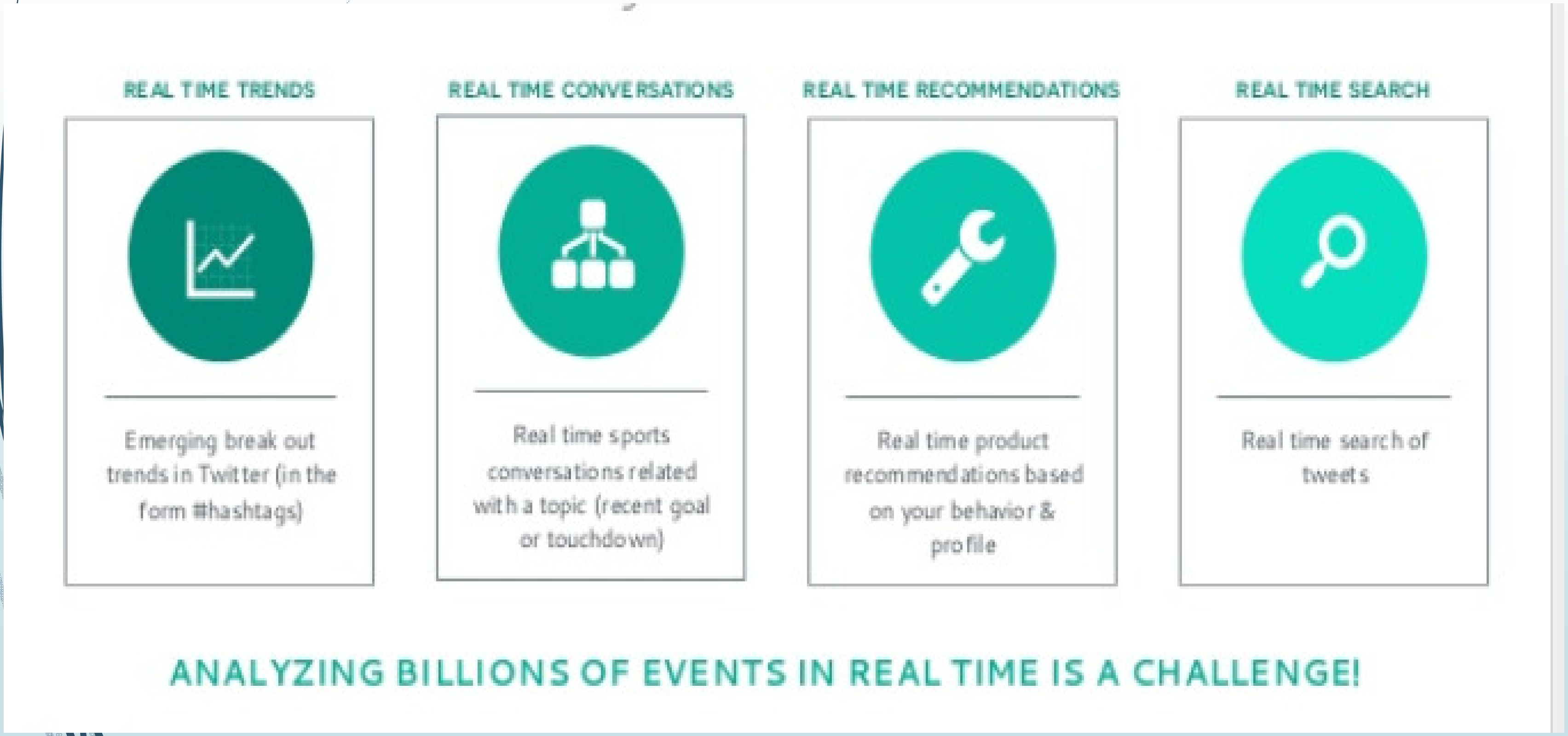
Twitter Heron: Stream Processing at Scale

Saiyam Kohli

CIS 611 Research Paper Presentation
-Sun Sunnie Chung

December 8th, 2016

TWITTER IS A REAL TIME



REAL TIME TRENDS	REAL TIME CONVERSATIONS	REAL TIME RECOMMENDATIONS	REAL TIME SEARCH
Emerging break out trends in Twitter (in the form #hashtags)	Real time sports conversations related with a topic (recent goal or touchdown)	Real time product recommendations based on your behavior & profile	Real time search of tweets

ANALYZING BILLIONS OF EVENTS IN REAL TIME IS A CHALLENGE!

ABSTRACT

- We process billions of events on Twitter every day. Analyzing these events in real time presents a big challenge. Storm has been the main data streaming engine for a long time. But in the recent years twitter data has been exponentially increased which is difficult to manage with storm.
- So to cover the loop-holes of the Storm. Twitter launch **Heron** — a real-time analytics platform that is fully API-compatible with Storm. The main goal to to increase performance predictability, improve developer productivity and ease manageability.
- At Twitter, Heron is used as our primary streaming system, running hundreds of development and production topologies. Since Heron is efficient in terms of resource usage, after migrating all Twitter's topologies to it we've seen an overall 3x reduction in hardware, causing a significant improvement in our infrastructure efficiency

INTRODUCTION(STORM ISSUES)

- . Twitter, like many other organizations, relies heavily on real-time streaming. For example, real-time streaming is used to compute the real-time active user counts (RTAC), and to measure the real-time engagement of users to tweets and advertisements. Storm at our current scale was becoming increasingly challenging due to issues related to scalability, debug-ability, manageability, and efficient sharing of cluster resources with other data services.
- In addition, Storm needs dedicated cluster resources, which requires special hardware allocation to run Storm topologies. This approach leads to inefficiencies in using precious cluster resources, and also limits the ability to scale on demand.
- With Storm, provisioning a new production topology requires manual isolation of machines, and conversely, when a topology is no longer needed, the machines allocated to serve that topology now have to be decommissioned

LACK OF BACKPRESSURE OF STORM

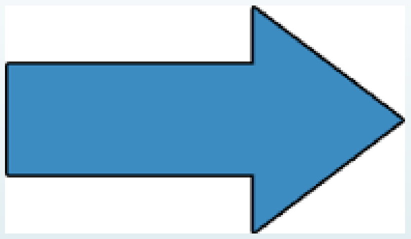
- Storm has no backpressure mechanism. If the receiver component is unable to handle incoming data/tuples, then the sender simply drops tuples. This is a fail-fast mechanism, and a simple strategy, but it has the following disadvantages:
- If acknowledgements are disabled, this mechanism will result in unbounded tuple drops, making it hard to get visibility about these drops.
- Work done by upstream components is lost.
- System behavior becomes less predictable.

EFFICIENCY OF STORM

Thee most common causes for reduced efficiency process are:

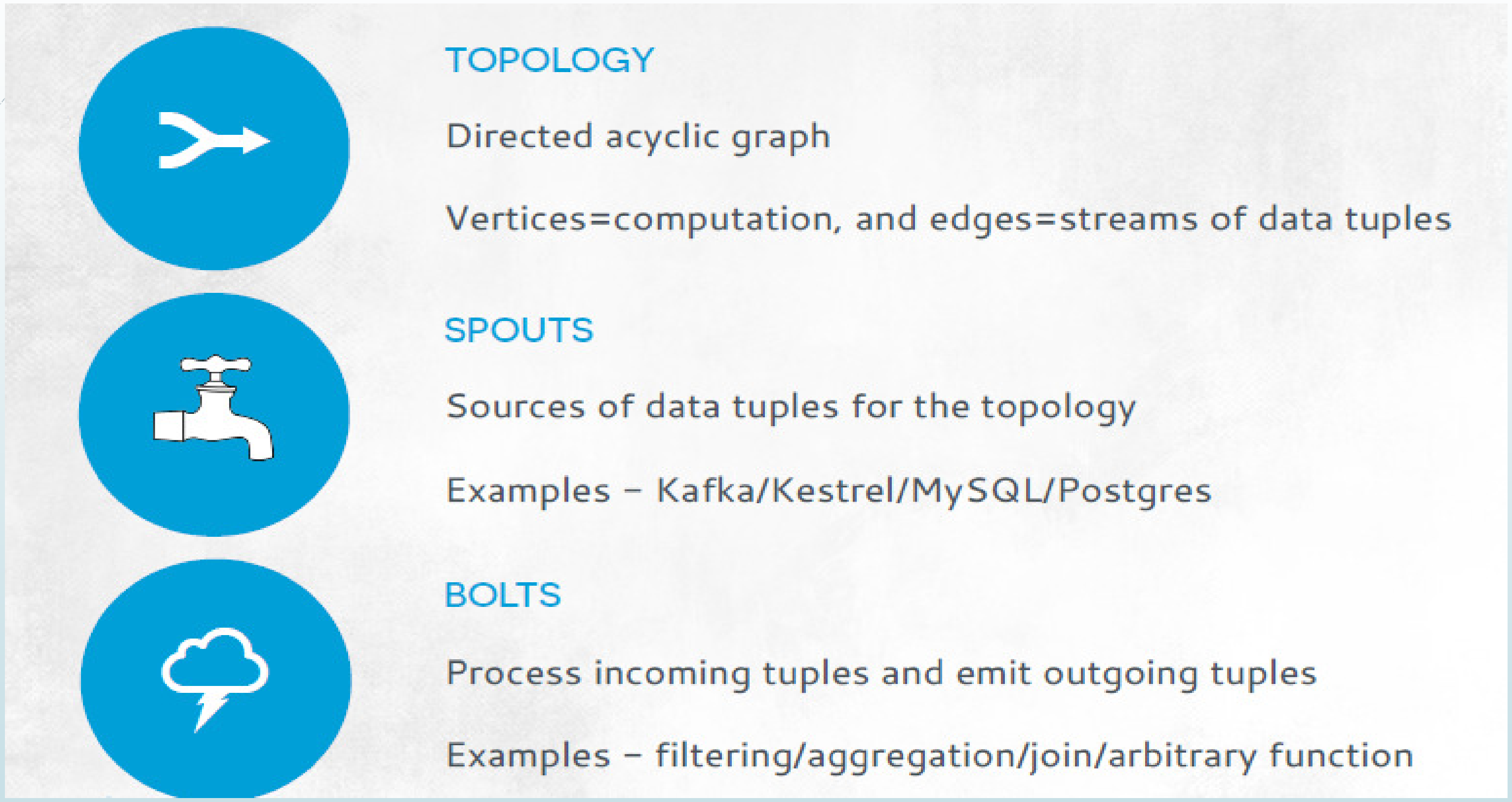
- **Suboptimal replays**– A tuple failure anywhere in the tuple tree leads to failure of the entire tuple tree. This effect is more pronounced with high fan-out topologies where the topology is not doing any useful work, but is simply replaying the tuples.
- **Long Garbage Collection cycles**– Topologies consuming large amount of RAM for a worker encounter garbage collection (GC) cycles greater than a minute, resulting in high latencies and high tuple failure rates.
- **Queue contention**– In some cases, there is a lot of contention at the transfer queues, especially when a worker runs several executors.

SOLUTION: Switch to HERON



HERON

HERON TERMINOLOGY



TOPOLOGY

Directed acyclic graph

Vertices=computation, and edges=streams of data tuples

SPOUTS

Sources of data tuples for the topology

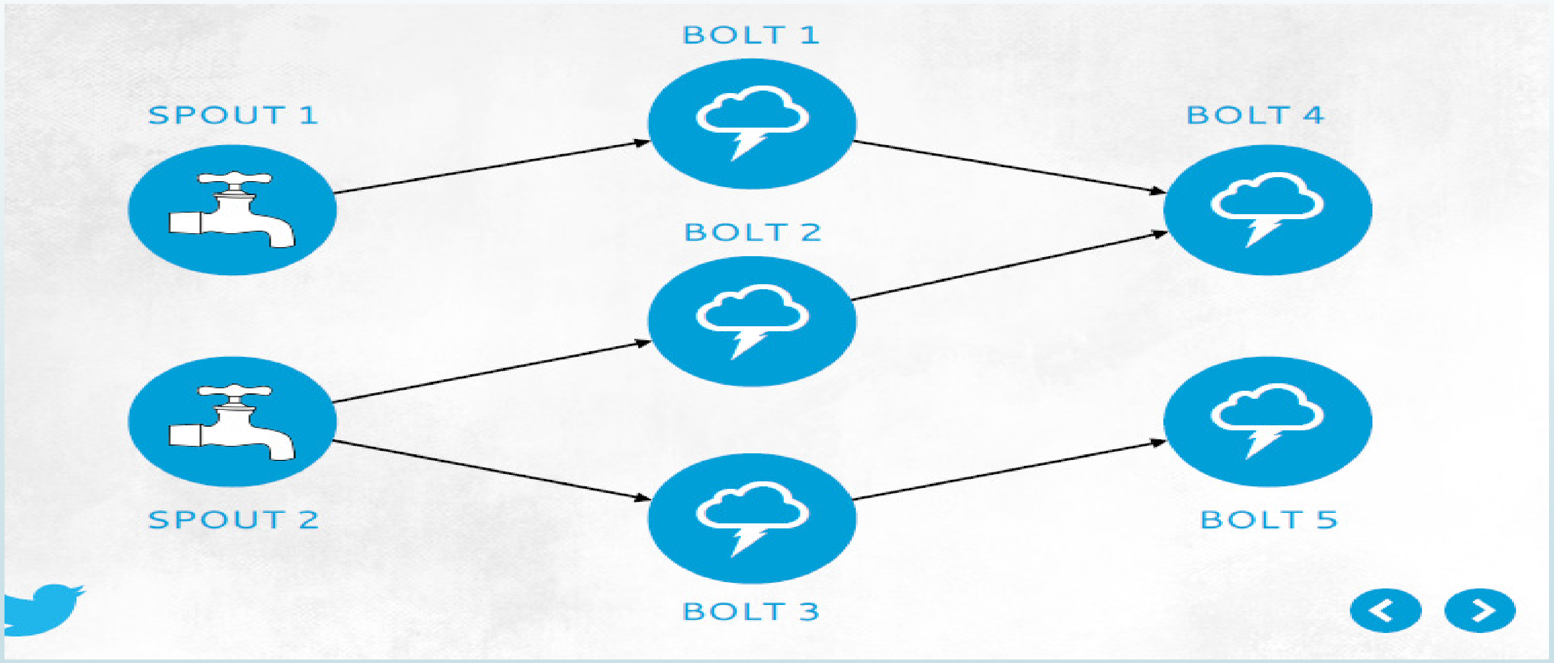
Examples – Kafka/Kestrel/MySQL/Postgres

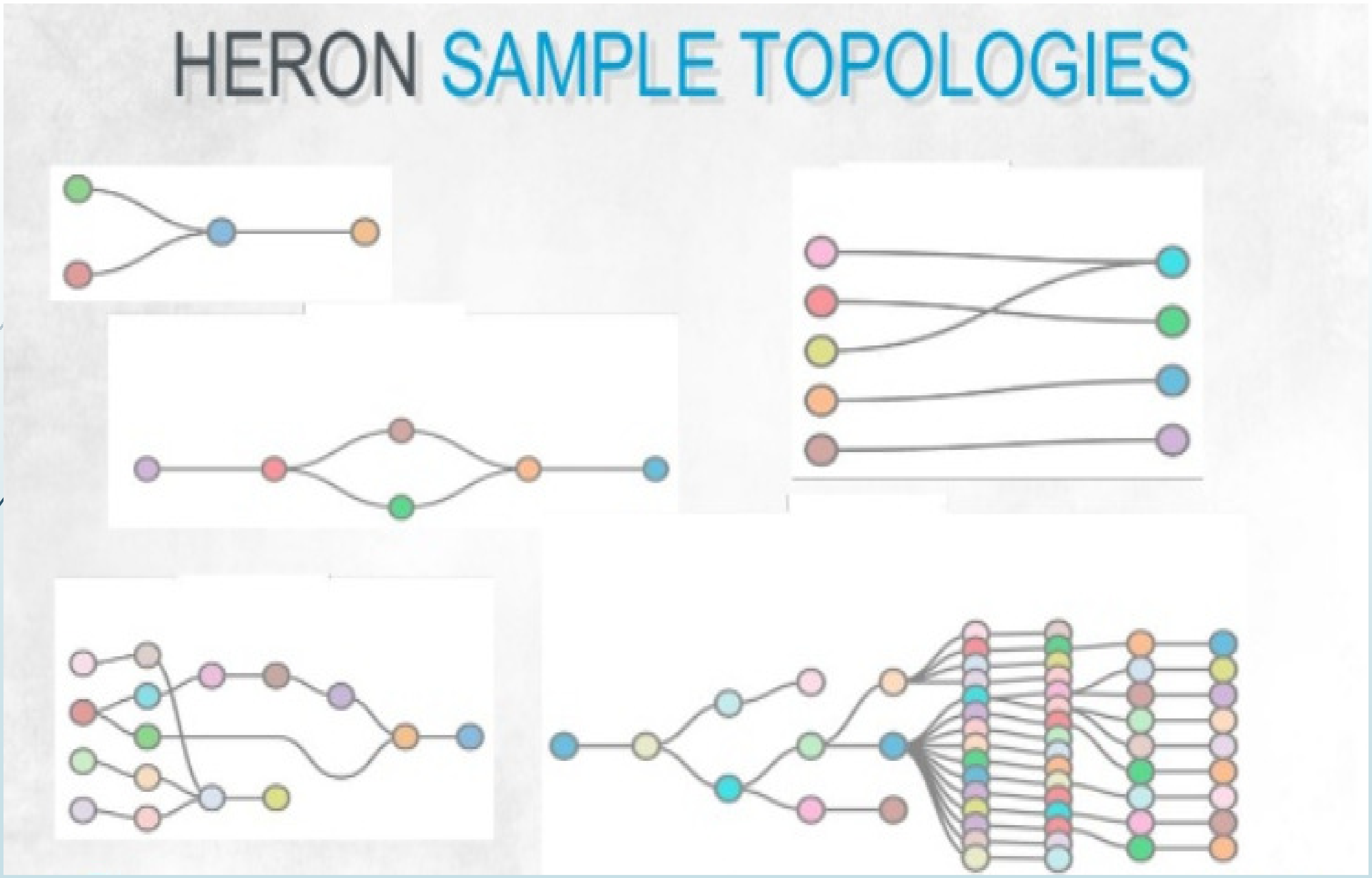
BOLTS

Process incoming tuples and emit outgoing tuples

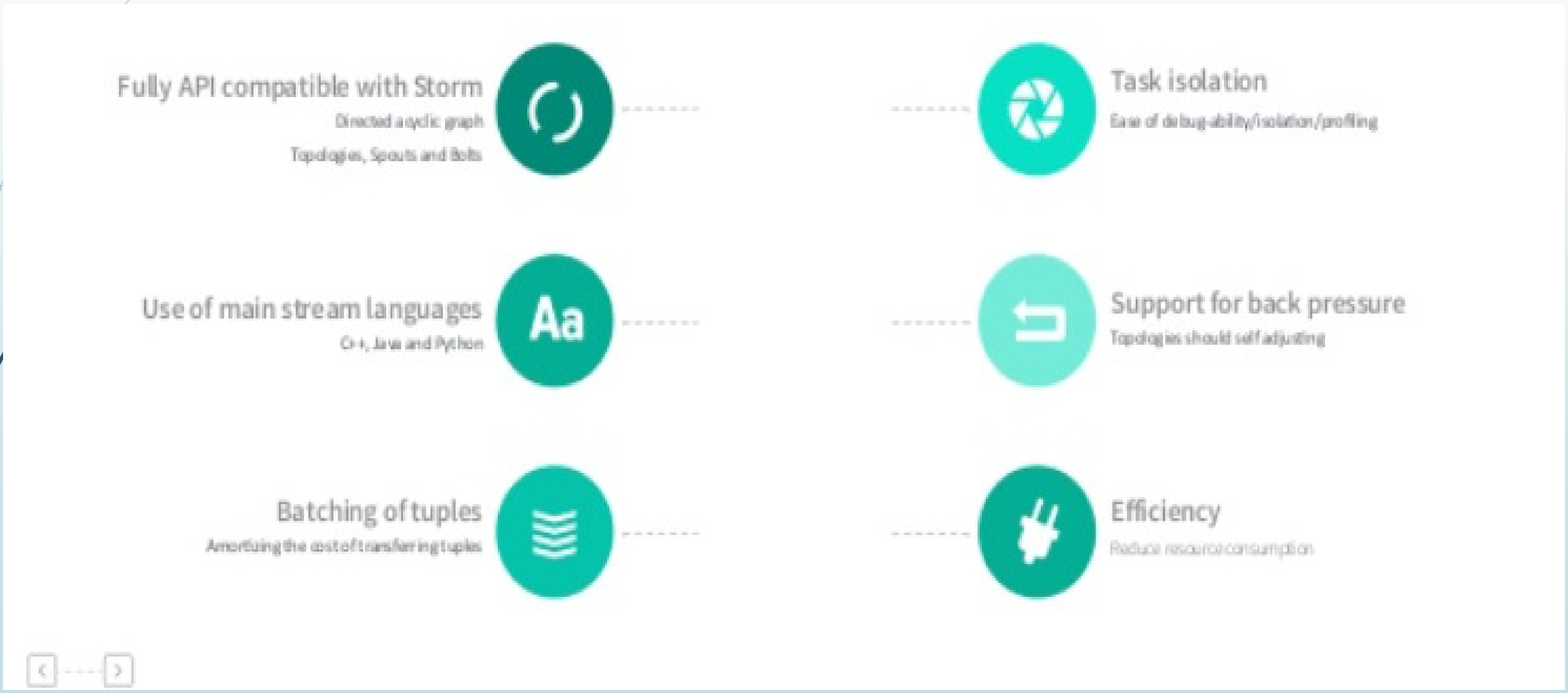
Examples – filtering/aggregation/join/arbitrary function

HERON TOPOLOGY




HERON SAMPLE TOPOLOGIES

GOAL OF HERON



Fully API compatible with Storm
Directed acyclic graph
Topologies, Spouts and Bolts

Use of main stream languages
C++, Java and Python

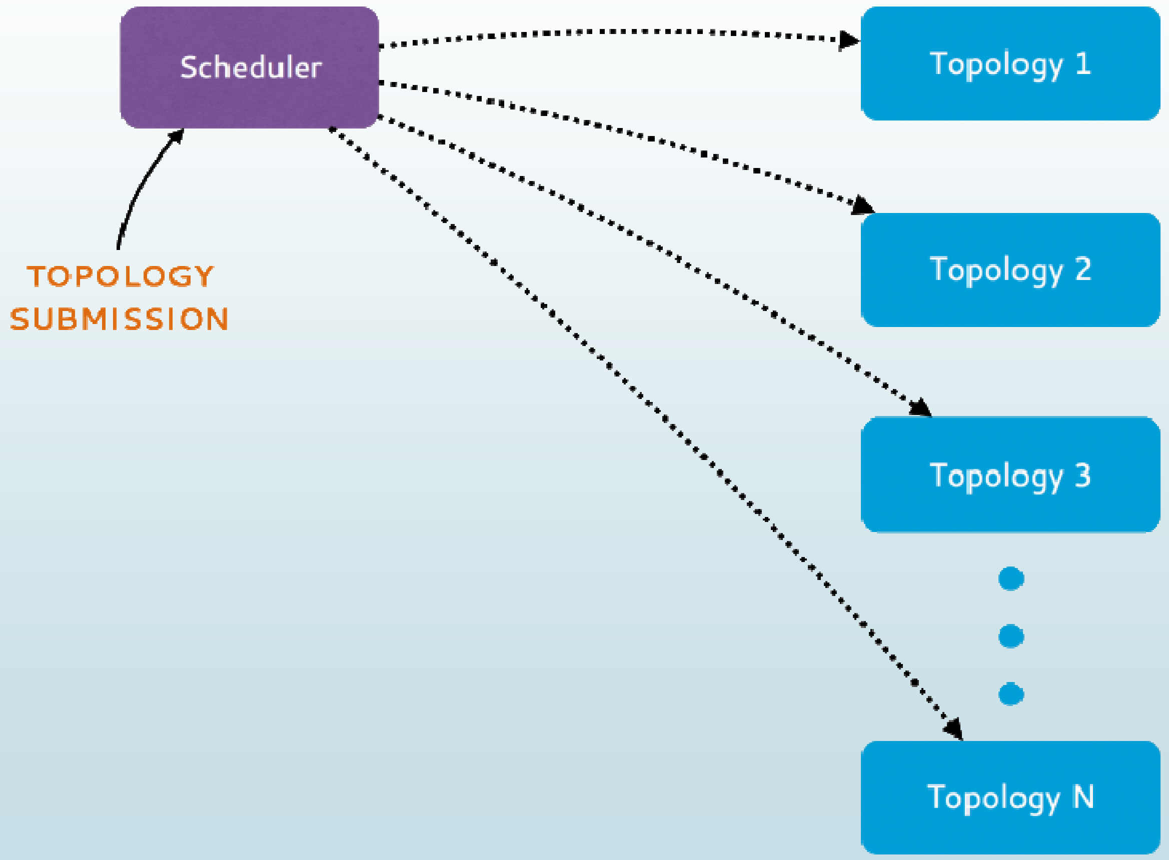
Batching of tuples
Amortizing the cost of transferring tuples

Task isolation
Ease of debug-ability/isolation/profiling

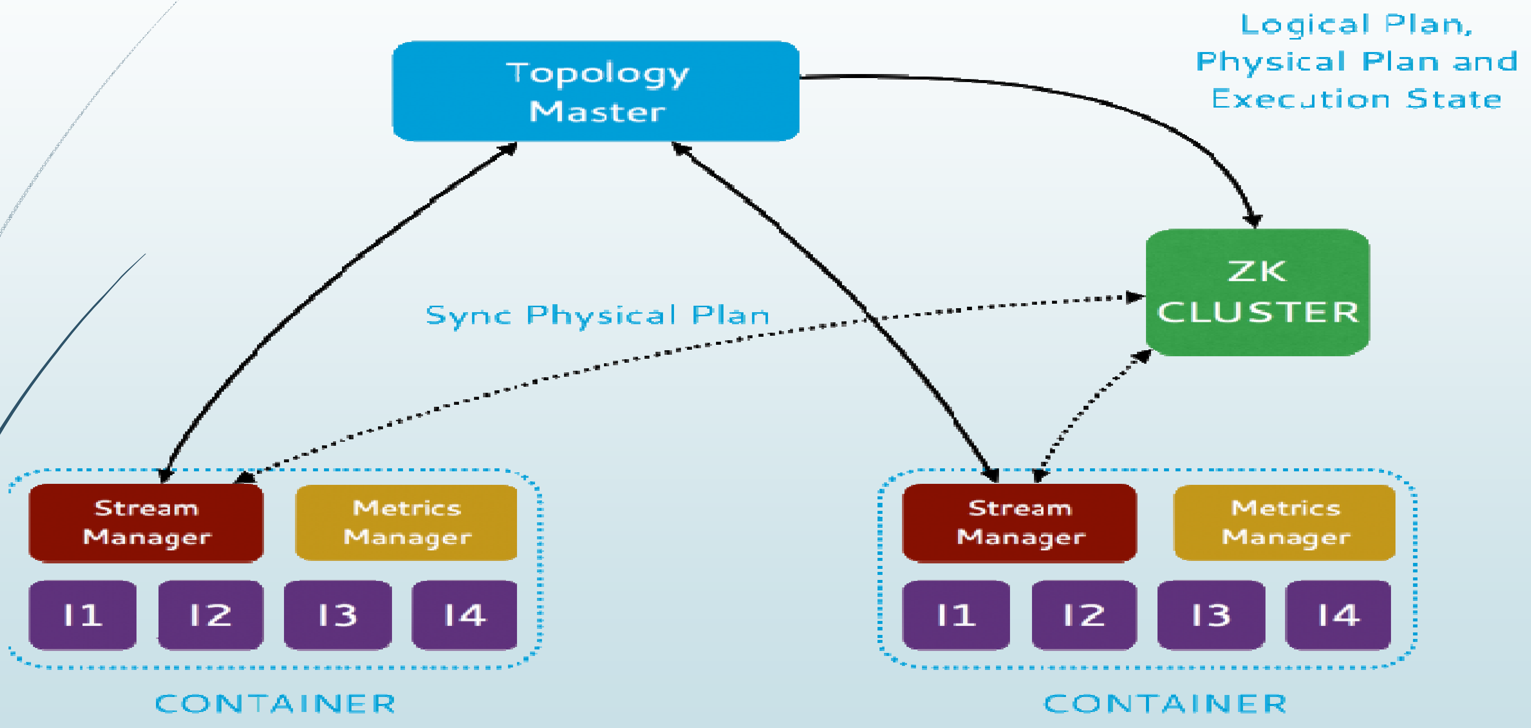
Support for back pressure
Topologies should self adjusting

Efficiency
Reduce resource consumption

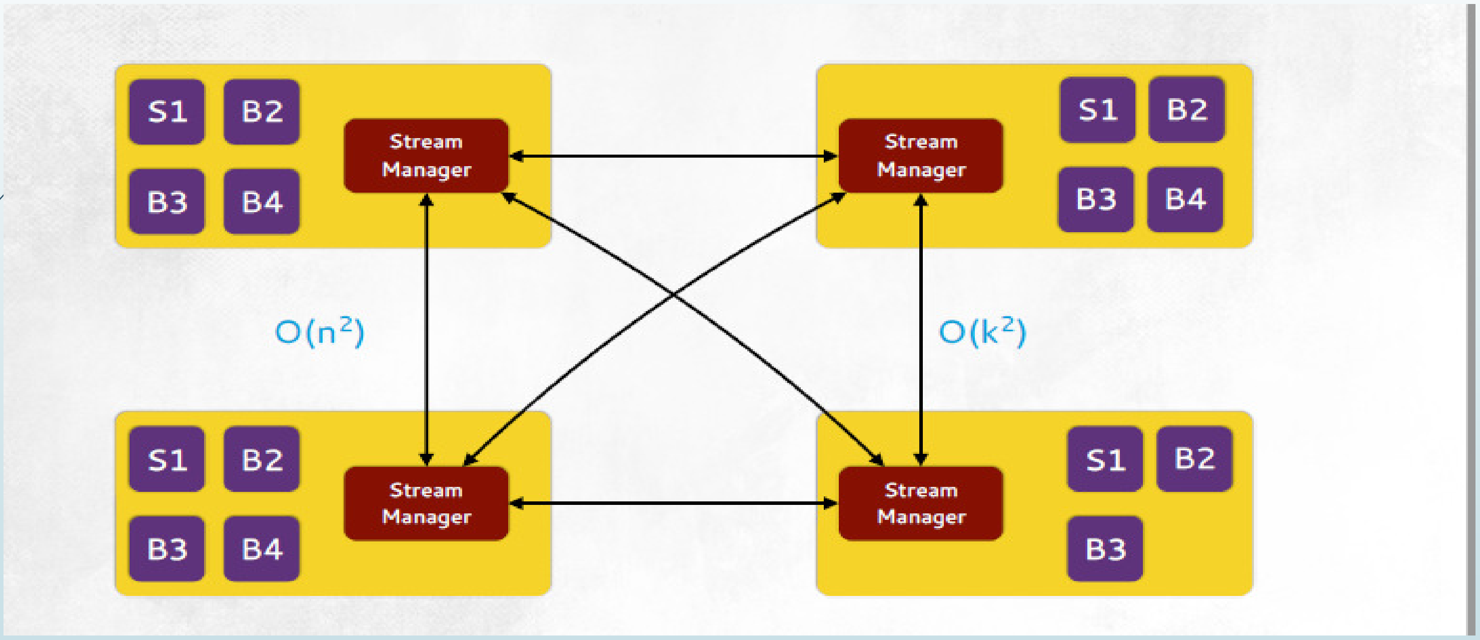
HERON ARCHITECHTURE:



TOPOLOGY ARCHITECTURE



STREAM MANAGER



ARCHITECHTURE EXPLANATION

- Users employ the Storm API to create and submit topologies to a scheduler. The scheduler runs each topology as a job consisting of several containers. One of the containers runs the topology master, responsible for managing the topology. The remaining containers each run a stream manager responsible for data routing, a metrics manager that collects and reports various metrics and a number of processes called Heron instances which run the user-defined spout/bolt code. These containers are allocated and scheduled by scheduler based on resource availability across the nodes in the cluster. The metadata for the topology, such as physical plan and execution details, are kept in Zookeeper.
- Users employ the Heron (spouts/bolts programming) API to create and deploy topologies to the Aurora schedule, using a Heron command line tool.

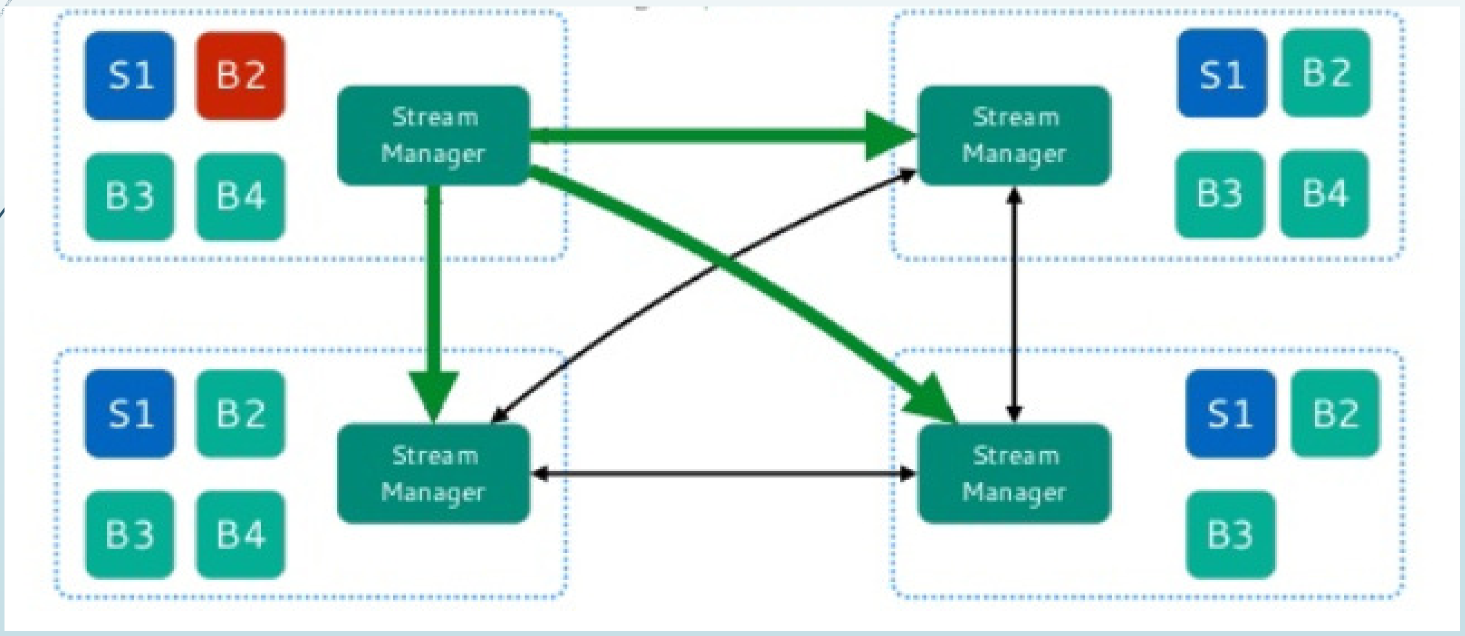
ARCHITECHTURE EXPLANATION

- Each topology is run as an Aurora job consisting of several containers The first container runs a process called the Topology Master. The remaining containers each run a Stream Manager, a Metrics Manager, and a number of processes called Heron Instances. Multiple containers can be launched on a single. The metadata for the topology (which includes information about he user who launched the job, the time of launch, and the execution details) are kept in Zookeeper.
- The Heron Instances are written in Java, as they need to run user logic code (which is written in Java). There is one JVM per Heron Instance.
- All Heron processes communicate with each other using protocol buffers (protobufs).

COMPONENTS OF CONTAINER

- **Topology Master:** The Topology Master(TM) is responsible for managing the topology throughout its existence. It provides a single point of contact for discovering the status of the topology.
- **Stream Manager:** The key function of the Stream Manager (SM)is to manage the routing of tuples efficiently. Each Heron Instance(HI) connects to its local SM to send and receive tuples
- **Heron Instance:** The main work for a spout or a bolt is carried out in the Heron instances(HIs). Unlike the Storm worker, each HI is a JVM process, which runs only a single task of the spout or the bolt.
- **Metrics Manager :**The Metrics Manager (MM) collects and exports metrics from all the components in the system. These metrics include system metrics and user metrics for the topologies. There is one metrics manager for each container, to which the Stream Manager and Heron Instances report their metrics.

BACKPRESSURE MECHANISM



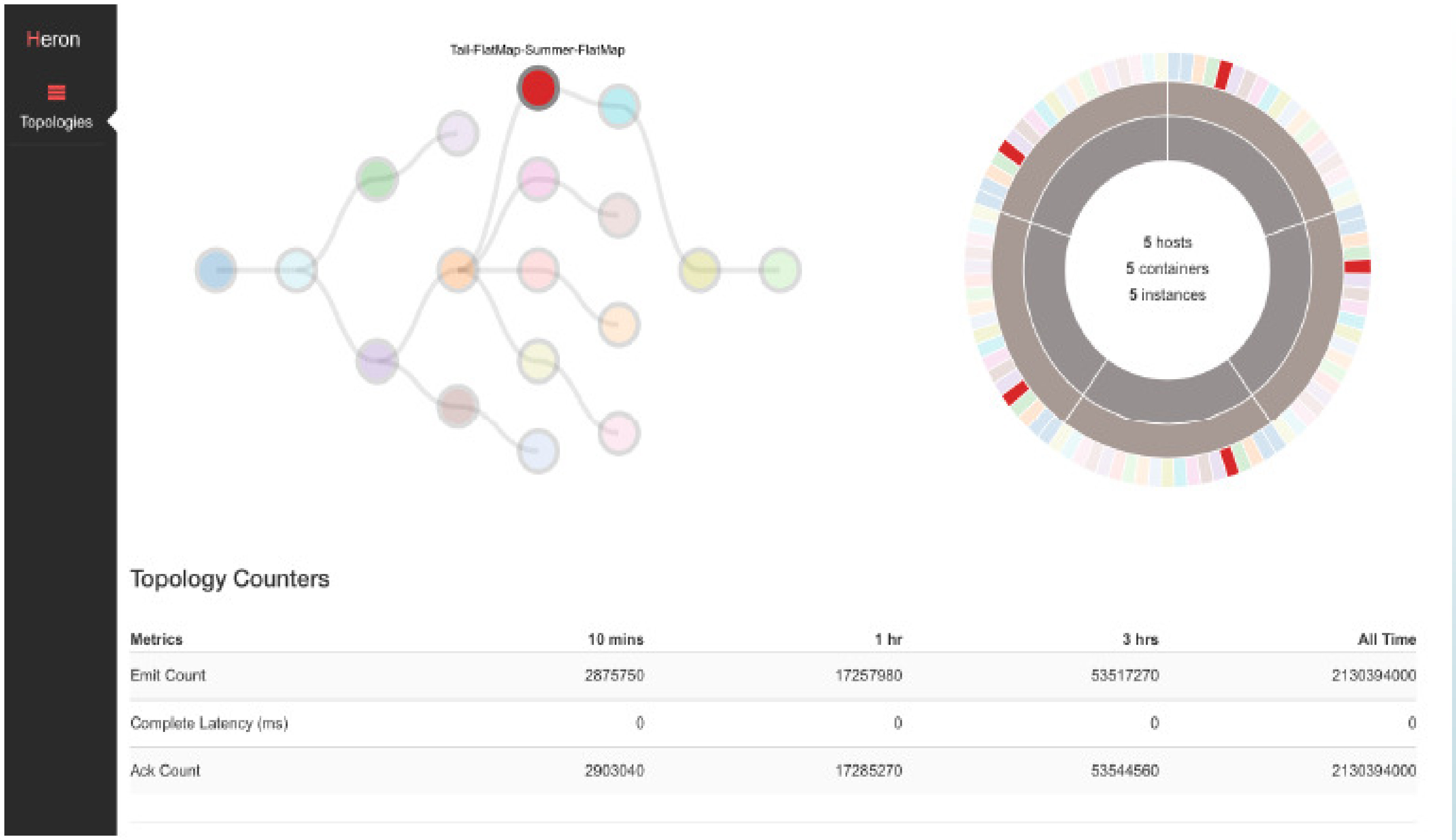
HERON IN PRODUCTION(IMPLEMENTATION)

- To get Heron working in production, we needed several additional functionalities, which include:
- a) the ability for users to interact with their topologies
- b) the ability for users to view metrics

To Accommodate such features Heron Tracker, Heron UI and Heron Viz

ADDITIONAL COMPONENTS

- **Heron Tracker**: The Heron Tracker acts as a gateway to access several information about topologies. It interfaces with the same Zookeeper instance that the topologies use to save their metadata, and collects additional information about the topologies. It keep a watch on the topology which are generated and killed.
- **Heron UI**:This component interact with topology using Heron Tracker API and display visual representation of topologies including logical and physical plan. The Logical plan is displayed as a direct acyclic graph with each node differently coded color. The physical plan is displayed as a concentric circles with inner circles representing hosts, middle depicting containers and outer representing the instance of components
- **Heron Viz:** Heron Viz is a service that creates the dashboard used to view the metrics collected by the Metrics Manager for a topology. This service periodically contacts the Heron Tracker for any new topology



Topology Counters

Metrics	10 mins	1 hr	3 hrs	All Time
Emit Count	2875750	17257980	53517270	2130394000
Complete Latency (ms)	0	0	0	0
Ack Count	2903040	17285270	53544560	2130394000

FEATURES OF HERON

- **Handling spikes and congestion:** Heron has a back pressure mechanism that dynamically adjusts the rate of data flow in a topology during execution, without compromising data accuracy. This is particularly useful under traffic spikes and pipeline congestions.
- **Easy debugging:** Every task runs in process-level isolation, which makes it easy to understand its behavior, performance and profile. Furthermore, the sophisticated UI of Heron topologies, shown in Figure 3 below, enables quick and efficient troubleshooting for issues.
- **Compatibility with Storm**: Heron provides full backward compatibility with Storm, so we can preserve our investments with that system. No code changes are required to run existing Storm topologies in Heron, allowing for easy migration.
- **Scalability and latency**: Heron is able to handle large-scale topologies with high throughput and low latency requirements. Furthermore, the system can handle a large number of topologies.

HERON VS STORM(WITH ACKNOWLEDGEMENTS)

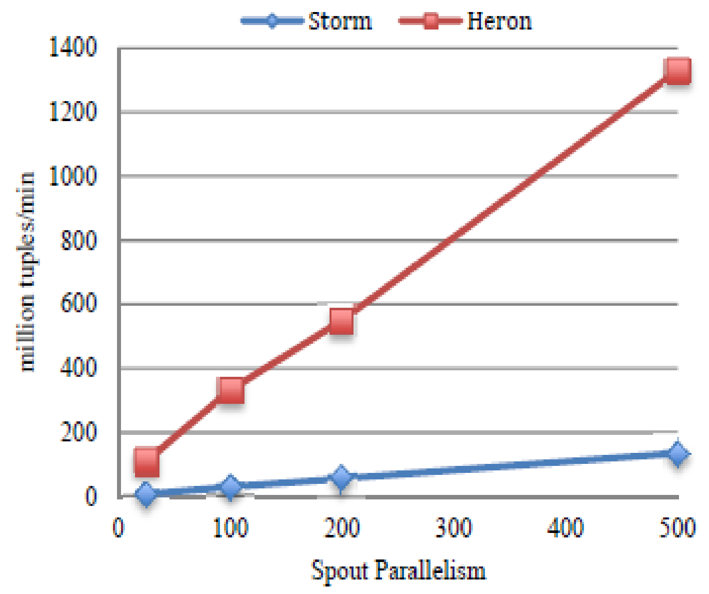


Figure 9: Throughput with acknowledgements

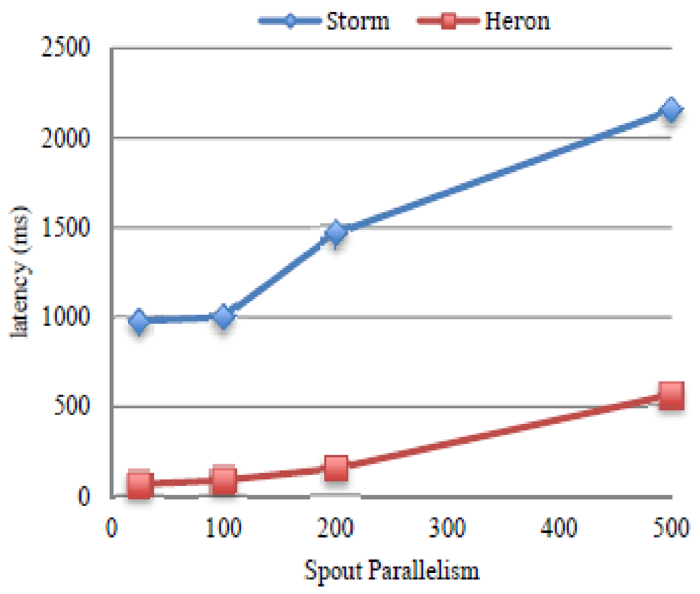


Figure 10: End-to-end latency with acknowledgements

HERON VS STORM(Continued)

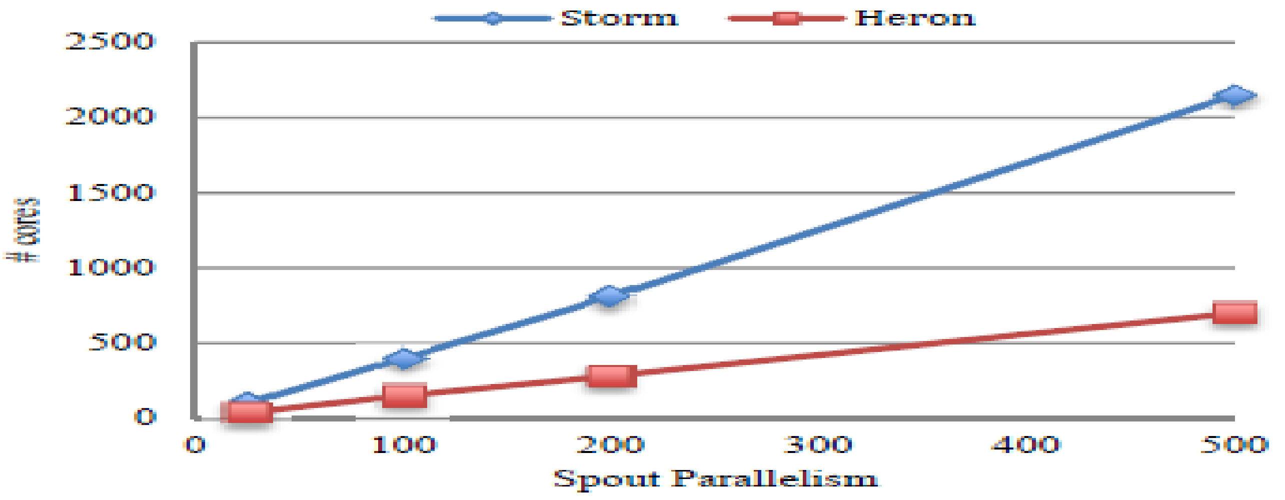


Figure 11: CPU usage with acknowledgements

HERON VS STORM(WITHOUT) ACKNOWLEDGEMENTS

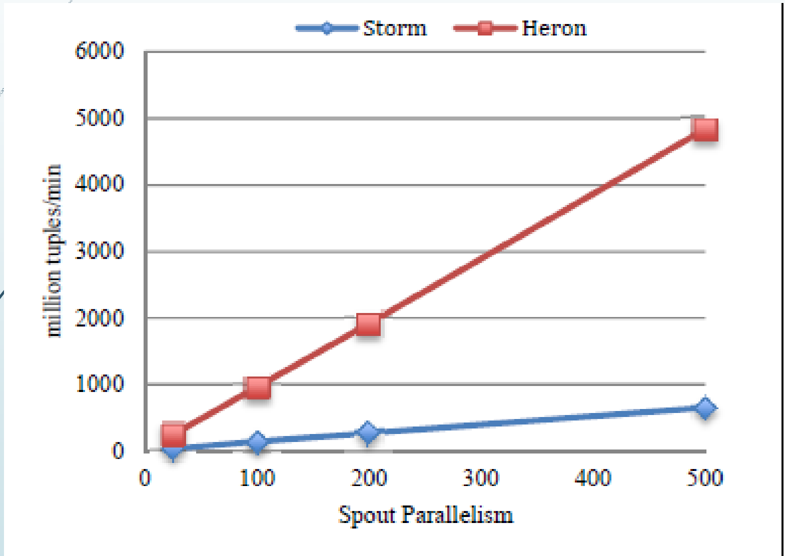


Figure 12: Throughput with acknowledgements disabled

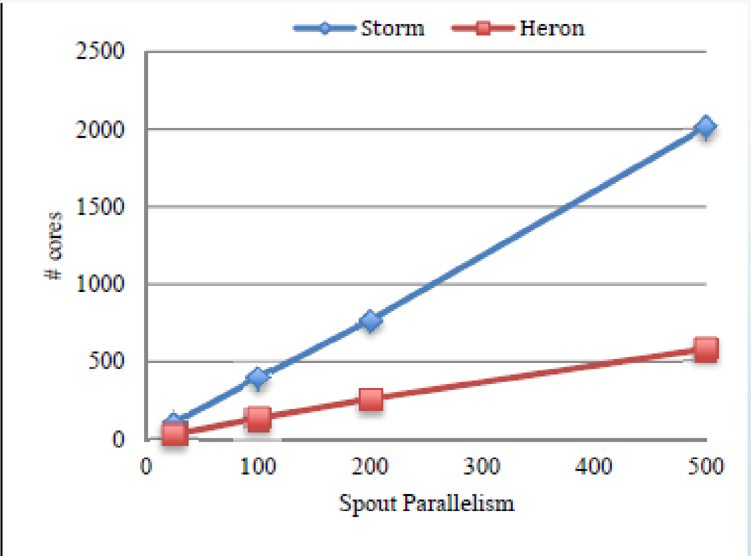
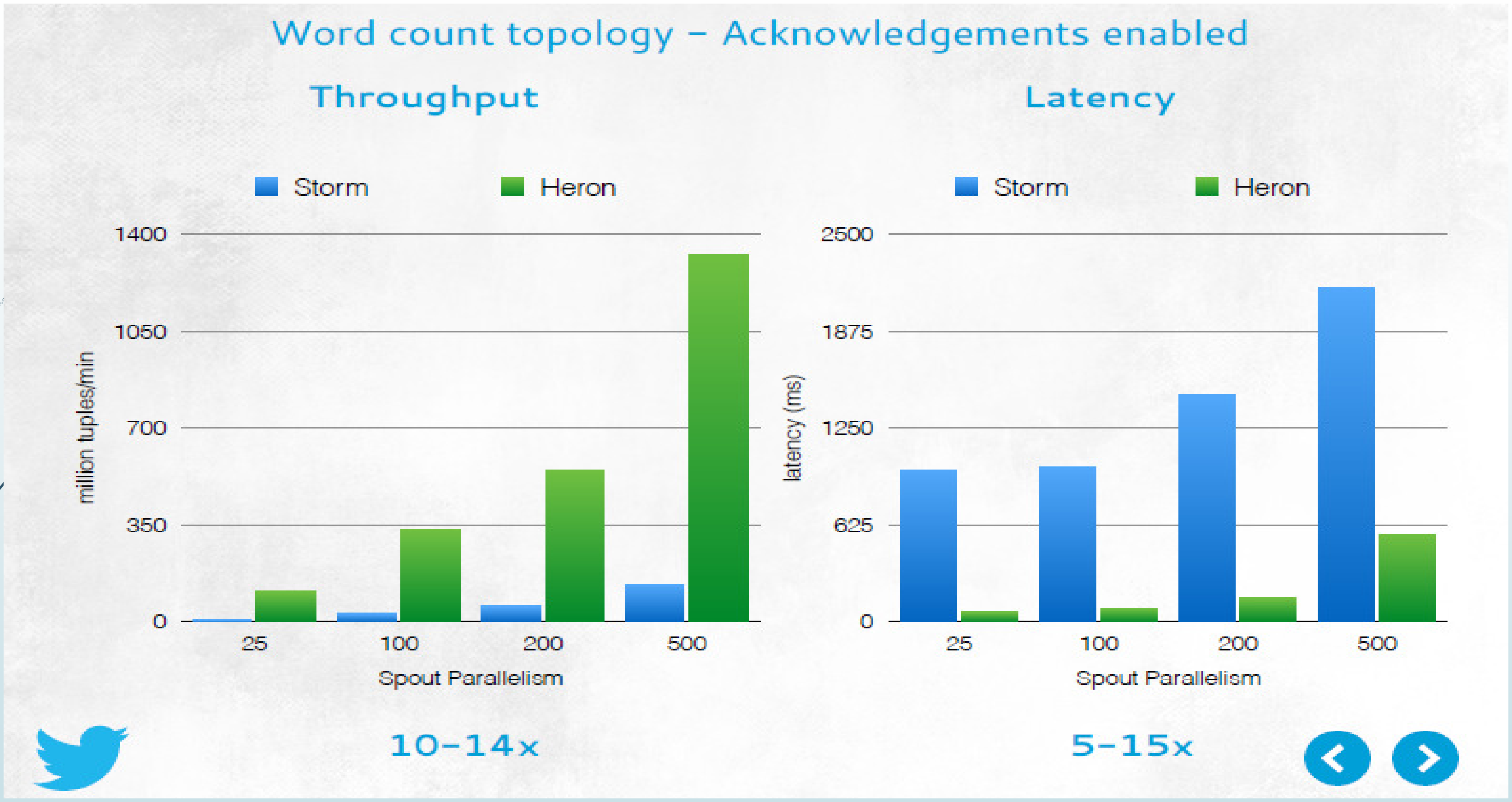
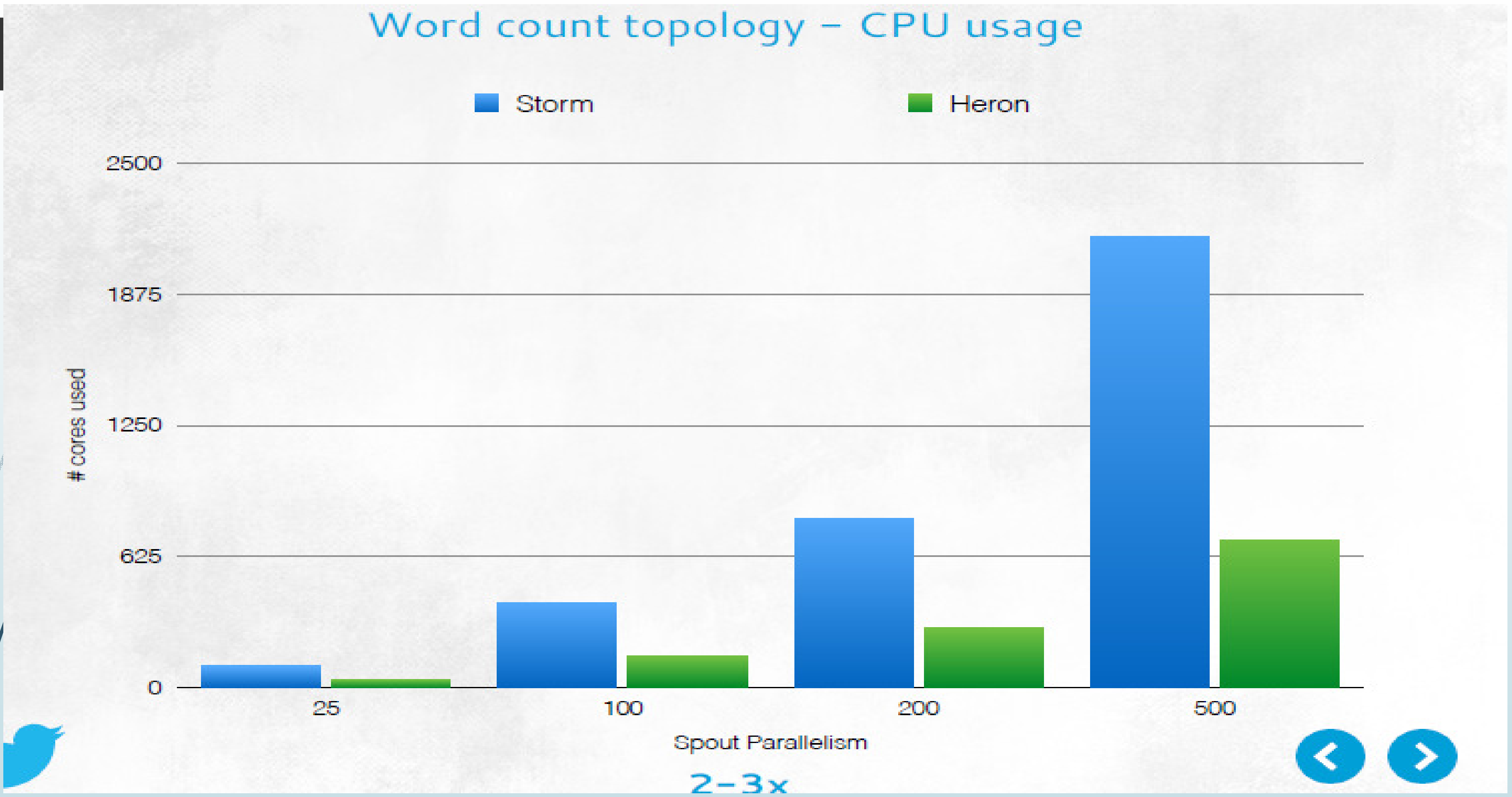


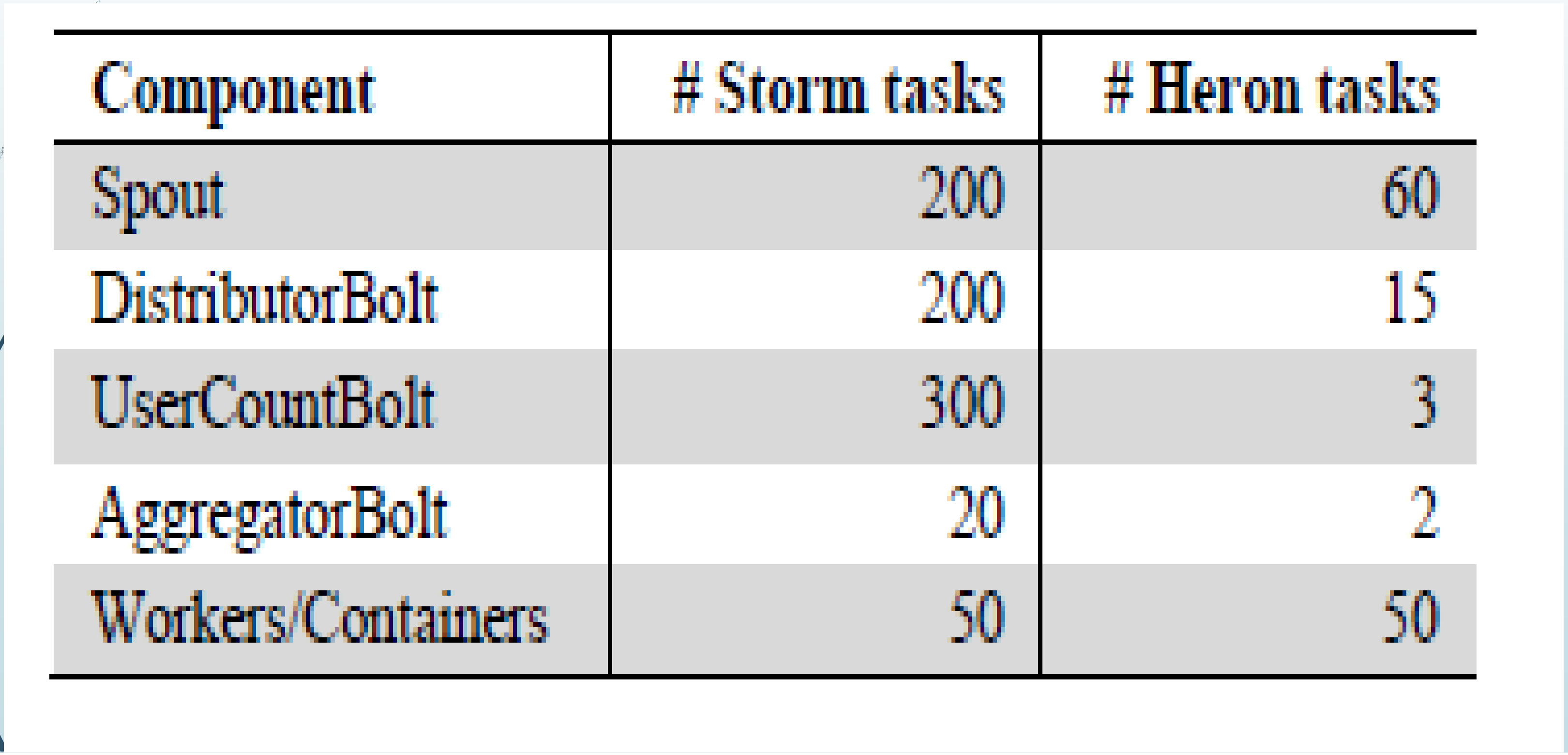
Figure 13: CPU usage with acknowledgements disabled


Word count topology – Acknowledgements enabled
Throughput
Latency
Storm
Heron
1400
1050
700
350
0
million tuples/min
25
100
200
500
Spout Parallelism
2500
1875
1250
625
0
latency (ms)
10-14x
5-15x


Word count topology – CPU usage
Storm
Heron
2500
1875
1250
625
0
# cores used
25
100
200
500
Spout Parallelism

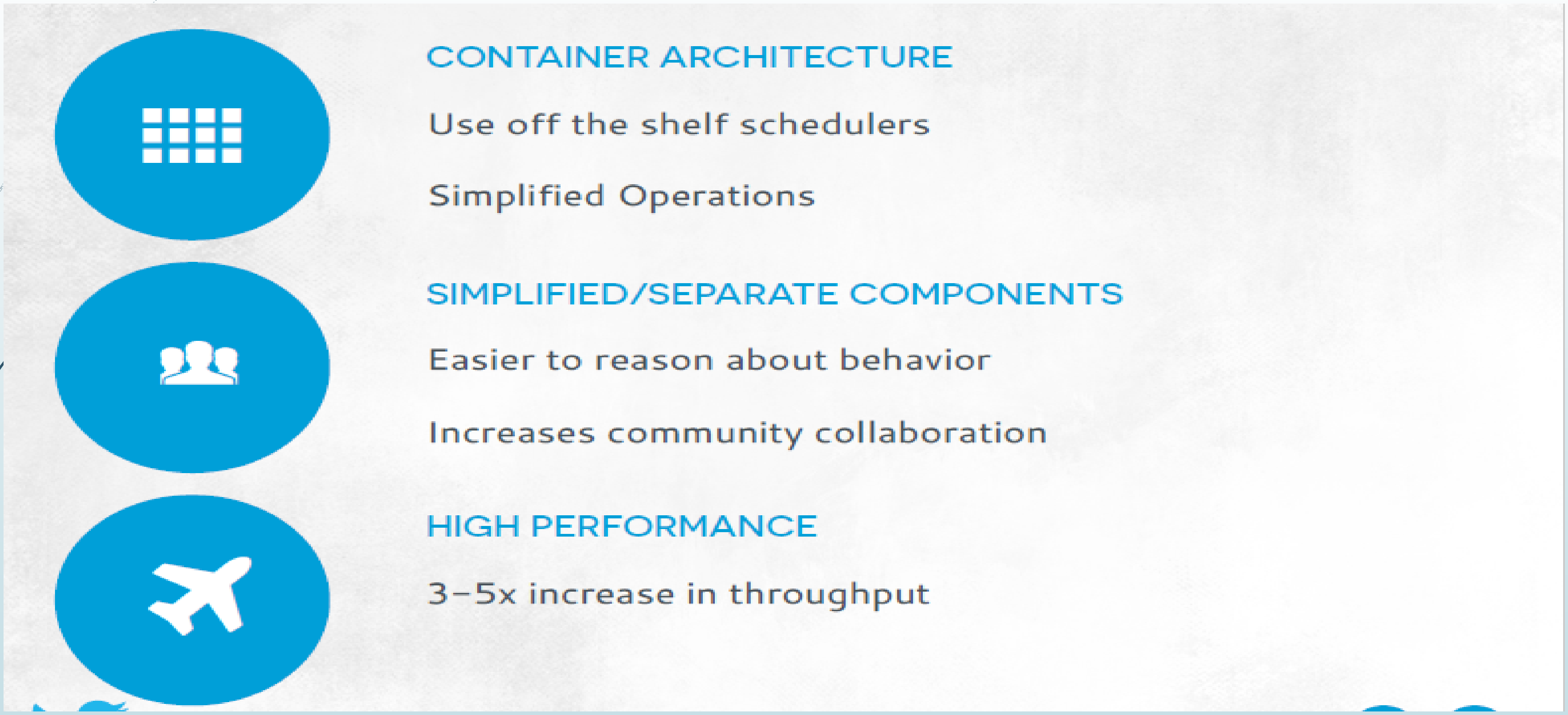
2–3x

STROM VS HERON EFFICIENCY



Component	# Storm tasks	# Heron tasks
Spout	200	60
DistributorBolt	200	15
UserCountBolt	300	3
AggregatorBolt	20	2
Workers/Containers	50	50

CONCLUSION



CONTAINER ARCHITECTURE

Use off the shelf schedulers

Simplified Operations

SIMPLIFIED/SEPARATE COMPONENTS

Easier to reason about behavior

Increases community collaboration

HIGH PERFORMANCE

3–5x increase in throughput

- THANK YOU