BDAPRO

APIs and Execution of Dataflow Programs

Chen Xu, Alireza RM, Quoc Cuong To

Slides prepared by Jonas Traub, Gábor Gévay, Alexander Alexandrov

Agenda

Theory

- Principles of parallelization frameworks (MapReduce)
- Principles and Execution Aspects of Dataflow Programs
- Comparison of runtime concepts

Practice

- Flink Batch Processing API
- Flink Stream Processing API

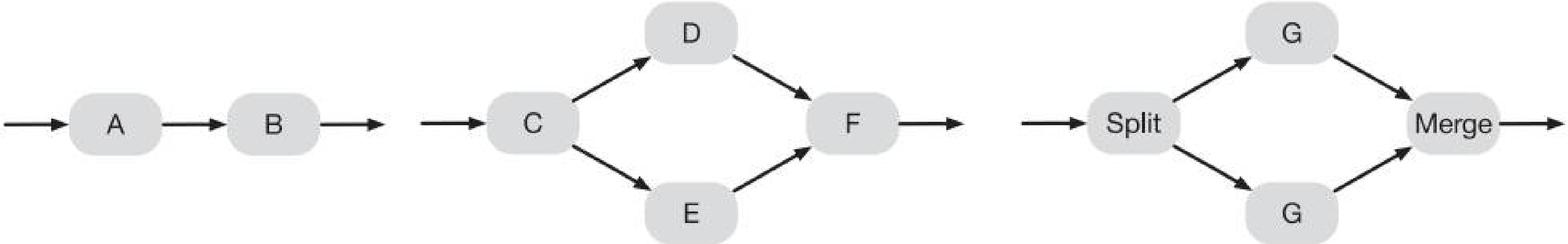
BDAPRO

Principles of Parallelization Frameworks

Chen Xu

Slides prepared by Jonas Traub, Gábor Gévay, Alexander Alexandrov

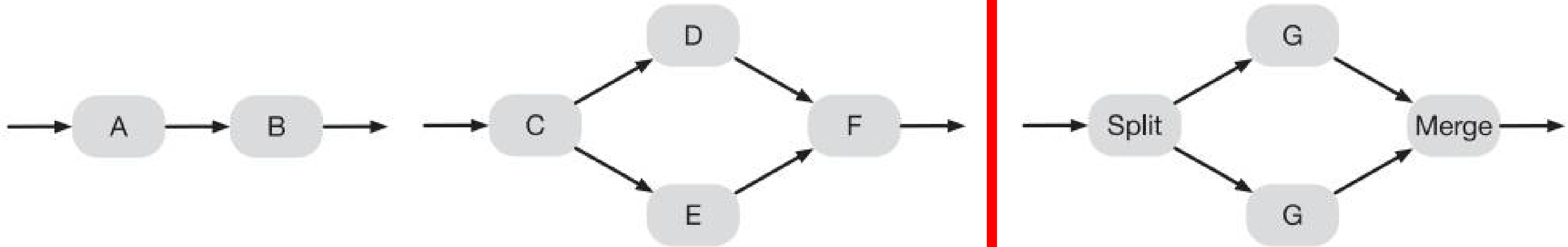
3 Types of Parallelization



(a) Pipeline-parallel A || B. (b) Task-parallel D || E. (c) Data-parallel G || G.

Source: Hirzel, M., Soulé, R., Schneider, S., Gedik, B., & Grimm, R. (2014). **A catalog of stream processing optimizations.** *ACM Computing Surveys (CSUR)*, *46*(4), 46.

3 Types of Parallelization

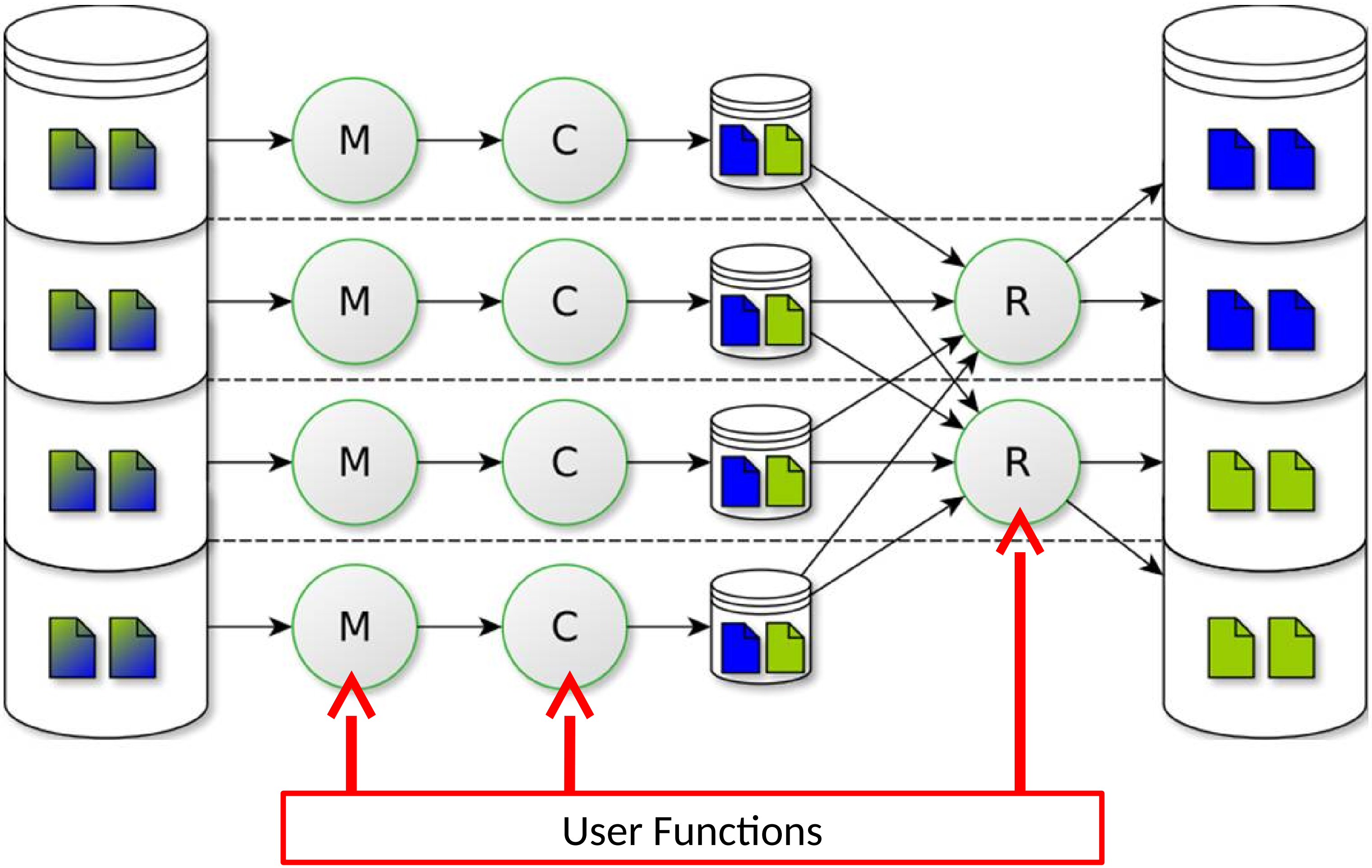


(a) Pipeline-parallel A || B. (b) Task-parallel D || E. (c) Data-parallel G || G.

"Big Data" requires **data** parallelism!

Source: Hirzel, M., Soulé, R., Schneider, S., Gedik, B., & Grimm, R. (2014). **A catalog of stream processing optimizations.** *ACM Computing Surveys (CSUR)*, *46*(4), 46.

MapReduce



MapReduce Paper: Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

MapReduce Example (Word Count)

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

MapReduce Paper: Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

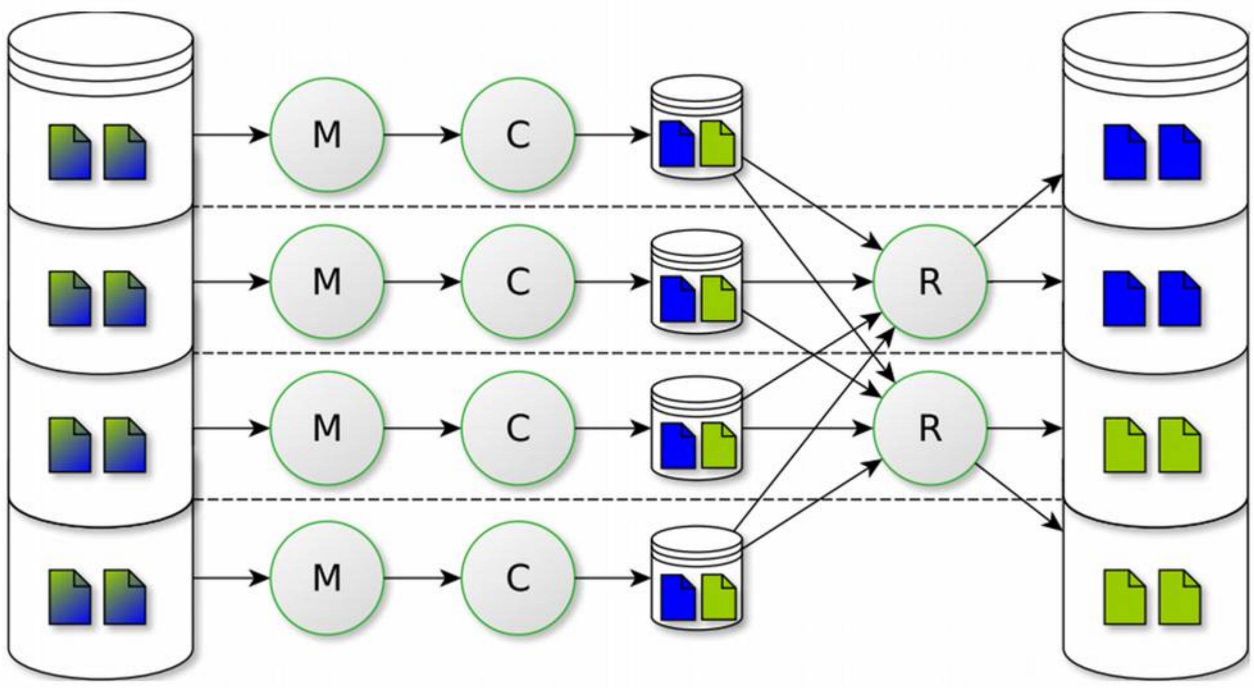
MapReduce Example (Word Count)

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

- Stateless functions!
- As many mapper instances as input (K,V) pairs.
- As many reducer instances as distinct keys in the output of the map phase
- Synchronization point and shuffling between Map and Reduce phase

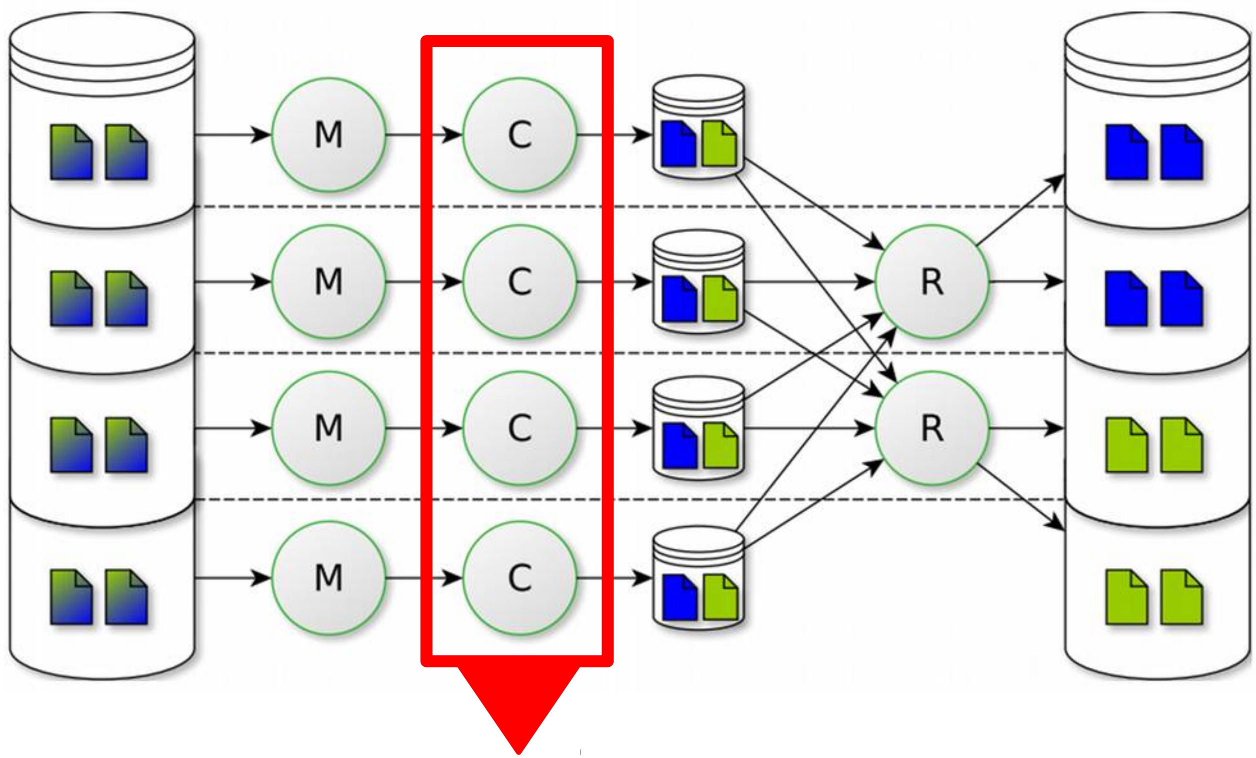
MapReduce Paper: Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

MapReduce Example (Word Count)



	Input	Output
File Line 1:	Beer Beer Tea Coffee	
File Line 2:	Tea Tea Beer Tea	
Map 1:	(1, Beer Beer Tea Coffee)	[(Beer,1),(Beer,1),(Tea,1),(Coffee,1)]
Map 2:	(2, Tea Tea Beer Tea)	[(Tea,1),(Tea,1),(Beer,1),(Tea,1)]
Reduce 1	(Beer,[1,1,1])	(Beer,3)
Reduce 2	(Coffee,[1])	(Coffee,1)
Reduce 3	(Tea,[1,1,1,1])	(Tea,4)

MapReduce Example (Word Count)



The combine function:

- Extension to the plain MapReduce model
- Allows local pre-aggregation
  - Several combine instance may be present
    for each distinct key in the output of the map phase.
  - No synchronization point is required
    between Map and Combine.

BDAPRO
Flink API Presentations

Chen Xu, Alireza RM, Quoc Cuong To

Slides prepared by Jonas Traub, Gábor Gévay, Alexander Alexandrov

Flink Online Training by Data Artisans

Lectures, Hands-On Tasks, and Reference Solutions:

http://dataartisans.github.io/flink-training/index.html

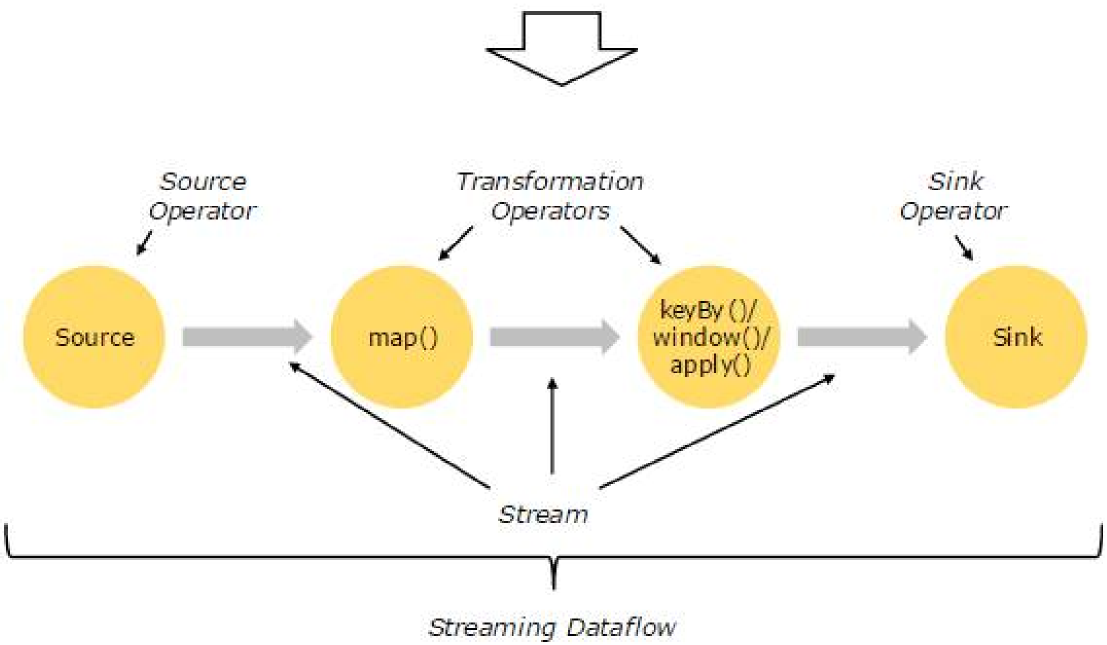
BDAPRO
Principles and Execution Aspects of Dataflow Programs

Alireza RM

Slides prepared by Jonas Traub, Gábor Gévay, Alexander Alexandrov

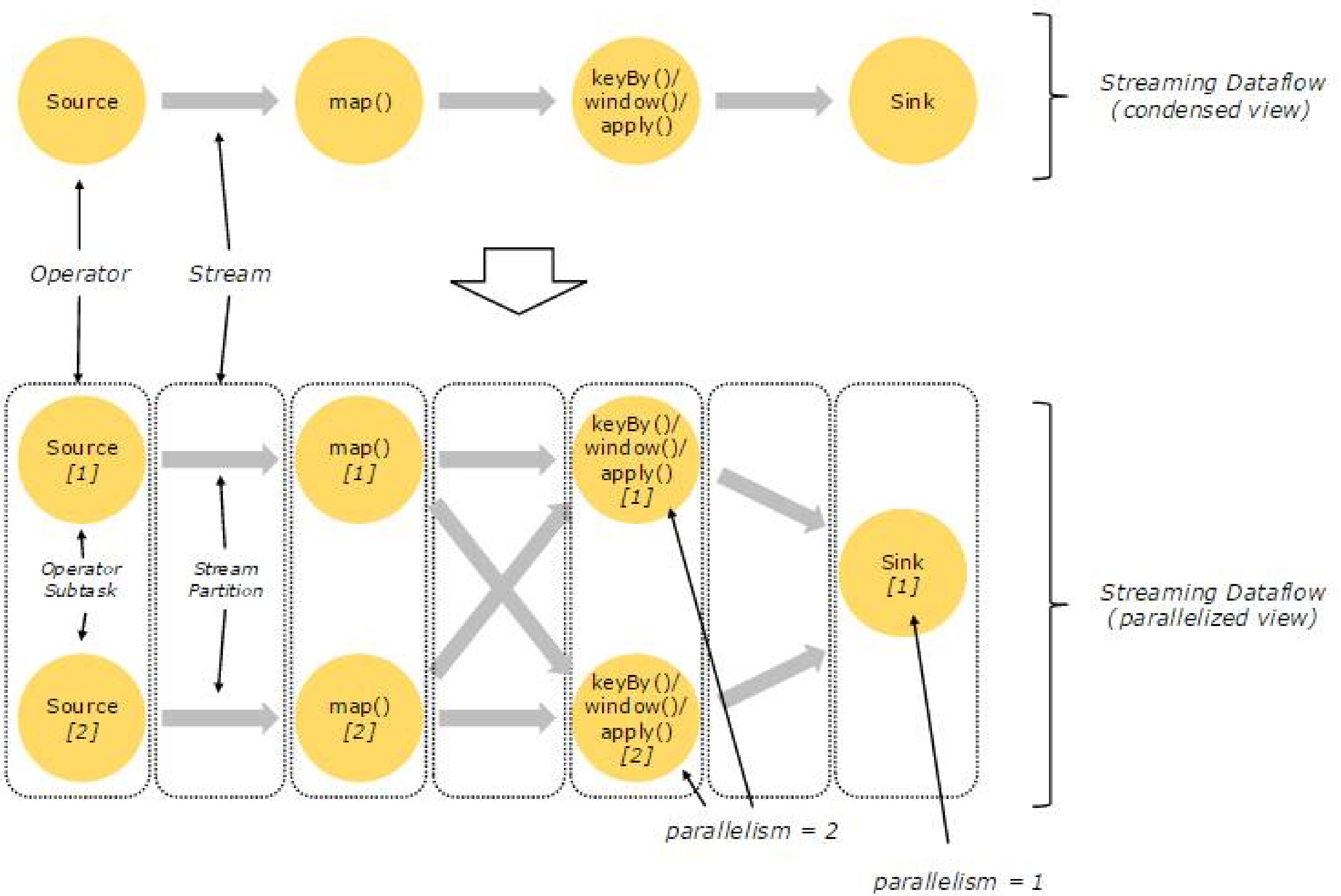
Programs and Dataflows

```
DataStream<String> lines = env.addSource(                                  Source
                              new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));              Transformation

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))                                      Transformation
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));                                      Sink
```



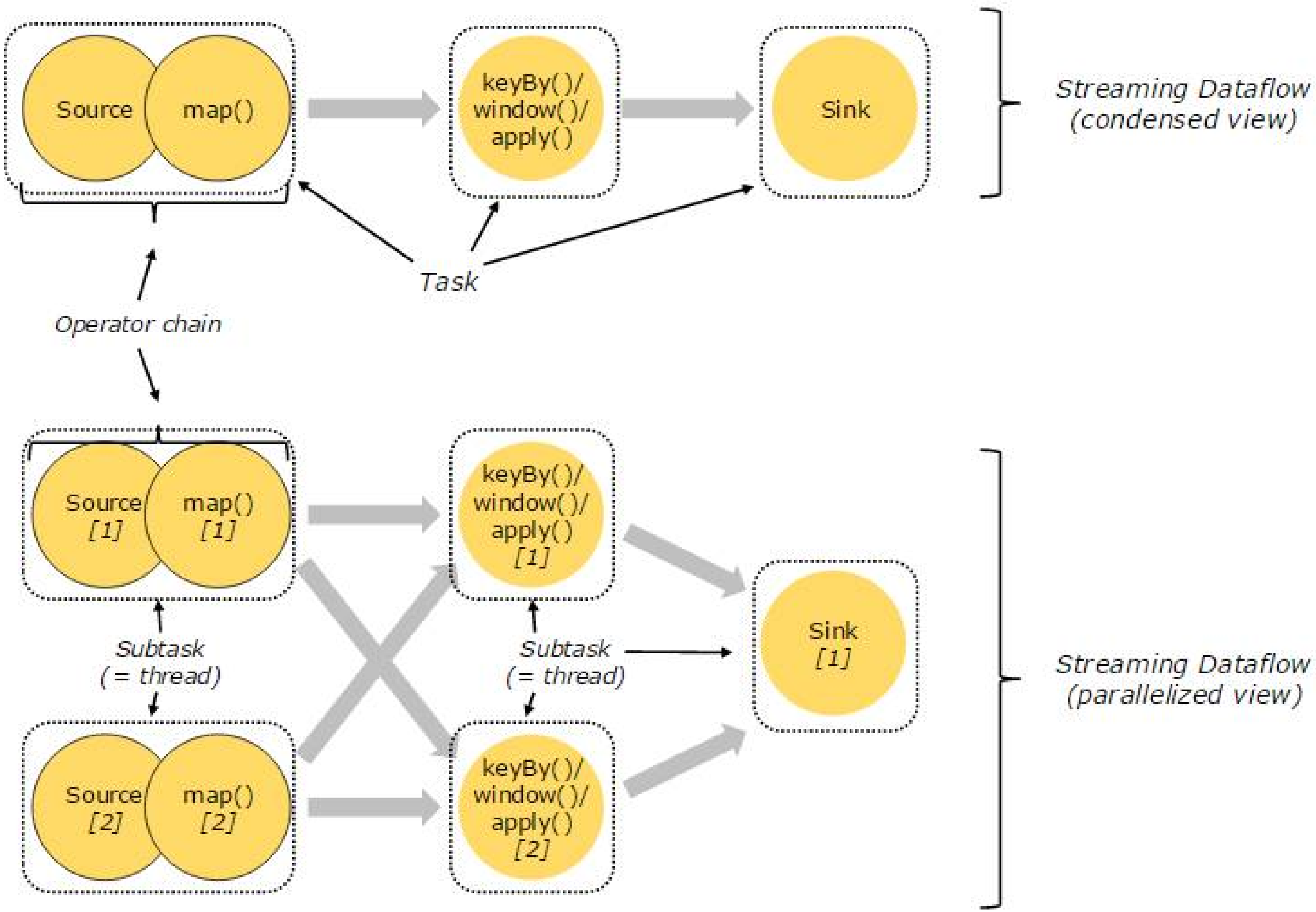
Source: https://ci.apache.org/projects/flink/flink-docs-master/concepts/concepts.html

Parallel Dataflows



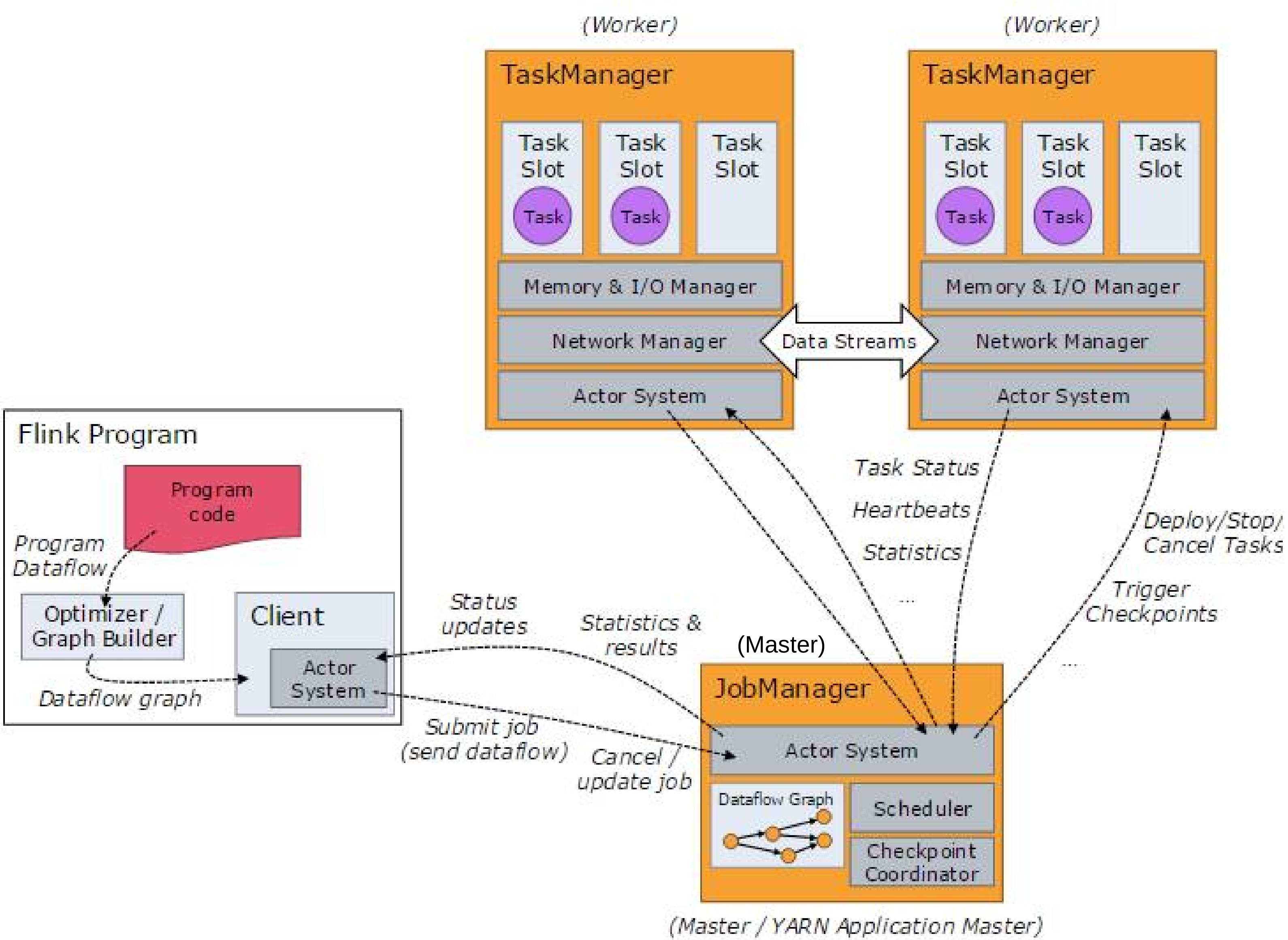
Source: https://ci.apache.org/projects/flink/flink-docs-master/concepts/concepts.html

Tasks & Operator Chains



Source: https://ci.apache.org/projects/flink/flink-docs-master/concepts/concepts.html

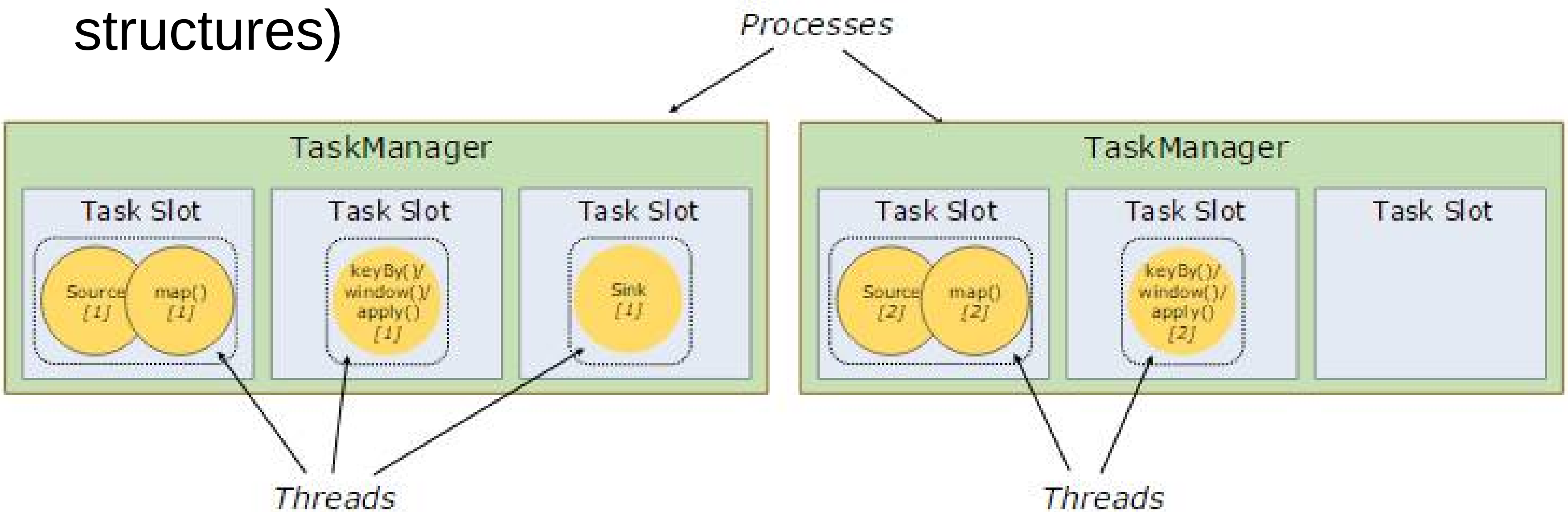
Distributed Execution (Flink Runtime)



Source: https://ci.apache.org/projects/flink/flink-docs-release-1.2/concepts/runtime.html

Workers, Slots, and Resources

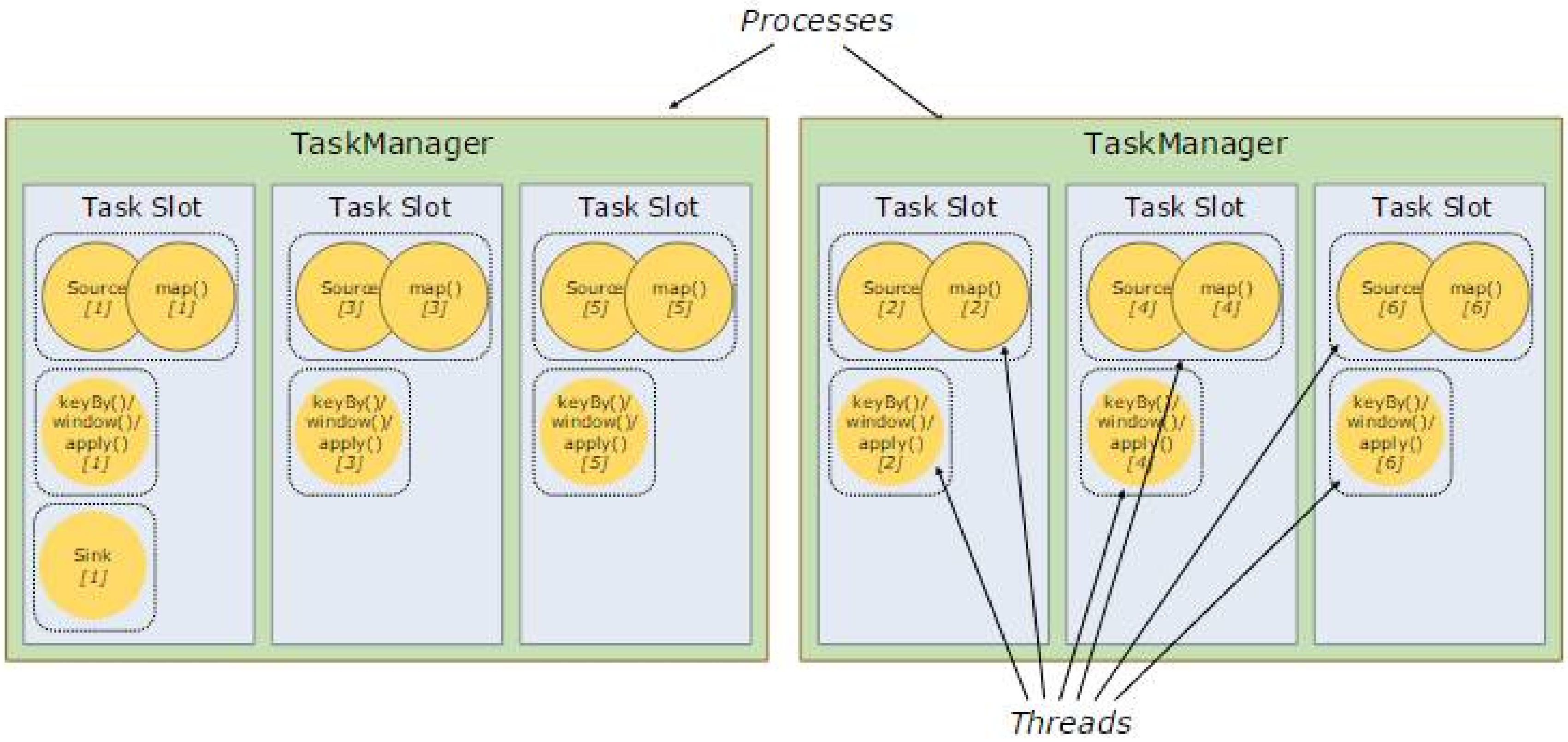
- Workers are JVM processes executing one or more tasks in separate threads
- Each worker has one or more task slots with fixed resources, e.g., memory
- One slot TaskManager runs in separate JVM otherwise subtasks share same JVM (share TCP, heartbeats, data structures)



Source: https://ci.apache.org/projects/flink/flink-docs-release-1.2/concepts/runtime.html

Workers, Slots, and Resources

- Subtasks of different tasks of same job can share slots
  - No need to calculate number of tasks of a program
  - Resource utilization by distributing heavy subtasks between TaskManagers



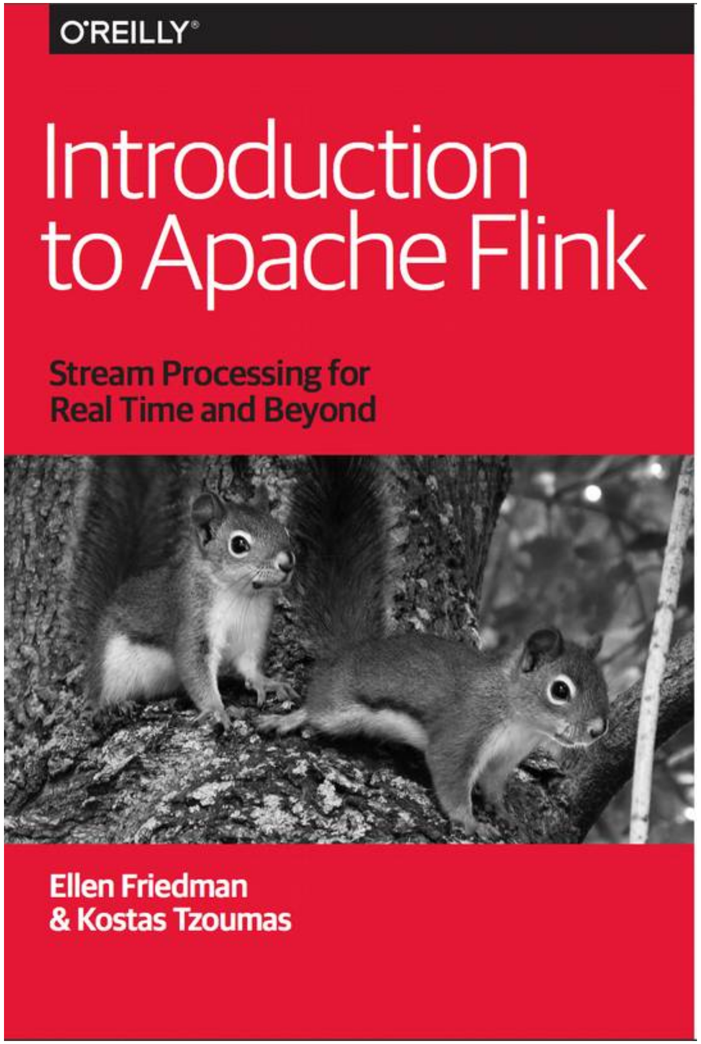
Source: https://ci.apache.org/projects/flink/flink-docs-release-1.2/concepts/runtime.html

Concepts of Dataflow Programs

Please find the full article in the Apache Flink Documentation:

https://ci.apache.org/projects/flink/flink-docs-master/concepts/concepts.html

Apache Flink: Recommended Reading



Introduction to Apache Flink

English; e-book; September 2016

Authors:
- Ellen Friedman
- Kostas Tzoumas

Available free
https://www.mapr.com/introduction-to-apache-flink

BDAPRO

Comparison of Runtime Concepts

Quoc Cuong To

Slides prepared by Jonas Traub, Gábor Gévay, Alexander Alexandrov

Pipelined vs. Batch Execution

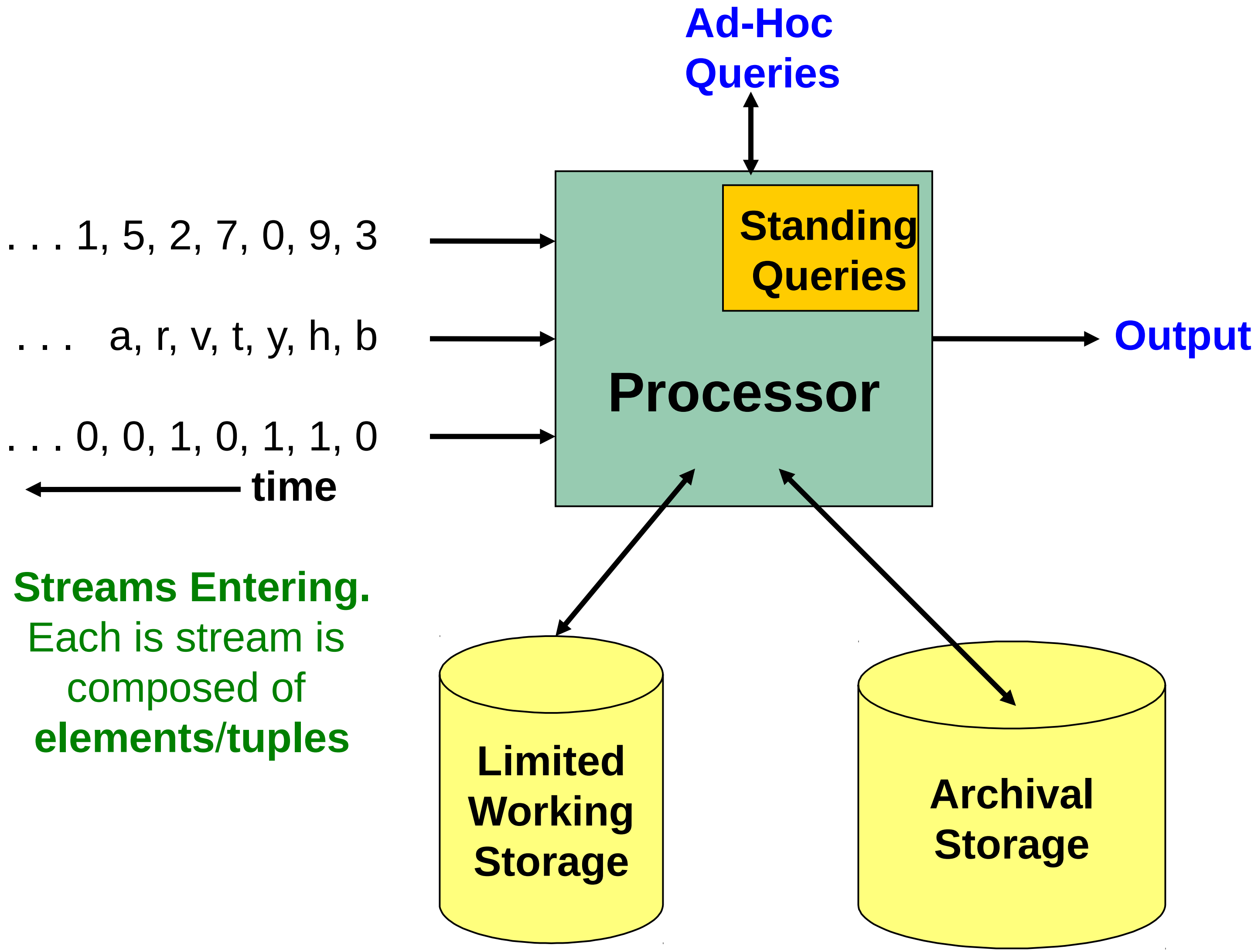
Batch Execution

- Finite data
- Allows synchronization points (w/o windowing)
- Stream processing in micro-batches

Pipelined Execution

- Conceptually infinite input data streams
- Enables low latency processing
- Native streaming support

The General Stream Processing Model



Streams Entering.
Each is stream is composed of **elements/tuples**

Souce: Rajaraman, A., & Ullman, J. D. (2012). Mining of massive datasets (Vol. 77). Cambridge: Cambridge University Press. Chapter 4 http://www.mmds.org/

Stream Processing vs. Batch Processing

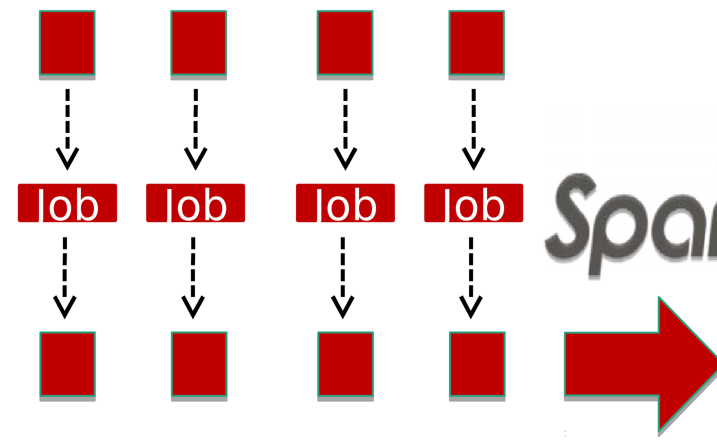
	Batch Processing	Stream Processing
INBOUND DATA	Data-items are pulled from storage as needed	Data-items are pushed to the system (externally controlled src.)
OPERATORS	Computation in stages; Operators run one after another Outputs are materialized in memory or on disk between stages	Full job graph is deployed; Long running operators Output data-items are directly sent to the next operator
QUERIES	Finite: Finished after the batch is processed	Long running: Continuously produce results for windows
RUNTIME	True streaming is not possible on a batch processing runtime	Batch processing can be done on a stream processing runtime

Native Streaming vs. D-Streams

Discretized Streams (D-Streams)

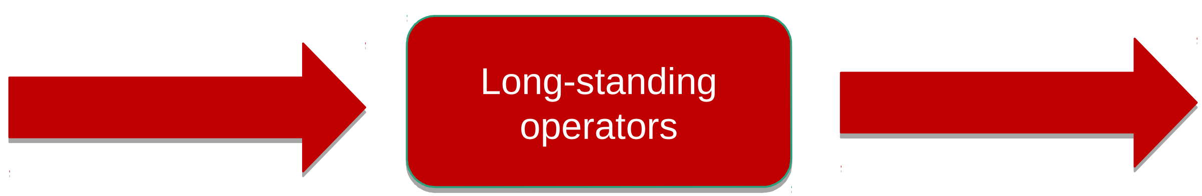
Paper by Zaharia, Matei, et al.:
"Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters." 2012.

Stream discretizer




```
while (true) {
  // get next few records
  // issue batch computation
}
```

Native streaming




```
while (true) {
   // process next record
}
```

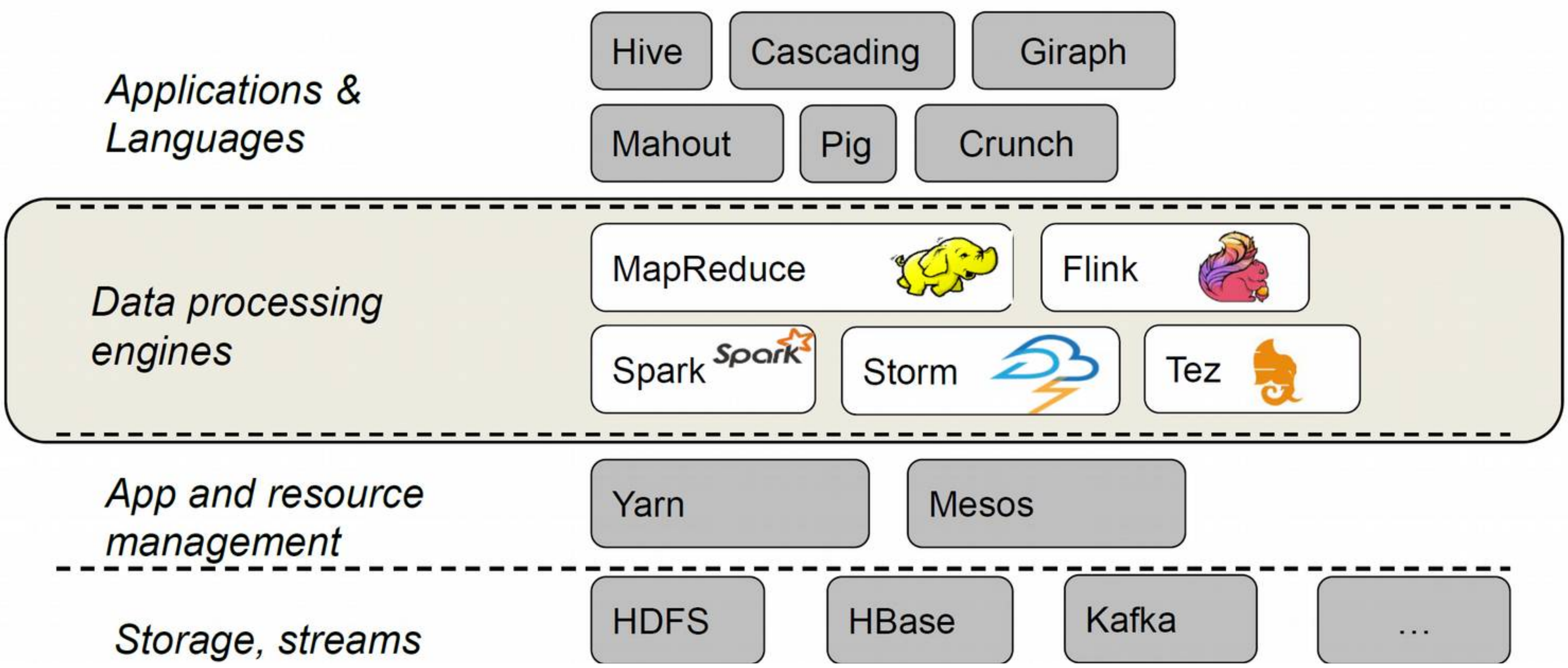
Slide by Gyula Fóra

Problems of Mini-Batch

- **Latency**: Each mini-batch schedules a new job, loads user libraries, establishes DB connections, etc
- **Programming model**: Does not separate business logic from recovery – changing the mini-batch size changes query results
- **Power**: Keeping and updating state across mini-batches only possible by immutable computations

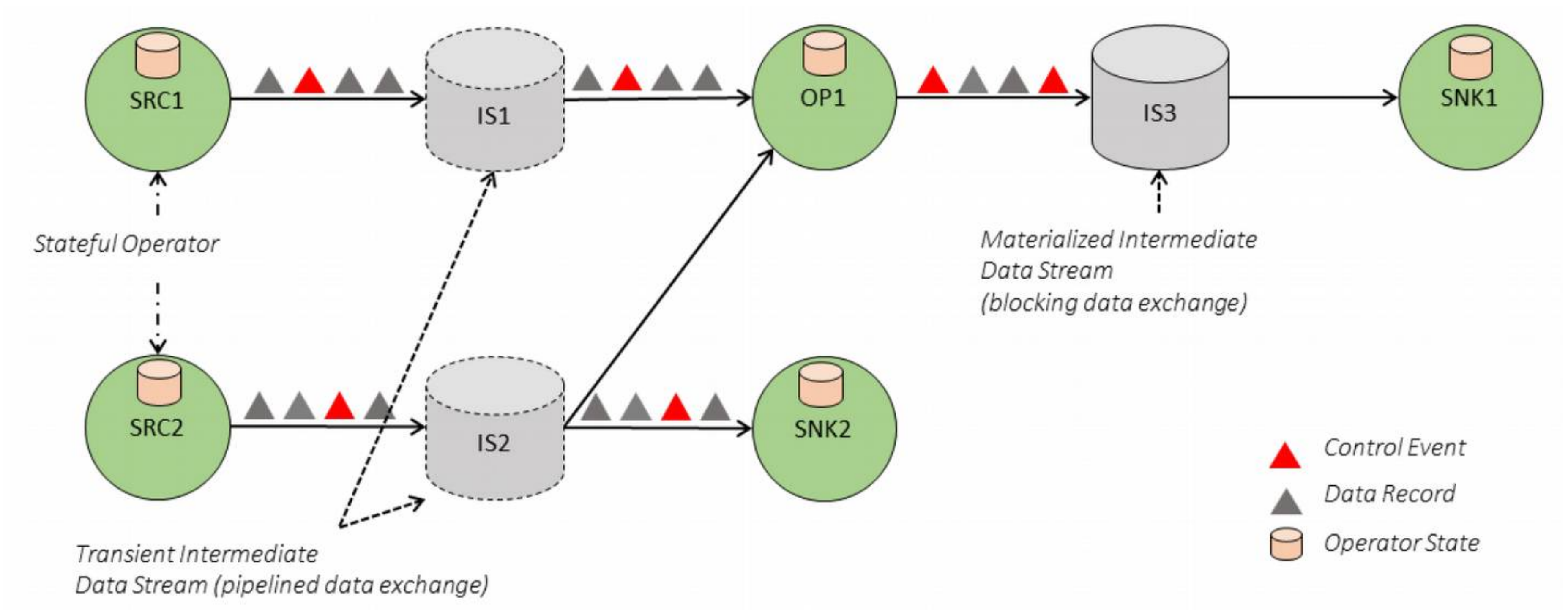
Slide by Tilmann Rabl

Flink in the Analytics Ecosystem



Flink Execution Model

- Flink program = DAG of operators and intermediate streams
- Operator = computation logic + state
- Intermediate streams = logical stream of records

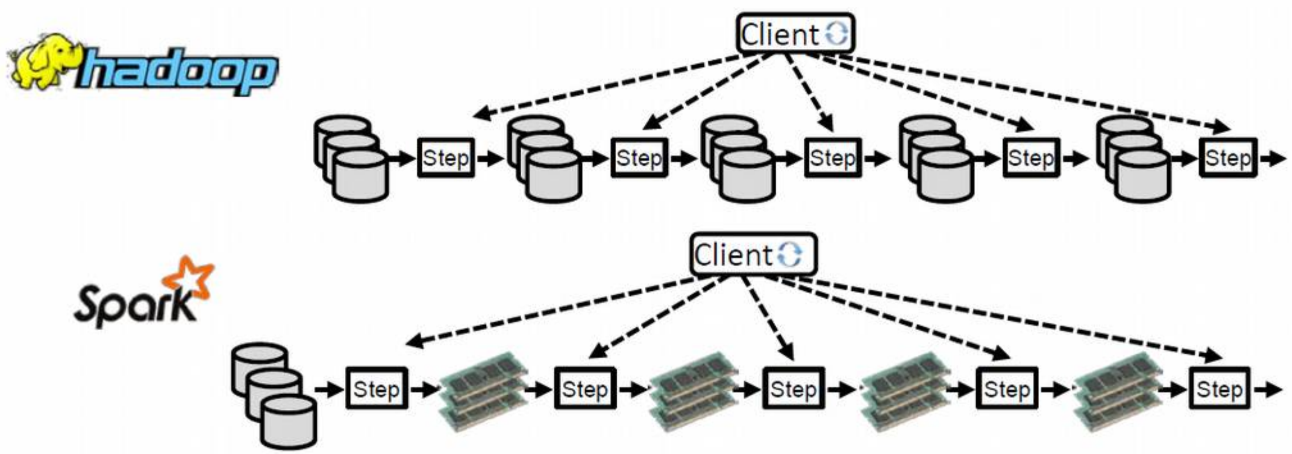


Built-in vs. driver-based looping

Driver-based looping

Loop outside the system, in driver program

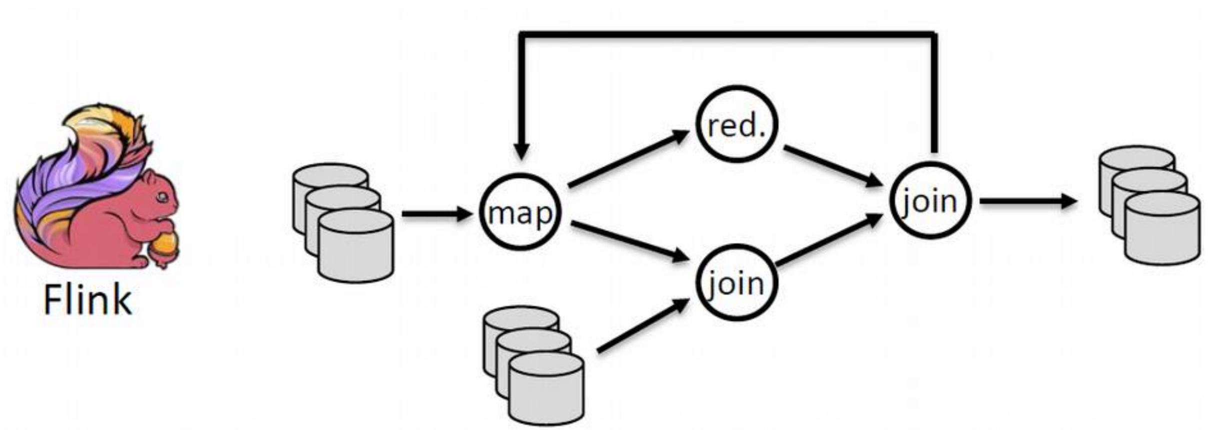
Iterative program looks like many independent jobs



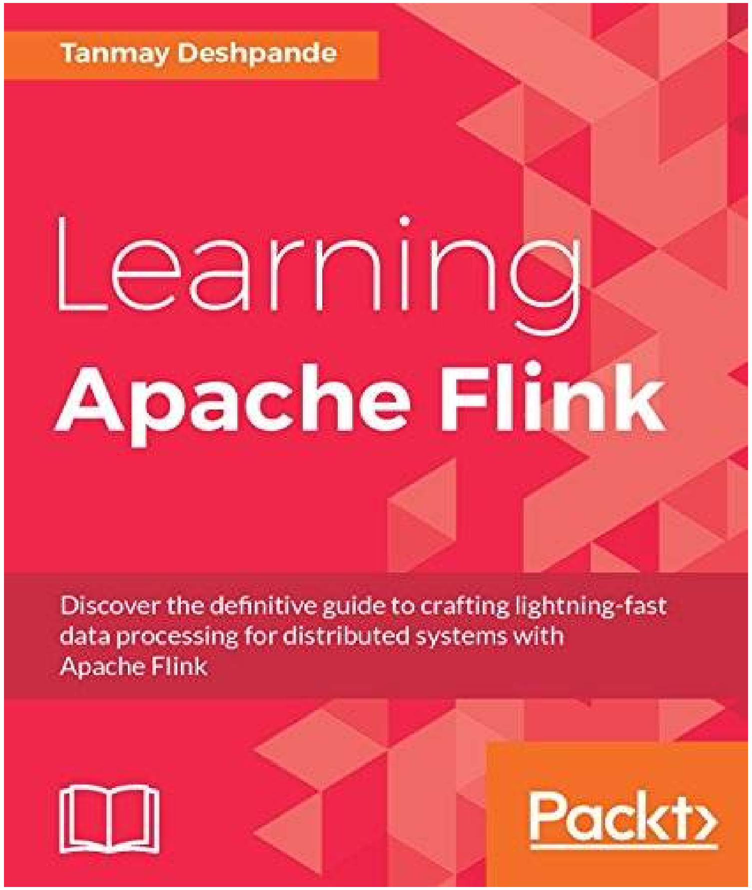
Native looping

Dataflows with feedback Edges

System is iteration-aware, can optimize the job



Apache Flink: Recommended Reading



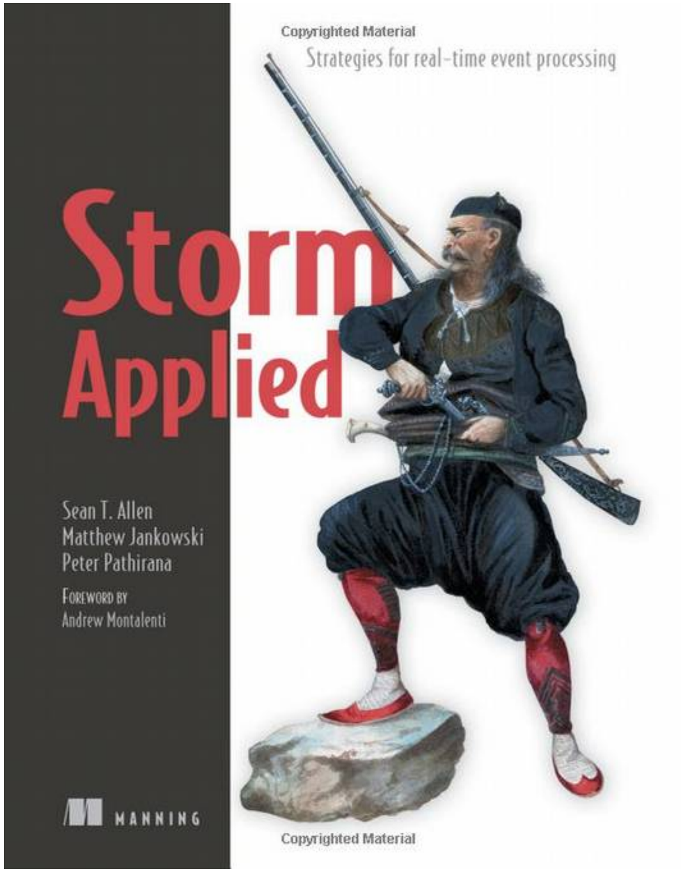
Learning Apache Flink

English; Paperback;

Authors:

Tanmay Deshpande

Apache Storm: Recommended Reading



Storm Applied: Strategies for Real-Time Event Processing

English; Paperback; April 2015

Authors:

- Sean T. Allen
- Peter Pathirana
- Matthew Jankowski

Available in TU-Berlin library

http://portal.ub.tu-berlin.de/TUB:TUB_LOCAL:tub_aleph002091017

BDAPRO Project Presentations

Chen Xu, Alireza RM, Quoc Cuong To