Apache Gearpump
next-gen streaming engine

Karol Brejna, Intel (karolbrejna@apache.org)

Huafeng Wang, Intel (huafengw@apache.org)

Apache: Big Data Europe 2016 Sevilla, Spain 14 November 2016

Agenda

- What is Gearpump?
- Why Apache Gearpump?
- Apache Gearpump features/internals
- What’s next for Apache Gearpump

What is Gearpump ?

- A super simple pump that consists of only two gears but very powerful at streaming water
- An Akka[2] based real-time streaming engine
- An Apache Incubator[1] project since Mar.8th, 2016

Why Gearpump ?

Stream processing is hard

- Fault tolerance
- Infinite Out-of-order data
- Low latency assurance (e.g real-time recommendation)
- Correctness requirement (e.g. charge advertisers for ads)
- Cheap to update applications (e.g. tune machine learning parameters)

Gearpump makes stream processing easier

- fault tolerant stream processing at latency of milliseconds
- handling out-of-order data
- event-time based window aggregation
- Akka-stream DSL and Apache Beam API support
- runtime DAG modification
- responsive UI with abundant metrics information

Gearpump on TAP

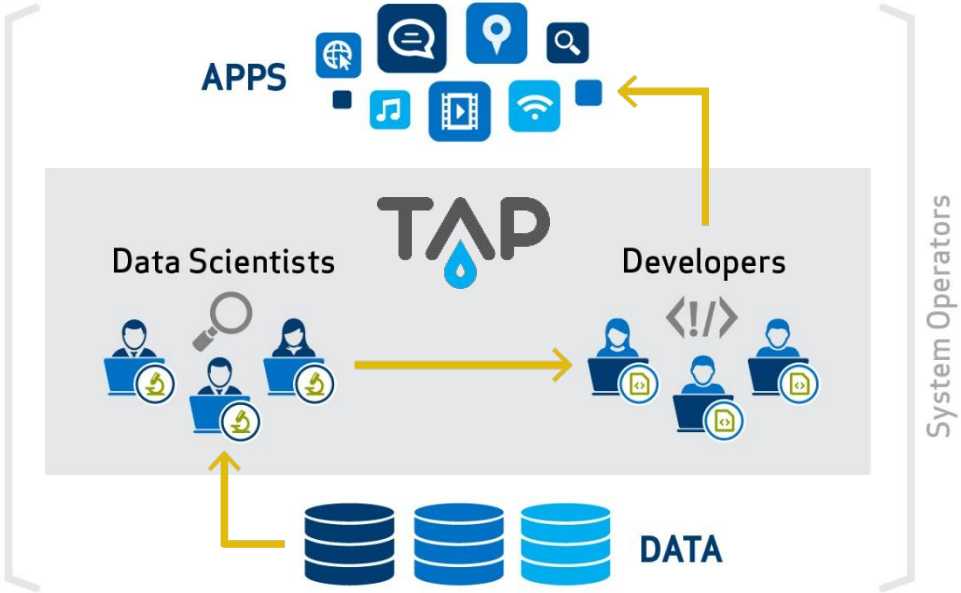
- Gearpump on Trusted Analytics Platform (TAP)
- Stream processing - performance experiments and results

Gearpump on TAP

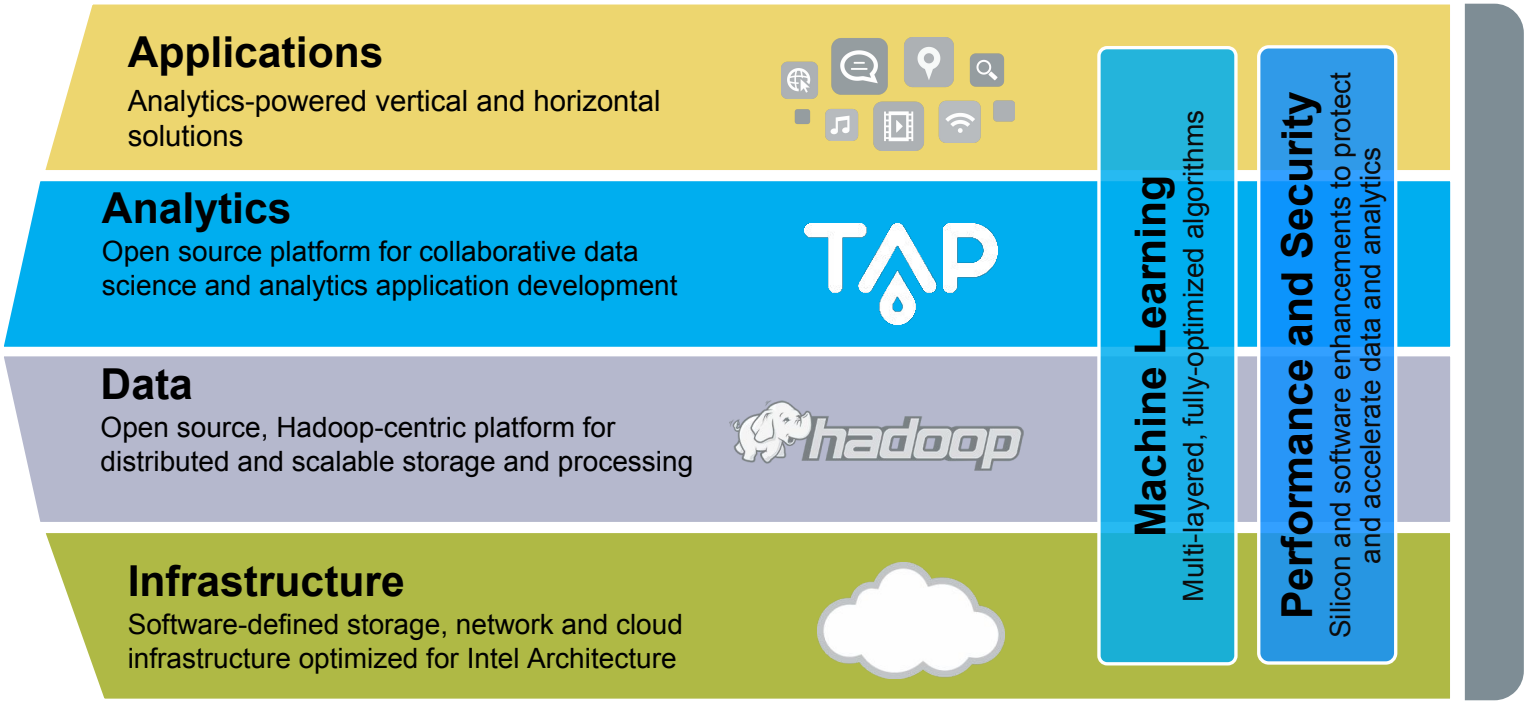
- Gearpump on Trusted Analytics Platform (TAP)
- Stream processing - performance experiments and results

Trusted Analytics Platform (TAP)

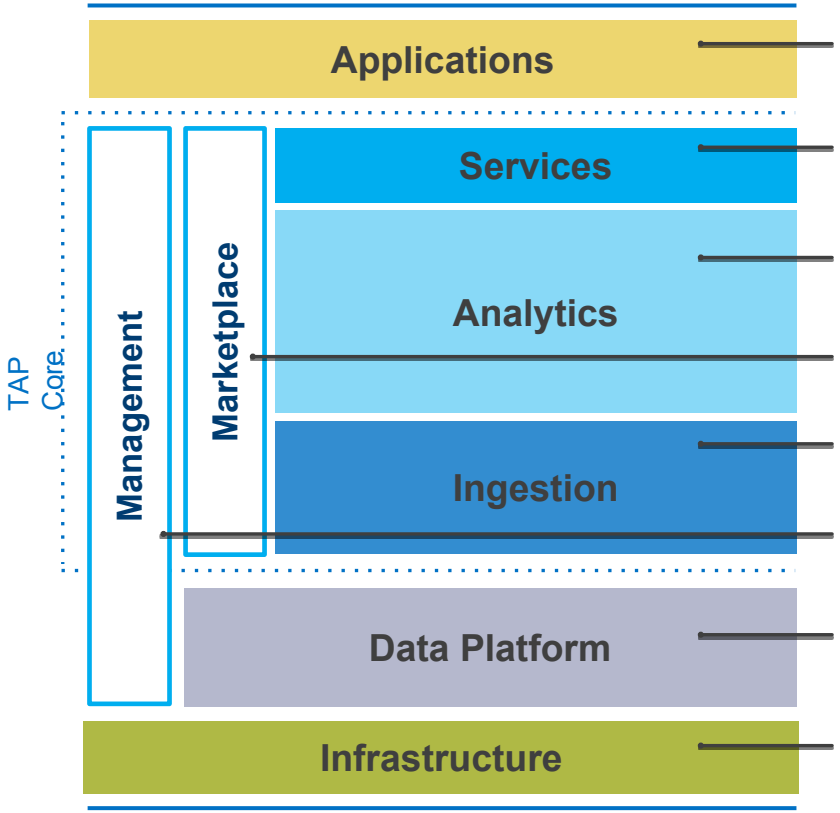
- Open Source project
- Collaborative, cloud-ready platform to build applications powered by Big Data Analytics
- Includes everything needed by data scientists, application developers and system operators
- Optimized for performance and security



Analytics Solutions – Big Data Scale Out



The Anatomy of Trusted Analytics Platform (TAP)



TAP-powered Big Data Analytics applications and solutions — Java, Go

Polyglot services and APIs for ***application developers*** — REST

Data Scientists workbench including models, algorithms, pipelines, engines and frameworks — ATK, **Spark***, Impala, H2O, **Hue,*** iPython

Extensible Marketplace of built-in tools, packages and recipes

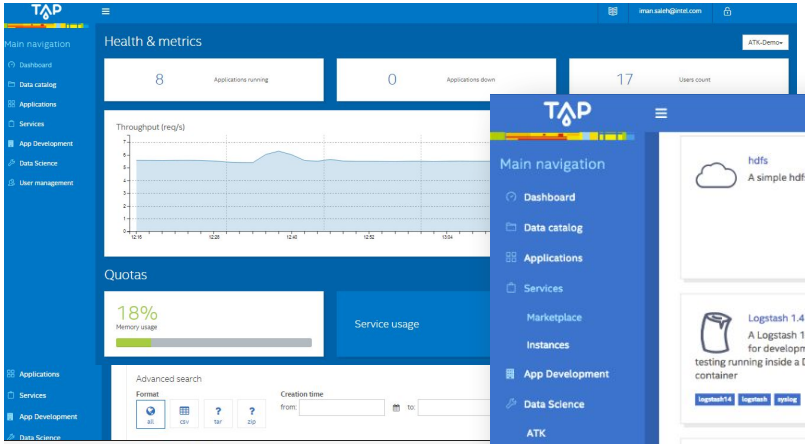
Message brokers and queues for batch and stream data ingestion — **Kafka*, GearPump**, RabbitMQ, MQTT, WS, REST

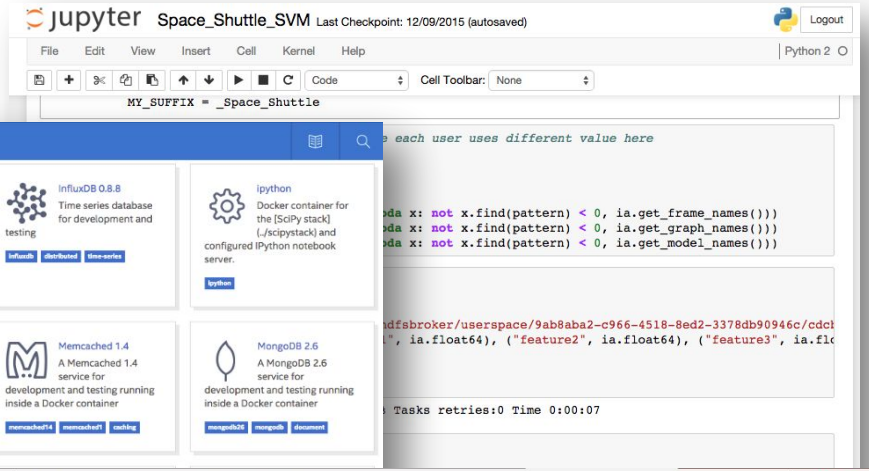
User, tenant, **security**, provisioning and monitoring for *system operators*

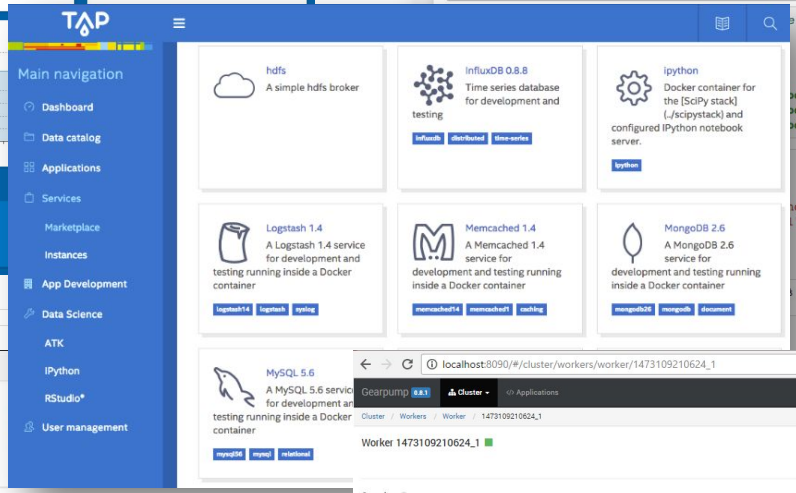
Distributed processing and scalable data storage — **Cloudera CDH (Hadoop/HDFS, Hbase)***, PostgreSQL, MySQL, Redis, MongoDB, InfluxDB, Cassandra

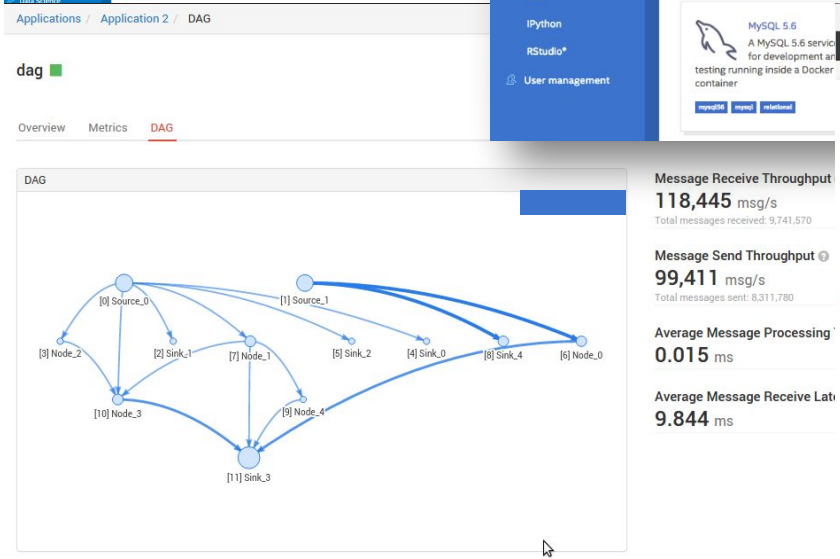
Public or private clouds — AWS, Rackspace, OVH, OpenStack, On/Off-prem

*** Leverages Cloudera Distribution of Apache Hadoop**

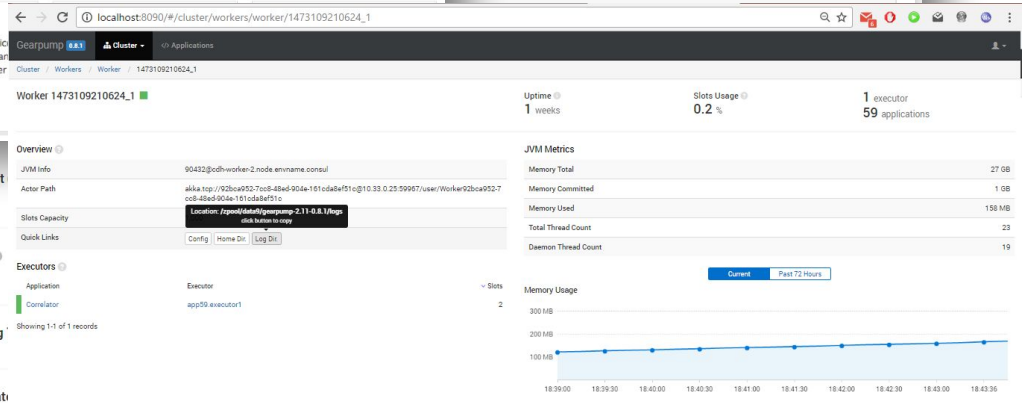

TAP
Main navigation
Dashboard
Data catalog
Applications
Services
App Development
Data Science
User management
Health & metrics
ATK-Demo
8
Applications running
0
Applications down
17
Users count
Throughput (req/s)
Quotas
18%
Memory usage
Service usage
Advanced search
Format
Creation time


jupyter
Space_Shuttle_SVM
Last Checkpoint: 12/09/2015 (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Python 2
Code
Cell Toolbar:
None
MY_SUFFIX = _Space_Shuttle
each user uses different value here
Tasks retries:0 Time 0:00:07


TAP
Main navigation
Dashboard
Data catalog
Applications
Services
Marketplace
Instances
App Development
Data Science
ATK
IPython
RStudio*
User management
hdfs
A simple hdfs broker
InfluxDB 0.8.8
Time series database for development and testing
ipython
Docker container for the [SciPy stack] (../scipystack) and configured IPython notebook server.
Logstash 1.4
A Logstash 1.4 service for development and testing running inside a Docker container
Memcached 1.4
A Memcached 1.4 service for development and testing running inside a Docker container
MongoDB 2.6
A MongoDB 2.6 service for development and testing running inside a Docker container
MySQL 5.6
testing running inside a Docker container


Applications / Application 2 / DAG
dag
Overview
Metrics
DAG
DAG
[0] Source_0
[1] Source_1
[3] Node_2
[2] Sink_1
[7] Node_1
[5] Sink_2
[4] Sink_0
[8] Sink_4
[6] Node_0
[10] Node_3
[9] Node_4
[11] Sink_3

Message Receive Throughput
118,445 msg/s
Total messages received: 9,741,570
Message Send Throughput
99,411 msg/s
Total messages sent: 8,311,780
Average Message Processing
0.015 ms
Average Message Receive Lat
9.844 ms


localhost:8090/#/cluster/workers/1473109210624_1
Gearpump
Cluster
Applications
Worker 1473109210624_1
Uptime
1 weeks
Slots Usage
0.2 %
1 executor
59 applications
Overview
JVM Info
Actor Path
Slots Capacity
Quick Links
Executors
JVM Metrics
Memory Total
Memory Committed
Memory Used
Total Thread Count
Daemon Thread Count
Memory Usage

Wearables

Customer behavior analysis using wearable devices

Healthcare

Predict individual health problems to improve care

Retail

Asset management with RFID data

Industrial

Predict equipment failures and optimization based on sensor data

Genomics

Execute privacy-preserving analytics on diverse distributed data sets

IoT Developer Platform

Enable development of data-generating IoT apps

Security

Detect threats – IT, grid, machines, sites

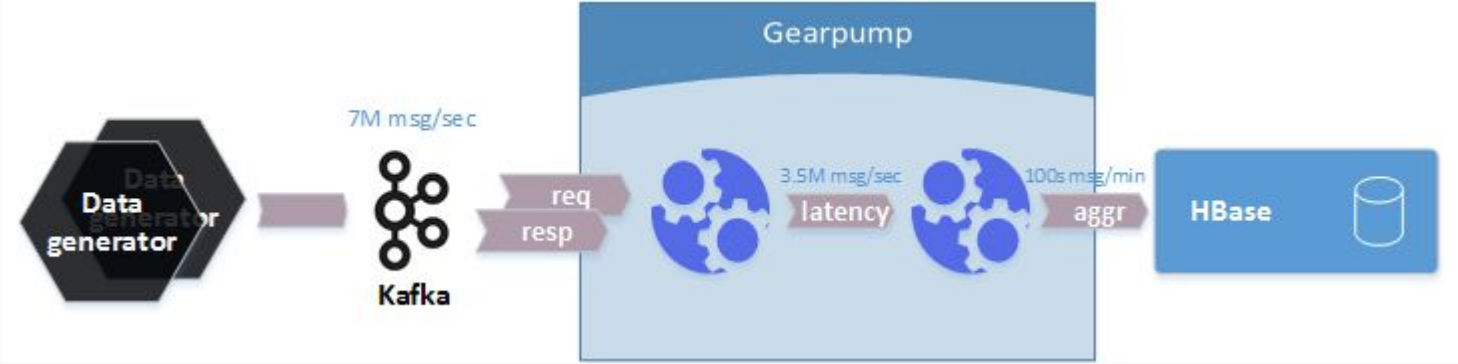
Forecasting

Financial, inventory, supply chain, etc.

Gearpump on TAP

- Gearpump on Trusted Analytics Platform (TAP)
- Stream processing - performance experiments and results

The problem

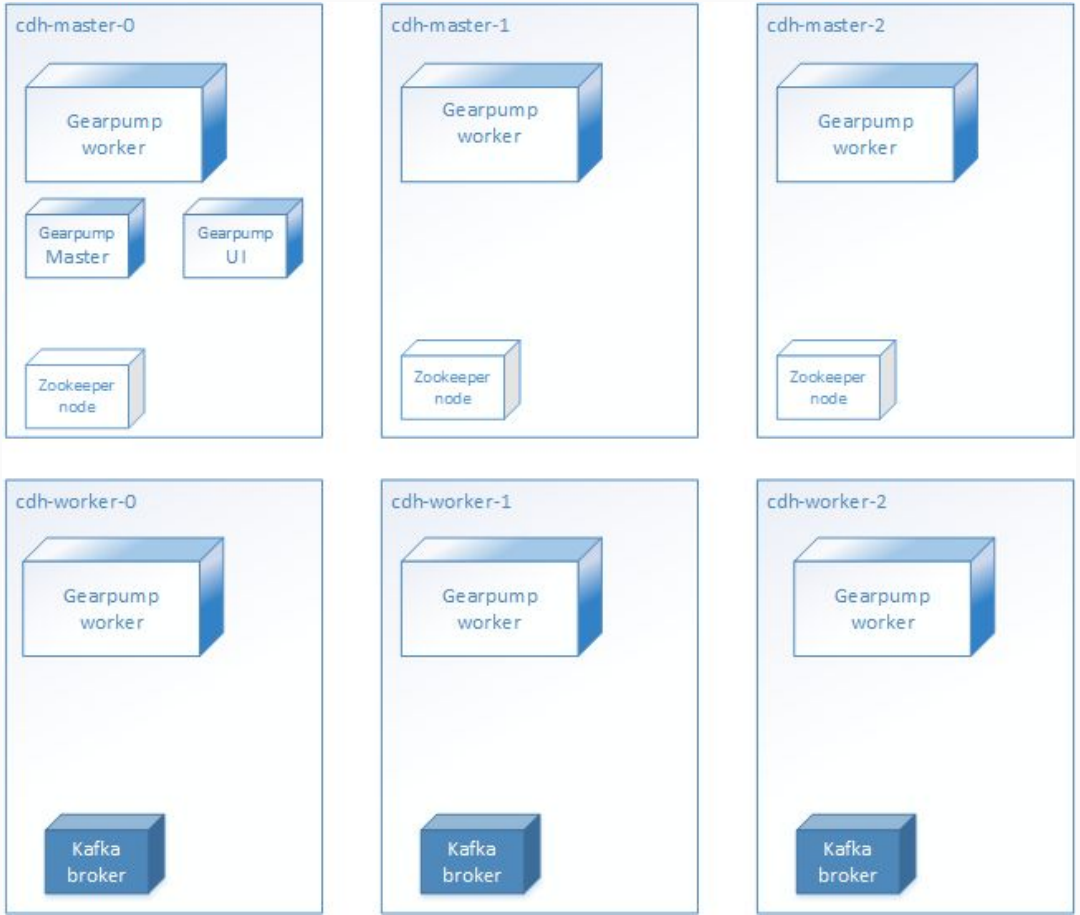


- correlate messages using a key in one second sliding window and produce latency stream messages
- consume latency messages and compute average latency per firm in one minute buckets
- send the aggregate message to HBase

The expectations

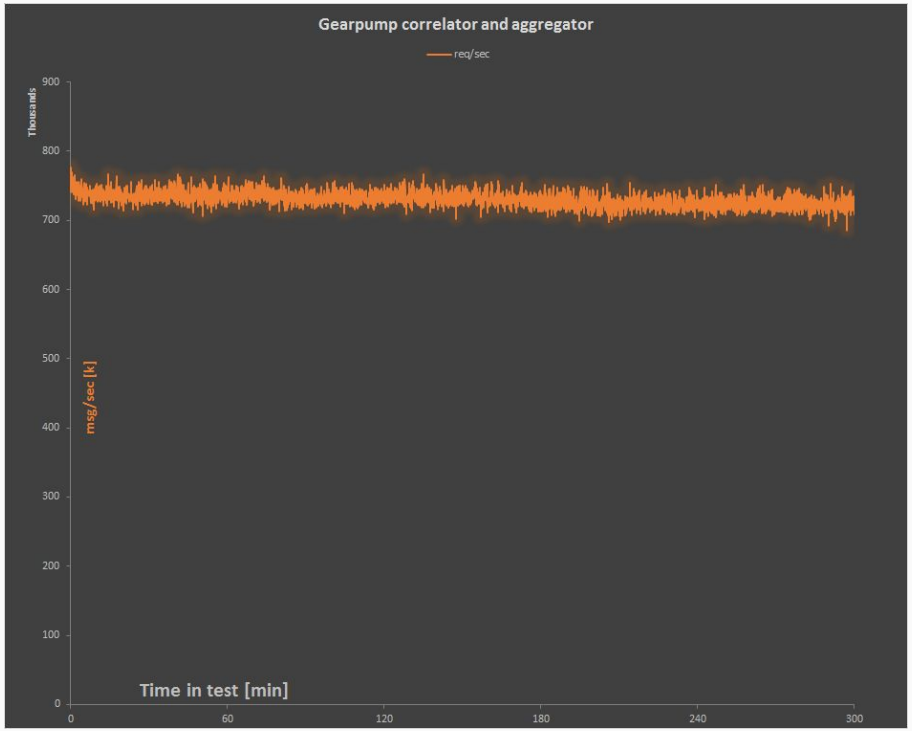
- Handle load of 0.5M msg per second all the time
- Handle load of 7M msg per second for peaks of 1 hour
- Message size 250-500 bytes
- Be able to scale for even more

The hardware



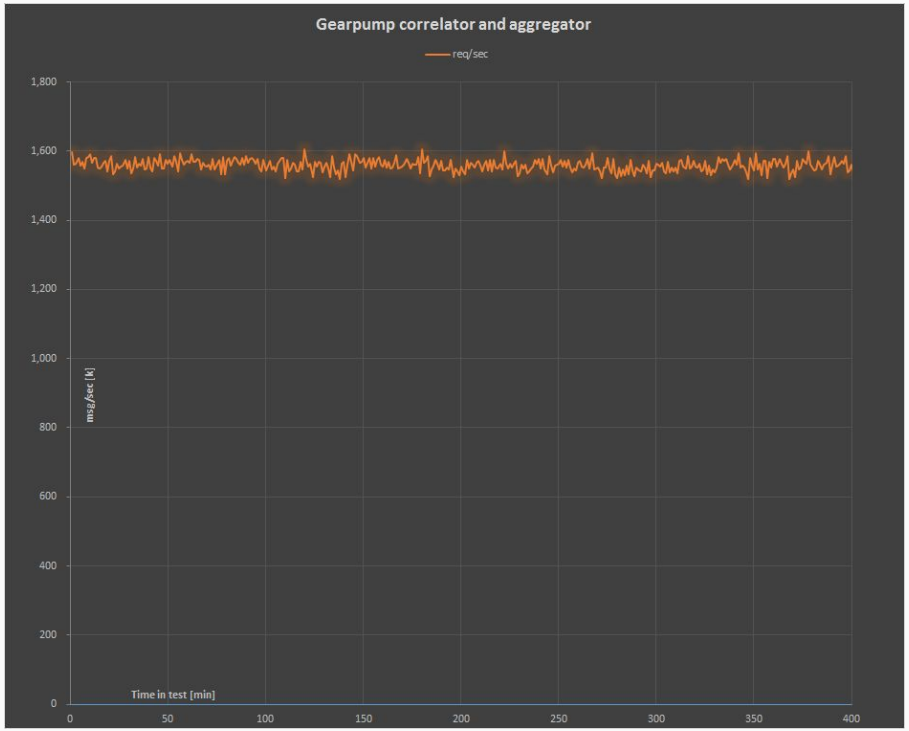
- CPU: Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz
- Memory: 256 Gbytes DDR4
- Storage: 8 SATA SSDs

The results (1) - let's start small: ~700k msg/sec



Initial attempt

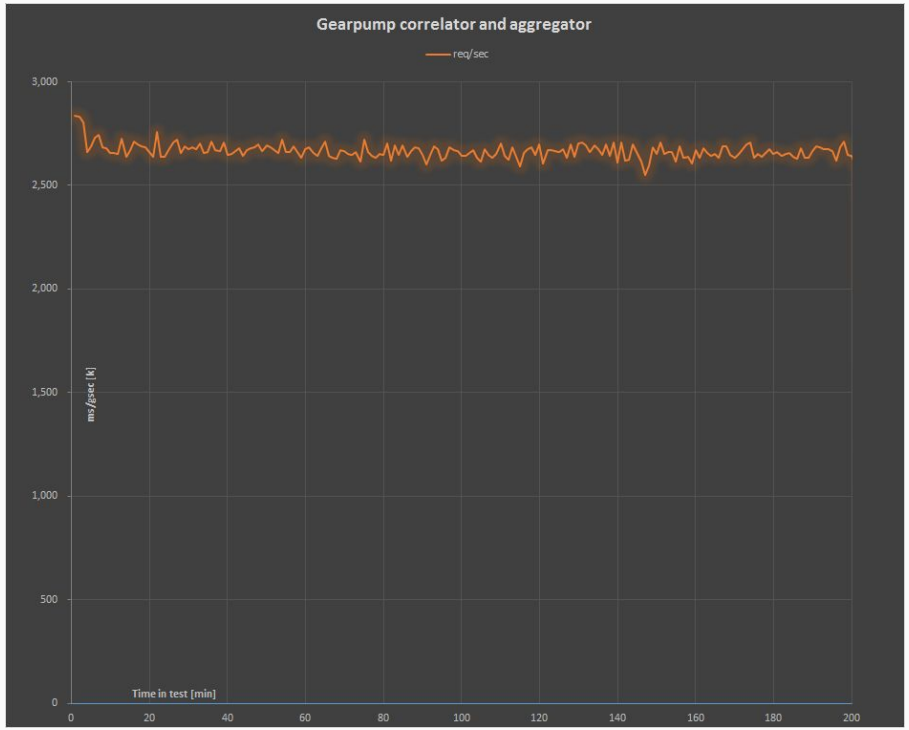
The results (2) - 8 executors: ~1.6M msg/sec



Findings:

- We need to improve Kafka Source - queue size, fetch frequency
- Improve Kafka partitions design for concurrency
- Network throughput may be a bottleneck (1.6M msg/sec * 0.5 k * 8 bit) - compression

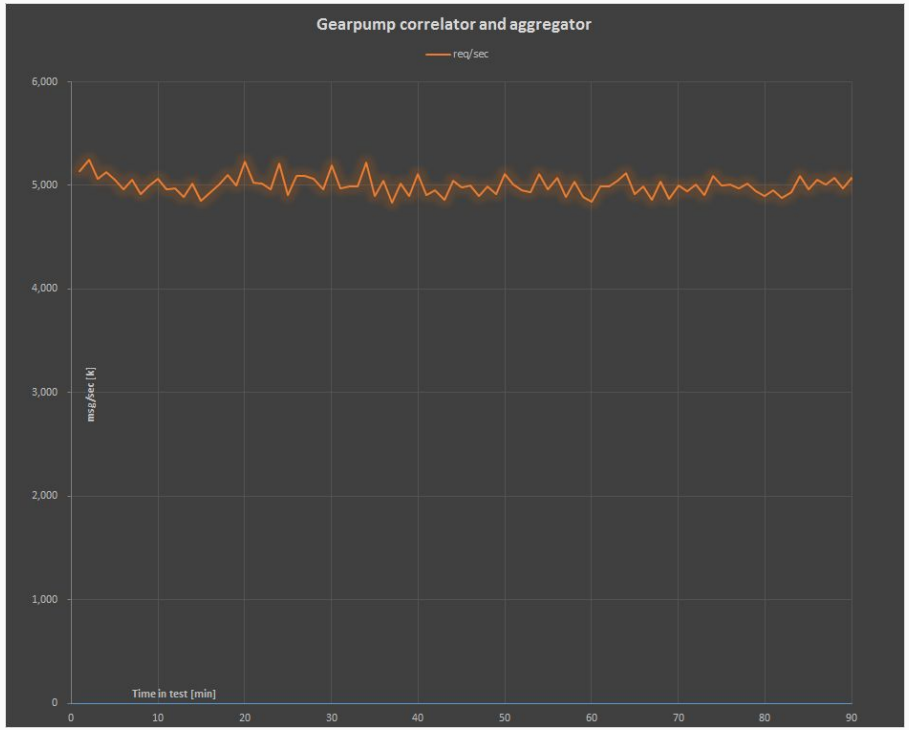
The results (3) - 16 executors: ~2.7M msg/sec



Findings:

- JVM defaults designed for moderate workloads - we need to pump them up
- Message marshalling starts to play significant role in performance - look for better alternatives

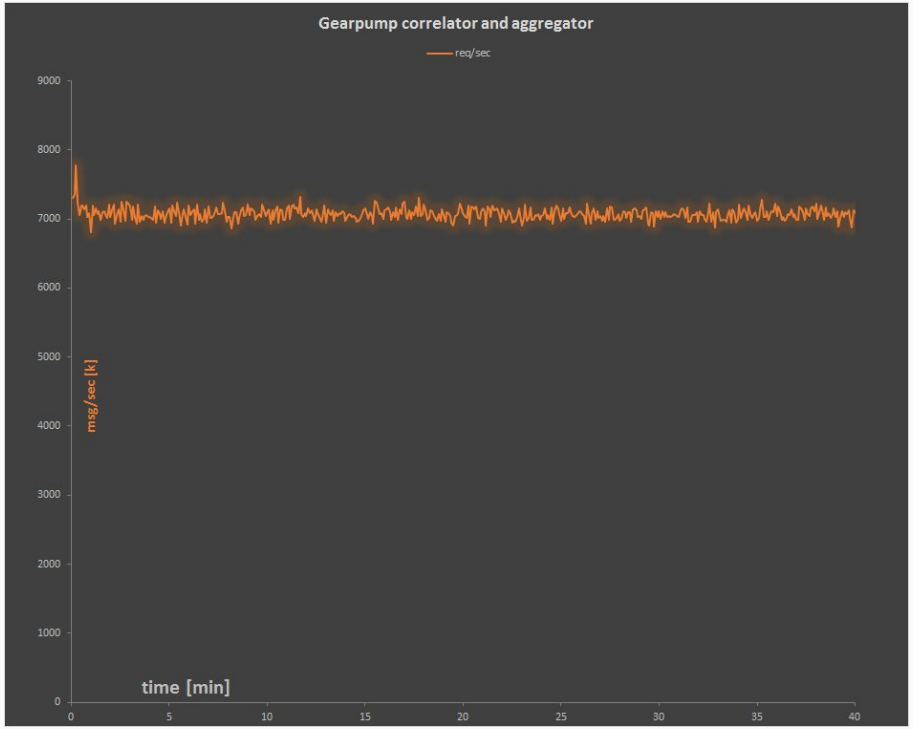
The results (4) - 32 executors: ~5M msg/sec



Findings:

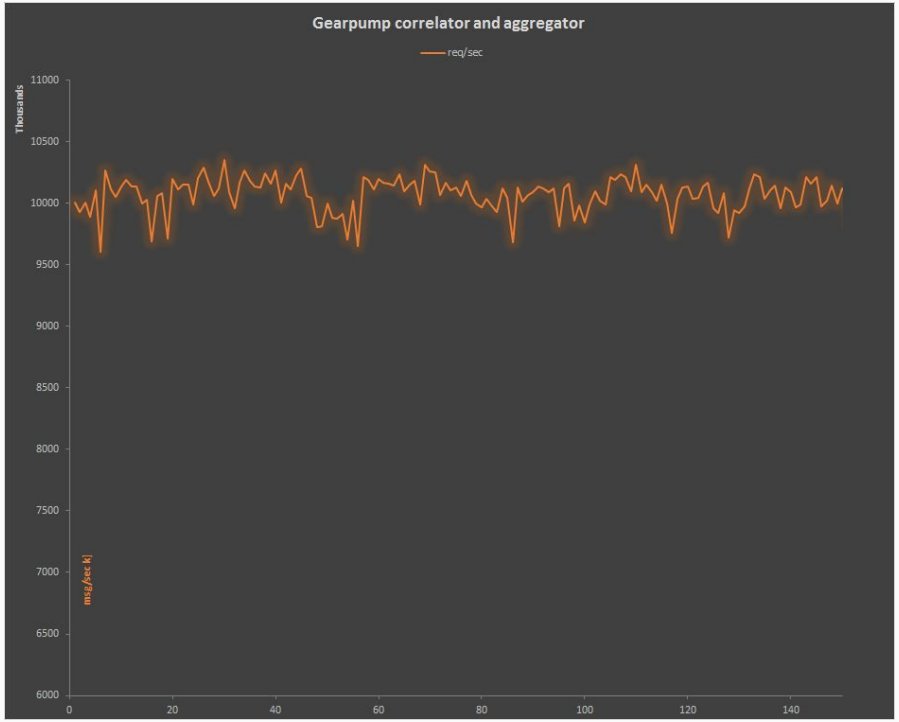
- Backpressure introduced by JVM ⇔ JVM communication - use task fusing

The results (5) - 48 executors: 7.4M msg/sec



Mission accomplished!!!

The results (6) - 64 executors



We can go even further..

The results - summary

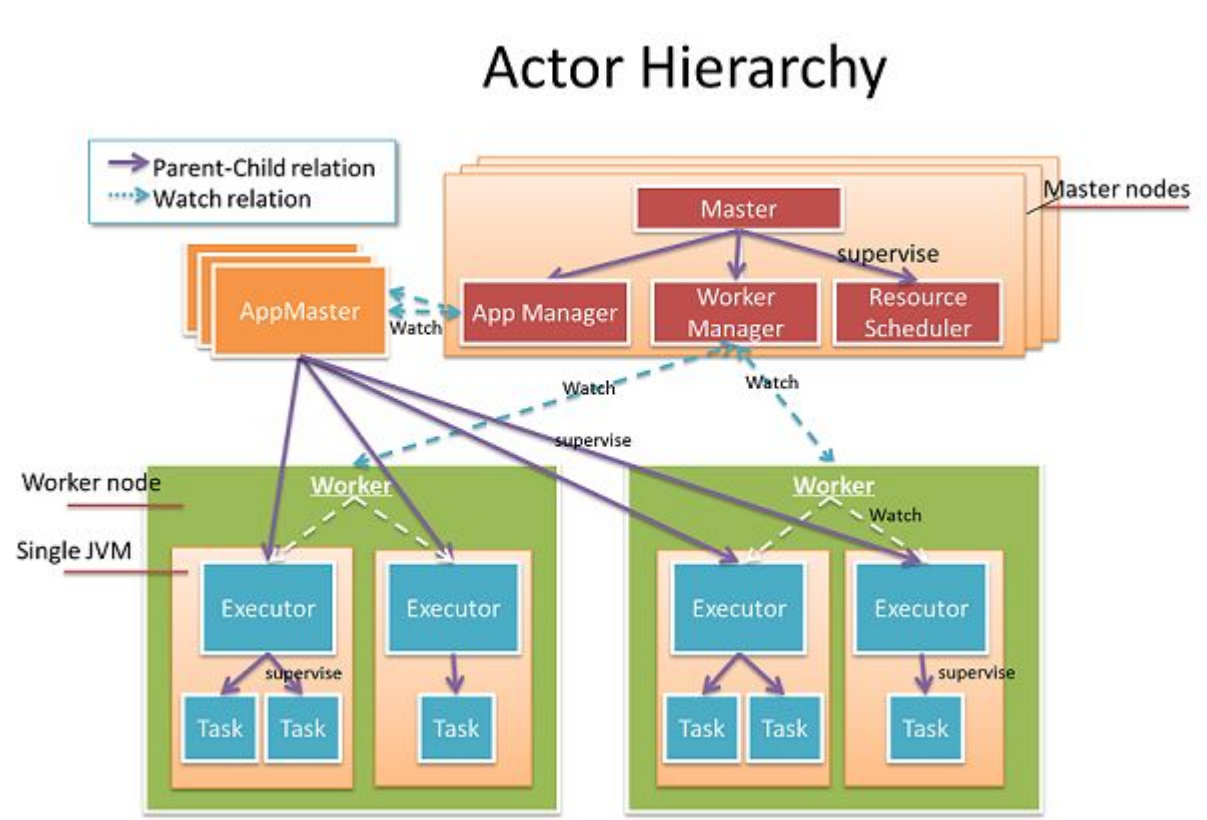
- Great performance numbers on decent hardware
- **Predictable** scalability

Executors number	Req/sec
8	1.6 M
16	2.7 M
32	5 M
48	7,4 M
64	10 M

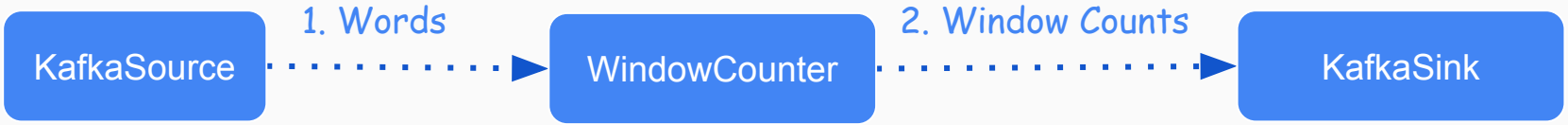
Gearpump features

Gearpump Architecture

- Actor concurrency
- Message passing communication
- error handling and isolation with supervision hierarchy
- Master HA with Akka Cluster



Use case - Windowed word count

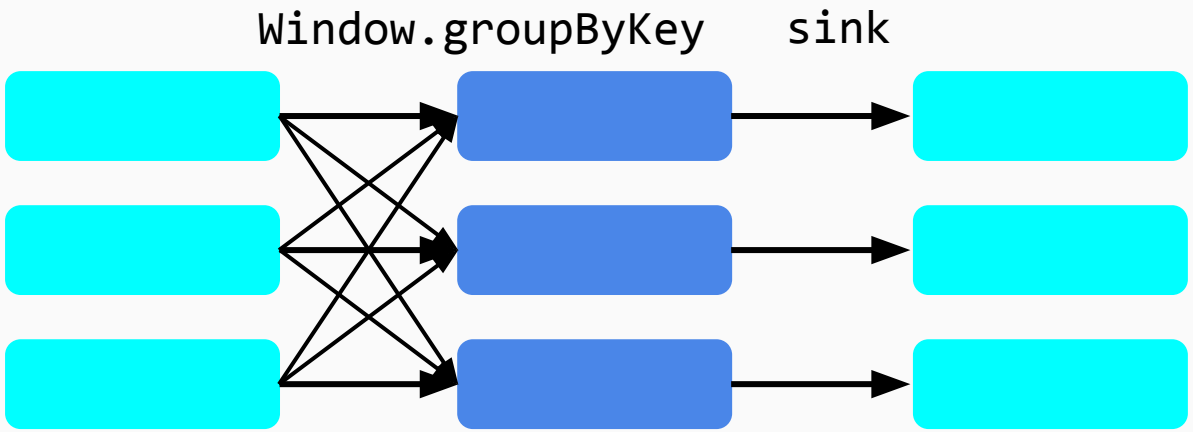


How Gearpump solves the hard parts

- User interface
- Flow control
- Out-of-order processing
- Exactly once
- Dynamic DAG

User interface - DSL

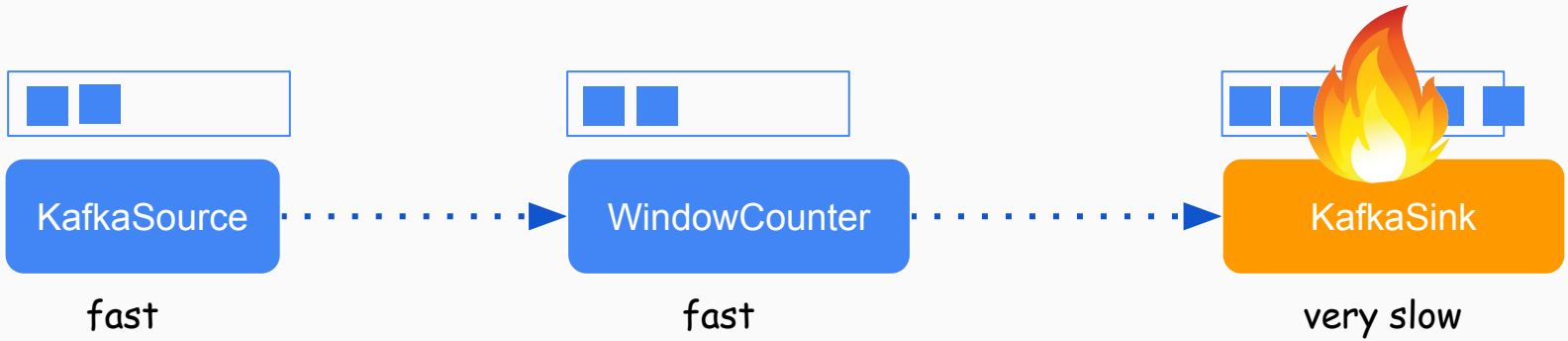
```
val app = StreamApp("dsl", context)
app.source[String](kafkaSource).
  flatMap(line => line.split("[\\s]+")).map((_, 1)).
  window(FixedWindow.apply(Duration.ofMillis(5L))
    .triggering(EventTimeTrigger)).
  // (word, count1), (word, count2) => (word, count1 + count2)
  groupBy(_._1).sum.sink(kafkaSink)
```



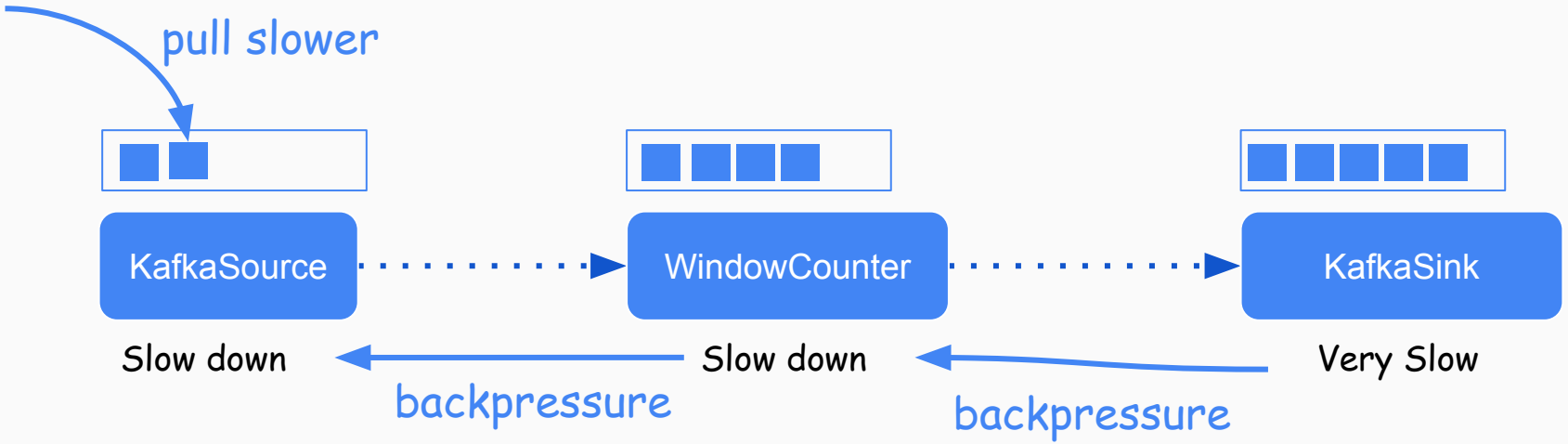
How Gearpump solves the hard parts

- User interface
- Flow control
- Out-of-order processing
- Exactly once
- Dynamic DAG

Without Flow Control - OOM



With Flow Control - Backpressure

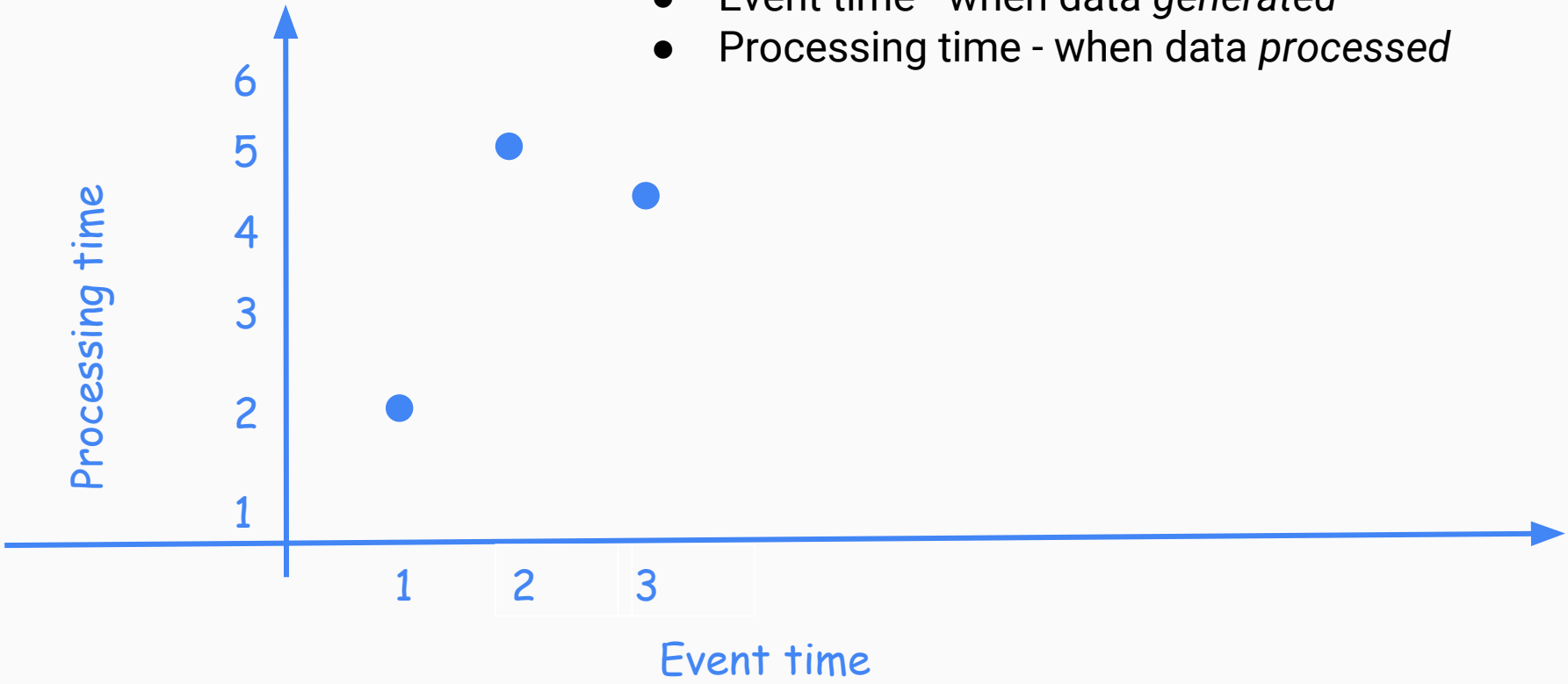


How Gearpump solves the hard parts

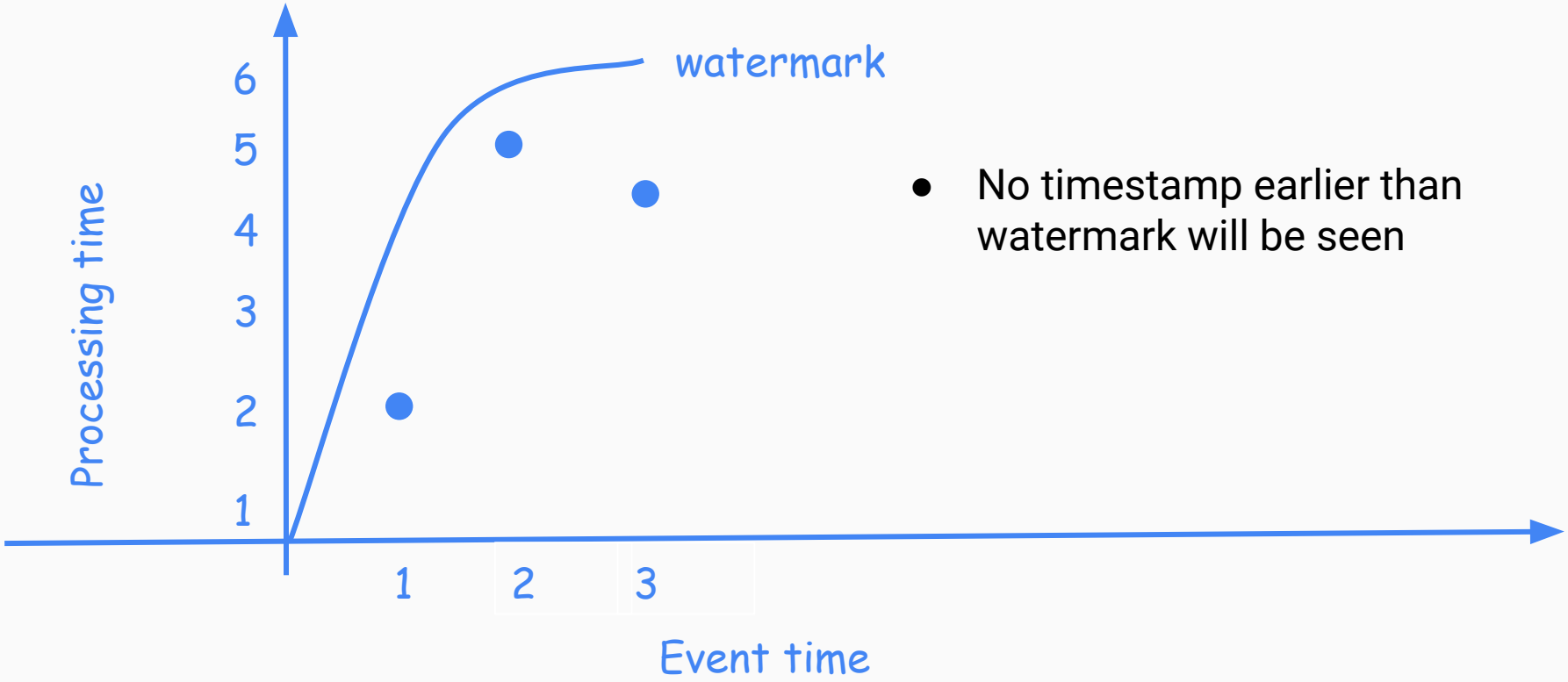
- User Interface
- Flow control
- Out-of-order processing
- Exactly Once
- Dynamic DAG

Out-of-order data

- Event time - when data *generated*
- Processing time - when data *processed*

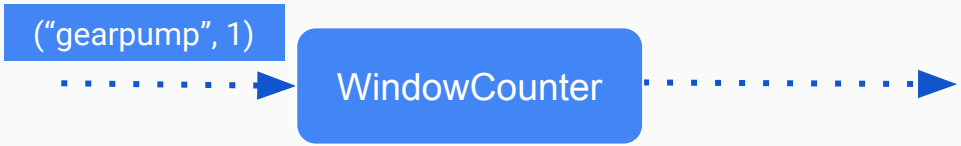


On Watermark[4]



- No timestamp earlier than watermark will be seen

When can window counts be emitted ?

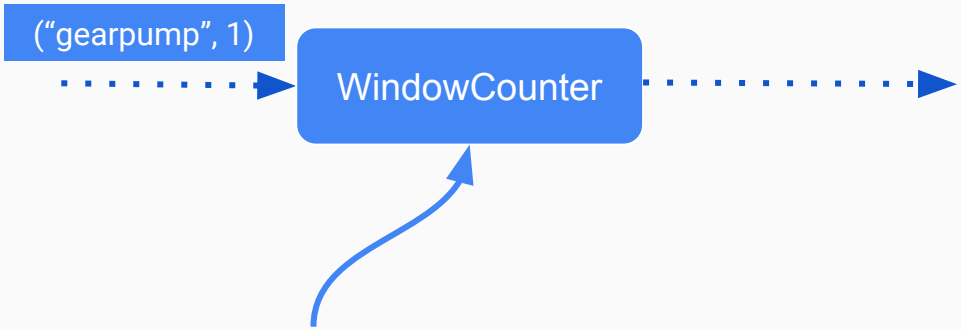


WindowCounter In-memory Table

window	messages
[0, 2)	(“gearpump”, 1)
[2, 4)	(“gearpump”, 3) (“gearpump”, 2)
[4, 6)	(“gearpump”, 5) (“gearpump”, 4)

- No window can be emitted since message as early as time 1 has not arrived

Out-of-order processing with watermark

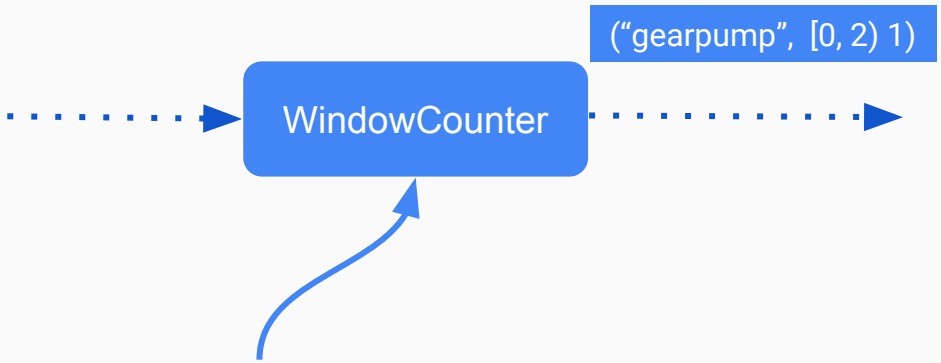


watermark = 0,
No window can be emitted

WindowCounter In-memory Table

window	messages
[0, 2)	(“gearpump”, 1)
[2, 4)	(“gearpump”, 3) (“gearpump”, 2)
[4, 6)	(“gearpump”, 5) (“gearpump”, 4)

Out-of-order processing with watermark

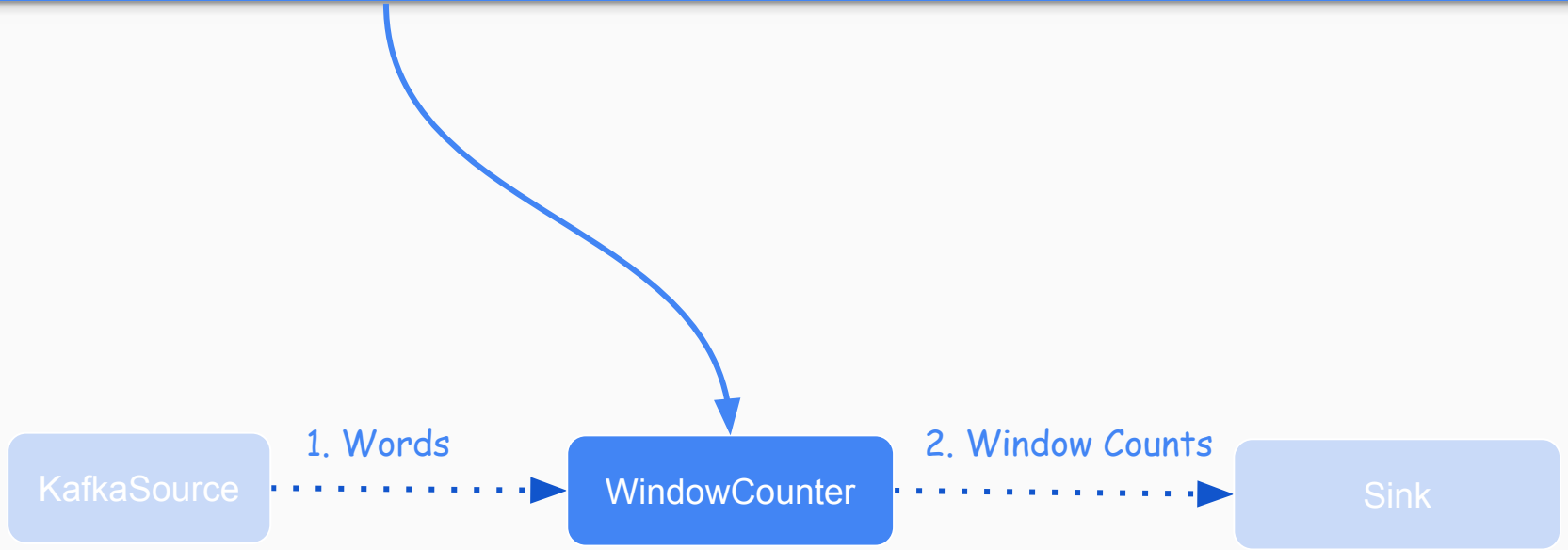


WindowCounter In-memory Table

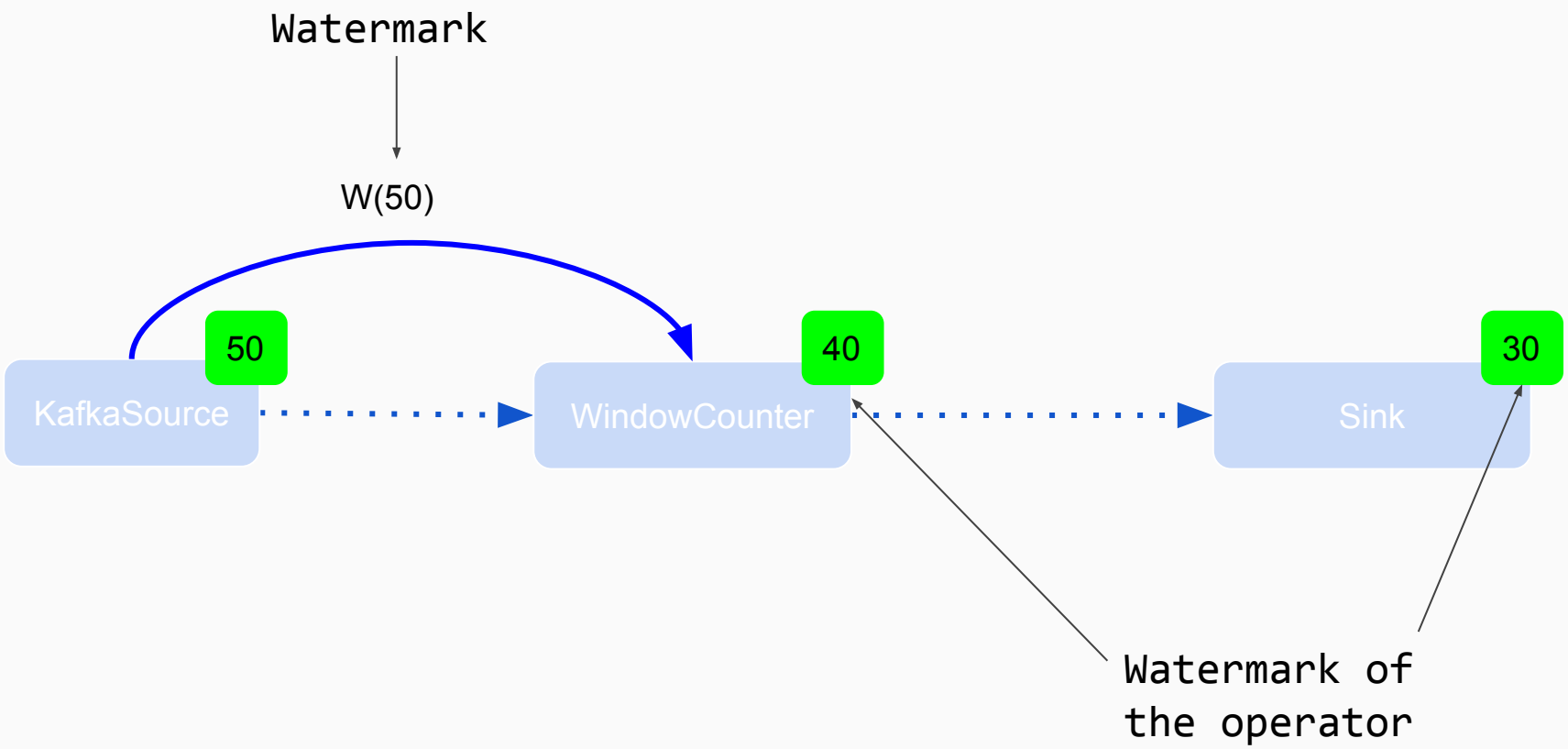
window	messages
[2, 4)	("gearpump", 3) ("gearpump", 2)
[4, 6)	("gearpump", 5) ("gearpump", 4)

watermark = 2,
Window [0, 2) can be emitted

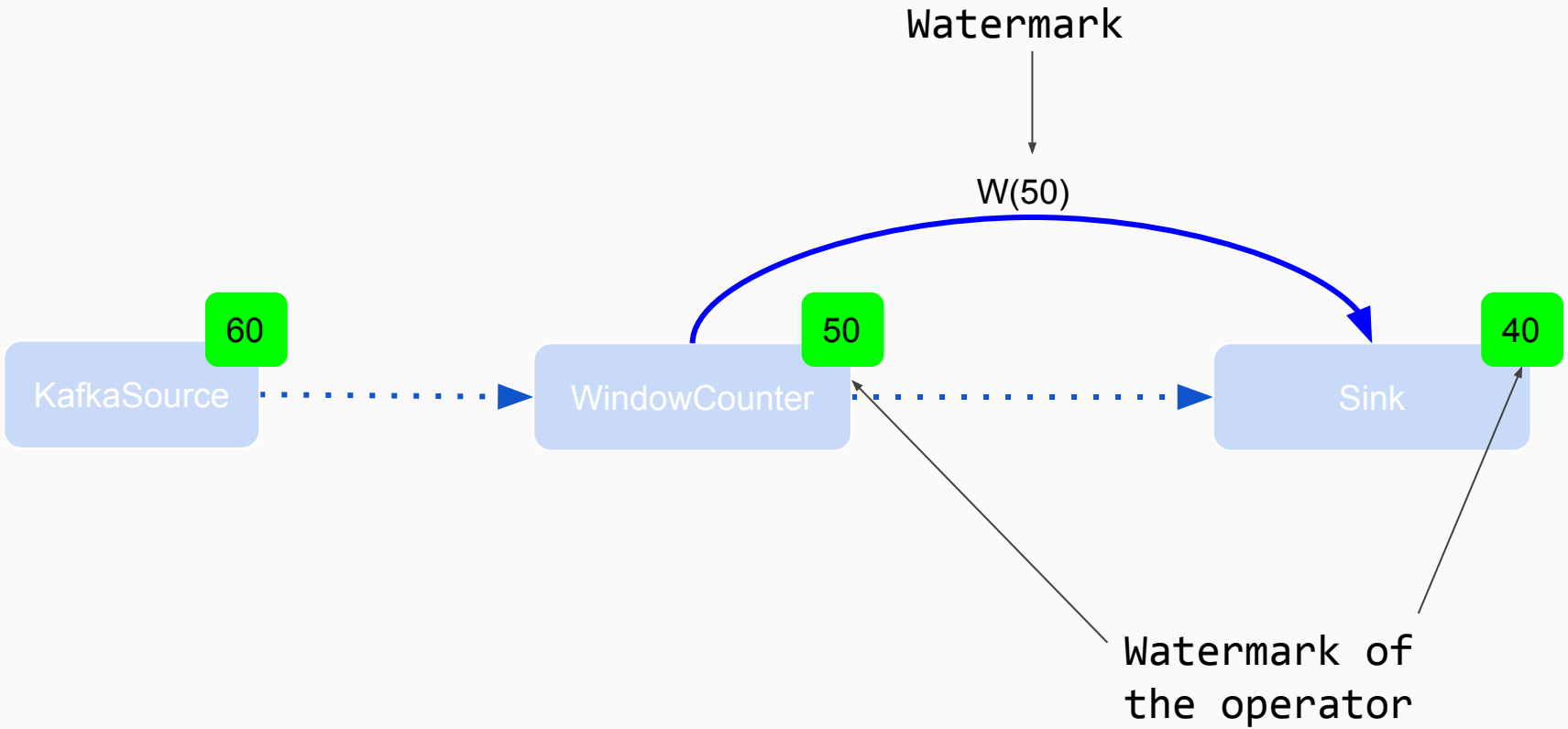
How to get watermark ?



From upstream



From upstream



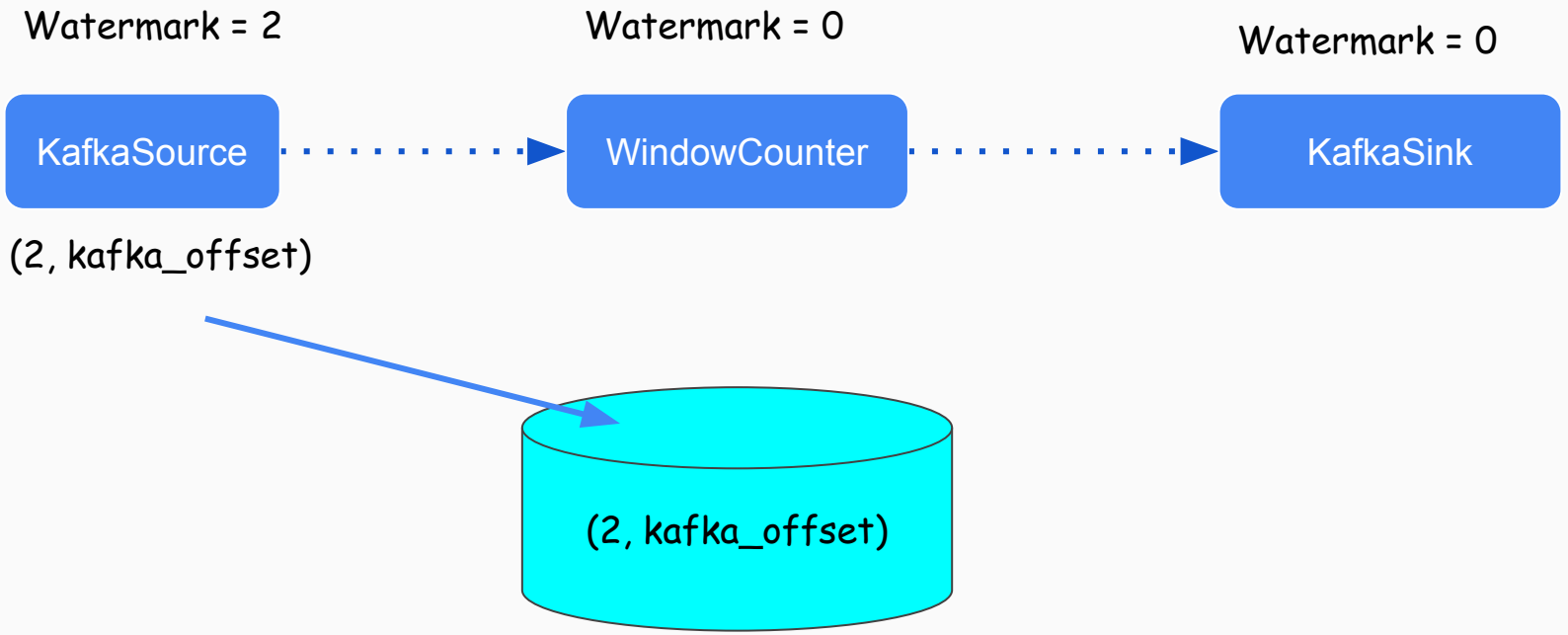
More on Watermark

- Source watermark defined by user
- Usually heuristic based
- Users decide whether to drop data arriving after watermark

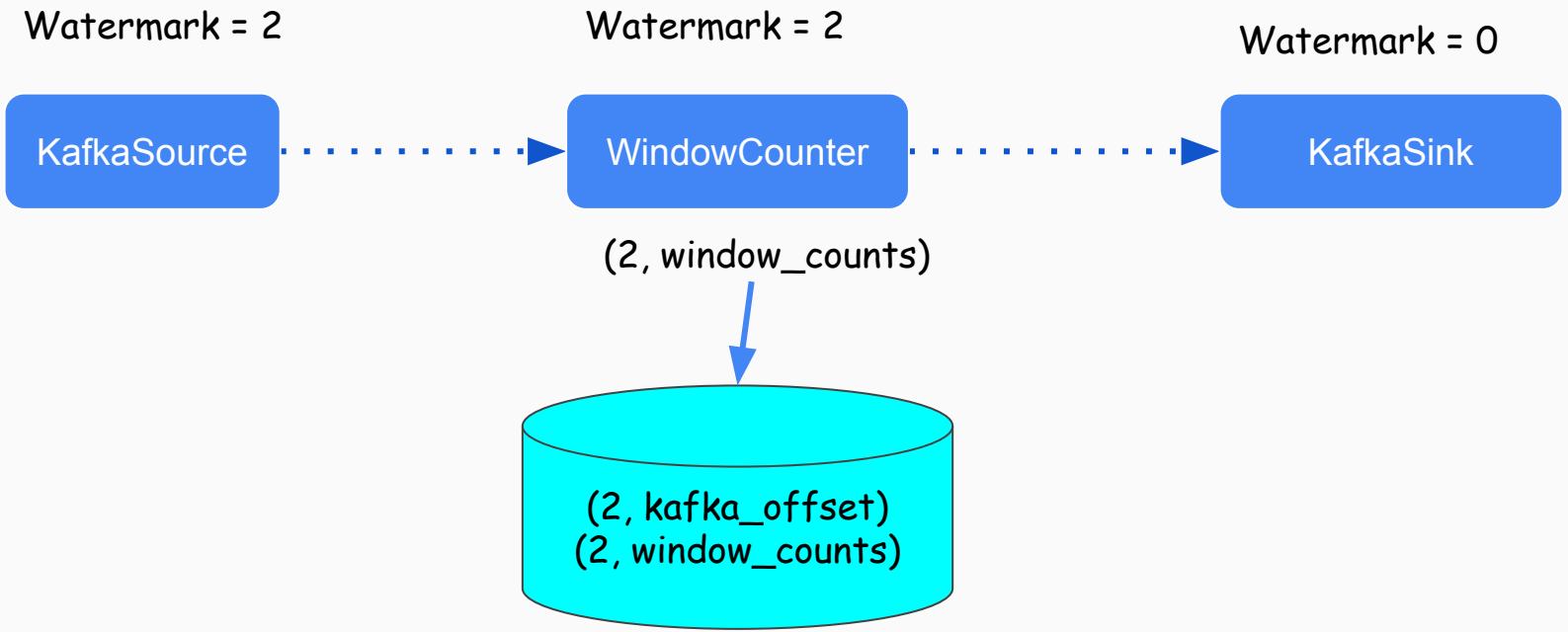
How Gearpump solves the hard parts

- User Interface
- Flow control
- Out-of-order processing
- Exactly once
- Dynamic DAG

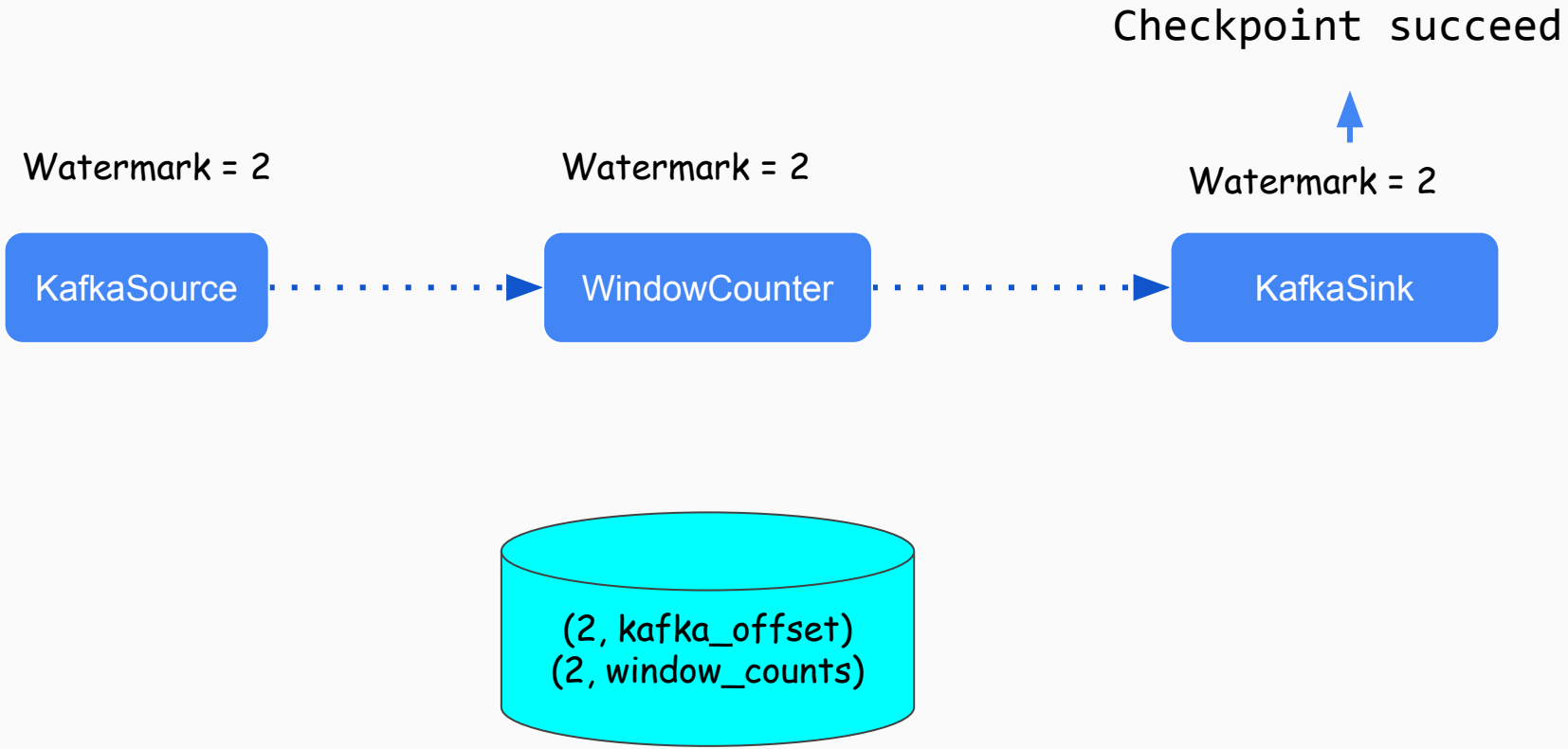
Exactly Once with asynchronous checkpointing



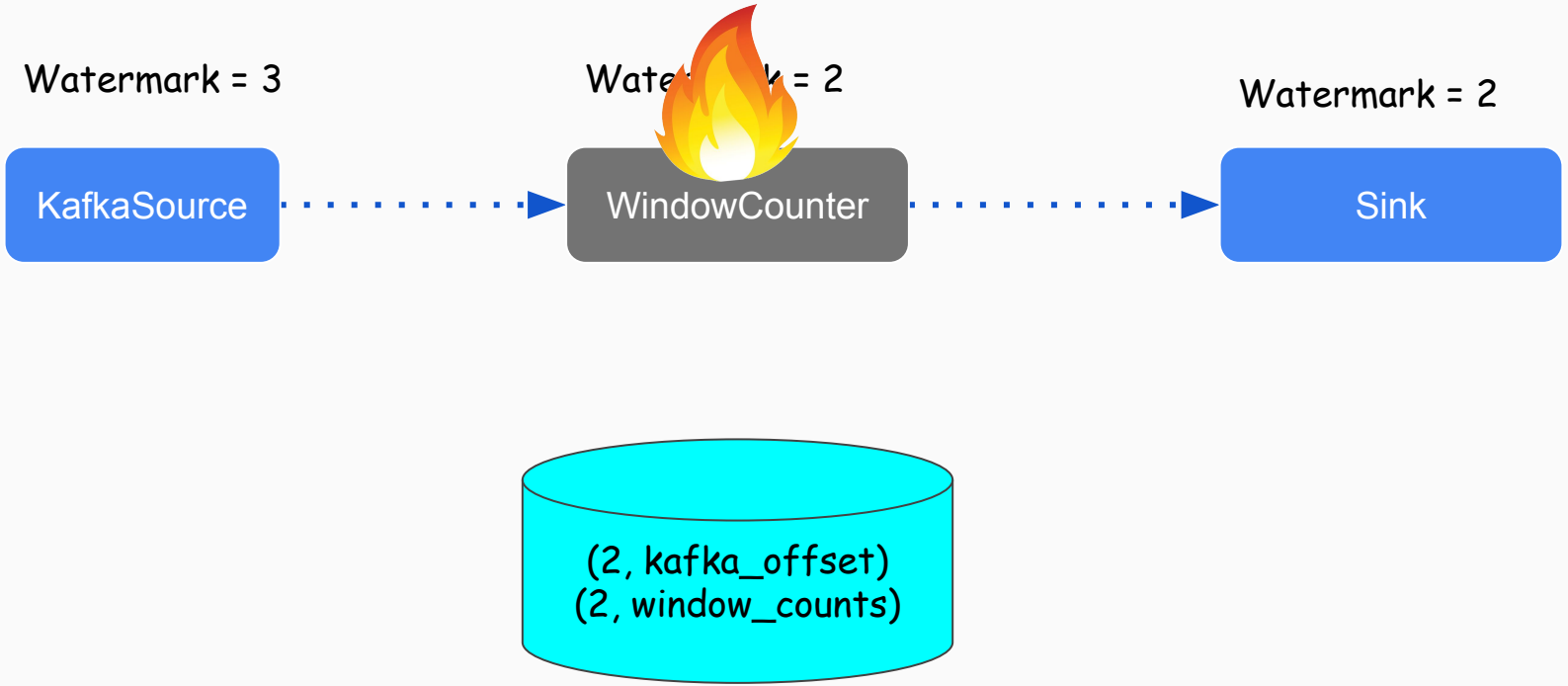
Exactly Once with asynchronous checkpointing



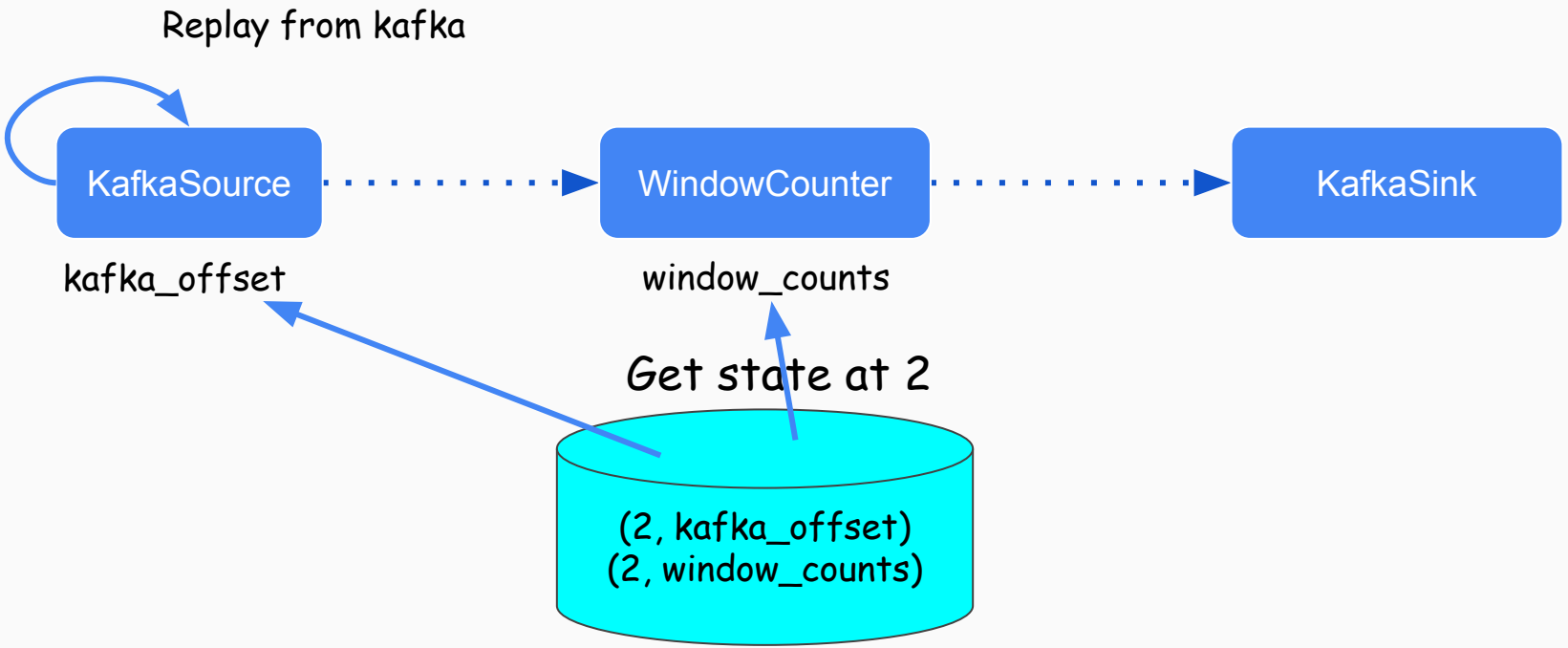
Exactly Once with asynchronous checkpointing



Crash


Watermark = 3
Watermark = 2
Watermark = 2
KafkaSource
WindowCounter
Sink
(2, kafka_offset)
(2, window_counts)

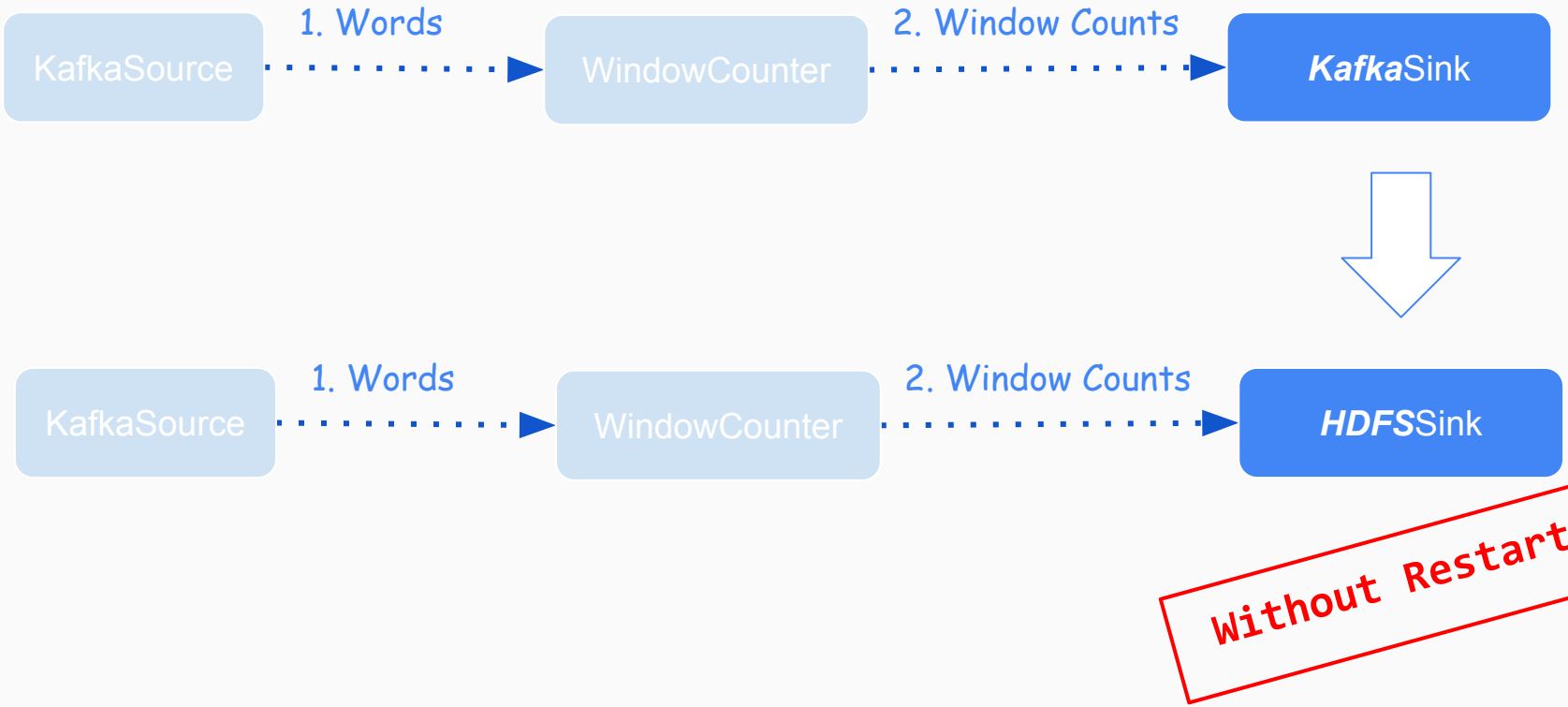
Recover to latest checkpoint at 2



How Gearpump solves the hard parts

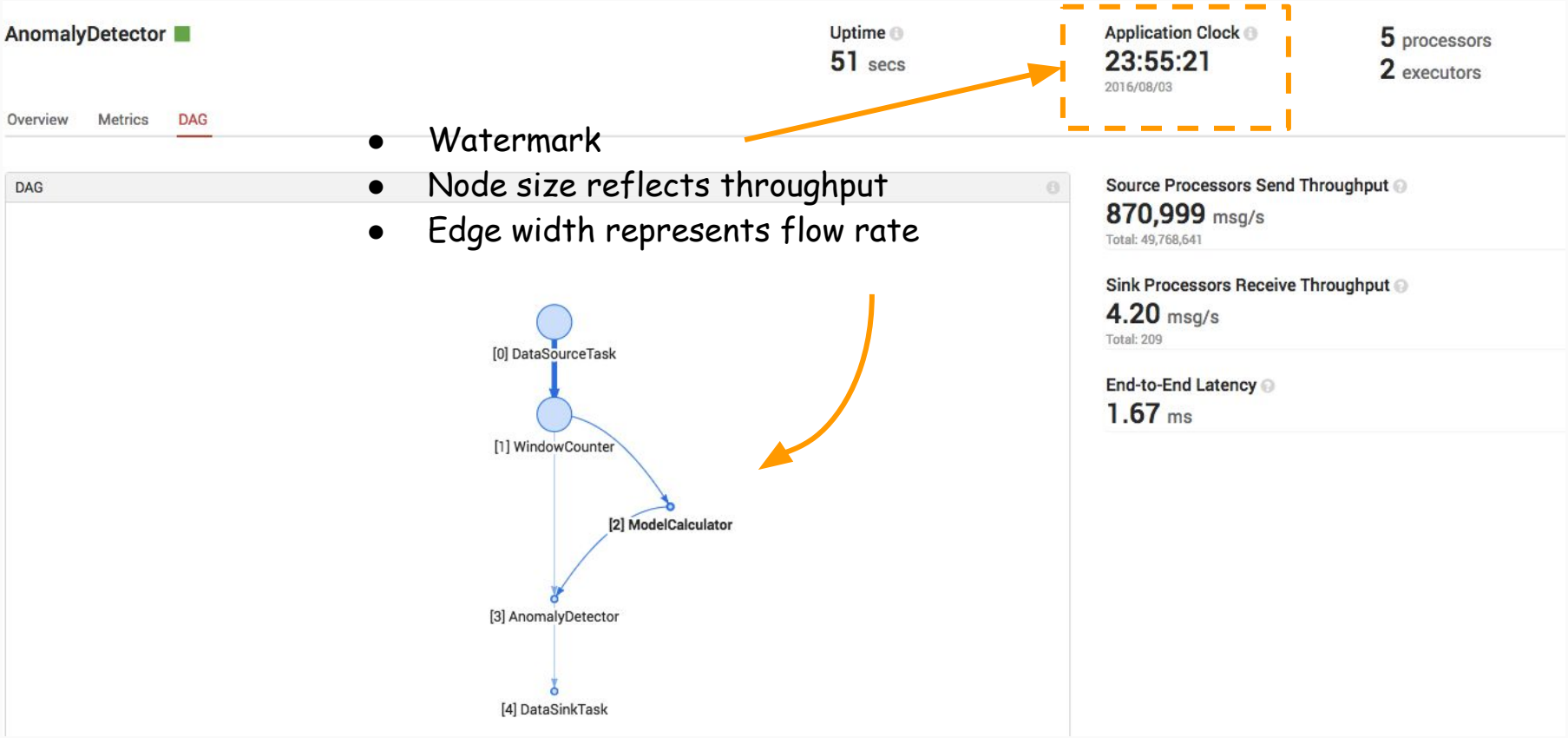
- User Interface
- Flow control
- Out-of-order processing
- Exactly Once
- Dynamic DAG

Update the DAG on-the-fly



Advanced features

DAG Visualization



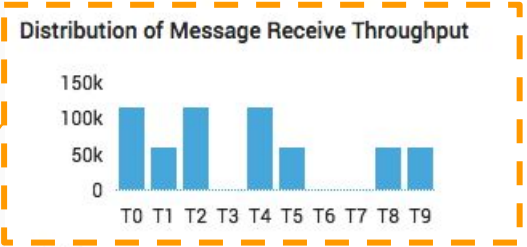
- Watermark
- Node size reflects throughput
- Edge width represents flow rate

DAG Visualization

- Data skew analysis

Tasks

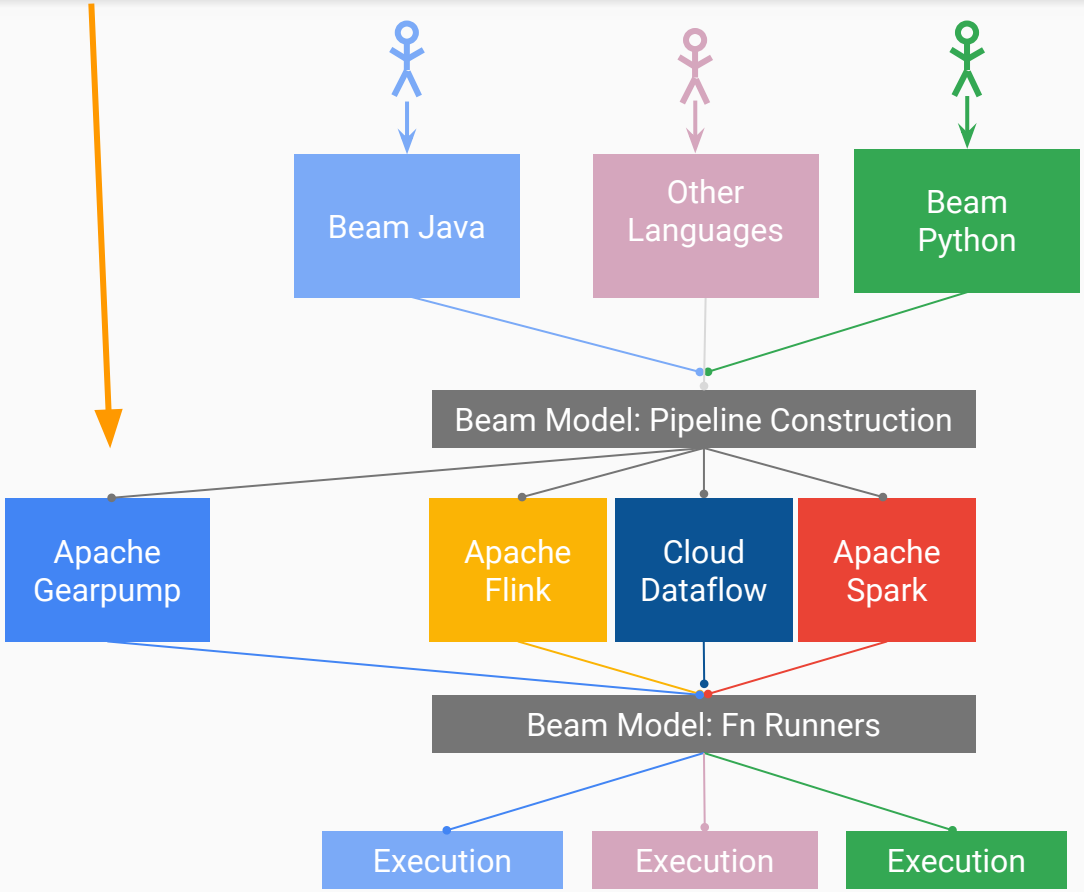
Message Receive Throughput



Task	Total Messages msg	Mean Rate msg/s	MA 1m msg/s
T0	7,837,232	162,695	112,886
T1	3,915,279	81,277	56,387
T2	7,832,785	162,600	112,825
T3	0	0	0
T4	7,833,889	162,434	112,403
T5	3,917,413	81,223	56,196
T6	0	0	0
T7	0	0	0
T8	3,914,714	81,154	56,106
T9	3,914,623	81,118	56,002

Showing 1-10 of 10 records

Apache Beam[6] Gearpump Runner



What’s next for Gearpump

Experimental features

- Web UI Authorization / OAuth2 Authentication
- CGroup Resource Isolation
- Binary Storm compatibility
- Akka Streams integration (Gearpump Materializer)

Summary

- Gearpump is good at streaming infinite out-of-order data and guarantees correctness
- Gearpump helps users to easily program streaming applications, get runtime information and update dynamically

References

1. gearpump.apache.org
2. akka.io
3. http://www.slideshare.net/SeanZhong/strata-singapore-gearpumpreal-time-dagprocessing-with-akka-at-scale
4. https://www.cs.cmu.edu/~pavlo/courses/fall2013/static/papers/p734-akidau.pdf
5. https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at
6. Apache Beam [project overview]
7. www.trustedanalytics.org - learn more about TAP

Get involved

Our home: http://gearpump.apache.org

Contribute code: https://github.com/apache/incubator-gearpump

Report issues: https://issues.apache.org/jira/browse/GEARPUMP

The team: Kam Kasravi, Manu Zhang, Huafeng Wang, Weihua Jiang, Sean Zhong, Karol Brejna, Stanley Xu, ..., **YOU**?

Learn More About TAP

www.trustedanalytics.org

Engage in Community events

Meetups, workshops, & webinars

http://trustedanalytics.org/#resources