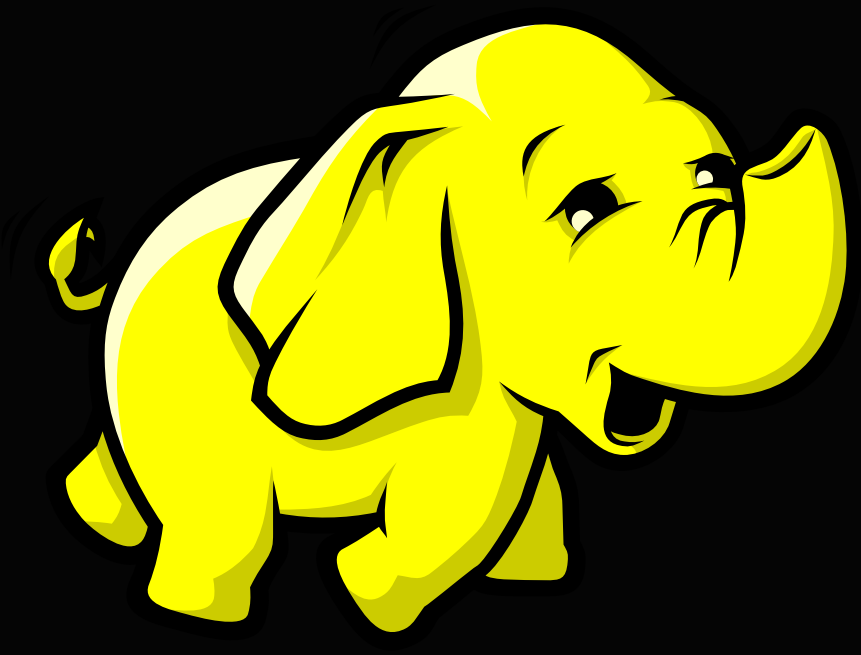
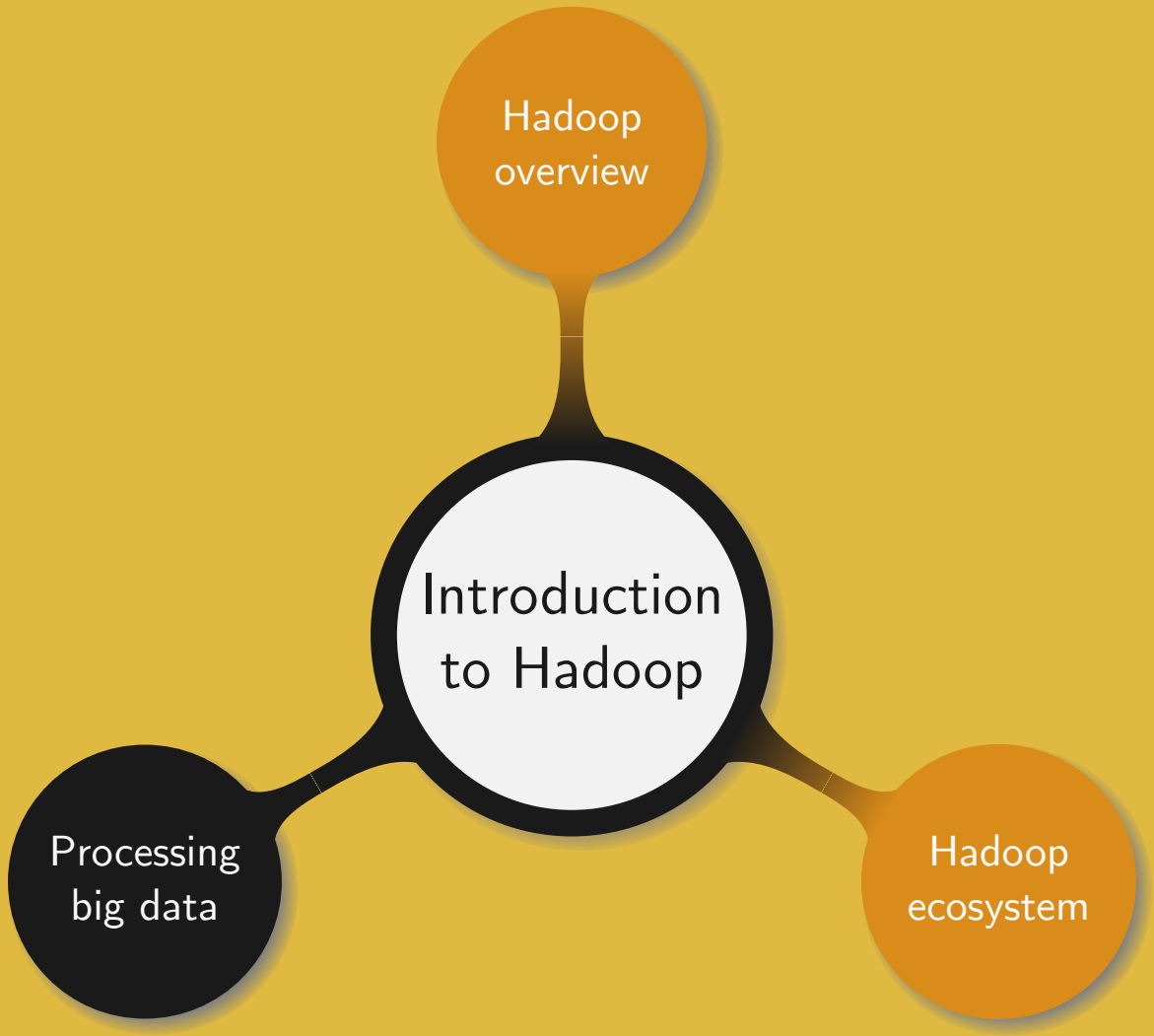


Methods and tools for big data

1. Introduction to Hadoop

Manuel – Summer 2021

Chapter organisation



Data analysis

Generated data is often:

- Stored, e.g. in databases
- Preprocessed, e.g. cleaned
- Analysed, e.g. machine learning

Most common advanced analytics:

- Supervised learning: predict a label based on some features
- Recommendation: suggest product based on users' behaviour
- Unsupervised learning: discover structure in the data
- Graph analytics: searching for patterns

Von Neumann bottleneck

Problem for a regular computer:

- Fast CPU
- Large memory
- Limited throughput

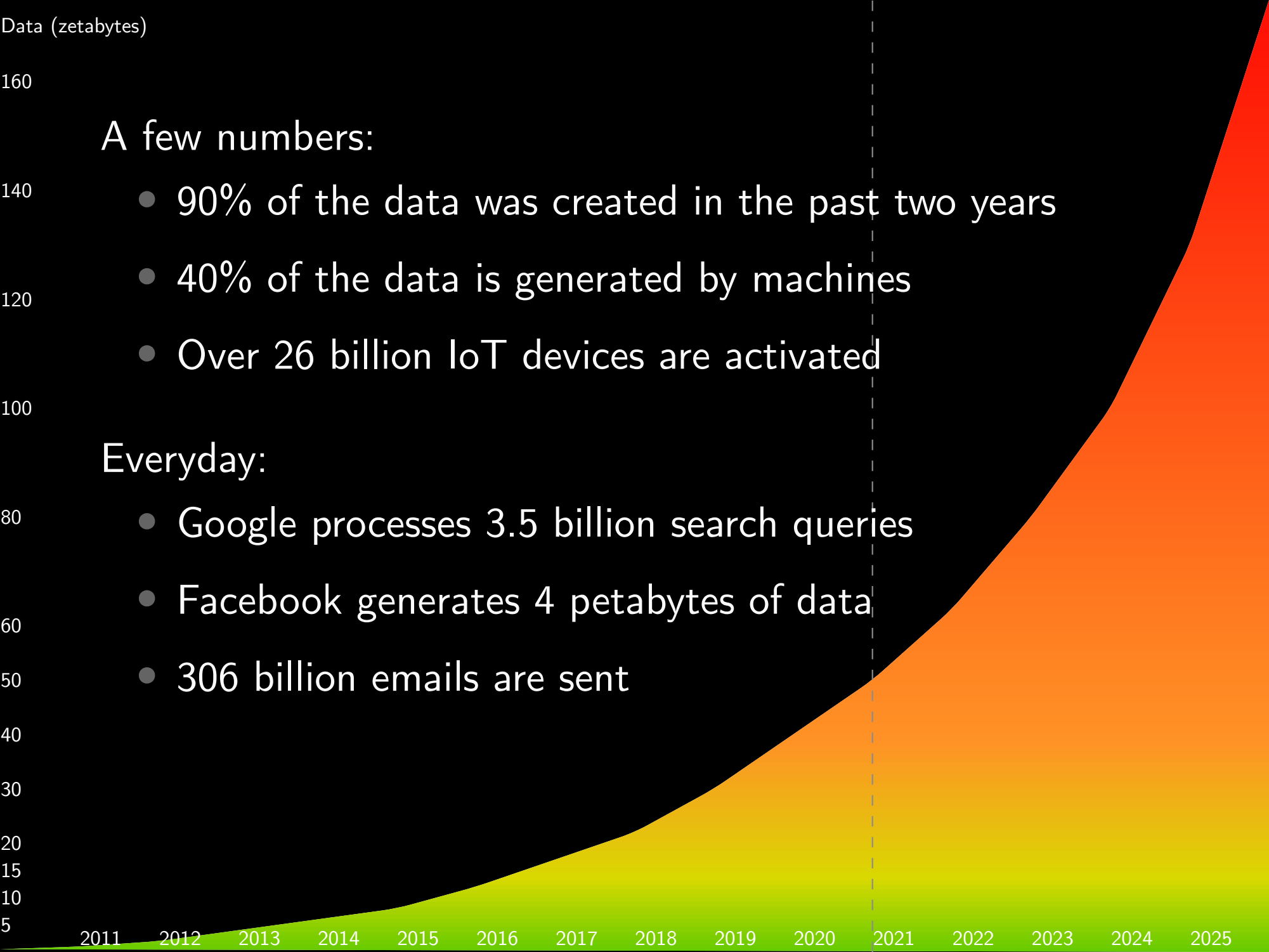
Mitigating the problem:

- Use caching
- Apply branch prediction
- Parallel read using RAID

Example. The speed of a disc read decreased relatively over time:

- 1990: 1.5 GB HDD at 4.4 MB/s
- Today: 1 TB HDD at 100 MB/s

Scanning a whole disc in 1990 took 5 min, today it takes over 2.5 h!



A few numbers:

- 90% of the data was created in the past two years
- 40% of the data is generated by machines
- Over 26 billion IoT devices are activated

Everyday:

- Google processes 3.5 billion search queries
- Facebook generates 4 petabytes of data
- 306 billion emails are sent

How to store and process data

as it grows very big?

Storing 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

Limitations of databases:

- Hard drive seek time increases slower than data transfer rate
- Data is often unstructured
- Slow to process as designed for read|write many times

Processing data

High-performance computing (HPC):

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

Limitations of HPC:

- Handling of node or process failure
- Require very high network bandwidth
- Expensive infrastructures, complex to extend
- Low level APIs

Sequential vs. parallel algorithms

Random Access Machine (RAM) model:

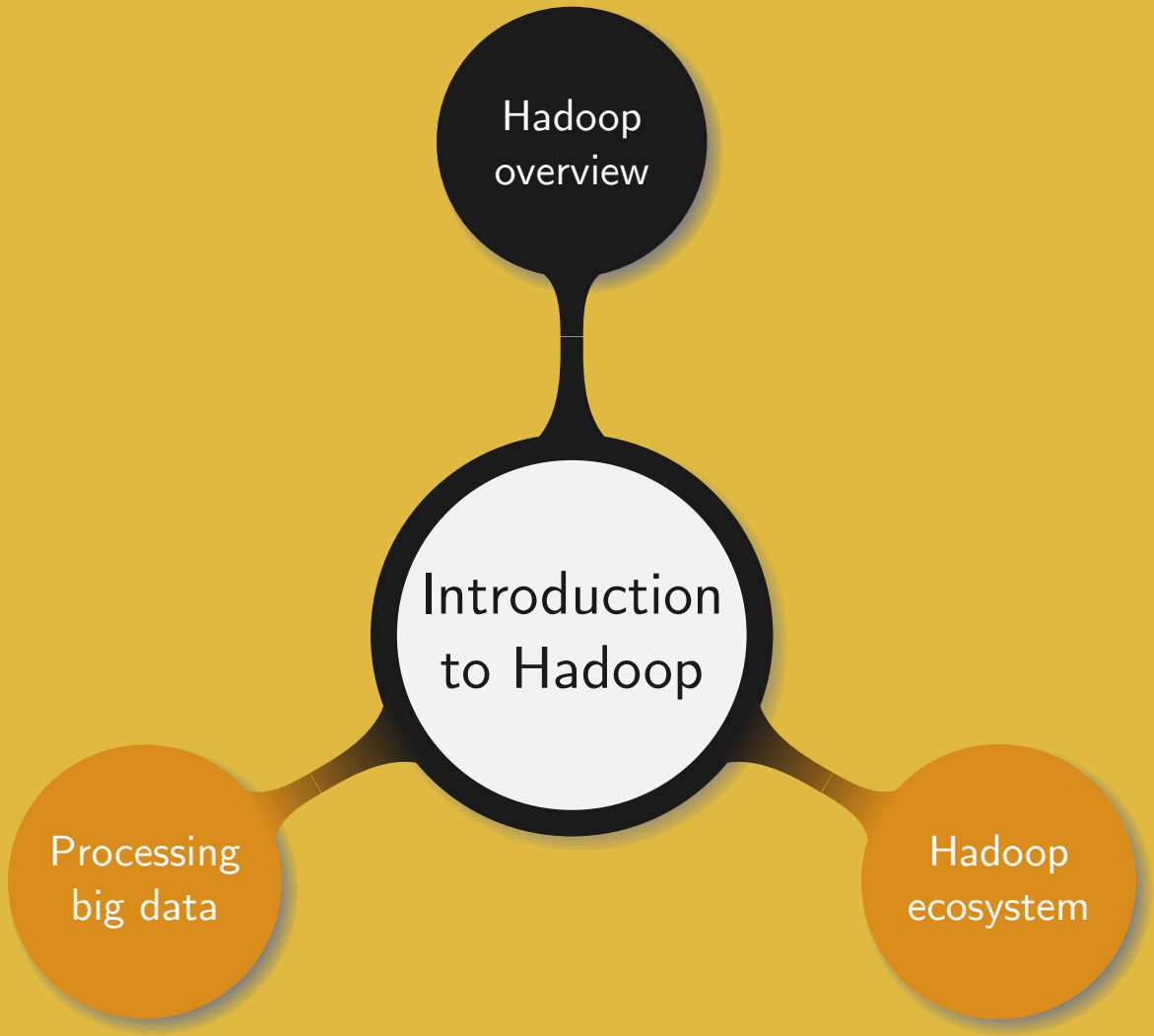
- A processor with a memory attached to it
- Each operation has a constant cost
- Runtime is proportional to the number of operations

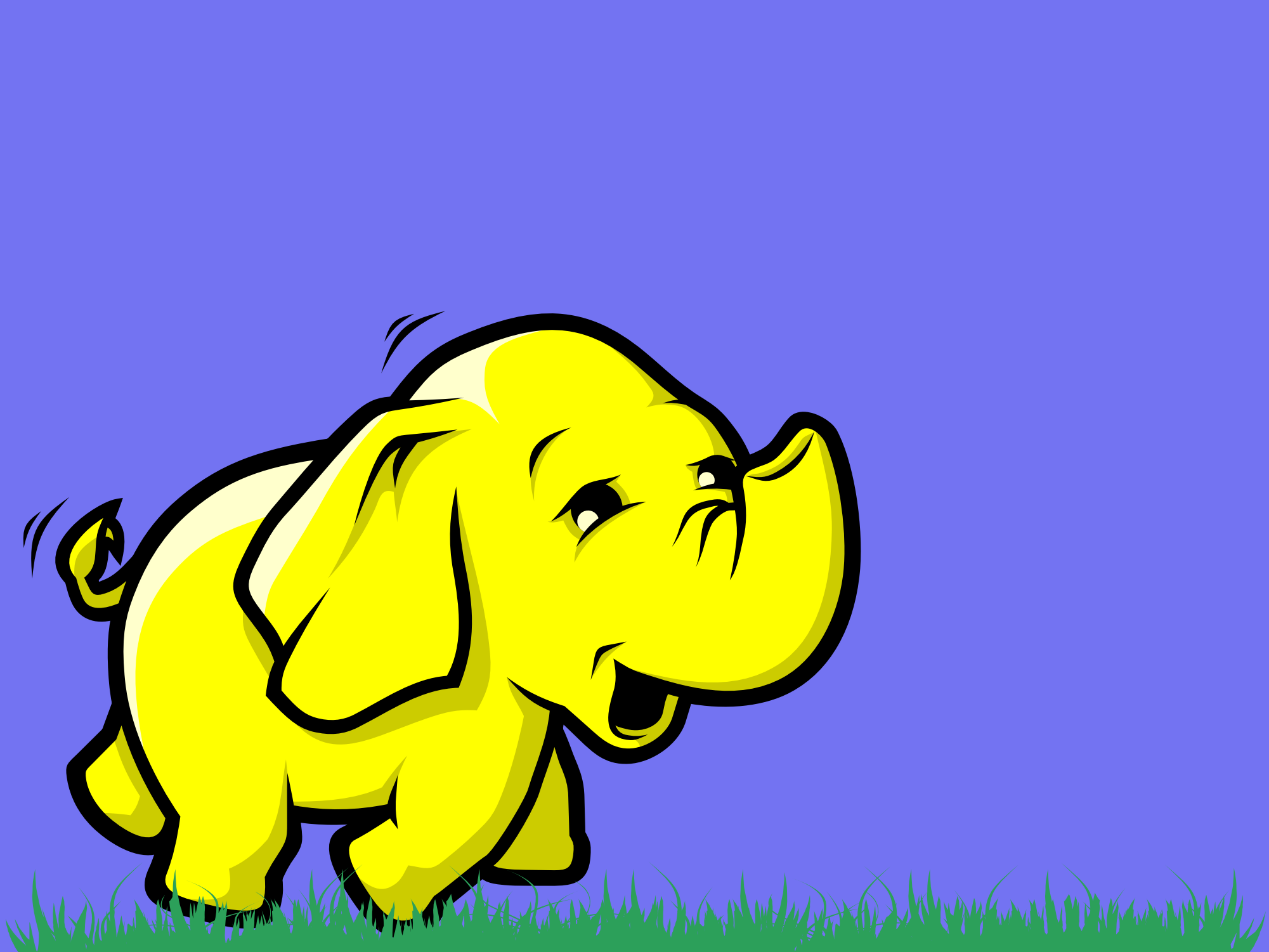
Parallel Random Access Machine (PRAM) model:

- Several processors with one or more memory modules attached
- Need to specify how how to deal with concurrent writes
- Each operation has a constant cost
- Runtime is defined when the slowest processor completes

When dealing with "real" big data we need a distributed system

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

Hadoop's goals

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

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

The end goal is to efficiently analyse massive amount of data

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

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

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 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

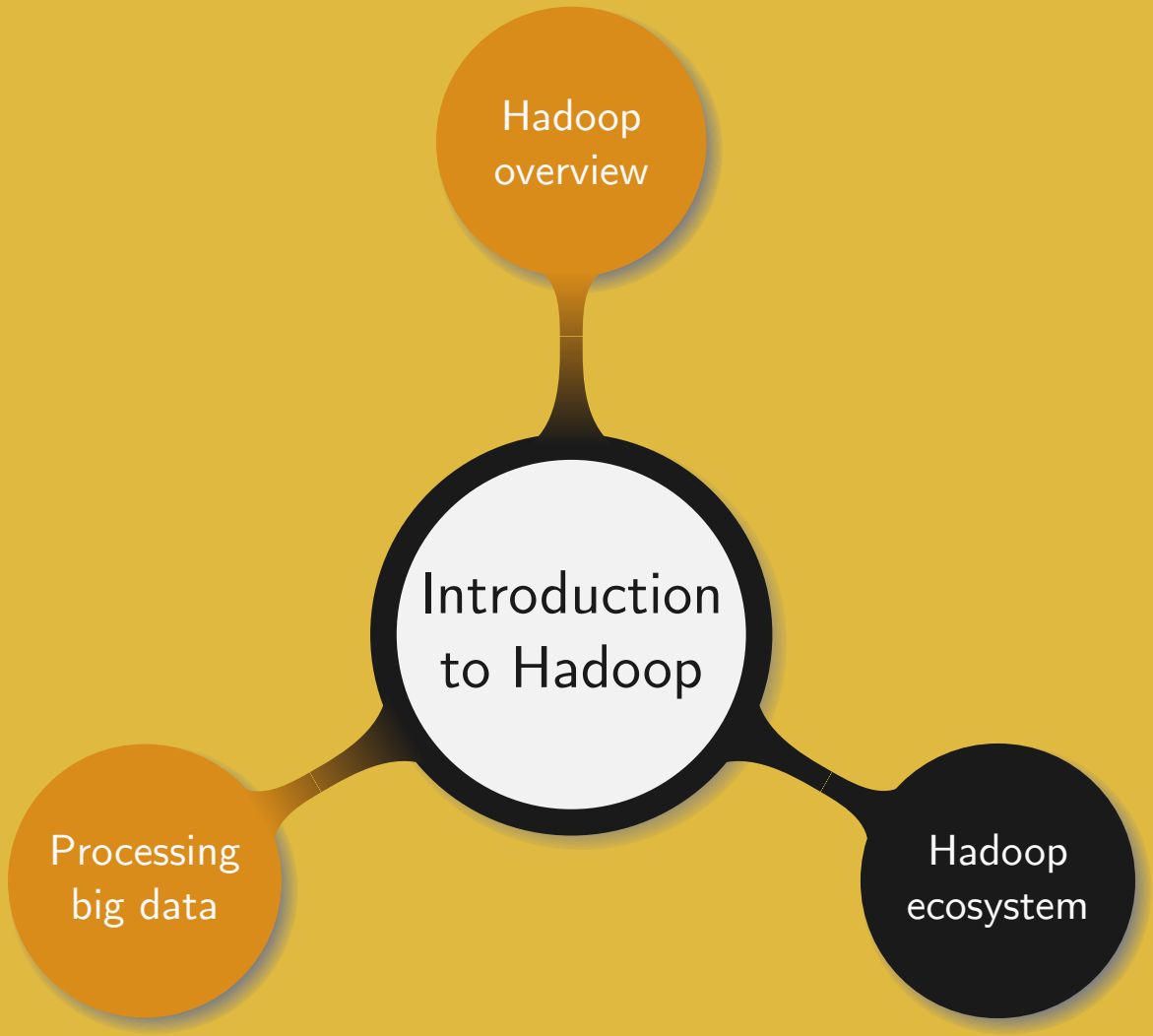
In Hadoop 1, MapReduce:

- Directly interacts with the filesystem
- Manages resources

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

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
- Optimistic execution: stream data in memory to minimise disk usage

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 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
- Exposes more simple framework and API to write YARN applications
- Expresses computation as a dataflow graph

HBase

Basics on HBase:

- 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:

- Access SQL data in HDFS using an SQL-like query language HQL
- Convert 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

Parquet:

- Columnar storage format
- Complex to handle

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

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

Analytics helpers

Major analytics helpers:

- Pig: high-level language to speak to MapReduce
- Hadoop streaming: write mappers/reducers in any language
- Mahout: set of scalable machine-learning algorithms for Hadoop
- MLlib: similar to Mahout, based on Spark (maintenance mode)
- Spark ML: similar to MLlib based on a higher level API
- 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

From batch to realtime processing

Lambda data architecture:

- Setup three layers:
  - Batch layer: store all incoming data and batch process it
  - Speed layer: analyse incoming data in real time
  - Serving layer: serve curated data that can be analysed by other tools
- Drawback: maintain two code sets for batch and speed layers

Kappa data architecture:

- Not a replacement but an alternative to lambda architecture
- Layers: batch layer is removed compared to lambda architecture
- Suitable for systems with strict end-to-end latency requirements
- Drawback: replay the whole stream in case of error

Realtime processing

Apache Storm:

- Distributed system for real-time processing of streaming data
- Able to process over a million records per second per cluster node
- Relies on Zookeeper for coordinating the nodes

Apache Kafka:

- Distributed platform used to create real-time streaming data pipelines
- Heavily relies on zerocopy (OS kernel level) to move data around
- Commonly used together with Spark, Flink, or Storm

Remote Dictionary Server (Redis):

- In-memory key-value data-store
- Extremely fast, simple, and versatile
- Benchmarked as the fastest DB in the world

AI and ML

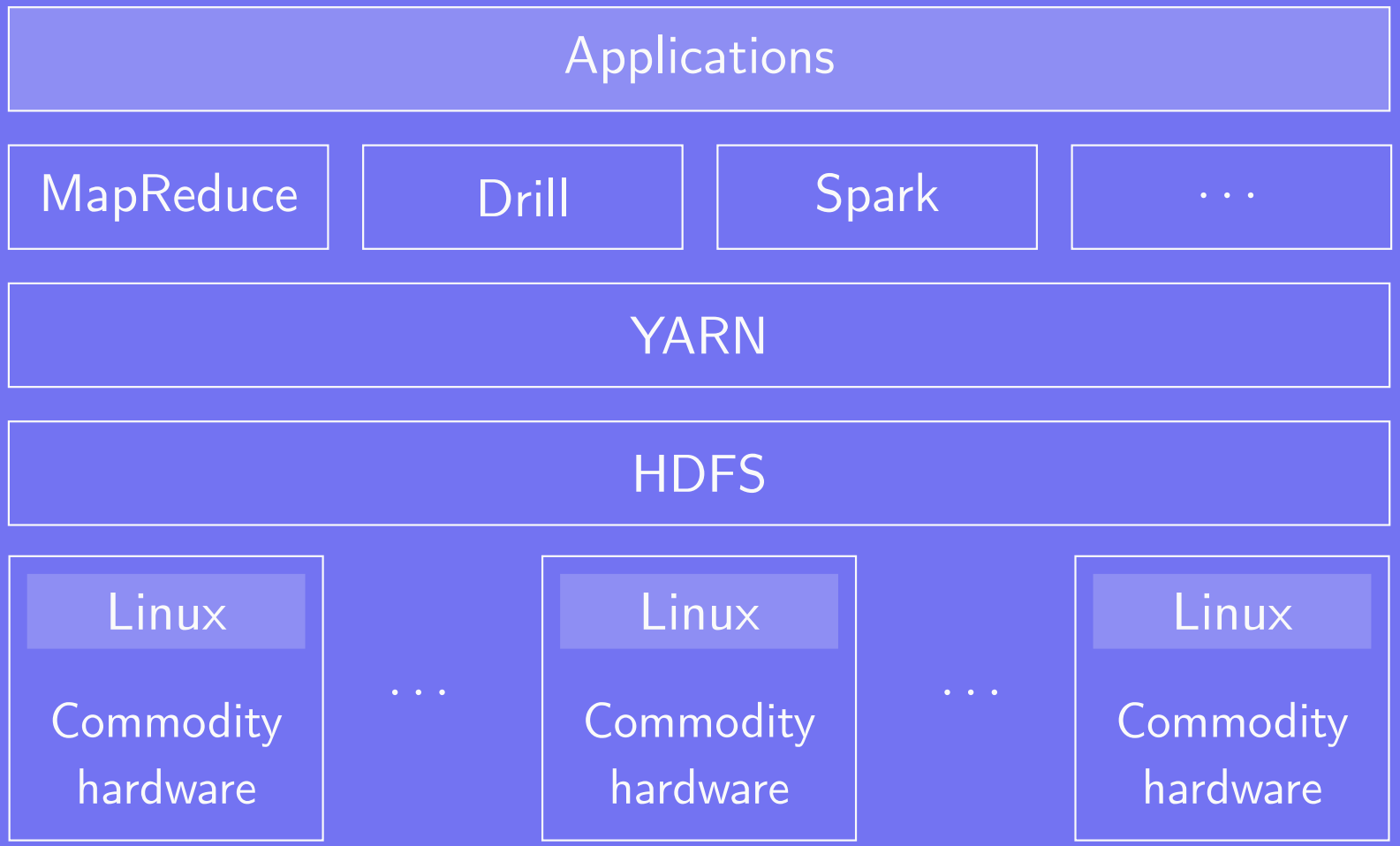
Ray:

- Spark's goal was to replace Hadoop and Ray's goal is to replace Spark
- Run fast machine learning or deep learning-based applications
- Hope to reach MPI power and granularity levels

TensorFlow:

- Python-friendly library for fast and easy machine learning computation
- Data is stored is a tensor
- Create dataflow graphs where the edges are tensors and vertices mathematical operations
- Can run on Hadoop (TonY), Spark, and Ray

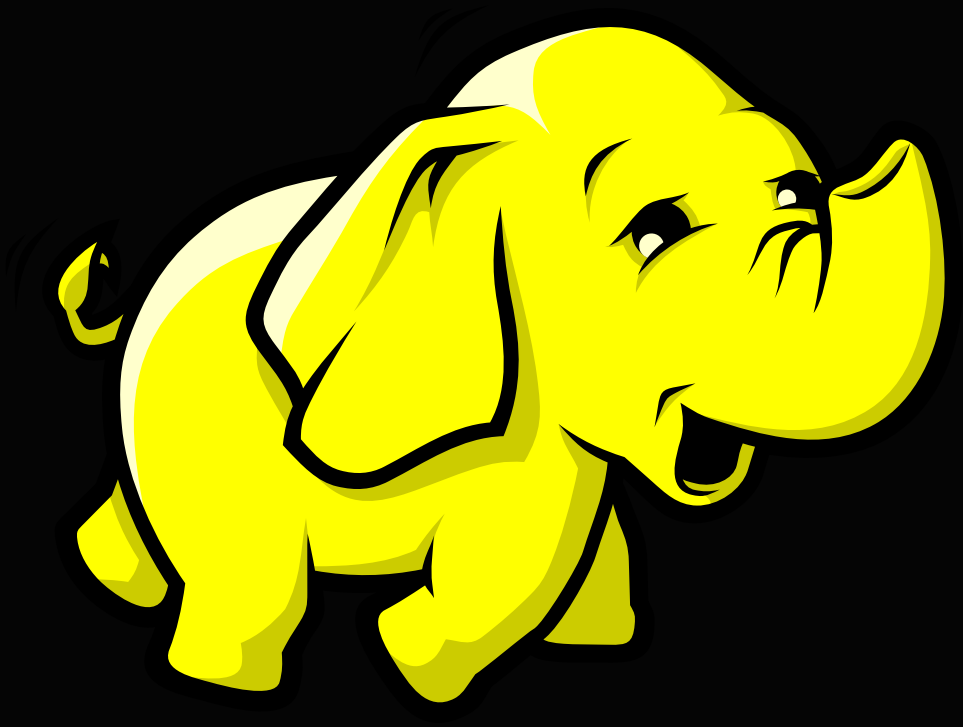
Hadoop ecosystem in VE472



Refer to Hadoop ecosystem table for more details

Key points

- What is big data?
- List limitations of common solutions when analysing large data sets?
- What is Hadoop?
- What are Hadoop's goals?
- Briefly describe the Hadoop ecosystem



Thank you!

References I

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