CompSci 516
Data Intensive Computing Systems

Lecture 18
NoSQL
and
Column Store

Instructor: Sudeepa Roy

Announcements

- HW3 (last HW) has been posted on Sakai

- Same problems as in HW1 but in MongoDB (NOSQL)

- Due in two weeks after today's lecture (~11/16)

Reading Material

NOSQL:

- "Scalable SQL and NoSQL Data Stores"

Rick Cattell, SIGMOD Record, December 2010 (Vol. 39, No. 4)

- see webpage http://cattell.net/datastores/ for updates and more pointers

Column Store:

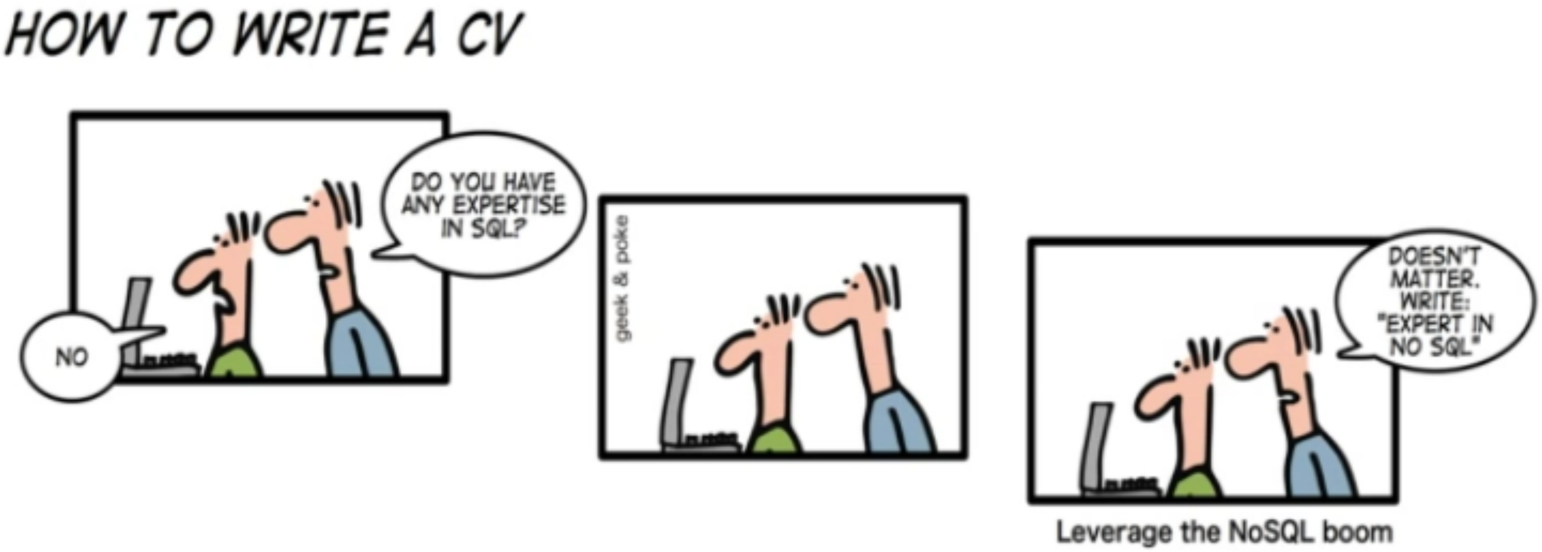
- D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos and S. Madden. *The Design and Implementation of Modern Column-Oriented Database Systems*. Foundations and Trends in Databases, vol. 5, no. 3, pp. 197–280, 2012.
- See VLDB 2009 tutorial: http://nms.csail.mit.edu/~stavros/pubs/tutorial2009-column_stores.pdf

Optional:

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

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

NoSQL


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

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 1-17) even if they are not used in some new systems!

Warnings!

- Material from Cattell’s paper (2010-11) – some info will be outdated
  - see webpage http://cattell.net/datastores/ for updates and more pointers
- We will focus on the basic ideas of NoSQL systems
- Optional reading slides at the end
  - there are also comparison tables in the Cattell’s paper if you are interested

OLAP vs. OLTP

- OLTP (OnLine Transaction Processing)
  - Recall transactions!
  - Multiple concurrent read-write requests
  - Commercial applications (banking, online shopping)
  - Data changes frequently
  - ACID properties, concurrency control, recovery
- OLAP (OnLine Analytical Processing)
  - Many aggregate/group-by queries – multidimensional data
  - Data mostly static
  - Will study OLAP Cube soon

New Systems

- We will examine a number of SQL and so- called “NoSQL” systems or “data stores”
- Designed to scale simple OLTP-style application loads
  - to do updates as well as reads
  - in contrast to traditional DBMSs and data warehouses
  - 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

New Systems vs. RDMS

- When you study a new system, compare it with RDBMS-s on its
  - data model
  - consistency mechanisms
  - storage mechanisms
  - durability guarantees
  - availability
  - query support
- 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 operations” throughput over many servers
2. the ability to replicate and to distribute (partition) data over many servers
3. a simple call level interface or protocol (in contrast to SQL binding)
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

- Three systems provided a “proof of concept” and inspired many other data stores

1. Memcached
2. Amazon’s Dynamo
3. Google’s BigTable

1. Memcached: main features

- popular open source cache
- supports distributed hashing (later)
- demonstrated that in-memory indexes can be highly scalable, distributing and replicating objects over multiple nodes

2. Dynamo : main features

- 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

3. BigTable : main features

- demonstrated that persistent record storage could be scaled to thousands of nodes

BASE (not ACID ☺)

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

- NOSQL systems typically do not provide ACID

- Basically Available
- Soft state
- Eventually consistent

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

Two foci for NoSQL systems

1. “Simple” operations

2. Horizontal Scalability

1. “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, joins, or read-mostly access

- Inspired by web, where millions of users may both read and write data in simple operations
  - e.g. 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

2. 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

2. 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 17)

What is different in NOSQL systems

- When you study a new NOSQL system, notice how it differs from RDBMS in terms of

1. Concurrency Control
2. Data Storage Medium
3. Replication
4. Transactions

Choices in NOSQL systems:
1. Concurrency Control

a) Locks
  - some systems provide one-user-at-a-time read or update locks
  - MongoDB provides locking at a field level

b) MVCC

c) None
  - do not provide atomicity
  - multiple users can edit in parallel
  - no guarantee which version you will read

d) ACID
  - pre-analyze transactions to avoid conflicts
  - no deadlocks and no waits on locks

Choices in NOSQL systems:
2. Data Storage Medium

a) Storage in RAM
  - snapshots or replication to disk
  - poor performance when overflows RAM

b) Disk storage
  - caching in RAM

Choices in NOSQL systems:
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

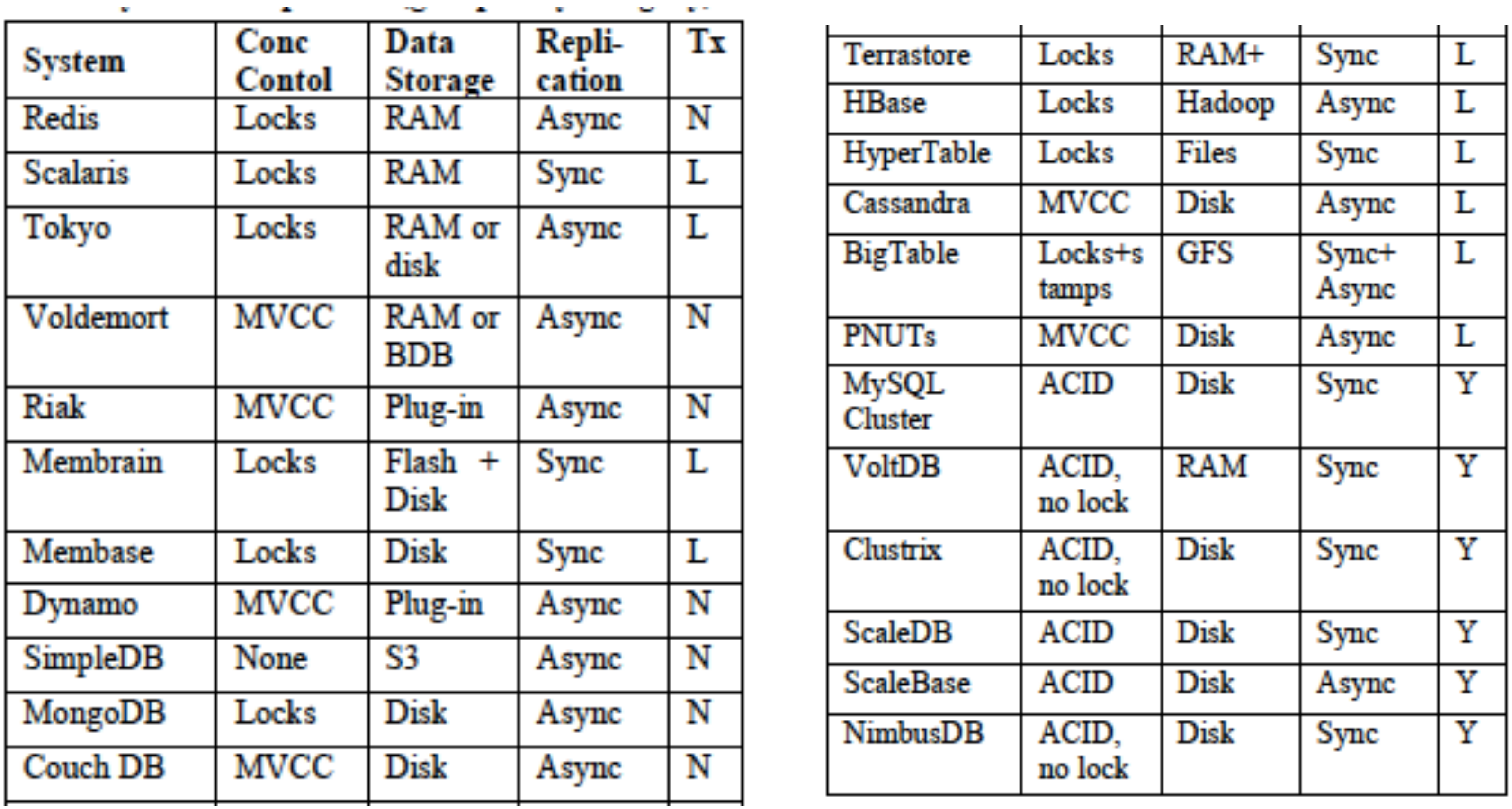
Choices in NOSQL systems: 4. Transaction Mechanisms

a) support

b) do not support

c) in between
   - support local transactions only within a single object or “shard”
   - shard = a horizontal partition of data in a database

Comparison from Cattell's paper (2011)



System	Conc Contol	Data Storage	Repli-cation	Tx
Redis	Locks	RAM	Async	N
Scalaris	Locks	RAM	Sync	L
Tokyo	Locks	RAM or disk	Async	L
Voldemort	MVCC	RAM or BDB	Async	N
Riak	MVCC	Plug-in	Async	N
Membrain	Locks	Flash + Disk	Sync	L
Membase	Locks	Disk	Sync	L
Dynamo	MVCC	Plug-in	Async	N
SimpleDB	None	S3	Async	N
MongoDB	Locks	Disk	Async	N
Couch DB	MVCC	Disk	Async	N
Terrastore	Locks	RAM+	Sync	L
HBase	Locks	Hadoop	Async	L
HyperTable	Locks	Files	Sync	L
Cassandra	MVCC	Disk	Async	L
BigTable	Locks+stamps	GFS	Sync+ Async	L
PNUTs	MVCC	Disk	Async	L
MySQL Cluster	ACID	Disk	Sync	Y
VoltDB	ACID, no lock	RAM	Sync	Y
Clustrix	ACID, no lock	Disk	Sync	Y
ScaleDB	ACID	Disk	Sync	Y
ScaleBase	ACID	Disk	Async	Y
NimbusDB	ACID, no lock	Disk	Sync	Y

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

Data Model Terminology for NoSQL

- The systems all store sets of attribute-value pairs
  - but use four different data structures

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:
  - store extensible records that can be partitioned vertically and horizontally across nodes
  - Some papers call these “wide column stores”
- Relational Databases:
  - store (and index and query) tuples
  - e.g. the new RDBMSs that provide horizontal scaling

Example NOSQL systems

- Key-value Stores:
  - Project Voldemort, Riak, Redis, Scalaris, Tokyo Cabinet, Memcached/Membrain/Membase
- Document Stores:
  - Amazon SimpleDB, CouchDB, MongoDB, Terrastore
- Extensible Record Stores:
  - Hbase, HyperTable, Cassandra, Yahoo's PNUTS
- Relational Databases:
  - MySQL Cluster, VoltDB, Clustrix, ScaleDB, ScaleBase, NimbusDB, Google Megastore (a layer on BigTable)

Key-value store: 1/2

- The simplest data stores
- data model similar to the memcached distributed in-memory cache
  - with a single key-value index for all the data
  - does not provide secondary indices or keys
- but unlike memcached, generally provide
  - a persistence mechanism
  - additional functionality like replication, versioning, locking, transactions, sorting, etc
- The client interface provides inserts, deletes, and index lookups

Key-value store: 2/2

- All key-value stores provide scalability through key distribution over nodes
- Voldemort, Riak, Tokyo Cabinet, and enhanced memcached systems can store data in RAM or on disk
  - The others store data in RAM, and provide disk as backup, or rely on replication and recovery so that a backup is not needed
- Scalaris and enhanced memcached systems use synchronous replication
  - the rest use asynchronous
- Scalaris and Tokyo Cabinet implement transactions
  - the others do not.
- Voldemort and Riak use multi-version concurrency control
  - the others use locks

Use Case : Key-value store

- if you have a simple application with only one kind of object, and you only need to look up objects up based on one attribute
- Suppose you have a web application
  - that does many RDBMS queries to create a tailored page when a user logs in
  - Suppose it takes several seconds to execute those queries, and the user's data is rarely changed
  - you might want to store the user's tailored page as a single object in a key-value store

Document store: 1/3

- Document stores support more complex data than the key-value stores
- "document store" may be confusing
  - these systems could store "documents" in the traditional sense (articles, Microsoft Word files, etc.)
  - but a document in these systems can be any kind of "pointerless object"
- Unlike the key-value stores, these systems generally support
  - secondary indexes
  - multiple types of documents (objects) per database, and
  - nested documents or lists
- Like other NoSQL systems, the document stores do not provide ACID transactional properties

Document store: 2/3

- The document stores are schema-less, except for
  - attributes (which are simply a name, and are not pre- specified)
  - collections (which are simply a grouping of documents), and
  - indexes defined on collections (explicitly defined, except in SimpleDB)
  - There are some differences in their data models, e.g. SimpleDB does not allow nested documents
- The document stores are very similar but use different terminology
  - e.g. a SimpleDB Domain = CouchDB Database = MongoDB Collection (= Terrastore Bucket)
  - SimpleDB calls documents "items"
  - an attribute is a field in CouchDB, or a key in MongoDB (or Terrastore)

Document store: 3/3

- Unlike the key-value stores, the document stores “typically” provide a mechanism to query collections based on multiple attribute value constraints
- do not provide explicit locks
  - have weaker concurrency and atomicity properties than traditional ACID-compliant databases
- Documents can be distributed over nodes in all of the systems
  - All of the systems can achieve scalability by reading (potentially) out-of-date replicas

Use case: Document Store

- application with multiple different kinds of objects
  - e.g. in a Department of Motor Vehicles application, with vehicles and drivers
- where you need to look up objects based on multiple fields
  - e.g., a driver's name, license number, owned vehicle, or birth date

Extensible Record Stores : 1/1

- Motivated by Google's success with BigTable
  - still the recent extensible record stores cannot come close to BigTable's scalability
- Basic data model is rows and columns
- Basic scalability model is splitting both rows and columns over multiple nodes
- Rows are split across nodes through sharding on the primary key
  - They typically split by range rather than a hash function
- Columns of a table are distributed over multiple nodes by using "column groups"
  - a way for the customer to indicate which columns are best stored together
- Both horizontal and vertical partitioning can be used simultaneously on the same table

Use case: Extensible Record Store

- uses cases similar to those for document stores:
  - multiple kinds of objects, with lookups based on any field.
- However, aimed at higher throughput, and may provide stronger concurrency guarantees,
  - at the cost of slightly morecomplexity than the document stores
- Suppose storing customer information for an eBay-style application, and you want to partition your data both horizontally and vertically:
  - cluster customers by country, so that you can efficiently search all of the customers in one country
  - separate the rarely-changed "core" customer information such as customer addresses and email addresses in one place, and
  - put certain frequently-updated customer information (such as current bids in progress) in a different place, to improve performance

Scalable RDBMS : 1/1

- Some RDBMSs are expected to provide scalability comparable with NoSQL data stores
- But, with two provisos:
  - Use small-scope operations: Operations that span many nodes, e.g. joins over many tables, will not scale well with sharding
  - Use small-scope transactions: Likewise, transactions that span many nodes are going to be very inefficient, with the communication and 2PC overhead
- Typical NOSQL systems make these two impossible
- Scalable RDBMS allows them, but penalizes a customer for these operations
- Have higher-level SQL language and ACID properties
  - but pay a price when they span nodes

Use case: Scalable RDBMS

- If your application requires many tables with different types of data
  - a relational schema centralizes and simplifies data definition and SQL simplifies operations
  - or for projects with many programmers
- However, more useful if the application does not require
  - updates or joins that span many nodes
  - transaction coordination
  - or, data movement

Consistent Hashing

in DynamoDB

Consistent Hashing (CH)

- Recall dynamic hashing schemes
- If the #of slots (directory size) changes, then almost all keys had to be remapped
- In consistent hashing (CH), with #keys = K and #slots = N, only K/N keys need to be remapped on average
- Applies to the design of Distributed Hash Table (DHTs) for Uniform Load Distribution
  - partition a keyspace among a set of sites/nodes
  - additionally provide an overlay network that connects nodes such that the nodes responsible for any key can be efficiently located

DynamoDB : CH 1/2

- [ref. the DynamoDB paper, sec 4.3]
- Must scale incrementally
- Consistent hashing is used to dynamically distribute data around a "ring" of nodes (=sites)
- The output of a hash function is treated as a circular ring
- Each node is assigned a random value in this space
  - represents the "position" on the ring

- Data item identified by a key
- Assign to a node by hashing the key to

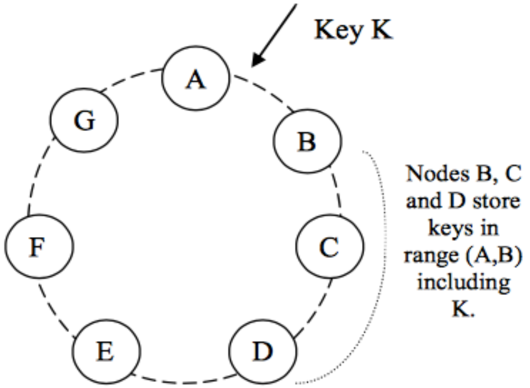


Figure 2: Partitioning and replication of keys in Dynamo ring.

DynamoDB : CH 2/2

- Data item identified by a key
- Assign to a node by hashing the key to yield its position on the ring
- Walk the ring clockwise to find the first node with a position larger than the item’s position
- Each node is responsible for the region in the ring between it and its predecessor node on the ring

- Note:
- departure or arrival of a node only affects its immediate neighbor
- The other nodes remain unaffected
- K/N on average!

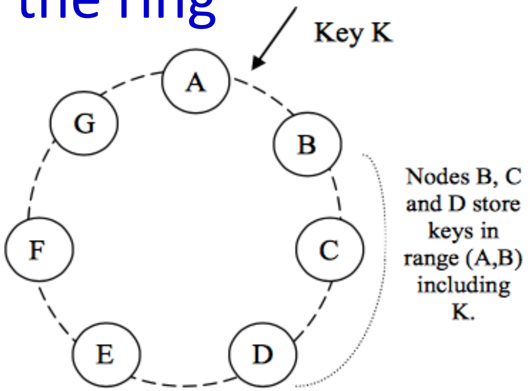


Figure 2: Partitioning and replication of keys in Dynamo ring.

DynamoDB : Challenges in CH

- However, this basic CH algorithm poses some challenges

1. Random position assignment of each node on the ring leads to non-uniform data and load distribution
2. The basic algorithm is oblivious to the heterogeneity in the performance of the nodes

- Solution: Dynamo uses a variant of CH
  - Each node gets assigned to multiple points in the ring
  - called "virtual node"
  - one node takes care of multiple virtual nodes

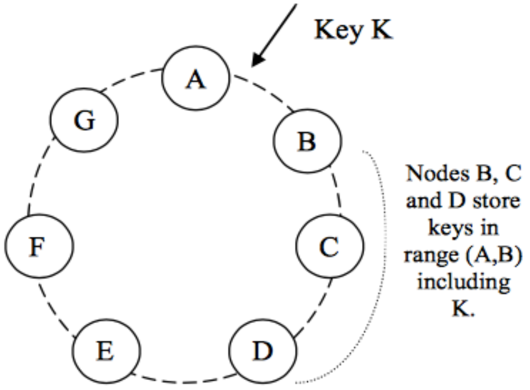


Figure 2: Partitioning and replication of keys in Dynamo ring.

DynamoDB: Virtual Nodes

- Using virtual nodes has advantages

1. If a node becomes unavailable (due to failures or routine maintenance), the load handled by this node is evenly dispersed across the remaining available nodes
2. When a node becomes available again, or a new node is added to the system, the newly available node accepts a roughly equivalent amount of load from each of the other available nodes
3. The number of virtual nodes that a node is responsible can decided based on its capacity, accounting for heterogeneity in the physical infrastructure

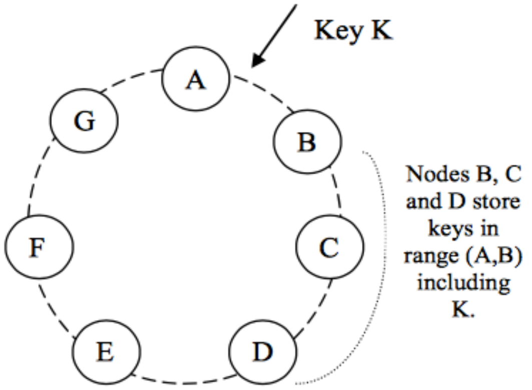


Figure 2: Partitioning and replication of keys in Dynamo ring.

DynamoDB: Replication

- Dynamo replicates its data on multiple (N) hosts for high availability and durability
- Each key k is assigned to a coordinator which is in charge of replication
  - coordinator handles all keys in its range
- Coordinator replicates each key it is in charge of
  - by storing it locally
  - replicating it at the N-1 clockwise succesor nodes in the ring
- Each node is in charge of region of the ring between it and its N-th predecessor

Node B replicates key K at nodes C and D

Node D will store keys in the range (A, B], (B, C], (C, D]

Note: there may be < N "physical" nodes

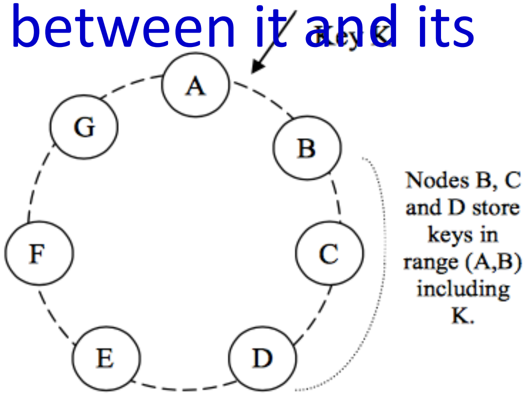


Figure 2: Partitioning and replication of keys in Dynamo ring.

CH History

- Proposed by CS theoreticians from MIT:
  - Karger-Lehman-Leighton-Panigrahy-Levine-Lewin
  - *"Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web"* – STOC 1997
- Consistent hashing gave birth to Akamai Technologies
  - Founded by Danny Lewin and Tom Leighton in 1998
  - Akamai's content delivery network is one of the largest distributed computing platforms
  - Now market cap $12B and 6200 employees
  - Managing web-presence of many major companies
- 2001: The concept of Distributed Hash Table (DHT) is proposed (how to look for a file) and CH was re-purposed
- Now used in Dynamo, Couchbase, Cassandra, Voldemort, Riak, ..

SQL vs. NOSQL

Arguments for both sides
still a controversial topic

Why choose RDBMS over NoSQL : 1/3

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, NoSQL is not needed

2. Relational DBMSs have taken and retained majority market share over other competitors in the past 30 years
   - (network, object, and XML DBMSs)

Why choose RDBMS over NoSQL : 2/3

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

Why choose RDBMS over NoSQL : 3/3

4. While no “one size fits all” in the SQL products themselves, there is a common interface with SQL, transactions, and relational schema that give advantages in training, continuity, and data interchange

Why choose NoSQL over RDBMS : 1/3

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

Why choose NoSQL over RDBMS : 2/3

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

Why choose NoSQL over RDBMS : 3/3

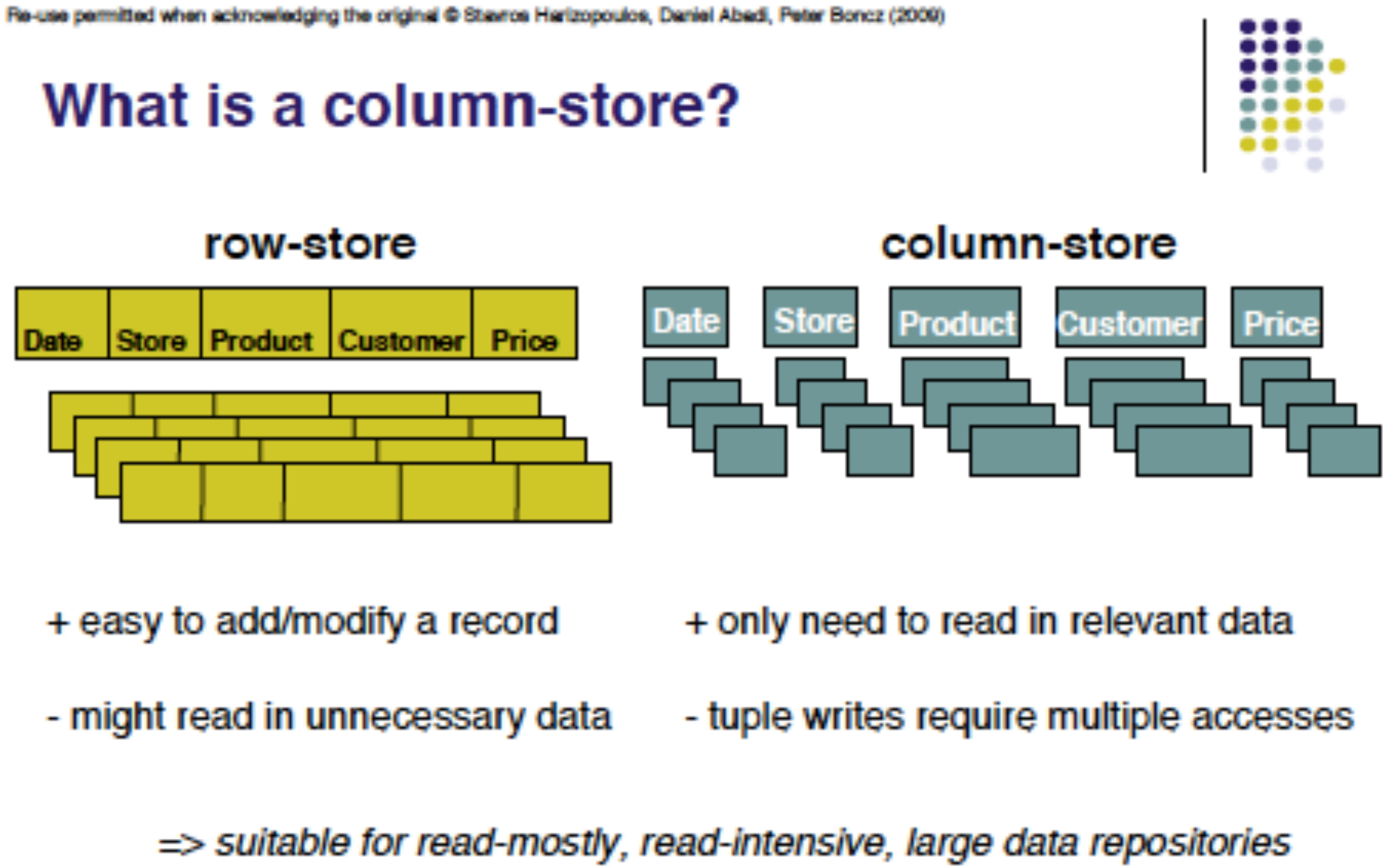
4. A relational DBMS makes “expensive” (multi- node multi-table) operations “too easy”
   - NoSQL systems make them impossible or obviously expensive for programmers

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

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, Vertica (earlier, C-store), SAP/Sybase IQ, Google Bigtable (with column groups)



What is a column-store?

row-store

column-store

+ easy to add/modify a record

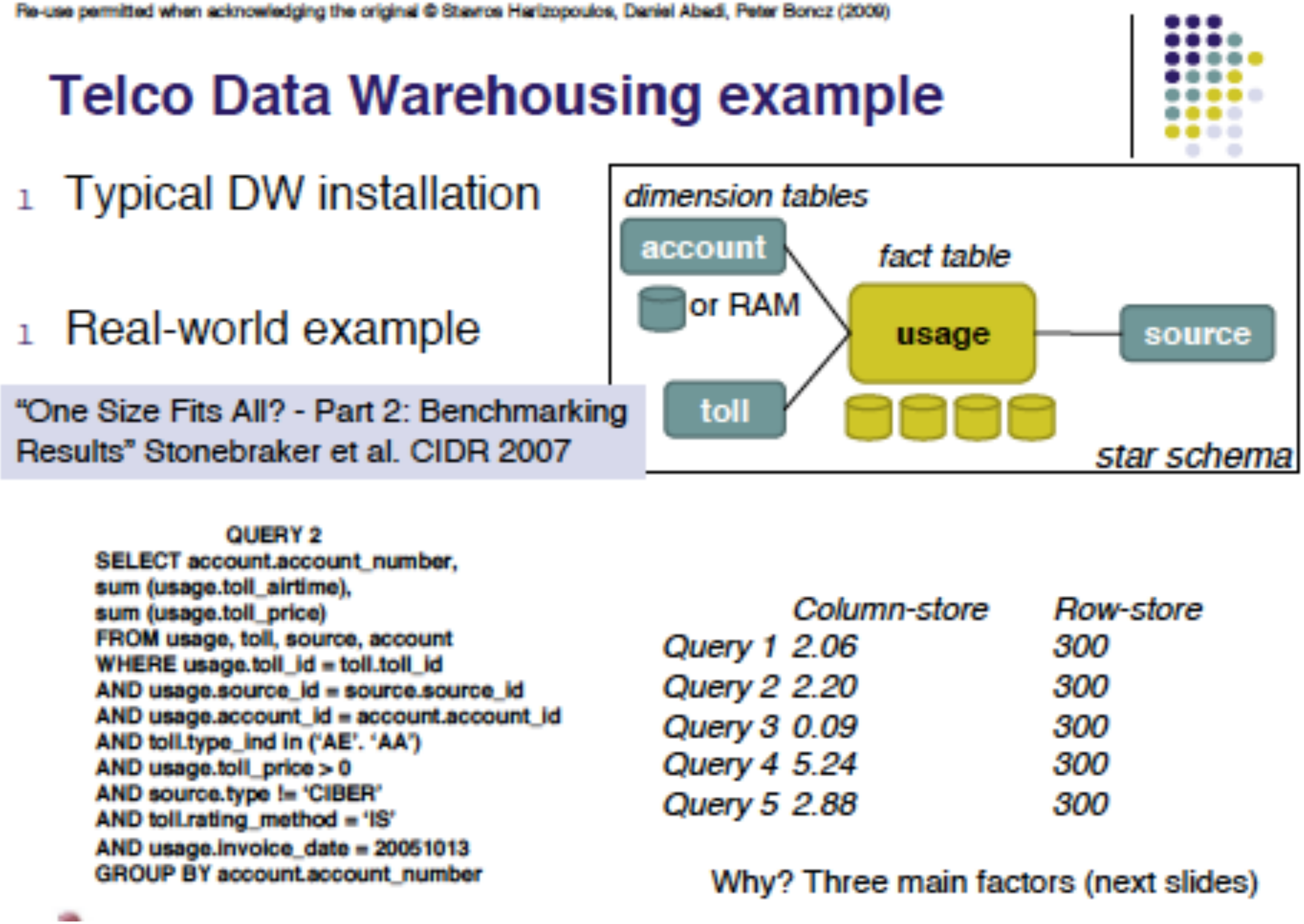
- might read in unnecessary data

+ only need to read in relevant 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

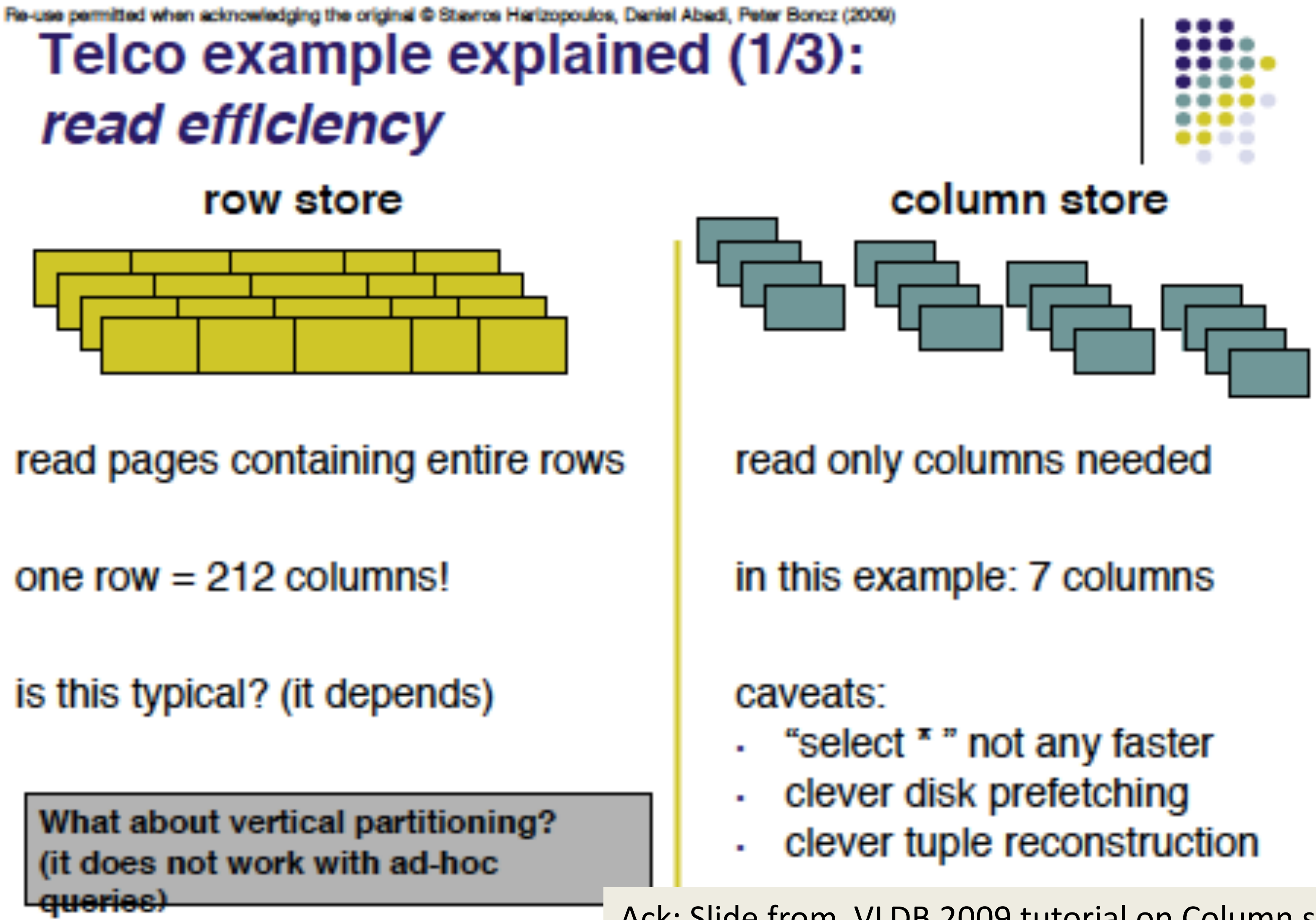
"One Size Fits All? - Part 2: Benchmarking Results" Stonebraker et al. CIDR 2007

```
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



Re-use permitted when acknowledging the original © Stavros Harizopoulos, Daniel Abadi, Peter Boncz (2009)

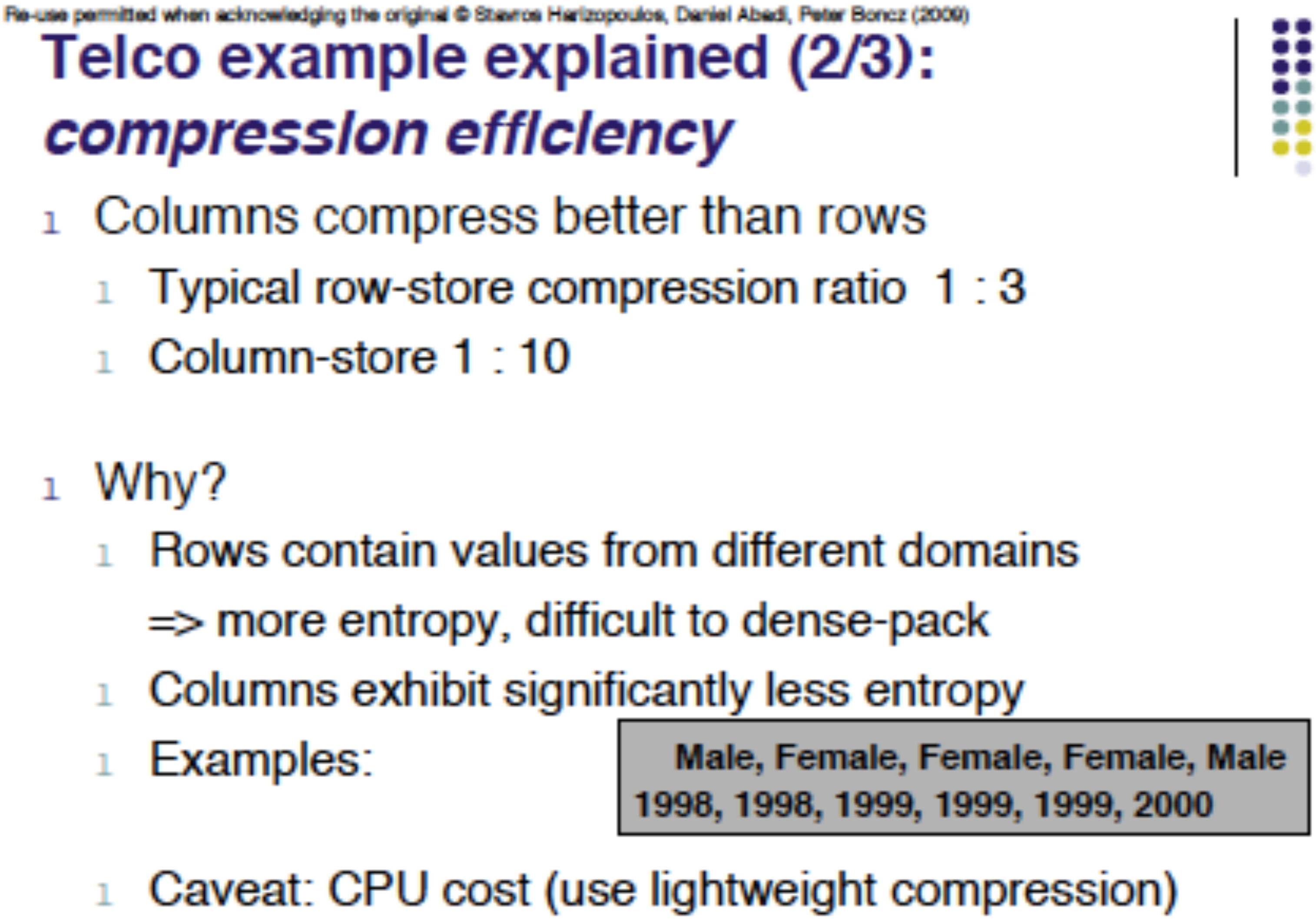
Telco example explained (1/3): *read efficiency*

row store	column store
read pages containing entire rows	read only columns needed
one row = 212 columns!	in this example: 7 columns
is this typical? (it depends)	caveats:

- "select * " not any faster
- clever disk prefetching
- clever tuple reconstruction

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

Ack: Slide from VLDB 2009 tutorial on Column store



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

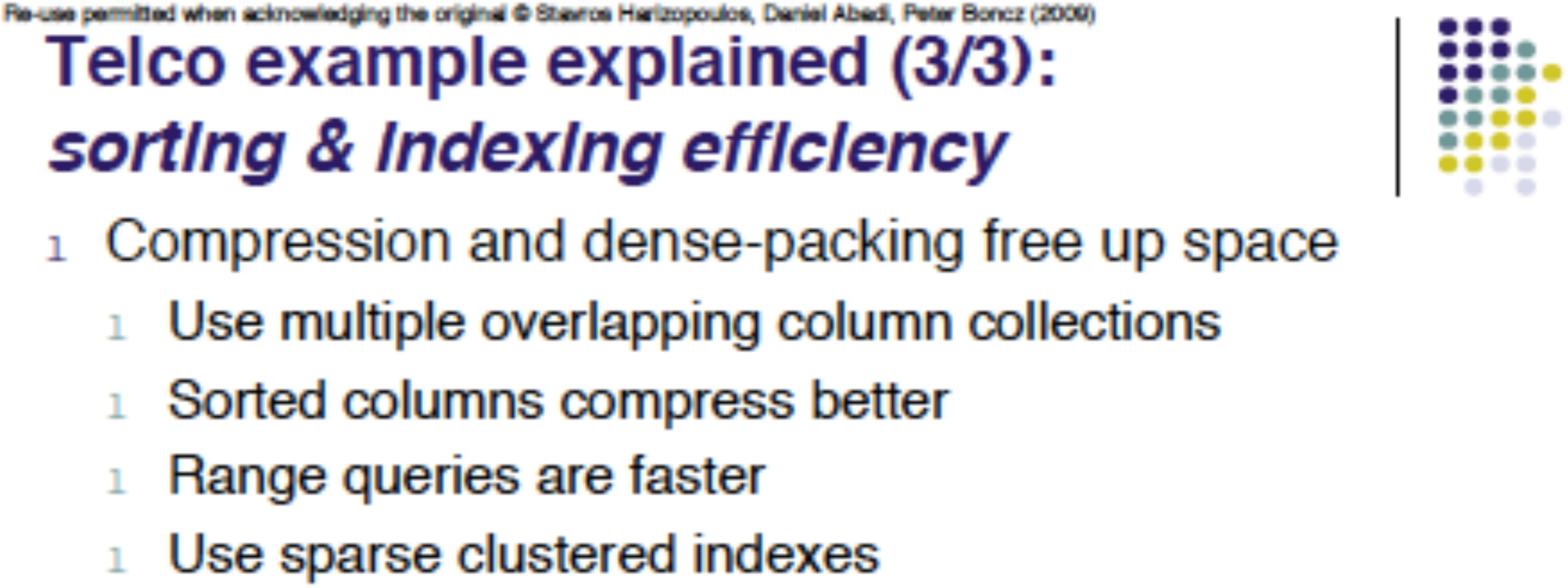
- Columns compress better than rows
  - Typical row-store compression ratio 1 : 3
  - Column-store 1 : 10

- 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

Ack: Slide from VLDB 2009 tutorial on Column store

Additional and Optional Slides on MongoDB

(May be useful for HW3)
https://docs.mongodb.com

Optional slide: Read yourself

MongoDB

- MongoDB is an open source document store written in C++
- provides indexes on collections
- lockless
- provides a document query mechanism
- supports automatic sharding
- Replication is mostly used for failover
- does not provide the global consistency of a traditional DBMS
  - but you can get local consistency on the up-to-date primary copy of a document
- supports dynamic queries with automatic use of indices, like RDBMSs
- also supports map-reduce – helps complex aggregations across docs
- provides atomic operations on fields

Optional slide: Read yourself

MongoDB: Atomic Ops on Fields

- The update command supports “modifiers” that facilitate atomic changes to individual values
  - $set sets a value
  - $inc increments a value
  - $push appends a value to an array
  - $pushAll appends several values to an array
  - $pull removes a value from an array, and $pullAll removes several values from an array
- Since these updates normally occur “in place”, they avoid the overhead of a return trip to the server
- There is an “update if current” convention for changing a document only if field values match a given previous value
- MongoDB supports a findAndModify command to perform an atomic update and immediately return the updated document
  - useful for implementing queues and other data structures requiring atomicity

Optional slide: Read yourself

MongoDB: Index

- MongoDB indices are explicitly defined using an ensureIndex call
  - any existing indices are automatically used for query processing
- To find all products released last year (2015) or later costing under $100 you could write:
- db.products.find(
  {released: {$gte: new Date(2015, 1, 1,)}, price
  {‘$lte’: 100},})

Optional slide: Read yourself

MongoDB: Data

- MongoDB stores data in a binary JSON-like format called BSON
  - BSON supports boolean, integer, float, date, string and binary types
  - MongoDB can also support large binary objects, eg. images and videos
  - These are stored in chunks that can be streamed back to the client for efficient delivery

Optional slide: Read yourself

MongoDB: Replication

- MongoDB supports master-slave replication with automatic failover and recovery
  - Replication (and recovery) is done at the level of shards
  - Replication is asynchronous for higher performance, so some updates may be lost on a crash