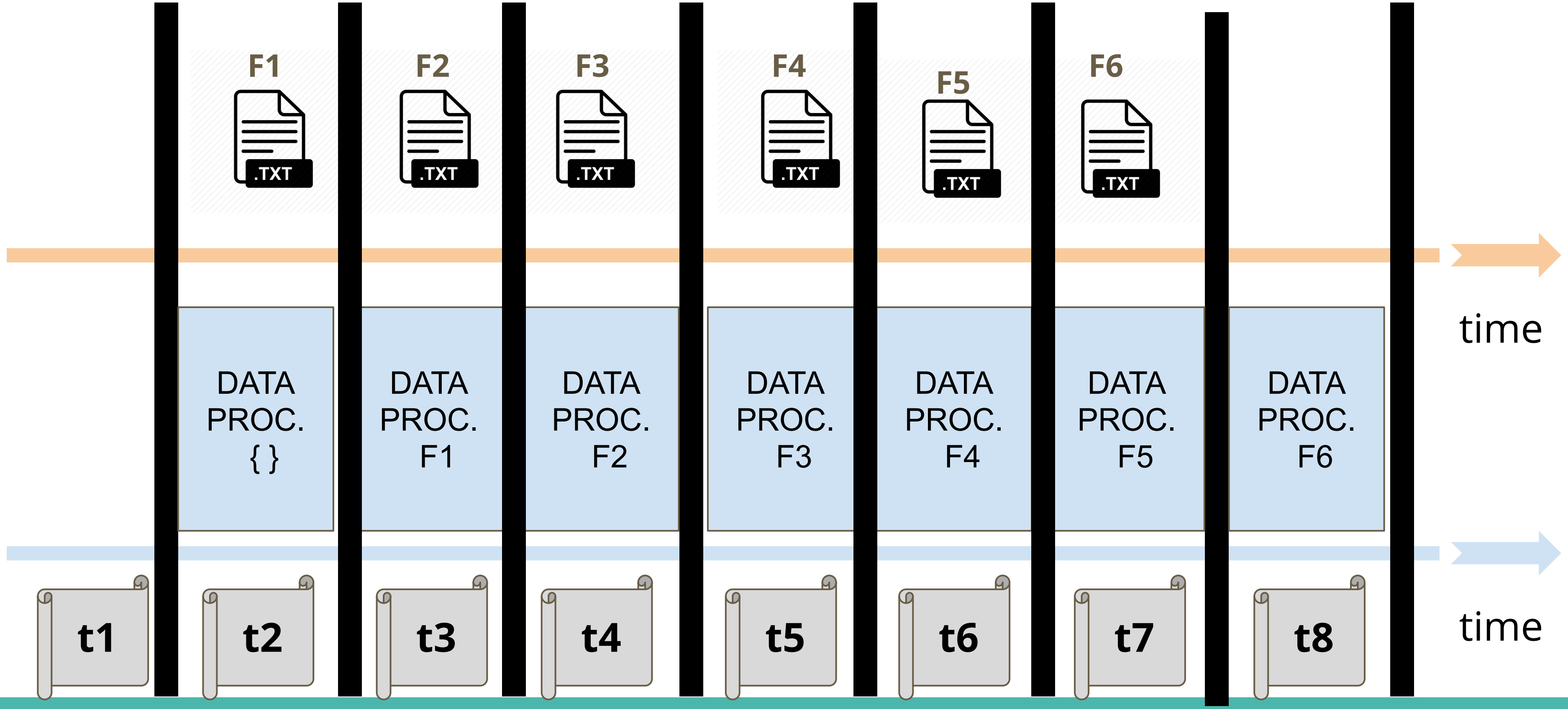
Stateless and Stateful Operations

Each window-based operation requires two parameters:

1. **Sliding Duration:** How often do you want to create a new window?
2. **Window Duration:** Each time a new window is created….
   how far do you want to look backwards?

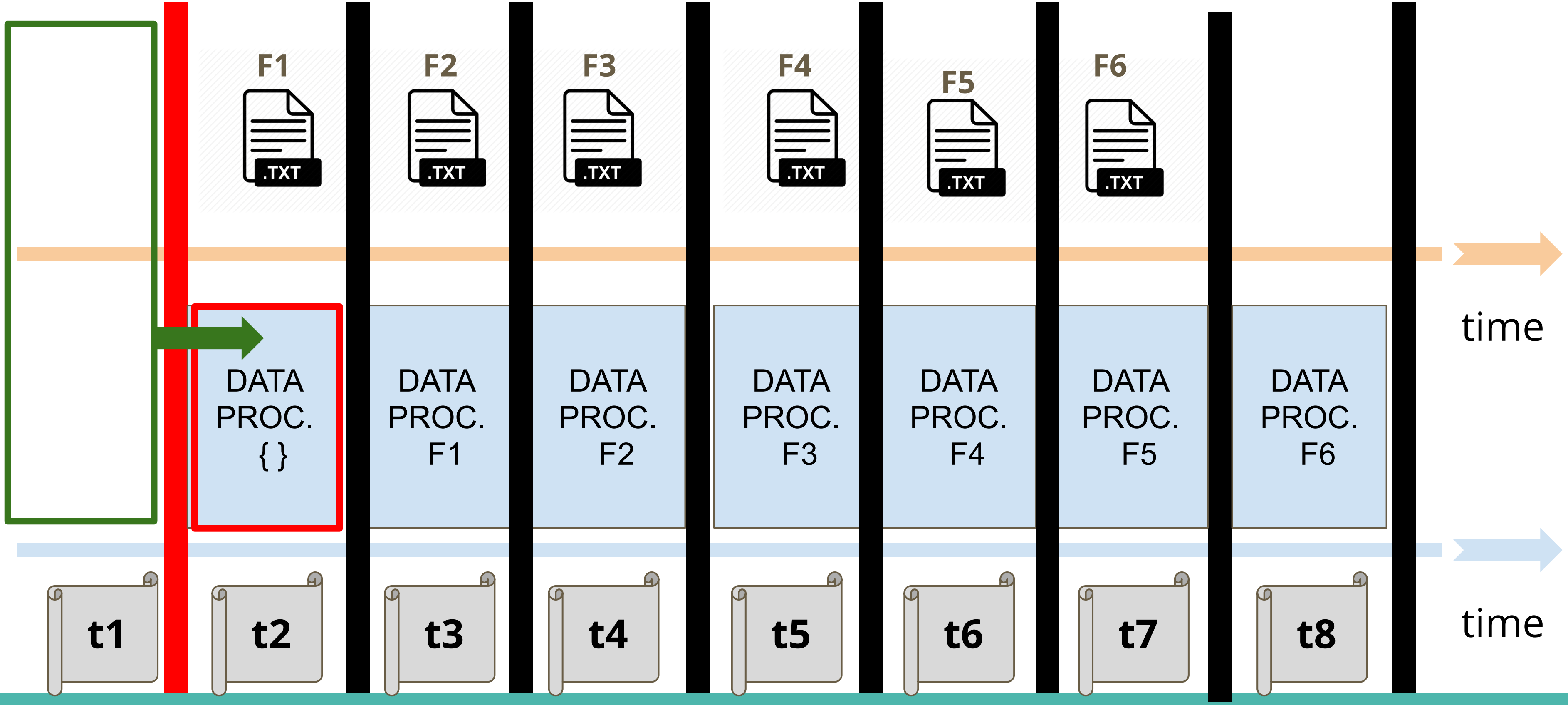
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2** and **Window Duration = 2** we create the following windows at the following times



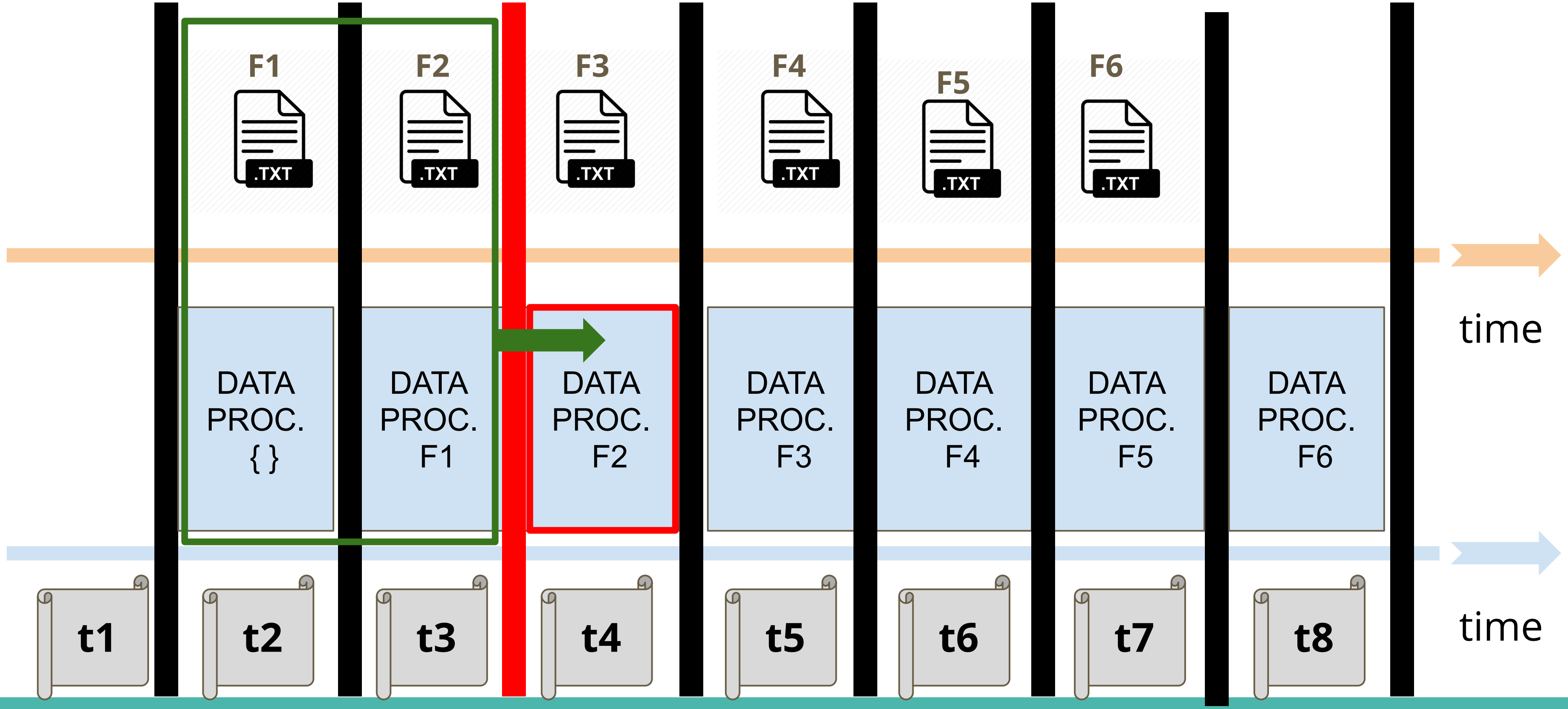
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2** and **Window Duration = 2** we create the following windows at the following times



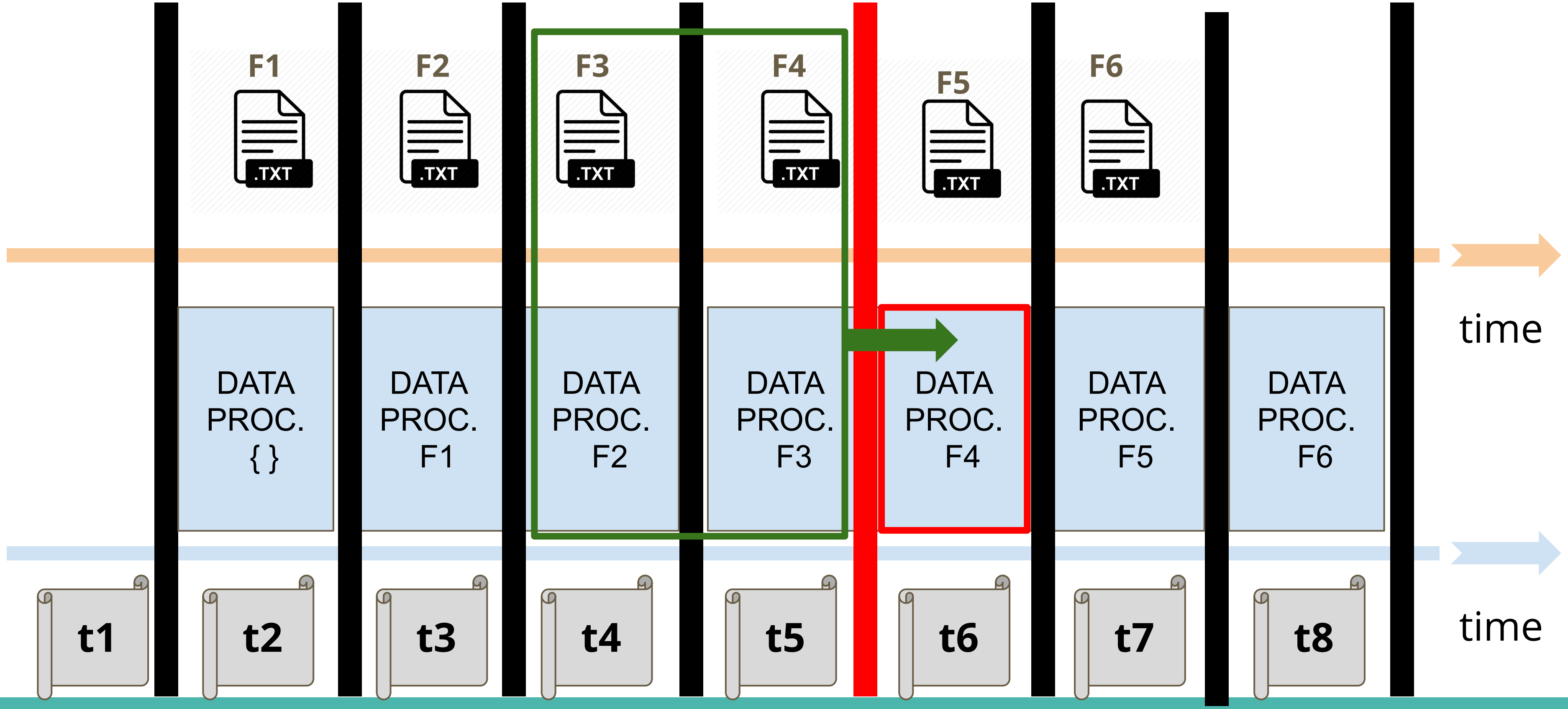
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2** and **Window Duration = 2** we create the following windows at the following times



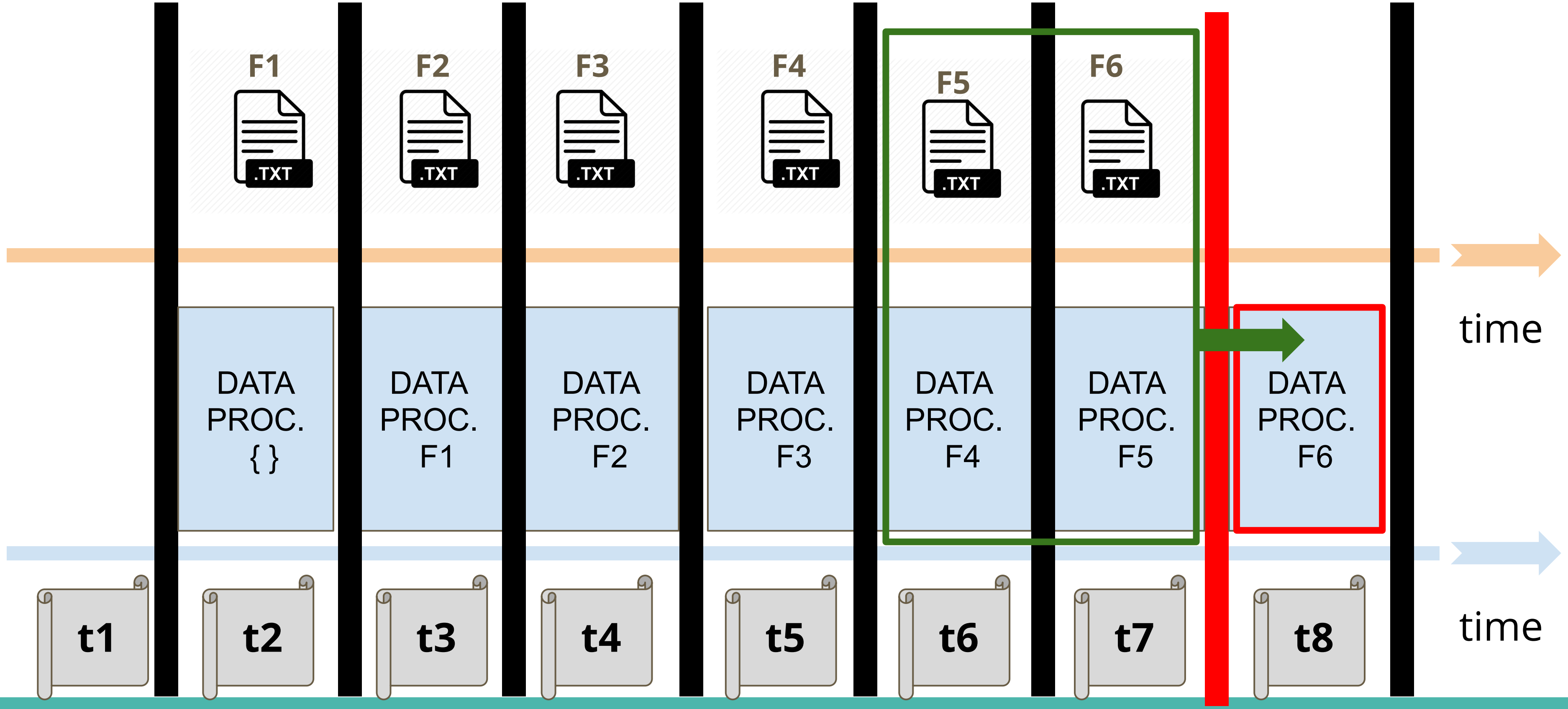
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2** and **Window Duration = 2** we create the following windows at the following times



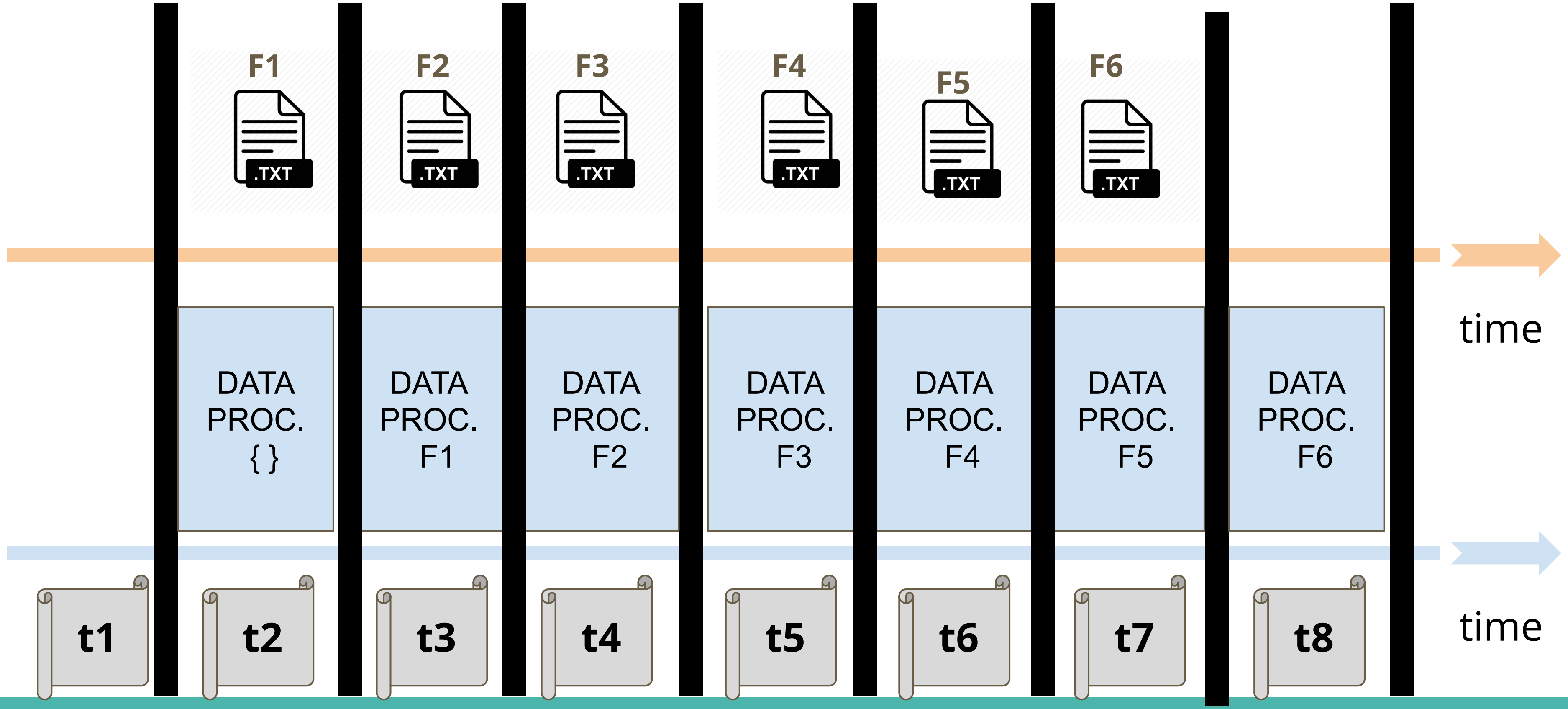
Stateless and Stateful Operations

For example, with a **Sliding Duration = 2** and **Window Duration = 2** we create the following windows at the following times



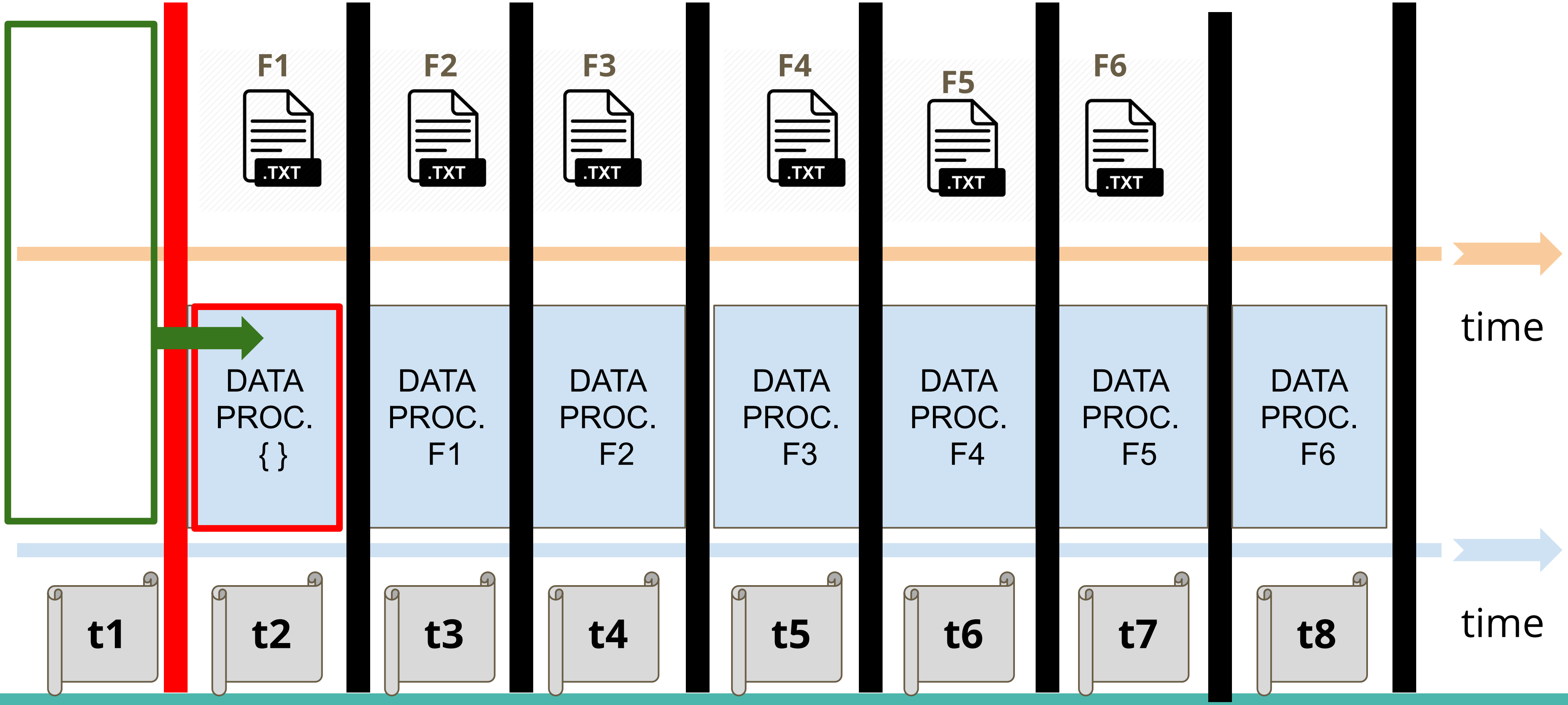
Stateless and Stateful Operations

Another example, now with a **Sliding Duration = 2** and **Window Duration = 3**
we create the following windows at the following times



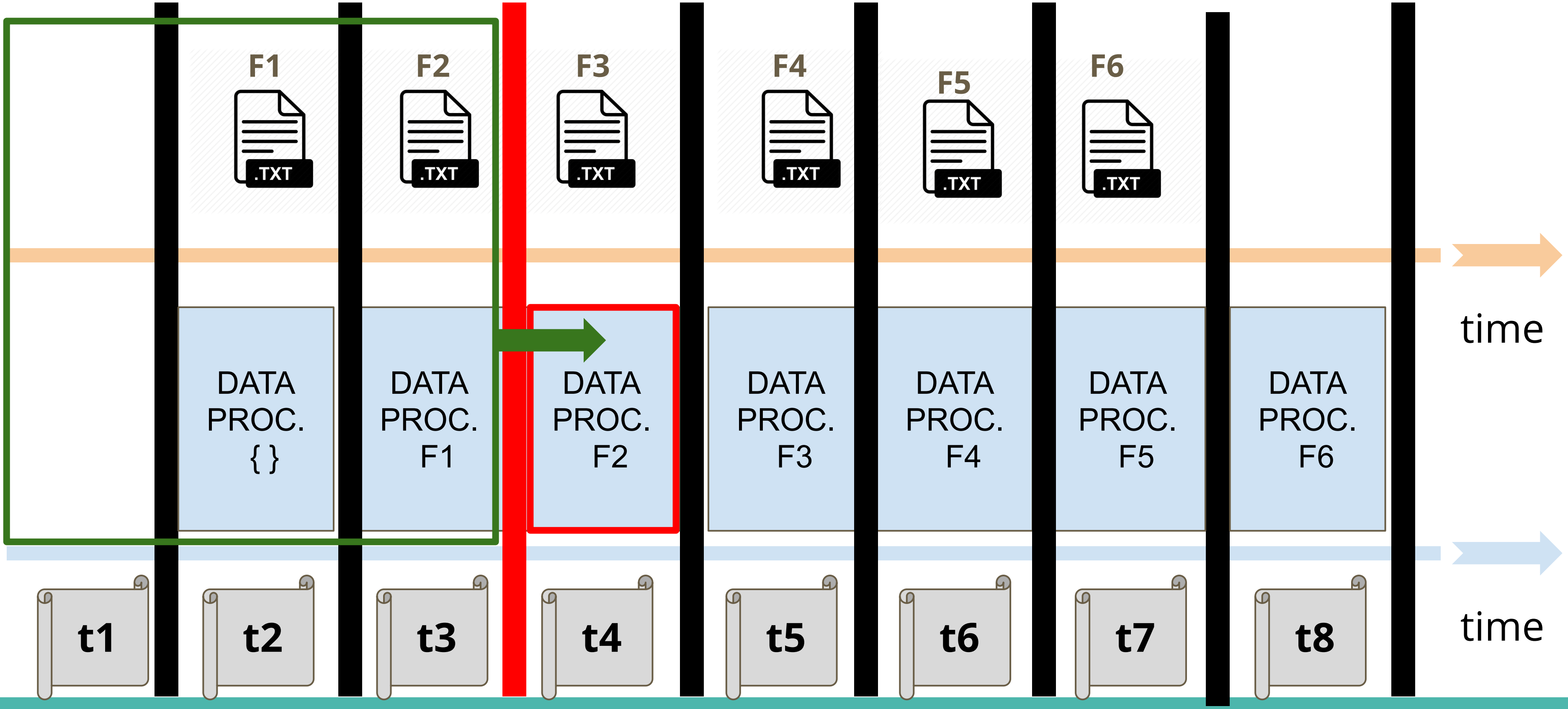
Stateless and Stateful Operations

Another example, now with a **Sliding Duration = 2** and **Window Duration = 3** we create the following windows at the following times



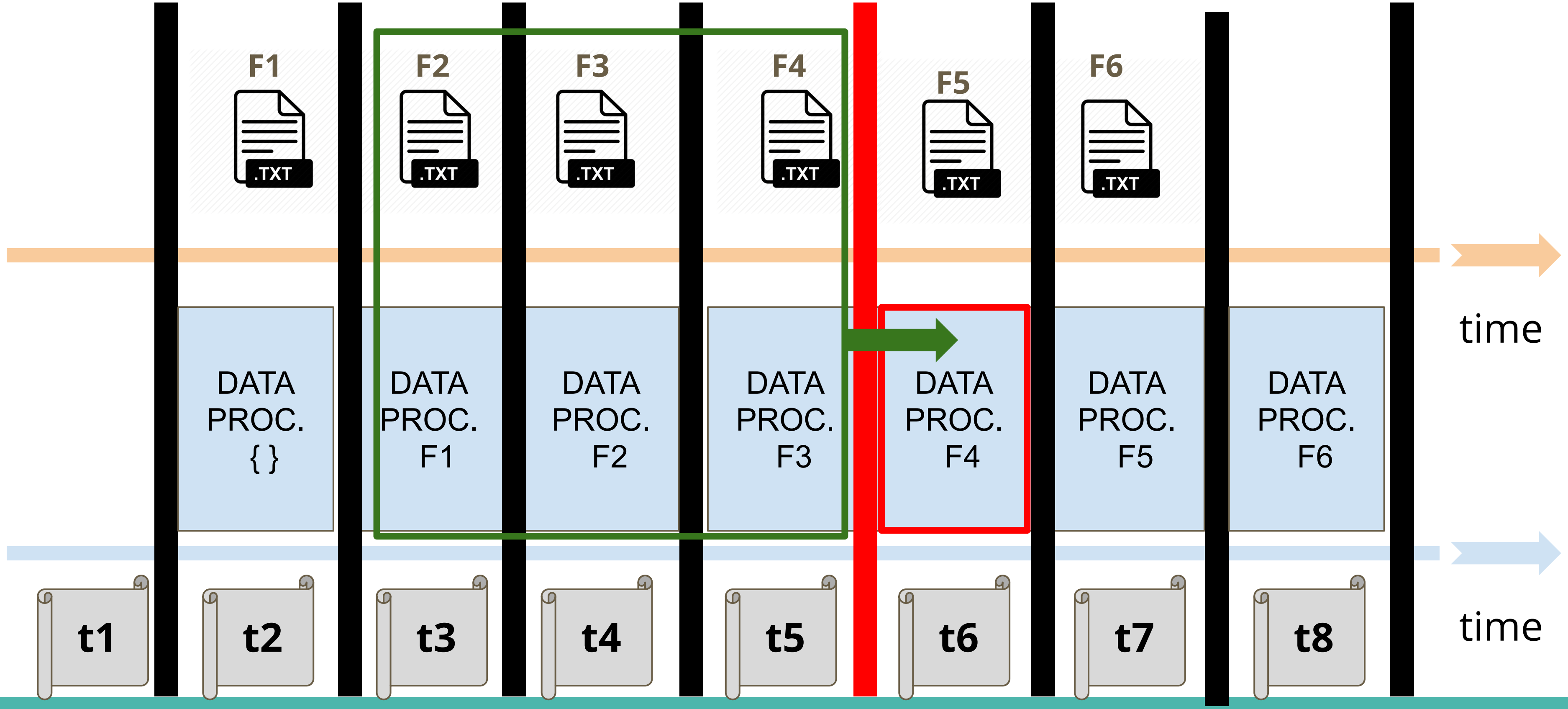
Stateless and Stateful Operations

Another example, now with a **Sliding Duration = 2** and **Window Duration = 3**
we create the following windows at the following times



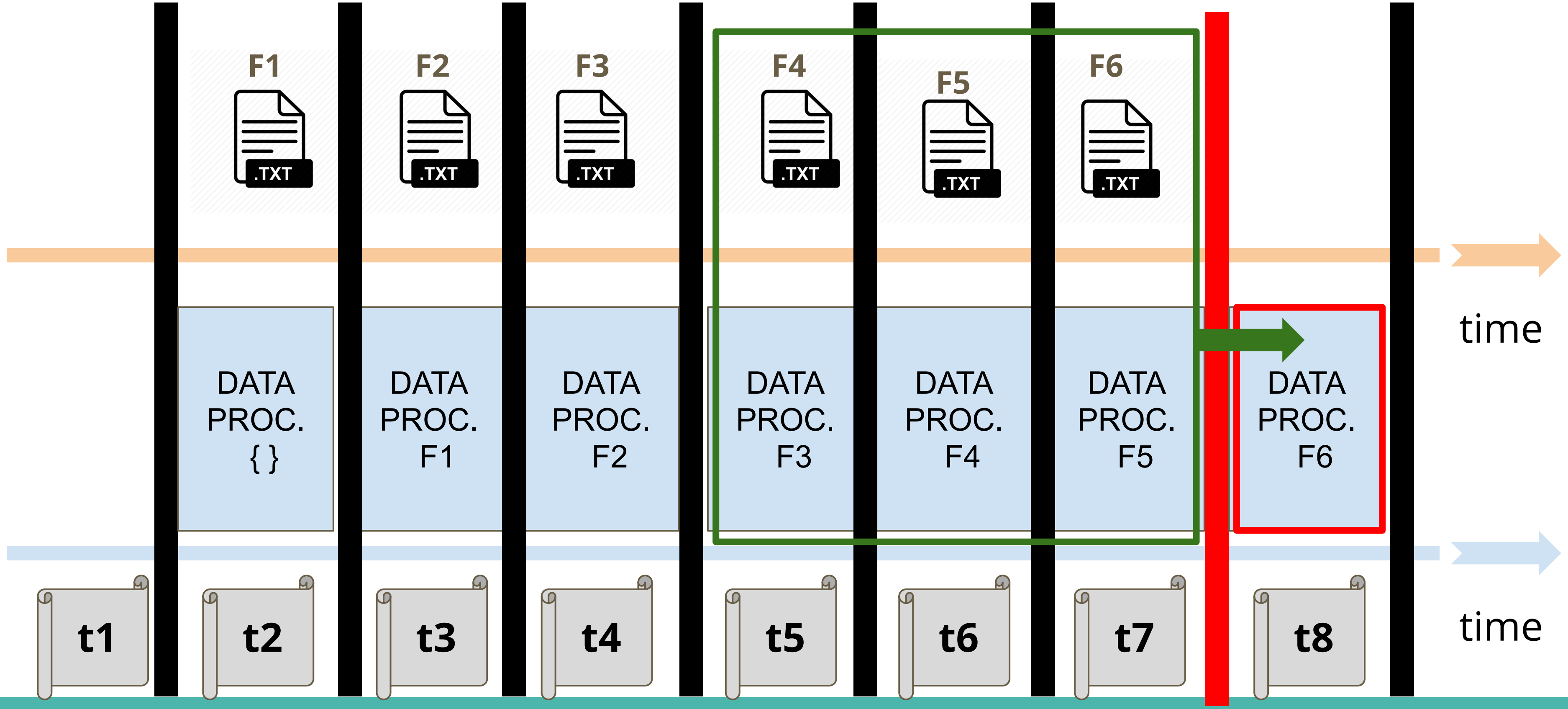
Stateless and Stateful Operations

Another example, now with a **Sliding Duration = 2** and **Window Duration = 3** we create the following windows at the following times



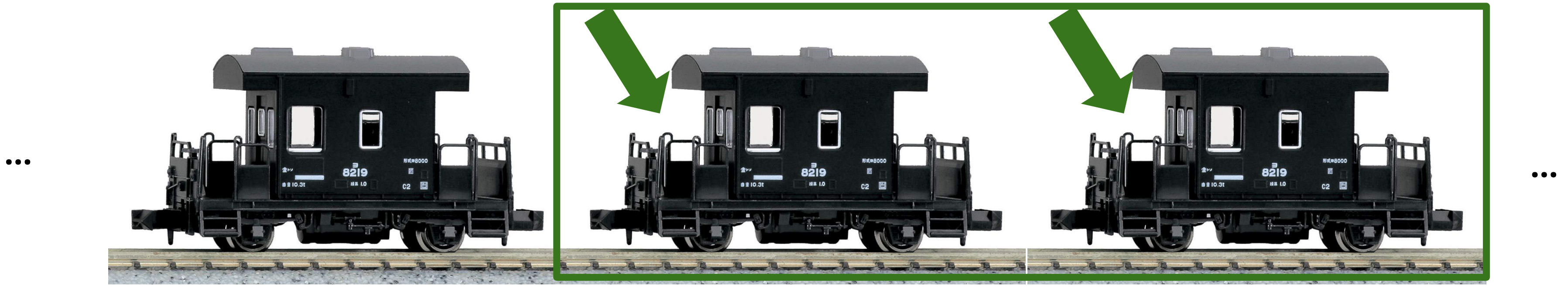
Stateless and Stateful Operations

Another example, now with a **Sliding Duration = 2** and **Window Duration = 3** we create the following windows at the following times



Stateless and Stateful Operations

As previously said, these Window-based operations
put together the relevant group of RDDs for this time period **ti**

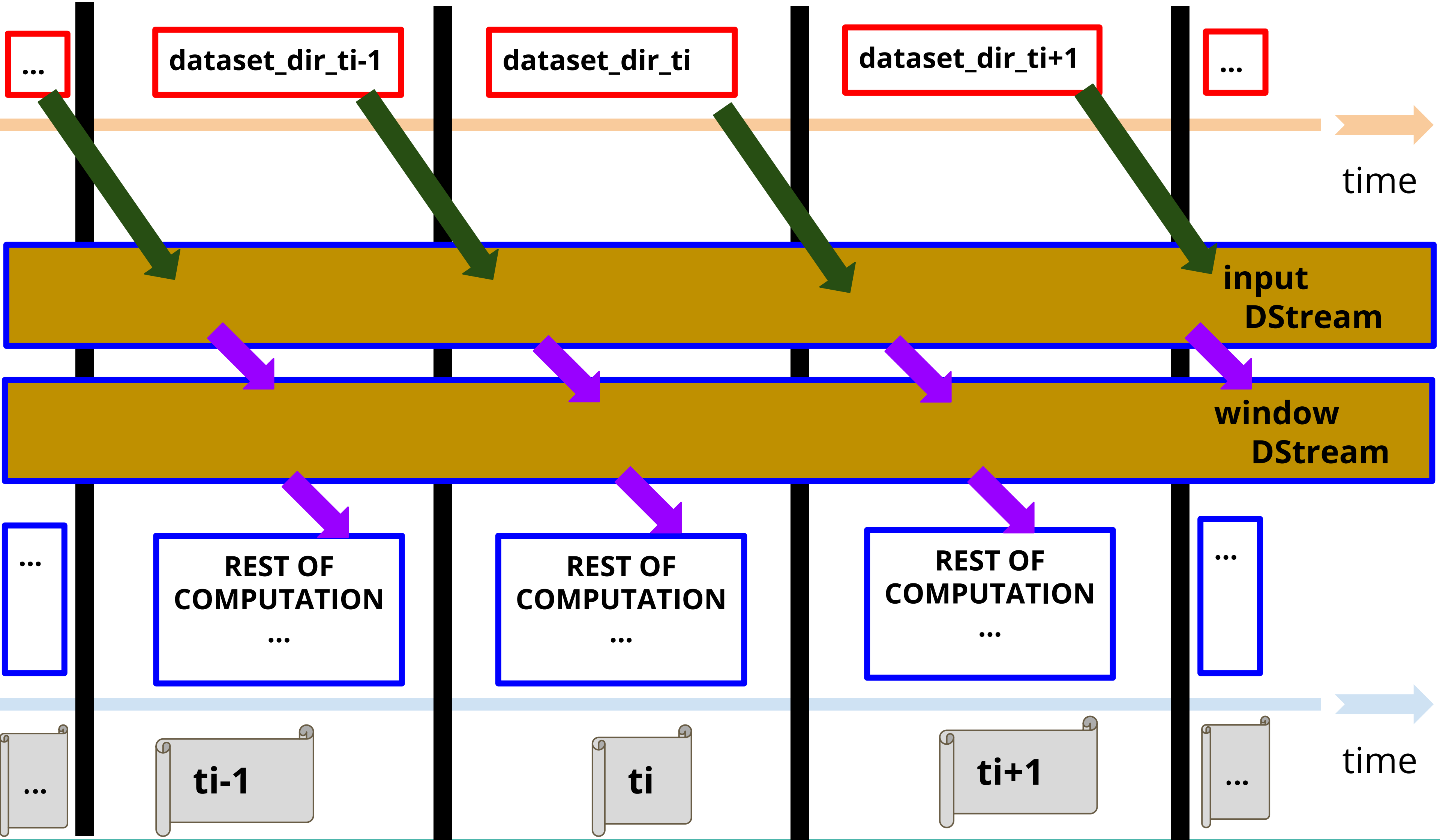


Stateless and Stateful Operations

Let’s reason with the following piece of code:

```
def my_model(ssc, monitoring_dir, time_step, window_dur, sliding_dur):

    # 1. Operation C1: textFileStream
    inputDStream = ssc.textFileStream(monitoring_dir)

    # 2. Operation T1: window
    windowDStream = inputDStream.window(window_dur * time_step,
                                        sliding_dur * time_step)

    ...
```

Stateless and Stateful Operations

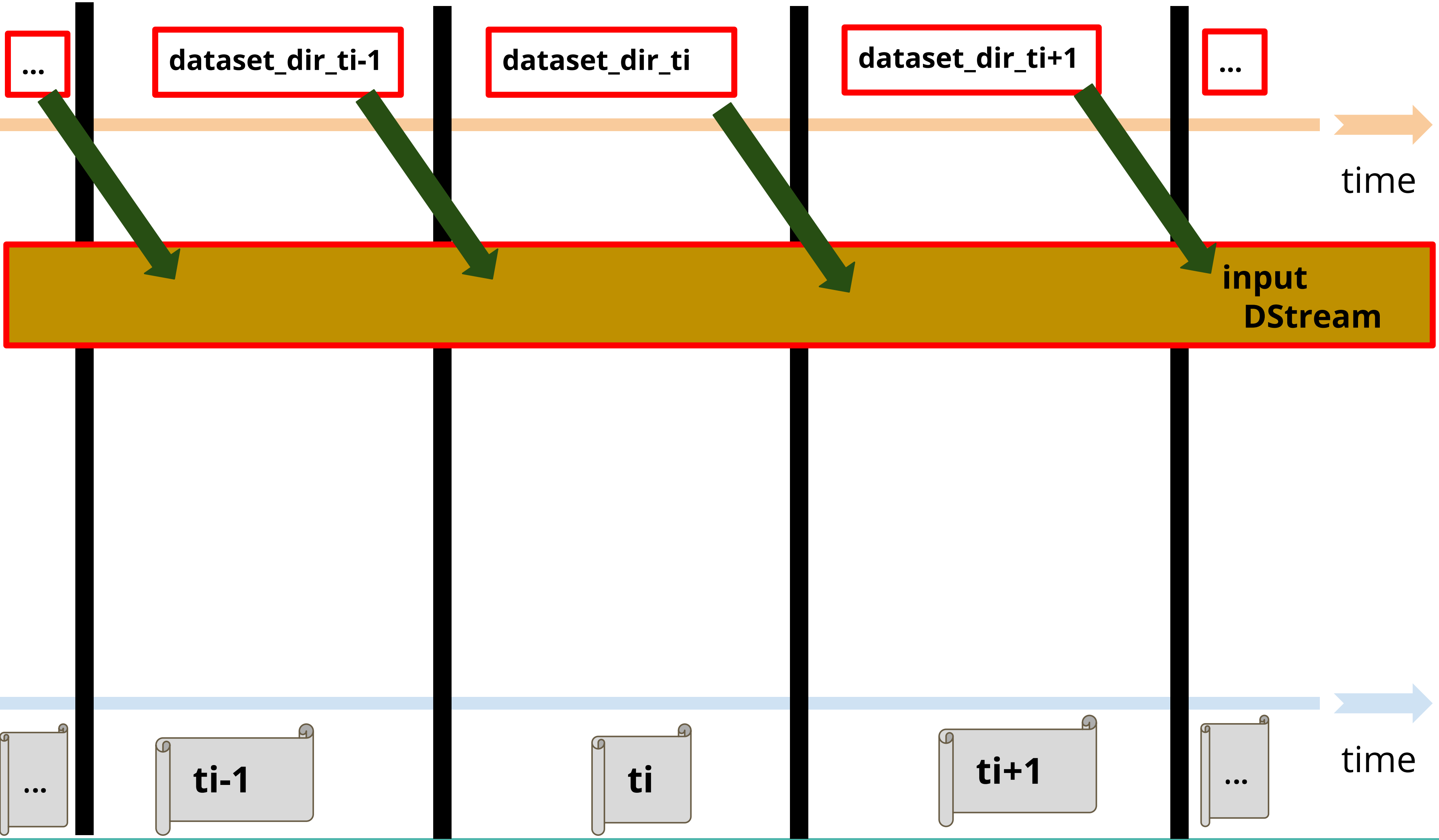


Stateless and Stateful Operations

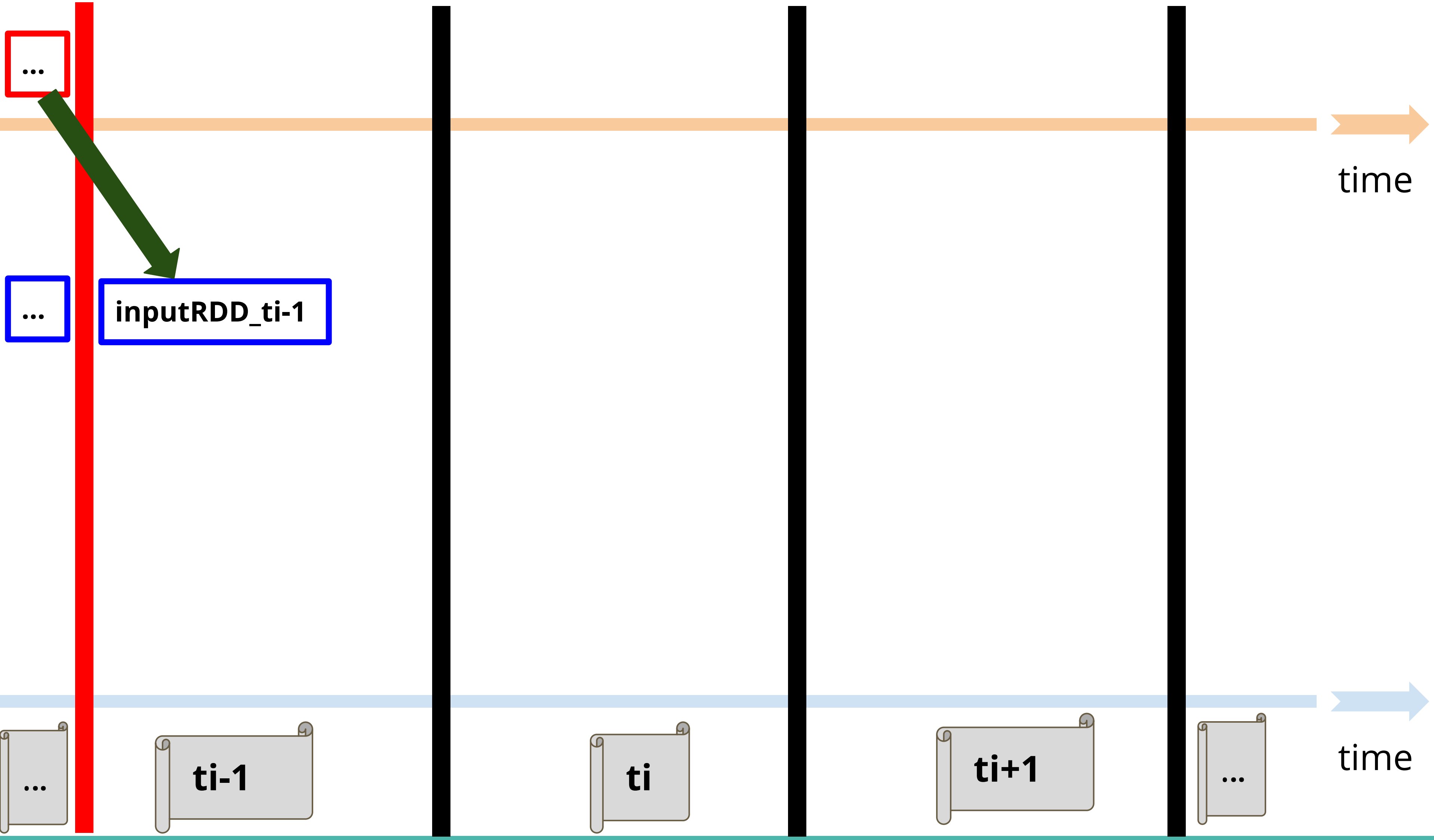
Let’s see the computation over time w.r.t. **textFileStream**

```
def my_model(ssc, monitoring_dir, time_step, window_dur, sliding_dur):

    # 1. Operation C1: textFileStream
    inputDStream = ssc.textFileStream(monitoring_dir)

    # 2. Operation T1: window
    windowDStream = inputDStream.window(window_dur * time_step,
                                        sliding_dur * time_step)

    ...
```

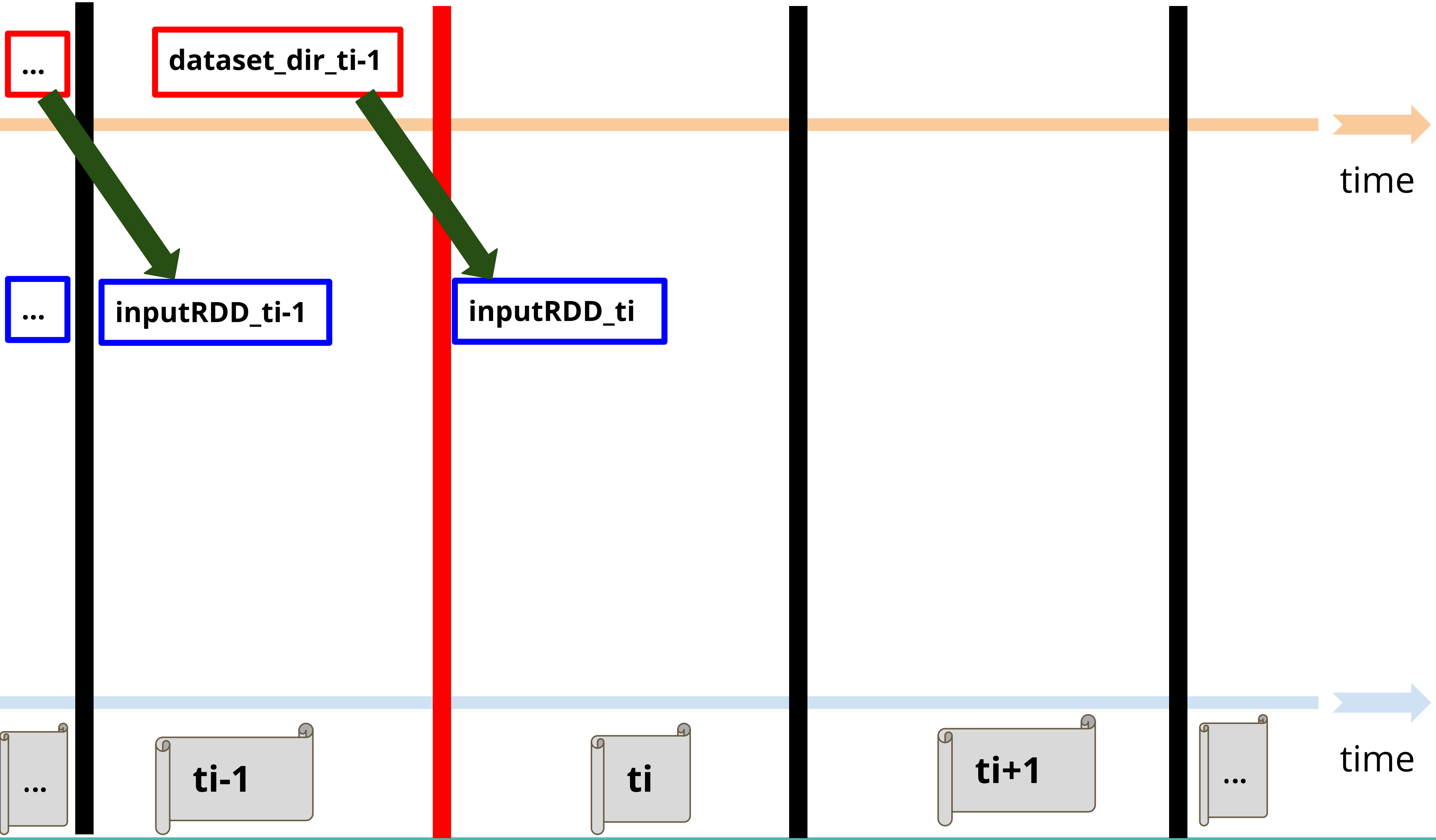
Stateless and Stateful Operations



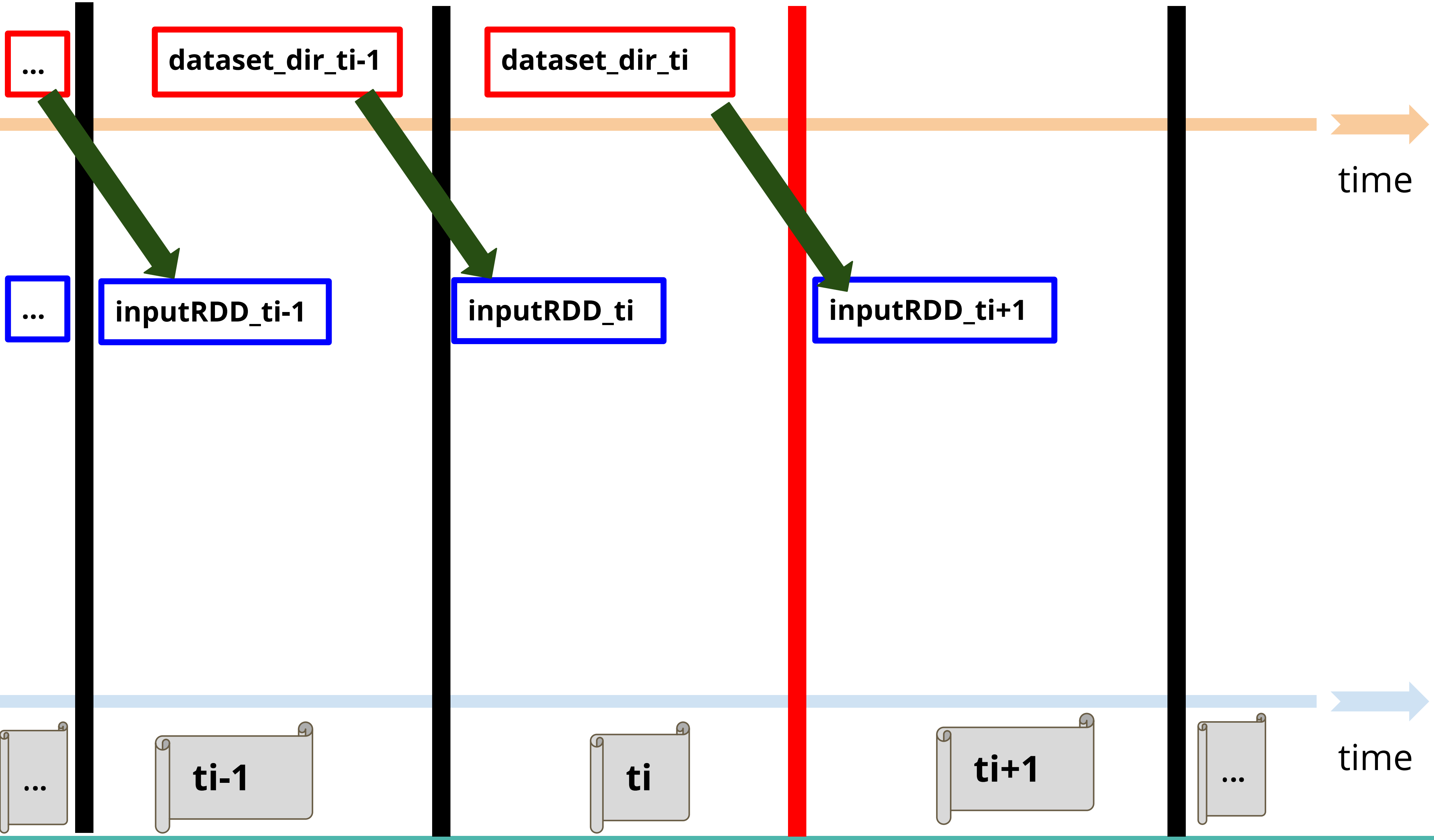
Stateless and Stateful Operations



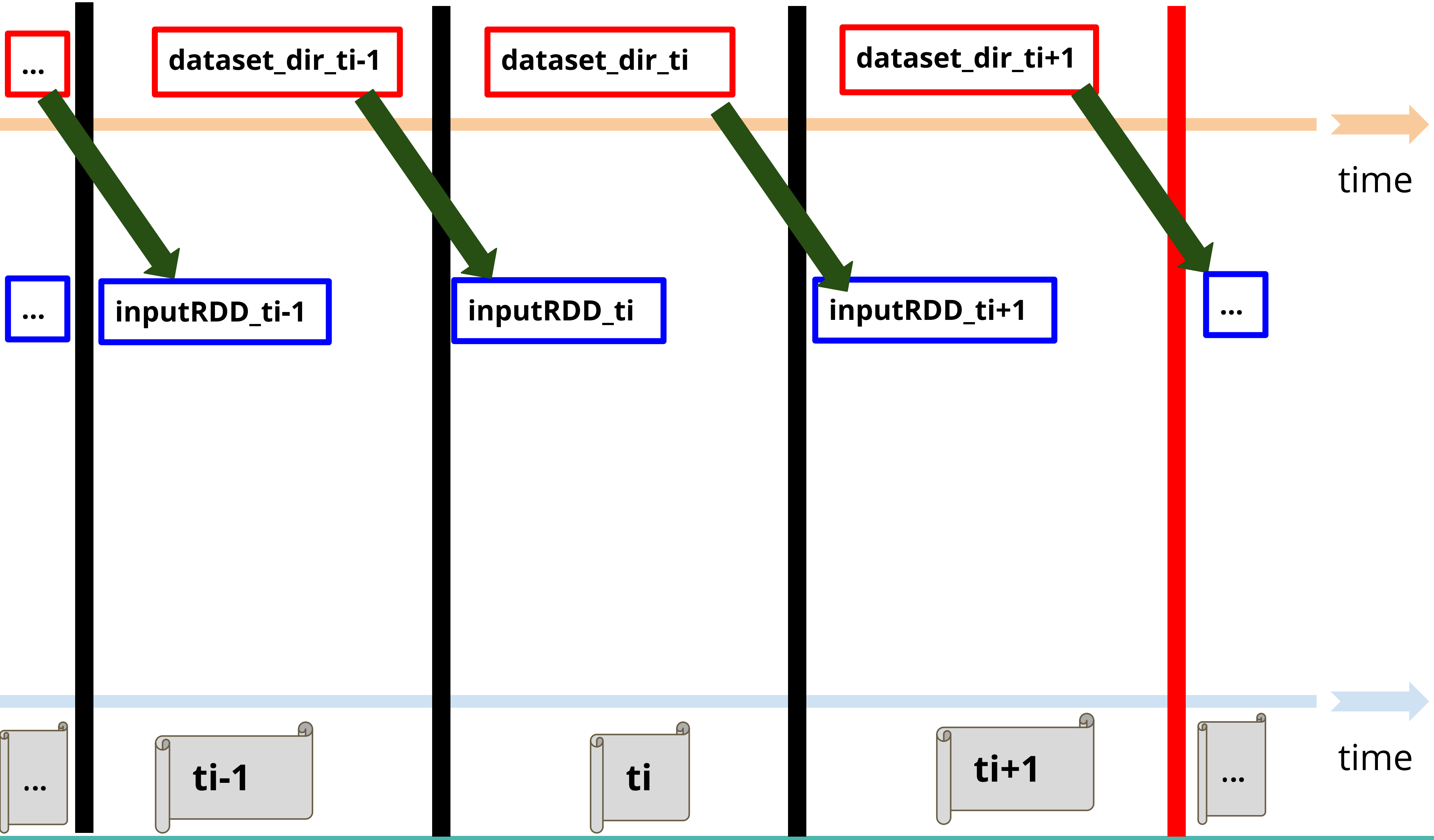
Stateless and Stateful Operations



Stateless and Stateful Operations



Stateless and Stateful Operations



Stateless and Stateful Operations

Let’s see the computation over time w.r.t. **window**

```
def my_model(ssc, monitoring_dir, time_step, window_dur, sliding_dur):

    # 1. Operation C1: textFileStream
    inputDStream = ssc.textFileStream(monitoring_dir)

    # 2. Operation T1: window
    windowDStream = inputDStream.window(window_dur * time_step,
                                        sliding_dur * time_step)

    ...
```

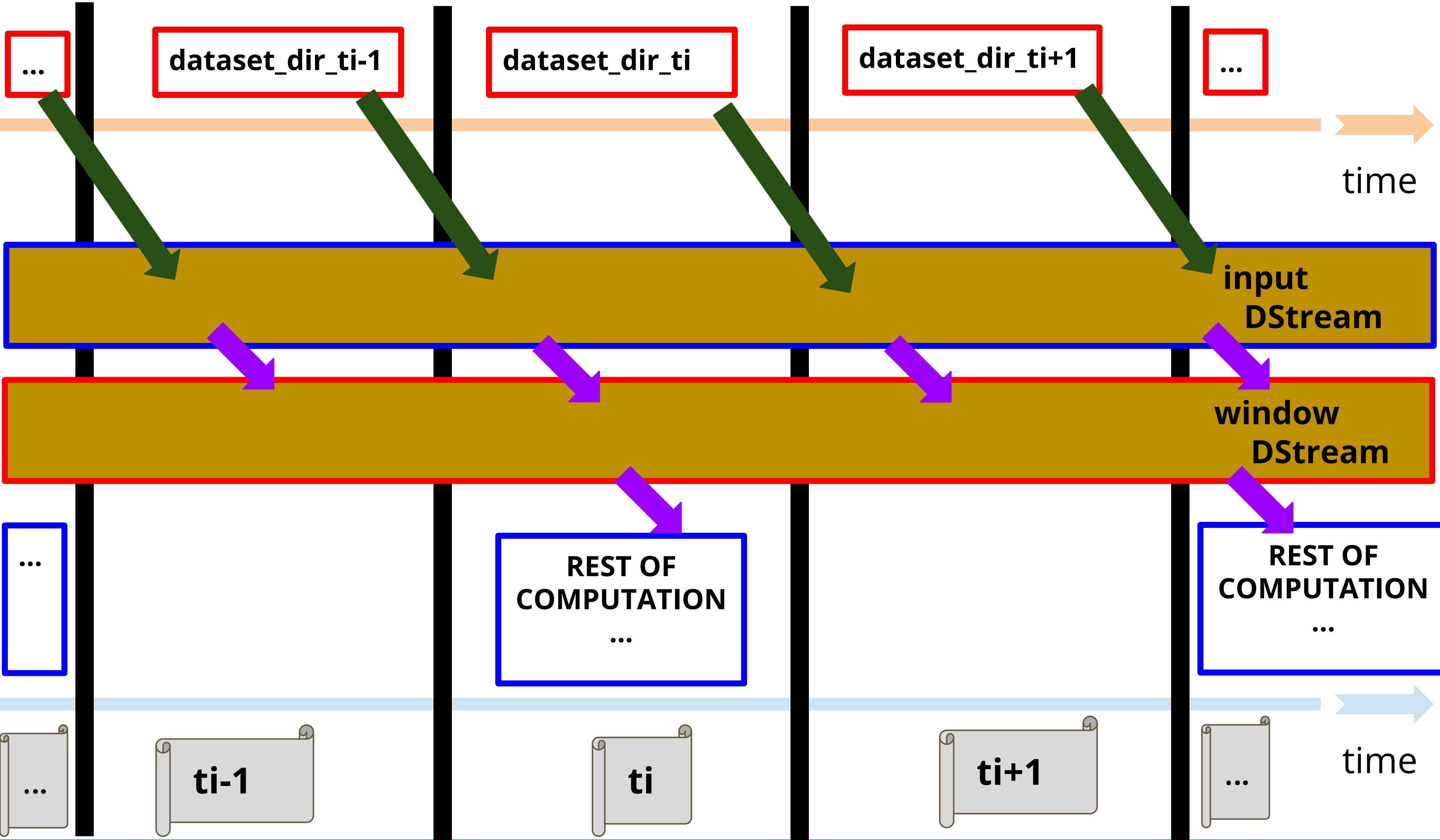
Stateless and Stateful Operations

Let's see the computation over time w.r.t. **window**

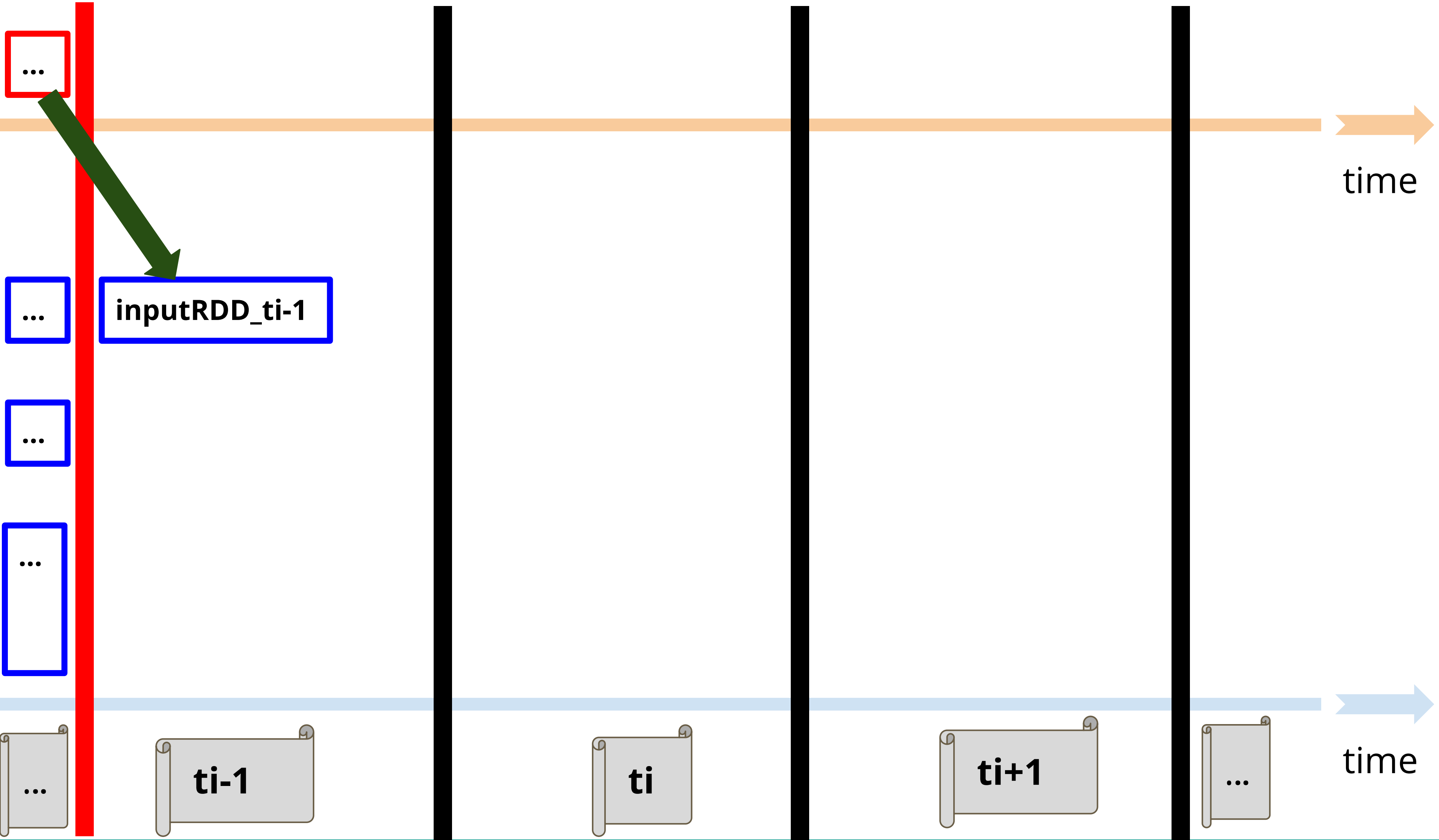
Example: **sliding_dur = 2**; **window_duration = 2**

```
def my_model(ssc, monitoring_dir, time_step, window_dur, sliding_dur):

    # 1. Operation C1: textFileStream
    inputDStream = ssc.textFileStream(monitoring_dir)

    # 2. Operation T1: window
    windowDStream = inputDStream.window(2 * time_step,
                                        2 * time_step)

    ...
```

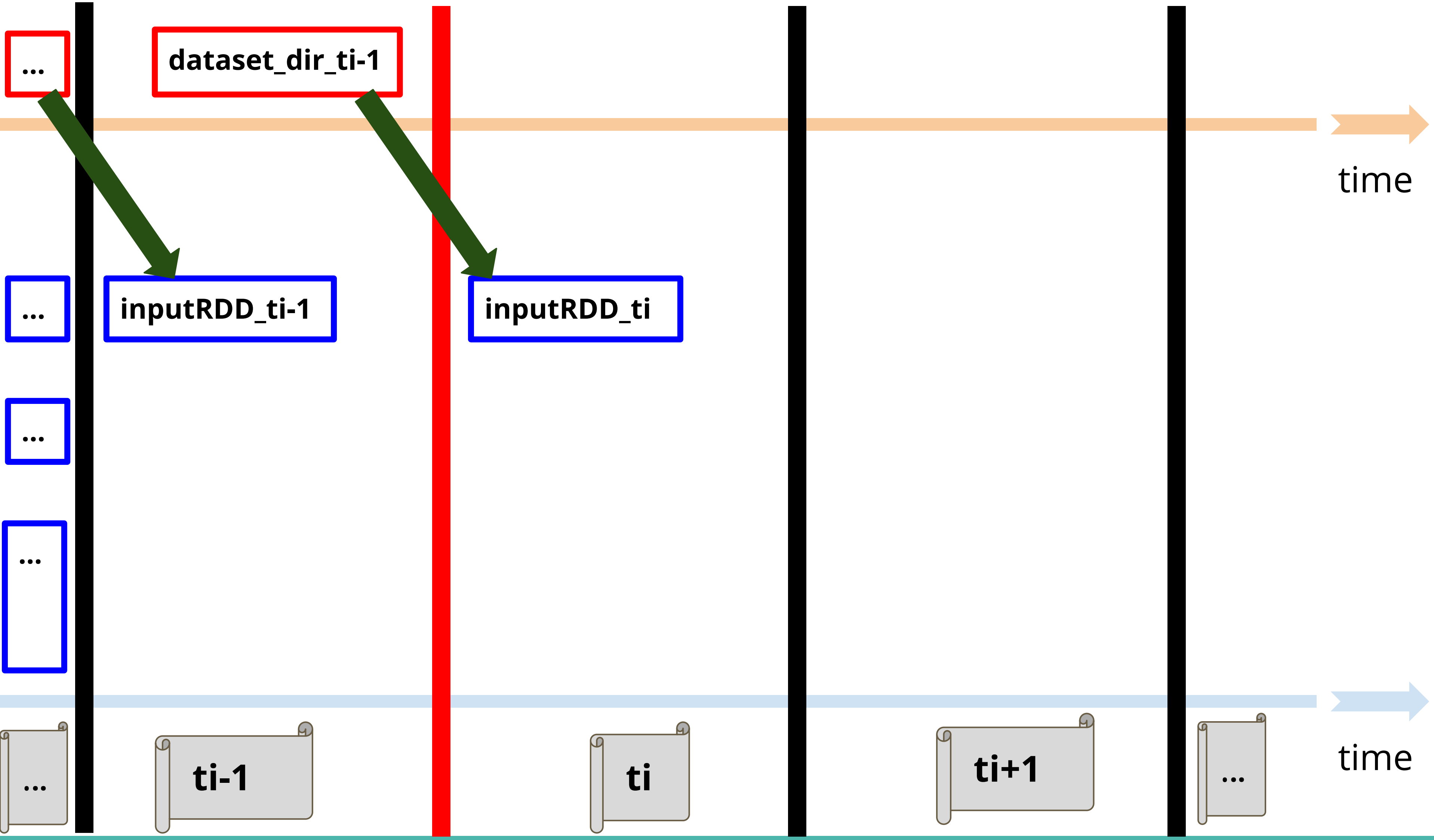
Stateless and Stateful Operations



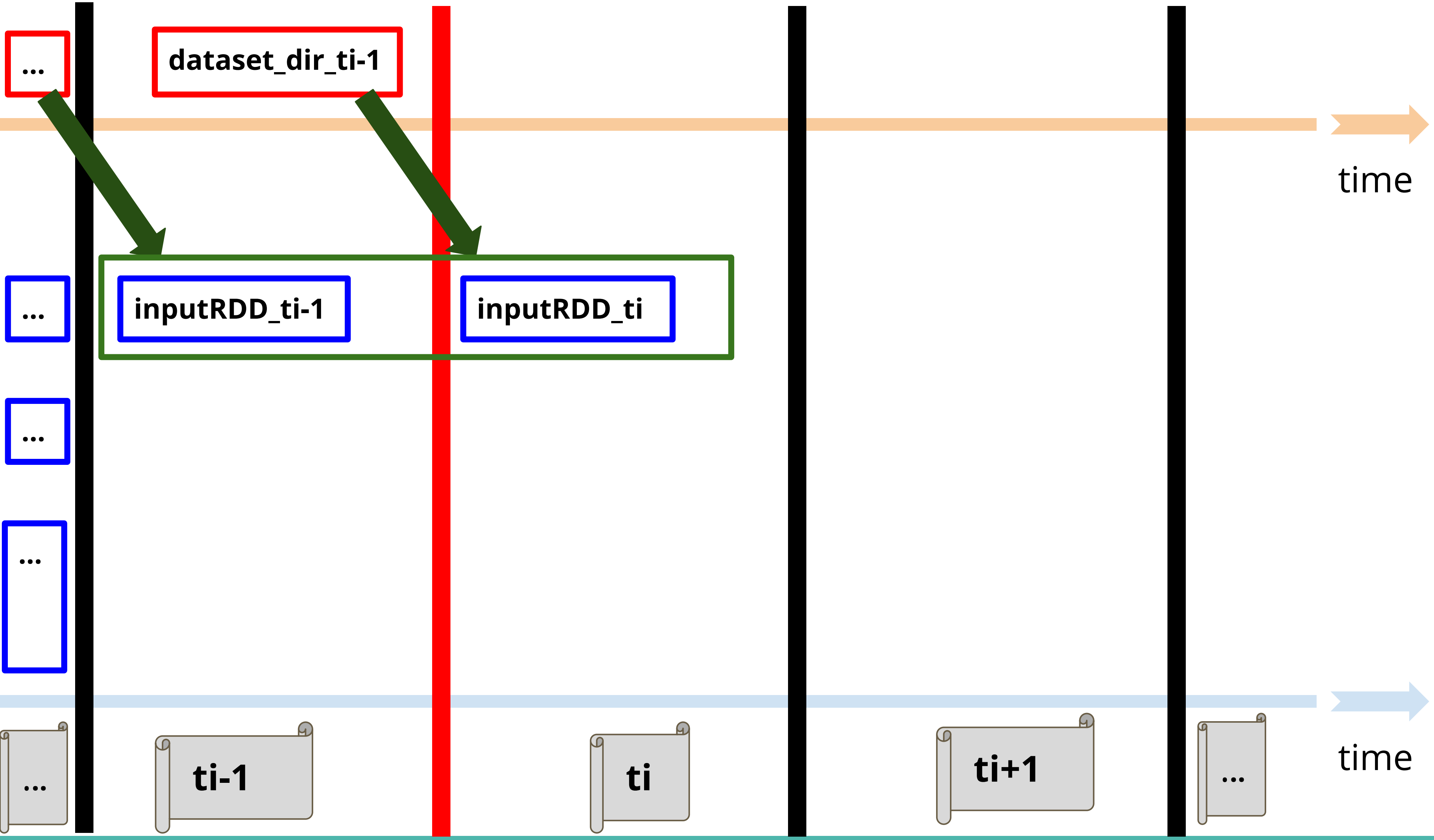
Stateless and Stateful Operations



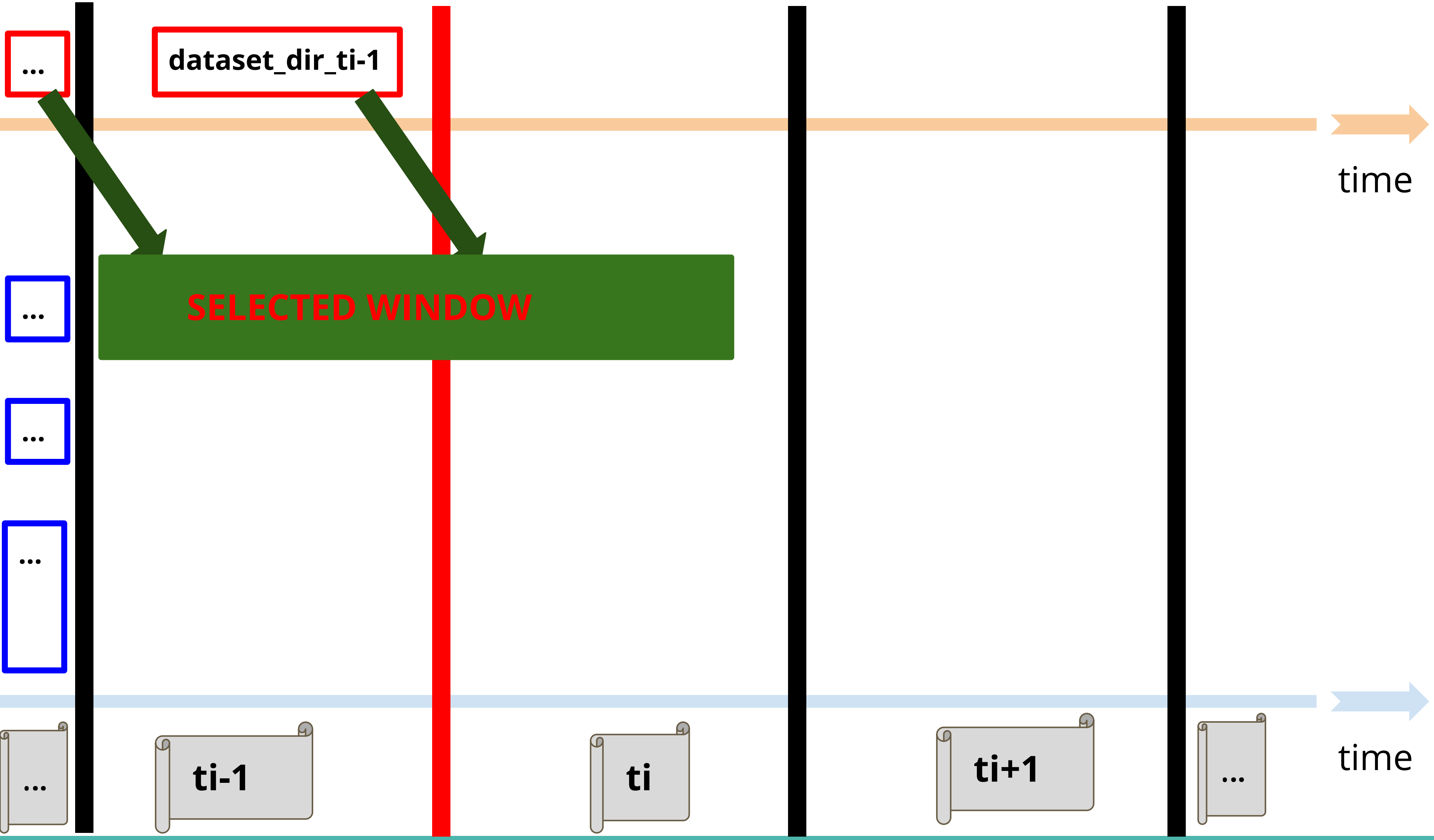
Stateless and Stateful Operations



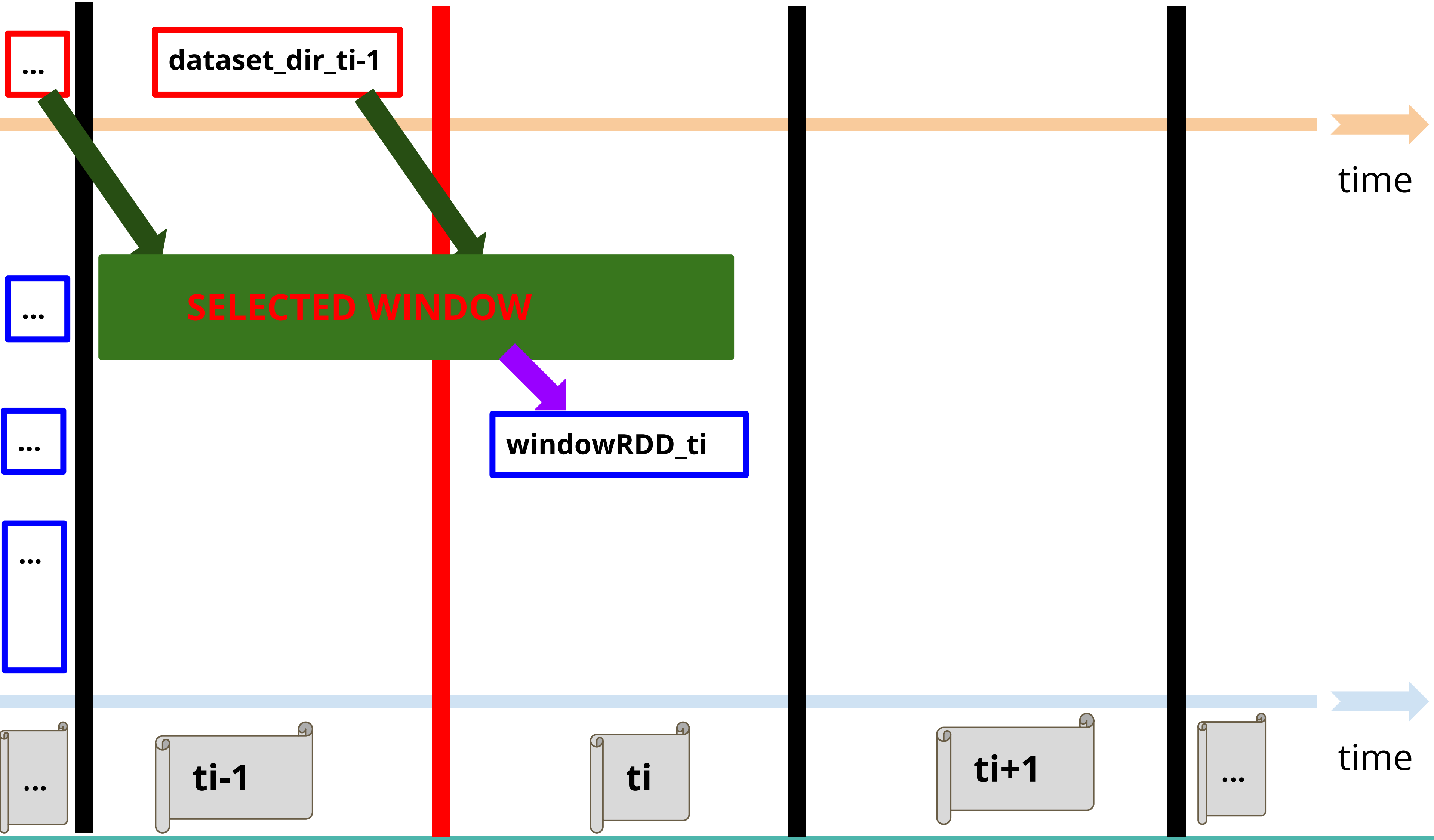
Stateless and Stateful Operations



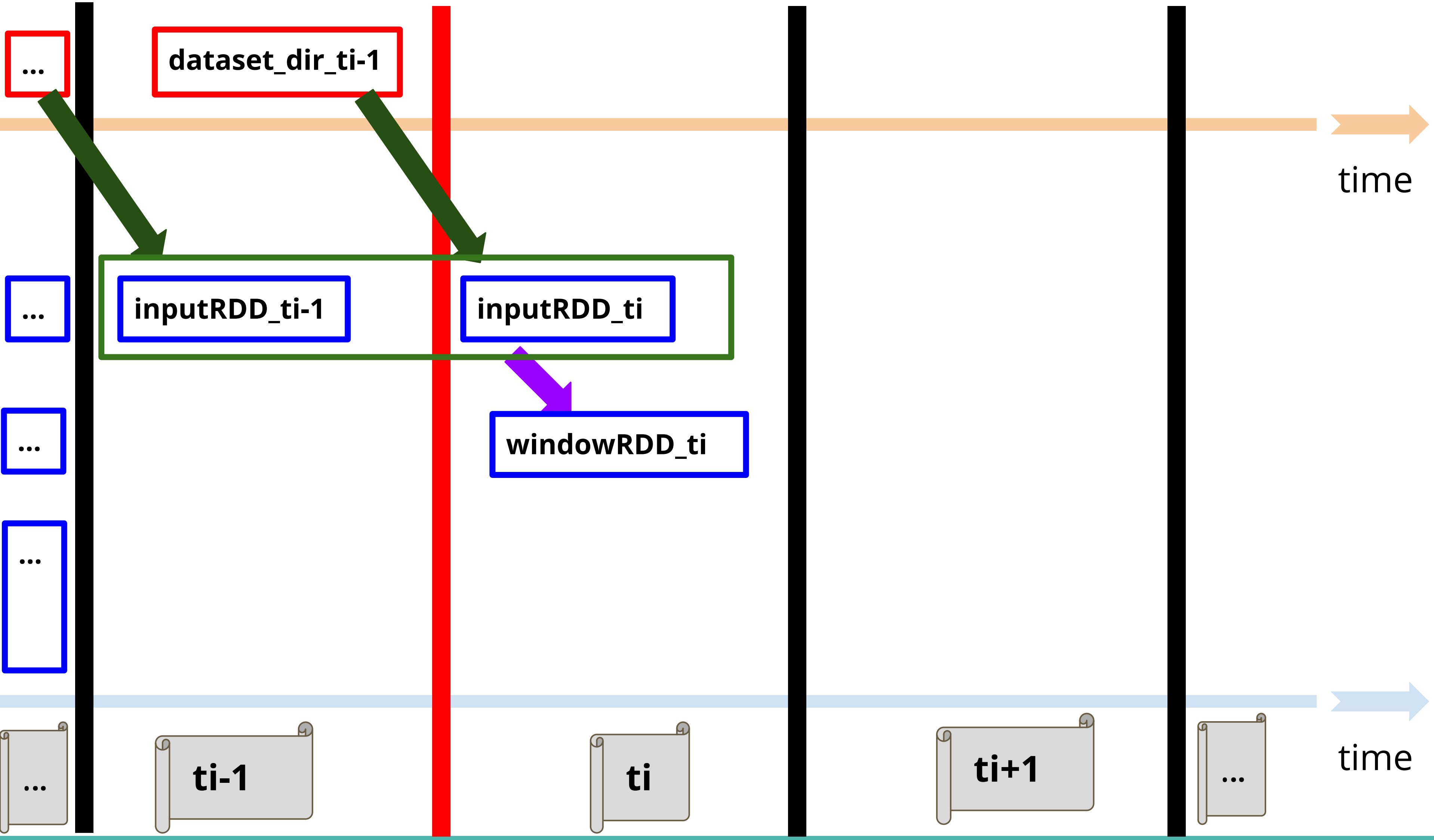
Stateless and Stateful Operations



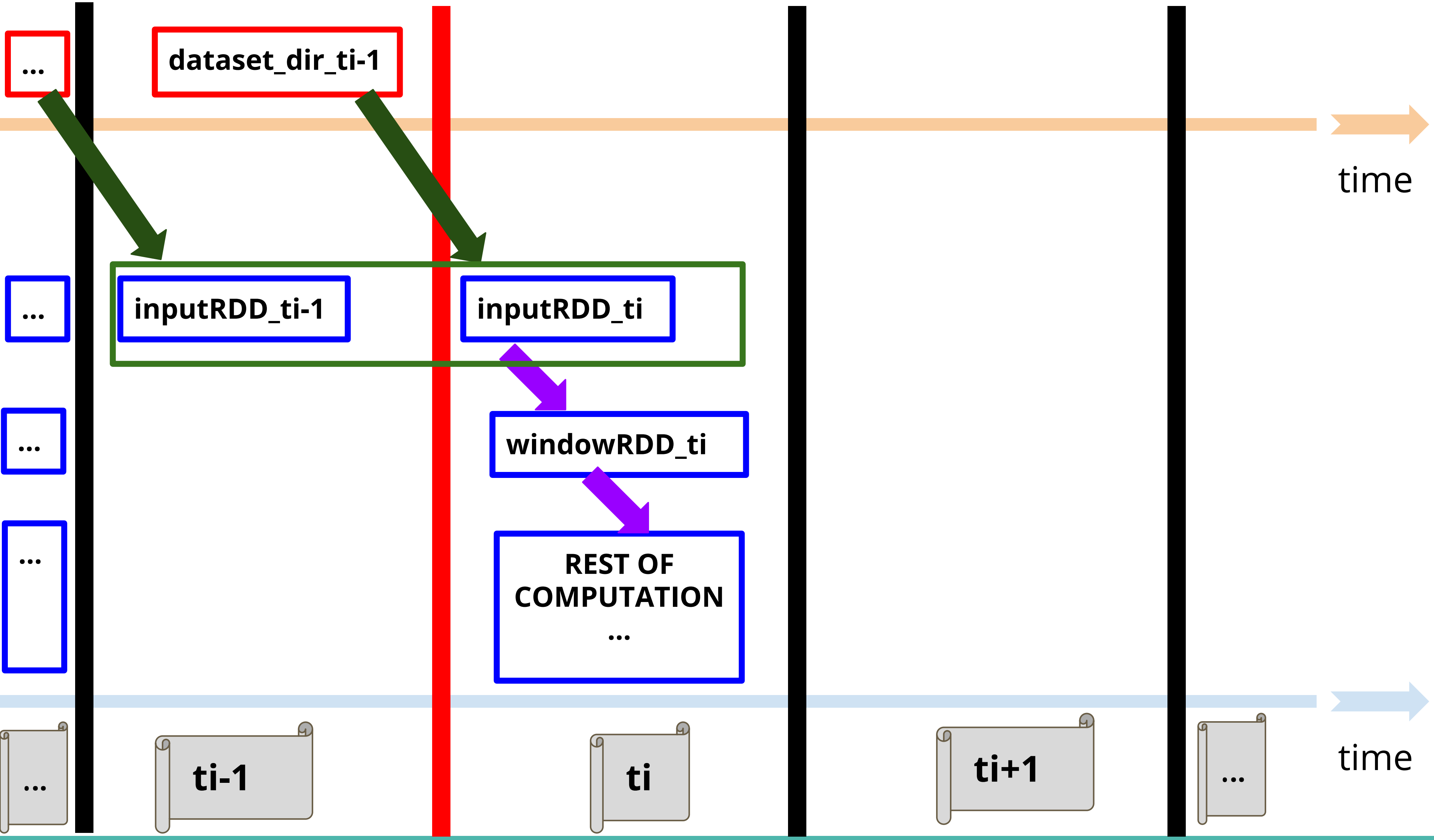
Stateless and Stateful Operations



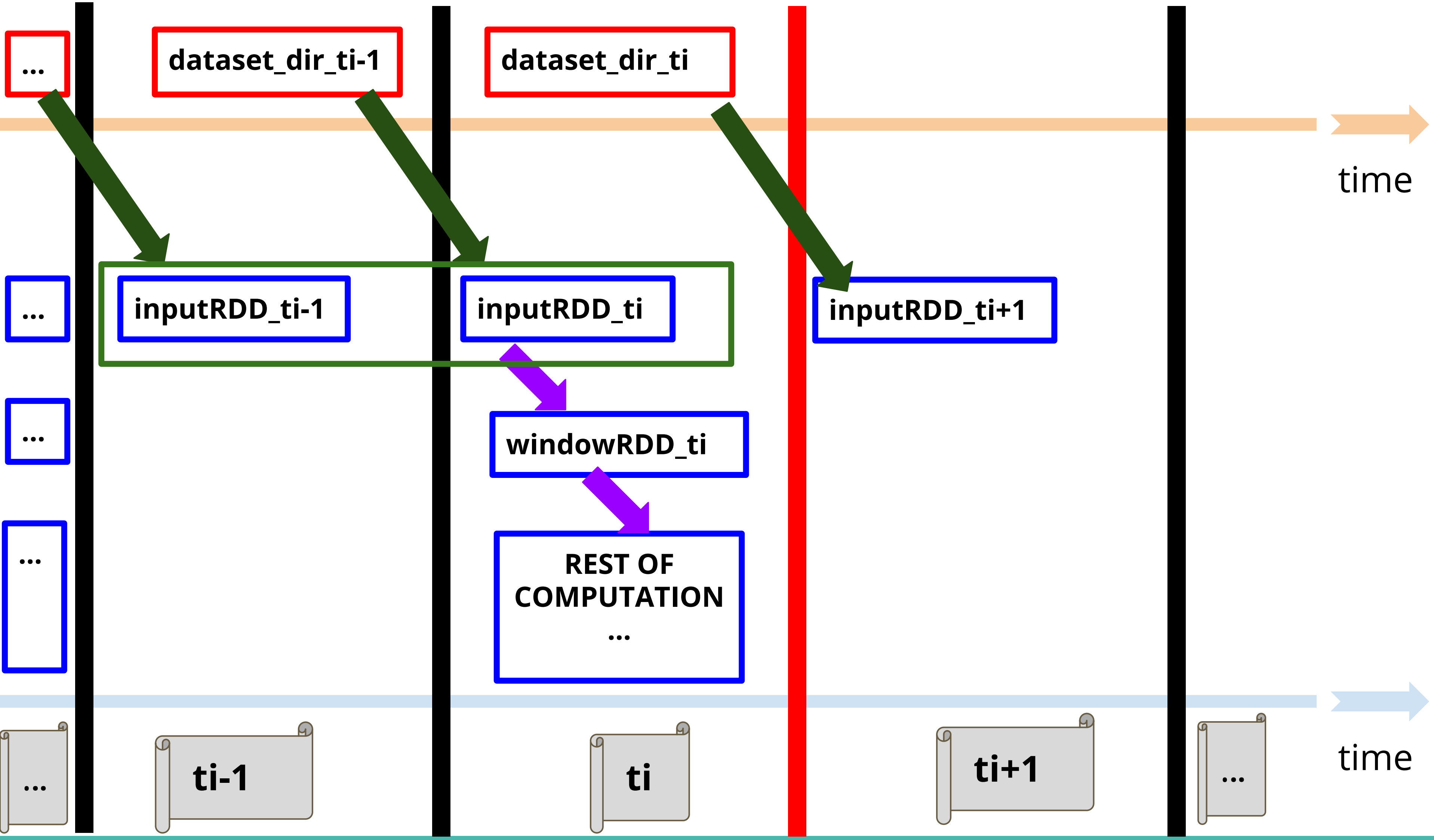
Stateless and Stateful Operations



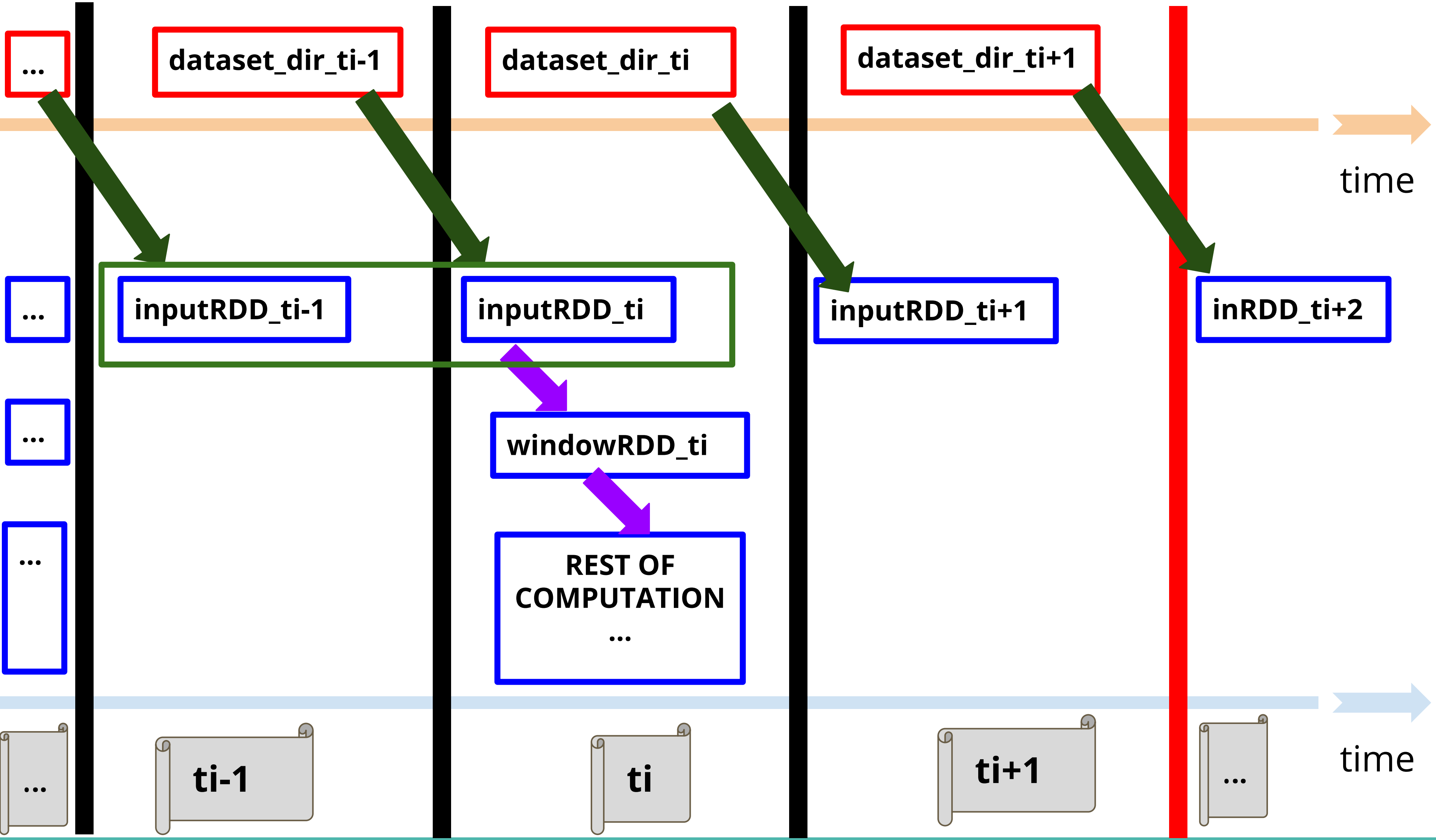
Stateless and Stateful Operations



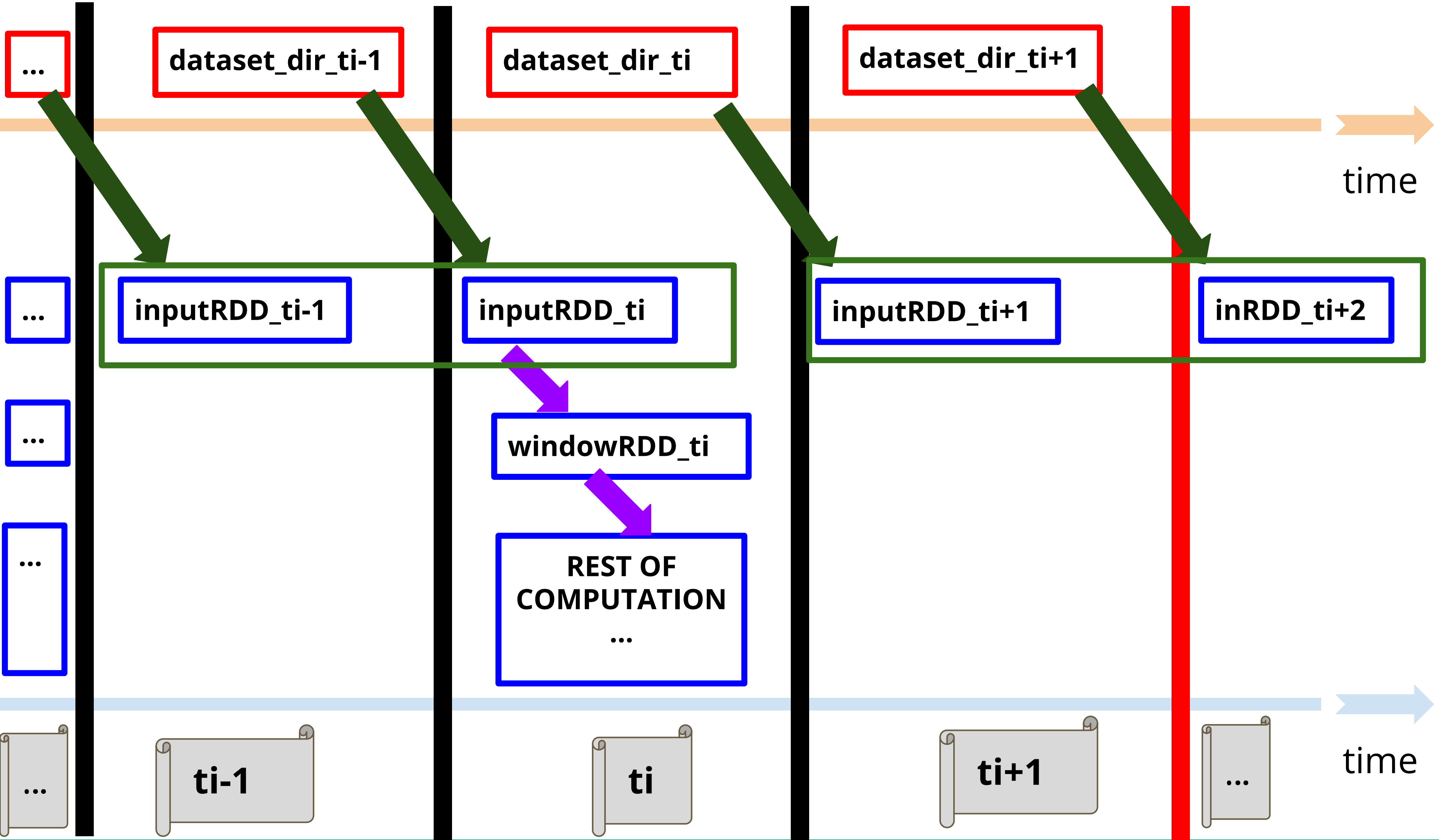
Stateless and Stateful Operations



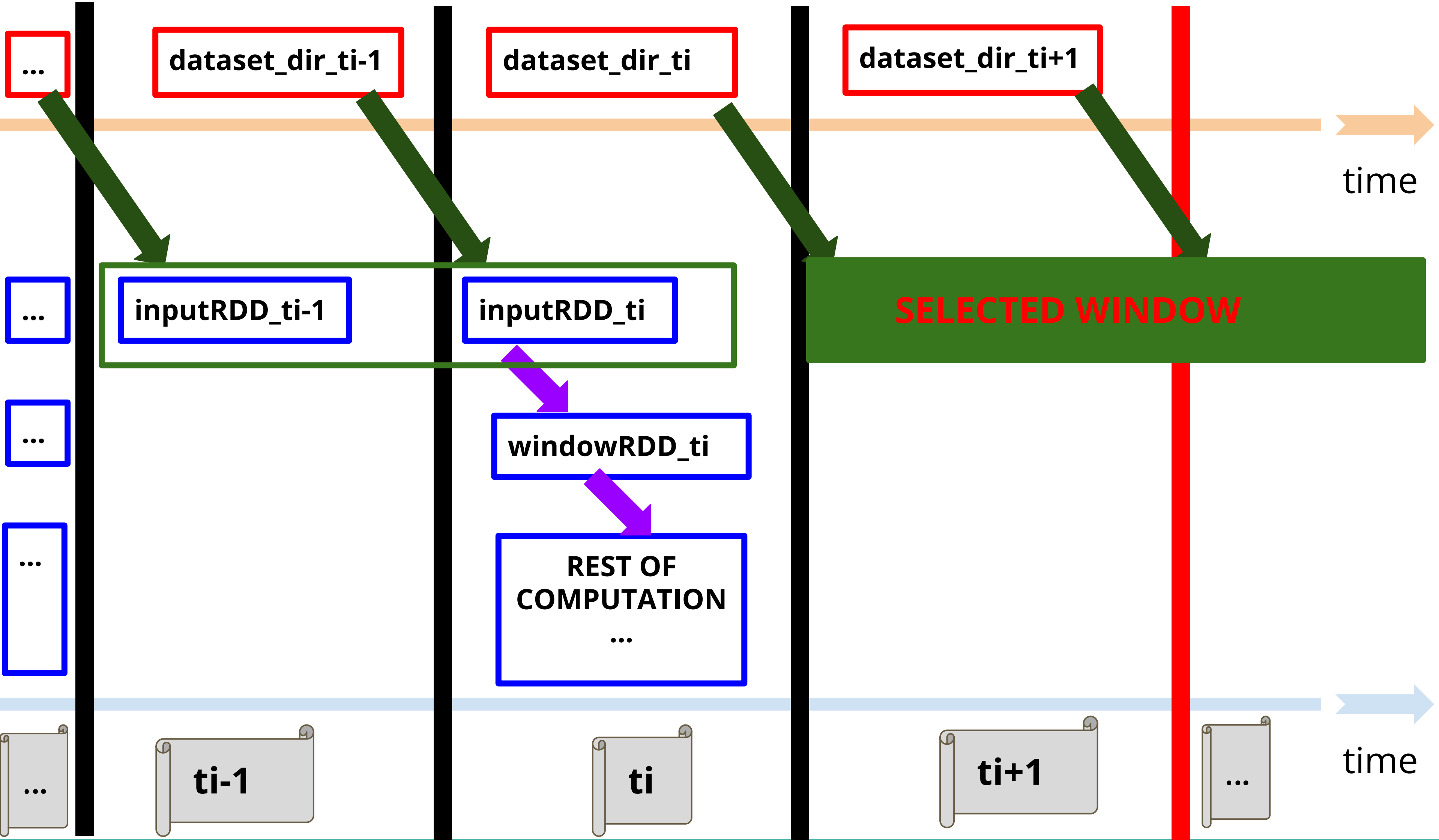
Stateless and Stateful Operations



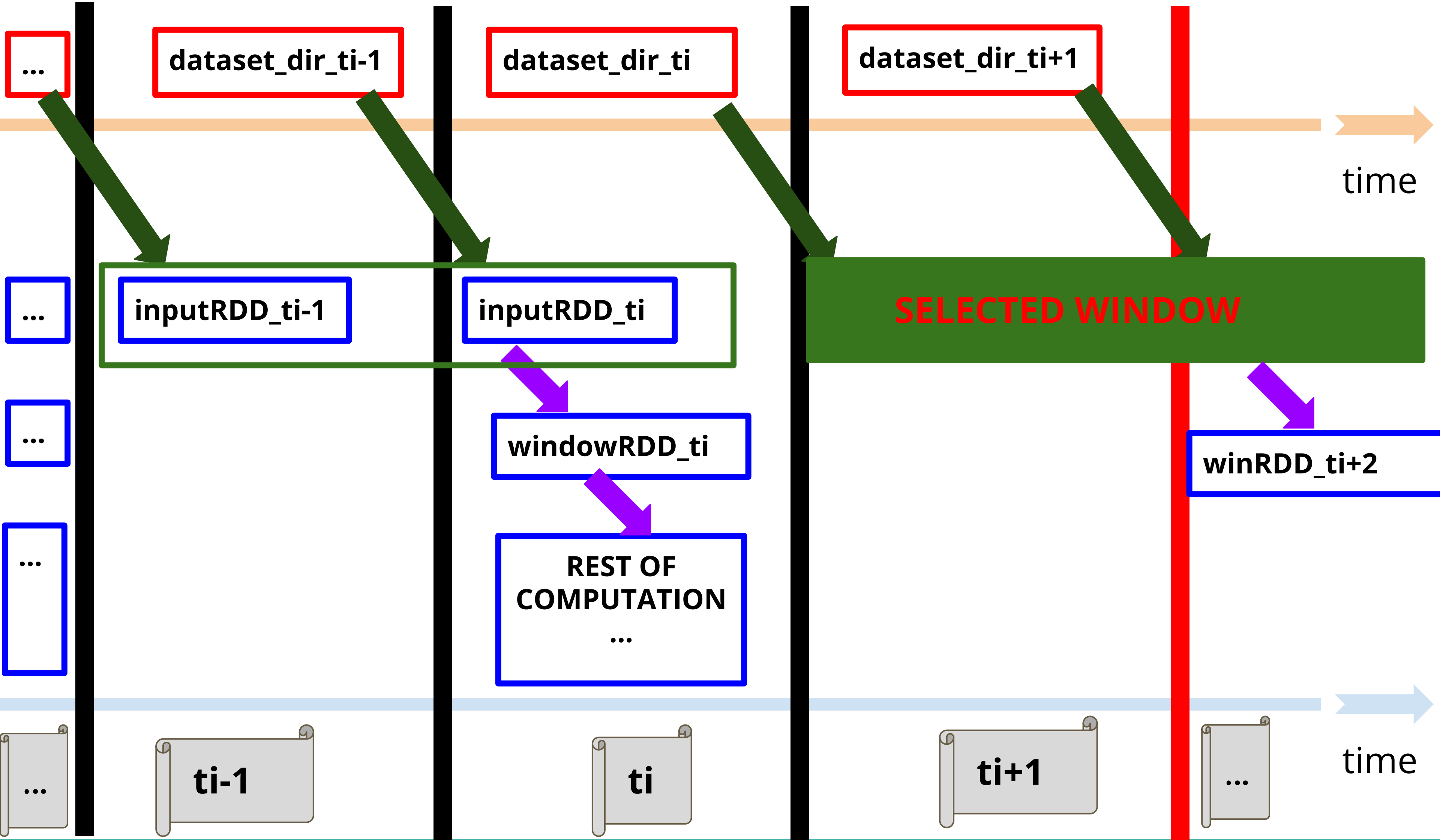
Stateless and Stateful Operations



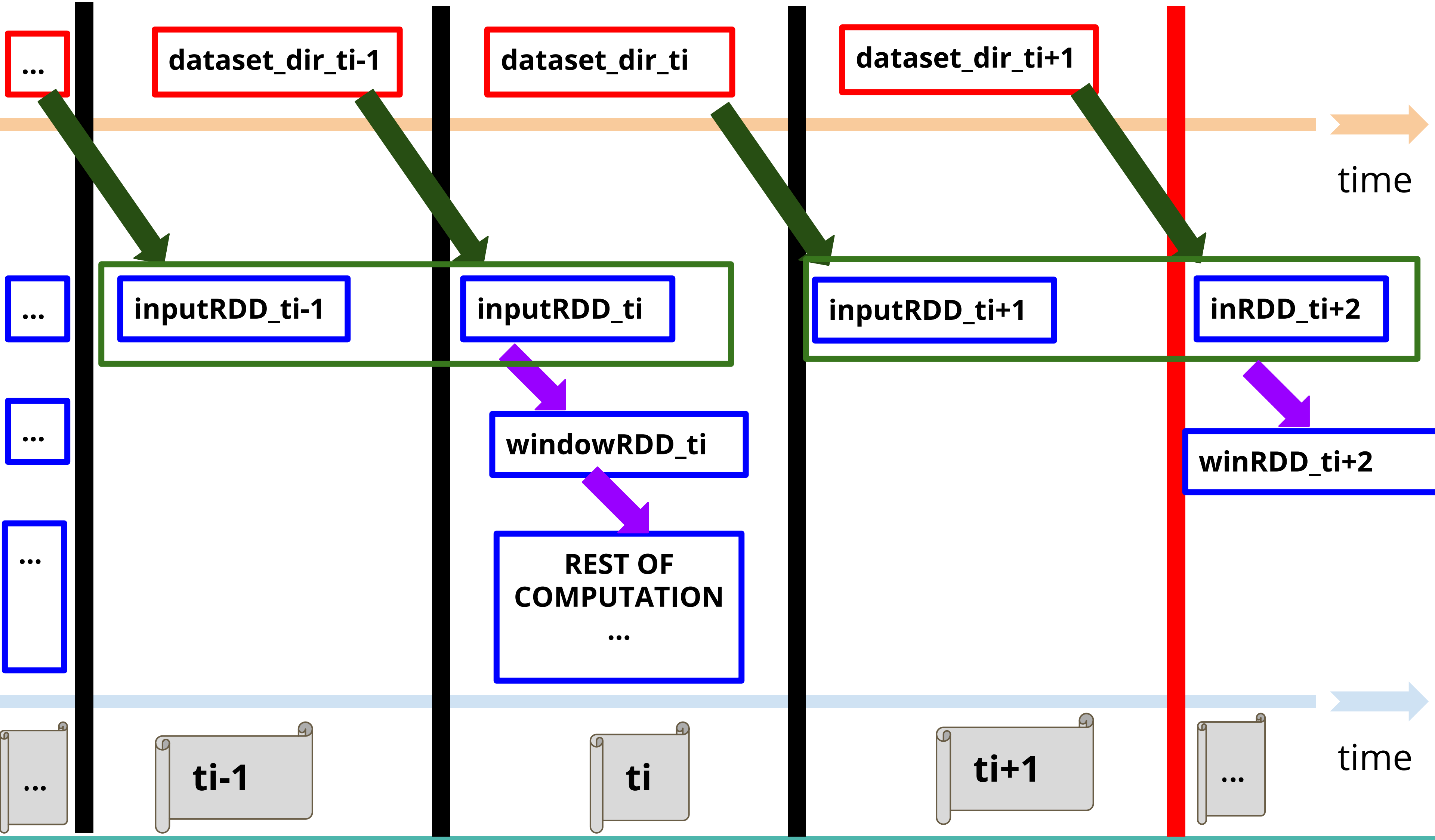
Stateless and Stateful Operations



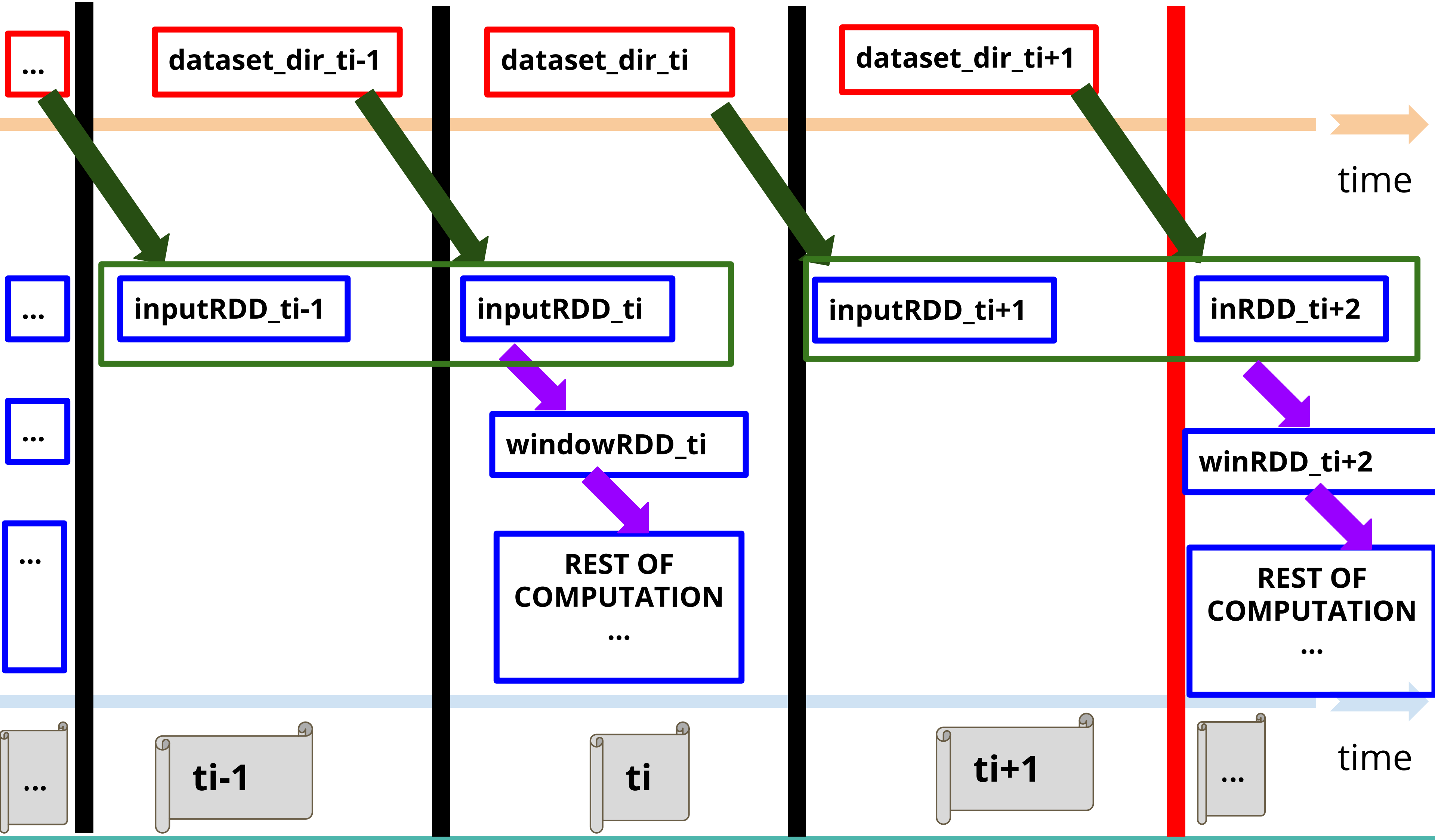
Stateless and Stateful Operations



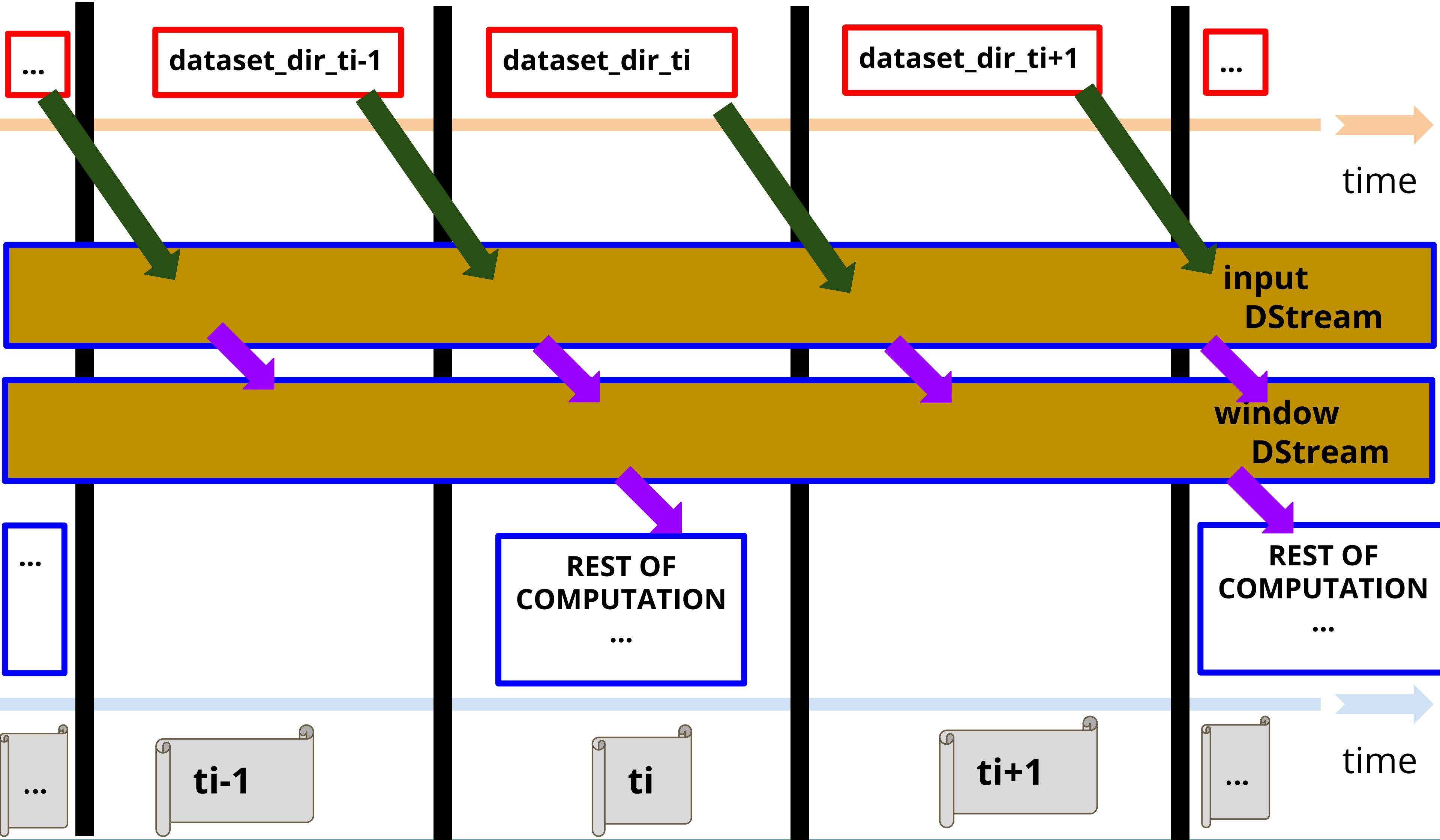
Stateless and Stateful Operations



Stateless and Stateful Operations



Stateless and Stateful Operations



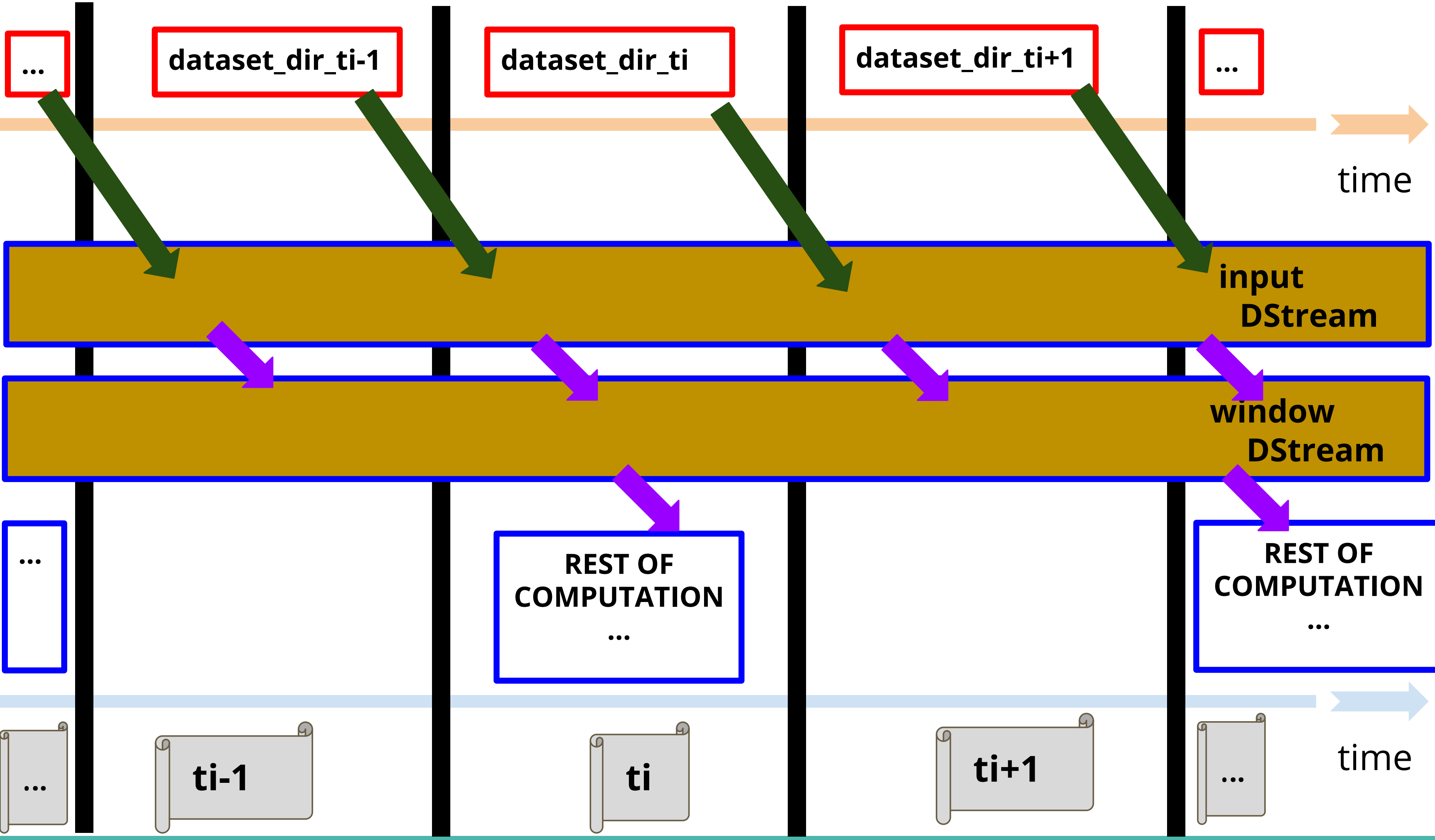
Stateless and Stateful Operations

Let’s see the computation over time w.r.t. **window**

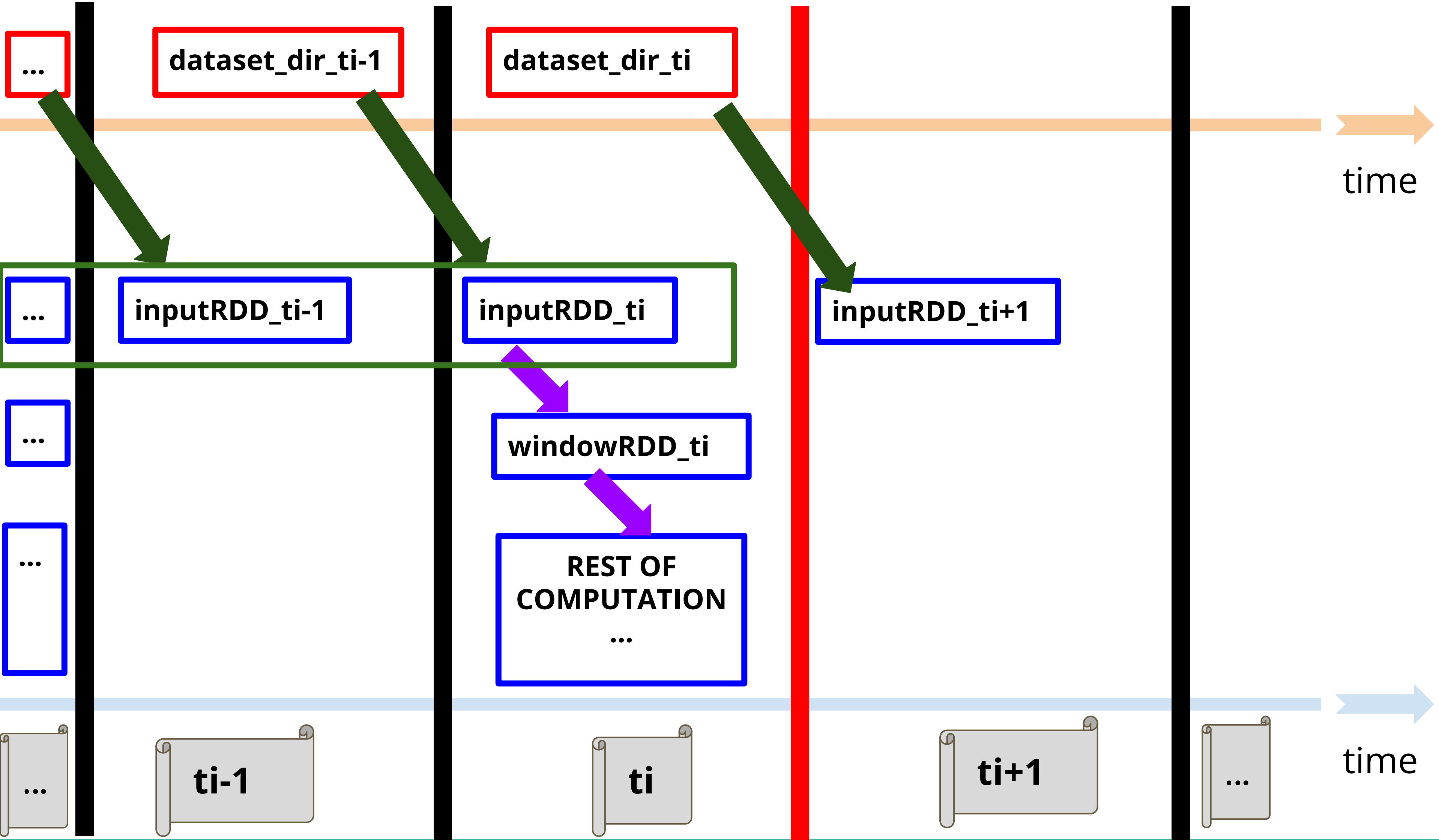
Another example: **sliding_dur = 2**; **window_duration = 3**

```
def my_model(ssc, monitoring_dir, time_step, window_dur, sliding_dur):

    # 1. Operation C1: textFileStream
    inputDStream = ssc.textFileStream(monitoring_dir)

    # 2. Operation T1: window
    windowDStream = inputDStream.window(3 * time_step,
                                        2 * time_step)

    ...
```

Stateless and Stateful Operations



Stateless and Stateful Operations



…

dataset_dir_ti-1

dataset_dir_ti

time

…

inputRDD_ti-1

inputRDD_ti

inputRDD_ti+1

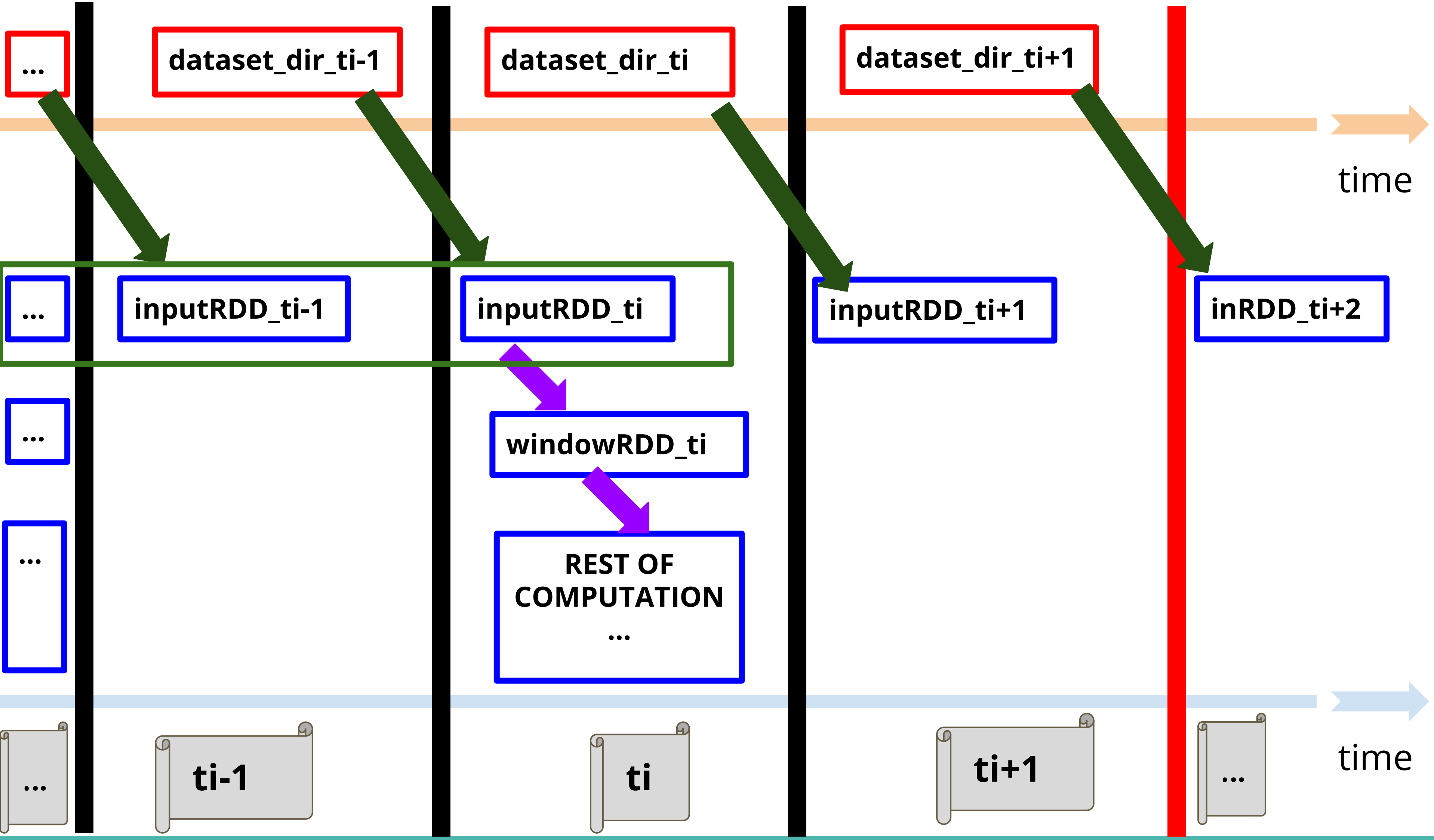
…

windowRDD_ti

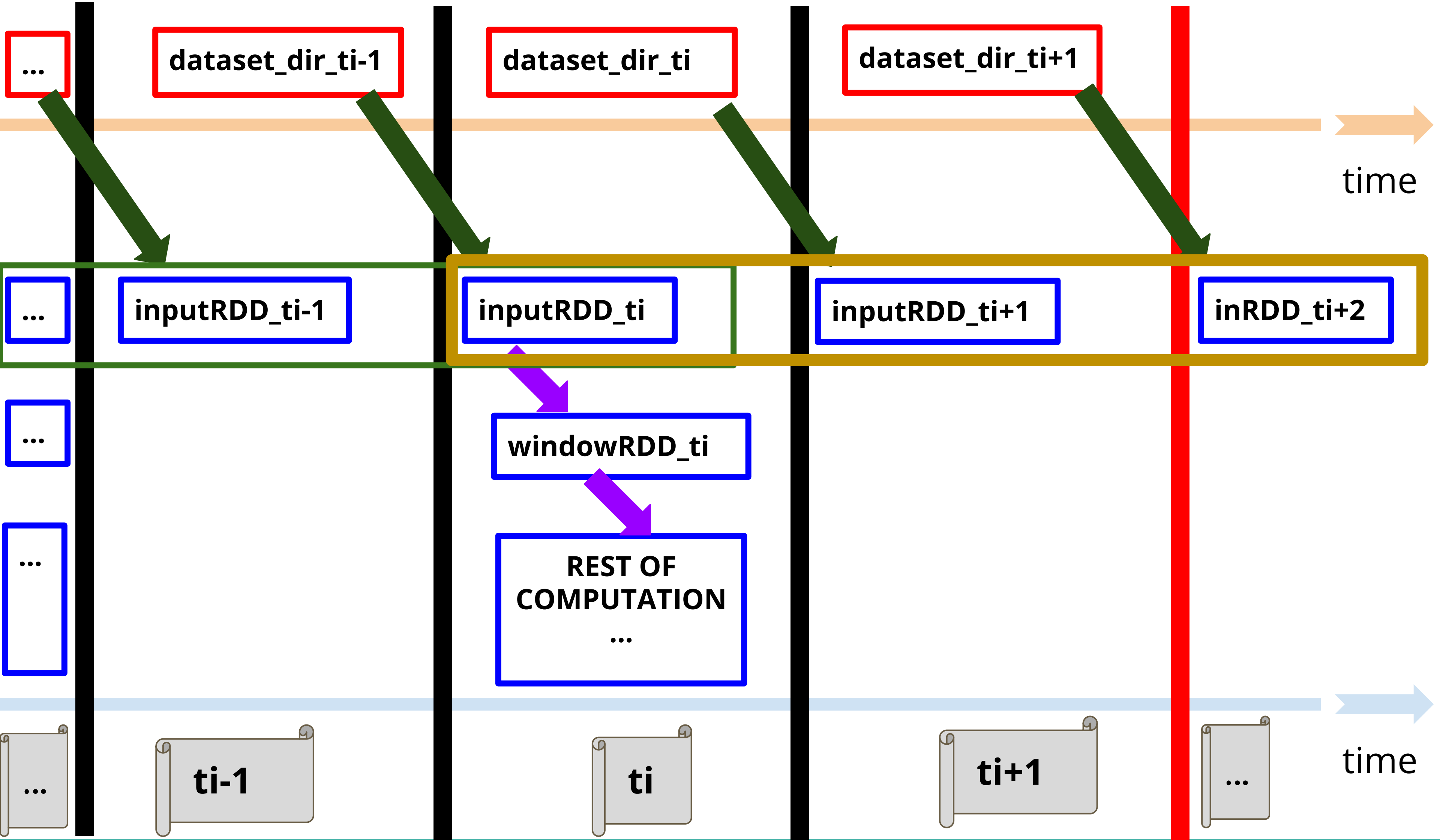
…

REST OF COMPUTATION …

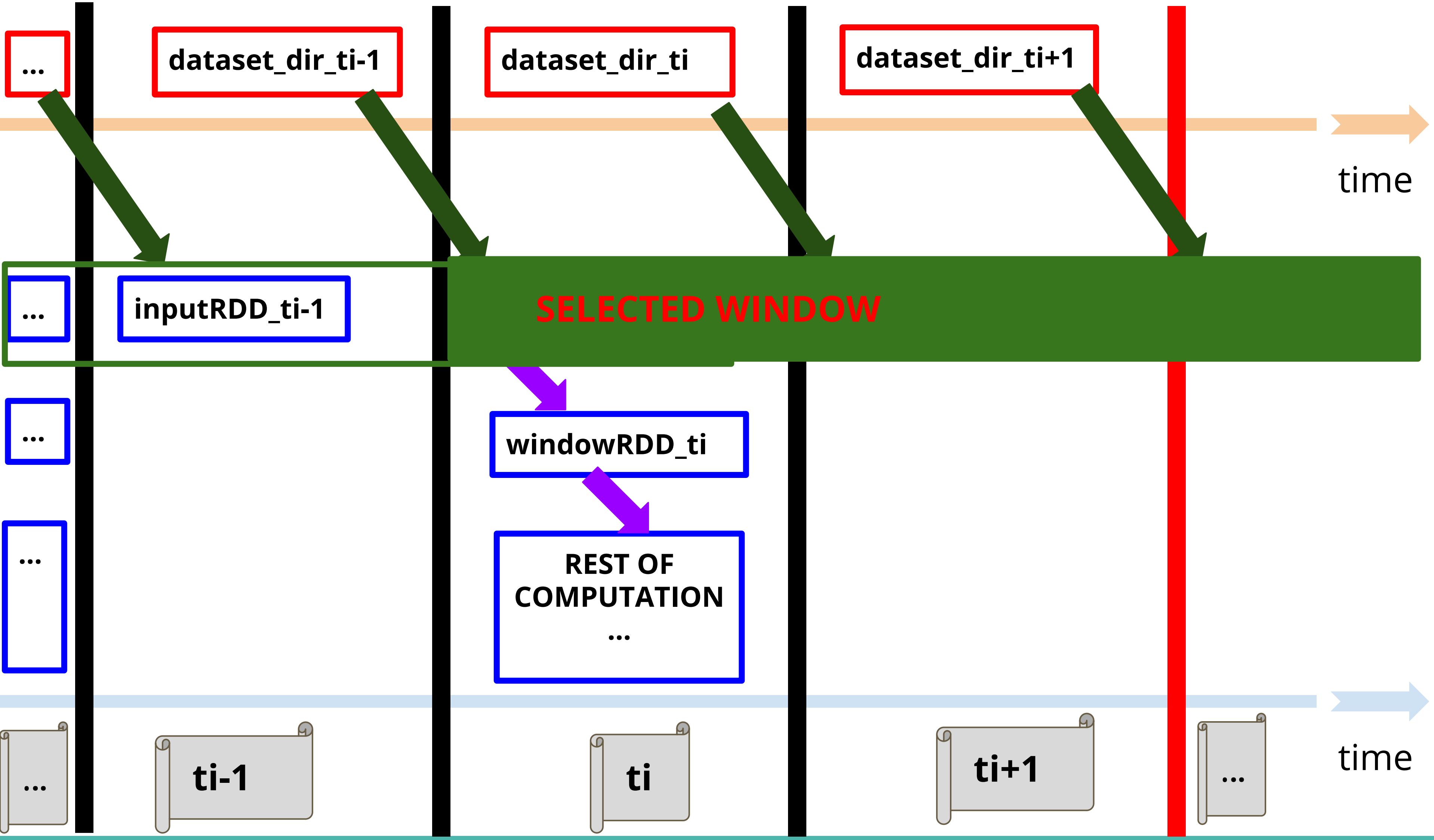
Stateless and Stateful Operations



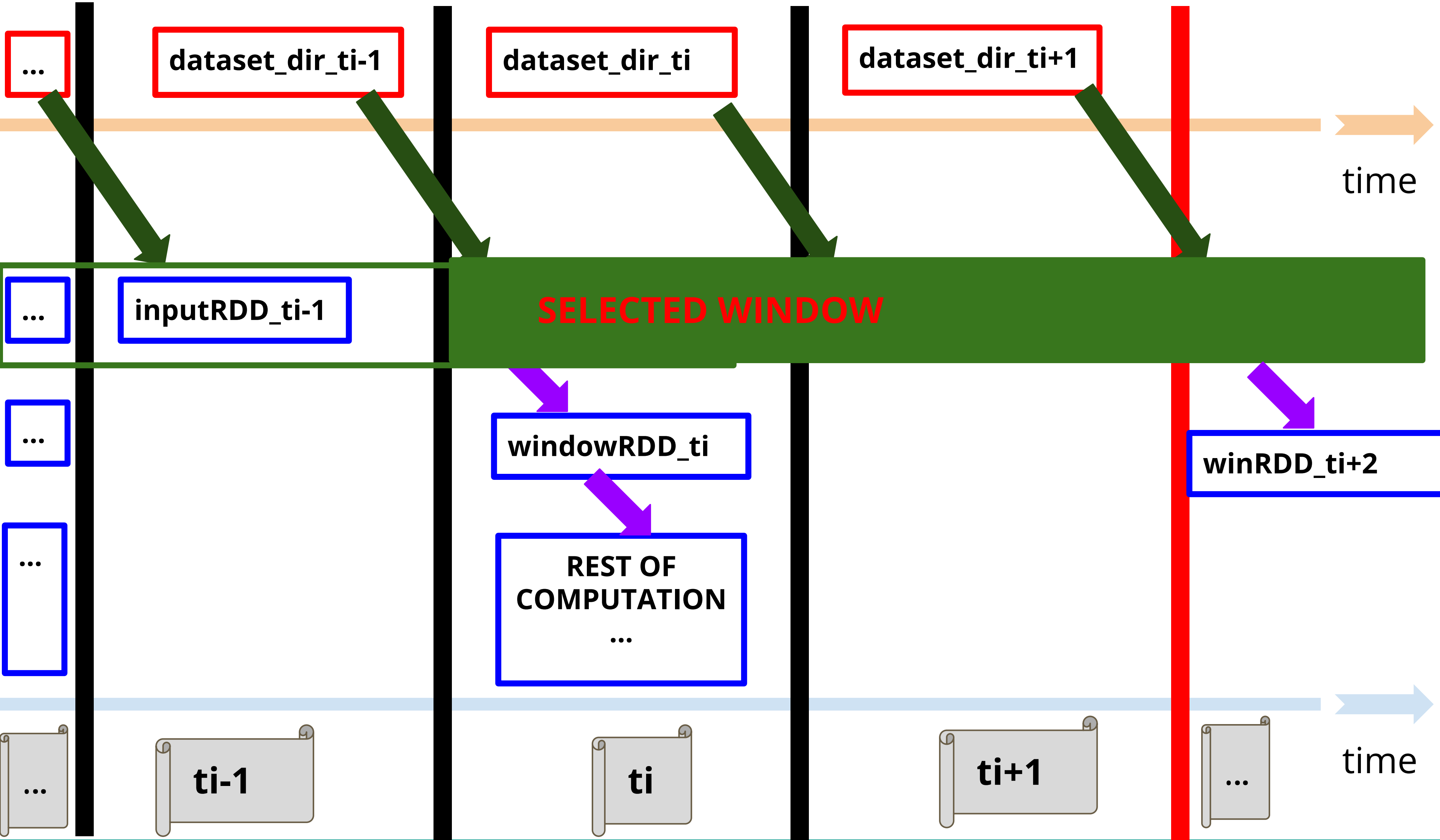
Stateless and Stateful Operations



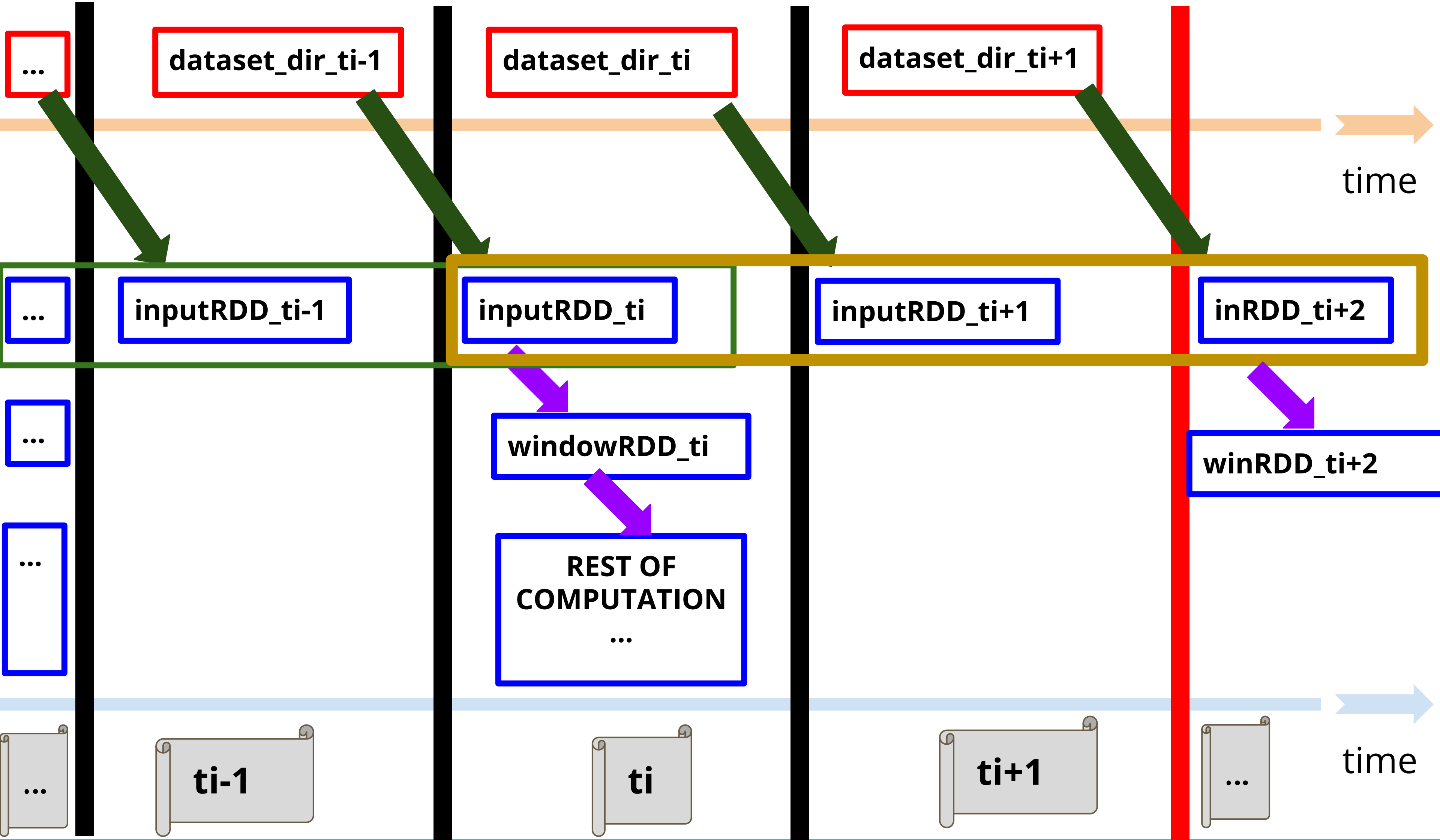
Stateless and Stateful Operations



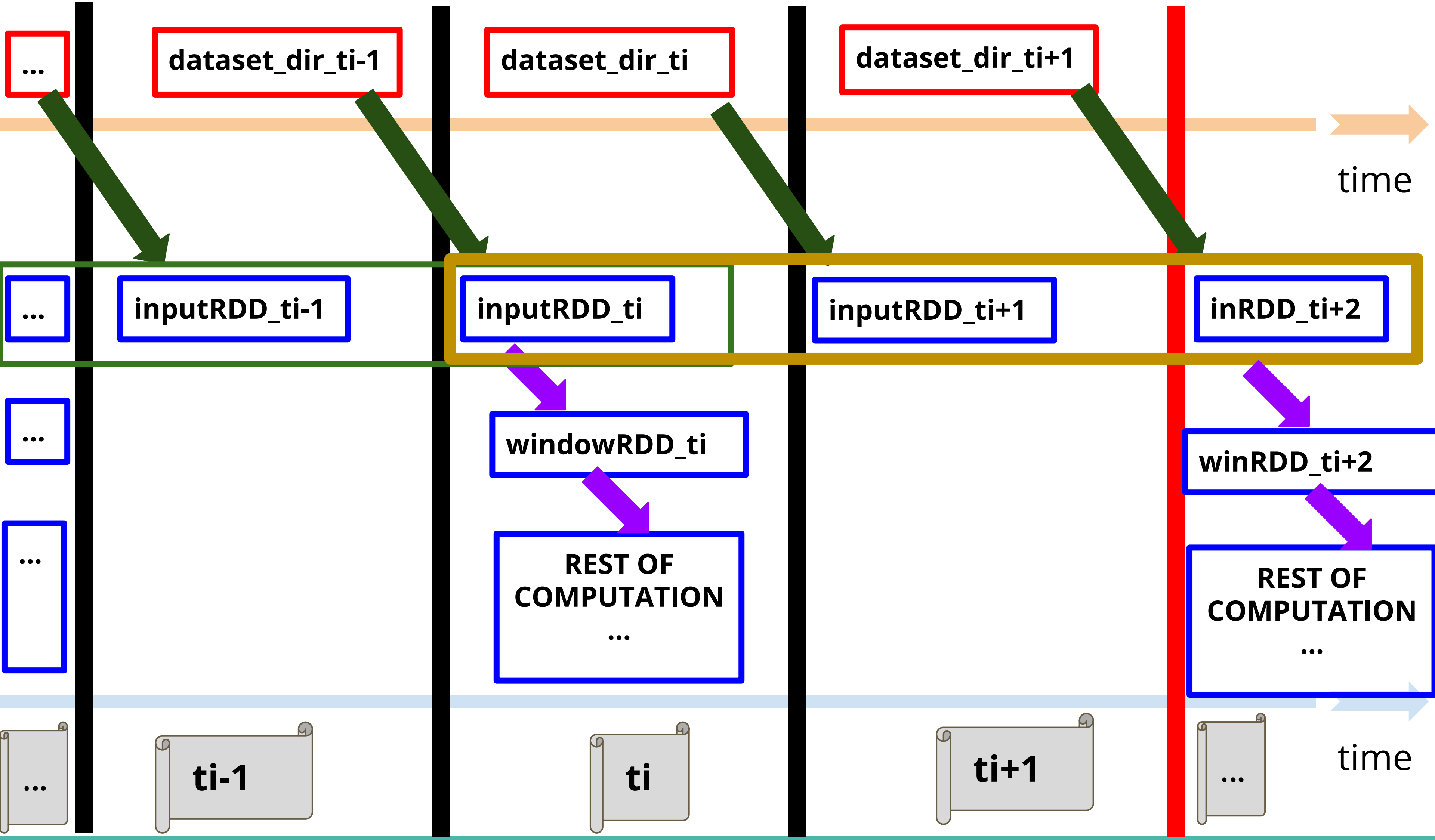
Stateless and Stateful Operations



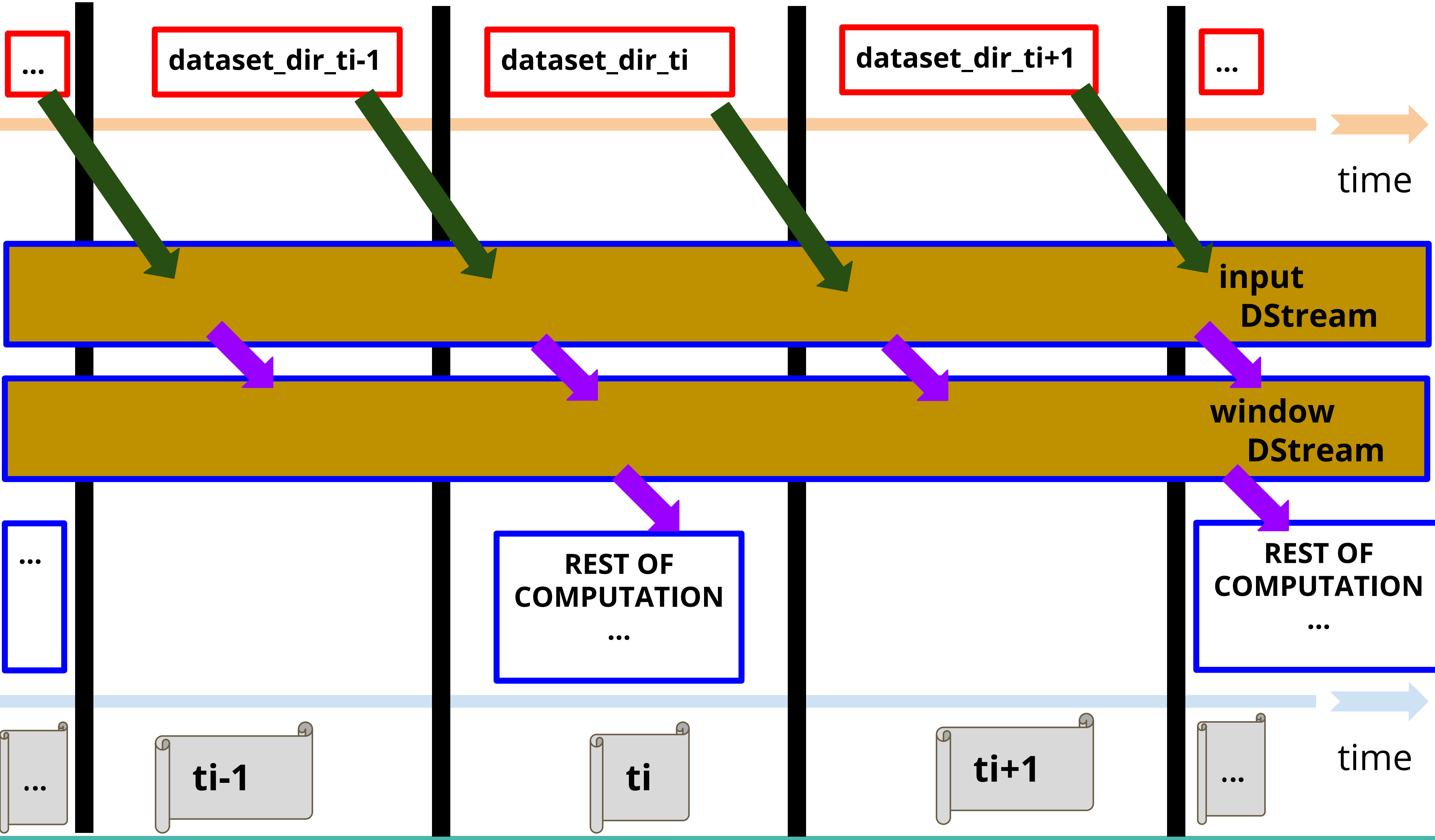
Stateless and Stateful Operations



Stateless and Stateful Operations



Stateless and Stateful Operations



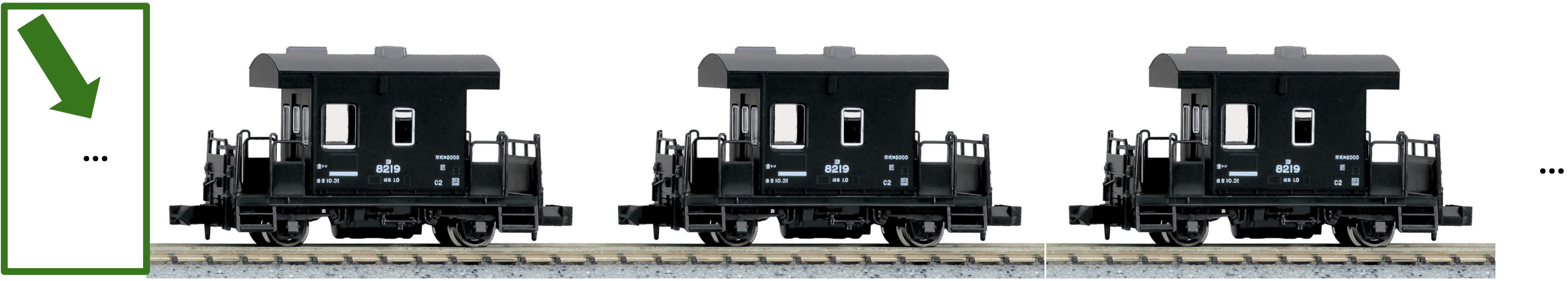
Stateless and Stateful Operations

Let's see the second category of Stateful Operations:

Update State Operations

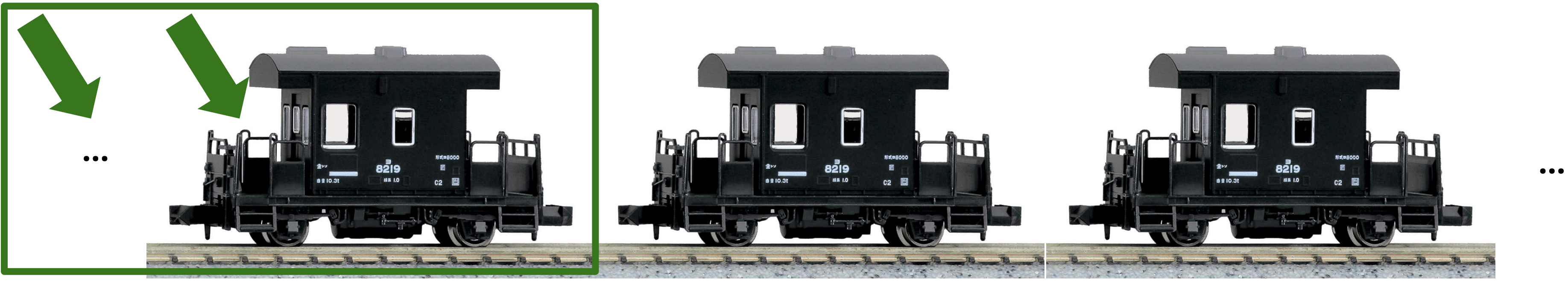
Stateless and Stateful Operations

On Update state operations, the **DStream aggregates the RDDs** processed so far over time.



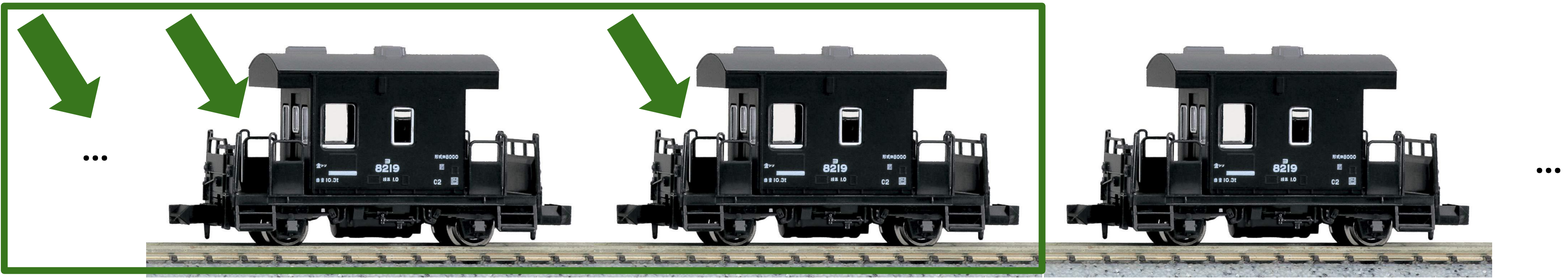
Stateless and Stateful Operations

On Update state operations, the **DStream aggregates the RDDs** processed so far over time.



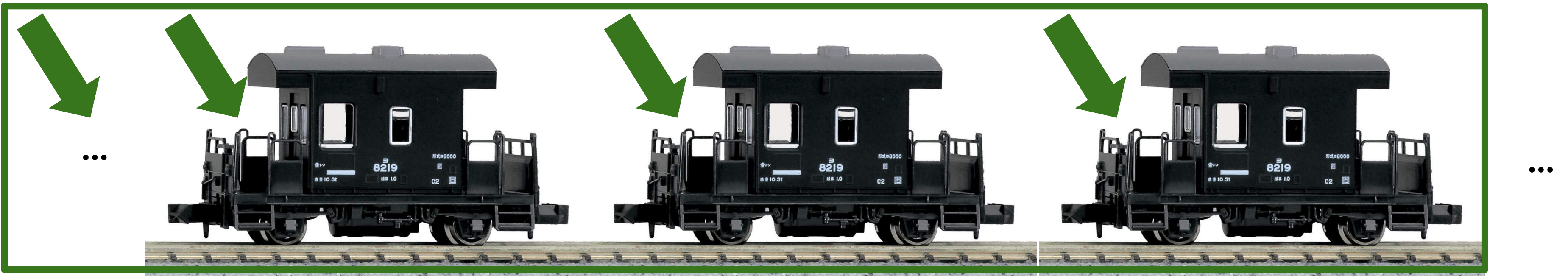
Stateless and Stateful Operations

On Update state operations, the **DStream aggregates the RDDs** processed so far over time.



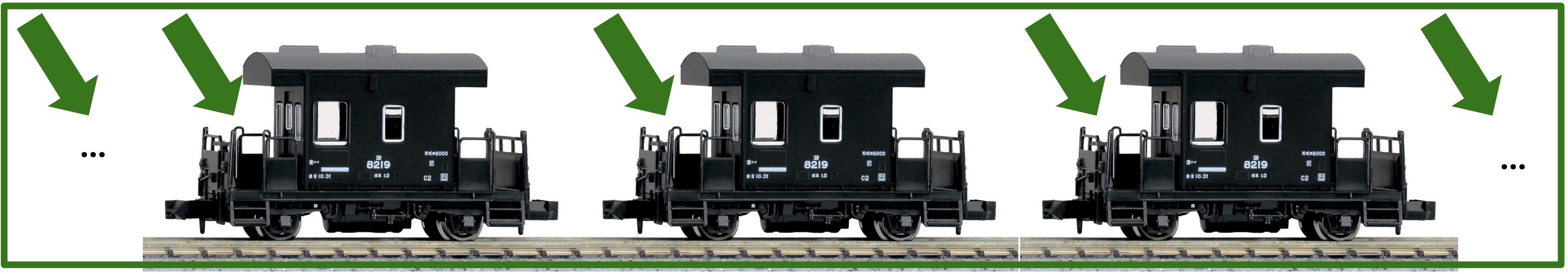
Stateless and Stateful Operations

On Update state operations, the **DStream aggregates the RDDs** processed so far over time.



Stateless and Stateful Operations

On Update state operations, the **DStream aggregates the RDDs** processed so far over time.



Stateless and Stateful Operations

On each time interval **ti**
the **updateStateByKey** operation requires a function `f` with 2 parameters:

Stateless and Stateful Operations

On each time interval **ti**
the **updateStateByKey** operation requires a function `f` with 2 parameters:

1. The list of new values `[v1, v2, …, vk]` found for the particular key `keyA` in the time interval **ti** being considered.

Stateless and Stateful Operations

On each time interval **ti**
the **updateStateByKey** operation requires a function `f` with 2 parameters:

1. The list of new values `[v1, v2, …, vk]` found for the particular key `keyA` in the time interval **ti** being considered.

2. The current aggregated value `val`$_a$ for `keyA` over the past time intervals **`[t1, t2, …, ti-1]`**.

Stateless and Stateful Operations

On each time interval **ti**
the **updateStateByKey** operation requires a function `f` with 2 parameters:

1. The list of new values `[v1, v2, …, vk]` found for the particular key `keyA` in the time interval **ti** being considered.

2. The current aggregated value `cur_agg_val` for `keyA` over the past time intervals **`[t1, t2, …, ti-1]`**.

   Note:
   ➢ If no previous pair `(keyA, v1)` was found during the past time intervals **`[t1, t2, …, ti-1]`**, then our function `f` will receive `None` as the current aggregated value `cur_agg_val` for `keyA`.
   ➢ Obviously, our function `f` needs to deal with this situation, as it will happen for the first appearance of each key in the dataset.

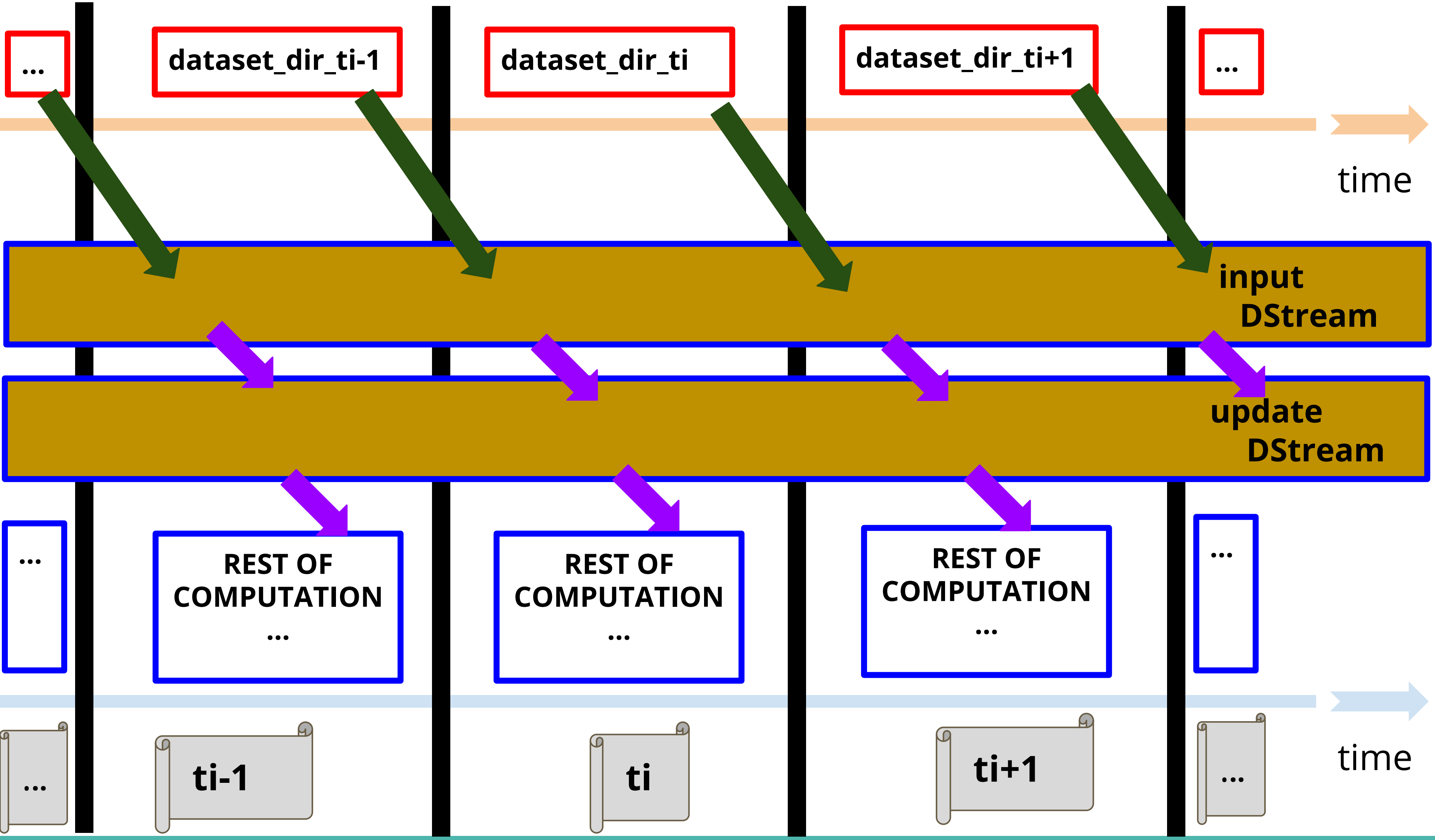
Stateless and Stateful Operations

Let’s reason with the following piece of code:

- We assume the streaming dataset we receive is already in (key, value) format.

```
def f( time_inteval_list_of_new_values, cur_agg_val ):
    ...


def my_model(ssc, monitoring_dir, time_step):

    # 1. Operation C1: textFileStream
    inputDStream = ssc.textFileStream(monitoring_dir)

    # 2. Operation T1: window
    updateDStream = inputDStream.updateStateByKey( f )

    ...
```

Stateless and Stateful Operations



...

dataset_dir_ti-1

dataset_dir_ti

dataset_dir_ti+1

...

time

input DStream

update DStream

...

REST OF COMPUTATION ...

REST OF COMPUTATION ...

REST OF COMPUTATION ...

...

...

ti-1

ti

ti+1

...

time

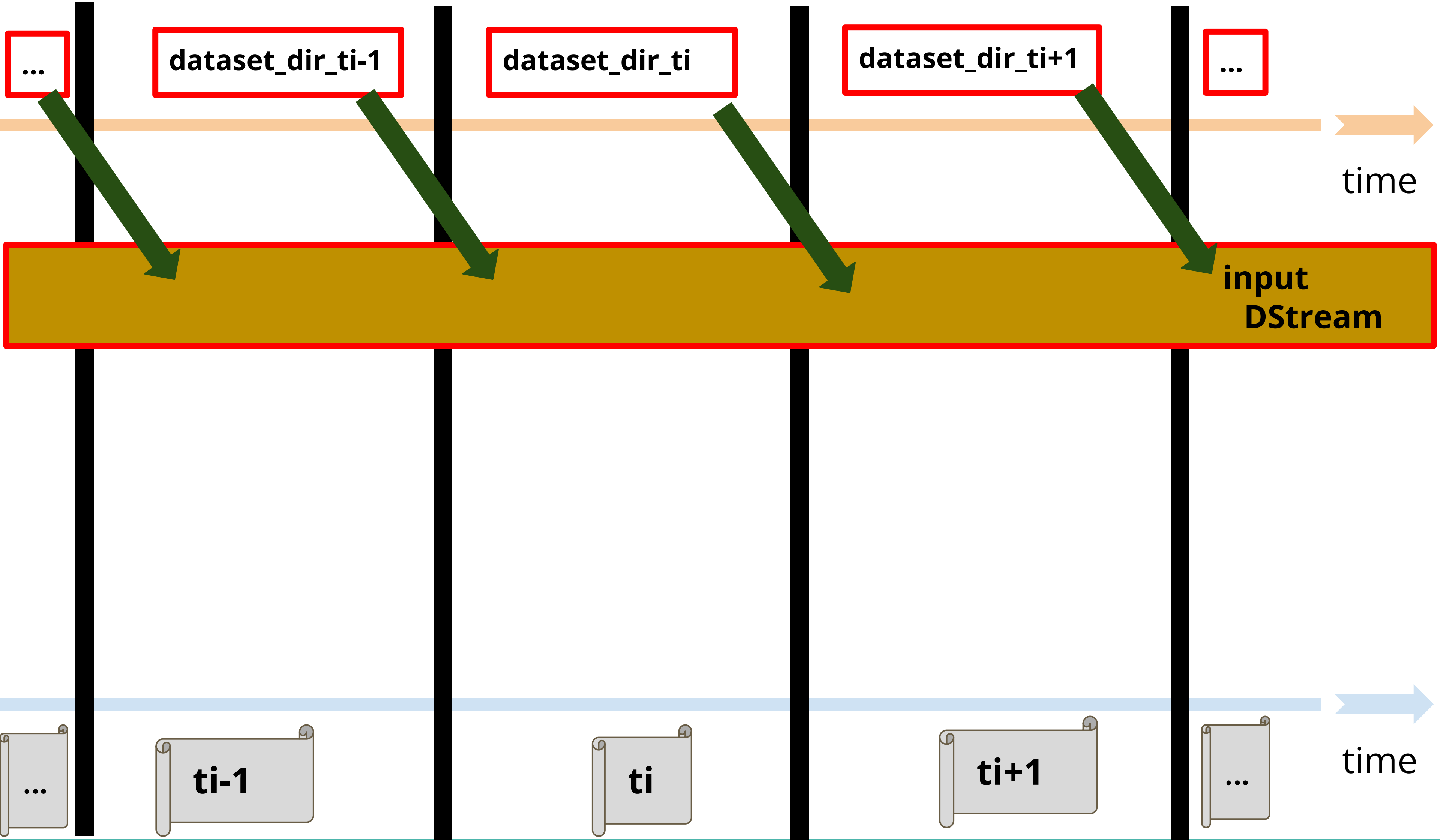
Stateless and Stateful Operations

Let’s see the computation over time w.r.t. **textFileStream**

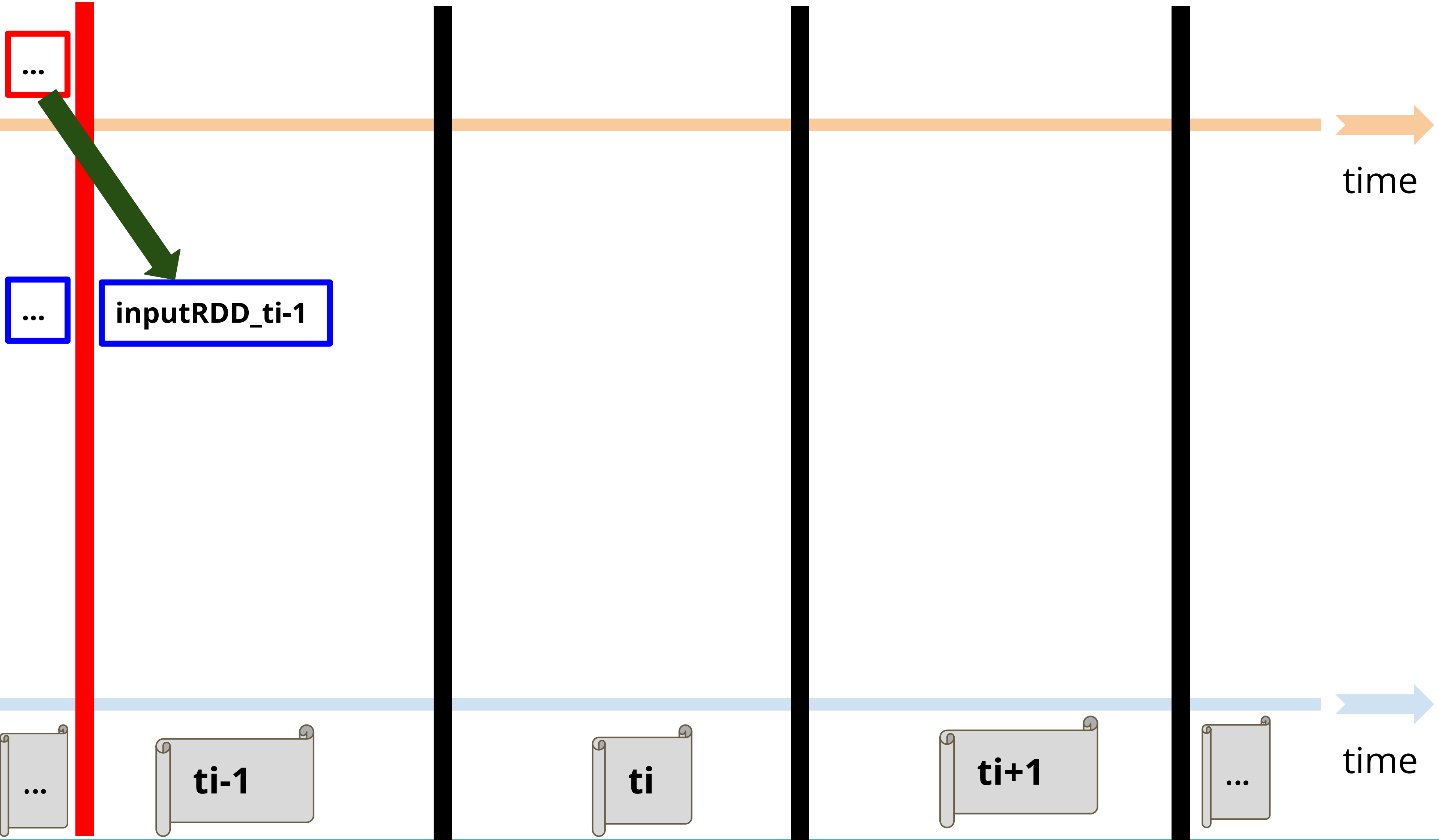
- We assume the streaming dataset we receive is already in (key, value) format.

```
def f( time_inteval_list_of_new_values, cur_agg_val ):
    ...


def my_model(ssc, monitoring_dir, time_step):
    # 1. Operation C1: textFileStream
    inputDStream = ssc.textFileStream(monitoring_dir)

    # 2. Operation T1: window
    updateDStream = inputDStream.updateStateByKey( f )

    ...
```

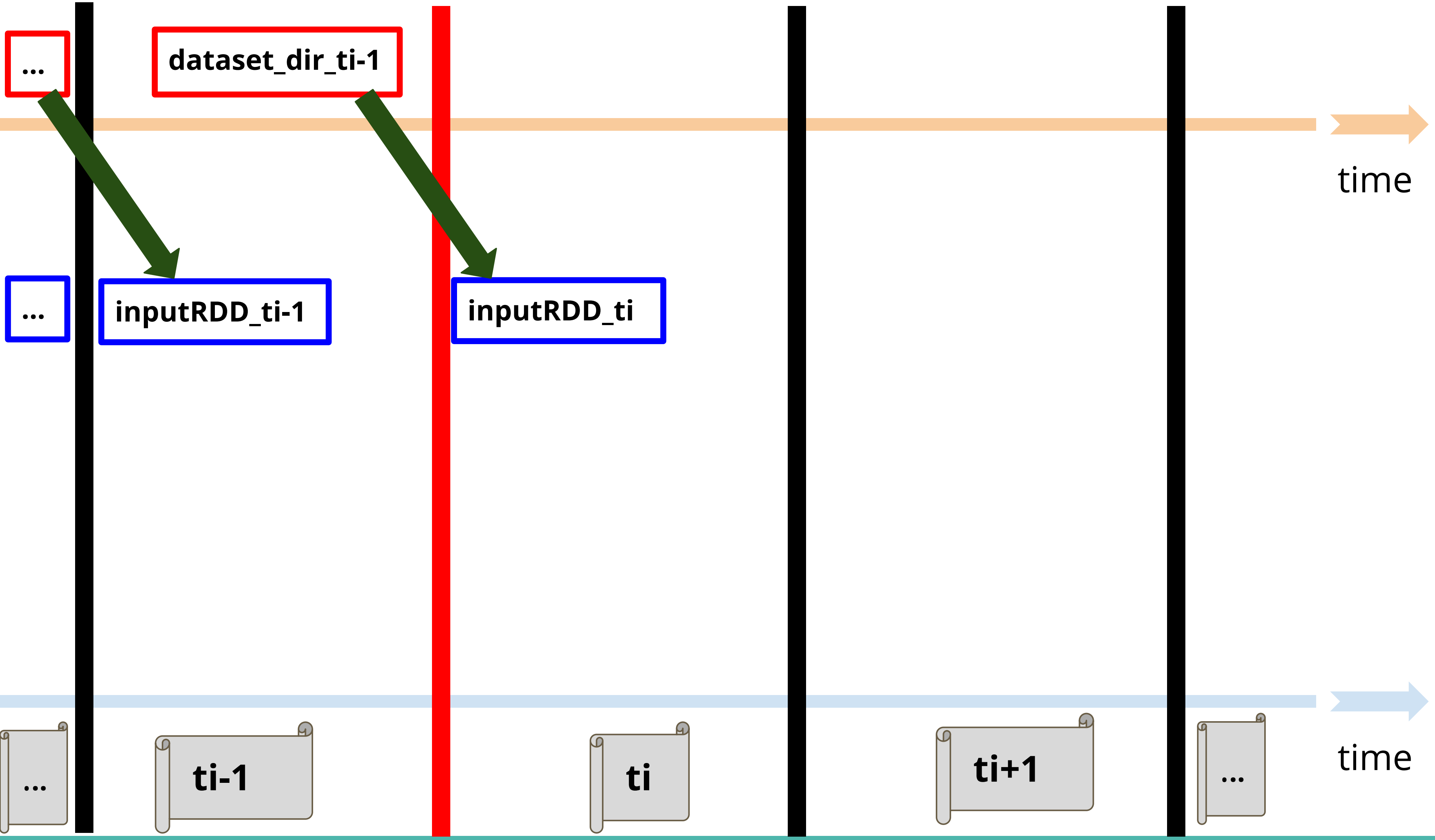
Stateless and Stateful Operations



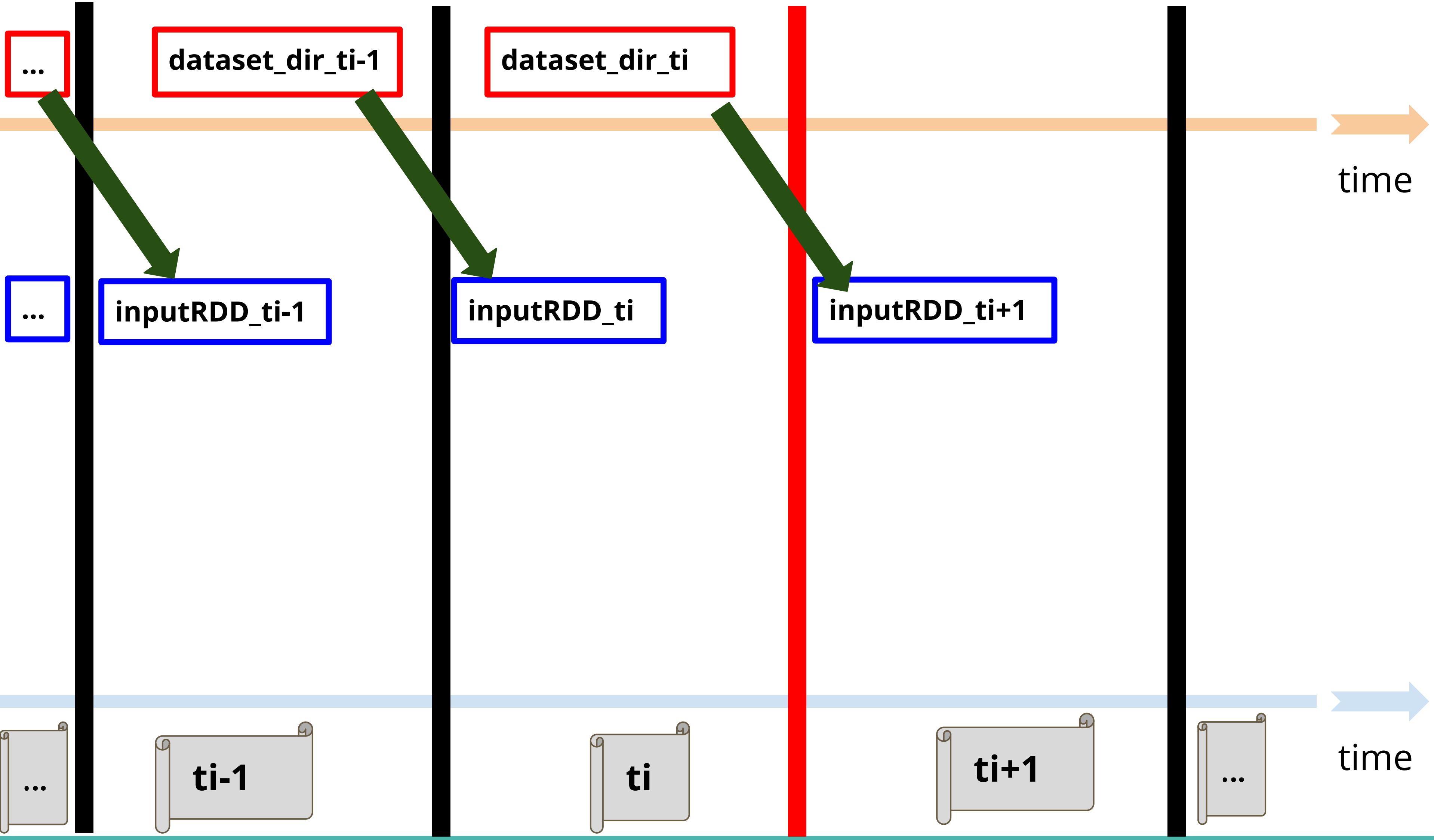
Stateless and Stateful Operations



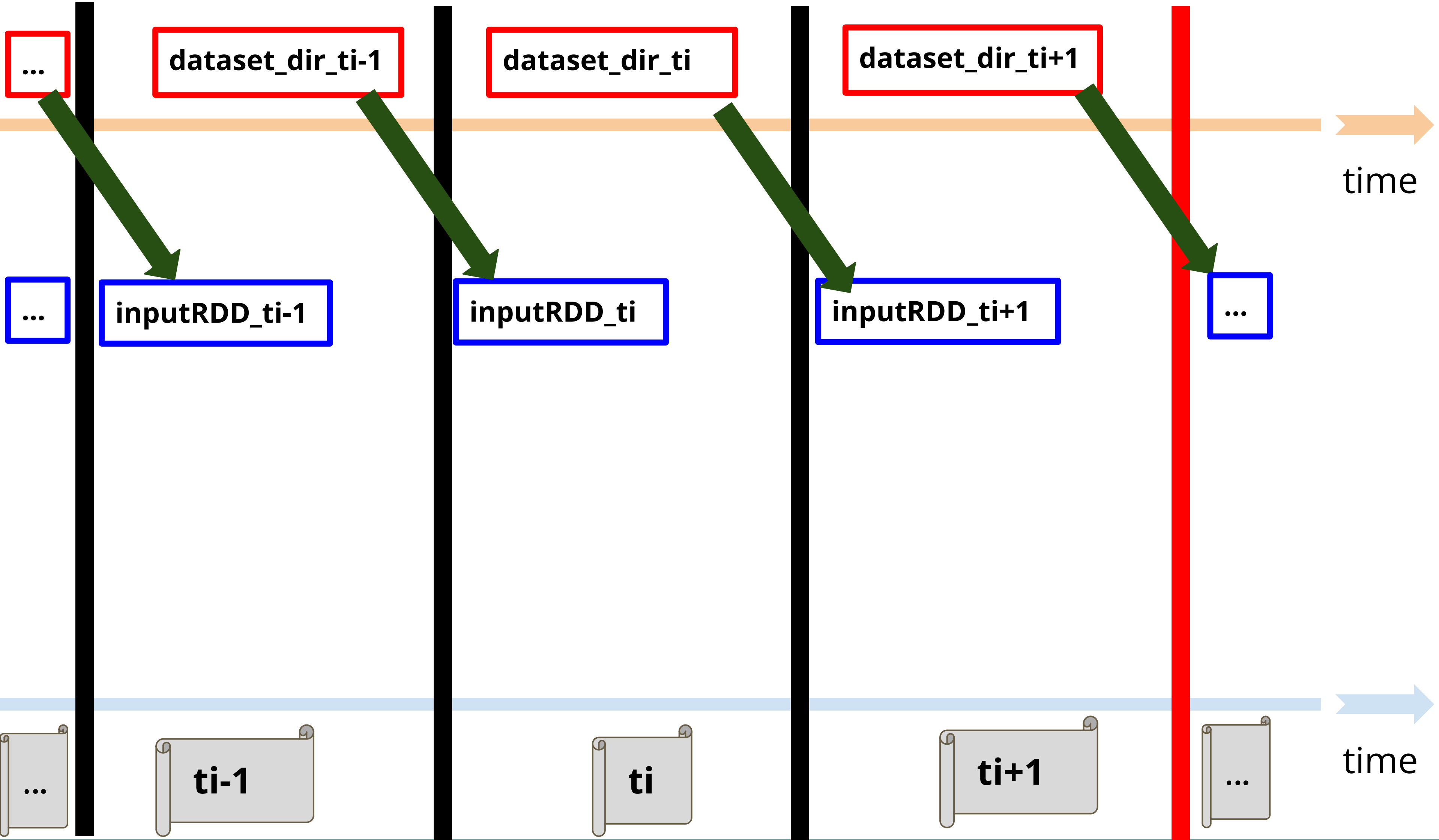
Stateless and Stateful Operations



Stateless and Stateful Operations



Stateless and Stateful Operations



Stateless and Stateful Operations

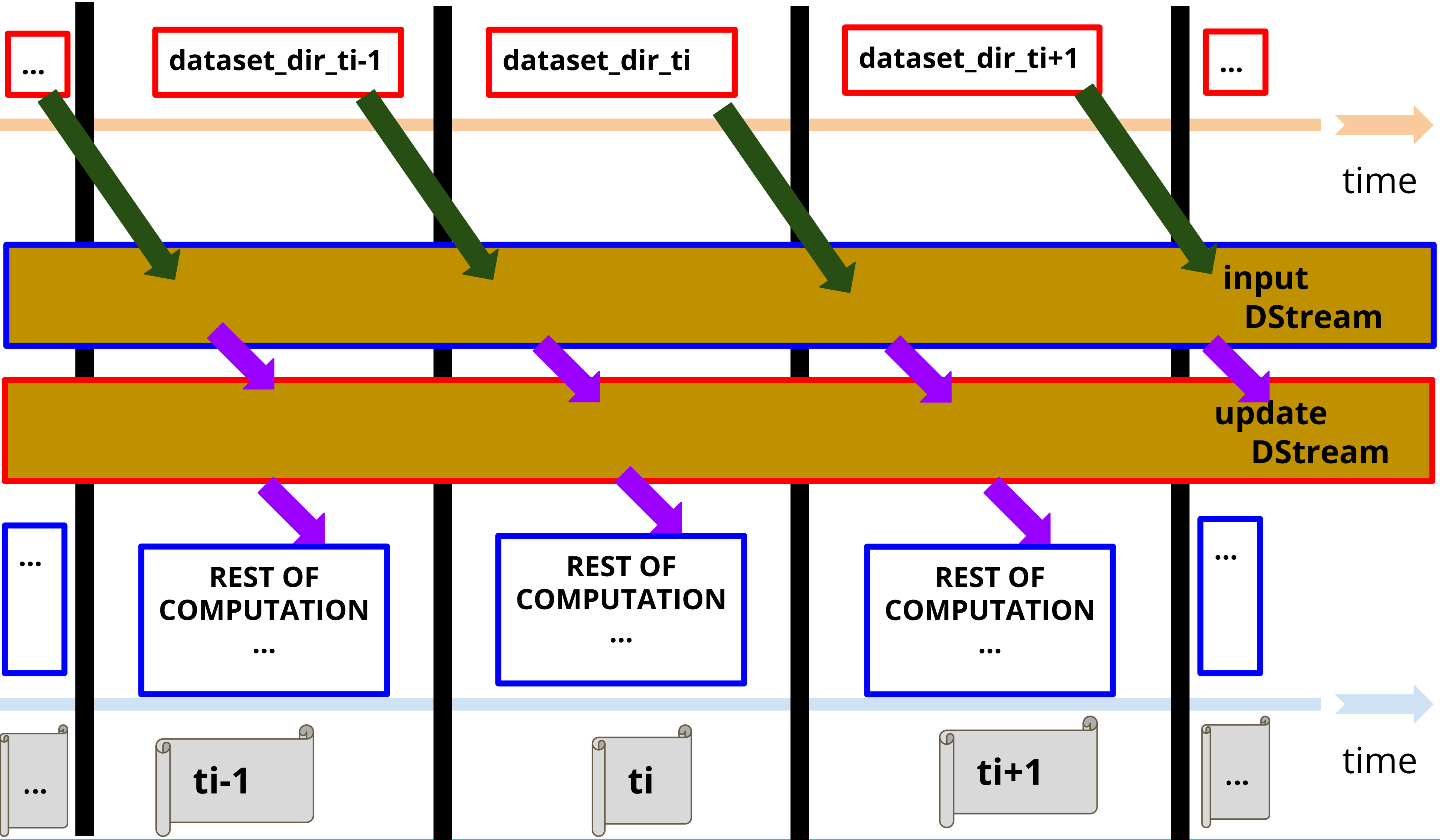
Let’s see the computation over time w.r.t. **updateStateByKey**

- We assume the streaming dataset we receive is already in (key, value) format.

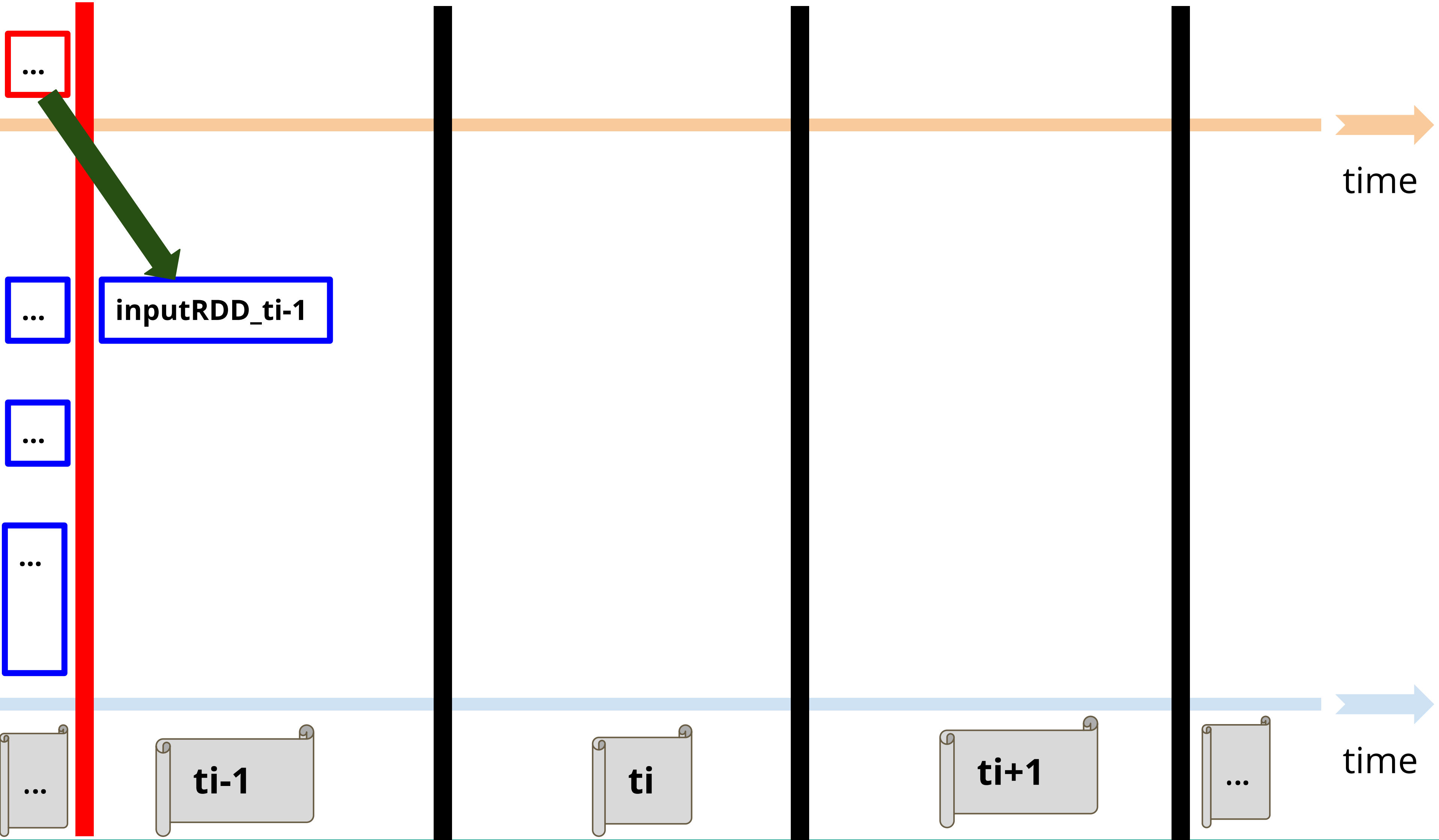
```
def f( time_inteval_list_of_new_values, cur_agg_val ):
    ...
```

```
def my_model(ssc, monitoring_dir, time_step):

    # 1. Operation C1: textFileStream
    inputDStream = ssc.textFileStream(monitoring_dir)

    # 2. Operation T1: window
    updateDStream = inputDStream.updateStateByKey( f )

    ...
```

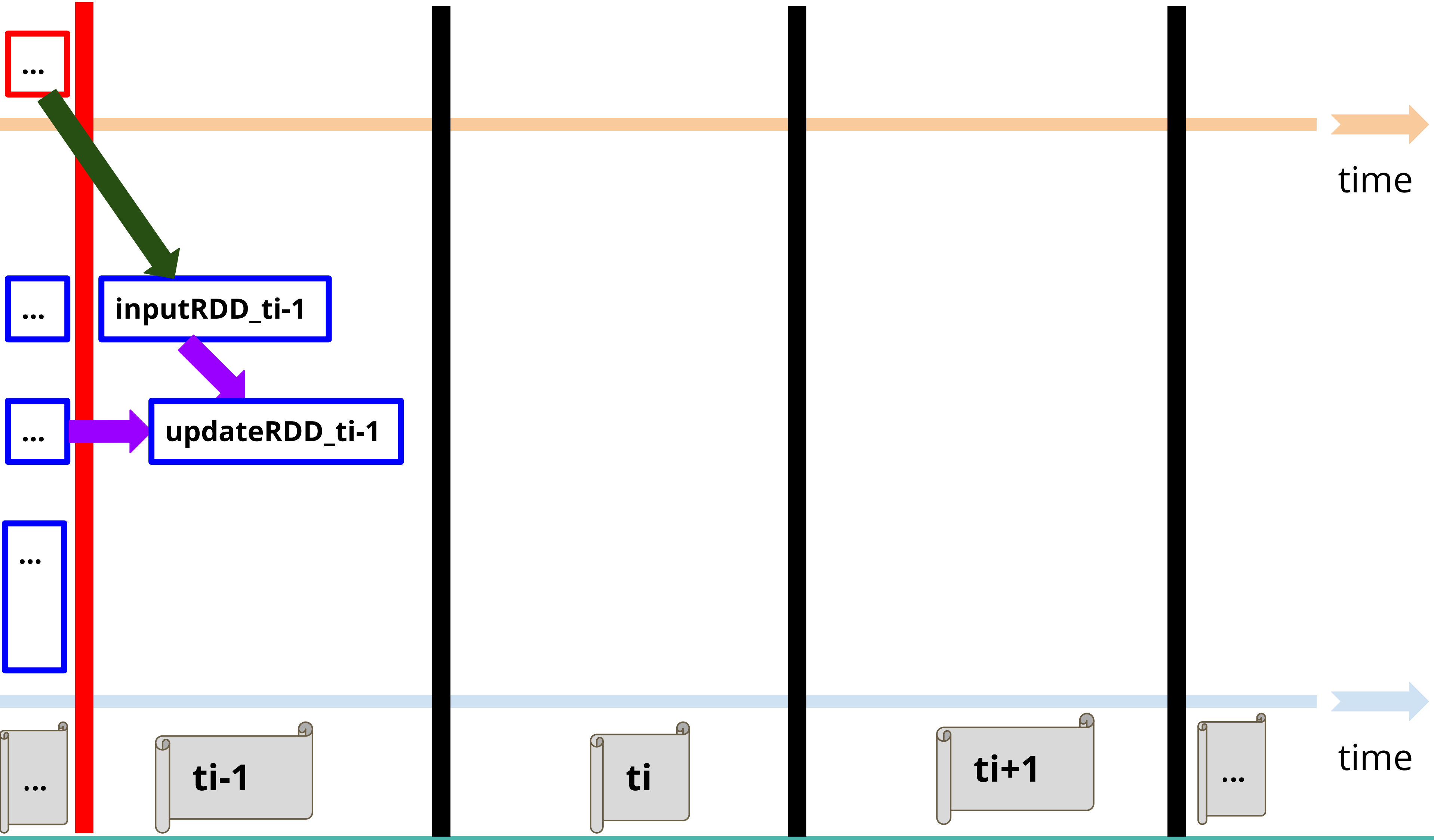
Stateless and Stateful Operations



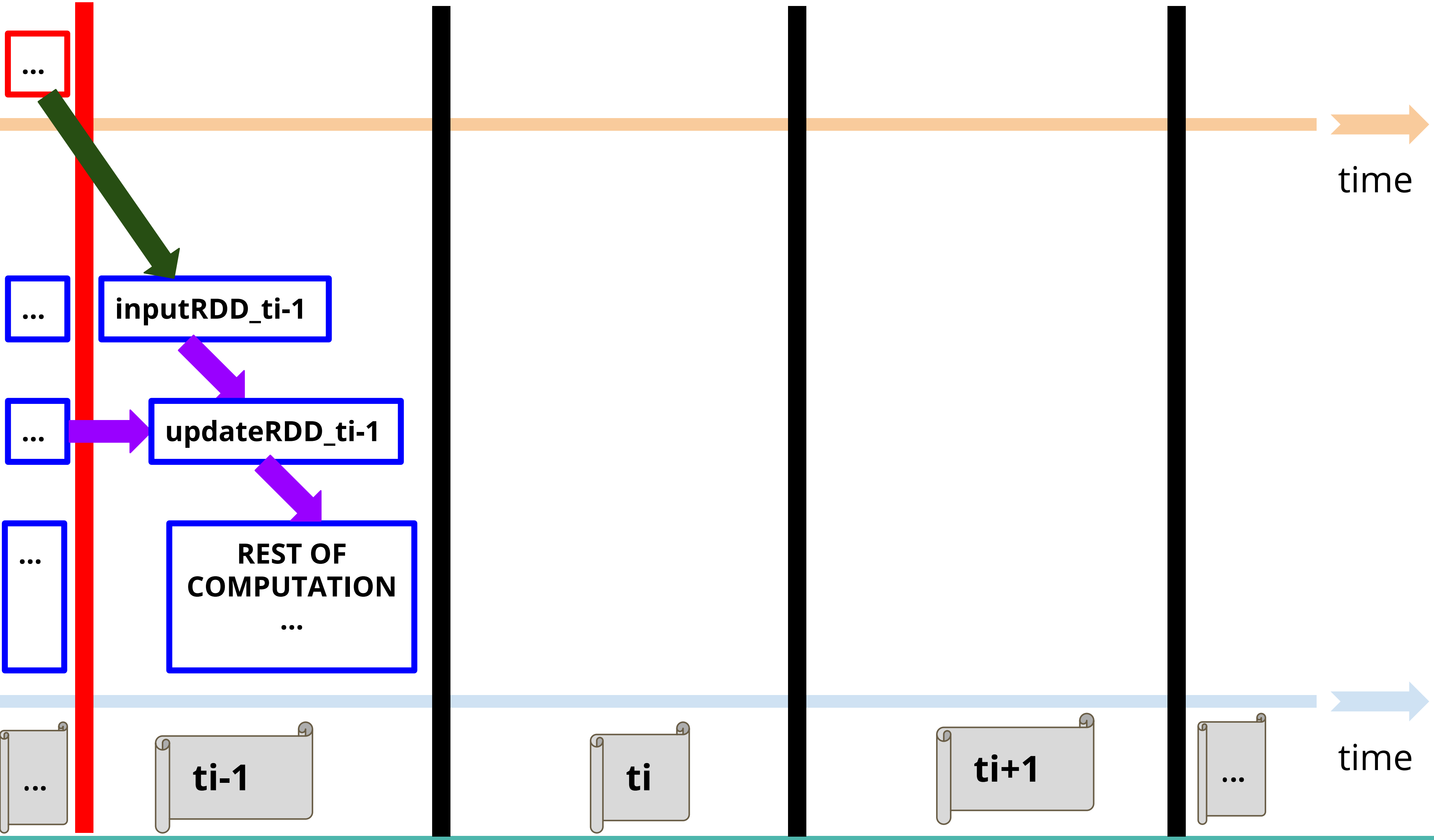
Stateless and Stateful Operations



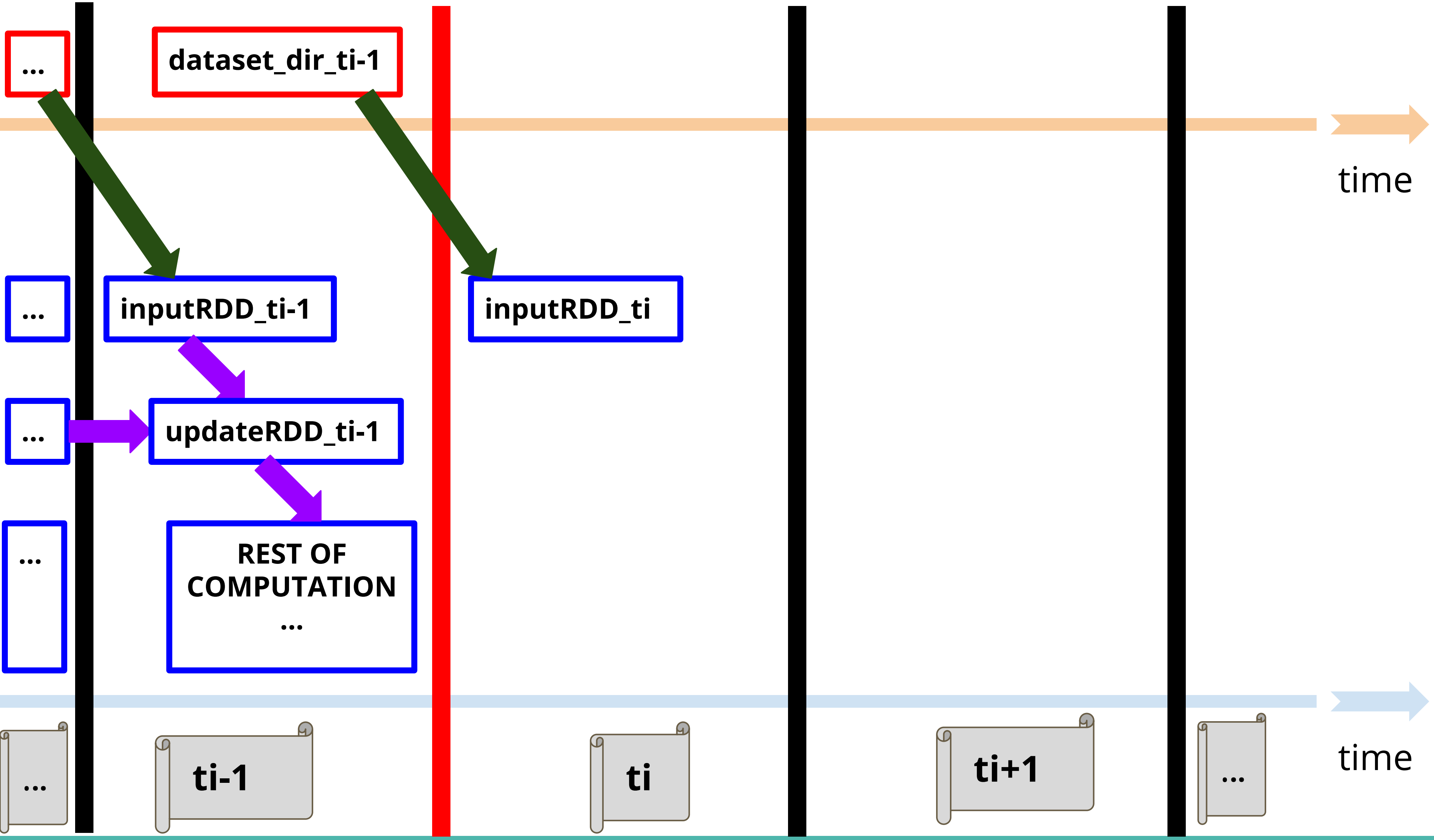
Stateless and Stateful Operations



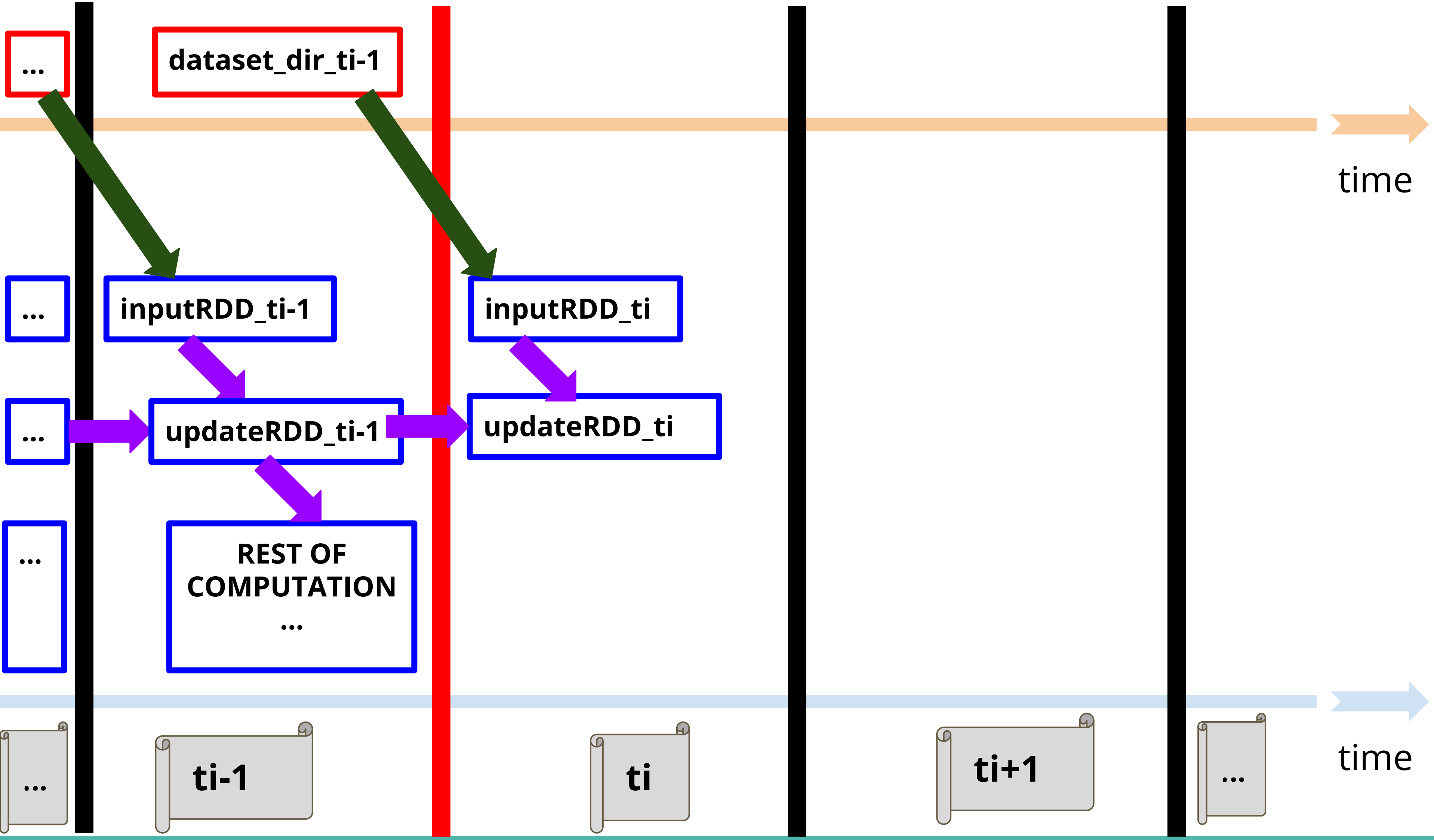
Stateless and Stateful Operations



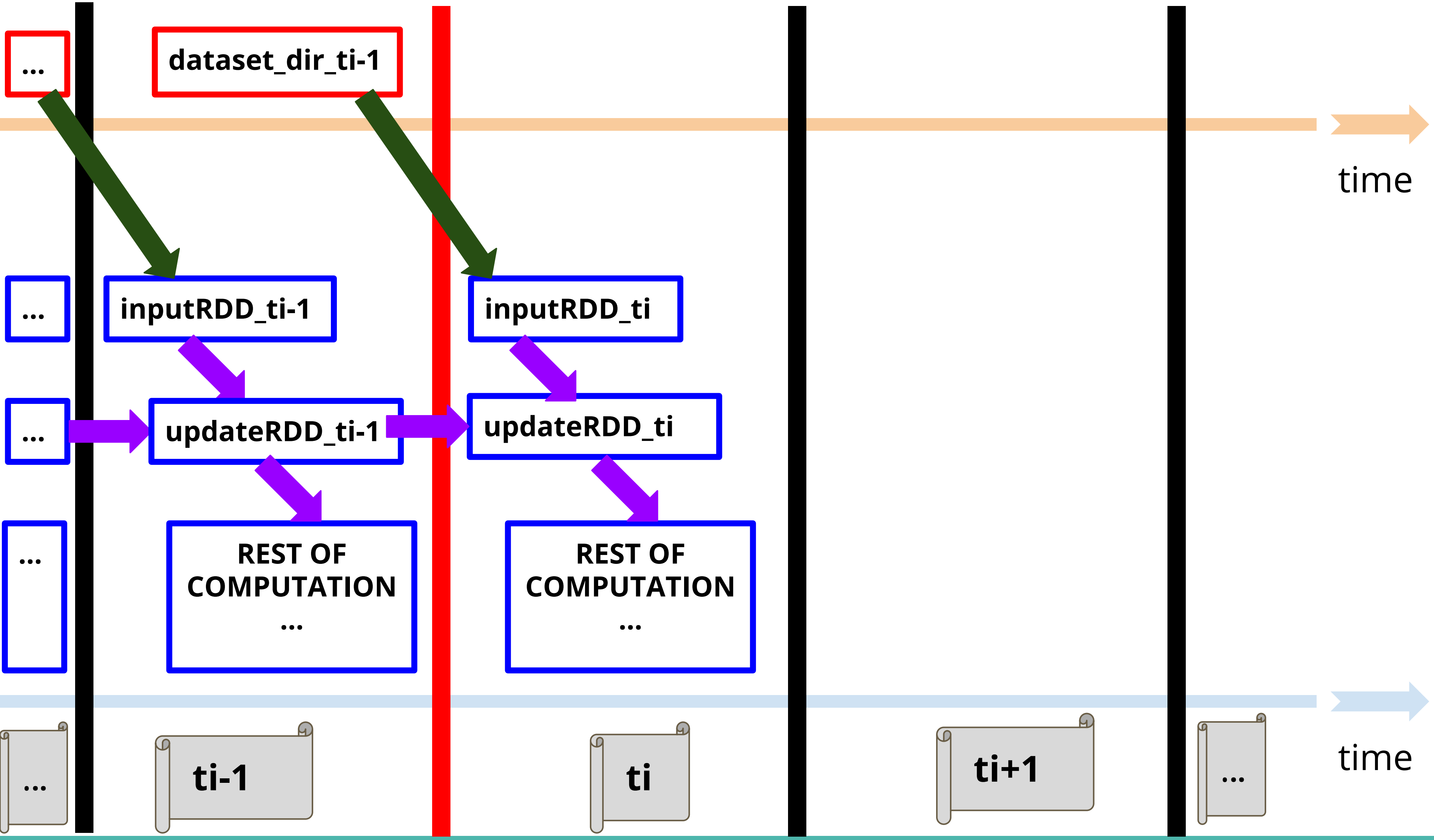
Stateless and Stateful Operations



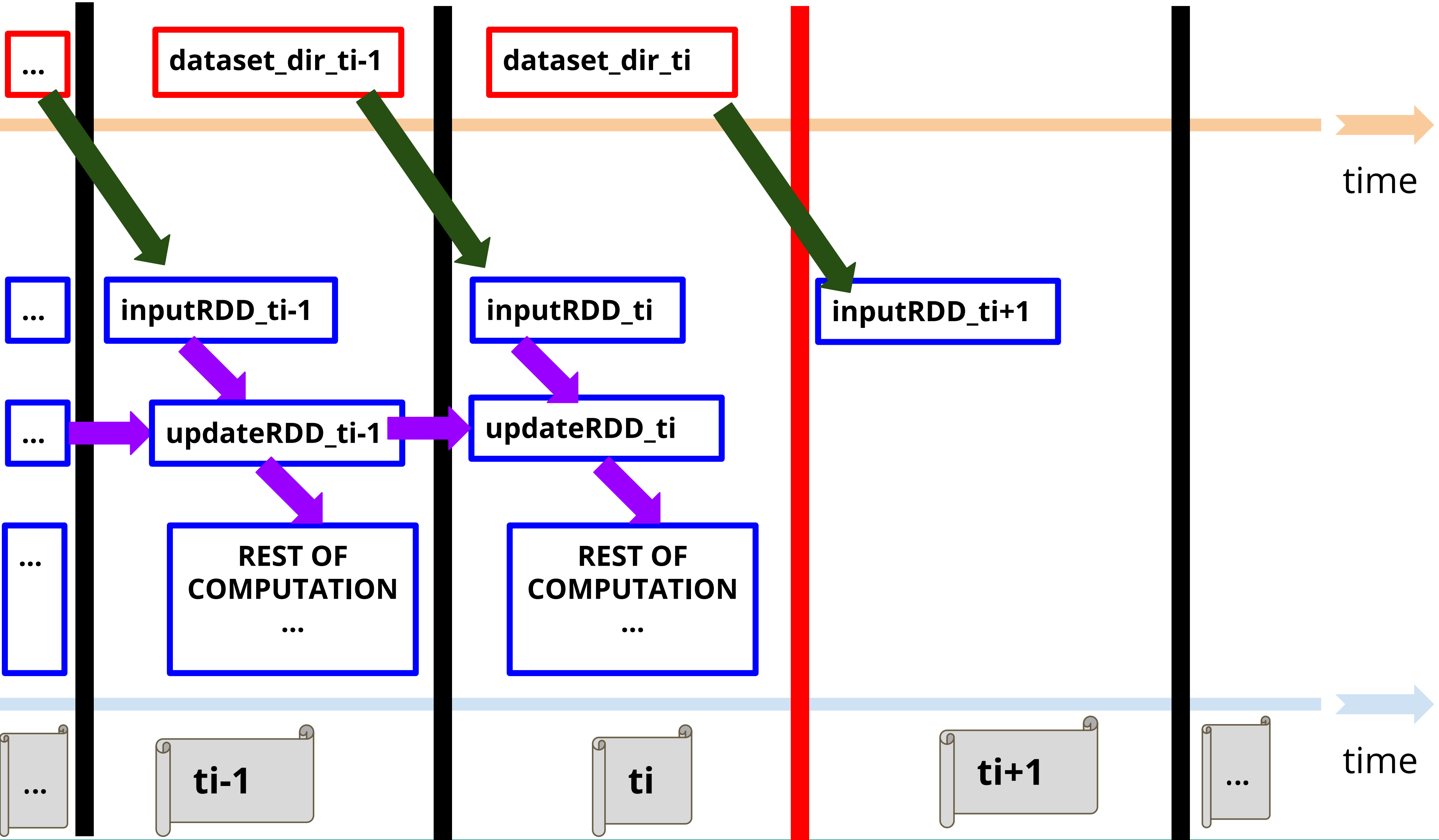
Stateless and Stateful Operations



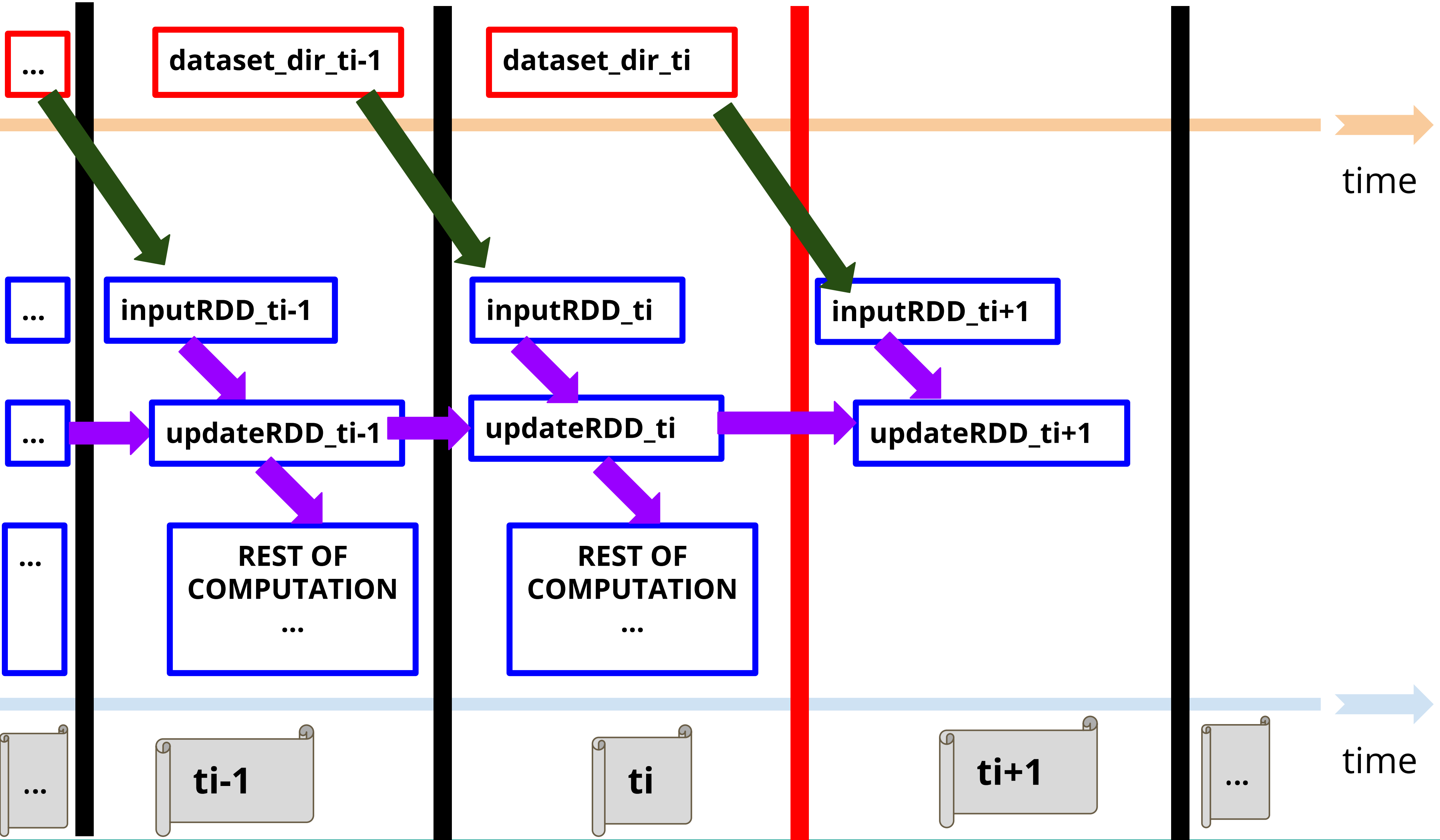
Stateless and Stateful Operations



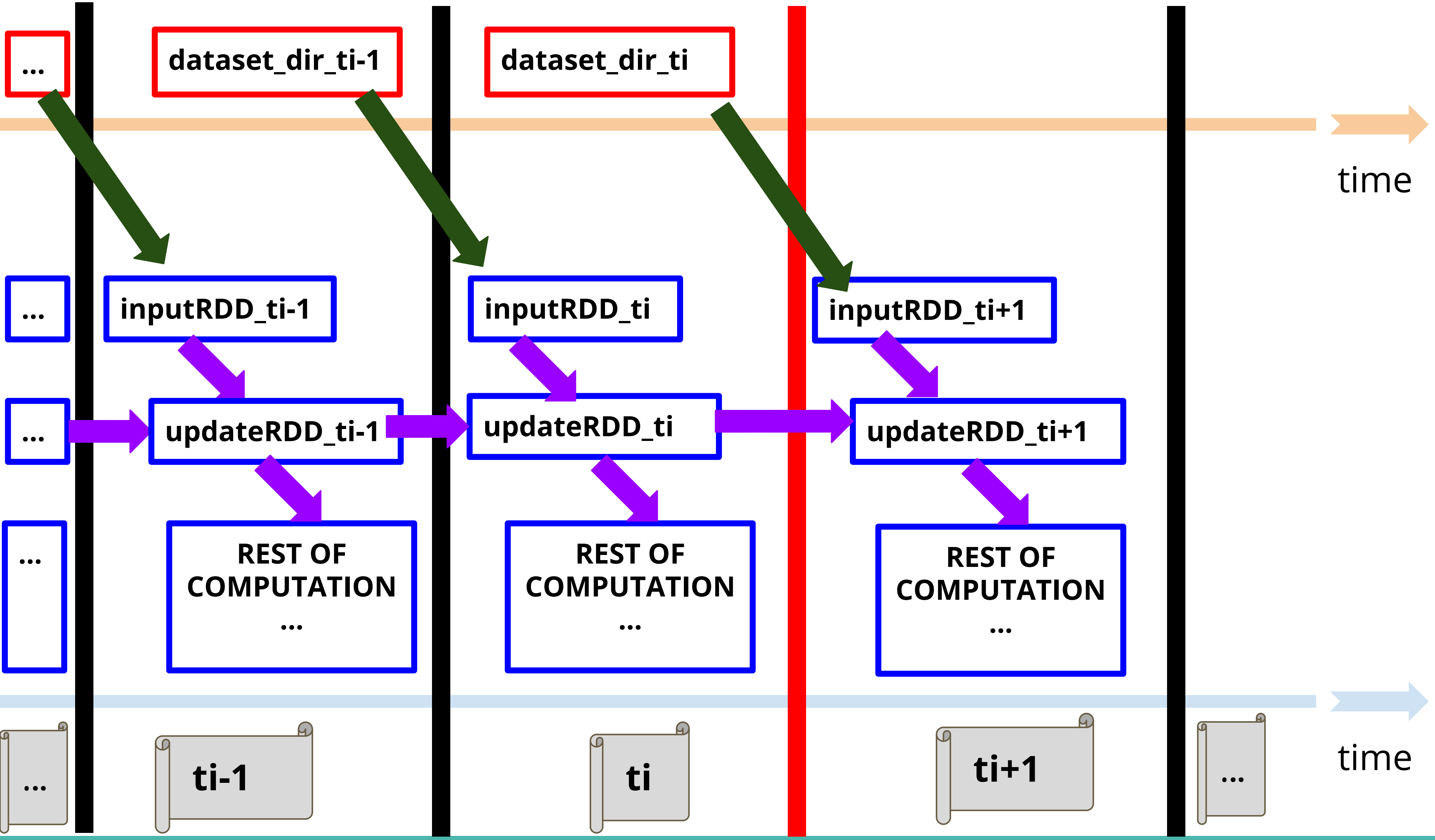
Stateless and Stateful Operations



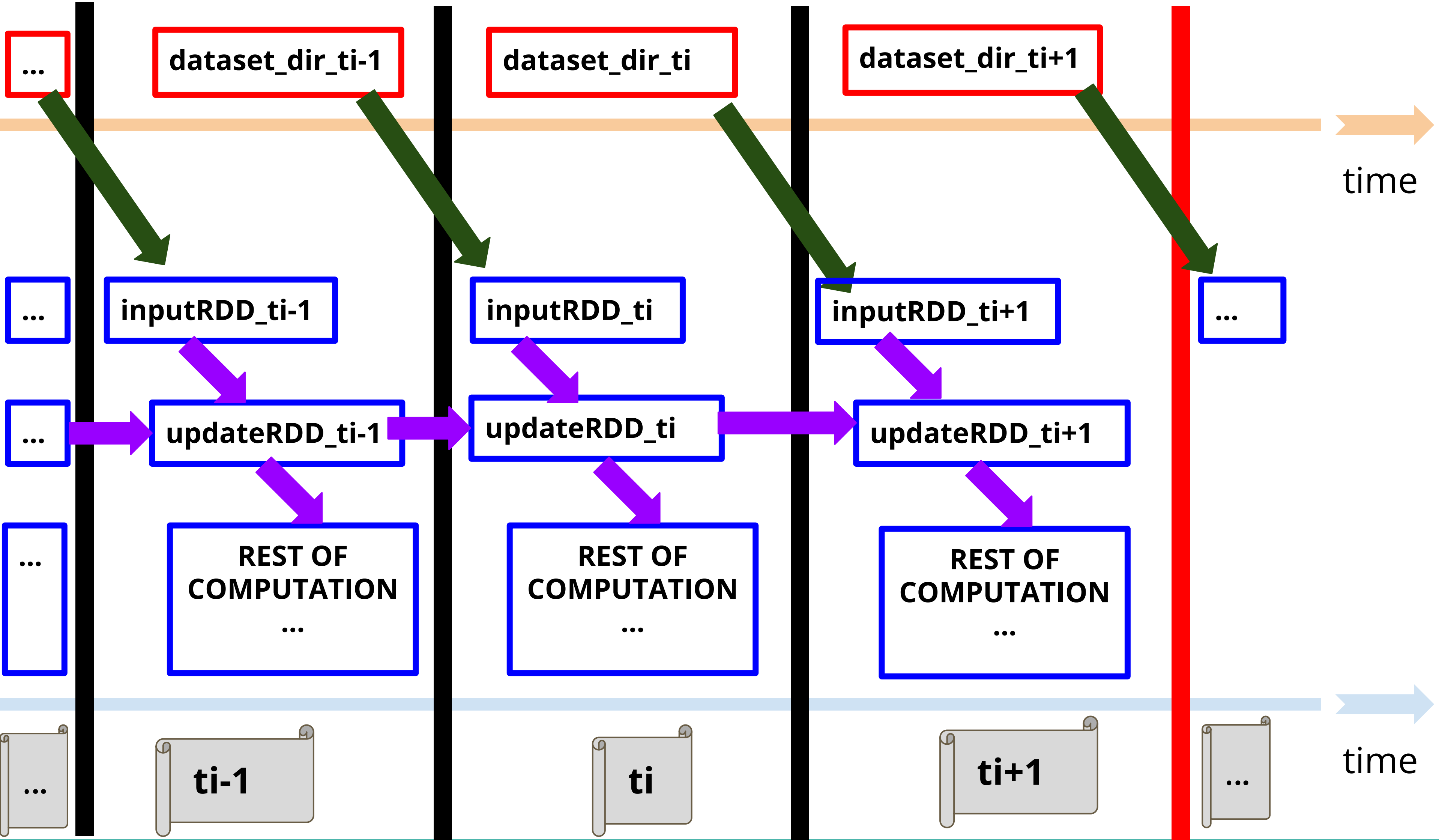
Stateless and Stateful Operations



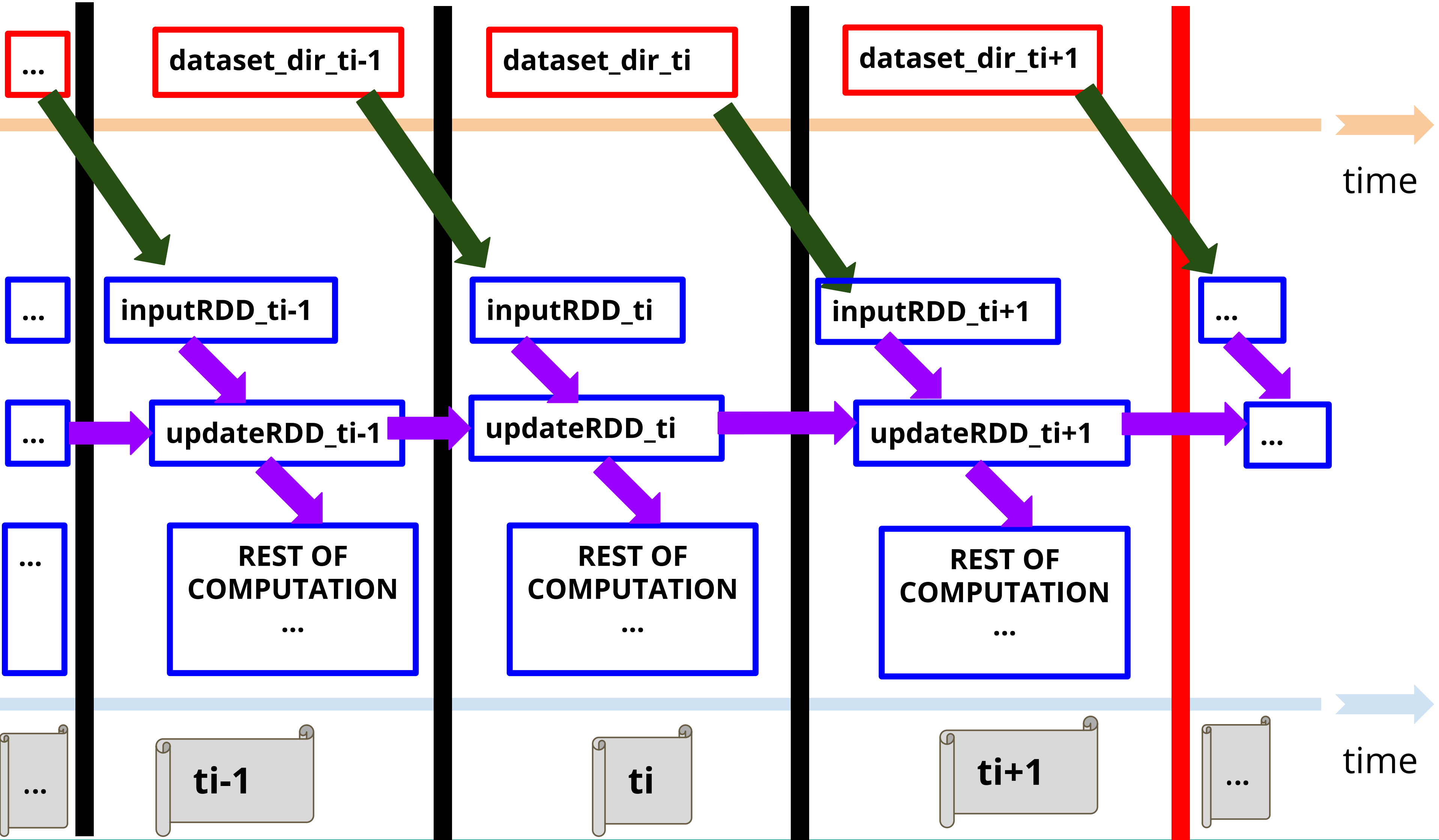
Stateless and Stateful Operations



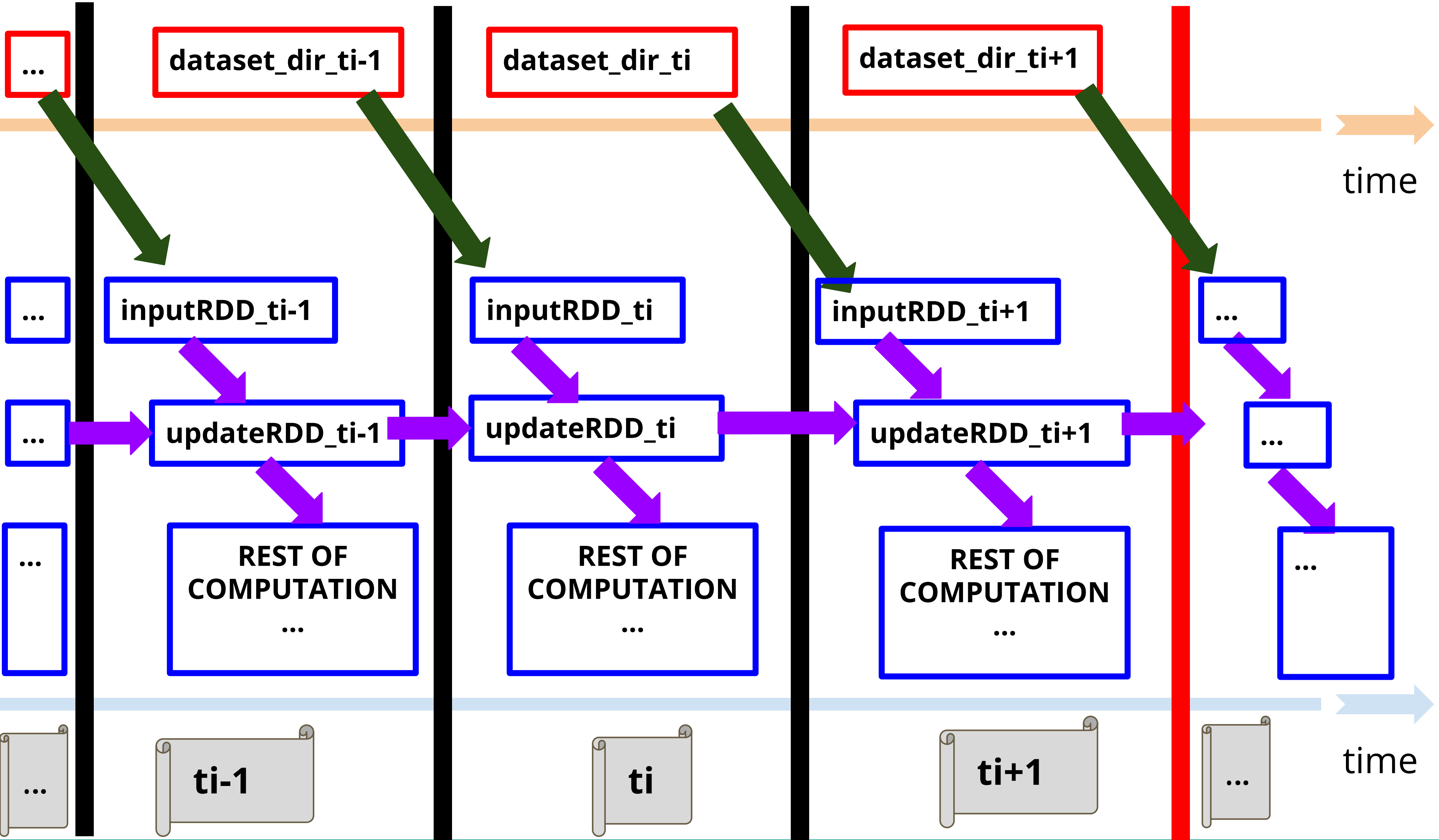
Stateless and Stateful Operations



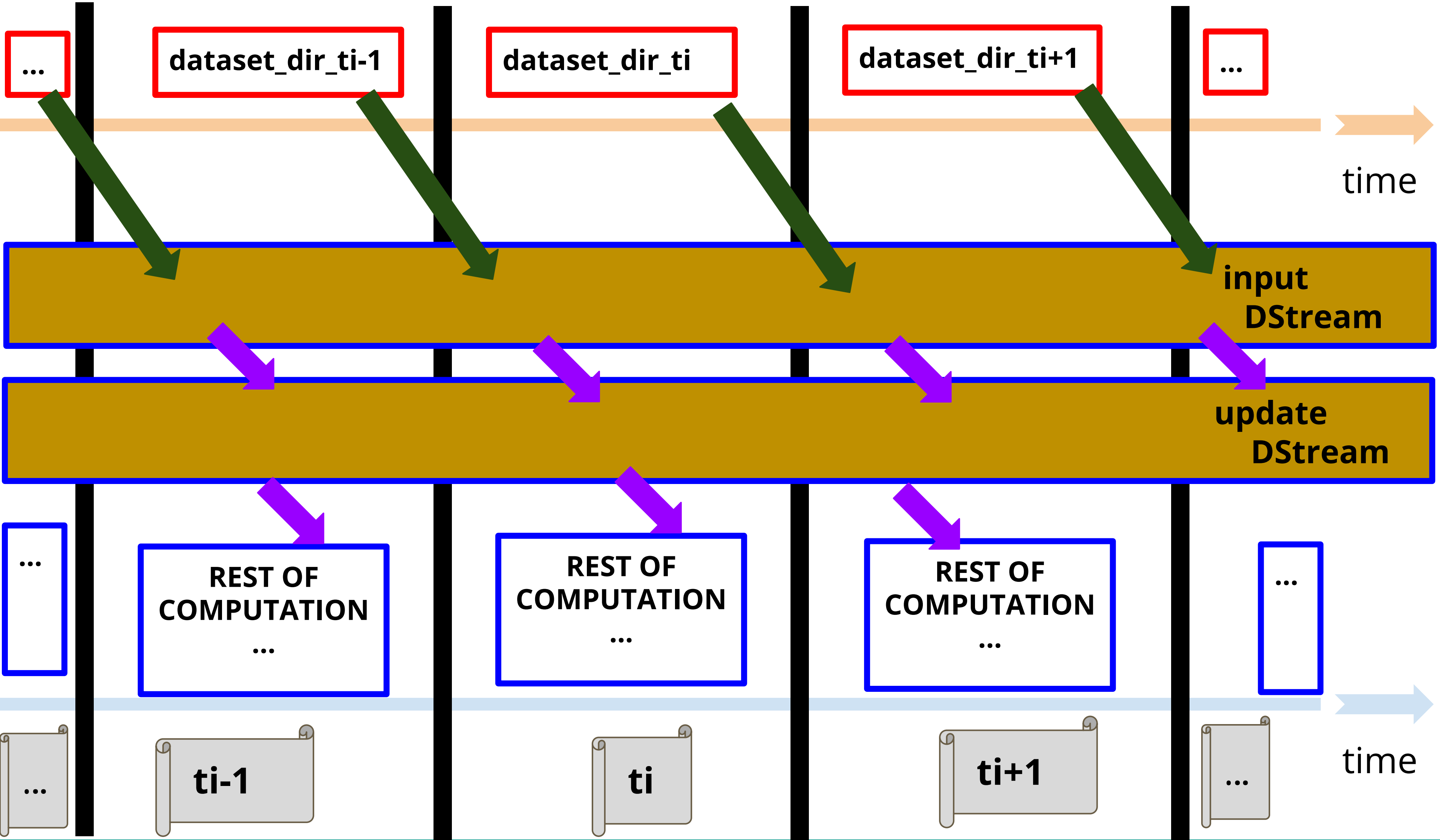
Stateless and Stateful Operations



Stateless and Stateful Operations



Stateless and Stateful Operations



Outline

1. Setting Up the Context.
2. Measurement Unit: Time Interval & Data Batch.
3. From RDDs to a DStream.
4. File Transfer Process.
5. Spark Streaming Context Process.
6. Stateless and Stateful Operations.

Thank you for your attention!