APACHE FLINK

Technologies for Big Data Mangement

NICCOLÒ VACCA

INDEX

- Introduzione ad Apache Flink
- Streaming in Apache Flink
- Flink SQL
- Flink Runtime
- Stateful Stream Processing
- Event Time & Watermarks
- Checkpoints & Recovery

INTRODUZIONE AD APACHE FLINK

COS'È APACHE FLINK?

Apache Flink è uno **stream processor** utilizzato prevalentemente in applicazioni real-time

Questo **Framework** è basato su 4 concetti principali:

- **STREAMING**
- **STATE**
- **TIME**
- **SNAPSHOT**

Flink offre **API** in Java, Scala, SQL e altri linguaggi e supporta sia **stream** che **batch** processing

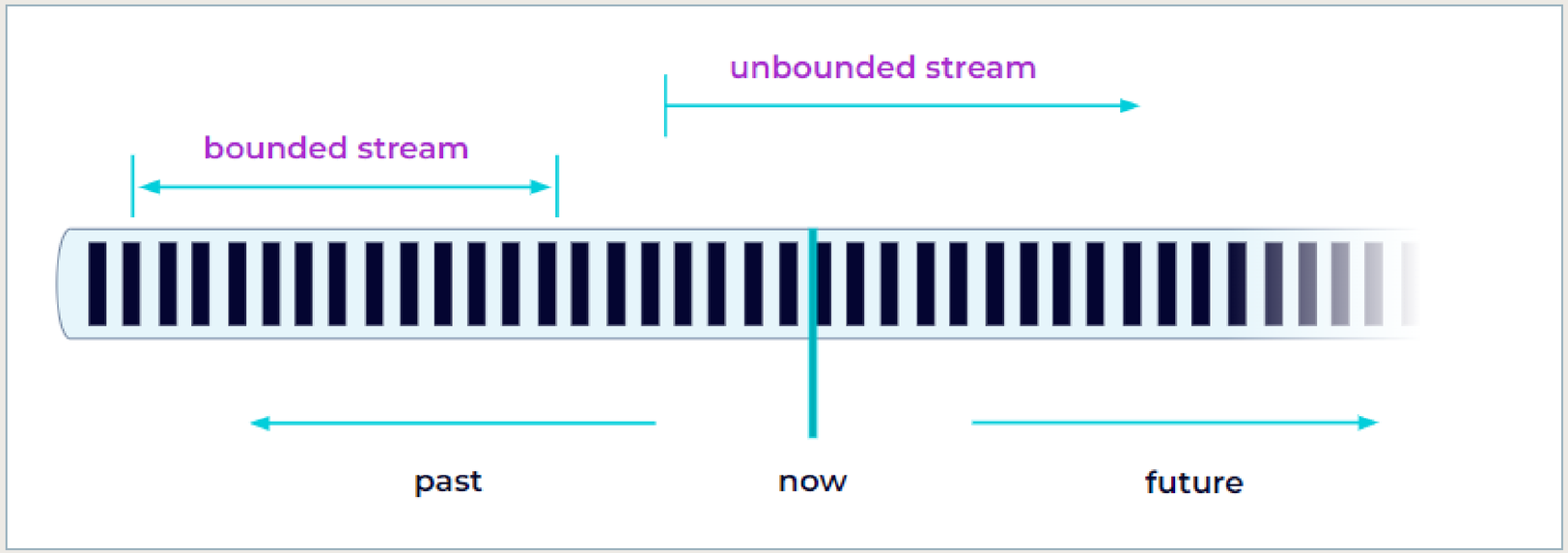
Flink ha una **community numerosa e attiva**, ed è utilizzato da molte aziende importanti, tra cui Netflix, Alibaba e Uber

STREAMING IN APACHE FLINK

STREAMING IN APACHE FLINK

Apache Flink cattura gli eventi da una sorgente, non appena questi si verificano (**Event Streaming**)

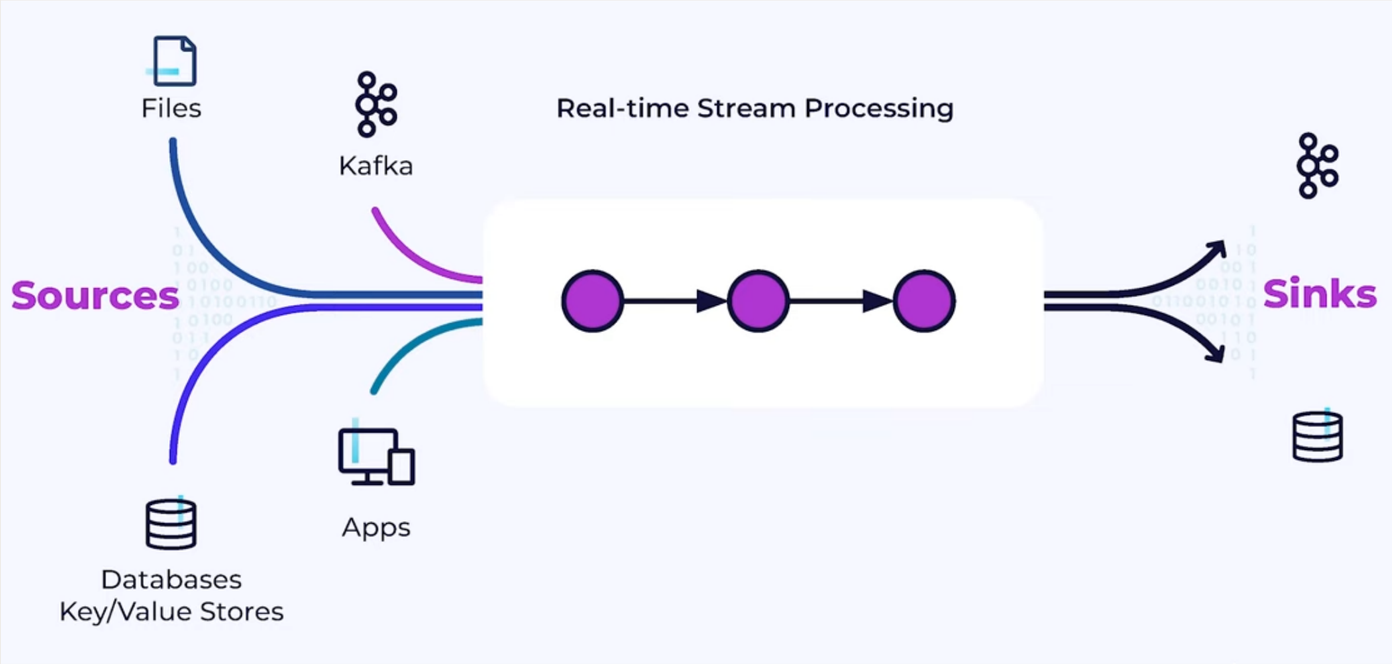
- Le sequenze di eventi letti formano uno **stream potenzialmente illimitato** che si estende nel futuro
- Se si vogliono analizzare dati historical vanno considerati invece stream finiti, con un timestamp di inizio e uno di fine



STRUTTURA DI UN' APPLICAZIONE FLINK

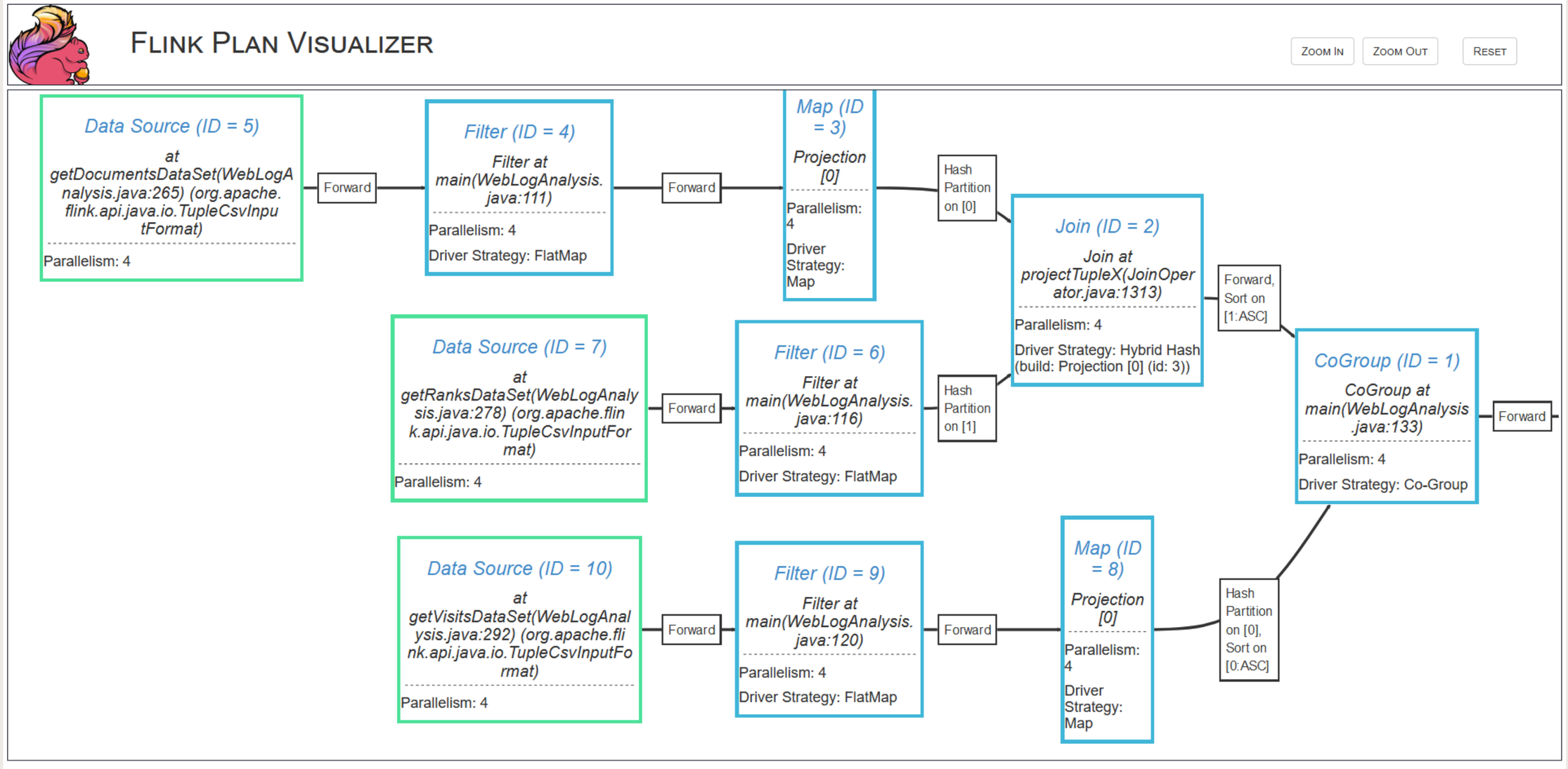
Tutte le Applicazioni Flink consumano dati da una o più **sorgenti** e producono dati su uno o più **sink**

La Business Logic è implementata grazie alle **API Flink** e viene eseguita in un **cluster Flink**



STRUTTURA DI UN' APPLICAZIONE FLINK

- Un'applicazione in esecuzione su Flink è chiamata **Job**
- I dati relativi agli eventi registrati, le computazioni da effettuare e i messaggi da scambiare vengono raccolte in una pipeline chiamata **Job Graph**
- I nodi "**Operator**" che si trovano nel Job Graph rappresentano le computazioni
- Il Job Graph è un **DAG**
- È possibile dividere gli stream di eventi in **sub-stream paralleli** che possono essere processati indipendemente


Flink Plan Visualizer
Zoom In
Zoom Out
Reset
Data Source (ID = 5)
at getDocumentsDataSet(WebLogAnalysis.java:265) (org.apache.flink.api.java.io.TupleCsvInputFormat)
Parallelism: 4
Forward
Filter (ID = 4)
Filter at main(WebLogAnalysis.java:111)
Parallelism: 4
Driver Strategy: FlatMap
Forward
Map (ID = 3)
Projection [0]
Parallelism: 4
Driver Strategy: Map
Hash Partition on [0]
Data Source (ID = 7)
at getRanksDataSet(WebLogAnalysis.java:278) (org.apache.flink.api.java.io.TupleCsvInputFormat)
Parallelism: 4
Forward
Filter (ID = 6)
Filter at main(WebLogAnalysis.java:116)
Parallelism: 4
Driver Strategy: FlatMap
Hash Partition on [1]
Join (ID = 2)
Join at projectTupleX(JoinOperator.java:1313)
Parallelism: 4
Driver Strategy: Hybrid Hash (build: Projection [0] (id: 3))
Forward, Sort on [1:ASC]
CoGroup (ID = 1)
CoGroup at main(WebLogAnalysis.java:133)
Parallelism: 4
Driver Strategy: Co-Group
Forward
Data Source (ID = 10)
at getVisitsDataSet(WebLogAnalysis.java:292) (org.apache.flink.api.java.io.TupleCsvInputFormat)
Parallelism: 4
Forward
Filter (ID = 9)
Filter at main(WebLogAnalysis.java:120)
Parallelism: 4
Driver Strategy: FlatMap
Forward
Map (ID = 8)
Projection [0]
Parallelism: 4
Driver Strategy: Map
Hash Partition on [0], Sort on [0:ASC]

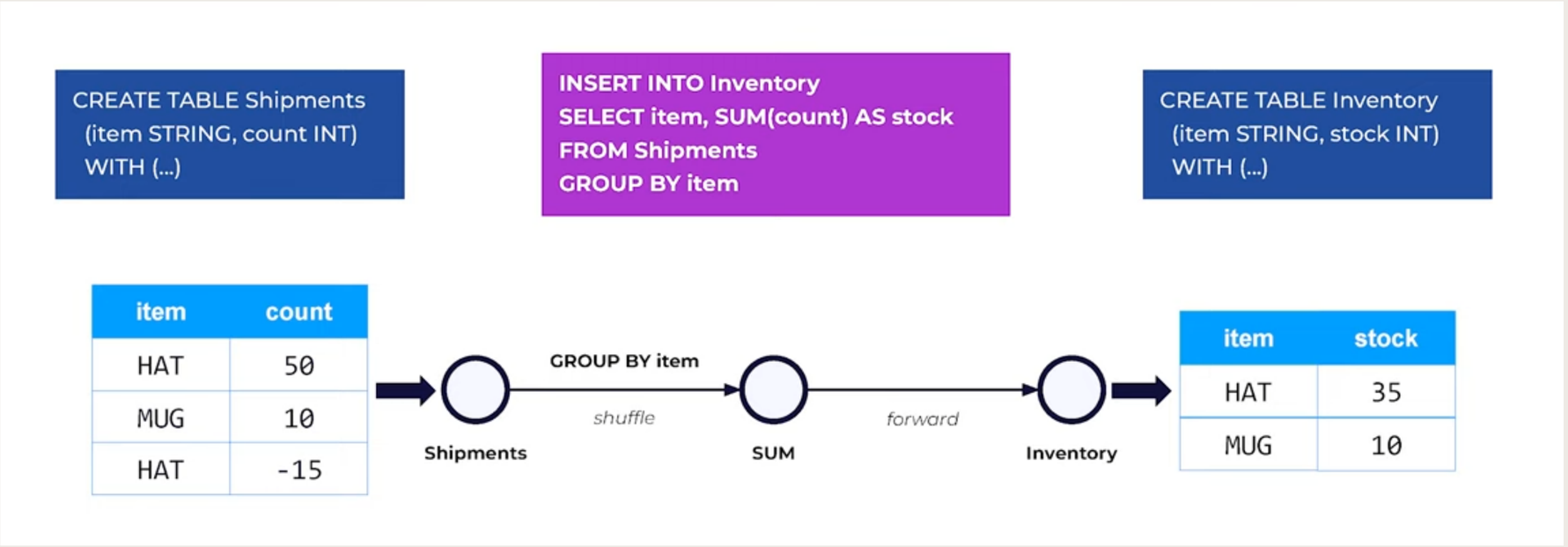
POSSIBILI OPERAZIONI FLINK

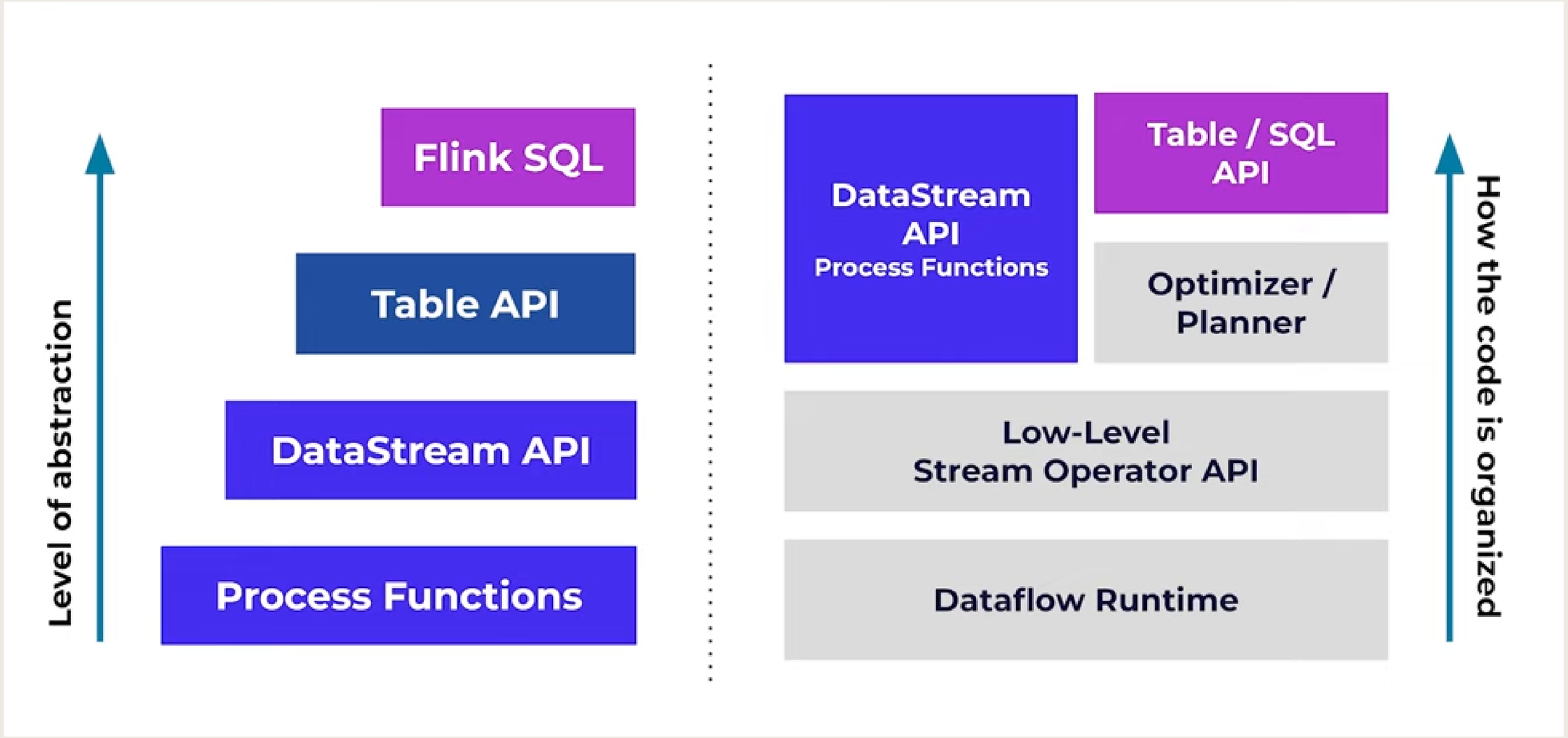
- **Forwarding**: Connessione più semplice tra i nodi, si limita a scambiare dati
- **Partitioning**: Divisione di una tabella in più parti relative a essa
- **Random Partitioning (o shuffling)**: Partiziona gli elementi randomicamente
- **Rebalancing**: Partiziona gli elementi seguendo l’approccio round-robin, per creare una distribuzione di carico bilanciata per ogni partizione

FLINK SQL

FLINK SQL

Flink SQL è un engine conforme agli standard ANSI per SQL in grado di trasformare istruzioni SQL in intere Applicazioni Flink.
Può processare dati sia in modalità Streaming che Batch, mantenendo i principi di scalabilità, performance e consistenza tipici di Flink

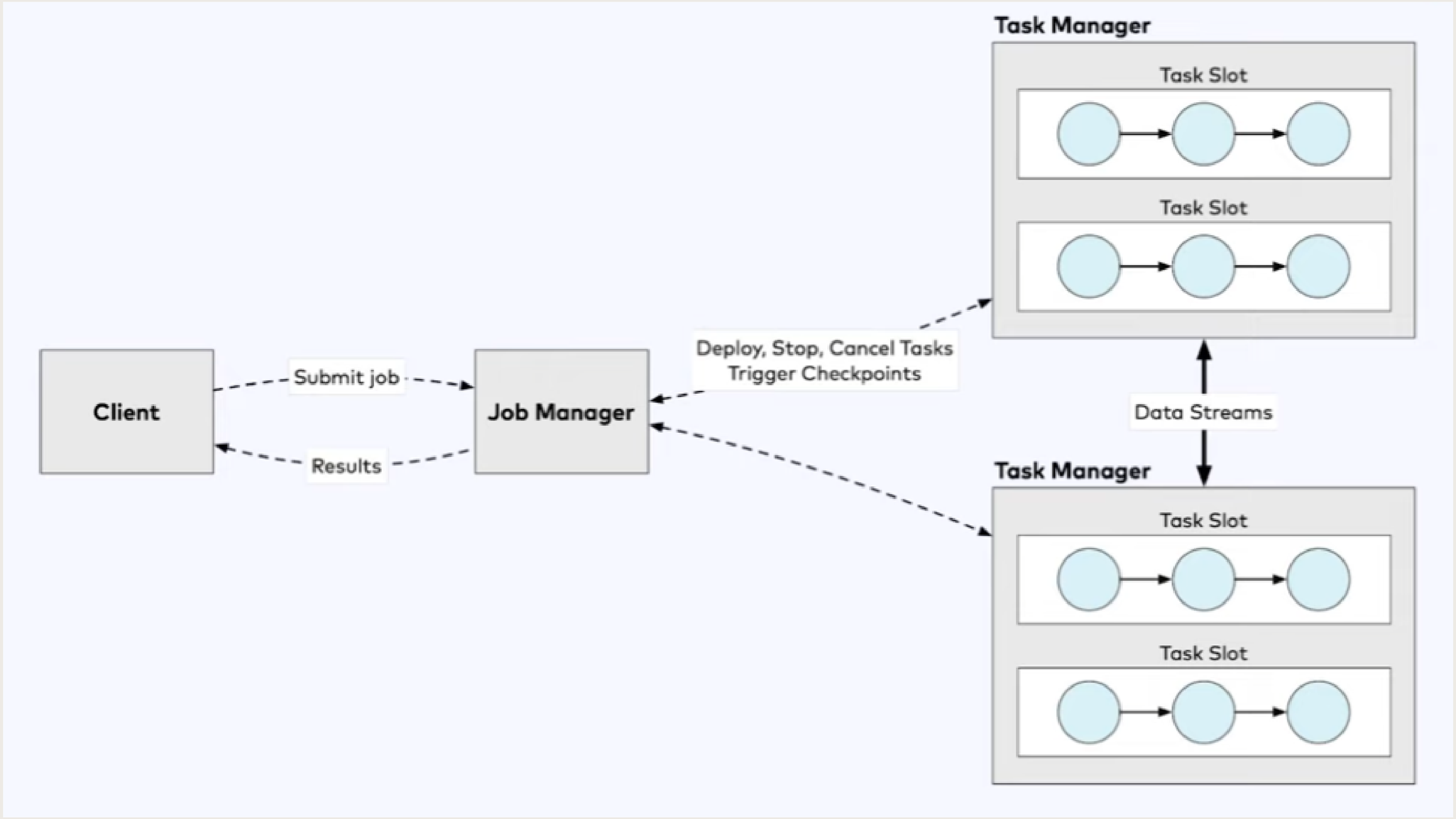



Level of abstraction
Flink SQL
Table API
DataStream API
Process Functions
DataStream API
Process Functions
Table / SQL API
Optimizer / Planner
Low-Level
Stream Operator API
Dataflow Runtime
How the code is organized

FLINK RUNTIME

FLINK RUNTIME

- Quando si scrive un’applicazione Flink usando le API Flink, la stessa App diventa un **Client Flink**
- Quando il Client viene eseguito, viene assemblato il Job Graph relativo e il **Job** viene **sottomesso al Job Manager**
- Il Job Manager istanzia le risorse necessarie per eseguire il Job (nodi **Task Manager**)
- Ciascun Task Manager fornisce diversi **Task Slot**, che si occupano di eseguire le singole computazioni del Job
- I Task Manager possono scambiarsi dati tra di loro
- Durante l’esecuzione, il **Job Manager** è responsabile della **gestione** dei **Checkpoint** e dei **Failure Recovery**


Task Manager
Task Slot
Task Slot
Client
Submit job
Results
Job Manager
Deploy, Stop, Cancel Tasks
Trigger Checkpoints
Data Streams
Task Manager
Task Slot
Task Slot

BATCH PROCESSING IN FLINK

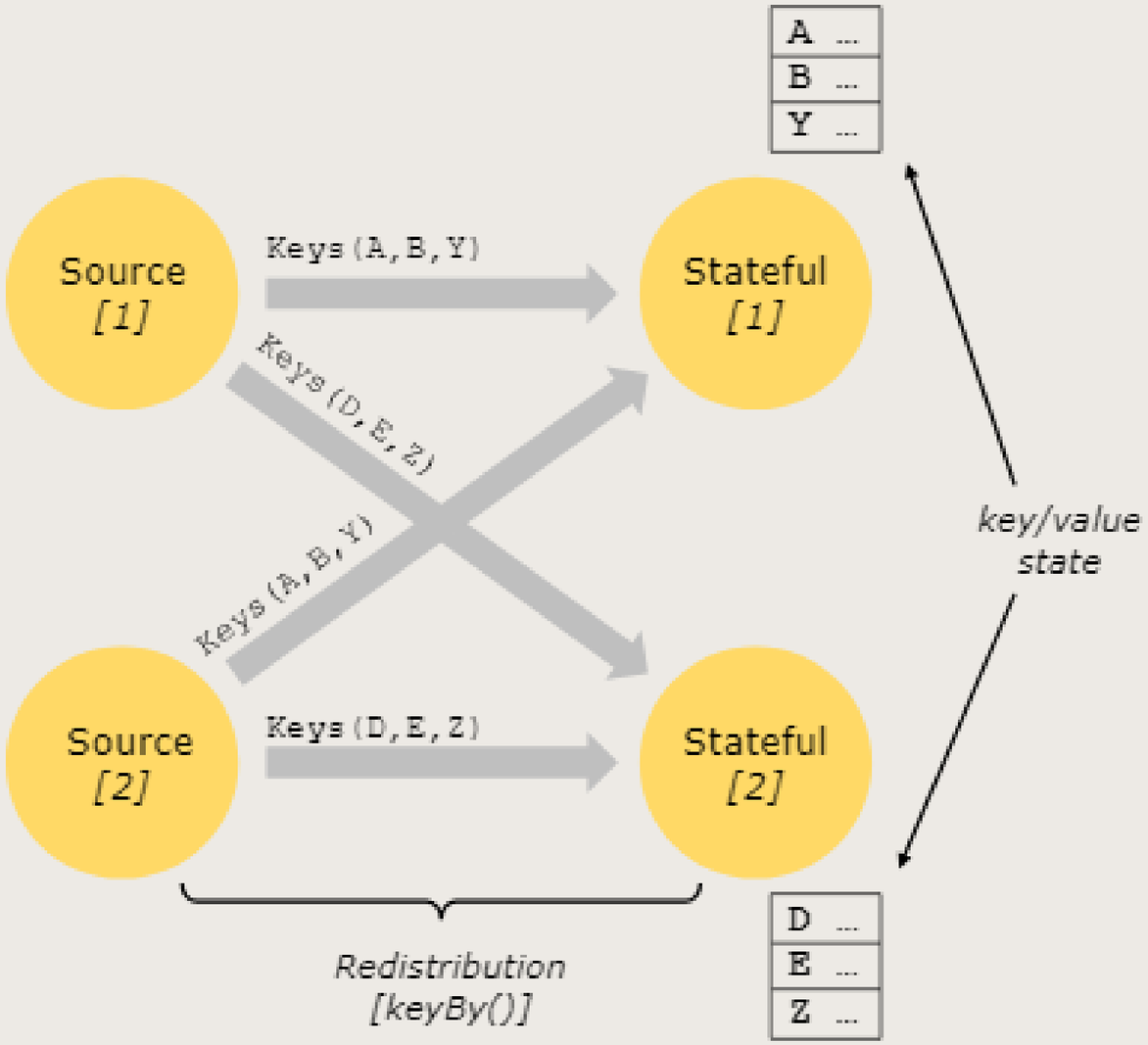
In Flink, il Batch processing è una **casistica specifica di runtime**, in cui l'**esecuzione** procede **in fasi** e dove i dati intermedi vengono salvati in dei **buffer** per venire processati successivamente.
I risultati finali vengono restituiti al termine del Job

STATEFUL STREAM PROCESSING

STATEFUL STREAM PROCESSING

Mentre molte operazioni in uno stream di dati esaminano semplicemente un singolo evento alla volta, alcune operazioni devono ricordare le **informazioni su più eventi** (ad esempio nelle time window). Queste operazioni sono chiamate **stateful**.

Gli stati vengono salvati su strutture di dati **key-value** distribuite, per cui ogni istanza parallela gestisce lo stato per le sue key specifiche.



Per garantire **performance elevate**, Flink salva gli stati dei Job localmente, per ogni nodo operator e per garantire **Fault Tolerance**, esegue periodicamente dei checkpoint degli stati, copiandoli di volta in volta in uno spazio di archiviazione remoto (Ad esempio in S3)

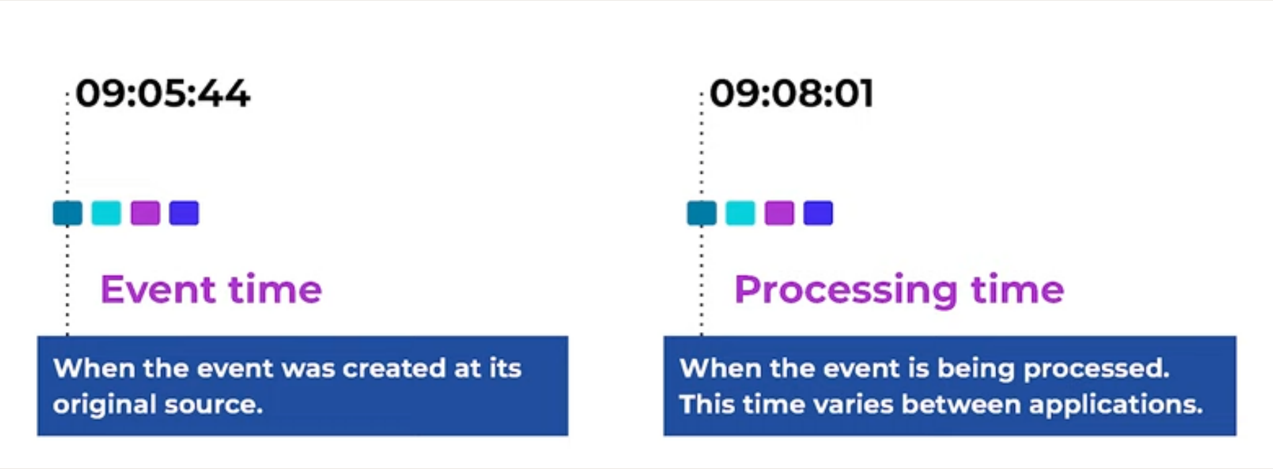
STATEFUL STREAM PROCESSING

Flink può salvare una **grande mole di dati**, a patto che gli stream da cui vengono salvati **non** siano **illimitati** (Stram Processing).
Per questo vanno aggiunti dei **vincoli temporali** alle computazioni (**Time Windows**), così di volta in volta lo spazio di archiviazione di Flink può essere svuotato e non arriverà mai al suo limite.

Quando la computazione è basata sull' Event Time, la gestione di finestre temporali può essere complessa

EVENT TIME
&
WATERMARKS

EVENT TIME & WATERMARKS

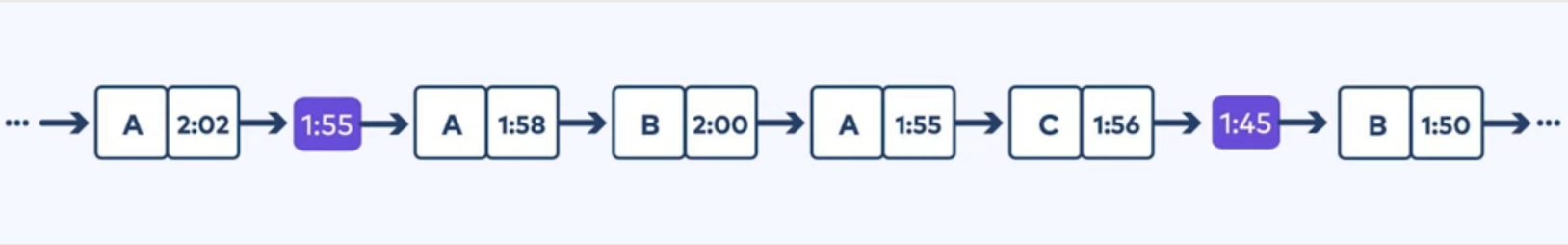


È possibile che un evento generato prima di un altro subisca più delay e venga letto più tardi, addirittura nella finestra temporale successiva

WATERMARKS

La soluzione a questo problema sono gli **Watermarks**, degli elementi generati all'interno di una stream e che contengono un timestamp uguale a:
max_timestamp - out_of_orderness - 1ms

Ogni watermark assume che tutti gli elementi letti prima siano stati generati precedentemente rispetto al timestamp del watermark stesso



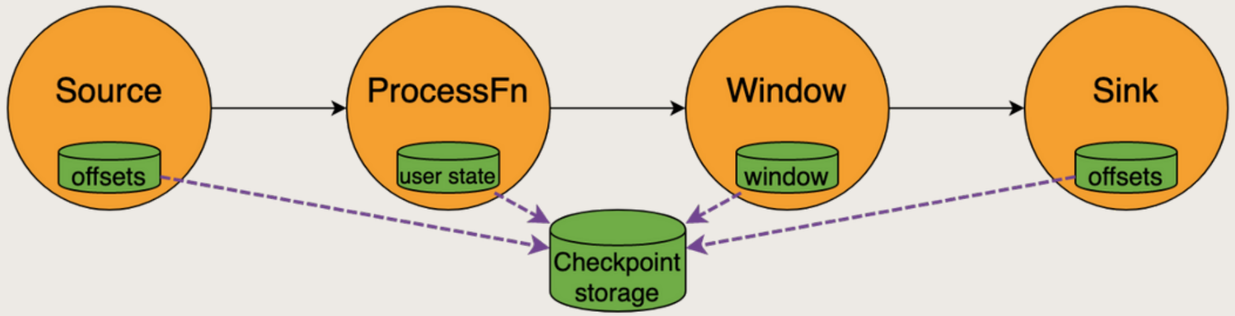
WATERMARKS

- Non sono necessari per applicazioni basate su Processing Time
- Non sono necessari per Batch Processing
- Servono per attivare azioni basate su Event Time
- Sono generati sulla base del valore out-of-orderness ipotizzato (ritardo massimo dei dati)

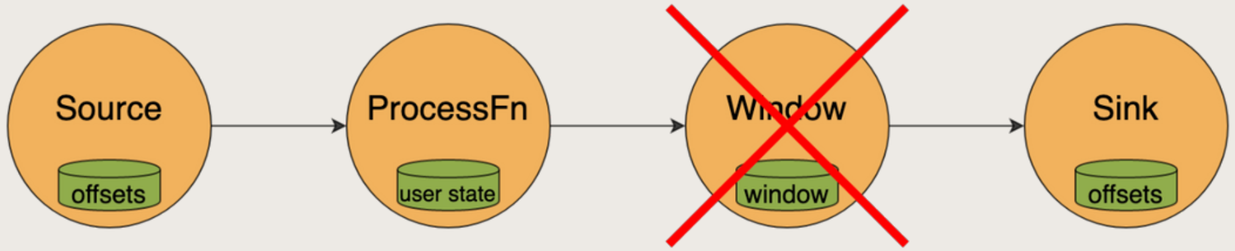
CHECKPOINTS & RECOVERY

CHECKPOINT

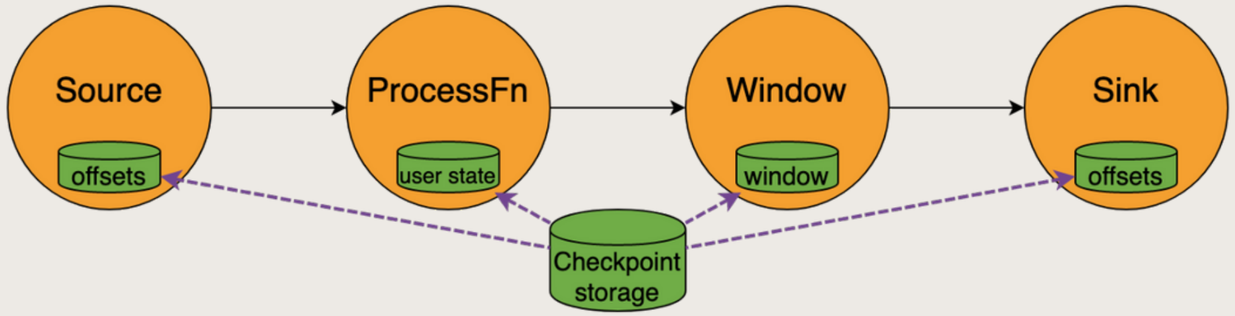
- Un checkpoint è uno **snapshot automatico** di tutti gli stati, creato da Flink
- È usato soprattutto per il **Failure Recovery**
- I checkpoint sono salvati in un **file system remoto**
- Esistono anche altri tipi di snapshot, i **savepoint**, creati manualmente
- Dopo un fail, Flink riavvia gli operator affetti per eseguire il **Recovery**



On checkpoint:
Save state to checkpoint storage



On failure:
Restart affected operators



After restart:
Recover data from checkpoint storage

THANK YOU