UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

6. Adattárolás

Dr. Bilicki Vilmos
Szegedi Tudományegyetem
Informatikai Tanszékcsoport
Szoftverfejlesztés Tanszék

Adattípusok

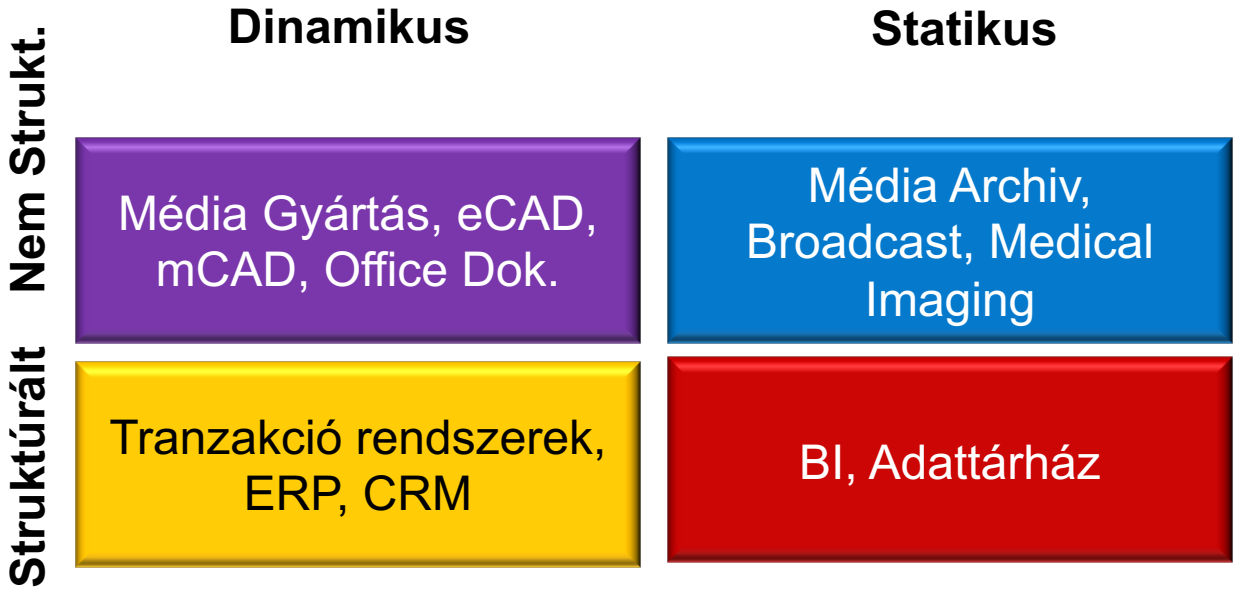
- Az adatot az alábbi módokon lehet csoportosítani:
  1. Struktúrált adat:
     - Van egy előre megadott modellünk amely az adatot könnyen tárolható és feldolgozható, kezelhető formába önti
     - pl.: relációs adat tárolás
  2. Nem struktúrált adat:
     - Az előző ellenkezője
     - pl.: Lapos, bináris fájlok szövegek, videó vagy audió információval
     - Megjegyzés: az adat nem teljesen struktúrálatlan (pl.: az audió konténernek meghatározott fomrája van)

Adattípusok

- További csoportosítás
  3. Dinamikus adat:
     - Az adat gyakran változik
     - Pl.: dokumentumok és tranzakciós bejegyzések egy pénzügyi rendszerben
  4. Statikus adat:
     - A dinamikus ellentéte
     - Pl.: Medicina CT/EMR képek

Miért érdemes az adatot osztályozni?

- E két dimenzió mentén megérthetjük az adat igényünket



- Relációs adatbázisok a struktúrált adatokra a legjobbak
- Fájl rendszerek vagy *NoSQL adatbázisok* a statikus vagy struktúrlatlan adatokra a legjobbak

Visszatekintés

- Relációs adatbázisok– az üzlet alappillérei (start-up -> nyílt forrású)
- Web alapú alkalmazások kiugrásokat, terhelési csúcsokat okoztak (Slashdot hatás)
  - Különösen igaz a lakossági eKereskedelem oldalakra
- A fejlesztők elkezdtek memória alapú (csak olvasható) gyorstárakat tenni az adatbázisok elé vagy az adatbázis gyorsítótárát kezdték az alkalmazásba integrálni (pl.: Ehcache)
- Az adatmennyiség növekedésével ez nem volt fenntartható

Horizontális Skálázás(Scale out)

- Nem kezelhetőek tetszőleges nagy adat halmazok egy gépen
- Az RDBMS világ nem az elosztottságra van megtervezve
- Több csomóponton futó megoldásokat kerestek -> scale out (horizontális skálázás)
- Többfajta megoldás:
  - Mester-szolga (master-slave)
  - Horizontális partícionálás (Sharding)

Igények

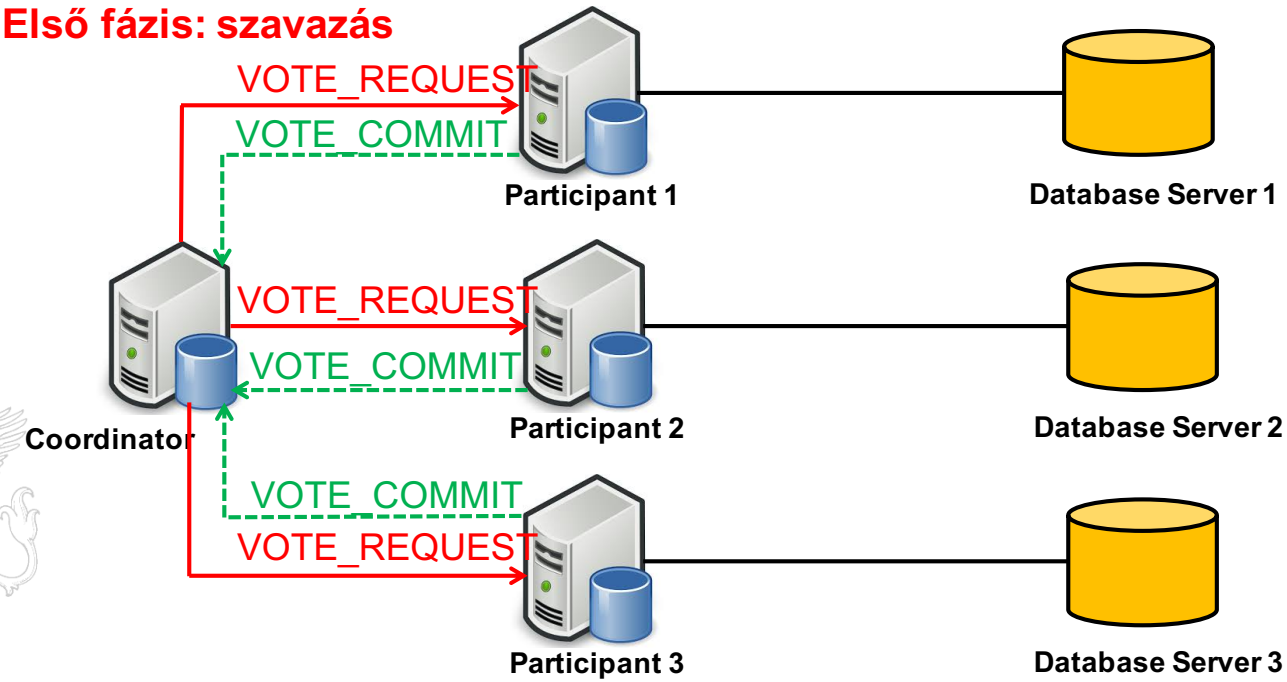
- Több gép több kapacitás
- Az alkalmazásnak transzparens
- Nincs egy meghibásodási pont

Mester/Szolga

- Minden írás a mesteren. Minden olvasás a replikált szolgákon
- A kritikus olvasások inkonzisztensek lehetnek mivel lehet, hogy az írások még nem propagálódtak
- Nagy adatmennyiség esetén a duplikálás problémát okozhat a mesternek

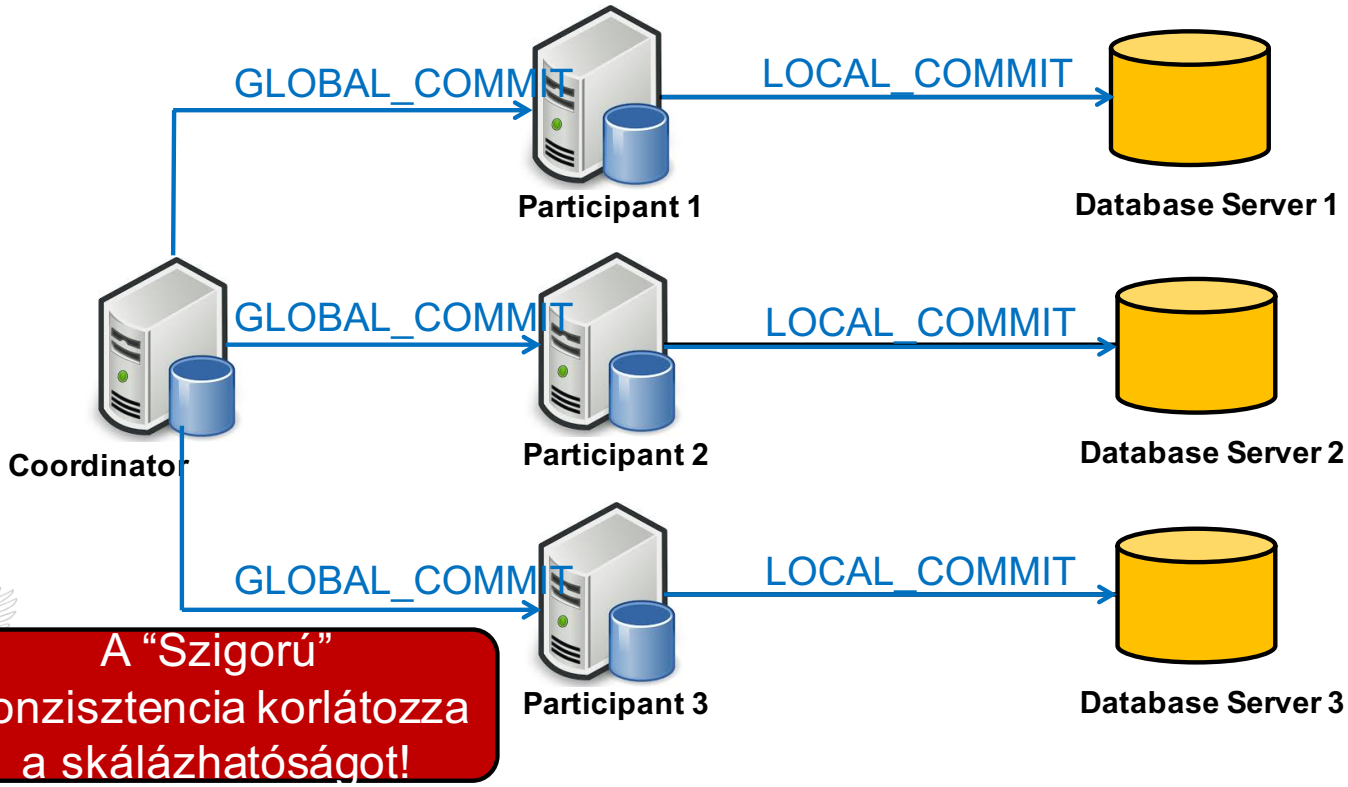
Két fázisú elfogadás

- A konzisztencia megőrzésére tranzakcionális megoldás



Két fázisú elfogadás

2. fázis: Elfogadás



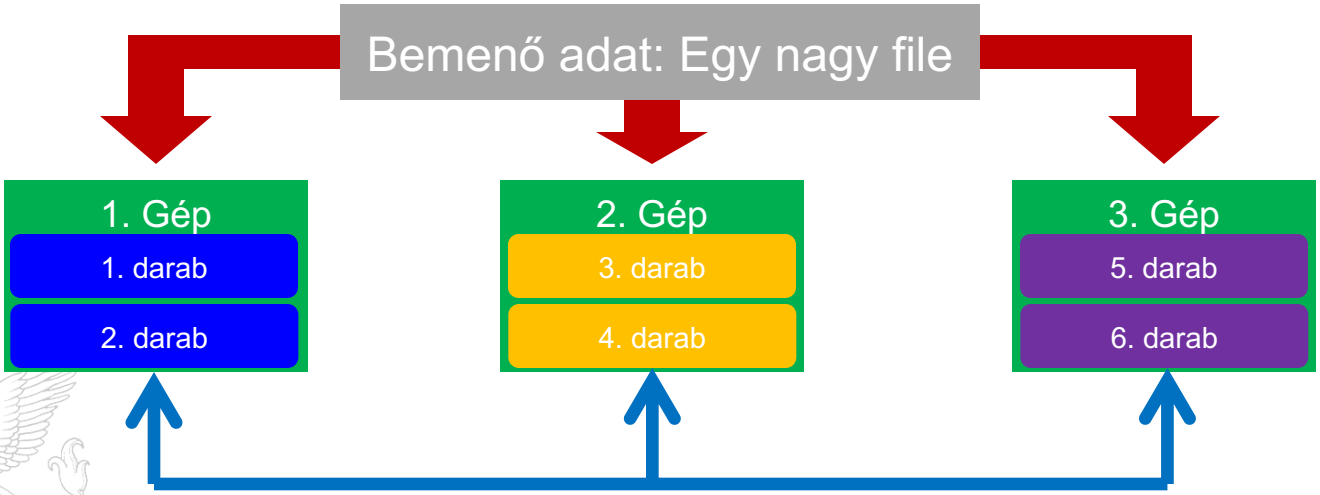
A “Szigorú” konzisztencia korlátozza a skálázhatóságot!

Partícionálás - Sharding

- Típusai
  - Vertikális (tábla szintű)
  - Tartomány alapú (sor szintű)
  - Kulcs vagy kivonat alapú
  - Címtár alapú
- Az olvasásra és az írásra is skálázódik
- Nem transzparens az alkalmazásnak ismernie kell a partícionálást!
- A kényszerek/egyesítések nem működnek partíciókon átívelően
- A partíciókon átívelően nincs referenciális integritás

Partícionálás

- Az adatot partíciókra – csíkokra bontjuk a párhuzamos kezelés elősegítésére



Pl.: az 1,3,5 darabok párhuzamosan elérhetőek

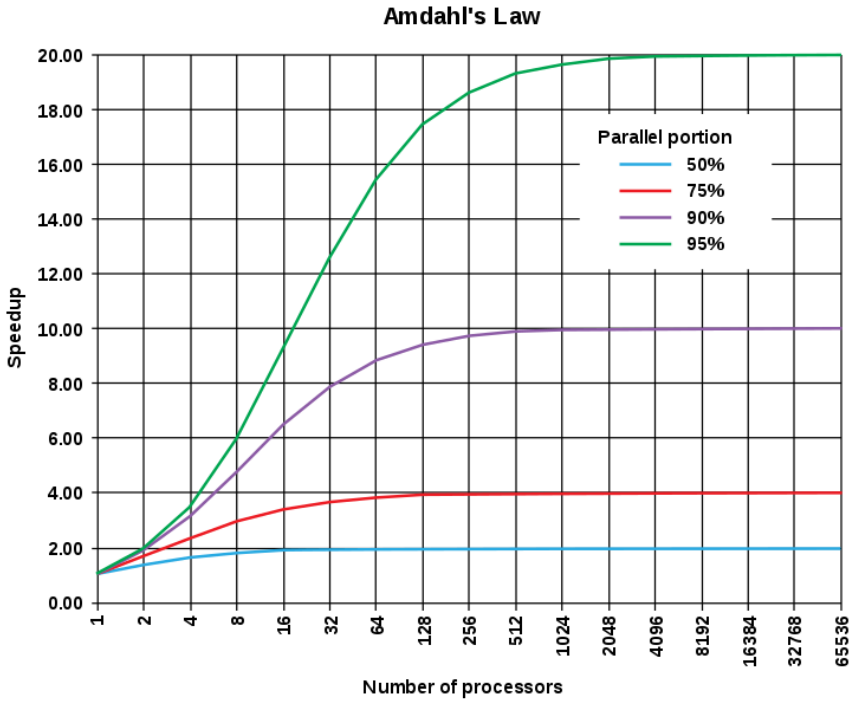
Amdahl törvénye

- Mennyivel gyorsabb egy párhuzamosított program?
  - Tegyük fel, hogy a soros futás T_1 időt igényel p párhuzamos gépen T_p időt
  - Tegyük fel, hogy az egész program s része nem párhuzamosítható míg $1-s$ része párhuzamosítható
  - A gyorsulás (***Amdahl képlete***):

$$\frac{T_1}{T_p} = \frac{T_1}{(T_1 \times s + T_1 \times \frac{1-s}{p})} = \frac{1}{s + \frac{1-s}{p}}$$

Amdahl törvénye: Példa

- Tegyük fel hogy:
  - A program 80%-a párhuzamosítható
  - 4 gépen futtatjuk párhuzamosan



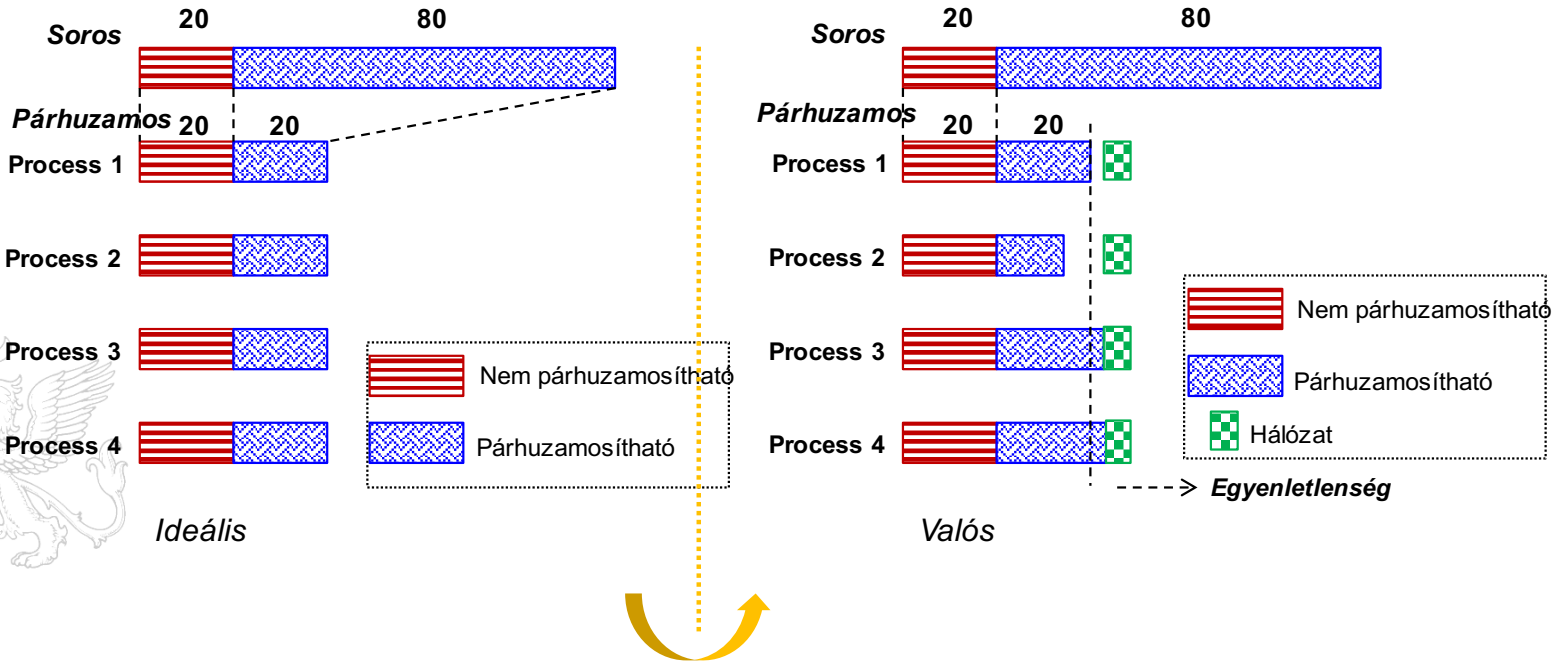
- A gyorsulás:

$$\frac{1}{s+\frac{1-s}{p}} = \frac{1}{0.2+\frac{0.8}{4}} = 2.5 \text{ times}$$

Annak ellenére, hogy 4 gépen fut csak 2,5-szeres a gyorsulás!

Valós Vs. Modell

- Amdahl törvénye is egy modellen alapul
- A valóságban a kommunikációs plussz teher és a munka terhelés egyenletlenség is számít



Egyébb skálázási megoldások

- Multi-Master replikáció
- Csak BESZÚRÁS, nincs FRISSÍTÉS/TÖRLÉS (verziózás)
- Nincs JOIN, rövidebb lekérés idő
  - Adat denormalizálás
  - A konzisztencia az alkalmazás dolga
- Memóriabeli adatbázisok

Mi a NoSQL?

- Kifejtve **N**ot **O**nly **SQL – Nem Csak SQL**
- Az nem relációs elven működő rendszerek egy osztálya
- Általában nincs rögzített séma és nincs join
- A NoSQL megoldások egy vagy több ACID képességet lazábban vesznek

Miért NoSQL?

- Az RDBMS nem tud mindent optimálisan megoldani
- Ahogyan vannak különböző programozási nyelvek úgy az adattárolásnál is szükséges több paradigmát is támogatni
- A NoSQL megoldás ma már elfogadott

Hogyan jutottunk ide?

- A közösségi oldalak robbanásszerű terjedése (Facebook, Twitter/Google 2008/2009) - nagy adat
- A felhő alapú szolgáltatások terjedése. Pl.: Amazon S3 (egyszerű tárolási megoldás)
- Dinamikus nyelvek terjedése (séma mentes)
- Nyílt forrású közösség

Dynamo és a BigTable

- Három cikk volt a NoSQL mozgalom csírája
  - BigTable (Google)
  - Dynamo (Amazon)
    - Pletyka protokoll (felderítés és hiba detektálás)
    - Elosztott kulcs-érték tár
    - Végső soron konzisztens
  - CAP tétel

CAP Tétel

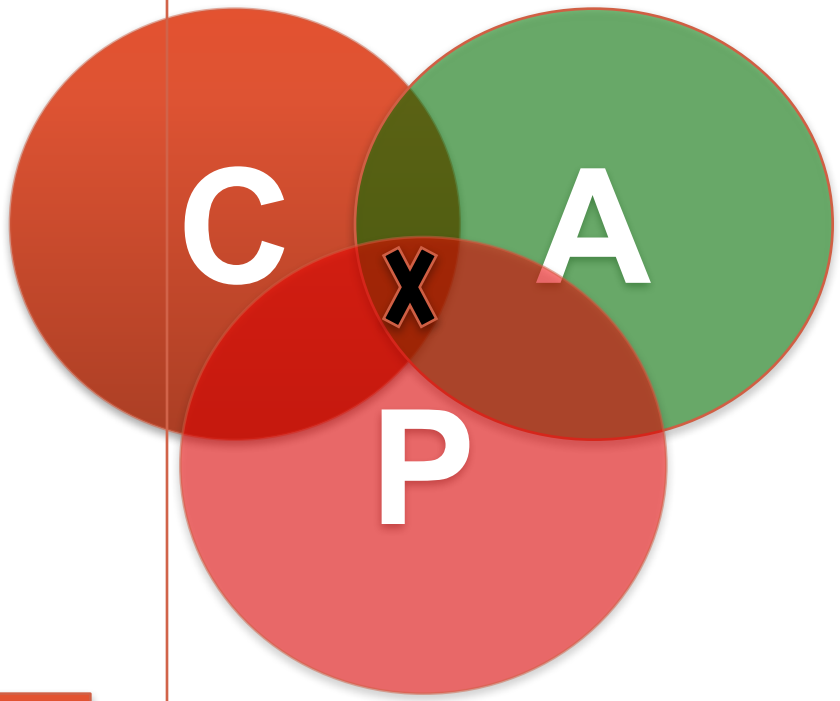
- Prof. Eric Brewer sejtése 2000
- Elosztott rendszer tervezési döntési pontok
- Nem lehet egyszerr mindhármat kielégíteni:
  - **Konzisztencia**
  - **Rendlkezésre állás**
  - **Partíció tűrés**

CAP Tétel

- **K**onzisztencia:
  - Adott időpillanatban minden csomópont ugyanazon adatot látja
- **R**endelkezésre álás:
  - A kieső csomópontok nem akadályozzák a működőket a működésben
- **P**artíció tűrés:
  - A rendszer partíciók esetén istovább működik
- Egy elosztott rendszer ezek közül csak tettőt tud teljesíteni **mindhármat nem**

A CAP tétel újragodolása

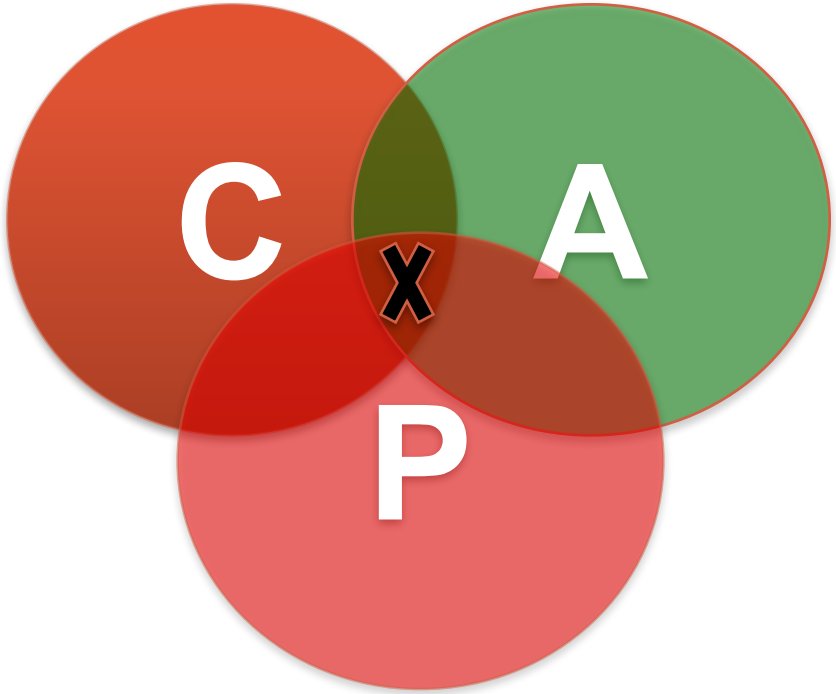
- Ezek közül kettő teljesül:
  - Consistency
  - Availability
  - Partition tolerance
- Válasszunk kettőt
- A fentiek alapján ezek lehetségesek:
  - CP
  - AP
  - CA



Bármilyen probléma?

Konzisztencia vagy rendelkezésre állás

- Nem bináris döntés
- AP csökkentis a konzisztencia követelményt, de nem szünteti azt meg
- CP csökkenti a rendelkezésre állás követelményt, de nem szünteti azt meg
- A CP és az AP is lehet köztes megoldás



A konzisztencia típusai

- Erős konzisztencia
  - Az írás után **bármely ezt követő** hozzáférés ugyanezt az eredményt adja vissza
- Gyenge konzisztencia
  - **Nem garantált** az, hogy az írást követő olvasások ugyanazt az értéket kapják
- **Végső soron konzisztens BASE**
  - A gyenge konzisztencia egy típusa
  - **B**asically **A**vailable: garantálja a rendelkezésre állást
  - **S**oft-State: a rendszer állapota változhat az időben
  - **E**ventual Consistency: a végén konzisztens lesz
  - Garantált, hogy amennyiben nincs több írás akkor előbb utóbb minden hozzáférés az utolsó eredményt fogja kapni

Végső soron konzisztens variációk

- Kauzális konzisztenica
  - A kauzális viszonyban lévő folyamatok konzisztens adatot fognak látni
- Olvasd a saját írásod konzistencia
  - A folyamat a saját írsát mindég visszakapja
- Viszony konzisztencia
  - Amíg a viszony tart addig a rendszer garantája az olvasd a saját írásodta konzisztenciát

Végső soron konzisztens variációk

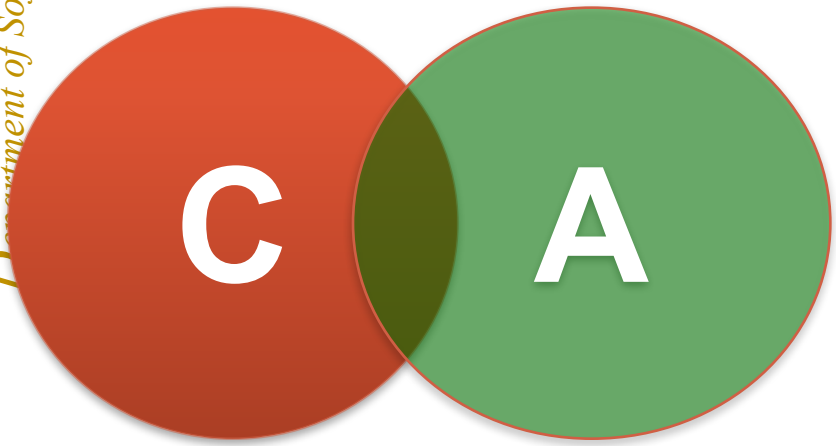
- Monoton olvasás konzisztenica
  - Amennyiben egy folyamat már egy értéket látott attól régebbit nem láthat
- Monoton írás konzisztencia
  - A rendszer az írásokat folyamat szinten sorosítja
- A gyakorlatban
  - Több típus is kombinálható
  - Monoton olvasás és olvasd az írásod javasolt

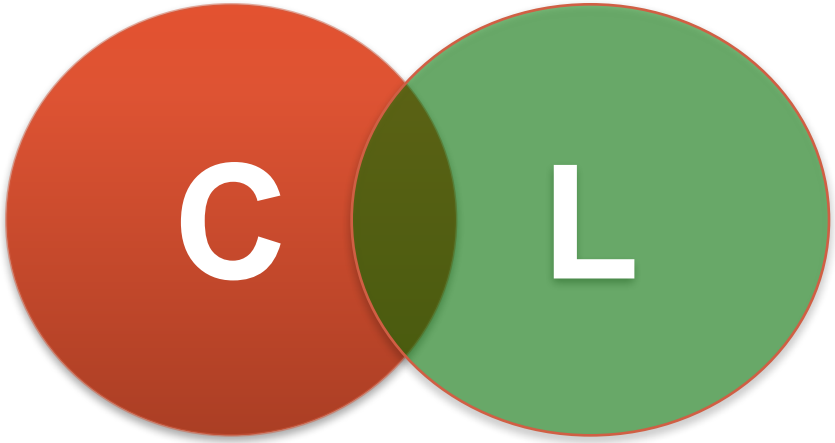
CAP -> PACELC

- Egy jóval gyakorlat közelibb megközelítésmód:
  - Amennyiben van **partíció (P)**, akkor döntenünk kell a **rendelkezésre állás és a konzisztencia között (A and C)**; **egyébként (E)**, amikor minden rendben van akkor **késleltetés (L) vagy konzisztencia(C)**?

Abadi, Daniel J. "Consistency tradeoffs in modern distributed database system design." Computer-IEEE Computer Magazine 45.2 (2012): 37.

PACELC





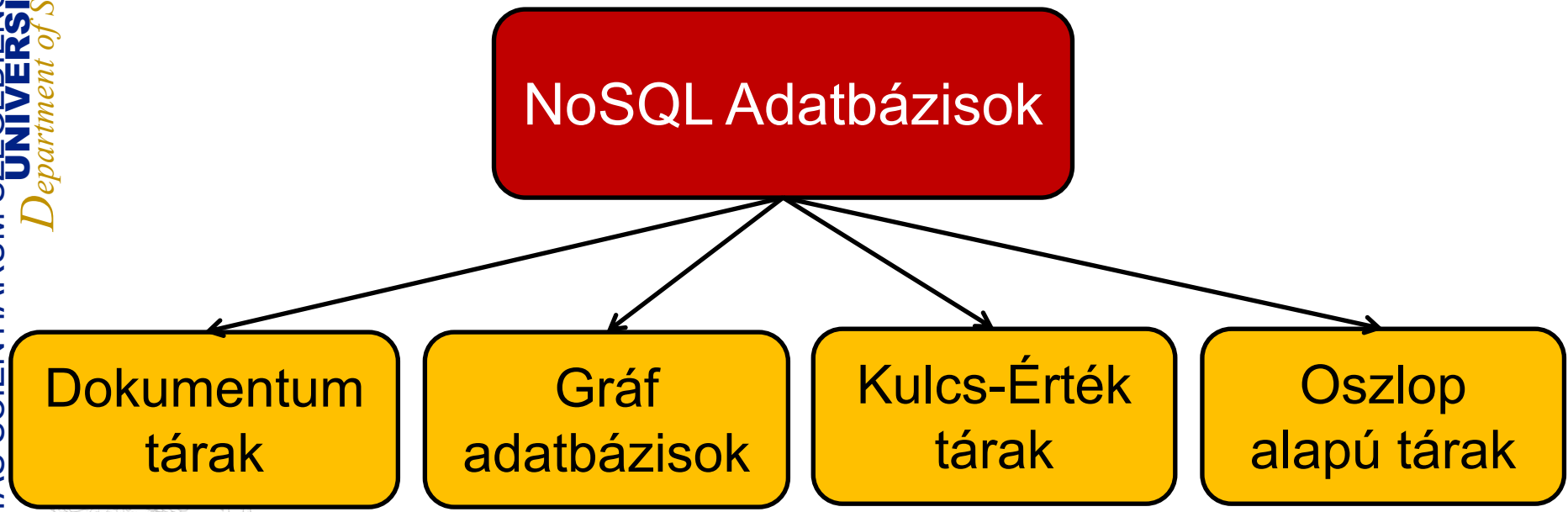
Partícionál t

Normál

NoSQL adatbázisok

- Főbb jellemzők:
  - Nincs szigorú séma követelmény
  - Nincs szigorú ACID képesség
  - Konzisztencia vs. Késleltetés/Rendelkezésre állás

A NoSQL adatbázisok típusai

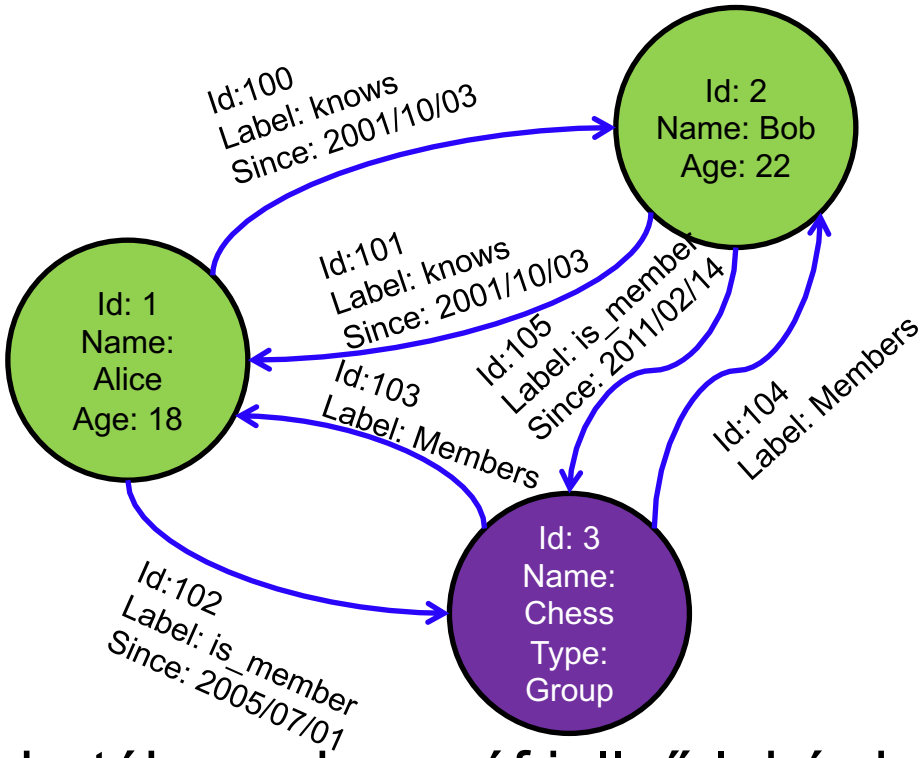


Dokumentum tárak

- A dokumentumokat szabványos formában tárolja (pl.: XML, JSON, PDF or Office Dokumentum)
  - Ezeket legtöbbször Binary Large Objects (BLOBs)-ként hivatkozzuk
- A dokumentumok indexelhetőek
  - Ezzel a klasszikus fájlrendszereknél gyorsabbak lehetnek
- Pl.: MongoDB és CouchDB (mindkettőnél alkalmazható a MapReduce)

Gráf adatbázisok

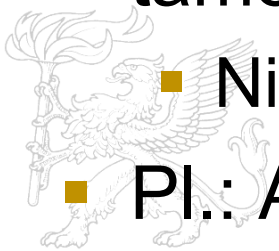
- Az adat csomópontokkal és élekkel reprezetálható



- Nagyon hatékonyak a gráf jellgű lekérdezéseknél (pl.:mi a legrövidebb útvonal két elem között)
- Pl.: Neo4j and VertexDB

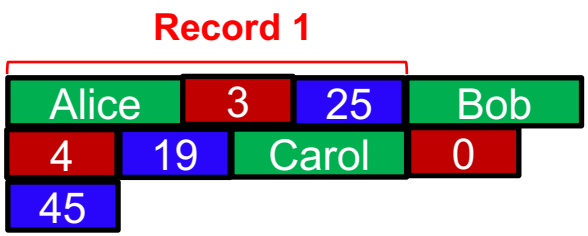
Kulcs-Érték tárolók

- A kulcsok komplexebb adatkohoz kötöttek (pl.: listák)
- A kulcsokat kivonat táblában tároljuk és így egyszerűen eloszthatjuk
- Az ilyen jellegű tárak a tipikus CRUD (create, read, update, and delete) műveleteket támogatják
  - Nincs join nincs aggregáció
- Pl.: Amazon DynamoDB és Apache Cassandra

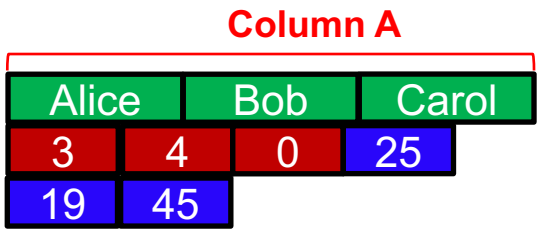


Oszlopos adatbázisok

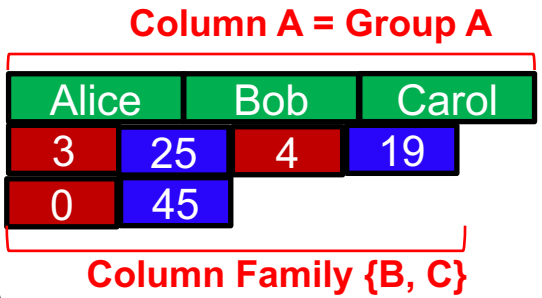
- Az oszlopos adatbázisok a klasszikus relációs és a kulcs-érték adatbázisok ötvözetei
  - Az értékek oszlop csoportokban vannak tárolva, oszlop rendezésben (nem sor rendezésben)



Sor rendezés



Oszlopos(oszlop rendezésű)



Oszlopos helyi csoportokkal

  - Az értékeket az oszlop alapján kapja vissza

- Pl.: HBase és Vertica

Polygot (soknyelvű) perzisztencia

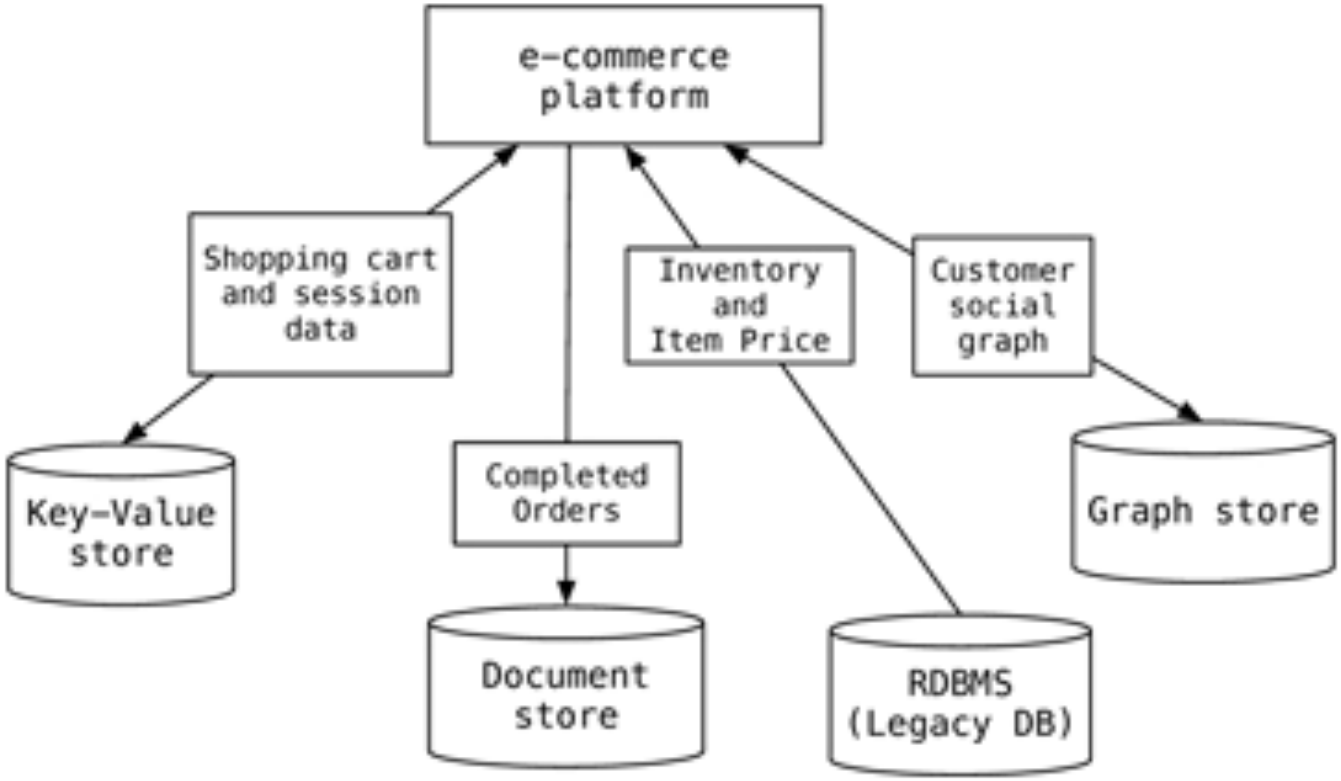


Figure 13.3. *Example implementation of polyglot persistence*

A legismertebb NoSQL megoldások

- Kulcs/Érték vagy ‘a nagy kivonat tábla’.
  - Amazon S3 (Dynamo)
  - Voldemort
  - Scalaris
- Séma mentes:
  - Cassandra (oszlop alapú)
  - CouchDB (dokumentum alapú)
  - Neo4J (gráf alapú)
  - HBase (oszlop alapú)

Kulcs/Érték

Előnye:

- Nagyon gyors
- Nagyon skálázható
- Egyszerű modell
- Horizontálisan szétkenhető

Hátránya:

- Sok adatstruktúra nehezen modellezhető kulcs-érték párokkal

Séma mentes

Előnye:

- Az adat modellje gazdagabb mint a kulcs/érték páré
- Végső soron konzisztens
- A legtöbb elosztott
- Jó teljesítmény

Hátránya:

- Nincs ACID tranzakció és JOIN

Közös előnyök

- Olcsó és könnyen megvalósítható (open source)
- Az adat több csomópontra is replikálódik (robosztus) és partícionálható
  - A kieső csomópontok könnyen helyettesíthetőek
  - Nincs egy meghibásodási pont
- Könnyen szétkenhető
- Nem kell séma
- Könnyen fel és le skálázható
- CAP

Mit adunk fel?

- join
- group by
- order by
- ACID tranzakciókat
- SQL időnként zavaró, de mégis igen hatékony
- Az SQL szintű könnyű integrálhatóságot

Keresés

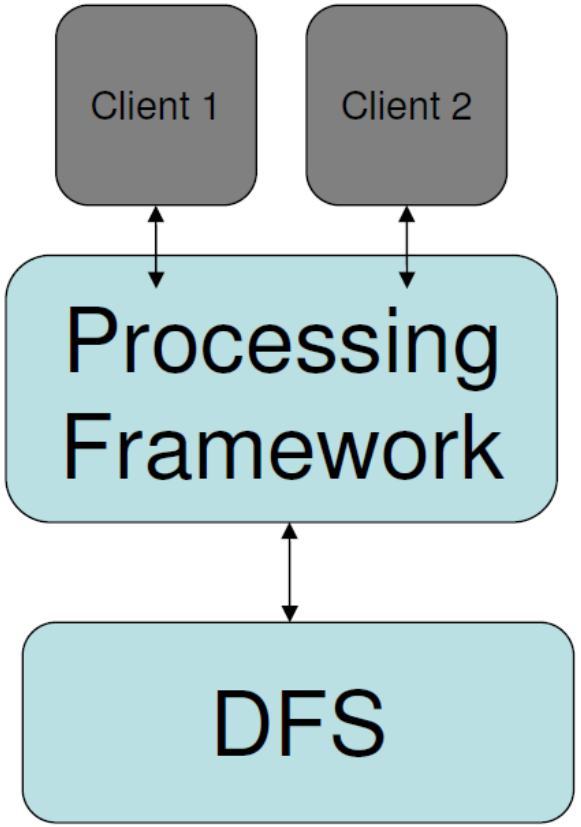
- Relációs
  - SELECT `column` FROM `database`,`table` WHERE `id` = key;
  - SELECT product_name FROM rockets WHERE id = 123;
- Cassandra (standard)
  - keyspace.getSlice(key, “column_family”, "column")
  - keyspace.getSlice(123, **new** ColumnParent(“rockets”), *getSlicePredicate*());

Tipikus NoSQL API

- get(key)
- put(key, value)
- delete(key)
- execute(key, operation, parameters)

Apache Hadoop

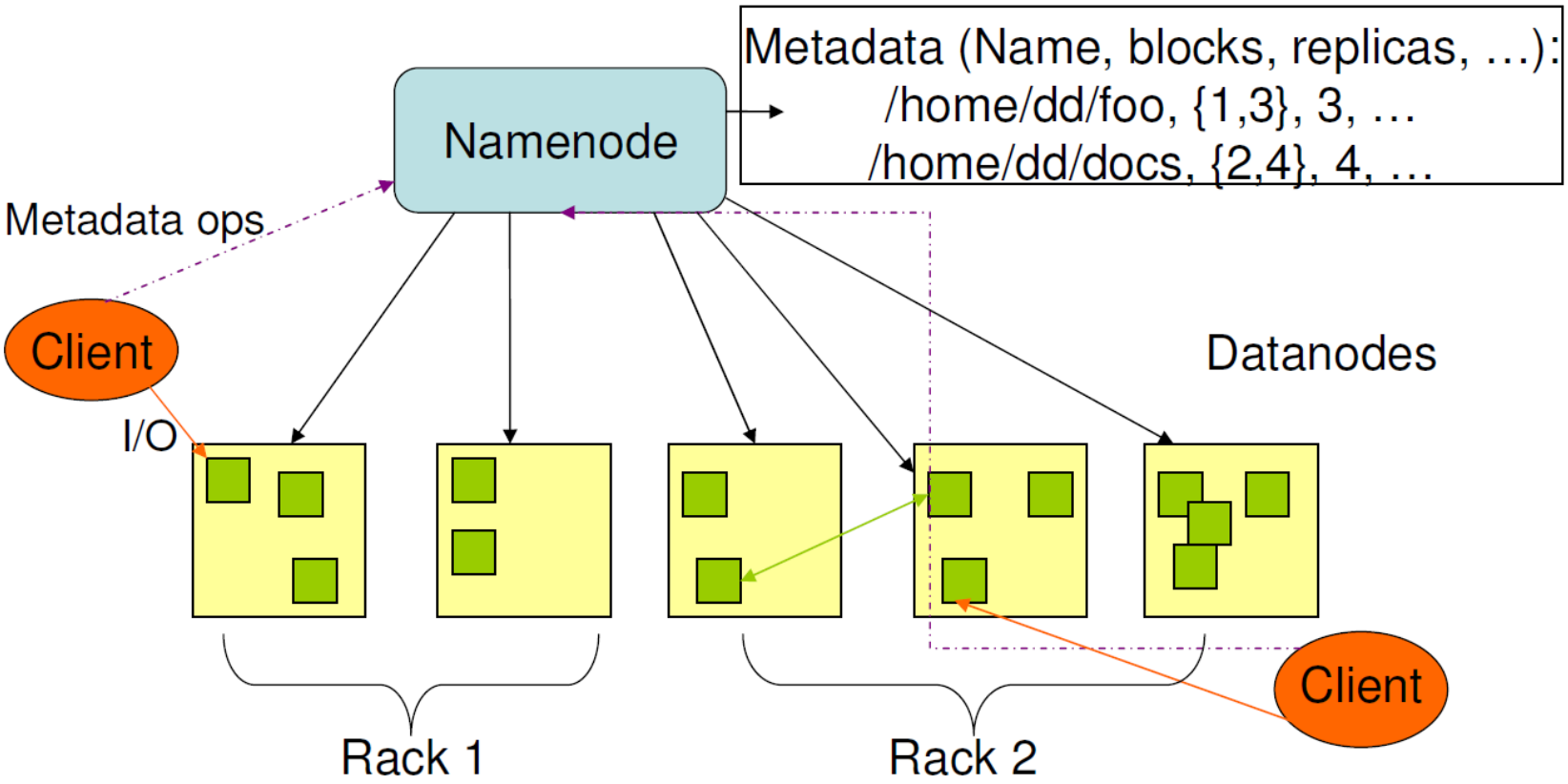
- Nem valós idejű feldolgozásra
- Nagyon nagy adatmennyiségek kezelésére
- Elemei:
  - Elosztott fájlrendszer
  - Elosztott feldolgozó keretrendszer
- Számos kiegészítő keretrendszer
  - Zookeper
  - Hbase
  - Hive
  - Pig Latin
  - …



HDFS

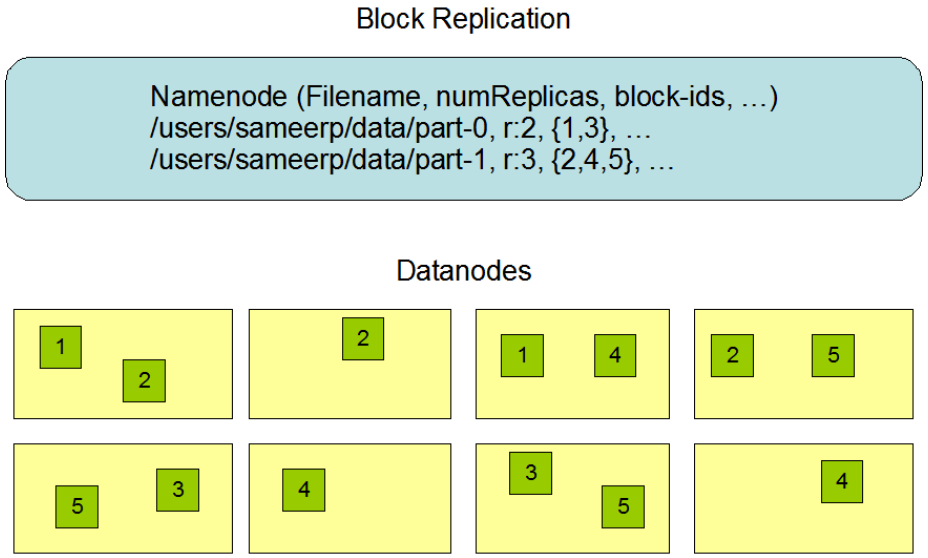
- Elosztott tároló rendszer
  - A Fájlok nagy blokkokra osztva a klaszterben több helyen tárolódnak
  - A replikáció alapegysége a blokk
  - Az adathoz mozgatja a számításokat
  - Egyszerű IO központú API
- Architektúra
  - Mester/Szolga megoldás
  - Mester – elnevezés csomópont
    - Minden metadatot itt kezelnek
    - Tranzakció, log itt van kezelve
    - Az írás/olvasás engedélyezése szintén itt történik
    - A blokk replikáció szintén itt van vezérelve
  - Szolga
    - Értesíti a mestert az általa tárolt blokk ID-kről
    - Kiszolgálja a kliensek írás/olvasás igényeit
    - Replikációt kezeli
    - Ismeri a helyzetét (rack)

HDFS architektúra



HDFS hibakezelés

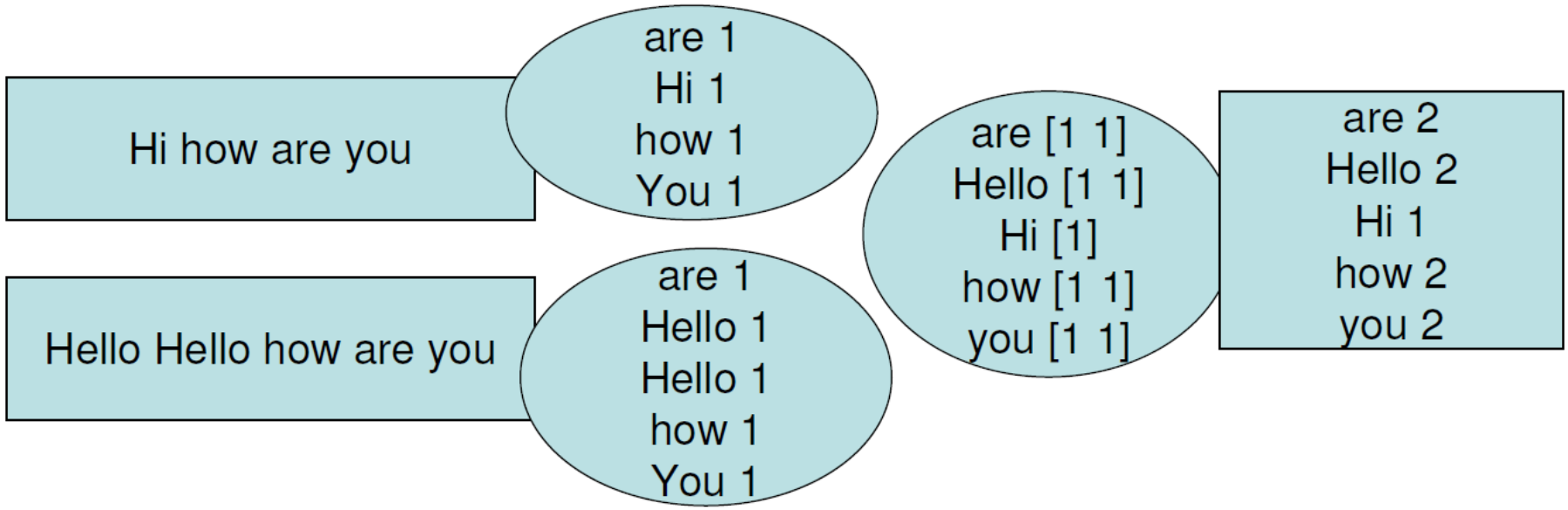
- Mester – Elnevezés csomópont kiesik
  - Másodlagos elnevezés csomópont
- Adat csomópont kiesés
  - Az elnevezés csomópont detektálja ezt (hearthbeat)
  - Kiiktatja a rendszerből
  - Replikák, cső replikáció



HDFS adat megbízhatóság

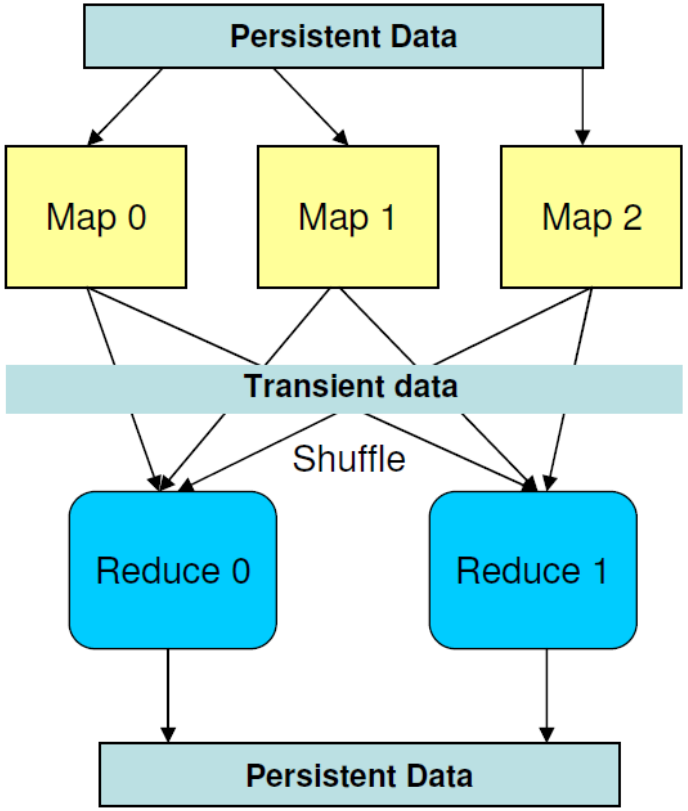
- CRC32 a blokkok hibadetektálására
- A kliens 512 bájtonként számol ellenőrző összeget
- Ez le van tárolva
- Amikor lekéri ellenőrzi
- Ha nem megfelelő akkor más replikához fordul

HDFS elosztott feldolgozás

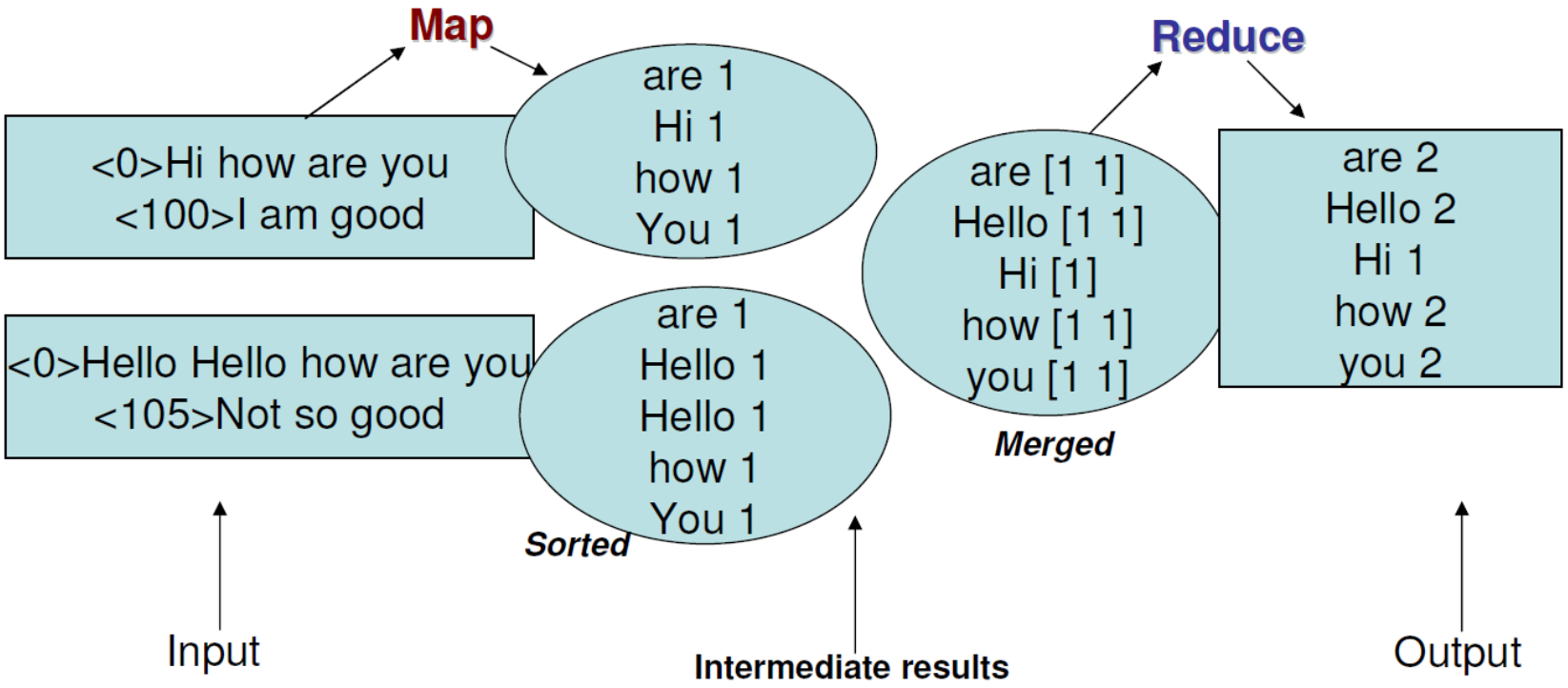


Map - Reduce

- A felhasználói logika:
  - Map feladatok
  - Reduce feladatok
- Az adat kulcs érték párok sorozata
- Map
  - Bemenet:
    - Kulcs 1, Érték 1
  - Kimenet
    - Kulcs 2, Érték 2
- Reduce
  - Kulcsonként egyszer hívódik meg sorbarendezett
  - Bemenet:
    - Kulcs 2, értékek sorozata
  - Kimenet:
    - Kulcs 3, Érték 3



Kulcsok és értékek sorozata

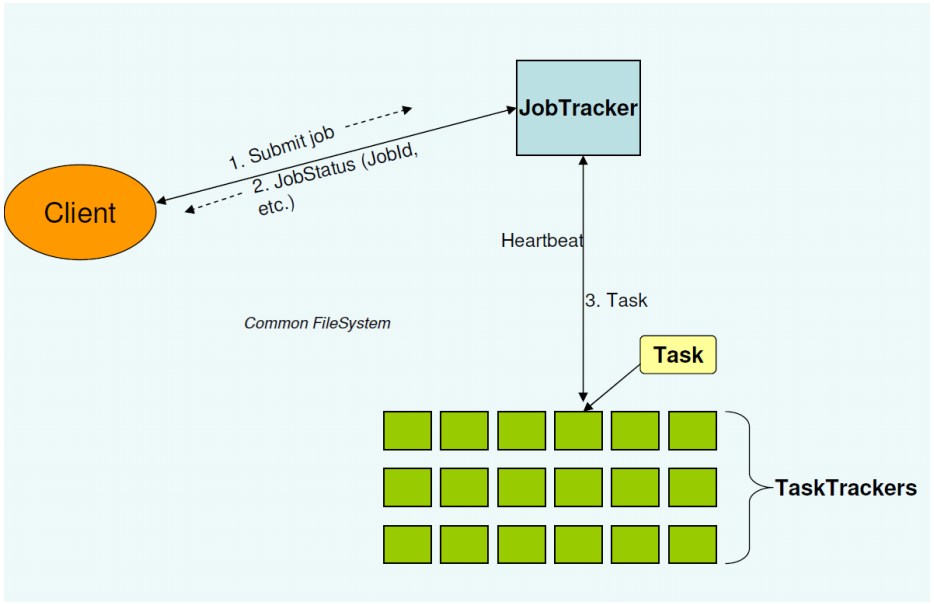


Kód

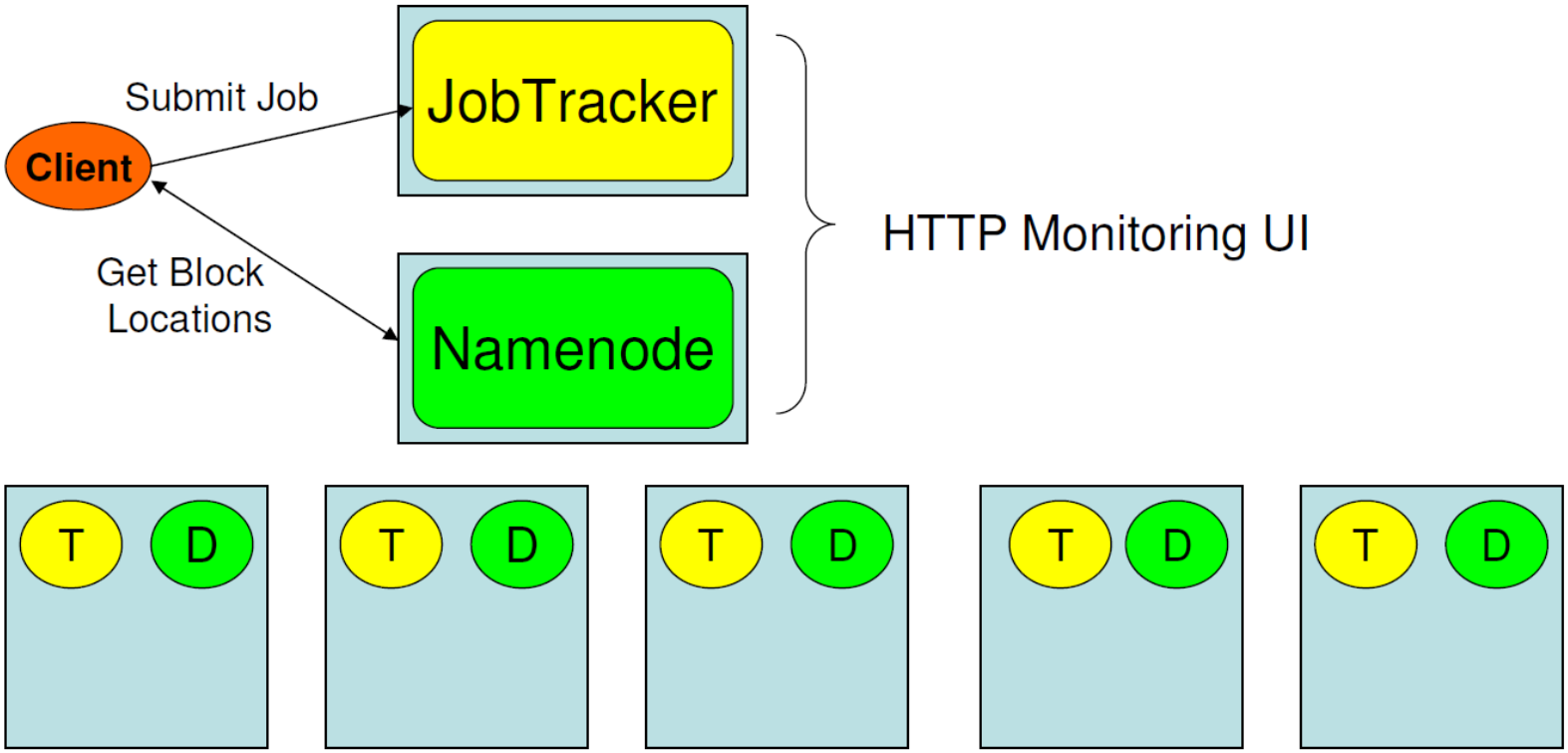
```
public void map(WritableComparable key, Writable value, OutputCollector output, Reporter reporter) throws
        IOException {

    String line = ((UTF8)value).toString();

    StringTokenizer itr = new StringTokenizer(line);

    while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
             output.collect(word, one);
    }
}


public void reduce(WritableComparable key, Iterator values, OutputCollector output,Reporter reporter) throws
        IOException {
    int sum = 0;

    while (values.hasNext()) {
       sum += ((IntWritable) values.next()).get();
     }

     output.collect(key, new IntWritable(sum));
}
```

Map Reduce Architektúra

- Mester/Szolga
- Mester – Munka követő:
  - A felhasználó által beküldött MR munkákat fogadja
  - Kiadja a Map és Reduce feladatokat a feladatkövetőknek
  - Monitorozza a Map és Reduce feladatokat, ha ezek sikertelenek akkor újra ütemezi őket
- Szolga – Feladatkövető
  - A Munka követő által kiadott feladatokat futtatja
  - Kezeli a tárolást és a köztes eredményeket



Map Reduce + HDFS



HBase

- HBase egy Bigtable megvalósítás
- A Hadoop platformra épül
- Egy ritka, többdimmenziós, elosztottan tárolt rendezett map-et valósít meg

```
Table        (HBase table)
    Region        (Regions for the table)
         Store          (Store per ColumnFamily for each Region for the table)
              MemStore           (MemStore for each Store for each Region for the table)
              StoreFile          (StoreFiles for each Store for each Region for the table)
                   Block              (Blocks within a StoreFile within a Store for each Region
```

Adat modell fogalmak

MAP

```
{
   "zzzzz" : "woot",
   "xyz" : "hello",
   "aaaab" : "world",
   "1" : "x",
   "aaaaa" : "y"
}
```

Rendezett

```
{
   "1" : "x",
   "aaaaa" : "y",
   "aaaab" : "world",
   "xyz" : "hello",
   "zzzzz" : "woot"
}
```

Oszlop családok

```
{
   // ...
   "aaaaa" : {
      "A" : {
         "foo" : "y",
         "bar" : "d"
      },
      "B" : {
         "" : "w"
      }
   },
```

Több dimenziós

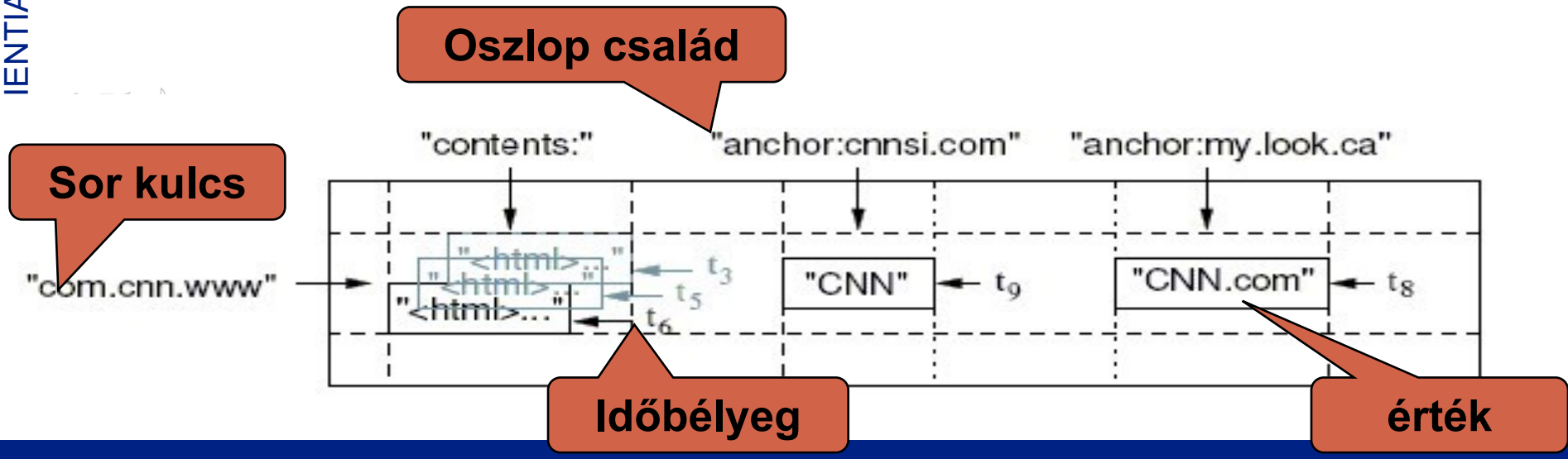
```
{
   "1" : {
      "A" : "x",
      "B" : "z"
   },
   "aaaaa" : {
      "A" : "y",
      "B" : "w"
   },
   "aaaab" : {
      "A" : "world",
      "B" : "ocean"
   },
   "xyz" : {
      "A" : "hello",
      "B" : "there"
   },
   "zzzzz" : {
      "A" : "woot",
      "B" : "1337"
   }
}
```

Verziózott

```
{
   // ...
   "aaaaa" : {
      "A" : {
         "foo" : {
            15 : "y",
            4 : "m"
         },
         "bar" : {
            15 : "d",
         }
      },
      "B" : {
         "" : {
            6 : "w"
            3 : "o"
            1 : "w"
         }
      }
```

Adat modell

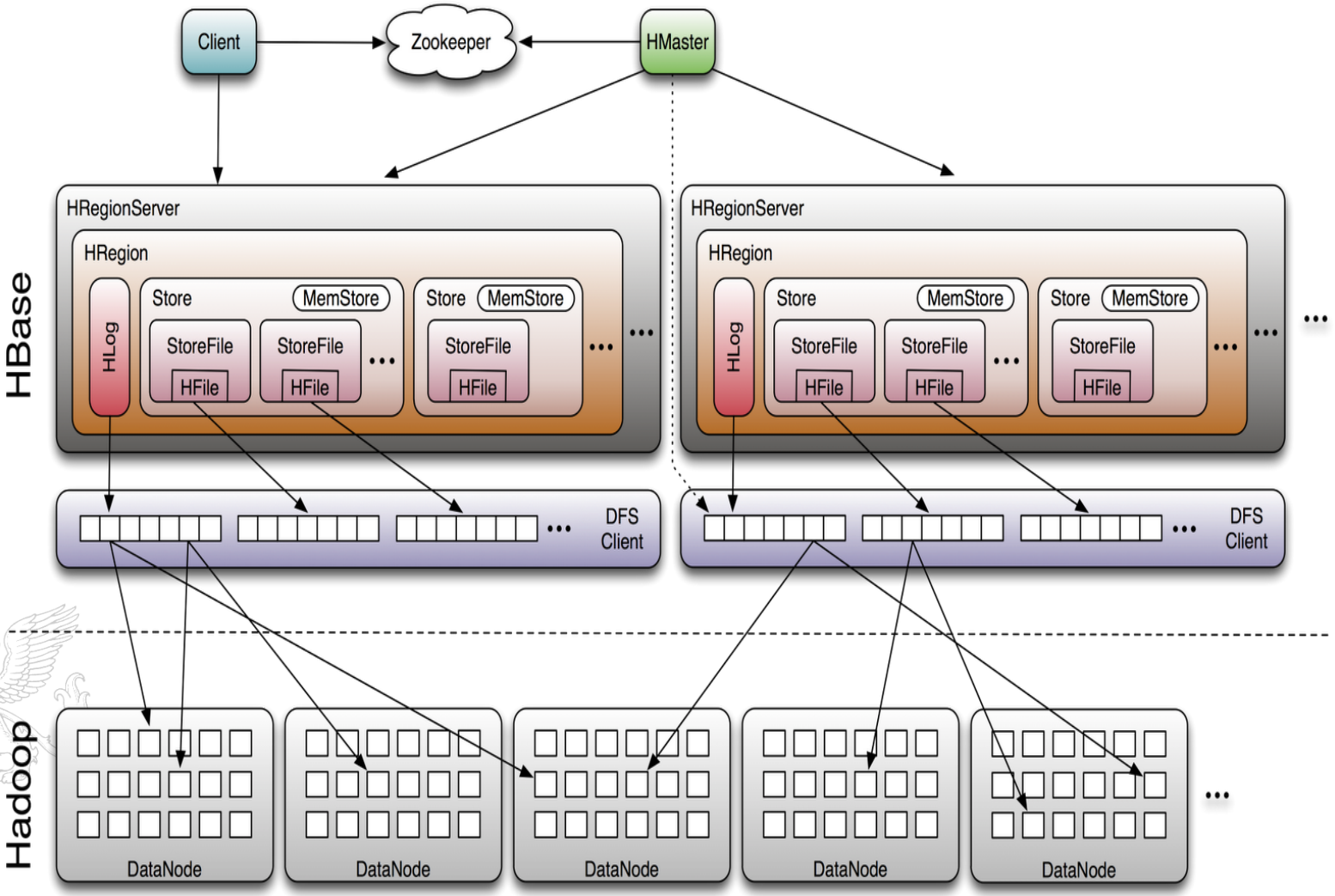
- A táblák a sor alapján rendezettek
- A tábla séma az oszlop családokat adja meg
  - Minden család tetszőleges számú oszlopot tartalmazhat
  - Minden oszlop időbélyeggel bír
  - Adott oszlop csak akkor létezik, ha van értéke, a null engedélyezett
  - Egy családon belül az oszlopokat közösen tárolják
- A tábla neveken kívül minden byte[]
- (Row, Family: Column, Timestamp) → Value



Architektúra

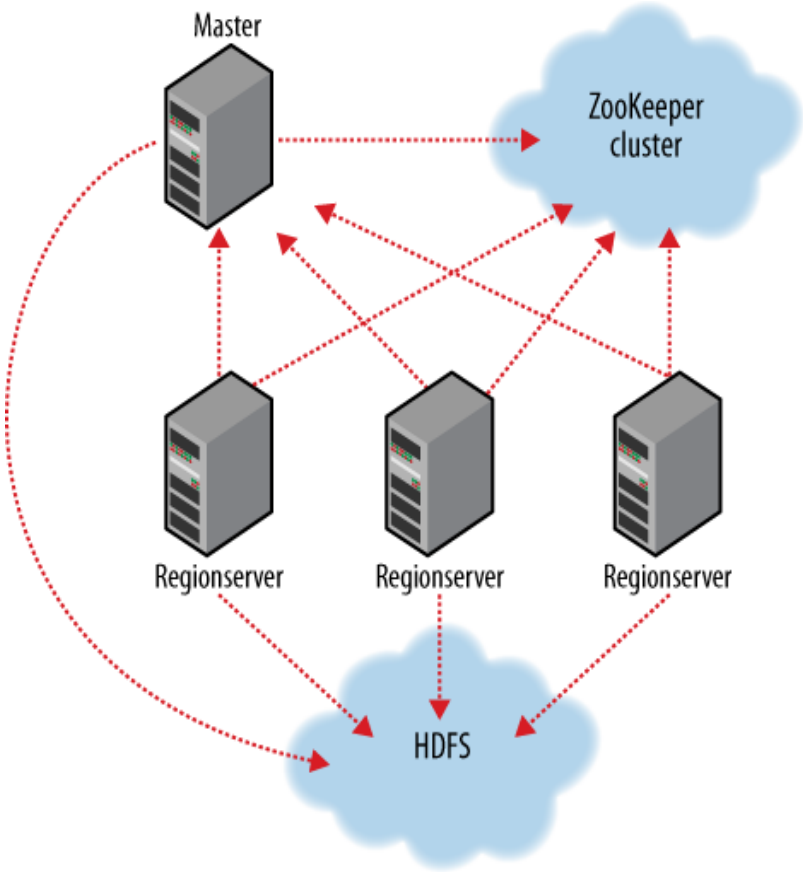
- *Mester*
  - A régió szervereket monitorozza
  - Terhelést elosztást végez a régiók között
  - A klienset a működő régió szerverhez irányítja
- *Régiószerver / szolga*
  - A kliensek kéréseit szolgálja (Írás/Olvasás/Scan)
  - Életjelet ad a mesternek
  - A sávszélessége a Régiók száma a régió szerverek számával skálázódik

Architektúra

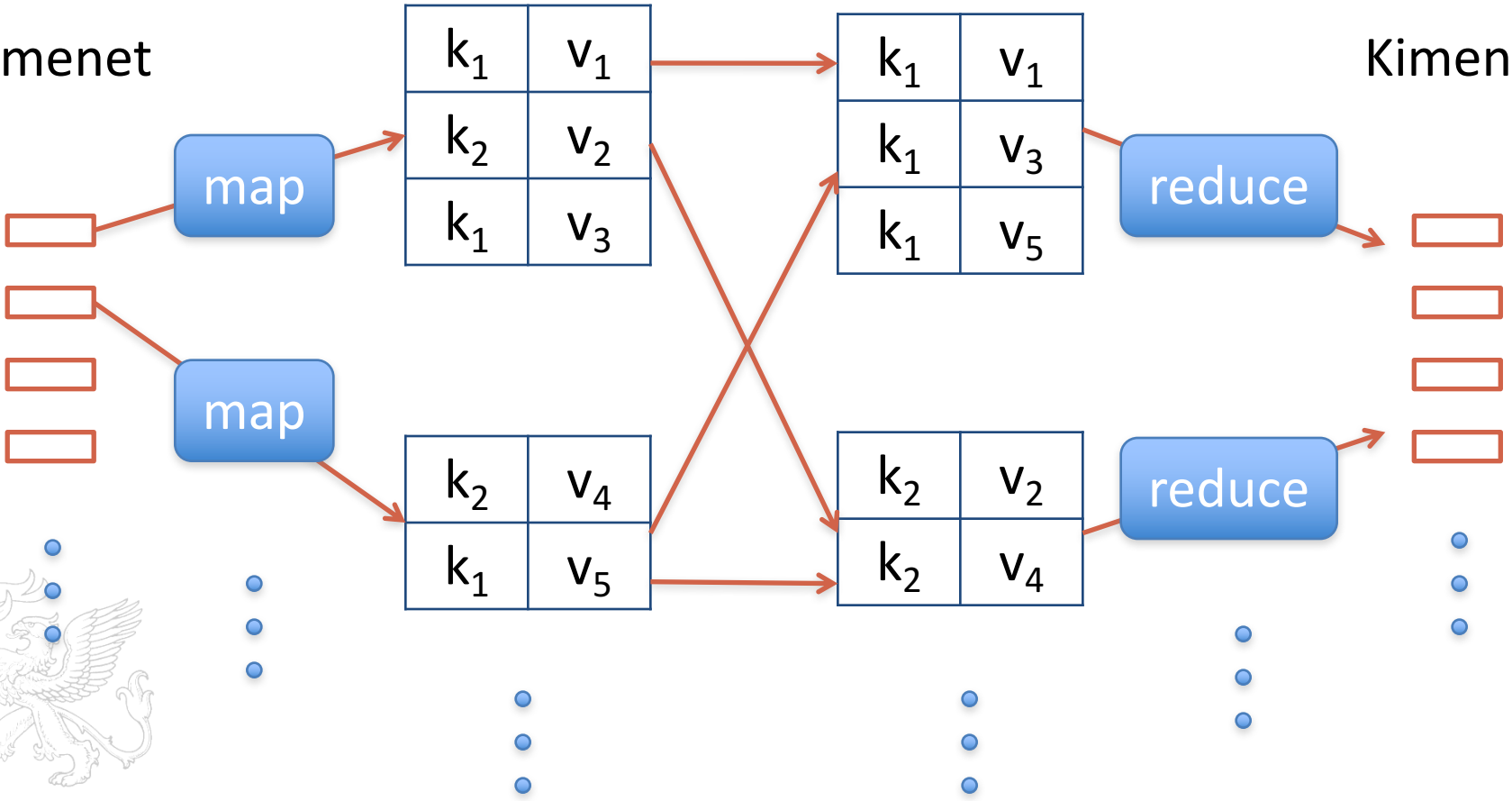


ZooKeeper

- HBase a kritikus adatait a ZooKeeper klaszter segítségével menedzseli
- Paxos döntő testület

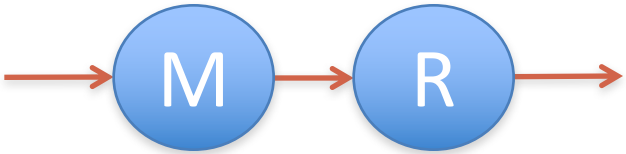


Map-Reduce

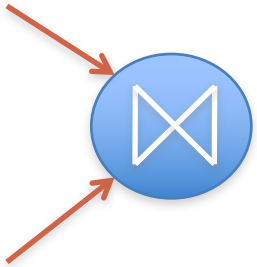

Bemenet
map
map
k_1 v_1
k_2 v_2
k_1 v_3
k_2 v_4
k_1 v_5
k_1 v_1
k_1 v_3
k_1 v_5
k_2 v_2
k_2 v_4
reduce
reduce
Kimenet

Problémák

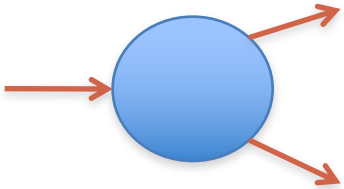
1. Nagyon rögzített adatfolyam



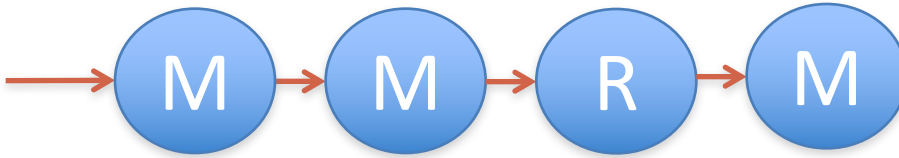
Más folyamok állandóan hackeltek



Join, Union



Split



Chains

2. Sok művelete kézzel kell kódolni
   - Join, filter, projection, aggregates, sorting, distinct
3. A szemantika a map-reduce funkciók mögé rejtőzik
   - Nehéz karbantartani és finomhangolni

Minta adat analízis

Keressük meg minden kategória
10 legnépszerűbb oldalát

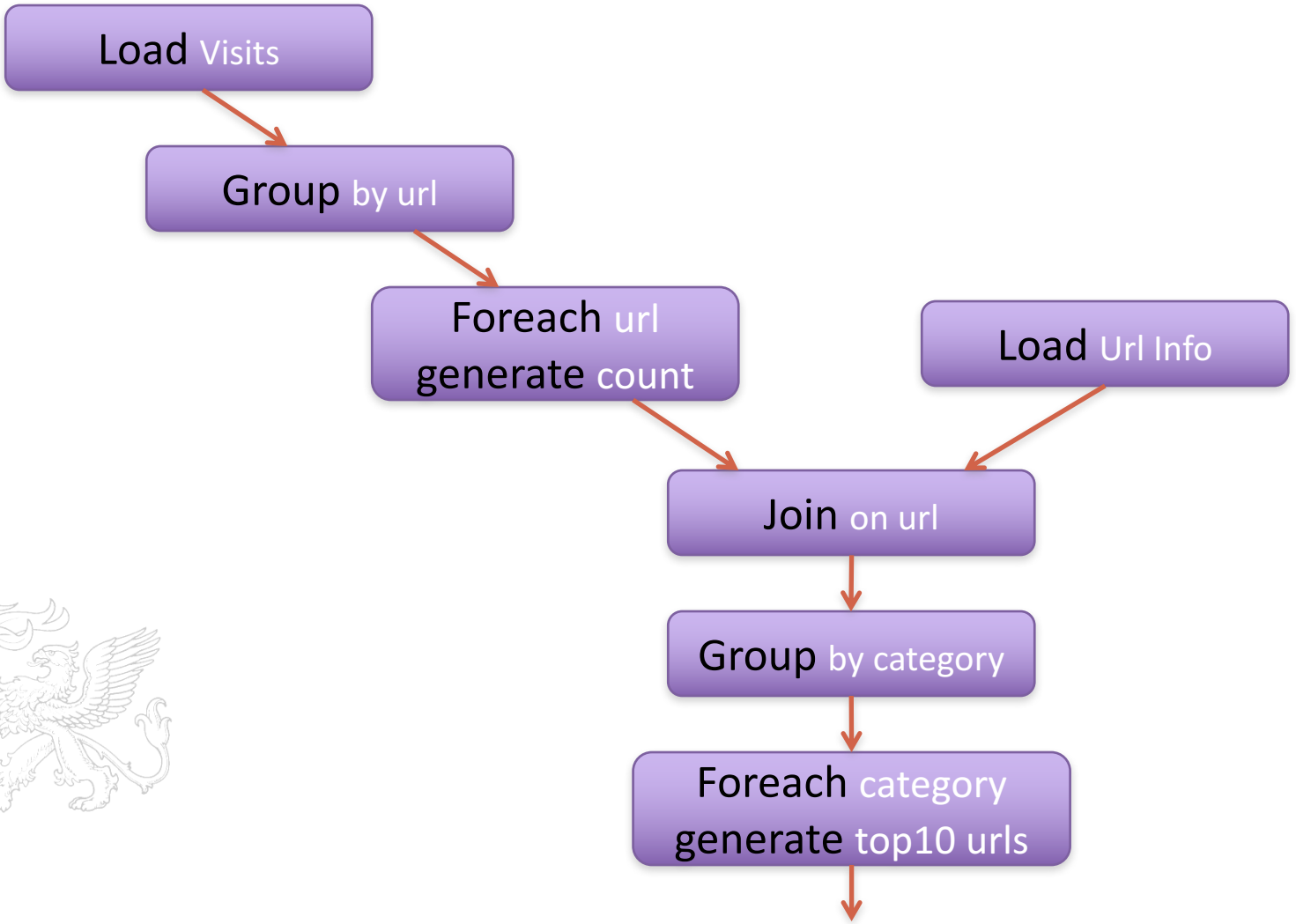
Visits

User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

Url Info

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

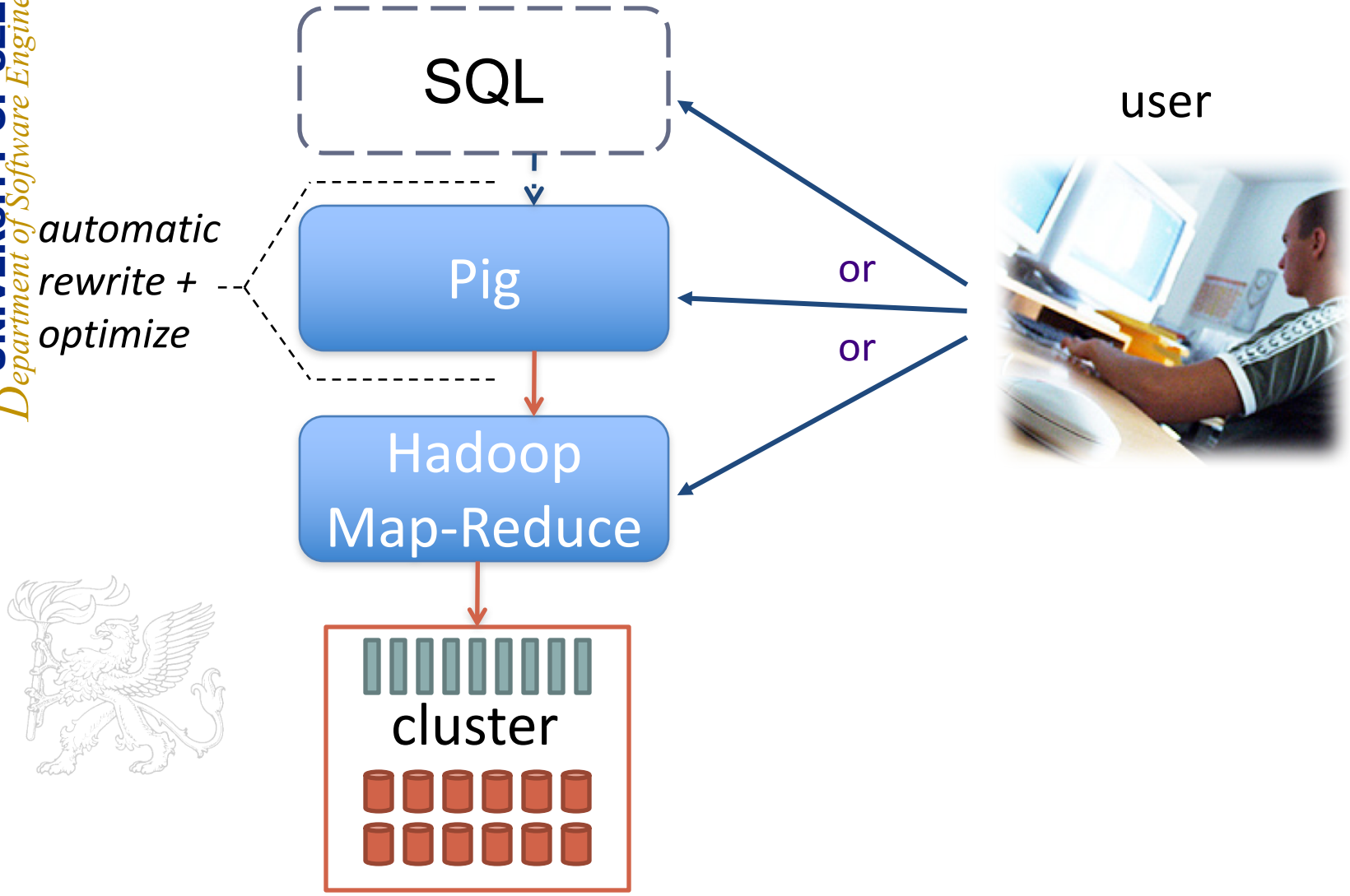
Adatfolyam


Load Visits
Group by url
Foreach url
generate count
Load Url Info
Join on url
Group by category
Foreach category
generate top10 urls

Pig Latin-ban

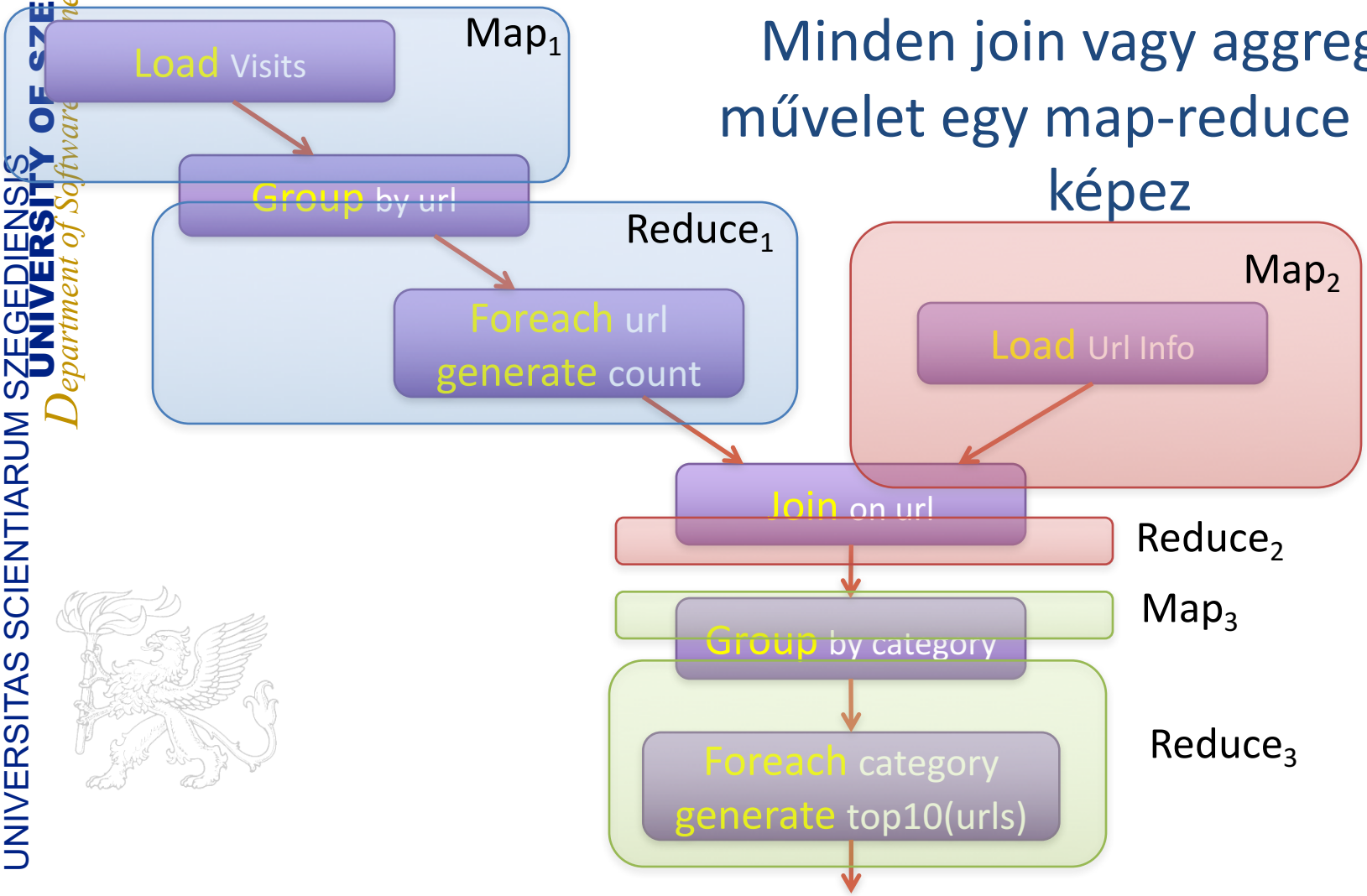
```
visits      = load '/data/visits' as (user, url, time);
gVisits     = group visits by url;
visitCounts = foreach gVisits generate url,
    count(visits);

urlInfo     = load '/data/urlInfo' as (url, category,
    pRank);
visitCounts = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate
    top(visitCounts,10);

store topUrls into '/data/topUrls';
```

Megvalósítás



Map-Reduce-ba fordítás

Minden join vagy aggregálás művelet egy map-reduce határt képez



Hive

- Nyílt forráskódú Hadoop alapú DW megvalósítás
- SQL szerű deklaratív nyelv - HiveQL
- Saját Map-Reduce szkriptek is beágyazhatóak

Hive Adatbázis

- Adat modell
  - Táblák
    - A relációs adatbázis tábláival analóg megoldások
    - Minden tábla egy HDFS könyvtár
    - Az adat fájlokba van sorosítva
    - Más külső adattárakat is támogat (NFS, …)
  - Partíciók
    - Egy táblának egy vagy több partíciója lehet, ez határozza meg az alkönyvtárak használatát
  - Vödrök
    - Az adat a partíción belül vödrökbe van osztva egy adott oszlop kivonata alapján. Minden vödör egy egy fájlba van tárova

hiveQL

- SQL szerű lekérdező nyelv többek között select, join, aggregate, union all és sub-query megoldásokat
- DDL lekérdezéseket is támogat, partíció és vödör definiálással
- Példa adat beszúrás:

```
LOAD DATA LOCAL INPATH '/logs/status_updates'
INTO TABLE status_updates PARTITION (ds='2009-03-20')
```

- Nem támogatja az UPDATE és DELETE parancsokat
- Több táblás INSERT

```
FROM (SELECT a.status, b.schoold, b.gender
        FROM status_updates a JOIN profiles b
          ON (a..userid = b.userid)
          and a.ds='2009-03-20')
) subq1
INSERT OVERWRITE TABLE gender_summary PARTITION (ds='2009-03-20')
  SELECT subq1.gender,COUNT(1) GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary PARTITION (ds='009-03-20')
  SELECT subq1.school, COUNT(1) GROUP BY subq1.school
```

HIVE Architektúra