High-throughput, low-latency, and exactly-once stream processing with Apache Flink

Sep 15

Zehao Wu

streaming infrastructure

- popularity of stream data platforms is skyrocketing
- two main properties
- high throughput across a wide spectrum of latencies
- strong consistency guarantees even in the presence of stateful computations

What makes a good stream system?

- Exactly-once guarantees
- Low latency
- High throughput
- Powerful computation model
- Low overhead of the fault tolerance mechanism in the absence of failures
- Flow control

Record acknowledgements

- Apache Storm
- sends back to the previous an acknowledgement
- source keeps a backup of all the tuples
- received acknowledgements from all, discarded
- not all have been received, replayed

Micro batches

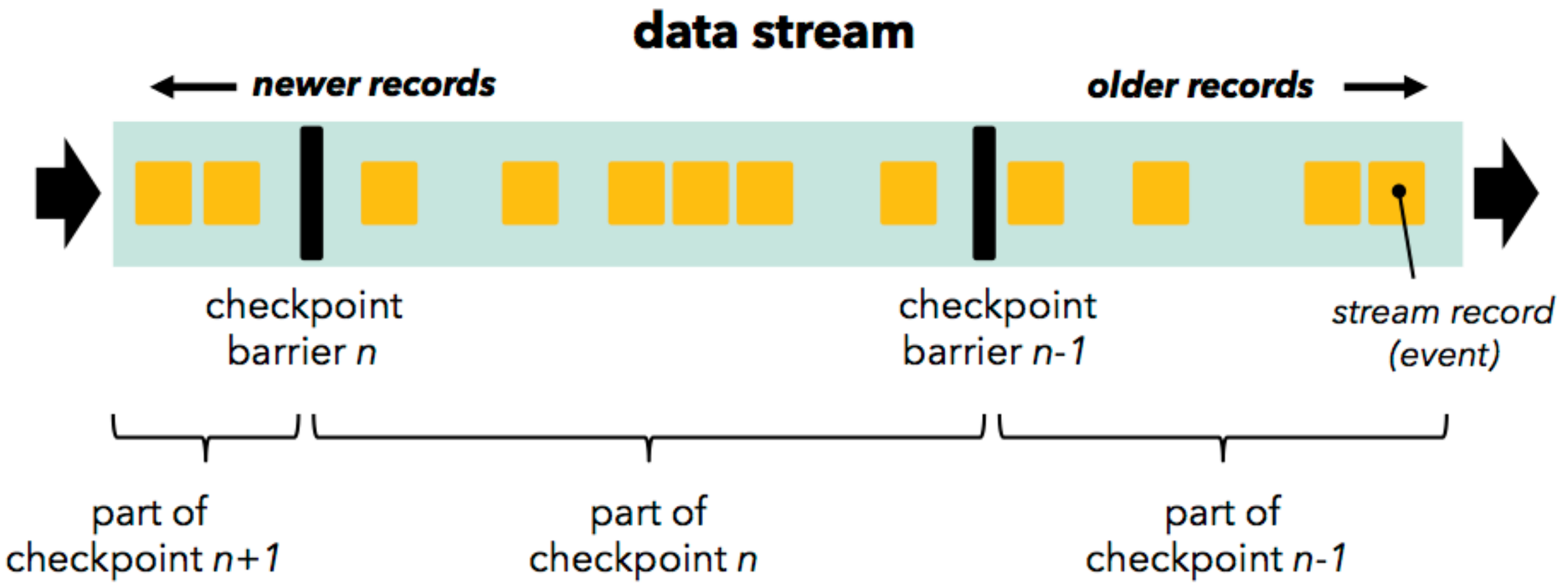
- Apache Storm Trident, Apache Spark Streaming
- broken down in a series of small, atomic batch jobs called micro-batches
- either succeed or fail
- at a failure, the latest can be simply recomputed

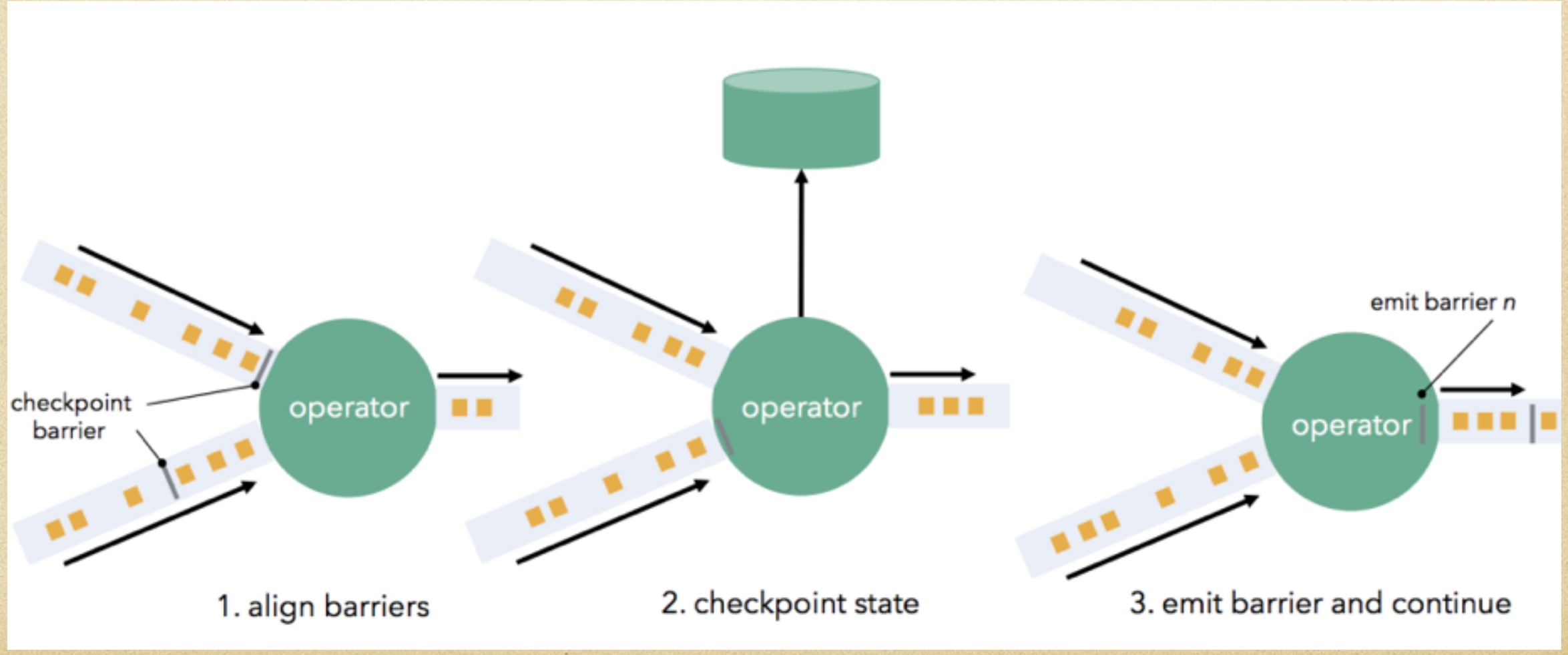
Transactional updates

- Google Cloud Dataflow
- Details will be covered in the next presentation

Distributed Snapshots

- Apache Flink
- Chandy Lamport algorithm, regular processing keeps going, while checkpoints happen in the background
- draw a consistent snapshot of that state
- storing that snapshot in durable storage
- restore from durable storage, rewind the stream source and hit the play button again


data stream
newer records
older records
checkpoint barrier n
checkpoint barrier n-1
stream record (event)
part of checkpoint n+1
part of checkpoint n
part of checkpoint n-1


checkpoint barrier
operator
operator
operator
emit barrier n
1. align barriers
2. checkpoint state
3. emit barrier and continue

	Record acks (Storm)	Micro-batching (Spark Streaming, Trident)	Transactional updates (Google Cloud Dataflow)	Distributed snapshots (Flink)
Guarantee	At least once	Exactly once	Exactly once	Exactly once
Latency	Very Low	High	Low (delay of transaction)	Very Low
Throughput	Low	High	Medium to High (Depends on throughput of distributed transactional store)	High
Computation model	Streaming	Micro-batch	Streaming	Streaming
Overhead of fault tolerance mechanism	High	Low	Depends on throughput of distributed transactional store	Low
Flow control	Problematic	Problematic	Natural	Natural
Separation of application logic from fault tolerance	Partially (timeouts matter)	No (micro batch size affects semantics)	Yes	Yes

Lacking

- High availability
- Event time and watermarks
- Improved monitoring of running jobs

Pros

- Fast
- Reliable
- Expressive
- Easy to use
- Scalable
- Hadoop-compatible

Cons

- Too many traditional database designs
- forced declarative
- harder to program
- the community not as active as Spark