Adatbázis-kezelés I.

NOSQL

Fogarassyné Vathy Ágnes Pannon Egyetem

Motivációs példák

- log fájlok feldolgozása
  - adatbetöltés nem szükséges
  - egyszerű matching
- Wikipedia oldalak
  - félig-strukturált szöveg
  - keresés a szövegben
- szociális hálózatok – pl. Facebook
  - Ki kinek hányadik ismerőse?
  - konzisztencia nem annyira fontos

Hagyományos RDBMS rendszerek tulajdonságai

A hagyományos RDBMS rendszerek…

- használata kényelmes
  - egyszerű adatmodell
    - *Minden adat betölthető RDBMS rendszerekbe?*
      - *Ha nem, akkor előkészítés szükséges.*
  - deklaratív lekérdező nyelv (egyszerű SQL)
    - *Mindig kihasználjuk a szolgáltatásait?*
      - *Például: kulcs szerinti kereséshez nem kell összegezni, vagy összekapcsolni.*
  - tranzakciókezelés (többfelhasználós támogatás, rollback,…)
    - *Mindig szükséges ilyen erős megkötés a konzisztenciára?*
      - *Nem minden esetben, néha fontosabb a sebesség, s relaxált konzisztencia megvalósítása is elegendő.*

Hagyományos RDBMS rendszerek tulajdonságai

A hagyományos RDBMS rendszerek…

- biztonságosak (felhasználói jogosultságrendszer, …)
  - *NoSQL rendszerekben az adatfeldolgozás gyakran offline módon történik*
    - *így a biztonság kevésbé fontos kritérium*
- tartós adattárolást biztosítanak
  - Hasonlóan fontos a NoSQL rendszerekben is

Hagyományos RDBMS rendszerek tulajdonságai

- Masszívak?
  - *Az emberek manapság sokkal több adatot tárolnak/kezelnek, mint amire a hagyományos DBMS rendszerek ki lettek fejlesztve.*
- Hatékonyak?
  - *Mit jelent manapság ás korábban a hatékonyság?*
    - 1 másodperc alatt millió/billió kérdések bombázza a szervereket - például: facebook, twitter

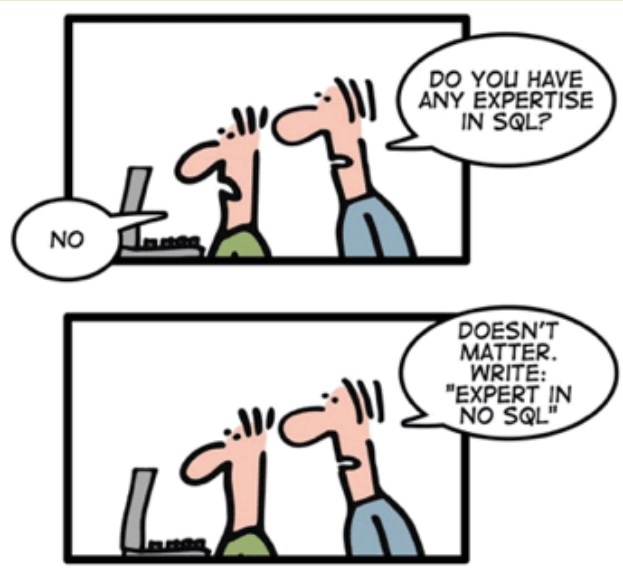
Motiváció

- A hagyományos RDBMS rendszerek nem képesek *minden* adatkezelési és adatelemzési problémát hatékonyan megoldani.

- Big Data
  - 2,5 exabyte ($2{,}5*10^{18}$) új adat / nap (2015)
  - a tárolt adatok 90% az elmúlt 2 évből származik

- NoSQL = „Not only SQL”,
  - ahol az SQL nem az SQL nyelvet, hanem inkább a hagyományos relációs DBMS-eket jelöli.

  - „NoSQL” ≠ „Don’t use SQL language”

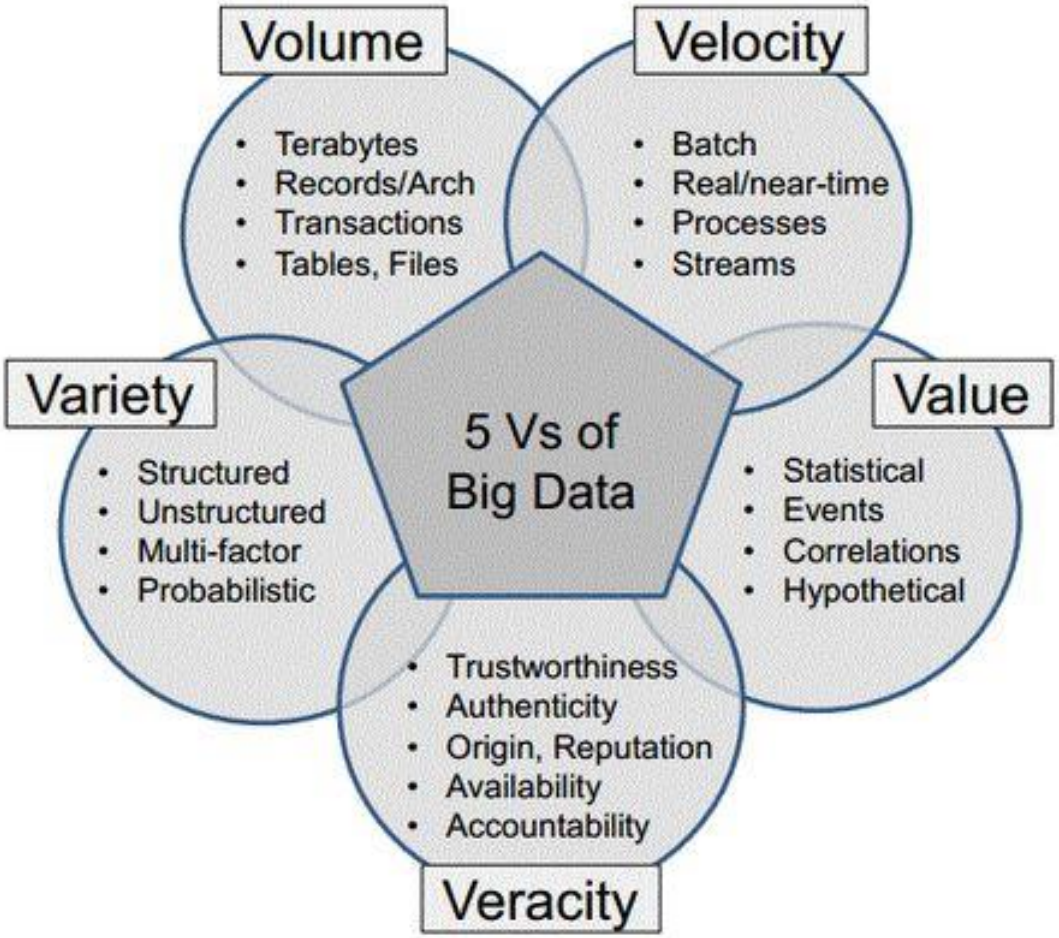


Big Data

- 2,5 exabyte ($2{,}5*10^{18}$ = 2,5 milliárd GB) új adat / nap (2015, IBM)
- a tárolt adatok 90% az elmúlt 2 évből származik

- A Big Data-t meghatározó **5V**:
  - Volume: óriási adatmennyiség keletkezik
  - Velocity: az adatok gyorsan jönnek létre és gyorsan változnak – nagy sebességű adatfeldolgozás szükséges
  - Variety: adatok változatossága – sokféle forrásból származnak, eltérő strukturáltságúak
  - Veracity: Adatok megbízhatósága kérdéses (gyakran zajosak, nem ellenőrzött adatok)
  - Value: a nagy adathalmazokban óriási érték, tudás rejtőzik – üzleti haszon

Big Data – 5 V


Volume
Terabytes
Records/Arch
Transactions
Tables, Files
Velocity
Batch
Real/near-time
Processes
Streams
Variety
Structured
Unstructured
Multi-factor
Probabilistic
5 Vs of Big Data
Value
Statistical
Events
Correlations
Hypothetical
Trustworthiness
Authenticity
Origin, Reputation
Availability
Accountability
Veracity

NoSQL jellemzői

- Főbb jellemzők
  - Jellemzően nem relációs adatmodell
  - Elosztott működés
  - Horizontális skálázhatóság
  - Nyílt forráskód
    - Vitatott, nem minden esetben teljesül, pl. Google BigTable
- További jellemzők:
  - Sémamentes, vagy gyengén strukturált
  - Replikáció támogatása
  - Egyszerű alkalmazásprogramozási interfész (API)
  - Fokozott konzisztencia

CAP

Elosztott rendszerekkel szemben megfogalmazott követelmények:

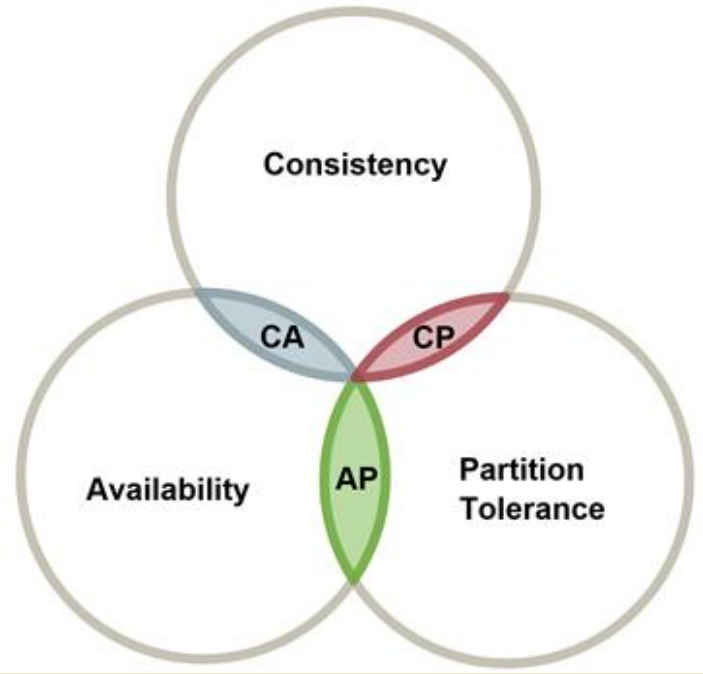
- Konzisztencia (**C**onsistency):
  - Egy elosztott rendszer akkor konzisztens, ha bármely csomópontjából bármely időpontban ugyanazon információt kapja vissza a kliens.
- Rendelkezésre állás (**A**vailability):
  - Egy elosztott rendszer rendelkezésre áll, ha minden működő csomóponthoz érkező kérésre nem hibaüzenettel válaszol, tehát a csomópontokon futtatott algoritmusoknak véges idő alatt be kell fejeződniük.
- Partíció tolerancia (**P**artition tolerance):
  - partíció: szerverek egy halmaza
  - A rendszer partíciótoleráns, ha akkor is helyesen működik, ha a partíciók között tetszőleges számú üzenet elveszik, vagy ha egy partíció meghibásodik.
  - Partíció kiesését követően az adatokat szinkronba hozza, hogy helyesen működjön az adatbázis.

http://weblabor.hu/cikkek/nosql-bevezeto

CAP tétel

- Sejtés: Eric Brewer, 2000
- Tétel: Nancy Lynch, Seth Gilbert, 2002

- *CAP tétel* : Elosztott rendszerben nem garantálható mindhárom CAP tulajdonság.
  - Consistency + Availability
    - pl: a rendszert egy gépen futtatjuk
  - Consistency + Partition Tolerance
    - pl.: hálózati partíció esetén elérhetetlenné válnak bizonyos adategységek
  - Availability + Partition Tolerance
    - konzisztencia gyengítésével elérhető
      pl.: bármit visszaadhat



CAP tétel kritikája

- Nem érvényes az alkalmazáshibák, az adatbáziskezelő összeomlását okozó tranzakciók esetén.
- A CAP tétel nem beszél teljesítményről.

- Teljesítménymértékek:
  - Áteresztőképesség [adategység/sec]
  - Késleltetés [sec]

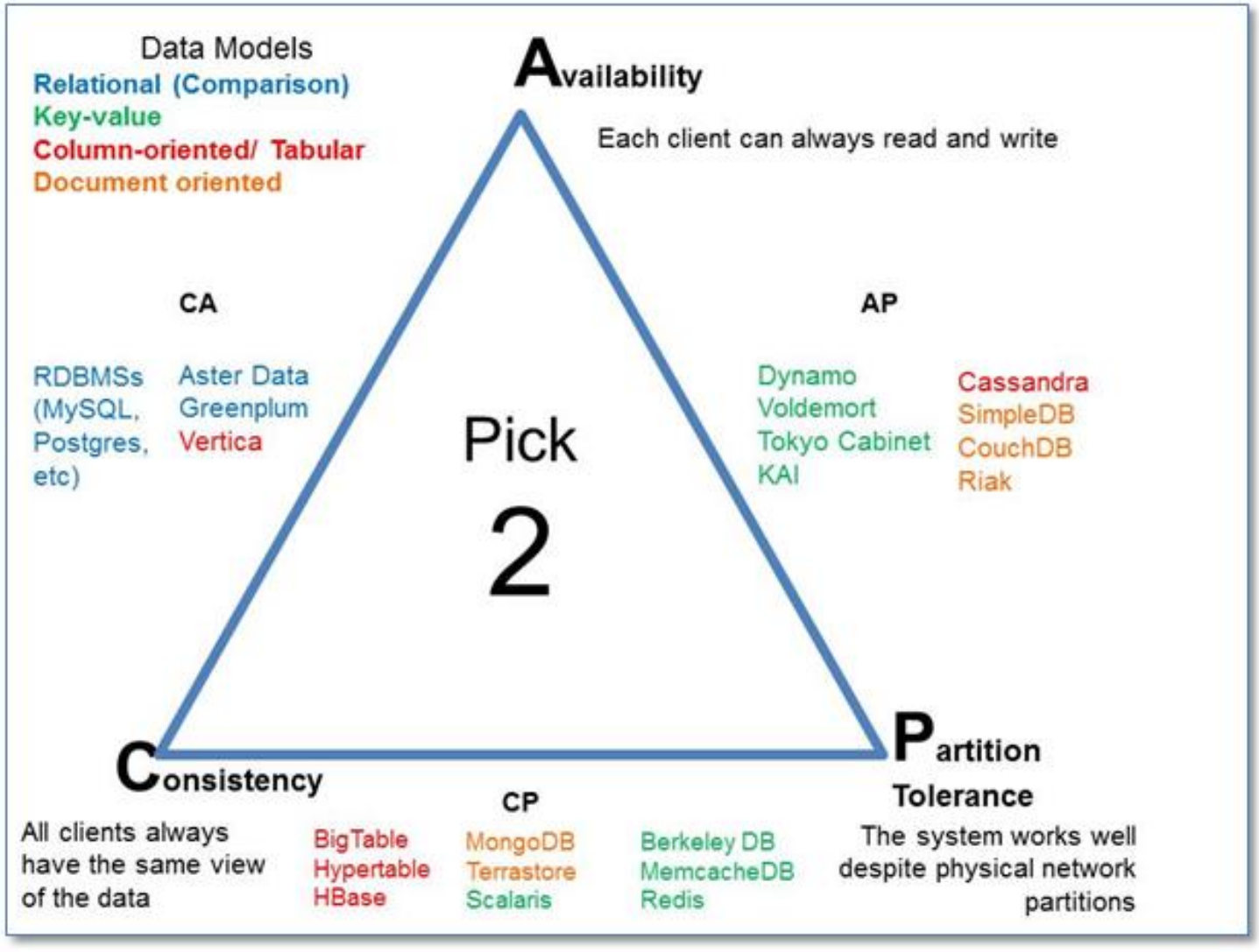
A teljesítmény jelentősége

- A késleltetés kiemelt szerepű:
  - Kísérletek kimutatták, hogy az Amazonnál a lapbetöltési idő minden 100 ms-os növekedése 1%-kal csökkentette az eladásokat.
  - A Google-nél a keresési találatok megjelenési idejének fél másodperces növekedése 20%-os bevételcsökkenést okozott.

Konzisztencia gyengítésével a késleltetés csökkenthető

- A késleltetés csökkentésének érdekében sok NoSQL rendszer (pl. az Amazon Dynamo) összefüggő hálózaton futtatva sem garantál atomi konzisztenciát, cserébe a rendszer hálózati partíció kialakulása esetén is elérhető marad.

CAP tétel – rendszerek



SQL vs. NoSQL rendszerek

- Mikor használjunk SQL alapú rendszert?
  - Fontos az ACID elvek biztosítása.
  - Az adathalmaz nem növekszik nagyon nagy mértékben.
  - Strukturált adatok.

- Mikor használjunk NoSQL rendszert?
  - Hatalmas, gyorsan változó adatmennyiség
    - Hatékony feldolgozási igény
  - Változó, vagy sémamentes struktúra
  - Cloud computing támogatott

A NoSQL rendszerek típusai

A NoSQL rendszerek típusai

- Kulcs-érték tárolók (Key-value stores)
  - adatreprezentáció: kulcs-érték párok
- Oszlopcsaládok (Column stores – Column Families)
  - oszlop-orientált hierarchikus adattárolás
- Dokumentum tárolók (Document databases)
  - dokumentumok tárolása (pl: XML, JSON, BSON, PDF, MS Word, YAML, …)
- Gráf adatbázisok (Graph databases)
  - gráfok tárolása
- …

Kulcs-érték tárolók

- Egyszerű adatbázis-kezelők:
  - A séma nélküli adatok tárolására fejlesztették ki
  - Adatmodell: (kulcs, érték) párok
    - Objektumokat tartalmaz (érték), amelyek egyedi kulcs által vannak azonosítva.
    - kulcs: generált, vagy adott adat
    - érték: tetszőleges adattípus lehet, pl.: sztring, integer, lista, halmaz, JSON, BLOB (basic large object) etc.
  - Az ötletet számos más NoSQL rendszer is alkalmazza (pl. oszlopcsaládok, gráfadatbázisok).

Például: *BerkeleyDB, Redis, Riak, Amazon's Dynamo, Cassandra, Project Voldemort, Azure Table Storage (ATS), Memcached, …*

Kulcs-érték tárolók – példa

Key	Value
“India”	{“B-25, Sector-58, Noida, India – 201301"}
“Romania”	{“IMPS Moara Business Center, Buftea No. 1, Cluj-Napoca, 400606",City Business Center, Coriolan Brediceanu No. 10, Building B, Timisoara, 300011"}
“US”	{“3975 Fair Ridge Drive. Suite 200 South, Fairfax, VA 22033"}

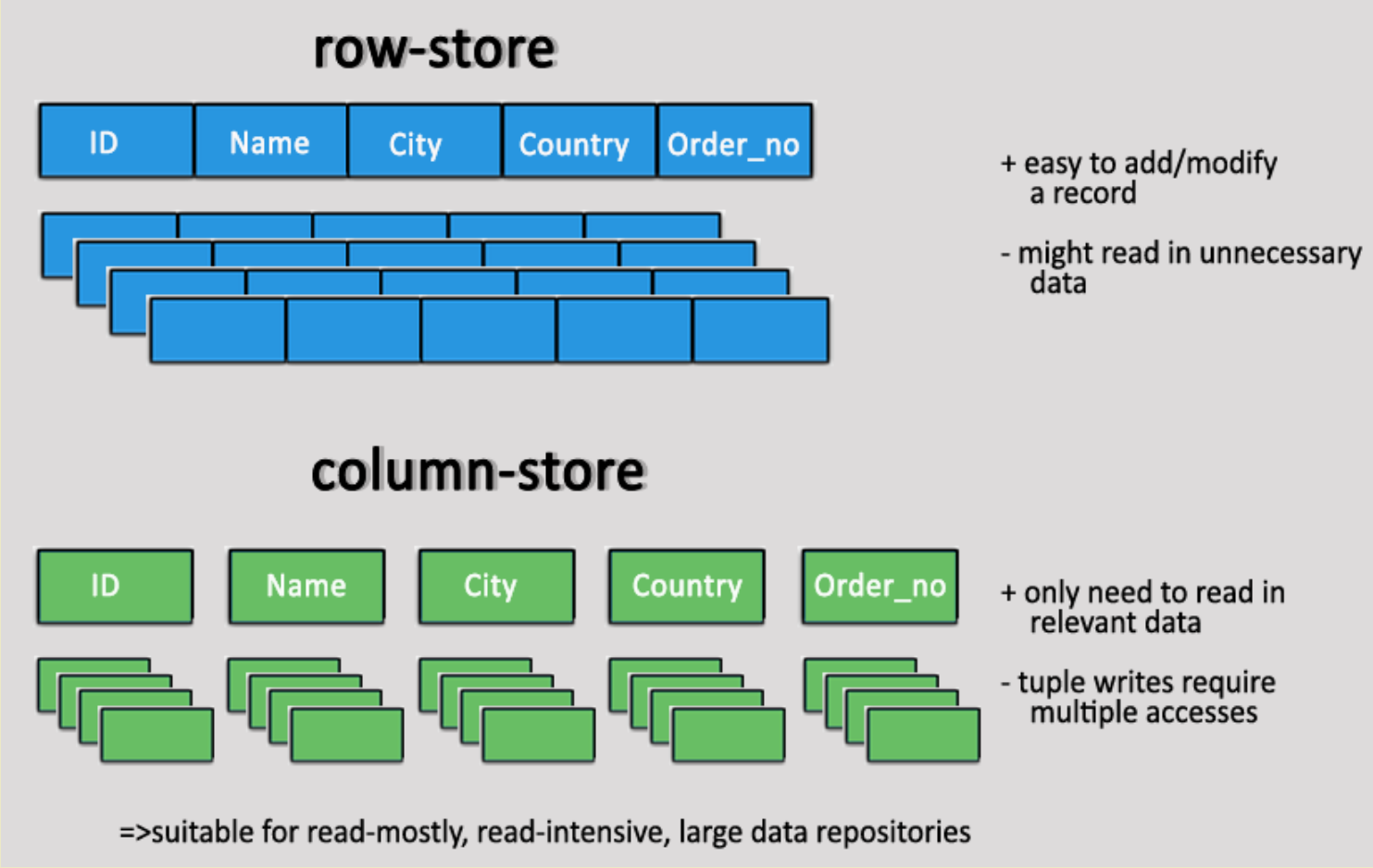
Kulcs-érték tárolók

- Műveletek:
  - Insert (key, value): új kulcs-érték beszúrása (egy adat hozzárendelése egy kulcshoz)
  - Fetch (key): adott kulcshoz tartozó értékek keresése
  - Update (key, value): adott kulcshoz tartozó érték módosítása
  - Delete (key): adott kulcshoz tartozó érték törlése

Oszlop alapú tárolás

- Hagyományos RDBMS rendszerek
  - soralapú tárolás: a rekordokat fizikailag sorrendbe tárolják
    - Előnye:
      - 1 sor beszúrása vagy módosítása
    - Hátránya:
      - Kevés attribútumot érintő lekérdezés: felesleges adatolvasások
- Oszlopalapú tárolás RDBMS rendszerekben:
  - oszlopalapú szervezés (’80-as évek)
    - Előnye:
      - Kevés oszlopot érintő lekérdezések (pl. összegzések, analitikus adattárházak)
    - Hátránya:
      - Kevés sort érintő lekérdezések: felesleges adatolvasások
  - Például: Sybase IQ, Vertica

Sor- vs. oszlopalapú tárolás



NoSQL oszlopcsaládok

- NoSQL oszlopcsaládok:
  - egy hierarchikus – nem relációs – adatmodell szerint, logikailag oszlopalapon szervezik az adatokat
    - Eltérően az oszlopalapú RDBMS rendszerektől, melyek az adatok fizikai tárolását valósítják meg oszlopalapon.

Például: Google's *Hbase, Cassandra, BigTable, Amazon SimpleDB …*

Oszlopcsaládok adatmodell

- Sorai: kulcs–érték párok
  - kulcs: attribútum neve – érték: attribútum értéke
- Oszlopcsalád: kulcs – érték párok, ahol az értékek az előzőekben leírt soroknak felelnek meg.

kulcs	oszlopkulcs0	oszlopkulcs1	...	oszlopkulcsN
	érték0	érték1	...	értékN

- Szuperoszlopcsalád: egy újabb kulcs-érték pár szinttel bővíti felfelé az oszlopcsaládok szintjét

kulcs	szuperoszlop kulcs0				...	szuperoszlop kulcsN			
	oszlop kulcs0	oszlop kulcs1	...	oszlop kulcsN		oszlop kulcs0	oszlop kulcs1	...	oszlop kulcsN
	érték0	érték1	...	értékN		érték0	érték1	...	értékN

Hbase Table – példa

- *Hbase tábla: User*
- *Oszlopcsaládok: Account, Personal, Friends*
- *Oszlopok az „Account” oszlopcsaládban: username, password*

User			
	Account	Personal	Friends
1001	username = “bob1987” password = “thisisapassword”	firstName = “Bob” lastName = “Smith” ssn = “4hfe94”	2004:firstName = “Alice” 2004:lastName = “Smith” 2004:email = “alice@gmail.com” 1714:firstName = “Charlie” ...
2004	username = “alice”	...	...
1714	...	...	...

Dokumentum tárolók

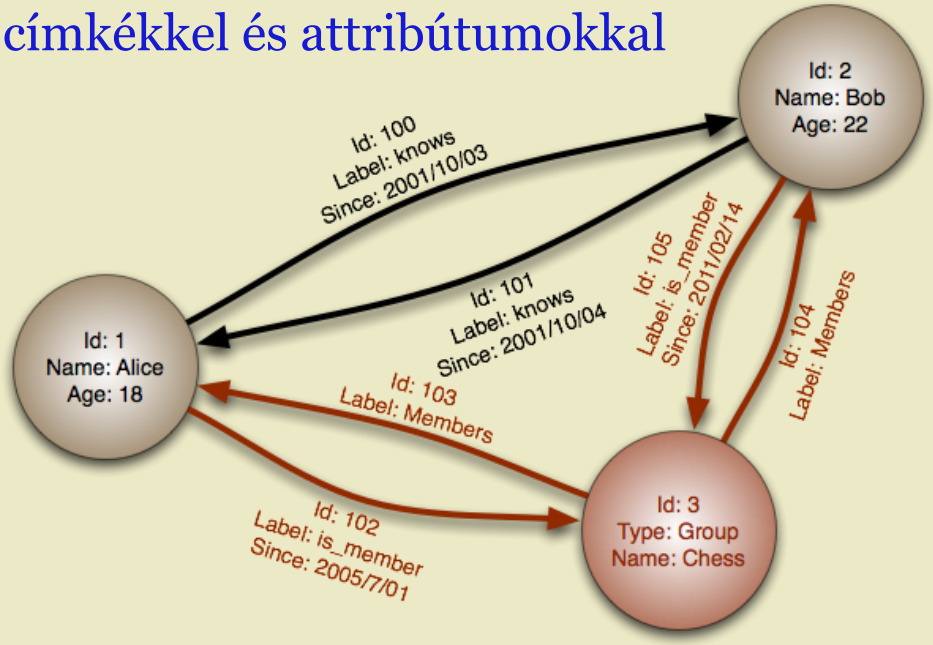
- Hasonló a kulcs érték tárolókhoz, DE: az érték egy dokumentum, amely struktúrával rendelkezik
  - Dokumentum:
    - Ismert ÉS laza struktúra
    - Pl: JSON, XML, egyéb féligstrukturált formátumok
- Műveletek:
  - Insert (key, document), Fetch (key), Update (key), Delete (key)
  - + Fetch a dokumentum tartalma alapján
- Például: *CouchDB, MongoDB, SimpleDB, Terrastore, …*

Példa

```
users: [
    {
        _id: "1001",
        username: "bob1987",
        password: "thisisapassword"
        personal: {
            firstName: "Bob",
            lastName: "Smith",
            ssn: "4hfe94"
        },
        friends: [
            {
                id: "2004",
                firstName: "Alice",
                lastName: "Smith",
                email: "alice@gmail.com"
            },
            {
                id: "1714",
                firstName: "Charlie",
                ...
            },
            ...
        ]
    },
    {
        _id: "2004",
        username: "alice",
        ...
    }
    {
        _id: "1714",
        ...
    }
    ...
]
```

Gráf adatbázisok

- Adatmodell: csúcsok és élek
  - A csúcsok rendelkezhetnek attribútumokkal (gyakran ID)
  - Az élek rendelkezhetnek címkékkel és attribútumokkal



- Lekérdezések pl:
  - egy lépéses távolság
  - teljes rekurzió
  - útvonal megadása

Például: *Neo4j, Pregel, FlockDB, HypergrahpDB, AllegroGraph, InfiniteGraph, …*

Összehasonlítás

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–Value Store	high	high	high	none	variable (none)
Column-Oriented Store	high	high	moderate	low	minimal
Document-Oriented Store	high	variable (high)	high	low	variable (low)
Graph Database	variable	variable	high	high	graph theory
Relational Database	variable	variable	low	moderate	relational algebra

Programozási modell

MAPREDUCE

MapReduce keretrendszer

- **MapReduce:** egy programozási modell az adatok *elosztott*, *párhuzamos* feldolgozásához.

- A Google fejlesztette ki
  - Hadoop: széles körben használt nyílt forráskódú MapReduce keretrendszer
  - Használja: Yahoo, Facebook, Twitter, LinkedIn, Ebay, Microsoft, Apple…

MapReduce keretrendszer

- A MapReduce keretrendszer:
  - az adatfeldolgozást *elosztott szervereken párhuzamosan* hajtja végre,
  - *menedzseli* a szerverek közti kommunikációt és adatcserét,
    - *master node*
    - *worker node*
  - *hibatűrést* és *skálázhatóságot* biztosít

- nincs adatmodell, az adatok tárolása:
  - fájlokban
    - GFS: Google File System (Google implementáció)
    - HDFS: Hadoop Distributed File System (Hadoop implementáció)
  - (vagy adatbázisokban.)

Párhuzamosított adatfeldolgozás

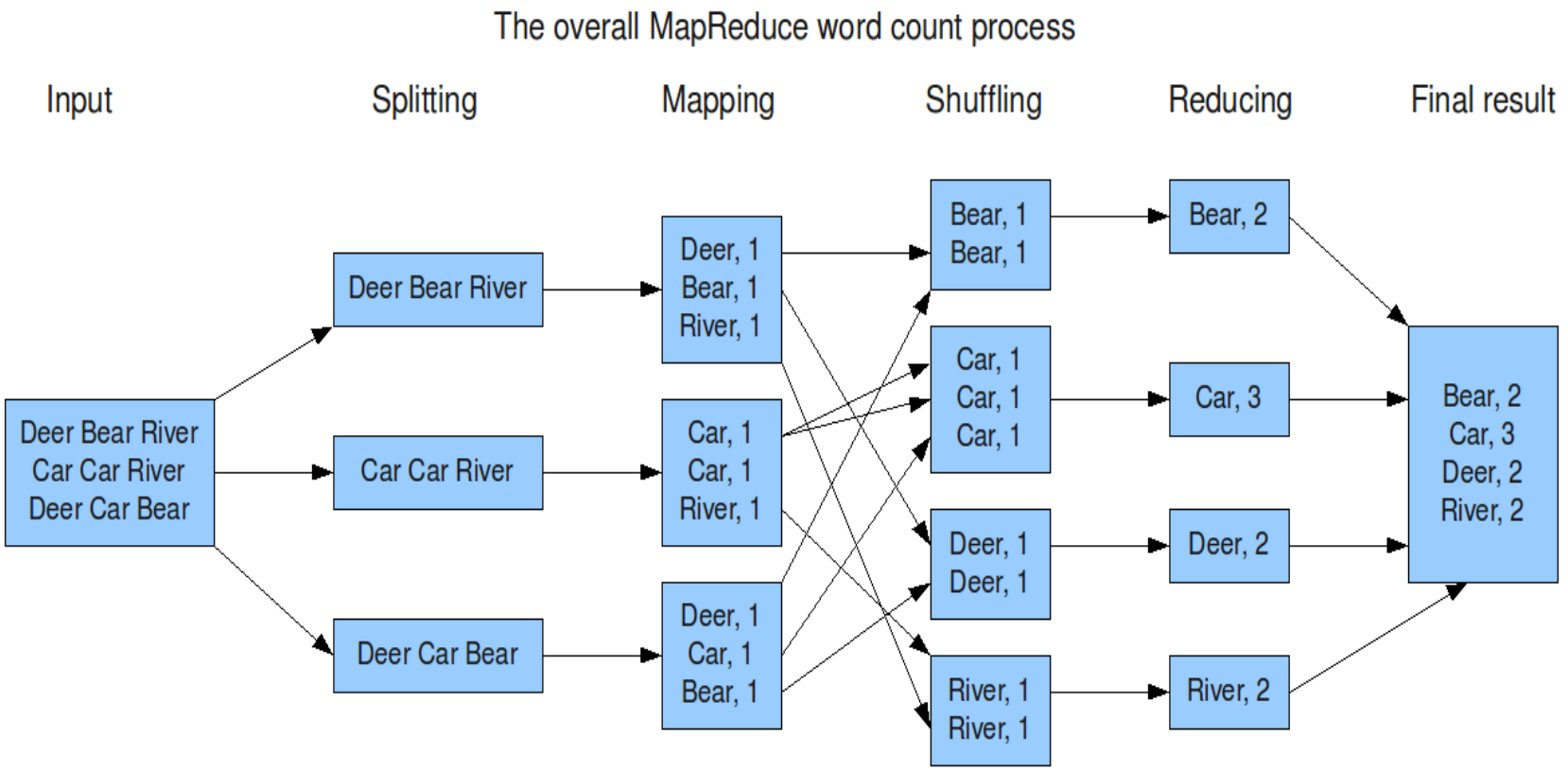
- **master** és **worker** node-ok
  - A **master node** beolvassa az inputot, kisebb részproblémákra osztja, és szétosztja a **worker nod**-oknak.
    - Egy worker node ugyanezt megteheti, ezáltal hierarchikus szerkezet alakul ki.
  - A worker node végrehajtja a feladatot és visszaadja az eredményt a master node-jának.

A MapReduce folyamat

Adatfeldolgozási folyamat:

- 5 (3) lépéses modell:
  1. Adatelőkészítés a mapper számára: a bemeneti adatok felosztása az egyes map processzek részére
  2. Map: a map() függvény alkalmazása és a részeredmény ideiglenes tárolása
  3. Rendezés: részeredmények kulcs szerinti rendezése – ugyanazon kulcsú részeredmények ugyanazon reduce processzekhez kerülnek
  4. Reduce: részeredmények párhuzamos feldolgozás a reduce processzekben
  5. Kimeneti adatok írása: minden reducer processz outputjának összegyűjtése és a végső kimenet előállítása

MapReduce példa – Szavak számolása



Map() és Reduce() függvények

- map(): kulcs-érték párok képzése

  map(elem) → <kulcs, érték> párok listája

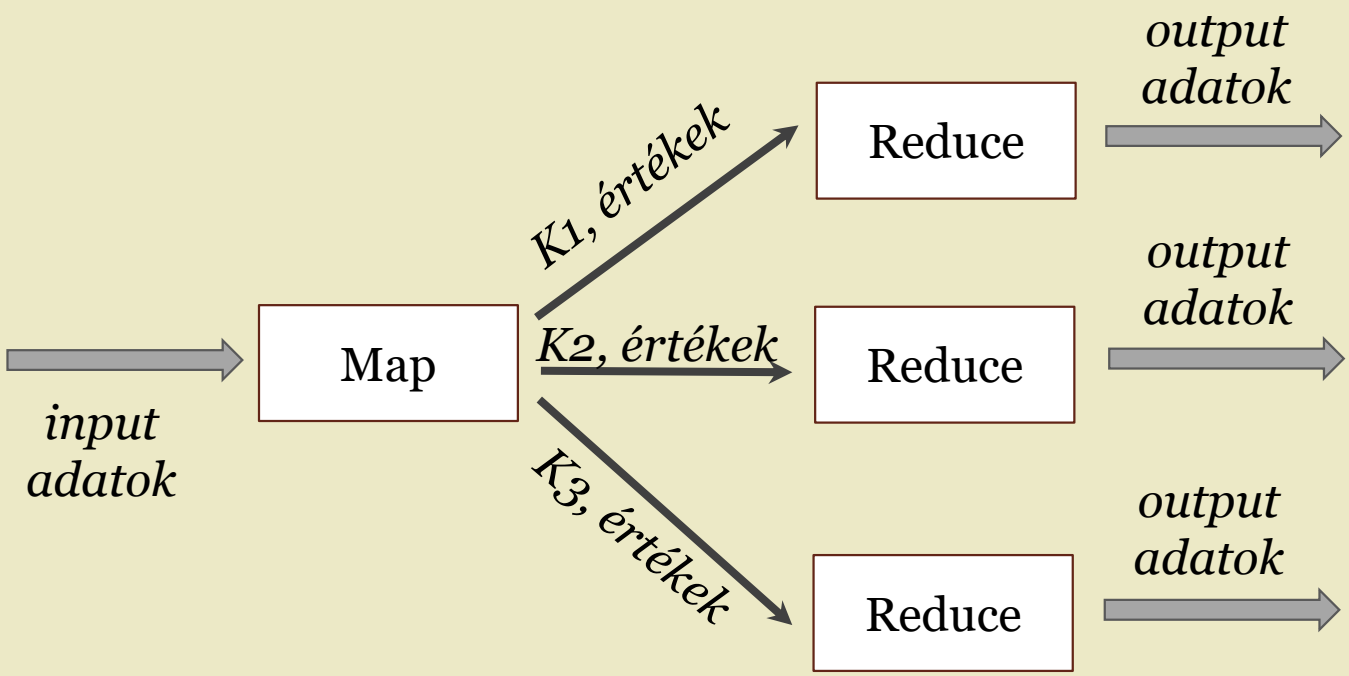
  - a kimenet tartalmazhat több bejegyzést is ugyanahhoz a kulcshoz
  - megvalósítása **kötelező**

- reduce(): redukált output előállítása

  reduce(kulcs, értékek listája) → 0 vagy több rekord

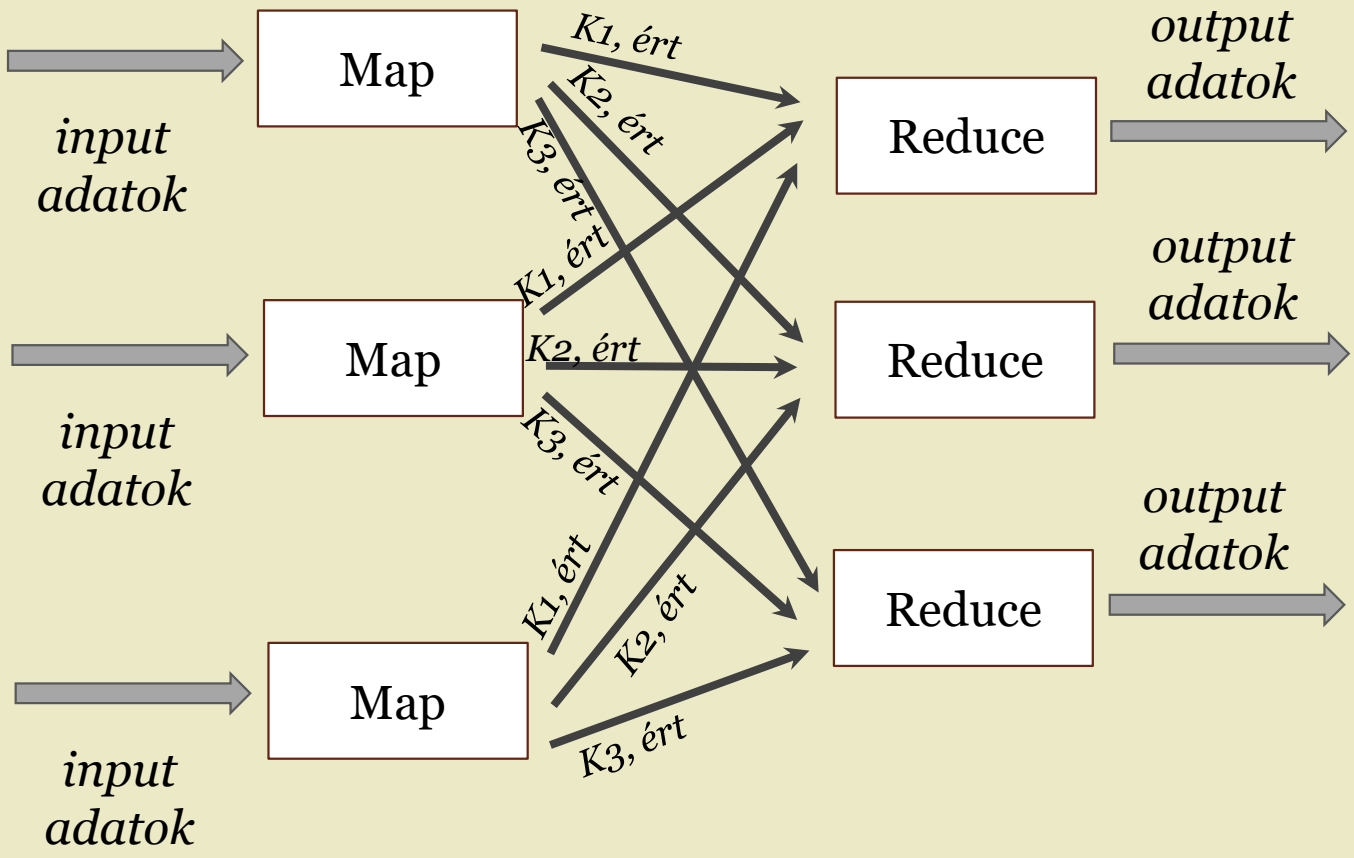
  - a műveletnek idempotensnek kell lennie, ugyanis a reduce függvény a feldolgozás során többször is lefuthat
    - Pl: a master node reduce művelettel összesíti a workerektől kapott eredményeket.
  - megvalósítása **opcionális**

MapReduce architektúra

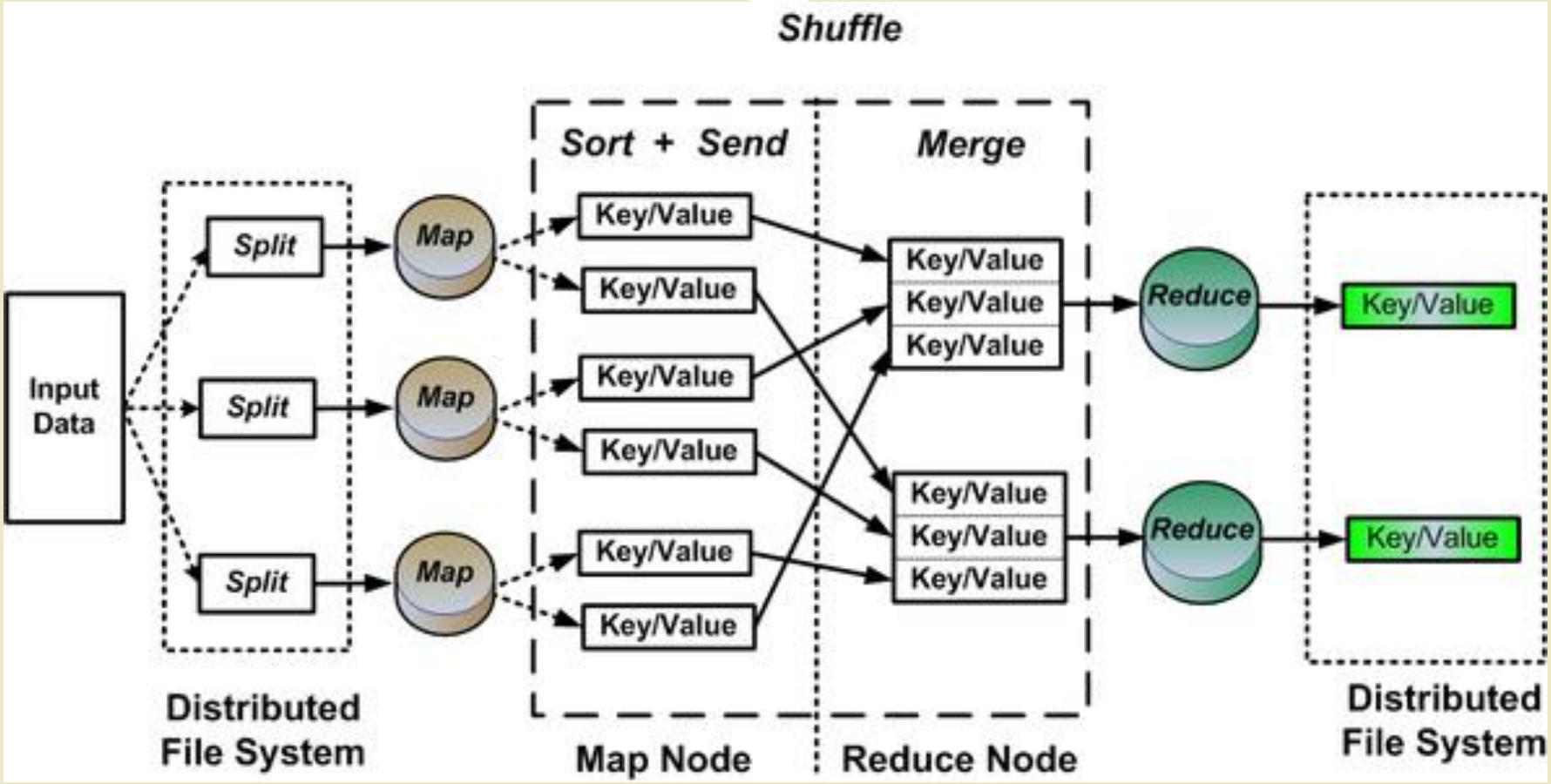


MapReduce architektúra II.

- a map() process párhuzamosítható

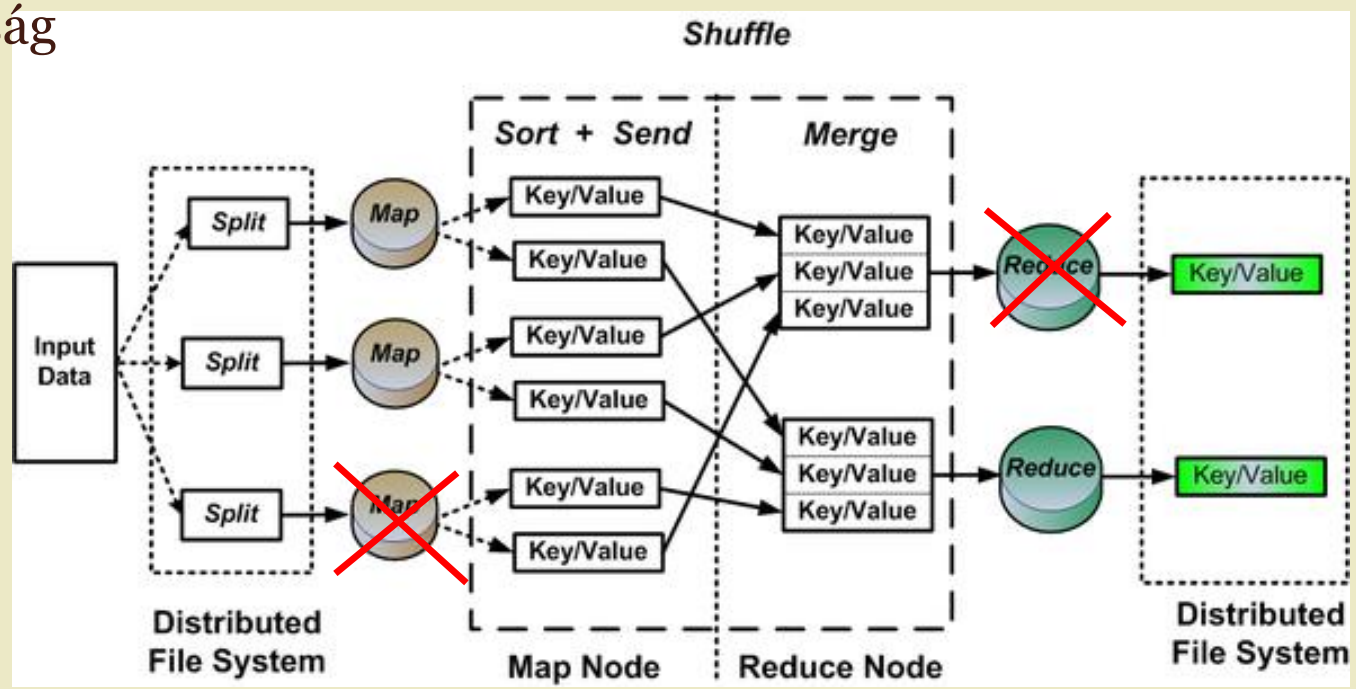


MapReduce architektúra III.



MapReduce architektúra

- MapReduce rendszer szolgáltatja:
  - Adatfeldolgozási folyamat
    - *read/split input → map → shuffle → reduce → write output*
  - hibatűrés
  - skálázhatóság



Megvalósítás – példa

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += ParseInt(pc)
  emit (word, sum)
```

- Each document is split into words, and each word is counted by the *map* function, using the word as the result key. The framework puts together all the pairs with the same key and feeds them to the same call to *reduce*. Thus, this function just needs to sum all of its input values to find the total appearances of that word.

ACID – NoSQL – NewSQL

NoSQL – Történeti fejlődés

- 70-es évek: kulcs-érték tárolók, hálós adatbázismodell
- 2004: Google – Map Reduce keretrendszer
- 2006: Google – BigTable adatbázis-kezelő
  - azóta folyamatosan jelennem meg a nagy vállalatok (Yahoo!, Facebook, LinkedIn, Amazon…) saját rendszerei
- 2009: NoSQL kifejezés

ACID – NoSQL, NewSQL

- NoSQL rendszerek: gyakran feladják az ACID elveket a teljesítmény fokozása végett

- NewSQL rendszerek:
  - *modern relációs DBMS-ek* melyek
  - a NoSQL rendszerekhez hasonló *skálázhatóságot* valósítnak meg az *OLTP* működés során oly módon, hogy
  - továbbra is biztosítják az *ACID elveket*

  - e.g: *Google Spanner, VoltDB, Clustrix, MemSQL, SAP HANA, FoundationDB, NuoDB*, …

RDBMS – NoSQL - NewSQL

Characteristic	RDBMS	NoSQL	NewSQL
ACID compliance (Data, Transaction integrity)	Yes	No	Yes
OLAP/OLTP	Yes	No	Yes
Data analysis (aggregate, transform, etc.)	Yes	No	Yes
Schema rigidity (Strict mapping of model)	Yes	No	Maybe
Data format flexibility	No	Yes	Maybe
Distributed computing	Yes	Yes	Yes
Scale up (vertical)/Scale out (horizontal)	Yes	Yes	Yes
Performance with growing data	Fast	Fast	Very Fast
Performance overhead	Huge	Moderate	Minimal
Popularity/community Support	Huge	Growing	Slowly growing

Ajánlott irodalom

- Az előadás jelentősen támaszkodott a következő BME jegyzetre:
  - https://db.bme.hu/~gajdos/2012adatb2/3.%20eloadas%20NoSQL%20adatb%e1zisok%20doc.pdf