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

- 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

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

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

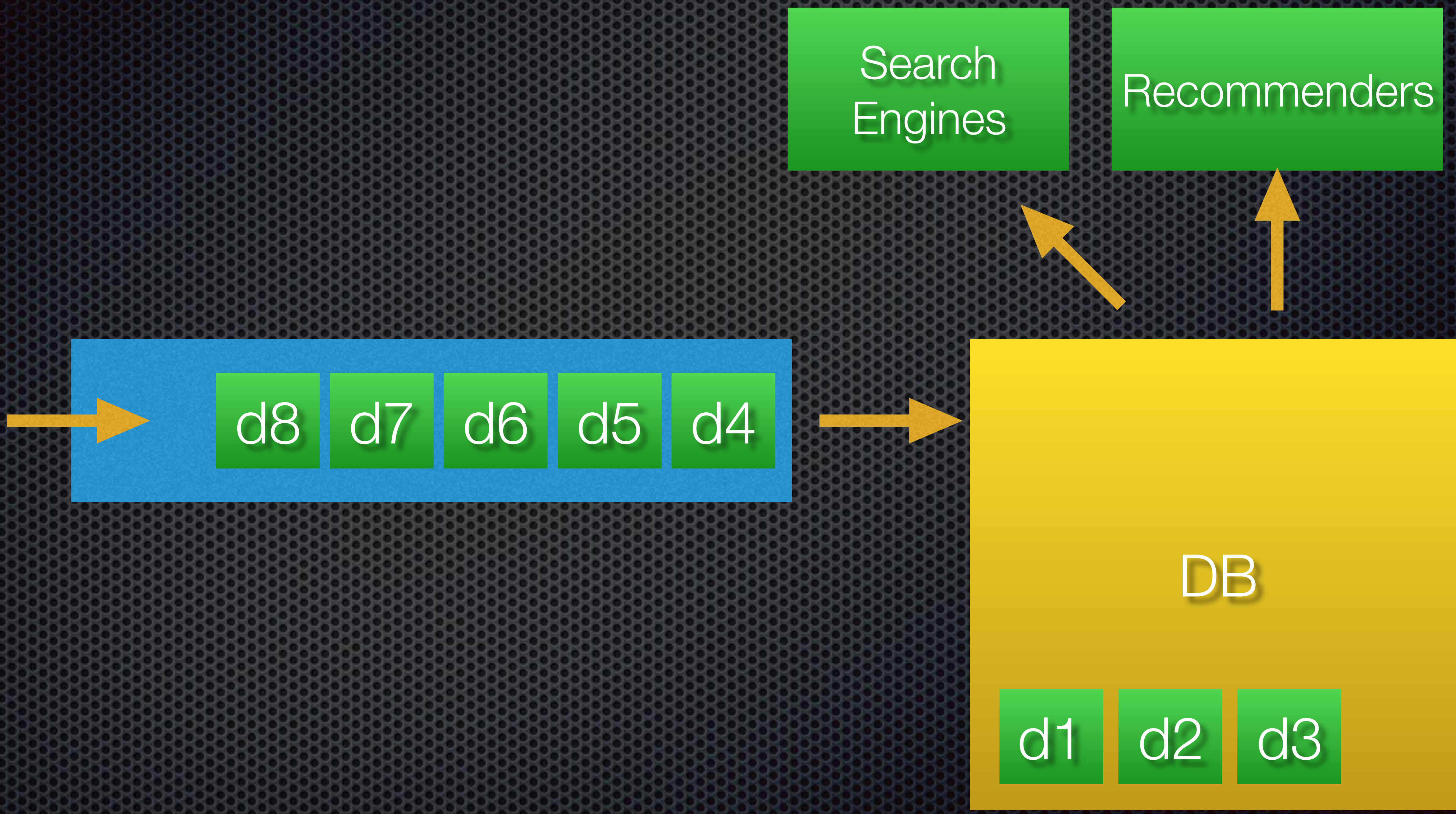
Predictive

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

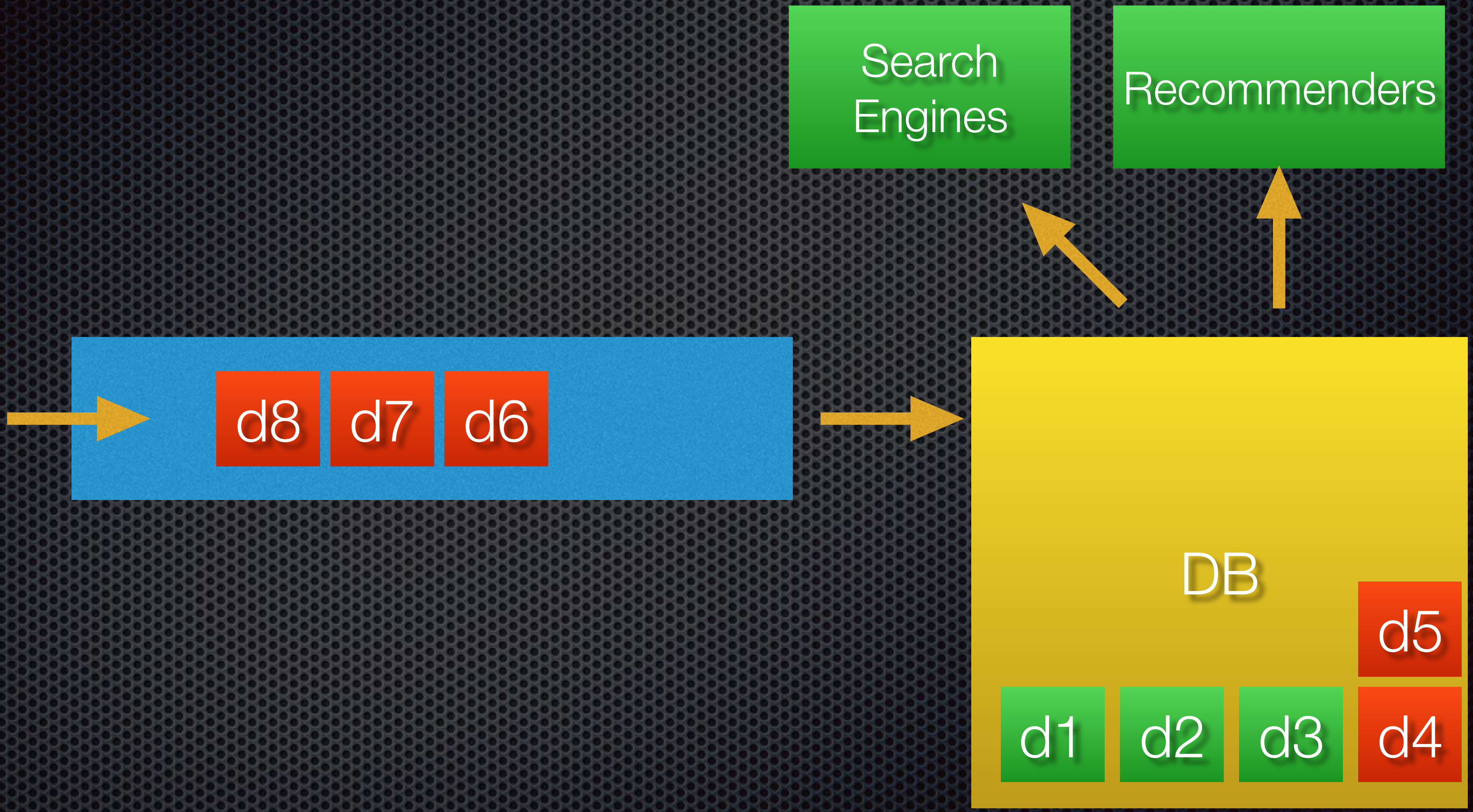
- audience : Customers

- downtime? : < 1 hour

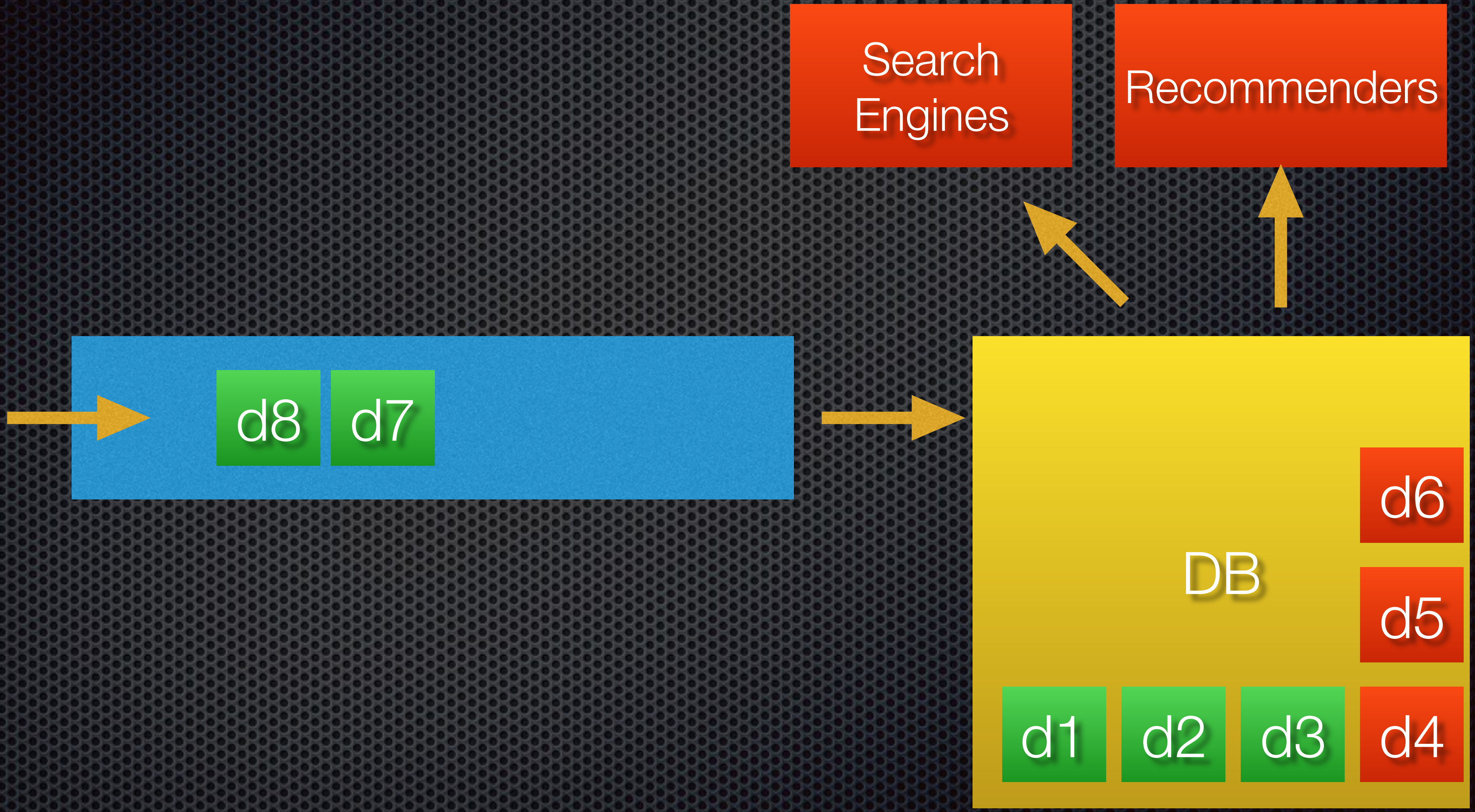
Why Do We Care About Resilience?



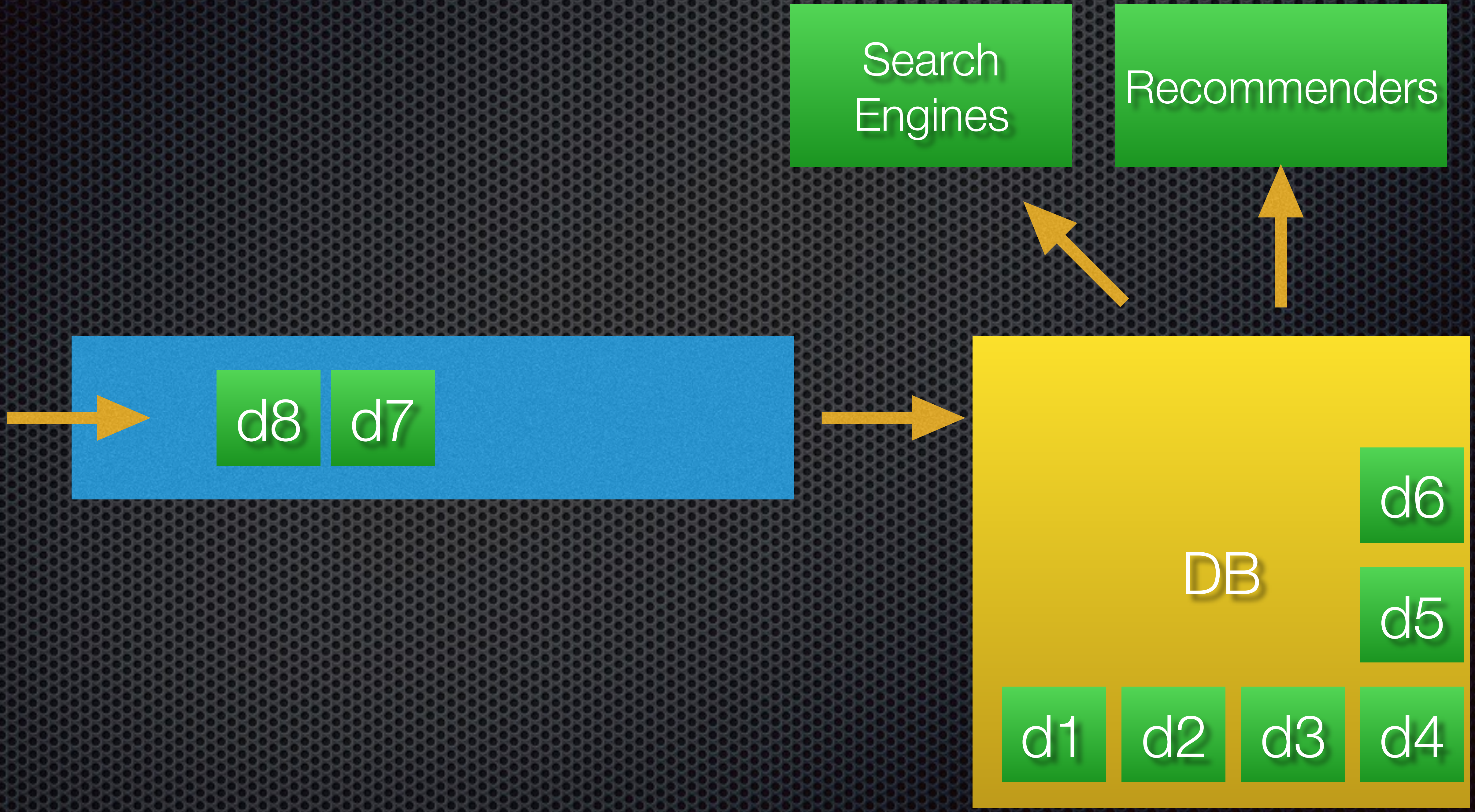
Why Do We Care About Resilience?



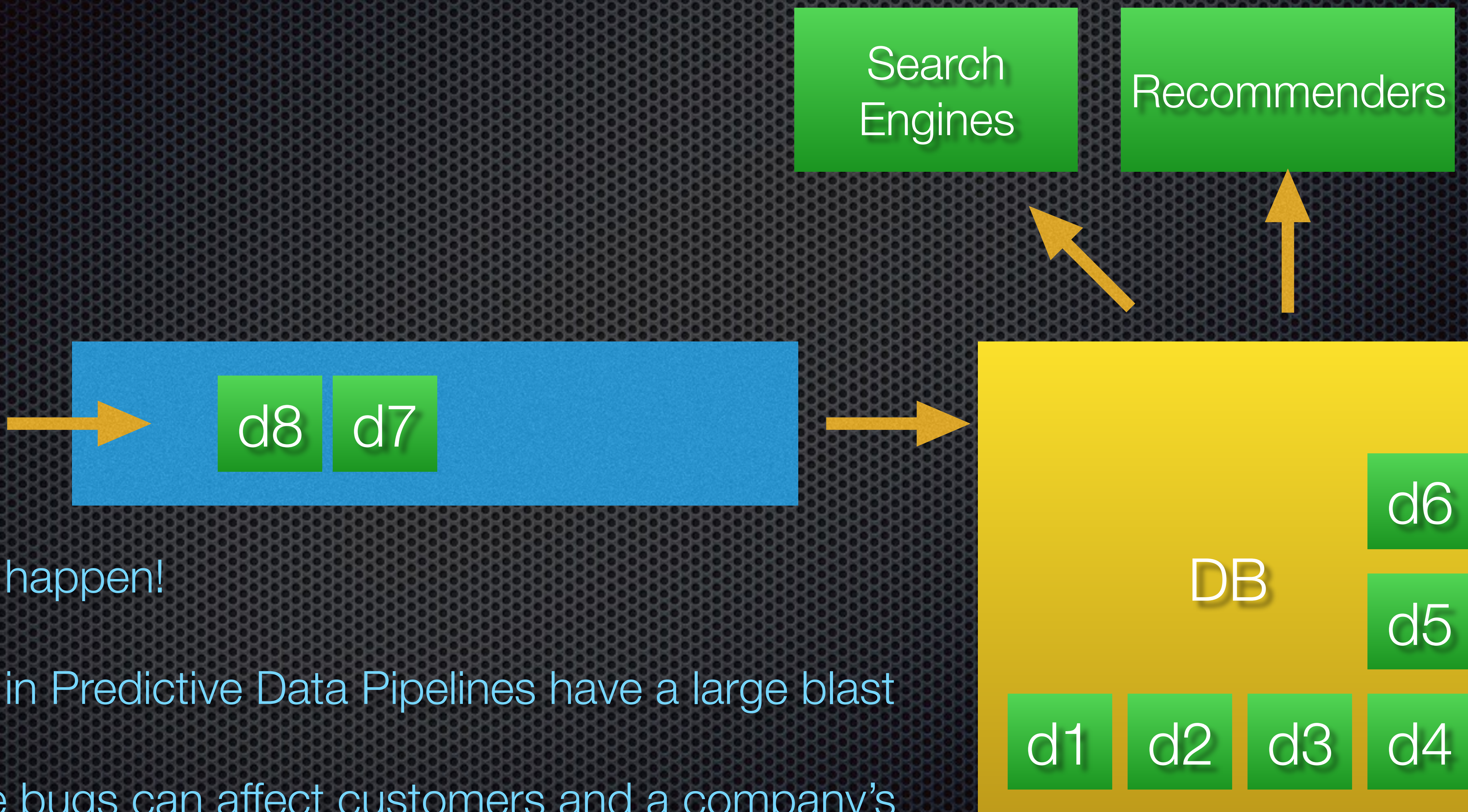
Why Do We Care About Resilience?



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

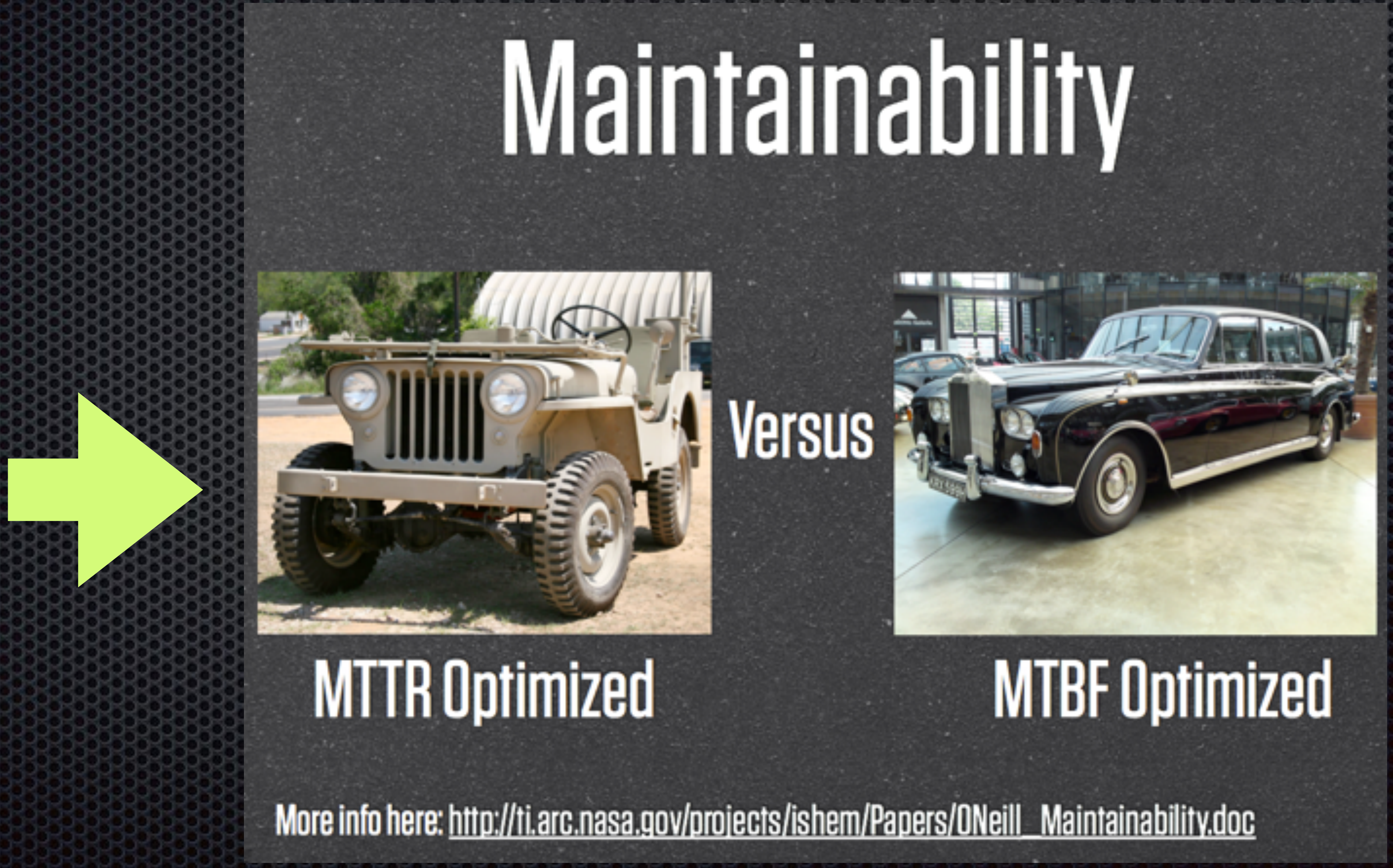
- Instrument
  - Instrument Correctness & Timeliness SLA metrics at each stage of the pipeline
- Monitor
  - Continuously monitor that SLA metrics fall within acceptable bounds (i.e. pre-defined SLAs)
- Alert
  - Alert when we miss SLAs

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



Implementation

Using AWS to meet Design Goals

SQS

Simple Queue Service

SQS - Overview

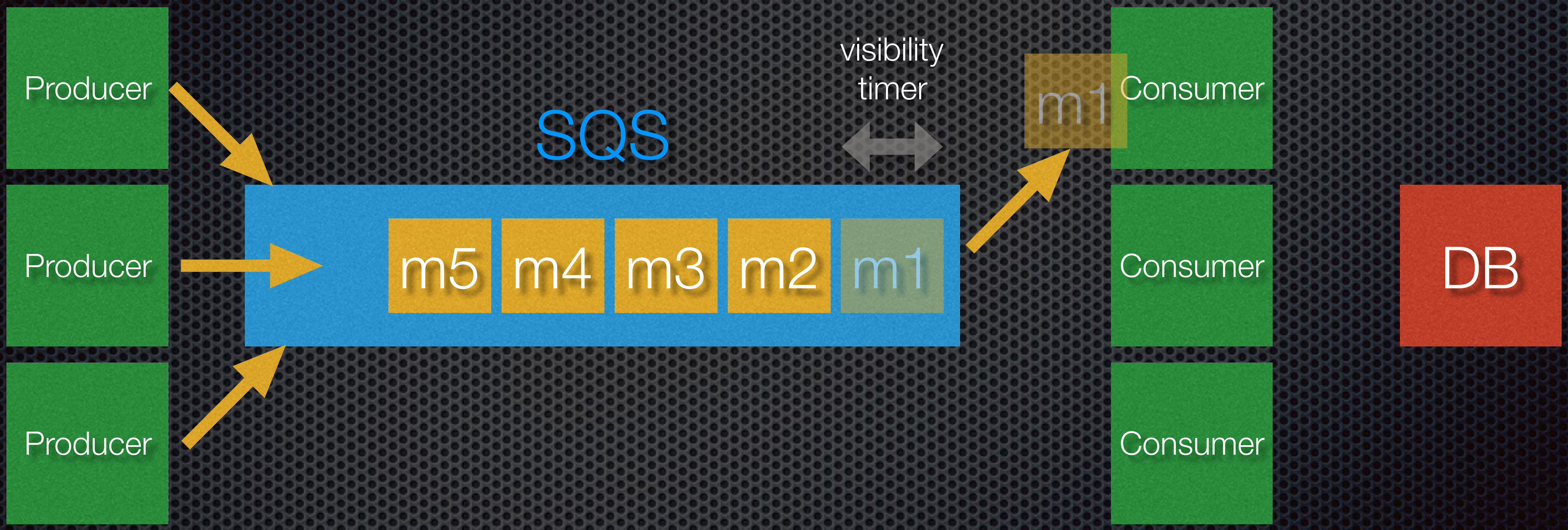
- AWS's low-latency, highly scalable, highly available message queue
  - Infinitely Scalable Queue (though not FIFO)
  - Low End-to-end latency (generally sub-second)
  - Pull-based

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

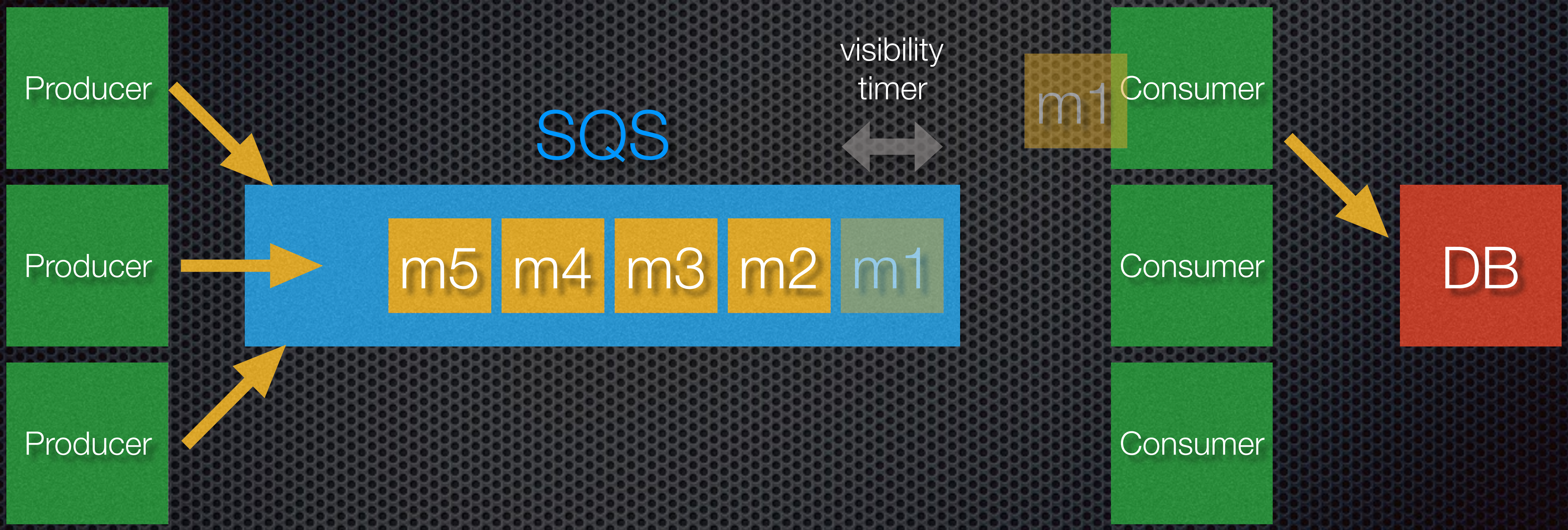
SQS - Typical Operation Flow

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



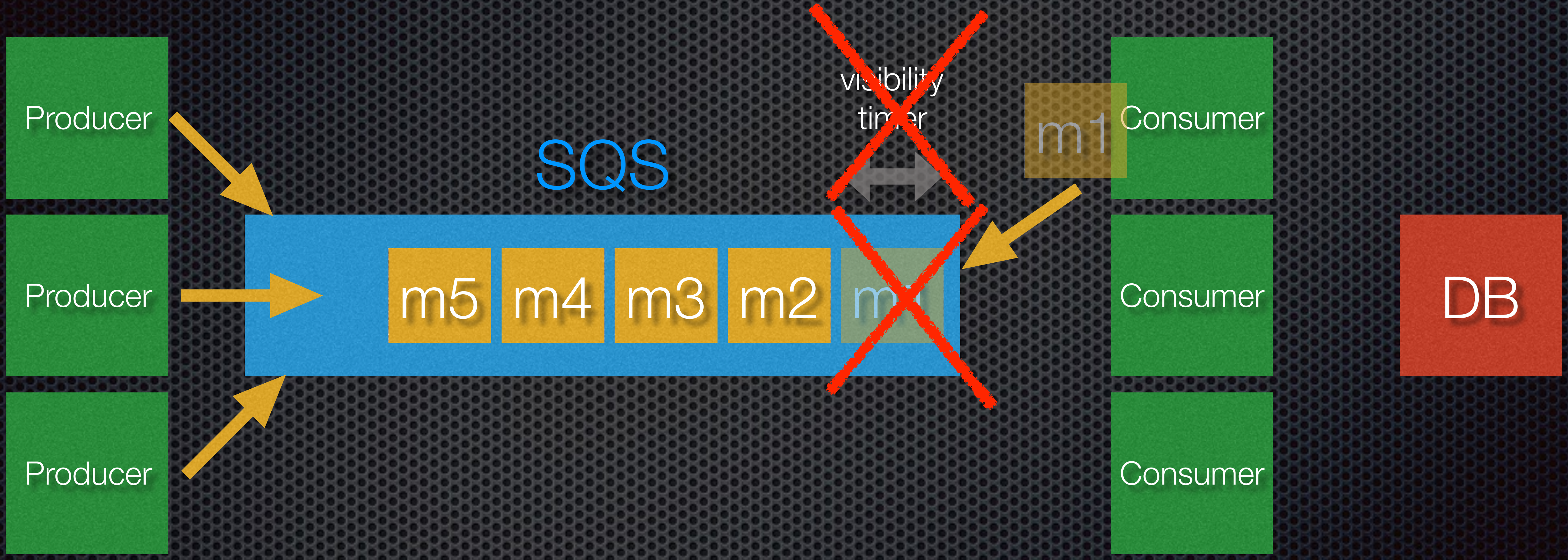
SQS - Typical Operation Flow

Step 2: Consumer persists message contents to DB



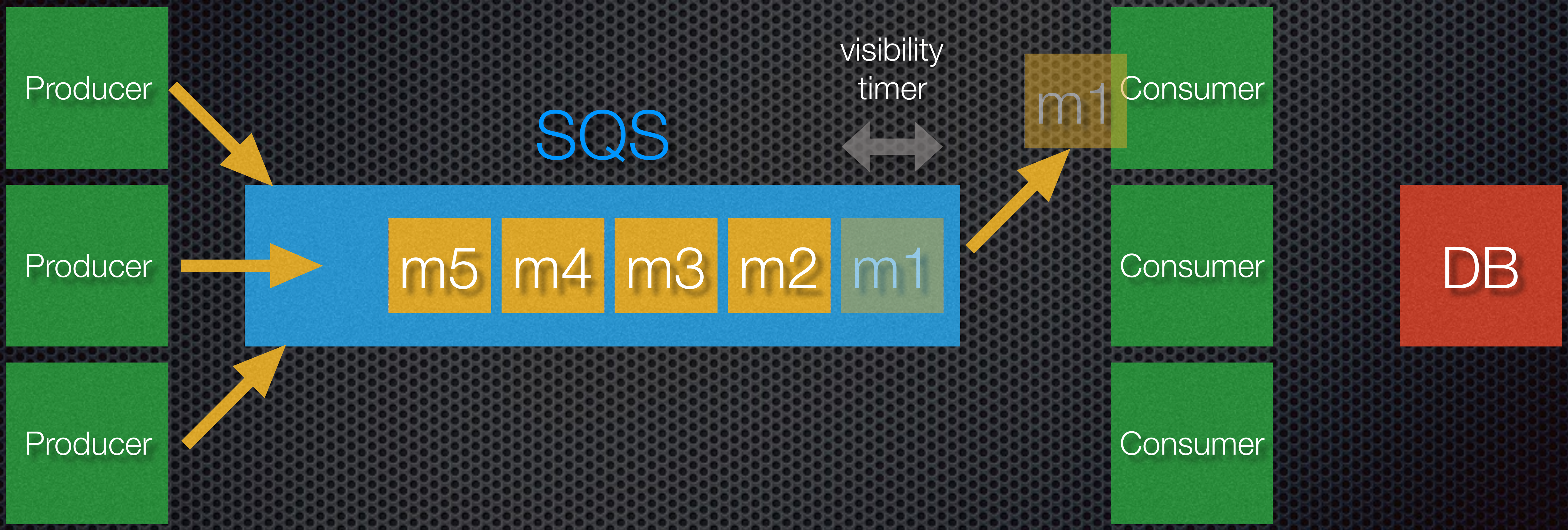
SQS - Typical Operation Flow

Step 3: Consumer ACKs message in SQS



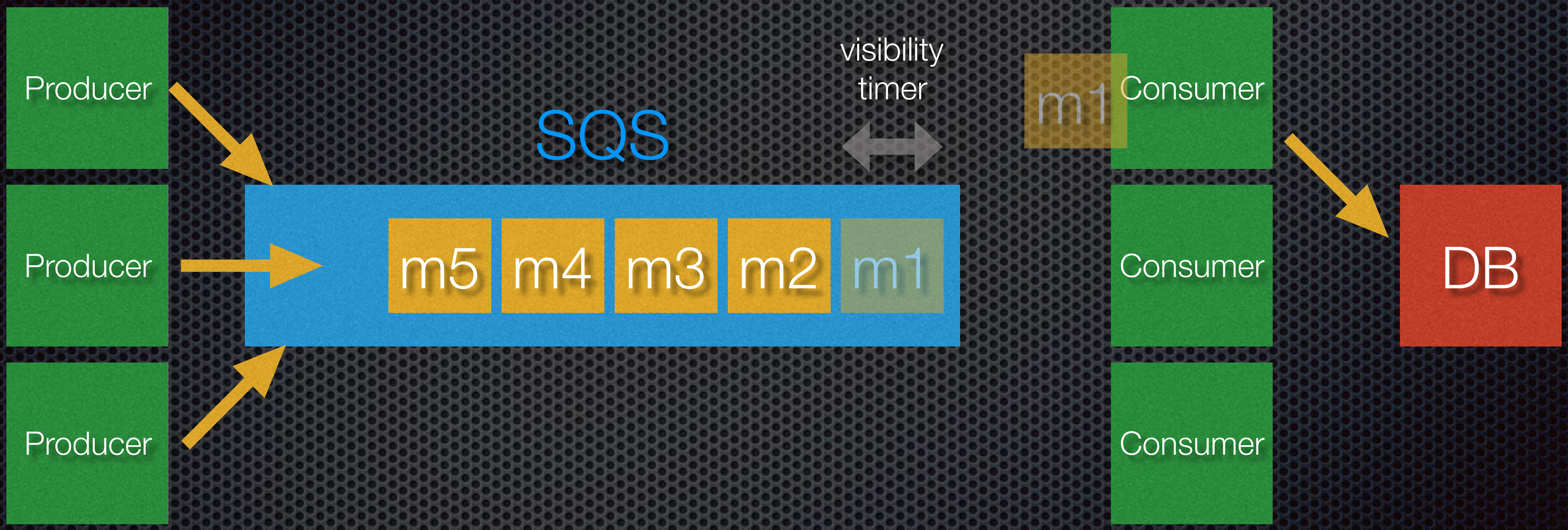
SQS - Time Out Example

Step 1: A consumer reads a message from SQS



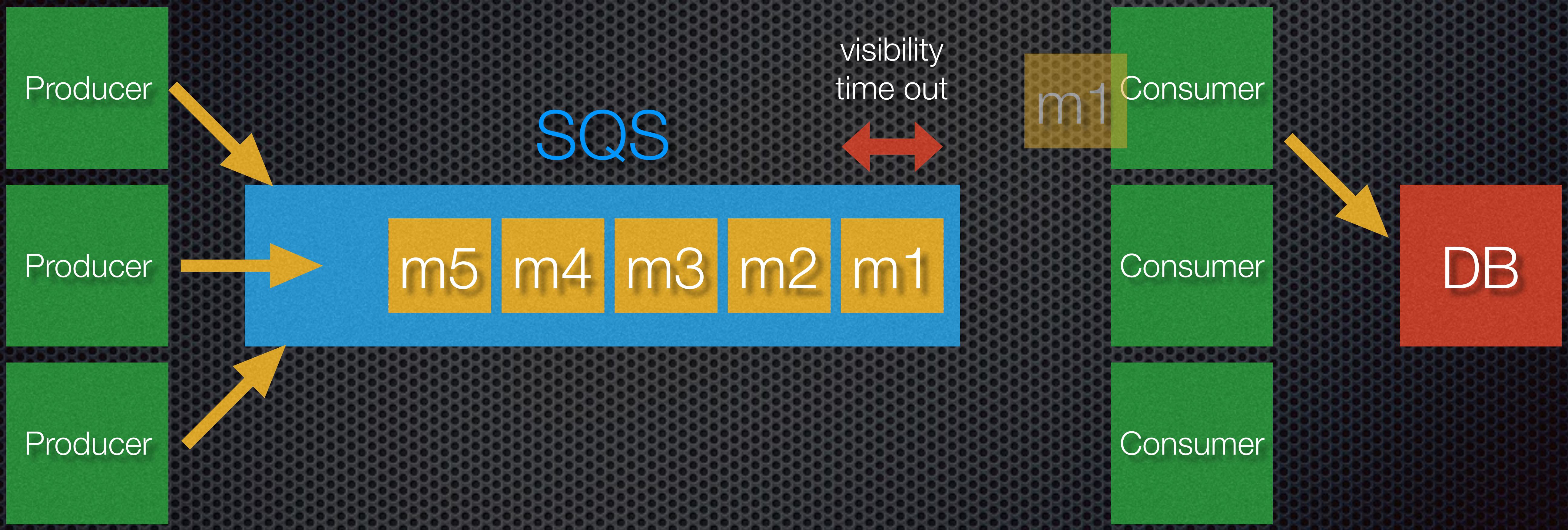
SQS - Time Out Example

Step 2: Consumer attempts persists message contents to DB



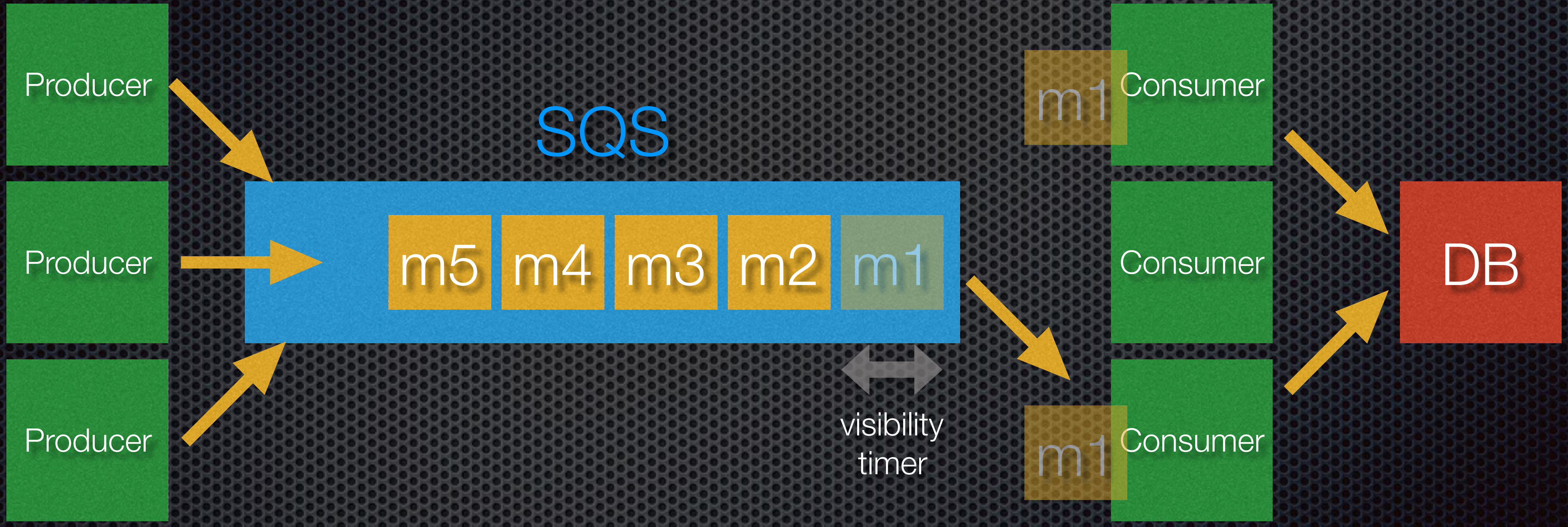
SQS - Time Out Example

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



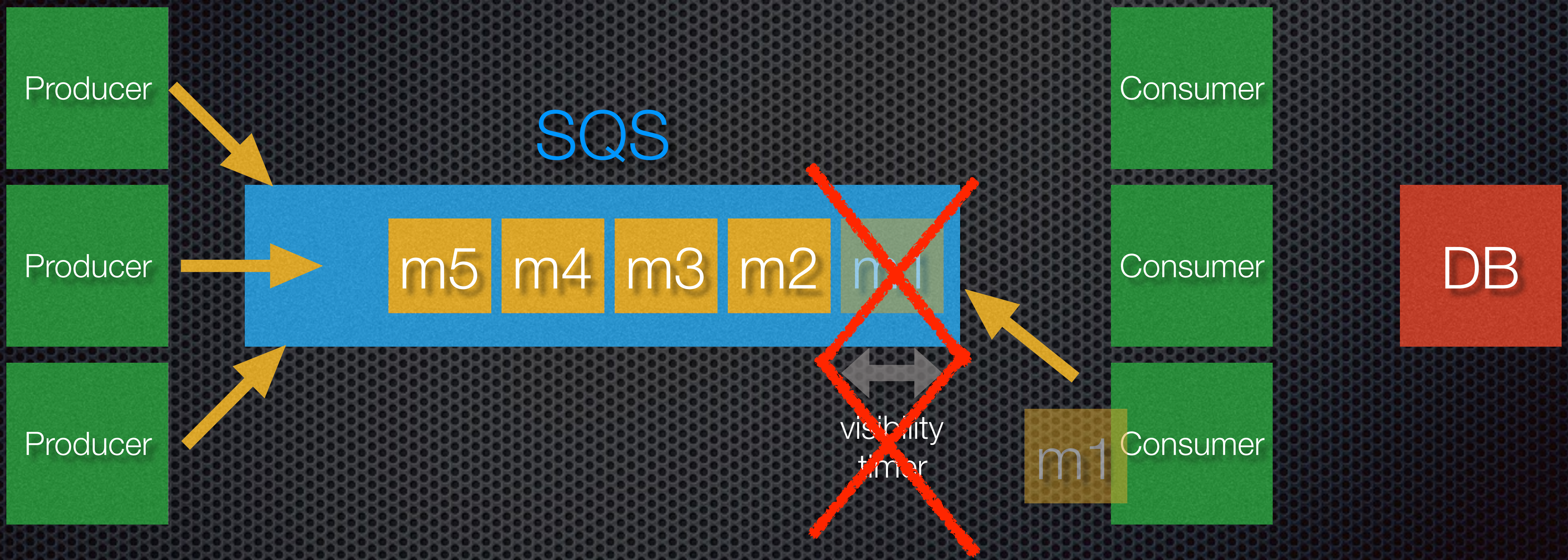
SQS - Time Out Example

Step 4: Another consumer reads and persists the same message

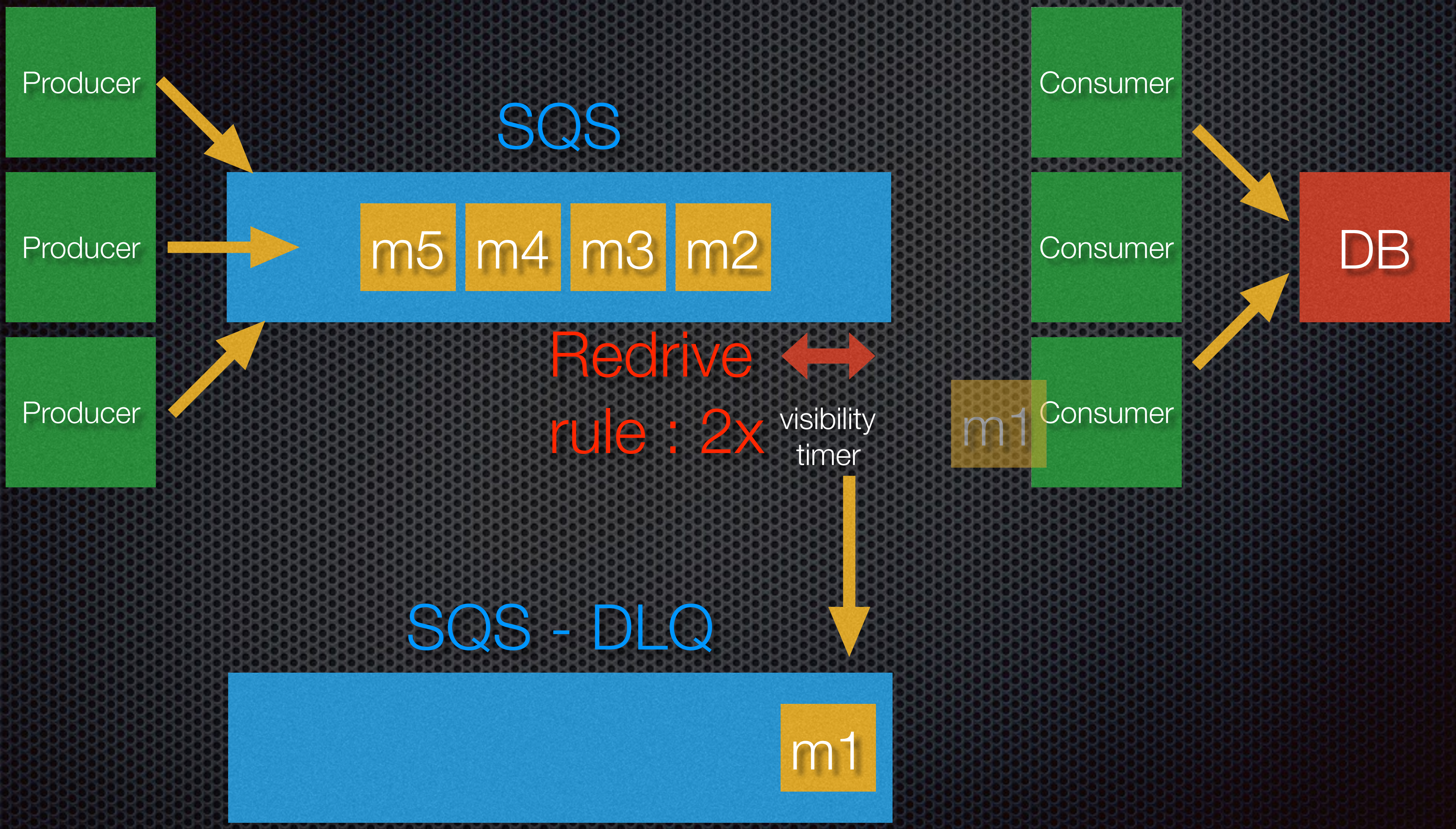


SQS - Time Out Example

Step 5: Consumer ACKs message in SQS



SQS - Dead Letter Queue



SNS

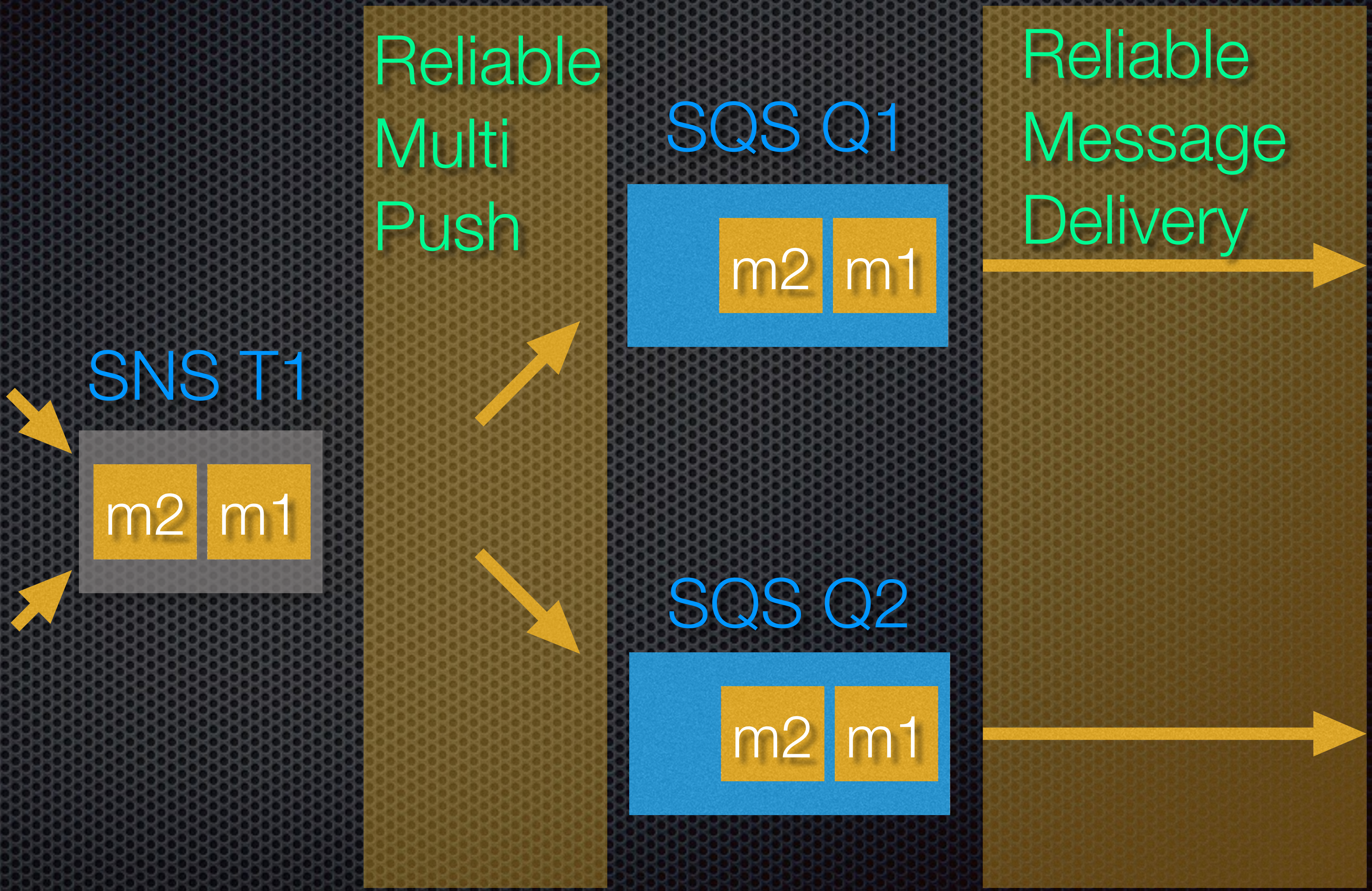
Simple Notification Service

SNS - Overview

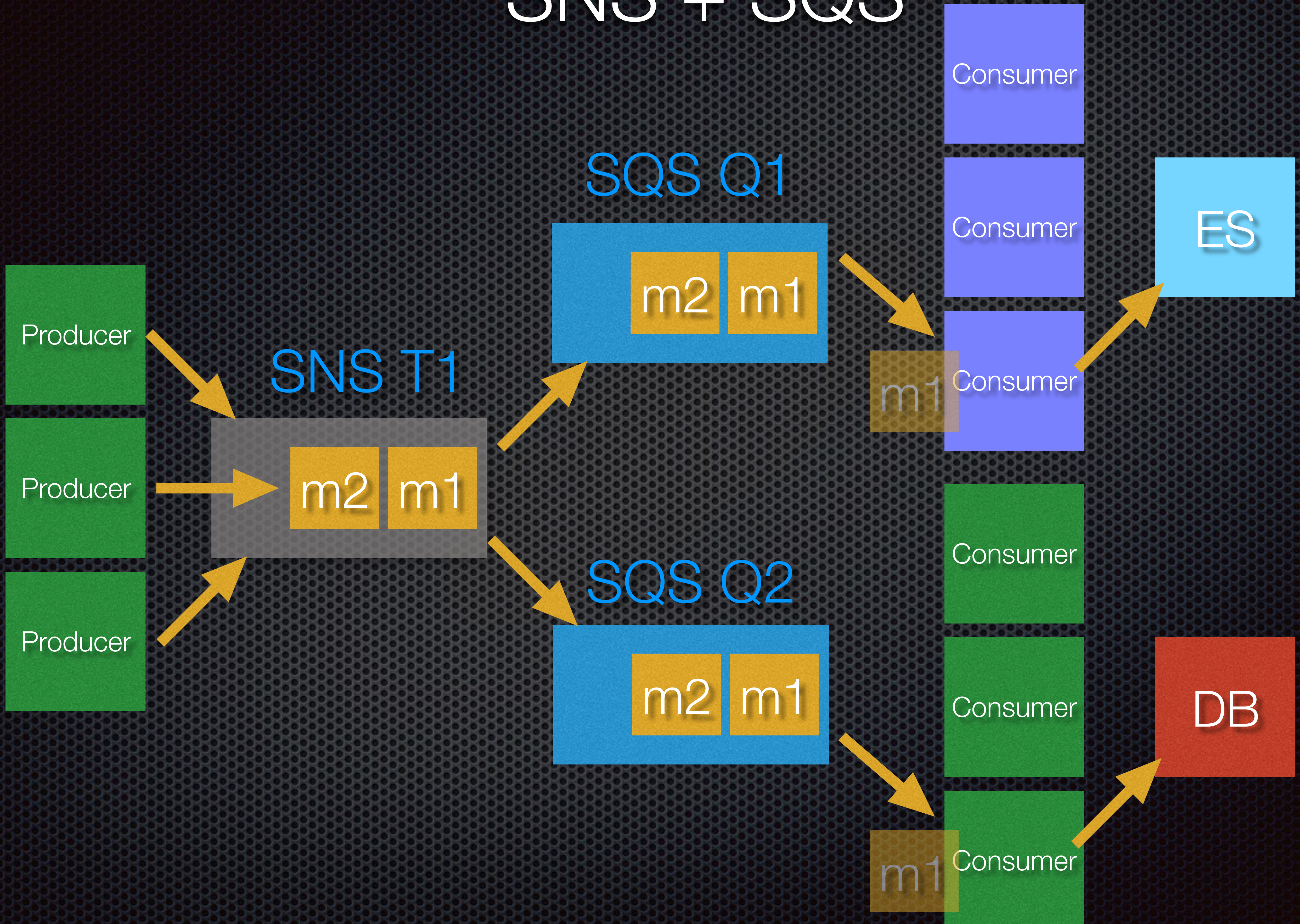
- Highly Scalable, Highly Available, Push-based Topic Service
  - Whereas SQS ensures each message is seen by at least 1 consumer
    - SNS ensures that each message is seen by every consumer
    - Reliable Multi-Push
  - Whereas SQS is pull-based, SNS is push-based
    - There is no message retention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

Can we work around this limitation?

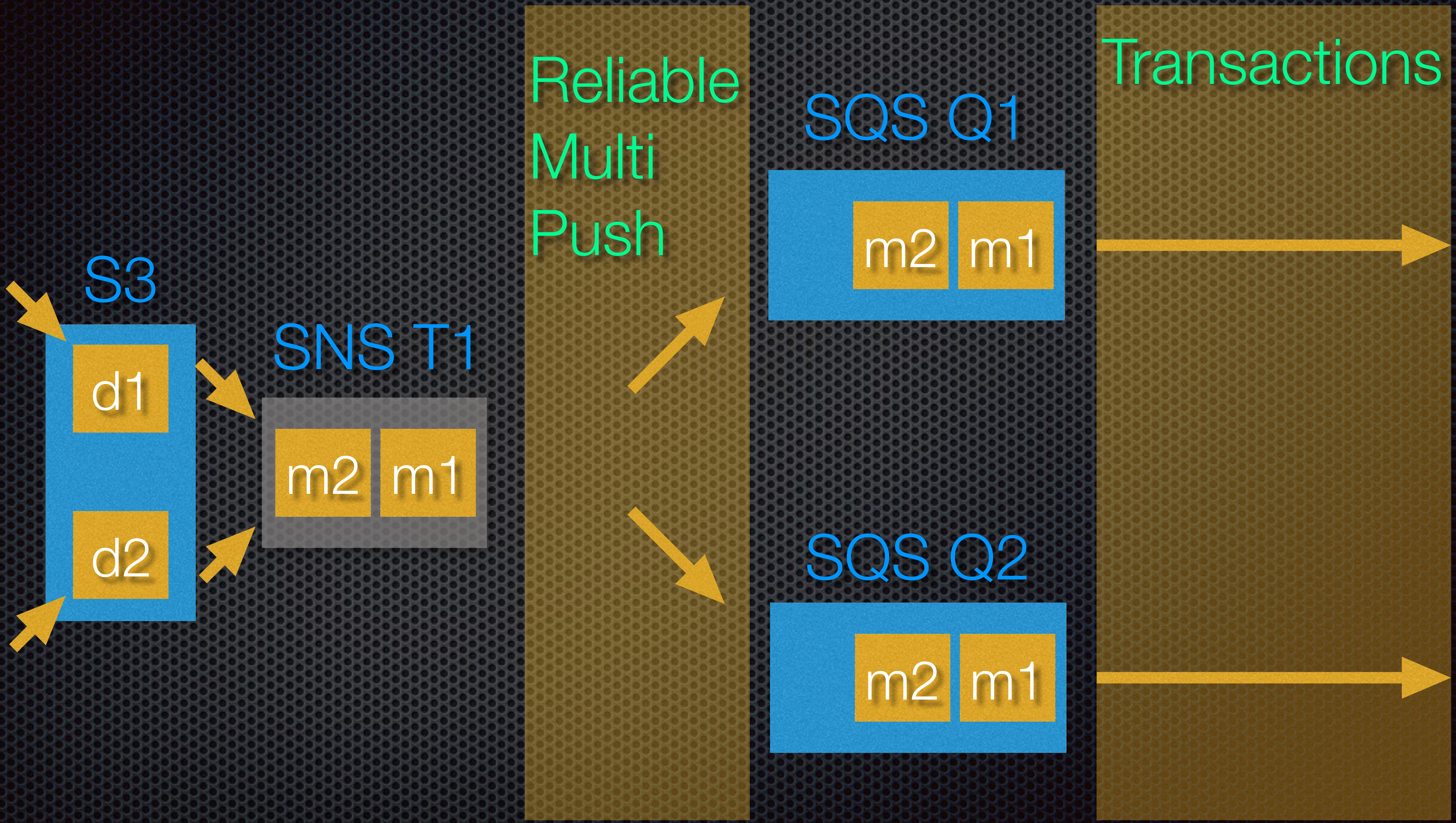
SNS + SQS Design Pattern



SNS + SQS


Producer
Producer
Producer
SNS T1
m2
m1
SQS Q1
m2
m1
SQS Q2
m2
m1
Consumer
Consumer
m1
Consumer
Consumer
Consumer
m1
Consumer
ES
DB

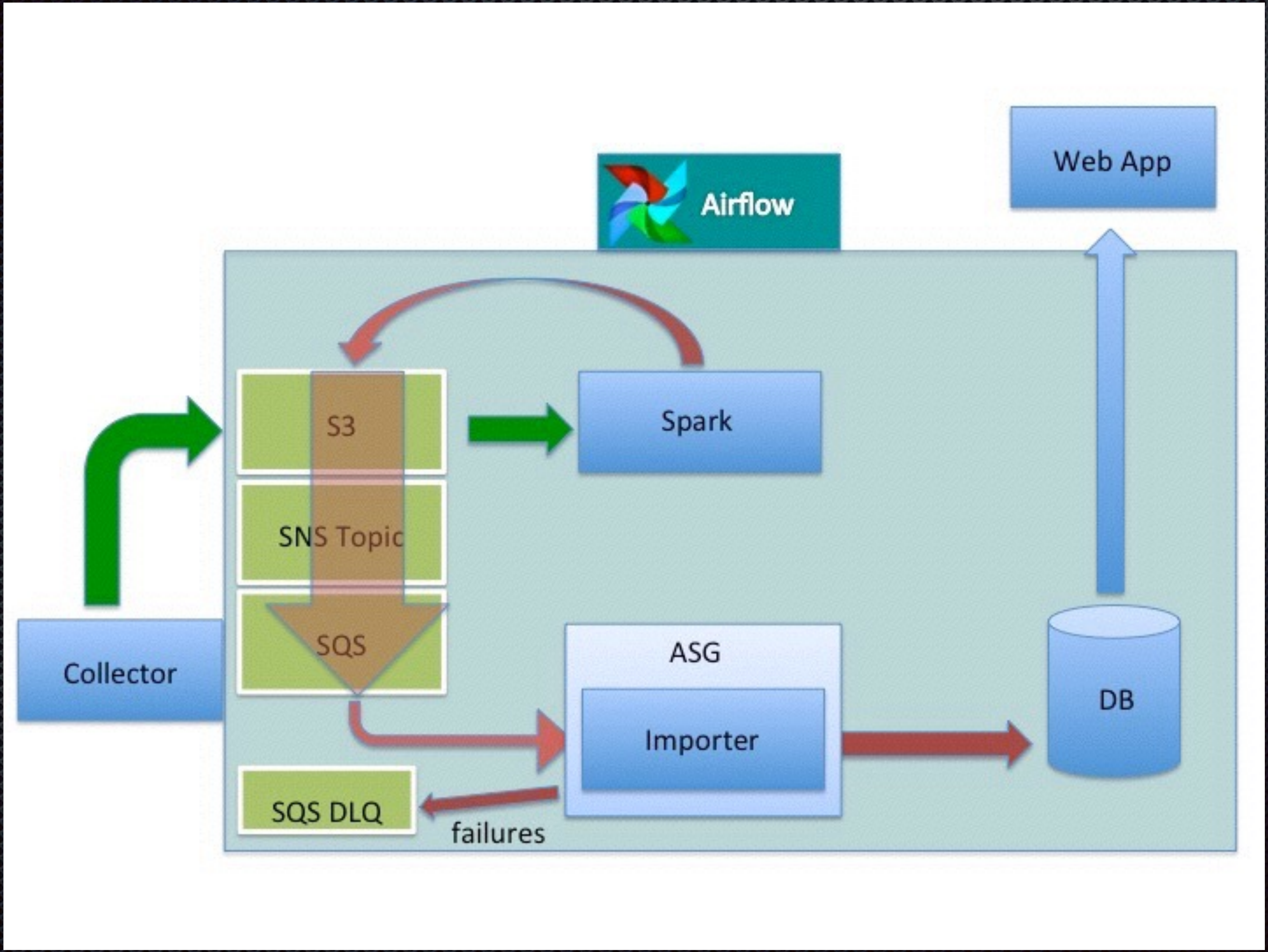
S3 + SNS + SQS Design Pattern



Batch Pipeline Architecture

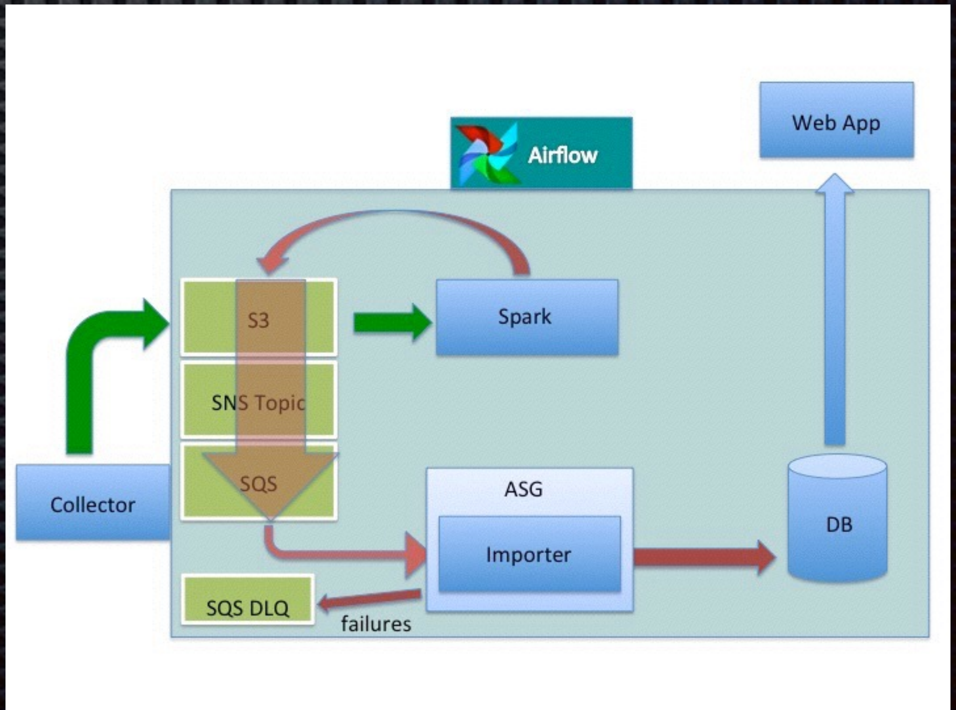
Putting the Pieces Together

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

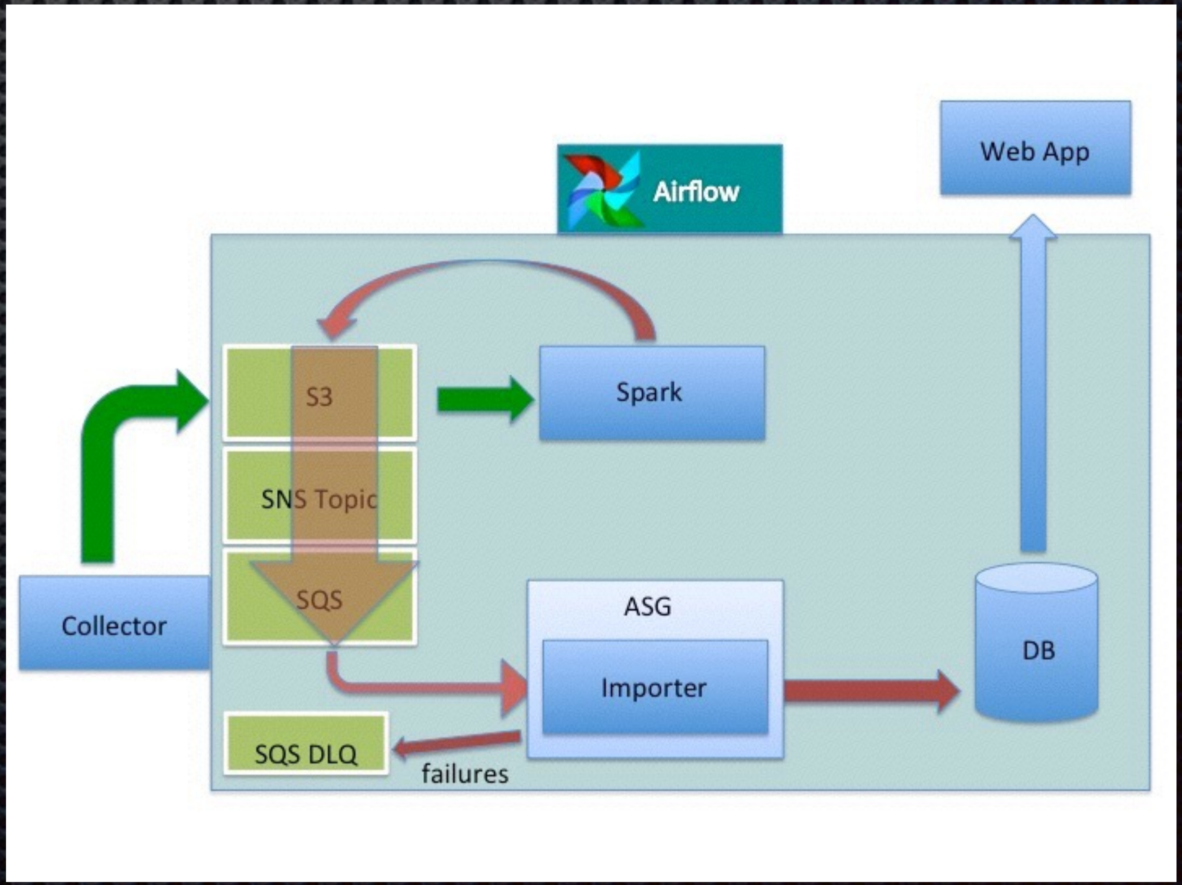


ASG

Auto Scaling Group

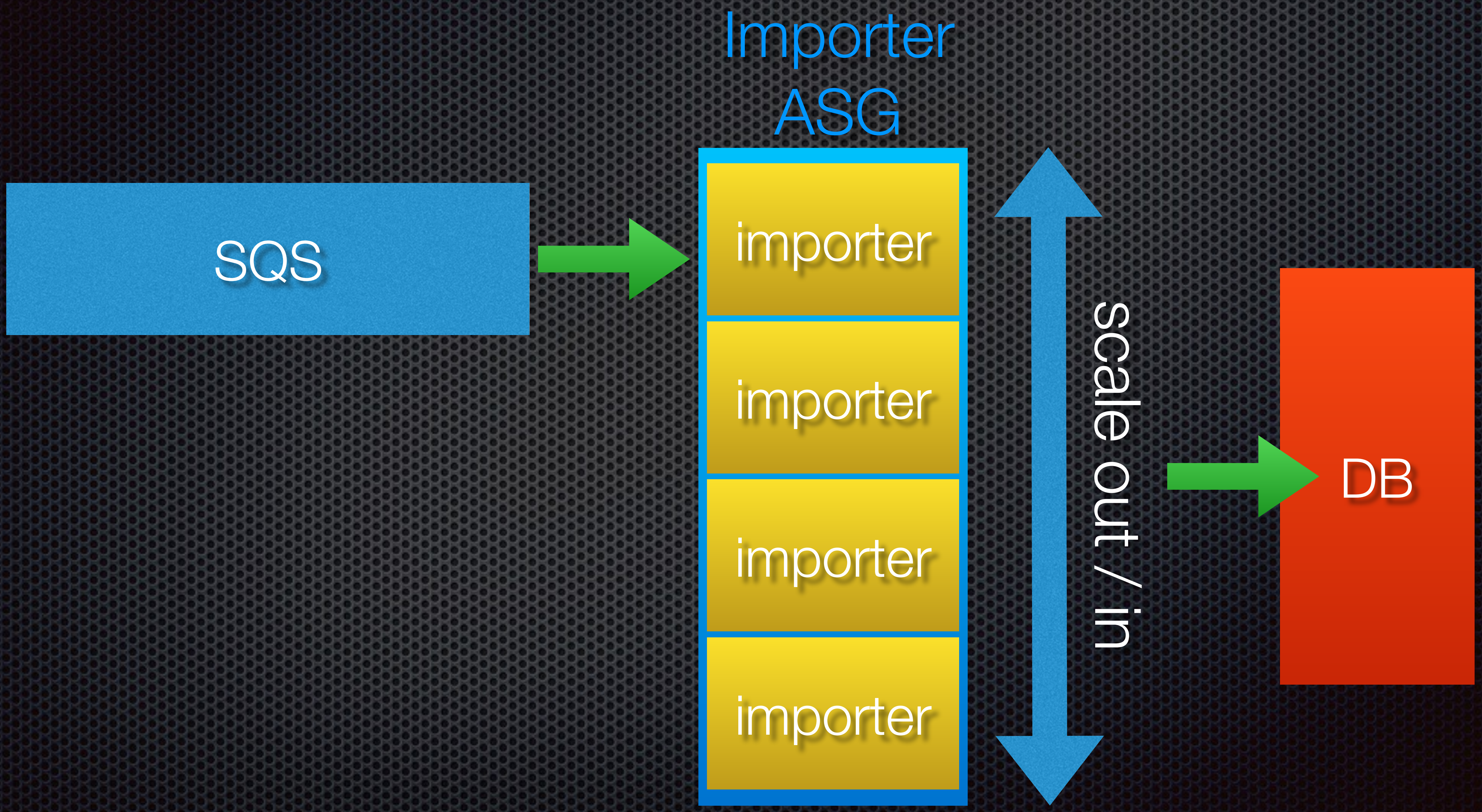
ASG - Overview

- 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!

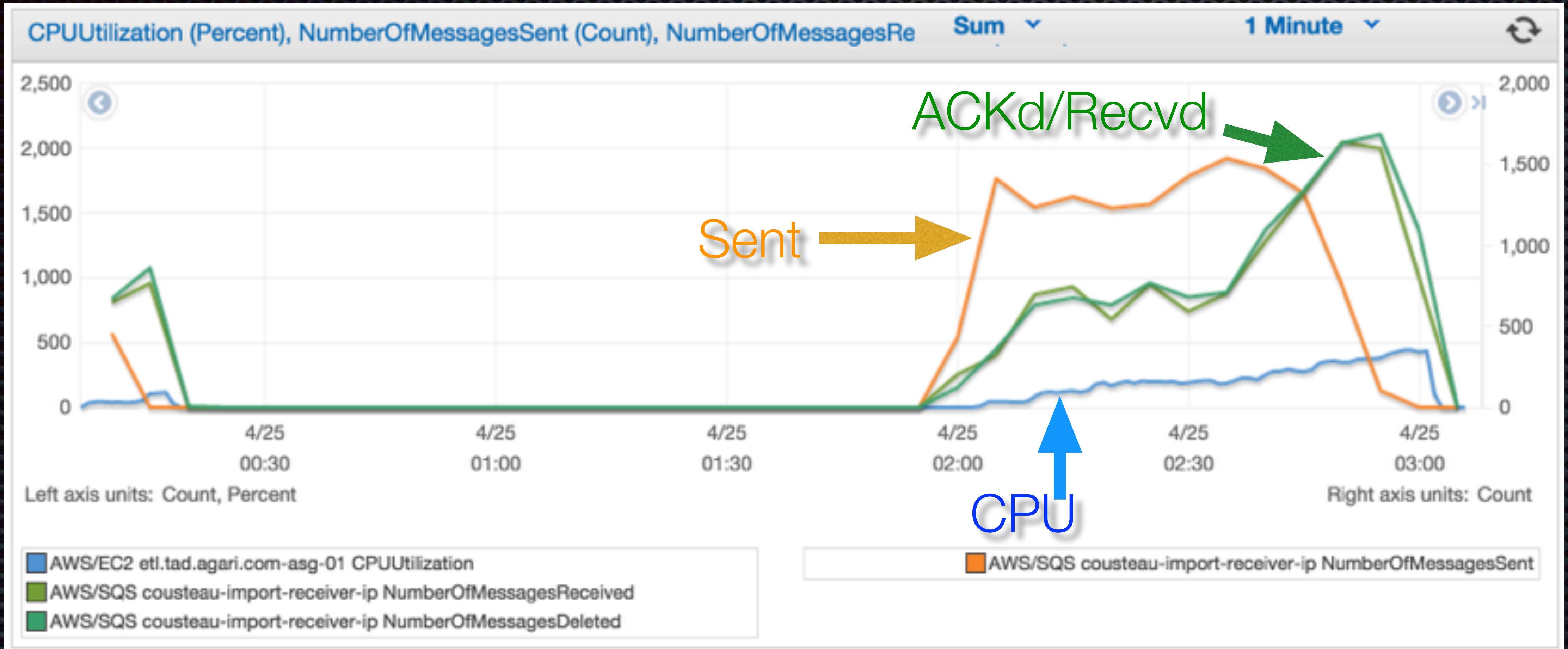


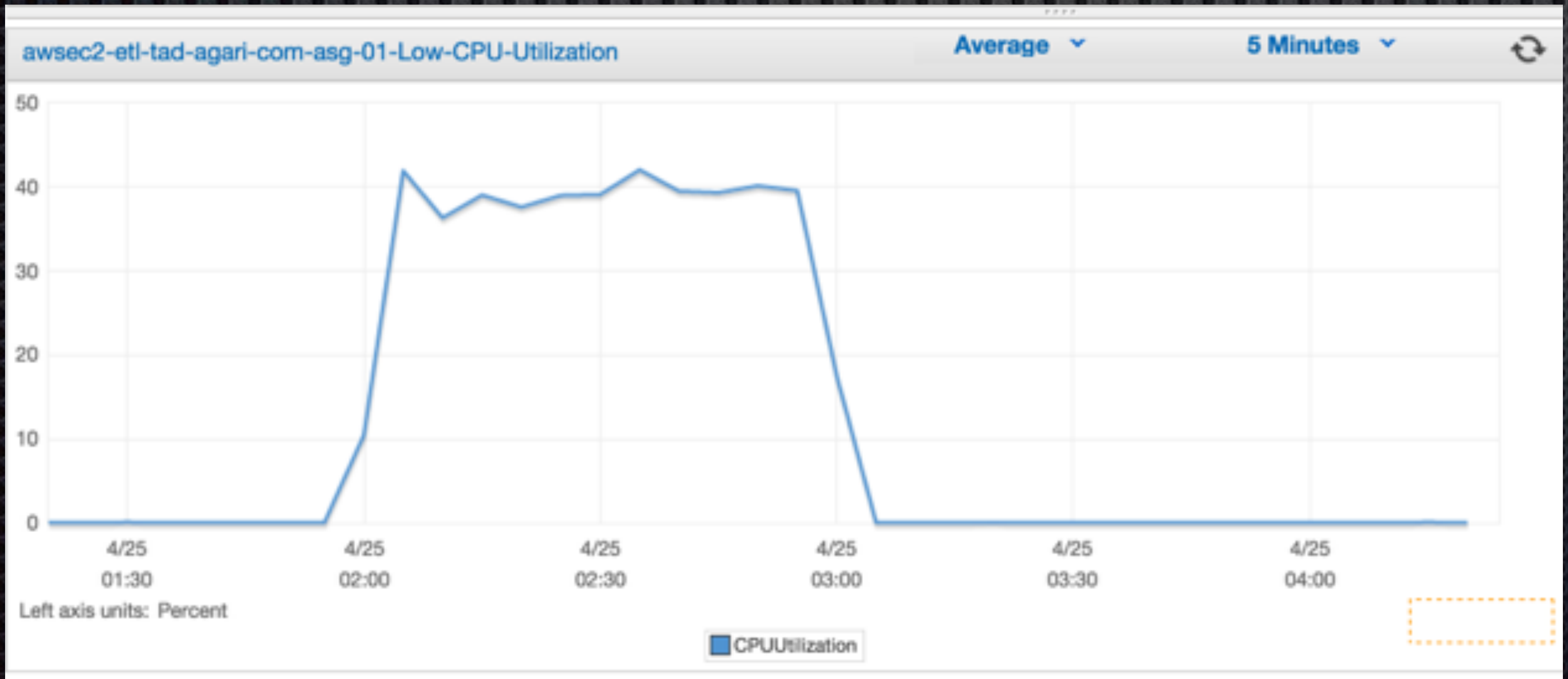
- When to use it?
  - Feed-processing, web traffic load balancing, zone outage, etc…

ASG - Data Pipeline



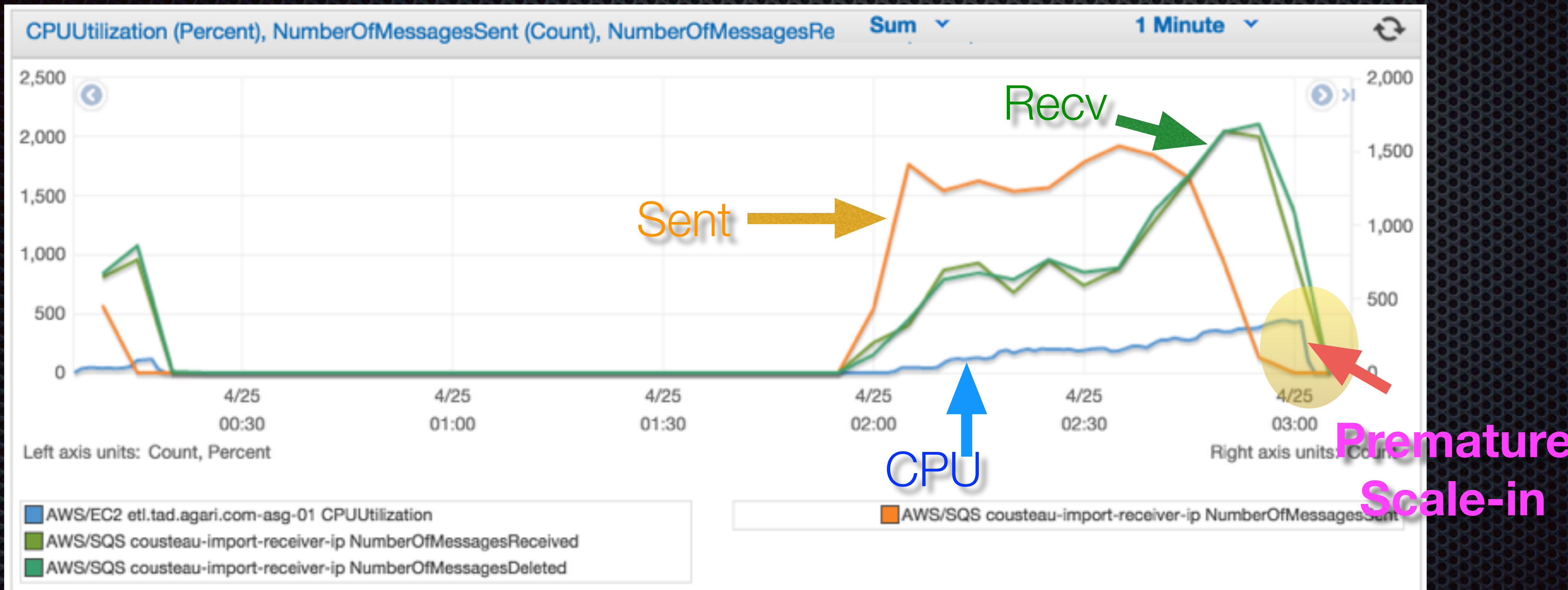
ASG : CPU-based





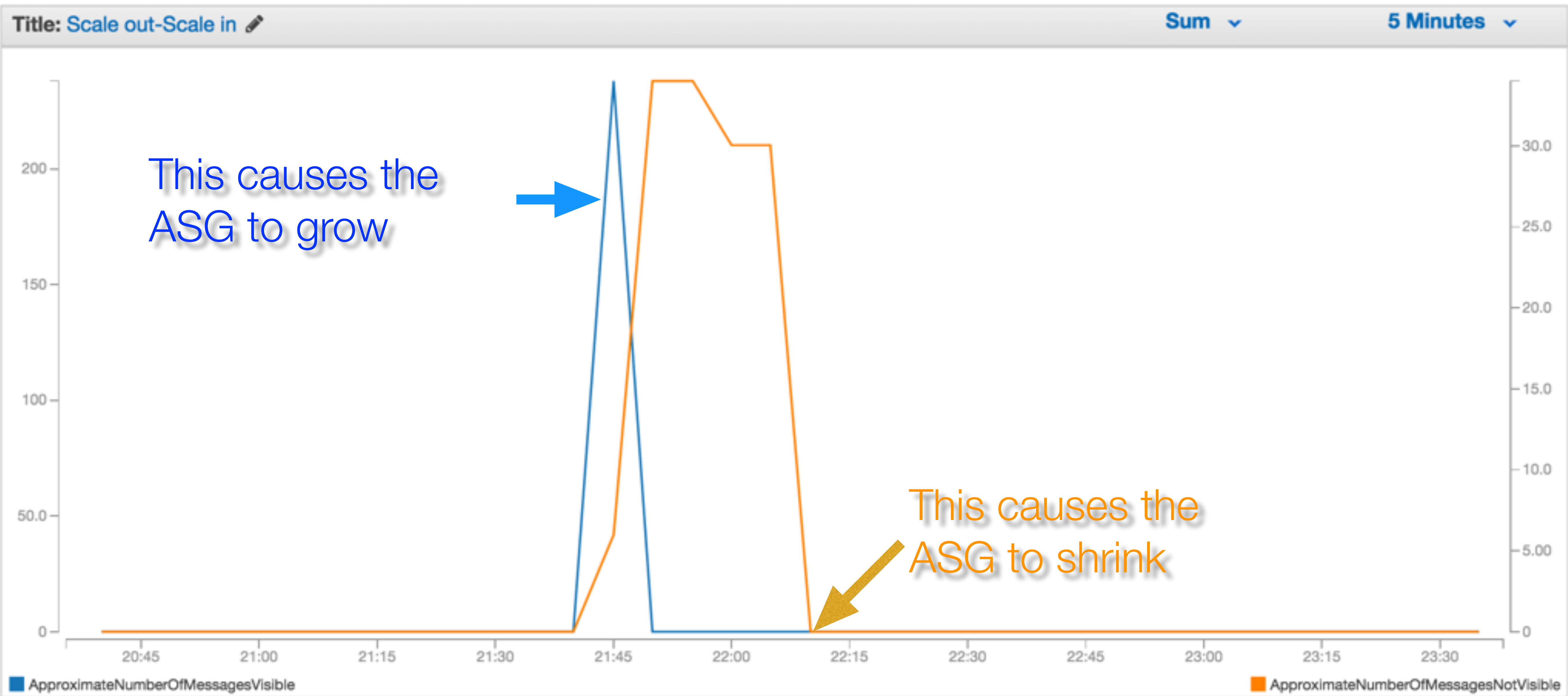
CPU-based auto-scaling is good at scaling in/out to keep the average CPU constant

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!

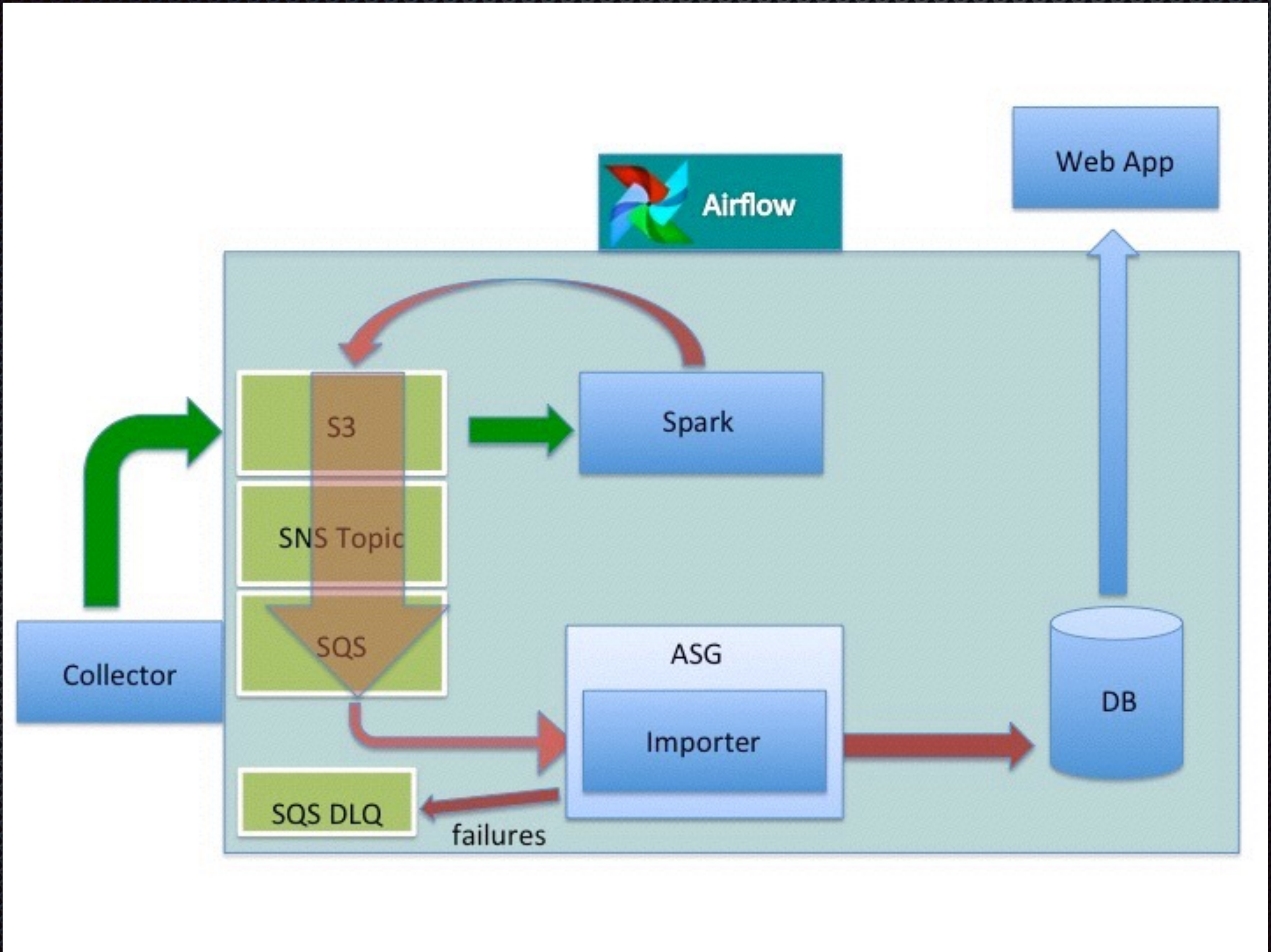
ASG - Queue-based



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

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | 00:31 UTC

DAGs

Show 10 entries | Search:

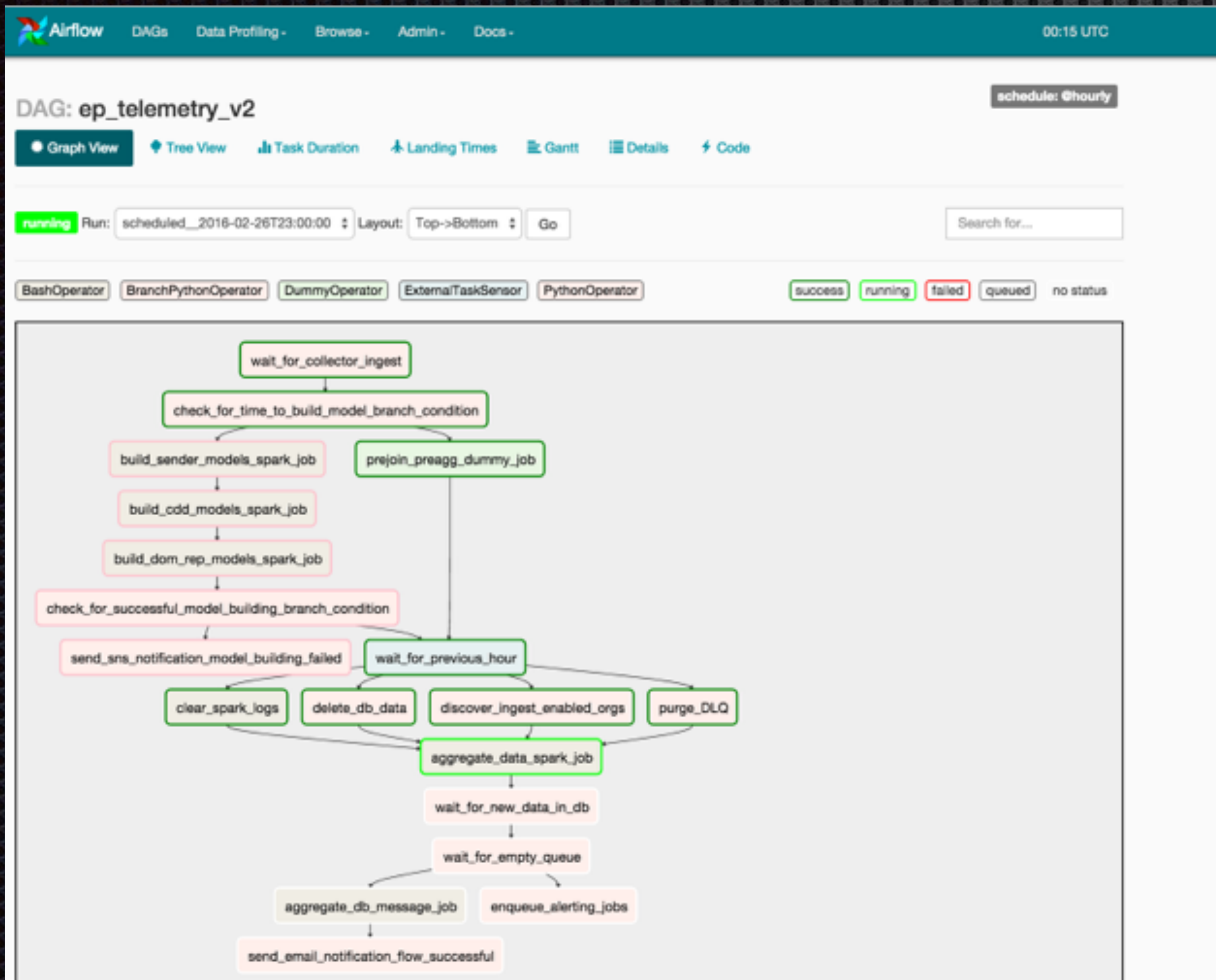
		DAG	Schedule	Owner	Statuses	Links
	On	db_backup_v1	0 4 * * *	aflury	25	
	On	emr_forwarders	0 8 * * *	kmandich	118	
	On	emr_model_building	0 1 * * *	kmandich	235	
	Off	ep_model_building_v1	0 1 * * *	sanand	253	
	Off	ep_reload_data	None	sanand	2	
	On	ep_telemetry_v2	@hourly	sanand	30671 7	
	Off	feeback_report	1 day, 0:00:00	kmandich	72	
	On	generate_spoofs	1 day, 0:00:00	kmandich	258 1 1	
	On	refresh_asn_data	0 10 * * *	kmandich	292	
	On	refresh_dns_cache	0 10 1,15 * *	kmandich		

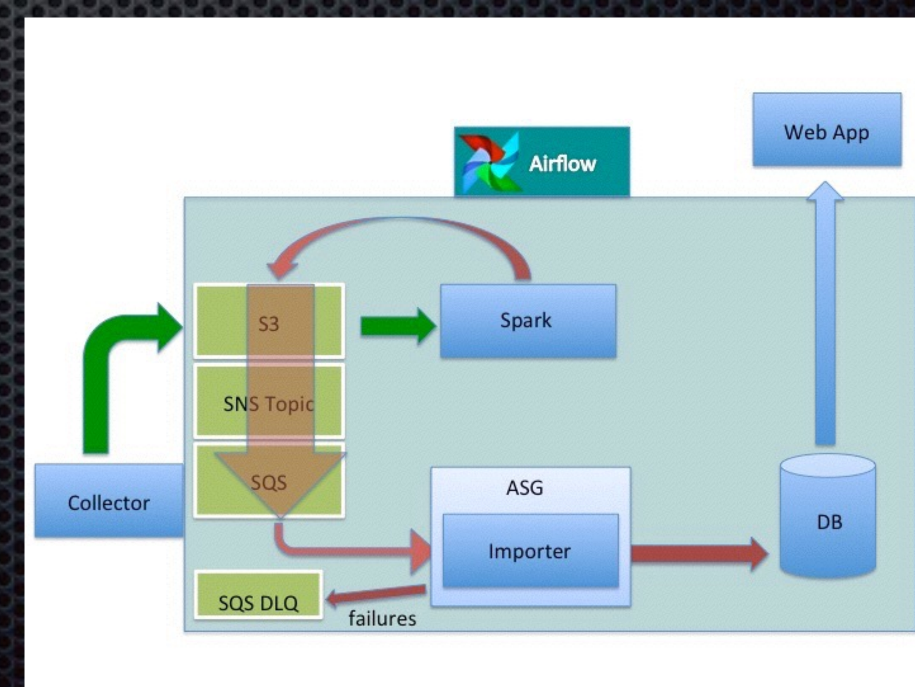
Showing 1 to 10 of 10 entries

Previous 1 Next

Airflow - Authoring DAGs

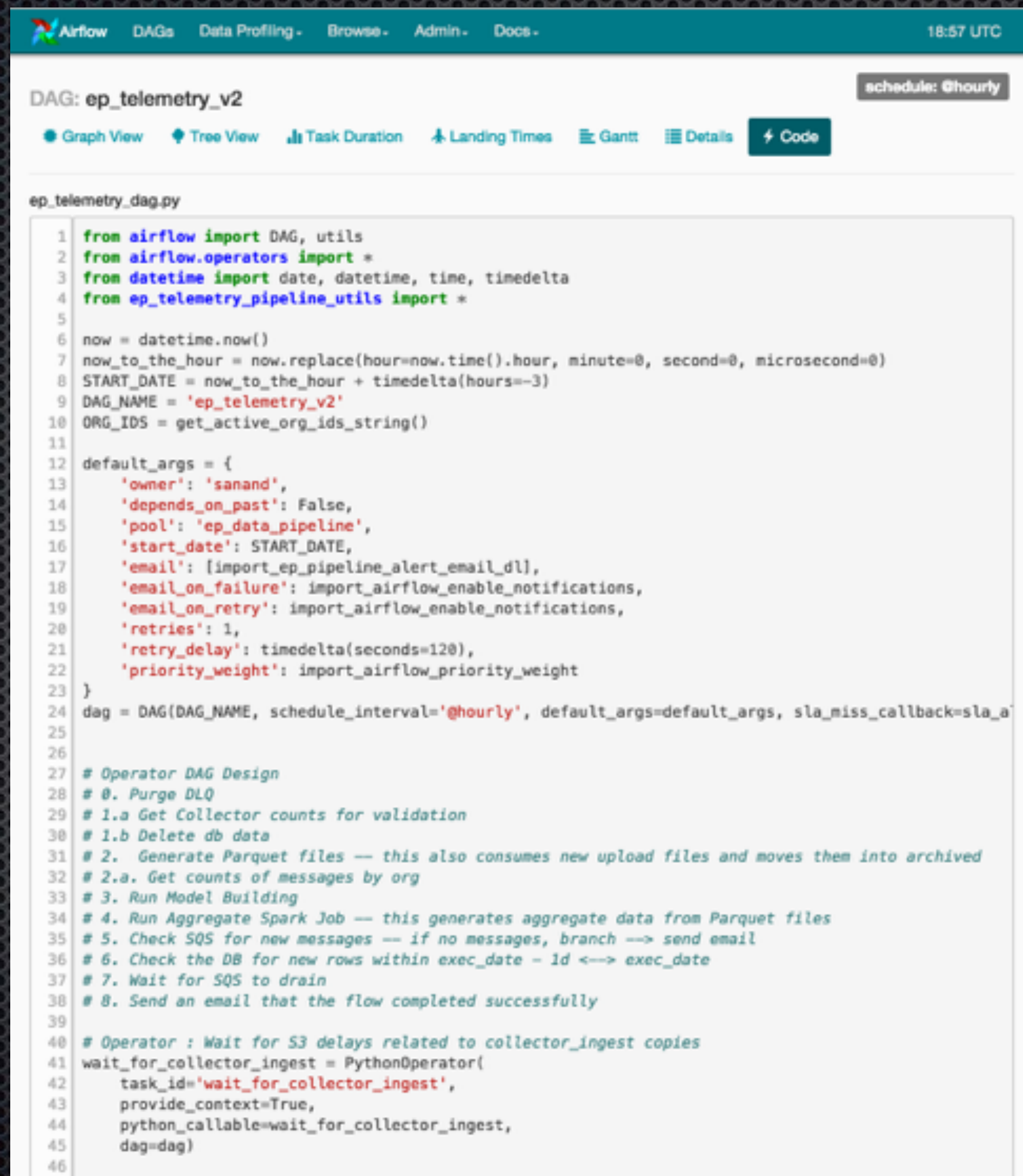
Airflow: Visualizing a DAG





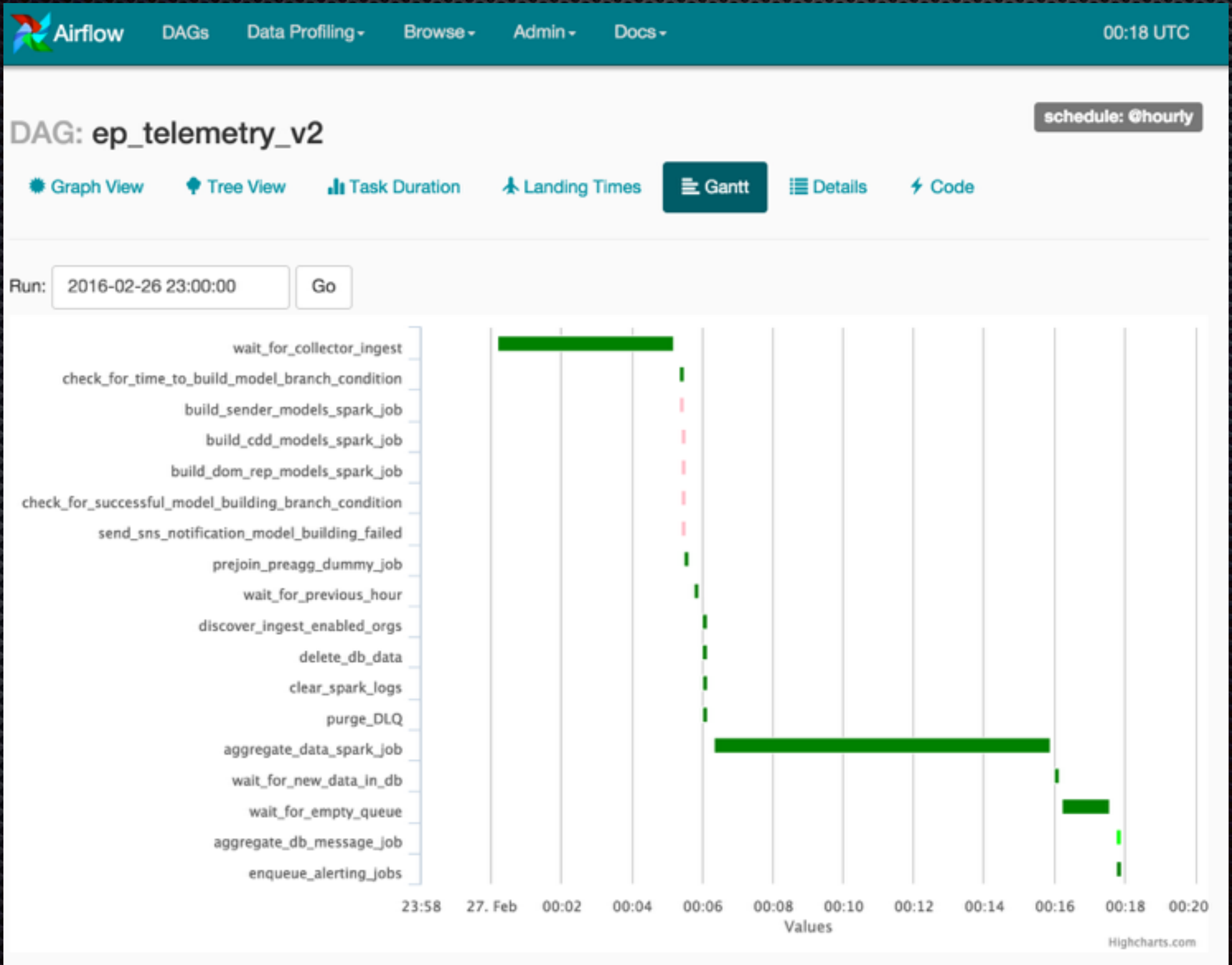
Airflow - Authoring DAGs

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



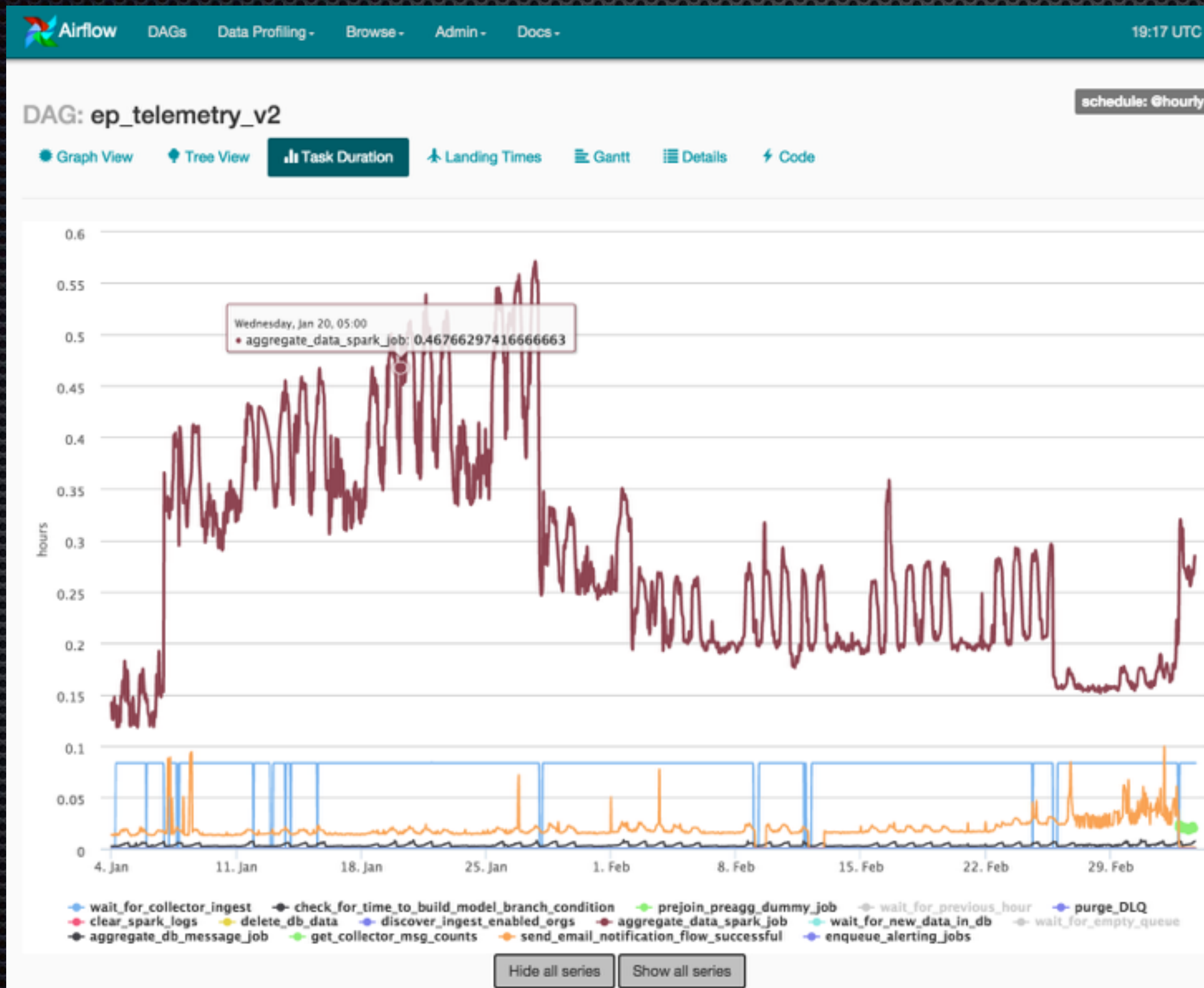
Airflow - Performance Insights

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

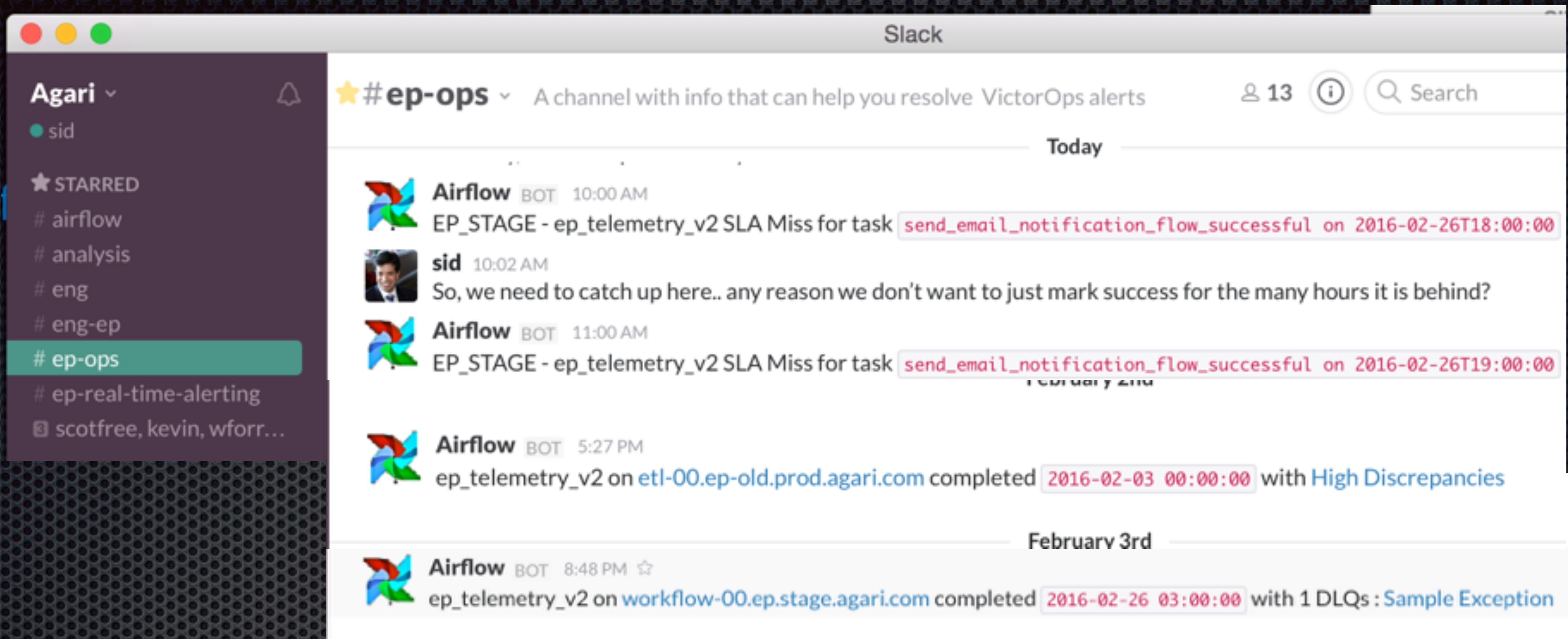


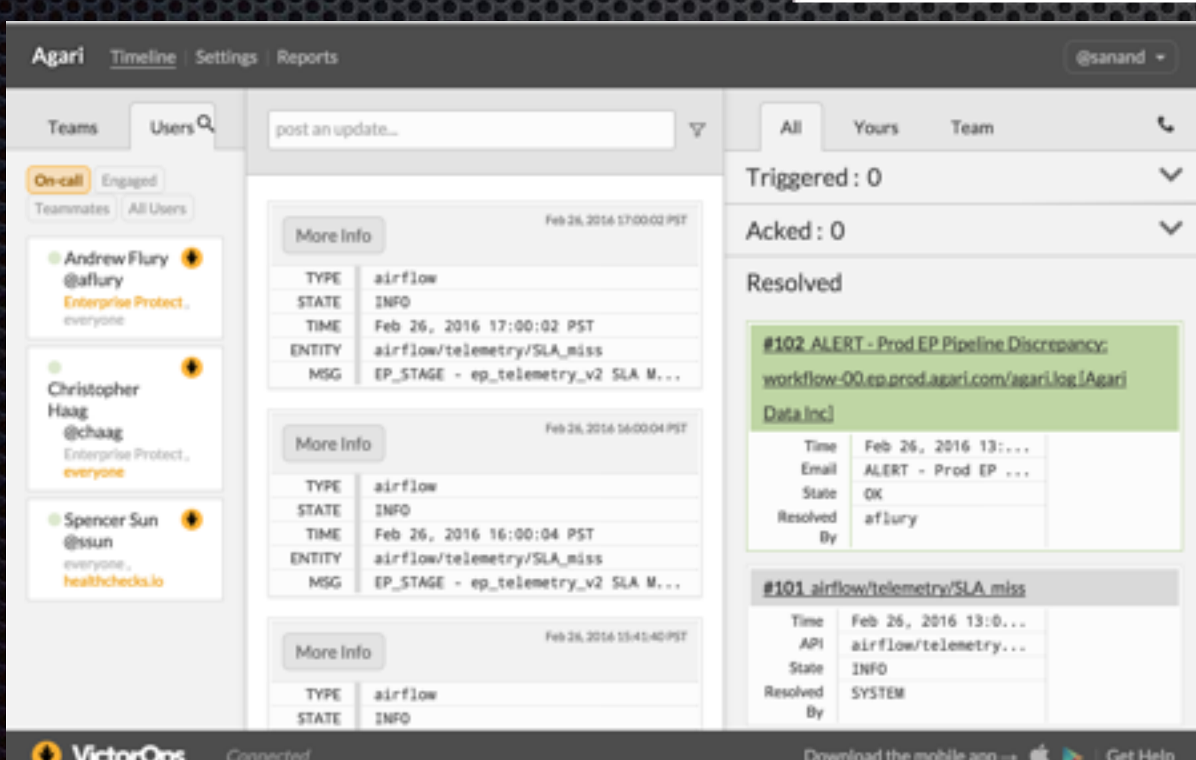
Airflow - Performance Insights

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

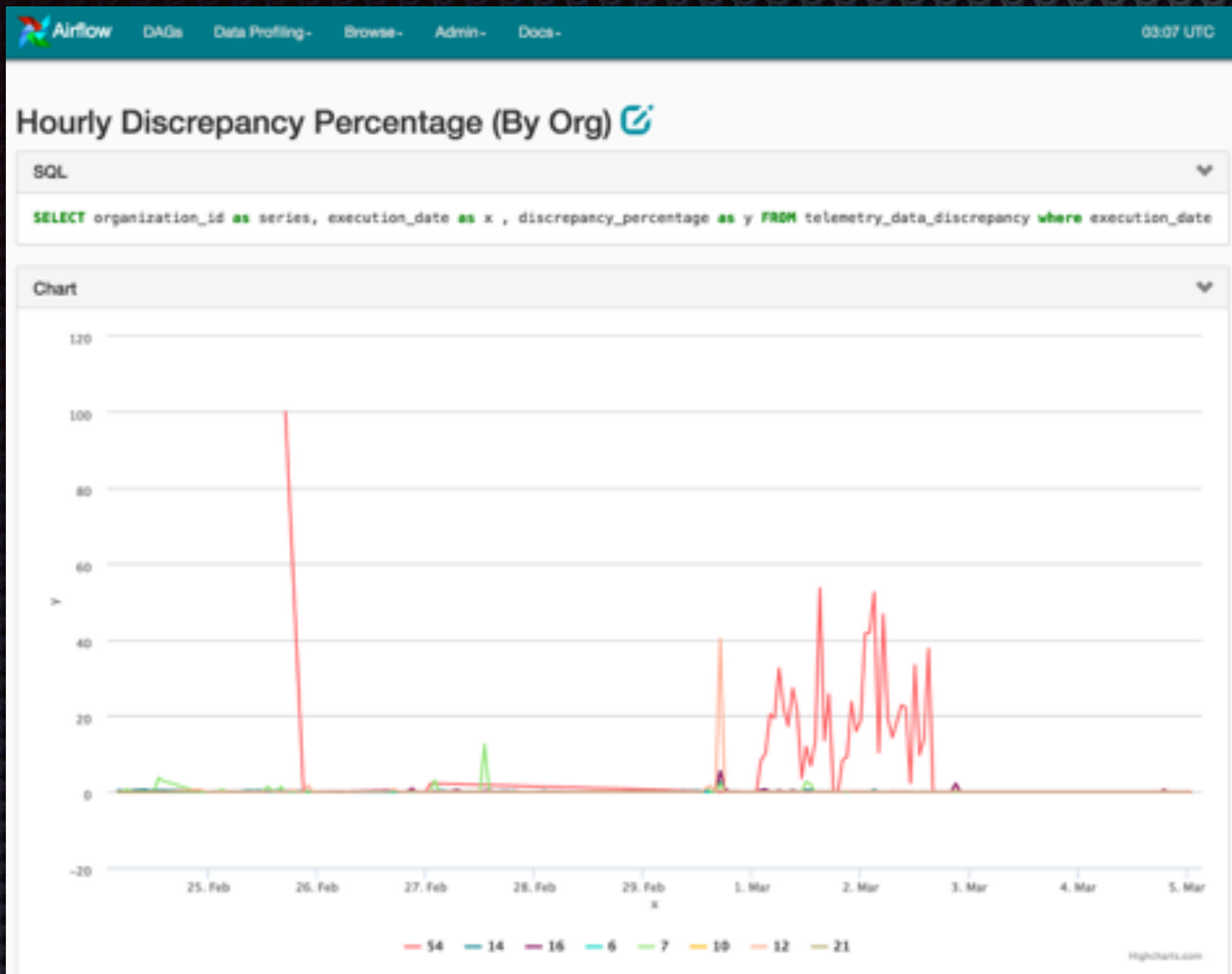


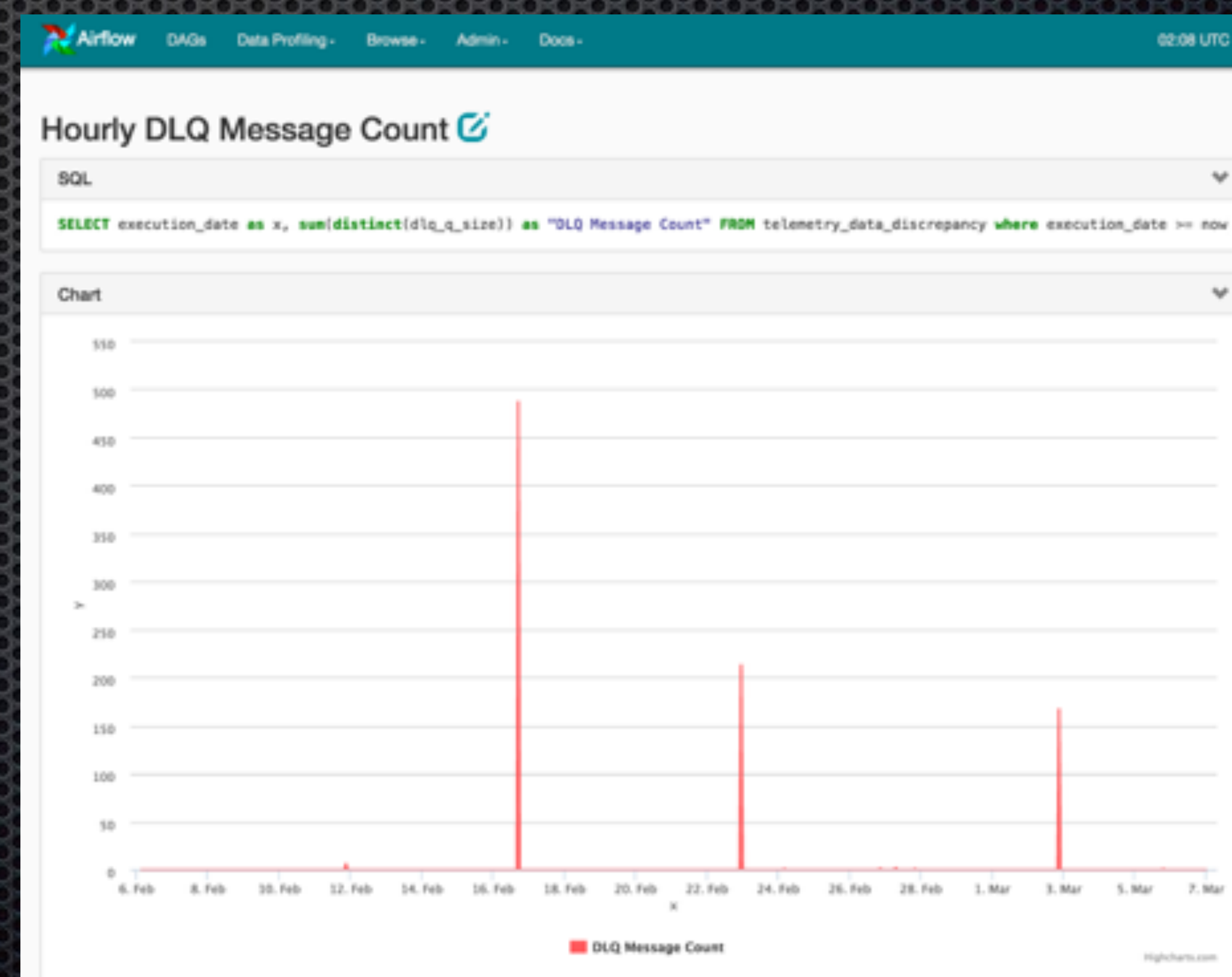
Airflow - Alerting



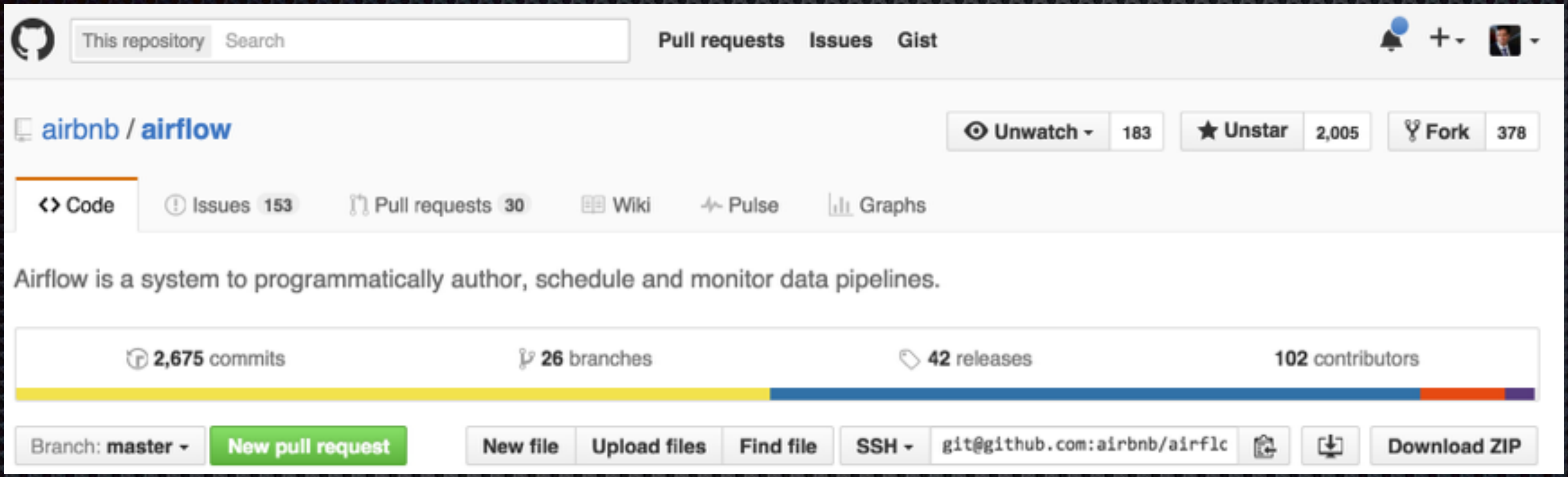


Airflow - Monitoring





Airflow - Join the Community



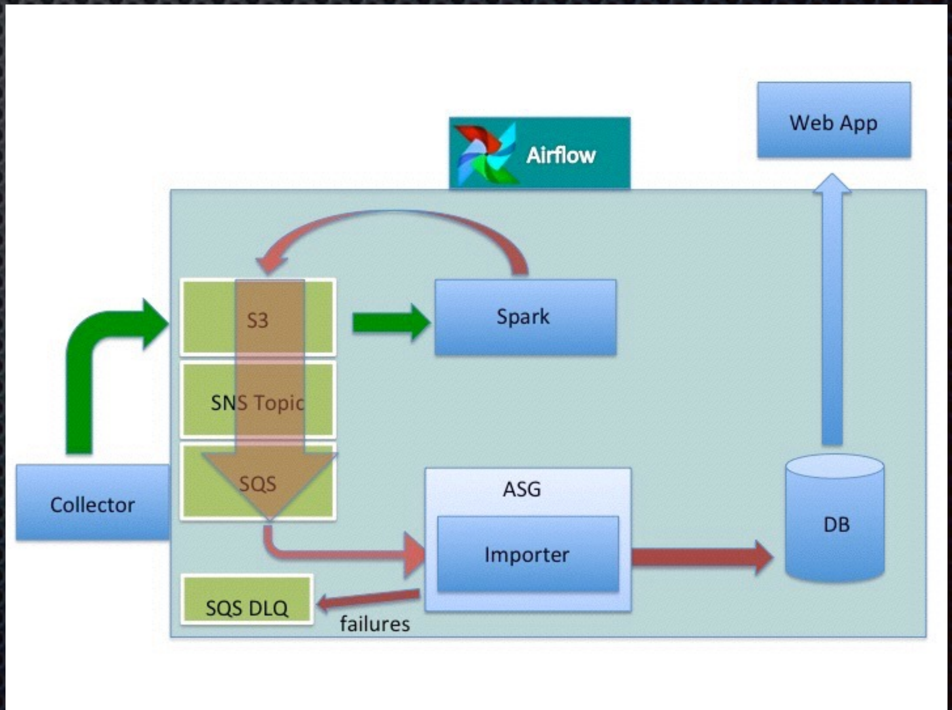
- With >30 Companies, >100 Contributors , and >2500 Commits, Airflow is growing rapidly!
- We are looking for more contributors to help support the community!
- Disclaimer : I'm a maintainer on the project

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)