# Predictive Maintenance with Deep Learning and Flink

Dongwon Kim, PhD
Solution R&D center
SK Telecom

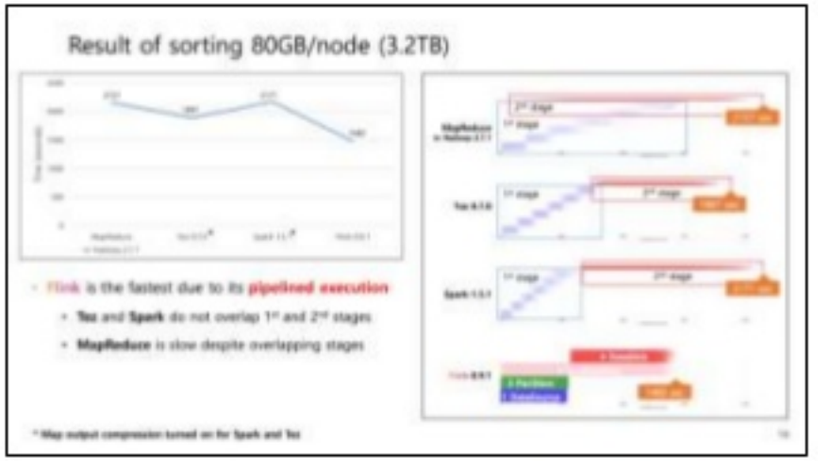
# About me (@eastcirclek)

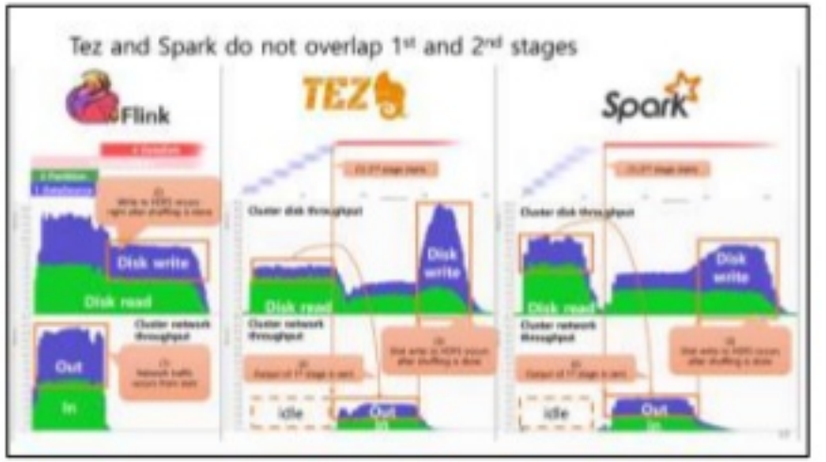
- Big Data processing engines
  - MapReduce, Tez, Spark, Flink, Hive, Presto

**Covered in this talk**

- Recent interest
  - Deep learning model serving (TensorFlow serving)
  - Containerization (Docker, Kubernetes)
  - Time-series data (InfluxDB, Prometheus, Grafana)

- Flink Forward 2015 Berlin
  - A Comparative Performance Evaluation of Flink



**Refinery** and **semiconductor** companies depend on **equipment**

SK energy

SK hynix

**Breakdown** **of equipment** largely affects company profit

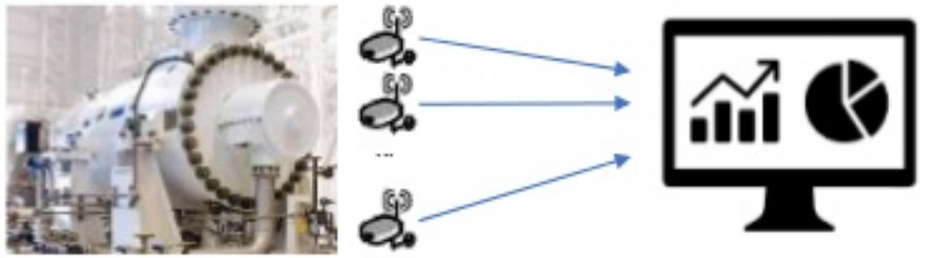
# **Equipment maintenance** to minimize **breakdown**


## Planned Maintenance

Shutdown equipment on a regular basis for parts replacement, cleaning, adjustments, etc.


## Predictive Maintenance (PdM)

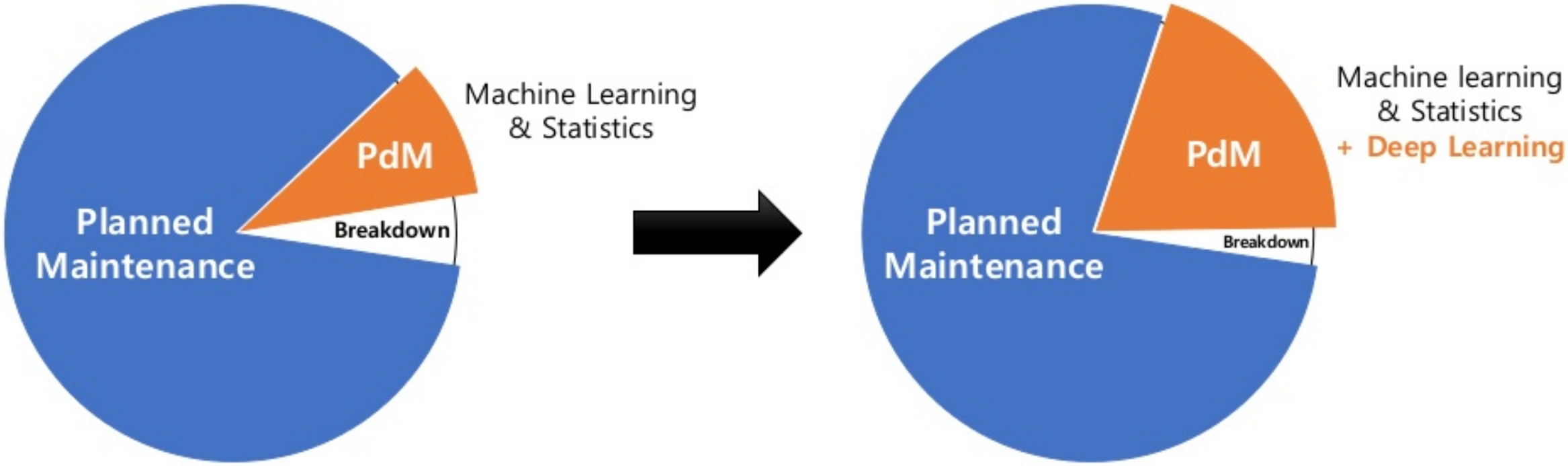
Unplanned maintenance based on
**prediction of equipment condition**



<equipment sensors> <predictive maintenance system>

# Our approach to **Predictive Maintenance**

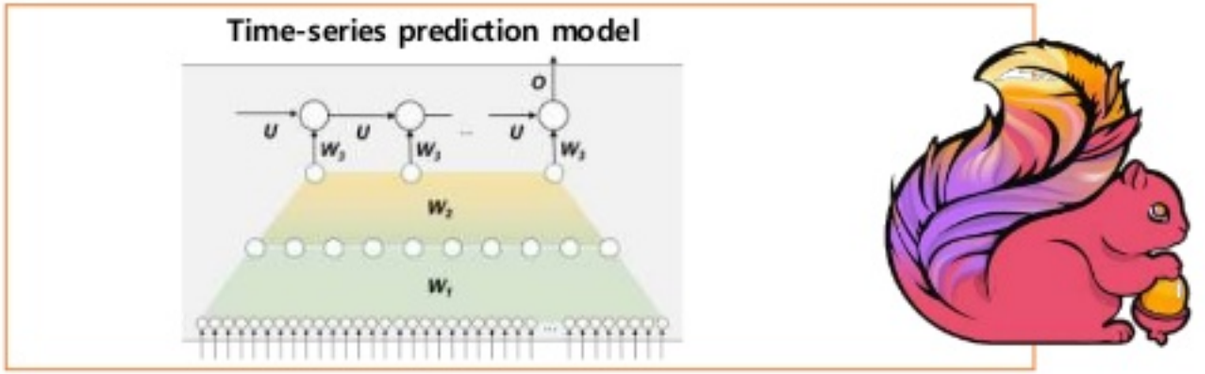
**Better prediction of equipment condition**
using **Deep Learning**



# Contents

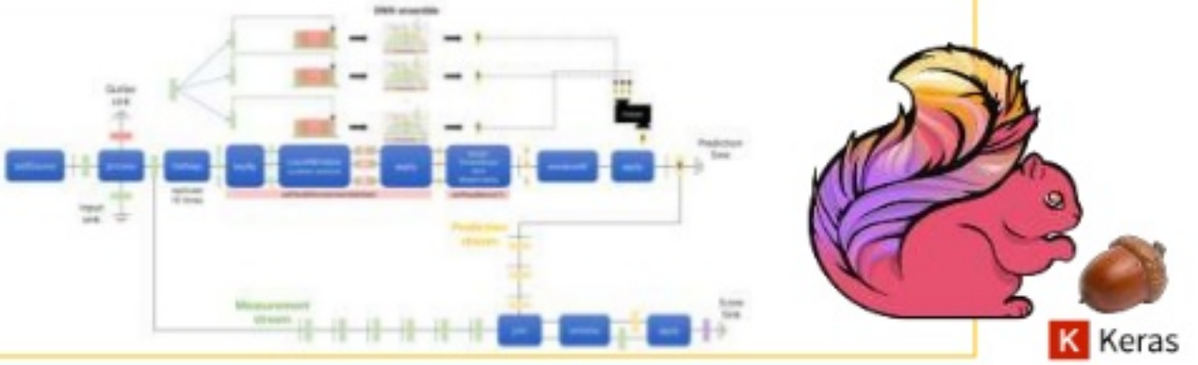
1

Why we use Flink
for our time-series prediction model



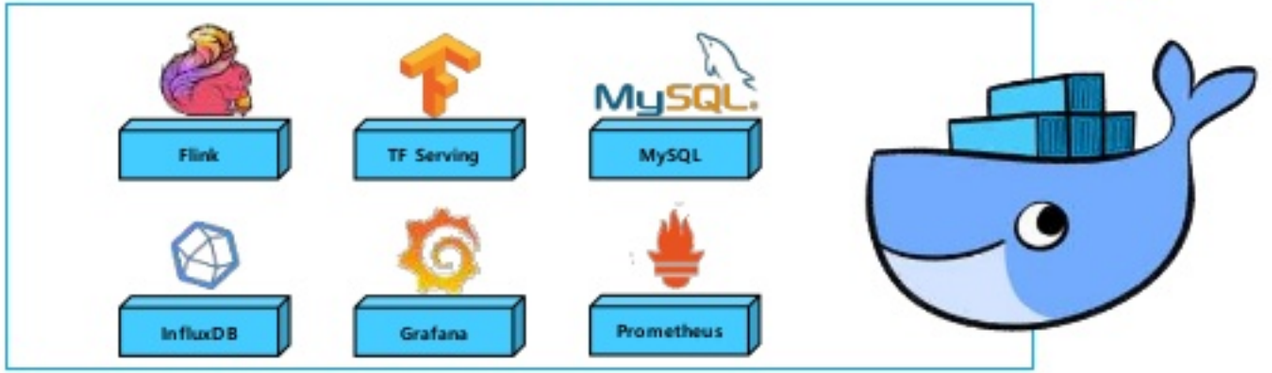

2

Flink pipeline design for
**rendezvous** and **DNN ensemble**

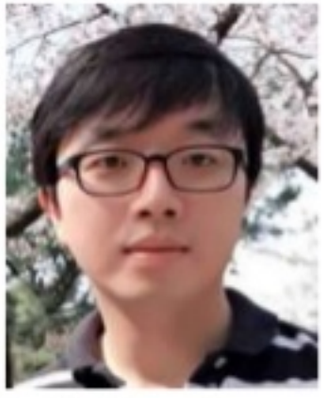



3

Solution packaging and monitoring
with **Docker** and **Prometheus**



# My team consists of two groups

| | Role | Toolbox |
|---|---|---|
|  **Model Developers** | history data → training → DNN model * | Keras, TensorFlow, python |
|  **Data Engineers** | live data → DNN model → prediction & alarm * | Apache Spark, Java |

* The diagram is based on Ted Dunning's slides (Flink Forward 2017 SF)

# **Model developers** give **Convolutional LSTM** to engineers

**Output (Ŷ)**
expected sensor values
after 2 days

Ŷ

O

RNN
(LSTM)

U U U

$W_3$ $W_3$ $W_3$

$W_2$

CNN

$W_1$

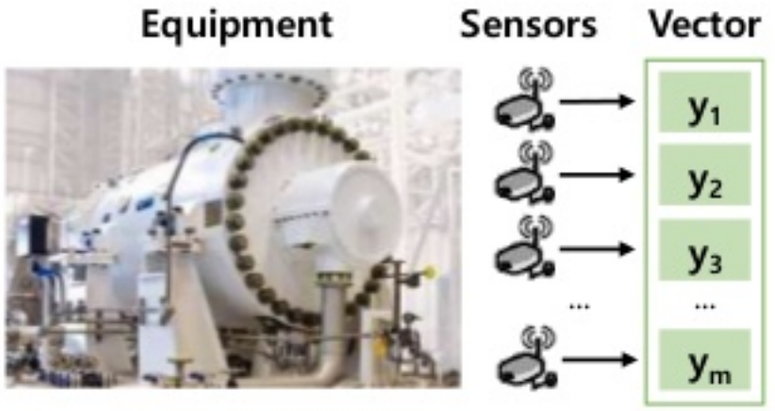
**Input (X)**
1 week time-series (10080 records)

* assuming one minute sampling rate

It **does not return** whether the equipment breaks down after 2 days

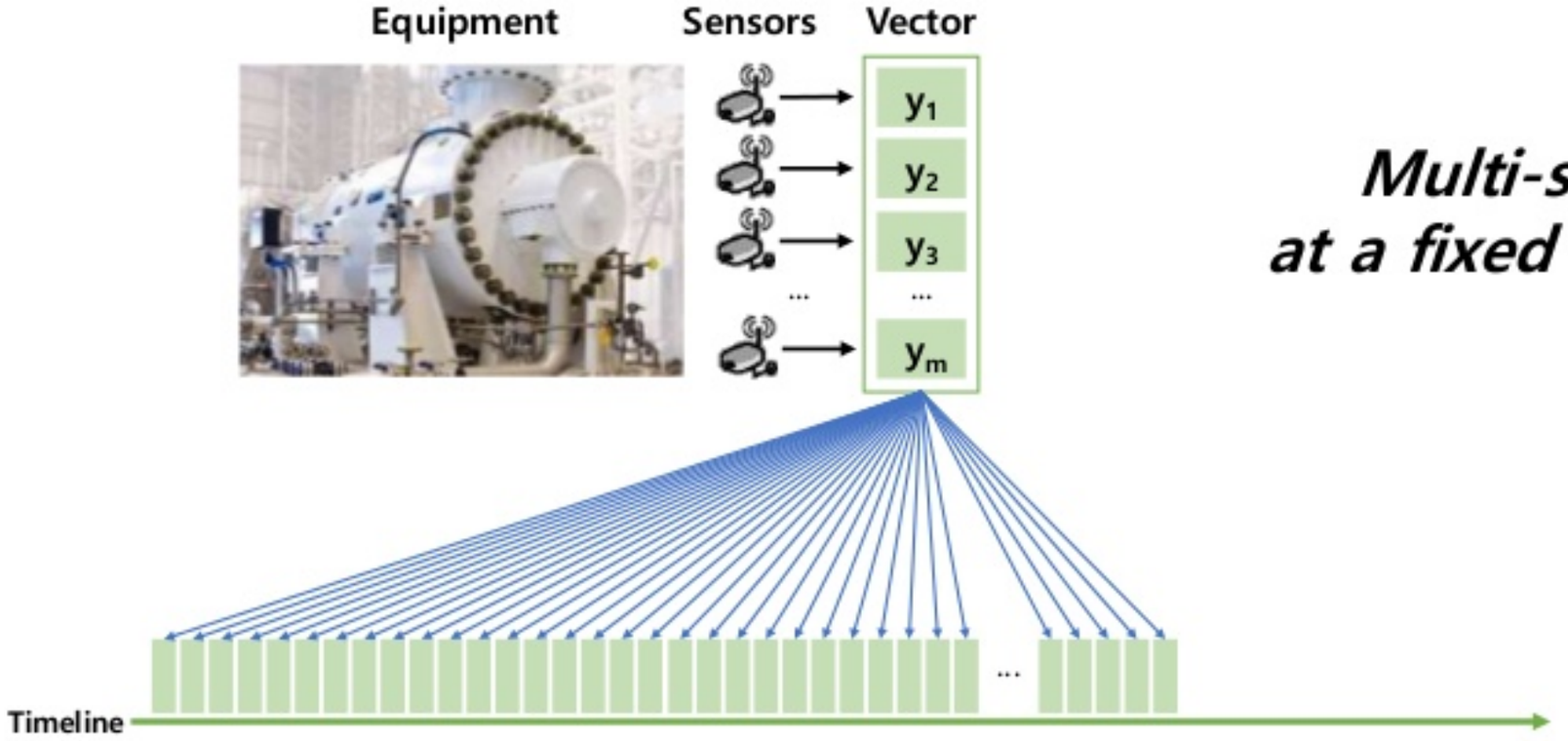
**Model developer**

# **Data engineers** apply **Convolutional LSTM** to live sensor data



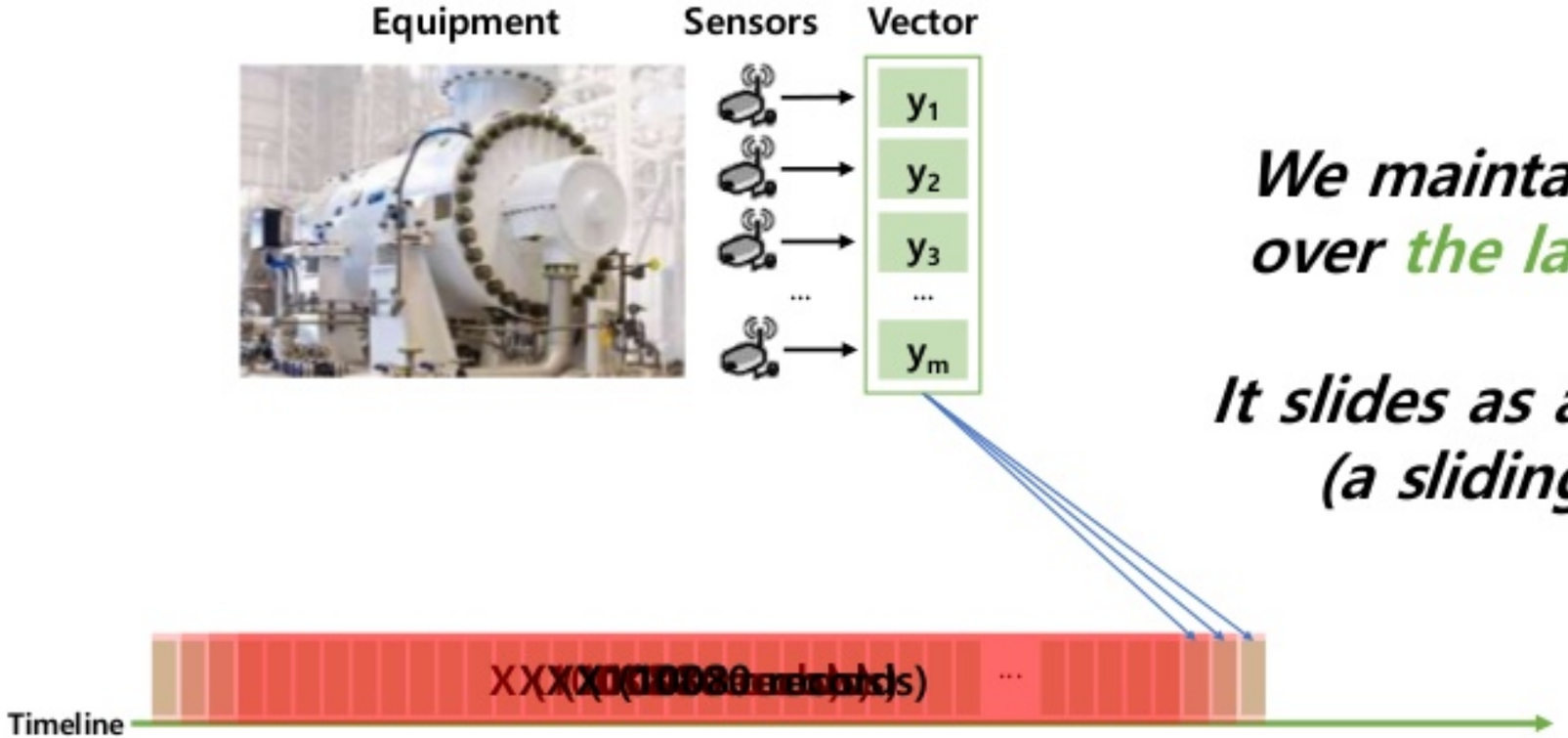

***We have multi-sensor data***

# **Data engineers** apply **Convolutional LSTM** to live sensor data




***Multi-sensor data arrive at a fixed interval of 1 minute***

# **Data engineers** apply **Convolutional LSTM** to live sensor data




***We maintain a count window over the latest 10080 records***

***It slides as a new record arrives (a sliding count window)***

# **Data engineers** apply Convolutional LSTM to live sensor data

RNN (LSTM)

CNN

$\hat{Y}$

$O$

$U$ $W_3$ $U$ $W_3$ ... $U$ $W_3$

$W_2$

$W_1$

$\hat{Y}$ $\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ ... $\hat{y}_m$

Given one week time-series (X), DNN returns predicted values after 2 days ($\hat{Y}$)

Raise an alarm if the distance of **two vectors** is above **a defined threshold**

2 days

X (10080 records)

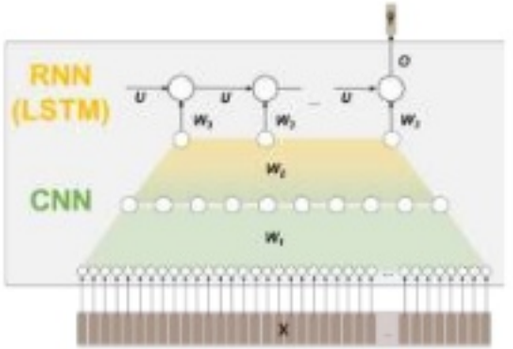
Y $y_1$ $y_2$ $y_3$ ... $y_m$

Timeline

***Whenever the sliding window moves, we apply Convolutional LSTM to*** ...

# Desired streaming topology by **data engineers**

**Requirement 1**
Count window to maintain
10080 records instead of
1 week event-time window




timeline

Stream source

Outlier filter

Sliding count window

X

Apply **DNN** to **X**

Ŷ

Prediction stream

**Requirement 2**
Joining of two streams
on event time
(Rendezvous)

Measurement stream

Y

Join

Score

Sink

# Proof-of-Concept of Spark Structured Streaming


**Types**

```
case class Input(timestamp: Timestamp, measured: Array[Float])
case class Prediction(timestamp: Timestamp, predicted: Array[Float])
case class Score(timestamp: Timestamp, alarm: Boolean)
```

**Input Streaming Dataset**

```
val input: Dataset[Input] =
  spark
    .sqlContext
    .readStream
    .option("maxFilesPerTrigger", 1)
    .textFile("data/")
    .withColumn("value", udf(parseRecord).apply($"value"))
    .select("value.*")
    .as[Input]
```

generate an input stream from local files

**Prediction Streaming Dataset**

```
val prediction: Dataset[Prediction] =
  input
    .groupBy(window($"timestamp", windowSize, triggerInterval))
    .agg(applyDNN.toColumn.name("predict"))
    .select("predict.*")
    .as[Prediction]
```

apply DNN to 1 week time-series, not 10080 records
(Sliding count window is not supported)

**Score Streaming Dataset**

```
val score: Dataset[Score] =
  input
    .join(prediction, "timestamp")
    .withColumn("score", udf(computeScore).apply($"timestamp", $"measured", $"predicted"))
    .select("score.*")
    .as[Score]
```

joining of two streams

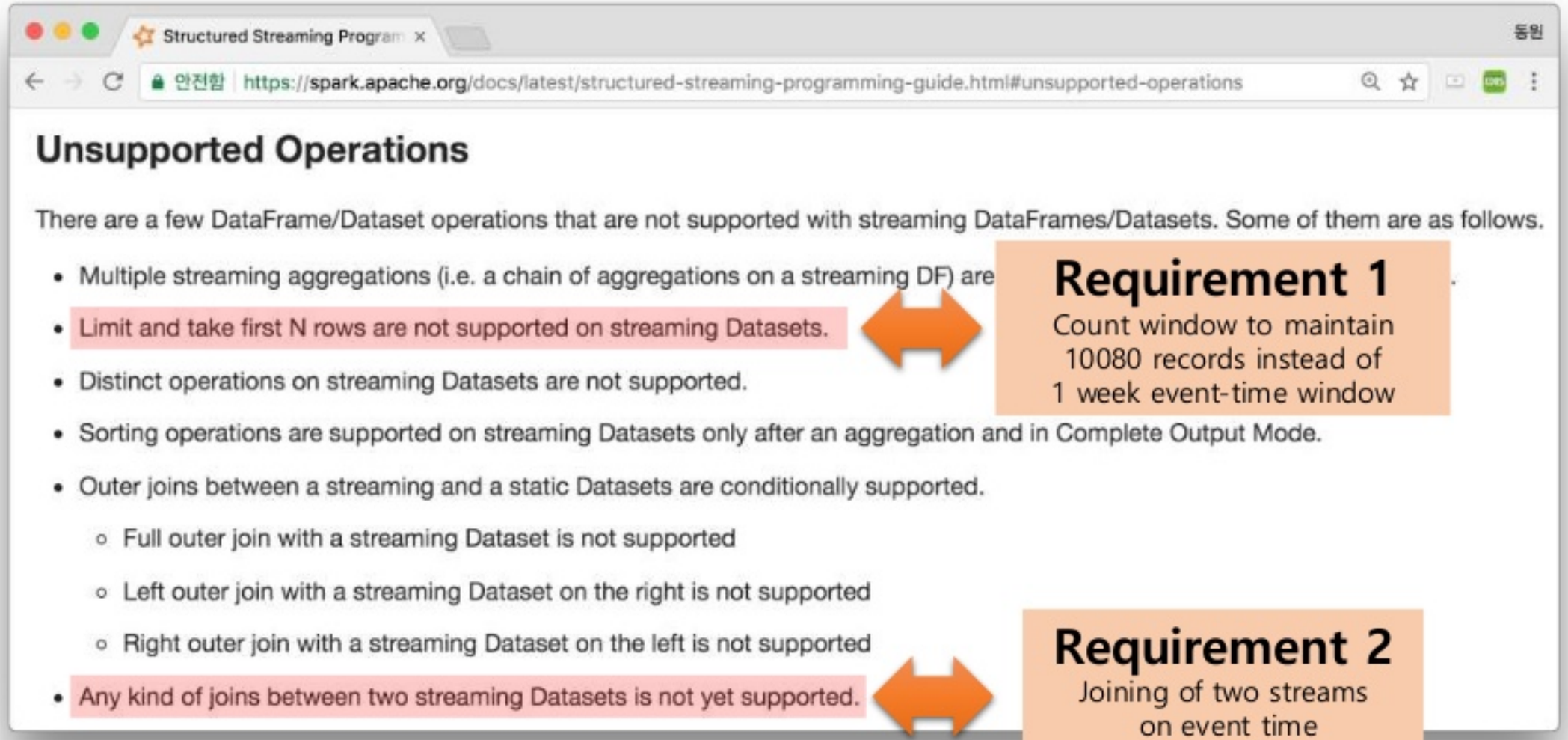
# Inner join between two streaming Datasets is not supported in Spark Structured Streaming

**Score Dataset**

```scala
val score: Dataset[Score] =
  input
    .join(prediction, "timestamp")
    .withColumn("score", udf(computeScore).apply($"timestamp", $"measured", $"predicted"))
    .select("score.*")
    .as[Score]
```

```
Exception in thread "main" org.apache.spark.sql.AnalysisException: Inner join between two streaming DataFrames/Datasets is not supported;;
Join Inner, (timestamp#10 = timestamp#33)
:- Project [value#7.timestamp AS timestamp#10, value#7.measured AS measured#11]
:  +- Project [UDF(value#0) AS value#7]
:     +- Project [value#0]
:        +- StreamingRelation DataSource(org.apache.spark.sql.SparkSession@3214bad,text,List(),None,List(),None,Map(maxFilesPerTrigger -> 1, path -> data/),None), FileSource[data/], [value#0]
+- Project [predict#28.timestamp AS timestamp#33, predict#28.predicted AS predicted#34]
   +- Aggregate [window#29], [window#29 AS window#18, predictor(Predictor@74a58a06, Some(createexternalrow(staticinvoke(class org.apache.spark.sql.catalyst.util.DateTimeUtils$, ObjectType(class java.sql.Tim
      +- Filter ((timestamp#10 >= window#29.start) && (timestamp#10 < window#29.end))
         +- Expand [ArrayBuffer(named_struct(start, ((((CEIL((cast((precisetimestamp(timestamp#10) - 0) as double) / cast(60000000 as double))) + cast(0 as bigint)) - cast(10 as bigint)) * 60000000) + 0), e
            +- Project [value#7.timestamp AS timestamp#10, value#7.measured AS measured#11]
               +- Project [UDF(value#0) AS value#7]
                  +- Project [value#0]
                     +- StreamingRelation DataSource(org.apache.spark.sql.SparkSession@3214bad,text,List(),None,List(),None,Map(maxFilesPerTrigger -> 1, path -> data/),None), FileSource[data/], [value#0]
```

# Unsupported Operations on Spark Structured Streaming (v2.2)



# That's why we move to Flink DataStream API


## Spark Structured Streaming

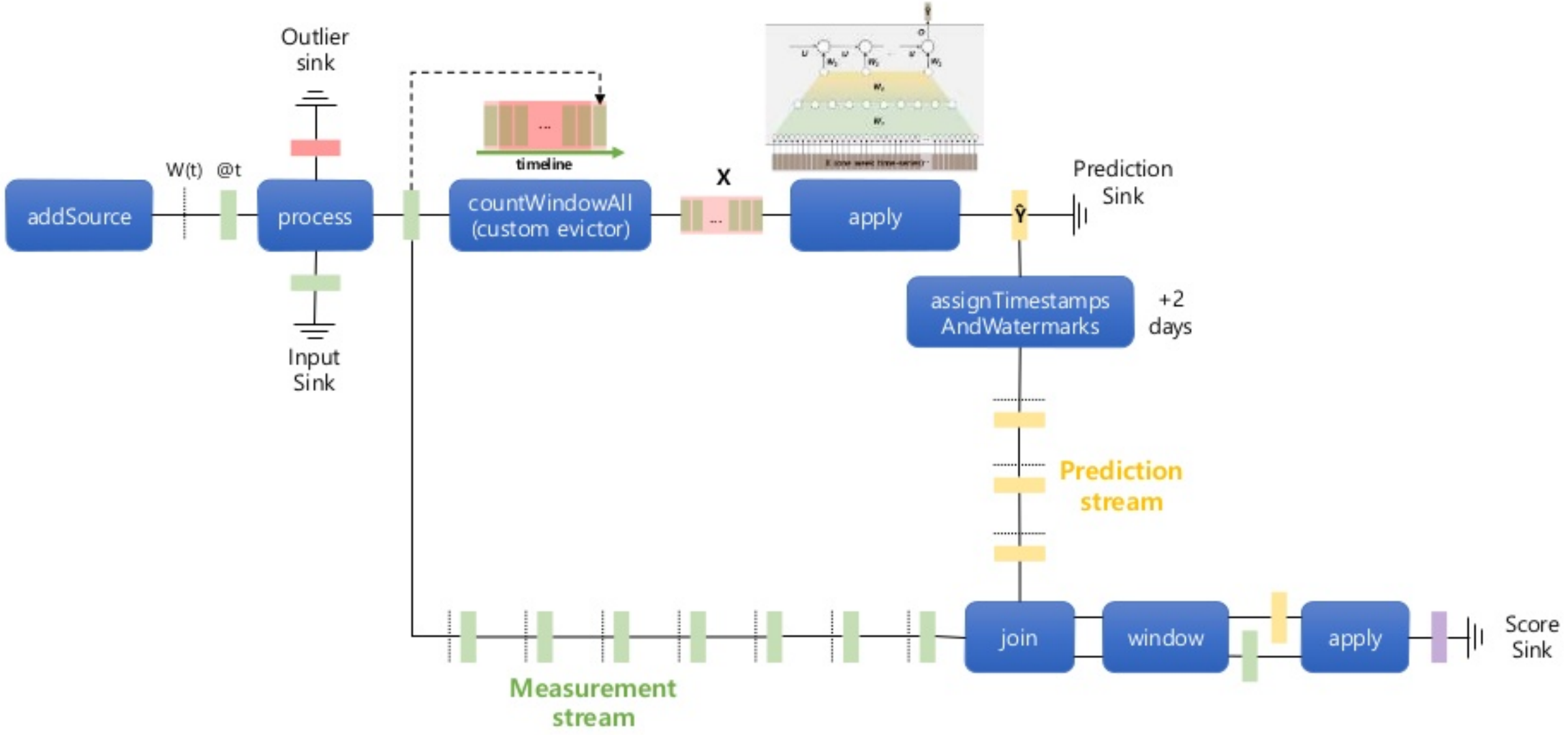
- **Sliding count window** : **not supported**
- **Joining of two streams** : **not supported**
- Micro-batch behind the scene
  - Continuous processing proposed in SPARK-20928

* it could be possible to use our Convolutional LSTM model using Spark Structured Stream in some other way

## Flink DataStream API

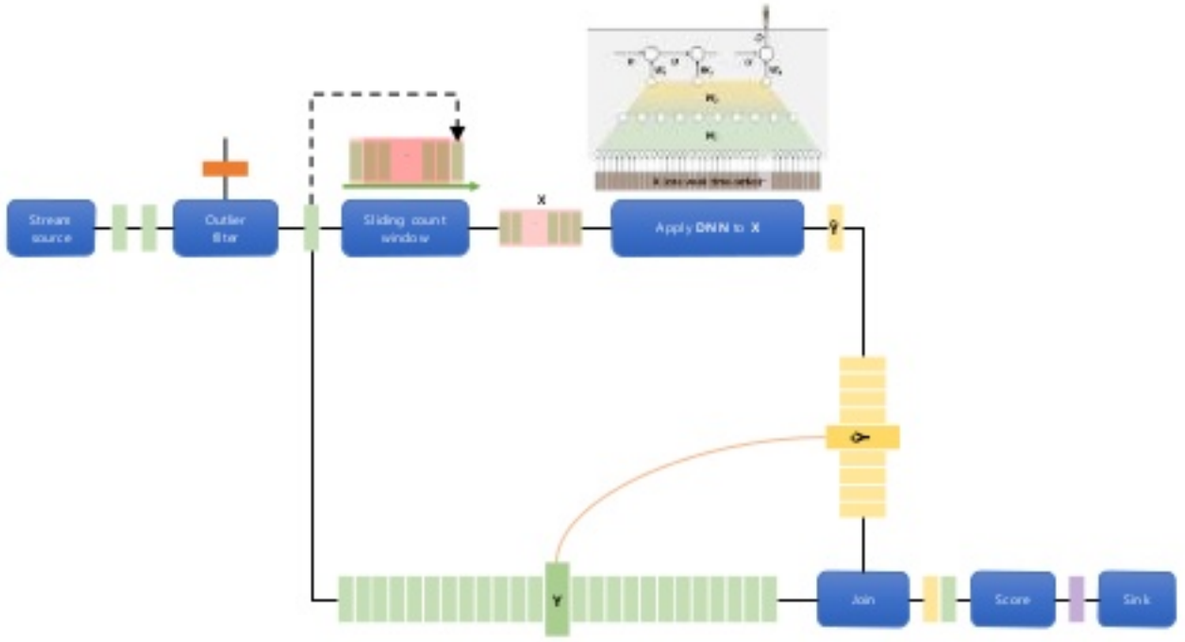
- **Sliding count window** : **supported**
- **Joining of two streams** : **supported**
- Scalability and performance proved by other use cases

# Data processing pipeline with **Flink DataStream API**

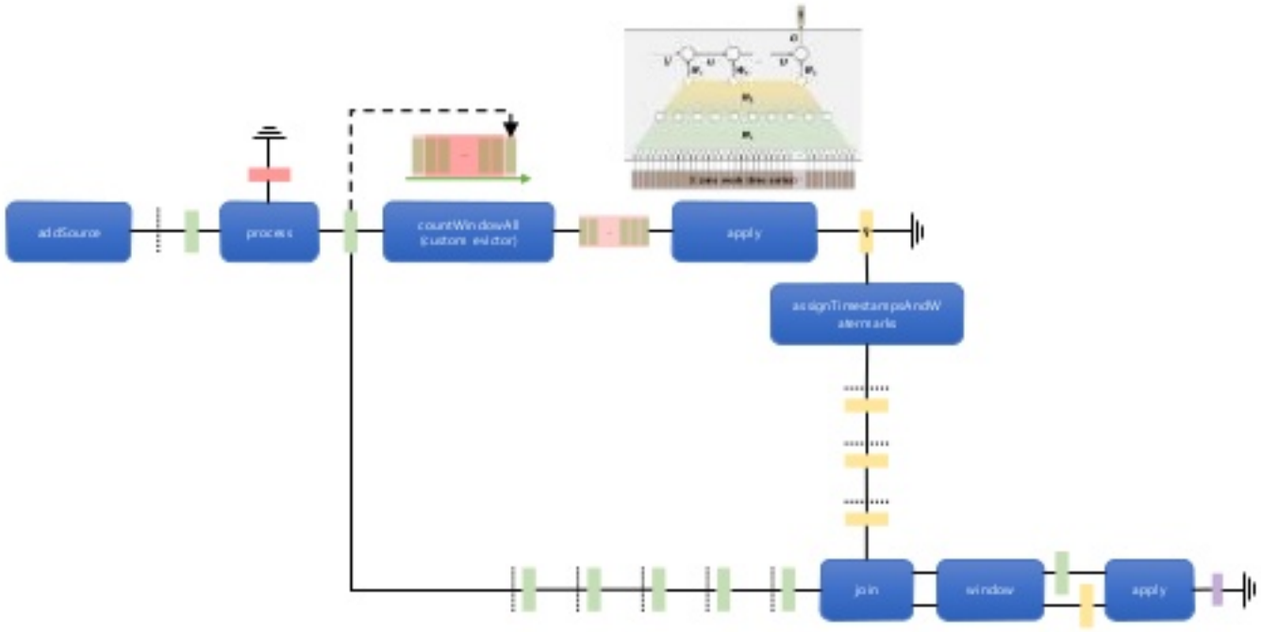


# Flink can faithfully implement our streaming topology design

## <Topology design>



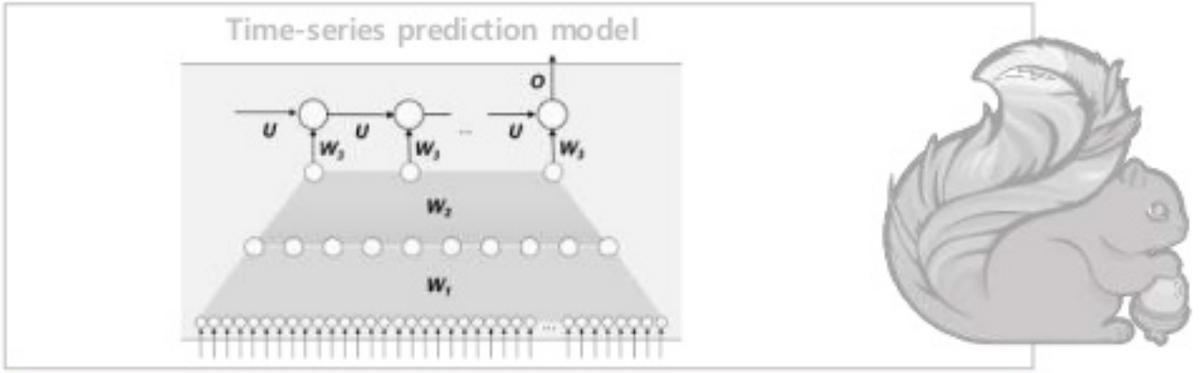

## <Flink implementation>



# Contents

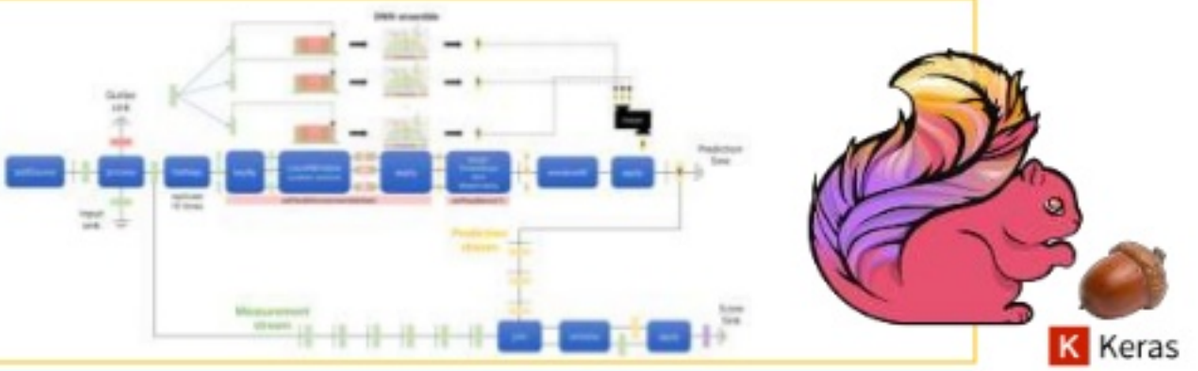
1

Why we use Flink
for our time-series prediction model



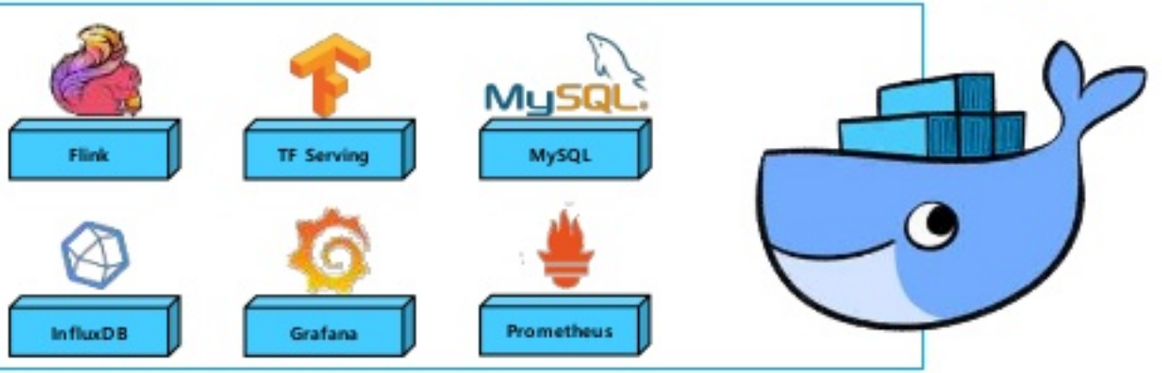

2

Flink pipeline design for
**rendezvous** and **DNN ensemble**

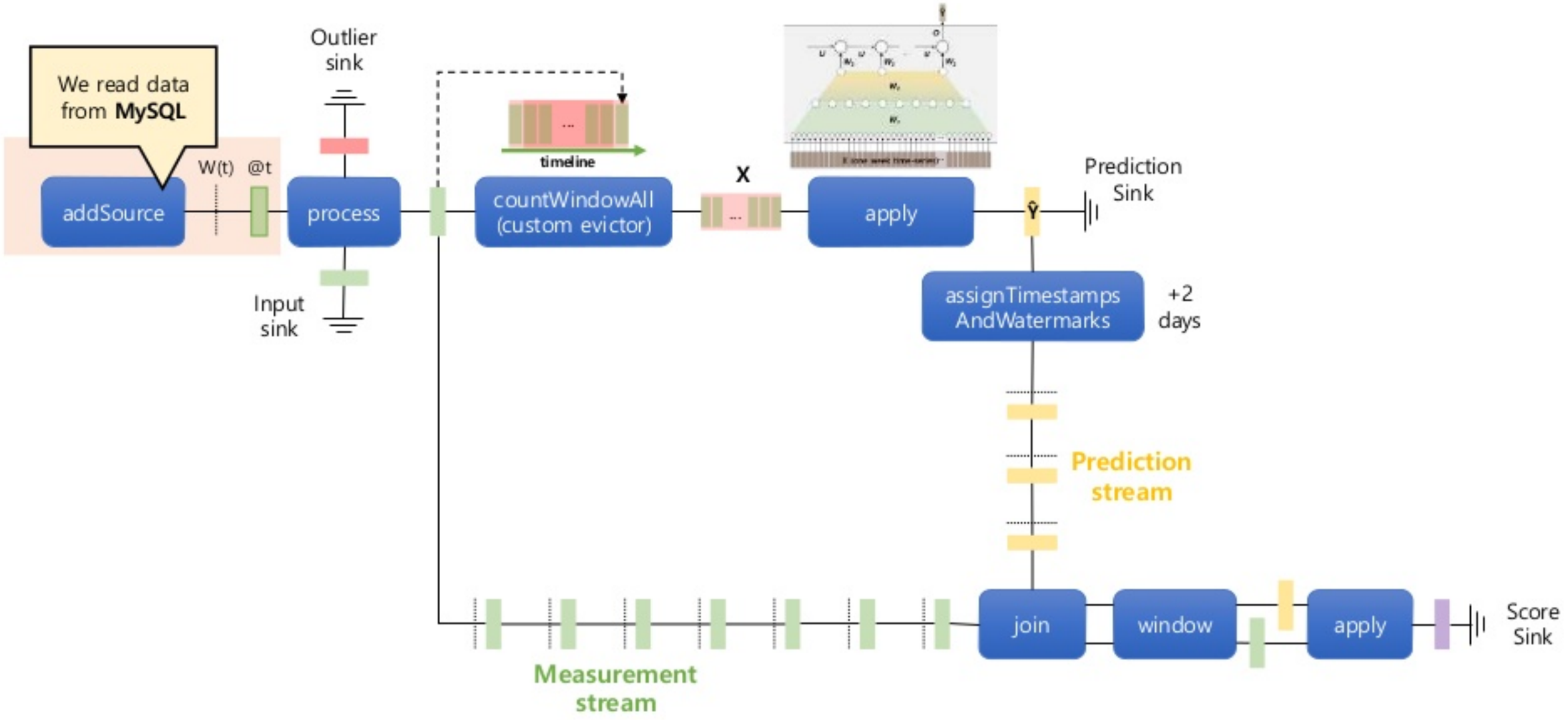



3

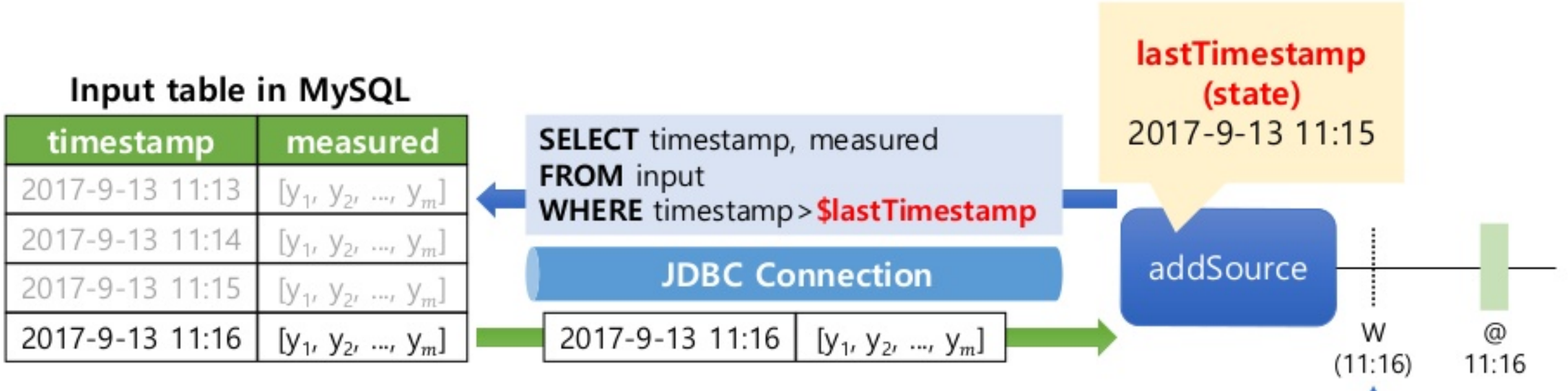
Solution packaging and monitoring
with **Docker** and **Prometheus**



# Data processing pipeline with **Flink DataStream API**

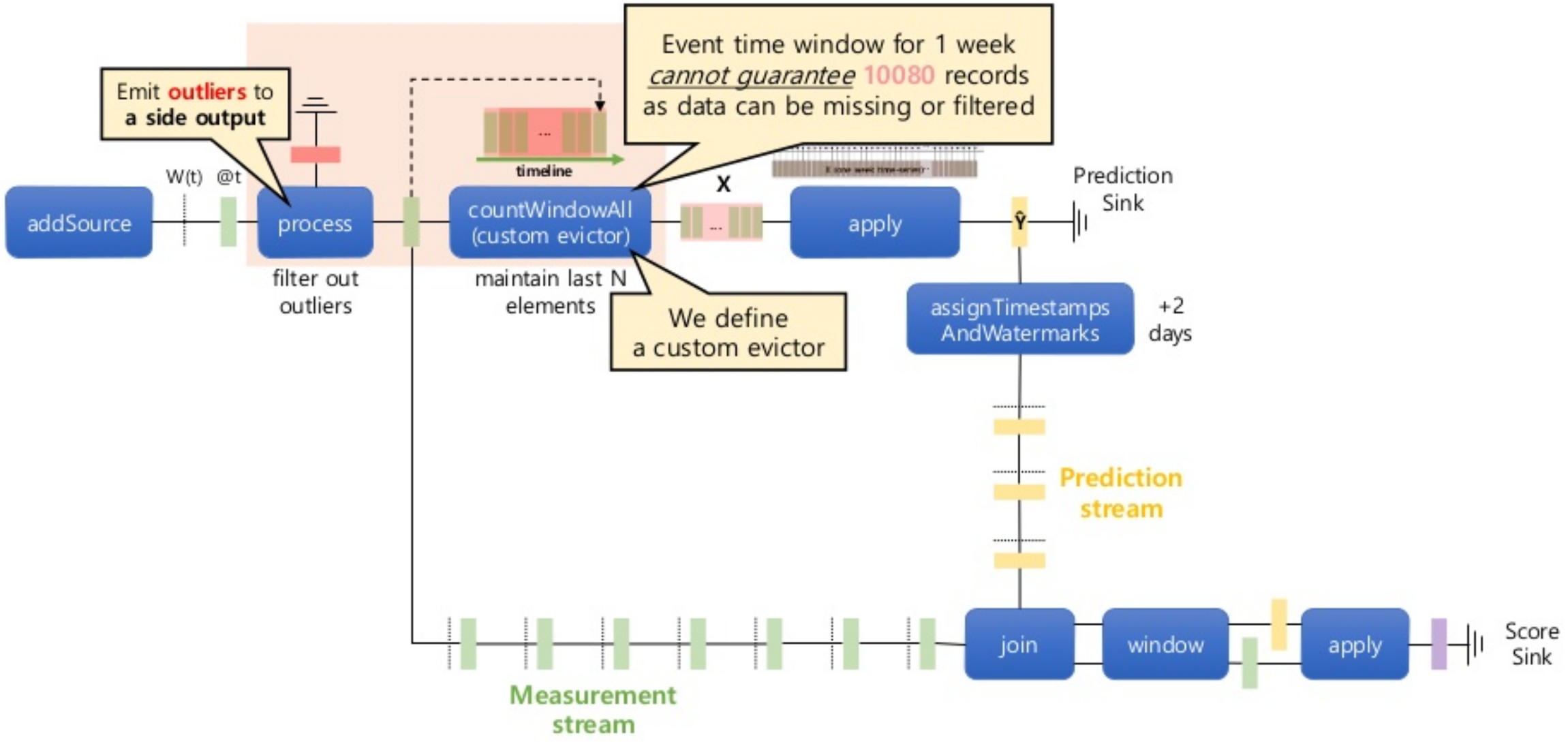


# **Stateful** custom source to read from MySQL

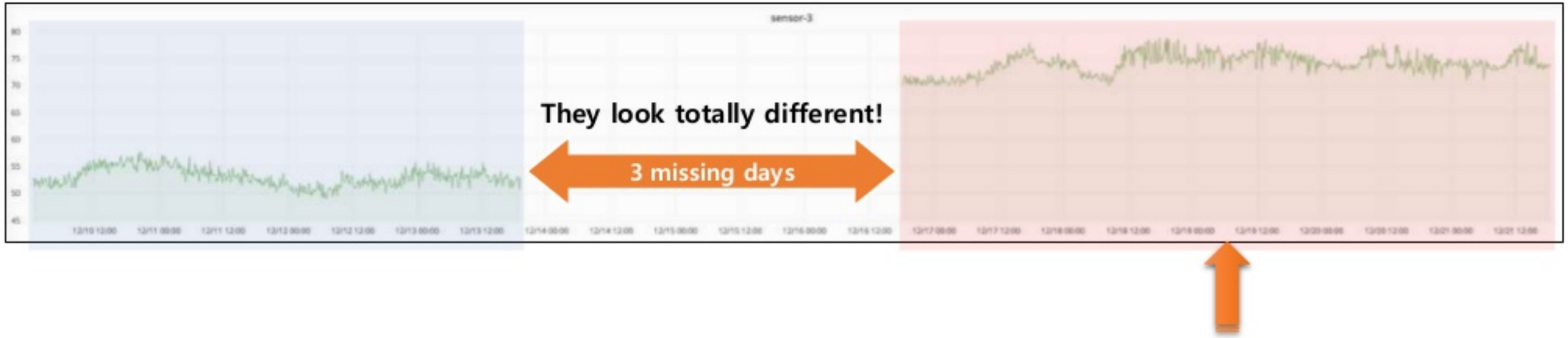



- We assume that sensor data arrive **in order**
  - Emit an input record and a watermark of the same time
  - Increase **lastTimestamp** afterward (11:15 → 11:16)
- Exactly-once semantics
  - Store **lastTimestamp** when taking a snapshot
  - Restore **lastTimestamp** when restarted

# Data processing pipeline with **Flink DataStream API**

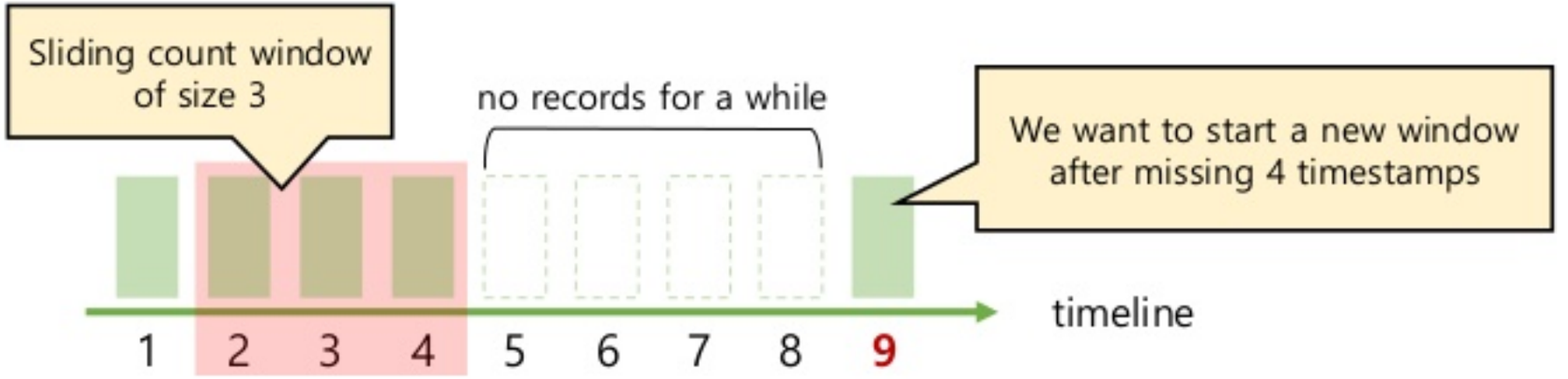


# What if data is **absent** or **filtered** for a long period of time?




**We'd better start a new sliding window for the time-series!**

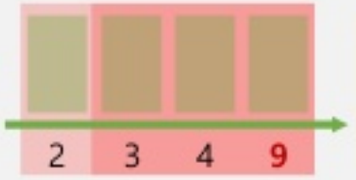
# How to start a new sliding count window after a long break




**EvictingWindowOperator**
adds a new input record to
InternalListState

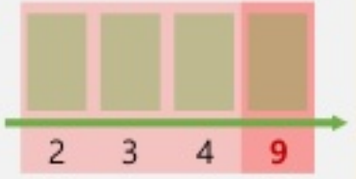

**CountTrigger.of(1)**
fires every time
a record comes in

**CountEvictor.of(3)**



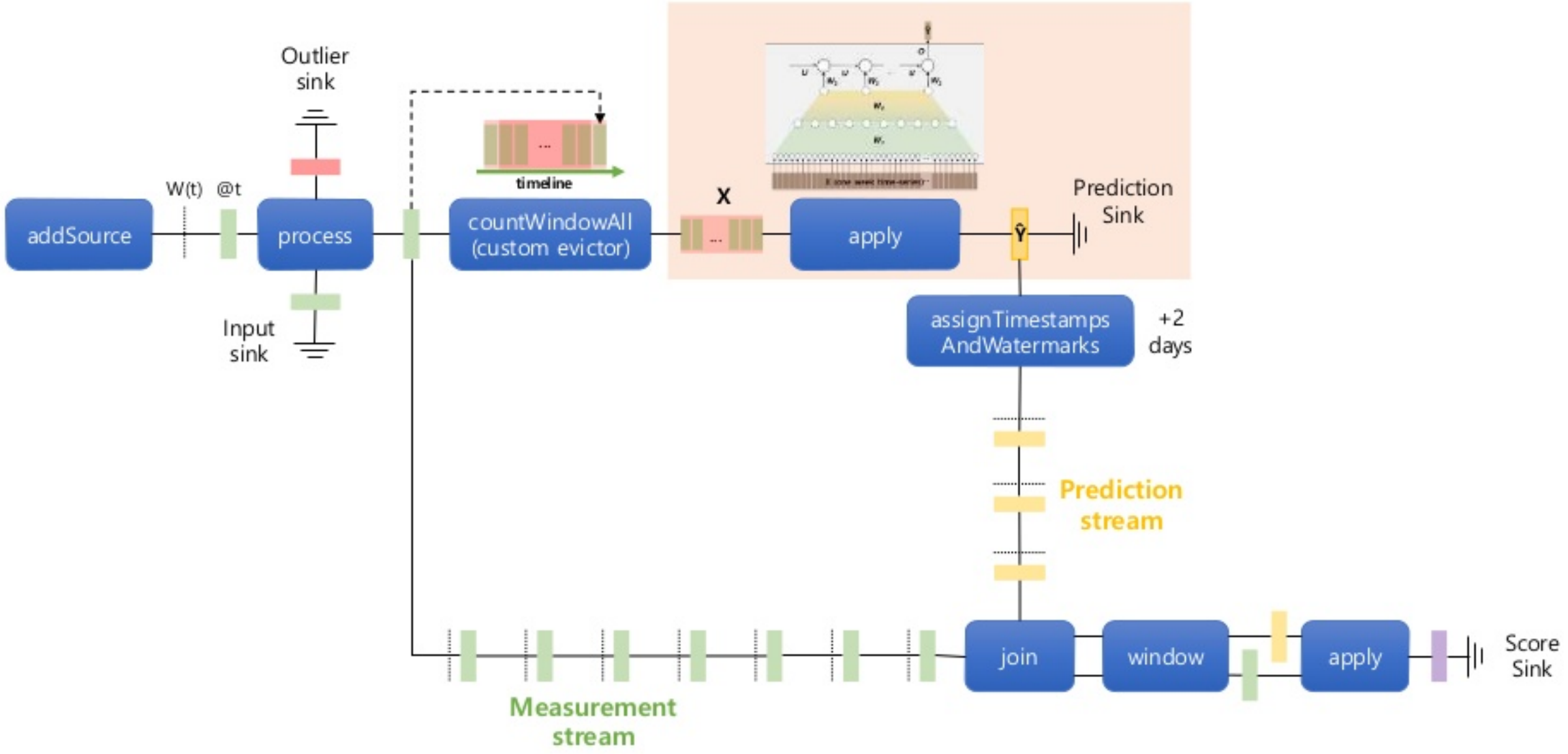

**CountEvictor** evicts
elements beyond
its capacity

**CustomEvictor.of(3, timeThreshold=4)**




**CustomEvictor** evicts
all but the last one
when the last one occurs
after **timeThreshold**

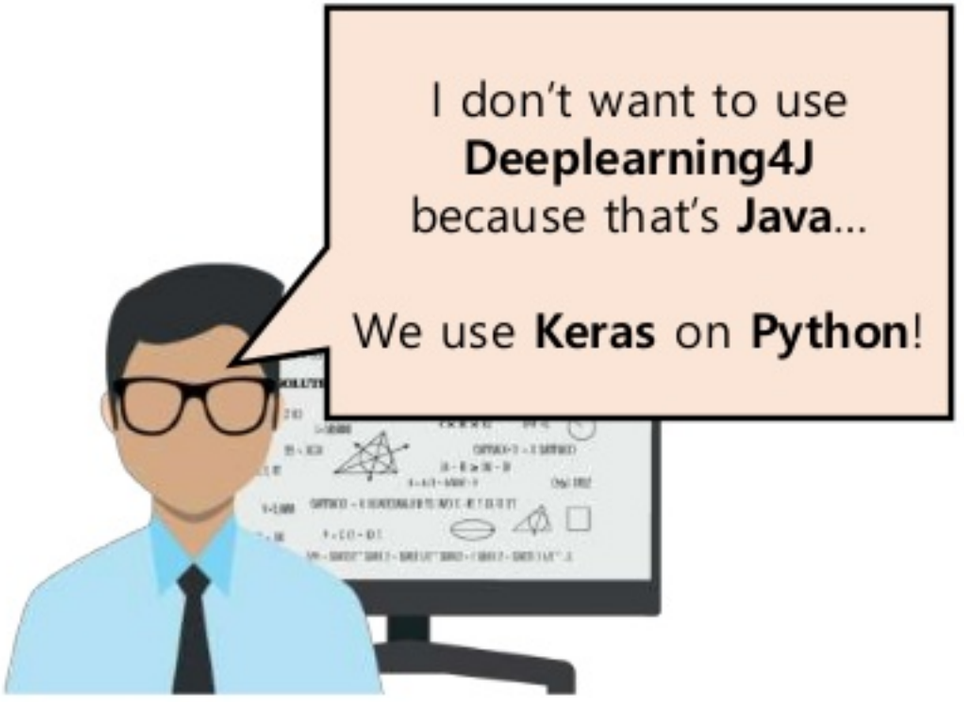
# Data processing pipeline with **Flink DataStream API**



# Working with model developers

I want to develop our solution on **JVM!**

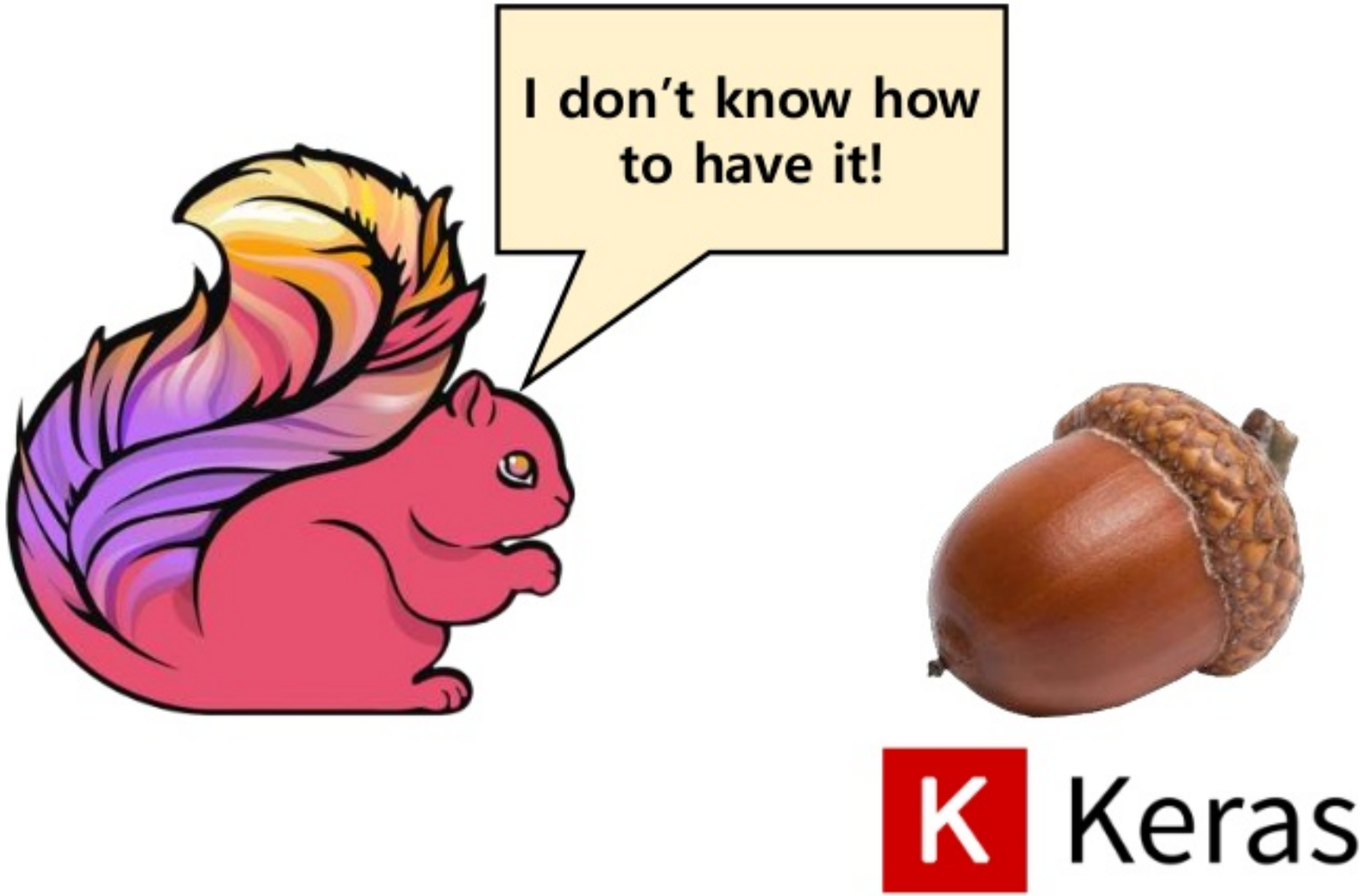
Why don't we develop models using **Deeplearning4J?**




They stick to using **Python**

They develop models using a Python library called K Keras

# How to load **Keras** models in **Flink**?



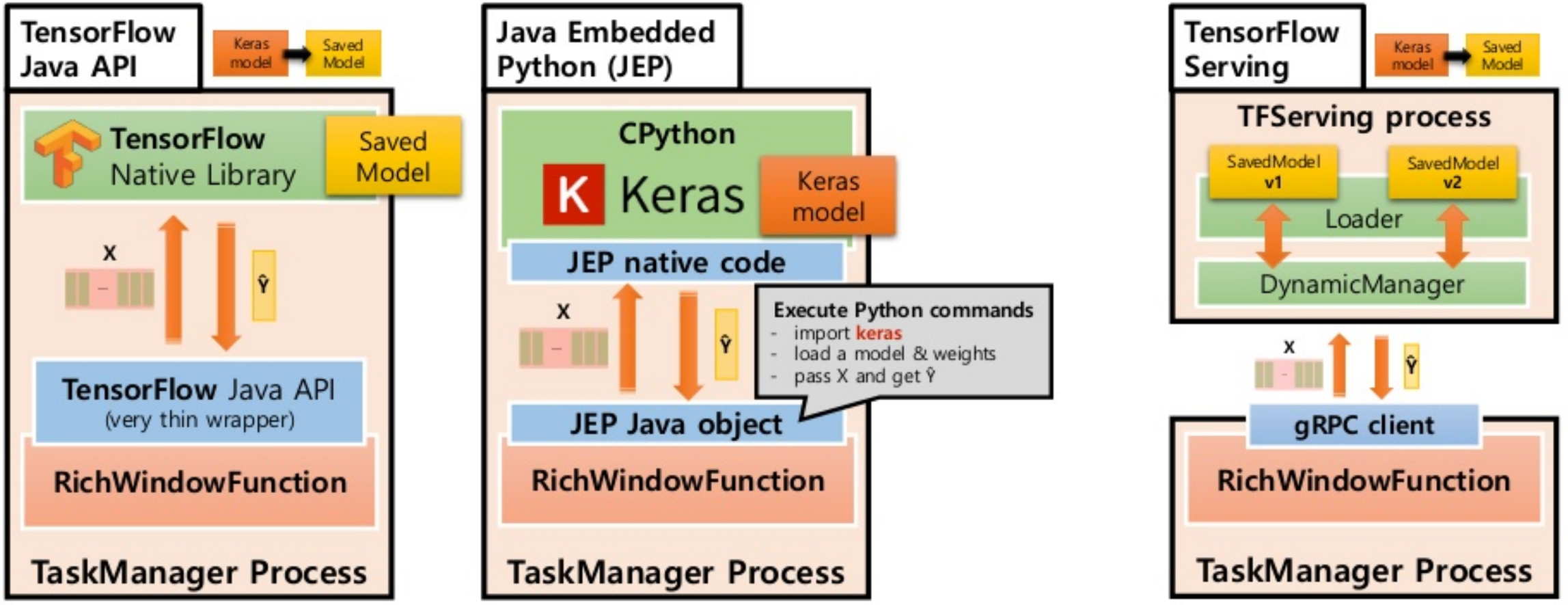
# Loading **Keras** models in JVM

- Convert **Keras** models to **TensorFlow SavedModel**
  - use tensorflow.python.saved_model.builder.**SavedModelBuilder**
  - **TensorFlow Java API** (Flink TensorFlow)
    - Do inference **inside** the JVM process
  - **TensorFlow Serving**
    - Do inference **outside** the JVM process

- Execute **Keras** through CPython inside JVM
  - Do inference **inside** the JVM process
  - **Java Embedded Python (JEP)** to ease the use of **CPython**
    - https://github.com/ninia/jep

- Use **KerasModelImport** from **Deeplearning4J**
  - Not mature enough

# Comparison of approaches to use **Keras** models in **JVM**



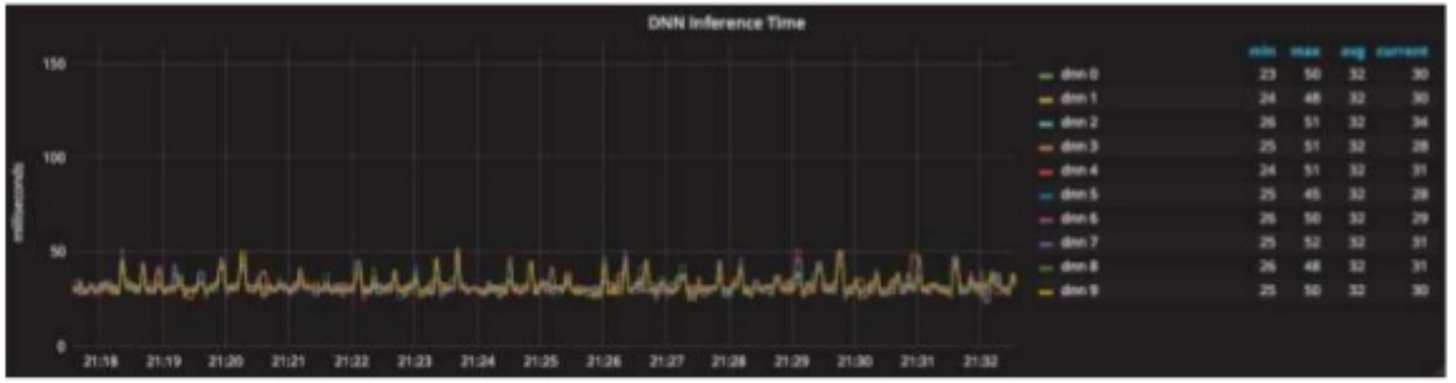
# Comparison of runtime inference performance

(* We do not batch inference calls)

**Keras inside CPython w/ TensorFlow backend**

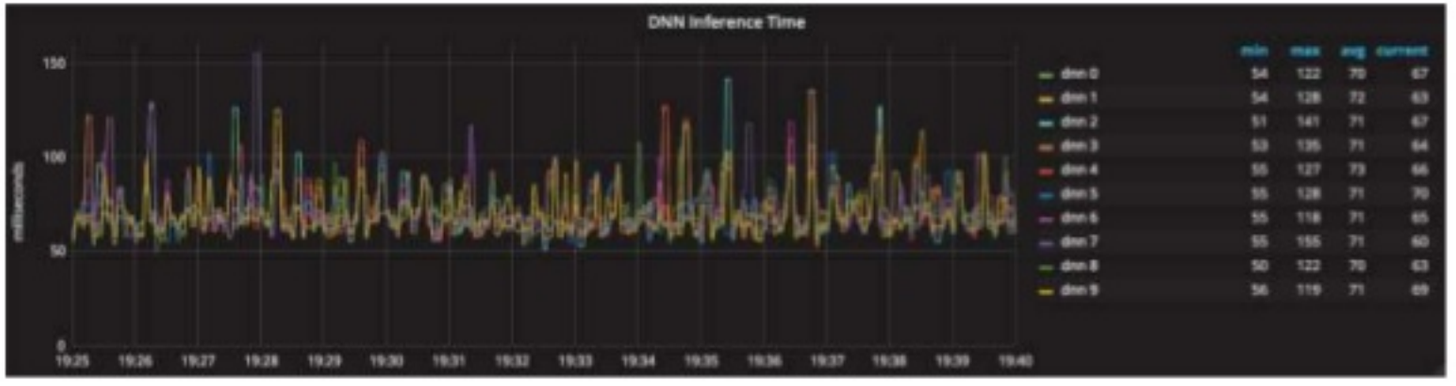
32 milliseconds per inference

(* Theano backend is extremely slow in our case)




**TensorFlow Serving**

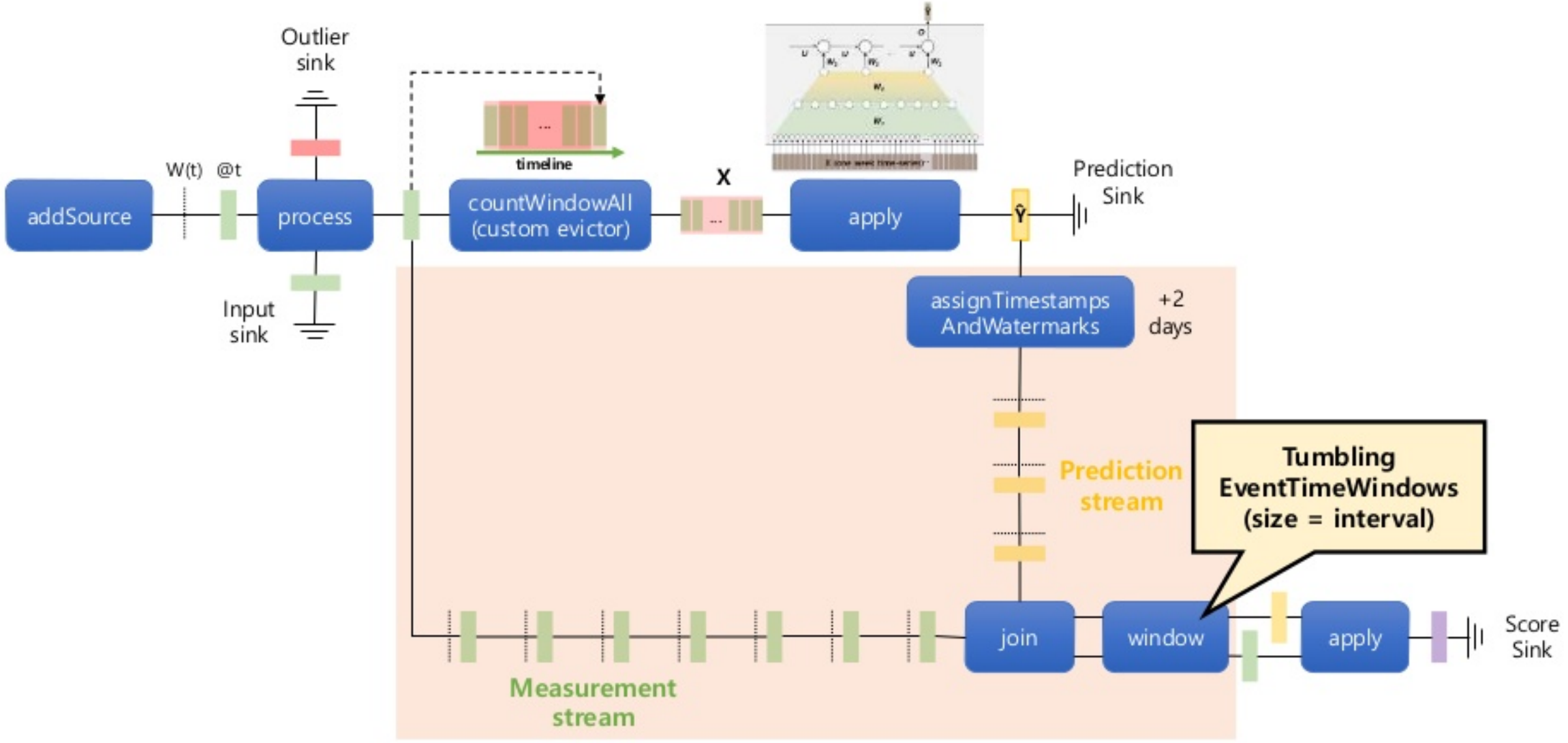
71.2 milliseconds per inference



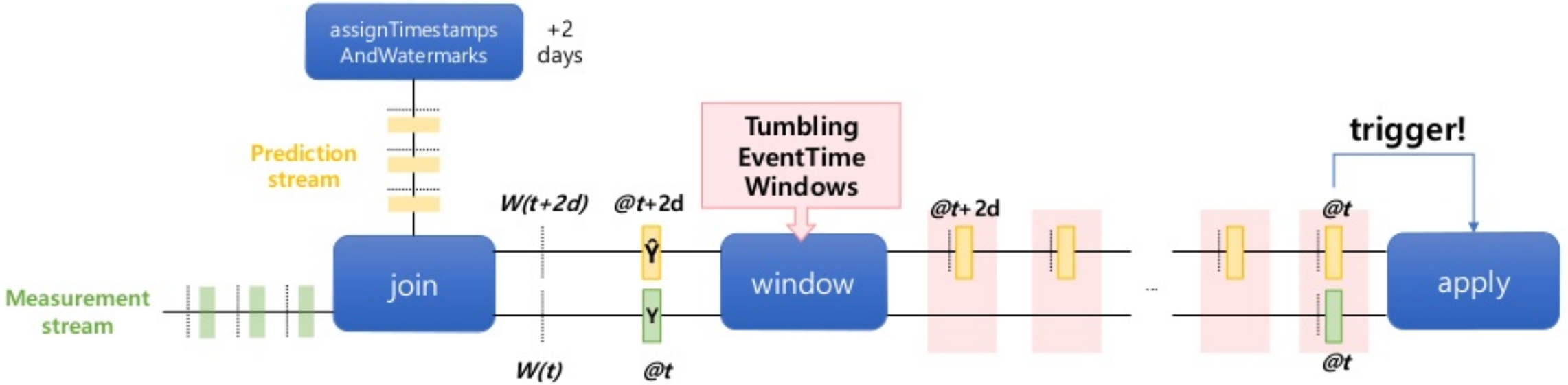

**TensorFlow Java API**

77.7 milliseconds per inference

# Data processing pipeline with **Flink DataStream API**

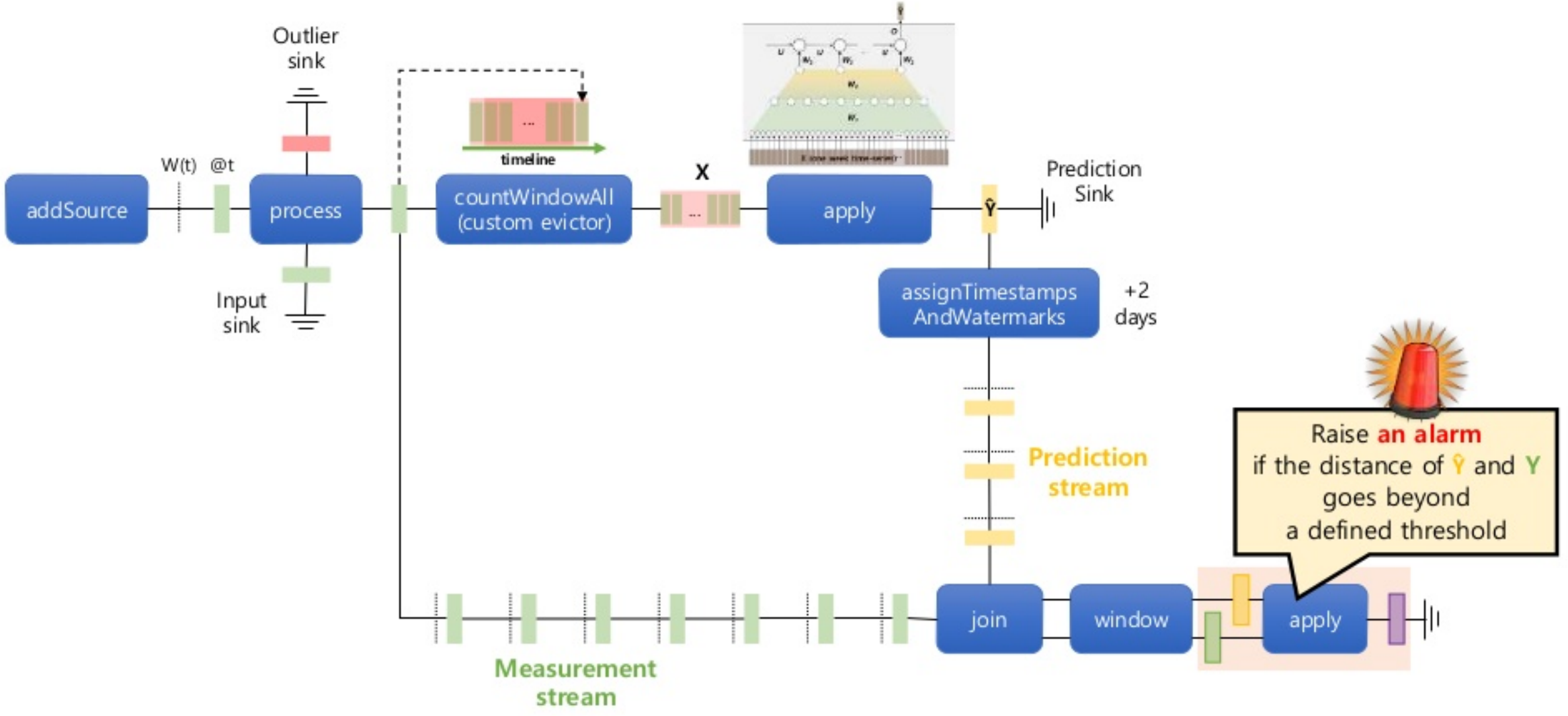


# Joining two streams on event time

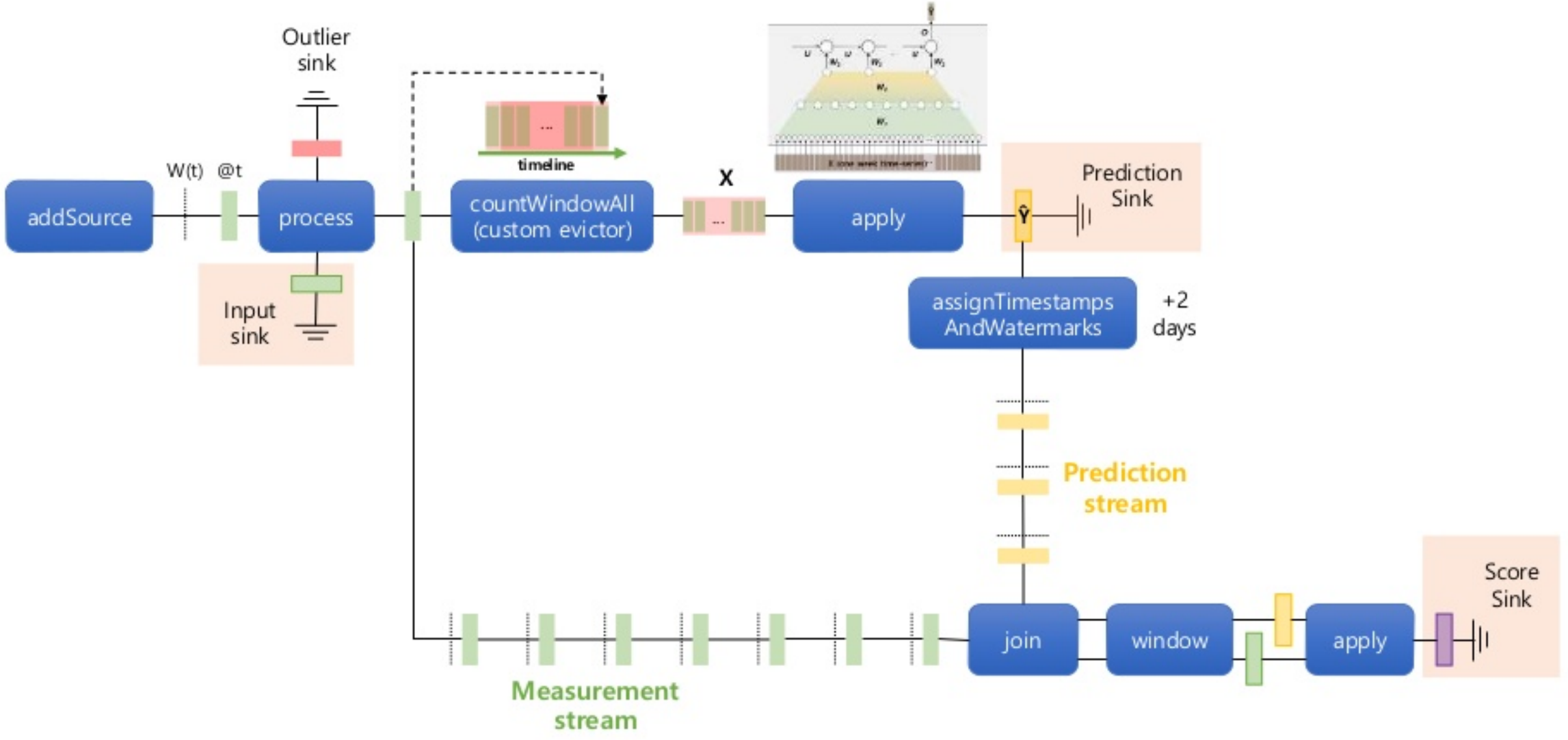



- At a certain time ***t***,
  - Y of timestamp ***t*** is arriving
  - Ŷ of timestamp ***t+2d*** is arriving
  - Ŷ of timestamp ***t*** has arrived **two days ago**
- TumblingEventTimeWindows.of( Time.seconds(**timeUnit**) )
  - To maintain **a window** for **a single pair of Y and Ŷ**
- A window is triggered when watermarks from both streams have arrived

# Data processing pipeline with **Flink DataStream API**



# Data processing pipeline with **Flink DataStream API**

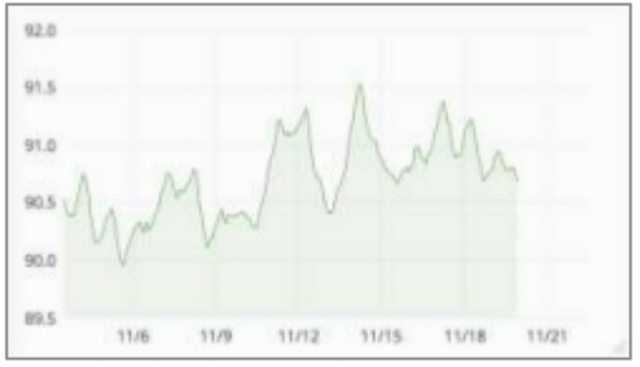


**Input**, **Prediction**, **Score** sinks write records to **InfluxDB**

We then plot time-series using **Grafana**

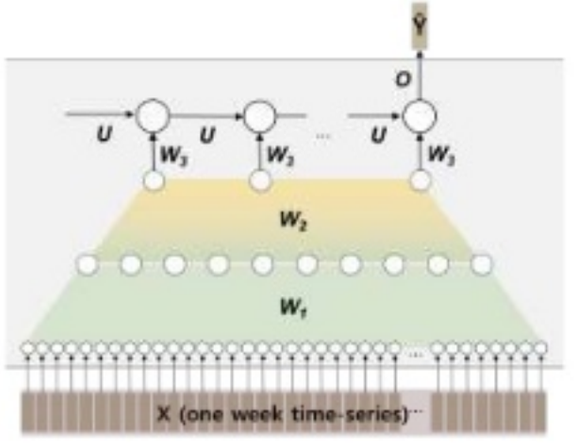
# Predicting from a single DNN is not enough!

**Measurement**



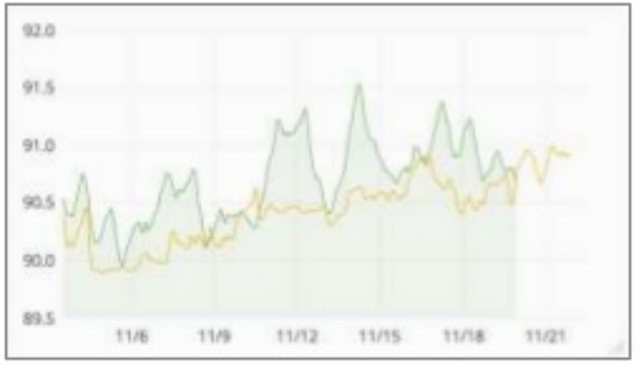

+

**Prediction from a single DNN**

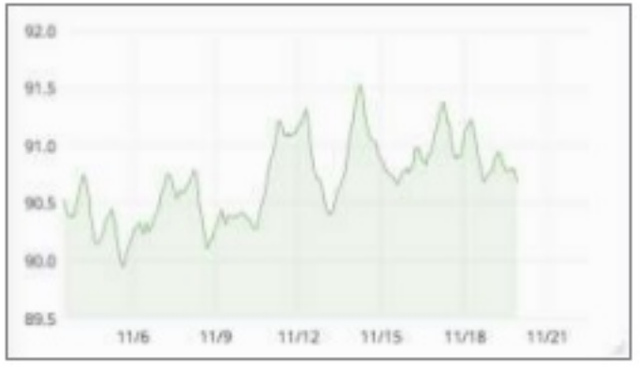



=

**Possibly biased prediction**

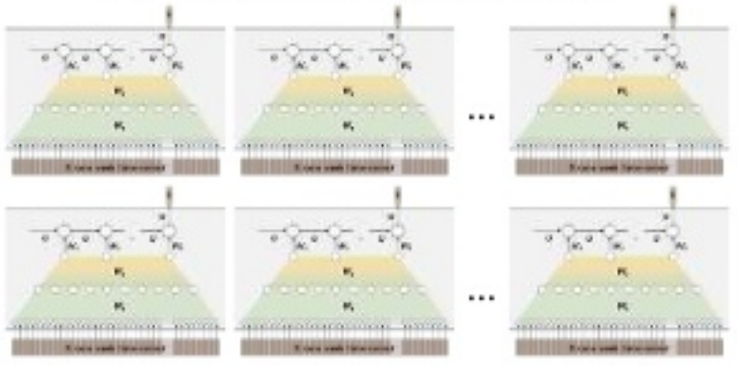



**Measurement**



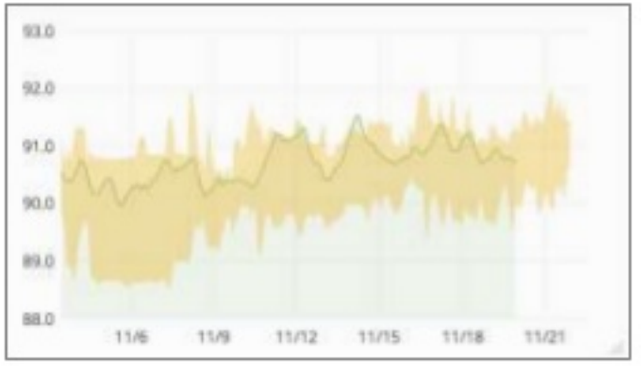

+

**Prediction from an ensemble of 10 DNNs**

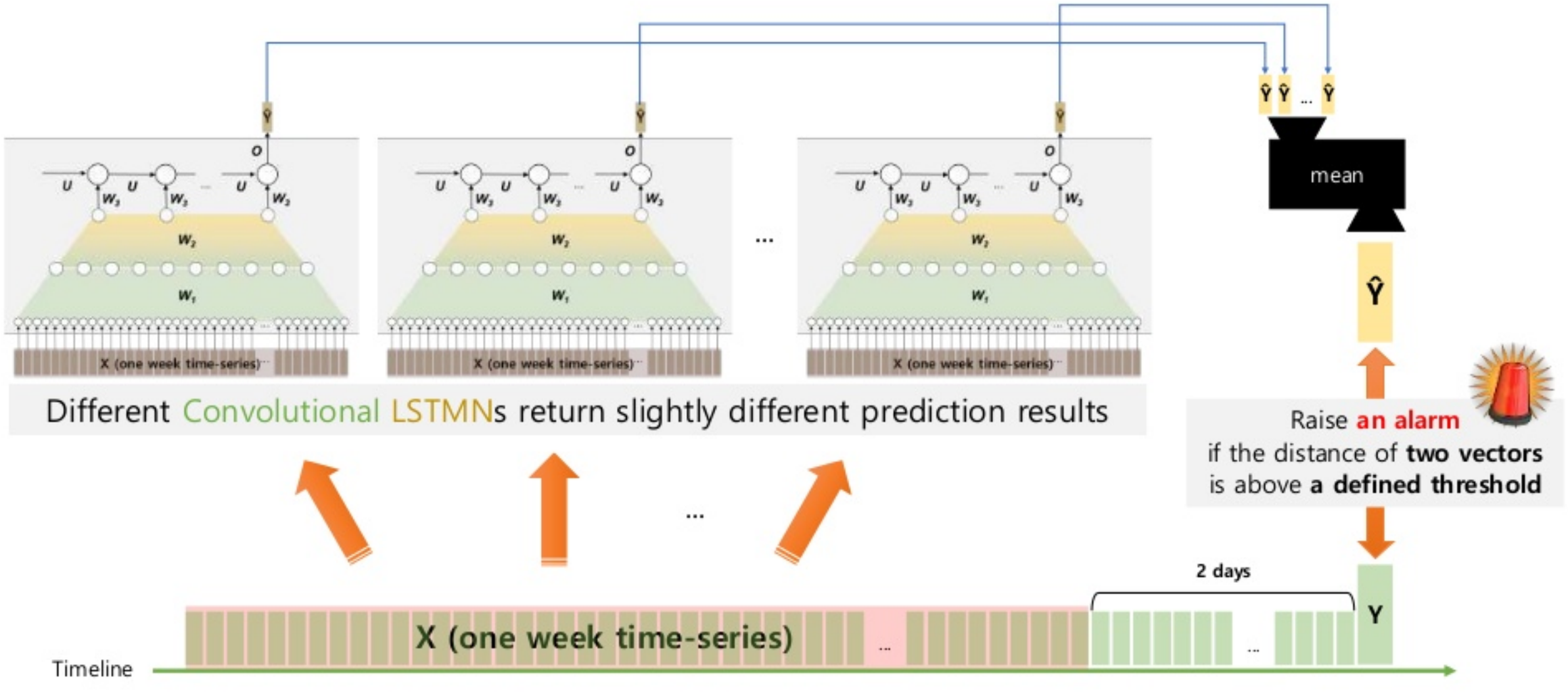


=

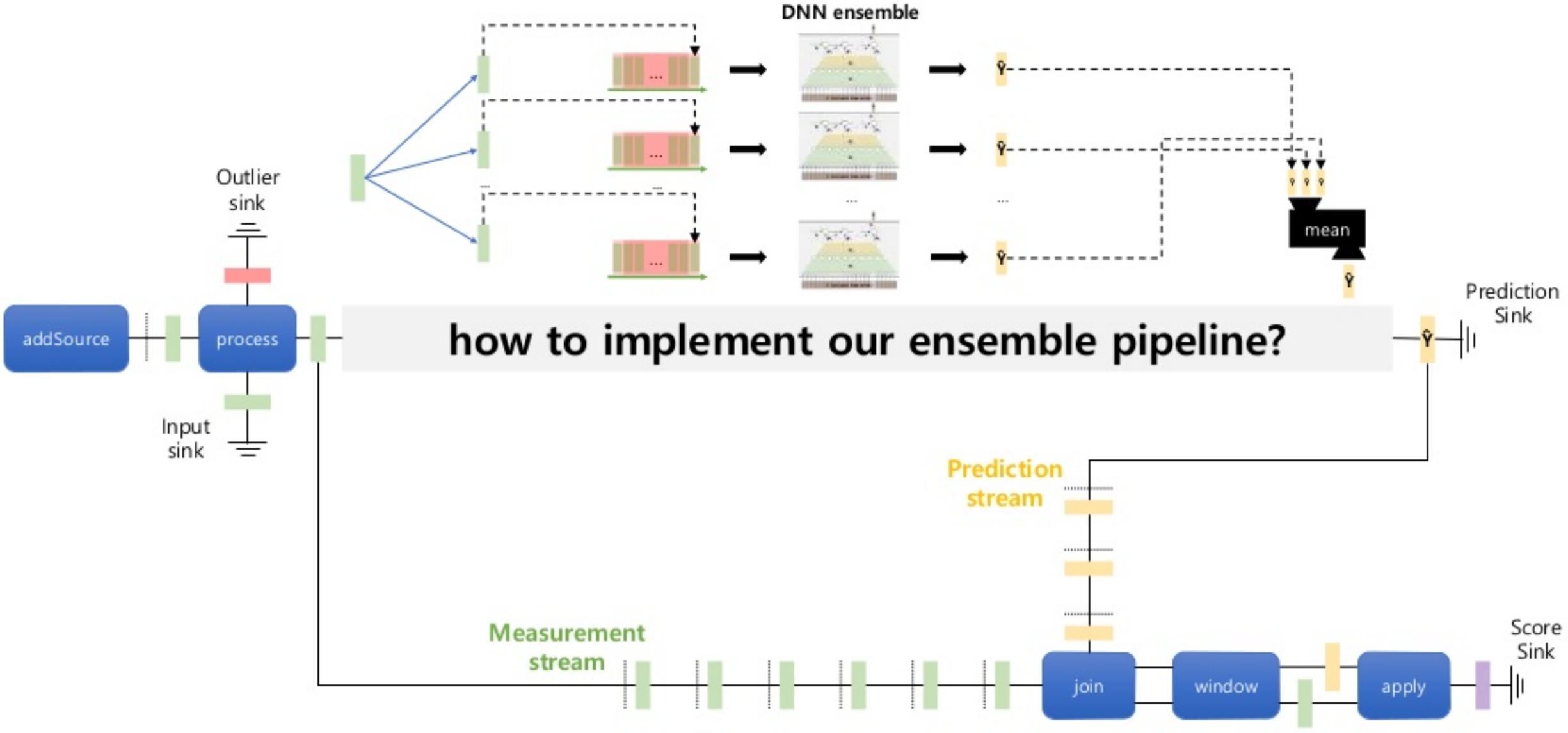
**More reliable prediction!**



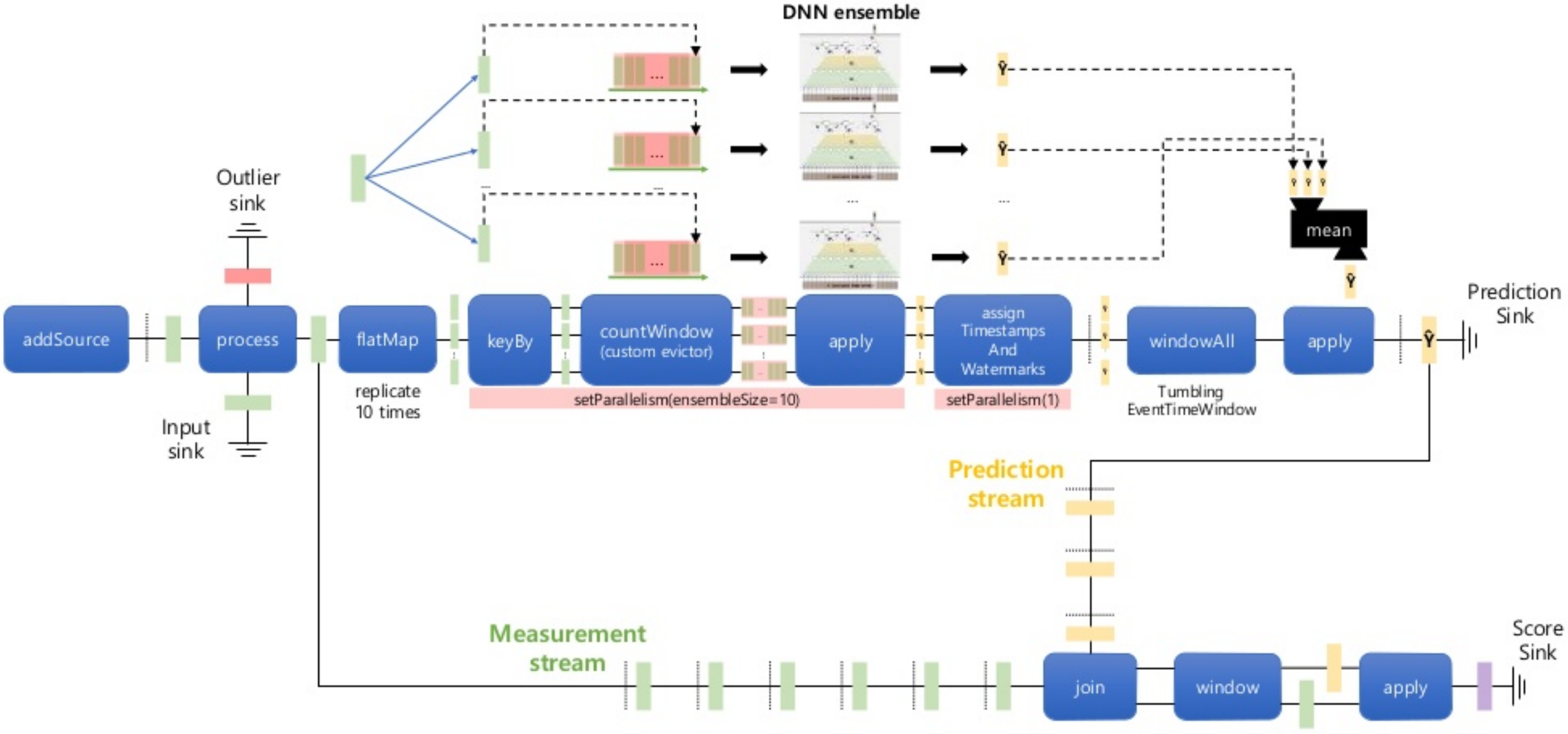
# DNN ensemble for **reliable** prediction



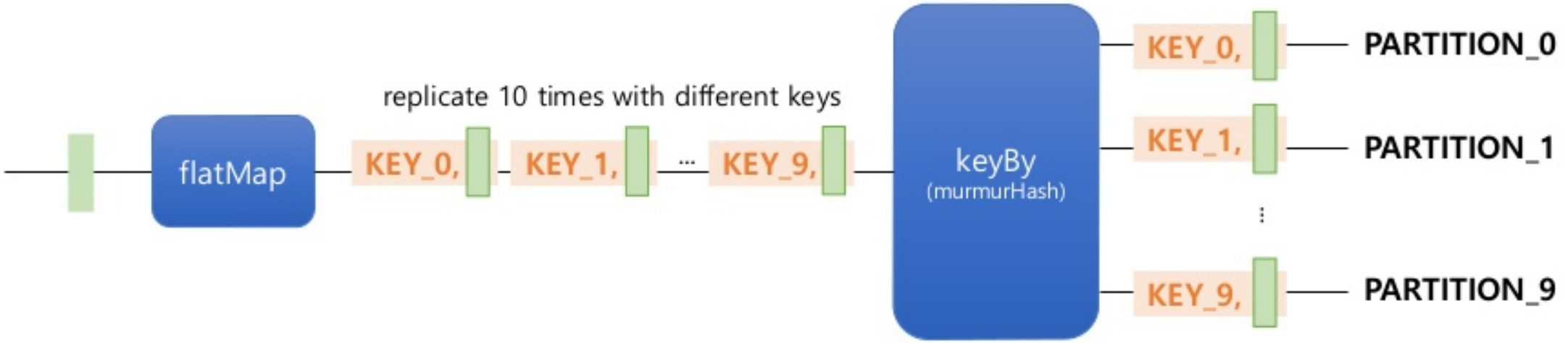
# Data processing pipeline with **Flink DataStream API**



# Data processing pipeline with **Flink DataStream API**



# Distribute 10 keys evenly over 10 partitions



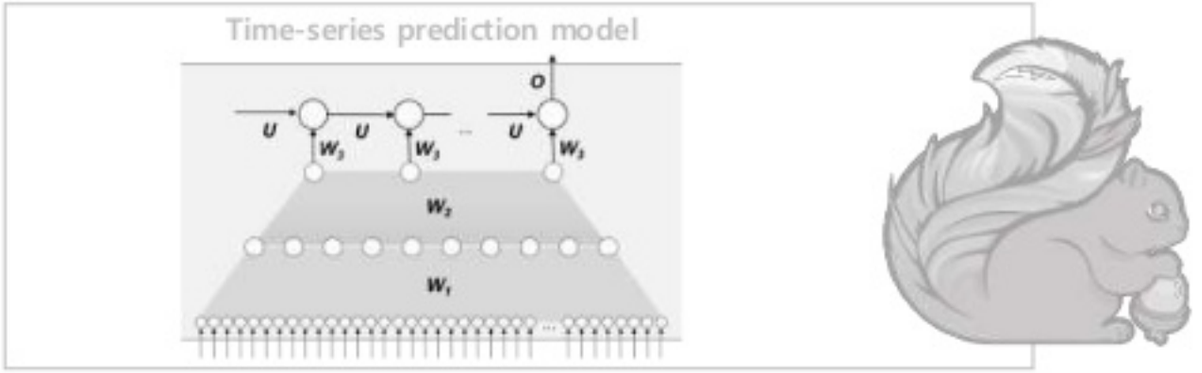

Carefully generate keys not to belong to the same partitions

**PARTITION** = murmurHash(**KEY**) / maxParallalism*(parallelism/maxParallism)

# Contents

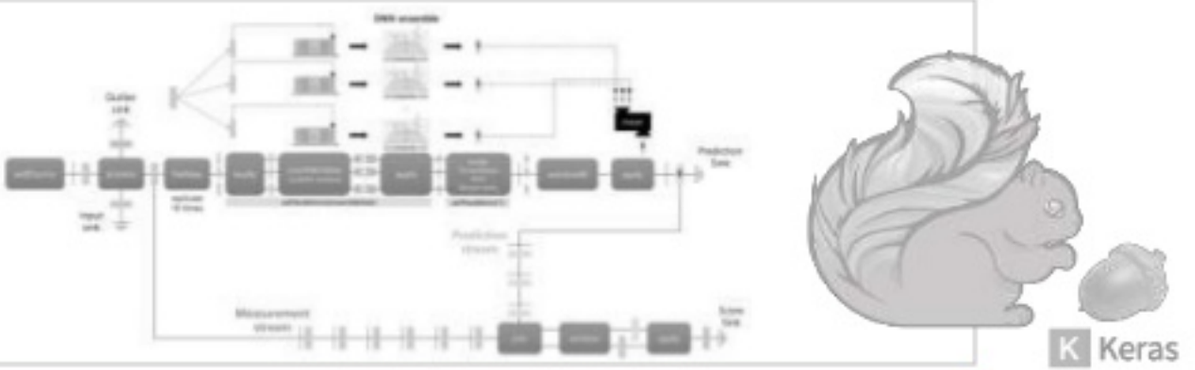
1

Why we use Flink
for our time-series prediction model




2

Flink pipeline design for
**rendezvous** and **DNN ensemble**

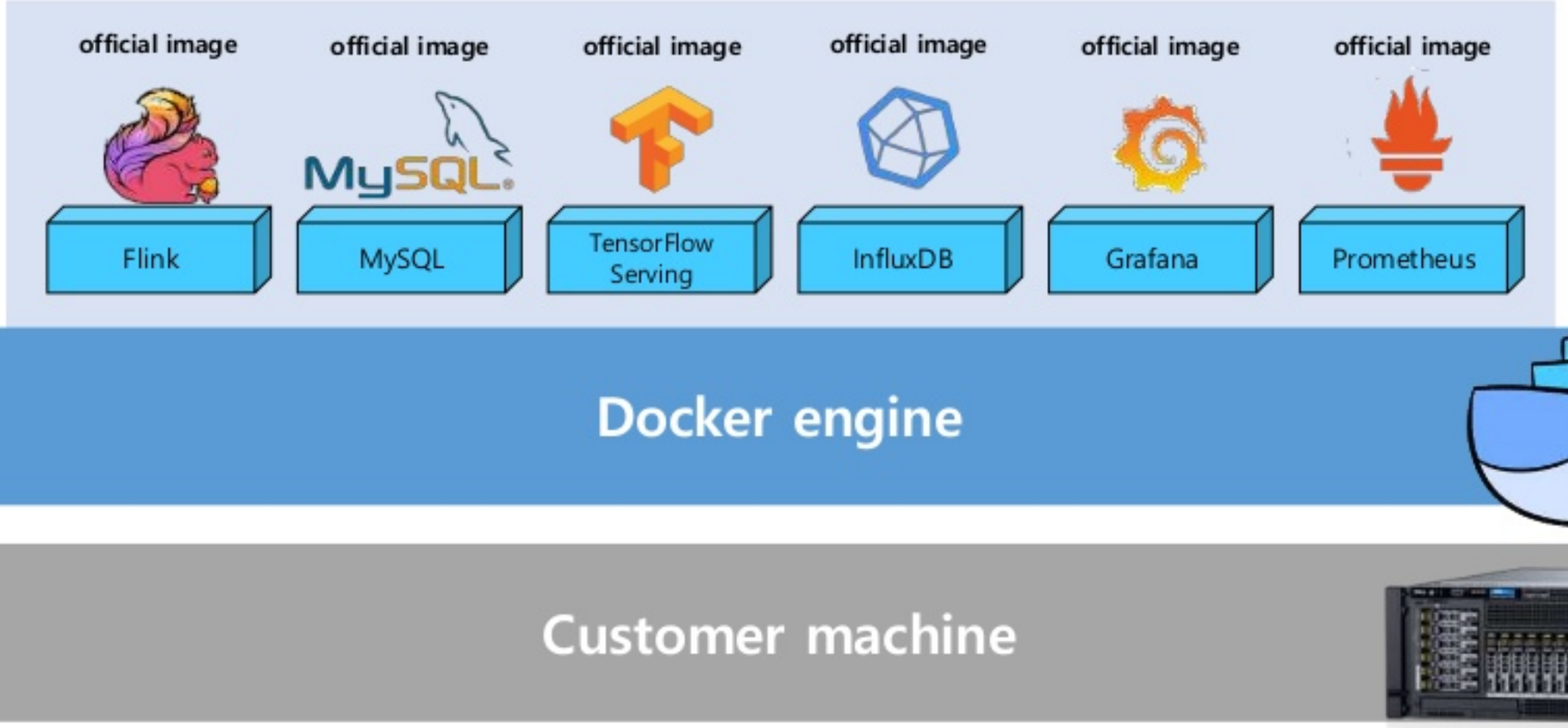



3

Solution packaging and monitoring
with **Docker** and **Prometheus**

# A **simple** software stack on top of **Docker**

***No custom Docker image!***
***A single yml file is okay to deploy our software stack!***



# Launch JobManager & TaskManager with some changes

**in the official repository of the Docker image for Flink**

## Running a cluster using Docker Compose

With Docker Compose you can create a Flink cluster:

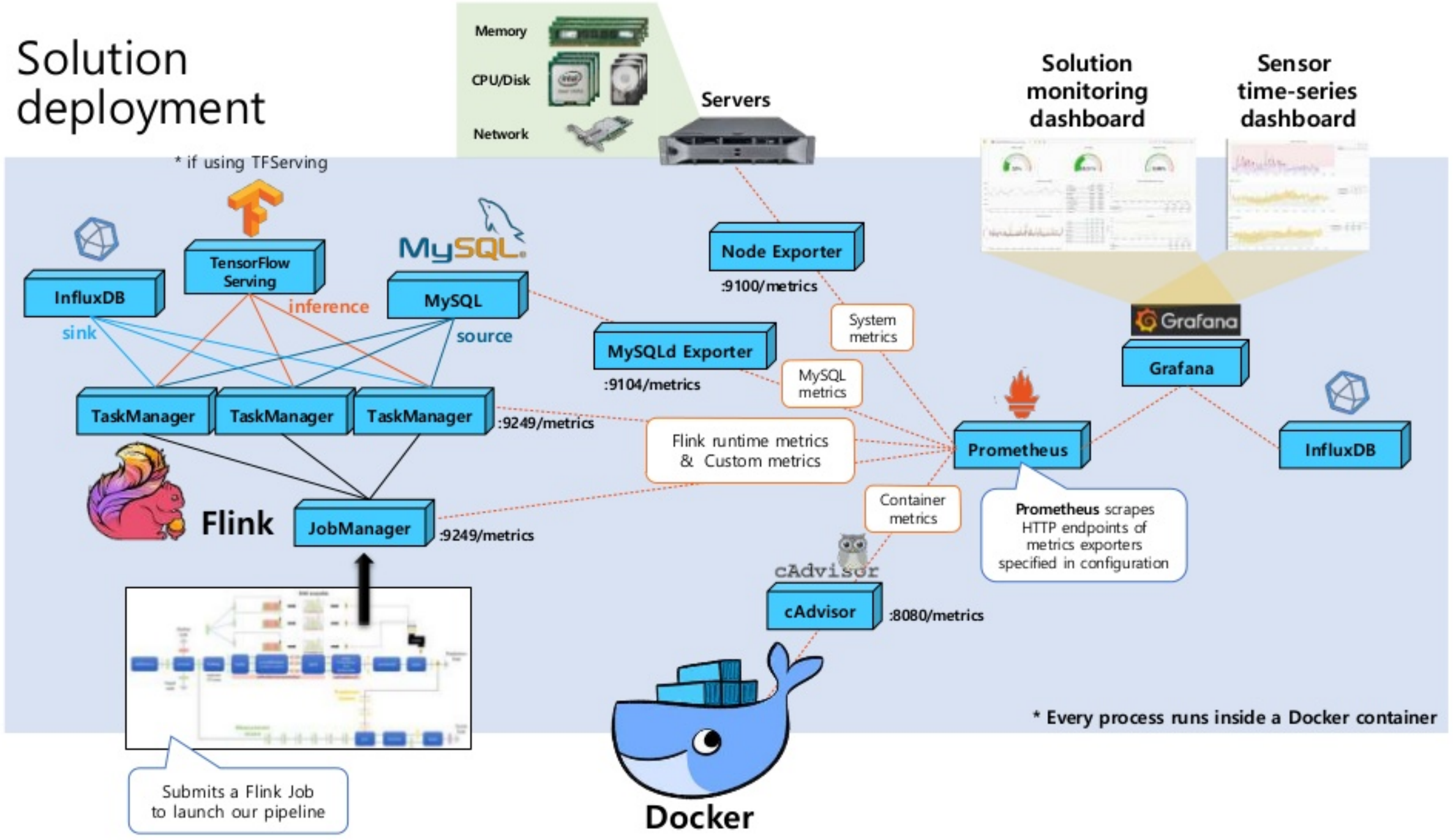
```
version: "2.1"
services:
  jobmanager:
    image: ${FLINK_DOCKER_IMAGE_NAME:-flink}
    expose:
      - "6123"
    ports:
      - "8081:8081"
    command: jobmanager
    environment:
      - JOB_MANAGER_RPC_ADDRESS=jobmanager

  taskmanager:
    image: ${FLINK_DOCKER_IMAGE_NAME:-flink}
    expose:
      - "6121"
      - "6122"
    depends_on:
      - jobmanager
    command: taskmanager
    links:
      - "jobmanager:jobmanager"
    environment:
      - JOB_MANAGER_RPC_ADDRESS=jobmanager
```

metrics.reporter : **prom**
metrics.reporter.prom.class :
**org.apache.flink.metrics.prometheus.PrometheusReporter**
jobmanager.heap.mb : **10240**
taskmanager.heap.mb : **10240**

```
volumes:
  - ${PWD}/flink-conf.yaml:/opt/flink/conf/flink-conf.yaml
  - ${PWD}/flink-metrics-prometheus-1.4-SNAPSHOT.jar:/opt/flink/lib/flink-metrics-prometheus-1.4-SNAPSHOT.jar
```

You need to get
**flink-metrics-prometheus-1.4-SNAPSHOT.jar**
by yourself
until Flink-1.4 is officially released

# Solution deployment

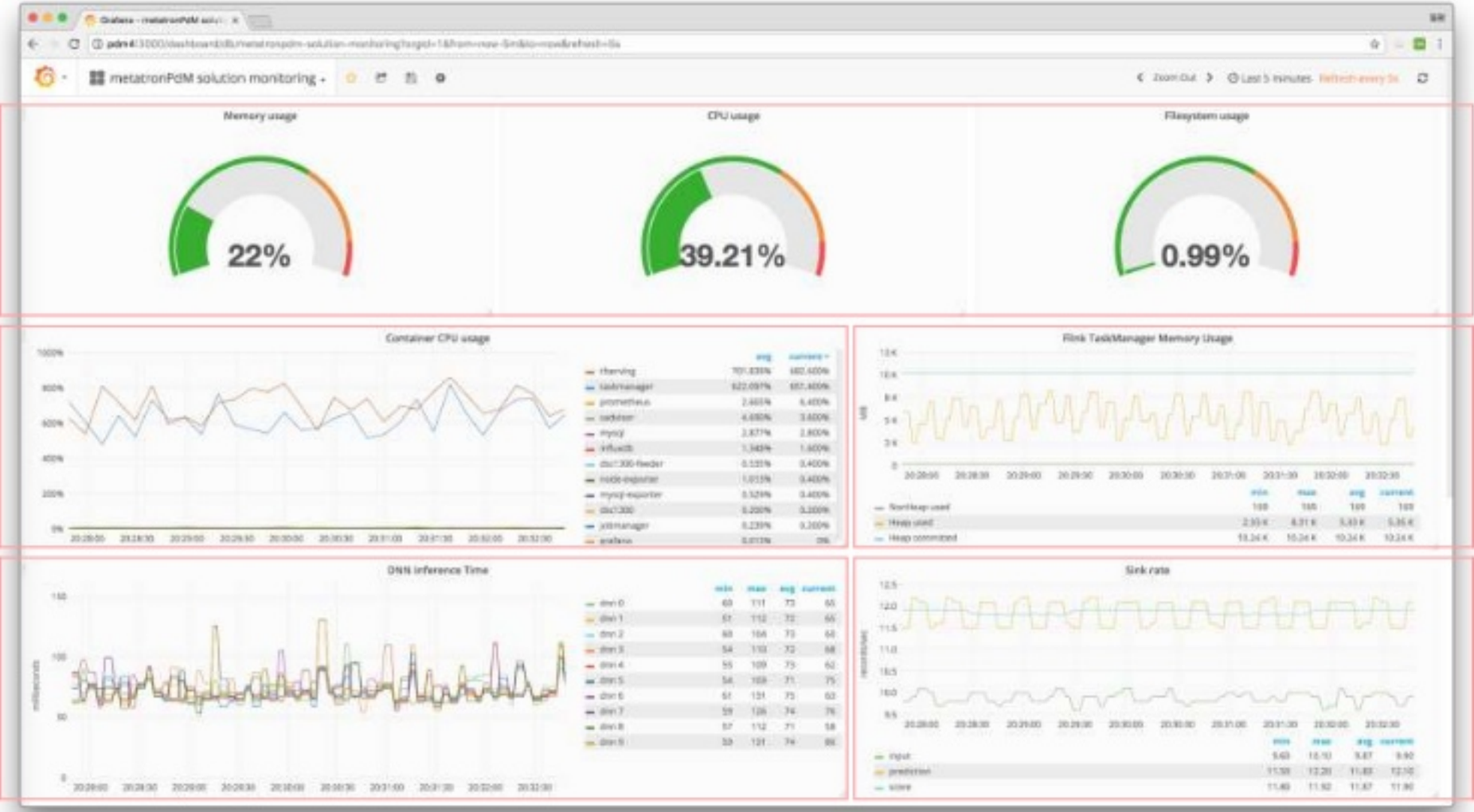


# Solution monitoring dashboard

Server Mem/CPU/FS usage (by **node exporter**)

Container CPU usage (by **cAdvisor**)

Inference time from each DNN (**custom** metrics)




TaskManager JVM memory Usage

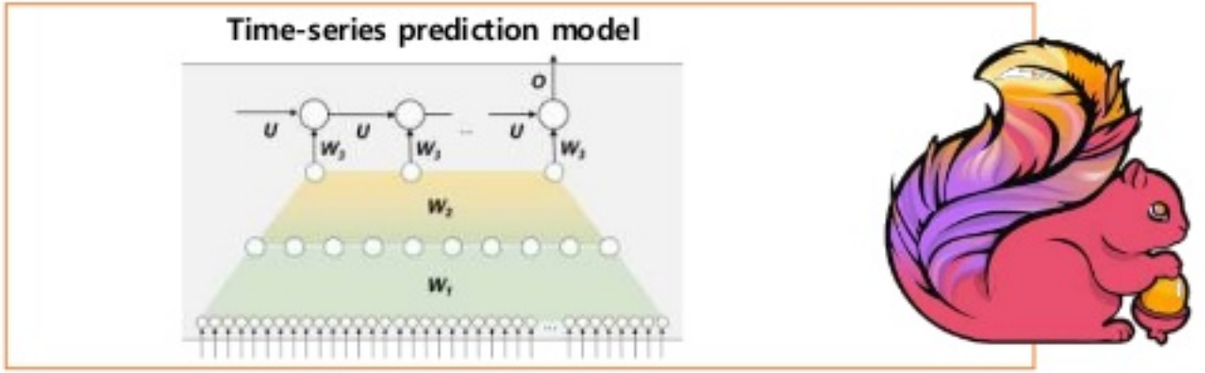
# records written to sinks (**custom** metrics)

* this dashboard is based on "Docker Dashboard" by Brian Christner

# Recap – contents

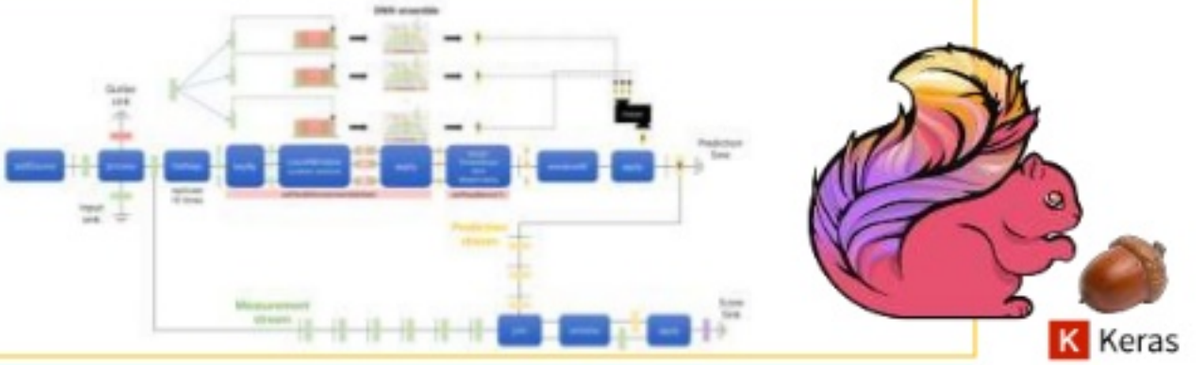
1

Why we use Flink
for our time-series prediction model



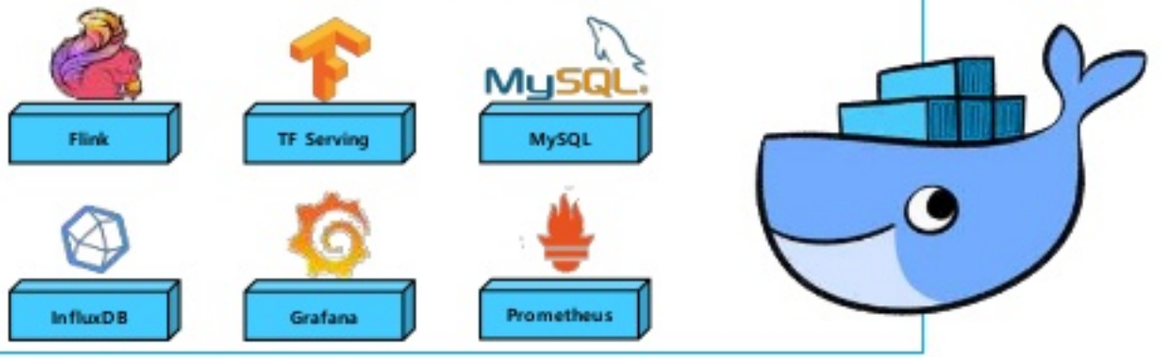

2

Flink pipeline design for
**rendezvous** and **DNN ensemble**



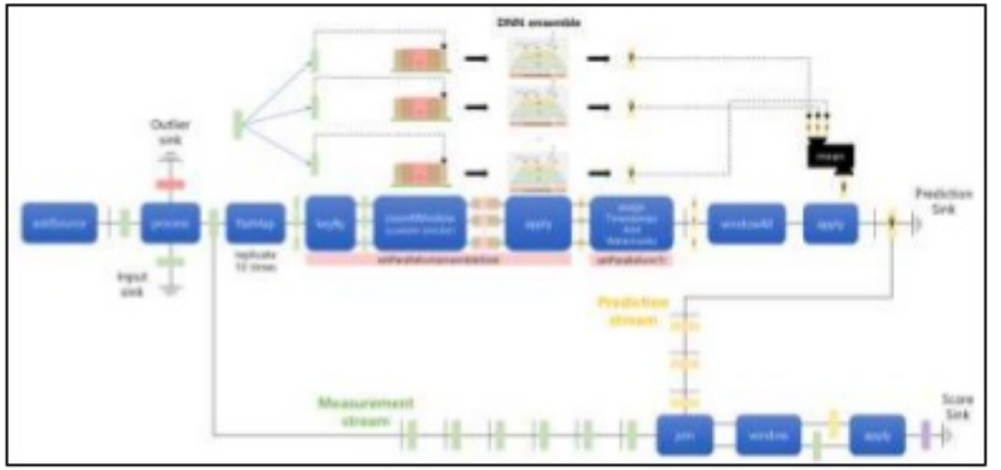

3

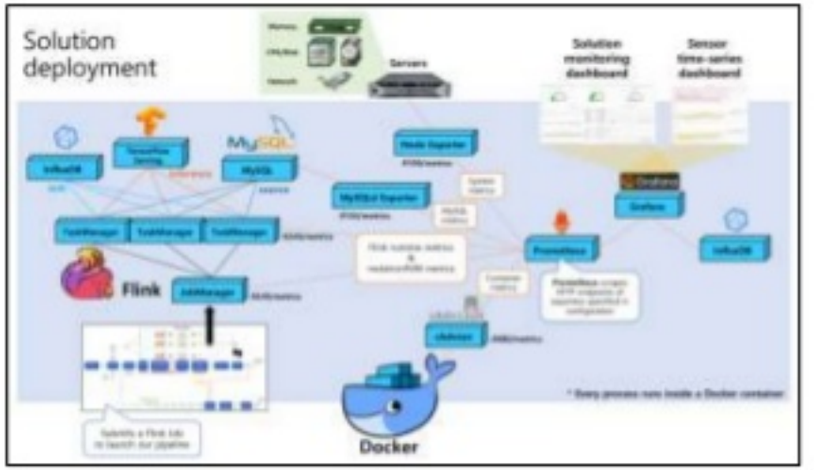
Solution packaging and monitoring
with **Docker** and **Prometheus**



# Conclusion

- **Flink** helps us concentrate on the core logic
  - DataStream API is just like ***a natural language*** in presenting streaming topologies
    - flexible windowing mechanisms (count window and evictor)
    - joining of two streams on event time
  - Thanks to it, we can focus on
    - implementation of custom source/sink to meet customer requirements
    - interaction with DNN ensembles
  - It has a nice ecosystem to help build a solution
    - Docker
    - Prometheus metric reporter



THE END

# You cannot use **TFServing** with **Flink-1.3.2**

- Netty binary incompatbility
  - **flink-runtime_2.11**:1.3.2
    - depends on **io.netty:4.0.27.Final**
  - **grpc-netty**:4.1.14
    - depends on **io.netty:4.1.14.Final**

**From "New and noteworthy in 4.1" page of Netty project**
4.1 contains multiple additions which might not be fully backward-compatible with 4.0

- You could use **grpc-okhttp** instead of **grpc-netty**
  - **grpc-okhttp** conflicts with another library (influxdb client for Java)

- You can use **TFServing** with **FLINK-1.4-SNAPSHOT**
  - [FLINK-7013] Add shaded netty dependency
    - **io.netty.*** is recently relocated to **org.apache.flink.shaded.netty4.***

# Looking forward to the official release of v1.4

Flink / FLINK-6221

Add Prometheus support to metrics

Agile Board

Export

**Details**

| | | | |
|---|---|---|---|
| Type: | Improvement | Status: | CLOSED |
| Priority: | Minor | Resolution: | Fixed |
| Affects Version/s: | 1.2.0 | Fix Version/s: | 1.4.0 |
| Component/s: | Metrics | | |
| Labels: | None | | |

**People**

| | |
|---|---|
| Assignee: | Maximilian Bode |
| Reporter: | Joshua Griffith |
| Votes: | 0 Vote for this issue |
| Watchers: | 7 Start watching this issue |

**Description**

Prometheus is becoming popular for metrics and alerting. It's possible to use statsd-exporter to load Flink metrics into Prometheus but it would be far easier if Flink supported Promethus as a metrics reporter. A dropwizard client exists that could be integrated into the existing metrics system.

**Dates**

| | |
|---|---|
| Created: | 30/Mar/17 16:15 |
| Updated: | 02/Jul/17 12:38 |
| Resolved: | 01/Jul/17 10:13 |

Flink / FLINK-6529 Rework the shading model in Flink / FLINK-7013

Add shaded netty dependency

Agile Board

Export

**Details**

| | | | |
|---|---|---|---|
| Type: | Sub-task | Status: | CLOSED |
| Priority: | Major | Resolution: | Fixed |
| Affects Version/s: | None | Fix Version/s: | None |
| Component/s: | Network | | |
| Labels: | None | | |

**People**

| | |
|---|---|
| Assignee: | Chesnay Schepler |
| Reporter: | Chesnay Schepler |
| Votes: | 0 Vote for this issue |
| Watchers: | 3 Start watching this issue |

**Issue Links**

links to GitHub Pull Request #4452

**Dates**

| | |
|---|---|
| Created: | 27/Jun/17 11:18 |
| Updated: | 07/Aug/17 13:18 |
| Resolved: | 28/Jun/17 09:00 |

**No more 1.4-SNAPSHOT on the customer site!**