THE MECHANICS OF TESTING LARGE DATA PIPELINES

MATHIEU BASTIAN

Head of Data Engineering, GetYourGuide

@mathieubastian

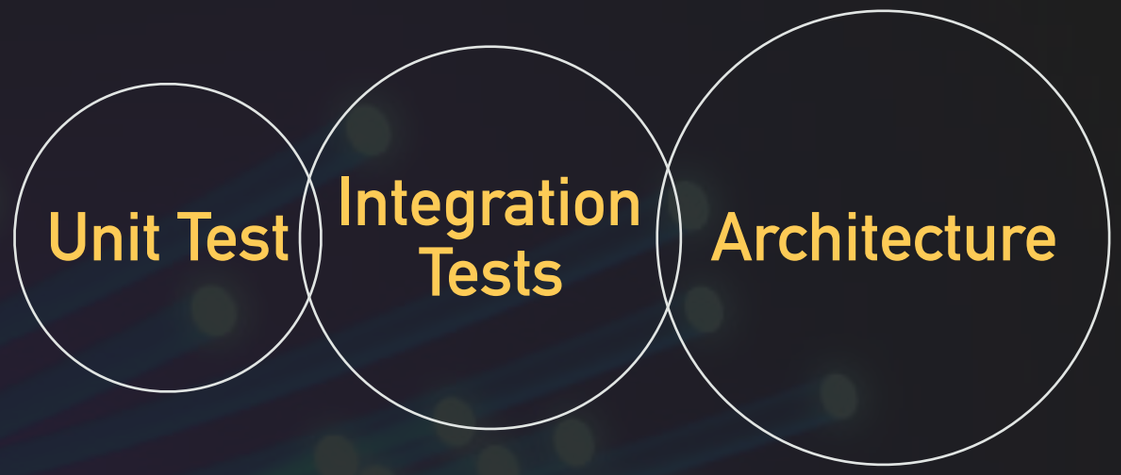
www.linkedin.com/in/mathieubastian

QCon LONDON

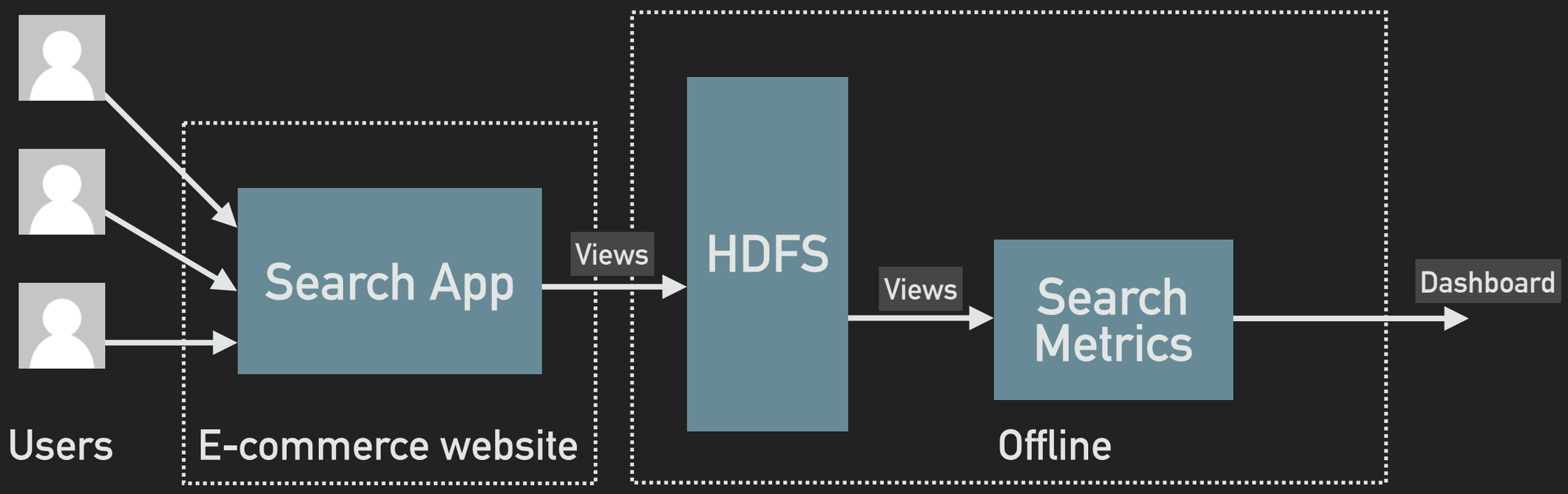
QCon London 2015

Outline

- Motivating example
- Challenges
- Testing strategies
- Validation Strategies
- Tools



Data Pipelines often start simple



They have one use-case and one developer

Recommender Systems

Topic Detection

Customer Churn Prediction

Sentiment Analysis

Anomaly Detection

Query Expansion

A/B Testing

Trending Tags

Standardization

Search Ranking

Signal Processing

Machine Translation

Sentiment Analysis

Content Curation

Fraud Prediction

Image recognition

Spam Detection

Bidding Prediction

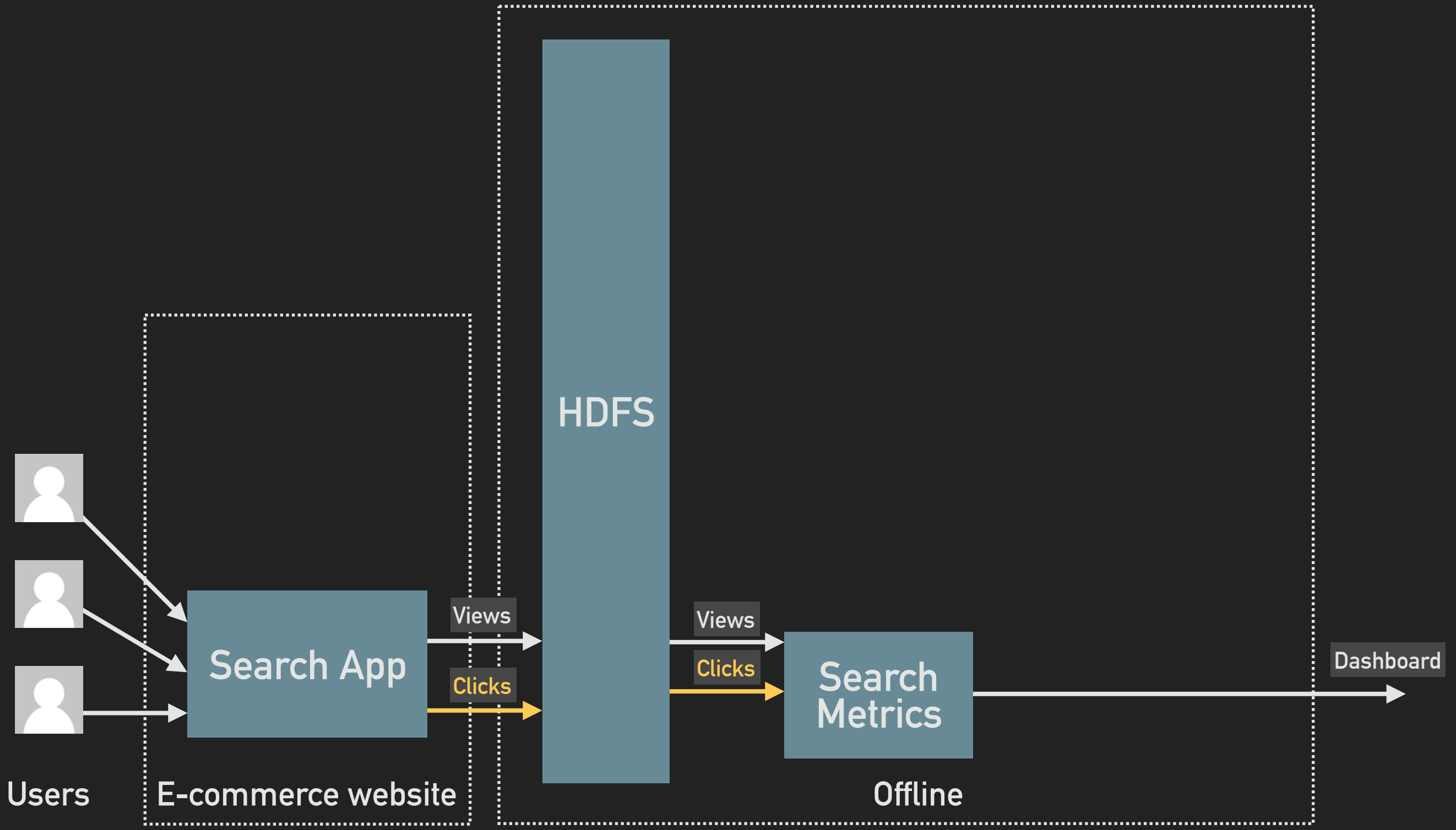
Optimal pricing

Funnel Analysis

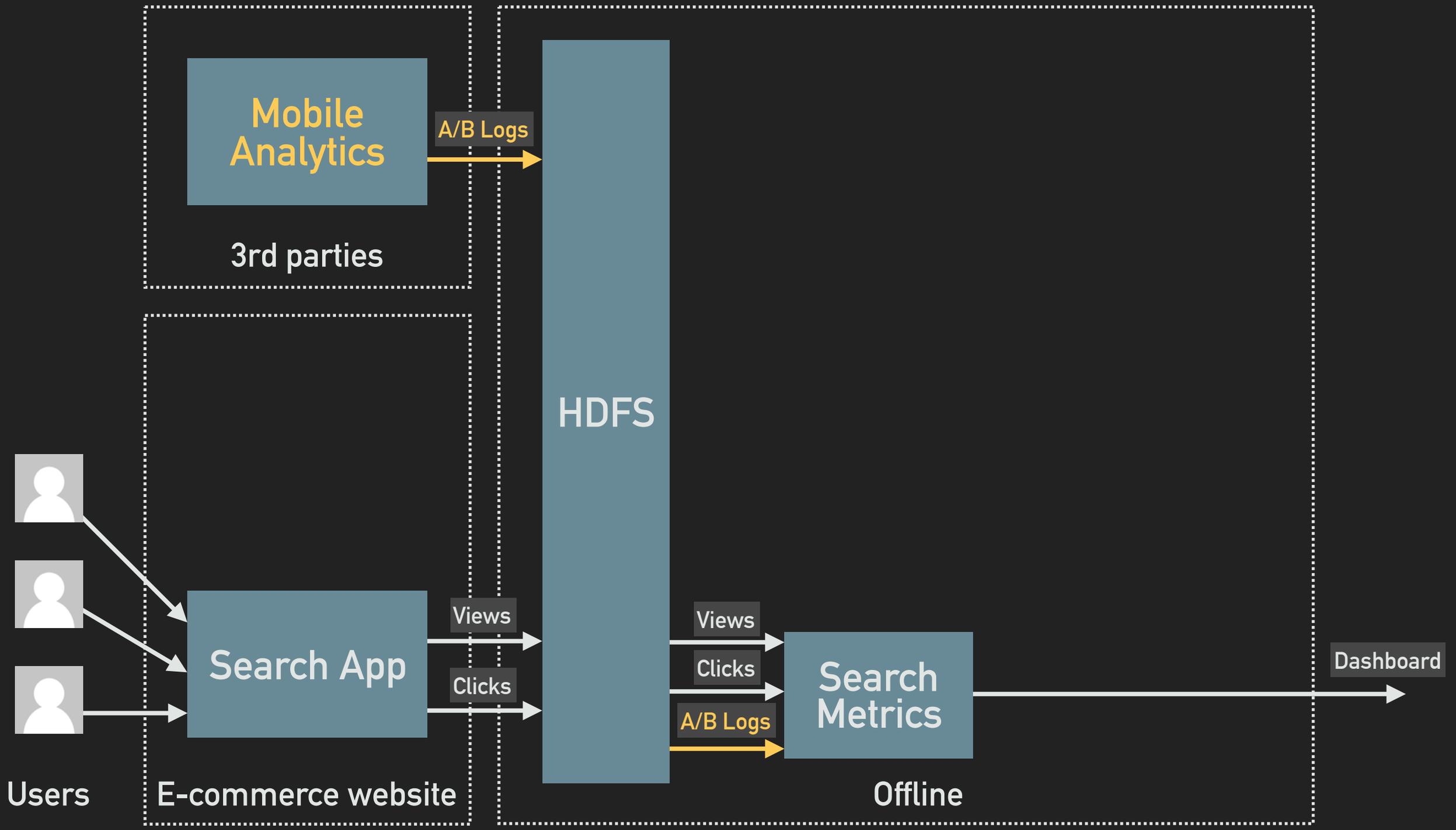
Location normalization

Related searches

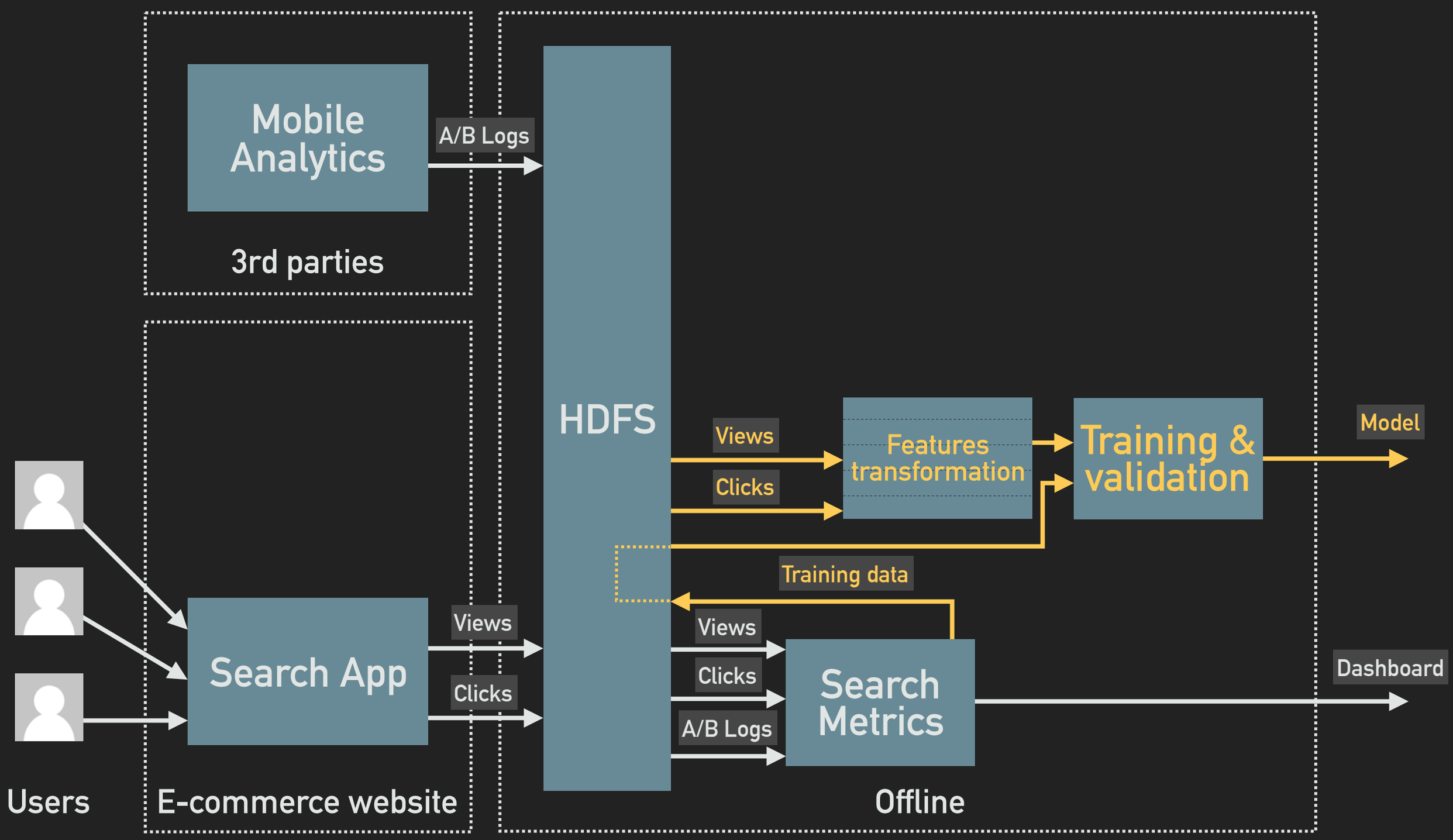
But there are many other use-cases



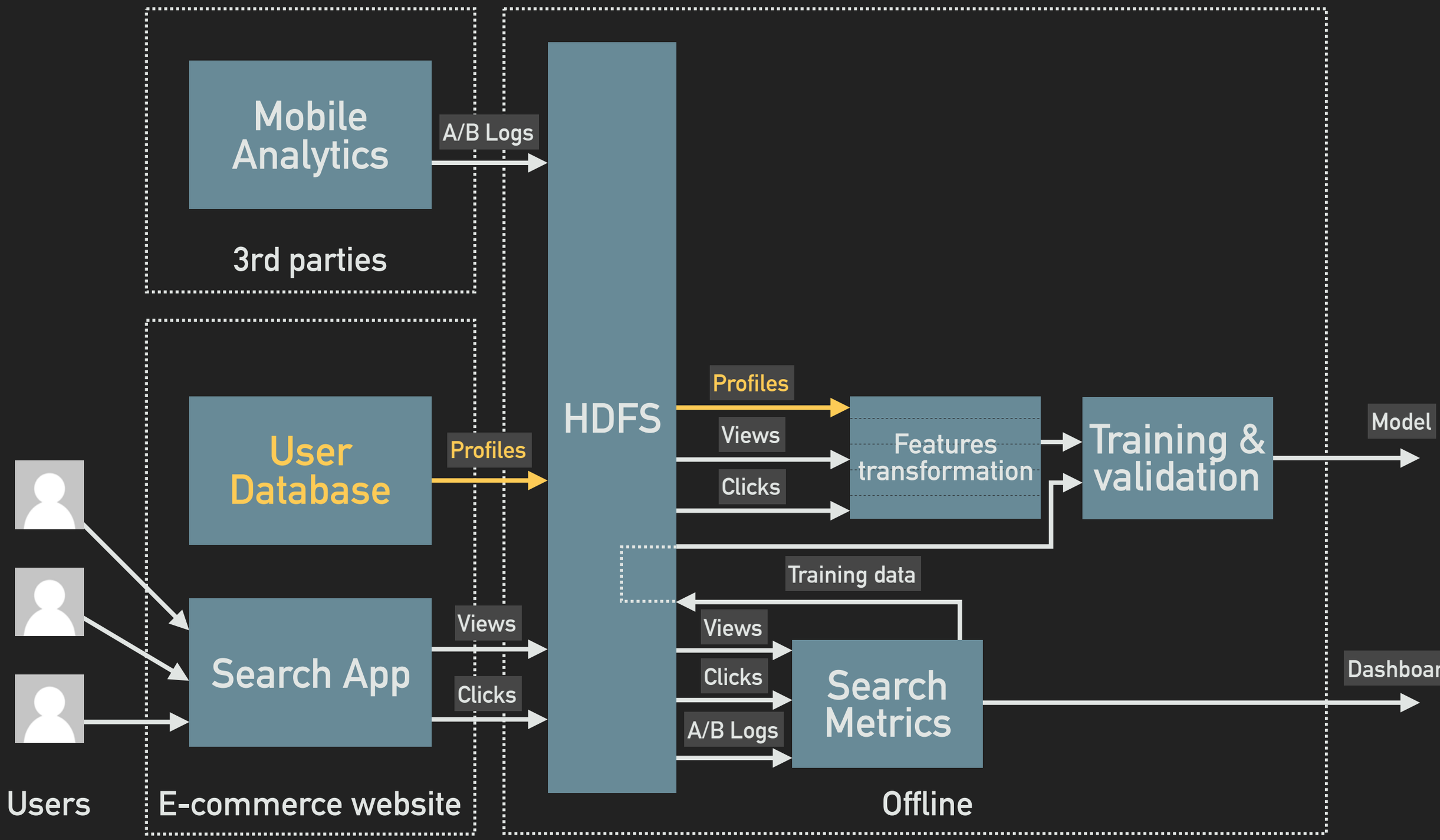
Developers add additional events and logs



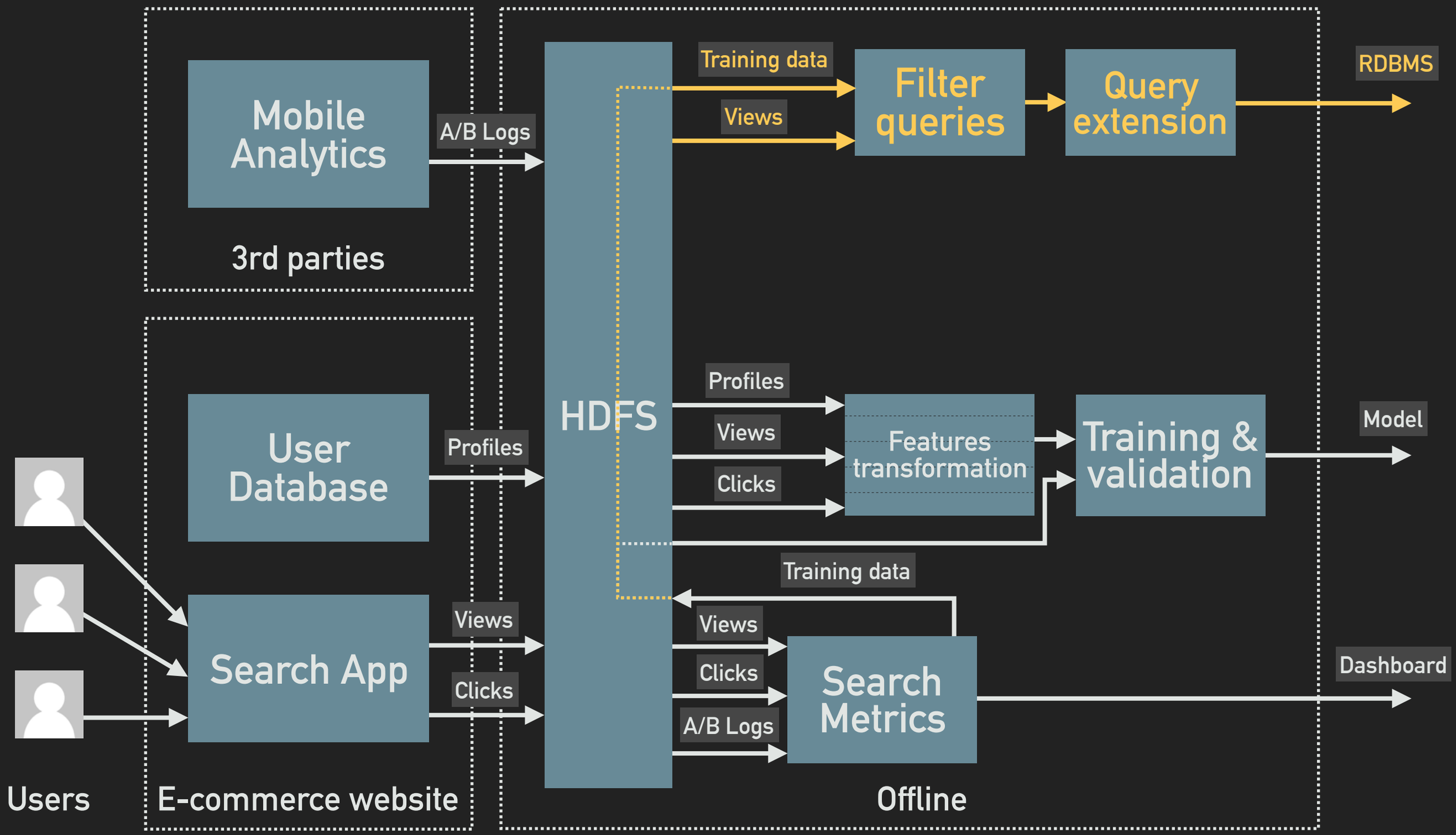
Developers add third-party data



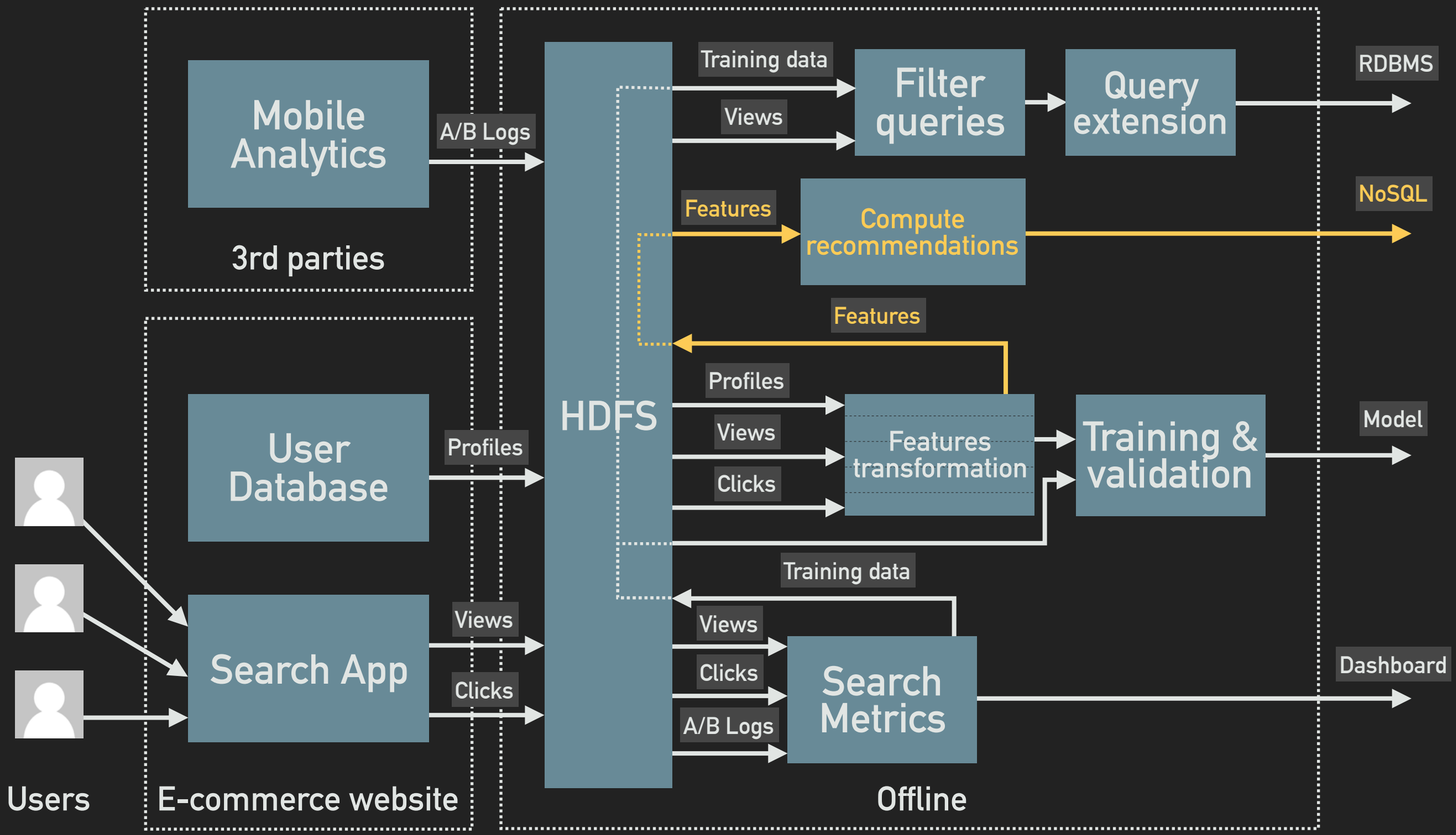
Developers add search ranking prediction



Developers add personalized user features



Developers add query extension



Developers add recommender system

Data Pipelines can grow very large

That is a lot of code and data

Code contain bugs

Industry Average: about 15 - 50 errors per 1000 lines of delivered code.

Data will change

Industry Average: ?

Embrace automated

testing of code

validation of data

Because it delivers

- Testing
  - Tested code has less bugs
  - Gives the confidence to iterate quickly
  - Scales well to multiple developers
- Validation
  - Reduce manual testing
  - Avoid catastrophic failures

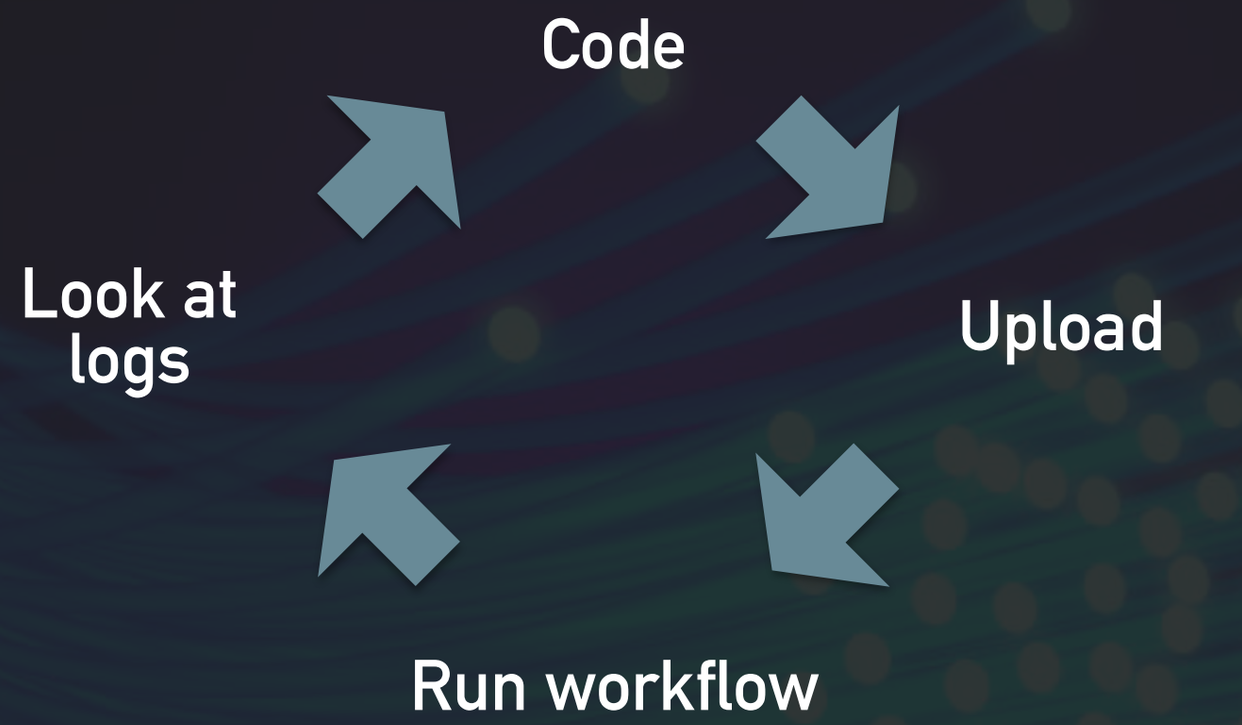
But it's challenging

- Testing
  - Need data to test "realistically"
  - Not running locally, can be expensive
  - Tooling weaknesses
- Validation
  - Data sources out of our control
  - Difficult to test machine learning models

Reality check

Source: @SteveGodwin, QCon London 2016

Manual testing



- Time Spent

Waiting	Coding	Looking at logs

Testing strategies

Prepare environment

- Care about tests from the start of your project
- All jobs should be functions (output only depends on input)
- Safe to re-run the job
  - Does the input data still exists?
  - Would it push partial results?
- Centralize configurations and no hard-coded paths
- Version code and timestamp data

Unit test locally

- Test locally each individual job
  - Tests its good code
  - Tests expected failures
- Need to overcome challenges with fake data creation
  - Complex structures and numerous data sources
  - Too small to be meaningful
- Need to specify a different configuration

Build from schemas

Fake data creation based on schemas. Compare:

```
Customer c = Customer.newBuilder().
   setId(42).
   setInterests(Arrays.asList(new Interest[]{
      Interest.newBuilder().setId(0).setName("Ping-Pong").build()
      Interest.newBuilder().setId(1).setName(“Pizza").build()}))
   .build();
```

vs

```
Map<String, Object> c = new HashMap<>();
c.put("id", 42);
Map<String, Object> i1 = new HashMap<>();
i1.put("id", 0);
i1.put("name", "Ping-Pong");
Map<String, Object> i2 = new HashMap<>();
i2.put("id", 1);
i2.put("name", "Pizza");
c.put("interests", Arrays.asList(new Map[] {i1, i2}));
```

Build from schemas

Avro Schema example


```
{
    "type": "record",
    "name": "Customer",
    "fields": [{
            "name": "id",
            "type": "int"
        }, {
            "name": "interests",
            "type": {
                "type": "array",
                "items": {
                    "name": "Interest",
                    "type": "record",
                    "fields": [{
                        "name": "id",
                        "type": "int"
                    }, {
                        "name": "name",
                        "type": ["string", "null"]
                    }]
                }
            }
        }
    ]
}
```

nullable field

Complex generators

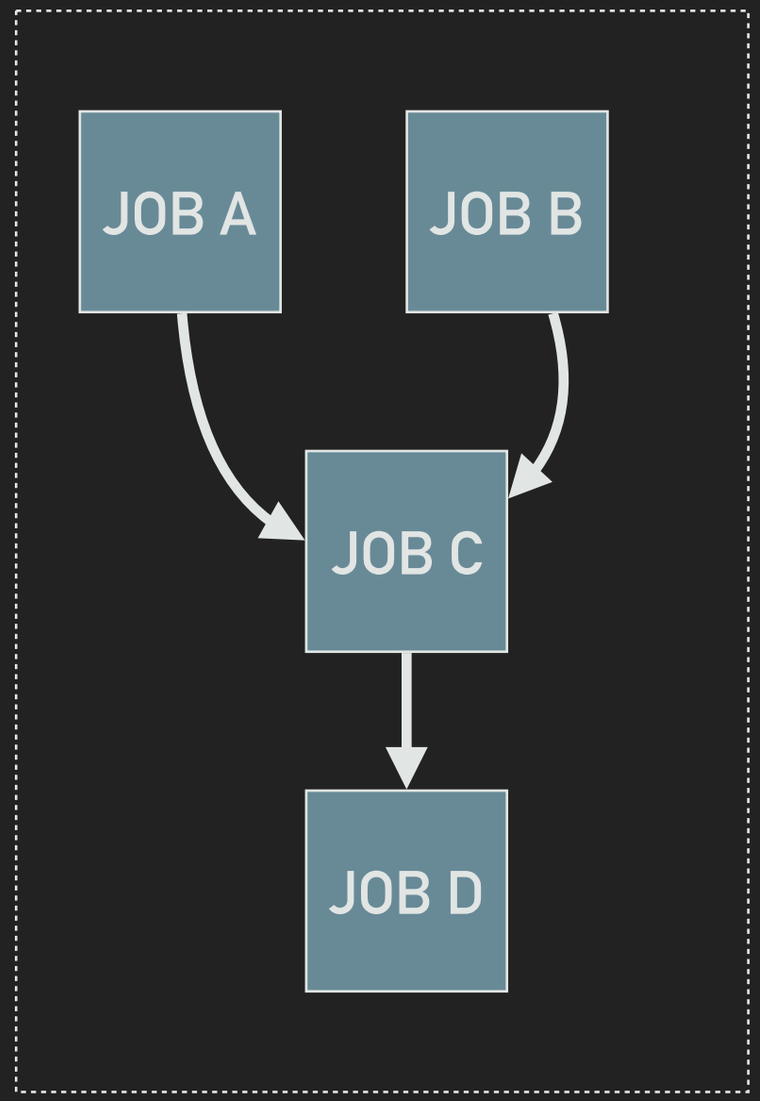
- Developed in the field of property-based testing

```
//Small Even Number Generator
val smallEvenInteger = Gen.choose(0,200) suchThat (_ % 2 == 0)
```

- Goal is to simulate, not sample real data
- Define complex random generators that match properties (e.g. frequency)
- Can go beyond unit-testing and generate complex domain models
- https://www.scalacheck.org/ for Scala/Java is a good starting point for examples

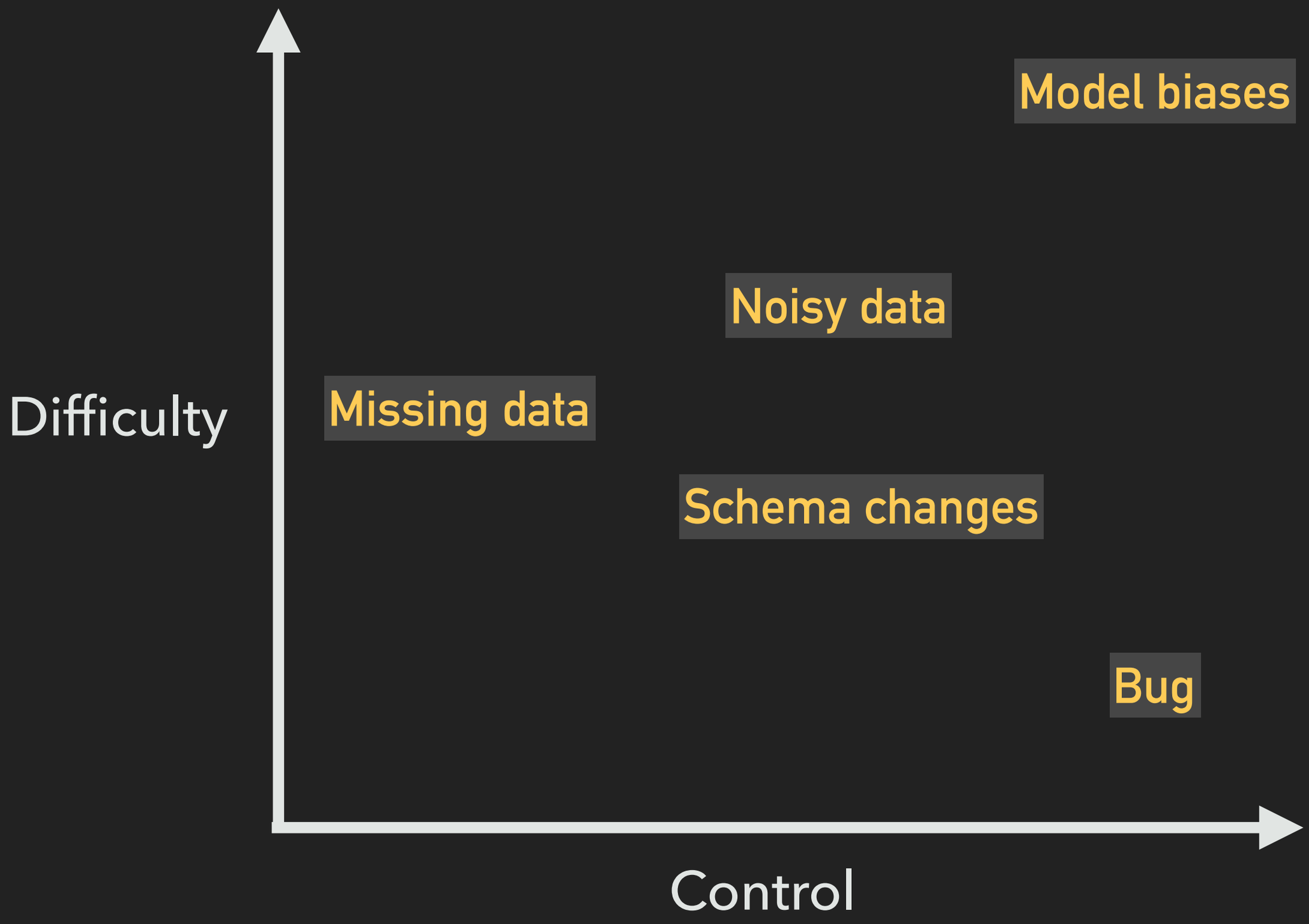
Integration test on sample data

- Integration test the entire workflow
  - File paths
  - Configuration
  - Evaluate performance
- Sample data
  - Large enough to be meaningful
  - Small enough to speed-up testing



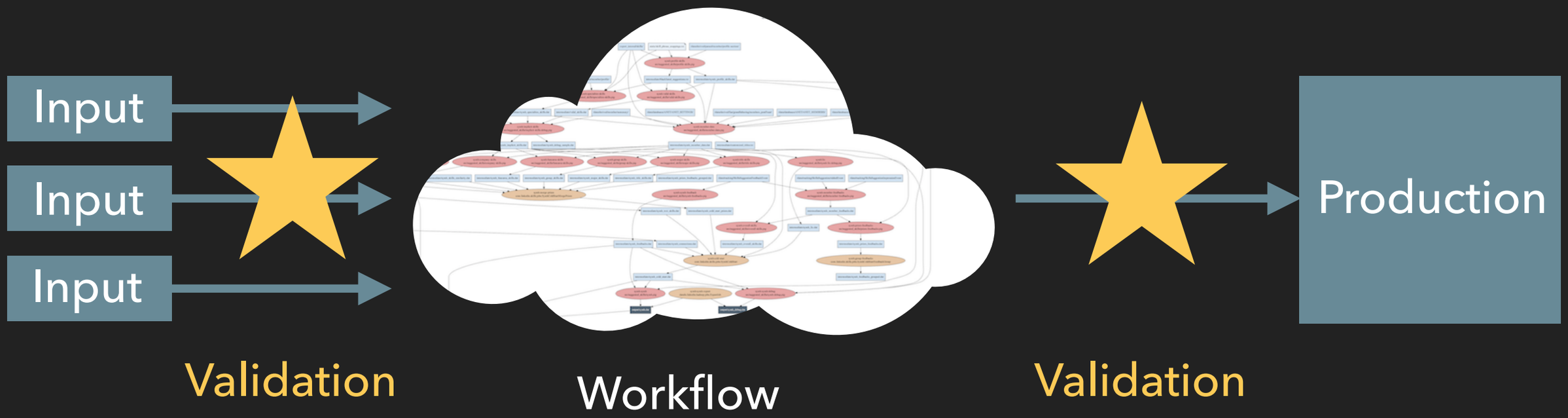
Validation strategies

Where it fail



Input and output validation

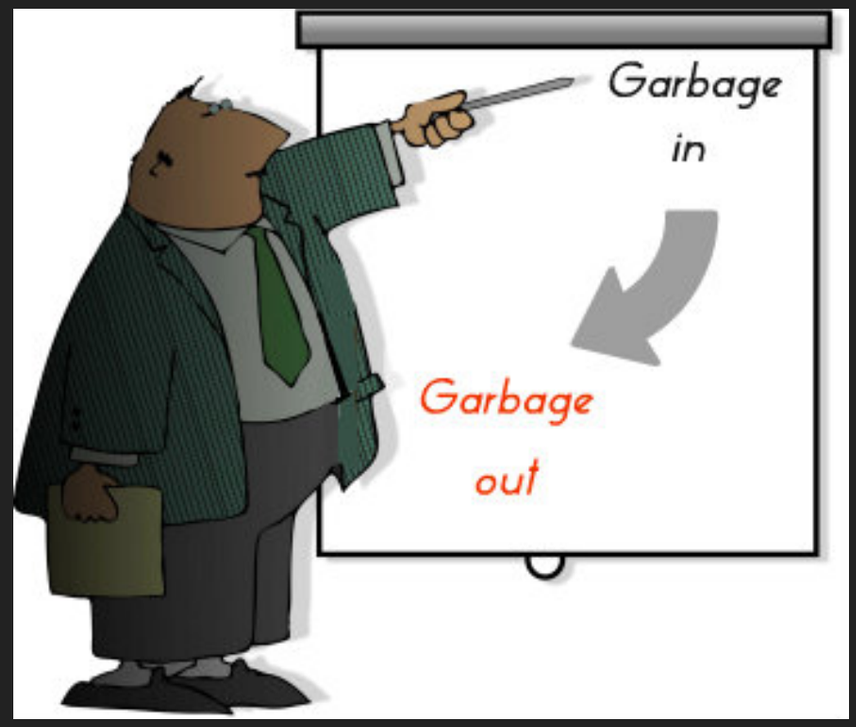
Make the pipeline robust by validating inputs and outputs



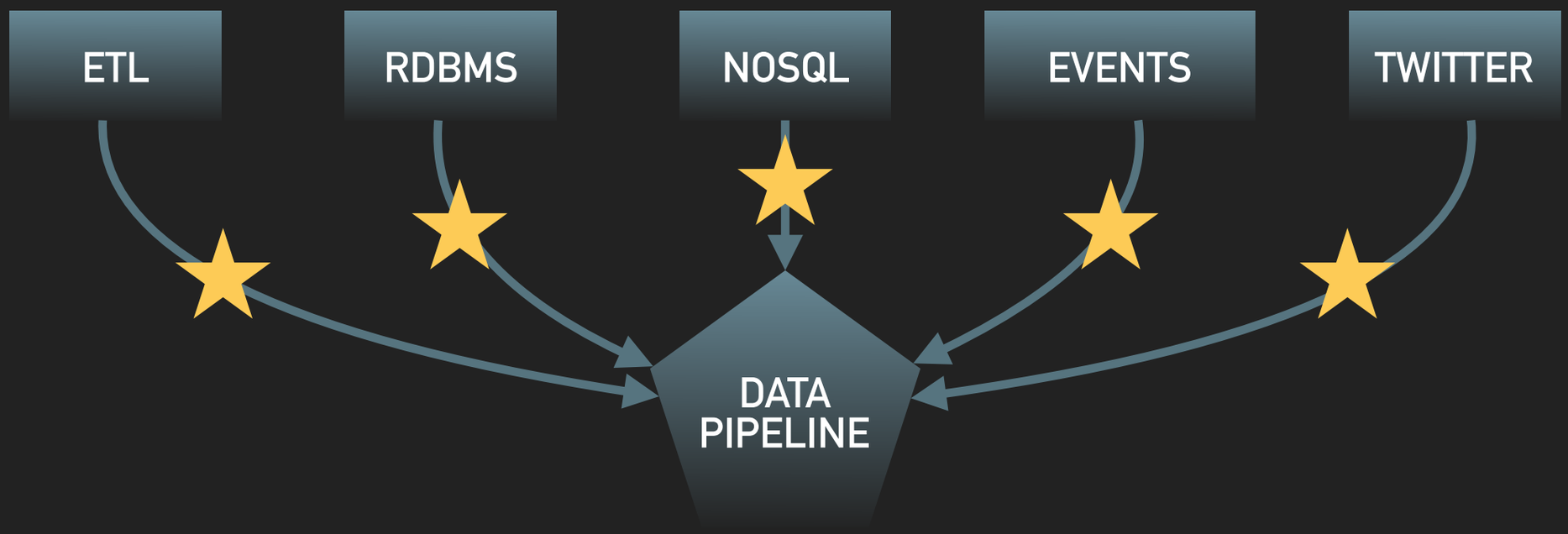
Input Validation

Input data validation

Input data validation is a key component of pipeline robustness.



The goal is to test the entry points of our system for data quality.



Why it matters

- Bad input data will most likely degrade the output
- It likely will fail silently

- Because data will change
  - Data migrations: maintenance, cluster update, new infrastructure
  - Events change due to product evolution
  - Data dependencies updated

Input data validation

- Validation code should
  - Detect pathological data and fail early
  - Deal with expected data variability
- Example issues:
  - Missing values, encoding issues, etc.
  - Schema changes
  - Duplicates rows
  - Data order changes

Pathological data

- Value
  - Validity depends on a single, independent value.
  - Easy to validate on streams of data

- Dataset
  - Validity depends on the entire dataset
  - More difficult to validate as it needs a window of data

Metadata validation

Analyzing metadata is the quickest way to validate input data

- Number of records and file sizes
- Hadoop/Spark counters
  - Number of map/reduce records, size
- Record-level custom counters
  - Average text length
- Task-level custom counters
  - Min/Max/Median values

Hadoop/Spark counters

Results can be accessed programmatically and checked

Counters for job_1419937768824_0002

Counters			
Name	Map	Reduce	Total
FILE: Number of bytes read	0	0	0
FILE: Number of bytes written	1110540	0	1110540
FILE: Number of large read operations	0	0	0
FILE: Number of read operations	0	0	0
FILE: Number of write operations	0	0	0
HDFS: Number of bytes read	44129667	0	44129667
HDFS: Number of bytes written	0	0	0
HDFS: Number of large read operations	0	0	0
HDFS: Number of read operations	97	0	97
HDFS: Number of write operations	20	0	20
S3N: Number of bytes read	0	0	0
S3N: Number of bytes written	44123904	0	44123904
S3N: Number of large read operations	0	0	0
S3N: Number of read operations	0	0	0
S3N: Number of write operations	0	0	0

Name	Map	Reduce	Total
Launched map tasks	0	0	10
Other local map tasks	0	0	10
Total megabyte-seconds taken by all map tasks	0	0	134645760
Total time spent by all map tasks (ms)	0	0	131490
Total time spent by all maps in occupied slots (ms)	0	0	131490
Total vcore-seconds taken by all map tasks	0	0	131490

Name	Map	Reduce	Total
CPU time spent (ms)	23840	0	23840
Failed Shuffles	0	0	0
GC time elapsed (ms)	6263	0	6263
Input split bytes	1340	0	1340
Map input records	10	0	10
Map output records	0	0	0
Merged Map outputs	0	0	0
Physical memory (bytes) snapshot	1677352960	0	1677352960
Spilled Records	0	0	0
Total committed heap usage (bytes)	1164443648	0	1164443648
Virtual memory (bytes) snapshot	21159137280	0	21159137280

Name	Map	Reduce	Total
Bytes Read	4423	0	4423

Name	Map	Reduce	Total
Bytes Written	0	0	0

Name	Map	Reduce	Total
BYTESCOPIED	44123904	0	44123904
BYTESEXPECTED	44123904	0	44123904
COPY	10	0	10

Control inputs with Schemas

- CSVs aren’t robust to change, use Schemas
- Makes expected data explicit and easy to test against
- Gives basic validation for free with binary serialization (e.g. Avro, Thrift, Protocol Buffer)
    - Typed (integer, boolean, lists etc.)
    - Specify if value is optional
- Schemas can be evolved without breaking compatibility

Output Validation

Why it matters

- Humans makes mistake, we need a safeguard
- Rolling back data is often complex
- Bad output propagates to downstream systems

Example with a recommender system

```
// One recommendation set per user
{
  "userId": 42,
  "recommendations": [{
    "itemId": 1456,
    "score": 0.9
  }, {
    "itemId": 4232,
    "score": 0.1
  }],
  "model": "test01"
}
```

Check for anomalies

Simple strategies similar to input data validation

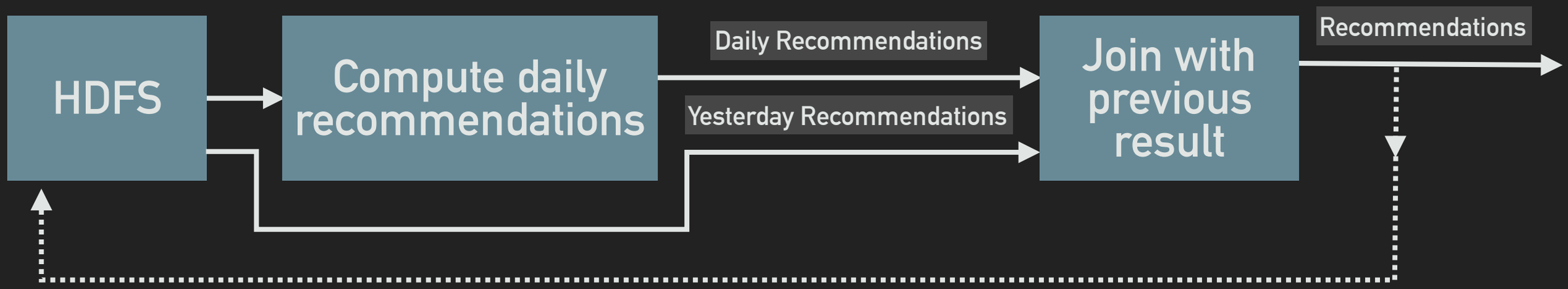
- Record level (e.g. values within bounds)
- Dataset level (e.g. counts, order)

Challenges around relevance evaluation

- When supervised, use a validation dataset and threshold accuracy
- Introduce hypothetical examples

Incremental update as validation

Join with the previous “best" output



- Allows fine comparisons
- Incremental framework can be extended to
  - Only recompute recommendations that have changed
  - Produce variations metric between different models

External validation

Even in automated environment it is possible to validate with humans

- Example: Search ranking evaluation

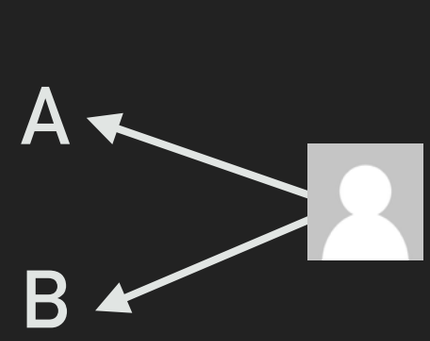
- Solution: Crowdsourcing
  - Complex validation that requires training
  - Can be automated through APIs

Mitigate risk with A/B testing

Gradually rolling out data products improvements reduces the need for complex output validation

- Experiment can be controlled online or offline
  - Online: Push multiple set of recommendations (1 per model)

```
userId -> [{
            "model": "test01",
            "recommendations": [{...}]
          }, {
            "model": "test02",
            "recommendations": [{...}]
          }]
```



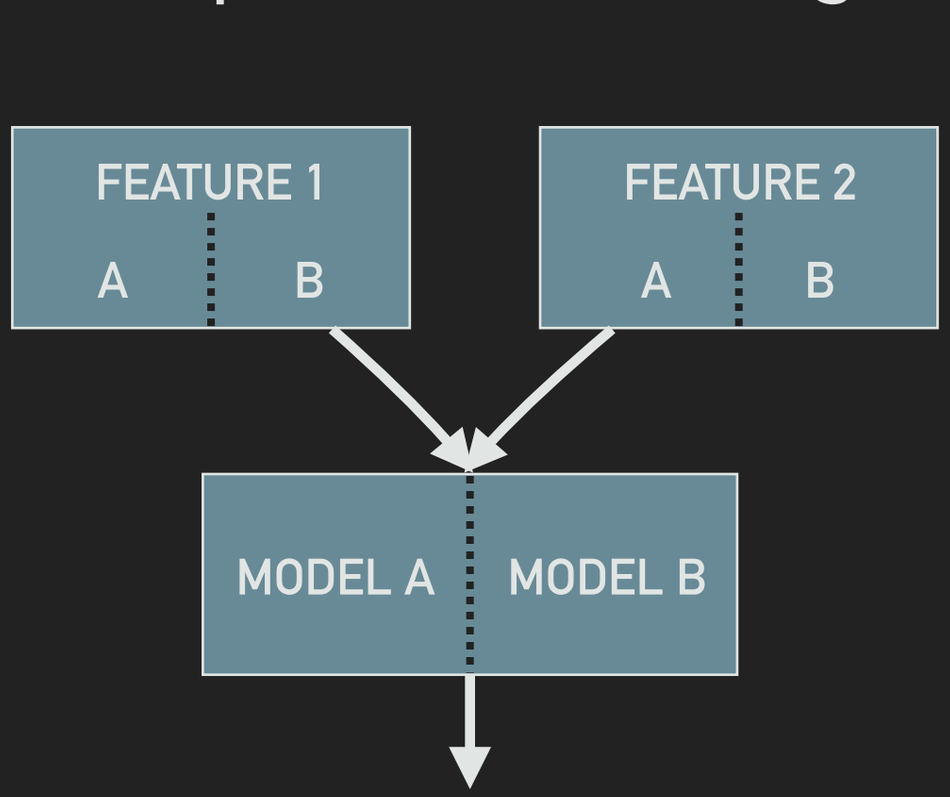
  - Offline: Split users and push unique set of recommendations

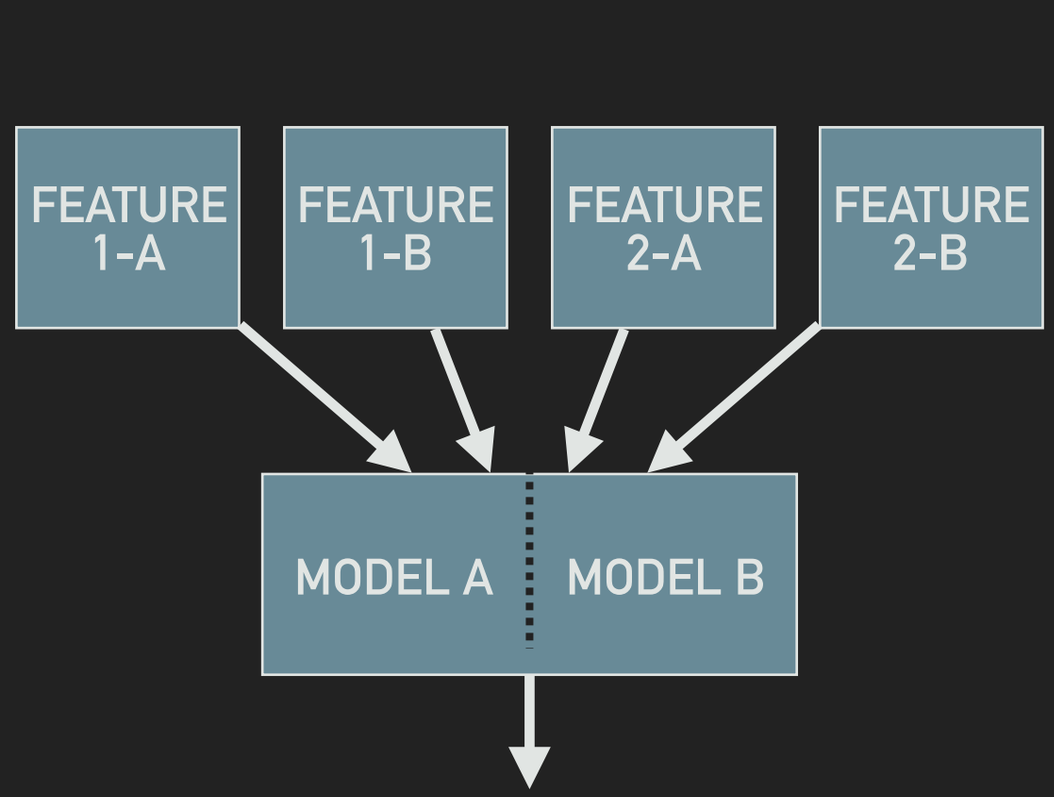
```
userId -> {
            "model": "test01",
            "recommendations": [{...}]
          }
```

Mitigate risk with A/B testing

Important

- Log model variation downstream in logs
- Encapsulate model logic





Thank You!

Appendix

Hadoop Testing

Two ways to test Hadoop jobs

- MRUnit
  - Java library to test MapReduce jobs in a simulated environment
  - Last release June 2014

- MiniCluster
  - Utility to locally run a fully-functional Hadoop cluster in a test environment
  - Ships with Hadoop itself

MiniMRCluster

- Advantages
  - Behaves like a real cluster, including setup and configuration
  - Can be used to test multiple jobs (integration testing)

- Disadvantages
  - Very slow compared to unit testing Java code

MRUnit

- Advantages
  - Faster
  - Less boilerplate code

- Disadvantages
  - Need to replicate job configuration
  - Only built to test map and reduce functions
  - Difficult to make it work with custom input formats (e.g. Avro)

MiniMRCluster setup*

Setup MR cluster and obtain FileSystem

```
@BeforeClass
public void setup() {
  Configuration dfsConf = new Configuration();
  dfsConf.set(MiniDFSCluster.HDFS_MINIDFS_BASEDIR, new File("./target/
hdfs/").getAbsolutePath());
  _dfsCluster = new MiniDFSCluster.Builder(dfsConf).numDataNodes(1).build();
  _dfsCluster.waitClusterUp();
  _fileSystem = _dfsCluster.getFileSystem();
  YarnConfiguration yarnConf = new YarnConfiguration();
  yarnConf.setFloat(YarnConfiguration.NM_MAX_PER_DISK_UTILIZATION_PERCENTAGE,
99.0f);
  yarnConf.setInt(YarnConfiguration.RM_SCHEDULER_MINIMUM_ALLOCATION_MB, 64);
  yarnConf.setClass(YarnConfiguration.RM_SCHEDULER, FifoScheduler.class,
ResourceScheduler.class);
  _mrCluster = new MiniMRYarnCluster(getClass().getName(), taskTrackers);
  yarnConf.set("fs.defaultFS", _fileSystem.getUri().toString());
  _mrCluster.init(yarnConf);
  _mrCluster.start();
}
```

* Hadoop version used 2.7.2

Keep the test file clean of boilerplate code

Best is to wrap the start/stop code into a *TestBase* class

```
/**
 * Default constructor with one task tracker and one node.
 */
public TestBase() { ... }

@BeforeClass
public void startCluster() throws IOException { ... }

@AfterClass
public void stopCluster() throws IOException { ... }

/**
 * Returns the Filesystem in use.
 *
 * @return the filesystem used by Hadoop.
 */
protected FileSystem getFileSystem() {
  return _fileSystem;
}
```

Initialize and clean HDFS before/after each test

Clean up and initialize file system before each test

```
private final Path _inputPath = new Path("/input");
private final Path _cachePath = new Path("/cache");
private final Path _outputPath = new Path("/output");

@BeforeMethod
public void beforeMethod(Method method) throws IOException {
  getFileSystem().delete(_inputPath, true);
  getFileSystem().mkdirs(_inputPath);
  getFileSystem().delete(_cachePath, true);
}

@AfterMethod
public void afterMethod(Method method) throws IOException {
  getFileSystem().delete(_inputPath, true);
  getFileSystem().delete(_cachePath, true);
  getFileSystem().delete(_outputPath, true);
}
```

Run MiniCluster Test

Clean up and initialize file system before each test

```
@Test
public void testBasicWordCountJob() throws IOException, InterruptedException,
ClassNotFoundException {
  writeWordCountInput();
  configureAndRunJob(new BasicWordCountJob(), "BasicWordCountJob", _inputPath,
_outputPath);
  checkWordCountOutput();
}

private void configureAndRunJob(AbstractJob job, String name, Path inputPath,
Path outputPath) throws IOException, ClassNotFoundException,
InterruptedException {
  Properties _props = new Properties();
  _props.setProperty("input.path", inputPath.toString());
  _props.setProperty("output.path", outputPath.toString());
  job.setProperties(_props);
  job.setName(name);
  job.run();
}
```

MRUnit setup

Setup MapDriver and ReduceDriver

```
BasicWordCountJob.Map mapper;
BasicWordCountJob.Reduce reducer;
MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;
ReduceDriver<Text, IntWritable, Text, IntWritable> reduceDriver;

@BeforeClass
public void setup() {
  mapper = new BasicWordCountJob.Map();
  mapDriver = MapDriver.newMapDriver(mapper);
  reducer = new BasicWordCountJob.Reduce();
  reduceDriver = ReduceDriver.newReduceDriver(reducer);
}
```

Run MRUnit test

Set Input/Output and run Test

```
@Test
public void testMapper() throws IOException {
  mapDriver.withInput(new LongWritable(0), new Text("banana pear banana"));
  mapDriver.withOutput(new Text("banana"), new IntWritable(1));
  mapDriver.withOutput(new Text("pear"), new IntWritable(1));
  mapDriver.withOutput(new Text("banana"), new IntWritable(1));
  mapDriver.runTest();
}

@Test
public void testReducer() throws IOException {
  reduceDriver.withInput(new Text("banana"), Arrays.asList(new IntWritable(1),
new IntWritable(1)));
  reduceDriver.withInput(new Text("pear"), Arrays.asList(new IntWritable(1)));
  reduceDriver.withOutput(new Text("banana"), new IntWritable(2));
  reduceDriver.withOutput(new Text("pear"), new IntWritable(1));
  reduceDriver.runTest();
}
```

Most common pitfall

- With both MiniMRCluster and MRUnit one spend most of the time
  - Creating fake input data
  - Verifying output data

- Solutions
  - Use rich data structures format (e.g. Avro, Thrift)
  - Use automated Java classes generation

Other common pitfalls

- MiniMRCluster
  - Enable Hadoop INFO logging so you can see real job failure causes
  - Beware of partitioning or sorting issues unrevealed when testing with too few rows and number of nodes
  - The API has changed over the years, difficult to find examples
- MRUnit
  - Custom serialization issues (e.g. Avro, Thrift)

Pig Testing

Introducing PigUnit

- PigUnit
  - Official library to unit tests Pig script
  - Ships with Pig (latest version 0.15.0)
  - The principle is easy
    1. Generate test data
    2. Run script with PigUnit
    3. Verify output
  - Runs locally but can be run on a cluster too

Script example

WordCount example

- Input and output are standard formats
- Uses variables *$input* and *$output*

```
text = LOAD '$input' USING TextLoader();

flattened = FOREACH text GENERATE flatten(TOKENIZE((chararray)$0)) as word;
grouped = GROUP flattened by word;
result = FOREACH grouped GENERATE group, (int)COUNT($1) AS cnt;
sorted = ORDER result BY cnt DESC;

STORE sorted INTO '$output' USING PigStorage('\t');
```

PigTestBase

Create PigTest object

```
protected final FileSystem _fileSystem;

protected PigTestBase() {
  System.setProperty("udf.import.list", StringUtils.join(Arrays.asList("oink.",
"org.apache.pig.piggybank."), ":"));
  fileSystem = FileSystem.get(new Configuration());
}
```

```
/**
 * Creates a new <em>PigTest</em> instance ready to be used.
 *
 * @param scriptFile the path to the Pig script file
 * @param inputs     the Pig arguments
 * @return new PigTest instance
 */
protected PigTest newPigTest(String scriptFile, String[] inputs) {
    PigServer pigServer = new PigServer(ExecType.LOCAL);
    Cluster pigCluster = new Cluster(pigServer.getPigContext());
    return new PigTest(scriptFile, inputs, pigServer, pigCluster);
}
```

Test using aliases

getAlias() allows to obtain the data anywhere in the script

```
@Test
public void testWordCountAlias() throws IOException, ParseException {
  //Write input data
  BufferedWriter writer = new BufferedWriter(new FileWriter(new
File("input.txt")));
  writer.write("banana pear banana");
  writer.close();

  PigTest t = newPigTest("pig/src/main/pig/wordcount_text.pig", new String[]
{"input=input.txt", "output=result.csv"});

  Iterator<Tuple> tuples = t.getAlias("sorted");
  Assert.assertTrue(tuples.hasNext());
  Tuple tuple = tuples.next();
  Assert.assertEquals(tuple.get(0), "banana");
  Assert.assertEquals(tuple.get(1), 2);
  Assert.assertTrue(tuples.hasNext());
  tuple = tuples.next();
  Assert.assertEquals(tuple.get(0), "pear");
  Assert.assertEquals(tuple.get(1), 1);
}
```

Test using mock and assert

- mockAlias allows to substitute input data
- assertOutput allows to compare String output data

```
@Test
public void testWordCountMock() throws IOException, ParseException {
  //Write input data
  BufferedWriter writer = new BufferedWriter(new FileWriter(new
File("input.txt")));
  writer.write("banana pear banana");
  writer.close();

  PigTest t = newPigTest("pig/src/main/pig/wordcount_text.pig", new String[]
{"input=input.txt", "output=null"});
  t.runScript();
  t.assertOutputAnyOrder("sorted", new String[]{"(banana,2)", "(pear,1)"});
}
```

Both of these tools have limitations

- Built around standard input and output (Text, CSVs etc.)
  - Realistically most of our data is in other formats (e.g. Avro, Thrift, JSON)
- Does not test the STORE function (e.g. schema errors)
- getAlias() is especially difficult to use
  - Need to remember field position: tuple.get(0)
- assertOutput() only allows String comparison
  - Cumbersome to write complex structures (e.g. bags of bags)

Example with Avro input/output

- Focus on testing script's output
- Difficulty is to generate dummy Avro data and compare result

```
text = LOAD '$input' USING AvroStorage();

flattened = FOREACH text GENERATE flatten(TOKENIZE(body)) as word;
grouped = GROUP flattened by word;
result = FOREACH grouped GENERATE group AS word, (int)COUNT($1) AS cnt;
sorted = ORDER result BY cnt DESC;

STORE result INTO '$output' USING AvroStorage();
```

- By default, PigUnit doesn't execute the STORE, but it can be overridden

```
pigTest.unoverride("STORE");
```

Simple utility classes for Avro

- *BasicAvroWriter*
  - Writes Avro file on disk based on a schema
  - Supports GenericRecord and SpecificRecord

- *BasicAvroReader*
  - Reads Avro file, the schema heads the file
  - Also supports GenericRecord and SpecificRecord

Test with Avro GenericRecord

- Create Schema with SchemaBuilder, write data, run script, read result and compare

```
@Test
public void testWordCountGenericRecord() throws IOException, ParseException {
  Schema schema = SchemaBuilder.builder().record("record").fields().
          name("text").type().stringType().noDefault().endRecord();
  GenericRecord genericRecord = new GenericData.Record(schema);
  genericRecord.put("text", "banana apple banana");
  BasicAvroWriter writer = new BasicAvroWriter(new Path(new
File("input.avro").getAbsolutePath()), schema, getFileSystem());
  writer.append(genericRecord);

  PigTest t = newPigTest("pig/src/main/pig/wordcount_avro.pig", new String[]
{"input=input.avro", "output=sorted.avro"});
  t.unoverride("STORE");
  t.runScript();

  //Check output
  BasicAvroReader reader = new BasicAvroReader(new Path(new
File("sorted.avro").getAbsolutePath()), getFileSystem());
  Map<Utf8, GenericRecord> result = reader.readAndMapAll("word");
  Assert.assertEquals(result.size(), 2);
  Assert.assertEquals(result.get(new Utf8("banana")).get("cnt"), 2);
  Assert.assertEquals(result.get(new Utf8("apple")).get("cnt"), 1);
}
```

Test with Avro SpecificRecord

- Use *InputRecord* and *OutputRecord* generated Java classes, write data, run script, read result and compare

```
@Test
public void testWordCountSpecificRecord() throws IOException, ParseException {
  InputRecord input = InputRecord.newBuilder().setText("banana apple banana").build();

  BasicAvroWriter<InputRecord> writer = new BasicAvroWriter<InputRecord>(new Path(new
File("input.avro").getAbsolutePath()), input.getSchema(), getFileSystem());
  writer.writeAll(input);

  PigTest t = newPigTest("pig/src/main/pig/wordcount_avro.pig", new String[]
{"input=input.avro", "output=sorted.avro"});
  t.unoverride("STORE");
  t.runScript();

  //Check output
  BasicAvroReader<OutputRecord> reader = new BasicAvroReader<OutputRecord>(new Path(new
File("sorted.avro").getAbsolutePath()), getFileSystem());
  List<OutputRecord> result = reader.readAll();
  Assert.assertEquals(result.size(), 2);
  Assert.assertEquals(result.get(0),
OutputRecord.newBuilder().setWord("banana").setCount(2).build());
  Assert.assertEquals(result.get(1),
OutputRecord.newBuilder().setWord("apple").setCount(1).build());
}
```

Common pitfalls

- PigUnit
  - Mocking capabilities are very limited
  - Overhead of 1-5 seconds per script
  - Cryptic error messages sometimes (NullPointerException)
- Pig UDFs
  - Can be tested independently

Spark Testing

Spark Testing Base

Base classes to use when writing tests with Spark

- https://github.com/holdenk/spark-testing-base
- Functionalities
  - Provides SparkContext
  - Utilities to compare RDDs and DataFrames
  - Simulate how Streaming works
  - Includes cool RDD and DataFrames generator

Thank You!

Extra Resources

- https://github.com/miguno/avro-hadoop-starter
- http://www.michael-noll.com/blog/2013/07/04/using-avro-in-mapreduce-jobs-with-hadoop-pig-hive/
- http://blog.cloudera.com/blog/2015/09/making-apache-spark-testing-easy-with-spark-testing-base/
- http://www.slideshare.net/hkarau/effective-testing-for-spark-programs-strata-ny-2015
- http://avro.apache.org/docs/current/
- http://www.confluent.io/blog/schema-registry-kafka-stream-processing-yes-virginia-you-really-need-one
- http://mkuthan.github.io/blog/2015/03/01/spark-unit-testing/
- https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying