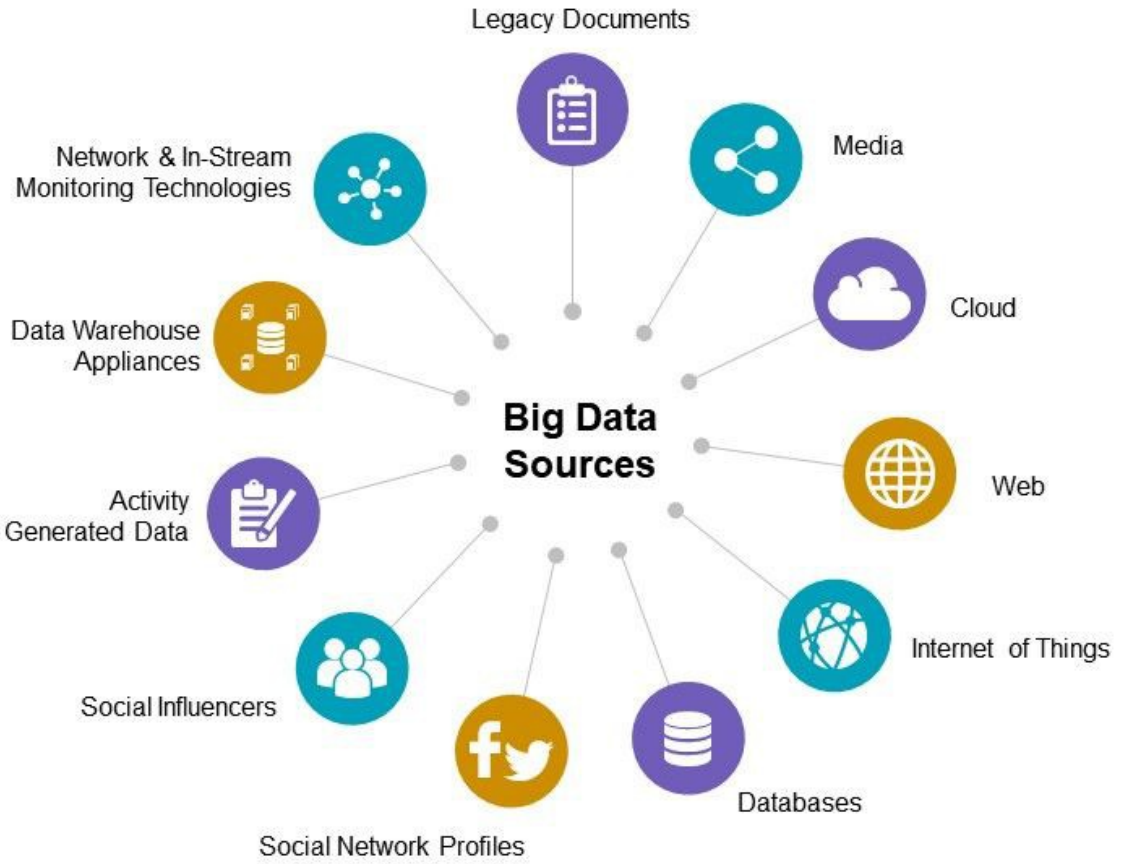
# Big Data Sources




This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

# *What Is Apache Flink ?*

- A stream processing framework
- Open source / Apache 2.0 license
- Written in Java and Scala
- For batch and stream processing
- For high volume , low latency
- Develop in Java, Scala, Python, SQL
- Automatic compilation/optimization into data flows

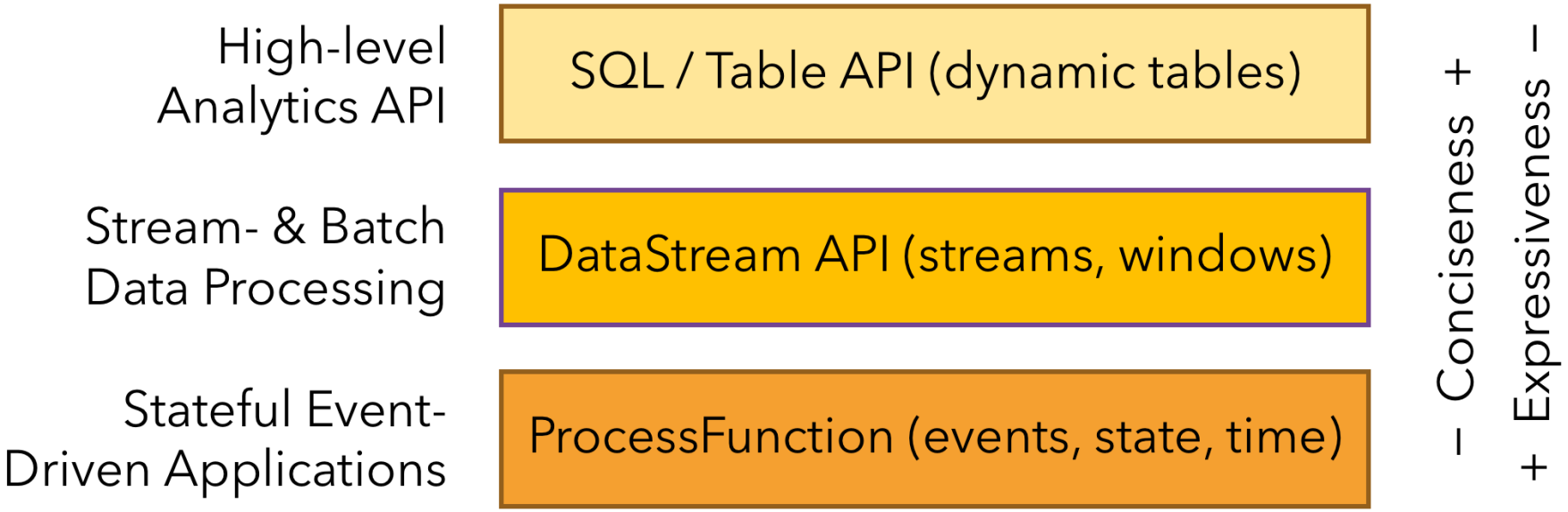
Apache Flink

# *How Does Flink Work ?*

- Process Unbounded and Bounded Data
- Uses file systems to consume/persistently store data i.e.
  - local, hadoop-compatible, Amazon S3, MapR FS, OpenStack Swift FS, Aliyun OSS and Azure Blob Storage
- Leverages In-Memory Performance
- Provides a rich function set for handling
  - Streams, state and time
  - When building applications
- Provides layered API's which provides a balance between
  - Conciseness and expressiveness
  - See next slide

# How Does Flink Work ?

Flink layered API's

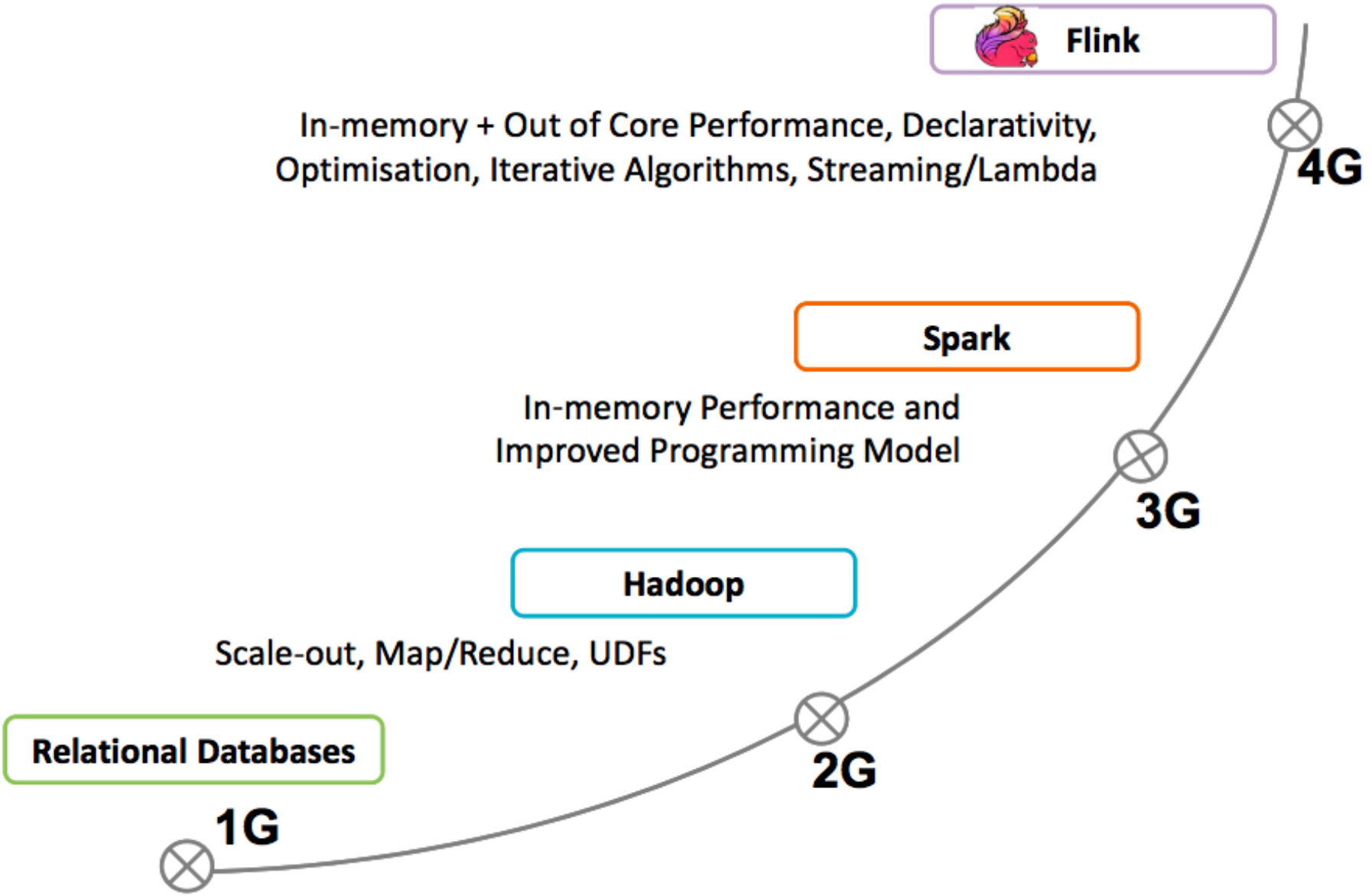


# *Flink API's*

- SQL & Table API
- DataStream API
- ProcessFunctions – event processing
- Flink also has libraries for common data processing
  - Complex Event Processing (CEP)
  - DataSet API
  - Gelly: library for scalable graph processing/analysis

# Flink Used By

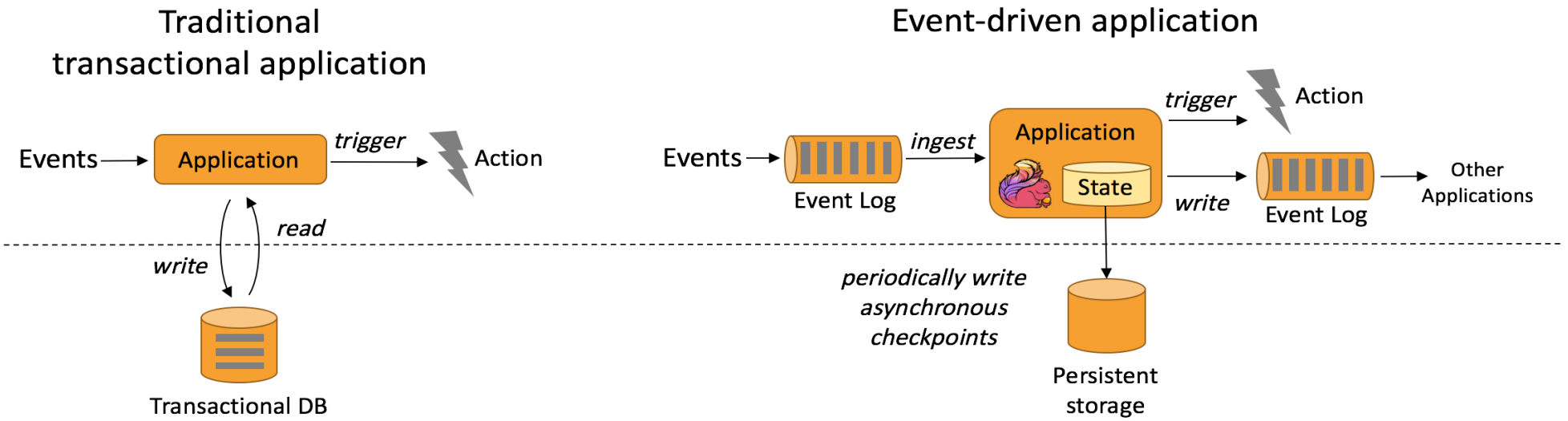
# *Evolution of Big Data Platforms*

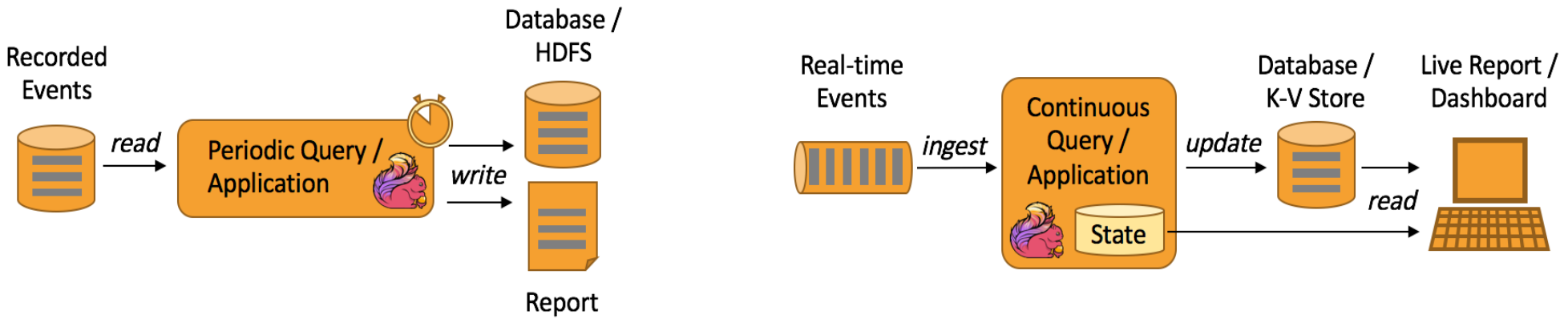


# Engine Comparison

| |  |  |  |
|---|---|---|---|
| API | MapReduce on k/v pairs | Transformations on k/v pair collections | Iterative transformations on collections |
| Paradigm | MapReduce | RDD | Cyclic dataflows |
| Optimization | none | Optimization of SQL queries | Optimization in all APIs |
| Execution | Batch sorting | Batch with memory pinning | Stream with out-of-core algorithms |

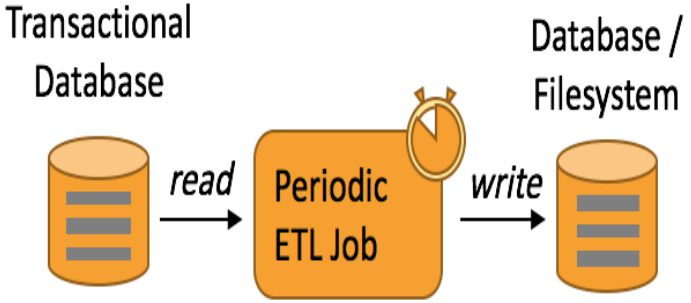
# Flink Use cases



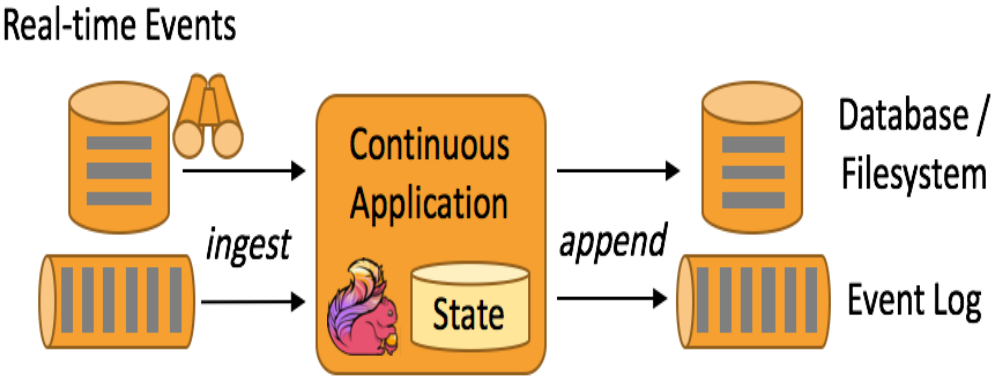



# *Flink Use cases*

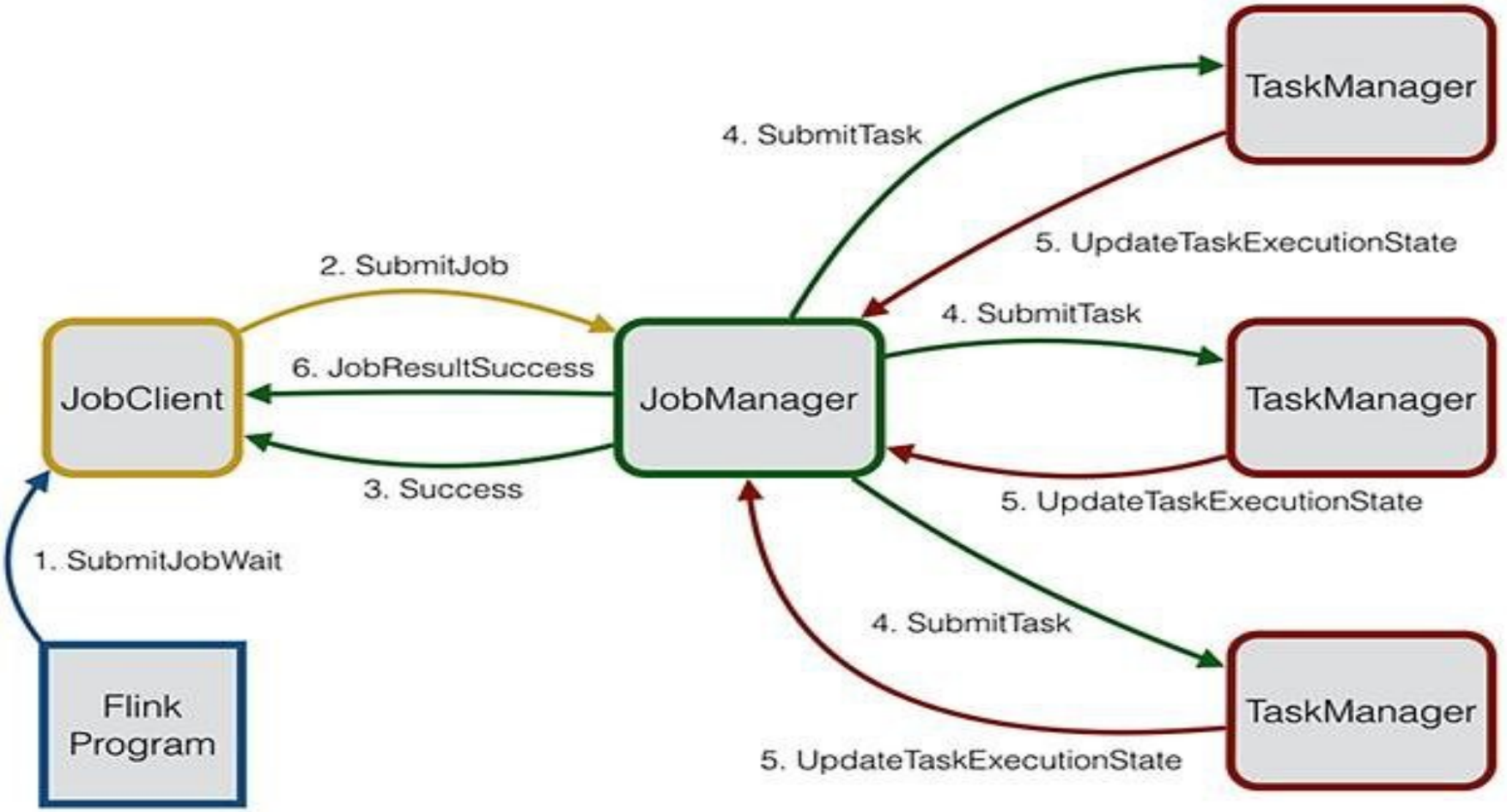
Periodic ETL



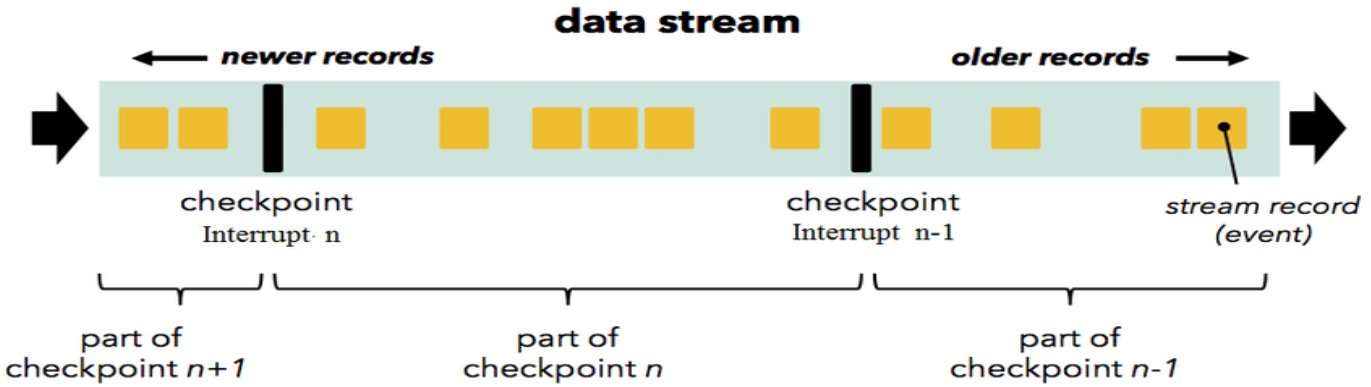

Data Pipeline



# The Flink Distributed Execution

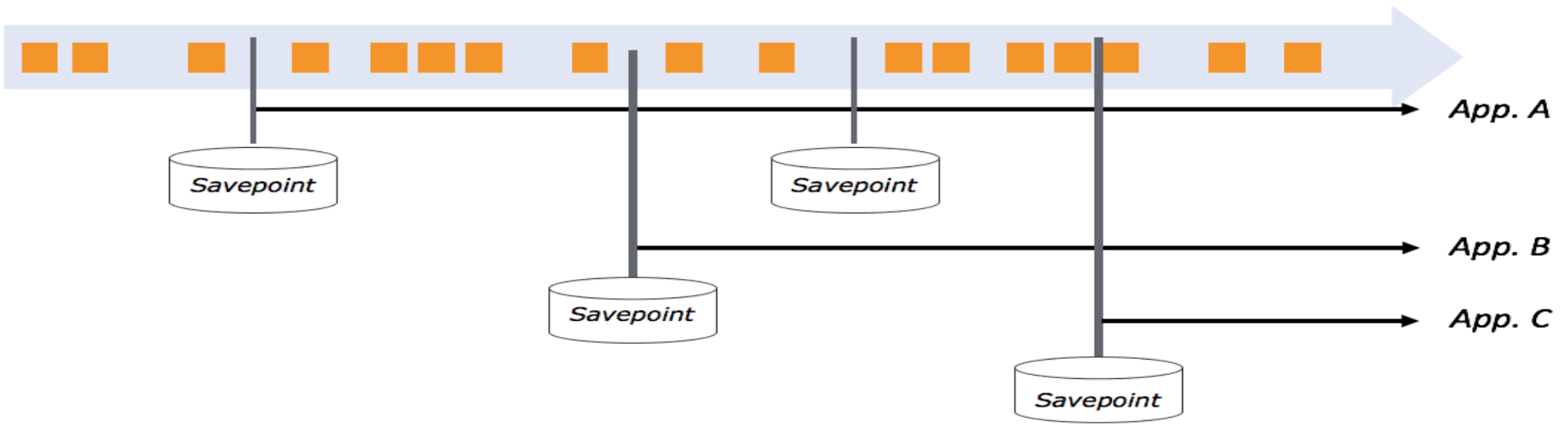


# Flink: Checkpoint and savepoint




- The checkpointing barriers split a stream into two parts: set of records that goes into the current snapshot and the records that go into the next snapshot
- The position of the barrier defines the extent of a checkpoint
- A JobManager initiates a checkpoint by sending a message with a new checkpoint ID to each data source task.
- When a data source task receives a message with a new checkpoint, it pauses its processing operation, checkpoint its local state to the remote storage and emits a checkpoint barrier into its regular output stream.
- The barrier is broadcasted to all the tasks to ensure that each task received a checkpoint from each of its input streams.
- After these steps, the task waits for the arrival of all barriers for the checkpoint, meantime continues processing records that did not contain a barrier yet.
- The records that forward an injected barrier should be buffered and not proceed
- Once all barriers arrive, the task checkpoints its operator state to the remote storage and forward a checkpoint barriers to each of its downstream connected tasks.
- After the checkpoint barrier has been emitted, the task starts processing the buffered records.

# Flink: Checkpoint and savepoint




- This mechanism offers the possibility of reprocessing and going back in time at a determinant point of the stream input that was a user's choice
- A savepoint is a globally consistent snapshot of the positions of all data sources and the state of all operators
- Checkpoints is used to recover the system in case of failure by copy- ing the application state and reading positions of the input.
- Savepoint is used to act as the way to restart, continue or reopen a paused application after manual backup and resume activity by the user.
- Checkpoints are automatic and periodic in nature because they are created and dropped by Flink without any user interaction. On the contrary, savepoints are managed manually by the user.

# *Flink: Pros*

- **High performance:**
  - high performance and low latenc
  - you don't need to do any manual configurations to get the best performance
- **Exactly-once guarantees:**
  - checkpoint processing helps to guarantee processing each record exactly once
  - After a failure, a state in stateful operators should be correctly restored.
- **Flow control:**
  - Backpressure from slow operators should be absorbed by the system to avoid crashes
- **Fault tolerance:**
  - snapshot mechanism helps in achieving a great degree of fault tolerance.
- **Event time semantics:**
  - Flink's allows to define windows based on time, counts, and sessions, which helps in dealing with such scenarios.
- **Memory management:**
  - Flink is supplied with its own memory man-agement inside a JVM which makes it independent of Java's default garbage collector.
- **Stream and batch in one platform:**
  - Flink provides APIs for both batch and stream data processing

# *Flink:Cons*

- **Less community and forums for discussion:**
  - Flink may be difficult to understand starting as a beginner
  - there are not many active communities and forums to exchange problems

- **Less open-source projects:**
  - There are not many open-source projectd to study and practice Flink

- **Immaturity:**
  - Immaturity in the industry is a disadvantage for Apache Flink, because is a new technology and many features are constantly being updated and modified.

- **API support:**
  - Flink is not the best choice if we need more API support apart from Java and Scala languages.

- **Data representation:**
  - Flink uses raw bytes as internal data represen- tation, which if needed, can be hard to program.