- **Distributed computing and apache spark**
  - How to handle massive data
    - need more hardware to store/process modern data
    - scale-up (one big machine)
      - can be very fast for medium scale problems
      - expensive, specialized hardware
      - eventually hit a wall
    - scale-out (many small machines, i.e. distributed)
      - commodity hardware, scales to massive problems
      - need to deal with network communication
      - added software complexity
  - What is apache spark
    - general, open-source cluster computing engine
    - well-suited for machine learning
      - fast iterative computations
      - efficient communication primitives
    - Simple and Expressive
      - APIs in Scala, Java, Python, R
      - Interactive shell
    - Integrated Higher-Level Libraries
      - Spark SQL, Spark streaming, MLib, GraphX
    -
- **What is Machine Learning**
  - A definition
    - Constructing and studying methods that learn from and make predictions on data
    - Broad area involving tools and ideas from various domains
      - Computer Science
      - Probability and Statistics
      - Optimization
      - Linear Algebra
  - Some Examples
    - Face Recognition
    - Link Prediction
    - Text or document classification
    - Protein structure prediction
    - Games
  - Terminology
    - Observations. Items or entities used for learning or evaluation e.g., emails
    - Features. Attributes (typically numeric) used to represent an observation, e.g. length, date, presence of keywords
    - Labels. Values/ categories assigned to observations e.g. spam, not-spam
    - Training and Test data. Observations used to train and evaluate a learning algorithm, e.g., a set of emails along with their labels
      - Training data is given to the algorithm for training
      - Test data is withheld at train time (use a validation set before test set)
  - Two Common Learning Settings
    - Supervised Learning. Learning from labeled observations
      - Labels ‘teach’ algorithm to learn mapping from observations to labels
    - Unsupervised learning. Learning from unlabeled observations

        - Learning algorithm must find latent structure from features alone
        - Can be goal in itself (discover hidden patterns, exploratory data analysis)
        - Can be means to an end (preprocessing for supervised task)
    - Examples of Supervised Learning
        - Classification: Assign a category to each item, e.g. spam detection
            - Categories are discrete
            - Generally no notion of 'closeness' in multi-class setting
        - Regression. Predict a real value for each item, e.g. stock prices
            - Labels are continuous
            - Can define 'closeness' when comparing predictions with labels
    - Examples of Unsupervised Learning
        - Clustering. Partition observations into homogeneous regions, e.g., to identify "communities" within large groups of people in social networks
        - Dimensionality Reduction. Transform an initial feature representation into a more concise representation, e.g., representing digital images
- **Typical Supervised Learning Pipeline**
    - Obtain Raw Data => Feature Extraction
        - Initial observations can be in arbitrary format
        - We extract features to represent observations
        - We can incorporate domain knowledge
        - We typically want numeric features
        - Success of entire pipeline depends on choosing good descriptions of observations
    - Feature Extraction <=> Unsupervised Learning
        - Preprocessing step for supervised learning
    - Obtain Raw Data => Feature Extraction => Supervised Learning
        - Train a supervised model using labeled data, e.g., Classification or Regression model
    - Obtain Raw Data => Feature Extraction => Supervised Learning => Evaluation
        - How do we determine the quality of the model we've just learned?
        - We can evaluate it on test / hold-out data, i.e., labeled data not used for training
        - If we don't like the results, we iterate
            - Feature Extraction => Supervised Learning => Evaluation
    - Obtain Raw Data => Feature Extraction => Supervised Learning => Evaluation => Predict
        - Once we're happy with our model, we can use it to make predictions on future observations, i.e., data without a known label
- **Sample Classification Pipeline**
    - Classification
        - Goal: Learn a mapping from observations to discrete labels given a set of training examples (supervised learning)
        - Example: Spam Classification
            - Observations are emails
            - Labels are {spam, not-spam} (Binary Classification)
            - Given a set of labeled emails, we ant to predict whether a new email is spam or not-spam
    - Other examples
        - Fraud detection: User activity -> {Fraud, not fraud}
        - Face detection: Images -> set of people
        - Link prediction: Users -> {Suggest link, don't suggest link}
        - Clickthrough rate prediction: User and ads -> {click, no click}
    - Classification Pipeline

- Obtain Raw data
  - Raw data consists of a set of labeled training observations
- Feature extraction
  - typically transforms each observations into a vector of real numbers (features)
  - success or failure of a classifier often depends on choosing good descriptions of observations
  - “Bag of Words”
    - Observations are documents
    - Build vocabulary
    - Derive feature vectors from Vocabulary
- Supervised Learning: Train classifier using training data
  - Common classifiers include Logistic regression, SVMs, Decision trees, Random Forests, etc.
  - Training (especially at scale) often involves iterative computations, e.g., gradient descent
  - Logistic Regression
    - Goal: Find linear decision boundary
      - Parameters to learn are feature weights and offset
      - Nice probabilistic interpretation
      - Covered in more detail later in course
- Evaluation
  - How can we evaluate the quality of our classifier?
    - We want good predictions on unobserved data
      - ‘Generalization’ ability
    - Accuracy on training data is overly optimistic since classifier has already learned from it
      - we might be ‘overfitting’
  - Overfitting and Generalization
    - Fitting training data does not guarantee generalization, e.g., lookup table
    - Left: perfectly fits training samples, but it is complex / overfitting
    - Right: misclassifies a few points, but simple / generalizes
    - Occam’s razor
  - How can we evaluate the quality of our classifier?
    - Idea: Create test set to simulate unobserved data
  - Evaluation: Split dataset into training / testing datasets
    - Train on training set (don’t expose test set to classifier)
    - Make predictions on test set (ignoring test labels)
    - Compare test predictions with underlying test labels
    - Various metrics to compare predicted and true labels
    - Accuracy is common for classification
  - Predict: Final classifier can then be used to make predictions on future observations, e.g., new emails we receive

- Linear algebra review
  - Matrices
  - Vectors
    - a matrix with many rows and one column
  - Transpose
    - swap the rows and columns of the matrix
  - Addition and subtraction
    - Element wise operations

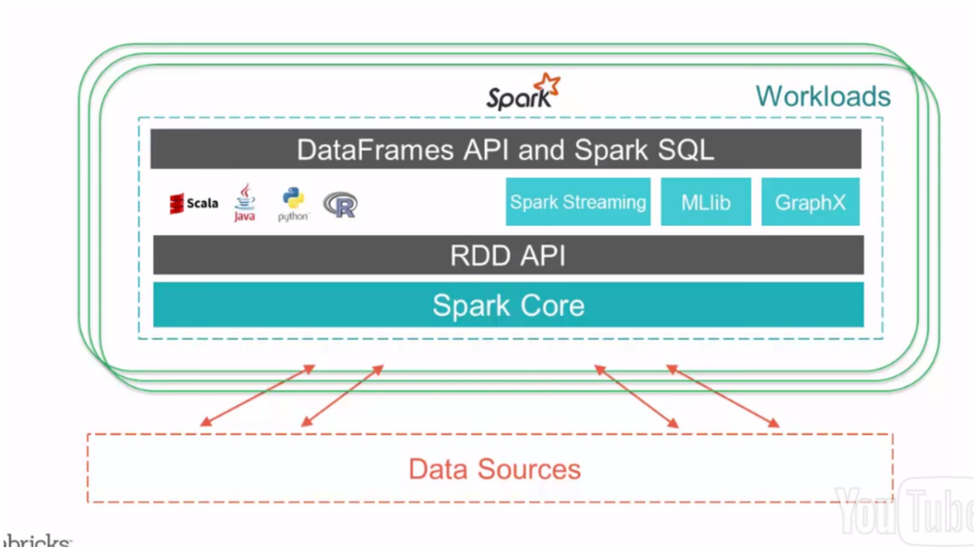
    - matrices must have the same dimensions
  - Matrix scalar multiplication
    - we multiply each matrix element by the scalar value
  - Scalar Product
    - Function that maps two vectors to a scalar
    - Performs pairwise multiplication of vector elements
    - Take the sum
    - The two vectors must be the same dimension
    - Also known as dot product or inner product
  - Matrix-Vector Multiplication
    - Matrix A and vector w
      - output y
    - y_i equals the scalar product between the with row of A and w
    - we repeat for each row of A, so if A has n rows, does y
    - To perform inner products, # columns in A must equal # rows of w
  - Scalar Product Revisited
    - Special case of Matrix-Vector Multiplication
      - Vectors assumed to be in column form (many rows, one column)
      - Transposed vectors are row vectors
      - Common notation for scalar product: x'w
  - Matrix Matrix Multiplication
    - Takes first row of first matrix scalar product of first column gives first row column for new matrix then second …
    - Take the second row …
    - To perform inner products the # columns in A must equal # rows of B
    - m x n * n x p = m x p
  - Outer Product
    - Special case of Matrix Matrix Multiplication involving two vectors
    - C_ij is the product of it entry of x and the nth entry of w
  - Identity Matrix
    - One is the scalar multiplication identity
    - Identity matrices are square, with ones on the diagonal entries
  - Inverse Matrix
    - Multiplying a matrix by its inverse results in the identity matrix
    - only a square matrix can have an inverse
  - Euclidean Norm for Vectors
    - The magnitude / length of a scalar is its absolute value
    - Vector norms generalize this idea for vectors
    - The Euclidean norm for x element R^m is denoted by llxll_2

        - $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_m^2}$
        - Equals absolute value when m=1
        - Related to scalar product: $\|\mathbf{x}\|_2^2 = \mathbf{x}^\top \mathbf{x}$

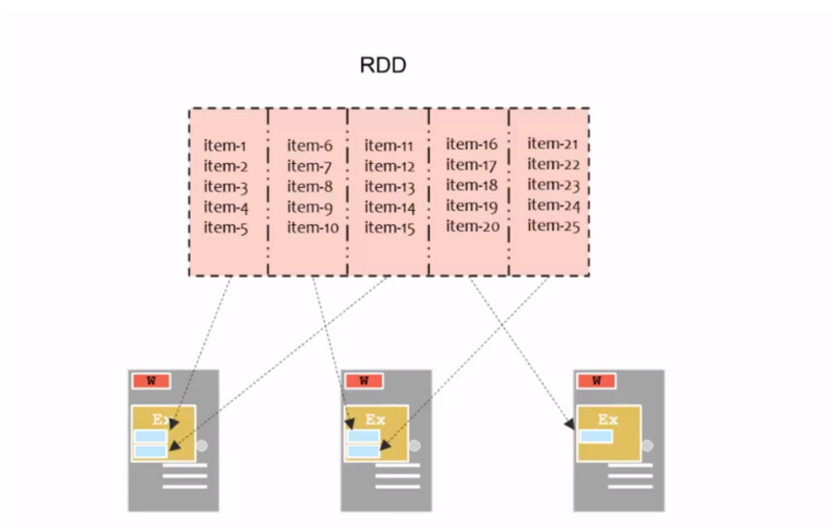
- **Big O Notation for Time and Space Complexity**
  - Big O Notation
    - describes how algorithms respond to changes in input size
      - Both in terms of processing time and space requirements

        - We refer to complexity and Big O notation synonymously
    - Required space proportional to units of storage
        - typically eight bytes to store a floating point number
    - Required time proportional to number of basic operations
        - arithmetic operations (+,-,x,/), logical tests (<,>,==)
- Notation: $f(x) = O(g(x))$
    - can describe an algorithm's time or space complexity
        - Informal definition: f does not grow faster than g
        - Formal definition: $|f(x)| <= C|g(x)|$ for all $x > N$
        - Ignores constants and lower-order terms
            - For large enough x, these terms won't matter
- $O(1)$ Complexity
    - Constant time algorithms perform the same number of operations every time they're called
    - E.g. performing a fixed number of arithmetic operations
    - Similarly, constant space algorithms require a fixed amount of storage every time they're called
        - E.g. storing the results of a fixed number of arithmetic operations
- $O(n)$ Complexity
    - Linear time algorithms perform a number of operations proportional to the number of inputs
        - E.g. adding two n-dimensional vectors requires $O(n)$ arithmetic operations
    - Similarly, linear space algorithms require storage proportional to the size of the inputs
        - E.g. adding two n-dimensional vectors results in a new n-dimensional vector which requires $O(n)$ storage
- $O(n^2)$ Complexity
    - Quadratic time algorithms perform a number of operations proportional to the square of the number of inputs
        - E.g. outer product of two n-dimensional vectors requires $O(n^2)$ multiplication operations (one per each entry of the resulting matrix)
    - Similarly, quadratic space algorithms require storage proportional to the square of the size of the inputs
        - E.g. outer product of two n-dimensional vectors requires $O(n^2)$ storage (one per each entry of the resulting matrix)
- Time and Space Complexity Can Differ
    - Inner product of two n-dimensional vectors
        - $O(n)$ time complexity to multiply n pairs of numbers
        - $O(1)$ space complexity to store result (which is scalar)
    - Matrix inversion of an $n \times n$ matrix
        - $O(n^3)$ time complexity to perform inversion
        - $O(n^2)$ space complexity to store result
- Matrix-Vector Multiply
    - Goal: multiply an $n \times m$ matrix an $m \times 1$ vector
    - Computing result takes $O(nm)$ time
        - There are n entries in the resulting vector
        - each entry computed via dot product between two m-dimensional vectors (a row of input matrix and input vector)
    - Storing result takes $O(n)$ space
        - The result is an n-dimensional vector

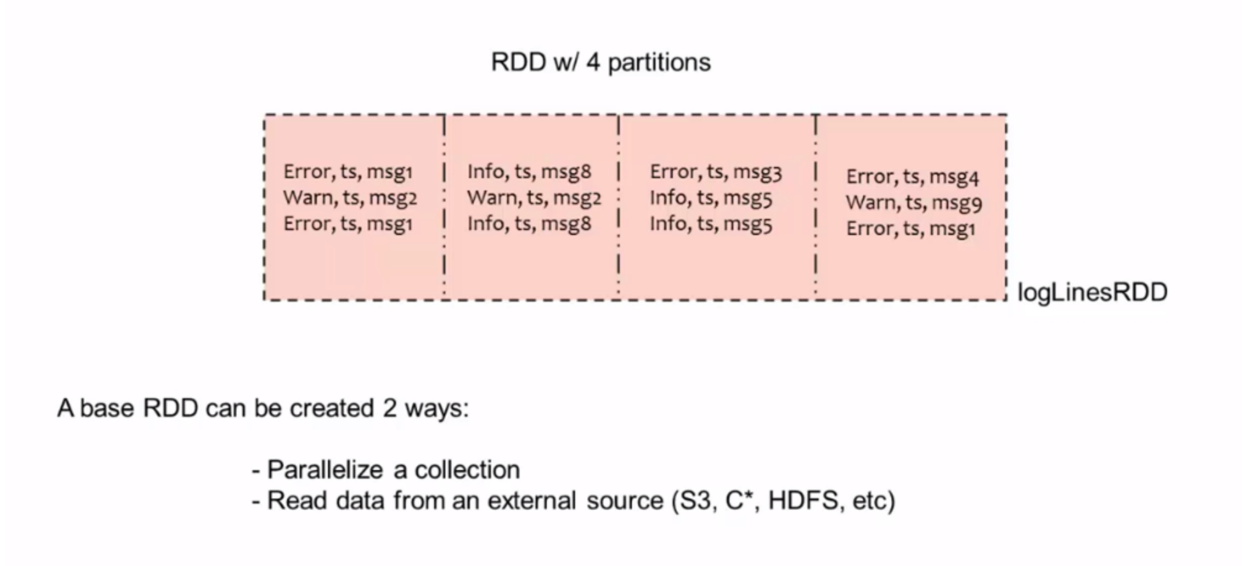
- E.g. Matrix-Matrix Multiply
  - Computing result takes O(npm) time
    - There are np entries in the resulting matrix
    - Each entry computed via dot product between two m-dimensional vectors
  - Storing takes O(np) total
- **RDD Fundamentals**



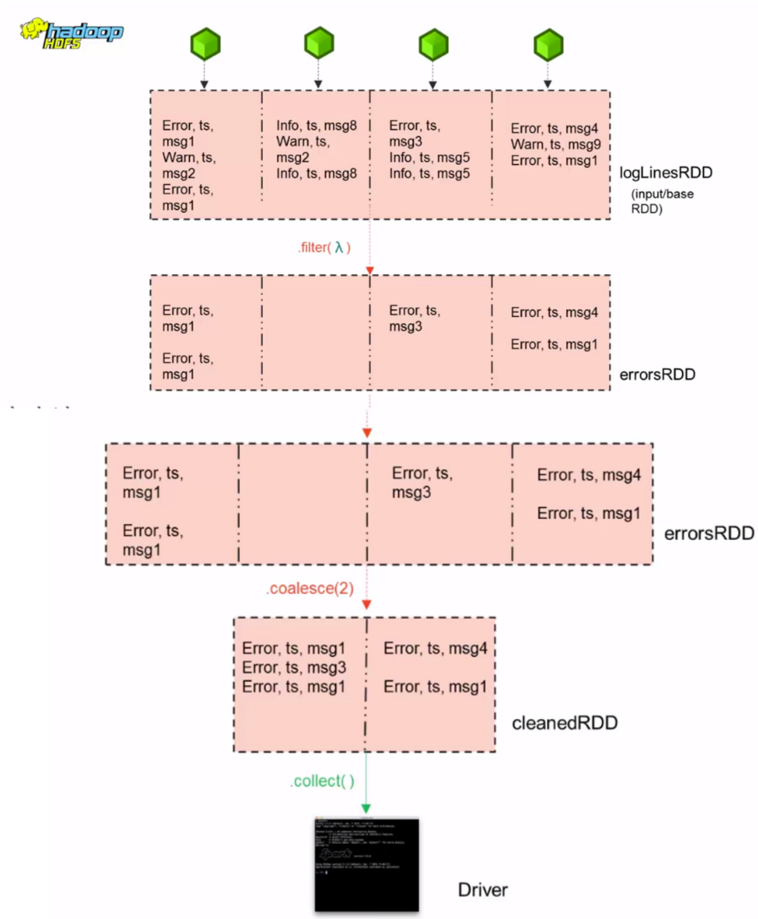
  - Driver program to worker machines
- Resilient Distributed Datasets (RDDs)
  - Write programs in terms of operations on distributed data
  - Partitioned collections of objects spread across a cluster
  - Diverse set of parallel transformations and actions
  - Fault tolerant



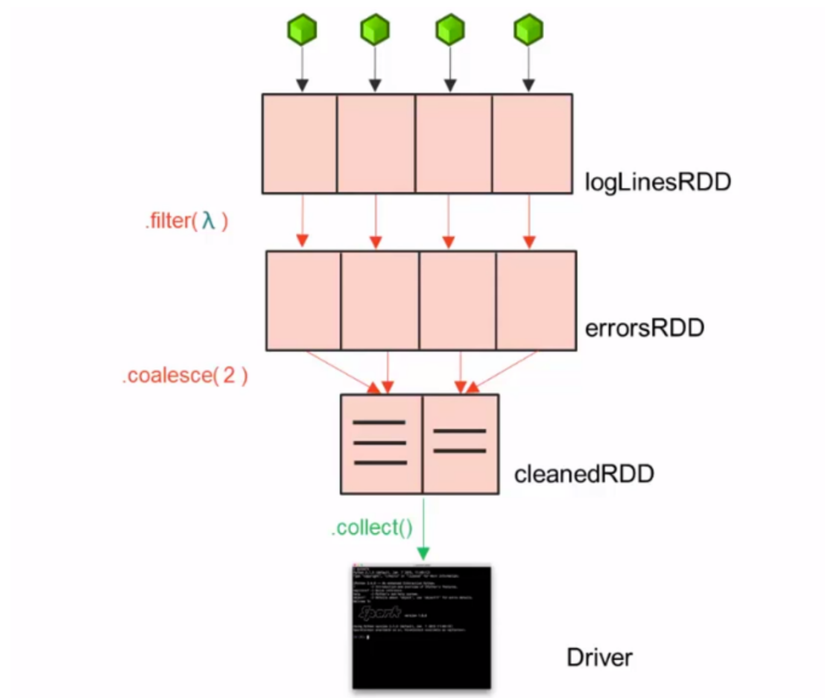
-
-



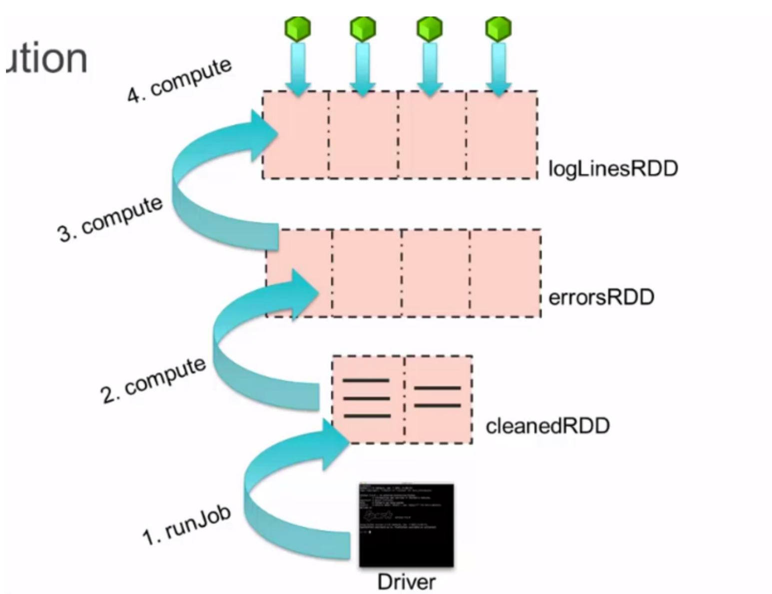
- Create a Base RDD
  - Parallelize in Python
    - wordsRDD = sc.parallelize([“fish”,”cats”,”dogs”])
    - Parallelize - Take an existing in-memory collection and pass it to SparkContext’s parallelize method
  - Read a local txt file in python
    - linesRDD = sc.textFile(“/path/to/README.md”)
    - Read from Text File
    - There are other methods to read data from HDFS, C*, S3, HBase, etc.
- Operations on Distributed Data
  - Two types of operations: transformations and actions
  - Transformations are lazy (not computed immediately)
  - Transformations are executed when an action is run
  - Persist (cache) distributed data in memory or disk



- DAG

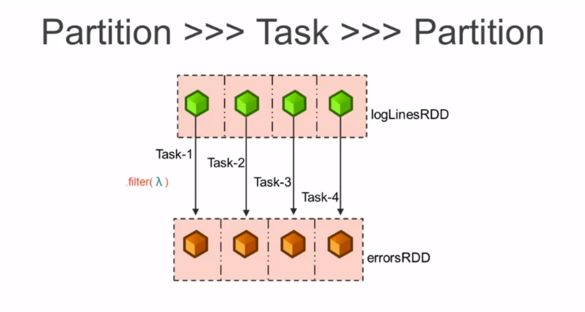


- Execution



• Collect operation completed

- Partition >>> Task >>> Partition



- Lifecycle of an RDD-based Spark Program
  - Create base RDD
  - Chain together transformations
    - example filter map
  - Cache intermediate RDDs
  - Perform actions
    - count collect to start parallel computation
- Transformations - are lazy and will not be computed until an action is called
  - Map()
  - intersection()
  - zipWithIndex()
  - flatMap()
  - distinct()
  - pipe()
  - filter()
  - groupByKey()
  - coalesce()
  - mapPartitions()
  - reduceByKey()
- Actions - result in a spark job being launched and cause any related transformations to be computed
  - reduce()
  - collect()
  - count()
  - first()
  - take()
  - takeOrdered()
  - saveAsTextFile()
- RDDs vs DataFrames
  - RDDs provide a low-level interface into Apache Spark
  - DataFrames have a schema
  - DataFrames are cached using Tungsten format
  - DataFrames are optimized via Catalyst
  - DataFrames are built on top of the RDD and core APIs
-
-
-
-

-
-
-
-
-
- sdf