Cloud Native Data Pipelines

Sid Anand (@r39132)

GoTo Chicago 2017

About Me

Work [ed | s] @

Co-Chair for

Committer & PPMC on

Father of 2

Apache Airflow



Agari

What We Do!



Agari : What We Do



Agari : What We Do



Agari : What We Do





Agari : What We Do



AGARI



Agari : What We Do

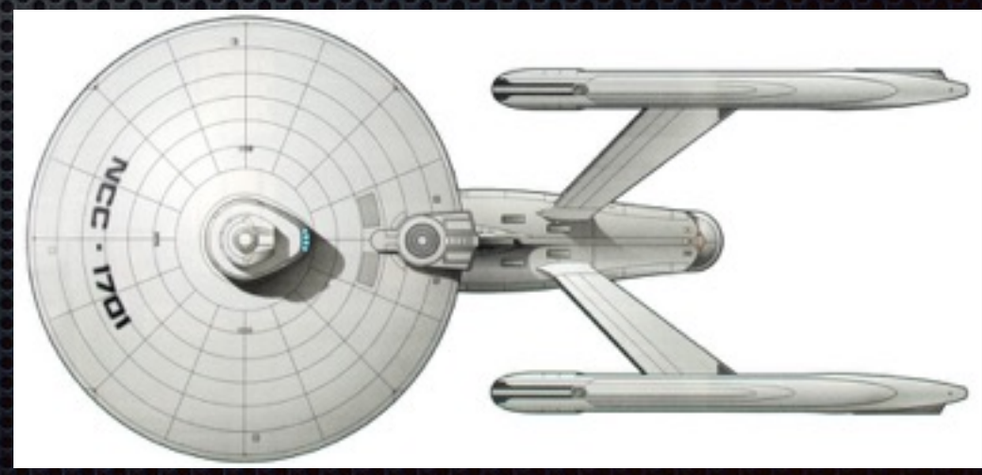




AGARI

AGARI





Agari : What We Do



Enterprise Customers

Batch

email metadata

Agari's Previous EP Version

apply trust models

email md + trust score



Agari : What We Do



Quarantine,
Label,
PassThrough

Enterprise Customers

Near-real time

email metadata

Agari's Current EP Version

apply trust models

email md + trust score

Motivation

Cloud Native Data Pipelines

Cloud Native Data Pipelines

Big Data Companies like LinkedIn, Facebook, Twitter, & Google have large teams to manage their data pipelines (100s of engineers)

Most start-ups have small teams (10s of engineers) & run in the public cloud. Can they leverage aspects of the public cloud to build comparable pipelines?

Cloud Native Data Pipelines

Cloud Native
Techniques

+

Open Source
Technogies

~ Data Pipelines seen in Big Data companies

Design Goals

Desirable Qualities of a Resilient Data Pipeline

Desirable Qualities of a Resilient Data Pipeline

Correctness

Operability

Timeliness

Cost

Desirable Qualities of a Resilient Data Pipeline

Correctness

- Data Integrity (no loss, etc…)
- Expected data distributions

Operability

- Minimize Operational Fatigue / Automate Everything
- Fine-grained Monitoring & Alerting of Correctness & Timeliness SLAs
- Quick Recoverability

Timeliness

- All output within time-bound SLAs

Cost

- Pay-as-you-go

Predictive Analytics @ Agari

Use Cases

Use Cases

Focus of this talk

(Enterprise Protect)



Build trust models

batch

Apply trust models (message scoring)

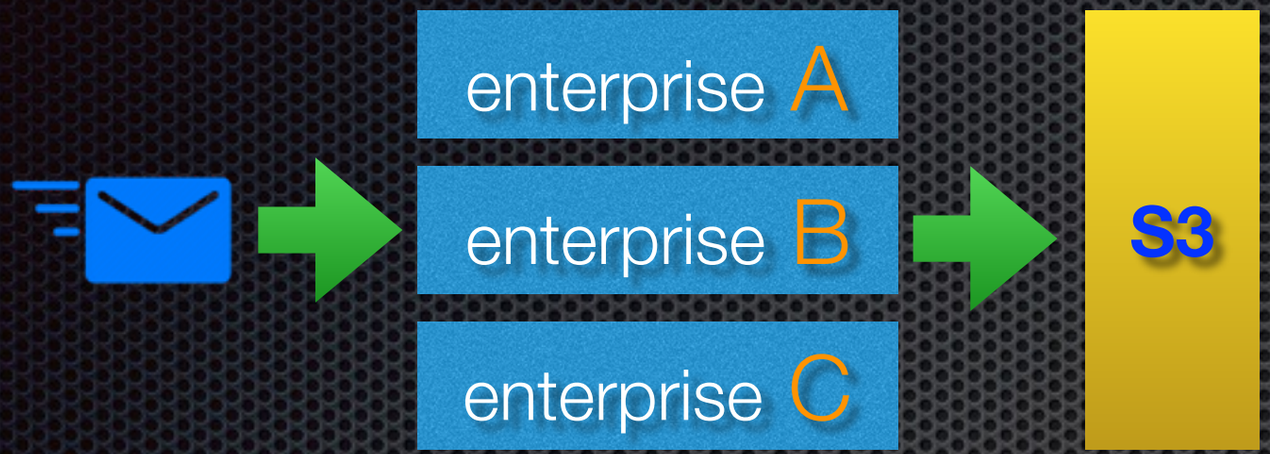
batch + near real time

Use-Case : Message Scoring (batch)

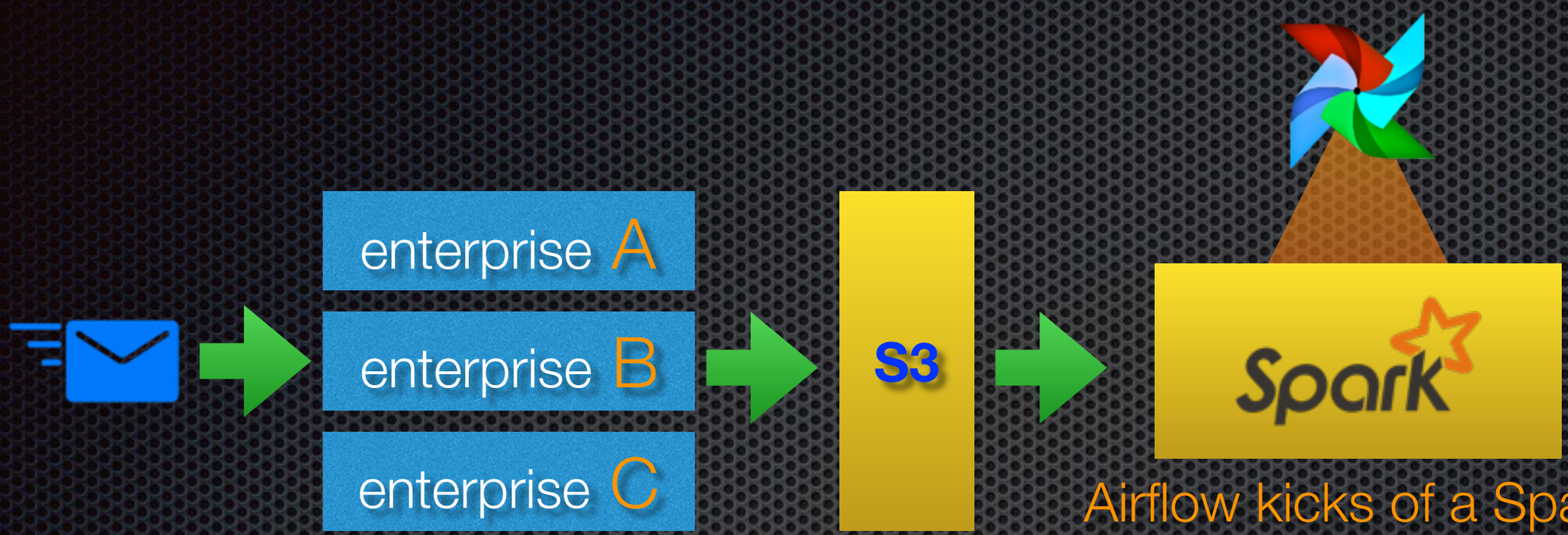
Batch Pipeline Architecture

Use-Case : Message Scoring

S3 uploads an Avro file
every 15 minutes

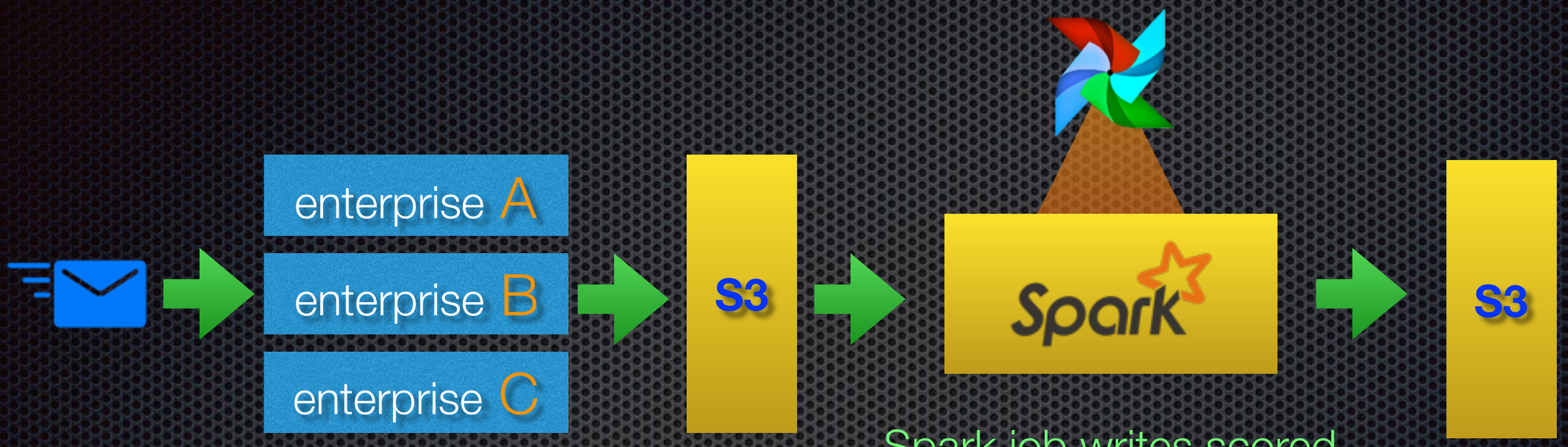


Use-Case : Message Scoring



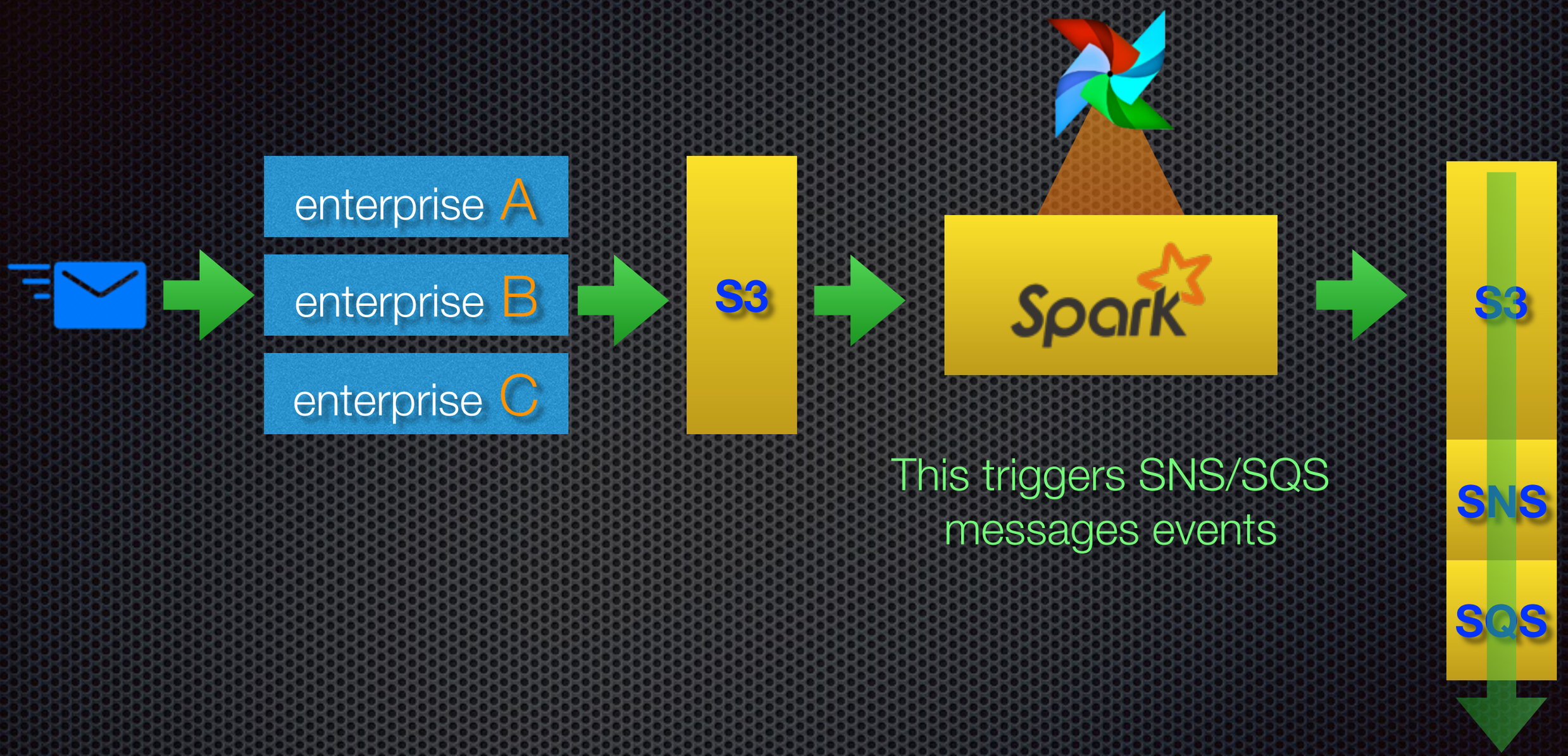
Airflow kicks of a Spark message scoring job every hour (EMR)

Use-Case : Message Scoring



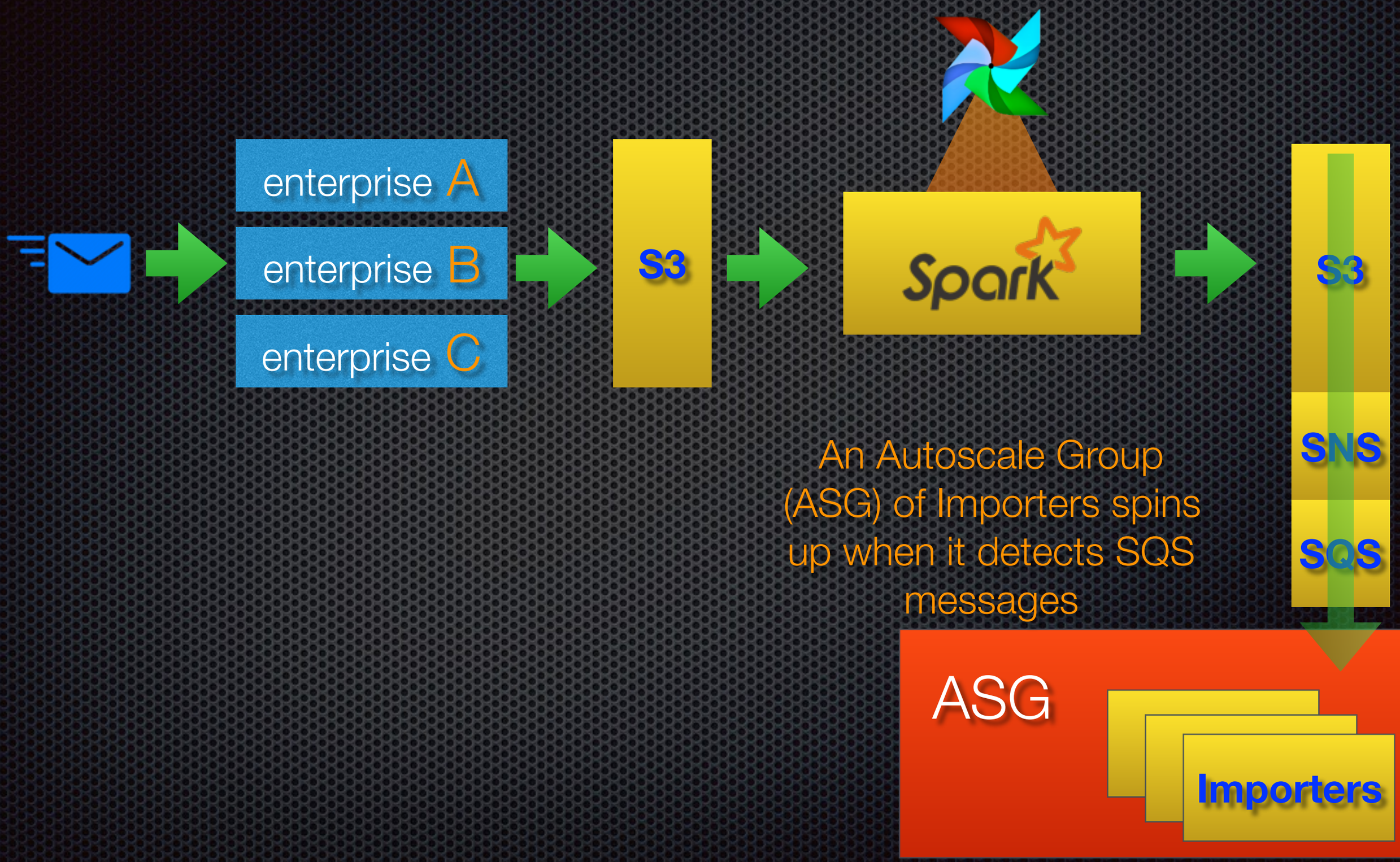
Spark job writes scored messages and stats to another S3 bucket

Use-Case : Message Scoring

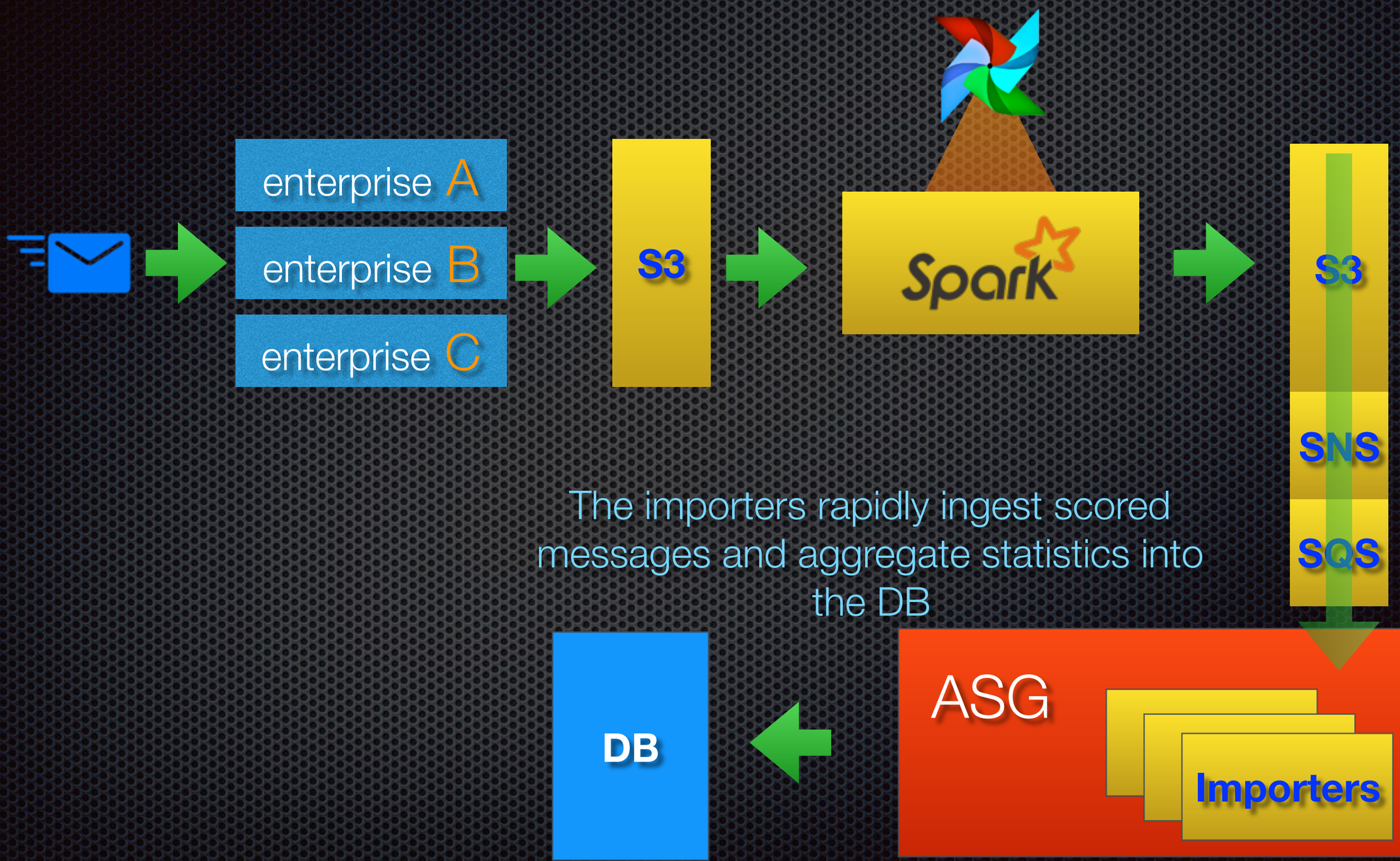


This triggers SNS/SQS messages events

Use-Case : Message Scoring

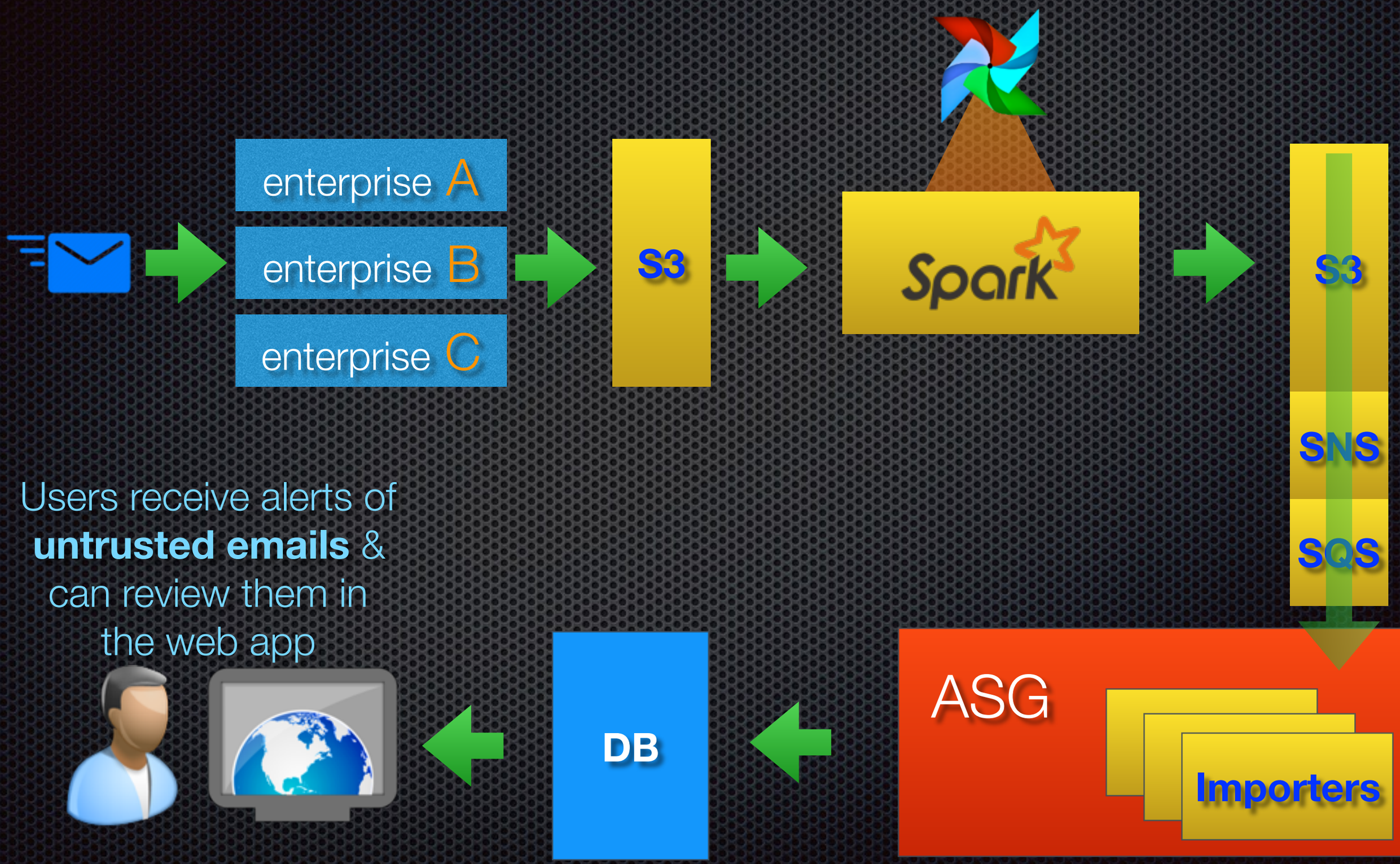


Use-Case : Message Scoring



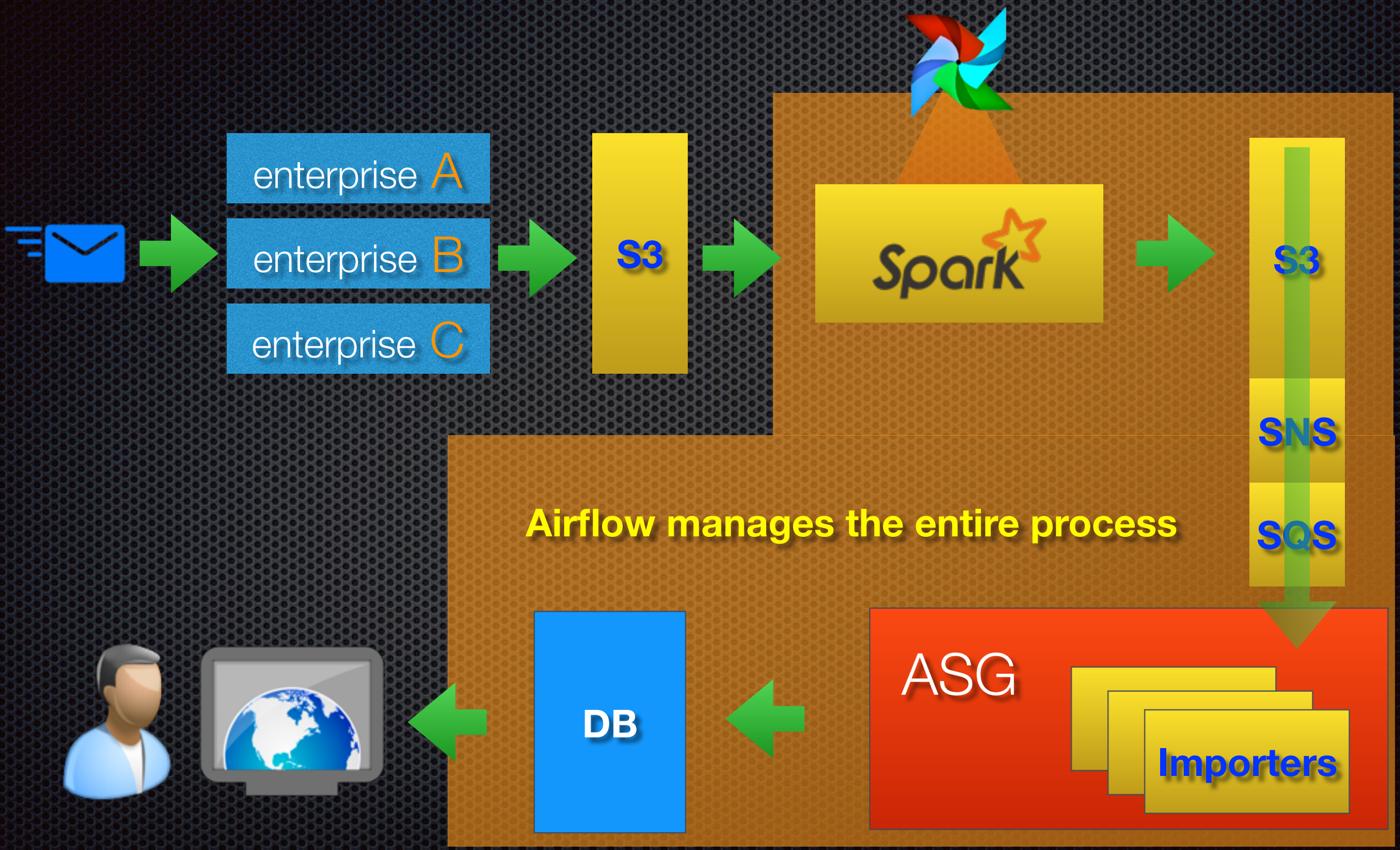
The importers rapidly ingest scored messages and aggregate statistics into the DB

Use-Case : Message Scoring



Users receive alerts of **untrusted emails** & can review them in the web app

Use-Case : Message Scoring



Architectural Components

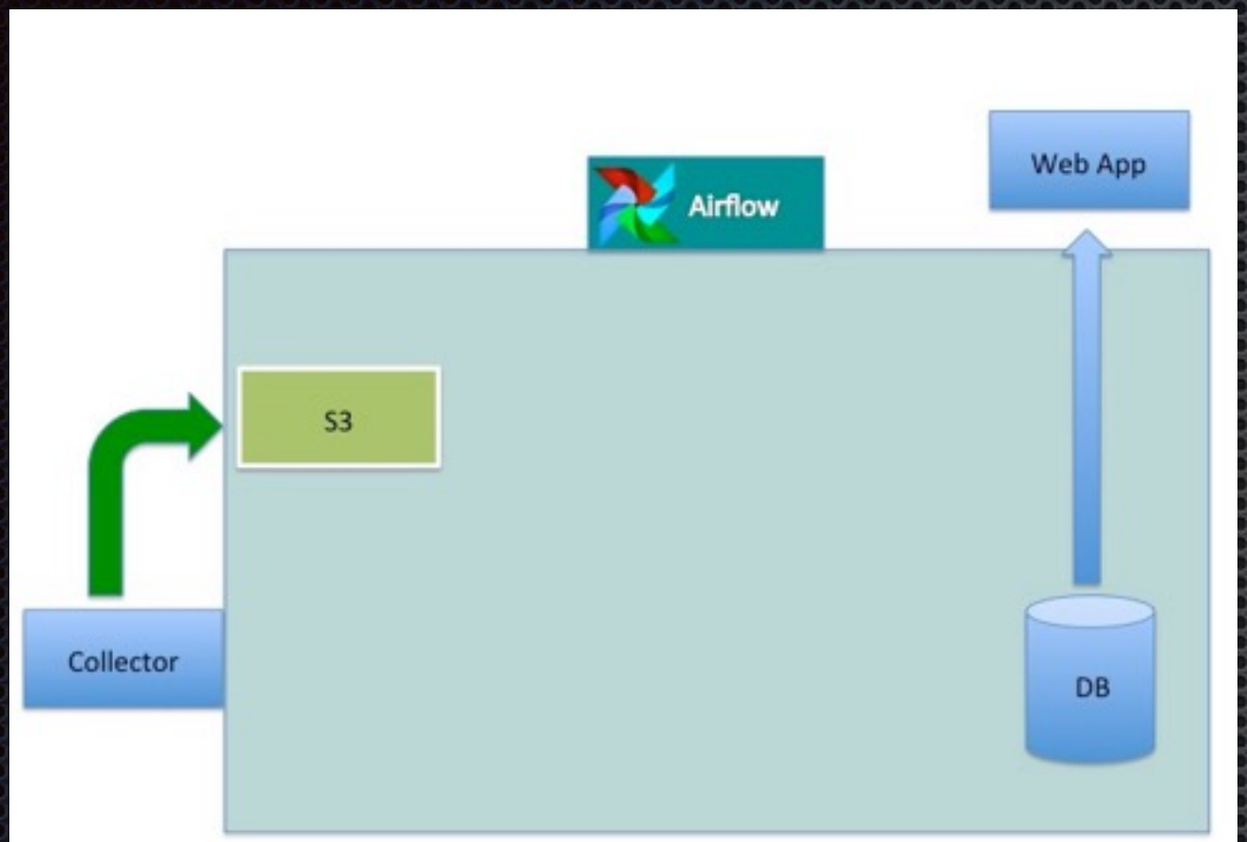
Component	Role	Uses	Salient Features	Operability Model
S3	Data Lake	• All data stored in S3 • All processing uses S3	Scalable, Available, Performant	**Serverless**
SNS ⊕ SQS	Messaging	• Reliable, Transactional, Pub/Sub	Scalable, Available, Performant	**Serverless**
ASG	General Processing	• Used for importing, data cleansing, business logic	Scalable, Available, Performant	**Managed**
Spark	Data Science Processing	• Aggregation • Model Building • Scoring	Nice programming model at the cost of debugging complexity	**We Operate**
	Workflow Engine	• Coordinates all Spark Jobs & complex flows	Lightweight, DAGs as Code, Steep learning curve	**We Operate**
DB PostgreSQL	Persistence for WebApp	• Holds subset of data needed for Web App	Rails + Postgres ‘nuff said	**We Operate**

Tackling Cost & Timeliness

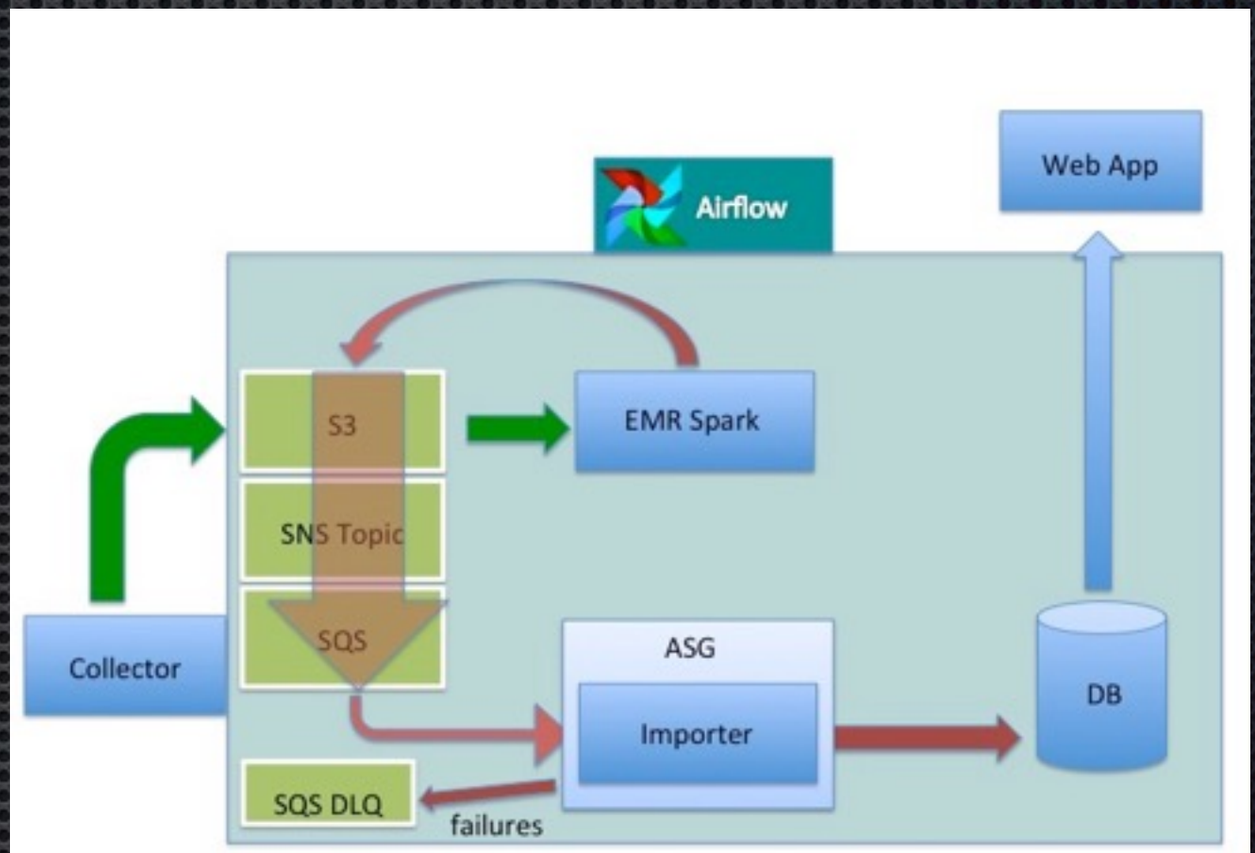
Leveraging the AWS Cloud

Tackling Cost

Between Daily Runs



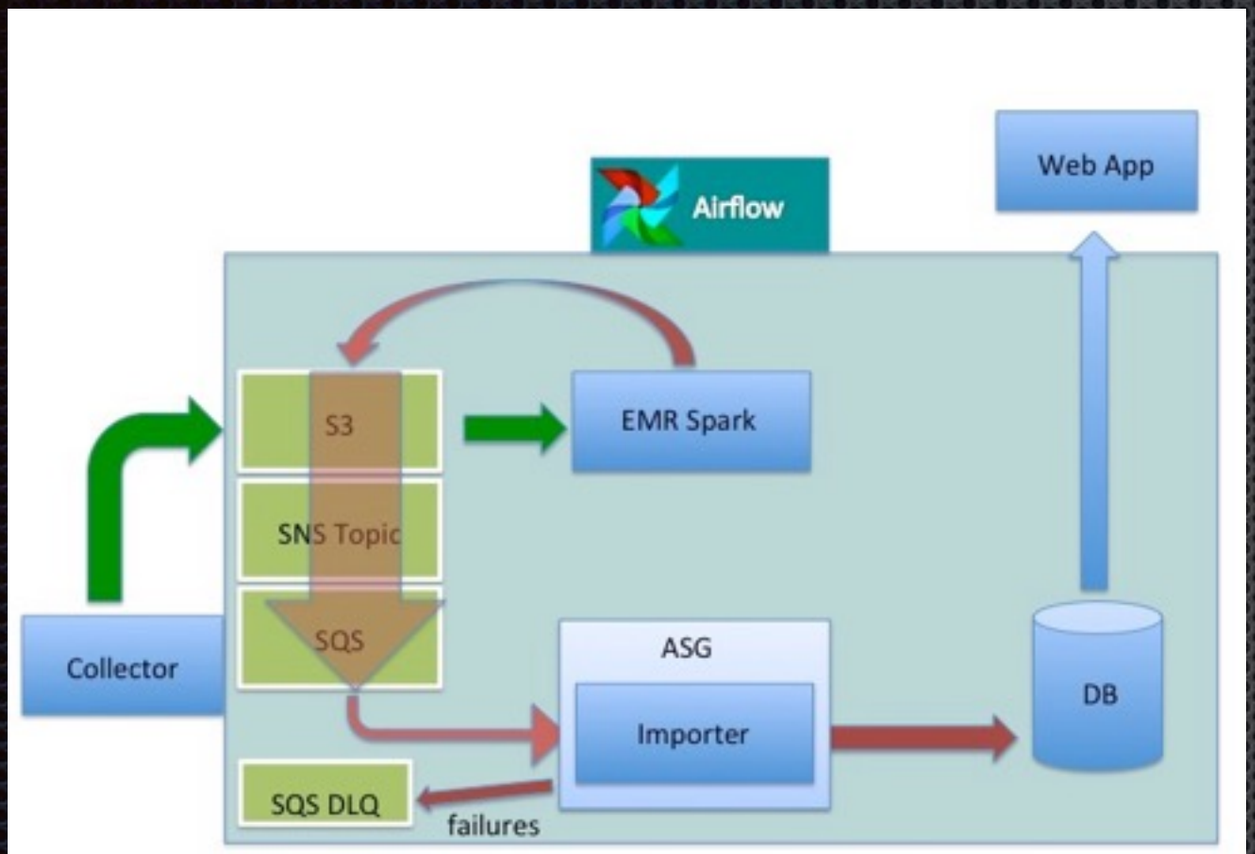
During Daily Runs



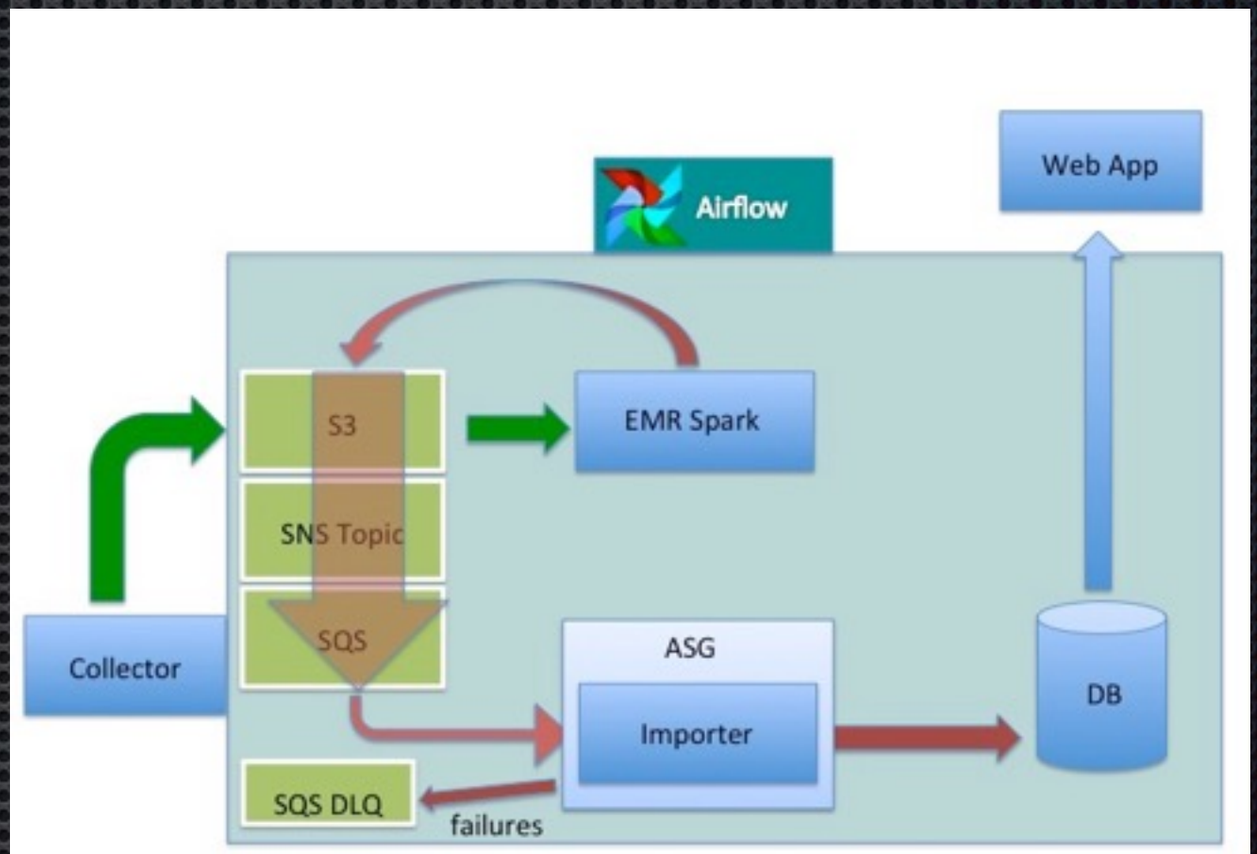
- When running daily, for 23 hours of a day, we didn't pay for instances in the ASG or EMR

Tackling Cost

Between Hourly Runs



During Hourly Runs



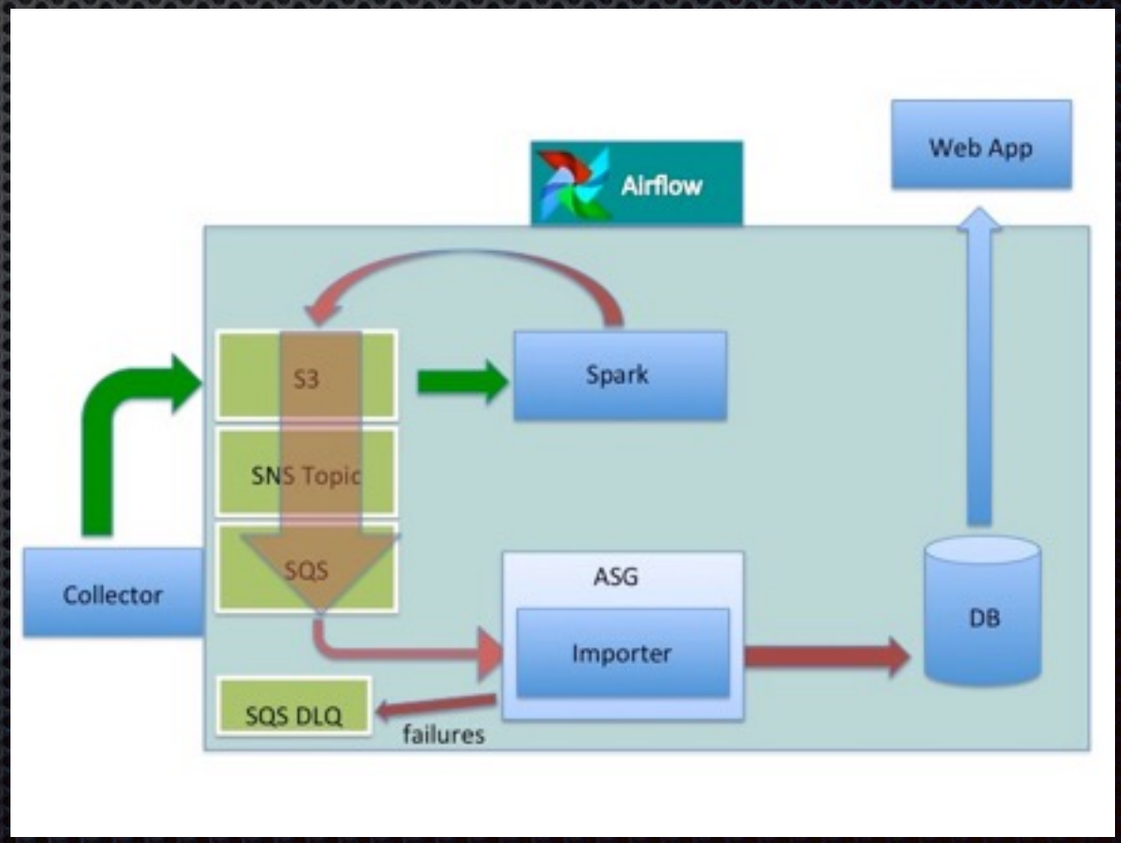
- When running daily, for 23 hours of a day, we didn't pay for instances in the ASG or EMR
- This does not help when runs are hourly since AWS charges at an hourly rate for EC2 instances!

Tackling Timeliness

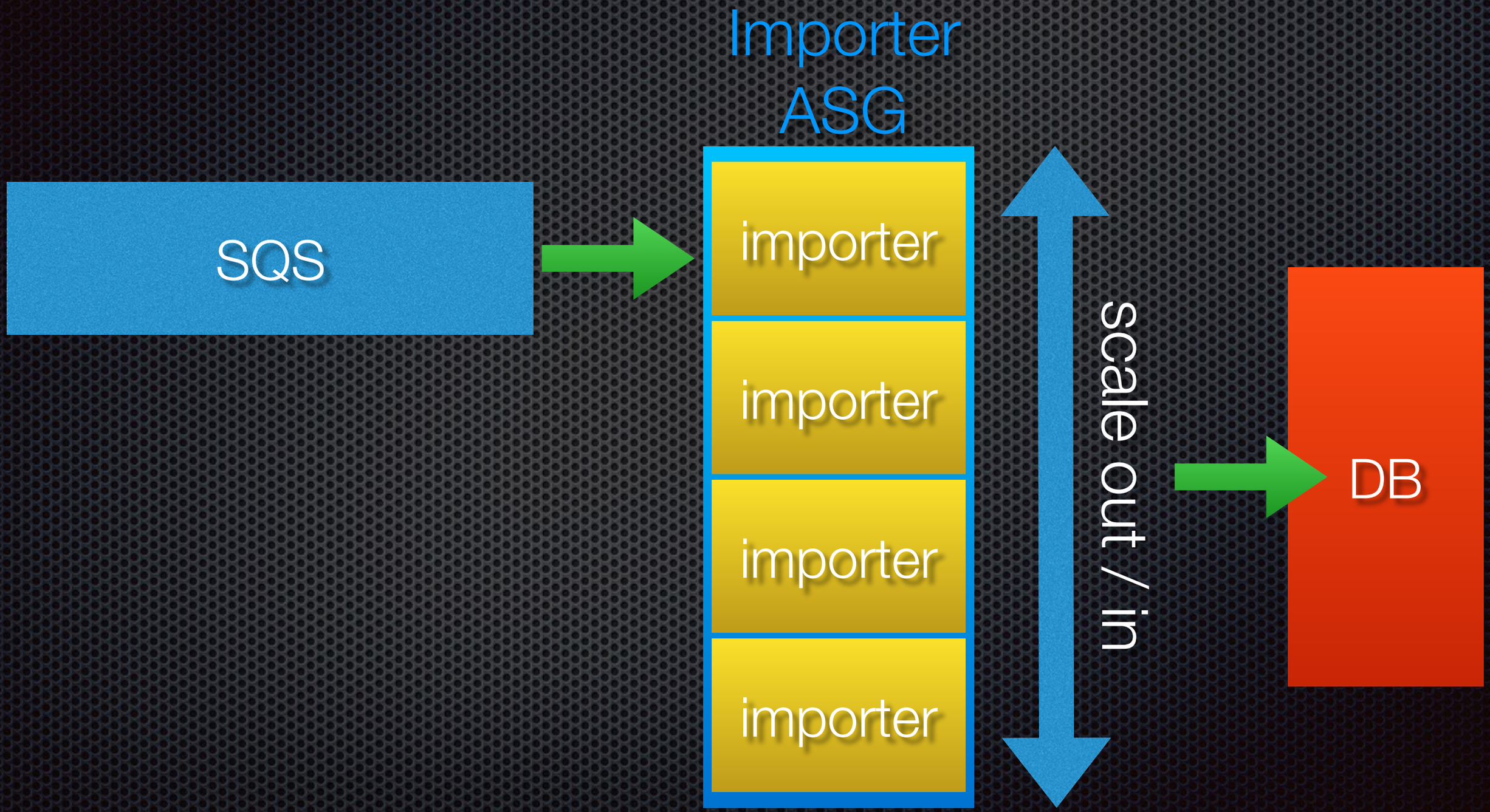
Auto Scaling Group (ASG)

ASG - Overview

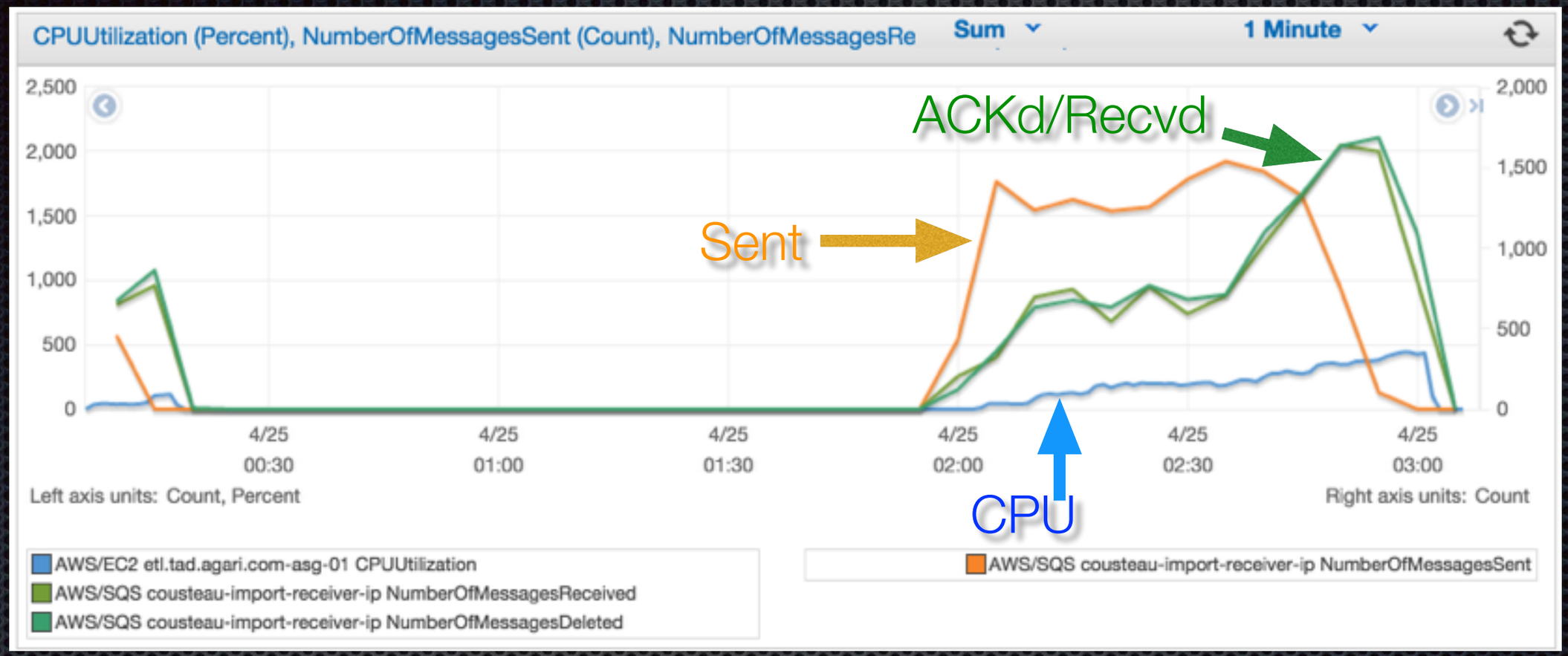
- What is it?
  - A means to automatically scale out/in clusters to handle variable load/traffic
  - A means to keep a cluster/service of a fixed size always up

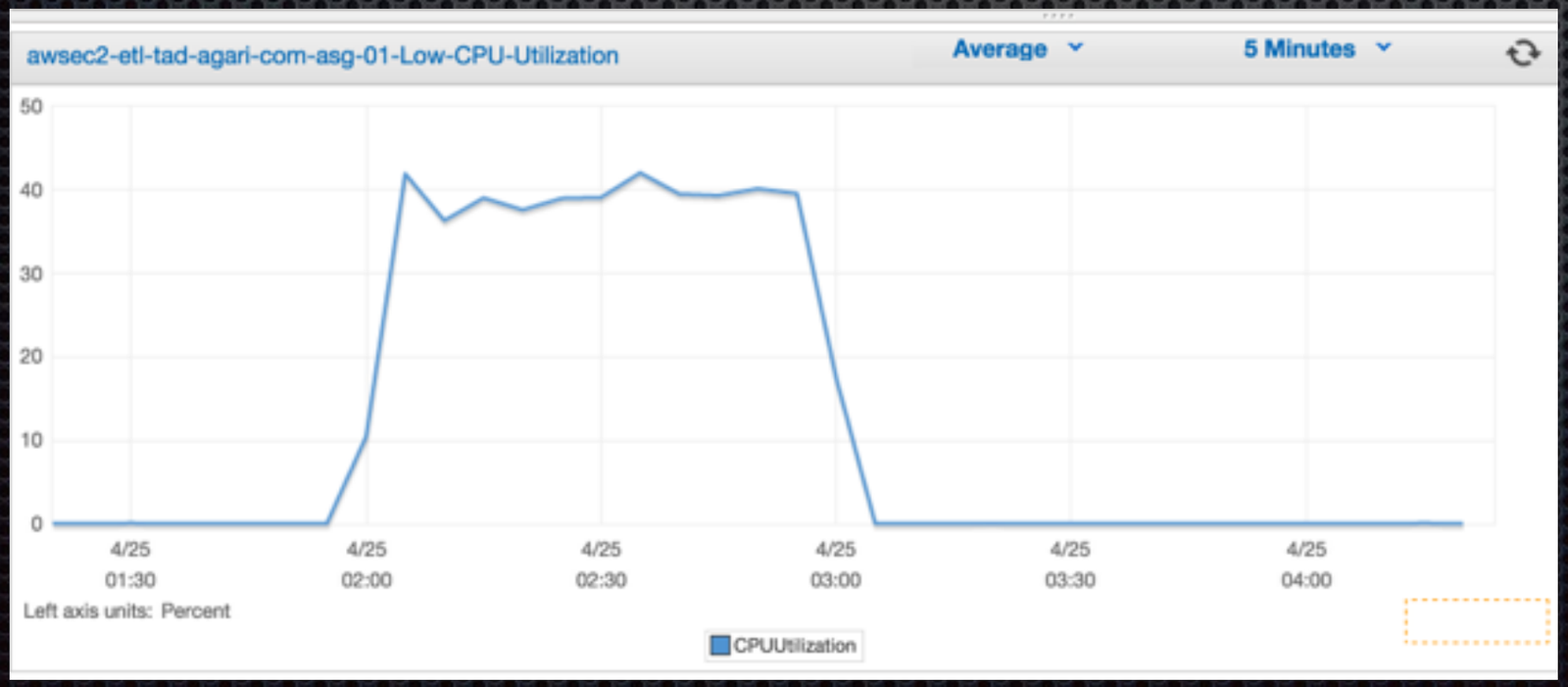


ASG - Data Pipeline



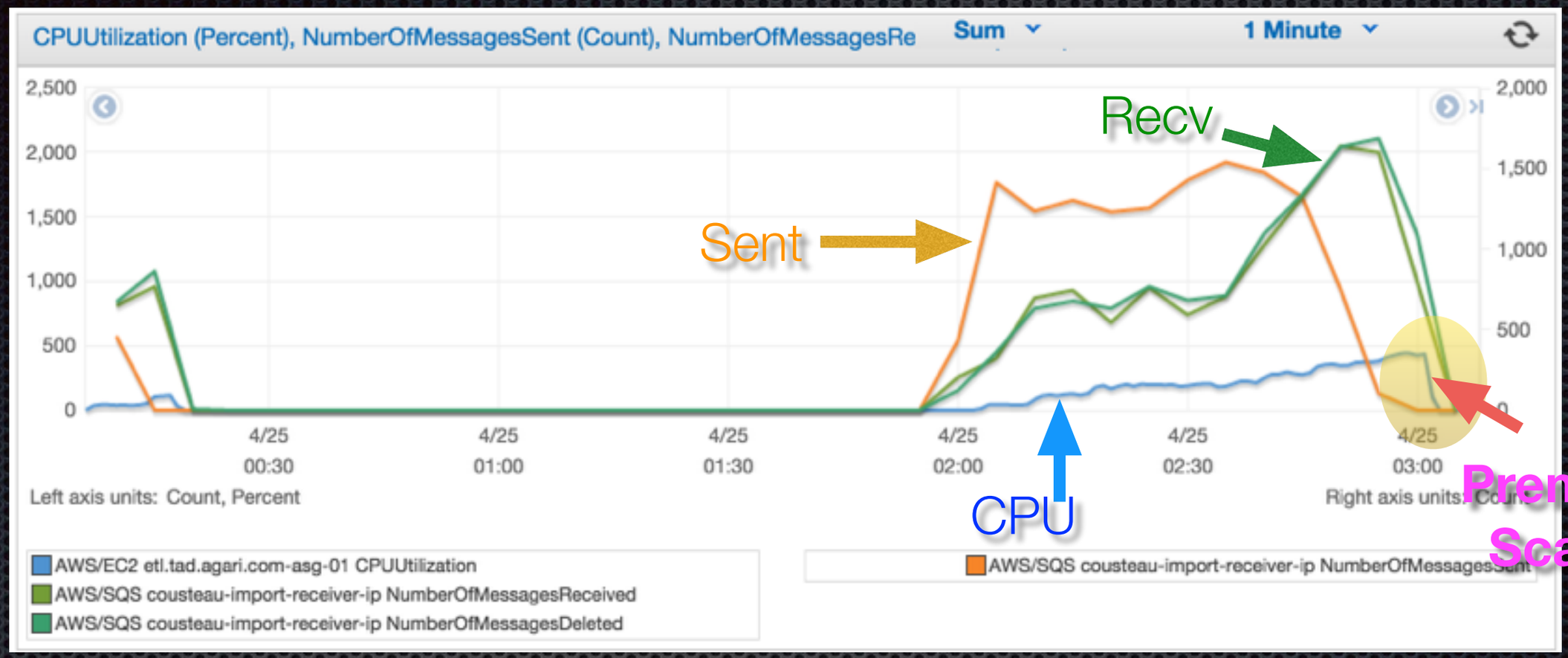
ASG : **CPU**-based





CPU-based auto-scaling is good at scaling in/out to keep the average CPU constant

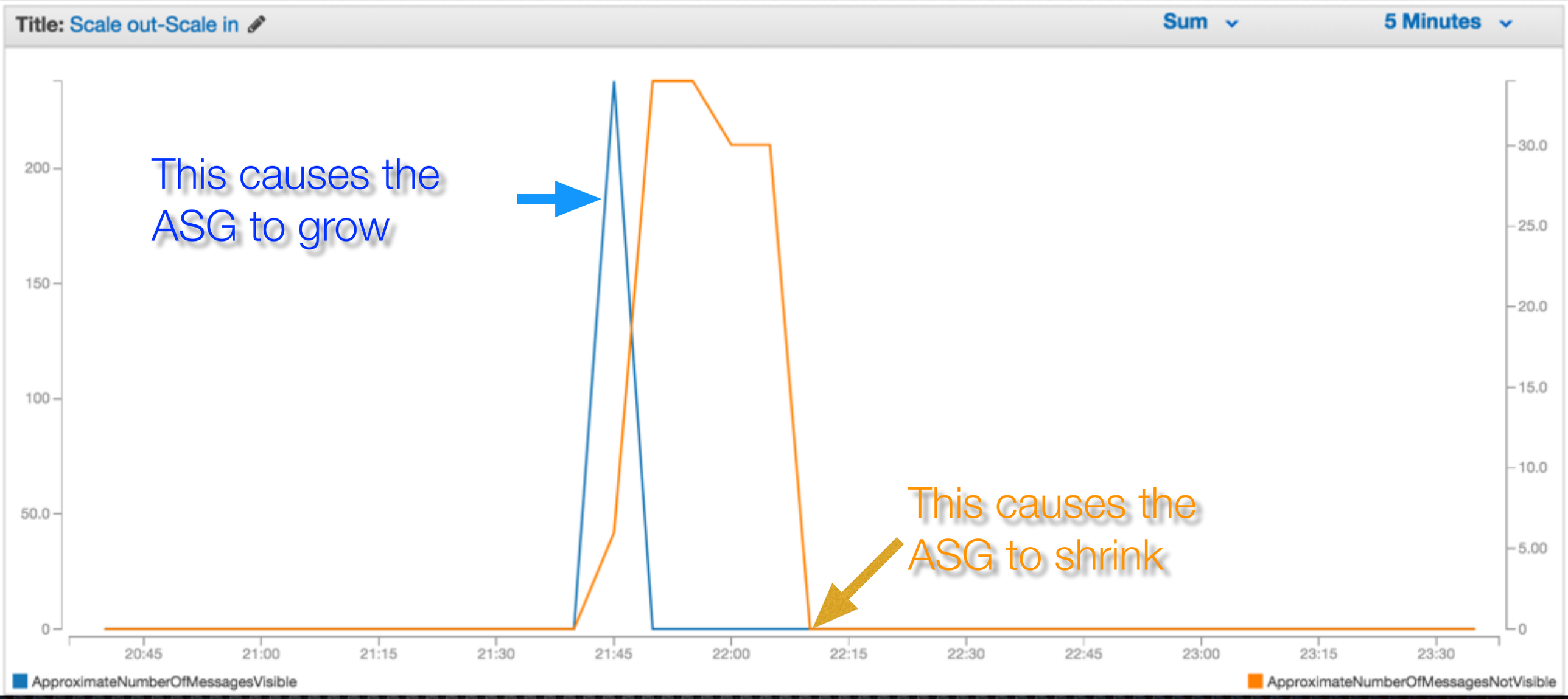
ASG : **CPU**-based



Premature Scale-in:

- The CPU drops to noise-levels before all messages are consumed
- This causes scale in to occur while the last few messages are still being committed

ASG : **Queue**-based



Scale-out: When Visible Messages > 0 (a.k.a. when queue depth > 0)

Scale-in: When Invisible Messages = 0 (a.k.a. when the last in-flight message is ACK'd)

Auto Scaling Groups

Build & Deploy

ASG - Build & Deploy

Component	Role	Details
	Spins up Cloud Resources	• Spins up SQS, Kinesis, EC2, ASG, ELB, etc.. and associate them using Terraform
	• A better version of Chef & Puppet • Sets up an EC2 instance	• Agentless, idempotent, & declarative tool to set up EC2 instances, by installing & configuring packages, and more
	• Spins up an EC2 instance for the purposes of building an AMI!	• Can be used with Ansible & Terraform to bake AMIs & Launch Auto-Scaling Groups

ASG - Build & Deploy

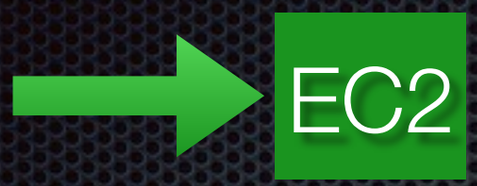




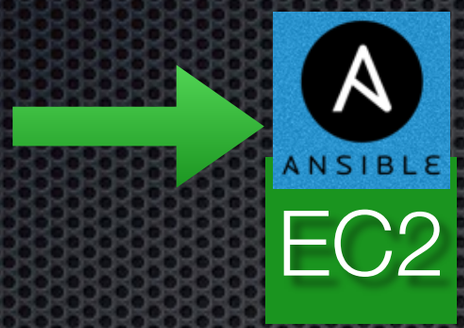
Step 1 : Packer spins up a temporary EC2 node - a blank canvas!

ASG - Build & Deploy





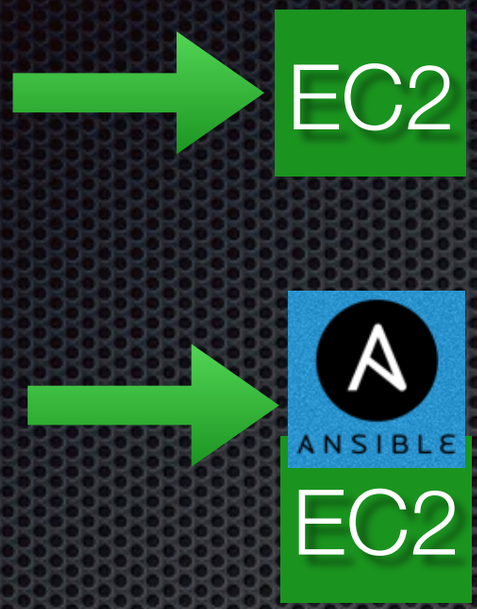
Step 1 : Packer spins up a temporary EC2 node - a blank canvas!



Step 2 : Packer runs an Ansible role against the EC2 node to set it up.

ASG - Build & Deploy





Step 1 : Packer spins up a temporary EC2 node - a blank canvas!

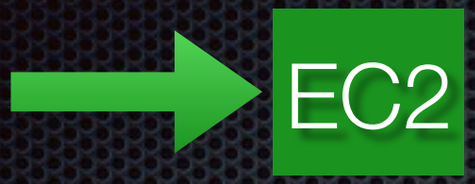
Step 2 : Packer runs an Ansible role against the EC2 node to set it up.



Step 3 : Snapshots the machine & register the AMI.

ASG - Build & Deploy





Step 1 : Packer spins up a temporary EC2 node - a blank canvas!

Step 2 : Packer runs an Ansible role against the EC2 node to set it up.



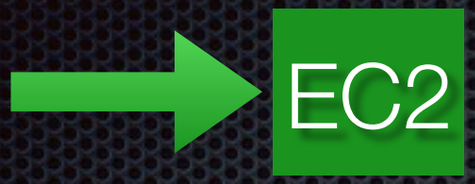
Step 3 : Snapshots the machine & register the AMI.



Step 4 : Terminates the EC2 instance!

ASG - Build & Deploy





Step 1 : Packer spins up a temporary EC2 node - a blank canvas!



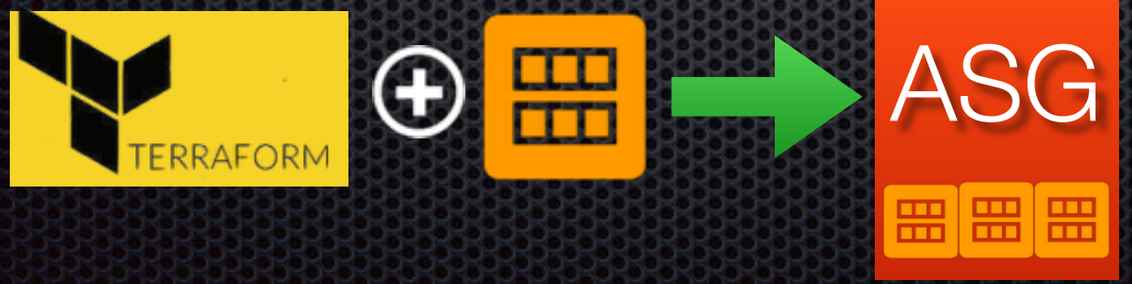
Step 2 : Packer runs an Ansible role against the EC2 node to set it up.



Step 3 : Snapshots the machine & register the AMI.



Step 4 : Terminates the EC2 instance!



Step 5 : Using the AMI, Terraform spins up an auto-scaled compute cluster (ASG)

Desirable Qualities of a Resilient Data Pipeline

Correctness

Operability

Timeliness

- ASG
- EMR Spark

Cost

Daily

- ASG
- EMR Spark

Hourly ASG

- No Cost Savings

Tackling Operability & Correctness

Leveraging Tooling

Tackling Operability : Requirements

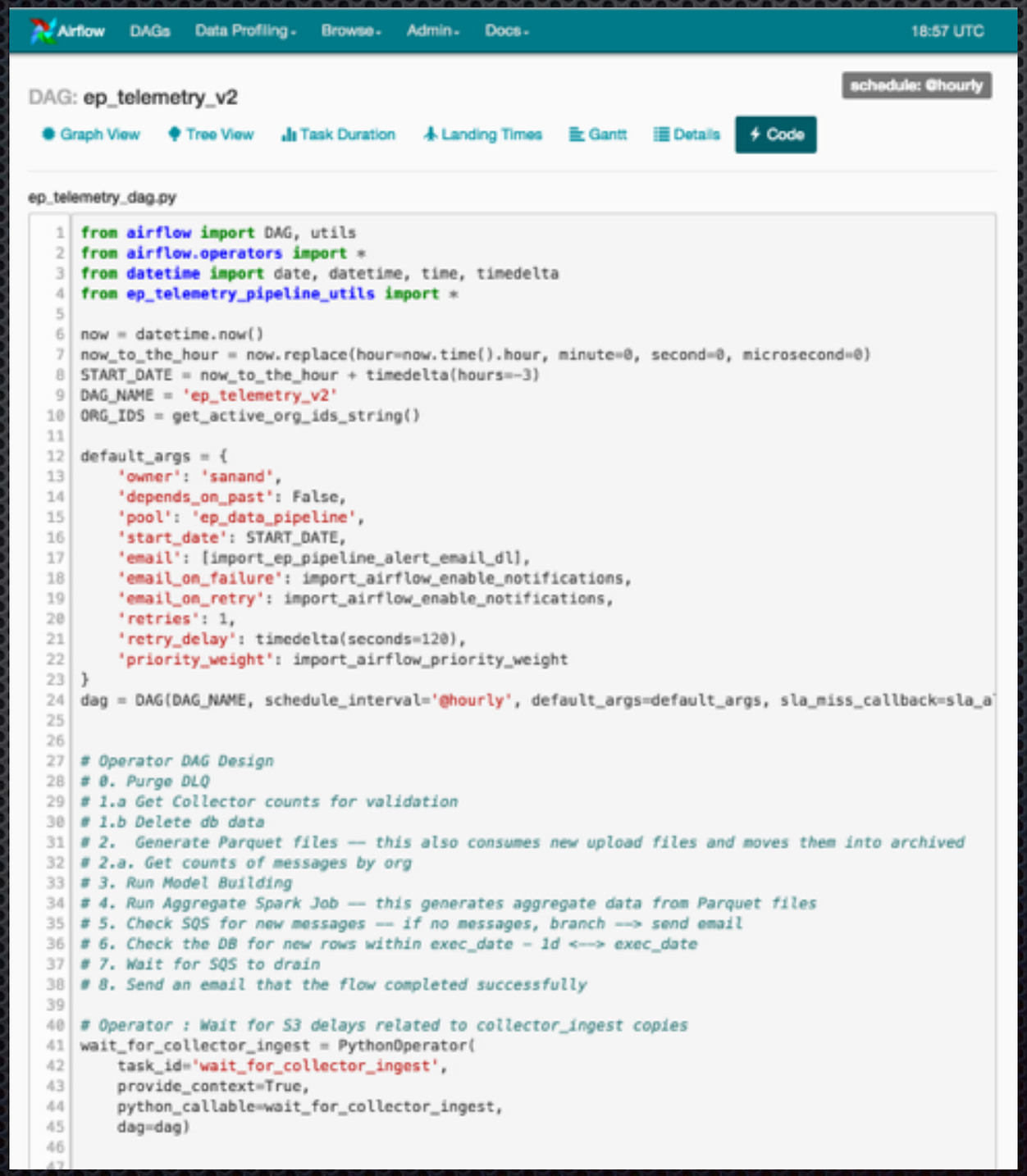
- A simple way to author, configure, manage workflows
- Provides visual insight into the state & performance of workflow runs
- Integrates with our alerting and monitoring tools

Apache Airflow 

Workflow Automation & Scheduling

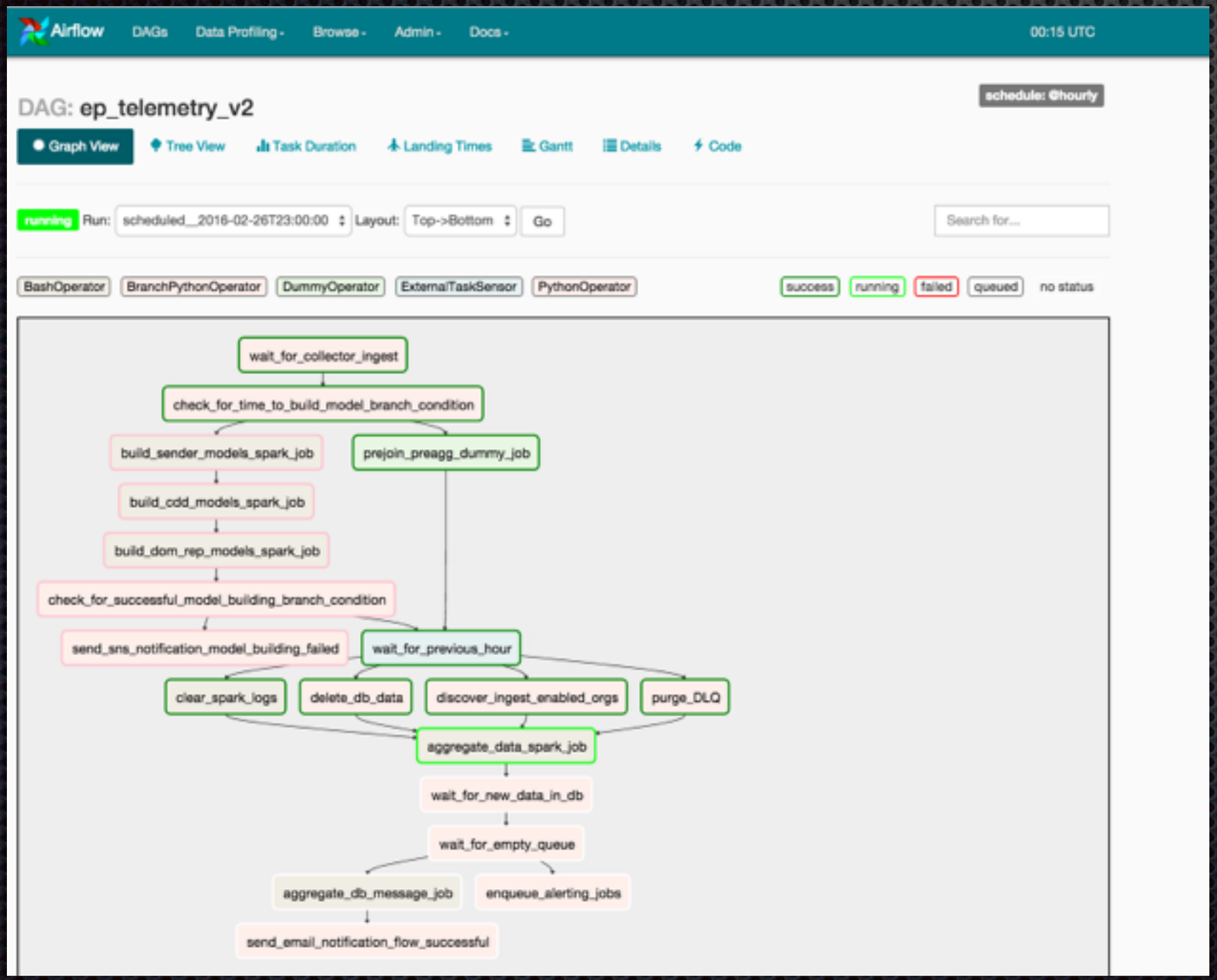
Apache Airflow - Authoring DAGs

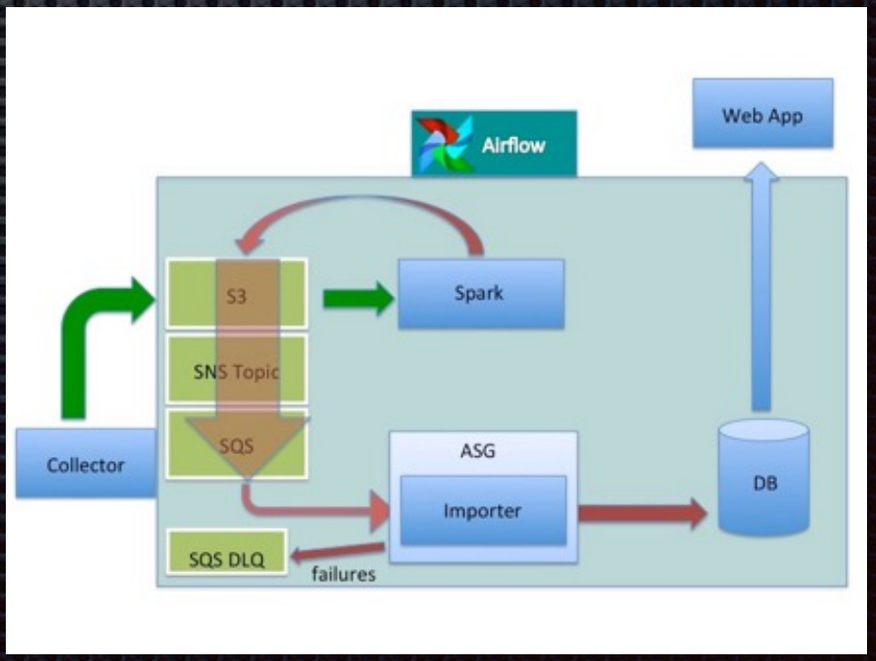
Airflow: Author DAGs in Python! No need to bundle many config files!



Apache Airflow - Authoring DAGs

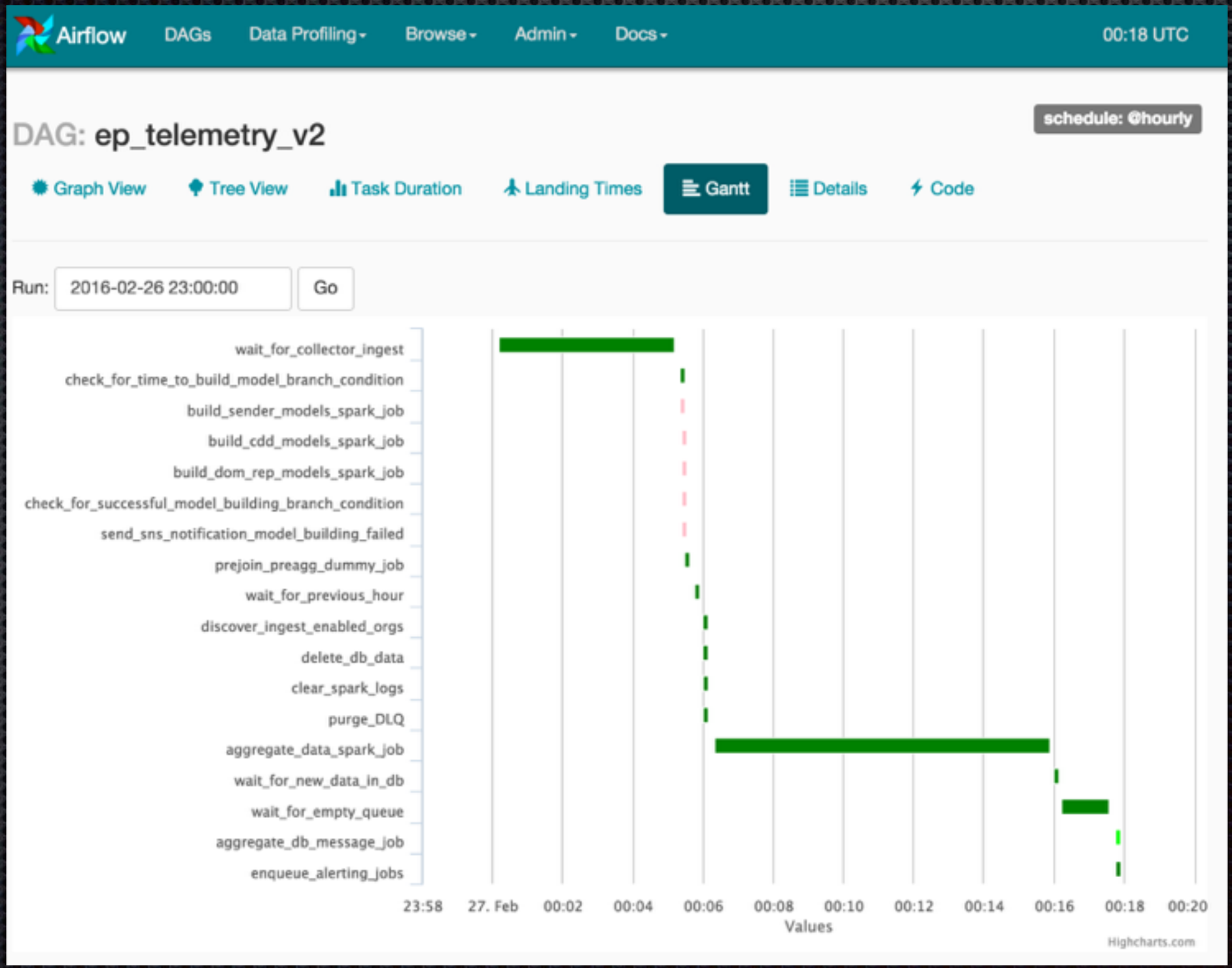
Airflow: Visualizing a DAG





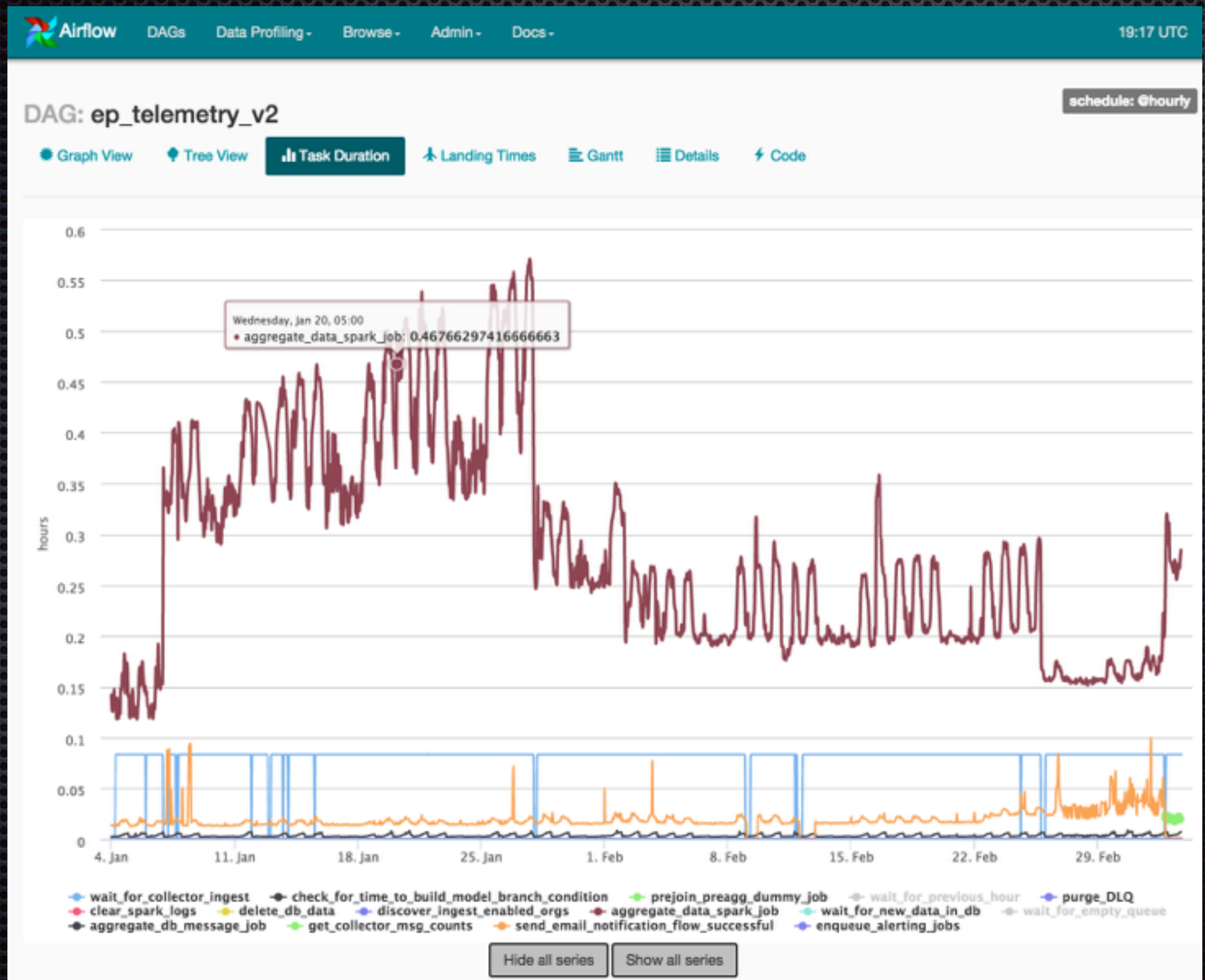
Apache Airflow - Perf. Insights

Airflow: Gantt chart view reveals the slowest tasks for a run!

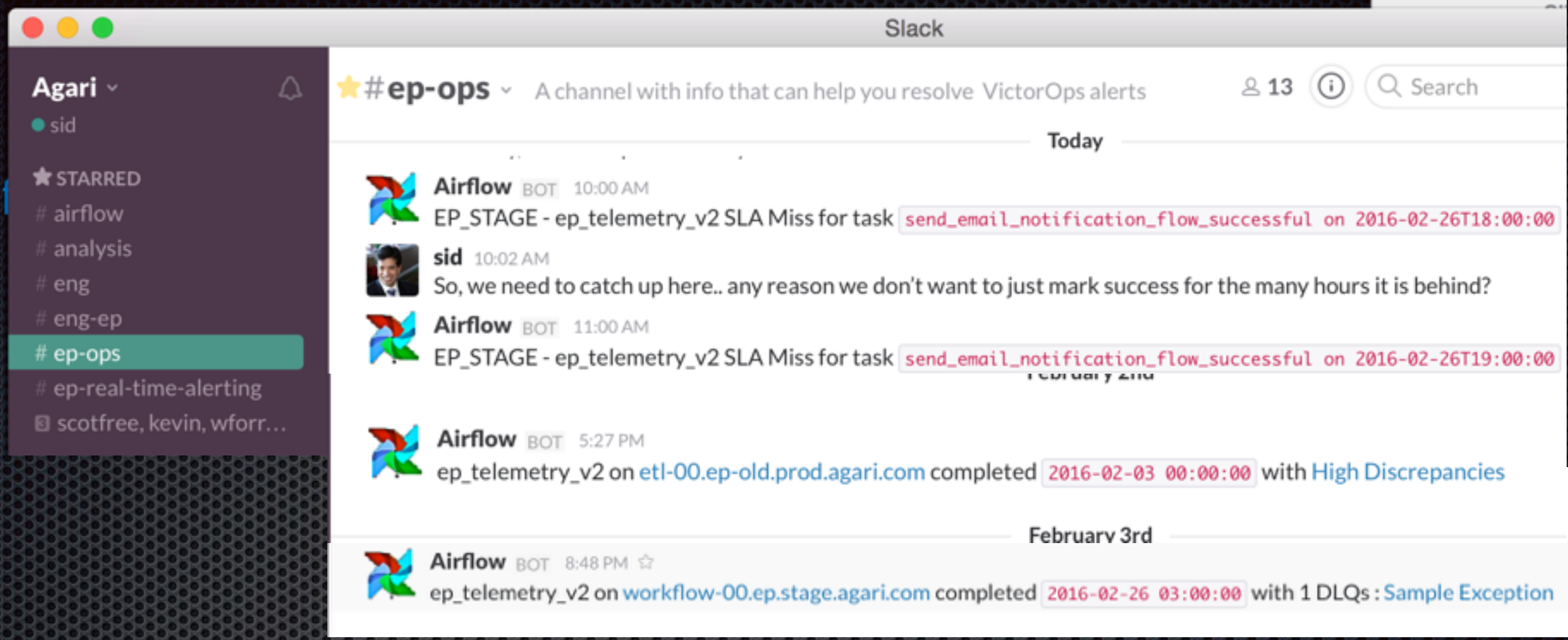


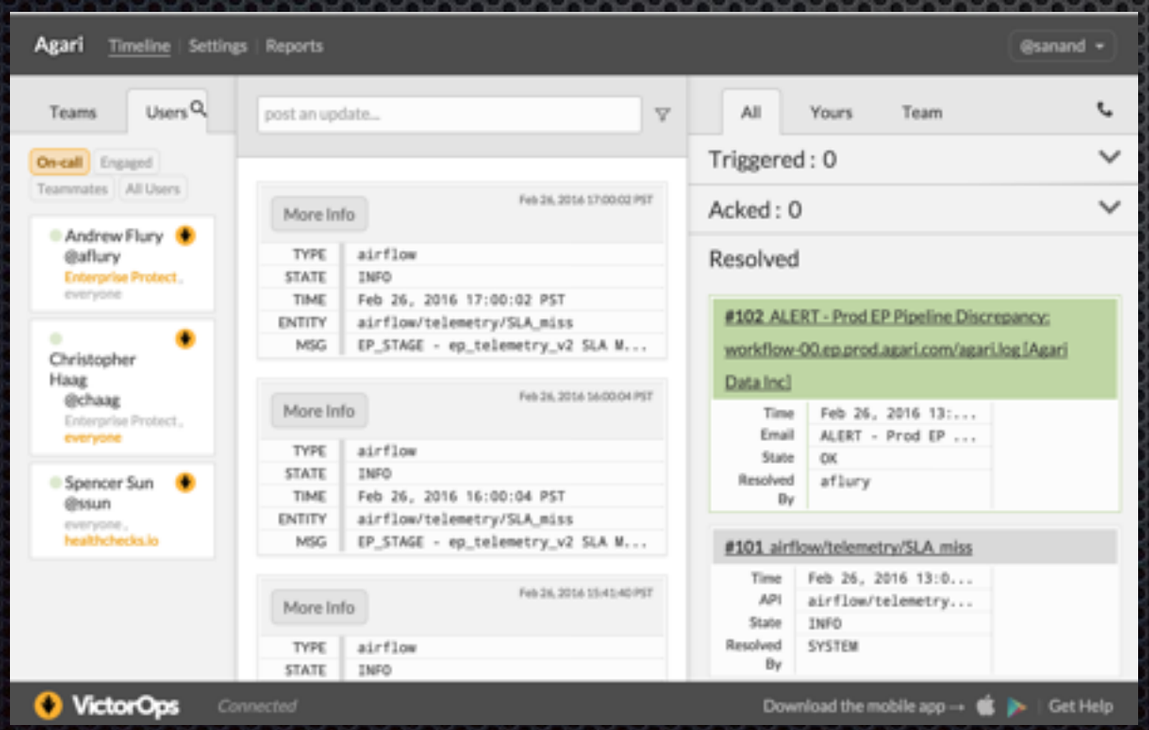
Apache Airflow - Perf. Insights

Airflow: Task Duration chart view show task completion time trends!



Apache Airflow - Alerting





Desirable Qualities of a Resilient Data Pipeline

Correctness

Operability

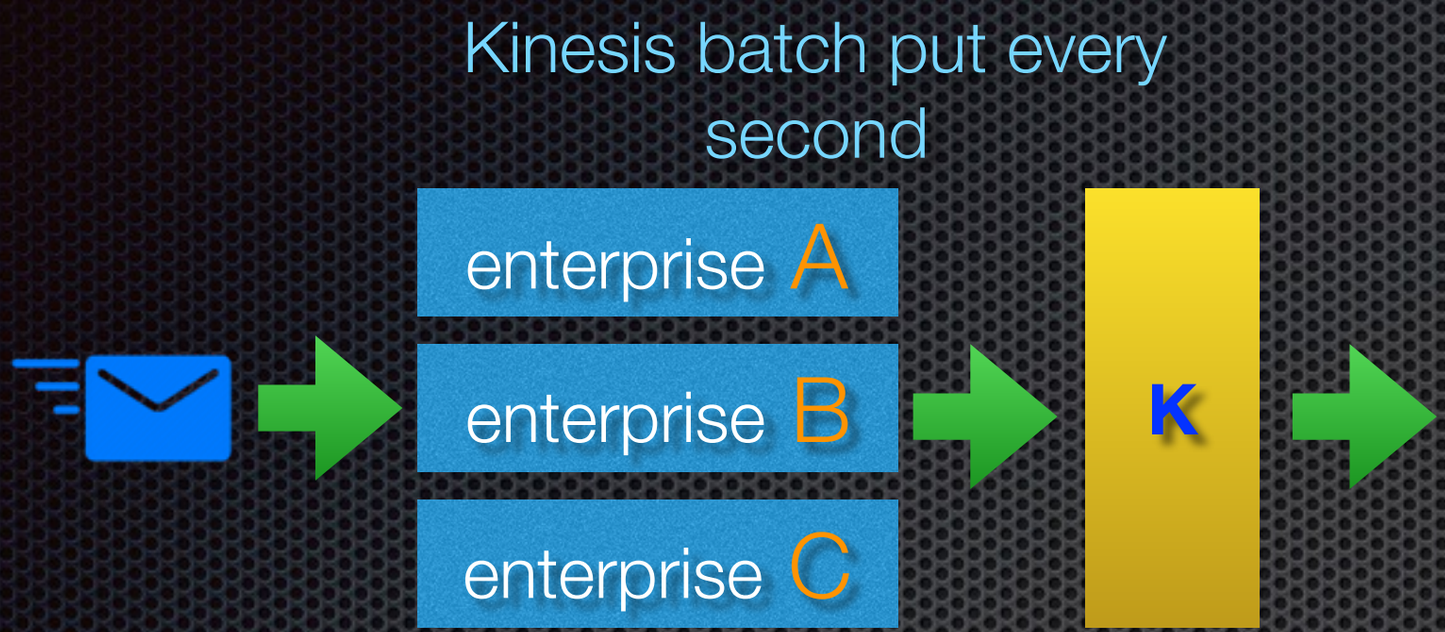
Timeliness

Cost

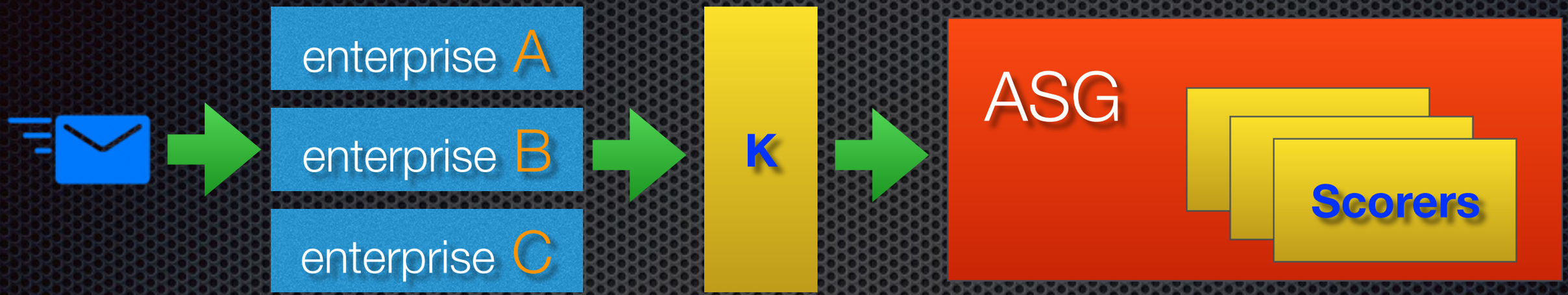
Use-Case : Message Scoring (near-real time)

NRT Pipeline Architecture

Use-Case : Message Scoring

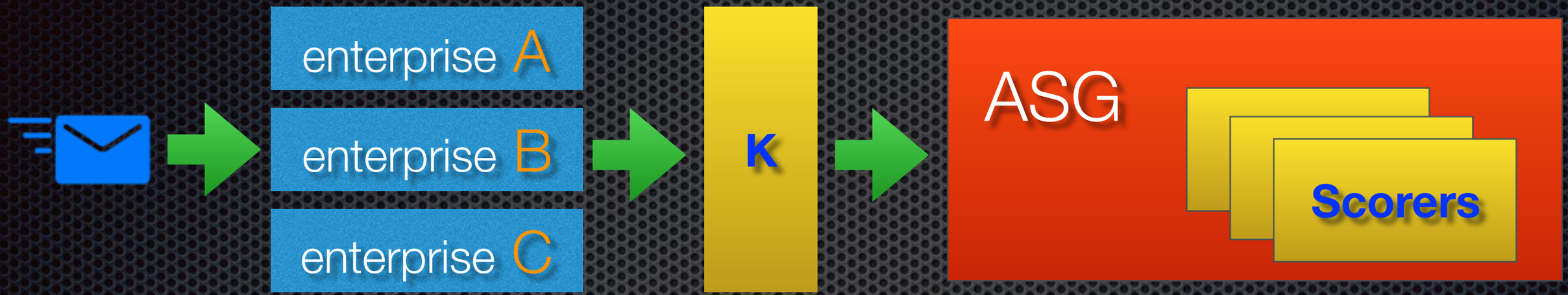


Use-Case : Message Scoring



As ASG of scorers is scaled up to one process per core per kinesis shard

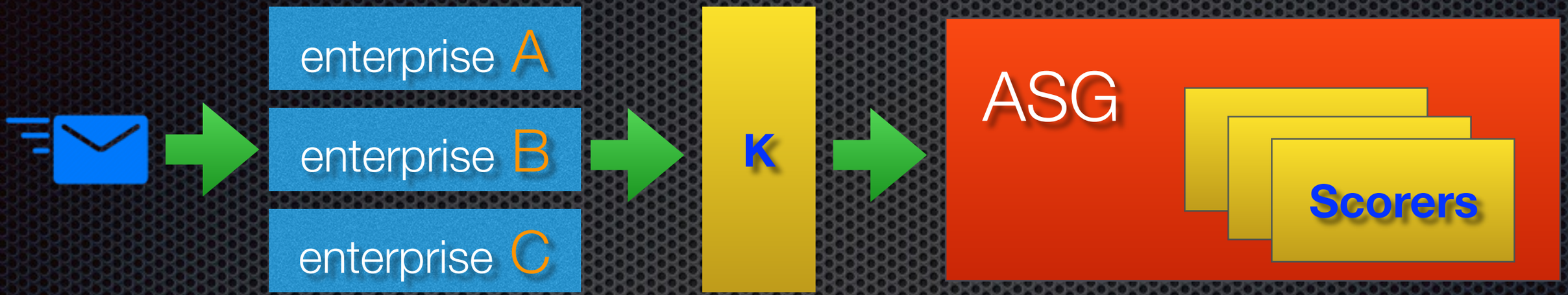
Use-Case : Message Scoring



Scorers apply the trust model and send scored messages downstream



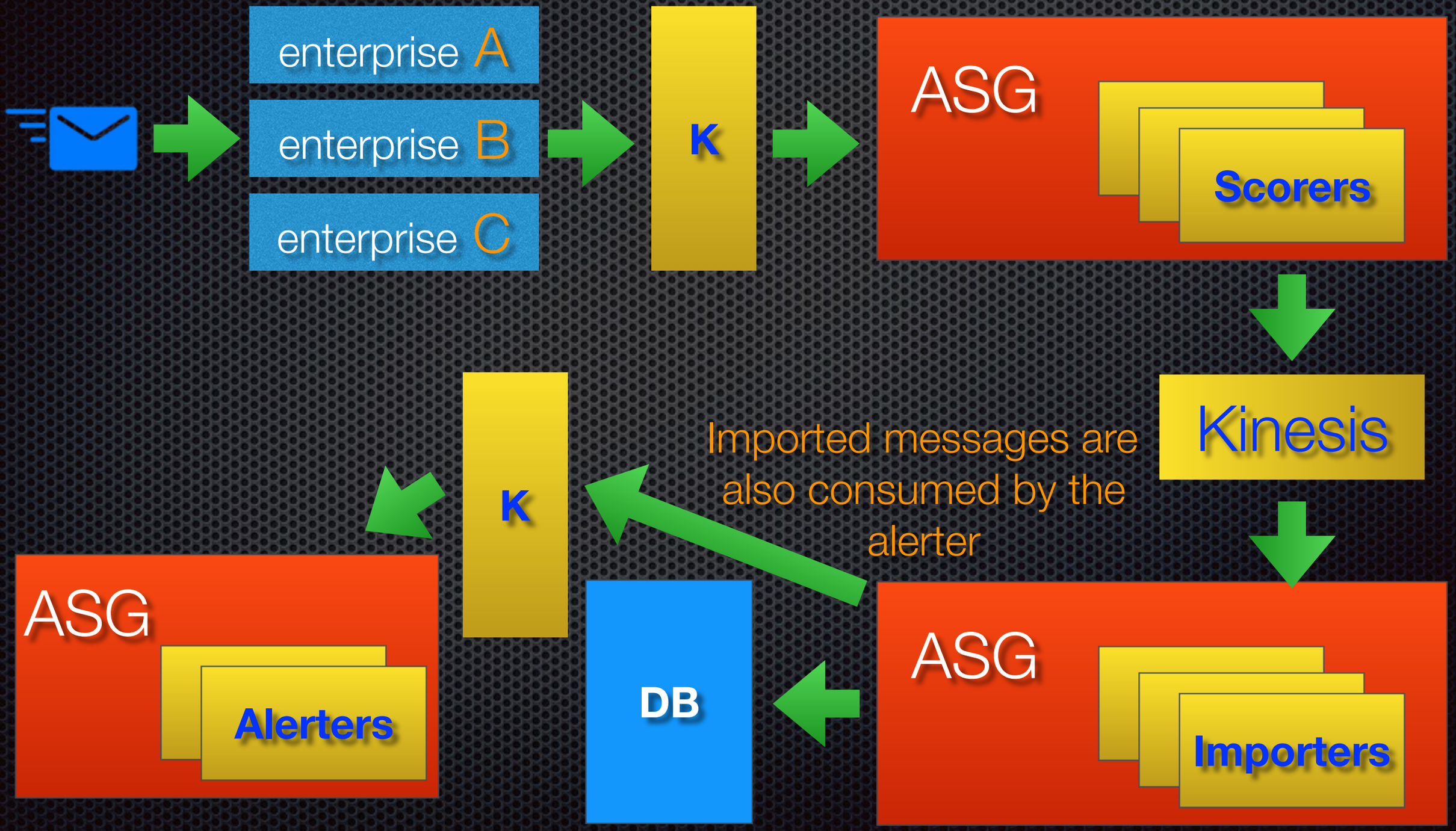
Use-Case : Message Scoring



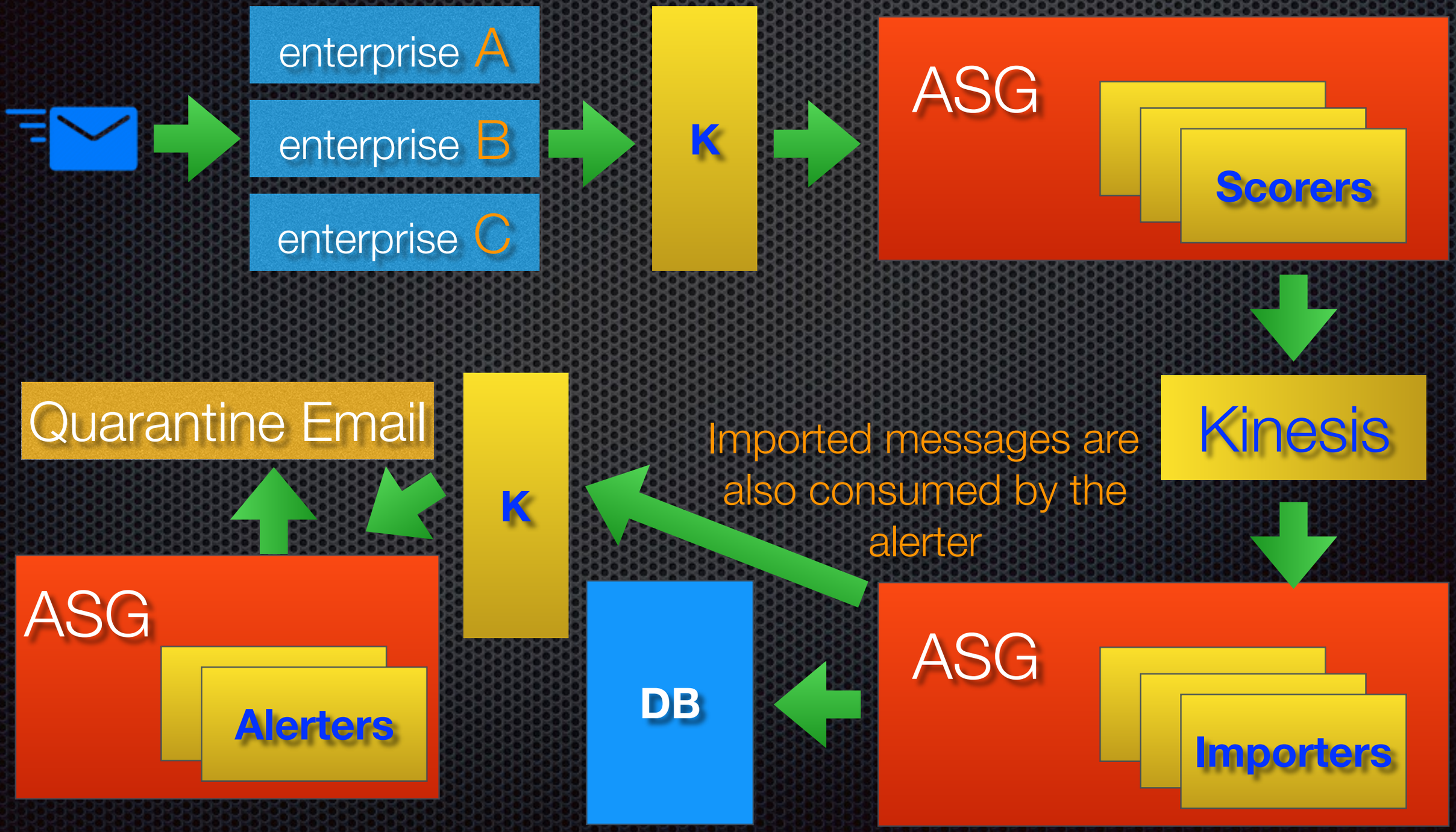
As ASG of importers is scaled up to rapidly import messages



Use-Case : Message Scoring



Use-Case : Message Scoring



Stream Processing Architecture

Component	Role	Details	Pros	Operability Model
S3	Data Lake	• All data stored in S3 via Kinesis Firehose	Scalable, Available, Performant, Serverless	**Serverless**
Kinesis	Messaging	• Streaming transport modeled on Kafka	Scalable, Available, Serverless	**Serverless**
	General Processing	• ASG Replacement except for Rails Apps	Scalable, Available, Serverless	**Serverless**
ASG	General Processing	• Used for importing, data cleansing, business logic	Scalable, Available, Managed	**Managed**
	Data Science Processing	• Model Building		**We Operate**
	Workflow Engine	• Nightly model builds + some classic Ops cron workloads	Lightweight, DAGs as Code	**We Operate**
DB 	Persistence for WebApp	• Holds **smaller** subset of data needed for Web App	Rails + Postgres 'nuff said	**We Operate**
  	Persistence for WebApp	• Aggregation + Search moved from DB to ES • Model Building queries moved to Elasticache Redis	Faster. more accurate for aggregates, frees up headroom for DB (polyglot persistence)	**Managed**

Innovations

NRT Pipeline Architecture

Apache Avro

What is Avro?

What is Avro?

- Avro is a self-describing serialization format that supports
  - primitive data types : int, long, boolean, float, string, bytes, etc…
  - complex data types : records, arrays, unions, maps, enums, etc…
  - many language bindings : Java, Scala, Python, Ruby, etc…

What is Avro?

- Avro is a self-describing serialization format that supports
  - primitive data types : int, long, boolean, float, string, bytes, etc…
  - complex data types : records, arrays, unions, maps, enums, etc…
  - many language bindings : Java, Scala, Python, Ruby, etc…

- The most common format for storing structured Big Data at rest in HDFS, S3, Google Cloud Storage, etc…

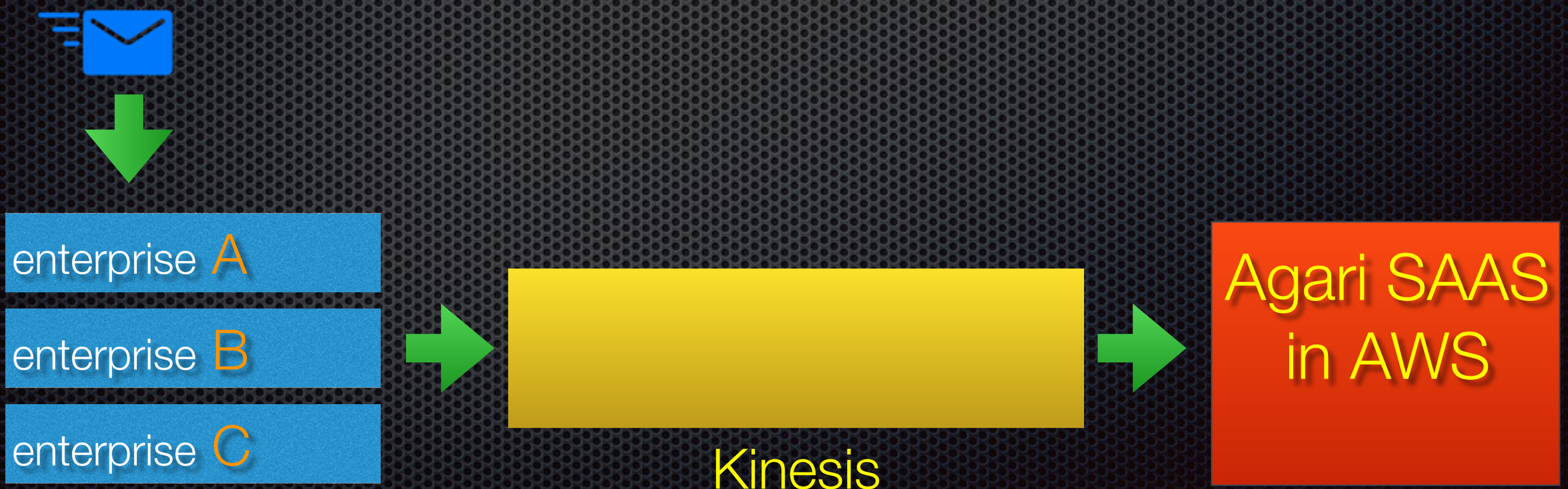
- Supports Schema Evolution!

Apache Avro

Why is it useful?

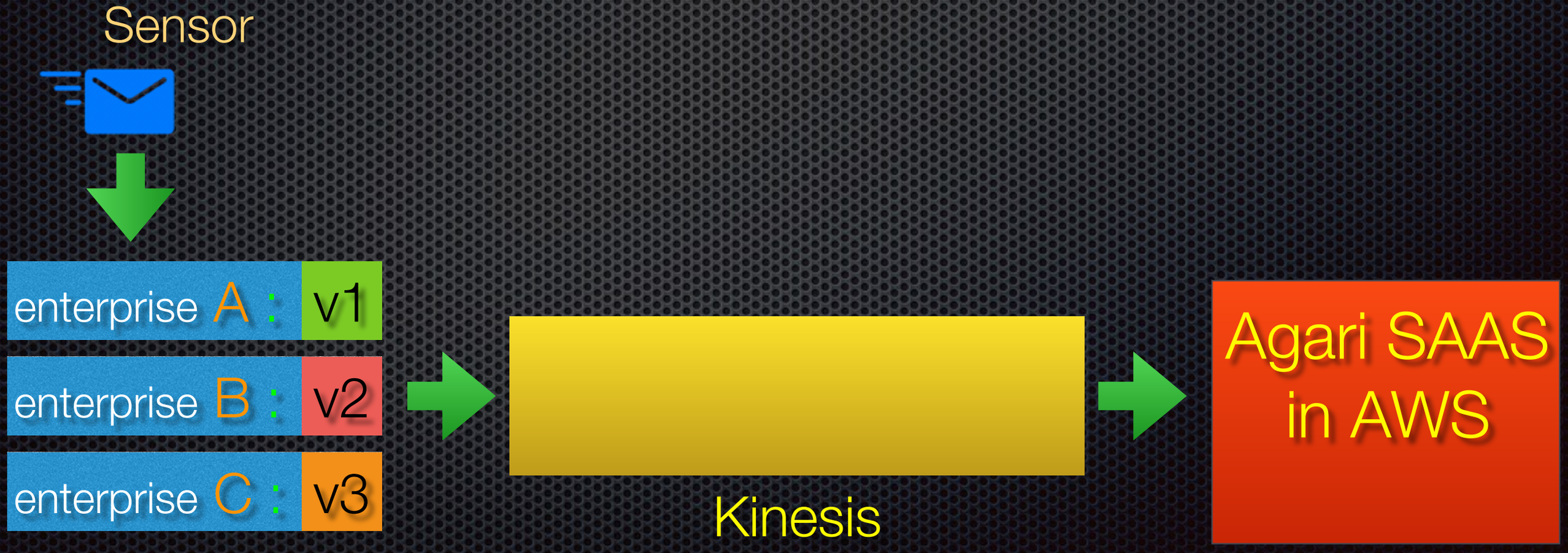
Why is Avro Useful?

- Agari is an IoT company!
  - Agari Sensors, deployed at customer sites, stream data to Agari's Cloud SAAS
  - Data is sent via Kinesis!



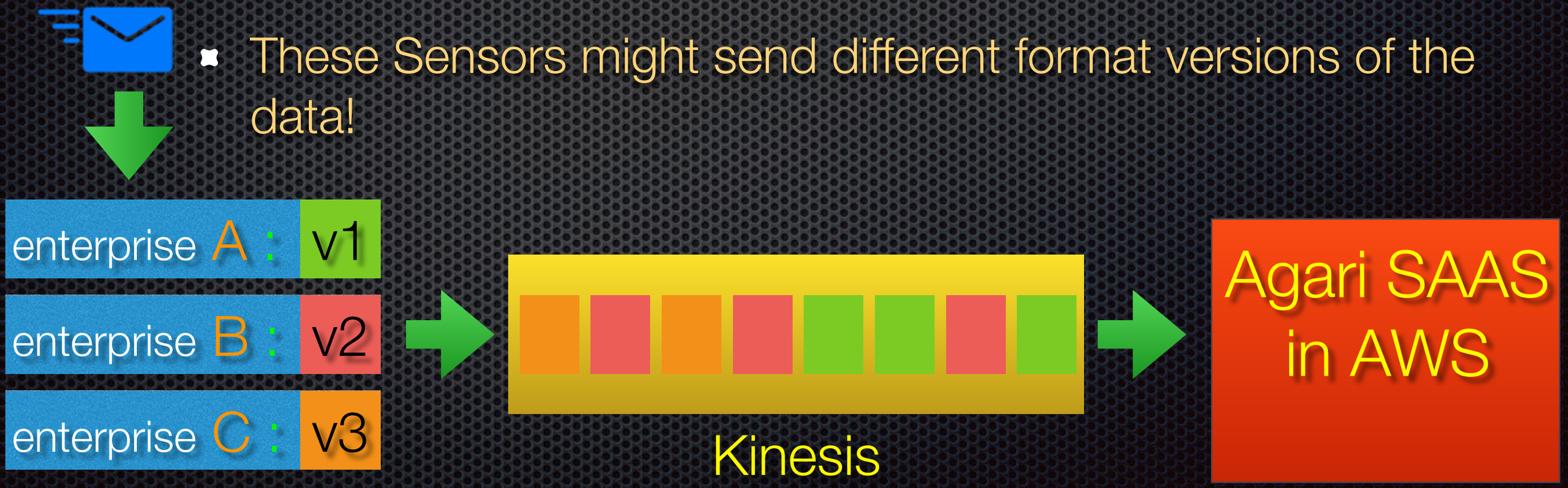
Why is Avro Useful?

- Agari is an IoT company!
  - Agari Sensors, deployed at customer sites, stream data to Agari's Cloud SAAS
  - Data is sent via Kinesis!
  - At any point in time, customers run different versions of the Agari



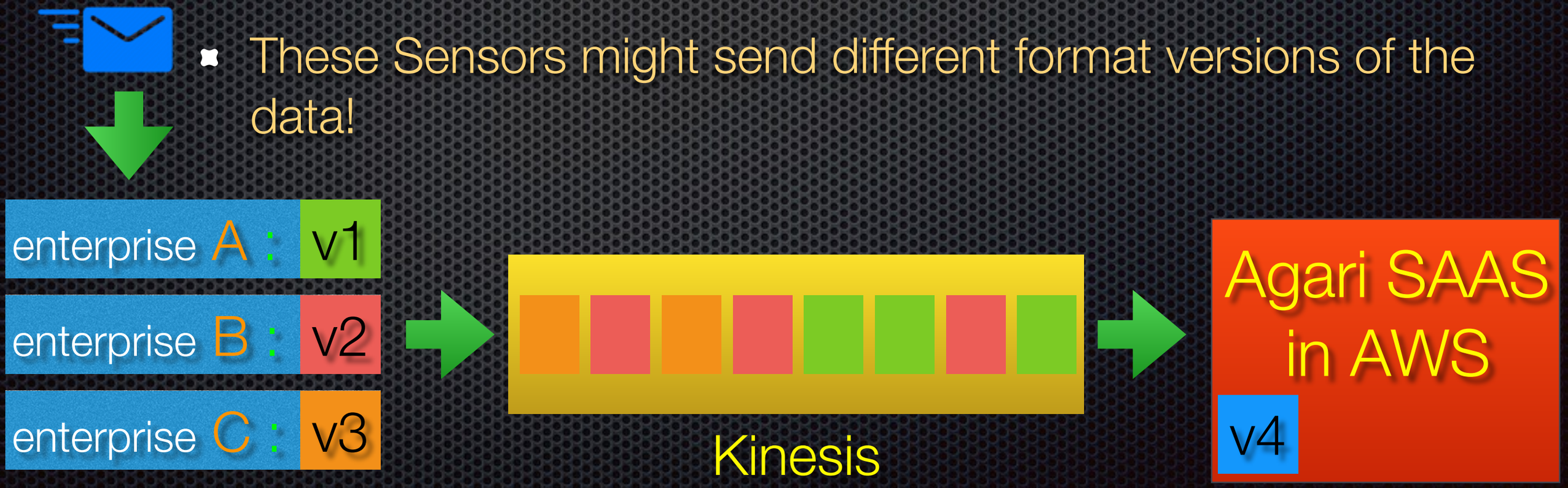
Why is Avro Useful?

- Agari is an IoT company!
  - Agari Sensors, deployed at customer sites, stream data to Agari's Cloud SAAS
  - Data is sent via Kinesis!
  - At any point in time, customers run different versions of the Agari Sensor
  - These Sensors might send different format versions of the data!



Why is Avro Useful?

- Agari is an IoT company!
  - Agari Sensors, deployed at customer sites, stream data to Agari's Cloud SAAS
  - Data is sent via Kinesis!
  - At any point in time, customers run different versions of the Agari Sensor
  - These Sensors might send different format versions of the data!



Why is Avro Useful?

- Avro allows Agari to seamlessly handle different IoT data format versions

```
datum_reader = DatumReader( writers_schema = writers_schema,
                            readers_schema = readers_schema)
```

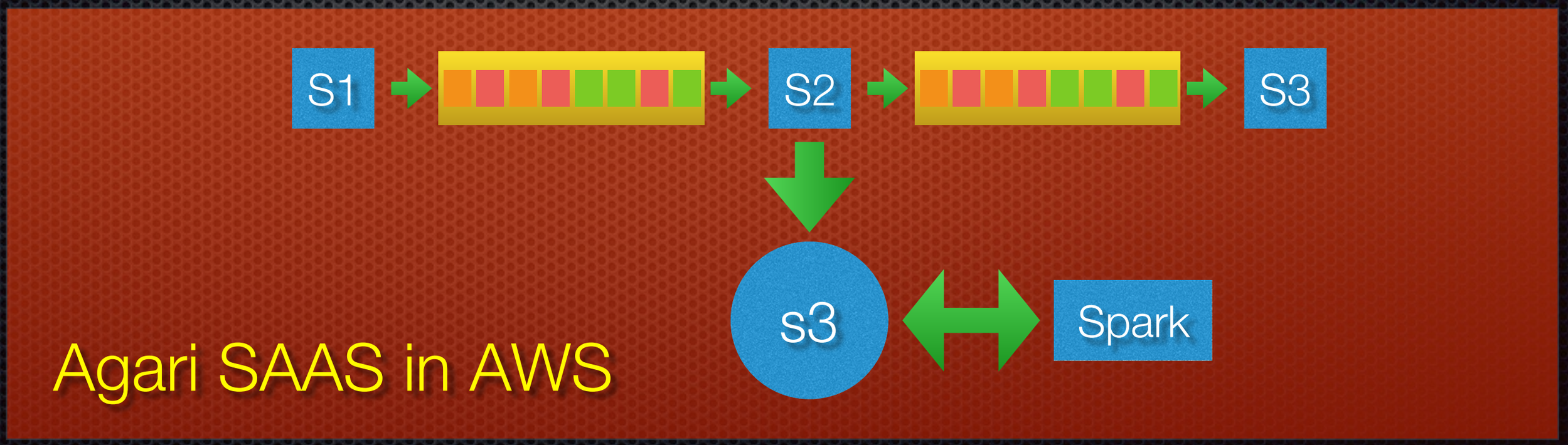
Requirements:

- Schemas are backward-compatible



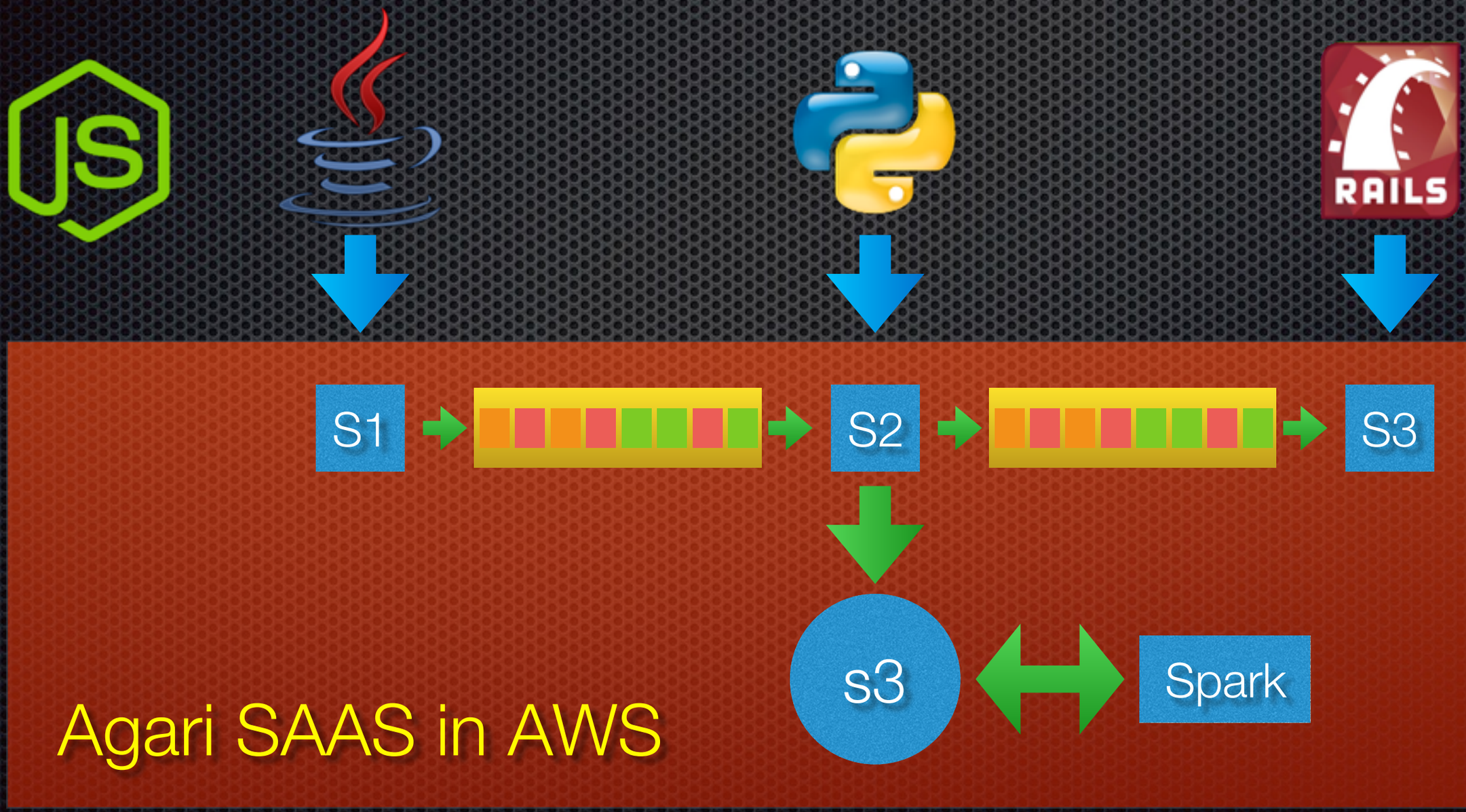
Why is Avro Useful?

- Avro Everywhere!
  - Avro is so useful, we don't just to communicate between our Sensors & our SAAS infrastructure
  - We also use it as the common data-interchange format between all services (streaming & batch) within our AWS deployment



Why is Avro Useful?

- Avro Everywhere!
  - Good Language Bindings :
    - Data Pipelines services are written in Java, Ruby, & Python



Apache Avro

By Example

Avro Schema Example

```
{"namespace": "agari",
 "type": "record",
 "name": "User",
 "fields": [
    {"name": "name", "type": "string"},
    {"name": "favorite_number",  "type": ["int", "null"]},
    {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

complex type (record)

Schema name : User

3 fields in the record: 1 required, 2 optional

Avro Schema Data File Example

```
{"namespace": "agari",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Schema

0.0001 %

Data	Data	Data	Data	Data	Data
Data	Data	Data	Data	Data	Data
Data	Data	Data	Data	Data	Data

Data

99.999 %

x 1,000,000,000

Avro Schema Streaming Example

```
{"namespace": "agari",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Data

Schema

99 %

Data

1 %

Avro Schema Streaming Example

```
{"namespace": "agari",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Data

Schema

99 %

OVERHEAD!!

Data

1 %

Apache Avro

Schema Registry

Avro Schema Registry

Schema Registry (**Lambda**)

register_schema

Message Producer **(P)**

```
{"namespace": "agari",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Avro Schema Registry

Schema Registry (**Lambda**)

register_schema returns a UUID

Message Producer **(P)**

Avro Schema Registry

Schema Registry (**Lambda**)

Message Producer **(P)**



Message Consumer **(C)**

Message Producer sends UUID + Data

Avro Schema Registry

Schema Registry (**Lambda**)

getSchemaById (UUID)



Message Producer **(P)**



Message Consumer **(C)**

Data

Avro Schema Registry

Schema Registry (**Lambda**)

getSchemaById (UUID)

```
{"namespace": "agari",
 "type": "record",
 "name": "User",
 "fields": [
    {"name": "name", "type": "string"},
    {"name": "favorite_number",  "type": ["int", "null"]},
    {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Message Producer **(P)**

Message Consumer **(C)**

Data

Avro Schema Registry

Schema Registry (**Lambda**)

getSchemaById (UUID)

```
{"namespace": "agari",
 "type": "record",
 "name": "User",
 "fields": [
    {"name": "name", "type": "string"},
    {"name": "favorite_number",  "type": ["int", "null"]},
    {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

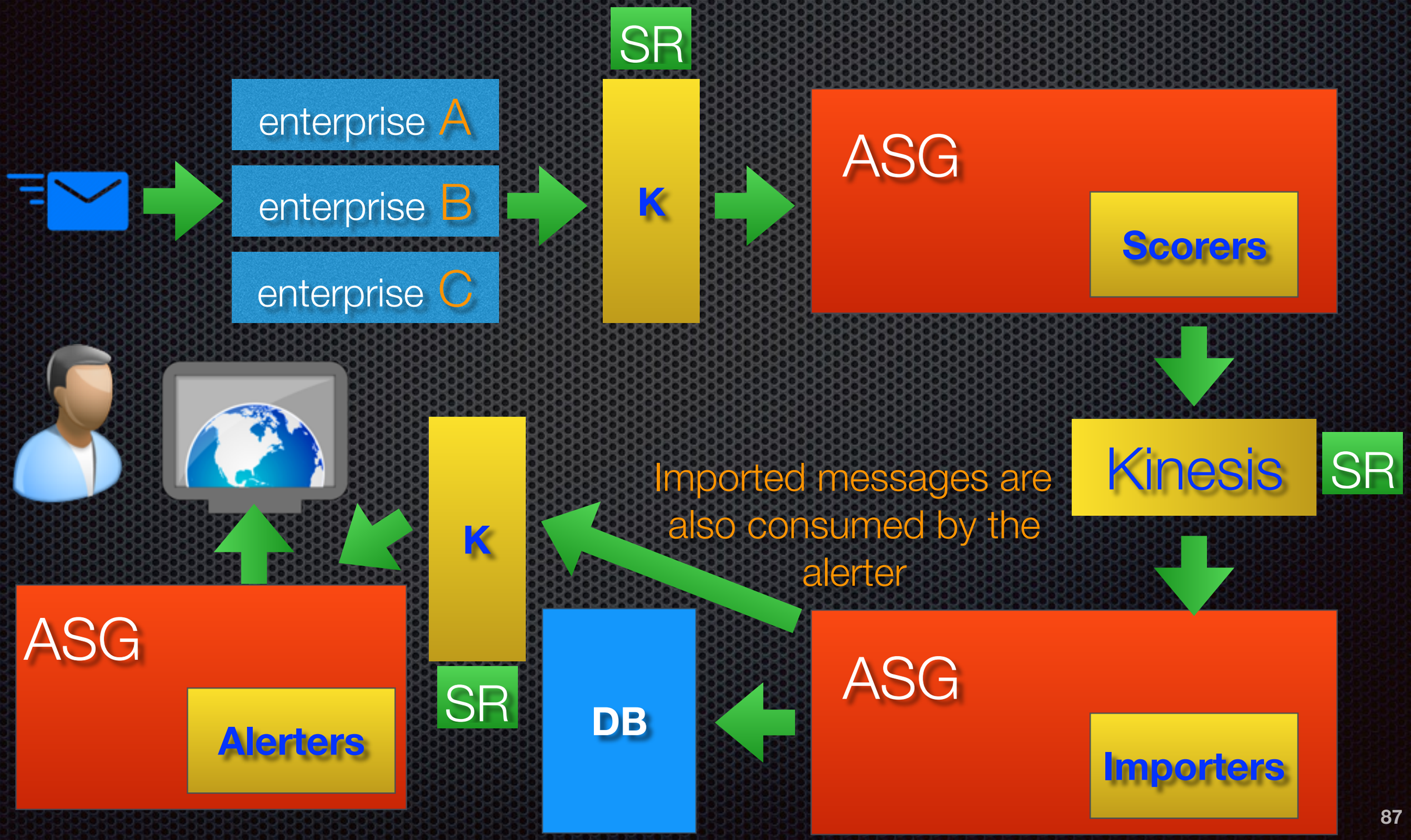
Message Producer **(P)**

Message Consumer **(C)**

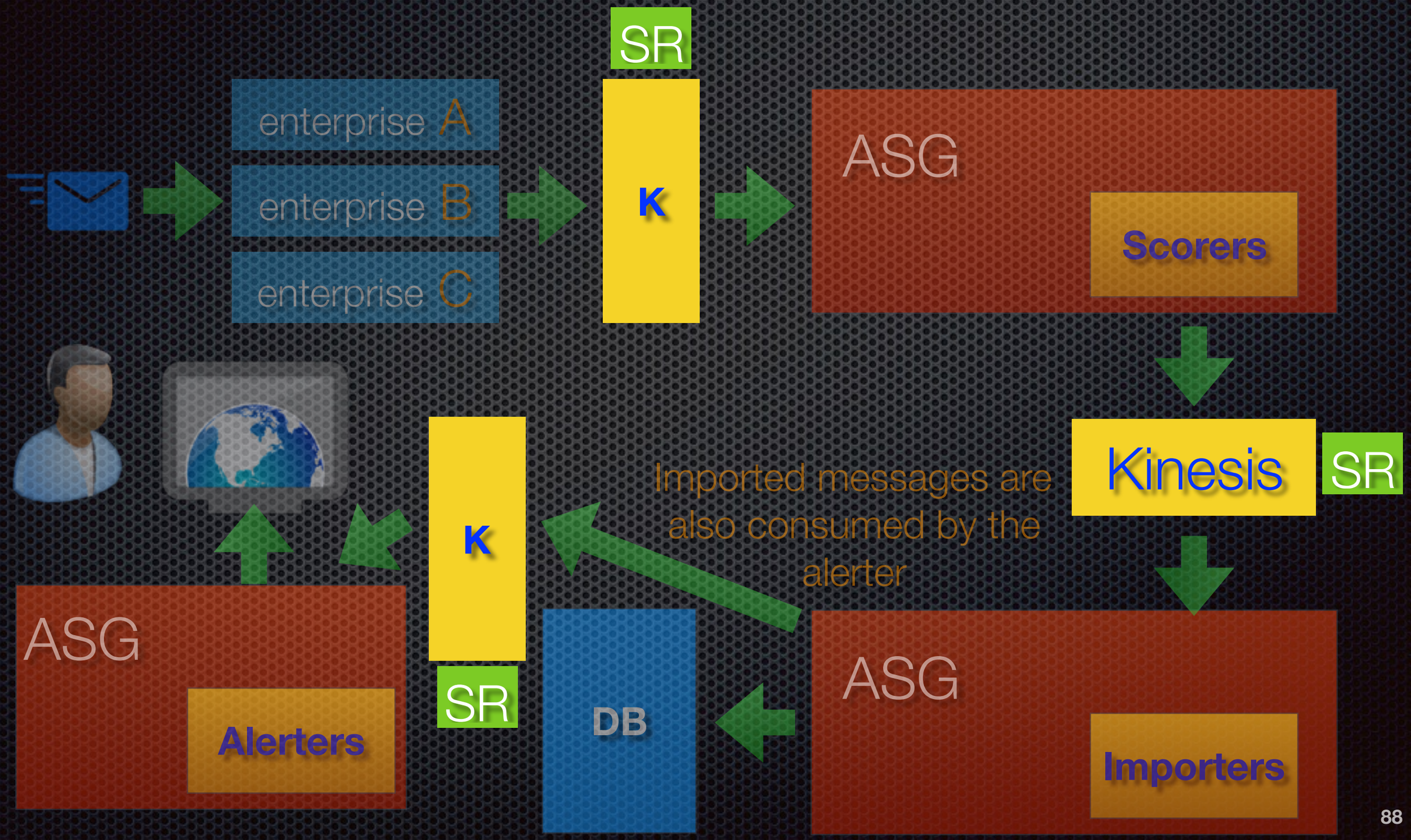
Message Consumers

- download & cache the schema
- then decode the data

Avro Schema Registry



Avro Schema Registry



Acknowledgments

None of this work would be possible without the essential contributions of the team below

- Vidur Apparao
- Stephen Cattaneo
- Jon Chase
- Andrew Flury
- William Forrester
- Chris Haag
- Chris Buchanan
- Neil Chapin
- Wil Collins
- Don Spencer
- Scot Kennedy
- Natia Chachkhiani
- Patrick Cockwell
- Kevin Mandich
- Gabriel Ortiz
- Jacob Rideout
- Josh Yang
- Julian Mehnle
- Gabriel Poon
- Spencer Sun
- Nathan Bryant

Questions?

(@r39132)

