A Quick Tour on Big-data

Ahmed Eldawy

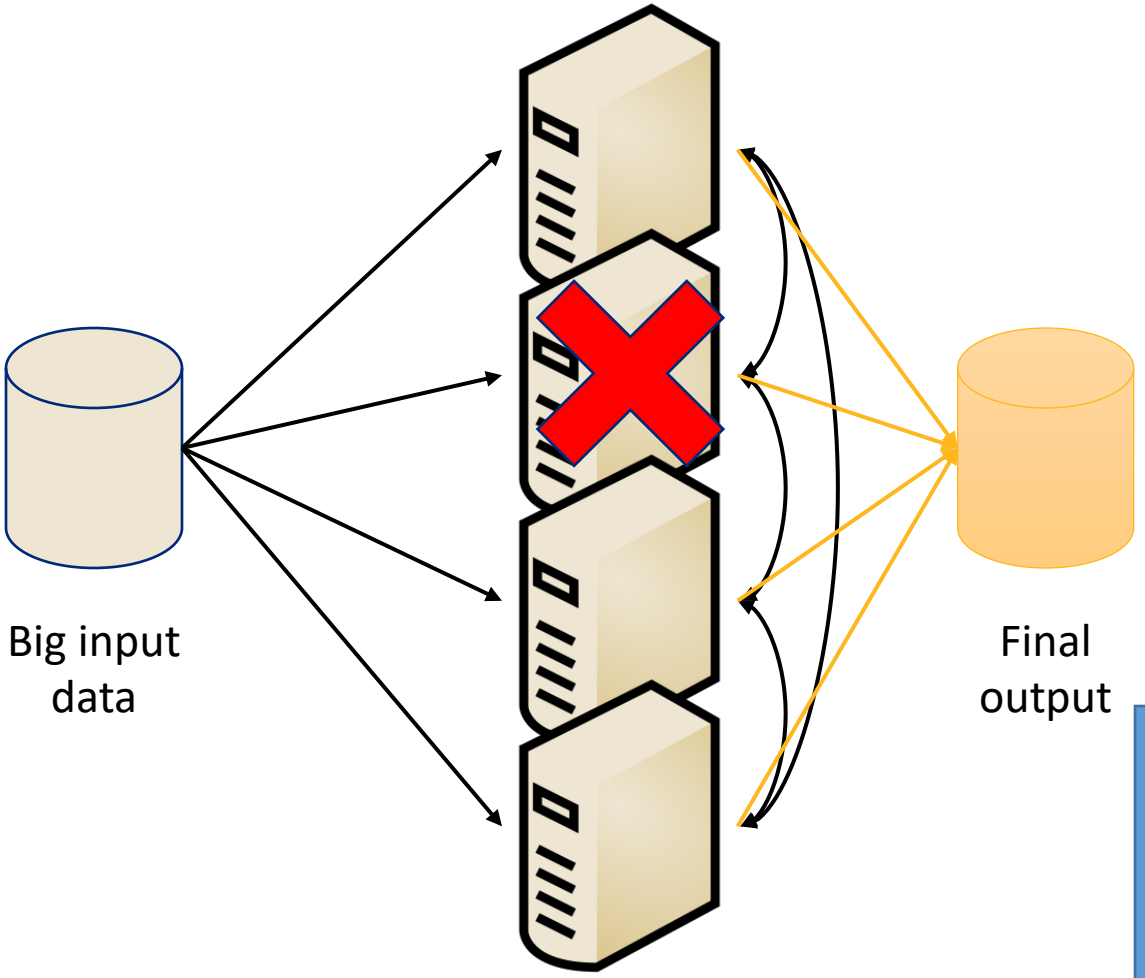
Distributed Data Processing

- The idea of distributed databases is older than you might think

  Richard Peebles, Eric G. Manning: A Computer Architecture for Large (Distributed) Data Bases. **VLDB 1975**: 405-427
- Distributed data structures and algorithms have always been around

- So, what is new?

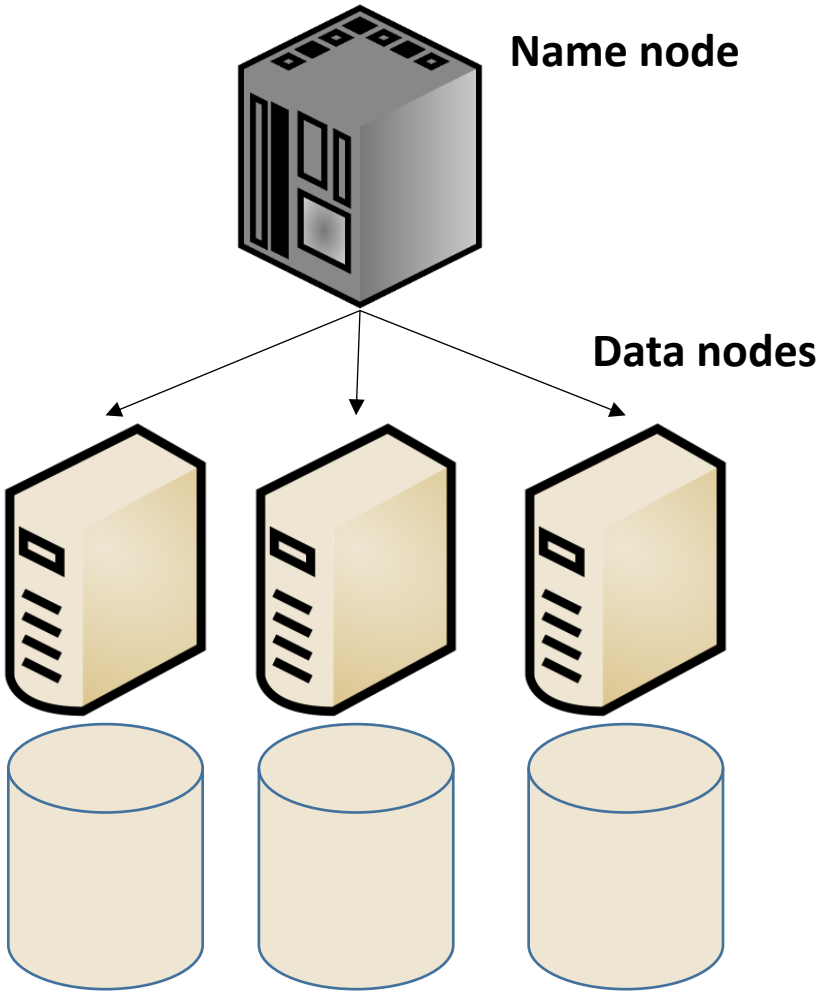
Distributed Data Processing



Data partitioning
Load balancing
Fault tolerance
Synchronization

Hadoop Distributed File System

- Partitions and stores data on multiple machines
- The same set of machines will process the data
- Accessible through simple shell commands
  - copyFromLocal
  - copyToLocal
  - …



MapReduce Computation

- A programing paradigm for expressing distributed algorithms
- Introduced by Google in 2004
  - Google File System for distributed storage
  - Google MapReduce for distributed processing
- Hadoop is the open source counterpart released in 2007 and contributed mainly by Yahoo!
  - HDFS
  - Hadoop MapReduce

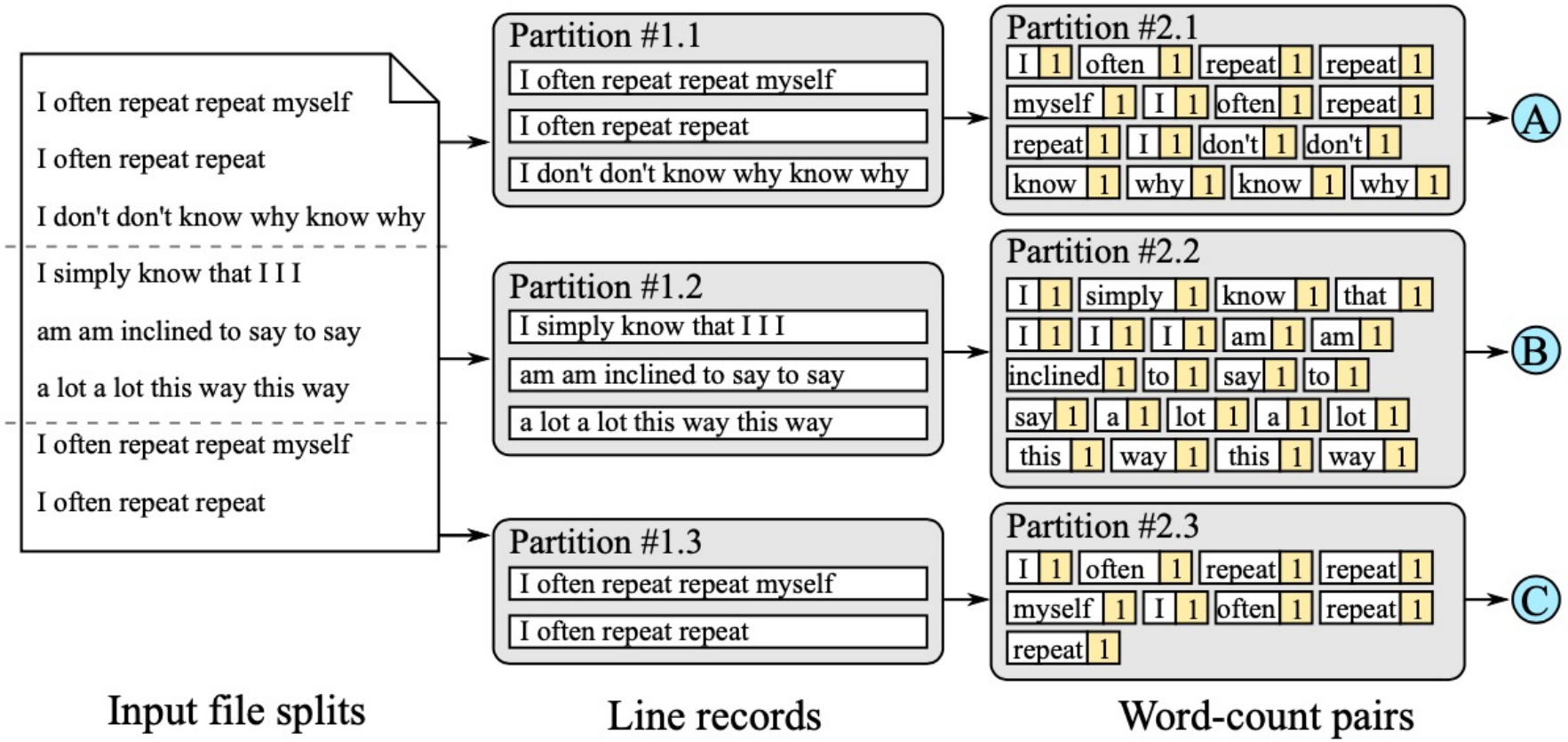
Word Count Example

Input.txt

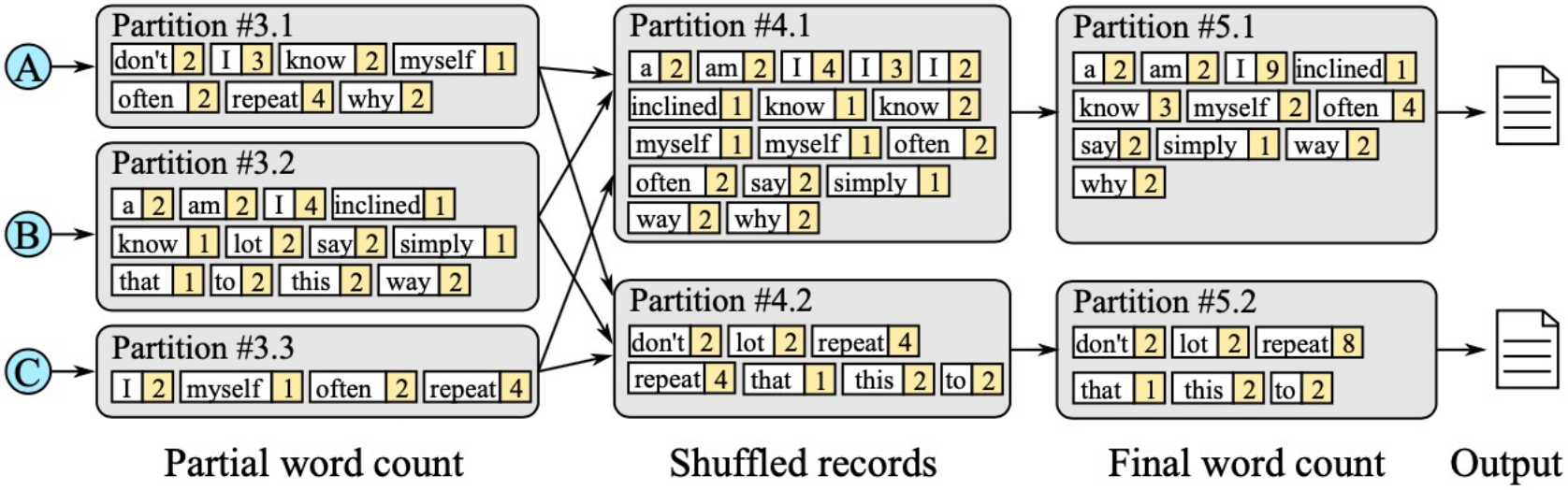
I often repeat repeat myself
I often repeat repeat
I don’t don’t know why know why
I simply know that I I I
am am inclined to say to say
a lot a lot this way this way
I often repeat repeat myself
I often repeat repeat

Word	Count
a	2
am	2
don't	2
I	9
inclined	1
know	3
lot	2
myself	2
often	4
repeat	8
say	2
simply	1
that	1
this	2
to	2
way	2
why	2

Word Count Walkthrough (1/2)



Word Count Walkthrough (2/2)



Word Count MapReduce

- Map: Line -> (word, 1) pairs
- Reduce: (word, {c}) → (word, Σc)

Complete Word Count in Hadoop

```
public static class TokenizerMapper
   extends Mapper<Object, Text, Text, IntWritable>{
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  public void map(Object key, Text value, Context context
  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

```
public static class IntSumReducer
   extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();
  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
  ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Source: https://hadoop.apache.org/docs/r3.2.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0

Spark

- Hadoop and MapReduce were a perfect research vehicle
- They helped in framing what we really want in a big data system
- Spark came as a new system designed from scratch to satisfy the real need of big data
- Simpler and more efficient

Complete Word Count in Spark

```
// In Scala shell
val lines = sc.textFile("data.txt")
val pairs = lines.flatMap(s => s.split("\\b"))
  .map(w => (w,1))
val counts = pairs.reduceByKey((a, b) => a + b)
counts.saveAsTextFile("word_count_output.txt")
```

```
//  In Java
SparkConf conf = new SparkConf();
JavaSparkContext sparkContext = new JavaSparkContext(conf);
JavaRDD<String> lines = sparkContext.textFile("data.txt");
JavaRDD<String> words = lines.flatMap(line ->
      Arrays.stream(line.split("\\b")).iterator());
JavaPairRDD<String, Integer> pairs = words.mapToPair(w -> new Tuple2(w, 1));
JavaPairRDD<String, Integer> counts = pairs.reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("word_count_output.txt");
```

Structured Big-data Processing

- A common use case in big-data is to process structured or semi-structured data
- Big-data systems were extended to support this use-case
- Examples include Pig and Hive for Hadoop, SparkSQL for Spark, and Algebricks in AsterixDB

SparkSQL Example

host	time	method	url	response	bytes
pppa006.compuserve.com	807256800	GET	/images/launch-logo.gif	200	1713
vcc7.langara.bc.ca	807256804	GET	/shuttle/missions/missions.html	200	8677

```
// Data loading
Dataset<Row> log_file = spark.read()
    .option("delimiter", "\t")
    .option("header", "true")
    .option("inferSchema", "true")
    .csv("nasa_log.tsv");
```

SparkSQL Example

```
// Filtering
Dataset<Row> ok_lines = log_file.filter("response=200");
long ok_count = ok_lines.count();
System.out.println("Number of OK lines is "+ok_count);

// Grouped aggregation using SQL
Dataset<Row> bytesPerCode = log_file.sqlContext()
    .sql("SELECT response, sum(bytes) from log_lines GROUP BY response");
```

Machine Learning on Big-data

- Many machine learning algorithms can be expressed using the basic constructs provided by big-data systems
- Many libraries were developed to express ML algorithms using big-data systems, e.g., Mahout on Hadoop and MLlib on Spark.

Basic features of MLlib

- Feature extraction, transformation, normalization, dimensionality reduction, and selection
- ML algorithms for supervised and unsupervised models
- Pipelines for constructing and evaluating ML models
- Persistence to save and load models

Example

House ID	Bedrooms	Area (sqft)	…	Price
1	2	1,200		$200,000
2	3	3,200		$350,000
…				

- Goal: Build a model that estimates the price given the house features, e.g., # of bedrooms and area

Initialization

- Similar to SparkSQL

```
val spark = SparkSession
  .builder()
  .appName("SparkSQL Demo")
  .config(conf)
  .getOrCreate()

// Read the input
val input = spark.read
  .option("header", true)
  .option("inferSchema", true)
  .csv(inputfile)
```

Transformations

```
// Create a feature vector
val vectorAssembler = new VectorAssembler()
  .setInputCols(Array("bedrooms", "area"))
  .setOutputCol("features")

val linearRegression = new LinearRegression()
  .setFeaturesCol("features")
  .setLabelCol("price")
  .setMaxIter(1000)
```

Create a Pipeline

```
val pipeline = new Pipeline()
  .setStages(Array(vectorAssembler, linearRegression))

// Hyper parameter tuning
val paramGrid = new ParamGridBuilder()
  .addGrid(linearRegression.regParam,
      Array(0.3, 0.1, 0.01))
  .addGrid(linearRegression.elasticNetParam,
      Array(0.0, 0.3, 0.8, 1.0))
  .build()
```

Cross Validation

```
val crossValidator = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(new
RegressionEvaluator().setLabelCol("price"))
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(5)
  .setParallelism(2)

val Array(trainingData, testData) =
input.randomSplit(Array(0.8, 0.2))
val model = crossValidator.fit(trainingData)
```

Apply the model on test data

```
val predictions = model.transform(testData)
// Print the first few predictions
predictions.select("price", "prediction").show(5)

val rmse = new RegressionEvaluator()
  .setLabelCol("price")
  .setPredictionCol("prediction")
  .setMetricName("rmse")
  .evaluate(predictions)
println(s"RMSE on test set is $rmse")
```

Semi-structured Big-data

- DBMS is good for tabular data
- JSON-like data
  - Nesting
  - Repetition
  - Nulls

Examples

```
{
 owner: “Alex”;
 ownerPhoneNumbers: [
  “951-555-7777”, “961-555-9999”
 ],
 contacts: [{
   name: “Chris”;
   phoneNumber: “951-555-6666”;
 }]
}
```

```
{
 owner: “Olivia”;
 ownerPhoneNumbers: [
  “951-555-2222”
 ],
 contacts: [{
   name: “Chris”;
   phoneNumber: null;
 }]
}
```

```
{
 owner: “Joe”;
 ownerPhoneNumbers: [
  “951-555-4444”, “961-555-3333”
 ]
}
```

```
{
  owner: "Jack",
  contacts: [
    {name: "Hill"},
    {name: "Bob",
     phoneNumber: "951-555-1234“}
  ]
}
```

```
{
 owner: “Violet”;
 ownerPhoneNumbers: [
  “961-555-1111”
 ]
}
```

Semi-structured Data Handling

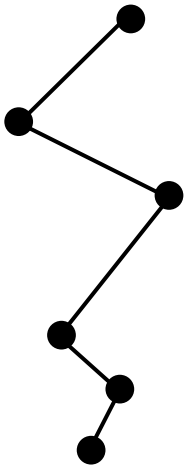
- AsterixDB: A big-data management system (BDMS)
- MongoDB: A document database
- Parquet: A column-based data format

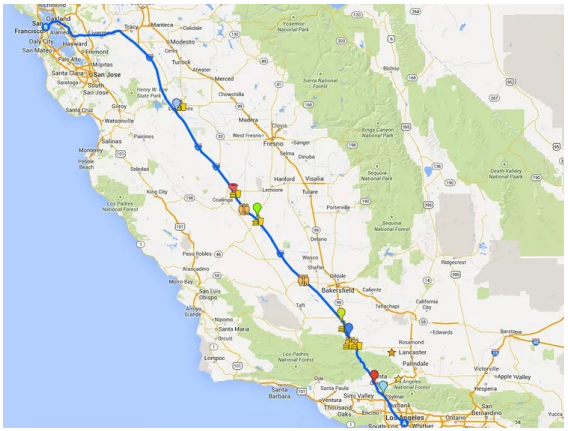
Big-Spatial Data (Vector)

Longitude

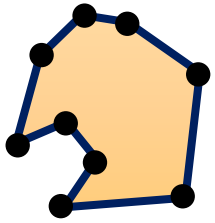
Latitude

Point





Line string or Polyline



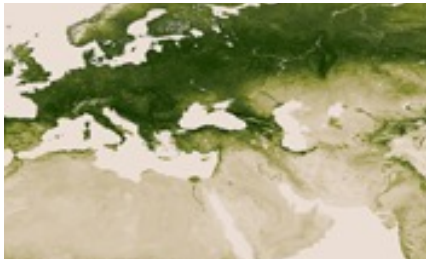
Polygon

GPS

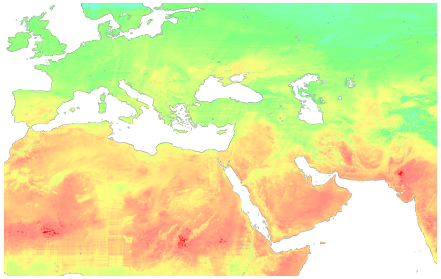
Big Spatial Data (Raster)

2D Array of values (pixels)

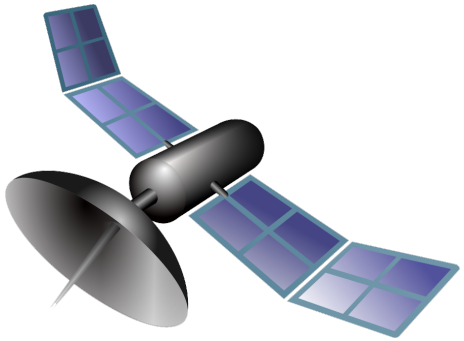
Camera



Vegetation



Temperature



Satellite

Next Steps

- We will dig deeper into each of these components
- The labs will give you hands-on experience with each component
- Assignments and mid-terms will test your understanding of how these components internally work