The magic behind your Lyft ride prices

A case study on machine learning and streaming

Strata Data, San Francisco, March 27th 2019

Rakesh Kumar | Engineer, Pricing
Thomas Weise | @thweise | Engineer, Streaming Platform

go.lyft.com/dynamic-pricing-strata-sf-2019

Agenda

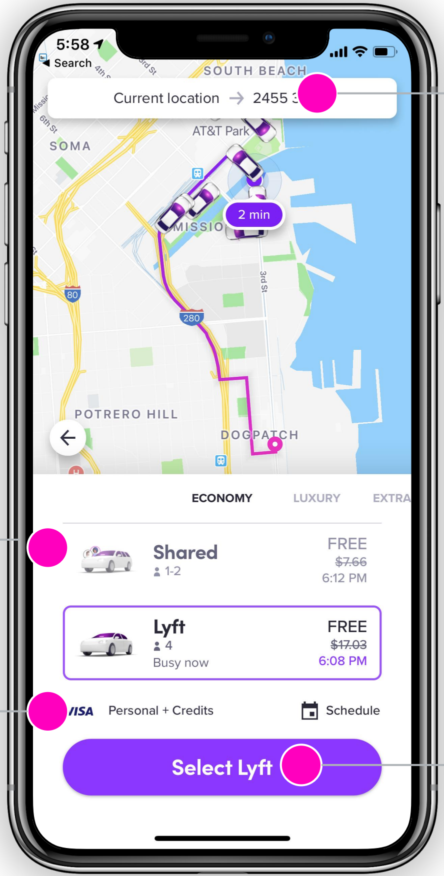
- Introduction to dynamic pricing
- Legacy pricing infrastructure
- Streaming use case
- Streaming based infrastructure
- Beam & multiple languages
- Beam Flink runner
- Lessons learned

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

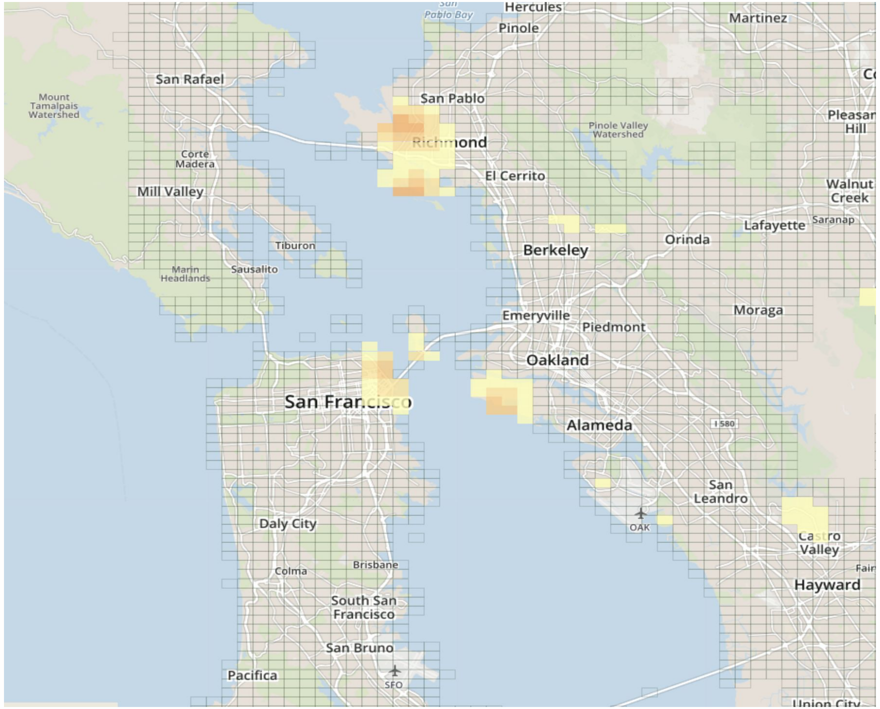
Introduction to Dynamic Pricing

What is prime time?

Location + time specific multiplier on the base fare for a ride

e.g. "in downtown SF at 5:00pm, prime time is 2.0"

Means we double the normal fare in that place at that time



Location: geohash6 (e.g. ‘9q8yyq’)

Time: calendar minute

Why do we need prime time?

- Balance supply and demand to maintain service level
- State of marketplace is constantly changing
- "Surge pricing solves the wild goose chase" (paper)

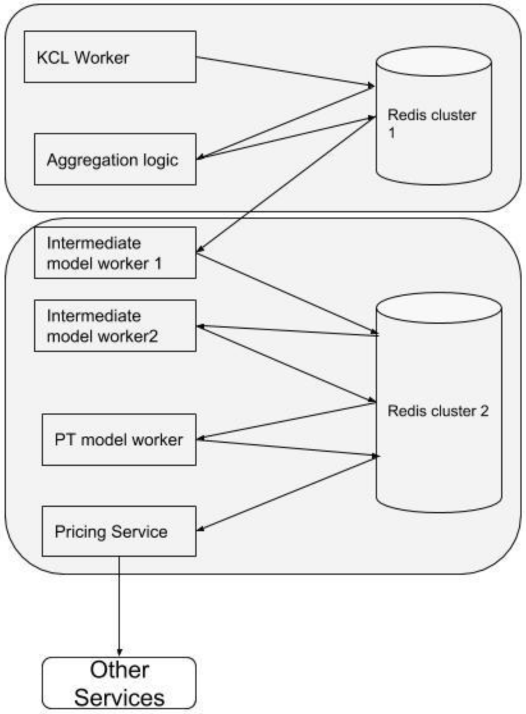
Legacy Pricing Infrastructure

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

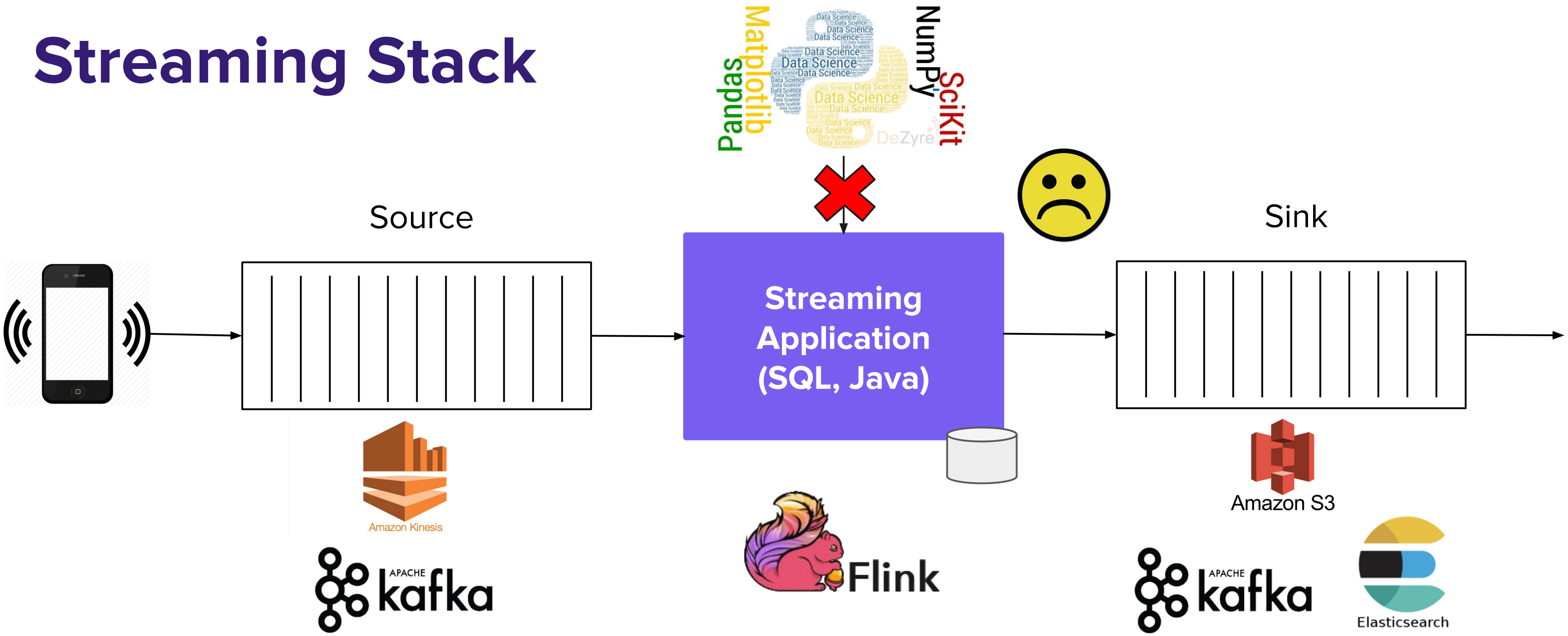
2. Code complexity (LOC)

3. Hard to add new features involving windowing/join (i.e. arbitrary demand windows, subregional computation)

4. No dynamic / smart triggers

Can we use Flink?

Streaming Stack

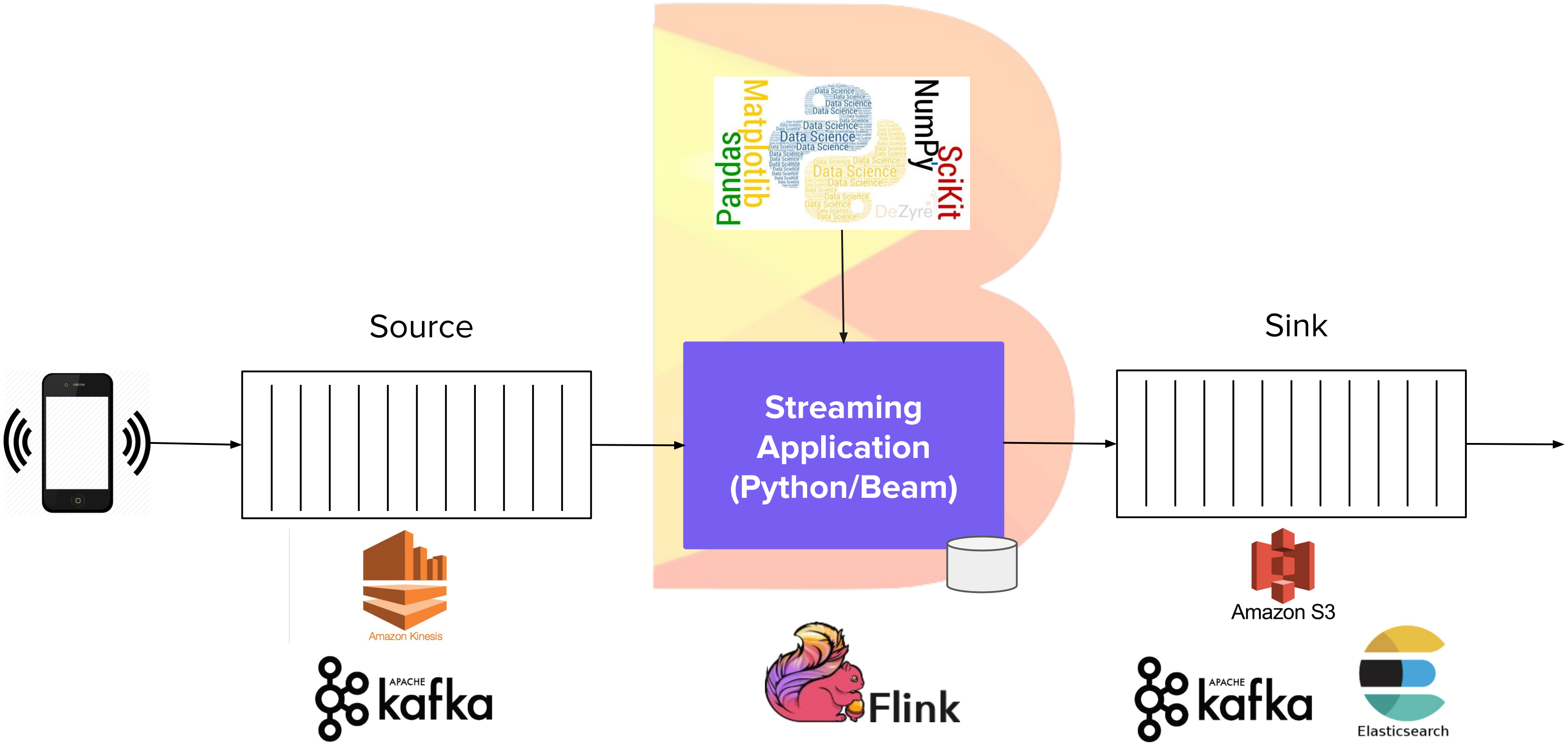


Stream / Schema Registry	**Deployment Tooling**	**Metrics & Dashboards**	**Alerts**	**Logging**
Amazon EC2	Amazon S3	Wavefront	Salt (Config / Orca)	Docker

Streaming and Python

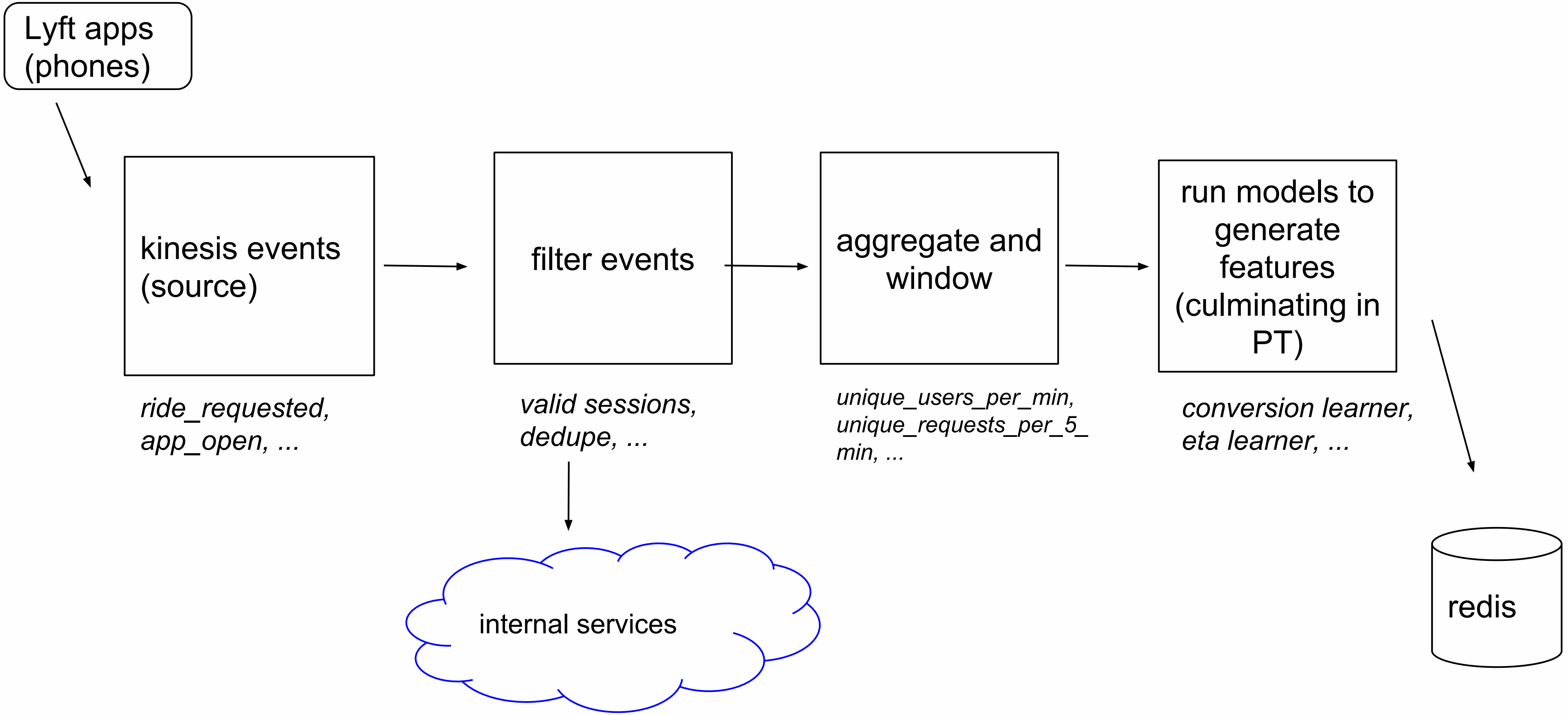
- Flink and many other big data ecosystem projects are Java / JVM based
  - Team wants to adopt streaming, but doesn’t have the Java skills
  - Jython != Python
- Use cases for different language environments
  - Python primary option for Machine Learning
- Cost of many API styles and runtime environments

Solution with Beam



Streaming based Pricing Infrastructure

Pipeline (conceptual outline)



Details of implementation

1. Filtering (with internal service calls)
2. Aggregation with Beam windowing: 1min, 5min (by event time)
3. Triggers: watermark or stateful processing
4. Machine learning models invoked using stateful Beam transforms
5. Final gh6:pt output from pipeline stored to Redis

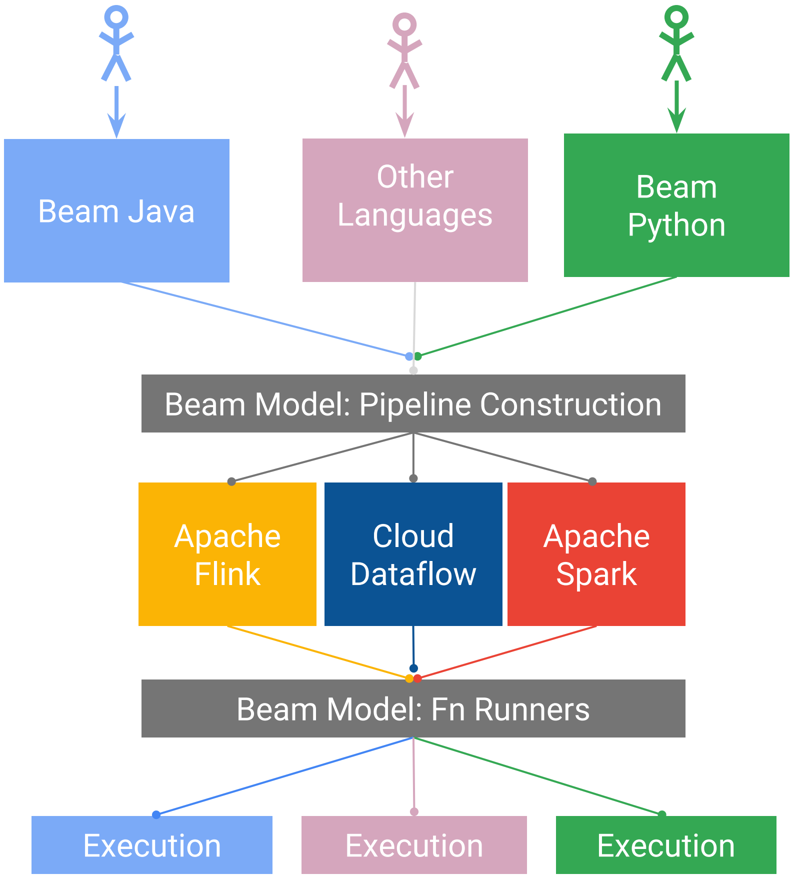
Gains

- 60% reduction in latency
- Reuse of model code
- 10K => 4K LOC
- 300 => 120 AWS instances

Beam and multiple languages

The Beam Vision

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

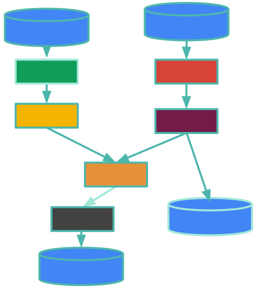


https://s.apache.org/apache-beam-project-overview

Multi-Language Support

- Initially Java SDK and Java Runners
- 2016: Start of cross-language support effort
- 2017: Python SDK on Dataflow
- 2018: Go SDK (for portable runners)
- 2018: Python on Flink MVP
- Next: Cross-language pipelines, more portable runners

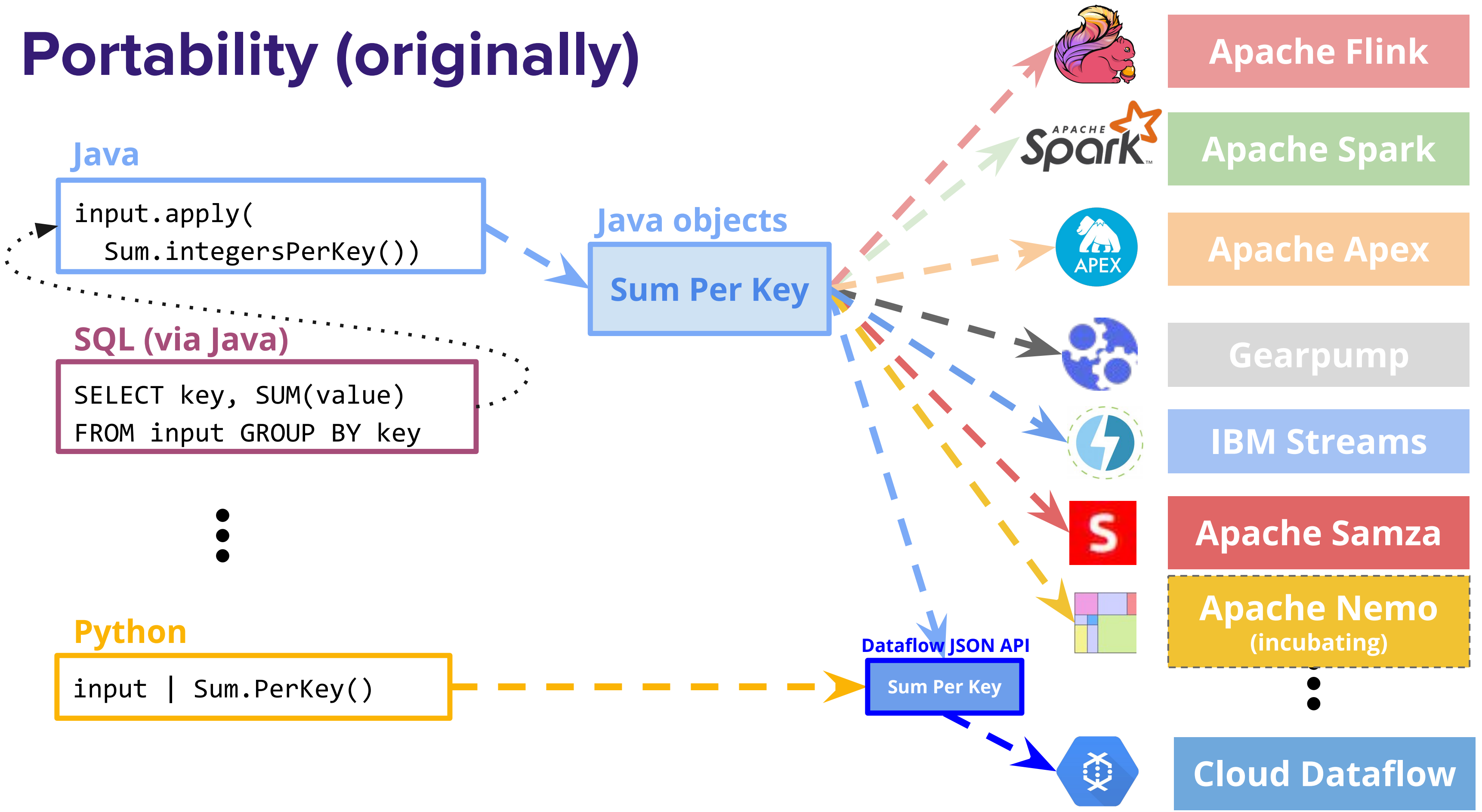
Python Example



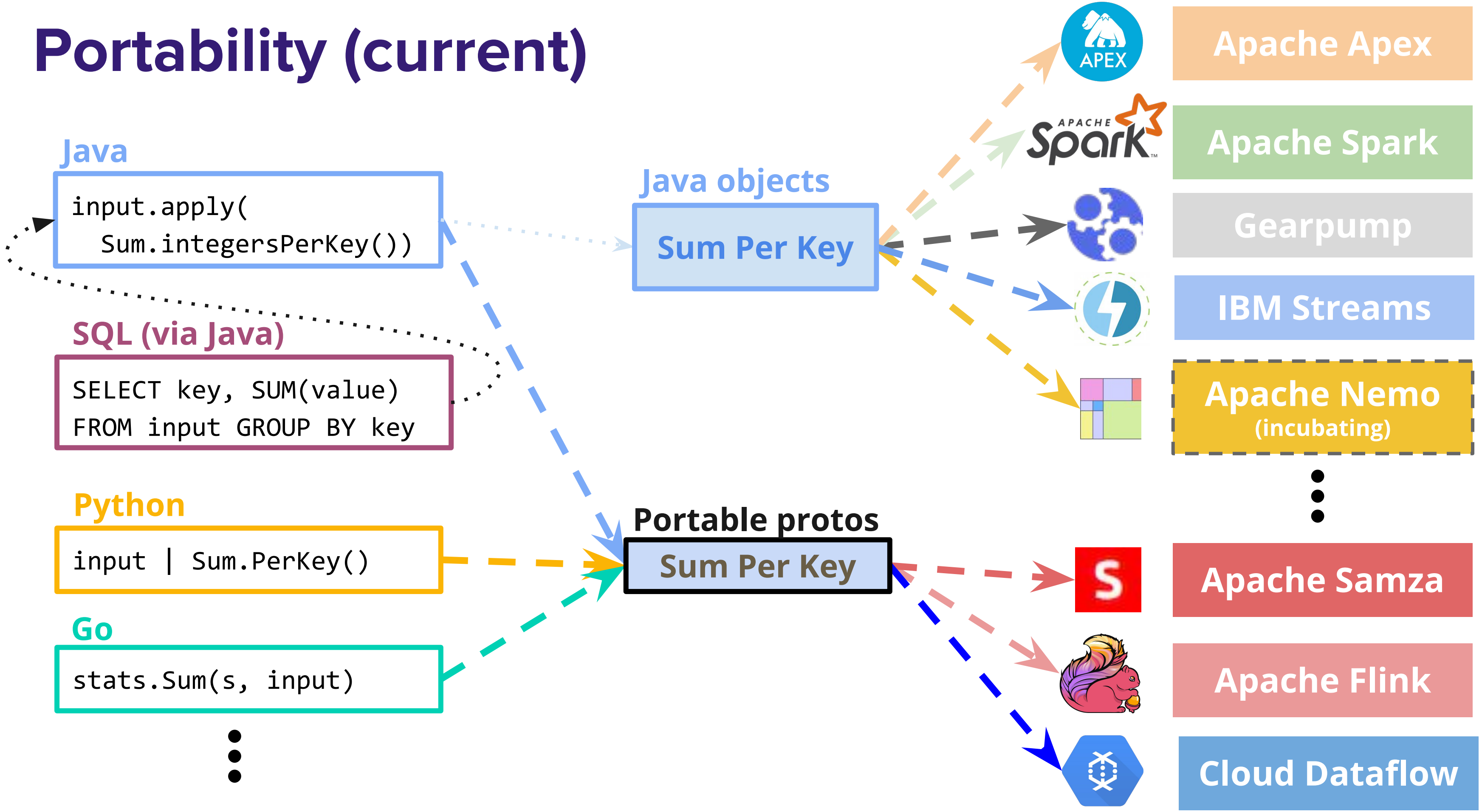
```
p = beam.Pipeline(runner=runner, options=pipeline_options)
(p
   | ReadFromText("/path/to/text*") | Map(lambda line: ...)
   | WindowInto(FixedWindows(120)
                trigger=AfterWatermark(
                    early=AfterProcessingTime(60),
                    late=AfterCount(1))
         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

(What, Where, When, How)

Portability (originally)



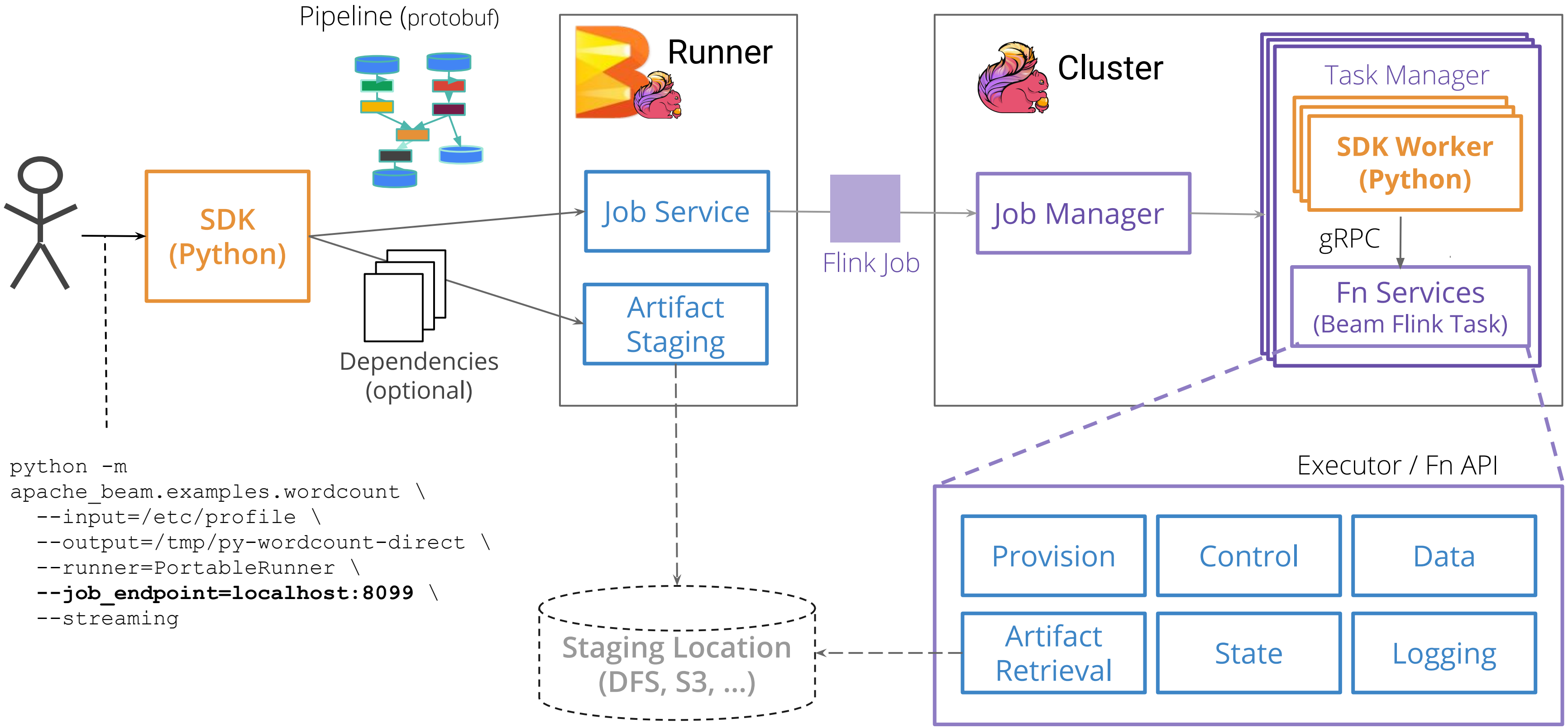
Portability (current)



https://s.apache.org/state-of-beam-sfo-2018

Beam Flink Runner

Portability Framework w/ Flink Runner




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

Portable Runner

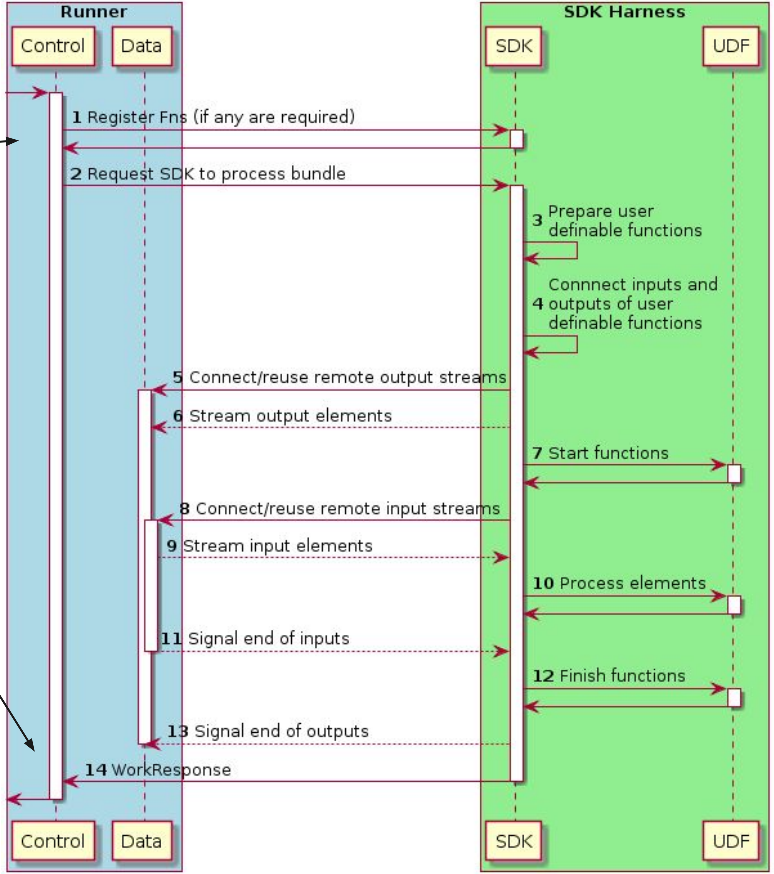
- Provide Job Service endpoint (Job Management API)
- Translate portable pipeline representation to native (Flink) API
- Provide gRPC endpoints for control/data/logging/state plane
- Manage SDK worker processes that execute user code
- Manage bundle execution (with arbitrary user code) via Fn API
- Manage state for side inputs, user state and timers

Common implementation for JVM based runners (/runners/java-fn-execution) and portable “Validate Runner” integration test suite in Python!

Fn API - Bundle Processing

Bundle size matters!

- Amortize overhead over many elements
- Watermark hold effect on latency

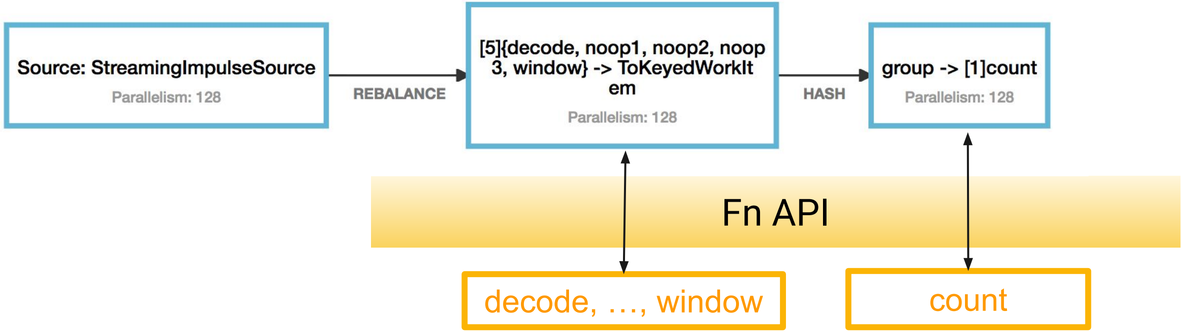


https://s.apache.org/beam-fn-api-processing-a-bundle

Lyft Flink Runner Customizations

- Translator extension for streaming sources
  - Kinesis, Kafka consumers that we also use in Java Flink jobs
  - Message decoding, watermarks
- Python execution environment for SDK workers
  - Tailored to internal deployment tooling
  - Docker-free, frozen virtual envs
- https://github.com/lyft/beam/tree/release-2.11.0-lyft

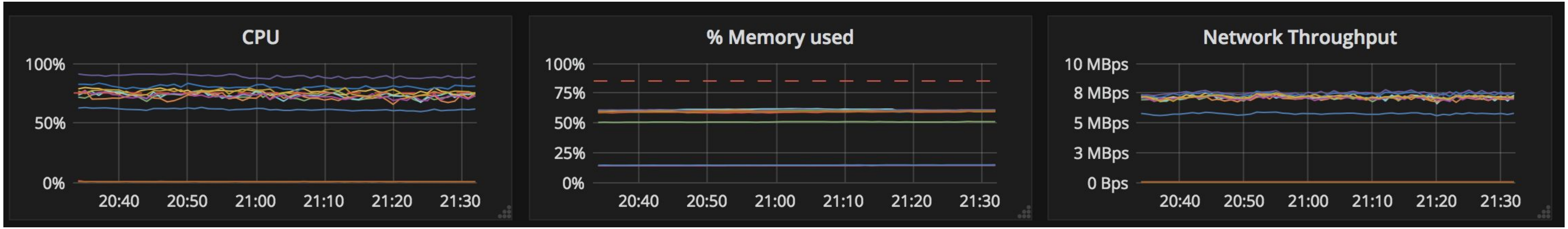
How slow is this ?



```
(messages
     | 'reshuffle' >> beam.Reshuffle()
     | 'decode' >> beam.Map(lambda x: (__import__('random').randint(0, 511), 1))
     | 'noop1' >> beam.Map(lambda x : x)
     | 'noop2' >> beam.Map(lambda x : x)
     | 'noop3' >> beam.Map(lambda x : x)
     | 'window' >> beam.WindowInto(window.GlobalWindows(),
                          trigger=Repeatedly(AfterProcessingTime(5 * 1000)),
                          accumulation_mode= AccumulationMode.DISCARDING)
     | 'group' >> beam.GroupByKey()
     | 'count' >> beam.Map(count)
 )
```

- Fn API Overhead 15% ?
- Fused stages
- Bundle size
- Parallel SDK workers
- TODO: Cython, protobuf C++ bindings

Fast enough for real Python work !



- c5.4xlarge machines (16 vCPU, 32 GB)
- 16 SDK workers / machine
- 1000 ms or 1000 records / bundle
- 280,000 transforms / second / machine (~ 17,500 per worker)
- **Python user code will be gating factor**

Beam Portability Recap

- Pipelines written in non-JVM languages on JVM runners
  - Python, Go on Flink (and others)
- Full isolation of user code
  - Native CPython execution w/o library restrictions
- Configurable SDK worker execution
  - Docker, Process, Embedded, ...
- Multiple languages in a single pipeline (future)
  - Use Java Beam IO with Python
  - Use TFX with Java
  - <your use case here>

Feature Support Matrix (Beam 2.11.0)

			Flink (master)	instructions					Dataflow				
			Java		Python		Go		Java		Python		Go
FEATURE			Batch	Streaming	Batch	Streaming	Batch	Streaming	Batch	Streaming	Batch	Streaming	Batch
	Impulse												
	ParDo												
		w/ side input					BEAM-3286	BEAM-3286					BEAM-3286
		w/ multiple output											
		w/ user state	M-3298				BEAM-2918/BEA	BEAM-2918/BEA	BEAM-2902/BEA	BEAM-2902/BEA	BEAM-2902/BEA	BEAM-2902/BEA	BEAM-2902/E
		w/ user timers											
		w/ user metrics											
	Flatten												
		w/ explicit flatten					BEAM-3300	BEAM-3300					BEAM-3300
	Combine												
		w/ first-class rep					BEAM-4276	BEAM-4276	BEAM-3513	BEAM-3513			BEAM-4276
		w/ lifting					BEAM-4276	BEAM-4276	BEAM-3711	BEAM-3711			BEAM-4276
	SDF						BEAM-3301	BEAM-3301					BEAM-3301
		w/ liquid sharding											
	GBK												
	CoGBK												
	WindowInto												
		w/ sessions					BEAM-4152	BEAM-4152					BEAM-4152
		w/ custom windowfn											
EXAMPLE			Batch	Streaming	Batch	Streaming	Batch	Streaming	Batch	Streaming	Batch	Streaming	Batch
	WordCap												
	WordCount												
		w/ write to Sink											
		w/ write to GCS											

https://s.apache.org/apache-beam-portability-support-table

Lessons Learned

Lessons Learned

- Python Beam SDK and portable Flink runner evolving
- Keep pipeline simple - Flink tasks / shuffles are not free
- Stateful processing is essential for complex logic
- Model execution latency matters
- Instrument everything for monitoring
- Approach for pipeline upgrade and restart
- Mind your dependencies - rate limit API calls
- Testing story (integration, staging)

We're Hiring! Apply at www.lyft.com/careers
or email data-recruiting@lyft.com

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

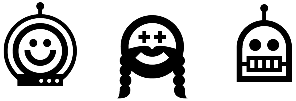
Observability

Software Engineer
San Francisco

Streaming

Software Engineer
San Francisco

Please ask questions!



This presentation:

http://go.lyft.com/dynamic-pricing-strata-sf-2019