AIM3
Stream Processing

Jonas Traub

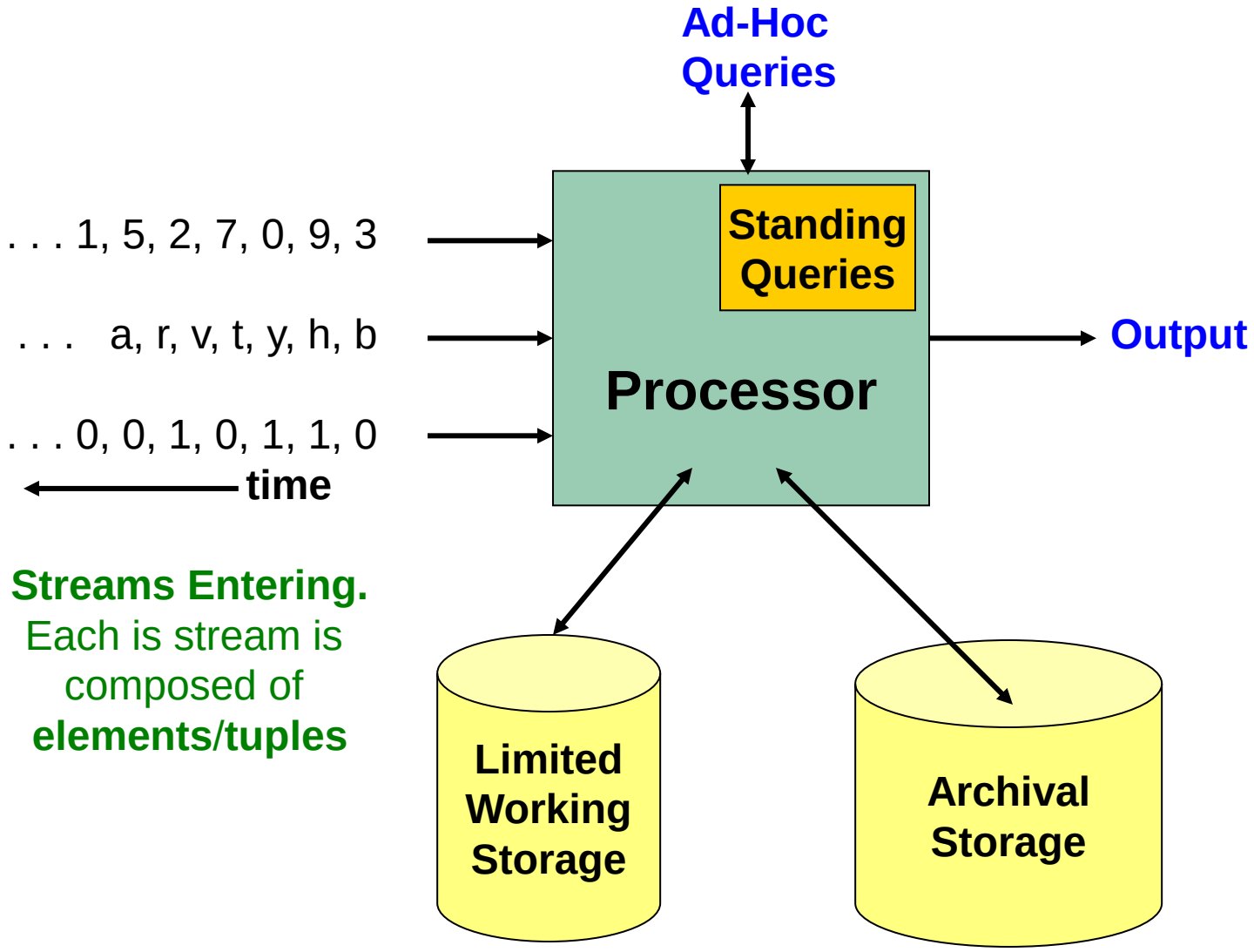
AIM3
Stream Processing
Jonas Traub

1. **Theory:**
   Introducing Stream Processing
2. **Practice:**
   Stream Processing Systems
3. **Theory applied in Practice:**
   Stream Processing Optimizations

AIM3
Introducing Stream Processing

Jonas Traub

The General Stream Processing Model

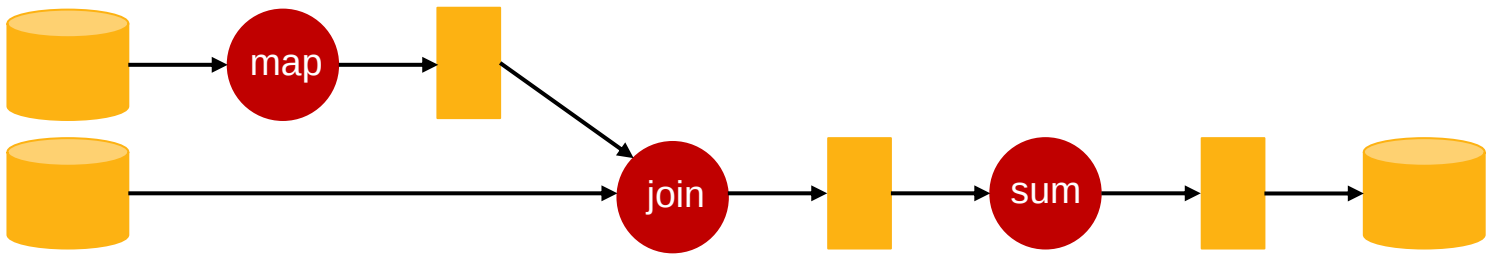


Streams Entering. Each is stream is composed of **elements/tuples**

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

Standing Queries / Execution Model

- Program = Graph of operators and intermediate results
- Operator = computation + state
- Intermediate result = logical stream of records



Essential Definitions

STREAM	A conceptual/possibly infinite sequence of data-items which comes from a stream source.
STREAM SOURCE	A stream source is a data source that possibly emits an infinite number of data-items.
DATA-ITEM	The smallest unit of data, which is processed by the streaming query, is called data-item.
WINDOW	A finite subsequence or chunk of data-items from a stream.

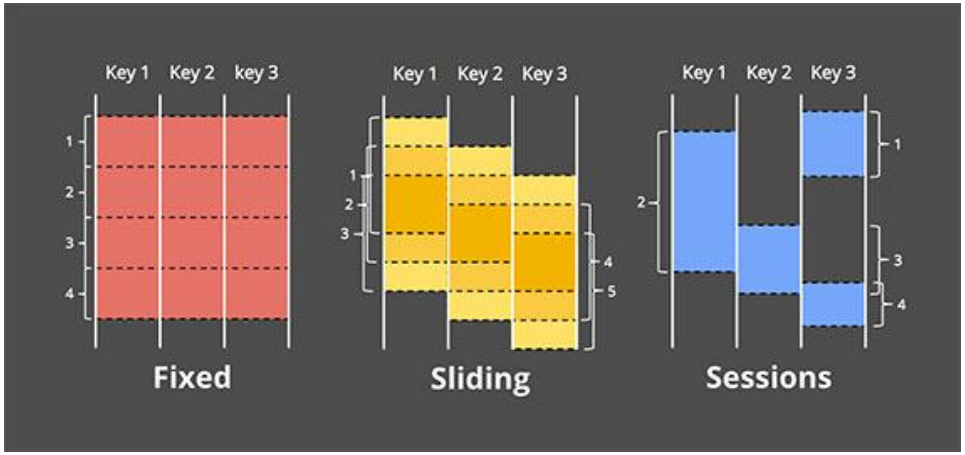
Definitions from:
Hirzel, Martin, et al. A catalog of stream processing optimizations. *ACM Computing Surveys (CSUR)*, 2014, 46. Jg., Nr. 4, S. 46.; http://dl.acm.org/citation.cfm?id=2528412

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

Stream Discretization

- Data is unbounded
  - We are interested in a (recent) part of it e.g. last 10 days
- Most common windows around: time and tuple count
  - Mostly in sliding, fixed, and tumbling form
- Need for data-driven window definitions
  - e.g., user sessions (periods of user activity followed by inactivity), price changes, etc.
  - Also check: Gedik, Buğra. **"Generic windowing support for extensible stream processing systems."** *Software: Practice and Experience* 44.9 (2014): 1105-1128.



The world beyond batch: Streaming 101, Tyler Akidau
https://beta.oreilly.com/ideas/the-world-beyond-batch-streaming-101
Great read!

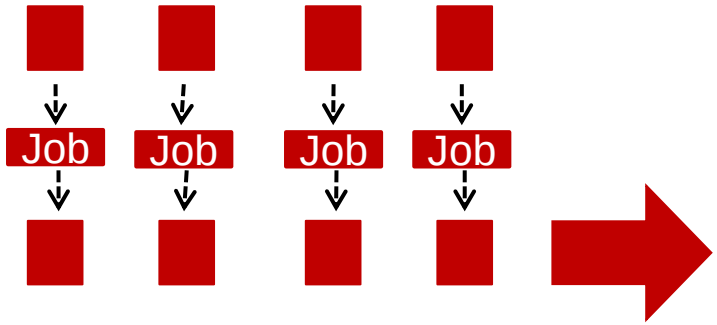
Micro Batching vs. Native Streaming

Discretized Streams (D-Streams)

Spark

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

Stream discretizer



Native streaming

Flink

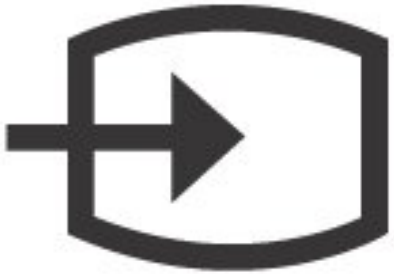
```
while (true) {
   // process next record
}
```

Different Notions of Time

Event Time

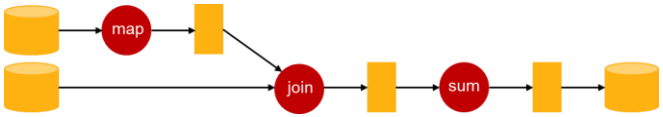
The time an event actually occurred / when it was captured

Ingestion Time



The time a tuple arrives at the system (Processing time at the data source operator)

Processing Time



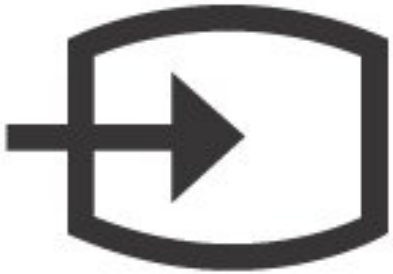
The time a tuple is processed by an operator (system time)

Different Notions of Time

Event Time

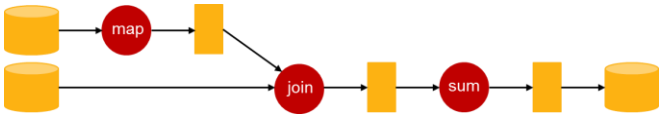
The time an event actually occurred / when it was captured

Ingestion Time



The time a tuple arrives at the system (Processing time at the data source operator)

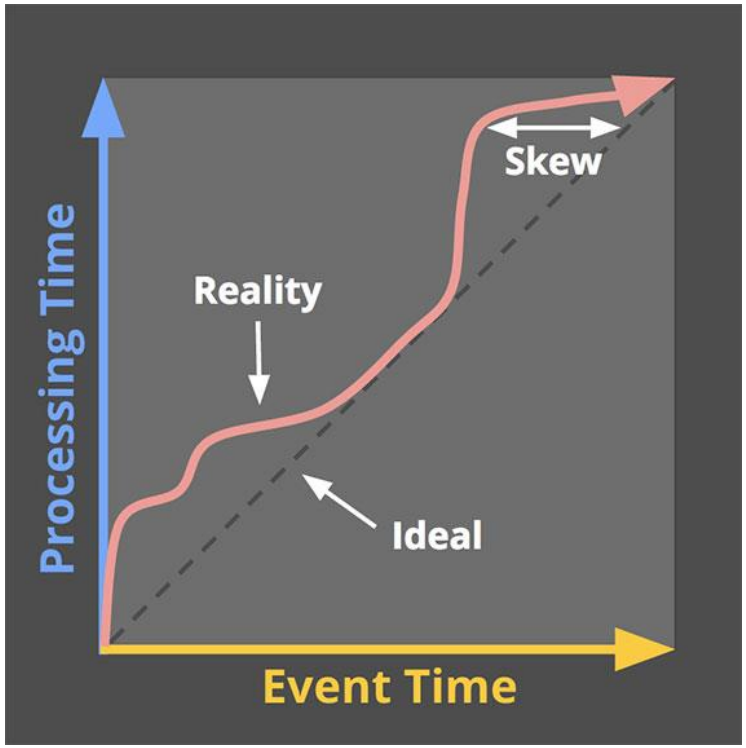
Processing Time



The time a tuple is processed by an operator (system time)

Times may diverge for the same tuple
➔ Leads to out-of-order stream processing

Processing Time, Event Time, and Watermarks



We discovered this topic on the board in class.

For repetition please consider:

Akidau, Tyler, et al.
"The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing."
Proceedings of the VLDB Endowment 8.12 (2015): 1792-1803.

Recent Reasearch on Window Aggregation

Cutty: Aggregate Sharing for User-Defined Windows

Paris Carbone† Jonas Traub‡ Asterios Katsifodimos‡
Seif Haridi† Volker Markl‡

†KTH Royal Institute of Technology
{parisc,haridi}@kth.se

‡Technische Universität Berlin & DFKI
firstname.lastname@tu-berlin.de

CIKM 2016

Link:
http://dl.acm.org/citation.cfm?id=2983807

Presentation:
http://www.slideshare.net/ParisCarbone/aggregate-sharing-for-userdefine-data-stream-windows

AIM3
Stream Processing Systems Overview

Jonas Traub

Agenda

Use Cases
- Spark, Storm, and Flink applications

Streaming Systems Overview
- Closed Source (Commercial)
- Open Source (Apache/Academia)

Apache Storm
- Basic Concepts
- Building a Topology
- Storm Internals

Apache Flink Streaming
- System Overview
- Windowing Semantics
- Code Examples

Agenda

Use Cases
- Spark, Storm, and Flink applications

Streaming Systems Overview
- Closed Source (Commercial)
- Open Source (Apache/Academia)

Apache Storm
- Basic Concepts
- Building a Topology
- Storm Internals

Apache Flink Streaming
- System Overview
- Windowing Semantics
- Code Examples

Use Cases

*"Storm powers a wide variety of Twitter systems, ranging in applications from **discovery, realtime analytics, personalization, search, revenue optimization**, and many more."*

http://storm.apache.org/documentation/Powered-By.html

*"Conviva monitors and **optimizes tens of millions of online video streams** daily for premium video brands. […] For example, customers can in **real-time identify the most popular videos** being watched and adjust their advertising strategy."*

http://www.conviva.com/using-spark-and-hive-to-process-bigdata-at-conviva/

*"**Learn network conditions in real-time** and feed it directly into the video player, to optimize the streams."*

http://siliconangle.com/blog/2014/06/30/4-use-cases-for-spark-transforming-large-scale-data-analysis-bigdatasv/

Flink provides *"[…] an **intermediate processing pipeline** that processes the raw data in Kafka and feeds the data back to Kafka in decoded and enriched format. […] deliver results with **low latency** and **keep the data moving in real time**."*

http://data-artisans.com/flink-at-bouygues.html

Use Cases

*"Storm powers a wide variety of Twitter systems, ranging in applications from **discovery, realtime analytics, personalization, search, revenue optimization**, and many more."*

http://storm.apache.org/documentation/Powered-By.html

*"Conviva monitors and **optimizes tens of millions of online video streams** daily for premium video brands. […] For example, customers can in **real-time identify the most popular videos** being watched and adjust their advertising strategy."*

http://www.conviva.com/using-spark-and-hive-to-process-bigdata-at-conviva/

*"**Learn network conditions in real-time** and feed it directly into the video player, to optimize the streams."*

http://siliconangle.com/blog/2014/06/30/4-use-cases-for-spark-transforming-large-scale-data-analysis-bigdatasv/

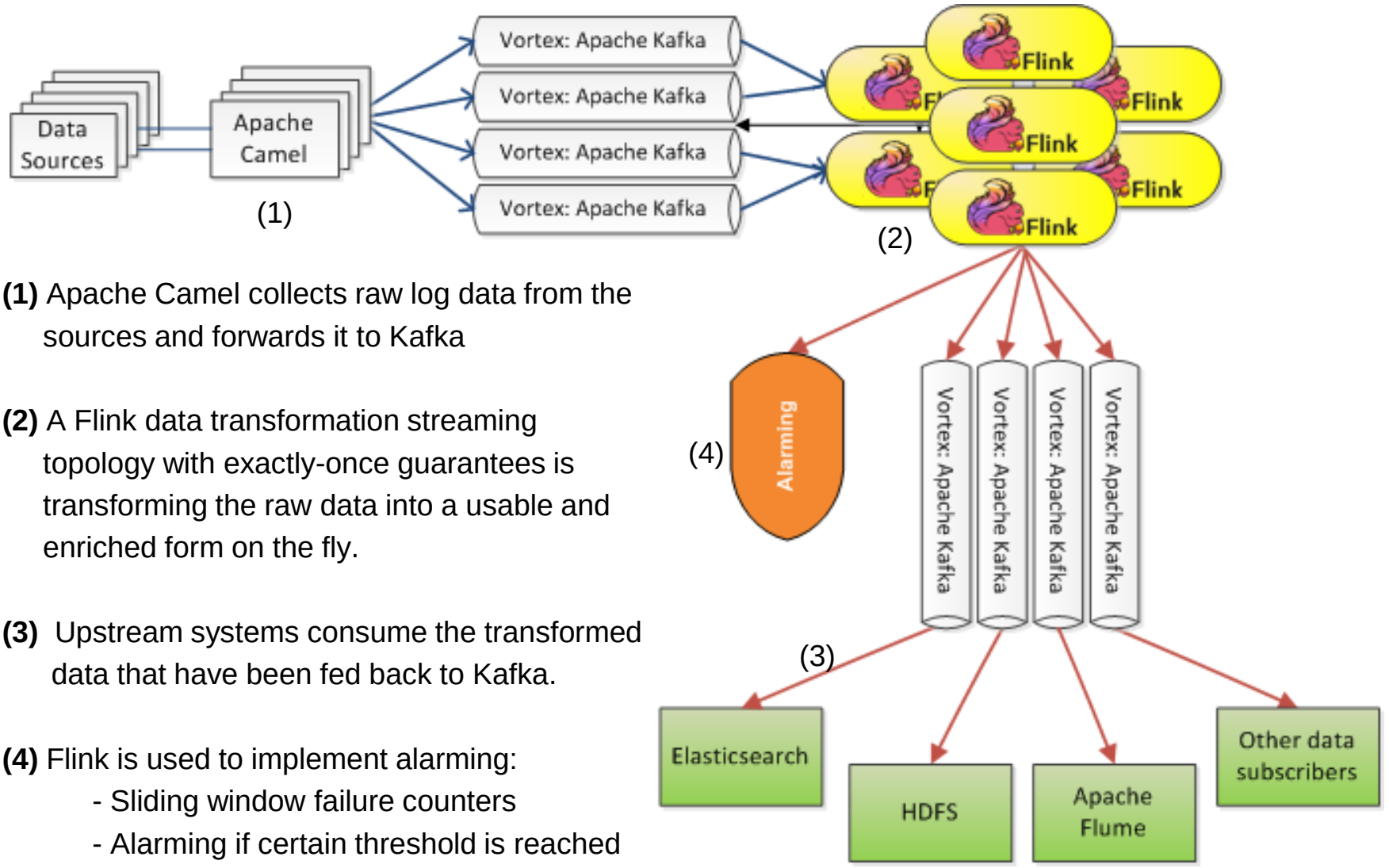
Flink provides *"[…] an **intermediate processing pipeline** that processes the raw data in Kafka and feeds the data back to Kafka in decoded and enriched format. […] deliver results with **low latency** and **keep the data moving in real time**."*

http://data-artisans.com/flink-at-bouygues.html

Main characteristics:

➔ Real time requirement ➔continuous data arrival ➔long/infinitly running queries

The Bouygues Use-Case



(1) Apache Camel collects raw log data from the sources and forwards it to Kafka

(2) A Flink data transformation streaming topology with exactly-once guarantees is transforming the raw data into a usable and enriched form on the fly.

(3) Upstream systems consume the transformed data that have been fed back to Kafka.

(4) Flink is used to implement alarming:
- Sliding window failure counters
- Alarming if certain threshold is reached

Source: http://data-artisans.com/flink-at-bouygues.html

Agenda

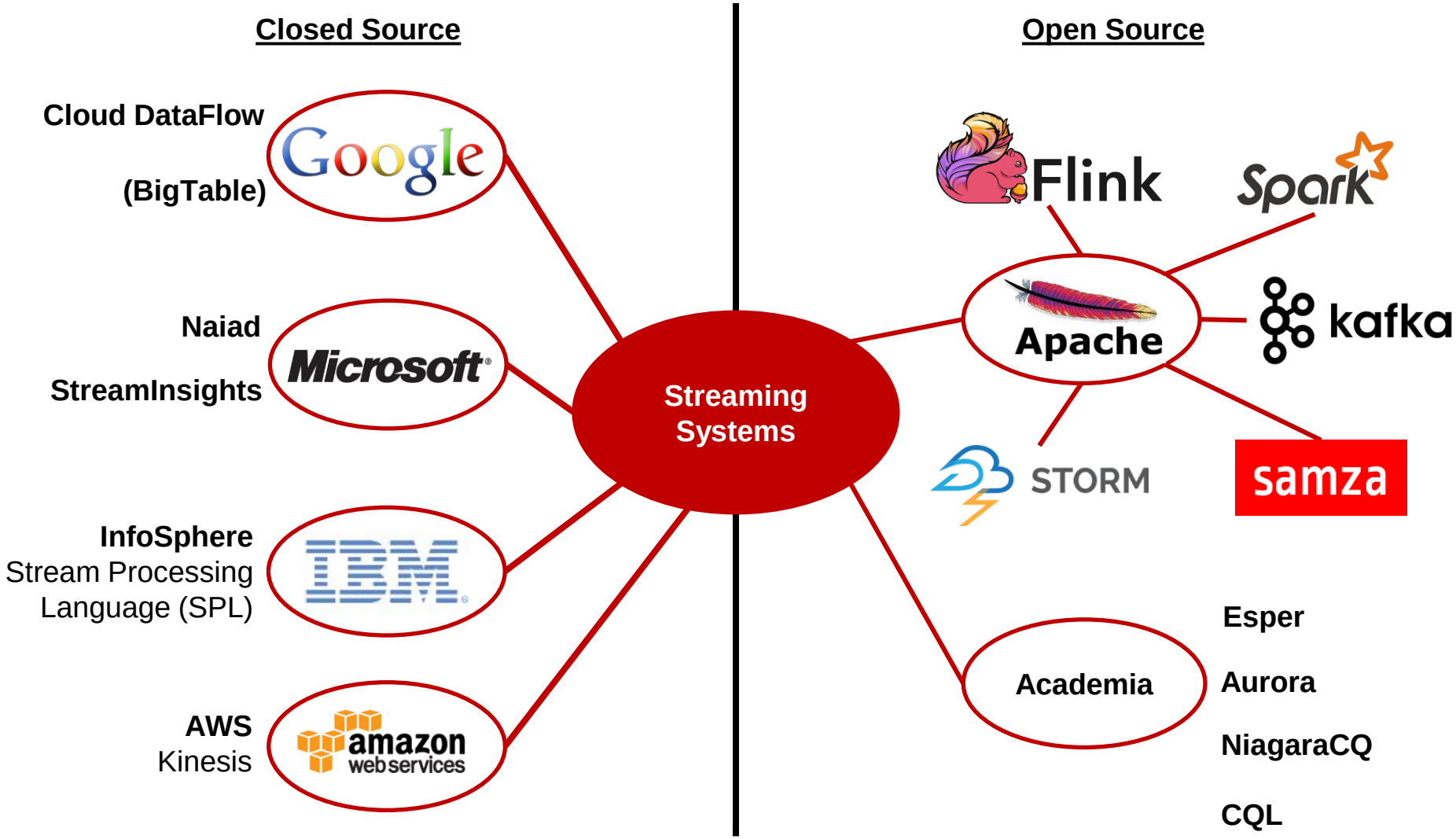
Use Cases
- Spark, Storm, and Flink applications

Streaming Systems Overview
- Closed Source (Commercial)
- Open Source (Apache/Academia)

Apache Storm
- Basic Concepts
- Building a Topology
- Storm Internals

Apache Flink Streaming
- System Overview
- Windowing Semantics
- Code Examples

Streaming Systems Overview



Closed Source/Commercial Systems

Cloud DataFlow:
- Unified primitives for batch and stream processing
- Runs in Google's cloud only
- Open Source SDK (programs can run on other systems)
- Check out the Apache Beam Project! (http://beam.apache.org/)

BigTable:
- Not a real streaming solution
- Allows to feed streams as source into a google DB
- Data can be immediately queried

Naiad:
- Goals of Naiad [1]:
  - High throughput (typical for batch processors)
  - Low latency (known from single system stream processors)
- Is able to process iterative data flows
- Can discretize windows only based on time

StreamInsights:
- Available through Microsoft's cloud
- Windows based on count-, time- and punctuation/snapshot
- Optimized for .NET framework applications

InfoSphere: Stream Processing Language (SPL)
- Well specified in several publications [2, 3, 4]
- Can be deployed in customer clusters
- Own SQL-like query language enables many optimization means
- window discretization based on trigger- and eviction policies

Open Source Systems by Apache (1/2)

- Reliable handling of huge numbers of concurrent reads and writes
- Can be used as data-source / data-sink for Storm, Samza, Flink, Spark and many more systems
- Fault tolerant: Messages are persisted on disk and replicated within the cluster. Messages (reads and writes) can be repeated

- True streaming over distributed dataflow
- Low level API: Programmers have to specify the logic of each vertex in the flow graph
- Full understanding and hard coding of all used operators is required
- Enables very high throughput (single purpose programs with small overhead)

- True streaming built on top of Apache Kafka and Hadoop YARN
- state is first class citizen
- Low level API

Open Source Systems by Apache (2/2)

Spark implements a batch execution engine

- The execution of a job graph is done in stages
- Operator outputs are materialized in memory (or disk) until the consuming operator is ready to consume the materialized data

Spark uses Discretized Streams (D-Streams) [5]

- Streams are interpreted as a series of deterministic batch-processing jobs
- Micro batches have a fixed granularity
- All windows defined in queries must be multiples of this granularity

former known as Stratosphere [6]

Flinks runtime is a native streaming engine

- Based on Nephele/PACTs [6]
- Queries are compiled to a program in the form of an operator DAG
- Operator DAGs are compiled to job graphs
- Job graphs are generic streaming programs

What means “true streaming”?

- The whole job graph is deployed concurrently in the cluster
- Operators are long-running: Continuously consume input and produce output
- Output tuples are immediately forwarded to succeeding operators and are available for further processing (enables pipeline parallelism)

Further open source systems

Esper [7]

- Open source Complex Event Processing (CEP) engine
- Tightly coupled to Java: **Executable on J2EE application servers**
- Describing events in Plain Old Java Objects (POJOs)
- Time-based or count-based windows

Aurora [8]

- First design and implementation that parallelizes stream computation including rich operation and windowing semantics
- Windows are always specified as ranges on some measure
- Was continued in Borealis Project

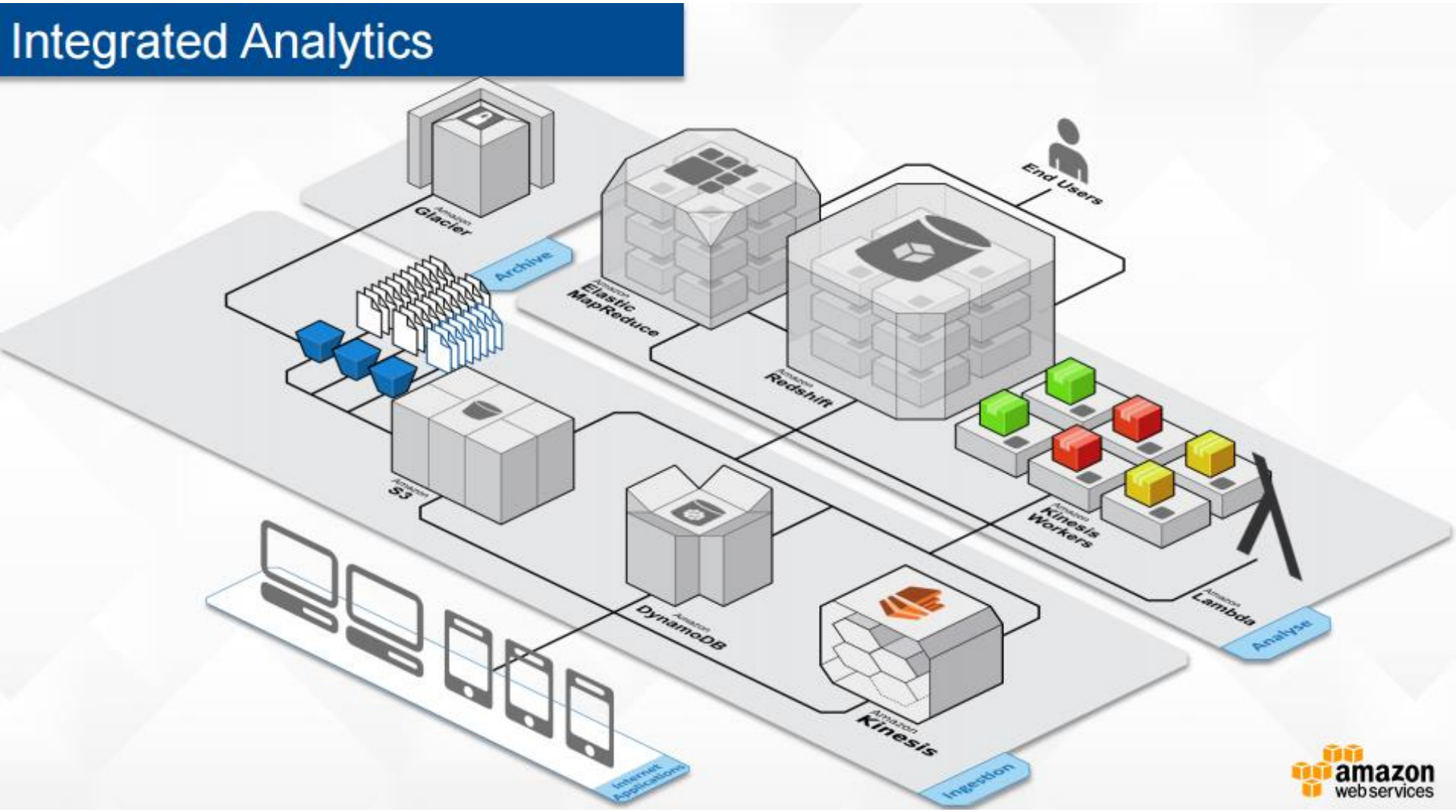
NiagaraCQ [9]

- Focuses more on scalability than on the flexibility
- Provides various optimizations techniques to share common computation within and across queries
- Only time-based windows are possible

CQL [10]

- Continuous query language
- Implemented by the STREAM DSMS at Stanford
- Captures a wide range of streaming application in an SQL-like query language

Cloud Based Streaming System (example)



Source: http://aws.amazon.com/de/campaigns/summit2015/

Agenda

Use Cases
- Spark, Storm, and Flink applications

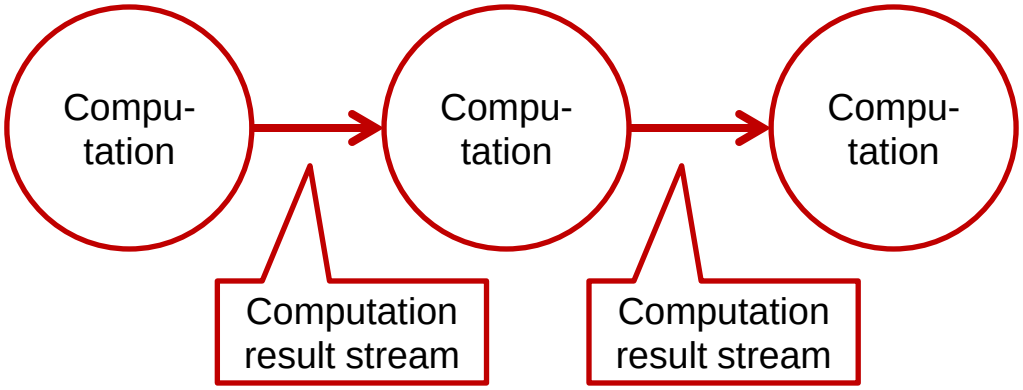
Streaming Systems Overview
- Closed Source (Commercial)
- Open Source (Apache/Academia)

Apache Storm
- Basic Concepts
- Building a Topology
- Storm Internals

Apache Flink Streaming
- System Overview
- Windowing Semantics
- Code Examples

Apache Storm: Basic Concepts

Topology:



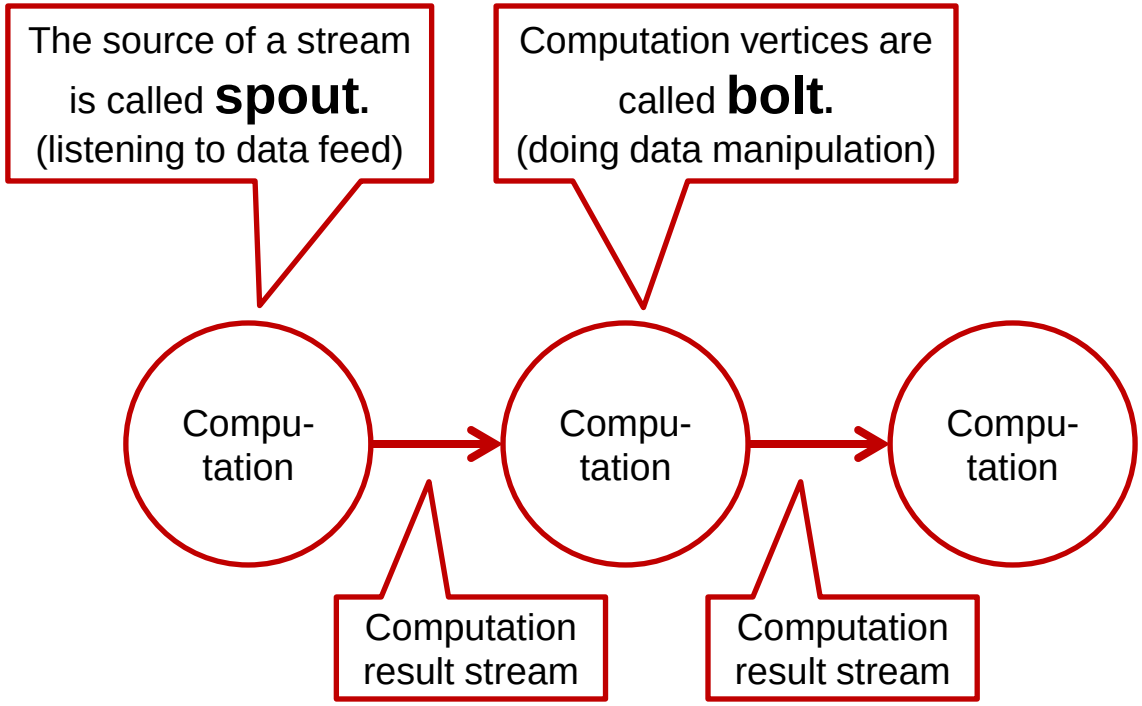
Programs are represented in a **topology**, which is a graph, whereas:

- **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:

- **vertices** 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

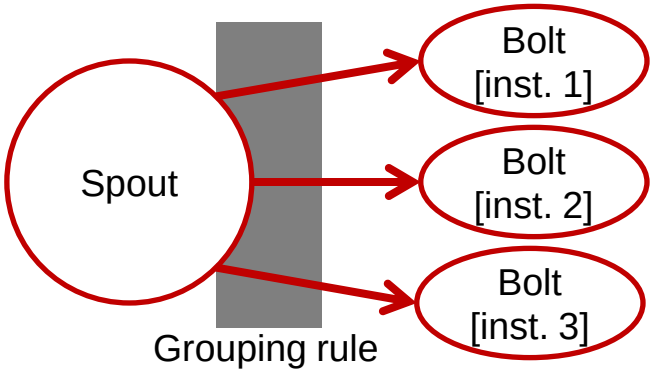
Apache Storm: Building the Topology

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();
```

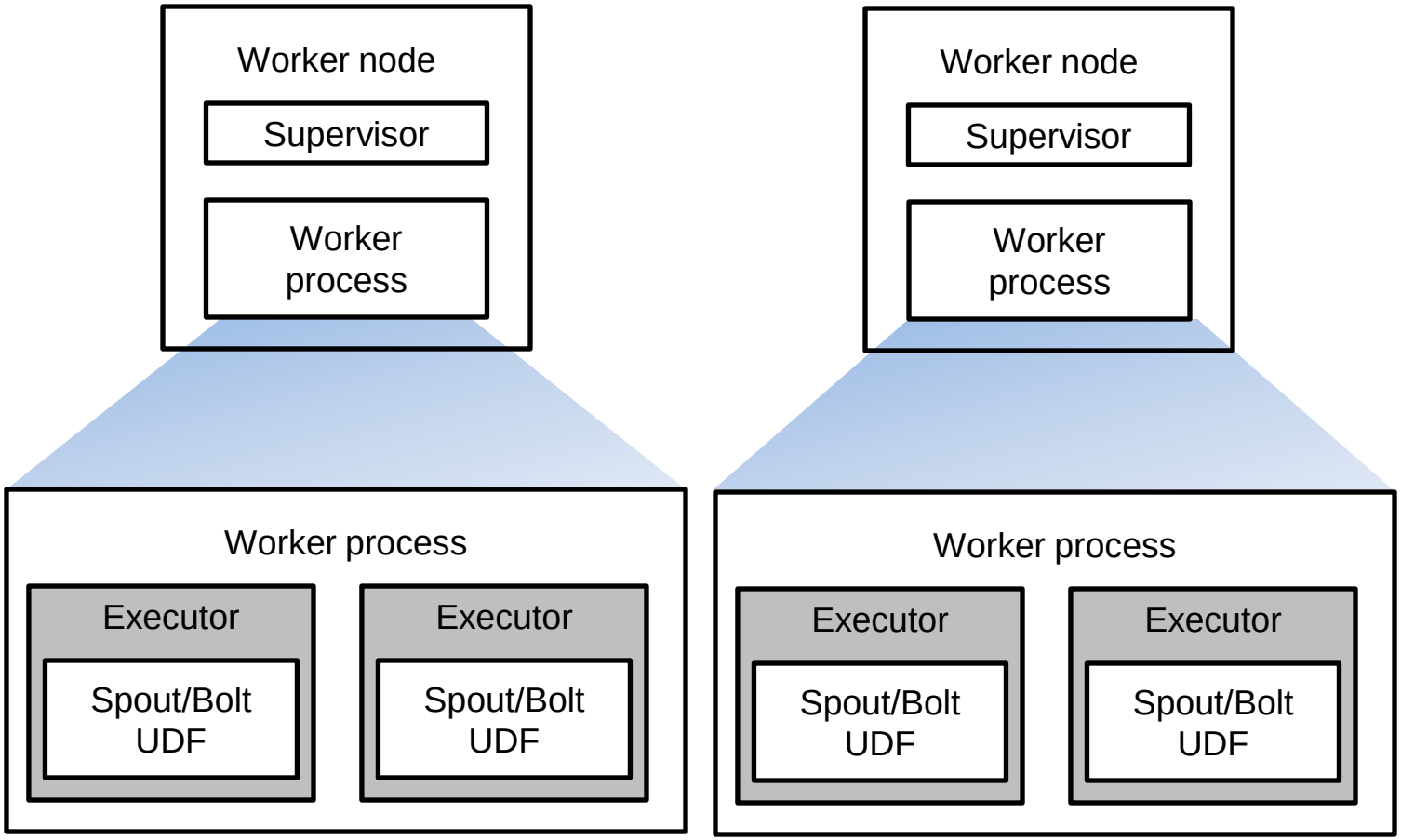
4) Use LocalCluster class to test the topology

```
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

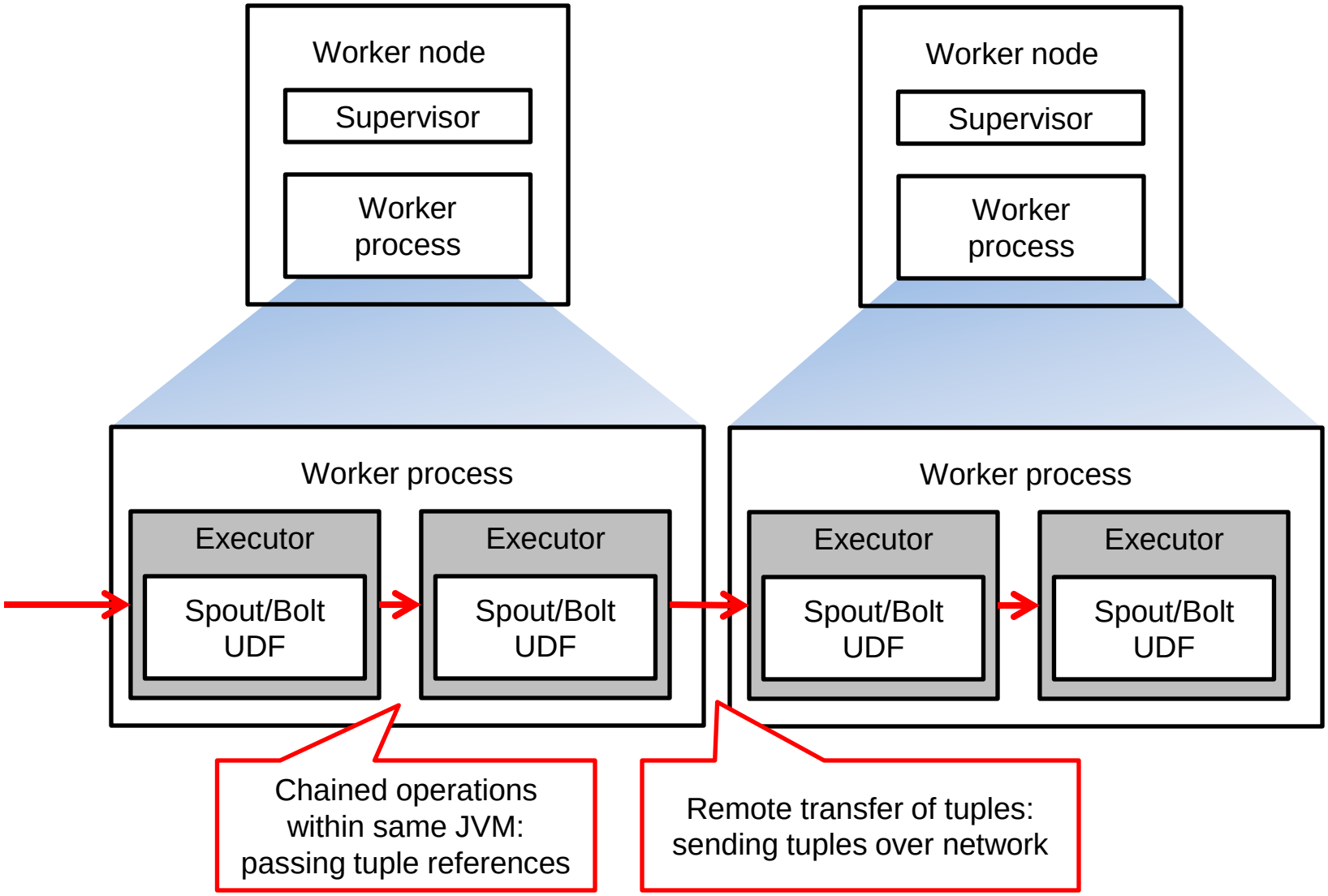
Cluster Architecture Overview



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

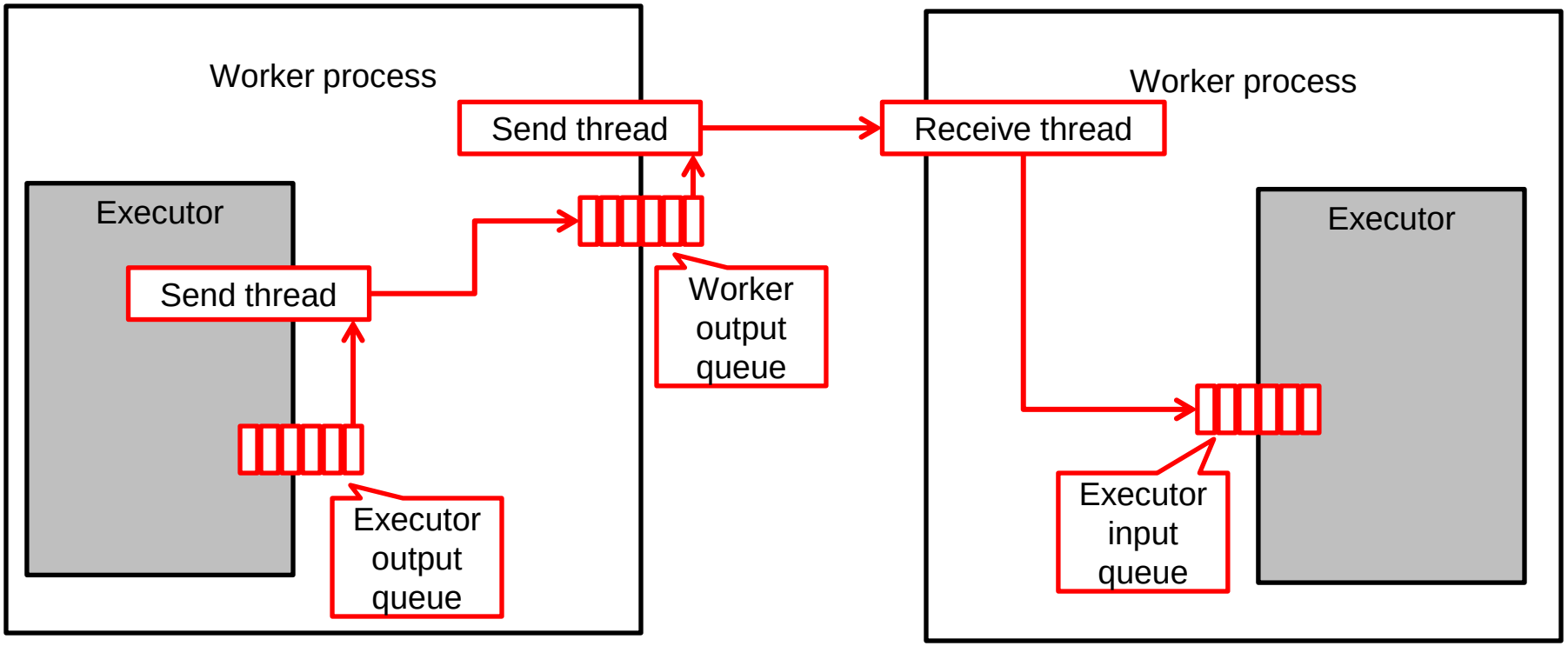
Apache Storm: Internals

Message passing between Executors



Apache Storm: Internals

Sending tuples between executors on different JVMs

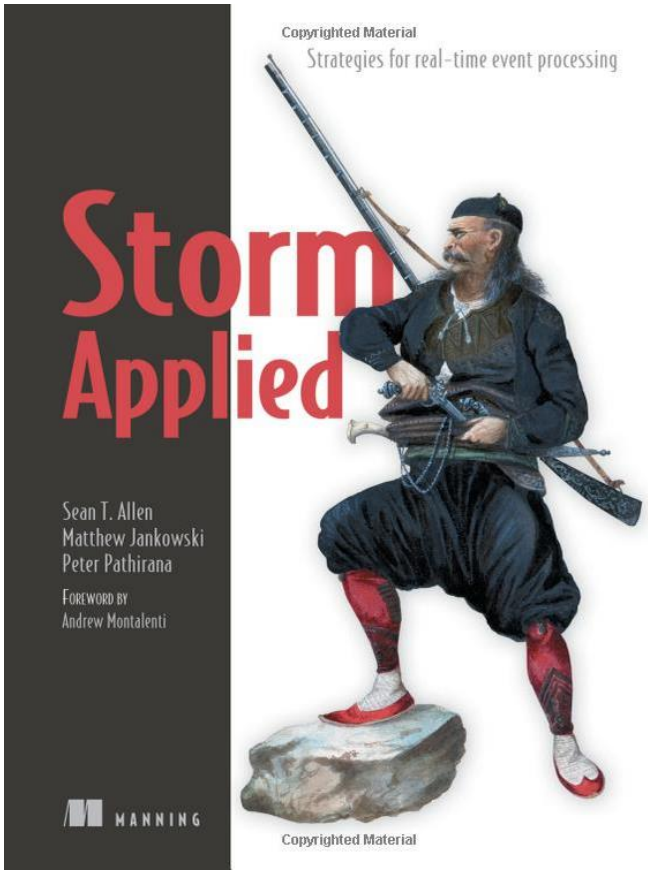


Remarks:

- 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

Agenda

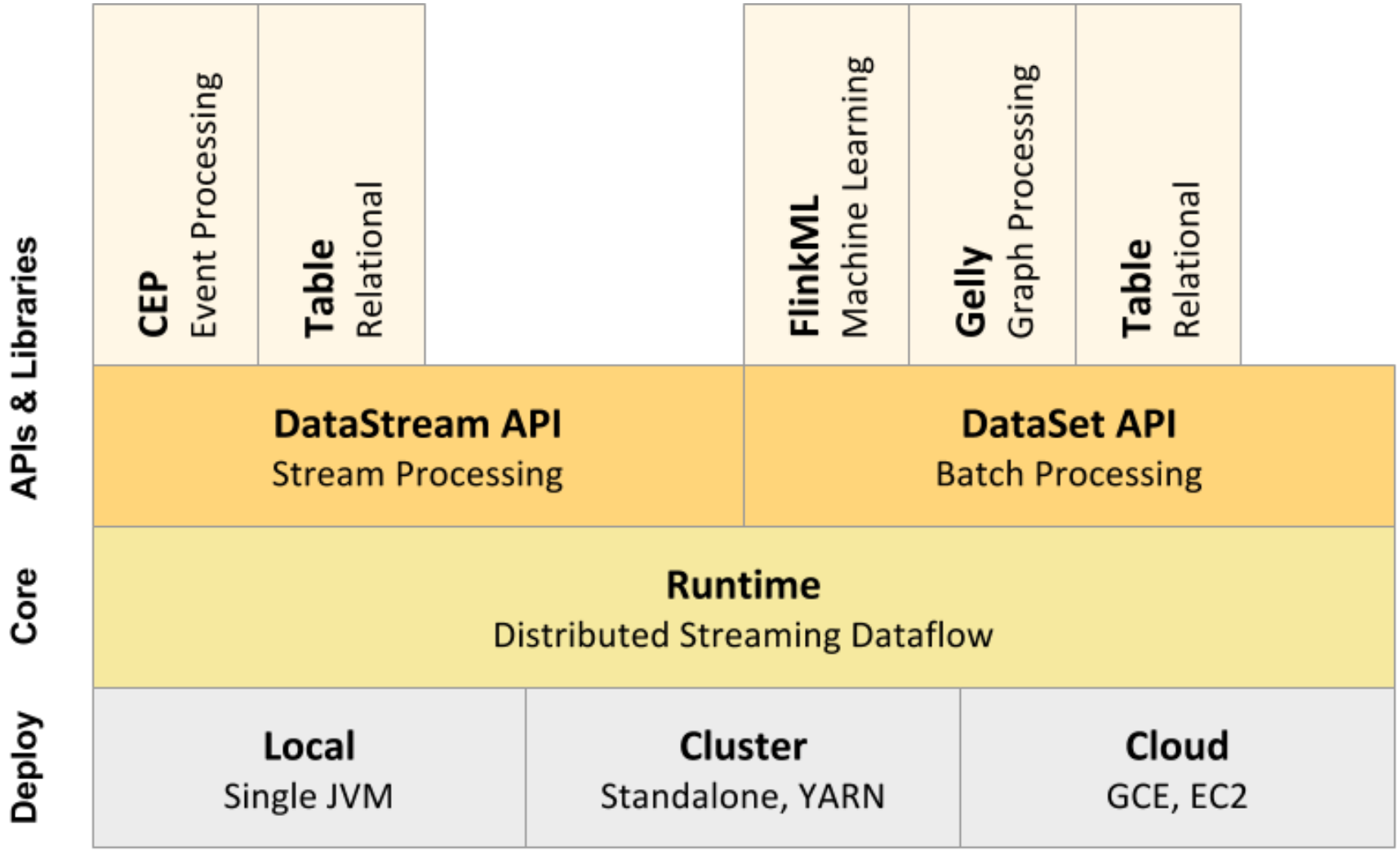
Use Cases
- Spark, Storm, and Flink applications

Streaming Systems Overview
- Closed Source (Commercial)
- Open Source (Apache/Academia)

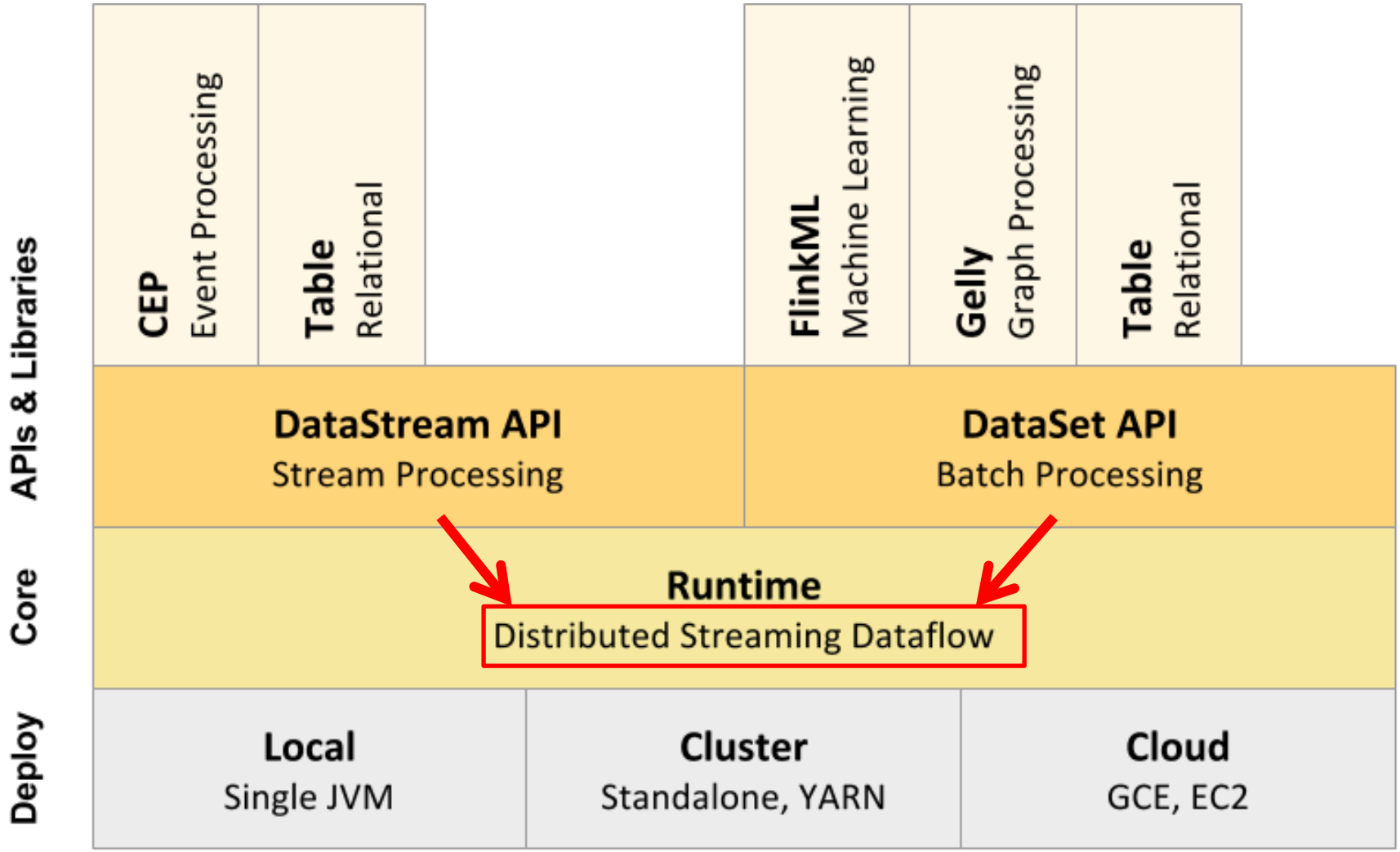
Apache Storm
- Basic Concepts
- Building a Topology
- Storm Internals

Apache Flink Streaming
- System Overview
- Windowing Semantics & Fault Tolerance
- Code Examples

Apache Flink



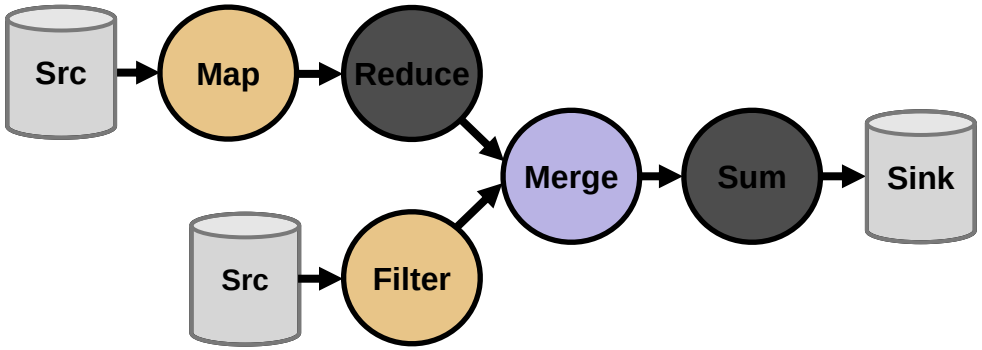
Apache Flink



Flink Streaming API overview

Data stream sources

- File system
- Message queue connectors
- Arbitrary source functionality



Stream transformations

- Basic transformations: *Map, Reduce, Filter, Aggregations*…
- Binary stream transformations: *CoMap, CoReduce*…
- Windowing semantics: *Policy based flexible windowing (Time, Count, Delta…)*
- Temporal binary stream operators: *Joins, Crosses*…
- Native support for iterations

Data stream outputs

For the details please refer to the programming guide:
http://ci.apache.org/projects/flink/flink-docs-master/apis/streaming_guide.html

Slide by Gyula Fóra

Word Count Example

DataSet API (batch):

```
val text = env.fromElements( „This is line 1", „Here comes line 2")

val counts = text.flatMap { _.toLowerCase.split("\\W+") filter { _.nonEmpty } }
          .map { (_, 1) }
          .groupBy(0)
          .sum(1)
```

Source: https://ci.apache.org/projects/flink/flink-docs-master/apis/batch/index.html

DataStream API (streaming):

```
val text = env.socketTextStream("localhost", 9999)

val counts = text.flatMap { _.toLowerCase.split("\\W+") filter { _.nonEmpty } }
          .map { (_, 1) }
          .keyBy(0)
          .timeWindow(Time.seconds(5))
          .sum(1)
          .print()
```

Source: https://ci.apache.org/projects/flink/flink-docs-master/apis/streaming/index.html

Fault Tolerance Solutions

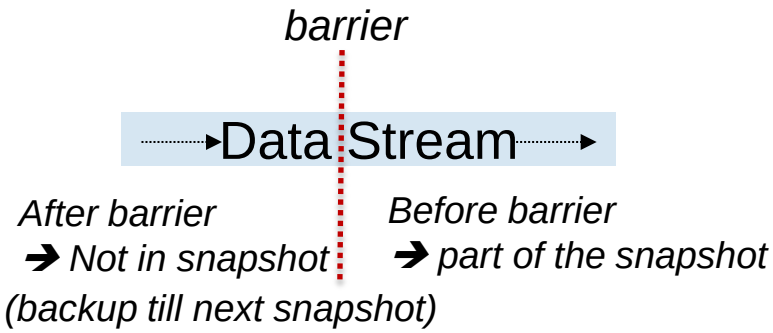
- Based on consistent global snapshots
- Algorithm inspired by Chandy-Lamport
- Low runtime overhead
- Stateful exactly-once semantics

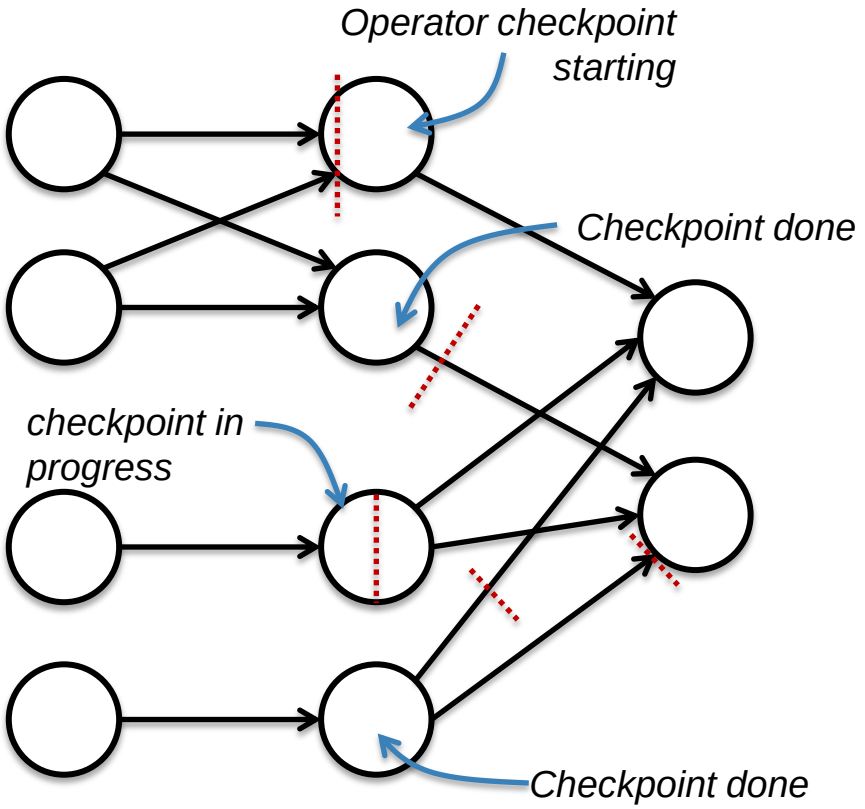
RDD re-computation

Message tracking/acks

Fault Tolerance with Snapshots

Push checkpoint barriers through the data flow



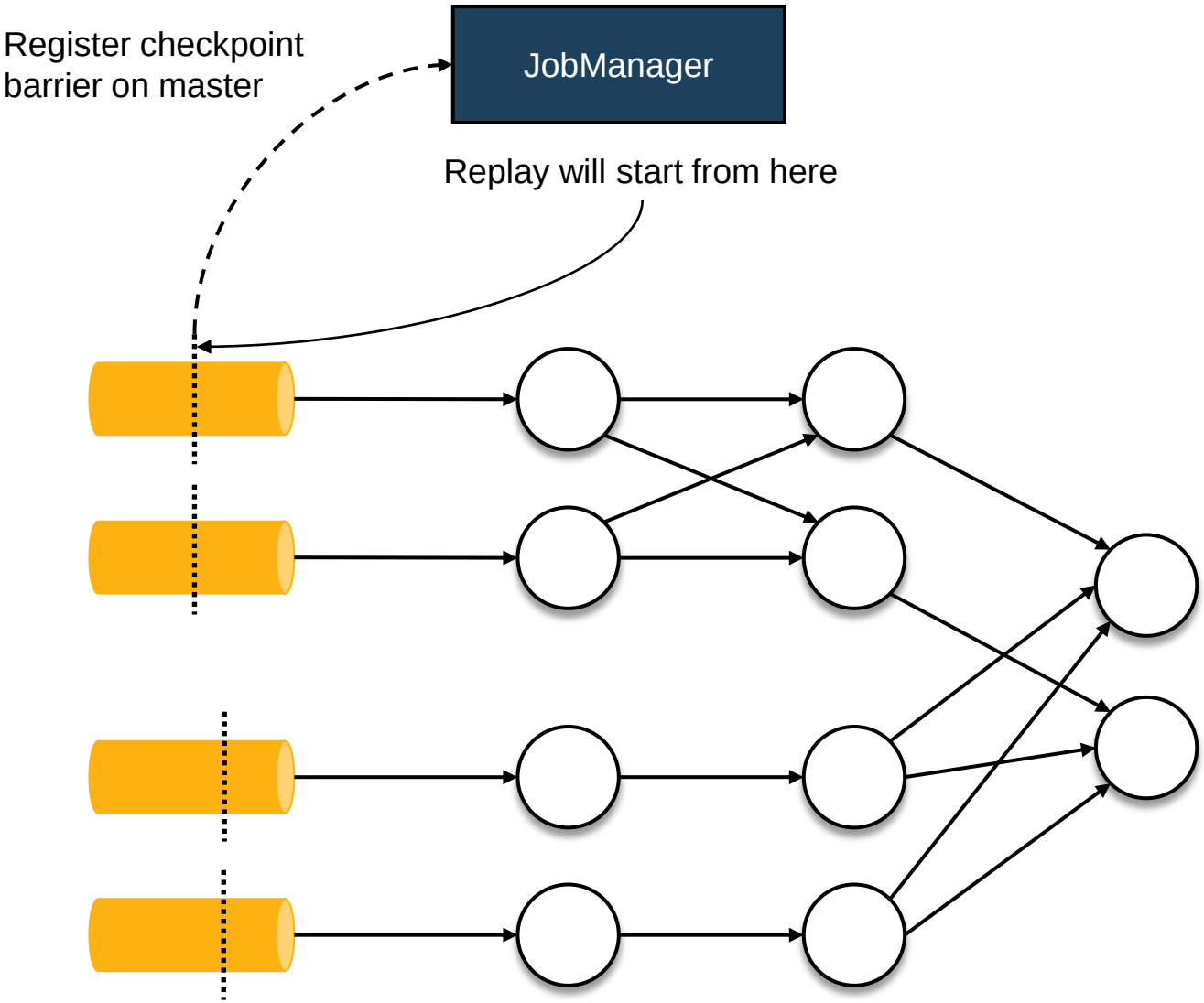


Asynchronous Barrier Snapshotting for globally consistent checkpoints

Recommended Reading: Lightweight asynchronous snapshots for distributed dataflows
P Carbone, G Fóra, S Ewen, S Haridi, K Tzoumas; arXiv preprint arXiv:1506.08603

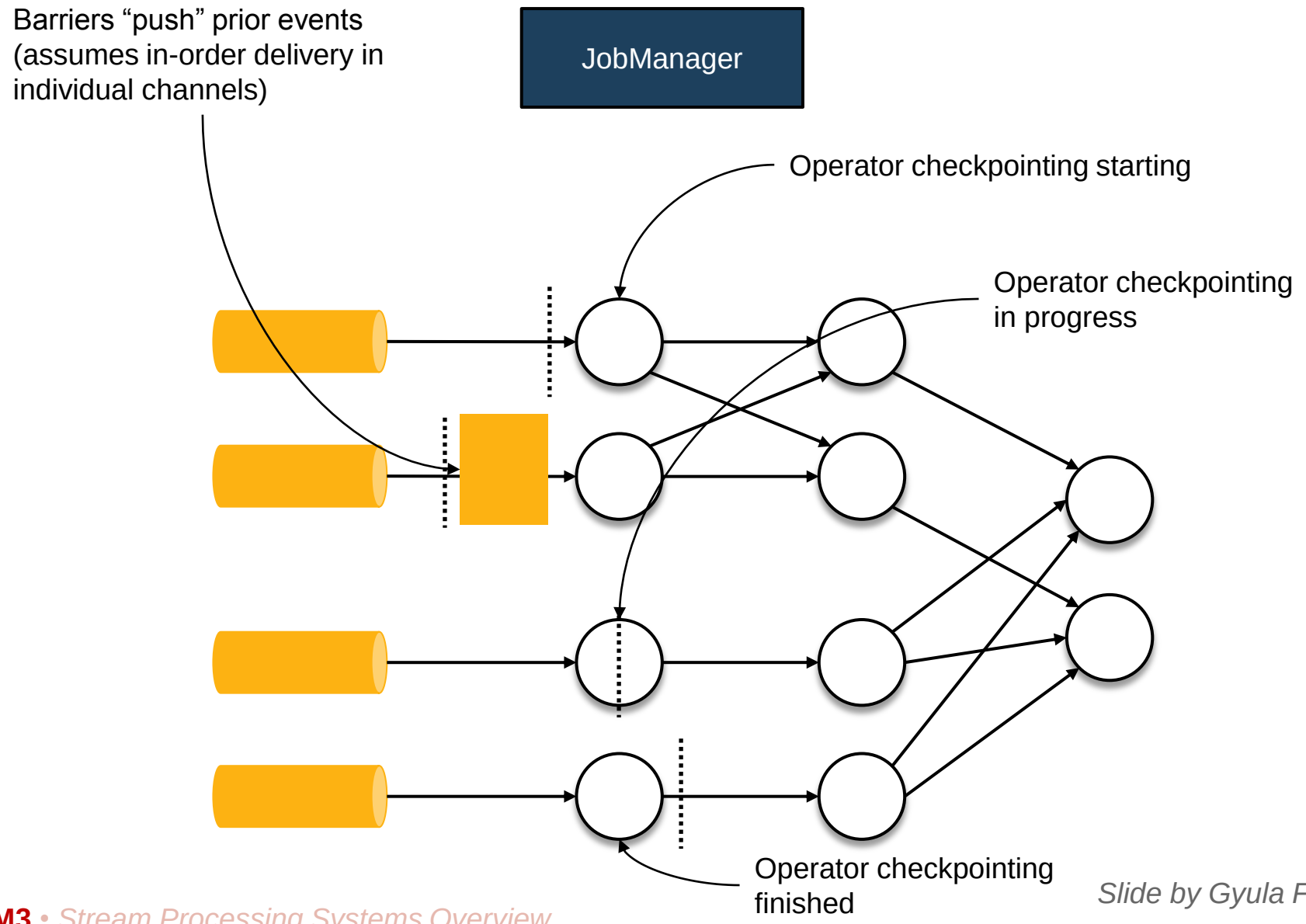
Slide by Gyula Fóra

Fault Tolerance with Snapshots



Slide by Gyula Fóra

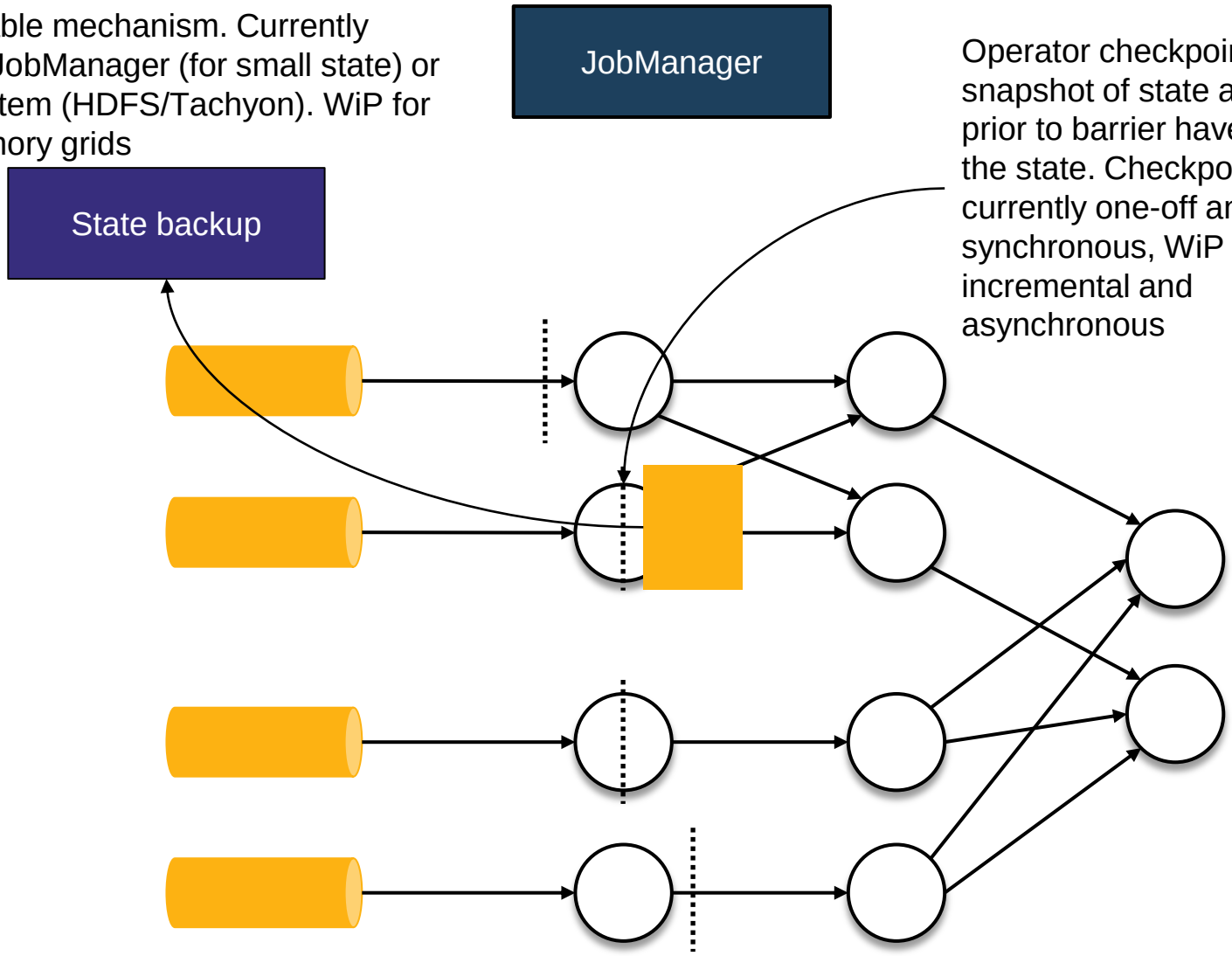
Fault Tolerance with Snapshots



Slide by Gyula Fóra

Fault Tolerance with Snapshots

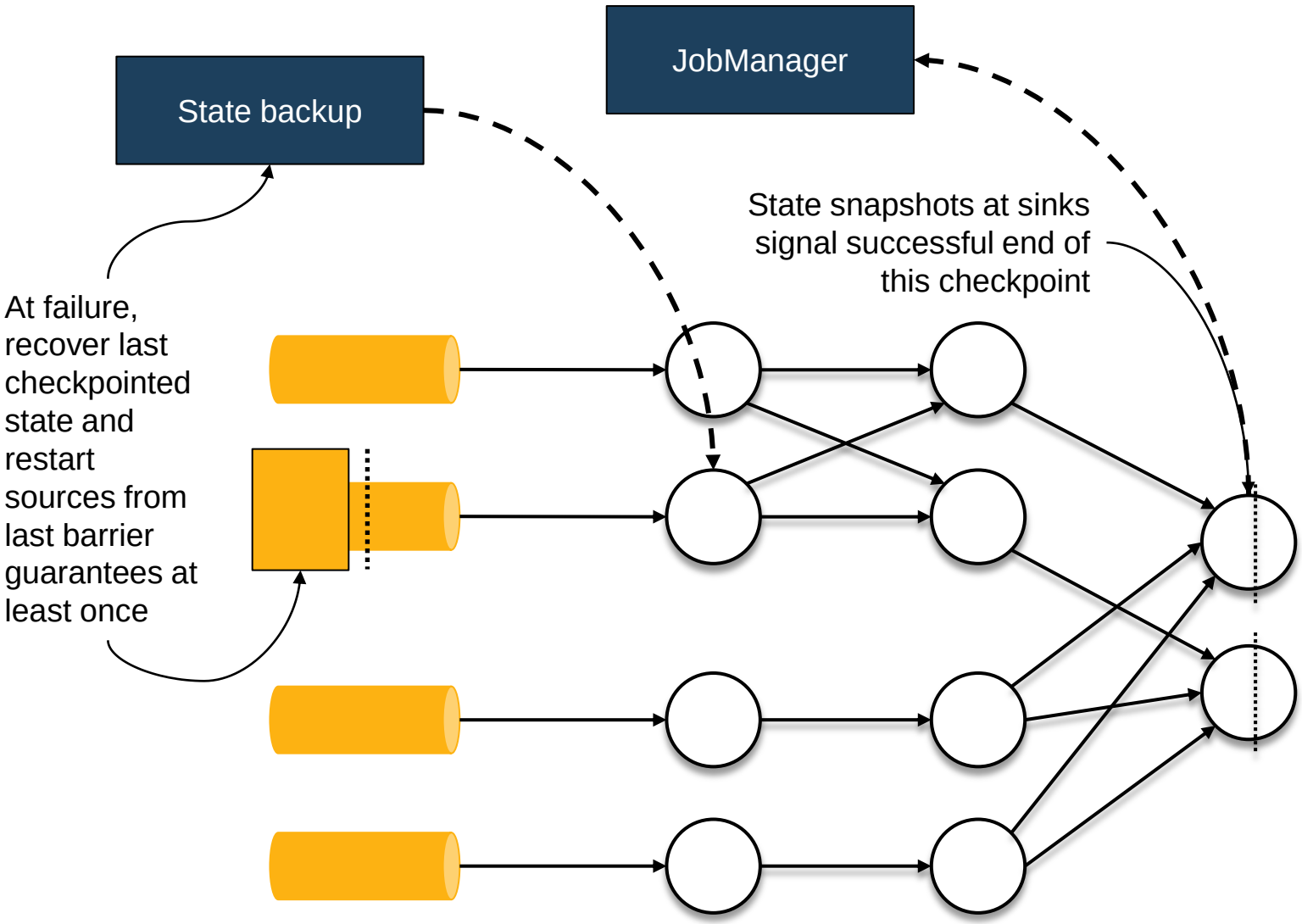
Pluggable mechanism. Currently either JobManager (for small state) or file system (HDFS/Tachyon). WiP for in-memory grids



Operator checkpointing takes snapshot of state after data prior to barrier have updated the state. Checkpoints currently one-off and synchronous, WiP for incremental and asynchronous

Slide by Gyula Fóra

Fault Tolerance with Snapshots



Slide by Gyula Fóra

References

(1) Murray, Derek G., et al. Naiad: a timely dataflow system. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013. S. 439-455.
http://dl.acm.org/citation.cfm?id=2522738

(2) Gedik, Buğra. Generic windowing support for extensible stream processing systems. *Software: Practice and Experience*, 2014, 44. Jg., Nr. 9, S. 1105-1128.
http://onlinelibrary.wiley.com/doi/10.1002/spe.2194/full

(3) Hirzel, Martin, et al. IBM Streams Processing Language: Analyzing big data in motion. *IBM Journal of Research and Development*, 2013, 57. Jg., Nr. 3/4, S. 7: 1-7: 11.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6517299

(4) Hirzel, Martin, et al. SPL stream processing language specification. *IBM Research, Yorktown Heights, NY, USA, Tech. Rep. RC24*, 2009, 897. Jg.T
http://domino.research.ibm.com/library/CyberDig.nsf/papers/DCF1CFDB512167AD85257677005DC3FB/$File/rc24897.pdf

(5) Zaharia, Matei, et al. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In: *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. USENIX Association, 2012. S. 10-10.
https://www.usenix.org/system/files/conference/hotcloud12/hotcloud12-final28.pdf

(6) Alexandrov, Alexander, et al. The Stratosphere platform for big data analytics. *The VLDB Journal-The International Journal on Very Large Data Bases*, 2014, 23. Jg., Nr. 6, S. 939-964.
http://dl.acm.org/citation.cfm?id=2691544

(7) Bernhardt, Thomas; Vasseur, Alexandre. Esper: Event stream processing and correlation. *ONJava, O'Reilly*, 2007.
http://www.onjava.com/lpt/a/6955

(8) Carney, Don, et al. Monitoring streams: a new class of data management applications. In: *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002. S. 215-226.
http://dl.acm.org/citation.cfm?id=1287389

(9) Chen, Jianjun, et al. NiagaraCQ: A scalable continuous query system for internet databases. In: *ACM SIGMOD Record*. ACM, 2000. S. 379-390.
http://dl.acm.org/citation.cfm?id=335432

(10) ARASU, Arvind; BABU, Shivnath; WIDOM, Jennifer. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal—The International Journal on Very Large Data Bases*, 2006, 15. Jg., Nr. 2, S. 121-142.
http://dl.acm.org/citation.cfm?id=1146463

AIM3
Stream Processing Systems Overview

Jonas Traub

An Overview of Stream Processing Optimizations

Presented by Jonas Traub

With materials from:

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

Agenda / Aim of the presentation

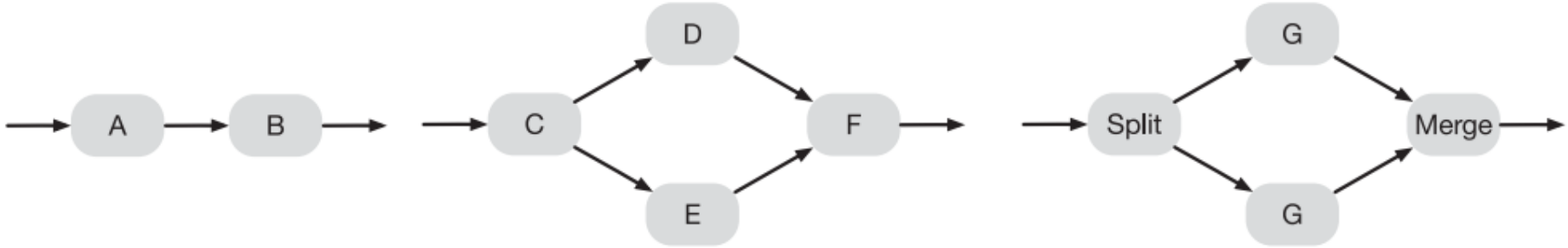
1. **Remember three types of parallelism**

2. **Show 11 optimization techniques for stream processing**

Aim:

→ Require less of your time than reading the paper
→ Show how to categorize known optimizations
→ Give entrance points for further reading

3 Types of Parallelization



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

11 Optimizations (numbered from 2 to 12 ☺)

Section	Optimization	Graph	Semantics	Dynamic
2.	Operator reordering	changed	unchanged	(depends)
3.	Redundancy elimination	changed	unchanged	(depends)
4.	Operator separation	changed	unchanged	static
5.	Fusion	changed	unchanged	(depends)
6.	Fission	changed	(depends)	(depends)
7.	Placement	unchanged	unchanged	(depends)
8.	Load balancing	unchanged	unchanged	(depends)
9.	State sharing	unchanged	unchanged	static
10.	Batching	unchanged	unchanged	(depends)
11.	Algorithm selection	unchanged	(depends)	(depends)
12.	Load shedding	unchanged	changed	dynamic

Structure of the Paper

Tag Line	gives a quick intuition for the policy, algorithm, and transformation underlying the optimization.
Example	sketches a concrete real-world application, which illustrates what the optimization does and motivates why it is useful.
Profitability	describes the conditions that a policy needs to consider for the optimization to improve performance. Each profitability subsection is based on a microbenchmark.
Safety	lists the conditions necessary for the optimization to preserve correctness. Formally, the optimization is only safe if the conjunction of the conditions is true.
Variations	surveys the most influential and unique work on this optimization in the literature.
Dynamism	identifies established approaches for applying the optimization dynamically instead of statically (i.e., during runtime).

Structure of the Paper

Tag Line	gives a quick intuition for the policy, algorithm, and transformation underlying the optimization.
Example	sketches a concrete real-world application, which illustrates what the optimization does and motivates why it is useful.
Profitability	describes the conditions that a policy needs to consider for the optimization to improve performance. Each profitability subsection is based on a microbenchmark.
Safety	lists the conditions necessary for the optimization to preserve correctness. Formally, the optimization is only safe if the conjunction of the conditions is true.
Variations	surveys the most influential and unique work on this optimization in the literature.
Dynamism	identifies established approaches for applying the optimization dynamically instead of statically (i.e., during runtime).

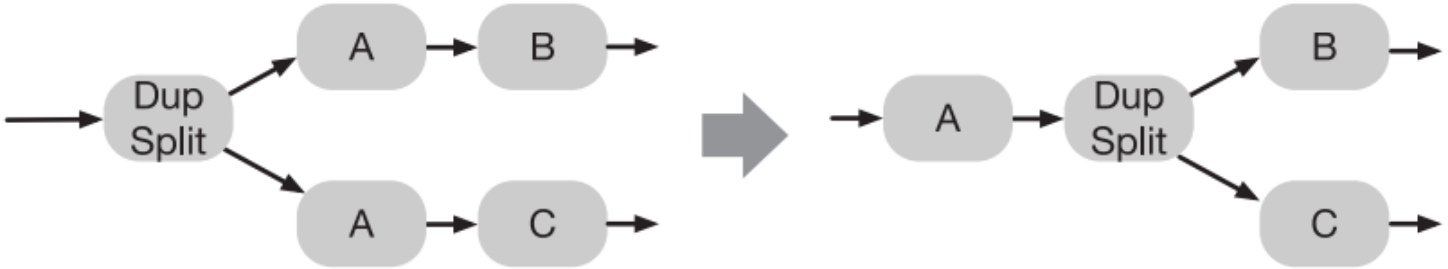
2) Operator Reordering

2. OPERATOR REORDERING (A.K.A. HOISTING, SINKING, ROTATION, PUSH-DOWN)

Move more selective operators upstream to filter data early.

3. REDUNDANCY ELIMINATION (A.K.A. SUBGRAPH SHARING, MULTIQUERY OPTIMIZATION)

Eliminate redundant computations.



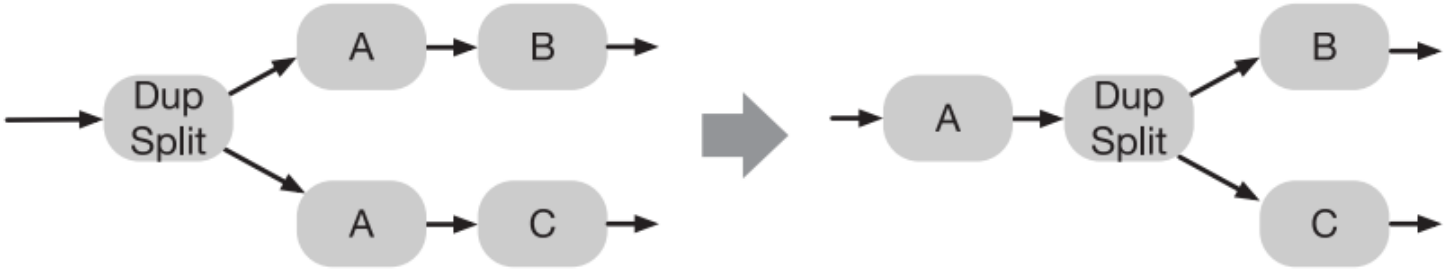
3) Redundancy Elimination

2. OPERATOR REORDERING (A.K.A. HOISTING, SINKING, ROTATION, PUSH-DOWN)

Move more selective operators upstream to filter data early.

3. REDUNDANCY ELIMINATION (A.K.A. SUBGRAPH SHARING, MULTIQUERY OPTIMIZATION)

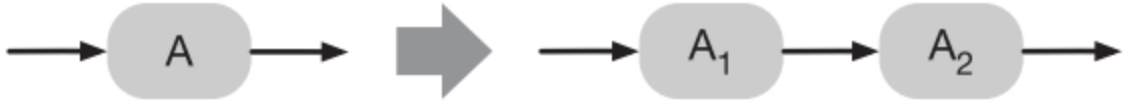
Eliminate redundant computations.



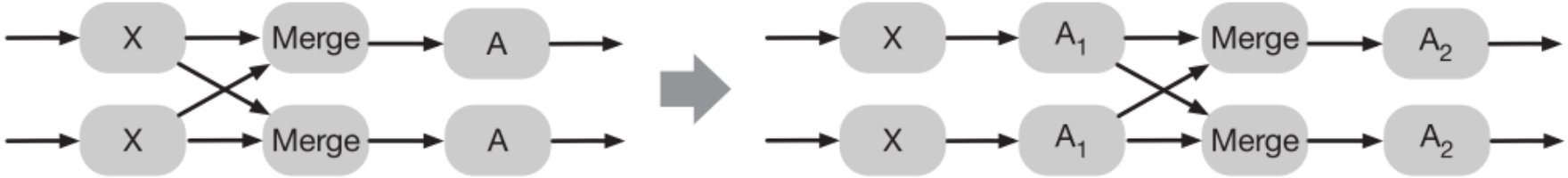
4) Operator Separation

4. OPERATOR SEPARATION (A.K.A. DECOUPLED SOFTWARE PIPELINING)

Separate operators into smaller computational steps.



Operator separation is profitable if it enables other optimizations such as operator reordering or fission, or if the resulting pipeline parallelism pays off when running on multiple cores.



5) Fusion

5. FUSION (A.K.A. SUPERBOX SCHEDULING)

Avoid the overhead of data serialization and transport.

5) Fusion

5. FUSION (A.K.A. SUPERBOX SCHEDULING)

Avoid the overhead of data serialization and transport.

In Apache Flink (and many other applications) we call this

chaining

Goal:

Reduce communication costs

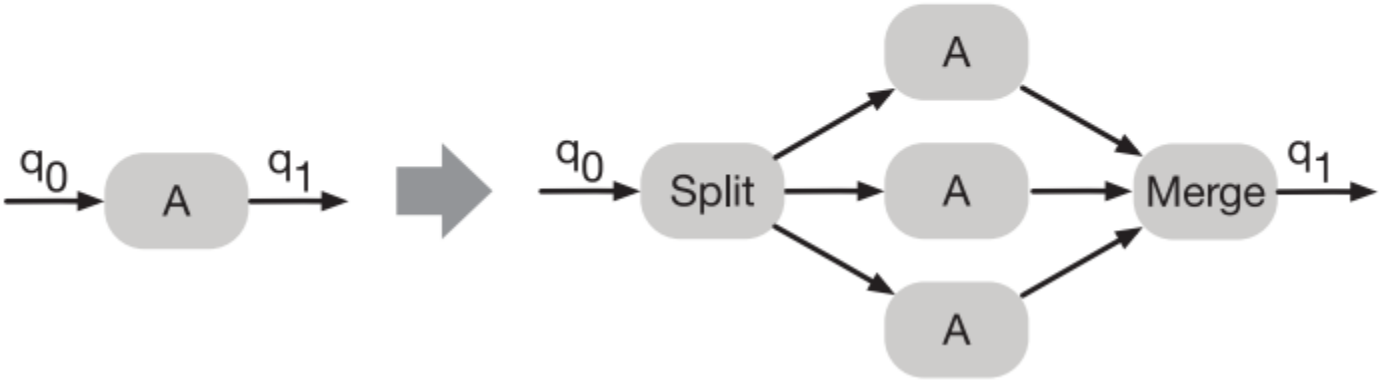
Method:

Shared memory among operators instead of network communication

6) Fission

6. FISSION (A.K.A. PARTITIONING, DATA PARALLELISM, REPLICATION)

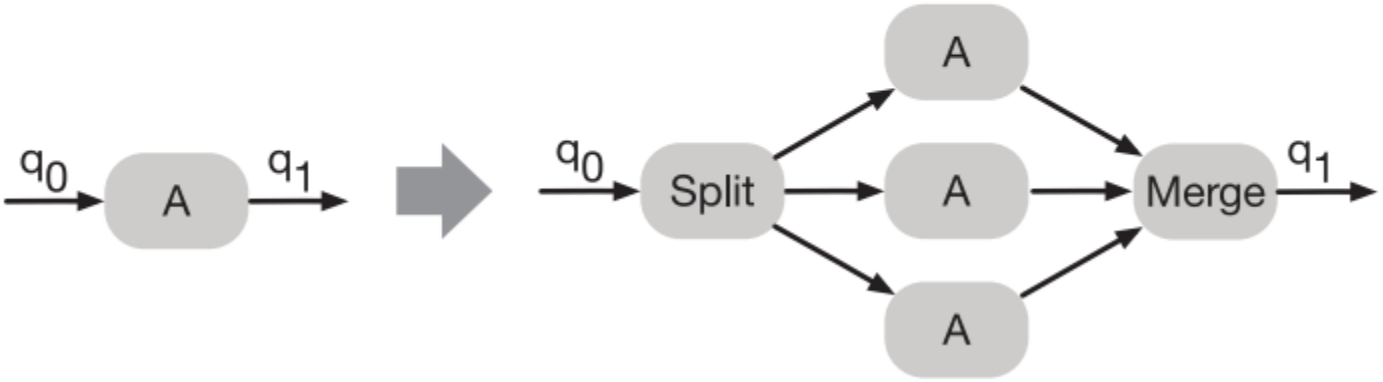
Parallelize computations.



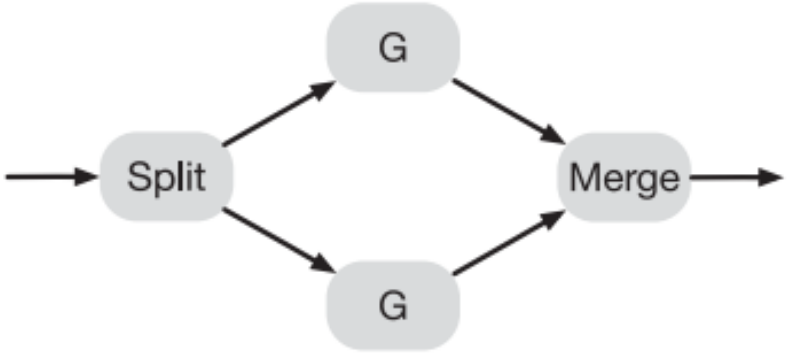
6) Fission

6. FISSION (A.K.A. PARTITIONING, DATA PARALLELISM, REPLICATION)

Parallelize computations.



Directly maps to data parallelism:

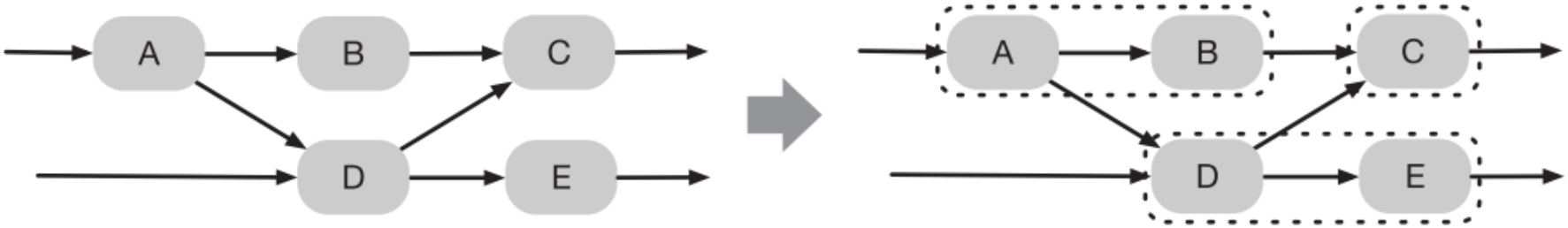


(c) Data-parallel G ‖ G.

7) Placement

7. PLACEMENT (A.K.A. LAYOUT)

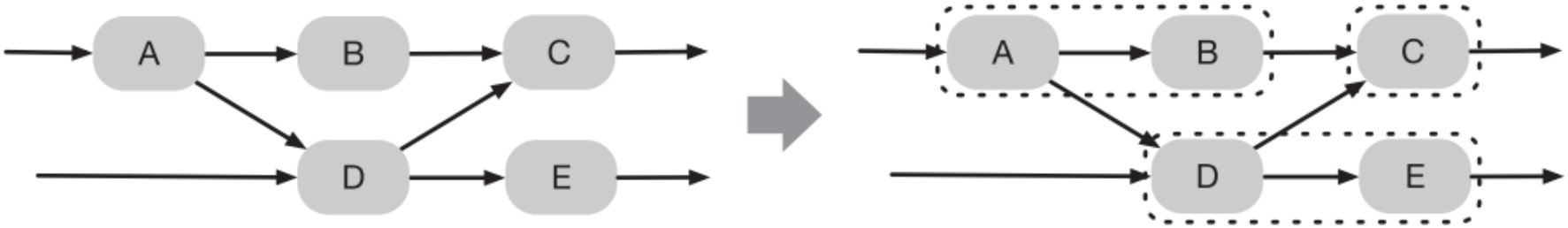
Assign operators to hosts and cores.



7) Placement

7. PLACEMENT (A.K.A. LAYOUT)

Assign operators to hosts and cores.



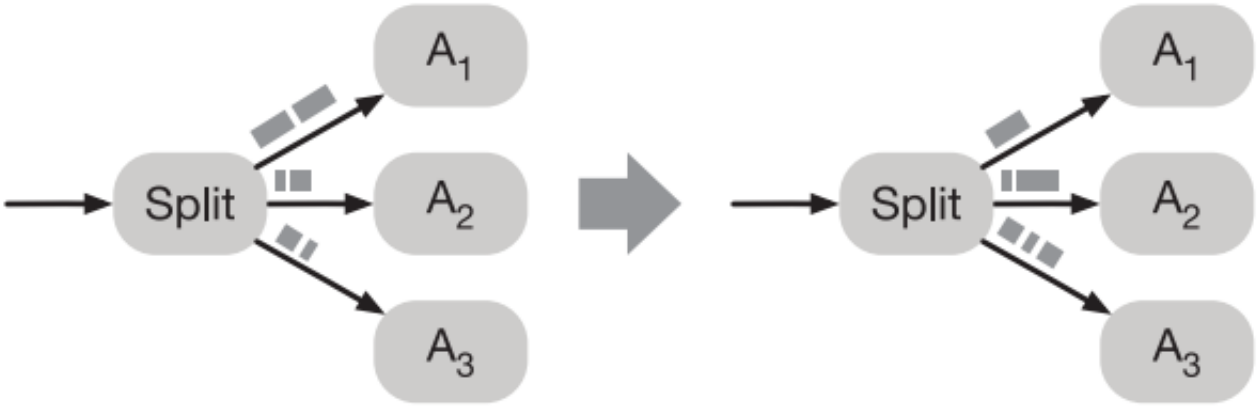
Assigning Operators to slots

Co-locating Data and Computations

8) Load Balancing

8. LOAD BALANCING

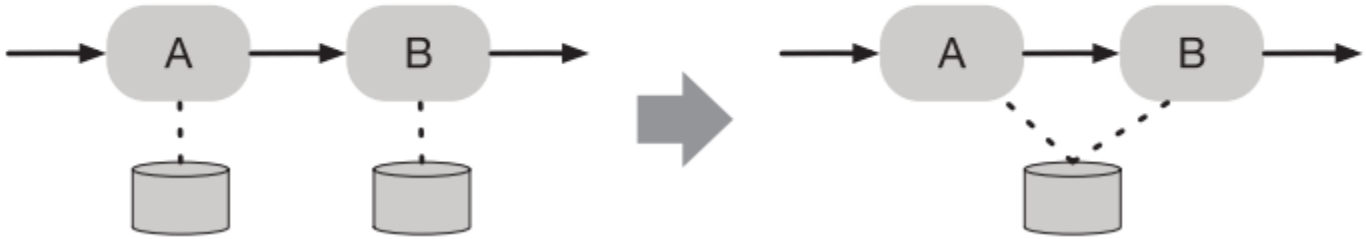
Distribute workload evenly across resources.



9) State Sharing

9. STATE SHARING (A.K.A. SYNOPSIS SHARING, DOUBLE-BUFFERING)

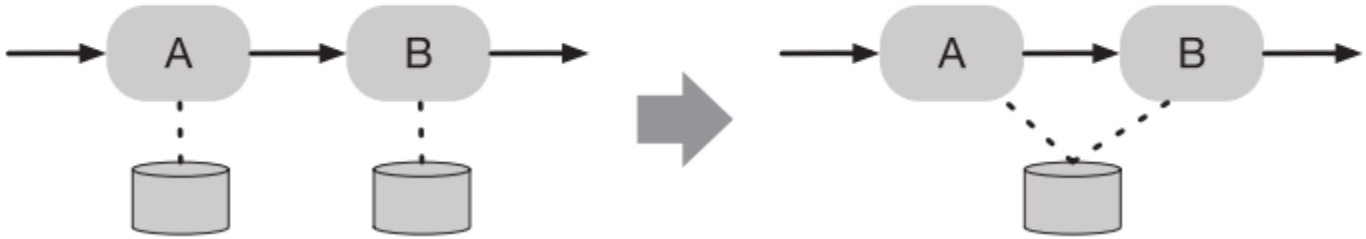
Optimize for space by avoiding unnecessary copies of data.



9) State Sharing

9. STATE SHARING (A.K.A. SYNOPSIS SHARING, DOUBLE-BUFFERING)

Optimize for space by avoiding unnecessary copies of data.



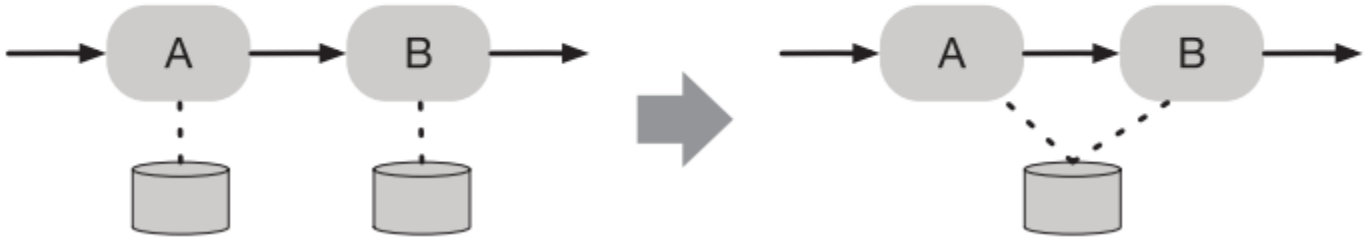
Distributed File Systems

A single storage layer for the whole cluster

9) State Sharing

9. STATE SHARING (A.K.A. SYNOPSIS SHARING, DOUBLE-BUFFERING)

Optimize for space by avoiding unnecessary copies of data.



Distributed File Systems

A single storage layer for the whole cluster

Chaining again…

Share memory among several operators instead of copying the data

10) Batching

10. BATCHING (A.K.A. TRAIN SCHEDULING, EXECUTION SCALING)

Process multiple data items in a single batch.

10) Batching

10. BATCHING (A.K.A. TRAIN SCHEDULING, EXECUTION SCALING)

Process multiple data items in a single batch.

Flink

"Under the hood" batch wise network traffic (buffering)

Spark

D-Streams*: All the stream processing is done in micro-batches
(Yes, you can sell a workaround as optimization ☺)

* Zaharia, Matei, et al. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In: *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. USENIX Association, 2012. S. 10-10. https://www.usenix.org/system/files/conference/hotcloud12/hotcloud12-final28.pdf

11) Algorithm selection

11. ALGORITHM SELECTION (A.K.A. TRANSLATION TO PHYSICAL QUERY PLAN)

Use a faster algorithm for implementing an operator.

The optimizer selects the (hopefully) optimal join implementation

12) Load Shedding

11. ALGORITHM SELECTION (A.K.A. TRANSLATION TO PHYSICAL QUERY PLAN)

Use a faster algorithm for implementing an operator.

The optimizer selects the (hopefully) optimal join implementation

12. LOAD SHEDDING (A.K.A. ADMISSION CONTROL, GRACEFUL DEGRADATION)

Degrade gracefully when overloaded.

An Overview of Stream Processing Optimizations

Presented by Jonas Traub

With materials from:

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

AIM3
Stream Processing
Jonas Traub