BDAPRO

APIs and Execution of Dataflow Programs

Jonas Traub, Gábor Gévay, Alexander Alexandrov

Today's lecture

Theory

- Principles of parallelization frameworks (MapReduce)
- Principles of dataflow programs
- Execution aspects of dataflow programs
- Comparison of runtime concepts

Practice

- Flink Batch Processing API
- Flink Stream Processing API

Project Pitches

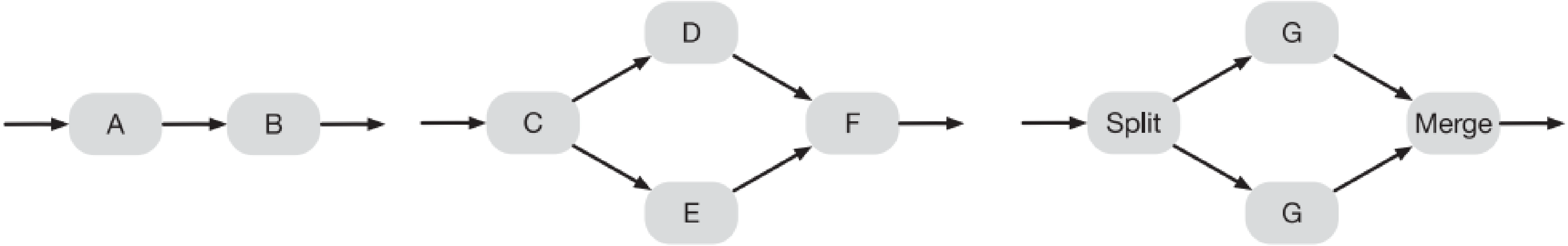
- Pitch of large tasks

BDAPRO

Principles of Parallelization Frameworks

Jonas Traub, Gábor Gévay, Alexander Alexandrov

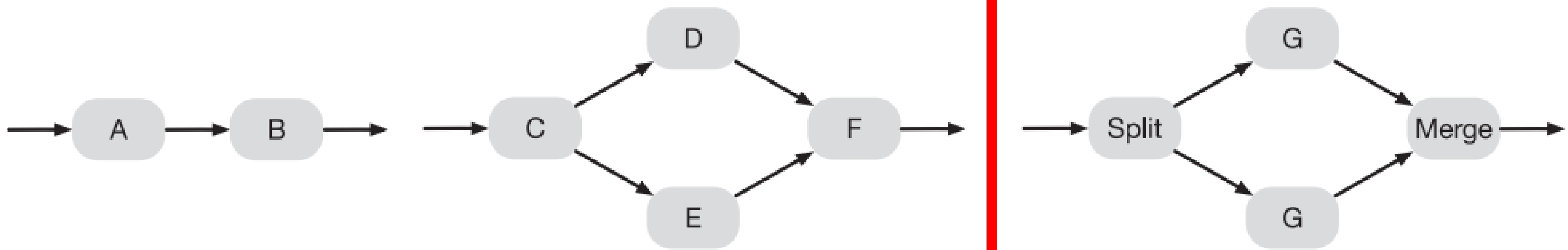
3 Types of Parallelization



(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

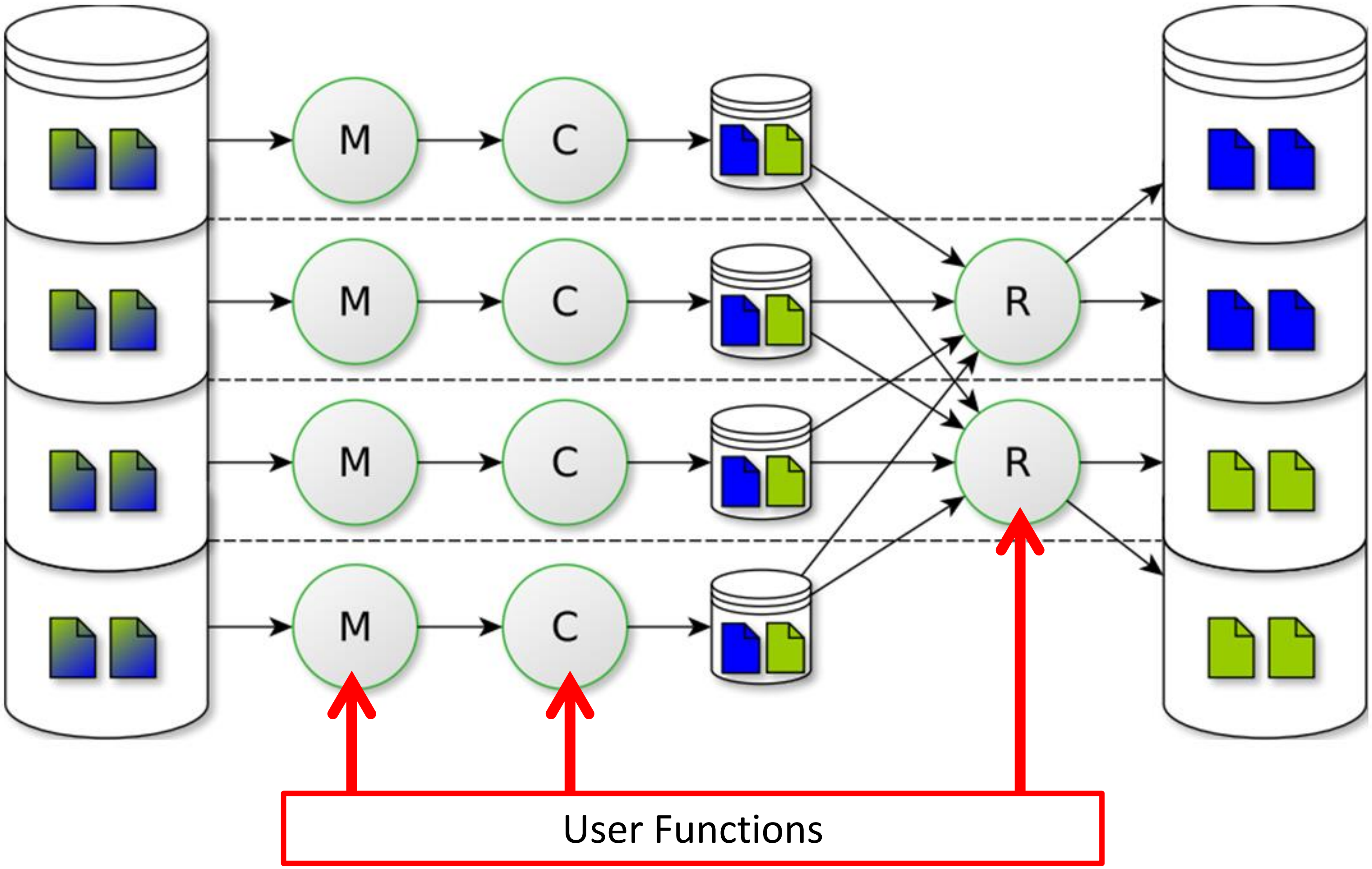


(a) Pipeline-parallel A || B. (b) Task-parallel D || E. (c) Data-parallel G || G.

"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.

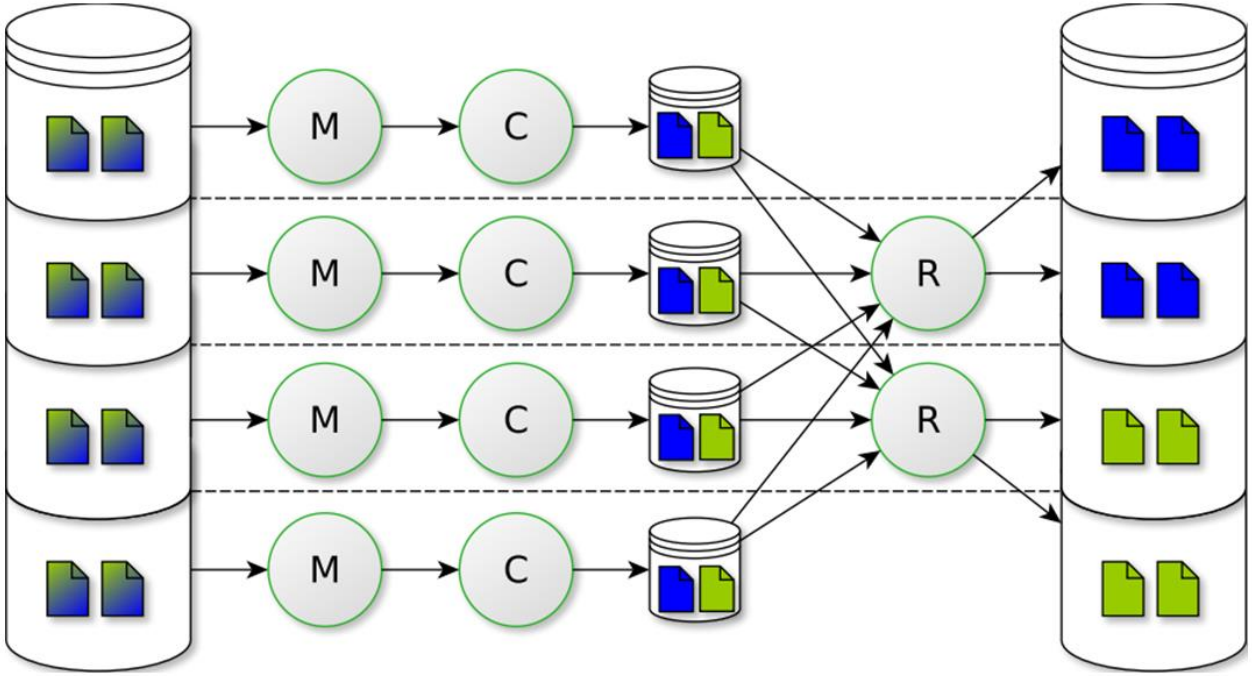
MapReduce Example (Word Count)

```
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

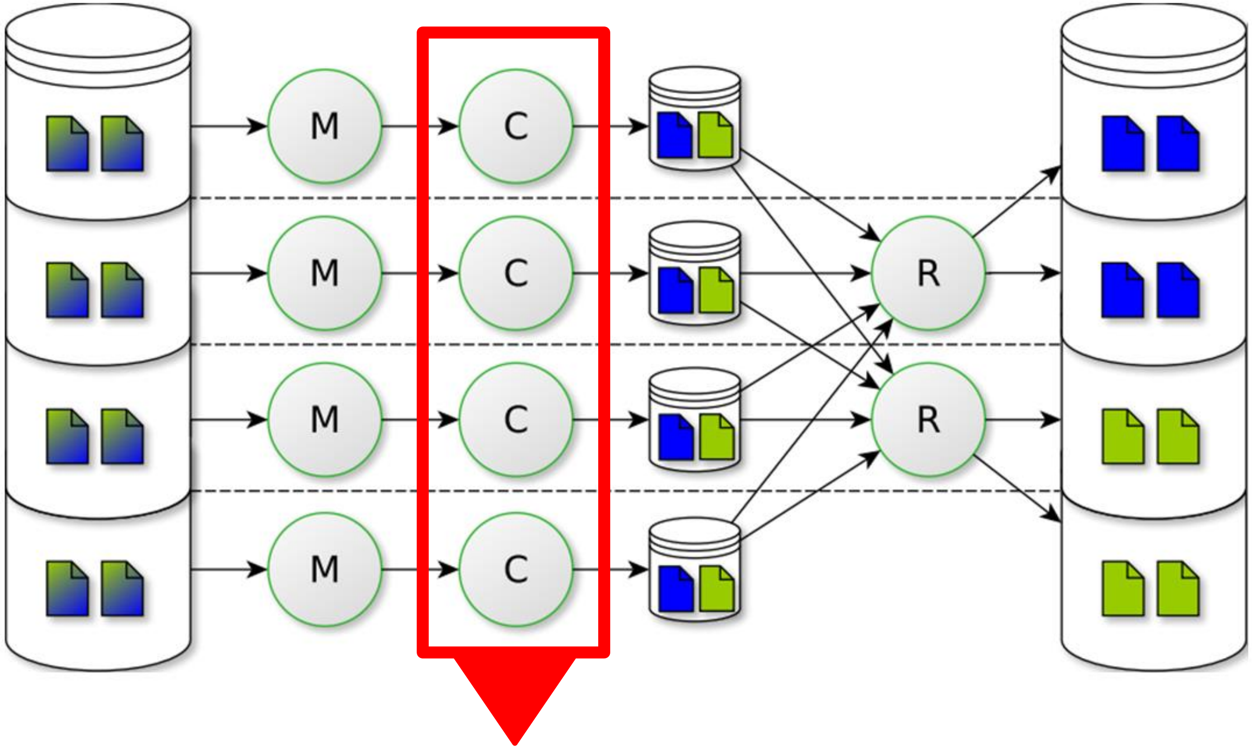
MapReduce Paper: Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

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

Jonas Traub, Gábor Gévay, Alexander Alexandrov

Lectures, Hands-On Tasks, and Reference Solutions:

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

BDAPRO
Principles and Execution Aspects of Dataflow Programs

Jonas Traub, Gábor Gévay, Alexander Alexandrov

Programs and Dataflows

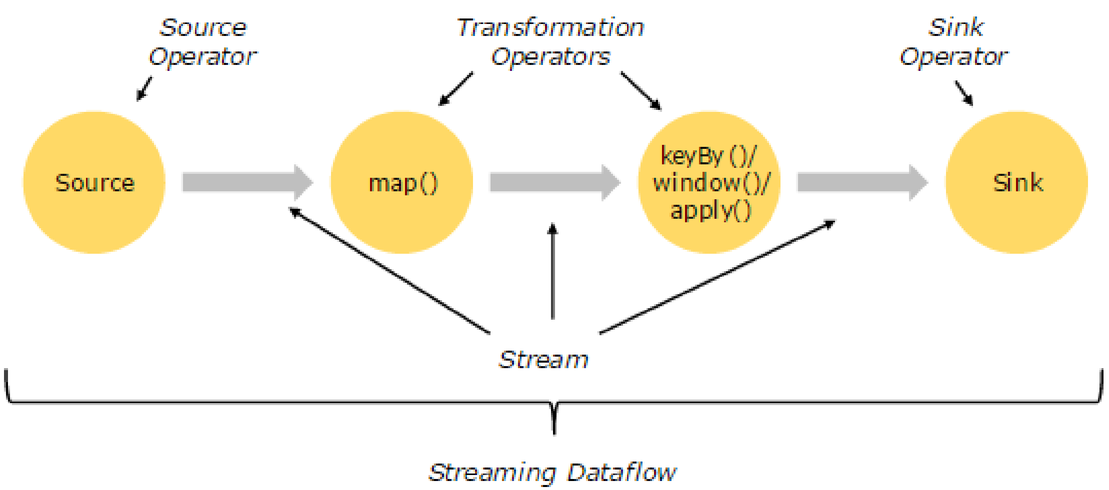
```
DataStream<String> lines = env.addSource(
                                new FlinkKafkaConsumer<>(…));

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

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

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

Source

Transformation

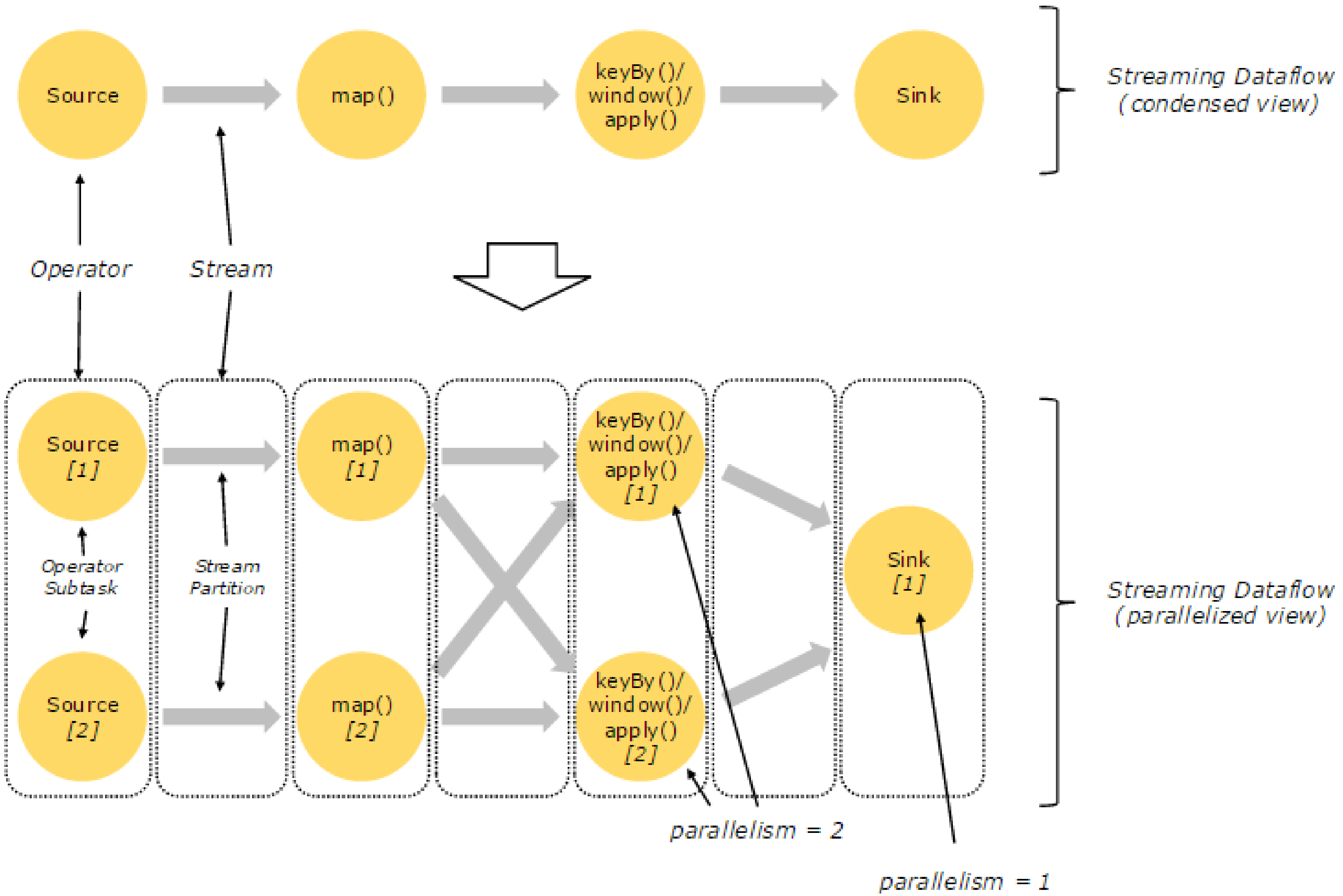
Transformation

Sink



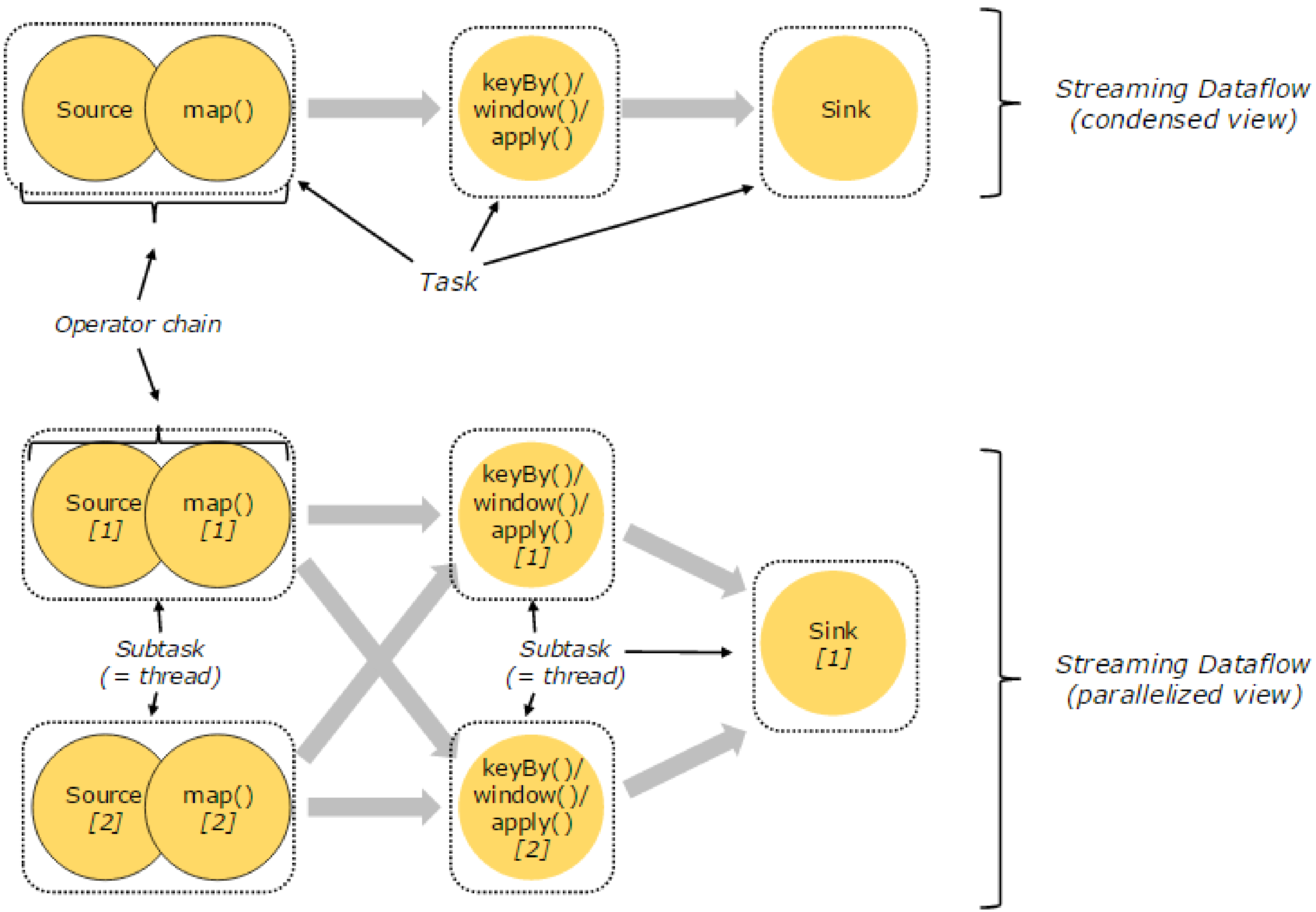
Source: https://ci.apache.org/projects/flink/flink-docs-master/concepts/concepts.html

Parallel Dataflows



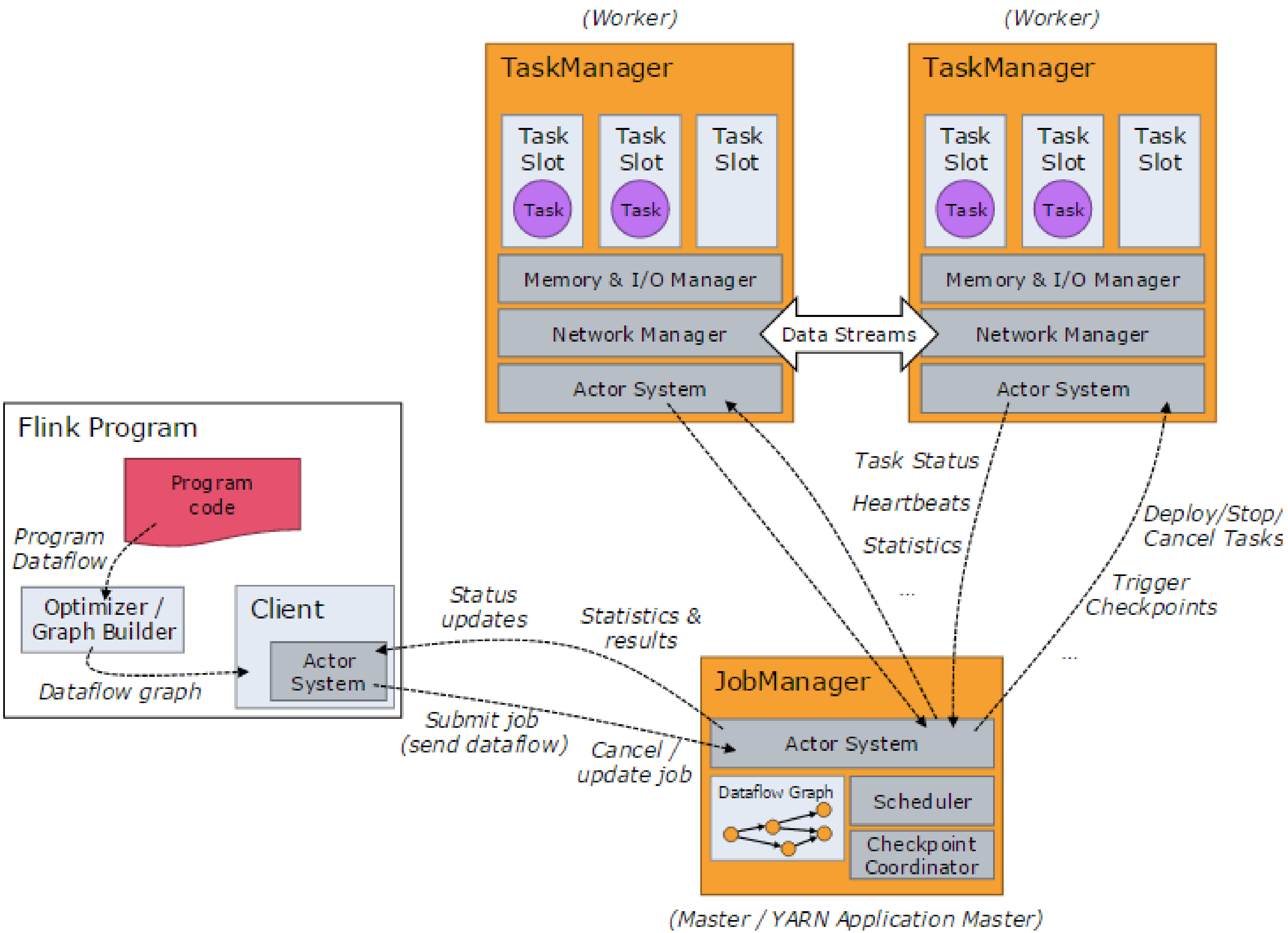
Source: https://ci.apache.org/projects/flink/flink-docs-master/concepts/concepts.html

Tasks & Operator Chains



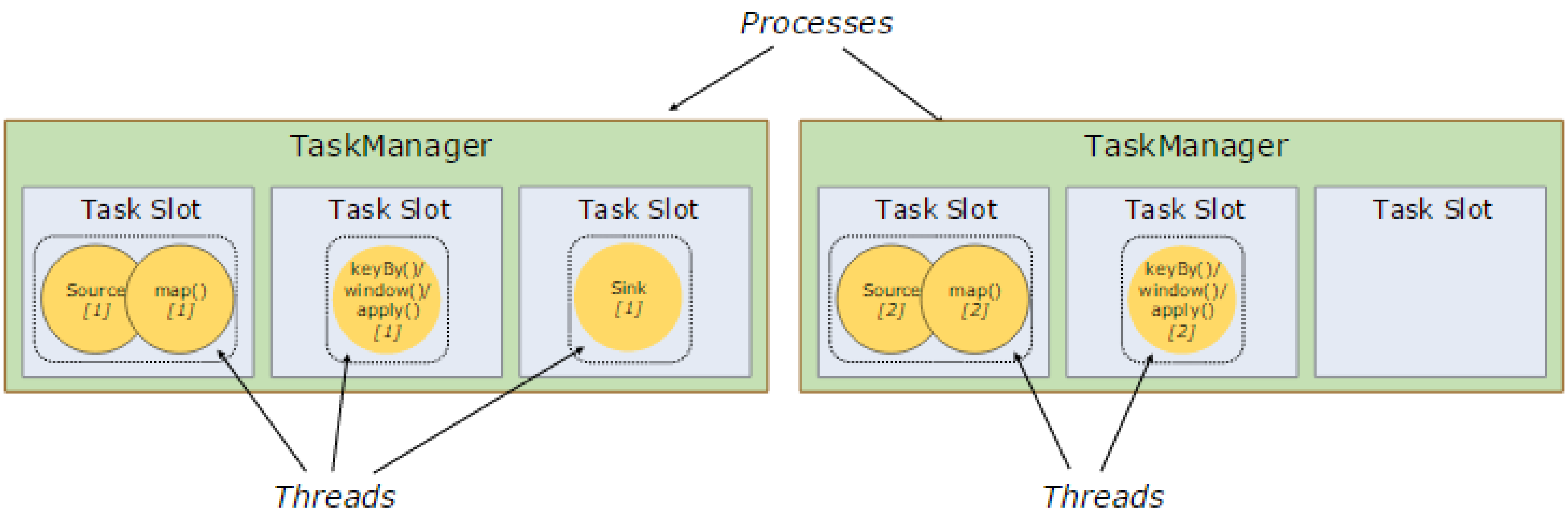
Source: https://ci.apache.org/projects/flink/flink-docs-master/concepts/concepts.html

Distributed Execution



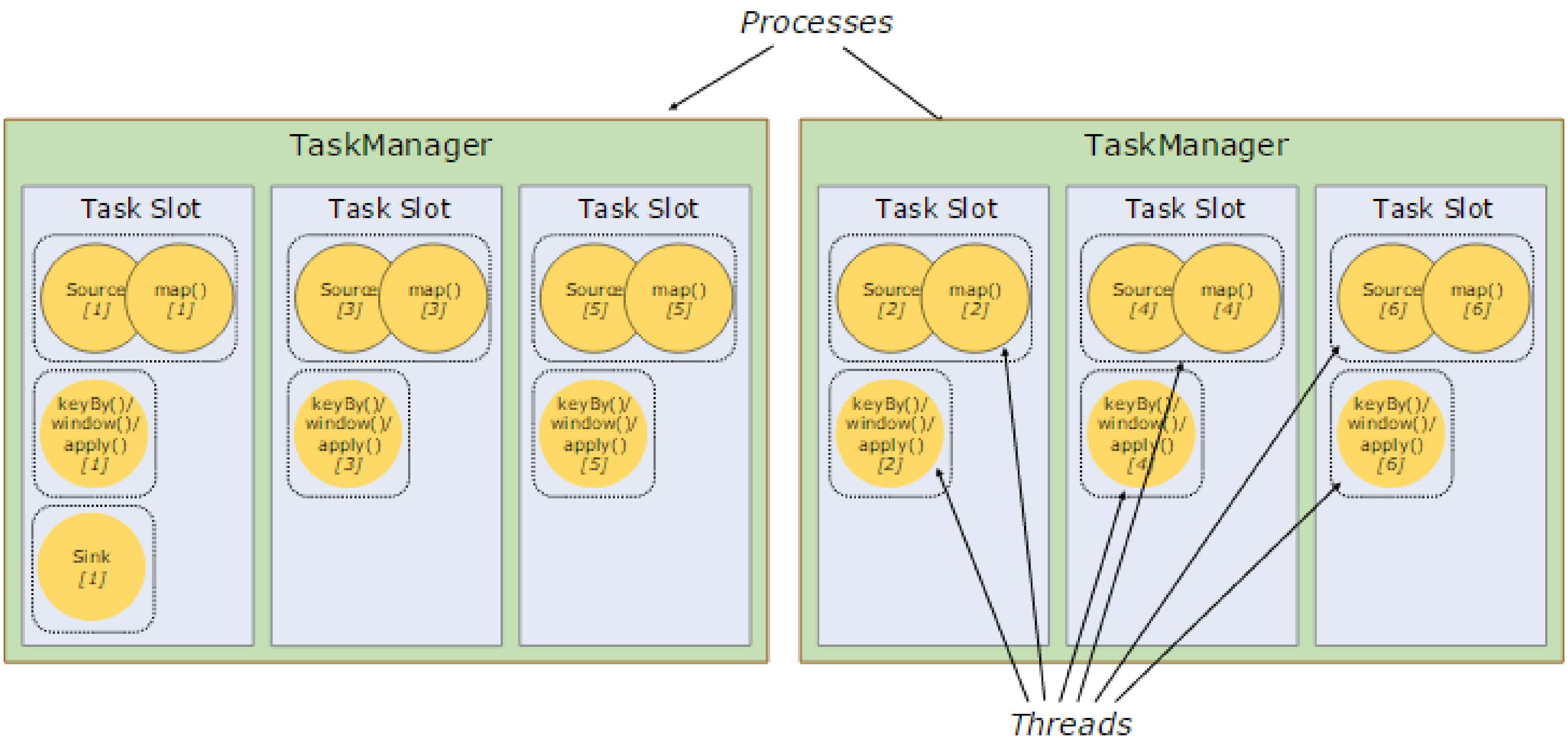
Source: https://ci.apache.org/projects/flink/flink-docs-master/concepts/concepts.html

Workers, Slots, and Resources



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

Workers, Slots, and Resources



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

BDAPRO

Comparison of Runtime Concepts

Jonas Traub, Gábor Gévay, Alexander Alexandrov

Pipelined vs. Batch Execution

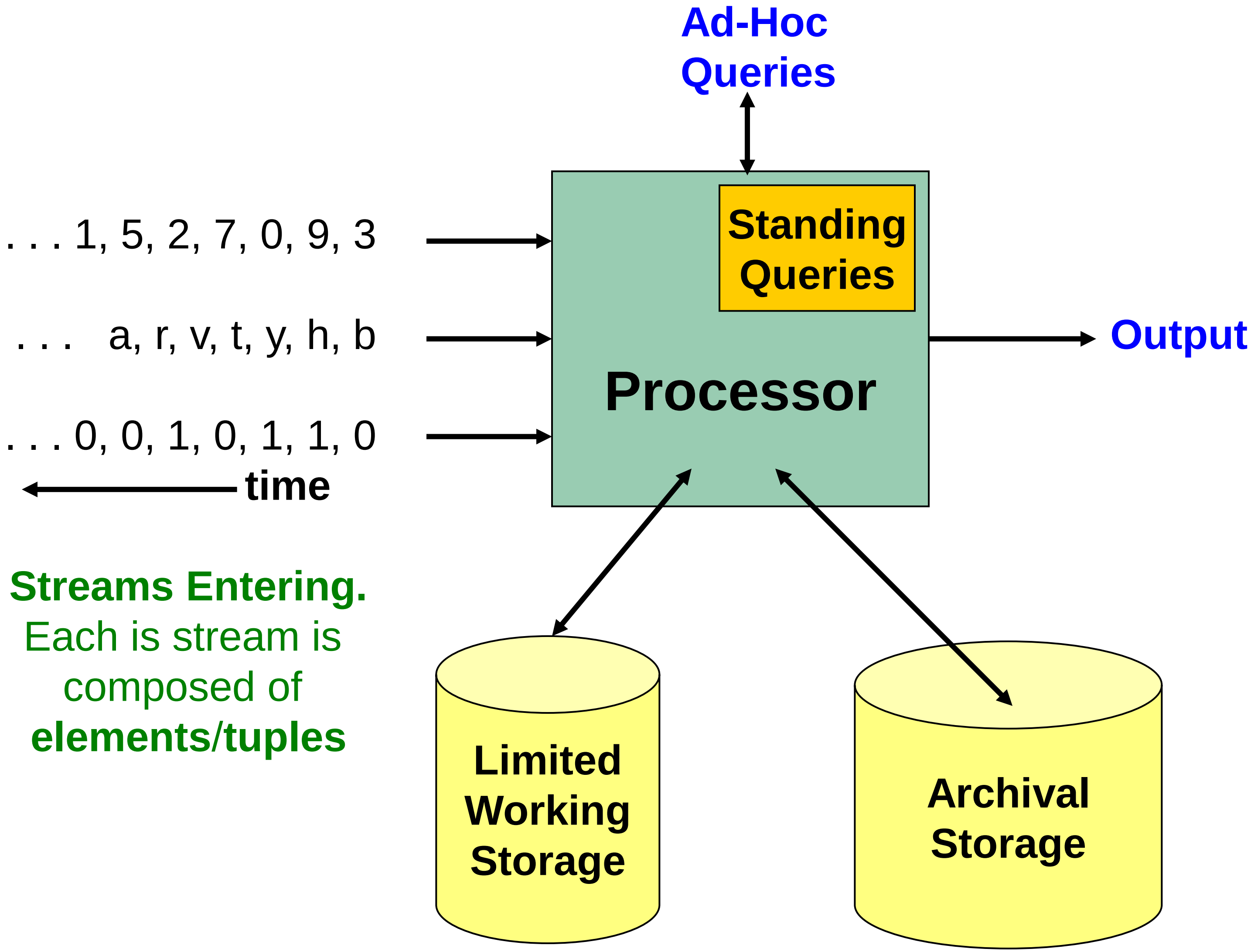
Batch Execution

- 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/

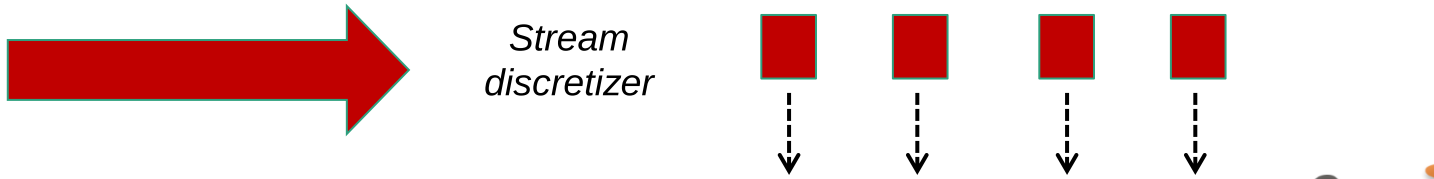
Stream Processing vs. Batch Processing

	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.




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

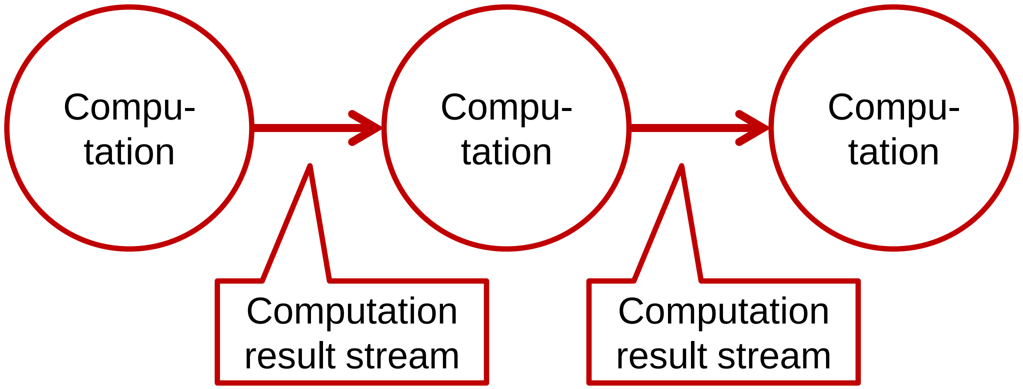
Native streaming


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

Slide by Gyula Fóra

Apache Storm: Basic Concepts

Topology:



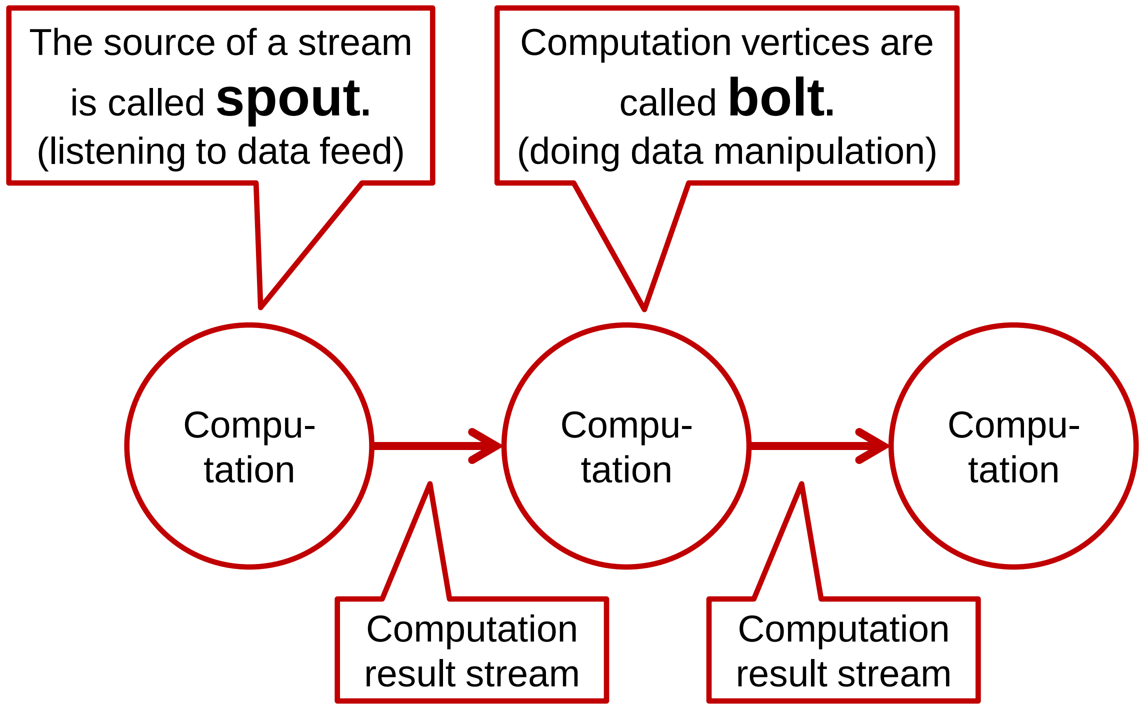
Programs are represented in a **topology**, which is a graph, whereas:

- **vertecies** are computations / data transformations
- **edges** represent data **streams** between the computation nodes
- such streams consist of an unbounded sequence of data-items/tuples

Source: Allen et al., Storm Applied: Strategies for Real-Time Event Processing

Apache Storm: Basic Concepts

Topology:



Programs are represented in a **topology**, which is a graph, whereas:

- **vertecies** are computations / data transformations
- **edges** represent data **streams** between the computation nodes
- such streams consist of an unbounded sequence of data-items/tuples

Source: Allen et al., Storm Applied: Strategies for Real-Time Event Processing

Apache Storm: Implementing Bolts

```
public class DoubleAndTripleBolt extends BaseRichBolt {
    private OutputCollectorBase _collector;

    @Override
    public void prepare(Map conf, TopologyContext context, OutputCollectorBase collector) {
        _collector = collector;
    }

    @Override
    public void execute(Tuple input) {
        int val = input.getInteger(0);
        _collector.emit(input, new Values(val*2, val*3));
        _collector.ack(input);
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("double", "triple"));
    }
}
```

Source: https://storm.apache.org/documentation/Tutorial.html

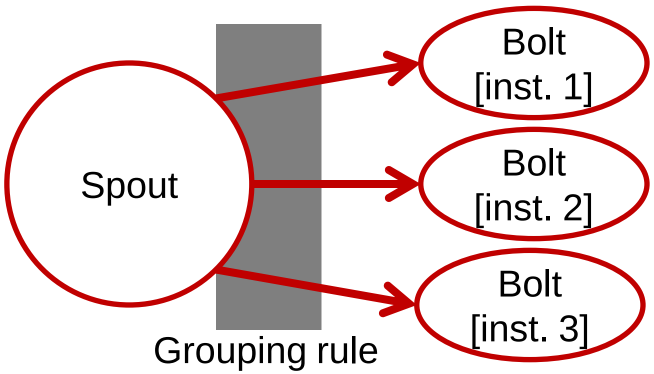
Apache Storm: Building the Topology

1) Use the TopologyBuilder class to connect spouts and bolts:

```
builder.setSpout("name",new MySpout());
builder.setBolt("name",new MyBolt());
```

2) Additionally, specify groupings to allow parallelization

```
builder.shuffleGrouping("BoltName");
```



3) Create topology using the factory method

```
StormTopology st=builder.createTopology();
```

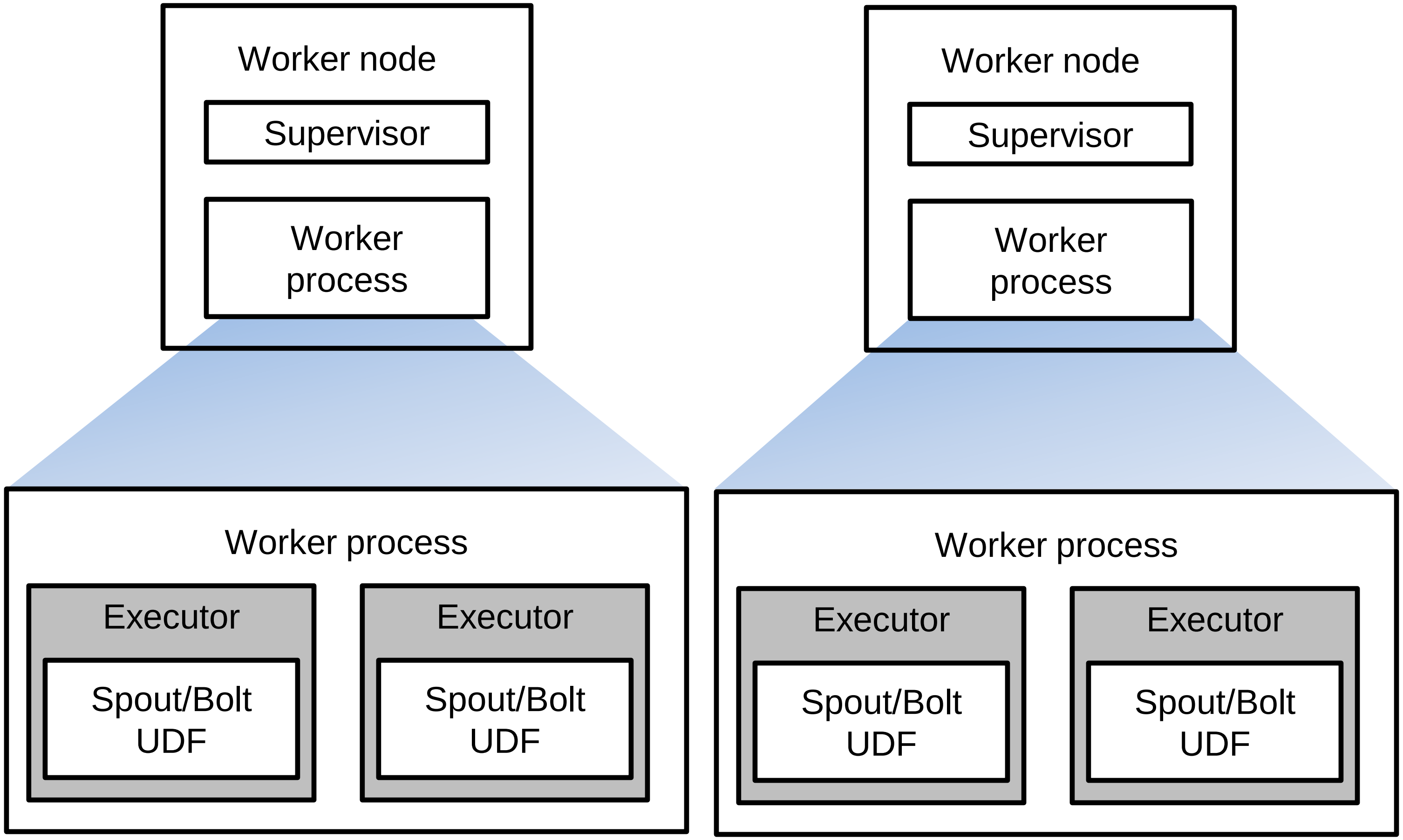
4) Use LocalCluster class to test the topology

```
LocalCluster cluster=new LocalCluster();
cluster.submitTopology("name",new Config(),st);
```

Source: Allen et al., Storm Applied: Strategies for Real-Time Event Processing

Apache Storm: Internals

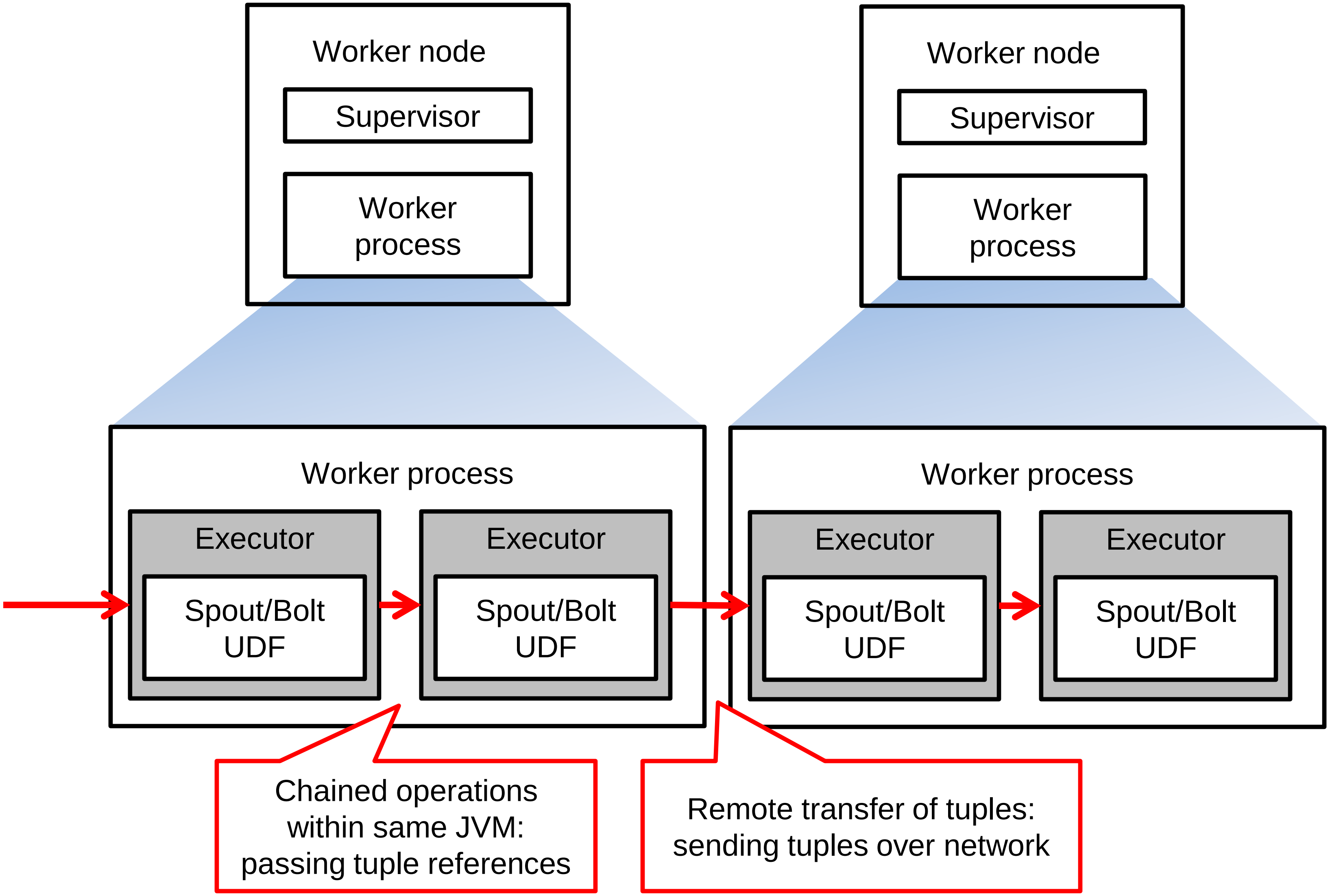
Cluster Architecture Overview



Source: Allen et al., Storm Applied: Strategies for Real-Time Event Processing

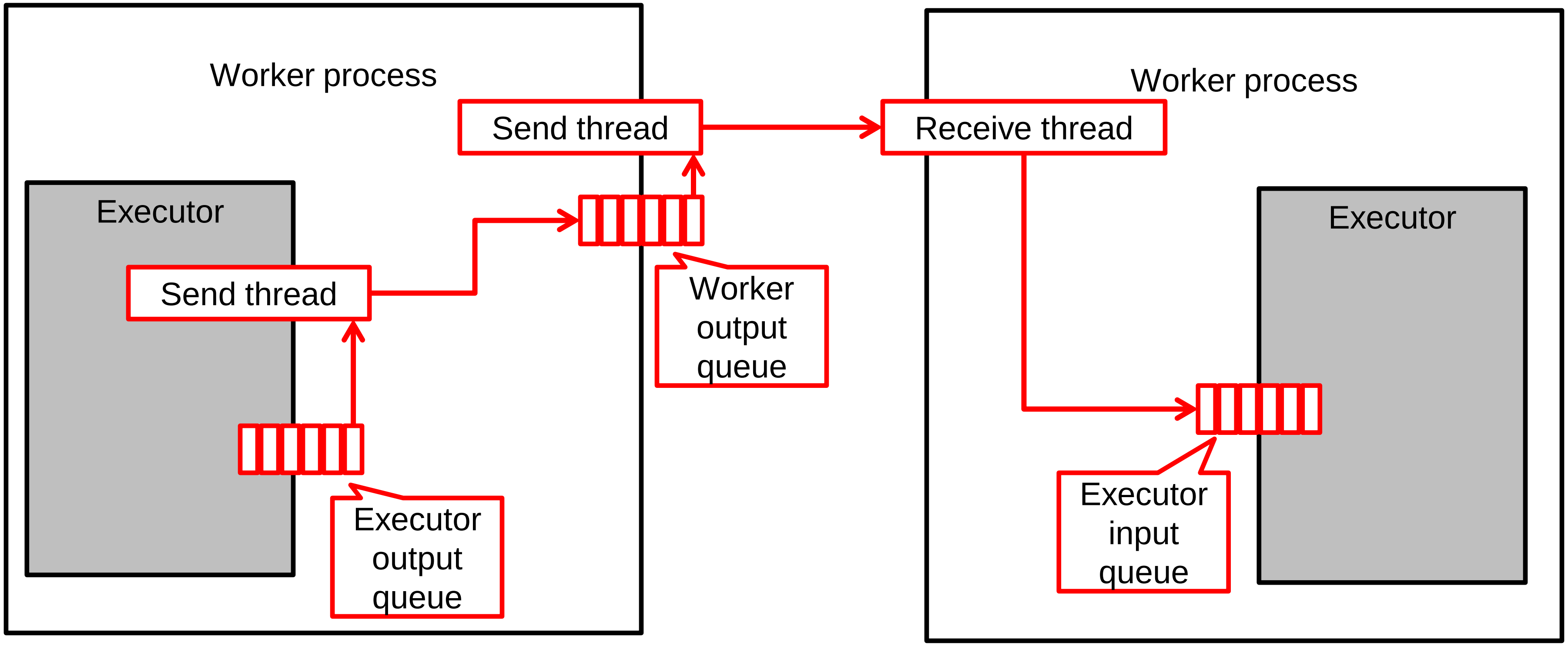
Apache Storm: Internals

Message passing between Executors



Apache Storm: Internals

Sending tuples between executors on different JVMs

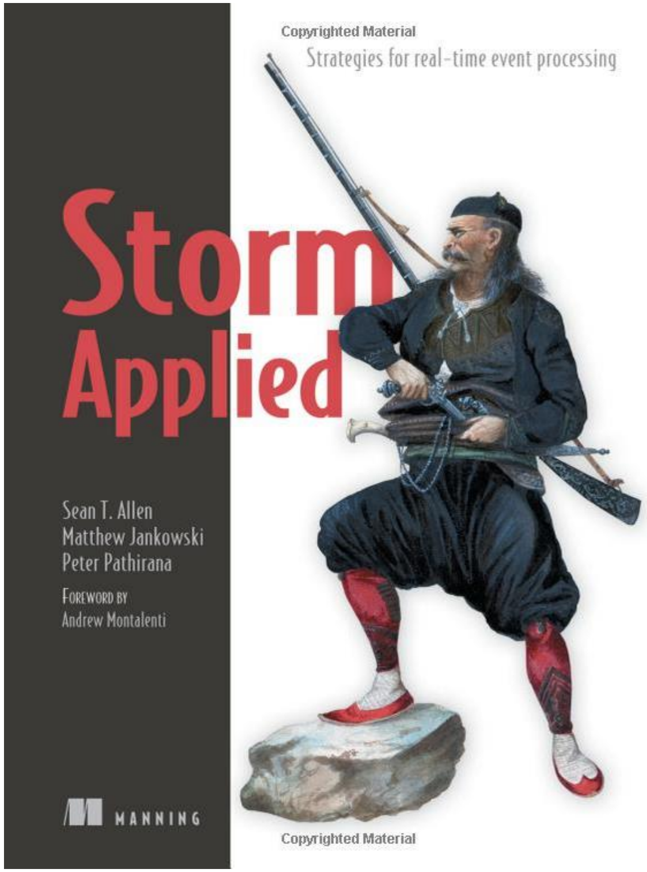


Remarks:

- It is important to configure the buffer sizes appropriate.
  Buffers can overflow which might cause massive performance decrease.
- The receive thread makes sure that tuples are forwarded to the correct executor instance.

Source: Allen et al., Storm Applied: Strategies for Real-Time Event Processing

Apache Storm: Recommended Reading



Storm Applied: Strategies for Real-Time Event Processing

Englisch; 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

Jonas Traub, Gábor Gévay, Alexander Alexandrov