DSECL ZG 522: Big Data Systems

Session 13: Spark - Part 1

Janardhanan PS

Professor

janardhanan.ps@wilp.bits-pilani.ac.in

Topics for today

- **Introduction**
- Getting started
    - ✓ Setup
    - ✓ Sample programs

BigData computing engines

- Batch processing
  - Hadoop
- Stream processing
  - Storm
- Interactive processing
  - Tez, Impala
- Graph processing
  - Neo4j

Is it possible to create a general purpose computing engine for Big Data analysis ?

Spark

- Cluster computing platform
  - ✓ Designed to be fast and general purpose
- Speed
  - ✓ Is important in processing large datasets as it creates difference when data is being explored
  - ✓ Extends MapReduce model to efficiently support more types of computations
  - ✓ Runs computations in memory
- Generality
  - ✓ Covers a wide variety of workloads which earlier required different distributed systems
  - ✓ Including
    - ❖ Batch applications
    - ❖ Iterative algorithms
    - ❖ Interactive queries
    - ❖ Streaming
  - ✓ Easy and inexpensive to combine different processing types in data pipelines

Spark language support

- Spark handles highly accessible, simple APIs in
  - ✓ Java
  - ✓ Python
  - ✓ Scala
  - ✓ SQL

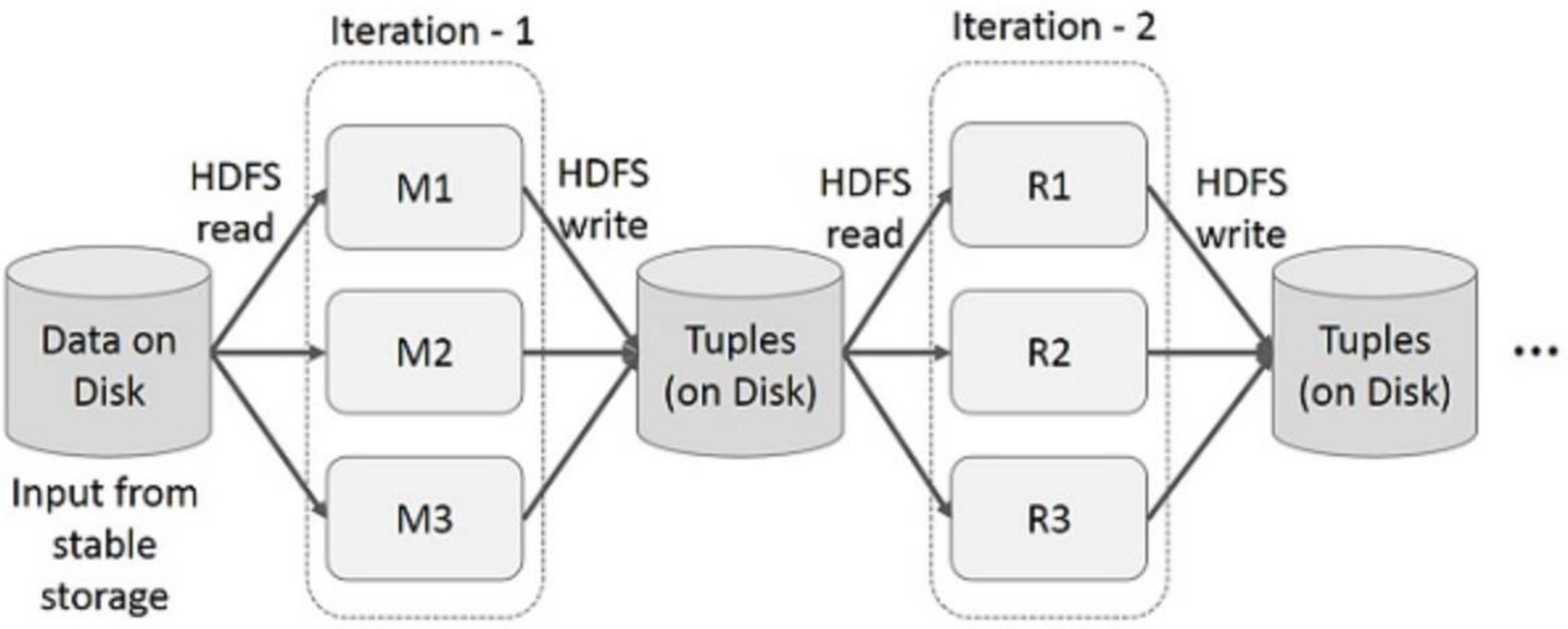
- Integrates easily with other big data tools / platforms like
  - ✓ Hadoop (HDFS) and other ecosystem tools
  - ✓ Kafka
  - ✓ AWS S3
  - ✓ Cassandra

Hadoop and Spark

- Spark introduced to address speed issue of Hadoop computation
- Not a modified version of Hadoop but can use Hadoop as an option
  - For cluster management using YARN
  - For HDFS storage
- Spark has own in-memory cluster computation engine using RDD
- Supports more than MapReduce
  - SQL, Streaming, ML, Graph

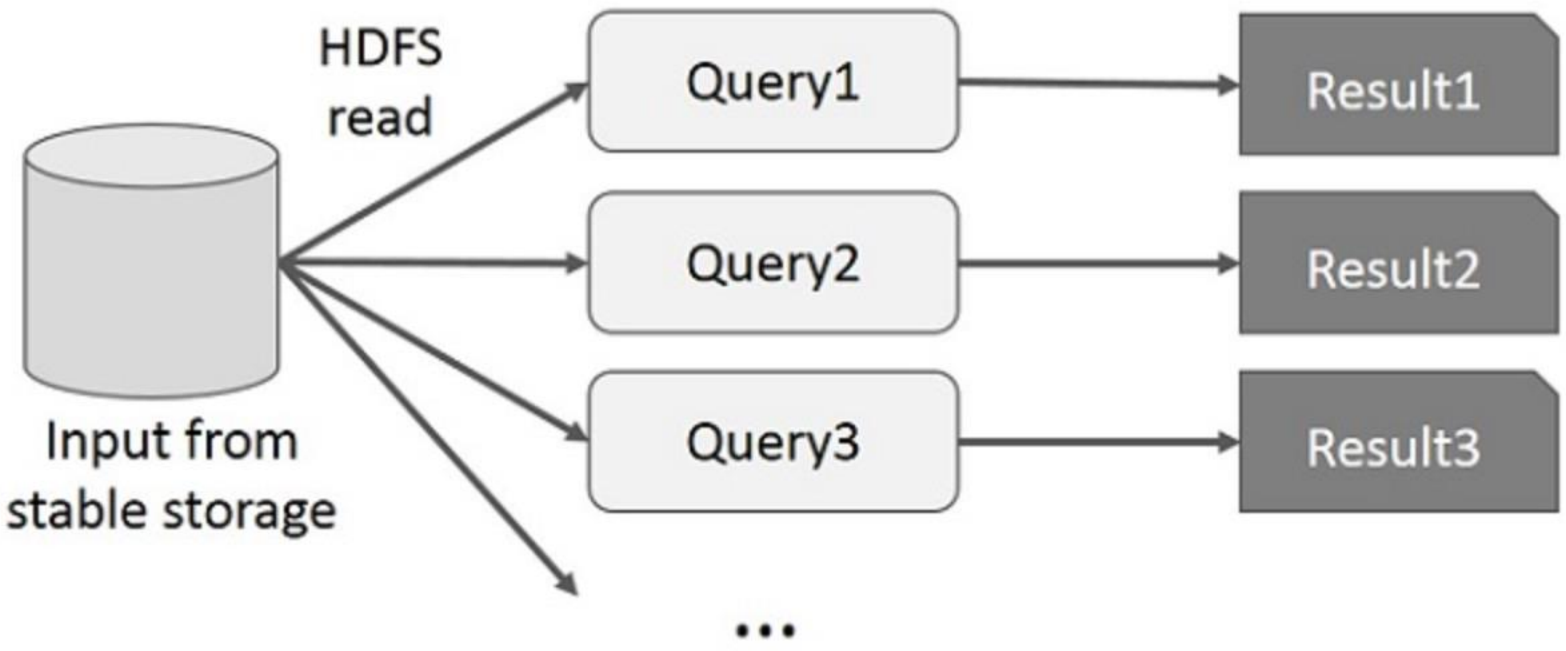
Hadoop MapReduce inefficiencies (1)

- Iterative applications incur significant overheads
  - Serialization
  - Disk I/O
  - Replication



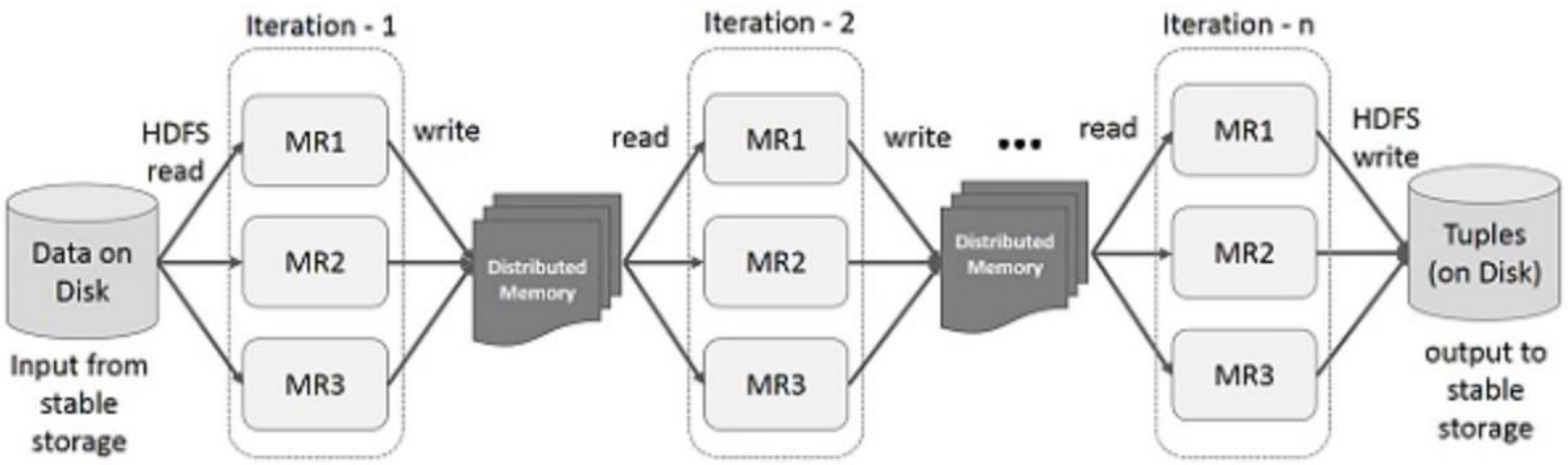
Hadoop MapReduce inefficiencies (2)

- Interactive operations incur significant overheads with each query reading from stable storage and disk I/O will dominate application execution time



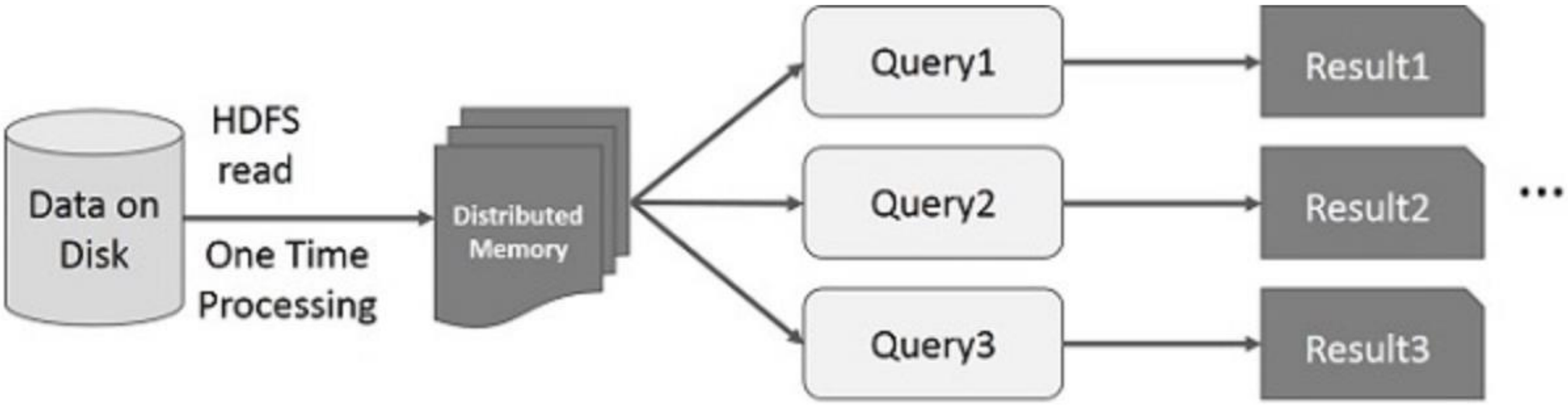
In-memory computing using RDD (1)

- Store intermediate results in distributed memory without complicating user programming
- Spill-over can be transparently stored on disk



In-memory computing using RDD (2)

- Store intermediate results in distributed memory without complicating user programming
- Spill-over can be transparently stored on disk



What is an RDD: Resilient Distributed Dataset

- Fundamental data structure in Spark
- Immutable distributed collection of objects
- Can be created from other RDDs, or
  - 'parallelise' an existing collection in driver program

```
val data = Array(1,2,3,4,5,6,7,8,9,10)
val rdd = sc.parallelize(data)
println("Number of Partitions: "+rdd.getNumPartitions)
println("Action: First element: "+rdd.first())
val rdd2 = rdd.max()
```

  - reference a data set in an external storage: HDFS, HBase etc.

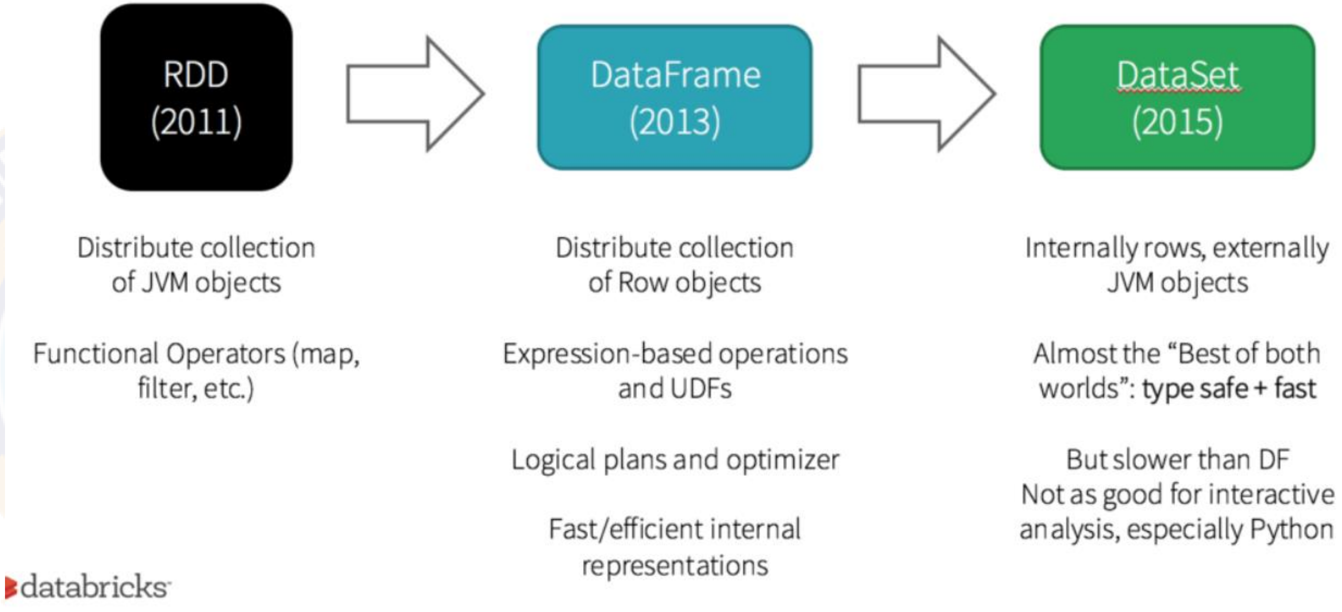
```
 val dstfile = sc.textFile("sample.txt")
dstfile.count()
```

Why is it called RDD

- Resilient
  - A lineage graph of operations helps to reconstruct when a node fails and part of an RDD is lost
- Distributed
  - Each RDD is divided into logical partitions for parallel computation on cluster
- Dataset
  - Can contain any type of objects depending on language used

Options in Spark APIs

- RDD useful when
  - dealing with unstructured data
  - don't want to impose schema
  - low level operations on data
  - not interested in optimisations done for structured data

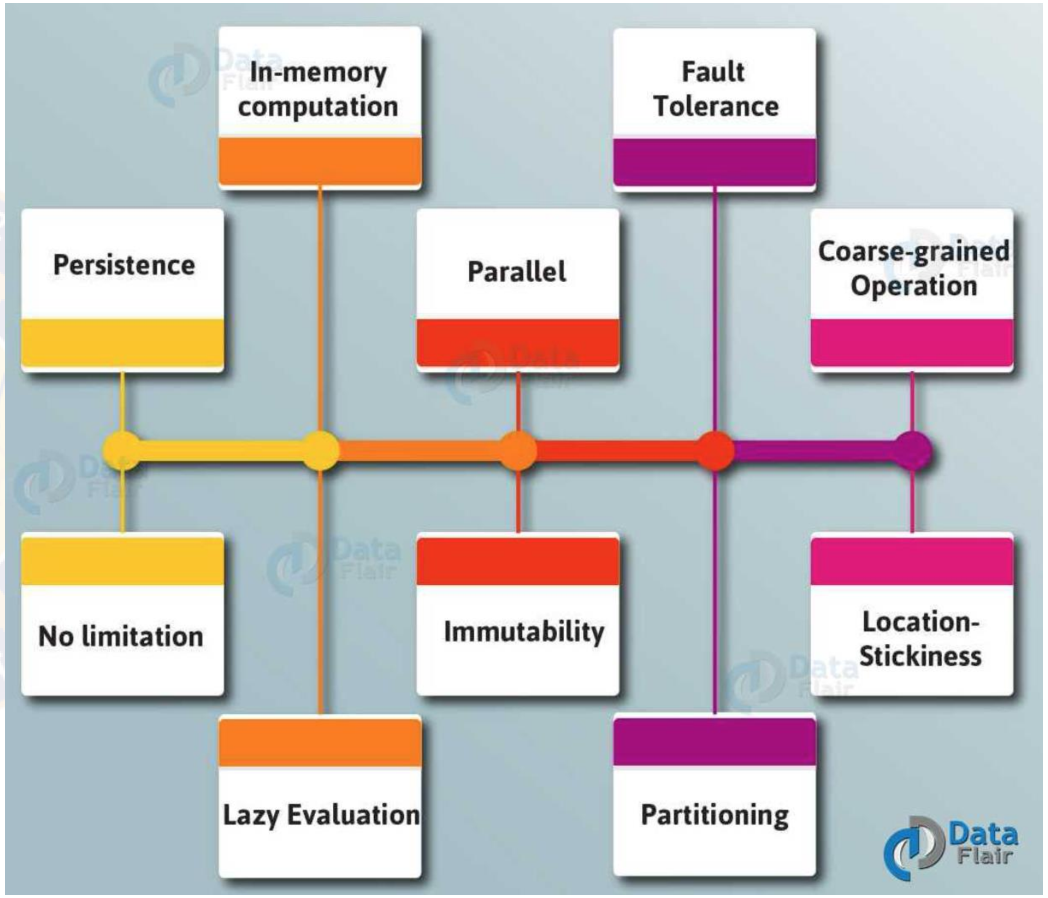


Named columns
Fastest option

Added type-safety

RDD features

- Keep data in memory as much as possible
- Evaluate only when an action triggers
- A failed lost RDD partition on a worker can be recovered from lineage of operations
- Cannot change once created
- Persist in memory or storage for reuse
- Parallelism through partitioning
- Put tasks closer to data location
- Apply operations to entire set of data at coarse grain and not one data item within RDD



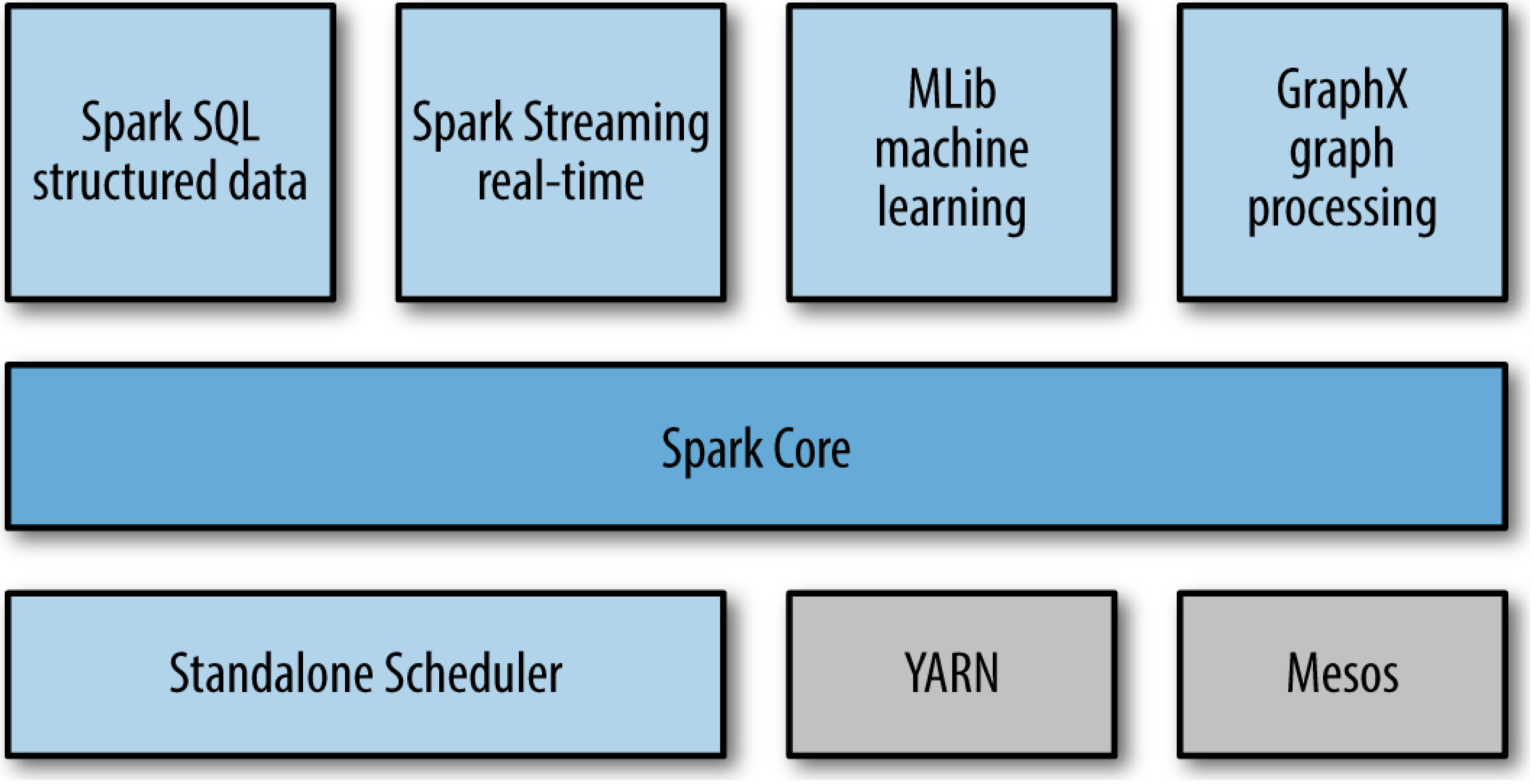
RDD limitations

- For structured data
  - RDDs do not exploit any optimizers
  - Better to use DataFrame or DataSets
- Since RDDs are in-memory JVM objects there are garbage collection and Java serialisation overheads
- Spill over data is put on disks which slows down performance, hence machines need to have enough memory given the data size and analysis

RDD and Distributed Shared Memory (DSM)

- Grain of R/W operation
  - RDD is coarse grained as it works at dataset level whereas DSM is at specific data item level
- Consistency
  - Immutable RDDs are trivially consistent whereas DSM makes sure of consistency if programmer follows a set of rules
- Fault recovery
  - New RDDs are created on each transformation. So following lineage of operations RDDs can be recovered. DSMs need checkpointing / rollback.
- Straggler mitigation: Problem of having slow tasks slow down end to end performance
  - RDDs make it little easier with backup tasks whereas in DSM it is difficult
- Out-of-memory behaviour
  - Spill over data goes to on-disk RDDs gradually degrading performance whereas in DSM system swaps may lead to instability

Unified Stack



Ref : https://www.oreilly.com/library/view/learning-spark/9781449359034/ch01.html

Spark Core

- Contains the basic functionality of Spark, including components for
  - ✓ task scheduling
  - ✓ memory management
  - ✓ fault recovery
  - ✓ interacting with storage systems etc.

- Home to the API that defines resilient distributed datasets (RDDs), which are Spark's main programming abstraction

- Provides many APIs for building and manipulating these collections

Spark SQL

- Spark's package for working with structured data
- Allows querying data via SQL as well as the Apache Hive variant of SQL
  - ✓ called the Hive Query Language (HQL)
- Allows developers to intermix SQL queries with the programmatic data manipulations supported by RDDs in
  - ✓ Python
  - ✓ Java
  - ✓ Scala

  all within a single application
- Tight integration with the rich computing environment provided by Spark makes Spark SQL unlike any other open source data warehouse tool

Spark Streaming

- Spark component that enables processing of live streams of data
- Examples of data streams include web servers log files
- Provides an API for manipulating data streams that closely matches the Spark Core's RDD API
- Designed to provide the same degree of fault tolerance, throughput, and scalability as Spark Core

MLlib

- Built in library containing common machine learning (ML) functionality
- Provides multiple types of machine learning algorithms, including
  - ✓ Classification
  - ✓ Regression
  - ✓ Clustering
  - ✓ Collaborative filtering
  - ✓ Supporting functionality such as data import and model evaluation
- Provides some lower-level ML primitives, including a generic gradient descent optimization algorithm
- All of these methods are designed to scale out across a cluster

GraphX

- Library for
  - ✓ manipulating graphs (e.g. a social network's relations graph)
  - ✓ performing graph-parallel computations
- Extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge
- Provides
  - ✓ various operators for manipulating graphs (e.g., subgraph and mapVertices)
  - ✓ library of common graph algorithms (e.g., PageRank and triangle counting)

Apache Spark GraphX

Cluster Managers

- Spark is designed to efficiently scale up from one to many thousands of compute nodes
- Can run over a variety of cluster managers, including
  - ✓ Hadoop YARN
  - ✓ Apache Mesos
  - ✓ Simple cluster manager included in Spark itself called the Standalone Scheduler
  - ✓ Kubernetes
- If you are just installing Spark on an empty set of machines
  - ✓ the Standalone Scheduler provides an easy way to get started
- If you already have a Hadoop YARN or Mesos cluster,
  - ✓ Spark's support for these cluster managers allows your applications to also run on them

- Launch Overheads of Spark Applications on Standalone and Hadoop YARN Clusters
  https://link.springer.com/chapter/10.1007/978-981-15-5558-9_5

Use cases for Spark

- Banking
  - Customer segmentation, credit risk assessment, targeted advertisement on products
- e-commerce
  - Clustering on streaming data for identifying trends and providing recommendations
- Travel
  - Provide personalised hotel recommendations, restaurant reservations, e.g. TripAdvisor
- Media and entertainment
  - Automatic game complexity level setting by mining patterns in real-time, advertisement using real-time analysis often combined with data from MongoDB, e.g. Pinterest, Yahoo
- Healthcare
  - Patient record analysis with past clinical data to proactively avoid hospitalisation, genomic sequencing
- Fog computing / IoT
  - Sensor data analysis on the edge

When not to use Spark

- Large batch processes with high memory requirements
- Multi user analysis environments where concurrent demand for memory is high
    - May not scale with number of concurrent users

Topics for today

- Introduction
- **Getting started**
    - ✓ Setup
    - ✓ Sample programs

Getting Started

- Spark can be used from Python, Java or Scala
- Spark is written in Scala and runs on JVM
- Spark can be run in local mode (standalone) or cluster mode
- Spark can be run on both Windows and Unix like systems

- Requirements
  - ✓ It’s easy to run locally on one machine — all you need is to have java installed on your system PATH, or the JAVA_HOME environment variable pointing to a Java installation
  - ✓ Spark runs on Java 8, Python 2.7+/3.4+ and R 3.1+. For the Scala API, Spark 2.4.5 uses Scala 2.12. You will need to use a compatible Scala version (2.12.x).

Getting Started (2)

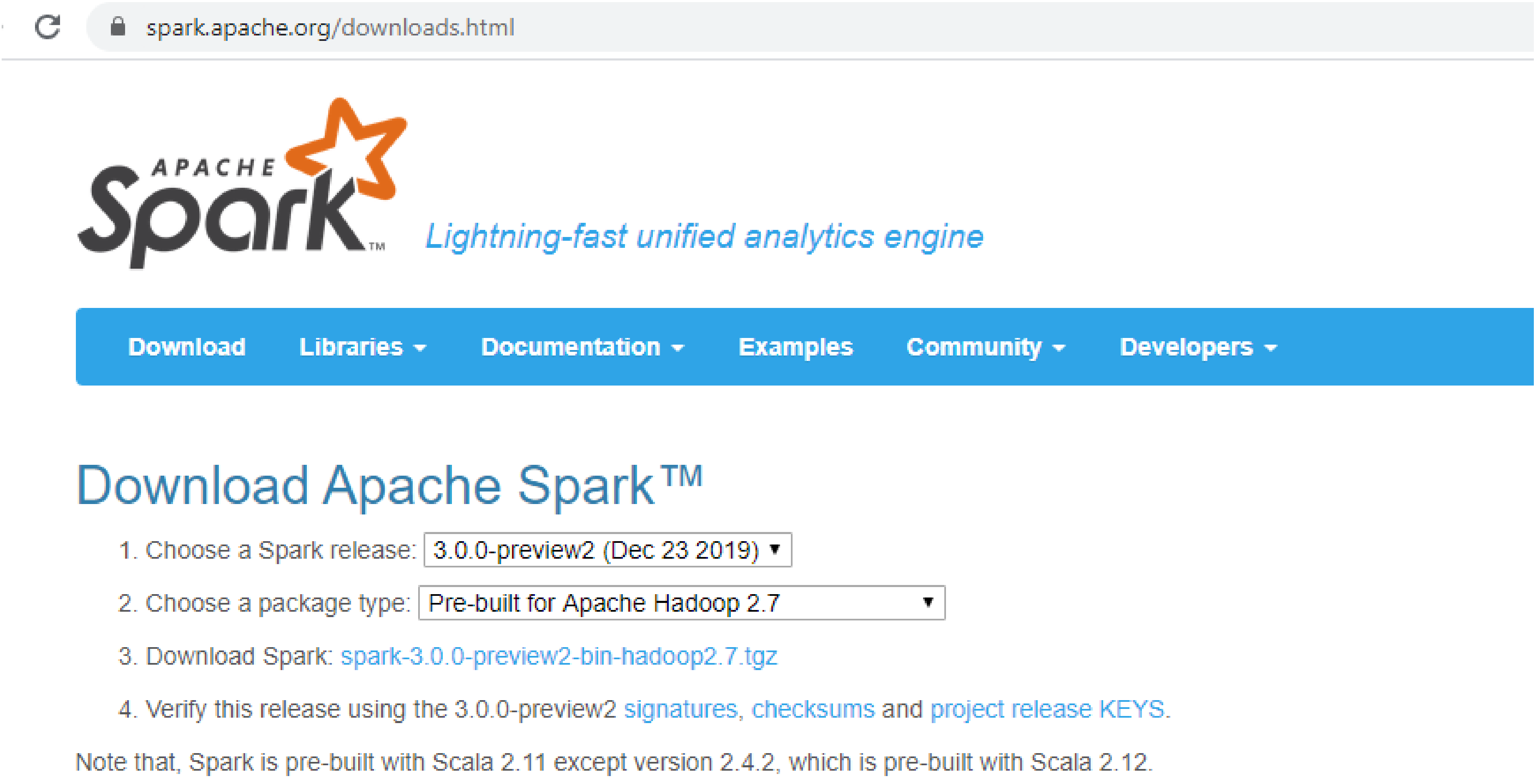
Download

- Needs to download and unpack the tarball
- Go to the Spark homepage to download the Spark release
  - ✓ https://spark.apache.org/downloads.html

- Steps -
  - ✓ Choose a Spark release:
  - ✓ Choose a package type:
  - ✓ Download Spark: spark-3.0.0-preview2-bin-hadoop2.7.tgz
  - ✓ Verify this release using the 3.0.0-preview2 signatures, checksums and project release KEYS.

Getting Started (3)

Download



Getting Started (4)

Unpack

- The tarball .tgz will get downloaded
- Open the terminal and unzip the tar ball
- Result into new directory with same name as tarball
- Change into the directory and list the content of the same

Applications   Places   Terminal   Wed 12:35

csishydlab@apache-spark:~/spark-2.4.4-bin-hadoop2.7

File   Edit   View   Search   Terminal   Help

```
[csishydlab@apache-spark ~]$ ls Downloads/
CentOS-7-x86_64-DVD-1908.torrent     Spark-DataFrame(1).html                               spark-streaming-kafka-0-8-assembly_2.11-2.4.4.jar
CentOS-7-x86_64-Everything-1908.iso  Spark-DataFrame.html                                  spark-streaming-kafka-assembly_2.10-1.4.1.jar
kafka_2.12-2.4.0.tgz                 Spark-DataFrame.ipynb
kafka_2.12-2.4.1.tgz                 spark-streaming-kafka-0-8-assembly_2.11-2.4.4(1).jar
[csishydlab@apache-spark ~]$ cd ~
[csishydlab@apache-spark ~]$ ls
data       kafka_2.12-2.4.0      output       scala-2.10.1.tgz                spark-streaming-kafka-0-8-assembly_2.11-2.4.4.jar  Templates
Desktop    kafka_2.12-2.4.0.tgz  Pictures     spark-2.4.4-bin-hadoop2.7       spark-streaming-kafka-assembly_2.10-1.4.1.jar      Videos
Documents  logs                  Public       spark-2.4.4-bin-hadoop2.7.tgz   spark-warehouse
Downloads  Music                 python_code  spark_kafka_demo.logs           streaming_kafka_demo.py
[csishydlab@apache-spark ~]$ cd spark-2.4.4-bin-hadoop2.7/
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$ ls
bin   data      jars        LICENSE   NOTICE  R          regression_metrics_example.py  sample_linear_regression_data.txt  yarn
conf  examples  kubernetes  licenses  python  README.md  RELEASE                        sbin
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$
```

Getting Started (5)

Spark Shells

- Interactive shell available for interactive data analysis
  - ✓ Similar to other shells like R , Bash or Windows command prompt
- Allows to interact with data that is distributed on disk or in memory across many machines
- Provides both Python and Scala shells
  - ✓ Python – execute bin/pyspark
  - ✓ Scala – execute bin/spark-shell

Applications Places Terminal

csishydlab@apache-spark:~/spark-2.4.4-bin-hadoop2.7

File Edit View Search Terminal Help

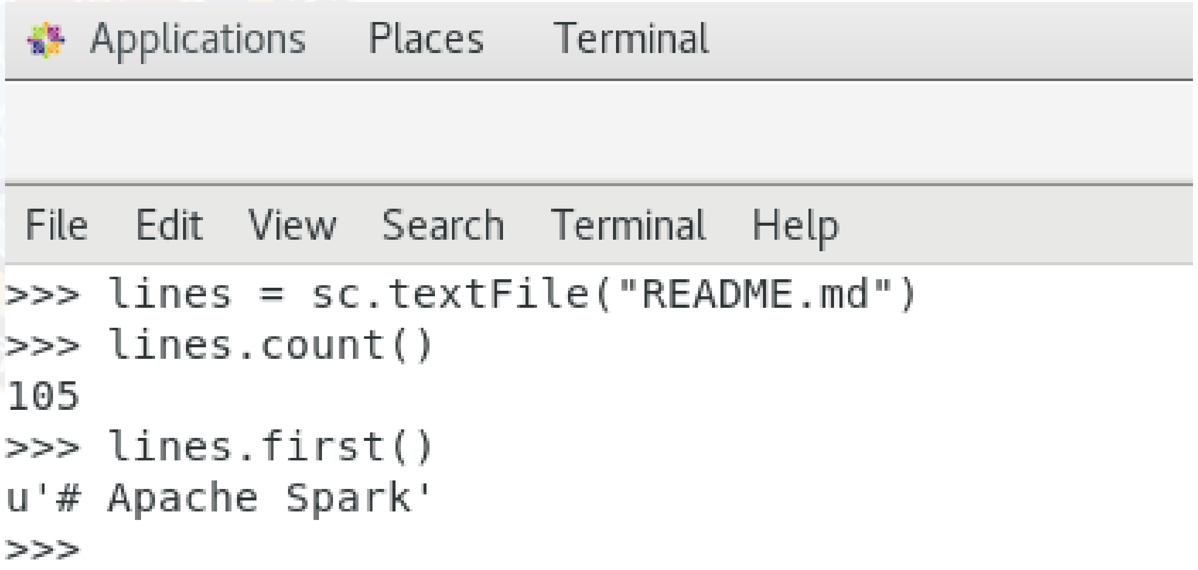
```
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$ bin/pyspark
Python 2.7.5 (default, Aug  7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
20/04/15 12:40:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.4
      /_/

Using Python version 2.7.5 (default, Aug  7 2019 00:51:29)
SparkSession available as 'spark'.
>>>
```

Getting Started (6)

Executing in shells

- Computations are expressed on collections distributed across the cluster
  - ✓ Termed as Resilient Distributed Datasets (RDD)
  - ✓ Sparks fundamental abstraction for distributed data and computation

- Example
  - ✓ Create RDD from local text file
  - ✓ Do some very simple analysis on it
  - ✓ Output the result
  - ✓ Exit the shell , Ctrl - D



Summary

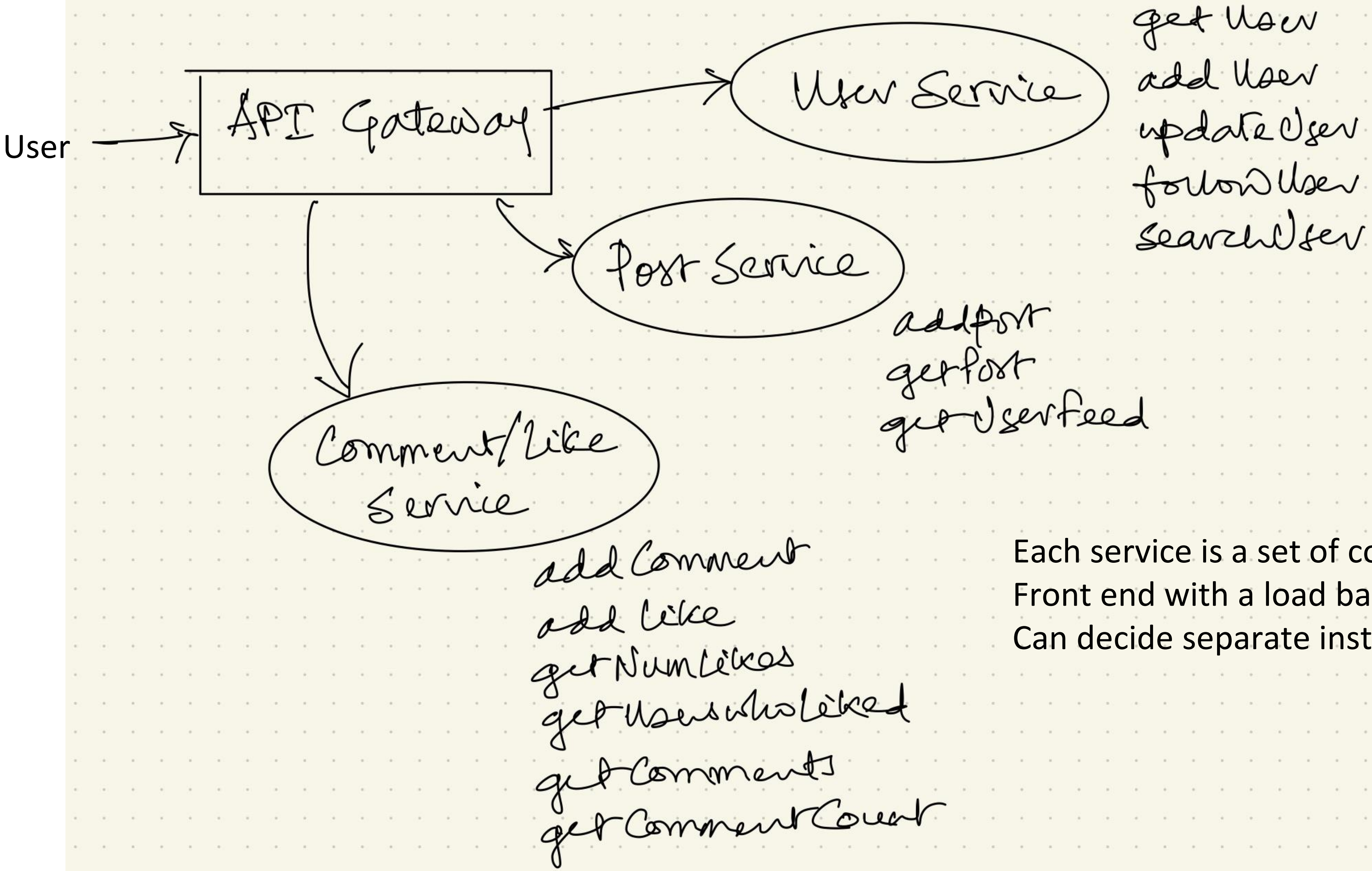
- Introduction to basic concepts
    - In-memory for speed
    - RDD basics, limitations, DSM comparison
    - General purpose big data computing covering multiple use cases
- Sample programs to get started
- Additional Reading
    - “Learning Spark” By Karau, Konwinski
    - Apache Spark documentation

Next Session:
Spark - Part 2

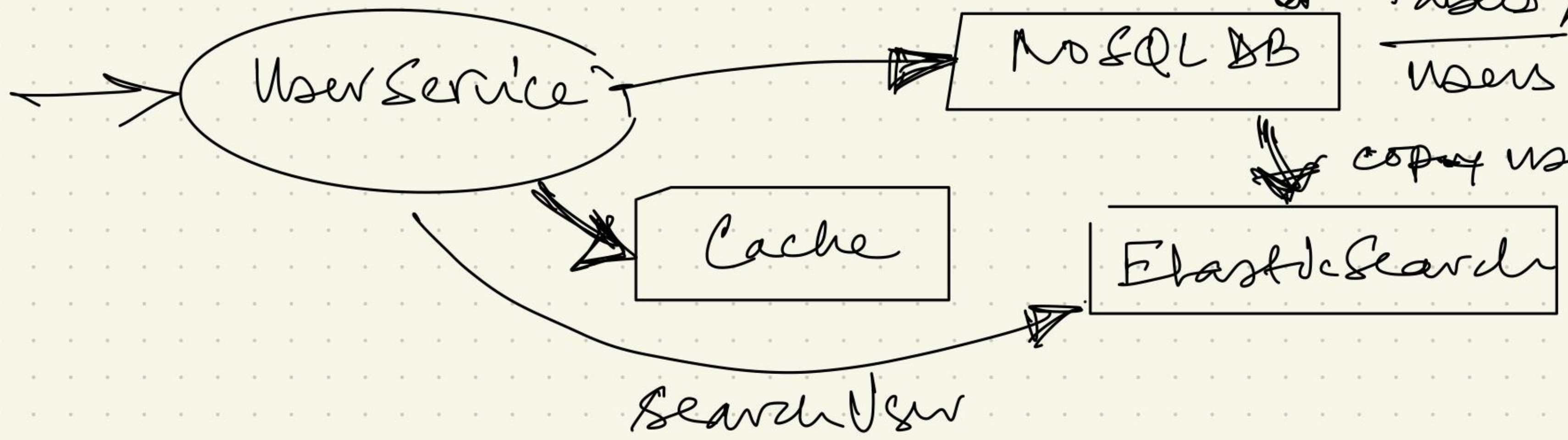
From Session 12:
System design problem - Instagram

System design exercise

Design an Instagram as a Cloud native Big Data Application.
How would you go about creating a design ?
How would you interact with users - Query? Notifications / Updates ?
What APIs would you have for users or internally ?
Identify the Big Data requirements - what kind of DBs/Storage would you need ?
May not be just one.
What is the data consistency desired ?
What compute instances would you have ?
Any load balancers, messaging systems ?
What about caching ?



Each service is a set of compute instances
Front end with a load balancer
Can decide separate instances for Read/Write to DB



* Tables / Collections

Eventual consistency

users, followers

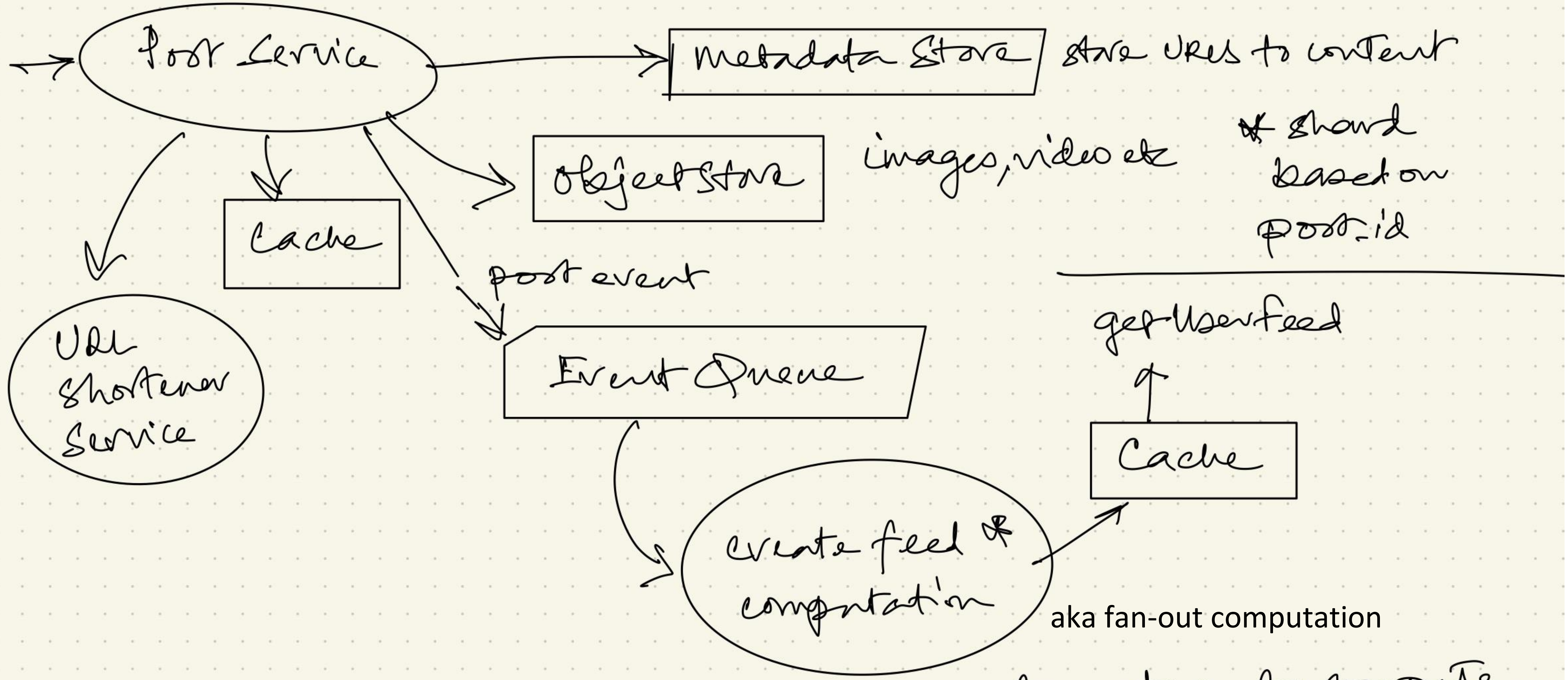
copy users for search

* use sharding by userid

Eventual consistency
Lazy updates

- update cache with comments
likes

getNumLikes : Lazy execution of query for exact count
Predict count for scale
(promise based cache design)



* for influencers posts do not auto compute feeds for others → will not scale