

Napredni modeli i baze podataka

predavanja
siječanj 2016.

NoSQL
3/3



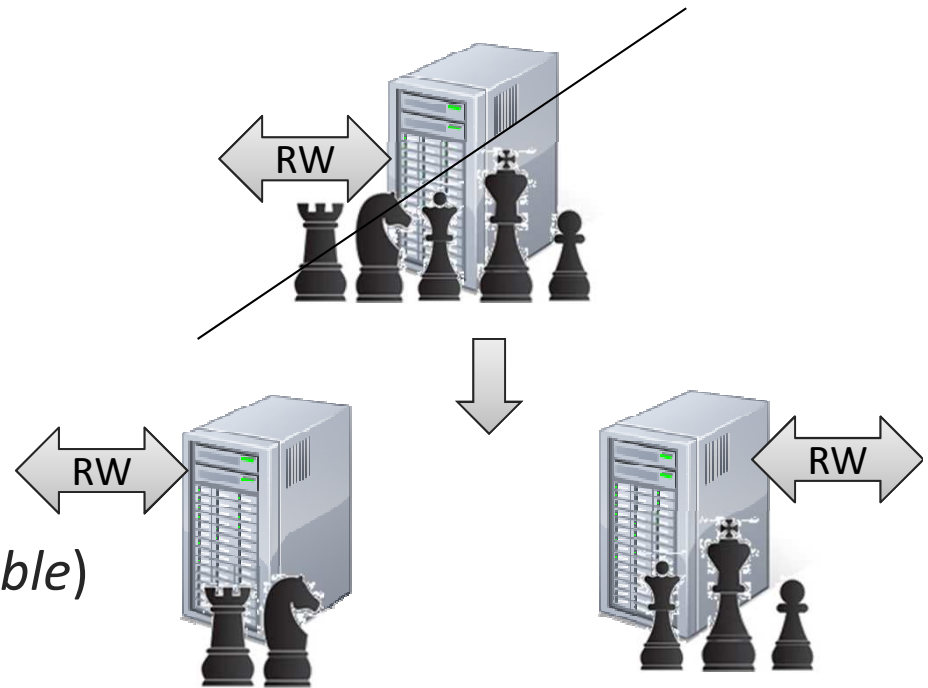
Distribucijski modeli

Distribucija podataka

- Popularnost NoSQL sustava ~ rad na klasteru
- Distribucija:
 - ✓ Obradivanje veće količine podataka
 - ✓ Veći R/W promet
 - ✓ Veća dostupnost
 - ✗ Složenost, novi problemi
- Dva načina distribucije:
 - Fragmentacija (*sharding*)
 - Replikacija (*replication*)
- Najbolji: bez distribucije 😊

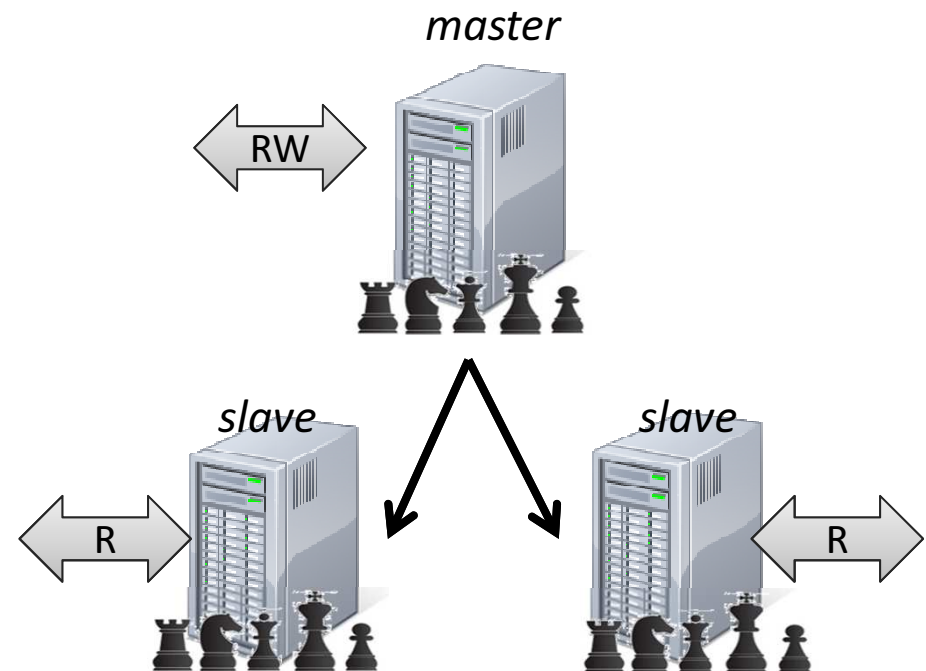
Fragmentacija

- Združiti podatke kojima se često zajedno pristupa - agregati
- Kako raspodijeliti po serverima?
 - Geografski
 - Jednoliko
 - Domenska pravila...
(npr. domenska imena kod *BigTable*)
- Auto-fragmentacija - BP obavlja fragmentaciju
- Poboljšava i čitanje i pisanje
- Ne poboljšava otpornost sustava na pogreške, čak suprotno (treba održavati više strojeva!)
- Kada se upustiti u fragmentaciju (odmah ili kasnije)?



Replikacija: *Master-Slave*

- ✓ Korisno za skaliranje kada imamo **puno čitanja**
 - *Read resilience* - ako *master* prestane raditi i dalje se može čitati
- ✓ Brz oporavak - odabir novog *mastera* (~ *hot backup*):
 - Ručno (konfiguracija)
 - Automatski ("izbori")
- *Read resilience* - različite konekcije za R i W (kako?)
- ✗ Nekonzistentnost (što ako *master* propagacija prestane raditi?)

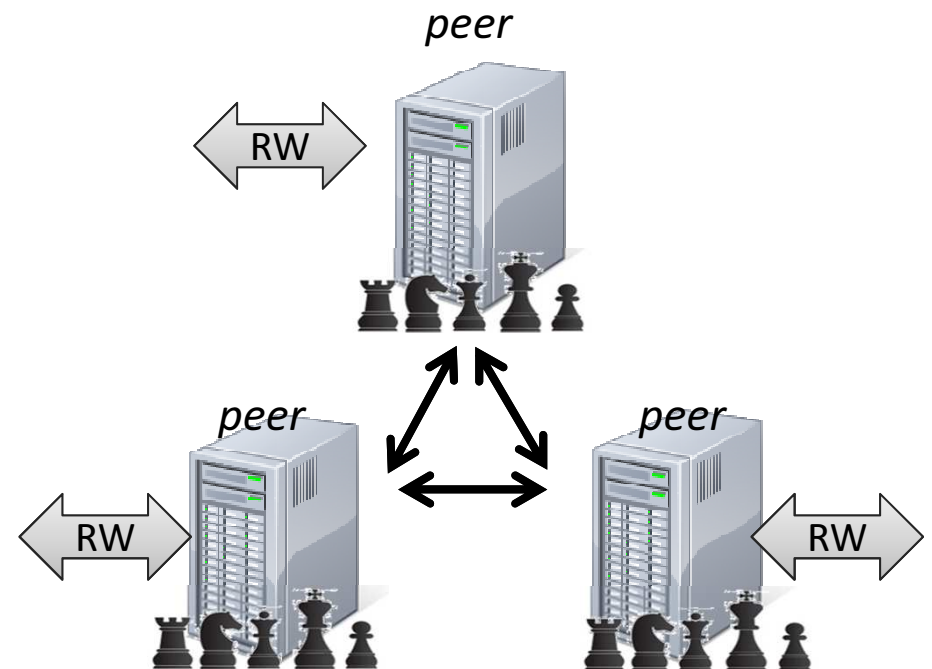


Replikacija: *Peer-to-Peer*

- ✓ Korisno za skaliranje
i za **čitanje** i za **pisanje**
- Ravnopravni čvorovi
- ✓ Lako poboljšati performanse
- dodati čvor!

✗ NEKONZISTENTNOST :

- ✗ R (isto kao i kod MS), uglavnom tranzijentno
- ✗ WW konflikt - *inconsistent writes are forever* ☹



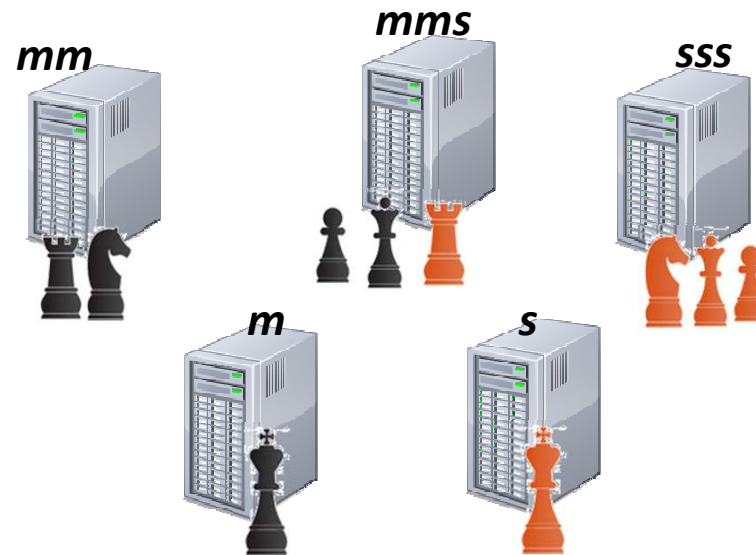
konzistencija

preformanse

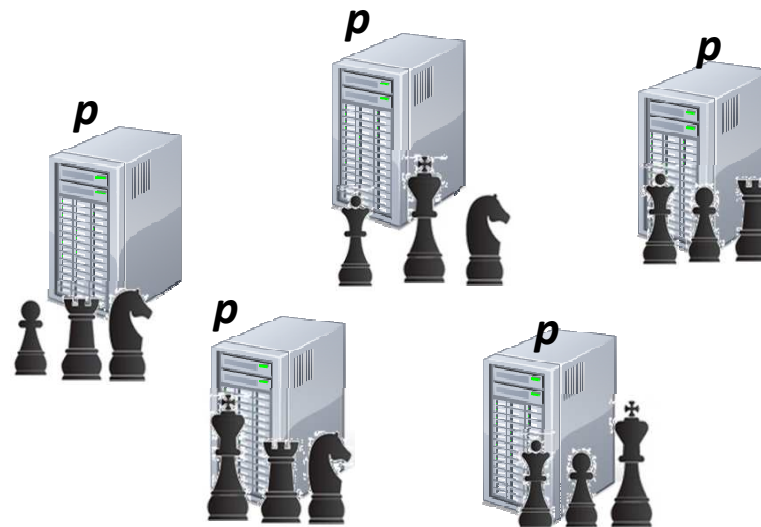


Fragmentacija + replikacija

- $MS+R=1$



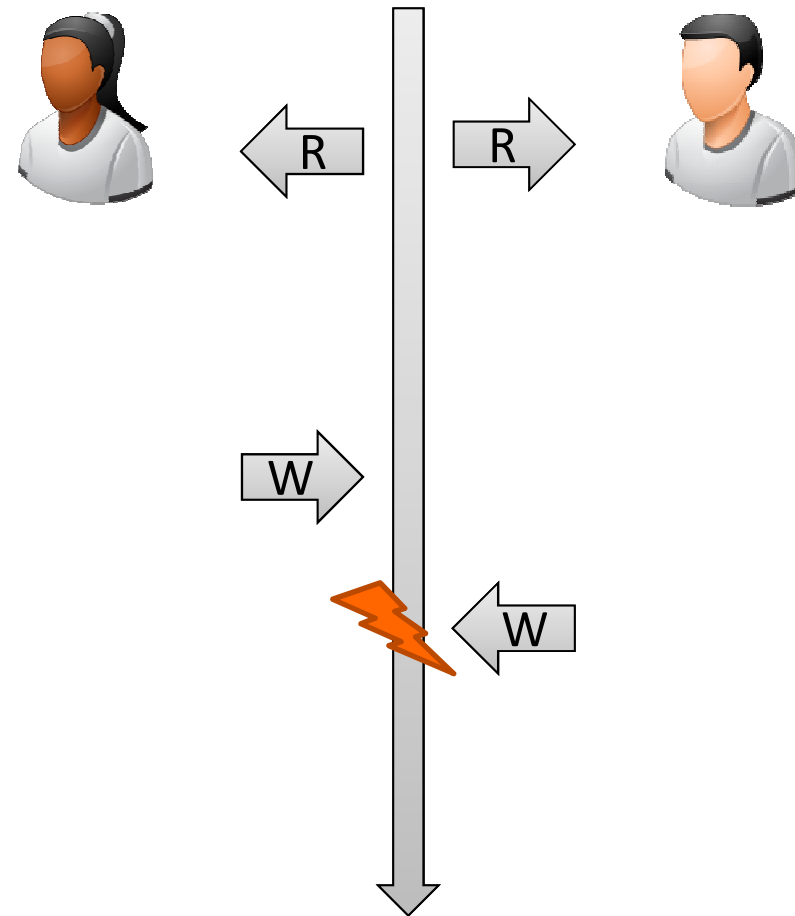
- $P2P+R=3$



Konzistencija (*Consistency*)

Konzistencija kod pisanja - primjer (1)

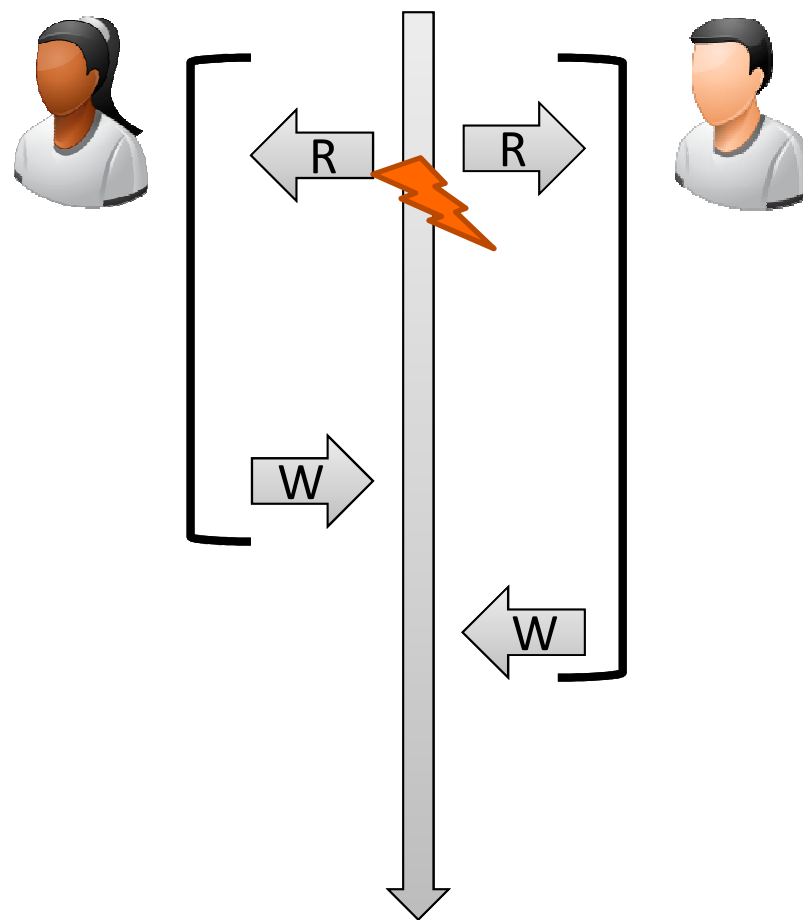
- RBP - *strong consistency*
- Svejedno, mogući problemi, primjer:



- Rješenja?

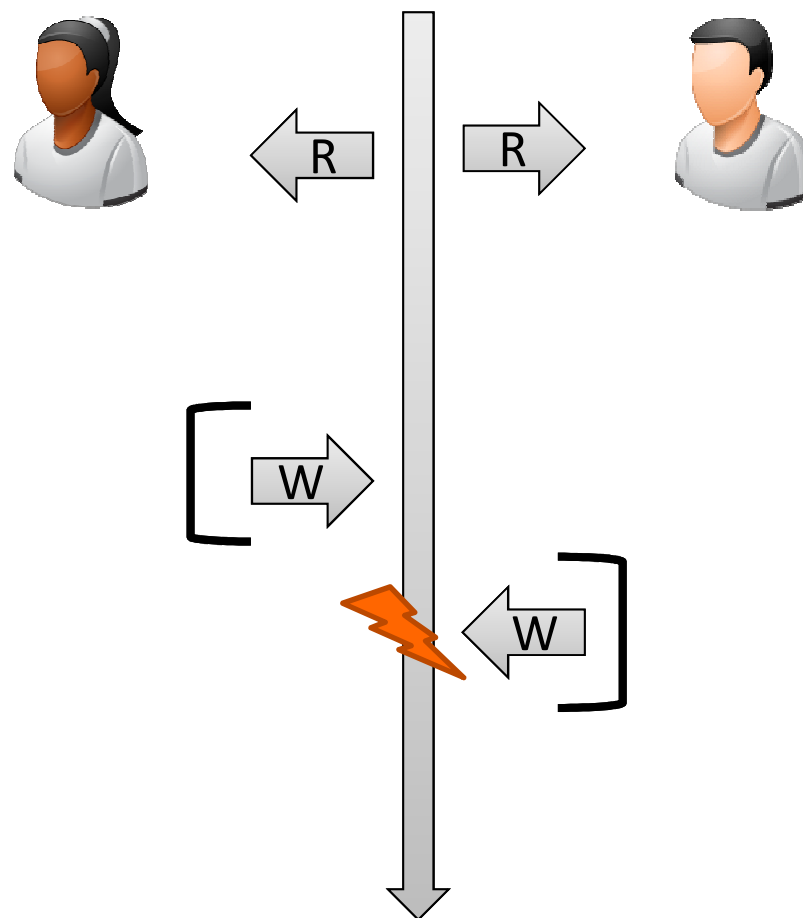
Konzistencija kod pisanja - primjer (2)

- ✓✗ Rješenje:
transakcije (performanse?)
- ✓✗ Pogodno za manji broj
korisnika



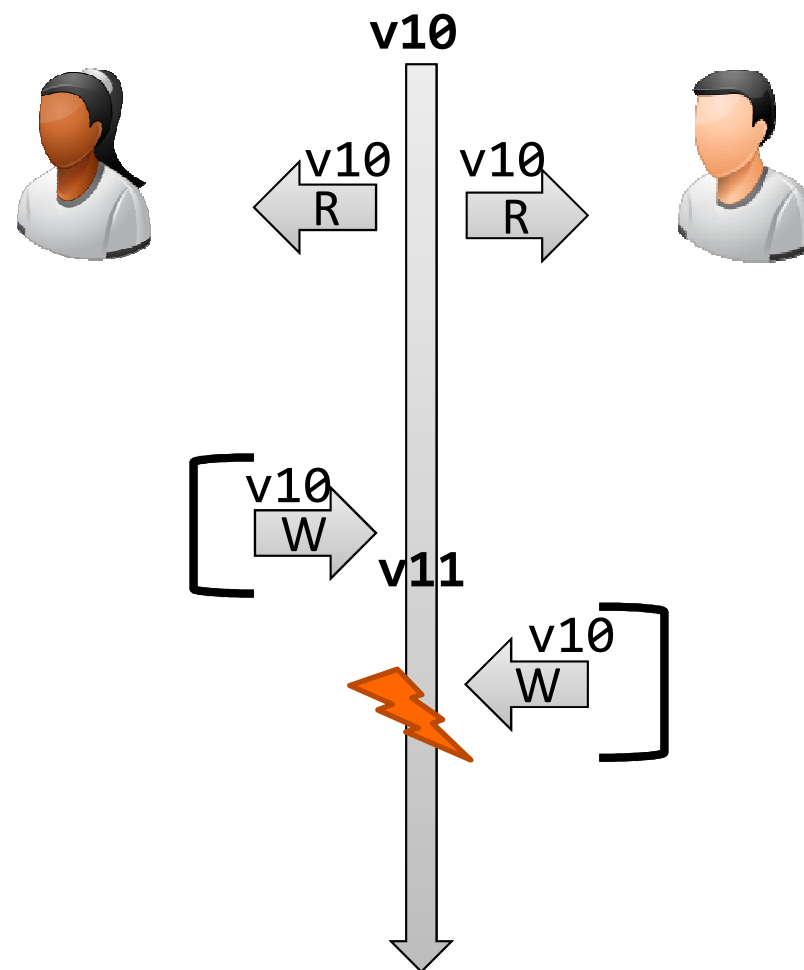
Konzistencija kod pisanja - primjer (3)

- ✓✗ Rješenje:
transakcije (performanse?)
- ✗ I dalje WW konflikt



Konzistencija kod pisanja - primjer (4)

✓✗ Bolje rješenje: *Offline locks*
(tj. verzije, *conditional update*)

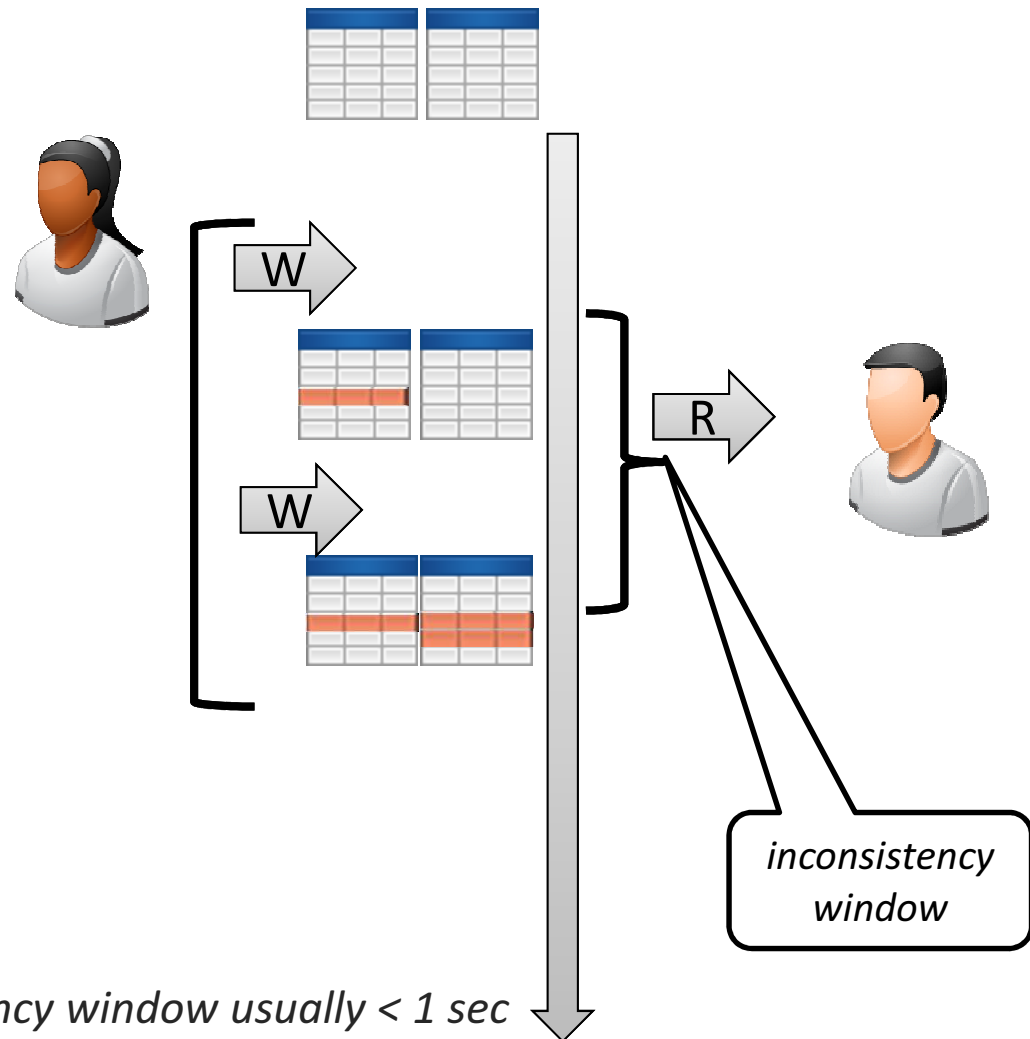


Konzistencija kod pisanja

- Npr. dva korisnika mijenjaju istu informaciju na njihovim zajedničkim web stranicama – *lost update*
- Dva pristupa:
 - **Pesimistični** - spriječiti WW konflikt (*write locks*)
 - **Optimistični** - dopustiti, detektirati, razriješiti (npr. *conditional update* (prethodni primjer), *automatic merge* ~ CVS)
- Oba pristupa se oslanjaju na konzistentnom poretku akcija
 - Kod jednog servera trivijalno – odabire se jedan ili drugi update
 - P2P? – različite vrijednosti
- Ali kako u distribuiranoj okolini?
 - Jedno rješenje - W preko samo jednog čvora
 - Ili: dopustiti više verzija, pa kasnije razriješiti
- Konflikte treba razriješiti

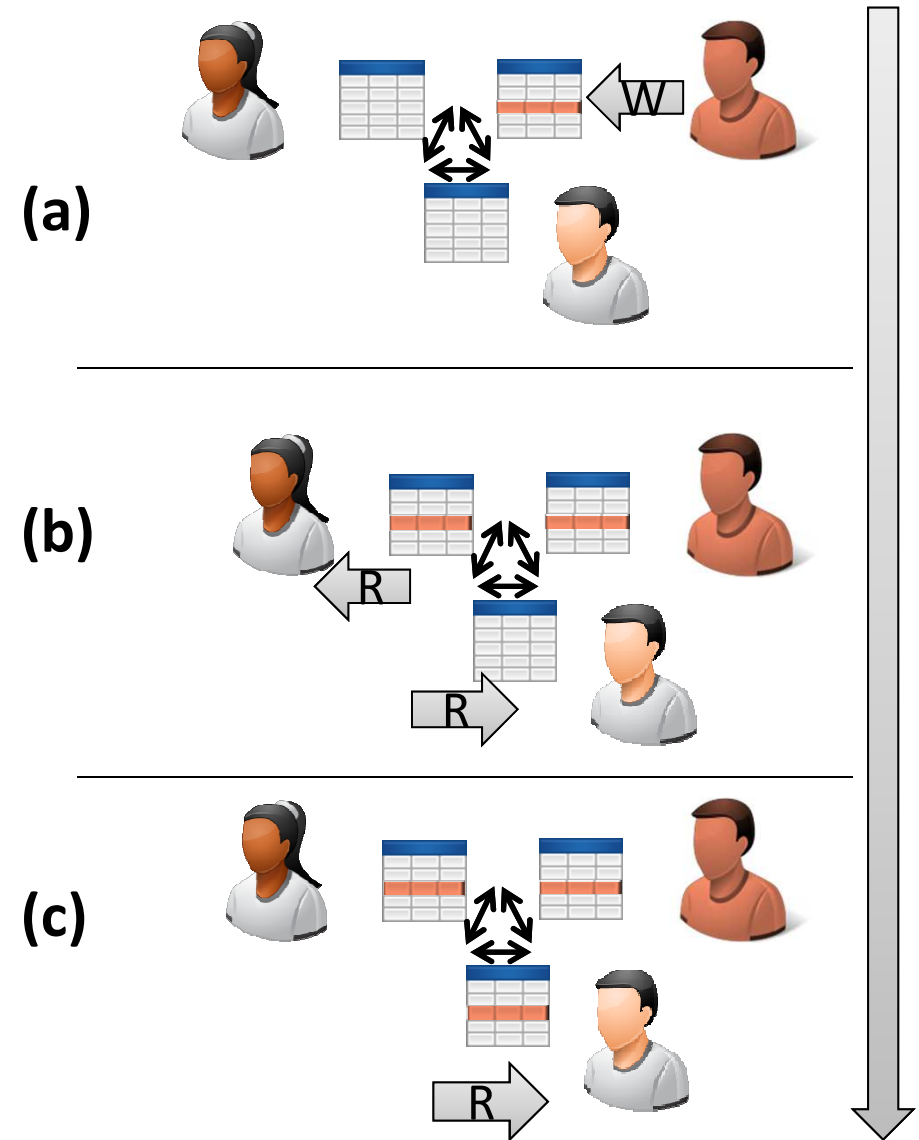
Konzistencija kod čitanja (1)

- **Logička (ne)konzistencija**
= osigurati se da različiti objekti zajedno imaju smisla
- Primjer:
inconsistent read
ili *RW conflict*
- ✓ RBP to rješavaju transakcijama
- ✓✗ NoSQL donekle modelom podataka (agregati)
- Npr. *Amanzon SimpleDB inconsistency window usually < 1 sec*



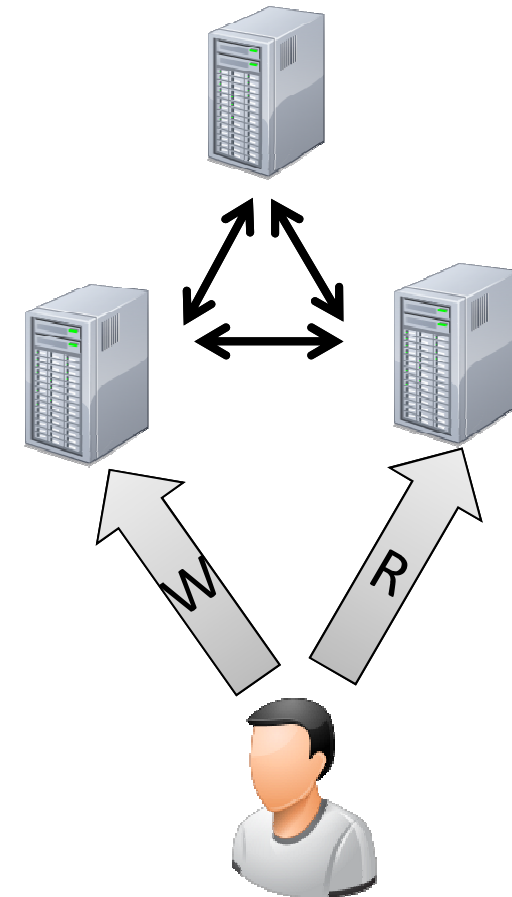
Konzistencija kod čitanja (2)

- **Replikacijska (ne)konzistencija**
= osigurati da sve **replike istog podatka** imaju istu vrijednost
- Nekonzistencija u (b)
- (c) - "*Eventually consistent*"
- Neovisna o logičkoj, ali:
 - Sama replikacija može produžiti *inconsistency window* logičke konzistencije
- Konzistencija nije globalno svojstvo aplikacije, obično se može postavljati *per request*: nekad slaba, nekad jaka



Konzistencija kod čitanja (3)

- Replikacijska (ne)konzistencija
- Primjer: *blog post*
- Read-your-writes consistency
- Rješenje:
 - *Sticky session, session affinity*
(može li kod MS S preuzeti posao W?
-> Iznimke ☹)
 - *Version stamps* (= server se mora pobrinuti da „nabavi” verziju podataka koje klijent traži)



Oslabljivanje konzistencije

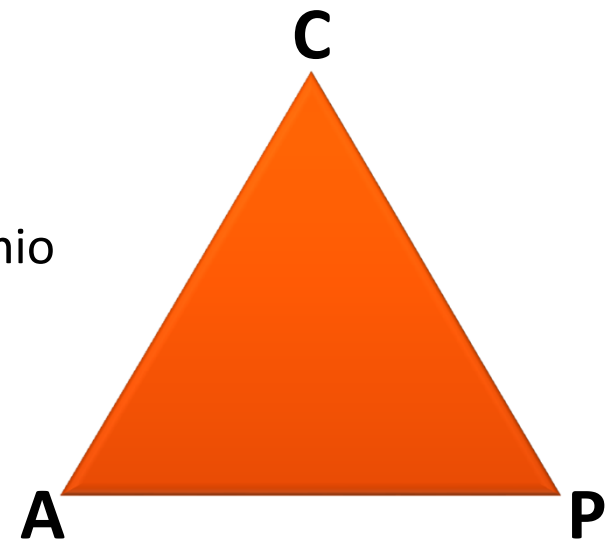
- Kompromis između performansi sustava i konzistencije
- Čak i RBP to mogu (rade) - razine izolacije
- Npr. MySQL je bio popularan prije nego što je podržavao transakcije
- *eBay*
- *Amazon*

CAP teorem

- E. Brewer, 2000.: *Towards Robust Distributed Systems*
- Dokazan 2002. - *Lynch & Gilbert*
- **Consistency, Availability, Partition tolerance**

U distribuiranim sustavima je moguće ostvariti samo **dva od tri** navedena svojstva.

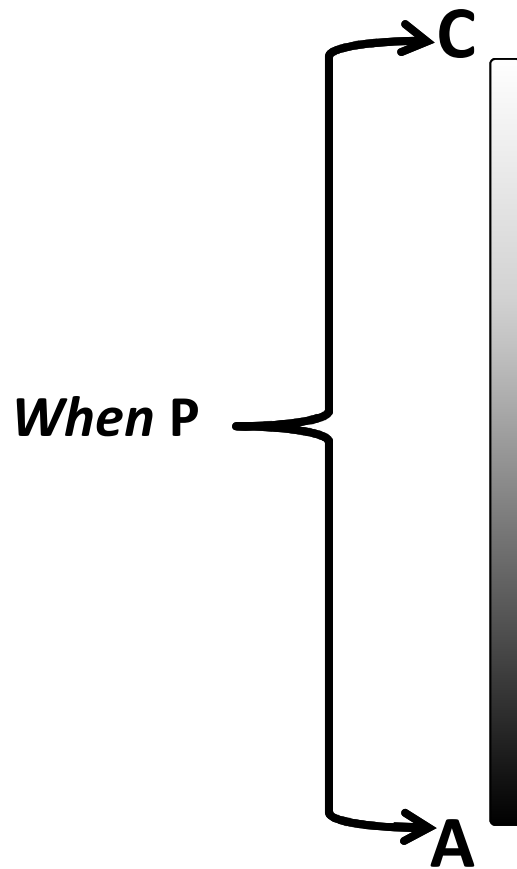
- **Consistency** (Cap <> aCid):
 - Svaki odgovor poslan klijentu je točan
- **Availability**
 - Svaki zahtjev koji je funkcionirajući server zaprimio mora rezultirati odgovorom (R & W)
- **Partition tolerance**
 - Rad sustava i u uvjetima kada nastanu izolirane skupine računala



Za "one koji žele znati više": <http://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf>

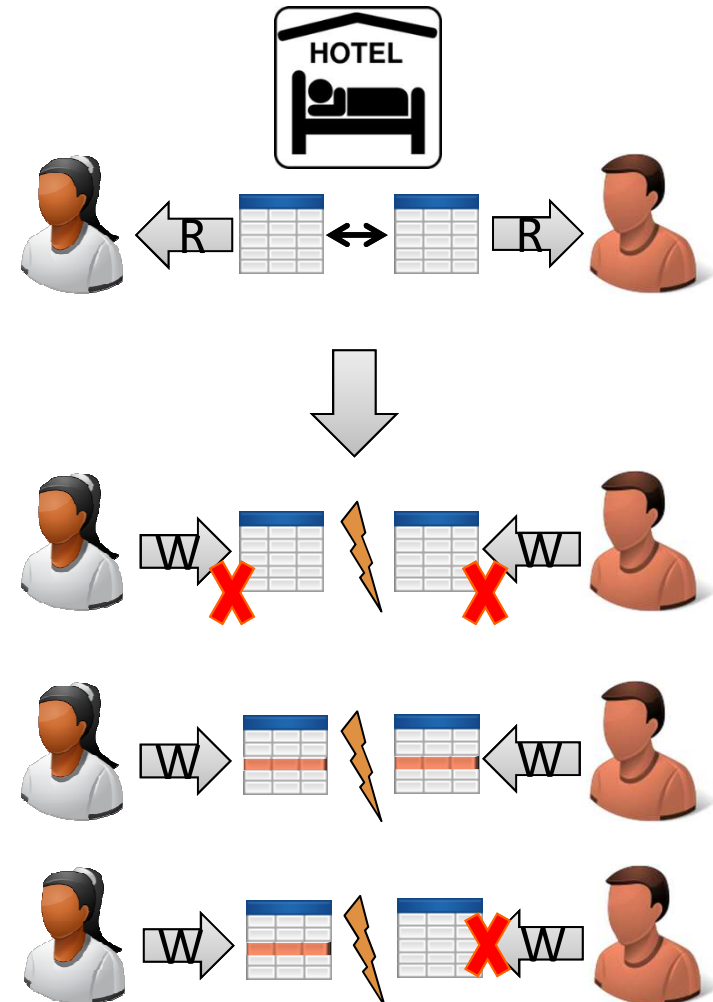
CAP theorem - preformulirano

▪ Primjer:



ili: Response time

(latency), primjer: *Amazon shopping cart*

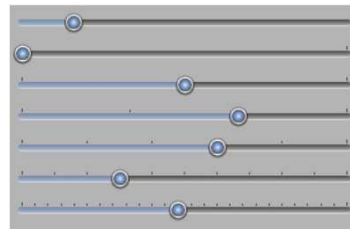


BASE

- Za mnoge primjene, dostupnost i toleriranje particija su važnije od stroge dosljednosti (npr. (velike) web aplikacije, tražilice, ...)
- BASE:
 - *Basically Available* – aplikacija je praktički uvijek dostupna (unatoč povremenim kvarovima)
 - *Soft-state* – ne mora uvijek biti konzistentan („mekano stanje”), sustav se stalno mijenja, protočan je
 - *Eventual consistency* – biti će, u konačnici, u nekom znanom stanju (izmjene će se, u konačnici, propagirati i svi će ih vidjeti)

ACID

Jaka dosljednost
Izolacija
Dostupnost?
Pesimistično (konzervativno)
...



nije ili-ili

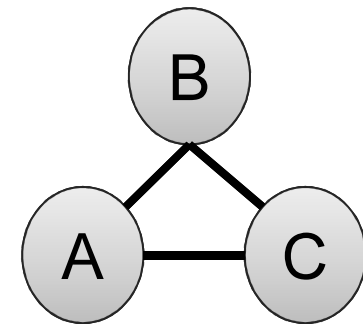
BASE

Slaba dosljednost (zastarjeli podaci)
Prvo dostupnost
Prihvatljivi približni odgovori
Agresivno (optimistično)
...

Kvorum

- $N = 3$ (N je faktor replikacije \leftrightarrow broj čvorova)
- **P2P** model
- Za **konzistenciju kod pisanja** je dovoljna **većina**:

$$W > N/2$$

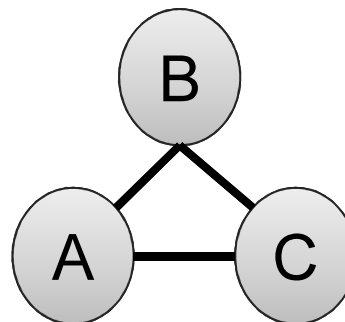


- Konzistencija kod čitanja ovisi o W
- Možemo imati konzistenciju kod čitanja i kad nemamo kod pisanja - čitati sve čvorove!
-> to ne znači da nećemo imati update conflict, ali ćemo ga detektirati
- Za **konzistenciju kod čitanja** je dovoljno:

$$R + W > N$$

CA kompromis - zaključno

- Što više čvorova je uključeno u operaciju:
 - ✓ Bolje konzistencija
 - ✗ Veća latencija
- ... i obratno!
- Nije binarna odluka
- Može se podešavati, čak i na razini zahtjeva!
- Primjer:
 - želimo brzo čitanje
 - žrtvujemo pisanje
 - $N = 3, W = 3, R = 1$



Primjer: Riak (1)



- `n_val` – faktor replikacije (default = 3)
- `r` - broj čitanja nakon kojeg se čitanje smatra uspješnim
- `w` - broj pisanja nakon kojeg se pisanje smatra uspješnim
- Npr. postavljanje parametara bucketa:

```
curl -X PUT http://localhost:8091/riak/animals \  
      -H "Content-Type: application/json" \  
      -d '{"props":{"w":2}}'
```

- Mogu se postaviti kod svakog zahtjeva!

Npr.

```
curl http://localhost:8091/riak/animals/floki \  
      -H "Content-Type: application/json" \  
      -d '{"props":{"r":3}}'
```

- Ili:

```
curl http://localhost:8091/riak/animals/floki?r=3
```

Primjer: Riak (2), kratice

- U Riaku su uveli znakovne kratice za uobičajene vrijednosti:

| Kratika | Definicija |
|---------|---|
| One | 1 |
| All | n_val |
| Quorum | $n_val/2+1$ |
| Default | Ono što je postavljeno za taj <i>bucket</i> |

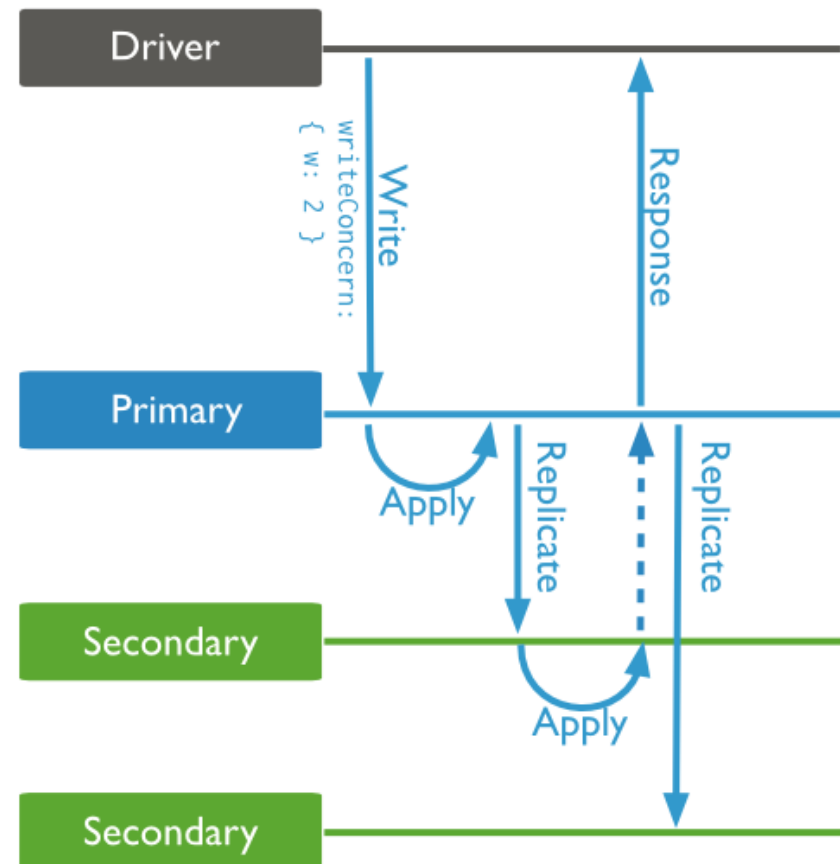
- Npr.

```
curl http://localhost:8091/riak/animals/floki?r=quorum
```


Mongo Write/Read concern

- Opisuje razinu potvrde servera kod obavljanja zadane operacije
- `{ w: <value>, j: <boolean>, wtimeout: <number> }`
 - w – broj instanci (0, 1, „majority”, <tag set>)
 - j: zapisano u dnevnik
 - wtimeout: vremenski limit
- Npr.:

```
db.student.insert(  
  { ime: "Đuro" },  
  { writeConcern:  
    { w: 2, wtimeout: 5000 } }  
)
```
- readConcern:
`{ level: <majority|local> }`



■ Oslabljivanje svojstva trajnosti

- *Relaxing durability*
- Opet, trade-off: *durability vs performance*
- Npr. in-memory BP
- Npr. pohranjivanje user-sessions
- Pojavljuje se (bez da planiramo) i kod replikacije (*replication durability*):
 - *Master* obradi *update*
 - *Master* prestane raditi prije nego je propagirao promjene
 - Slave čvorovi izaberu novi *master* čvor
 - Što se događa kad *master* ponovo proradi?
 - Trade-off: možemo zahtijevati da master obavi barem jednu replikaciju prije nego potvrdi klijentu

Primjer: Riak - Durability



- Kod pisanja:
 - i. Objekt se prvo piše u memoriju (*buffer*)
 - ii. Potvrđuje se uspješno pisanje
 - iii. Objekt se piše na disk
- Potencijalni gubitak podataka između (ii) i (iii).
- Ipak, može se eksplicitno zadati da se (iii) obavi prije (ii) na proizvoljnom broju čvorova, npr. na jednom (*dw = durable write*):

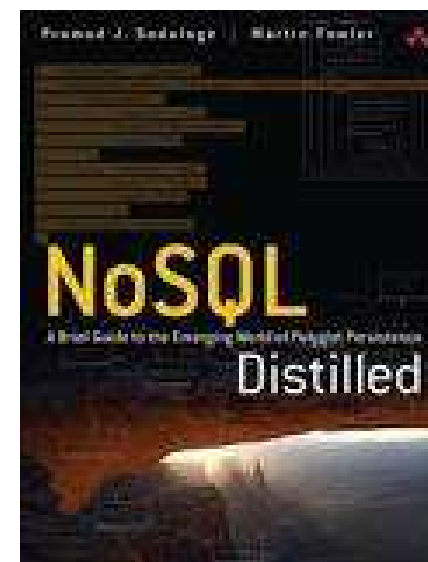
```
$ curl -X PUT http://localhost:8091/riak/animals \
-H "Content-Type: application/json" \
-d '{"props":{"dw":"one"}}'
```

- Ili kod Monga: `w:0`

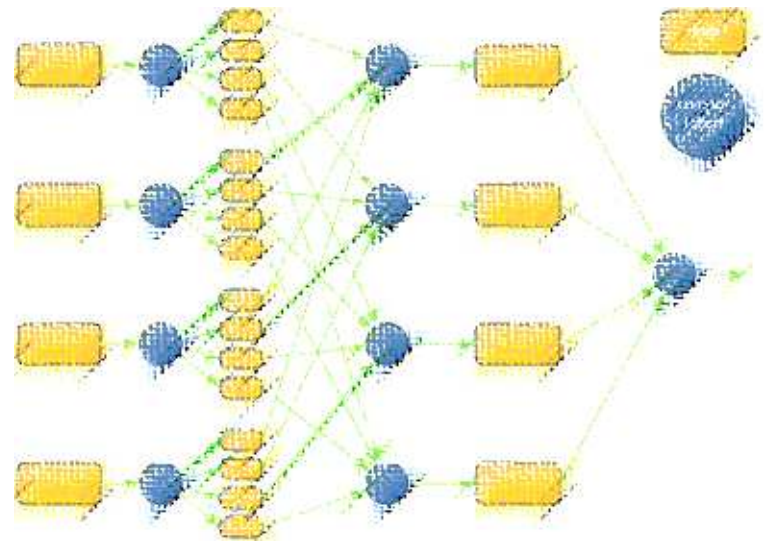
Dodatna literatura

- Posjećamo, predavanja se u velikoj mjeri temelje na:

- NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence
Sadalage, Fowler

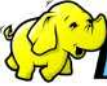


- Posebice za one koji nisu bili na predavanju se preporuča pogledati prezentaciju Martina Fowlera „Introduction to NoSQL”:
 - http://www.youtube.com/watch?v=qI_g07C_Q5I



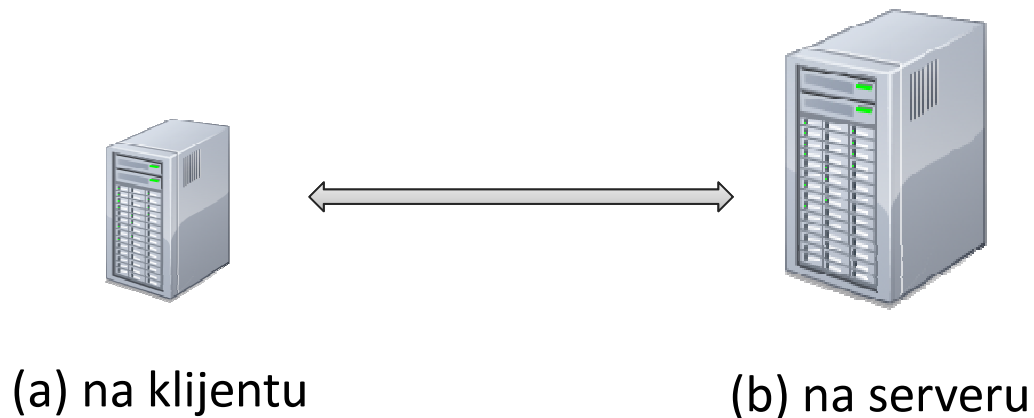
MapReduce

MapReduce

- Mnoge NoSQL baze podataka, bez obzira na tip, omogućuju MapReduce algoritam
- Razvio Google 2004.
-  **hadoop** – (među ostalim i) implementacija *MapReduce*
- Pogodan za određenu vrstu (paralelnih) problema, npr.:
 - pretraživanje velike količine teksta,
 - izgradnja indeksa riječi,
 - brojanje pristupa web stranicama,
 - itd.

MapReduce

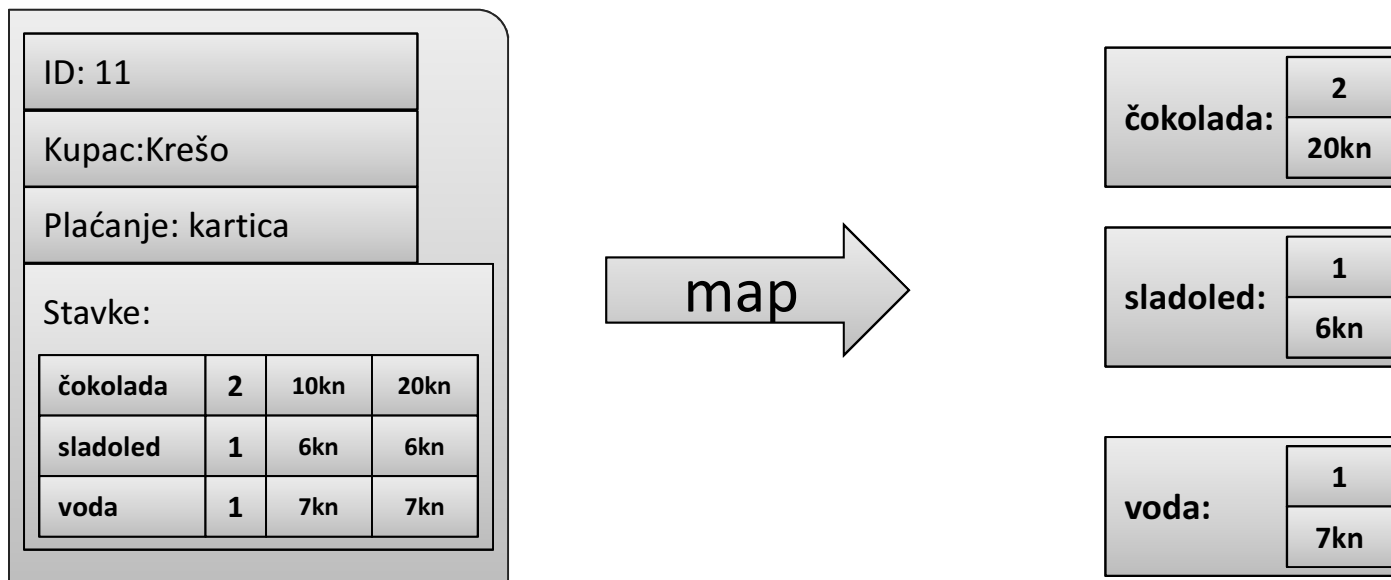
- Klasteri mijenjaju paradigmu **izračunavanja** (a ne samo pohrane)
- Suprotno, opcije izračunavanja na centraliziranoj BP:



- M/R - potpuno drugačija paradigma - izračunavanje na N servera
- Minimizirati mrežni promet, računati na vlastitim podacima
- M/R je *pattern*, implementacije variraju

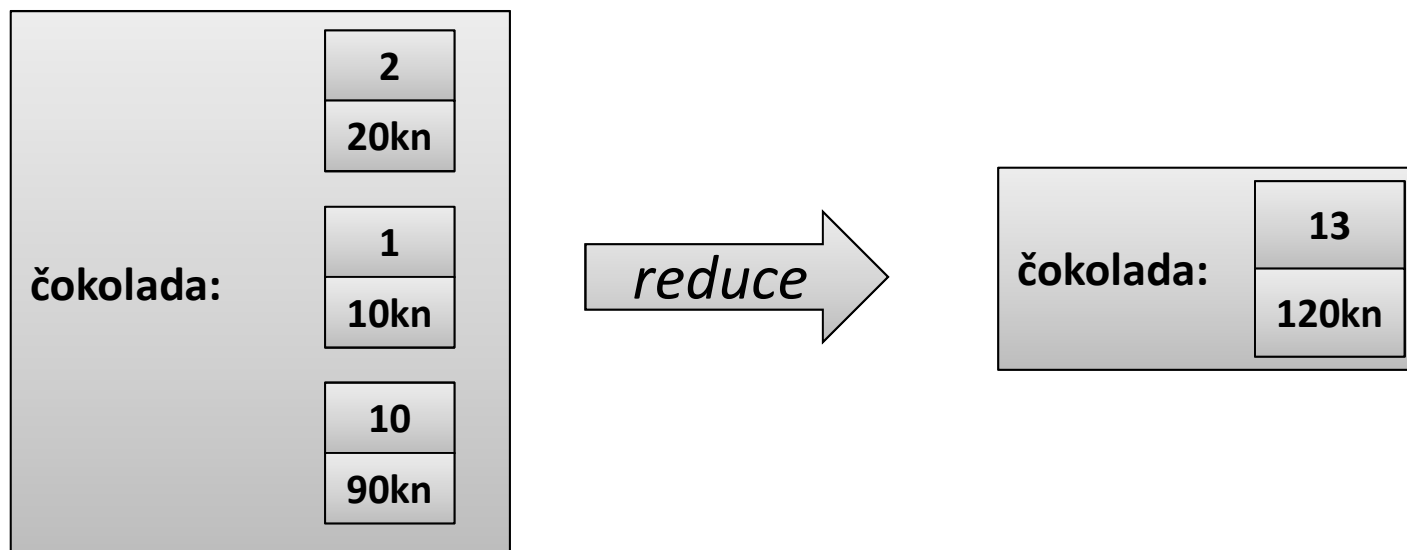
M/R - osnovna ideja: *map*

- Npr. hoćemo vidjeti prihod po proizvodima u zadnjih mjesec dana
- Map je funkcija: **map(k1, v1) → list (k2, v2)**
 - argument: **jedna** ključ-vrijednost (agregat)
 - rezultat: **lista** odnosno 0 ili više (k2,v2) parova
- Map funkcije se obavljaju nezavisno - **paralelizam**



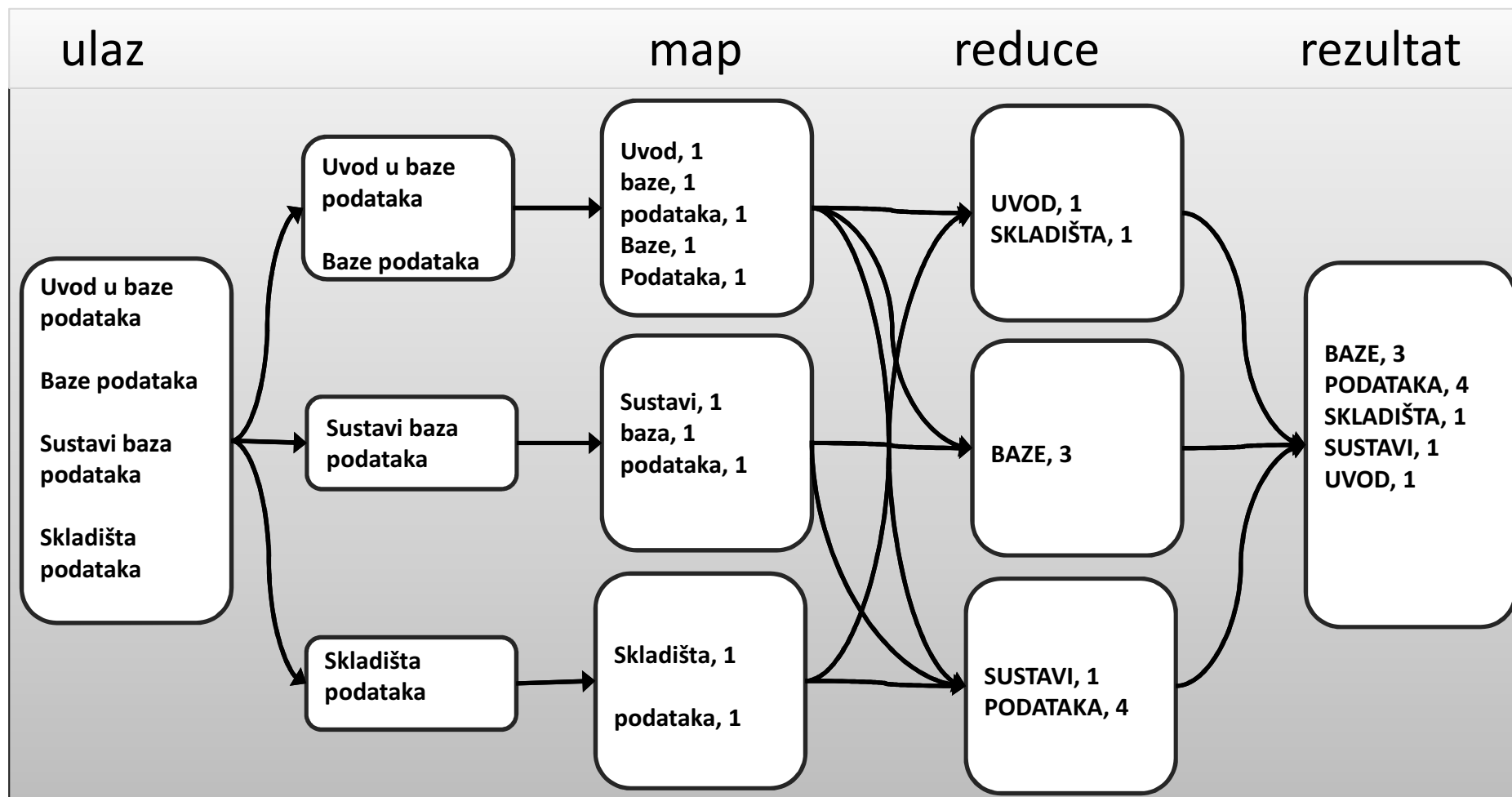
M/R - osnovna ideja: *reduce*

- Reduce je funkcija: **reduce(k2, list(v2)) → list(v3)**
 - argument: **lista vrijednosti** za neki ključ
 - rezultat: 0 ili više vrijednosti, tipično 0 ili 1 vrijednost



M/R primjer

- Prebrojati broj pojavljivanja riječi



M/R primjer: mincemeat

- <https://github.com/michaelfairley/mincemeatpy>

```
import mincemeat
data = ["Humpty Dumpty sat on a wall",
        "Humpty Dumpty had a great fall",
        "All the King's horses and all the King's men",
        "Couldn't put Humpty together again", ]

# The data source can be any dictionary-like object
datasource = dict(enumerate(data))

def mapfn(k, v):
    for w in v.split():
        yield w, 1

def reducefn(k, vs):
    result = sum(vs)
    return result

s = mincemeat.Server()
s.datasource = datasource
s.mapfn = mapfn
s.reducefn = reducefn
results = s.run_server(password="changeme")
print results
```

1. Pokrenuti na serveru:

```
python example.py
```

2. Pokrenuti na klijentu (klijentima):

```
python mincemeat.py -p
changeme [server address]
```

3. Server ispisuje:

```
{'a': 2, 'on': 1, 'great': 1,
'Humpty': 3, 'again': 1, 'wall':
1, 'Dumpty': 2, 'men': 1, 'had':
1, 'all': 1, 'together': 1,
"King's": 2, 'horses': 1, 'All':
1, "Couldn't": 1, 'fall': 1,
'and': 1, 'the': 2, 'put': 1,
'sat': 1}
```

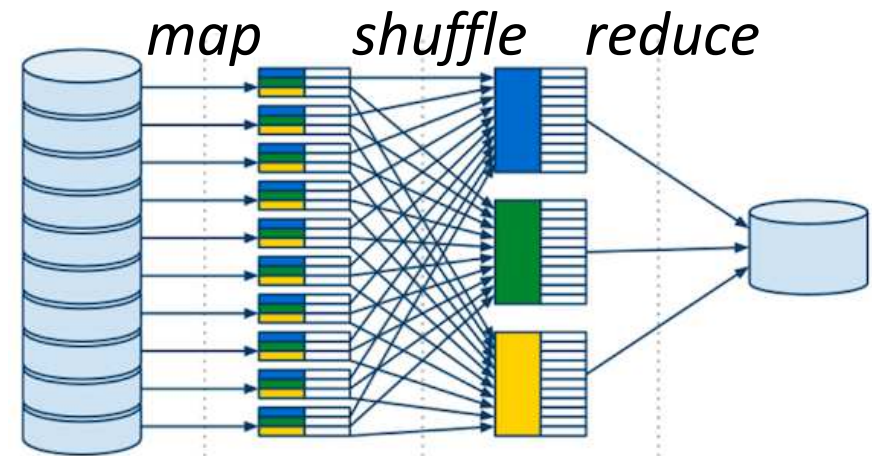
M/R ograničenja

- *Trade-off*:
algoritamska fleksibilnost vs paralelizacija (tj. izračun na klasteru)
- Primijetimo ograničenja:
 - U *map* funkciji možemo djelovati nad samo jednim agregatom
 - U *reduce* funkciji možemo djelovati nad samo jednim ključem

M/R framework

- Orkestrira izvođenje:

- Upravlja čvorovima (isti čvorovi se mogu koristiti i za M i R fazu)
- Paralelno pokreće zadatke
- Upravlja razmjenom podataka
- Oporavak od pogreške (što ako neki čvor prestane raditi?)

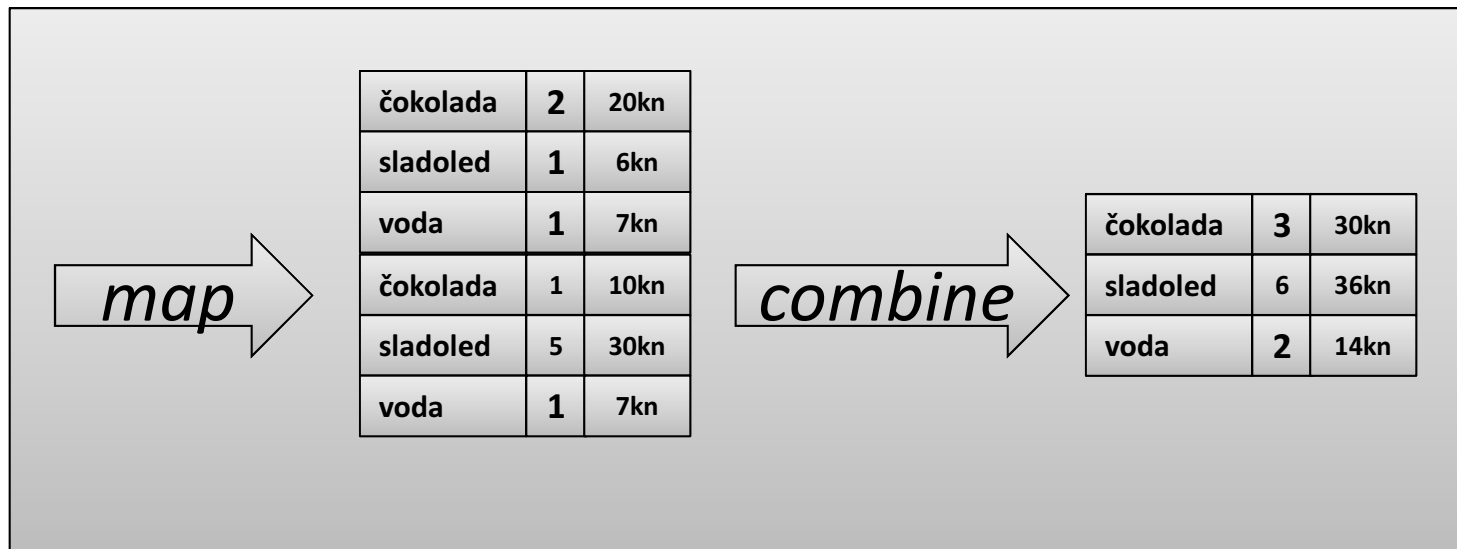


- Ili:

1. Priprema ulazne podatke za *map* funkciju, dodjeljuje i dostavlja *map* čvorovima
2. Pokreće korisničku *map* funkciju na čvorovima
3. Grupira, (particionira) i raspoređuje (*shuffle*) izlaz map funkcija i dostavlja *reduce* čvorovima
4. Pokreće korisničku *reduce* funkciju na čvorovima
5. Objedinjava sve rezultate *reduce* funkcije sa svih čvorova

M/R - *combiner*

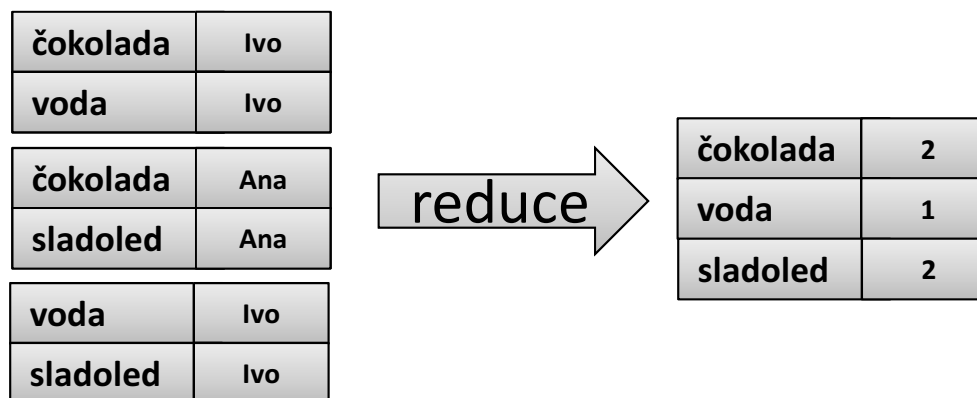
- *combiner* funkcija sažima sve podatke s istim ključem u jedan zapis



- *combiner* ~ *reducer*

CR - *Combinable Reducer*

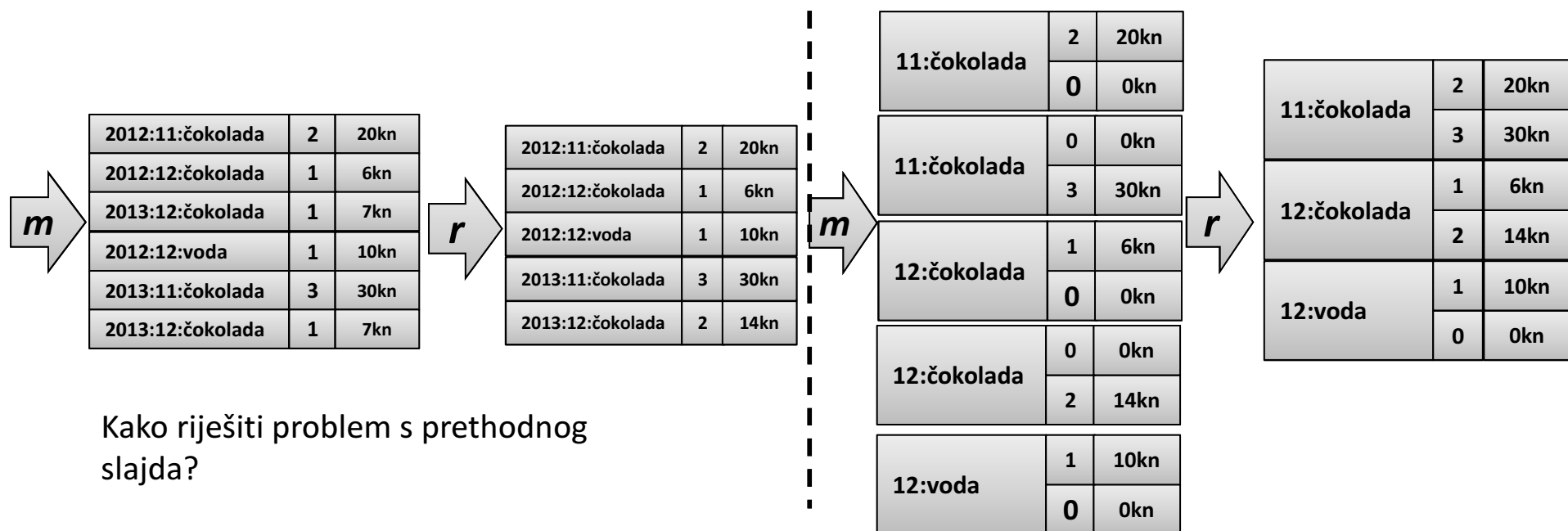
- CR je *reduce* funkcija čiji izlaz odgovara ulazu
- CR:
 - ✓ Zamjenjuje *combiner*, poziva se odmah na istom *map* čvoru
 - ✓ Mogu se pozivati prije nego što je u potpunosti gotova *map* faza
- Neki problemi se ne mogu riješiti s CR-om, npr. broj **različitih** kupaca koji su kupili neki proizvod:



- Neki M/R frameworks zahtijevaju da sve reduce funkcije budu CR
→ ulančavanje M/R

Ulančavanje M/R

- Izlaz iz jedne M/R faze je ulaz u drugu fazu
- Izlaz iz prve faze se može snimiti (za druge potrebe) kao materijalizirani pogled
- Npr. hoćemo usporediti prodaju proizvoda **za svaki mjesec** 2013. godine s prodajom istog proizvoda **prethodne** godine:
 - Prva faza: prodaja proizvoda po svim **mjesecima** svih **godina**
 - Druga faza: uspoređuje dva ista **mjeseca** 2013. i 2012. godine



M/R primjer - JOIN

- Kako bi napravili?

```
SELECT nazProizvod, SUM(kolicina)
  FROM stavka, proizvod
 WHERE stavka.sifProizvod = proizvod.sifProizvod
 GROUP BY sifProizvod, nazProizvod
```

- Tko želi znati više:
 - www.cs.kent.edu/~jin/Cloud12Spring/DatabaseMapReduce.pptx

MapReduce/Hadoop

- Premošćuje **jaz** između specifikacije paralelnog algoritma na visokoj razini i same implementacije
- Obično se implementacija bazira na specijaliziranom datotečnom sustavu optimiranom za distribuirani pristup (GFS, HDFS)
- Najpoznatiji softver: Hadoop = HDFS + M/R
- Projekti koji nadograđuju osnovnu funkcionalnost:
 - **Pig**
[http://en.wikipedia.org/wiki/Pig_\(programming_language\)](http://en.wikipedia.org/wiki/Pig_(programming_language)): ***Pig** is a high-level platform for creating **MapReduce** programs used with **Hadoop**. The language for this platform is called **Pig Latin**.*
 - **Hive** (donosi shemu, HiveQL - upitni jezik nalik SQL-u)
http://en.wikipedia.org/wiki/Apache_Hive : ***Apache Hive** is a **data warehouse** infrastructure built on top of **Hadoop** for providing data summarization, query, and analysis.*

Primjeri



mongoDB

mongoDB

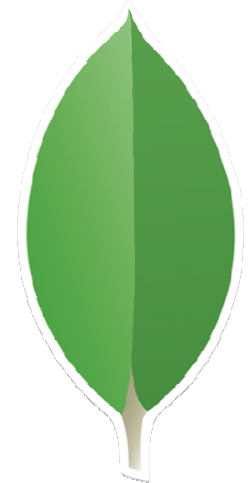
- Document baza podataka (JSON dokumenti pohranjeni kao BSON)
- Master - Slave
- Svojstva:
 - Ad-hoc upiti
 - Indeksiranje
 - MS replikacija
 - Sharding + load balancing
 - File storage
 - Aggregation
 - ServerSide Javascript
 - Language binding
 - FLOSS

<http://db-engines.com/>

The most popular database management systems

| December 2015 | Score |
|---|-------|
| 1. Oracle | 1498 |
| 2. MySQL | 1299 |
| 3. Microsoft SQL Server | 1123 |
| 4. MongoDB | 301 |
| 5. PostgreSQL | 280 |

[» more](#)



Dohvat podataka

1. Dohvat se obavlja pomoću **find funkcije** koja je oblika:

`db.collection.find(query, projection)`

- Upit se zadaje pomoću js/json objekata pri čemu su na raspolaganju razni mongo operatori (logički, usporedbe, ...), npr.

```
db.inventory.find(  
  {  
    $or: [ { qty: { $gt: 100 } }, { price: { $lt: 9.95 } } ]  
  }  
)
```

- Više u **Uputama za projekt** i na: <https://docs.mongodb.org/manual/tutorial/query-documents/>

2. **Aggregation pipeline** – agregacija pomoću koncept cjevovoda
3. **Map/Reduce** – agregacija pomoću M/R koncepta, funkcije se pišu u JS-u

Indeksi (1)

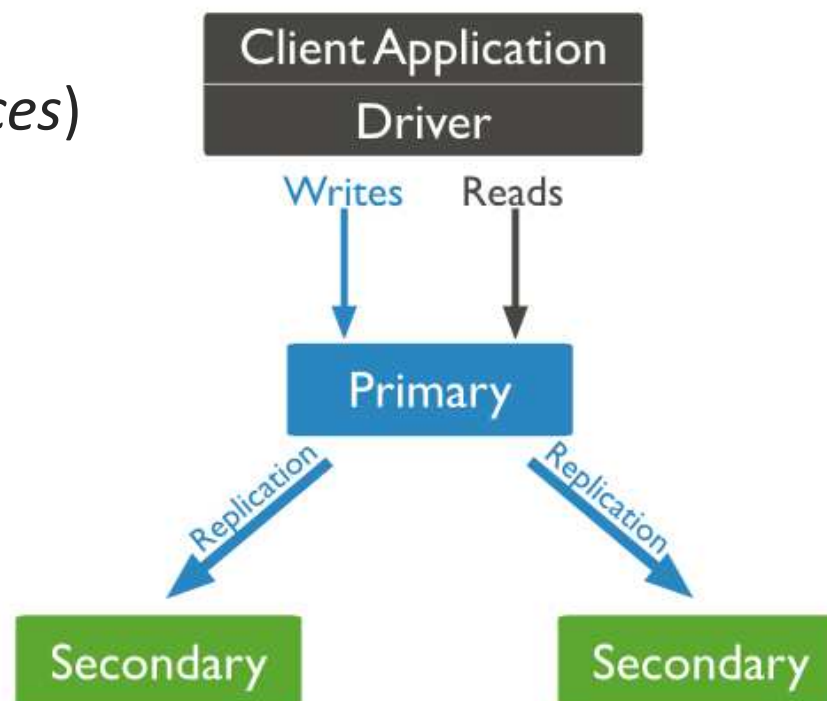
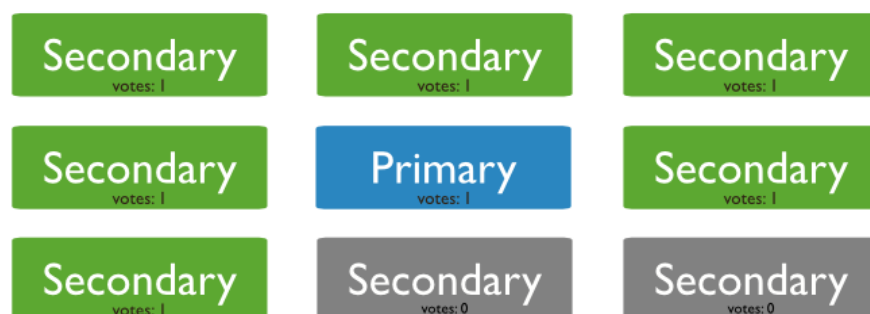
- *Index vs collection scan*
- Definiraju se na razini kolekcije, može se indeksirati bilo koji (pod)atribut dokumenta
- Vrste indeksa:
 - Default _id
 - Jednostavni (jedan atribut)
 - Kompozitni (više atributa)
 - Multikey indeks (indeksiraju se elementi polja!) – ako je atribut polje Mongo to radi automatski
 - Geoprostorni indeksi (2d indexes, 2sphere indexes)
 - Tekst indeksi (uklanjaju stop riječi, svode na korijen)
 - Hash indeksi („to support hash based sharding”) – ravnomjernija distribucija podataka

Indeksi (2)

- Svojstva:
 - Unique
 - Partial – samo dokumenti u kolekciji koji zadovoljavaju zadane uvjete
 - Sparse – samo dokumenti koji imaju atribut, može se kombinirati s unique
 - TTL indeks – mogu se koristiti za uklanjanje „starih” dokumenata iz baze

Replikacija

- Standardno (u produkciji) RS (*replica sets*) od **tri** člana. RS-ovi omogućuju redundanciju i *fault tolerance*.
- Ako dođe do kvara mastera, ostali čvorovi organiziraju izbore i izabiru novog mastera (*failover proces*)
- Do 50 članova, ali samo 7 s pravom glasa



Izbori, arbiter

- Moguće je dodati i „arbiter” čvorove koji nemaju podatke i čija jedina uloga je omogućit kvorum u RS
- Dodaju se kad imamo paran broj čvorova, kako bi dobili neparan
- Npr.

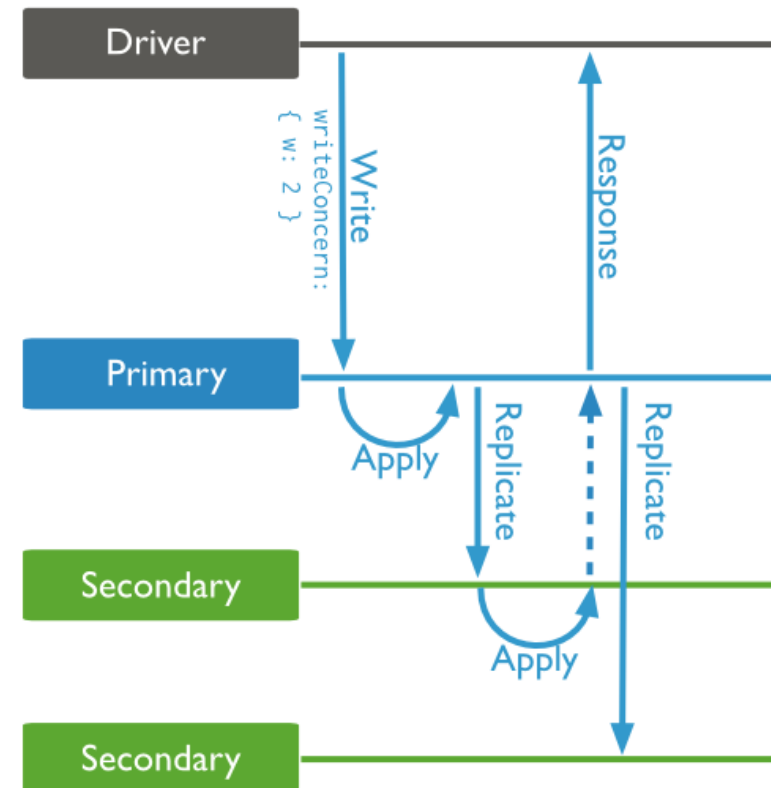


| Br. Čvorova | Potrebna većina za izbore | Fault tolerance |
|-------------|---------------------------|-----------------|
| 3 | 2 | 1 |
| 4 | 3 | 1 |
| 5 | 3 | 2 |
| 6 | 4 | 2 |

Mongo Write/Read concern

- Opisuje razinu potvrde servera kod obavljanja zadane operacije
- `{ w: <value>, j: <boolean>, wtimeout: <number> }`
 - `w` – broj instanci (0, 1, „majority”, <tag set>)
 - `j`: zapisano u dnevnik
 - `wtimeout`: vremenski limit
- Npr.:

```
db.products.insert(  
  { item: „čokolada", qty : 10 },  
  { writeConcern:  
    { w: 2, wtimeout: 5000 } }  
)
```
- `readConcern`:
`{ level: <majority|local> }`

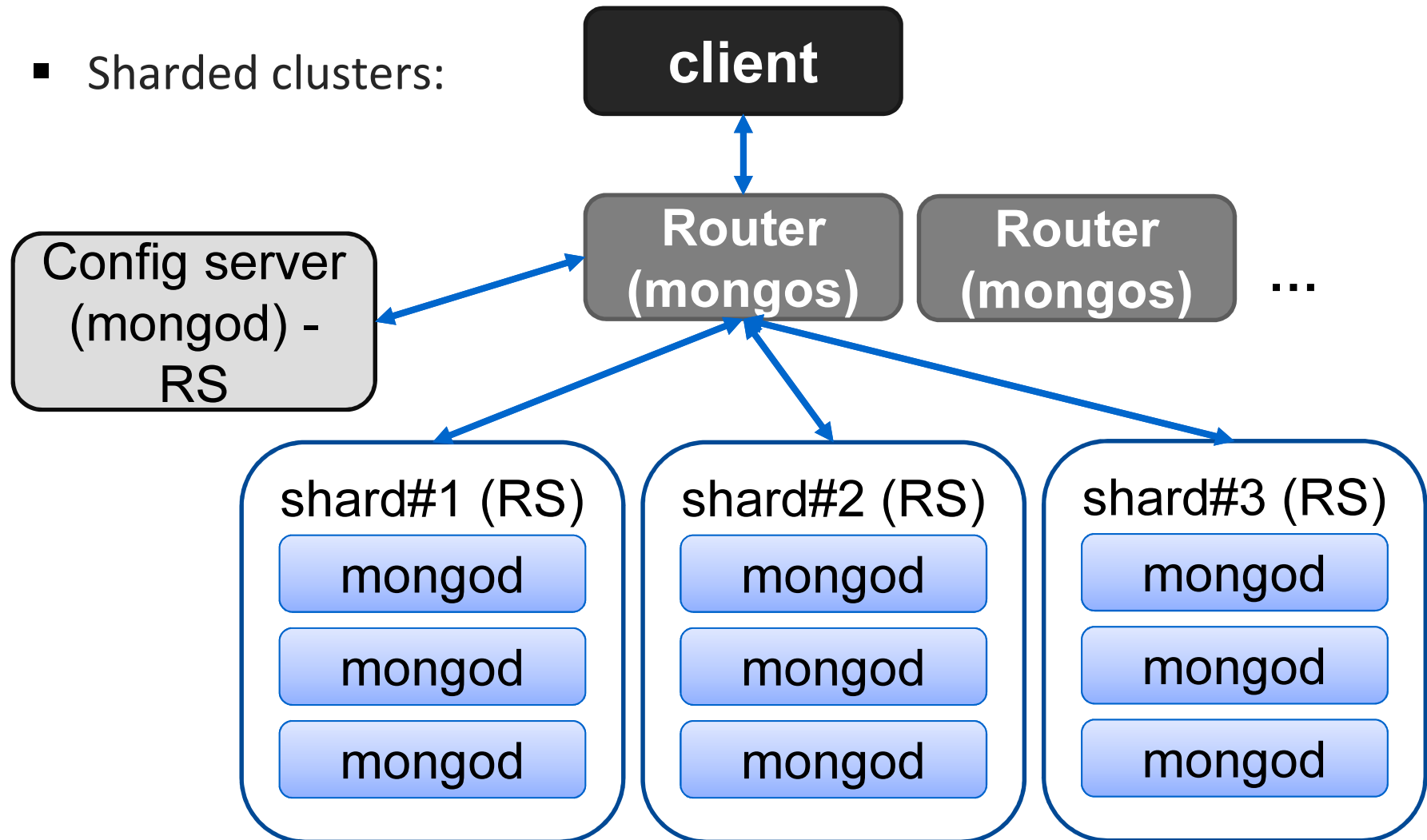


Atomarnost i „transakcije”

- Operacije su atomarne **na razini dokumenta** (može imati ugniježdene dokumente)
- Operacija koje mijenja više dokumenata nije, kao transakcija, atomarna
- `$isolated operator` – simulira transakciju tako što postavlja exclusive lock na cijelu kolekciju
 - Ne radi na sharded clusteru
 - Ne omogućuje *all-or-nothing atomicity*, npr. kod greške nema rollbacka
- Two-Phase Commits:
 - primjer na: <https://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/>
 - „The example transaction above is intentionally simple ...
... Production implementations would likely be more complex. ”

Sharding (1)

- Sharded clusters:

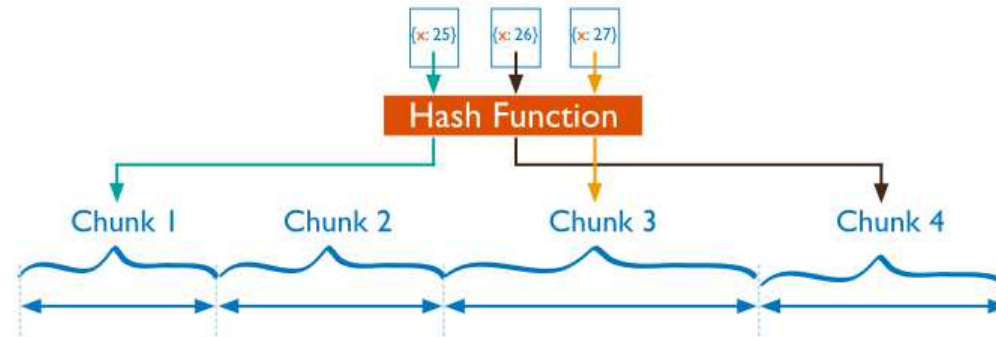
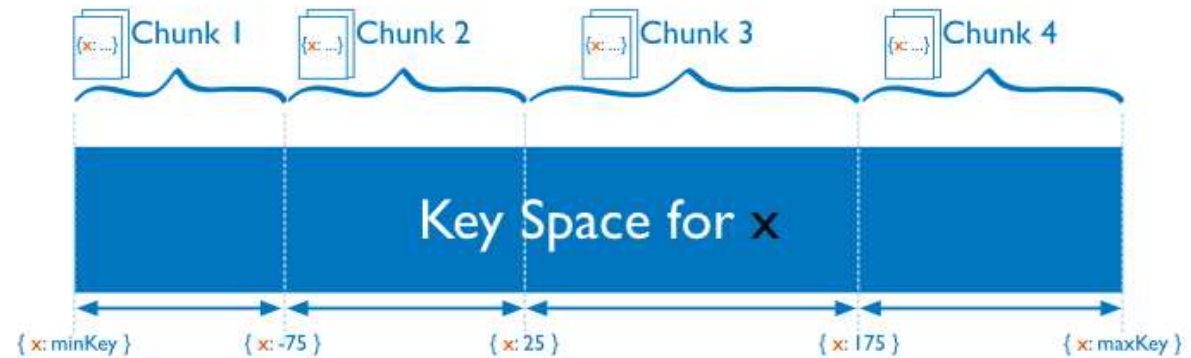


Sharding (2)

- Shard key – indeksirani atribut ili skup atributa koji postoji u svakom dokumentu
- Ključevi (shard key) se raspoređuju u grumene (chunk) koji se ravnomjerno (sljedeći slajd) raspoređuju po čvorovima

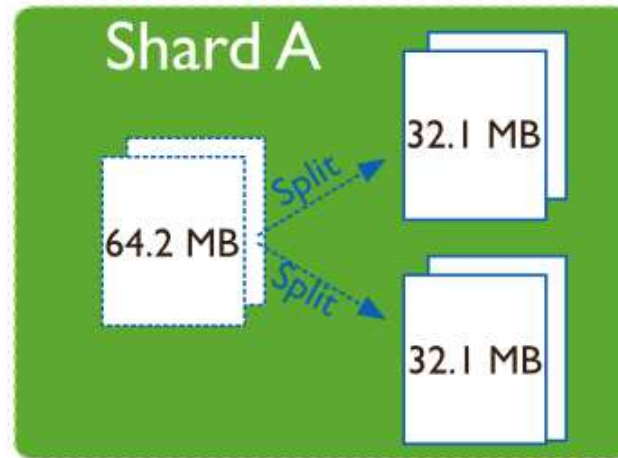
Sharding (3)

- Range based sharding
 - range based queries brži, ali mogu narušiti distribuciju čime se ugrožava skaliranje
- Hash based sharding
- Tag aware sharding

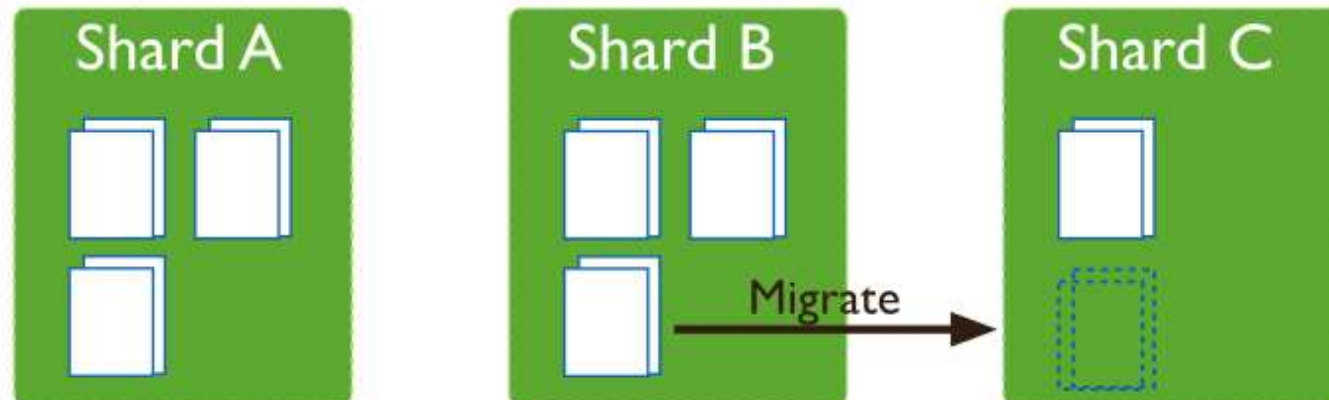


Sharding (4) – balansiranje opterećenja

- Splitting



- Balancing



Sharding (5)

- Npr. zanimljiva prezentacija:

MongoDB for Time Series Data Part 3: Sharding

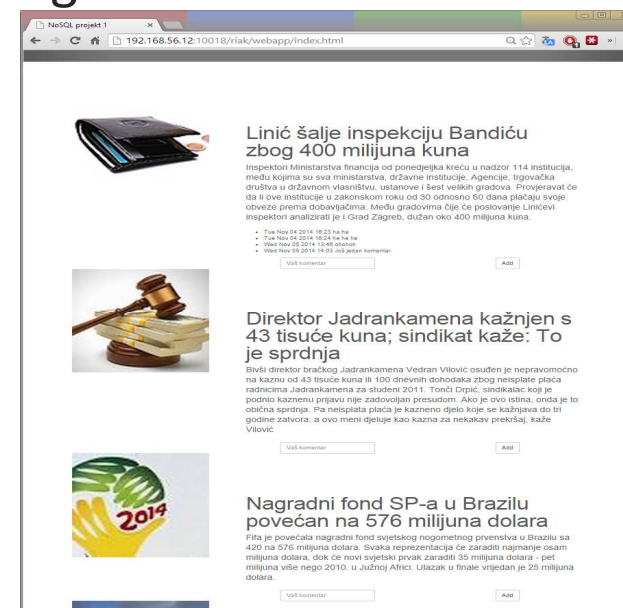
<https://www.mongodb.com/presentations/mongodb-time-series-data-part-3-sharding>

Projekt iz NoSQL-a

- **Rok za predaju: srijeda, 20.1.2016. u 9:00.**
- **Detaljno u Uputama objavljenim na stranici Materijali**
- **Minimalni portal temeljen na Mongu (10 bodova)**
 - Napraviti web portal koji ispisuje najnovijih (kako?) N (npr. N=10) vijesti. Svaka vijest treba biti (otprilike) sljedećeg oblika:

| Naslov | Slika |
|--------|-------|
| Tekst | |
| Autor | |

- Omogućiti komentiranje vijesti
- **MapReduce upiti (3+7 = 10 bodova)**
 - 3 = Upit koji vraća listu članaka poredanu silazno po broju komentara.
 - 7 = Upit koji za svakog autora vraća prvih 10 najkorištenijih riječi.



Za one koji žele znati više

- Na stranici Materijali su objavljene su i upute za rad s Riakom, pa možete pogledati

