Spark Streaming for Realtime Auctions

@russellcardullo
Sharethrough

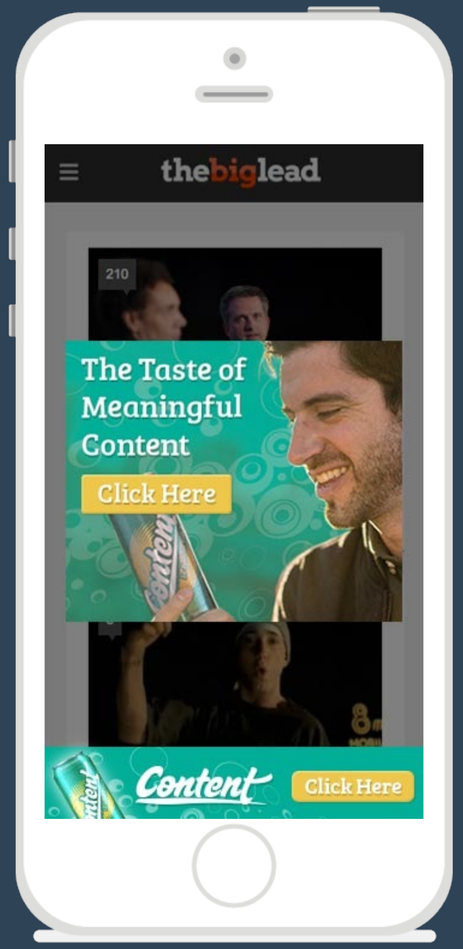
sharethrough
ENGINEERING

Agenda

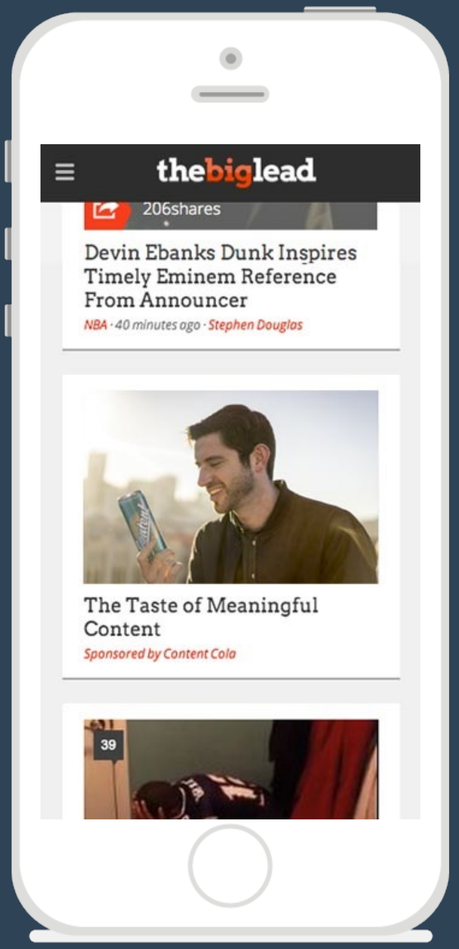
- Sharethrough?
- Streaming use cases
- How we use Spark
- Next steps

sharethrough
ENGINEERING

Sharethrough

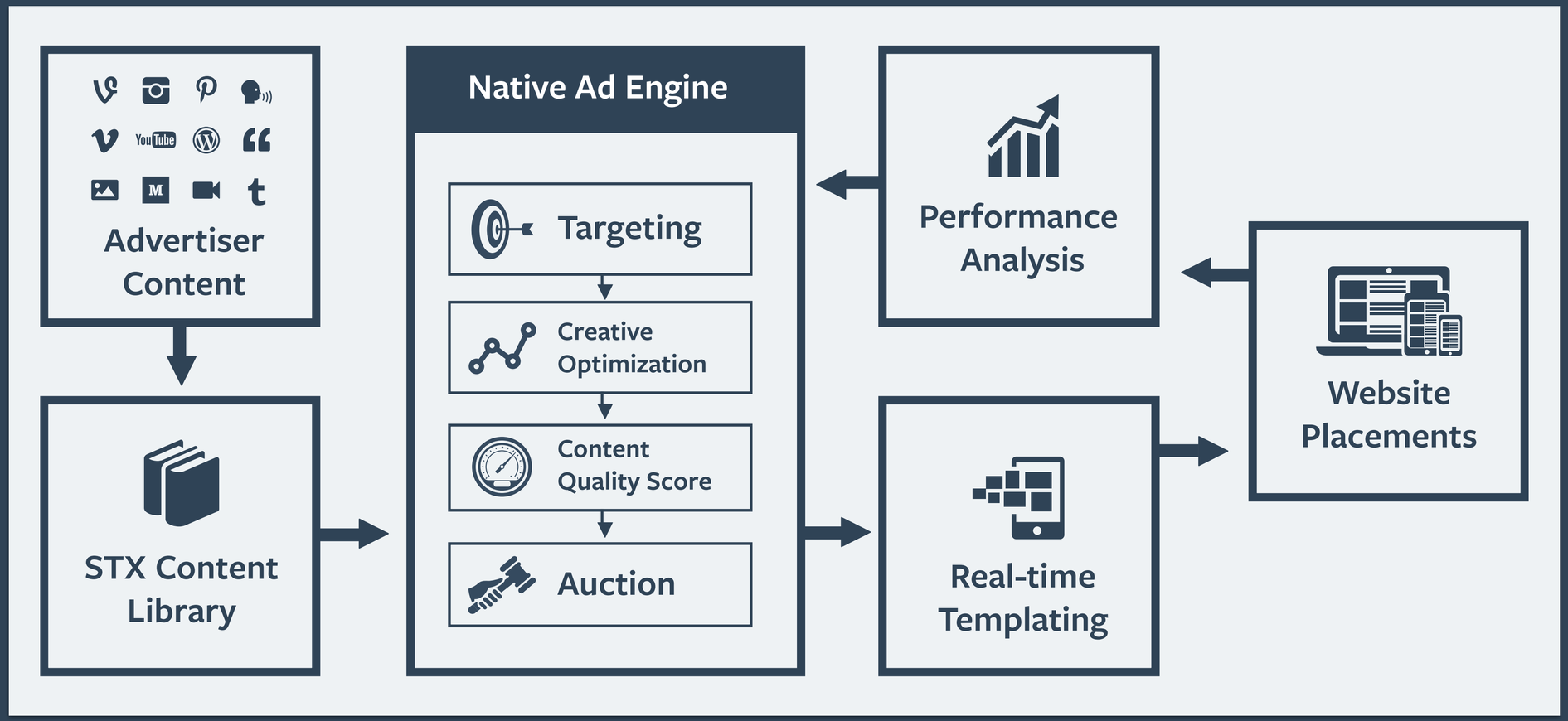


vs



sharethrough
ENGINEERING

The Sharethrough Native Exchange



sharethrough
ENGINEERING

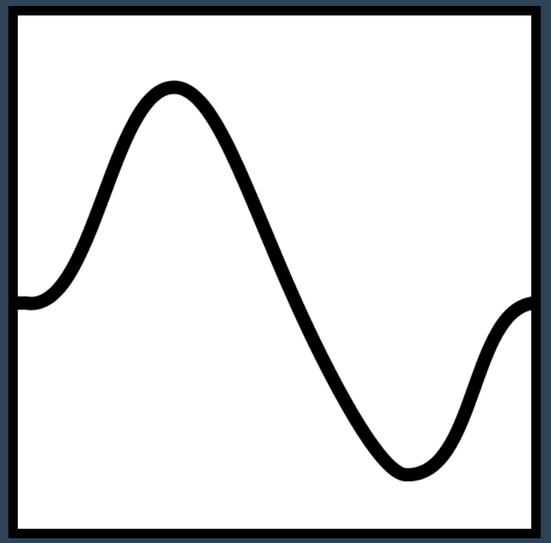
How can we use streaming data?

Use Cases

$$\mu^* = \max_k \{ \mu_k \}$$

Creative Optimization

Spend Tracking

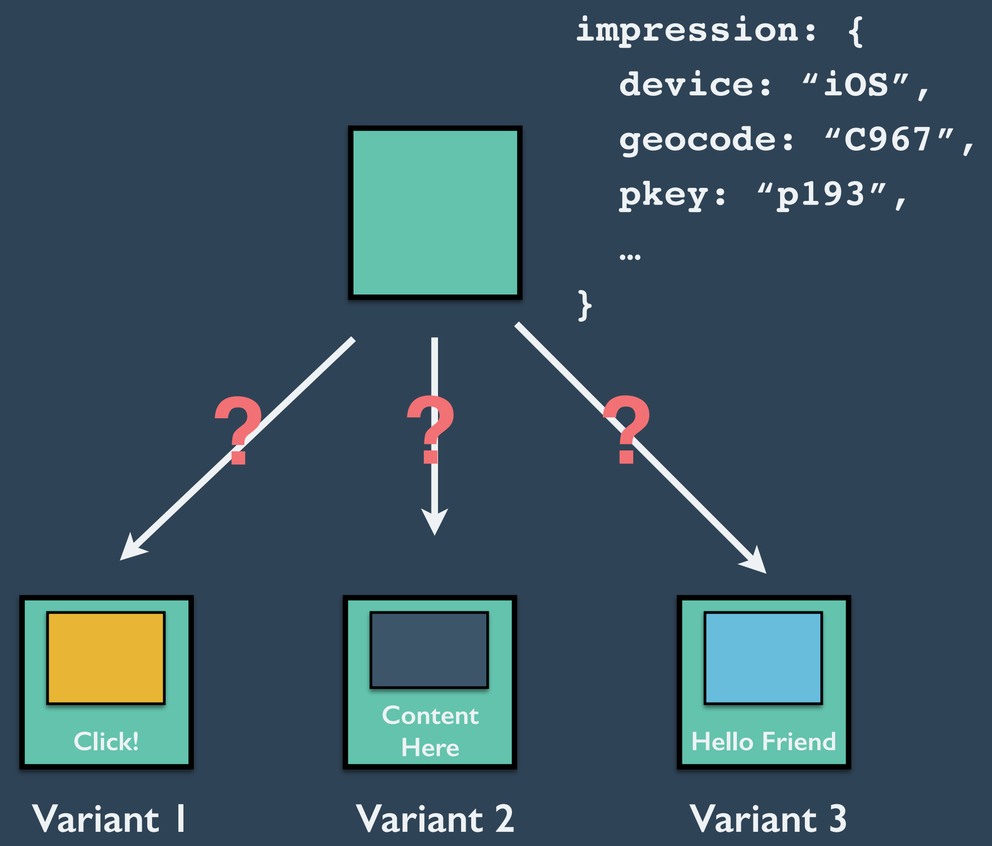


Operational Monitoring

sharethrough
ENGINEERING

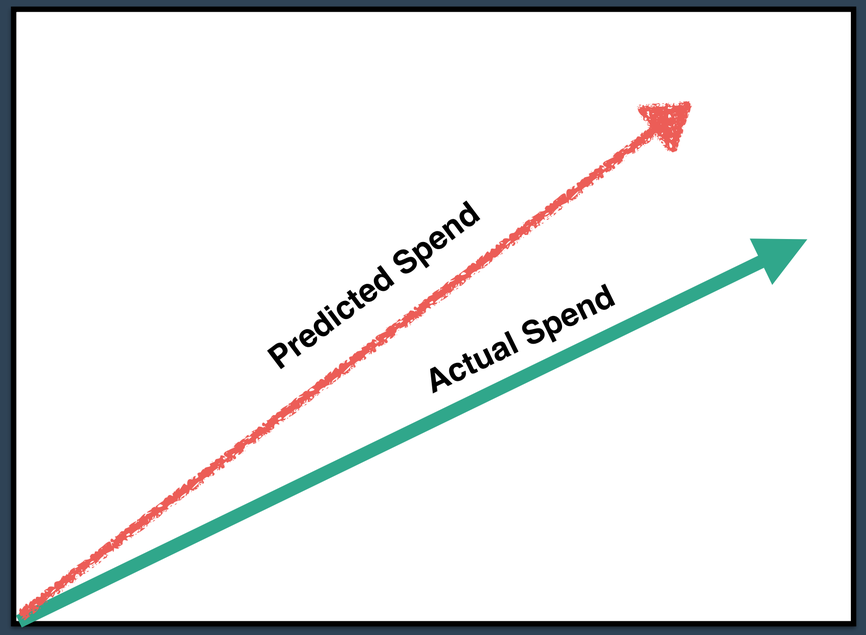
Creative Optimization

- Choose best performing variant
- Short feedback cycle required



sharethrough ENGINEERING

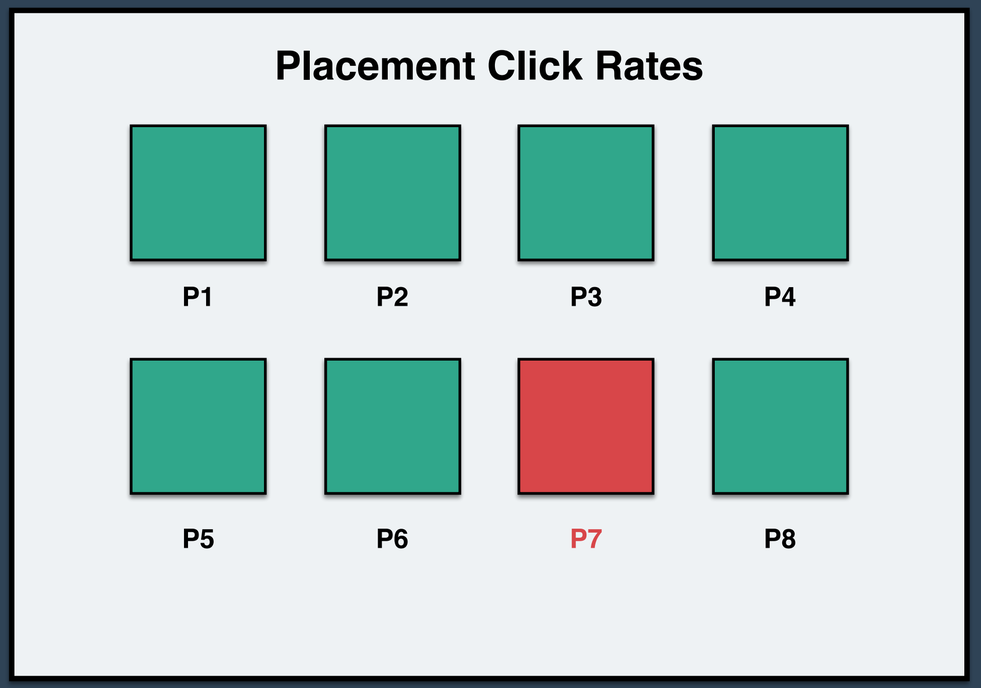
Spend Tracking



- Spend on visible impressions and clicks
- Actual spend happens asynchronously
- Want to correct prediction for optimal serving

Operational Monitoring

- Detect issues with content served on third party sites
- Use same logs as reporting



sharethrough
ENGINEERING

We can directly measure business impact of using this data sooner

sharethrough
ENGINEERING

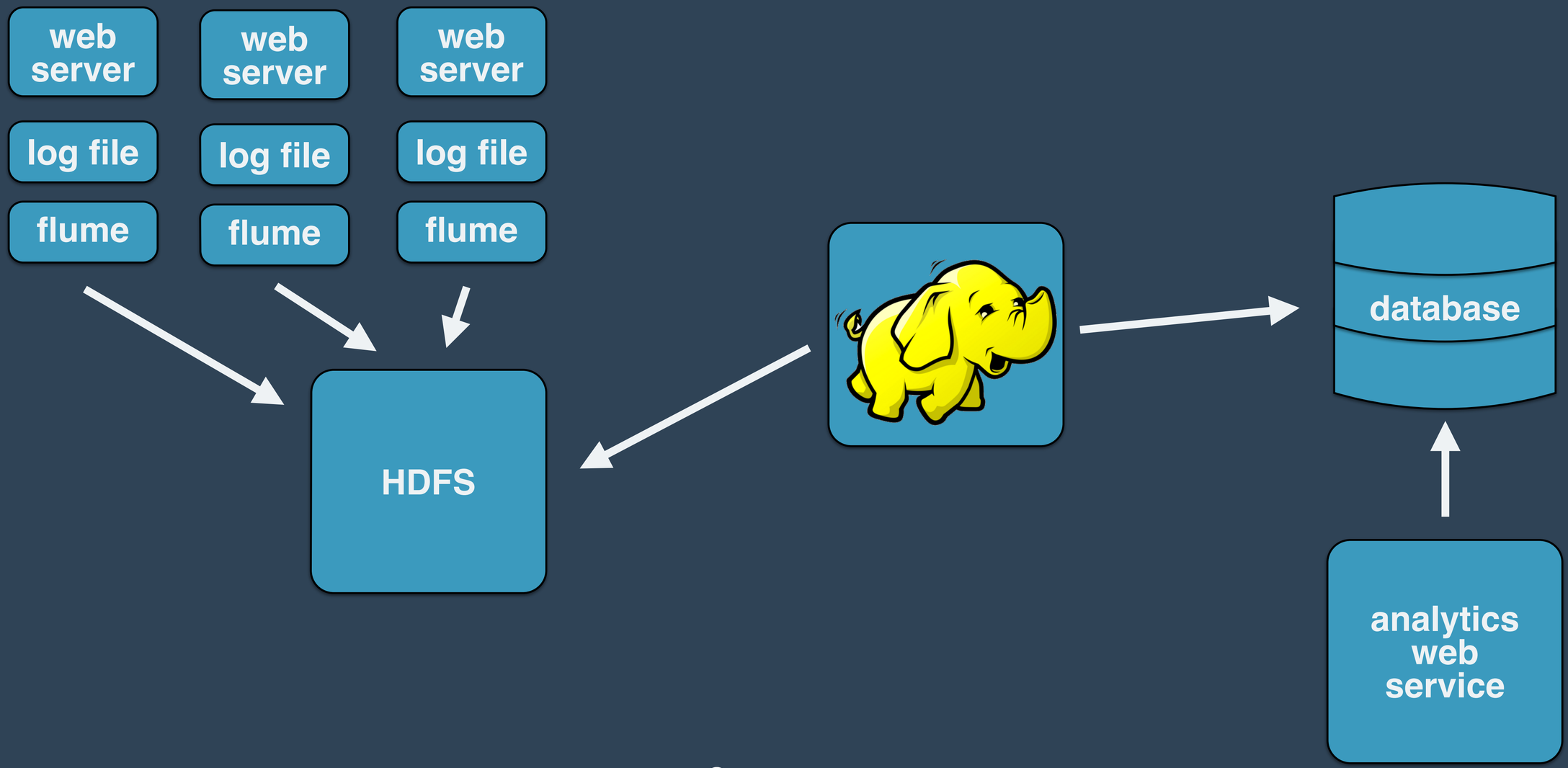
Why use Spark to build these features?

Why Spark?

- Scala API
- Supports batch and streaming
- Active community support
- Easily integrates into existing Hadoop ecosystem
- But it doesn’t require Hadoop in order to run

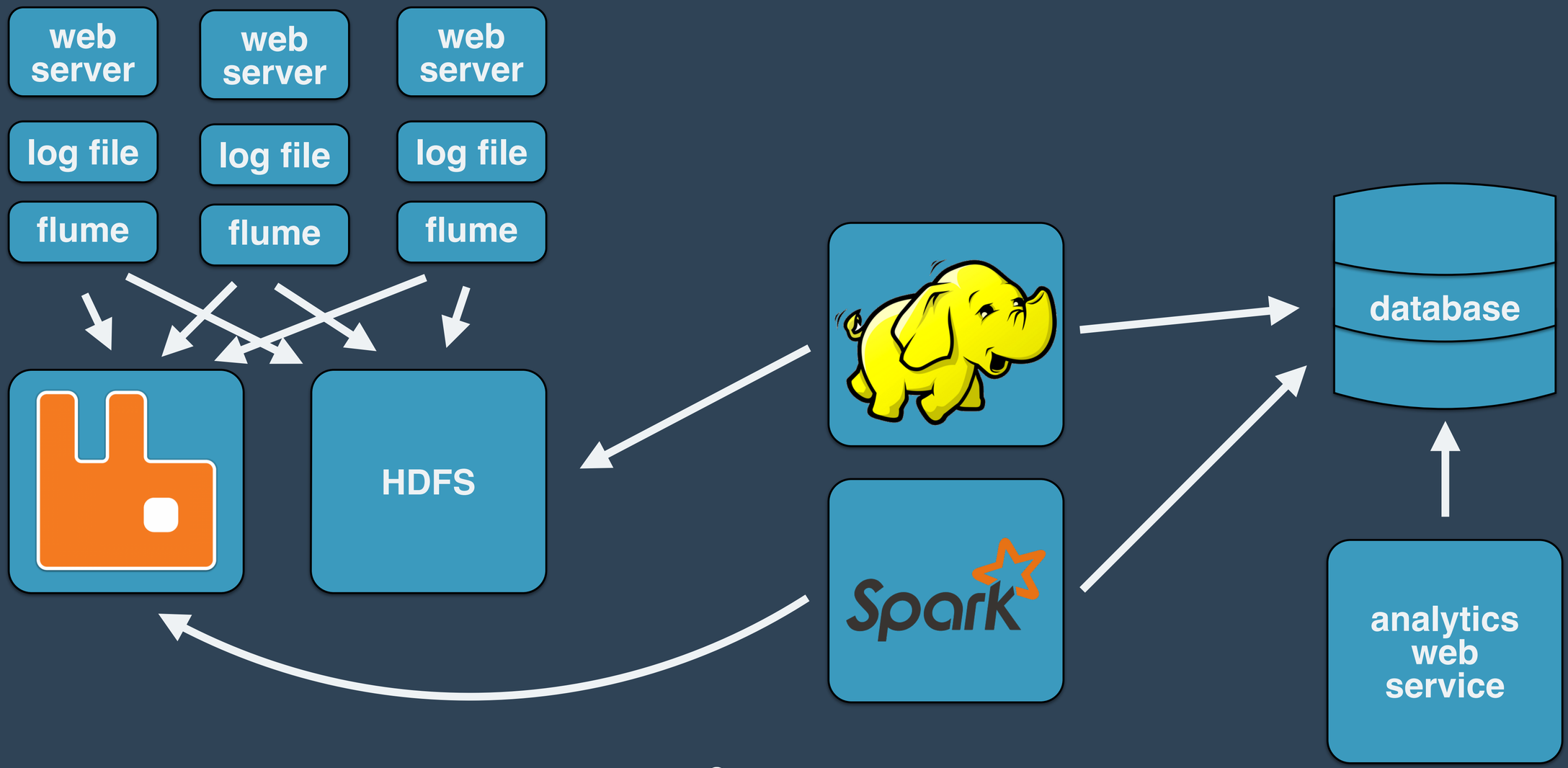
How we've integrated Spark

Existing Data Pipeline



sharethrough ENGINEERING

Pipeline with Streaming



sharethrough
ENGINEERING

Batch

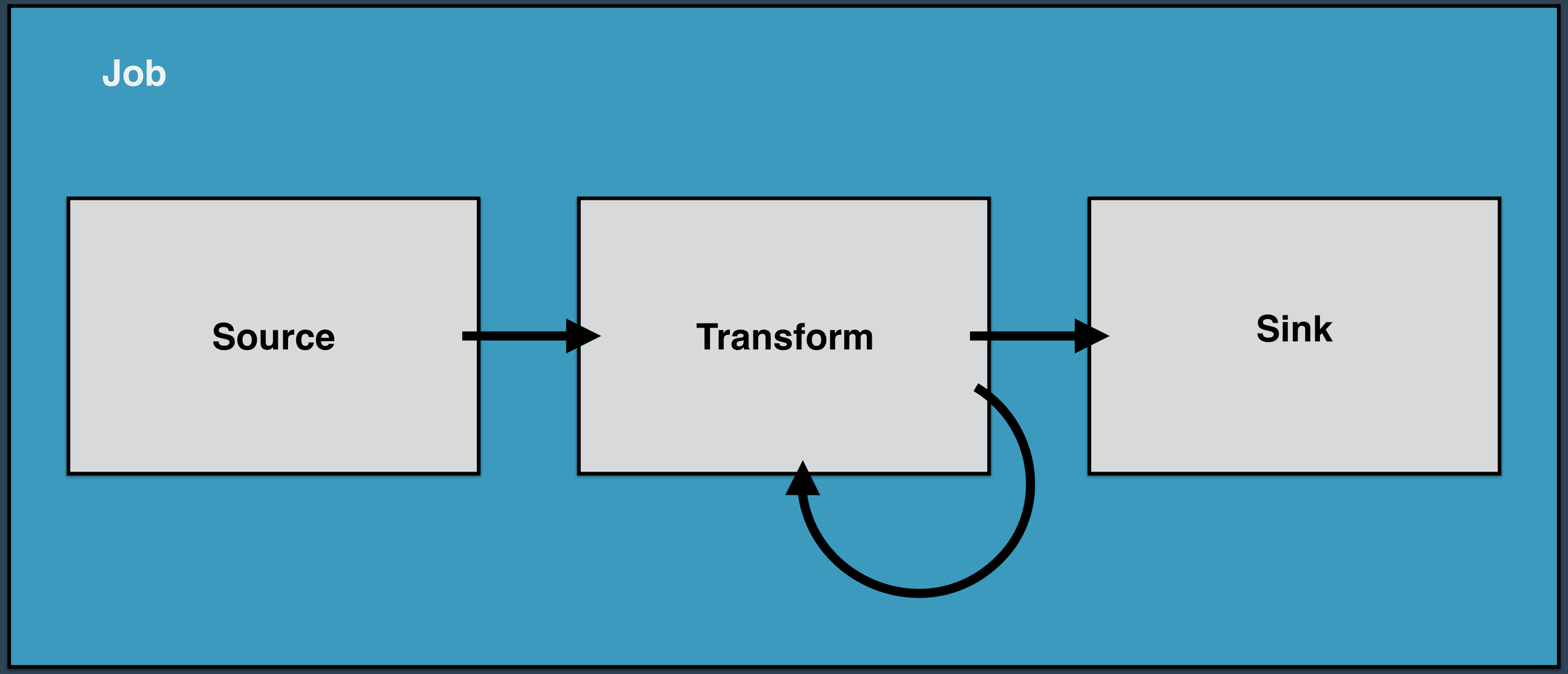
- Daily reporting
- Billing / earnings
- Anything with strict SLA
- Correctness > low latency

Streaming

- “Real-Time” reporting
- Low latency to use data
- Only reliable as source
- Low latency > correctness

Spark Job Abstractions

Job Organization



Sources

```
case class BeaconLogLine(
  timestamp: String,
  uri: String,
  beaconType: String,
  pkey: String,
  ckey: String
)

object BeaconLogLine {

  def newDStream(ssc: StreamingContext, inputPath: String): DStream[BeaconLogLine] = {
    ssc.textFileStream(inputPath).map { parseRawBeacon(_) }
  }

  def parseRawBeacon(b: String): BeaconLogLine = {
    ...
  }
}
```

case class for pattern matching

generate DStream

encapsulate common operations

sharethrough ENGINEERING

Transformations

type safety from case class

```
def visibleByPlacement(source: DStream[BeaconLogLine]): DStream[(String, Long)] = {
  source.
    filter(data => {
      data.uri == "/strbeacon" && data.beaconType == "visible"
    }).
    map(data => (data.pkey, 1L)).
    reduceByKey(_ + _)
}
```

Sinks

custom sinks for new stores

```
class RedisSink @Inject()(store: RedisStore) {

  def sink(result: DStream[(String, Long)]) = {
    result.foreachRDD { rdd =>
      rdd.foreach { element =>
        val (key, value) = element
        store.merge(key, value)
      }
    }
  }
}
```

Jobs

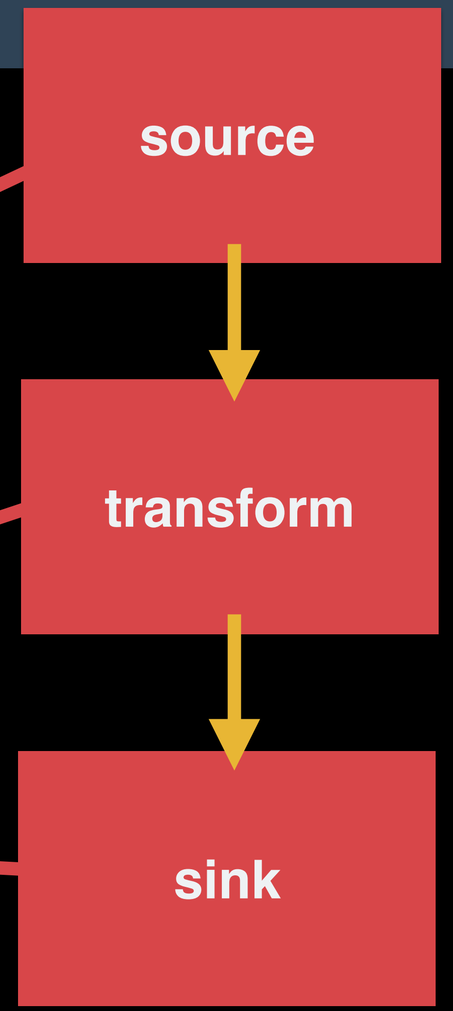
```
object ImpressionsForPlacements {

  def run(config: Config, inputPath: String) {
    val conf = new SparkConf().
                setMaster(config.getString("master")).
                setAppName("Impressions for Placement")

    val sc = new SparkContext(conf)
    val ssc = new StreamingContext(sc, Seconds(5))

    val source = BeaconLogLine.newDStream(ssc, inputPath)
    val visible = visibleByPlacement(source)
    sink(visible)

    ssc.start
    ssc.awaitTermination
  }
}
```



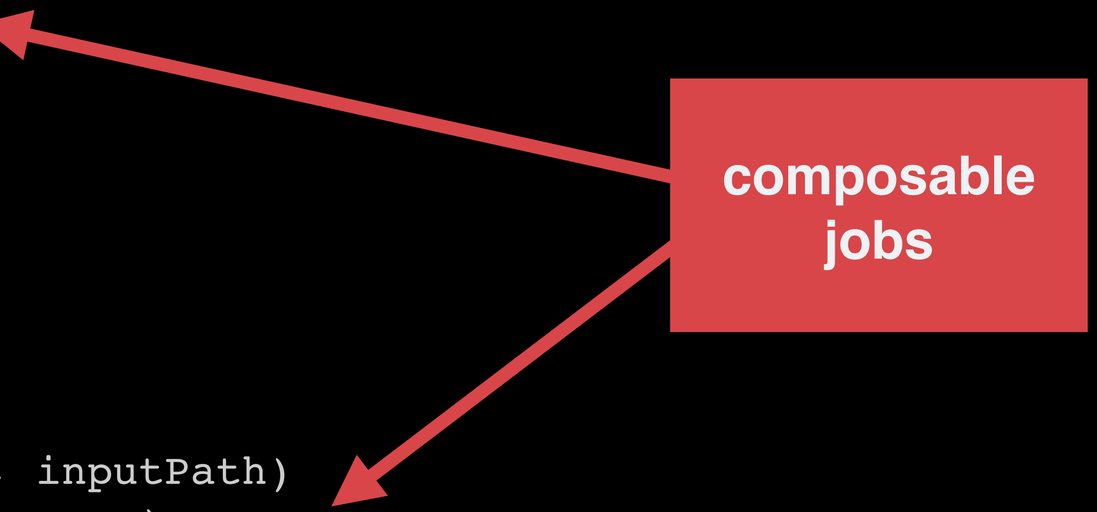
Advantages?

Code Reuse

```
object PlacementVisibles {
  …
  val source = BeaconLogLine.newDStream(ssc, inputPath)
  val visible = visibleByPlacement(source)
  sink(visible)
  …
}

…

object PlacementEngagements {
  …
  val source = BeaconLogLine.newDStream(ssc, inputPath)
  val engagements = engagementsByPlacement(source)
  sink(engagements)
  …
}
```



Readability

```
ssc.textFileStream(inputPath).
  map { parseRawBeacon(_) }.
  filter(data => {
    data._2 == "/strbeacon" && data._3 == "visible"
  }).
  map(data => (data._4, 1L)).
  reduceByKey(_ + _).
  foreachRDD { rdd =>
    rdd.foreach { element =>
      store.merge(element._1, element._2)
    }
  }
```

sharethrough
ENGINEERING

Readability

```
val source = BeaconLogLine.newDStream(ssc, inputPath)
val visible = visibleByPlacement(source)
redis.sink(visible)
```

Testing

```
def assertTransformation[T: Manifest, U: Manifest](
  transformation: T => U,
  input: Seq[T],
  expectedOutput: Seq[U]
): Unit = {
    val ssc = new StreamingContext("local[1]", "Testing", Seconds(1))
    val source = ssc.queueStream(new SynchronizedQueue[RDD[T]]())
    val results = transformation(source)

    var output = Array[U]()
    results.foreachRDD { rdd => output = output ++ rdd.collect() }
    ssc.start
    rddQueue += ssc.sparkContext.makeRDD(input, 2)
    Thread.sleep(jobCompletionWaitTimeMillis)
    ssc.stop(true)

    assert(output.toSet === expectedOutput.toSet)
 }
```

function, input, expectation

test

sharethrough ENGINEERING

Testing

```
test("#visibleByPlacement") {

  val input = Seq(
    "pkey=abcd, …",
    "pkey=abcd, …",
    "pkey=wxyz, …",
  )

  val expectedOutput = Seq( ("abcd",2),("wxyz", 1) )

  assertTransformation(visibleByPlacement, input, expectedOutput)
}
```

use our test helper

Other Learnings

sharethrough
ENGINEERING

Other Learnings

- Keeping your driver program healthy is crucial
  - 24/7 operation and monitoring
  - Spark on Mesos? Use Marathon.
- Pay attention to settings for spark.cores.max
  - Monitor data rate and increase as needed
- Serialization on classes
  - Java
  - Kryo

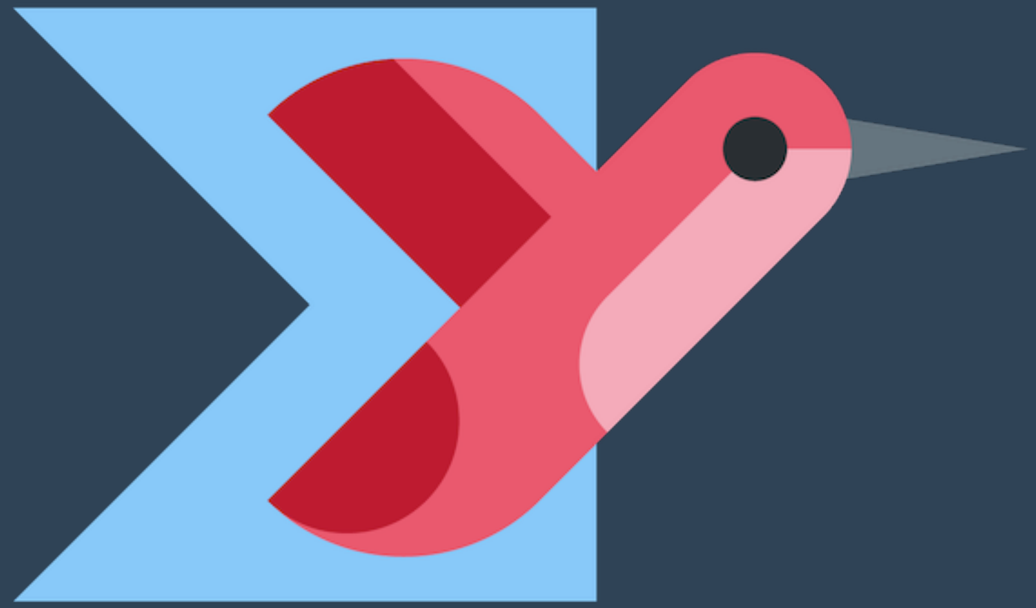
sharethrough
ENGINEERING

What's next?

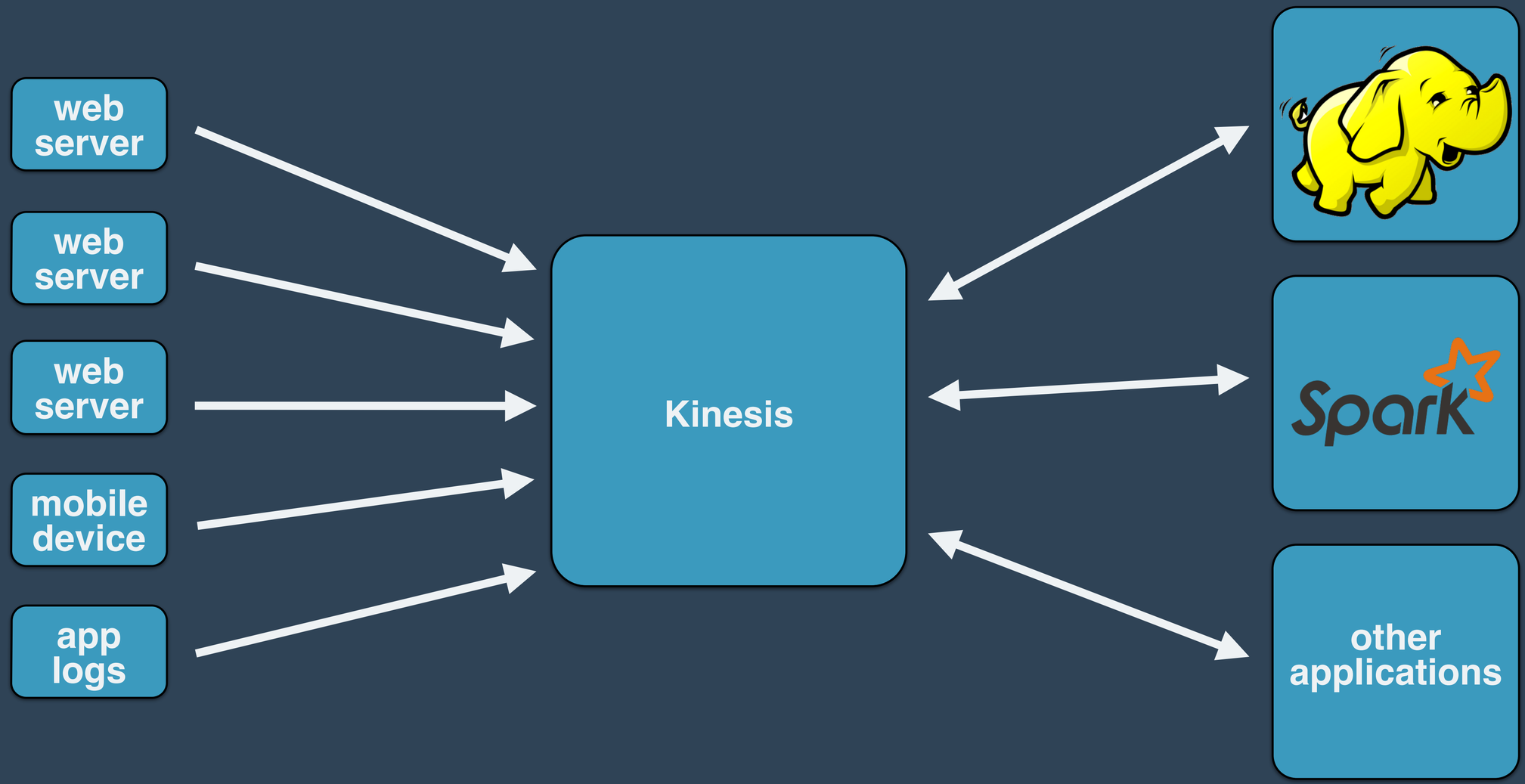
sharethrough
ENGINEERING

Twitter Summingbird

- Write-once, run anywhere
- Supports:
  - Hadoop MapReduce
  - Storm
  - Spark (maybe?)



Amazon Kinesis



sharethrough ENGINEERING

Thanks!

sharethrough
ENGINEERING