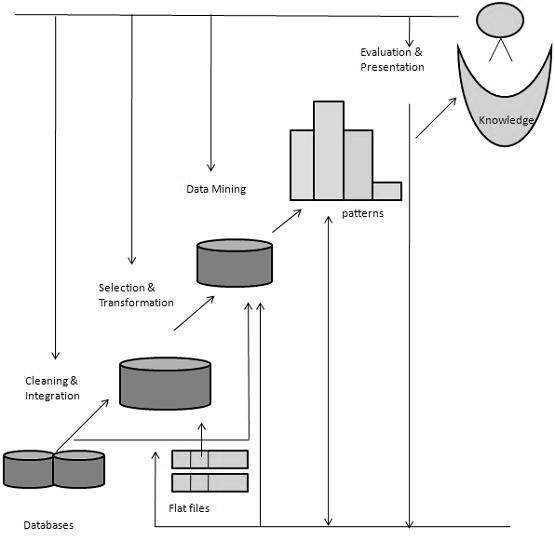
***Data mining***

**Introduction:**

* Can explain the term data mining in their own words.

Data mining is het proces van nuttige, bruikbare data te halen of te genereren uit een grotere set raw data. Dit kun je doen door bepaalde data patronen te analyseren met verschillende soorten software en eventueel de nuttige vergaarde data visueel te maken of nieuwe relaties in de data te vinden.

* Can describe the different steps in a data mining pipeline.
* Data collection:
  + Verzamel data van betrouwbare bronnen
  + Sla de data op
* Data cleaning:
  + Noisy, inconsistent en onvolledige data
  + Omgaan met ontbrekende velden
* Data integration:
  + Verschillende databronnen
  + Verschillende formaten
  + Redundant en dubbele data
* Data selection:
  + Relevante data ophalen
* Data transformation:
  + Normalisatie, aggregatie, generalisatie, etc.
* Data mining:
  + Extract patronen uit de data
  + Classificatie, predictie, clustering, etc.
* Pattern evaluation:
  + Interessante patronen filteren
* Knowledge representation:
  + Visualiseer de vergaarde kennis uit de data
* Can explain the different issues arising in data mining.

Mining methodology & user interaction:

* Verschillende soorten kennis minen in databases: verschillende gebruikers kunnen geïnteresseerd zijn in verschillende soorten kennis.
* Interactief minen van kennis op meerdere lagen van abstractie: mining moet interactief zijn omdat het gebruikers toelaat om te zoeken naar patterns en verfijnen van data mining-aanvragen op basis van de geretourneerde resultaten.
* Background knowledge inbrengen: om gevonden patterns te declareren
* Data mining query languages en ad hoc data mining (specifiek antwoord voor een business query)
* Presentatie en visualisatie van data mining resultaten
* Omgaan met noise of incomplete data
* Evalueren van patterns

Performance issues:

* Efficiëntie en scalability van data mining algoritmes
* Parallele, gedistribueerde en incremental mining algoritmes: The incremental algorithms, update de database zonder terug data te minen van scratch.

Diverse data types issues:

* Omgaan met relationele en complexe data types
* Informatie minen van heterogene databases (identieke software en werken samen om user inputs te verwerken) en globale informatiesystemen

**Big Data:**

* Can explain the term big data in their own words.

Enorm grote datasets die geanalyseerd kunnen worden om zo patronen, trends en associaties te herkennen. Vooral omtrent menselijke gewoontes en interacties.

* Can explain where big data comes from.

Enorme data boom: smart transport, smart factories, smart cities, smart buildings, social media, etc.

* Can explain the CAP theorem and their different components.

De CAP theorem tool is gemaakt voor system designers bewust te maken van de trade-offs wanneer ze data systemen ontwikkelen.

Kan maar 2 van de 3 hebben, big data is gedistribueerd dus altijd Partition-tolerance nodig.

Consistency:

Iedereen ziet dezelfde data op hetzelfde moment.

Availability:

Het systeem blijft normaal werken ook als er een node uitvalt.

Partition-tolerance:

Het systeem blijft normaal werken ook als er netwerk failures zijn.

* Can describe the difference between a data warehouse and a data lake.

Data warehouse:

Inkomende data wordt gecleaned en georganiseerd in een single consistent schema voordat het in het warehouse gezet wordt. Analyse wordt hier direct op de warehouse data gedaan.

Data lake:

Inkomende data wordt in het lake gegooid in zijn raw form zonder cleaning. Hier selecteren en organiseren we de data voor elke specifieke need.

* Can explain the difference between batch and real-time processing.

Batch processing:

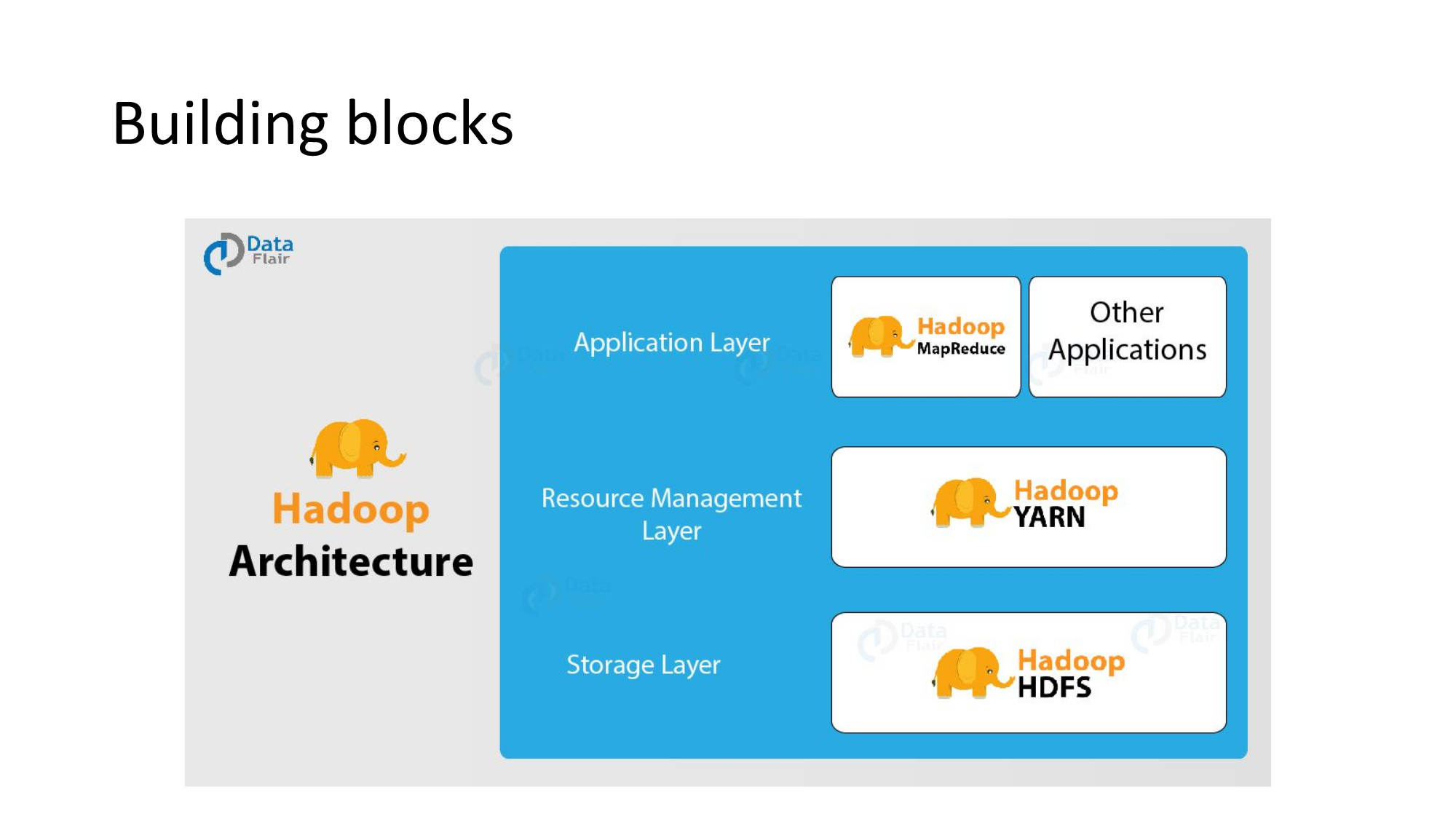
Het processen van de data wordt gedaan op een bepaald moment elke keer wanneer er genoeg data is verzameld.

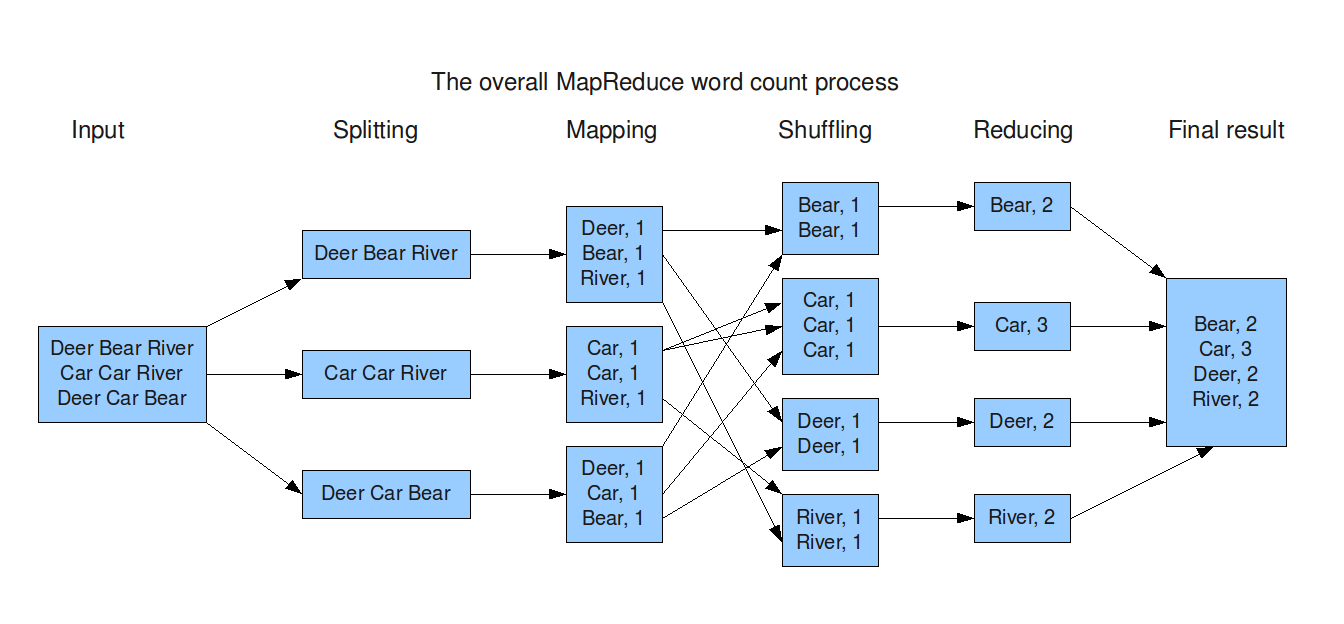
Real-time processing:

Data wordt onmiddellijk verwerkt wanneer het binnenkomt, de database wordt geupdate op dezelfde tijd dat het event gebeurd is.

**Hadoop:**

* Can list and visualize in an architectural diagram the different main building blocks of Hadoop.
* Can explain the principles of Map and Reduce in their own words.
* Can describe the main properties of HDFS in their own words.





MapReduce (how to execute):

* **MapReduce** is een processing techniek en een programma model voor distributed computing
* **Map** neem een set van data en convert deze naar een andere set van data, waarbij alle individuele elementen worden verdeeld in tupels (key/value pairs)
* **Reduce** neemt de output van de map-task en combineert deze data in kleiner set van tupels
* definieert welke computation je wilt uitvoeren op de data

YARN (resource management -> welke job wordt door welk systeem uitgevoerd):

* Yet Another Resource Negotiator
* verantwoordelijk voor het toewijzen van systeembronnen aan de verschillende applicaties die in een Hadoop-cluster worden uitgevoerd

HDFS (storage -> stores de job uitgevoerd door YARN)

* Hadoop Distributed Filesystem
* verspreid op verschillende machines in de cluster
* ontworpen voor te runnen op commodity/low-cost hardware
* high fault tolerant (hardware failure is de norm)
* vooral gemaakt voor batch processing (data access heeft high throughput)
* supports zeer grote data sets
* Can explain the functionality of the namenode and datanodes in HDFS.

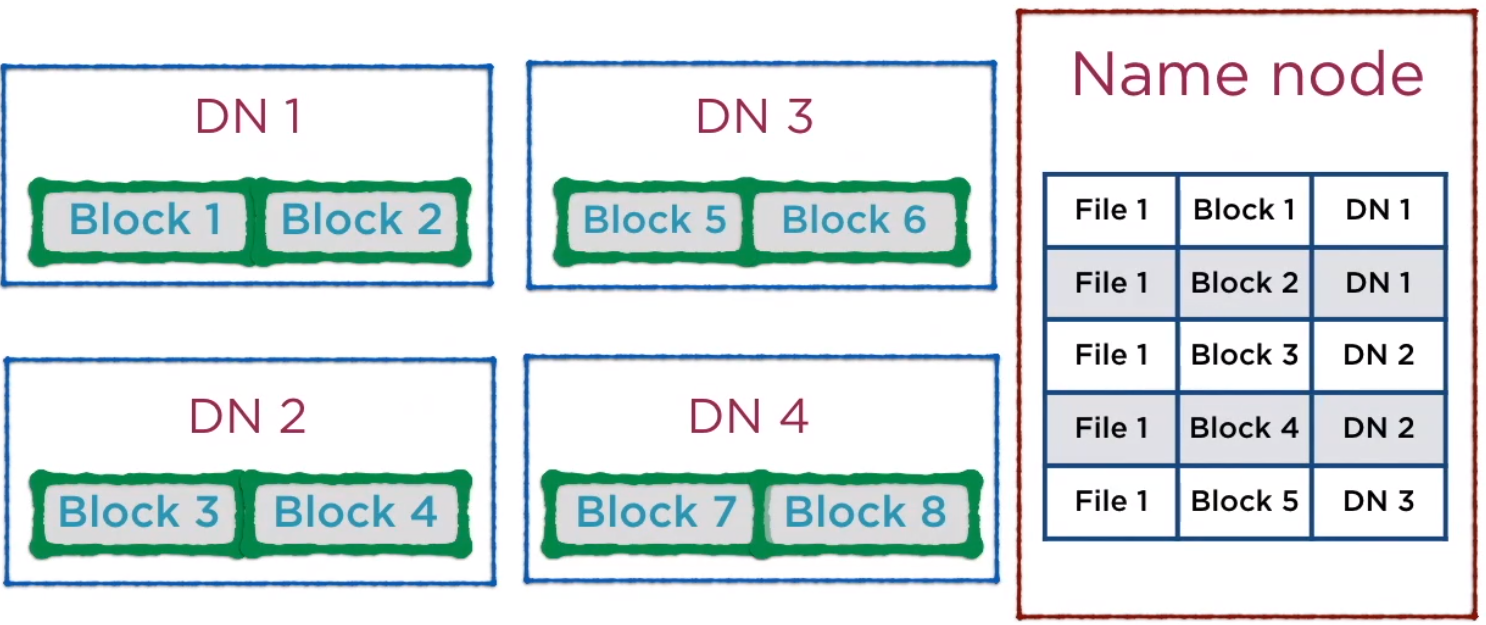
HDFS stores data over verschillende disks en systemen in de cluster en maakt gebruik van een master/slave architecture.

Namenode (table of contents):

* Master node.
* Coördineert opslag over de data nodes (slave nodes).
* Controleert de operaties van de data jobs.
* Gebruikersdata vloeit nooit door de NameNode.
* Manages the overall file system.
* Elke verandering naar de file system namespace of de properties hiervan is gerecord door de NameNode.

Datanode (holds the actual data):

* Schrijft data in blokken weg naar de lokale opslag.
* Repliceert data blokken naar andere datanodes.
* DataNodes are rack-aware (repliceert niet alle data naar dezelfde rek server, zodat als er een outage is niet alle data verloren is).
* Can describe the HDFS file storing process.



Opslaan:

1. Groot bestand wordt opgedeeld in meerdere kleinere “blocks” gelijk in fysieke grootte (aanbevolen grootte per block is 128MB, heeft te maken met seek en transfer time).
2. De blocks van data van de file worden verspreid over de datanodes zodat niet de hele file op één node staat. De locatie van elke block wordt bijgehouden in de namenode.
3. Is ook de optie voor replication en fault-tolerance, niet de hele file wordt meerdere keren opgeslagen. Iedere block wordt hier dan meerdere keren opgeslagen.

Uitlezen:

1. Een client stuurt een request naar de namenode met de file die de client wil terugkrijgen.
2. De namenode gaat nu kijken waar de eerste block van deze file staat en gaat deze block ophalen in de juiste datanode, de inhoud hiervan wordt teruggestuurd naar de client.
3. Hierna gaat de namenode kijken waar de volgende block staat en herhaalt dit proces tot het de hele file heeft teruggegeven aan de client.

* Can explain how Hadoop deals with failures.

Replica's van blokken worden opgeslagen op verschillende locaties. Namenode maakt gebruikt van een transactielog namelijk EditLog voor elke aanpassing te loggen dat gebeurt met de file system metadata. Het gehele file system namespace (inclusief de mapping van de blokken) wordt opgeslagen in een file genaamd FslImage. Deze files kunnen samen een namenode reconstructen maar is een computationally heavy operation.

* Can explain Combiners and Partitioners in MapReduce in their own words.

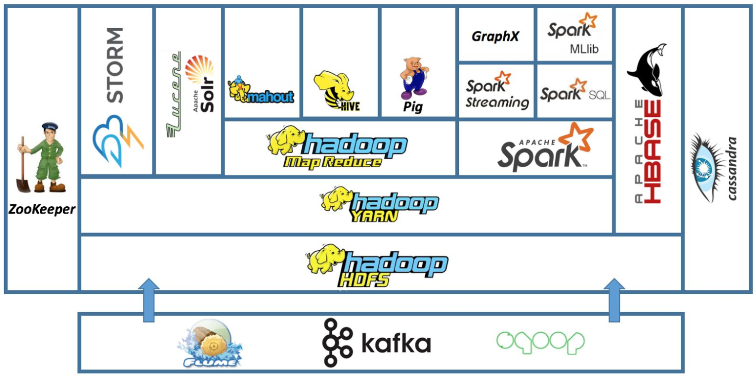
De belangrijkste functie van een Combiner is het samenvatten van de map output recorders met dezelfde sleutel. De output (key/value -collection) van de combiner wordt als invoer via het netwerk naar de daadwerkelijke Reducer-taak verzonden. Combiners verminderen network congestion en reduces volume of data transfers tussen map en reduce.

Het aantal partitioners is gelijk aan het aantal reducers. Dit betekend dat een partitioner data zal verdelen op basis van het aantal reducers. Zorgt ervoor dat alle values met een single key naar dezelfde reducer gaan.

* Can implement MapReduce for a given problem.

**Hadoop ecosystem & Kafka:**

* Understands the Hadoop ecosystem, and can describe the different components (data ingestion, messaging, coordination) in a general way.



Data ingestion & Collection:

* Sqoop
  + SQL to hadoop
  + RDBMS import/export between data warehouses and Hadoop
* Flume
  + Verzamelt log data van gedistribueerde web servers
  + Special purpose tool om data in HDFS te sturen
  + Specifiek voor Hadoop
* Kafka
  + General purpose
  + Ondersteunt data streams voor meerdere applicaties

Messaging:

* NiFi
  + Automatiseert de flow van data tussen de software systemen
  + Tools om data te processen
  + Kan om met een grote variatie databronnen en formaten, om deze dan te transformeren en in een andere databron te duwen.

Coordination:

* Zookeeper
  + Distributed configuration service
  + Distributed synchronization service
  + Naming registry
  + Coordinates clusters

Query languages:

* Hive
* Pig

Database systems:

* HBase (CP)
* Cassandra (AP)

File system:

* Hadoop (HDFS, MapReduce)
* Can explain the role of Zookeeper in the Hadoop ecosystem.

Zookeeper wordt gebruikt om grote clusters van machines te managen en te coördineren.

Als er veel machines in een netwerk hangen kunnen er problemen ontstaan die Zookeeper kan oplossen zoals:

Race condition

* Gebeurt als een machine 2 of meer operations tegelijk probeert te doen.
  + Opgelost door de serialization property van Zookeeper

Deadlocks

* Gebeurt als 2 of meer machines dezelfde gedeelde resource willen gebruiken, ze proberen elkaars resources te accessen om meer precies te zijn wat zorgt voor een lock of the system.
  + De synchronisatie in Zookeeper zorgt ervoor dat die niet kan gebeuren.

Partial failure of a process

* Gebeurt als een process halverwege failed, wat kan zorgen voor inconsistente data
  + Zookeeper lost dit op door zijn atomicity, maw ofwel lukt het hele process en schrijft dit data weg, ofwel mislukt het process en wordt er geen data aangepast.
* Can explain the role of NiFi in the Hadoop ecosystem.

Nifi wordt gebruikt om de flow van data tussen software systemen te automatiseren en data te processen.

Nifi kan ook om met een grote variatie databronnen en formaten, om deze te dan te transformeren en in een andere databron te duwen. Het kan dus data valideren, verrijken en transformeren.

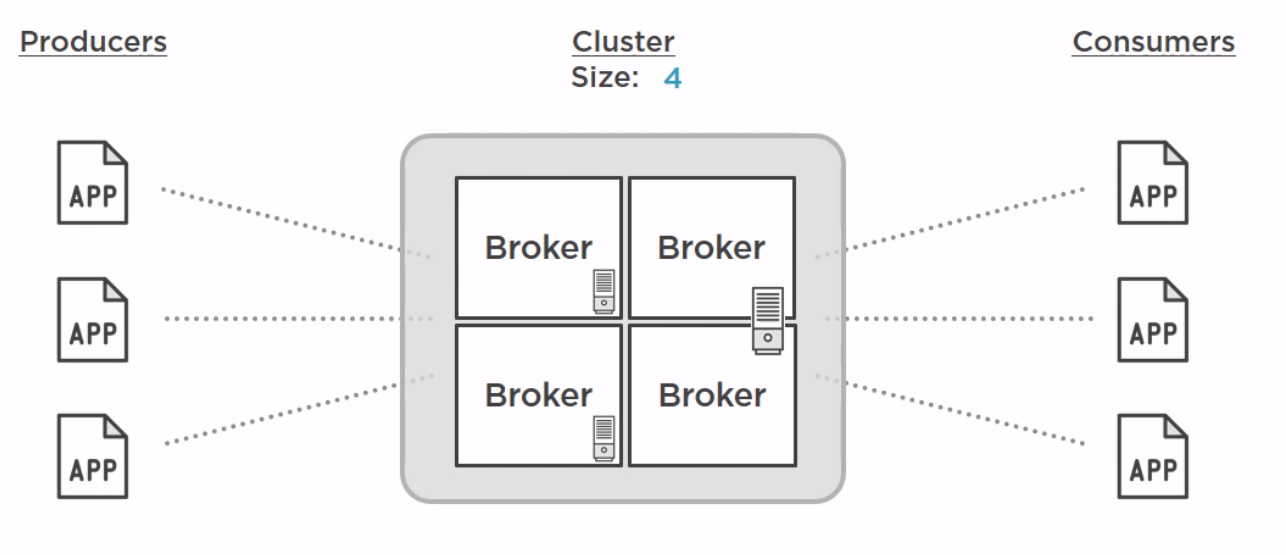
* Can explain the problem-situation leading to the development of Kafka.

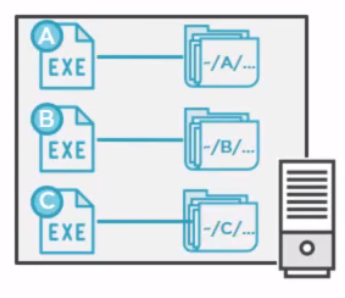
LinkedIn wou een low-latency oplossing hebben om grote sets van data in een architectuur te krijgen om het zo in Hadoop te gooien bijvoorbeeld, maar toen in de tijd was er geen oplossing voor “real-time” processing voor het binnenkrijgen van data dus hadden ze Kafka ontwikkeld.

* Can describe the different attributes of Kafka (distributed messaging, scalable, etc).

Kafka

* Distributed messaging framework
* High throughput / high volume
* Horizontally scalable
* Loosely coupled producers and consumers
  + Publisher-subscriber model
* Functions as a distributed commit log
* Can explain the publish/subscribe model of Kafka.





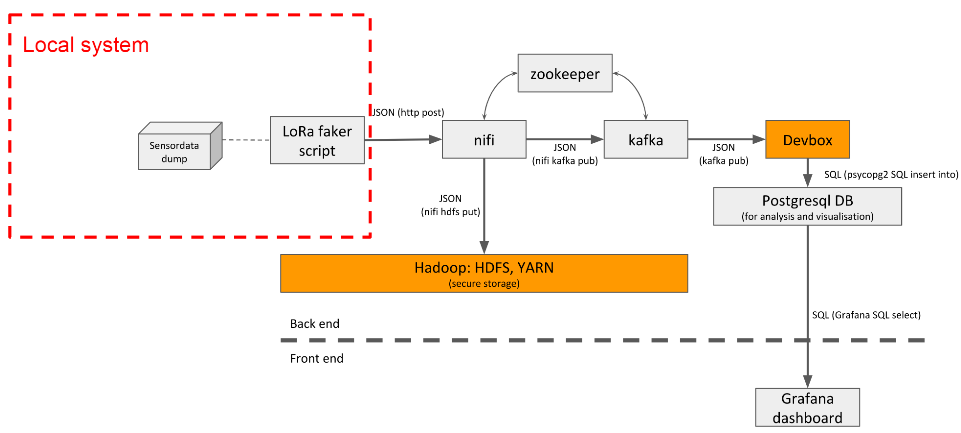
Een producer (publisher) kan naar een bepaalde topic publishen, in een broker zitten 1 of meerdere topics. Op 1 machine (fysiek of virtueel) kunnen 1 of meerdere brokers draaien. In een cluster zitten 1 of meerdere brokers.

Als een consumer (subscriber) bepaalde data nodig heeft kan deze subscriben op een specifieke topic in een broker.

* Can set up a minimal Hadoop ecosystem using HDFS, Kafka and NiFi.

**Smart eXperience Campus:**

* Can setup the SXC hadoop ecosystem.
* Understands the architecture of Smart eXperience Campus and can explain the different components.



* Can implement MapReduce in a Hadoop ecosystem.

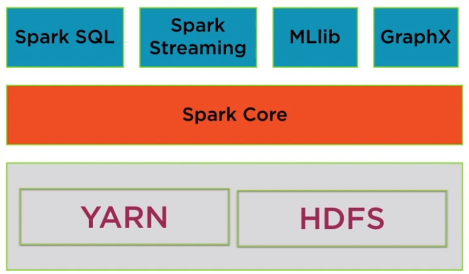
**Spark:**

* Can describe the differences between Spark and Hadoop's MapReduce program model.

Het grootste verschil tussen Spark en Hadoop’s MapReduce is dat Spark het eenmaal na het inlezen volledig in memory werkt wat veel sneller is dan het steeds wegschrijven en terug ophalen wat de MapReduce van Hadoop doet.

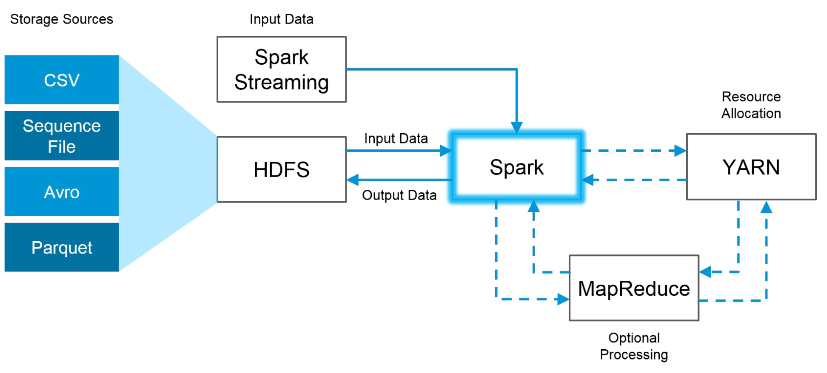
Het voordeel aan Hadoop’s MapReduce is wel dat deze met veel grotere datasets om kan gaan dan Spark, maar Spark kan dan weer near real-time processing doen.

* Can situate Spark in a Hadoop ecosystem and architecture, and explain its function in that framework.

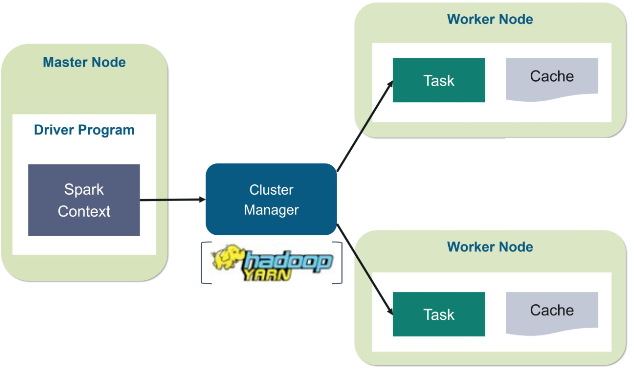


Spark wordt vooral gebruikt in het Hadoop ecosysteem voor:

* Real-time (stream) en batch processing
* SQL support
* Machine Learning Library
* Graph processing framework



1. Spark krijgt data binnen via HDFS of Spark streaming
2. Spark deel de gevraagde taak op in kleinere taken (basic units of execution) in de Spark context die draait in de master node.
3. Deze kleinere taken stuurt Spark door naar de clustermanager (YARN).
4. Dan zorgt YARN ervoor dat al deze kleine taken verdeeld worden over al de worker of slave nodes.
5. De uitkomst van deze taken worden dan direct teruggestuurd naar de Spark Context in de master node.



* Can explain what an RDD is and its characteristics (resilient, distributed, dataset, immutable, cacheable).

Een RDD (Resilient Distributed Dataset) is een collectie van entities (ints, strings, objects, records, rows, …).

Resilient:

* Als een node crasht die een RDD heeft, kan die bepaalde RDD gereconstrueerd worden van zijn raw form. Dit komt omdat er voor elke RDD elke transformatie (chain of transformations) of inlezing wordt bijgehouden de metadata houdt alles bij van hoe de RDD naar de huidige staat gekomen is (lineage van de RDD, helemaal terug naar de bron).

Distributed:

* De volledige inhoud van een RDD is gesplitst over de datanodes in de cluster. (partitioned)

Dataset:

* Collection of entities (rows).

Immutable:

* Eens dat er een RDD is kan deze niet aangepast worden, transformaties maken gewoon een nieuwe RDD ipv de originele aan te passen.

Cacheable:

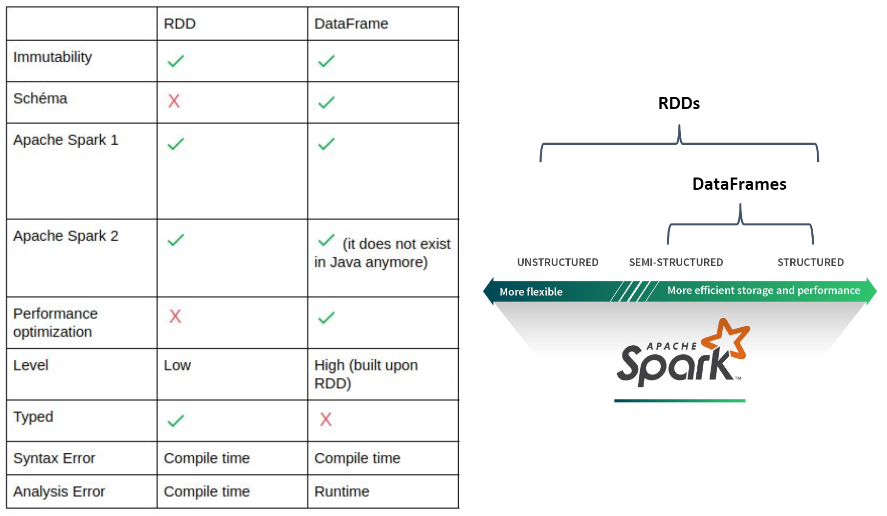
* Zijn volledig in-memory objects.
* Can describe in their own words the different methods possible on an RDD (transformations, actions).

Op een RDD kan je maar 2 soorten operaties toepassen:

* Transformaties
  + Transformeert de originele RDD in een nieuwe RDD. Transformaties worden pas toegepast als er via een action iets van de RDD request wordt (lazy evaluation). De volledige lijst van transformaties worden bijgehouden in een Directed Acyclic Graph (DAG).
* Actions
  + Een methode die een result krijgt van een RDD (read data from RDD).
* Can explain the relationship between Spark dataframes and RDDs in their own words.

Dataframes zijn op RDDs gebouwd en zijn dus ook partitioned, immutable, resilient, …

Dataframes zijn wel semi-structured of structured en zijn efficiënter omtrent opslag en performantie, maar zijn dus ook minder flexibel.



* Can explain the need for accumulators and broadcast variables in a distributed Spark system.

Normaal werd gebruik gebruik gemaakt van Closures, probleem hiermee is dat je hier 1 kopie per task moet hebben (dus op 1 node kon meerder keer dezelfde variabele staan), en deze variabele kon enkel gekopieerd worden naar iedere task vanaf de master, dus niet peer to peer.

* Can explain what broadcast variables and accumulators are and describe the differences between them.

Broadcast variable:

* Gedeelde, read-only variable
* 1 kopie per node
* Kan gekopieerd worden van worker node naar worker node (peer to peer).

Accumulator:

* Gedeelde, read-write variable
* 1 kopie per node
* Vaak gebruikt als counter of sum
* Kan enkel werken met additions die associatief en commutatief zijn.
  + Associatief: A + (B + C) = (A + B) + C
  + Commutatief: A + B = B + A
* Native support voor volgende types
  + Long
  + Double
  + Collections
* Can work with jupyter notebooks in Spark and use the Dataframe and RDD programming abstractions to process and analyze data.

**Stream processing:**

* Can explain in their own words what stream processing is.

Stream processing is de data onmiddellijk verwerken op het moment dat het gemaakt wordt of het de data binnenkrijgt.

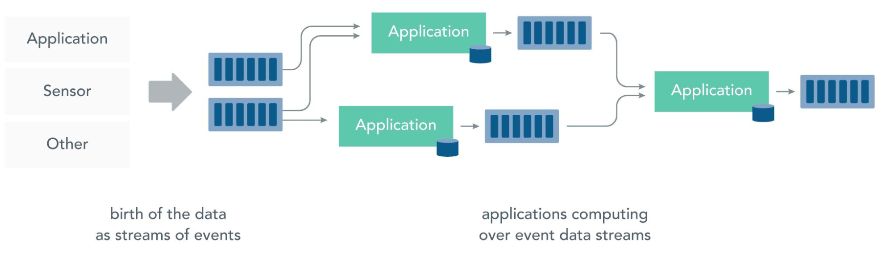
* Can explain the various needs for stream processing using a real world example, and contrast this with the limitations of batch processing.

Bij sommige toepassingen is het belangrijk dat de data die binnenkomt onmiddellijk verwerkt wordt in plaats van op een vast moment in de avond.

Een voorbeeld hiervan is de website van een bank, als er enorm veel requests zijn op deze site en de site kan deze niet aan en geeft errors moeten deze errors onmiddellijk verwerkt worden zodat deze ook in real-time gemonitord kunnen worden.

Moest hier gebruik gemaakt worden van batch processing dan zagen de beheerders van de website pas op het einde van de dag na het vaste moment in de avond dat de data verwerkt wordt bijvoorbeeld dat er deze ochtend veel problemen waren met de website en is deze data dus eigenlijk al nutteloos geworden.

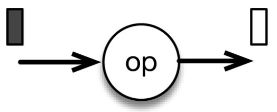
* Can describe a stream processing pipeline.



De hele tijd vloeit er data door de stream processing pipeline, vanaf dat het moment dat er data binnenkomt gaat deze hierdoor en gaat een stream processing applicatie hier iets mee doen (een actie triggeren, een statistic updaten, dit event onthouden voor een toekomstige referentie).

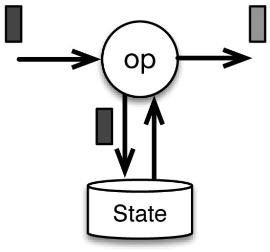
Zo een processing applicatie kan ook meerdere data streams combineren en kan ook als output een nieuwe datastream hebben voor een eventuele volgende processing applicatie.

* Can explain the difference between stateless and stateful stream processing.



Stateless stream processing:

* Transformaties die toegepast worden op een single piece of data.
  + Bijvoorbeeld:
    - Batch processing van een file
    - map(), filter()



Stateful stream processing:

* Transformaties die accumuleren over de data heen.
* Geaccumuleerde informatie over alle of een deel van de entities in een stream.
* Houdt de contextuele state bij.
  + Wordt gebruikt om informatie afgeleid uit vorige geziene events op te slaan.
* Can explain how stream processing works in Spark on the basis of DStreams.

DStreams (Discretized Streams) zijn opgebouwd uit meerdere RDDs

* Can implement a stream processing pipeline in Spark based on existing architectures.
* Can implement stream processing algorithms (data analysis and machine learning).

**Recommendation systems:**

* Can give a definition of a recommendation system in their own words.
* Can describe why recommendation systems are needed.
* Can describe the pipeline of a recommendation system (data => insight/monetization).
* Can visualize the most commonly used taxonomy of recommendation systems.
* Can explain how content based filtering works in their own words and give an example of its execution.
* Can explain the characteristics and difficulties of content based filtering.
* Can explain what the Rating matrix, or User-Item interactions matrix is, and what it's used for.
* Can explain memory-based user-user collaborative filtering in their own words.
* Can explain the characteristics and difficulties of memory-based user-user collaborative filtering.
* Can explain memory-based item-item collaborative filtering in their own words.
* Can explain the characteristics and difficulties of memory-based item-item collaborative filtering.
* Can explain the differences between user-user and item-item memory based collaborative filtering.
* Can explain the problems and difficulties of Nearest-Neighbour (memory) based collaborative filtering.
* Can implement a memory based collaborative filtering recommendation system.
* Can explain model based collaborative filtering in their own words.
* Can explain matrix factorization in the context of model based collaborative filtering in their own words.
* Can explain the differences between memory based and model based collaborative filtering.
* Can implement a matrix factorization model based collaborative filtering recommendation system.
* Can explain the "cold start problem" in the context of collaborative filtering.
* Can explain the difficulties in evaluating a recommendation system.