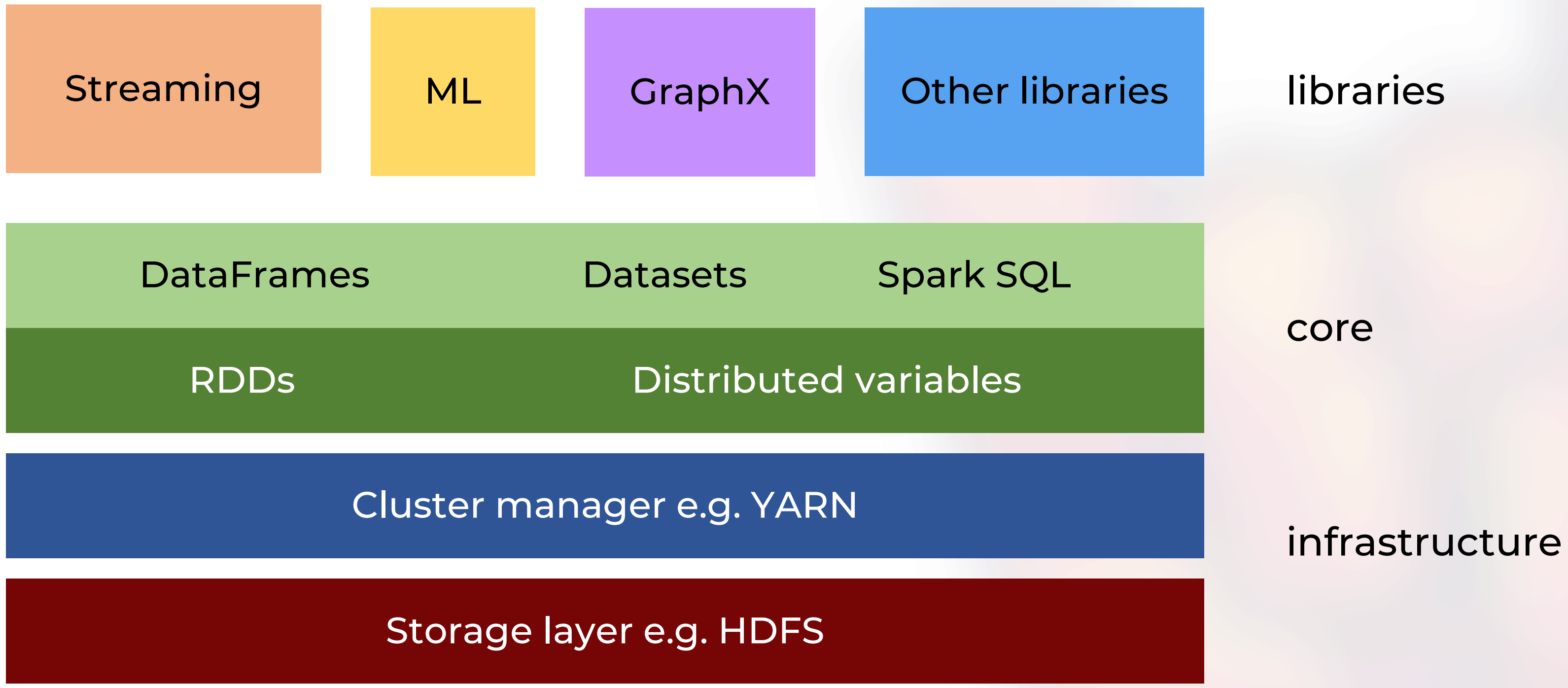
First Principles

Objective

Spark terminology & concepts

Optimization goals

Spark Layered Architecture



Spark Layered Architecture

Storage

- files, HDFS, S3, databases

Cluster manager

- standalone (bundled with Spark)
- YARN
- Mesos
- Kubernetes

Spark Core

- RDDs are the "first citizens" of Spark and have changed little since the beginning
- DataFrames & SQL: lots of optimizations out of the box

High-level libs

- Streaming: infinite data at sub-second latency
- ML: process machine learning models on big data at scale
- GraphX: process links between data
- all other libraries based on Spark Core (and RDDs in particular)

Spark Architecture

A cluster of worker nodes performs the work/data processing

Executor = worker logical node (JVM)

- performs work for a single application
- usually more than one per application
- launched in JVM containers with their own memory/CPU resources
- can be 0 or more deployed on the same physical machine

Driver = Spark application main JVM

- one per application
- starts the application and sends work to the executors

Cluster manages the deployment of the executors

- driver requests executors & resources from the cluster manager

For performance

- driver is close to the worker nodes (same physical rack or at least same LAN)
- worker nodes close to each other – otherwise shuffle* operations are expensive

RDDs

Distributed typed collections of JVM objects

The "first citizens" of Spark: all higher-level APIs reduce to RDDs

Pros: can be highly customized

- distribution can be controlled
- order of elements can be controlled
- arbitrary computation hard/impossible to express with SQL

Cons: hard to work with

- for complex operations, need to know the internals of Spark
- poor APIs for quick data processing

DataFrames

High-level distributed data structures

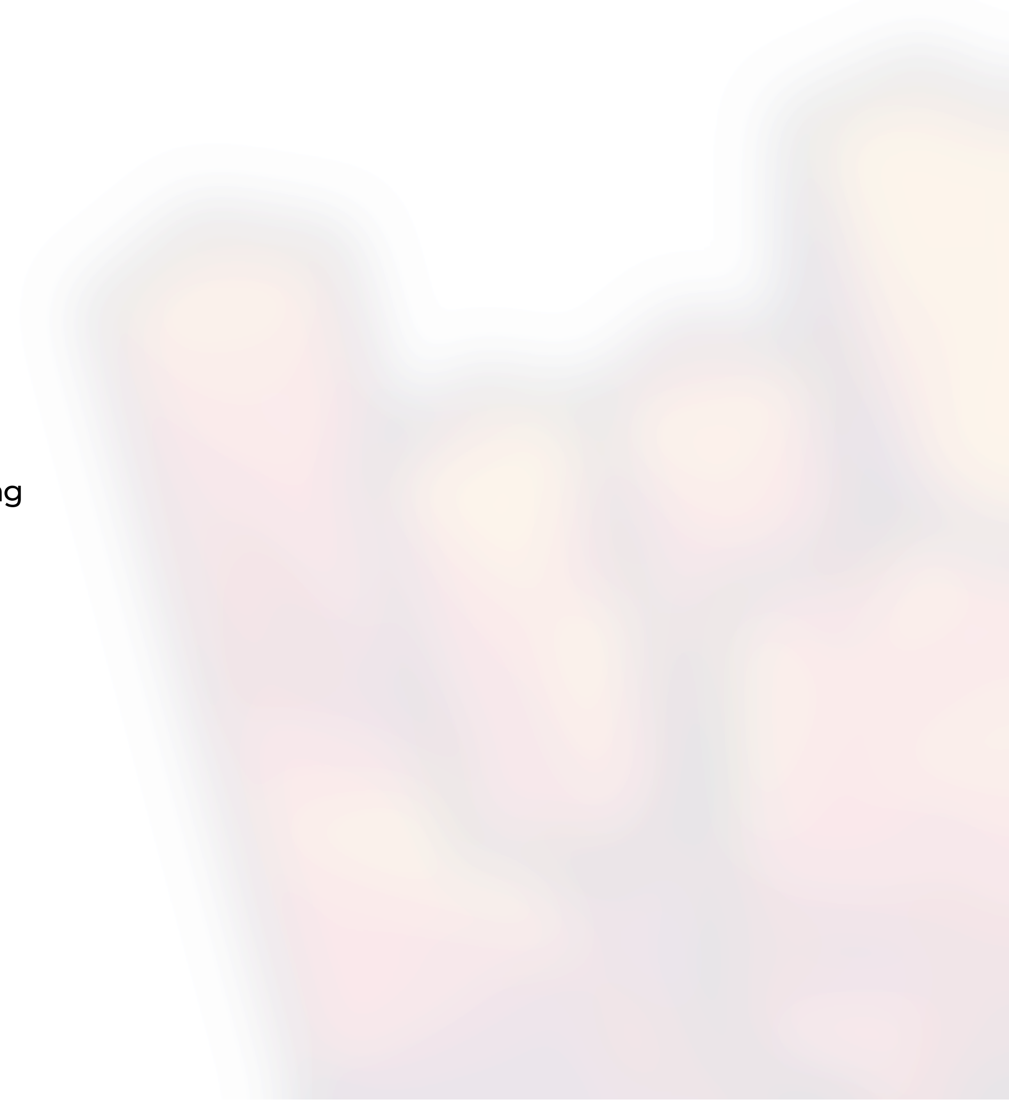
- contain Rows
- have a schema
- have additional API for querying
- support SQL directly on top of them
- generate RDDs after Spark SQL planning & optimizing

Pros

- easy to work with, support SQL
- already heavily optimized by Spark

Cons

- type-unsafe
- unable to compute *everything*
- hard to optimize further



Dataset[T]

Distributed typed collections of JVM objects

- support SQL functions of DataFrames
- support functional operators like RDDs

DataFrame = Dataset[Row]

Pros

- easy to work with, support for both SQL and functional programming
- some Spark optimizations out of the box
- type-safe

Cons

- memory and CPU expensive to create JVM objects
- unable to optimize lambdas

Performance Tips

Use DataFrames most of the time

- express almost anything with SQL
- Spark already optimizes most SQL functions

Use RDDs only in custom processing

Do not switch types

- DFs ⇄ RDD[YourType] or Dataset[YourType] is expensive
- In Python switching types is disastrous

Computing Anything

Lazy evaluation

- Spark waits until the last moment to execute the DF/RDD transformations

Planning

- Spark compiles DF/SQL transformations to RDD transformations (if necessary)
- Spark compiles RDD transformations into a graph before running any code
- logical plan = RDD dependency graph + narrow/wide transformations sequence
- physical plan = optimized sequence of steps for nodes in the cluster
- optimizations*

Transformations vs Actions

- transformations describe how new DFs are obtained
- actions start executing Spark code

Transformations vs Actions

- transformations return RDDs/DFs
- actions return something else e.g. Unit, a number etc.

Spark App Execution

An action triggers a job

A job is split into stages

- each stage is dependent on the stage before it
- a stage must fully complete before the next stage can start
- for performance: (usually) minimize the number of stages

A stage has tasks

- task = smallest unit of work
- tasks are run by executors

An RDD/DataFrame/Dataset has partitions

Concepts Relationships

App decomposition

- 1 job = 1 or more stages
- 1 stage = 1 or more tasks

Tasks & Executors

- 1 task is run by 1 executor
- each executor can run 0 or more tasks

Partitions & Tasks

- processing one partition = one task

Partitions & Executors

- 1 partition stays on 1 executor
- each executor can load 0 or more partitions in memory or on disk

Executors & Nodes

- 1 executor = 1 JVM on 1 physical node
- each physical node can have 0 or more executors

Optimization Goals

Optimize the TIME it takes for a job to run

- understanding how Spark works internally
- writing efficient code

We will not optimize*

- memory usage
- cluster resource (CPU, mem, bandwidth) usage
- compute time via configs

Write good code first

Impossible to squeeze perf out of bad code

Spark rocks