CompSci 516
Data Intensive Computing Systems

Lecture 21 (b)
NoSQL
(and Column Store)

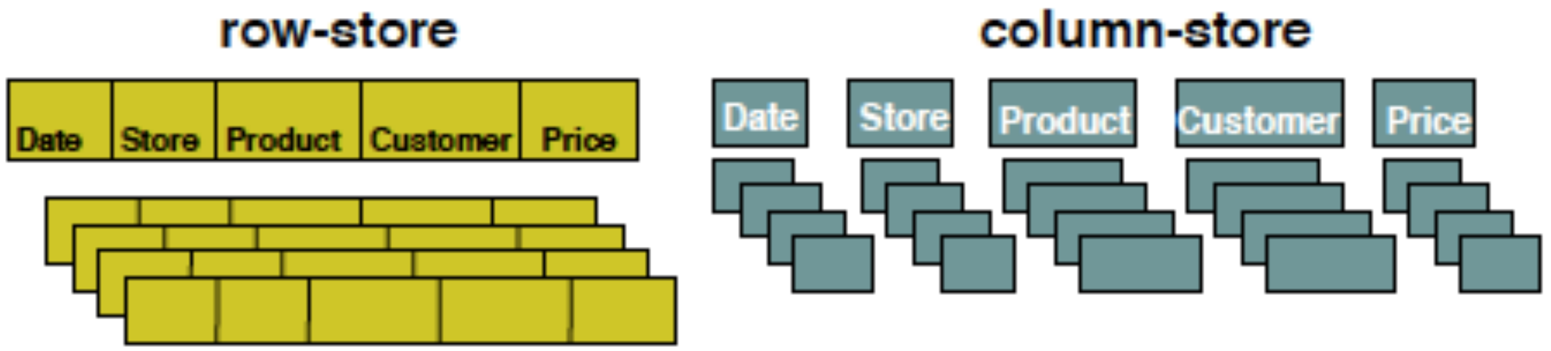
Instructor: Sudeepa Roy

Column Store

Row vs. Column Store

- Row store
  - store all attributes of a tuple together
  - storage like "row-major order" in a matrix
- Column store
  - store all rows for an attribute (column) together
  - storage like "column-major order" in a matrix
- e.g.
  - MonetDB, Verica (earlier, C-store), SAP/Sybase IQ, Google Bigtable (with column groups)
- Optional reading:
  - VLDB 2009 tutorial (link on course webpage)
  - only a few slides are taken from that tutorial in this lecture

What is a column-store?



row-store	column-store
+ easy to add/modify a record	+ only need to read in relevant data
- might read in unnecessary data	- tuple writes require multiple accesses

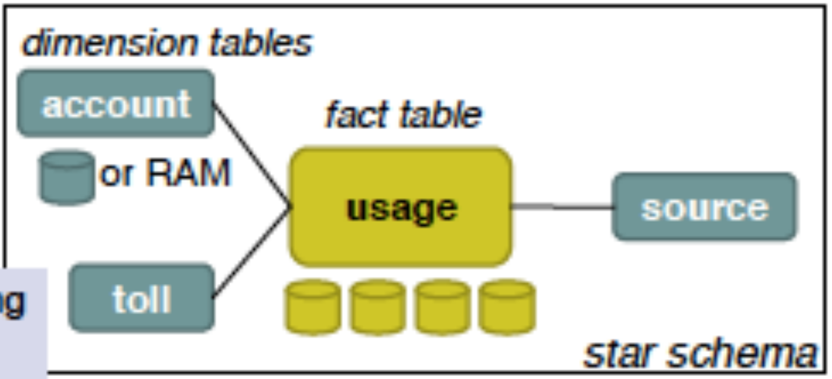
=> suitable for read-mostly, read-intensive, large data repositories

Ack: Slide from VLDB 2009 tutorial on Column store

Telco Data Warehousing example

- Typical DW installation
- Real-world example

"One Size Fits All? - Part 2: Benchmarking Results" Stonebraker et al. CIDR 2007

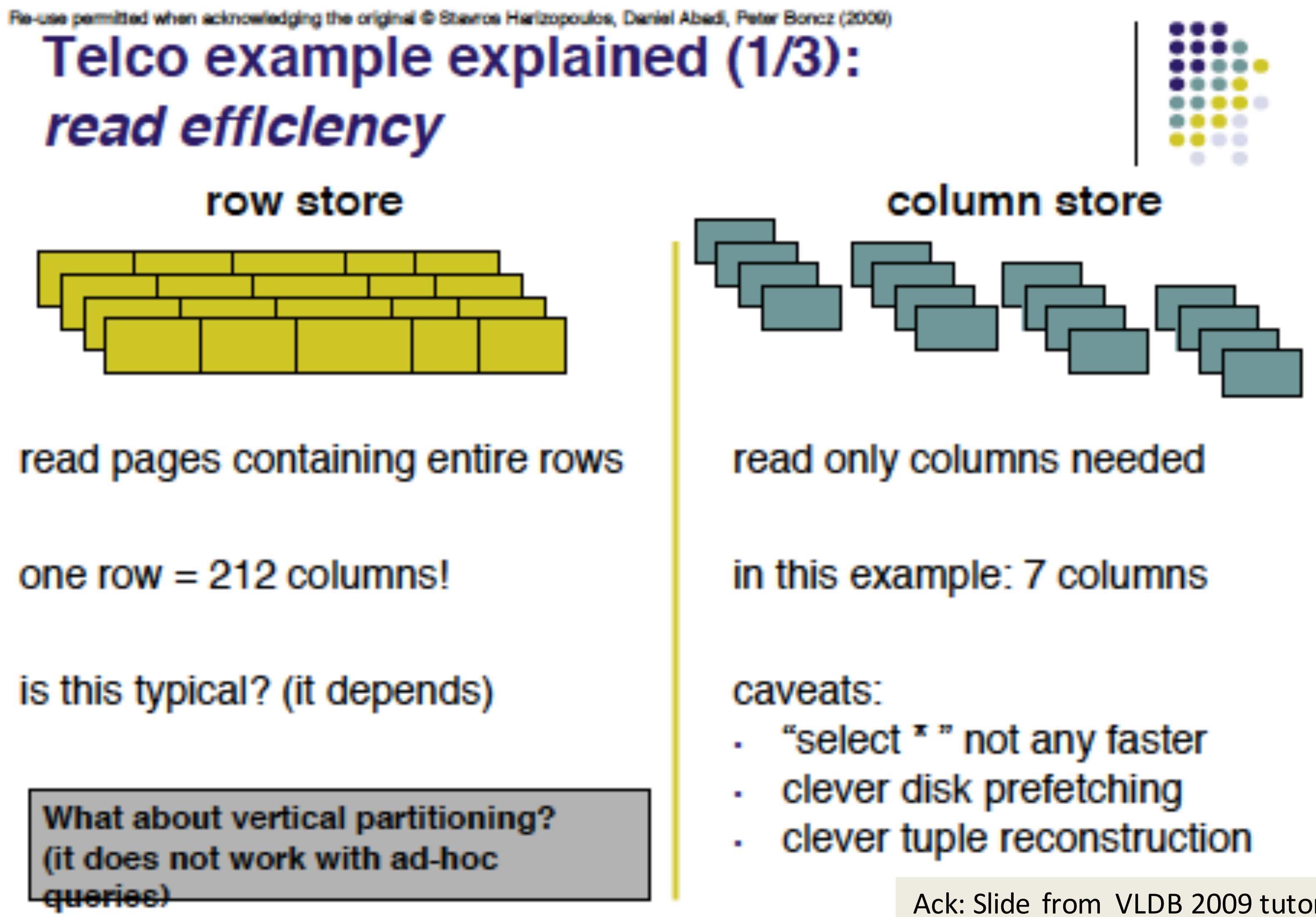


```
QUERY 2
SELECT account.account_number,
sum (usage.toll_airtime),
sum (usage.toll_price)
FROM usage, toll, source, account
WHERE usage.toll_id = toll.toll_id
AND usage.source_id = source.source_id
AND usage.account_id = account.account_id
AND toll.type_ind in ('AE'. 'AA')
AND usage.toll_price > 0
AND source.type != 'CIBER'
AND toll.rating_method = 'IS'
AND usage.invoice_date = 20051013
GROUP BY account.account_number
```

	Column-store	*Row-store*
Query 1	*2.06*	*300*
Query 2	*2.20*	*300*
Query 3	*0.09*	*300*
Query 4	*5.24*	*300*
Query 5	*2.88*	*300*

Why? Three main factors (next slides)

Ack: Slide from VLDB 2009 tutorial on Column store



Telco example explained (1/3): *read efficiency*

row store

read pages containing entire rows

one row = 212 columns!

is this typical? (it depends)

What about vertical partitioning? (it does not work with ad-hoc queries)

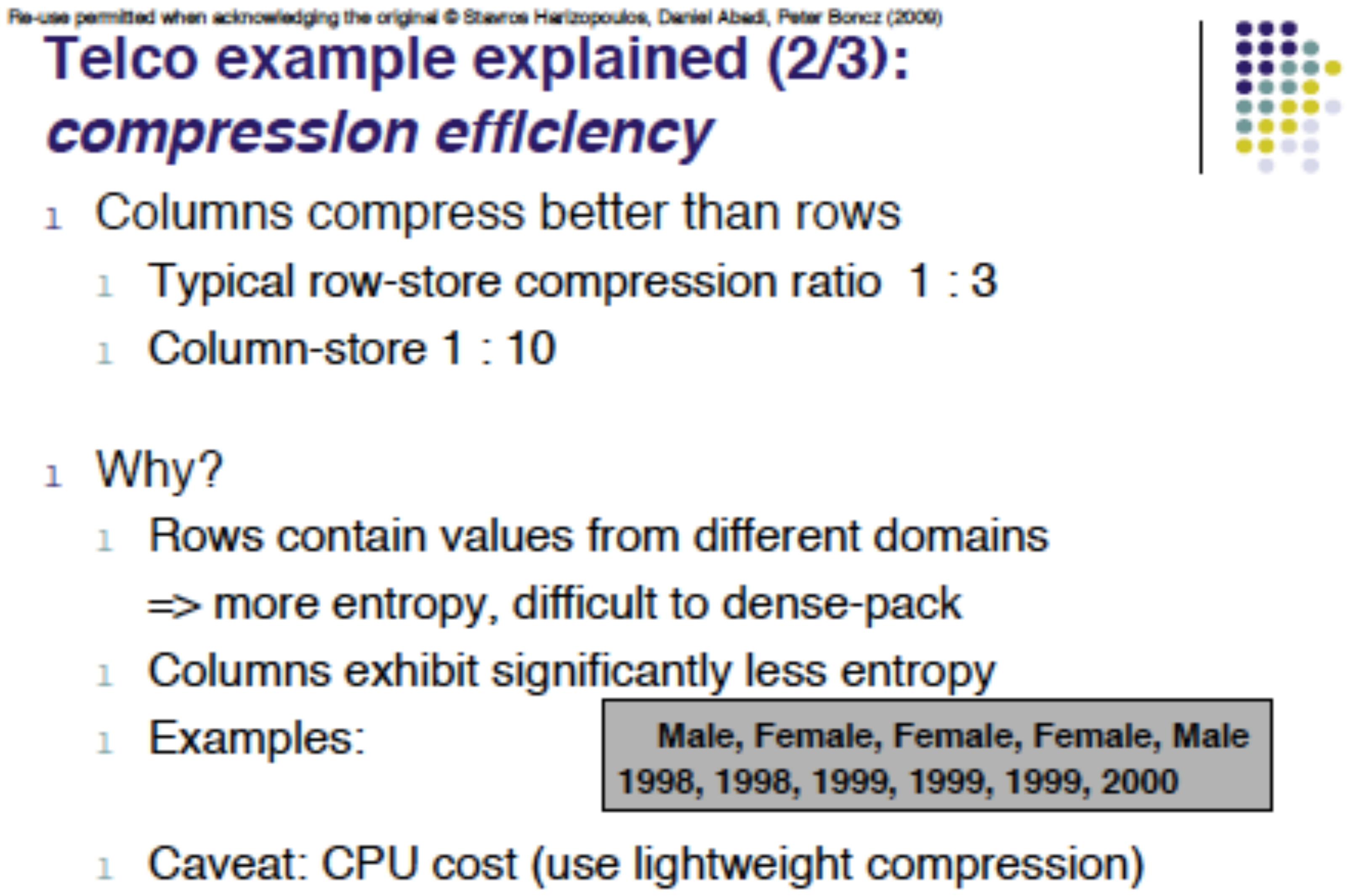
column store

read only columns needed

in this example: 7 columns

caveats:
- “select * ” not any faster
- clever disk prefetching
- clever tuple reconstruction

Ack: Slide from VLDB 2009 tutorial on Column store



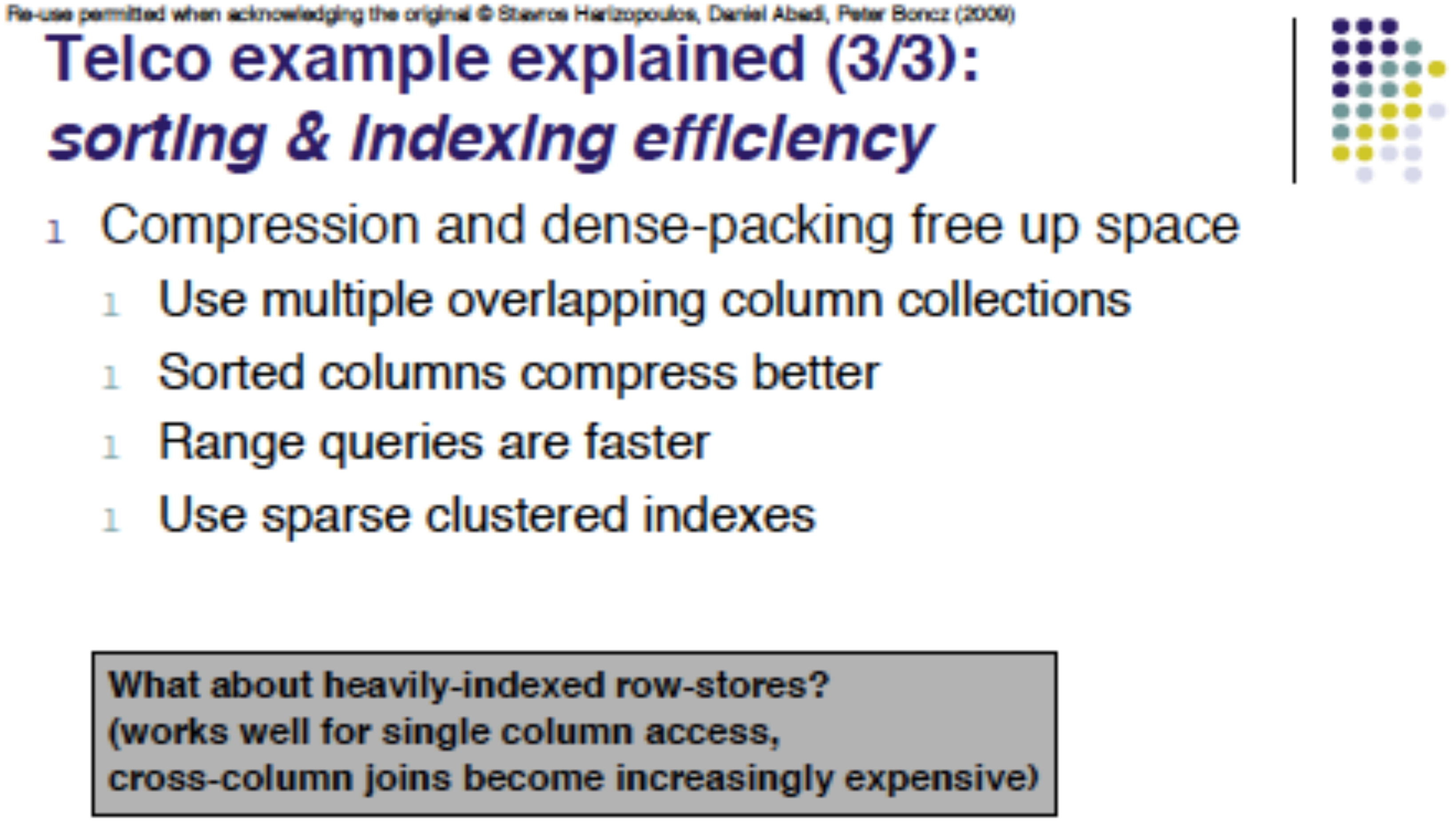
Telco example explained (2/3): *compression efficiency*

- Columns compress better than rows
  - Typical row-store compression ratio 1 : 3
  - Column-store 1 : 10
- Why?
  - Rows contain values from different domains
    => more entropy, difficult to dense-pack
  - Columns exhibit significantly less entropy
  - Examples:

    Male, Female, Female, Female, Male
    1998, 1998, 1999, 1999, 1999, 2000

  - Caveat: CPU cost (use lightweight compression)

Ack: Slide from VLDB 2009 tutorial on Column store



Telco example explained (3/3): *sorting & indexing efficiency*

- Compression and dense-packing free up space
  - Use multiple overlapping column collections
  - Sorted columns compress better
  - Range queries are faster
  - Use sparse clustered indexes

What about heavily-indexed row-stores?
(works well for single column access,
cross-column joins become increasingly expensive)

Ack: Slide from VLDB 2009 tutorial on Column store

NoSQL

Reading Material: NoSQL

Main:

1. "Scalable SQL and NoSQL Data Stores"
Rick Cattell, SIGMOD Record, December 2010 (Vol. 39, No. 4)

Optional:

2. "Dynamo: Amazon's Highly Available Key-value Store" By Giuseppe DeCandia et. al. SOSP 2007

3. "Bigtable: A Distributed Storage System for Structured Data"
Fay Chang et. al. OSDI 2006

So far -- RDBMS

- Relational Data Model
- Relational Database Systems (RDBMS)
- RDBMSs have
  - a complete pre-defined fixed schema
  - a SQL interface
  - and ACID transactions

Today

- NoSQL: ”new” database systems
  - not typically RDBMS
  - relax on some requirements, gain efficiency and scalability
  - new systems choose to use/not use several concepts we learnt so far
    - e.g. System X does not use locks but use multi-version CC (MVCC) or,
    - System Y uses asynchronous replication
  - therefore, it is important to understand the basics (Lectures 1-20) even if they are not used in some new systems

Warnings!

- Material from Cattell's paper (2010-11) – some info will be outdated
- We will focus on the basic ideas of NoSQL systems
- Optional reading slides
  - posted on the webpage
  - a few slides on MongoDB too (in HW5)
  - there are also comparison tables in the Cattell's paper though if you are interested

New Systems

- We will examine a number of SQL and so- called “NoSQL” data stores
  - designed to scale simple OLTP-style application loads in contrast to traditional DBMSs and data warehouses
  - aside: OLAP vs. OLTP?
  - to provide good horizontal scalability for simple read/write database operations distributed over many servers
  - Originally motivated by Web 2.0 applications, these systems are designed to scale to thousands or millions of users
  - To do updates as well as reads

New Systems vs. RDMS

- Contrast the new systems with RDBMS on their
  - data model
  - consistency mechanisms
  - storage mechanisms
  - durability guarantees
  - availability
  - query support
  - and other dimensions.
- These systems typically sacrifice some of these dimensions
  - e.g. database-wide transaction consistency, in order to achieve others, e.g. higher availability and scalability

NoSQL

- Many of the new systems are referred to as "NoSQL" data stores
- NoSQL stands for "Not Only SQL" or "Not Relational"
  - not entirely agreed upon
- Next: six key features of NoSQL systems

NoSQL: Six Key Features

1. the ability to horizontally scale “simple operation” throughput over many servers
2. the ability to replicate and to distribute (partition) data over many servers
3. a simple call level interface
4. a weaker concurrency model than the ACID transactions of most relational (SQL) database systems
5. efficient use of distributed indexes and RAM for data storage
6. the ability to dynamically add new attributes to data records

Important Examples of New Systems

- Memcached
  - popular open source cache
  - supports distributed hashing
  - demonstrated that in-memory indexes can be highly scalable, distributing and replicating objects over multiple nodes
- Dynamo
  - pioneered the idea of eventual consistency as a way to achieve higher availability and scalability
  - data fetched are not guaranteed to be up-to-date, but updates are guaranteed to be propagated to all nodes eventually
- Google's BigTable
  - demonstrated that persistent record storage could be scaled to thousands of nodes

BASE (not ACID ☺)

- Basically Available, Soft state, Eventually consistent

- Recall ACID for RDBMS desired properties of transactions:
  - Atomicity, Consistency, Isolation, and Durability

ACID vs. BASE

- The idea is that by giving up ACID constraints, one can achieve much higher performance and scalability

- The systems differ in how much they give up
  - e.g. most of the systems call themselves "eventually consistent", meaning that updates are eventually propagated to all nodes
  - but many of them provide mechanisms for some degree of consistency, such as multi-version concurrency control (MVCC)

“CAP” Theorem

- Often Eric Brewer’s CAP theorem cited for NoSQL
- A system can have only two out of three of the following properties:
  - Consistency,
  - Availability
  - Partition-tolerance
- The NoSQL systems generally give up consistency
  - However, the trade-offs are complex

"Simple" Operations

- Reading or writing a small number of related records in each operation
  - e.g. key lookups, reads and writes of one record or a small number of records
- This is in contrast to complex queries or joins
- Inspired by web, where millions of users may both read and write data in "simple database operations"
  - e.g. applications may search and update multi-server databases of electronic mail, personal profiles, web postings, wikis, customer records, online dating records, classified ads, and many other kinds of data

Horizontal Scalability

- Shared-Nothing Horizontal Scaling
- The ability to distribute both the data and the load of these simple operations over many servers
  - with no RAM or disk shared among the servers
- Not "vertical" scaling
  - where a database system utilizes many cores and/or CPUs that share RAM and disks
- Some of the systems we describe provide both vertical and horizontal scalability

End of lecture 21

Horizontal vs. Vertical Scaling

- Effective use of multiple cores (vertical scaling) is important
  - but the number of cores that can share memory is limited
- horizontal scaling generally is less expensive
  - can use commodity servers
- Note: horizontal and vertical partitioning are not related to horizontal and vertical scaling
  - except that they are both useful for horizontal scaling (Lecture 19)

In Lecture 22 – up to end

What is different in NOSQL systems

- Points how NOSQL systems differ from each other and from RDBMSs:

1. Concurrency Control
   a) Locks
   b) MVCC
   c) None (do not provide atomicity)
   d) ACID (pre-analyze transactions to avoid conflicts)
2. Data Storage Medium
   a) Storage in RAM – snapshots or replication to disk
   b) Disk storage – caching in RAM
3. Replication – whether mirror copies are always in sync
   a) Synchronous
   b) Asynchronous (faster, but updates may be lost in a crash)
   c) Both (local copies synchronously, remote copies asynchronously)
4. Transaction Mechanisms
   a) support
   b) do not support
   c) in between – support local transactions only within a single object or shard

Data Model Terminology for NoSQL

- Unlike SQL/RDBMS, the terminology for NoSQL is often inconsistent
  - we are following notations in Cattell's paper
- All systems provide a way to store scalar values
  - e.g. numbers and strings
- Some of them also provide a way to store more complex nested or reference values
- The systems all store sets of attribute-value pairs
  - but use different data structures
- Next:
  1. Tuple
  2. Document
  3. Extensible Record
  4. Object

1. Tuple

- same as before
- A "tuple" is a row in a relational table
  - attribute names are pre-defined in a schema
  - the values must be scalar
  - the values are referenced by attribute name
  - in contrast to an array or list, where they are referenced by ordinal position

2. Document

- Allows values to be nested documents or lists as well as scalar values
- The attribute names are dynamically defined for each document at runtime
- A document differs from a tuple in that the attributes are not defined in a global schema
  - and a wider range of values are permitted

3. Extensible Record

- A hybrid between a tuple and a document
  - families of attributes are defined in a schema
  - but new attributes can be added (within an attribute family) on a per-record basis
  - Attributes may be list-valued

4. Object

- Analogous to an object in programming languages
  – but without the procedural methods

- Values may be references or nested objects

Data Store Categories

- The data stores are grouped according to their data model
- Key-value Stores:
  - store values and an index to find them, based on a programmer-defined key
- Document Stores:
  - store documents -- The documents are indexed and a simple query mechanism is provided
- Extensible Record Stores:
  - These systems store extensible records that can be partitioned vertically and horizontally across nodes
  - Some papers call these "wide column stores"
- Relational Databases:
  - These systems store (and index and query) tuples
  - e.g. the new RDBMSs that provide horizontal scaling

SQL vs. NOSQL

Still, a controversial topic

Advantages: RDBMS

1. If new relational systems can do everything that a NoSQL system can, with analogous performance and scalability, and with the convenience of transactions and SQL, why would you choose a NoSQL system?
2. Relational DBMSs have taken and retained majority market share over other competitors in the past 30 years: network, object, and XML DBMSs
3. Successful relational DBMSs have been built to handle other specific application loads in the past:
   - read-only or read-mostly data warehousing
   - OLTP on multi-core multi-disk CPUs
   - in-memory databases
   - distributed databases, and
   - now horizontally scaled databases
4. While we don't see "one size fits all" in the SQL products themselves, we do see a common interface with SQL, transactions, and relational schema that give advantages in training, continuity, and data interchange

Advantages: NOSQL - 1

1. We haven't yet seen good benchmarks showing that RDBMSs can achieve scaling comparable with NoSQL systems like Google's BigTable
2. If you only require a lookup of objects based on a single key
   - then a key-value store is adequate and probably easier to understand than a relational DBMS
   - Likewise for a document store on a simple application: you only pay the learning curve for the level of complexity you require
3. Some applications require a flexible schema
   - allowing each object in a collection to have different attributes
   - While some RDBMSs allow efficient "packing" of tuples with missing attributes, and some allow adding new attributes at runtime, this is uncommon

Advantages: NOSQL - 2

4. A relational DBMS makes "expensive" (multi- node multi-table) operations "too easy"
   - NoSQL systems make them impossible or obviously expensive for programmers

5. While RDBMSs have maintained majority market share over the years, other products have established smaller but non-trivial markets in areas where there is a need for particular capabilities
   - e.g. indexed objects with products like BerkeleyDB, or graph-following operations with object-oriented DBMSs