DESIGNING DATA-INTENSIVE APPLICATIONS IN SERVERLESS ARCHITECTURE

@matvi3nko

Nikolay Matvienko

Senior Software Engineer and Node.js expert at Grid Dynamics

Diagnostics, performance improvement consultant.

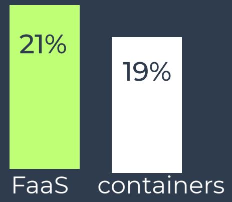
Work in USA and in Russia

Designed serverless computing platforms with AWS

You can find me at twitter.com/**matvi3nko** github.com/**matvi3nko**

DESIGNING SERVERLESS

1. SERVERLESS COMPUTING HAS BECOME VERY POPULAR
2. SERVERLESS SOLUTIONS ARE ~60% AND UP TO SEVERAL TIMES CHEAPER

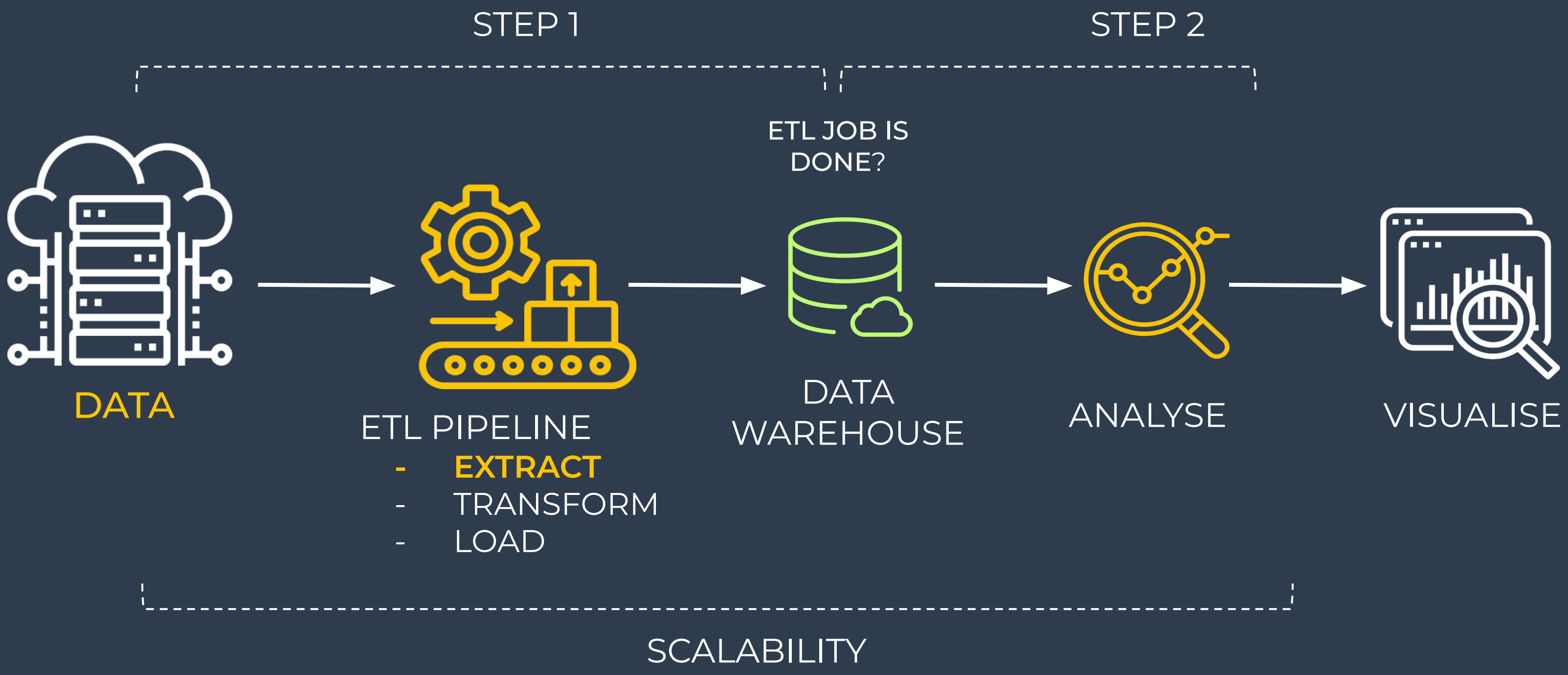


3. EVERY YEAR AWS ANNOUNCES NEW SERVICES
4. BUT THE LACK OF PATTERNS AND MISUSE CAN MAKE THE SOLUTION SEVERAL TIMES

MORE EXPENSIVE

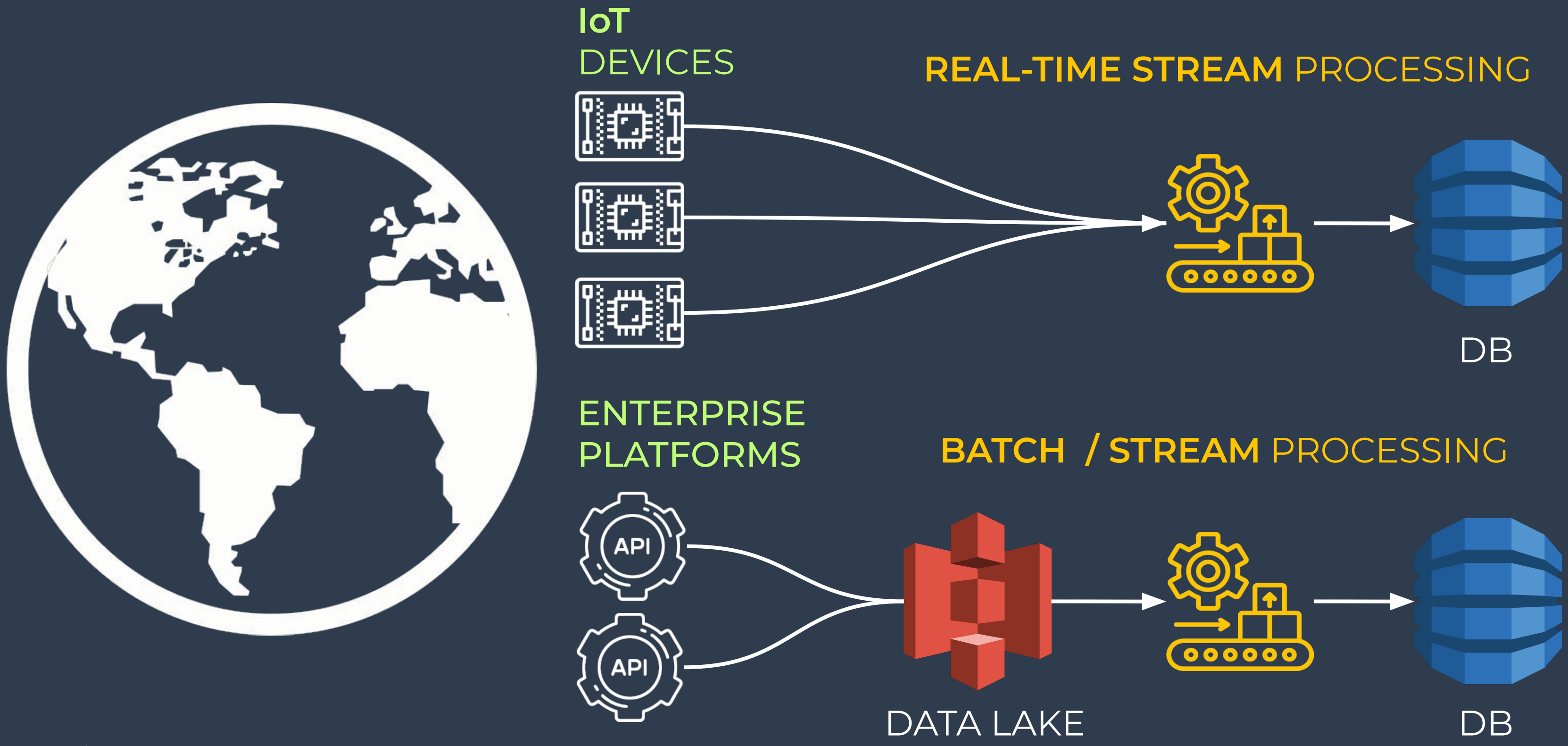
@matvi3nko

DATA-INTENSIVE APPLICATIONS



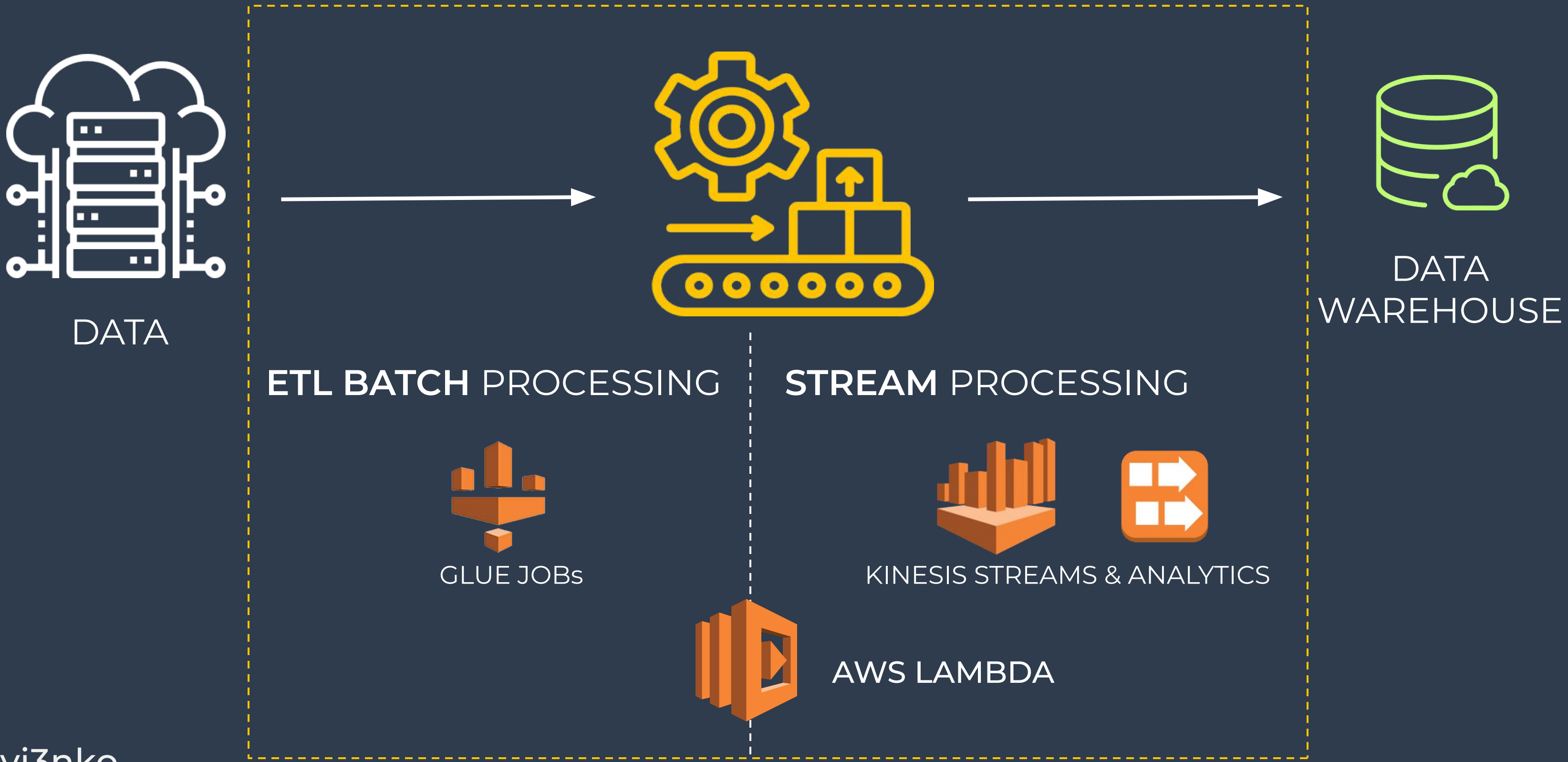
@matvi3nko

WHERE DOES THE DATA COME FROM?

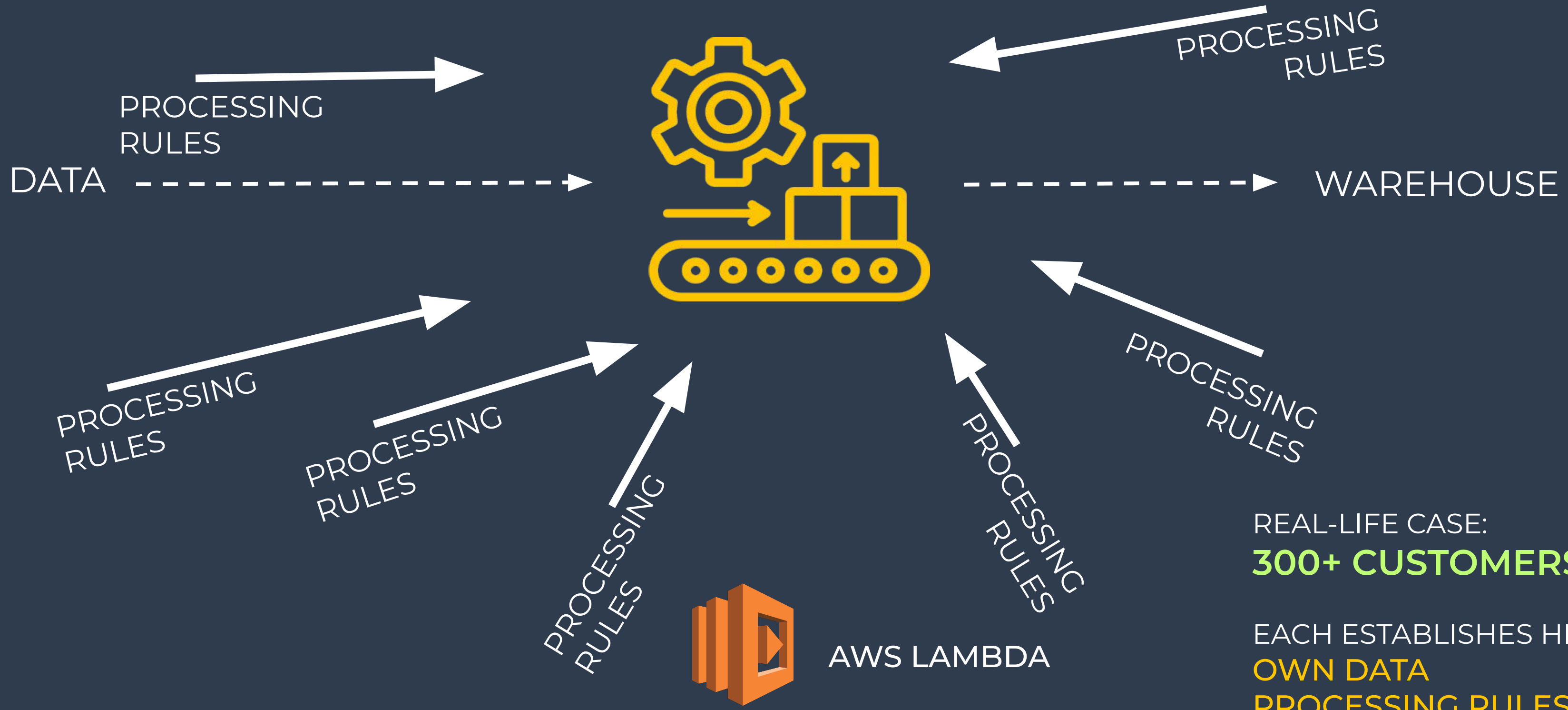

IoT DEVICES
REAL-TIME STREAM PROCESSING
DB
ENTERPRISE PLATFORMS
BATCH / STREAM PROCESSING
API
API
DATA LAKE
DB

@matvi3nko

USE EXISTING OR BUILD YOUR OWN PIPELINE



DYNAMICALLY UPDATED PROCESSING LOGIC

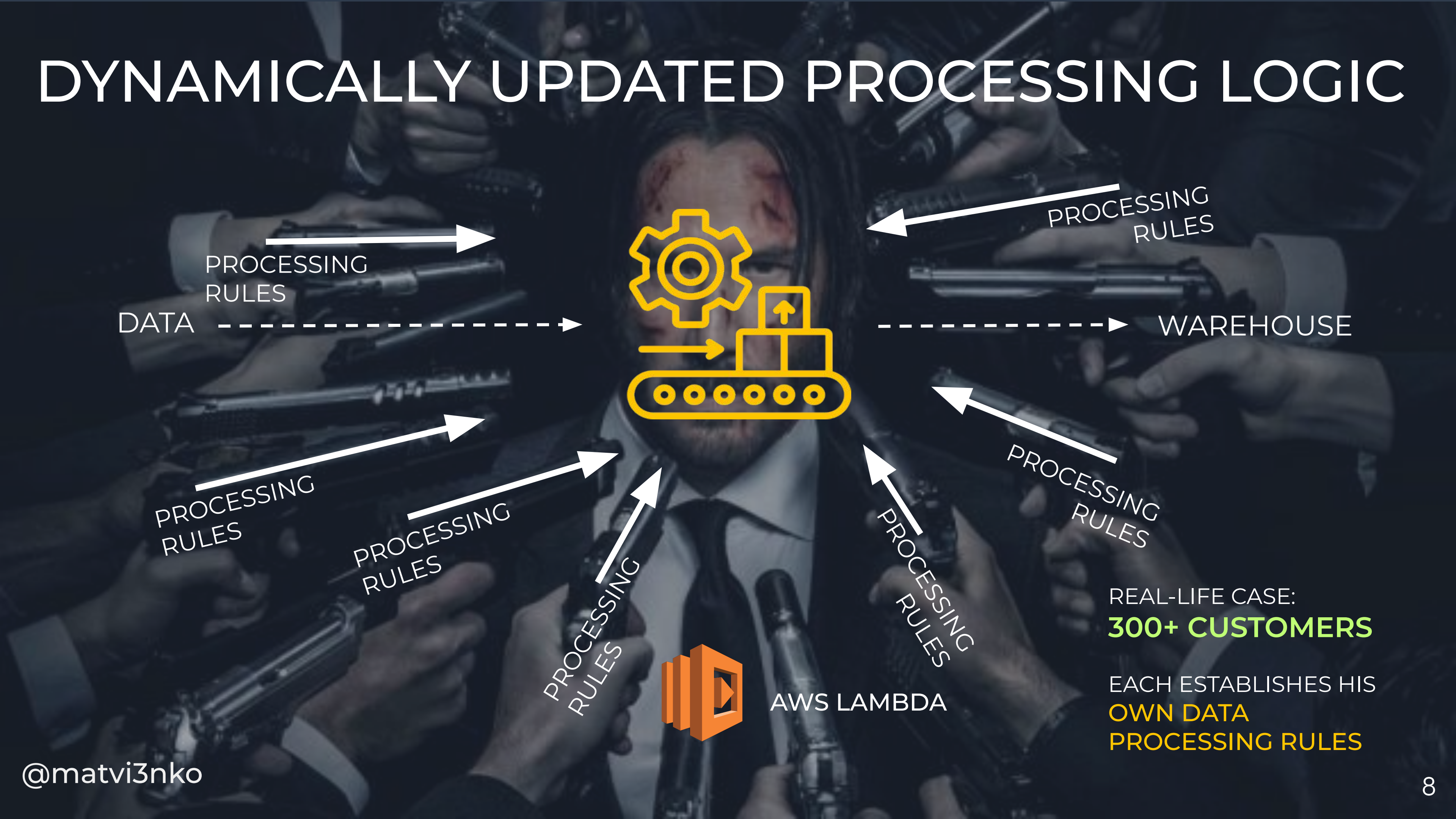


REAL-LIFE CASE:
300+ CUSTOMERS

EACH ESTABLISHES HIS
OWN DATA
PROCESSING RULES

@matvi3nko

DYNAMICALLY UPDATED PROCESSING LOGIC



REAL-LIFE CASE:
300+ CUSTOMERS

EACH ESTABLISHES HIS
OWN DATA
PROCESSING RULES

@matvi3nko

SERVERLESS COMPUTE SERVICE

AWS **LAMBDA**

SERVERLESS FUNCTION

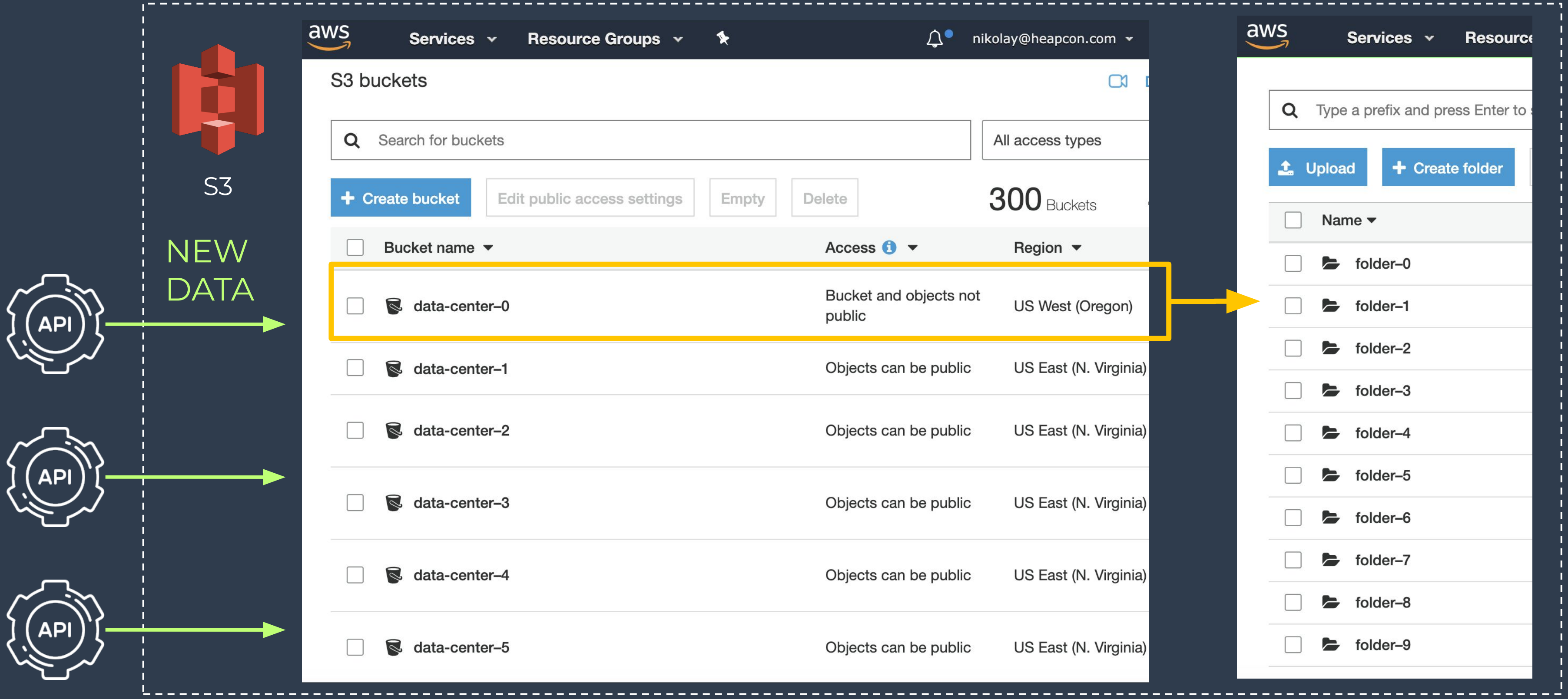
```
export const handler = async (event) => {
 const data = event.Records[0].body;

 // - TRANSFORM data
 // - WRITE to DB or
 // - PUT TO QUEUE/STREAM
 return 'success';
};
```

- CHOOSE YOUR CODE LANGUAGE
- NO INFRASTRUCTURE TO MANAGE
- TRIGGERED BY EVENTS
- HIGH SCALABLE
- STATELESS
- COST-EFFECTIVE

* Symbol in the presentation

@matvi3nko

AWS S3 FOR DATA LAKE

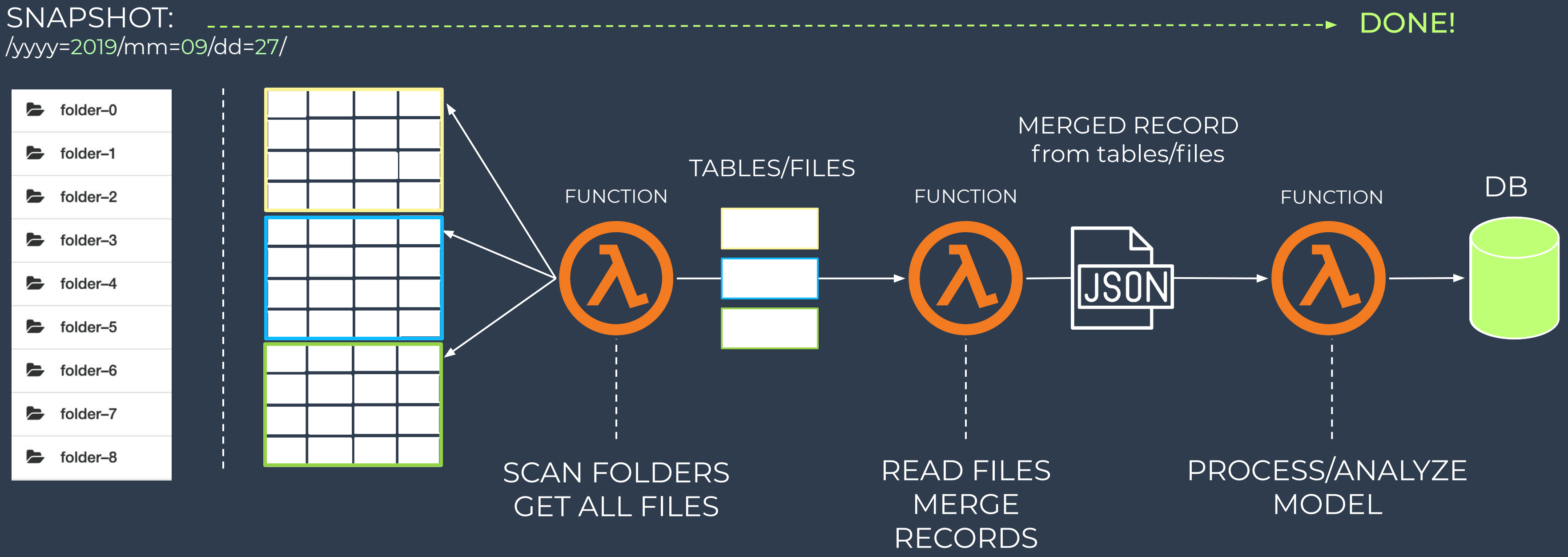


Bucket name	Access	Region
data-center-0	Bucket and objects not public	US West (Oregon)
data-center-1	Objects can be public	US East (N. Virginia)
data-center-2	Objects can be public	US East (N. Virginia)
data-center-3	Objects can be public	US East (N. Virginia)
data-center-4	Objects can be public	US East (N. Virginia)
data-center-5	Objects can be public	US East (N. Virginia)

Name
folder-0
folder-1
folder-2
folder-3
folder-4
folder-5
folder-6
folder-7
folder-8
folder-9

BATCH DATA PROCESSING

AWS LAMBDA ICON REPLACED ON



@matvi3nko

TRANSPORT SERVICES

NOTIFICATION
TOPIC

FAN-OUT

- PUB/SUB
- NOT WORRIED ABOUT DELIVERY

KINESIS
STREAMS

DATA STREAMS

REAL-TIME PROCESSING

- BATCH 1–10000 RECORDS
- NOT AUTOSCALE
- 6 MB MESSAGE SIZE LIMIT
- EXACTLY ONCE PROCESSING
- HOURLY COST

QUEUE

BATCH PROCESSING

- BATCH 1–10 RECORDS
- AUTOSCALE
- 256 KB MESSAGE SIZE LIMIT
- AT LEAST ONCE PROCESSING

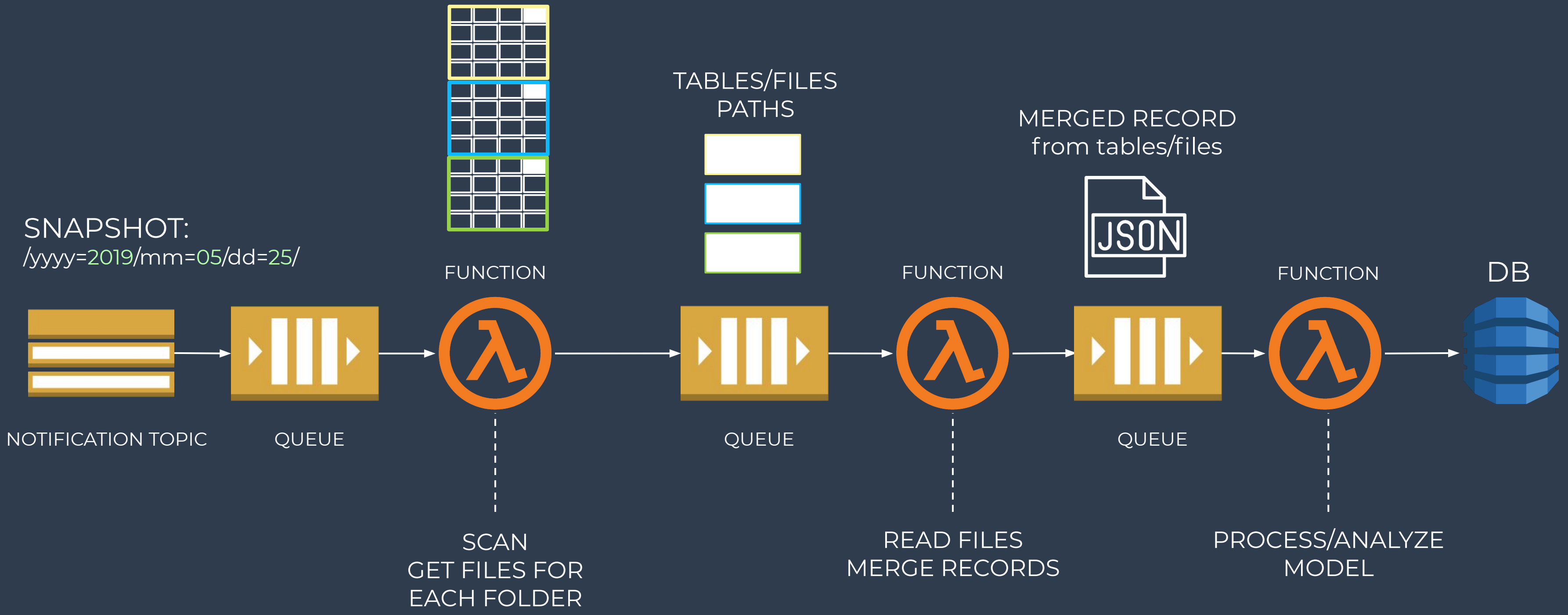
DYNAMODB
STREAMS

DYNAMODB
NoSQL DB

DISTRIBUTED TRANSACTIONS

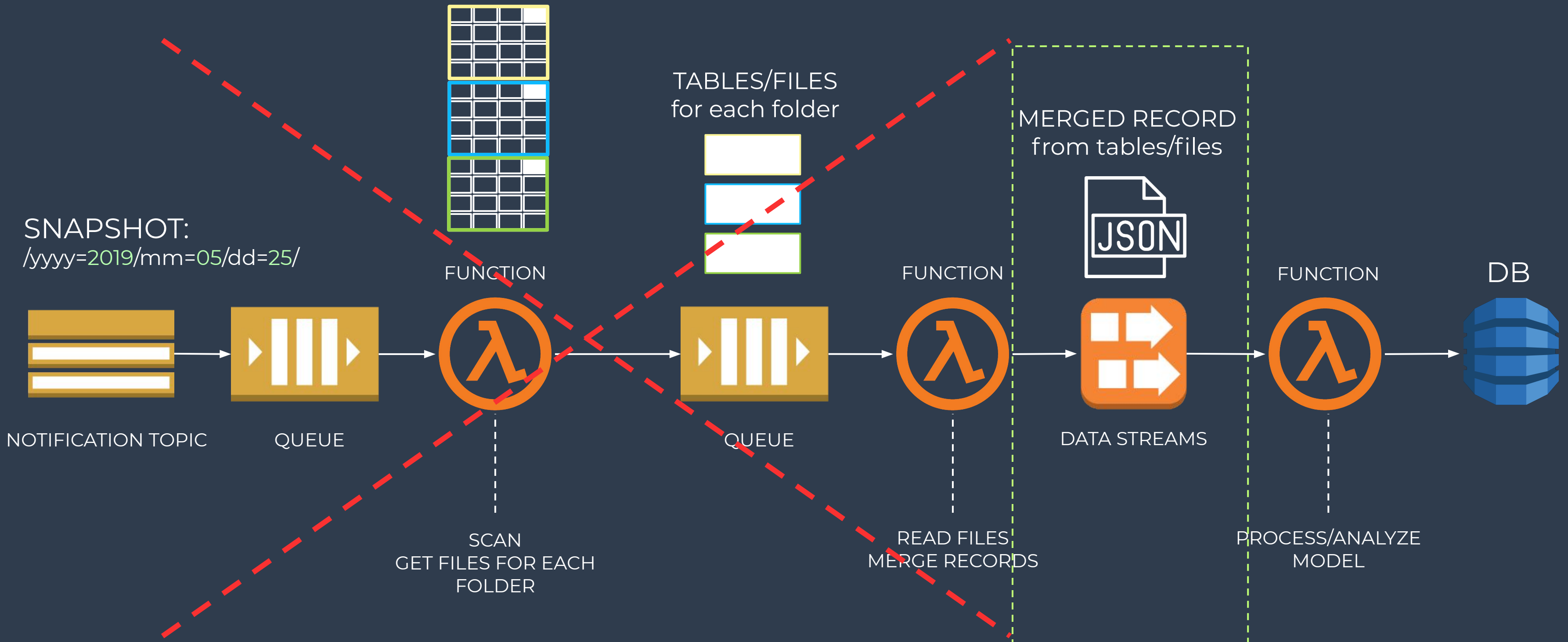
- BATCH 1–1000 RECORDS
- AUTOSCALE
- 6 MB MESSAGE SIZE LIMIT
- EXACTLY ONCE PROCESSING
- REACT ON EACH CHANGE

BATCH PROCESSING ARCHITECTURE



@matvi3nko

REAL-TIME PROCESSING ARCHITECTURE

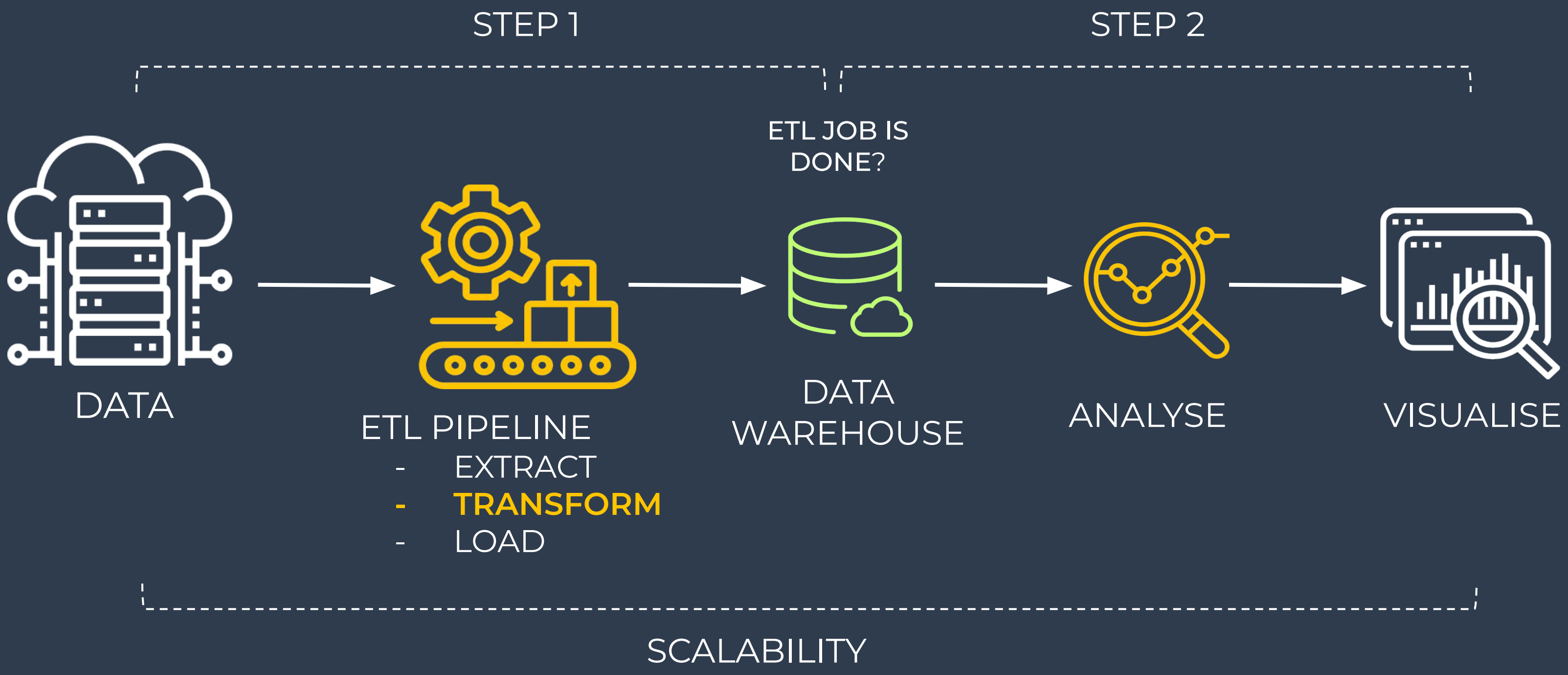


@matvi3nko

DATA EXTRACTION PATTERNS

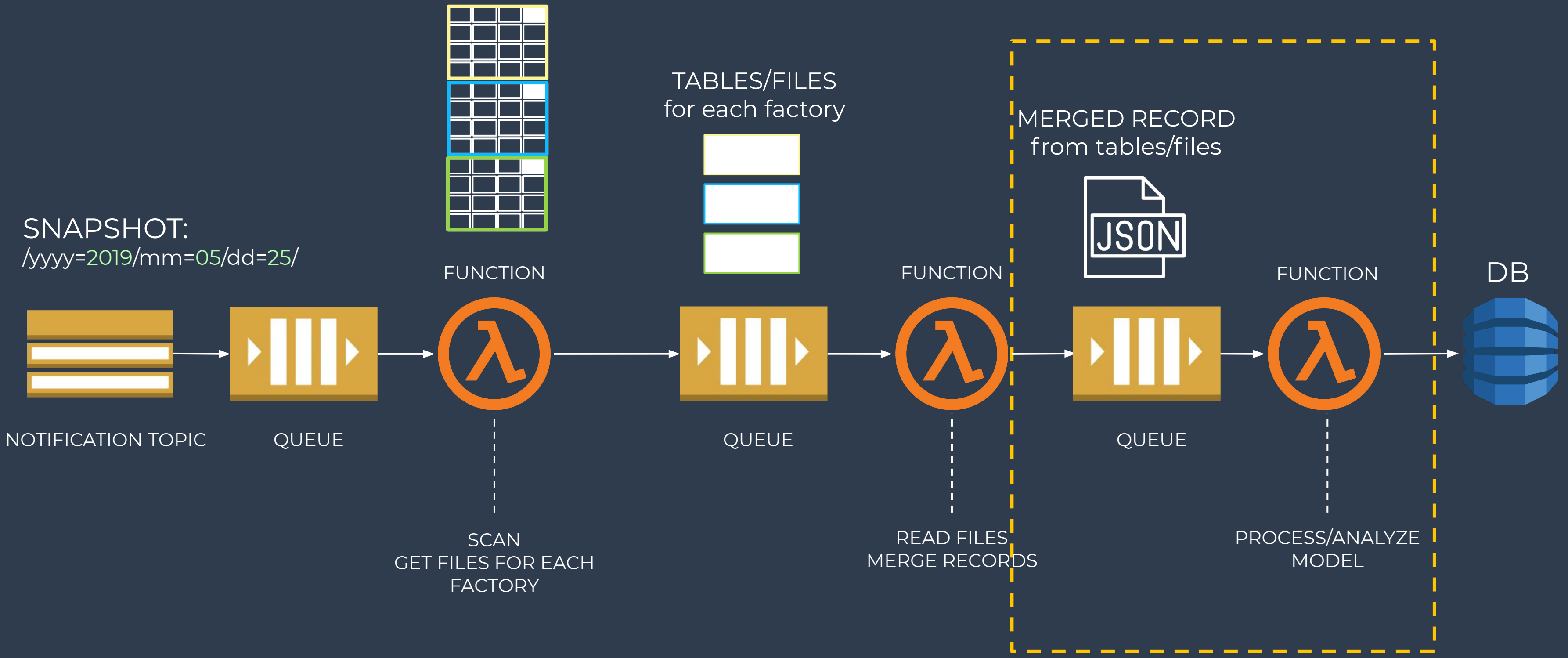
1. MOVE FROM BIG DATA TO A LARGE NUMBER OF MESSAGES
2. USE THE QUEUES FOR MESSAGES, AND DATA STREAMS TO TRANSFER MODELS / LARGE COLLECTION
3. BUT DO NOT RUSH TO USE STREAMS. CHOOSE TRANSPORT FOR YOUR NEEDS

DATA TRANSFORMATION



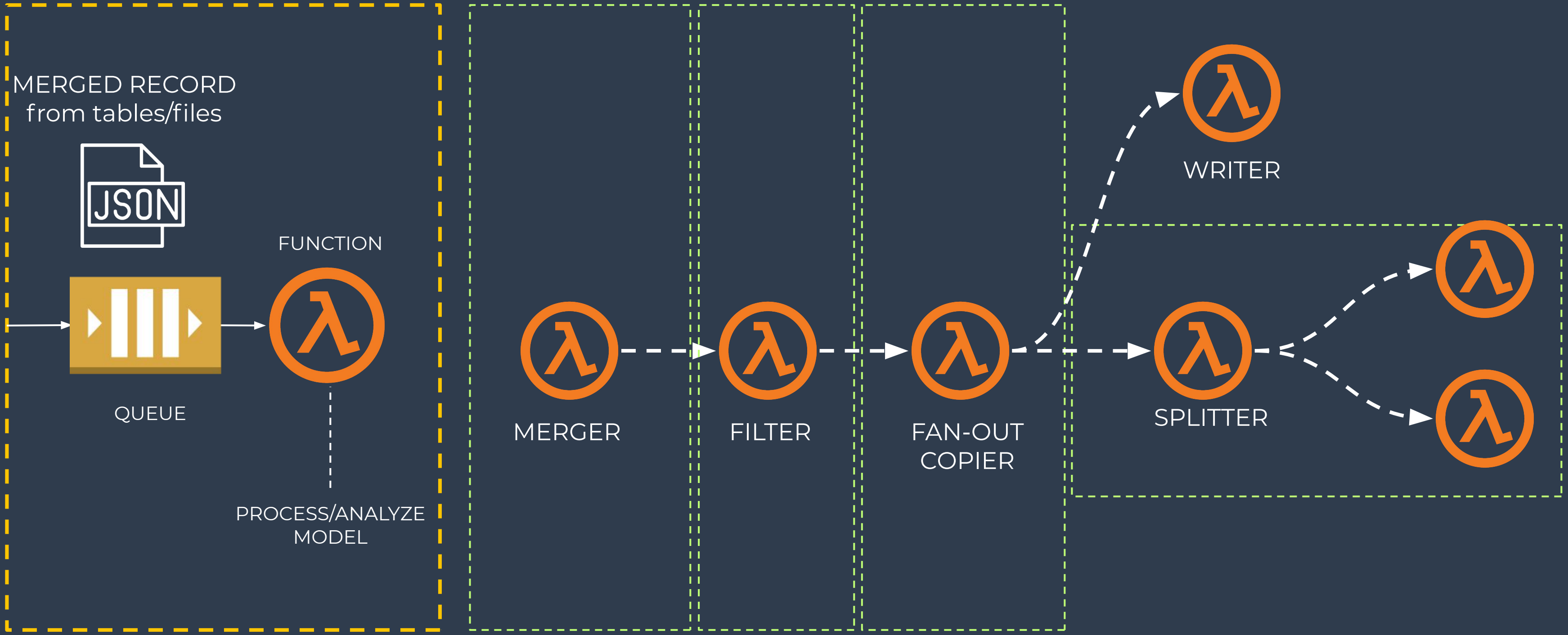
@matvi3nko

TRANSFORM SECTION

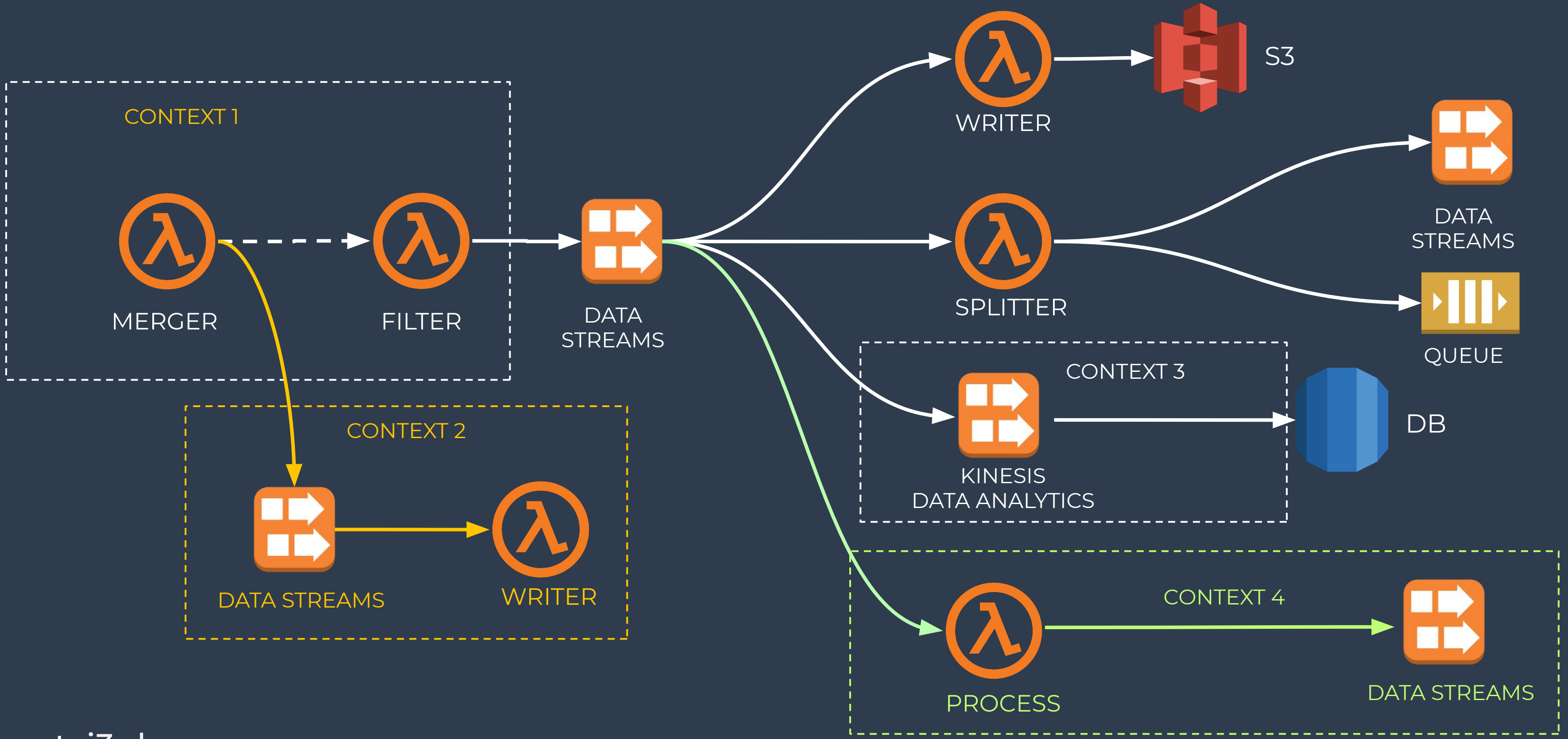


@matvi3nko

SEPARATE FUNCTIONS BY RESPONSIBILITY

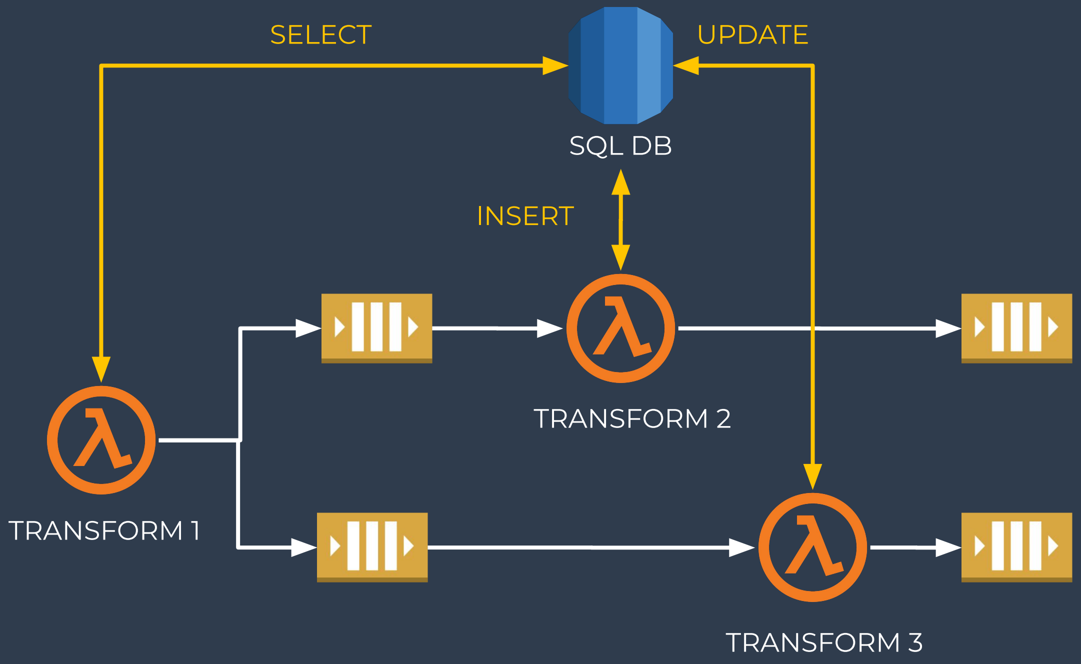


BOUNDARY CONTEXT



@matvi3nko

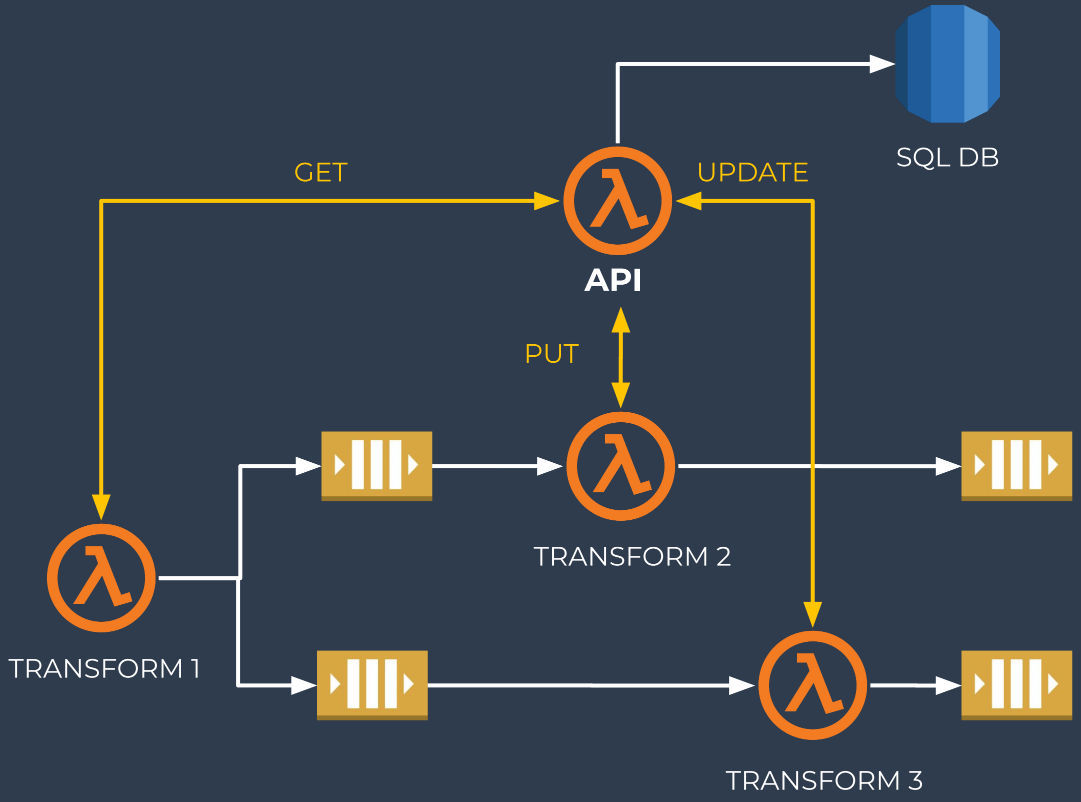
QUERIES & CONNECTIONS PROBLEMS



PROBLEMS:

1. MANY LAMBDAS HAS TO QUERY SQL DB
2. A LOT OF CONNECTIONS
3. A LOT OF DEPENDENCIES

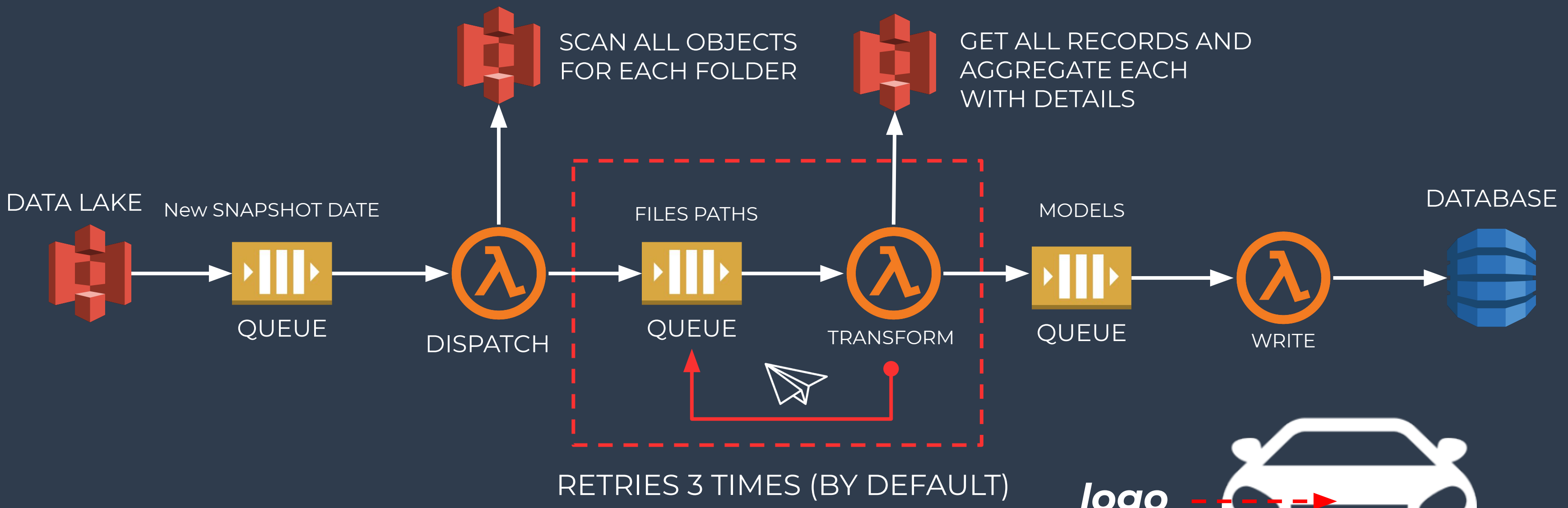
DB QUERY LOGIC ENCAPSULATION

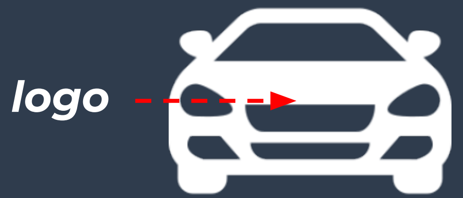


BENEFITS:

1. QUERIES & LOGIC IS HIDDEN BEHIND THE API
2. LESS CONNECTIONS, CONTROLLED CONNECTIONS POOL
3. LESS DEPENDENCIES

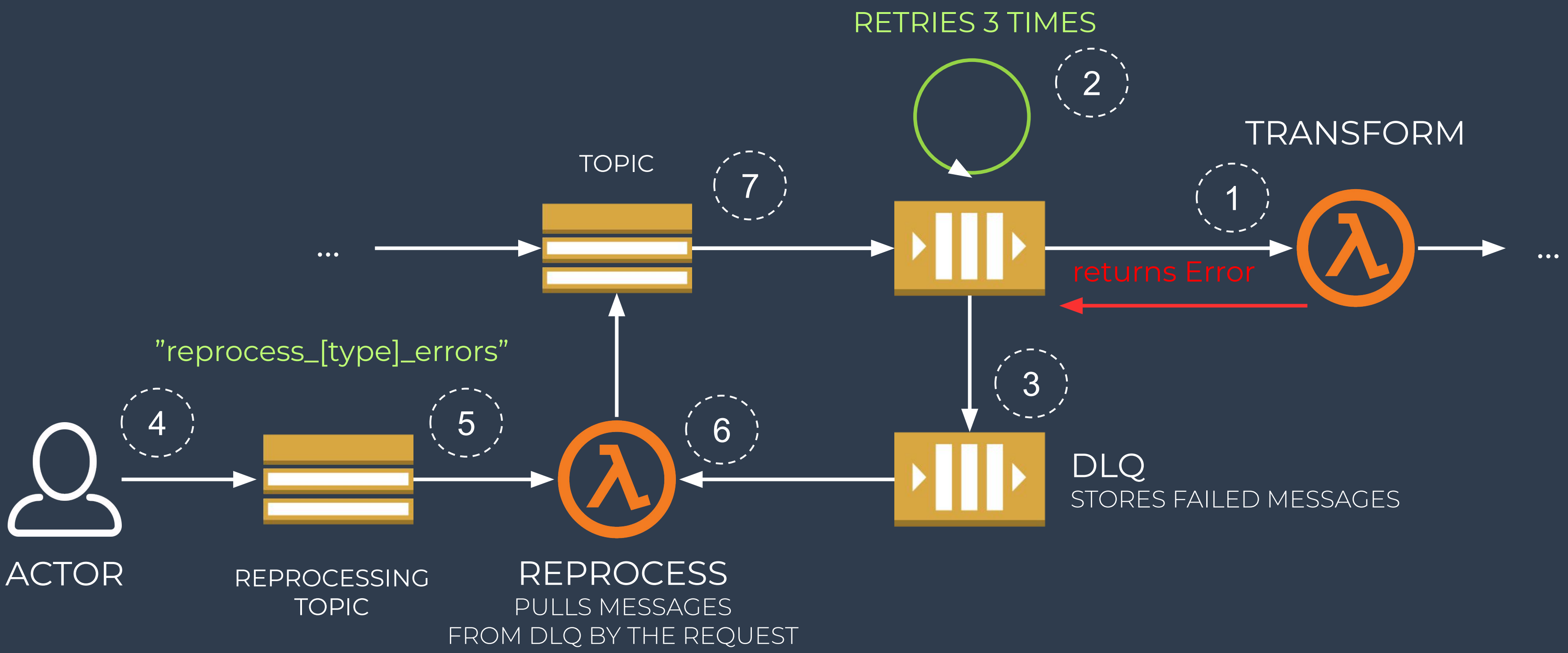
RELIABILITY: RETRY STRATEGY





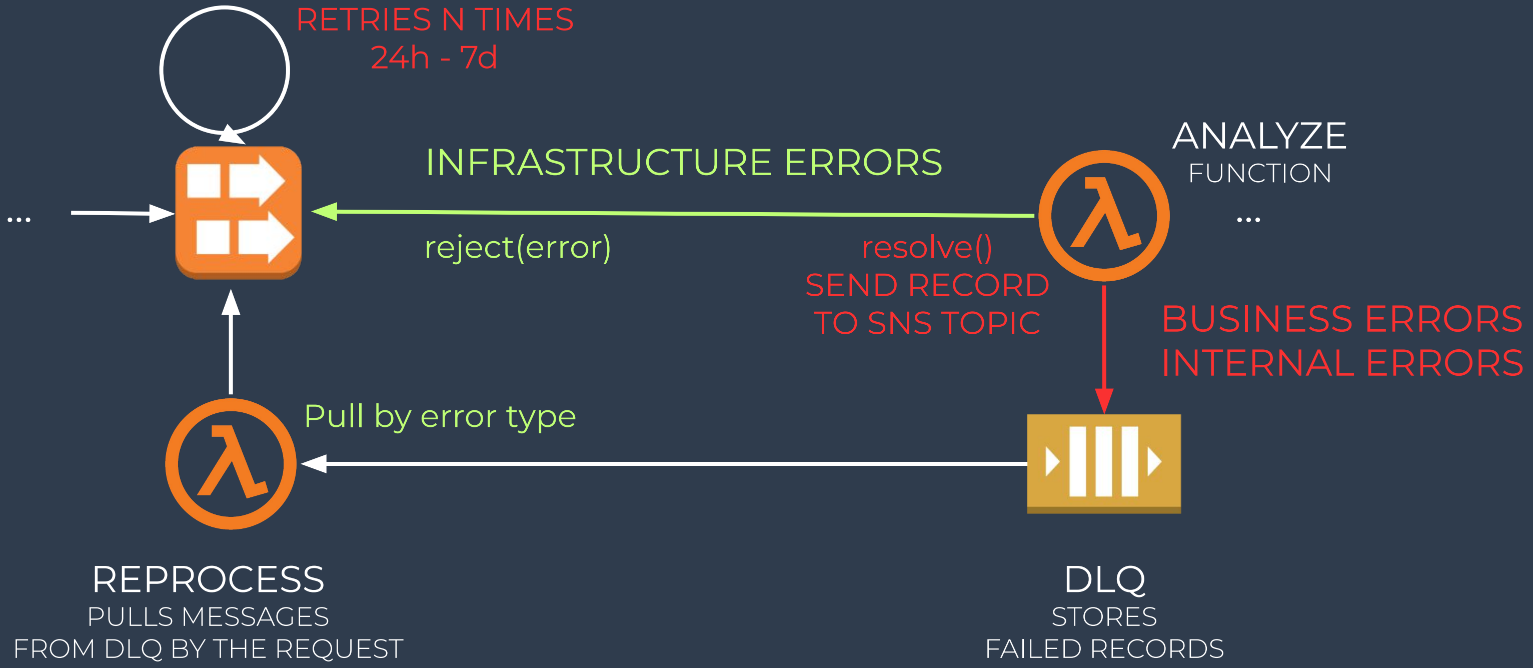
@matvi3nko

DEAD LETTER QUEUE



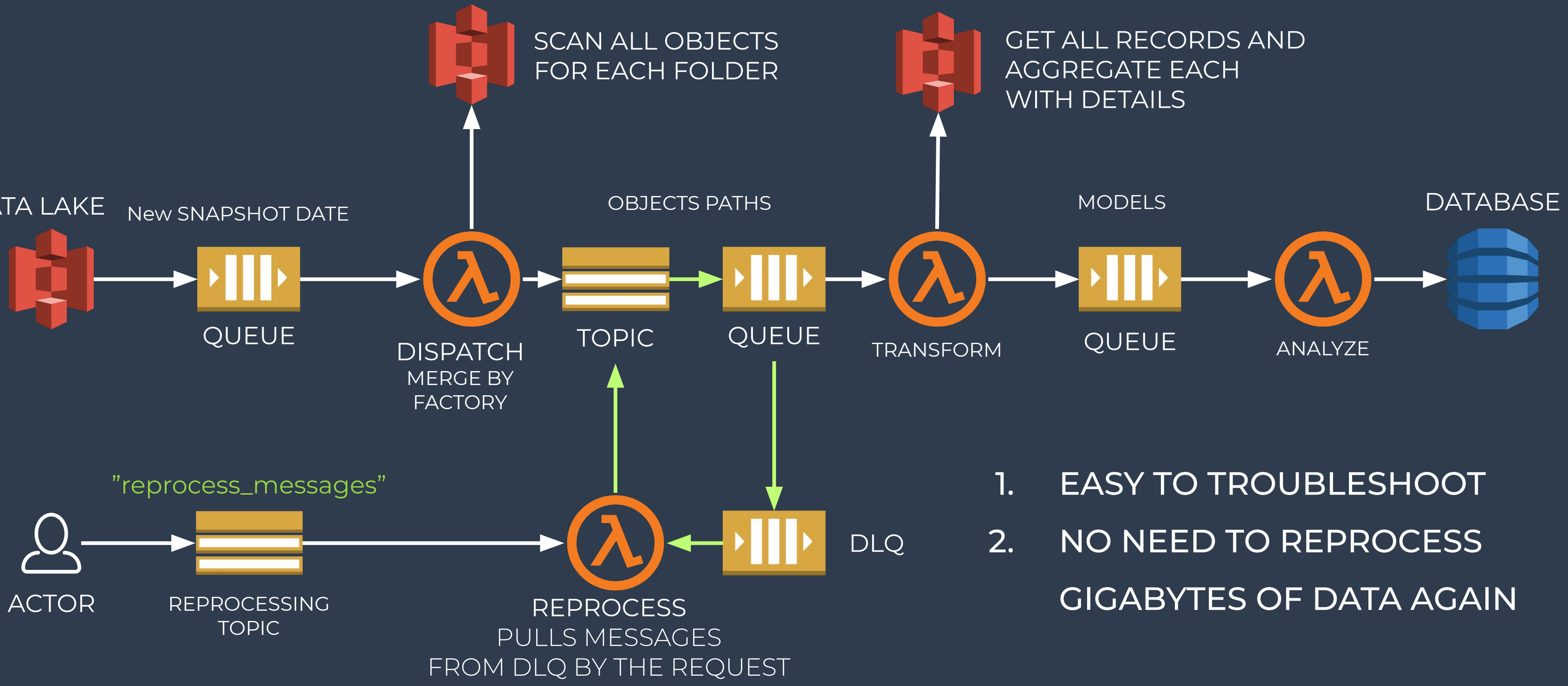
@matvi3nko

KINESIS ERROR HANDLING



@matvi3nko

DLQ FOR THE QUEUE



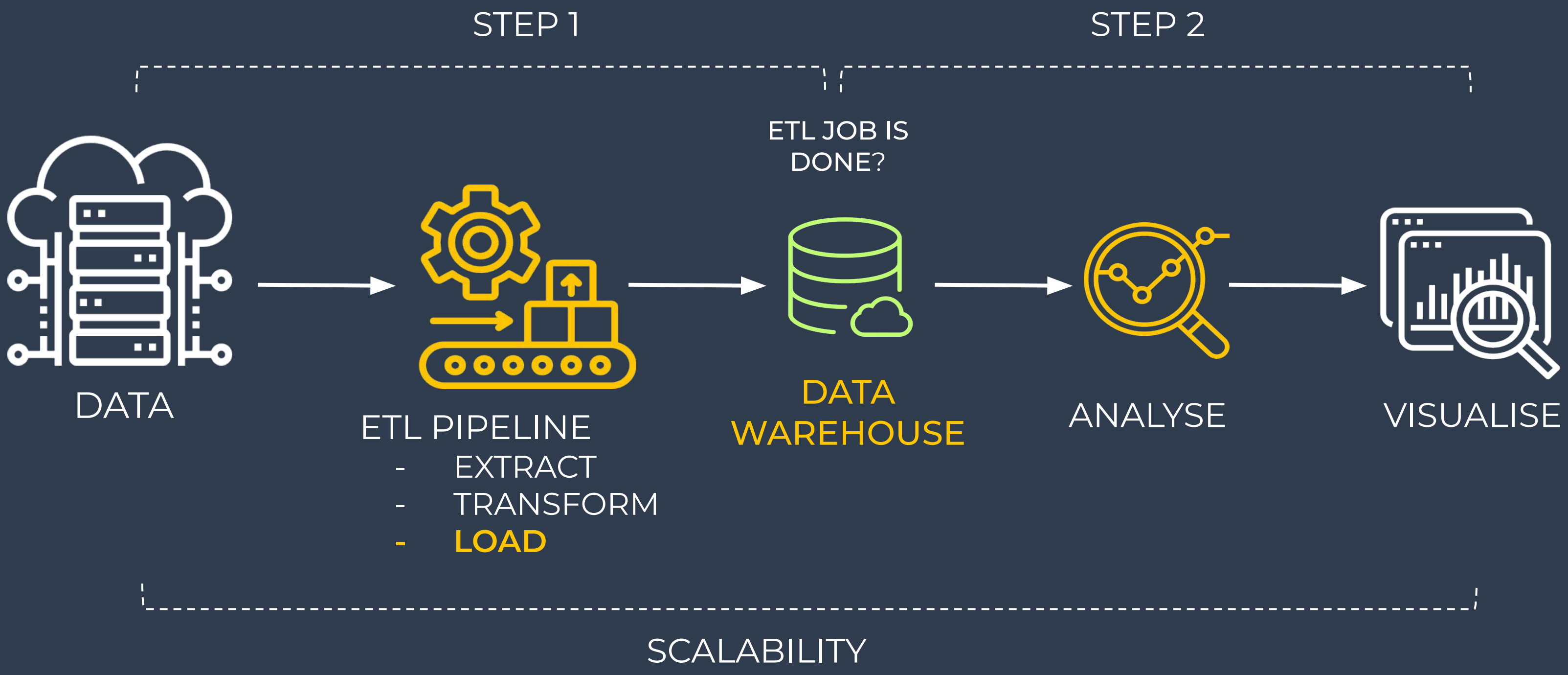
1. EASY TO TROUBLESHOOT
2. NO NEED TO REPROCESS GIGABYTES OF DATA AGAIN

@matvi3nko

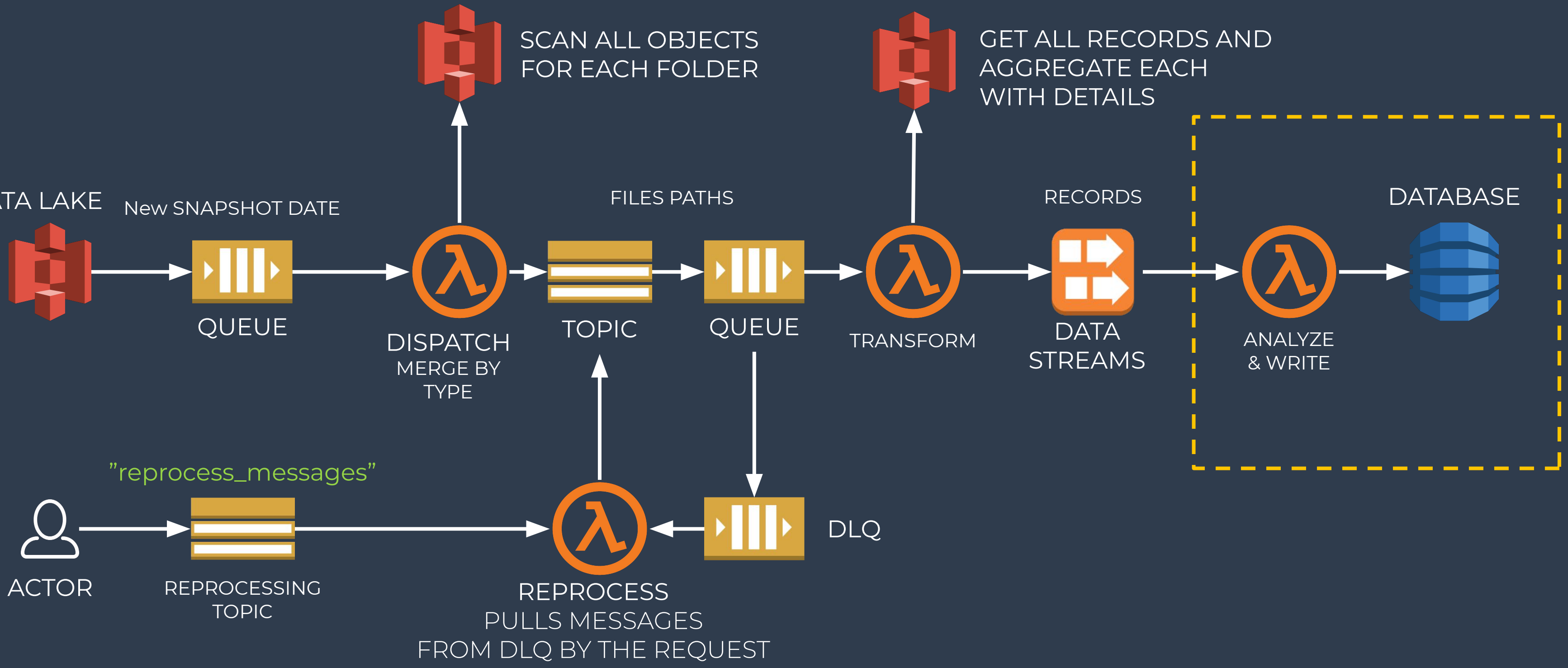
DATA TRANSFORMATION PATTERNS

1. ONE LAMBDA FUNCTION – ONE RESPONSIBILITY
2. DIVIDE THE PIPELINE INTO BOUNDARY CONTEXTS WITH FIXED DATA INTERFACES
3. ENCAPSULATE DB QUERIES BEHIND API FUNCTION
4. USE DEAD LETTER QUEUES FOR RELIABILITY

DATA LOADING

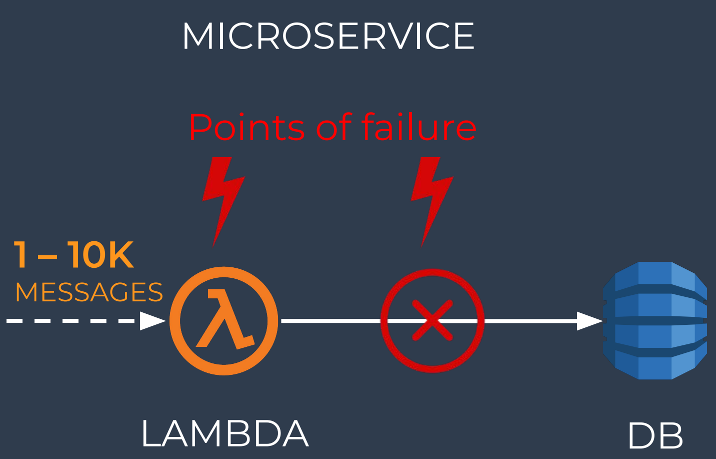


LOAD SECTION



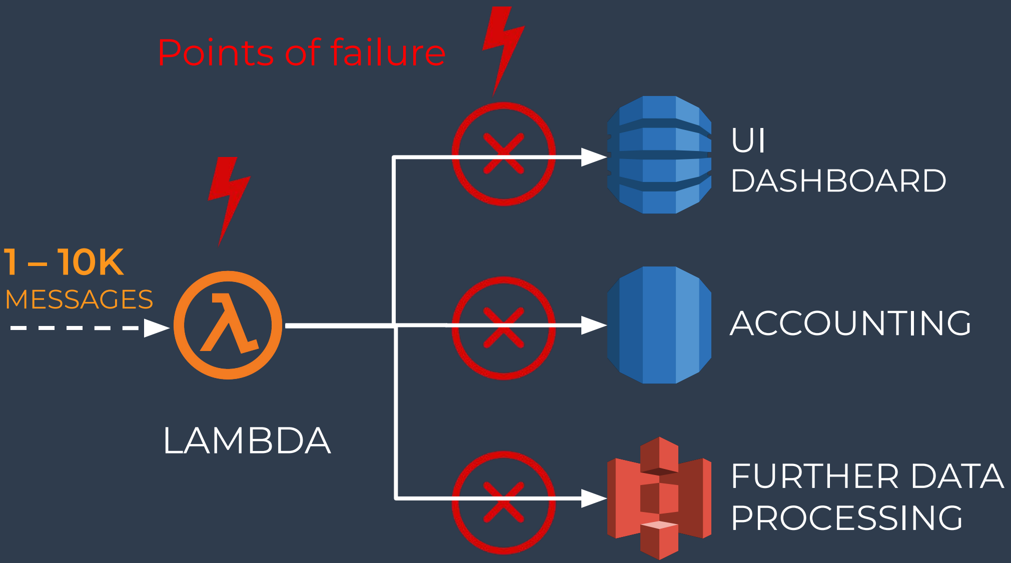
@matvi3nko

WRITING TO A DATABASES



PROCESSES DATA
ANR/OR FILTERS DATA

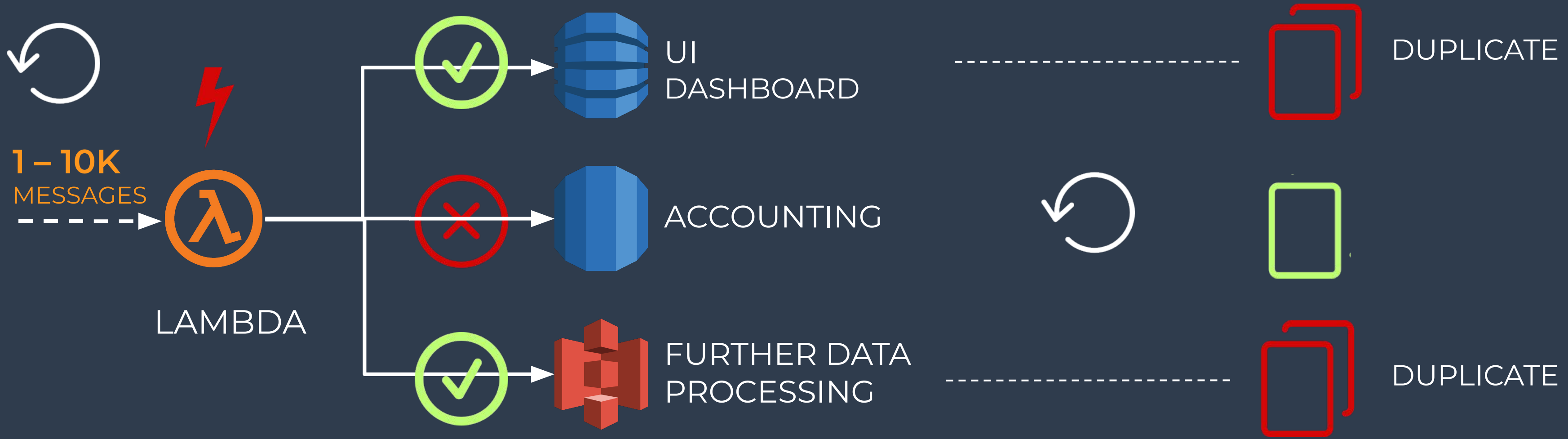
WRITES TO DB



1. RETRY THE BATCH?
2. RETRY FAILED RECORD MANUALLY IN LAMBDA?

@matvi3nko

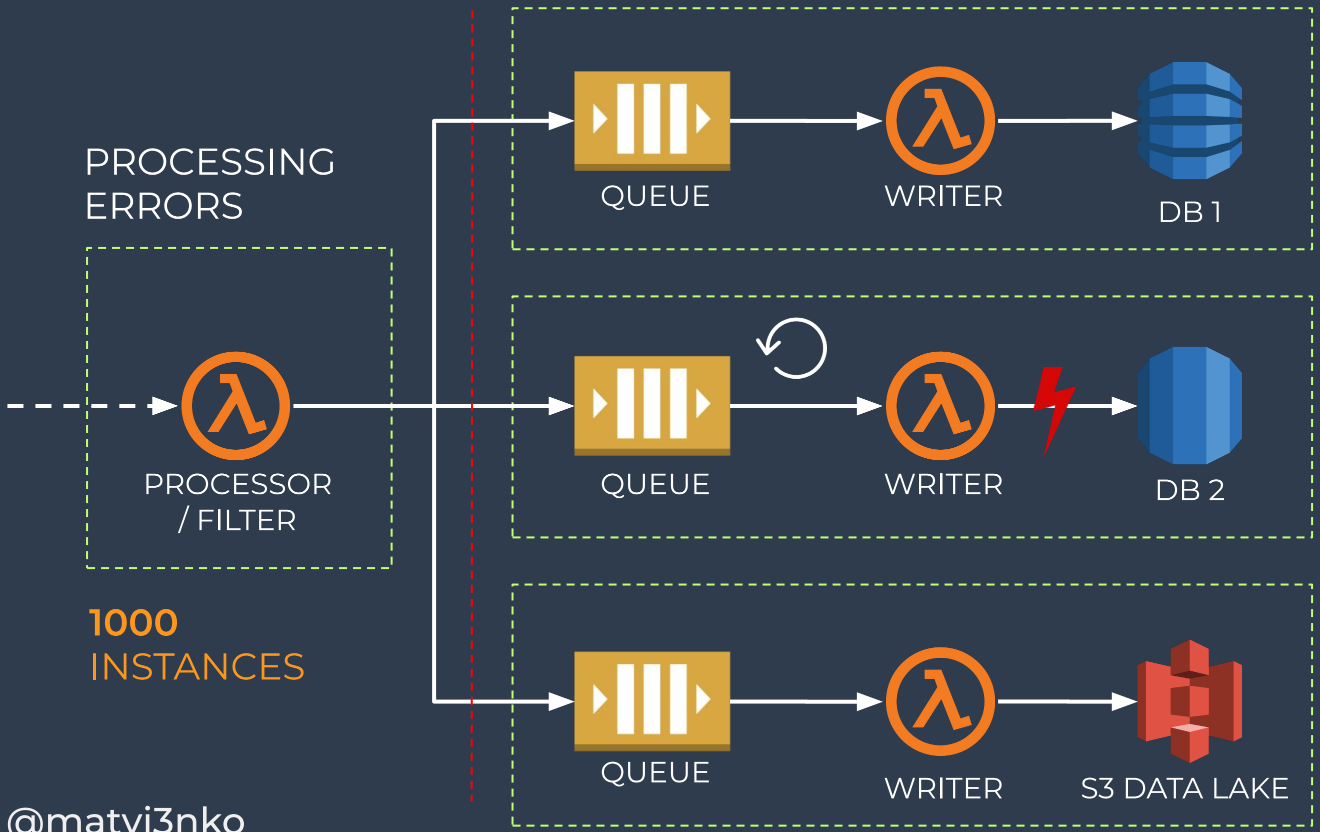
RETRY PROBLEM



1. [BAD] REMOVE/ROLLBACK SUCCESSFUL RECORDS

2. [BAD] ORCHESTRATE REPEAT IN LAMBDA

DECOUPLED WRITERS



INFRASTRUCTURE ERRORS

200 INSTANCES

INDEPENDENT RETRIES

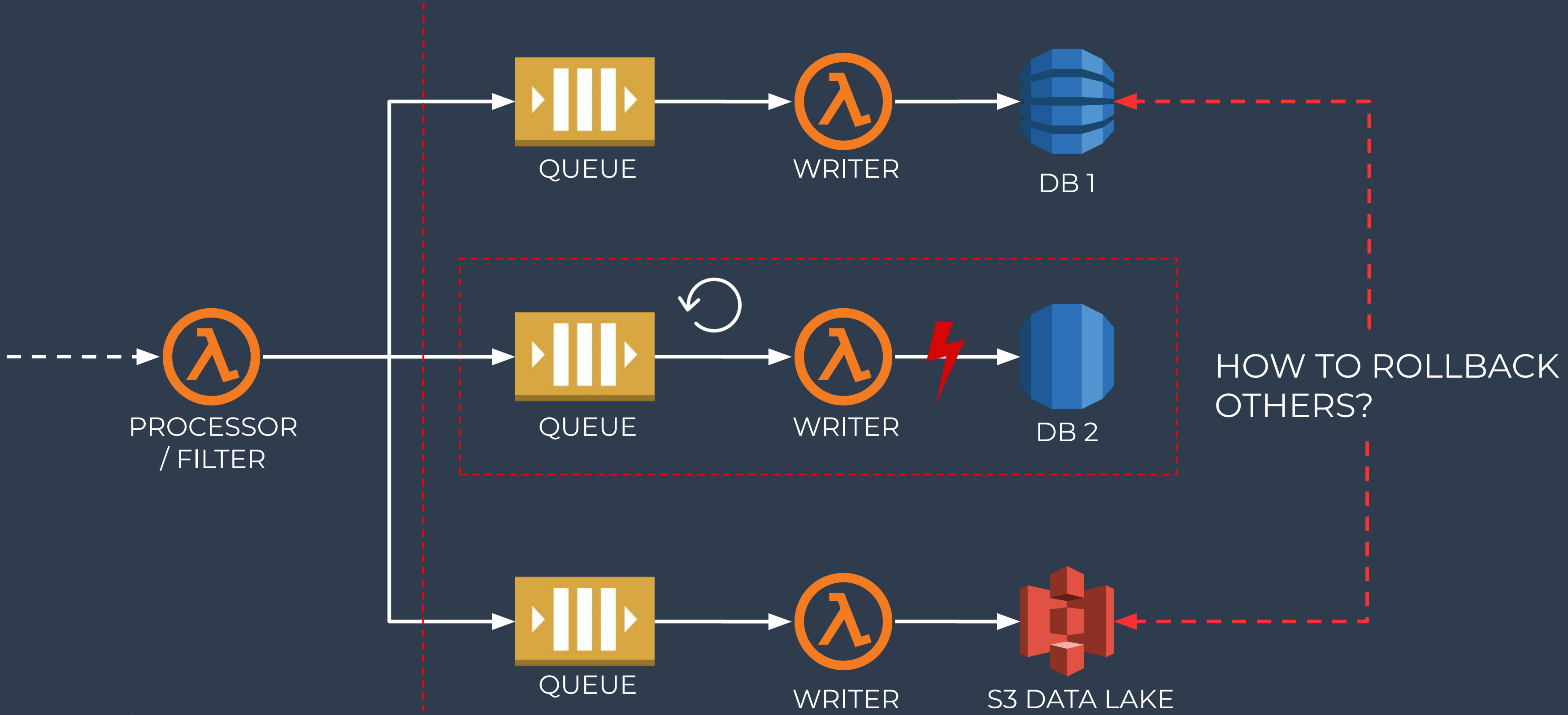
100 INSTANCES

BETTER SCALABILITY

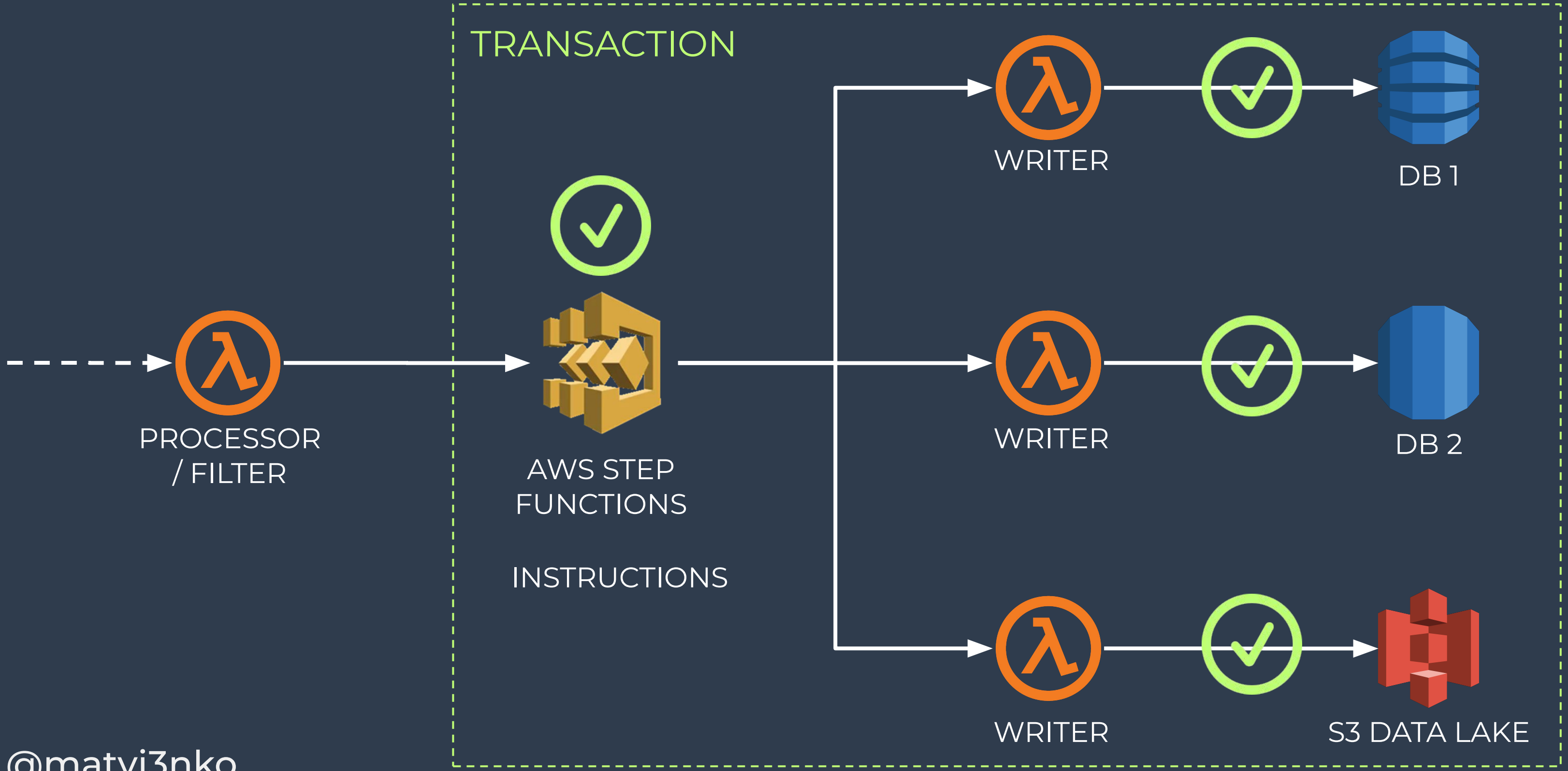
500 INSTANCES

@matvi3nko

DISTRIBUTED TRANSACTIONS



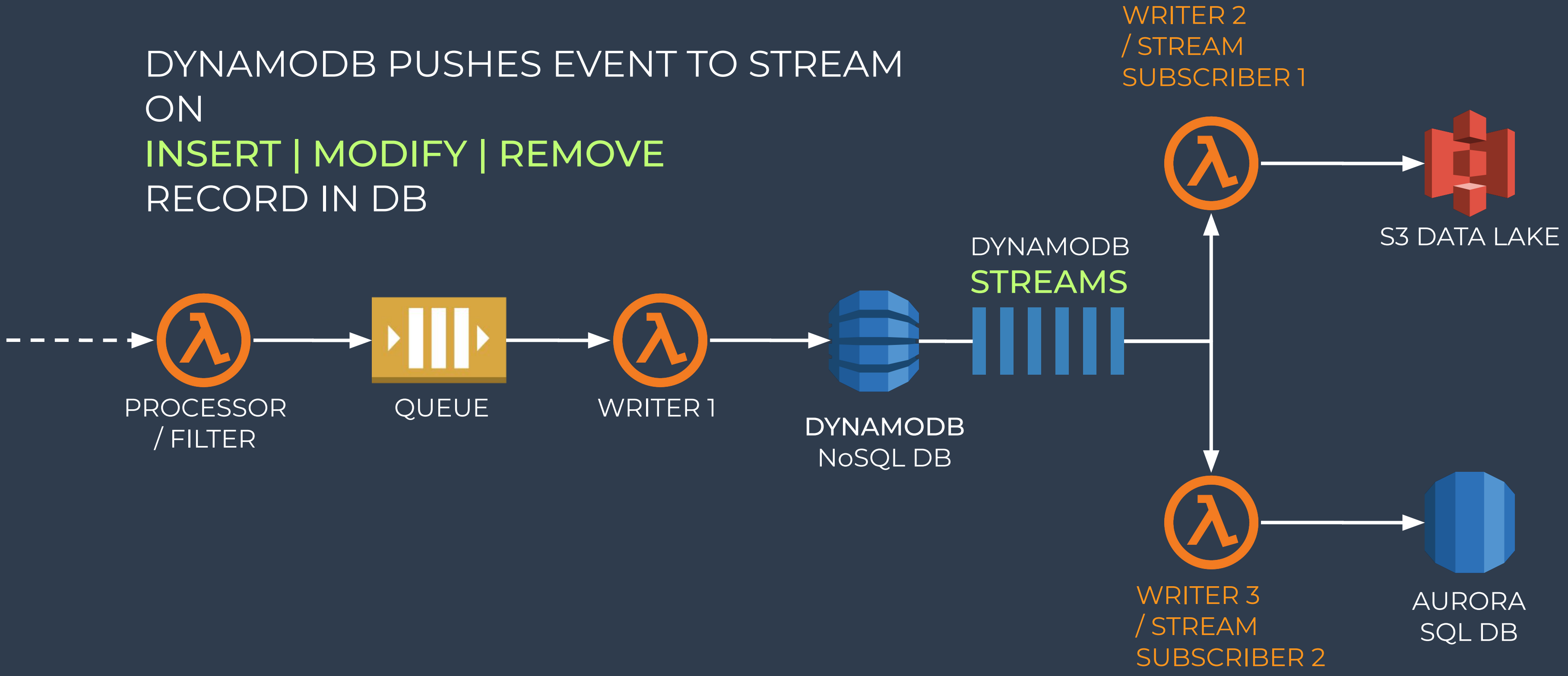
USE AWS STEP FUNCTIONS



@matvi3nko

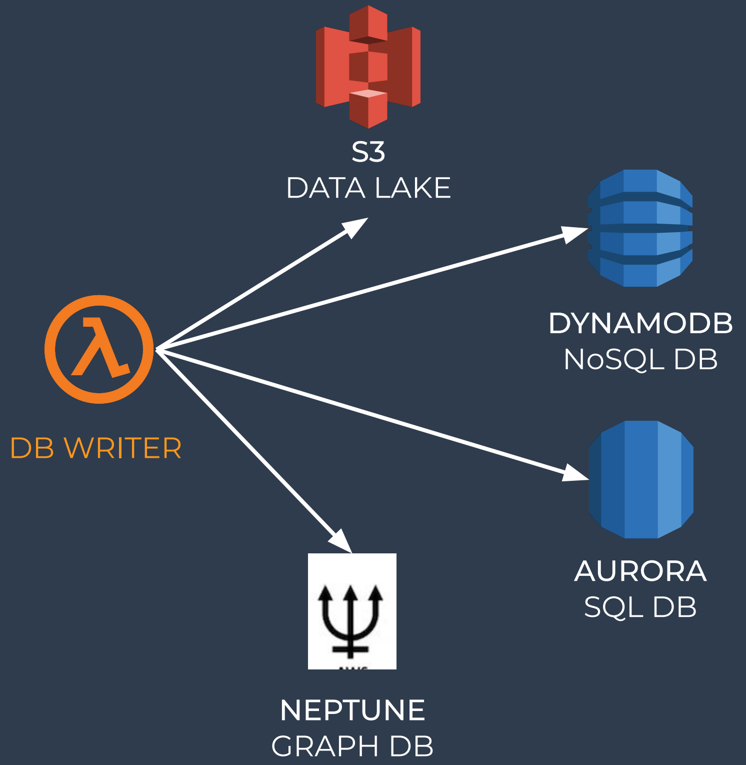
DISTRIBUTED TRANSACTIONS

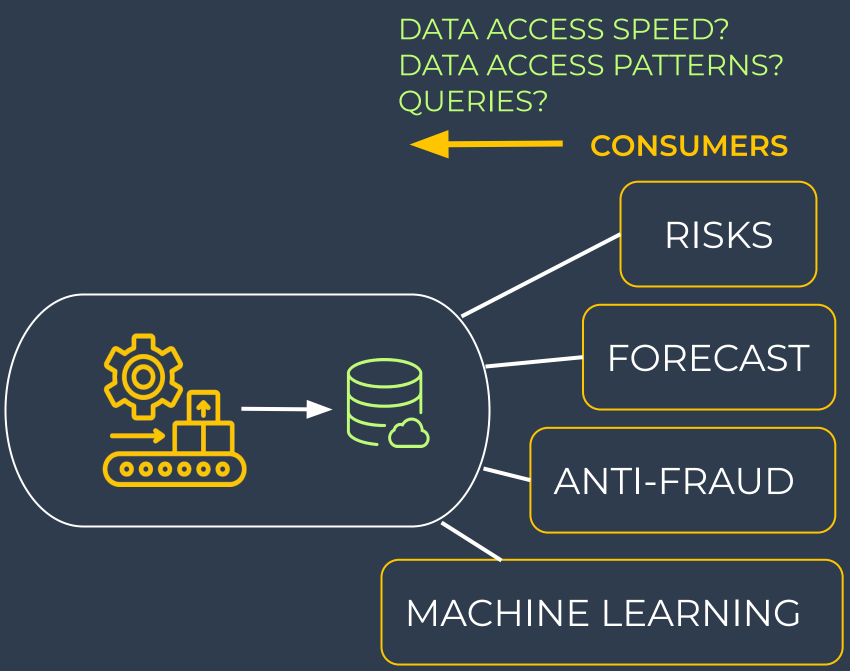
DYNAMODB PUSHES EVENT TO STREAM
ON
INSERT | MODIFY | REMOVE
RECORD IN DB



@matvi3nko

DATA WAREHOUSE



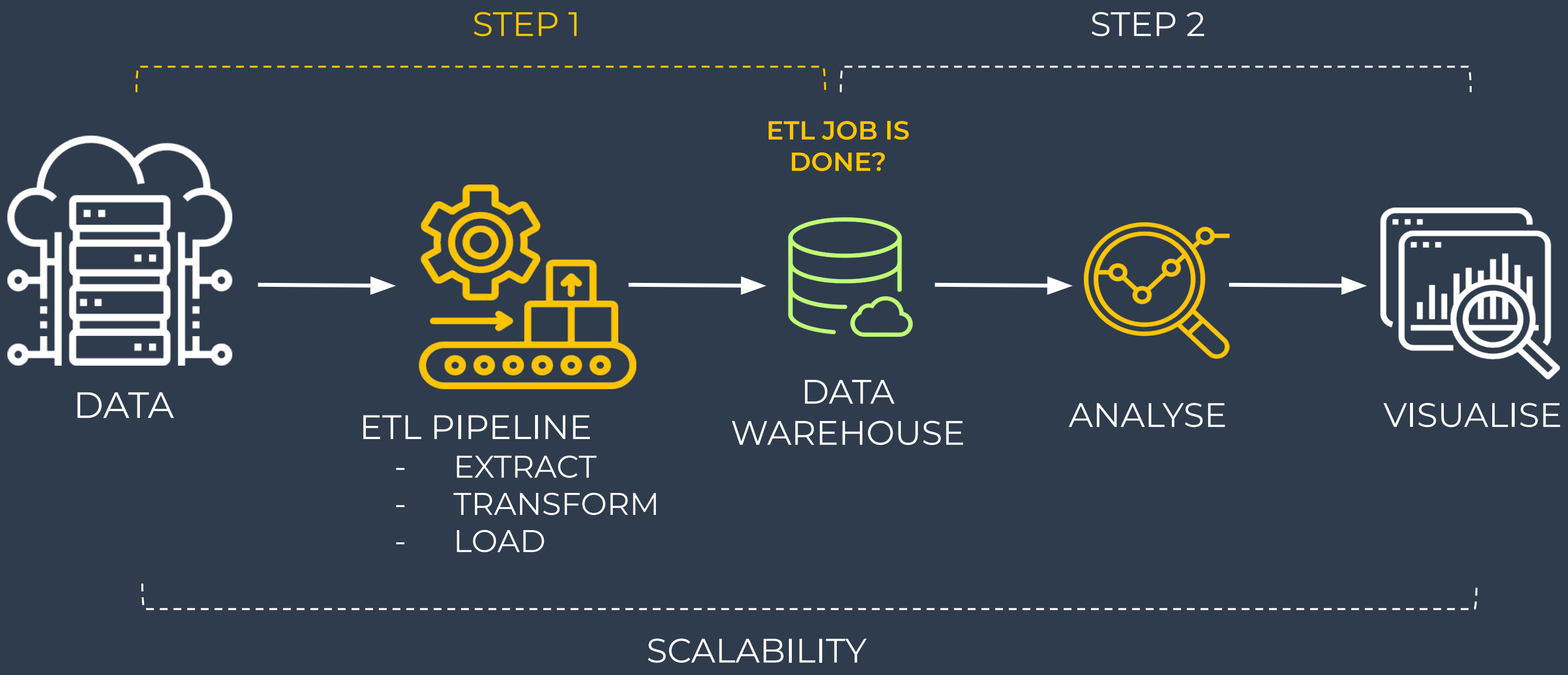


@matvi3nko

DATA LOADING PATTERNS

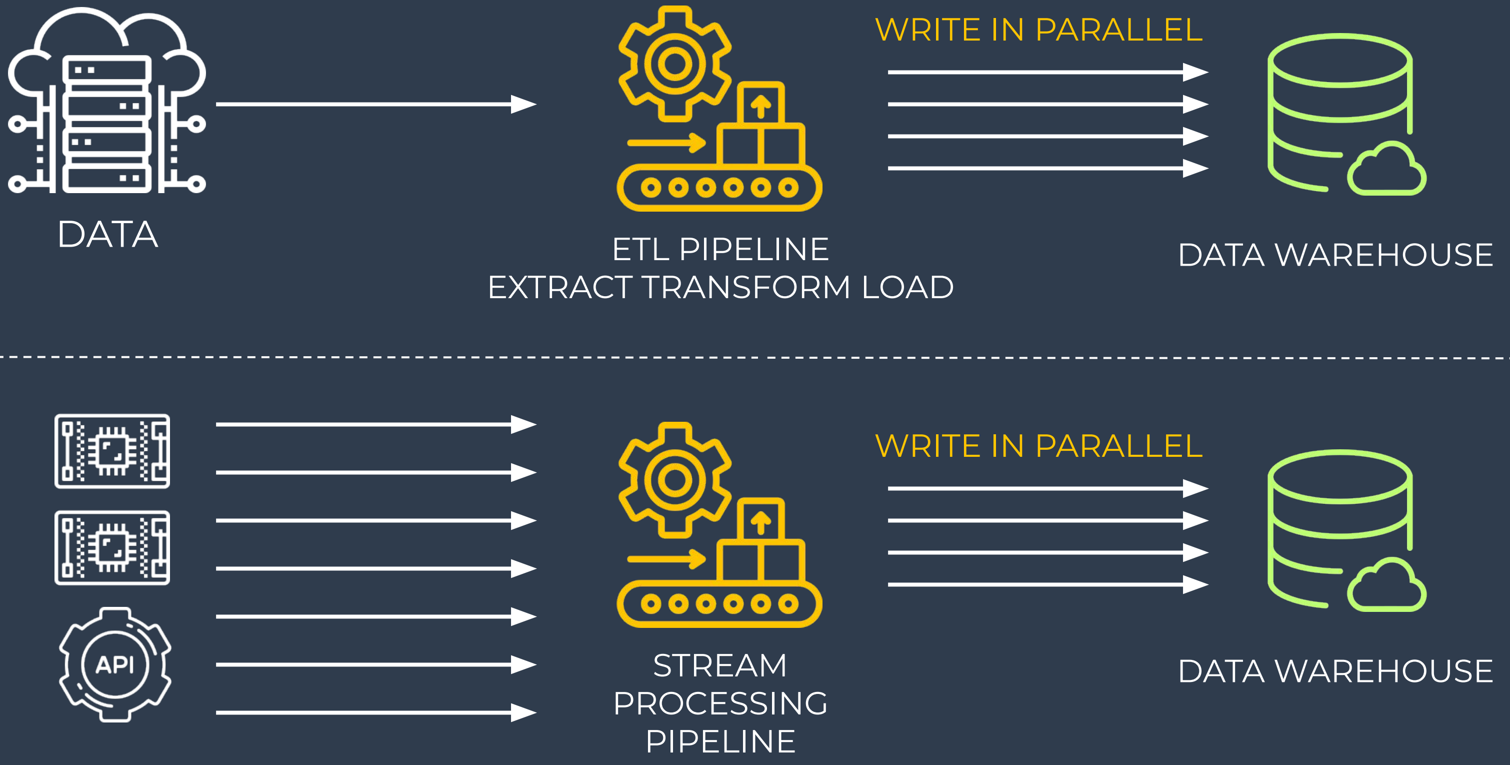
1. DECOUPLE FUNCTIONS BY POINTS OF FAILURE
2. USE INFRASTRUCTURE AS A CODE VS ORCHESTRATIONS IN THE CODE
3. USE AWS STEP FUNCTIONS AND DYNAMODB STREAMS FOR TRANSACTIONS
4. PIPELINE IS THE CORE, THINK ABOUT FUTURE PLATFORMS AROUND IT AND HOW THEY WILL HAVE ACCESS TO DATA

HOW TO UNDERSTAND THAT THE JOB IS COMPLETED?



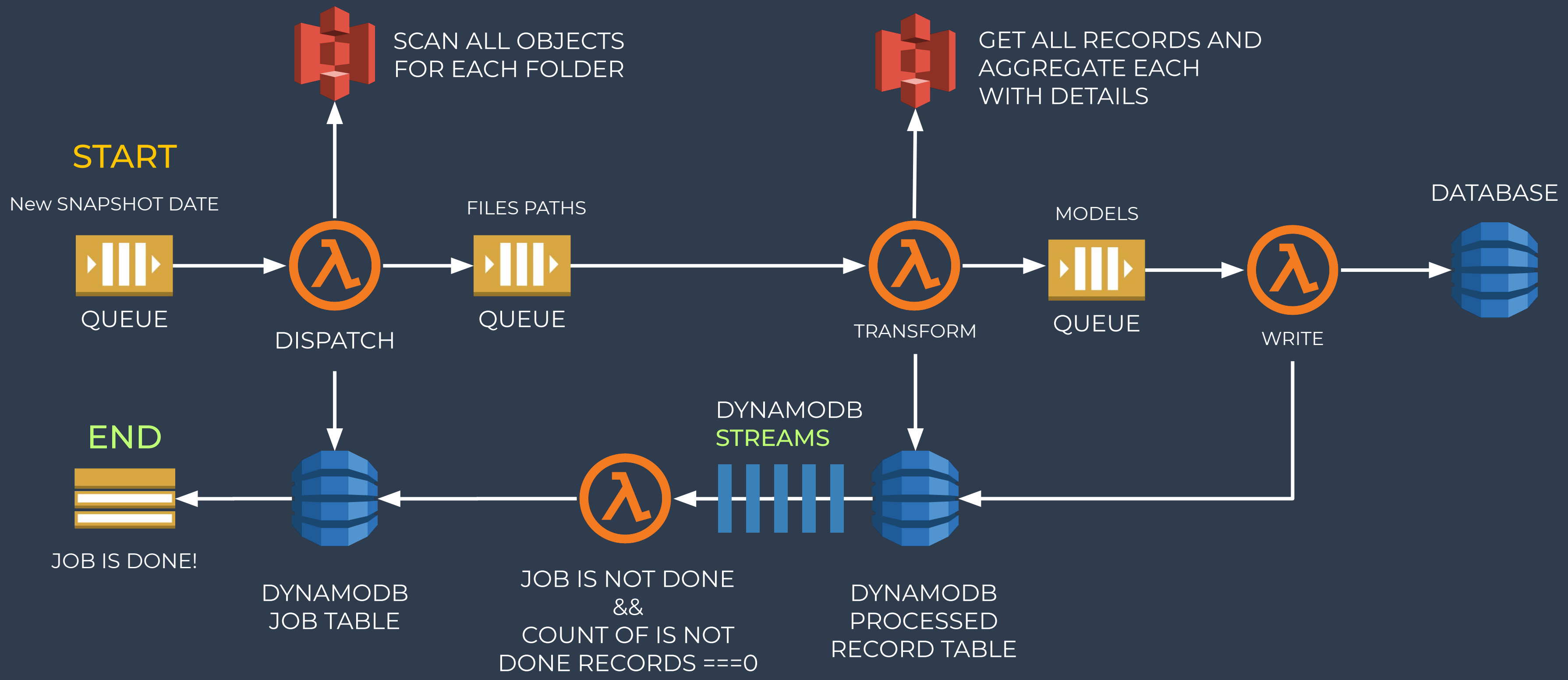
@matvi3nko

THE PROBLEM OF DISTRIBUTED DATA PROCESSING



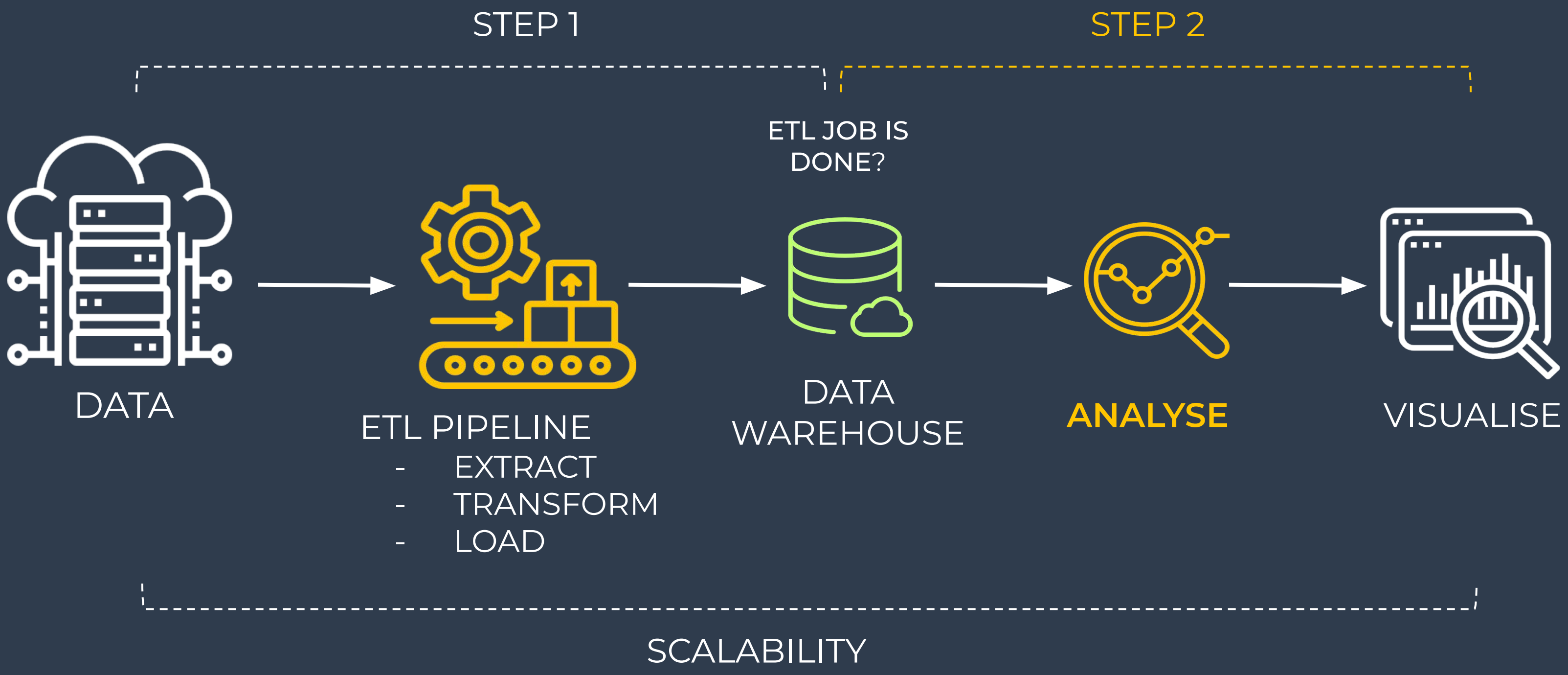
@matvi3nko

ETL JOB STATE CONTROL



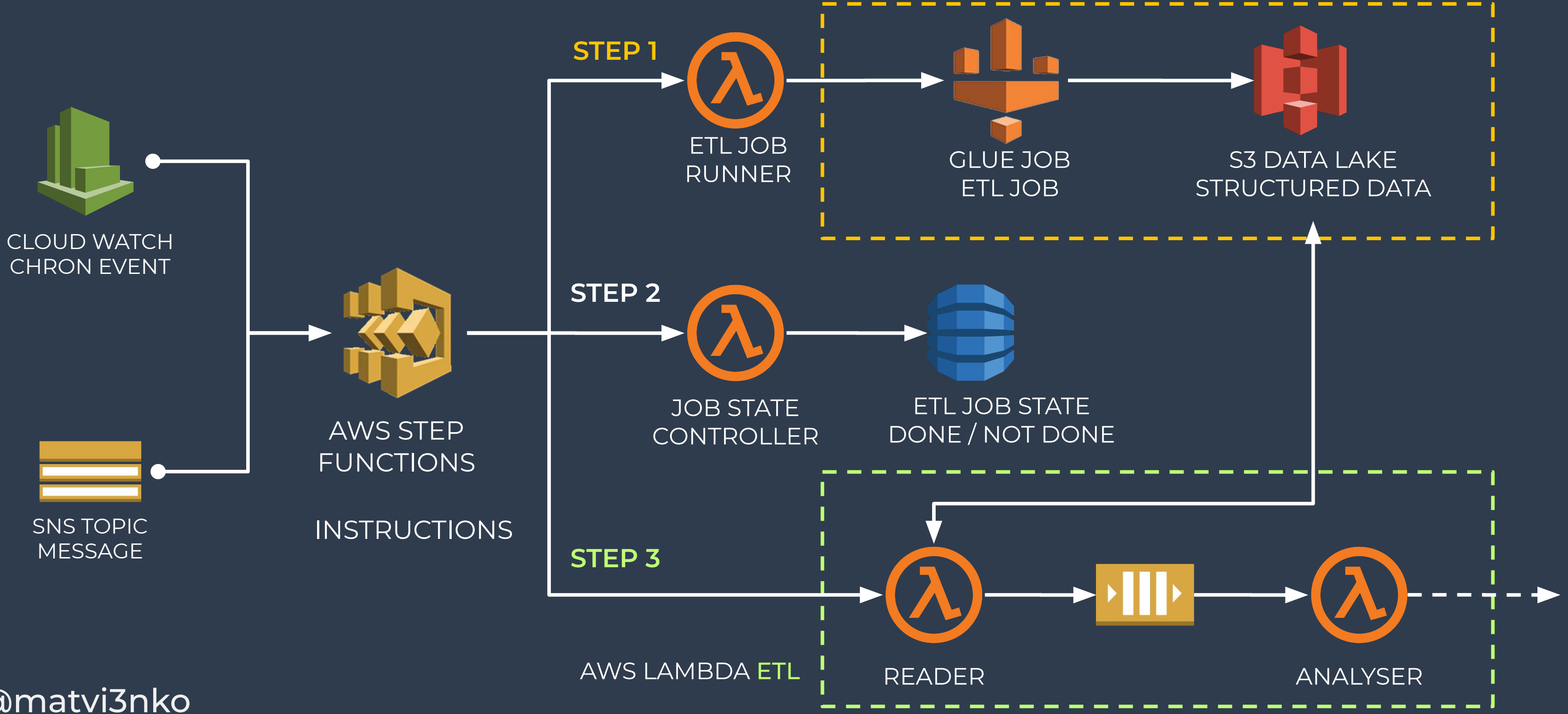
@matvi3nko

DATA ANALYSIS



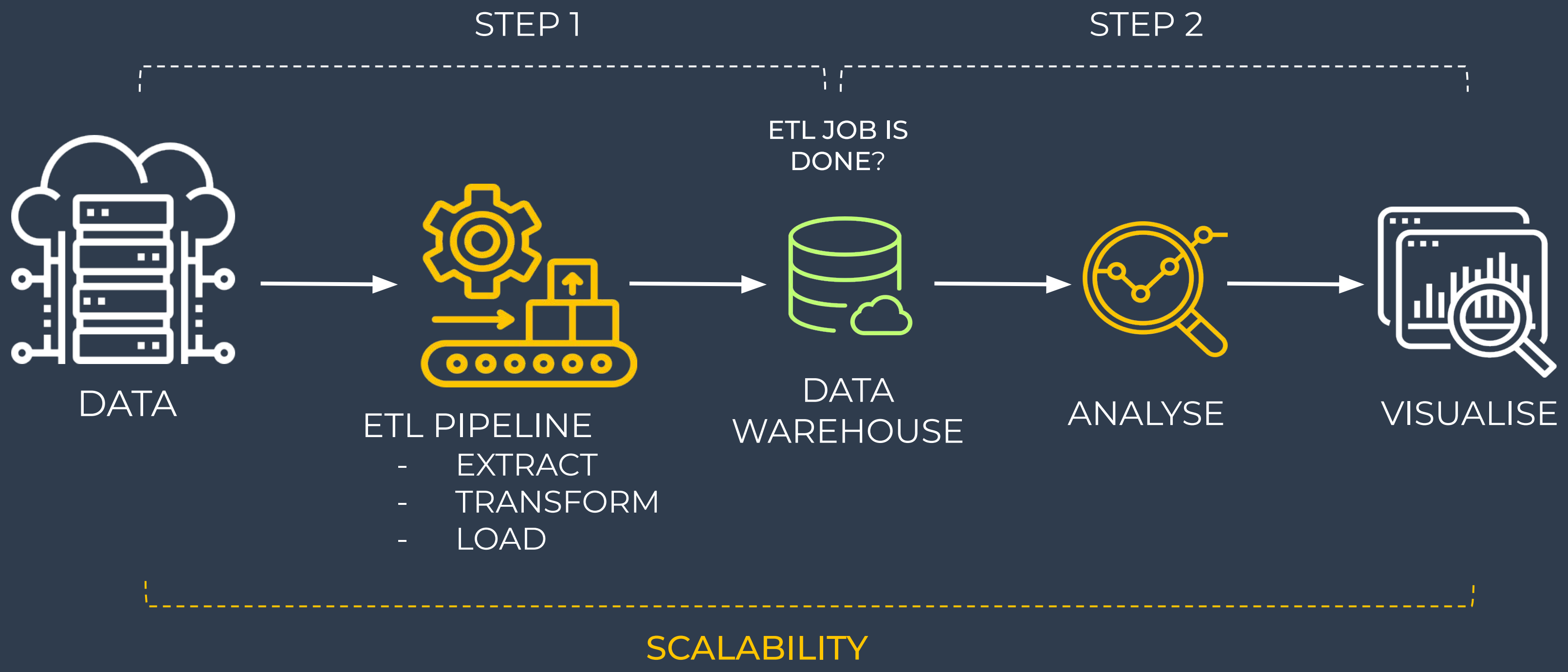
@matvi3nko

ORCHESTRATE MULTIPLE ETL JOBS USING AWS STEP FUNCTIONS AND AWS LAMBDA

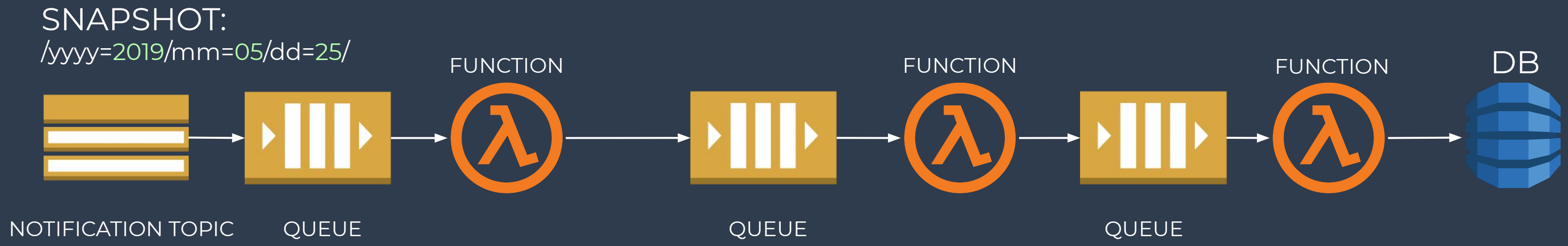


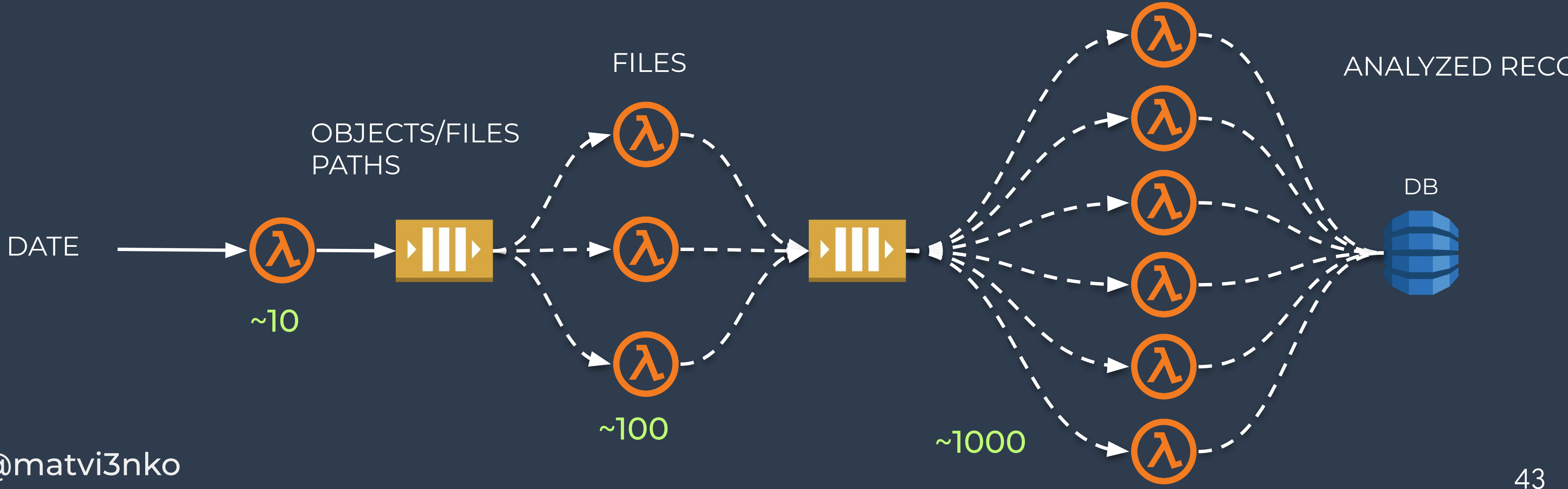
@matvi3nko

DATA ANALYSIS



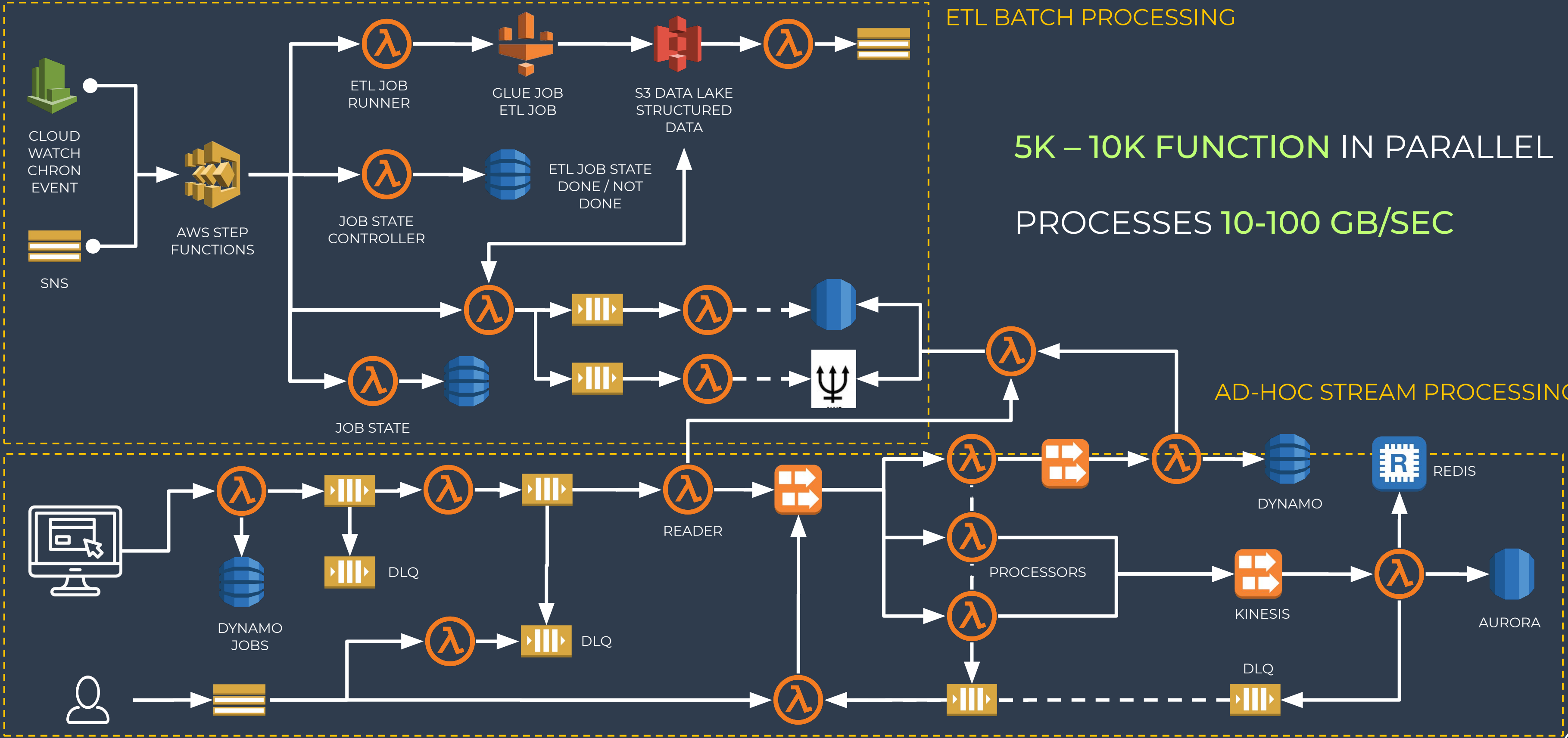
SCALABILITY





@matvi3nko

BLUEPRINT



5K – 10K FUNCTION IN PARALLEL

PROCESSES 10-100 GB/SEC

@matvi3nko

CONCLUSIONS

1. DESIGN MISTAKES = SOLUTION PRICE
2. IN SERVERLESS YOU CAN BUILD VERY BUILD A VERY FLEXIBLE ARCHITECTURE

   YOU CAN SWITCH FROM BATCH TO STREAM PROCESSING
3. YOU CAN RUN THOUSANDS OF LAMBDA FUNCTIONS IN PARALLEL

   AND PROCESS GBs OF DATA PER SEC

THANKS!

Nikolay Matvienko

Grid Dynamics
We are now in Belgrade too.

You can find me at twitter.com/**matvi3nko** github.com/**matvi3nko**