CORSO I.F.T.S

“TECNICHE PER LA PROGETTAZIONE E LA GESTIONE DI DATABASE ”

Ing. Mariano Di Claudio

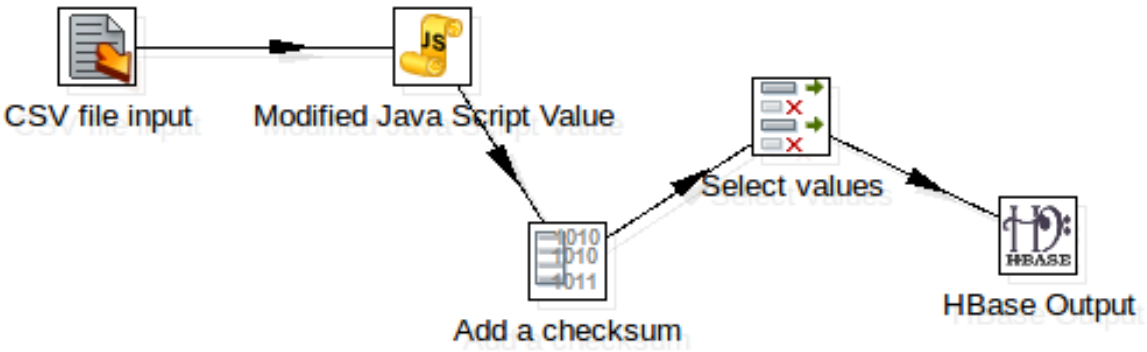
Lezione del 22/10/2014

Indice

1. **PDI Kettle e Hbase**
   - Dettagli esercizio di data ingestion e trasformazione
2. **Riepilogo lezioni precedenti**

Trasformazione Hbase Ouput

- Questa trasformazione prende ingresso il file musei in formato CSV, e dopo aver definito una chiave, carica i dati in una tabella (opere_pubbliche) di HBase.

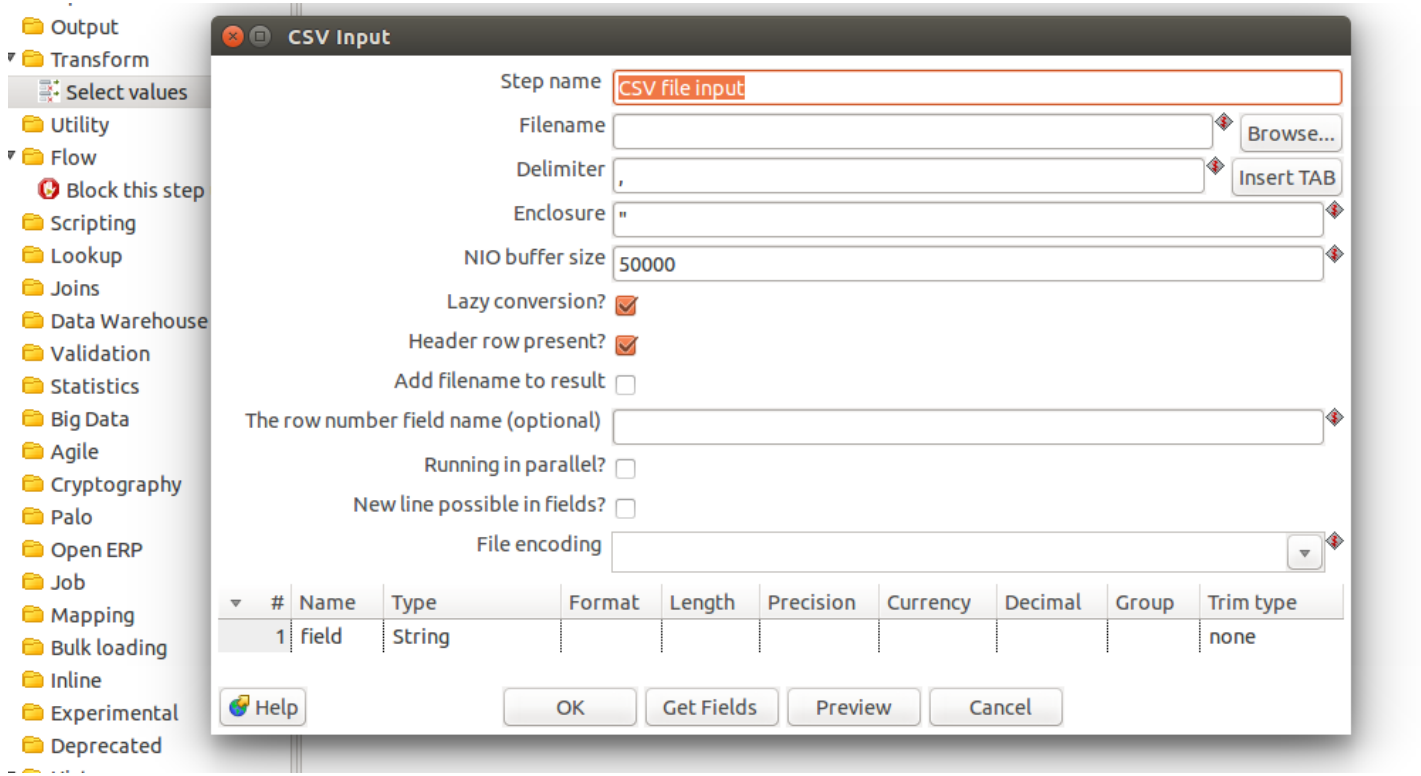


- La trasformazione si compone di 5 step.

Trasformazione Hbase Ouput - Step

CSV file input

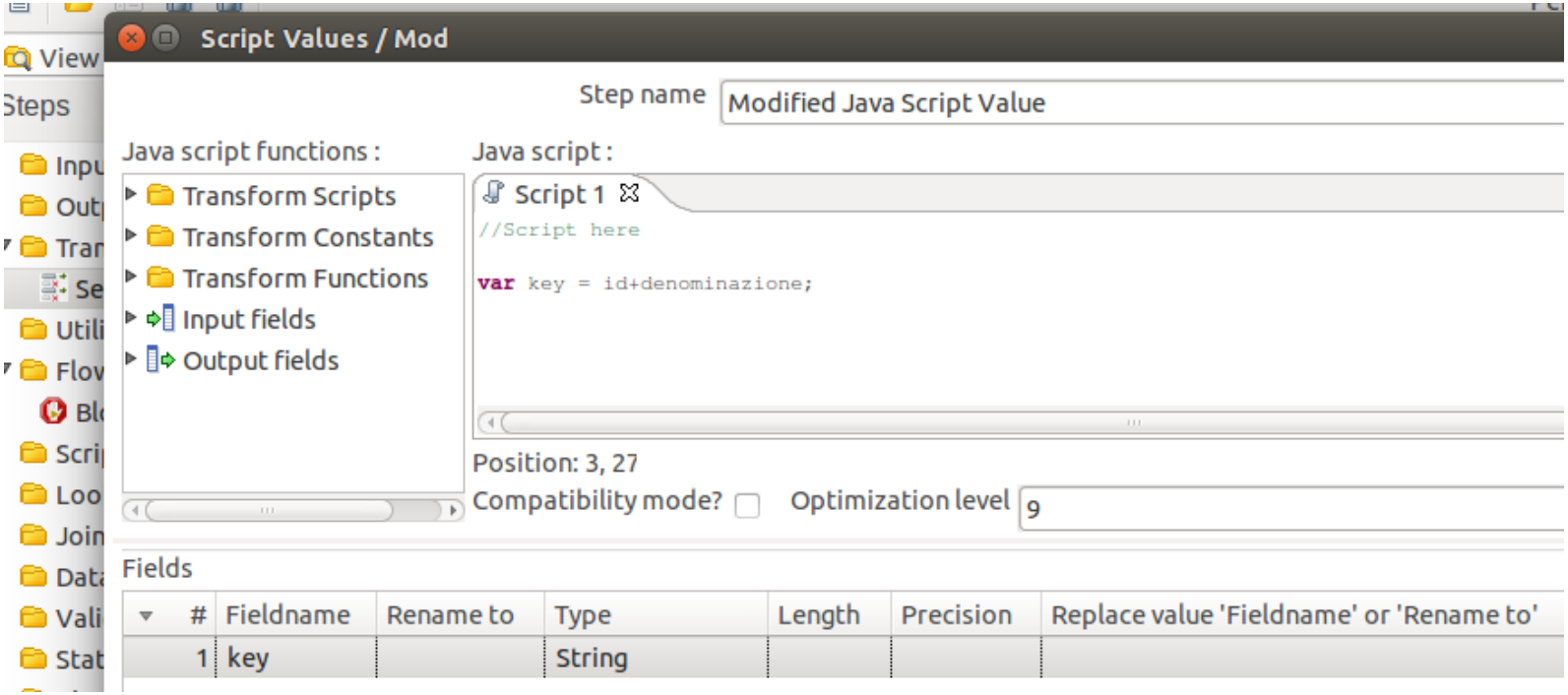
- In questo step viene selezionato il file CSV, si sceglie il tipo di separatore utilizzato, e tramite il pulsante *Get field* si selezionano i campi da importare (se ne può stabilire il tipo e altri parametri).



Trasformazione Hbase Ouput - Step

Modified Java Script value

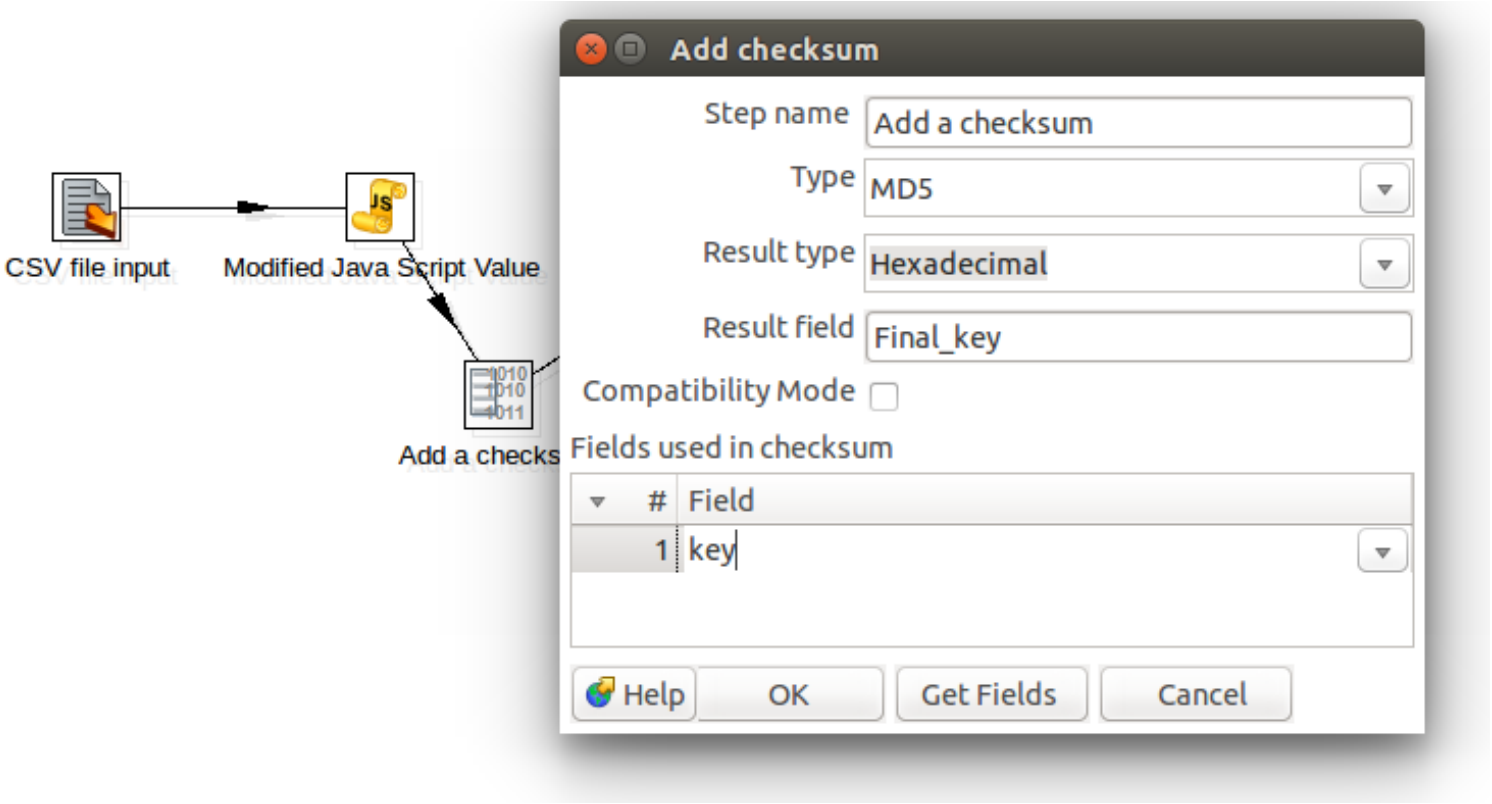
- In questo step è possibile aggiungere del codice JavaScript. Si definirà una variabile concatenando due campi di input, e alla fine la stessa variabile verrà utilizzata per definire un field in uscita dallo step.



Trasformazione Hbase Ouput - Step

Add a Checksum

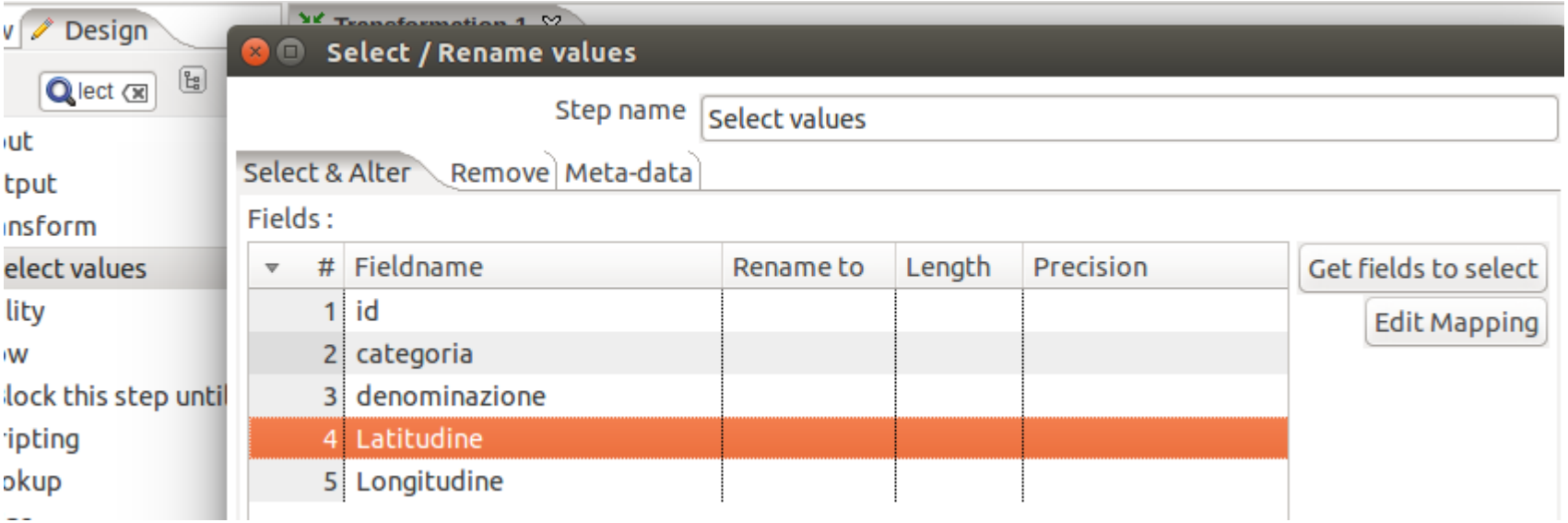
- Questo step permette di scegliere con quale algoritmo (MD5, CRC32) codificare un campo (solitamente la chiave), e di definirne il nuovo nome in uscita.



Trasformazione Hbase Ouput - Step

Select Values

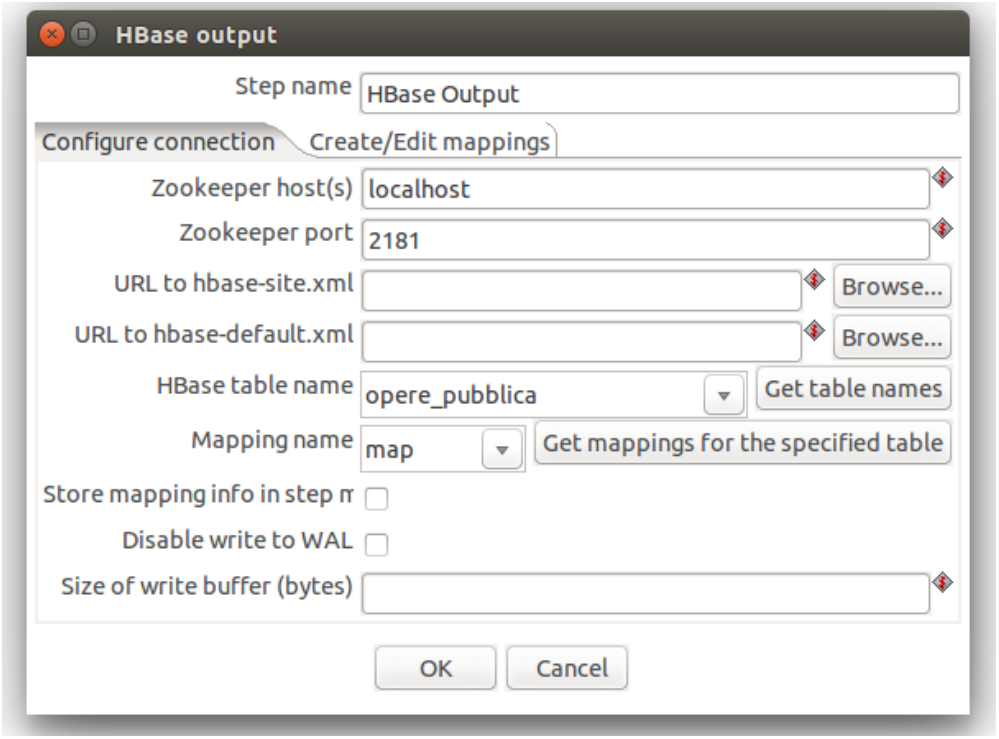
- In questo step è possibile selezionare i campi (uno o più) che si vuole far passare allo step successivo; si può anche utilizzare l'opzione *remove* per selezionare i campi da bloccare.



Trasformazione Hbase Ouput - Step

Hbase Output

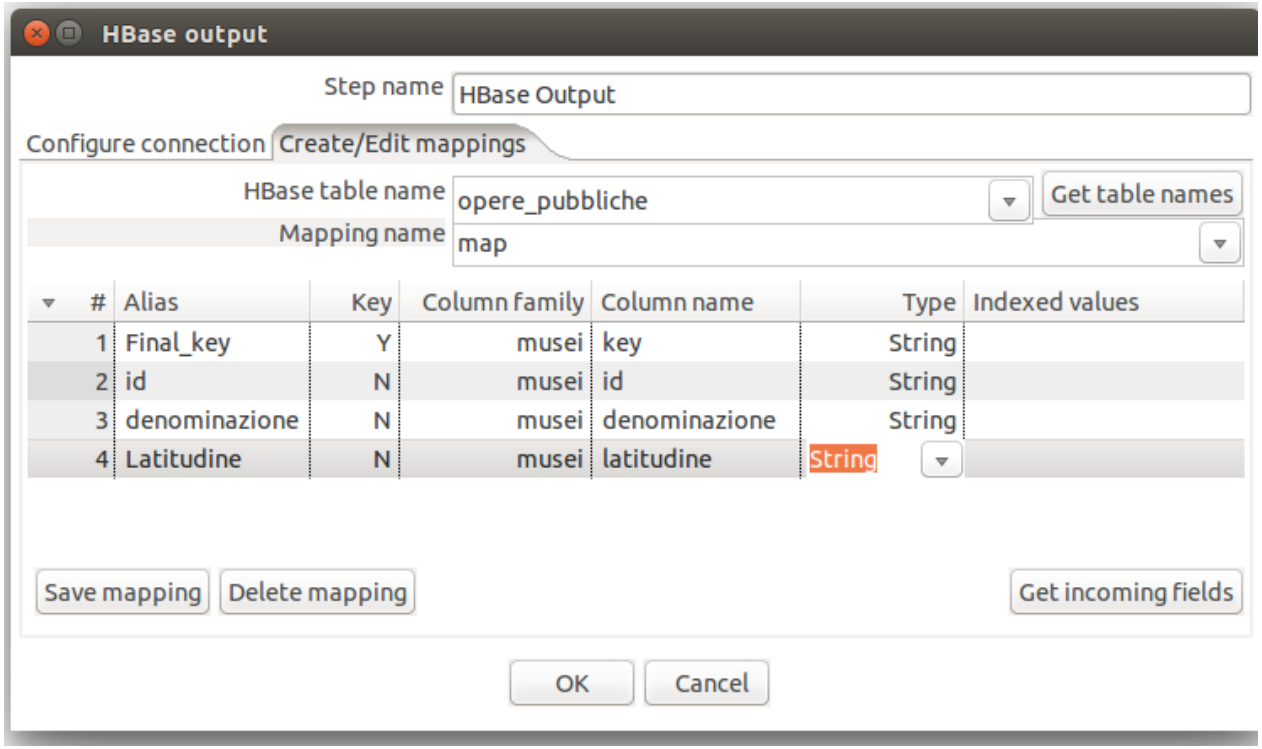
- In questo step si settano i parametri per caricare i dati in una tabella HBase.
- *Nella prima tab si definiscono l'IP della macchina che ospita il DB e la porta (2181).*
- Nella seconda tab si selezionano la tabella, i campi da caricare e il mapping.



Trasformazione Hbase Ouput - Step

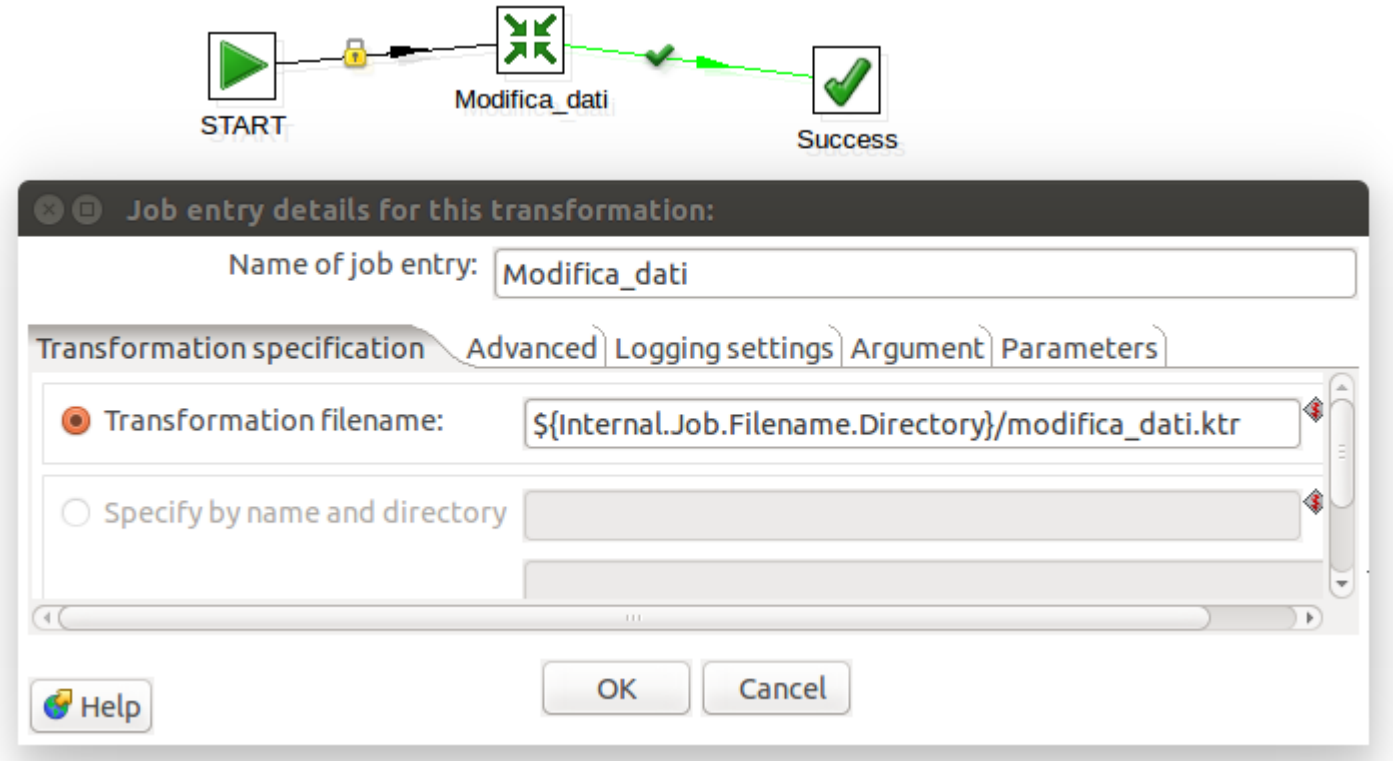
Hbase Output

- In questo step si settano i parametri per caricare i dati in una tabella HBase.
- Nella prima tab si definiscono l'IP della macchina che ospita il DB e la porta (2181).
- *Nella seconda tab si selezionano la tabella, i campi da caricare e il mapping.*



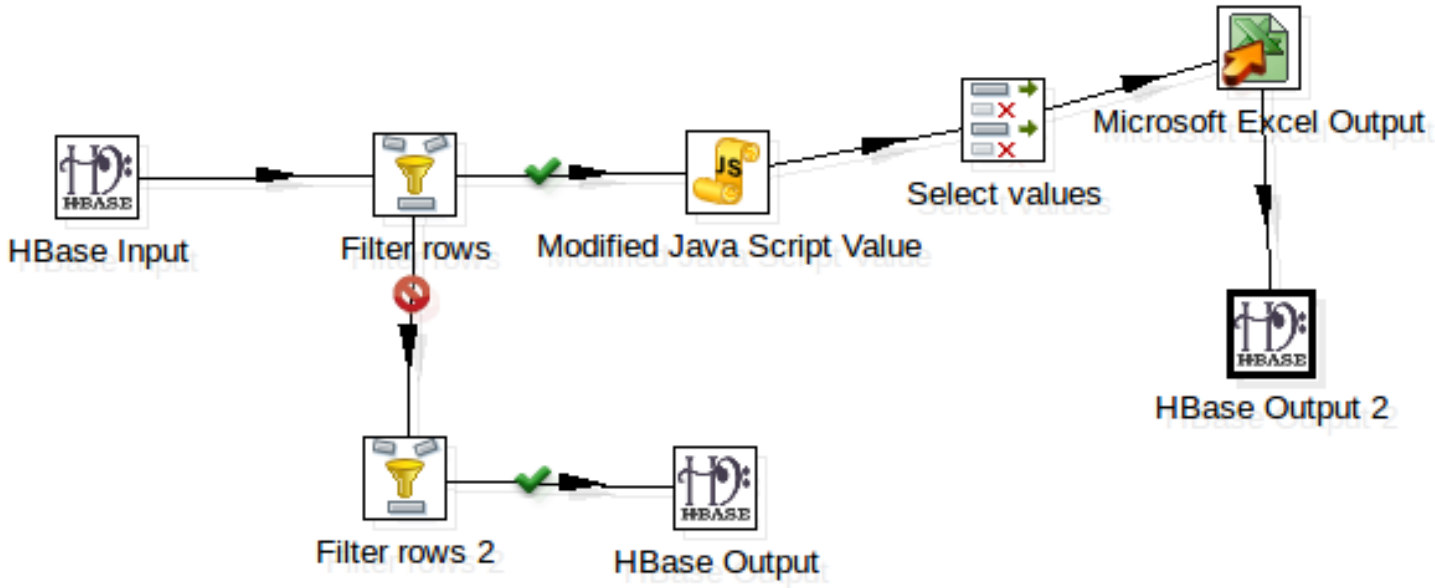
Job

- Creazione di un job che al suo interno conterrà lo **step iniziale** (start), lo **step Modifica_dati** (trasformazione selezionata) e lo **step finale** (Success)



Trasformazione Mod_dati

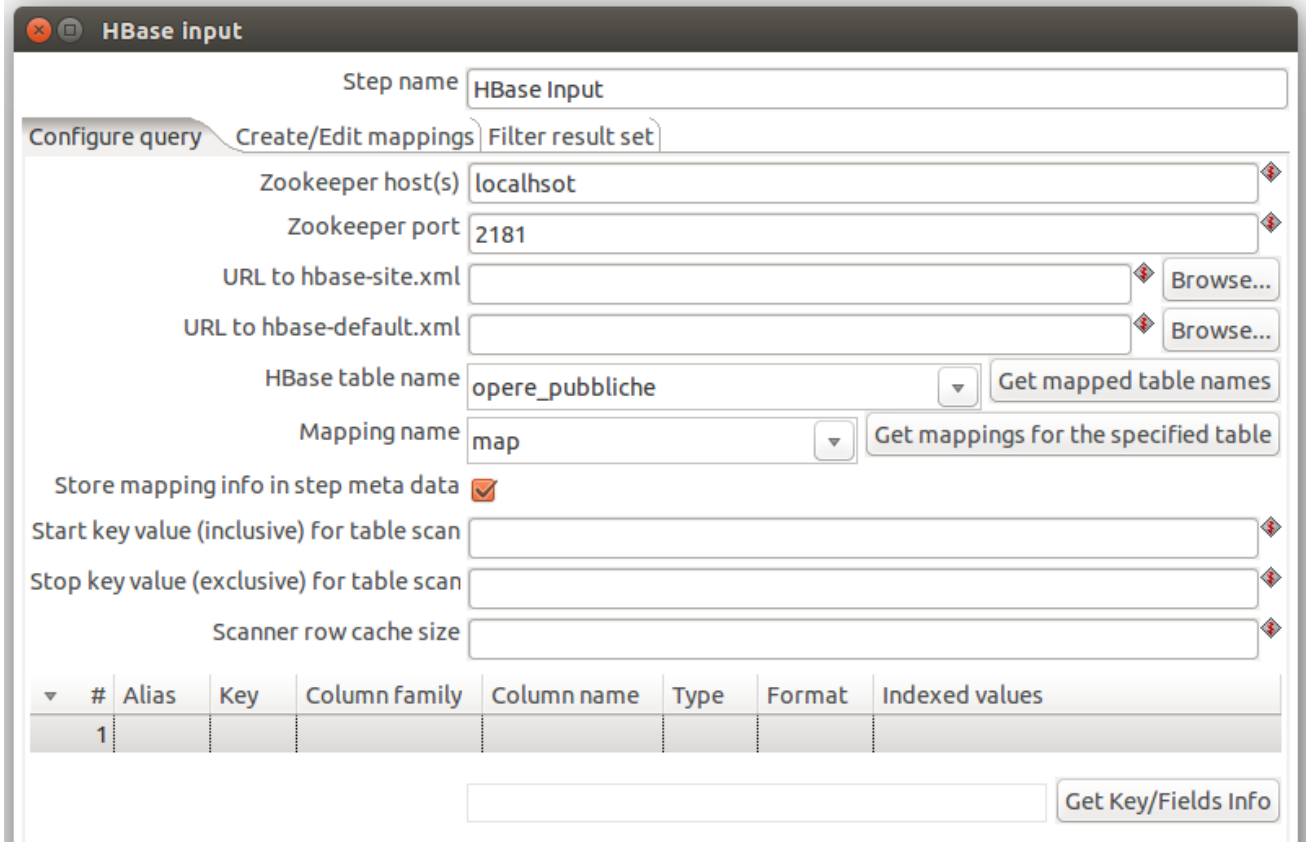
- Questa trasformazione recupera i dati presenti su HBase mediante lo step *HBase Input.* Poi opera applica gli step *Filter rows e Modified Java Script Value,* per effettuare alcune operazioni di pulizia sui dati prima di memorizzarli nuovamente su HBase.



Trasformazione Mod_dati - Step

Hbase Input

- In questo step si settano i parametri per recuperare i dati presenti su Hbase.
- Si può effettuare un filtraggio impostando le condizioni nella tab ***Filter result set***

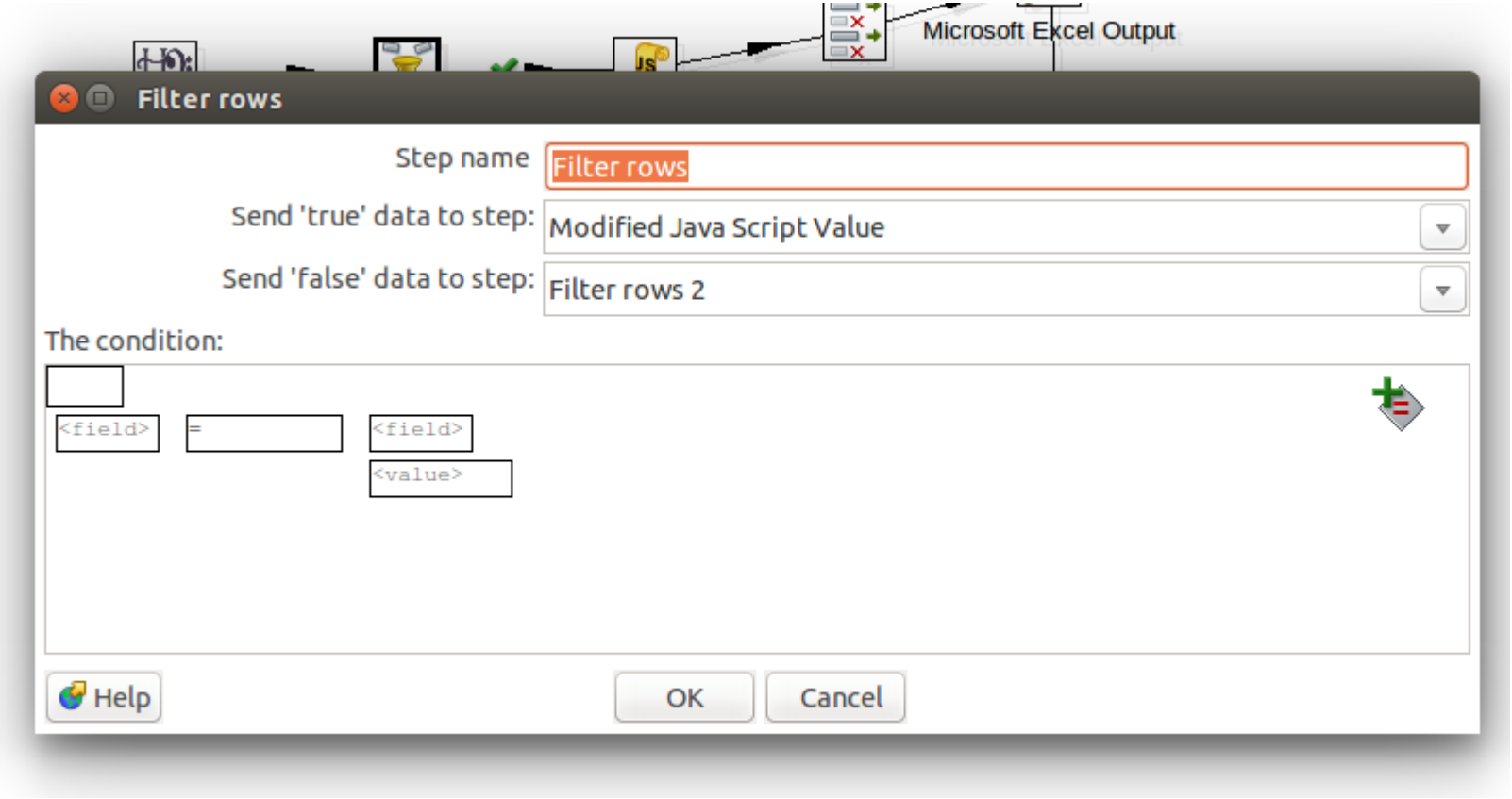


Trasformazione Mod_dati - Step

Filter rows

- E’ un altro modo per filtrare il flusso di dati sulla base di una specifica condizione.

Questo step va a creare una biforcazione del flusso di dati.

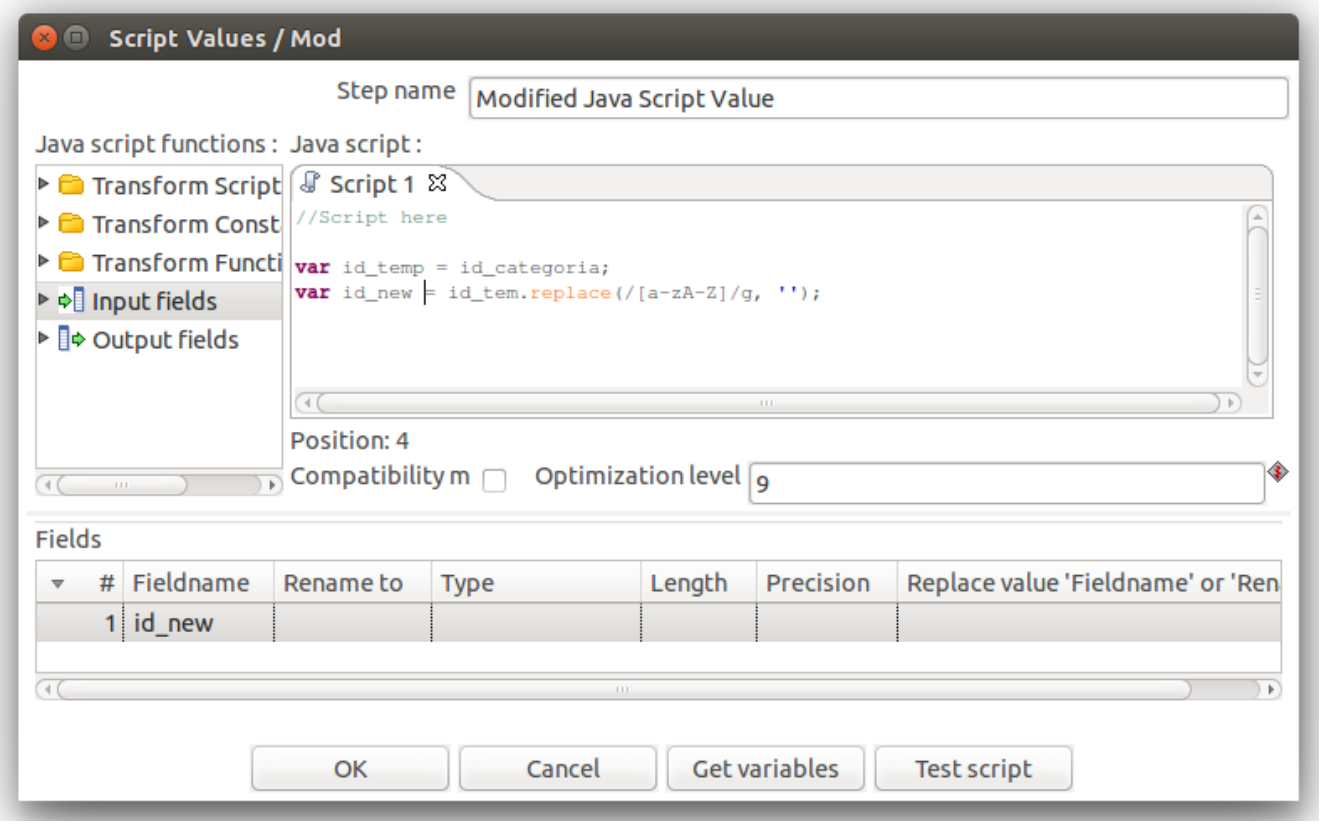


Trasformazione Mod_dati - Step

Modified Java Script Value

- In questo step all'interno del codice javascritp verrà utilizzata un'espressione regolare.

L'obiettivo è quello di rimpiazzare all'interno di un determinato campo tutto i caratteri letterali lasciando solo quelli numerici.



Riepilogo del corso

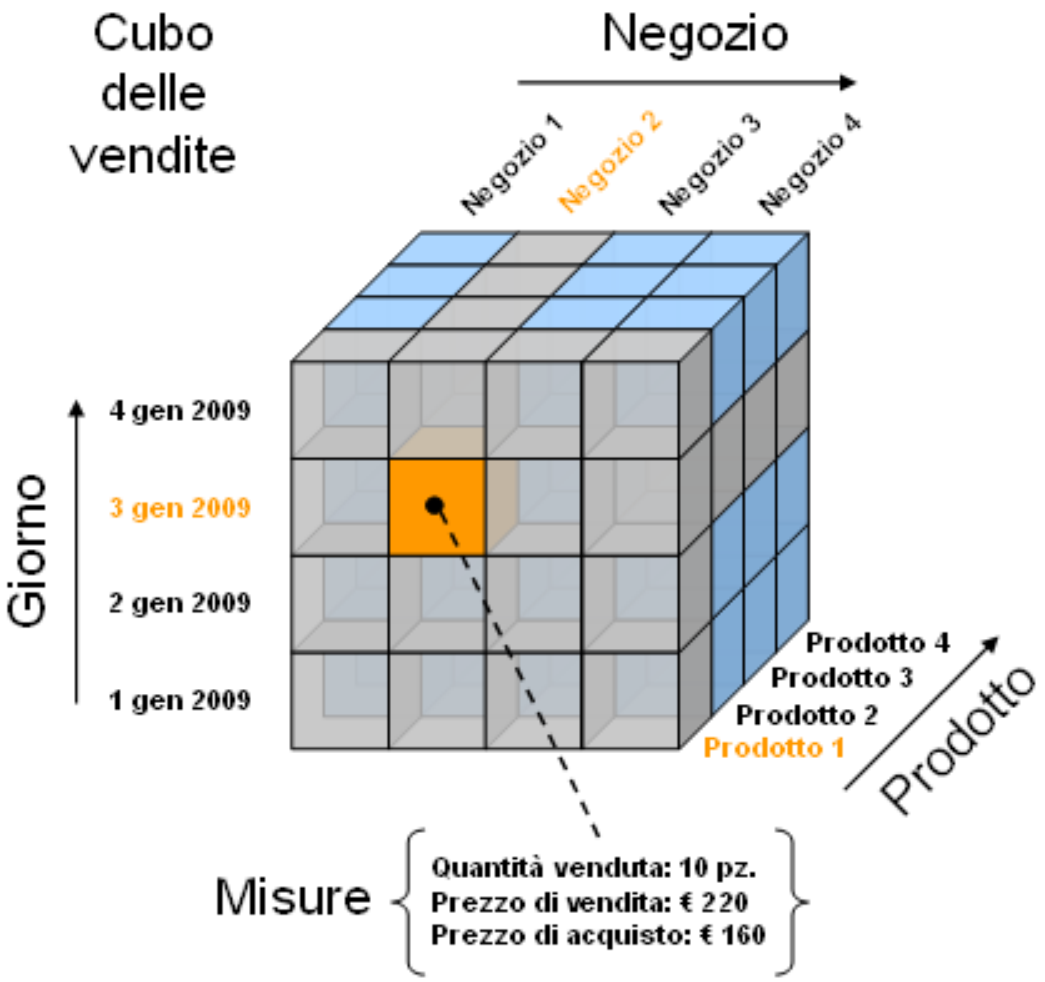
Sistemi OLTP vs Sistemo Olap

- **OLTP** (*On Line Transaction Processing*) sono sistemi orientati alle transazioni e presentano un modello dati fortemente ***normalizzato.***

(+) La normalizzazione favorisce **inserimenti**, **cancellazioni e modifiche dei dati** (attività transazionali).

- I sistemi di tipo **OLAP (***On Line Analytical Processing***)** sono orientati all'analisi**.** Hanno una struttura multidimensionale, chiamata ***Ipercubo*** (spesso semplificata in tre dimensioni).

Ipercubo



Navigazione dei dati più semplice grazie ad operazioni di:

- Drill down
- Drill-up
- Slicing
- Dicing

Come si è arrivati ai Big Data

Tra le principali fonti di dati, che nel tempo hanno contribuito allo sviluppo del fenomeno dei Big Data troviamo:

- **Fonti Operazionali**
- **Sensori, DCS (Distributed Control System) e strumenti scientifici**
- **Dati non-strutturati e semi-strutturati**

Basi di dati Operazionali

Sono quei **dati** che hanno a che fare con **l'attività giornaliera di un'azienda** (industrie, banche o GDO). Alcuni esempi sono:

- *Applicativi di gestione della produzione* (materie prime, consumi…)
- *Applicativi di gestione degli acquisti* (prodotti, ordini, magazzino…)
- *Applicativi di contabilità* (fatture, saldo, movimenti…)
- *Applicativi di gestione del personale* (anagrafica, premi, malattie…)
- *Applicativi di gestione del cliente* (abitudini, marketing mirato…)

In alcuni casi i **dati operazionali arrivano a creare dei volumi rilevanti**. Esempio consideriamo una banca di grandi dimensioni:

Basi di dati Operazionali

- Le basi di dati operazionali in genere fanno riferimento ai **database relazionali** o **RDBMS** (*Relational Data Base Management System*), tra i più famosi ci **sono MySQL**, **IBM DB2**, **Oracle**, **Microsoft SQL Server**.

- **Aumento dei dati** = Gestione e storicizzazione complessa e onerosa in termini di risorse.

- Gli RDBMS mettono a disposizione alcune tecniche di ottimizzazione:
  - **Indicizzazione**
  - **Compressione**
  - **Partizionamento**

Basi di dati Operazionali

- **Indicizzazione**: Utilizzo di Indici (strutture ordinate).
  - **(+)** Recupero rapido di informazioni.
  - **(-)** Scritture lente e aumento dello spazio occupato dal DB.

- **Compressione:** Applicazione di algoritmi di compressione.
  - **(+)** Meno spazio per il salvataggio dei dati.
  - **(-)** Tempo di esecuzione degli algoritmi e decompressione dei dati dopo averli recuperati.

- **Partizionamento:** Suddivisione di una tabella in più parti sulla base di uno specifico criterio.
  - **(+)** Query limitate ad una parte limitata del DB (es. tutti i record da una certa data in poi).
  - **(-)** I vantaggi si perdono se le query impattano più partizioni.

Sensori, DCS e strumenti scientifici

- **Dati** prodotti da sistemi computerizzati utilizzati per il monitoraggio e controllo di impianti industriali.
- Gli impianti generalmente sono costituiti da **numerosi componenti (e sensori) distribuiti**, che inviano i dati ad una postazione centralizzata.
- Le rilevazioni vengono realizzate in intervalli temporali molto piccoli, anche **meno di 1 secondo**.

Dati non-strutturati e semi-strutturati

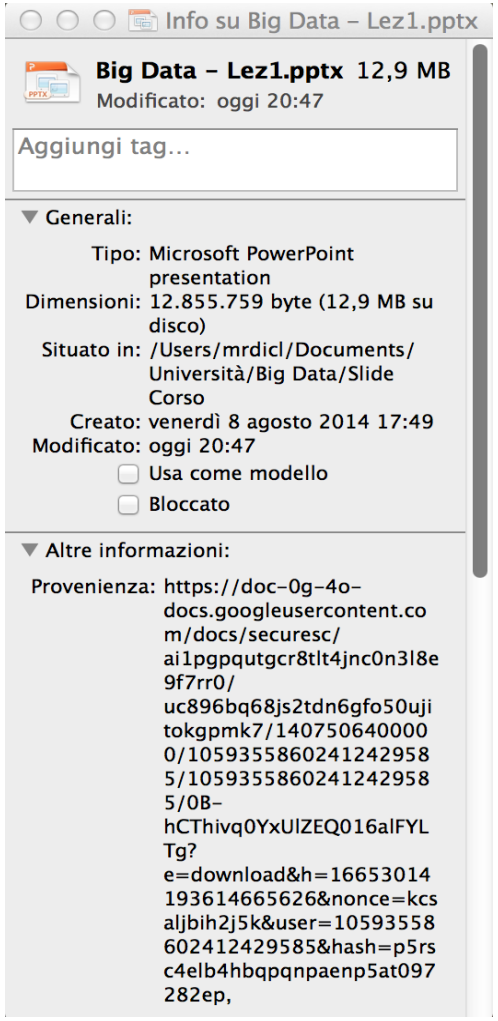
Sono dei **dati che non presentano una struttura predefinita** e che quindi **non** si prestano ad essere gestiti con uno **schema tabellare**. Alcuni esempi presenti in un contesto aziendale sono:

- **Documenti di varia tipologia** (PDF, Word, Excel, PowerPoint etc.)
- **E-mail**
- **Immagini in vari formati** (JPEG, TIFF, GIF, RAW etc.)
- **Strumenti Web 2.0** (Forum e Wiki)

Alcuni di questi documenti in realtà non sono del tutto privi di struttura, si potrebbero definire **semi-strutturati,** per la presenza di informazioni aggiuntive rappresentabili in tabella, **i metadati**.

Dati non-strutturati e semi-strutturati

- **Metadati:** sono dei dati utilizzati per descrivere altri dati.
- Sono **facilmente estraibili** dai documenti che descrivono e messi in tabelle.
- Possono essere utilizzati per operare delle ricerche di e sui documenti.



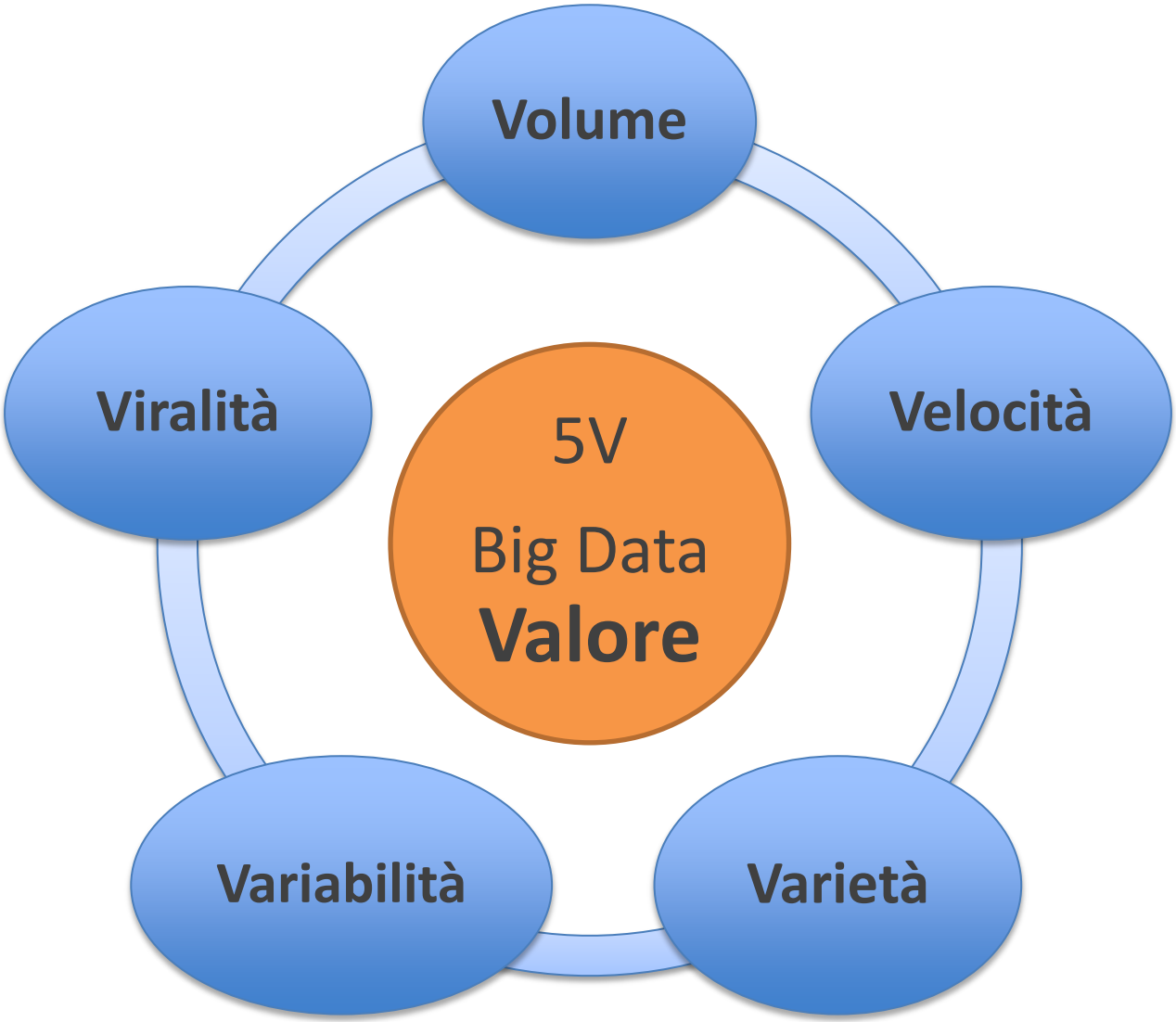
Big Data

I Big Data sono dati che superano i limiti degli strumenti tradizionali.

- *Sono **dati** solitamente disponibili in **grandi volumi**, che si presentano in **differenti formati** (spesso privi di struttura) e con **caratteristiche eterogenee**, prodotti e diffusi generalmente con una **elevata frequenza**, e che **cambiano spesso** nel tempo.*

- Per questo motivo sono identificati con le **5V (+1).**

I Big Data


Volume
Velocità
Varietà
Variabilità
Viralità
5V
Big Data
Valore

Big Data: le 5 V - Volume

Volume: forse la caratteristica più immediata, dal momento che si tratta di dati presenti in **grandi quantità**. In **1 minuto** infatti:

- **100 mila tweet** trasmessi nel mondo.
- **35 mila "Like" FB** a siti ufficiali di organizzazioni.
- **160 milioni (circa) di email** inviate.
- **2 mila check-in su 4square** effettuati.

- Ciò va aggiunto alle restanti **"attività digitali"**, generando una enorme mole di dati e informazioni a loro volta incrociabili.

- Aziende, martketeers, analisti (ma anche la politica) sono le figure più ingolosite dalle potenzialità di tutto ciò.

Big Data: le 5 V - Volume

- Alcune tipologie di Big Data sono **transitorie**:
  - Dati generati da sensori.
  - Log dei web server.
  - Documenti e pagine web.

- Il **primo passo** quando si opera con i Big Data é allora l'**immagazzinamento**. L'**analisi (e la pulizia)** avvengono in una fase successiva (per evitare di perdere potenziali informazioni).

- Ciò richiede **importanti investimenti in termini di storage** e di **capacità di calcolo** adatta all'analisi di grandi moli di dati.

- Tecnologia open source più diffusa e utilizzata: **Apache Hadoop**.

Big Data: le 5 V - Velocità

Velocità: è una caratteristica che ha più di un significato.

- Si riferisce in primis alla **elevata frequenza con cui i dati vengono generati** – si ripercuote sulla quantità (Volume).
- Il secondo aspetto riguarda la **velocità con cui le nuove tecnologie** permettono di **accedere e di analizzare questi dati**.

Maggiore è la velocità di accesso ai dati
Maggiore sarà la velocità in un processo decisionale
Maggiore/migliore competitività sui diversi panorami del mercato

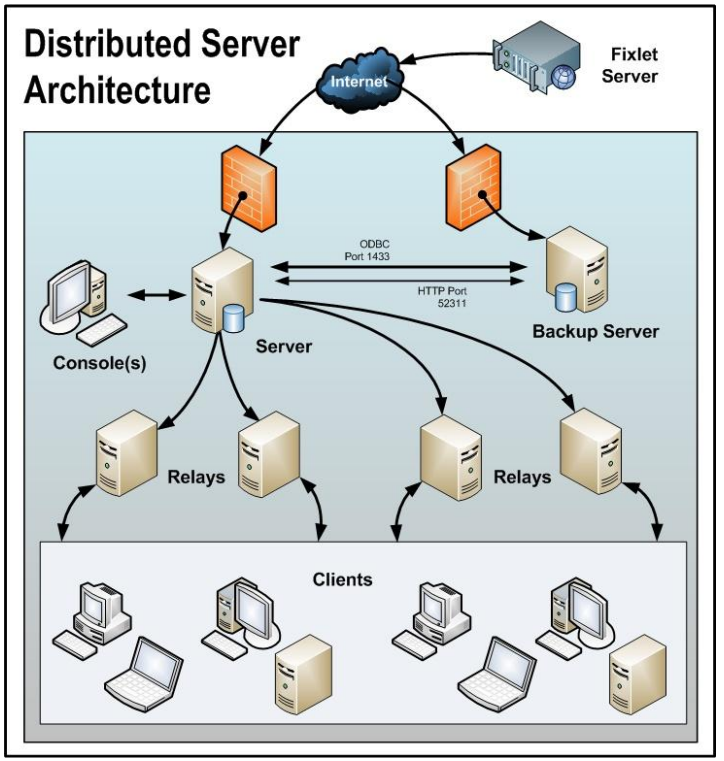
Quali tecnologie?!

Big Data: le 5 V - Velocità

Velocità: è una caratteristica che ha più di un significato.

- Particolarmente adatte sono le **architetture distribuite.**

- **Gestione** di **strutture dati anche complesse**.
- **Accesso ai dati in tempo reale**.
- **Velocità di elaborazione** grazie a tecniche di calcolo distribuito.
- Database non relazionali come i ***column DB*** e ***key/value DB*** (NoSQL).



Big Data: le 5 V - Varietà

Varietà: caratteristica che ha a che fare con la forma in cui i dati si presentano.

- Nel contesto **Big Data** le **informazioni** da trattare sono dati non-strutturati (o semi-strutturati). Non adatti ad essere lavorati con le tecniche tradizionali dei database relazionali.

- Dati come **email**, **immagini**, **video**, **audio**, **stringhe di testo** a cui dare un significato non si possono memorizzare in una tabella.

- Per la gestione e il salvataggio di questi dati si ricorre spesso ai ***database NoSQL***. Non impongono uno schema rigido per organizzare i dati (*schemaless database*).

Big Data: le 5 V - Variabilità

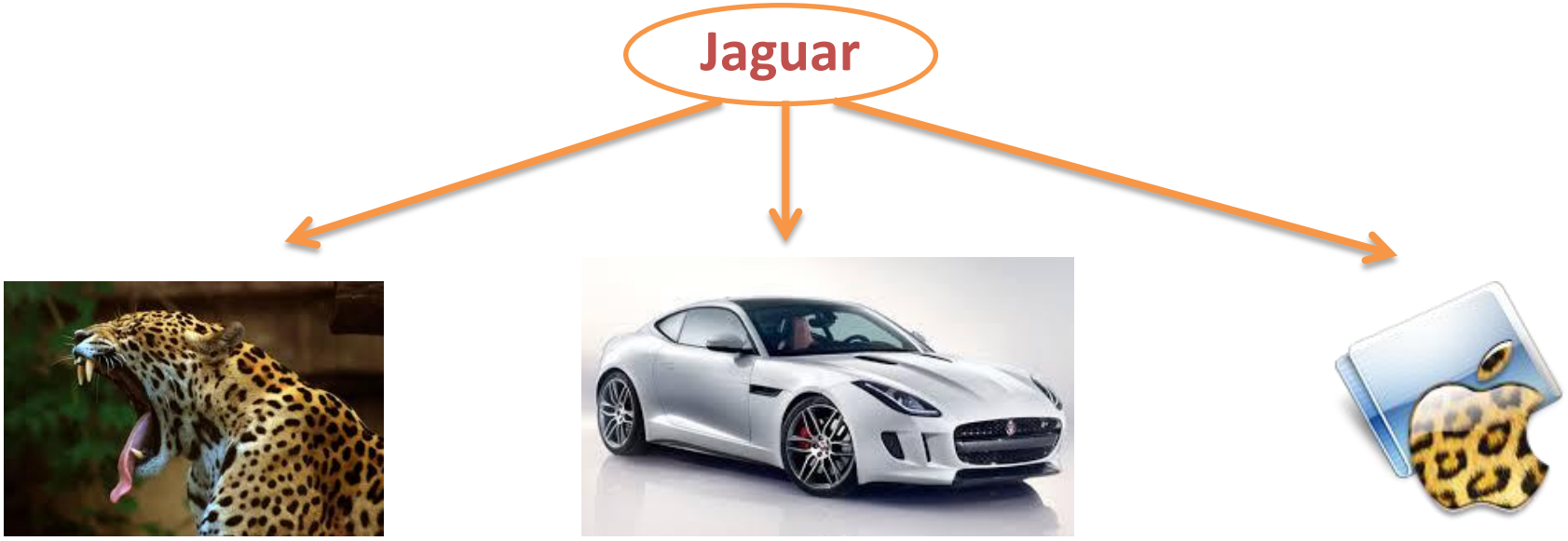
Variabilità: caratteristica relativa alla **contestualizzazione** di un dato.

- Il significato o l'**interpretazione di uno stesso dato** può **variare** in base al contesto in cui esso viene raccolto e analizzato.
- Esempio la frase "*leggete il libro*", essa avrà un **significato positivo** in un blog che parla di letteratura, mentre avrà una **connotazione negativa** in un blog per appassionati di cinema.
- Il **significato** di un dato **può essere differente** anche in base al **momento in cui viene fatta l'analisi**, spesso è fondamentale l'analisi in tempo reale (Velocità).

Big Data: le 5 V - Variabilità

Variabilità: caratteristica relativa alla **contestualizzazione** di un dato.

E' importante trovare dei meccanismi che riescano a dare una **semantica ai dati** in base al contesto in cui sono espressi.



Big Data: le 5 V - Viralità

Viralità: caratteristica che ha a che fare su **quanto e come i dati si diffondono (**Propagazione dei dati**)**.

- La **grande quantità** di dati (spesso correlati tra loro) e **l'alta velocità con cui sono prodotti** implica una **diffusione virale** delle informazioni.

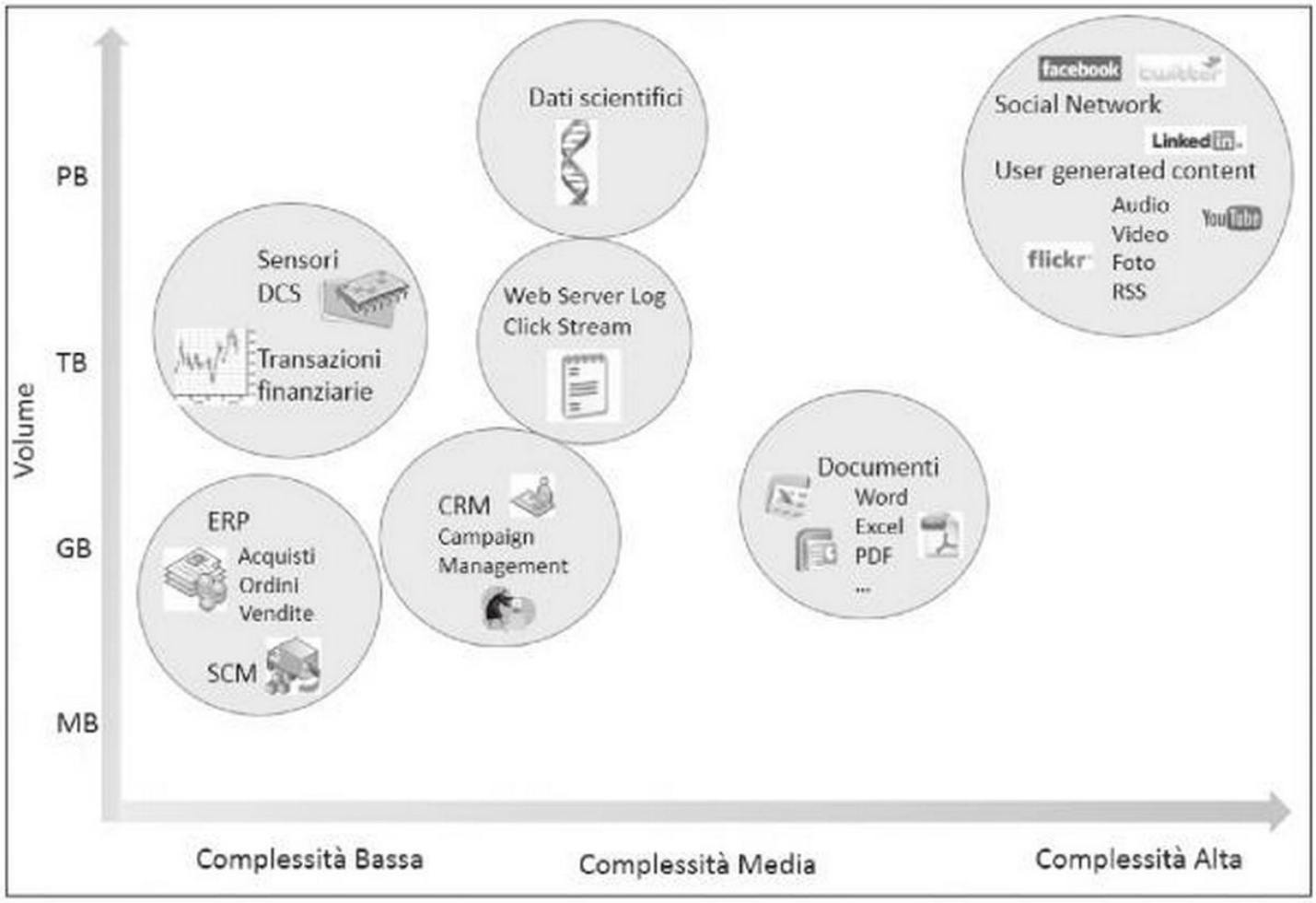
- **Esempio**: **una notizia o un evento** diffusi tra diversi canali. **Diffusione amplificata** con i collegamenti nei vari **social network**.

Big Data: le 5 V - Viralità

Virale è anche la **crescita del Volume** dei dati generati dalle attività digitali dell'uomo (user-generated content):

- Nel **2010** è stata stimata una produzione di **1,2 zettabyte** di dati (**1ZB corrisponde a mille miliardi di GB**).
- Nel **2011** è cresciuta a **1,8ZB**.
- Nel **2013** si è arrivati a **2,7ZB**.
- La proiezione per il **2015** parla di **8ZB**.

Big Data: le 5 V



Classificazione dei dati per volume e complessità

Big Data: le 5 V - Valore

Valore: è necessario comprendere e gestire in modo adeguato i dati e tutti questi aspetti ad essi legati in modo da riuscire ad **estrarre il potenziale informativo**.

- I Big Data **nascondono un grande valore**. Al primo utilizzo di solito se ne estrae soltanto una parte, il *valore rimanente rimane "dormiente"* fino ad un successivo utilizzo.

- E' quindi importante adottare metodologie e tecnologie che permettano la **continua integrazione di nuove informazioni**, in seguito ad un utilizzo reiterato, con l'obiettivo di **costruire una base di conoscenza sempre più ampia**.

Problematiche

- **Elevato numero di campi applicativi** diversi tra loro.
- I **differenti canali** attraverso i quali i dati vengono raccolti.
- Identificare una possibile **architettura adattabile** a tutte le aree.
- Come è possibile scoprire il "**Valore**" dei Big Data?

- **Utilizzo di complesse analisi e processi di modellazione.**
- **Formulazione di ipotesi -> implementazione di modelli semantici, visuali e statistici -> validazione.**

Teorema di Brewer (o Teorema **CAP**)

- Il **Teorema CAP** (Consistency – Availability – Partition tolerance) è fondamentale per capire il comportamento di **sistemi SW distribuiti**, e progettarne l'architettura in modo da rispettare requisiti non funzionali stringenti, tra cui:
  - ***Elevate prestazioni.***
  - ***Continua disponibilità.***
  - ***Sistemi geograficamente distribuiti.***
- Il **Web 2.0**, è popolato da applicazioni che lavorano su bilioni e trilioni di dati ogni giorno . La **scalabilità** è un concetto chiave.
- A tal proposito si stanno sviluppando **database che sono distribuiti** sulla rete per realizzare una ***scalabilità orizzontale***.

Teorema di Brewer (o Teorema **CAP**)

***Il Teorema CAP** afferma*

"sebbene sia altamente desiderabile per un sistema software distribuito fornire simultaneamente **totale coerenza** (*Consistency*), **continua disponibilità** (*Availability*) e **tolleranza alle partizioni** (*Partition tolerance*), ciò **non è possibile.**

E' necessario stabilire, di volta in volta in funzione dei requisiti di una specifica applicazione, quali di queste tre garanzie sacrificare.

- E' importante tenere a mente questo teorema, perchè specie in applicazioni Web2.0, fornire agli utenti una pessima esperienza può avere una diffusione virale a causa dei vari social network (fonti Amazon e Google).

Teorema di Brewer (o Teorema **CAP**)

Consistency *(Totale coerenza)*
Un sistema distribuito è **completamente coerente** se preso un dato che viene scritto su un **nodoA** e viene letto da un altro **nodoB,** il sistema ritornerà l'ultimo valore scritto (quello *consistente*).

- Se si considera la cache di un singolo nodo la **totale consistenza è garantita**, così come la tolleranza alle partizioni.
- **Non** si hanno però **sufficiente disponibilità** (fault-tolerance) e **buone performance**.

- Se la **cache è distribuita** su due o più nodi, aumenta la disponibilità, ma vanno **previsti dei meccanismi complessi** che permettano ad ogni nodo di accedere ad un repository virtuale distribuito (e leggere lo stesso valore di dato).

Teorema di Brewer (o Teorema **CAP**)

Availability *(disponibilità)*
Un sistema (distribuito) è **continuamente disponibile** se ogni nodo **è sempre in grado di rispondere ad una query o erogare i propri servizi** a meno che non sia *indisponibile.*

- Banalmente **un singolo nodo non garantisce la continua disponibilità**.

- Una **cache distribuita** mantiene nei vari nodi delle aree di backup in cui sono memorizzati i dati presenti su altri nodi.

- Per realizzare la **continua disponibilità** si ricorre alla **ridondanza dei dati (su più nodi)**. Ciò però richiede meccanismi per garantire la consistenza e problematiche riguardo la tolleranza alle partizioni.

Teorema di Brewer (o Teorema **CAP**)

Partition-Tolerance *(Tolleranza alle partizioni)*
È la capacità di un sistema di essere tollerante ad una aggiunta o una rimozione di un nodo nel sistema distribuito (*partizionamento*) o alla perdita di messaggi sulla rete.

1. Si consideri una **configurazione** in cui **un solo cluster** è composto da **nodi su due diversi data center**.
2. Supponiamo che i **data center perdano la connettività di rete**. I **nodi** del cluster **non riescono più a sincronizzare** lo stato del sistema.
3. I **nodi si riorganizzano in sotto-cluster**, tagliando fuori quelli dell'altro data center.

Il sistema continuerà a funzionare, in modo non coordinato e con possibile perdita di dati (es. assegnazione della stessa prenotazione a clienti diversi).

Teorema di Brewer (o Teorema **CAP**)

Poiché non è possibile garantire simultaneamente **completa consistenza**, **continua disponibilità** e **tolleranza alle partizioni,** quando si progetta un sistema distribuito è necessario valutare attentamente quale soluzione di compromesso accettare tra le seguenti coppie possibili **CA**, **CP** e **AP**

Consistency/Availability (CA)

- E' il compromesso offerto solitamente dai **RDBMS.**
- **I dati sono coerenti** su tutti i nodi (attivi e disponibili).
- **Scritture/letture sempre possibili**, e dati aggiornati propagati tra i nodi del cluster (**dati sempre aggiornati**).

(-) Possibili problemi legati alle performance e alla scalabilità.
(-) Possibile disallineamento tra i dati nel caso di partizioni di nodi.

Teorema di Brewer (o Teorema **CAP**)

Poiché non è possibile garantire simultaneamente **completa consistenza**, **continua disponibilità** e **tolleranza alle partizioni,** quando si progetta un sistema distribuito è necessario valutare attentamente quale soluzione di compromesso accettare tra le seguenti coppie possibili **CA**, **CP** e **AP**

Consistency/Partition-Tolerance (CP)

- Compromesso preferito da soluzioni come **Hbase, MongoDB, BigTable.**
- **I dati sono coerenti** su tutti i nodi e sono garantite le partizioni, assicurando la sincronizzazione dei dati.

(-) Possibili problemi di disponibilità, dati non più disponibili se un nodo va giù.

Teorema di Brewer (o Teorema **CAP**)

Poiché non è possibile garantire simultaneamente **completa consistenza**, **continua disponibilità** e **tolleranza alle partizioni,** quando si progetta un sistema distribuito è necessario valutare attentamente quale soluzione di compromesso accettare tra le seguenti coppie possibili **CA**, **CP** e **AP**

Availability/Partition-Tolerance (AP)

- Compromesso usato da soluzioni come **CouchDB, Riak, Apache Cassandra.**
- **I nodi restano online** anche se impossibilitati a parlarsi.
- E' necessario un processo di risincronizzazione dei dati per eliminare eventuali conflitti quando la partizione è risolta.

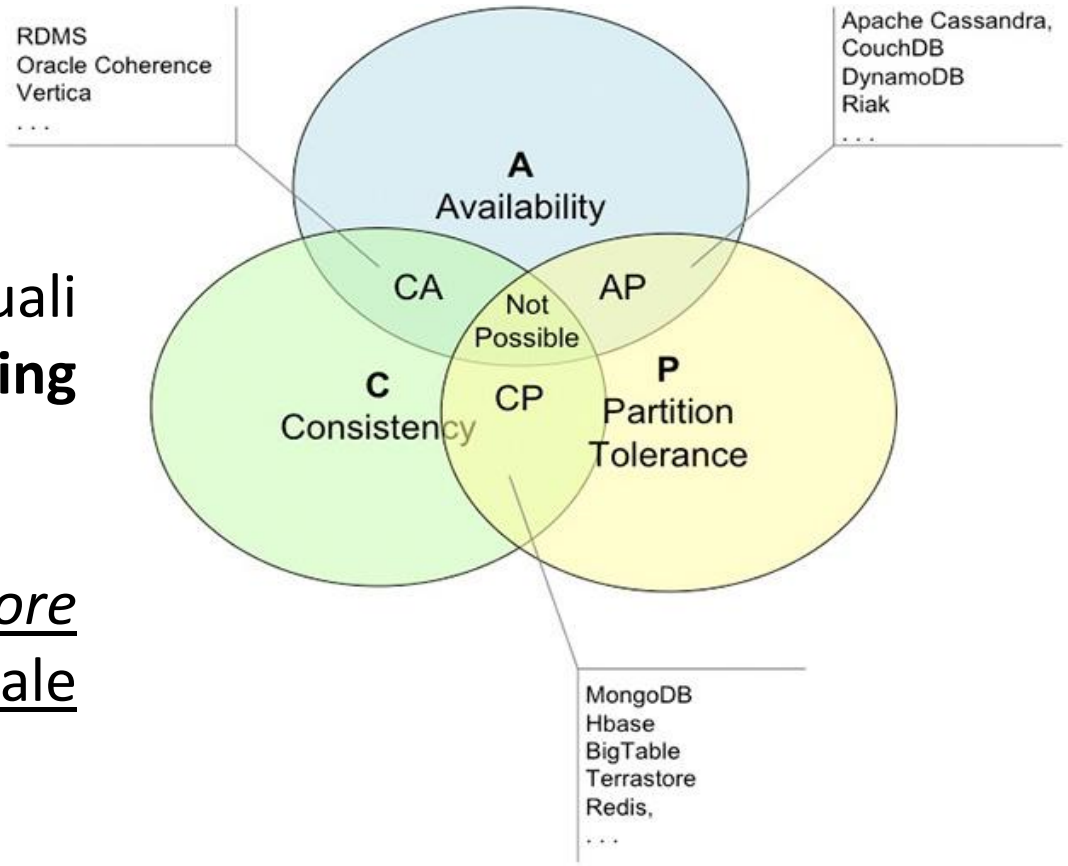
(+) Buone prestazioni in termini di latenza e scalabilità.

Teorema di Brewer (o Teorema **CAP**)

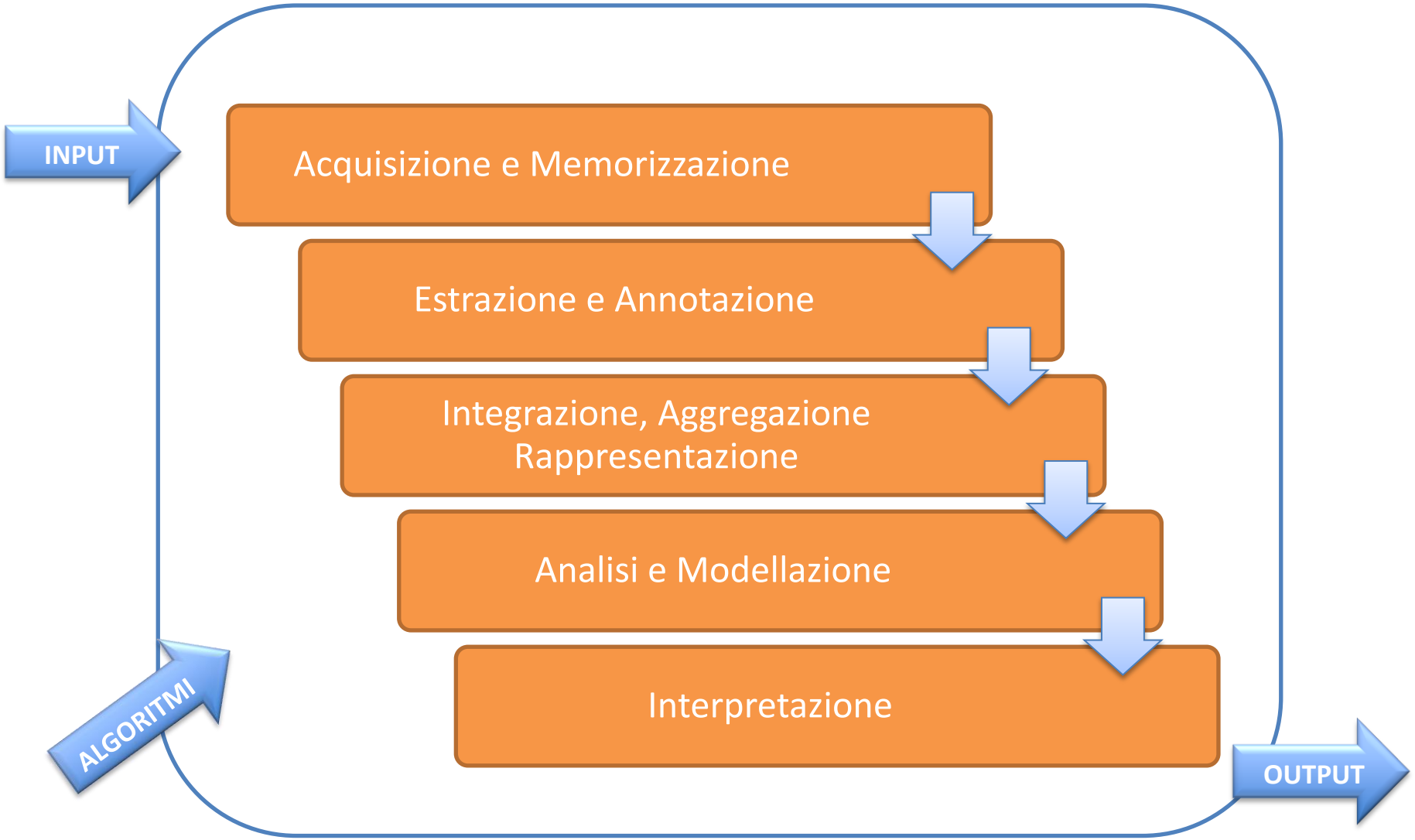
Osservazione

La maggior parte delle attuali soluzioni prevedono il **tuning** della modalità operativa.

<u>Cioè lasciano allo *sviluppatore* la possibilità di scegliere quale *garanzia sacrificare*.</u>



Pipeline di Analisi dei Big Data



Pipeline: Acquisizione dei Dati e Memorizzazione

- Grandi quantità di dati possono essere **filtrati e compressi** a diversi ordini di grandezza.
  - Sfida: Definire dei filtri opportuni in modo che non vadano perse informazioni di interesse.

- **Dettagli** inerenti a **condizioni sperimentali e procedure** possono essere richiesti per interpretare i risultati correttamente.
  - Sfida: Generazione automatica dei metadata corretti.

- Possibilità di ricerca sia all'interno dei metadata che nei dati di sistema.
  - Sfida: Creare e utilizzare delle strutture dati ottimizzate che consentano le ricerche in tempi accettabili.

Pipeline: Estrazione delle Informazioni e Pulizia

- Le **informazioni** raccolte spesso **non** sono in un **formato pronto per l'analisi** (es. immagini di sorveglianza VS immagini scattate da fotografi)
  Sfida: <u>Realizzare un processo di estrazione delle informazioni che le fornisca in un formato adatto alla fase di analisi.</u>

- I Big Data sono **incompleti** a causa di **errori** commessi durante la fase di acquisizione.
  Sfida: <u>Definire dei vincoli e modelli per la gestione e correzione automatica di errori in diversi domini Big Data.</u>

Pipeline: Integrazione, Aggregazione e Rappresentazione dei Dati

- I **Dati sono eterogenei** e può non essere abbastanza raccoglierli all'interno di repository.
  Sfida: Creare delle strutture dati di memorizzazione che siano in grado di adattarsi alle differenze nei dettagli sperimentali.

- I **modi di memorizzare dati** sono **diversi**, alcuni modelli hanno dei vantaggi rispetto ad altri per determinati scopi.
  Sfida: Creare dei tool di supporto al processo di progettazione dei database e alle tecniche di sviluppo tenendo conto del contesto applicativo e d'uso dei dati.

Pipeline: Query, Modellazione Dati e Analisi

- I **metodi per investigare e interrogare** i Big Data sono **differenti** dalle tradizionali analisi statistiche.

  Sfida: Creare delle tecniche per l'elaborazione di query complesse e scalabili (sull'ordine dei TeraByte), considerando delle risposte interattive nel tempo.

- **Big Data interconnessi** formano delle **reti di dati eterogenee**, in cui la **ridondanza dei dati** può essere sfruttata per compensare l'assenza di alcune informazioni, per verificare situazioni di conflitto e evitare che ci siano relazioni nascoste.

  Sfida: Rendere coordinati i sistemi DB e le interrogazioni SQL, con i tool di analisi che realizzano diverse forme di elaborazione non-SQL (data mining, analisi statistica).

Nuovi Concetti

Datification

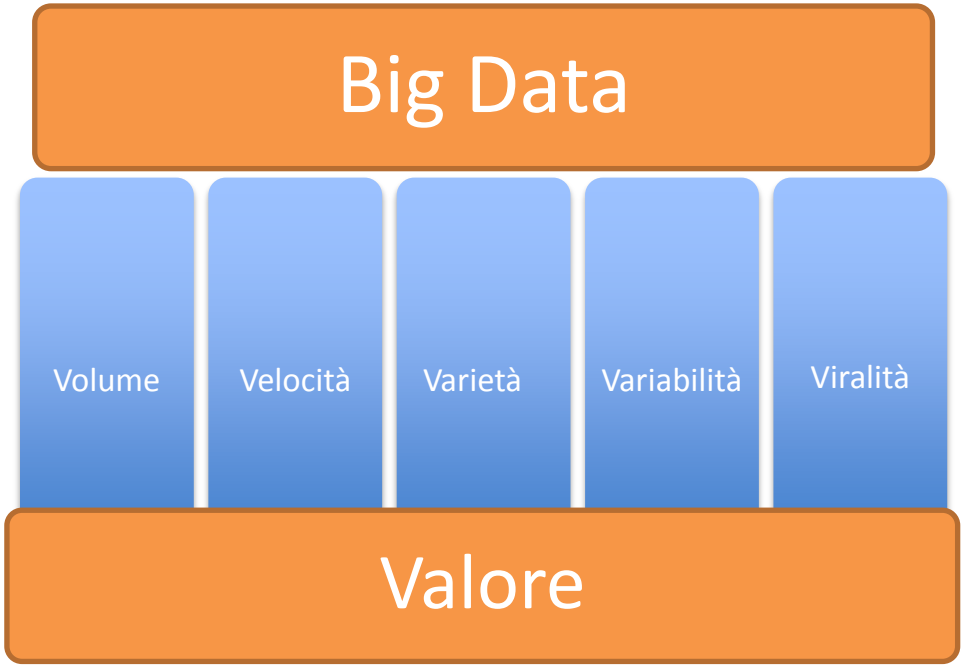
- Prendere **informazioni su qualsiasi cosa** e **trasformarle in un qualsiasi formato dati** in modo da renderle **quantificabili.**

- **Utilizzare queste informazioni** in un nuovo modo con l'obiettivo di **tirar fuori il loro valore implicito e nascosto**.

- *Quando i dati sono pochi è desiderabile che siano accurati* (campionamento random). I Big Data hanno cambiato il concetto di aspettativa della precisione: *Trattare queste grandi quantità di dati spesso imprecise e imperfette permette di fare delle previsioni superiori* (Analisi Predittiva).

Campi di Applicazione

- Il Problema dei Big Data si riferisce alla combinazione di un **grande volume di dati** che deve essere **trattato in tempi abbastanza rapidi**.



- Sono molte aree applicative in cui i Big Data sono attualmente utilizzati con risultati interessanti ed **eccellenti prospettive future** per affrontare le **principali sfide** come **Analisi dei Dati**, **Modellazione**, **Organizzazione** e **Ricerca** (Data Retrieval).

Campi di Applicazione

Investimenti crescenti nei Big Data possono portare fondazioni, enti e organizzazioni di nuova generazione a interessanti **scoperte** *in campo scientifico, nella medicina*, **vantaggi e guadagni** nel *settore ICT e in contesti Business,* **nuovi servizi e opportunità** per *cittadini digitali e utenti web*.

- **Sanità e Medicina**
- **Ricerca Scientifica (Analisi dei Dati)**
- **Istruzione**
- **Settore Energetico e dei Trasporti**
- **Social Network – Servizi Internet – Web Data**
- **Finanza/Business – Marketing**
- **Sicurezza**

Criticità e rischi dei Big Data

Come ogni "nuova tecnologia" **i Big Data offrono grandi prospettive e potenzialità**, ma non presentano esclusivamente caratteristiche positive. Vi sono alcuni **aspetti critici** che è bene prendere in considerazione:

- Problematiche legate alla **qualità e all'affidabilità dei dati.**
- Problematiche relative alla **privacy e alla proprietà dei dati.**

Criticità e rischi dei Big Data – Qualità dei dati

La **qualità dei dati** è determinata da un insieme di caratteristiche:

- **Completezza:** la presenza di **tutte le informazioni necessarie** a descrivere un oggetto, entità o evento (es. anagrafica).
- **Consistenza:** i dati **non devono essere in contraddizione**. Ad esempio il saldo totale e movimenti, disponibilità di un prodotto richiesto da soggetti differenti, etc.
- **Accuratezza:** i dati devono essere corretti, cioè **conformi a dei valori reali**. Ad esempio un indirizzo mail non deve essere solo ben formattato ***nome@dominio.it***, ma deve essere anche valido e funzionante.

Criticità e rischi dei Big Data – Qualità dei dati

La **qualità dei dati** è determinata da un insieme di caratteristiche:

- **Assenza di duplicazione:** Tabelle, record, campi dovrebbero essere memorizzati **una sola volta**, evitando la presenza di copie. Le informazioni duplicate comportano una doppia manutenzione e possono portare problemi di sincronia (consistenza).

- **Integrità:** è un concetto legato ai database relazionali, in cui sono presenti degli strumenti che permettono di implementare dei **vicoli di integrità**. Esempio un **controllo sui tipi di dato** (presente in una colonna), o sulle chiavi identificative (impedire la presenza di due righe uguali).

Criticità e rischi dei Big Data – Qualità dei dati

Nei contesti applicativi che coinvolgono l'uso di database tradizionali, la **qualità complessiva dei dati** può essere minata da:

- **Errori nelle operazioni di data entry** (campi e informazioni mancanti, errati o malformati).
- **Errori nei software di gestione dei dati** (query e procedure errate).
- **Errori nella progettazione delle basi di dati** (errori logici e concettuali).

Criticità e rischi dei Big Data – Qualità dei dati

Nel mondo Big Data invece:

- **Dati operazionali**: i problemi relativi alla qualità sono conosciuti e esistono diversi strumenti per realizzare in modo automatico la pulizia dei dati.

- **Dati generati automaticamente:** i dati scientifici o provenienti da sensori sono privi di errori di immissione. Spesso però sono "deboli" a livello di contenuto informativo, c'è la necessità di integrarli con dati provenienti da altri sistemi per poi analizzarli.

- **Dati del Web:** Social network, forum, blog generano dati semistrutturati. La parte più affidabile sono i metadati (se presenti), il testo invece è soggetto a errori, abbreviazioni, etc

Criticità e rischi dei Big Data – Qualità dei dati

Nel mondo Big Data invece:

- **Disambiguare le informazioni:** Uno stesso dato può avere significati differenti (es. calcio). La sfida è cerca di trovare quello più attinente al contesto in esame. Un aiuto sono i ***tag***, etichettando i dati si cerca di evidenziare l'ambito di pertinenza.

- **Veridicità:** Notizie, affermazioni, documenti non sempre veri o corrispondenti alla realtà.

OSS. *La qualità dei dati è però legata anche al contesto in cui essi sono analizzati. Operazioni di filtraggio e pulizia devono essere fatte procedendo per gradi per evitare di eliminare dati potenzialmente utili.*

Criticità e rischi dei Big Data – Privacy

Il tema Big Data si apre a problemi di Privacy, proprietà e utilizzo dei dati da parte di terzi.

- **Dati del Web:** gli *user-generated-content* sono condivisi accessibili a tutti. E' etico il loro utilizzo?
- **Dati sensibili:** i dati presenti nei DB degli ospedali relativi alla storia clinica dei pazienti sono opportunamente protetti?
- **Dati di posizione:** l'uso di smartphone, GPS, sistemi di pagamento elettronico, ma anche social network lasciano delle tracce da cui è possibile ricavare gli spostamenti degli utenti.

Panoramica sulle tecnologie Big Data

Considerando le tecnologie e le soluzioni adatte a lavorare con i **Big Data** bisogna prestare attenzione a **4 classi di aspetti fondamentali**:

- **Aspetti di *Data Management***
- **Aspetti *Architetturali***
- **Aspetti di *Accesso e Rappresentazione dei Dati***
- **Aspetti di *Ingestion, Mining e Data Analysis***

Aspetti fondamentali

1. **Aspetti di Data Management**
   Le principali soluzioni attualmente disponibili si stanno muovendo nella direzione di essere in grado di gestire adeguatamente **quantità di dati crescenti** (più o meno rapidamente) nel tempo.

2. **Aspetti Architetturali**
   Nell'elaborazione di un insieme molto grande di dati è importante **ottimizzare il carico di lavoro**, ad esempio adottando **un'architettura parallela (distribuita),** o fornendo una **allocazione dinamica delle risorse di calcolo**.

Aspetti fondamentali

3. **Aspetti di Accesso ai Dati e di Rappresentazione**
   E' opportuno avere dei tool che consentano una **visualizzazione scalabile dei risultati** ottenuti dall'analisi dei Big Data.

4. **Aspetti di Ingestion, Mining e Data Analysis**
   L'immagazzinamento, l'Analisi statistica, le Facet Query, la risposta alle query in modo rapido, con **dati aggiornati e pertinenti** alle ricerche effettuate, **coerenza e consistenza** sono importanti caratteristiche che un'architettura ideale in questo contesto deve sostenere o garantire.

NoSQL Definizione

“Not Only SQL” – Identifica un sottoinsieme di strutture sw di memorizzazione, **progettate per ottimizzare e migliorare le performance delle principali operazioni su dataset di grandi dimensioni.**

Perché NoSQL?

- ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability), sono le proprietà logiche che devono soddisfare le transazioni nei DB tradizionali. *Questo paradigma però non è dotato di una buona scalabilità*.
- Le Applicazioni Web hanno esigenze diverse: **alta disponibilità**, **scalabilità** ed **elasticità**, cioè bassa latenza, schemi flessibili, distribuzione geografica (a costi contenuti).
- I DB di nuova generazione (NoSQL) sono maggiormente adatti a soddisfare questi bisogni essendo **non-relazionali, distribuiti, open source e scalabili orizzontalmente**.

Paradigma ACID

Atomicità

- Una transazione **è indivisibile nella sua esecuzione** e la sua **esecuzione deve essere o totale o nulla**, non sono ammesse esecuzioni parziali.

Consistenza

- Quando inizia una transazione il **database si trova in uno stato coerente** e quando la transazione termina il **database deve essere in un altro stato coerente**. Cioè non devono essere violati eventuali vincoli di integrità, quindi **non devono verificarsi contraddizioni (*inconsistenza*) tra i dati archiviati nel DB.**

Paradigma ACID

Isolamento

- Ogni transazione deve essere **eseguita in modo isolato e indipendente dalle altre transazioni**, l'eventuale fallimento di una transazione non deve interferire con le altre transazioni in esecuzione.

Durabilità (*o Persistenza*)

- Quando una transazione richiede un *commit work*, **le modifiche apportate non dovranno più essere perse**. **Per evitare** che nel lasso di tempo fra il momento in cui il DB si impegna a scrivere le modifiche e quello in cui le scrive effettivamente si verifichino **perdite di dati dovuti a malfunzionamenti, vengono tenuti dei registri di log dove sono annotate tutte le operazioni sul DB**.

NoSQL – Pro & Cons

PRO

- *Schema free*
- *Alta Disponibilità*
- *Scalabilità*
- *Supporto ad una facile replicazione*
- *API semplici*
- *Eventually Consistent / **BASE** (no ACID)*

CONTRO

- *Limitate Funzionalità di query*
- *Difficoltà di spostamento dei dati da un NoSQL DB ad un altro sistema (ma il 50% sono JSON-oriented);*
- *Assenza di modalità standard per accedere ad un archivio dati NoSQL*

Eventually Consistent

- **Modello di consistenza** utilizzato nei sistemi di calcolo distribuito.

- Il sistema di storage **garantisce che se un oggetto non subisce nuovi aggiornamenti**, alla fine (quando la finestra di inconsistenza si chiude) **tutti gli accessi restituiranno l'ultimo valore aggiornato.**

- Nasce con l'**obiettivo** di **favorire** le ***performance***, ***la scalabilità*** e la reattività nel servire un ***elevato numero di richieste*** (rischio minimo di letture di dati non aggiornati).

- Approccio **BASE** (**B**asically **A**vailable, **S**oft state, **E**ventual consistency).

- Conosciuta con il nome di **"Replicazione Ottimistica".**

Approccio BASE

- Le proprietà **BASE** sono state introdotte *Eric Brewer* (Teorema CAP).

- Queste proprietà rinunciano alla **consistenza** per garantire una maggiore **scalabilità** e **disponibilità** delle informazioni.

- **B**asically **A**vailable: Il sistema deve garantire la disponibilità delle informazioni.

- **S**oft State: Il sistema può cambiare lo stato nel tempo anche se non sono effettuate scritture o letture.

- **E**ventual consistency: Il sistema può diventare consistente nel tempo (anche senza scritture) grazie a dei sistemi di recupero ella consistenza.

NoSQL DB - Dettagli

Caratteristiche chiave dei Database NoSQL sono sicuramente:

- **Ambiente Multi-Nodo**

Solitamente si crea un modello composto da un insieme di **più nodi distribuiti** (***cluster***), che possono essere aggiunti o rimossi.

- **Sharding dei dati**

Mediante opportuni algoritmi i **dati vengono suddivisi e distribuiti su più nodi**, recuperandoli quando necessario (es. *algoritmo di gossip*).

- **Replica delle informazioni**

I **dati** distribuiti sui vari nodi, spesso sono anche **copiati un certo numero di volte** per *garantire la disponibilità delle informazioni*.

NoSQL DB - Dettagli

I *NoSQL DB* sono oggetto di **continui studi** al fine di **migliorare**:

- **Performance**

Vengono studiati e implementati dei ***nuovi algoritmi*** al fine di **aumentare le prestazioni complessivi dei sistemi NoSQL**.

- **Scalabilità orizzontale**

E' fondamentale riuscire ad **aumentare/diminuire la dimensione di un cluster aggiungendo/rimuovendo nodi in modo "invisibile"**; cioè l'intero *sistema non deve fermarsi* o accorgersi di cosa sta accadendo.

- **Performance sulla singola macchina**

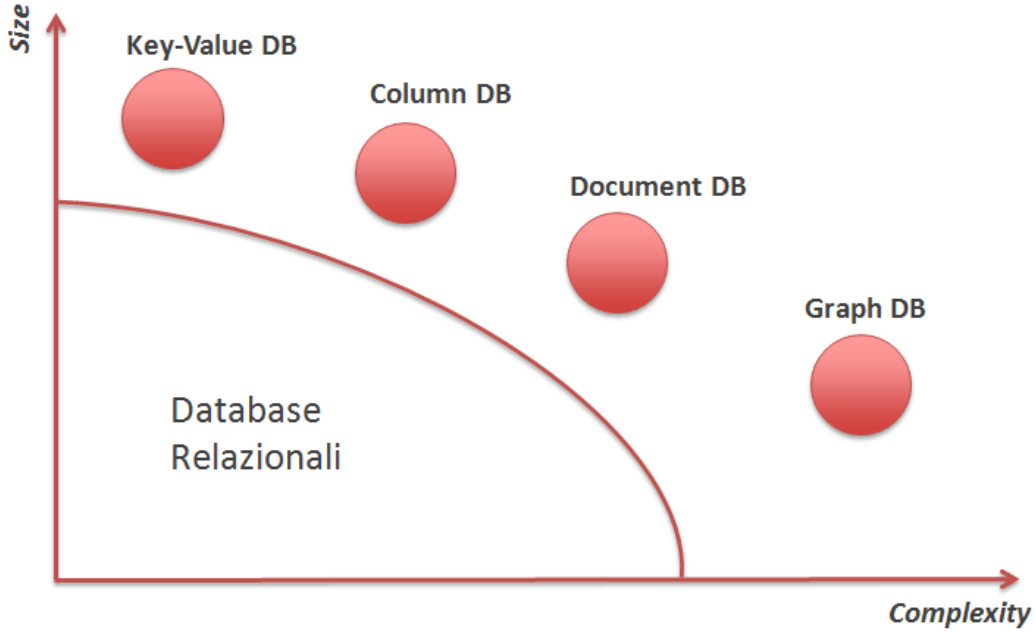
E' importante riuscire ad **aumentare anche le prestazioni di ciascuna macchina**, poiché spesso sono responsabili di ***ricercare informazioni nel cluster*** a nome di applicativi esterni.

Tipologie di Database NoSQL

- **Key-Value DB**
- **Col-Family/Big Table DB**
- **Document DB**
- **Graph Database**
- **XML DB**
- Object DB
- Multivalue DB
- ACID NoSQL

Comparazione

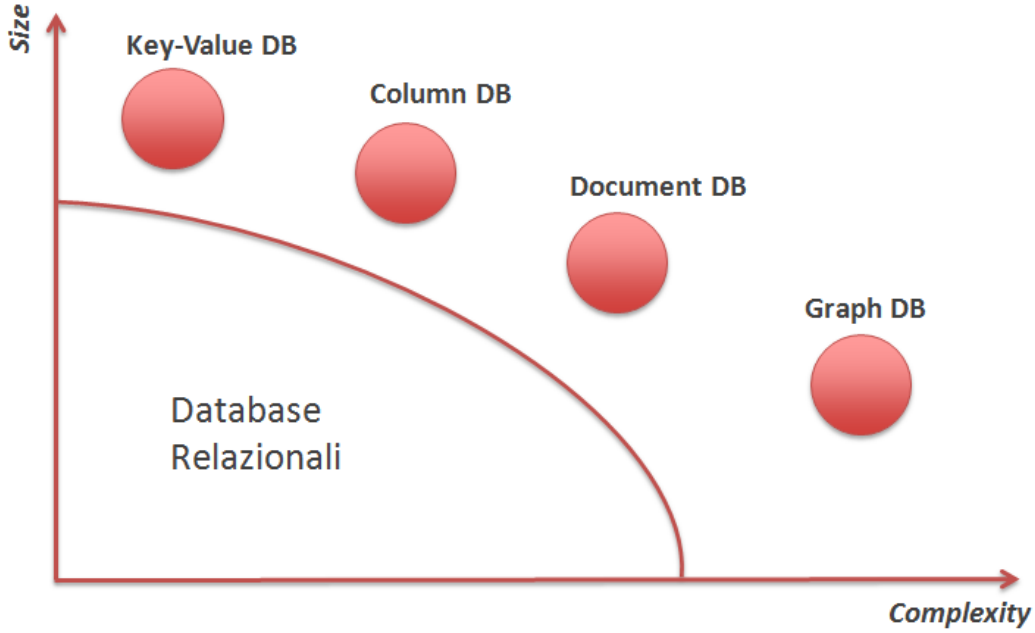
Valutazione delle soluzioni NoSQL DB sulla base dei parametri ***size*** e ***operational complexity***.



- I **Graph DB** hanno **una struttura di memorizzazione particolarmente complessa**, così come **complessi sono gli algoritmi** per navigarli (*algoritmi di routing*).
- **L'aumento della complessità** implica una **diminuzione della capacità di memorizzazione** (size).

Comparazione

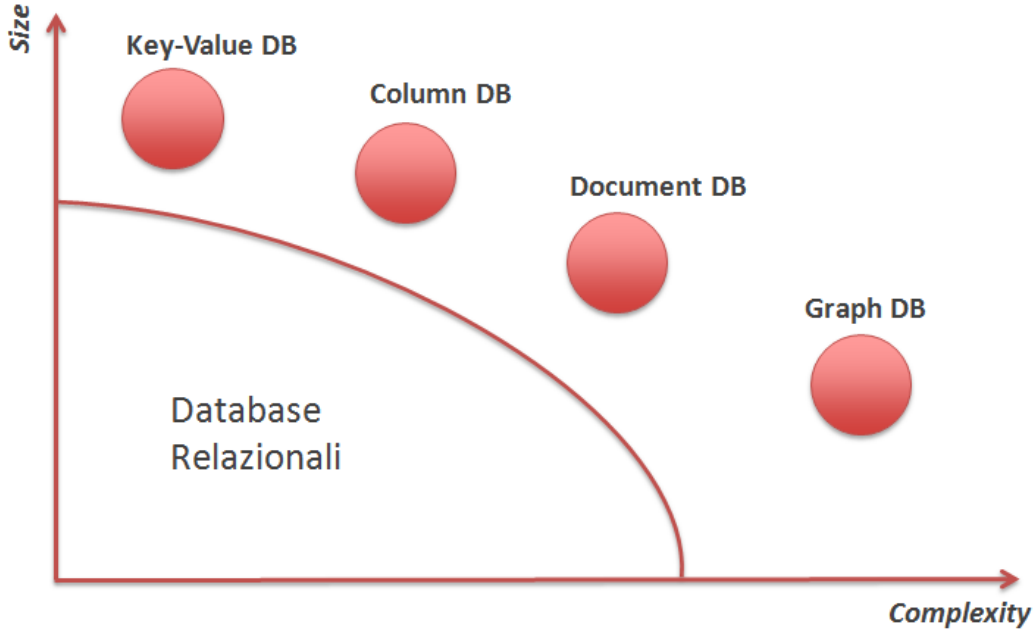
Valutazione delle soluzioni NoSQL DB sulla base dei parametri ***size*** e ***operational complexity***.



- La **famiglia dei DB a grafo** fornisce delle **routine di accesso ai dati molto performanti**, ma hanno il **limite di non poter trattare troppi dati**.

Comparazione

Valutazione delle soluzioni NoSQL DB sulla base dei parametri ***size*** e ***operational complexity***.



- La **famiglia dei DB chiave-valore** è invece adatta a **memorizzare grandi quantità di dati**, poiché utilizzano **l'*hash table*** come struttura dati di memorizzazione. Le **chiavi** sono quindi **scorrelate** tra loro e ciò garantisce una **buona scalabilità orizzontale**.

Soluzioni NoSQL Database

MongoDB (www.mongodb.org)

- E' un **NoSQL database**, ***Open Source***.

- E' di tipo **Document-oriented Database** con schemi dinamici (*schemaless*). I dati archiviati sotto forma di *document* in stile **JSON (BSON)**.

- Garantisce il **supporto di indici flessibili** e **query complesse**.

- **Replicazione integrata** di dati per **un'elevata disponibilità**.

- Modulo **auto-sharding** per la **scalabilità orizzontale**.

- Soluzione **sviluppata** per raggiungere **elevate performance** su operazioni di **lettura/scrittura**, e in grado si essere **altamente scalabile con il recupero automatico dei fallimenti**.

JSON

- Acronimo di ***JavaScript Object Notation***, è un formato adatto per lo scambio dei dati in applicazioni client-server.

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

BSON

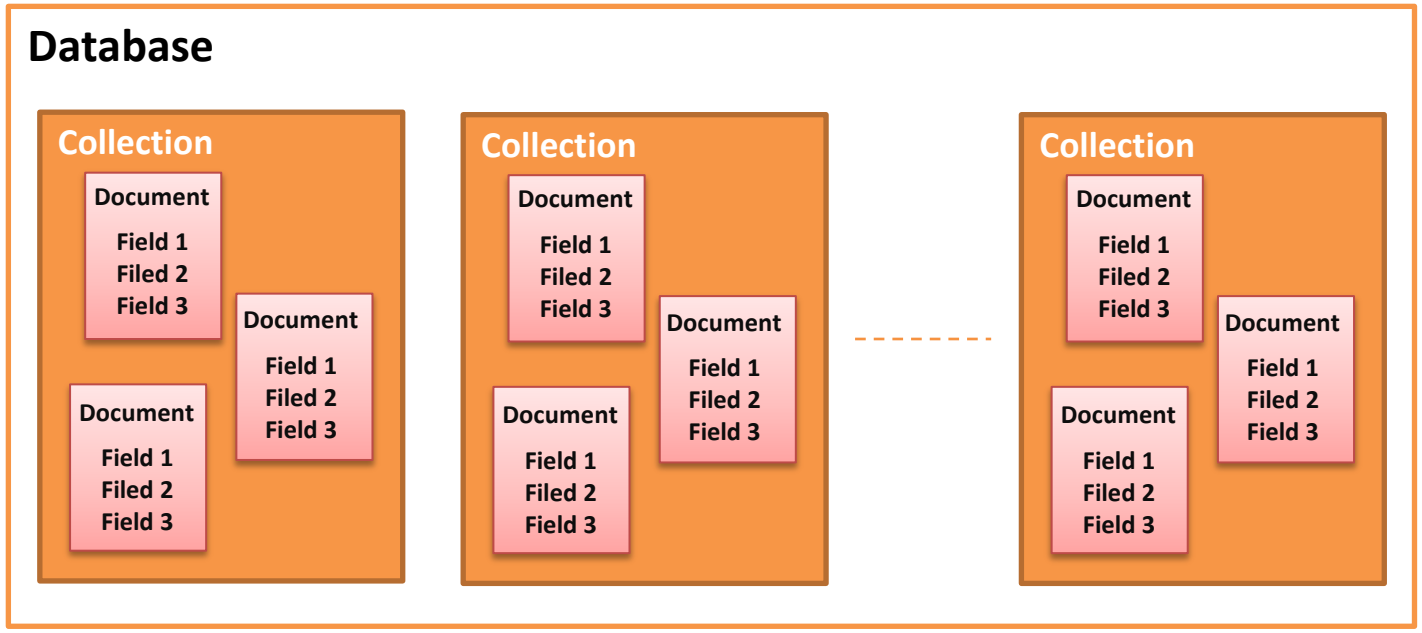
- ***Binary JSON***, è un'estensione del formato JSON che introduce anche il supporto di tipi di dati aggiuntivi: *double, string, array, binary, null, date, int32, int64, object id…*

```
{"hello": "world"}                → "\x16\x00\x00\x00\x02hello\x00
                                    \x06\x00\x00\x00world\x00\x00"

                                  "\x31\x00\x00\x00\x04BSON\x00\x26\x00
                                    \x00\x00\x020\x00\x08\x00\x00
{"BSON": ["awesome", 5.05, 1986]} → \x00awesome\x00\x011\x00\x33\x33\x33\x33\x33\x33
                                    \x14\x40\x102\x00\xc2\x07\x00\x00
                                    \x00\x00"
```

MongoDB – Data Model

- Un sistema basato su **MongoDB** contiene un **set di Database** cosi strutturati:



- Un ***database*** contiene un **set di *collections***, composte al loro interno da un **insieme di *documents***. Ogni ***documents*** è composto da un **set di campi** (***fields***), ciascuno dei quali è una **coppia chiave/valore**.

MongoDB e RDBMS

E’ possibile individuare una sorta di **analogia** tra i concetti dei ***database relazionali*** e quelli presenti in ***MongoDB***.

SQL	Mondo	Differenza
Database	***Database***	
Tabella	***Collection***	Le collection non impongono restrizioni sugli attributi (schemaless)
Riga	***Document***	Volori come oggetti strutturati
Colonna/Campo	***Field***	Maggior supporto ai tipi di dato
Indici	***Indici***	Alcuni indici caratteristici
Join	***Embedding/linking***	Gestione delle referenze lato applicazione
Primary Key	***Object ID***	

MongoDB – Data Model

- La ***chiave*** è una **stringa.**

- Il ***valore*** di un *field* può essere un tipo base (*string, double, date…*), oppure essere a sua volta un **document** o un *array di document*.

- Ciascun documento ha il **campo predefinito "_id"**, che può essere settato in fase di inserimento, altrimenti il sistema ne assegnerà uno univoco in modo automatico.

```
{
  _id: ObjectId ('243252351224df35435bd35wre3'),      //ID del documento
  nominativo: { nome: 'Gino', cognome: 'Bianchi'},    //valore di tipo documento
  data_nascita: new Date('Feb 12, 1975'),
  PIVA: '112334797634',                               //valori con tipo di dato base
  promozioni_utilizzate: ['PROMO 1', 'PROMO4'],       //array
}
```

MongoDB – Modellazione dei dati

La **modellazione dei dati** in MongoDB tiene conto in particolare di **tematiche** che riguardano:

- L'***indicizzazione***
- La ***normalizzazione*** o ***denormalizzazione*** dei dati
- Lo ***Sharding***

MongoDB - Denormalizzazione

- **MongoDB** ha uno **schema flessibile**. I *documenti* **non hanno vincoli di struttura** imposti dalle *collection*, infatti:

    1. Stessa ***collection*** -> ***documenti*** con **differenti campi o tipi di dato**.

    1. Un **campo** di un documento può contenere una **struttura complessa** come un ***array di valori*** o un ***altro documento***.

In base alla seconda caratteristica è possibile:

- **Denormalizzare la struttura dati. Tutti i dati** che descrivono un entità e quelli ad essi collegati **sono inclusi all'interno del documento**.

- **Si mantengono i documenti separati**, aggiungendo però ad ognuno di essi un riferimento che permetta di legarli, es. *il campo "_id"*.

MongoDB - Denormalizzazione

Struttura Denormalizzata

```
db.autori.insert({
    "_id": 3,
    "Nome": "Gabriele",
    "Cognome": "D'Annunzio",
    "opere": [
                {   "titolo": "il piacere",
                    "anno": "1889",
                },
                {   "titolo": "il fuoco",
                    "anno": "1900",
                },
                {   "titolo": "la figlia di Iorio",
                    "anno": "1903",
                }
            ]
    })
```

Struttura Normalizzata

```
db.autori.insert({
    "_id": 3,
    "Nome": "Gabriele",
    "Cognome": "D'Annunzio",
})
db.libri.insert({
    "titolo": "il piacere",
    "anno": "1889",
    "id_autore": 3
})
db.libri.insert({
    "titolo": "il fuoco",
    "anno": "1900",
    "id_autore": 3
})
db.libri.insert({
    "titolo": "la figlia di Iorio",
    "anno": "1903",
    "id_autore": 3
})
```

- **MongoDB non supporta le *JOIN***. Se si adotta la *struttura normalizzata*, le **relazioni** tra i documenti si esplicitano tramite **query in cascata** che recuperano i documenti.

- **Convenzione DBRef** – specifica un ***documento*** tramite: ***database -> collection -> _id***

MongoDB - Interazione

- Da **riga di comando** è possibile *eseguire delle* ***semplici istruzioni***, oppure *lanciare* ***script*** *salvati su file*.

Istruzione	Esempio	Descrizione
db	`db`	Restituisce il database corrente.
use <nome db>	`use impiegati`	Cambia il db corrente in impiegati.
db.<collection>.insert()	`db.studenti.insert( { nome: 'Gianni', Cognome: 'Verdi'})`	Inserisce un utente nella collection studenti senza specificare l'ID
db.<collection>.find()	`dc.studenti.find()`	Elenca tutti i documenti della collection studenti
db.<collection>.drop()	`db.studenti.drop()`	Elimina l'intera collection dal DB

- Oltre ai comandi per effettuare ***inserimenti, modifiche e recupero*** *di documenti*, esistono anche i **cursori**, *cioè degli oggetti che permettono di* ***iterare su un set di documenti*** *restituiti come* ***risultato di una query***.

MongoDB - Interazione

```java
import com.mongodb.MongoClient;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCursor;

public class MongoDB_test {
    public static void main(String args[]) throws Exception {
        //istanza studente
        MongoClient mongoClient = new MongoClient();
        //recuperiamo il database
        DB db = mongoClient.getDB("test");
        //creiamo una collection
        DBCollection coll = db.createCollection ("studenti", null);
        //creiamo un documento
        BasicDBObject doc =
            new BasicDBObject("nome", "Gianni").
            append("cognome", "Verdi").
            append("indirizzo", newBasicDBObject("comune", "Firenze")).
            append("provincia", "FI"));
        //aggiungiamo il documento alla collection
        coll.insert(doc);
        //creiamo un documento per l'esecuzione di una query
        BasicDBObject query = new BasicDBObject ("cognome", "Verdi");
        //utilizziamo un cursore per iterare sui risultati della
        //query e mostrare ogni documento
        DBCursor cursor = coll.find(query);
        try {
            while(cursor.hasNext()){
                System.out.println(cursor.next());
            }
        } finally {
                cursor.close();
        }
    System.out.println("");
    }
}
```

Nell' esempio con l'uso delle **API Java** si:

- **Creano le istanze** del **client *MongoDB*** e del **database *test***.
- **Crea** la nuova **collection *"studenti"***.
- **Crea** un **documento** che modella uno studente, con i campi ***nome, cognome e indirizzo*** (*documento annidato*).
- **Inserisce** il documento nella collection.
- **Crea** e si esegue la **query** che effettua una ricerca sul campo cognome.
- **Usa** un ***cursore*** per stampare a video i documenti recuperati dalla query.

MongoDB – Esempi di Query

1. ***Selezione di tutti i documenti in una collezione***

```
db.inventario.find()
```

2. ***Corrispondenza esatta su un Array (campo multivalore)***

```
db.inventario.find({tags:['frutta',‘carne',‘pesce']}
)
```

3. ***Specifica condizione AND***

```
db.inventario.find({type:‘carne',prezzo:{$lt:9.95}})
```

4. ***Corrispondenza su un campo in un Subdocument***

```
db.inventario.find({'producer.company': 'ABC123’})
```

Tecnologie per i Big Data

- Tra le **tecnologie** che permettono la gestione e l'analisi *grandi moli di dati non-strutturati* (**Big Data**) troviamo sicuramente:
  - ***Hadoop*** (con le *"componenti"* **HDFS** e **MapReduce**).
  - Meccanismi di *sharding* e *replicazione* dei **NoSQL DB**.

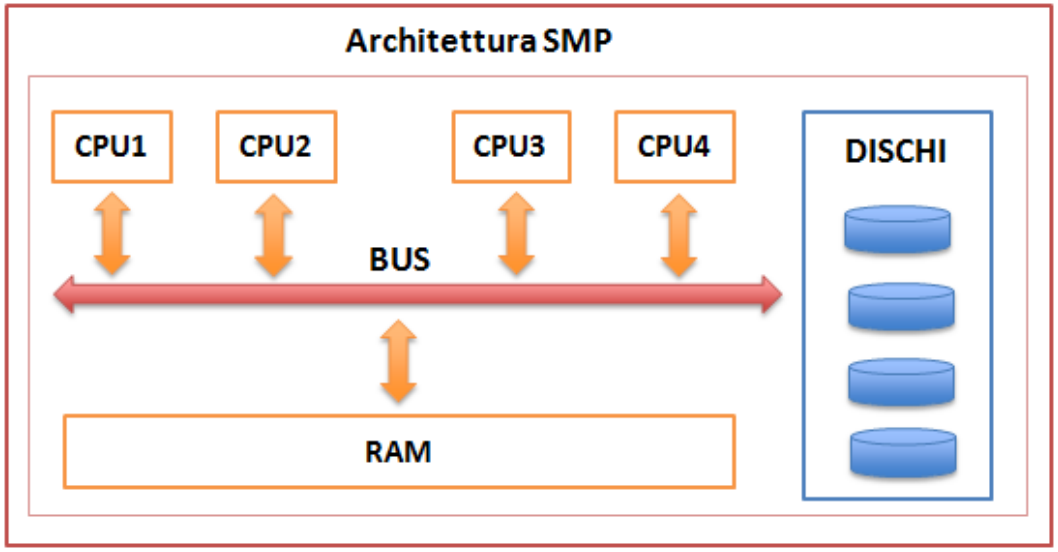
- Queste soluzioni permettono:
  - L'**uso di HW di costo medio/basso**, ottenendo comunque buone capacità di calcolo.
  - La **gestione dei fallimenti** (*fault-tolerance*).
  - L'**ottimizzazione delle performace** via SW.

Osservazione

I **Big Data** sono caratterizzati dalla ***Variabilità***, cioè una *mancanza di struttura ben definita*. Ciò complica la loro rappresentazione in **modello relazionale**.

Tecnologie per i Big Data - SMP

- **Dati caratterizzati solo da *Volume* e *Velocità***. In questo caso gli RDBMS potrebbero fornire un supporto di memorizzazione, come?!
- Generalmente gli **RDBMS** sono ospitati su **sistemi di tipo SMP** (*Symmetric Multi Processing*):
  - Composti da **più processori diversi** che **condividono** lo stesso ***OS***, la stessa ***memoria RAM*** e lo stesso ***bus Input/Output*** (canale di collegamento tra processore e device esterni che consente il passaggio dei dati).



shared everything

Tecnologie per i Big Data - SMP

- Gli **SMP** sono una **soluzione economica** per la **scalabilità dei sistemi**, realizzata *aggiungendo RAM, CPU e dischi* fino al limite installabile.
- **Applicazioni** sviluppate per far **lavorare in parallelo i vari processori** ottimizzando la **suddivisione del carico di lavoro**.
- In **ottica RDBMS**, questi sistemi sono **efficienti in applicazioni OLTP** (*On Line Transaction Processing – molti inserimenti, update e scritture*).

(-) Non adatti ad elaborare **grandi moli di dati**, poiché il **sovraccarico del BUS** può portare all'***effetto collo di bottiglia***.

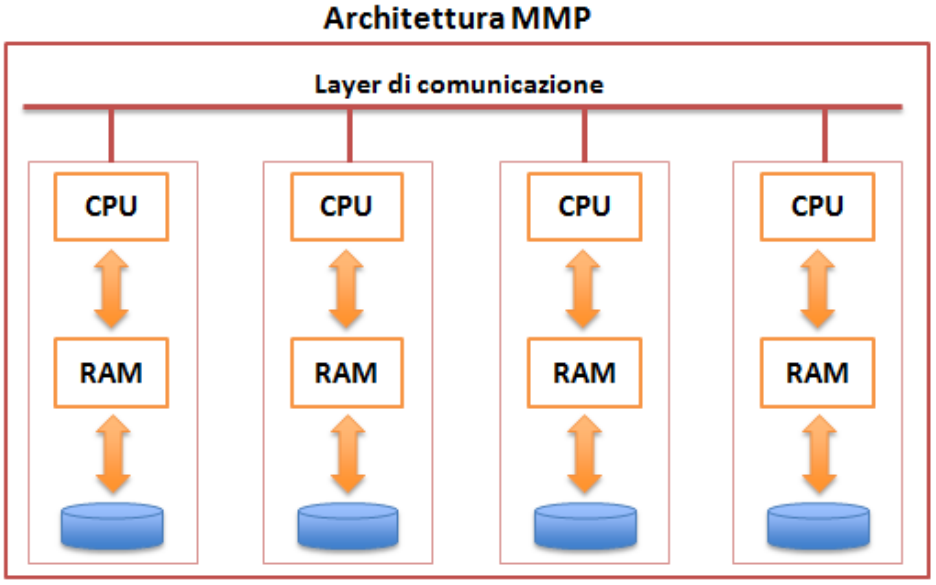
- **Possibili soluzioni** sono le **architetture NUMA** (*Non Uniform Memory Access*).

Processori accorpati in moduli, possono accedere alla memoria del proprio modulo o di altri (con tempi di accesso differenti).

scalabilità verticale

Tecnologie per i Big Data - MPP

- Quando il **Volume dei dati aumenta** in modo considerevole, i **sistemi MPP** (*Massive Parallel Processing*) sono **maggiormente adatti** per l'elaborazione.
- **Ogni processore** ha un **sistema di I/O** e una **RAM *dedicati.***
- **Operazioni** da svolgere sono **suddivise in *task*** *paralleli e indipendenti*.
- **Processori comunicano** tra loro mediante un'**interfaccia di *messaging*** (scompaiono i limiti legati alla *condivisione del bus*).



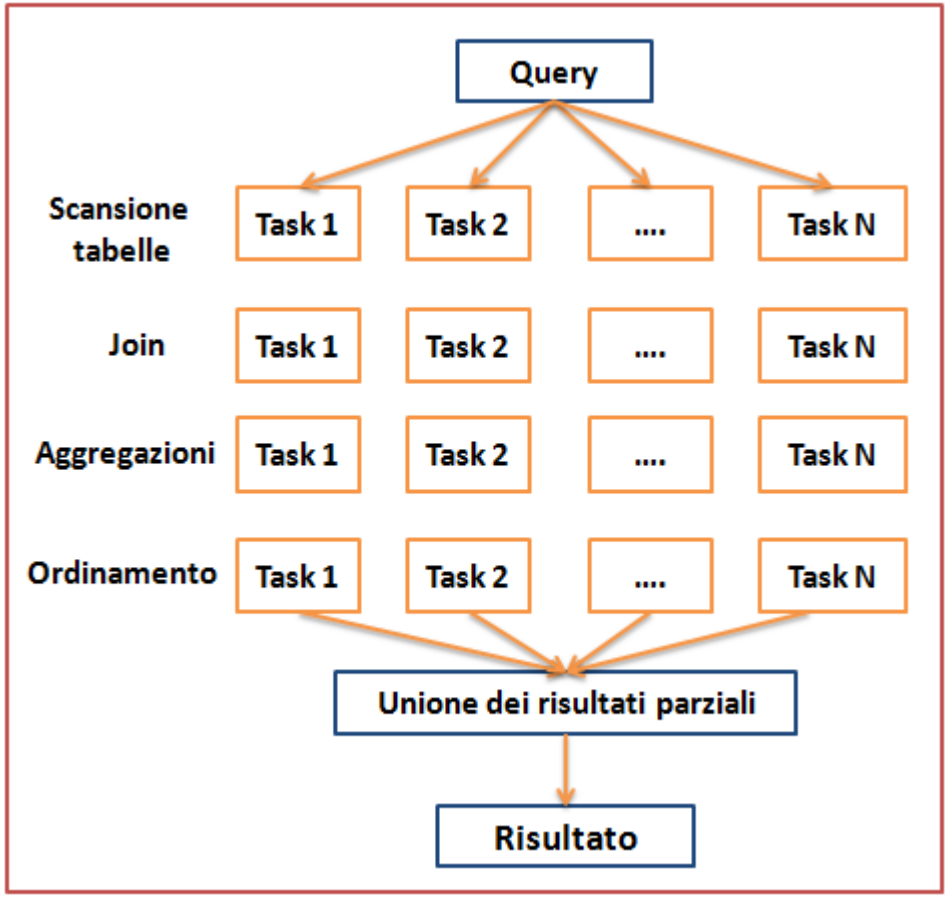
shared nothing
o
parellel everithing

Tecnologie per i Big Data - MMP

- **Funzionamento** dei sistemi **MPP** composto solitamente da **due fasi**:
  1. ***Fase di comunicazione***, *definizione dei processi da elaborare*.
  2. ***Fase di calcolo***
- Questi sistemi hanno una forte **impostazione distribuita**, la **rete che connette i vari *nodi* è estremamente importante** e deve essere regolata in modo che non si presentino dei colli di bottiglia.
- Una corretta e attenta **gestione delle repliche dei dati tra i nodi** permette di **raggiungere e garantire un'*alta disponibilità***.

- Architetture ***shared-nothing*** (o parallel everything)
- ***Risorse HW dedicate*** (CPU, RAM, dischi)
- SW ad-hoc per sfruttare la ***parallelizzazione delle operazioni***

Tecnologie per i Big Data - MPP



Schema di esecuzione query su architettura MMP

- Una **query SQL** è **suddivisa** in una **serie di operazioni** che vengono **eseguite** in *parte in sequenza* e in *parte in parallelo*.

- Ogni **operazione** è a sua volta ***scomposta in N task***, in modo da sfruttare *l'architettura parallela minimizzando la contesa di risorse*.

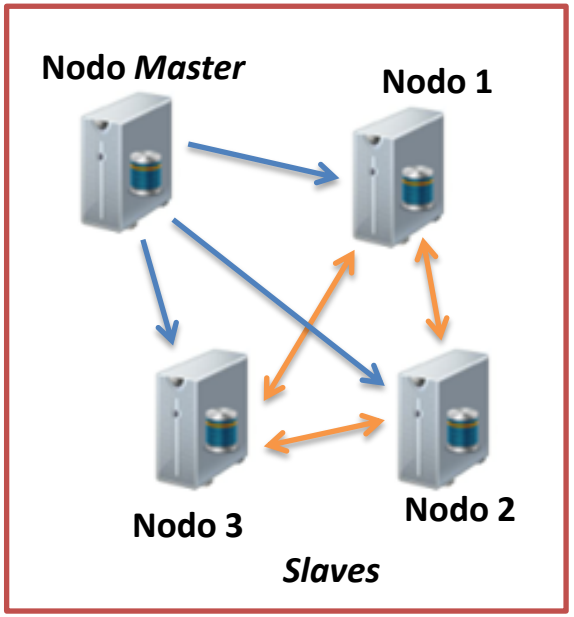
- Si può osservare che gli *unici punti critici* sono all'**inizio della query** e al **termine del processo**, nel momento in cui vanno riuniti i risultati provenienti da diversi nodi.

Tecnologie per i Big Data – MPP Vs. Hadoop

- I **sistemi MPP** sono uno strumento **ideale** per **la gestione di dati** che presentano una **rappresentazione tabellare**, e riescono anche a far fronte al fattore ***Volume*** dei Big Data.
- Rispetto alla ***Variabilità*** (assenza di struttura), i sistemi MPP presentano gli stessi limiti degli RDBMS tradizionali.

Hadoop *è una piattaforma software open source finalizzata al calcolo su un sistema di computer autonomi (nodi) collegati in rete.*

- **HDFS:** ***file system distribuito*** *per salvare dati su un cluster di computer*.
- **MapReduce:** ***paradigma di programmazione*** *realizzato per offrire scalabilità e tolleranza ai guasti.*



Aspetti Architetturali

Dimensione del Cluster

- Il **numero di nodi** in ogni cluster influisce i t**empi di completamento di un processo** (*job, funzione, etc.*).
- Un **numero maggiore di nodi** corrisponde ad un **tempo di completamento minore** di un job.

Dati di input

- **Maggiore** è la **dimensione del dataset** inizialmente in ingresso, più sarà il **tempo di elaborazione dei dati** e di **produzione dei risultati**.

Data Node

- **Un numero maggiore di nodi in un cluster** corrisponde ad una **maggiore potenza computazionale** e quantità di **memoria disponibile**, ciò comporta un **minor tempo** nel **completamento di un *job***.

Aspetti Architetturali

Localizzazione dei Dati

- **Non è possibile assicurare** che i **dati** siano **disponibili localmente su un nodo**. Spesso è necessario **ricercare i blocchi di dati** da elaborare con un significativo **aumento dei tempi di completamento**.

Network

- Il **network influisce in modo importante sulle performance** finali di un sistema di gestione dei Big Data.
- Le **connessioni tra diversi nodi e cluster** possono risultare utili nelle operazioni di lettura/scrittura.
- E' ideale un **network che assicuri un'elevata disponibilità e resilienza**, ottenute con meccanismi di ridondanza.

Aspetti Architetturali

CPU

- Un **numero maggiore di processi da eseguire** si traduce in un **utilizzo più intenso della CPU**.
- La potenza della CPU a disposizione influisce quindi sulle performance dell'intero sistema.

Memoria

- In caso di **applicazioni *"memory-intensive"*** è consigliabile avere una **quantità di memoria** su ciascun nodo server che sia in grado di **coprire le esigenze dell'intero cluster**.
- Spesso 2-4 GB di memoria su ciascun server potrebbero **non risultare sufficienti** (es. SiiMobility 12GB memoria).

- Database **NoSQL** *Open Source*, scritto in *Erlang con C/C++*

- E' un **DB di tipo Chiave-Valore** che implementa i principi descritti nel paper *Amazon's Dynamo*.

- E' un sistema **Masterless**, basato sul principio ***"eventually consistent"***

- I **dati** sono **distribuiti automaticamente** sui vari **nodi** utilizzando un ***"consistent hashing"***.

- Mette a disposizione diversi **strumenti** e **componenti aggiuntivi**.

Riak Caratteristiche

Scalabilità

- I **dati** sono ***"ribilanciati"*** **in maniera automatica**, senza tempi di latenza, quando viene **eliminata o aggiunta una macchina**. I **dati** sono **distribuiti** sul cluster e le performance crescono in modo lineare con l'aggiunta di nuova capacità.

Disponibilità

- Quando un **nodo non è più disponibile** un suo ***"vicino"*** si assume le responsabilità delle operazioni di scrittura o degli aggiornamenti.

Semplicità operazionale

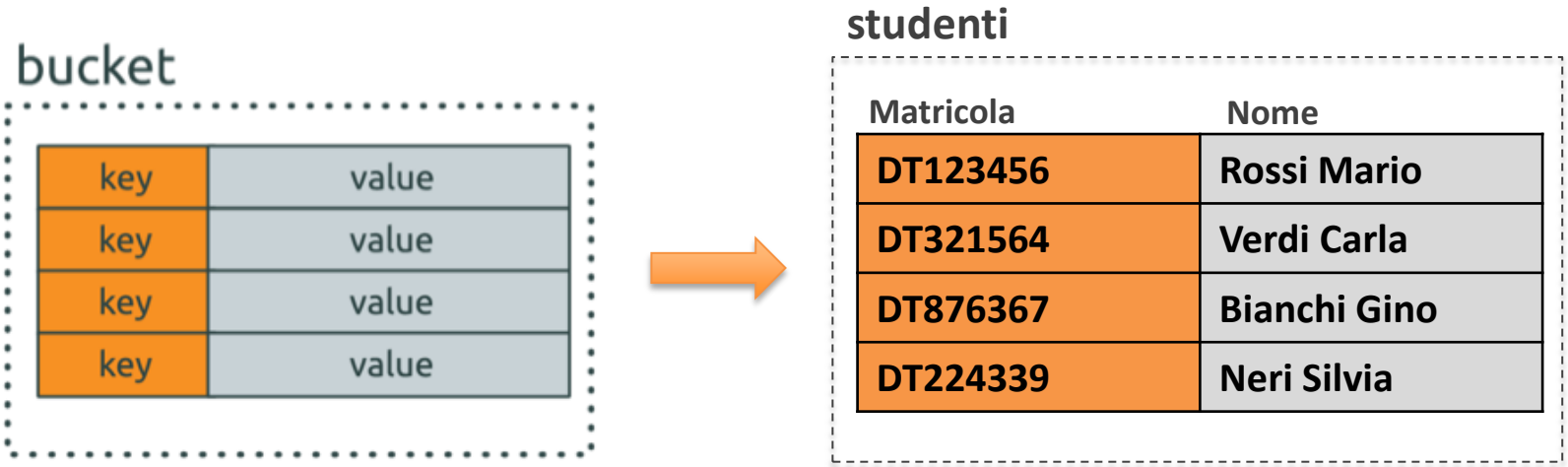
- **Aggiungere una nuova macchina** ad un *cluster Riak* è **un'operazione semplice** senza un eccessivo carico operazionale.

Riak Caratteristiche

Modello Dati Semplice

- I dati sono memorizzati come **coppie chiave-valore** all'interno di una struttura denominata **bucket** (*secchio*).

- E' accettato *qualsiasi tipo di dato*, gli *oggetti sono memorizzati su disco in formato binario*, secondo la **coppia bucket-chiave.**



Sviluppare codice per un modello di questo tipo è più **semplice ed efficiente**, adatto in particolare per applicazioni che richiedono *interazioni rapide e frequenti*.

Riak Caratteristiche

- Gli **oggetti** sono *l'unica unità di memorizzazione dei dati in Riak*, sono cioè strutture a cui è possibile accedere mediante la ***coppia Bucket-Chiave***.

- I ***Buckets*** *sono uno spazio virtuale* in cui *archiviare chiavi che fanno riferimento a dati simili*, questa logica ovviamente aumenta la velocità di esecuzione delle query.

- I ***Buckets*** possono essere pensati come ***tabelle se li si compara ai database relazionali*** oppure a delle ***cartelle se comparati ad un file system***.

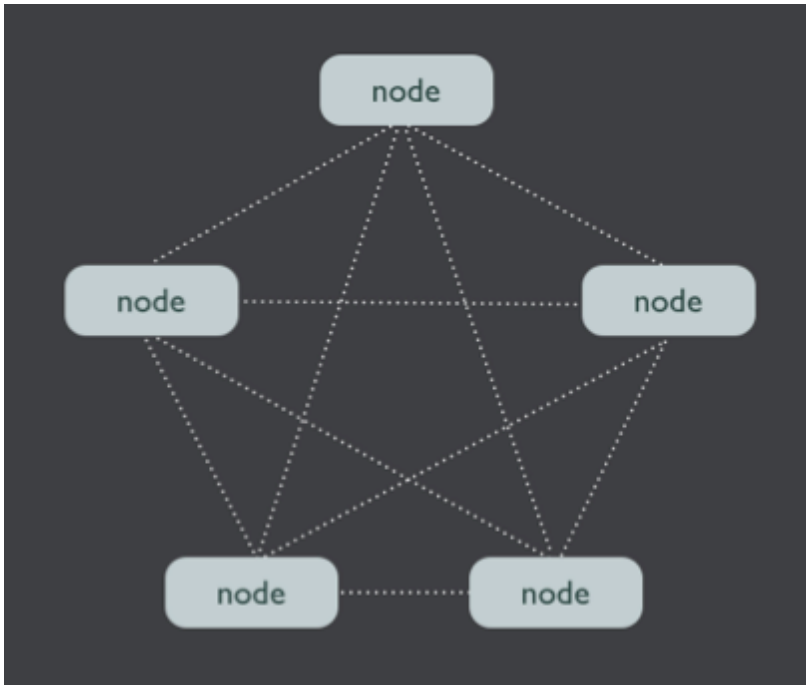
Riak Caratteristiche - Parametri

- Un **attributo** molto potente è ***n_val***, che permette di *indicare il numero di copie che ogni oggetto deve avere dentro il cluster*.
- Per specificare se un *oggetto può essere una copia* di uno già esistente si usa **allow_mult**.
- L'attributo ***Precommit*** può contenere una *lista di funzioni* (Scritte in Erlang e Javascript) *da eseguire **prima** di scrivere un oggetto*.
- L'attributo ***Postcommit*** può contenere una *lista di funzioni* (Javascript, Erlang) *da eseguire dopo aver scritto un oggetto.*

Riak Caratteristiche

Masterless Design

- **Non esistono ruoli *master/slave***. Ogni nodo può inoltrare o rispondere a qualsiasi richiesta in ingresso poiché i dati sono replicati su nodi multipli (*è consigliato un cluster di almeno 5 nodi*).



Questa configurazione garantisce:

- **Uguaglianza tra i nodi** del cluster.
- **Scalabilità** e **disponibilità nelle operazioni di scrittura** (non c'è bisogno dell'autorizzazione da parte del master).

Riak - Approfondimenti (Consistent Hashing)

Questo meccanismo rende possibile la **ridistribuzione automatica** dei dati sui nodi, assicura che i **dati** siano **equamente distribuiti** utilizzando una **struttura** ad ***"anello"*** *(ring)***.**

- Ogni **coppia bucket/chiave** viene ***mappata in un "anello" ordinato*** tramite una **funzione hash da 160-bit** (2^{160} valori).

- L'**anello circolare (*ring*) è diviso** in un certo **numero di partizioni**. Le *partizioni sono di uguale dimensione e ciascuna di esse corrisponde ad un intervallo di valori sull'anello*.

- Ogni ***vnodes*** è **responsabile di una partizione**, e i **nodi del cluster** cercano di **avere lo stesso numero di vnodes**.

Riak - Approfondimenti (Consistent Hashing)

Questo meccanismo rende possibile la **ridistribuzione automatica** dei dati sui nodi, assicura che i **dati** siano **equamente distribuiti** utilizzando una **struttura** ad ***"anello"*** *(ring)***.**

- *Un Nodo nel cluster è responsabile di*:

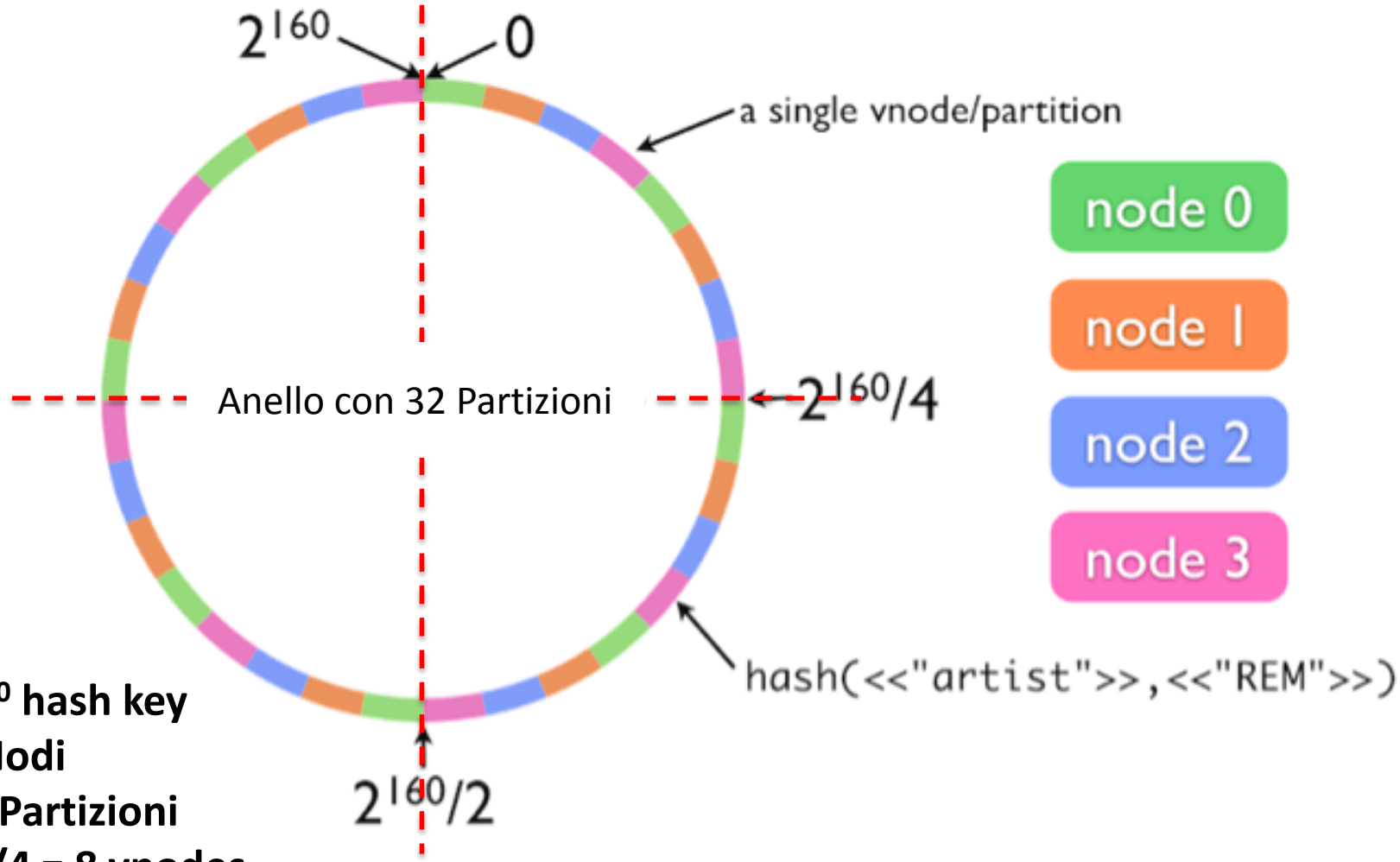
  1/(numero di nodi) dell'anello

- *Un Nodo nel cluster sarà composto da:*

  (numero di partizioni)/(numero nodi) vnodes

Ad **intervalli regolari ogni nodo rivendica (controlla) le sue partizioni** nel cluster, in modo da **mantenere una distribuzione uniforme** dei dati e **evitare** che un nodo sia **responsabile di più di una replica di una chiave**.

Riak - Approfondimenti (Consistent Hashing)



- **2^{160} hash key**
- **4 Nodi**
- **32 Partizioni**
- **32/4 = 8 vnodes**

Riak - Approfondimenti (Consistent Hashing)

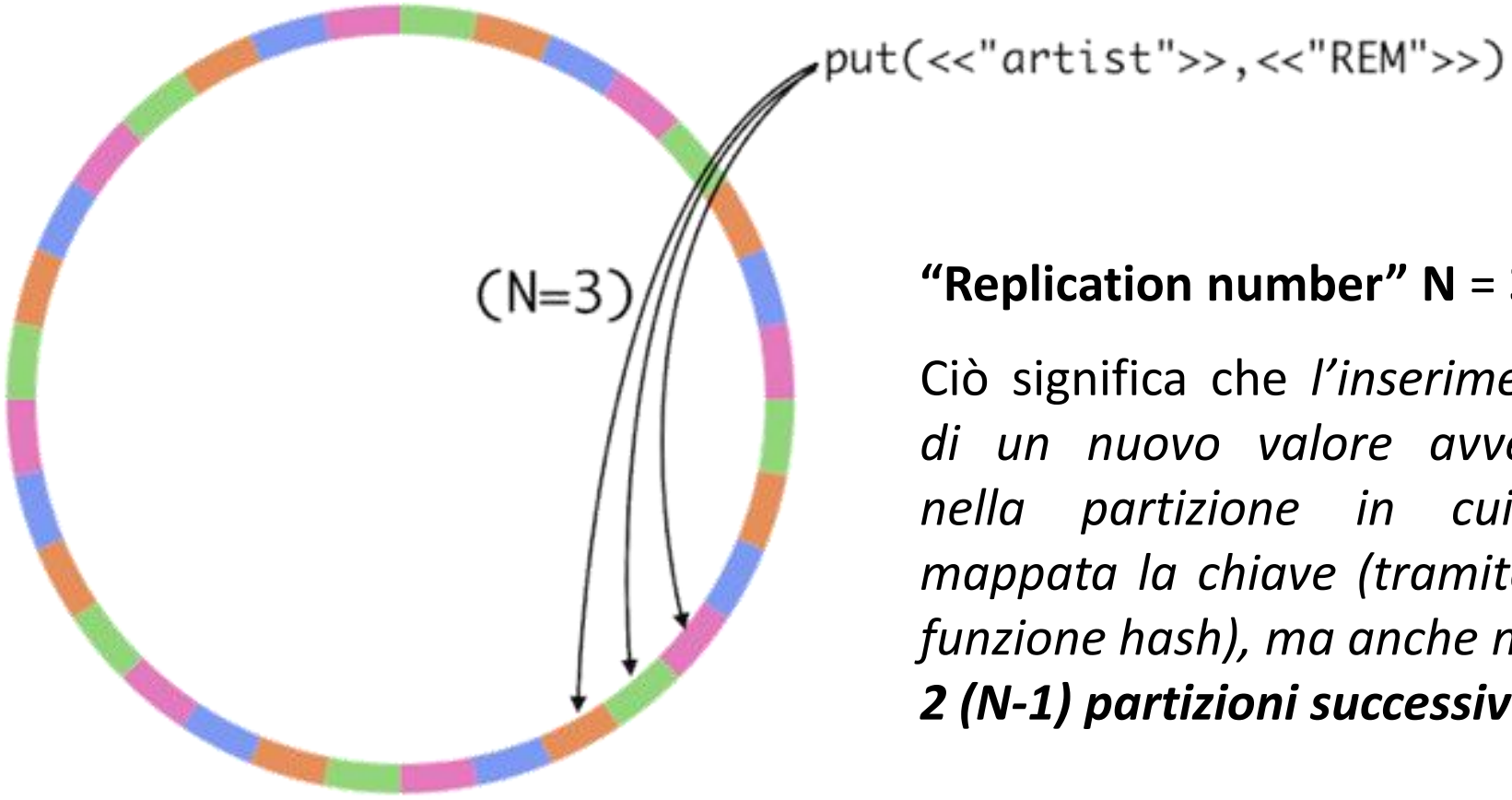
Quando si **memorizza un nuovo valor**e nel cluster **ogni nodo può partecipare come coordinatore.**

- Il **nodo di coordinamento** legge lo **stato del cluster** per determinare quale ***vnode* è responsabile della partizione relativa alla chiave** del valore da inserire.

- La richiesta di ***"put"*** viene inviata prima al **vnode responsabile**, e poi anche ad altri **N-1 vnodes**, responsabili delle **"prossime" N-1 partizioni sul ring**.

- **N** è un **parametro configurabile** del bucket che indica il **numero di copie (repliche) del valore da memorizzare** (in questa richiesta è possibile specificare anche il parametro W).

Riak - Approfondimenti (Consistent Hashing)



“Replication number” N = 3.

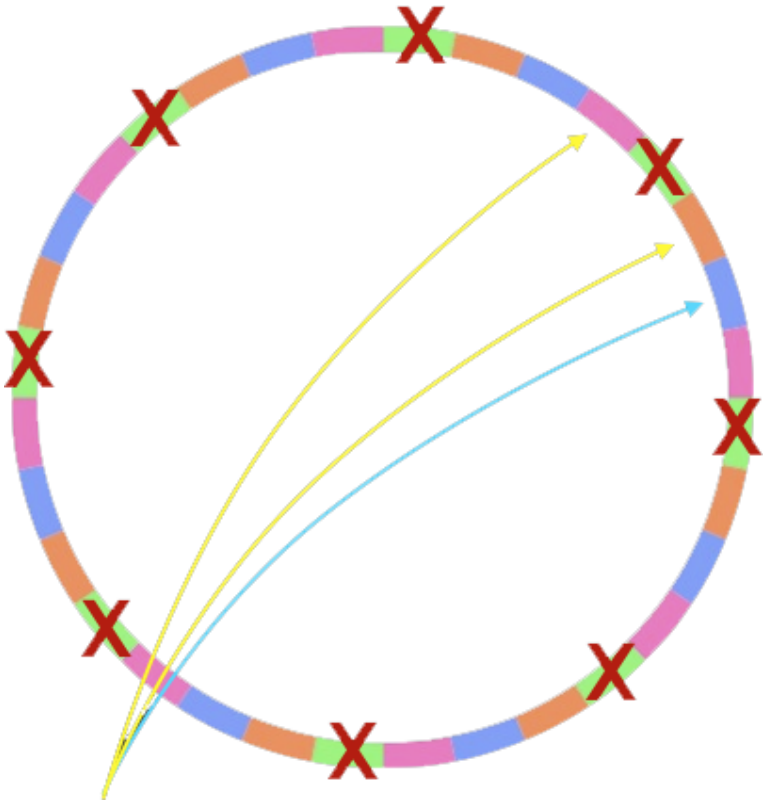
Ciò significa che *l’inserimento di un nuovo valore avverrà nella partizione in cui è mappata la chiave (tramite la funzione hash), ma anche nelle* ***2 (N-1) partizioni successive.***

Riak - Approfondimenti (Fallimento Nodo)

- Quando **un nodo fallisce**, le **richieste** vengono **inviate ad una *"replica secondaria"***, in cui sono stati **copiati i dati** dalle altre **repliche primarie ancora attive**.

- Se il **nodo viene ripristinato**, tramite l'***handoff*** i dati vengono nuovamente ritrasferiti sul **nodo originale**.

- ***Il sistema recupera il suo normale funzionamento***.



Riak - Aggiunta/Rimozione nodo

- Un **nuovo nodo** può essere **aggiunto ad un cluster esistente**. Ciò si realizza mediante una **richiesta join**:

  riak-admin cluster **join** riak@192.168.15.2

- Se l'operazione ha **successo**, il **nodo** riceve lo stato dell'anello e **comincia ad essere attivo** all'interno del cluster.

- Lo stato dell'anello è sempre conosciuto da tutti i nodi del cluster tramite un **"gossip protocol".** *Quando un nodo subisce una modifica (es. sul relativo spazio delle chiavi) la comunica ad una latro nodo e così via a cascata.*

- Dopo l'***handoff*** (*trasferimento di dati da un nodo esistente a quello appena aggiunto*), il **nuovo nodo è pronto** a soddisfare le richieste di interrogazione.

Riak - Vector Clock

- E' un **metodo per tener traccia della versione più recente di un dato.**
- Quando un **nuovo valore** è memorizzato in Riak, viene **etichettato con un "*vector clock*"** che ne stabilisce la sua **versione iniziale**.

```
a85hYGBgzGDKBVIcR4M2cgczH7HPYEpkzGNlsP/VfYYvCwA=
```

- Ad ogni **aggiornamento** il **vector clock è esteso** con uno specifico meccanismo *in modo tale che Riak possa comparare due repliche di un oggetto e determinare*:
  - **Se un oggetto è discendente diretto di un altro oggetto.**
  - **Se più oggetti sono discendenti diretti di un oggetto comune.**
  - **Se più oggetti sono scorrelati tra loro.**

Riak - Vector Clock

- Tramite il ***vector clock*** è possibile **tracciare quali modifiche sono state applicate** ad un oggetto e chi le ha realizzate.

- E' come avere una **sorta di array in cui sono conservati le modifiche** dei vari client e *l'ordine con cui sono state eseguite*.

- In questo modo **è possibile capire se un oggetto è aggiornato** oppure se si sono verificati dei conflitti in fase di scrittura.

- Usando **questa conoscenza**, Riak può applicare, quando possibile, dei ***meccanismi di auto-riparazione di dati non sincronizzati***; o almeno fornire ai client con la possibilità di riconciliare i cambiamenti divergenti con meccanismi legati alle specifiche applicazioni.

eXistdb – Nuova soluzione di indicizzazione

- **2006**, viene introdotta ***"dynamic level numbering" (DLN),*** che consiste in **ID gerarchici** (*secondo la classificazione decimale di Dewey*).

- ***Schema di numerazione basato su ID di lunghezza variabile*** che evita di porre un limite concettuale sulle dimensioni del documento da indicizzare.

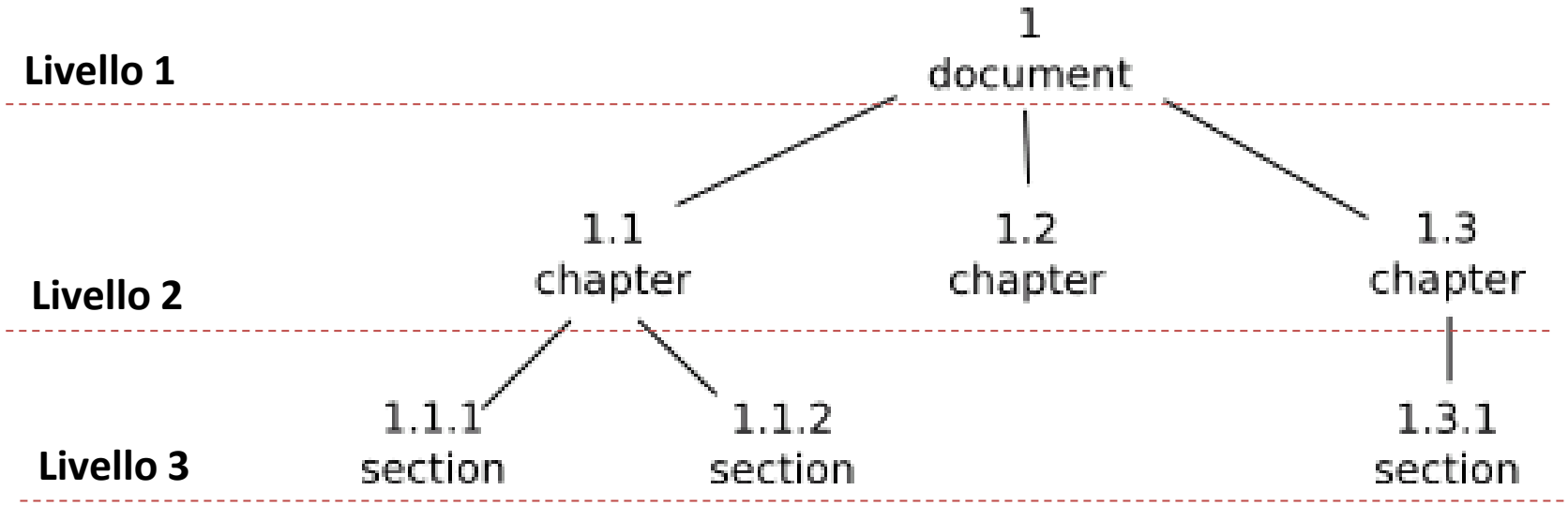
- Ciò porta al **miglioramento** di diversi aspetti, tra cui gli ***aggiornamenti rapidi dei nodi senza re-indicizzazione.***

- **L'ID è una sequenza di valori numerici**, separati da un separatore. **Il nodo radice ha ID=1**. *Tutti i **nodi sottostanti** consistono nell'avere l'**ID** del proprio **nodo padre** usato **come prefisso** a cui va aggiunto un **valore corrispondente al livello.***

eXistdb – Nuova soluzione di indicizzazione

- ***1** è il **nodo radice*** (livello 1), ***1.1 è il primo figlio*** sul secondo livello, ***1.2 è il secondo figlio*** sul secondo livello…e così via. Si può osservare come il **numero delle cifre di un ID, identifica il numero del livello**.

- Utilizzando **questo schema**, determinare la ***relazione tra una qualsiasi coppia di ID dati (nodi) diventa un'operazione banale***, così come identificare il rapporto antenato-discendente.



Processo di elaborazione dei Big Data

- Uno dei contesti in cui i **Big Data** trovano una forte applicazione è quello della ***Business Intelligence***, in cui *contribuiscono a creare dei sistemi di supporto alle decisioni*.

Molte delle tecnologie utilizzate nelle 4 fasi di elaborazione sono legate ***all'ecosistema Hadoop*** (che insieme alle sue componenti è sviluppato in Java).

Acquisizione dei Big Data

- L'**acquisizione** delle diverse tipologie di dati può essere realizzate con mezzi differenti, le **principali categorie** sono:

  - ***API di chi fornisce i dati.*** Si parla principalmente di dati provenienti da Social Network o app di condivisione.

  - ***Importazione dei dati mediante strumenti ETL*** (Sqoop o PDI).

  - ***Utilizzo di software di web scraping.*** Raccolta automatica di dati dal web, ad esempio mediante parser HTML.

  - ***Lettura di stream di dati.*** Tecnologie per acquisire flussi di dati, come ad esempio le piattaforme CEP (dati provenienti dal web o da sensori).

Immagazzinamento e Organizzazione

- In questa fase bisogna considerare **due aspetti**:
  1. La **gestione di grandi moli di dati**.
  2. Dati **non strutturati o semi-strutturati**.

- La **principale tecnologia software** adatta a questo scopo è ***Hadoop***, piattaforma di calcolo distribuito che presenta o supporta diverse componenti:
  - **Hadoop common**: strato sw che fornisce funzioni di supporto ad altri moduli.
  - **HDFS**: file system distribuito che offre una elevata capacità di accesso ai dati.
  - **YARN**: sistema di scheduling e gestione delle risorse condivise.
  - **MapReduce**: sistema di parallel processing per grandi moli di dati.
  - **Hbase**: Database NoSQL di tipo Column Family.

Integrazione

Prima di passare alla fase di analisi è spesso necessario eseguire delle ***integrazioni/elaborazioni*** sui dati immagazzinati.

- **Sqoop**

permette di realizzare **l'integrazione con DB esterni**, spostando dati da e verso Hadoop.

- **Apache Tika**

Sw che permette di **trattare formati differenti** (PDF, HTML, XML, Word, Excel) nello stesso modo. Specie per dati provenienti dal Web o da repository di documenti.

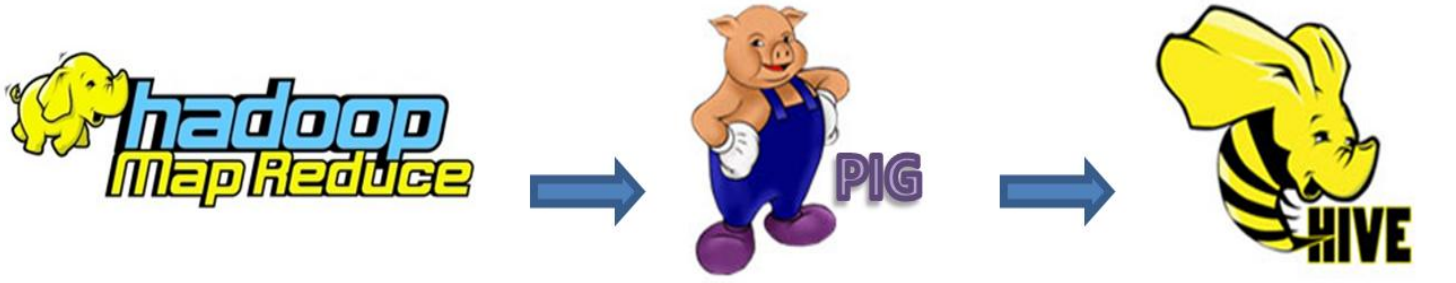
- **Hive**

Sistema di datawarehousing costruito su Hadoop, che permette l'aggregazione di dati, l'esecuzione di query e l'analisi di grandi dataset mediante **HiveQL (simile ad SQL)**. Così si nasconde la complessità di scrivere funzioni MapReduce.

Analisi

- ***L'analisi dei Big Data*** *nel mondo Hadoop può essere fatta con diversi strumenti che sfruttano* ***MapReduce e HDFS****.* Tra i diversi tool a disposizione, importanti sono **Pig** (ecosistema Hadoop), **R** (no Hadoop) e **Mahout** (ecosistema Hadoop - non integrerà i nuovi algoritmi di MapReduce).

- **Pig** con il suo ***linguaggio Pig Latin*** semplifica la scrittura di sequenze di operazioni rispetto a MapReduce; infatti ci colloca tra **MapReduce** e **Hive**, come dimostra il codice necessario per contare le occorrenze di ciascuna parola in un file testuale.



Hadoop

- Hadoop è un **framework open source** per **l'elaborazione, la memorizzazione e l'analisi** di grandi quantità di dati distribuiti e non strutturati.

- E' stato progettato con l'obiettivo di poter **scalare da un singolo server a migliaia di macchine** con un alto livello di **tolleranza ai guasti**.

Hadoop

- **Hadoop** è stato **ispirato da MapReduce**, una *funzione "user-defined" sviluppata da Google* nei primi anni del 2000 per l'indicizzazione del web.

- **Hadoop** è ora un **progetto della Apache Software Foundation**, dove centinaia di collaboratori lavorano per migliorare continuamente la tecnologia di base.

Idea chiave

Invece di processare un **enorme blocco di dati** con una **singola macchina**, ***Hadoop rompe i Big Data in più parti*** *in modo che ogni parte possa essere elaborata e analizzata allo stesso tempo, con un* ***approccio di calcolo parallelo e distribuito***.

Hadoop – Caratteristiche chiave

Hadoop cambia l'economia e le dinamiche del calcolo computazionale su larga scala, definendo una **soluzione di elaborazione che è**:

Scalabile

- Nuovi **nodi possono essere aggiunti**, se necessario, **senza dover modificare** i *formati di dati*, le *modalità di caricamento dei dati*, come vengono *scritti i job*, o le applicazioni ad un livello superiore.

Performante

- Hadoop porta un **calcolo parallelo e massivo su dei commodity server**. Il risultato è una *riduzione consistente del costo per terabyte* dello spazio di archiviazione.
- La **ripartizione dei dati sui nodi** di calcolo permette di *minimizzare i tempi di accesso*, eliminando onerosi trasferimenti di rete.

Hadoop – Caratteristiche chiave

Hadoop cambia l'economia e le dinamiche del calcolo computazionale su larga scala, definendo una **soluzione di elaborazione che è**:

Flessibile

- Hadoop è **schema-less**, e in grado di ***gestire dati strutturati e non, provenienti da più fonti***. I dati provenienti da sorgenti multiple possono essere uniti ed aggregati in modi arbitrari che permettono un'analisi più approfondita di rispetto ai sistemi tradizionali.

Fault Tolerant

- Quando si perde un nodo, il sistema **reindirizza il lavoro** su un'altra macchina che possiede i dati (copia), garantendo una **elaborazione continua senza tempi di attesa**.

Hadoop – Componenti chiave

Le **componenti chiave** che costituiscono il nucleo della piattaforma **Hadoop** sono:

Hadoop common

- Strato di sw comune che ***fornisce funzioni di supporto agli altri moduli***.

HDFS

- E' il **filesystem distribuito** in grado di ***gestire dati in diversi formati*** e di ***regolarne l'accesso*** in modo efficace. ***Garantisce che i dati siano ridondanti*** sui nodi realizzando così la tolleranza ai guasti

Hadoop – Componenti chiave

Le **componenti chiave** che costituiscono il nucleo della piattaforma **Hadoop** sono:

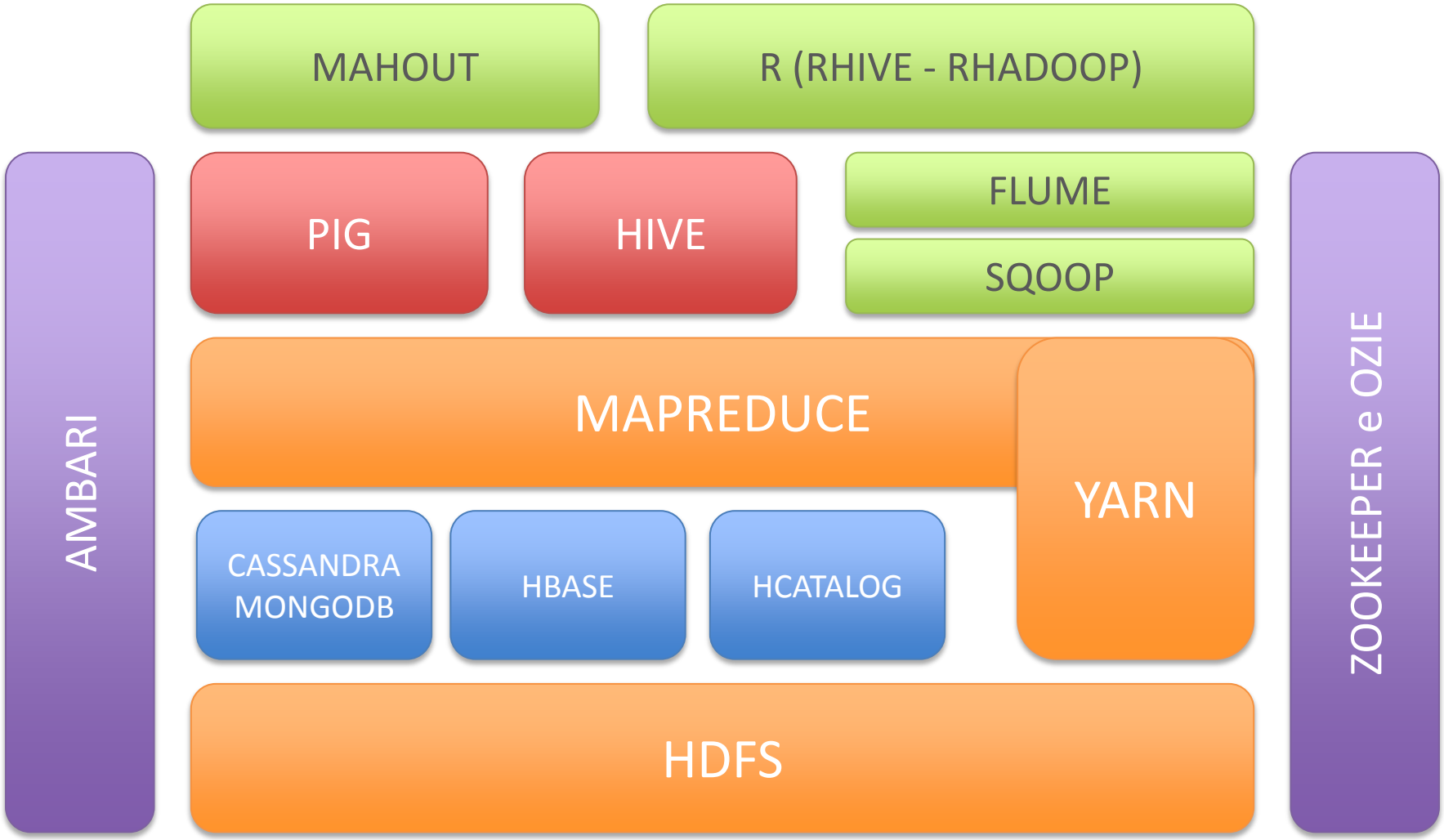
YARN

- Framework di supporto alla creazione di applicazioni o infrastrutture per il calcolo distribuito (presente nelle versioni 0.23.X e 2.X.X).

MapReduce

- E' il sistema di ***parallel processing*** di grandi quantità di dati. Lavora secondo il principio ***dividi et impera***, in cui un problema complesso viene suddiviso in sottoproblemi insieme ai dati, elaborati in modo separato.

Hadoop – Architettura (componenti aggiuntivi)



Hadoop

Uno dei principali obiettivi di Hadoop è quello di consentire la ***scansione di grandi dataset*** e produrre dei risultati attraverso un ***sistema di "batch-processing" distribuito*** e altamente scalabile.

- Apache Hadoop è caratterizzato da **due componenti chiave**:
  - ***HDFS*** (*Hadoop Distributed FileSystem*)
  - ***MapReduce***

Hadoop - HDFS

- ***HDFS*** è un **filesystem distribuito**, eseguito sul filesystem nativo.
- Nasce con l'obiettivo di ***soddisfare requisiti*** di **affidabilità** e **scalabilità**.
- Può gestire un **numero elevato di file**, anche di notevoli dimensioni (*nell'ordine di **GB** e **TB***), tramite **cluster** che possono contenere anche ***migliaia di nodi***.
- Può utilizzare ***commodity hardware.*** Ciò comporta la possibilità di *guasti frequenti e/o nodi non disponibili*.

Capacità di effettuare operazioni di recovery automatiche.

Dal punto di vista dell'architettura un *cluster Hadoop* è composto da diverse ***tipologie di nodi*** (processi che girano sui vari server)

- ***NameNode***
- ***DataNode***
- ***SecondaryNameNode***

HDFS - NameNode

- E' ***l'applicazione in esecuzione sul server principale***.
- Si occupa della **gestione del filesystem** e controlla l'accesso ai file.
- Gestisce il ***namespace,*** cioè l'*elenco* dei nomi **dei file** e **dei blocchi** (da *64 o 128 MB*) in cui sono suddivisi.
- Determina la **distribuzione dei blocchi sui vari DataNode** e le **strategia di replicazione dei file** in modo da garantire l'affidabilità del sistema.
- Controlla **lo stato e l'esecuzione dei vari nodi** del cluster.

HDFS - NameNode

Il NameNode salva il ***namespace*** su due file:

1. ***fsimage –*** è l'ultima immagine del namespace

2. ***journal –*** è il log dei cambiamenti avvenuti nel namespace dall'ultimo aggiornamento del fsimage.

- All'**avvio** il NameNode **unisce il file fsimage con il log dei cambiamenti**, in modo da *ottenere una fotografia aggiornata dello stato del cluster*.

- Questo **snapshot va poi a sovrascrivere l'fsimage** esistente.

HDFS - DataNode

I **DataNode** sono i **processi che girano sui vari nodi del cluster**.

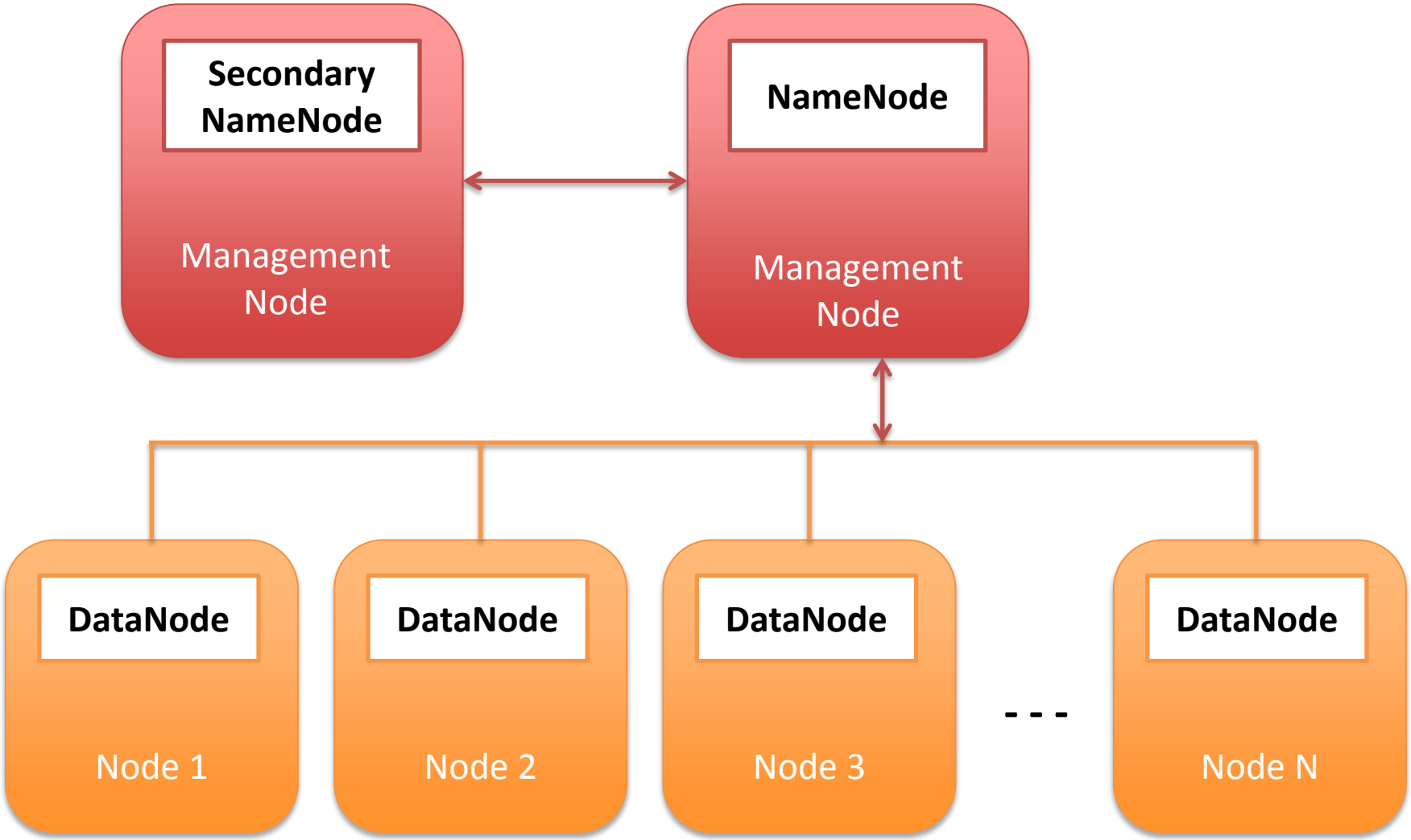
- Solitamente si ha **un DataNode per ogni nodo**.
- Gestiscono fisicamente lo storage di ogni nodo.
- **Eseguono le operazioni di lettura e scrittura richieste dai client**, occupandosi della *creazione, cancellazione e replica dei vari blocchi di dati*.

HDFS - SecondaryNameNode

Non deve essere confuso come un ***nodo di failover per il NameNode***.

- E' infatti un ***servizio di supporto*** **al NameNode** per far si che esso lavori in modo più efficiente.
- Questo processo **scarica periodicamente** il file *fsimage* e il *journal* (dal NameNode), li **unisce in un unico snapshot** che poi **restituisce al NameNode**.
- In alcune versioni questo processo è chiamato ***CheckPointNode***.

HDFS - Architettura



HDFS - Approfondimento

- Nelle ***versioni precedenti la 2.X.X***. il *NameNode era un'applicazione che girava su un solo server*. Ciò poteva portare a **problemi di indisponibilità**.

Soluzione: *due diversi server configurati come NameNode*, **uno attivo** e **l'altro in standby**. Come?

1. Utilizzo del ***Quorum Journal Manager,*** cioè un insieme di processi (JournalNode) a cui il NameNode comunica le modifiche. Il nodo in standby le legge e si mantiene sincronizzato.

2. Utilizzo di uno ***storage condiviso,*** su cui il nodo attivo salva le modifiche del HDFS. Il nodo in standby le legge mantenendosi sincronizzato.

HDFS – File e blocchi

- HDFS organizza i **file come sequenze di blocchi** di uguale **dimensione,** di solito ***64 o 128 MB***, *ridondanti su più nodi*.
- Per ciascun singolo file **è possibile configurare** sia il ***numero di repliche*** che la ***dimensione dei blocchi***.
- Il **meccanismo di replicazione** serve sia a ***garantire la disponibilità*** in caso di guasti, ma anche per ***recuperare i dati in modo più efficiente***.
- Infatti in HDFS per **rispondere ad una richiesta di lettura dati**, vengono **selezionate le repliche presenti sui nodi più vicini** al client che ha effettuato la richiesta.

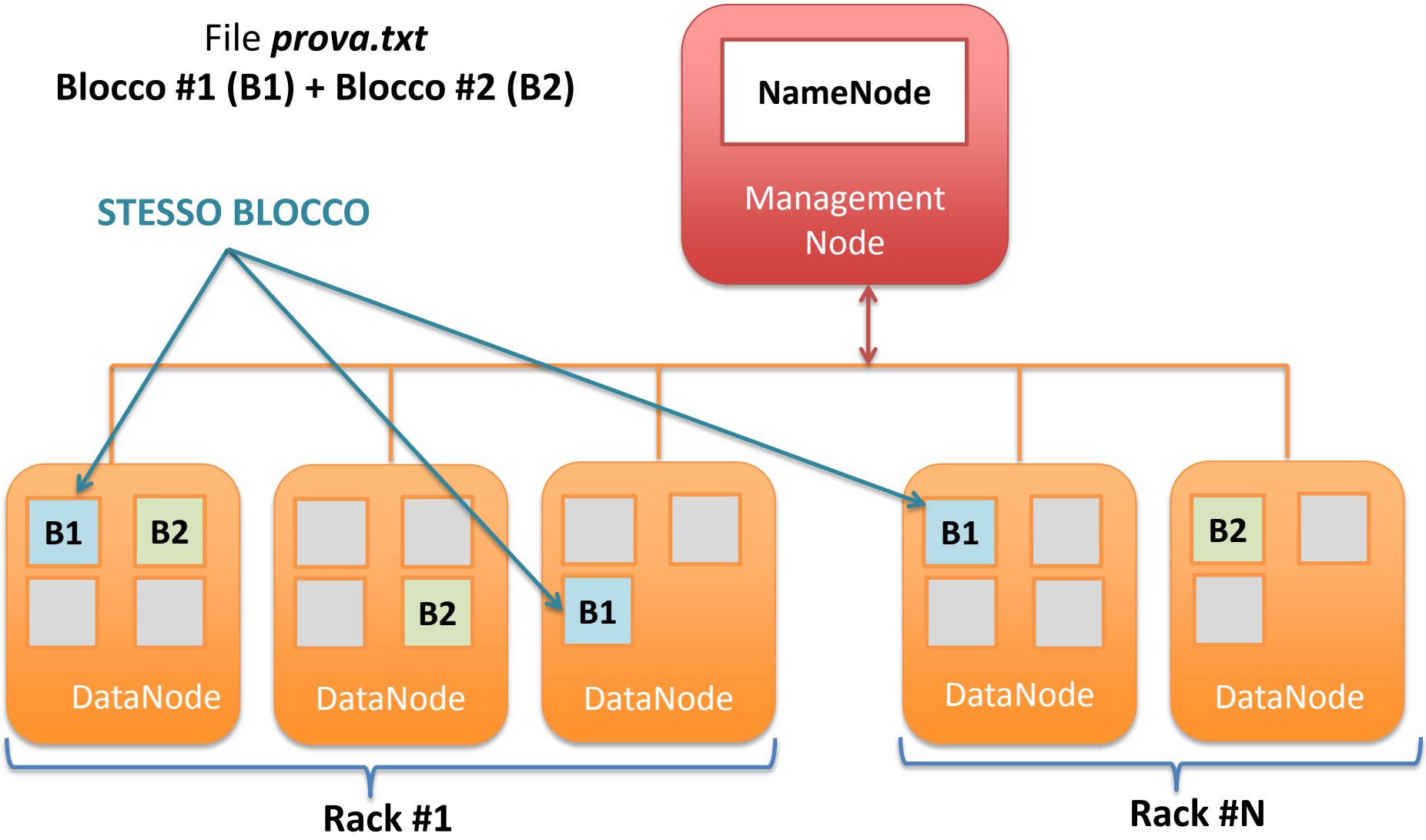
HDFS – File e blocchi

I **file *sono replicati su più macchine*** al momento del caricamento.

- Uno **stesso blocco è memorizzato** su un **numero multiplo di nodi**.
- Ciò ***favorisce*** le **operazioni di lettura** e **garantisce tolleranza ai guasti** (*disponibilità dei dati*).
- Il **valore di *"replicazione"*** di default è pari a **3**.
  - *Prima replica* sul **rack locale**
  - *Seconda replica* sempre sul **rack locale, ma su un nodo differente**.
  - *Terza replica* su un **rack differente**.

HDFS – File e blocchi

File ***prova.txt***
Blocco #1 (B1) + Blocco #2 (B2)



HDFS – Creazione dei file

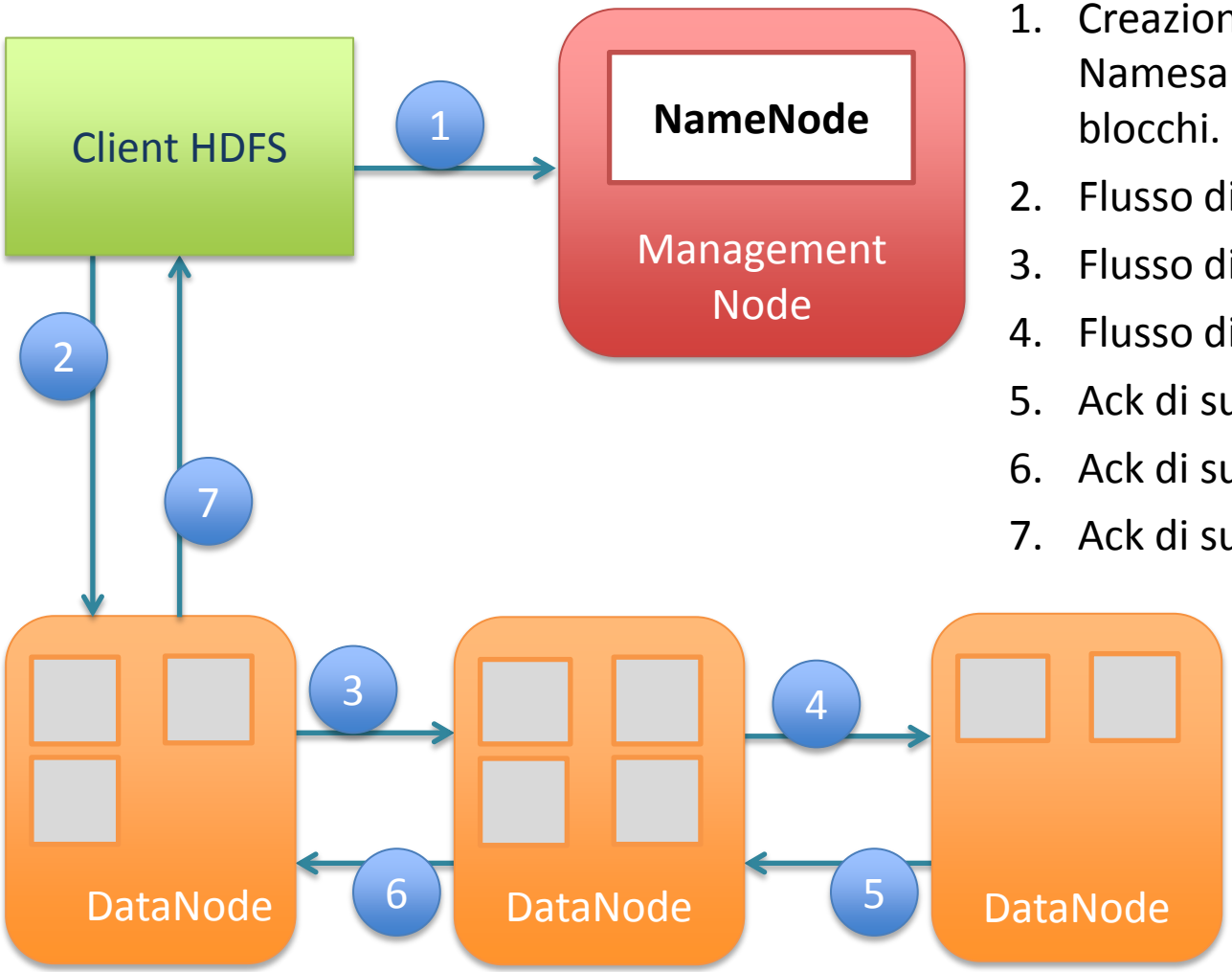
Un file *non viene creato* direttamente **attraverso il NameNode**.

1. Il ***client HDFS crea un file temporaneo in locale***, e solo quando la sua *dimensione è maggiore di quella di un blocco* sarà preso in carico dal **NameNode**.

2. Il **NameNode crea il file** nella gerarchia del filesystem, **identifica DataNode e i blocchi** in cui memorizzare il file.

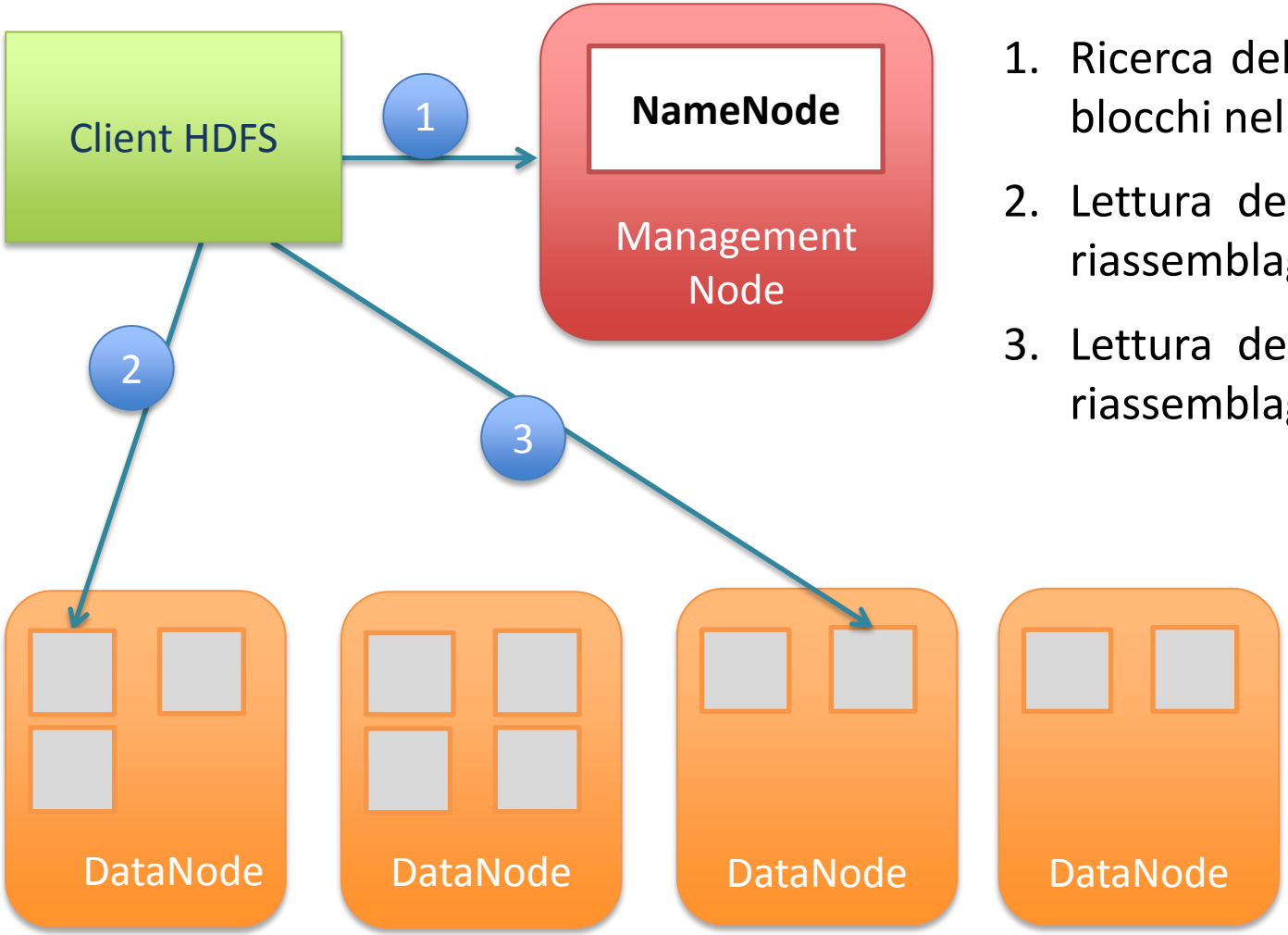
3. Il **client HDFS** riceve queste informazioni e ***provvede a copiare il file dalla cache locale alla locazione finale***.

HDFS – Scrittura di un file



1. Creazione di un nuovo file nel Namesapce, e individuazione dei blocchi.
2. Flusso di dati al primo Nodo
3. Flusso di dati al secondo Nodo
4. Flusso di dati al terzo Nodo
5. Ack di successo/fallimento
6. Ack di successo/fallimento
7. Ack di successo/fallimento

HDFS – Lettura di un file



1. Ricerca della posizione dei blocchi nel cluster.
2. Lettura dei vari blocchi e riassemblaggio del file
3. Lettura dei vari blocchi e riassemblaggio del file

Hadoop – Gli Archivi

Va ricordato che **HDFS ha dei problemi nel trattare file di piccole dimensioni** (inferiore a quella di un blocco).

- I **file utilizzano spazio nel Namespace**, cioè l'elenco di file gestito dal NameNode.
- **Namespace ha un limite di memoria** dovuto al server che ospita il NameNode.
- ***Problema –*** troppi file di dimensione inferiore al singolo blocco occuperebbero il Namespace, ma lo spazio del disco rimarrebbe parzialmente libero.

Hadoop – Gli Archivi

Soluzione – L'uso di **archivi Hadoop** risolve il problema. In questo modo **file di piccole dimensioni sono compattati in file più grandi** che possono essere acceduti in parallelo.

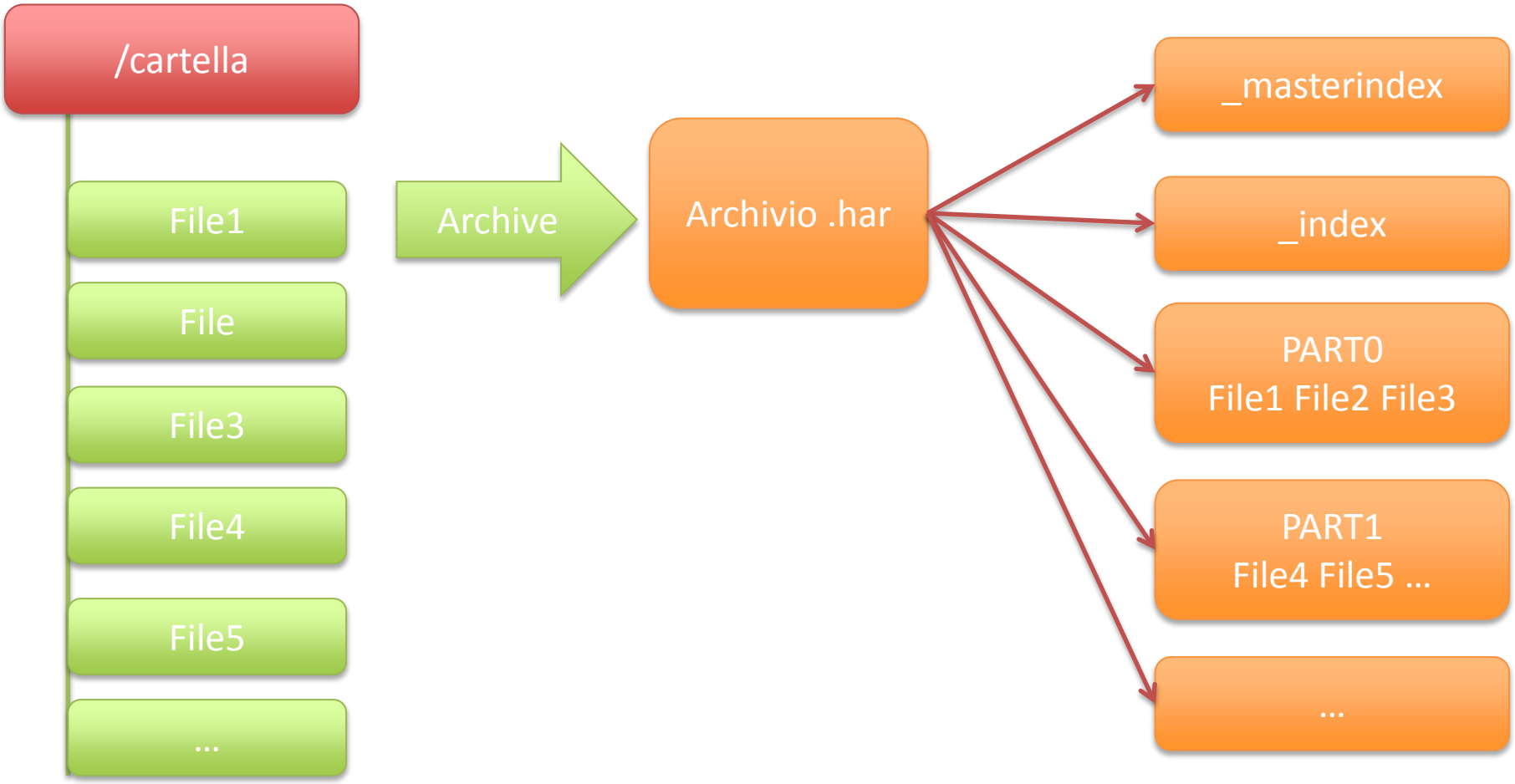
Gli archivi Hadoop si creano attraverso il comando `hadoop archive`, hanno estensione **.har**, e si compongono di tre parti:

1. ***Master index***, che contiene la posizione dei file originali nell'archivio.

2. ***Index,*** che contiene lo stato dei file.

3. ***Le parti,*** che contengono i dati.

Hadoop – Gli Archivi



HDFS – Accesso al Filesystem

L'accesso al filesystem e la sua gestione può avvenire in **due modi**:

- ***Tramite la shell dei comandi.***
- ***Tramite la Web API.***

Comando	Descrizione	Esempio
mkdir	Crea una cartella in uno specifico percorso.	hdfs dfs –mkdir hdfs://namenode/data/test_folder
rmr	Elimina le cartelle e le sottocartelle	hdfs dfs hdfs://namenode/data/test
copyFromLocal	Copia i file dal filesystem locale	hdfs dfs –copyFromLocal file_locale hdfs://namenode/data/test

Hadoop - MapReduce

- *MapReduce* **è l'elemento chiave del sistema di calcolo distribuito Hadoop**, *è il responsabile dell'elaborazione dei dati*.
- Framework per la creazione di **applicazioni che elaborano dati in parallelo** secondo il principio di ***functional programming***.
- MapReduce lavora secondo il principio ***dividi et impera***.

Un'operazione di calcolo è **suddivisa in sottoparti** ognuna delle quali è **processata in modo autonomo**. Quando ogni parte del problema è stata calcolata ***i risultati parziali vengono riassemblati o "ridotti".***

Hadoop - MapReduce

- *MapReduce* si occupa automaticamente della ***suddivisione dei vari task***, del ***monitoraggio*** e della loro ***ri-esecuzione*** in caso di malfunzionamenti.

- Questo framework **lavora in congiunzione con HDFS**, infatti i *nodi di calcolo si trovano in corrispondenza dei DataNode*.

- I vari *task sono quindi eseguiti li dove fisicamente sono memorizzati i dati*.

Aumento dell'efficienza di calcolo

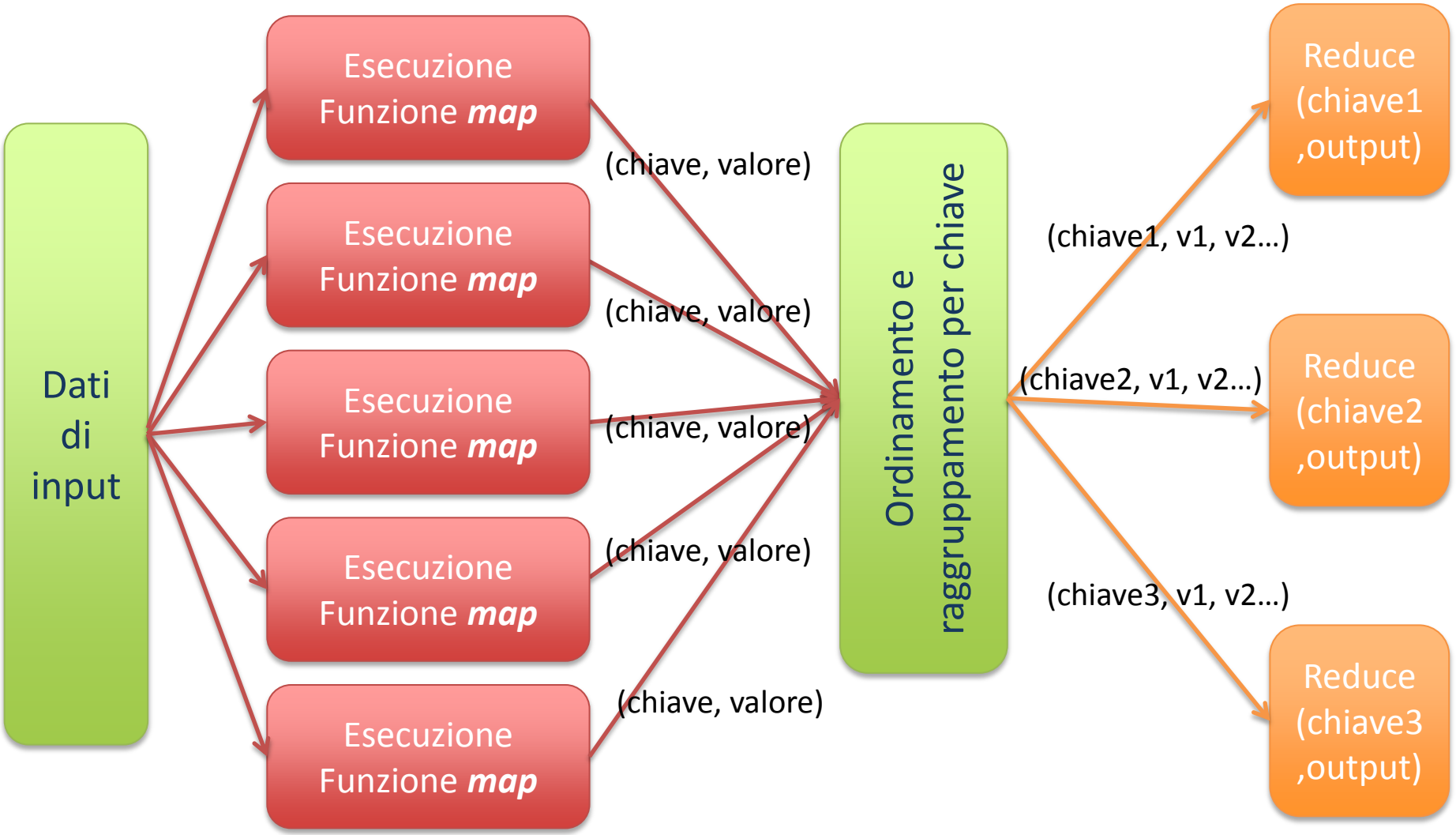
MapReduce - Job

In MapReduce le elaborazioni sono definite ***"job"***, ognuno dei quali è composto da:

- I ***dati di ingresso***, presenti du HDFS.
- *Una funzione* ***map***, che converte i dati in un insieme di coppie *chiave/valore.*
- *Una funzione* ***reduce***, che elabora i valori associati a ciascuna chiave e crea in output una o più coppie chiave valore.
- Il **risultato**, che viene scritto su un file su HDFS.

Osservazione: prima di eseguire la funzione *reduce*, c'è una fase in cui ***le coppie chiave valore sono ordinate*** per chiave e ***i valori appartenenti alla stessa chiave, sono raggruppati***.

MapReduce - Job



MapReduce - Componenti

Il *framework MapReduce* è costituito da **due componenti chiave**:

1. ***JobTracker –*** l'elemento che si occupa di **gestire le risorse** (CPU e memoria) e il **ciclo di vita del job**. Si occupa di **distribuire i vari task** tra i nodi (secondo un criterio di vicinanza dei dati) e di **rieseguirli nel caso in cui si verificassero degli errori**.

1. ***TaskTracker –*** sono gli elementi che **girano sui vari nodi** del cluster **responsabili dell'esecuzione dei task** sotto le direttive del JobTracker.

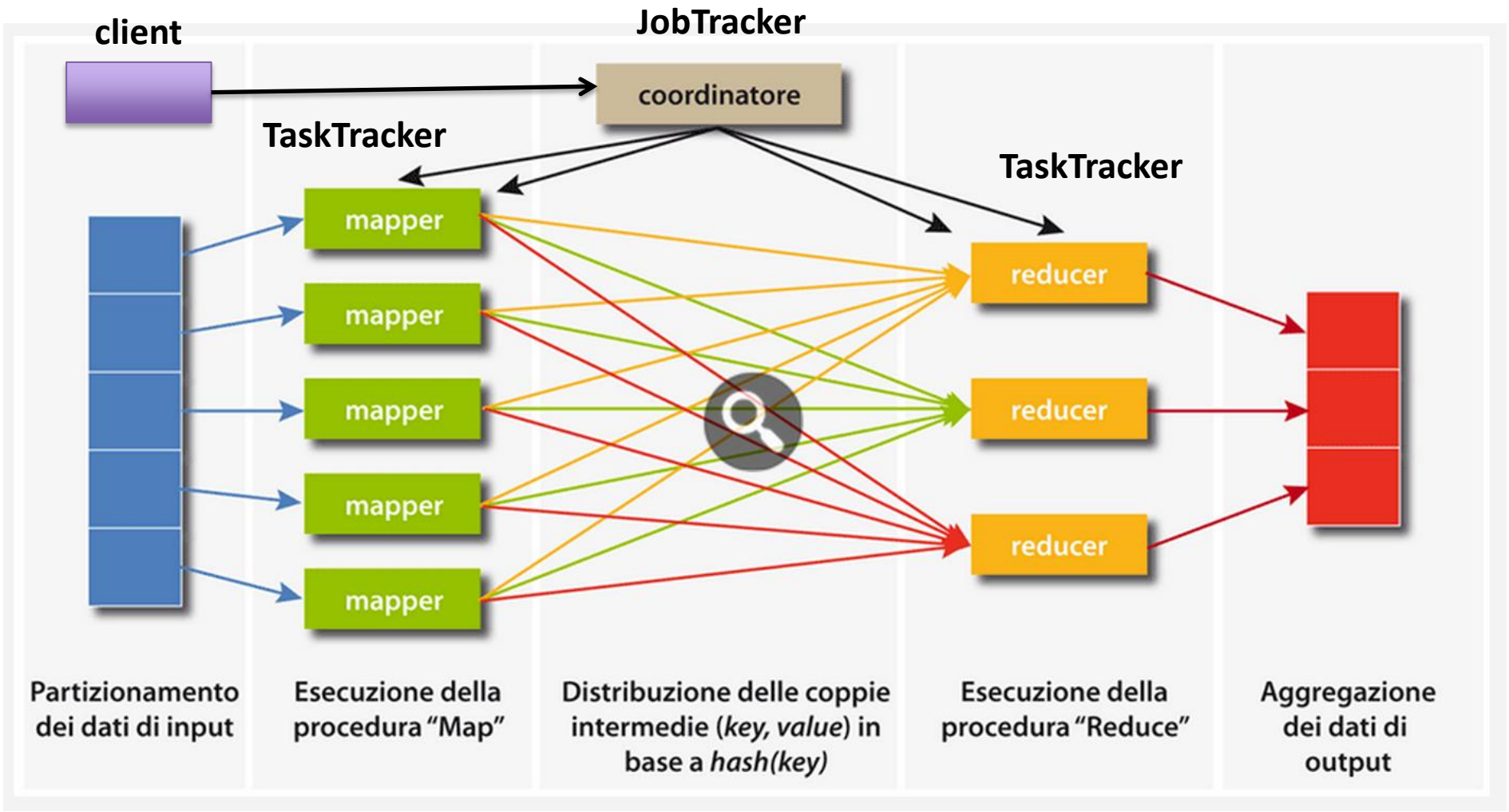
MapReduce - Funzionamento

Le applicazioni che sfruttano MR devono specificare: ***file di input***, ***file di output***, ***funzioni di map*** e ***reduce*** (contenute nel job).

Il client Hadoop comunica il job (come archivio .jar) al JobTracker, che si occupa di distribuire l'esecuzione sui vari nodi.

- Il **JobTracker** determina il **numero di parti in cui è suddiviso l'input**, attivando un certo **numero di TaskTracker** in base alla vicinanza.
- I **TaskTracker estraggono i dati e attivano la funziona *map*** che genera ***coppie chiave/valore***.
- Finita la fase di map, il **JobTracker attiva la fase di reduce** in cui i TaskTracker prima ordinano i risultati per chiave e poi li "riducono".
- Infine i vari **file di output prodotti saranno aggregati** in un unico risultato.

MapReduce - Funzionamento



MapReduce – Scrivere un Job

Scrivere un **Job MapReduce** non è un'operazione banale; consiste infatti nella creazione di **tre classi Java**: ***mapper***, ***reducer*** e ***driver***.

mapper

- Questa classe estende la classe base *MapReduceBase* e implementa l'interfaccia *Mapper.*
- La maggior parte del lavoro è svolto dal metodo ***map.***

reducer

- Questa classe estende la classe base *MapReduceBase* e implementa ovviamente l'interfaccia *Reducer.*
- La maggior parte del lavoro è svolto dal metodo ***reduce.***

MapReduce – Scrivere un Job

Scrivere un **Job MapReduce** non è un'operazione banale; consiste infatti nella creazione di **tre classi Java**: ***mapper***, ***reducer*** e ***driver***.

driver

- Questa classe ha il **compito di inizializzare il job**, di **passare i parametri di input** e **definire la posizione in cui sarà memorizzato l'output**.

OSSERVAZIONE – vi è un'ulteriore classe che si va a collocare nel workflow tra i *mapper* e i *reducer*, la classe ***combiner.***

- Questa classe è **simile a reducer**, ma limitatamente ad un solo nodo; cioè **si occupa di aggregare i dati elaborati su un nodo prima di passarli ad un altro** con l'obiettivo di ridurre il traffico di rete.

Hadoop – Installazione

Un cluster Hadoop può essere installato in **tre differenti modalità**:

1. **Modalità Standalone (Locale)**

è la configurazione di default, Hadoop consiste in un singolo processo Java in esecuzione.

2. **Modalità Pseudo-Distribuita**

Il cluster è installato su un singolo nodo su cui sono eseguiti più daemon Hadoop, ognuno su uno specifico processo Java.

3. **Modalità Fully-Distributed**

E' la configurazione più interessante, il cluster è costituito da un insieme di nodi fisici (collegati in rete) su ciascuno dei quali è eseguito un processo Hadoop.

HBase

- **Hbase** è un ***NoSQL database*** di tipo ***column-oriented***, ispirato a BigTable.
- Fa parte **dell'ecosistema Hadoop** (è infatti un progetto Apache).
- Integrazione con Hadoop elevata, fa **uso di HDFS** per la ***persistenza dei dati.***

Il ***modello dati*** di Hbase si basa sui **concetti** di ***tabella*** e di ***column family***.

Hbase – Modello dati

- Una ***Tabella*** è composta da una o più ***column family***.
- Una ***column family*** solitamente contiene ***più colonne***.
- **Ogni riga** all'interno di una tabella è identificata da un ***rowId.***
- I ***valori*** sono memorizzati in ***celle***, identificate dalla tripla **(row – column - version)**. *Version è un timestamp*.
- Per definire il nome di una colonna si utilizza la seguente sintassi:

nome_column_family:nome_colonna

Hbase – Operazioni sul modello dati

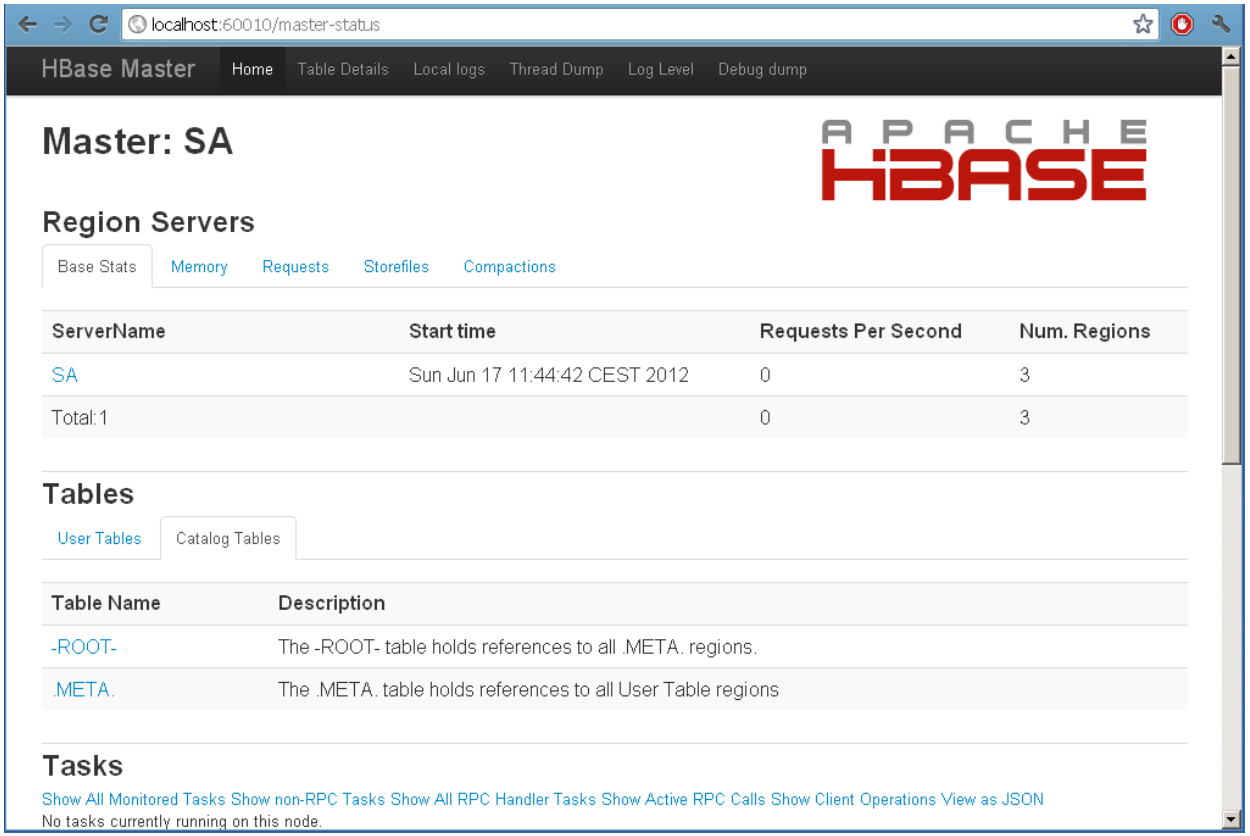
Alcune delle **possibili operazioni** eseguibili su questo **modello dati** sono:

- ***Put –*** Inserimento di una nuova riga (o aggiornamento).
- ***Get*** – estrazione dei valori da una singola riga.
- ***Scan –*** Estrazione di valori da più righe.
- ***Delete –*** Cancellazione di una riga.
- ***Disable –*** Disabilitazione di una tabella.

Hbase – Interfaccia

L'interazione con Hbase può avvenire:

- Tramite ***riga di comando*** (utilizzo della shell)
- ***Interfaccia web*** (http://ip_server_hbase:60010)



Hbase – Architettura

Hbase nasce con l'idea di implementare un **DB distribuito basato su HDFS**.

- Sul **NameNode** è presente l'istanza del ***servizio Master***.

*Il Master ha il compito di **monitorare i RegionServer** e di **gestire e controllare i cambiamenti dei metadati**.*

- Sui **DataNode** sono presenti le istanze dei ***servizi RegionServer***.

*I RegionServer gestiscono le **region**, che sono elementi che si occupano della **disponibilità e della distribuzione delle tabelle**.*

Hbase – Architettura

Hbase nasce con l'idea di implementare un **DB distribuito basato su HDFS**.

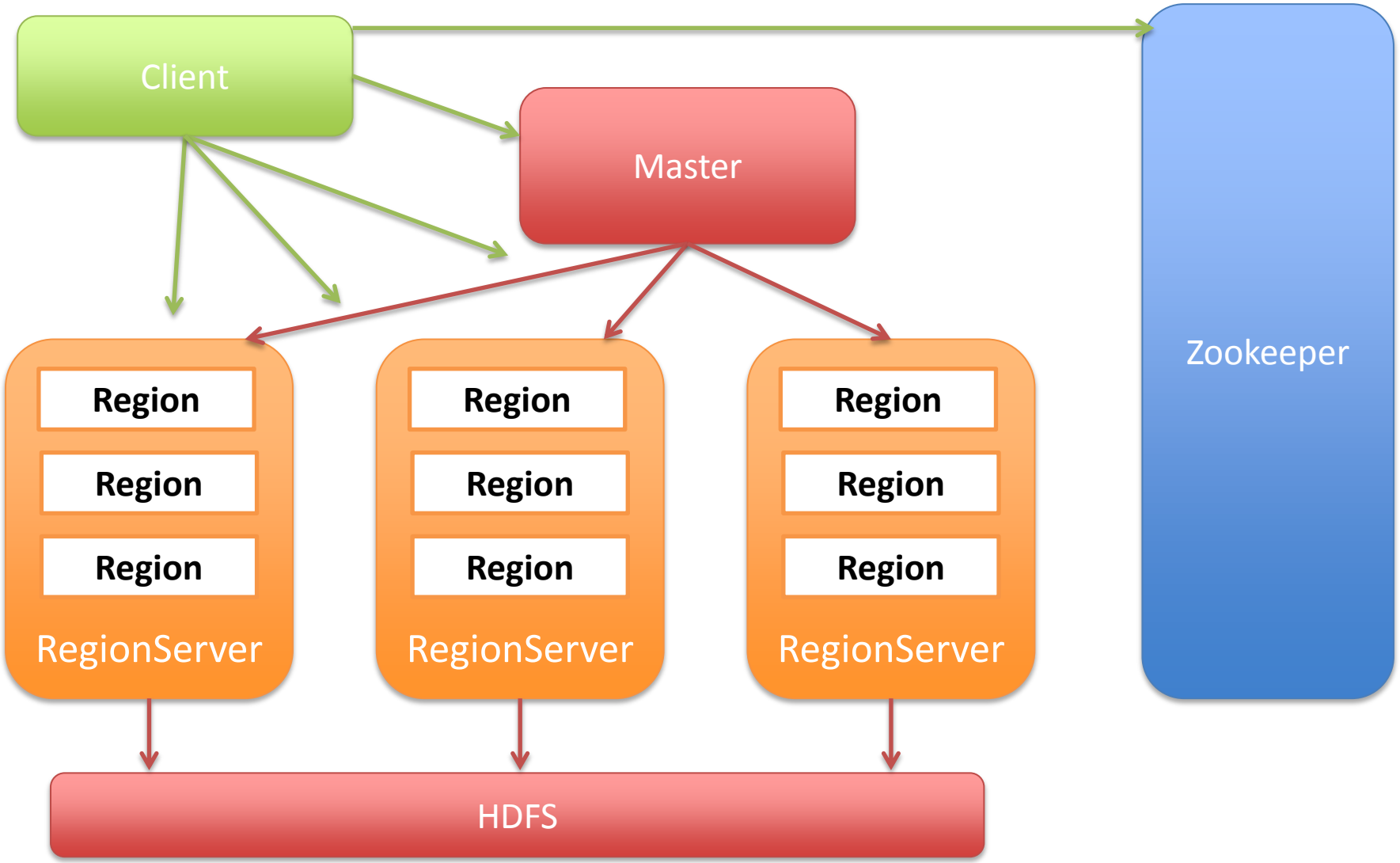
- Sul **NameNode** è presente l'istanza del ***servizio Master***.

*Il Master ha il compito di **monitorare i RegionServer** e di **gestire e controllare i cambiamenti dei metadati**.*

- Sui **DataNode** sono presenti le istanze dei ***servizi RegionServer***.

*I RegionServer gestiscono le **region**, che sono elementi che si occupano della **disponibilità e della distribuzione delle tabelle**.*

Hbase – Architettura



Hbase – Interazione

- Lo strumento principale per interagire con HBase è lo script ***hbase***, che si trova nella cartella ***bin*** dell'installazione.

- Il comando ***shell*** è quello che consente di interagire con il modello dati e di eseguire operazioni si gestione del cluster.

Comando	Descrizione	Esempio
create	Crea una tabella, specificando la colum family	*create 'tab_studenti', 'anagrafica'*
list	Restituisce l'elenco delle tabelle, si può filtrare usando delle reg exp	*list '^t'*
put	Inserisce una nuova riga	*put 'tab_studenti', 'stud1', 'anagrafica:nome', 'Gino'* *put 'tab_studenti', 'stud1', 'anagrafica:cognonome', 'Rossi'*

Hbase – Interazione

Comando	Descrizione	Esempio
scan	Restituisce un insieme di righe di una tabella	*scan 'tab_studenti', {COLUMNS => ['anagrafica:nome'], LIMIT => 10}*
get	Restituisce una riga	*get 'tab_studenti', 'stud1'*
delete	Elimina il valore di una cella	*delete 'tab_studenti', 'stud1', 'anagrafica:cognome'*
disable	Disabilita una tabella	*disable 'tab_studenti'*
drop	Elimina una tabella	*drop 'tab_studenti'*

Hbase – Utilizzo da Shell

1. Connessione ad Hbase

I comandi per interfacciarsi con HBase via shell si trovano nella cartella ***bin/***. Per attivare la shell di HBase lanciare da terminale il seguente comando.

```
$ bin/ ./hbase shell
hbase(main):001:0>
```

2. Creazione di una Tabella

Per la creazione di una tabella va utilizzato il comando ***create***, e va poi specificato il nome della tabella e della ColumnFamily.

```
$ hbase> create 'persone', 'dati'
0 row(s) in 1.2200 seconds
```

3. Informazioni su una Tabella

Per ottenere informazioni su una tabella utilizzare il comando ***list.*** Se non si specifica il nome della tabella si avrà l'elenco di tutte le tabelle presenti nel DB.

```
hbase> list 'persone'
TABLE
persone
1 row(s) in 0.0350 seconds
```

4. Popolare la Tabella

Per inserire valori all'interno di una tabella utilizzare il comando ***put***

```
hbase> put 'persone', 'row1', 'dati:nome', 'gino'
0 row(s) in 0.1770 seconds

hbase> put 'persone', 'row1', 'dati:cognome', 'rossi'
0 row(s) in 0.0160 seconds

hbase> put 'persone', 'row2', 'dati:cognome', 'verdi'
0 row(s) in 0.0260 seconds

hbase> put 'persone', 'row2', 'dati:professione', 'musicista'
0 row(s) in 0.0350 seconds
```

5. Scansione della Tabella

La scansione di una tabella può essere fatta mediante il comando ***scan.*** Si può decidere si limitare il numero di risultati da visualizzare oppure ottenere la visualizzazione completa.

```
hbase> scan 'persone'
 ROW          COLUMN+CELL
 row1         column=dati:nome, timestamp=140375941, value=gino
 row1         column=dati:cognome, timestamp=140375943, value=rossi
 row2         column=dati:cognome, timestamp=140375950, value=verdi
 row2         column=dati:professione, timestamp=140375950, value=musicista
 4 row(s) in 0.0440 seconds
```

6. Ottenere i dati di una singola riga

In questo caso utilizzare il comando ***get***, specificando oltre al nome della tabella anche quello della riga.

```
hbase> get 'persone', 'row1'
COLUMN                          CELL
dati:nome                       timestamp=1403759475114, value=gino
dati:cognome                    timestamp=1403759475218, value=rossi
2 row(s) in 0.0230 seconds
```

7. Disabilitare e cancellare una tabella

Per disabilitare una tabella utilizzare il comando ***disable***, per cancellarla utilizzare il comando ***drop***.

```
hbase> disable 'persone'
0 row(s) in 1.6270 seconds

hbase> drop 'persone'
0 row(s) in 0.2900 seconds
```

8. Per uscire dalla shell HBase utilizzare il comando *quit*

Pentaho Data Integration (PDI)

- **Pentaho** è un framework che contiene diversi pacchetti tra loro integrati che consentono la gestione completa:

- ***Problematiche della Business Intelligence***
- ***Problematiche dei Data Warehouse.***
- ***Problematiche legate ai Big Data.***

- **Kettle** e' la ***componente ETL*** di Pentaho, ovvero si occupa del trasferimento e della trasformazione dei dati.

Pentaho Data Integration (PDI)

ETL (*Extract, Transform e Load*)

- E' un processo utilizzato per lavorare con database e data warehouse che si compone di tre fasi.

1. ***Estrazione*** *dei dati dalle sorgenti di informazioni.*

2. ***Trasformazione*** *dei dati per adattarli alle esigenze operazionali e migliorarne la qualità.*

3. ***Caricamento*** *dei dati in sistemi di sintesi (database operazionali, repository, data warehouse o data mart…)*

I **tool ETL** sono molto utili **per preparare i dati** raccolti alla successiva **fase di analisi** ed eventuale traduzione in RDF.

Kettle – Caratteristiche principali

- **Kettle è sviluppato in Java**, garantisce quindi la *compatibilità e portabilità con i principali sistemi operativi* (Windows, Linux, OS X..).

- E' disponibile sia in **versione open source** che **enterprise**.

- Offre la possibilità di **interfacciarsi con i principali Database di tipo NoSQL** (Hbase, Cassandra, MongoDB, CouchDB…).

- Sviluppo e configurazione di procedure realizzato tramite **interfaccia grafica con un approccio drag&drop** delle componenti necessarie.

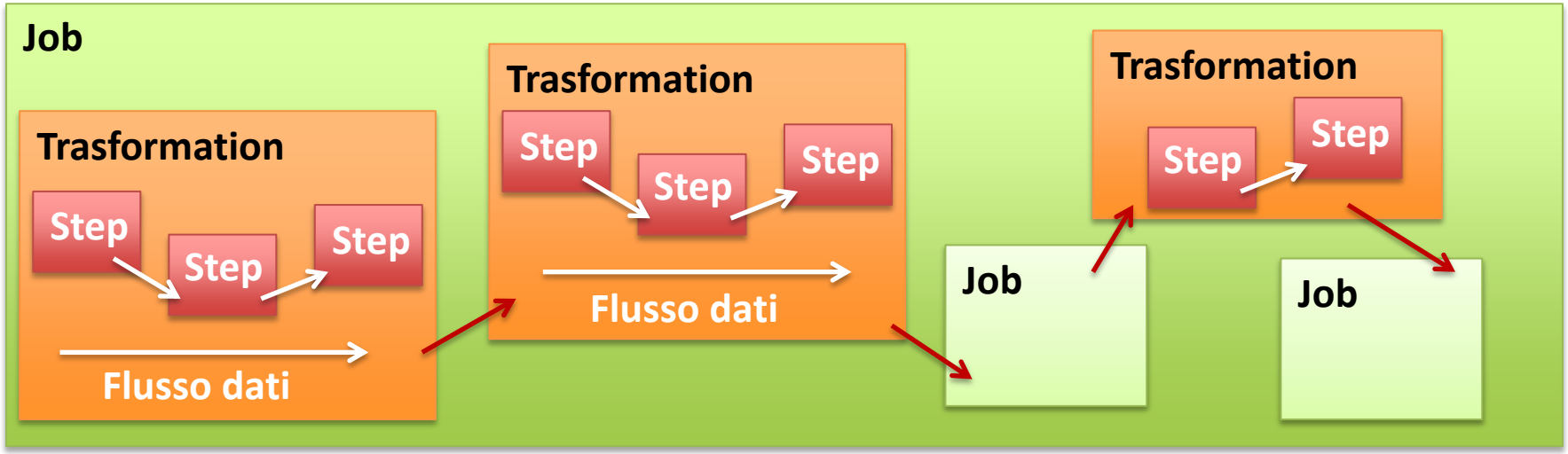
(-) *Kettle non è in grado di trasformare i dati elaborati in triple RDF*, il che implica la necessità di utilizzare altri strumenti in una fase successiva come ad esempio Karma.

Kettle – Dettagli

- Dal punto di vista del funzionamento Kettle è basato sui concetti chiave:
  - **Job** (file con estensione .kjb)
  - **Trasformazione** (con estensione .ktr), composta da diversi ***step***.
- I componenti principali di Kettle sono:
  - **Spoon** – Editor grafico per la modellazione dei processi ETL.
  - **Pan** – Esecuzione da riga di comando delle trasformazioni.
  - **Kitchen** – Esecuzione da riga di comando dei Job.

Kettle – Struttura operativa

I **componenti** operativi di **Kettle** sono **organizzati** nel seguente modo:



- I **dati sono visti come flusso di righe** da uno step all'altro.
- Gli **step** sono eseguiti **parallelamente in thread separati**, *non c'è obbligatoriamente un punto di inizio o di fine di una trasformazione*.
- I **job gestiscono l'esecuzione sequenziale** delle entità di livello inferiore, *trasformazioni o altri job*.