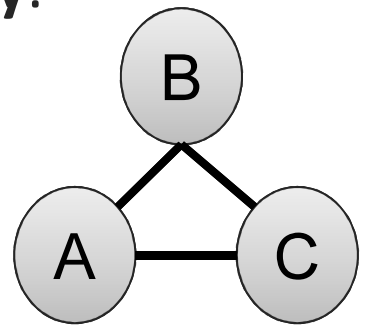
Advanced Databases

Lectures
December 2014.

NoSQL
3/3

Quorum

- N = 3 (replication factor <> node number)
- **P2P** model
- For **consistency on write** it suffice to have **majority**:



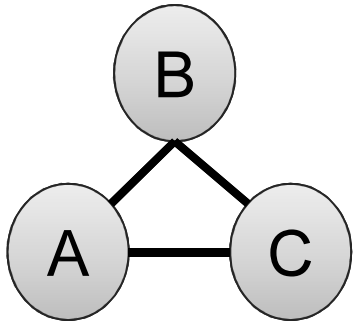
W > N/2

- Consistency on reads depends on W
- It is possible to have consistency on reads even when there is no consistency on writes –read all nodes!
  -> This does not mean that this will prevent the update conflict, but it will detect it
- For **consistency on read** it suffice to have :

R + W > N

CA compromise - summary

- The more the nodes involved in communication:
  - ✓ Better consistency
  - ✗ Higher latency
- ... and vice versa
- Not a binary decision
- Can be tuned, even on per request basis!
- Example:
  - We want fast reads
  - Sacrifice the writes:
  - **N = 3, W = 3, R = 1**



Example: Riak

- n_val – number of replicas
- r – number of replica reads to consider the read successful
- w – number of replica writes to consider the write successful
- E.g. setting bucket parameters:

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

- Can be assigned on every request!

  E.g.

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

- Or:

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

Example: Riak (2), abbreviations

- Riaku introduced abbreviations for common values:

Abbr	Definition
`One`	1
`All`	n_val
`Quorum`	n_val/2+1
`Default`	Whatever is set for the *bucket*

- Npr.

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

Relaxing durability

- Again, trade-off: *durability vs performance*
- E.g. in-memory BP
- E.g. storing user-sessions
- Also happens (un-invited) on replication (*replication durability*):
  - Master processes update
  - Master fails before propagating update
  - Slaves elect a new master node
  - What happens when master comes back online?

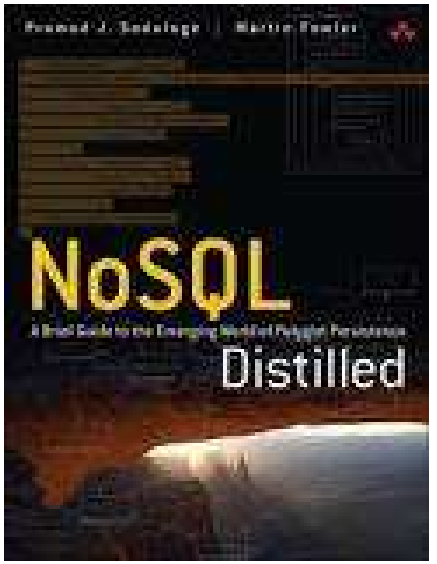
Example: Riak - durability

- On write:
  i. Object is written to memory
  ii. Write acknowledged
  iii. Object persisted to disk
- Potential data loss between (ii) and (iii).
- However, it can be explicitly assigned for (iii) to be done before (iii) on any number of nodes, e.g. (on one node):
  dw = **d**urable **w**rite

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

Additional literature

- Reminder: the lectures are largely based upon:
  - NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence
  - Sadalage, Fowler



- Especially for those who missed the class, we advise to watch the presentation by Martin Fowler„Introduction to NoSQL”:
  - http://www.youtube.com/watch?v=qI_g07C_Q5I

Conflict detection

Version stamps

- Version stamp: field changing on every data update, e.g. HTTP ETag
- Options for generating version stamps:
  1. **Counter**
     - ✓ Can tell the newer version
     - × Single master needed to generate
  2. **GUID**
     - ✓ Anyone can generate, no duplicates
     - × Cannot tell which one is newer
  3. **Hash**
     - ✓ Anyone can generate; if large enough ~ GUID
     - ✓ Deterministic - the same hash for the same content (in different nodes)
     - × Cannot tell which one is newer
  4. **Timestamp**
     - ✓ Small, can tell the newer version
     - × Clock synchronisation, granularity?
  5. **Some combination of the above** (e.g. counter+hash)

Order and causal dependeny of events

- In a distributed system, we want to know:
  - Order of events
  - Potential conflicts
- Mark all events with time stamp (physical clock)?
  - The problem of synchronizing clocks (eg quartz watch err: 1s/11.6 days)
  - We can not determine the causal dependence (causality)
- Ordering (*happened-before*, *Lamport 1978.*), we know only on the basis of:

  (a) Order of **internal** events of the same process

  (b) Order of same message **send** and **receive**
- Therefore, e ε {internal, send, receive}
- **Definition**:

  Causal relation *happened-before* (denoted with→) is defined as:

  i. If there exists a process p_i : e $\rightarrow_i$ e' , then e → e'

  ii. For each message m, *send(m)* → *receive(m)*

  iii. (transitivity) If e, e' i e'' are such events that e→e' and e'→e'' then also the following is true: e→e''.

Logical clock

- Logical clocks:
  - Used to determine the order of events
  - To the event e the C(e) is assigned (~ physical time)
- Clock consistency:
  - Weak (chronology): $e_1 \rightarrow e_2 \Rightarrow C(e_1) < C(e_2)$
  - Strong (causal dependency): $C(e_1) < C(e_2) \Rightarrow e_1 \rightarrow e_2$

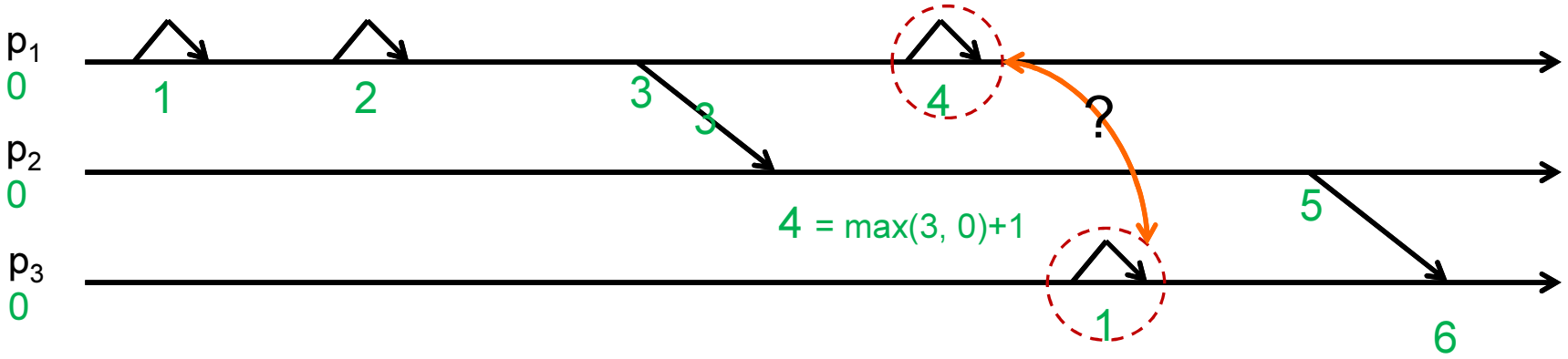
Lamport's clock

- Each process has its own internal logical clock c
- Algorithm:
  - On internal event: ++c
  - On msg send ++c and include c in the msg
  - On msg recieve, c' being the current own clock value :

    $$c = 1 + \max(c, c')$$

```
public class LamportClock {
   int c;
   public LamportClock() {
      c = 1;
   }
   public int getValue() {
      return c;
   }
   public int tick() { //on internal events
      return ++c;
   }
   public void sendAction() {
      c++;
      //include c in message
   }
   public void receiveAction(int sentValue){
      c = Util.max(c, sentValue) + 1;
   }
}
```

Lamport's clock - example



- Disadvantage:
  - Weak: $e_1 \rightarrow e_2 \Rightarrow C(e_1) < C(e_2)$
  - ~~Strong: $C(e_1) < C(e_2) \Rightarrow e_1 \rightarrow e_2$~~
  - The reason that the reverse does not hold is that logical clock assigns numbers to states and numbers are always comparable. That way, two incomparable states can seem comparable.
- If we negate: $C(e_1) >= C(e_2) \Rightarrow e_1 \not\rightarrow e_2$
  or, in other words:
  $C(e_1) >= C(e_2) \Rightarrow$ e_1 might have happened before e_2,
  or they are not comparable according to →,
  but surely, we can't say that e_1 happened before e_2

For the next lecture

- Read:
  - Wikipedia:
    http://en.wikipedia.org/wiki/Lamport_timestamps
  - Lecture 2. Unit 4. Logical clocks and vector clocks, ID2203:
    http://www.youtube.com/watch?v=TH1dA7dPB2c

- Might want to read:
  - I. Podnar Žarko, K. Pripužić, I. Lovrek, M. Kušek, Raspodijeljeni sustavi, radna inačica udžbenika v.1.0, 2012:
    http://www.fer.unizg.hr/_download/repository/Rassus-2013_knjiga_v_1_0.pdf
  - Primjena Lamportovog sata:
    http://www.mcs.csueastbay.edu/~cclee/cs6580/logicalclockspart1.ppt

Vektor clock (1)

- Lamport's clock sat -> not good enough for distributed data/processes
- Mattern [1989] and Fidge [1991], the intent is to achieve:
  - Weak: $e_1 \rightarrow e_2 \Rightarrow V(e_1) < V(e_2)$
  - Strong: $V(e_1) < V(e_2) \Rightarrow e_1 \rightarrow e_2$
- Describes **causal** relations
- Idea: each process has its own counter (clock):
  - $V_p[p]$ number of events processed by process **p**
  - $V_p[m]$ number of events process **p** thinks process **m** processed

Example

- P_1 processed 7 events, P_2 thinks it processed 5, P_3 thinks it processed 6.
- P_2 processed 8 events, P_1 thinks it processed 8, P_3 thinks it processed 4
- P_3 processed 12 events, P_1 and P_2 think it processed 9

	1	2	3
P_1	7	8	9
P_2	5	8	9
P_3	6	4	12

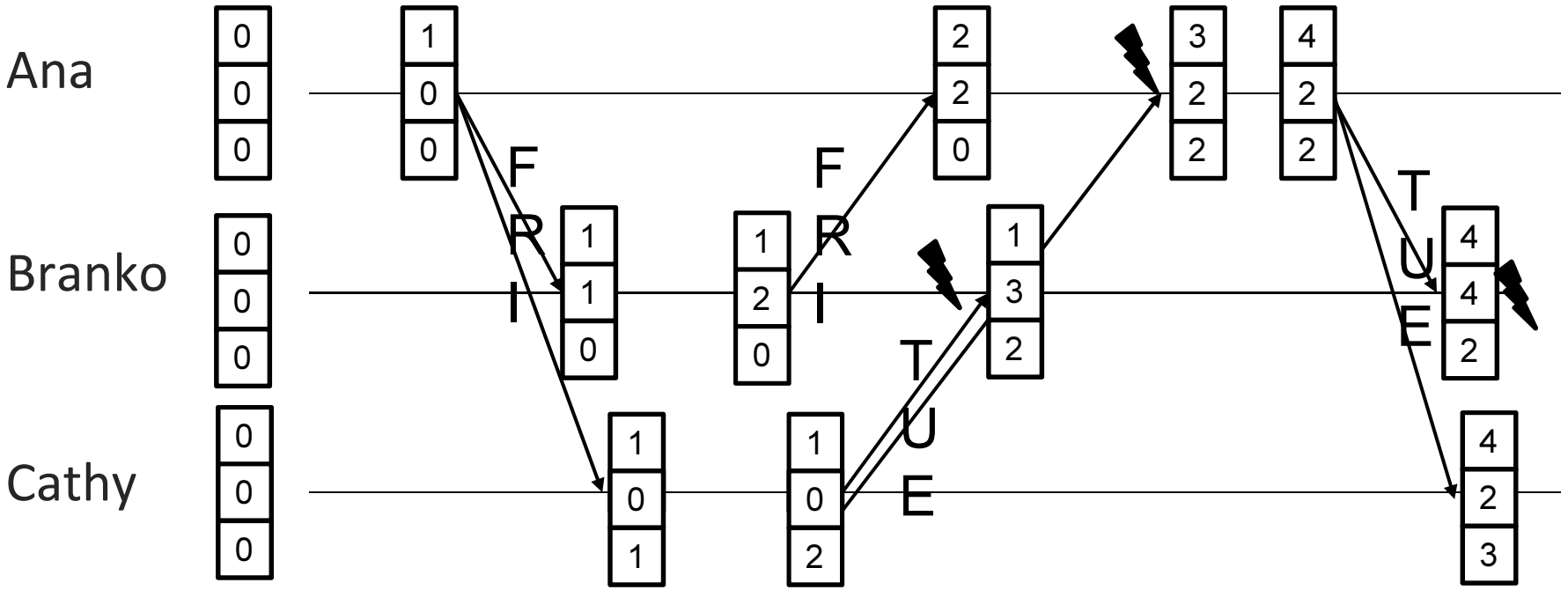
Vektor clock(2)

- Partial ordering:
  - V=V' ⇔ V[i] = V'[i], i=1,…,N
  - V≤V' ⇔ V[i] ≤ V'[i], i=1,…,N
  - V<V' ⇔ V≤V' i V≠V'
- $e_1 || e_2$ iff they are not comparable, that is: neither $e_1 \leq e_2$ nor $e_2 \leq e_1$ is true (partial ordering)
- Rules:
  - VC1: Initially $V_i[j] = 0$, for i,j = 1,2,..N
  - VC2: In process p_i, just before (any) events:
    $V_i[i] = V_i[i] + 1$
  - VC3: Process p_i attaches his V_i to every message it sends
  - VC4: When p_i receives the message with vector W, it sets:
    $V_i[j] = \max(V_i[j], W[j])$ - *note that VC2 is also performed*

Compare vectors:

V_1	7	7	7
V_2	6	7	6
V_3	6	7	8

Example – dinner date



Reminder:

VC1: Initially Vi[j] = 0, for i,j = 1,2,..N
VC2: In process pi, just before (any) events:
Vi[i] = Vi[i] + 1
VC3: Process pi attaches his Vi to every message it sends
VC4: When pi receives the message with vector W, it sets:
Vi[j] = max(Vi[j], W[j]) - note that VC2 is also performed

Vector clocks **are not meant to** resolve conflicts!

Example: Riak vector clocks (1)

- Riak uses vector clocks (http://basho.com/why-vector-clocks-are-easy/)
- Include in every request:
  - `X-Riak-ClientId` to identify " the actor" i.e. client
  - `X-Riak-Vclock`
- Should the conflict occur, Riak bases the decision on `allow_mult` property:
  - `True`: keeps (and returns) both values (`"siblings"`), each value with its own `"vtag"`
  - `False`: „last wins”

Example: Riak vector clocks (2)

- Create a new bucket „dogovor” with the setting allow_mult = true (default is false) , which allows for the creation of a sibling

```
$ curl -v -X PUT
http://192.168.56.12:10018/buckets/dogovor/props
-H "Content-Type: application/json„
 -d '{"props":{"allow_mult":true}}'
```

- Suppose that Ljilja , Krešo and Jasna are assessing student Igor’s homework .
- Krešo inserts grade one:

```
$ curl -v -X PUT
http://192.168.56.12:10018/buckets/dogovor/keys/igor -H
"Content-Type: application/json" -H "X-Riak-ClientId:
kreso" -d '{"grade":"1"}'
```

Example: Riak vector clocks (3)

- Let us retrieve the vector clock:

```
$ curl -v
http://192.168.56.12:10018/buckets/dogovor/keys/igor

npr. X-Riak-Vclock:
a85hYGBgzGDKBVIcKlYHQoIWzJiQwZTImMfKsCZA6gxfFgA=
```

- Ljilja and Jasna read, and Ljilja inserts grade **two** while including the Vclock in the PUT request:

```
$ curl -i -X PUT
http://192.168.56.12:10018/buckets/dogovor/keys/igor \
        -H "X-Riak-ClientId: ljilja" \
        -H "X-Riak-Vclock:
a85hYGBgzGDKBVIcKlYHQoIWzJiQwZTImMfKsCZA6gxfFgA=" \
        -H "Content-Type: application/json" \
        -d '{"grade" : 2}'
```

Example: Riak vector clocks (4)

- Jasna agrees with Krešo, inserts one, while including (now obsolete) Vclock in the request:

```
$ curl -i -X PUT
http://192.168.56.12:10018/buckets/dogovor/keys/igor \
        -H "X-Riak-ClientId: jasna" \
        -H "X-Riak-Vclock:
a85hYGBgzGDKBVIcKlYHQoIWzJiQwZTImMfKsCZA6gxfFgA=" \
        -H "Content-Type: application/json" \
        -d '{"grade" : 1}'
```

- What should Riak do now*?
- Both versions (siblings) can be retrieved it we add:
  -H "Accept: multipart/mixed" to the request:

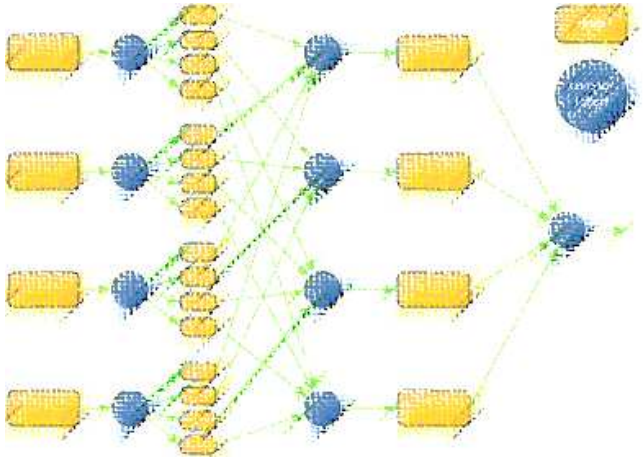
```
$ curl -v
http://192.168.56.12:10018/buckets/dogovor/keys/igor -H
"Accept: multipart/mixed"
```

**This is a shortened version of the example from the project instructions*

Vektor clock (4)

- Detecting, not resolving conflicts
- Additional literature:
  - Wikipedia:
    - http://en.wikipedia.org/wiki/Lamport_timestamps
    - http://en.wikipedia.org/wiki/Vector_clocks
  - Lecture 2. Unit 4. Logical clocks and vector clocks, ID2203: http://www.youtube.com/watch?v=TH1dA7dPB2c
  - I. Podnar Žarko, K. Pripužić, I. Lovrek, M. Kušek, Raspodijeljeni sustavi, radna inačica udžbenika v.1.0, 2012: http://www.fer.unizg.hr/_download/repository/Rassus-2013_knjiga_v_1_0.pdf

- For those „who want to know more” (that is, not obligatory):
  - Lamport clock application: http://www.mcs.csueastbay.edu/~cclee/cs6580/logicalclockspart1.ppt
  - Additional memory consumption, that is, message size growth:
    - An efficient implementation of vector clocks: http://www.sciencedirect.com/science/article/pii/002001909290028T
    - Riak vector clock prunning: http://docs.basho.com/riak/latest/references/appendices/concepts/Vector-Clocks/#Vector-Clock-Pruning
  - Interval Tree Clocks: generalization for unknown number of processes:: https://github.com/sinabz/itc4j



MapReduce

MapReduce

- Many NoSQL databases, regardless of type, implement MapReduce algorithm
- Developed and patented by Google in 2004., e.g. *Google File System*
-  – (among other things) MapReduce implementation
- Suitable for a particular type of (parallel) problems
  - E.g. search large amounts of text, the construction of the word index, counting access to web pages, etc.

MapReduce

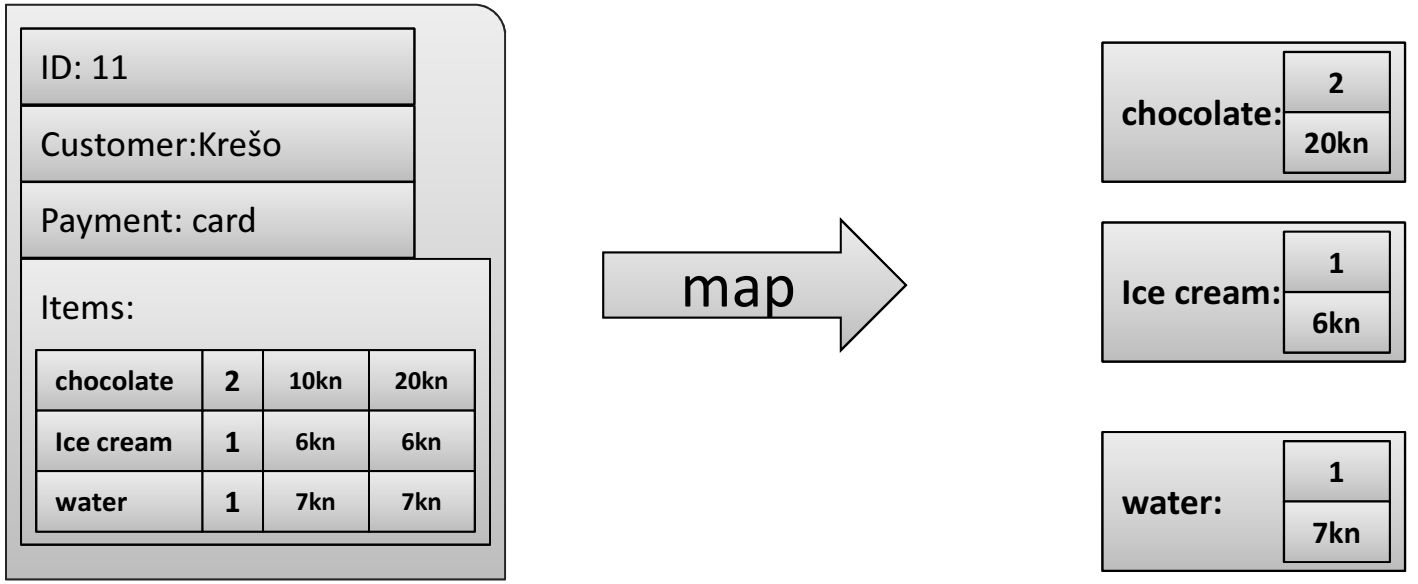
- Clusters alter the **computing** paradigm (not just storage)
- Traditionally, on a centralized DB:

(a) On client (b) On server

- M/R – different paradigm – computation at N servers
- Minimize the network traffic, compute on own data
- M/R is a *pattern*, implementations differ

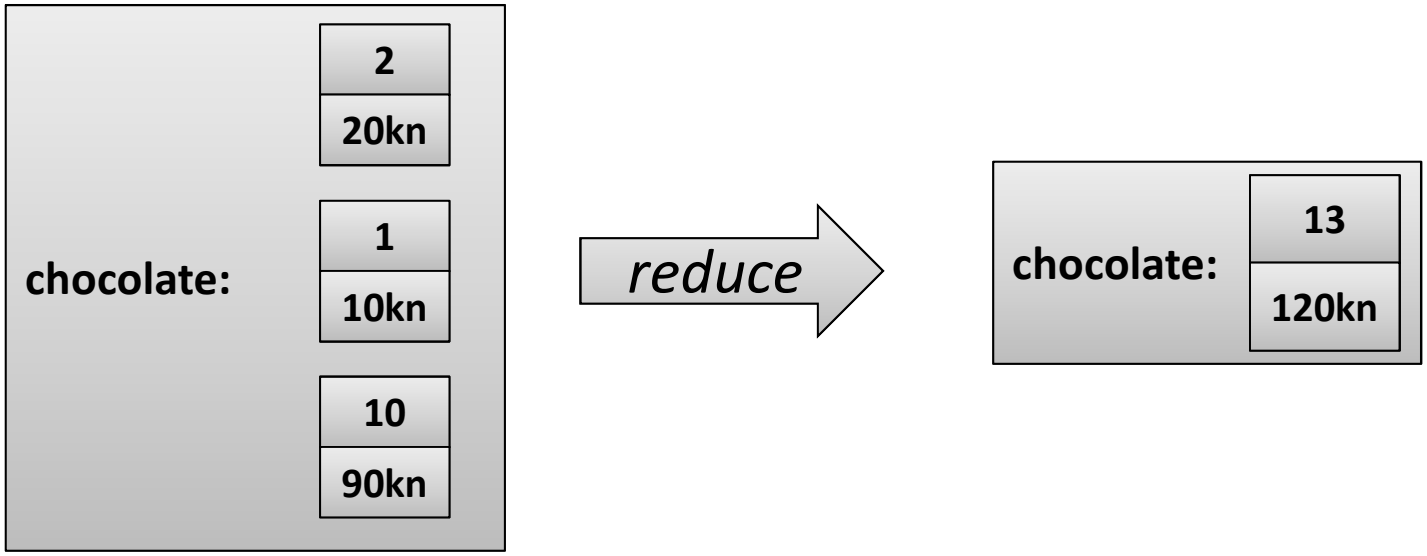
M/R – basic idea: *map*

- E.g. Sales by products in the previous month
- Map is a function: **map(k1, v1) → list (k2, v2)**
  - argument: **one** key-value (aggregate)
  - result: list, i.e. 0 or more (k2,v2) pairs
- Map functions are executed independently - **parallelism**



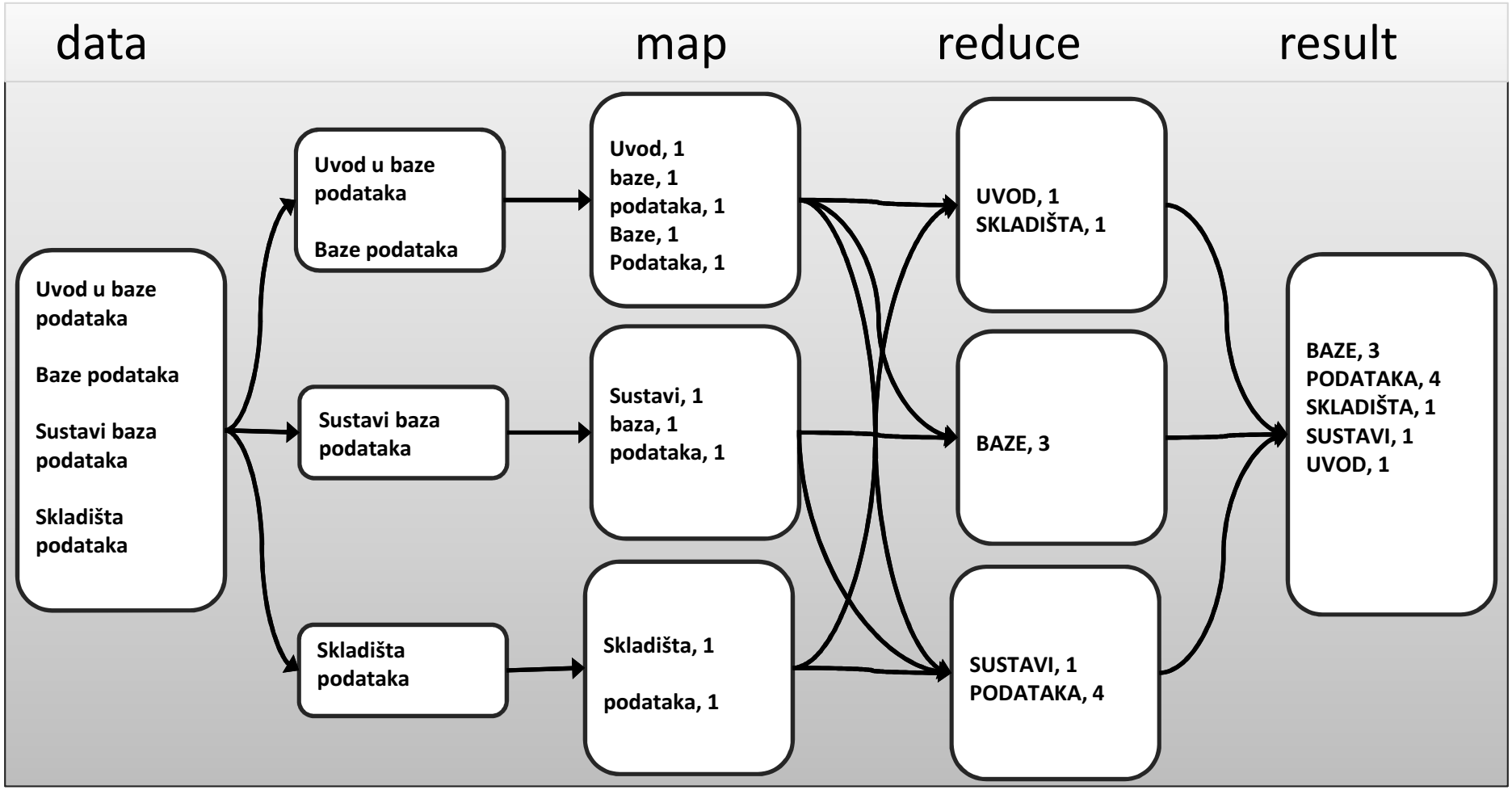
M/R - osnovna ideja: *reduce*

- Reduce is a function: **reduce(k2, list(v2)) → list(v3)**
  - argument: **list of values** for a key
  - result: 0 or more values, typically 0 or 1 value



M/R example

- Counting words



M/R example: 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. Run on server:

```
python example.py
```

2. Run on client(s):

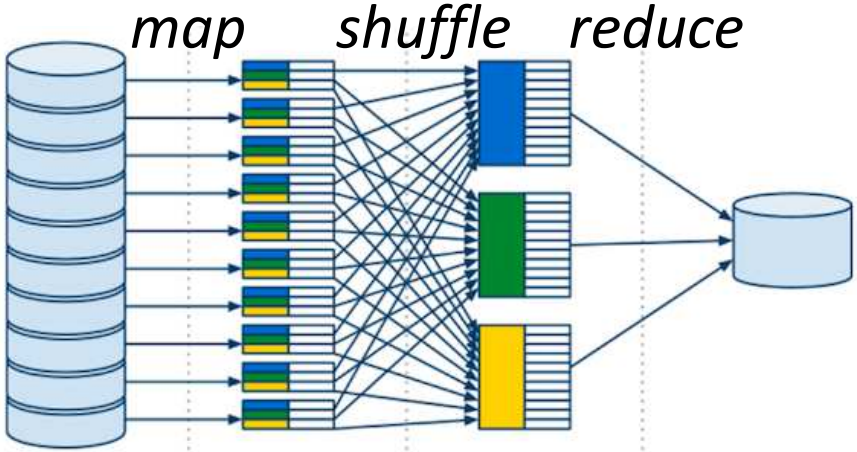
```
python mincemeat.py -p
changeme [server address]
```

3. Server prints:

```
{'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 framework

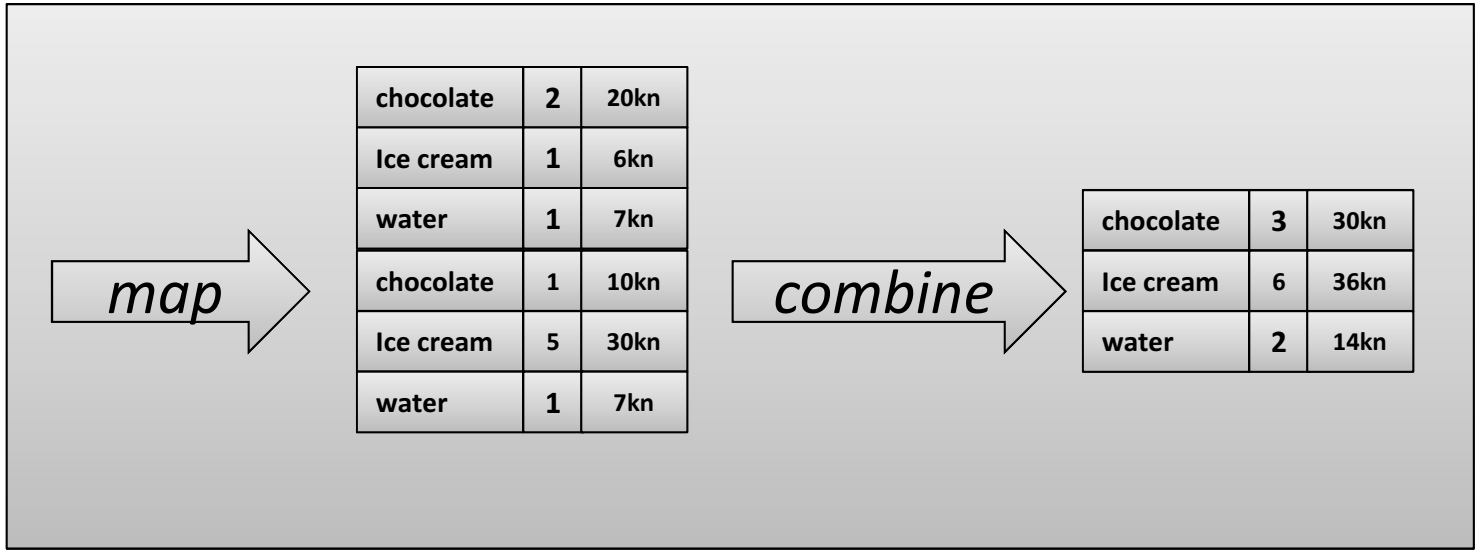
- Orchestrates the execution:
  - Manages nodes (same nodes can be used for M and R phase)
  - Executes (in parallel) tasks
  - Controls the data exchange
  - Error recovery (what happens when a node fails?)



- Or, in other words:
  1. Prepares input data for the *map* function, assigns and delivers to *map* nodes
  2. Runs *map* function on nodes
  3. Groups, (partitions) i disseminates (*shuffle*) map function output and delivers to *reduce* nodes
  4. Runs *reduce* function on nodes
  5. Brings together all the results from reduce function from all nodes

M/R - *combiner*

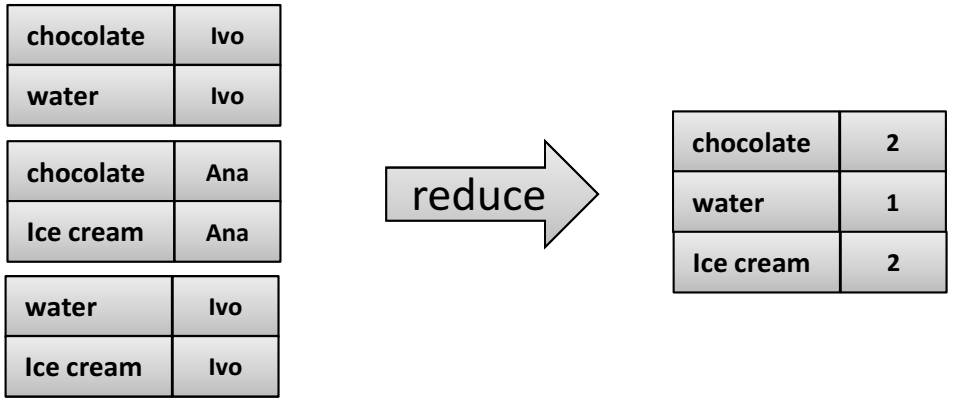
- *combiner* function combines (aggregates, reduces) all data with the same key into one record



- *combiner ~ reducer*

CR - *Combinable Reducer*

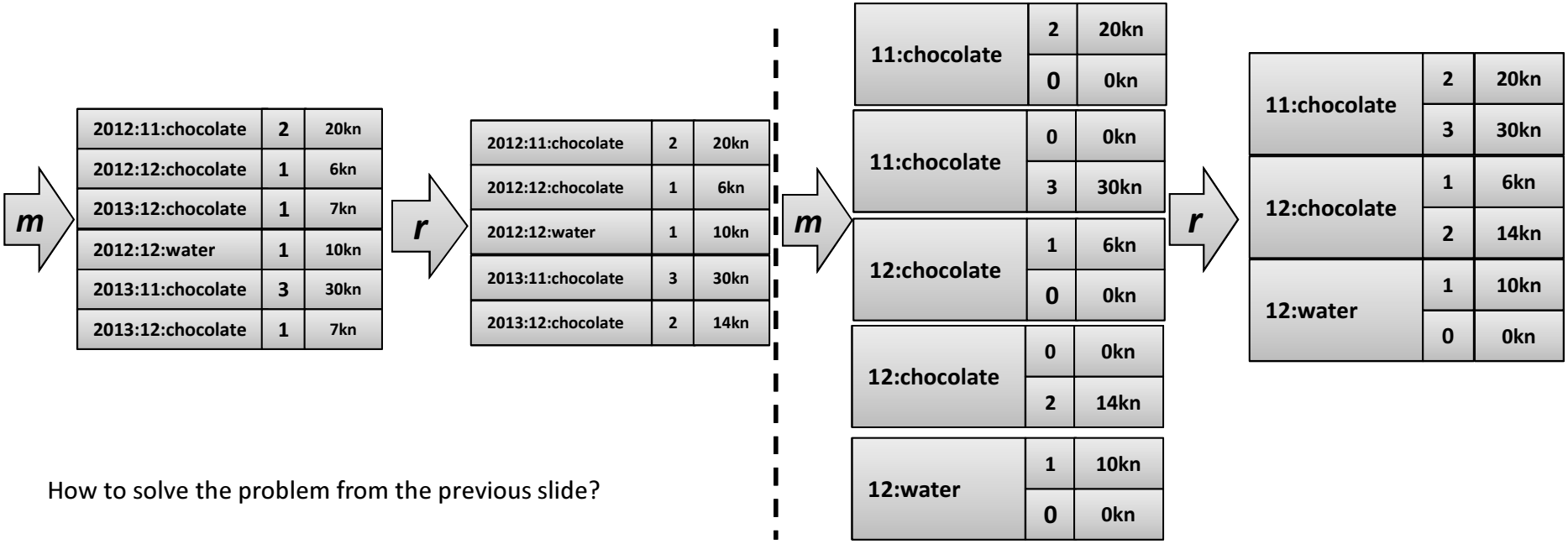
- CR is *reduce* function whose **outputs match its inputs.**
- CR:
  - ✓ Replaces *combiner*, executed on the same map node
  - ✓ Can be called before *map* phase is complete (on all nodes)
- Certain problems are not suitable for CR, e.g. The number of **distinct** customers who bought certain product:



  - Certain M/R frameworks demand all reduce functions to be CRs → M/R chaining

M/R chaining

- One M/R phase output is another's input
- First phase output can be saved (to be used in other queries) as materialized view (→ incremental M/R?)
- E.g. We want to compare the sales of products **for each month of the year** 2013. with the sales of the same product **previous** year:
  - Phase one: product sales by months and years
  - Phase two: compare the same months of years 2013. and 2012.



How to solve the problem from the previous slide?

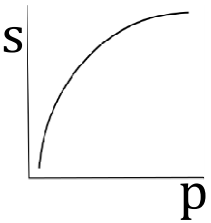
M/R example - JOIN

- How to do this in M/R?

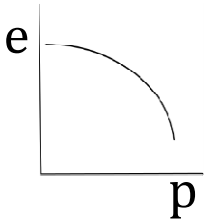
```
SELECT prodName, SUM(quantity)
  FROM orderItems, product
 WHERE orderItems.idProduct = product.idProduct
 GROUP BY idProduct, prodName
```

M/R – review (1)

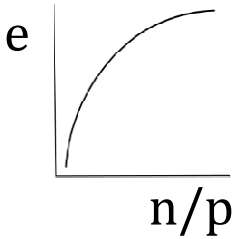
- $s = \frac{T_1}{T_P}$ s = *speedup*, P = number of proces(or)s



- $e = \frac{T_1}{PT_P}$ e = *efficiency*



- Scalable algorithm:
  n = amount of work



M/R – review (2)

P - number of processes,
D - data,
wD – useful work,
σ date reduction factor

- $s = \frac{T_1}{T_p} \quad e = \frac{T_1}{PT_p}$
- *map* nodes write to disk: $\sigma \frac{D}{P}$
- *reduce* nodes read: $\sigma \frac{D}{P \cdot P} P = \sigma \frac{D}{P}$
- therefore, additionally: $2\sigma \frac{D}{P}$
- $$e_{M/R} = \frac{wD}{P\left(\frac{wD}{P} + 2c\frac{\sigma D}{P}\right)} = \frac{1}{1 + 2c\frac{\sigma}{w}}$$
- Not dependant on P?
- *combiner*

MapReduce/Hadoop

- Bridges the gap between the specification of parallel algorithms at a high level and the implementation
- Usually, the implementation is based on a specialized file system optimized for distributed access (GFS, HDFS)
- The most famous software: Hadoop HDFS + = M / R
- Projects that build on the basic functionality:
  - **Pig**

    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**

    *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.*

Bloom filter

Bloom filter (1)

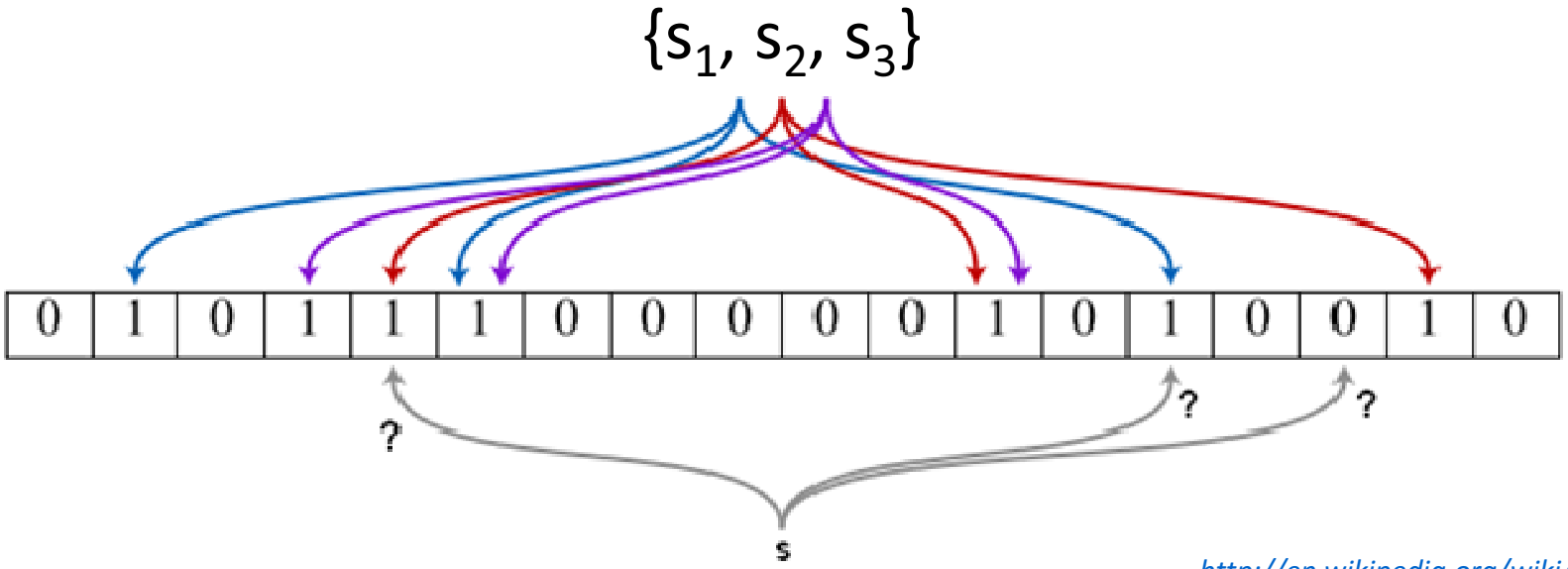
- Spatially efficient **probabilistic data structure** that is used for testing the membership of elements in a set
- Developed by B. H. Bloom 1970.* for the spell checking program
- The efficiency is achieved at the expense of the low probability of wrong answer, namely:
- Bloom filter can give **a false positive response**:
  - It can report an element to **be** a member of the set **although it is not**
  - But not vice versa: it will never report an element to be a member of the set if it is not a member

**Bloom, B. Space/time tradeoffs in hash coding with allowable errors, Communications of the ACM, 13(7):422-426, Srpanj 1970.*

Bloom filter (2)

- $S = \{s_1, s_2, \ldots, s_n\}$
- S is represented with an array of **m** bits (initially 0)
- **k** independent hash (uniform) functions:

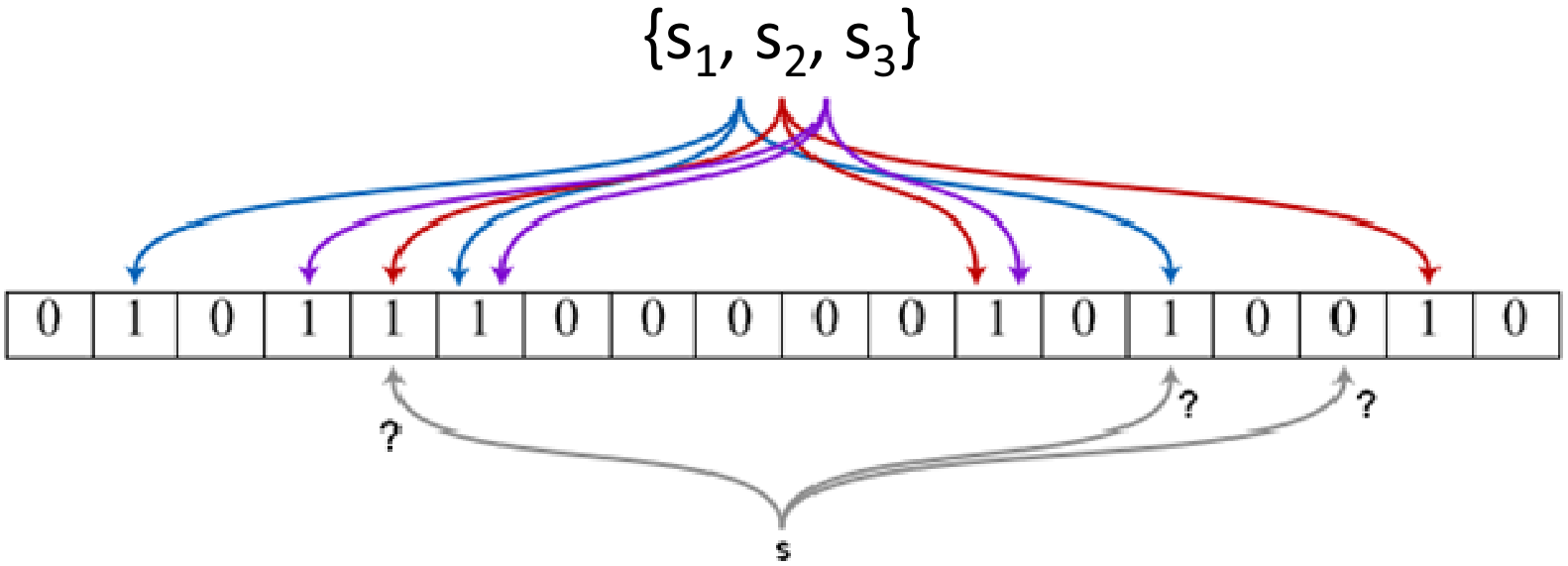
  $h_1, h_2, \ldots h_k$ with the range of {0, m-1}
- Elements can be added to the set (but not removed)



http://en.wikipedia.org/wiki/Bloom_filter

Bloom filter (3)

- Adding to the set:
  - set $h_1(s)$, $h_2(s)$, ... $h_k(s)$ bits to 1
- Membership checking:
  - (probably) **is a member** iff ∀ $h_i(s)$==1, i=1..k
  - otherwise: (100%) **not a member**



Bloom filter (4) – false positive probability

- One element, probability that the bit **is not** set to 1 (for one hash function):

$$1 - \frac{1}{m}$$

- …that is, for k hash functions:

$$\left(1 - \frac{1}{m}\right)^k$$

- For n elements:

$$\left(1 - \frac{1}{m}\right)^{nk}$$

- On contrary, possibility of bit being **set** to 1:

$$1 - \left(1 - \frac{1}{m}\right)^{nk}$$

- Finally, on testing, we check k bits:

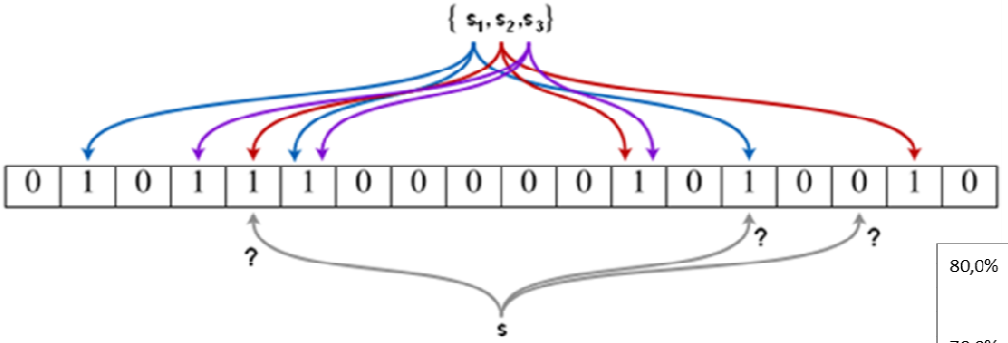
$$\left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k \approx \left(1 - e^{-nk/m}\right)^k$$

- For given n and m, k_{min} is:

$$k_{min} = \frac{m}{n} \ln 2 \approx 0.7 \frac{m}{n}$$

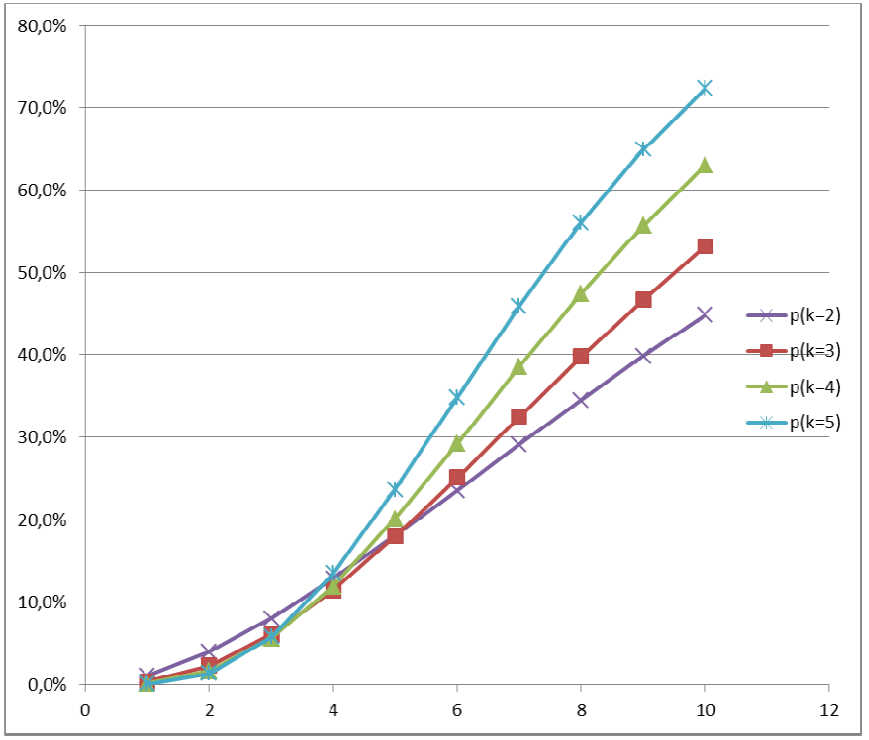
Bloom filter (5) – false positive probability

- In our example (m=18, n=3, k=3) we check for s:



$$k_{min} = \frac{m}{n} \ln 2 = \frac{18}{3} \ln 2 \approx 4{,}16$$

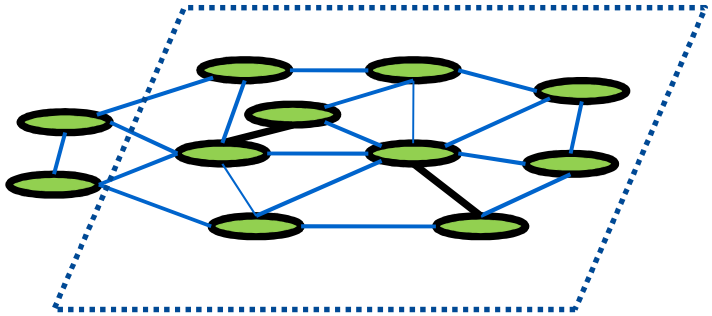
n	p(k=2)	p(k=3)	p(k=4)	p(k=5)
1	1,1%	0,4%	0,2%	0,1%
2	3,9%	2,3%	1,6%	1,4%
3	**8,0%**	**6,0%**	**5,6%**	**5,7%**
4	12,8%	11,4%	11,9%	13,5%
5	18,1%	18,0%	20,1%	23,7%
6	23,6%	25,1%	29,2%	34,9%
7	29,1%	32,5%	38,5%	46,0%
8	34,5%	39,8%	47,5%	56,1%
9	39,8%	46,7%	55,7%	64,9%
10	44,8%	53,2%	63,0%	72,3%



DHTs - Distributed Hash Tables

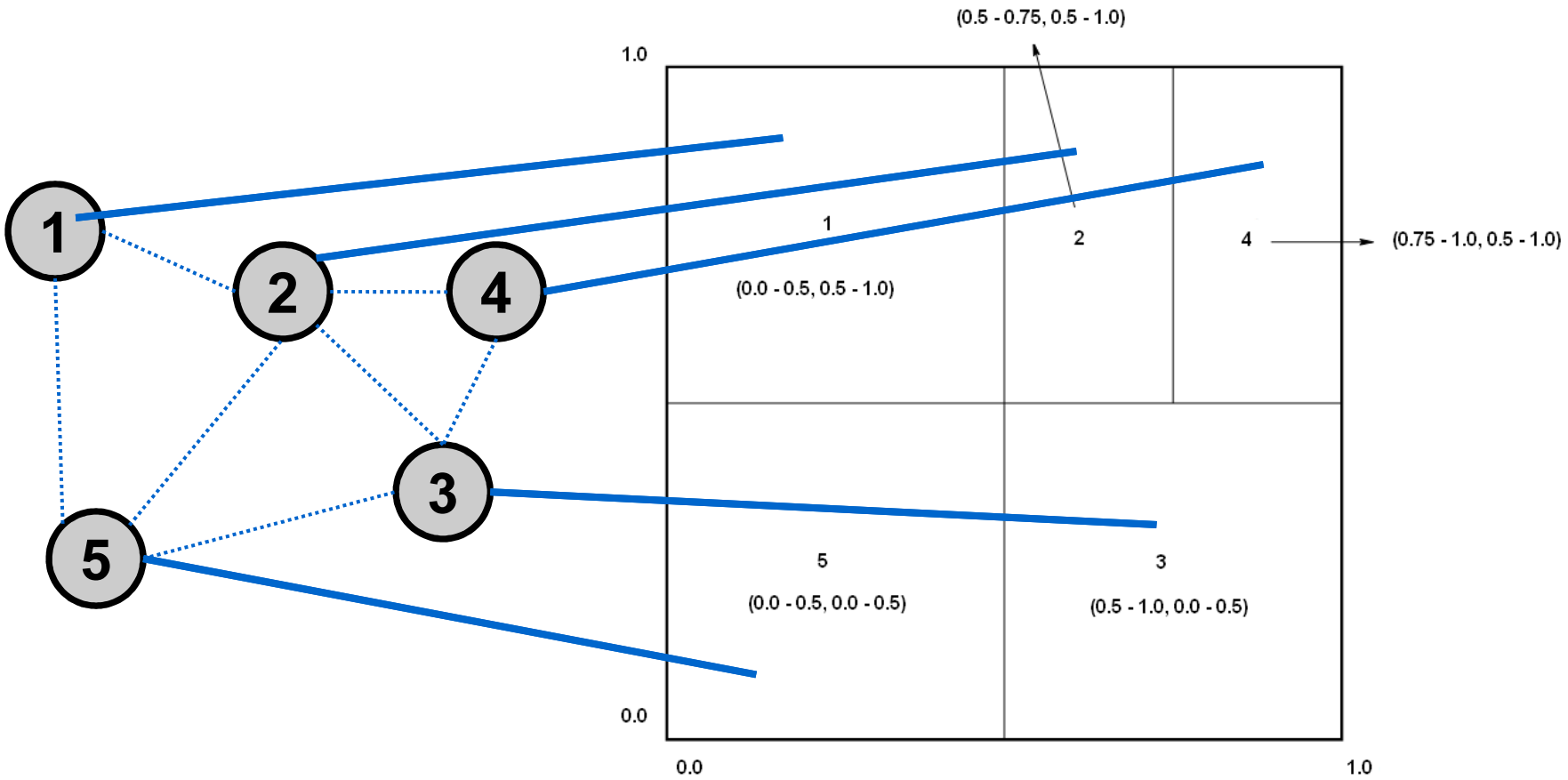
Distributed hash tables

- Distributed hash tables (DHTs) are **decentralized distributed systems** that provide hash table functionality, that is the two following functions:
  - `put(key, values)`
  - `get(key)`



- Each node that participates in the system can call these two functions, and is at the same time responsible for implementing these functions.
- Responsibility for mapping key to value is distributed among the participants in the system in such a way that changes in the composition of the participants do not cause major changes in the system.
- Provide the infrastructure for the development of many complex applications such as distributed file systems, P2P systems for the exchange of data, systems for data distribution, messaging systems, etc.

Distributed hash tables



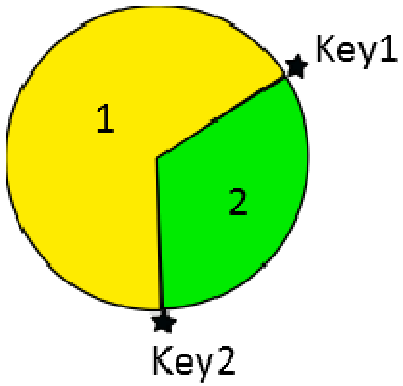
Distributed Hash Tables

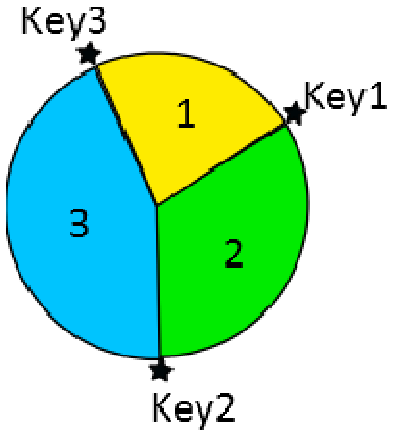
- P2P: Gnutella, Napster, … BitTorrent
- Properties:
  - **decentralization** – no central authority
  - **scalability** – system works efficiently with very large number of participants
  - **Resilience to errors** – system is stable and reliable enough despite the continuous arrival, departure and failure of participants
- To be able to achieve these good properties, each participant maintains contact with only a small number of participants in the system, typically with O (log (n)). In this way, the work that has to be done due to changes in the membership is limited.
- Each participant in the system becomes responsible for a specific part of the key space.
- Most distributed hash tables, when mapping keys to nodes, uses some sort of **consistent hashing techniques**.

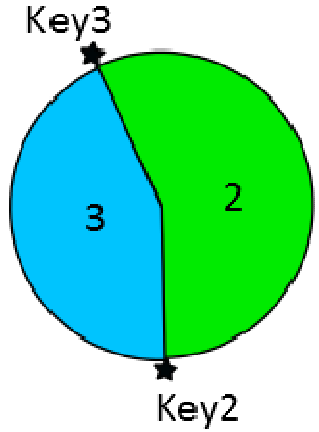
Consistent hashing

- *David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy.* ***Consistent Hashing and Random Trees: Tools for Relieving Hot Spots on the World Wide Web****. STOC 1997*
- Idea: hash objects to n slots (nodes, computers)
- Common approach: **`hash(o) % n`**
- What if **n** changes?
- The idea of consistent hashing is to:
  - If a **new node is added** it takes over only a small („just”) part of the overall load
  - If the node **departs** from the system, its load is distributed to others

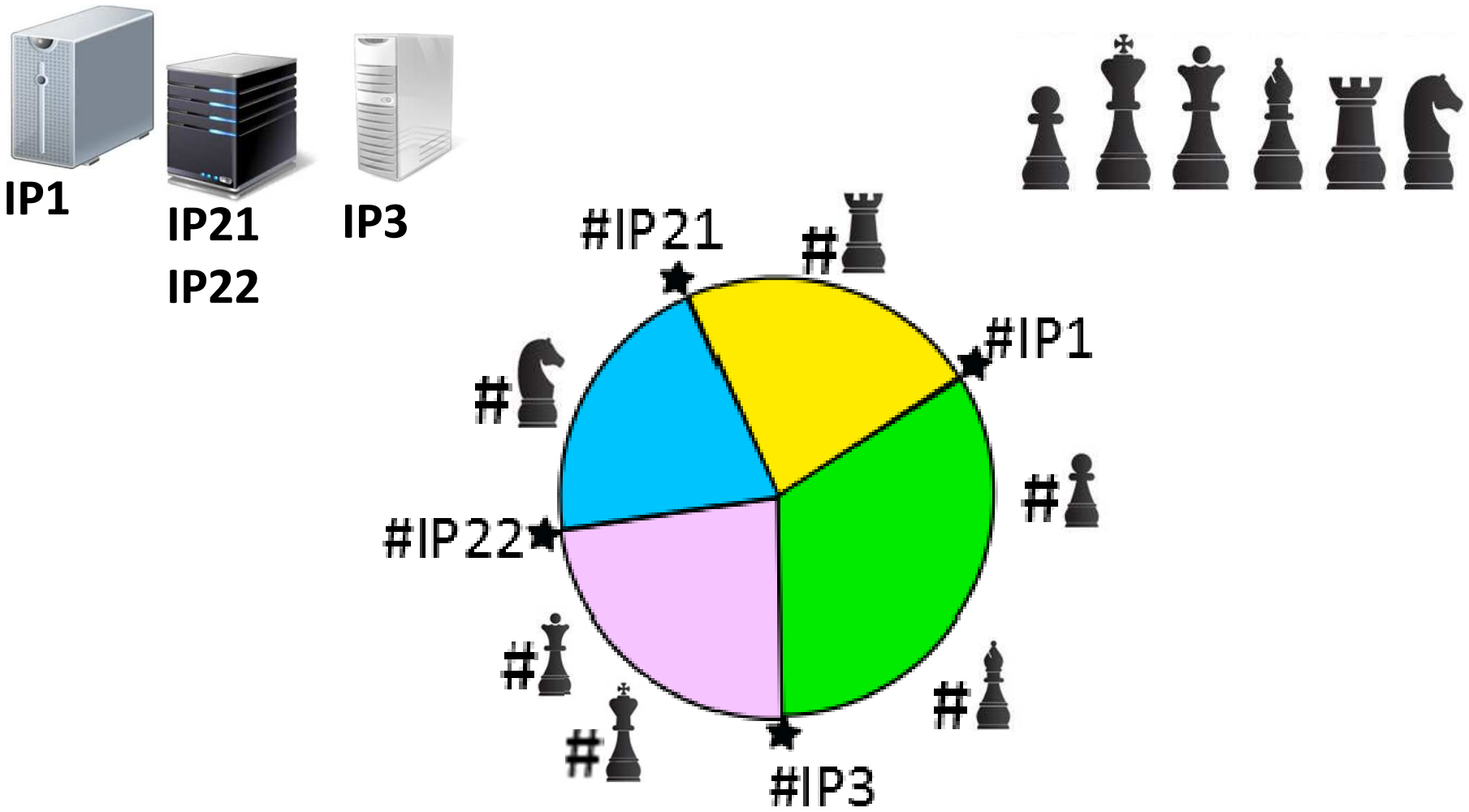
How to minimize changes when a node joins/leaves?







How to minimize changes when a node joins/leaves?



http://www.tomkleinpeter.com/2008/03/17/programmers-toolbox-part-3-consistent-hashing/
http://en.wikipedia.org/wiki/Consistent_hashing
http://blogs.msdn.com/b/csliu/archive/2009/09/17/consistent-hashing-theory-implementation.aspx

NoSQL DB examples


riak

Riak

- Inspired with *Amazon Dynamo*
- Distributed KV database
- V - anything (*plain text*, JSON, image, video, …)
- Simple HTTP interface(*„Riak speaks web”*)
- Fault tolerant
- Scalable
  - It is easy to add a node to the cluster
  - Automatic data distribution
  - *„a near-linear performance increase as you add capacity”*
- Disadvantages:
  - Ad-hoc queries
  - Relationships (ref.int.)

Instalacija i pokretanje

- Install Erlang, install (from source) Riak
- To start three servers:

```
$ dev/dev1/bin/riak start
$ dev/dev2/bin/riak start
$ dev/dev3/bin/riak start
```

- Join them in a cluster:

```
$ dev/dev2/bin/riak-admin join -f dev1@127.0.0.1
$ dev/dev3/bin/riak-admin join -f dev2@127.0.0.1

$ dev1/bin/riak-admin cluster plan
$ dev2/bin/riak-admin cluster commit
```

Key Value repository

- REST:
  - `/riak/bucket/`**`key`** `# Old format`
  - `/buckets/bucket/keys/`**`key`** `# New format`
- Put (storing data)

```
$ curl -v -X PUT http://localhost:8091/riak/test/helloworld \
-H "Content-Type: text/html" \
-d "<html><body>hello world</body></html>"
```

- Get (retireve data):

```
$ curl http://localhost:8091/riak/test/helloworld
<html><body>hello world</body></html>
$ curl http://localhost:8092/riak/test/helloworld
<html><body>hello world</body></html>
$ curl http://localhost:8093/riak/test/helloworld
<html><body>hello world</body></html>
```

Links

- One way links to other values:
- `-H "Link: </riak/slike/igor>; riaktag=\"slika\""`
- "Link walking" – it is possible to follow the links
- E.g., insert (put) image:

```
$ curl -X PUT http://localhost:8091/riak/slike/igor.png \
-H "Content-type: image/png" \
--data-binary @homer.png
```

Links (2)

- Link the mentor and an image:

```
$ curl -v -X PUT http://localhost:8091/riak/nastavnici/igor?returnbody=true \
-H "Content-Type: application/json" \
-H "Link: </riak/slike/igor>; riaktag=\"slika\"" \
-d '{"ime" : „Igor M", "zavod" : "ZPR"}'
```

- Dodajemo još i studenta:

```
$ curl -v -X PUT http://localhost:8091/riak/studenti/0036342145?returnbody=true \
-H "Content-Type: application/json" \
-H "Link: </riak/nastavnici/igor>; riaktag=\"mentor\"" \
-d '{"ime" : "Pero Perić", "modul" : "PIIS"}'
```

- *Link-walking*: append to url: `/_,_,_`
  i.e. `/bucket/tag/keep(=0/1)`, e.g.

```
$ curl http://localhost:8091/riak/studenti/0036342145/_,_,_
```

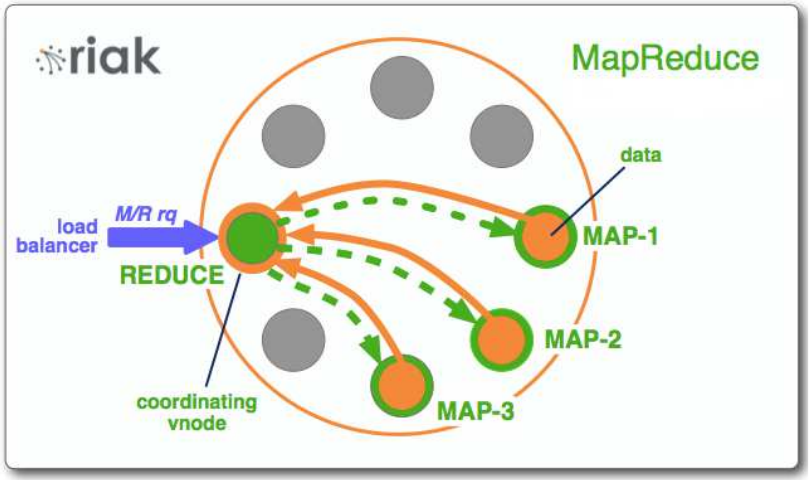
  - Returns all linked values – here just the mentor

-
```
$ curl http://localhost:8091/riak/studenti/0036342145/_,_,1/_,slika,_
```

  - Returns mentor and his image (1 defines that all intermediate objects be returned)

Riak MapReduce

- *„In practice, your MapReduce job code is likely less than 10 kilobytes, it is more efficient to send the code to the gigs of data being processed, than to stream gigabytes of data to your 10k of code.”* riakdocs
- Query + MR phases
- Query defines data sources for the initial map phase
- Two phases:
  - **Map** – all nodes responsible for input data are instructed to execute map function on own data
  - **Reduce** is CR. Takes a list as an input and returns a list. Can be invoked more than once!



Assigning M/R query

HTTP POST to /mapred

```
$ curl -X POST -H "content-type:application/json" \
http://localhost:8091/mapred --data @-
{
        "inputs":[...],
        "query":[
                {"link" : ...},
                {"map":{
                        ...
                        }
                },
                {"reduce":{
                        ...
                        }
                }
        ],
        "timeout": 90000
}
(Ctrl+D)
```

Can be a list of:

- bucket/key values, but
- Only a bucket name can be given, then all values from the bucket will be taken

Data can also be filtered.

Functions are assigned (js or erlang) either directly or as „stored procedures”.

keep=false (default)

- M i R can be omitted and/or combined (more than once, e.g. M/R/R)!

Example – just map – counting word „pizza”

```
$ curl -XPUT http://localhost:8098/buckets/training/keys/1 \
    -H 'Content-Type: text/plain' -d 'pizza data goes here'
$ curl -XPUT http://localhost:8098/buckets/training/keys/2 \
    -H 'Content-Type: text/plain' -d 'pizza pizza pizza pizza'
$ curl -XPUT http://localhost:8098/buckets/training/keys/3 \
    -H 'Content-Type: text/plain' -d 'nothing to see here'
$ curl -XPUT http://localhost:8098/buckets/training/keys/4
    -H 'Content-Type: text/plain' -d 'pizza pizza pizza'
```

```
$ curl -XPOST http://localhost:8098/mapred \
    -H 'Content-Type: application/json' \
    -d '{   "inputs":"training",
            "query":[
                {"map":{"language":"javascript",
                        "source": "
               function(riakObject) {
                  var val = riakObject.values[0].data.match(/pizza/g);
                  return [[riakObject.key, (val ? val.length : 0 )]];
               }
             "}}]
        }'
```

```
["1",1],["2",4],["3",0],["4",3]]
```

Example: Map+Reduce (1)

- Classic example of word counting
- Advice: try the functions first, e.g. in chrome's js console
- Let us firstly store the js function in Riak:

```
// map.js
function(v) {
   var words = v.values[0].data.toLowerCase().match(/\w*/g);
   var counts = [];
   for(var i in words)
      if (words[i] != '') {
         var count = {};
         count[words[i]] = 1;
         counts.push(count);
      }
    return counts;
}
```

```
curl -v -X PUT http://localhost:8091/riak/nmbp/nmbp_map.js \
   -H "Content-Type: text/javascript" \
   --data-binary @map.js
```

Example: Map+Reduce (2)

```
// reduce.js
function(values) {
  var result = {};
  for (var value in values) {
    for(var word in values[value]) {
      if (word in result)
        result[word] += values[value][word];
      else
        result[word] = values[value][word];
    }
  }
  return [result];
}
```

```
curl -v -X PUT http://localhost:8091/riak/nmbp/nmbp_reduce.js \
   -H "Content-Type: text/javascript" \
   --data-binary @reduce.js
```

Example: Map+Reduce (3)

- Finally, run the M/R:

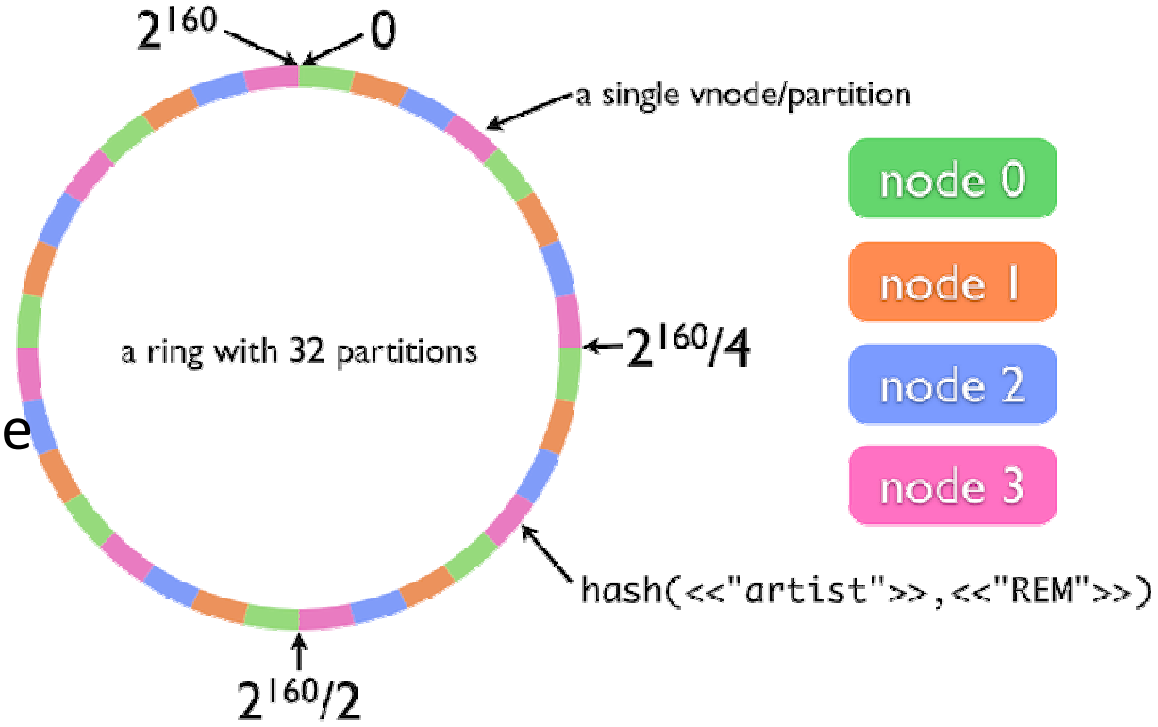
```
$ curl -X POST -H "content-type:application/json" \
http://localhost:8091/mapred --data @-
{
    "inputs":[...],
        "query":[
            {"map":{
                    "language":"javascript",
                    "bucket":"nmbp",
                    "key":"nmbp_map.js"
                }
                },
            {"reduce":{
                    "language":"javascript",
                    "bucket":"nmbp",
                    "key":"nmbp_reduce.js"
                }
            }
    ]
}
```

Important comment

- Virtual machine features last year's Riak version
- Look for the M/R documentation on the last year's docs (Riak javascript M/R code has been since deprecated) :
- http://docs.basho.com/riak/1.4.1/dev/advanced/mapreduce/
- (of course, you can also use Erlang)

Riak ring

- 160-bit integer
- Q partitions (def=64)
- Image shows Q=32, so there are 32 vnodes
- Each vnode is responsible for some key range
- N = 4 physical machines
- Each responsible for about Q/N vnodes




```
$ curl -H "Accept: text/plain" http://localhost:8091/stats
…
"ring_ownership":
[{'dev3@127.0.0.1',21},{'dev2@127.0.0.1',21},{'dev1@127.0.0.1',22}]"
```

Consistency & Durability

- n_val – number of replicas
- r – number of replica reads to consider the read successful
- w – number of replica writes to consider the write successful
- Can be assigned on every request!
  E.g.

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

Durability

- On write:
  i. Object is written to memory
  ii. Write acknowledged
  iii. Object persisted to disk
- Potential data loss between (ii) and (iii).
- However, it can be explicitly assigned for (iii) to be done before (iii) on any number of nodes, e.g. (on one node):

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

Conflict resolution

- Riak uses vector clocks (http://basho.com/why-vector-clocks-are-easy/)
- Include in every request:
  - `X-Riak-ClientId` to identify " the actor" i.e. client
  - `X-Riak-Vclock`
- Should the conflict occur, Riak bases the decision on `allow_mult` property:
  - `True`: keeps (and returns) both values (`"siblings"`), each value with its own `"vtag"`
  - `False`: „last wins”

Advanced features

- Hooks ~ DB triggers
  - Pre-commit hook (js or erlang)
  - Post-commit hook (erlang)
- Search (off by default, uses *hooks* to maintain GIN)
- Indexing
  - (off by default) uses new storage (LevelDB)
  - Additional indexes are assigned in the header (not part of the value)
- For „those who want to know more”:

  http://docs.basho.com/riak/latest/tutorials/querying/

Riak, in conslusion (1)

- For:
  - High availability
  - No single point of failure
  - Uses HTTP, multiple client libraries available *(PHP, Java, C#, Python, Ruby, ...)*
  - (can also use Protobuf)
  - No schema
- Against:
  - Ad hoc queries, search options
  - No schema, no ACID, …

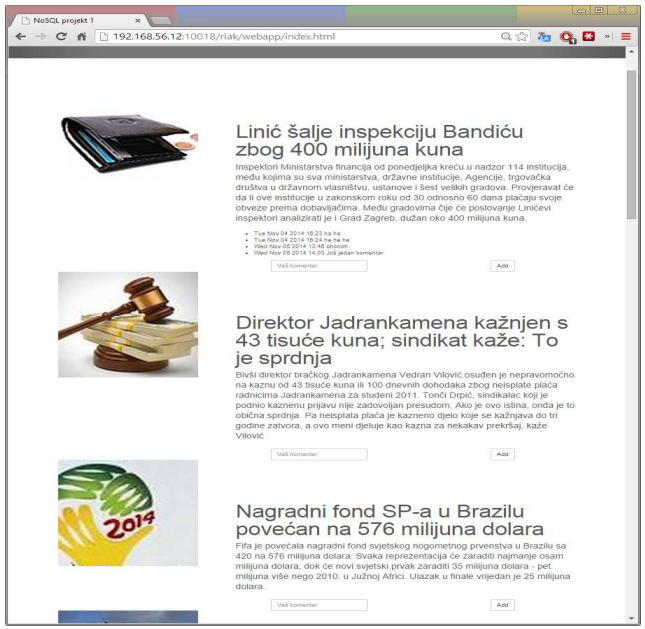
Riak, in conclusion (2)

- Use for:
  - Session info
  - User profiles, preferences
  - Shopping cart data
- Don’t use:
  - Data with many relationships
  - Transactions crossing aggregate boundaries
  - Queries based on the content (of the values)
  - Set operations

NoSQL project

- **Deadline: Tuesday, 14.1.2014. at 9:00.**
- **Detailed instructions on course's webpage**
- **Minimal web portal based on Riaku (10 points)**
  - Construct a web portal showing N most recent articles (npr. N=10)
    Each article takes on the following form:

Title	Picture
Text	
Author	



- **MapReduce queries (3+7=10 points)**
  - 3 = query returning list of article titles ordered descending by the comment counts
  - 10 = query returning 10 most used words for each author.


APACHE
HBASE

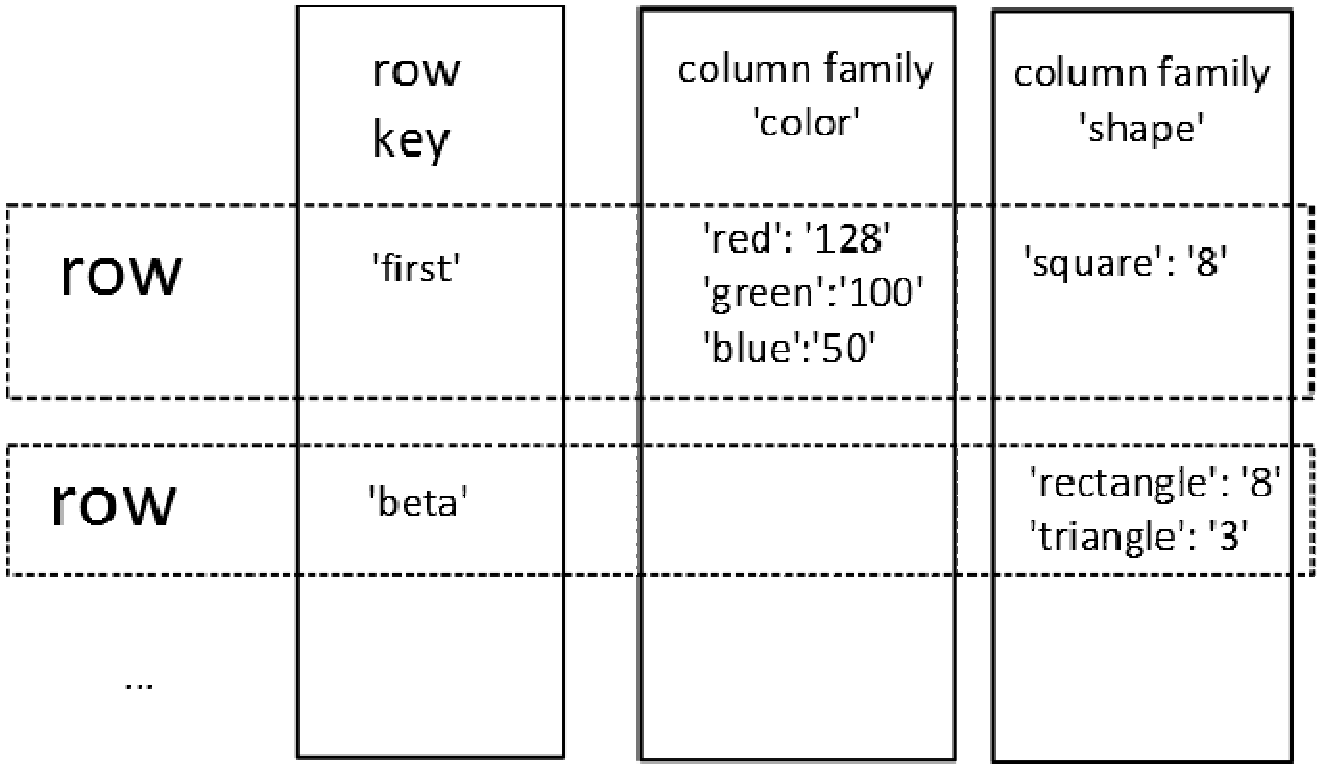
HBase

- Column family DB, inspired with Google Big Table*
- *“Sparse, distributed, persistent multidimensional sorted map.”*
- Uses Hadoop
- Only for large data = GB+:
  "This project's goal is the hosting of very large tables -- billions of rows X millions of columns -- atop clusters of commodity hardware."
- Good properties:
  - Scalable
  - Versioning
  - Compression
  - Memory resident tables
  - Fault tolerant

*Chang et al. [2006], Bigtable: A Distributed Storage System for Structured Data

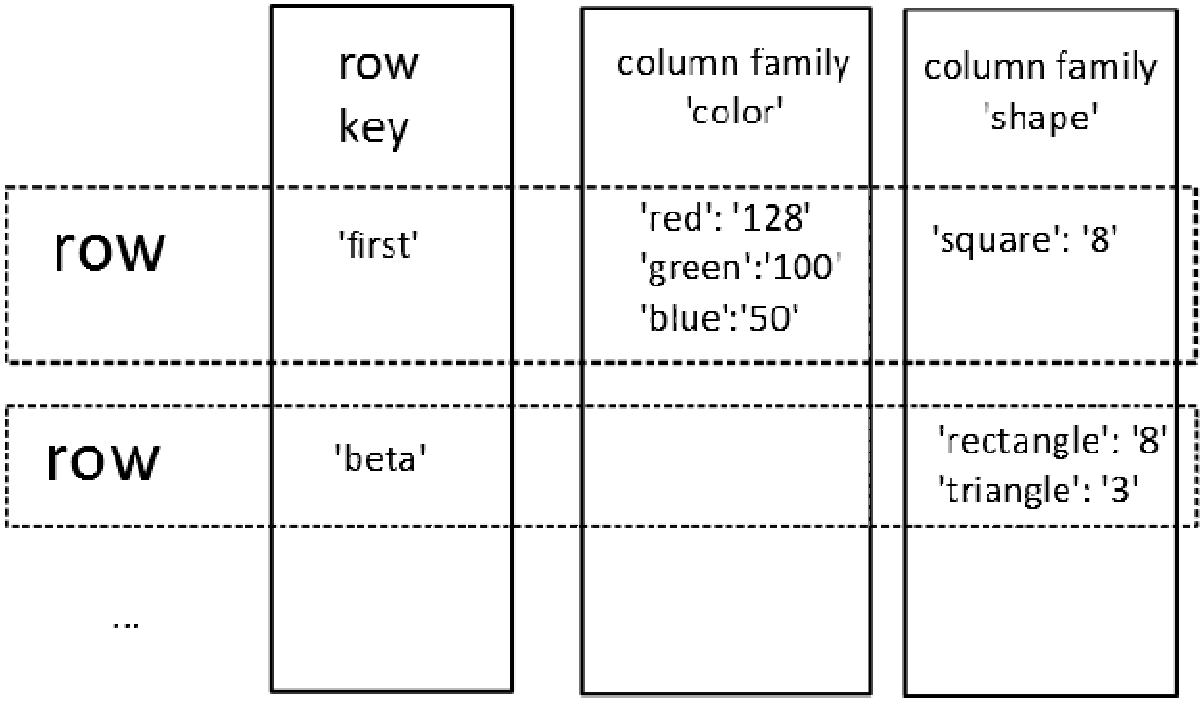
Structure (reminder)

- `get('first', ' color:green')`



Structure

- "table"* - key value - *map of maps*
- "row" (sorted)
- "column family"
- "column"
- "value", e.g.
  first/color:green
  is '100'

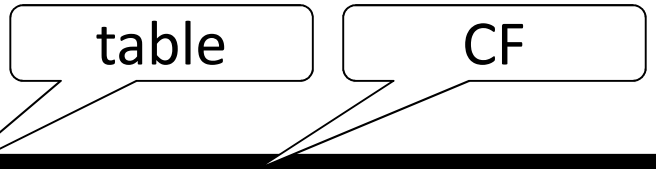


The terminology is reminiscent of the relational database, but the concepts are fundamentally very different.
To emphasize this, the names here are written in quotation marks.

Column families

- All Hbase operations are atomic on the row level – consistent rows
- Why not put all columns in the single CF?
  - CFs can be separately configured

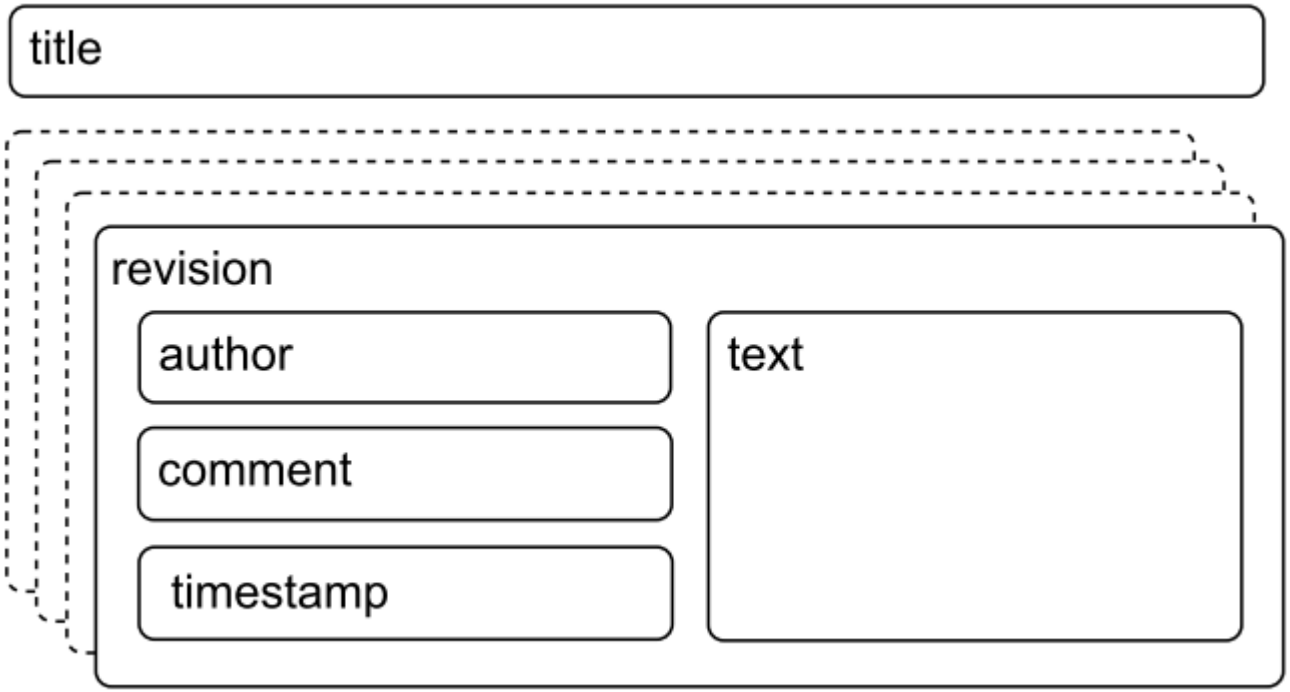
Versions




```
hbase(main):002:0> create 'test', 'semaphore'

hbase(main):004:0> put 'test', '1', 'semaphore:', 'red'

hbase(main):005:0> put 'test', '1', 'semaphore:', 'yellow'

hbase(main):006:0> put 'test', '1', 'semaphore:', 'green'
0 row(s) in 0.0040 seconds

hbase(main):007:0> get 'test', '1'
COLUMN                          CELL
 semaphore:                       timestamp=1372341842883, value=green

hbase(main):010:0> get 'test', '1', {COLUMN => 'semaphore:', VERSIONS => 4}
COLUMN                          CELL
 semaphore:                       timestamp=1372341842883, value=green
 semaphore:                       timestamp=1372341838768, value=yellow
 semaphore:                       timestamp=1372341829568, value=red
```

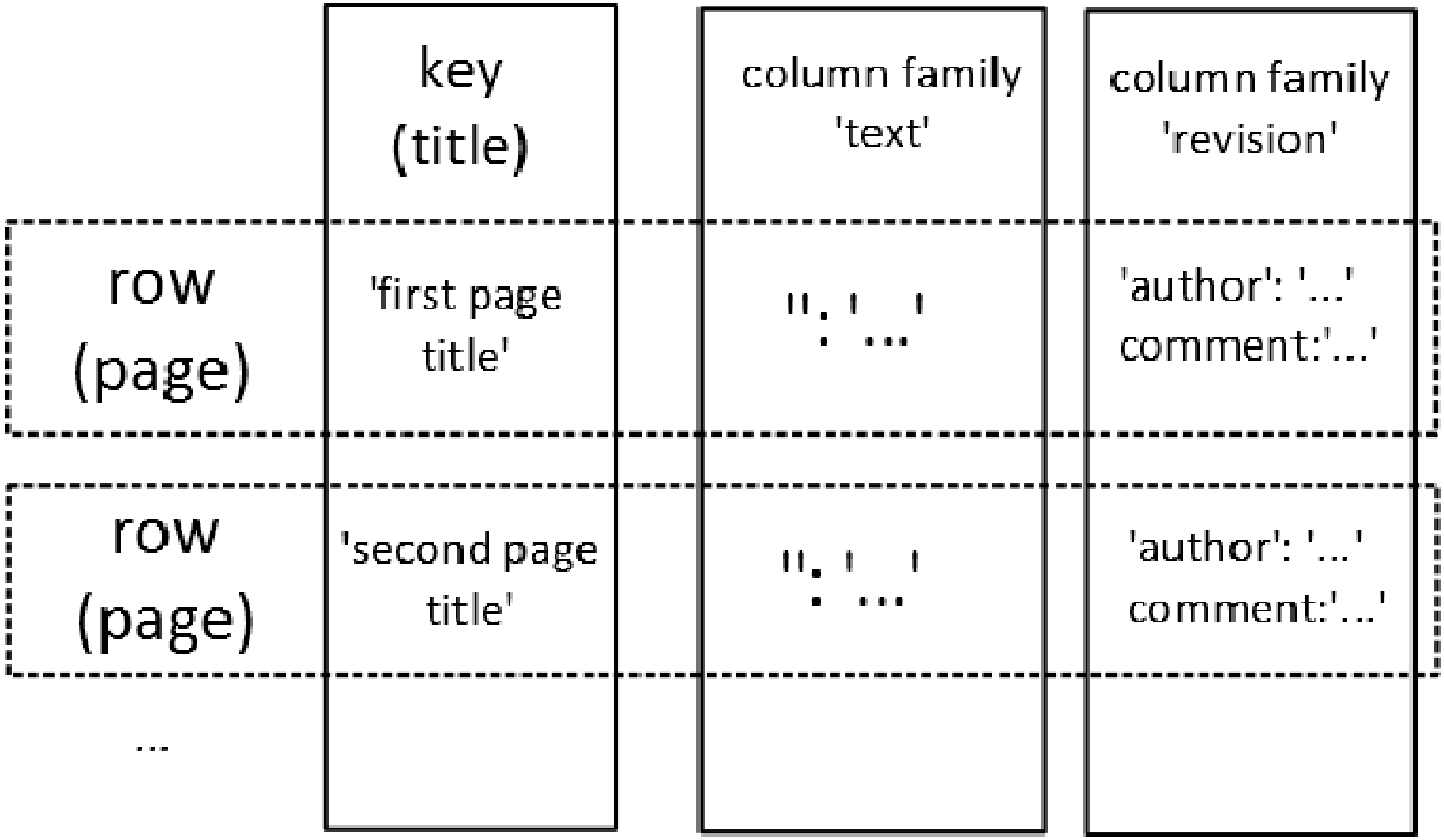
Example - wiki*

```
hbase(main):007:0> create 'wiki',
{NAME => 'text', VERSIONS => org.apache.hadoop.hbase.HConstants::ALL_VERSIONS
},
{NAME => 'revision', VERSIONS =>
org.apache.hadoop.hbase.HConstants::ALL_VERSIONS}
0 row(s) in 0.2040 seconds
```



**Taken from the book: Seven databases in seven weeks*

Example - wiki (2)



Primjer - wiki (3) - alter table

Unesimo prvi redak:

```
hbase(main):002:0> put 'wiki', 'HomePage', 'text:', 'Welcome'
0 row(s) in 0.6750 seconds

hbase(main):003:0> put 'wiki', 'HomePage', 'revision:author', 'igor'
0 row(s) in 0.0050 seconds

hbase(main):004:0> put 'wiki', 'HomePage', 'revision:comment', 'My first entry'
0 row(s) in 0.0130 seconds

hbase(main):005:0> get 'wiki', 'HomePage'
COLUMN                          CELL
 revision:author                timestamp=1372332177394, value=igor
 revision:comment               timestamp=1372332182860, value=My first entry
 text:                          timestamp=1372332177312, value=Welcome
3 row(s) in 0.0340 seconds
```

Not ideal - timestamp is different. Shell does not allow multiple commands at once, must use API (e.g. Java) for that.

Example - wiki (4) - alter table

- To alter the table, you must „disable” it

```
hbase(main):004:0> disable 'wiki'
0 row(s) in 2.0530 seconds

hbase(main):005:0> alter 'wiki',
{ NAME => 'text', COMPRESSION => 'GZ', BLOOMFILTER => 'ROW'}

Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 1.0970 seconds

hbase(main):006:0> enable 'wiki'
0 row(s) in 1.1080 seconds
```

Primjer - wiki (5) - import wikipedije

- Download the wikipedia dump from the internet and pipe it to hbase table (using ruby script that parses xml)*

```
hbase(main):004:0> curl
http://dumps.wikimedia.org/enwikibooks/20130626/enwikibooks-20130626-pages-
articles-multistream.xml.bz2 | bzcat | ./bin/hbase shell
./ruby/import_from_wikipedia.rb
```

```
…
 95  116M   95  110M    0     0   765k      0  0:02:35  0:02:28  0:00:07  800k 100500 records
inserted (Ba Zi/Date Selection in 1927)
101000 records inserted (Development Cooperation Handbook/The video resources linked to this
handbook/The Documentary Story/Searching for the questions to ask)
 97  116M   97  113M    0     0   756k      0  0:02:37  0:02:33  0:00:04  489k 101500 records
inserted (Template:DPL)
 97  116M   97  113M    0     0   753k      0  0:02:38  0:02:34  0:00:04  380k 102000 records
inserted (Ba Zi/HP H6)
 98  116M   98  114M    0     0   748k      0  0:02:39  0:02:36  0:00:03  380k 102500 records
inserted (Managing your business with anyPiece/Inventory system for recycle business/Storage
Packaging/Storage packaging - History)
103000 records inserted (How to Ace FYLSE/October 2012 Exam)
 99  116M   99  115M    0     0   750k      0  0:02:38  0:02:37  0:00:01  480k 103500 records
inserted (ElementarySpanish/Meeting people/Lesson 1)
100  116M  100  116M    0     0   752k      0  0:02:38  0:02:38 --:--:--  712k
```

*For details, see: *Seven databases in seven weeks*

Example - wiki (6) – wikipedia import

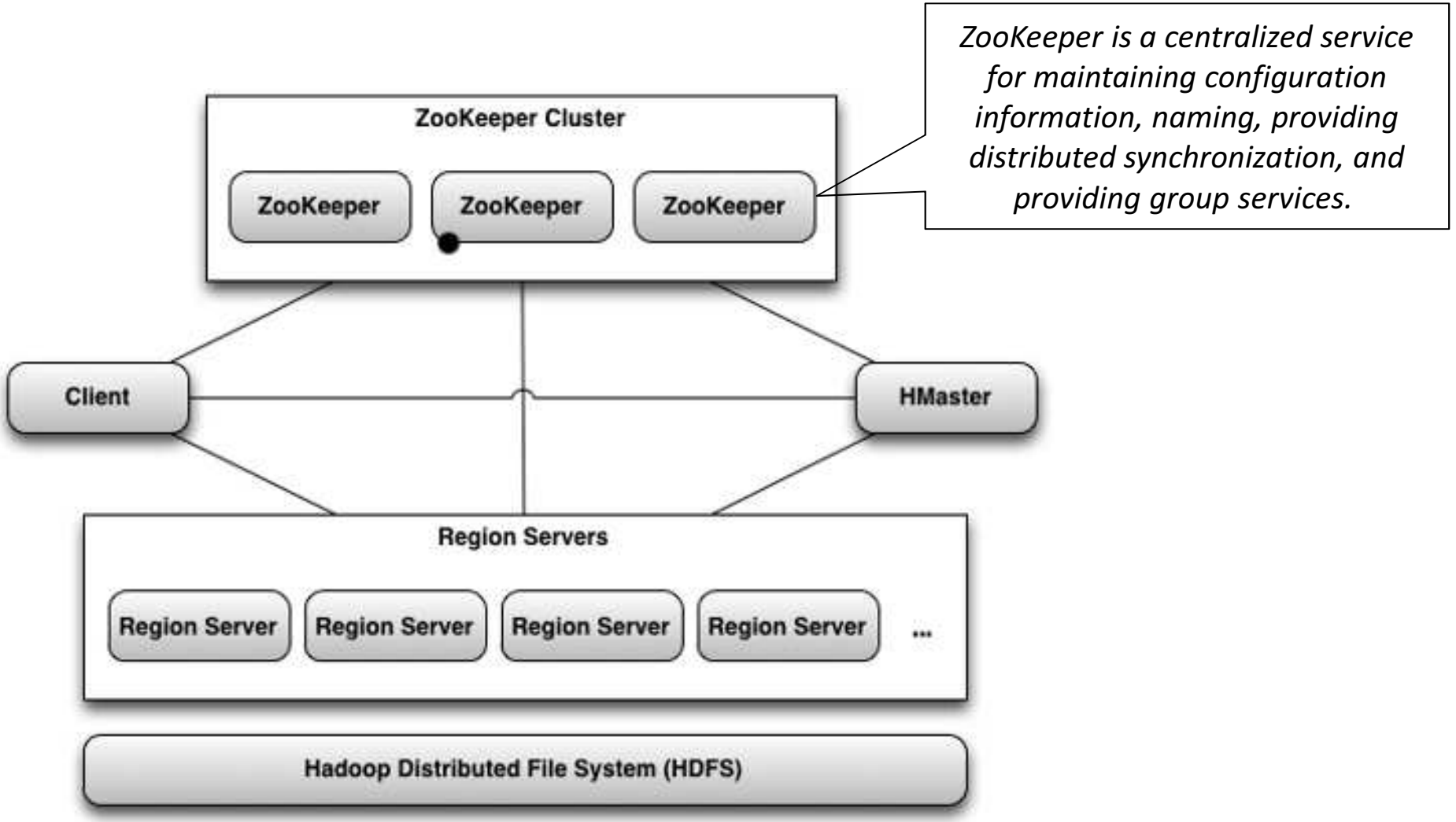
```
hbase(main):004:0> count 'wiki', INTERVAL => 10000
Current count: 10000, row: Blender 3D: Noob to Pro/Basic Animation/Rendering
Current count: 20000, row: Chemical engineering
Current count: 30000, row: Development Cooperation Handbook/The video resources linked to
this handbook/The Documentary
Story/The KFI story
Current count: 40000, row: File:NotesCornell.png
Current count: 50000, row: Hebrew Roots/The Law and the Covenants/Covenants:The Covenant
of Israeli Sovereignty
Current count: 60000, row: Mandarin Chinese/Sentences/He is a student
Current count: 70000, row: Programming:Game Maker/Intro
Current count: 80000, row: Structural Biochemistry/Protein function/Heme
group/Hemoglobin/Affinity Constant
Current count: 90000, row: Template:User language/sah
Current count: 100000, row: Wikijunior Languages/Finnish
103927 row(s) in 8.5280 seconds

hbase(main):005:0> get 'wiki', 'Chemical engineering'
COLUMN                          CELL
 revision:author                timestamp=1277299531, value=Adrignola
 revision:comment               timestamp=1277299531, value=Redirected page to
[[Subject:Chemical engineering]]
 text:                          timestamp=1277299531, value=#REDIRECT [[Subject:Chemical
engineering]]
3 row(s) in 0.0470 seconds
```

Wiki – data stored by regions

```
[root@minerva hbase]# du -h
4.0K    ./.oldlogs
4.0K    ./.corrupt
4.0K    ./.logs/minerva.zpr.fer.hr,45246,1371804716705
8.0K    ./.logs
4.0K    ./.archive
4.0K    ./.tmp
4.0K    ./-ROOT-/.tmp
12K     ./-ROOT-/70236052/.oldlogs
4.0K    ./-ROOT-/70236052/.tmp
12K     ./-ROOT-/70236052/info
12K     ./-ROOT-/70236052/recovered.edits
52K     ./-ROOT-/70236052
68K     ./-ROOT-
12K     ./.META./1028785192/.oldlogs
4.0K    ./.META./1028785192/.tmp
12K     ./.META./1028785192/info
40K     ./.META./1028785192
44K     ./.META.
12M     ./wiki/c089c61a331c1404a0d47f7f2b118efe/revision
4.0K    ./wiki/c089c61a331c1404a0d47f7f2b118efe/.tmp
68M     ./wiki/c089c61a331c1404a0d47f7f2b118efe/text
79M     ./wiki/c089c61a331c1404a0d47f7f2b118efe
4.0K    ./wiki/.tmp
10M     ./wiki/2d11659bf990bcf13a9d8fa1d62025e2/revision
4.0K    ./wiki/2d11659bf990bcf13a9d8fa1d62025e2/.tmp
65M     ./wiki/2d11659bf990bcf13a9d8fa1d62025e2/text
75M     ./wiki/2d11659bf990bcf13a9d8fa1d62025e2
153M    ./wiki
154M    .
```

Two regions

Tipična arhitektura Hbase klastera



Preuzeto s: http://www.packtpub.com/sites/default/files/Article-Images/7140_08_01.png

Hbase, in conclusion(1)

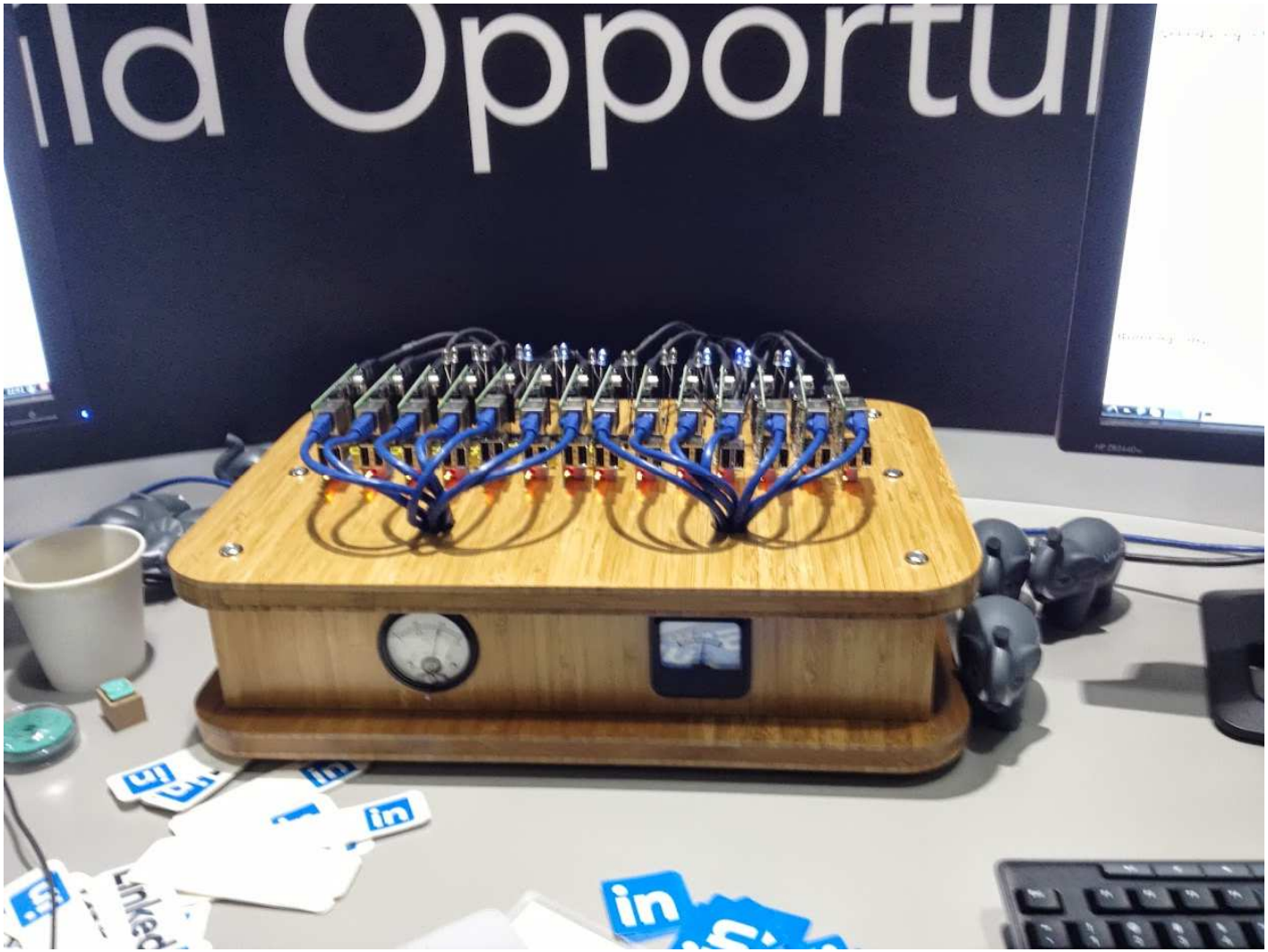
- Advantages:
  - Robust
  - Scalable (GB->TB)
  - Versioning
- Disadvantages:
  - At least five nodes
  - Administration
  - The adjoining "ecosystem" (+/-)
  - No sorting, (except row keys) or indexing
  - No data types (uninterpreted bytes)

Hbase, in conclusion(2)

- Use for:
  - *Event logging*
  - *Content management*

- Don’t use for:
  - „Small” date
  - ACID
  - Queries that aggregate data (must be done client-side)
  - Early development stages (while query patterns are not known)

Što je ovo?




Neo4j

Neo4j

- “whiteboard friendly,”
- All about relationships
- In different „sizes”:
  - Embedded
  - Cluster support, MS replication
- Can store tens of billions of nodes and edges

- Query languages:
  - Gremlin
  - Cypher
- Language binding
- Node/edge properties can be indexed (for query start objects)
  (uses *Lucene* to index)

Cypher

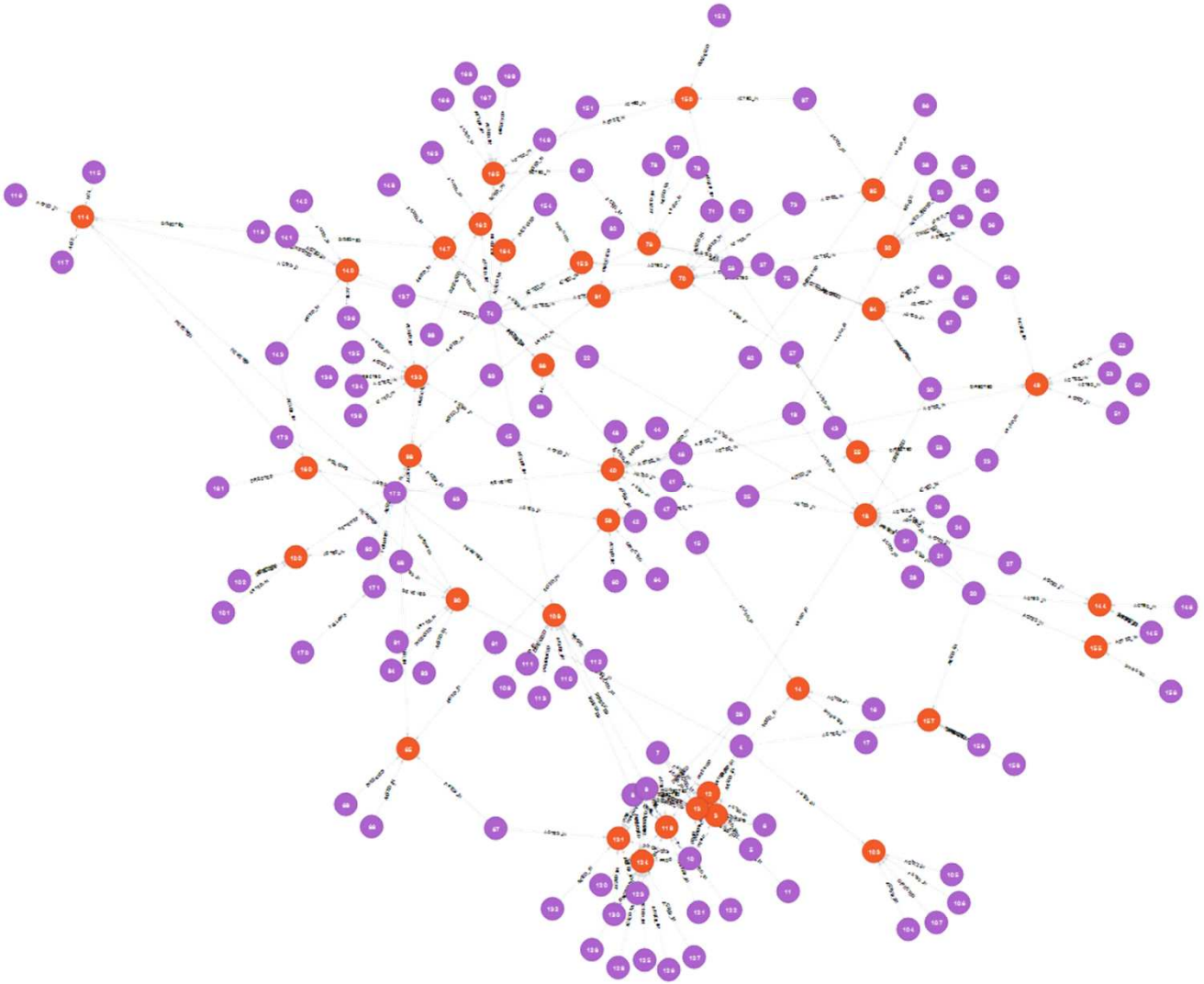
- See: http://docs.neo4j.org/refcard/1.9/
- On exam, you can be asked to give a simple Cypher query. Reference card will be available to you during the exam.

```
START
[MATCH]
[WHERE]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

Example: insert

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the
Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (AndyW:Person {name:'Andy Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
  (AndyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)

CREATE (Emil:Person {name:"Emil Eifrem", born:1978})
CREATE (Emil)-[:ACTED_IN {roles:["Emil"]}]->(TheMatrix)

CREATE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded', released:2003,
tagline:'Free your mind'})
...
```

Example: movies and actors



Cypher - examples

- Get Tom Hanks:

```
MATCH (tom {name: "Tom Hanks"})
RETURN tom
```

- Get nodes with title = „Cloud Atlas”:

```
MATCH (cloudAtlas {title: "Cloud Atlas"})
RETURN cloudAtlas
```

- Get names of ten persons (node type is Person):

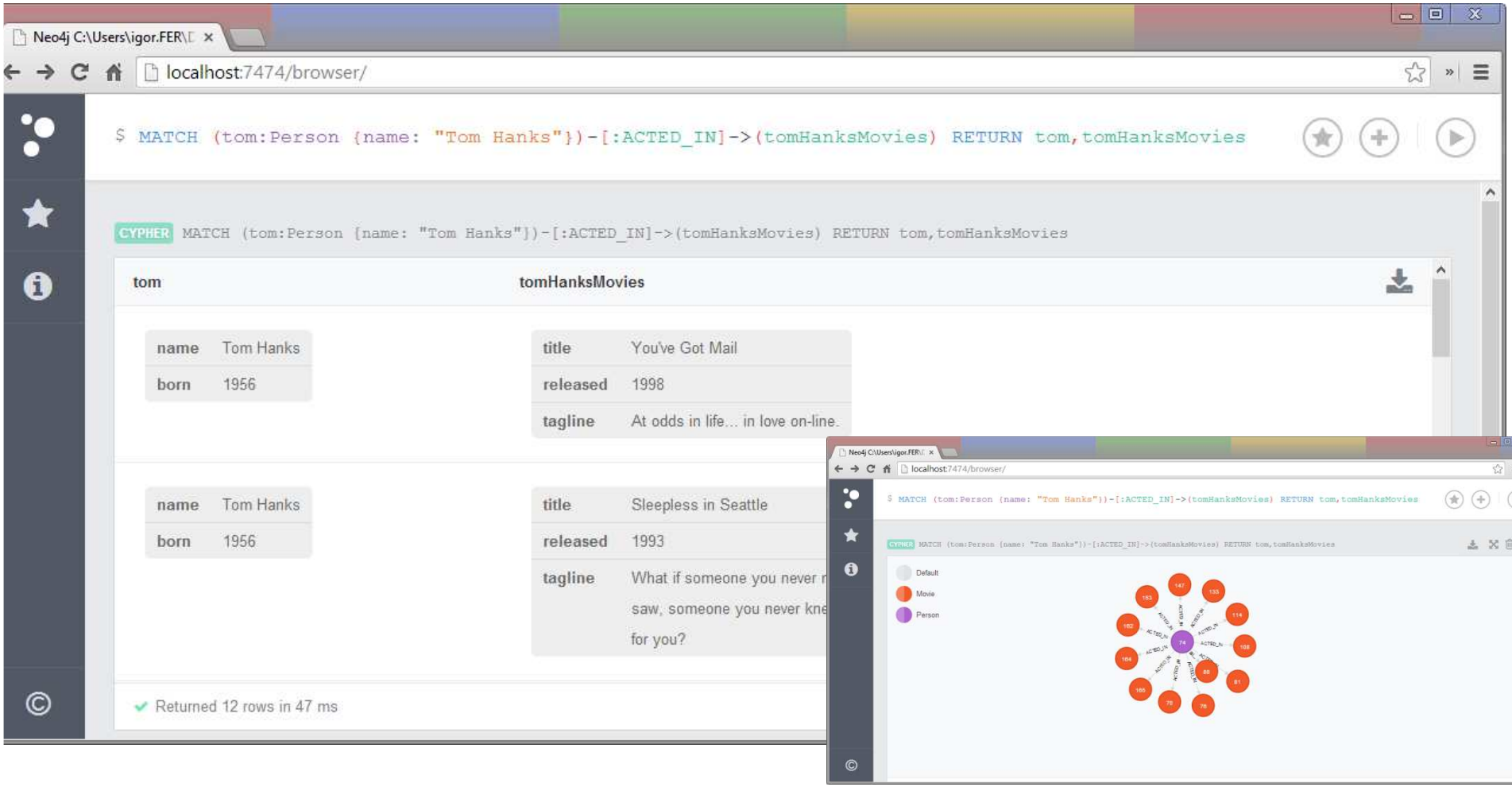
```
MATCH (people:Person)
RETURN people.name LIMIT 10
```

- Get movies from the 1990’s:

```
MATCH (nineties:Movie)
WHERE nineties.released > 1990 AND nineties.released < 2000
RETURN nineties.title
```

Example: movies starring TH

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies)
RETURN tom,tomHanksMovie
```



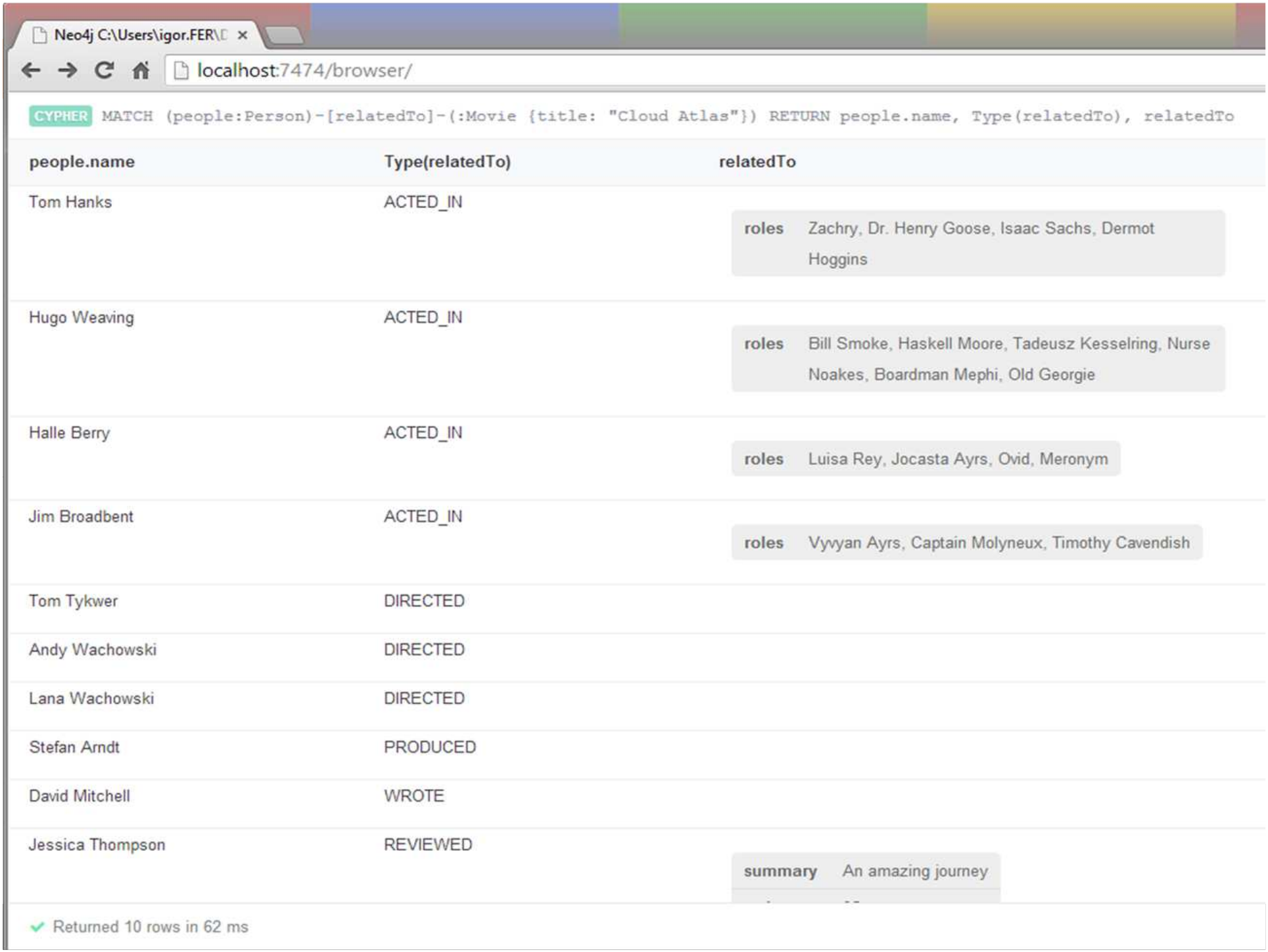
Primjer

- TH's co-actors:

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors)
RETURN coActors.name
```

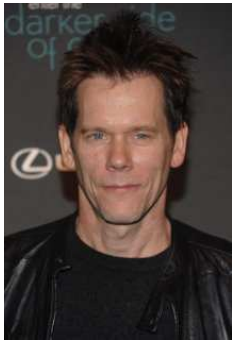
- In what way are persons related to the „Cloud Atlas"?

```
MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"})
RETURN people.name, Type(relatedTo), relatedTo
```


Neo4j C:\Users\igor.FER\
localhost:7474/browser/
CYPHER MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"}) RETURN people.name, Type(relatedTo), relatedTo
people.name
Type(relatedTo)
relatedTo
Tom Hanks
ACTED_IN
roles Zachry, Dr. Henry Goose, Isaac Sachs, Dermot Hoggins
Hugo Weaving
ACTED_IN
roles Bill Smoke, Haskell Moore, Tadeusz Kesselring, Nurse Noakes, Boardman Mephi, Old Georgie
Halle Berry
ACTED_IN
roles Luisa Rey, Jocasta Ayrs, Ovid, Meronym
Jim Broadbent
ACTED_IN
roles Vyvyan Ayrs, Captain Molyneux, Timothy Cavendish
Tom Tykwer
DIRECTED
Andy Wachowski
DIRECTED
Lana Wachowski
DIRECTED
Stefan Arndt
PRODUCED
David Mitchell
WROTE
Jessica Thompson
REVIEWED
summary An amazing journey
Returned 10 rows in 62 ms

Six Degrees of Kevin Bacon

*A small proportion of actors have an undefined **Bacon number**, meaning that they cannot be linked to Bacon in any number of connections at all. According to the Oracle of Baconwebsite, approximately 12% of all actors cannot be linked to Bacon using its criteria, but this number is difficult to verify.* http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon



```
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood)
RETURN DISTINCT hollywood
```

- Embedded shortest path algorithm – shortest path KB->MR

```
MATCH p=shortestPath(
  (bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

Brisanje

- Delte all persons, movies and relationships

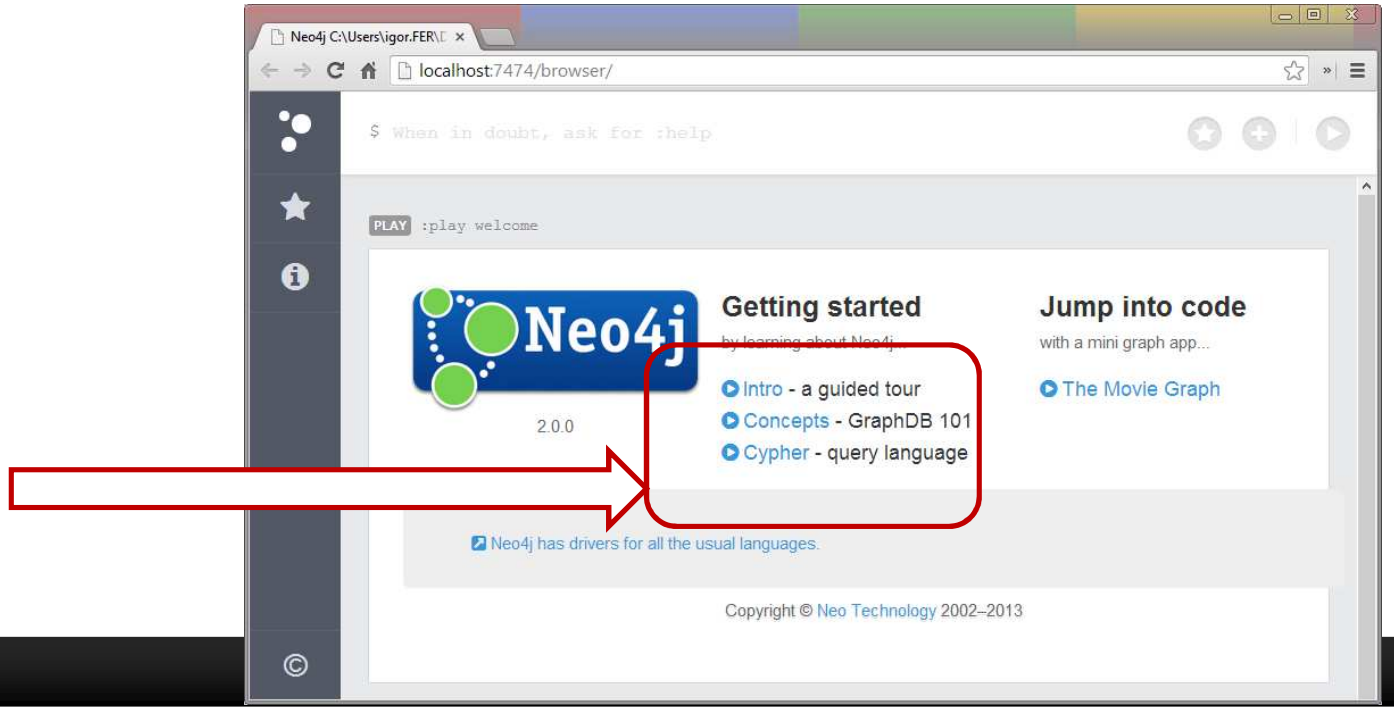
```
MATCH (a:Person),(m:Movie)
OPTIONAL MATCH (a)-[r1]-(), (m)-[r2]-()
DELETE a,r1,m,r2
```

- Check – print everything (should not print anything after the command above):

```
MATCH (n) RETURN n
```

Excerices

- Install Neo4j http://www.neo4j.org/download/other_versions
- Complete Getting started tutorials, which will give you the basic Cypher skills
- There is also an online tutorial (http://www.neo4j.org/learn/cypher) which is more extensive than this (you don't have to complete it)



Neo4j, in conclusion (1)

- Advantages:
  - Typless, schemaless, no restrictions to forming relationships
  - Indexing
  - Query languages
  - REST interface
  - Fast
  - Enterprise edition, High availability
- Disadvantages:
  - Community free, Enterpise not
  - Cannot replicate subgraphs (only entire graph)

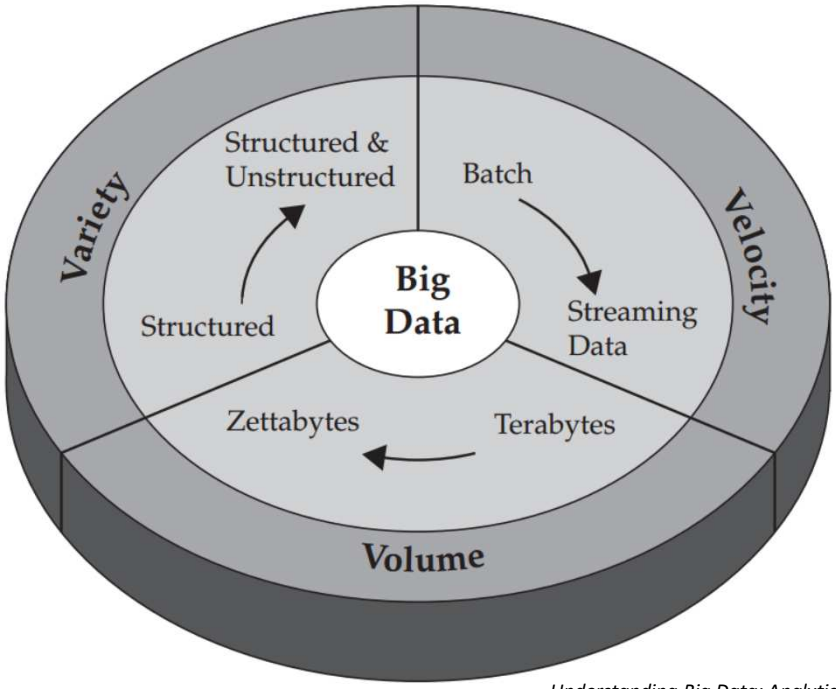
Neo4j, in conclusion (2)

- Use for:
  - *Connected data (e.g. Social networks)*
  - *Routing, dispatch, location-based services*
  - *Recommendation engine*
- Don’t use for:
  - Updating all values or sets of values

Big Data

Big Data (1)

- Those data which can not be managed, processed or analyzed using traditional methods and tools.
- The challenges include capture, curation, storage, search, sharing, transfer, analysis, and visualization.*
- 3Vs:
  - **Volume**
  - **Variety**
  - **Velocity**

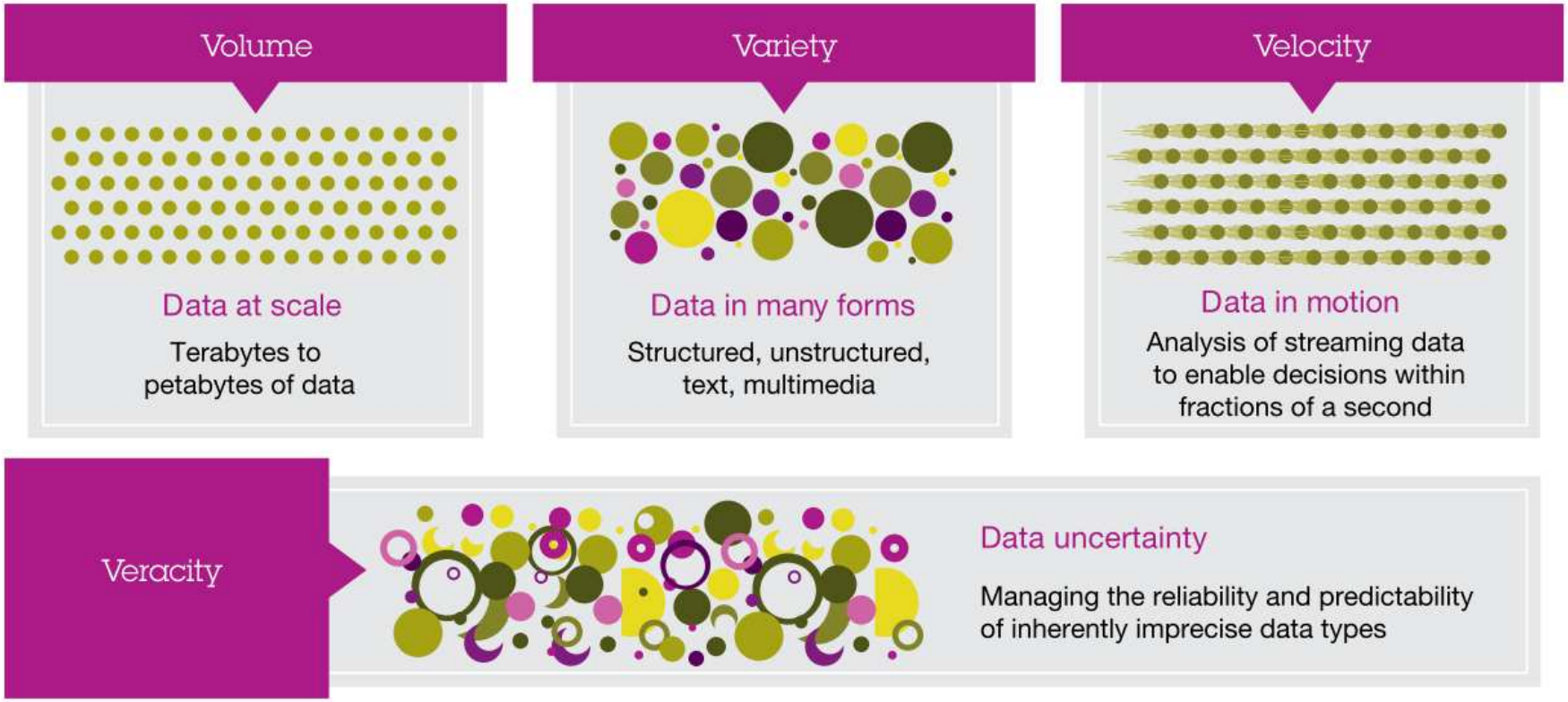


Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, Zikopoulos, Eaton, DeRoos, Deutsch, Lapis

*http://en.wikipedia.org/wiki/Big_data

Big Data (2)

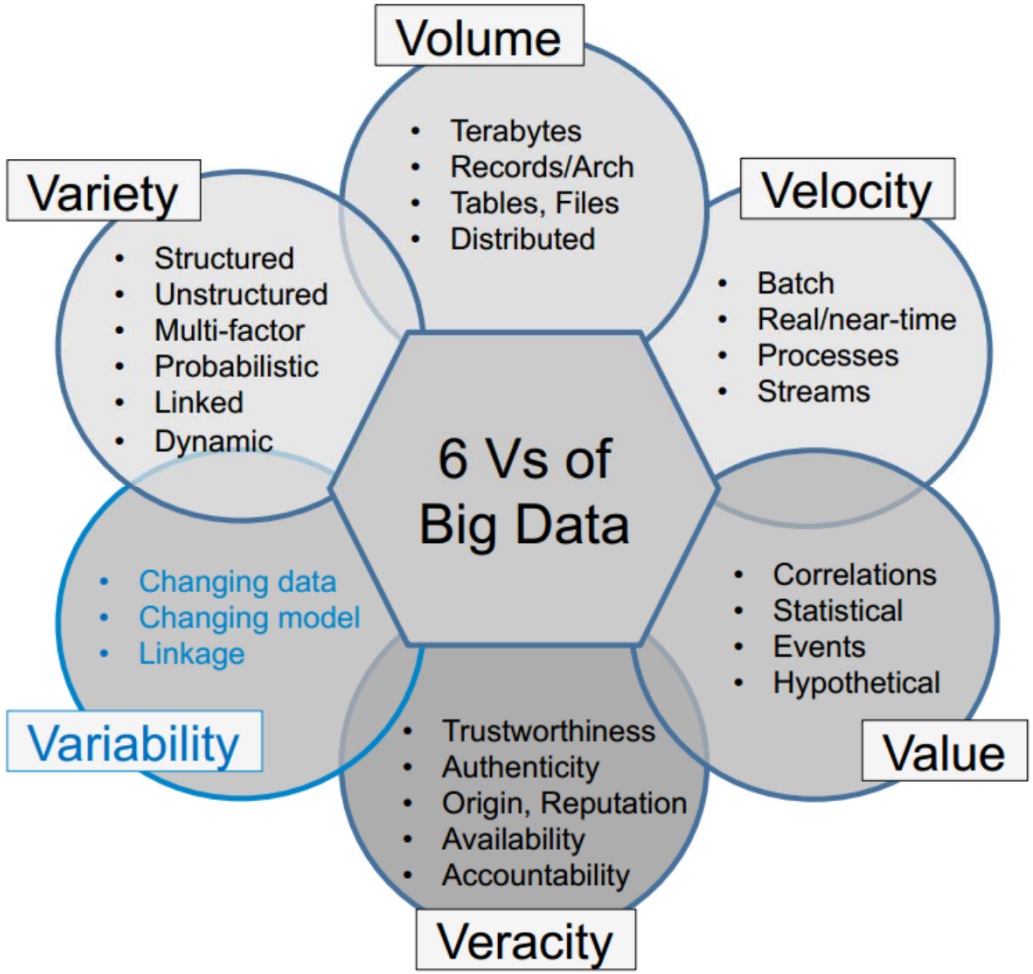
- **+Veracity**



IBM Institute for Business Value: „Analytics: The real-world use of big data"

Big Data (3)

- +Value
- +Variability

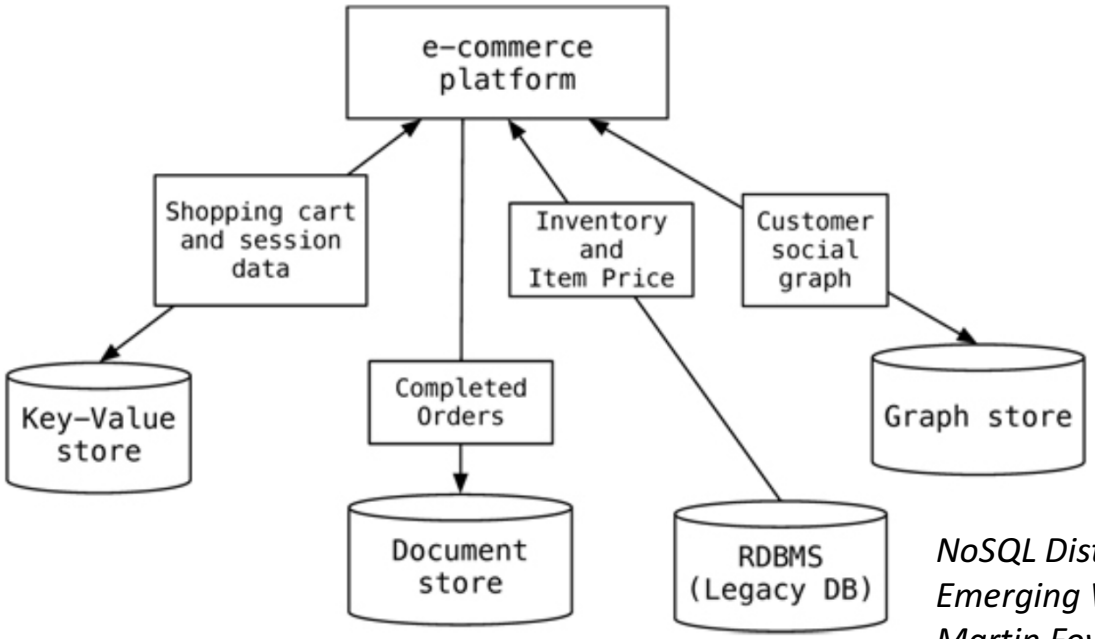


https://rd-alliance.org/sites/default/files/attachment/Breakout_reports.pdf

Polyglot persistance

Polyglot persistence (1)

- Different databases are intended to solve different problems
- A hybrid approach to data persistence:
  Combining technologies to best meet different data persistence requirements
- E.g.



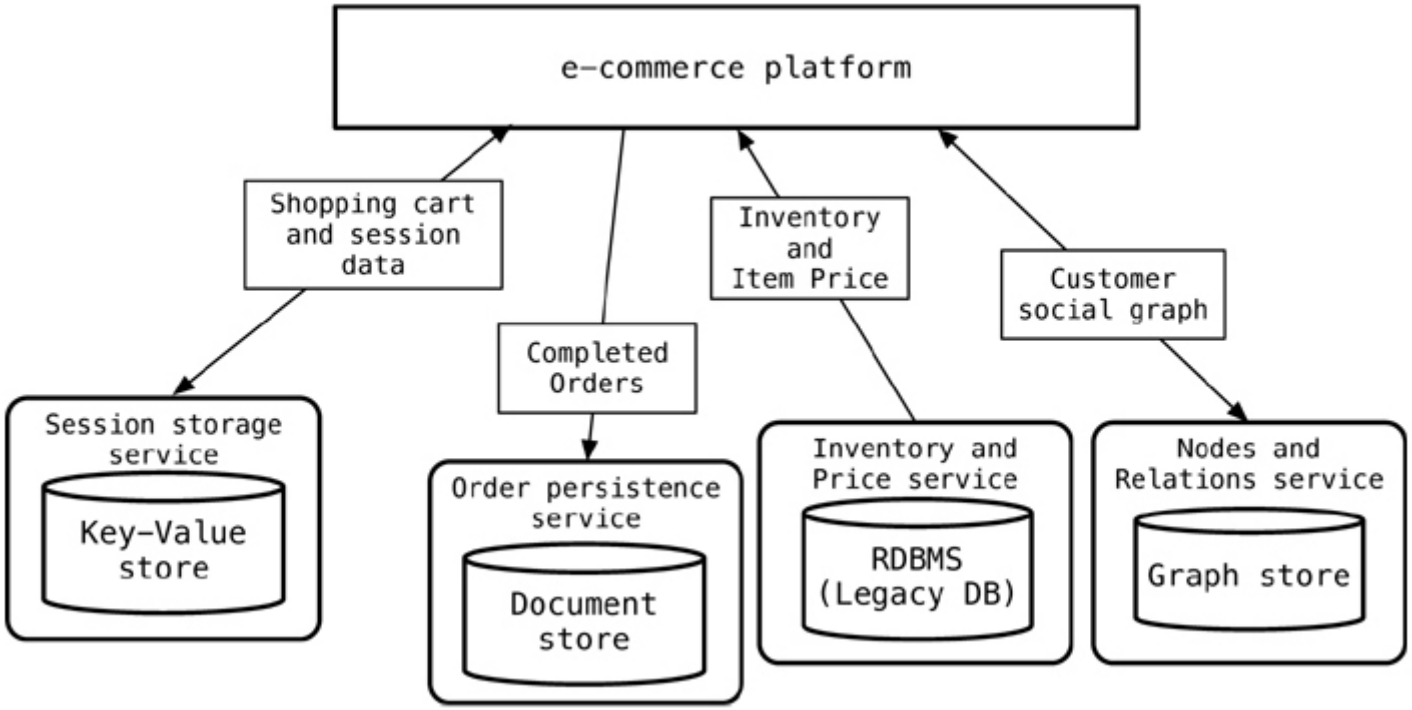
NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Martin Fowler

Or, example:

- http://www.martinfowler.com/bliki/PolyglotPersistence.html

Polyglot persistence (2)

- Or, if we wrap the DBs in web services (to minimize the impact on the rest of the sytem):



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

Polyglot persistance (3)

- Problems:
  - *Poly-skilled DBA*: supervision, *backup*, *restore*,...
  - Support, tools, upgrades, drivers, etc.
  - Security, access rules
  - Development complexity, maintenance, *deployment*, ...

NoSQL summary

NoSQL summary: 10=5+5

- **10 things: 5 advantages**
- http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-nosql-databases/

1. **Elastic scaling**
   - Horizontally scalable
2. **Big Data**
   - Can work withBD
3. **Goodbye DBAs?**
   - Automatic recovery, tuning, distribution
4. **Economics**
   - Mostly free of charge, FLOSS software
   - Based on *commodity* computers
5. **Flexible Data Models**
   - Schemaless

NoSQL summary : 10=5+5

- **10 things: 5 challenges**
- http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-nosql-databases/

1. **Maturity**
   - Still actively developed
2. **Support**
   - Mostly FLOSS; startups, limited support and credibility
3. **Administration**
   - Requires a lot of knowledge to install and administer
4. **Analytics & BI**
   - Focused on the web
   - Limited ad-hoc query capabilities
5. **Expertise**
   - Small experts market

NoSQL, summary:

- They are not a substitute for relational systems, but dedicated software for specific problems
- The most common use case: increase availability and efficiency at the expense of consistency
- BASE not ACID
- Large amounts of data
- Flexible data schema
- Horizontal scalability
- It is not a mature technology, lack of standards
- Mostly free / open source
- Relatively easy to set up (for testing)