

02-flink-powered-customer-experience-scaling-from-5-billion-down-to-one	2
03-the-trade-desks-year-in-flink	41
04-managing-flink-on-kubernetes-flinkk8soperator	73
08-high-cardinality-data-stream-processing-with-large-states	99
13-building-financial-identity-platform-using-apache-flink	124
14-elastic-data-processing-with-apache-flink-and-apache-pulsar	157
18-creating-millions-of-user-sessions-using-complex-event-processing	208
19-hunting-for-attack-chains-in-event-streams	248
20-deploying-onnx-models-on-flink	274
22-developing-and-operating-real-time-applications-with-oceanus	294
23-apache-beam-portability-in-the-times-of-real-time-streaming	315
24-adventures-in-scaling-from-zero-to-5-billion-data-points-per-day	346
27-scaling-a-real-time-streaming-warehouse-with-apache-flink-parquet-and-kubernetes	419
29-using-flink-to-inspect-live-data-as-it-flows-through-a-data-pipeline	447
31-moving-from-lambda-and-kappa-architectures-to-kappa-at-uber1	468

FLINK-POWERED CUSTOMER EXPERIENCE: SCALING FROM 5 BILLION DOWN TO ONE

Dave Torok

Distinguished Architect

Comcast Corporation

2 April, 2019

Flink Forward – San Francisco 2019



A global media and technology company with several businesses, including Comcast, NBCUniversal, and Sky.

COMCAST		NBCUniversal					sky	
Products & Services		Cable Networks		Broadcast	Film	Parks	Products & Services	
xfinity xFi	xfinity xi			 NBC	 TELEMUNDO	 UNIVERSAL PICTURES	 UNIVERSAL STUDIOS HOLLYWOOD	 sky Q  sky mobile
xfinitymobile	xfinity home				NBCUniversal Owned Television Stations  NBC Owned Television Stations 	 FOCUS FEATURES	 UNIVERSAL ORLANDO RESORT	 sky store 
COMCAST BUSINESS	COMCAST SPOTLIGHT [®]			 NBC Sports	 NBC NEWS	 UNIVERSAL PICTURES HOME ENTERTAINMENT	 UNIVERSAL STUDIOS JAPAN	 sky broadband 
Comcast Spectacor					 TeleXitos	 COZI	 UNIVERSAL PICTURES	Channels & Content
COMCAST SPECTACOR								 sky atlantic 
WELLS FARGO CENTER		Digital & Other	Snap Inc.*			VOX MEDIA*	BuzzFeed	
COMCAST VENTURES								

COMCAST CUSTOMER RELATIONSHIPS

**30.3 MILLION OVERALL CUSTOMER
RELATIONSHIPS AT 2018 YEAR END**

**25.1 MILLION RESIDENTIAL HIGH-SPEED
INTERNET CUSTOMERS AT 2018 YEAR
END**

**1.2 MILLION RESIDENTIAL HIGH-SPEED
INTERNET CUSTOMER NET ADDITIONS IN
2018**



DELIVER THE ULTIMATE CUSTOMER EXPERIENCE

IS THE CUSTOMER HAVING A GOOD EXPERIENCE
FOR HIGH SPEED DATA (HSD) SERVICE?



IF THE CUSTOMER ENGAGES US DIGITALLY, CAN
WE OFFER A SELF-SERVICE SOLUTION?



IF THERE IS AN ISSUE CAN WE OFFER OUR AGENTS
AND TECHNICIANS A DIAGNOSIS TO HELP SOLVE
THE PROBLEM QUICKER?



REDUCE THE TIME TO RESOLVE ISSUES

REDUCE COST TO THE BUSINESS AND THE
CUSTOMER

REDUCE COST AND TIME TO RESOLUTION

CUSTOMER SELF-SERVICE

COST

\$ ~0

TIME
< 15 MINUTES

TALK TO AN AGENT

**COST ORDER OF
MAGNITUDE:**

\$ 5-10

TIME:
< AN HOUR

TECHNICIAN VISIT TO HOME

**COST ORDER OF
MAGNITUDE:**

\$ 50-100

TIME:
AT LEAST A DAY



How do we personalize
the conversation?

CUSTOMER EXPERIENCE INDICATORS

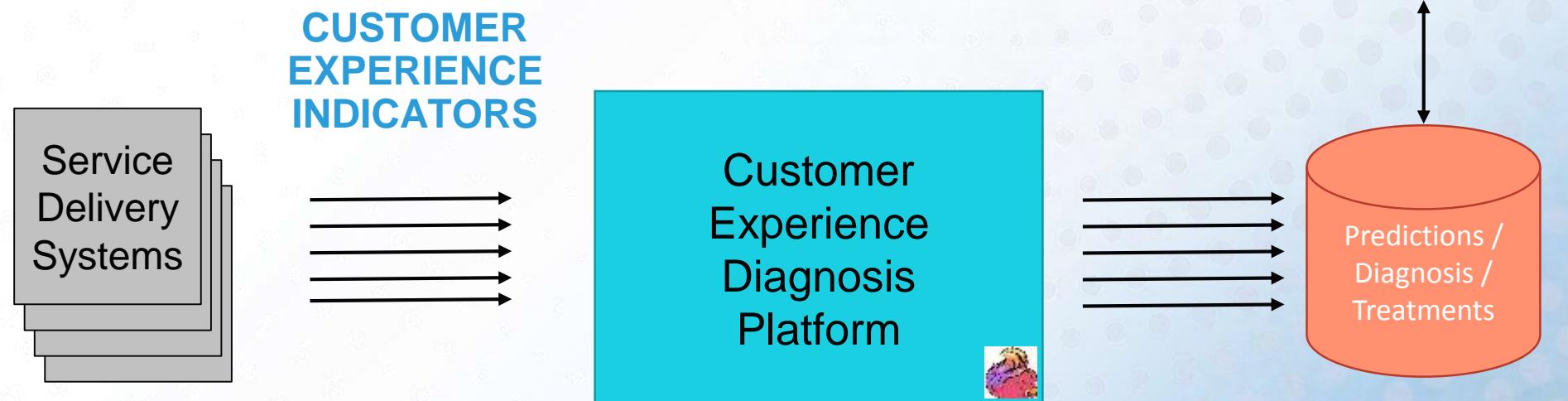


Comcast collects, stores, and uses all data in accordance with our privacy disclosures to users and applicable laws.



COMCAST

HIGH LEVEL CONCEPT



Comcast collects, stores, and uses all data in accordance with our privacy disclosures to users and applicable laws.

SIMPLIFY INDICATORS

```
"packetErrors": ["327"],  
"systemUptime":  
    {"string": "4:26:34.27"},  
"octets": ["174756692"],  
"timerError": { "int": 0 },  
"sysInfoHwphase2": {"int": 1}  
"firmwareD1Error": { int": 0 },  
"sysErrorTr69NotRegistered": null,  
"sysShSnmpsubagentRestartMw": null,  
"wifiErrorKernelDriverNotconnected":  
null,  
"forcedNonDeviceRebootError":  
    { "int": 0 }
```

SIMPLIFY INDICATORS

```
"packetErrors": ["327"],  
"systemUptime":  
    {"string": "4:26:34.27"},  
"octets": ["174756692"],  
"timerError": { "int": 0 },  
"sysInfoHwphase2": {"int": 1}  
"firmwareD1Error": { int": 0 },  
"sysErrorTr69NotRegistered": null,  
"sysShSnmpsubagentRestartMw": null,  
"wifiErrorKernelDriverNotconnected":  
null,  
"forcedNonDeviceRebootError":  
    { "int": 0 }
```

GREEN
CUSTOMER EXPERIENCE
IS GOOD



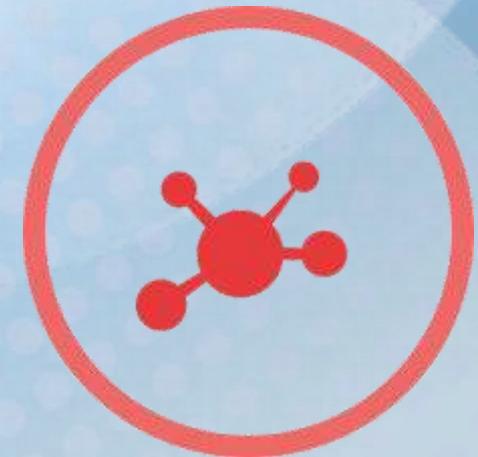
SIMPLIFY INDICATORS

```
"packetErrors": ["327"],  
"systemUptime":  
    {"string": "4:26:34.27"},  
"octets": ["174756692"],  
"timerError": { "int": 0 },  
"sysInfoHwphase2": {"int": 1}  
"firmwareD1Error": { int": 0 },  
"sysErrorTr69NotRegistered": null,  
"sysShSnmpsubagentRestartMw": null,  
"wifiErrorKernelDriverNotconnected":  
null,  
"forcedNonDeviceRebootError":  
    { "int": 0 }
```

GREEN
CUSTOMER EXPERIENCE
IS GOOD



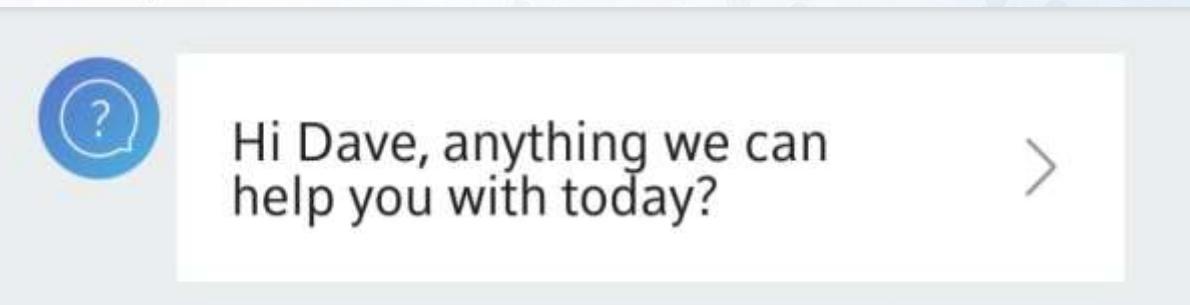
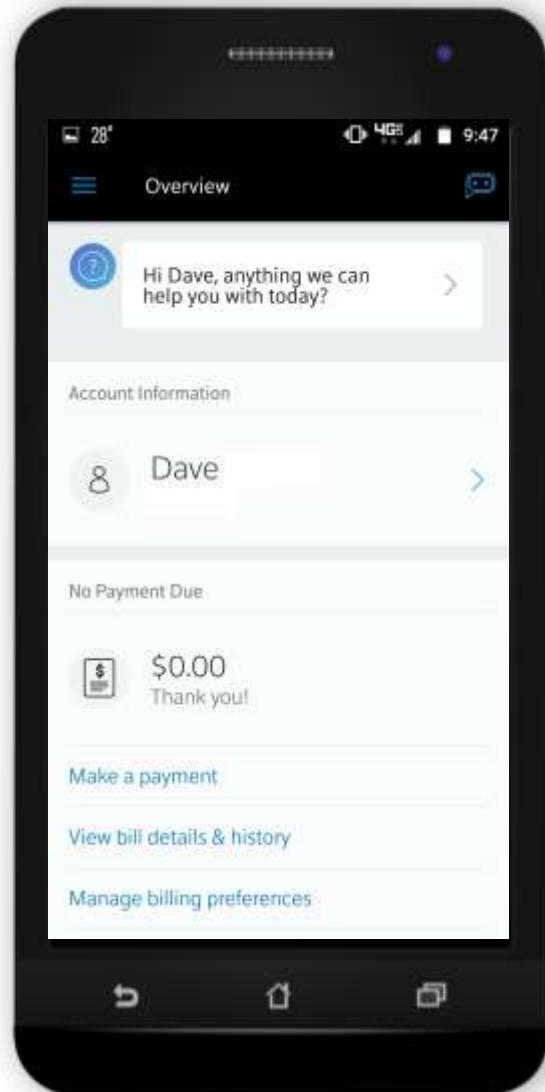
RED
POSSIBLE IMPACT TO
CUSTOMER EXPERIENCE



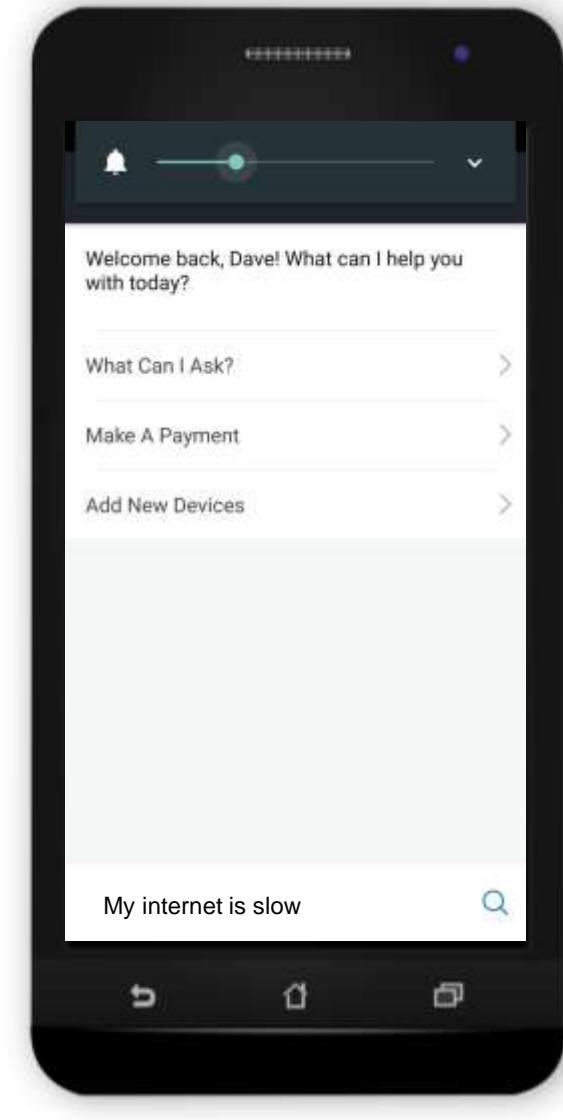


What is the Digital Experience?

XFINITY ASSISTANT EXPERIENCE



XFINITY ASSISTANT EXPERIENCE



My internet is slow

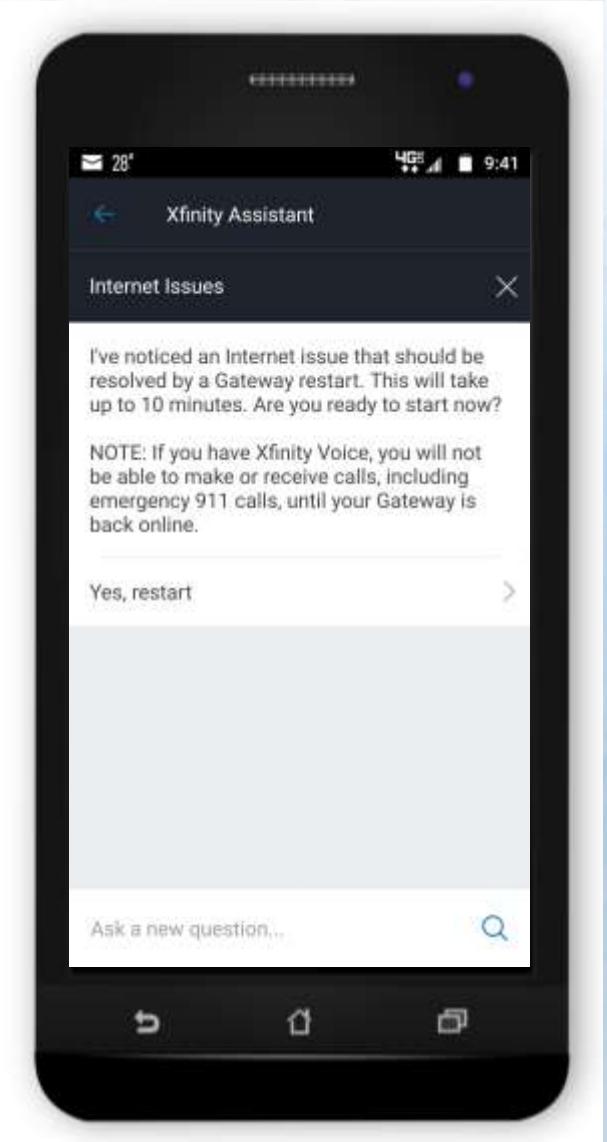
XFINITY ASSISTANT EXPERIENCE

FLINK POWER!

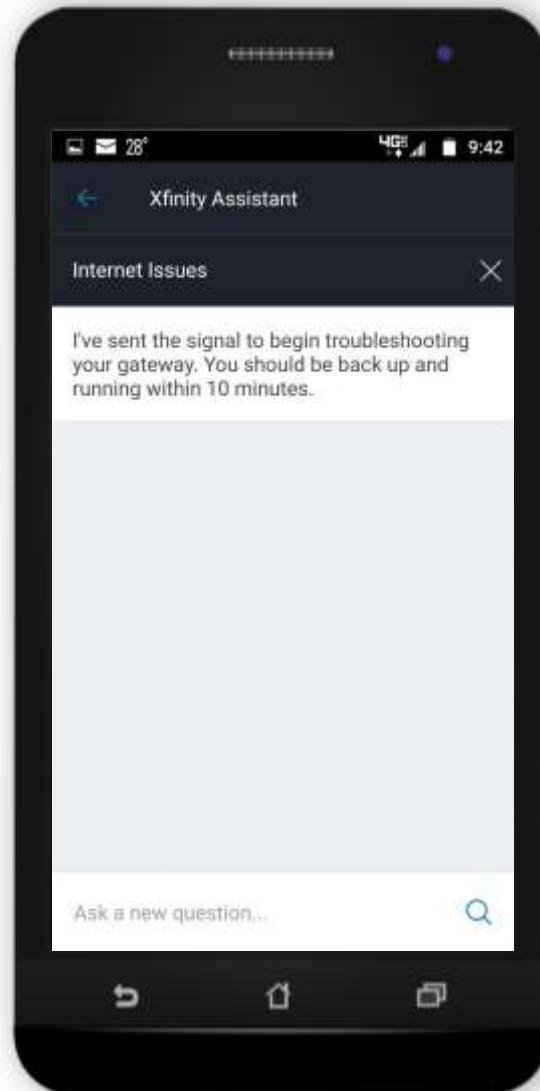


XFINITY ASSISTANT EXPERIENCE

I've noticed an Internet issue that should be resolved by a Gateway restart. This will take up to 10 minutes. Are you ready to start now?

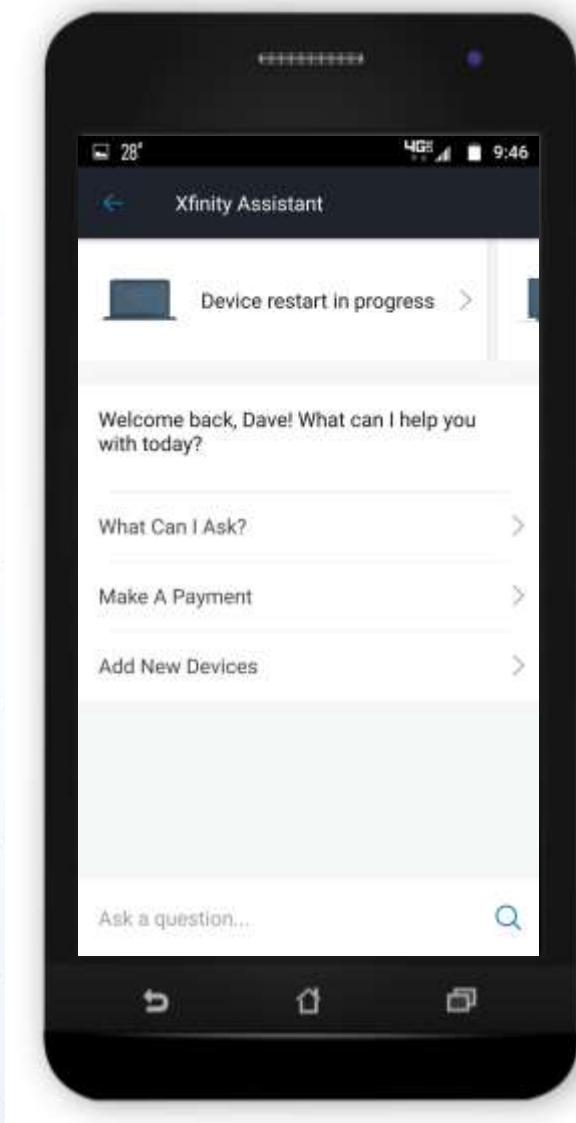


XFINITY ASSISTANT EXPERIENCE



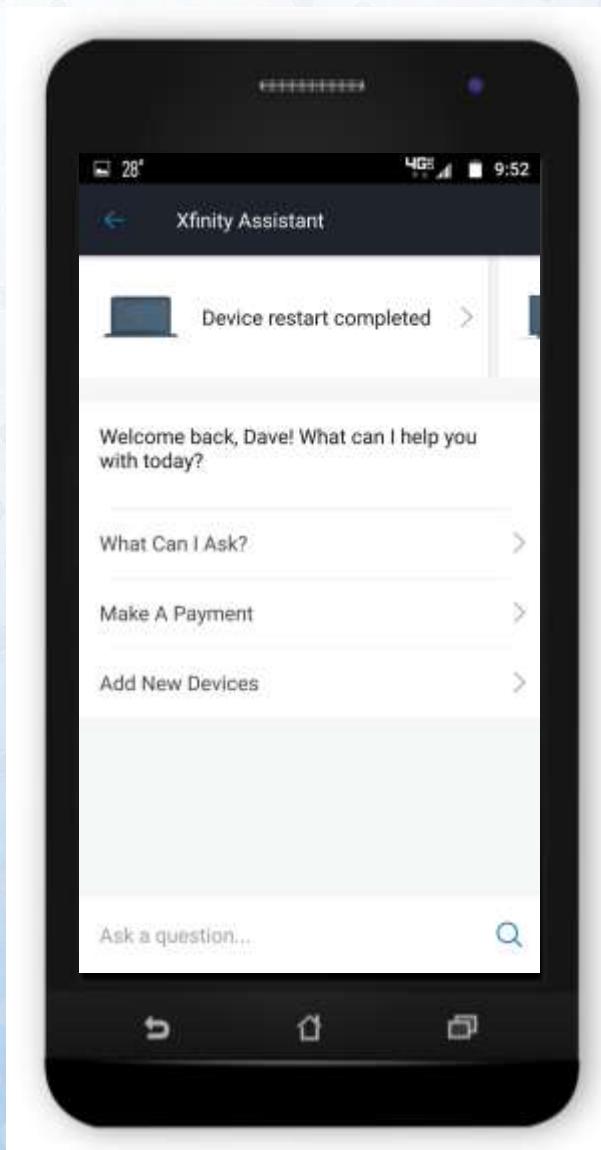
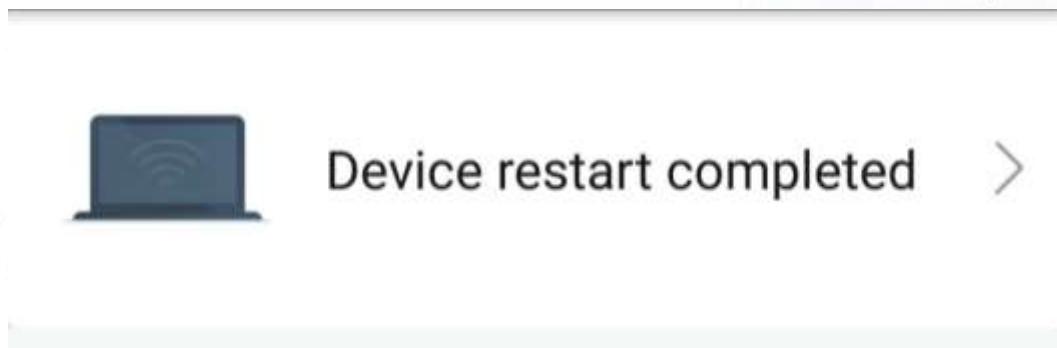
I've sent the signal to begin troubleshooting your gateway. You should be back up and running within 10 minutes.

XFINITY ASSISTANT EXPERIENCE

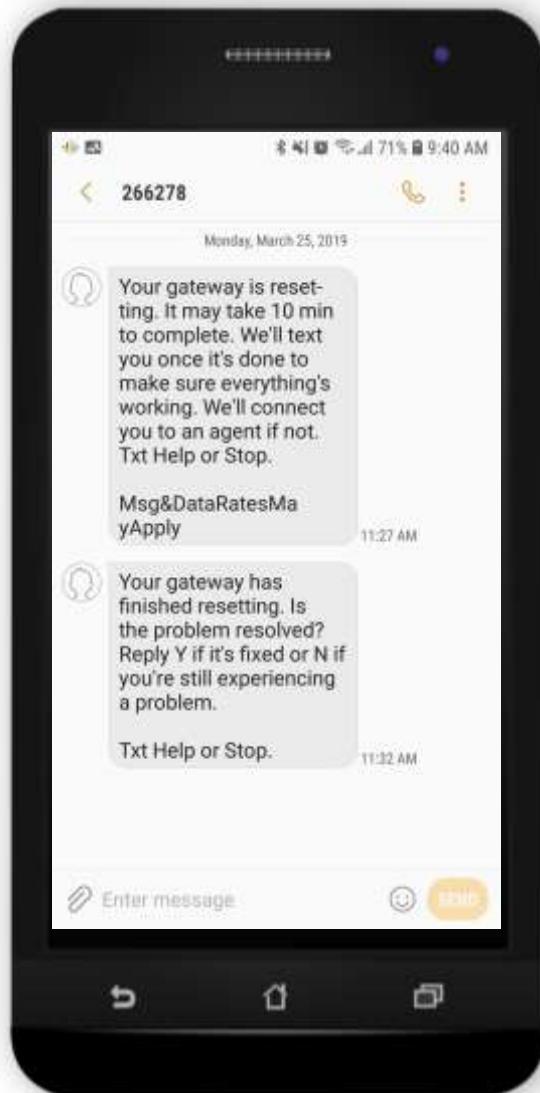


Device restart in progress >

XFINITY ASSISTANT EXPERIENCE



XFINITY ASSISTANT EXPERIENCE



Your gateway has finished resetting. Is the problem resolved? Reply Y if it's fixed or N if you're still experiencing a problem.

Txt Help or Stop.

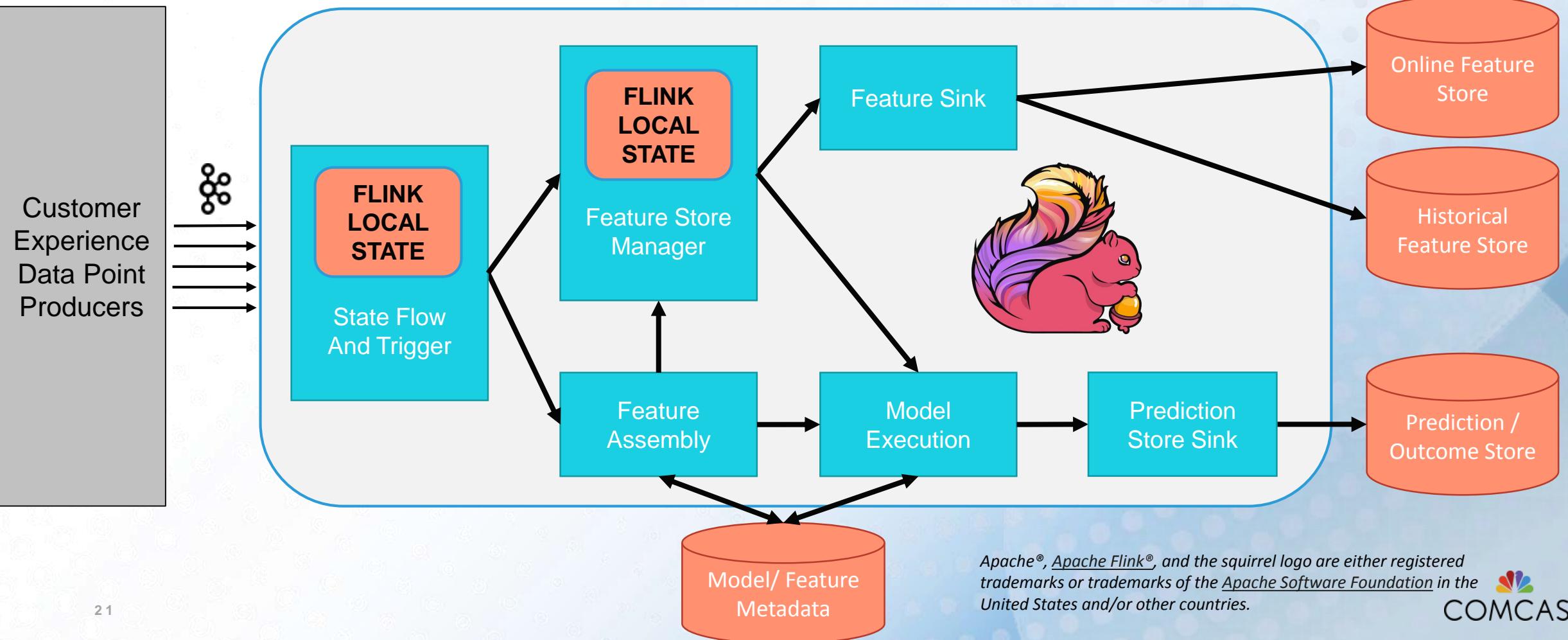
FOLLOWING UP ON THE INTERACTION:

Is the problem resolved?

If so, great!

If not, offer to talk with an agent.

POWERED BY APACHE FLINK®



CHALLENGES ALONG THE WAY

THE
TRIGGER AND
DIAGNOSIS
PROBLEM

THE
REST
PROBLEM

THE
INEFFICIENT
OBJECT HANDLING
PROBLEM

THE
FEATURE STORE
PROBLEM

THE
VOLUME
PROBLEM

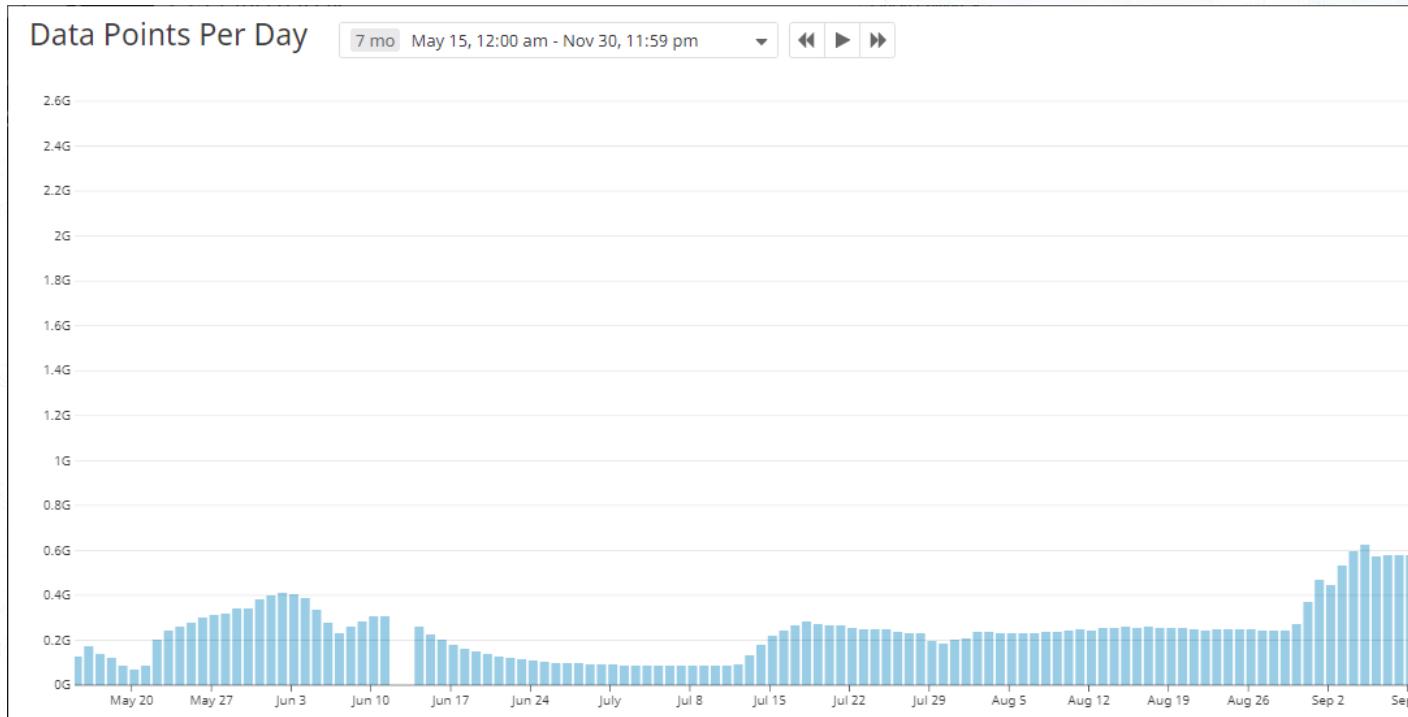
THE
CUSTOMER STATE
PROBLEM

THE
CHECKPOINT
PROBLEM

THE
TRIGGER AND
DIAGNOSIS
PROBLEM

THE
REALLY HIGH VOLUME
AND FEATURE STORE
PROBLEM #2

DATA POINT VOLUME MAY–NOVEMBER 2018



DATA POINT VOLUME MAY–NOVEMBER 2018



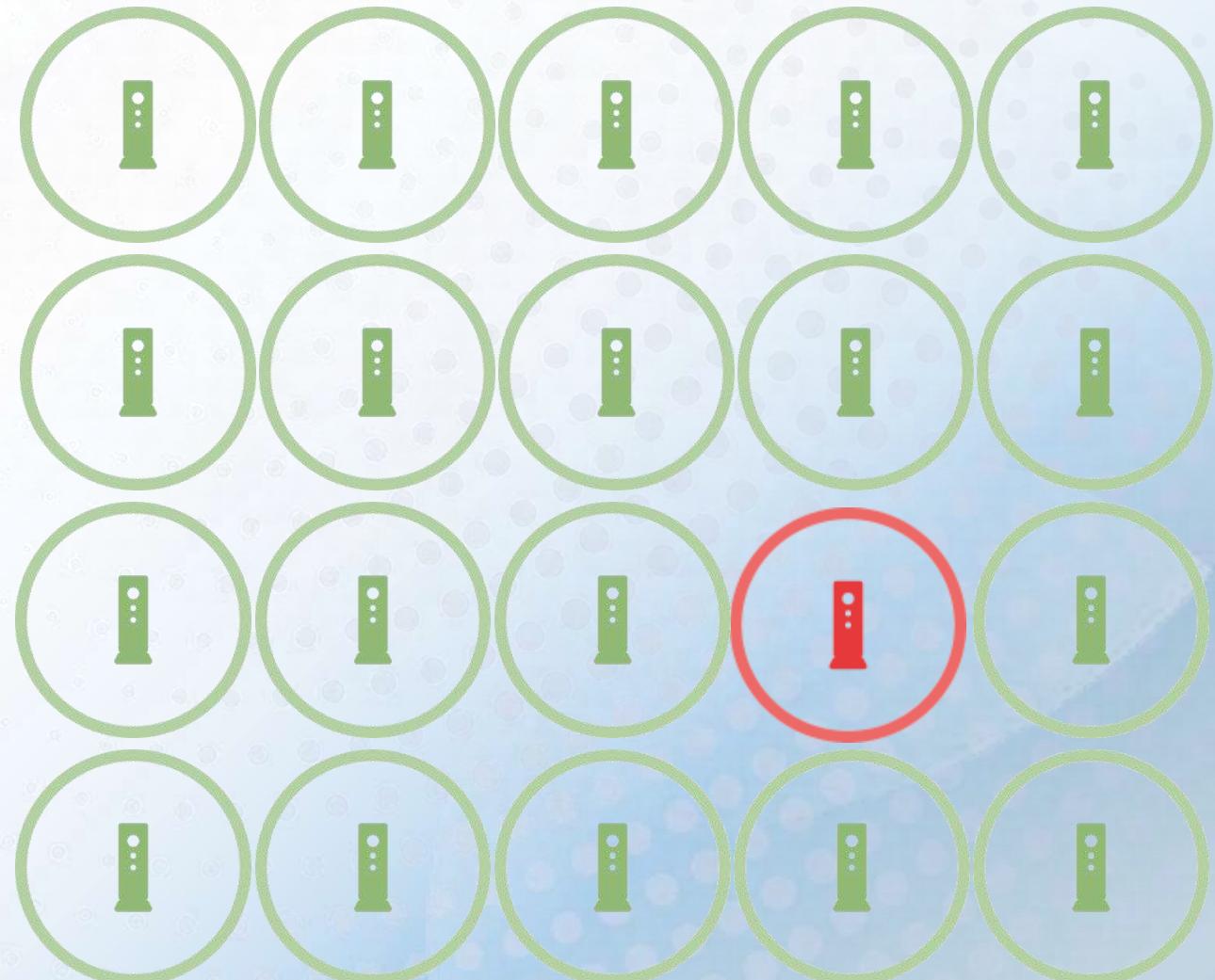
DATA POINT VOLUME MAY–NOVEMBER 2018



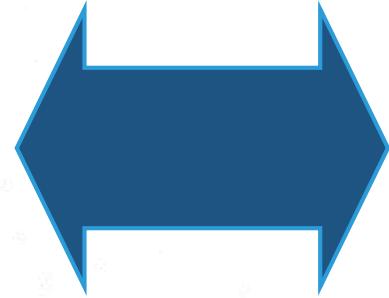
LET'S ADD ANOTHER 3 BILLION!

15 MILLION DEVICES

3 BILLION DATA POINTS

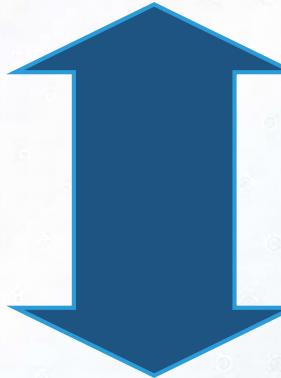


MEETING THE CHALLENGE



HORIZONTAL SCALING

TO MEET THE VOLUME



VERTICAL SCALING

TO PROCESS HIGH-VOLUME DATA SOURCES



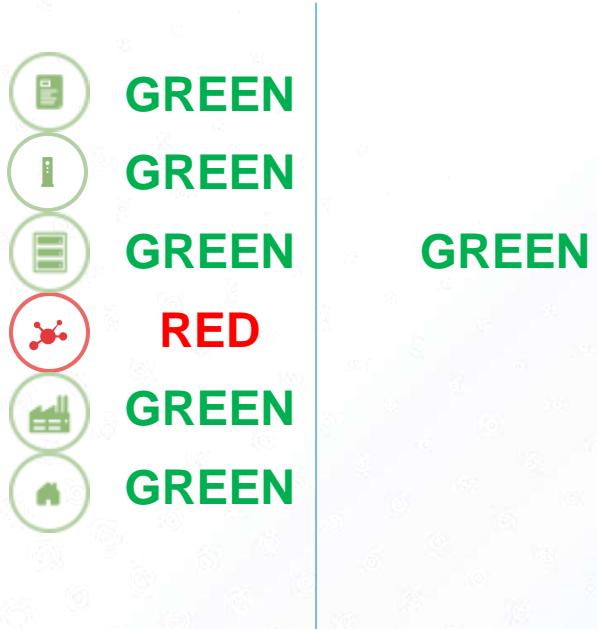
FLINK FEATURES

QUERYABLE STATE
CHECKPOINTING
GLOBAL WINDOWS
CONNECTED STREAMS
STREAMING SINKS

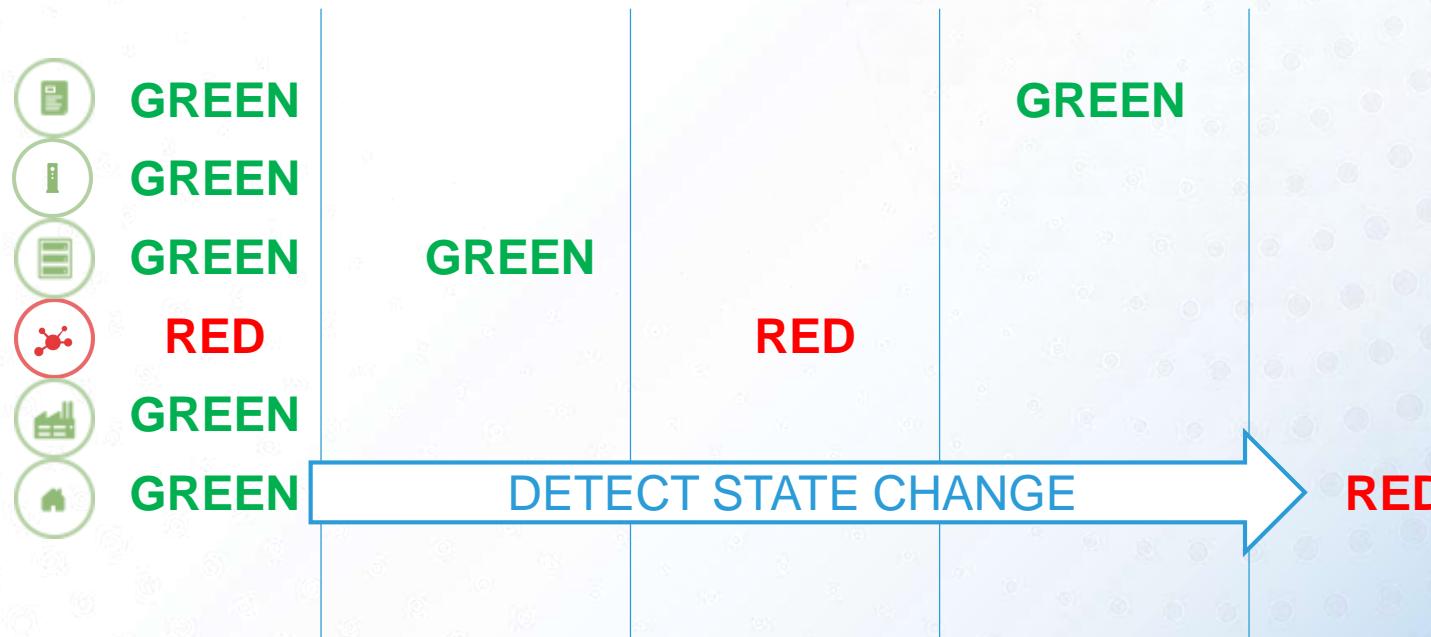
USING FLINK STATE TO TRIGGER DIAGNOSIS

	GREEN
	GREEN
	GREEN
	RED
	GREEN
	GREEN

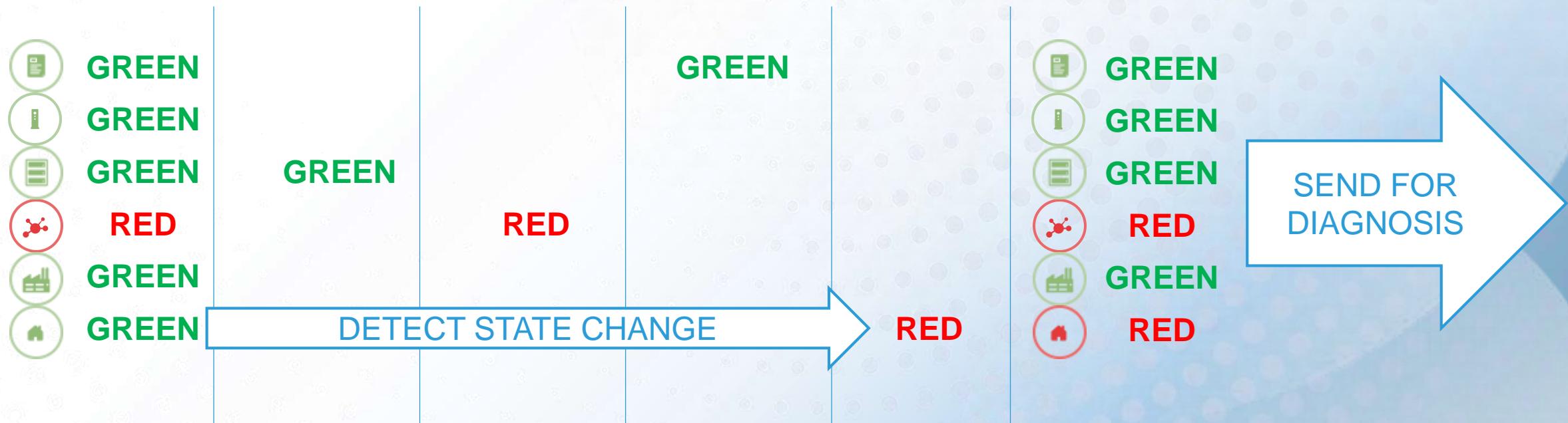
USING FLINK STATE TO TRIGGER DIAGNOSIS



USING FLINK STATE TO TRIGGER DIAGNOSIS



USING FLINK STATE TO TRIGGER DIAGNOSIS



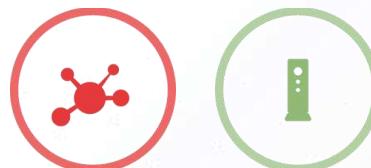
KEEPING TRACK OF DATA POINT STATE

25+ MILLION HIGH SPEED INTERNET CUSTOMERS

10+ DATA POINT TYPES

TWO FLINK STATES TO MANAGE:

RED / GREEN “TRIGGER”



[ENTITY ID]

(DATA POINT TYPE, STATE)

~1 GB TOTAL

QUERYABLE STATE
FEATURE STORE

UP TO 15KB JSON
PER DATA POINT

~ 1 TB TOTAL

WHERE ARE WE TODAY

INDICATORS

725+
MILLION
DATA
POINTS
PER DAY

HIGH-VOLUME INDICATORS

6 BILLION
PER DAY

TOTAL

~7 BILLION
DATA POINTS
PER DAY

WHERE ARE WE TODAY – FLINK CLUSTERS

CLUSTERS

14

INSTANCES

150

VCPU

1100

RAM

5.8 TB

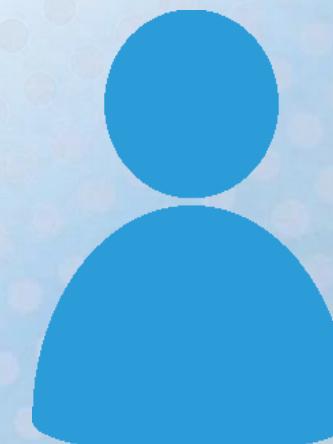
RESULTS

MILLIONS
OF UNIQUE USERS
REACHED

MILLIONS
OF OPERATIONAL
DOLLARS SAVED

INCREASED
CUSTOMER
SATISFACTION WITH
DIGITAL INTERACTION

REDUCED
TIME TO RESOLVE
COMMON ISSUES
THROUGH SELF-
SERVICE



RESULTS

MILLIONS
OF UNIQUE USERS
REACHED

MILLIONS
OF OPERATIONAL
DOLLARS SAVED

INCREASED
CUSTOMER
SATISFACTION WITH
DIGITAL INTERACTION

REDUCED
TIME TO RESOLVE
COMMON ISSUES
THROUGH SELF-
SERVICE

SCALING THE EXPERIENCE TO ONE



**For more details, please attend my
technical presentation!**

ADVENTURES IN SCALING

FROM ZERO TO ~~5~~ 7 BILLION

DATA POINTS PER DAY

5:30 PM – 6:10 PM

Carmel Meeting Room

WE'RE HIRING!



PHILADELPHIA
WASHINGTON, D.C.
MOUNTAIN VIEW
DENVER

COMCAST



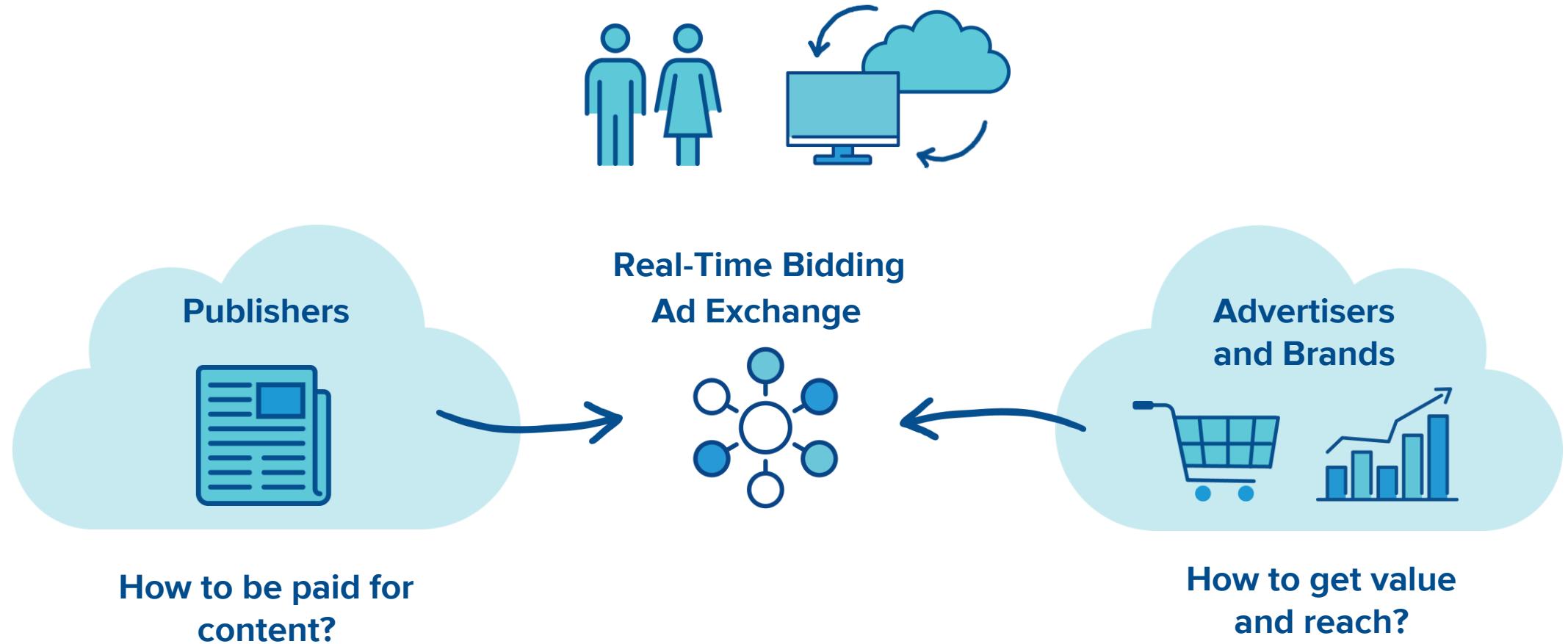


The Trade Desk's Year with Flink

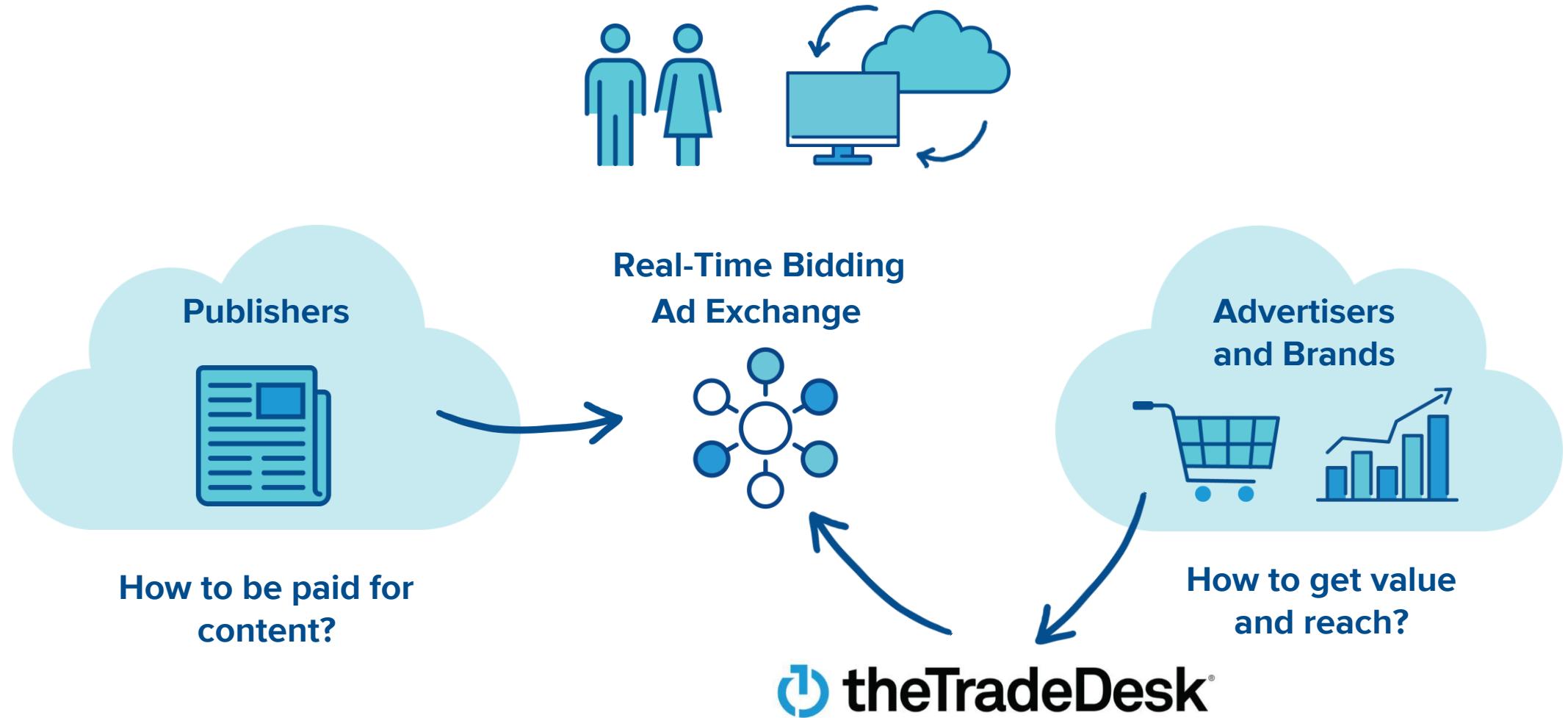
Jonathan Miles

 theTradeDesk®

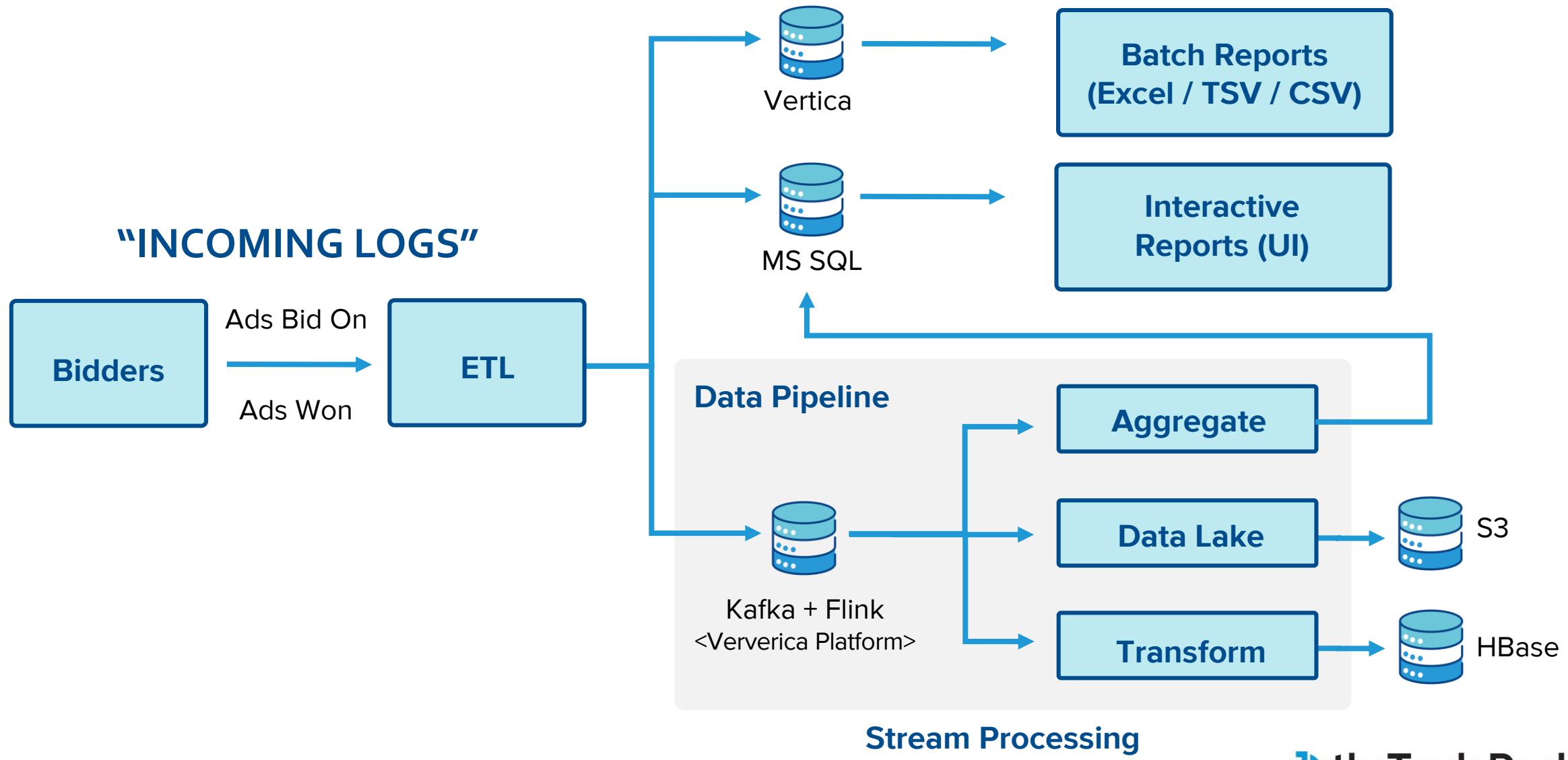
The Crux of (or Intro to) AdTech



The Crux of (or Intro to) AdTech



The Trade Desk Data Systems

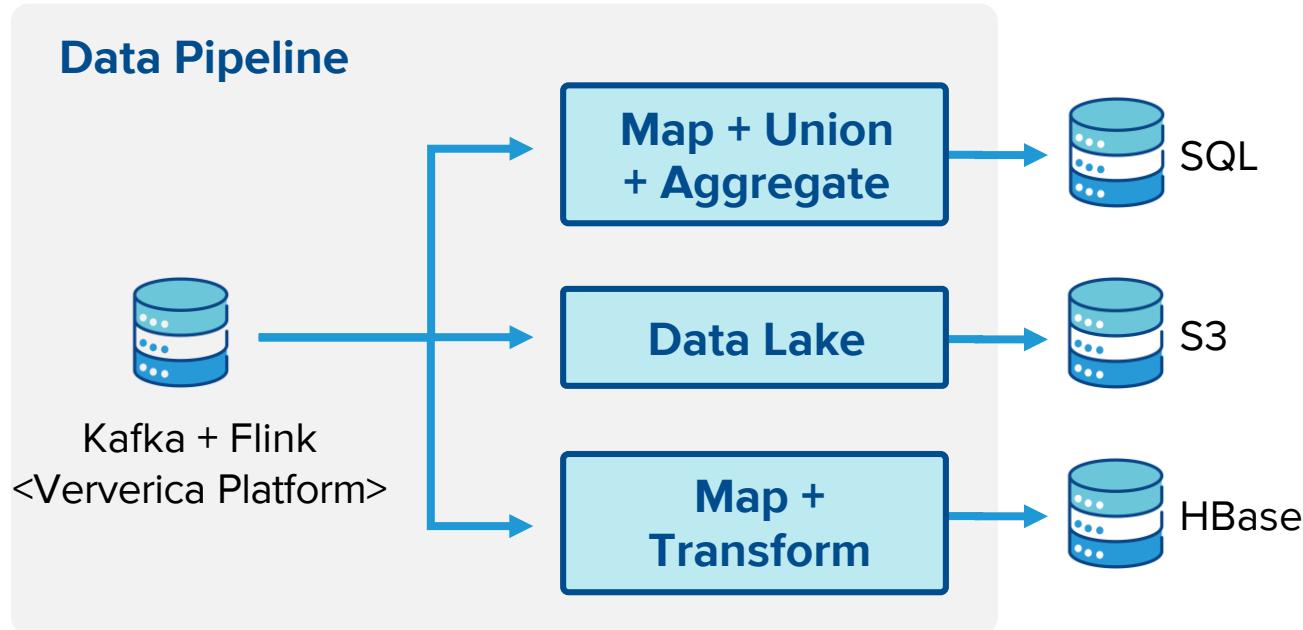




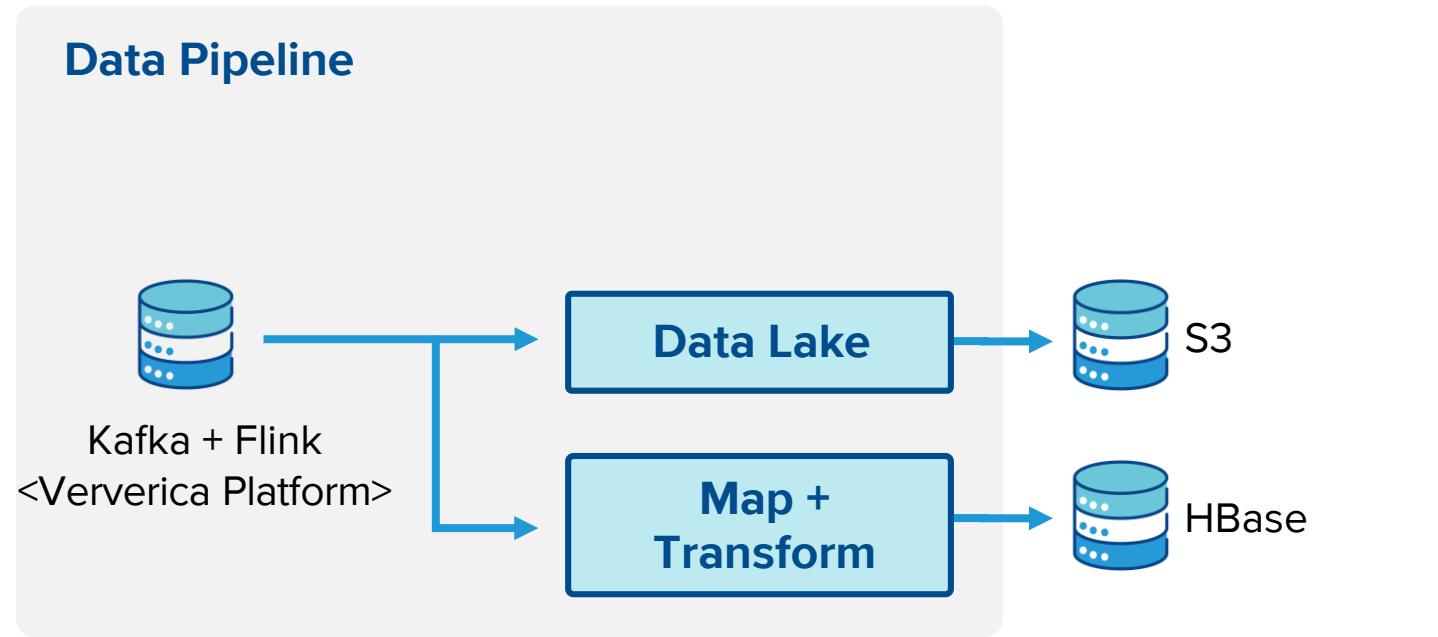
Flink Highlights

- Apache Flink
 - High-level/abstract, lower testing surface area
 - Most of the distributed systems work is done for you
 - Common sysadmin challenges
- Ververica Platform
 - Orchestration with App Manager on Kubernetes
- Ververica Support
 - When embracing open-source, The Trade Desk needed "enterprise grade" software (revenues)
 - Lean on Ververica, allowing us to work on our business
 - Gateway to Flink community

Overview of Flink jobs at TTD

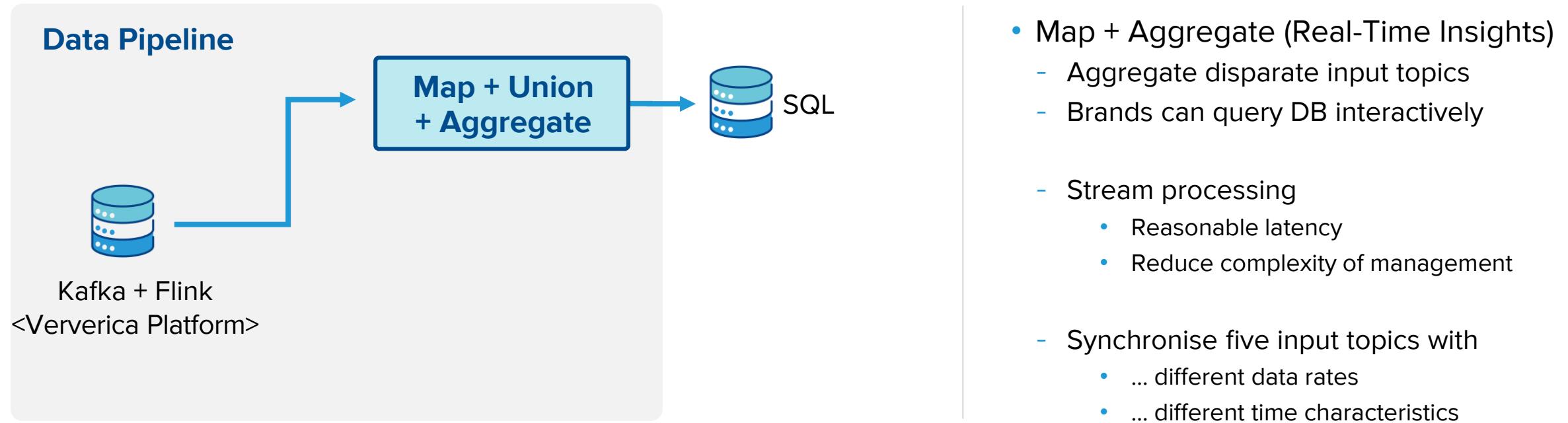


Anatomy: simpler jobs

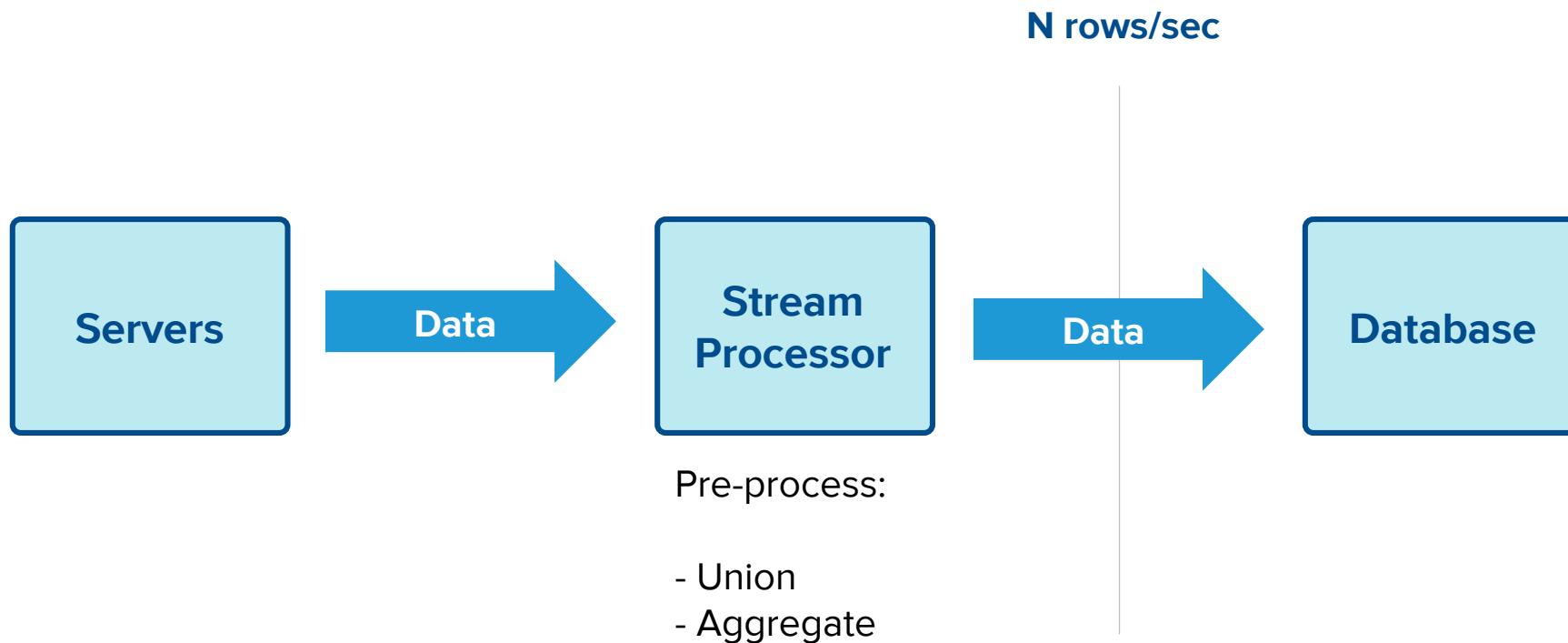


- Data Lake
 - Five jobs, one for each input topic
 - Scaled independently
 - Internal Data Science
 - ... also advertising partners
- Map + Transform
 - Two jobs, for different topics
 - Single input and output to Hbase
 - More Data Science use cases

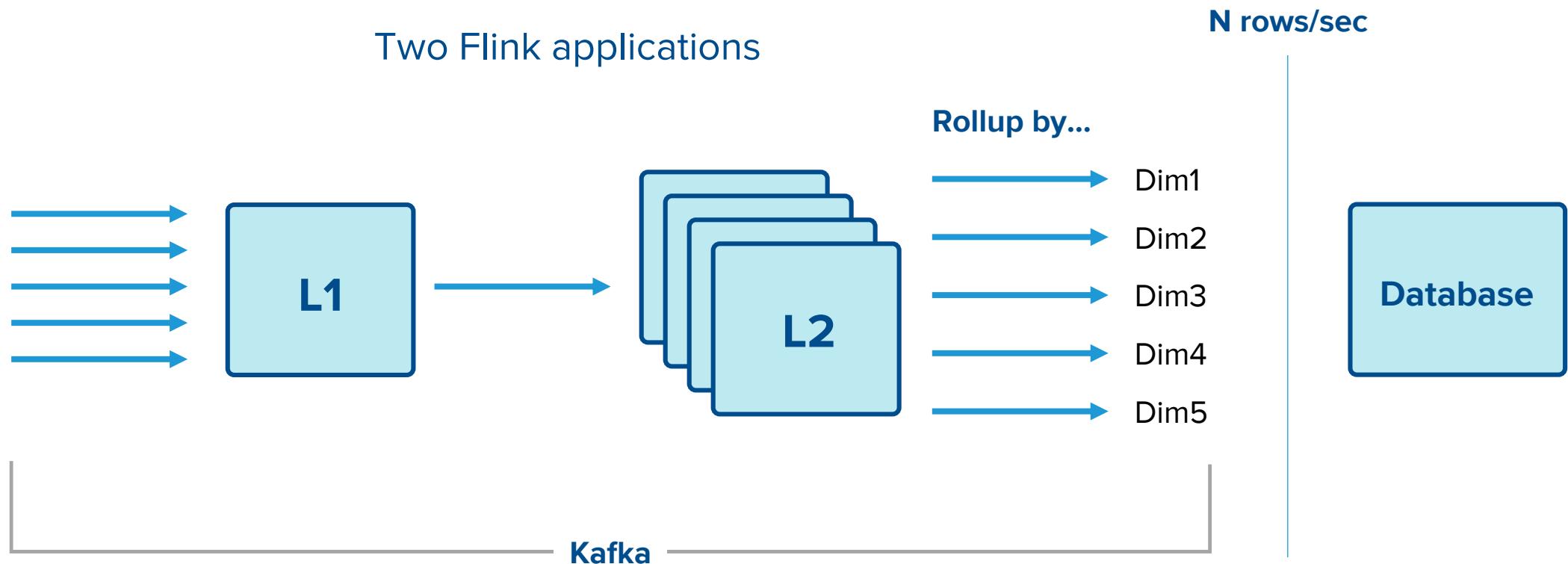
Anatomy: challenging jobs



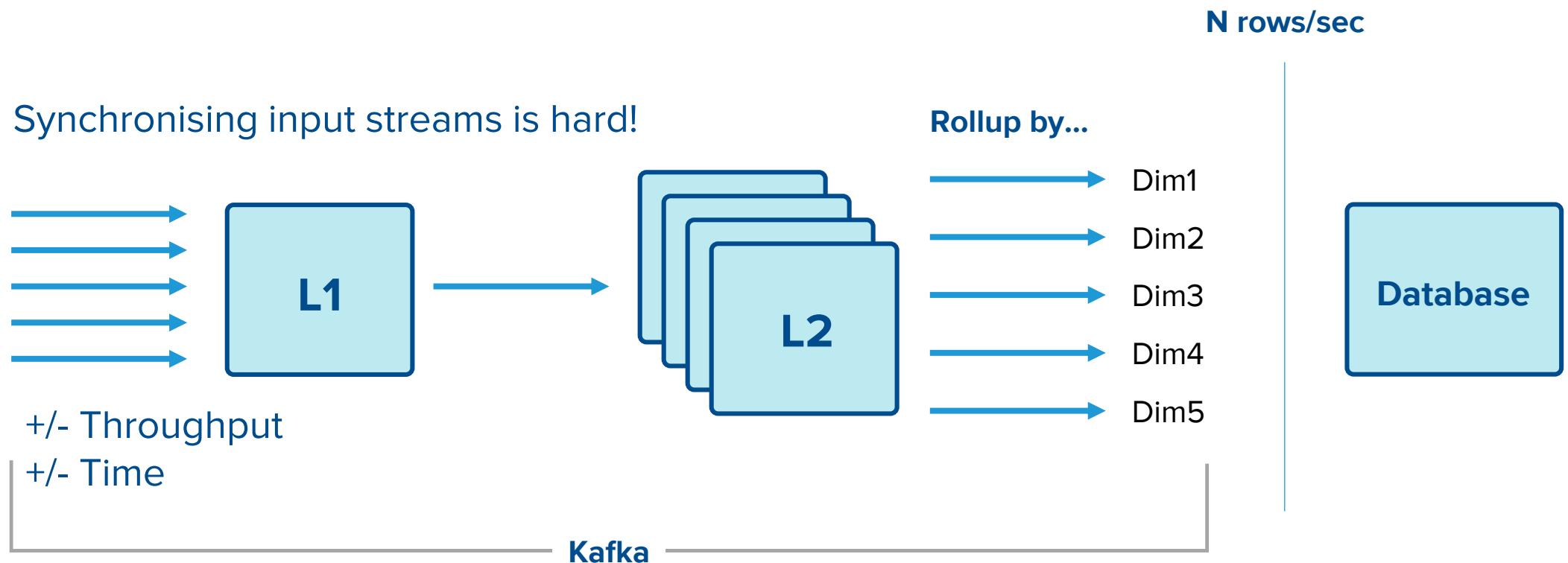
Anatomy: challenging jobs



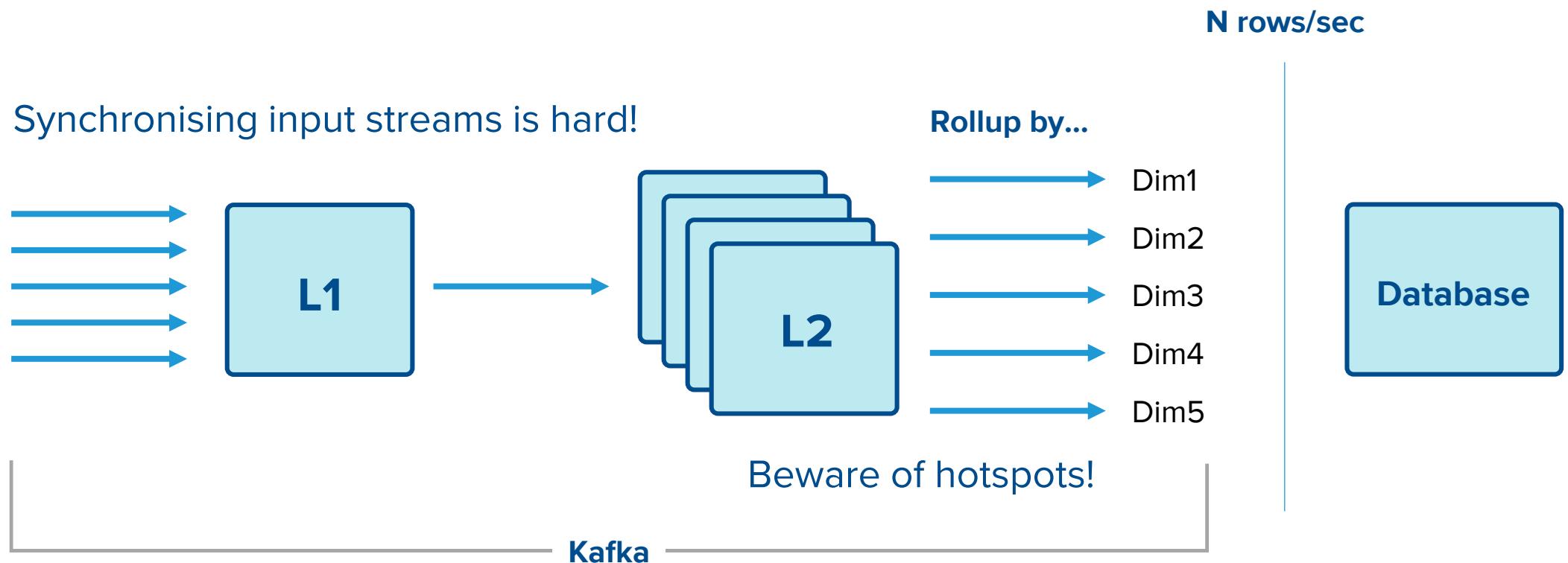
Anatomy: challenging jobs



Anatomy: challenging jobs

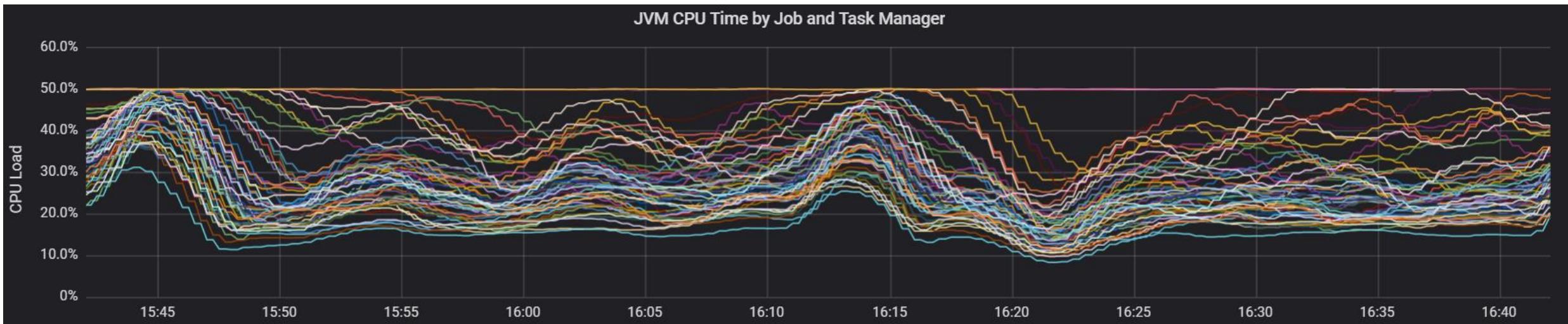


Anatomy: challenging jobs



Anatomy: challenging jobs

Hotspots...



... caused global slow-down during catchup.

Tips: synchronizing input streams is hard

Synchronising input streams is hard!



SQL Table

Key	T	I1C1..n	I2C1..n	I3C1..n	I4C1..n	I5C1..n
K1	T1	Value	Value	Value	Value	Value
K2	T2	Value	Value	Value	Value	Value

```
SELECT SUM(I1.C1), SUM(I1.C2)...
FROM Input1 I1
JOIN Input2 I2 ON (I2.key=I1.key...)
JOIN Input3 I3 ON (I3.key=I1.key...)
JOIN Input4 I4 ON (I4.key=I1.key...)
JOIN Input5 I5 ON (I5.key=I1.key...)
WHERE Input1.key = @key
GROUP BY Key, Window
```

Tips: synchronizing input streams is hard

Synchronising input streams is hard!

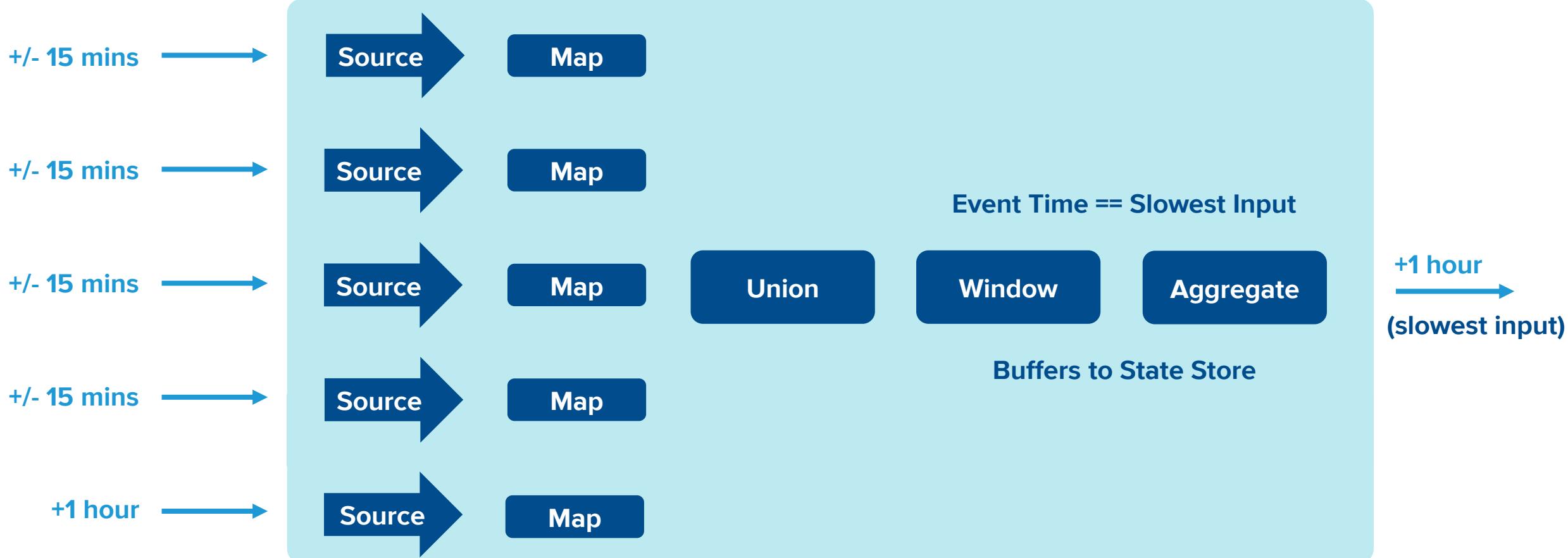


THREE CHALLENGES:

- **Time synchronisation**
- Throughput synchronisation
- Late data

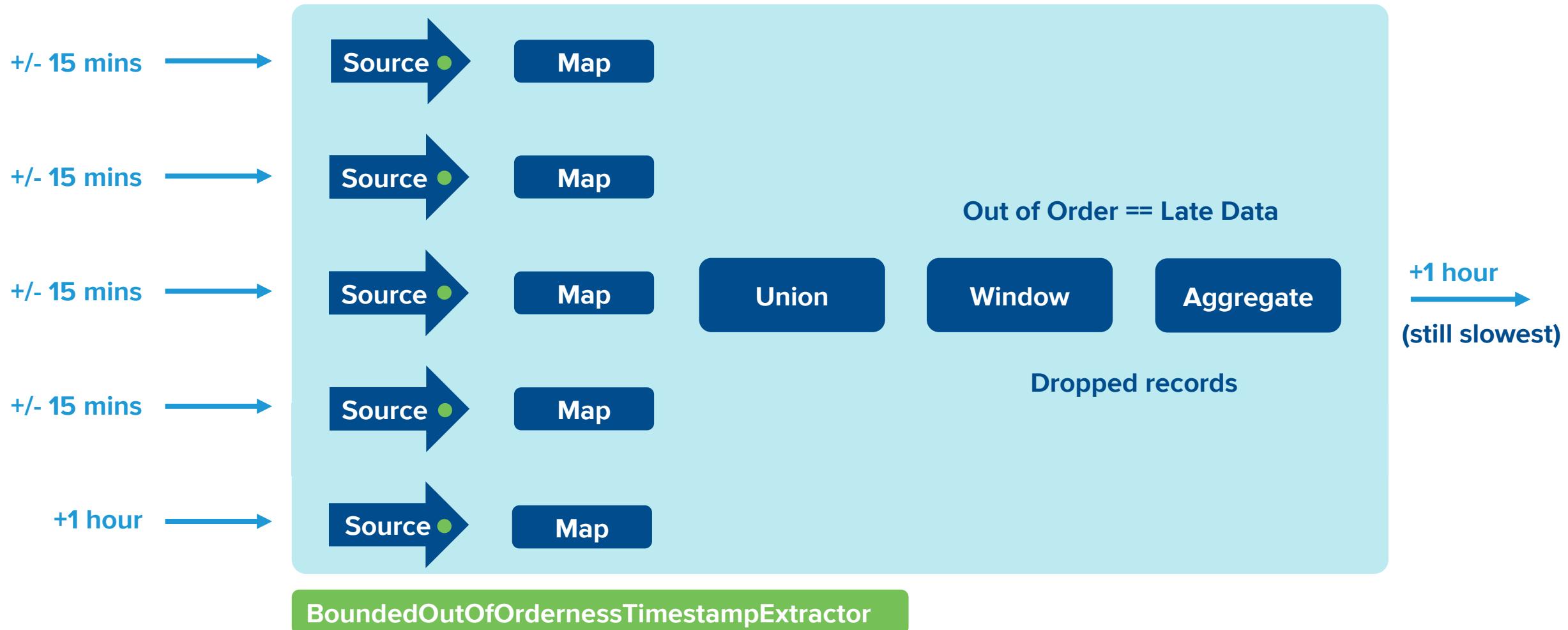
Tips: synchronizing input streams is hard

Time synchronisation



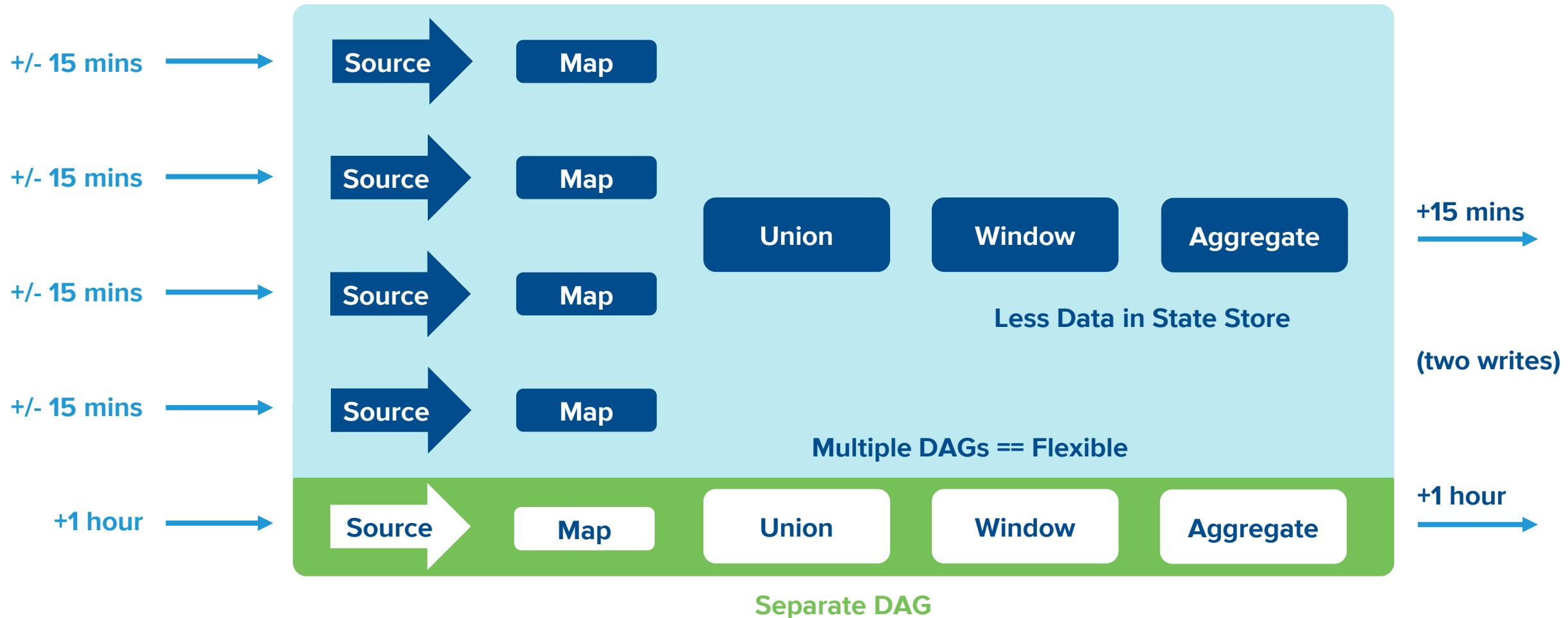
Tips: synchronizing input streams is hard

Time synchronisation (out of order)



Tips: synchronizing input streams is hard

Time synchronisation (different latency)



Tips: synchronizing input streams is hard

Synchronising input streams is hard!

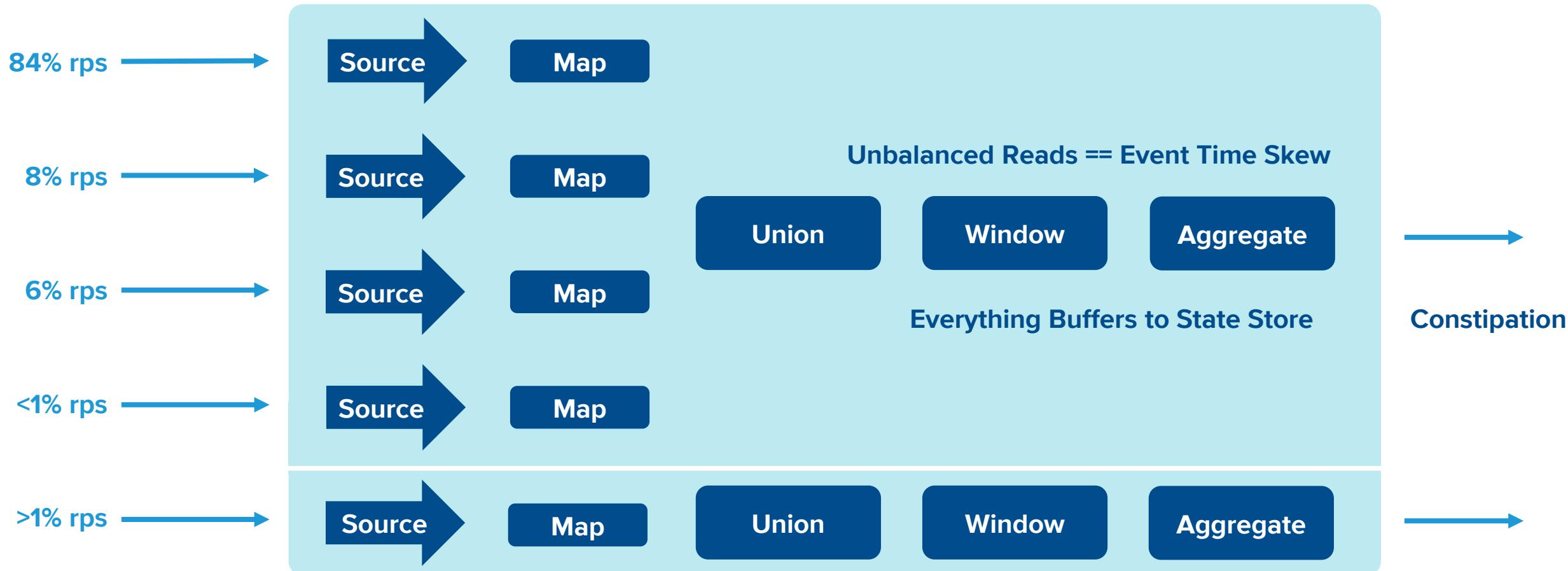


THREE CHALLENGES:

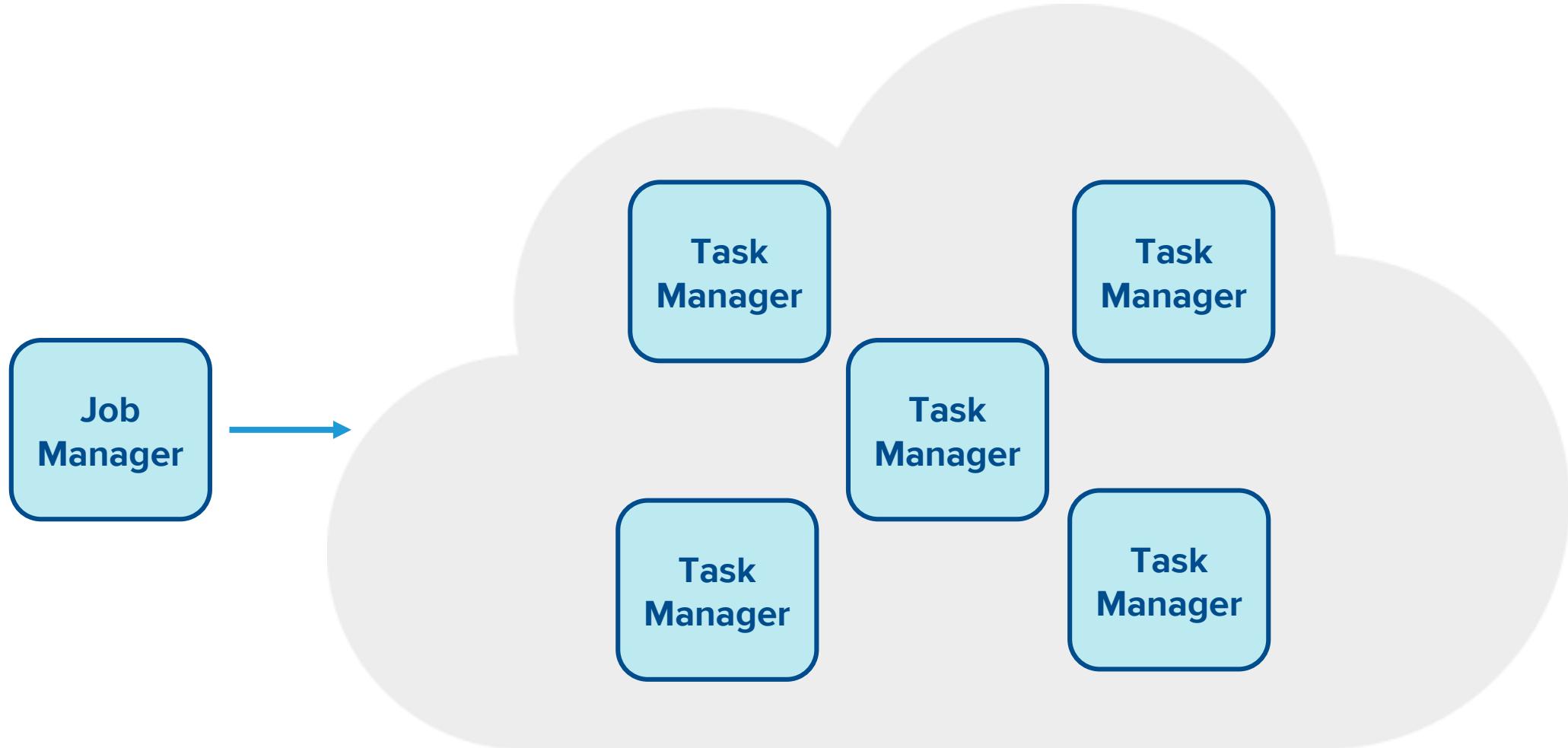
- Time synchronisation
- Throughput synchronisation
- Late data

Tips: synchronizing input streams is hard

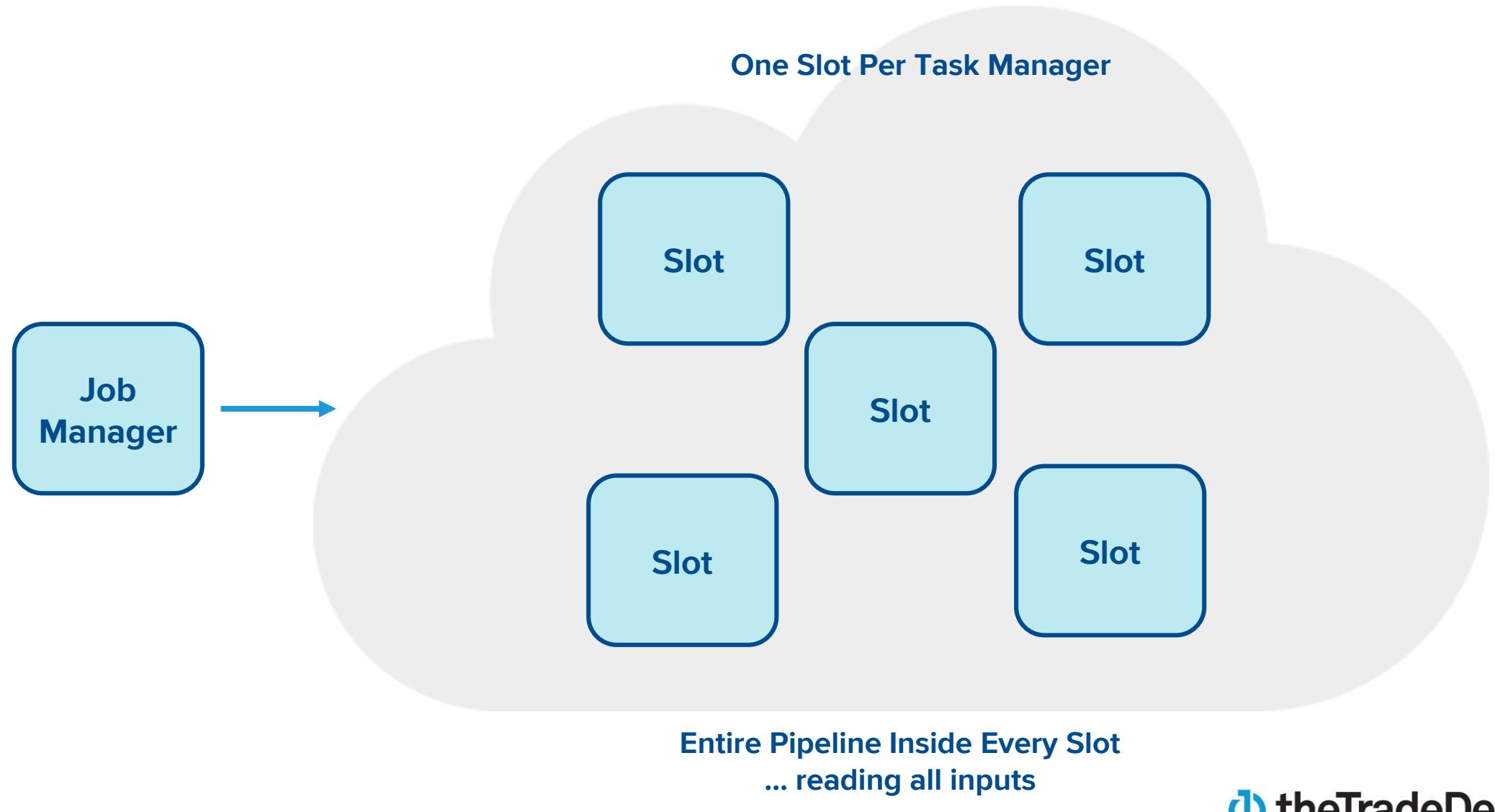
Throughput balancing



Tips: synchronizing input streams is hard



Tips: synchronizing input streams is hard



Tips: synchronizing input streams is hard

Throughput balancing

Every Slot Has These Two DAGs (e.g.)

84% rps →



8% rps →



6% rps →



<1% rps →



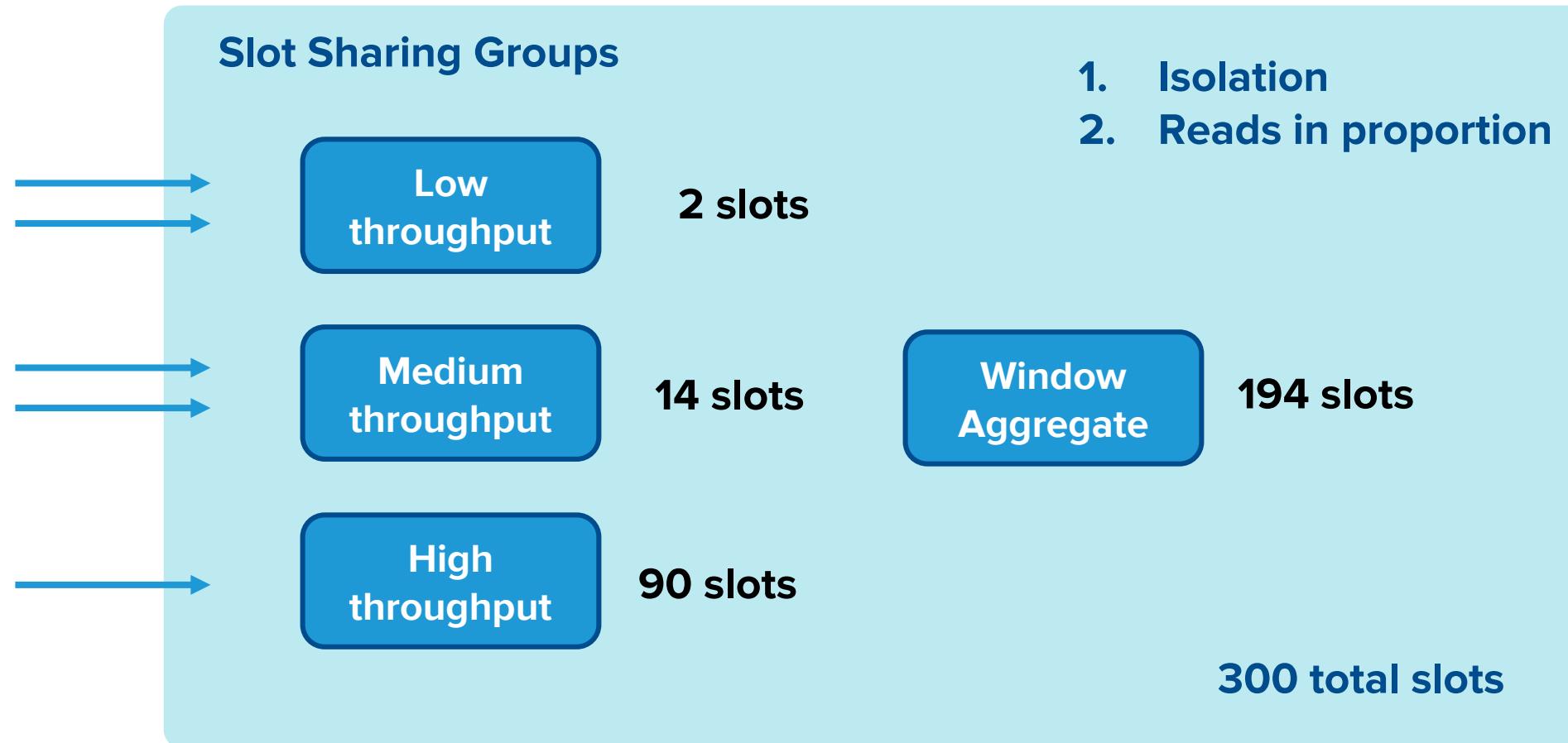
>1% rps →



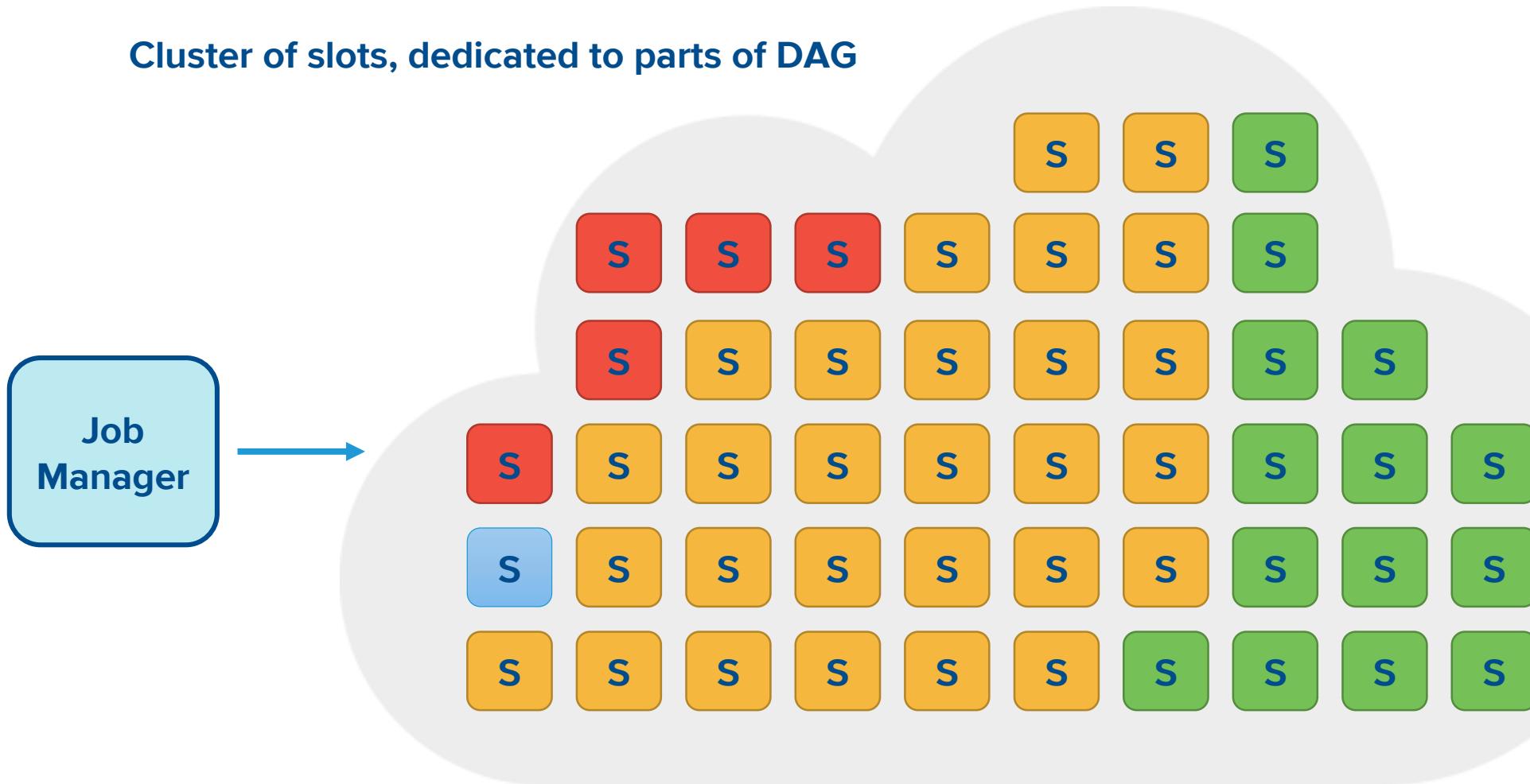
300 instances of every node in the DAG...

... leads to some being starved

Tips: synchronizing input streams is hard



Tips: synchronizing input streams is hard



... ongoing work to make this easier out of the box

Tips: synchronising input streams is hard

	Shared Slots (default)	Slot Groups
Control rate of consumption		
Easy attribution of effects		
Even distribution of resource		

Tips: synchronizing input streams is hard

Synchronising input streams is hard!

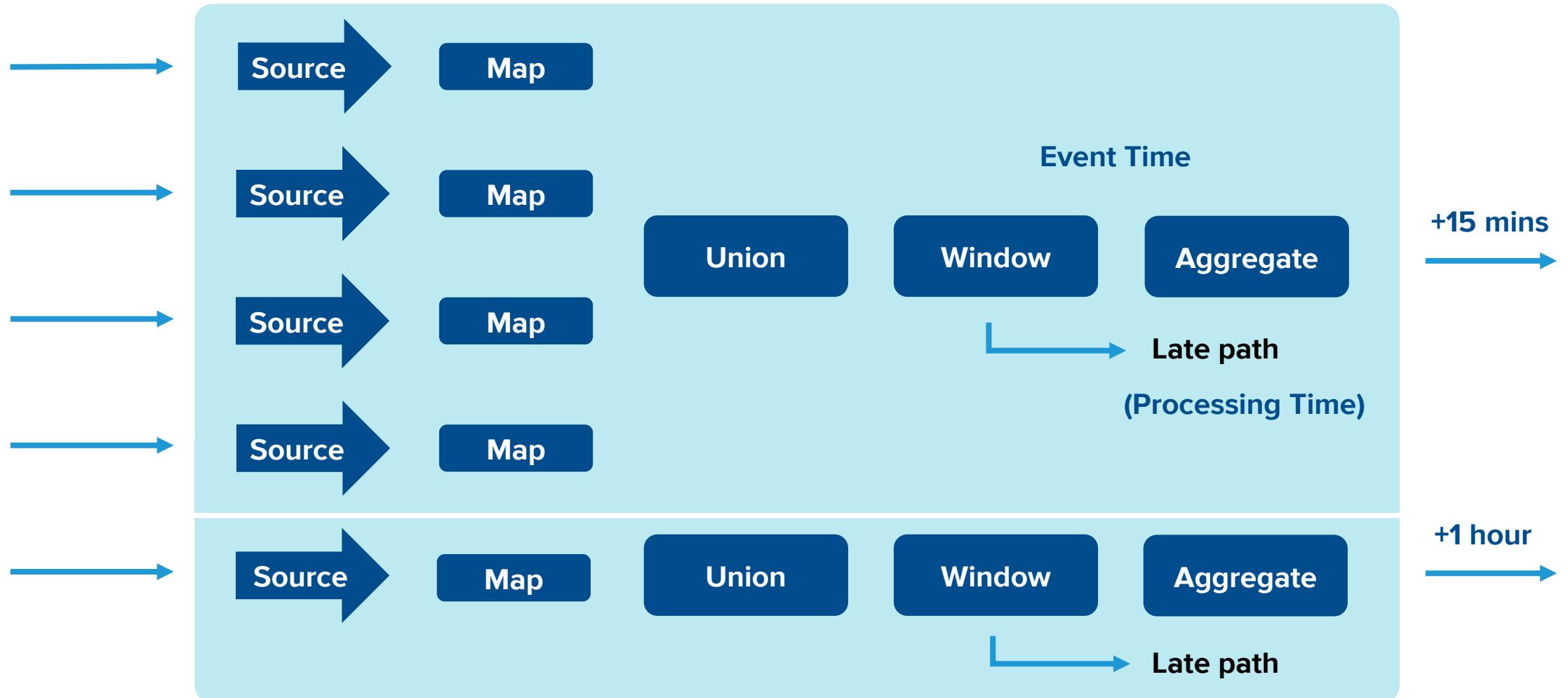


THREE CHALLENGES:

- Time synchronisation
- Throughput synchronisation
- Late data

Tips: synchronizing input streams is hard

Late Data



A photograph showing two people, a man and a woman, working on a computer. The woman is in the foreground, pointing at the screen, while the man is leaning in behind her, also looking at the screen. They appear to be in an office or workshop environment.

Tips: synchronizing input streams is hard

- Flink's flexibility and demanding use-case
- Unbalanced time
 - `BoundedOutOfOrdernessTimestampExtractor` (+/- 15 minutes)
 - Separate DAG in same job (+1 hour)
 - Actual late data shunted to separate path
- Unbalanced throughput, Flink won't balance reads for you
 - TTD: Slot Model → Slot Groups
 - Future/community: Event time synchronization across sources ([FLINK-10886](#))
- Constipation and laxative
 - Time-based distribution of individual topic
 - Processing-time eviction
- Real-time data is okay, catchup is main problem

A close-up photograph of a person's hands typing on a laptop keyboard. The person is wearing a blue and white plaid shirt. The laptop has a white backlit keyboard. The background is a light-colored wooden desk.

Future

- Why was Flink a good choice?
 - Generalised framework, well defined semantics
 - Most distributed systems problems under control
 - Flexibility
- CDC project
 - Generalised/democratised access to snapshots of database tables
- What Flink features are we excited about?
 - Event time synchronization across sources ([FLINK-10886](#))
 - Ability to take checkpoints during high backpressure (alignment sentinels stuck in same queue as messages)
- We can dream...
 - Generalised solution to hotspots problem

Plug

- Diagnostic tool, flink-diag.py
 - GitHub?
 - Come talk to us?
- We're hiring
 - Come change the world of advertising with us!
- Questions?
 - Many people from The Trade Desk attending today
 - Feel free to come talk to us!



Thank you



Managing Flink on Kubernetes

Anand Swaminathan (@anand12100)

Ketan Umare (@ketanumare)

April 2, 2019



Agenda

1

Kubernetes Primer

Quick Introduction to concepts in Kubernetes

2

Background

Summary of Lyft's legacy Flink Deployment

3

Solution

Flink Kubernetes Operator

4

Demo

5

Ecosystem

6

Roadmap

About us



Kubernetes Primer

History

- Google's internal infrastructure is containerized and runs on Borg/Omega
- K8s was open sourced in 2014, re-incarnation of the internal infrastructure
- Kubernetes automates - deployment, scaling and management of containerized apps.
- Containers are scheduled based on CPU/GPU/Memory/Disk etc

Production-Grade Container Scheduling and Management <https://kubernetes.io>

kubernetes

go

cncf

containers

 76,018 commits

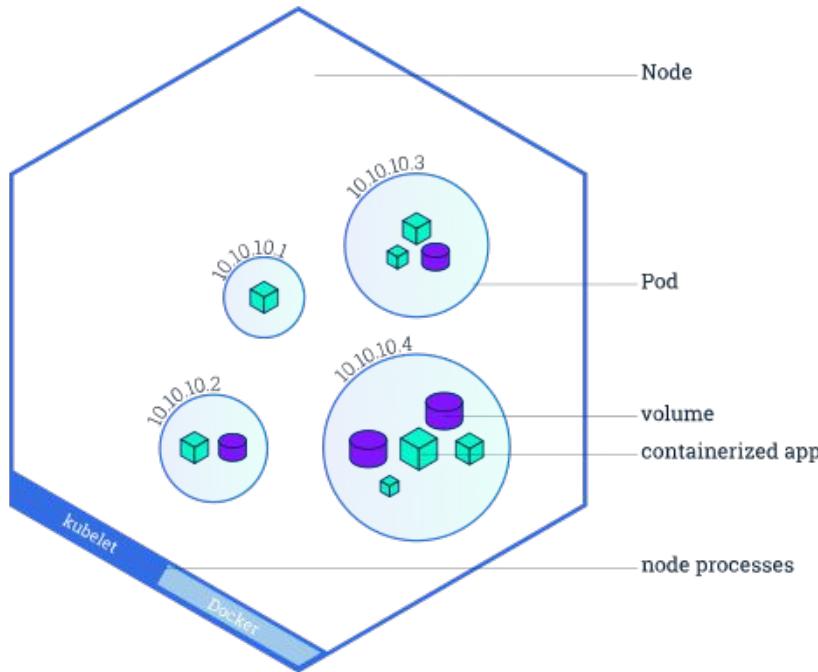
 40 branches

 489 releases

 2,048 contributors

Kubernetes Primer

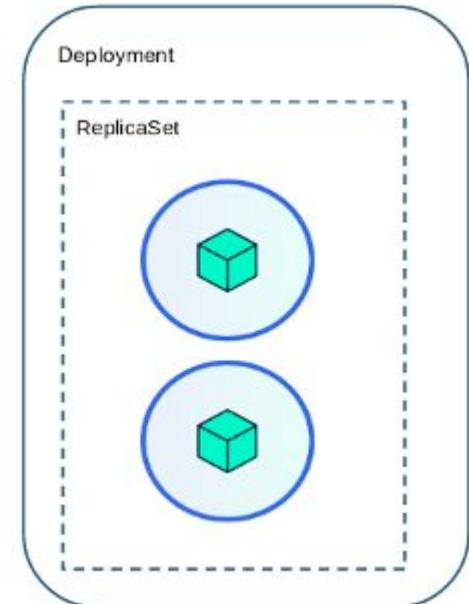
Pods



- A Pod is a group of **one or more Containers** as one unit
- Pods have **no durability** guarantees
- Each Pod has a **unique IP Address**
- **Containers** in a Pod can communicate using **localhost**
- **Multiple Pods** can be located on the **same node** - machine

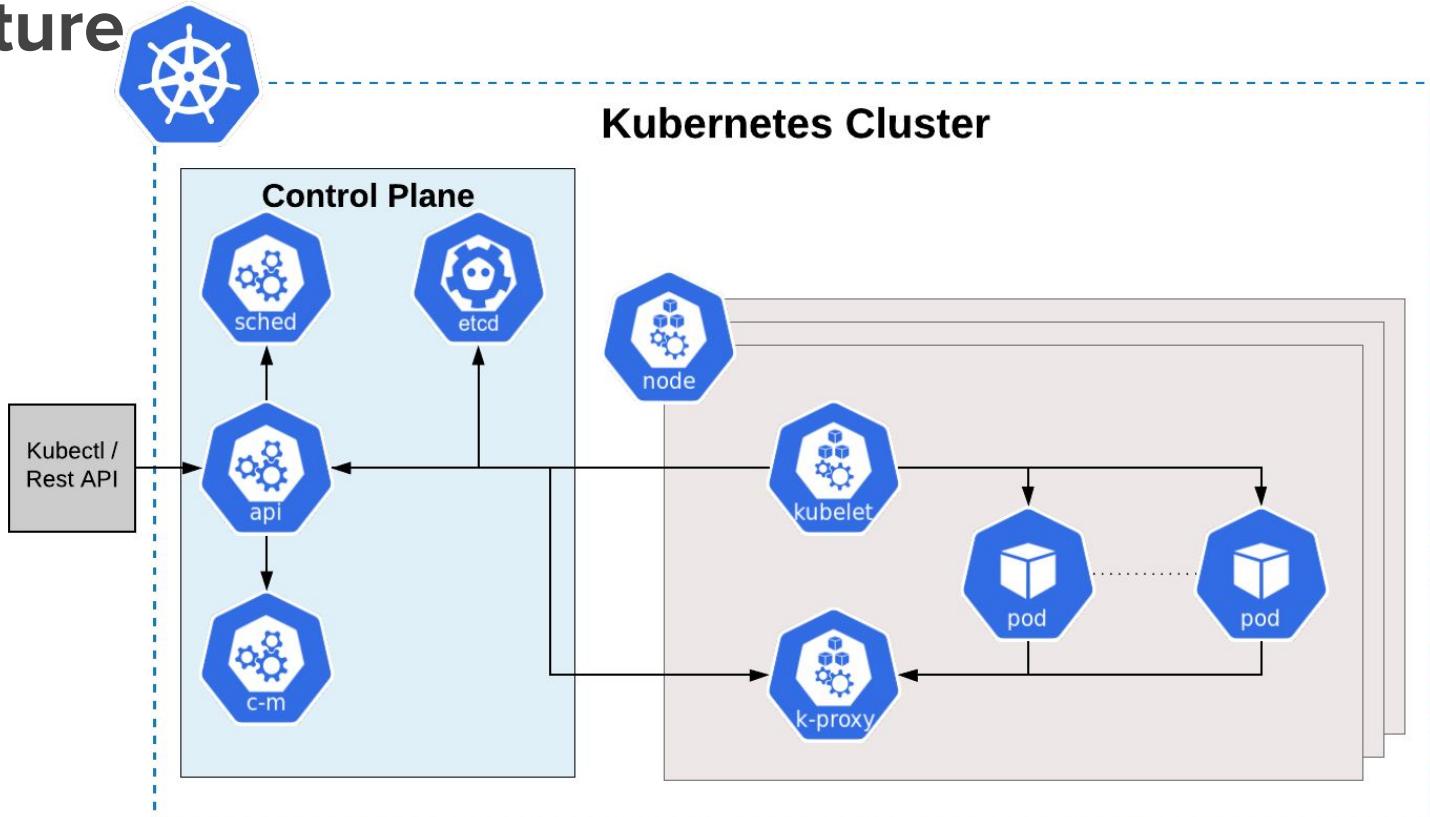
Other Concepts

- **Deployments** abstraction that enables rolling out changes to a set of pods
- **Service** abstraction to access sets of pods - like a load balancer within a k8s cluster
- **Ingress** abstraction to expose a service to the outside world (HTTP/HTTPS)
- **Controller** A reconciliation loop that drives current state towards desired state



Kubernetes Primer

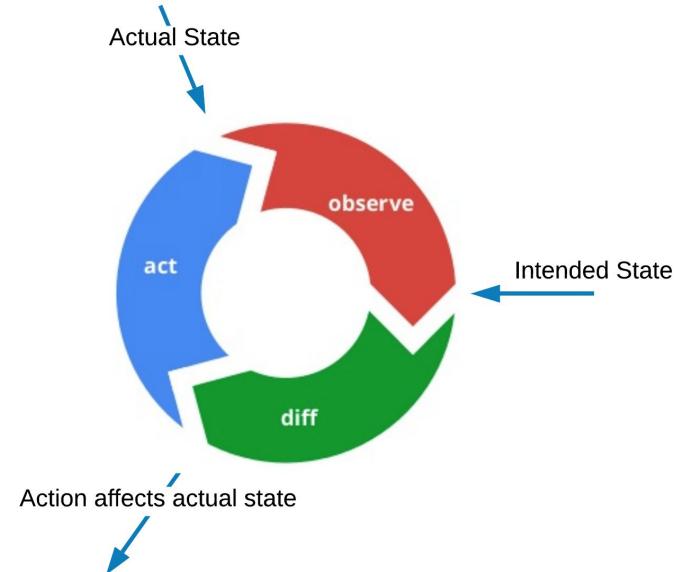
Architecture



Kubernetes Primer

Control Loops

- **Control loops** are fundamental building block industrial control systems
- **Desired State** refers to the intended state as requested
- **Current/Observed State** is the state of the system as **observed** by the controller
- Controller runs control loops
- Drive **Current State -> Desired State**
- This is the cornerstone of Kubernetes

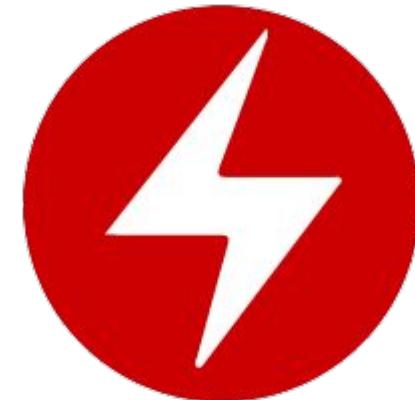


Custom Resources

- Custom Resource Definitions (**CRD**) allow **extending** Kubernetes API
- Custom resources are **optional** extensions
- Custom resources can be **added**/removed **dynamically**
- They can be manipulated using known tools - **kubectl** & kube clients
- State stored in **etcd**
- Custom control loops (**controllers**) are used to manage the state of the resource.
- CRD is essentially the **desired state**.

Operators

- Controller + CRD = Kubernetes Operator
- Term coined by **CoreOS - 2017**
- **Manages a complex applications** lifecycle on Kubernetes.
- Core library to author operators @
[SIG/controller-runtime](#)



Background

OK how does this relate

- @Lyft we started working on **Flyte** - a modern take at Pipelines/Workflows
- Orchestration is pervasive **throughout various sectors** of our Industry
 - Machine learning
 - Data engineering and processing
 - ETL
- Kubernetes has a solution to many of our problems
 - Deployment, Versioning, cluster management etc
- In parallel **Streaming Platform** started working on Flink for streaming applications

Background

Legacy deployment of Flink @Lyft

- Hosted on **AWS**
- Separate **AutoScalingGroups** for Task Managers and Job Managers
- Machines provisioned and bootstrapped by **SaltStack**
- Every deployment needs **provisioning of machines**
- Users started running multiple jobs in the same Flink Cluster
- **Multi-tenancy hell !**

Introducing Flink-k8s-operator

Goals

- Abstract out the complexity from application developers
 - **Hosting**
 - **Configuration**
 - **Management**
- Separate Flink cluster for each Flink application.
- **Deploy and rollback** support
- Support Flink application updates - **scaling**
- Simplified interface for instituting best practices
- **Scale to 100s** of flink applications

Solution

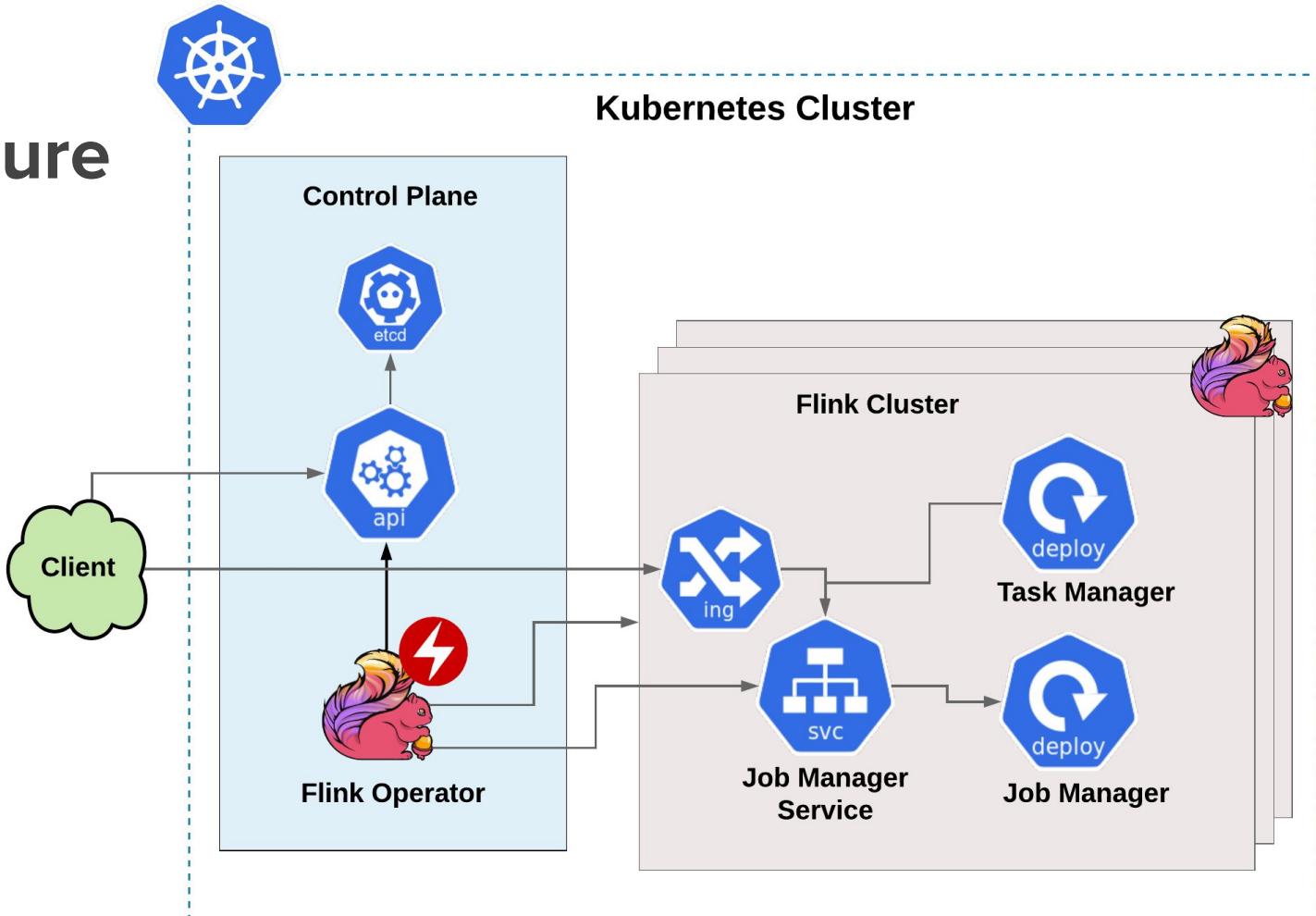
Flink Operator - CRD

- Each custom resource corresponds to a Flink application
- Each Flink application runs a single **Flink job**
- **Docker image** should be runnable

```
apiVersion: flink.k8s.io/v1alpha1
kind: FlinkApplication
metadata:
  name: flink-speeds-working-stats
  namespace: flink
  annotations:
    iam.amazonaws.com/role: 'arn:aws:iam::100:role/abc-iad'
  labels:
    app: app-name
    environment: staging
spec:
  image: '100.dkr.ecr.us-east-1.amazonaws.com/abc:xyz'
  flinkJob:
    jarName: name.jar
    parallelism: 10
  deploymentMode: Single
```

Solution

Architecture

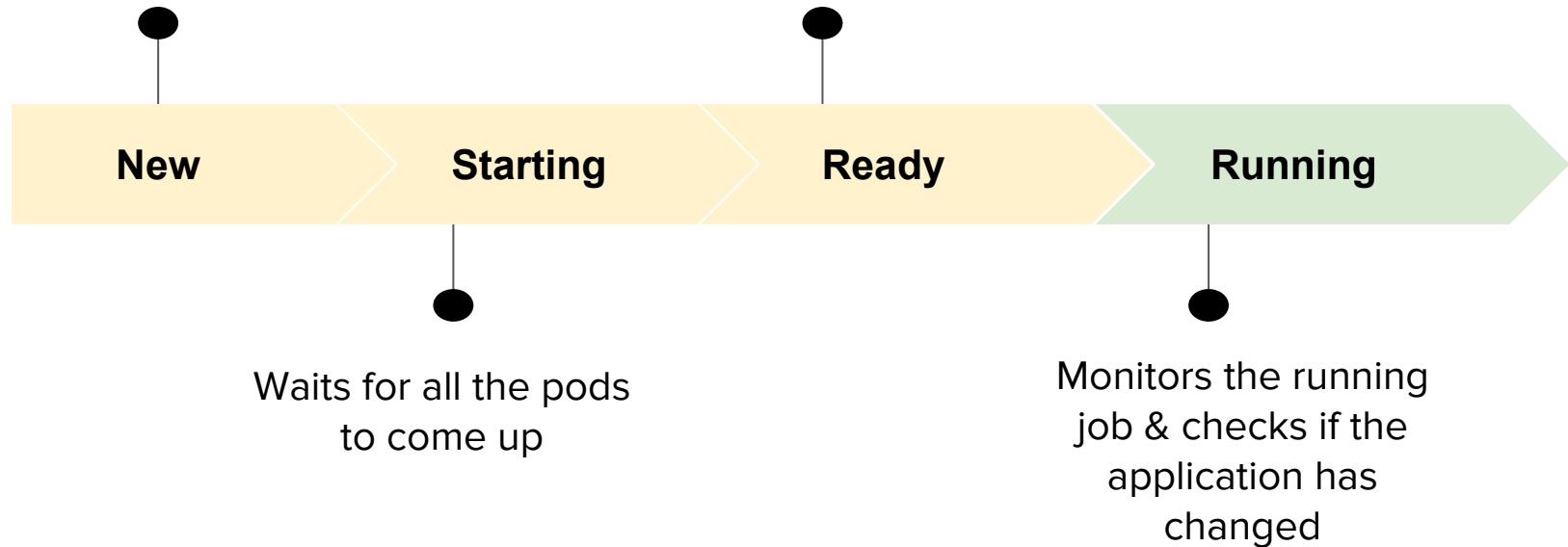


Solution

Operator Walkthrough

Creates a new Flink cluster in K8s

Polls Flink jobmanager REST API & submits a new job

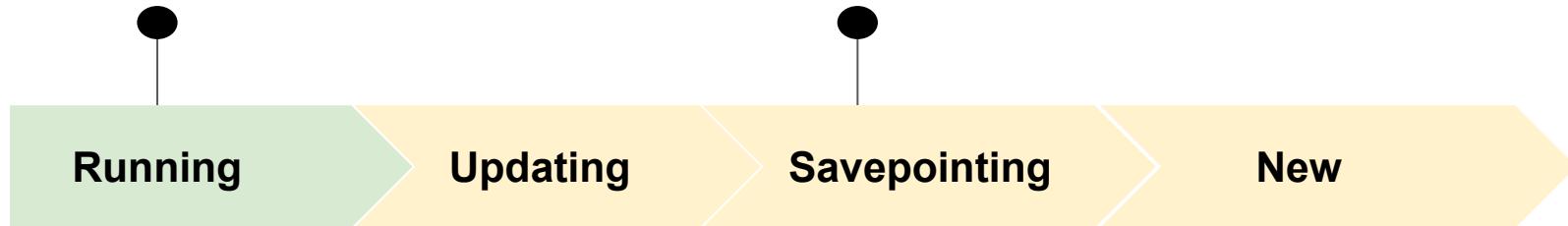


Solution

Operator Walkthrough

Operator
detects the
update to CRD

Waits for the savepoint
to succeed, and
updates savepoint
location in CRD



If needed,
updates cluster,
cancels Job with
savepoint

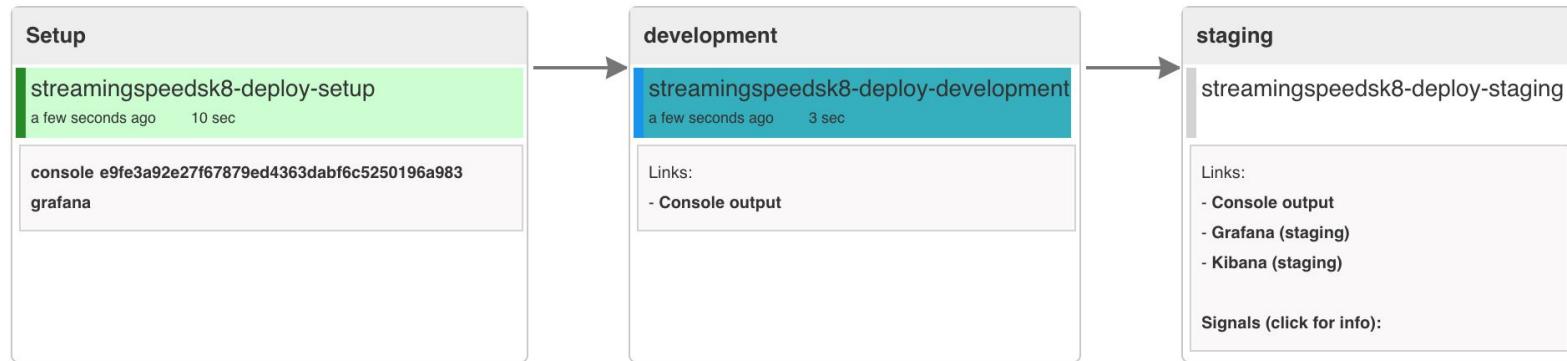
Brings up a new
cluster and tries to
transition to **Running**

Demo

Ecosystem

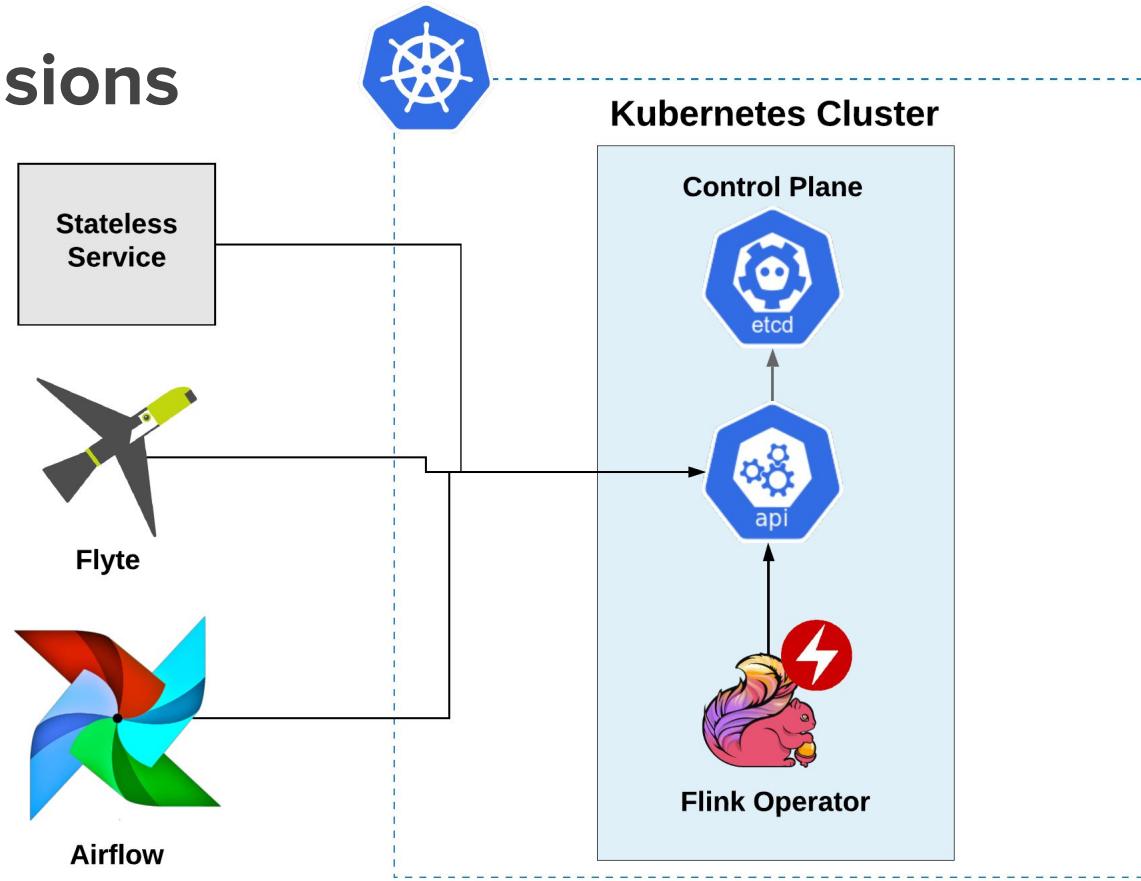
Deployment @Lyft

- Jenkins based deployment
- Each stage creates or updates the resource in Kubernetes



Ecosystem

Future Extensions



Roadmap

Open Source

- **Last week of April***
- Project status: **Alpha**
- @Lyft:
 - Active development and testing in staging.
- Future
 - Flink Job failure handling
 - Tooling to manage CRD

Coming soon: <https://github.com/lyft/flinkk8soperator>

We're Hiring! Apply at www.lyft.com/careers

Data Engineering

Engineering Manager
San Francisco

Software Engineer
San Francisco, Seattle, &
New York City

Data Infrastructure

Engineering Manager
San Francisco

Software Engineer
San Francisco & Seattle

Experimentation

Software Engineer
San Francisco

Observability

Software Engineer
San Francisco

Streaming

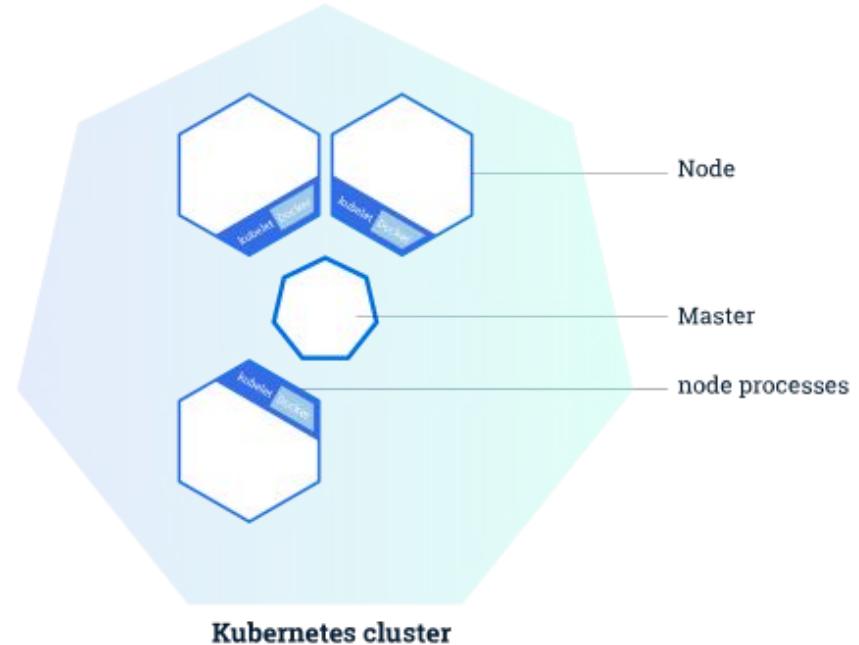
Software Engineer
San Francisco

**Thank you
Questions please!**

Background

Example of Deployment

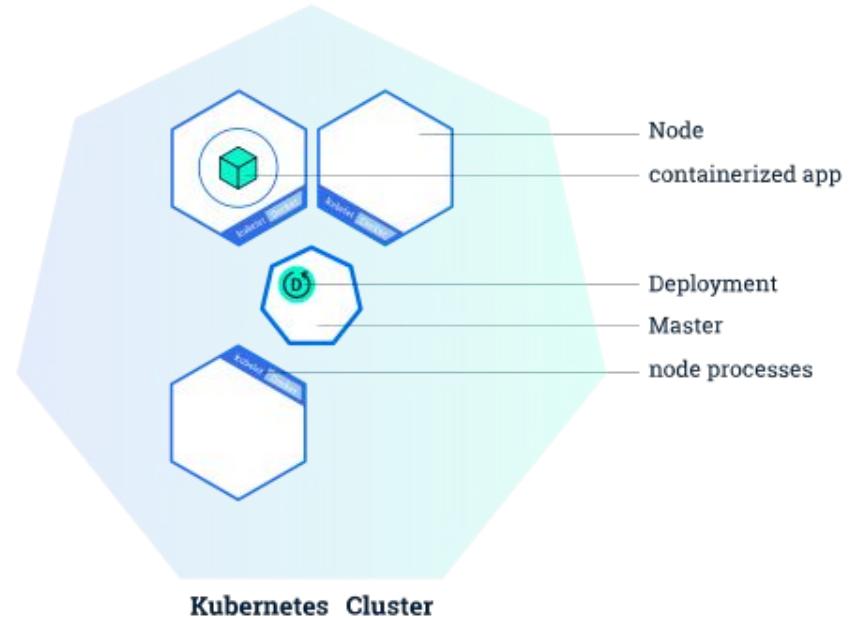
- User requests for a **Deployment** @ master
- Master accepts the request
- **Desired State:** 1 Pod running
- **Current State:** 0 Pods running



Background

Kubernetes 101

1. Master requests **Pod creation**
 - o Current State: Deployment unhealthy
2. Master receives **pod created** event
 - o Current State: Deployment healthy
3. Now if the pod crashes/dies etc
 - o Current State: Deployment unhealthy
4. Goto 1



High Cardinality Data Stream Processing with Large States

Ning Shi, Klaviyo



Help ecommerce businesses grow



- Real-time analytics on consumer events
- Event-triggered actions based on analytics
- Best-in-class email marketing

Use Case

- Thousands of types of events
- Close to 100,000 events per second
- *“How many unique individuals opened emails from this campaign between 3:00am-4:00am?”*
- *“How many unique individuals purchased at least two black iPhone chargers after reading this campaign email yesterday?”*

```
{  
  "email": "john@example.com",  
  "message_id": "ABCDE",  
  "timestamp": 1544153562,  
  "ip": "127.0.0.1",  
  "browser": "Safari 12.0.1"  
}
```

Dimension Name	Dimension Value
provider	example.com
message_id	ABCDE
campaign	Holiday Sale!

browser	Safari 12.0.1

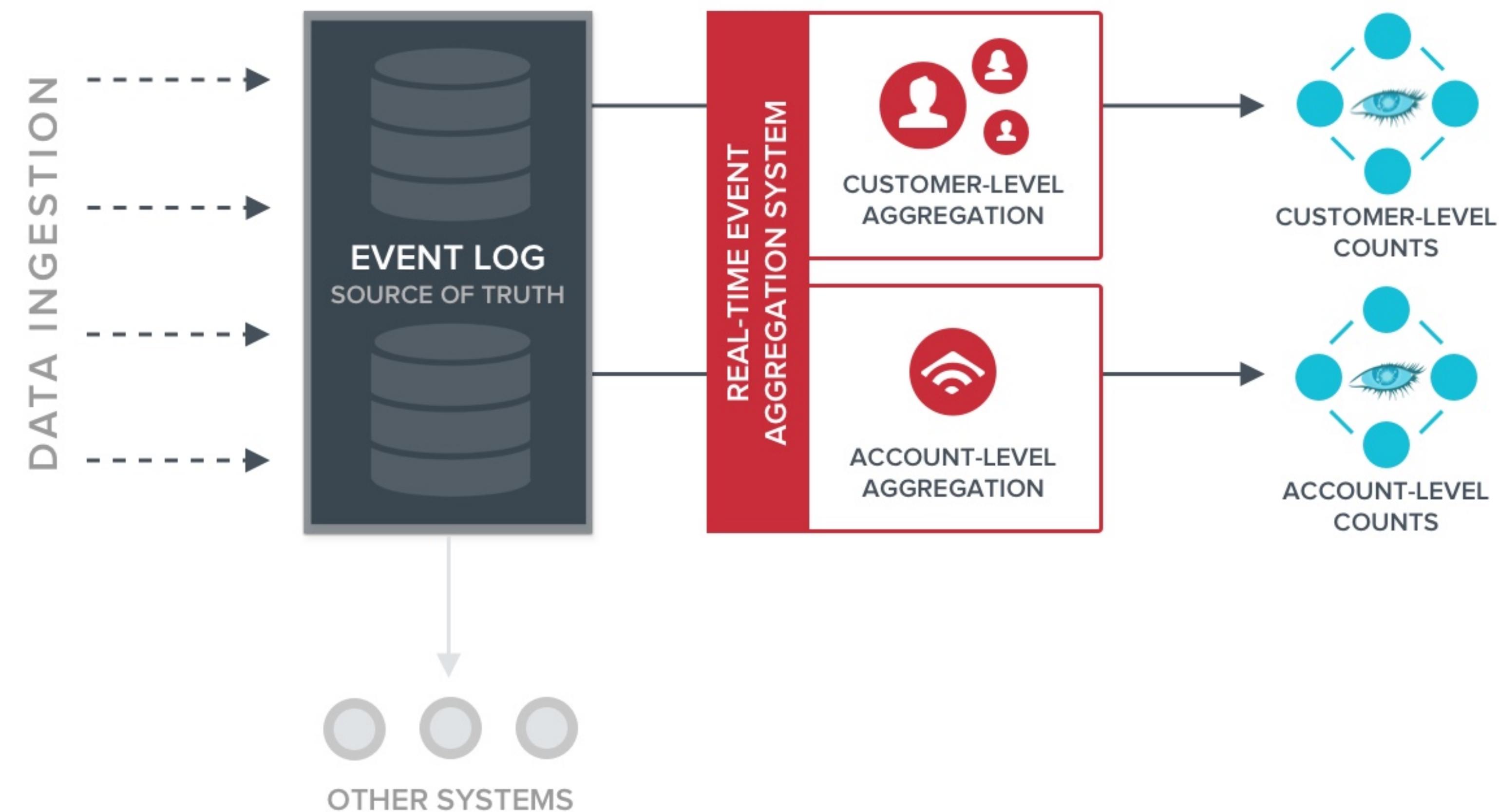


Technical Challenges

- Over 1 billion user profiles
- Large state (1.5TB compressed) for duplicate detection
- High fan-out ratio (one to hundreds)
- Millions of metrics to aggregate per second

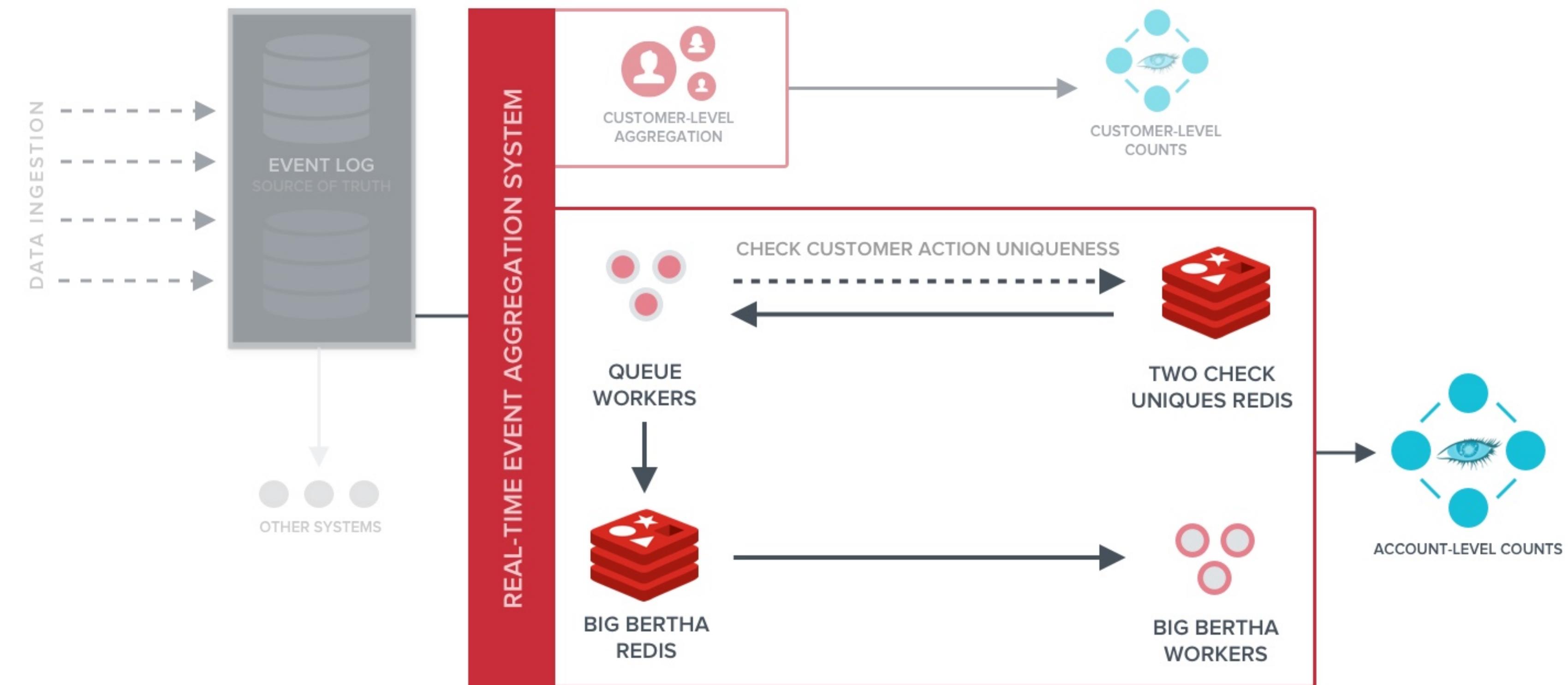


Event Processing Pipeline



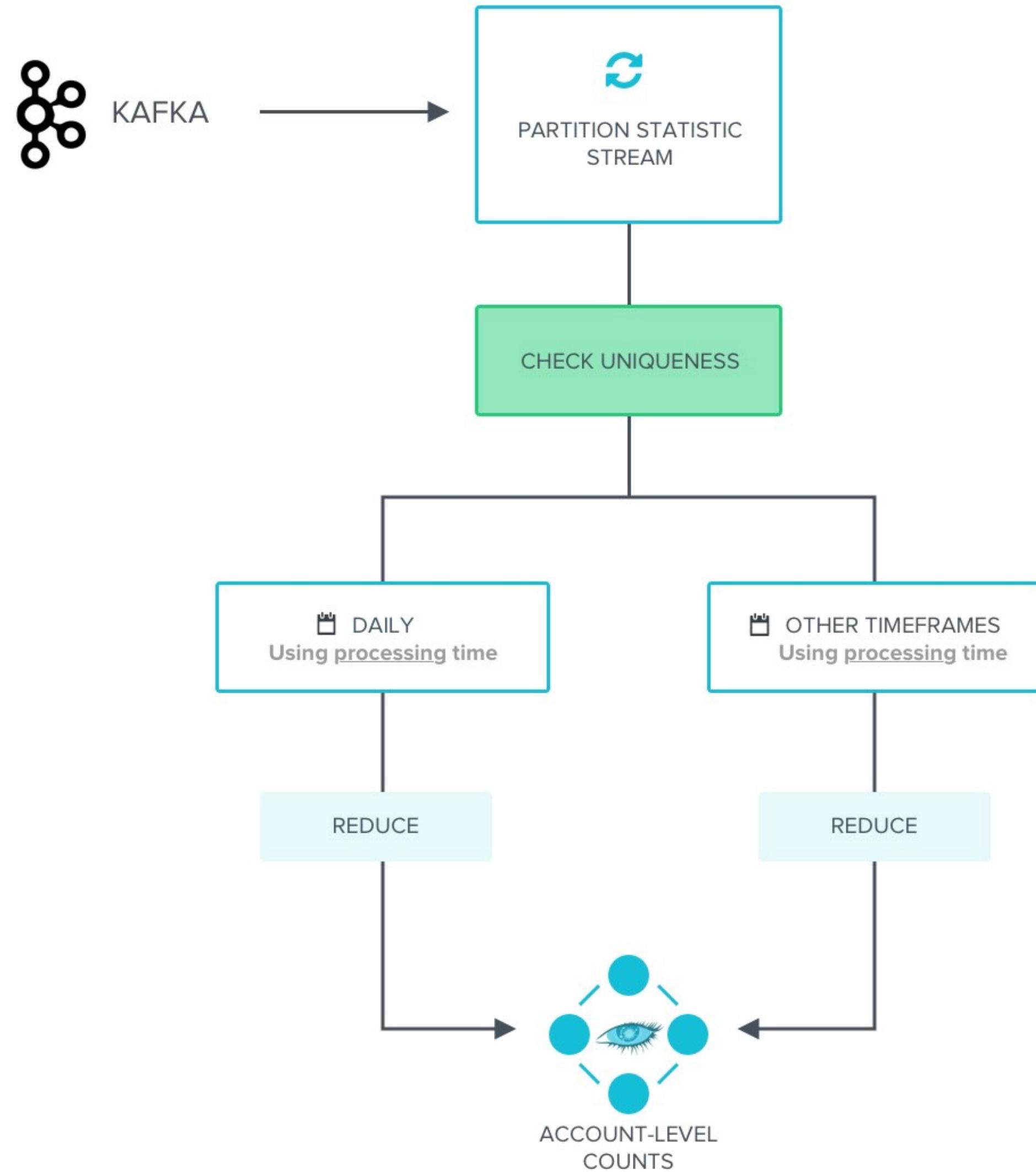
Initial Version

- Custom Python code
- Nondeterministic
- Too many moving parts



Abacus

- Apache Flink
- Exactly-once processing
- Easy to scale
- Significantly better performance (90% reduction of EC2 instances)



Lessons Learned

- Code changes
- Configuration



Code Changes



Time

Processing Time

- Wall clock time on each TaskManager
- Nondeterministic
- Different on each TaskManager

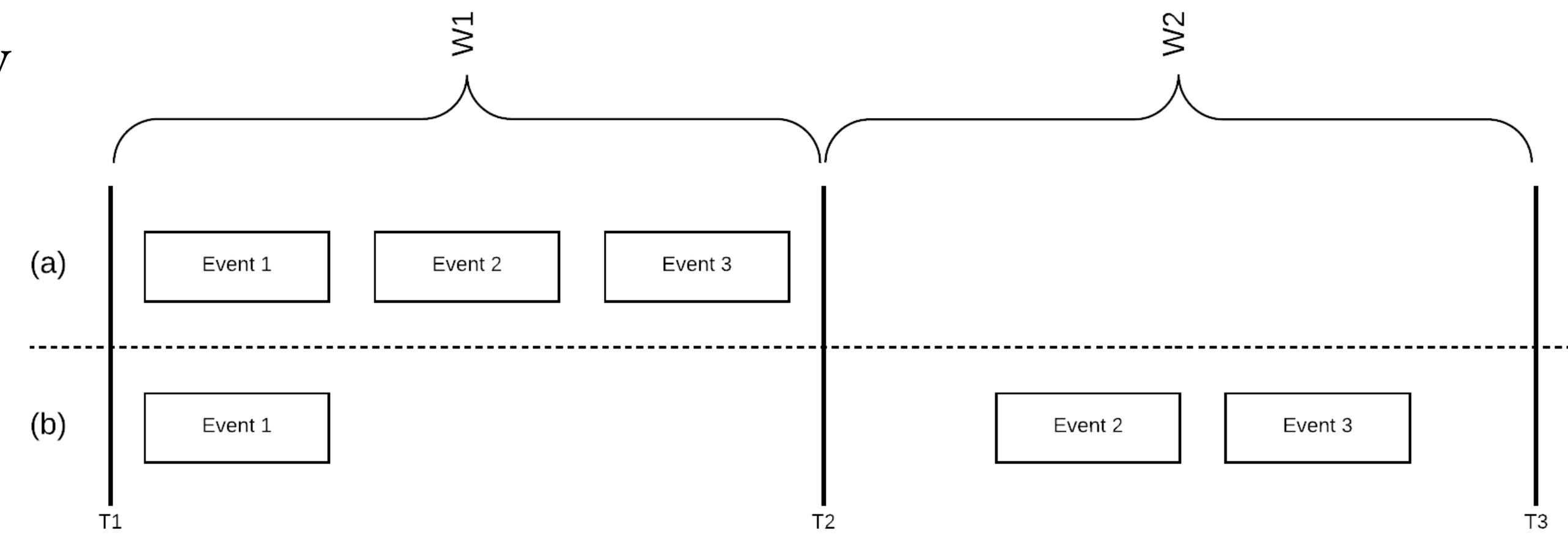
Event Time

- Intrinsic to events
- Deterministic
- Synchronized across cluster



Processing Time

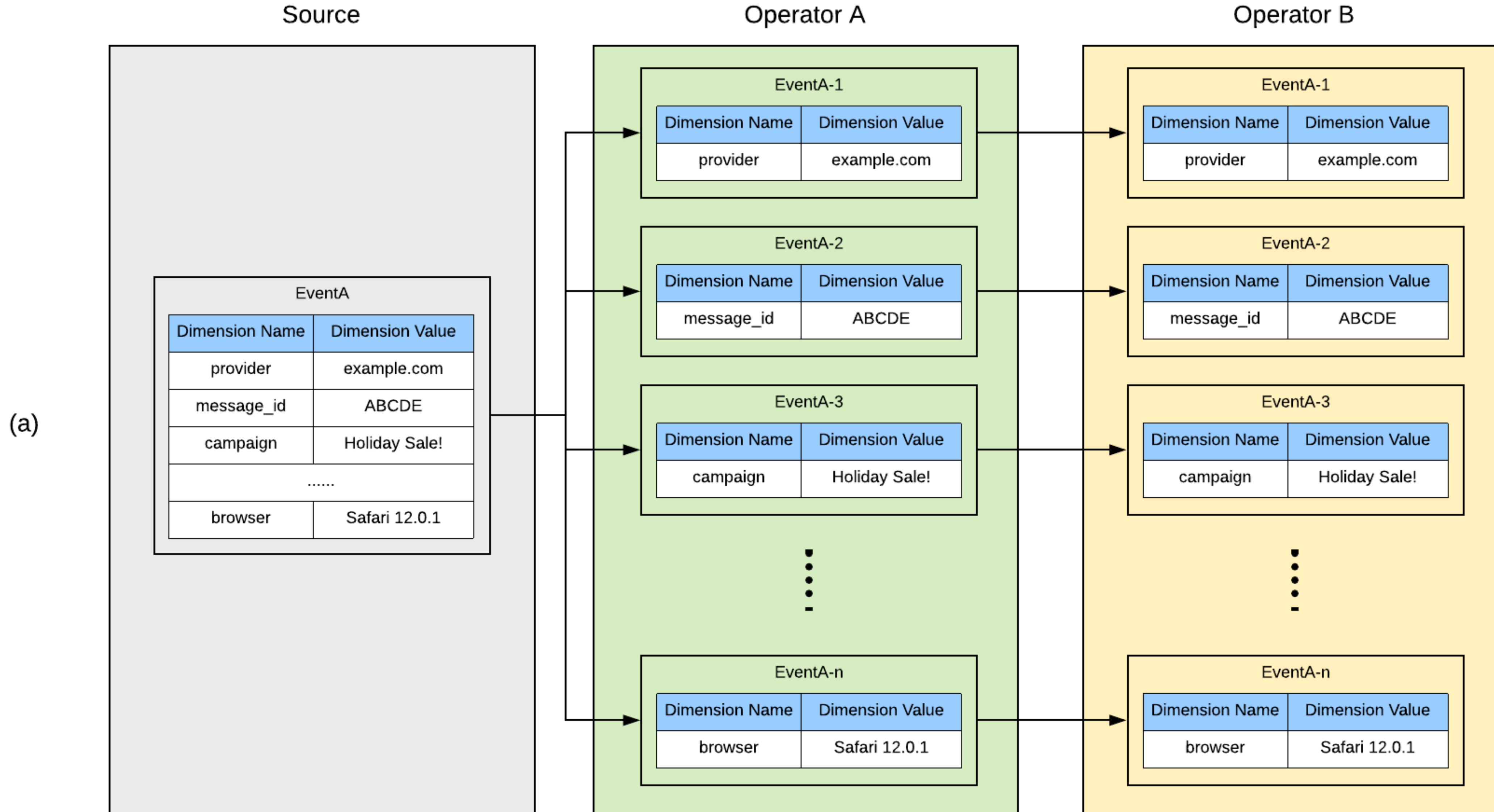
- Many unsynchronized external event sources (off by hours)
- Historical event synchronization (off by years)
- Handle all events in a single job with processing time



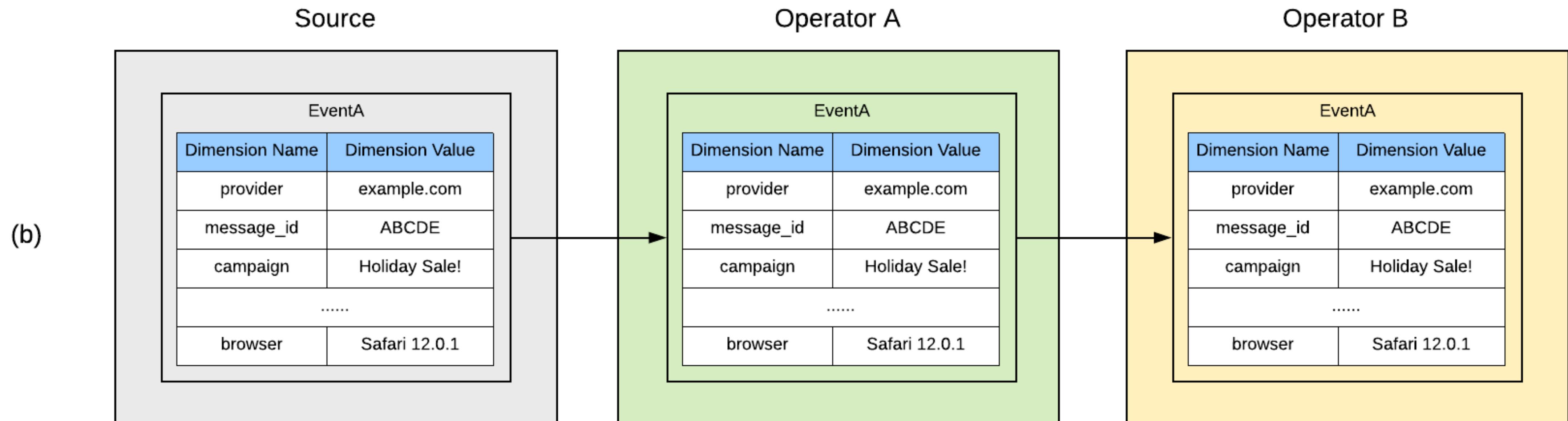
Flink Backward



Fan-out Approach (a)

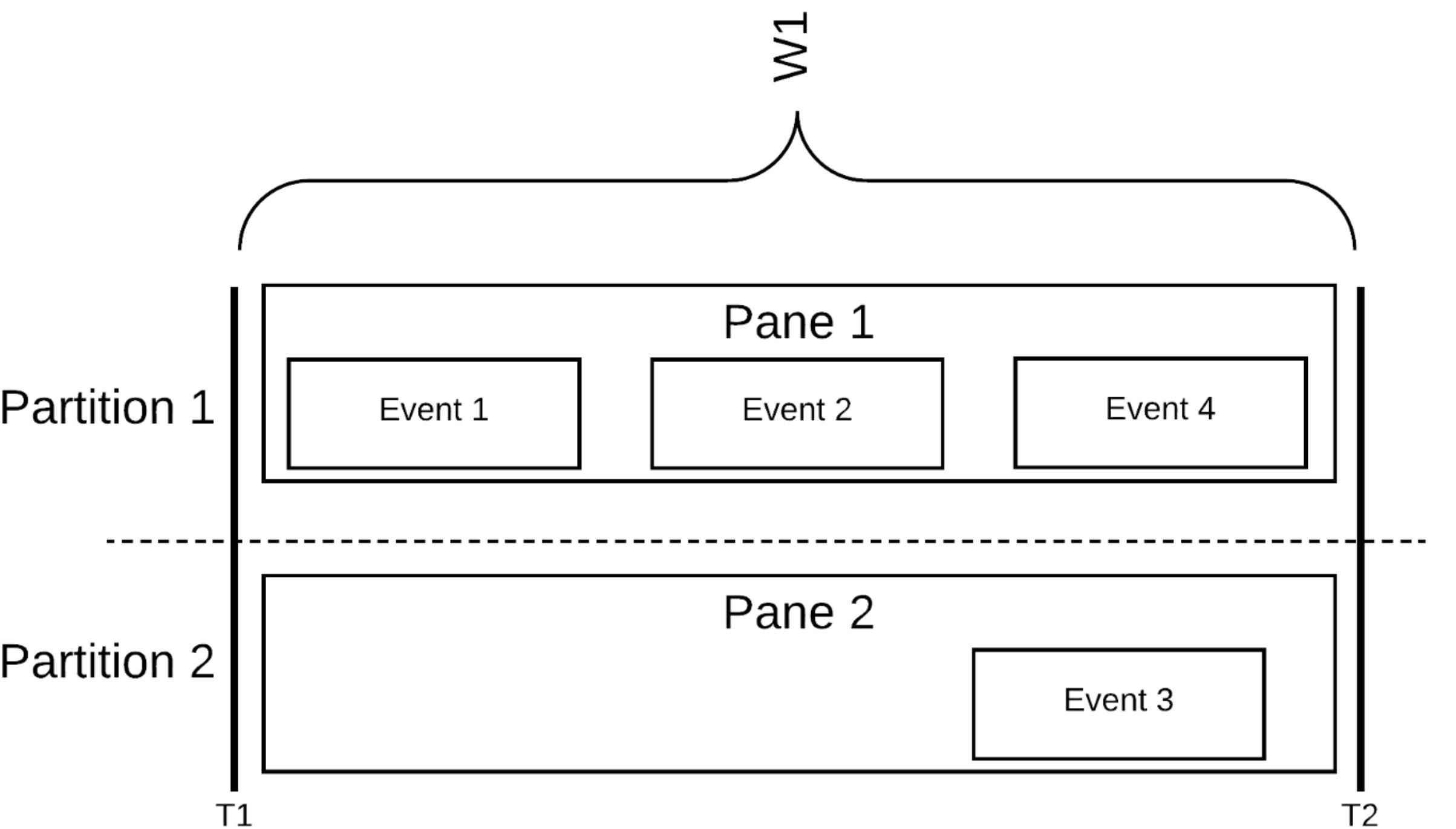


Fan-out Approach (b)



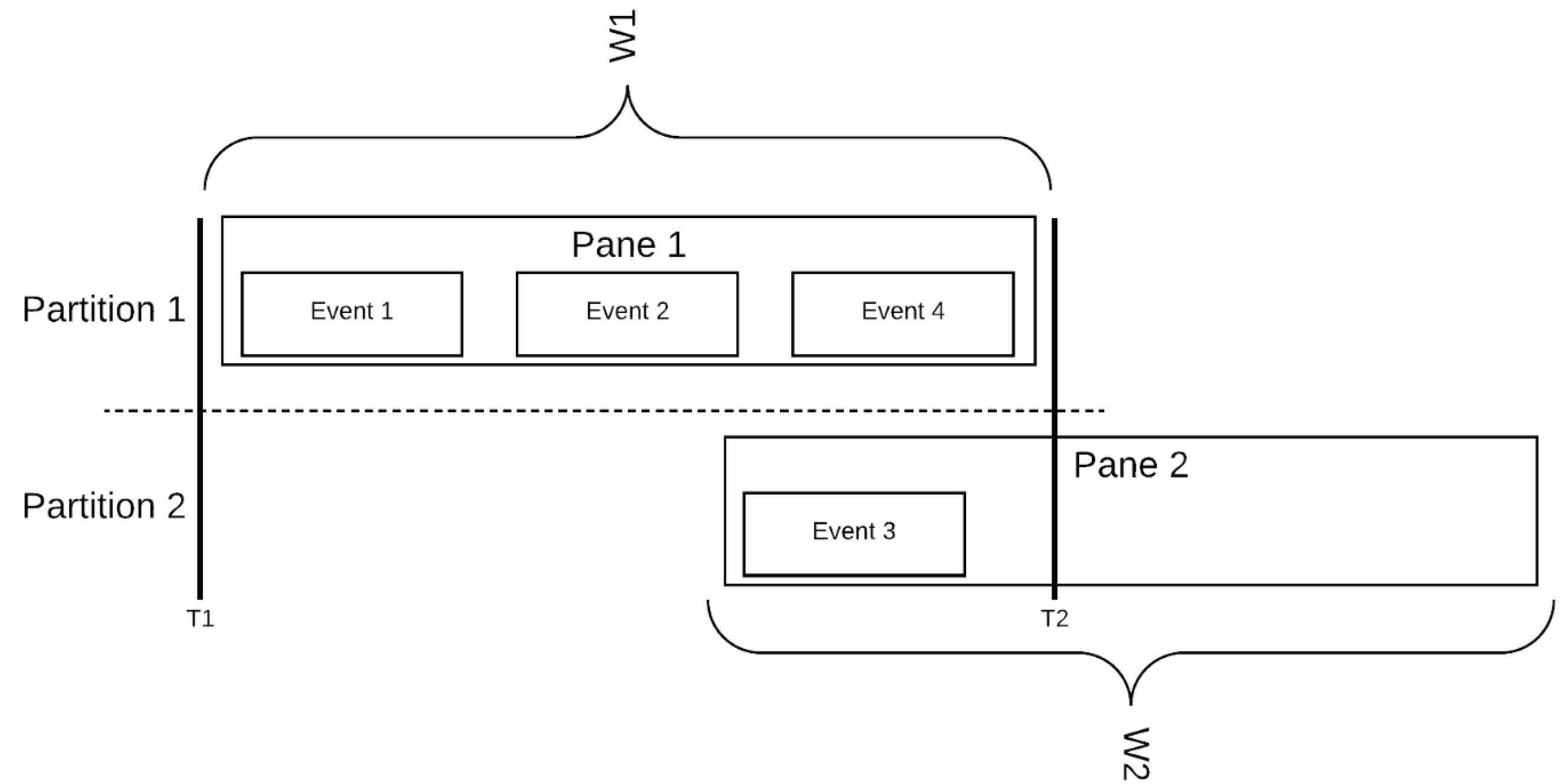
Windowing

- A window on a partition is called a pane
- Tumbling and sliding window panes fire simultaneously
- Triggers live on Java heap prior to Flink 1.6, or in RocksDB after Flink 1.6
- Millions of triggers have huge impact



Windowing

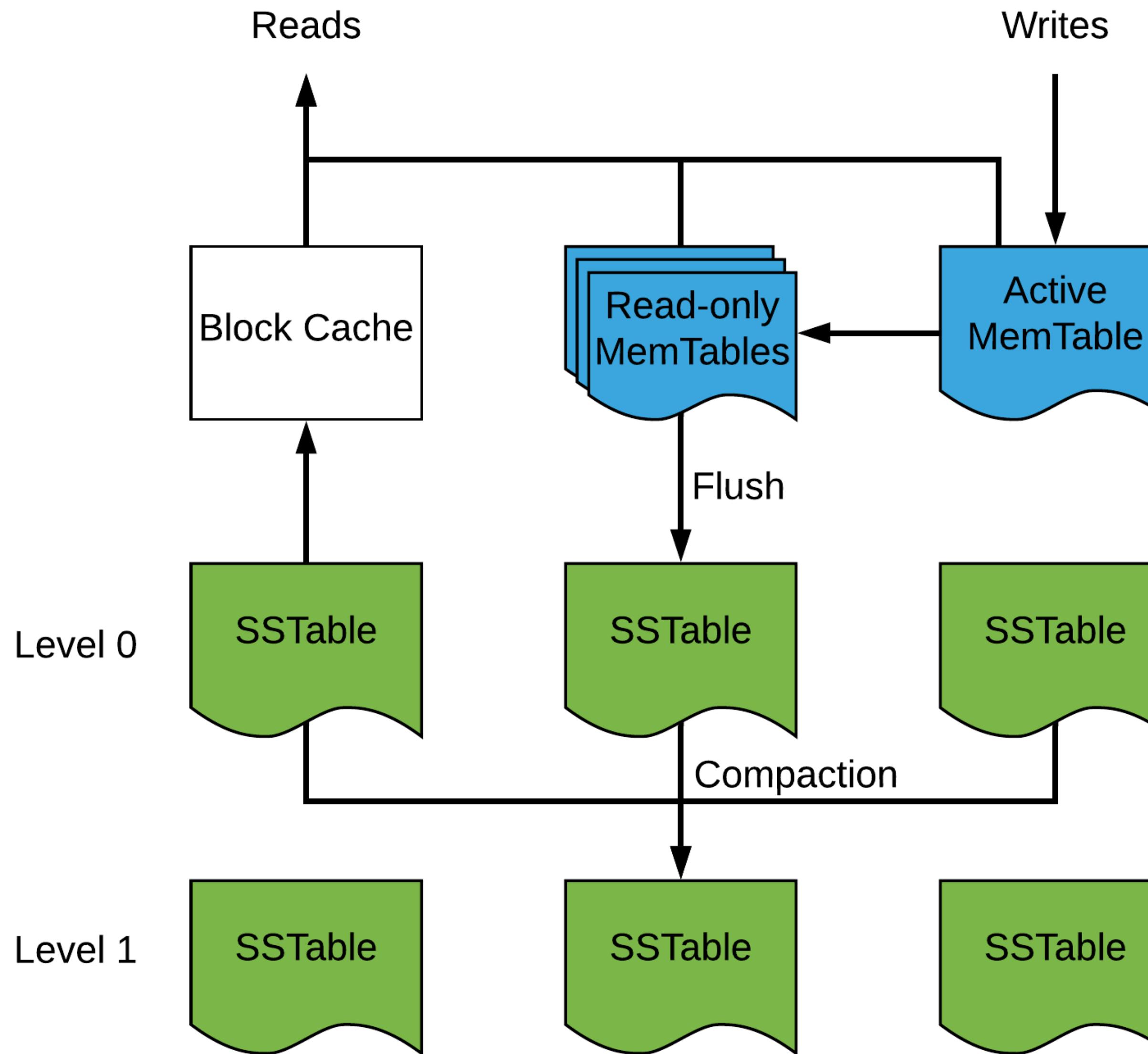
- Pane opens only when there is event on partition
- Pane closes after fixed amount of time after earliest event in the pane
- Staggered trigger firings



Configuration

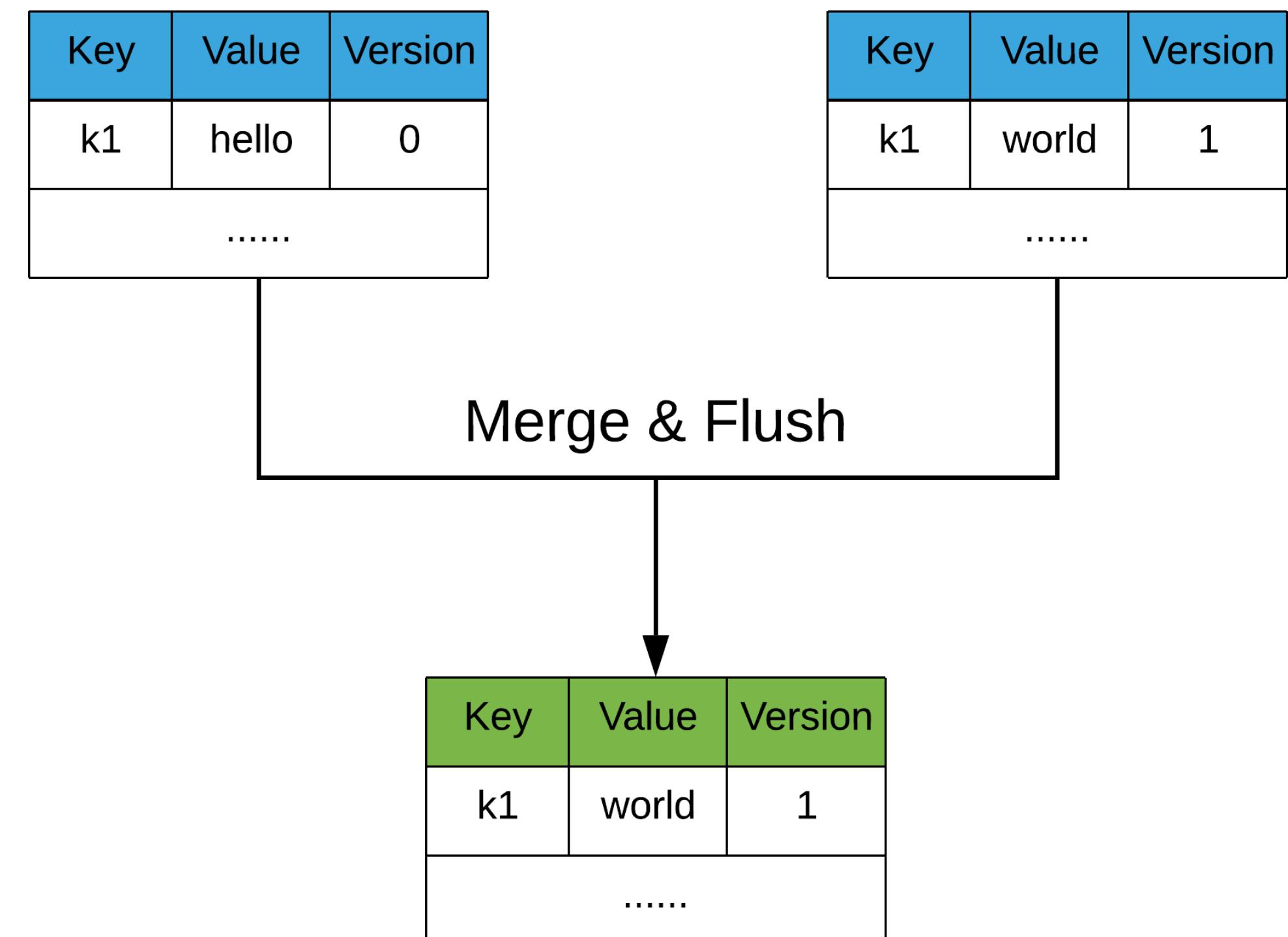


RocksDB State Backend



RocksDB State Backend

- **Block cache size:** Cache of uncompressed blocks are read from SSTables for reads.
- **Write buffer size:** Size of MemTable.
- **Write buffer number:** Number of MemTables before flushing to disk.
- **Minimum write buffers to merge:** Number of MemTables to merge before flushing to SSTable.



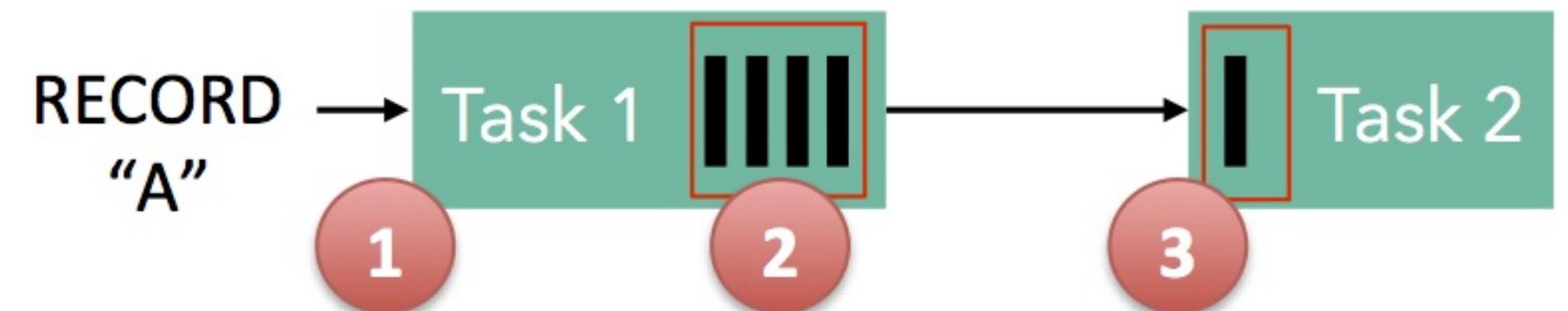
State TTL

- Feature introduced in Flink 1.6
- Can be managed by RocksDB
- Expired keys only removed on reads (or on savepoint restore)
- Flink 1.8 will add continuous cleanup and clean on compaction



Back Pressure

- Event consumption slower than event production
- Higher parallelism for expensive operator, at the cost of potentially shuffling events
- Rate limit source to avoid back pressure altogether
- Rate limiting other operators may increase checkpoint alignment time



Capacity Planning

- Job bounded by CPU, memory, or both?
- Use fewer slots than CPU cores
- Kryo serialization is expensive
- Slots per host also affects memory used by RocksDB



Summary

- “Time is of the essence”
- Reduce internal events
- Spiky workload is bad
- Understand RocksDB
- Avoid back pressure
- Do capacity planning



Questions

Real-time Analytics: klaviyo.tech/tagged/counting
Join Us: bit.ly/klaviyocareers

✉️ ning.shi@klaviyo.com

🐦 [@ihsgnin](https://twitter.com/ihsgnin)



Identity Graph with Flink

Flink Forward SF 2019

Vivek Thakre
Principal Engineer @Intuit

Intuit

Powering prosperity around the world



Identity Graph : What are we solving?

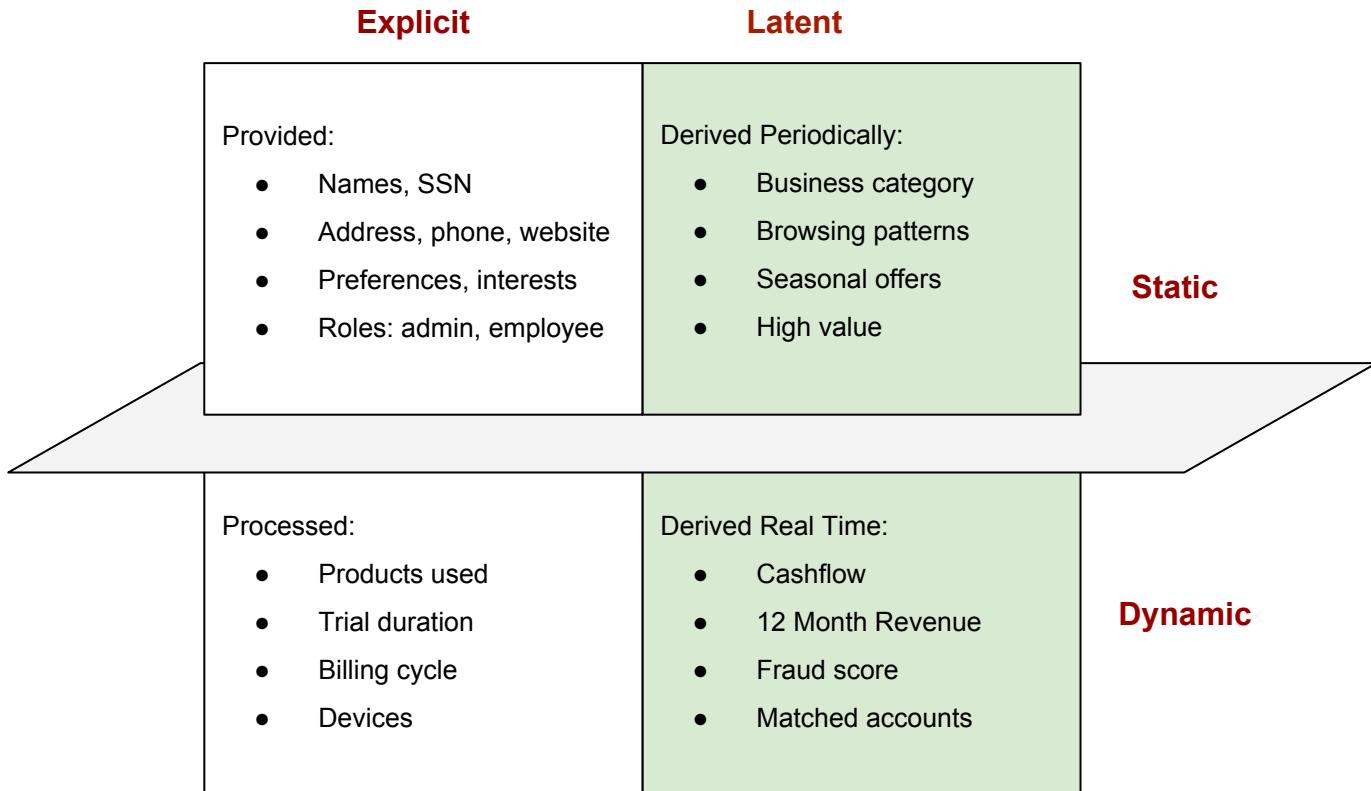
- As an Intuit customer, I keep having to introduce myself to multiple customer care agents
- I get offers for Products and Credit Cards that I already have
- As a Customer Success Agent, I spent minutes collecting information from my customers before I can help them
- As a marketer, I don't have visibility into my customer's journey once they become a customer

Identity Graph

- Goal
 - Personalize the journey for everyone in Intuit's ecosystem
- What
 - Unified Profile Platform and a Service which composes data from various data sources and products into real-time accessible views that personalization services use to change the customer experience in real-time



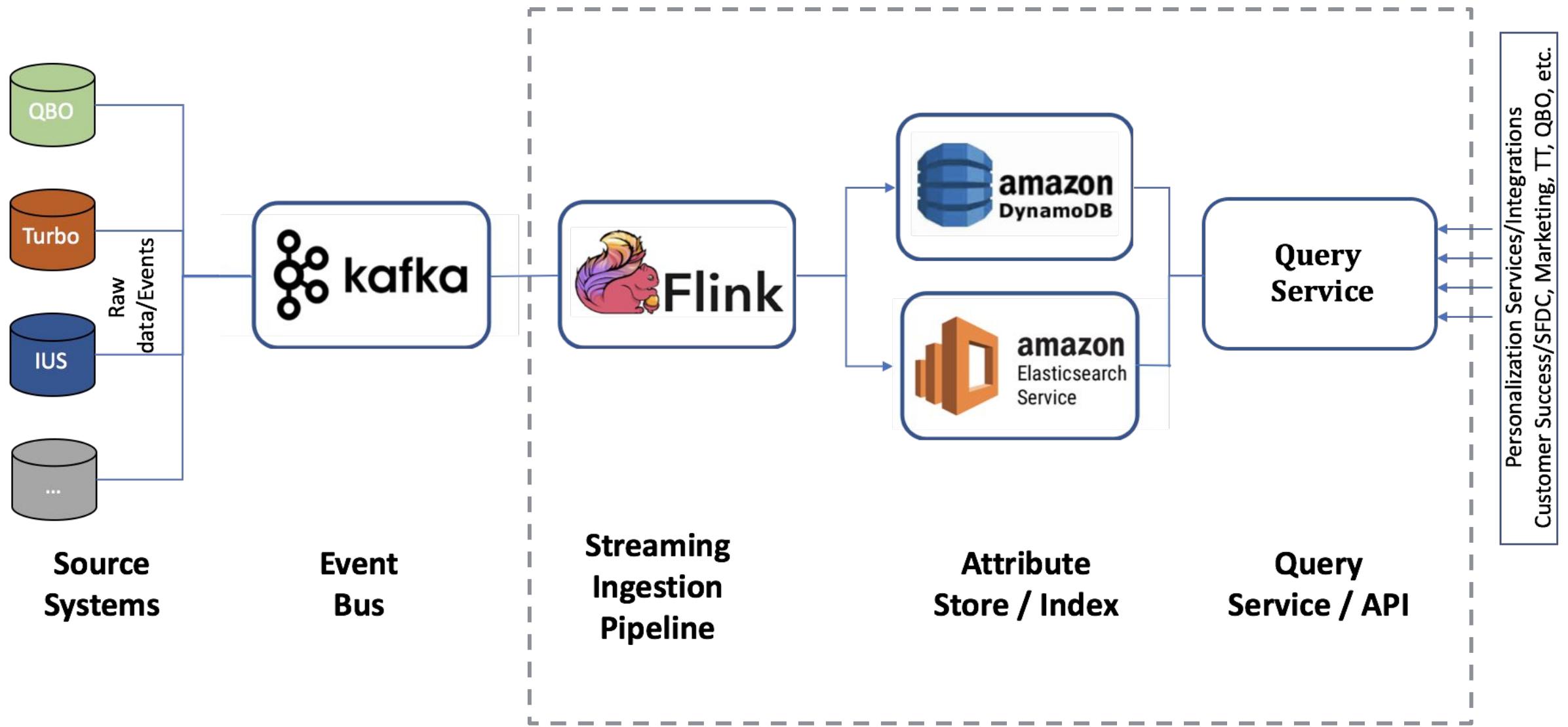
Profile Attributes and Metadata



Attribute Metadata

- Name
- Description
- Data Type
- Date Created
- Path
- Source
- Data Classification Level
- Encryption Level
- Identifiable
- Authorization Level
- ...

Identity Graph : High Level Data Flow

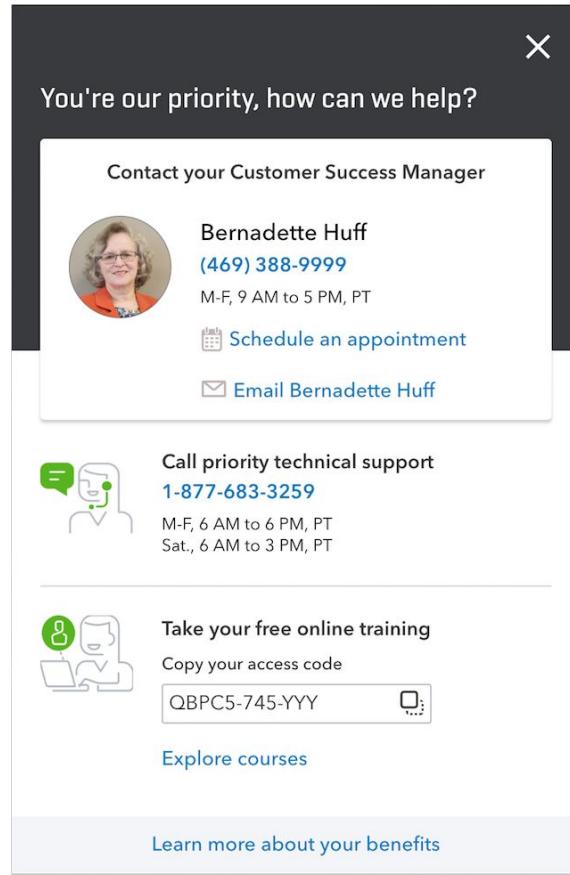


Use Cases and Ingestion Pipeline

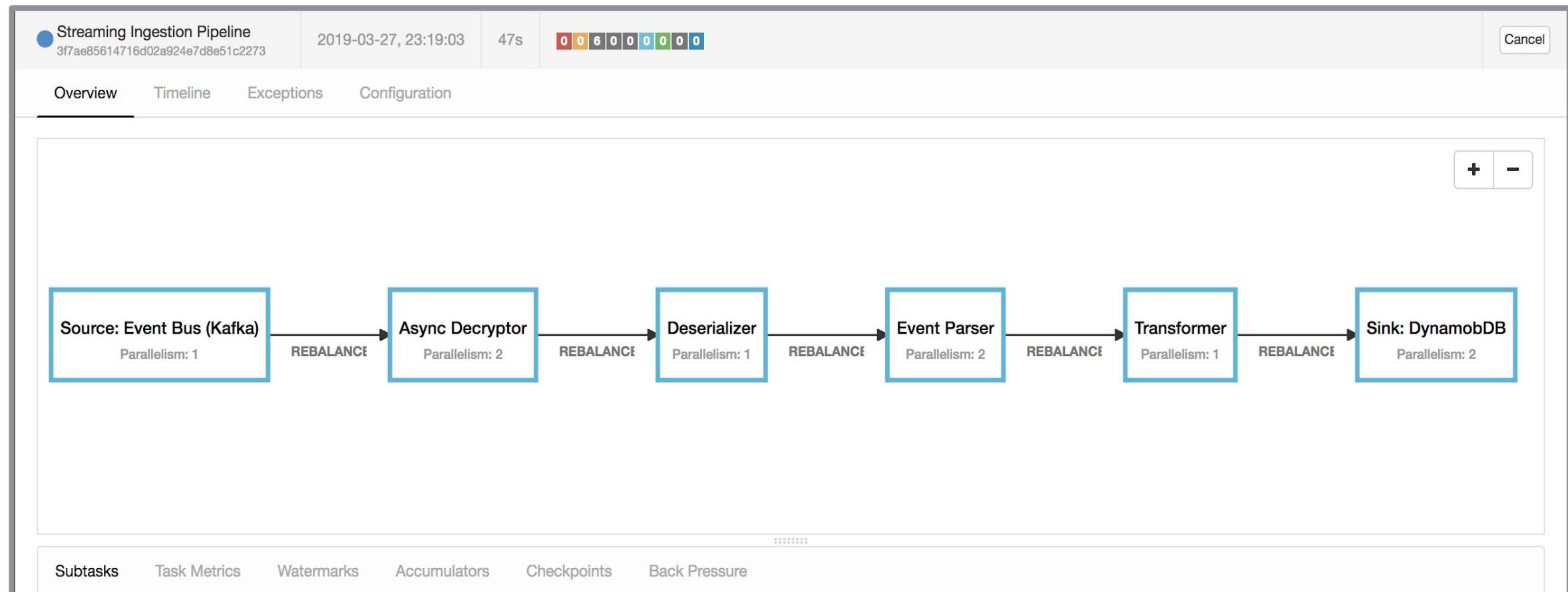
- Different use cases have different requirements
 - Snapshots vs Delta
 - PII vs Non PII Data
 - Point Queries vs Search Capabilities
 - Predefined Schema vs Dynamic
 - Different data reconciliation criteria
- Abstracted common components (Sources, Sinks, Operators)
- Common Components made configurable

Use Case : Priority Circle

Enable Quickbooks Advanced to display Customer Success Manager details for eligible customers



Streaming Ingestion Pipeline



Decryptor using Async IO API

- Higher Streaming Throughput
- Decryptor service supports Async requests
- Implement AsyncFunction with callback
- Configurable failure handler on timeout
- Capacity : backpressure is triggered once the capacity is exhausted
- Stream order needs to be preserved

```
DataStream<String> decryptedInputStream = AsyncDataStream
    .orderedWait(inputStream, new AsyncDecryptor(parameters),
        parameters.getInt("decryptor.timeout", 1000), TimeUnit.MILLISECONDS,
        parameters.getInt("decryptor.capacity", 100)).name("Async Decryptor")
```

DynamodbSink : Merge Example - Events with same ID

Event 1

```
{  
  "identity" : {  
    "realmID" : "1000040055"  
  },  
  "ordinal" : 1553000000,  
  "payroll" : {  
    "insights" : {  
      "daysSinceSignup" : 10,  
      "hasContractors" : "true",  
      "employee" : {  
        "hourlyEECount" : 35  
      }  
    }  
  }  
}
```



Event 2

```
{  
  "identity" : {  
    "realmID" : "1000040055"  
  },  
  "ordinal" : 1553500000,  
  "payroll" : {  
    "insights" : {  
      "daysSinceSignup" : 20,  
      "hasContractors" : "true"  
    }  
  }  
}
```



Merged Event

```
{  
  "identity" : {  
    "realmID" : "1000040055"  
  },  
  "ordinal" : 1553500000,  
  "payroll" : {  
    "insights" : {  
      "daysSinceSignup" : 20,  
      "hasContractors" : "true",  
      "employee" : {  
        "hourlyEECount" : 35  
      }  
    }  
  }  
}
```

Dynamodb Sink with Low Latency

- invoke for each event
- read item
- apply merge function (incoming and existing) - configurable
- update with optimistic locking
 - conditional update with version number
 - on failure retry above steps
- configurable Failure Handlers
- retries with exponential backoff

Dynamodb Sink with High Throughput

- writes are buffered and merged* for same ids
- during flush
 - batch read for all Ids, apply merge with existing and batch write
- implements checkpointed
- buffer is flushed when
 - size is \geq BATCH_SIZE
 - during checkpoint if buffer is not empty
- configurable Failure Handlers
- retries with exponential backoff for unprocessed items

Domain Event Transformations

- Events are published from different sources with varying schema and format
- Need to transform events into a unified schema
- Transformation rules are domain driven
- Utilizing Jolt library to perform transformations
- Transformations based on EventType and transformation-specs
- Transformation specs in classpath

Jolt : JSON to JSON transformation library written in Java where the "specification" for the transform is itself a JSON document.

<https://github.com/bazaarvoice/jolt>

Jolt Transformation

Jolt Spec JSON Validate

```
1 {  
2   "operation": "default",  
3   "spec": {  
4     "identity_business_realmid": ""  
5   }  
6 },  
7 {  
8   "operation": "shift",  
9   "spec": {  
10     "identity_business_realmid": "identity.realmID",  
11     "sb_business_payroll_insights_dayssincepayrollsSignup":  
12       "payroll.insights.daysSinceSignup",  
13     "sb_business_payroll_insights_hascontractors":  
14       "payroll.insights.hasContractors",  
15     "sb_business_payroll_insights_numhourlyemployees":  
16       "payroll.insights.employee.hourlyEECount",  
17     "sb_business_payroll_payrollusage_qbobillingchannel":  
18       "billing.payroll.qboChannel"  
19   }  
20 },  
21 },  
22 {  
23   "operation": "modify-overwrite-beta",  
24   "spec": {  
25     "identity": {  
26       "realmID": "=toString"  
27     }  
28   }  
29 }
```

Json Input JSON Validate

```
1 [ {  
2   "identity_business_realmid": 1000040055,  
3   "sb_business_payroll_insights_dayssincepayrollsSignup":  
4     "sb_business_payroll_insights_numhourlyemployees": 0,  
5   "sb_business_payroll_insights_hascontractors": 0,  
6   "sb_business_payroll_insights_hasworkerscompensation":  
7     "sb_business_payroll_payrollusage_qbobillingchannel":  
8   }  
9 }
```

Output / Errors Transform Sort Output?

```
1 {  
2   "identity" : {  
3     "realmID" : "1000040055"  
4   },  
5   "payroll" : {  
6     "insights" : {  
7       "daysSinceSignup" : 0,  
8       "hasContractors" : 0,  
9       "employee" : {  
10         "hourlyEECount" : 0  
11       }  
12     }  
13   },  
14   "billing" : {  
15     "payroll" : {  
16       "qboChannel" : "other"  
17     }  
18   }  
19 }
```

Dynamic Transformations : Metadata Service

Profile Self Serve

Search Attributes

Name ✓

Manage Requests

Admin

Manage Schema

Manage Mapping Specs

Manage Projected Views

Generate Jolt Specs

Manage Users

Manage Sandboxes

Batch Operations

Description

Specification

Specification Type

Delete... Cancel Save



Dynamic Transformations : Event Driven

- Metadata Service Source
 - Rest based Source implen
 - Configurable with Poll Interval
 - Creates Event with Jolt Mappings Specs

```
DataStream<MappingSpecification> joltMappingsSpecsStream = env  
    .addSource(new MetadataServiceSource())
```

- Broadcast
 - Describe State for Jolt Mappings Specs
 - Broadcast Source Stream with described state

```
MapStateDescriptor<Void, MappingSpecification> joltMappingsSpecsStateDesc =  
new MapStateDescriptor<>("JoltMappinsSpecsDesc", Types.VOID,  
TypeInformation.of(new TypeHint<MappingSpecification>() {}));
```

```
BroadcastStream<Map<String, MappingSpecification>>  
broadcastedJoltMappingsSpecsStream = joltMappingsSpecsStream  
    .broadcast(joltMappingsSpecsStateDesc);
```

Dynamic Transformations : JoltTransformationFunction

```
DataStream<ProfileEntity> transformedStream =  
    inputStream.connect(broadcastedJoltMappingsSpecsStream)  
        .process(new JoltTransformerFunction());
```

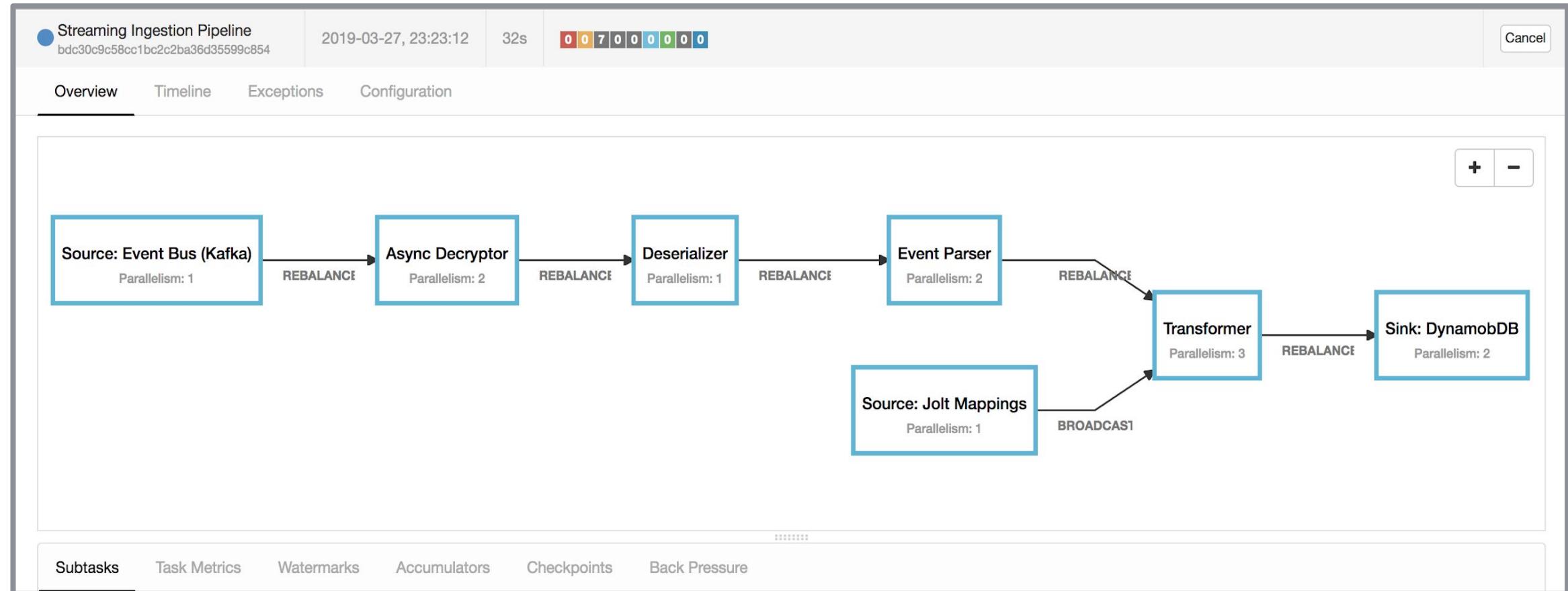
- Connect Broadcasted Stream with Event Stream
- JoltTransformationFunction Implements BroadcastProcessFunction
- BroadcastProcessFunction : processBroadcastElement
 - update broadcasted state

```
BroadcastState<Void, MappingSpecification> bcJoltMappindsSpecsState =  
    ctx.getBroadcastState(joltMappingsSpecsStateDesc);  
    bcJoltMappindsSpecsState.put(null, joltMappingsSpecs);
```

- BroadcastProcessFunction : processElement
 - processElement : read state and apply transformation on event

```
MappingSpecification joltMappingsSpecs =  
    ctx.getBroadcastState(joltMappingsSpecsStateDesc)  
        .get(null);
```

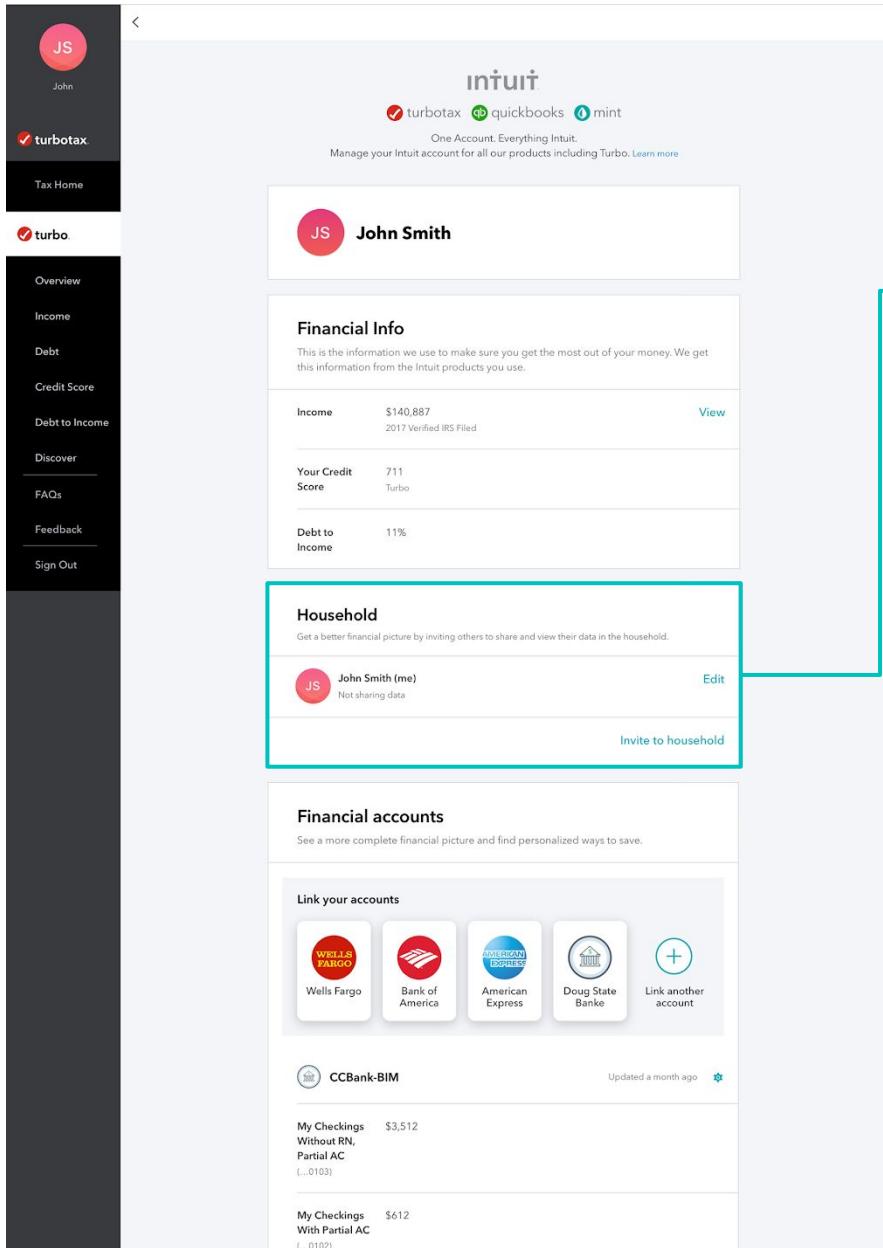
Ingestion Pipeline with Dynamic Transformation



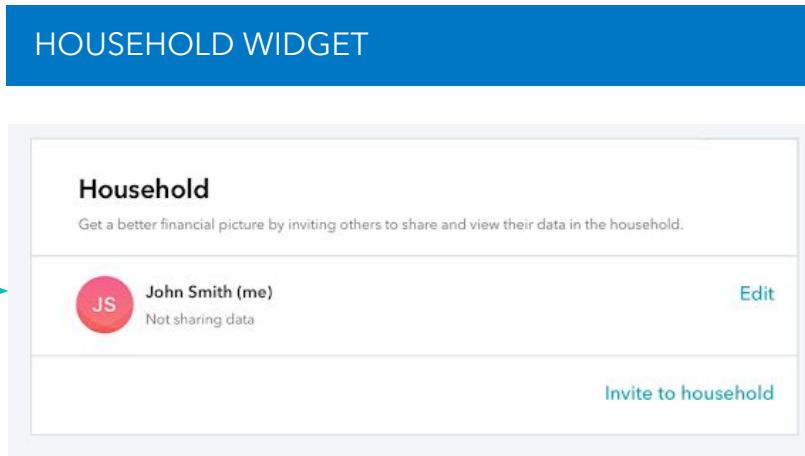
Dynamic Transformation with Broadcast

- ‘Side Input’ is better fit for this pattern
- For static and slowly evolving data
- FLIP-17 Side Inputs for DataStream API

Use Case : Turbo Household



The screenshot shows the Intuit Turbo Household interface. On the left is a sidebar with user profile (JS, John), account selection (turbotax, turbo), and navigation links (Overview, Income, Debt, Credit Score, Debt to Income, Discover, FAQs, Feedback, Sign Out). The main area displays financial information for John Smith, including income (\$140,887), credit score (711), and debt-to-income ratio (11%). Below this is the 'Household' section, which is highlighted with a teal border. It shows a list of household members (John Smith (me)) and includes an 'Edit' button and an 'Invite to household' button. At the bottom, there's a 'Financial accounts' section with a 'Link your accounts' button and a list of linked accounts (Wells Fargo, Bank of America, American Express, Doug State Banke) along with a 'Link another account' button.



The screenshot shows the Household Widget interface. It has a blue header bar with the text 'HOUSEHOLD WIDGET'. Below it is a white card with the title 'Household' and a sub-instruction 'Get a better financial picture by inviting others to share and view their data in the household.' It lists a single household member, 'John Smith (me)', with the status 'Not sharing data'. There is an 'Edit' button next to the member's name and an 'Invite to household' button at the bottom right. A teal arrow points from the 'Household' section in the main screenshot to this widget.

Purpose

Allow users to link their profiles in order to share financial data and see a household view of their finances.

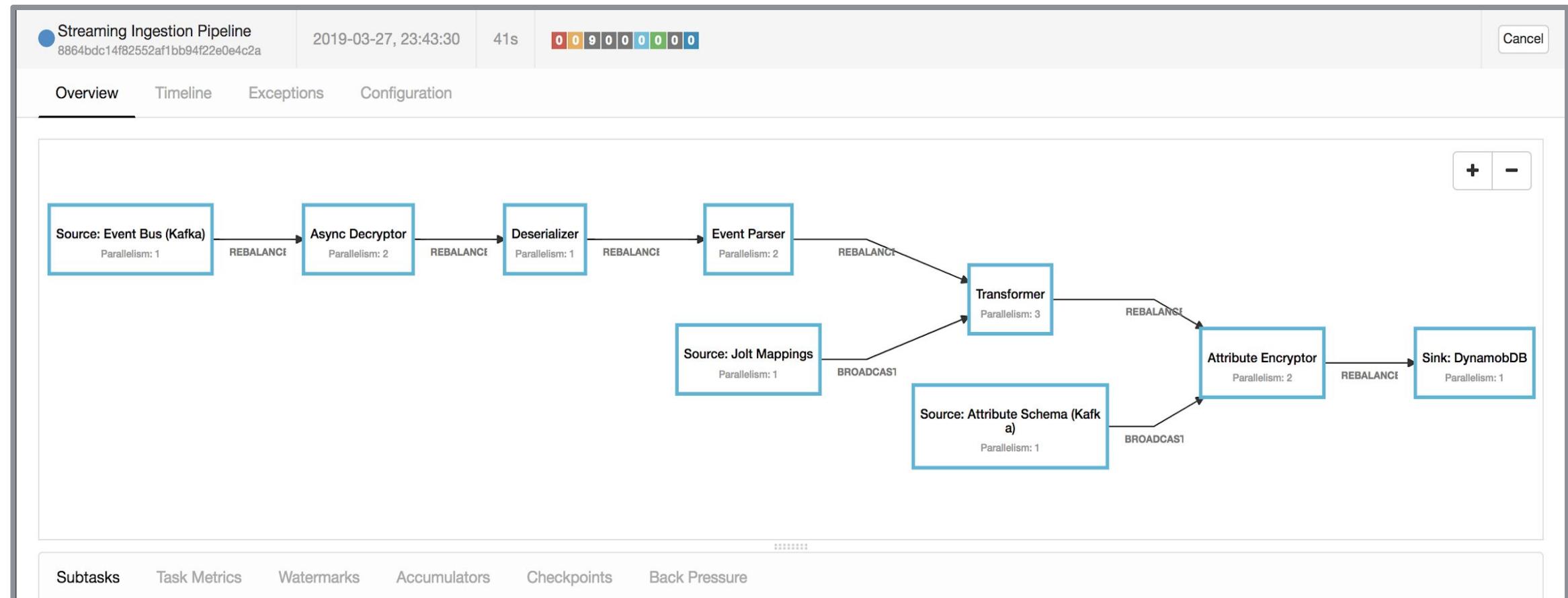
Release 1.0

Base capability to allow individuals to connect their profiles to share their **credit scores** via one way data sharing.

Attribute Level Encryption

- Attribute schema updates along with metadata is published to event-bus
- Broadcast metadata to Encryptor Function
- Based on metadata, determine highly sensitive attributes and get encryption keys
- Encryption is performed utilizing external service

Ingestion Pipeline with Attribute Level Encryption



Use Case : VCI or Contact Info

Search Results

Guided Tour | Help for this Page 

 Search Feeds

united  Options...

 **Records**

Reports (0)
People (0)
Accounts (5)
Leads (0)
Documents (0)
Contacts (4)
Attachments (0)

 **Accounts (5)**

Action	Account Name	Account Site	Phone	Account Owner Alias
Edit	Express Logistics and Transport		(503) 421-7800	YName
Edit	United Oil & Gas, UK		+44 191 4956203	YName
Edit	University of Arizona		(520) 773-9050	YName
Edit	United Oil & Gas Corp.		(212) 842-5500	YName
Edit	United Oil & Gas, Singapore		(650) 450-8810	YName

 **Contacts (4)**

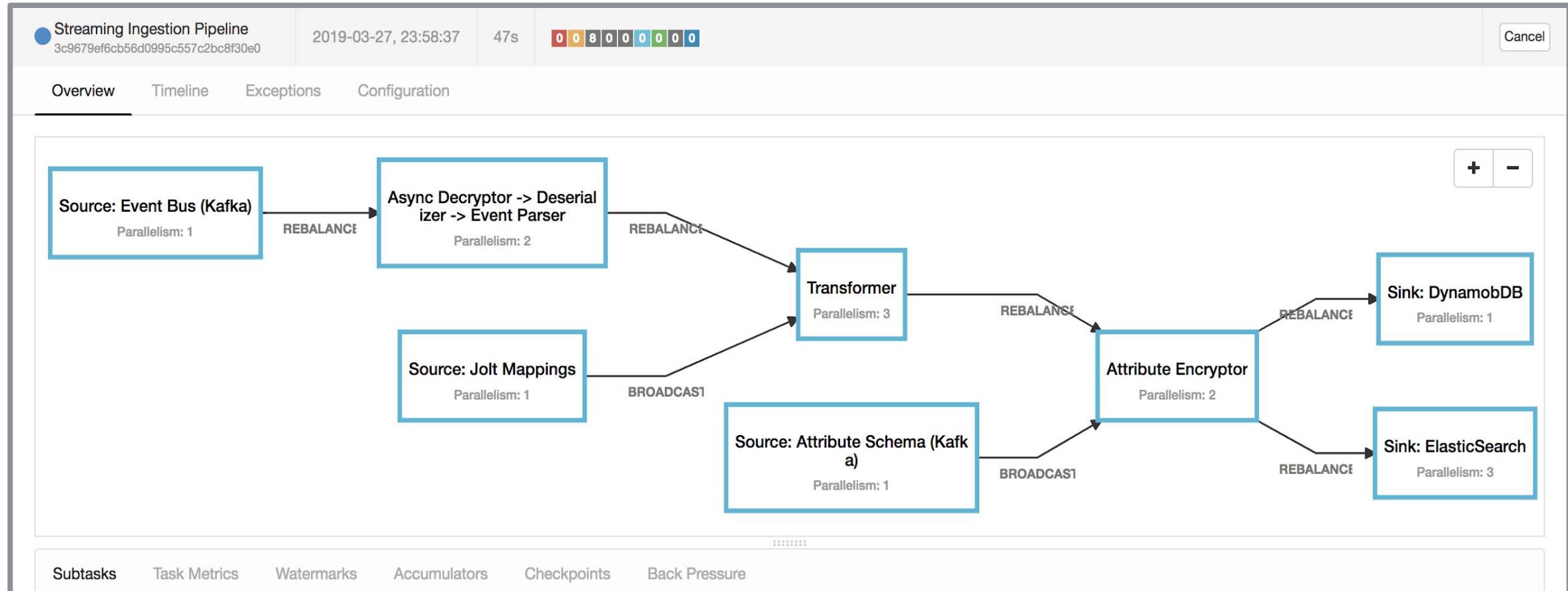
Action	Name	Account Name	Account Site	Phone	Email	Contact Owner Alias
Edit	Mr. Josh Davis	Express Logistics and Transport		(503) 421-7800	j.davis@expressl&t.net	YName
Edit	Ms. Jane Grey	University of Arizona		(520) 773-9050	jane_gray@ua.edu	YName
Edit	Ms. Ashley James	United Oil & Gas, UK		+44 191 4956203	ajames@uog.com	YName
Edit	Ms. Babara Levy	Express Logistics and Transport		(503) 421-7800	b.levy@expressl&t.net	YName

ElasticSearch Sink

- Utilize side-output streams
- Elasticsearch Connector that is bundled with Flink
- Implements checkpointed
- IndexableAttributes : documentId, attribute and value

```
OutputTag<IndexableAttributes> indexableAttribute =  
    new OutputTag<IndexableAttributes>("indexableAttribute"){};  
DataStream<IndexableAttributes> attributeIndexerStream =  
    encryptedStream.getSideOutput(indexableAttribute);  
attributeIndexerStream.addSink(new ElasticSearchSink(parameters));
```

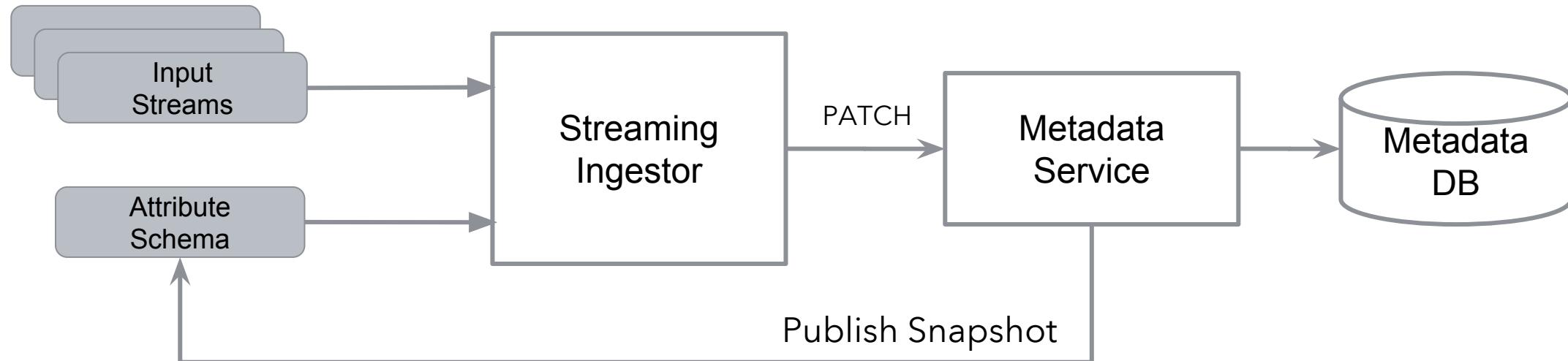
Ingestion Pipeline with Search Capabilities - Side Output



Use Case : Rapid Experimentation

- Experimentation Platform run 1000s of experiments
- Data Scientists and analyst run models and create and derive new attributes
- Data is ingested with new attributes but its not a part of schema
- Schema needs to be updated dynamically
- New attributes needs to be discoverable and usable within 2s

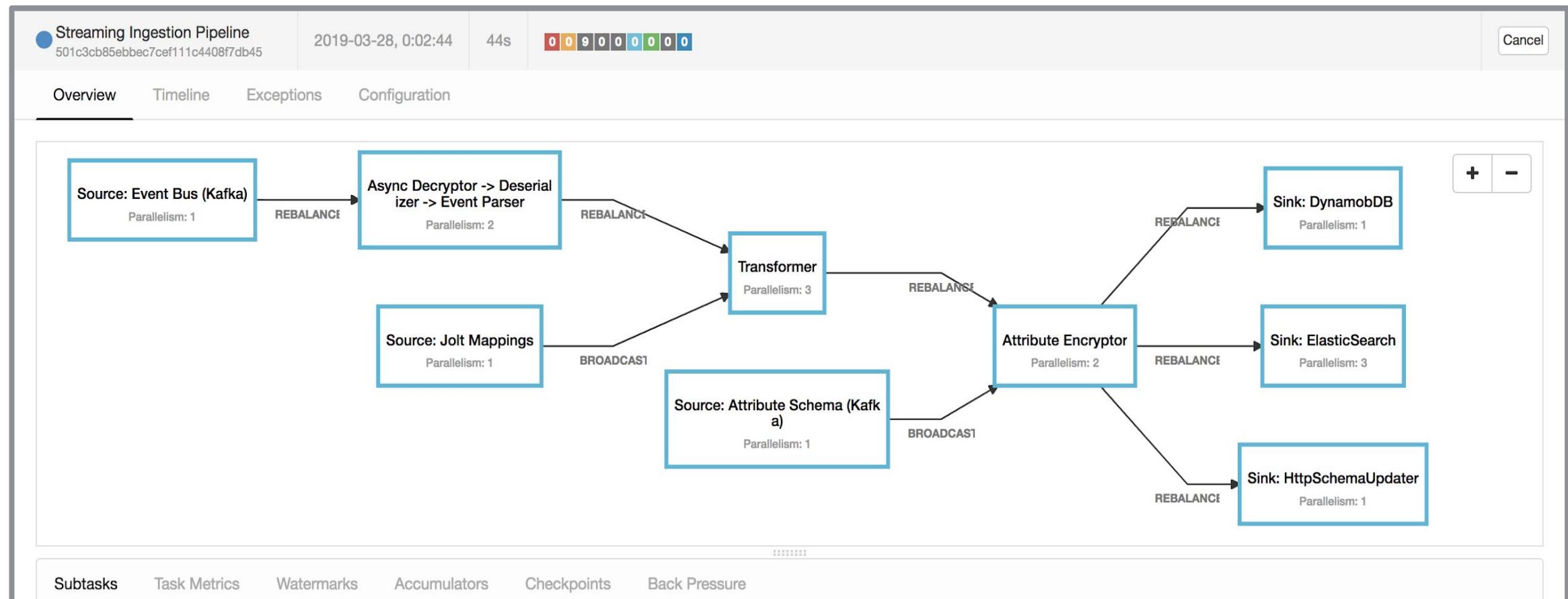
Dynamic Schema Update



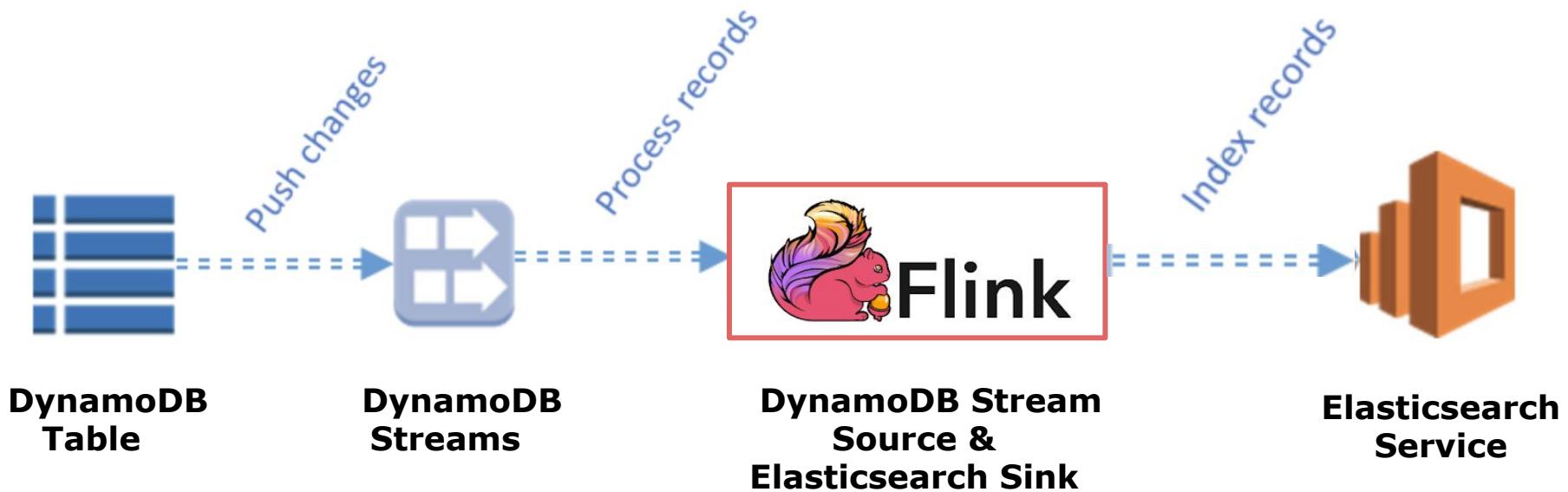
Dynamic Schema Update : HttpSchemaUpdaterSink

- Compute diff between existing schema and incoming event schema
- Utilize Side-output stream to create schema change stream
- Update metadata via http patch call to Metadata Service
- Created HttpSchemaUpdaterSink
 - Buffer and eliminate duplicate schema updates
 - Implements ProcessingTimeCallback
 - register future time with new update
 - Flush requests when
 - updates count \geq MAX_UPDATES or
 - triggered by timer after SCHEMA_UPDATE_INTERVAL

Streaming Ingestion Pipeline



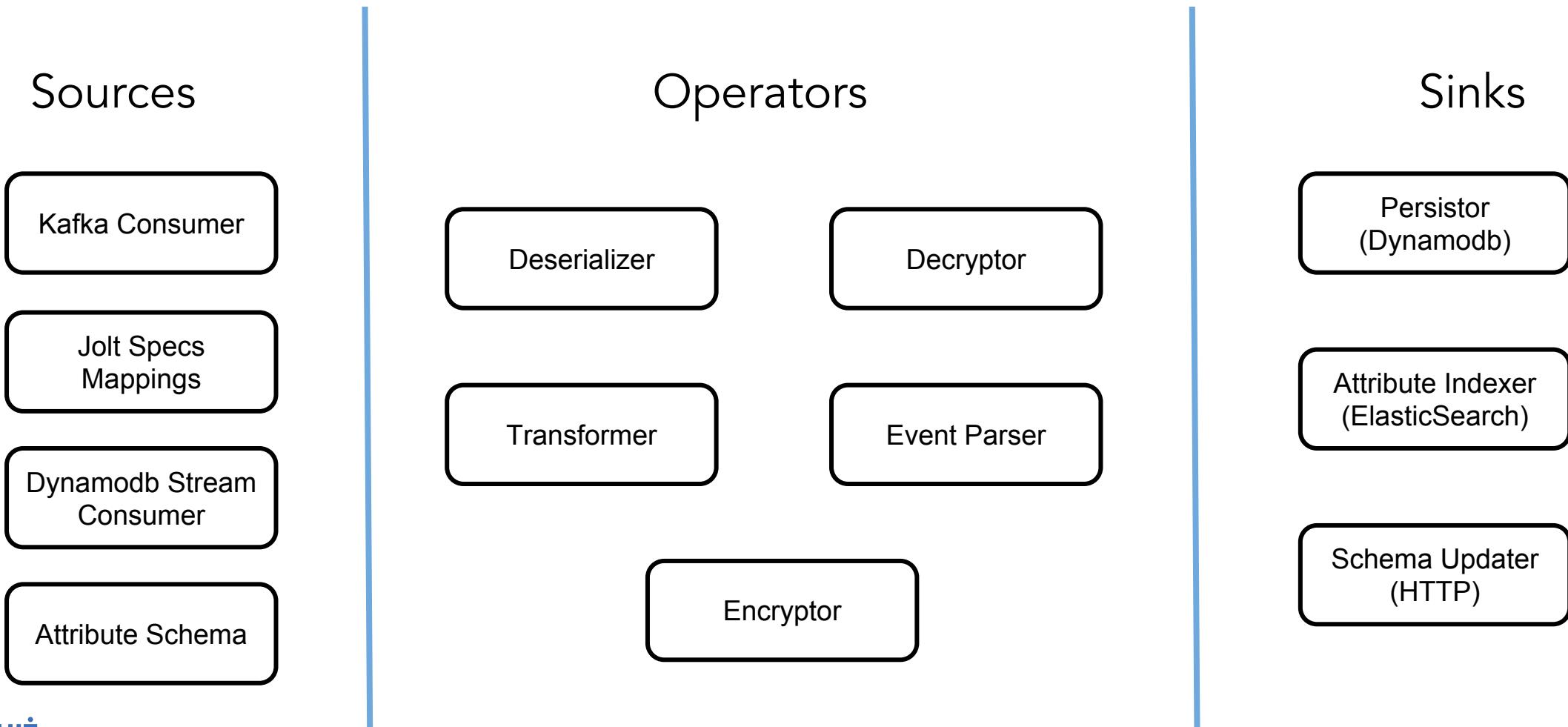
Dynamodb Streams to Elasticsearch



- Utilizing `FlinkDynamoDBStreamsConsumer` in Flink 1.8
- [FLINK-4582](#)

Streaming Ingestion Pipeline

Building blocks of Pipeline, Abstracted as Components, New Components are created with requirements



Q&A

Thank You



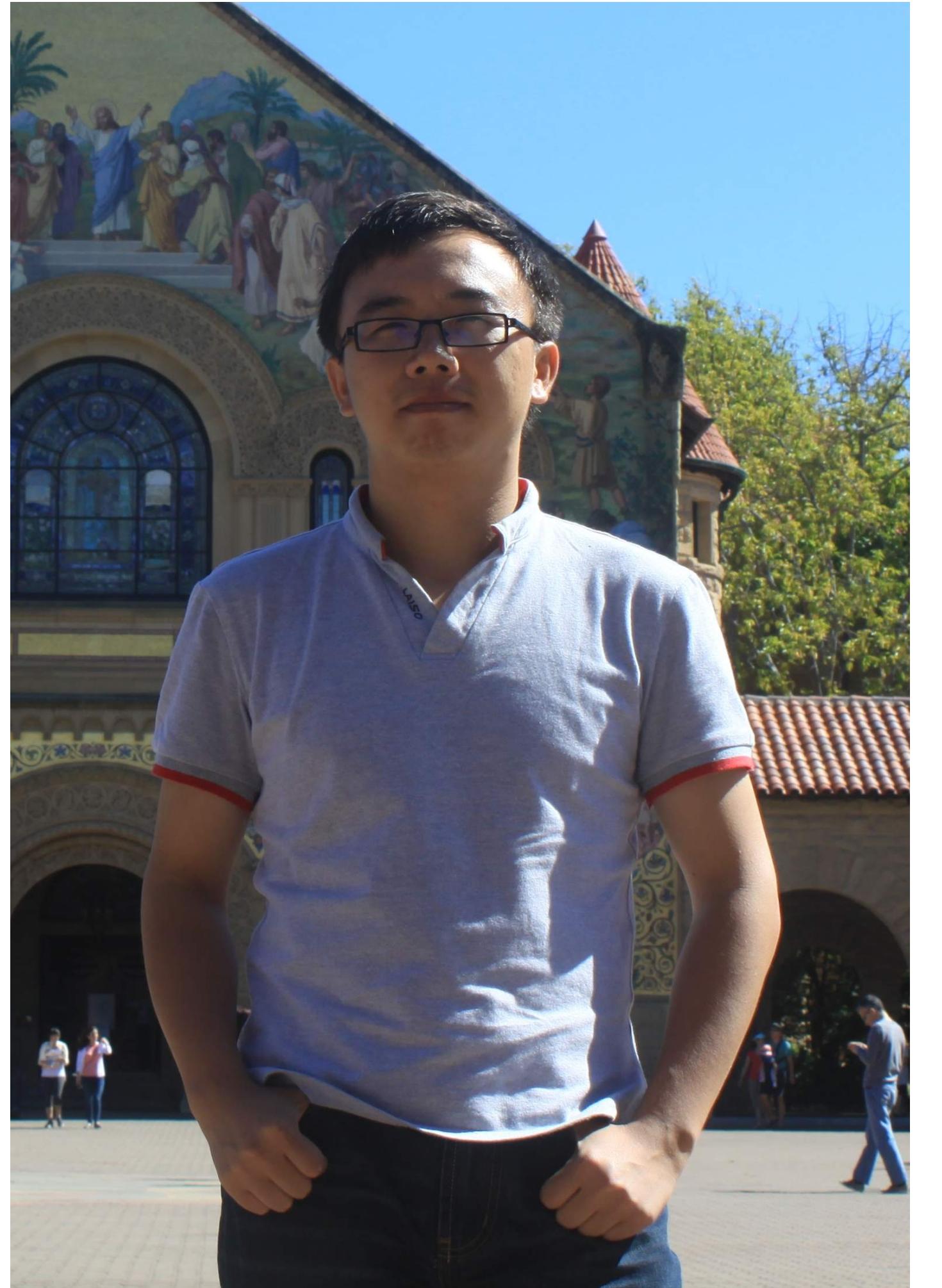
Elastic Data Processing with Apache Flink and Apache Pulsar

Sijie Guo (sijieg)

2019-04-02

Who am I

- Apache Pulsar PMC Member
- Apache BookKeeper PMC Member
- Interested in technologies around Event Streaming



Agenda

- What is Apache Pulsar?
- A Pulsar View on Data - Segmented Stream
- Pulsar - Access Pattern & Tiered Storage
- Pulsar - Schema
- When Flink meets Pulsar

What is Apache Pulsar?

Pub/Sub Messaging



2003



2010



2012



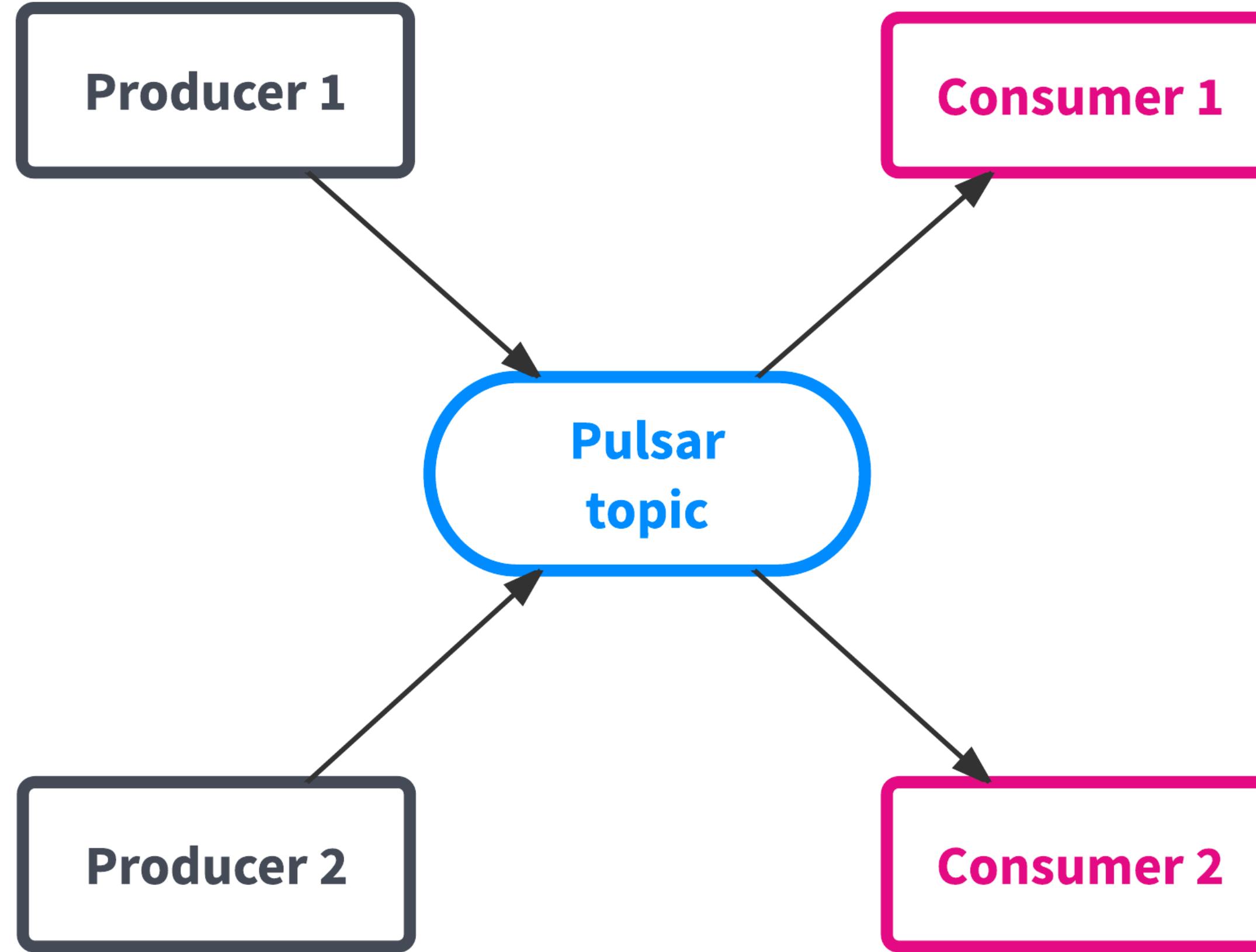
2006



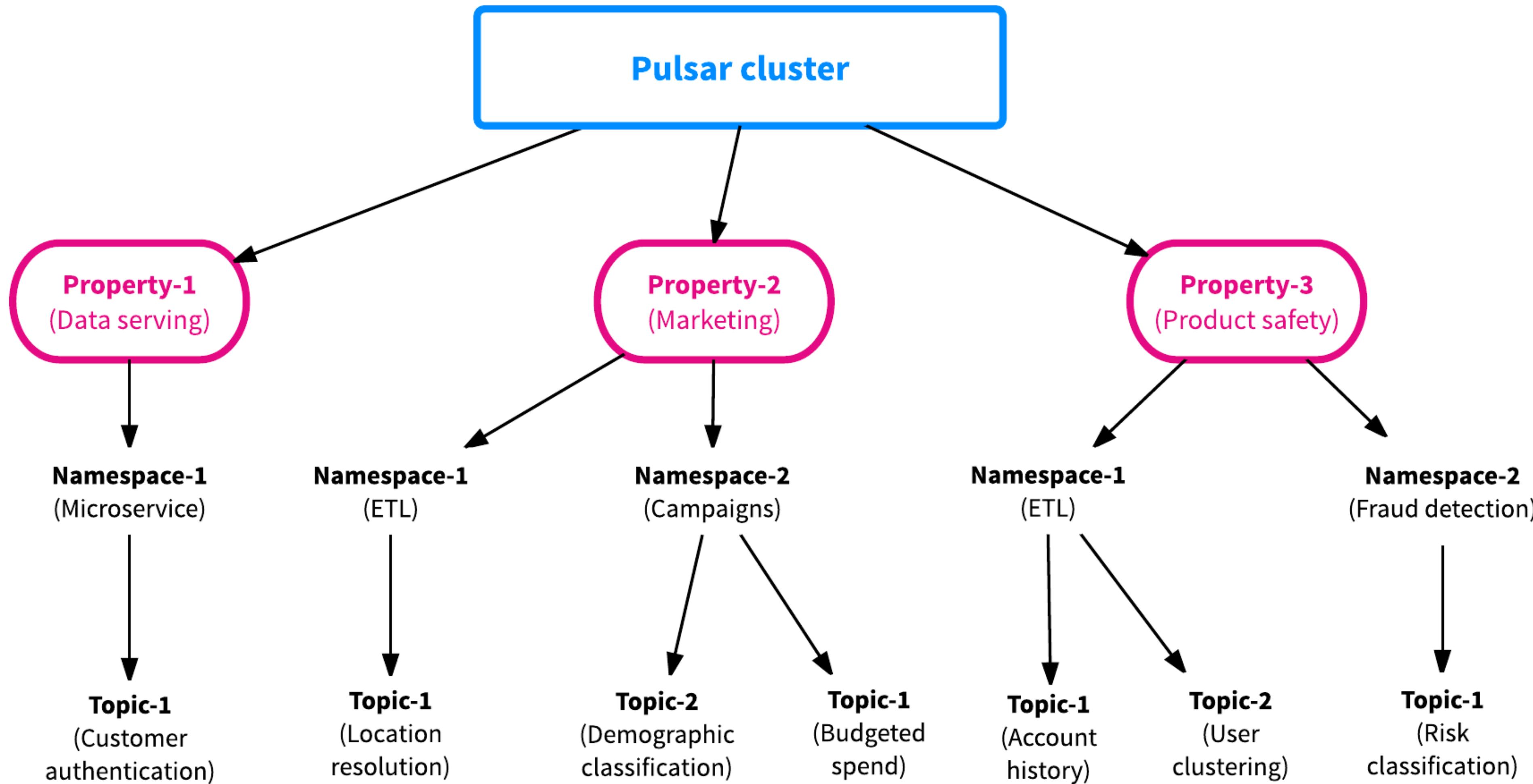
2011

**“Flexible Pub/Sub messaging
backed by durable log/stream storage”**

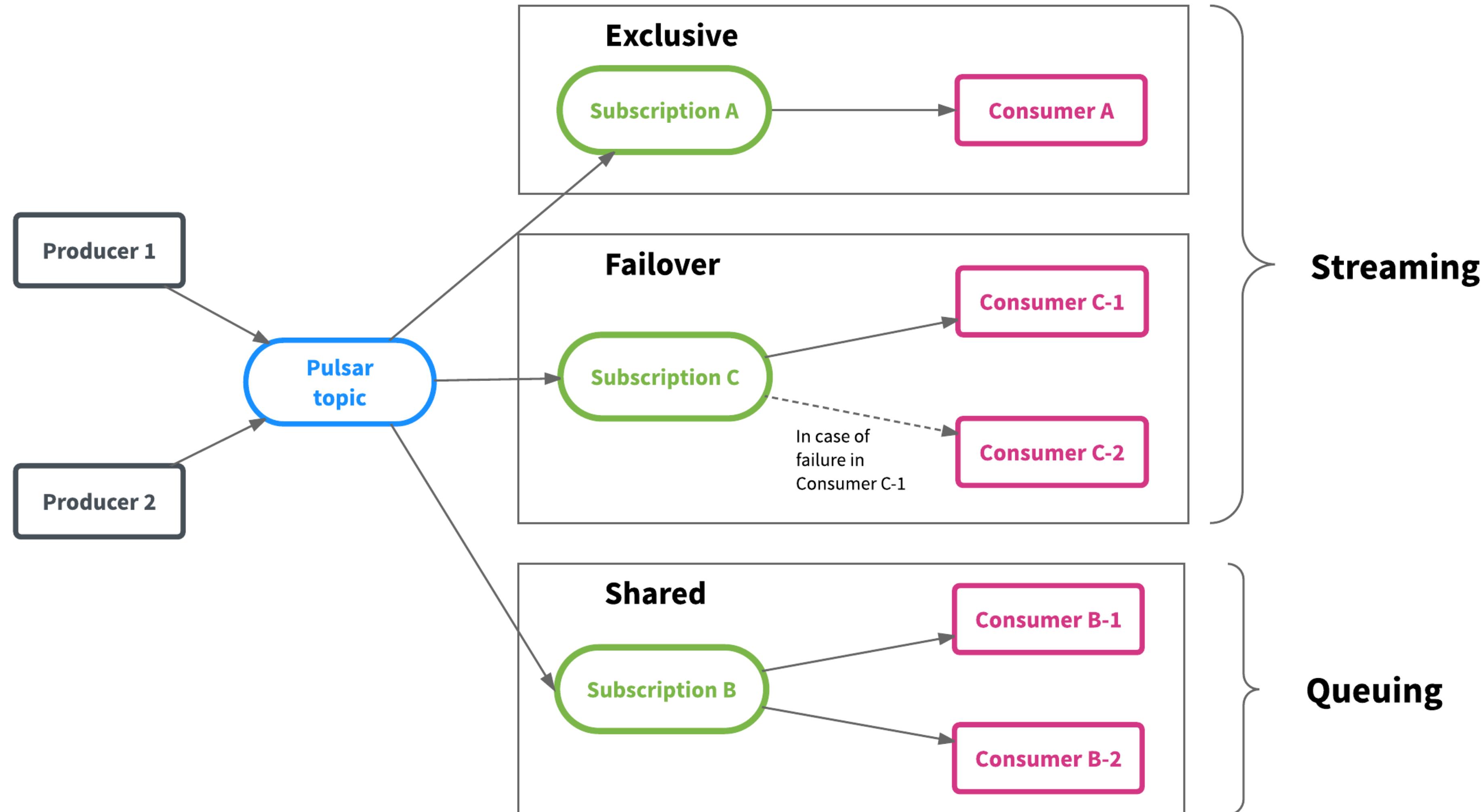
Pulsar - Pub/Sub



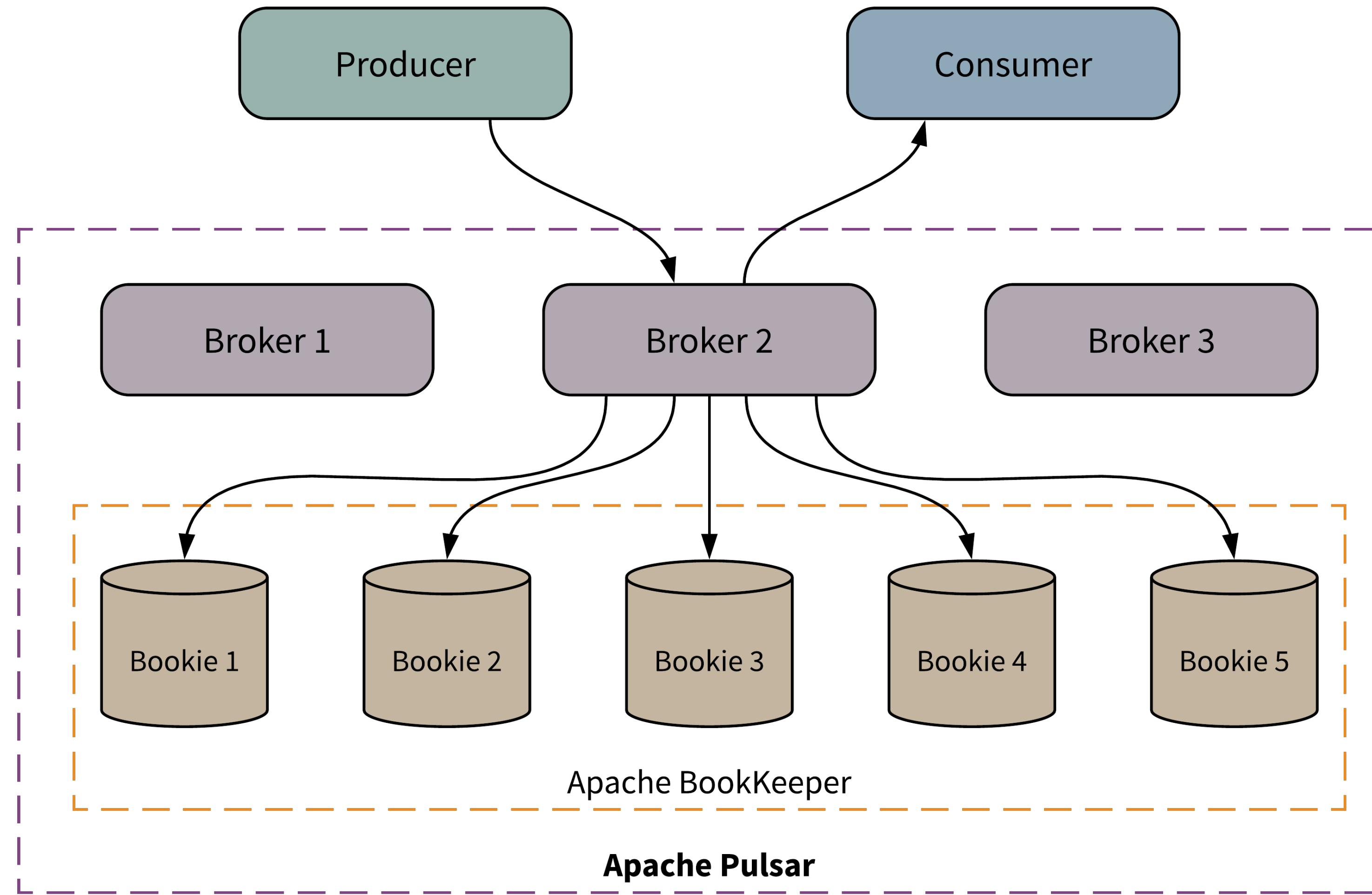
Pulsar - Multi Tenancy



Pulsar - Queue + Streaming



Pulsar - Cloud Native



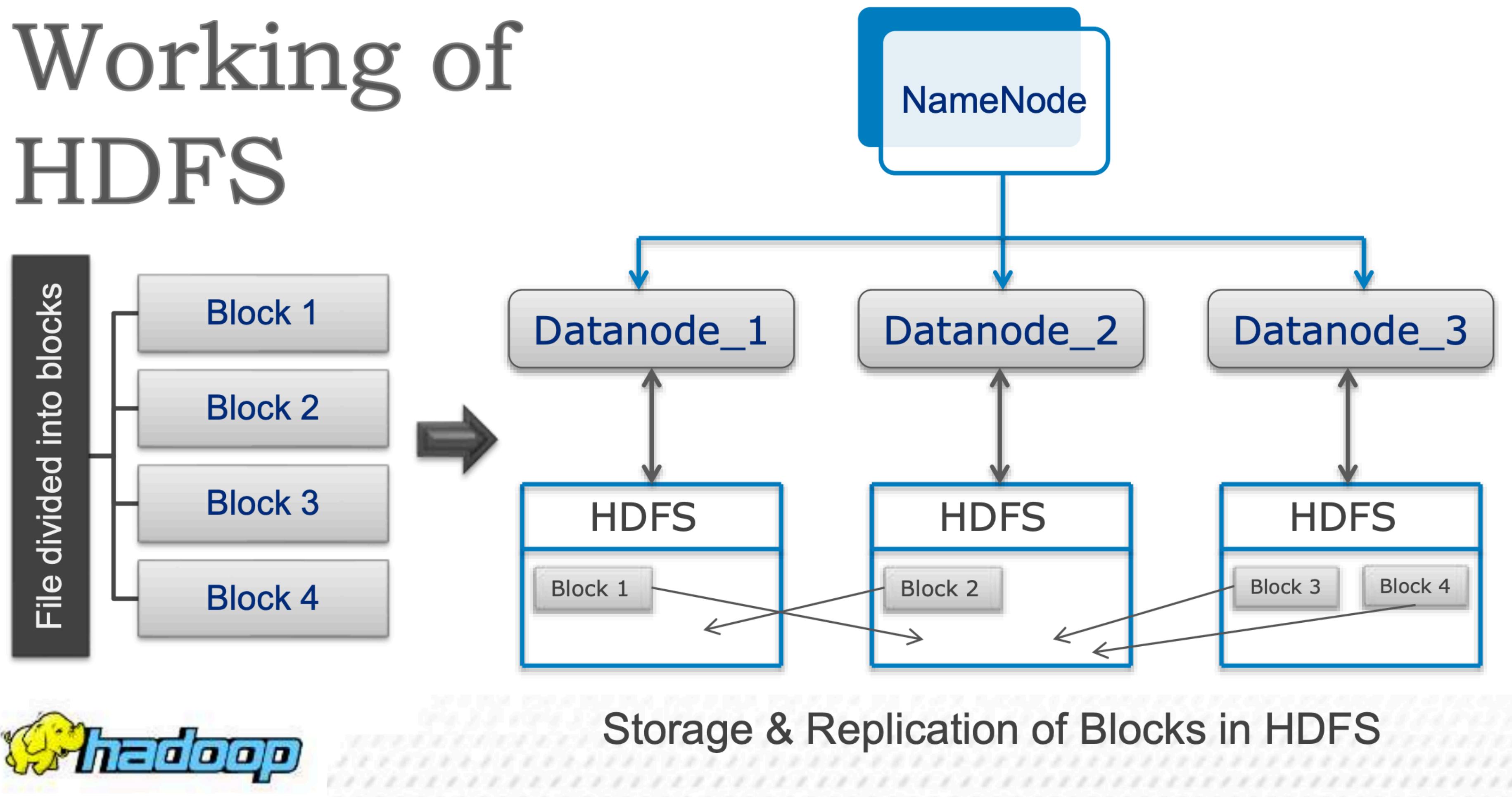
Layered Architecture

- Independent Scalability
- Instant Failure Recovery
- Balance-free on cluster expansions

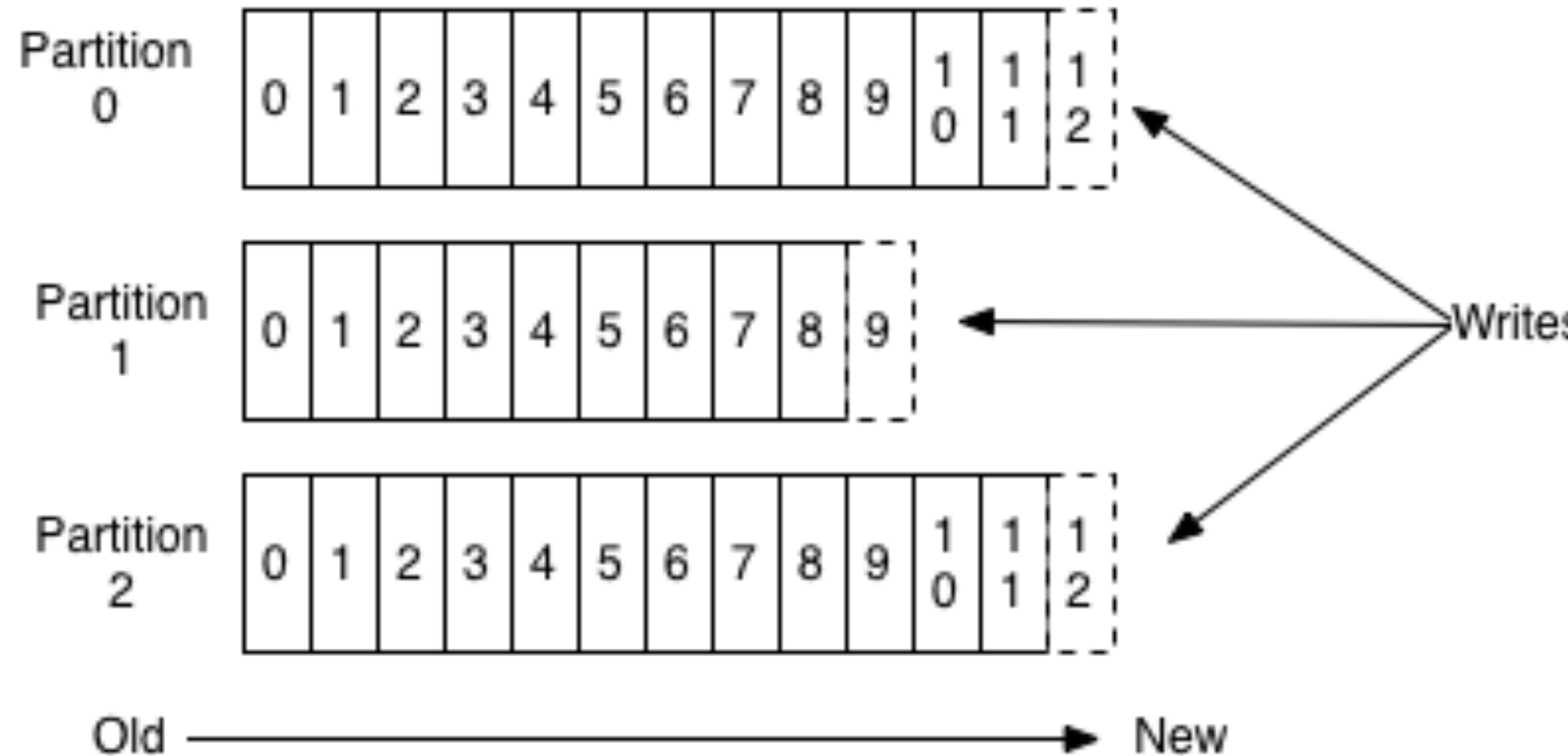
A Pulsar View on Data

Batch - HDFS

Working of HDFS



Stream - Pub/Sub

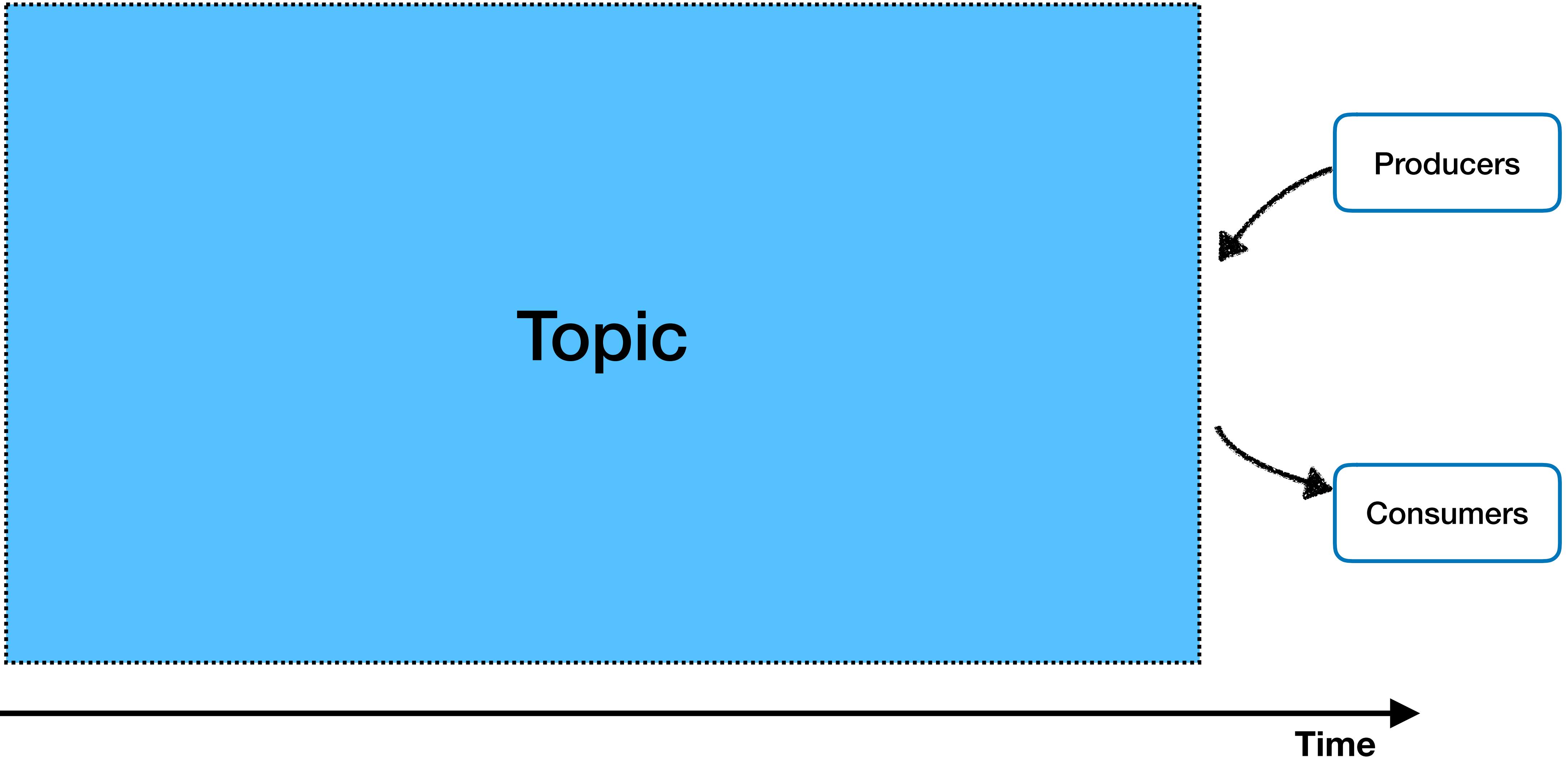


A Flink View on Computing

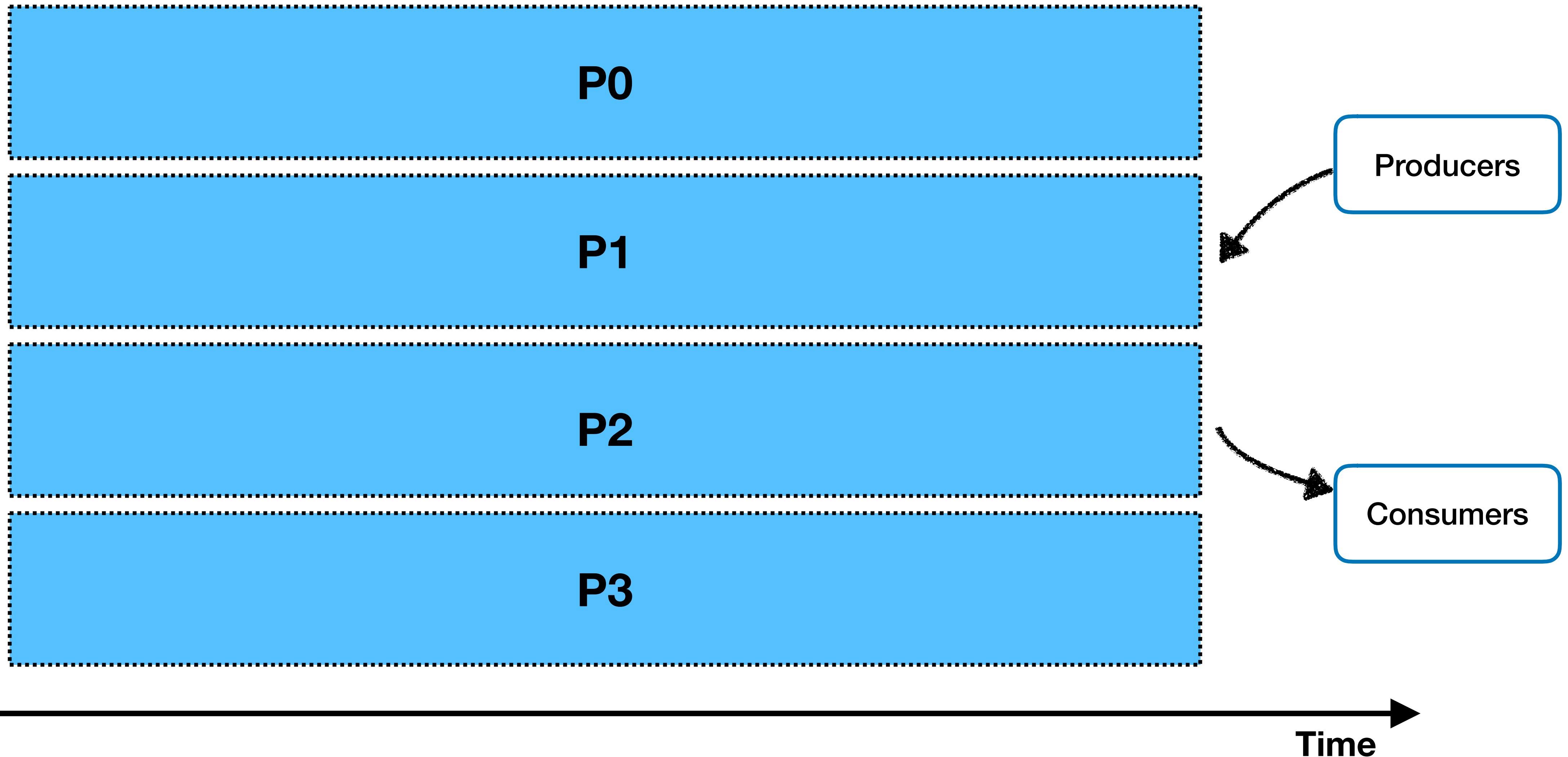
“Batch processing is a special case of
Stream processing”

Pulsar = *Segmented Stream*

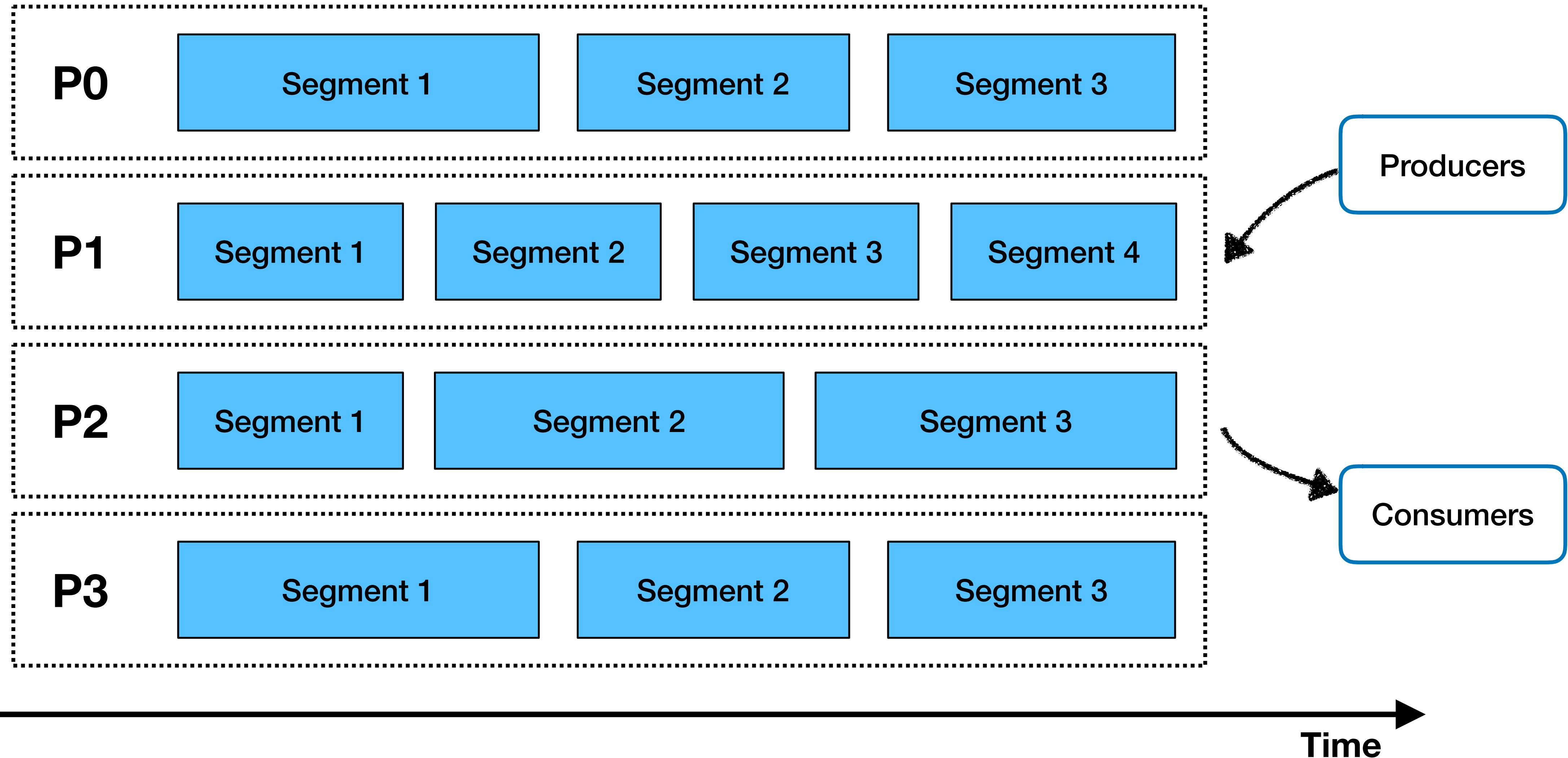
Topic



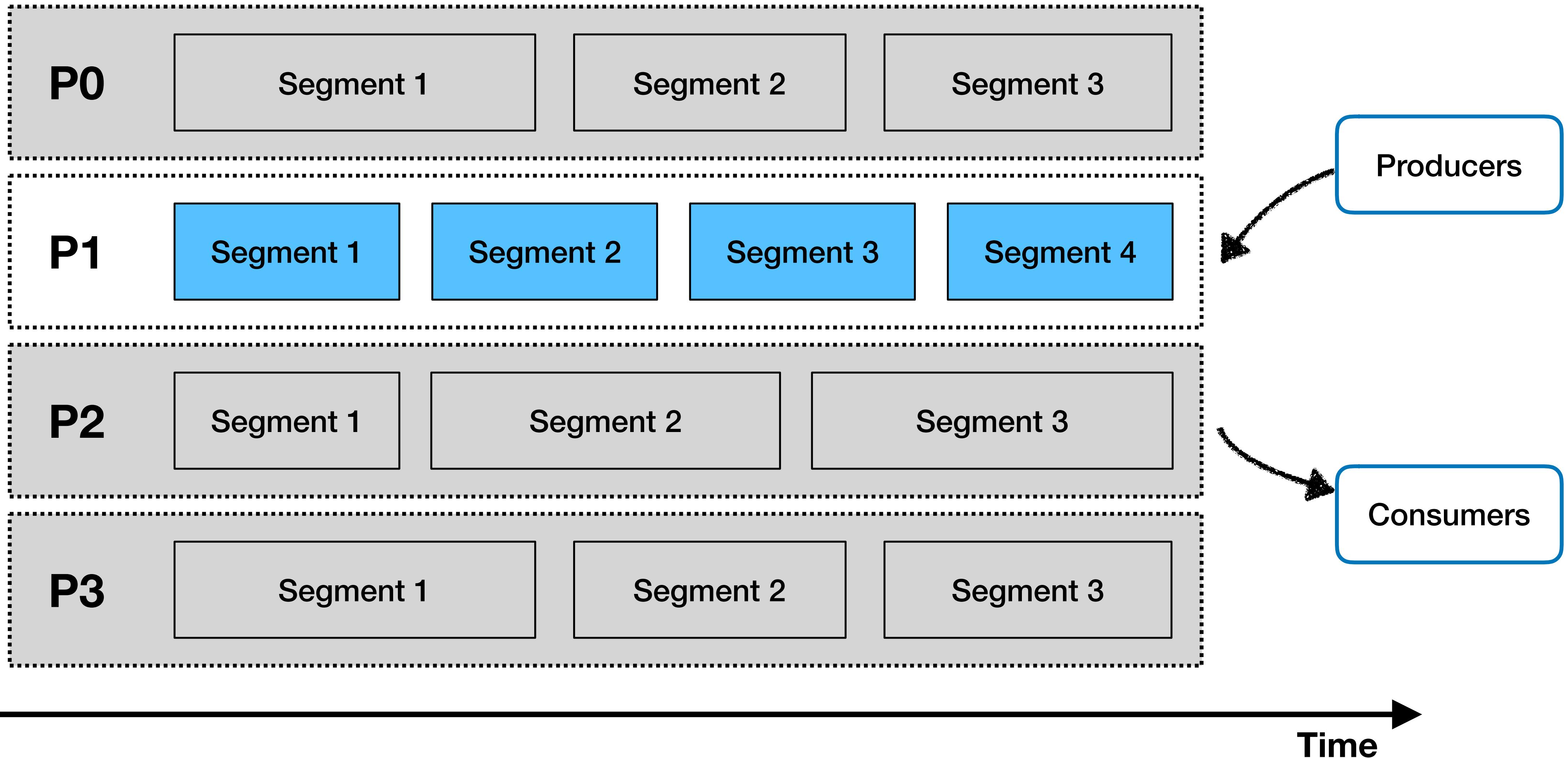
Partitions



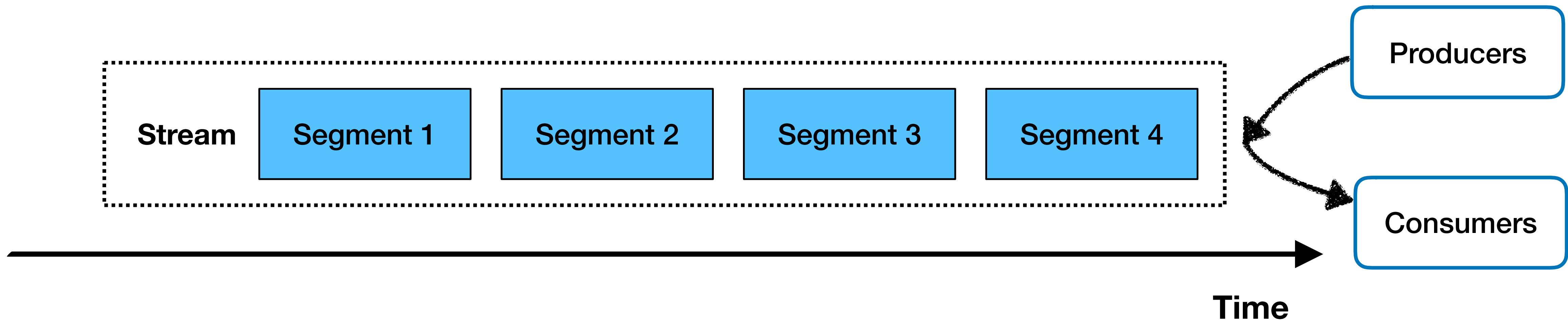
Segments



Stream



Stream



Segmented Stream

- Segmented Stream Systems
 - Apache Pulsar, Twitter EventBus, EMC Pravega
 - All Apache BookKeeper based
 - Used BK in a different way
 - Pulsar, EventBus - Uses BK as the segment store
 - Pravega - Uses BK as the journal only

Access Patterns

✓ Write

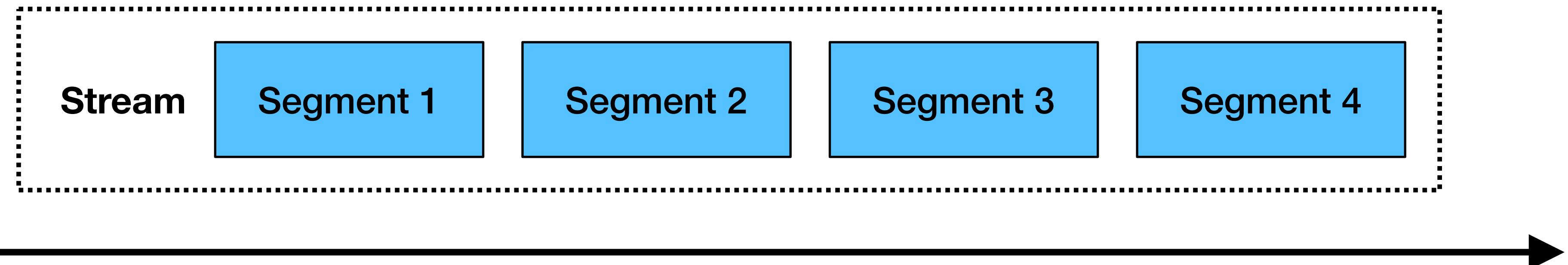
写

✓ Tailing Read

追尾

✓ Catchup Read

追读



Access Patterns

✓ Write

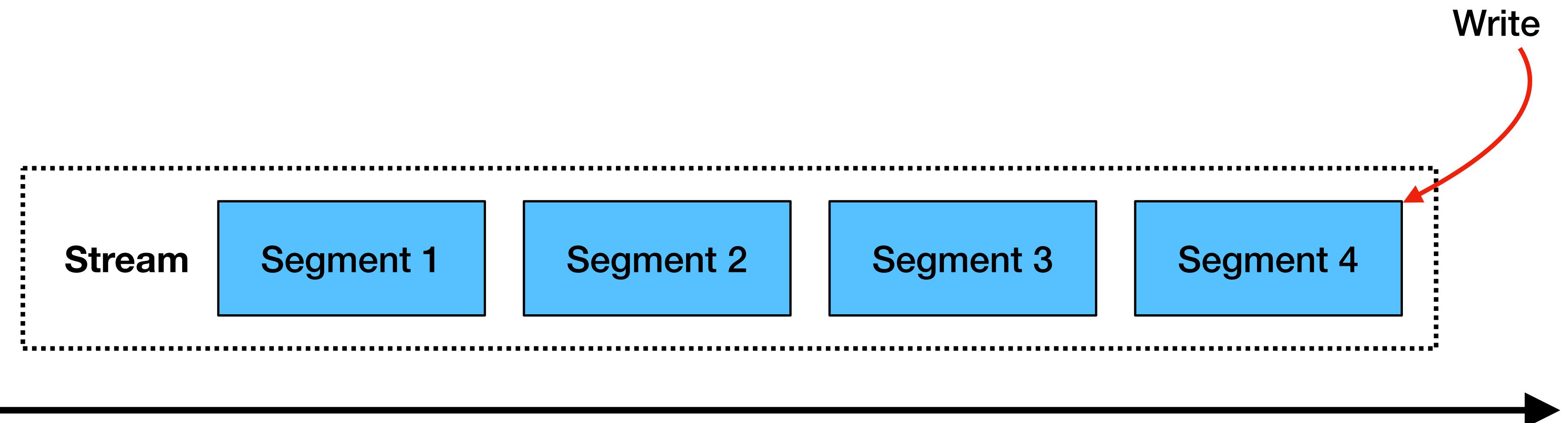
写

✓ Tailing Read

追尾

✓ Catchup Read

追读



Access Patterns

✓ Write

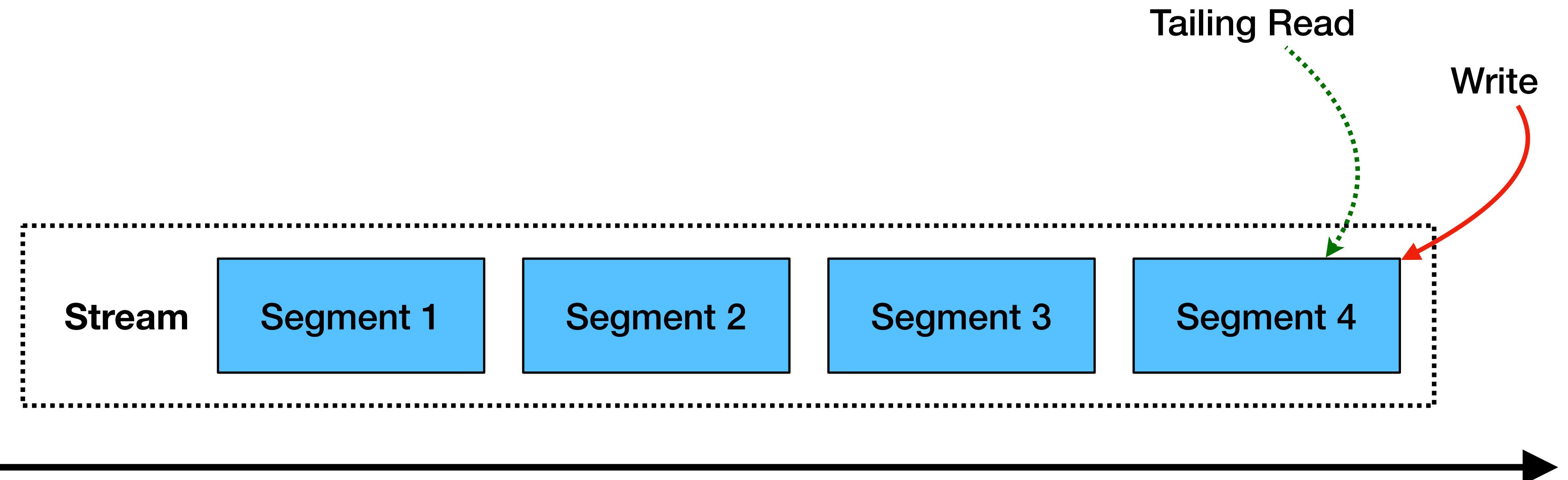
写

✓ Tailing Read

追尾

✓ Catchup Read

追赶读



Tailing Read

Write

Access Patterns

✓ Write

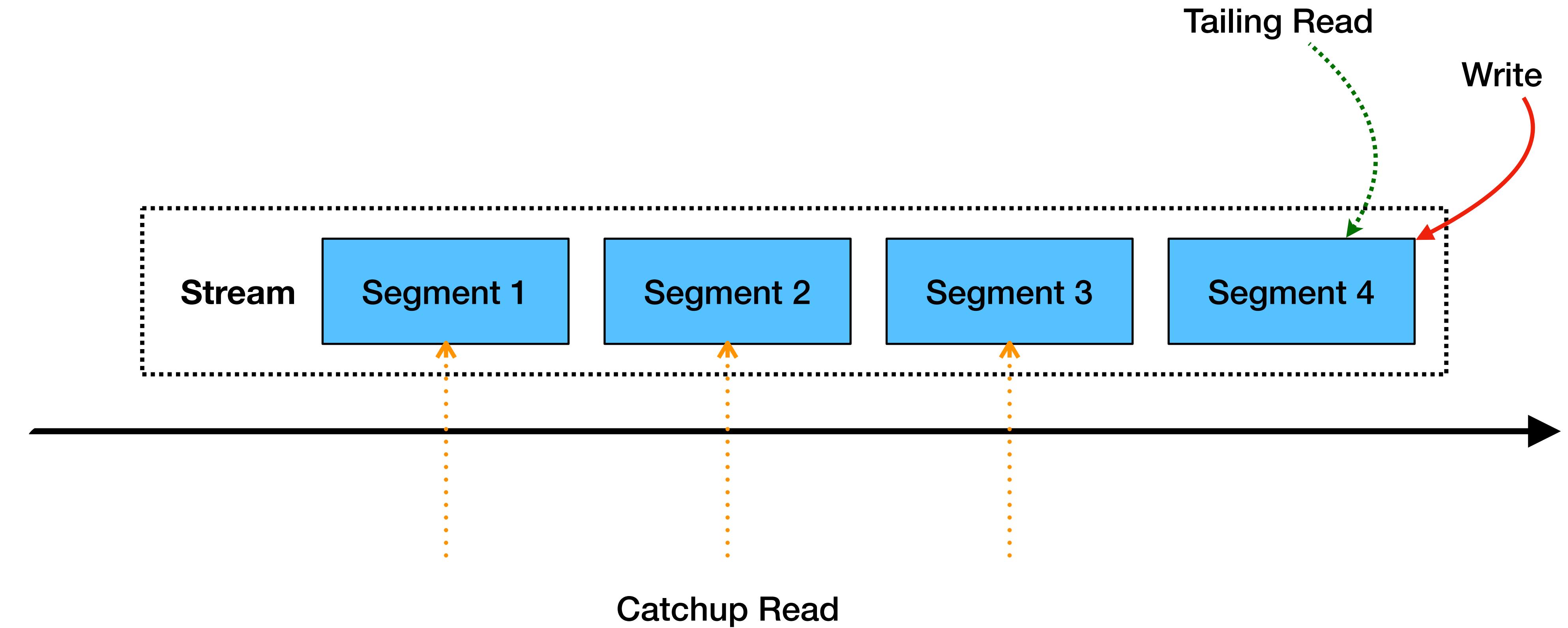
写

✓ Tailing Read

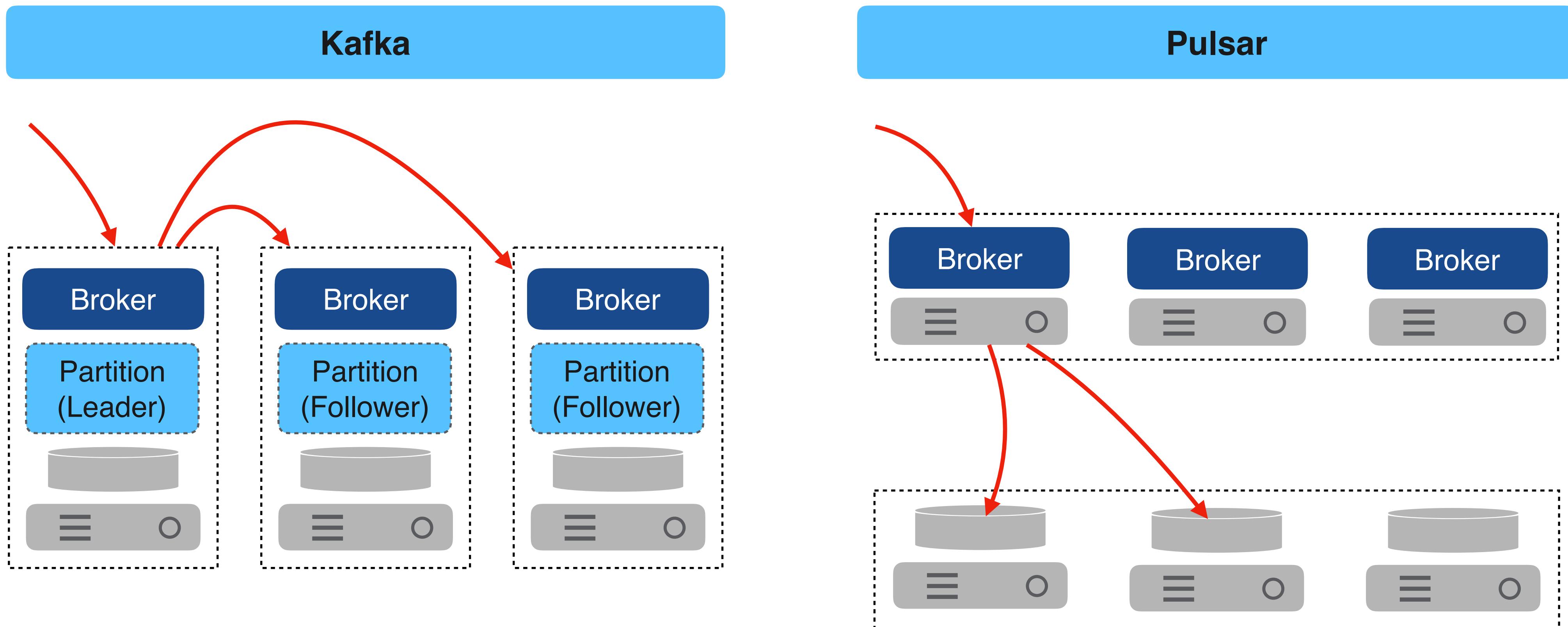
追尾

✓ Catchup Read

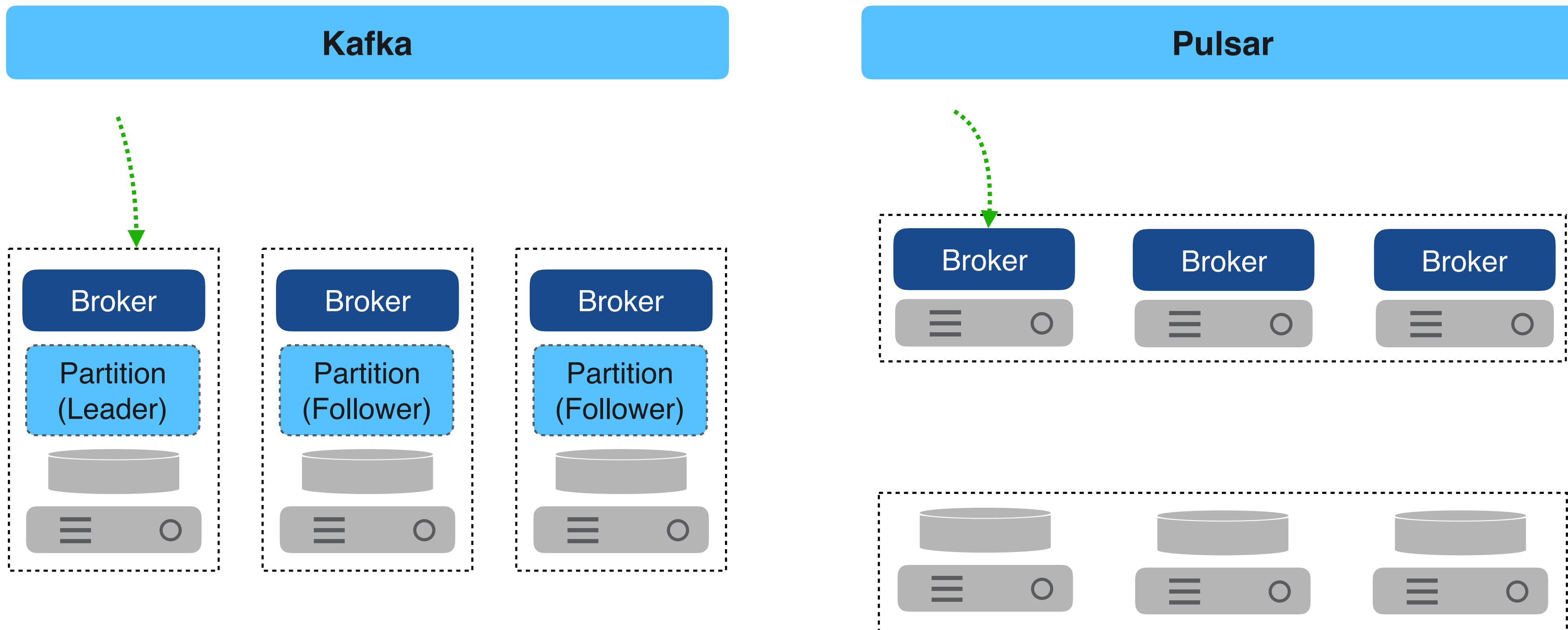
追赶读



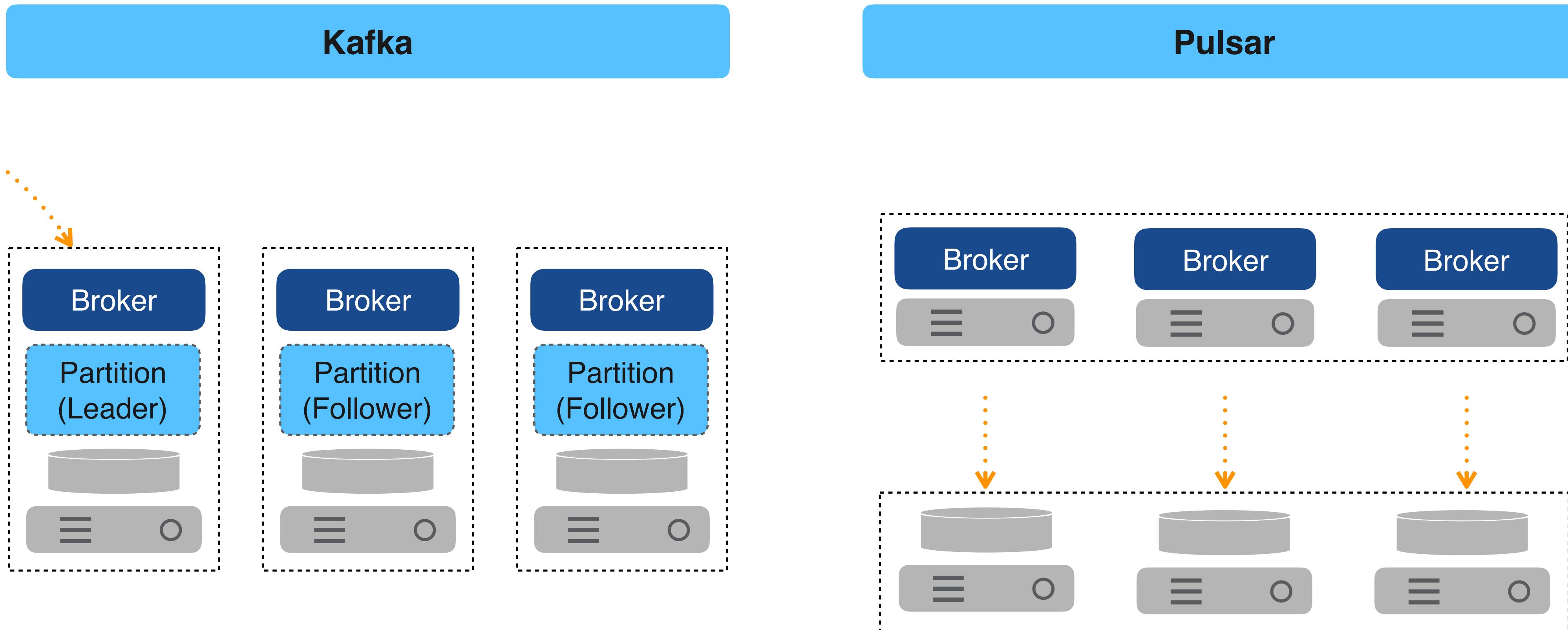
Write



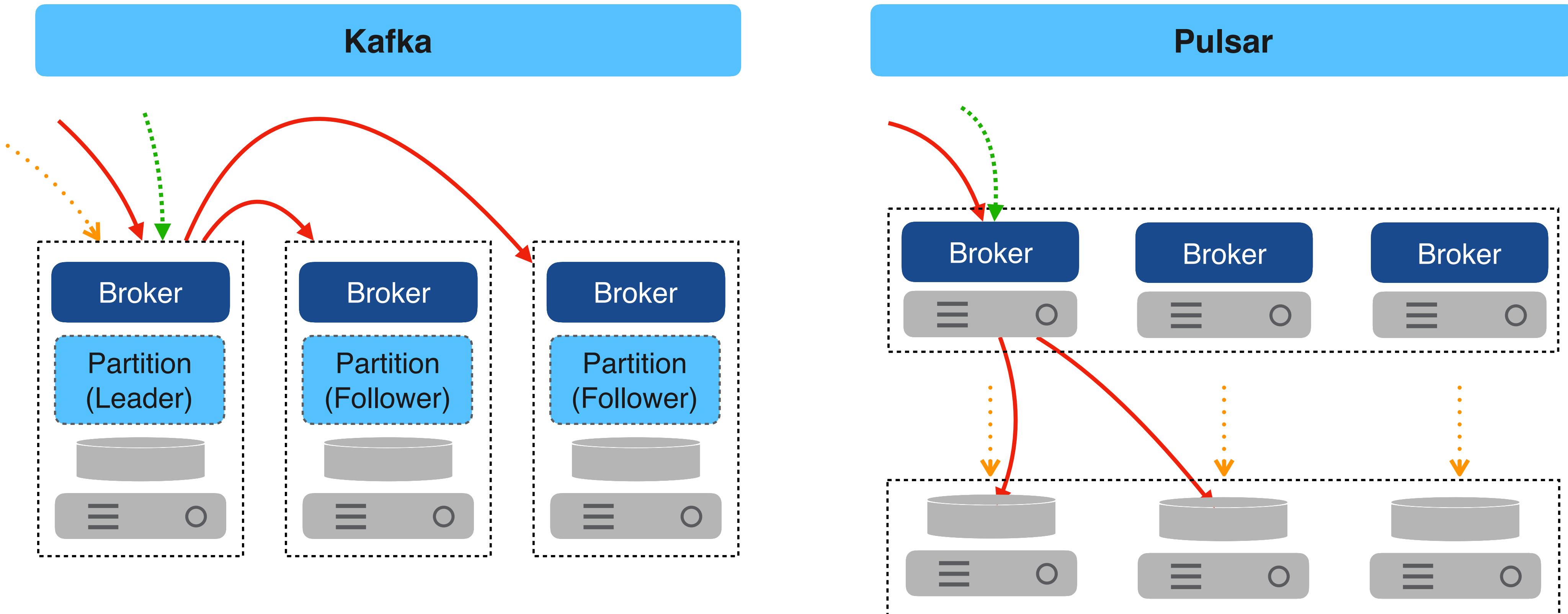
Tailing Read



Catchup Read



IO Isolation



Infinite Stream

✓ Write

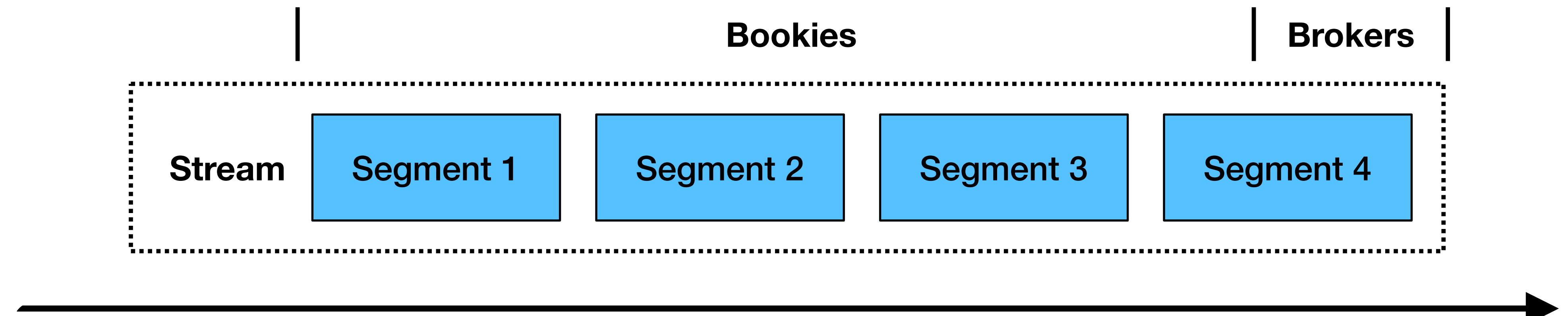
写

✓ Tailing Read

追尾

✓ Catchup Read

追读



Infinite Stream

✓ Write

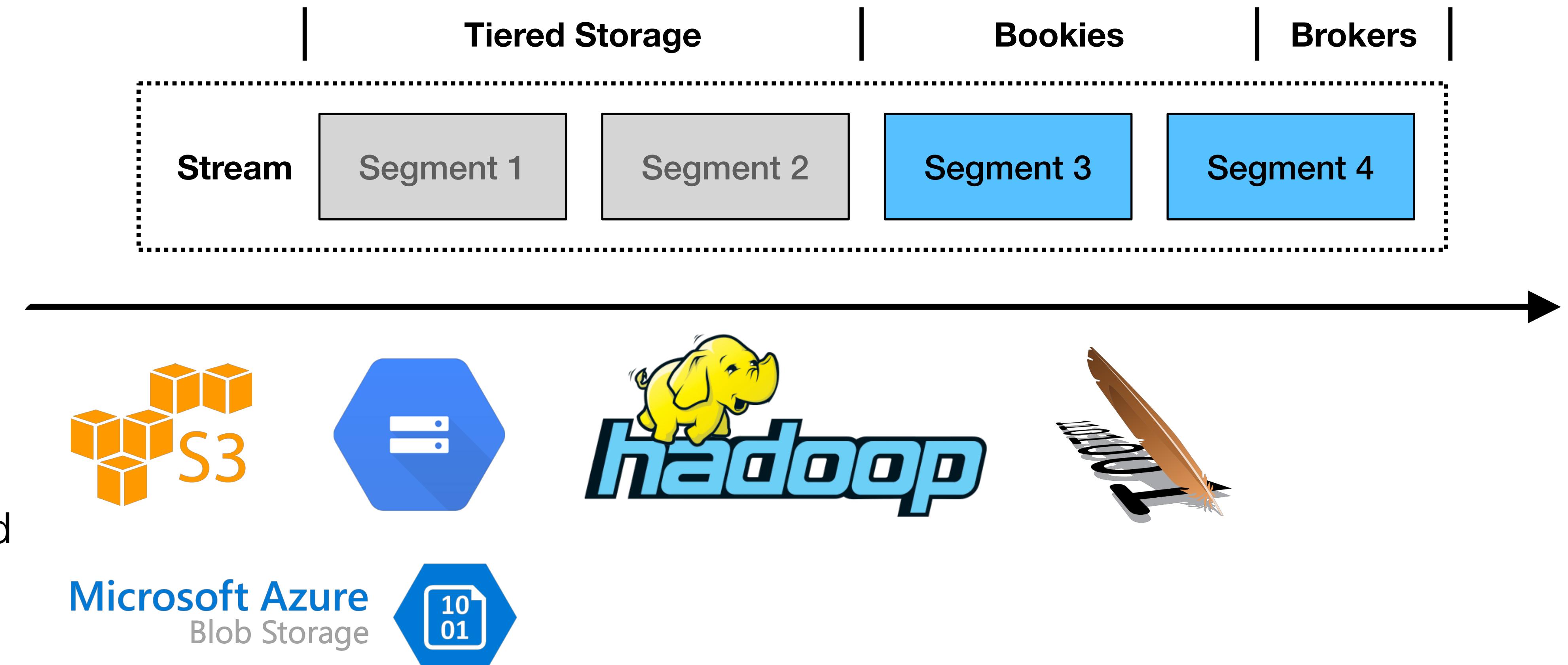
写

✓ Tailing Read

追尾

✓ Catchup Read

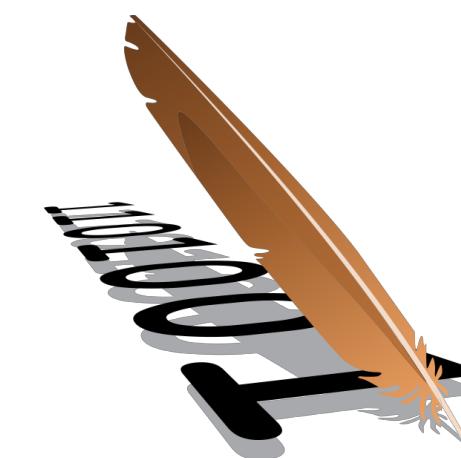
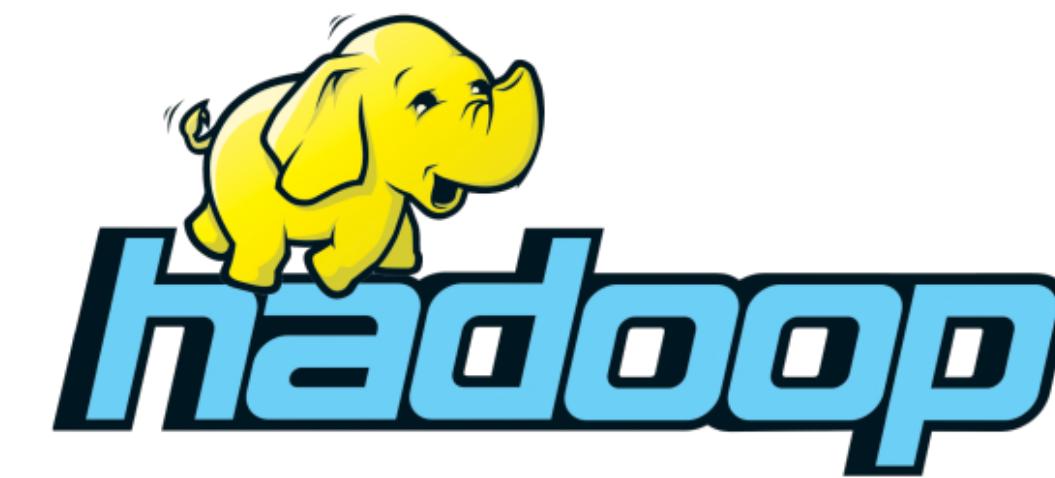
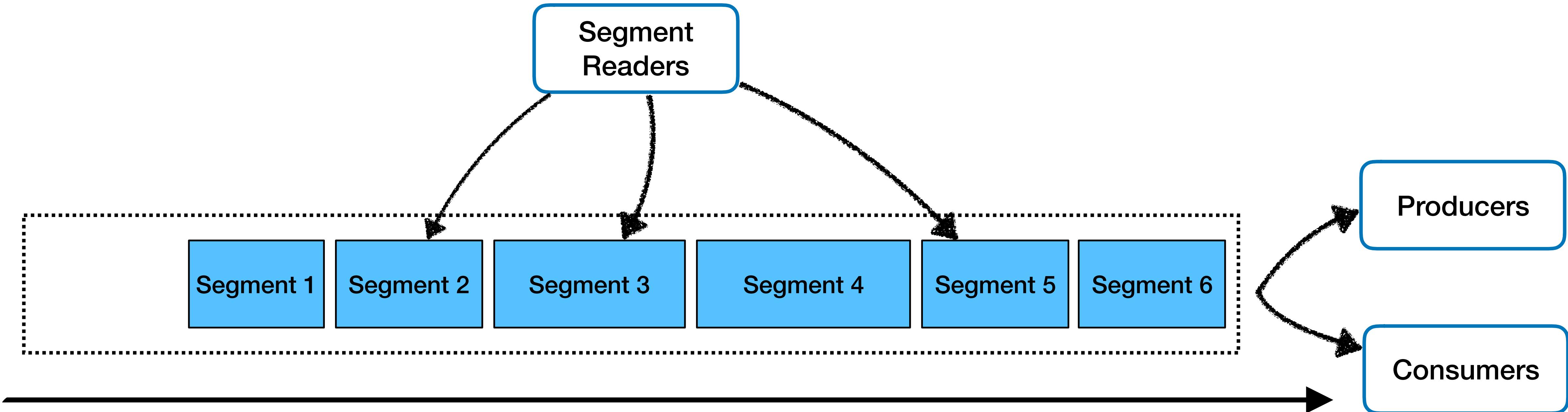
追读



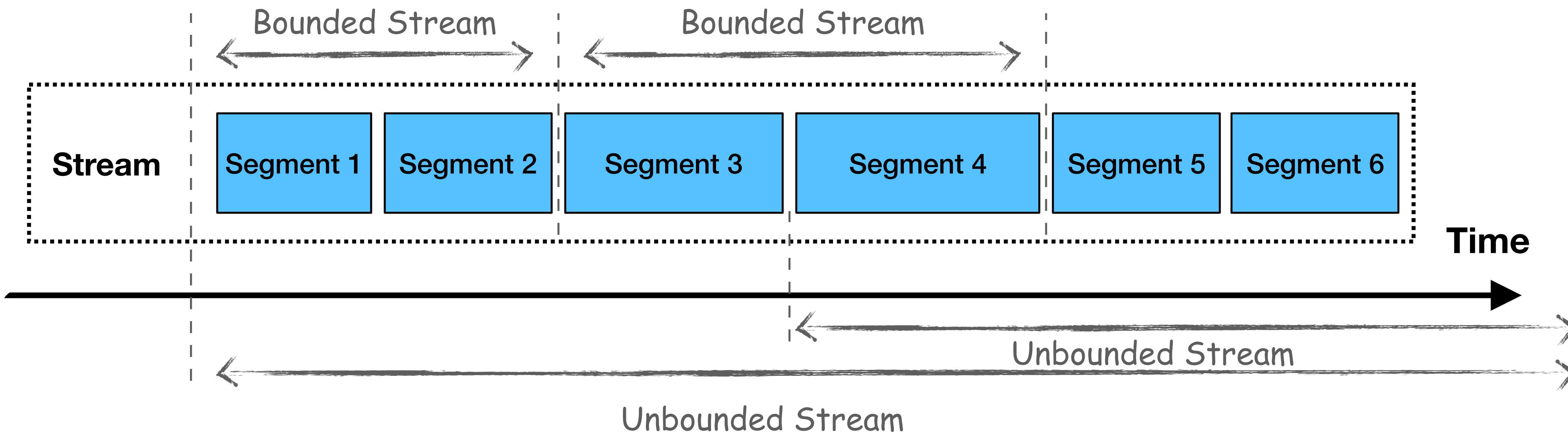
Tiered Storage

- Offloader
 - When: size-based, time-based, or triggered by pulsar-admin
 - How: copy a segment to tiered storage, and delete it from bookkeeper
 - Access: broker knows how to read the data back, or bypass read the offloaded segments directly
- Available Offloaders
 - Cloud Offloder : AWS, GCS, Azure, ...
 - HDFS, Ceph, ...

Stream as a Unified View on Data



Data Processing on Pulsar



When Flink meets Pulsar

Goals

- Flink + Pulsar
 - Streaming Connectors
 - Source Connectors
 - PulsarCatalog: Schema Integration
 - PulsarStateBackend
- Pulsar for the unified view of Data, Flink for the unified view of Computing

Done

Streaming Source -> Streaming Sink

```
PulsarSourceBuilder<String> builder = PulsarSourceBuilder.builder(new SimpleStringSchema())
    .serviceUrl(serviceUrl)
    .topic(inputTopic)
    .subscriptionName(subscription);
SourceFunction<String> src = builder.build();
DataStream<String> input = env.addSource(src);

DataStream<WordWithCount> wc = input
    .flatMap((FlatMapFunction<String, WordWithCount>) (line, collector) -> {
        for (String word : line.split("\\"s")) {
            collector.collect(new WordWithCount(word, 1));
        }
    })
    .returns(WordWithCount.class)
    .keyBy("word")
    .timeWindow(Time.seconds(5))
    .reduce((ReduceFunction<WordWithCount>) (c1, c2) ->
        new WordWithCount(c1.word, c1.count + c2.count));

if (null != outputTopic) {
    wc.addSink(new FlinkPulsarProducer<>(
        serviceUrl,
        outputTopic,
        wordWithCount -> wordWithCount.toString().getBytes(UTF_8),
        wordWithCount -> wordWithCount.word
    )).setParallelism(parallelism);
} else {
    // print the results with a single thread, rather than in parallel
    wc.print().setParallelism(1);
}
```

Streaming Source -> Streaming Table Sink

```
PulsarSourceBuilder<String> builder = PulsarSourceBuilder.builder(new SimpleStringSchema())
    .serviceUrl(serviceUrl)
    .topic(inputTopic)
    .subscriptionName(subscription);
SourceFunction<String> src = builder.build();
DataStream<String> input = env.addSource(src);

DataStream<WordWithCount> wc = input
    .flatMap((FlatMapFunction<String, WordWithCount>) (line, collector) -> {
        for (String word : line.split("\\s")) {
            collector.collect(
                WordWithCount.newBuilder().setWord(word).setCount(1).build()
            );
        }
    })
    .returns(WordWithCount.class)
    .keyBy(ROUTING_KEY)
    .timeWindow(Time.seconds(5))
    .reduce((ReduceFunction<WordWithCount>) (c1, c2) ->
        WordWithCount.newBuilder().setWord(c1.getWord()).setCount(c1.getCount() + c2.getCount()).build()
    );

tableEnvironment.registerDataStream("wc",wc);
Table table = tableEnvironment.sqlQuery("select word, `count` from wc");
table.printSchema();
TableSink sink = null;
if (null != outputTopic) {
    sink = new PulsarAvroTableSink(serviceUrl, outputTopic, ROUTING_KEY, WordWithCount.class);
} else {
    // print the results with a csv file
    sink = new CsvTableSink("./examples/file", "|");
}
table.writeToSink(sink);
```

Batch Sink

```
// create PulsarOutputFormat instance
final OutputFormat pulsarOutputFormat =
    new PulsarOutputFormat(serviceUrl, topic, wordWithCount -> wordWithCount.toString().getBytes());

// create DataSet
DataSet<String> textDS = env.fromElements(EINSTEIN_QUOTE);

// convert sentences to words
textDS.flatMap(new FlatMapFunction<String, WordWithCount>() {
    @Override
    public void flatMap(String value, Collector<WordWithCount> out) throws Exception {
        String[] words = value.toLowerCase().split(" ");
        for(String word: words) {
            out.collect(new WordWithCount(word.replace(".", ""), 1));
        }
    }
})

// filter words which length is bigger than 4
.filter(wordWithCount -> wordWithCount.word.length() > 4)

// group the words
.groupBy(new KeySelector<WordWithCount, String>() {
    @Override
    public String getKey(WordWithCount wordWithCount) throws Exception {
        return wordWithCount.word;
    }
})

// sum the word counts
.reduce(new ReduceFunction<WordWithCount>() {
    @Override
    public WordWithCount reduce(WordWithCount wordWithCount1, WordWithCount wordWithCount2) throws Exception {
        return new WordWithCount(wordWithCount1.word, wordWithCount1.count + wordWithCount2.count);
    }
})

// write batch data to Pulsar
.output(pulsarOutputFormat);
```

Case Study - Zhaopin.com

Zhaopin.com

Zhaopin.com is the biggest online recruitment service provider in China

Zhaopin.com provides job seekers a comprehensive resume service, latest employment, and career development related information, as well as in-depth online job search for positions throughout China

Zhaopin.com provides professional HR services to over 2.2 million clients and its average daily page views are over 68 million.

Job Search

The screenshot shows the Zhaopin.com website interface for a job search. The search bar at the top has '职位' (Position) selected and 'java' typed in. Below the search bar are navigation links: 首页 (Home), 北京站 (Beijing Station), 校园招聘 (Campus Recruitment), 高端职位 (High-end Positions), 海外招聘 (Overseas Recruitment), 智联教育 (ZhiLian Education), and 智联招聘 (ZhiLian Recruitment). The main search results area displays two job listings:

- 云计算/虚拟化运维工程师** (置顶) | **1万-2万/月**
北京 | 1-3年 | 本科
绩效奖金 年终分红 加班补助 房补 带薪年假
新华网股份有限公司
- Java开发工程师(006283)** (置顶) | **2万-4万/月**
北京 | 5-10年 | 大专
健身俱乐部 五险一金 年底双薪 餐补 带薪年假
神州优车股份有限公司

Below the job listings, there is a sidebar with a recruitment advertisement for English training.

智能匹配 (highlighted in blue) | **薪酬最高** | **最新发布**

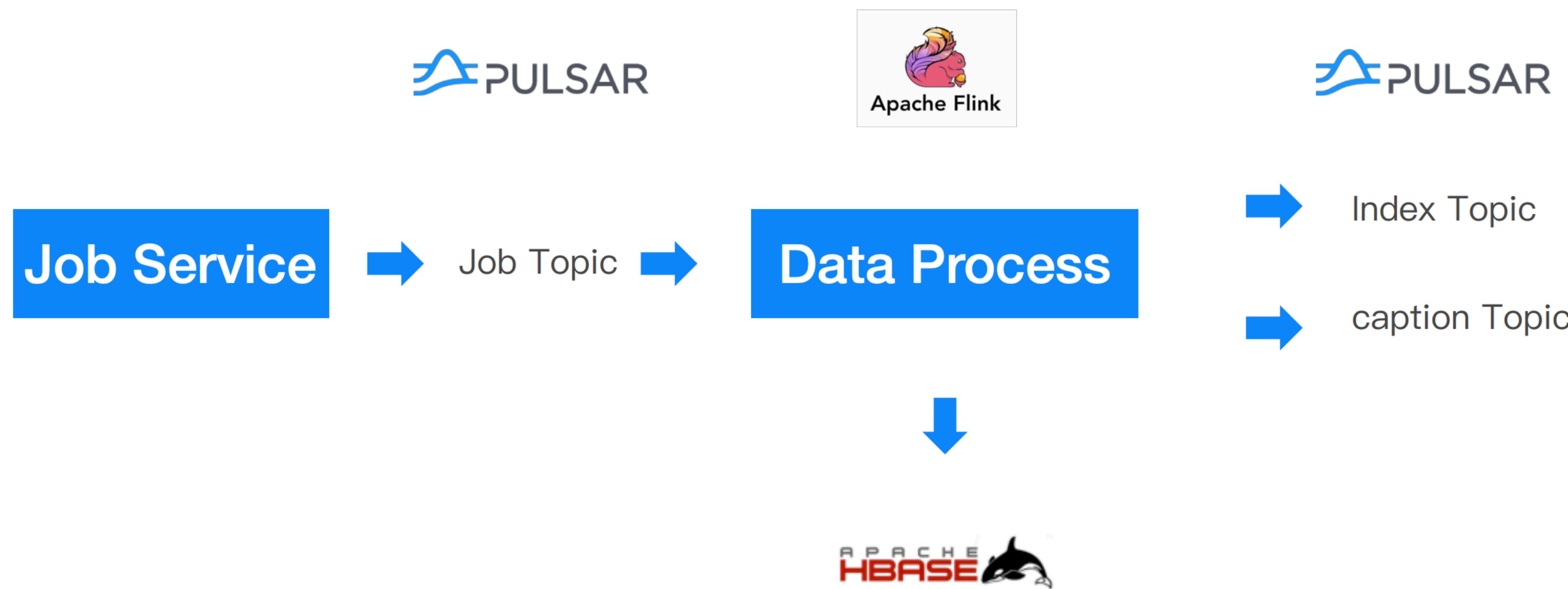
Job results:
1. **java开发工程师 (支付方向)** (顶 急)
10K-18K | 北京 | 1-3年 | 本科
中金支付有限公司 [ZP名企]
国企 | 100-499人
14薪 节日福利 置顶

2. **云计算/虚拟化运维工程师** (顶 急)
10K-20K | 北京 | 1-3年 | 本科
新华网股份有限公司 [最佳雇主]
国企 | 1000-9999人
绩效奖金 年终分红 加班补助 房补 带薪年假 置顶

The screenshot shows a job search interface with a search bar containing 'java'. The results page displays a single job listing:

java高级开发工程师 | **1万-2万/月**
北京 | 3-5年 | 本科
绩效奖金 交通补助 创业公司 每年多次调薪
ERP Java Node.js J2EE 数据库 Linux
北京琪佳淼网络科技有限公司

Data Processing



Metrics

50+ Namespaces

3000+ Topics

6+ billion Messages per day

3TB Storage per day

20+ Core Services

Roadmap

Batch Source

- Read Segments in Parallel
- Bypass Brokers
- Access tiered storage directly
- Scan Trimmer
 - Select Segments by Publish Time

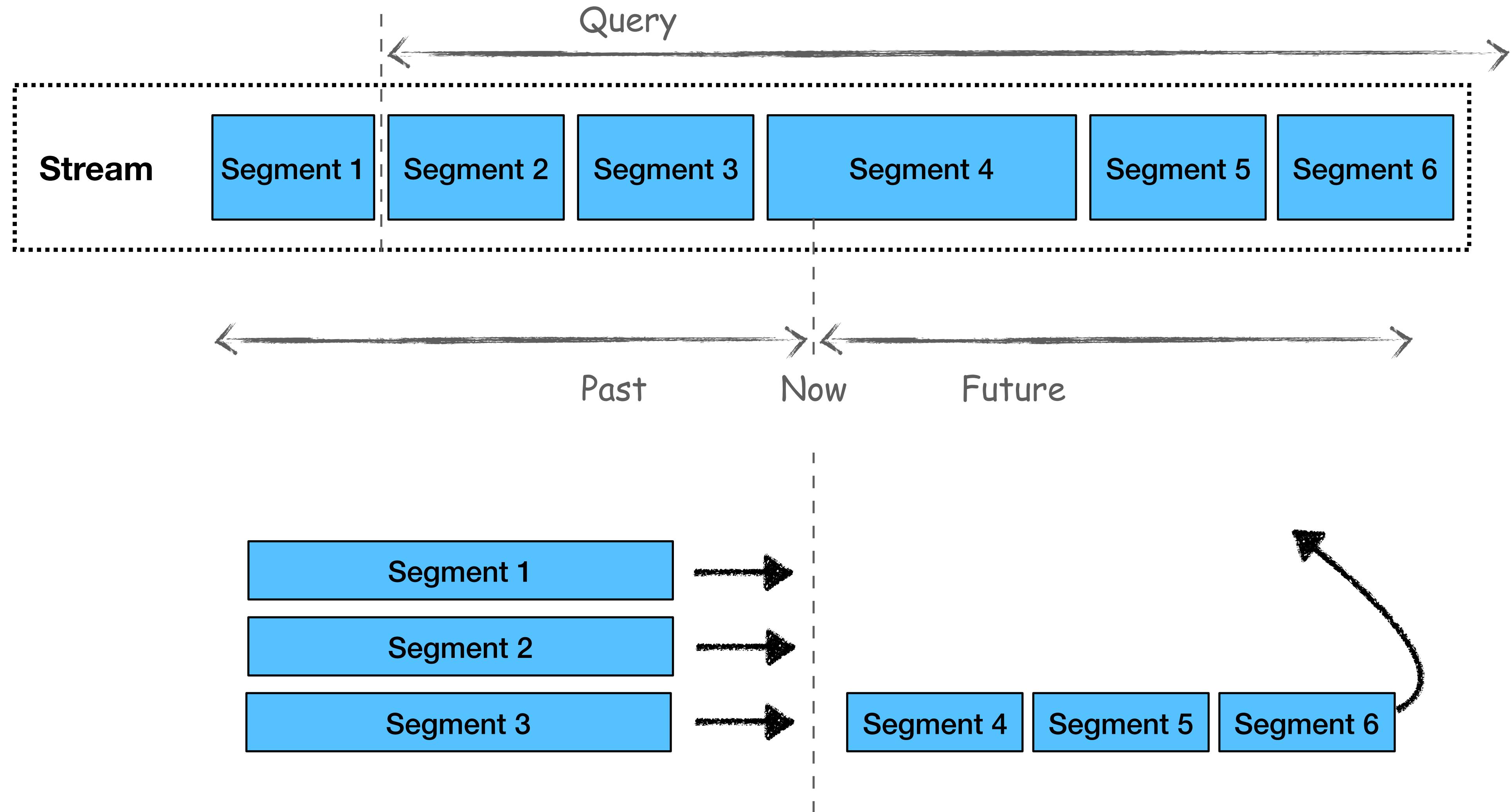
Schema Integration

- Pulsar has builtin schema registry
 - Primitive types, Avro, Json, Protobuf, ...
- Schema Evolution & Multi-versioning schemas
- PulsarCatalog

State Backend

- BookKeeperStateBackend
 - Save State as Segments to BookKeeper

Unified Data Processing



Community

- ✓ Twitter: **@apache_pulsar**
- ✓ Wechat Subscription: **ApachePulsar**
- ✓ Mailing Lists

dev@pulsar.apache.org, users@pulsar.apache.org

- ✓ Slack
- <https://apache-pulsar.slack.com>

- ✓ Localization
- <https://crowdin.com/project/apache-pulsar>

- ✓ Github
- <https://github.com/apache/pulsar>
- <https://github.com/apache/bookkeeper>

Creating millions of user sessions using complex event processing

Prem Santosh

pxu@yelp.com

[@premsantosh](https://twitter.com/premsantosh)



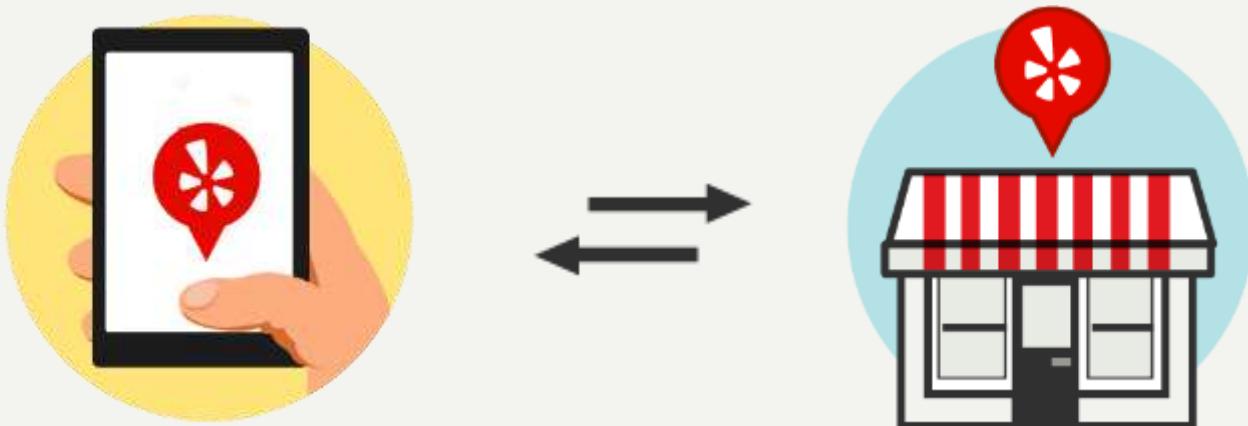
Agenda

- Motivation
- Sessionizer Specs
- Sessionizer
- Issues
 - Bot traffic
 - Duplicate sessions
 - Stale topics
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State



Yelp's Mission

Connecting people with great
local businesses.



Agenda

- Motivation
- Sessionizer Specs
- Sessionizer
- Issues
 - Bot traffic
 - Duplicate sessions
 - Stale topics
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State

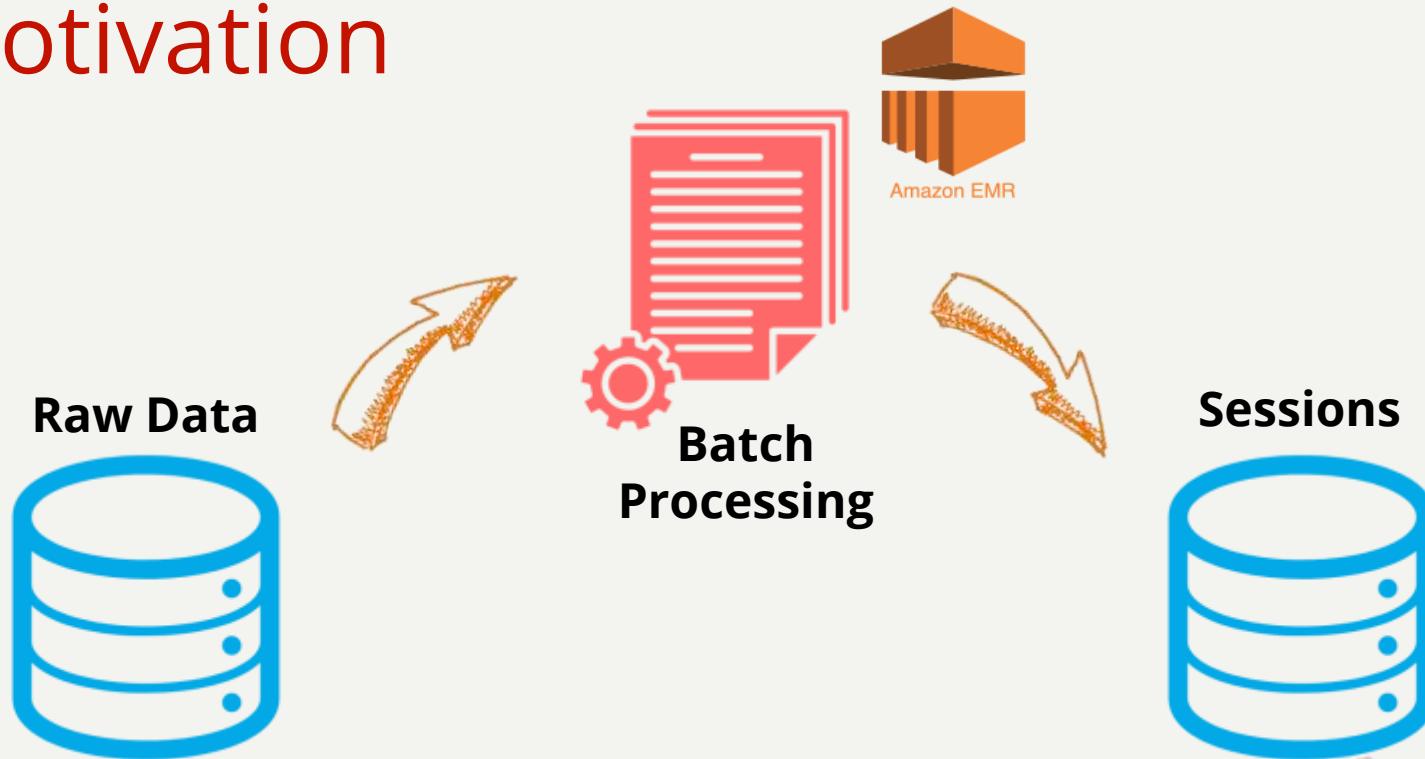


Why Sessions?

- Fundamental element of engagement with Yelp
- Understand intent of users
- Generate other product metrics using sessions



Motivation



Motivation

- Slow
- Expensive
- One day late
- Sessions crossing midnight



Agenda

- Motivation
- Sessionizer Specs
- Sessionizer
- Issues
 - Bot traffic
 - Duplicate sessions
 - Stale topics
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State



Sessionizer Specs

- 1 Master
 - M4.xlarge (4 CPU, 16GB)
- 6 Core
 - M4.16xlarge (6 * 64 CPUs, 6 * 256GB)
- 6 Input kafka topics
- Total input throughput ~25K msg/sec



Agenda

- Motivation
- Sessionizer Specs
- **Sessionizer**
- Issues
 - Bot traffic
 - Duplicate sessions
 - Stale topics
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State



User Session



A photograph showing the interior of a restaurant. The walls are decorated with blue and white patterned wallpaper. There are two hanging lantern-style light fixtures. In the foreground, there's a dark surface, possibly a bar or counter.

Pizza Rollio

★★★★★ 36 reviews

\$\$ · Salad, Coffee & Tea, Pizza

(917) 261-6779

261 W 18th St

Chelsea

"Agree with the last review, very atmospheric and Feels like in Italy inside! **Pizza** crust is a number one for me, your place got it just right. Were there last Thursday on our..." [read more](#)

Start Order

Offers takeout and delivery

A photograph of a single slice of pepperoni pizza on a white plate. The pizza has a thin crust and is topped with melted cheese and pepperoni. A small bowl of red dipping sauce is placed next to the plate.

Sauce Pizzeria

★★★★★ 98 reviews

Pizza, Italian

(646) 983-4007

345 E 12th St

East Village

"I've been eyeing this place for the last 3 months since it opened. Of course I heard about it from El Prez and Frankie from barstool. 9.1. They gave it super high praise and I just..." [read more](#)

Start Order

Offers takeout and delivery

User Session

Sauce Pizzeria Claimed



98 reviews

Pizza, Italian

[Edit](#)

[★ Write a Review](#)

[Add Photo](#)

[Share](#)

[Save](#)

Review Highlights



"I am excited to go back and try their [Al Pastor Pie](#) :) Oh, and did I mention they serve soft serve?!" in 2 reviews

[\\$25 Al Pastor Pie](#)



"Tried 4 amazing slices with my bf...my favorite was the [upside](#) down cheese but everything we had was delicious." in 21 reviews



"Thin, [crispy](#) crust, a well-balanced sauce, and this amazingly salty mozzarella all in perfect proportion." in 10 reviews

Order Food

Delivery Takeout

\$1.99+ fee • \$0 min • 45-55 mins

Delivery Address

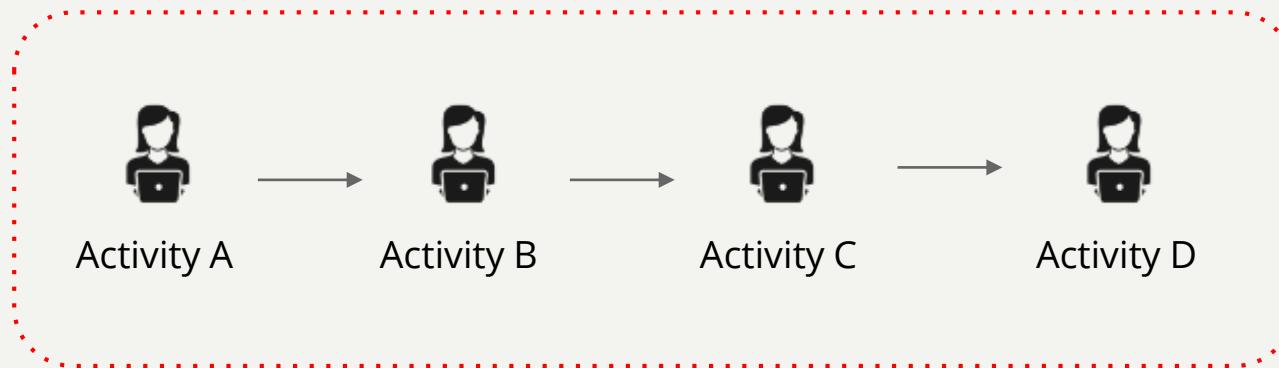
Enter delivery address

[View Menu](#)

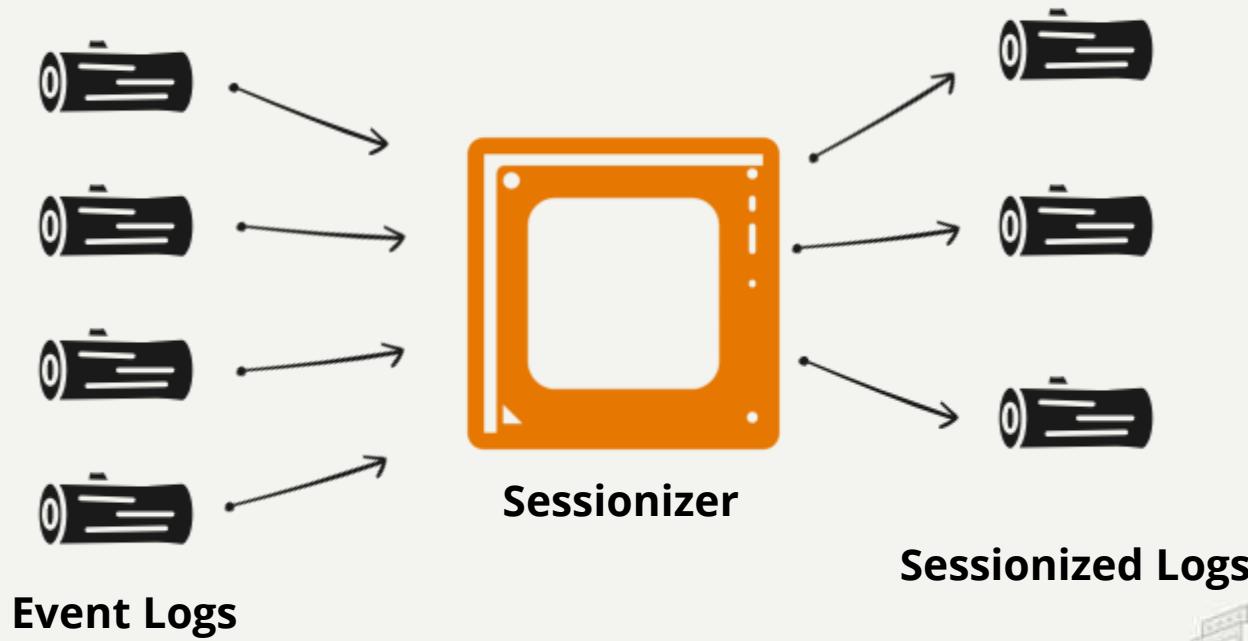


User Session

UserID = XYZ

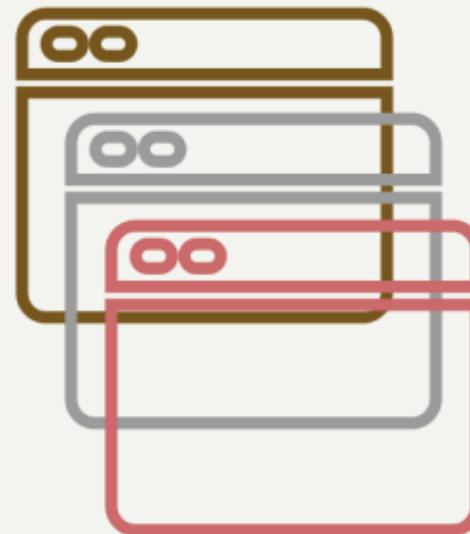
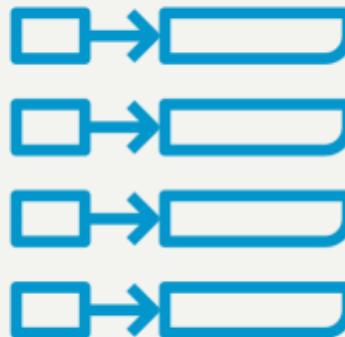


Sessionizer



Sessionizer Operators

KeyBy: UserID



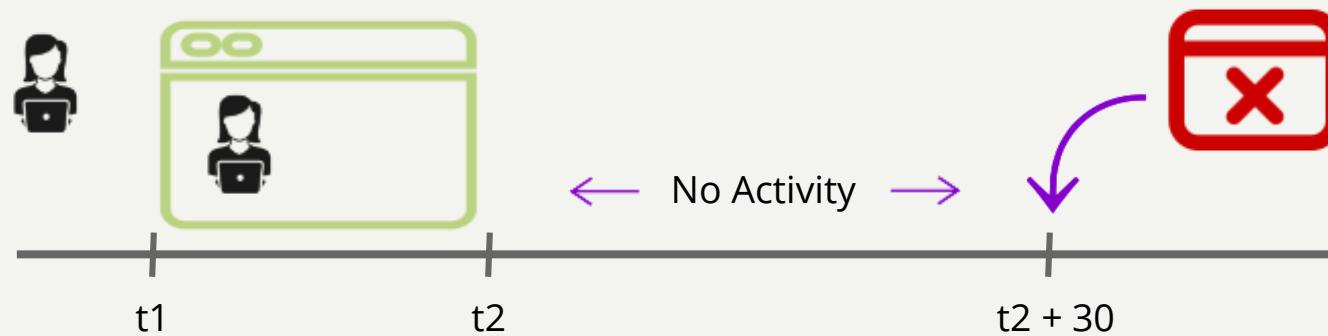
.window(EventTimeSessionWindows)

Agenda

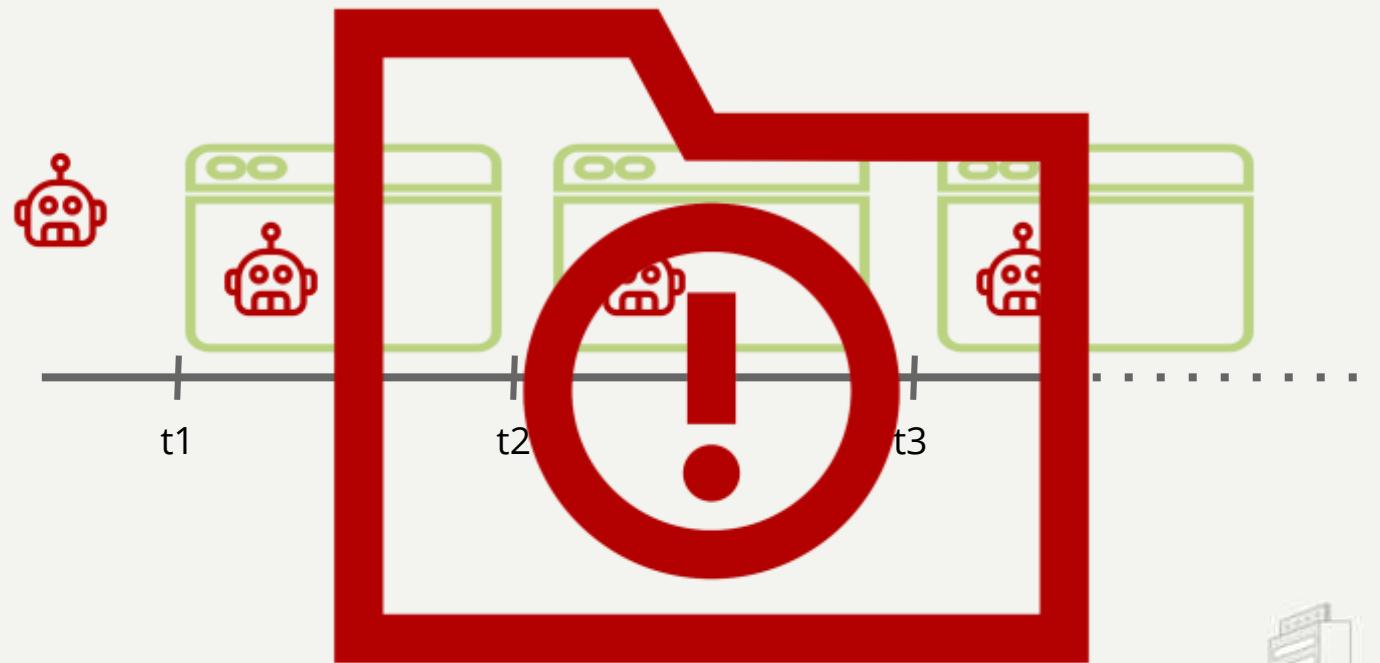
- Motivation
- Sessionizer Specs
- Sessionizer
- Issues
 - Bot traffic
 - Duplicate sessions
 - Stale topics
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State



Never ending bot sessions



Never ending bot sessions



Never ending bot sessions

```
class SessionLengthAndLateDataTrigger(..) extends Trigger {  
  
    def onElement(.., ctx: TriggerContext): TriggerResult = {  
        if(ctx.getCurrentWatermark > startTimestamp + maxSessionLength) {  
            // Perform cleanup actions  
            TriggerResult.FIRE_AND_PURGE  
        }  
    }  
}
```

Agenda

- Motivation
- Sessionizer Specs
- Sessionizer
- Issues
 - Bot traffic
 - **Duplicate sessions**
 - Stale topics
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State



Duplicate Sessions

Session ID	Number of Events
31cca9e6-4207-a323-36c121deaf73	45
95bff7c34-4de5-bc10-ef969a876cf7	27
31cca9e6-4207-a323-36c121deaf73	46
.....



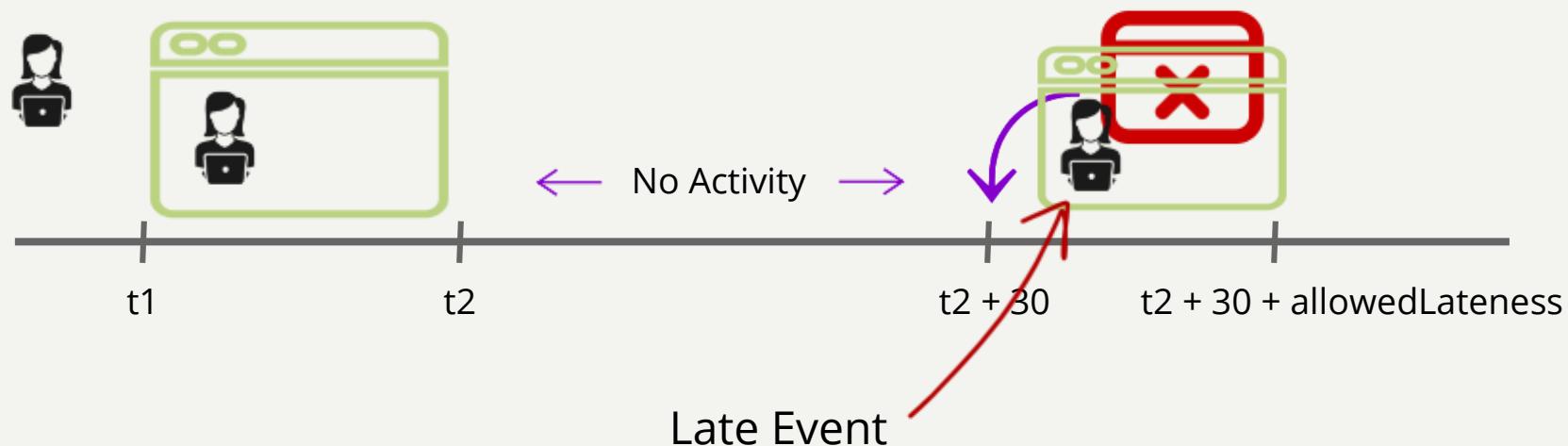
Duplicate Sessions



```
val sessionStream = messageStream
    .keyBy(message => { getUserId(message) })
    .window(EventTimeSessionWindows.withGap(sessionDuration))
    .allowedLateness(allowedLateness)
```



Duplicate Sessions



Agenda

- Motivation
- Sessionizer Specs
- Sessionizer
- Issues
 - Bot traffic
 - Duplicate sessions
 - **Stale topics**
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State

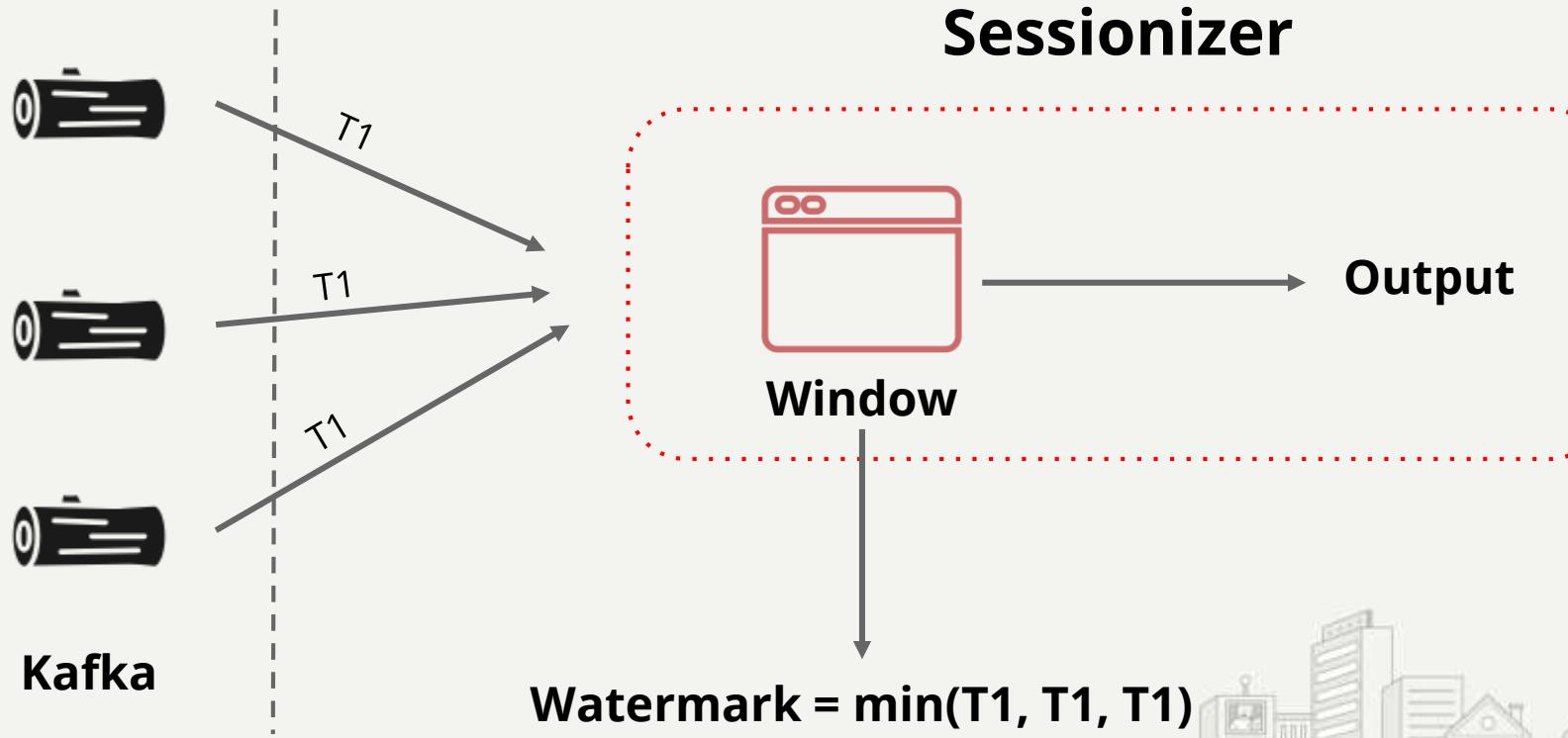


Stale Topics

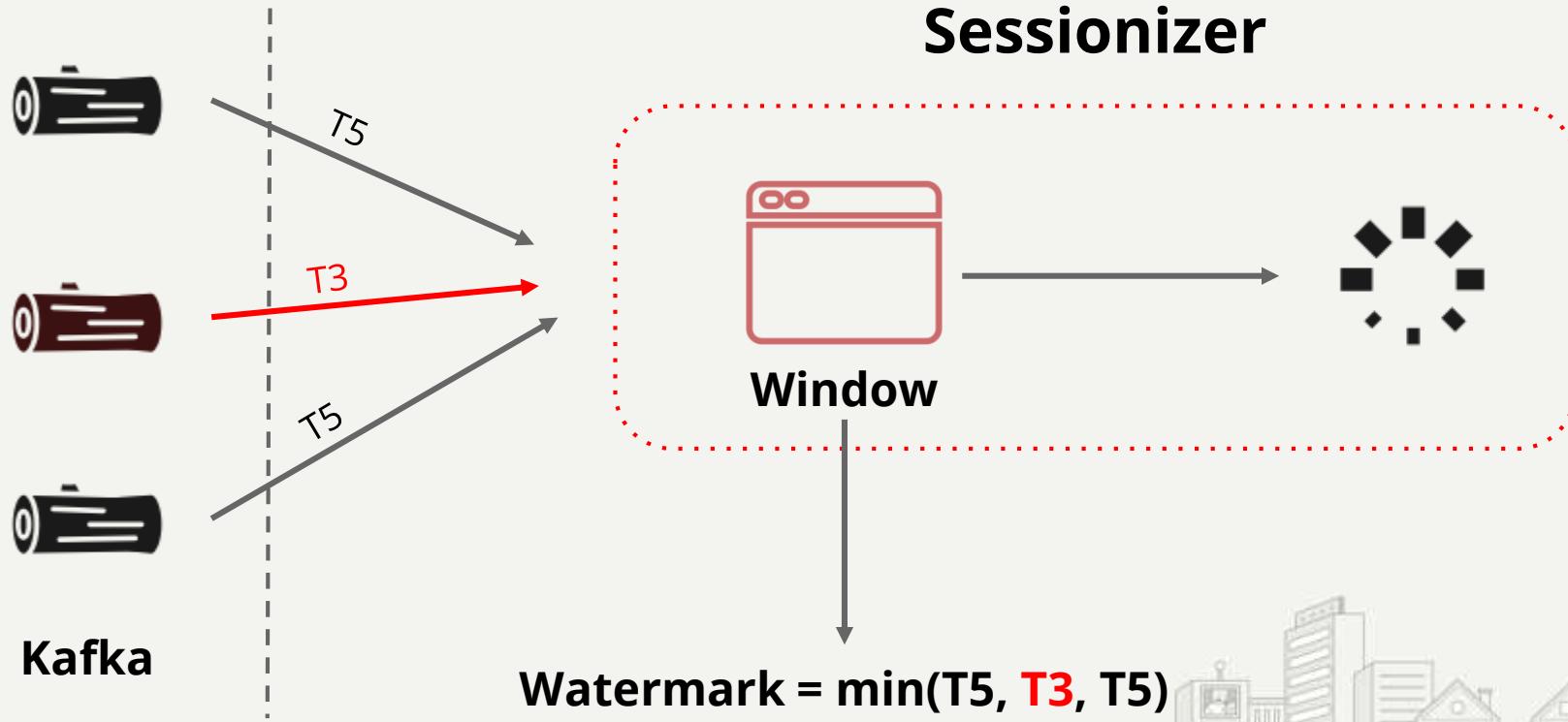
- Event time processing
- Event time watermark: to signal progress in event time.
- Watermarks are crucial when events can be **out-of-order**



Stale Topics



Stale Topics

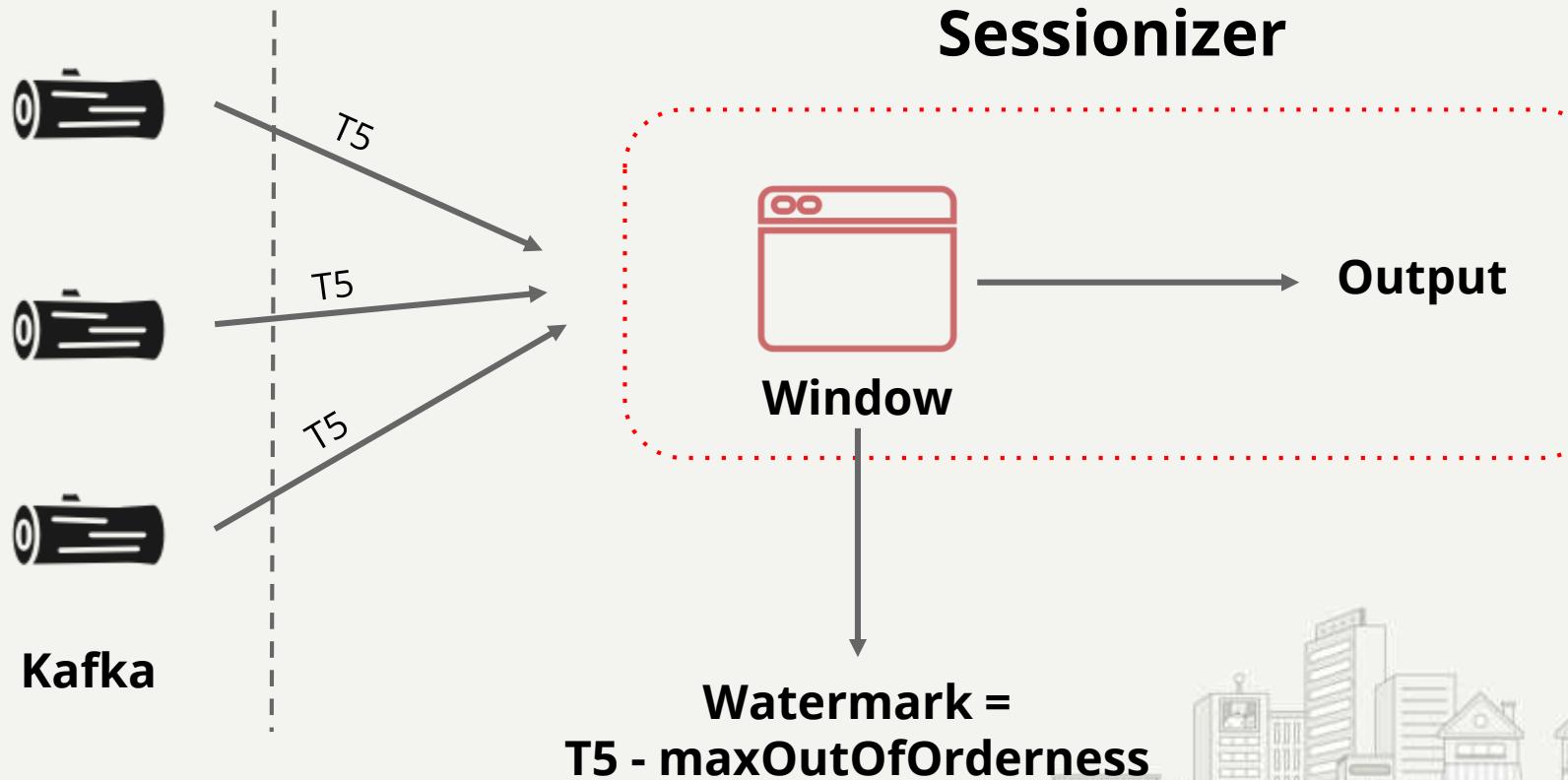


Stale Topics (Solution)

- Internal state stores time T since last watermark was seen.
- If $T > 10\text{mins}$, override and set watermark



Stale Topics (Solution)



Agenda

- Motivation
- Sessionizer Specs
- Sessionizer
- Issues
 - Bot traffic
 - Duplicate sessions
 - Stale topics
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State



S3 Throttling/ Checkpointing

- Due to large ingestion rate, we were checkpointing frequently.
- Caused us to be throttled by S3 , causing checkpoint failures



Amazon S3 Announces Increased Request Rate Performance

Posted On: Jul 17, 2018

Amazon S3 now provides increased performance to support at least 3,500 requests per second to add data and 5,500 requests per second to retrieve data, which can save significant processing time for no additional charge. Each S3 prefix can support these request rates, making it simple to increase performance significantly.

Fixed by [FLINK-9061](#)



Agenda

- Motivation
- Sessionizer Specs
- Sessionizer
- Issues
 - Bot traffic
 - Duplicate sessions
 - Stale topics
 - S3 Checkpointing / Throttling
 - Flink Internal Timer State



Flink Internal Timer State

- Slow data structure to store timer state
- Stores timers in-memory



Flink Internal Timer State

- Slow data structure to store timer state
 - HeapInternalTimerService $O(n)$ deletion operation
 - High CPU usage
 - 50 million session mark → super slow processing



Flink Internal Timer State

- Stores timers in-memory
 - HeapInternalTimerService only implementation
 - Session count ↑ results OOM
 - [FLINK-9485](#) rocksDB implementation



Flink Internal Timer State

- Slow data structure to store timer state
 - [FLINK-9423](#)
- Stores timers in-memory
 - [FLINK-9485](#)



Thank You!



pxu@yelp.com



@premsantosh



A large group of diverse people, mostly young adults, are gathered in what appears to be a modern office or event space. They are all smiling and raising their hands in the air, suggesting excitement, celebration, or participation in a group activity. The background shows office elements like cubicles and a staircase. The overall atmosphere is positive and energetic.

We're Hiring!

www.yelp.com/careers/



[fb.com/YelpEngineers](https://www.facebook.com/YelpEngineers)



[@YelpEngineering](https://twitter.com/YelpEngineering)



engineeringblog.yelp.com



github.com/yelp

The background of the slide features a photograph of a wolf in a snowy, mountainous environment. A large, semi-transparent blue diagonal shape covers the left side of the slide. Overlaid on the image is a fine, light blue grid pattern.

Hunting for Attack Chains in Event Streams

Ray Ruvinskiy and Jonathan Walsh

April 2, 2019

About the Presenters

Ray Ruvinskiy

- Lead of the Detection Automation Team. Our team's job is to help our Security Analysts avoid the noise and zero in on the actionable.

Jonathan Walsh

- Member of the Threat Operations and Analysis team. We utilize the tools provided by the Detection Automation Team to improve our threat detection.

About Arctic Wolf Networks

What is a SOC?

Security Operations Centres (SOCs)
deal with security issues on an
organizational level.



About Arctic Wolf Networks

SOC-as-a-Service

AWN CyberSOC manages the security detection for our customer, gathering data from deployed network sensors and logs from other security tools.

These logs are processed by our system and the output is provided to human experts to make decisions on.



Alert on actionable data

Rising above the noise

Balance between being informative
and being noisy

Focus on what the client needs to deal
with immediately





“

Crying wolf may have been the boy's undoing, but the true irony was that the wolves were always lurking nearby.

Wes Fesler

American football player

Lifecycle of an attack

Attributes shared by network breaches

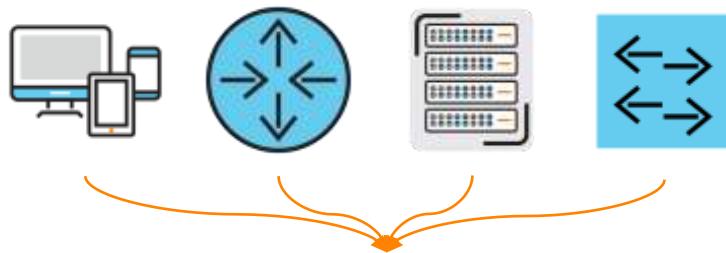


Source: <https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html>

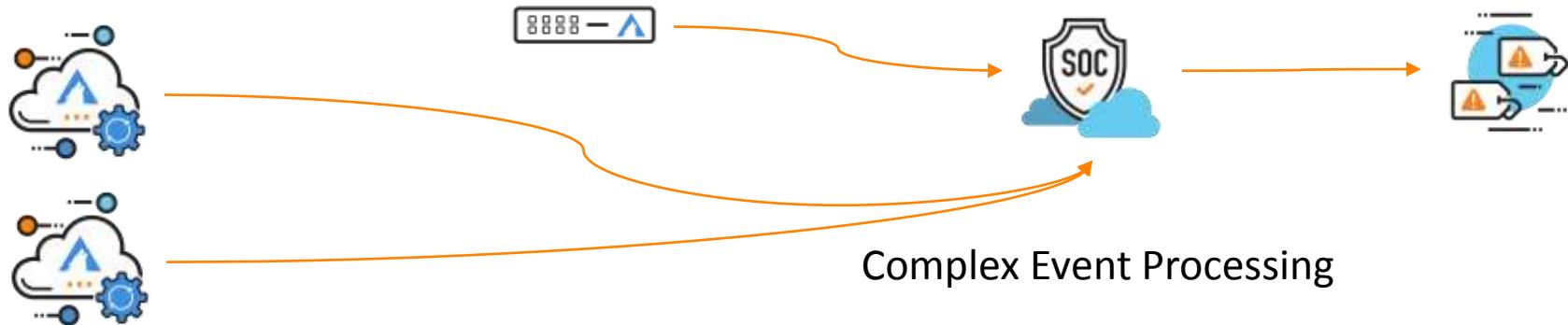
Applying Attack Chain to WannaCry Outbreaks

- Reconnaissance – social networking
- Weaponization – build ransomware binary
- Deliver/exploit – spearphishing
- Command and control – encrypted backchannel
- Actions on Objectives – encryption and lateral movement

Indicators at each of these steps can be noisy and prone to false positives. Correlation of events focuses us on what is relevant.



Over 13 billion events a day



Why Flink?

Built for streaming

Out-of-order events

Scalability



FlinkCEP

Our experience dates to 1.2.1

Difficulties with expressivity and scalability

- At the time, no counts

Esper

Mature and expressive streaming CEP framework
EPL – SQL-like Event Processing Language

EPL

PsExec services created on at least 10 devices in 5 minutes

```
select ... from PsExecEvent#time(5 minutes)  
#unique(target_ip)  
#length_batch(10);
```

Typed event, with
properties

Stream composition

Property in
PsExecEvent

EPL

Endpoint virus detection followed by network IDS exploit detection

```
select ... from pattern [  
    AntiVirusEvent -> IDSExploitEvent where timer:within(15 minutes)  
];
```



"followed by" operator

EPL

A minimum number of login failures with no successes interspersed, then a success

```
create window LoginFailureWindow#time(5 minutes) as LoginEvent;

on LoginEvent(not login_success) as login_failure
    merge LoginFailureWindow as failures
    where login_failure.id = failures.id
    when not matched then insert select *;

on LoginEvent(login_success) as login_success
    select and delete ... from LoginFailureWindow as failures
    having count(*) >= 10;
```

Complex Event Processing with Flink and Esper

Putting it all together

A single, continuously-running Flink job

Complex Event Processing with Flink and Esper

Putting it all together

Events are ingested

Source

Events are parsed and subjected to
sanity filtering

FlatMap

Timestamp extraction, watermark
assignment

TimestampExtractor

SplitStream used to create a
stream per rule

Split

Complex Event Processing with Flink and Esper

Putting it all together

Each of the per-rule streams
aggressively filters events of interest
and converts to Tuples

- Reduce per-event memory footprint as much as possible

keyBy rule-specific partition key to ensure unrelated events are processed in isolation

- e.g., for a login failure rule we key by customer ID and username

FlatMap

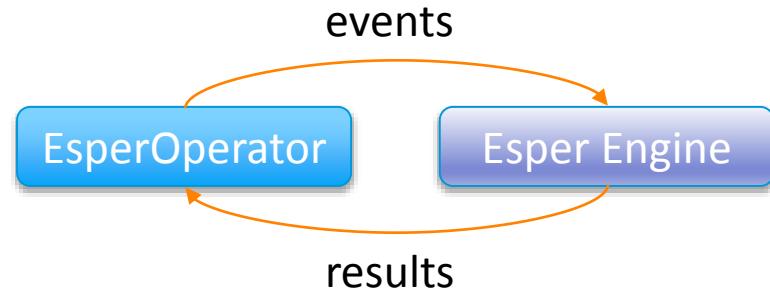
KeyBy

Complex Event Processing with Flink and Esper

Putting it all together

Esper operator

- Per-rule (potentially multi-statement) EPL that defines the logic
- Priority queue to order events by timestamp
 - Same as FlinkCEP
- Send tuples to Esper engine
- Register subscriber for results



Complex Event Processing with Flink and Esper

Putting it all together

Esper operator output formatted as AWN events with a special type

These AWN events then undergo another filter pass where some are whitelisted depending on specific rules defined by Security Analysts

Events that survive this pass then brought to Security Analyst attention for investigation

EsperOperator

Escalation Service



Flink/Esper at AWN

AWS EMR (Elastic MapReduce)

- Managed Hadoop environment
- Easiest way to get started two years ago (Flink 1.2)

Flink 1.4.2

m5.2xlarge core nodes

- 8 virtual cores, 32 GB RAM
- 8 task slots per node

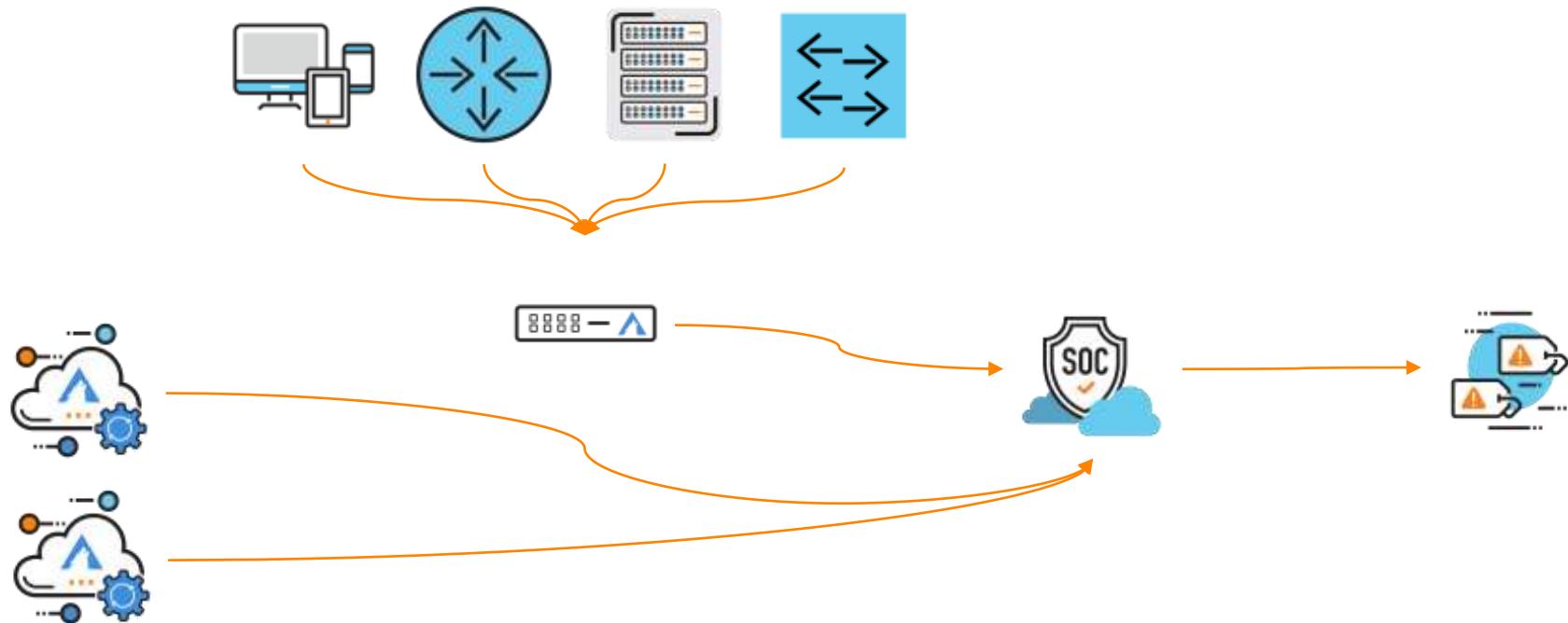
Flink/Esper at AWN

Over 30 rules

Over 13 billion events a day passing through Flink

- But only a small subset is evaluated by the rules

6 m5.2xlarge instances



Flow of Time Difficulties

Different data sources have different delays

- Events from cloud data sources (e.g., Office365) can in particular trickle in slowly

Time zone adjustments

- Normalize to UTC but *a lot* of data sources to keep track of

In summary

Together, Flink's simple deployment patterns and scalability and Esper's powerful CEP engine enable us to pick out attack chains in streams of billions of events.

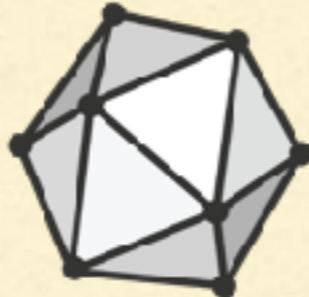
Thank You

For more information, visit us at: www.arcticwolf.com



Follow us on:





ONNX

ONNX MEETS FLINK

The long trudge towards integrating PyTorch, Chainer, CNTK, MXNet and other models in Flink streaming applications.

Overview

- The Problem/Motivation
 - ONNX
 - Overview
 - Limitations
 - End-to-end example with Java Embedded Python
-

Goals and challenges

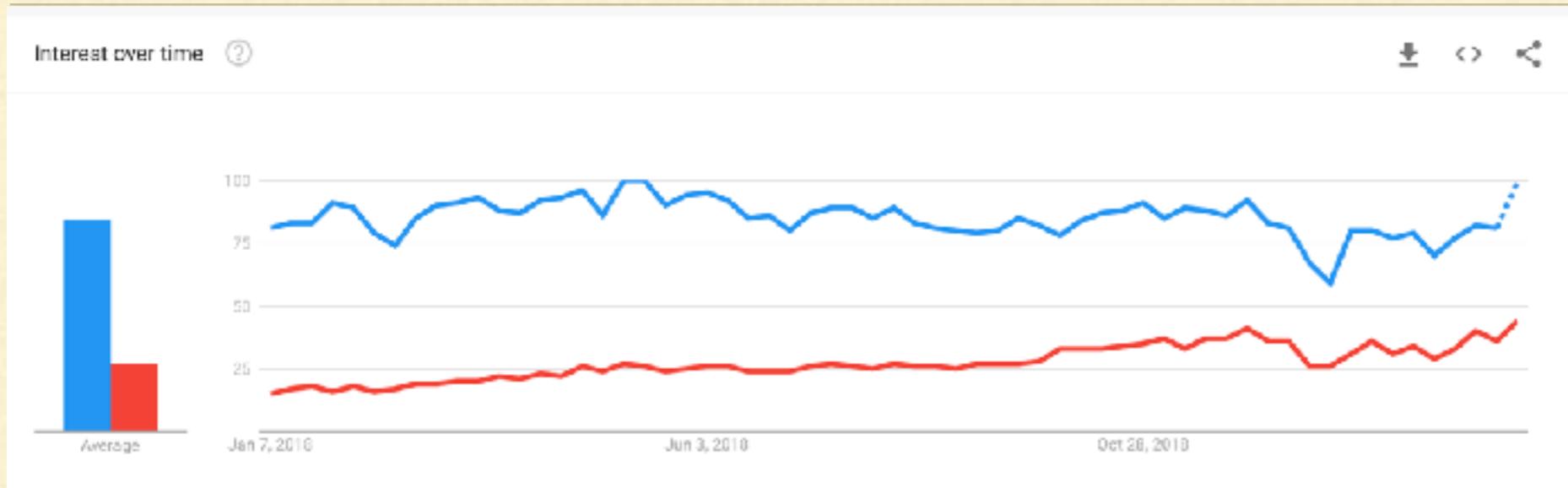
- Goals

- Remove barrier between A.I. “research” and “production.”
- Enable access to recent state of the art models from major conference and Python based frameworks
- Specifically, integrate deep learning models written in Python frameworks like PyTorch, CNTK, Chainer into Flink pipelines for realtime inference on streaming data.

- Challenge(s)

- Poor Python support in Flink and vice-versa poor ONNX support in Java
 - Converting a model to ONNX itself can be quite arduous
 - It can be challenging to rewrite pre-processing code in Java
-

The rise of PyTorch

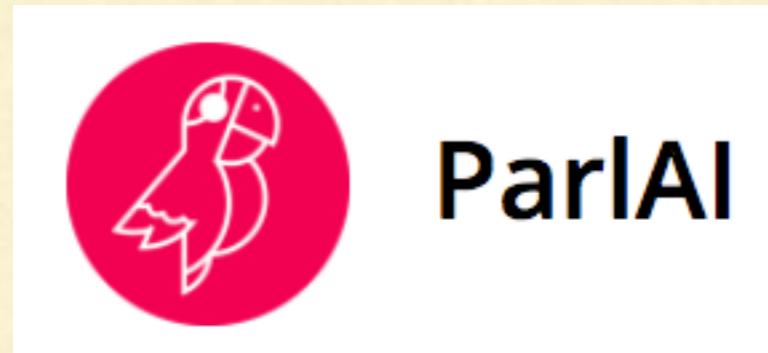


Tensorflow
PyTorch

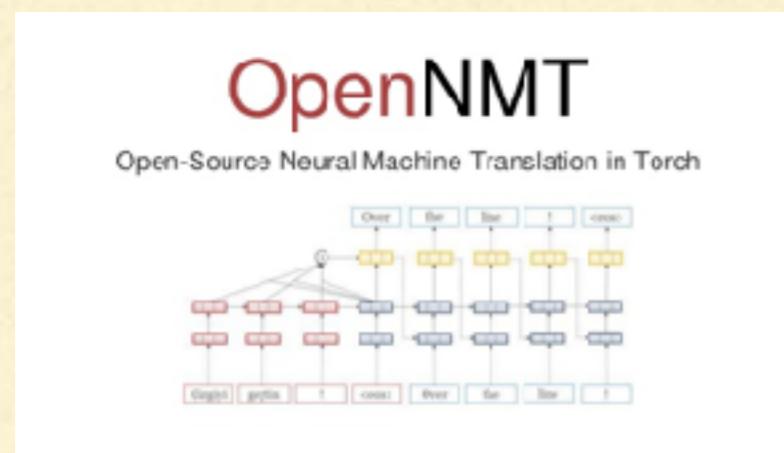
- International Conference Learning Representations (ICLR) statistics
 - 2018: 87 papers mentioned PyTorch (compared to 228 that mentioned Tensorflow)
 - 2019 252 papers mentioned PyTorch (compared to 266 that mentioned Tensorflow) Roughly a 190% increase!

PyTorch Powered frameworks

AllenNLP
flair



NLP



Other



torchvision

torchcv



What is ONNX?

Open neural network exchange format



WHY USE ONNX?

- Backends that (at least in theory) run in a large number of environments.
- Can export models from a variety of formats to a standard format
- Exported models generally smaller (in terms of space) than full models.

Overview of possible ways to integrate ONNX models into Flink

- Create a micro-service and use in conjunction with Flink AsynclO.
- Use Java embedded Python (JEP) and run using Caffe2 (or Tensorflow)
- Load model natively into Java/Scala and run with a JVM backend framework

ONNX frameworks overview

ONNX Scoreboard Measure supported operations

Latest Update	2018-08-30 19:13:19.606024
ONNX	1.2.2
PyTorch/Caffe2	0.5.0a0+e85f3fc
TensorFlow	1.10.1

Ops

Op	tensorflow	caffe2
ATen (Experimental)	Failed!	Failed!
Abs	Passed!	Passed!
Acos	Passed!	Passed!
Add	Passed!	Passed!
Affine (Experimental)	Failed!	Failed!
And	Passed!	Passed!
ArgMax	Passed!	Passed!
ArgMin	Passed!	Passed!
Asin	Passed!	Passed!
Atan	Passed!	Passed!
AveragePool	Passed!	Passed!
BatchNormalization	Passed!	Passed!
Cast	Passed!	Failed!
Cell	Passed!	Passed!
Clip	Passed!	Passed!

Models

Model	tensorflow	caffe2
resnet50	9/9 nodes covered: Passed!	9/9 nodes covered: Passed!
bvlc_alexnet	8/8 nodes covered: Passed!	8/8 nodes covered: Passed!
inception_v2	12/12 nodes covered: Passed!	12/12 nodes covered: Passed!
squeezezenet_010	7/7 nodes covered: Passed!	7/7 nodes covered: Passed!
inception_v1	10/10 nodes covered: Passed!	10/10 nodes covered: Passed!
vgg19	7/7 nodes covered: Passed!	7/7 nodes covered: Passed!
shufflenet	11/11 nodes covered: Passed!	11/11 nodes covered: Passed!
densenet121	10/10 nodes covered: Passed!	10/10 nodes covered: Passed!
Summary	8/8 model tests passed	8/8 model tests passed

OPTIONS

- ONNX options
 - Current backends
 -  Caffe2 (Python, C++)
 -  CNTK (C++, C#, Python, **Java experimental**)
 -  Tensorflow-ONNX (Python) [Not analogous to Tensorflow)
 -  VESPA (**Java**)
 - Menoh (C++, **Java**, C#, Ruby, NodeJS)

Menoh in Java

Only 19 of the 116 ops available (so pretty limited for now)

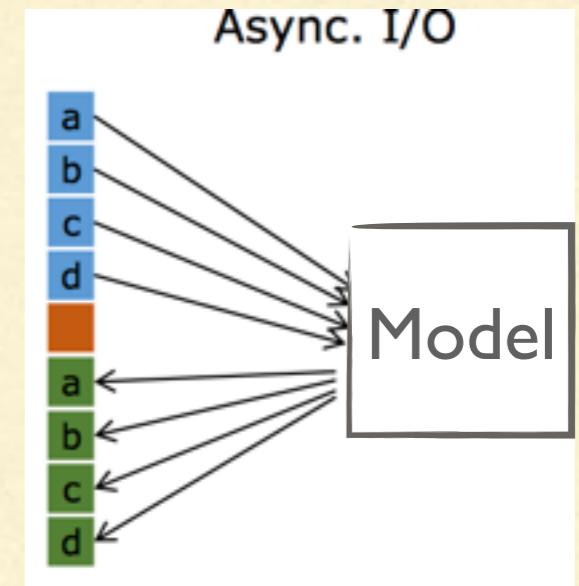
```
import jp.preferred.menoh.ModelRunner;
import jp.preferred.menoh.ModelRunnerBuilder;
try {
    ModelRunnerBuilder builder = ModelRunner
        // Load ONNX model data
        .fromOnnxFile("squeezeonnet.onnx")
        // Define input profile (name, dtype, dims) and output profile (name, dtype)
        // Menoh calculates dims of outputs automatically at build time
        .addInputProfile(conv11InName, DType.FLOAT, new int[] {batchSize, channelNum, height
width})
        .addOutputProfile(fc60utName, DType.FLOAT)
        .addOutputProfile(softmaxOutName, DType.FLOAT)
        // Configure backend
        .backendName("mkldnn")
        .backendConfig("");
    ModelRunner runner = builder.build()
} {
    // The builder can be deleted explicitly after building a model runner
    builder.close();
```

WHEN NOT TO USE ONNX?

- Export process in many cases is difficult and time consuming!
- Backends have limited support for various operations.
 - For instance, Yolo2 still cannot be run on even Caffe2 or Tensorflow backend due to lack of support of ImageScaler.
- Some models have to be re-trained before exporting

AsyncIO and Microservice

- Flink calls model “API” using AsyncIO similar to any other API connection
- Pros
 - Use Docker container to capture exact model dependencies (smaller container than with Flink+Model)
 - No (extensive) re-writing of code needed
- Cons
 - Have to handle scaling/maintaining a separate service



Java Embedded Python (JEP)

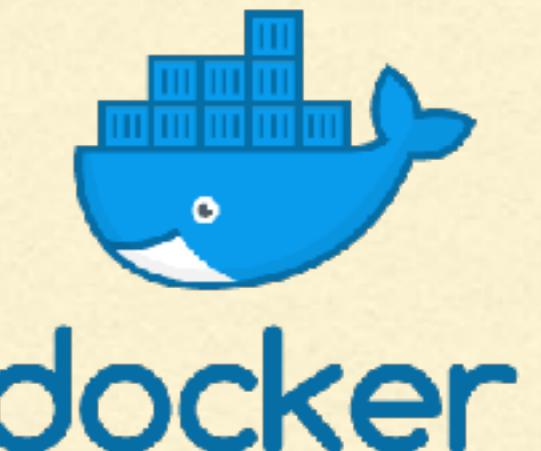
- Uses JNI and the Cython API to start up the Python interpreter inside the JVM
 - Faster than many alternatives
 - Can use pretty much any Python library including numpy, Tensorflow, PyTorch, Keras, etc
 - Automatically converts Java primitives, Strings, and `jep.NDArrays` sent into the Python interpreter into Python primitives, strings, and `numpy.ndarrays`
-

Using PyTorch directly with JEP

- Setup can be a bit painful
 - Have to get Python dependencies on all Flink nodes
 - Job needs path to Python
 - “Unsatisfied Link Error” is very common

Bootstrap script possible for EMR on AWS

Easiest way solution use: Kubernetes
AIStream JEP Flink Docker container





- NLP framework written in PyTorch with a state of the art named entity recognition (NER) model.

```
from flair.data import Sentence
from flair.models import SequenceTagger

# make a sentence
sentence = Sentence('I love Berlin .')

# load the NER tagger
tagger = SequenceTagger.load('ner')

# run NER over sentence
tagger.predict(sentence)
```

Task	Language	Dataset	Flair	Previous best
Named Entity Recognition	English	Conll-03	93.18 (F1)	92.22 (Peters et al., 2018)
Named Entity Recognition	English	Ontonotes	89.3 (F1)	86.28 (Chiu et al., 2016)
Emerging Entity Detection	English	WNUT-17	49.49 (F1)	45.55 (Aguilar et al., 2018)

Task	Language	Dataset	Flair	Previous best
Named Entity Recognition	English	Conll-03	93.18 (F1)	92.22 (Peters et al., 2018)
Named Entity Recognition	English	Ontonotes	89.3 (F1)	86.28 (Chiu et al., 2016)
Emerging Entity Detection	English	WNUT-17	49.49 (F1)	45.55 (Aguilar et al., 2018)

- Easy to train and combine with new methods
- Framework handles complex preprocessing and models PyTorch subclasses (therefore exporting to ONNX is not fun)

Named entity recognition on Flink data stream with Flair

```
import jep.Jep;
import jep.JepConfig;
import org.apache.flink.api.common.functions.RichMapFunction;
import jep.SharedInterpreter;
import org.apache.flink.configuration.Configuration;

public class FlairMap extends RichMapFunction<TweetData, String> {
    String tweetText = tweet.tweetText;
    private SharedInterpreter j;
    tweetText = tweetText.replaceAll("[^A-Za-z0-9]", " ");
    try {
        @Override
        public void open(Configuration c) {
            j.eval("s=Sentence(text)");
            try {
                eval("model.predict(s)");
                Object result = j.getValue("s.get_spans('ner')");
                return result.toString();
            } catch (Exception e) {
                j.eval("from flair.models import SequenceTagger");
            }
            catch (jep.JepException e) {
                e.printStackTrace();
            }
            throw e;
        }
    } catch (jep.JepException e) {
        e.printStackTrace();
    }
}
```

Sentiment Analysis with Flair

```
from flair.models import TextClassifier
from flair.data import Sentence
classifier = TextClassifier.load('en-sentiment')
sentence = Sentence('Twitter is a really good company')
# print(sentence)
# print(tweetText)
# print(tweetText.replaceAll("[^A-Za-z0-9]", " "))
print("Sentence sentiment is: " + sentence.labels)

try {
    j.set("text", tweetText);
    j.eval("s=Sentence(text)");
    j.eval("model.predict(s)");
    Object result = j.getValue("s.labels");
    return result.toString();
}
catch(jep.JepException e){
    e.printStackTrace();
    throw e;
}
```

Putting it all together

https://github.com/isaacmg/dl_java_stream

- Consume data from Twitter Source using Flink Twitter Connector
- Filter out non-English Tweets
 - Alternatively could load multilingual NER model(s)
- Named Entity Recognition on Tweets (remove non-entities)
- Sentiment Analysis on Tweet (entity, label, sentiment)
- Convert to Table. Run query

```
SELECT entity, sentiment, count(entity)
FROM Tweets
GROUP BY entity, sentiment
```

Conclusions

- Currently easiest to either use JEP or a micro-service + AsyncIO
 - Saves time converting model to ONNX
 - No need to re-write code
 - Promising frameworks in the works like Menoh, VESPA, DI4J etc should eventually support ONNX natively but aren't mature enough yet.
-



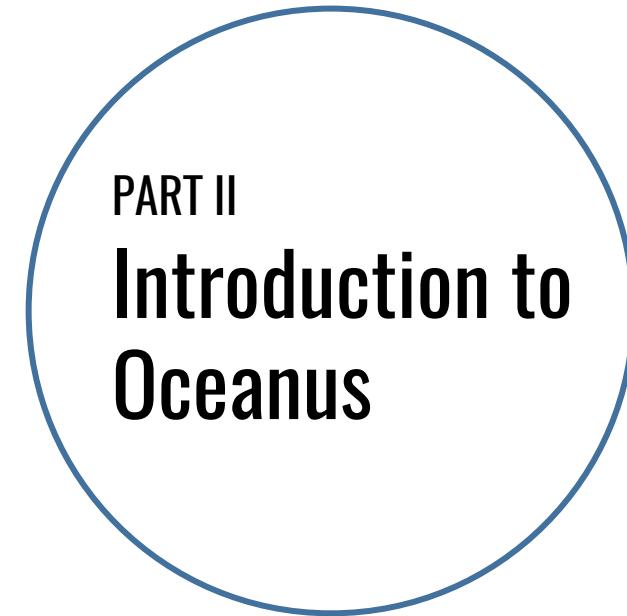
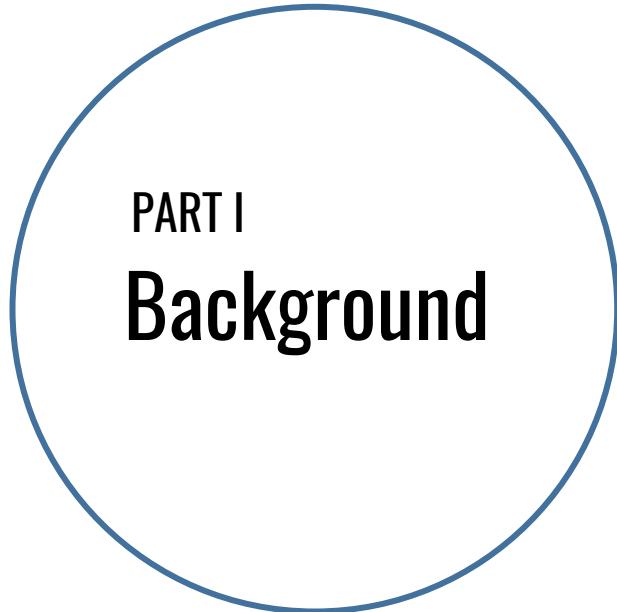
Tencent 腾讯

Developing and Operating Real-Time Applications at Tencent

Xiaogang SHI
robbieshi@tencent.com

Outline

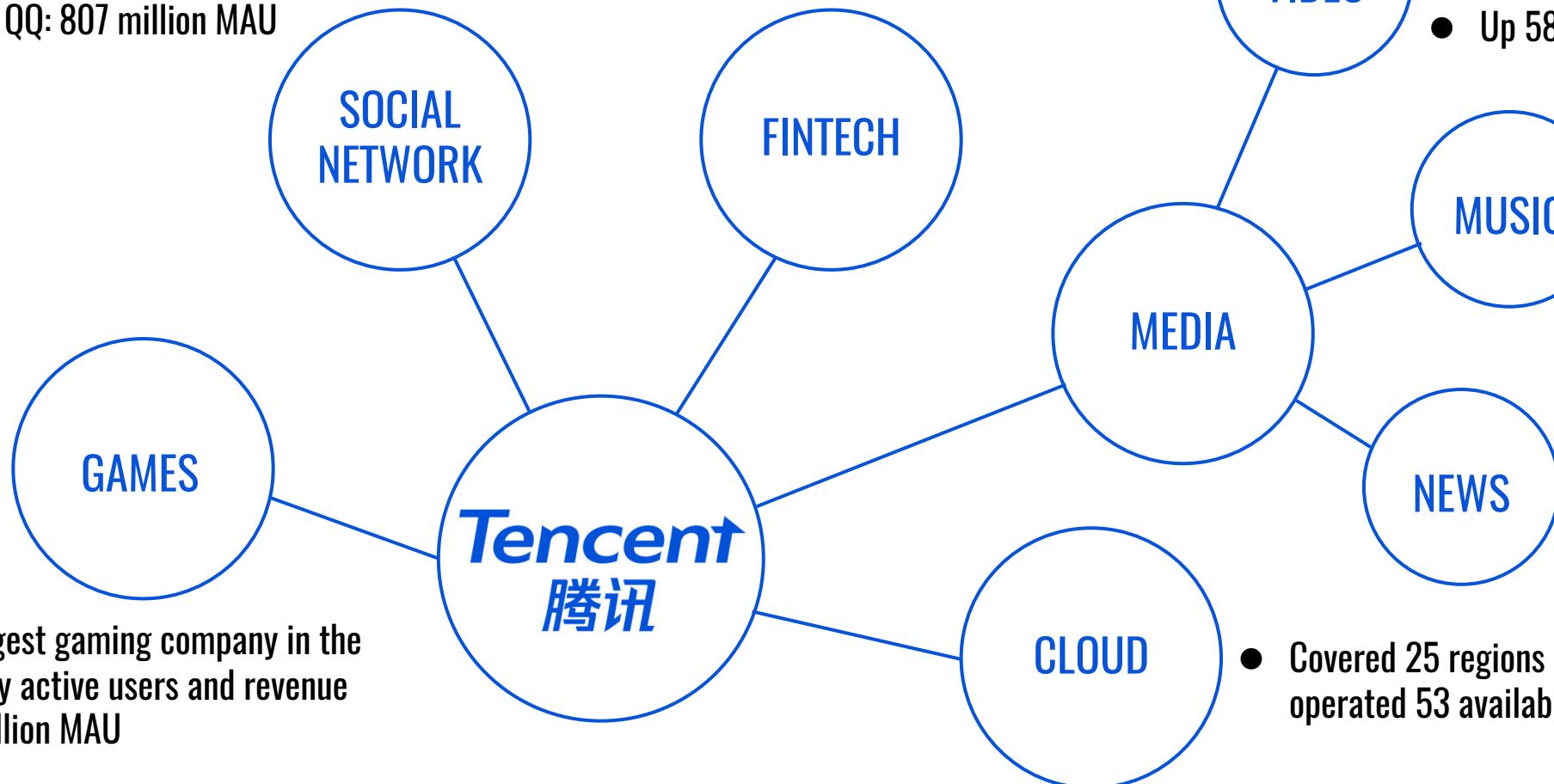
1



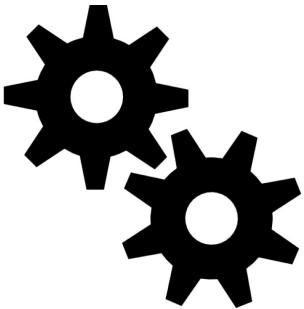
About Tencent



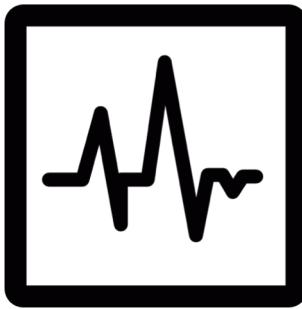
- The largest social communities in China in terms of MAU
- Weixin & Wechat: 1 billion MAU
- QQ: 807 million MAU



Real-Time Applications at Tencent



ETL



Monitoring



Real-Time BI



Online Learning

210 Million

Maximum number
of messages received
per second

17 Trillion

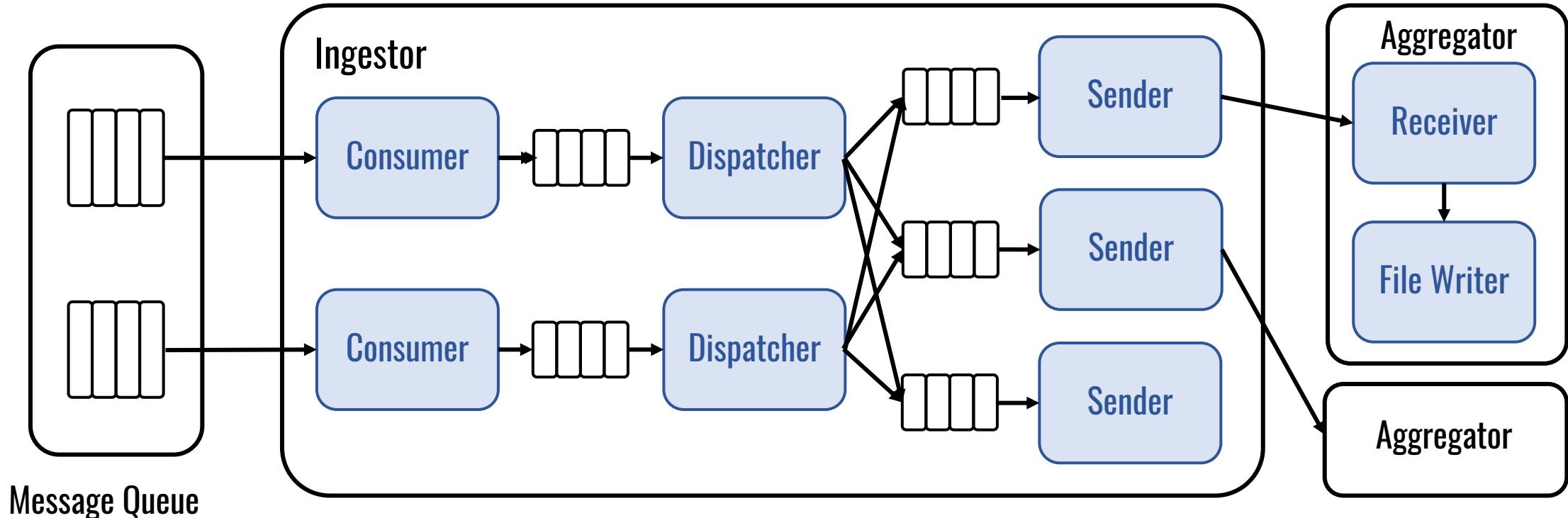
Number of messages
received per day

3 PB

Amount of data
received per day

Why Flink?

An ETL pipeline as an example

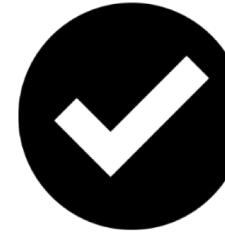


It's very difficult to improve performance while ensuring correctness.

Flink's Strength



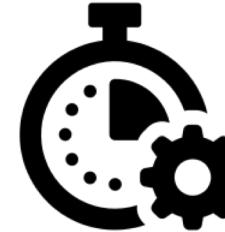
Efficient support for states
Automatic distribution while
rescaling



AT-LEAST and EXACTLY-
ONCE guarantees while
tolerating failures



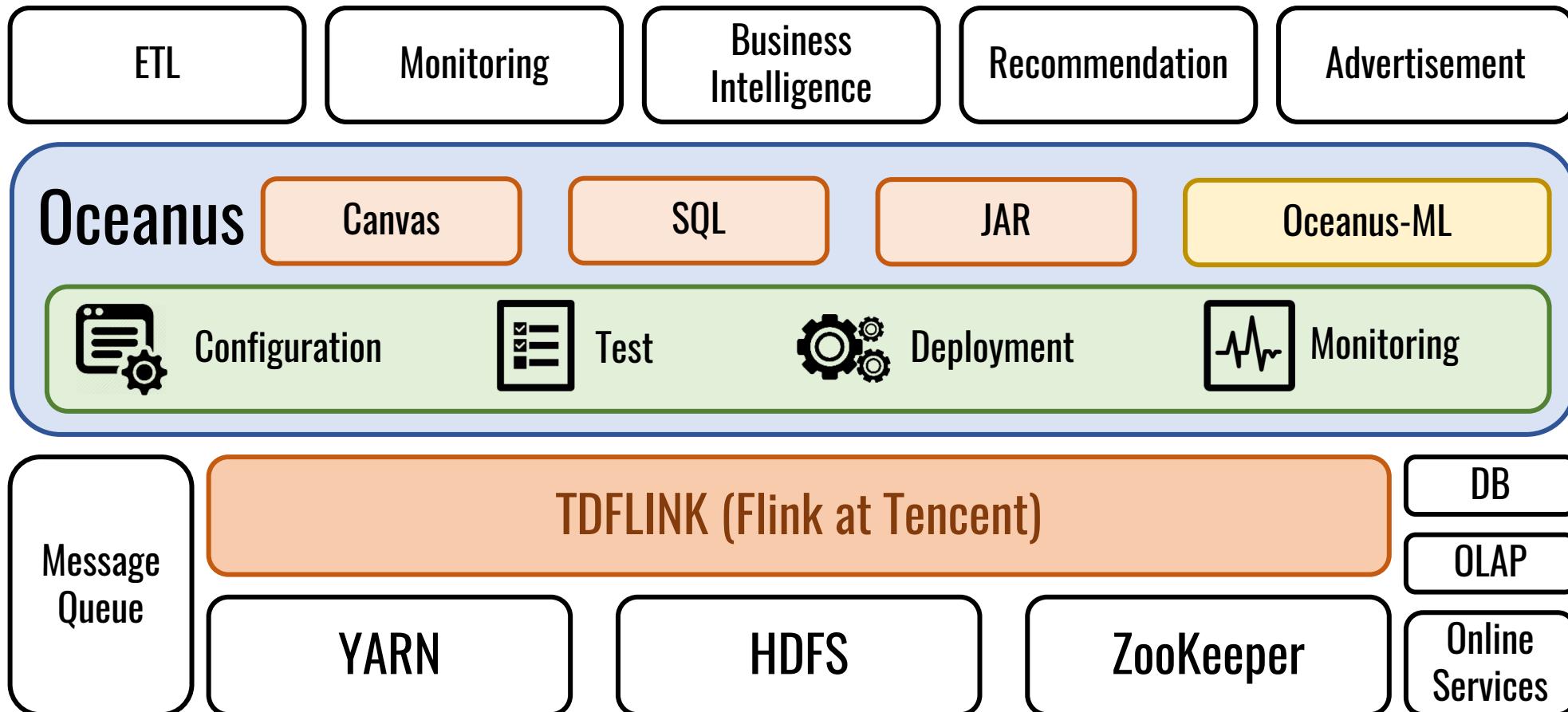
Flexible and powerful
programming interfaces for
stream processing



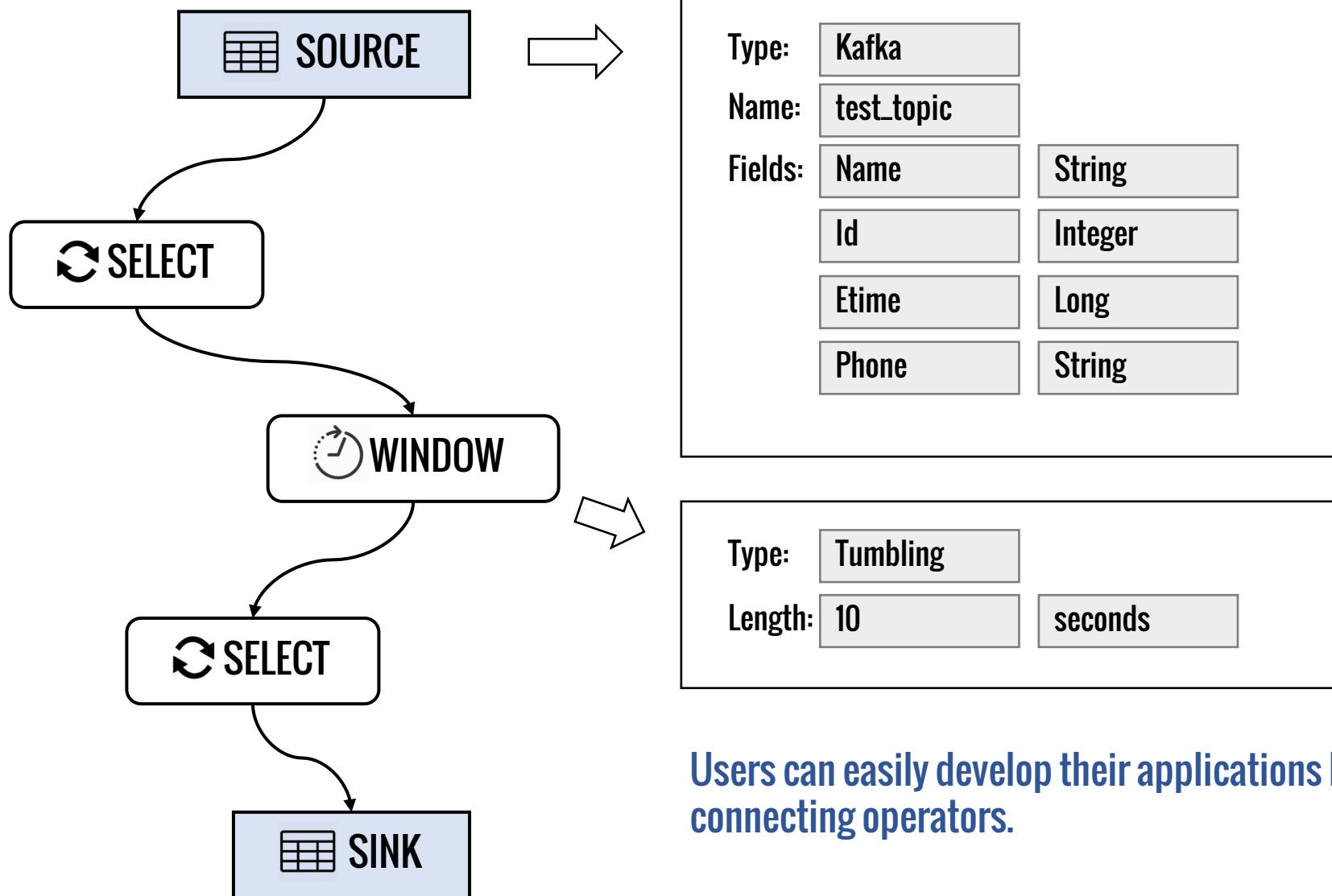
Low latency and high
throughput

Oceanus Overview

A unified platform to develop and operate real-time applications

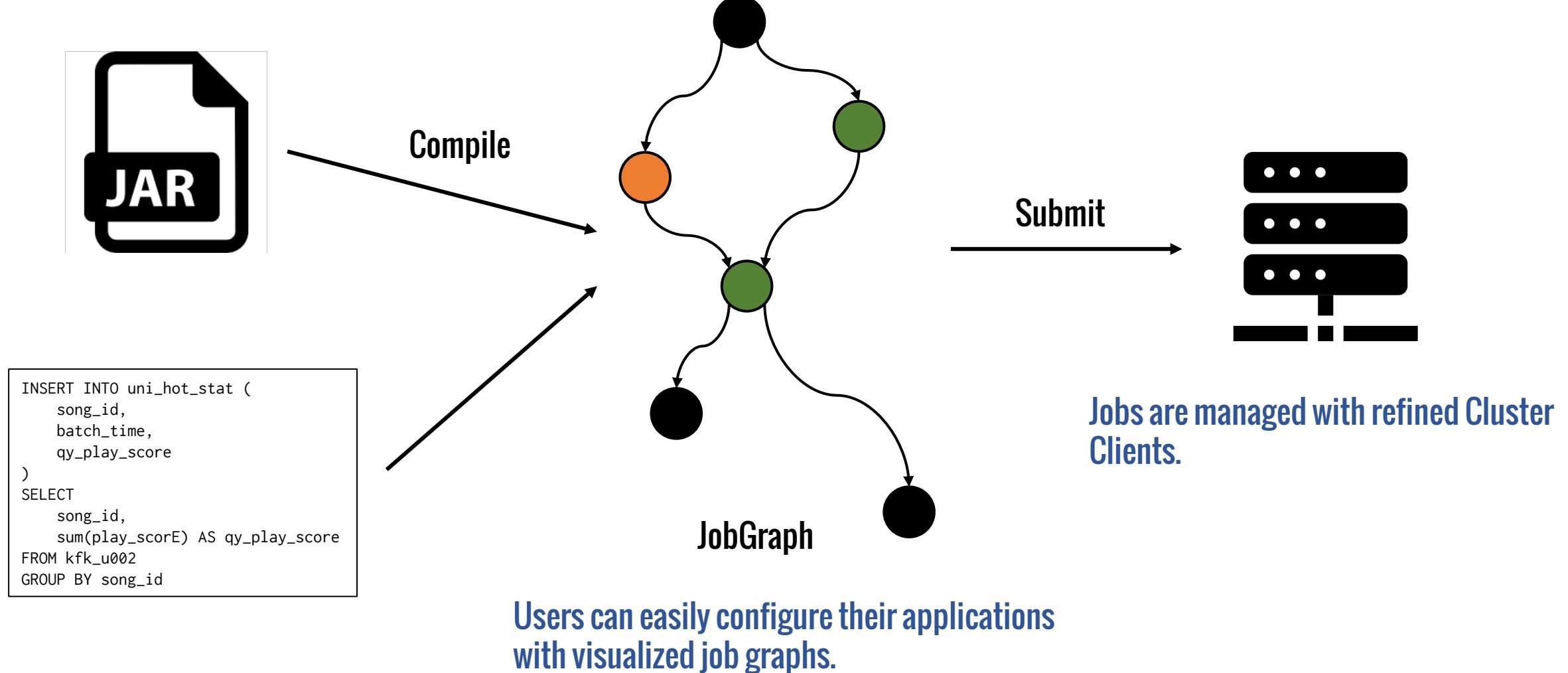


Developing with Oceanus



Users can easily develop their applications by dragging and connecting operators.

Developing with Oceanus



Operating with Oceanus

Improve operating efficiency with rich metrics.

Job Details								
	Vertices	Metrics (Records)	Metrics (Bytes)					
ID	Name	Watermark	InQueue Usage	OutQueue Usage	Input Records	Output Records	Input Rate	Output Rate
		max ▾	max ▾	max ▾	sum ▾	sum ▾	sum ▾	sum ▾
0	Source: Collection Source bc764cd8ddf7a0cff126f51c16239658	N/A	0.00%	100.00%	0	3,425,953,507,895	0	3,527,340
1	Map -> Sink: Unnamed 20ba6b65f97481d5570070de90e4e791	N/A	0.00%	0.00%	3,425,953,469,803	0	3,527,350	0

1. An operator is bottleneck if its in-queue is full while its out-queue is not full.
2. The ratio between the throughput of different operators remain roughly the same when the parallelism changes. We can utilize this property to configure the parallelism (Don't work well with window operators).
3. There may exist data skew when the difference between the maximum and the minimum throughput is very large.

Operating with Oceanus

10

Much information can be obtained from thread stacks.

Checkpoint timeout: lock unreleased by blocked user functions, slow hdfs writes, blocked user checkpoint functions
Performance issues

Threads					
ID	Name	CPU ↓	State	Stack	
70	Map -> Sink: Unnamed (1/1)	99.77%	RUNNABLE	<pre>org.apache.flink.streaming.api.operators.AbstractStreamOperator.setKeyContextElement(AbstractStreamOperator.java:625 org.apache.flink.streaming.api.operators.AbstractStreamOperator.setKeyContextElement(AbstractStreamOperator.java:61 org.apache.flink.streaming.runtime.io.StreamInputProcessor.processInput(StreamInputProcessor.java:201) org.apache.flink.streaming.runtime.tasks.OneInputStreamTask.run(OneInputStreamTask.java:107) org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:301) org.apache.flink.runtime.taskmanager.Task.run(Task.java:721) java.lang.Thread.run(Thread.java:748)</pre>	
68	Source: Collection Source (1/1)	80.32%	RUNNABLE	<pre>org.apache.flink.runtime.state.KeyGroupRangeAssignment.assignToKeyGroup(KeyGroupRangeAssignment.java:59) org.apache.flink.runtime.state.KeyGroupRangeAssignment.assignKeyToParallelOperator(KeyGroupRangeAssignment.java:48) org.apache.flink.streaming.runtime.partitioner.KeyGroupStreamPartitioner.selectChannels(KeyGroupStreamPartitioner.java:101) org.apache.flink.streaming.runtime.partitioner.KeyGroupStreamPartitioner.selectChannels(KeyGroupStreamPartitioner.java:97) org.apache.flink.runtime.io.network.api.writer.RecordWriter.emit(RecordWriter.java:101) org.apache.flink.streaming.runtime.io.StreamRecordWriter.emit(StreamRecordWriter.java:81) org.apache.flink.streaming.runtime.io.RecordWriterOutput.pushToRecordWriter(RecordWriterOutput.java:107) org.apache.flink.streaming.runtime.io.RecordWriterOutput.collect(RecordWriterOutput.java:89) org.apache.flink.streaming.runtime.io.RecordWriterOutput.collect(RecordWriterOutput.java:45) org.apache.flink.streaming.api.operators.AbstractStreamOperators\$CountingOutput.collect(AbstractStreamOperator.java:7 org.apache.flink.streaming.api.operators.AbstractStreamOperator\$CountingOutput.collect(AbstractStreamOperator.java:7 org.apache.flink.streaming.api.operators.StreamSourceContexts\$NonTimestampContext.collect(StreamSourceContexts.java:43) org.apache.flink.streaming.api.functions.source.FromIteratorFunction.run(FromIteratorFunction.java:43) org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:94) org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:58) org.apache.flink.streaming.runtime.tasks.SourceStreamTask.run(SourceStreamTask.java:99) org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:301) org.apache.flink.runtime.taskmanager.Task.run(Task.java:721) java.lang.Thread.run(Thread.java:748)</pre>	
62	SystemResourcesCounter probing thread	0.19%	TIMED_WAITING	<pre>java.lang.Thread.sleep(Native Method) org.apache.flink.runtime.metrics.util.SystemResourcesCounter.run(SystemResourcesCounter.java:128)</pre>	

Improvement to Flink

Job Management

- Avoid split-brain with Zookeeper transactions
- Non-disruptive recovery of job masters
- Fine-grained recovery of tasks with cached result partitions

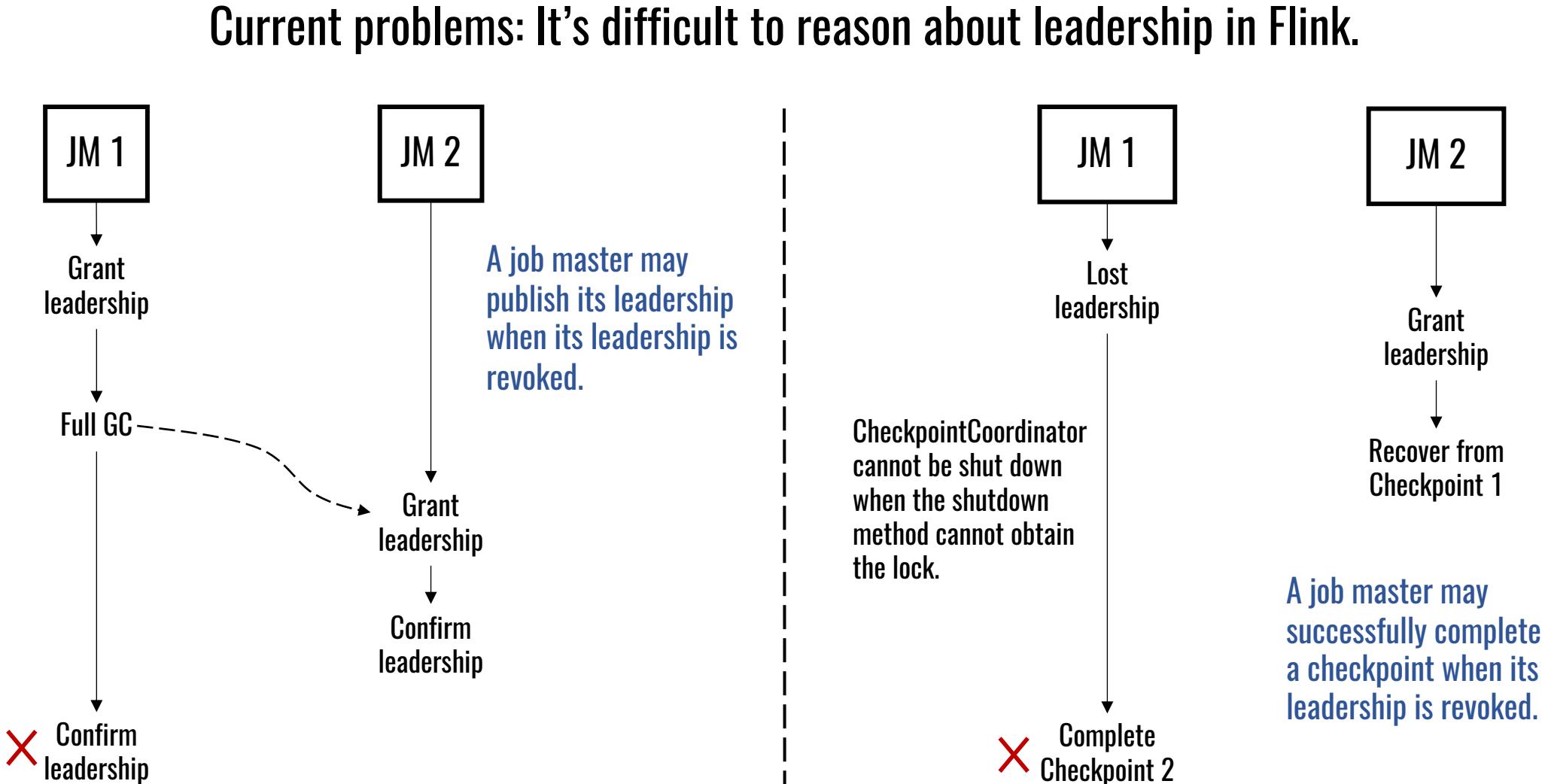
Resource Management

- Fine-grained resource allocation
- Improve scheduling efficiency

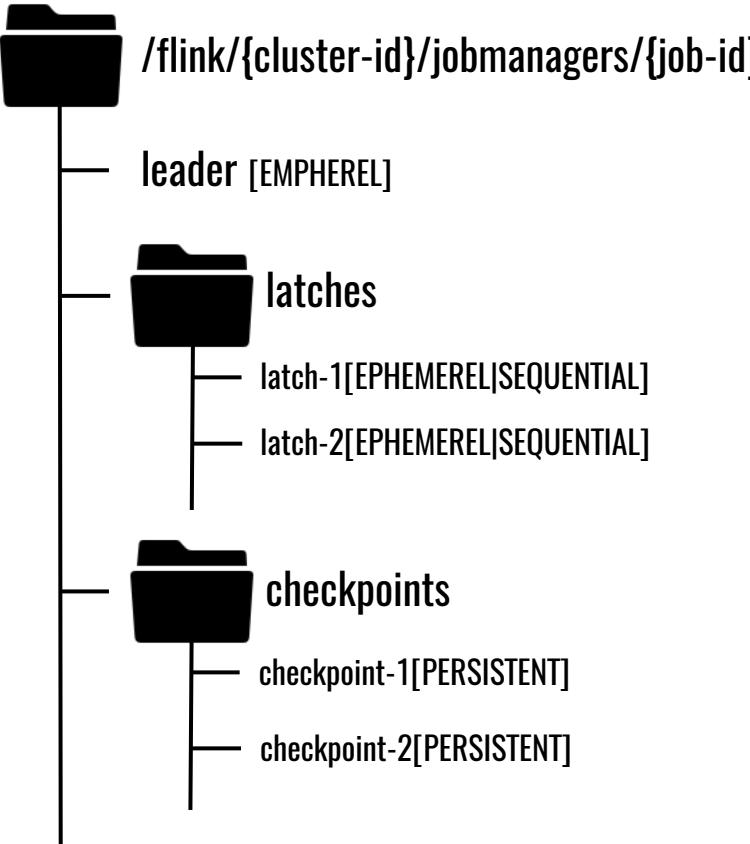
Performance & Usability

- Local keyed streams
- Incremental windows
- UDX
- DimJoin
- Top N

Refine leader coordination



Refine leader coordination

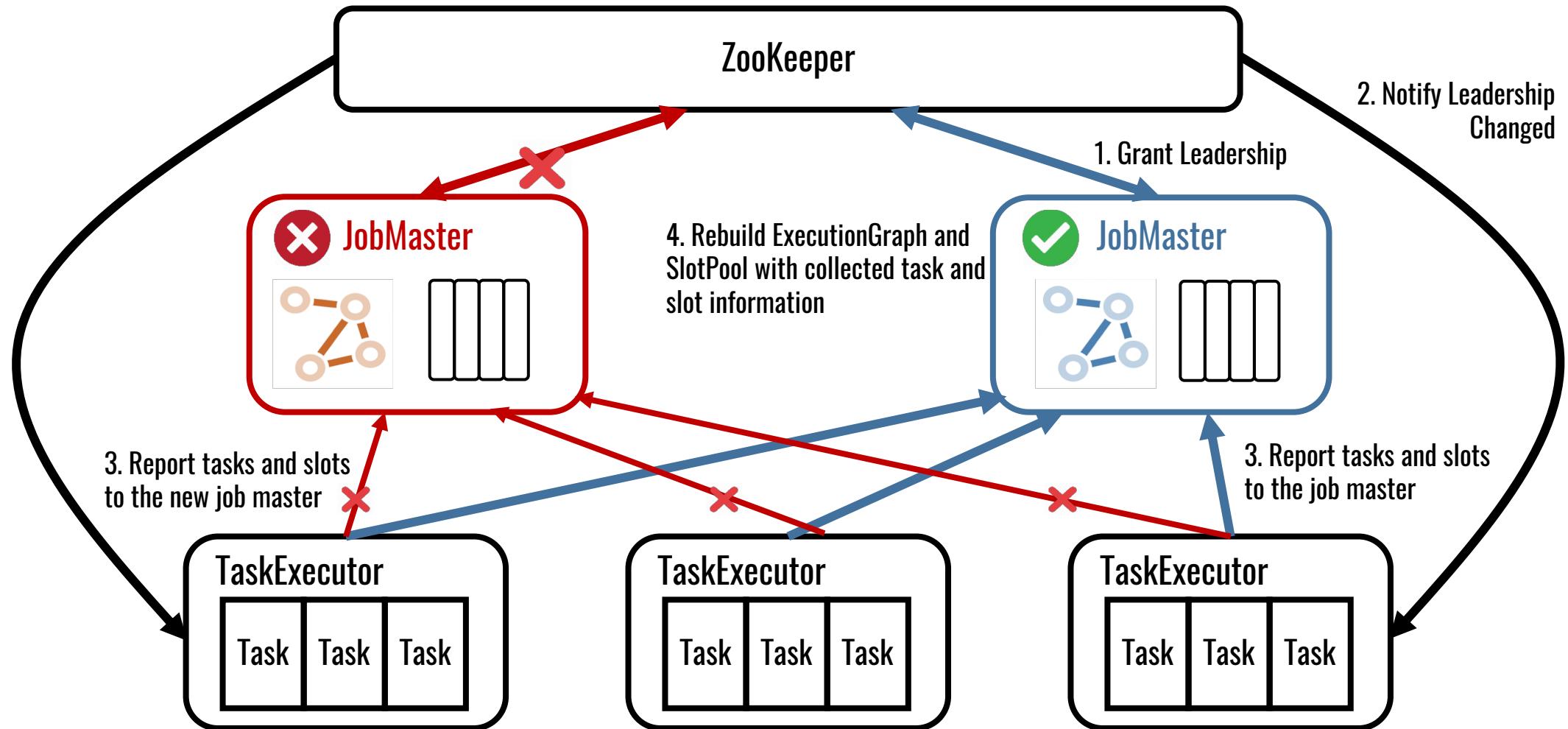


- Each leader contender creates a EPHEMERAL and SEQUENTIAL latch.
- The contender whose latch's sequential number is smallest is elected as the leader.
- A leader's leadership is granted as long as its latch exists.
- Each contender can only access states when it has granted leadership and its latch still exists.

```
zkClient.inTransaction()
    .check().forPath(myLatch).and()
    .setData().forPath(dataPath).and()
    .commit()
```

Non-disruptive recovery of job masters

Avoid restarting tasks when the job master fails.

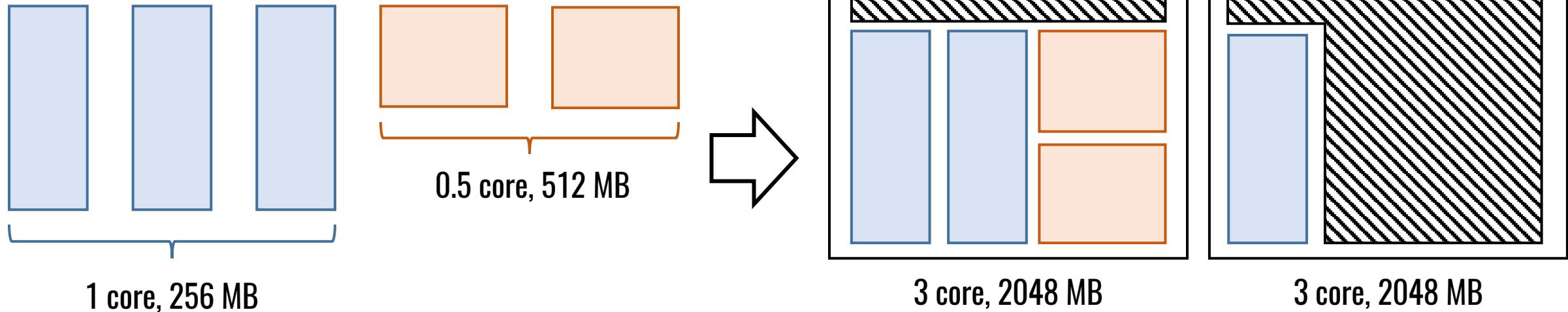


Fine-grained resource allocation



Current Problems:

- The resource specification for operators does not take effect in resource allocation.
- Slots are allocated according to the number of available slots in task managers, instead of the amount of available resources.
- Yarn containers may be killed when the used resources exceed the allocated ones.

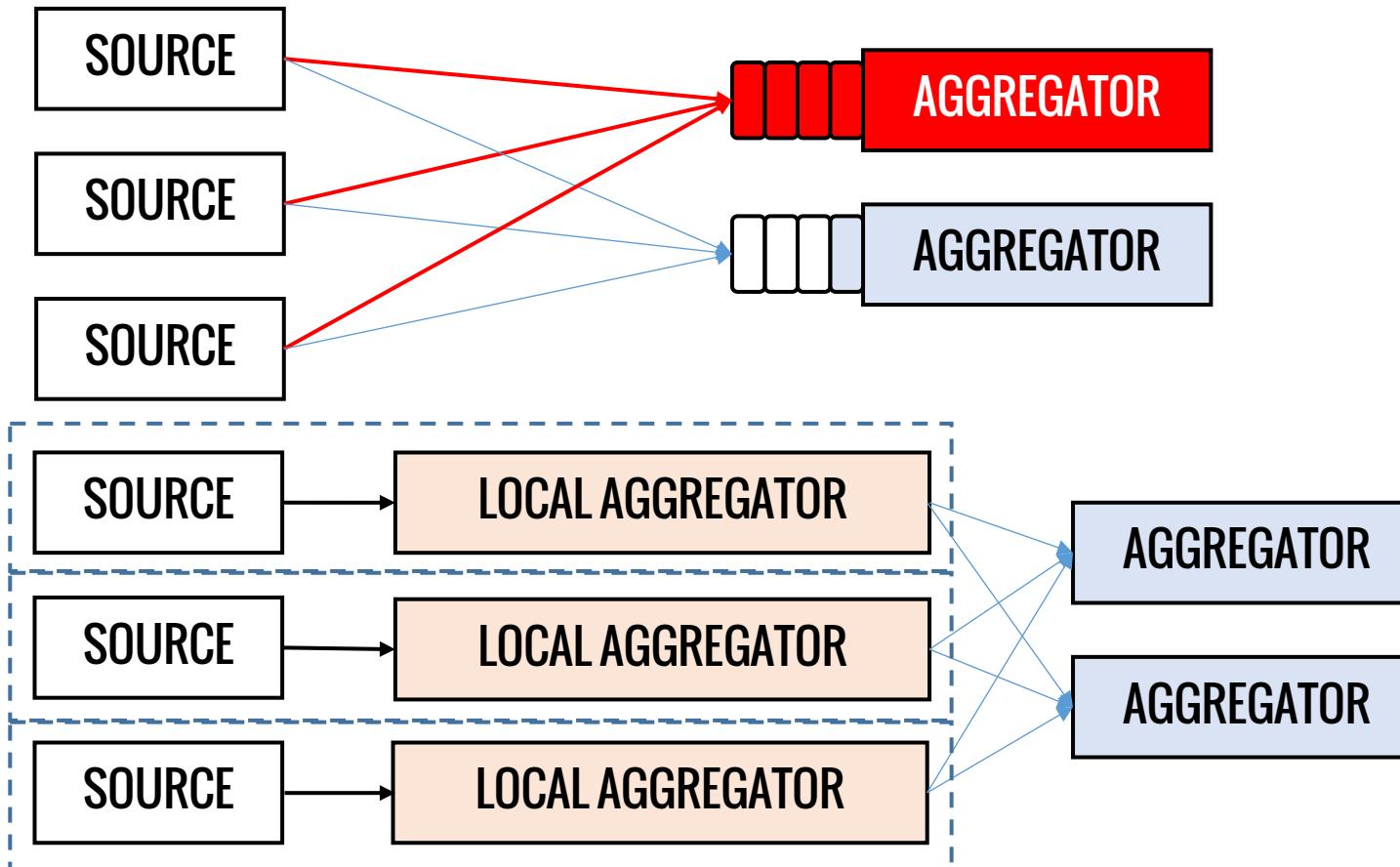


- Users can specify the resources needed by operators.
- A slot's resources are calculated by accumulating the resources of the operators in the slot.

- Slot instances are created and destroyed dynamically.
- A task manager creates a slot instance if its available resources are sufficient for the slot.
- A task manager destroys the slot instance if the tasks in the slot finish.

Local keyed streams

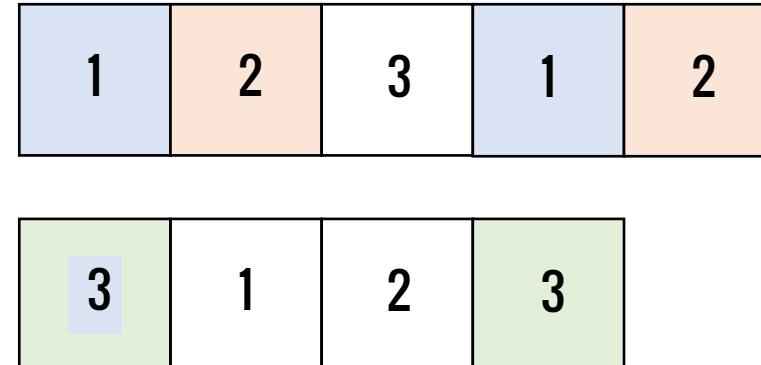
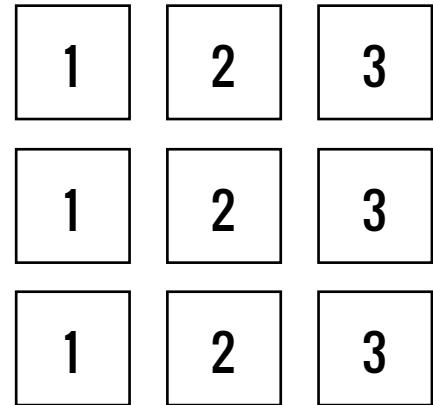
Current Problems: Performance is significantly degraded by data skew.



`words.keyBy().count()`

`words`
`.localKeyBy().window().count()`
`.keyBy().sum()`

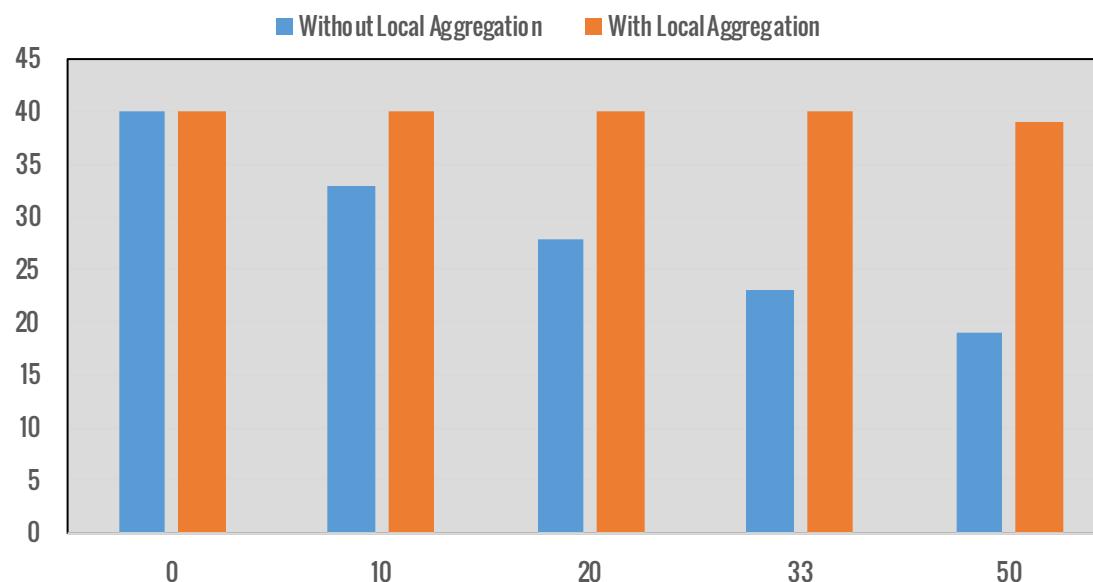
Local keyed streams

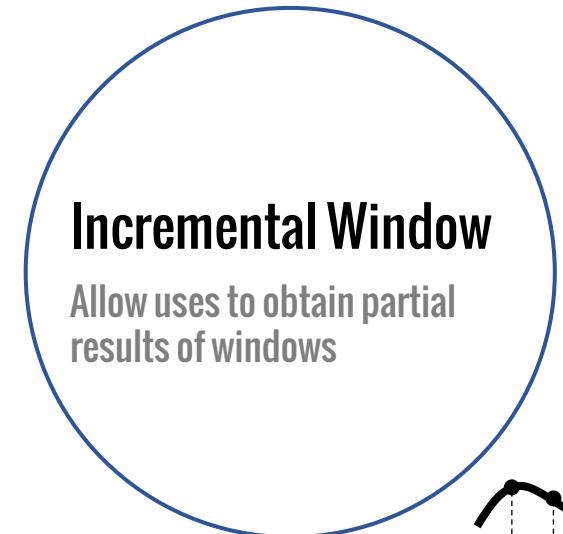
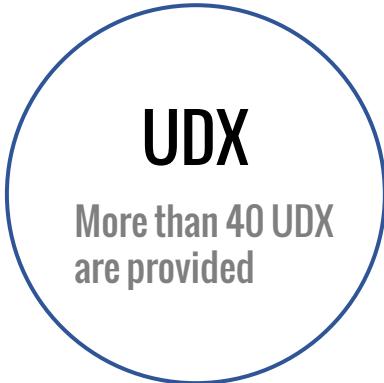


Each task has a complete key group range.

Groups are distributed to tasks according the number.

Groups with the same id are merged at restoring.





Future Work



Improve scheduling efficiency

Unified checkpoint mechanism for both streaming and batch jobs

Incorporating partitioning and timing into optimizer

SuperSQL: efficient data analytics across data sources (Hive, HBase, PostgreSQL, etc) and data centers

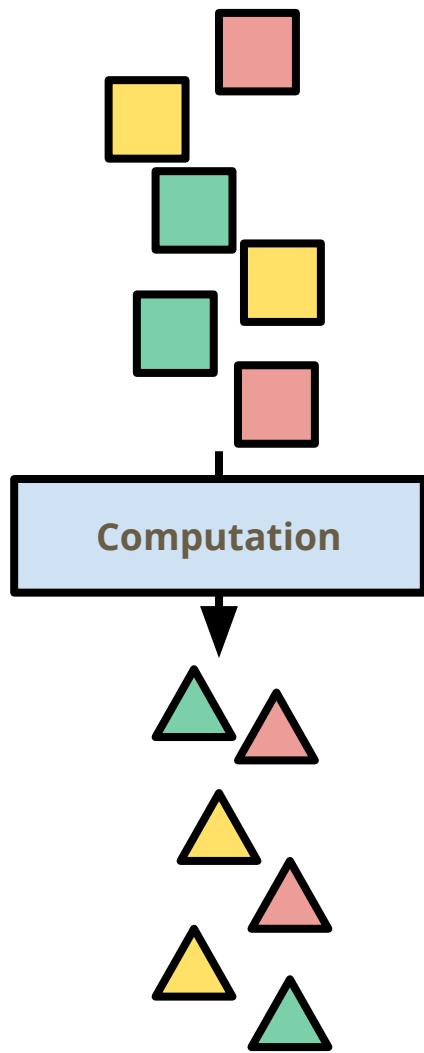
Thank You

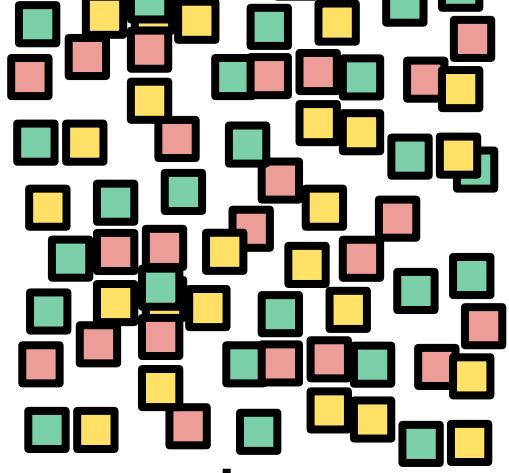
Apache Beam: Portability in the times of Real-time Streaming

Pablo Estrada
Apache Beam Committer
Software Engineer @ Google
pabloem@apache.org - polecito@

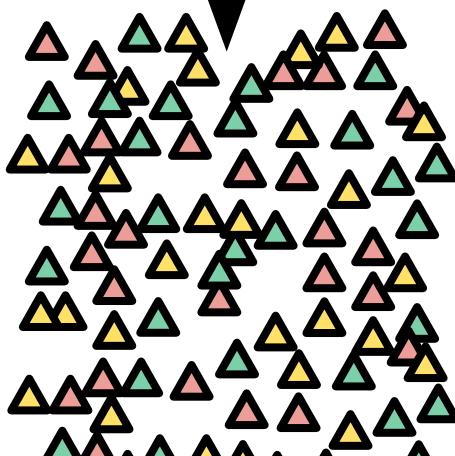
Agenda

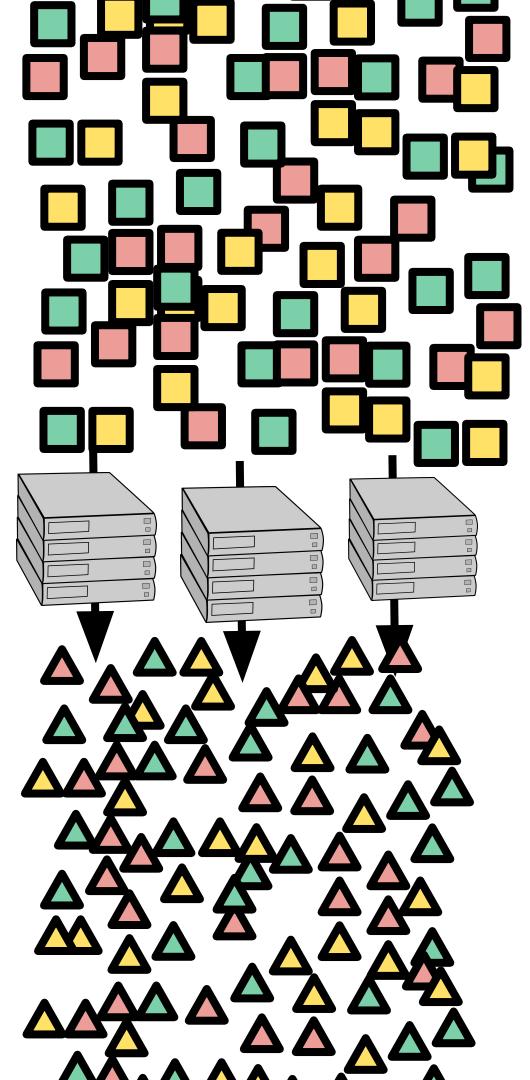
- Quick intro to Beam
- What is Portability
- Why we invest in Beam
- Flink + Beam integration architecture
 - Talk about the rough edges
- Bleeding edge of Beam / Upcoming features
- How to get involved

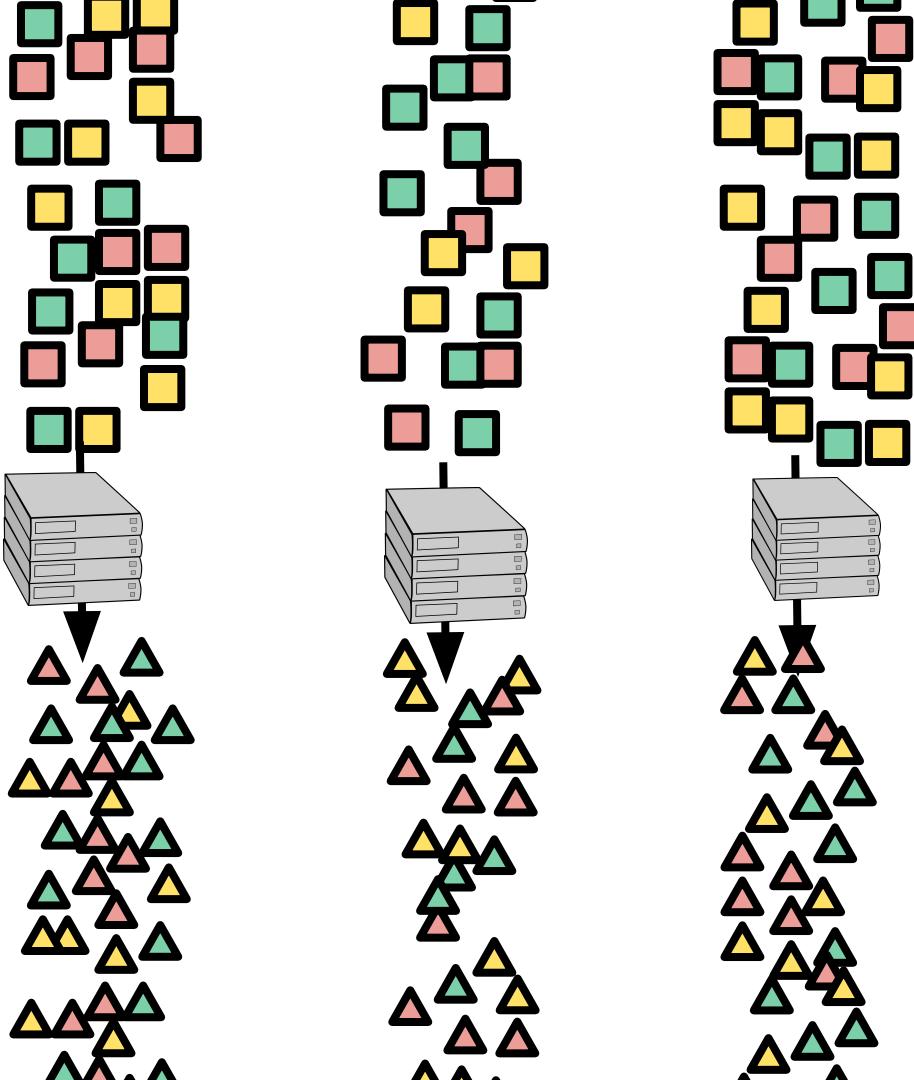


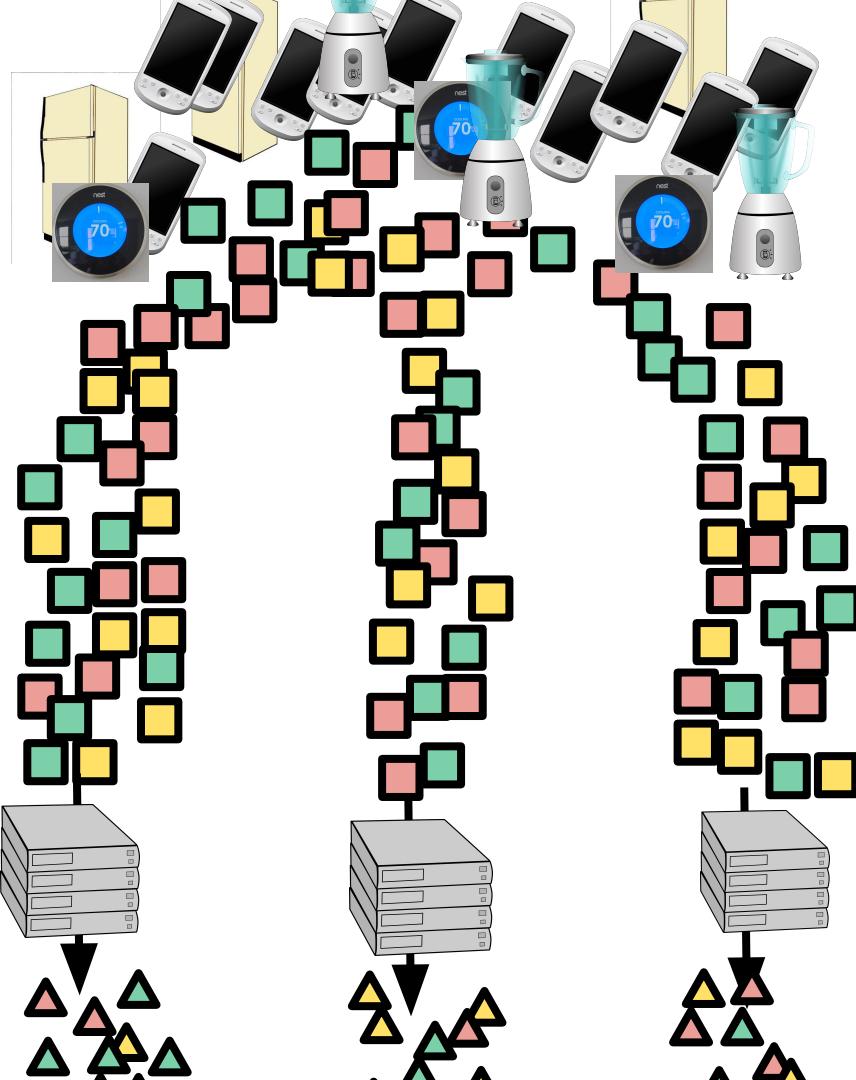


Computation

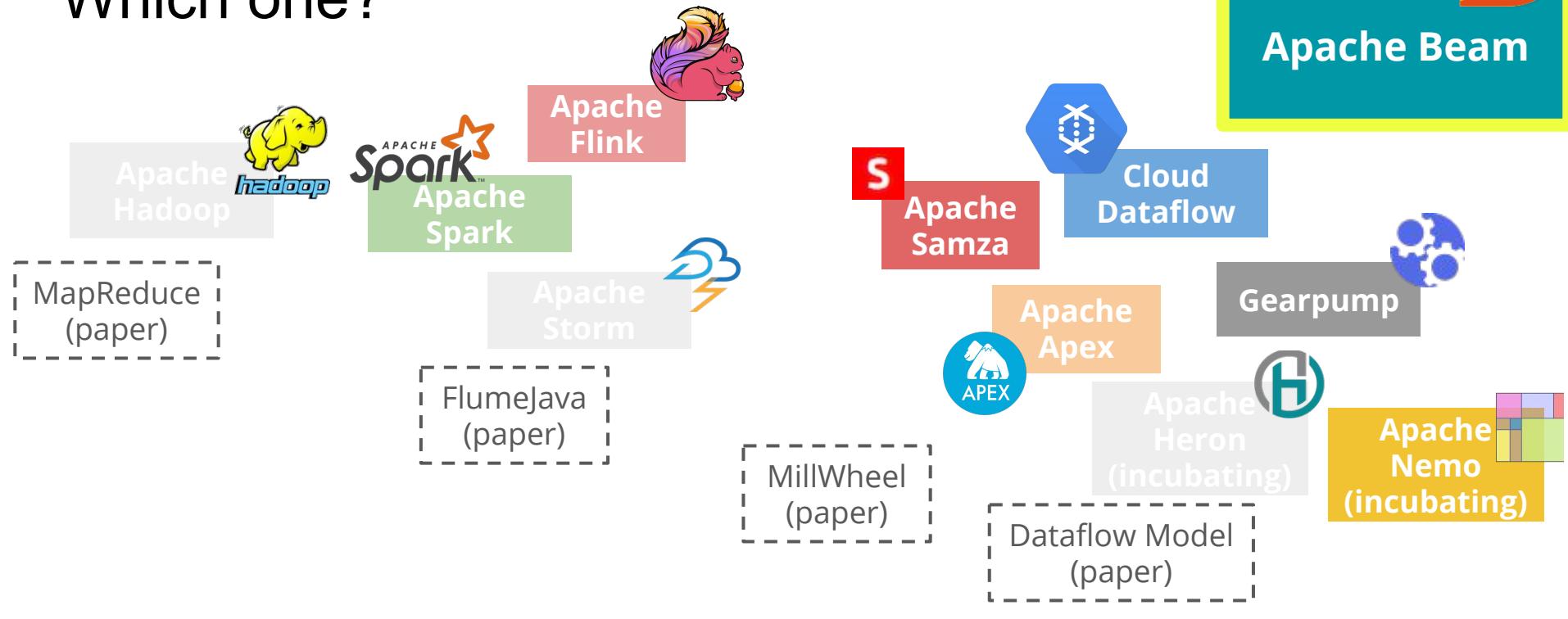








Which one?



2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018

Apache Beam?

Beam is a **portable, unified** programming
model for Batch and Streaming.

Unified?

Same API for Batch and Streaming computations

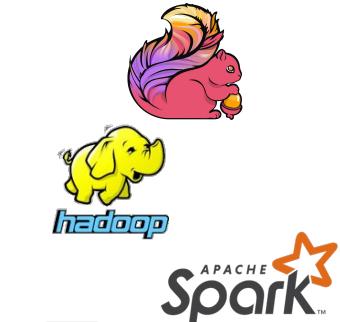
Portability?

Runner portability + Language portability

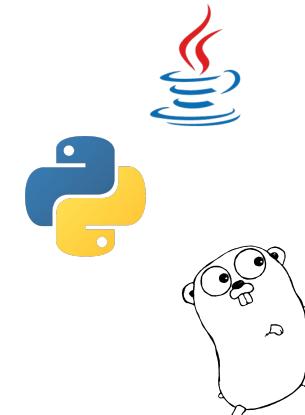
- Run your pipeline in the runner of your choice.
- Write your pipeline in the language that works / is most convenient for you.

Portability?

Runner portability + Language portability



⋮



⋮

The Beam Vision

Java

```
input.apply(  
    Sum.integersPerKey())
```

Python

```
input | Sum.PerKey()
```

Go

```
stats.Sum(s, input)
```

SQL

```
SELECT key, SUM(value) FROM  
input GROUP BY key
```

⋮

Sum Per Key



Cloud Dataflow



Apache Flink



Apache Spark



Apache Apex



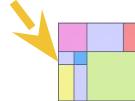
Gearpump



IBM Streams



Apache Samza



Apache Nemo
(incubating)

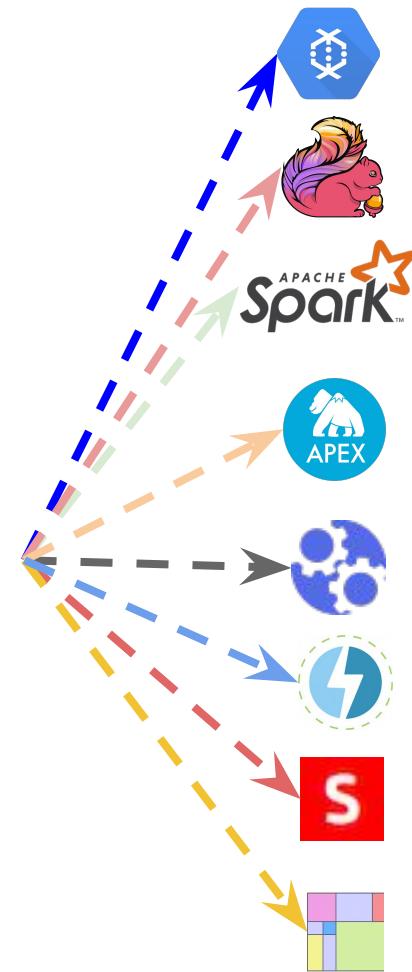
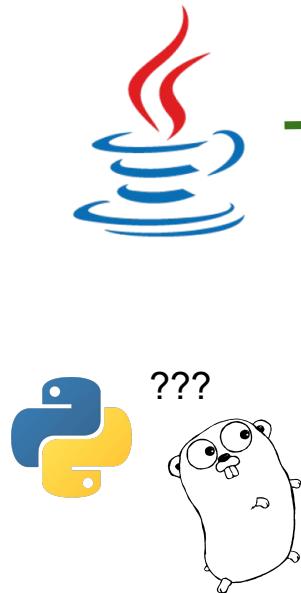
<https://beam.apache.org/roadmap/portability/>

⋮

Interesting Abstractions

- User defined functions / Business logic
- Event time vs Processing time
- DAG execution

Runner portability (today)



Cloud Dataflow

Apache Flink

Apache Spark

Apache Apex

Gearpump

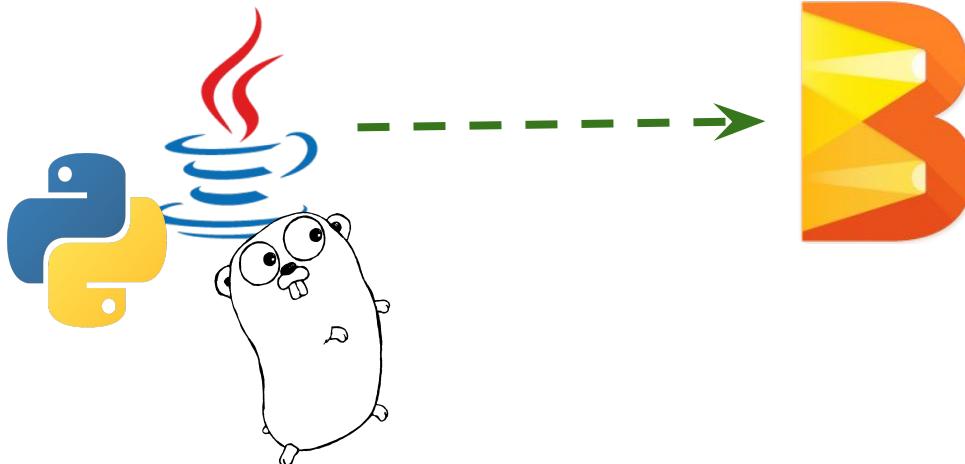
IBM Streams

Apache Samza

Apache Nemo
(incubating)

Language portability (today)

In-progress + MVP



Cloud Dataflow



Apache Flink



Apache Spark



Apache Apex



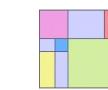
Gearpump



IBM Streams



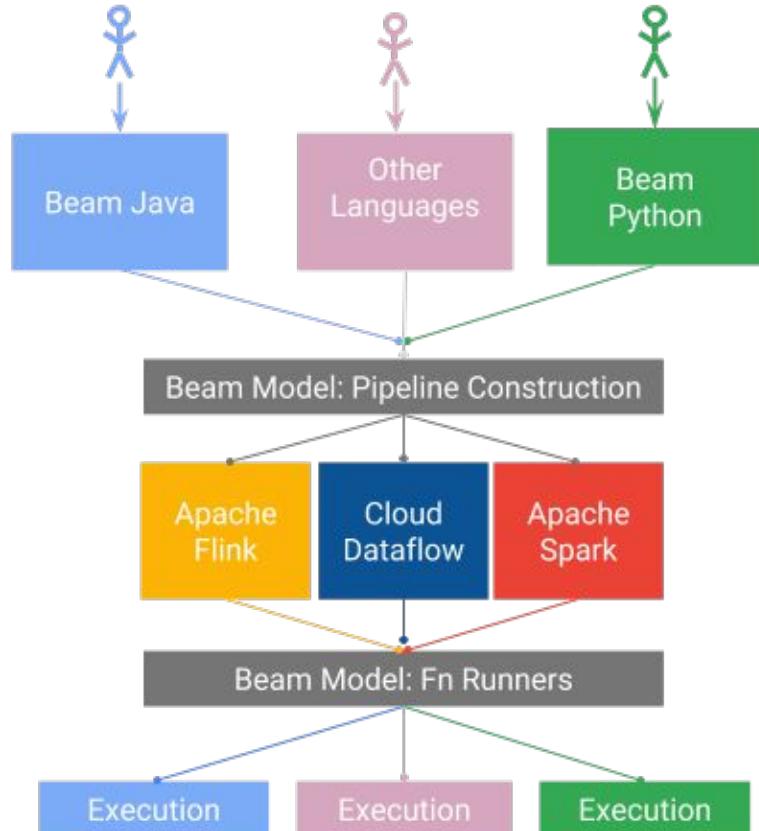
Apache Samza



Apache Nemo
(incubating)

Language portability

- Doesn't mean "all languages for free!"
- It does mean: Reducing the cost-of-entry for a new SDK
- Result of abstracting:
 - DAG execution from UDF / Business Logic



Why continue developing Beam?

- Open and free technologies are important for Cloud
 - Working on Apache Beam helps us make Google Cloud Dataflow a better product
 - Try and meet customers where they are: On prem? On other clouds? On our own cloud?
- Contribute to the big data ecosystem
 - Some awesome collaborations between Beam / Flink / other projects
 - Single APIs for Batch and Streaming
 - Event Time Processing / Windowing
 - Streaming SQL
 - Splittable DoFn (Splittable / Redistributable Maps)

Investments in Beam



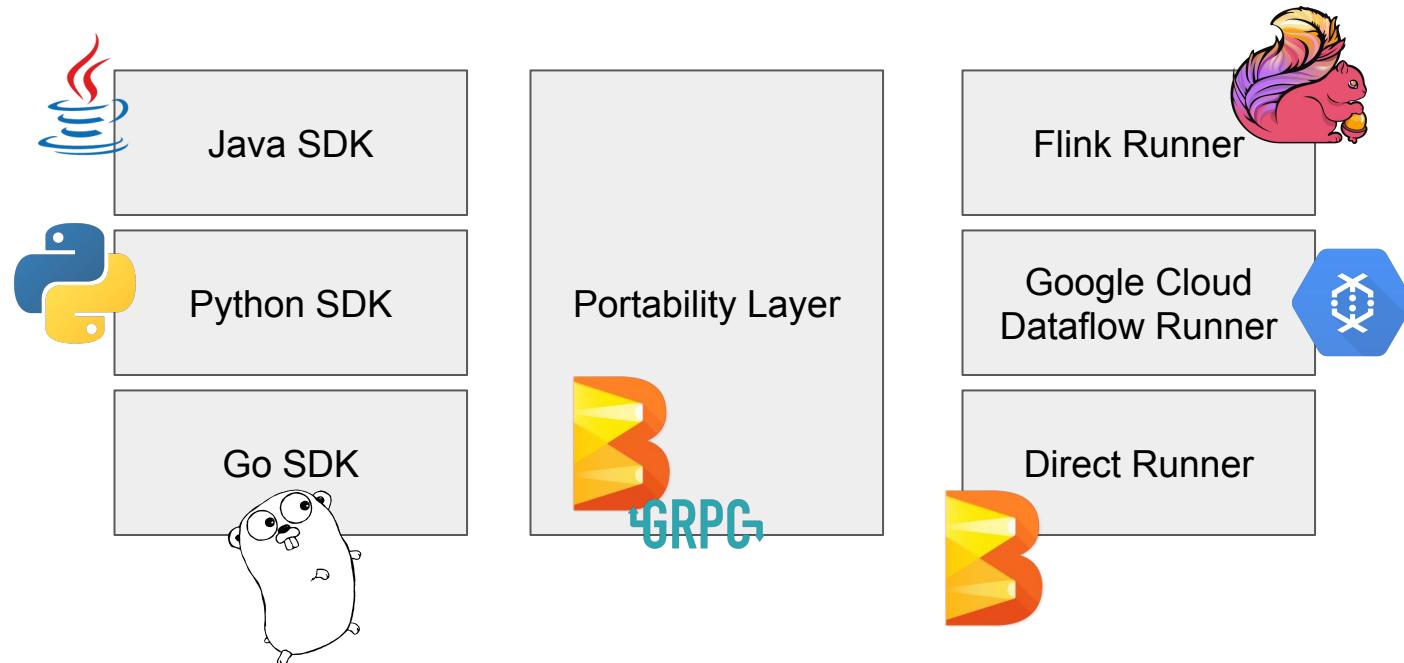
Model
Analysis

TensorFlow
Transform

Data
Validation



Beam today



Go Word-count

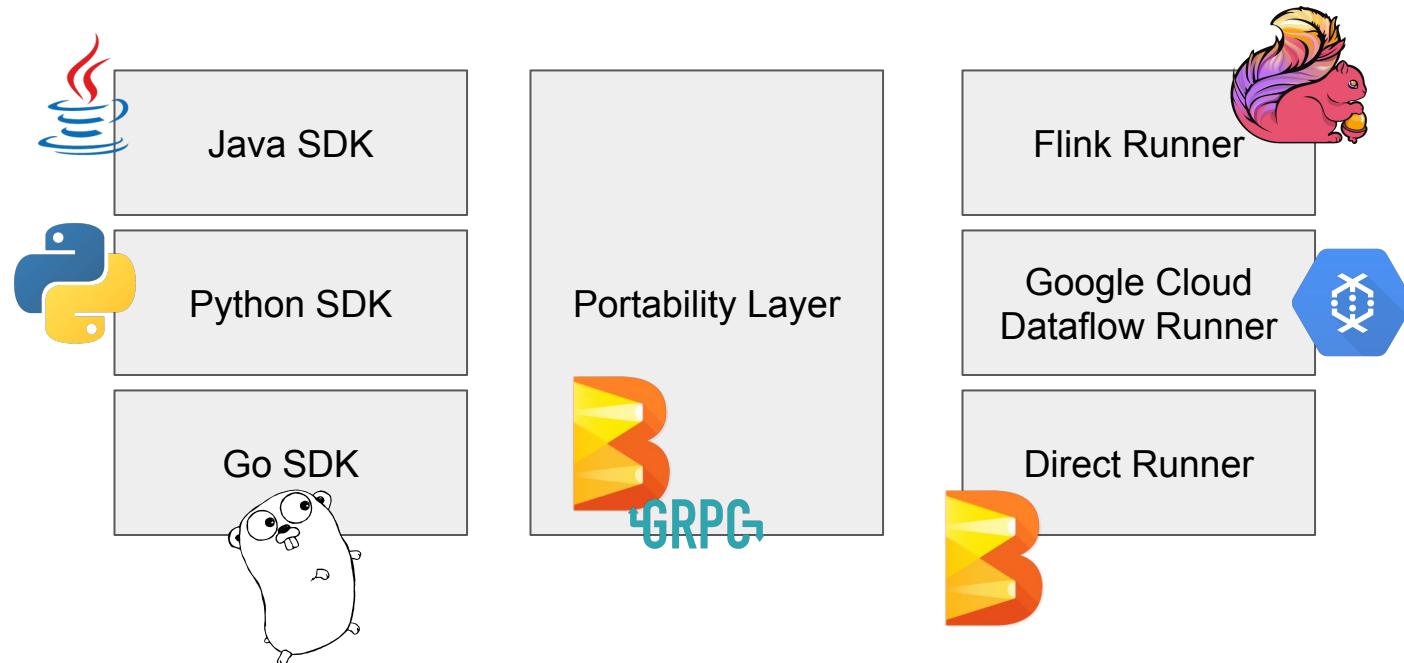
```
func main() {
    flag.Parse()
    beam.Init()

    p := beam.NewPipeline()
    s := p.Root()

    lines := textio.Read(s, *input)
    counted := CountWords(s, lines)
    formatted := beam.ParDo(s, formatFn, counted)
    textio.Write(s, *output, formatted)

    beamx.Run(context.Background(), p)
}
```

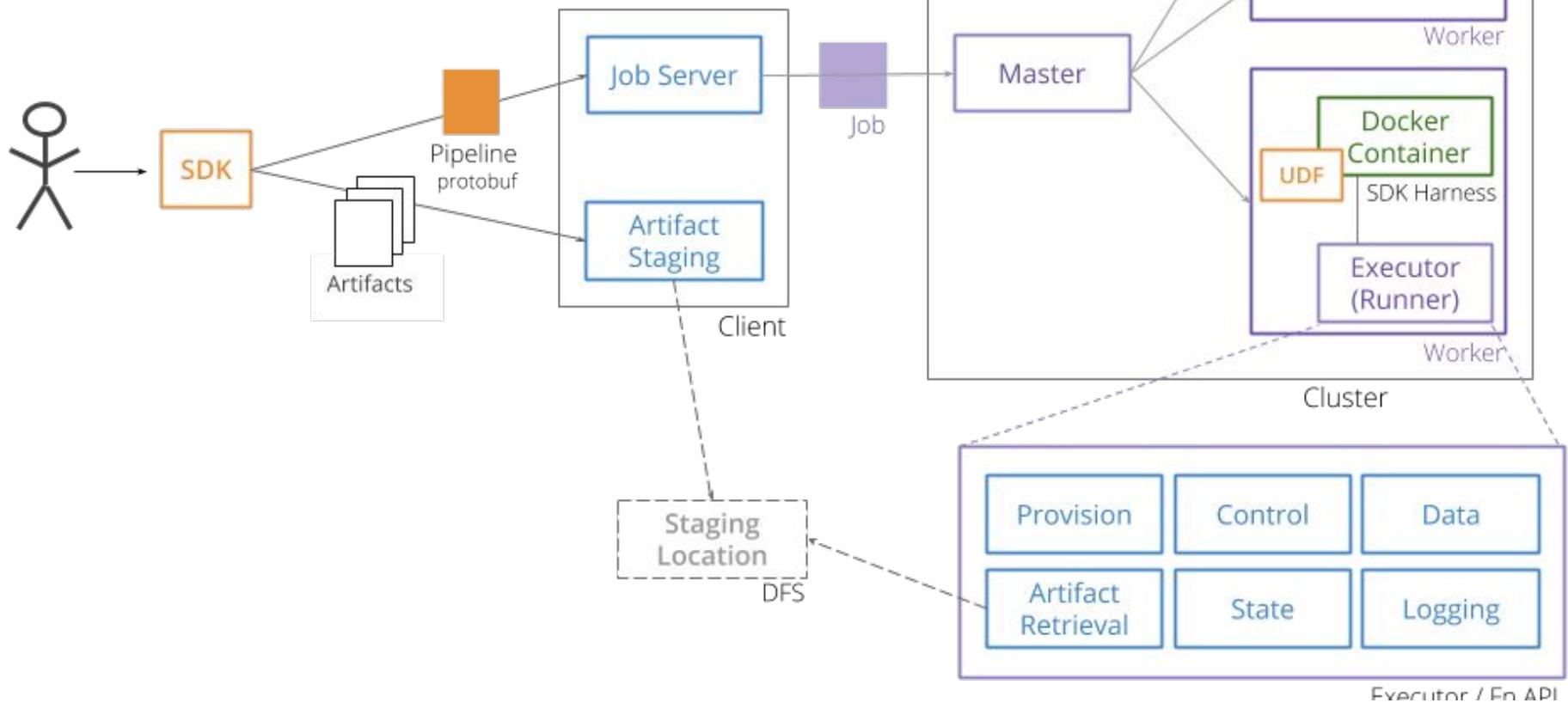
Beam today



Apache Beam Portable Flink Runner Architecture

- Executor setup (Language environment)
- State, Logging, Artifact retrieval, etc..
- Job Submission API
- Artifact Staging API
- ...

Apache Beam Portable Flink Runner Architecture



Apache Beam Portable Flink Runner Rough Edges

- Data Sources
 - Parallel Reader API not yet there
 - “Splittable DoFn”
- Deployment
 - Still a little awkward, but working to improve
- Very large scale
 - TFX pipelines generate lots of Flink tasks (~250 tasks for basic pipelines)
 - Complex data exchange: many small messages and few >XXMB messages.
 - Able to run pipelines with 100s of MB as input, discovering issues with GBs.

Beam today (bleeding edge)

- Experimenting with multi-language pipelines

Beam today (bleeding edge)

- Python pipeline with Java-based Count

```
65  def run(p, input_file, output_file):
66      # Read the text file[pattern] into a PCollection.
67      lines = p | 'read' >> ReadFromText(input_file)          Read file
68
69      counts = (lines
70              | 'split' >> (beam.ParDo(WordExtractingDoFn()
71                                .with_output_types(bytes))
72              | 'count' >> beam.ExternalTransform(
73                                'pytest:beam:transforms:count', None, EXPANSION_SERVICE_ADDR))
74
75      # Format the counts into a PCollection of strings.
76      def format_result(word_count):
77          (word, count) = word_count
78          return '%s: %d' % (word, count)
79
80      output = counts | 'format' >> beam.Map(format_result)
81
82      # Write the output using a "Write" transform that has side effects.
83      # pylint: disable=expression-not-assigned
84      output | 'write' >> WriteToText(output_file)           Write out
85
86      result = p.run()
87      result.wait_until_finish()
88
```

Features in the roadmap

- Great connectors for Beam on Python + Go
- Multi-language pipelines (e.g. SQL + Python ML)

How can I try it out?

- Clone Beam repository and try running our tests?
 - <https://github.com/apache/beam>
- Write a Go pipeline (and run it on Flink!)
 - Or Python pipeline :)
 - <https://beam.apache.org/roadmap/portability/#python-on-flink>
- Try TFX (<https://www.tensorflow.org/tfx/>)

Want to get involved?

Connect with us in the way you like best:

- Follow [@ApacheBeam](https://twitter.com/ApacheBeam) on Twitter
- Subscribe to the Apache Beam Youtube channel
- Chat on <https://the-asf.slack.com/#beam>
- User discussions on user@beam.apache.org
 - (to subscribe, send empty mail to user-subscribe@beam.apache.org)
- Development discussions on dev@beam.apache.org
 - (to subscribe, send empty mail to dev-subscribe@beam.apache.org)

END

ADVENTURES IN SCALING FROM ZERO TO 5 BILLION DATA POINTS PER DAY

Dave Torok

Distinguished Architect

Comcast Corporation

2 April, 2019

Flink Forward – San Francisco 2019



COMCAST CUSTOMER RELATIONSHIPS

**30.3 MILLION OVERALL CUSTOMER
RELATIONSHIPS AT 2018 YEAR END**

**25.1 MILLION RESIDENTIAL HIGH-SPEED
INTERNET CUSTOMERS AT 2018 YEAR
END**

**1.2 MILLION RESIDENTIAL HIGH-SPEED
INTERNET CUSTOMER NET ADDITIONS IN
2018**



DELIVER THE ULTIMATE CUSTOMER EXPERIENCE

IS THE CUSTOMER HAVING A GOOD EXPERIENCE
FOR HIGH SPEED DATA (HSD) SERVICE?



IF THE CUSTOMER ENGAGES US DIGITALLY, CAN
WE OFFER A SELF-SERVICE SOLUTION?



IF THERE IS AN ISSUE CAN WE OFFER OUR AGENTS
AND TECHNICIANS A DIAGNOSIS TO HELP SOLVE
THE PROBLEM QUICKER?



REDUCE THE TIME TO RESOLVE ISSUES

REDUCE COST TO THE BUSINESS AND THE
CUSTOMER

HIGH LEVEL CONCEPT



Comcast collects, stores, and uses all data in accordance with our privacy disclosures to users and applicable laws.

CUSTOMER EXPERIENCE INDICATORS



Comcast collects, stores, and uses all data in accordance with our privacy disclosures to users and applicable laws.



COMCAST

NINE ADVENTURES IN SCALING

THE
TRIGGER AND
DIAGNOSIS
PROBLEM

THE
REST
PROBLEM

THE
INEFFICIENT
OBJECT HANDLING
PROBLEM

THE
FEATURE STORE
PROBLEM

THE
VOLUME
PROBLEM

THE
CUSTOMER STATE
PROBLEM

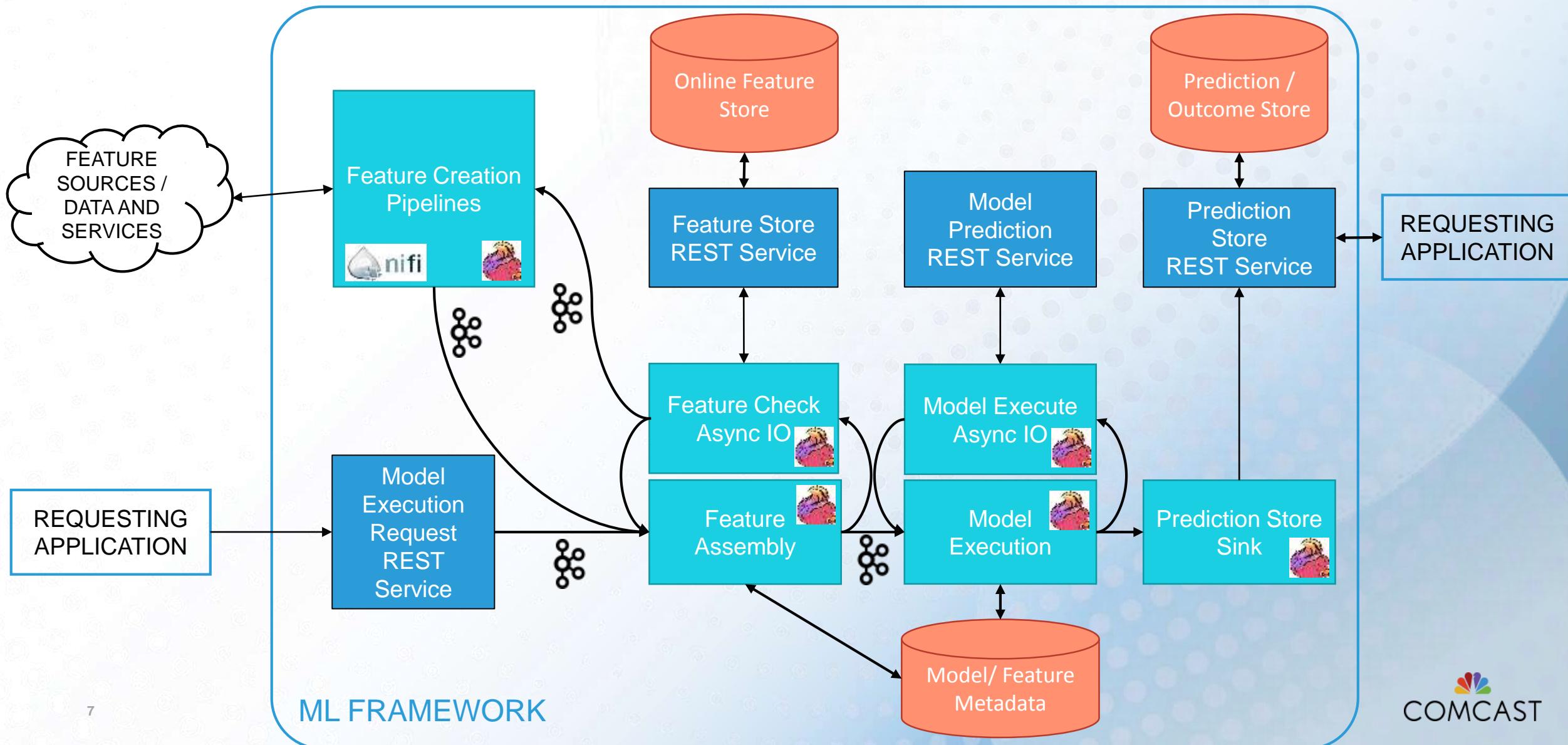
THE
CHECKPOINT
PROBLEM

THE
TRIGGER AND
DIAGNOSIS
PROBLEM

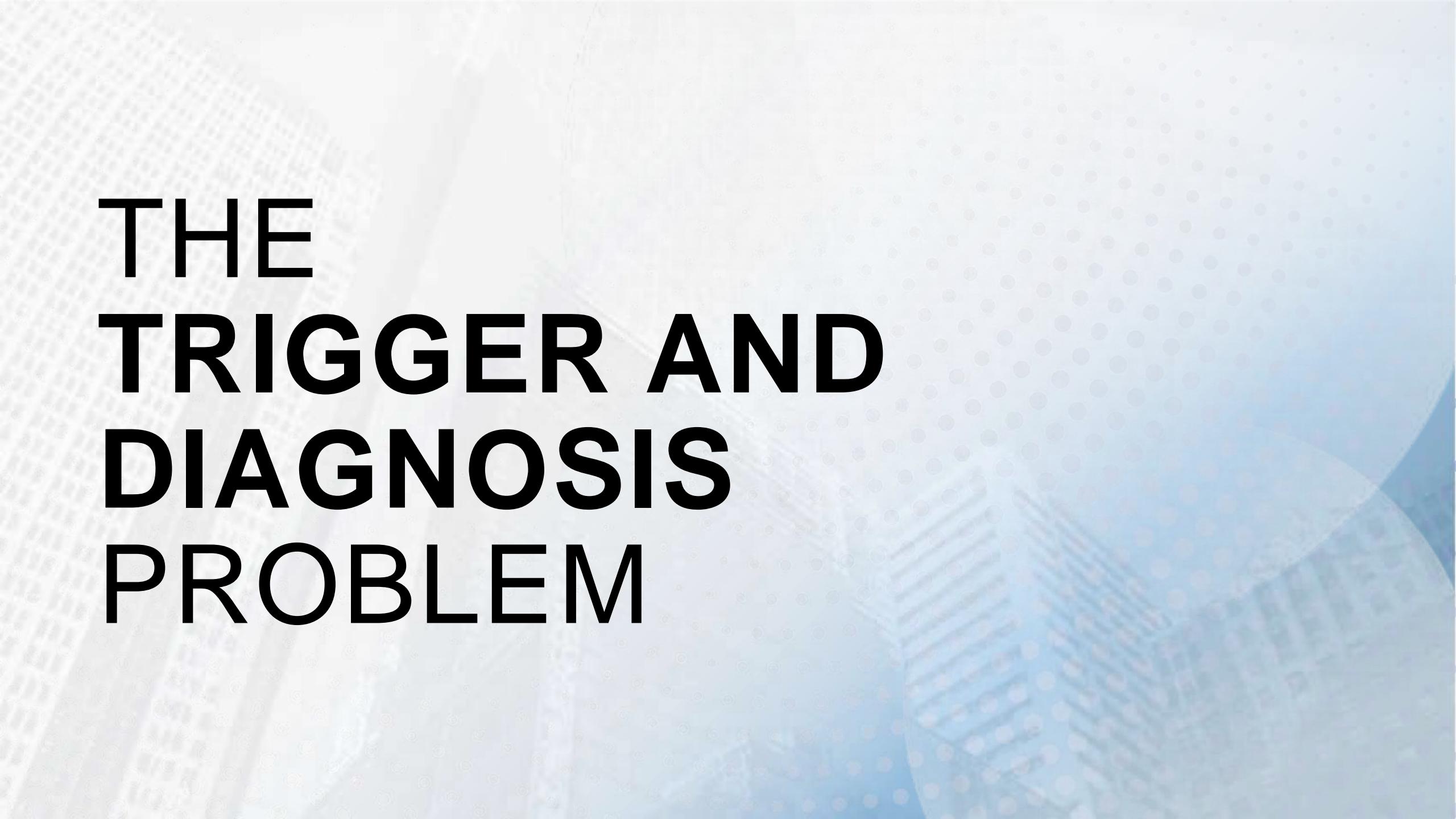
THE
REALLY HIGH VOLUME
AND FEATURE STORE
PROBLEM #2

ML FRAMEWORK ARCHITECTURE – 2018

"EMBEDDING FLINK THROUGHOUT AN OPERATIONALIZED STREAMING ML LIFECYCLE" – SF FLINK FORWARD 2018



THE TRIGGER AND DIAGNOSIS PROBLEM



MAY 2018 – INITIAL VOLUMES

INDICATOR #1

9 MILLION

INDICATOR #2

166 MILLION

INDICATOR #3

1.2 MILLION

INDICATOR #4

2500 (SMALL TRIAL)

TOTAL

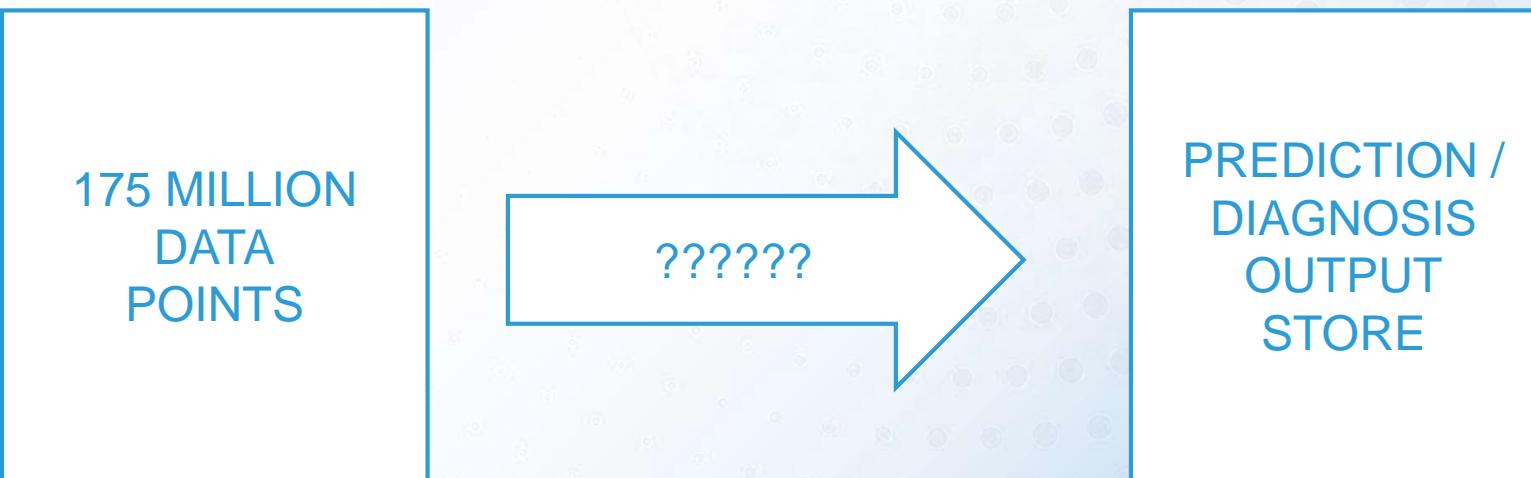
175

MILLION / DAY

Comcast collects, stores, and uses all data in accordance with our privacy disclosures to users and applicable laws.



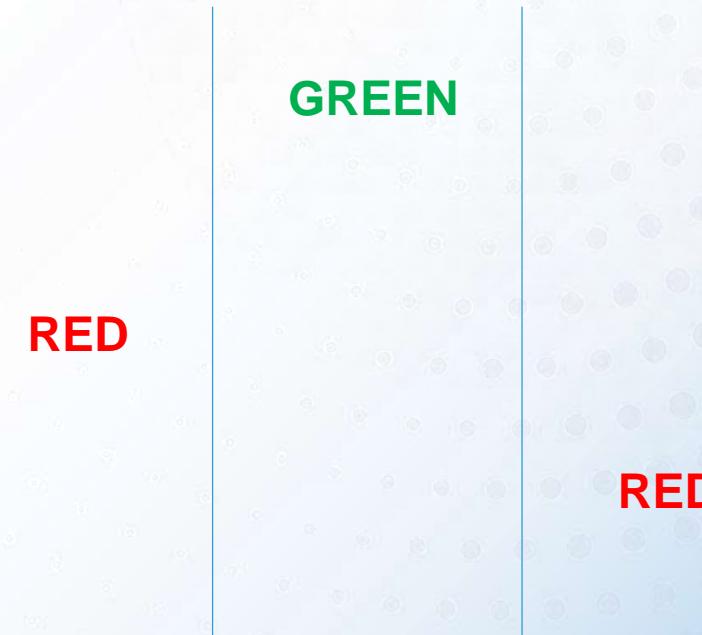
175 MILLION PREDICTIONS?



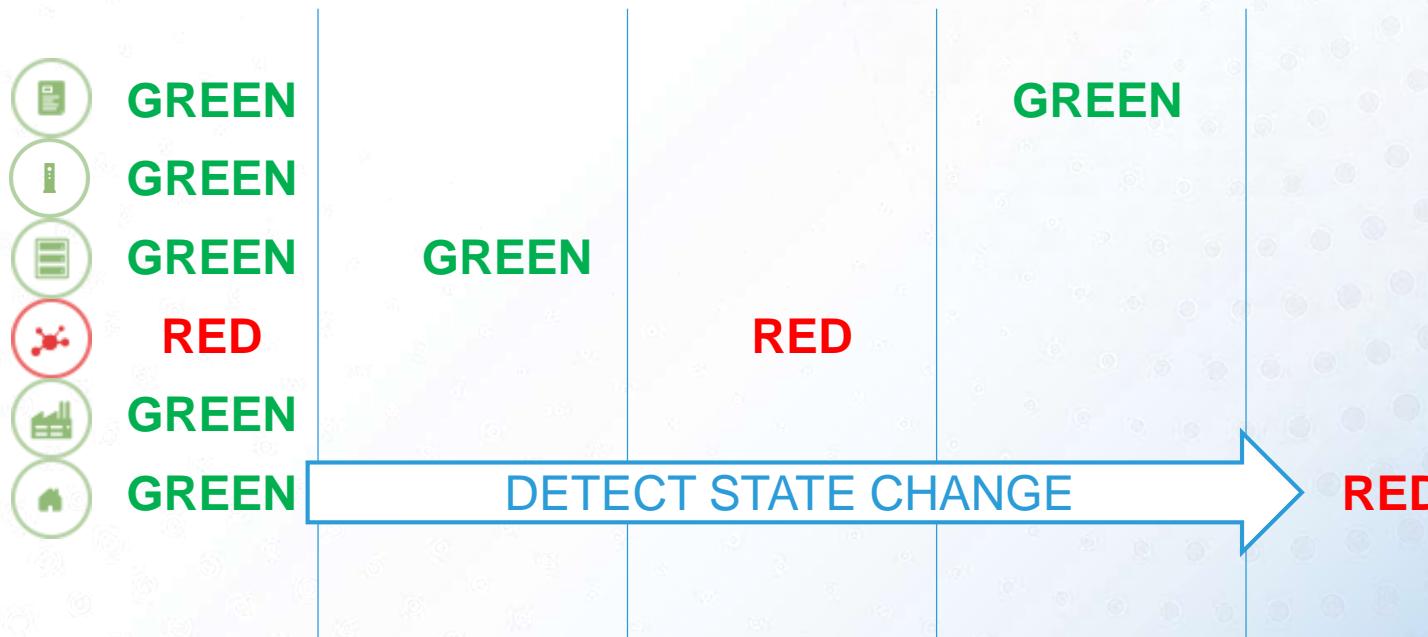
INTRODUCE A 'TRIGGER' CONDITION

	GREEN
	GREEN
	GREEN
	RED
	GREEN
	GREEN

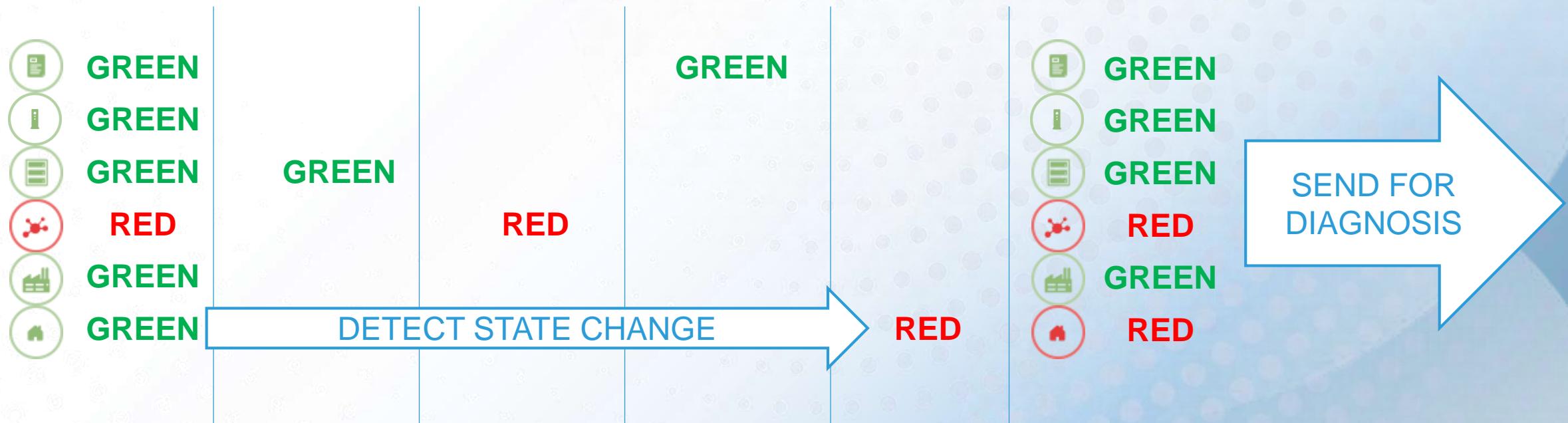
INTRODUCE A ‘TRIGGER’ CONDITION



INTRODUCE A ‘TRIGGER’ CONDITION

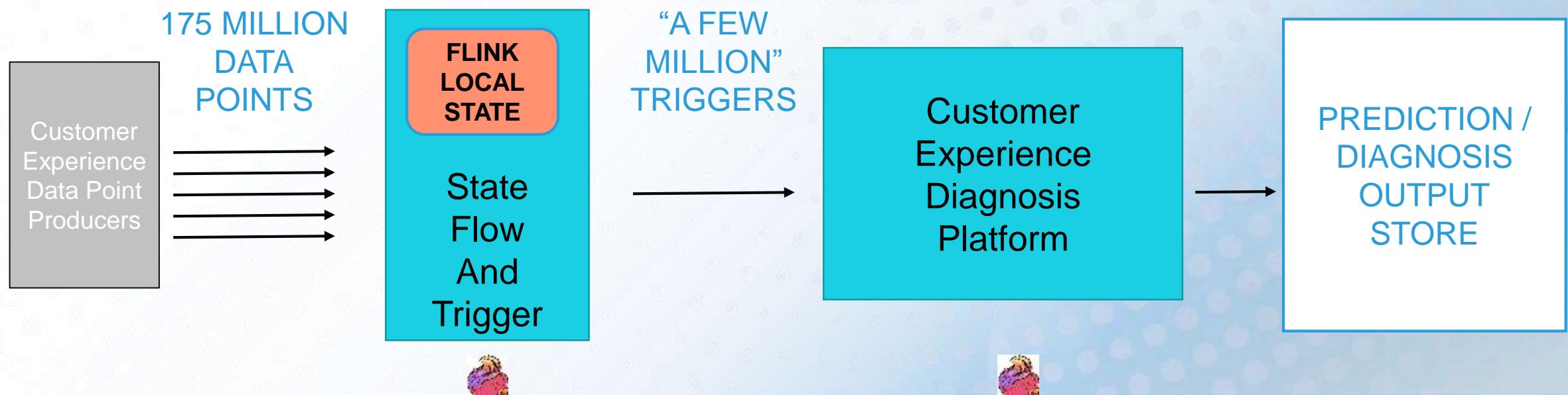


INTRODUCE A ‘TRIGGER’ CONDITION



STATE MANAGER AND TRIGGER

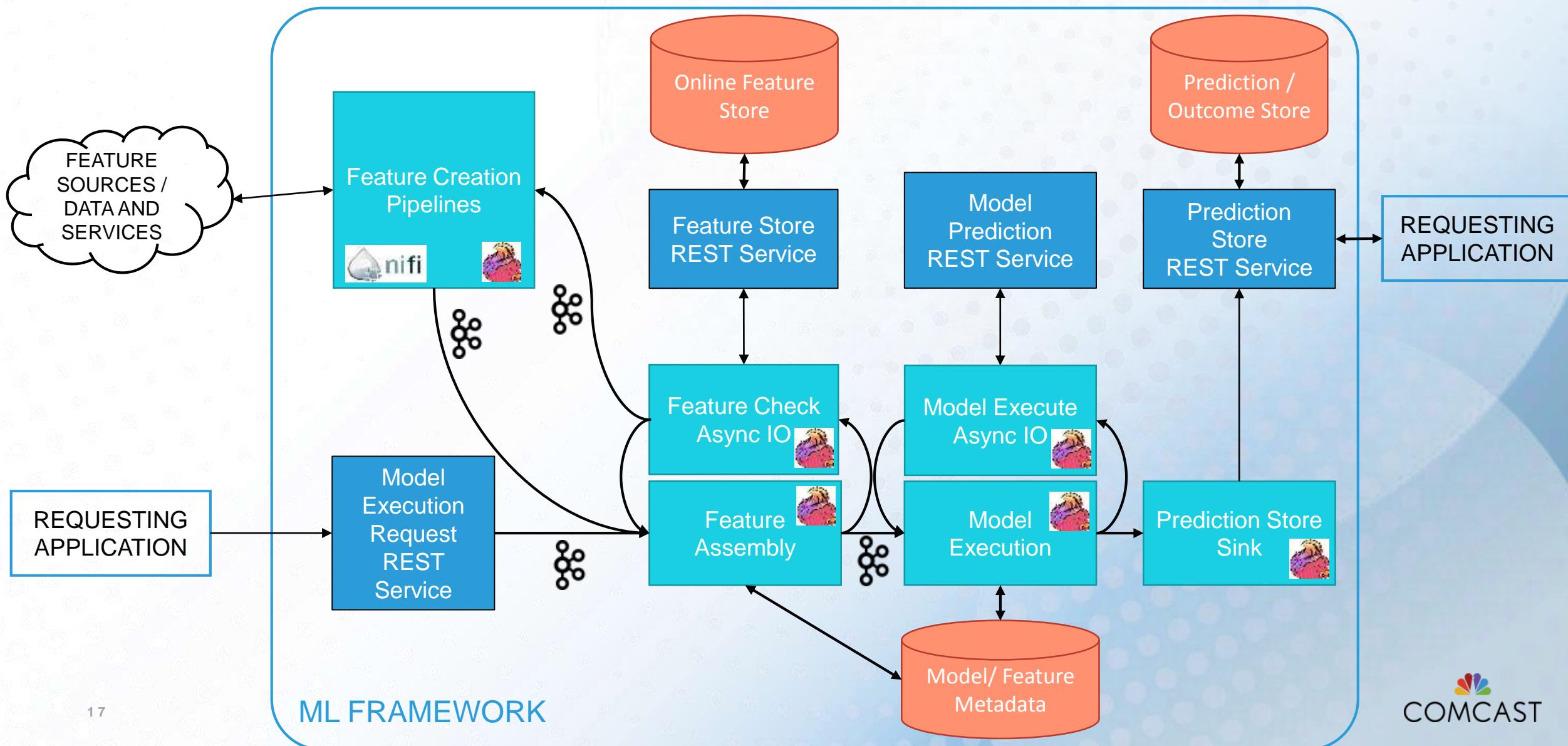
INTRODUCE FLINK LOCAL STATE



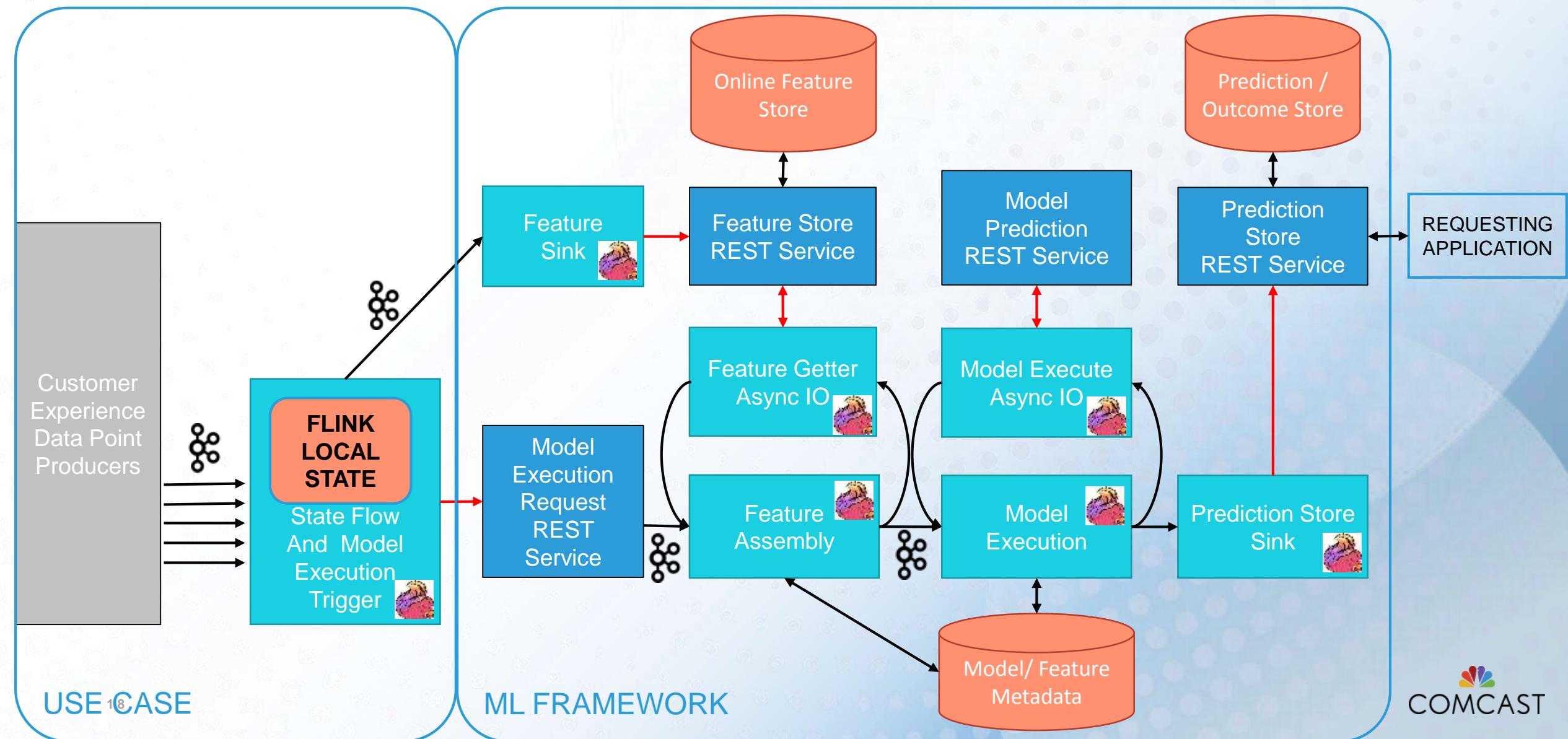
THE REST PROBLEM

ML FRAMEWORK ARCHITECTURE – 2018

"EMBEDDING FLINK THROUGHOUT AN OPERATIONALIZED STREAMING ML LIFECYCLE" – SF FLINK FORWARD 2018



INITIAL PLATFORM ARCHITECTURE



ORIGINAL ML FRAMEWORK ARCHITECTURE



EVERY COMMUNICATION
WAS A REST SERVICE



ML FRAMEWORK
ISOLATED FROM DATA
POINT USE CASE
BY DESIGN



FLINK ASYNC + REST IS
DIFFICULT TO SCALE
ELASTICALLY

TUNING ASYNC I/O – LATENCY AND VOLUME

MAX THROUGHPUT PER SECOND = (SLOTS) * (THREADS) * (1000 / TASK DURATION MSEC)
(12 SLOTS) * (20 THREADS) * (1000 / 300) = 800 / SECOND

APACHE HTTP CLIENT / EXECUTOR THREADS

setMaxTotal

setDefaultMaxPerRoute (THREADS)

FLINK ASYNC MAX OUTSTANDING

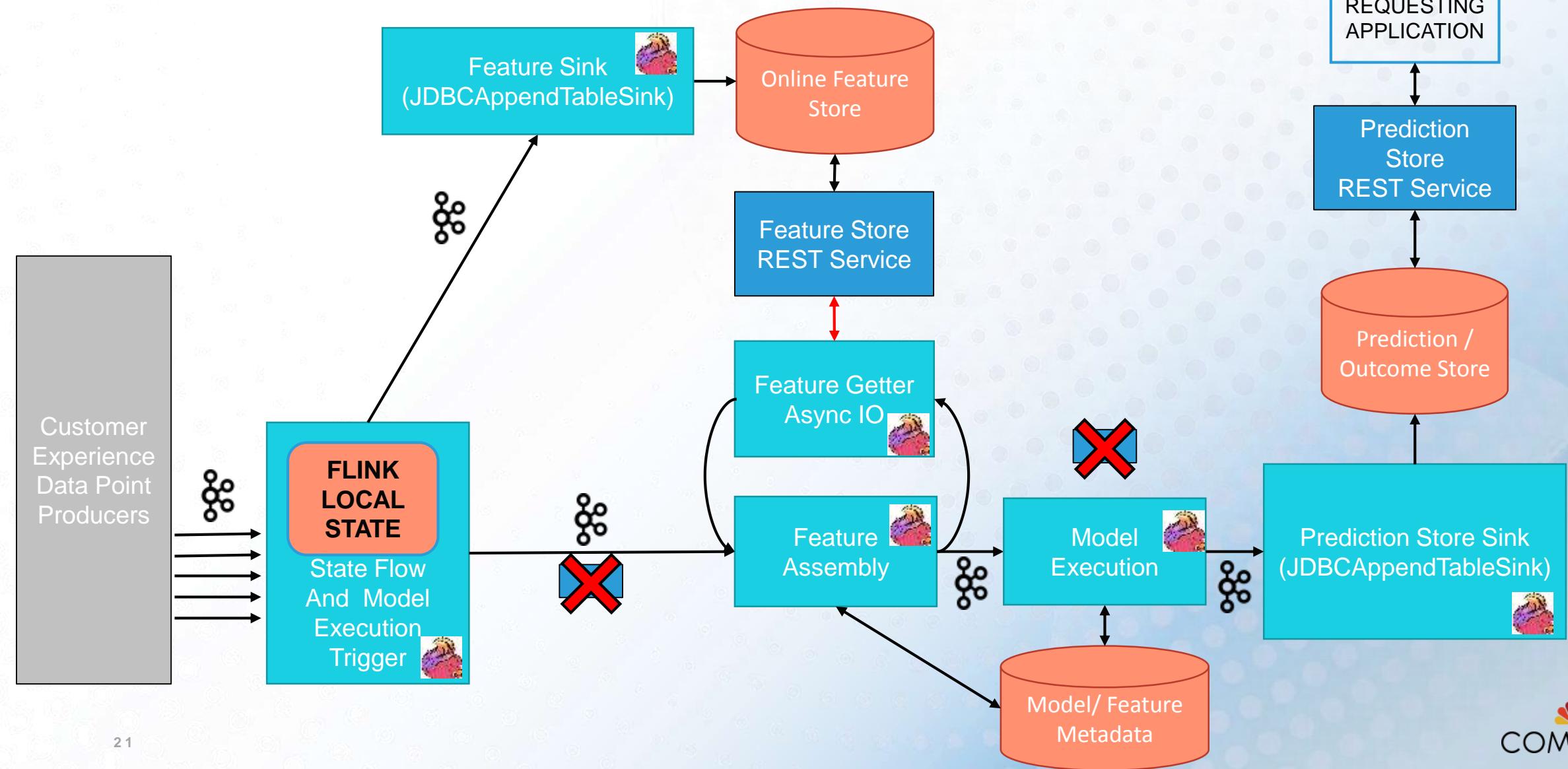
RichAsyncFunction “Capacity”

AsyncDataStream.unorderedWait() with max capacity parameter

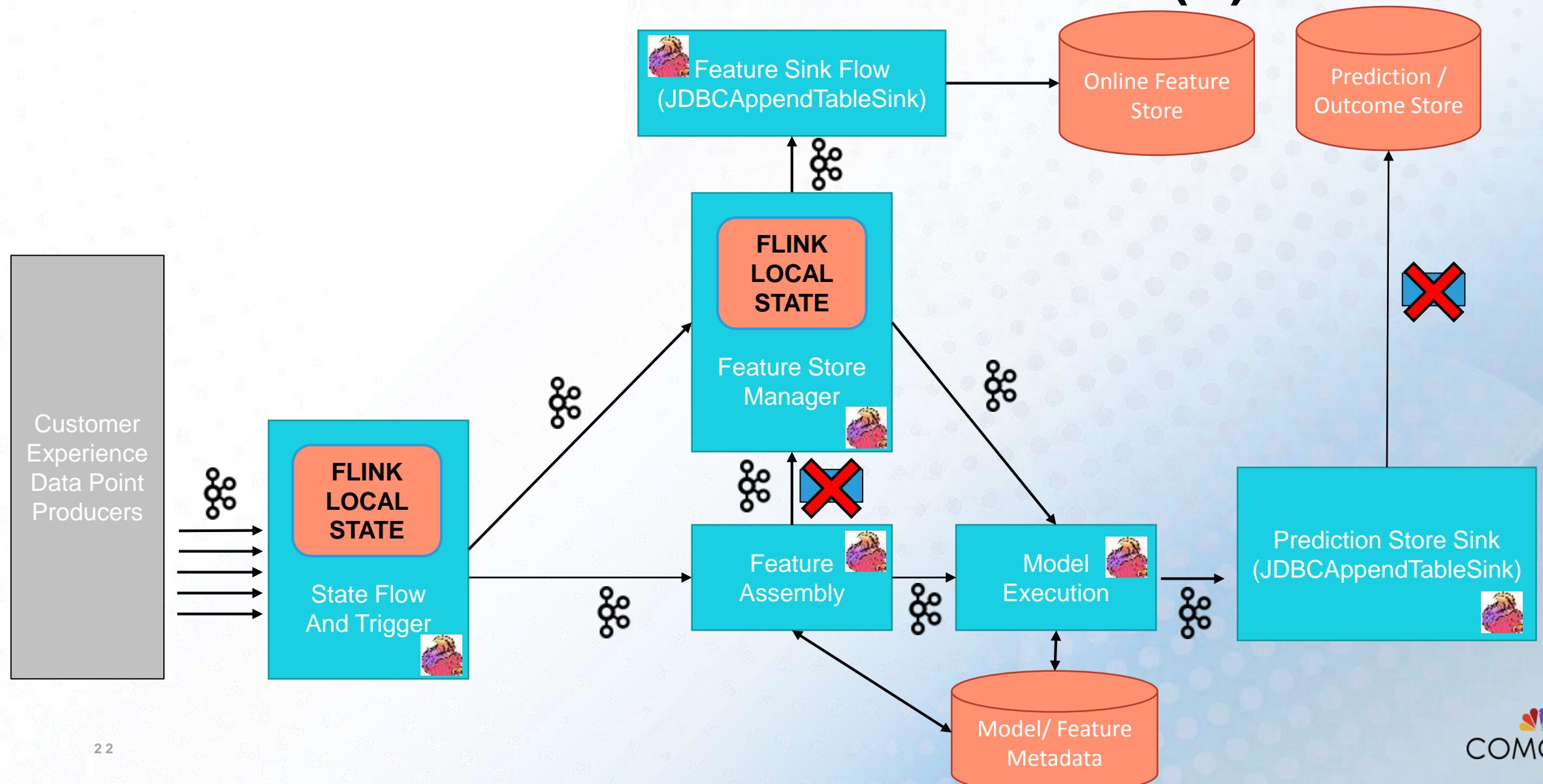
RATE THROTTLING

```
if (numberOfRequestsPerPeriod > 0) {  
    int numberOfProcesses = getRuntimeContext()  
        .getNumberOfParallelSubtasks();  
  
    numberOfRequestsPerPeriodPerProcess =  
        Math.max(1, numberOfRequestsPerPeriod /  
            numberOfProcesses);  
}
```

REPLACE REST CALLS WITH KAFKA (I)



REPLACE REST CALL WITH KAFKA (II)



THE INEFFICIENT OBJECT HANDLING PROBLEM

HAS THIS EVER HAPPENED TO YOU?

```
String transformedJson =  
new JoltJsonTransformer()  
    .transformJson(myJsonValue,  
    pTool.getRequired(SPEC_PATH));  
  
JSONObject inputPayload = new  
JSONObject(transformedJson);  
try {  
    aType = inputPayload.getJSONObject("info")  
        .getJSONObject("aType")  
        .getString("string");  
} catch (Exception e) {  
    // ignore, invalid object  
    aType = STATUS_INVALID;  
}
```

New “JOLT” JSON Transformer
(but don’t cache it)
→ New ObjectMapper()
(but don’t cache it)
→ Parse JSON string into Map
→ Load and parse “Transform” JSON
from the resource path
(but don’t cache it)
→ Serialize transformed Map *back*
into string result

HAS THIS EVER HAPPENED TO YOU?

```
String transformedJson =  
new JoltJsonTransformer()  
    .transformJson(myJsonValue,  
    pTool.getRequired(SPEC_PATH));  
  
JSONObject inputPayload = new  
JSONObject(transformedJson);  
  
try {  
    aType = inputPayload.getJSONObject("info")  
        .getJSONObject("aType")  
        .getString("string");  
} catch (Exception e) {  
    // ignore, invalid object  
    aType = STATUS_INVALID;  
}
```

New “JOLT” JSON Transformer
(but don’t cache it)
→ New ObjectMapper()
(but don’t cache it)
→ Parse JSON string into Map
→ Load and parse “Transform” JSON
from the resource path
(but don’t cache it)
→ Serialize transformed Map *back*
into string result
Then let’s make a NEW JSONObject
from that string

HAS THIS EVER HAPPENED TO YOU?

```
String transformedJson =  
new JoltJsonTransformer()  
    .transformJson(myJsonValue,  
        pTool.getRequired(SPEC_PATH));  
  
JSONObject inputPayload = new  
JSONObject(transformedJson);  
try {  
  
    aType = inputPayload.getJSONObject("info")  
        .getJSONObject("aType")  
        .getString("string");  
  
} catch (Exception e) {  
    // ignore, invalid object  
    aType = STATUS_INVALID;  
}
```

New “JOLT” JSON Transformer
(but don’t cache it)
→ New ObjectMapper()
(but don’t cache it)
→ Parse JSON string into Map
→ Load and parse “Transform” JSON
from the resource path
(but don’t cache it)
→ Serialize transformed Map *back*
into string result
Then let’s make a NEW JSONObject
from that string
Go deep into the path to get value

HAS THIS EVER HAPPENED TO YOU?

```
String transformedJson =  
new JoltJsonTransformer()  
    .transformJson(myJsonValue,  
        pTool.getRequired(SPEC_PATH));  
  
JSONObject inputPayload = new  
JSONObject(transformedJson);  
try {  
    aType = inputPayload.getJSONObject("info")  
        .getJSONObject("aType")  
        .getString("string");  
}  
} catch (Exception e) {  
    // ignore, invalid object  
    aType = STATUS_INVALID;  
}
```

New “JOLT” JSON Transformer
(but don’t cache it)
→ New ObjectMapper()
(but don’t cache it)
→ Parse JSON string into Map
→ Load and parse “Transform” JSON
from the resource path
(but don’t cache it)
→ Serialize transformed Map *back*
into string result

Then let’s make a NEW JSONObject
from that string
Go deep into the path to get value
Cast exception? STATUS_INVALID

SLIGHTLY MORE EFFICIENT SOLUTION

```
if (mapper == null)  
    mapper = new ObjectMapper();  
  
Map<?, ?> resultMap =  
JsonUtils.readValue(mapper, (String)  
result, Map.class);  
  
MapTreeWalker mapwalker = new  
MapTreeWalker(resultMap);  
final String aType =  
    mapwalker.step("info")  
    .step("aschemaInfo")  
    .step("aType")  
    .step("string")  
    .<String>get()  
    .orElse(STATUS_INVALID);
```

Cache our new ObjectMapper()

Parse JSON string into Map *ONCE*

Lightweight Map Traversal utility

Java Stream syntax

Optional.ofNullable

Optional.orElse

FLINK OBJECT EFFICIENCIES

```
StreamExecutionEnvironment env;  
env.getConfig().enableObjectReuse();
```

REDUCE OBJECT
CREATION AND
GARBAGE
COLLECTION

SEMANTIC ANNOTATIONS

```
@NonForwardedFields  
@ForwardedFields
```

BE CAREFUL OF
SIDE EFFECTS
WITHIN
OPERATORS

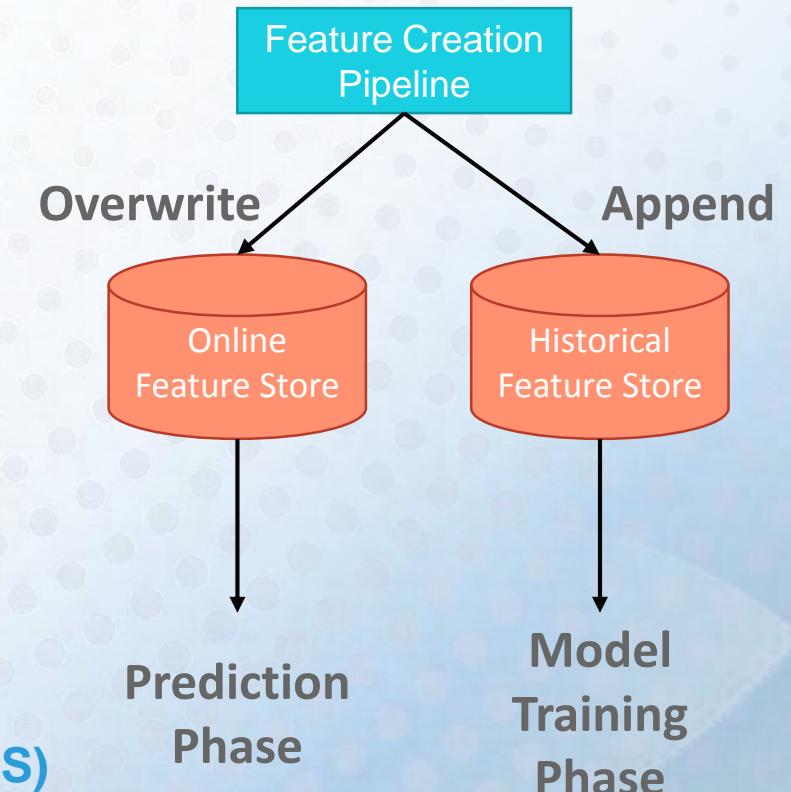
THE FEATURE STORE PROBLEM

WHAT IS A FEATURE STORE?

“FEATURE” IS A DATA POINT INPUT TO ML MODEL

WE NEED TO:

- **STORE ALL THE INPUT DATA POINTS**
- **SNAPSHOT AT ANY MOMENT IN TIME**
- **ASSOCIATE WITH A DIAGNOSIS (MODEL OUTPUT)**
- **HAVE ACCESS TO THE DATA POINTS (API FOR OTHER APPS)**
- **STORE ALL DATA POINTS FOR ML MODEL TRAINING**



FIRST TRY: AWS AURORA RDBMS

“HEY IT SHOULD GET US UP TO 10,000 / SECOND”

...THE UPSERT PROBLEM

DIDN'T WORK MUCH PAST 3,000 / SECOND

...SOON OUR INPUT RATE WAS 20,000 / SECOND



MITIGATION: STORE ONLY AT DIAGNOSIS TIME

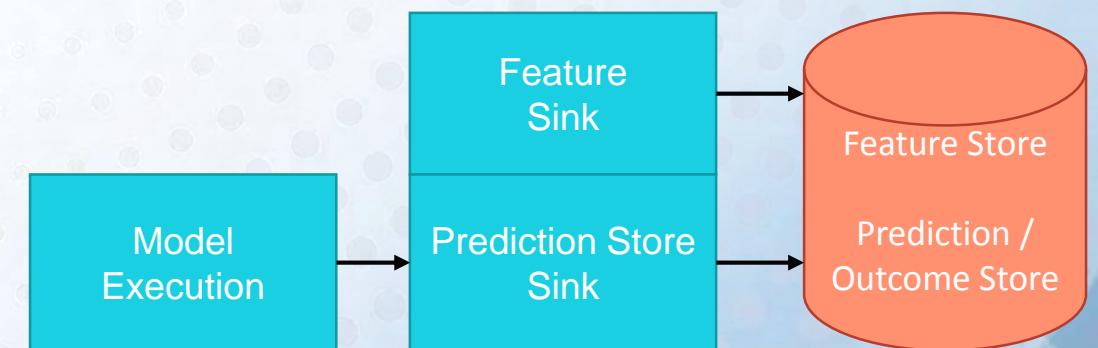
STOPPED STORING ALL RAW DATA POINTS

ONLY STORE DATA POINTS ALONGSIDE
DIAGNOSIS

CON: ONLY A SMALL % OF DATA POINTS

**CON: DATA NOT CURRENT, ONLY AS OF
RECENT DIAGNOSIS**

CON: LIMITED USEFULNESS



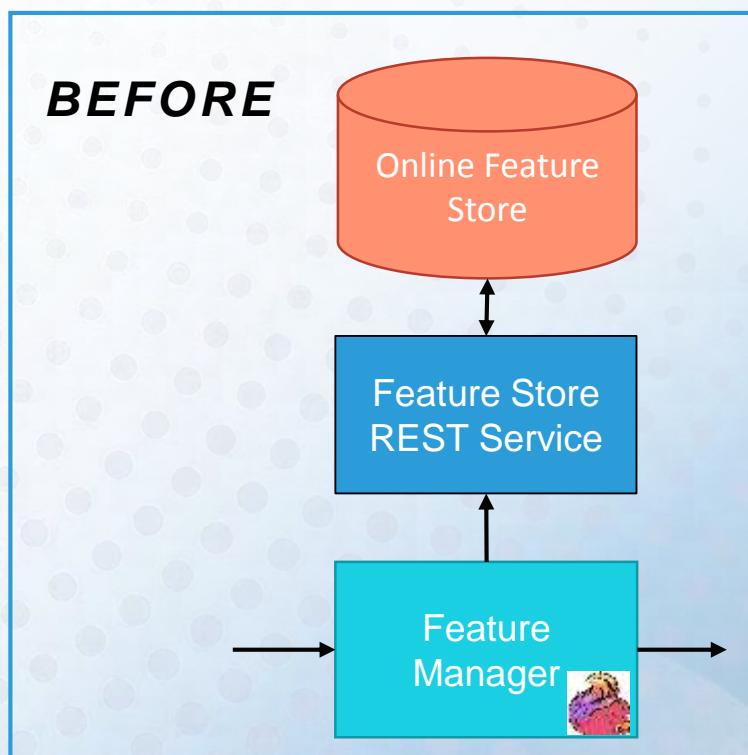
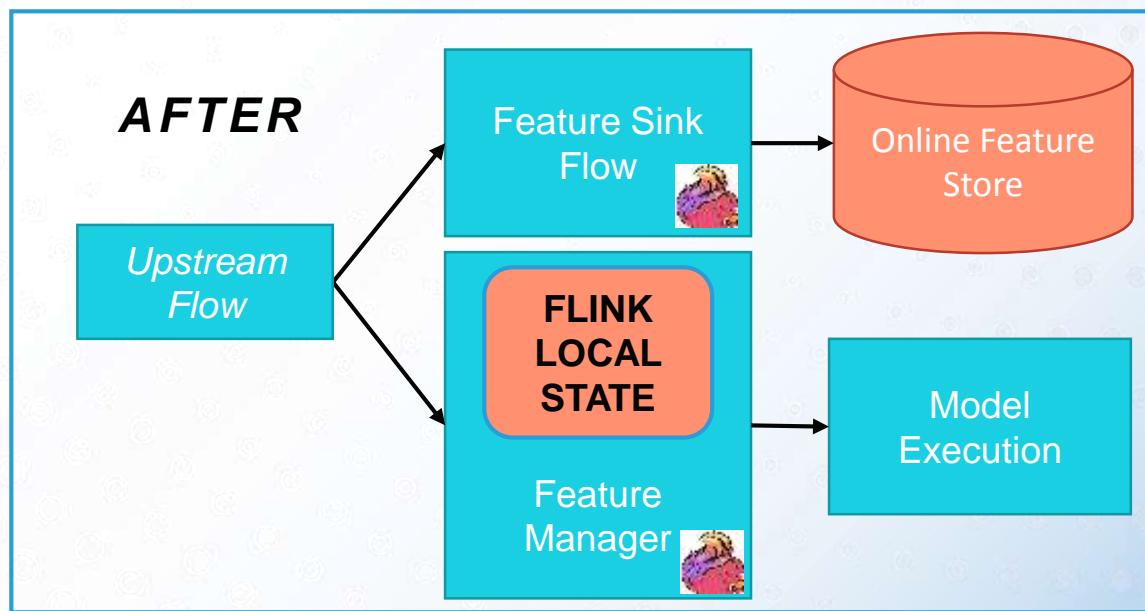
OPTIMIZING THE WRITE PATH

WHICH FLINK FLOW WRITES TO FEATURE STORE?

FEATURE STORE NOT NEEDED FOR TRIGGER

FEATURE STORE NOT NEEDED FOR MODEL EXECUTION

SEPARATE 'TRANSFORM AND SINK' FLOW



ALTERNATIVE DATA STORES

REDIS

STORE ONLY LATEST DATA POINTS?

CASSANDRA

COST AND COMPLEXITY?

FLINK QUERYABLE STATE

PRODUCTION HARDENING?

READ VOLUME?

TIME SERIES DB

DRUID

INFLUX

REDIS

WE ONLY NEED < 7 DAYS OF DATA, ONLY LATEST
VALUE, SO USE REDIS AS THE KV STORE

ACCESSIBLE BY OTHER CONSUMERS OR VIA A
SERVICE

COULDN'T PUT MORE THAN 8,000 / SECOND FOR A
'REASONABLE' CLUSTER SIZE

FLINK SINK CONNECTOR IS NOT CLUSTER-ENABLED

ONE OBJECT PUT PER CALL

JEDIS VS LETTUCE FRAMEWORK AND PIPELINING

SOME OPTIMIZATION IF ABLE TO BATCH REDIS
SHARDS BY PRE-COMPUTING

FLINK QUERYABLE STATE

REUSE OUR FEATURE REQUEST/RESPONSE
FRAMEWORK

SCALABLE, FAST

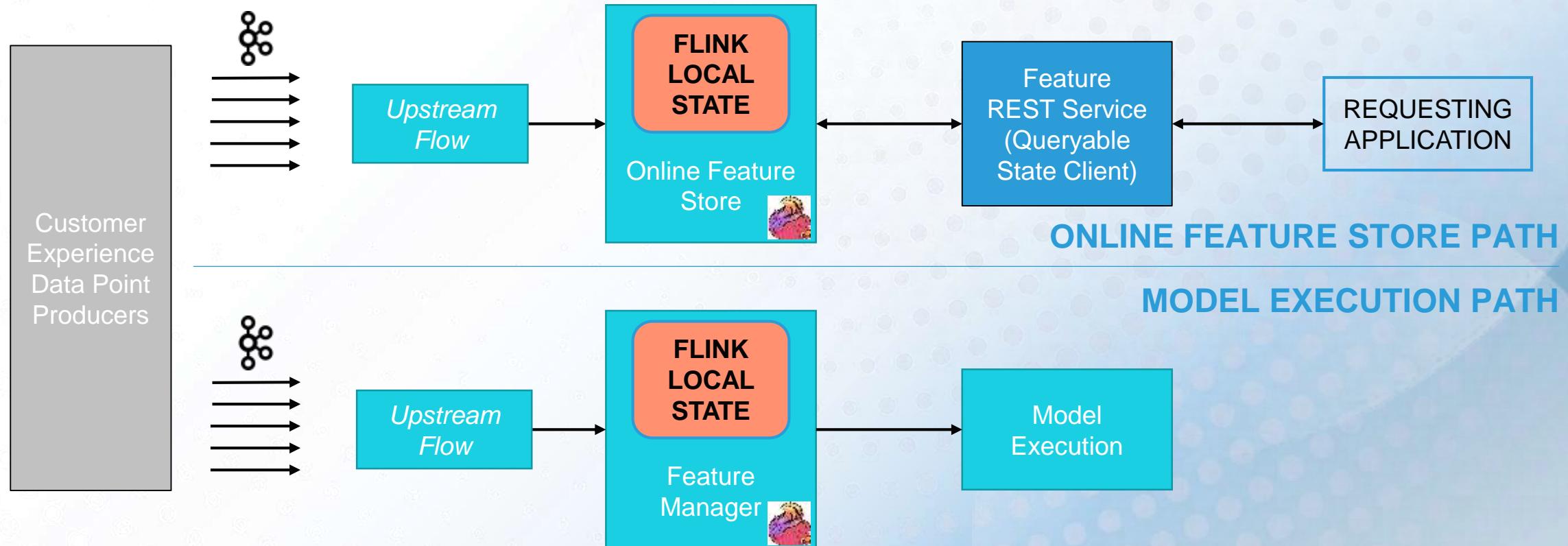
OUR READ LOAD IS LOW (10 / SECOND)

EXPENSIVE (RELATIVELY)

NOT A “PRODUCTION” CAPABILITY

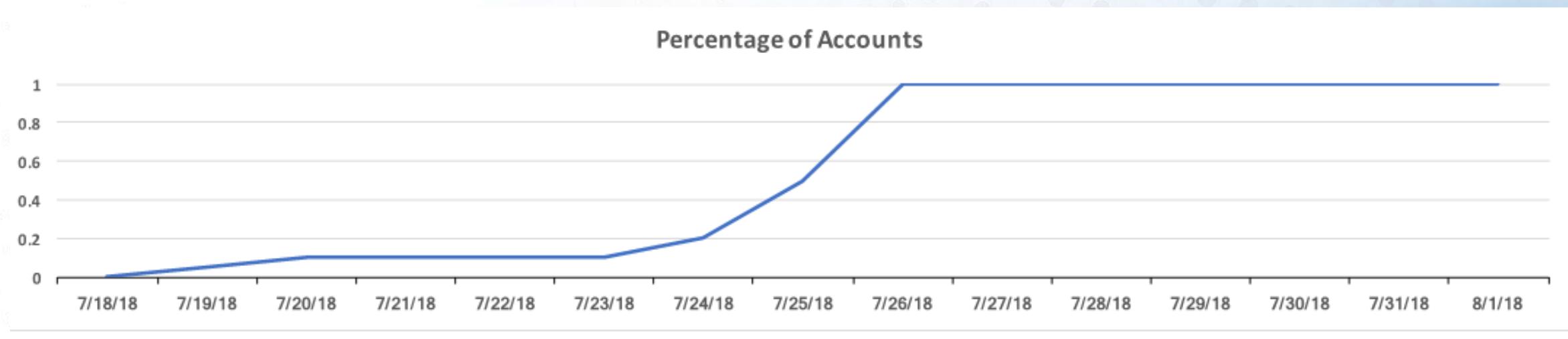
NOT A “DATABASE” TECHNOLOGY

QUERYABLE STATE - ONLINE FEATURE STORE

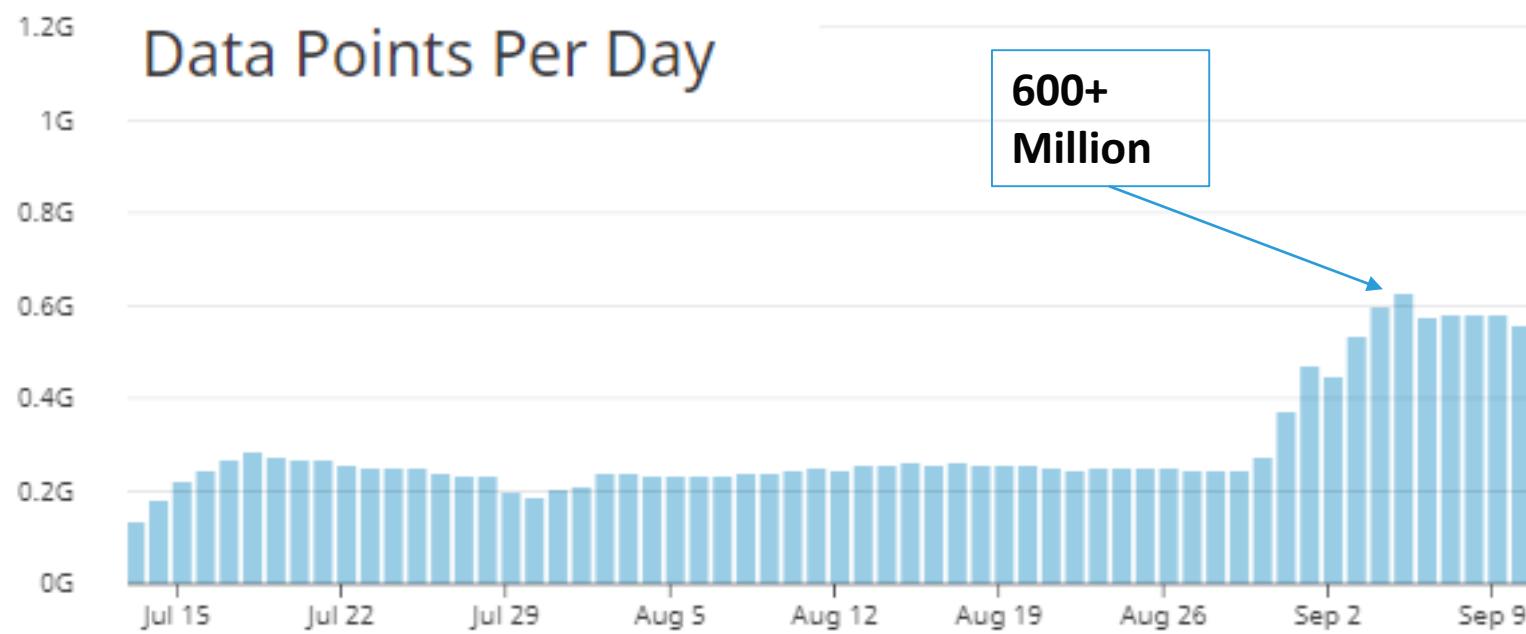


THE VOLUME PROBLEM

RAMPED UP PRODUCTION – JULY 2018

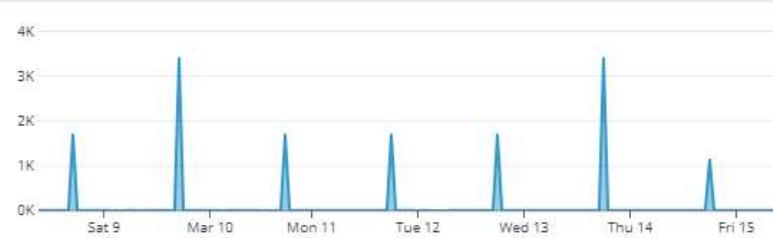


ADDED NEW DATA POINT TYPES



INCREASED DATA POINT FREQUENCY

ONCE EVERY 24 HOURS



INCREASED DATA POINT FREQUENCY

ONCE EVERY 24 HOURS

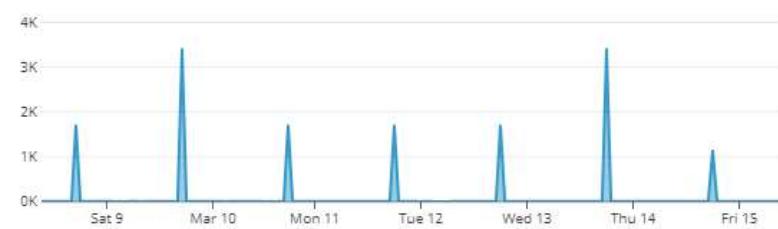


ONCE EVERY 4 HOURS



INCREASED DATA POINT FREQUENCY

ONCE EVERY 24 HOURS



ONCE EVERY 4 HOURS



ONCE EVERY 5 MINUTES ???



SYMPTOM: KAFKA LATENCY INCREASING

FLINK CLUSTER COULDN'T KEEP UP WITH KAFKA VOLUME



KAFKA OPTIMIZATION

INCREASE PARTITIONS

1 KAFKA PARTITION
READ BY
1 FLINK TASK THREAD

MATCH PARALLELISM TO PARTITIONS (OR EVEN MULTIPLE)

100 KAFKA PARTITIONS
50 SLOT PARALLELISM

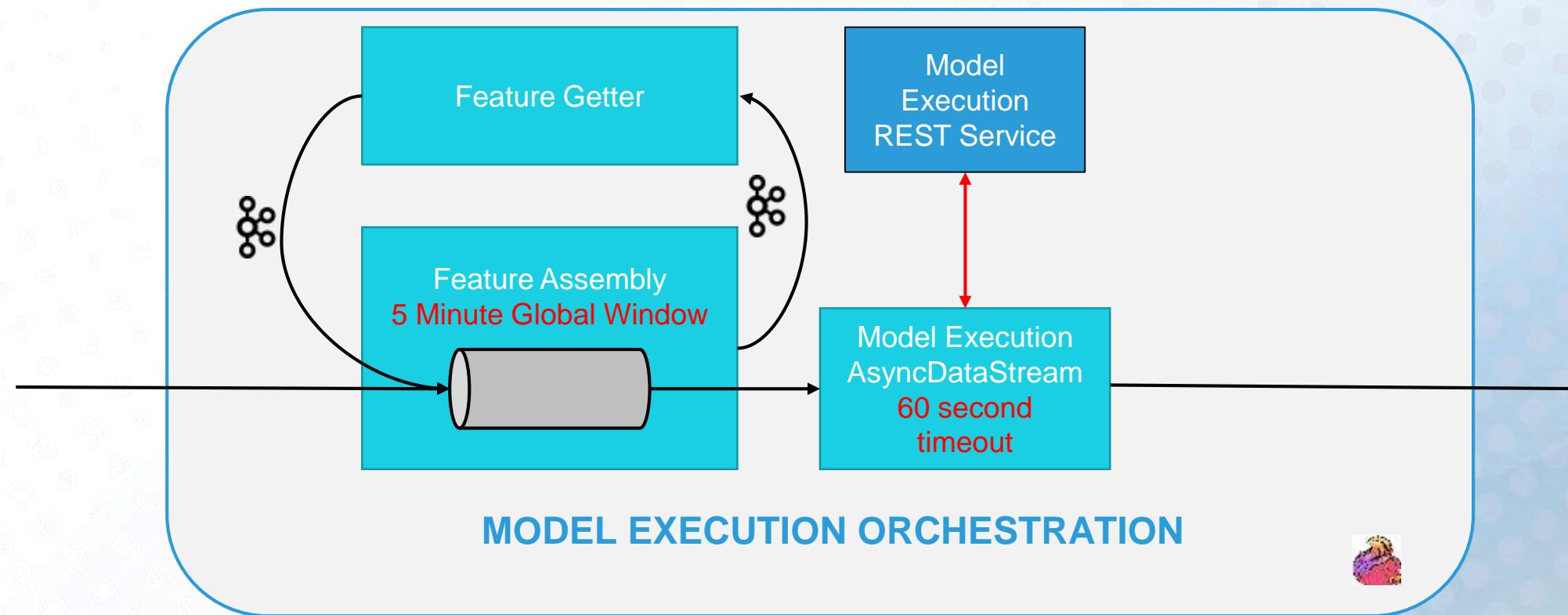
50 KAFKA PARTITIONS
50 SLOT PARALLELISM

ADJUST KAFKA CONFIGURATION

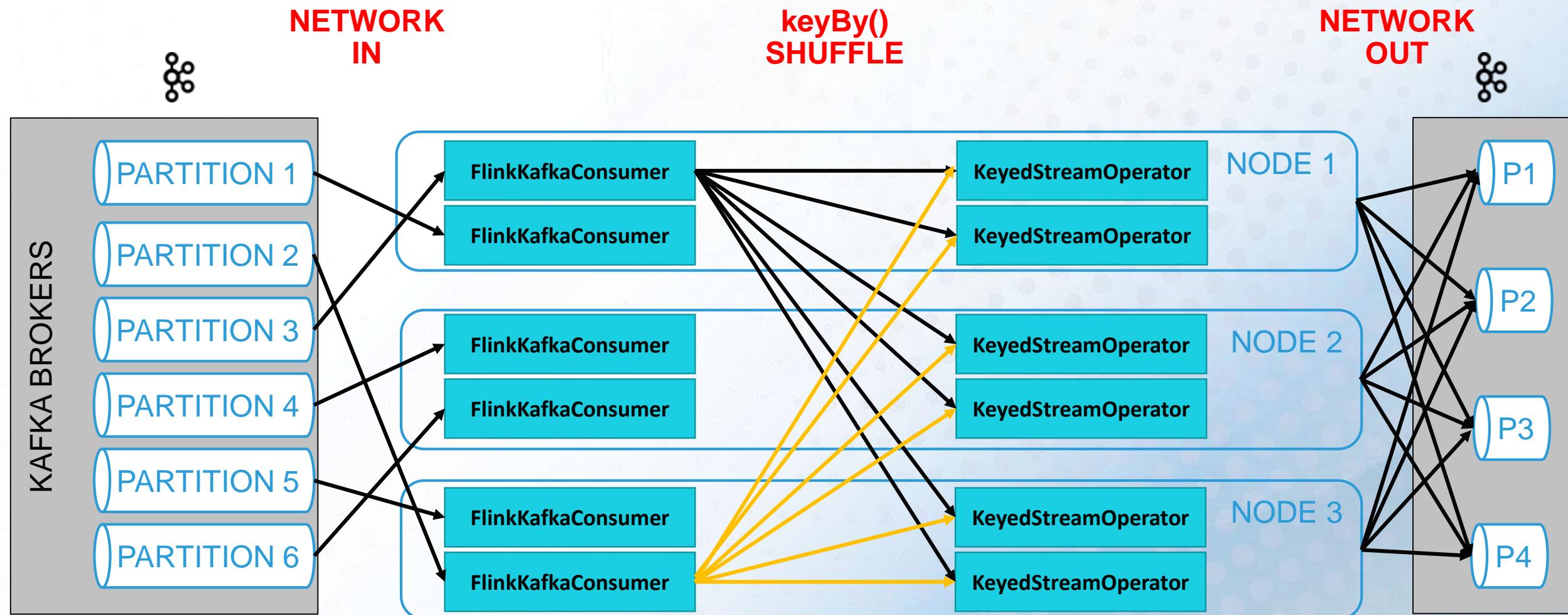
KAFKA CONSUMER SETTINGS
MAX.POLL.RECORDS = 500-10000
FETCH.MIN.BYTES = 1-102400
FETCH.MAX.WAIT.MS = 500-1000
KAFKA PRODUCER SETTINGS
BUFFER.MEMORY = 268435456
BATCH.SIZE = ~~131072~~-524288
LINGER.MS = 0-50
REQUEST.TIMEOUT.MS = 90000-600000

MODEL EXECUTION BACKUP PROBLEM

GENERAL SLOWNESS BACKS UP MODEL EXECUTION PIPELINE



CLUSTER NETWORK I/O TRAFFIC



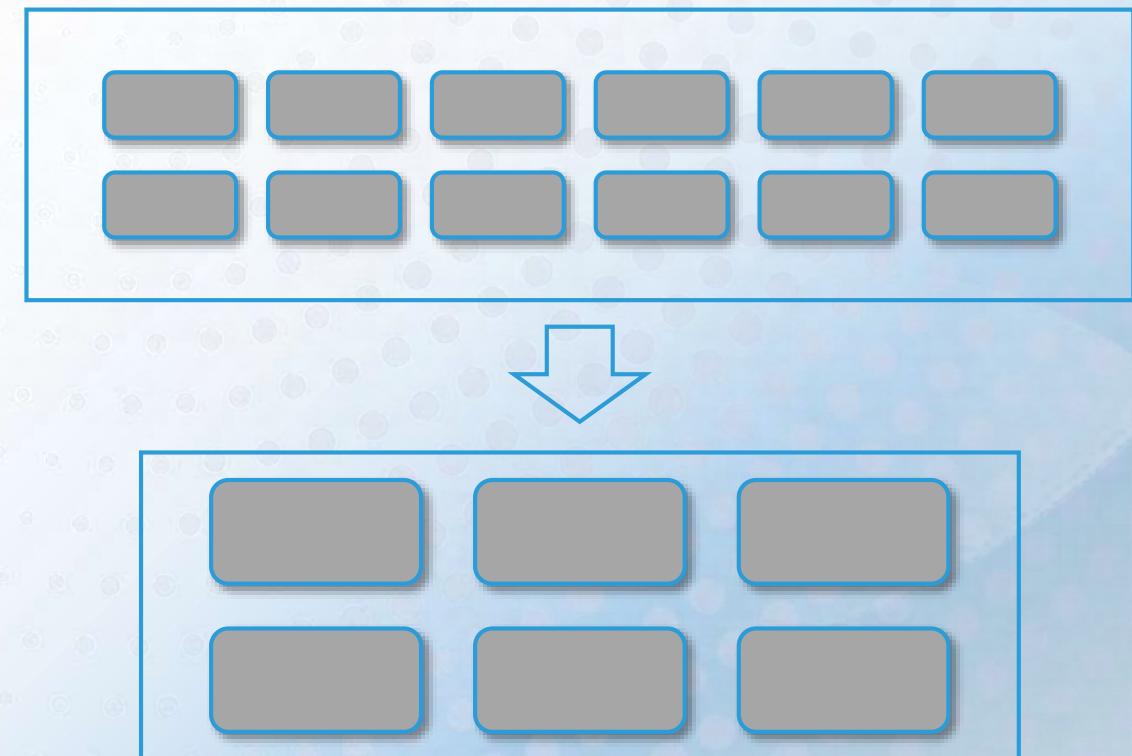
CLUSTER NETWORK I/O VOLUME

**(10,000 MESSAGES / SECOND) * (1KB / MESSAGE) =
10 MB / SECOND**

100 MBITS/SEC NETWORK

**SOLUTION – LARGER NODE TYPES WITH MORE
VCPU AND HIGHER PARALLELISM... FEWER
NODES IN CLUSTER**

SWEET SPOT SEEMED TO BE ~20 NODES



GOOD READ: [HTTPS://WWW.VERVERICA.COM/BLOG/HOW-TO-SIZE-YOUR-APACHE-FLINK-CLUSTER-GENERAL-GUIDELINES](https://www.ververica.com/blog/how-to-size-your-apache-flink-cluster-general-guidelines)

THE CUSTOMER STATE PROBLEM

KEEPING TRACK OF DATA POINT STATE

25+ MILLION HIGH SPEED INTERNET CUSTOMERS

10+ DATA POINT TYPES

TWO FLINK STATES TO MANAGE:

RED / GREEN “TRIGGER”

MAPSTATE () / KEYBY (ENTITY ID)

[ENTITY ID] (DATA POINT TYPE, STATE)

25 MILLION CUSTOMERS

10+ DATA POINT TYPES

AVG 33 BYTES PER ENTITY ID

~1GB TOTAL

QUERYABLE STATE FEATURE STORE

UP TO 15KB (JSON) PER DATA POINT

$10\text{KB} * 25 \text{ MILLION} = 250 \text{ GB}$

$1\text{KB} * 25 \text{ MILLION} = 25 \text{ GB}$

APPROXIMATE TOTAL = 1 TB

THE CHECKPOINT PROBLEM

CHECKPOINT TUNING

FEATURE MANAGER – UP TO 1TB OF STATE

2.5 MINUTES WRITING A CHECKPOINT EVERY 10 MINUTES

TURNED OFF FOR AWHILE!

(RATIONALE: WOULD ONLY TAKE A FEW HOURS TO RECOVER ALL STATE)

INCREMENTAL CHECKPOINTING HELPED

LONG ALIGNMENT TIMES ON HIGH PARALLELISM

REDUCED THE SIZE OF STATE FOR 4 AND 24 HOUR WINDOWS USING FEATURE LOOKUP TABLE

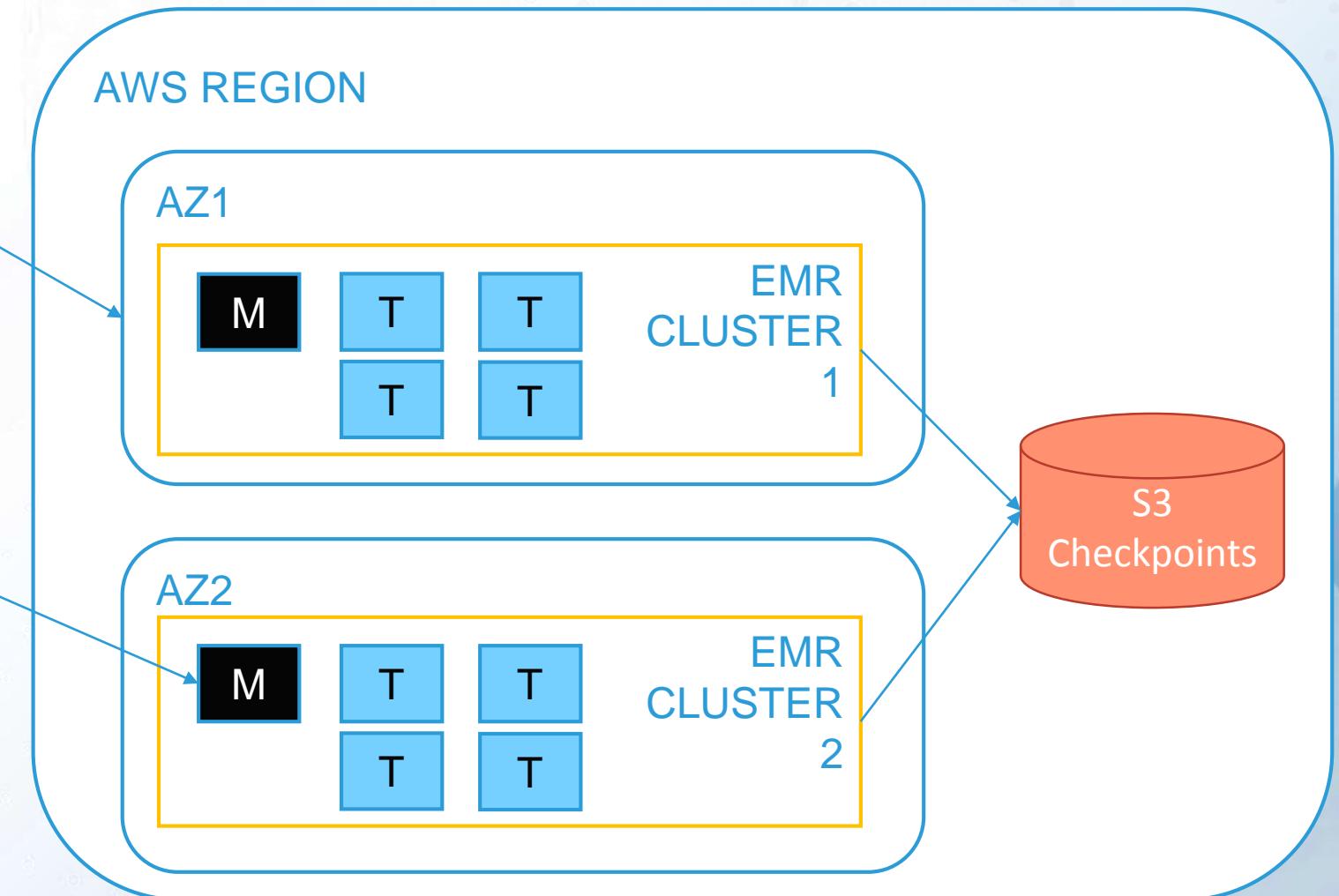
THE HA PROBLEM

AMAZON EMR AND HIGH AVAILABILITY

EMR FLINK CLUSTER

SINGLE AVAILABILITY ZONE (AZ)

SINGLE MASTER NODE
NOT HA



AMAZON AWS EMR

SESSIONS KEPT DYING

TMP DIR IS PERIODICALLY CLEARED
BREAKS ‘BLOB STORE’ ON LONG RUNNING JOBS

SOLUTION - CONFIGURE TO NOT BE IN /TMP

`jobmanager.web.tmpdir`
`blob.storage.directory`

EMR FLINK VERSION LAGS
APACHE RELEASE BY 1-2 MONTHS

TECHNOLOGY OPTIONS

FLINK ON KUBERNETES

AMAZON EKS (ELASTIC KUBERNETES SERVICE)

SELF-MANAGED KUBERNETES ON EC2

DATA ARTISANS-VERVERICA PLATFORM

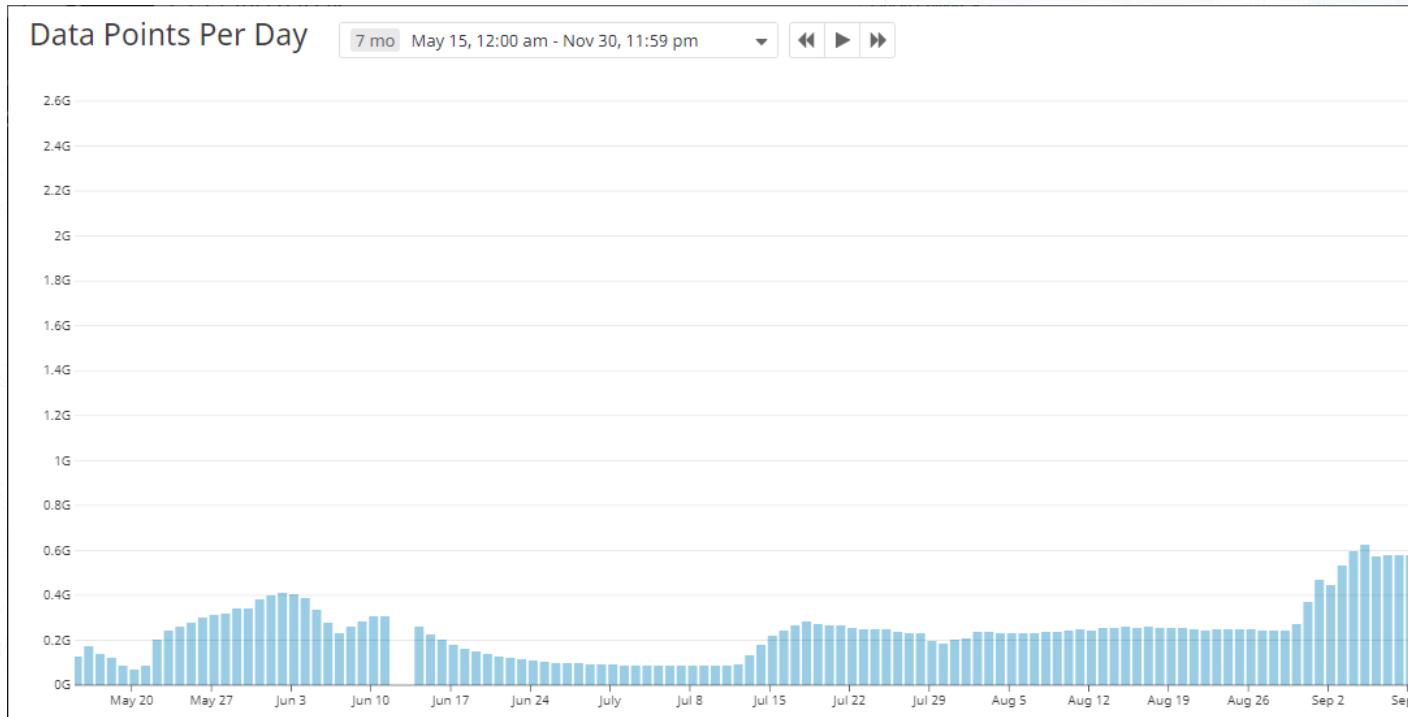
Bonus:

Reduced Cost / Overhead by having fewer Master Nodes

Less overhead for smaller (low parallelism) flows

THE REALLY HIGH VOLUME AND FEATURE STORE PROBLEM #2

DATA POINT VOLUME MAY–NOVEMBER 2018



DATA POINT VOLUME MAY–NOVEMBER 2018



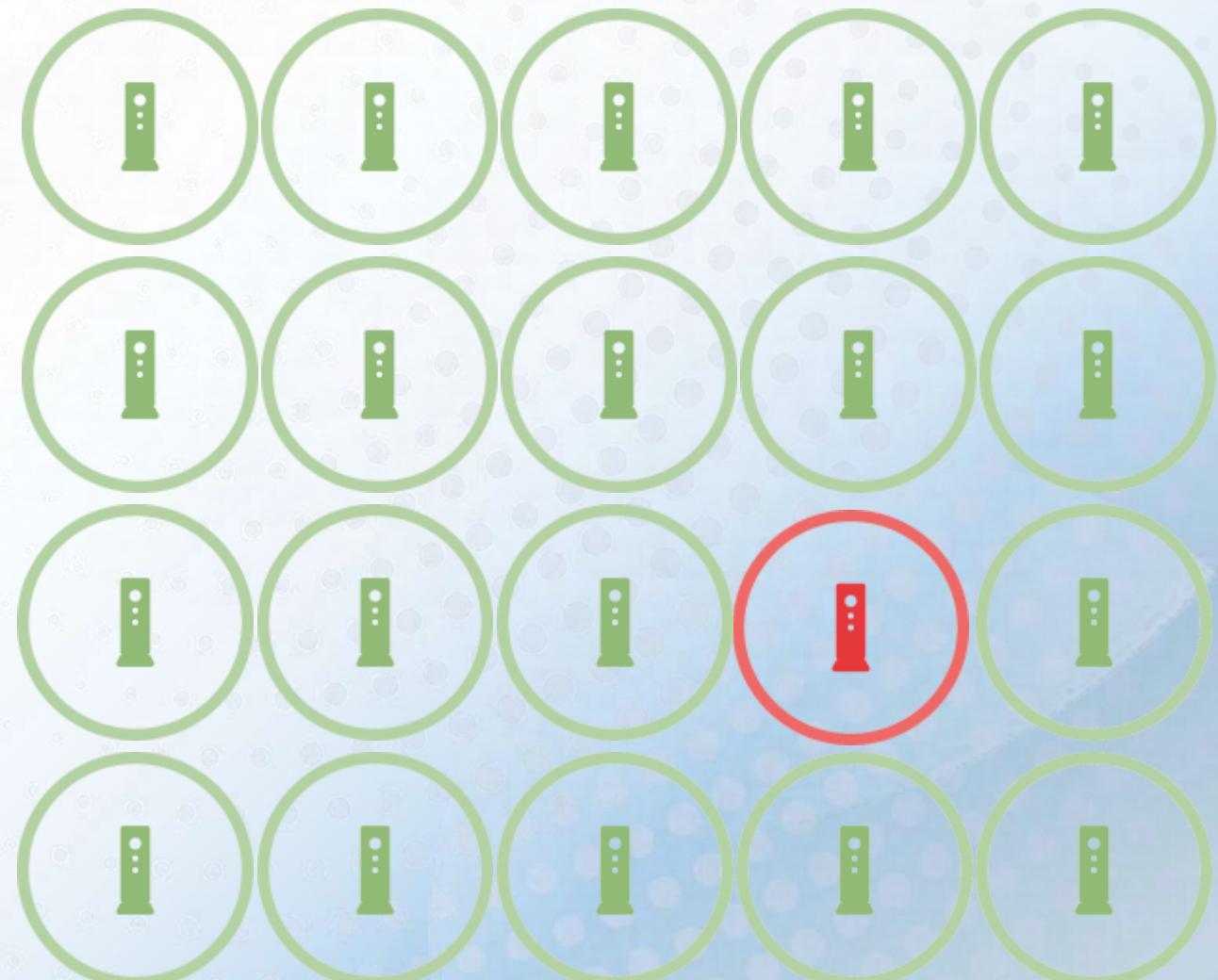
DATA POINT VOLUME MAY–NOVEMBER 2018



LET'S ADD ANOTHER 3 BILLION!

15 MILLION DEVICES

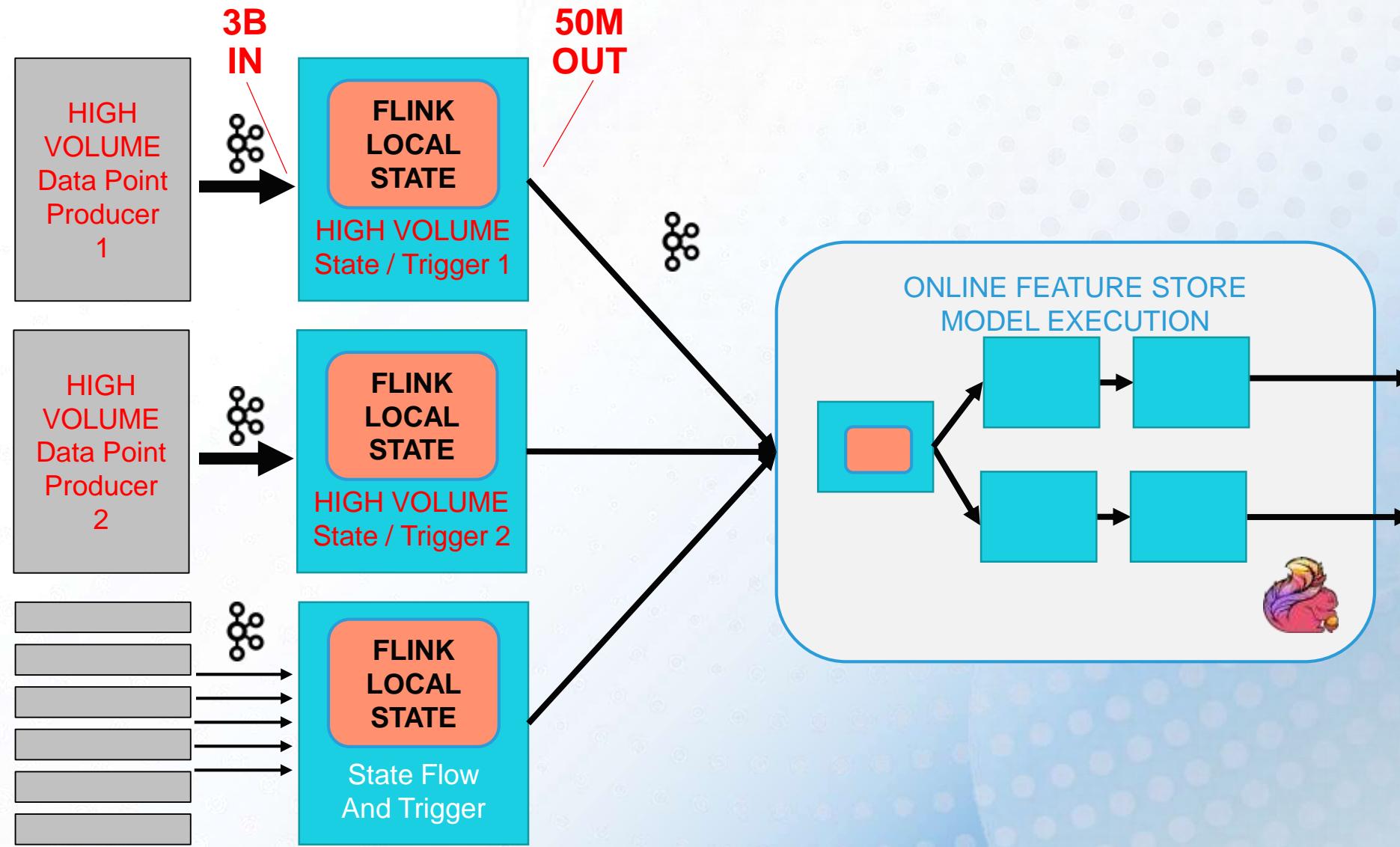
3 BILLION DATA POINTS



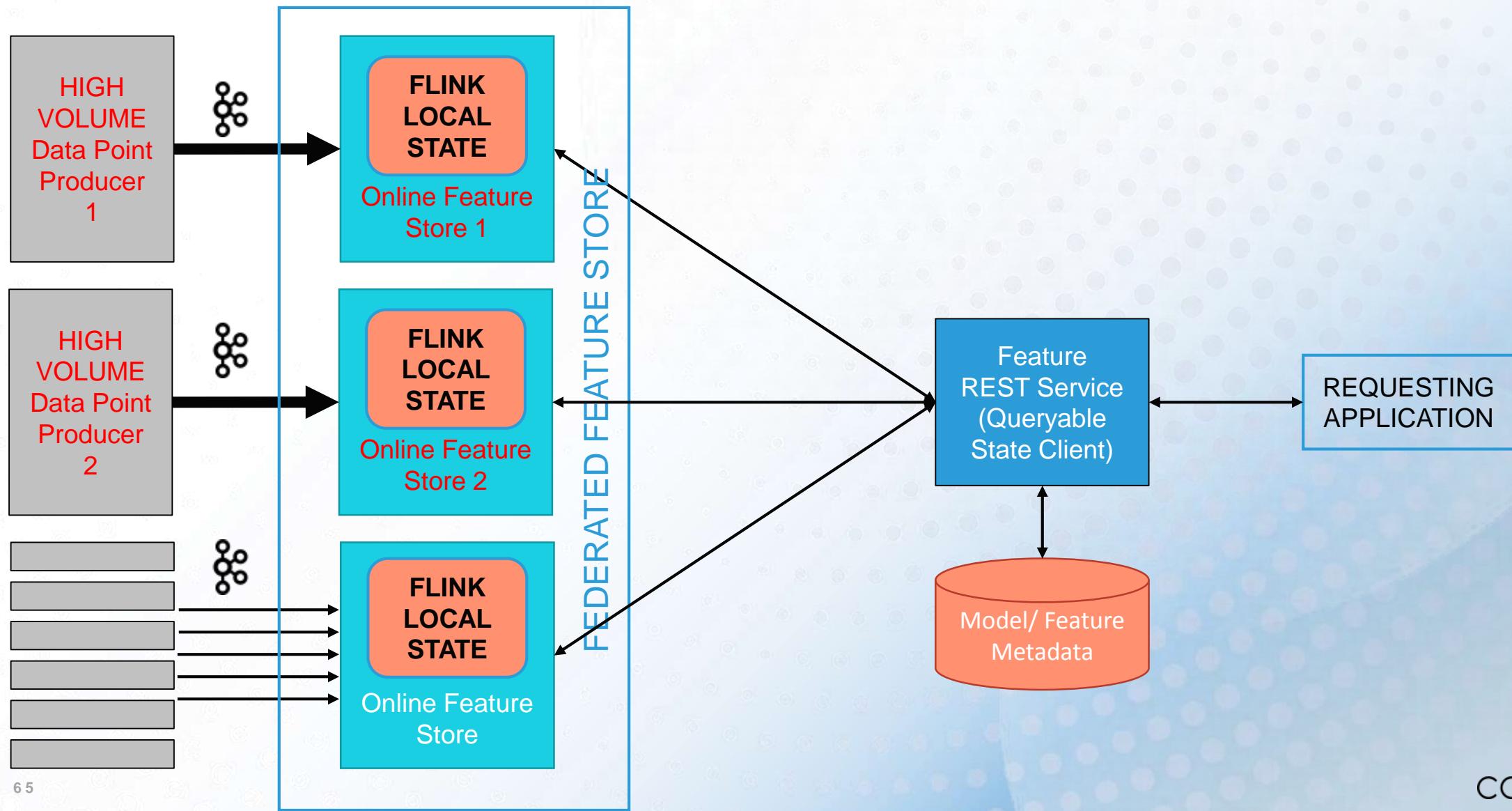
SOLUTION APPROACH

- 1 ISOLATE TRIGGER FOR HIGH-VOLUME DATA POINT TYPES
- 2 ISOLATE ONLINE FEATURE STORE FOR HIGH- VOLUME DATA POINT TYPES
- 3 ISOLATE DEDICATED FEATURE STORAGE FLOW TO HIGH-SPEED FEATURE STORE
- 4 SEPARATE HISTORICAL FEATURE STORAGE (S3 WRITER) FROM REAL TIME MODEL EXECUTION FLOWS

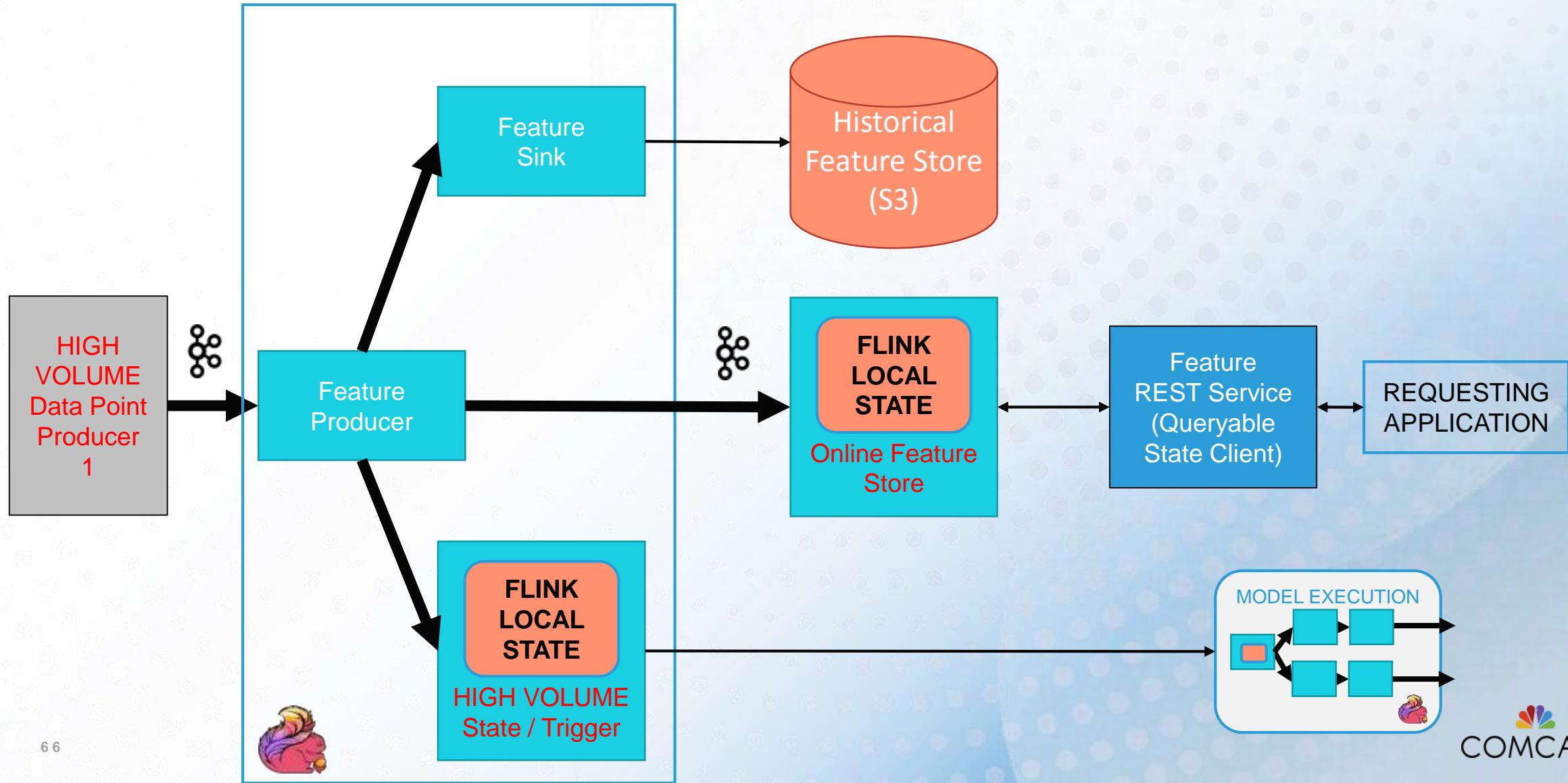
1: SEPARATE HIGH VOLUME TRIGGER FLOWS



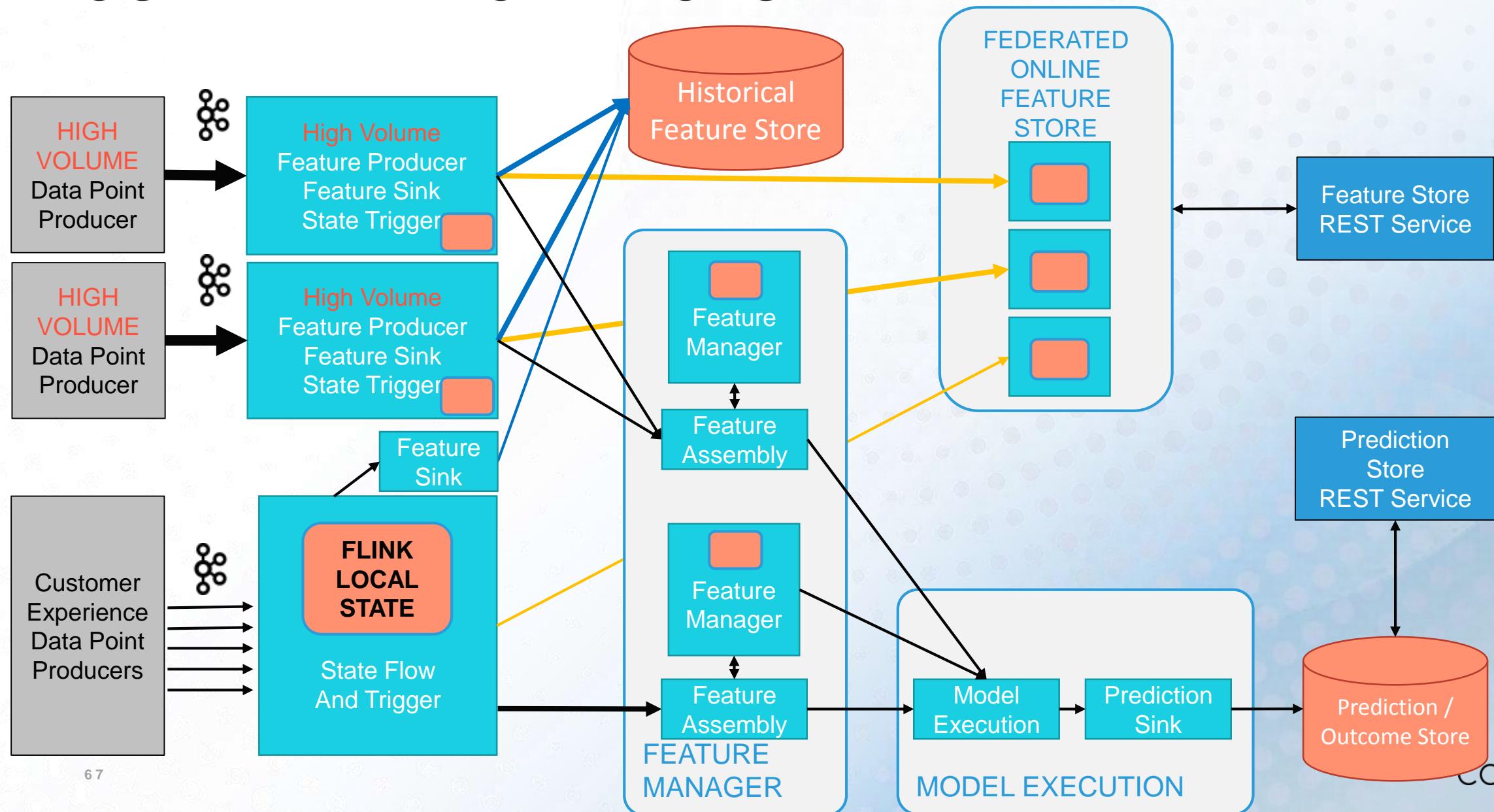
2: FEDERATED ONLINE FEATURE STORE



3+4: HISTORICAL FEATURE STORE



CURRENT ARCHITECTURE



WHERE ARE WE TODAY

INDICATORS

725+
MILLION
DATA
POINTS
PER DAY

HIGH-VOLUME INDICATORS

6 BILLION
PER DAY

TOTAL

~7 BILLION
DATA POINTS
PER DAY

WHERE ARE WE TODAY – FLINK CLUSTERS

CLUSTERS

14

INSTANCES

150

VCPU

1100

RAM

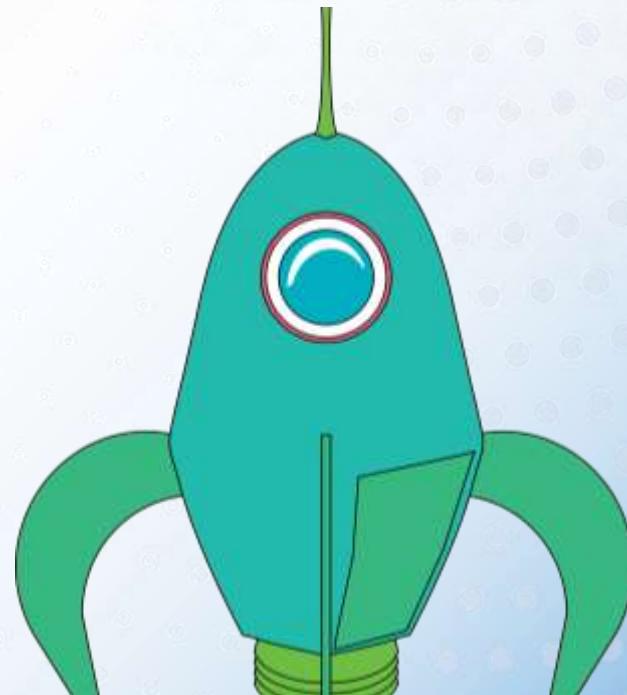
5.8 TB

FUTURE WORK

EXPAND
CUSTOMER EXPERIENCE
INDICATORS TO OTHER
COMCAST PRODUCTS

IMPROVED ML MODELS
BASED ON RESULTS AND
CONTINUOUS
RETRAINING

MODULAR
MODEL EXECUTION
VIA KUBEFLOW
AND GRPC CALLS



WE'RE HIRING!



PHILADELPHIA
WASHINGTON, D.C.
MOUNTAIN VIEW
DENVER

COMCAST



THANK YOU!





Scaling Warehouse with Flink, Parquet & Kubernetes

Aditi Verma &
Ramesh Shanmugam

Aditi Verma

Sr Software Engineer



@aditiverma89 <averma@branch.io>

Ramesh Shanmugam

Sr Data Engineer



@rameshs01 <rshanmugam@branch.io>

Agenda

- Background
- Moving data with Flink @ Branch
- Scale & Performance
- Flink on Kubernetes
- Auto Scaling & Failure Recovery



12B requests per day (+70% y/y)

3B user sessions per day

50 TB of data per day

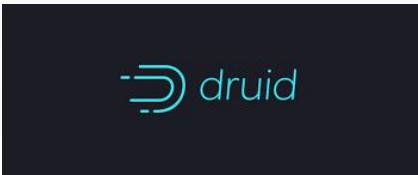
200K events per second

60+ Flink pipelines

5+ Kubernetes cluster



branch



kubernetes



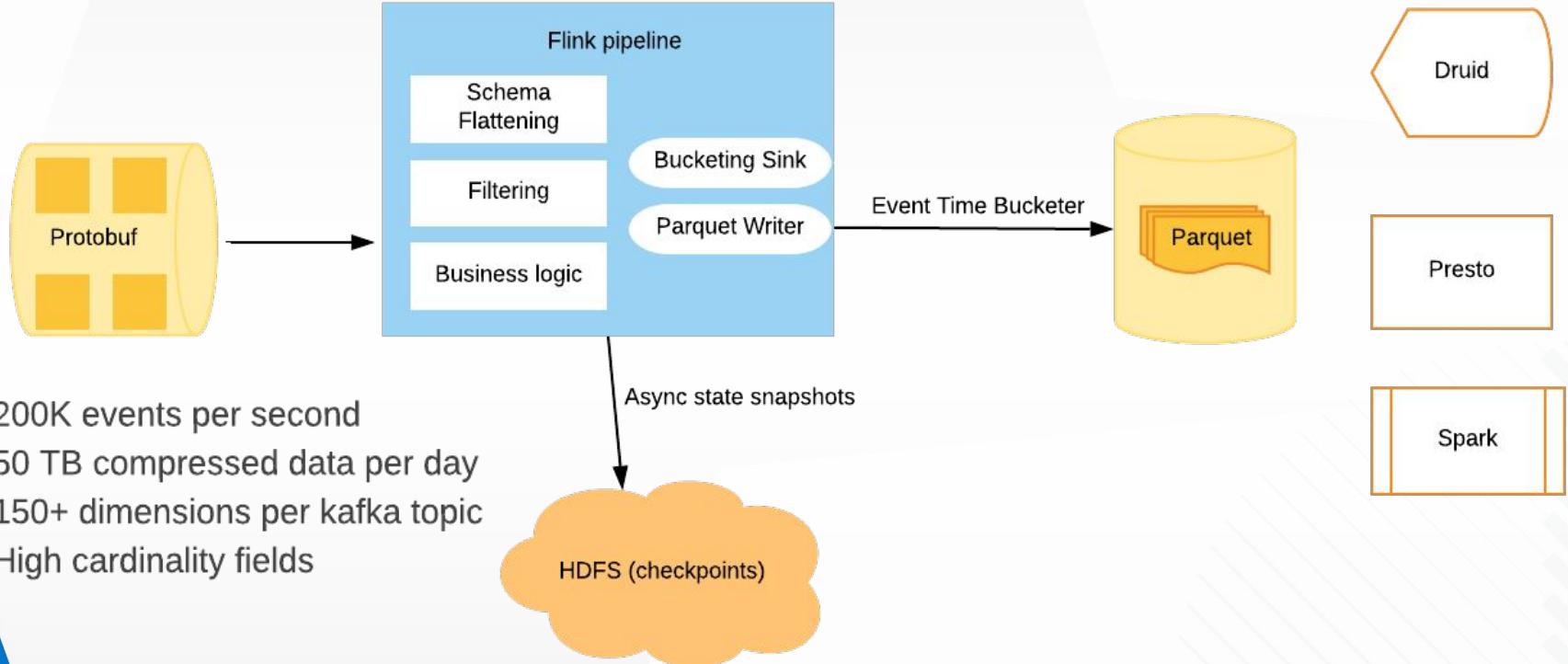
Moving data with Flink @ Branch



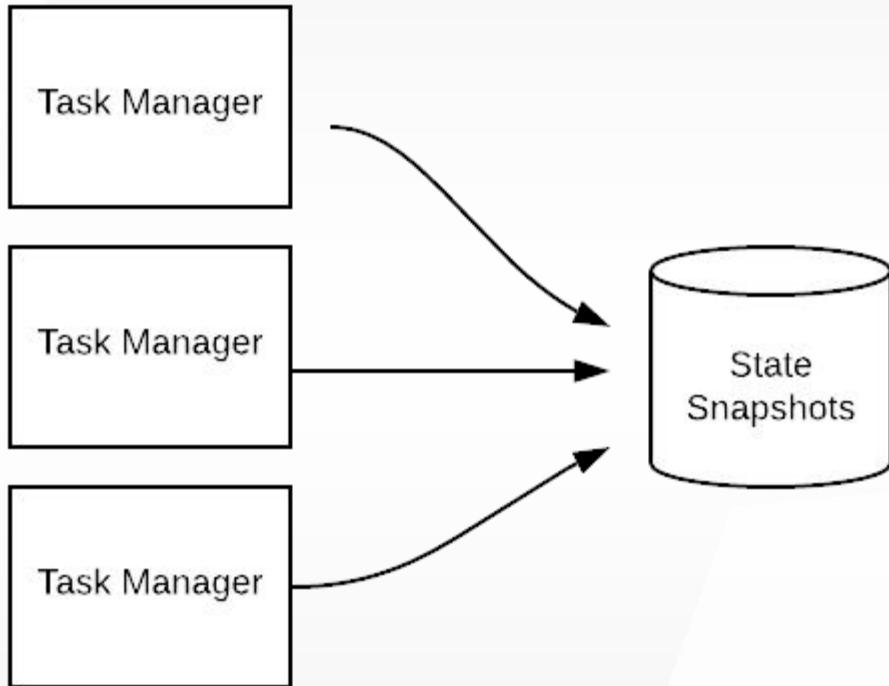
*"Life is 10% what happens to you and 90% how
you react to it."*
— **Charles R. Swindoll**

Receive information
Process it
React to it
FAST!!

Flink @ Branch



State Backend



- Relatively small state backend
- File system backed state



Parquet

- Higher compression
- Read heavy data set: ingested to Druid and Presto (3M+ queries/day)
- Avro data format
- Memory intensive writes



Writing parquet with Flink

Two approaches:

- 1) Close the file with checkpointing



Writing parquet with Flink

Two approaches:

- a) ~~Close the file with checkpointing~~
- b) Bucketing file sink
 - i) Configured with custom event-time bucketer, parquet writer and batch size
 - ii) Files are rolled out with a timeout of 10 min within a bucket

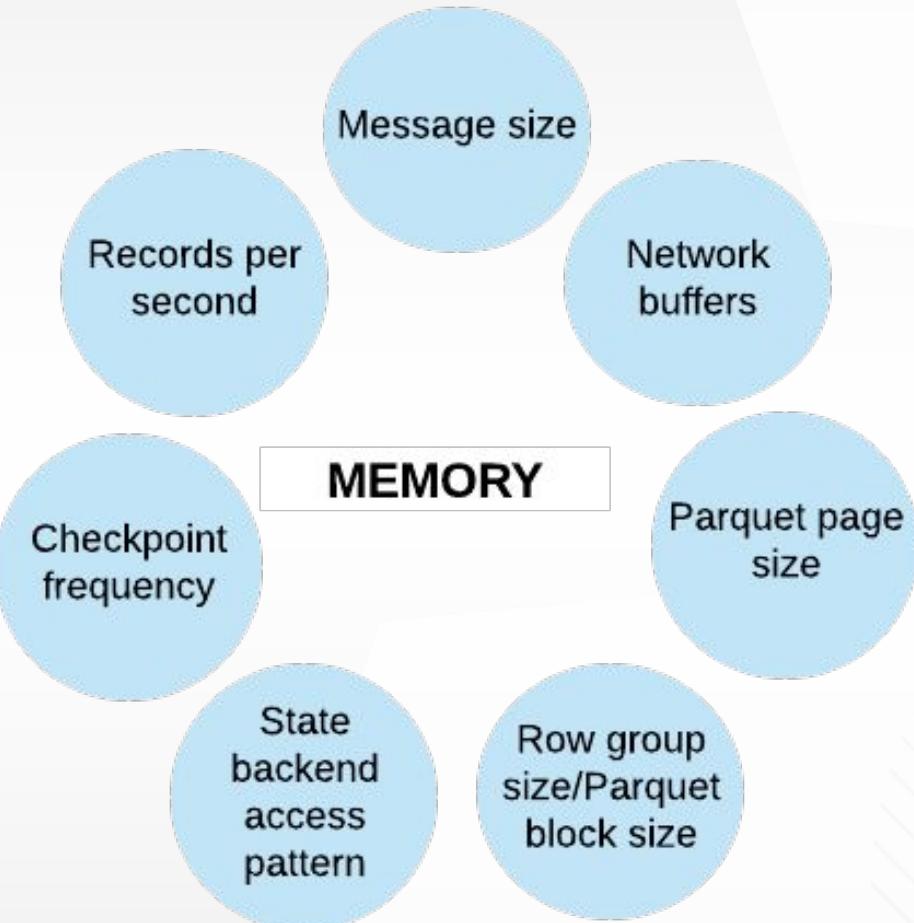


Performance and Scale

- 100% traffic increase each year
- Higher parallelism impacts application performance and state size
- Kafka partitions < Flink parallelism requires rebalance on the input stream
- Task manager timeouts



MEMORY



Analyzing memory usage

- ❖ Network Buffers
- ❖ Memory Segments
- ❖ User code
- ❖ Memory and GC stats
- ❖ JVM parameters



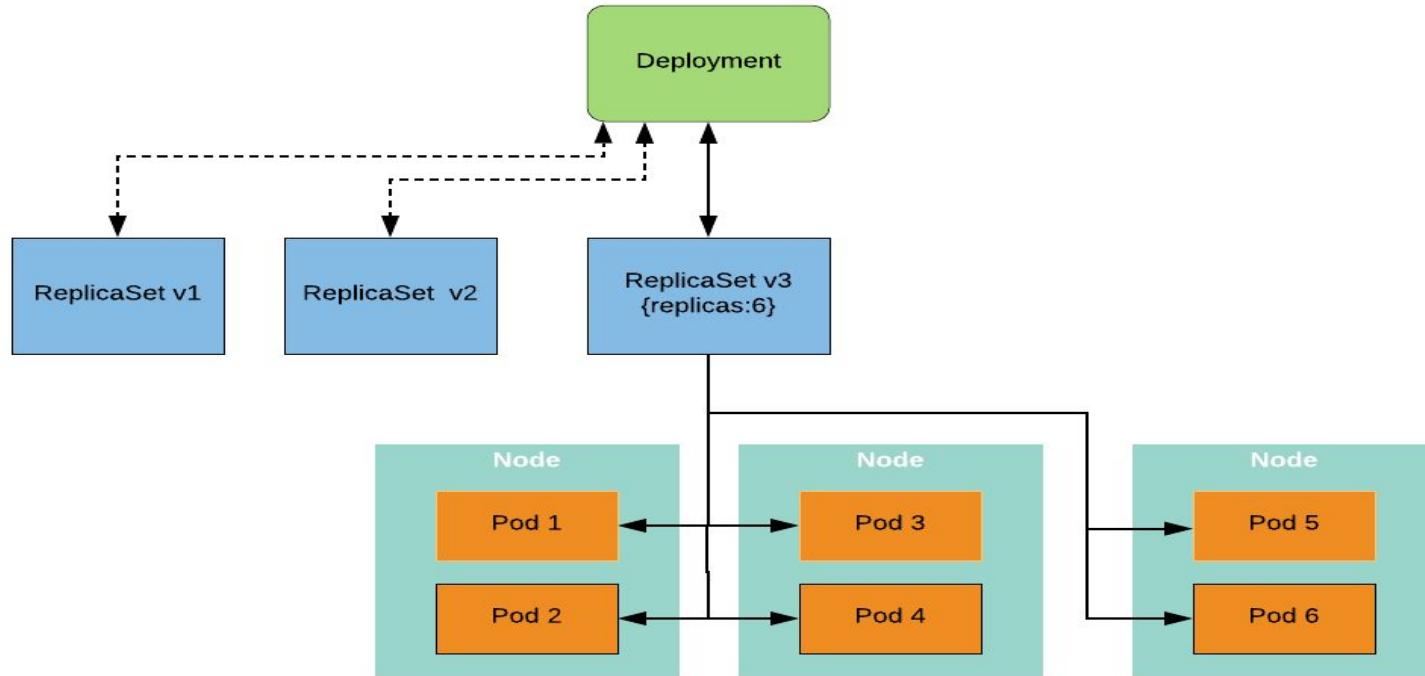
Containerizing Flink - Mesos

- Longer start-up time on Mesos
- Moved to containerizing Flink application on Kubernetes
- Kubernetes is resource oriented, declarative



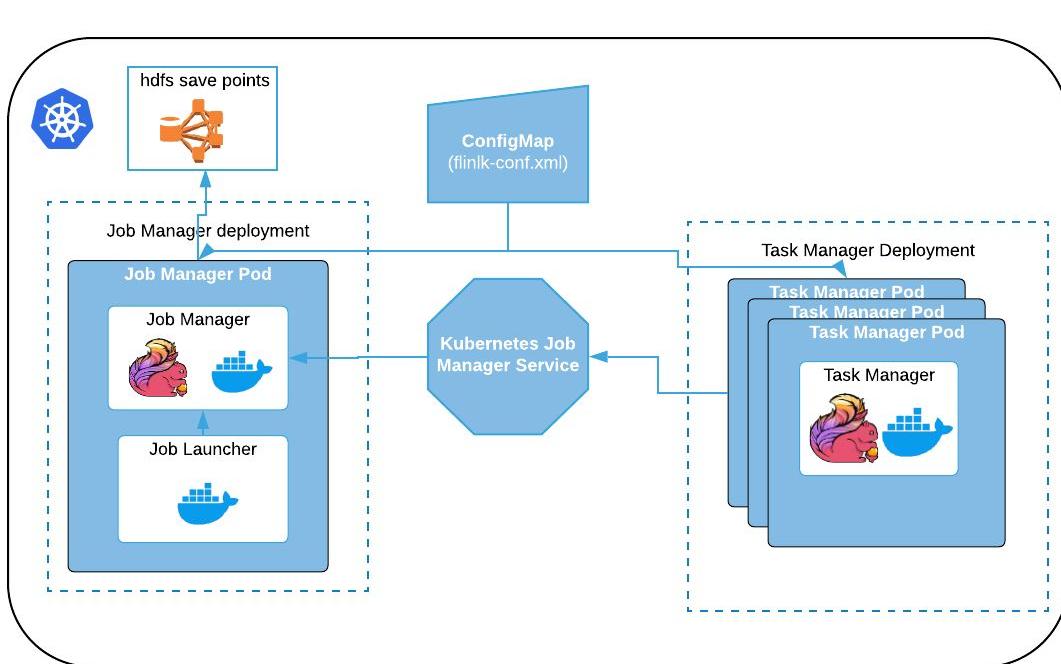


Kubernetes Terms





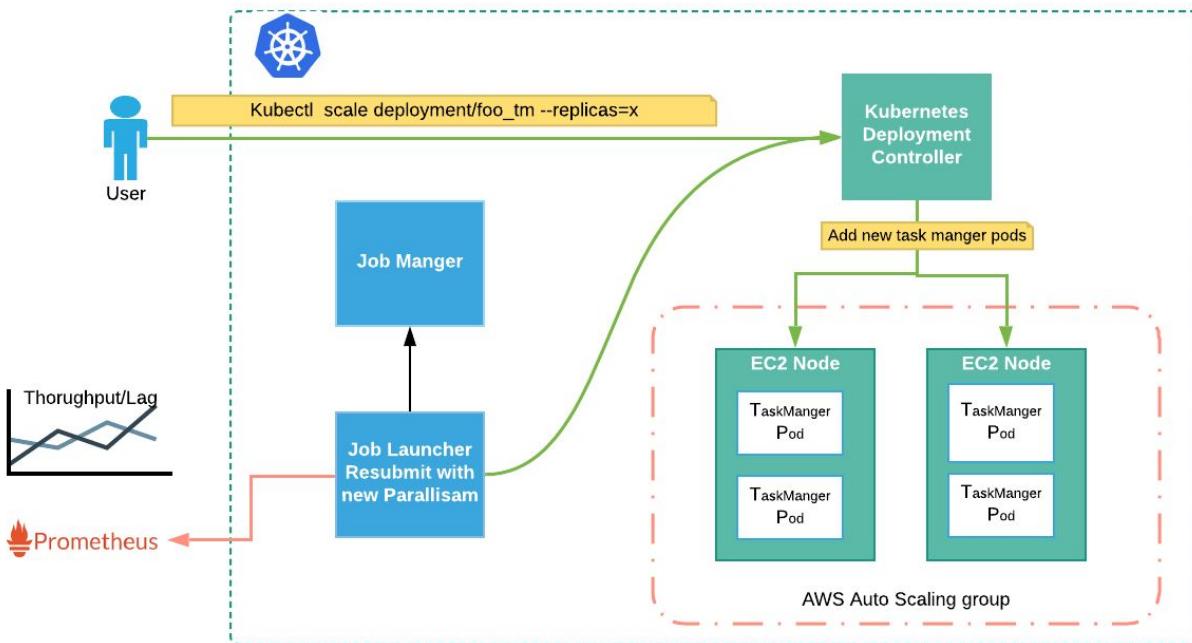
Flink on Kubernetes @ Branch



- Single job per cluster
- Docker image
 - flink image - Task manager + job manager
 - Job launcher - custom launcher + job jar
- Job launcher
 - Application jar
 - Uploads jar
- Config map - flink config.xml
 - jobmanager.rpc.address



Auto Scaling

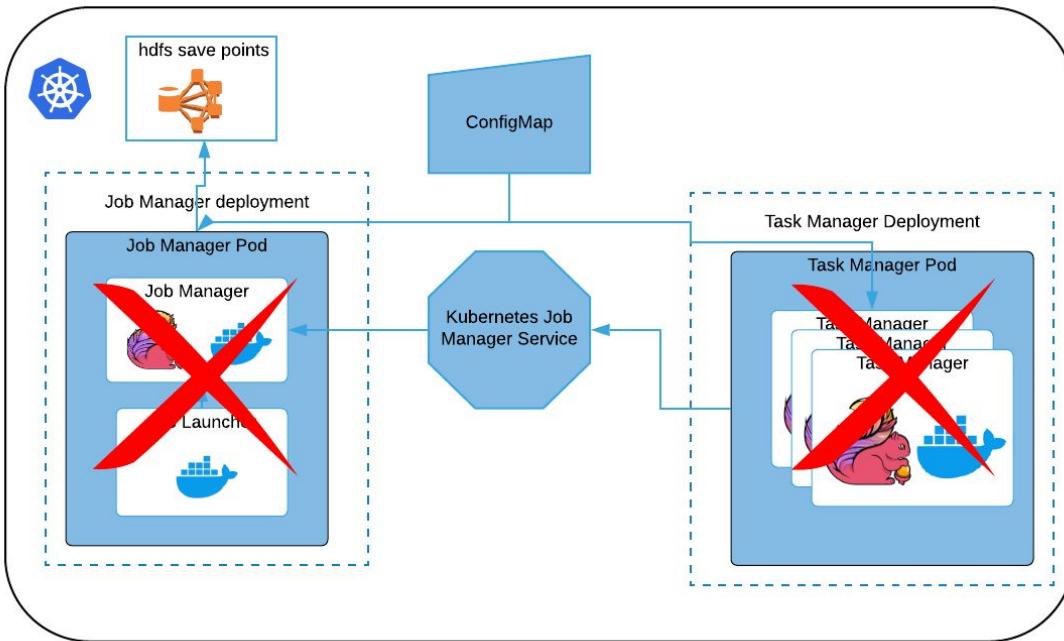


- When & How much scale
 - Auto - Job launcher Scale
- Kubernetes Deployment Controller
- Flink job with new parallelism

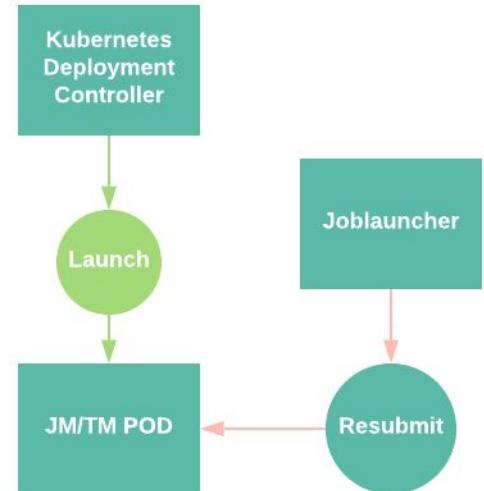
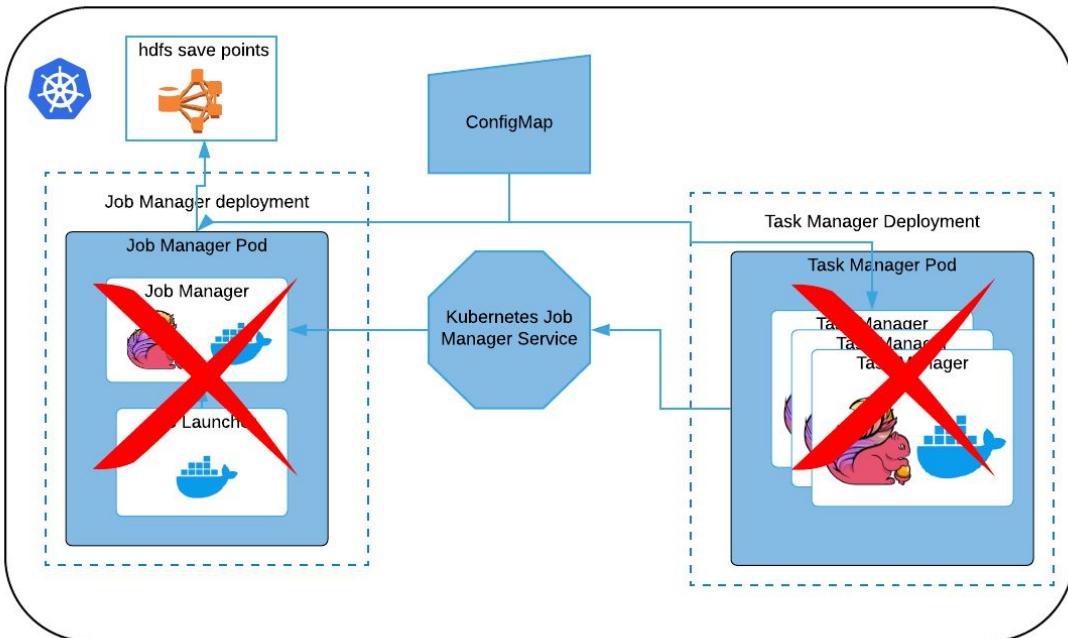


Failure Recovery

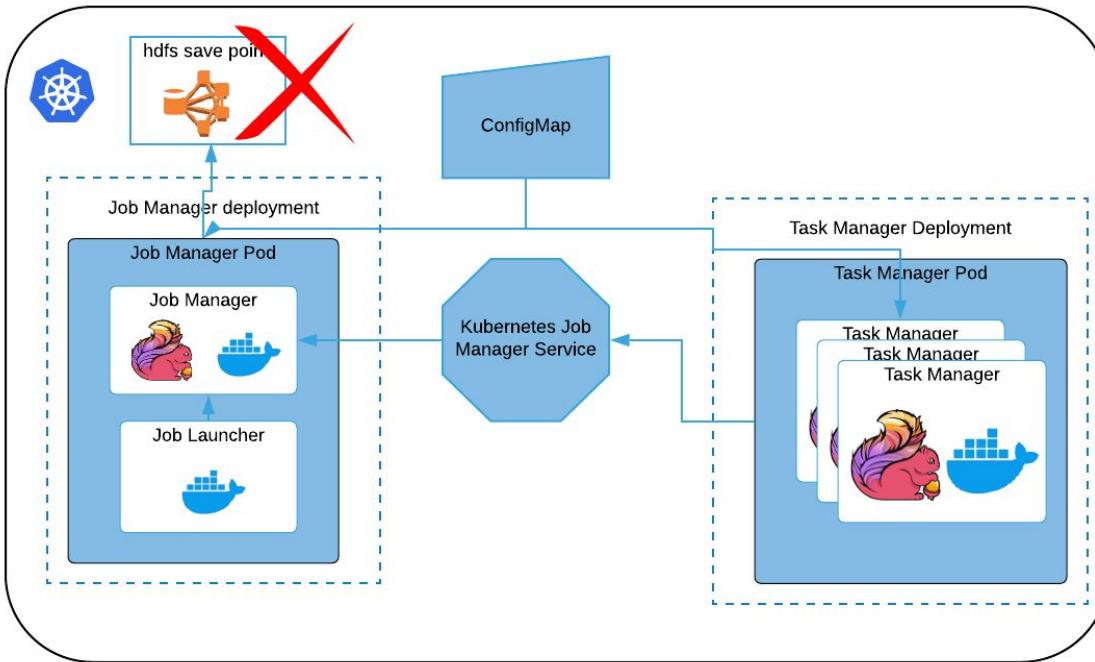
Job / Task Manager Goes Down?



Job / Task Manager Goes Down?



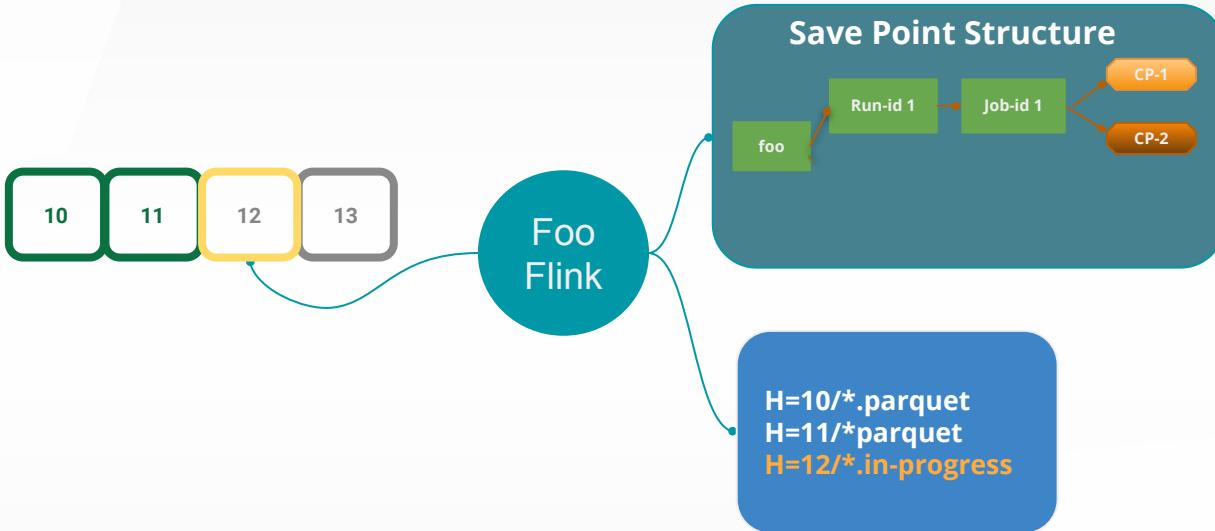
Savepoint Failure



- Reasons
 - Truncation
 - Schema mismatch
 - Hdfs outage

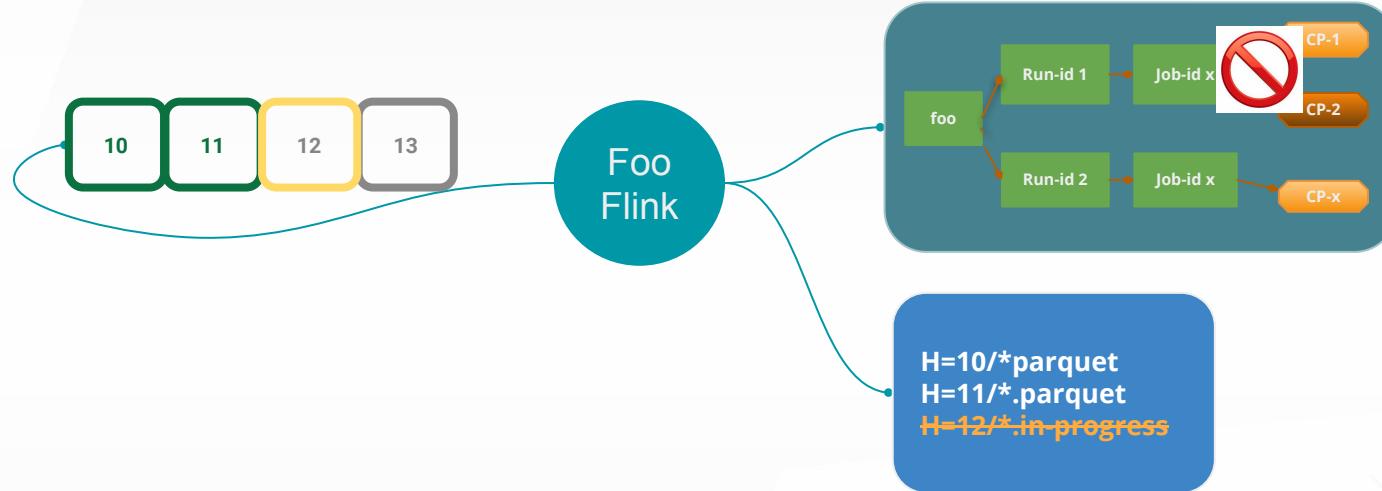


Savepoint Structure

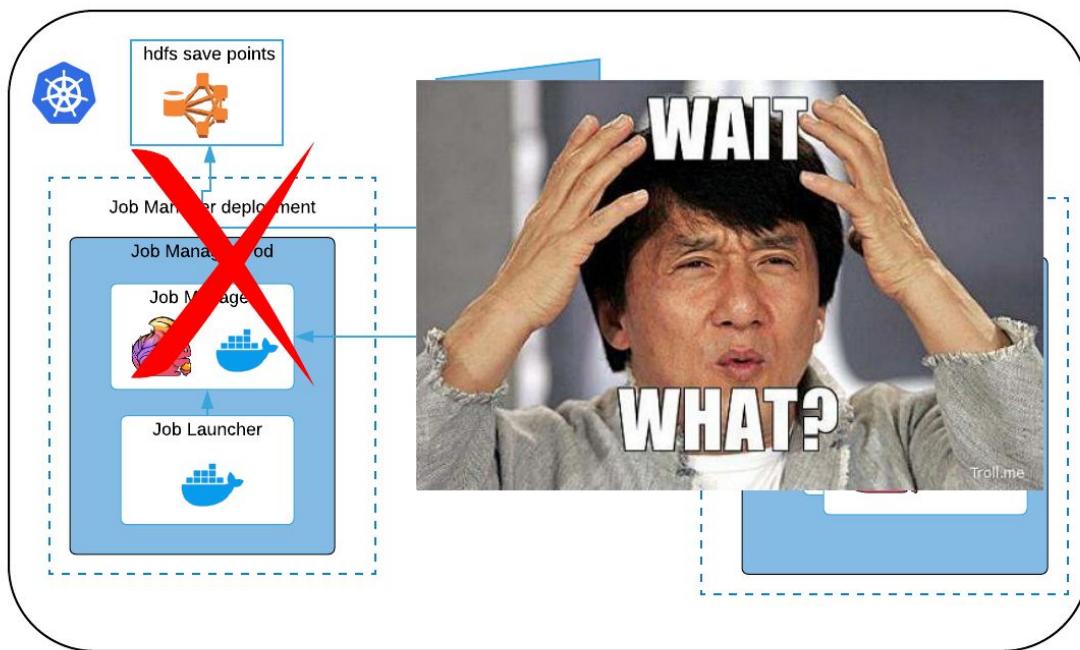


- job/run-id/flink-job-id/cp-x
- Run id - incremental number
- Job id - flink job name

Savepoint failure recovery



Auto Recovery does not work?



- Continuous monitoring and proper alerts
- start job from latest offset
- Have different backfill route



Next Steps....

- Parquet memory consumption (when too many buckets open)
 - Window + Rocks db => Parquet
 - Two stage process
 - row oriented streaming
 - batch to convert columnar



Q & A

Using Flink to inspect live data as it flows through a data pipeline

Matt Dailey | @matthew_dailey1

April 2, 2019

splunk>

Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2019 Splunk Inc. All rights reserved.

About Me

- Principal Software Engineer @ Splunk
- Using Apache Flink for about 2 years
- Before that, Hadoop ecosystem for 6 years



 @matthew_dailey1

splunk > Data Stream Processor DSPDEMO2

Add Data Manage Pipelines Merge Templates Theme: Enterprise

mdaley Test Pipeline v11

Validate Start Preview

```

graph LR
    A[Read from Splunk Pipelines] --> B[Log File Filter]
    B --> C[Write to Index]
    A --> D[Syslog Filter]
    D --> E[Extract Fields]
    E --> F[Write to Index]
  
```

View Configurations Preview Results

8 of 100 events.

filter

Table ▾ 10 Per Page ▾

#	timestamp	nanos	host	source	body
>	1554239796660	432272	host	syslog	Jul 20 17:44:56 192.90.223.98 %ASA-6-30507: Built dynamic TCP translation from MS-WIN10-00:97124/50866 to outside:98.23.39.86/50866
>	1554239797383	225511	host	syslog	Jul 20 17:29:34 BUSDEV-006 Mar 05 2013 16:01:03 BUSDEV-006 : %ASA-6-30201: Built inbound TCP connection 1671358331 for outside:ff05.179.171.62160 (RegentTW@ACMETECH.CC)
>	1554239798101	765113	host	syslog	Jul 20 17:03:45 190.233.208.250 %ASA-6-30202: Built ICMP connection for ifaddr 254.179.6.252/35949 gaddr 190.233.208.250/0 laddr 10.149.94.44/0
>	1554239798898	908931	host	syslog	Jul 20 17:01:58 215.236.50123 %ASA-6-30203: Built outbound TCP connection 9 for outside:168.172.204.22 [215.236.60.123/22] to inside:10.149.105.161/53496 [215.236.60.123/53496]
>	1554239799600	700586	host	syslog	Jul 20 17:25:45 sps-sys-004 Apr 19 2013 11:24:32 %ASA-6-30203: Built inbound TCP connection 215721377 for outside:205.59.147.171/61284 [91.74.141.66:81284] to inside:10.149.105.161/53496 [215.236.60.123/53496]

splunk listen to your data

Authors of pipelines would like to know...

Why does my data look like *that* at the end of my pipeline?

What do we need to
build to answer this?

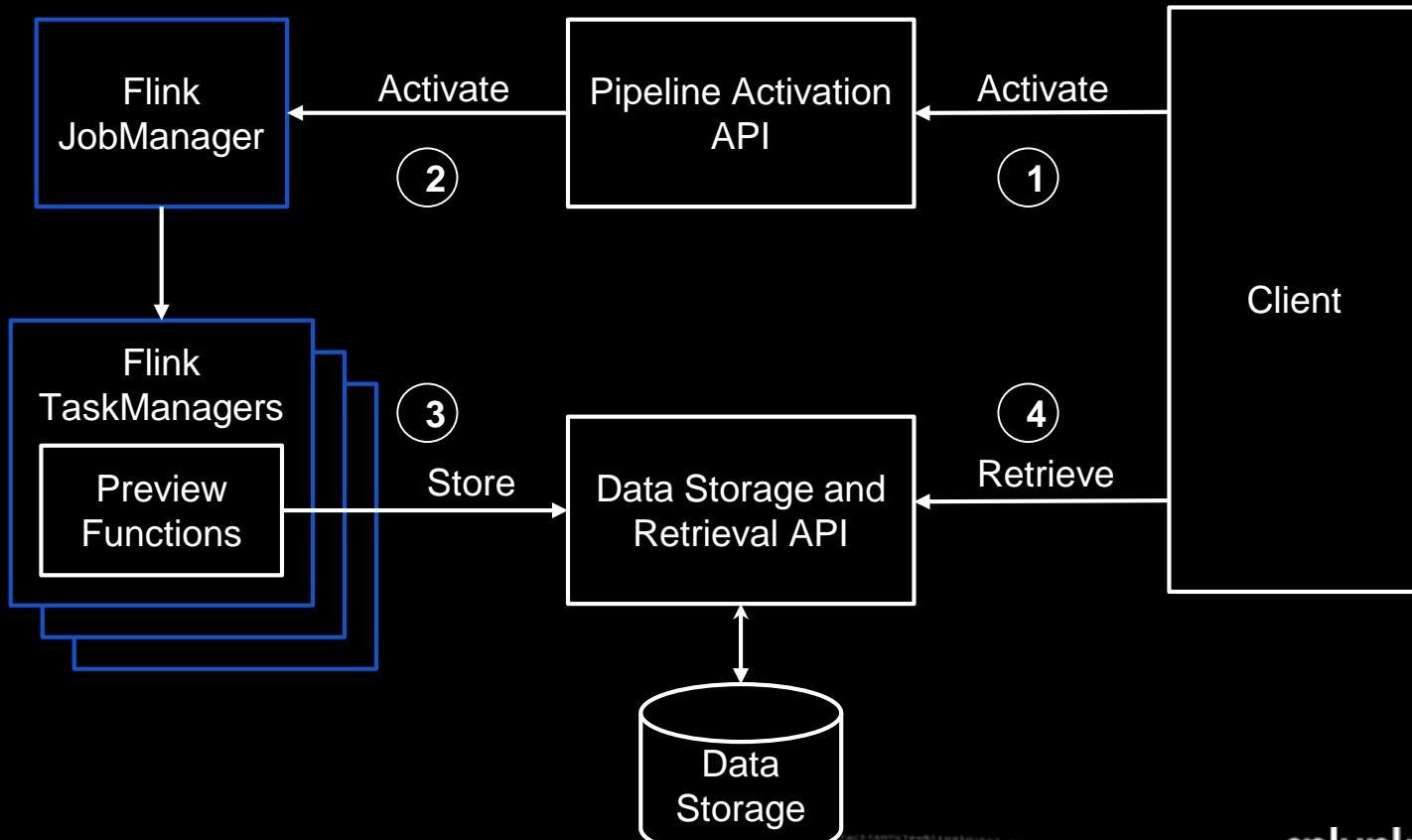
What we need

- A place to display the data
- A way to store the data for display
- A way for a pipeline to send the data to storage
- A way to test the pipelines during authoring
- A name for the feature

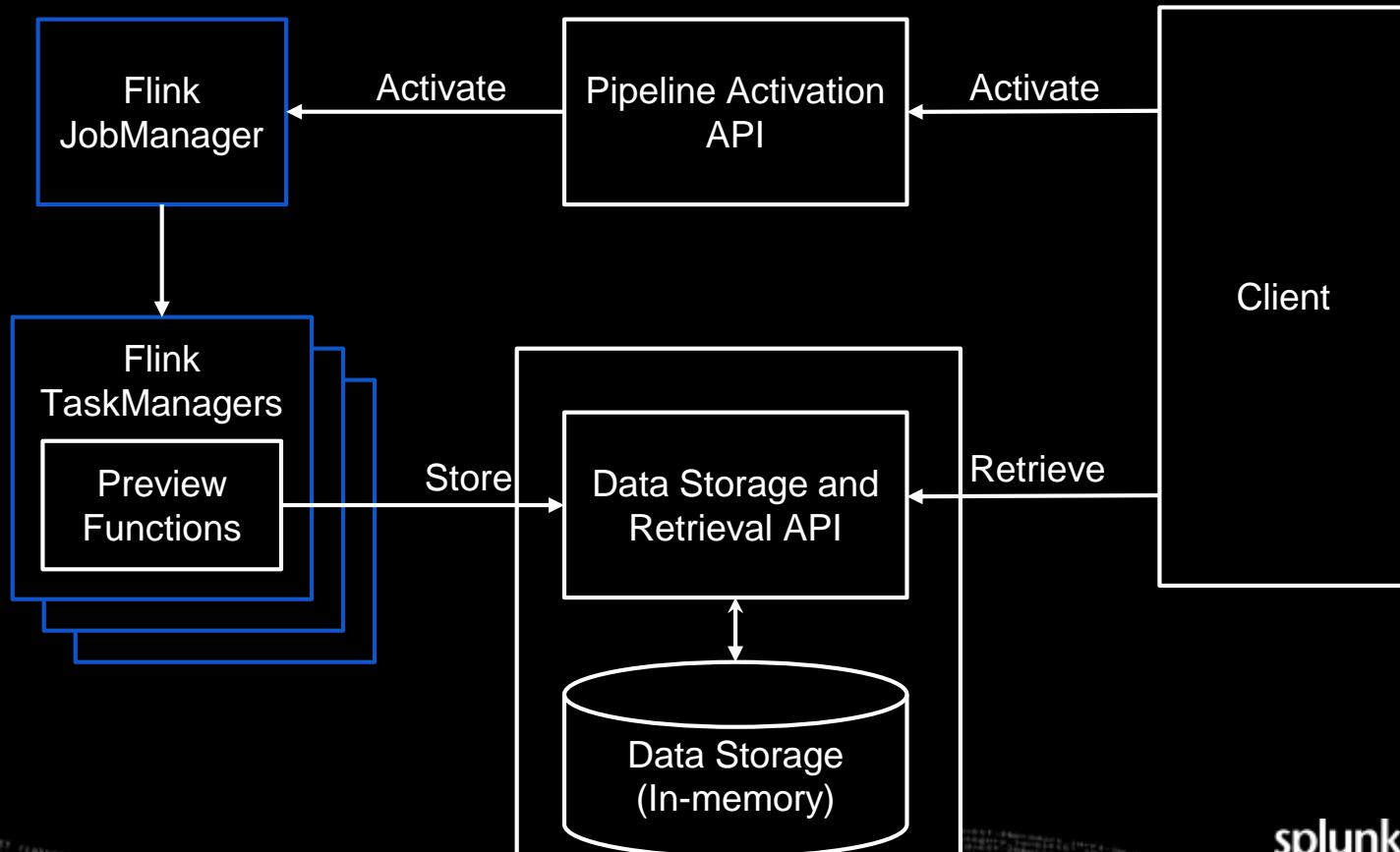
Design

- A place to display the data
 - API and User Interface
- A way to store the data for display
 - Ephemeral storage for ephemeral data
- A way for a pipeline to send the data to storage
 - Function in the pipeline sends data, limits data volume
- A way to test the pipelines during authoring
 - Run a separate pipeline for debugging
- A name for the feature
 - Preview

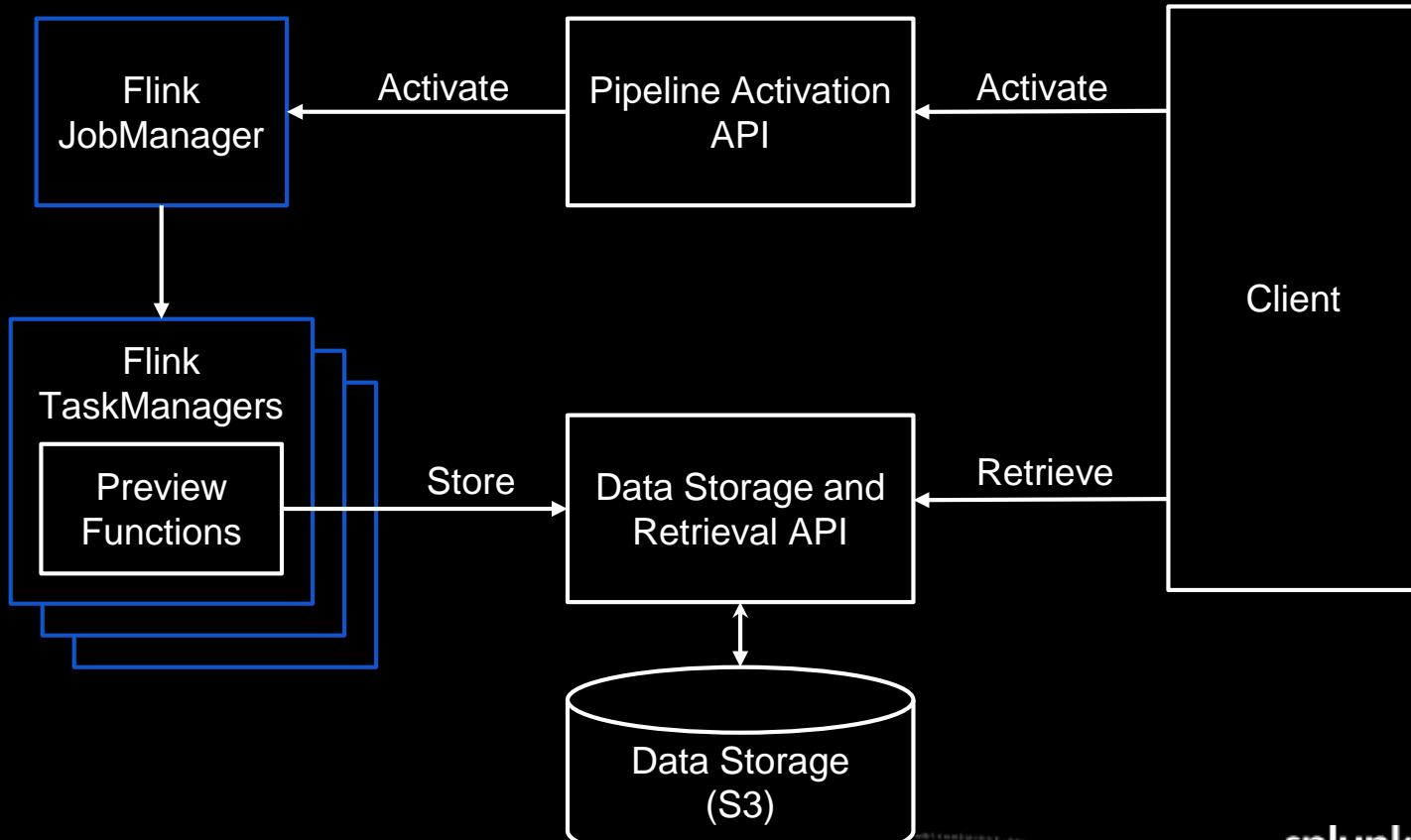
General Architecture



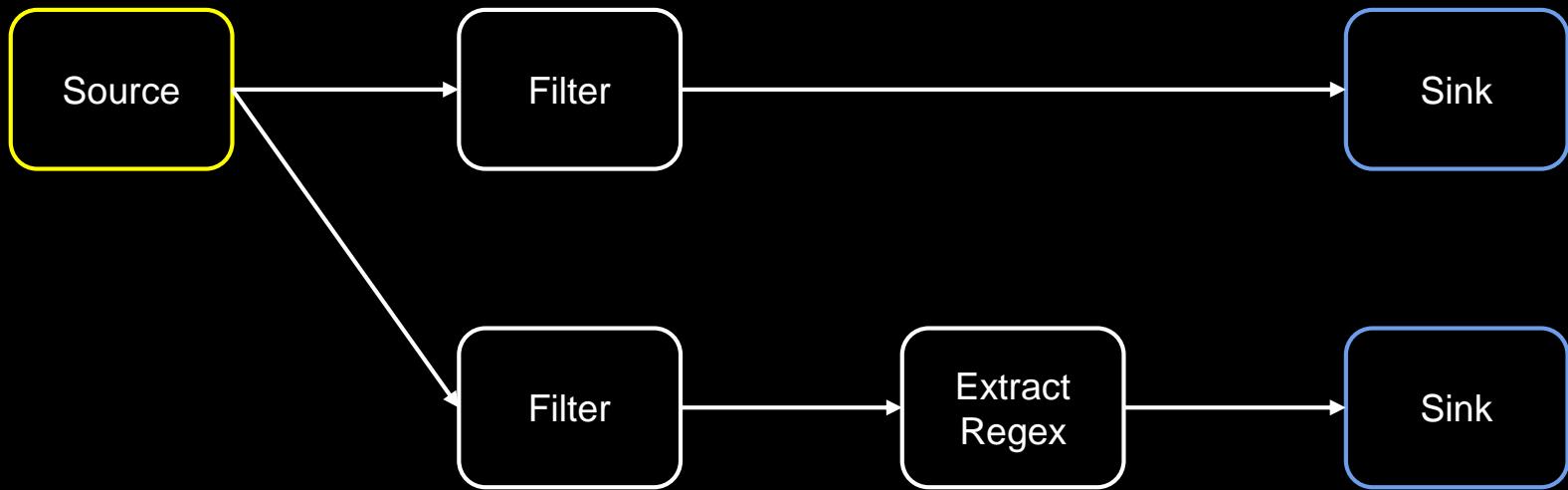
Implementation 1.0



Implementation 1.1

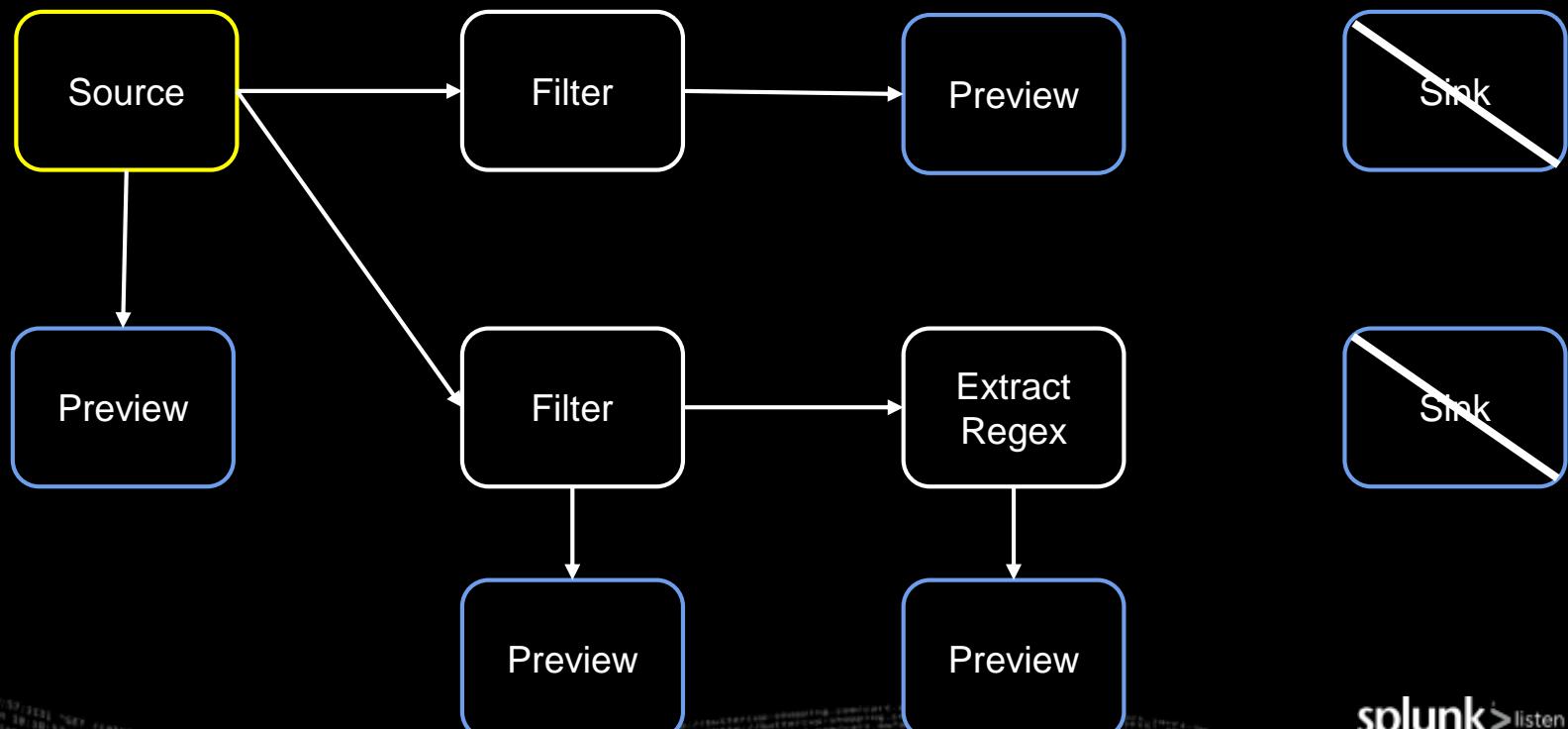


Inserting Preview functions



Inserting Preview functions

*Preview nodes are also sinks



Preview, the function

PreviewSink extends RichSinkFunction

open():

create (HTTP) connection to the Data Storage service

close():

close connection to the Data Storage service

invoke():

increment count of records seen

if preview has hit its upper limit (100), then do nothing

else, (JSON) serialize the input record and send to the Data Storage service

What about checkpoints?

- Checkpoints allow functions to recover state following a failure and restart
- Preview could use checkpoints to store the count of input records
- Without checkpoints, preview functions could send duplicate data

What about savepoints?

- Savepoints are checkpoints you can use to stop and resume your pipeline
- Preview could use savepoints to test against the same data multiple times

What about low-cardinality data?

- Low-cardinality data = data that arrives infrequently, e.g., once a day
- Need to “import data” and run that data through a preview
- This could also work in a framework for automated testing

Can we retrieve data for production pipelines?

- The same architecture should work
- Don't remove sink nodes
- Think through what data is stored and displayed
 - Probably want *most recent* 100 records

What we made

- A place to display the data
 - API and User Interface
- A way to store the data for display
 - Ephemeral storage for ephemeral data
- A way for a pipeline to send the data to storage
 - Function in the pipeline sends data, limits data volume
- A way to test the pipelines during authoring
 - Run a separate pipeline for debugging
- A name for the feature
 - Preview

Thanks for listening!

Questions?

We're hiring!

Matt Dailey

 @matthew_dailey1

splunk>

Acknowledgements

- Joey Echeverria
- Sergey Sergeev
 - Yassin Mohamed
 - Sharon Xie
 - Bashar Abdul-Jawad
 - Maria Yu

KAPPA +

Moving from Lambda and Kappa Architectures to Kappa+ at
UBER

ROSHAN NAIK

FLINK FORWARD 2019 – SAN FRANCISCO

PROBLEM

Realtime jobs often need an offline counterpart:

- **Backfill** – Retroactively fix, or recompute values once all data has arrived.
- **Offline Experimentation & Testing:** Before taking the job online.
- **Online and offline feature generation:** for ML.
- **Bootstrapping State:** for realtime jobs.

CURRENT SOLUTIONS

1. Lambda Architecture [2011]

- Nathan Marz (Creator of Apache Storm)
 - ["How to beat the CAP theorem"](#)
- Evidence of prior art [1983]:
 - Butler Lampson (Turing Award Laureate)
 - ["Hints for Computer System Design"](#) – Xerox PARC
- Core Idea: Streaming job for realtime processing. Batch job for offline processing.

2. Kappa Architecture [2014]

- Jay Krepps (Creator of Kafka, CoFounder/CEO Confluent)
 - ["Questioning the Lambda Architecture"](#)
- Core Idea: Long data retention in Kafka. Replay using realtime code from an older point.

LIMITATIONS : LAMBDA ARCHITECTURE

- Maintain dual code for Batch and Streaming
- Batch APIs often lack required constructs (e.g. sliding windows)
- **Variation:** Unified API : SQL / Beam. – Offline job run in batch mode
- **Limitations of Batch mode (e.g. Spark):**
 - Divide large jobs into smaller ones to limit resource consumption
 - Manual/automated sub job co-ordination
 - Windows that span batch boundaries are problematic

LIMITATIONS : KAPPA ARCHITECTURE

- **Longer retention in Kafka: Expensive, Infeasible**
 - Kafka not really a data warehouse. More expensive than HDFS.
 - Retention beyond a few days not feasible. Single node storage limits partition size.
- **Workaround 1:** Tiered Storage (Pulsar)
 - Data duplication: Usually need a separate queryable copy in Hive/warehouse.
 - Low utilization: old data accessed only by Backfill jobs.
- **Workaround 2:** Mini batches
 - Load small batches into Kafka and process one batch at a time.
 - Sort before loading in Kafka. Try to recreate original arrival order.
 - Expensive: Copying to Kafka and sorting are both costly.
- **Issues when using multiple sources**
 - Low volume topic drains faster → messes up windowing → dropped data or OOM

DESIRED CHARACTERISTICS

- Reuse code for Online and Offline Processing.
- Windowing should work well in offline mode as well.
- No splitting jobs. Single job should processes any amount of data.
- Hardware requirements should not balloon with size of input data.
- Not have to rewrite jobs using new APIs.
- Efficient.

KAPPA+

Introducing the Architecture

KEY CHANGE IN PERSPECTIVE

- **Decoupling Concepts:**
 - Bounded vs Unbounded (Nature of Data)
 - Batch vs Streaming (Nature of Compute)
 - Offline vs Realtime (Mode of Use)
- **Instead of thinking:** How to enable any job in Streaming and Batch mode.
 - Lambda / SQL / Beam / Unified APIs
- **Think:** Limits to job types that can run in Realtime (and Offline) mode.
 - Kappa+

Impact:

- No need to support every type of batch job (departure from Unified API approach).
- Identify the types of jobs to support: Kappa+ job classification system.

ARCHITECTURE

Central Idea - counter intuitive

- Use Streaming compute to process data directly from warehouse. (i.e. not tied to Kafka)

Architectural Components:

1. Job classification system

- 4 categories

2. Processing model

- Same processing basic model with tweaks based on job category

Assumes: Data in warehouse (Hive/Hdfs/etc) is partitioned by time (hourly/daily/etc).

JOB CLASSIFICATION SYSTEM

- **Category 1 : Stateless Jobs**

- No windowing. Memory not a concern.
- Data order usually not concern.

- **Category 2 : Windowing with aggregation (Low - Medium Memory)**

- **Eg:** Aggregated Windows: sum / avg / count / min / max / reduce
- Retains only aggregate value in each window.
- Order of data is important. But solvable without need for strict ordering.

- **Category 3 : Windowing with retention (High Memory)**

- Holds on to all records till window expiration.
- **Eg.** Joins, pattern analysis within window.
- Memory requirements much higher than cat 2.

- **Cat 4 : Global Windows with retention**

- **E.g.** Sorting entire input data. Joins without windowing.
- Not found in realtime jobs.

PROCESSING MODEL

1. Partially ordered reads

- **Strict Ordering across partitions:** Only one partition at a time, older partitions first.
 - Constrains memory/container requirements to what is needed to process 1 partition.
 - Single job can process any number of partitions with finite resources.
 - Helps windowing correctness.
- **Un-Ordered reads within partition**
 - Read records/files within a partition in any order. Opens up concurrency and high throughputs.
 - Order could be exploited if necessary.

2. Emit watermark when switching to next partition

- Allows Out-Of-Order reads within partition and windowing correctness.

3. Lockstep progression, in case of multiple sources

- All sources move to next partition at the same time.
- Prevents low volume sources from racing ahead.

HANDLING EACH CATEGORY

- Cat 1 : Stateless
 - Nothing special. Set parallelisms based on desired throughput.
- Cat 2 : Windowing with aggregation (Low – Med mem)
 - Employ **memory state backend**.
 - **Windowing parallelism** based on amount of data hosted in memory for one partition. Other parallelisms, based on throughput.
- Cat 3 : Windowing with retention (High Mem)
 - **Either:** Use RocksDB state backend.
 - **Or:** reduce partition size, and use Mem state backend.
 - **Or:** Look into exploiting order within partition.

BENEFITS OVER BATCH

BATCH (SQL/ BEAM/ UNIFIED API)

1. Resource requirements grows with total data volume.
 - Tricky to estimate and allocate
2. Split into smaller jobs and coordinate them.
 - Windows that cross batch boundaries are problematic.
3. Results visible after all data is processed.

Note: Kappa+ processing model could be adopted in Unified APIs to address these limitations.

KAPPA+

1. Resources bounded by amount needed to process 1 partition.
 - Easier to estimate and allocate
2. Single job can process any number of partitions.
3. Results visible after each partition.

IMPLEMENTATION

Architecture is not tied any Streaming Engine.

ADOPTING KAPPA+ ON STREAMING ENGINES

- **No new APIs.**
- **Hdfs/Hive/etc. Sources need behavioral change:**
 1. One partition at a time, older partitions first.
 2. Concurrent reads within partition.
 3. Lock step progression in case of multiple sources.
 - Kafka source needs to only support #3 since data is already in order.
- **Watermarking:**
 - Emit watermarks at the end of partition to flush windows.

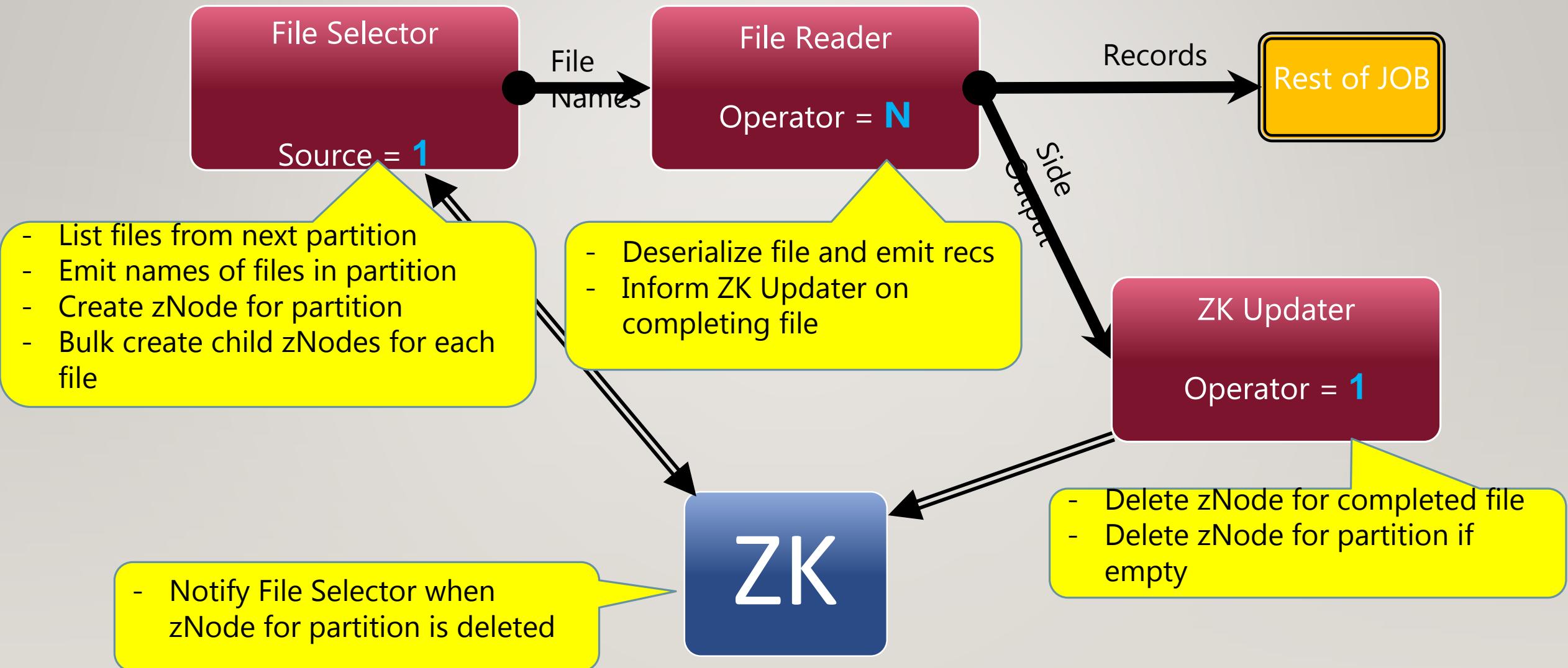
A JOB SUPPORTING REALTIME & OFFLINE

```
dataSource = offlineMode ? hiveSource : kafkaSource;  
watermaker = offlineMode ? new OfflineWM() : new RealtimeWM();  
dataSource.assignWatermarkGenerator(watermarker);  
  
// Same logic. Adjust parallelisms for offline & online modes.  
job = dataSource.transform(...)  
      .filter(...)  
      .keyBy(...)  
      .window(...)  
      ...
```

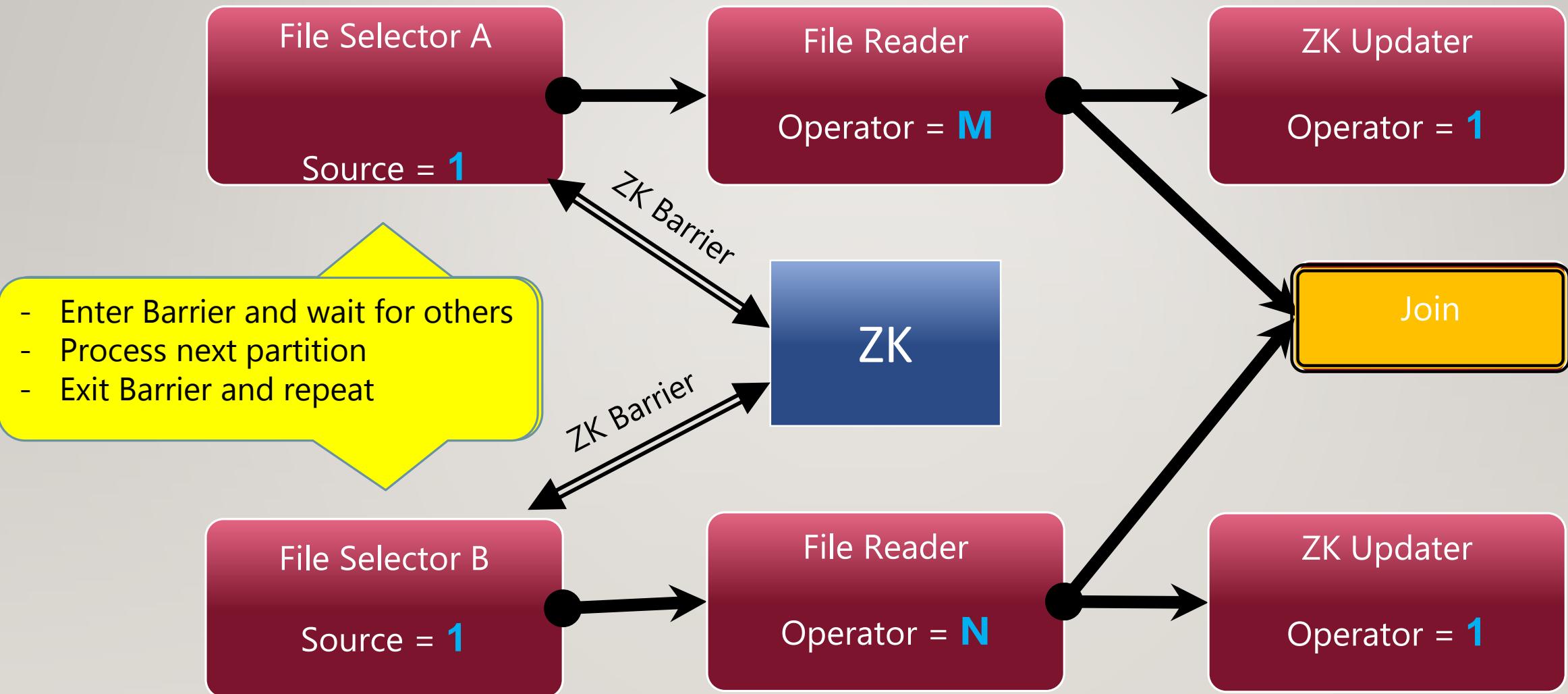
KAPPA+ ON FLINK

UBER internal Hive (/HDFS) source with Kappa+ support.

ONE PARTITION AT A TIME & CONCURRENT READS



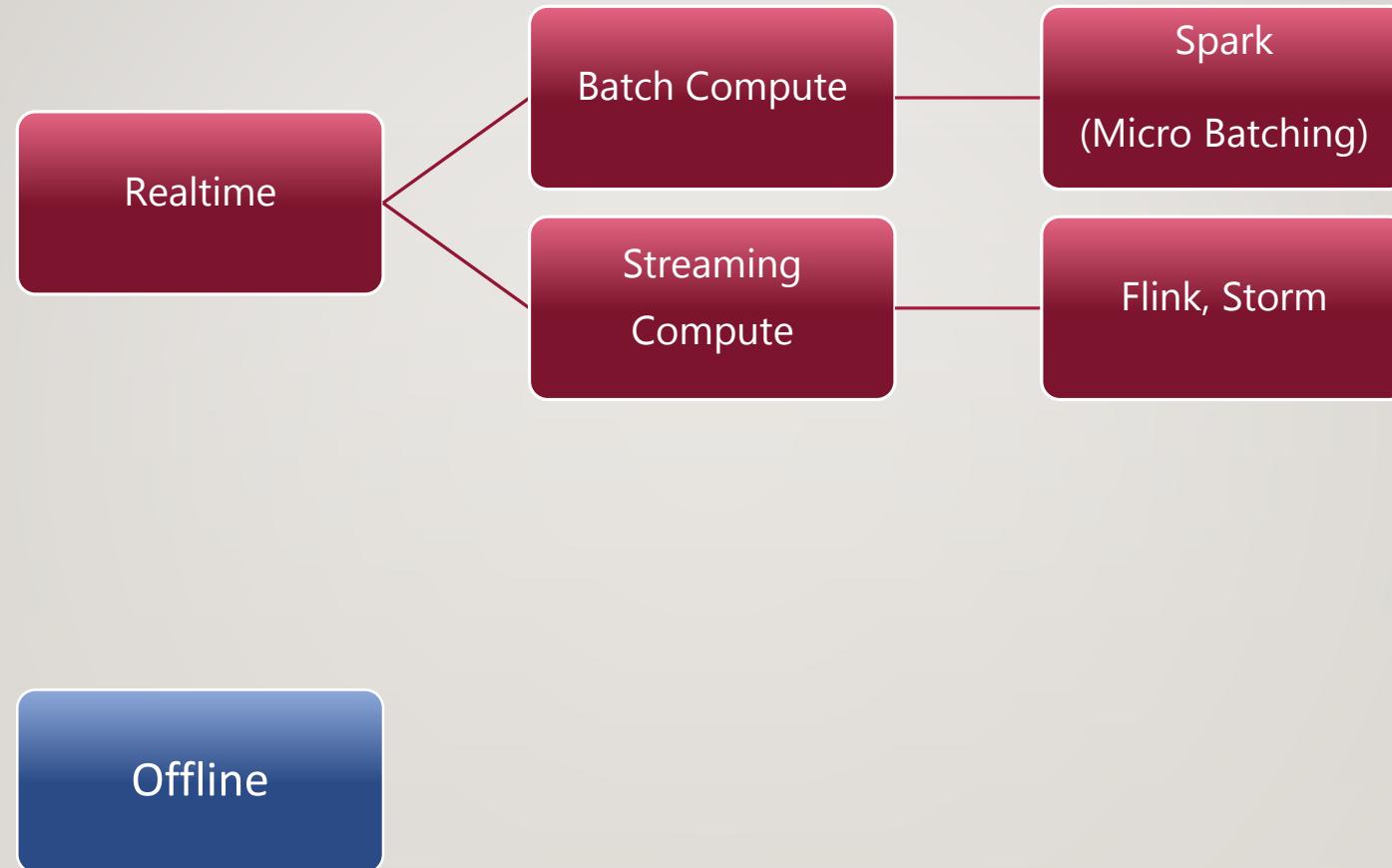
MULTI SOURCE LOCK STEP PROGRESSION



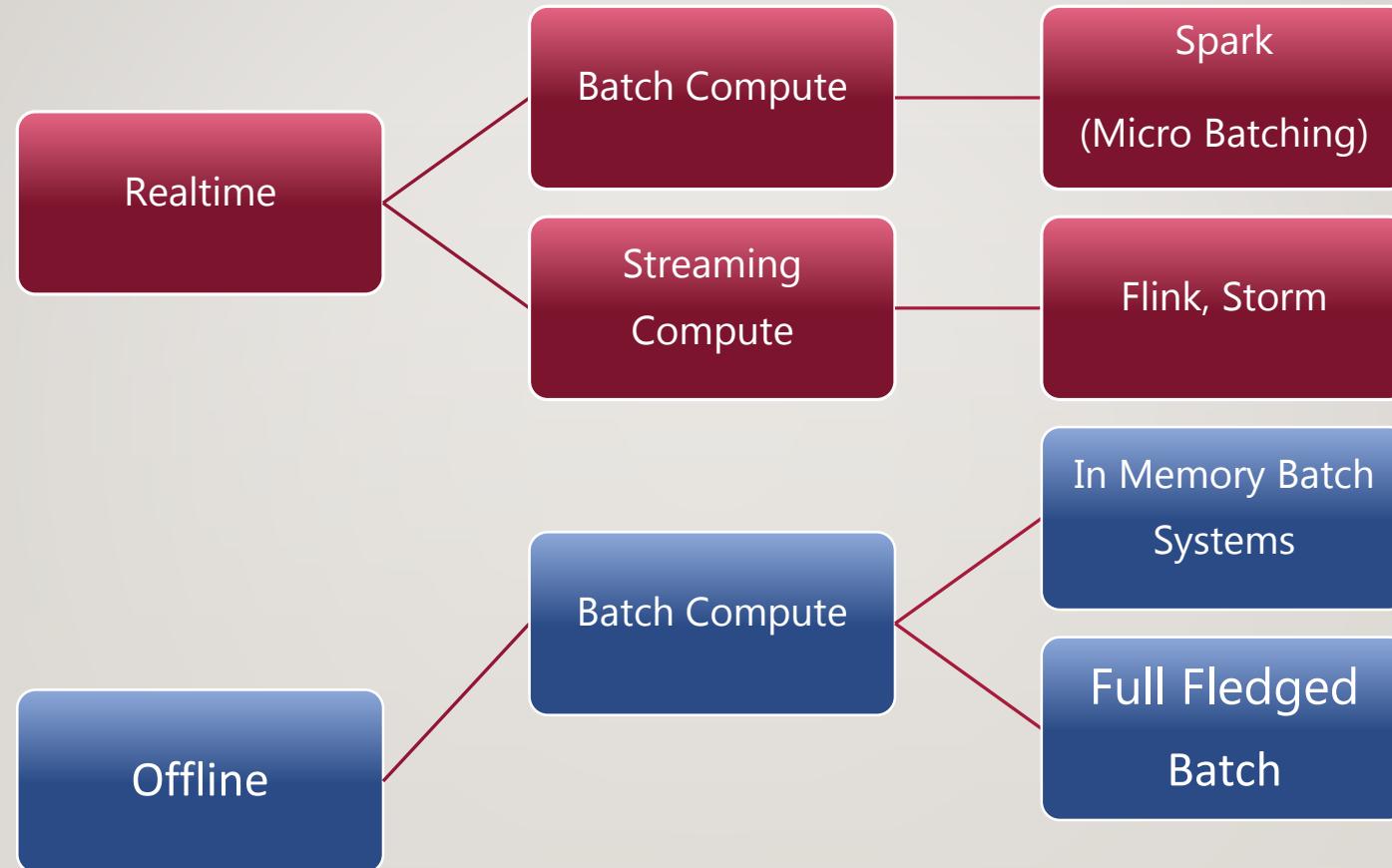
DETAILS

1. **Time Skew:** If arrival time is used for partitioning data in the warehouse, instead of event creation time (used by job). There can be two types of data skews:
 - **Forward skew:** Some events move into a future partition. For example due to late arrival.
 - Could lead to appearance of missing data.
 - Consider processing an additional partition after the last one, if this is an issue.
 - **Backward skew:** Some events moving into an older partition.
 - Can lead to appearance of data loss. As the events are not in the partition that you processed.
 - Improper watermarking can close Windows prematurely and cause data loss.
2. **Differing partition strategies:** Job has two sources. First source reads Hive table with daily partitions, second source reads table with hourly partition.
 - **Solution:** Watermark progression dictated by daily (i.e. larger) partition
3. May need to **throttle** throughput of offline job if writing to production critical destination.

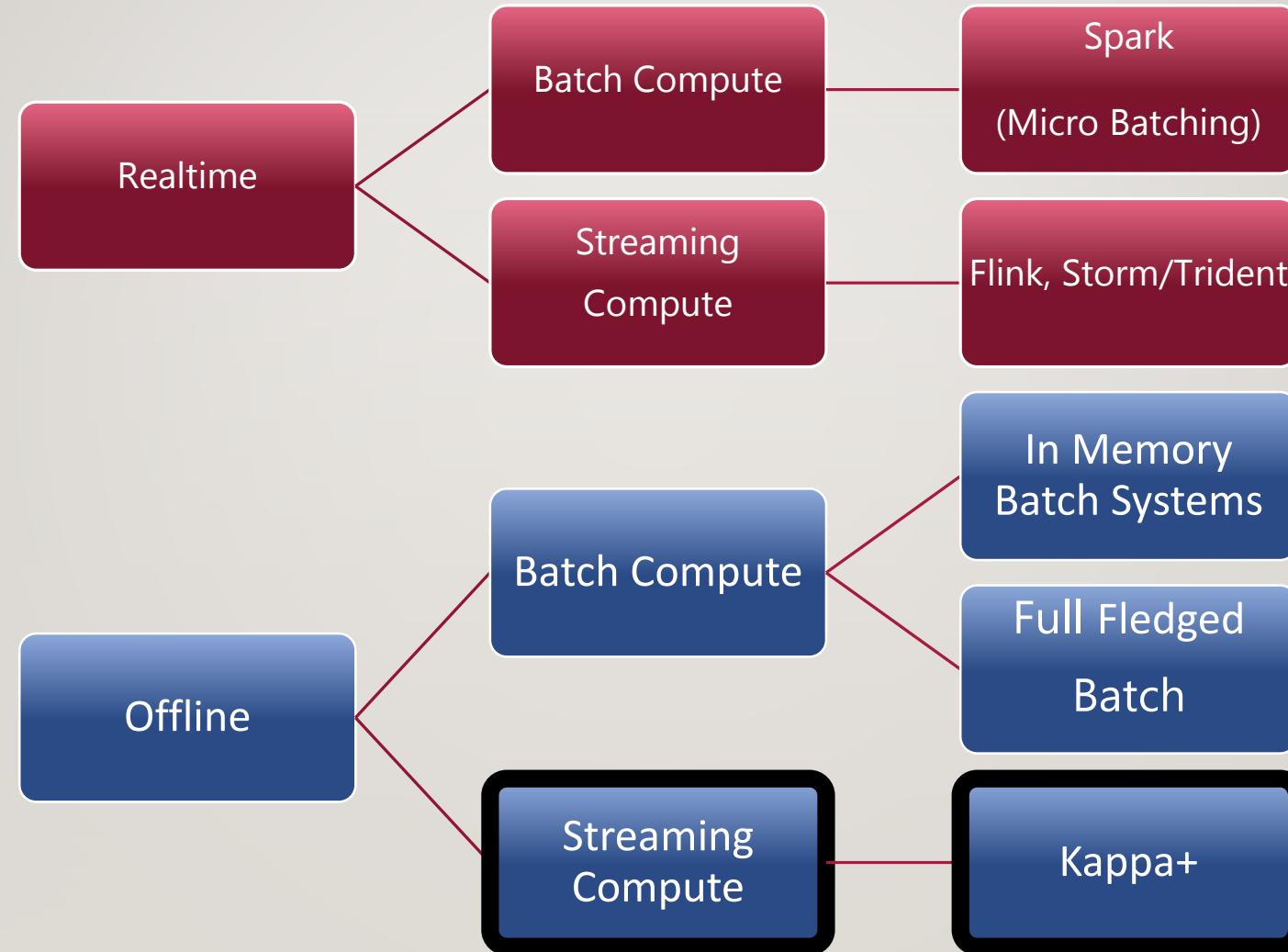
DISTRIBUTED COMPUTING



DISTRIBUTED COMPUTING



DISTRIBUTED COMPUTING



QUESTIONS

Email: [@roshan@uber.com](mailto:roshan@uber.com)
[@UberEng](https://twitter.com/@UberEng)

UBER Engineering Blog: eng.uber.com

Twitter: [@naikrosh](https://twitter.com/@naikrosh),

UBER is hiring!! Realtime Platform needs your expertise!