732A54 / TDDE31
Big Data Analytics

Topic: DBMSs for Big Data

Olaf Hartig

olaf.hartig@liu.se

LINKÖPING UNIVERSITY

Relational Database Management Systems

- Well-defined formal foundations (*relational data model*)

schema

Relation Name

Attributes

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

instance/ state

Tuples

Figure from “Fundamentals of Database Systems” by Elmasri and Navathe, Addison Wesley.

Relational Database Management Systems

- Well-defined formal foundations (*relational data model*)
- *SQL* – powerful declarative language
  - querying
  - data manipulation
  - database definition
- Support of transactions with *ACID* properties (***A***tomicity, ***C***onsistency preservation, ***I***solation, ***D***urability)
- Established technology (developed since the 1970s)
  - many vendors
  - highly mature systems
  - experienced users and administrators

Business world has evolved

- Organizations and companies (whole industries) shift to the digital economy powered by the Internet
- Central aspect: new IT applications that allow companies to *run their business* and to *interact with costumers*
  - Web applications
  - Mobile applications
  - Connected devices ("Internet of Things")

Image source: https://pixabay.com/en/technology-information-digital-2082642/

New Challenges for Database Systems

- Increasing numbers of concurrent users/clients
  - tens of thousands, perhaps millions
  - globally distributed
  - expectations: consistently high performance and 24/7 availability (no downtime)
- Different types of data
  - huge amounts (generated by users and devices)
  - data from different sources together
  - frequent schema changes or no schema at all
  - semi-structured and unstructured data
- Usage may change rapidly and unpredictably

Image source: https://www.flickr.com/photos/groucho/5523369279/

``NoSQL”

- Some interpretations (without precise definition):
  - “no to SQL”
  - “not only SQL”
  - “not relational”
- 1998: first used for an RDBMS* that omitted usage of SQL
- 2009: picked up again to name a conference on
  “open-source, distributed, non-relational databases”
- Since then, “NoSQL database” loosely specifies a class of non-relational DBMSs
  - Relax some requirements of RDBMSs to gain efficiency and scalability for use cases in which RDBMSs are a bad fit

*RDBMS = relational database management system

Scalability

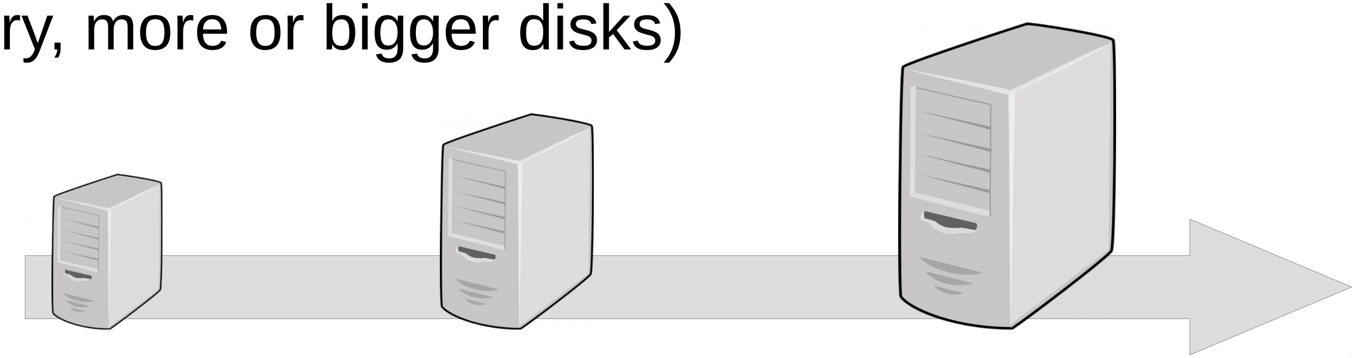
- Data scalability: system can handle
  growing amounts of data
  without losing performance

- Read scalability: system can handle
  increasing numbers of read operations
  without losing performance

- Write scalability: system can handle
  increasing numbers of write operations
  without losing performance

Vertical Scalability *vs.* Horizontal Scalability

- Vertical scalability ("scale up")
  - Add resources to a server (e.g., more CPUs, more memory, more or bigger disks)



- Horizontal scalability ("scale out")
  - Add nodes (more computers) to a distributed system

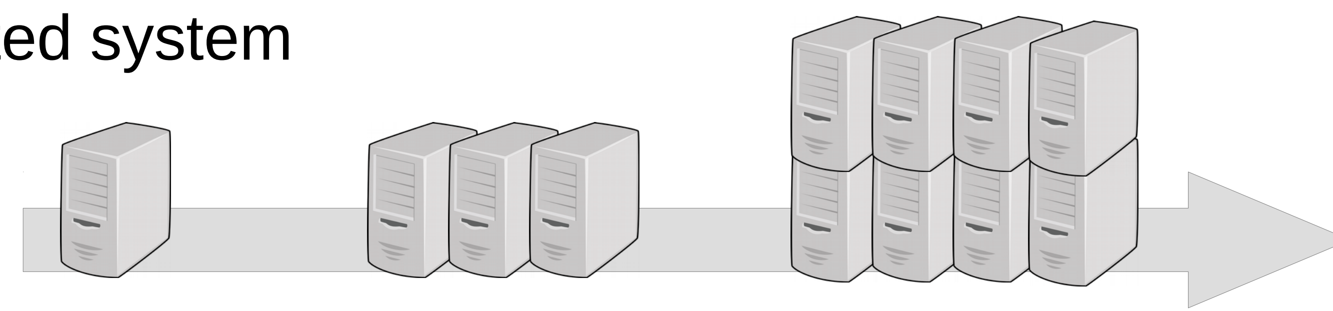


Image source: https://pixabay.com/en/server-web-network-computer-567943/

Typical* Characteristics of NoSQL Systems

- Ability to scale horizontally over many commodity servers with high performance, availability, and fault tolerance
  - achieved by giving up ACID guarantees
  - and by partitioning and replication of data
- Non-relational data model, no requirements for schemas
  - data model limitations make partitioning effective

*Attention, there is a *broad variety* of such systems and not all of them have these characteristics to the same degree

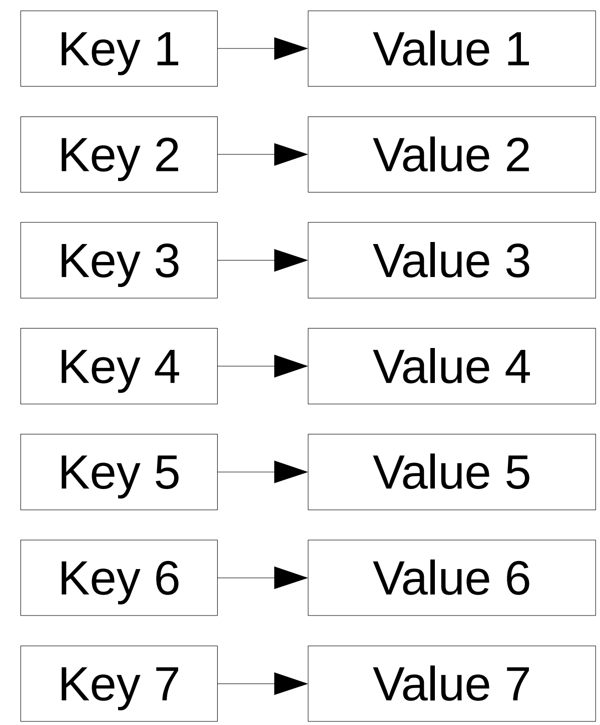
NoSQL Data Models

Data Models

- Key-value model
- Document model
- Wide-column models
- Graph database models

Key-Value Stores: Data Model

- Database is simply a set of key-value pairs
  - keys are unique
  - values of arbitrary data types
- Values are opaque to the system



Example

- Assume a relational database consisting of a single table:

User

<u>login</u>	name	website	twitter
alice12	Alice	http://alice.name/	NULL
bob_in_se	Bob	NULL	@TheBob
charlie	Charlie	NULL	NULL

- How can we capture this data in the key-value model?

Example

- Assume a relational database consisting of a single table:

User

login	name	website	twitter
alice12	Alice	http://alice.name/	NULL
bob_in_se	Bob	NULL	@TheBob
charlie	Charlie	NULL	NULL

- How can we capture this data in the key-value model?

alice12 → Alice, http://alice.name/

bob_in_se → Bob, , @TheBob

charlie → Charlie

Example

- Let's add another table:

Fav

user	favorite
alice12	bob_in_se
alice12	charlie

User

login	name	website	twitter
alice12	Alice	http://alice.name/	NULL
bob_in_se	Bob	NULL	@TheBob
charlie	Charlie	NULL	NULL

- How can we capture this data in the key-value model?

alice12	Alice, http://alice.name/
bob_in_se	Bob, , @TheBob
charlie	Charlie

Example

- Let's add another table:

Fav

user	favorite
alice12	bob_in_se
alice12	charlie

User

login	name	website	twitter
alice12	Alice	http://alice.name/	NULL
bob_in_se	Bob	NULL	@TheBob
charlie	Charlie	NULL	NULL

- How can we capture this data in the key-value model?

alice12 →	Alice, http://alice.name/, , [bob_in_se,charlie]
bob_in_se →	Bob, , @TheBob
charlie →	Charlie

Key-Value Stores: Querying

- Only CRUD operations in terms of keys
  - CRUD: create, read, update, delete
  - put(*key*, *value*); get(*key*); delete(*key*)
- No support for value-related queries
  - Recall that values are opaque to the system (i.e., no secondary index over values)
- Accessing multiple items requires *separate requests*
  - Beware: often no transactional capabilities

alice12 →	Alice, http://alice.name/, , [bob_in_se,charlie]
bob_in_se →	Bob, , @TheBob
charlie →	Charlie

Key-Value Stores: Querying

- Only CRUD operations in terms of keys
  - CRUD: create, read, update, delete
  - put(*key*, *value*); get(*key*); delete(*key*)
- No support for value-related queries
  - Recall that values are opaque to the system (i.e., no secondary index over values)
- Accessing multiple items requires *separate requests*
  - Beware: often no transactional capabilities
- Advantage of these limitations: partition the data based on keys (*"horizontal partitioning"*, also called *"sharding"*) and distributed processing can be very efficient

Example (cont'd)

- Assume we try to find all users for whom Bob is a favorite
- It is possible (how?), but very inefficient
- What can we do to make it more efficient?

alice12	→	Alice, http://alice.name/, , [bob_in_se,charlie]
bob_in_se	→	Bob, , @TheBob
charlie	→	Charlie

Example (cont'd)

- Assume we try to find all users for whom Bob is a favorite
- It is possible (how?), but very inefficient
- What can we do to make it more efficient?
  - Add redundancy (downsides: more space needed, updating becomes less trivial and less efficient)

alice12 →	Alice, http://alice.name/, , [bob_in_se,charlie], [ ]
bob_in_se →	Bob, , @TheBob, [ ], [alice12]
charlie →	Charlie, , , [ ], [alice12]

Key-Value Stores: Use Cases

- Whenever values need to be accessed only via keys
- Examples:
  - Storing Web session information
  - User profiles and configuration
  - Shopping cart data
  - Caching layer that stores results of expensive operations (e.g., complex queries over an underlying database, user-tailored Web pages)

Examples of Key-Value Stores

- In-memory key-value stores
  - Memcached
  - Redis
- Persistent key-value stores
  - Berkeley DB
  - Voldemort
  - RiakDB

Data Models

- Key-value model
- Document model
- Wide-column models
- Graph database models

Image source: https://pxhere.com/en/photo/1188160

Document Stores: Data Model

- Document: a set of fields consisting of a name and a value
  - field names are unique within the document
  - values are scalars (text, numeric, boolean) or lists

login : "alice12"
name : "Alice"
website : "http://alice.name/"
favorites : ["bob_in_se", "charlie"]

User

login	name	website	twitter
alice12	Alice	http://alice.name/	NULL
bob_in_se	Bob	NULL	@TheBob
charlie	Charlie	NULL	NULL

Fav

user	favorite
alice12	bob_in_se
alice12	charlie

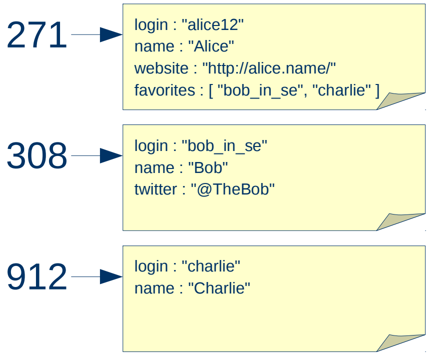
Document Stores: Data Model

- Document: a set of fields consisting of a name and a value
  - field names are unique within the document
  - values are scalars (text, numeric, boolean) or lists
  - in some systems, values may also be other documents

```
login : "alice12"
name : "Alice"
website : "http://alice.name/"
favorites : [ "bob_in_se", "charlie" ]
address : {
        street : "Main St"
        city : "Springfield"
 }
```

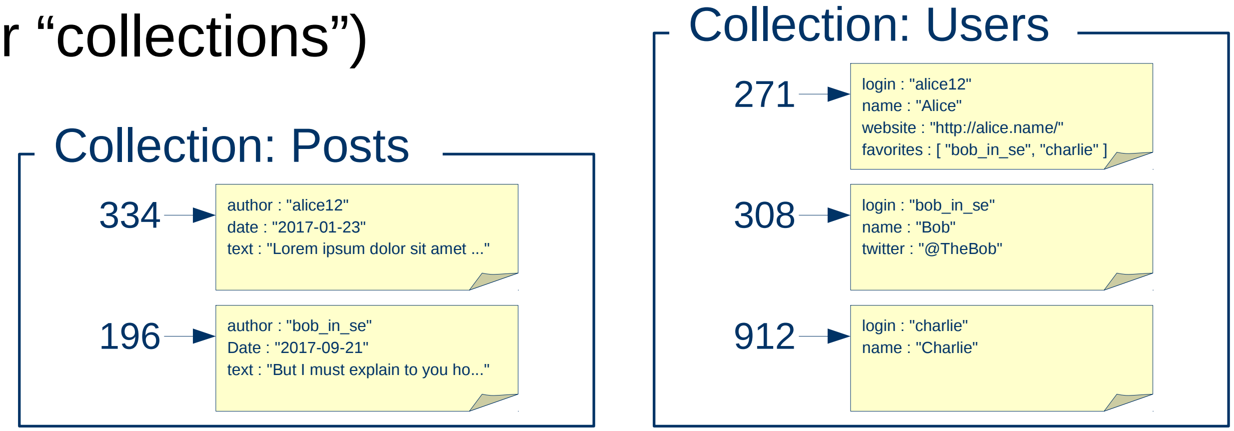
Document Stores: Data Model

- Document: a set of fields consisting of a name and a value
  - field names are unique within the document
  - values are scalars (text, numeric, boolean) or lists
  - in some systems, values may also be other documents
- Database is a set of documents (or multiple such sets)
  - each document additionally associated with a unique identifier (typically system-generated)
  - *schema free*: different documents may have different fields



Document Stores: Data Model

- Document: a set of fields consisting of a name and a value
  - field names are unique within the document
  - values are scalars (text, numeric, boolean) or lists
  - in some systems, values may also be other documents
- Database is a set of documents (or multiple such sets)
  - each document additionally associated with a unique identifier (typically system-generated)
  - *schema free*: different documents may have different fields
  - grouping of documents into separate sets (called “domains” or “collections”)



Document Stores: Data Model

- Document: a set of fields consisting of a name and a value
  - field names are unique within the document
  - values are scalars (text, numeric, boolean) or lists
  - in some systems, values may also be other documents
- Database is a set of documents (or multiple such sets)
  - each document additionally associated with a unique identifier (typically system-generated)
  - *schema free*: different documents may have different fields
  - grouping of documents into separate sets (called "domains" or "collections")
- Partitioning based on collections and/or on document IDs
- Secondary indexes over fields in the documents possible
  - different indexes per domain/collection of documents

Document Stores: Querying

- Querying in terms of conditions on document content
- Queries expressed in terms of program code using an API or in a system-specific query language

Document Stores: Querying

- Querying in terms of conditions on document content
- Queries expressed in terms of program code using an API or in a system-specific query language
- Examples (using MongoDB's query language):
  - Find all docs in collection *Users* whose *name* field is *"Alice"*

    ```
    db.Users.find( {name: "Alice"} )
    ```

  - Find all docs in collection *Users* whose *age* is greater than 23

    ```
    db.Users.find( {age: {$gt: 23}} )
    ```

  - Find all docs about *Users* who favorite Bob

    ```
    db.Users.find( {favorites: {$in: ["bob_in_se"]}} )
    ```

login : "alice12"
name : "Alice"
website : "http://alice.name/"
favorites : ["bob_in_se", "charlie"]

Document Stores: Querying

- Querying in terms of conditions on document content
- Queries expressed in terms of program code using an API or in a system-specific query language
- Examples (using MongoDB's query language):
  - Find all docs in collection *Users* whose *name* field is *"Alice"*

    ```
    db.Users.find( {name: "Alice"} )
    ```

  - Find all docs in collection *Users* whose *age* is greater than 23

    ```
    db.Users.find( {age: {$gt: 23}} )
    ```

  - Find all docs about *Users* who favorite Bob

    ```
    db.Users.find( {favorites: {$in: ["bob_in_se"]}} )
    ```

- However, no cross-document queries (like joins)
  - have to be implemented in the application logic

Document Stores: Use Cases

- Whenever we have items of similar nature but slightly different structure
- Examples:
  - Blogging platforms
  - Content management systems
  - Event logging
- Fast application development

Examples of Document Stores

- Amazon's SimpleDB
- CouchDB
- Couchbase
- MongoDB

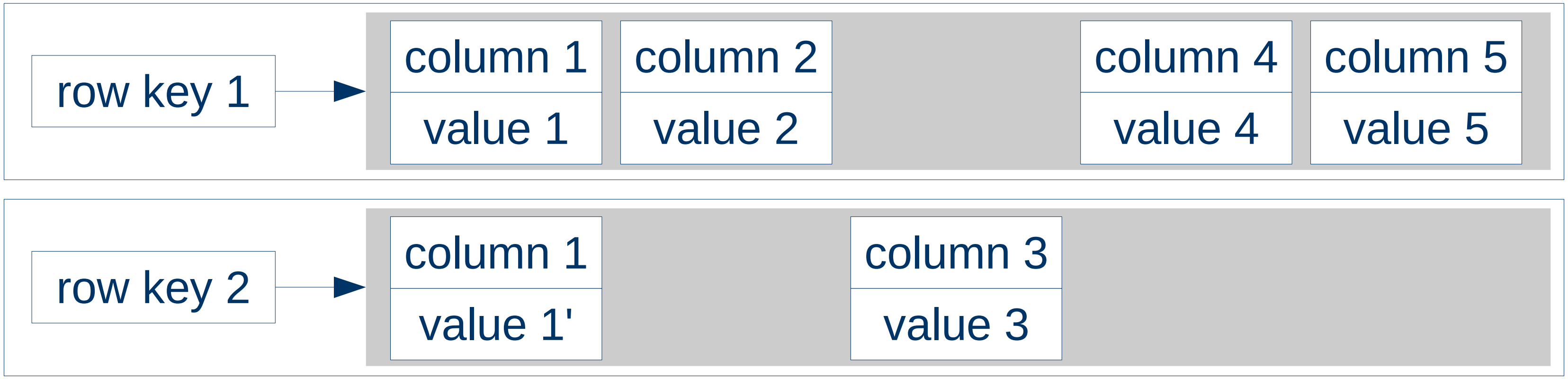
Data Models

- Key-value model
- Document model
- Wide-column models
- Graph database models

also called
column-family models
or
extensible-record models

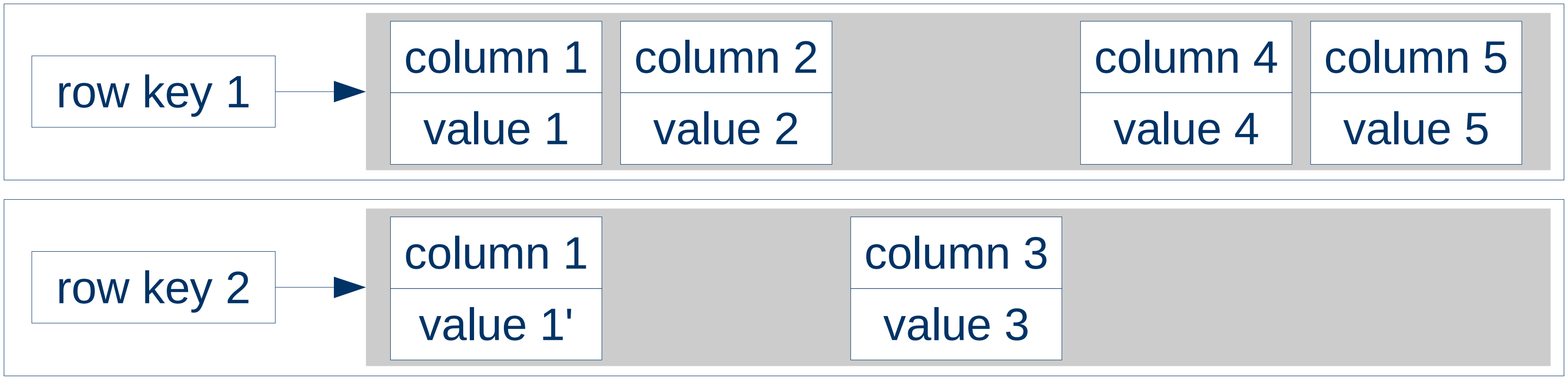
Wide-Column Stores: Data Model (Basic)

- Database is a set of “rows” each of which …
  … has a unique key, and
  … a set of key-value pairs (called “columns”)
- Schema free: different rows may contain different columns



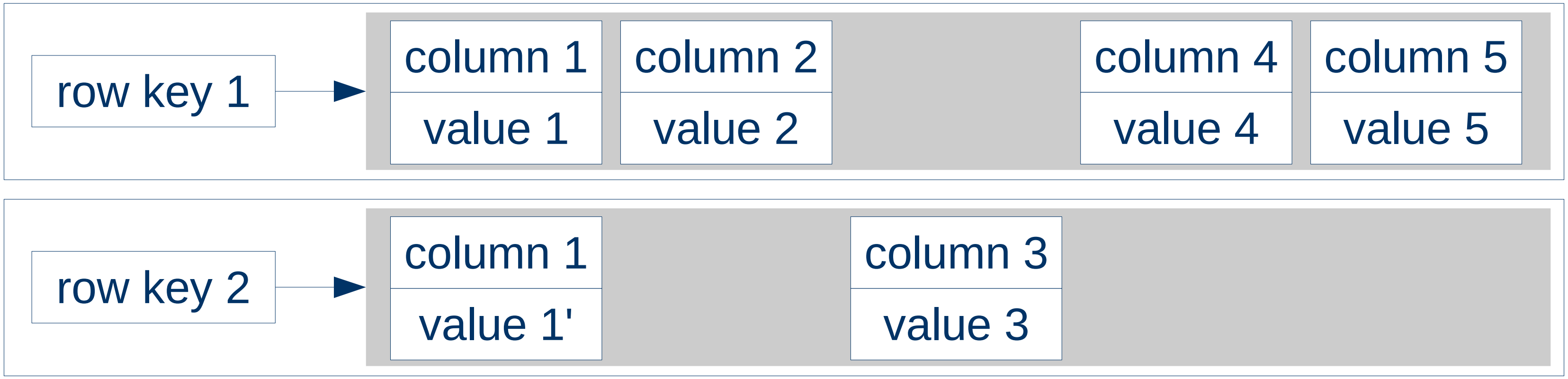
Wide-Column Stores: Data Model

- Database is a set of “rows” each of which …
  … has a unique key, and
  … a set of key-value pairs (called “columns”)
- Schema free: different rows may contain different columns
- Like a single, very wide relation (SQL table) that is
  a) extensible, b) schema-free, and c) potentially sparse



Wide-Column Stores: Data Model

- Database is a set of “rows” each of which …
  … has a unique key, and
  … a set of key-value pairs (called “columns”)
- Schema free: different rows may contain different columns
- Like a single, very wide relation (SQL table) that is
  a) extensible, b) schema-free, and c) potentially sparse
- Like the document model without nesting



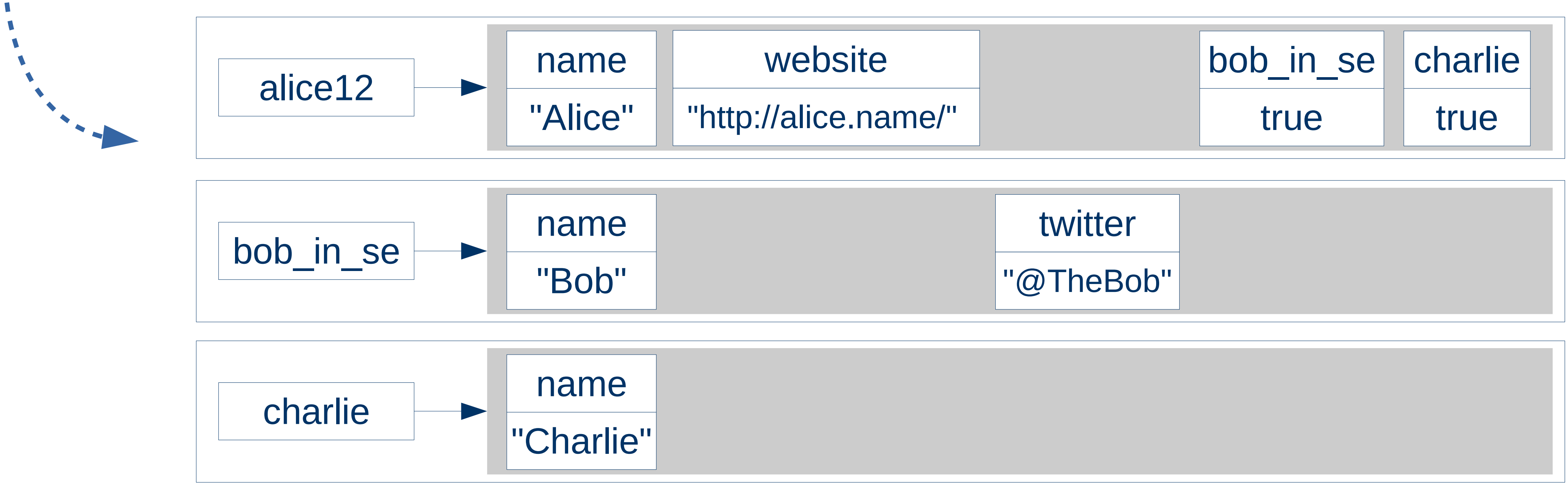
Example (cont'd)

User

login	name	website	twitter
alice12	Alice	http://alice.name/	NULL
bob_in_se	Bob	NULL	@TheBob
charlie	Charlie	NULL	NULL

Fav

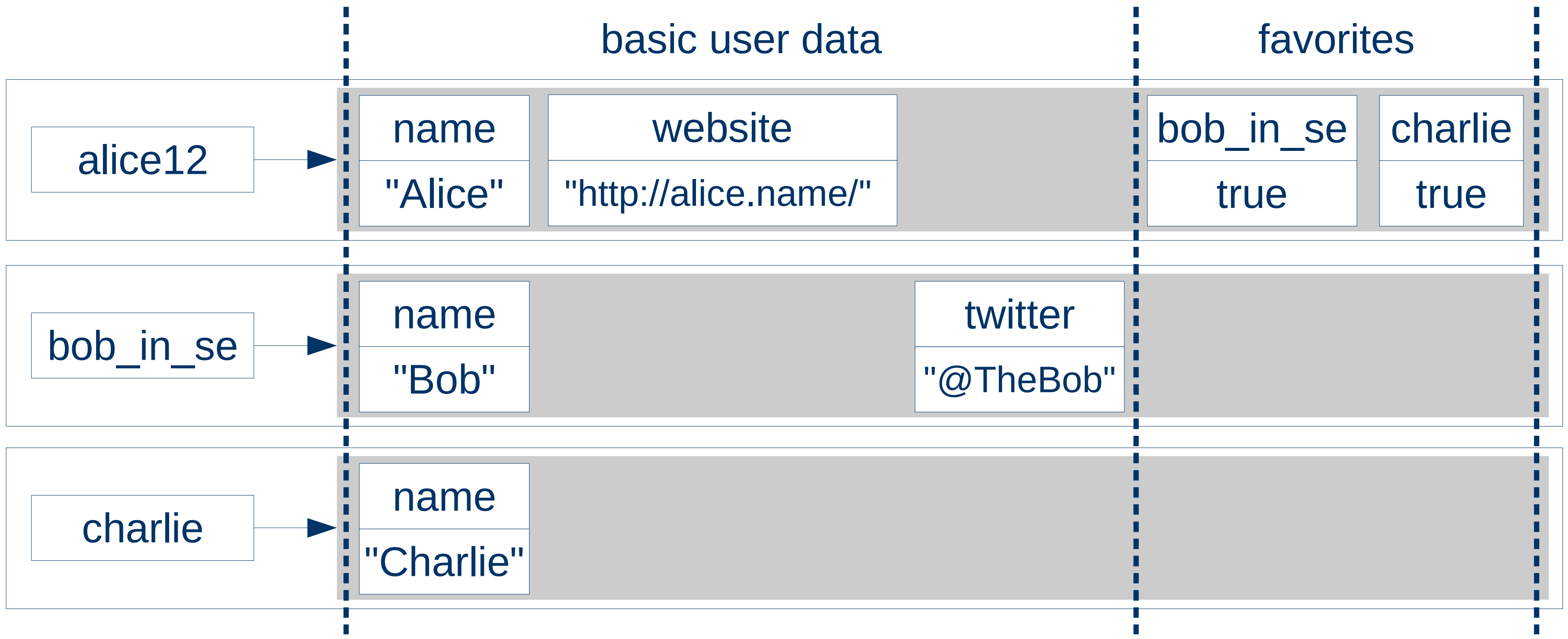
user	favorite
alice12	bob_in_se
alice12	charlie



Wide-Column Stores: Data Model (cont'd)

- Columns may be grouped into so called “column families”
  - Hence, values are addressed by

    (*row key*, *column family*, *column key*)



Wide-Column Stores: Data Model (cont'd)

- Columns may be grouped into so called "column families"
  - Hence, values are addressed by

    (*row key*, *column family*, *column key*)

- Data may be partitioned …

  … based on row keys (*horizontal partitioning*),

  … but also based on column families (*vertical partitioning*),

  … or even on both

Wide-Column Stores: Data Model (cont'd)

- Columns may be grouped into so called “column families”
  - Hence, values are addressed by

    (*row key*, *column family*, *column key*)

- Data may be partitioned …

  … based on row keys (*horizontal partitioning*),

  … but also based on column families (*vertical partitioning*),

  … or even on both

- Secondary indexes can be created over arbitrary columns

Wide-Column Stores: Querying

- Querying in terms of keys or conditions on column values
- Queries expressed in a system-specific query language or in terms of program code using an API
  - Conceptually similar to queries in document stores
- No joins
  - Again, must be implemented in the application logic

Wide-Column Stores: Use Cases

- Similar to use cases for document store
- Analytics scenarios
  - Web analytics
  - Personalized search
  - Inbox search

Examples of Wide-Column Stores

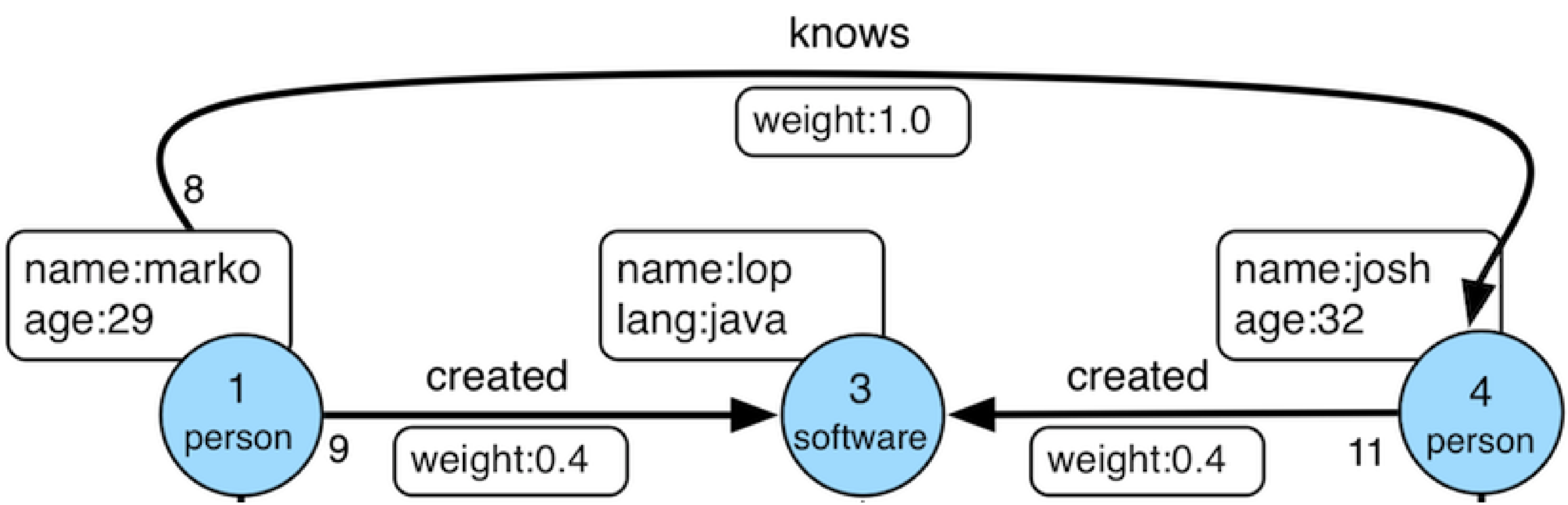
- Basic form (no column families):
  - Amazon SimpleDB
  - Amazon DynamoDB
- With column families:
  - Google's BigTable
  - Hadoop HBase
  - Apache Cassandra

Data Models

- Key-value model
- Document model
- Wide-column models
- Graph database models

Graph Database Systems: Data Model

- Database is some form of a graph (nodes and edges)
  - plus some extra features
- Prominent example: *Property Graphs* in which any node and any edge may additionally have a label as well as key-value pairs (called "properties")



Graph Database Systems: Querying

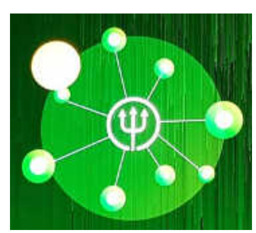
- Graph pattern matching
- Traversal queries
  - e.g., shortest paths, navigational expressions
- Graph algorithms
  - e.g., PageRank, connected components, clustering

Graph Database Systems: Use Cases

- Complex networks
  - e.g., social, information, technological, biological
- Concrete example use cases:
  - Location-based services
  - Recommendation
  - Fraud detection

Examples of (Property) Graph Systems

- Neo4j
- TigerGraph
- InfiniteGraph
- JanusGraph
- Cambridge Semantics' AnzoGraph
- Amazon Neptune



Data Models

- Key-value model
- Document model
- Wide-column models
- Graph database models

There are also *multi-model NoSQL stores*

Examples:

- OrientDB (key-value, documents, graph)
- ArangoDB (key-value, documents, graph)
- Cosmos DB (key-value, documents, wide-column, graph)

Typical* Characteristics of NoSQL Systems

- Ability to scale horizontally over many commodity servers with high performance, availability, and fault tolerance
  - achieved by giving up ACID guarantees
  - and by partitioning and replication of data
- Non-relational data model, no requirements for schemas
  - data model limitations make partitioning effective

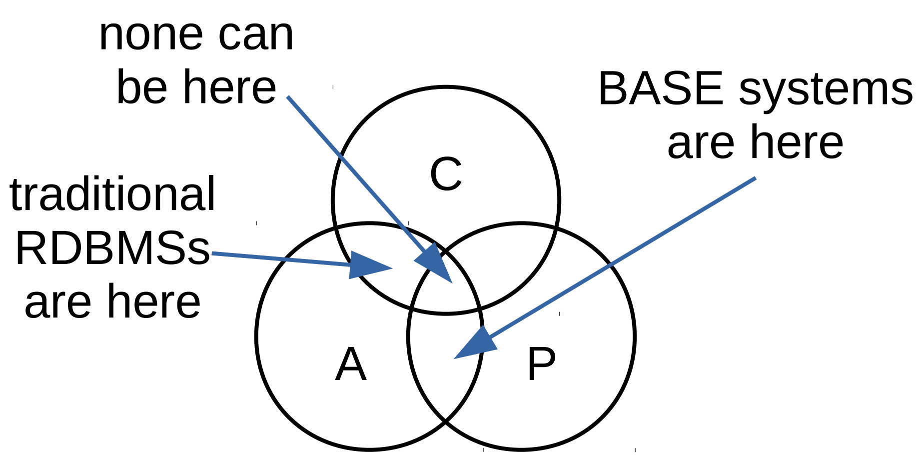
*Attention, there is a *broad variety* of such systems and not all of them have these characteristics to the same degree

BASE rather than ACID

… and the CAP theorem

CAP Theorem

- Only 2 of 3 properties can be guaranteed at the same time in a distributed system with data replication



- ***C**onsistency*: the same copy of a replicated data item is visible from all nodes that have this item
  - Note that this is something else than consistency preservation in ACID
- ***A**vailability*: all requests for a data item get a response
  - response may be that the operation cannot be completed
- ***P**artition Tolerance*: system continues to operate even if it gets partitioned into isolated sets of nodes

What is BASE?

- Idea: by giving up ACID guarantees, one can achieve much higher performance and scalability in a distributed DB system

- ***B***asically ***A***vailable: system available whenever accessed, even if parts of it unavailable
- ***S***oft state: the distributed data does not need to be in a consistent state at all times
- ***E***ventually consistent: state will become consistent after a certain period of time

- BASE properties suitable for applications for which some inconsistency may be acceptable

Consistency Models

- *Weak consistency:* no guarantee that all subsequent accesses will return the updated value
  - *eventual consistency*: if no new updates are made, eventually all accesses will return the last updated value
  - *inconsistency window*: the period until all replicas have been updated in a lazy manner
- *Strong consistency*: after an update completes, every subsequent access will return the updated value
  - may be achieved without consistency in the CAP theorem

Consistency Models (cont'd)

- Let:
  - N be the number of nodes that store replicas of a data item
  - R be the number of nodes required for a successful read
  - W be the number of nodes required for a successful write
- Then:
  - Consistency as per CAP requires $W = N$
  - Write consistency requires $W > ½ N$
  - Strong consistency (for reads) requires $R + W > N$
  - High read performance means a great N and $R = 1$
  - Fault tolerance/availability (and relaxed consistency) $W = 1$

Experiences with BASE NoSQL Systems

- Lack of a powerful declarative database language has to be handled in application code (likely less efficient)
  - NoSQL vendors have started to support SQL or created their own, SQL-like languages (e.g., CQL, UnQL)
- Giving up strong transactional and consistency requirements is not an option for many applications
  - "cannot sell the last item twice"
- Even for applications where it seemed so, inconsistencies have to be handled in application code
- Huge burden for application developers
  - e.g., Google found that NoSQL systems caused their developers to spend too much time writing code to handle inconsistent data and that using transactions makes them more productive

NewSQL

Characteristics of NewSQL Systems

- SQL and ACID guarantees for transactional read-write workloads (OLTP), just like traditional RDBMSs
- Performance and scalability as NoSQL systems, through innovative software architecture

- Modern RDBMSs designed either to meet scalability requirements of distributed architectures -or- to improve performance so horizontal scalability is no longer a necessity

Categorization

- New architectures
  - New systems built from scratch to operate on shared-nothing resources, with components to support multi-node concurrency control, fault tolerance through replication, flow control, and distributed query processing
  - VoltDB, CockroachDB, MariaDB Xpand (formerly Clustrix), SingleStore (formerly MemSQL), SAP HANA, NuoDB
- Transparent sharding middleware
  - Centralized component that routes queries, coordinates transactions, and manages data placement, replication, and partitioning across a cluster of single-node DBMS instances
  - MariaDB MaxScale, ScaleArc
- Database as a service (DBaaS)
  - Amazon Aurora, ClearDB, Google Cloud Spanner

www.liu.se

LINKÖPING UNIVERSITY