

# Resilient Predictive Data Pipelines

Sid Anand (@r39132)

QCon London 2016

# About Me

Work [ed | s] @



Co-Chair for



Maintainer on



[airbnb/airflow](#)

Report to



# Motivation

Why is a Data Pipeline talk in this High Availability Track?

# Different Types of Data Pipelines

## ETL

## Predictive

**VS**

- used for : loading data related to business health into a Data Warehouse

- user-engagement stats (e.g. social networking)
- product success stats (e.g. e-commerce)

- audience : Business, BizOps

- downtime? : 1-3 days

- used for :

- building recommendation products (e.g. social networking, shopping)
- updating fraud prevention endpoints (e.g. security, payments, e-commerce)

- audience : Customers

- downtime? : < 1 hour

# Different Types of Data Pipelines

## ETL

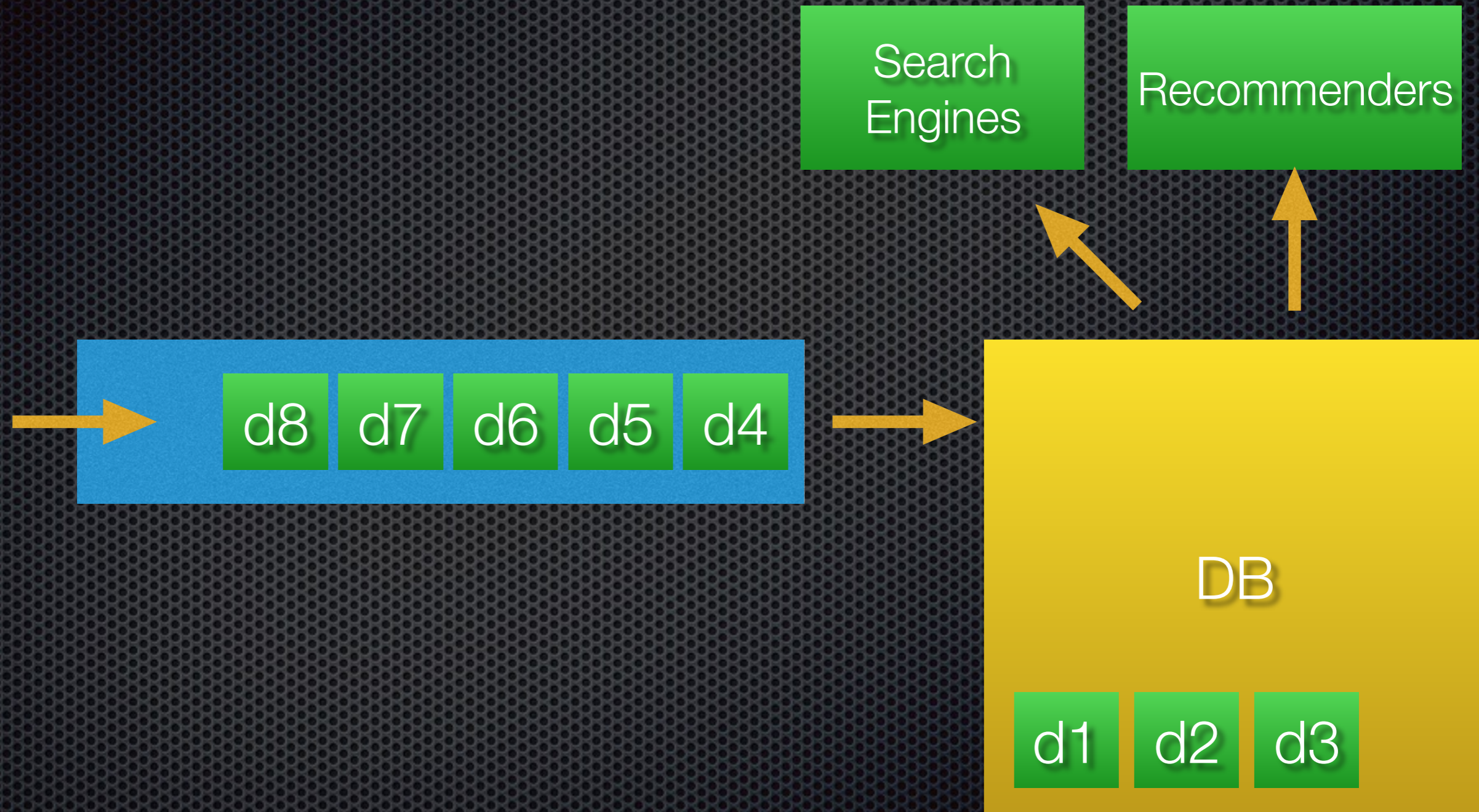
- used for : loading data into a Data Warehouse —> Reports
  - user-engagement stats (e.g. social networking)
  - product success stats (e.g. e-commerce)
- audience : Business
- downtime? : 1-3 days

VS

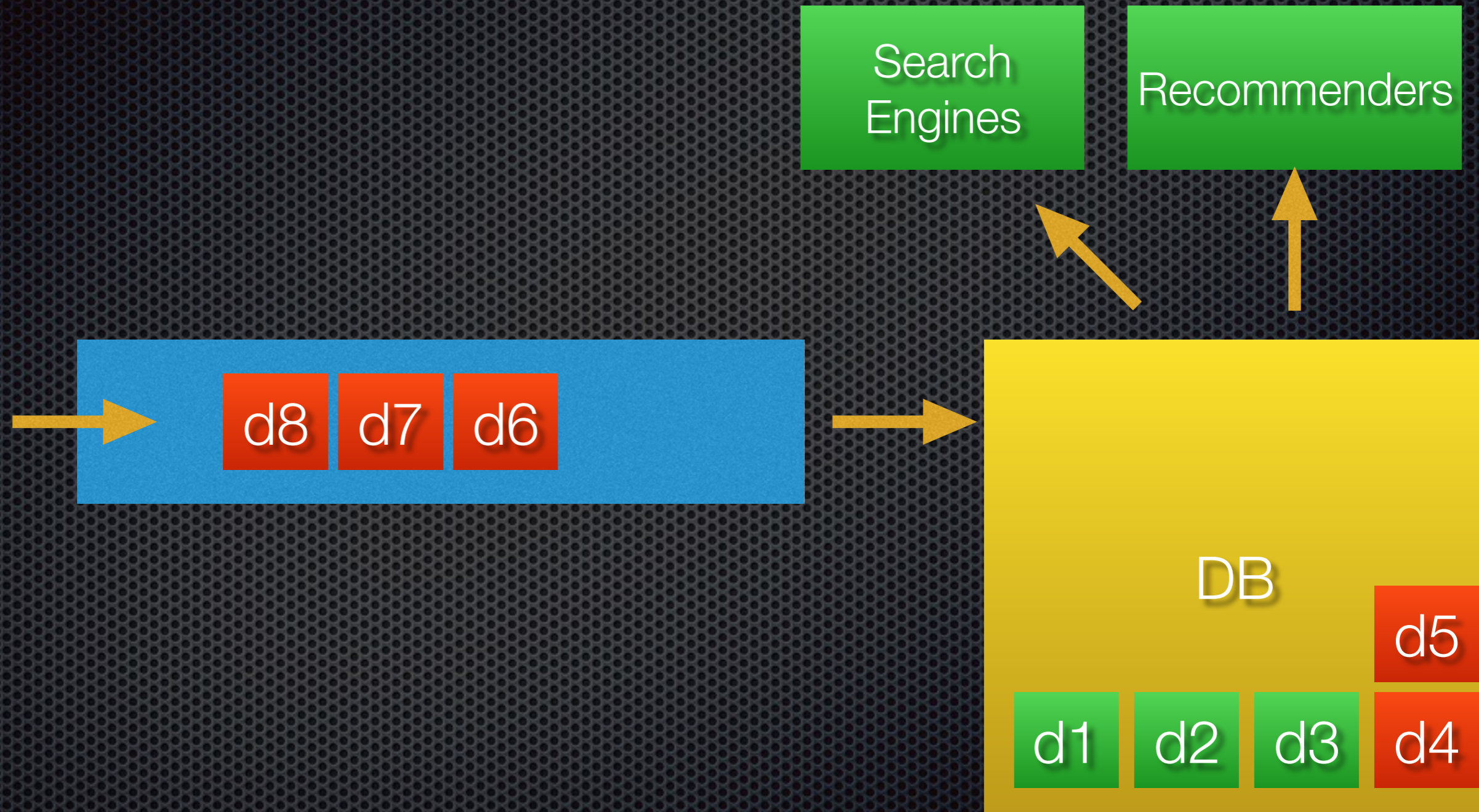
## Predictive

- used for :
  - building recommendation products (e.g. social networking, shopping)
  - updating fraud prevention endpoints (e.g. security, payments, e-commerce)
- audience : Customers
- downtime? : < 1 hour

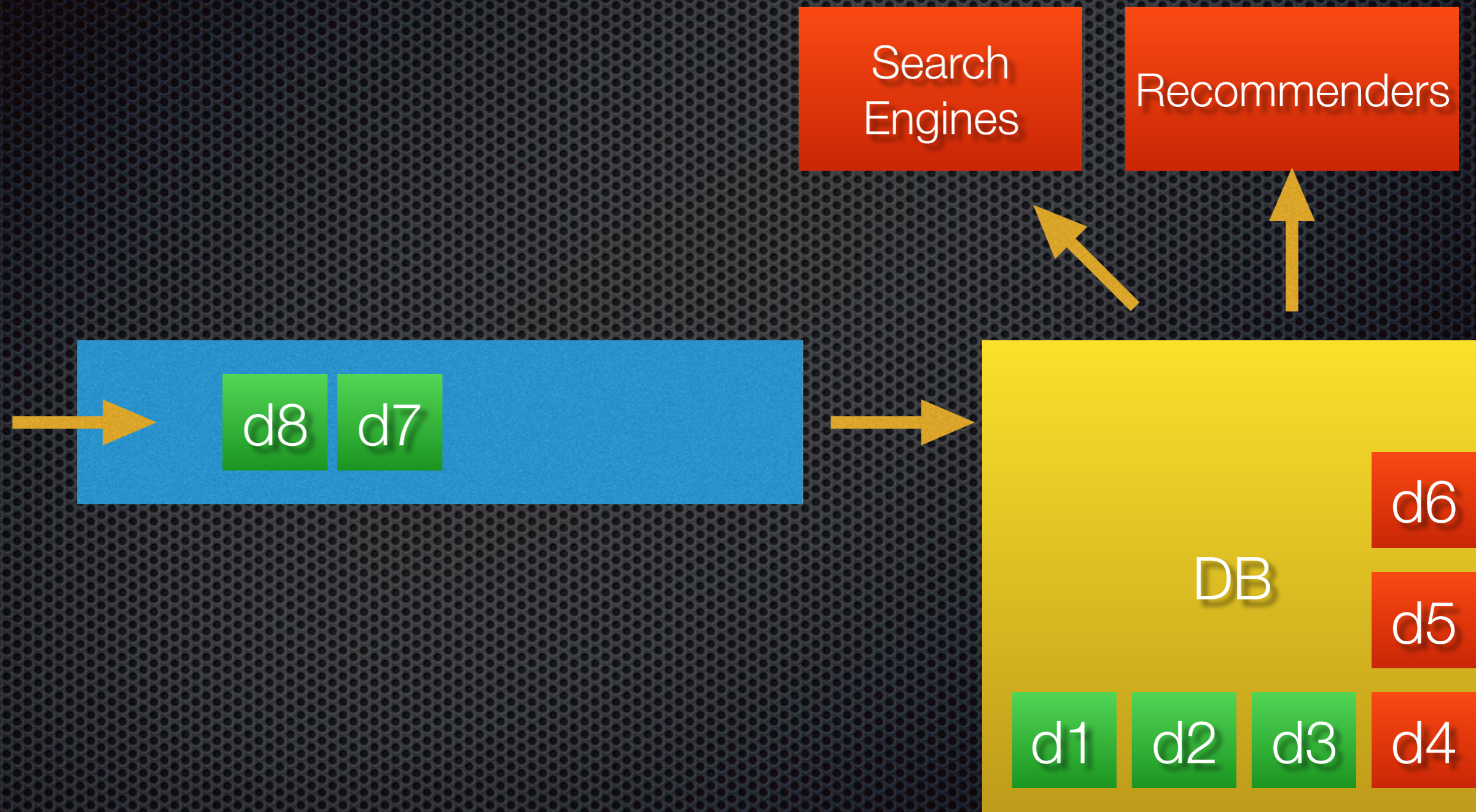
# Why Do We Care About Resilience?



# Why Do We Care About Resilience?

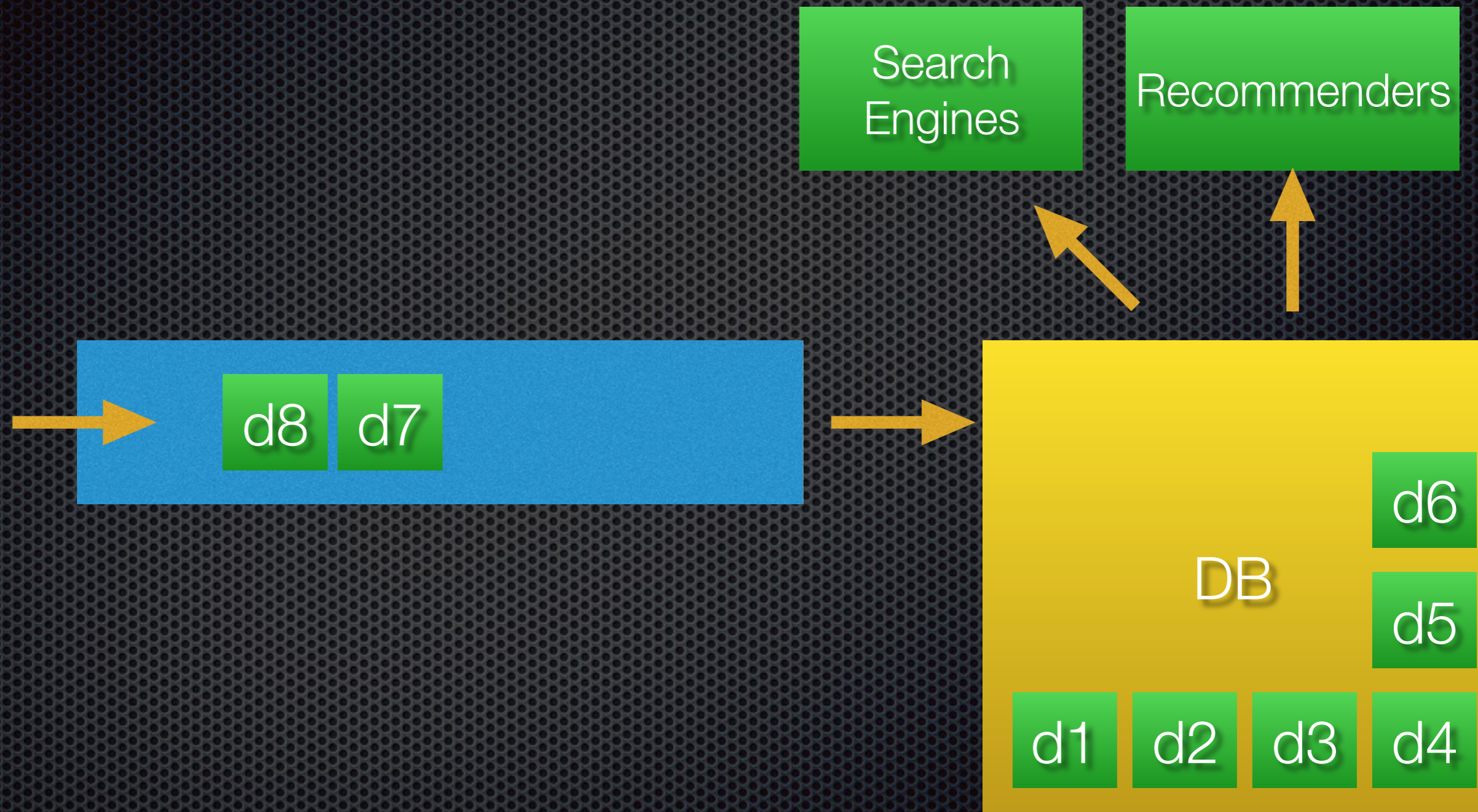


# Why Do We Care About Resilience?

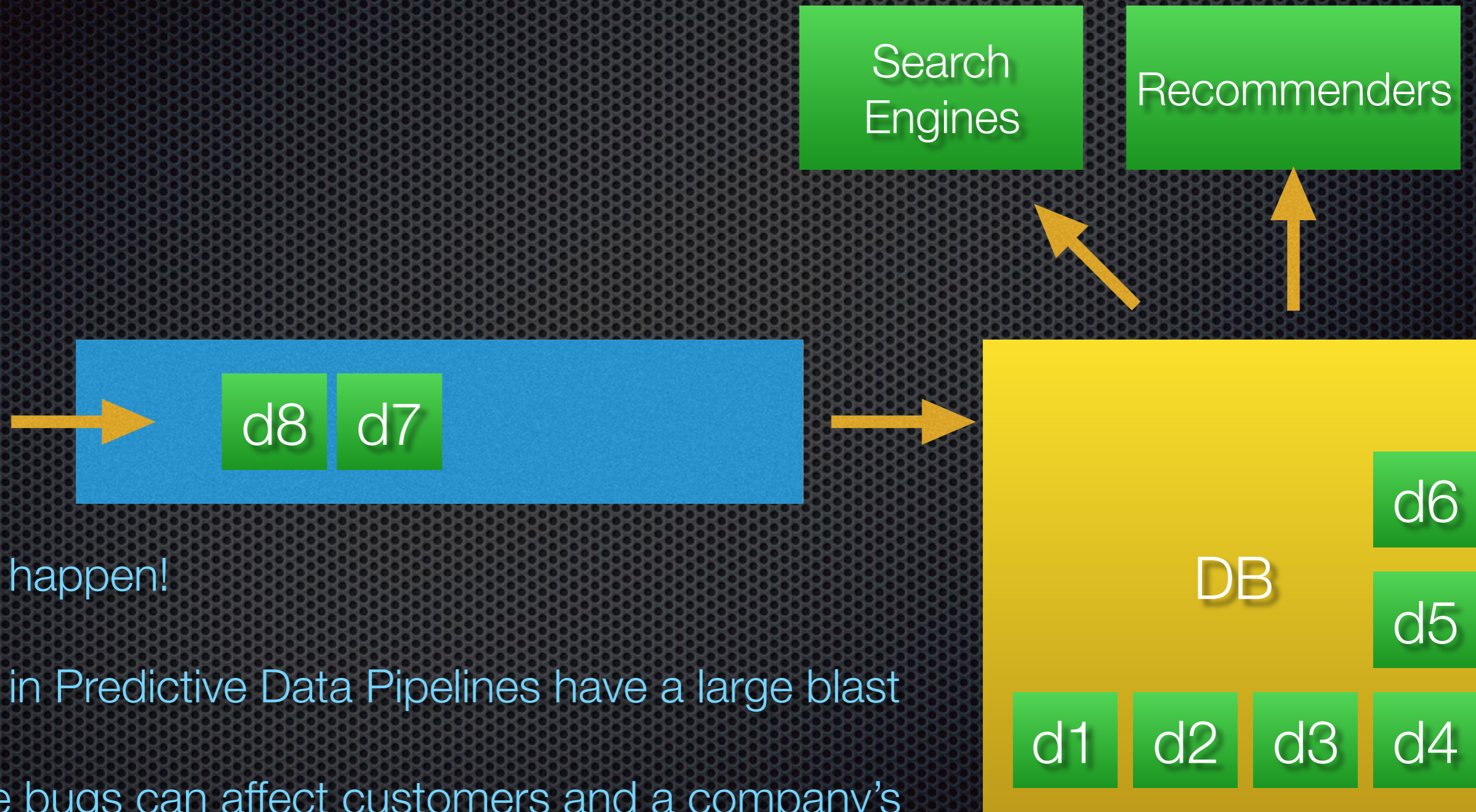




# Why Do We Care About Resilience?



# Any Take-aways?



- Bugs happen!
- Bugs in Predictive Data Pipelines have a large blast radius
  - The bugs can affect customers and a company's profits & reputation!

# Design Goals

Desirable Qualities of a Resilient Data Pipeline

# Desirable Qualities of a Resilient Data Pipeline

- Scalable
- Available
- Instrumented, Monitored, & Alert-enabled
- Quickly Recoverable

# Desirable Qualities of a Resilient Data Pipeline

- Scalable
  - Build your pipelines using [infinitely] scalable components
    - The scalability of your system is determined by its least-scalable component
- Available
- Instrumented, Monitored, & Alert-enabled
- Quickly Recoverable

# Desirable Qualities of a Resilient Data Pipeline

- Scalable
  - Build your pipelines using [infinitely] scalable components
    - The scalability of your system is determined by its least-scalable component
- Available
  - Ditto
- Instrumented, Monitored, & Alert-enabled
- Quickly Recoverable

# Instrumented

Instrumentation must reveal SLA metrics at each stage of the pipeline!

What SLA metrics do we care about? **Correctness** & **Timeliness**

- **Correctness**
  - No Data Loss
  - No Data Corruption
  - No Data Duplication
  - A Defined Acceptable Staleness of Intermediate Data
- **Timeliness**
  - A late result == a useless result
  - Delayed processing of **now()**'s data may delay the processing of future data

# Instrumented, Monitored, & Alert-enabled

- Instrument
  - Instrument **Correctness** & **Timeliness** SLA metrics at each stage of the pipeline
- Monitor
  - Continuously monitor that SLA metrics fall within acceptable bounds (i.e. **pre-defined SLAs**)
- Alert
  - Alert when we miss SLAs



# Desirable Qualities of a Resilient Data Pipeline

- Scalable
  - Build your pipelines using [infinitely] scalable components
    - The scalability of your system is determined by its least-scalable component
- Available
  - Ditto
- Instrumented, Monitored, & Alert-enabled
- Quickly Recoverable

# Quickly Recoverable

- Bugs happen!
- Bugs in Predictive Data Pipelines have a large blast radius
- Optimize for MTTR

## Maintainability



**MTTR Optimized**

**Versus**



**MTBF Optimized**

More info here: [http://ti.arc.nasa.gov/projects/ishem/Papers/ONeill\\_Maintainability.doc](http://ti.arc.nasa.gov/projects/ishem/Papers/ONeill_Maintainability.doc)

# Implementation

Using AWS to meet Design Goals

# SQS

Simple Queue Service

# SQS - Overview

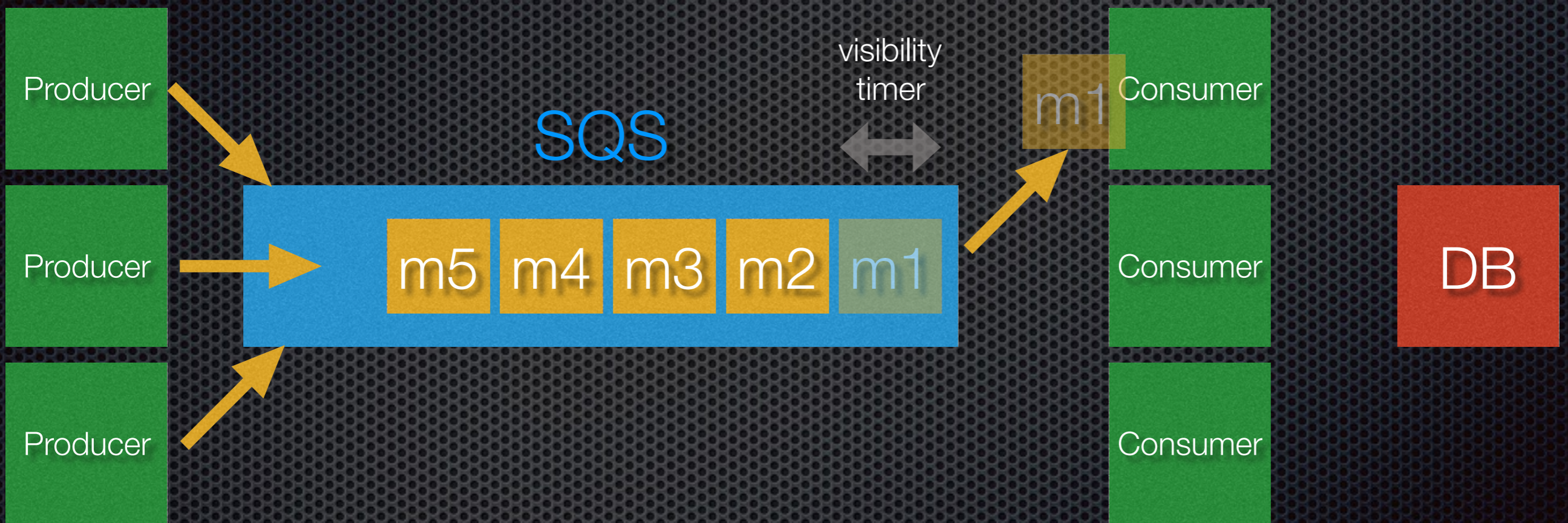
- ✦ AWS's low-latency, highly scalable, highly available message queue
  - ✦ Infinitely Scalable Queue (though not FIFO)
  - ✦ Low End-to-end latency (generally sub-second)
  - ✦ Pull-based

## How it Works!

- ✦ **Producers** publish messages, which can be batched, to an SQS queue
- ✦ **Consumers**
  - ✦ consume messages, which can be batched, from the queue
  - ✦ commit message contents to a data store
  - ✦ ACK the messages as a batch

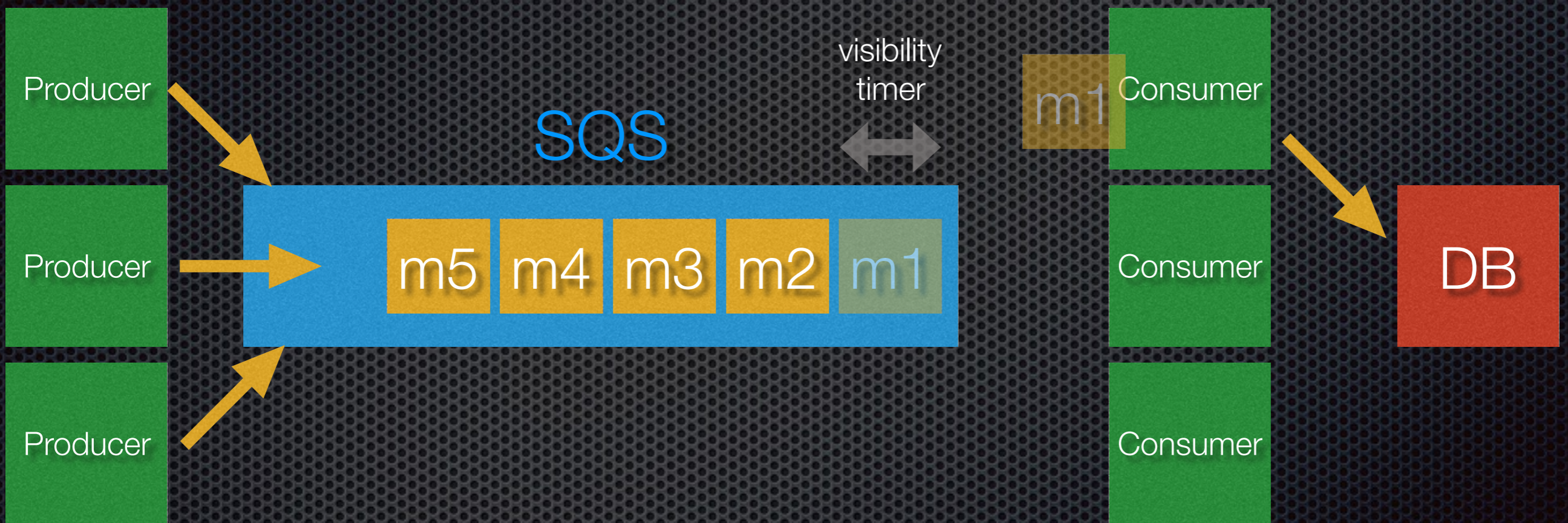
# SQS - Typical Operation Flow

**Step 1:** A consumer reads a message from SQS. This starts a visibility timer!



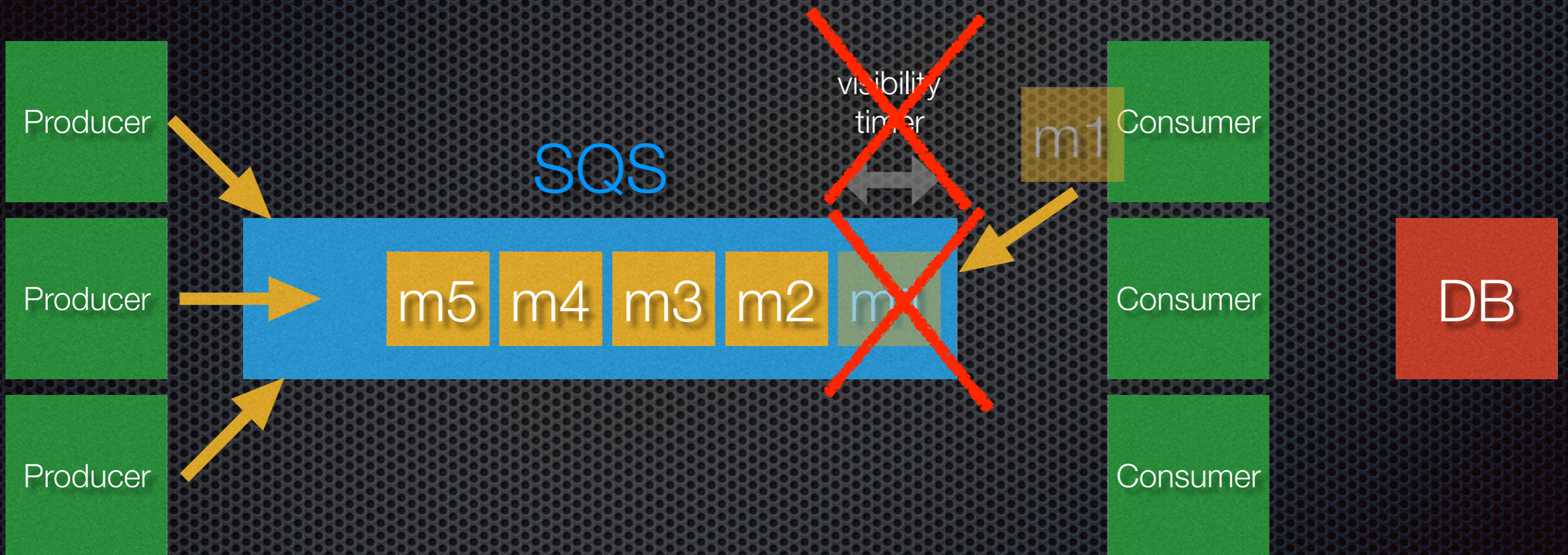
# SQS - Typical Operation Flow

**Step 2:** Consumer persists message contents to DB



# SQS - Typical Operation Flow

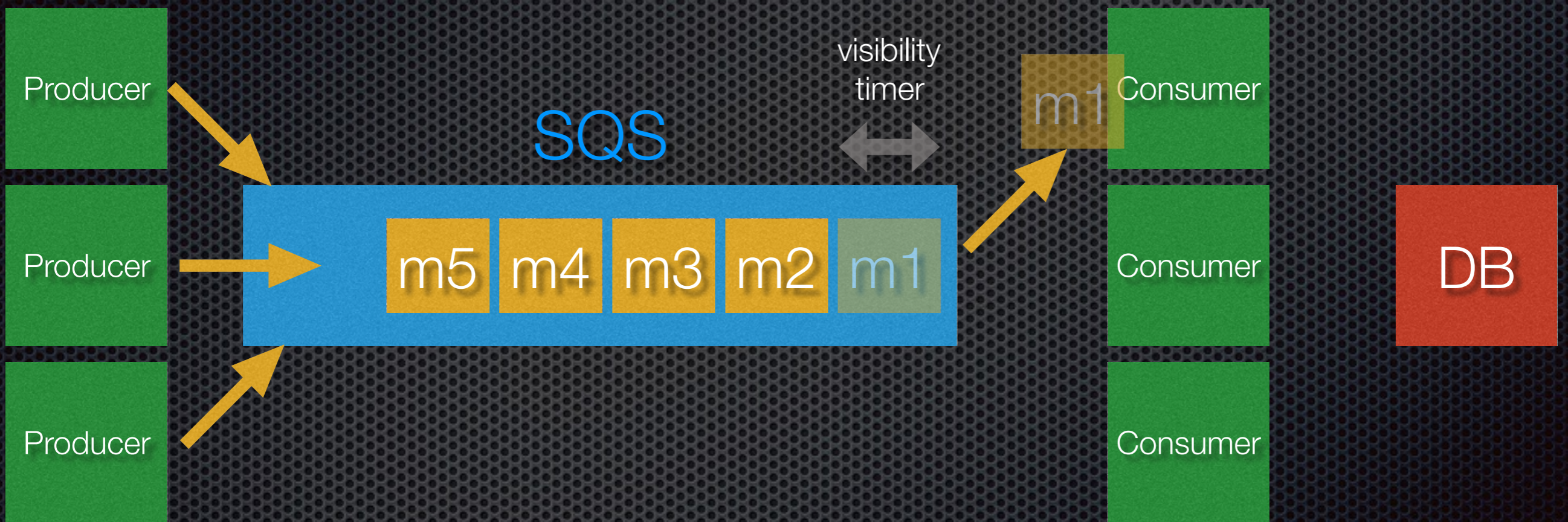
## Step 3: Consumer ACKs message in SQS





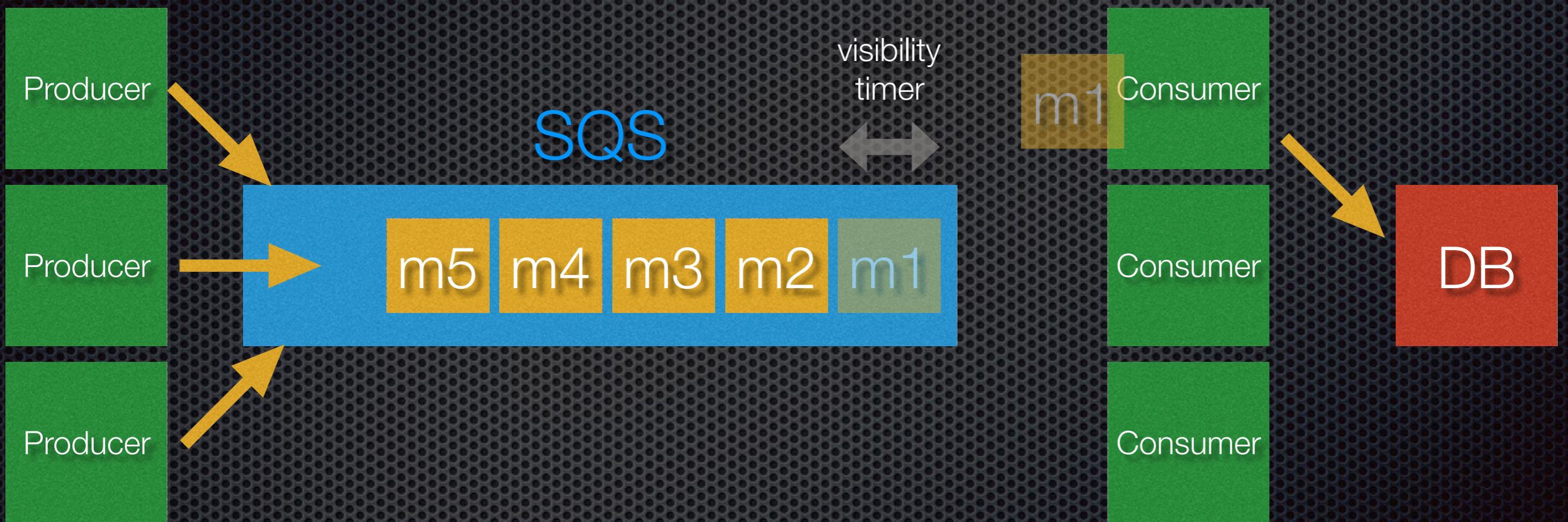
# SQS - Time Out Example

**Step 1:** A consumer reads a message from SQS



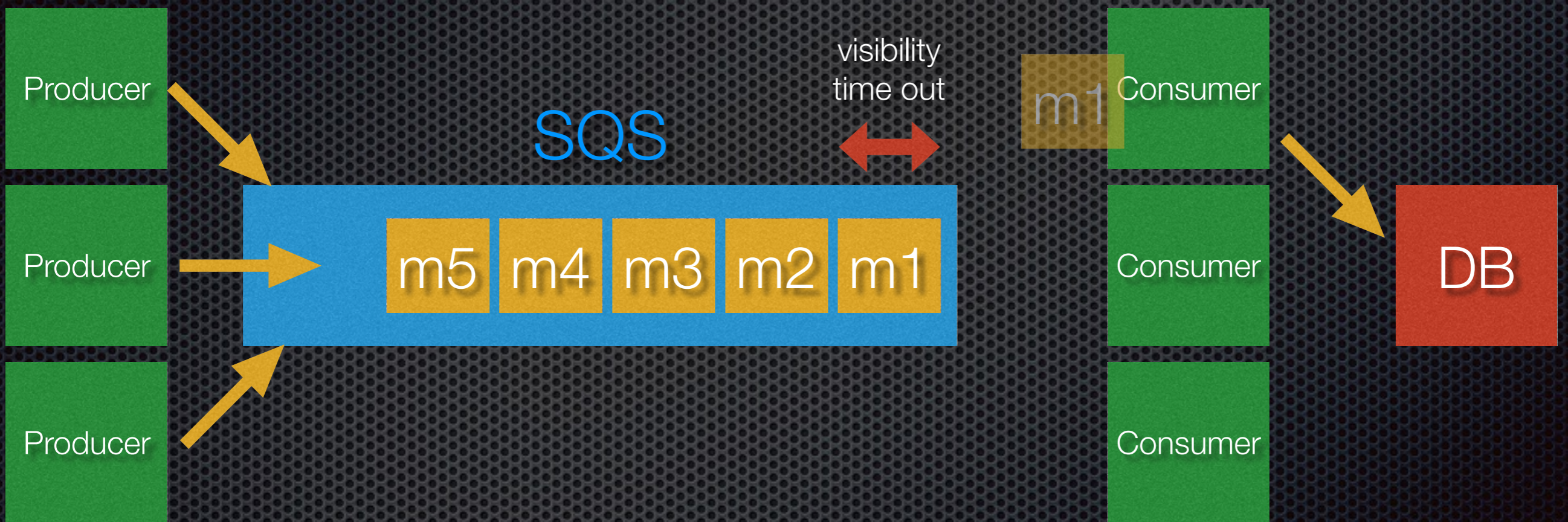
# SQS - Time Out Example

**Step 2:** Consumer attempts persists message contents to DB



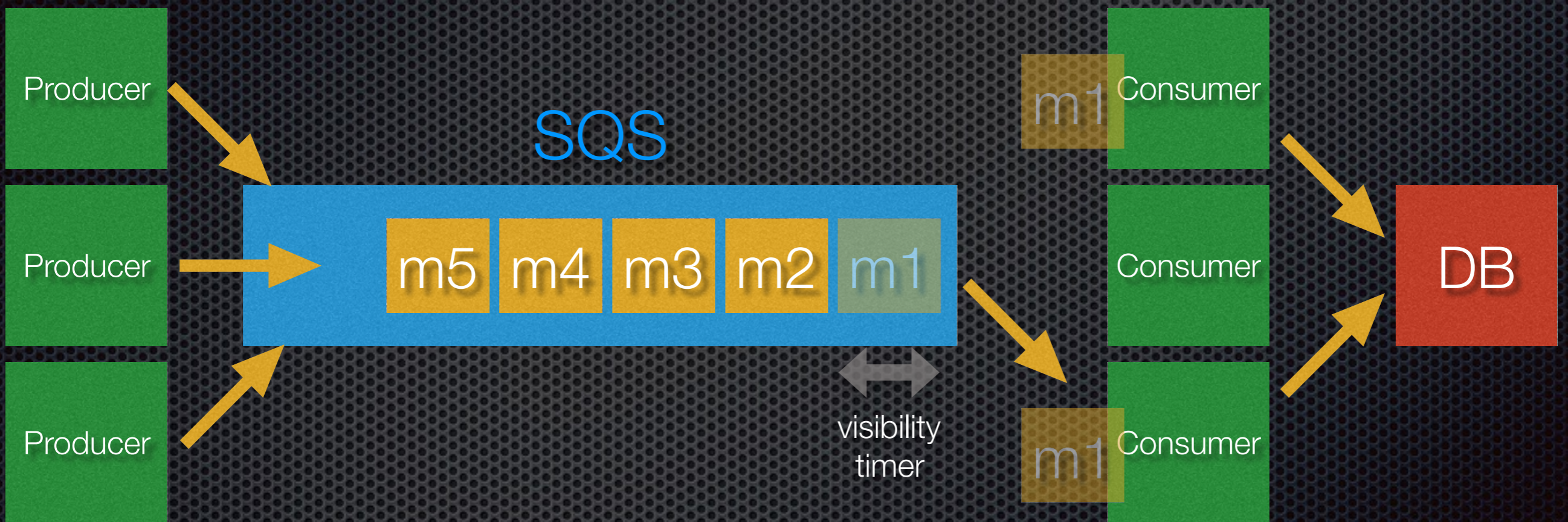
# SQS - Time Out Example

**Step 3:** A Visibility Timeout occurs & the message becomes visible again.



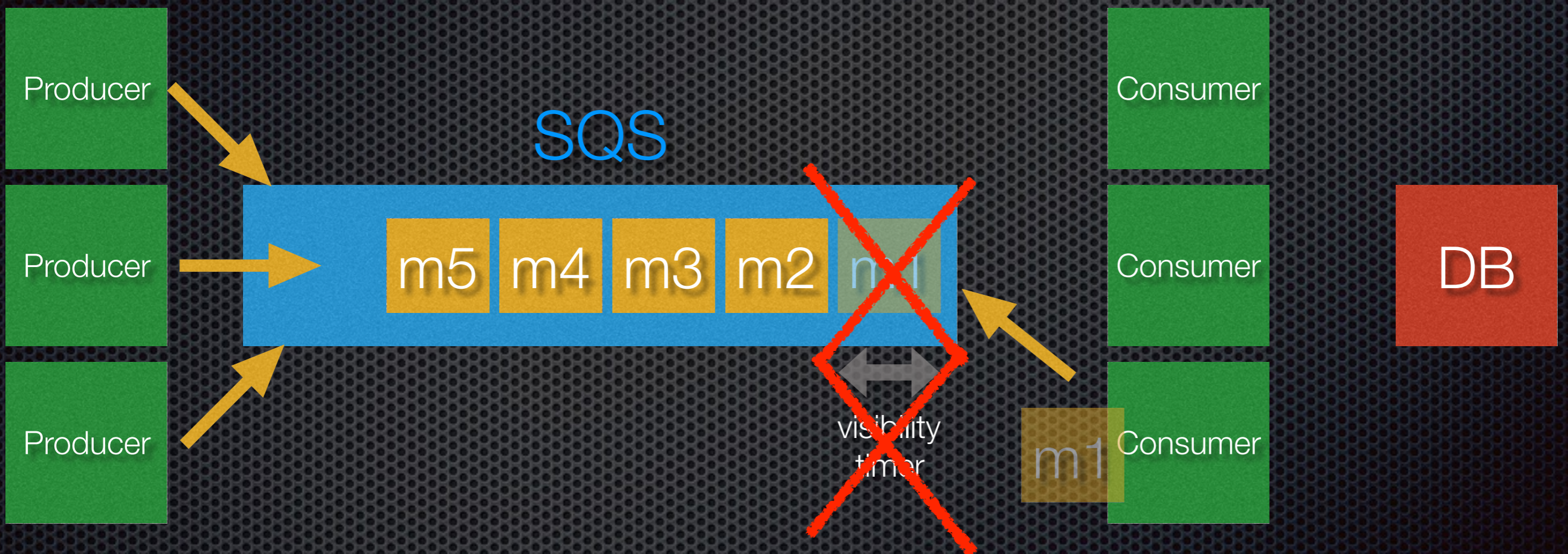
# SQS - Time Out Example

**Step 4:** Another consumer reads and persists the same message

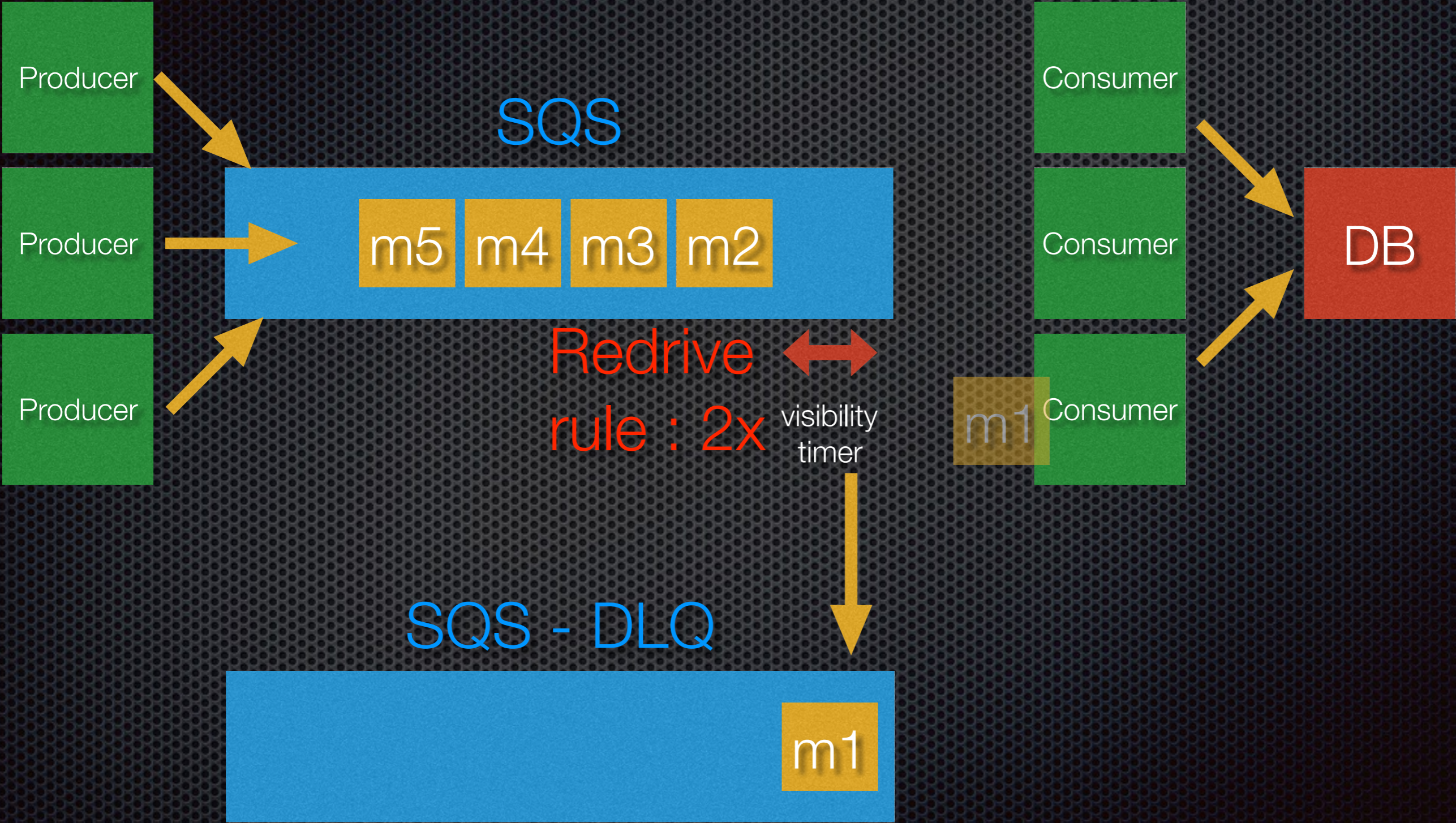


# SQS - Time Out Example

**Step 5:** Consumer ACKs message in SQS



# SQS - Dead Letter Queue



# SNS

Simple Notification Service

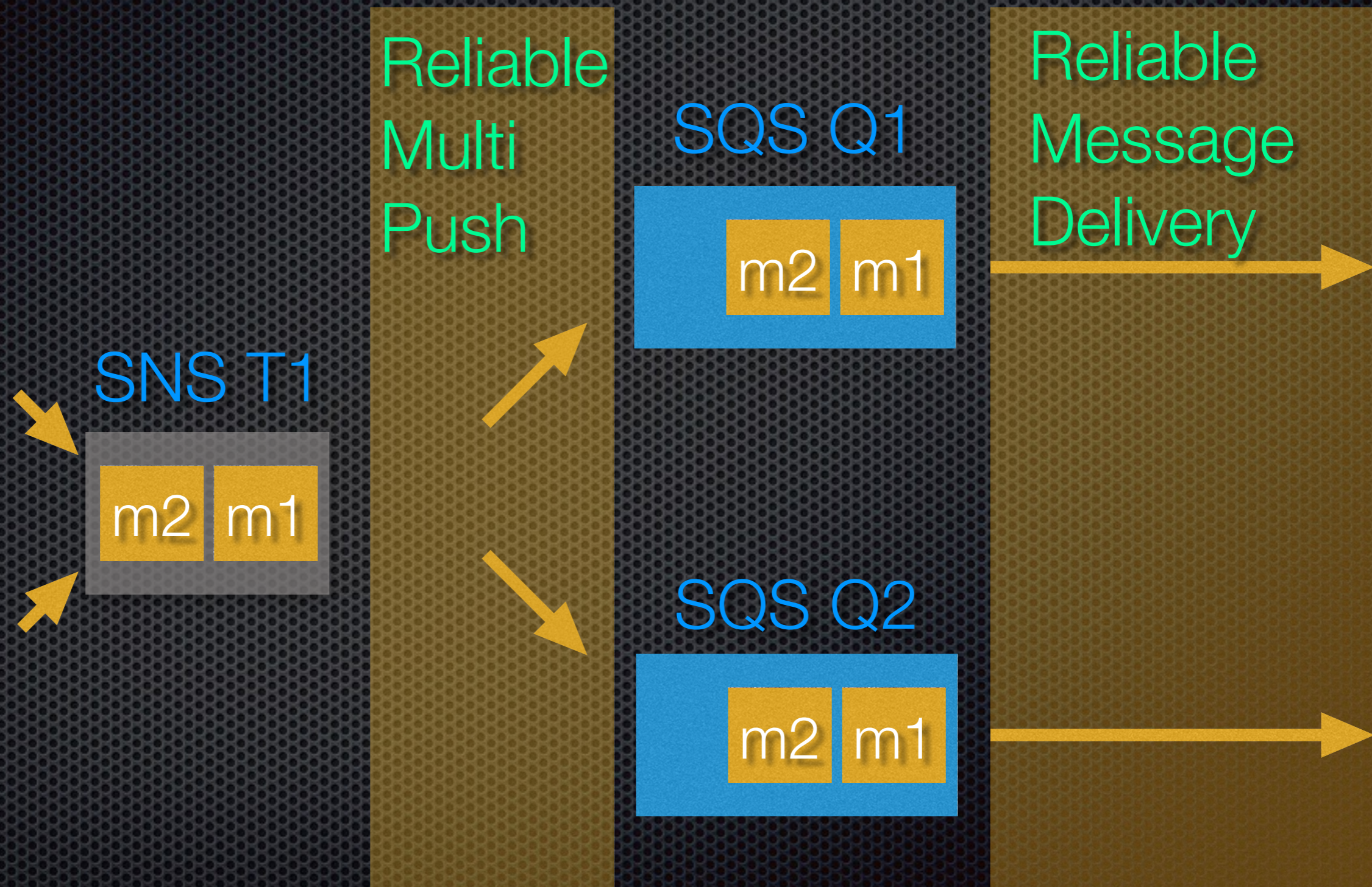
# SNS - Overview

- ✦ Highly Scalable, Highly Available, **Push-based Topic** Service
  - ✦ Whereas SQS ensures each message is seen by at least 1 consumer
    - ✦ SNS ensures that each message is seen by **every consumer**
    - ✦ **Reliable Multi-Push**
  - ✦ Whereas SQS is **pull-based**, SNS is **push-based**
    - ✦ There is no message retention & there is a finite retry count
    - ✦ **No** Reliable Message Delivery

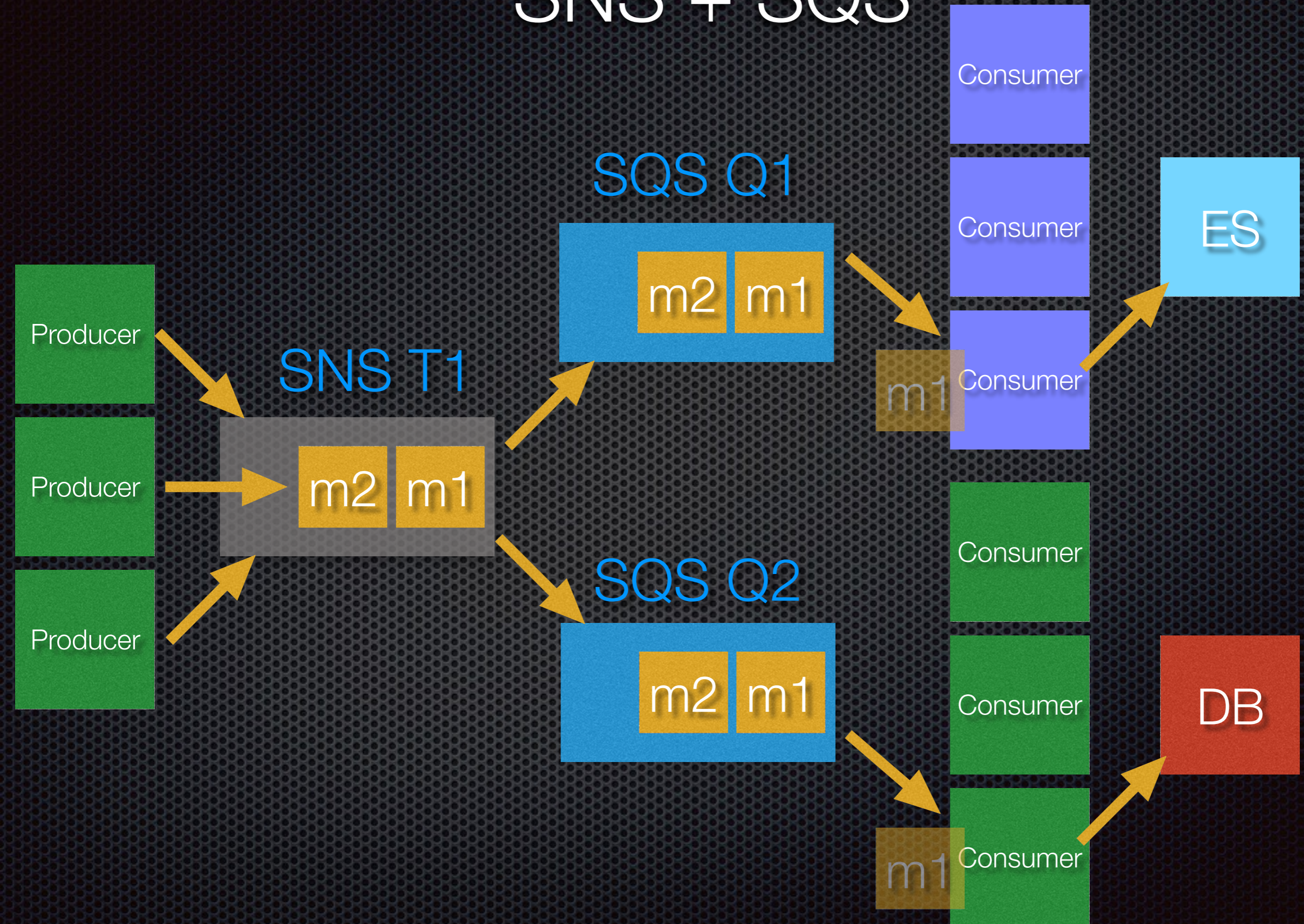
Can we work around this limitation?



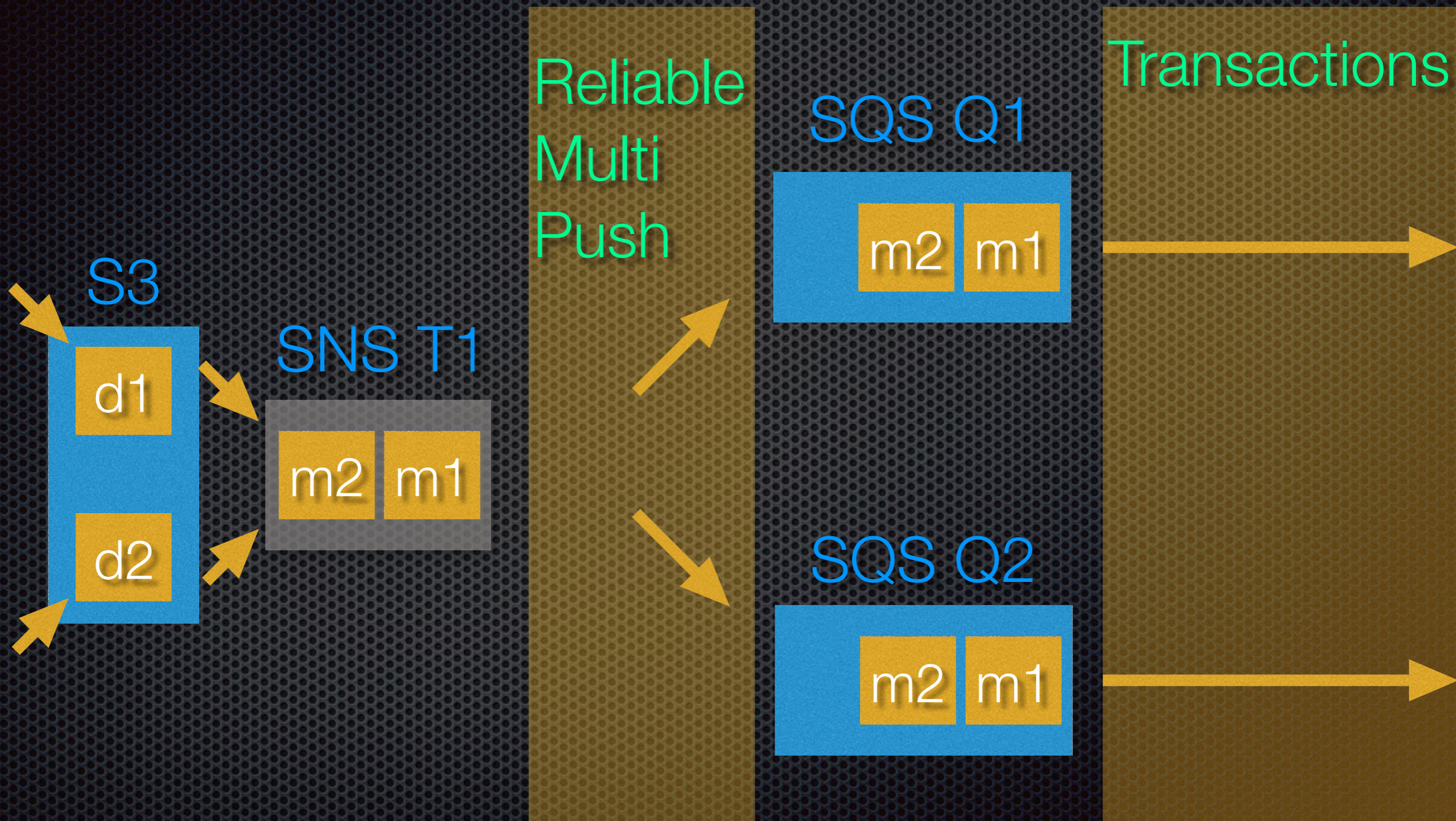
# SNS + SQS Design Pattern



# SNS + SQS



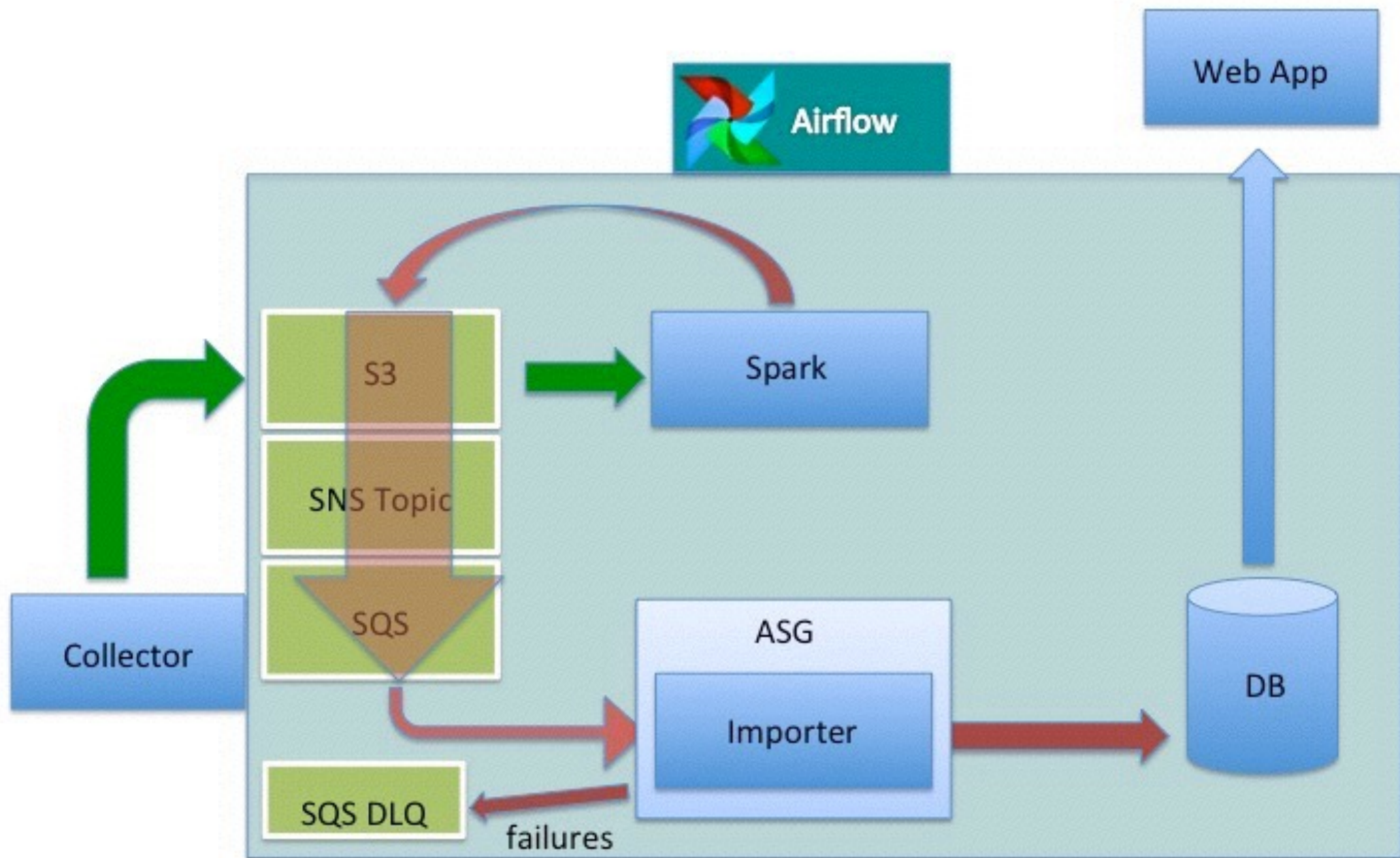
# S3 + SNS + SQS Design Pattern




# Batch Pipeline Architecture

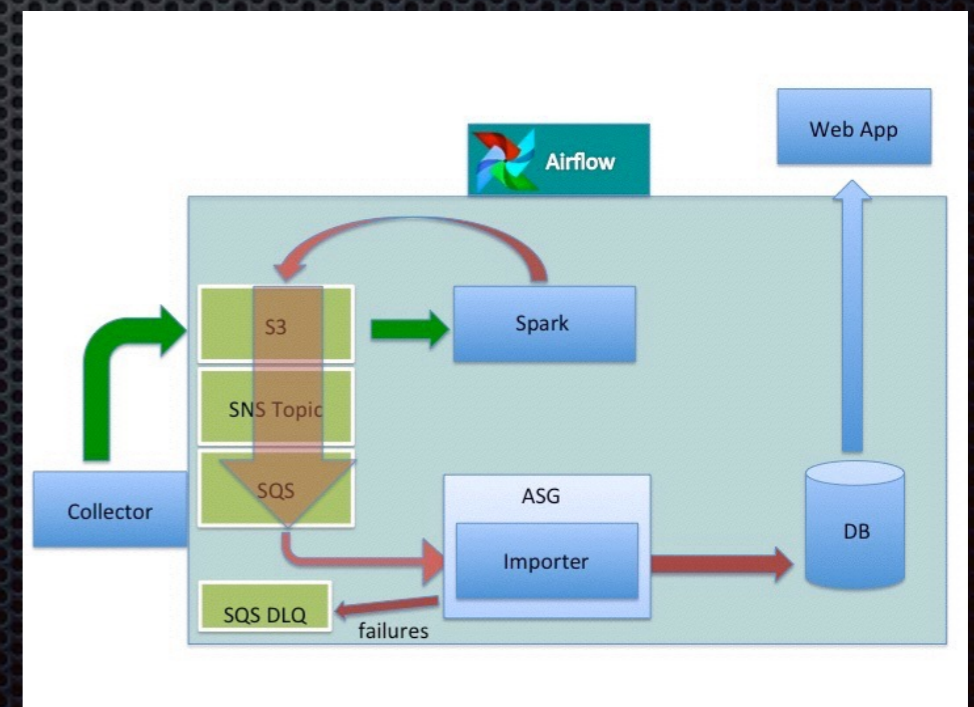
Putting the Pieces Together

# Architecture



# Architectural Elements

- ✦ A Schema-aware Data format for all data ([Avro](#))
- ✦ The entire pipeline is built from **Highly-Available/Highly-Scalable** components
  - ✦ [S3](#), [SNS](#), [SQS](#), [ASG](#), [EMR Spark](#) (exception [DB](#))
- ✦ The pipeline is never blocked because we use a **DLQ** for messages we cannot process
- ✦ We use queue-based auto-scaling to get high on-demand ingest rates
- ✦ We manage everything with Airflow 
- ✦ Every stage in the pipeline is idempotent
- ✦ Every stage in the pipeline is instrumented



# ASG

Auto Scaling Group

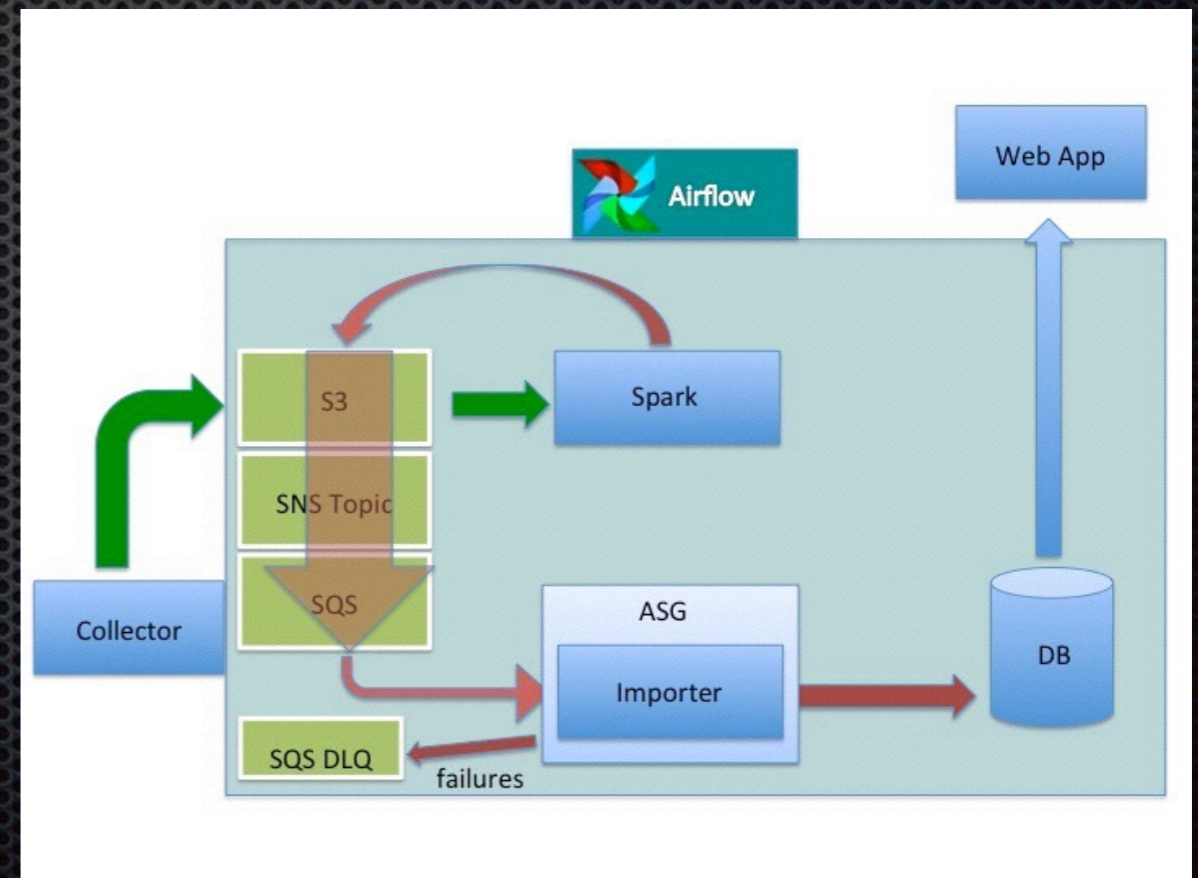
# ASG - Overview

## ✦ What is it?

- ✦ A means to automatically scale out/in clusters to handle variable load/traffic
- ✦ A means to keep a cluster/service always up
- ✦ Fulfills AWS's pay-per-use promise!

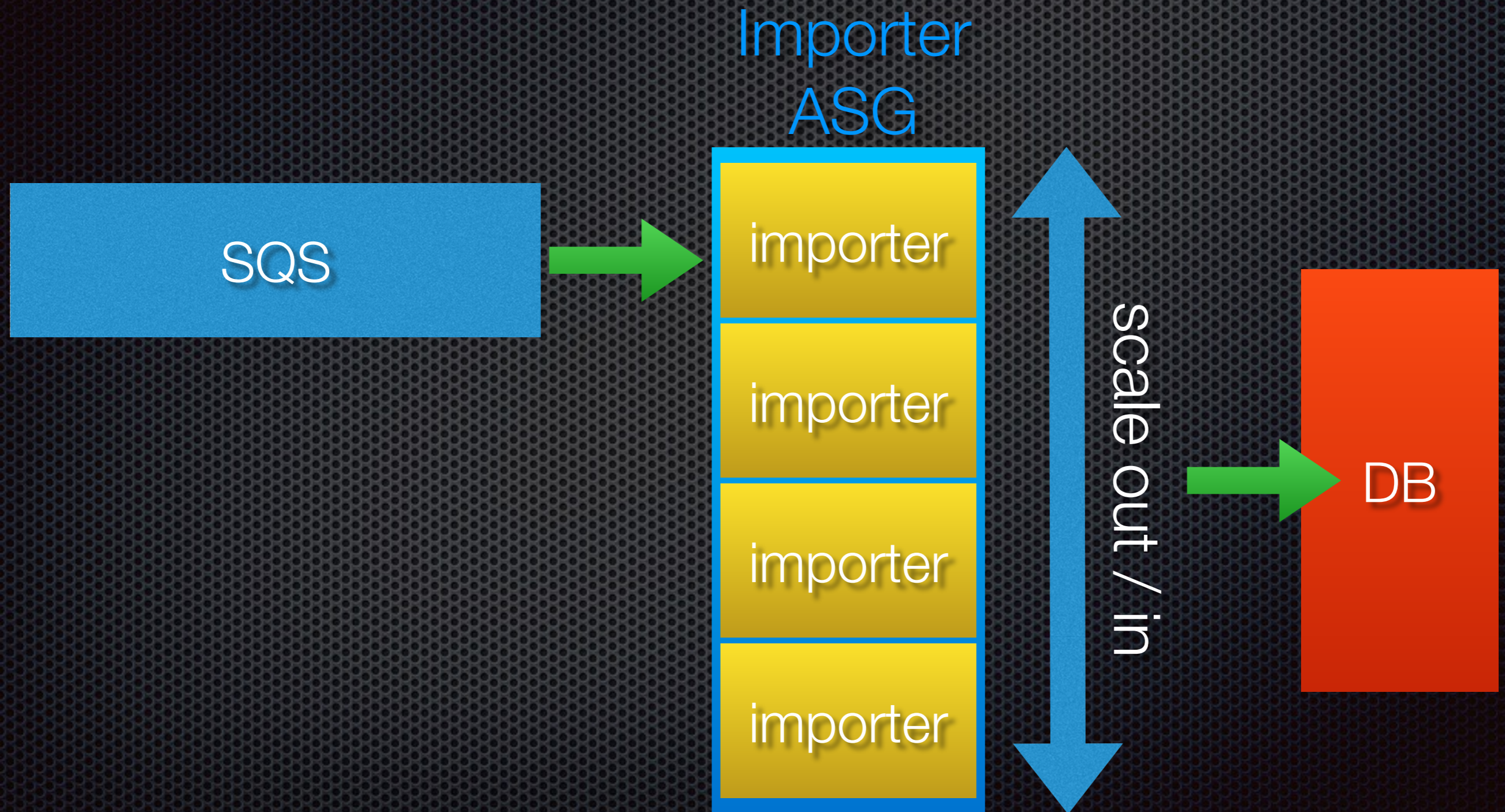
## ✦ When to use it?

- ✦ Feed-processing, web traffic load balancing, zone outage, etc...

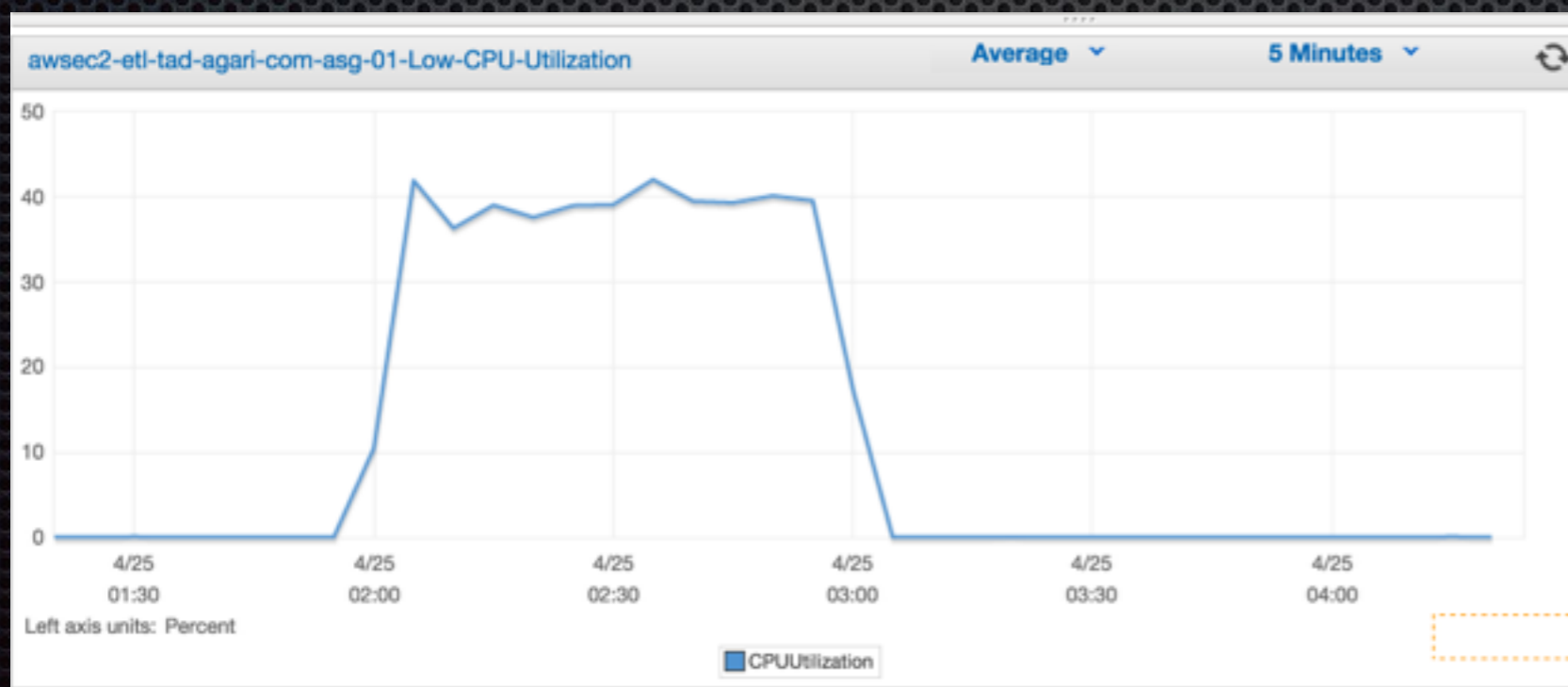
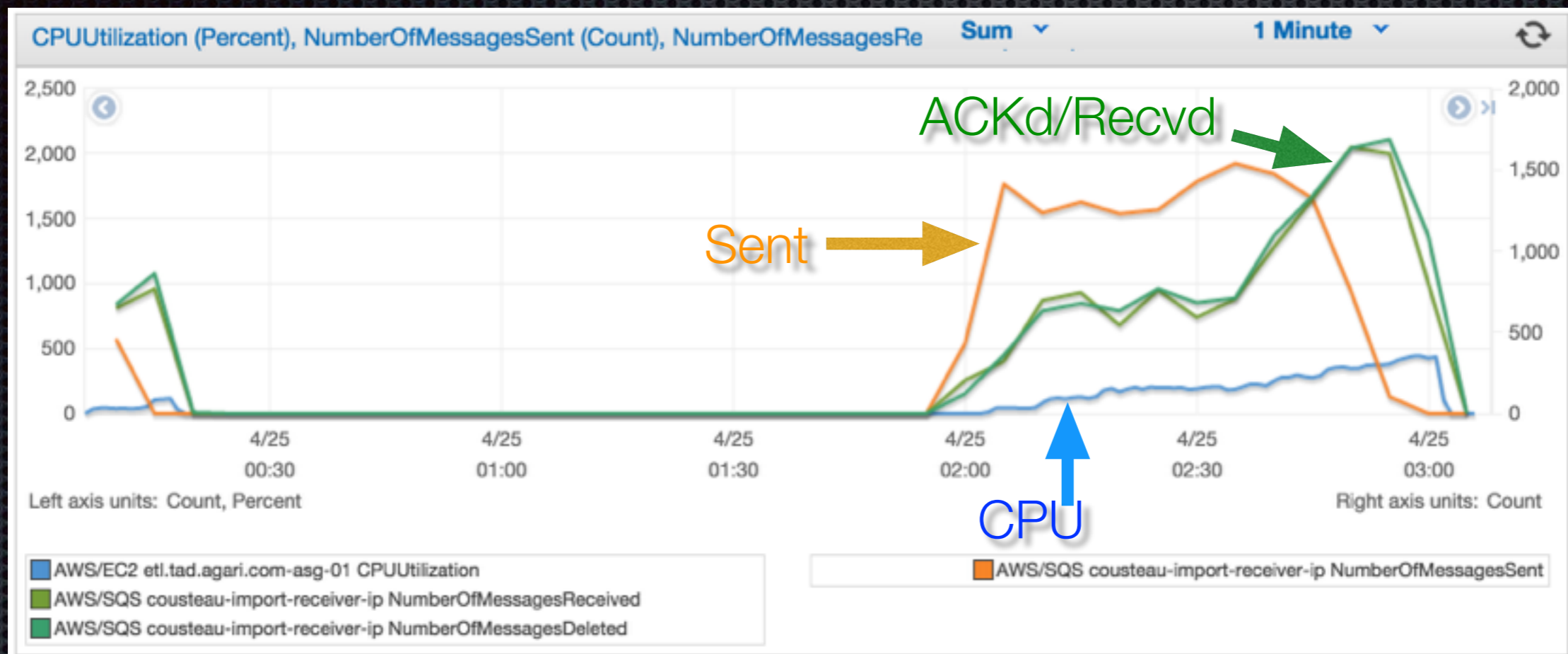




# ASG - Data Pipeline

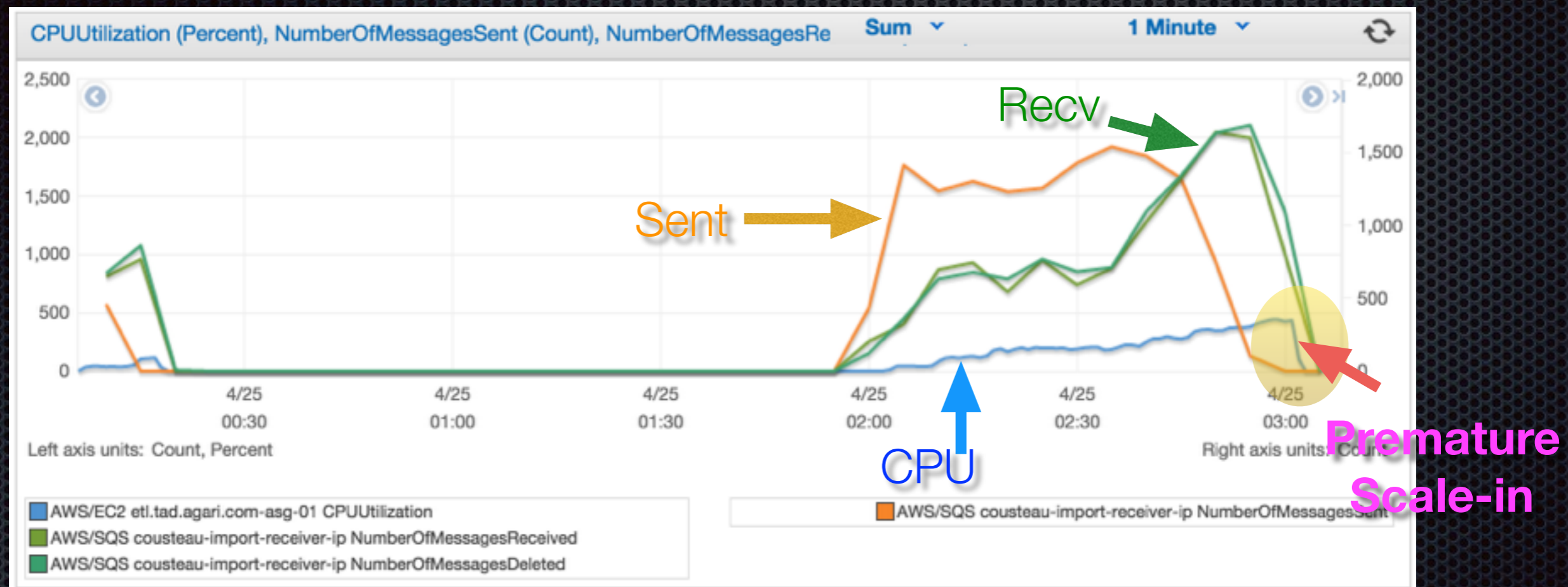


# ASG : CPU-based



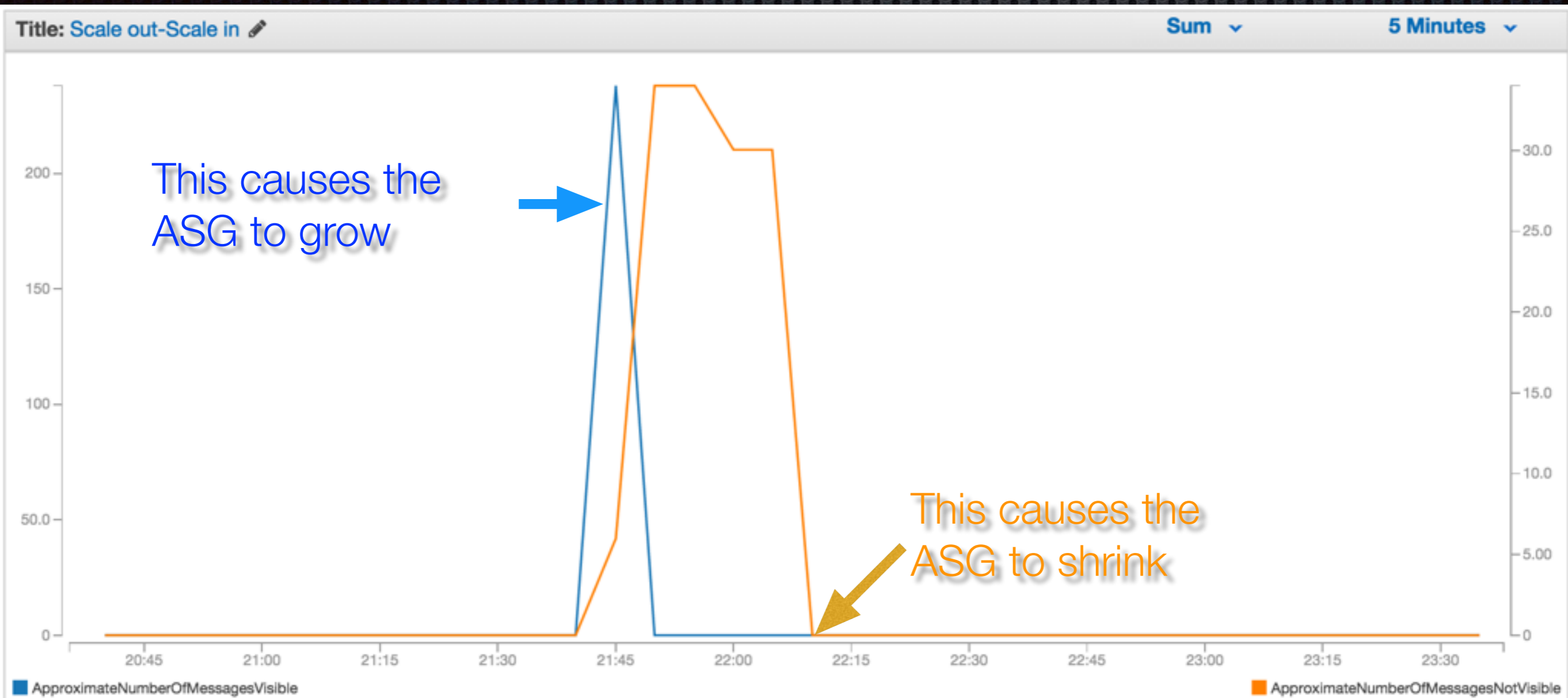
CPU-based auto-scaling is good at scaling in/out to keep the average CPU constant

# 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 resulting in a long time-to-drain for the queue!

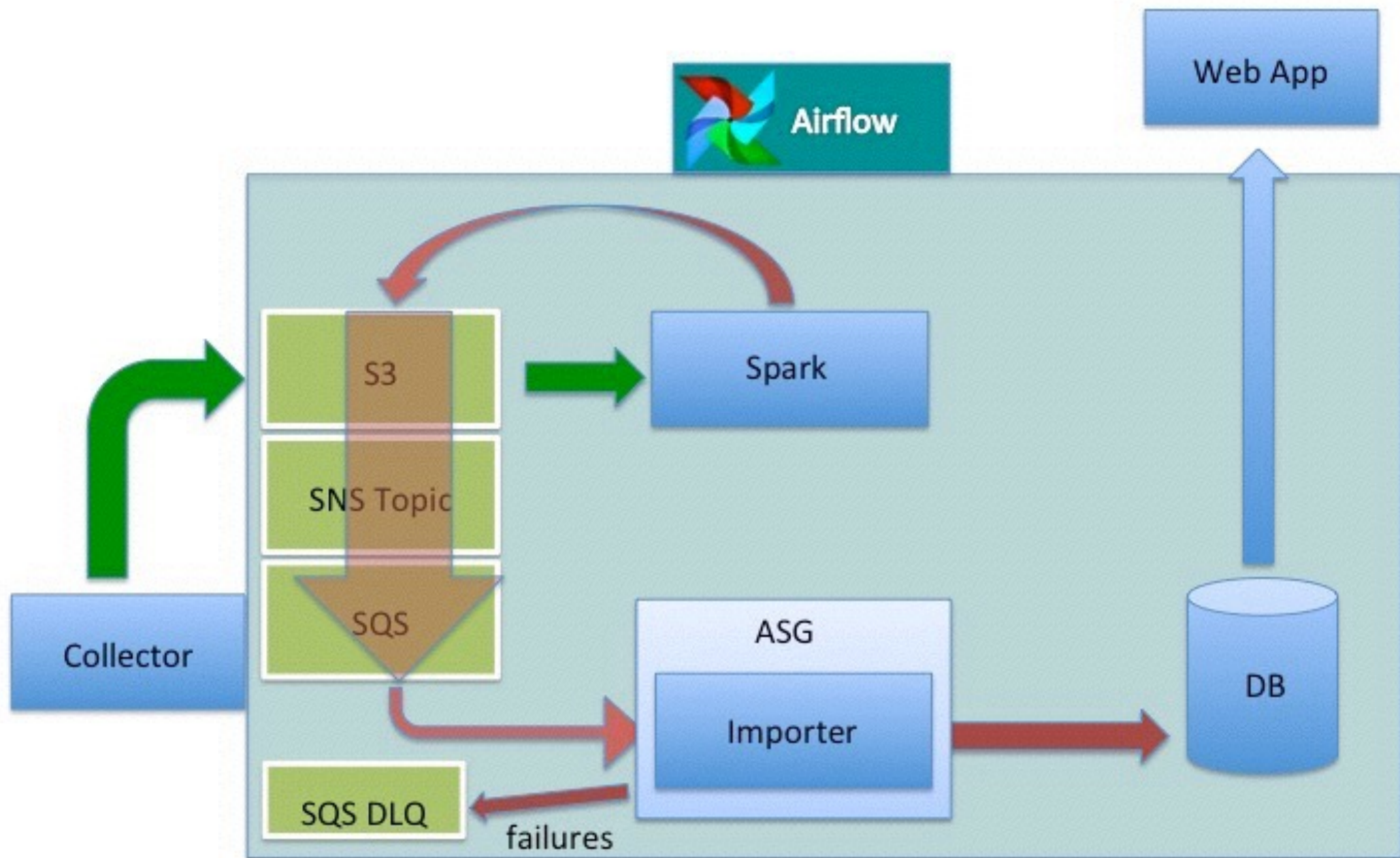
# ASG - Queue-based



**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)

# Architecture



# Reliable Hourly Job Scheduling

Workflow Automation & Scheduling

# Our Needs

- ✦ Historical Context
  - ✦ Our first cut at the pipeline used cron to schedule hourly runs of Spark
- ✦ Problem
  - ✦ We only knew if Spark succeeded. What if a downstream task failed?
- ✦ We needed something smarter than cron that
  - ✦ Reliably managed a graph of tasks (**DAG - Directed Acyclic Graph**)
    - ✦ Orchestrated hourly runs of that DAG
    - ✦ Retried failed tasks
    - ✦ Tracked the performance of each run and its tasks
    - ✦ Reported on failure/success of runs

Airflow 

Workflow Automation & Scheduling



# Airflow - DAG Dashboard

Airflow: It's easy to manage multiple DAGs

The screenshot shows the Airflow DAG Dashboard. The top navigation bar includes the Airflow logo, 'DAGs', 'Data Profiling', 'Browse', 'Admin', and 'Docs' menus, along with the current time '00:31 UTC'. The main heading is 'DAGs'. Below the heading, there is a 'Show 10 entries' dropdown and a search box. The main content is a table with the following columns: 'DAG', 'Schedule', 'Owner', 'Statuses', and 'Links'. Each row represents a DAG with its name, schedule, owner, and a set of status indicators (circles) representing different task states. The 'Links' column contains icons for various actions like refresh, view, and delete.

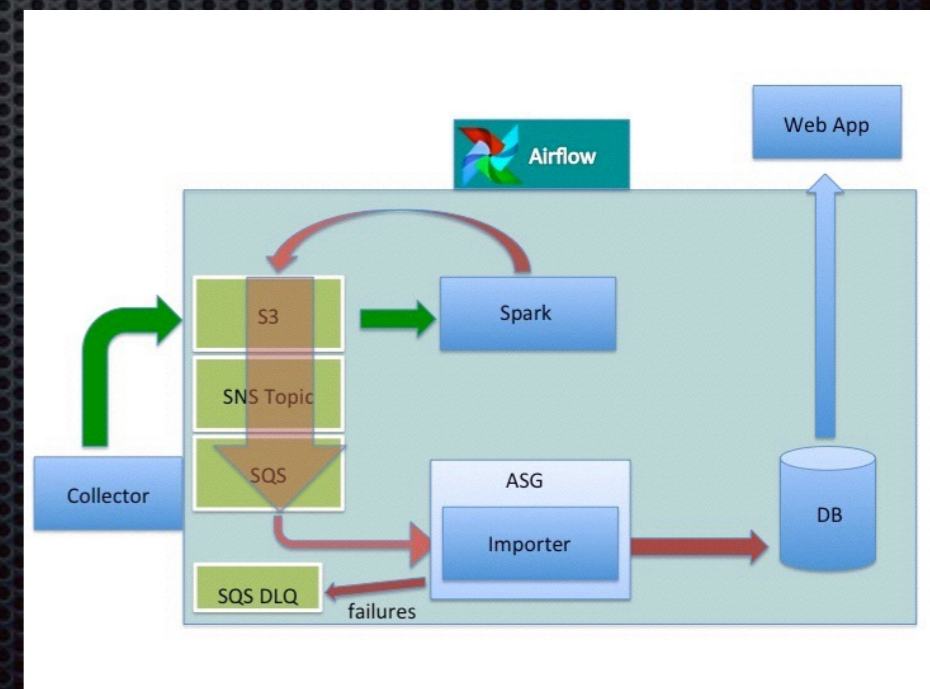
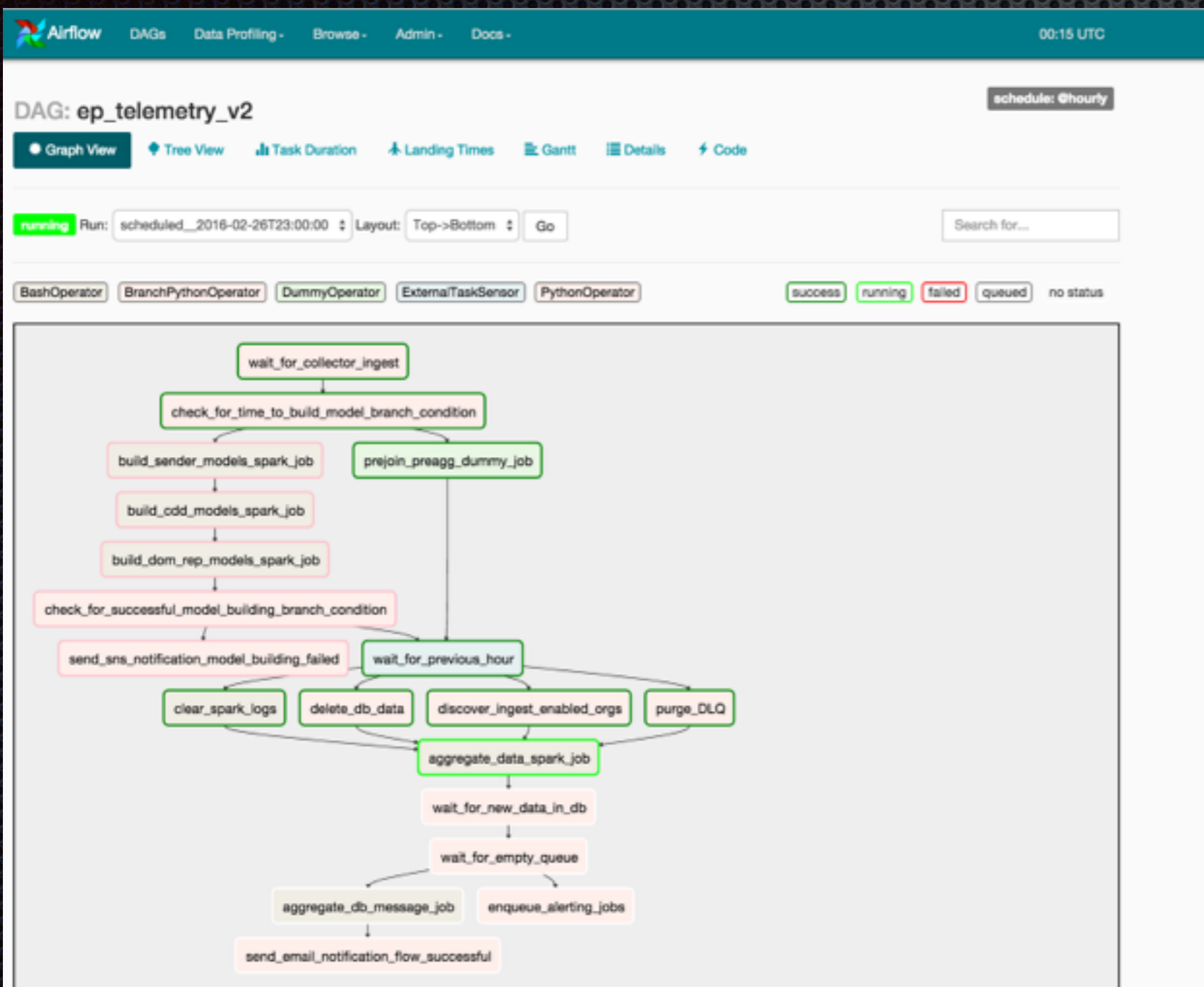
		DAG	Schedule	Owner	Statuses	Links
	<input checked="" type="checkbox"/>	db_backup_v1	0 4 ***	aflury	25	
	<input checked="" type="checkbox"/>	emr_forwarders	0 8 ***	kmandich	118	
	<input checked="" type="checkbox"/>	emr_model_building	0 1 ***	kmandich	235	
	<input type="checkbox"/>	ep_model_building_v1	0 1 ***	sanand	253	
	<input type="checkbox"/>	ep_reload_data	None	sanand	2	
	<input checked="" type="checkbox"/>	ep_telemetry_v2	@hourly	sanand	30671  7	
	<input type="checkbox"/>	feedback_report	1 day, 0:00:00	kmandich	72	
	<input checked="" type="checkbox"/>	generate_spooofs	1 day, 0:00:00	kmandich	258  1  1	
	<input checked="" type="checkbox"/>	refresh_asn_data	0 10 ***	kmandich	292	
	<input checked="" type="checkbox"/>	refresh_dns_cache	0 10 1,15 **	kmandich		

Showing 1 to 10 of 10 entries

Previous **1** Next

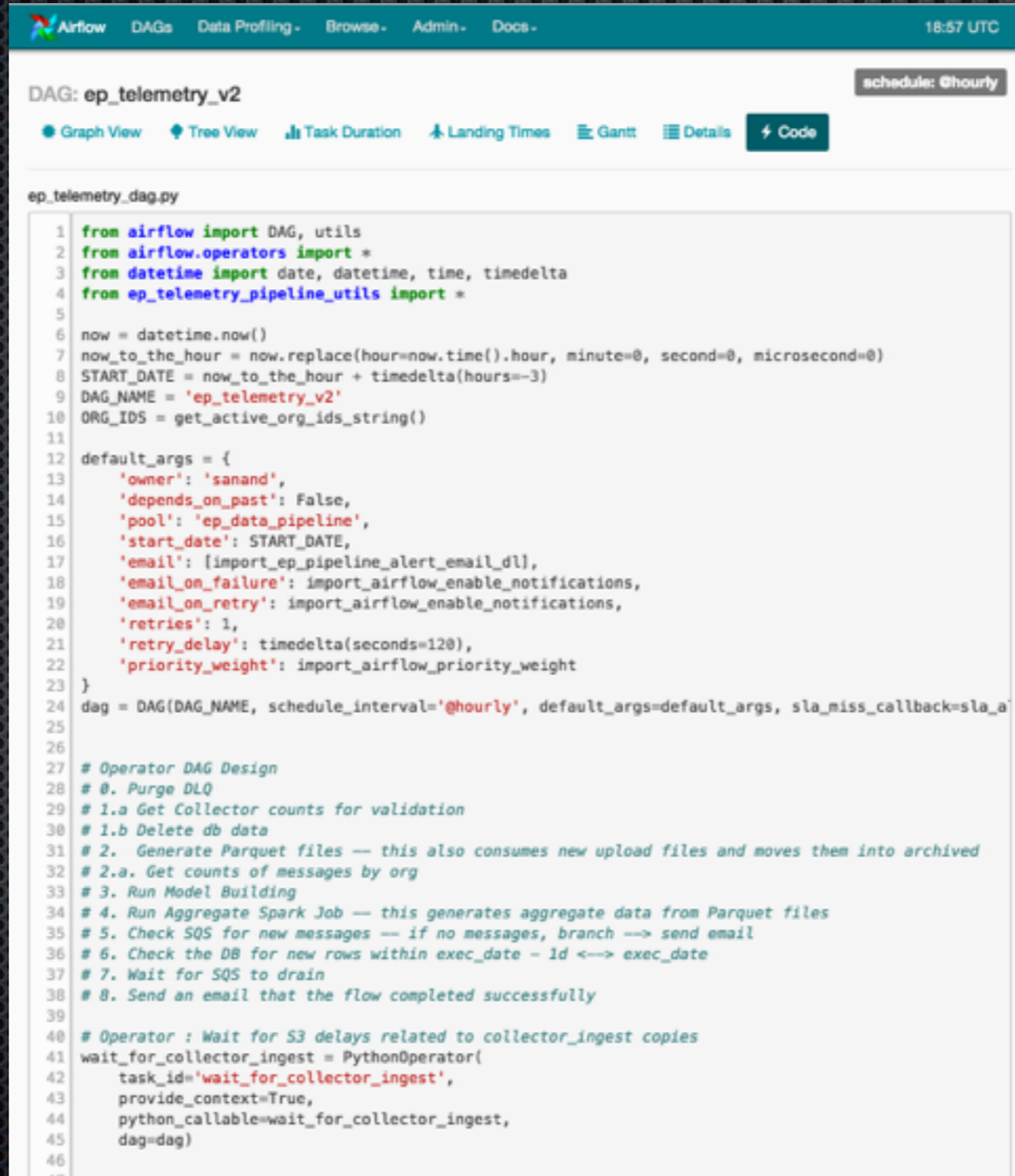
# Airflow - Authoring DAGs

## Airflow: Visualizing a DAG



# Airflow - Authoring DAGs

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

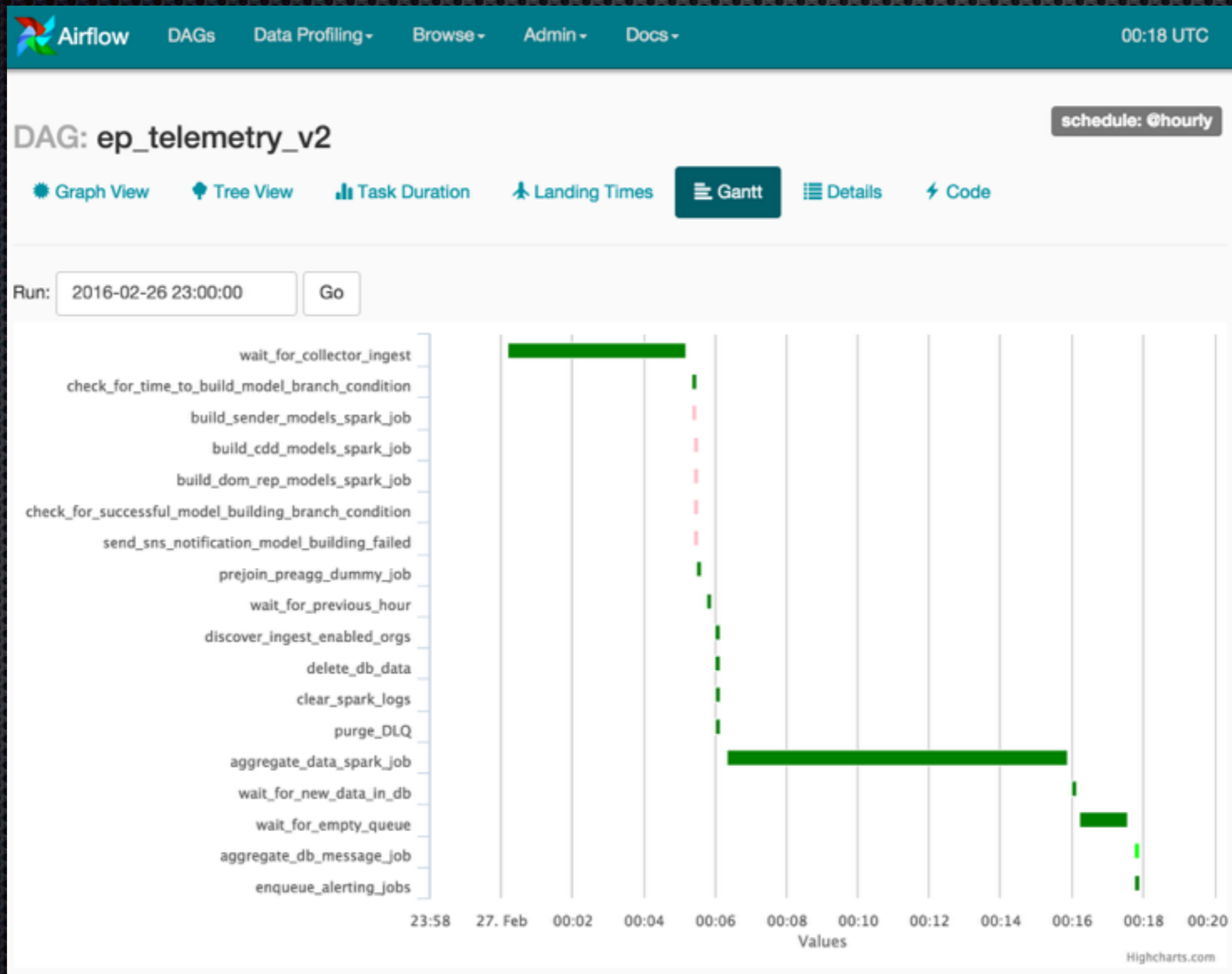


The screenshot shows the Airflow web interface for a DAG named 'ep\_telemetry\_v2'. The interface includes a navigation bar with links for DAGs, Data Profiling, Browse, Admin, and Docs, along with the current time '18:57 UTC'. Below the navigation bar, there are tabs for 'Graph View', 'Tree View', 'Task Duration', 'Landing Times', 'Gantt', 'Details', and 'Code'. The 'Code' tab is selected, displaying the Python code for the DAG in a file named 'ep\_telemetry\_dag.py'. The code includes imports for Airflow DAG and utils, datetime, and a custom pipeline utils module. It defines a DAG with a schedule interval of '@hourly' and a set of default arguments. The DAG is then instantiated with these arguments and a list of tasks, including a PythonOperator for waiting for collector ingest delays.

```
1 from airflow import DAG, utils
2 from airflow.operators import *
3 from datetime import date, datetime, time, timedelta
4 from ep_telemetry_pipeline_utils import *
5
6 now = datetime.now()
7 now_to_the_hour = now.replace(hour=now.time().hour, minute=0, second=0, microsecond=0)
8 START_DATE = now_to_the_hour + timedelta(hours=-3)
9 DAG_NAME = 'ep_telemetry_v2'
10 ORG_IDS = get_active_org_ids_string()
11
12 default_args = {
13     'owner': 'sanand',
14     'depends_on_past': False,
15     'pool': 'ep_data_pipeline',
16     'start_date': START_DATE,
17     'email': [import_ep_pipeline_alert_email_dl],
18     'email_on_failure': import_airflow_enable_notifications,
19     'email_on_retry': import_airflow_enable_notifications,
20     'retries': 1,
21     'retry_delay': timedelta(seconds=120),
22     'priority_weight': import_airflow_priority_weight
23 }
24 dag = DAG(DAG_NAME, schedule_interval='@hourly', default_args=default_args, sla_miss_callback=sla_a
25
26
27 # Operator DAG Design
28 # 0. Purge DLQ
29 # 1.a Get Collector counts for validation
30 # 1.b Delete db data
31 # 2. Generate Parquet files — this also consumes new upload files and moves them into archived
32 # 2.a. Get counts of messages by org
33 # 3. Run Model Building
34 # 4. Run Aggregate Spark Job — this generates aggregate data from Parquet files
35 # 5. Check SQS for new messages — if no messages, branch —> send email
36 # 6. Check the DB for new rows within exec_date - 1d <=> exec_date
37 # 7. Wait for SQS to drain
38 # 8. Send an email that the flow completed successfully
39
40 # Operator : Wait for S3 delays related to collector_ingest copies
41 wait_for_collector_ingest = PythonOperator(
42     task_id='wait_for_collector_ingest',
43     provide_context=True,
44     python_callable=wait_for_collector_ingest,
45     dag=dag)
46
47
```

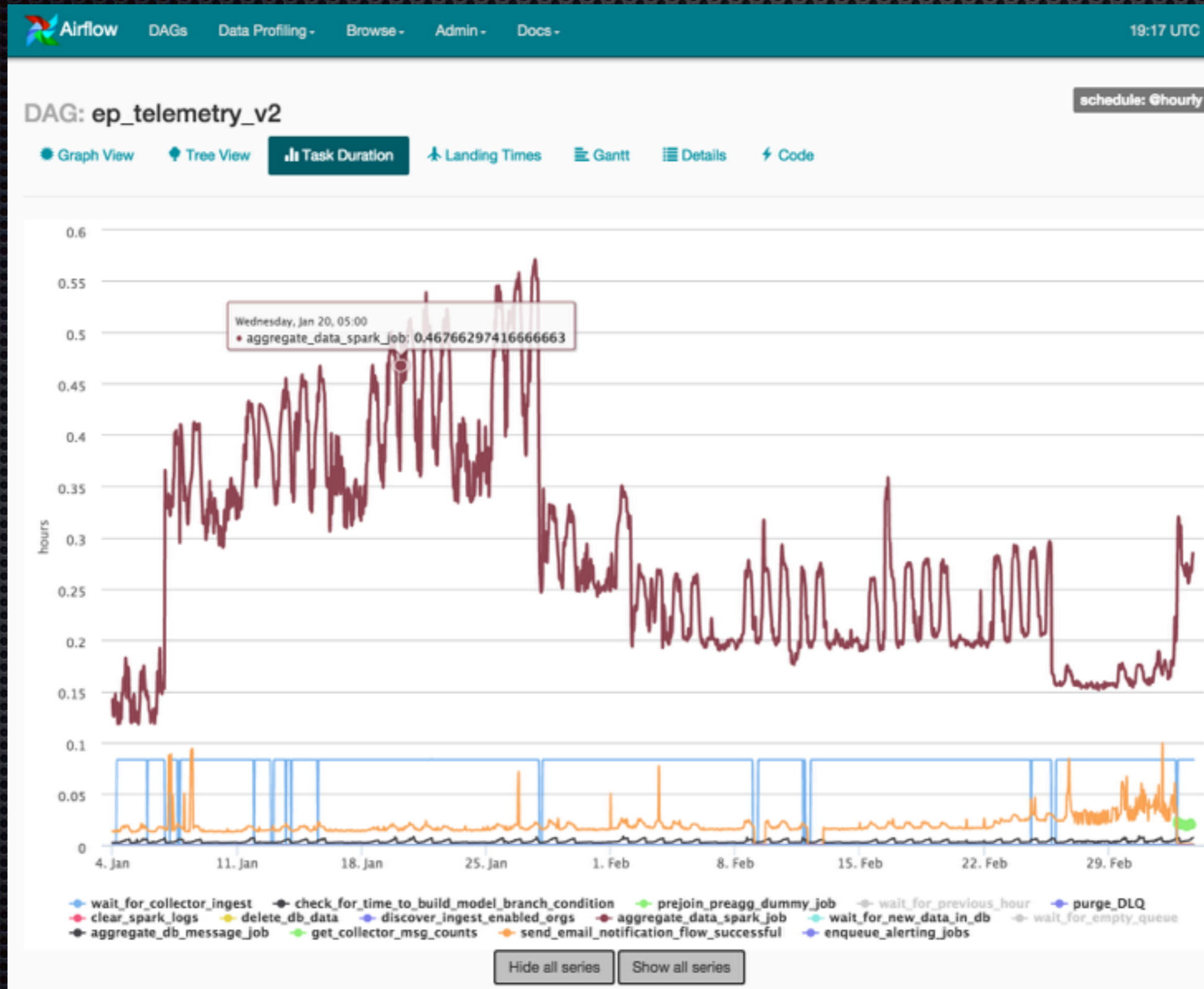
# Airflow - Performance Insights

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



# Airflow - Performance Insights

Airflow: ...And we can easily see performance trends over time



# Airflow - Alerting

Slack interface showing a channel named **#ep-ops** with 13 members. The channel description is "A channel with info that can help you resolve VictorOps alerts".

**Today**

- Airflow BOT** 10:00 AM: EP\_STAGE - ep\_telemetry\_v2 SLA Miss for task `send_email_notification_flow_successful` on `2016-02-26T18:00:00`
- sid** 10:02 AM: So, we need to catch up here.. any reason we don't want to just mark success for the many hours it is behind?
- Airflow BOT** 11:00 AM: EP\_STAGE - ep\_telemetry\_v2 SLA Miss for task `send_email_notification_flow_successful` on `2016-02-26T19:00:00`
- Airflow BOT** 5:27 PM: ep\_telemetry\_v2 on `etl-00.ep-old.prod.agari.com` completed `2016-02-03 00:00:00` with **High Discrepancies**

**February 3rd**

- Airflow BOT** 8:48 PM: ep\_telemetry\_v2 on `workflow-00.ep.stage.agari.com` completed `2016-02-26 03:00:00` with 1 DLQs : **Sample Exception**

VictorOps interface showing alert details for Airflow tasks. The interface includes a sidebar with user avatars (Andrew Flury, Christopher Haag, Spencer Sun) and a main area with alert details.

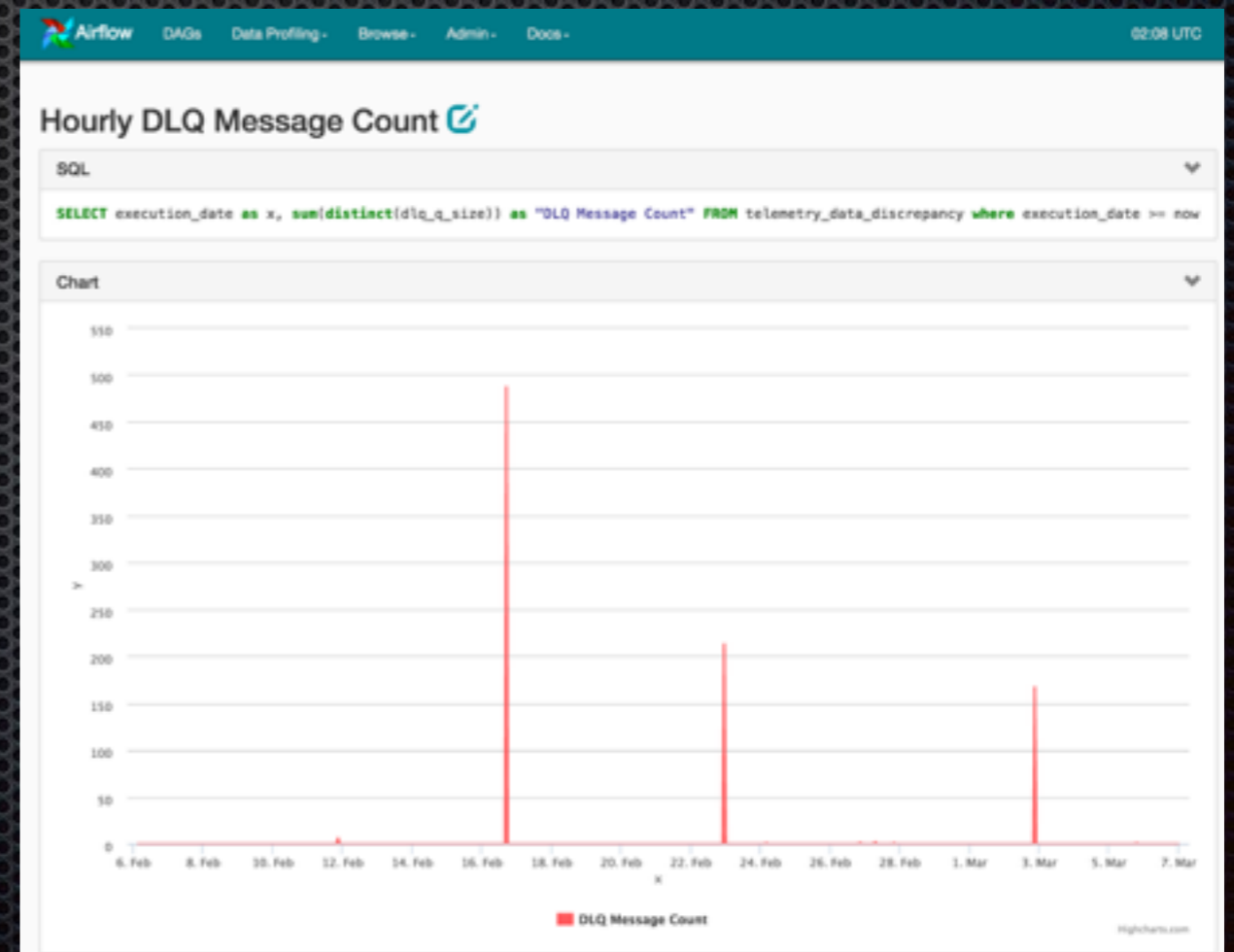
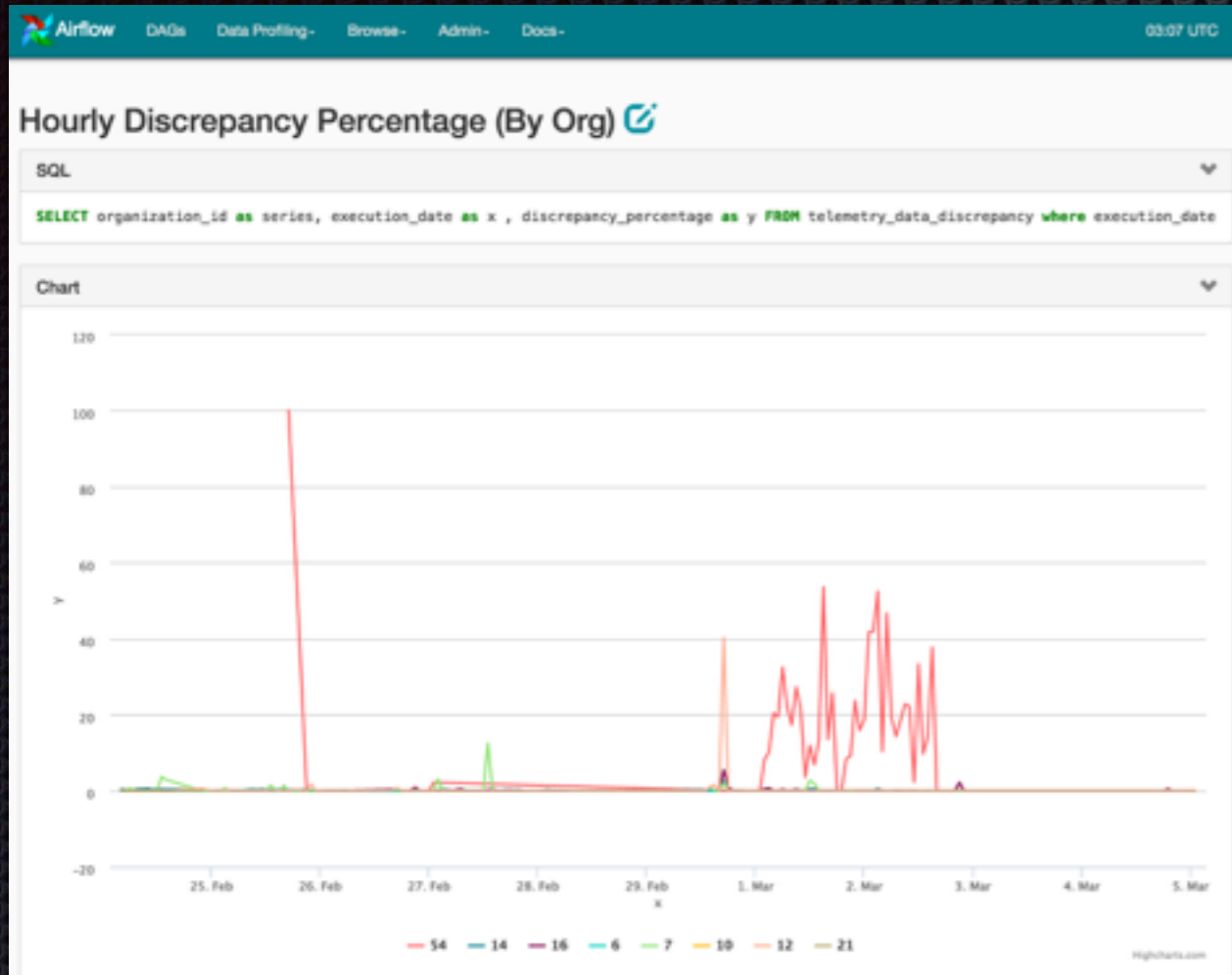
**Alert #102:** #102\_ALERT - Prod EP Pipeline Discrepancy: workflow-00.ep.prod.agari.com/agari.log [Agari Data Inc]

Time	Feb 26, 2016 13:...
Email	ALERT - Prod EP ...
State	OK
Resolved By	aflury

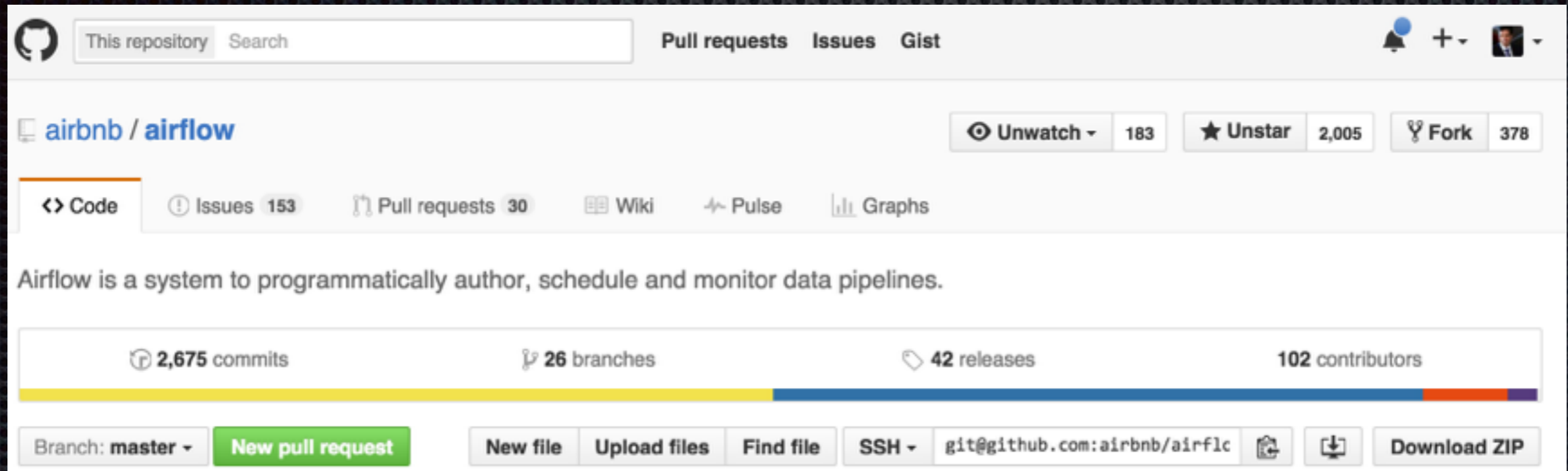
**Alert #101:** #101 airflow/telemetry/SLA\_miss

Time	Feb 26, 2016 13:0...
API	airflow/telemetry...
State	INFO
Resolved By	SYSTEM

# Airflow - Monitoring



# Airflow - Join the Community



The screenshot shows the GitHub repository page for Airbnb's Airflow project. The repository name is 'airbnb / airflow'. It has 183 watchers, 2,005 stars, and 378 forks. The repository is currently on the 'master' branch. The page shows 2,675 commits, 26 branches, 42 releases, and 102 contributors. The description states: 'Airflow is a system to programmatically author, schedule and monitor data pipelines.' The page also includes navigation links for Code, Issues (153), Pull requests (30), Wiki, Pulse, and Graphs. At the bottom, there are buttons for 'New pull request', 'New file', 'Upload files', 'Find file', 'SSH', 'git@github.com:airbnb/airflc', and 'Download ZIP'.

- ✦ With >30 Companies, >100 Contributors , and >2500 Commits, Airflow is growing rapidly!
- ✦ We are looking for more contributors to help support the community!
- ✦ Disclaimer : I'm a maintainer on the project

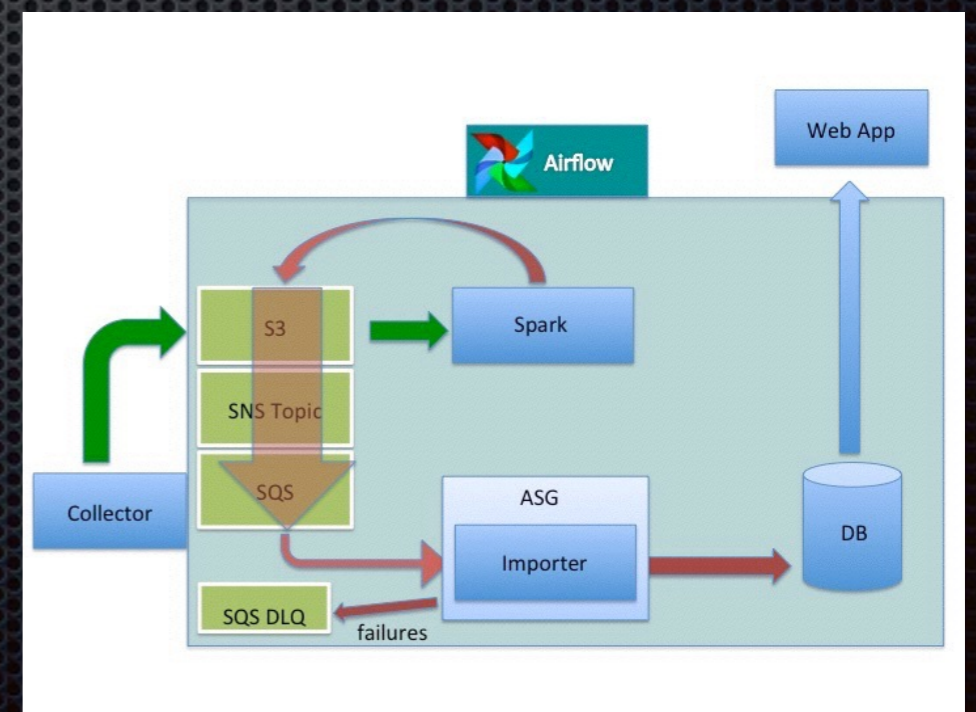


# Design Goal Scorecard

## Are We Meeting Our Design Goals?

# Desirable Qualities of a Resilient Data Pipeline

- Scalable
- Available
- Instrumented, Monitored, & Alert-enabled
- Quickly Recoverable



# Desirable Qualities of a Resilient Data Pipeline

- Scalable
  - Build using scalable components from AWS
    - SQS, SNS, S3, ASG, EMR Spark
  - Exception = DB (WIP)
- Available
  - Build using available components from AWS
  - Airflow for reliable job scheduling
- Instrumented, Monitored, & Alert-enabled
  - Airflow
- Quickly Recoverable
  - Airflow, DLQs, ASGs, Spark & DB



# Questions? (@r39132)

