Streaming your shared ride

Berlin Buzzwords, June 17th 2019
go.lyft.com/berlin-buzzwords-2019

Thomas Weise | @thweise

lyft

Agenda

1. Streaming Architecture Overview
2. Streaming Analytics
3. Streaming Applications
4. Integrations / Connectors
5. Deployment

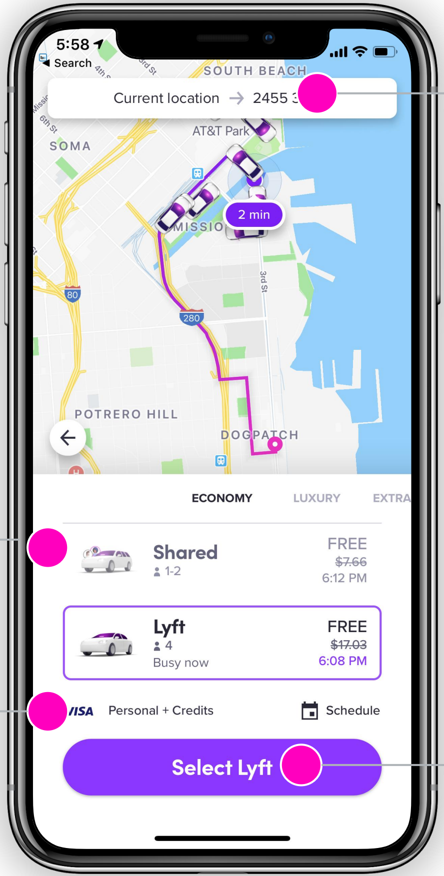
(Streaming) Data at Lyft

Pricing

Dynamic Pricing
Supply/Demand curve
ETA

Fraud

Behaviour Fingerprinting
Monetary Impact
Imperative to act fast



Core Experience

Top Destinations

User Delight

Notifications
Detect Delays
Coupons

Data platform users

Data Modelers

Analysts

Product Managers

Analytics

General Managers

Biz ops

Data Scientists

Engineers

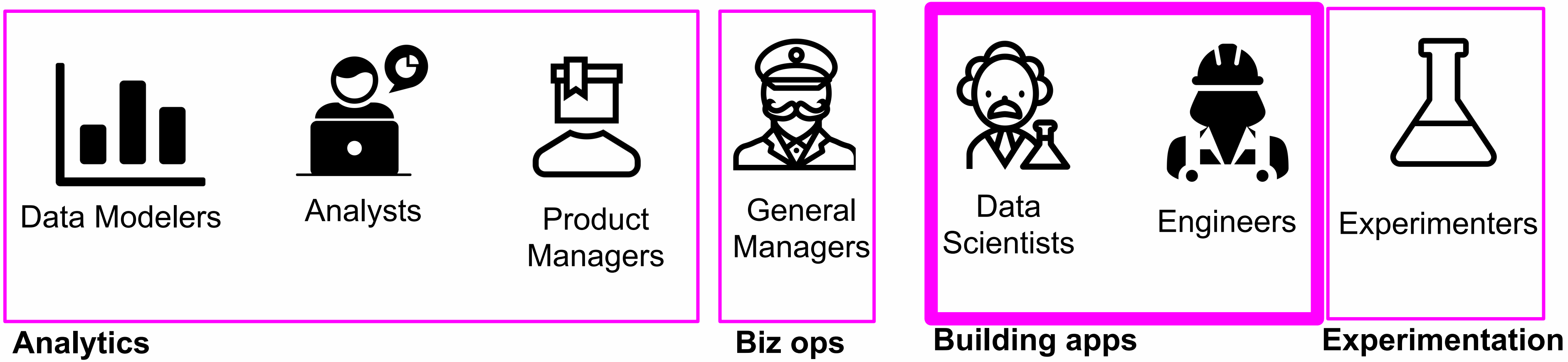
Building apps

Experimenters

Experimentation

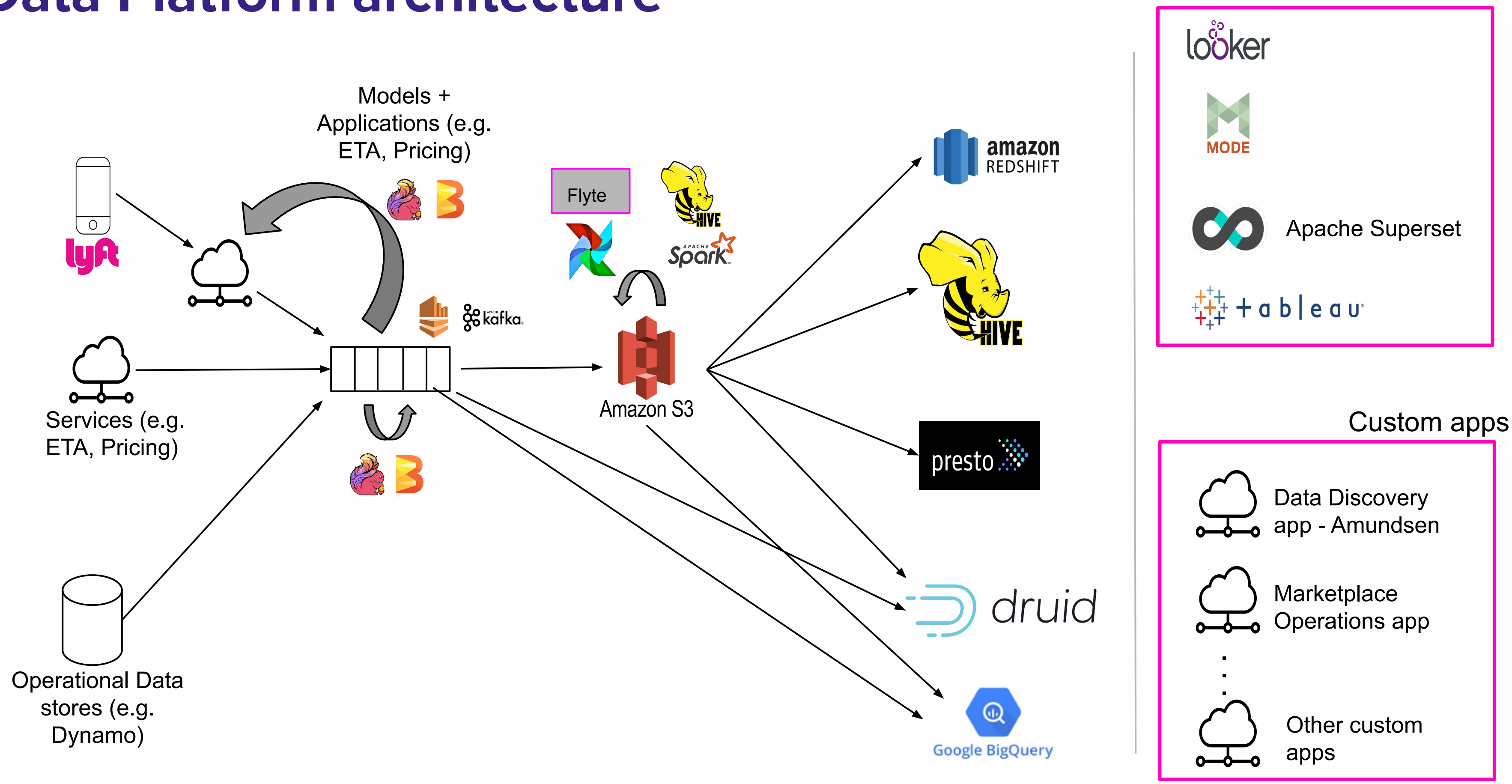
Data Platform

Data platform users

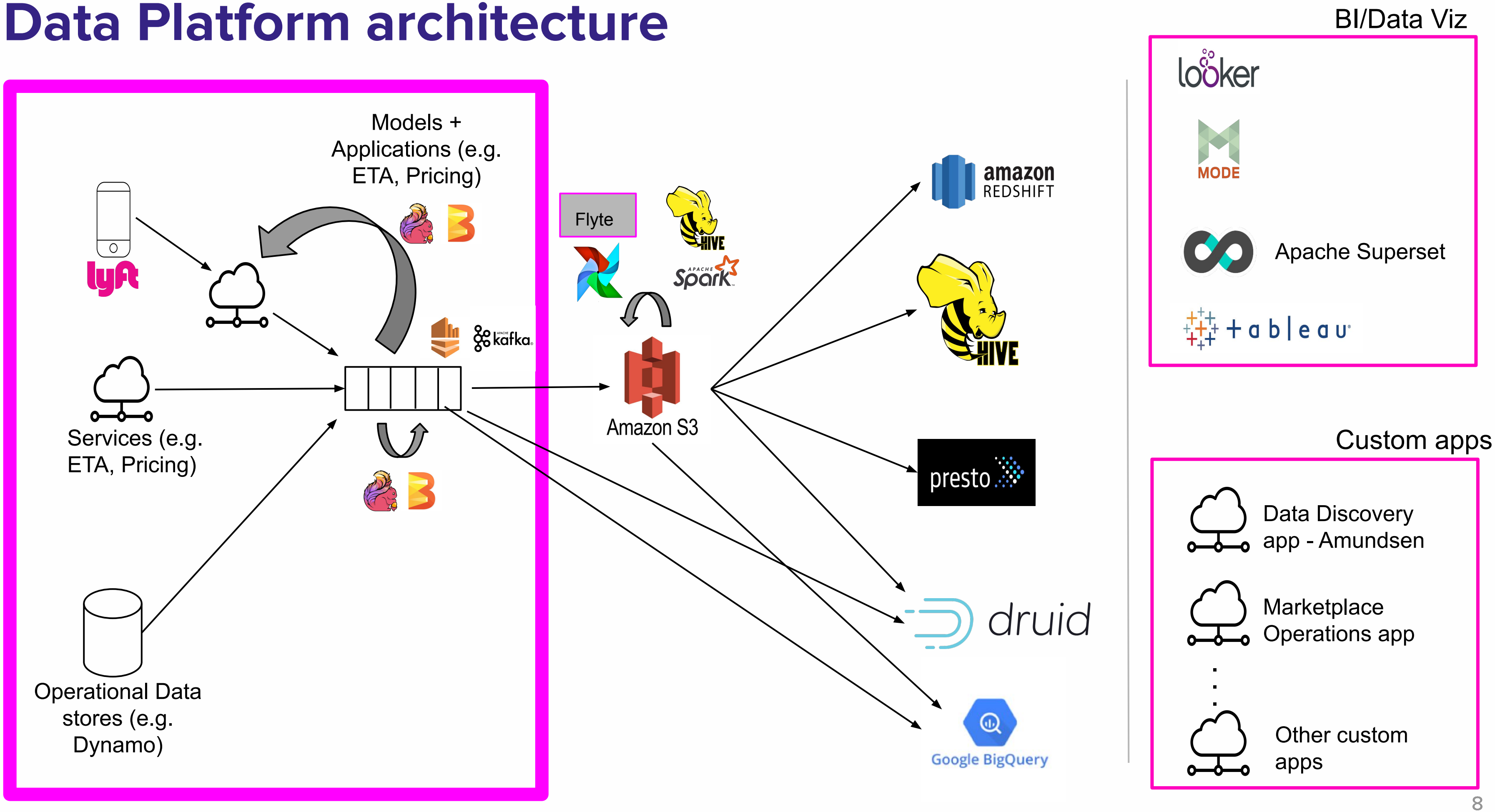


Data Platform

Data Platform architecture



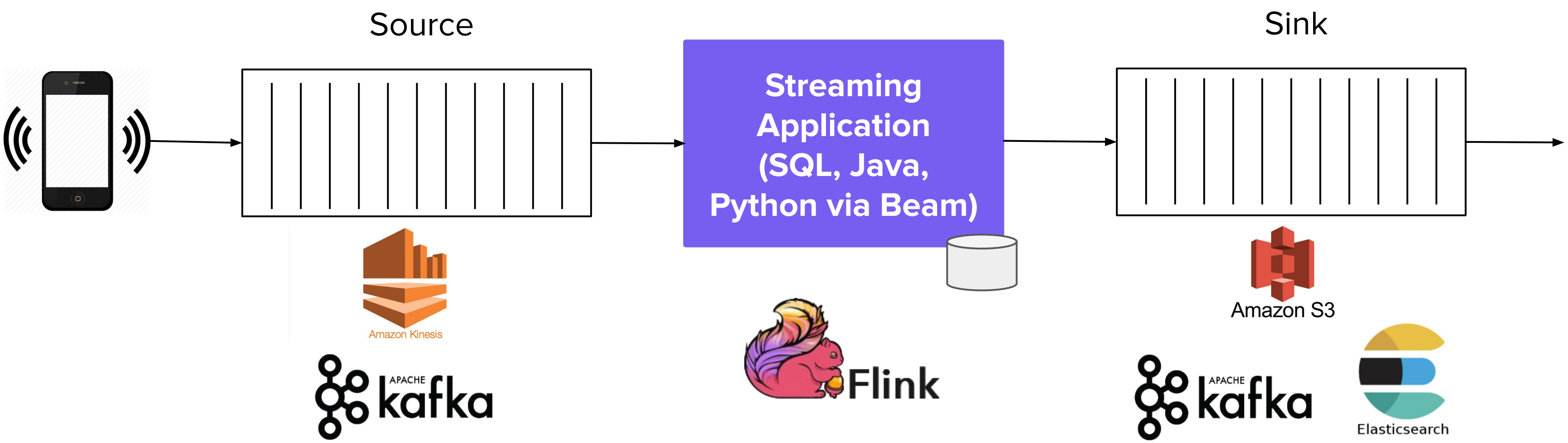
Data Platform architecture



PubSub: From Kinesis to Kafka

- Latency
  - Kinesis exhibits high tail latency, with p99 write latency over 100ms and p999 write latency reaching a few seconds
  - Kafka: <20ms p99 write latency and <75ms p999
  - Difficult to achieve latency SLA **and** durability
- Fanout limitation
  - Each Kinesis shard can support at most five read transactions per second, with a maximum total read rate of 2MB per second (per shard)
  - Even enhanced fanout capability with Kinesis 2.0 API is still limited to 5 consumers by default
- Scalability limitation
  - Limit on number of shards (by default 500), we are using 1000s
  - Increase only by factor of 2
  - Resharding is manual, disruptive and time-consuming
  - Cost increases with number of shards

Streaming Compute Stack



Stream / Schema Registry	**Deployment Tooling**	**Metrics & Dashboards**	**Alerts**	**Logging**
Amazon EC2	Amazon S3	Wavefront	~~Salt (Config / Orca)~~	~~Docker (Development)~~

Flink Abstraction Levels

```
SELECT room, TUMBLE_END(rowtime, INTERVAL '1' HOUR), AVG(temp)
FROM sensors
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), room
```

High-level Analytics API — Stream SQL / Tables *(dynamic tables)*

Stream- & Batch Data Processing — DataStream API *(streams, windows)*

```
val stats = stream
    .keyBy("sensor")
    .timeWindow(Time.seconds(5))
    .sum((a, b) -> a.add(b))
```

Stateful Event-Driven Applications — Process Function *(events, state, time)*

```
def processElement(event: MyEvent, ctx: Context, out: Collector[Result]) = {
  // work with event and state
  (event, state.value) match { … }

  out.collect(…) // emit events
  state.update(…) // modify state

  // schedule a timer callback
  ctx.timerService.registerEventTimeTimer(event.timestamp + 500)
}
```

Use Case Categorization

	Analytics	Applications
Paradigm	Declarative	Imperative
Language	SQL, Table API	Transforms in Java, Python, ...
Schema	External (tables)	Expressed in programming language
Execution	Optimized by system	As programmed
State and time	Automatic state and triggers	Explicit control over state and triggers
Use cases	Many (data preparation, feature generation, ...)	Fewer with complex, use case specific logic
TTV/TCO	Lower (self-serve, fully managed, fast onboarding)	Higher (skill set, longer to implement)

Analytics use case

Dryft

fully managed data processing engine, powering real-time features and events

- **Need - Consistent Feature Generation**
  - The value of your machine learning results is only as good as the data
  - Subtle changes to how a feature value is generated can significantly impact results
- **Solution - Unify feature generation**
  - Batch processing for bulk creation of features for training ML models
  - Stream processing for real-time creation of features for scoring ML models
- **How - Flink SQL within fully managed service**
  - Use Flink as the processing engine using streaming or bulk data
  - Add automation to launch and maintain feature generation programs at scale

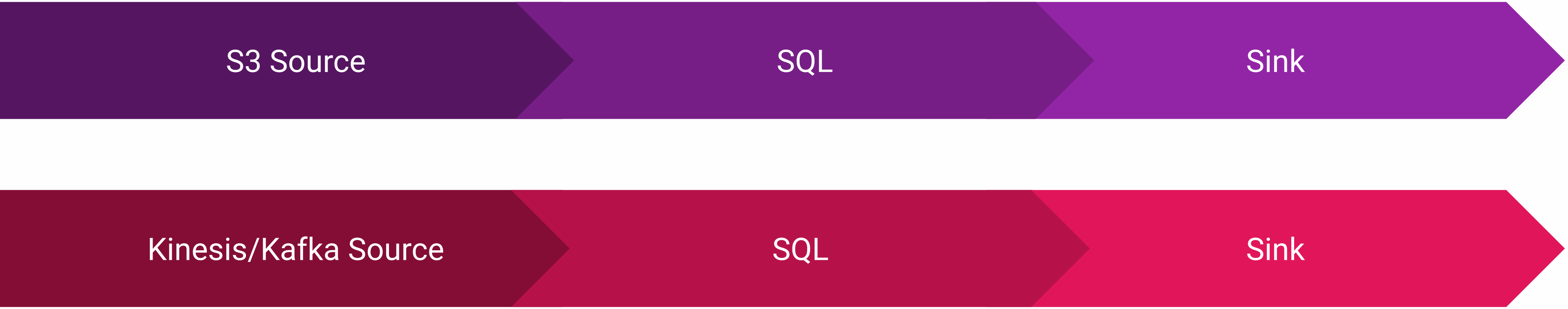
https://www.slideshare.net/SeattleApacheFlinkMeetup/streaminglyft-greg-fee-seattle-apache-flink-meetup-104398613/#11

Dryft Program Specification

Configuration file	decl_ride_completed.sql

{
 "source": "dryft",
 "query_file": "decl_ride_completed.sql",
 "kinesis": {
 "stream": "declridecompleted" },
 "features": {
 "n_total_rides": {
 "description": "All time ride count per user",
 "type": "int",
 "version": 1 }
 }
 }
	SELECT COALESCE(user_lyft_id, passenger_lyft_id, passenger_id, -1) AS user_id,
	 COUNT(ride_id) as n_total_rides
	 FROM event_ride_completed
	 GROUP BY COALESCE(user_lyft_id, passenger_lyft_id, passenger_id, -1)

Dryft Program Execution



- Backfill - read historic data from S3, process, sink to S3
- Real-time - read stream data from Kinesis/Kafka, process, sink to DynamoDB

Bootstrapping

- Read historic data from S3
- Transition to reading real-time data
- https://www.ververica.com/flink-forward/resources/bootstrapping-state-in-apache-flink

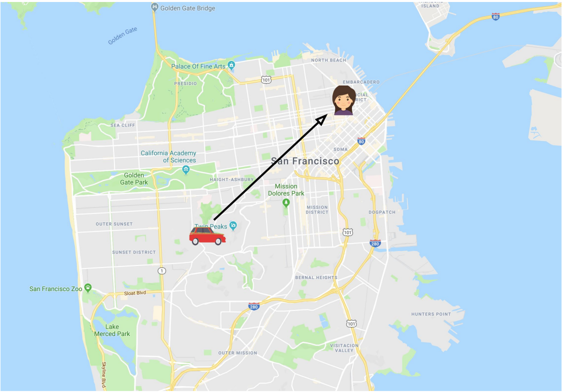
Benefits

- Low latency computation on streaming data
- Fast onboarding
- Minimal development time
- Fully managed
- Self Service
- Reliable

Applications use case

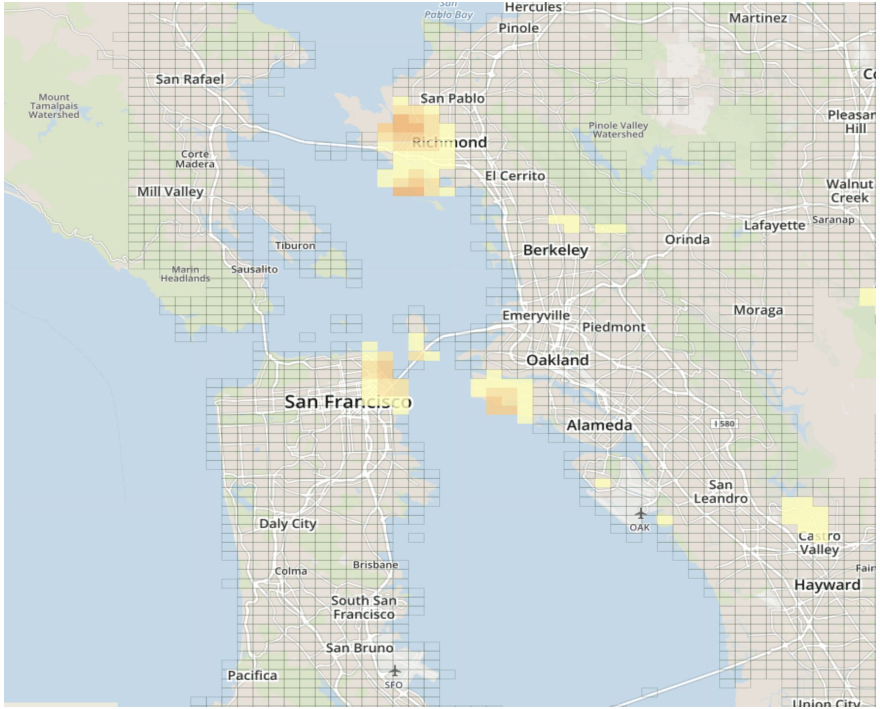
Dynamic Pricing

- Dynamic Pricing - price evaluated minutely per location bucket
- An Imbalanced Market is Inefficient
  - Too many available drivers: bad
  - Too few available drivers: bad
  - Solution: Price lever controls passenger request rate, which maintains healthy supply levels
- Result: increase price if demand >> supply



What is PrimeTime?

- Belief: There exists some set of optimal price multipliers per location/time bucket
- PrimeTime- Lyft product that sets a multiplier for each gh6 each minute
- Example: In ‘9q8yyv’, at 5:01pm PST, PrimeTime = 2.0
- Scale: Millions of geohashes prices every minute

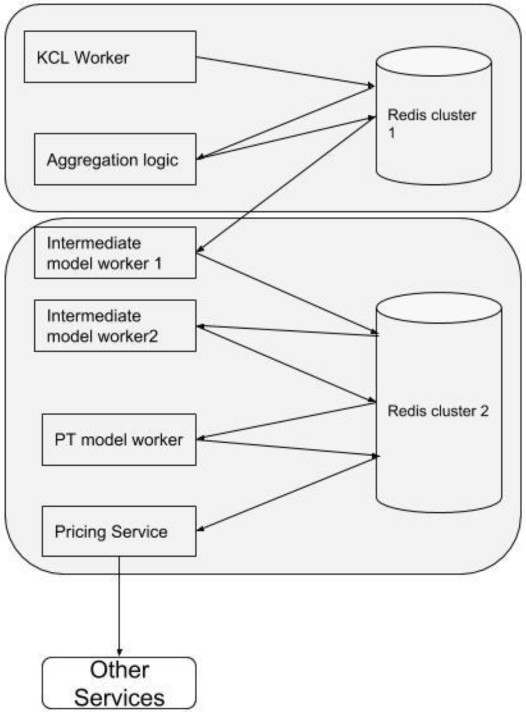


Legacy architecture: A series of cron jobs

- Ingest high volume of client app events (Kinesis, KCL)
- Compute features (e.g. demand, conversation rate, supply) from events
- Run ML models on features to compute primetime for all regions (per min, per gh6)

SFO, calendar_min_1: {gh6: 1.0, gh6: 2.0, ...}

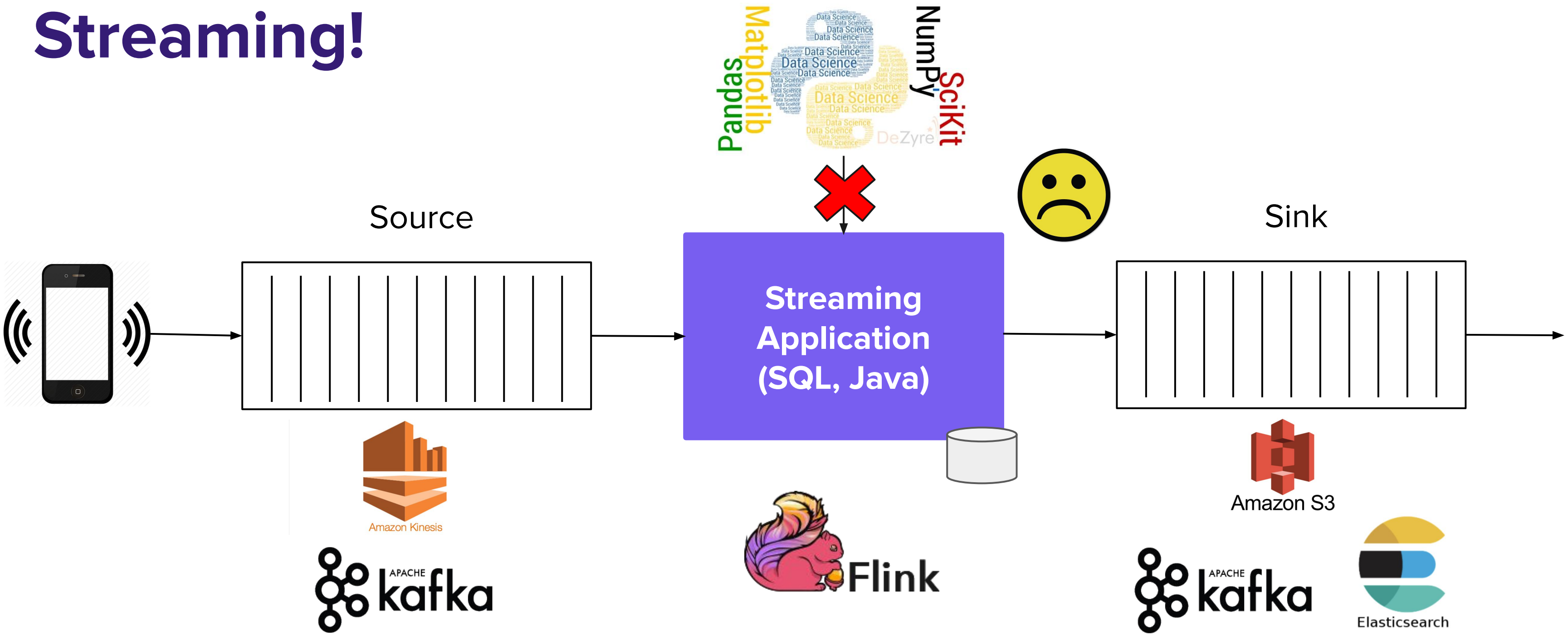
NYC: calendar_min_1: {gh6, 2.0, gh6: 1.0, ...}



Problems

1. Latency
2. Code complexity (LOC)
3. Hard to add new features involving windowing/join (i.e. arbitrary demand windows, subregional computation)
4. No data driven / smart triggers

Streaming!



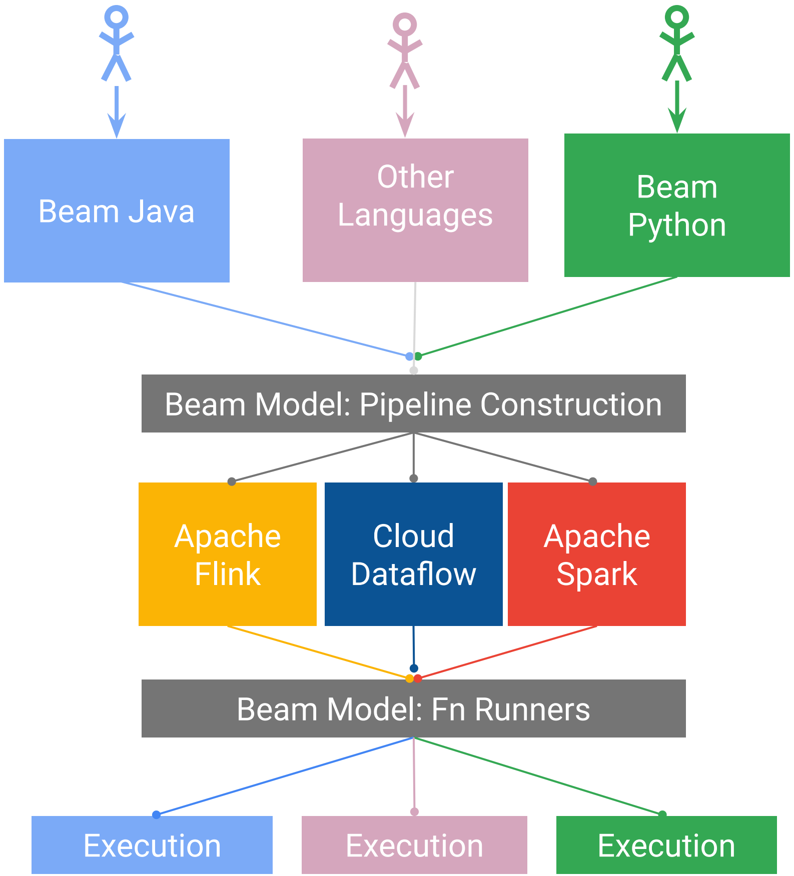
Stream / Schema Registry	**Deployment Tooling**	**Metrics & Dashboards**	**Alerts**	**Logging**
Amazon EC2	Amazon S3	Wavefront	Salt (Config / Orca)	Docker

Apache Beam

1. **End users:** who want to write pipelines in a language that's familiar.

2. **SDK writers:** who want to make Beam concepts available in new languages. Includes **IOs**: connectors to data stores.

3. **Runner writers:** who have a distributed processing environment and want to support Beam pipelines



https://s.apache.org/apache-beam-project-overview

Multi-Language Support

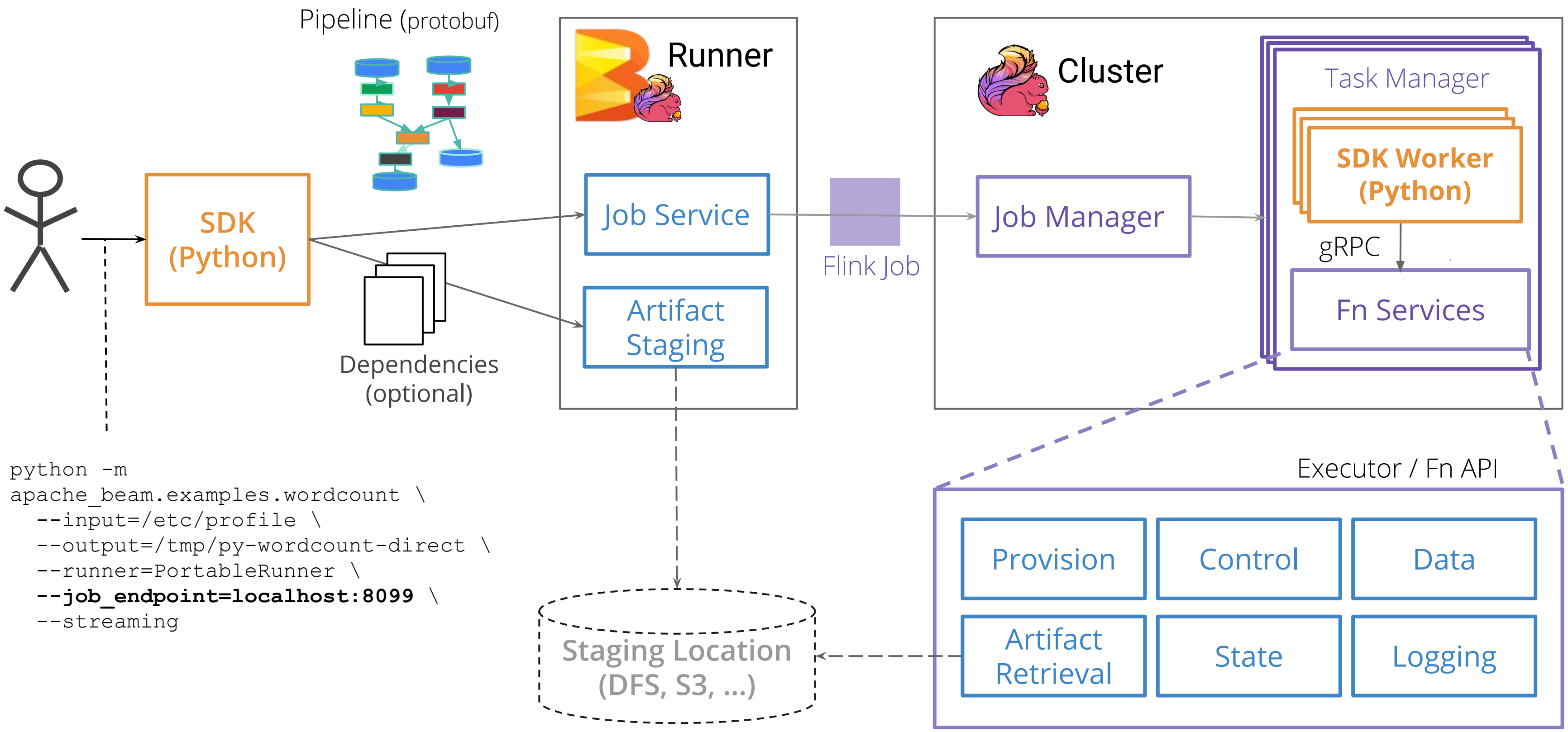
- Started with Java SDK and Java Runners
- 2016: Initiate cross-language support effort
- 2017: Python SDK on Dataflow
- 2018: Go SDK (for portable runners)
- 2018: Python on Flink MVP
- Next: Cross-language pipelines, Samza and other (?) runners

Python Example

```
p = beam.Pipeline(runner=runner, options=pipeline_options)
(p
  | ReadFromText("/path/to/text*") | Map(lambda line: ...)
  | WindowInto(FixedWindows(120)
               trigger=AfterWatermark(
                   early=AfterProcessingTime(60),
                   late=AfterCount(1))
               accumulation_mode=ACCUMULATING)
  | CombinePerKey(sum))
  | WriteToText("/path/to/outputs")
)
result = p.run()
```

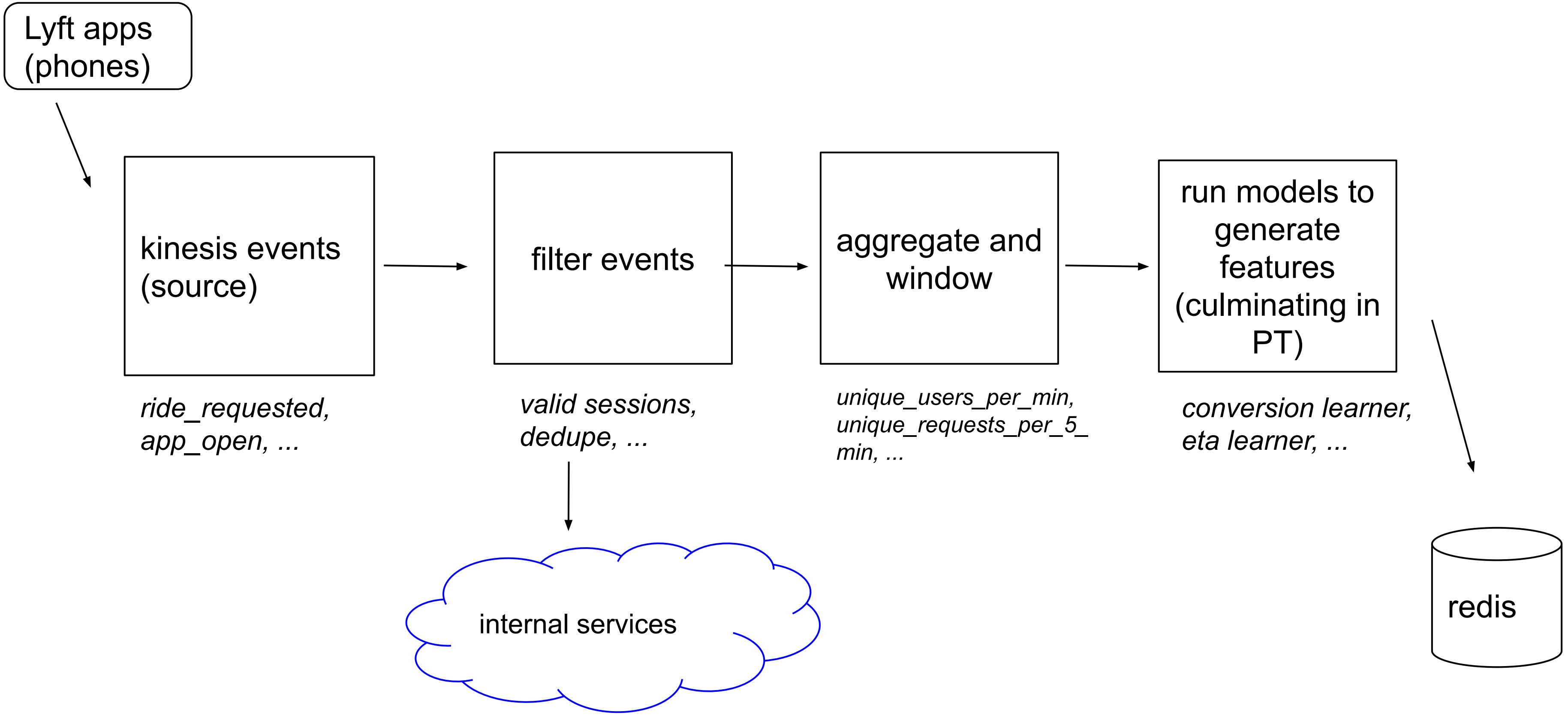
(**What, Where, When, How**)

Python via Beam on Flink




```
python -m
apache_beam.examples.wordcount \
  --input=/etc/profile \
  --output=/tmp/py-wordcount-direct \
  --runner=PortableRunner \
  --job_endpoint=localhost:8099 \
  --streaming
```

Pipeline (conceptual outline)



Benefits

- Latency: 3 minutes -> 30s
  - Latency now dominated by model execution
- Reuse of model code
- 10K => 4K LOC
- Fewer AWS instances

Integrations

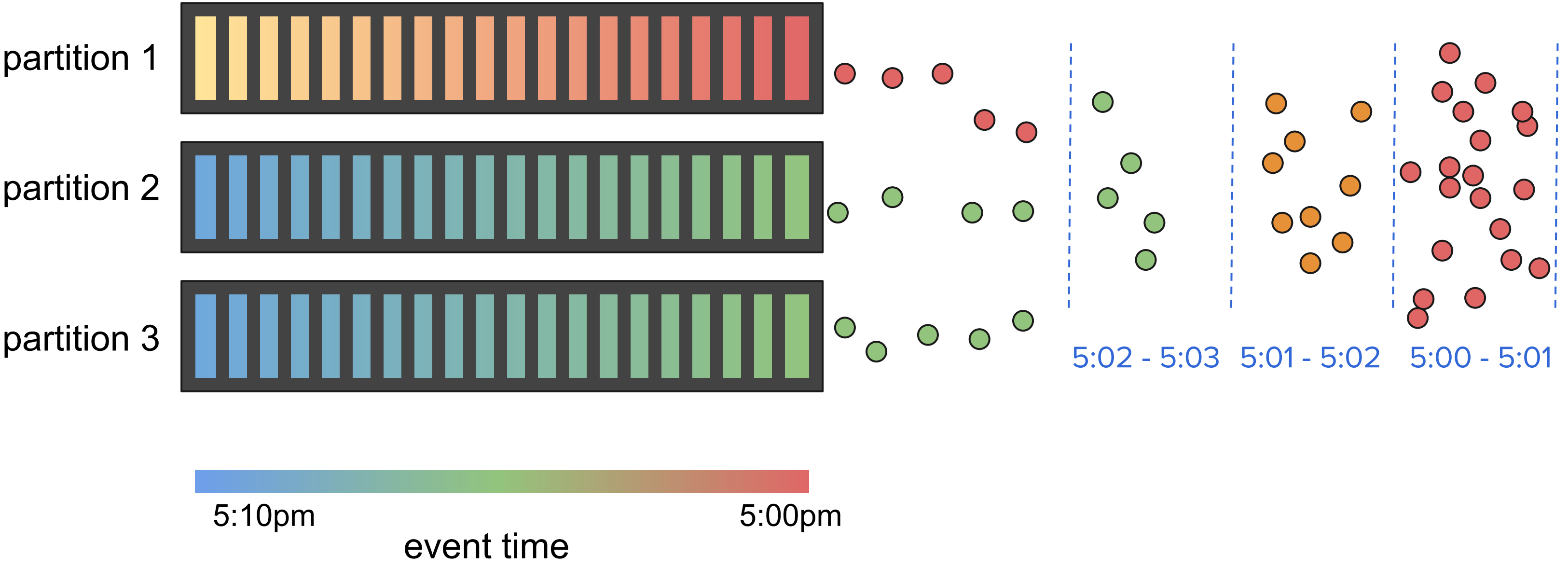
Flink Connectors

- Kinesis Consumer
  - also as custom Beam source
- Kafka Consumer & Producer
- S3 Read & Write
- Elasticsearch
- DynamoDB Streams (special Kinesis Consumer)
- Checkpointing!
  - S3 for checkpoint storage

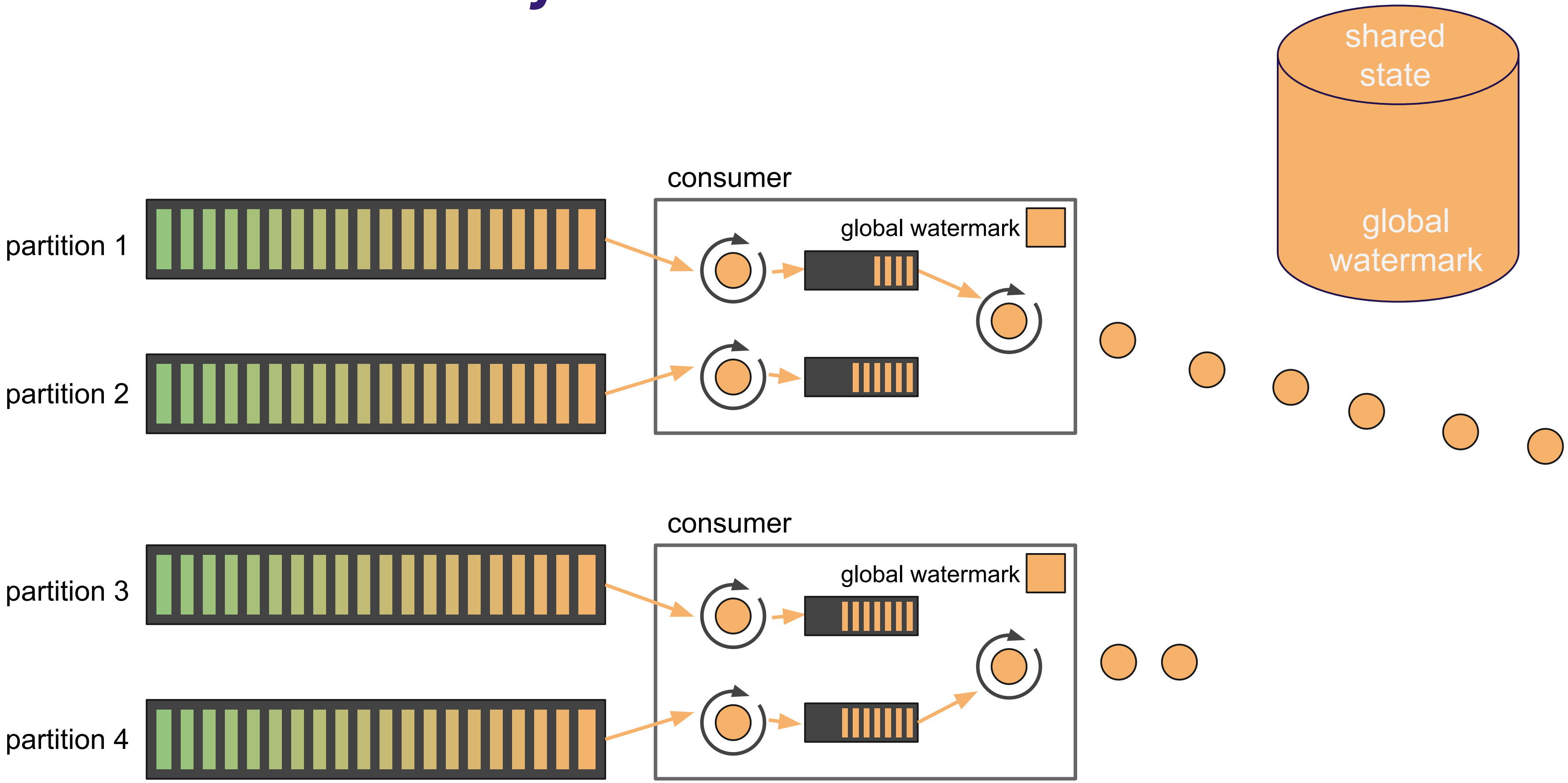
Challenges

- Production readiness
  - Observability, Configuration, Performance
- AWS integration
  - Transient service errors => retries
  - S3 hot shards with checkpointing => entropy injection
- Event time
  - Source watermarks
  - Watermark skew
- Rate controls

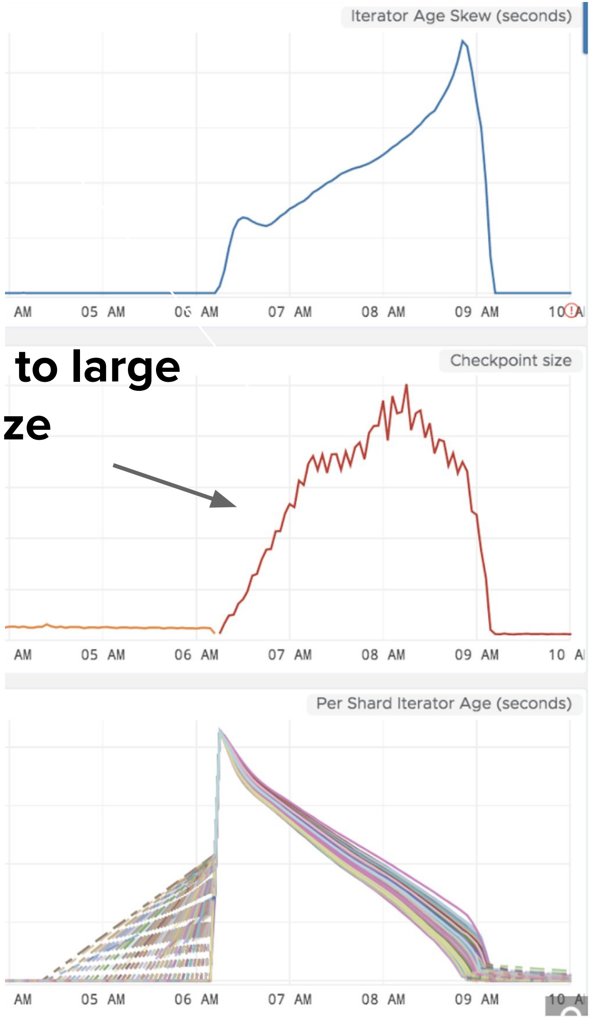
Watermark Skew



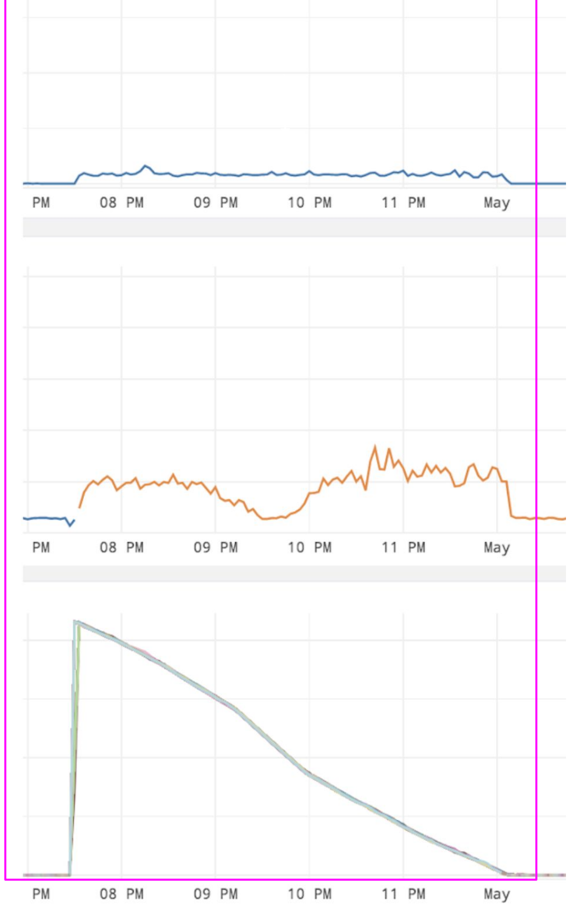
Solution: Source synchronization



skew leading to large state size



with synchronization



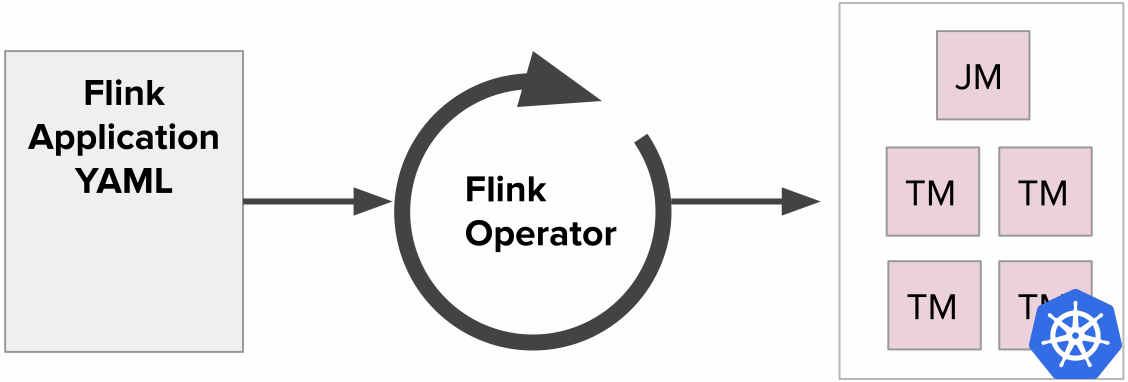
Contribution to Flink

- Support for global aggregates: FLINK-10887
  - Released with Flink 1.8.0
- Synchronization in Kinesis Consumer: FLINK-10921
  - Upcoming Flink 1.9.0
- Synchronization in Kafka Consumer: FLINK-12675
- Long term: New Source Interface: FLIP-27
  - Framework developed by Flink community
  - Will include watermark alignment capability

Deployment

Flink on Kubernetes

- Goal: Improve stability, flexibility, ease of use, and speed of development
- How? By building a Kubernetes operator that manages Flink applications
- Check it out: https://github.com/lyft/flinkk8soperator



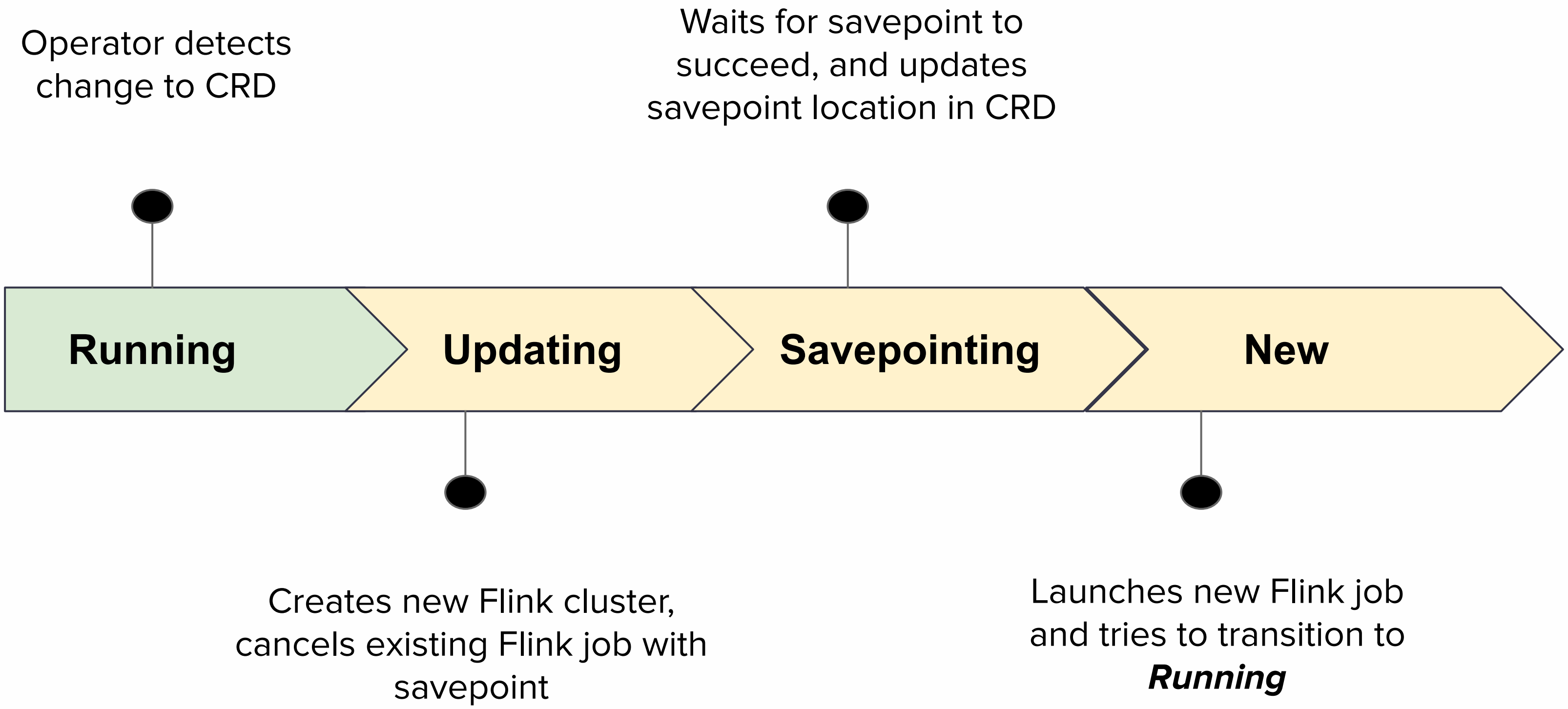
https://www.slideshare.net/FlinkForward/flink-forward-san-francisco-2019-managing-flink-on-kubernetes-flinkk8soperator-anand-swaminathan-ketan-umare

Flink Operator - CRD

- Custom resource represents Flink application
- Single **Flink job** **(**“Flink cluster” == Flink application)
- **Docker image** contains all dependencies
- CRD modifications trigger update (includes parallelism and other Flink configuration properties)

```yaml
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
```

(Stateful) Upgrade



Beam Summit Europe
Berlin 2019

19-20 JUNE, 2019

SAVE THE DATE

FREE EVENT

Beam Summit North America

Las Vegas 2019

11 - 12 SEPTEMBER

BEAM SUMMIT

@

Save the Date!

APACHECON North America

20 YEARS OF APACHE

Las Vegas, Nevada

September 9-12, 2019

Q & A

We are hiring! lyft.com/careers

Slides: go.lyft.com/berlin-buzzwords-2019