TDDD43 Advanced Data Models and Databases

NoSQL Databases

Huanyu Li
huanyu.li@liu.se

Based on slides by Olaf Hartig

LINKÖPING
UNIVERSITY

NoSQL in general

- "NoSQL" is interpreted differently (without precise definition)
  - "no to SQL"
  - "not only SQL"
  - "not relational"

- Non-relational databases has been around since the late 1960s
- 1998: first used for a RDBMS without SQL interface
- 2009: picked up again to name the conference "NOSQL 2009" about "open-source, distributed, non-relational databases"

- Since then, "NoSQL database" loosely specifies class of non-relational DBMSs

Focuses of this lecture

- What are key characteristics of NoSQL systems?
- How do DBs supported by NoSQL systems look like?
- What can you do with these DBs?
    - (in comparison to the DBs supported by RDBMSs)

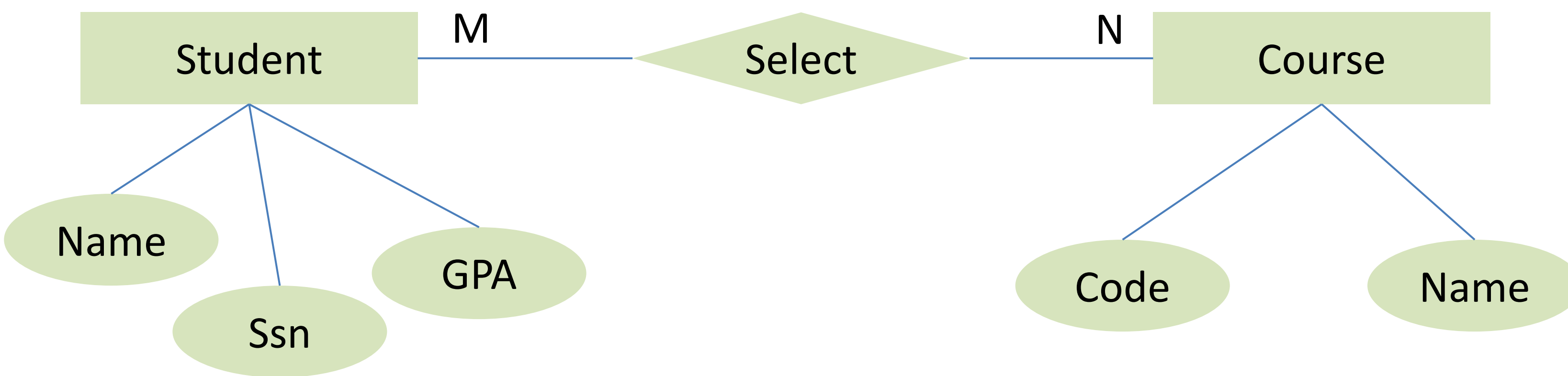
Outline

- Why NoSQL comes to the stage
- NoSQL data models
- The performance and scability of NoSQL
- NoSQL consistency models and basic techniques

Recap of Relational Database Management Systems

- With well-defined formal foundations
  - Schema level (Entity-Relationship)
  - Relational Model (relation/table, attribute/column, tuple/row)

Recap of Relational Database Management Systems



Relation/Table

COURSE

Attributes/Columns

Tuples/Rows

Code	Name
TDDD43	Advanced Data Models and Databases
TDDE31	Big Data Analytics

Recap of Relational Database Management Systems

- With well-defined formal foundations
- SQL – structured query language
  - query, data manipulation, database definition

- Support of transactions with ACID properties
  - ***A***tomicity, ***C***onsistency, ***I***solation, ***D***urability

- Established technology
  - Many vendors
  - Highly mature systems
  - Experienced users and administrators

But, the business world has changed…

- More organizations and companies have shifted to the digital economy powered by the Internet

- New IT applications that allow companies to run their business and to interact with costumers, are required and prioritized
  - Web and Mobile applications
  - Connected devices (“Internet of Things”)

- As a result, new challenges come…

Therefore, the scalability of a system is important

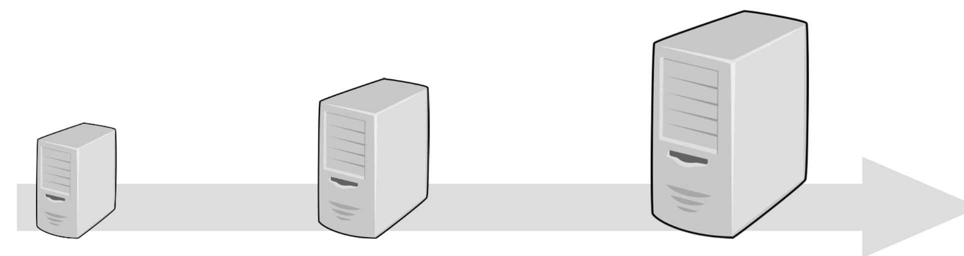
- Data scalability
  - Handle growing amounts of data,
    without losing performance

- Read scalability
  - Handle increasing numbers of read operations,
    without losing performance

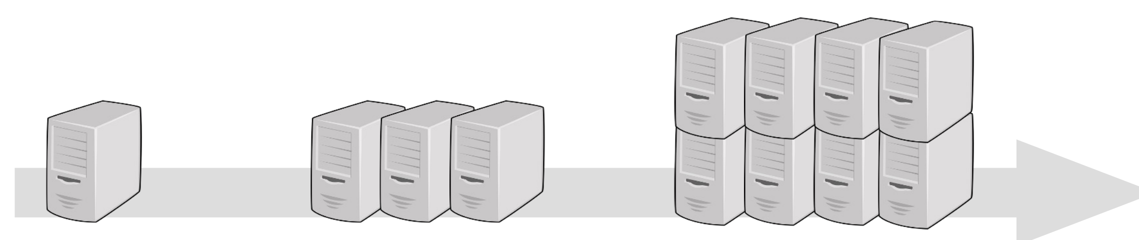
- Write scalability
  - Handle increasing numbers of write operations,
    without losing performance

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



To achieve much higher performance and scalability

- NoSQL supports (some inconsistency may be acceptable):
  - Basically Available
    - System available whenever accessed, even if parts of it unavailable
  - Soft state
    - the distributed data does not need to be in a consistent state at all times
  - Eventually consistent
    - state will become consistent after a certain period of time

We will get back to transactional access of NoSQL systems later again!

Typical Characteristics of NoSQL systems

- Ability to scale horizontally over many commodity servers with high performance, availability and fault tolerance
  - achieved by guaranteeing basically available, soft state, eventually consistent
  - and by partitioning and replication of data
- Non-relational data model, no requirements for schemas

- “Typical” means there is a broad variety of such systems, but not all of them have these characteristics to the same degree

Outline

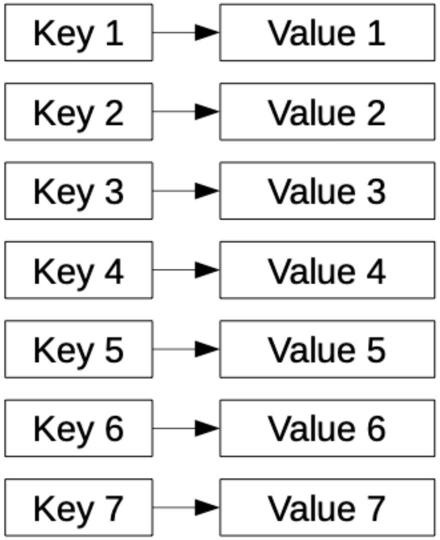
- ✓ Why NoSQL comes to the stage
- ➢ NoSQL data models
- ➢ BASE, ACID, and CAP
- ➢ NoSQL consistency models and basic techniques

Data Models for NoSQL

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

NoSQL Data Models – Key value stores

- Simplest form of NoSQL databases
- Schema-free, a dictionary of key/value pairs
  - Keys are unique
  - Values are of arbitrary types
- Efficient in storying distributed data
- Not suitable for
  - Representing structures and relations
  - Accessing multiple items, since the access is by key



Example - Key value stores

- Assuming a relational database consisting of a table:

User

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

- How to represent such data using the key-value model?

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

Example - Key value stores

- Let's add another table

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

- How to represent such data using 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
  - 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

- However, partition the data based on keys (horizontal partitioning or sharding) and distributed processing can be very efficient

Example - Key value stores

- Assume we try to find all users for whom Bob is a favorite
- It is possible, but very inefficient

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

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

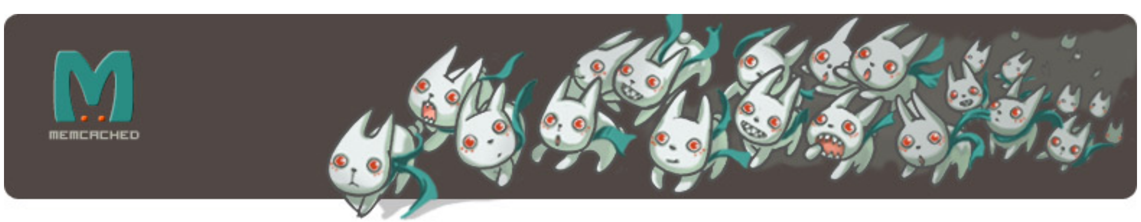
Example - Key value stores

- Assume we try to find all users for whom Bob is a favorite
- It is possible, 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]

Examples of Key value stores

- Open-source examples
  - Redis
  - Memcached



Data Models for NoSQL

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

NoSQL Data Models – Document stores

- Store data as documents
- A dictionary of key/value pairs
  - Keys are unique
  - Values are documents, semi-structured data
    - XML, JSON, BSON (binary), etc.
- Efficient in storying distributed data
- Not suitable for
  - Representing structures and relations
  - Accessing multiple items, since the access is by key

NoSQL Data Models – Document stores

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

Fav

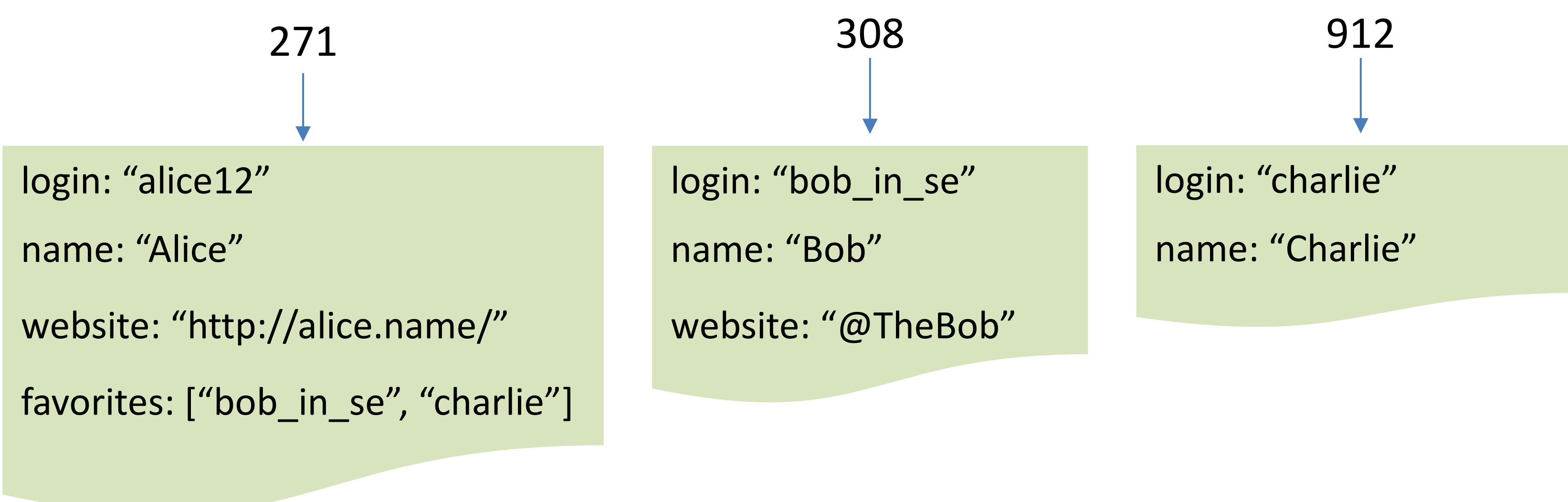
user	favorite
alice12	bob_in_se
alice12	charlie

User

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

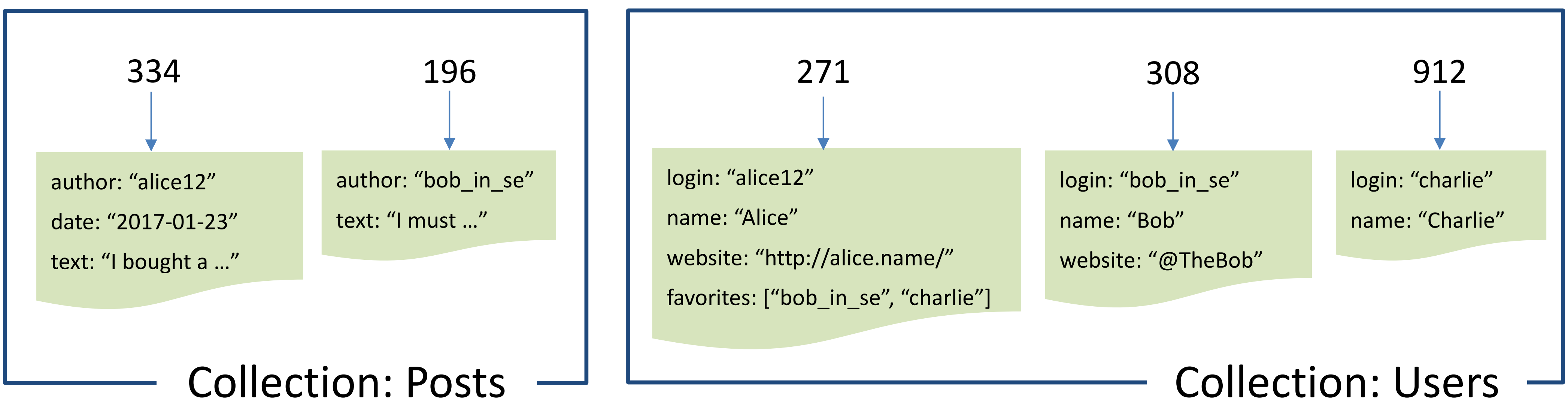
NoSQL Data Models – Document stores

- Document Store based Database
  - A set of documents (or multiple such sets)
  - Each document additionally associated with a unique identifier
  - Schema free: different documents may have different fields



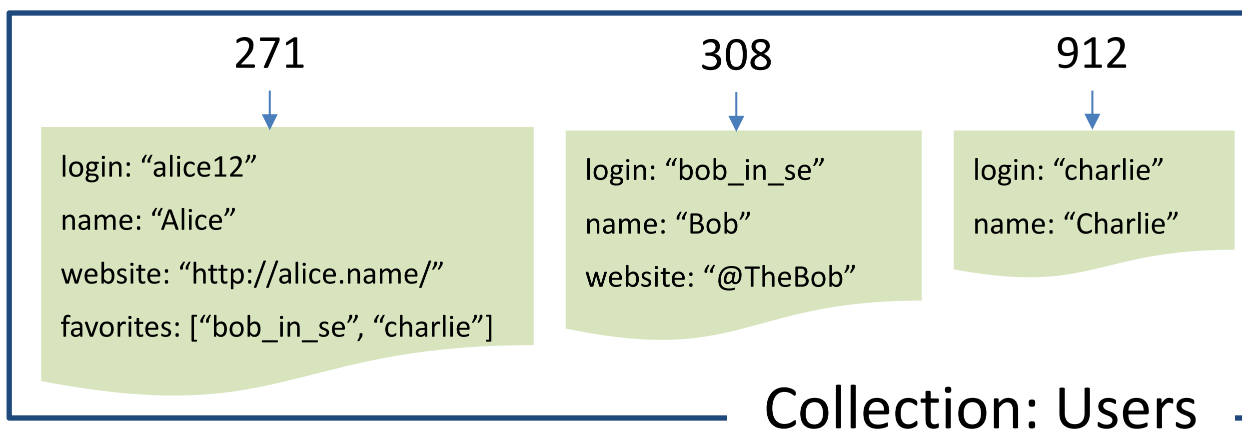
NoSQL Data Models – Document stores

- Document Store based Database
  - A set of documents (or multiple such sets)
  - Each document additionally associated with a unique identifier
  - Schema free: different documents may have different fields
  - Grouping of documents into separate sets (called “domains” or “collections”)



NoSQL Data Models – Document stores

- Document Store based Database
  - A set of documents (or multiple such sets)
  - Each document additionally associated with a unique identifier
  - 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
- Depending on specific systems, queries may be expressed:
  - program code using an API
  - in a system-specific query language

Document stores: Querying

- Querying in terms of conditions on document content
- Depending on specific systems, queries may be expressed:
  - program code using an API
  - in a system-specific query language
- Examples (based on 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 in collection *Users* whose 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
- Depending on specific systems, queries may be expressed:
  - program code using an API
  - in a system-specific query language
- Examples (based on 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:{$qt: 23}})
  - Find all docs in collection *Users* whose favorite Bob

    db.Users.find({favorites: {$in: ["bob_in_se"]}})
- However, no cross-document queries (e.g., joins)
  - have to be implemented in the application logic

NoSQL Data Models – Document store

- Examples
  - MongoDB

Data Models for NoSQL

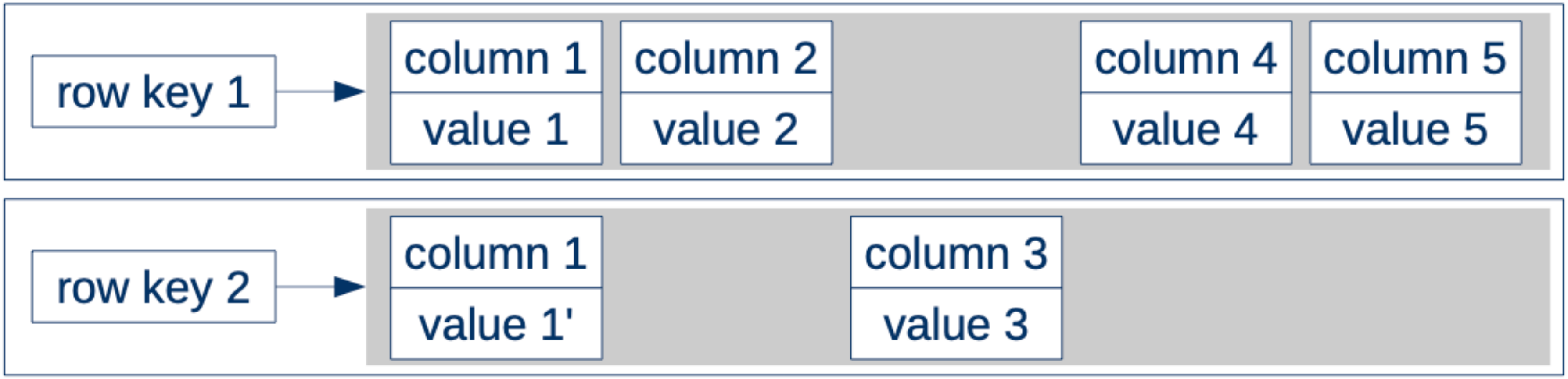
- ✓ Key-value model
- ✓ Document model
- ➢ Wide-column models
- ➢ Graph database models

NoSQL Data Models – Wide column stores

- Also called column-family or extensible-record
- Store data in rows, each row has a unique key and column families
- Schema-free
  - Keys are unique
  - Values are varying column families
    - Columns consist of key value pairs

NoSQL Data Models – Wide column store

- Like a single, very wide relation (SQL table) but extensible, schema-free and potentially sparse
- Like the document model without nesting



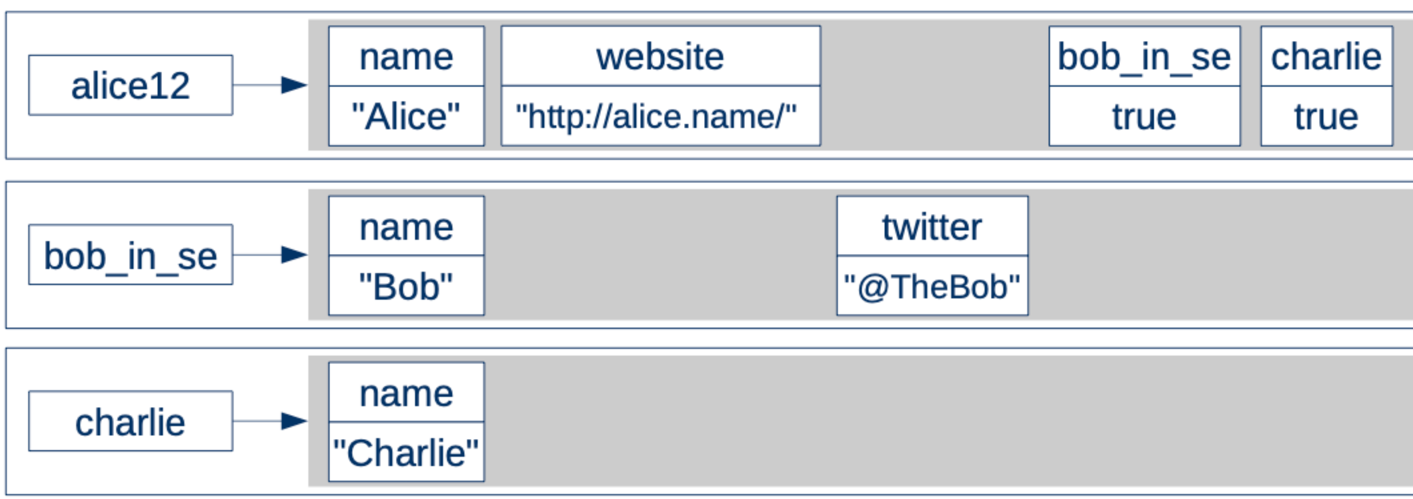
NoSQL Data Models – Wide column store

Fav

user	favorite
alice12	bob_in_se
alice12	charlie

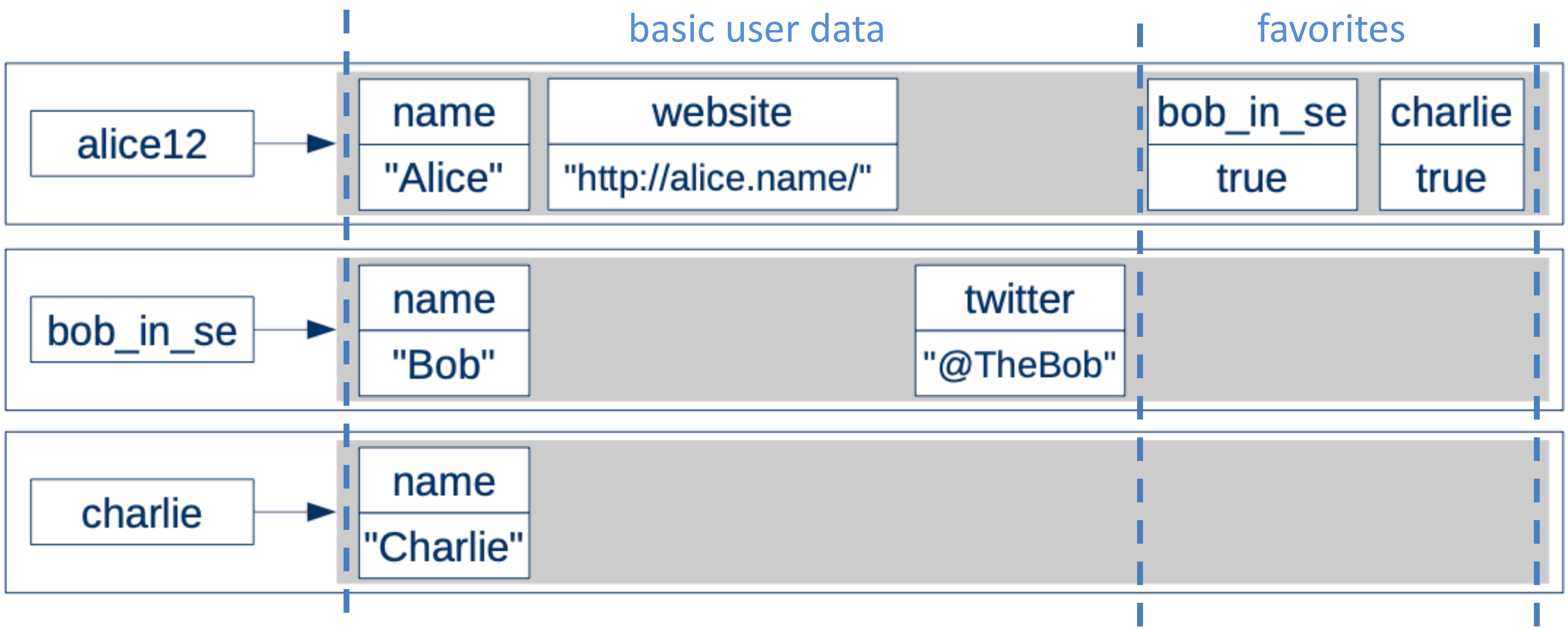
User

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



NoSQL Data Models – Wide column store

- Columns may be grouped into "column families"
  - Therefore, values are addressed by row key, column family, and column key



NoSQL Data Models – Wide column store

- Columns may be grouped into “column families”
  - Therefore, values are addressed by row key, column family, and 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
- Conceptually similar to queries in document stores
  - program code using an API
  - in a system-specific query language
  - Again, no joins, have to be implemented in the application logic
- Better than key value stores for querying and indexing
- Not suitable for
  - Representing structures and relations

NoSQL Data Models – Wide Column store

- Examples
  - Google BigTable
  - Apache HBase
  - Apache Cassandra

Data Models for NoSQL

- ✓ Key-value model
- ✓ Document model
- ✓ Wide-column models
- ➢ Graph database models
  - Next lecture tomorrow

Data Models for NoSQL

- ✓ Key-value model
- ✓ Document model
- ✓ Wide-column models
- ➢ **Graph database models**

➢ There are NoSQL systems that are based on multi models

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

Outline

- ✓ Why NoSQL comes to the stage
- ✓ NoSQL data models
- ➢ NoSQL consistency models and basic techniques
  - ➢ ACID, BASE and CAP
  - ➢ Consistent Hashing
  - ➢ Vector Lock

Typical Characteristics of NoSQL systems

- Ability to scale horizontally over many commodity servers with high performance, availability and fault tolerance
  - achieved by guaranteeing basically available, soft state, eventually consistent
    - in another word, by giving up ACID guarantees
  - and by partitioning and replication od data
- Non-relational data model, no requirements for schemas

- "Typical" means there is a broad variety of such systems, but not all of them have these characteristics to the same degree

ACID

- Atomicity
  - The entire transaction takes place at once or doesn't happen at all
- Consistency
  - The database must be consistent before and after the transaction
- Isolation
  - Multiple transactions occur independently without interference
- Durability
  - The changes of a successful transaction occurs even if the system failure occurs

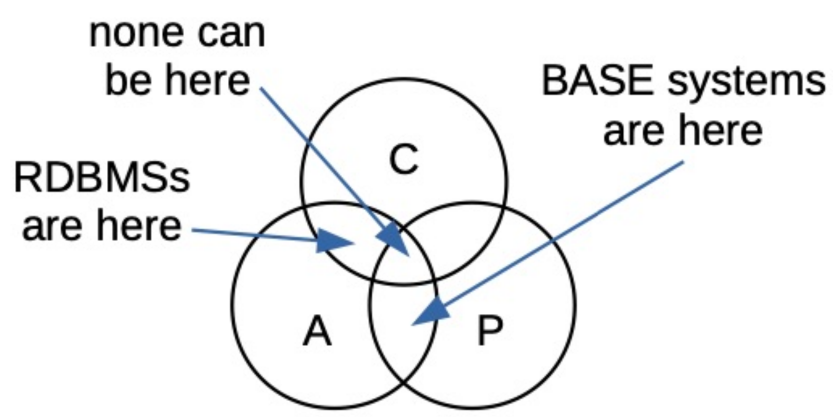
NoSQL, BASE rather than ACID

- Giving up ACID guarantees, to achieve much higher performance and scalability
- ***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

CAP Theorem for distributed data store

- ***C***onsistency
  - After an update, all readers in a distributed system see the same data
  - All nodes are supposed to contain the same data at all times
- ***A***vailability
  - All requests will be answered, regardless of crashes or downtimes
- ***P***artition Tolerance
  - System continues to operate, even if two sets of servers get isolated

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



Outline

- ✓ Why NoSQL comes to the stage
- ✓ NoSQL data models
- ➢ NoSQL consistency models and basic techniques

Consistency models

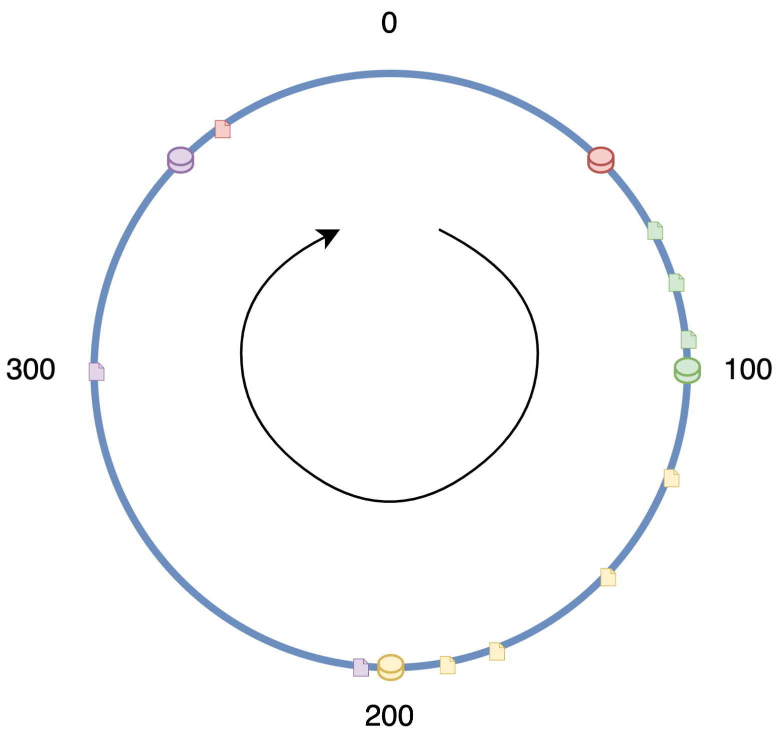
- Strong consistency
  - After the update completes. Any subsequent access will return the updated value
- Weak consistency
  - The system does not guarantee that subsequent accesses to the system will return the updated value
  - Inconsistency window: the period until all replicas have been updated in a lazy manner
  - Eventual consistency: if no new updates are made, eventually all accesses will return the last updated value
    - Employed by many NoSQL databases

NoSQL Techniques

- Basic techniques (widely applied in NoSQL systems)
  - Distributed data storage, replication (Consistent hashing)
  - Recognize order of distributed events and potential conflicts (Vector clock)
  - Distributed query strategy (MapReduce)

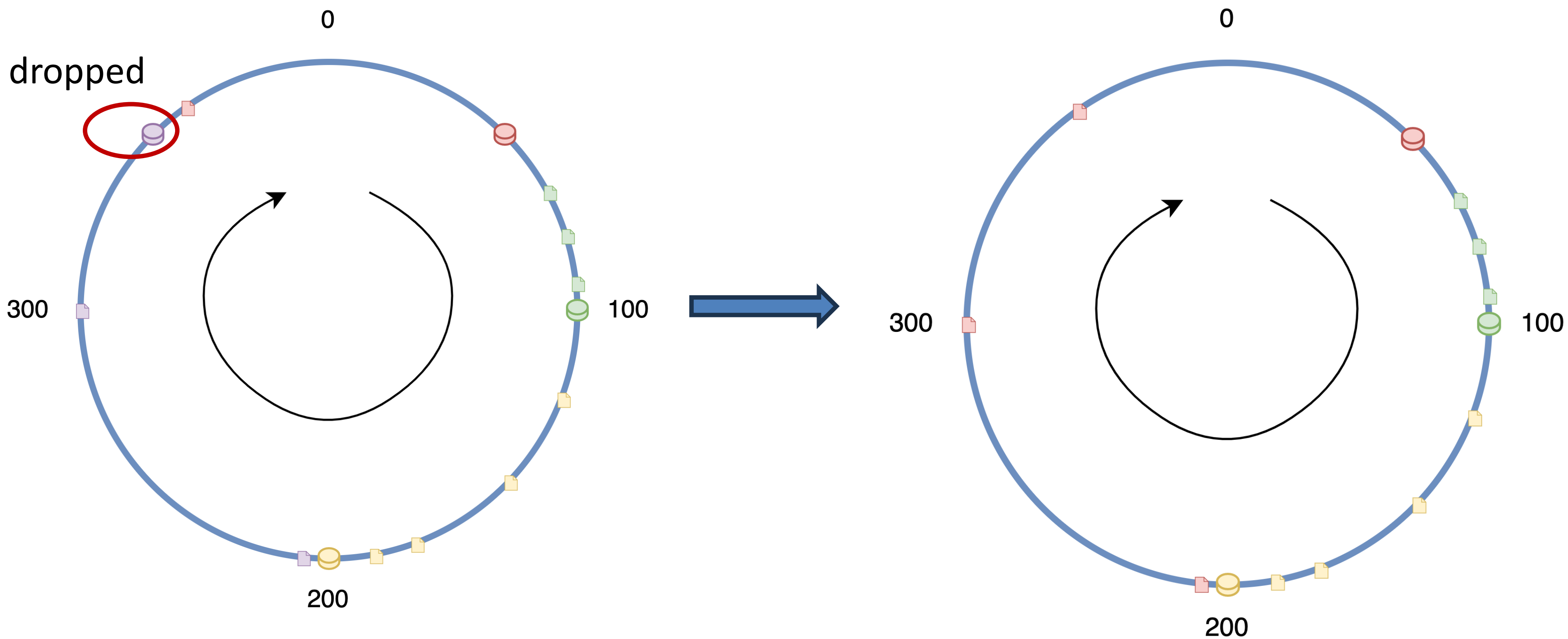
NoSQL Consistent hashing

- A virtual ring structure (hash ring)
- Use the same hashing function to hash both the node (server) identifiers (IP addresses) and data keys
- The ring is traversed in the clockwise direction
- Each node is responsible for the region of the ring between the node and its predecessor on the ring



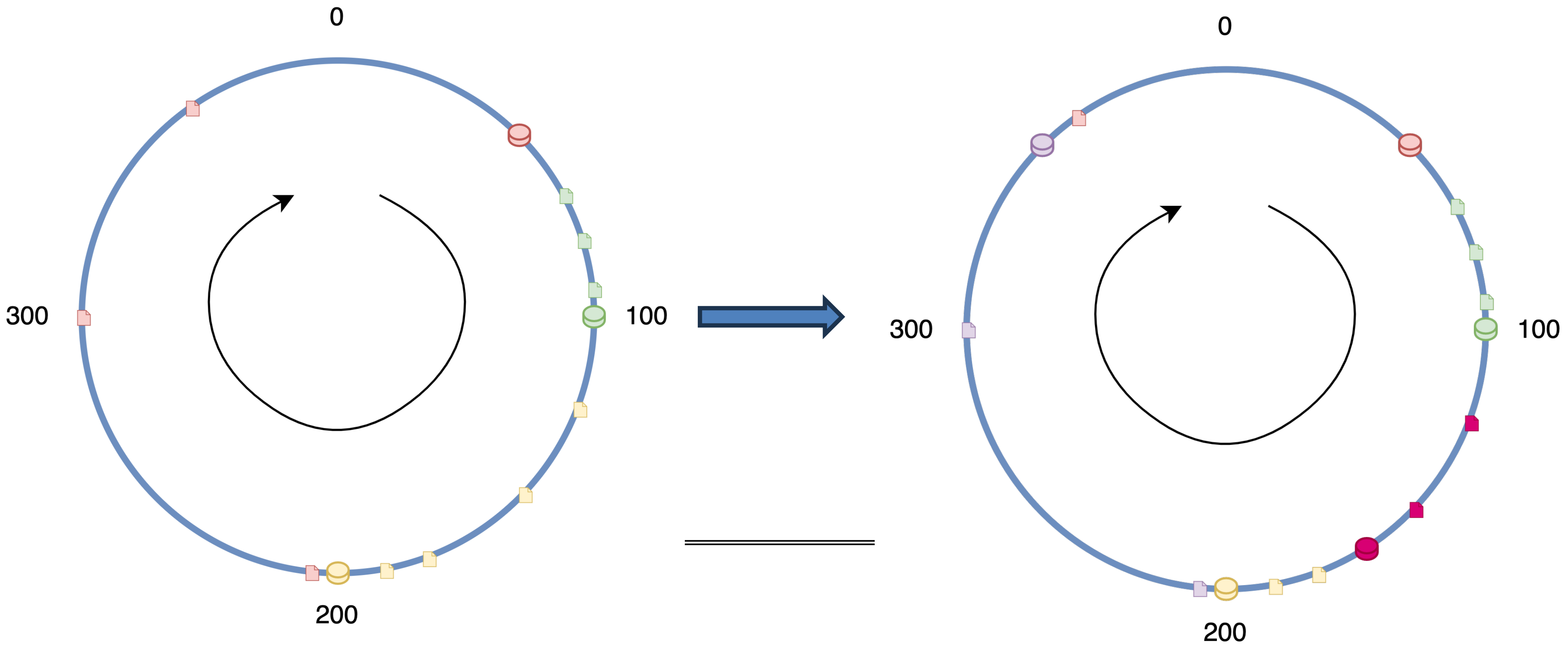
Consistent hashing – Node Removal

- If a node is dropped out or gets lost
  - Its responsible data will be redistributed to an adjacent node



Consistent hashing – Node Addition

- If a node is added
  - Its hash value is added to the hash table
  - the hash realm is repartitioned, and hash data will be transferred to new neighbor
  - No need to update remaining nodes



Vector clock

- MVCC (Multi-version concurrency control)
- Commonly used in DBMS

- Vector clock is an extension of MVCC
  - A vector clock is an array/vector of N logical clocks (N is the number of processes)
  - Each process has a vector clock
  - When processes communicate to each other, vector timestamps are piggybacked

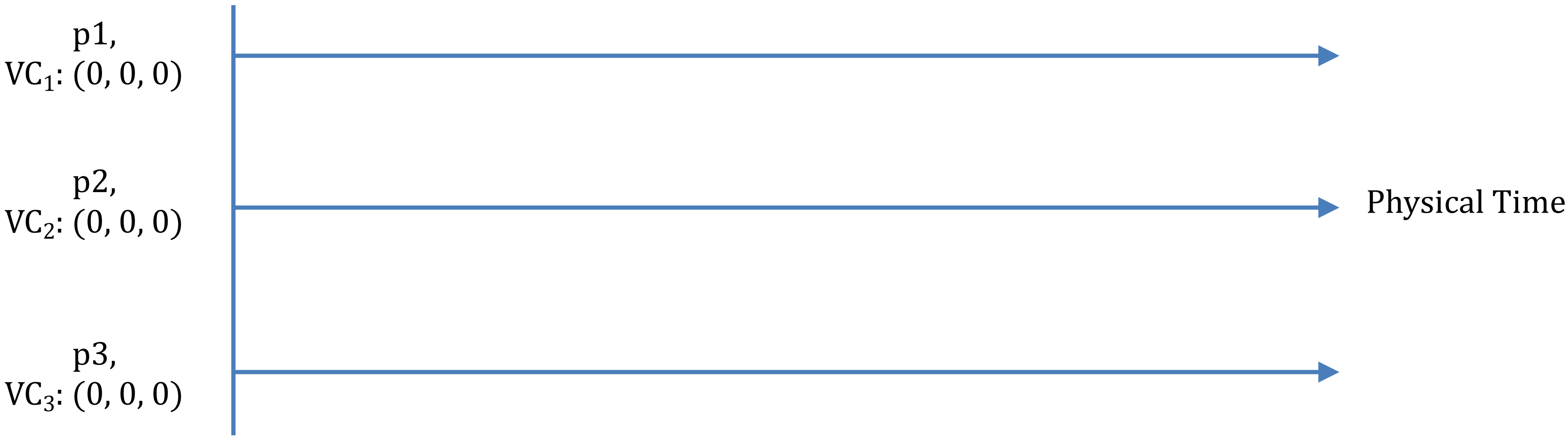
- How are vector clocks maintained?

Vector clock Maintainance

- Let VC_i denote the vector clock for process i (p_i):
- Initialize all clocks for all processes as zero
  - $VC_i[j] = 0$; for i, j $= 1, 2, \ldots N$
- Just before a process (p_i) timestamps an internal event (e.g., sending messages), its own logical clock in its vector will be incremented by one
  - $VC_i[i] = VC_i[i] + 1$
- The new vector of p_i is piggybacked when p_i send messages to other processes
- When a process (p_j) receives a message from another process (p_i), it first increments its own logical clock by one, then compare its vector with the vector it receives and take all the maximum values for each logical clocks
  - $VC_j[i] = VC_j[i] + 1$
  - $VC_j[k] = \max(VC_j[k], VC_i[k])$; for $k = 1,2, \ldots N$

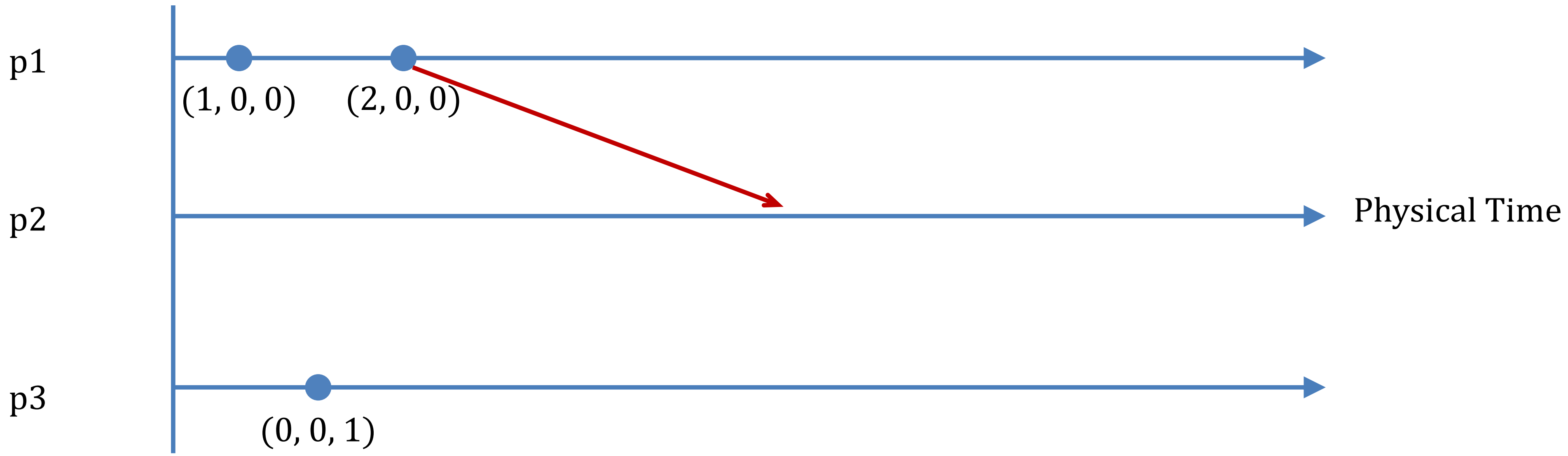
Vector clock Maintainance

- Let VC_i denote the vector clock for process i (p_i):
- Initialize all clocks for all processes as zero
  - $VC_i[j] = 0$; for i, j = 1, 2, … N



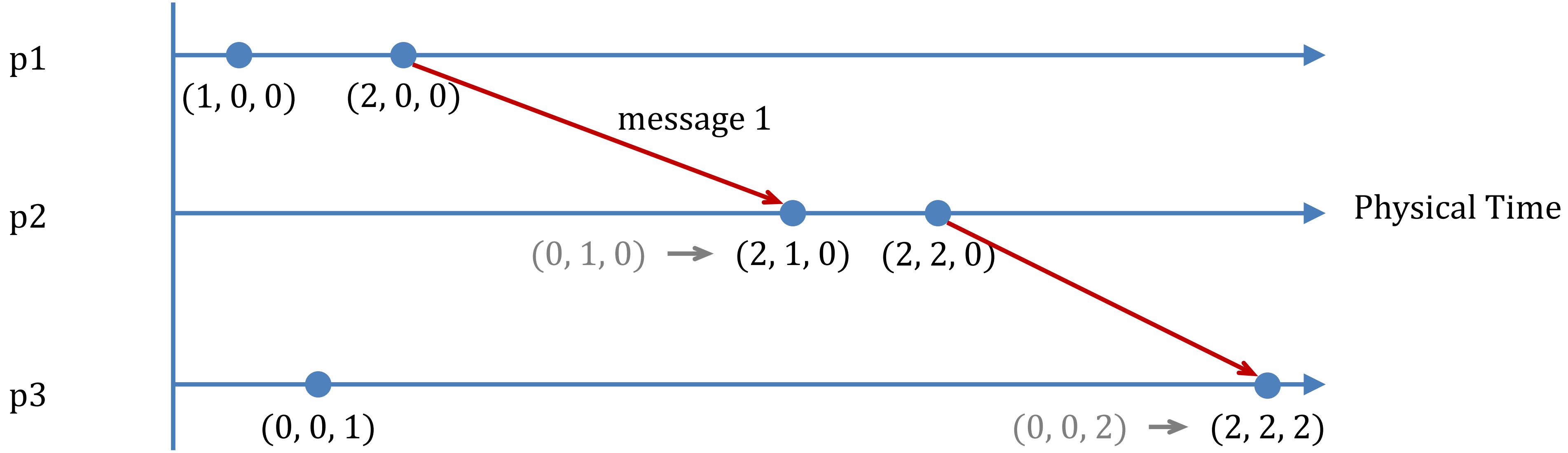
Vector clock Maintainance

- Just before a process (p_i) timestamps an internal event (e.g., sending messages), its own logical clock in its vector will be incremented by one
  - $VC_i[i] = VC_i[i] + 1$



Vector clock Maintainance

- The new vector of p_i is piggybacked when p_i send messages to other processes
- When a process (p_j) receives a message from another process (p_j), it first increments its own logical clock by one, then compare its vector with the vector it receives and take all the maximum values for each logical clocks
  - $VC_j[i] = VC_j[i] + 1$
  - $VC_j[k] = \max(VC_j[k], VC_i[k])$; for $k = 1,2, \ldots N$

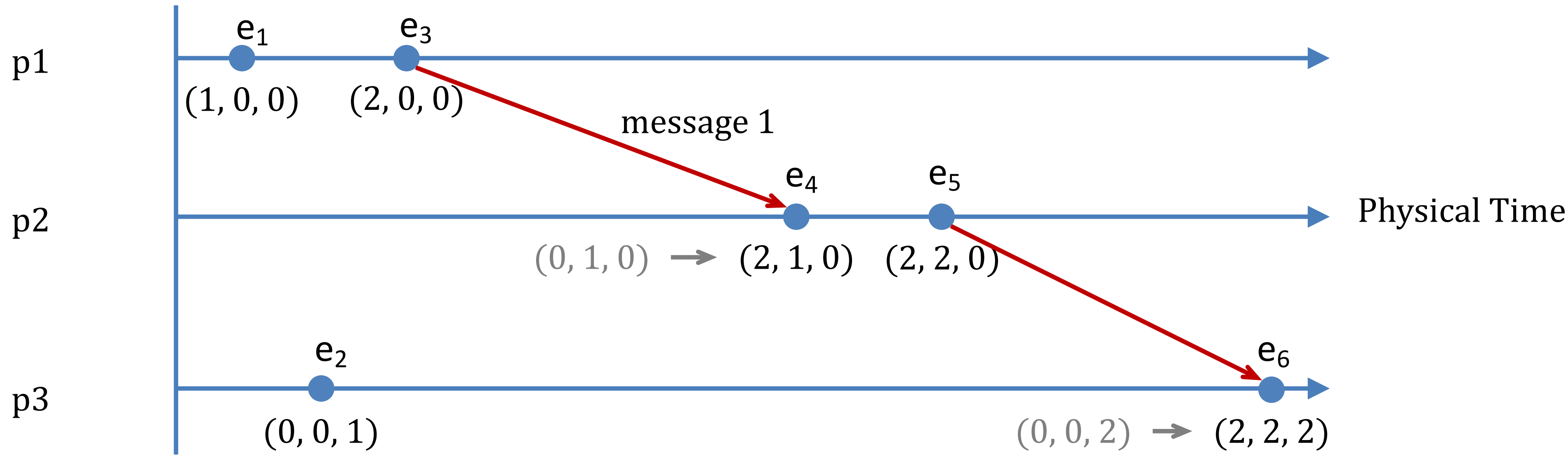


Vector clock Maintainance

- Properties (comparing vector clocks for two events):
  - VC = VC' iff. VC[i] = VC'[i]; for i = 1, 2, … N
  - VC ≤ VC' iff. VC[i] ≤ VC'[i]; for i = 1, 2, … N
  - VC < VC' iff. VC[i] ≤ VC'[i]; for i = 1, 2, … N, meanwhile there exists a process p_j that, VC[j] < VC'[j]
- For two events e and e', e→e' iff. VC(e) < VC(e')
  - event e happens before event e'
- How to detect if two events are conflict to each other?
  - For two events e and e', if neither VC(e) ≤ VC(e') nor VC(e') ≤ VC(e) satisfies, then the two events are concurrent

Vector clock Maintainance

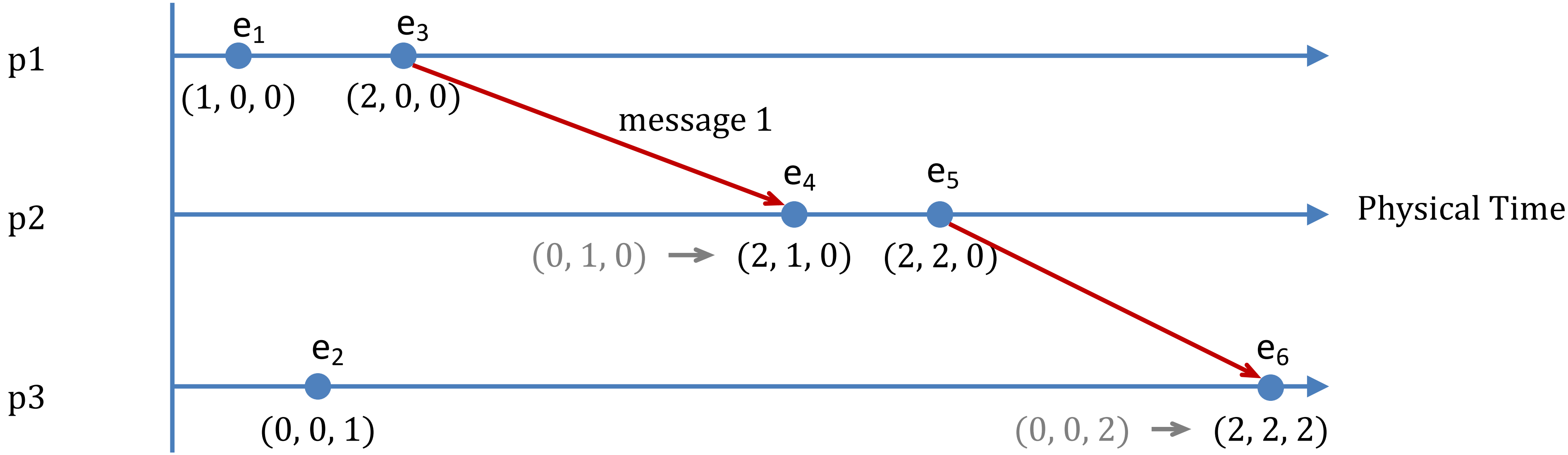
- How to detect if two events are conflict to each other?
  - For two events e and e', if neither $VC(e) \leq VC(e')$ nor $VC(e) \geq VC(e')$ satisfies, then the two events are concurrent



Event	Event	Conflict?
e_1: (1, 0, 0)	e_2: (0, 0, 1)	
e_2: (0, 0, 1)	e_4: (2, 1, 0)	
e_1: (1, 0, 0)	e_4: (2, 1, 0)	
e_4: (2, 1, 0)	e_6: (2, 2, 2)	

Vector clock Maintainance

- How to detect if two events are conflict to each other?
  - For two events e and e', if neither VC(e) ≤ VC(e') nor VC(e) ≥ VC(e') satisfies, then the two events are concurrent



Event	Event	Conflict?
e_1: (1, 0, 0)	e_2: (0, 0, 1)	concurrent
e_2: (0, 0, 1)	e_4: (2, 1, 0)	concurrent
e_1: (1, 0, 0)	e_4: (2, 1, 0)	
e_4: (2, 1, 0)	e_6: (2, 2, 2)	

Summary

- NoSQL systems support non-relational data models
  - Schema free
  - Support for semi-structured and unstructured data
  - Limited query capabilities (no joins!)

- NoSQL systems provide high (horizontal) scalability with high performance, availablility, and fault tolerance
  - Achieved by:
    - Data partitioning
    - Data replication
    - Giving up consistency requirements

www.liu.se

LINKÖPING UNIVERSITY