Amazon Kinesis

Capture, Deliver, and Process Real-time Data Streams on AWS

Adi Krishnan, Principal PM Amazon Kinesis
Nick Barrash, SDE, AdRoll

10/08/2015

amazon web services

What to Expect from the Session

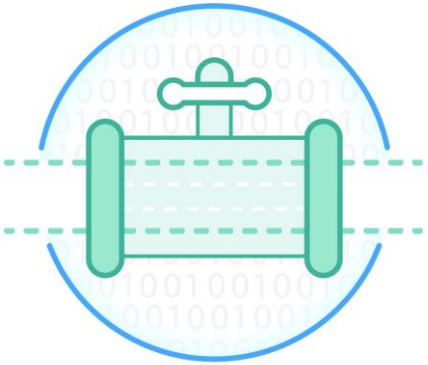
- Amazon Kinesis: System Overview
  - Amazon Kinesis Streams (**this session**)
  - Amazon Kinesis Firehose (available now)
  - Amazon Kinesis Analytics (announced)

- Top 10 things you need to know about Kinesis Streams!
  - Amazon Kinesis Streaming Data Ingestion Model (5 things)
  - Streaming applications with Kinesis Client Library (5 things)

- AdRoll: Re-architecting our real-time data pipeline
  - Featuring Nick Barrash, SDE AdRoll

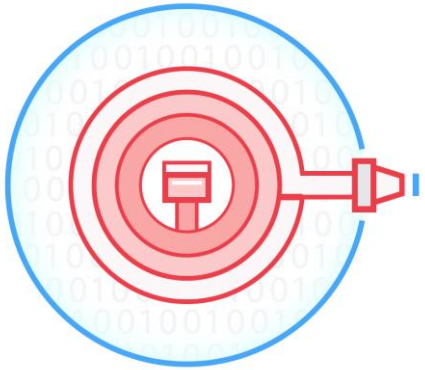
Amazon Kinesis: Streaming data done the AWS way

Services that make it easy to work with real-time data streams on AWS



Amazon Kinesis Streams

Build your own custom applications that process or analyze streaming data



Amazon Kinesis Firehose

Easily load massive volumes of streaming data into Amazon S3 and Redshift

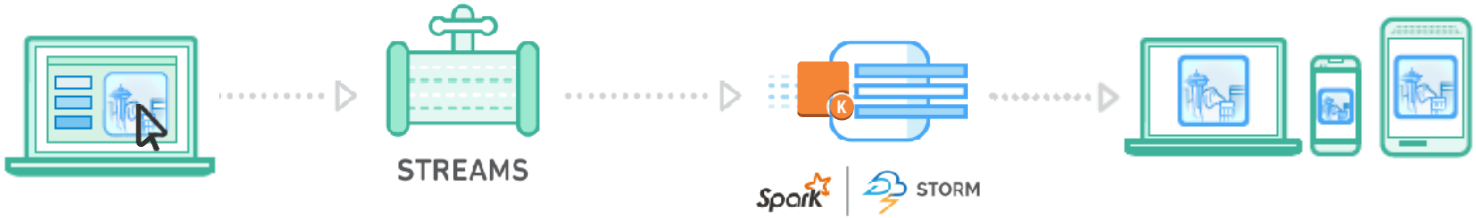
Amazon Kinesis Analytics

Easily analyze data streams using standard SQL queries

Amazon Kinesis Streams

Amazon Kinesis Streams

Build your own data streaming applications



Send clickstream data to Kinesis Streams

Kinesis Streams stores and exposes clickstream data for processing

Custom application built on Kinesis Client Library makes real-time content recommendations

Readers see personalized content suggestions

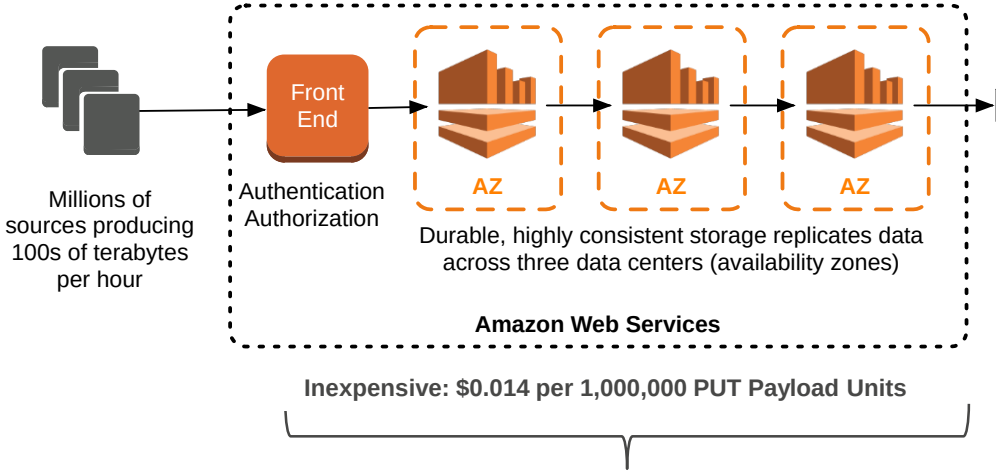
Easy Administration: Create a new stream, set desired capacity and partitioning to match your data throughput rate and volume.

Build real-time applications: Perform custom record processing on streaming data using Kinesis Client Library, Apache Spark/ Storm, AWS Lambda, and more.

Low cost: Cost-efficient for workloads of any scale.

Amazon Kinesis Streams (re:Invent 2013)

Fully managed service for real-time processing of streaming data



Amazon Kinesis Streams: Streaming Data Ingestion

Putting Data into Kinesis

Simple Put* interface to capture and store data in Streams

- A provisioned entity called a Stream composed of Shards
- Producers use a PUT call to store data in a Stream.
- Each record <= 1 MB. PutRecord or PutRecords
- A partition key is supplied by producer and used to distribute (MD5 hash) the PUTs across (hash key range) of Shards
- Unique Sequence# returned upon successful PUT call
- Approximate arrival timestamp affixed to each record

Topic #1: Thinking about ingestion model
Workload determines partition key strategy

Managed Buffer

- Care about a reliable, scalable way to capture data
- Defer all other aggregation to consumer
- Generate Random Partition Keys
- Ensure a high cardinality for Partition Keys with respect to shards, to spray evenly across available shards

Streaming Map-Reduce

- Streaming Map-Reduce: leverage partition keys as a natural way to aggregate data
- For e.g. Partition Keys per billing customer, per DeviceId, per stock symbol
- Design partition keys to scale
- Be aware of “hot partition keys or shards ”

Topic #2: Kinesis PutRecords API

High throughput API for efficient writes to Kinesis

- PutRecords {Records {Data,PartitionKey}, StreamName}
  - Supports 500 records.
  - Record can be =<1 MB, and up to 5 MB for whole request
  - Can include records with different partition keys
  - Ordering not guaranteed
- Successful response includes ShardID and SeqNumber values
- Unsuccessful Response

```
{
    "FailedRecordCount": number,
    "Records": [
        {
            "ErrorCode": "string",
            "ErrorMessage": "string",
            "SequenceNumber": "string",
            "ShardId": "string"
        }
```

Topic #2: Kinesis PutRecords API (Continued)
Dealing with unsuccessful records

- Example Failed Response

```
{
    "FailedRecordCount”: 1,
    "Records": [
        {
            "SequenceNumber": "21269319989900637946712965403778482371",
            "ShardId": "shardId-000000000001"

        },
        {
            “ErrorCode”:”ProvisionedThroughputExceededException”,
            “ErrorMessage”: "Rate exceeded for shard shardId-000000000001 in stream exampleStream

        },
        {
            "SequenceNumber": "21269319989999637946712965403778482985",
            "ShardId": "shardId-000000000002"
        }
    ]
}
```

- Examines the PutRecordsResult objects to resend
- Check FailedRecordCount parameter to confirm failed records.
- If so, each putRecordsEntry with ErrorCode != NULL should be added to a subsequent request

Topic #3: Kinesis Producer Library
Highly configurable library to write to Kinesis

- Collects records and uses PutRecords for high throughput writes

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration()
        .setRecordMaxBufferedTime(3000)
        .setMaxConnections(1)
        .setRequestTimeout(60000)
        .setRegion("us-west-1");

final KinesisProducer kinesisProducer = new KinesisProducer(config);
```

- Integrates seamlessly with the Amazon Kinesis Client Library (KCL) to de-aggregate batched records
- Submits Amazon CloudWatch metrics on your behalf to provide visibility into producer performance
- https://github.com/awslabs/amazon-kinesis-producer

Topic #4: Extended Retention in Kinesis

New!

Default 24 Hours but configurable to 7 days

- 2 New APIs
  - `IncreaseStreamRetentionPeriod(String StreamName, int RetentionPeriodHours)`
  - `DecreaseStreamRetentionPeriod(String StreamName, int RetentionPeriodHours)`
- Use it in one of two-modes
  - as always-on extended retention
  - Raise retention period in response to operational event and/ planned maintenance
- Extended Retention is priced at US$ 0.020/ Shard-Hour

Topic #5: Dealing with provisioned throughput exceeded
Metrics and Re-sharding (SplitShard/ MergeShard)

- Keep track of your metrics
- Monitor CloudWatch metrics: PutRecord.Bytes + GetRecords.Bytes metrics keep track of shard usage
- Retry if rise in input rate is temporary
- Reshard to increase number of shards
- SplitShard – Adds more shards
- MergeShard – Removes shards
- Use the Kinesis Scaling Utility - https://github.com/awslabs/amazon-kinesis-scaling-utils

Metric	Units
PutRecords.Bytes	Bytes
PutRecords.Latency	Milliseconds
PutRecords.Success	Count
PutRecords.Records	Count
Incoming Bytes	Bytes
Incoming Records	Count

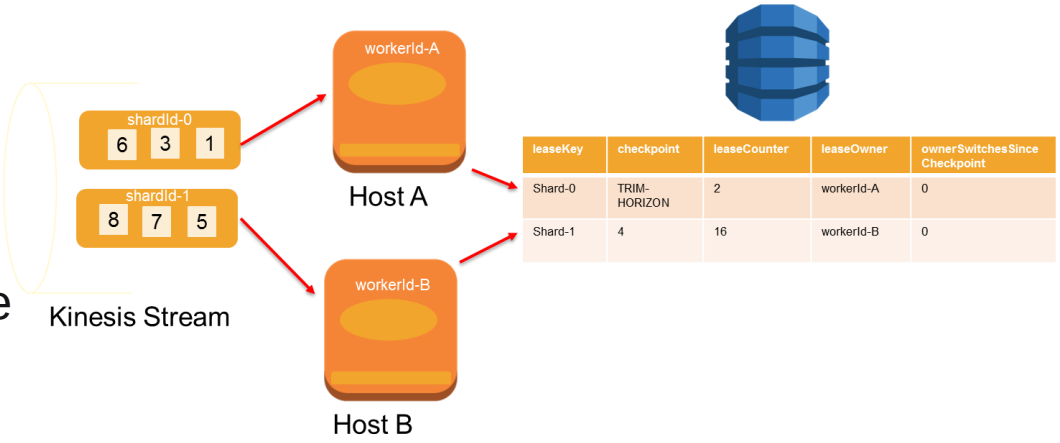
Amazon Kinesis Streams: Build Apps w/ Kinesis Client Library

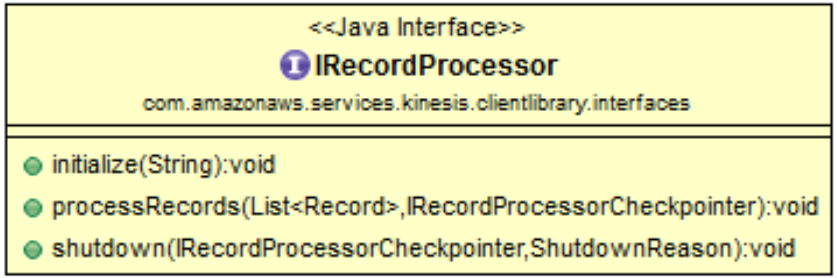
Building Applications: Kinesis Client Library
For fault-tolerant stream processing applications

- Open Source Java client library, also available for Python, Ruby, Node.JS, .NET. Available on Github
- Connects to the stream and enumerates the shards
- Coordinates shard associations with other workers (if any)
- Instantiates a record processor for every shard it manages
- Pulls data records from the stream
- Pushes the records to the corresponding record processor
- Checkpoints processed records
- Balances shard-worker associations when the worker instance count changes
- Balances shard-worker associations when shards are split or merged

State Management with Kinesis Client Library

- A named KCL application has 1 Worker per EC2 instance
- One worker maps to one or more record processors
- One record processor maps to one shard and processes data records from that shard
- The KCL uses the IRecordProcessor interface to communicate with your application
- ProcessRecords() contains the business logic





Topic #1: Empty records returned even with data in stream

- App calls GetRecords in a loop with no back-offs.
- Each call to GetRecords also returns a ShardIterator value, which must be used in the next iteration.
- GetRecords operation is non-blocking. Returns with either relevant data records or with an empty Records element.
- An empty Records element is returned if:
  - There is no more data currently in the shard
  - There is no data near the part of the shard pointed to by the ShardIterator
- In production, the only time the continuous loop should be exited is when the NextShardIterator value is NULL.
  - When NextShardIterator is NULL, current shard has been closed and ShardIteratorvalue would otherwise point past the last record.
  - If app never calls SplitShard or MergeShards, the shard remains open and the calls to GetRecords never return a NextShardIterator value that is NULL.

Topic #2: Some records are skipped
Handle exceptions with processRecords

- Most common cause: Unhandled exception thrown from processRecords.
- Any exception thrown from processRecords is absorbed by KCL.
- To avoid infinite retries on recurring failures, KCL does not resend the batch of records processed at the time of the exception.
- It calls processRecords for the next batch of data records without restarting the record processor. This effectively results in consumer applications observing skipped records.
- Handle all exceptions within processRecords appropriately.

Shard-level metrics on KCL are your best friend

Metric Name	Description
KinesisDataFetcher.getRecords.Success	Number of successful GetRecords operations per Amazon Kinesis stream shard.
KinesisDataFetcher.getRecords.Time	Time taken per GetRecords operation for the Amazon Kinesis stream shard.
UpdateLease.Success	Number of successful checkpoints made by the record processor for the given shard.
UpdateLease.Time	Time taken for each checkpoint operation for the given shard.
DataBytesProcessed	Total size of records processed in bytes on each ProcessTask invocation.
RecordsProcessed	Number of records processed on each ProcessTask invocation.
ExpiredIterator	Number of ExpiredIteratorException received when calling GetRecords.
MillisBehindLatest	Time the current iterator is behind from the latest record in the shard.
RecordProcessor.processRecords.Time	Time taken by the record processor’s processRecords method.

KCL Topic #3: Consumer is reading slower than expected

- Multiple applications have total reads exceeding the per-shard limits.
  - In this case, increase shards in the Amazon Kinesis stream.

- The maximum number of GetRecords per call limit may have been configured with a low value.
  - May have configured the worker with a low value for the maxRecords property. Recommend system defaults for this property.

- The logic inside processRecords call may be taking longer than expected for a number of possible reasons; the logic may be CPU intensive, I/O blocking, or bottlenecked on synchronization.
  - Test run empty record processors and compare the read throughput.

KCL Topic #4: Records of same shard are processed by different record processors at the same time

- One record processor on a particular shard BUT multiple record processors may temporarily process the same shard.
  - A worker instance loses connectivity
  - New and original record processors from the unreachable worker process data from the same shard
- If record processor has shards taken by another record processor, it must handle two cases to perform graceful shutdown:
  - After current call to processRecords is completed, KCL invokes shutdown method on record processor with shutdown reason 'ZOMBIE'. Record processors are expected to clean up any resources and then exit.
  - When checkpointing from a 'zombie' worker, the KCL throws a Shutdown Exception. After receiving, code is expected to exit the current method cleanly.

KCL Topic #5: Duplicates in processing
Consumer retries can cause duplicates

- Consumer (data processing application) retries happen when record processors restart. Record processors for the same shard restart in the following cases:
    - A worker terminates unexpectedly
    - Worker instances are added or removed
    - Shards are merged or split

- In all these cases, the shards-to-worker-to-record-processor mapping is continuously updated to load balance processing. Shard processors that were migrated to other instances restart processing records from the last checkpoint.

KCL Topic #5 Continued …
Make final destination resilient to duplicates

- In KCL app, ensure data being processed is persisted to durable store like DynamoDB, or S3, prior to check-pointing.

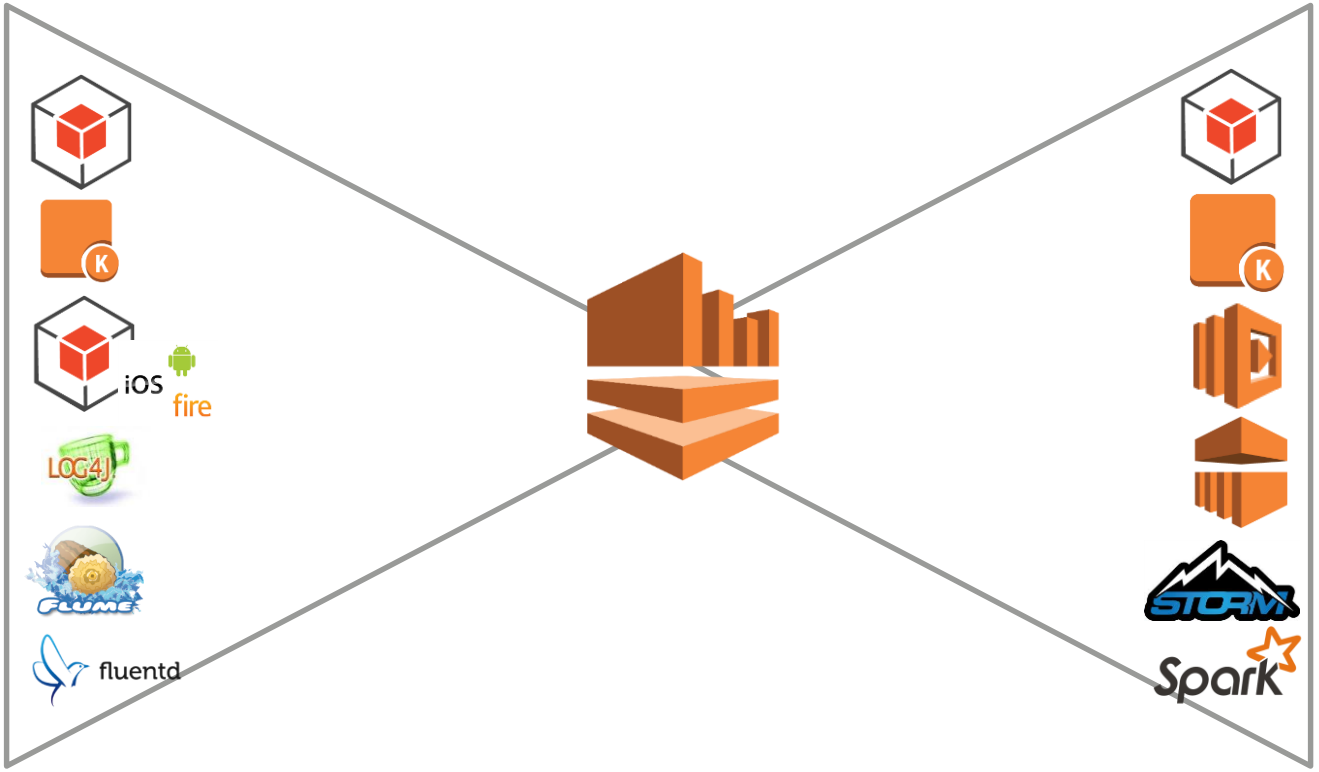
- Duplicates: Make the authoritative data repository (usually at the end of the data flow) resilient to duplicates. That way the rest of the system has a simple policy – keep retrying until you succeed.

- Idempotent Processing: Use number of records since previous checkpoint, to get repeatable results when the record processors fail over.
  - Record Processor uses a fixed number of records per Amazon S3 file, such as 5000.
  - The file name uses this schema: Amazon S3 prefix, shard-id, and First-Sequence-Num. Something like sample-shard000001-10001.
  - After uploading the S3 file, checkpoint by specifying Last-Sequence-Num. In this case, checkpoint at record number 15000.

Sending & Reading Data from Kinesis Streams

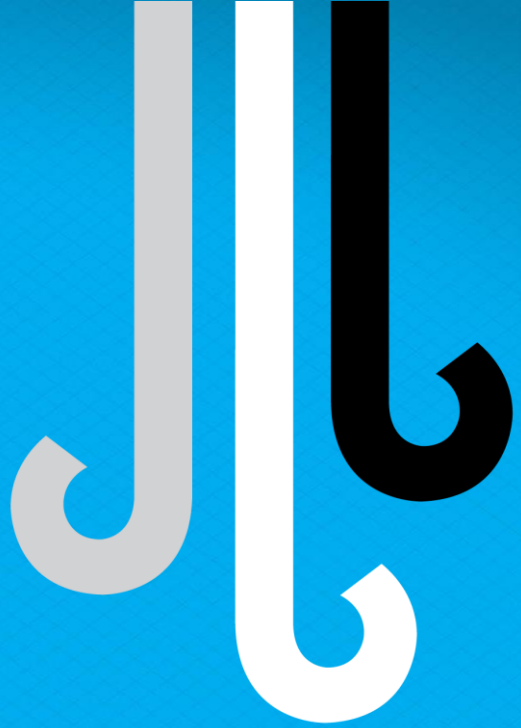
Sending

- AWS SDK
- Kinesis Producer Library
- AWS Mobile SDK
- LOG4J
- Flume
- Fluentd



Consuming

- Get* APIs
- Kinesis Client Library + Connector Library
- AWS Lambda
- Amazon Elastic MapReduce
- Apache Storm
- Apache Spark



Nick Barrash

Kinesis at Adroll

Menu

- AdRoll’s real-time data pipeline
  - Old architecture
  - Motivation to move to Kinesis
- Producer applications
- Consumers applications
- Kinesis pipeline architectures

AdRoll
Our Mission

AdRoll is a performance marketing platform that enables brands to leverage their data to run intelligent, ROI-positive marketing campaigns

AdRoll
Our Data

- Real-time data at AdRoll
  - Advertiser’s website clickstream data
  - Serving ads
    - Impressions / Clicks / Conversions
    - Real-time bidding (RTB) data
- Recency directly correlated with performance

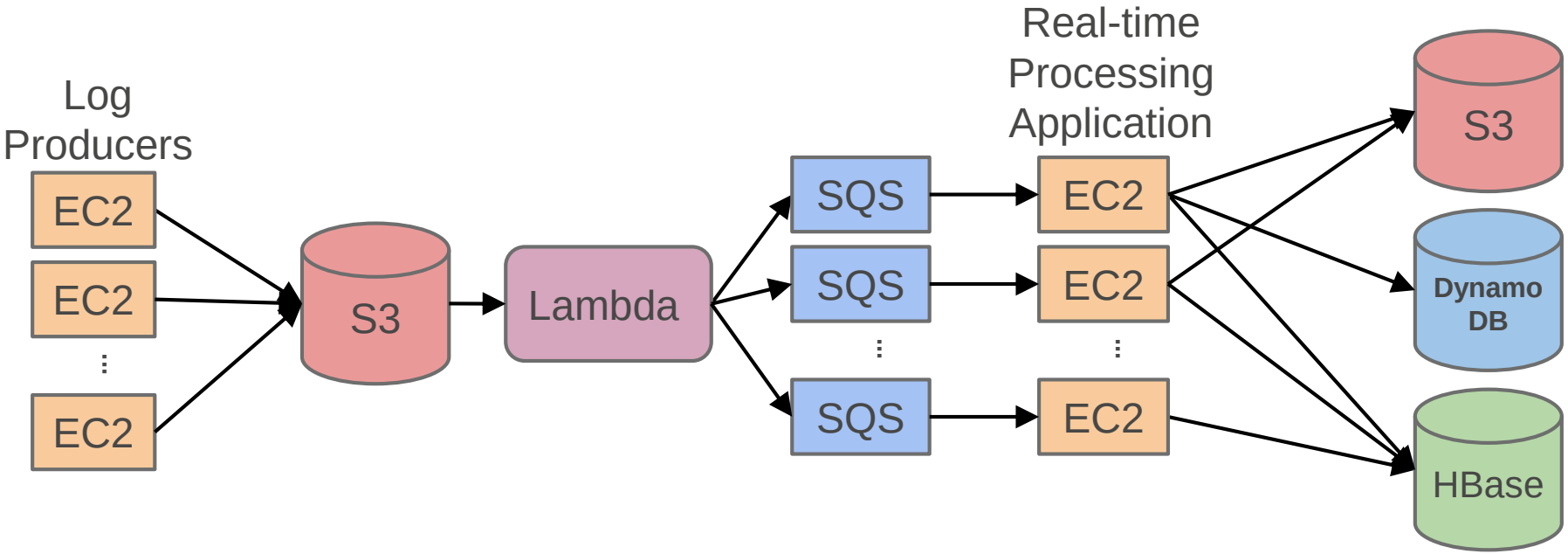
AdRoll
Our Scale

- Daily impression data: 200 million events / 240GB
- Daily user data: 1.6 billion events / 700GB
- Daily RTB data: 60 billion events / 80TB

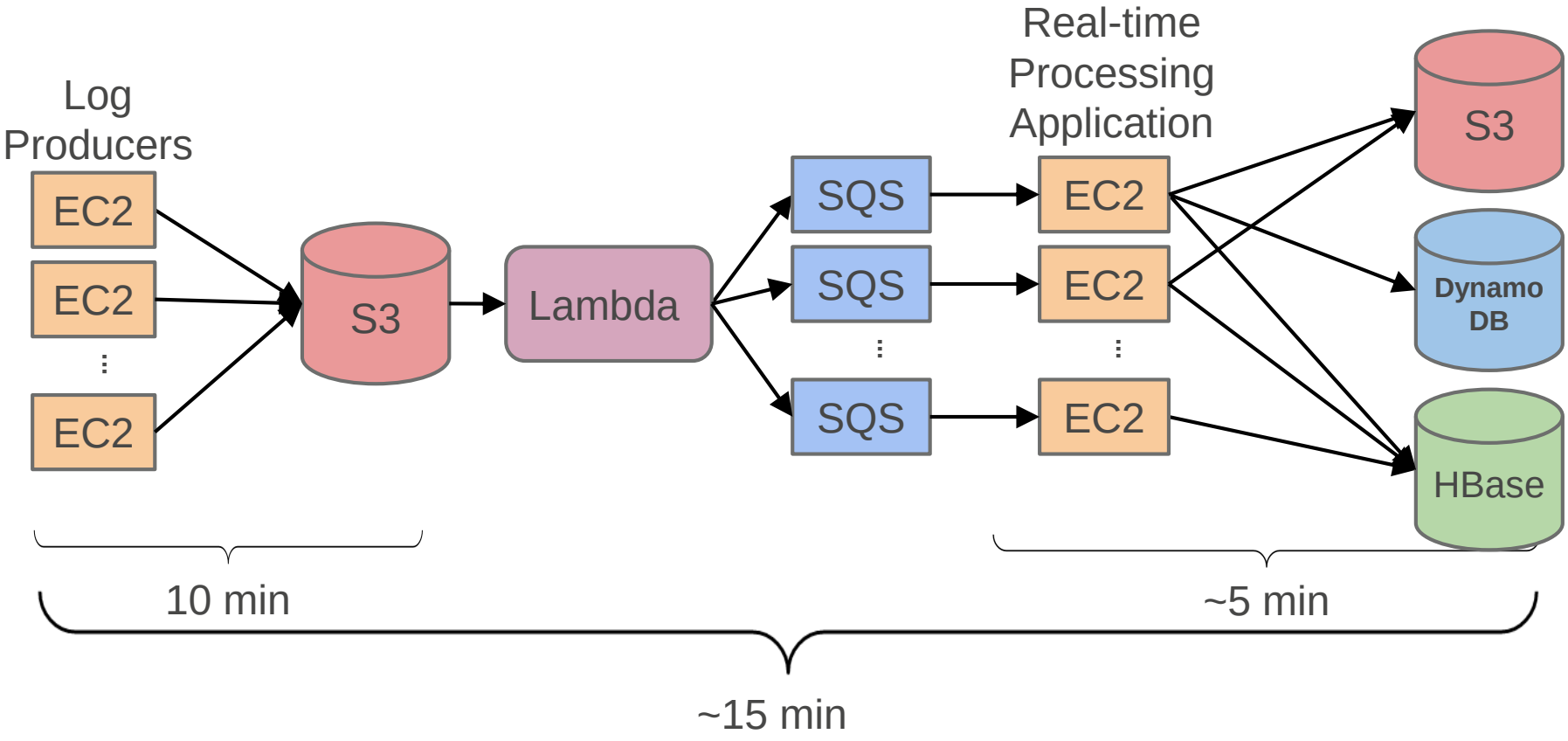
AdRoll
Our Scale

- Daily impression data: 200 million events / 240GB
- Daily user data:1.6 billion events / 700GB
- Daily RTB data: 60 billion events / 80TB

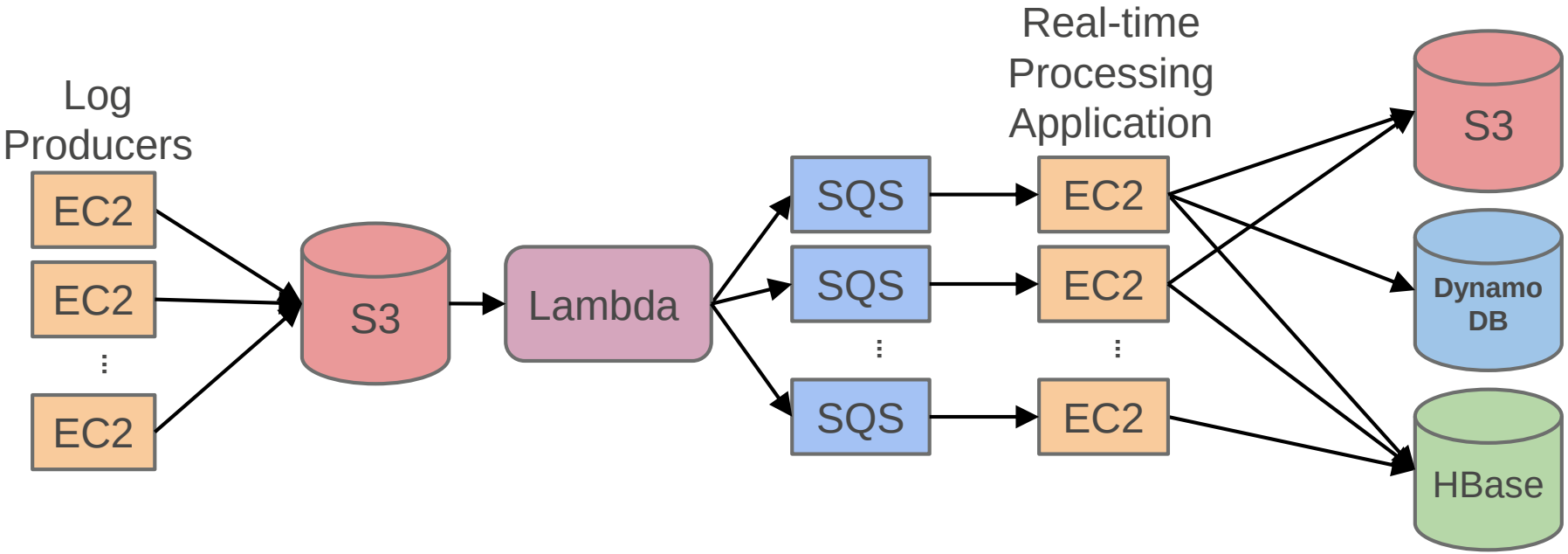
Data Pipeline Architecture
Pre-Kinesis



Data Pipeline Architecture
Pre-Kinesis



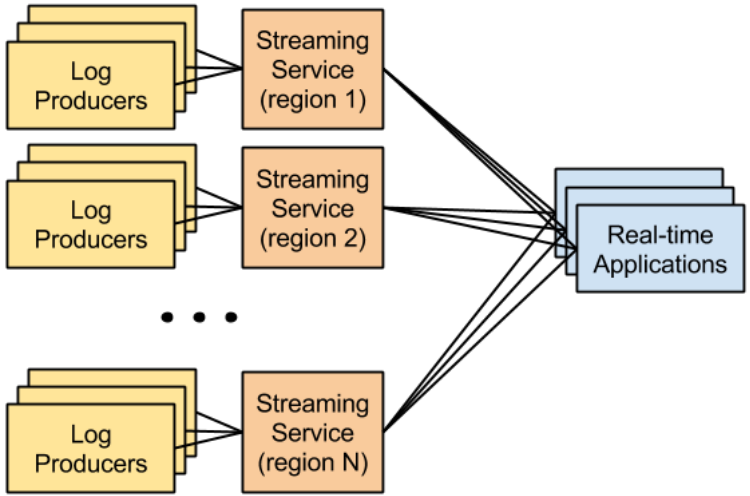
Data Pipeline Architecture
Pre-Kinesis



...streaming service?

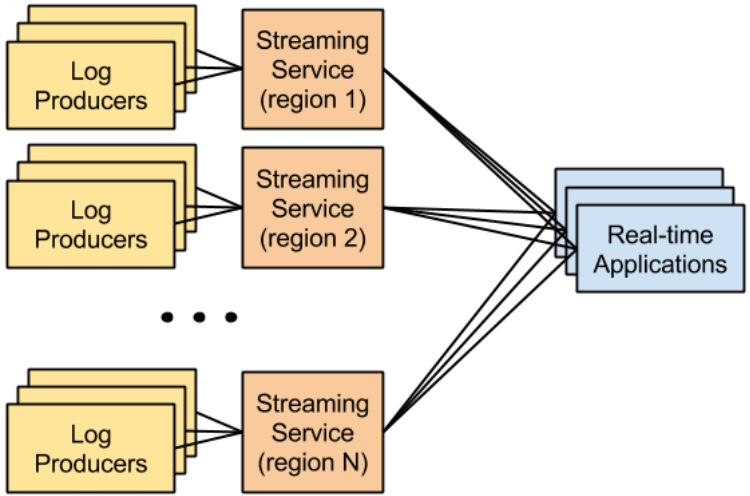
Streaming Service Requirements

- Low latency
- Horizontally scalable
- Durable
- High availability
- Globally deployable



Streaming Service Requirements

- Low latency
- Horizontally scalable
- Durable
- High availability
- Globally deployable



And so we tried...

Solution?

+

Kafka

- Had Issues with
  - Allocating appropriate amount of disk space per topic
  - Setting up cross datacenter mirroring
- Couldn’t spare manpower required to manage Kafka ourselves

Streaming Service Requirements

- Low ops burden
- Low latency
- Horizontally scalable
- Durable
- High availability
- Globally deployable

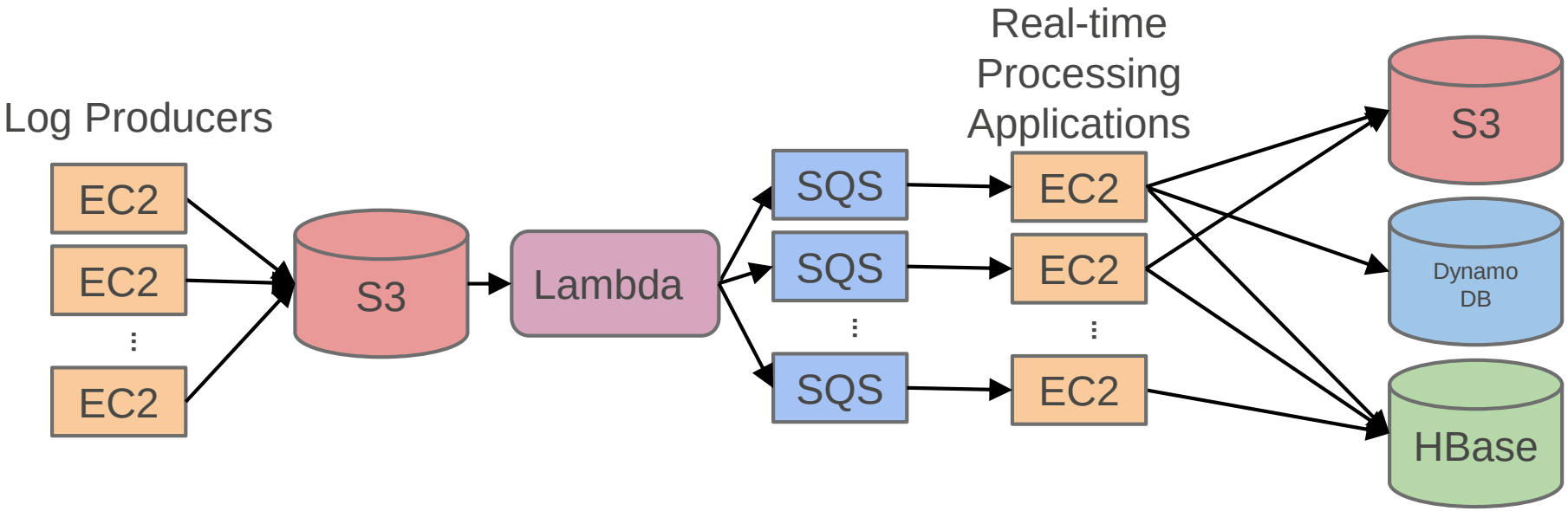
Streaming Service Requirements

- Low ops burden
- Low latency
- Horizontally scalable
- Durable
- High availability
- Globally deployable

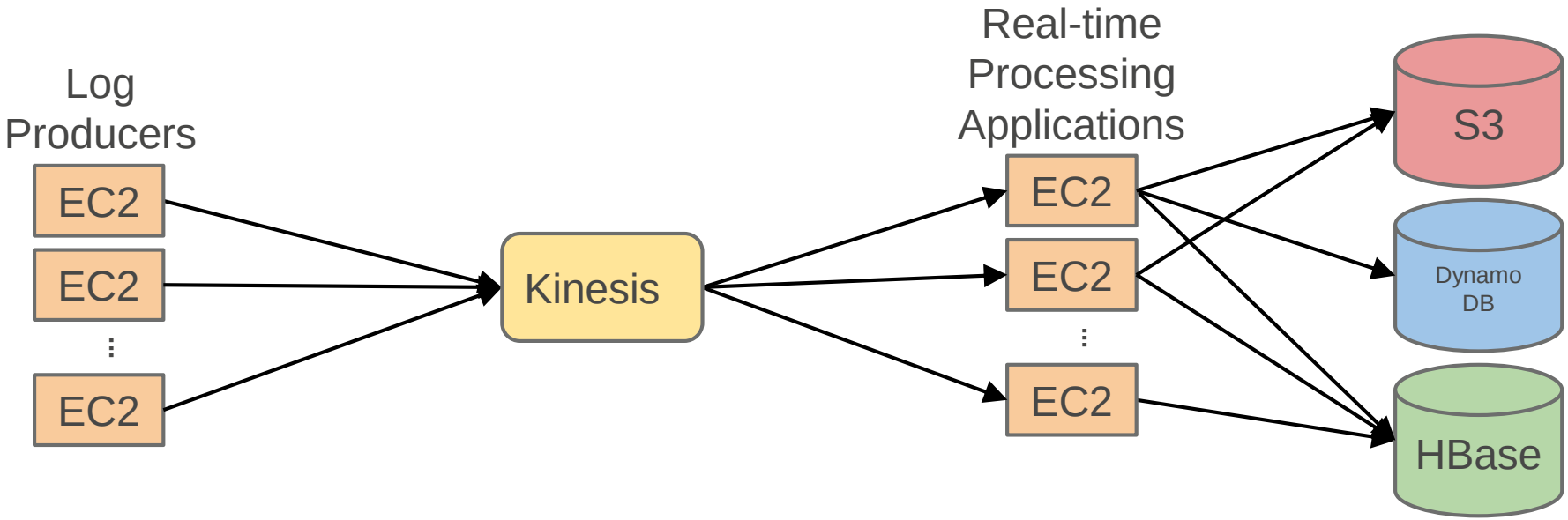
And so next we tried...

Better Solution!

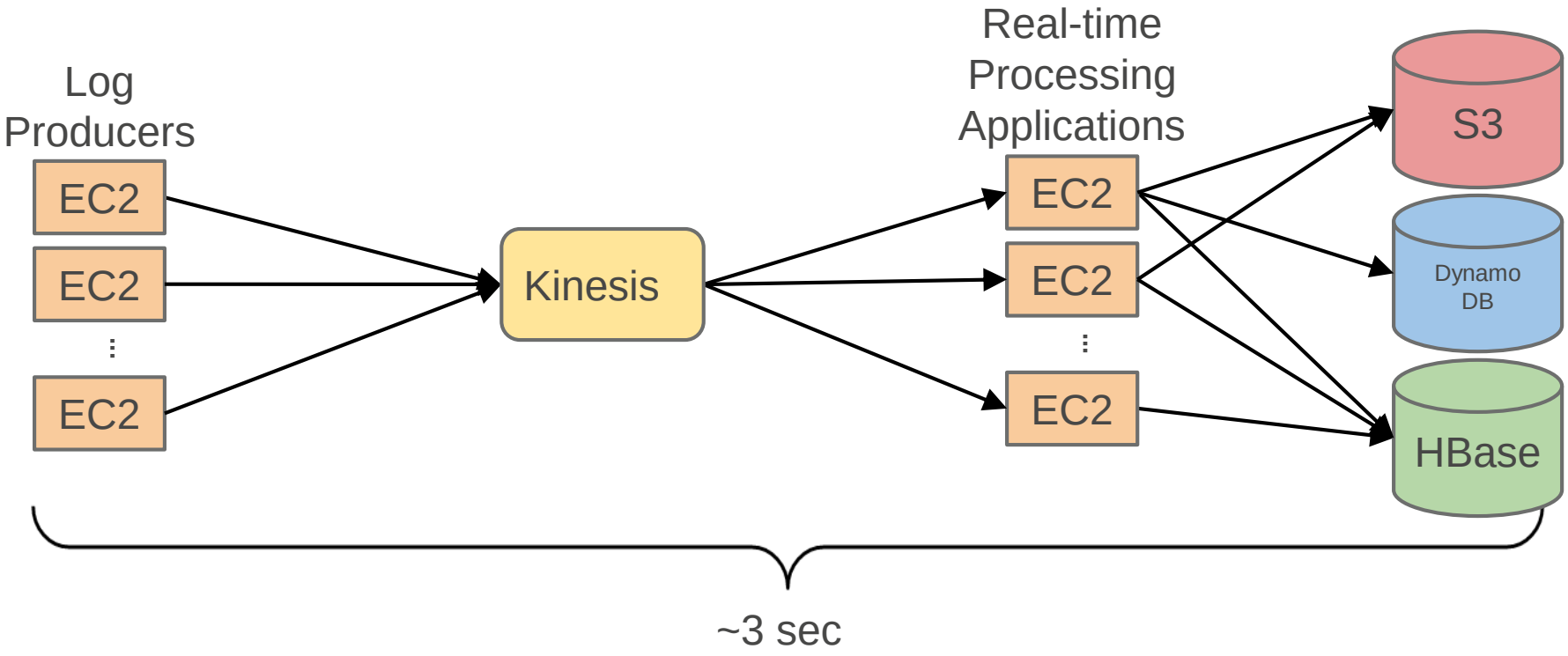
Data Pipeline Architecture
Pre-Kinesis



Data Pipeline Architecture
Post-Kinesis



Data Pipeline Architecture
Post-Kinesis



Kinesis Usage

- Across 5 regions...
- 155 streams
- 264 shards
- 2.5 billion events (100 million Kinesis records) / day

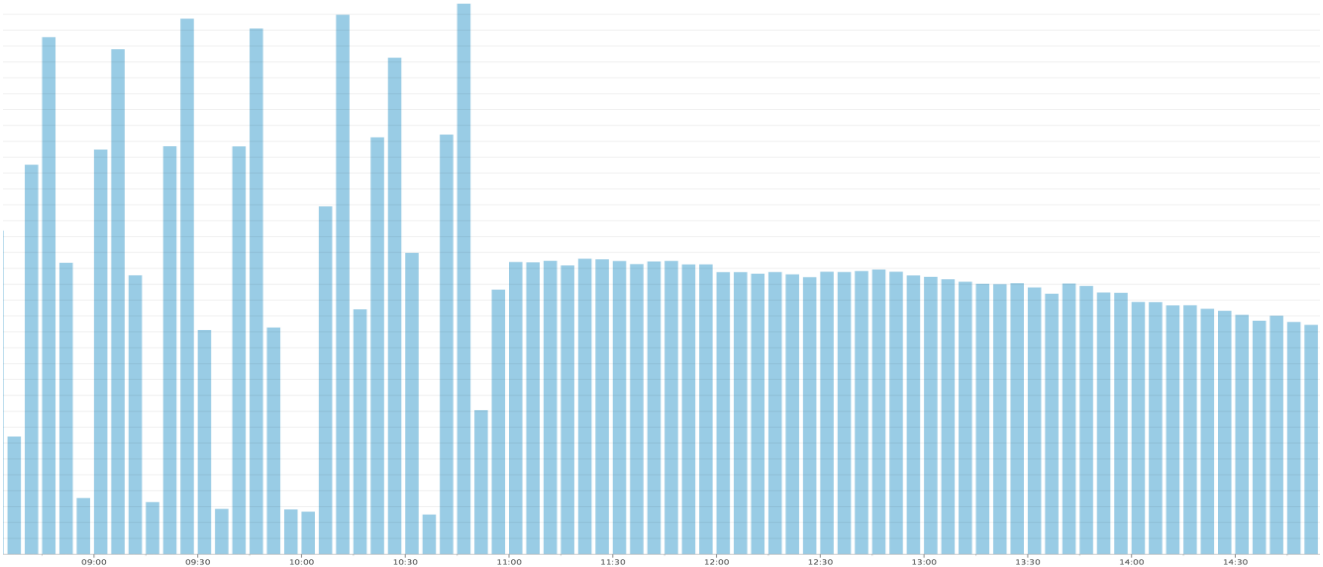
Kinesis Usage

- Across 5 regions...
- 155 streams
- 264 shards
- 2.5 billion events (100 million Kinesis records) / day

▶ Kinesis	$3,055.77

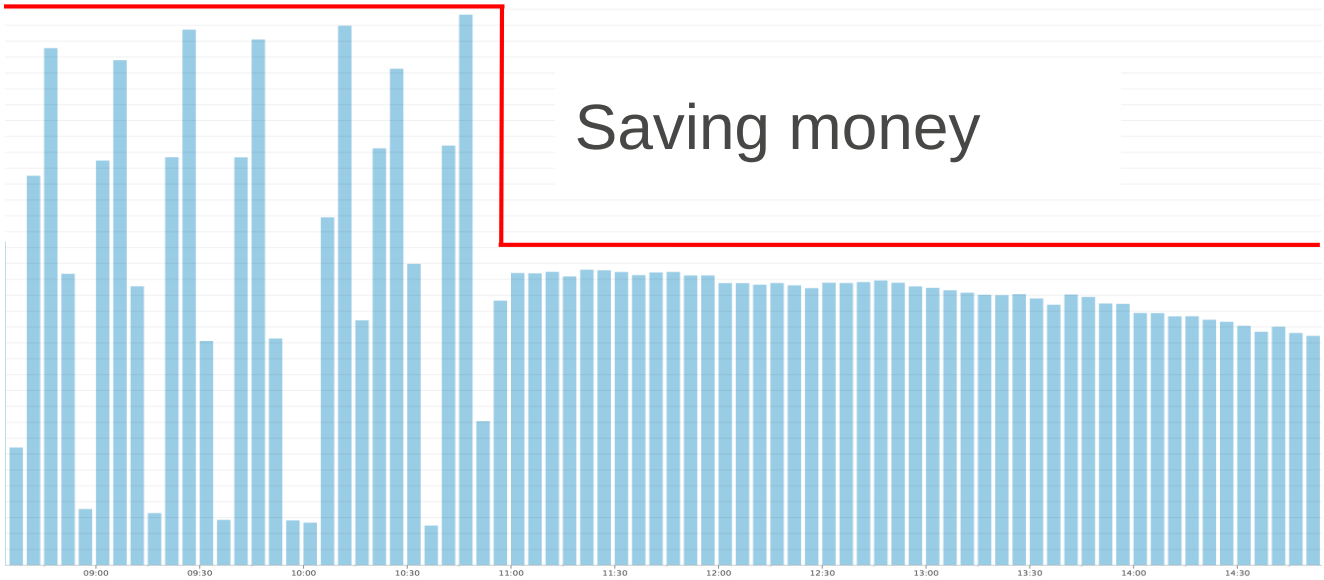
Kinesis Wins

- 15 minutes -> 3 seconds
- While simultaneously...



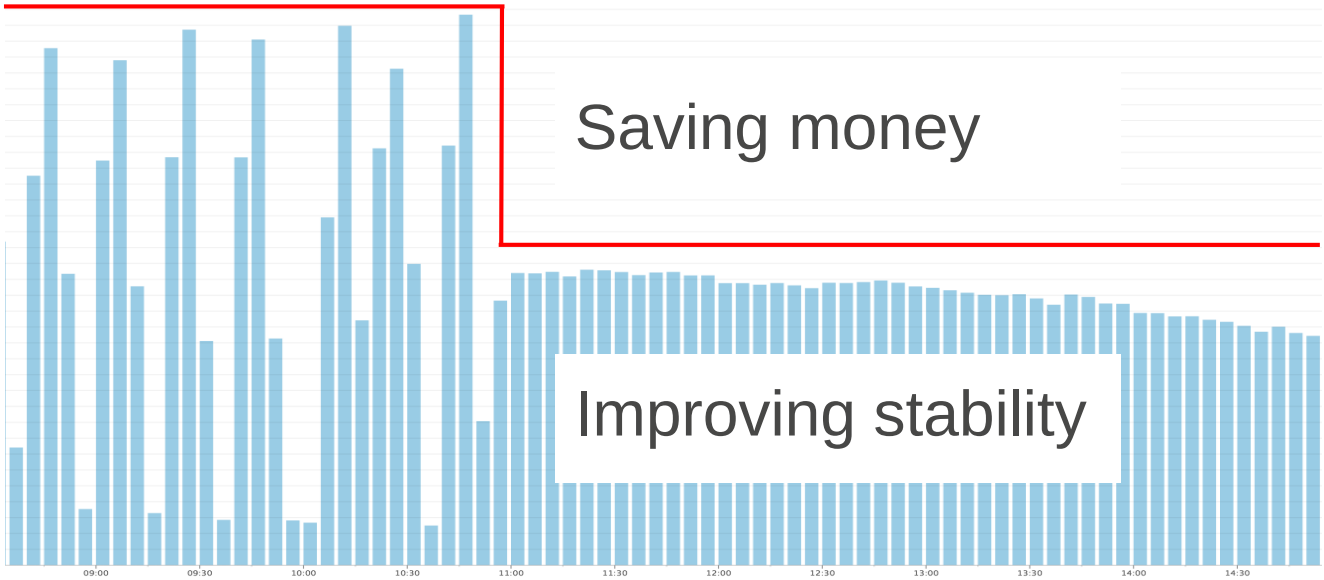
Kinesis Wins

- 15 minutes -> 3 seconds
- While simultaneously...



Kinesis Wins

- 15 minutes -> 3 seconds
- While simultaneously...

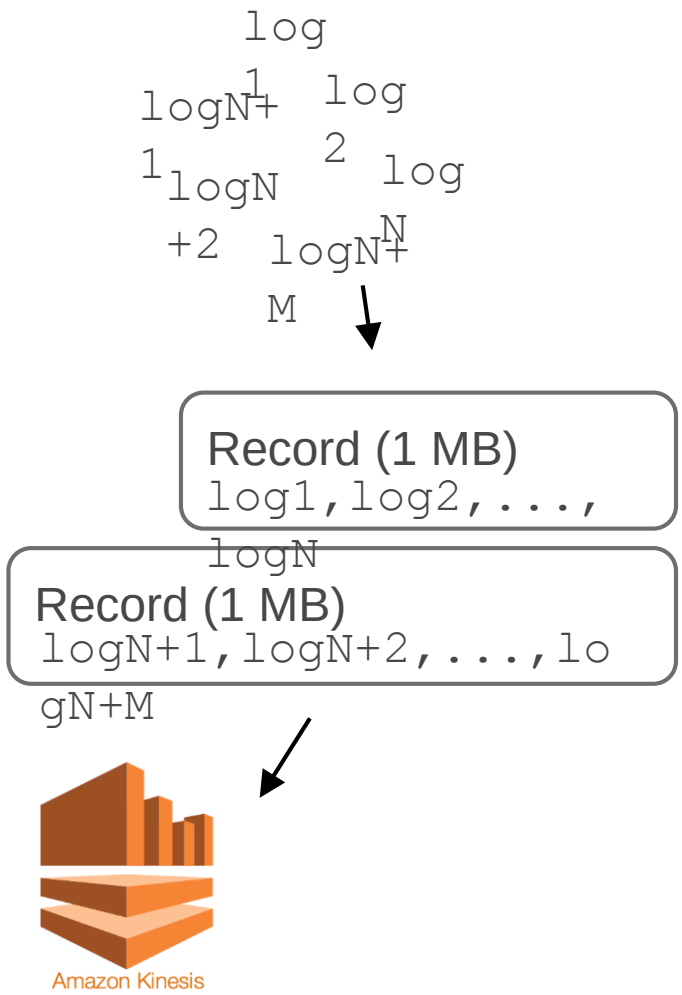


Kinesis Producers
Our Setup

- 300+ servers generating logs
  - 700 records per second per host
- Primarily written in Erlang
  - github.com/AdRoll/kinetic

Kinesis Producers Best Practices

- Concatenate logs
  - Flush at max record size
  - Or 1 second has passed
  - Big cost saver
- Retries + Exponential Backoff (+ Jitter?)
- Randomized partition keys



Kinesis Consumers
Our Setup

- 12 major Kinesis consumer applications
- github.com/awslabs/kinesis-storm-spout (storm)
- github.com/AdRoll/kinetic (erlang)

Kinesis Consumers
Storm spout library (Amazon's version)

```
public class KinesisShardGetter {
...
  AmazonKinesisClient kinesisClient;
...
  // read and emit Kinesis records
  public List<Record> getNext(int numRecords) {
    GetRecordsRequest request = new GetRecordsRequest();
    request.setLimit(numRecords);
    GetRecordsResult result = kinesisClient.getRecords(request);
    return result.getRecords();
  }
```

Kinesis Consumers
Storm spout library (Amazon's version)

```
public class KinesisShardGetter {
...
  AmazonKinesisClient kinesisClient;
...
  // read and emit Kinesis records
  public List<Record> getNext(int numRecords) {
    GetRecordsRequest request = new GetRecordsRequest();
    request.setLimit(numRecords);
    GetRecordsResult result = kinesisClient.getRecords(request);
    return result.getRecords();
  }
```

Kinesis Consumers
Storm spout library (Amazon's version)

```
public class KinesisShardGetter {
...
  AmazonKinesisClient kinesisClient;
...
  // read and emit Kinesis records
  public List<Record> getNext(int numRecords) {
    GetRecordsRequest request = new GetRecordsRequest();
    request.setLimit(numRecords);
    GetRecordsResult result = kinesisClient.getRecords(request);
    return result.getRecords();
  }
```

Kinesis Consumers
Storm spout library (Amazon's version)

```
public class KinesisShardGetter {
...
  AmazonKinesisClient kinesisClient;
...
  // read and emit Kinesis records
  public List<Record> getNext(int numRecords) {
    GetRecordsRequest request = new GetRecordsRequest();
    request.setLimit(numRecords);
    GetRecordsResult result = kinesisClient.getRecords(request);
    return result.getRecords();
  }
```

Kinesis Consumers
Storm spout library (AdRoll's version)

```
AmazonKinesisAsyncClient kinesisClient;
final ConcurrentLinkedQueue<Record> recordsQueue;
...
// read Kinesis records
public void readRecords() {
  final GetRecordsRequest request = new GetRecordsRequest();
  kinesisClient.getRecordsAsync(request, new AsyncHandler<GetRecordsRequest, GetRecordsResult>() {

    @Override
    public void onSuccess(GetRecordsRequest request, GetRecordsResult result) {
      recordsQueue.addAll(result.getRecords());
      readRecords();
    }
    ...
  }
```

```
@Override
public List<Record> getNext(int numRecords) {
  List<Record> records = new ArrayList<>();
  for (int i = 0; i < numRecords; i++) {
    if (!recordsQueue.isEmpty()) {
      records.add(recordsQueue.pop());
    } else { break; }
  }
  return records;
}
```

Kinesis Consumers
Storm spout library (AdRoll's version)

```
AmazonKinesisAsyncClient kinesisClient;
final ConcurrentLinkedQueue<Record> recordsQueue;

// read Kinesis records
public void readRecords() {
  final GetRecordsRequest request = new GetRecordsRequest();
  kinesisClient.getRecordsAsync(request, new AsyncHandler<GetRecordsRequest, GetRecordsResult>()

    @Override
    public void onSuccess(GetRecordsRequest request, GetRecordsResult result) {
      recordsQueue.addAll(result.getRecords());
      readRecords();
    }

  }
```

```
@Override
public List<Record> getNext(int numRecords) {
  List<Record> records = new ArrayList<>();
  for (int i = 0; i < numRecords; i++) {
    if (!recordsQueue.isEmpty()) {
      records.add(recordsQueue.pop());
    } else { break; }
  }
  return records;
}
```

Kinesis Consumers
Storm spout library (AdRoll's version)

```
Amaz
fina ConcurrentLinkedQueue<Record> recordsQueue;

// read Kinesis records
public void readRecords() {
  final GetRecordsRequest request = new GetRecordsRequest();
  kinesisClient.getRecordsAsync(request, new AsyncHandler<GetRecordsRequest, GetRecordsResult>() {

    @Override
    st, GetRecordsResult result) {
      recordsQueue.addAll(result.getRecords());
    }

  }
```

```
@Override
public List<Record> getNext(int numRecords) {
  List<Record> records = new ArrayList<>();
  for (int i = 0; i < numRecords; i++) {
    if (!recordsQueue.isEmpty()) {
      records.add(recordsQueue.pop());
    } else { break; }
  }
  return records;
}
```

Kinesis Consumers
Storm spout library (AdRoll's version)

```
AmazonKinesisAsyncClient kinesisClient;
final ConcurrentLinkedQueue<Record> recordsQueue;

// read Kinesis records
public void readRecords() {
  final GetRecordsRequest request = new GetRecordsRequest();
  kinesisClient.getRecordsAsync(request, new AsyncHandler<GetRecordsRequest, GetRecordsResult>() {

    @Override
    public void onSuccess(GetRecordsRequest request, GetRecordsResult result) {
      recordsQueue.addAll(result.getRecords());
      readRecords();
    }
  }
```

```
@Override
public List<Record> getNext(int numRecords) {
  List<Record> records = new ArrayList<>();
  for (int i = 0; i < numRecords; i++) {
    if (!recordsQueue.isEmpty()) {
      records.add(recordsQueue.pop());
    } else { break; }
  }
  return records;
}
```

Kinesis Consumers
Storm spout library (AdRoll's version)

```
AmazonKinesisAsyncClient kinesisClient;
final ConcurrentLinkedQueue<Record> recordsQueue;
...
// read Kinesis records
public void readRecords() {
  final GetRecordsRequest request = new GetRecordsRequest();
  kinesisClient.getRecordsAsync(request, new AsyncHandler<GetRecordsRequest, GetRecordsResult>() {

    @Override
    public void onSuccess(GetRecordsRequest request, GetRecordsResult result) {
      recordsQueue.addAll(result.getRecords());
      readRecords();
    }
    ...
  }
```

```
public List<Record> getNext(int numRecords) {
  List<Record> records = new ArrayList<>();
  for (int i = 0; i < numRecords; i++) {
    if (!recordsQueue.isEmpty()) {
      records.add(recordsQueue.pop());
    } else { break; }
  }
  return records;
}
```

Kinesis Consumers
Storm spout library (AdRoll's version)

```
Amaz
fina ConcurrentLinkedQueue<Record> recordsQueue;
...
// read Kinesis records
public void readRecords() {
  final GetRecordsRequest request = new GetRecordsRequest();
  kinesisClient.getRecordsAsync(request, new AsyncHandler<GetRecordsRequest, GetRecordsResult>() {

    @Override
    public void onSuccess(GetRecordsRequest request, GetRecordsResult result) {
      recordsQueue.addAll(result.getRecords());
      readRecords();
    }
    ...
  }
```

```
public List<Record> getNext(int numRecords) {
  List<Record> records = new ArrayList<>();
  for (int i = 0; i < numRecords; i++) {
    if (!records
      records.ad recordsQueue.pop());
    } else { bre
  }
  return records;
```

Kinesis Pipeline Architecture
Kinesis limitations

- Each shard can support up to 5 transactions per second for reads, up to a maximum total data read rate of 2 MB per second.
- Each shard can support up to 1,000 records per second for writes, up to a maximum total data write rate of 1 MB per second (including partition keys).

Kinesis Pipeline Architecture
Kinesis limitations

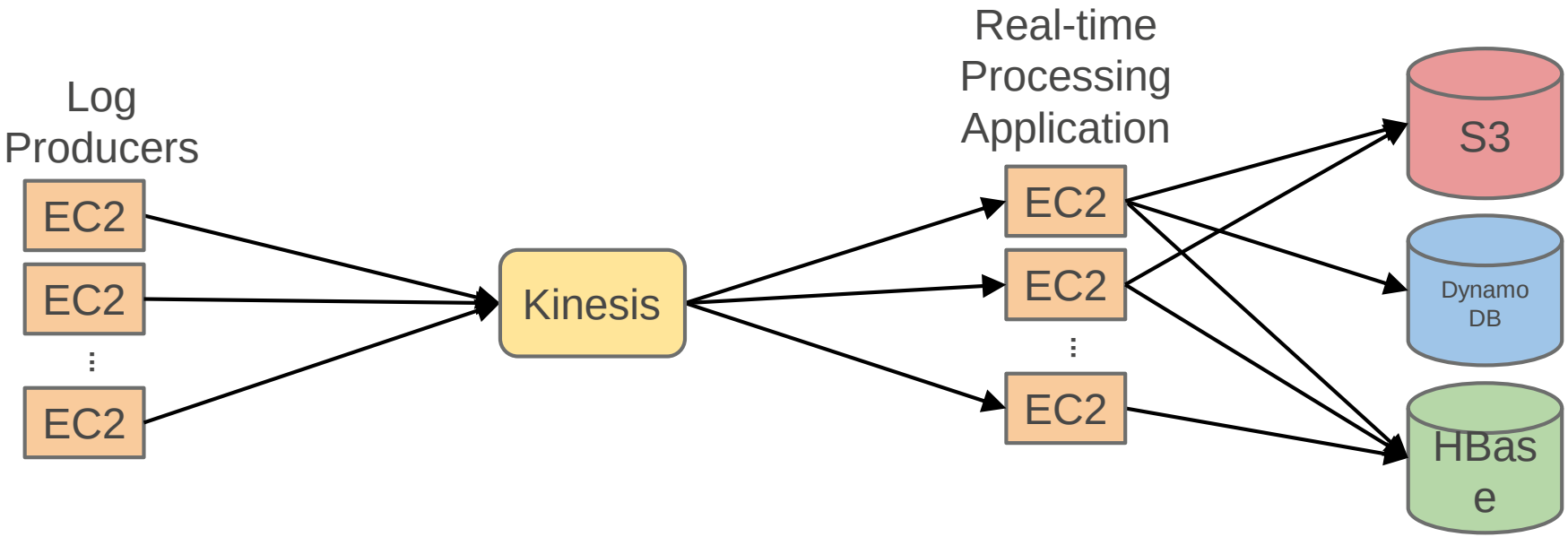
- Each shard can support up to 5 transactions per second for reads, up to a maximum total data read rate of 2 MB per second.
- Each shard can support up to 1,000 records per second for writes, up to a maximum total data write rate of 1 MB per second (including partition keys).

Kinesis Pipeline Architecture
Kinesis limitations

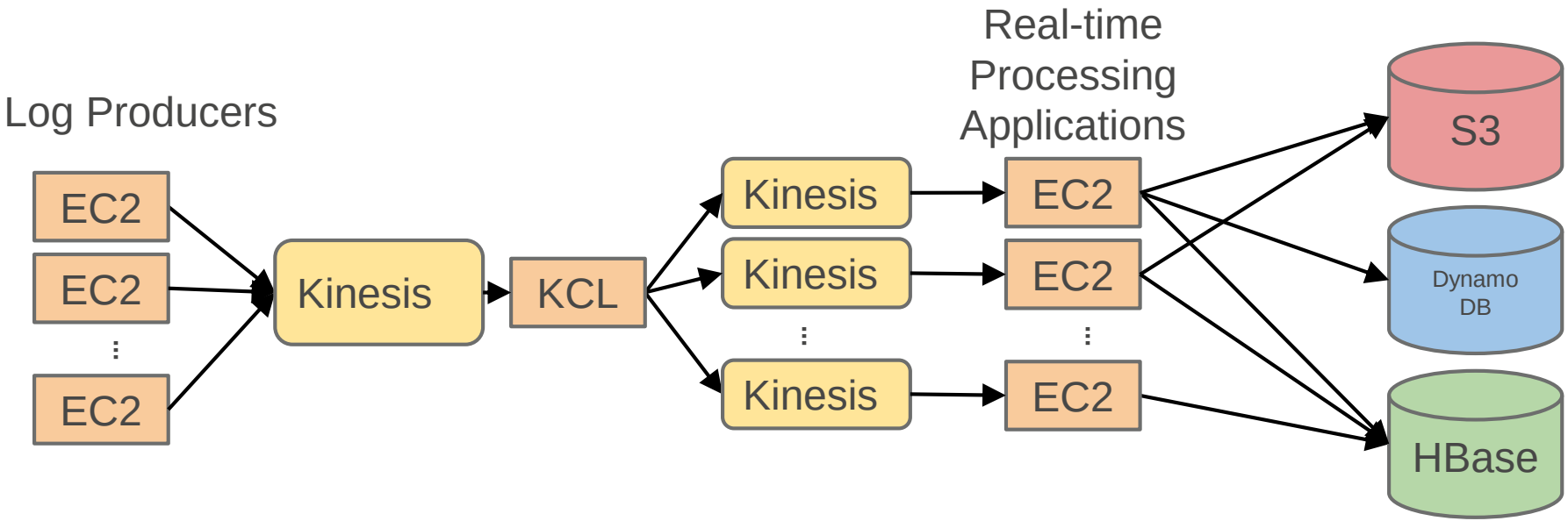
- Each shard can support up to 5 transactions per second for reads, up to a maximum total data read rate of 2 MB per second.
- Each shard can support up to 1,000 records per second for writes, up to a maximum total data write rate of 1 MB per second (including partition keys).

- Tough to have large number of applications per stream
  - Coordinate 0.2MB reads every N/5 seconds?
    - New applications?
    - Testing?

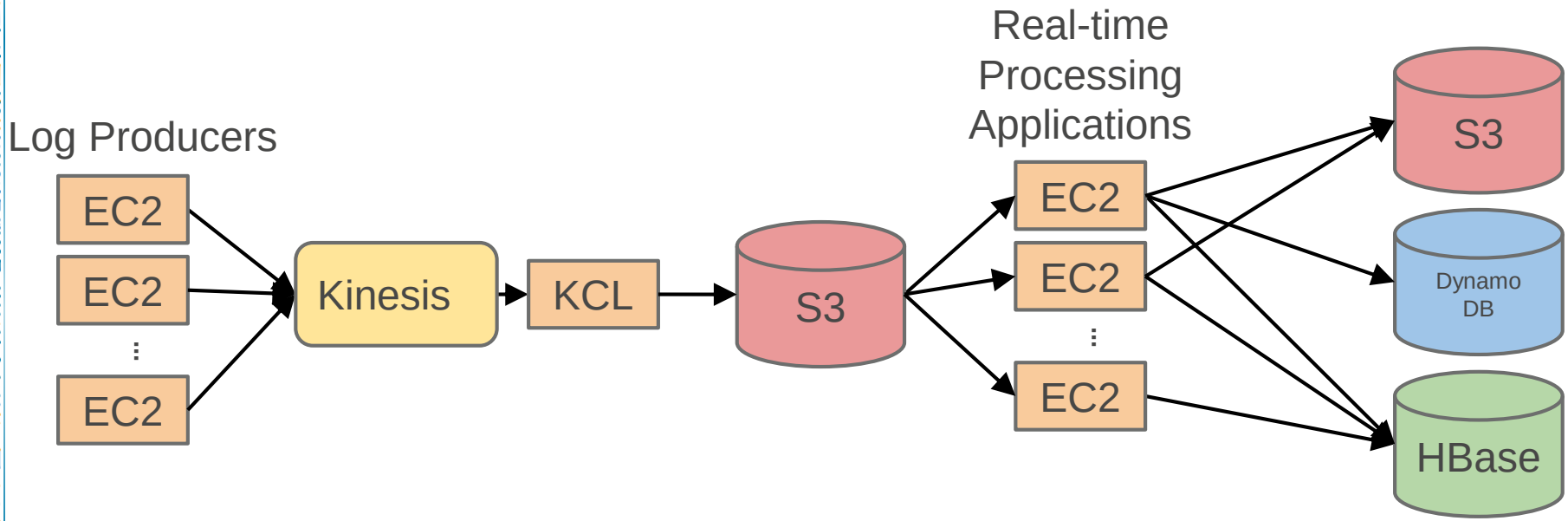
Kinesis Pipeline Architecture
Naive



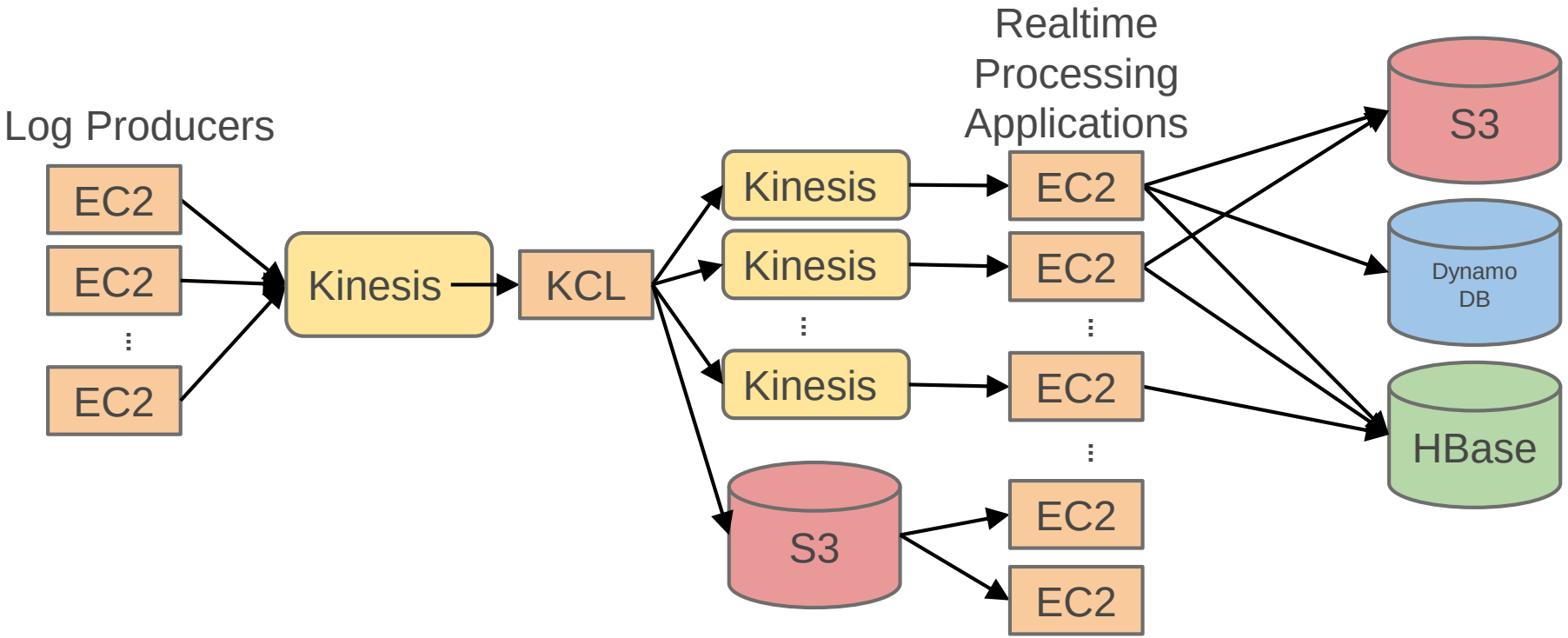
Kinesis Pipeline Architecture
Application Kinesis streams



Kinesis Pipeline Architecture
Flush to S3



Kinesis Pipeline Architecture
Application Streams + S3

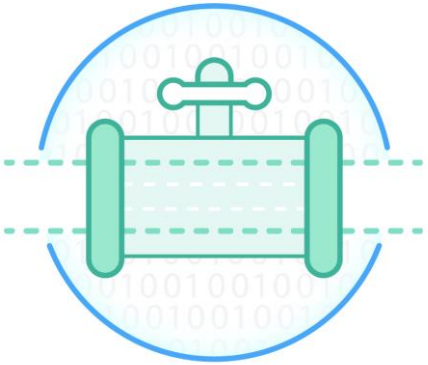


Summary

- Consider moving Kinesis near your data producers
- Better (cheaper?) to work with smooth, constant load instead of bursty batch load
- Batch data inside Kinesis records
- Try to make asynchronous requests
- Put a KCL application in the middle of your streams to allow for multiple real-time applications

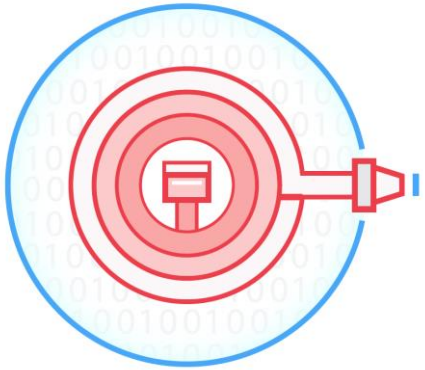
Amazon Kinesis: Streaming data made easy

Services make it easy to capture, deliver, and process streams on AWS



Amazon Kinesis Streams

Build your own custom applications that process or analyze streaming data



Amazon Kinesis Firehose

Easily load massive volumes of streaming data into Amazon S3 and Redshift

Amazon Kinesis Analytics

Easily analyze data streams using standard SQL queries

Thank you!

adad

Appendix

Remember to complete your evaluations!