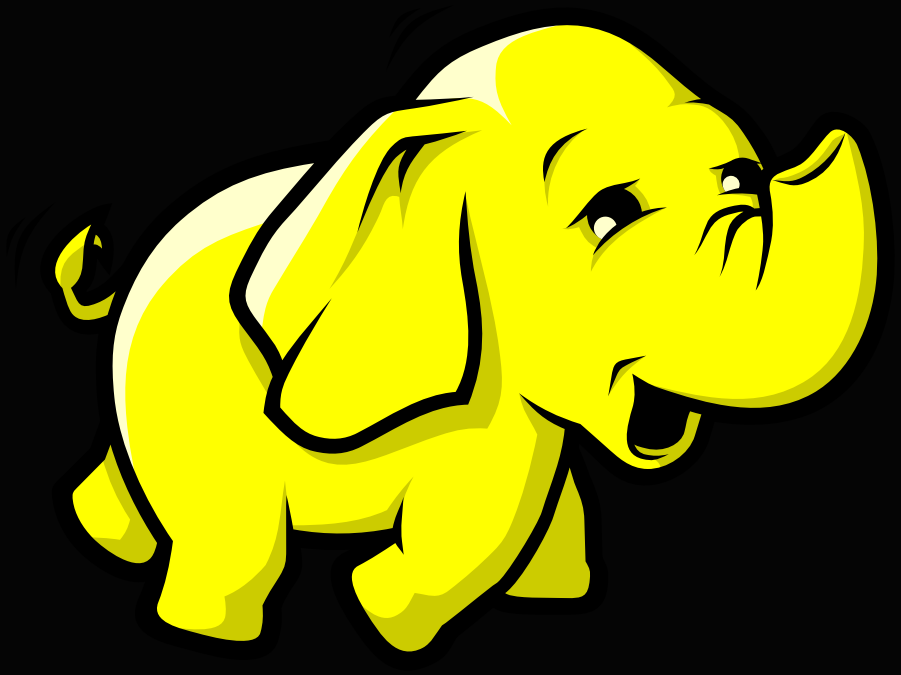
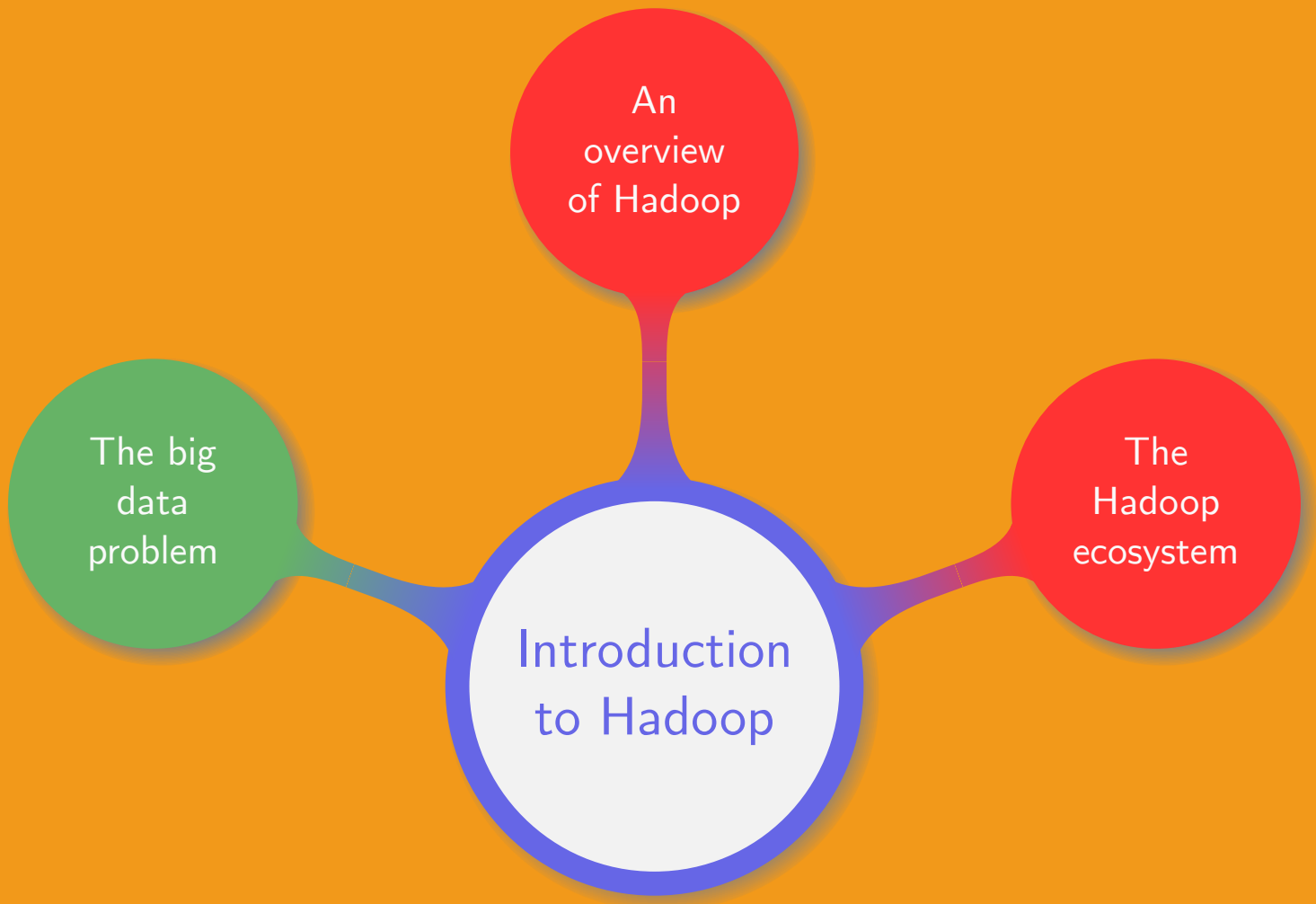


Methods and tools for big data

9. Introduction to Hadoop

Jing & Manuel – Summer 2020

Chapter organisation



Von Neumann bottleneck

Problem for a “simple computer”:

- Fast CPU
- Large memory
- Limited throughput

Von Neumann bottleneck

Problem for a “simple computer”:

- Fast CPU
- Large memory
- Limited throughput

Mitigating the problem:

- Use caching
- Apply branch prediction
- Parallel read from multiple locations

Amount of data

Total amount of data:

- Estimation in 2013: 4.4 billions TB
- Prediction for 2020: 44 billions TB

Amount of data

Total amount of data:

- Estimation in 2013: 4.4 billions TB
- Prediction for 2020: 44 billions TB

Sources generating data:

- Individuals (photos, videos, wechat, etc.)
- Machines (logs, GPS traces, internet of things, etc.)

Processing data

Relational Database Management Systems:

- Data size: gigabytes
- Access: interactive and batch
- Update: read/write small proportions of the data
- Structure: schema defined at writing time
- Efficiency: low-latency retrieval for small amount of data

Processing data

High-performance computing:

- Distributes computation across a cluster of machines
- Uses message passing interface
- Fits compute-bound jobs
- Data-flow controlled by programmer

Processing data

Volunteer computing:

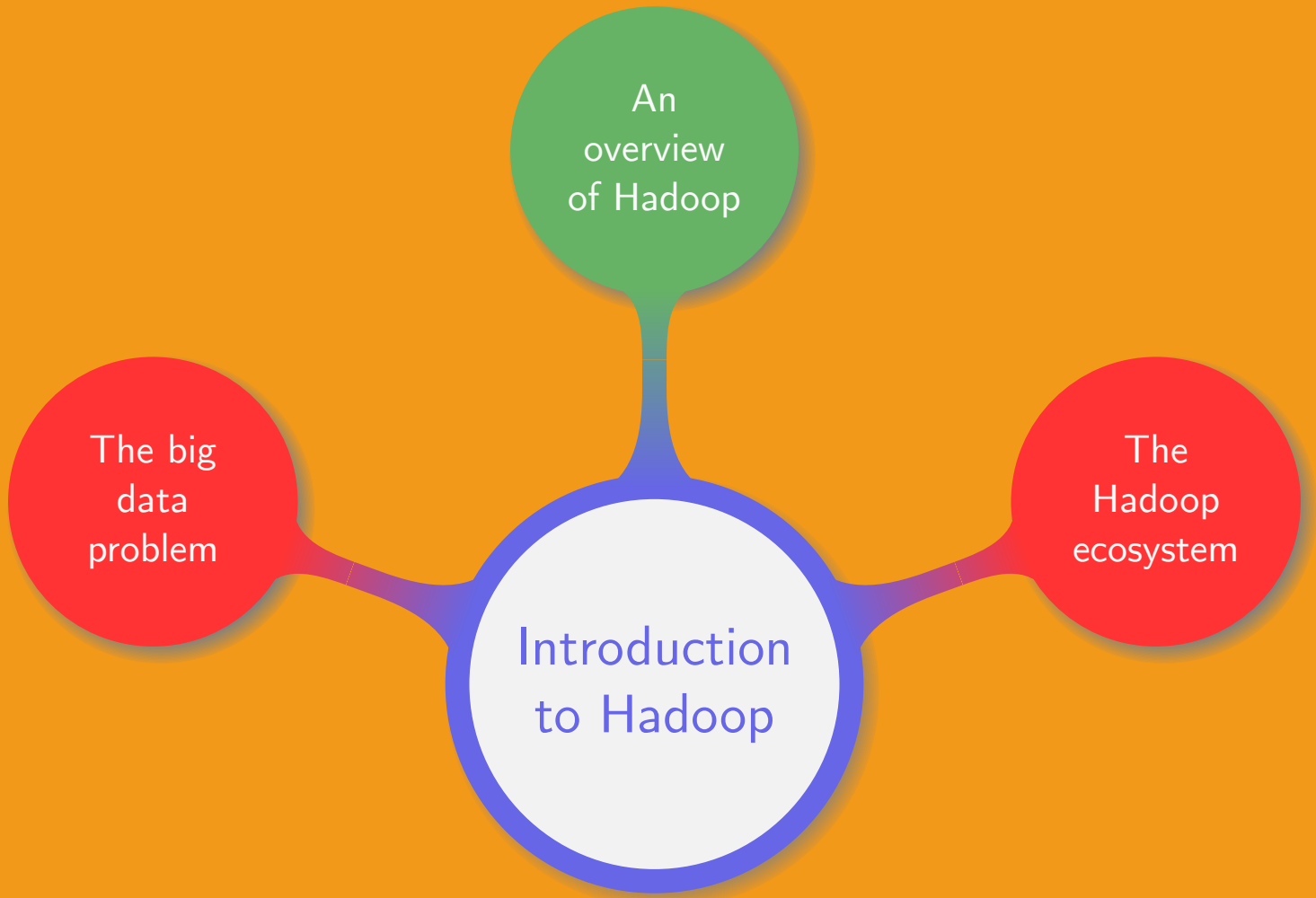
- Volunteers come from all around the world
- Split a task into “work units”
- Send each work units to a few people
- Compute-bound task
- Data transfer time small compared to processing time

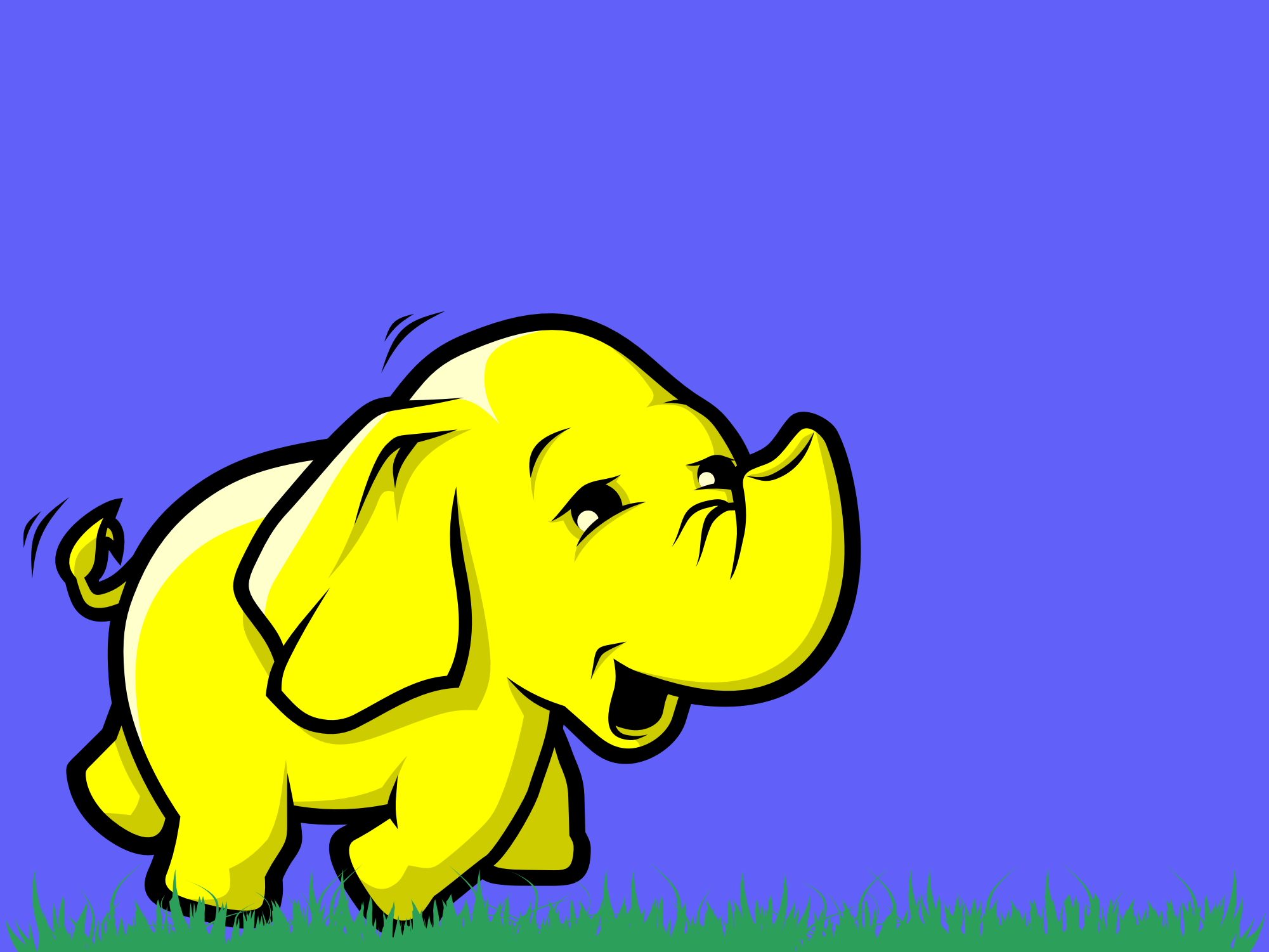
Limitations of those solutions

Previous solutions cannot be use to analyse large data sets:

- Database:
  - Hard drive seek time increases slower than data transfer rate
  - Data often unstructured
  - Slow to process as designed for read/write many times
- High-performance computing:
  - Handling of node or process failure
  - Require very high network bandwidth
- Volunteer computing: bandwidth limitation

Chapter organisation





A short history

The birth of Hadoop:

- 2002: Nutch, an open source web search engine
- 2003: paper describing Google File System (GFS)
- 2004:
  - NDFS: open source implementation of GFS for Nutch
  - Paper describing data processing on large clusters (MapReduce)
- 2005: open source implementation of MapReduce for Nutch
- 2006:
  - NDFS and MapReduce moved out of Nutch
  - Hadoop 0.1.0 released
  - Hadoop is run in production at Yahoo!

A short history

The adolescence of Hadoop:

- 2007 – 2008:
  - Number of companies using Hadoop jumps from 3 to over 20
  - Creation of Cloudera, first Hadoop distributor
- 2009:
  - MapR, new Hadoop distributor
  - HDFS and MapReduce become separate projects
- 2010 – 2011:
  - Many new “components” added to the Hadoop ecosystem
  - Receive two prizes at the Media Guardian Innovation Awards

A short history

The maturity of Hadoop:

- 2012:
  - Hadoop 1.0 released
  - YARN ready to replace MapReduce (Hadoop 2.0)
- 2013 – 2014:
  - More than half of the Fortune 50 use Hadoop
  - Spark and Drill added to the Hadoop ecosystem
- 2017: Hadoop 3.0 released

A short history

Hadoop's records:

- 2006: sort 1.8 TB of data in less than 48 h
- 2008: sort 1 TB of data in 209 s
- 2009: sort 1 TB of data in 62 s
- 2014: sort 100 TB of data in less than 23 min 30s

very fast

Goal of Hadoop

Context where to adopt Hadoop:

- Massive amount of data to analysed
- Data stored over hundreds or thousands of computers
- Computation must be completed even if some nodes fail
- Cluster composed of commodity or high-end hardware

Goal of Hadoop

Context where to adopt Hadoop:

- Massive amount of data to analysed
- Data stored over hundreds or thousands of computers
- Computation must be completed even if some nodes fail
- Cluster composed of commodity or high-end hardware

The goal is to efficiently analyse massive amount of data

write once, and read many times
no update needed for hadoop

The Hadoop framework

Hadoop is composed of core modules:

- Hadoop common: base libraries and utilities used by other modules
- Hadoop Distributed File System (HDFS): distributed file system
- Hadoop MapReduce: implementation of the MapReduce model
- Apache Yet Another Resource Negotiator (YARN): manages the cluster resources and schedules the user's tasks

The Hadoop framework

Hadoop is composed of core modules:

- Hadoop common: base libraries and utilities used by other modules
- Hadoop Distributed File System (HDFS): distributed file system
- Hadoop MapReduce: implementation of the MapReduce model
- Apache Yet Another Resource Negotiator (YARN): manages the cluster resources and schedules the user's tasks

Languages:

- Mainly Java
- Some C
- Shell scripts for command line utilities

Java可以多平台运作。Hadoop需要在很多不同的电脑上跑

HDFS overview

Characteristics of HDFS:

- Large files: at least hundreds of megabytes to terabytes
- Streaming data access: write once, read many times
- Commodity hardware: inexpensive common hardware

CD-ROM: 700 MB
DVD: 4.7GB
BR: 25 GB

A block of filesystem: about 4 KB

HDFS overview

Characteristics of HDFS:

- Large files: at least hundreds of megabytes to terabytes
- Streaming data access: write once, read many times
- Commodity hardware: inexpensive common hardware

Limitations of HDFS:

- High throughput at the expense of latency
- The “Master node” keeps the filesystem metadata in memory
- Write always in append mode, by a single writer

MapReduce overview

Programming paradigm composed of three main steps:

- Map:
  - A master node distributes the work and ensures exactly one copy of the redundant data is processed
  - Each worker node considers its local data and transforms it into key-value pairs
- Shuffle: each worker node redistributes its pairs based on the keys
- Reduce: each worker node combines a set of pairs into a smaller one

MapReduce overview

MapReduce requirements:

- Mapping operations must be independent of each others
- Parallelism is limited by the number of sources and nearby CPUs
- Either all the output sharing the same key must be processed by a single reducer or the reduction must be associative

$$(a*b)*c = a*(b*c)$$

MapReduce overview

MapReduce requirements:

- Mapping operations must be independent of each others
- Parallelism is limited by the number of sources and nearby CPUs
- Either all the output sharing the same key must be processed by a single reducer or the reduction must be associative

MapReduce benefits:

- Highly scalable on commodity hardware
- Possible to recover for partial failure
- Great efficiency due to parallelism

YARN overview

A *container* is an environment with restricted resources where application-specific processes are run

YARN overview

A *container* is an environment with restricted resources where application-specific processes are run

YARN provides two types of daemons:

- Resource manager:
  - One per cluster
  - Manages the resources for the whole cluster
- Node manager:
  - One per cluster node
  - Launches and monitors containers

MapReduce vs. YARN

In Hadoop 1, MapReduce:

- Directly interacts with the filesystem
- Manages resources

MapReduce vs. YARN

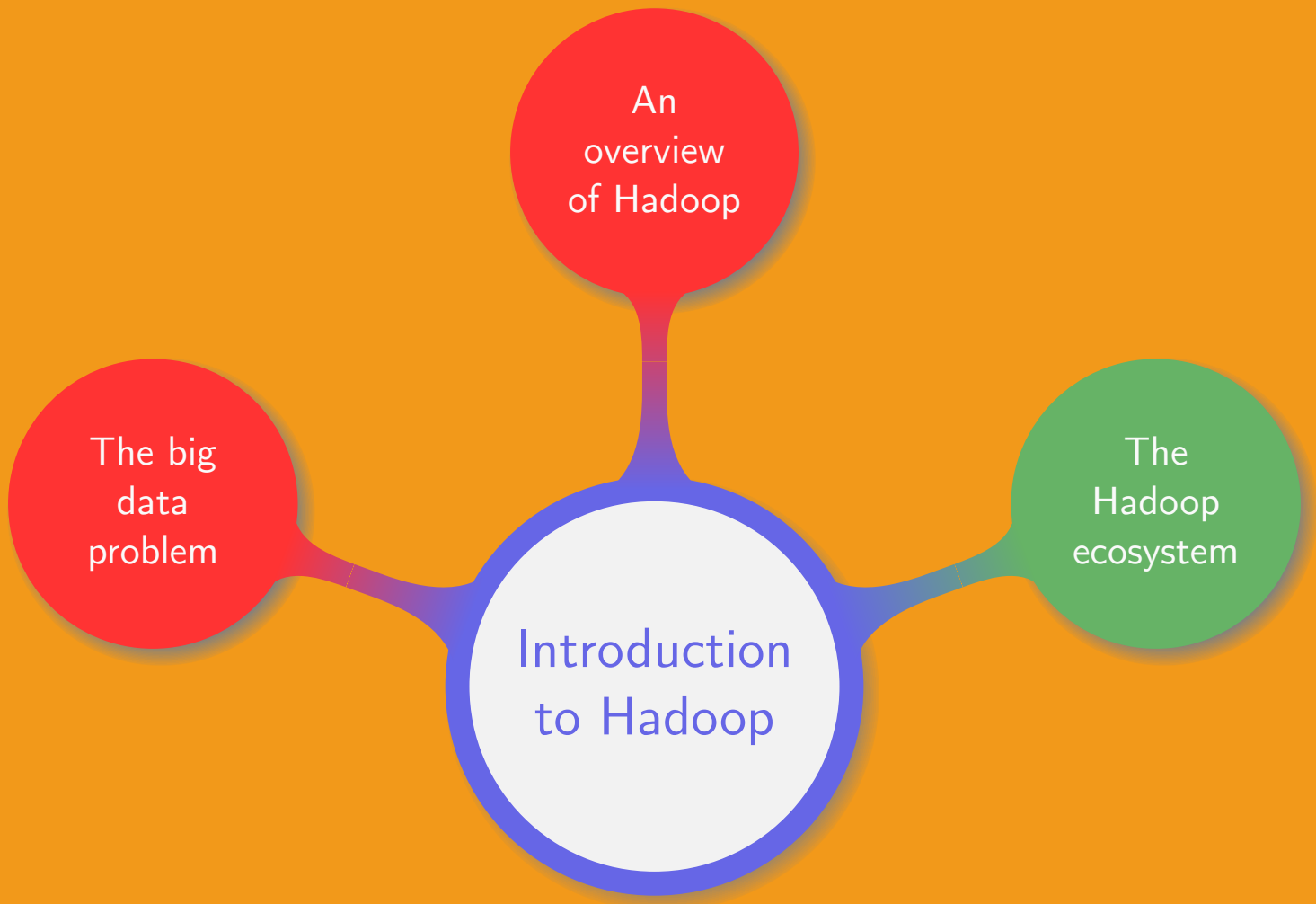
In Hadoop 1, MapReduce:

- Directly interacts with the filesystem
- Manages resources

In Hadoop 2, YARN:

- Manages the resources
- Interacts with the filesystem
- Hides low level details from the user
- Offers an intermediate layer supporting many other distributed programming paradigms

Chapter organisation



Mesos

Goal: global scalable resource manager, not restricted to Hadoop

Mesos scheduling:

- Determine the available resources
- Offer "various options" to an application scheduler
- Allow any number of scheduling algorithm to be developed, plugged, and used simultaneously
- Each framework decides what scheduling algorithm to use
- Mesos allocates resources across the schedulers, resolves conflicts, and ensures a fair share of the resources

Myriad

Goal: use Mesos to manage YARN resource requests

Simplified strategy:

1. A job requests resources to YARN
2. YARN uses the Myriad scheduler to allocate resources
3. Myriad scheduler matches requests to Mesos' resources offers
4. YARN allocates the resources

Myriad

Goal: use Mesos to manage YARN resource requests

Simplified strategy:

1. A job requests resources to YARN
2. YARN uses the Myriad scheduler to allocate resources
3. Myriad scheduler matches requests to Mesos' resources offers
4. YARN allocates the resources

Benefits:

- Get the best from both worlds
- Give more flexibility to YARN

Spark

Goals:

- Be a full replacement for MapReduce
- Efficiently support multi-pass applications
- Write and read from the disk as little as possible
- As much as possible take advantage of the memory

Main ideas:

- Resilient Distributed Dataset (RDD): contains the data to be transformed or analysed
- Transformation: modifies an RDD into a new one
- Action: analyses an RDD

Drill

Goals:

- Integrate into Hadoop as a MapReduce replacement
- Be an interactive ad-hoc analysis system for read-only data
- Be easily expandable using storage plugins
- Enjoy data agility

Main ideas:

- Columnar execution: shredded, in-memory columnar data representation
- Runtime compilation and code generation: compile and re-compile queries at runtime

make sure it is safe

- Optimistic execution: stream data in memory and minimises the disk usage

如果全放在mem，万一有node failure就没了

Spark vs. Drill

When to use Spark or Drill:

- Drill is an ANSI SQL:2003
- Spark has SQL query capabilities
- Drill allows fine grained security at the file level

drill 只搞 database

- Drill is best used as a distributed SQL query engine
- Spark is best used to perform complex math, statistics, or machine learning

Flink

Basics on Flink:

- Allows the execution of dataflow programs following a data-parallel and pipelined approach
- Provides a high-throughput and low-latency streaming engine
- Nicely handles node failures
- Can connect to various storage types

Tez

Basics on Tez:

- Targets batch and interactive data processing applications
- Intends to improve MapReduce paradigm 没有spark好
- Exposes more simple framework and API to write YARN applications
- Expresses computation as a dataflow graph

HBase

Basics on HBase: based on mapreduce

- NoSQL database system for distributed filesystems
- Low latency access to small amount of data in a large data set
- Fast scan across tables
- Random access to rows

Common use cases:

- Applications requiring sparse rows
- Not good for relational analytics and transactional needs

Hive, Spark SQL, and Presto

Basics on Hive:

- Allows SQL data access in HDFS using an SQL-like query language HQL
- Converts queries to MapReduce, Tez, or Spark jobs
- Warning: does not fully comply to ANSI-standard SQL

Hive, Spark SQL, and Presto

Basics on Hive:

- Allows SQL data access in HDFS using an SQL-like query language HQL
- Converts queries to MapReduce, Tez, or Spark jobs
- Warning: does not fully comply to ANSI-standard SQL

Basics on Spark SQL (formerly Shark):

- Was initially a port of Hive to Spark
- Follows Spark in-memory computing model
- Is “mostly” compatible with HQL

Hive, Spark SQL, and Presto

Basics on Hive:

- Allows SQL data access in HDFS using an SQL-like query language HQL
- Converts queries to MapReduce, Tez, or Spark jobs
- Warning: does not fully comply to ANSI-standard SQL

Basics on Spark SQL (formerly Shark):

- Was initially a port of Hive to Spark
- Follows Spark in-memory computing model
- Is "mostly" compatible with HQL

Basics on Presto:

- Supports ANSI-SQL standard
- Uses a custom engine, not based on MapReduce
- Can access various data sources through storage plugins

Serialization and storage

Avro:

- Input: a schema describing the data and the data
- Output: generates the code to read/write data

Serialization and storage

Avro:

- Input: a schema describing the data and the data
- Output: generates the code to read/write data

Parquet:

- Columnar storage format
- Complex to handle

Serialization and storage

Avro:

- Input: a schema describing the data and the data
- Output: generates the code to read/write data

Parquet:

- Columnar storage format
- Complex to handle

serial data storage?

Java Script Object Notation (JSON):

- Not part of Hadoop
- Often preferred to XML by Hadoop community
- Represent data using key-value pairs

Management and monitoring

Ambari:

- Production-ready, easy to use web-based GUI for Hadoop
- Eases the installation and monitoring of a cluster

Management and monitoring

Ambari:

- Production-ready, easy to use web-based GUI for Hadoop
- Eases the installation and monitoring of a cluster

Zookeeper:

- Effective mechanism to store and share small amounts of states and configuration across the cluster
- Not a replacement for any key-value store
- Has built-in protections to prevent using it as large data-store
- Used as a coordination service

similar to dbus
(enable interation between
different programs)

Analytics helpers

Major analytics helpers:

- Pig: high-level language to speak to MapReduce
- Hadoop streaming: write mappers/reducers in any language
- Mahout: collection of scalable machine-learning algorithms for Hadoop
- MLLib: similar to Mahout, based on Spark
- Hadoop Image Processing Interface: package allowing to examine images and determine their differences and similarities

Data transfer

Moving data to and from Hadoop:

- Sqoop: transfer data between HDFS and relational databases
- Flume: distributed system for collecting, aggregating, and moving large amount of data from various sources into HDFS
- Distributed Copy (DistCP):
  - Part of basic Hadoop tools
  - Used to move data between the clusters
  - Is the basis for more advanced Hadoop recovery tools

Data transfer

Moving data to and from Hadoop:

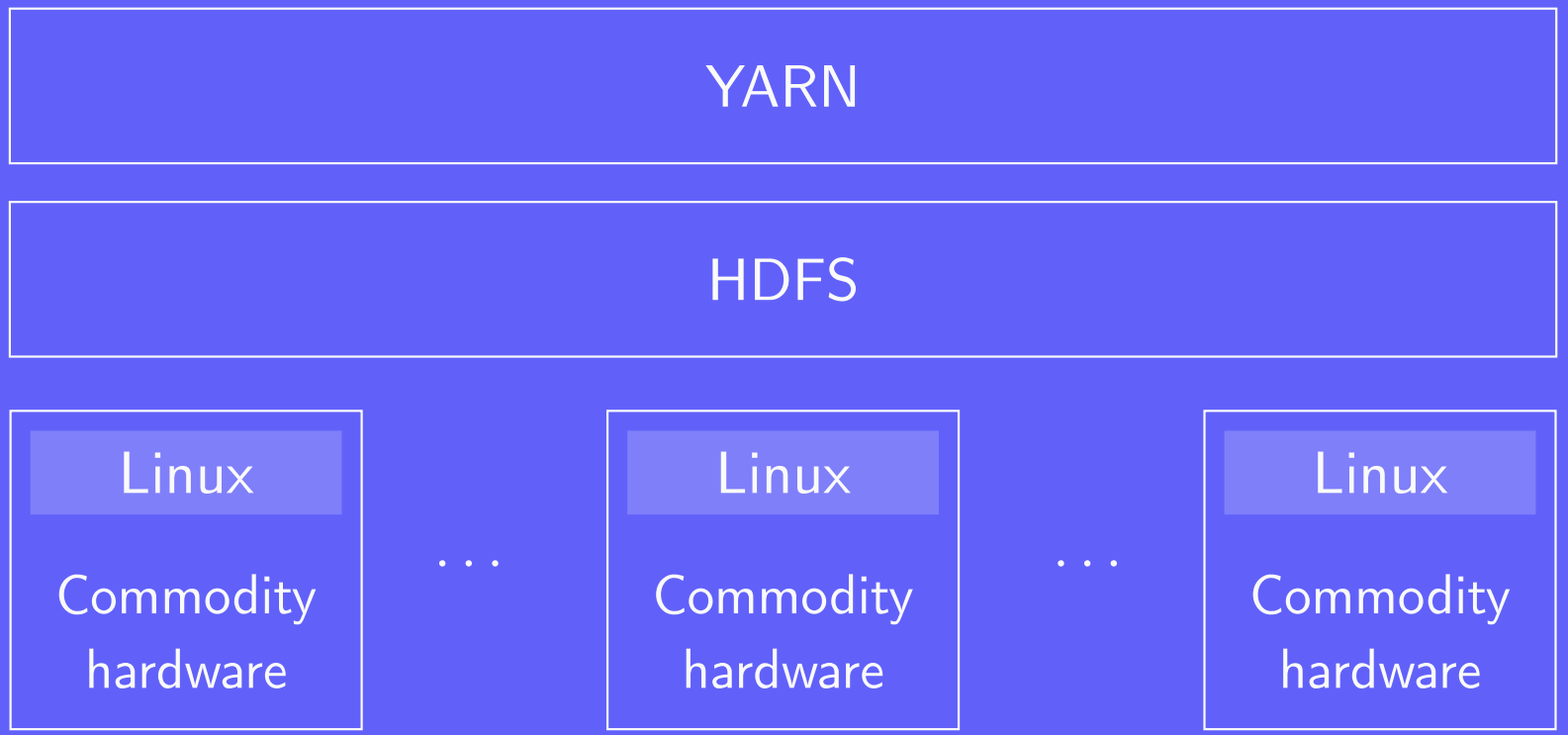
- Sqoop: transfer data between HDFS and relational databases
- Flume: distributed system for collecting, aggregating, and moving large amount of data from various sources into HDFS
- Distributed Copy (DistCP):
  - Part of basic Hadoop tools
  - Used to move data between the clusters
  - Is the basis for more advanced Hadoop recovery tools

Refer to Hadoop ecosystem table for more details

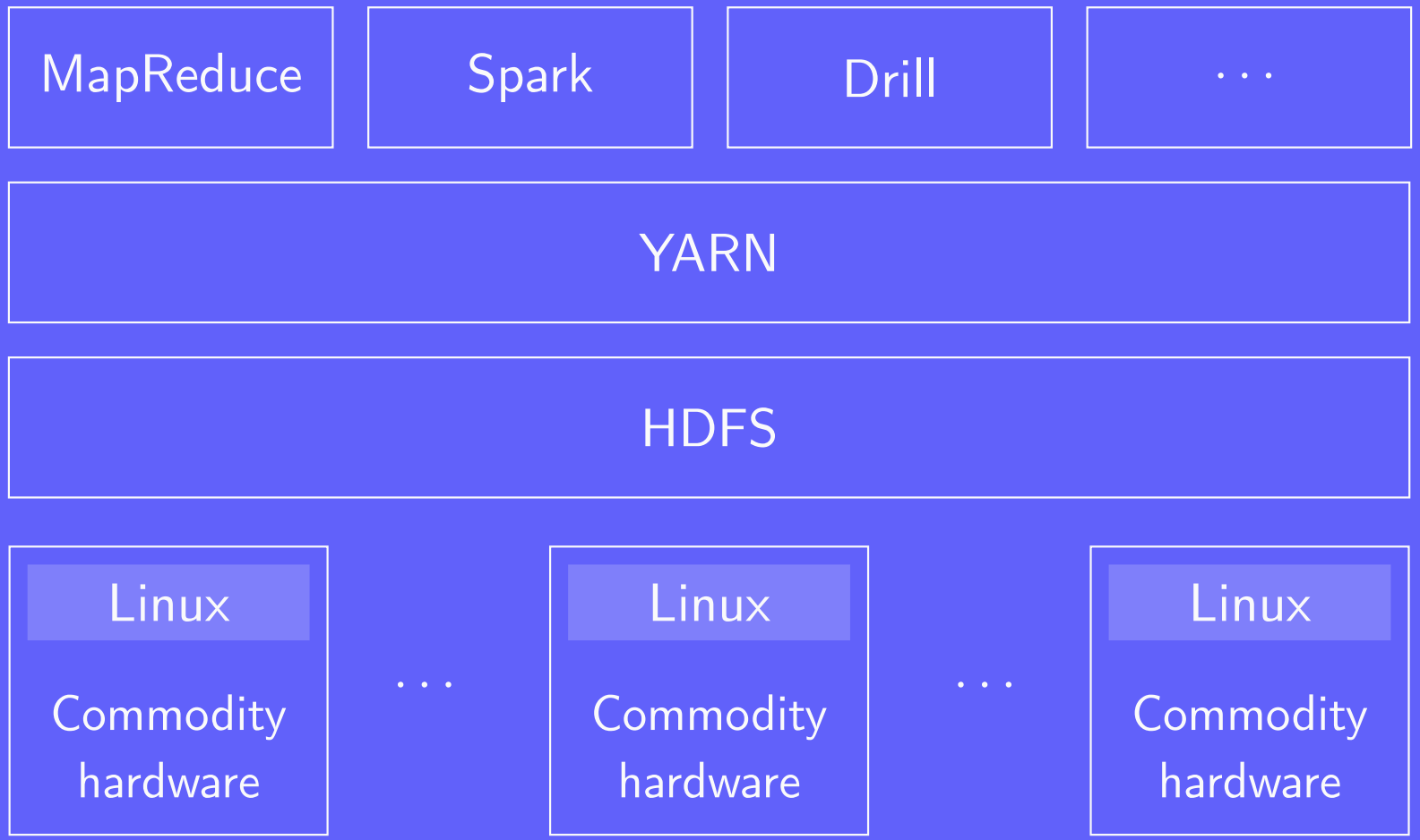
Hadoop ecosystem in VE{4|5}72

Hadoop ecosystem in VE{4|5}72

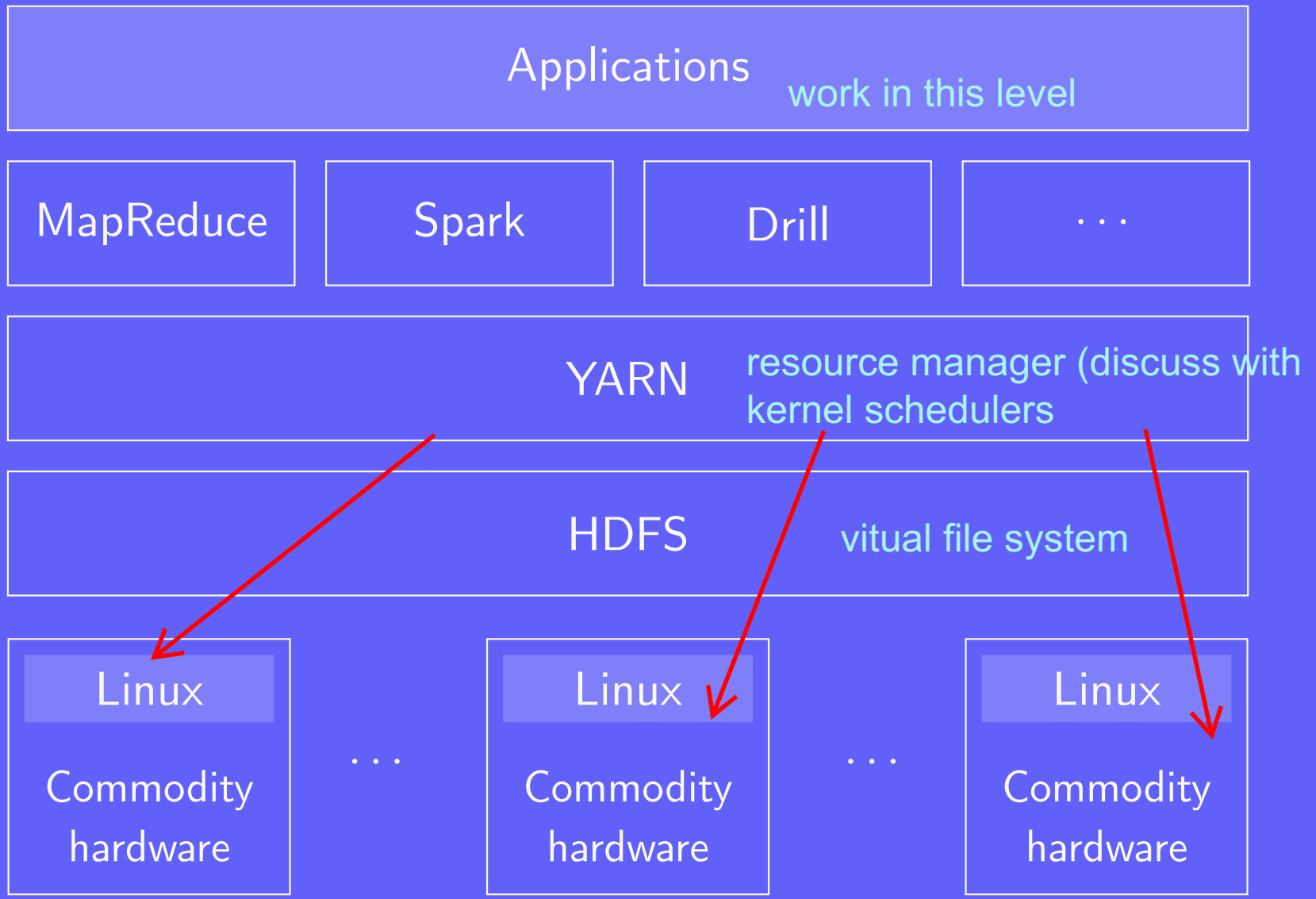
Hadoop ecosystem in VE{4|5}72



Hadoop ecosystem in VE{4|5}72

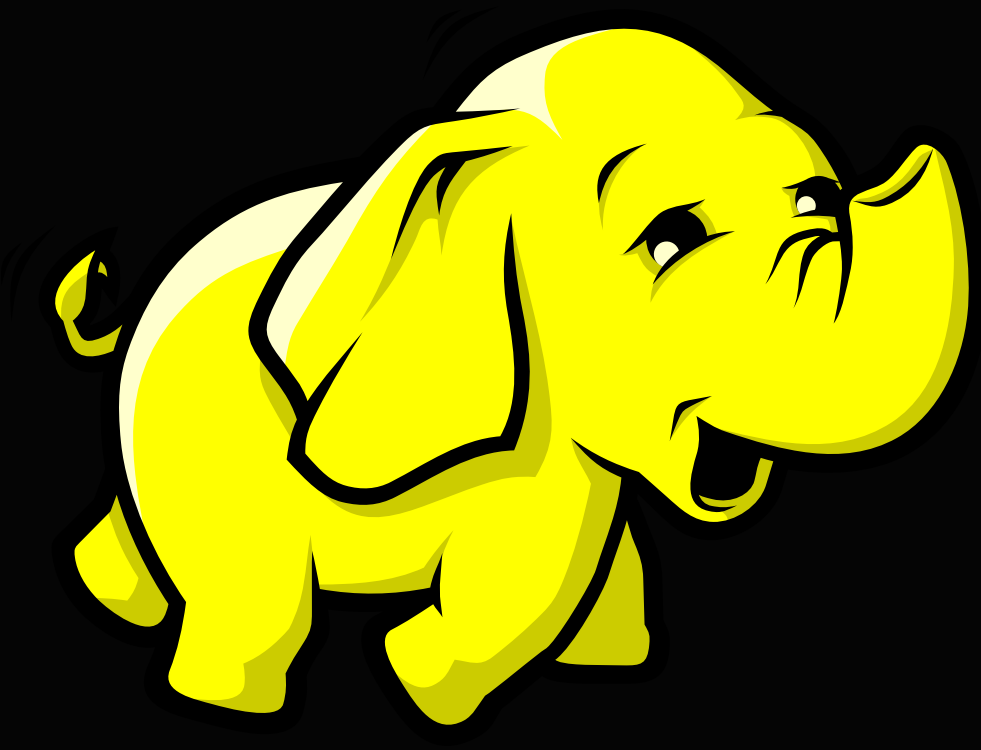


Hadoop ecosystem in VE{4|5}72



Key points

- Cite some limitation of common solutions when analysing large data sets?
- What is Hadoop?
- What are Hadoop's goals?
- Quickly describe the Hadoop ecosystem



Thank you!

References I

9.10 https://upload.wikimedia.org/wikipedia/commons/0/0e/Hadoop_logo.svg