Large Scale Multimedia Data Processing on Spark and Related Applications at Baidu

Quan Wang
Baidu USA

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

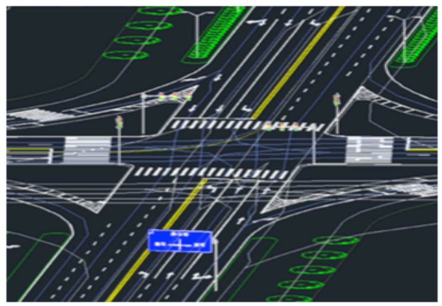
Motivations

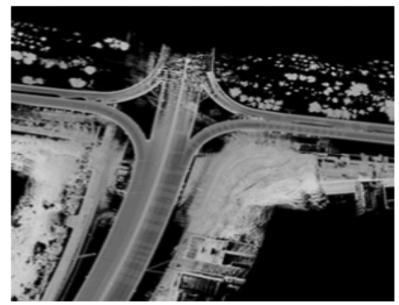
- Why Multimedia?

- Examples of large scale multimedia processing at Baidu:
  - HD map generation and simulation for self-driving cars
  - Image feature exaction and transform for CTR predictions

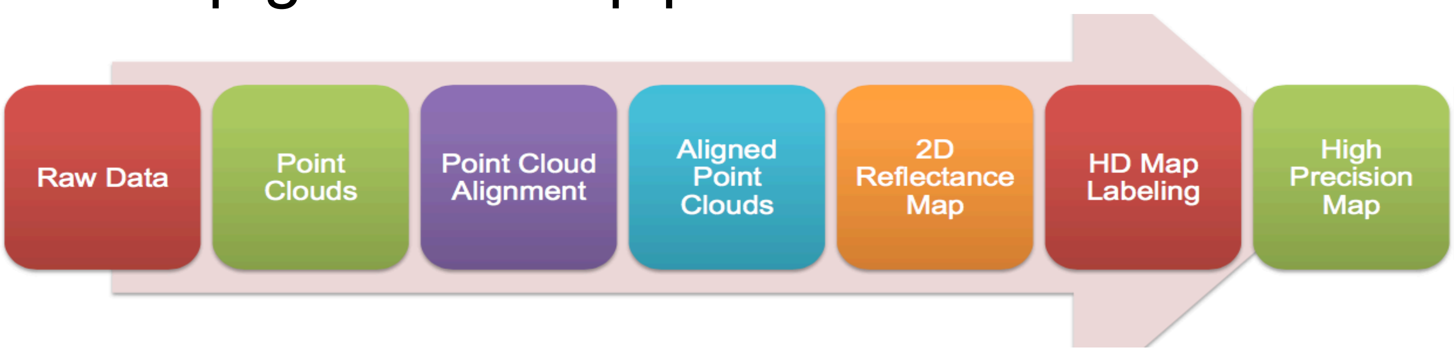
Application Example: Self-Driving Cars

- Maps for navigation, planning and localization:



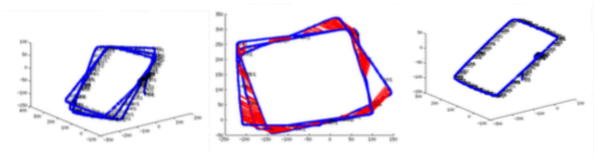


- HD Map generation pipeline:

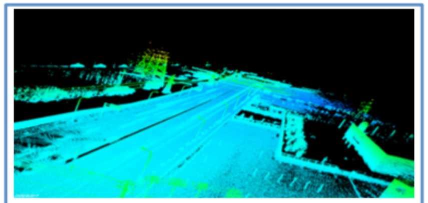


Data challenge in Self-driving Cars Project

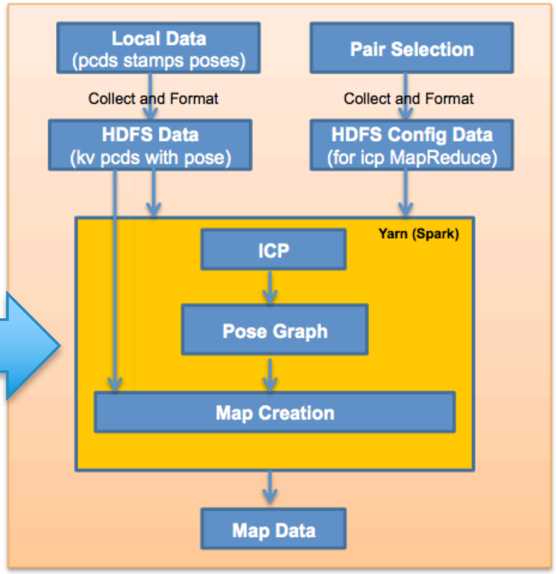
- Backend map generation and simulation:
  - Point clouds inputs of 40MB/s for one single 3D LiDAR sensor
  - Counting all 2D/3D sensors => TB level of data, per hour, per car
- Example: HD map generation on Spark



Loop closure detection && Global Optimization



Pointcloud Alignment



Another Example: Baidu Image Search

- Billions of images need feature extractions & transformations for deep learning applications:
  - Image recognition and classification
  - Ranking for best picture to show
  - CTR prediction

Challenges

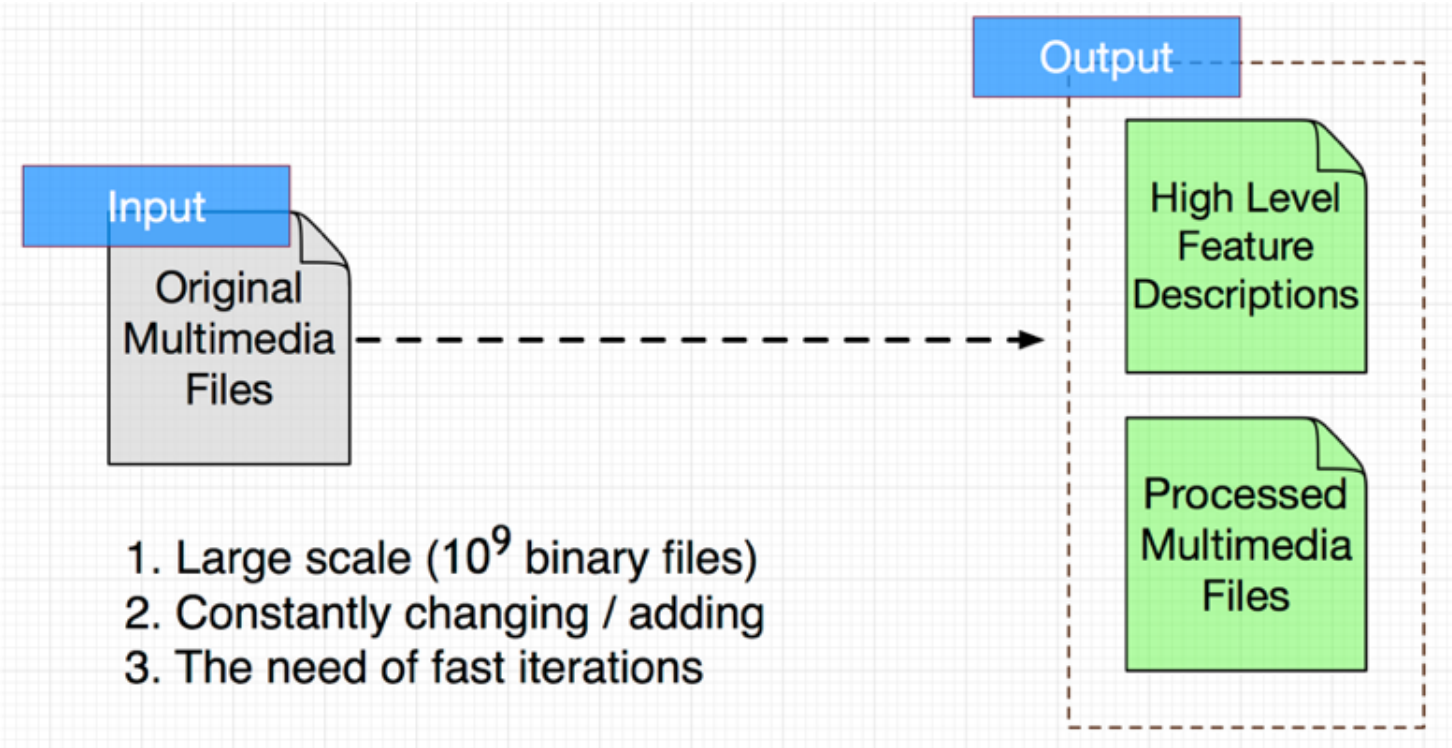
- Core functions for feature extraction
- Efficient large scale distributed executions of feature extraction with multimedia input support
- Plug and play for any feature extraction executable never designed for Spark; Flexible and easy to use for platform users

Feature Extraction Core Function

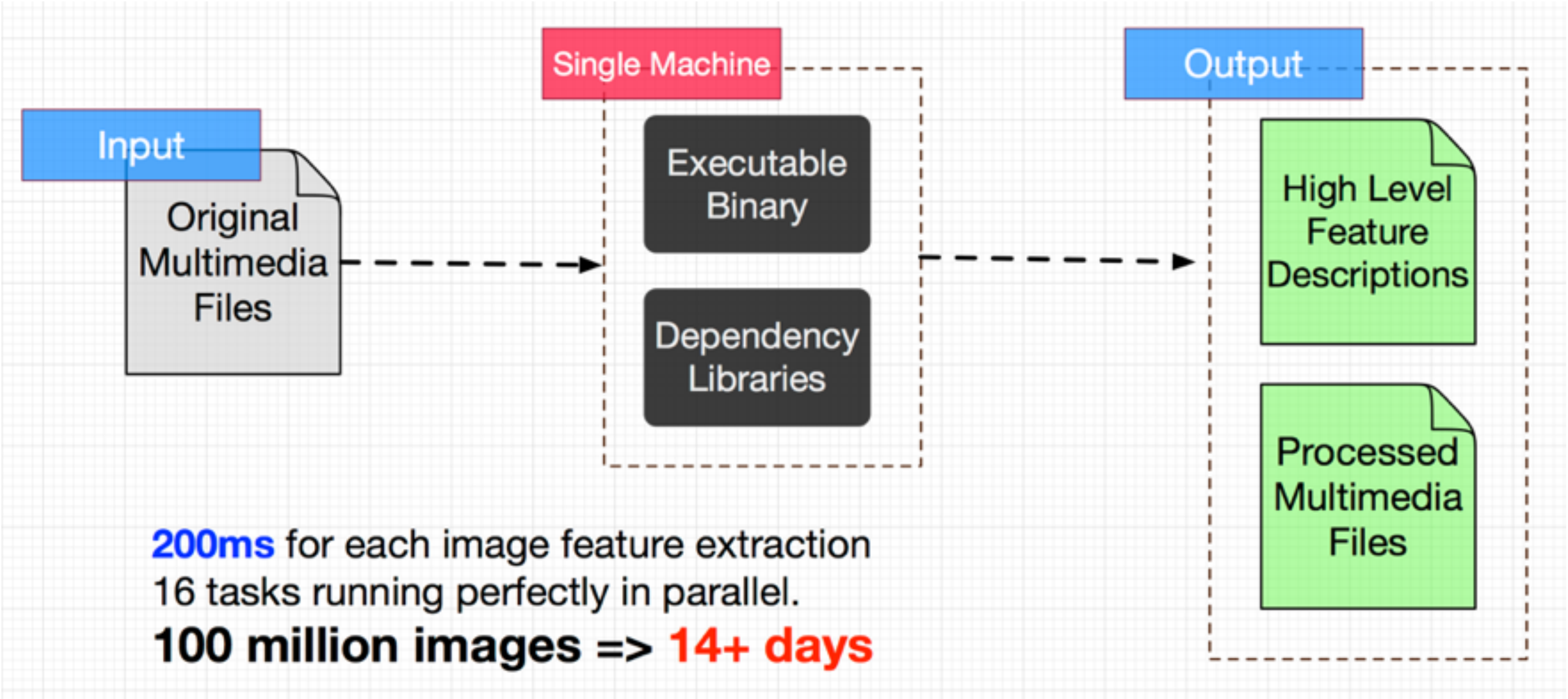
- Feature extraction C++ program depends on CDNN + OpenCV library
  - Compute the per pixel difference based on a pre-computed mean
  - Feed the difference values into a pre-computed CDNN model
  - Produce image features after multi-layers of computations

- Need streaming/pipe based function support

End to End Requirements

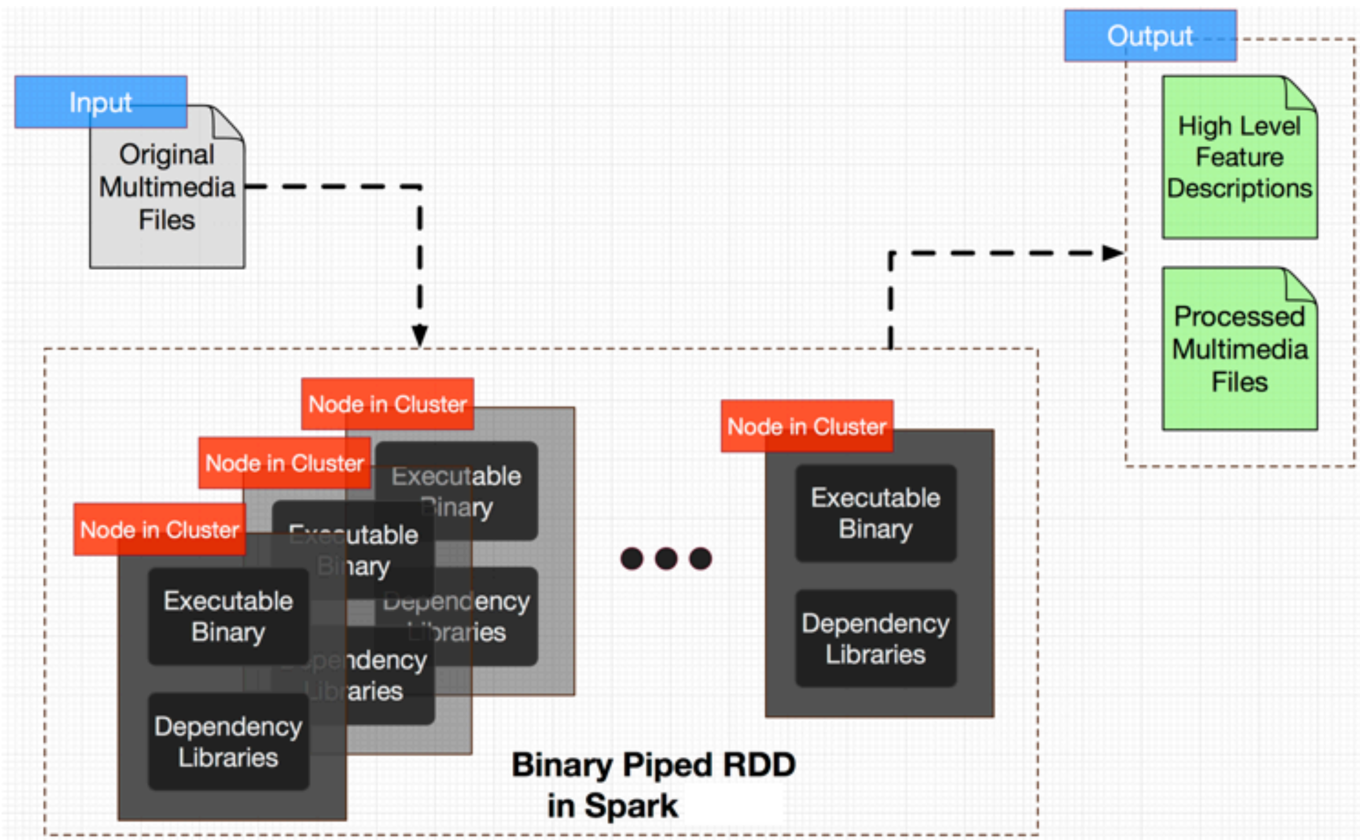


Single Node Execution

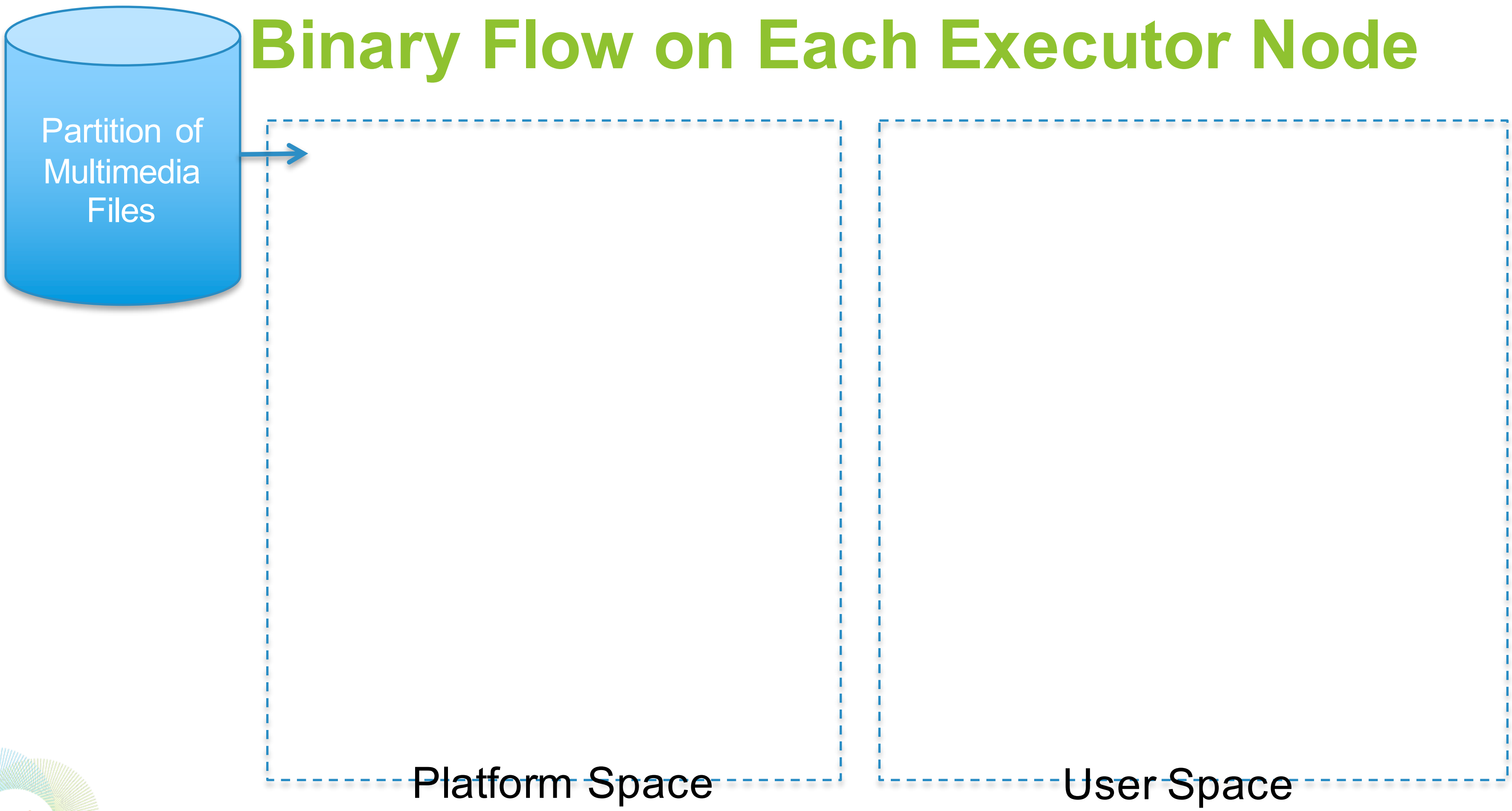


Distribute to 500 machines
How to manage?

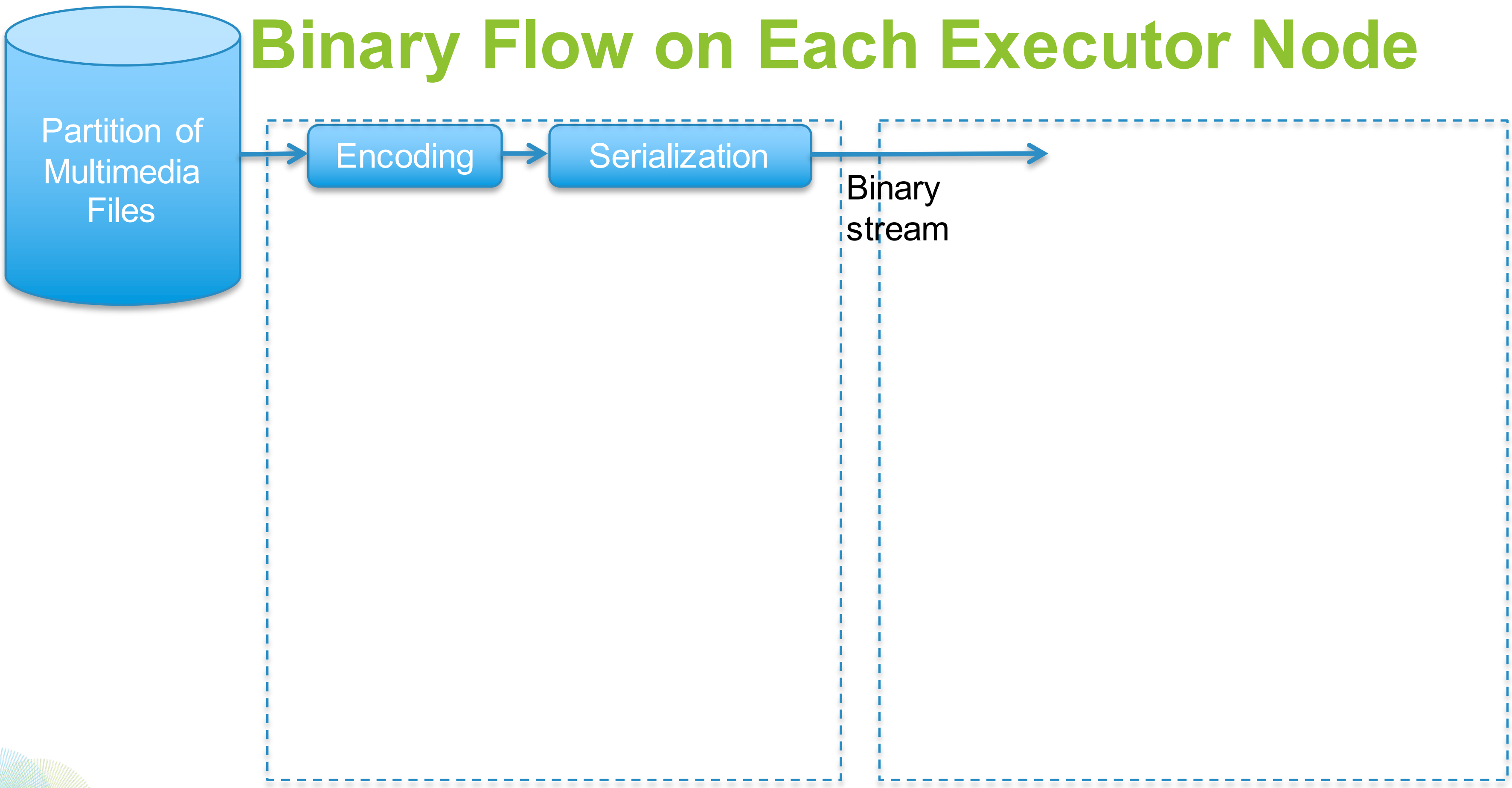
Distributed Execution



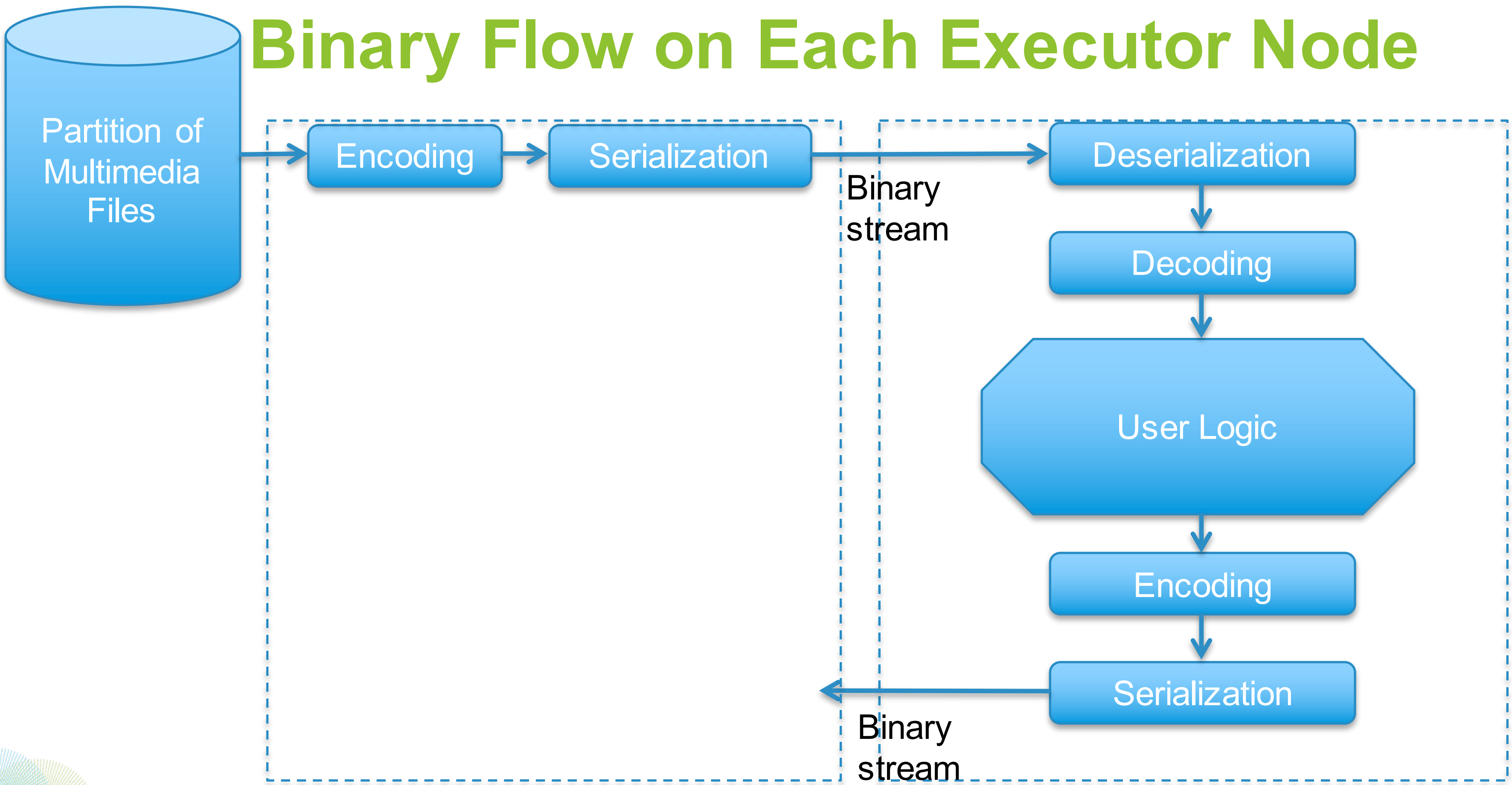
Technical Details:
Binary Flow on Each Executor Node



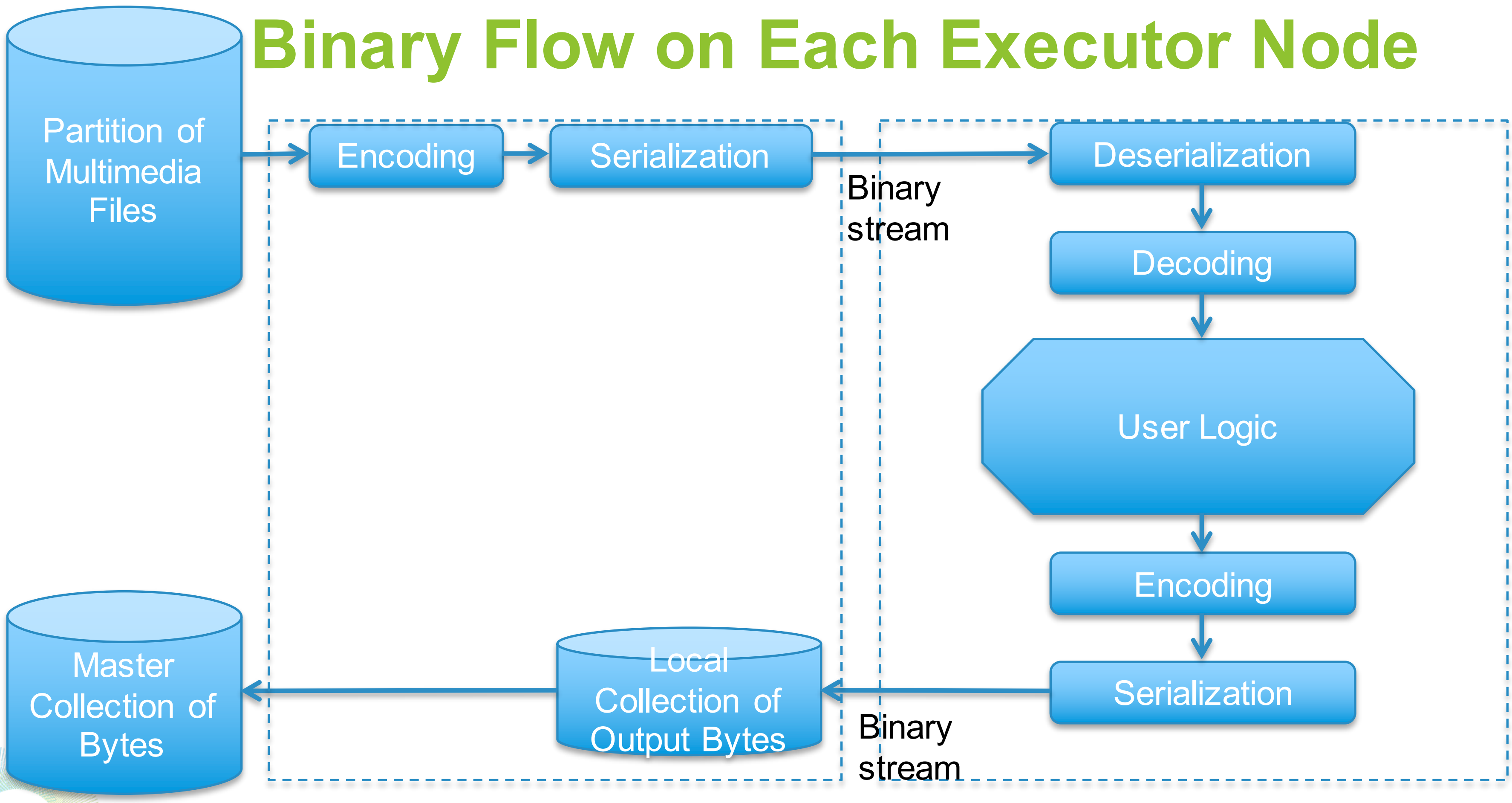
Technical Details:
Binary Flow on Each Executor Node



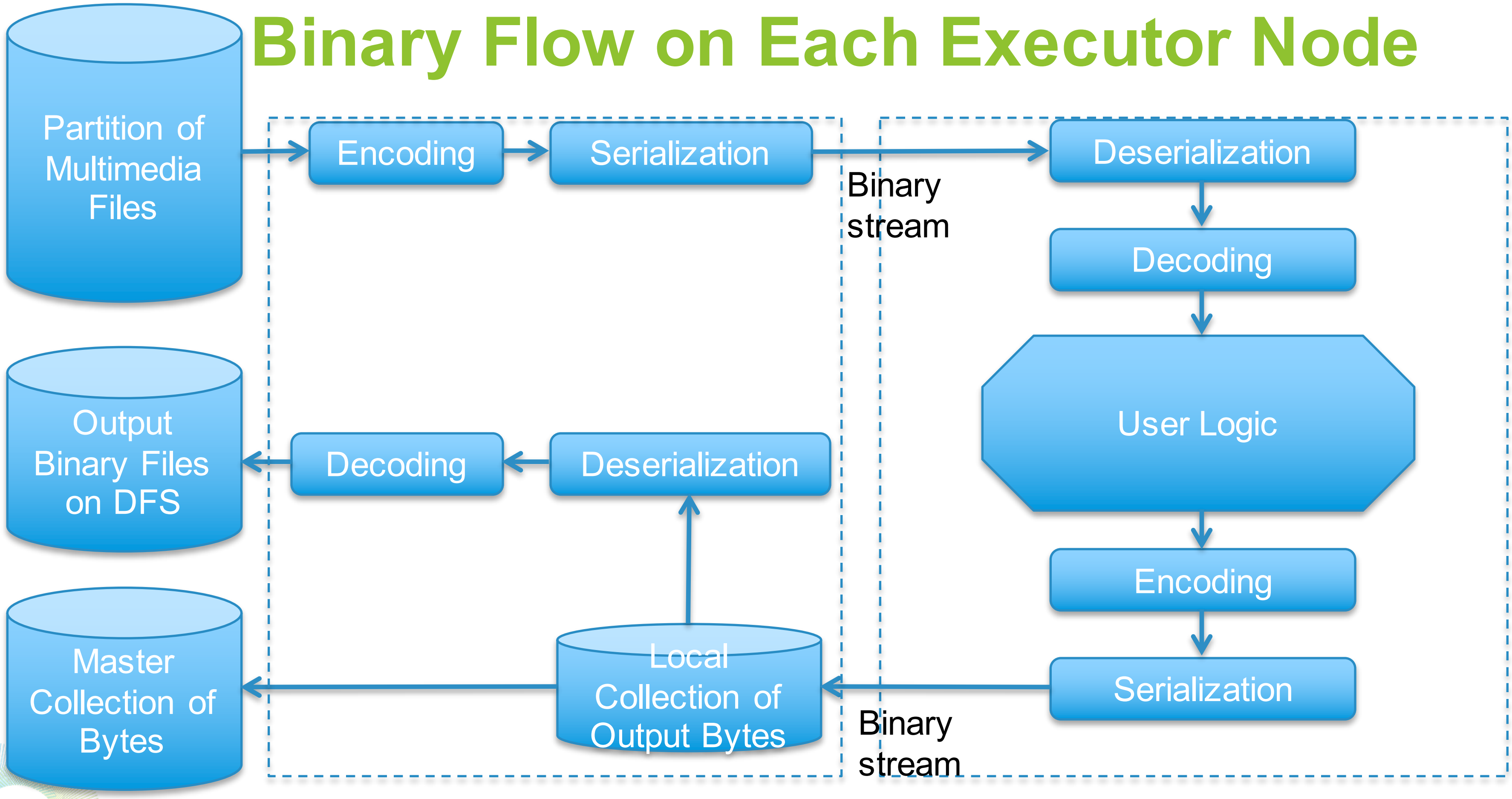
Technical Details:
Binary Flow on Each Executor Node



Technical Details:
Binary Flow on Each Executor Node



Technical Details:
Binary Flow on Each Executor Node



Implementation Highlights

- All data serialization and encoding inside Spark RDD, thus linear scalable
- Flexible output format (feature vectors, ranking scores, processed binaries, etc.)
- Easy to plug in customized encoding/serialization functions directly into platform
- Support of passing spark internal information (e.g. partition id, task attempt id) into user program

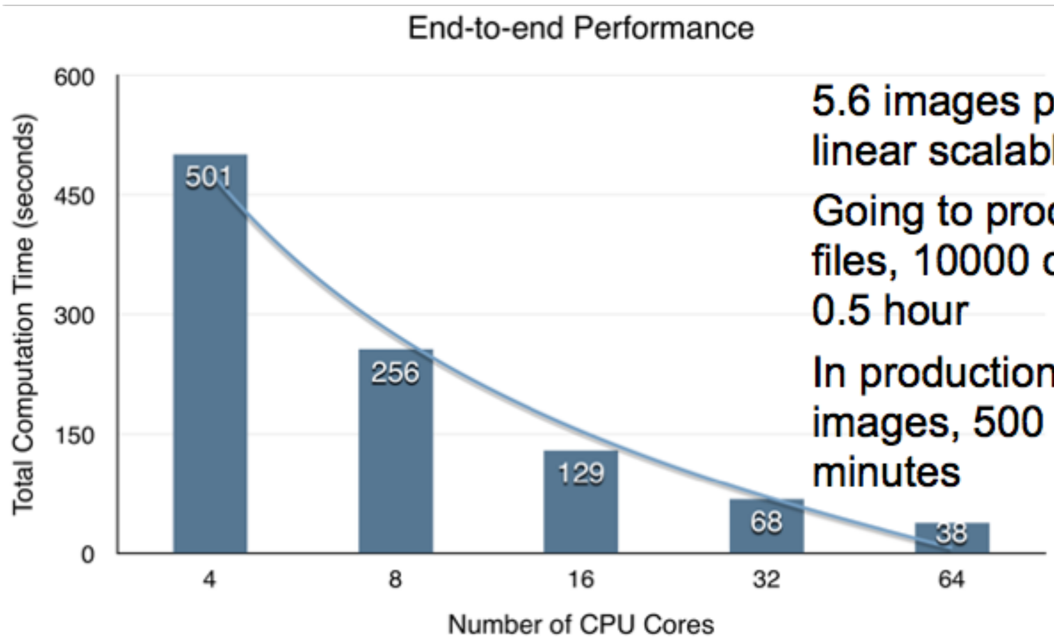
Flexibility

```scala
/**
 * A Binary Piped RDD which pipes the contents of each parent partition to an external command in the format
 * of binary streams and returns the output as a collection of bytes.
 *
 * @param savePath Optional file system path to save the decoded and de-serialized output bytes,
 *                 only needed for the saveResults API
 * @param printPipeContext Optional plug in function to add a context header to the binary stream
 * @param printRDDElement Optional plug in function for customized serialization and encoding
 * @param saveRDDElement Optional plug in function for customized decoding and de-serialization
 */
class BinPipedRDD[T: ClassTag](
                              prev: RDD[(String, PortableDataStream)],
                              command: Seq[String],
                              savePath: String,
                              envVars: Map[String, String],
                              printPipeContext: (Array[Byte] => Unit) => Unit,
                              printRDDElement: ((String, PortableDataStream), Array[Byte]=> Unit) => Unit,
                              saveRDDElement: (Iterator[Byte]) => Unit,
                              separateWorkingDir: Boolean)
  extends RDD[Byte](prev) {
```

Flexible output format (feature vectors, ranking scores, processed binaries, etc.)
Easy to plug in Customized encoding/serialization functions directly into Spark

Performance Results

Running on a Spark cluster, over 11k images inputs with archive function on, running feature extraction on each image



5.6 images per core per second, and linear scalable~!

Going to production: 100 million image files, 10000 cores cluster => less than 0.5 hour

In production: daily incremental 3 million images, 500 cores cluster => within 20 minutes

Conclusion

- General data intelligence and analysis
  - Binary input format + Pipe based bin/lib execution

Missing functionality in Spark/Hadoop

- Introduce the Binary Piped RDD for:
  - Platform level abstraction of input data format in their original binary form
  - Seamless streaming to and from existing executable/libraries for high level data analysis and understanding
  - Linear scalability with input data

THANK YOU.

Questions?

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO