AWS re:Invent

Speed and Reliability at Any Scale – Combining SQS and DB Services

Jonathan Desrocher, Amazon Web Services

Colin Vipurs, Shazam Entertainment Ltd.

November 14, 2013

amazon web services

© 2013 Amazon.com, Inc. and its affiliates. All rights reserved. May not be copied, modified, or distributed in whole or in part without the express consent of Amazon.com, Inc.

AWS Messaging = Amazon SQS + Amazon SNS

Simplicity	• Loose coupling sets you free!
Reliability	• Availability • Durability
Scalability	• Throughput • Elasticity

Amazon SQS Core Features

- Designed to provide high durability
- Holds messages until you explicitly delete them
- Unlimited backlog up to 14 days
- Amazon CloudWatch metrics and alerts for queue depth, message rate, and more

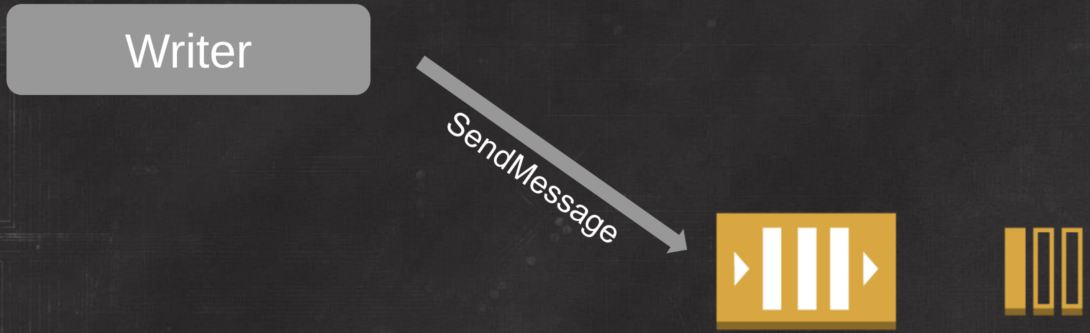
New and improved

- Payload size of up to 256KB
- Message batching for higher throughput and reduced costs
- Supports long polling for reduced costs and latency
- Cross-origin resource sharing support

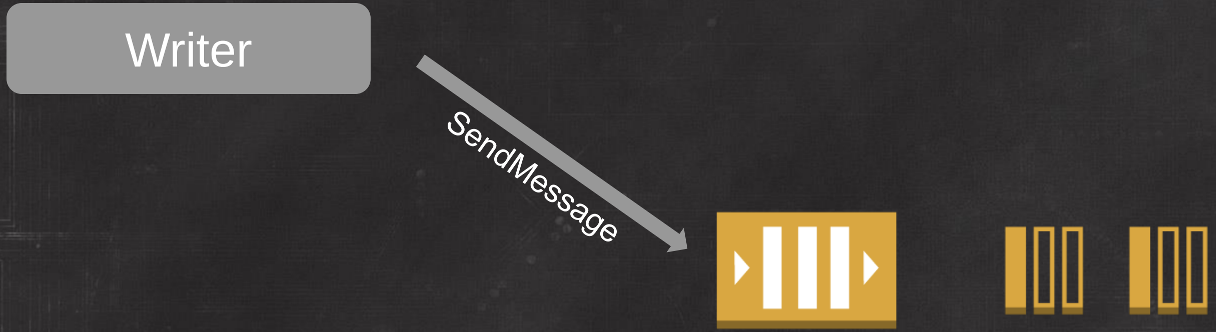
SQS Core mechanics

Basic Message Lifecycle

Basic Message Lifecycle



Basic Message Lifecycle

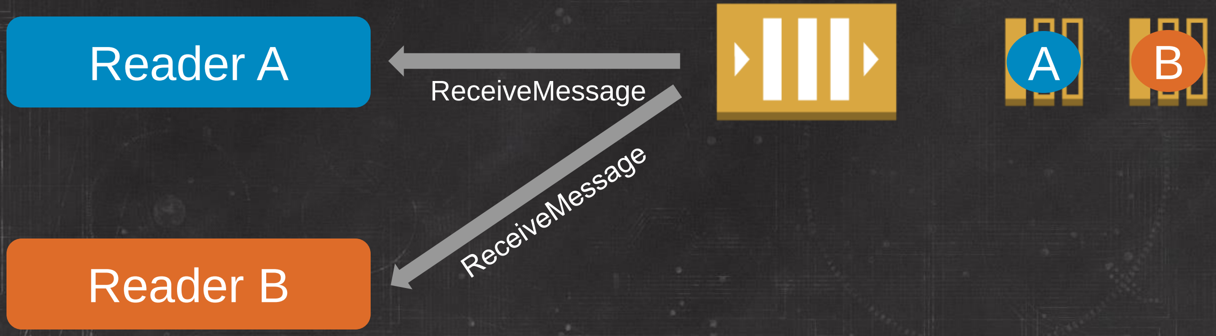


Basic Message Lifecycle

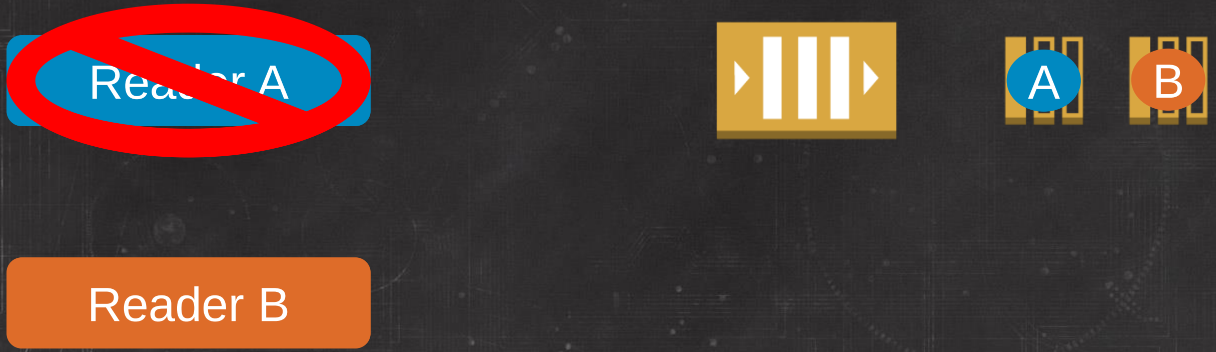
Reader A
Reader B

AWS re:Invent

Basic Message Lifecycle



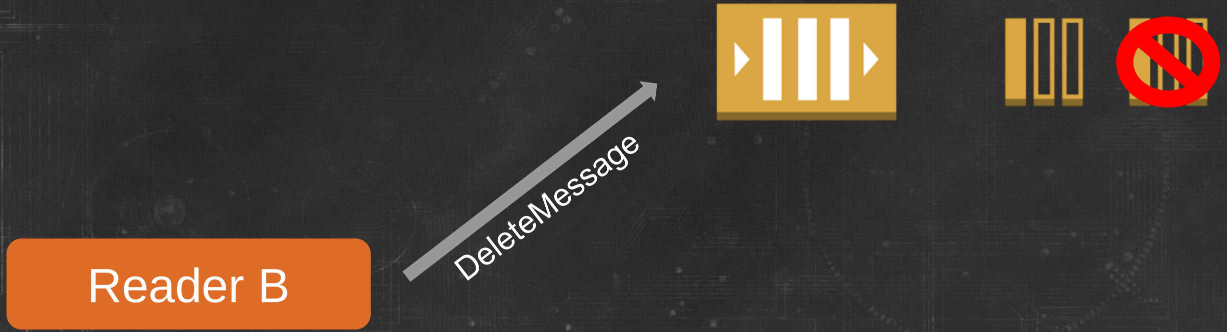
Basic Message Lifecycle



Basic Message Lifecycle

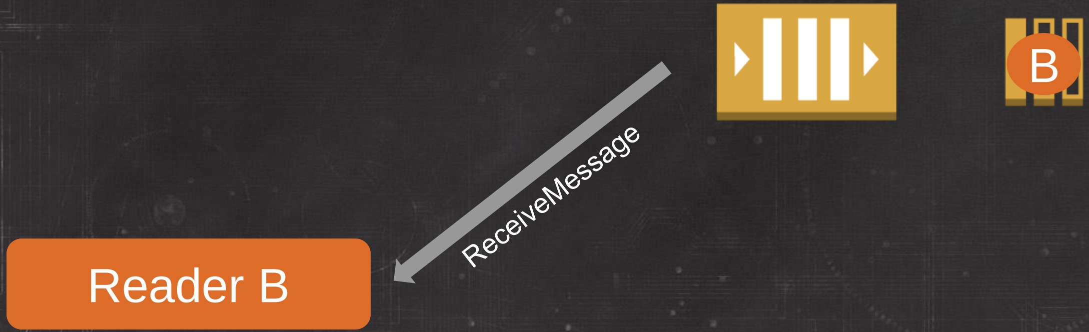
Reader B

Basic Message Lifecycle

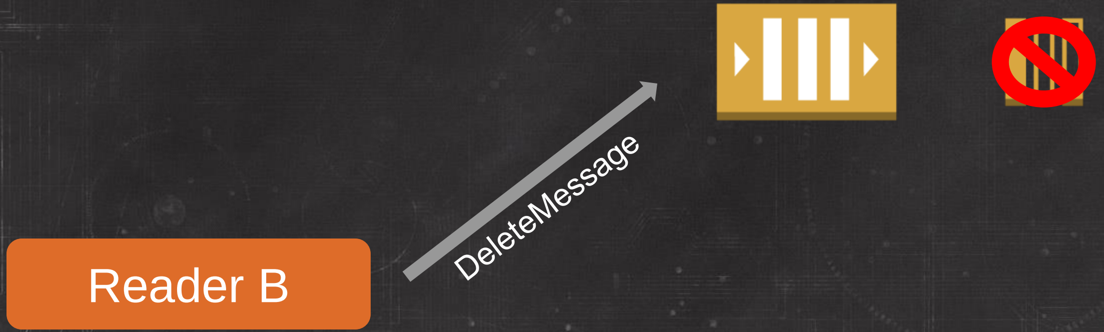


AWS re:Invent

Basic Message Lifecycle



Basic Message Lifecycle



AWS re:Invent

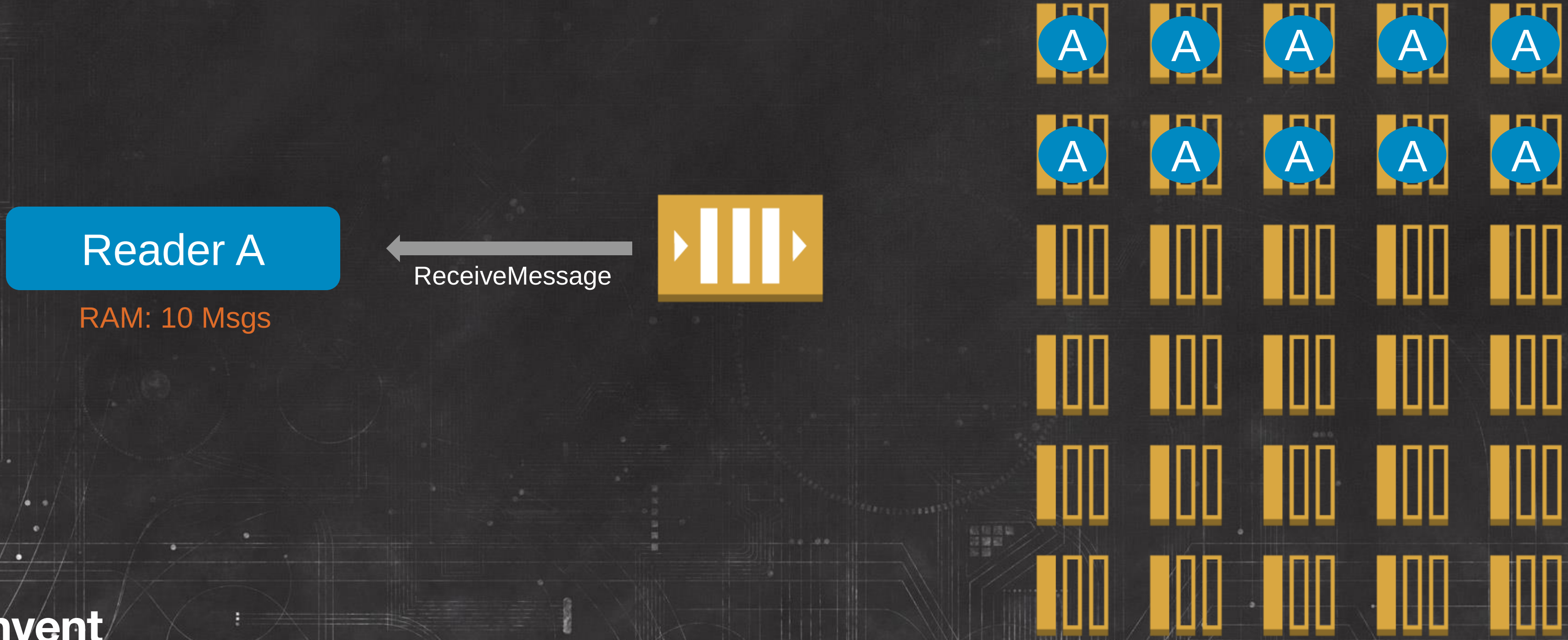
Basic Message Lifecycle

Reader B

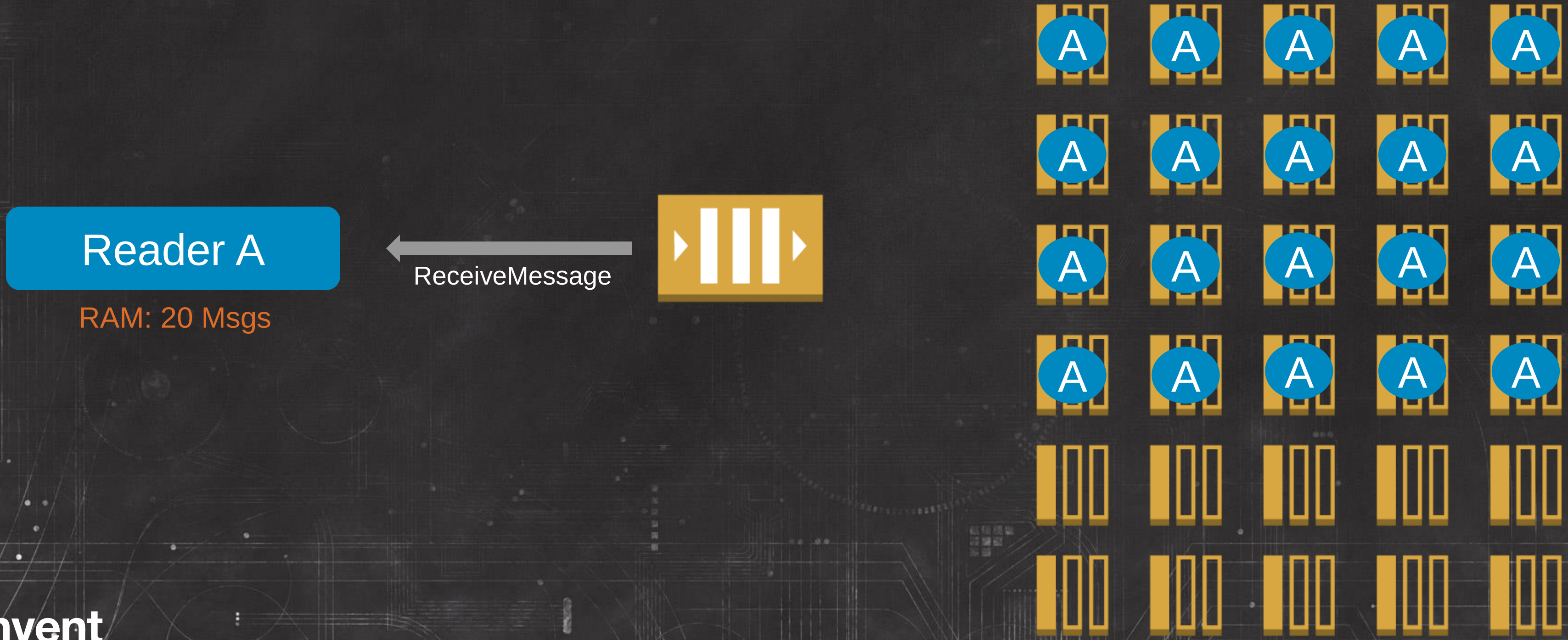
That covers reliability.
Now let’s go for the scale!

Bulk Transactional Reads

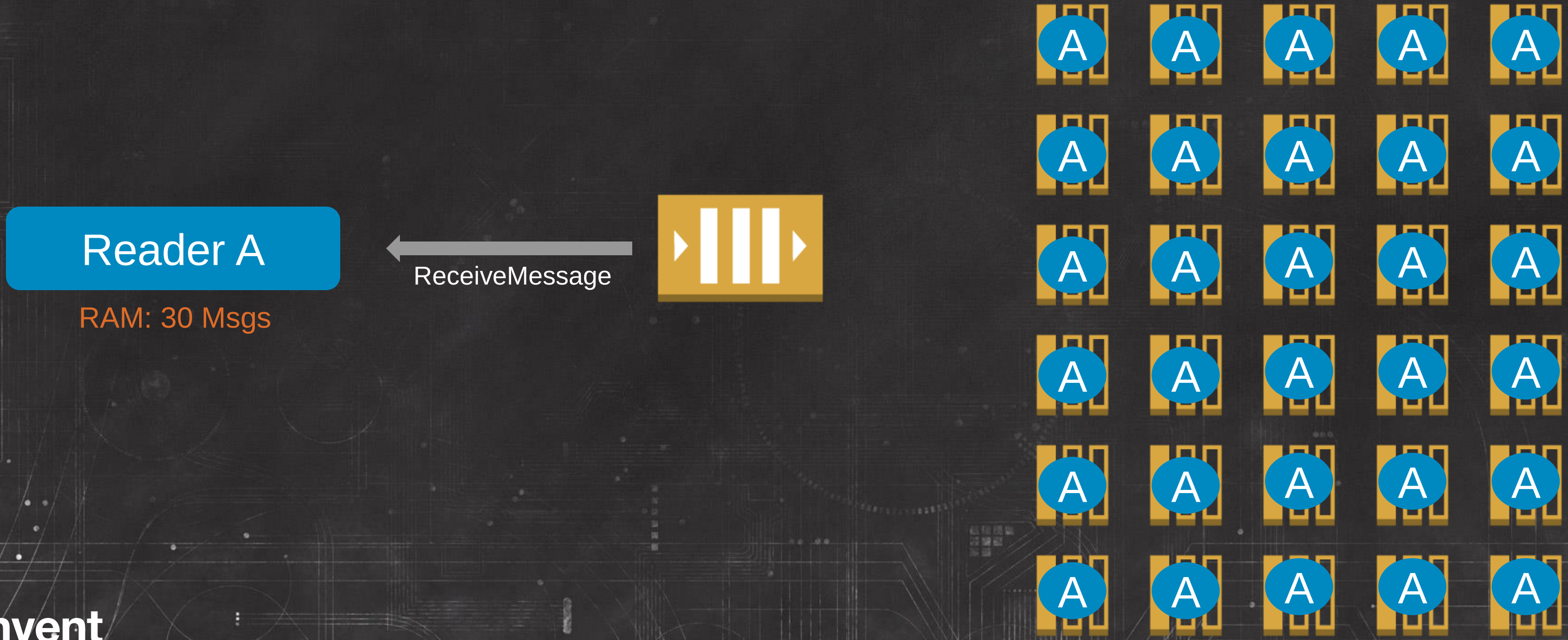
Bulk Transactional Reads



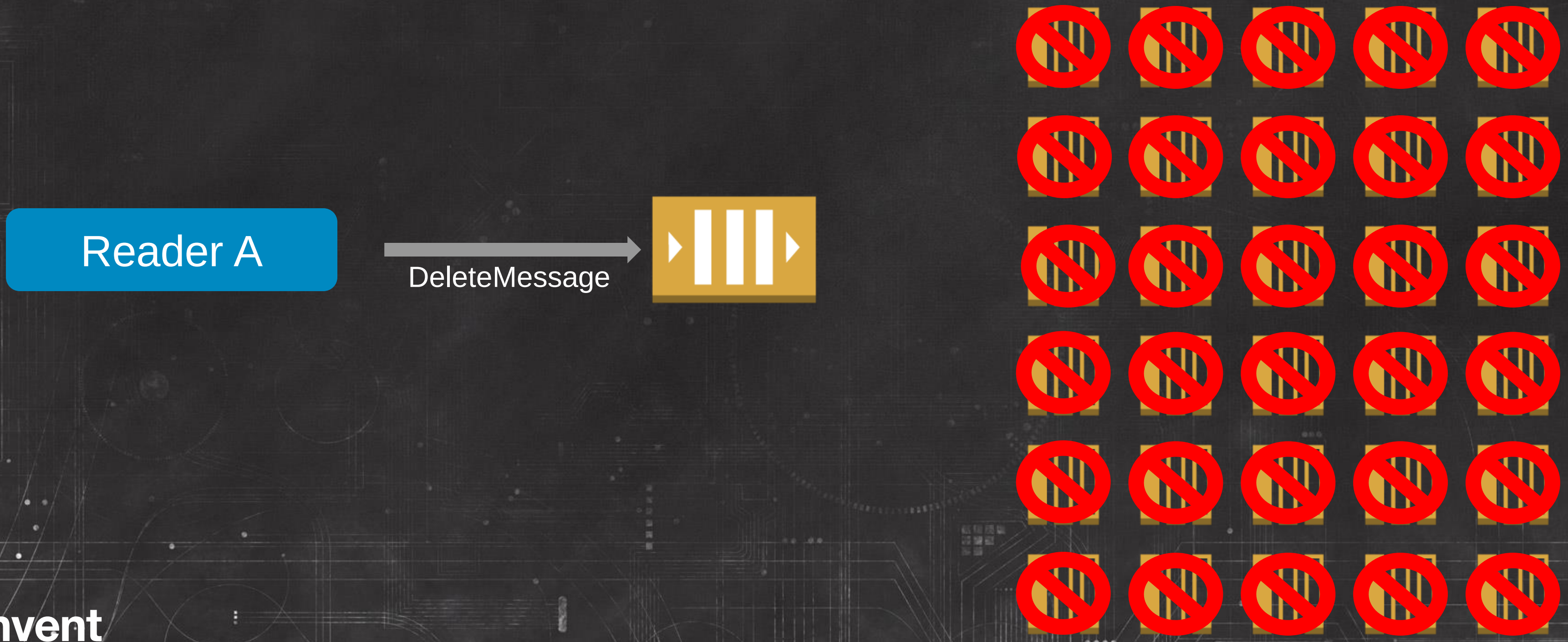
Bulk Transactional Reads



Bulk Transactional Reads



Bulk Transactional Reads

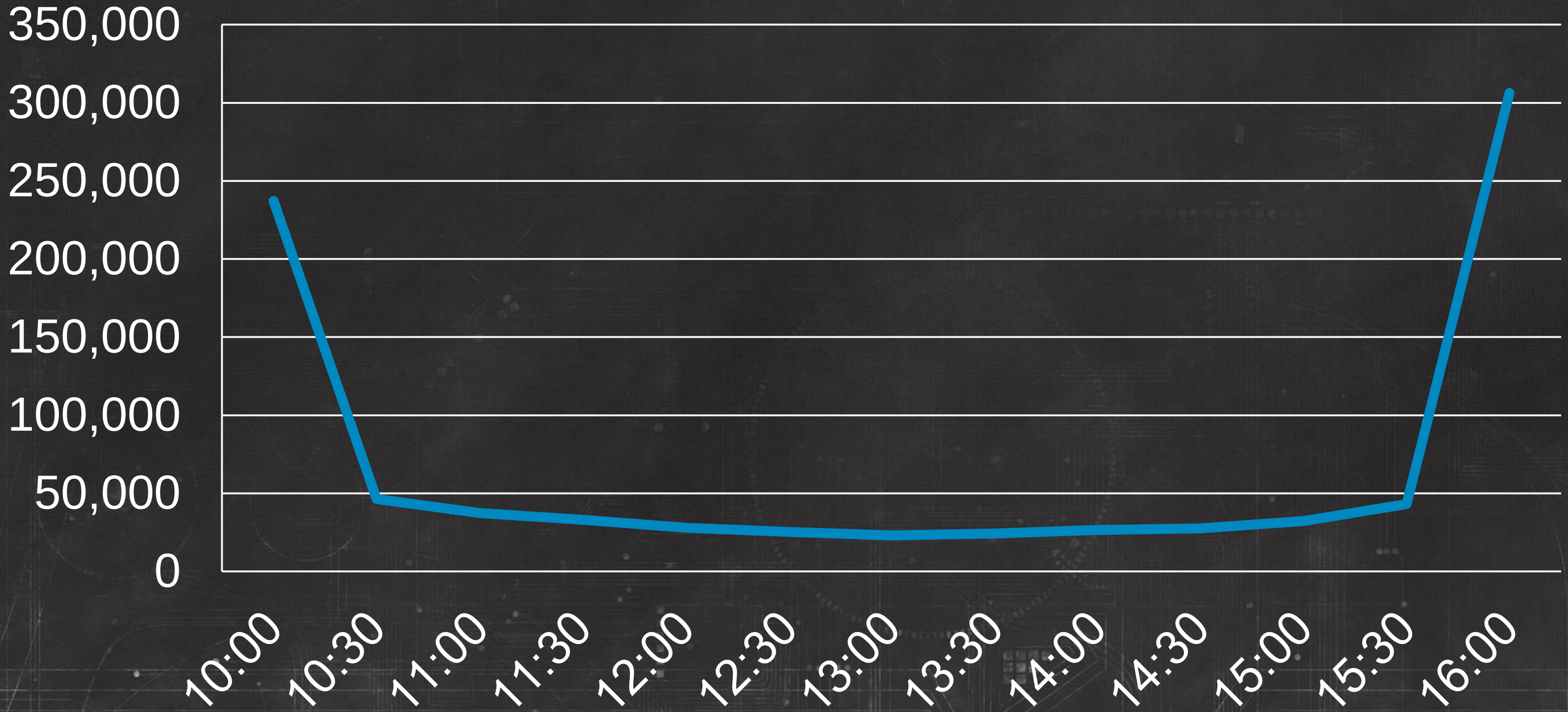


AWS re:Invent

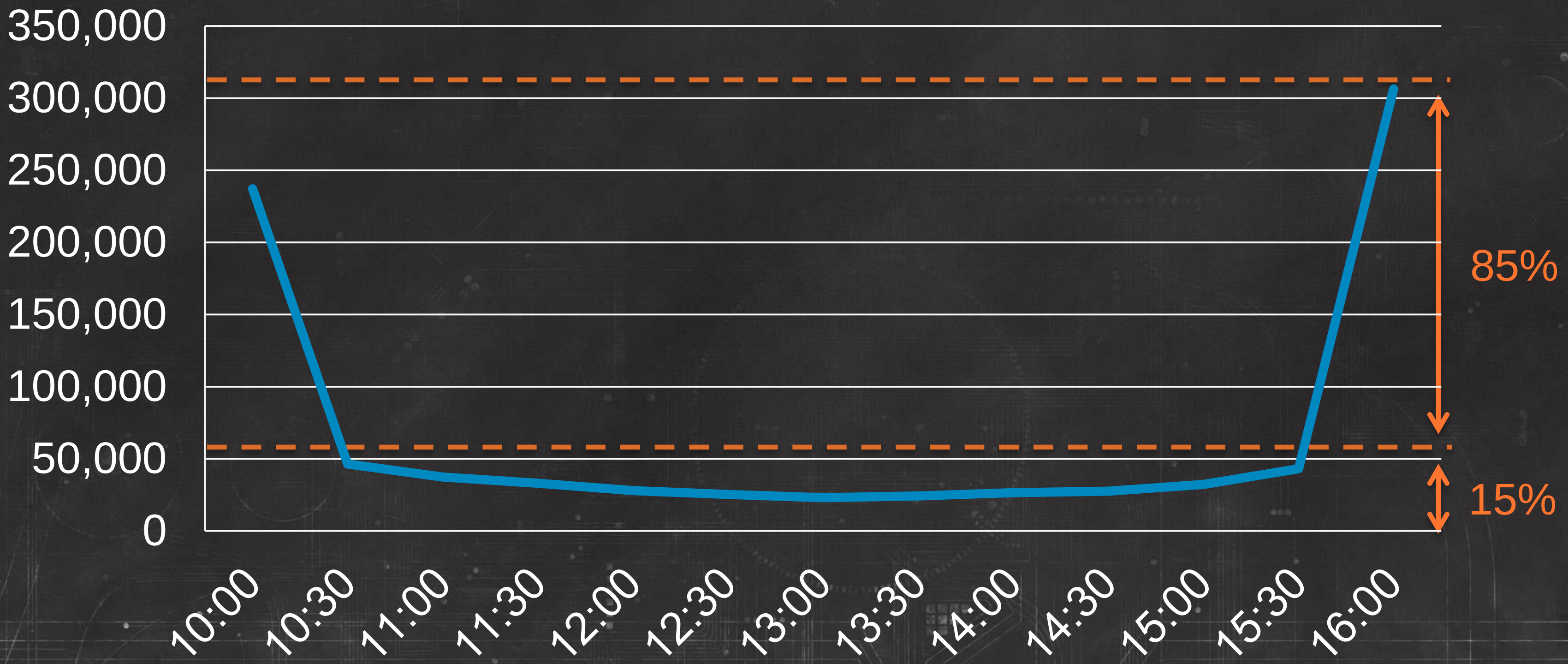
Bulk Transactional Reads

Let’s take it to real life!

Scalability example: market trade volume by half hour



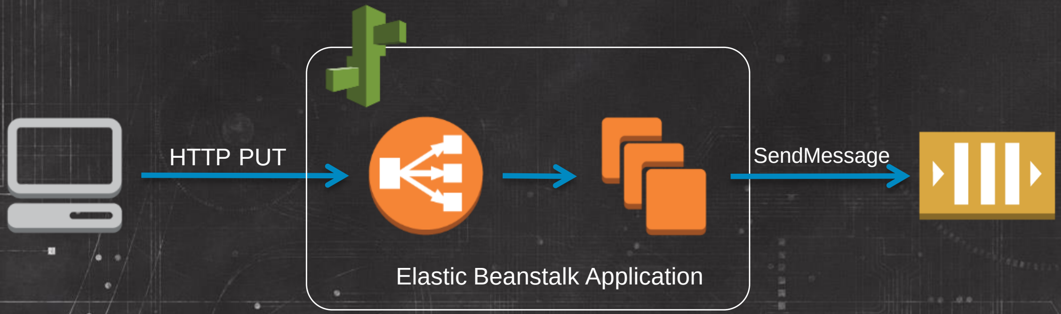
Scalability example: market trade volume by half hour



Design pattern #1:
Batch processing

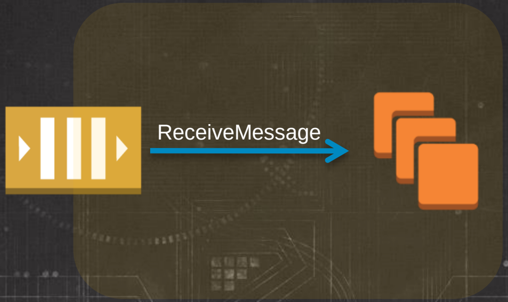
Batch Processing

- Use SQS as a scalable and resilient short-term storage solution.
- Simply configure the appropriate retention period and send away!



Batch Processing

- When appropriate, launch a fleet of Amazon EC2 workers and process the messages en masse.



Design pattern #2:
IAM Roles for Amazon EC2

Using IAM Roles for Amazon EC2

- Create an IAM role with the appropriate permissions to Amazon SQS.
- Launch EC2 instances with this role.
- Done!
  - Audit logs will correlate the EC2 instance ID to the SQS API calls.
  - IAM will automatically rotate the credentials on our behalf.

```
{
  "Statement": [
    {
      "Sid": "Stmt1384277213171",
      "Action": [
        "sqs:ChangeMessageVisibility",
        "sqs:DeleteMessage",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "sqs:ListQueues",
        "sqs:ReceiveMessage"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-
1:455320512810:Sensor_Ingestion"
    }
  ]
}
```

Using IAM Roles for Amazon EC2

- Use the AWS SDK on the Instance
- No need to type credentials
  - Not in code
  - Not in a configuration file
  - Not via the console either

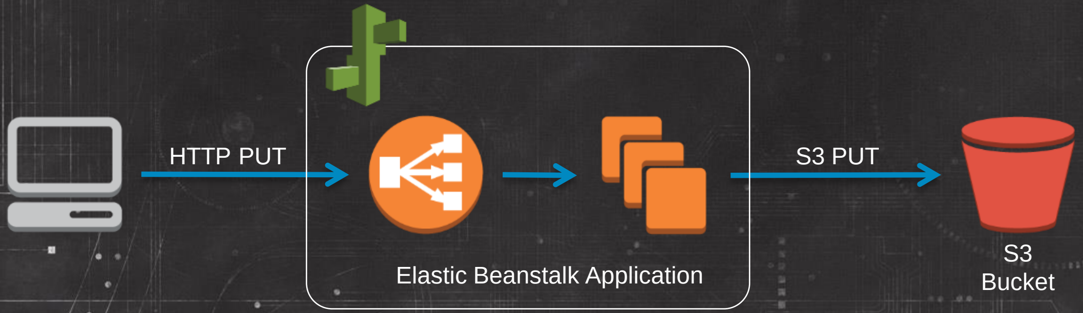
```
require ‘rubygems’
require ‘aws-sdk’

sqs = AWS::SQS.new()
myqueue = sqs.queues.named("Sensor_Ingestion")

myqueue.poll do |msg|
  # Do something with the message
end
```

Design pattern #3:
Using SQS to durably batch writes

AWS re:Invent

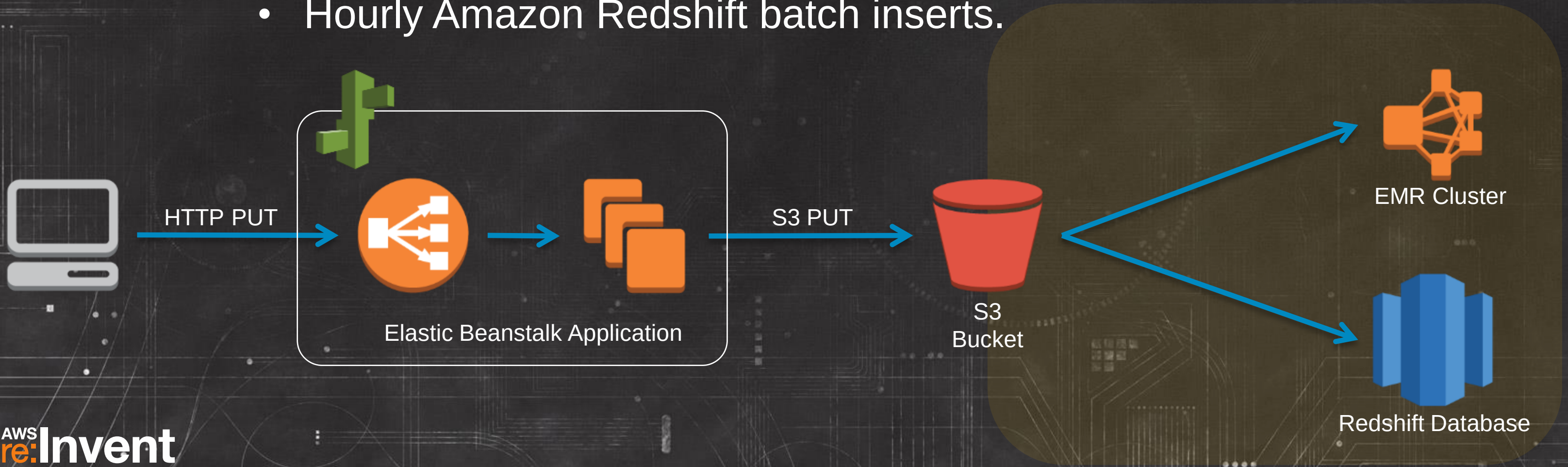
Using SQS to durably batch writes

- The application:
  - An AWS Elastic Beanstalk application.
  - Clients upload data to the application through HTTP PUTs.
  - Each upload is 100KB in size.
  - Amazon S3 will be used as the permanent data store.



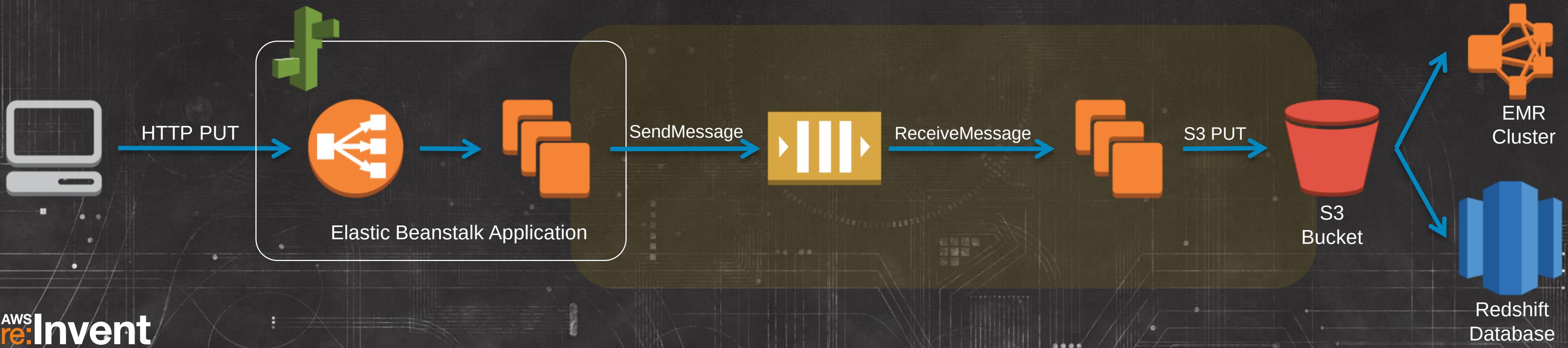
Using SQS to durably batch writes

- The challenge:
  - We have an external constraint that requires us to batch the upload into Amazon S3.
    For example:
    - Amazon EMR best practices call for Amazon S3 object size of >10MB.
    - Hourly Amazon Redshift batch inserts.



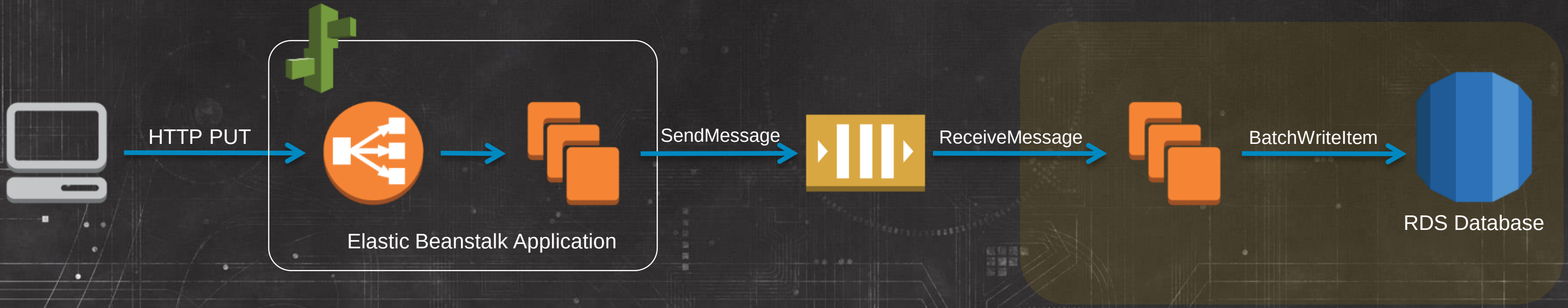
Using SQS to durably batch writes

- Enter SQS:
  - Persist individual client PUTs as SQS messages.
  - Have an Amazon EC2 worker role that performs the following logic:
    - Receive SQS message and add to an in-memory buffer.
    - Once buffer is full, upload to Amazon S3.
    - Upon acknowledgement from S3, delete SQS messages from queue.



Using SQS to durably batch writes

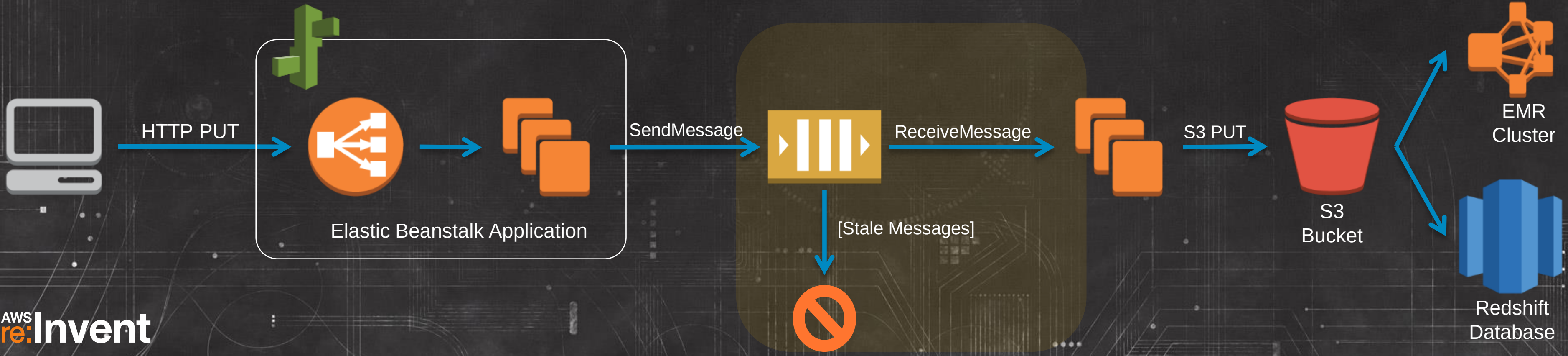
- Also to consider:
  - Some data stores are optimized for read workloads
  - Buffering the writes with Simple Queue Service will ensure both speed and reliability of data ingestion.



Design pattern #4:
Discarding stale messages

Discarding stale messages

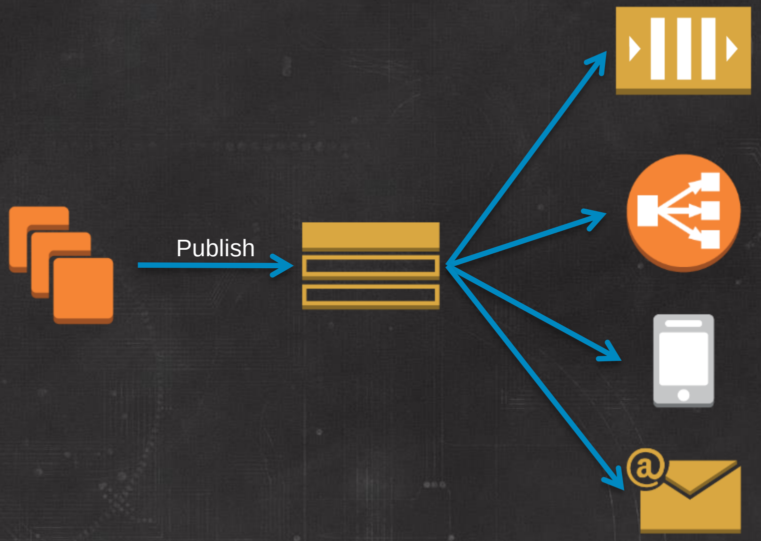
- Controlled via the MessageRetentionPeriod property.
- Useful when there is no business value for data older than X minutes.
  - *"Transactions that don't complete within 5 minutes are abandoned, enabling client-side failure handling".*



Design pattern #5:
Simple Notification Service Fan-out

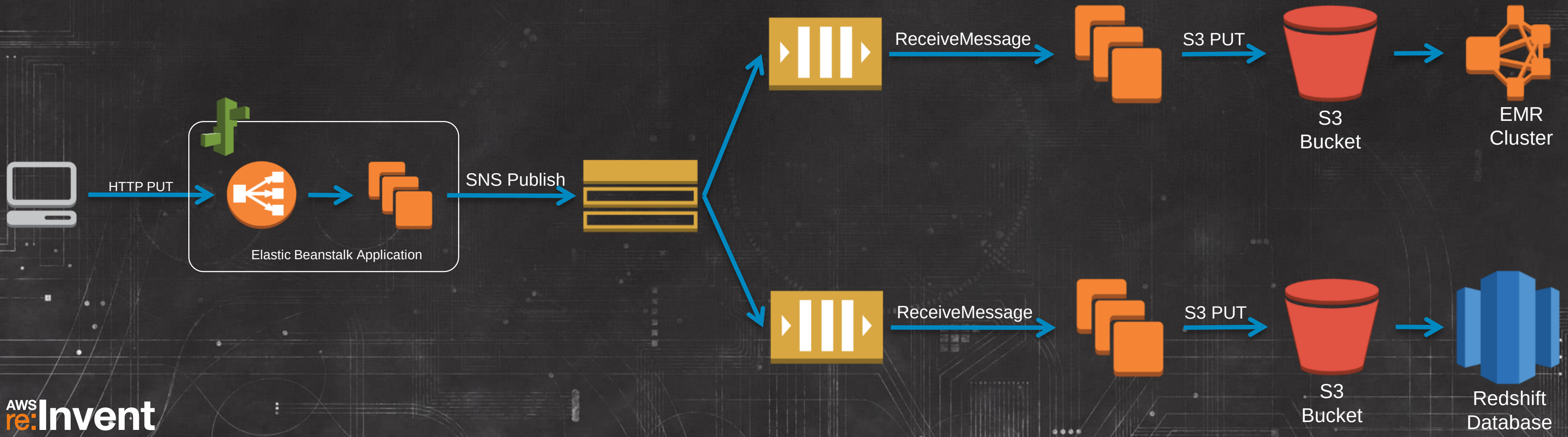
Simple Notification Service Fan-out

- Atomically distribute a message to multiple subscribers over different transport methods
  - SQS queues
  - HTTP/S endpoints
  - SMS
  - Email
- Also used to abstract different mobile device push providers (MBL308)
  - Apple Push Notification Service
  - Google Cloud Messaging for Android
  - Amazon Device Messaging



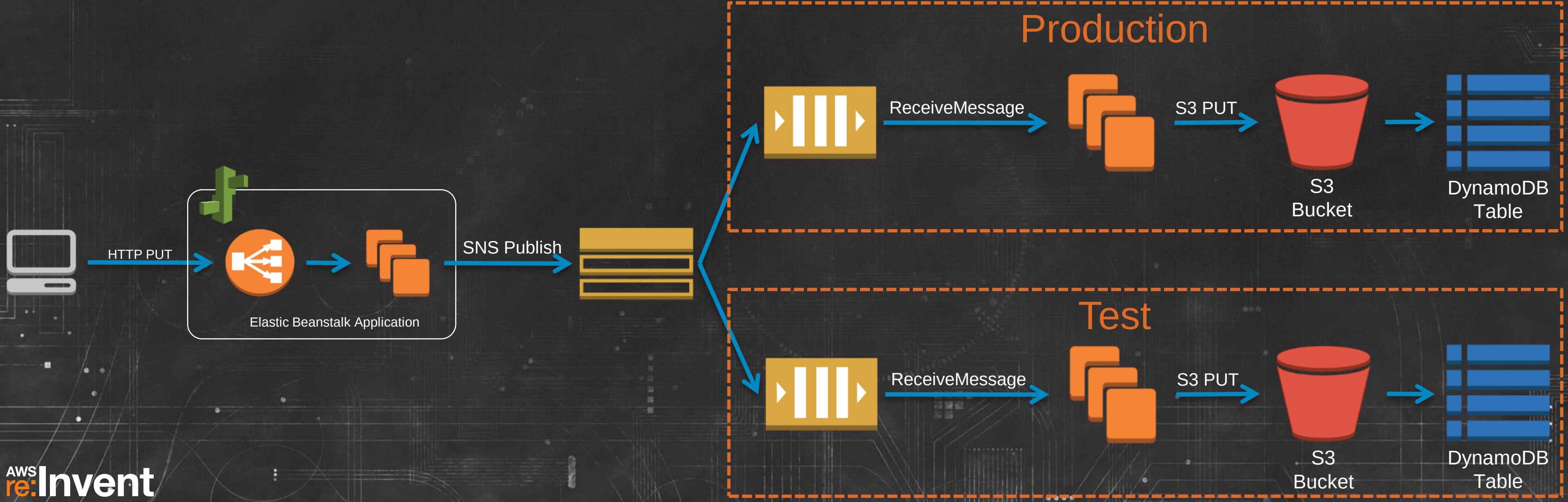
Simple Notification Service Fan-out

- Perform different operations on the same data
  - Split different facets of the payload into different systems.
  - Duplicate the data into short-term and longterm storage systems.



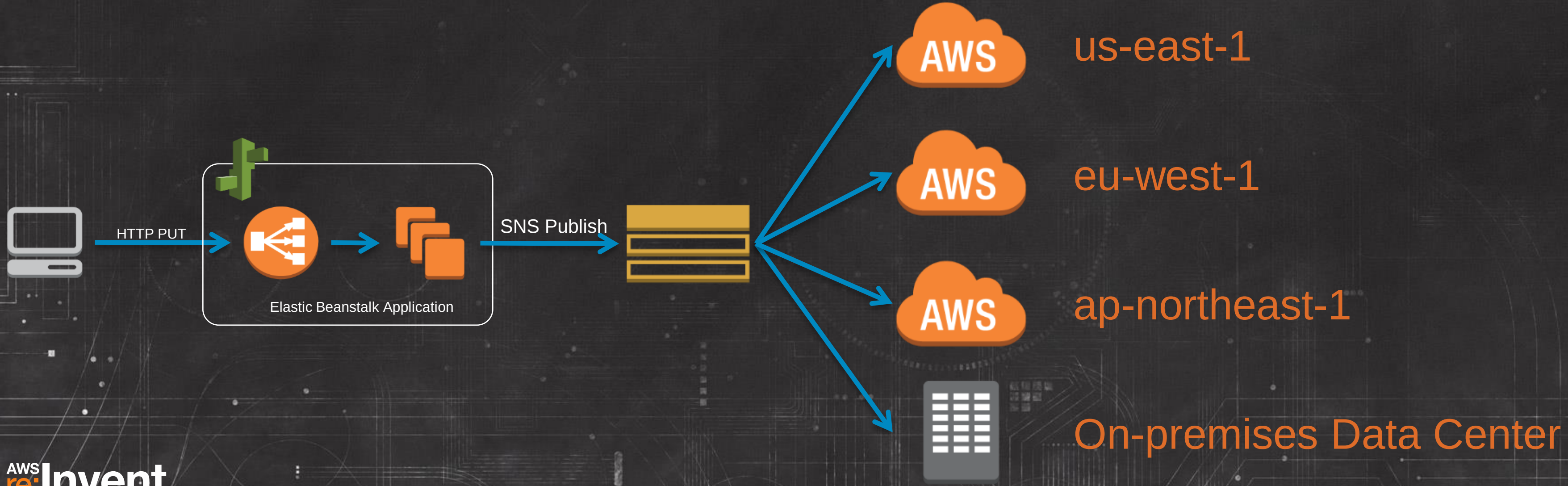
Simple Notification Service Fan-out

- Deliver the same data to different environments



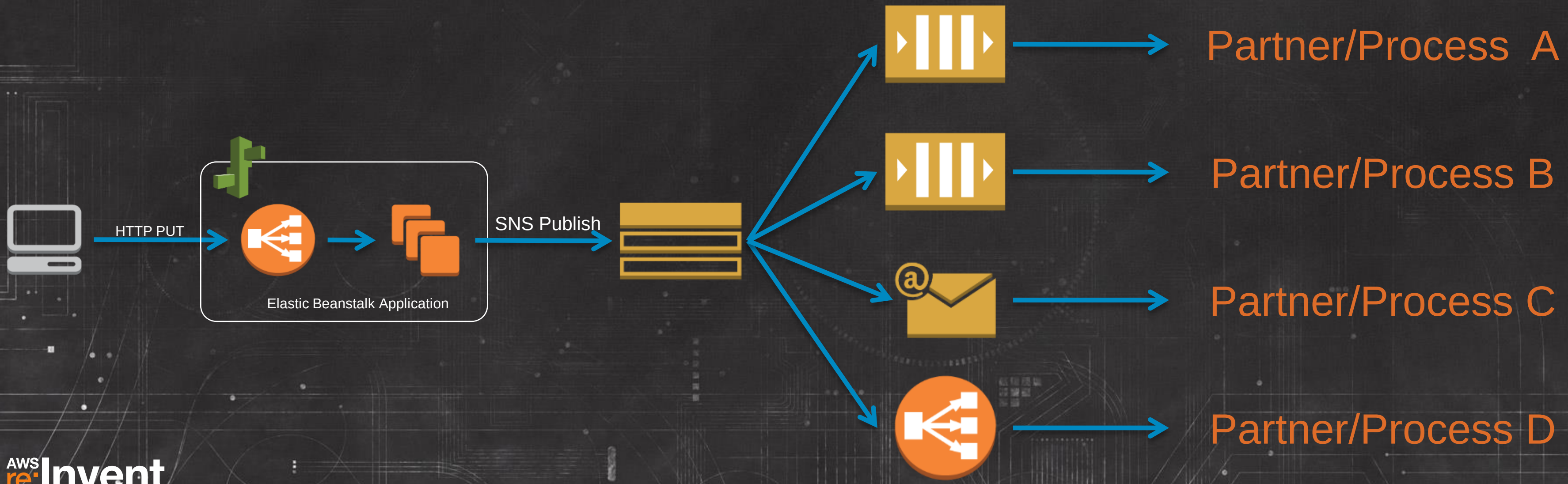
Simple Notification Service Fan-out

- Distribute the same data to a multiple external environments:
  – Push data to different locations worldwide.
  – Seamlessly synchronize AWS and on-premises environments.
  – Pro tip: MessageID field is consistent across locations.



Simple Notification Service Fan-out

- Each recipient can have its own preferred transport protocol:
  - SQS for guaranteed delivery
  - Email for human-friendly delivery
  - HTTP/S for real-time push



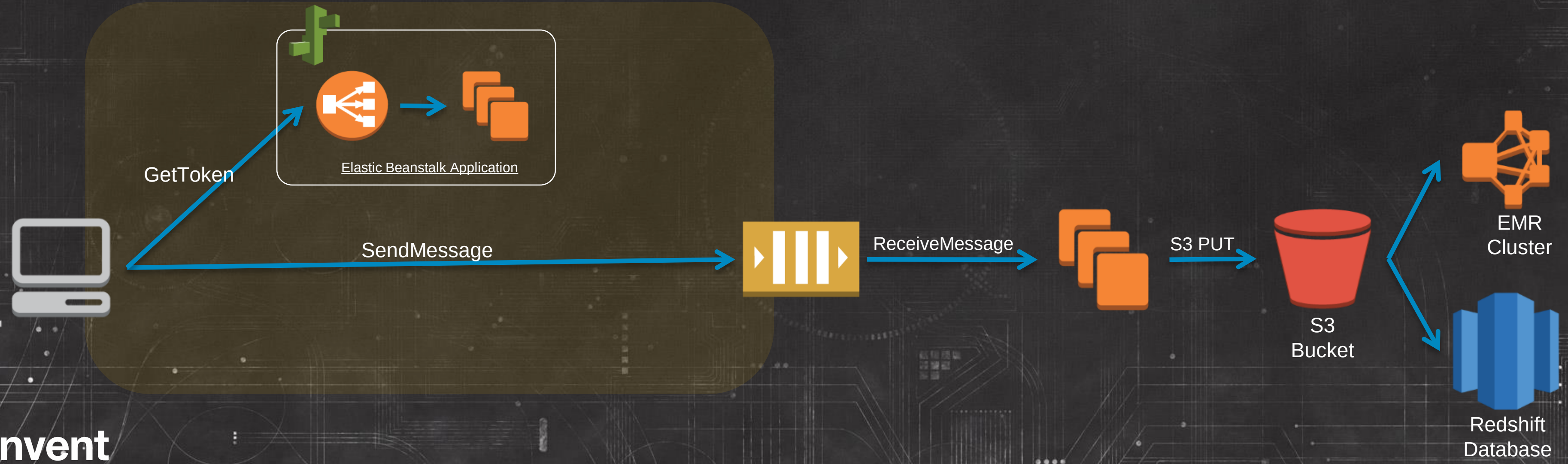
Design pattern #6:
Send messages from the browser

Send messages from the browser

- Make direct calls to AWS services such as SQS and DynamoDB directly from the user’s browser.
- Authentication is based on STS tokens.
- Supports S3, SQS, SNS and DynamoDB.

Send messages from the browser

- Back to our sample architecture:
  - Browser authenticates against Elastic Beanstalk application
  - Response includes location of SQS Queue and STS Token for direct authentication.



Colin Vipurs

Shazam Entertainment Ltd.

How Shazam works

YOU COULD WIN A CAMRY

NO PURCHASE NECESSARY. Open to legal residents of 50 U.S. & DC, 18+ [illegible]
Ends 2/5/12 at 10pm ET. To enter online & for full Official Rules, visit www.toyota.com/camryeffect
Data charges may apply. Void where prohibited. Sponsor: Toyota Motor Sales, U.S.A., Inc.

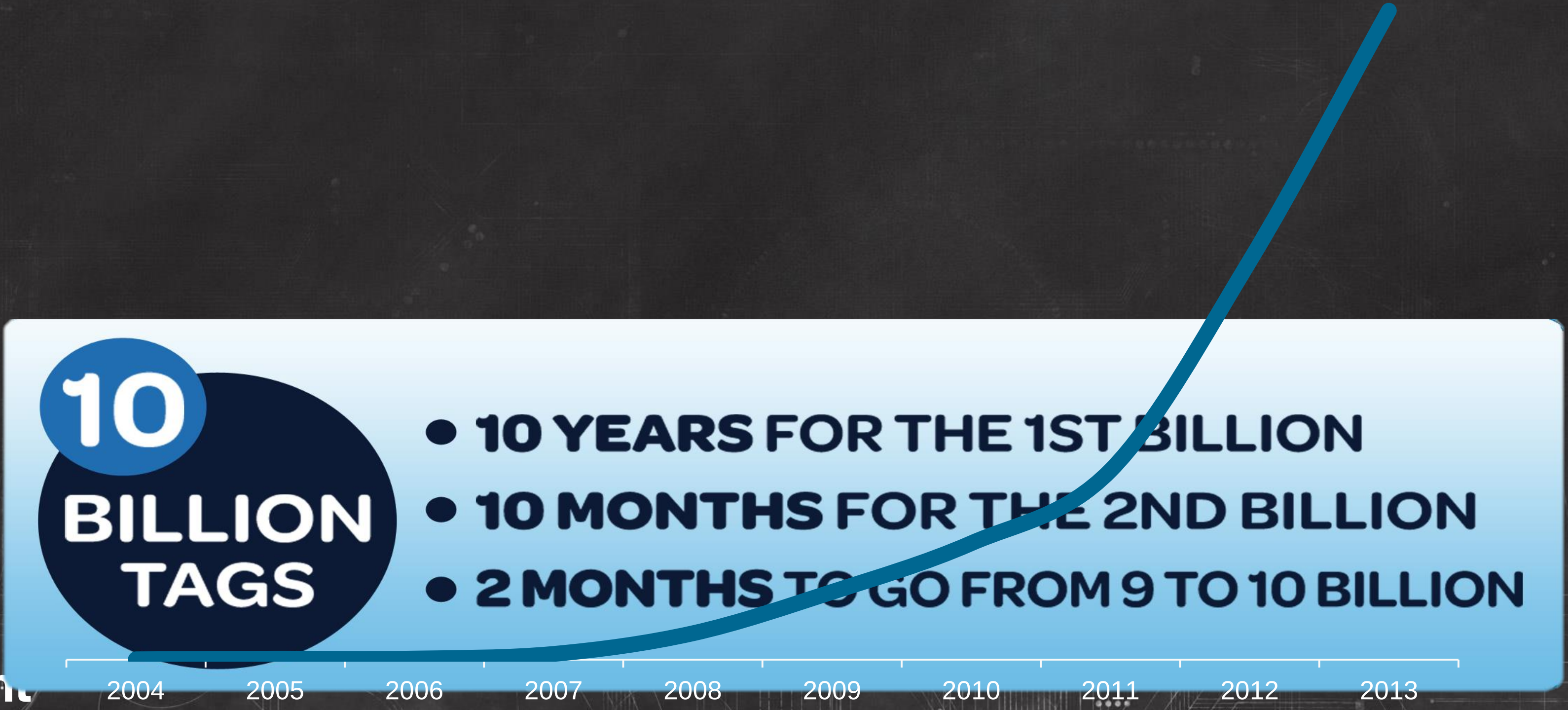
375 MILLION USERS

75 MILLION MONTHLY ACTIVE USERS

10 MILLION NEW USERS PER MONTH

10 BILLION TAGS

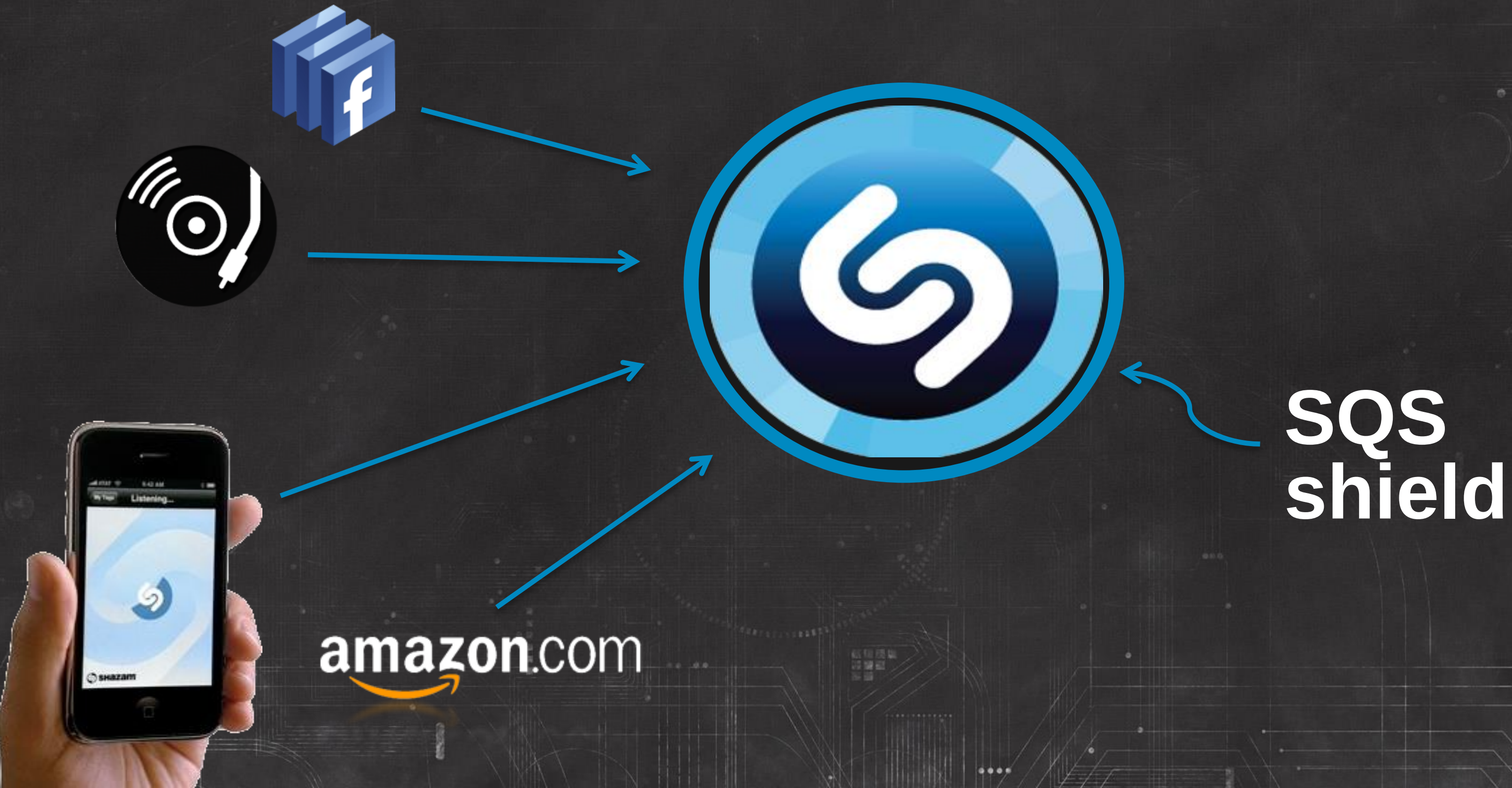
- 10 YEARS FOR THE 1ST BILLION
- 10 MONTHS FOR THE 2ND BILLION
- 2 MONTHS TO GO FROM 9 TO 10 BILLION



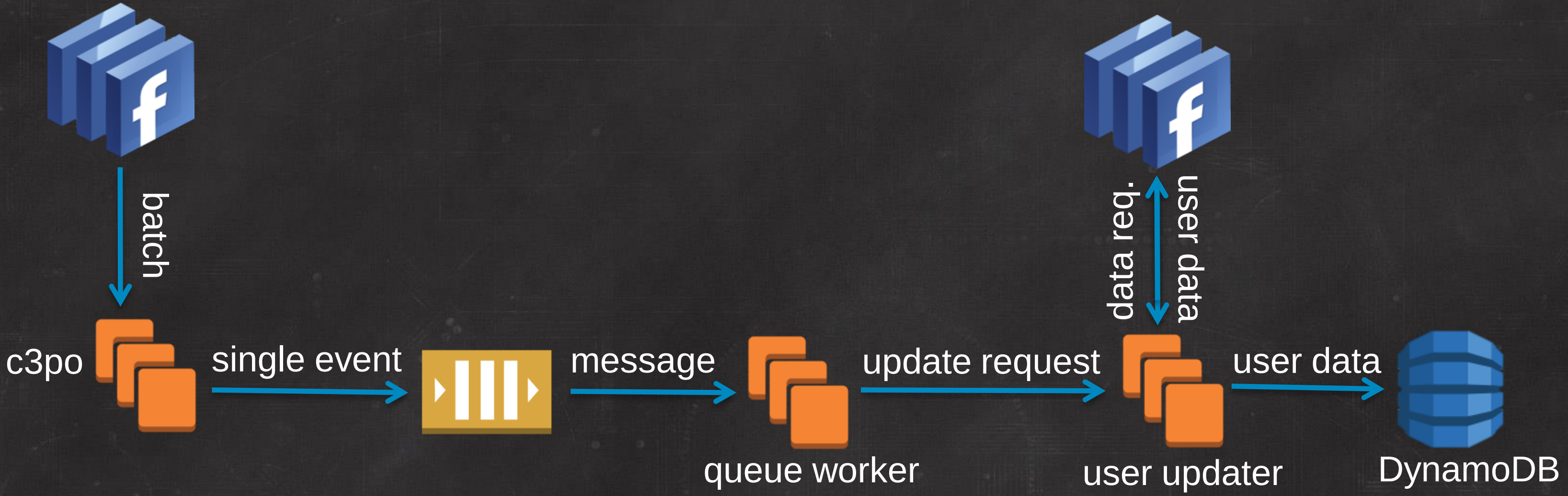
10 BILLION TAGS

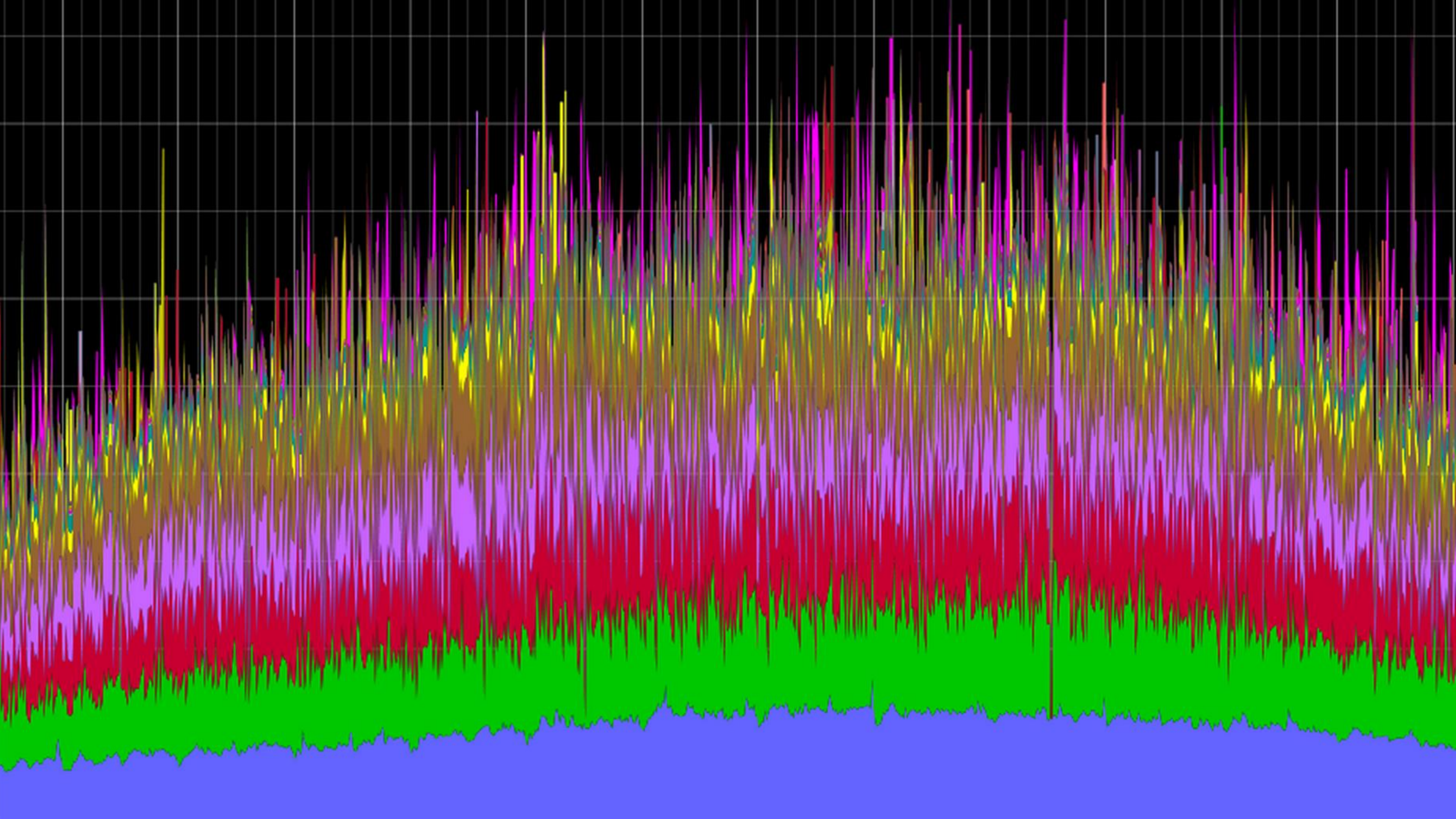
- 10 YEARS FOR THE 1ST BILLION
- 10 MONTHS FOR THE 2ND BILLION
- 2 MONTHS TO GO FROM 9 TO 10 BILLION

Amazon SQS for surge protection

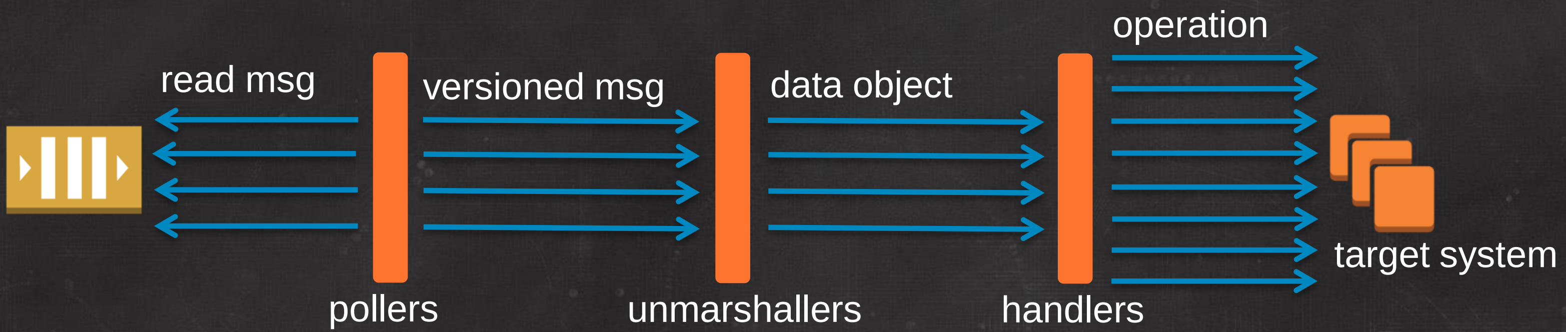


Facebook Realtime Updates

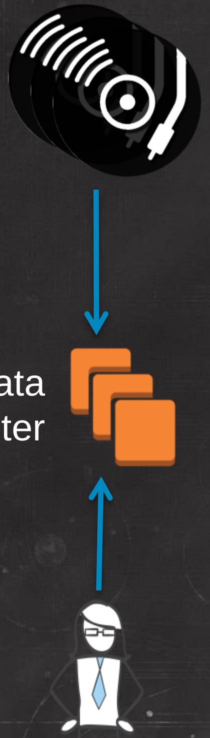




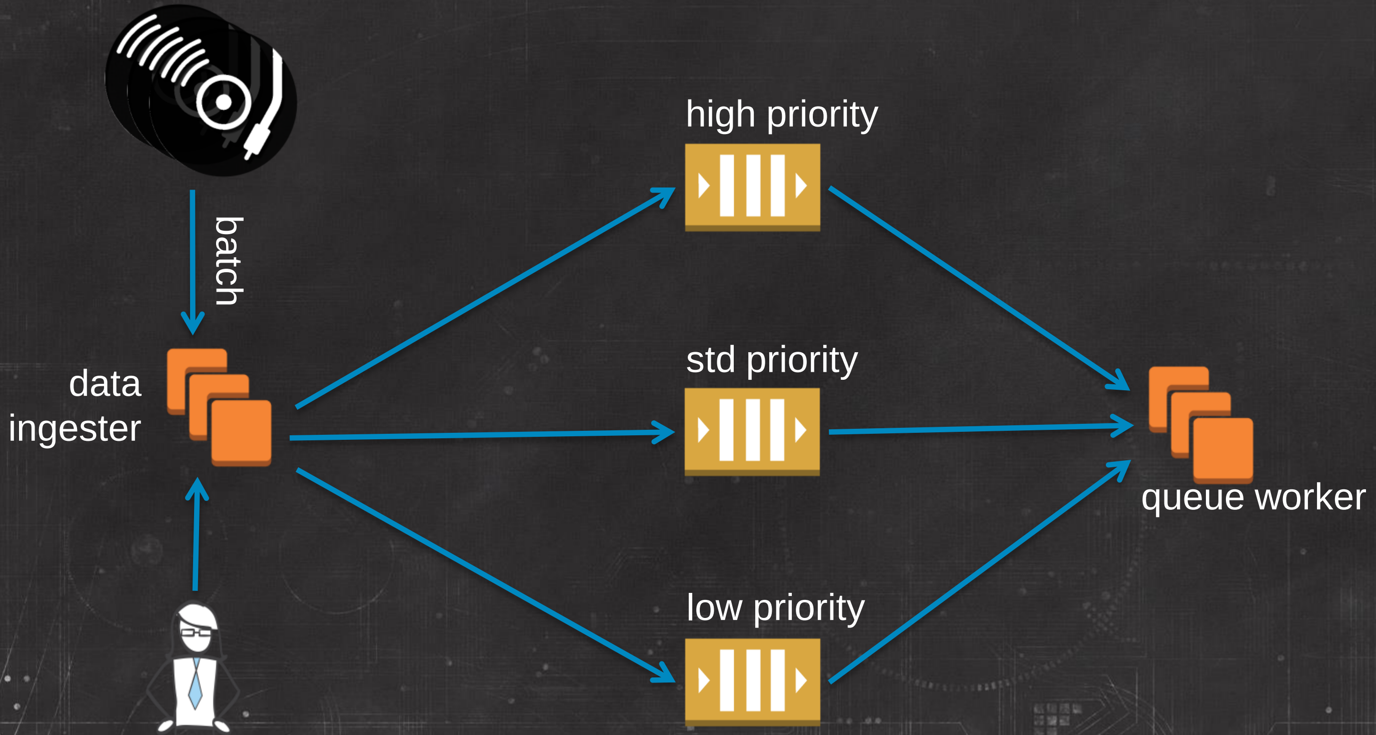
Queue Worker Anatomy



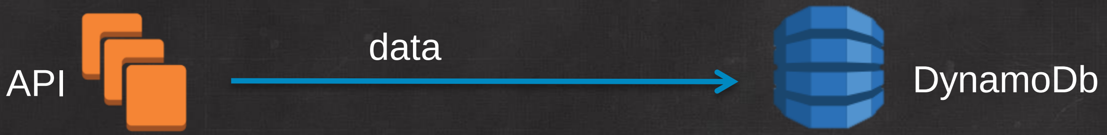
SQS for SLAs

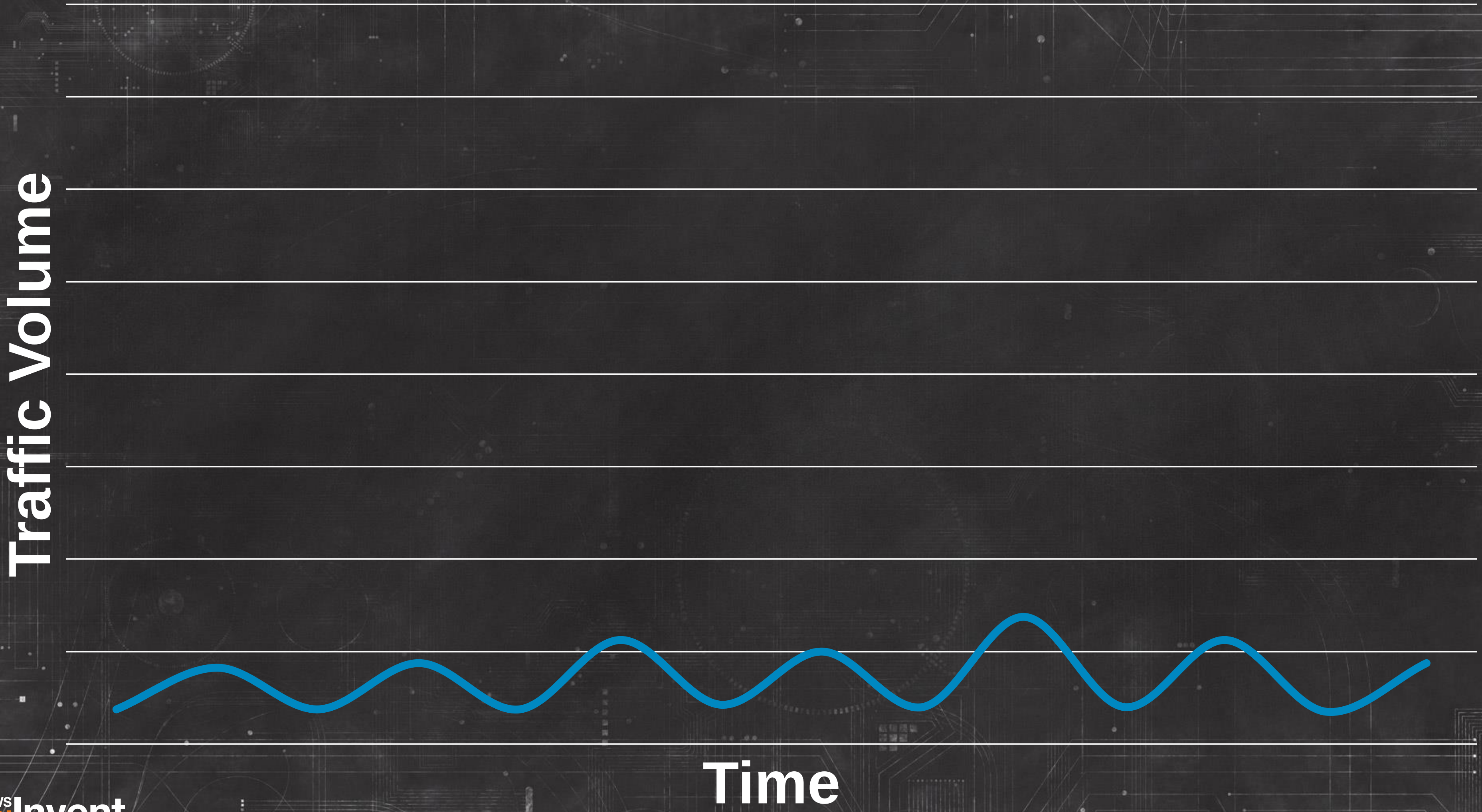


SQS for SLAs

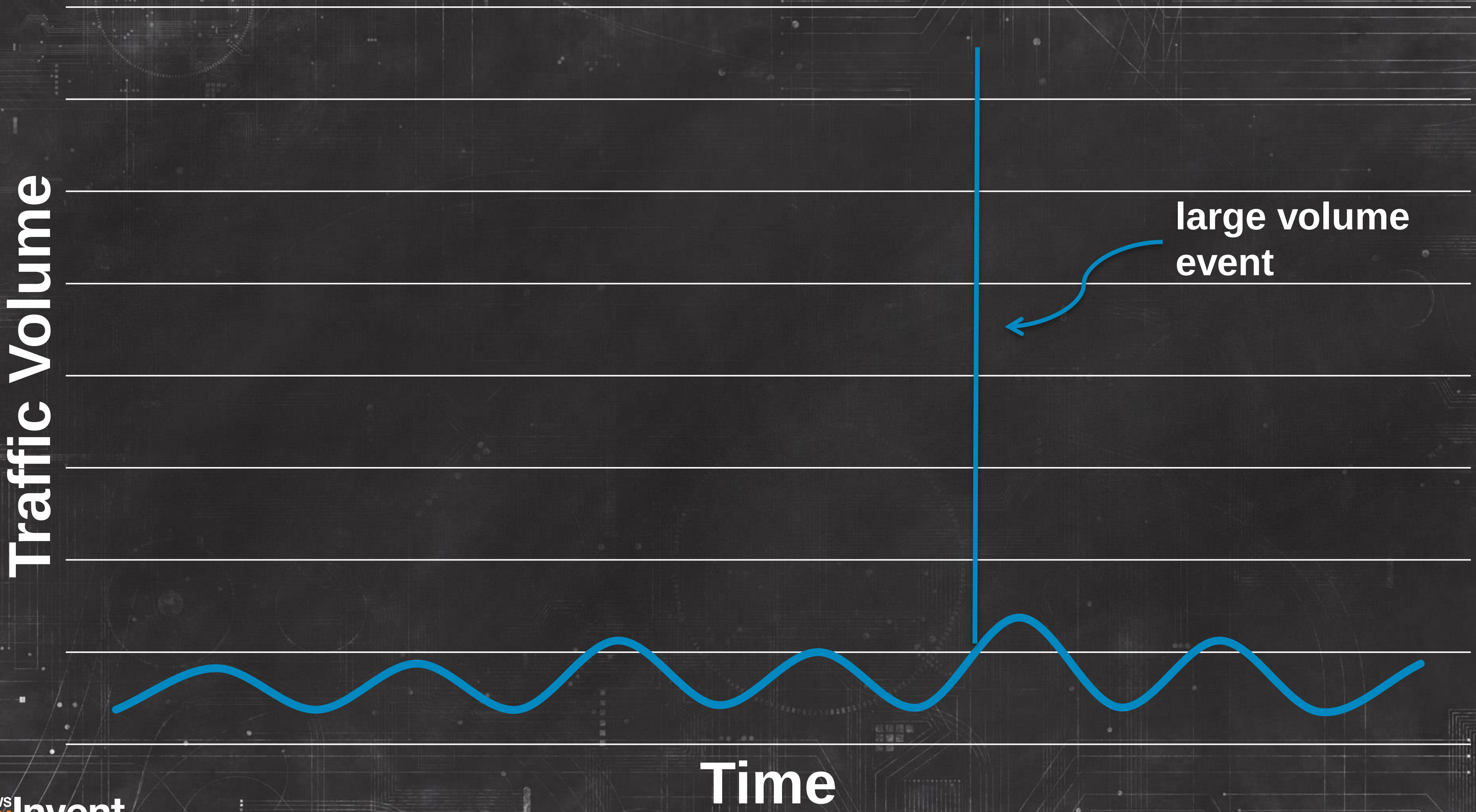

batch
data ingester
high priority
std priority
low priority
queue worker

SQS as DynamoDB Buffer



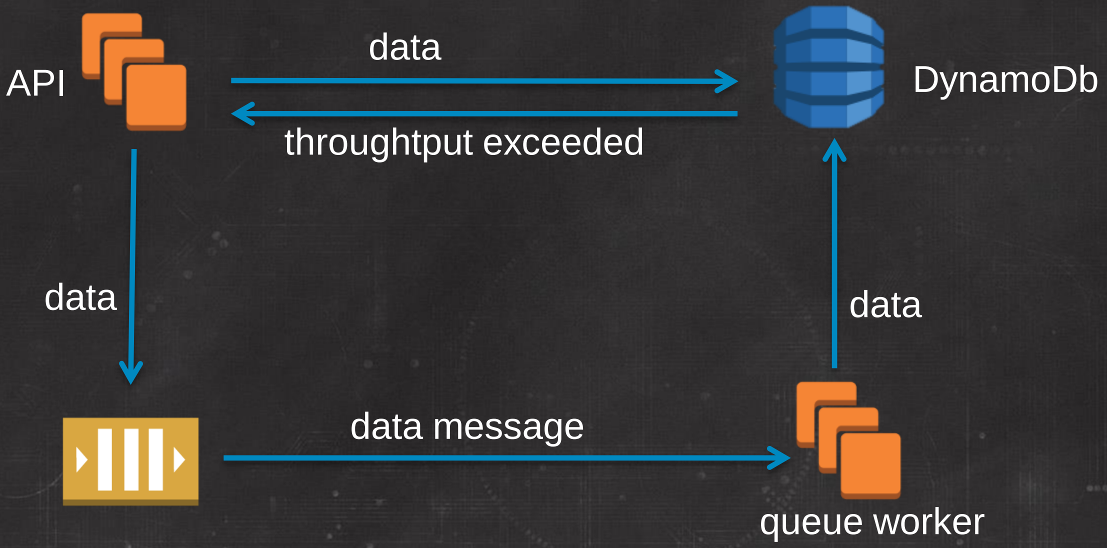


AWS re:Invent

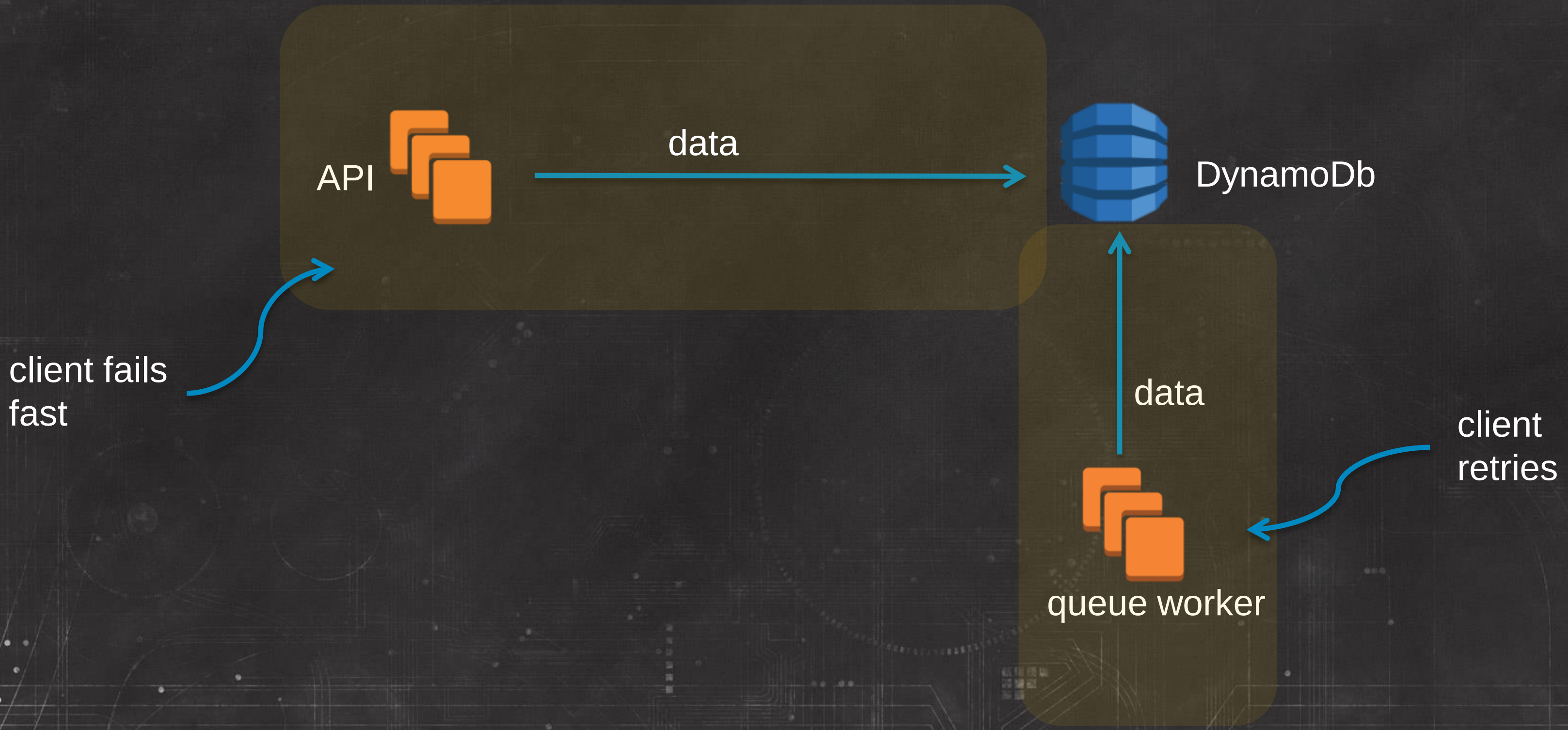


AWS re:Invent

SQS as DynamoDB Buffer



SQS as DynamoDB Buffer

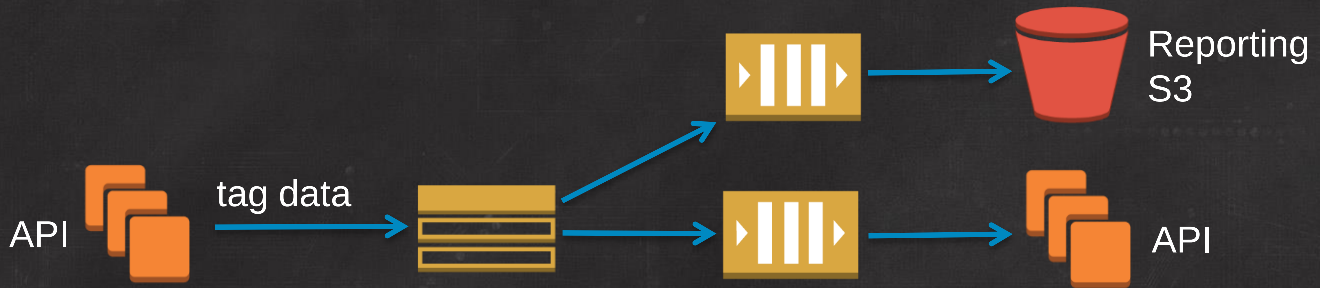


SQS as DynamoDB Buffer

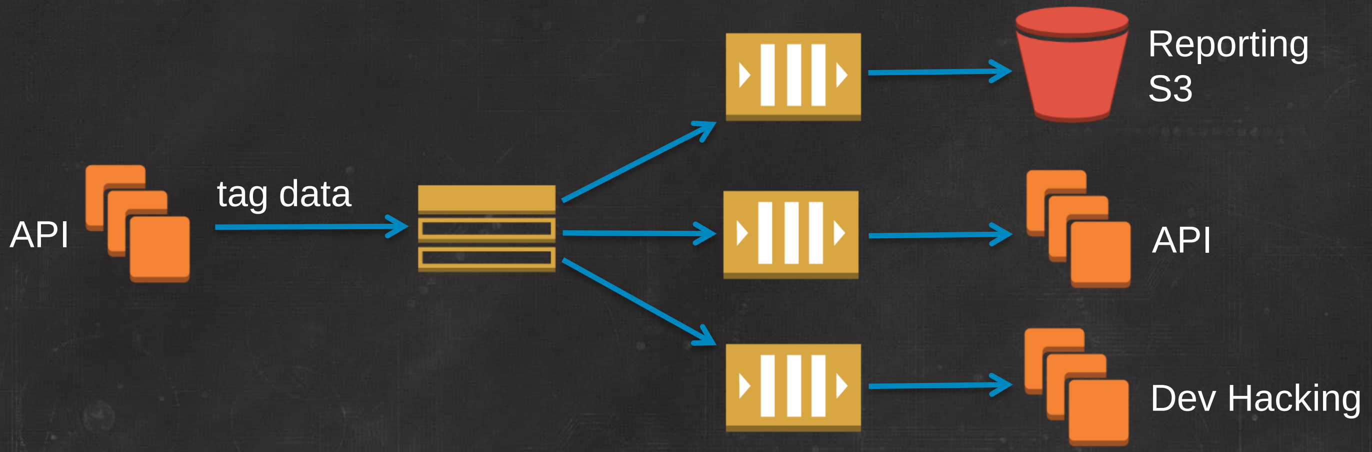
```
public interface Writer <T>  {
    void write(T t);
}
```

SNS/SQS Datastore Segregation

SNS/SQS Datastore Segregation



SNS/SQS Datastore Segregation



SQS for Shazam is…

- Protection from the outside world
- Short term, unbounded persistence
- Cost effective elastic capacity
- Scalable data segregation

Thank you Colin!

Design patterns recap

1. Batch processing
2. IAM Roles for EC2
3. Using SQS to durably batch writes
4. Discard stale messages
5. Simple Notification Service Fan-out
6. Send messages from the browser

Additional messaging resources

- Application Services Booth
- re:Invent sessions:
  - ARC301 Controlling the Flood: Massive Message Processing with AWS SQS and DynamoDB
  - MBL308 Engage Your Customers with Amazon SNS Mobile Push
- AWS Support and Discussion Forums
- AWS Architecture Center: http://aws.amazon.com/architecture
- Documentation: http://aws.amazon.com/documentation/sqs

Next stop:

ARC301 - Controlling the Flood!

Right here in this room.

AWS re:Invent

AWS re:Invent

Please give us your feedback on this presentation

SVC206

As a thank you, we will select prize winners daily for completed surveys!

Thank You

amazon web services