Hive to Spark – Journey and Lessons Learned

William Lau, Kent Buenaventura
Unity Technologies

Agenda

- Our journey moving our data pipeline to use Spark
- Key factors that made the migration successful
- Walkthrough of how we use Spark in our data pipeline

Analytics @ Unity

- The leading game development platform
  - 3D Engine, Tools, and Services (Analytics, Ads)
  - 34% of top 1000 games are made with Unity
- Analytics for game developers
- We are hiring!

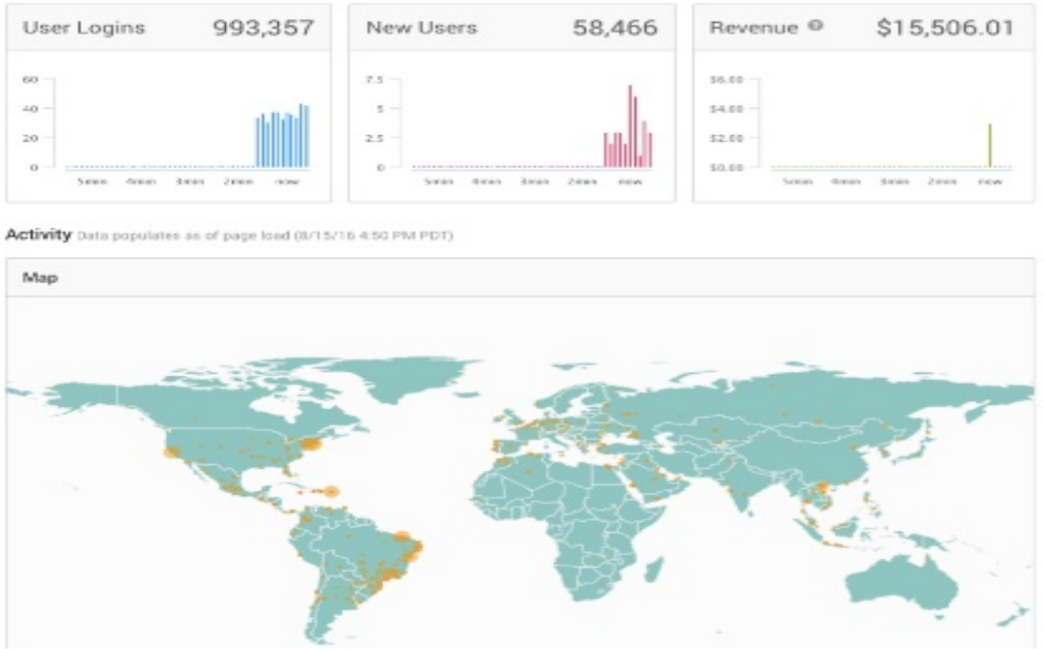
Made with Unity

Unity Analytics

Q1 2016:

Games **Game Installs** **Unique Devices**

Unity Analytics



Background

Legacy Data Pipeline (Hive based)

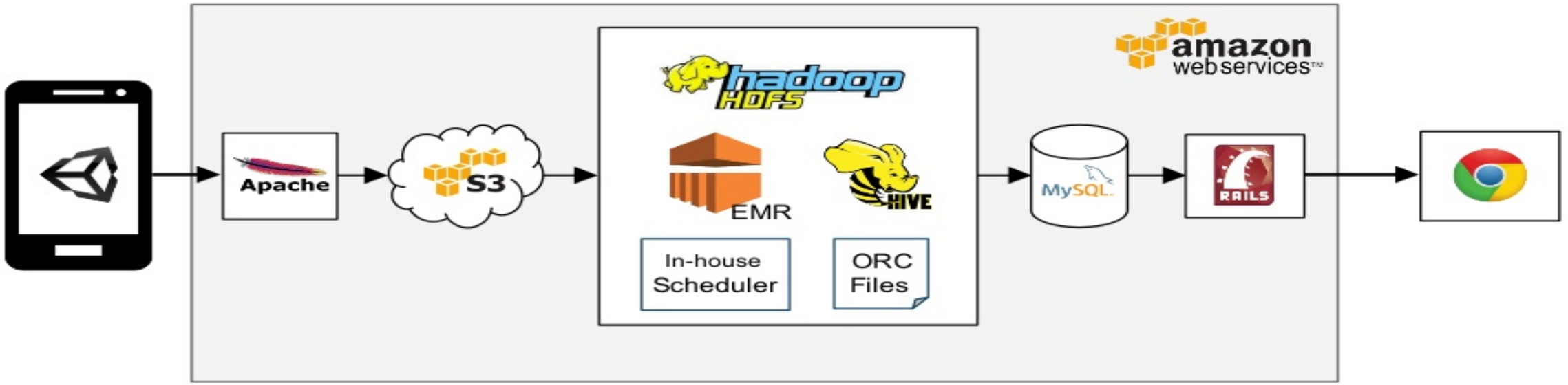
- Scaling and Performance issues
- Operational challenges
- Overly complex

Migrating to Spark

Key decision points:

- Invest in next generation technology stack
- Spark's ecosystem - community, libraries, tools
- Case studies show solid scaling/performance
- Single stack for batch and streaming

Architecture with Legacy Pipeline



Architecture with New Pipeline

Spark Development

- Spark 1.6.0 on Cloudera 5.7.x
- Directly on RDDs with Scala/Java
  - –control with lower level operators
  - –easy for unit testing
  - –strongly typed case classes
- Why not DataSet/SparkSQL?
  - –interested, waiting for stable Spark 2.0

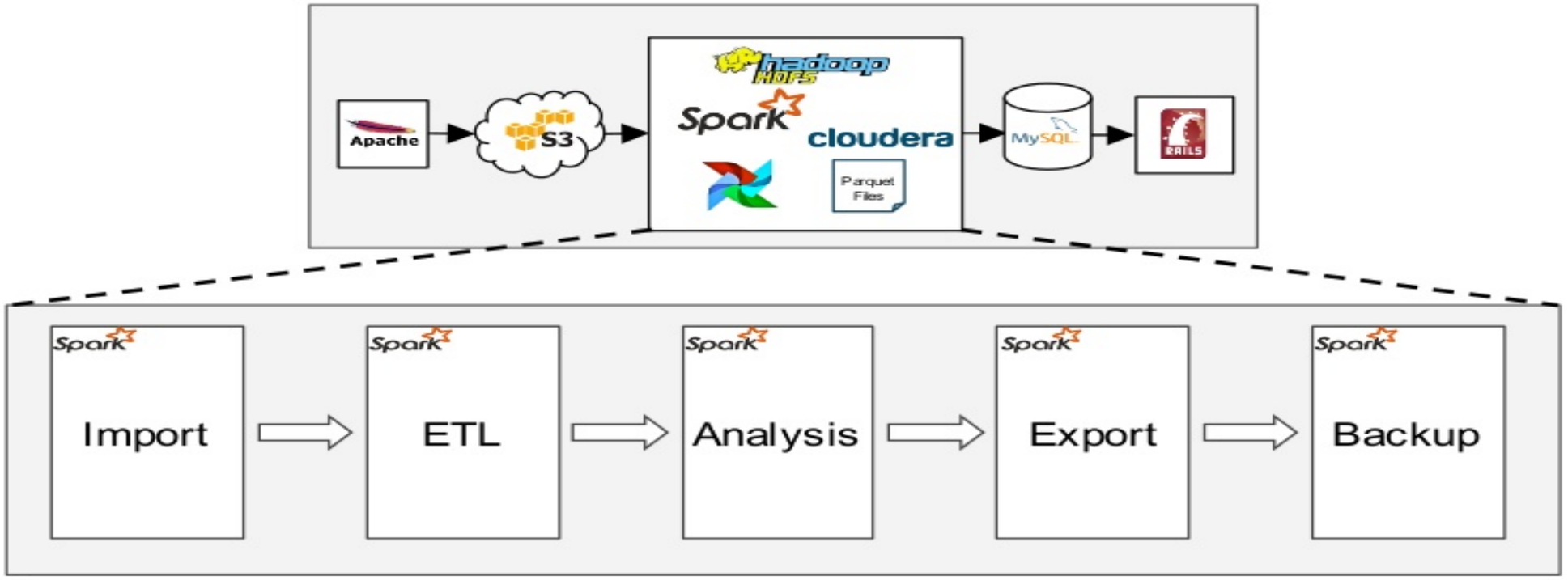
Development Approach

- What?
  - Fast Iterations
- Why?
  - Catch bugs and integration issues early
  - Scaling and Performance tuning
  - Feedback

Development Approach

- How?
  - Unit and Parity Tests
  - E2E pipeline in Dev environment
  - Staging environment
    - realistic cluster size and workload
    - use production data early
    - run repeatable tests - consistent and stable environment

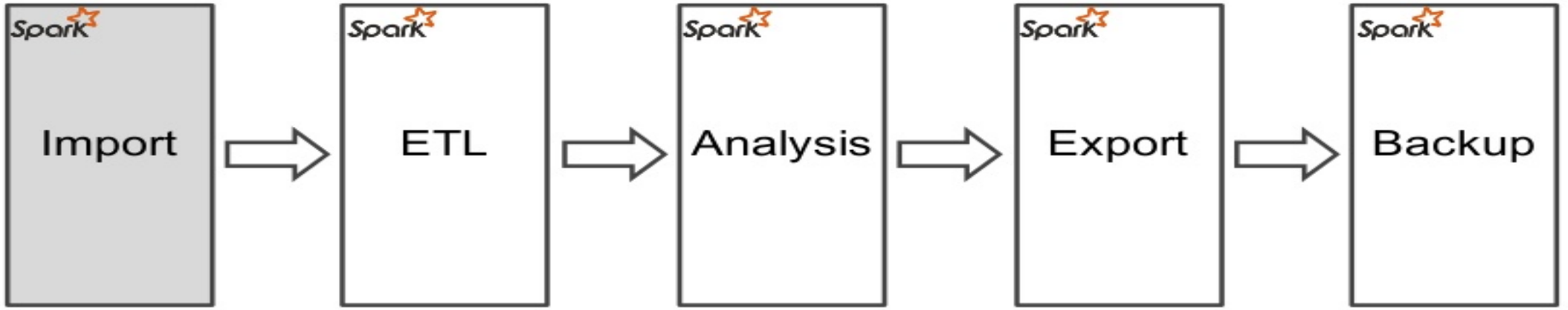
Analytics Data Pipeline



Pipeline Principle

- What?
  - Idempotent
- Why?
  - Fault Tolerance - retry at run and stage level
  - Backtrack and start from older run
- How?
  - Immutable input and output
  - Parquet for persistent read-only data

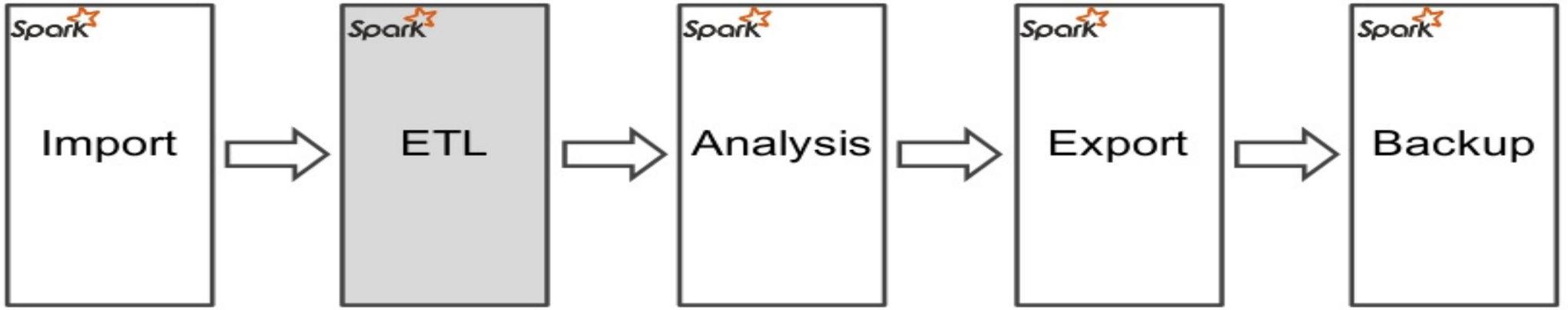
Import


Spark
Import
Spark
ETL
Spark
Analysis
Spark
Export
Spark
Backup

Import

- What?
  - Snapshot data needed for the pipeline run
  - Example: metadata stored in MySQL
- Why?
  - Ensure idempotence on retries
- How?
  - Example: use Spark JDBC to read from MySQL
    - sqlContext.read.format("jdbc").options(jdbcOptions).load()
      - on large datasets use: “partitionColumn”
      - do the filtering on the DataFrame side when using partitionColumn (dbtable->TABLE_NAME not dbtable->SELECT *… WHERE...)

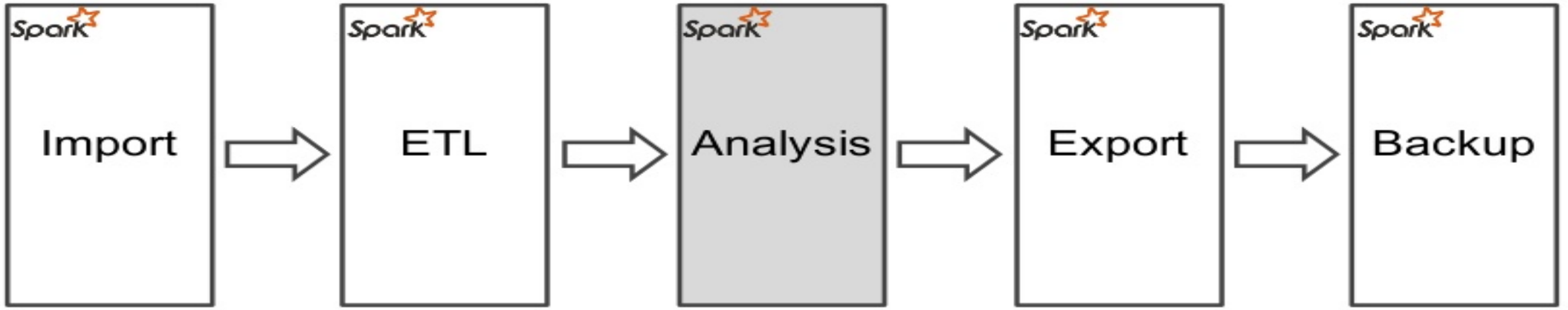
ETL


Spark
Import
Spark
ETL
Spark
Analysis
Spark
Export
Spark
Backup

ETL

- What?
  - Sanitize, transforming, grouping of data
- Why?
  - Consistency and quality of data for subsequent stages
- How?
  - Converting input JSON (json4s) to Case Classes
  - Transformations (RDD)
  - Save to Parquet (SparkSQL)
- Tips:
  - Avoid Nulls
  - Repartition output RDDs to remove skews

Analysis


Spark
Import
Spark
ETL
Spark
Analysis
Spark
Export
Spark
Backup

Analysis

- What?
  - Computing user level and aggregate metrics
- Why?
  - Metrics consume by front-end or internally
- How?
  - Single Spark job
    - Advantage: reuse of cached RDDs
    - Disadvantage: complex DAGs, recompute tree on failure, cached RDDs spill to disks

Analysis

- Handling Skews
  - Prefer Union-ReduceByKey over Join-Aggregate pattern
  - Broadcast variables for Map Join
  - Skew-Join (by Tresata)
- Reduce shuffles
  - Partition aware ops
  - Piggyback repartition with prior Shuffle operations reduceByKey()

Analysis

- Partition memory size
  - Fewer large partitions can cause OOM
  - Too many small partitions has high shuffle overhead cost
  - Collect and use dataset statistics:
    - Control partition sizes via controlling number of partitions
    - Adjust partition sizes that works for your Executor memory allocation
    - As data grows, scale with more number of partitions and executors

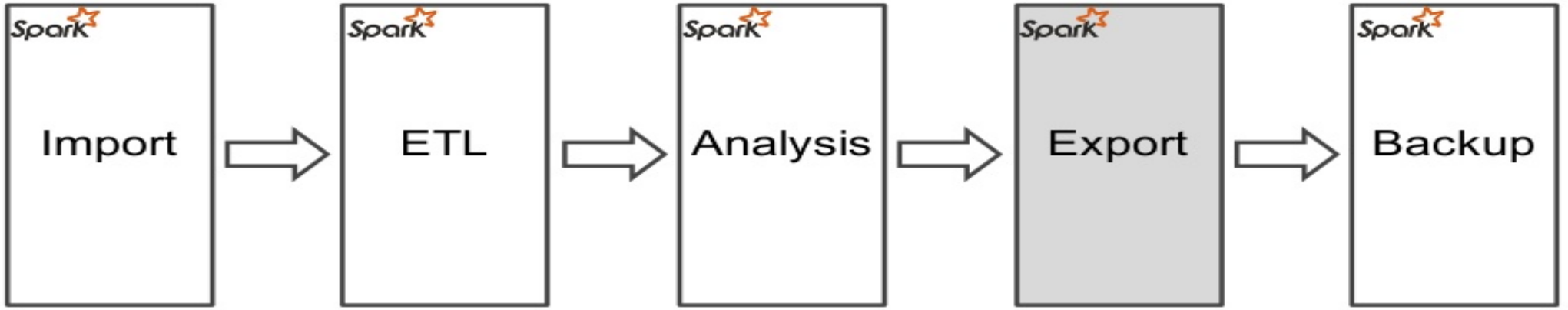
Analysis

- Partitioner matters
  - Resulting number of partitions on Joins depends on input RDD's partitioners
    - Joins mostly use the DefaultPartitioner Logic
      - Takes partitioner with largest number
      - if no partitioner, takes the largest RDD partitioner number
  - RDD has partitions but not necessarily a partitioner
    - Reading from parquet does not give a partitioner
    - Some operations remove the partitioner e.g map, keyBy

Hive Differences

- Memory:
  - Spark RDD operations are sensitive to partition size and often hit OOM
  - Hive/MapReduce can run with larger reducer dataset but will be slow
- Skew Handling Joins
  - Spark has Broadcast Join
  - Hive has bucket map join and other special skew handling Joins

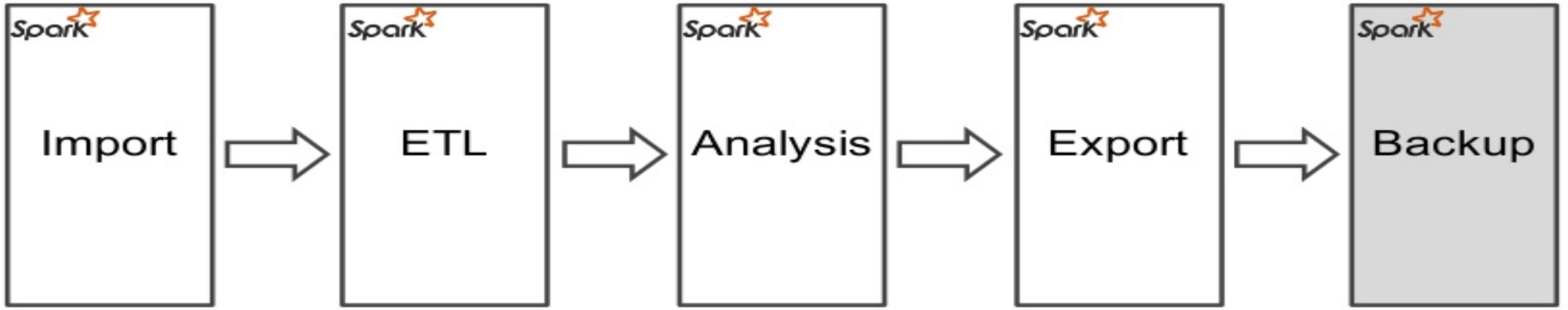
Export


Spark
Import
Spark
ETL
Spark
Analysis
Spark
Export
Spark
Backup

Export

- What?
  - exporting computed metrics to external store e.g. MySQL
- Why?
  - make it easy to consume analytics data
- How?
  - LOAD INFILE command with CSV
  - Batch to larger CSV files (coalesce and spark-csv)
  - Sort by Primary Key
  - Parallelize

Backup


Spark
Import
Spark
ETL
Spark
Analysis
Spark
Export
Spark
Backup

Backup

- What?
  - Backing up processed data to S3 (Parquet-S3-Backup)
- Why?
  - HDFS for co-located working dataset
  - S3 for long term backup
- How?
  - Coalesce parquet files locally before saving
  - Use s3a:// uri
  - Use DirectParquetOutputCommitter to skip writing to temp folder
    - Default requires a S3 copy and delete (SLOW)
    - Set "`spark.speculation=false`" to prevent data loss
    - Deprecated* in Spark 2.0, use "mapreduce.fileoutputcommitter. algorithm.version, 2" instead for Spark 2.0+

Airflow

- What?
  - Pipeline workflow management
  - Schedule and backfill
- Why?
  - Simple to setup and easy to use
  - Fits our short/mid term needs
- How?
  - BashOperators, PythonOperators, MySqlOperators
  - spark-submit

Hive Data Migration

- One-off Spark Job
- Read Legacy Hive tables in S3 using ORC format
- Spark can load hive table directly

```
hiveContext.sql("CREATE EXTERNAL TABLE …
                    LOCATION 's3a://…'")
val dataframe = hiveContext.sql("select * from
TABLE_NAME")
```

- Validate, Sanitize, Transform (like ETL)

Results

- Highlights our success with migrating to new architecture using Spark
- Not representative of Hive vs Spark
  - we made optimizations along the way
- Both systems have same total EC2 instances cost
  - Different instance types

	Old	New	New (2x workers)
Complete Run	12 hrs	2.5 hrs	1.5 hrs
Analysis Stage	8 hrs	2 hrs	1.25 hrs

Links

- Block/Skew Join
  - –https://github.com/tresata/spark-skewjoin
- Parquet S3 Backup
  - –https://github.com/UnityTech/parquet-s3-backup
- MySQL Uploader
  - –https://github.com/UnityTech/mysql-uploader

THANK YOU.

williaml@unity3d.com
kentb@unity3d.com