Carnegie Mellon University

Lecture #10

ADVANCED DATABASE SYSTEMS

Storage Models & Data Layout

@Andy_Pavlo // 15-721 // Spring 2018

Altibase Challenges Oracle, IBM & Microsoft

ALTIBASE

Mature, battle-tested database is now open source

NEWS PROVIDED BY
Altibase →
Feb 12, 2018, 10:47 ET

NEW YORK, Feb. 12, 2018 /PRNewswire/ -- On February 12 in 2018, Altibase, an enterprise grade relational database, announced that it is now open source.

"The database industry is going open source – the trend is clear," says Altibase Chairman, Paul Nahm. "But for discerning and prudent enterprise clients with mission critical applications there is still one big question: Is there an open source database I can trust to be reliable 24/7? The answer is as of today: Yes, Altibase."

Resources Blog Journalists

CISION PR Newswire

News Products Contact

News in Focus Business & Money Science & Tech Lifestyle & Health

Altibase Challenges Ora & Microsoft

Mature, battle-tested database is now open source

NEWS PROVIDED BY
Altibase →
Feb 12, 2018, 10:47 ET

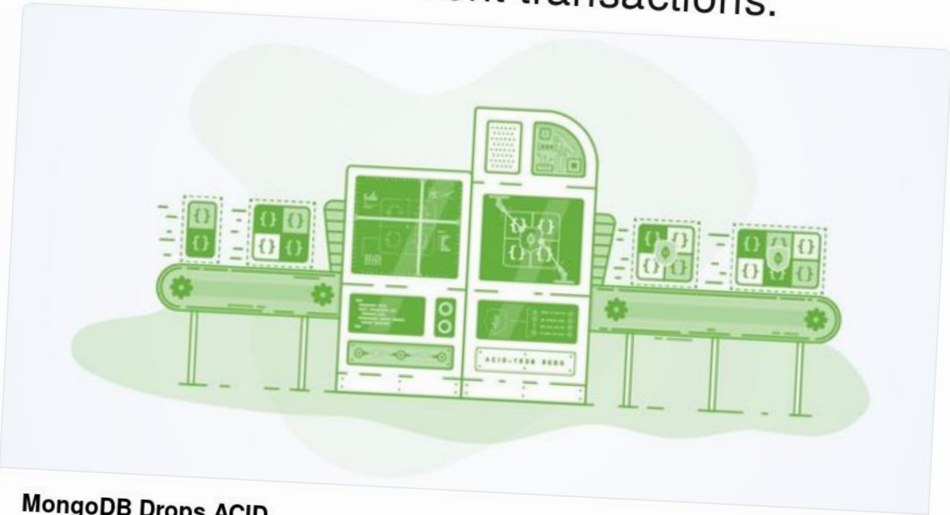
NEW YORK, Feb. 12, 2018 /PRNewswire/ -- On February
announced that it is now open source.

"The database industry is going open source – the tren
and prudent enterprise clients with mission critical a
database I can trust to be reliable 24/7? The answer i

Log In Sign Up

Eliot @eliothorowitz Following

Today I announced that MongoDB 4.0 will support multi-document transactions:



MongoDB Drops ACID
MongoDB 4.0 will add support for multi-document transactions, making it the only database to combine the speed, flexibility, and power of the document model wit...
mongodb.com

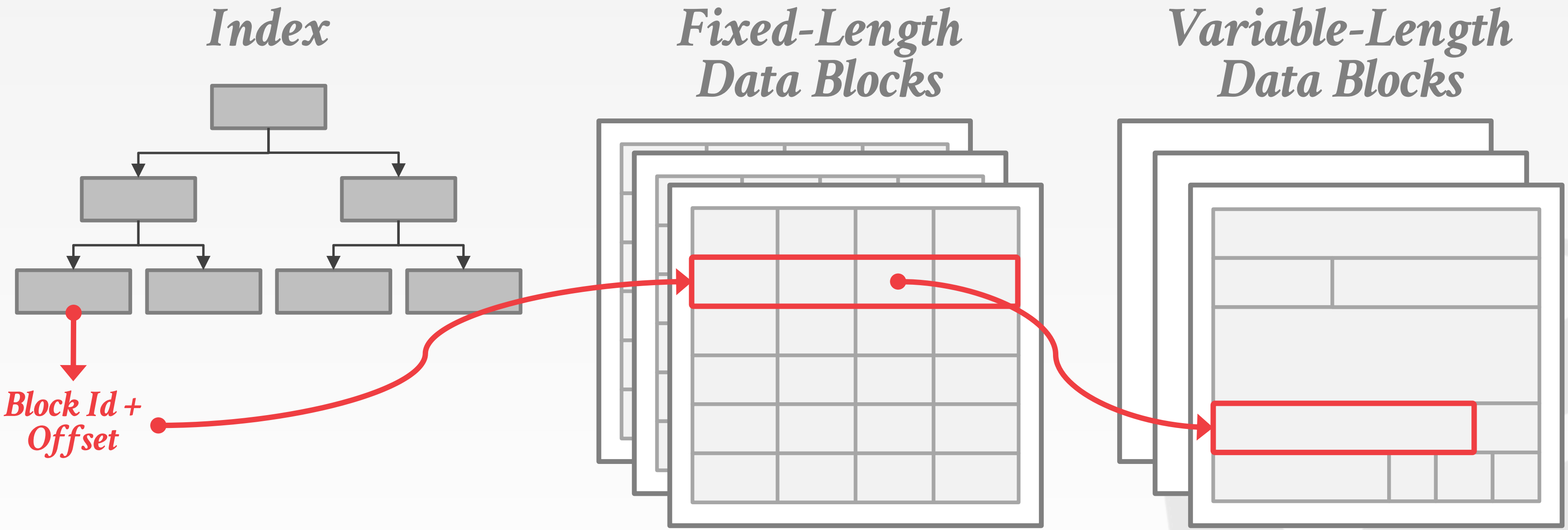
4:10 PM - 15 Feb 2018

97 Retweets 173 Likes

TODAY'S AGENDA

Type Representation
In-Memory Data Layout
Storage Models
System Catalogs

DATA ORGANIZATION



DATA ORGANIZATION

One can think of an in-memory database as just a large array of bytes.

→ The schema tells the DBMS how to convert the bytes into the appropriate type.

→ Each tuple is prefixed with a header that contains its meta-data.

Storing tuples with as fixed-length data makes it easy to compute the starting point of any tuple.

Mapping virtual memory pages to database pages.

MEMORY PAGES

OS maps physical pages to virtual memory pages.

The CPU's MMU maintains a TLB that contains the physical address of a virtual memory page.

→ The TLB resides in the CPU caches.
→ It can't obviously store every possible every possible entry for a large memory machine.

When you allocate a block of memory, the allocator keeps that it aligned to page boundaries.

TRANSPARENT HUGE PAGES

Maintain larger pages automatically (2MB to 1GB)
→ Each page has to be a contiguous blocks of memory.
→ Greatly reduces the # of TLB entries

With THP, the OS will to reorganize pages in the background to keep things compact.
→ Split larger pages into smaller pages.
→ Combine smaller pages into larger pages.
→ Can cause the DBMS process to stall on memory access.

Almost every DBMS says to disable this feature:
→ Oracle, MemSQL, NuoDB, MongoDB, Sybase IQ

Source: Alexandr Nikitin

DATA REPRESENTATION

INTEGER/BIGINT/SMALLINT/TINYINT
→ C/C++ Representation

FLOAT/REAL vs. **NUMERIC/DECIMAL**
→ IEEE-754 Standard / Fixed-point Decimals

VARCHAR/VARBINARY/TEXT/BLOB
→ Pointer to other location if type is ≥64-bits
→ Header with length and address to next location (if segmented), followed by data bytes.

TIME/DATE/TIMESTAMP
→ 32/64-bit integer of (micro)seconds since Unix epoch

VARIABLE PRECISION NUMBERS

Inexact, variable-precision numeric type that uses the "native" C/C++ types.

Store directly as specified by IEEE-754.

Typically faster than arbitrary precision numbers.
→ Example: **FLOAT**, **REAL**/**DOUBLE**

VARIABLE PRECISION NUMBERS

Rounding Example

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    float x = 0.1;
    float y = 0.2;
    printf("x+y = %.20f\n", x+y);
    printf("0.3 = %.20f\n", 0.3);
}
```

Output

```
x+y = 0.30000001192092895508
0.3 = 0.29999999999999998890
```

FIXED PRECISION NUMBERS

Numeric data types with arbitrary precision and scale. Used when round errors are unacceptable.

→ Example: **NUMERIC**, **DECIMAL**

Typically stored in a exact, variable-length binary representation with additional meta-data.

→ Like a **VARCHAR** but not stored as a string

Demo…

POSTGRES: NUMERIC

of Digits

Weight of 1st Digit

Scale Factor

Positive/Negative/NaN

Digit Storage

```
typedef unsigned char NumericDigit;
typedef struct {
  int ndigits;
  int weight;
  int scale;
  int sign;
  NumericDigit *digits;
} numeric;
```

POSTGRES: NUMERIC

of Digits

Weight of 1st Digit

Scale Factor

Positive/Negative/NaN

Digit Storage

```
typedef unsigned char NumericDigit;
typedef struct {
  int ndigits;
  int weight;
  int scale;
  int sign;
  NumericDigit *digits;
} numeric;
```

#

Weight of

Sca

Positive/Negat

Digi

```
NumericDigit;
```

```
;
```

```
/* ----------
 * add_var() -
 *
 *  Full version of add functionality on variable level (handling signs).
 *  result might point to one of the operands too without danger.
 * ----------
 */
int
PGTYPESnumeric_add(numeric *var1, numeric *var2, numeric *result)
{
    /*
     * Decide on the signs of the two variables what to do
     */
    if (var1->sign == NUMERIC_POS)
    {
        if (var2->sign == NUMERIC_POS)
        {
            /*
             * Both are positive result = +(ABS(var1) + ABS(var2))
             */
            if (add_abs(var1, var2, result) != 0)
                return -1;
            result->sign = NUMERIC_POS;
        }
        else
        {
            /*
             * var1 is positive, var2 is negative Must compare absolute values
             */
            switch (cmp_abs(var1, var2))
            {
                case 0:
                    /* ----------
                     * ABS(var1) == ABS(var2)
                     * result = ZERO
                     * ----------
                     */
                    zero_var(result);
                    result->rscale = Max(var1->rscale, var2->rscale);
                    result->dscale = Max(var1->dscale, var2->dscale);
                    break;

                case 1:
                    /* ----------
                     * ABS(var1) > ABS(var2)
                     * result = +(ABS(var1) - ABS(var2))
                     * ----------
                     */
                    if (sub_abs(var1, var2, result) != 0)
                        return -1;
                    result->sign = NUMERIC_POS;
                    break;

                case -1:
                    /* ----------
                     * ABS(var1) < ABS(var2)
                     * result = -(ABS(var2) - ABS(var1))
                     * ----------
```

DATA LAYOUT

```
CREATE TABLE AndySux (
  id INT PRIMARY KEY,
  value BIGINT
);
```

char[]

header	id	value

DATA LAYOUT

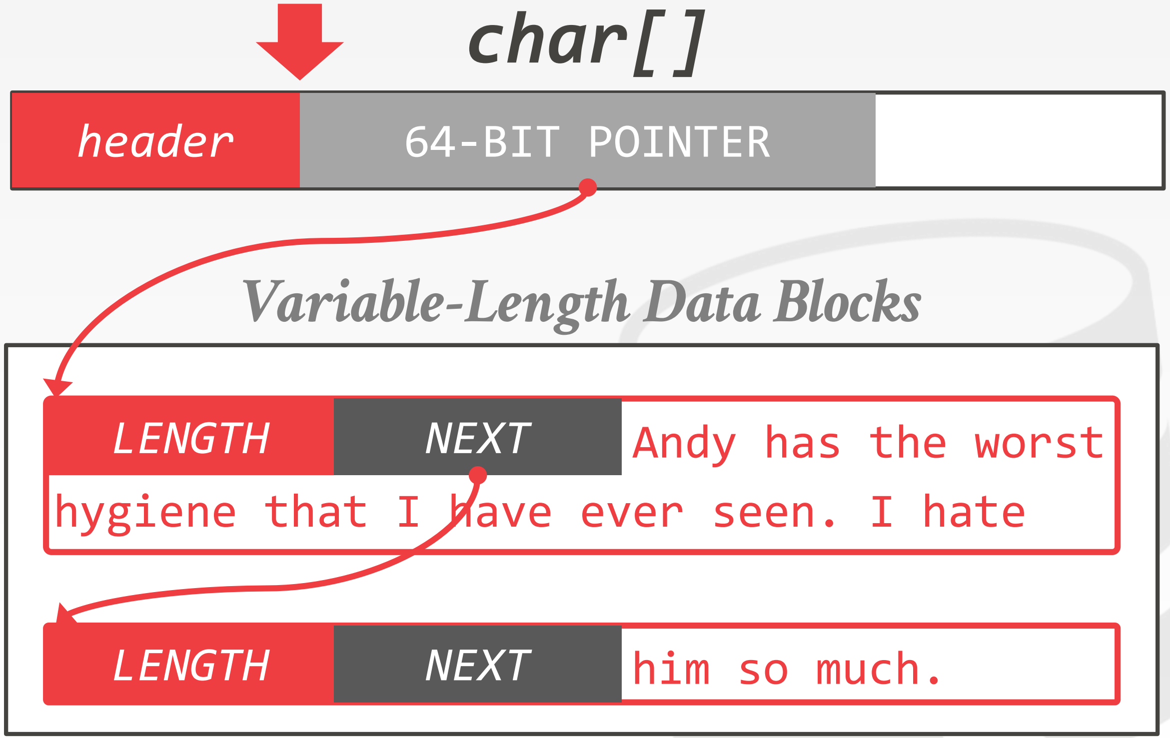
```sql
CREATE TABLE AndySux (
  id INT PRIMARY KEY,
  value BIGINT
);
```


```cpp
reinterpret_cast<int32_t*>(address)
```

VARIABLE-LENGTH FIELDS

```
CREATE TABLE AndySux (
  value VARCHAR(1024)
);
```

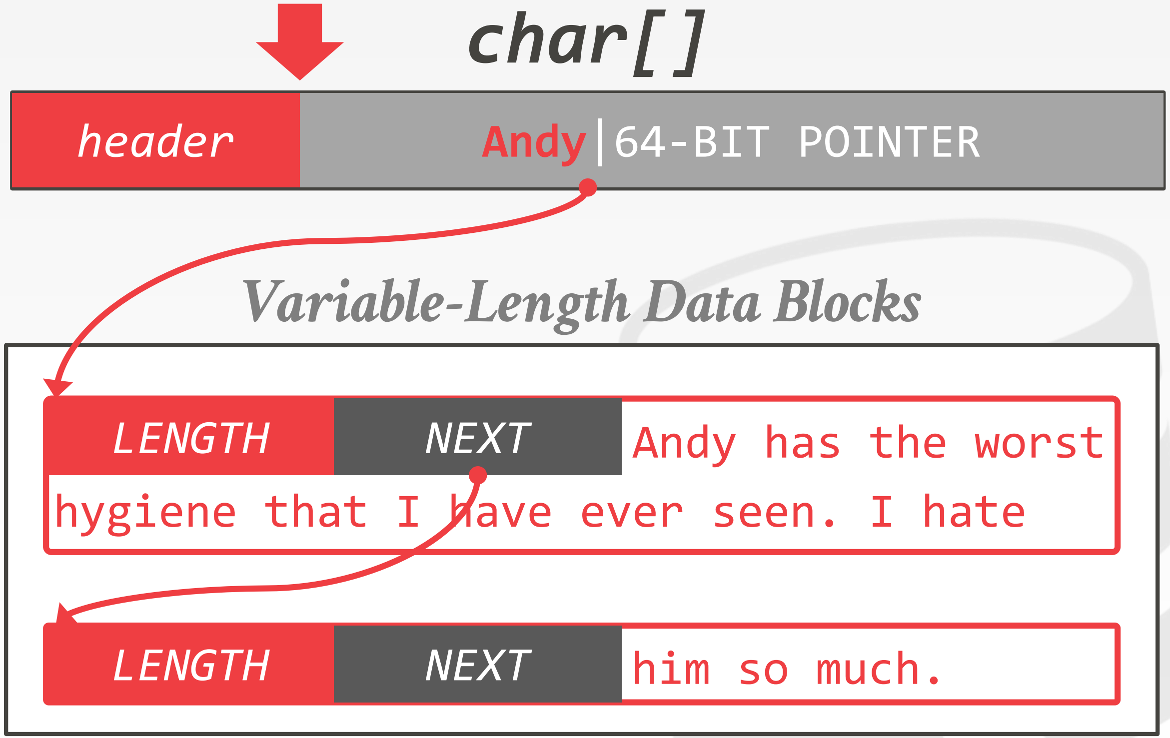
```
INSERT INTO AndySux
VALUES ("Andy has the worst
hygiene that I have ever
seen. I hate him so much.");
```



VARIABLE-LENGTH FIELDS

```
CREATE TABLE AndySux (
  value VARCHAR(1024)
);
```

```
INSERT INTO AndySux
VALUES ("Andy has the worst
hygiene that I have ever
seen. I hate him so much.");
```



NULL DATA TYPES

Choice #1: Special Values

→ Designate a value to represent **NULL** for a particular data type (e.g., **INT32_MIN**).

Choice #2: Null Column Bitmap Header

→ Store a bitmap in the tuple header that specifies what attributes are null.

Choice #3: Per Attribute Null Flag

→ Store a flag that marks that a value is null.

→ Have to use more space than just a single bit because this messes up with word alignment.

NULL DATA TYPES

Integer Numbers

Data Type	Size	Size (Not Null)	Synonyms	Min Value	Max Value
BOOL	2 bytes	1 byte	BOOLEAN	0	1
BIT	9 bytes	8 bytes			
TINYINT	2 bytes	1 byte		-128	127
SMALLINT	4 bytes	2 bytes		-32768	32767
MEDIUMINT	4 bytes	3 bytes		-8388608	8388607
INT	8 bytes	4 bytes	INTEGER	-2147483648	2147483647
BIGINT	12 bytes	8 bytes		-2 ** 63	(2 ** 63) - 1

messes up with word alignment.

DISCLAIMER

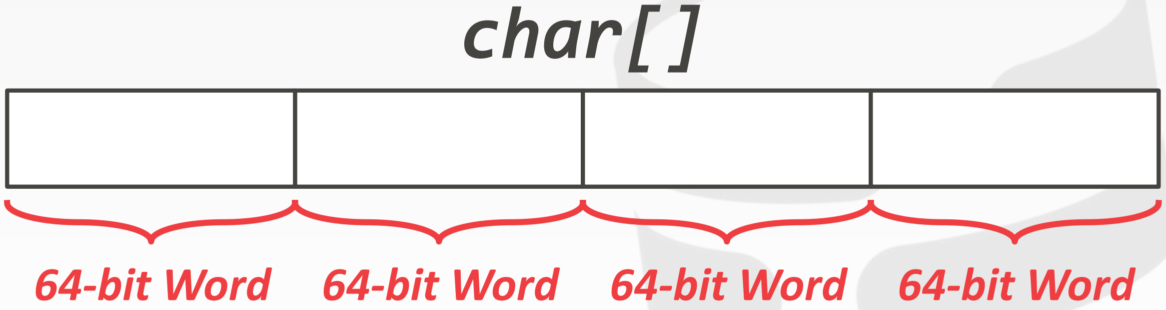
The truth is that you only need to worry about word-alignment for cache lines (e.g., 64 bytes).

I'm going to show you the basic idea using 64-bit words since it's easier to see…

WORD-ALIGNED TUPLES

All attributes in a tuple must be word aligned to enable the CPU to access it without any unexpected behavior or additional work.

```
CREATE TABLE AndySux (
  id INT PRIMARY KEY,
  cdate TIMESTAMP,
  color CHAR(2),
  zipcode INT
);
```

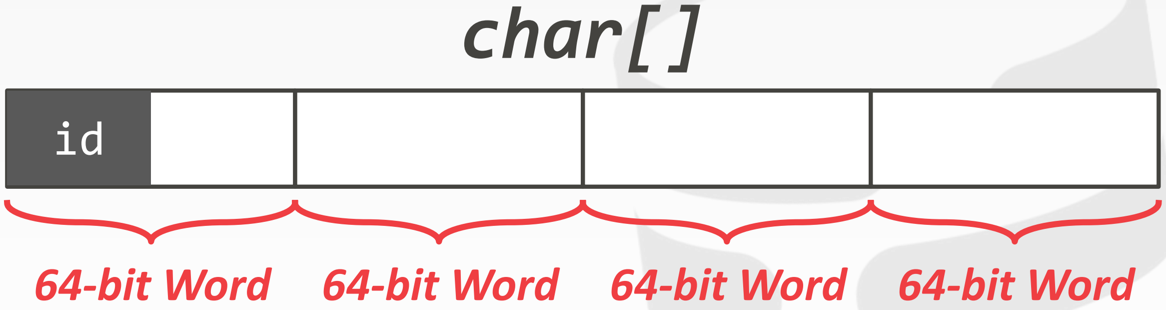


WORD-ALIGNED TUPLES

All attributes in a tuple must be word aligned to enable the CPU to access it without any unexpected behavior or additional work.

32-bits

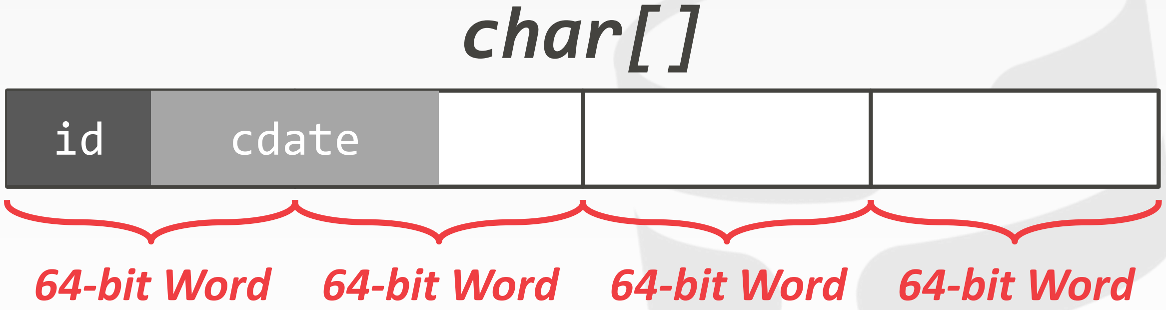
```
CREATE TABLE AndySux (
  id INT PRIMARY KEY,
  cdate TIMESTAMP,
  color CHAR(2),
  zipcode INT
);
```



WORD-ALIGNED TUPLES

All attributes in a tuple must be word aligned to enable the CPU to access it without any unexpected behavior or additional work.

```
         CREATE TABLE AndySux (
32-bits    id INT PRIMARY KEY,
64-bits    cdate TIMESTAMP,
           color CHAR(2),
           zipcode INT
         );
```

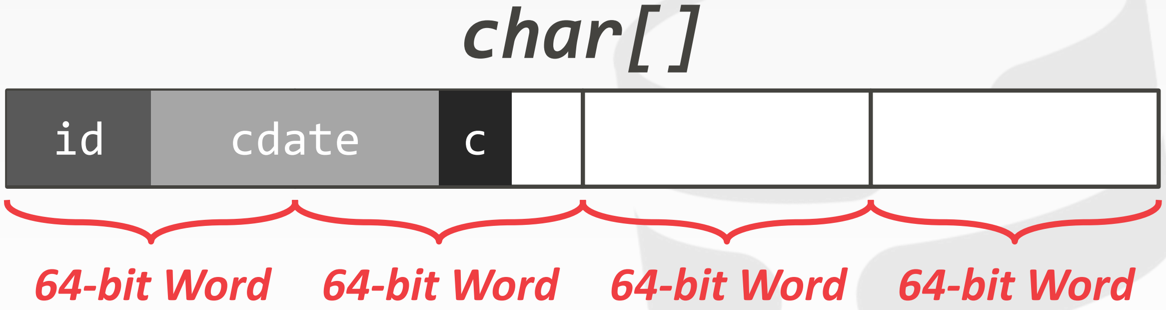


WORD-ALIGNED TUPLES

All attributes in a tuple must be word aligned to enable the CPU to access it without any unexpected behavior or additional work.

```
CREATE TABLE AndySux (
  id INT PRIMARY KEY,
  cdate TIMESTAMP,
  color CHAR(2),
  zipcode INT
);
```

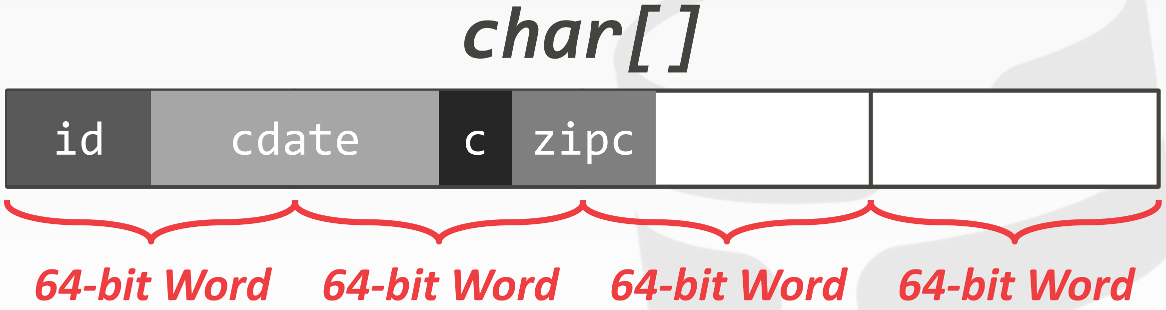
32-bits
64-bits
16-bits



WORD-ALIGNED TUPLES

All attributes in a tuple must be word aligned to enable the CPU to access it without any unexpected behavior or additional work.

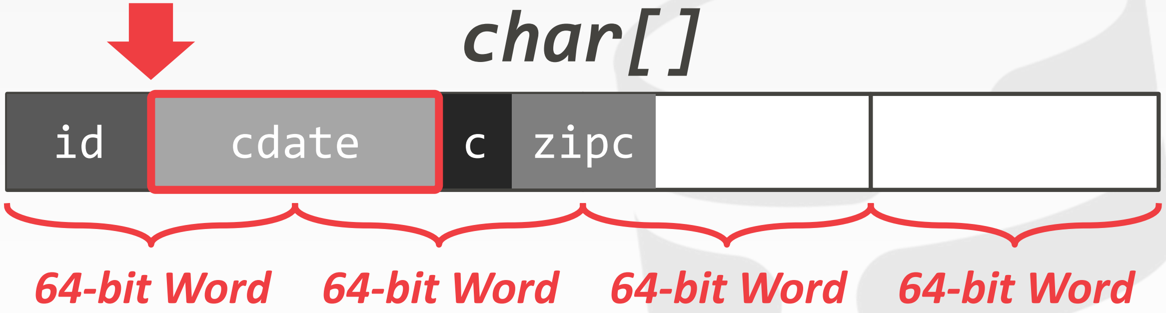
```
         CREATE TABLE AndySux (
32-bits    id INT PRIMARY KEY,
64-bits    cdate TIMESTAMP,
16-bits    color CHAR(2),
32-bits    zipcode INT
         );
```



WORD-ALIGNED TUPLES

All attributes in a tuple must be word aligned to enable the CPU to access it without any unexpected behavior or additional work.

```
CREATE TABLE AndySux (
32-bits  id INT PRIMARY KEY,
64-bits  cdate TIMESTAMP,
16-bits  color CHAR(2),
32-bits  zipcode INT
);
```



WORD-ALIGNED TUPLES

If the CPU fetches a 64-bit value that is not word-aligned, it has three choices:

→ Execute two reads to load the appropriate parts of the data word and reassemble them.

→ Read some unexpected combination of bytes assembled into a 64-bit word.

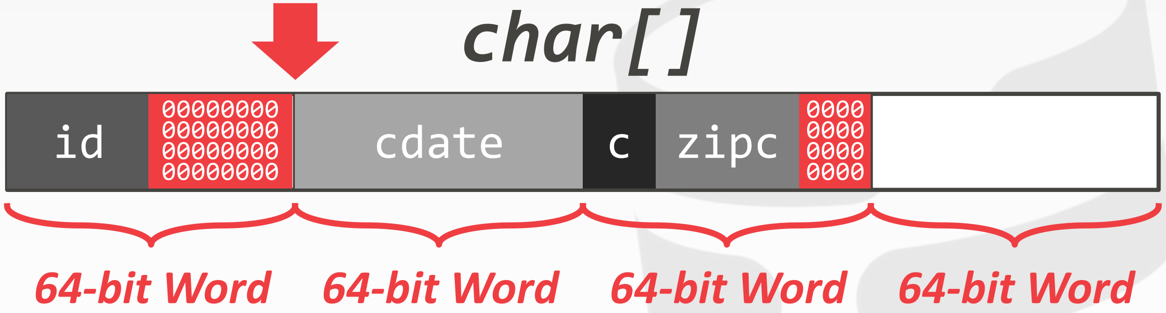
→ Throw an exception

Source: Levente Kurusa

WORD-ALIGNED TUPLES

All attributes in a tuple must be word aligned to enable the CPU to access it without any unexpected behavior or additional work.

```
        CREATE TABLE AndySux (
32-bits id INT PRIMARY KEY,
64-bits cdate TIMESTAMP,
16-bits color CHAR(2),
32-bits zipcode INT
        );
```



STORAGE MODELS

N-ary Storage Model (NSM)

Decomposition Storage Model (DSM)

Hybrid Storage Model

N-ARY STORAGE MODEL (NSM)

The DBMS stores all of the attributes for a single tuple contiguously.

Ideal for OLTP workloads where txns tend to operate only on an individual entity and insert-heavy workloads.

Use the tuple-at-a-time iterator model.

NSM PHYSICAL STORAGE

Choice #1: Heap-Organized Tables

→ Tuples are stored in blocks called a heap.

→ The heap does not necessarily define an order.

Choice #2: Index-Organized Tables

→ Tuples are stored in the primary key index itself.

→ Not quite the same as a clustered index.

N-ARY STORAGE MODEL (NSM)

Advantages

→ Fast inserts, updates, and deletes.
→ Good for queries that need the entire tuple.
→ Can use index-oriented physical storage.

Disadvantages

→ Not good for scanning large portions of the table and/or a subset of the attributes.

DECOMPOSITION STORAGE MODEL (DSM)

The DBMS stores a single attribute for all tuples contiguously in a block of data.

→ Sometimes also called **vertical partitioning**.

Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table's attributes.

Use the vector-at-a-time iterator model.

DECOMPOSITION STORAGE MODEL (DSM)

1970s: Cantor DBMS

1980s: DSM Proposal

1990s: SybaseIQ (in-memory only)

2000s: Vertica, Vectorwise, MonetDB

2010s: "The Big Three"
Cloudera Impala, Amazon Redshift,
SAP HANA, MemSQL

TUPLE IDENTIFICATION

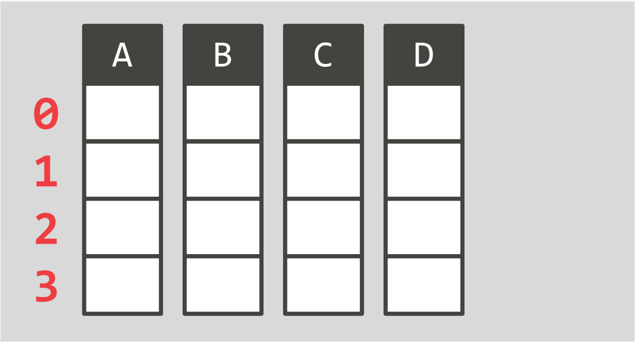
Choice #1: Fixed-length Offsets

→ Each value is the same length for an attribute.

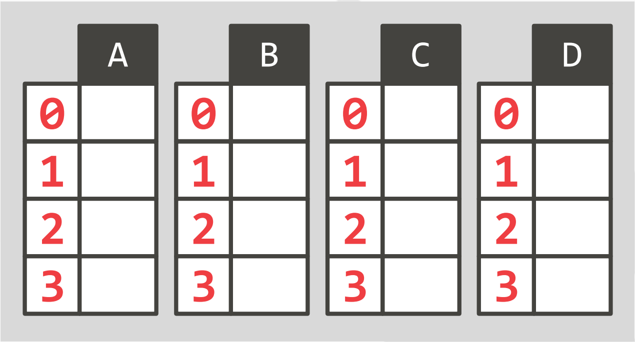
Choice #2: Embedded Tuple Ids

→ Each value is stored with its tuple id in a column.

Offsets



Embedded Ids



DECOMPOSITION STORAGE MODEL (DSM)

Advantages

→ Reduces the amount wasted work because the DBMS only reads the data that it needs.
→ Better compression.

Disadvantages

→ Slow for point queries, inserts, updates, and deletes because of tuple splitting/stitching.

OBSERVATION

Data is "hot" when first entered into database

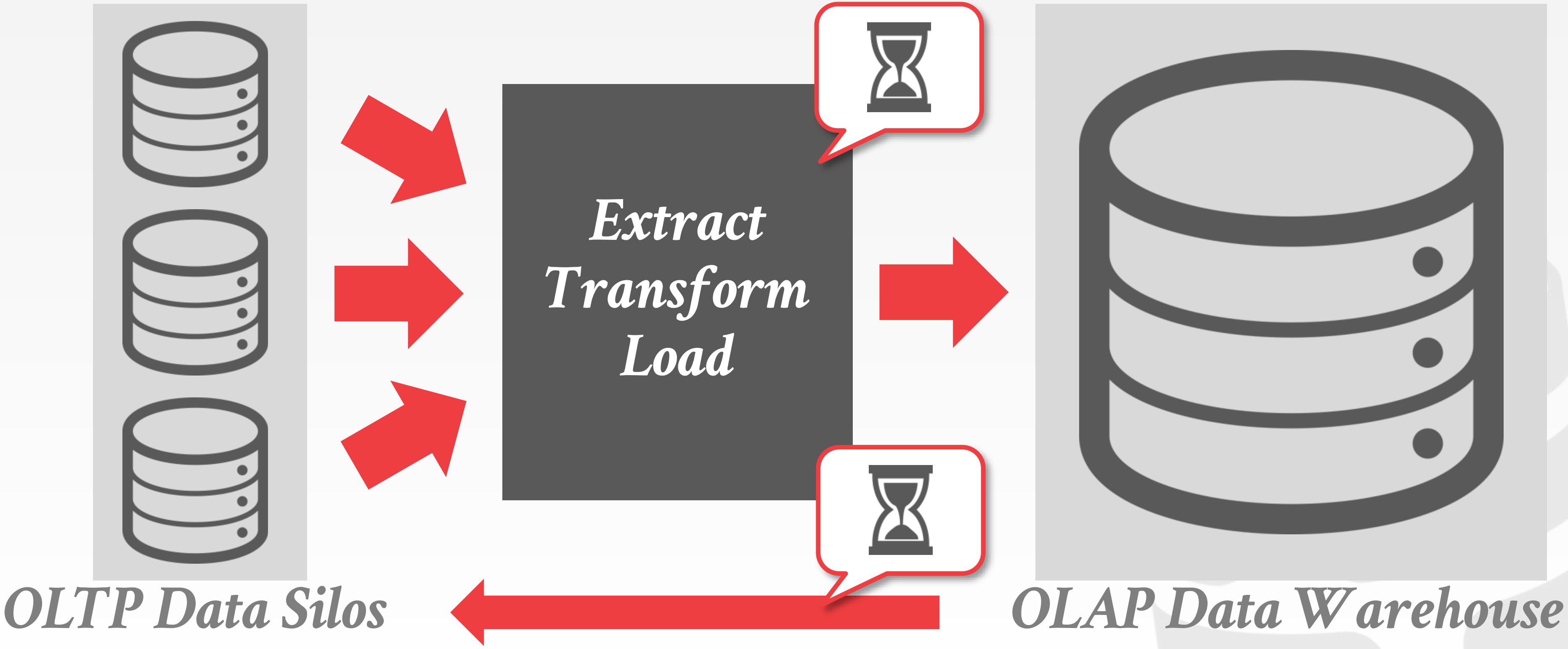
→ A newly inserted tuple is more likely to be updated again the near future.

As a tuple ages, it is updated less frequently.

→ At some point, a tuple is only accessed in read-only queries along with other tuples.

What if we want to use this data to make decisions that affect new txns?

BIFURCATED ENVIRONMENT



HYBRID STORAGE MODEL

Single logical database instance that uses different storage models for hot and cold data.

Store new data in NSM for fast OLTP
Migrate data to DSM for more efficient OLAP

HYBRID STORAGE MODEL

Choice #1: Separate Execution Engines

→ Use separate execution engines that are optimized for either NSM or DSM databases.

Choice #2: Single, Flexible Architecture

→ Use single execution engine that is able to efficiently operate on both NSM and DSM databases.

SEPARATE EXECUTION ENGINES

Run separate "internal" DBMSs that each only operate on DSM or NSM data.

→ Need to combine query results from both engines to appear as a single logical database to the application.
→ Have to use a synchronization method (e.g., 2PC) if a txn spans execution engines.

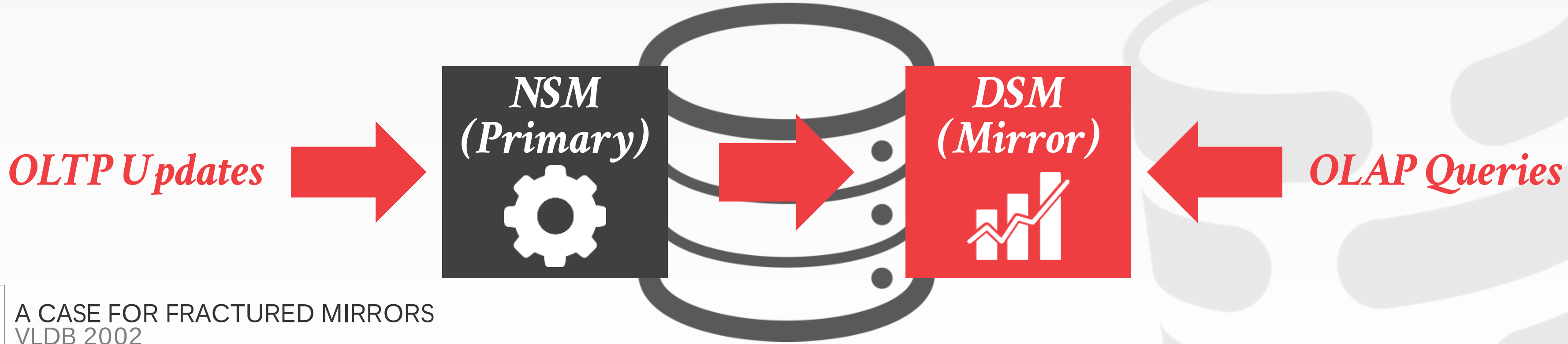
Two approaches to do this:

→ **Fractured Mirrors** (Oracle, IBM)
→ **Delta Store** (SAP HANA)

FRACTURED MIRRORS

Store a second copy of the database in a DSM layout that is automatically updated.

→ All updates are first entered in NSM then eventually copied into DSM mirror.

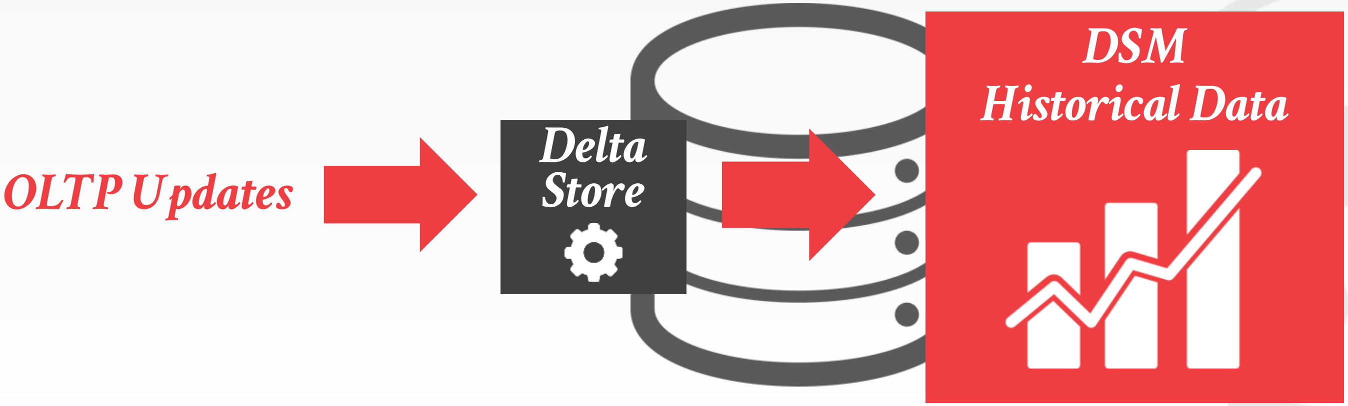


A CASE FOR FRACTURED MIRRORS
VLDB 2002

DELTA STORE

Stage updates to the database in an NSM table.

A background thread migrates updates from delta store and applies them to DSM data.



CATEGORIZING DATA

Choice #1: Manual Approach

→ DBA specifies what tables should be stored as DSM.

Choice #2: Off-line Approach

→ DBMS monitors access logs offline and then makes decision about what data to move to DSM.

Choice #3: On-line Approach

→ DBMS tracks access patterns at runtime and then makes decision about what data to move to DSM.

PELOTON ADAPTIVE STORAGE

Employ a single execution engine architecture that is able to operate on both NSM and DSM data.

→ Don't need to store two copies of the database.

→ Don't need to sync multiple database segments.

Note that a DBMS can still use the delta-store approach with this single-engine architecture.

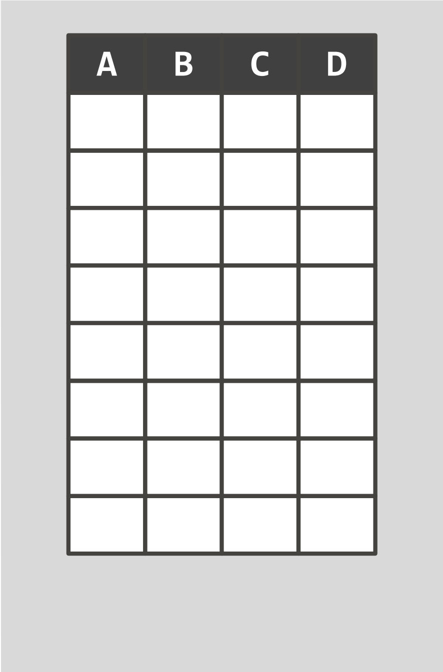
BRIDGING THE ARCHIPELAGO BETWEEN ROW-STORES AND COLUMN-STORES FOR HYBRID WORKLOADS
SIGMOD 2016

PELOTON ADAPTIVE STORAGE

```
UPDATE AndySux
   SET A = 123,
       B = 456,
       C = 789
 WHERE D = “xxx”
```

```
SELECT AVG(B)
  FROM AndySux
 WHERE C = “yyy”
```

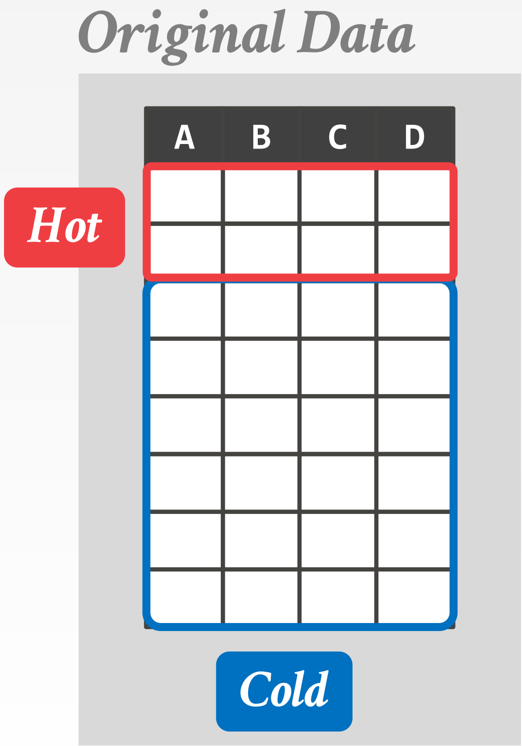
Original Data



PELOTON ADAPTIVE STORAGE

```
UPDATE AndySux
   SET A = 123,
       B = 456,
       C = 789
 WHERE D = "xxx"
```

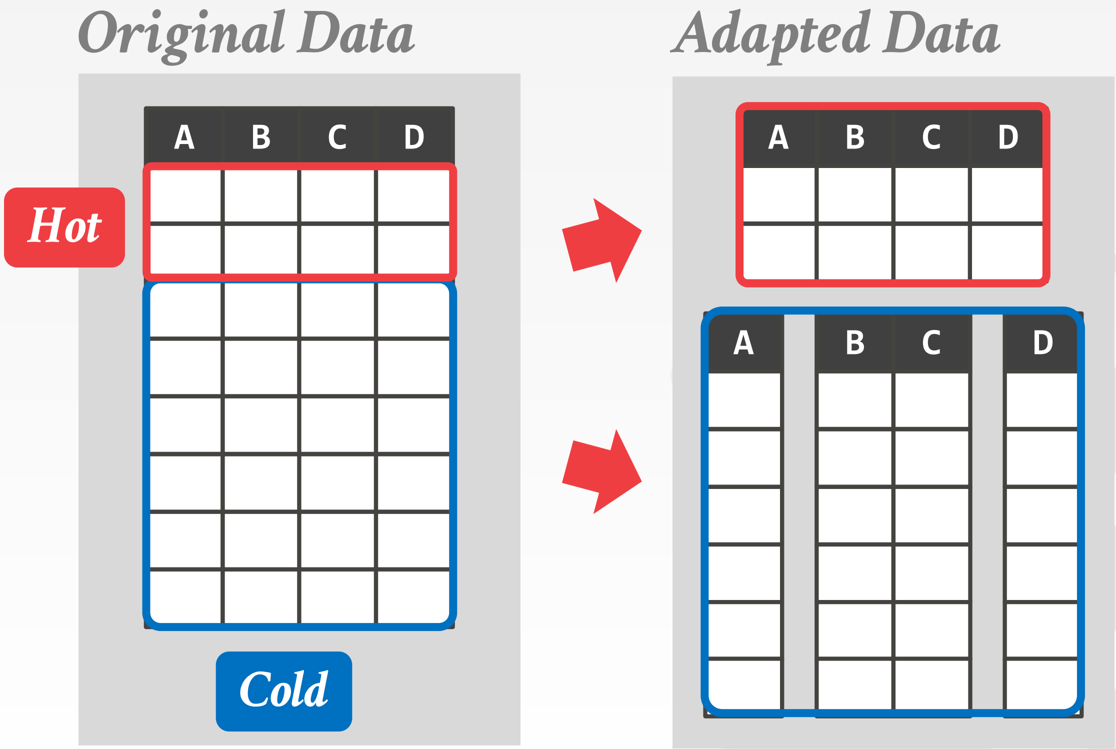
```
SELECT AVG(B)
  FROM AndySux
 WHERE C = "yyy"
```



PELOTON ADAPTIVE STORAGE

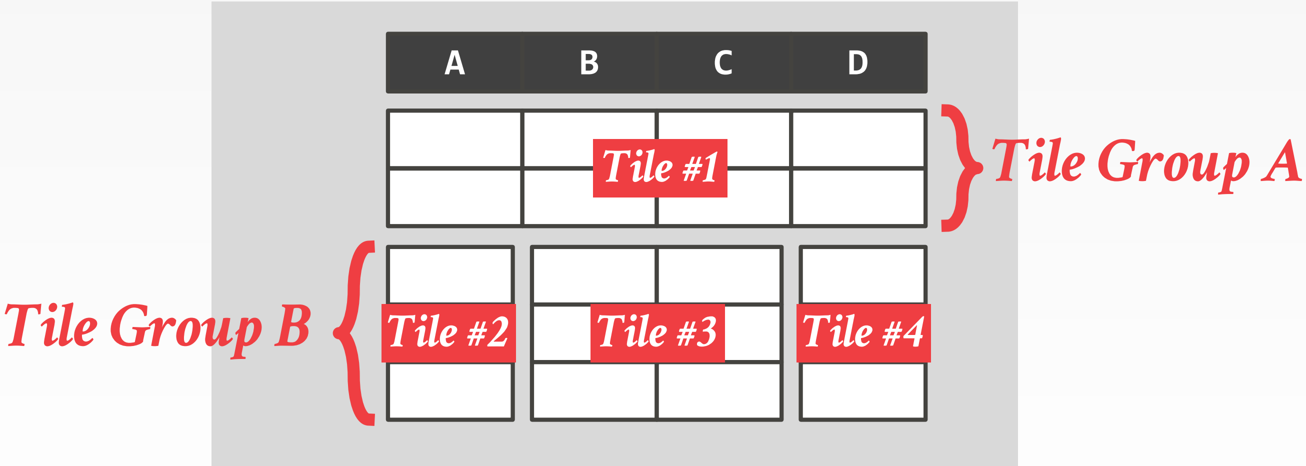
```
UPDATE AndySux
   SET A = 123,
       B = 456,
       C = 789
 WHERE D = “xxx”
```

```
SELECT AVG(B)
  FROM AndySux
 WHERE C = “yyy”
```



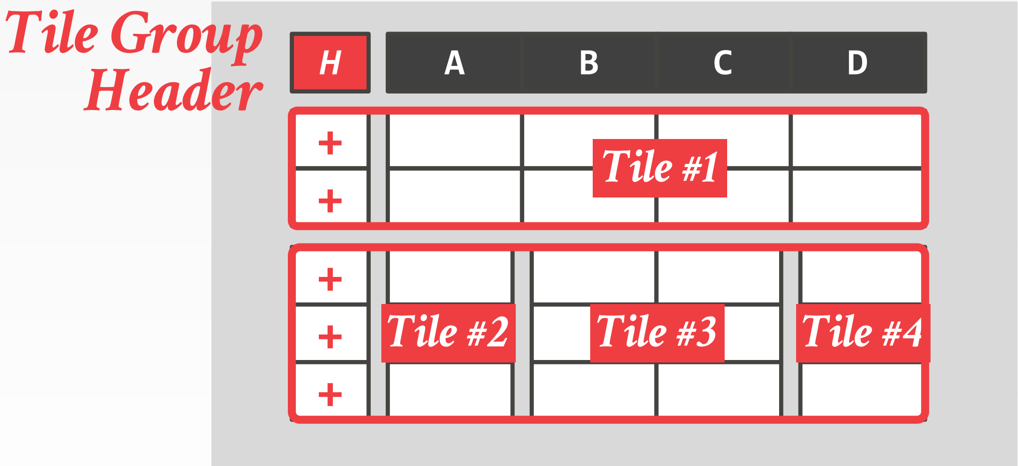
TILE ARCHITECTURE

Introduce an indirection layer that abstracts the true layout of tuples from query operators.



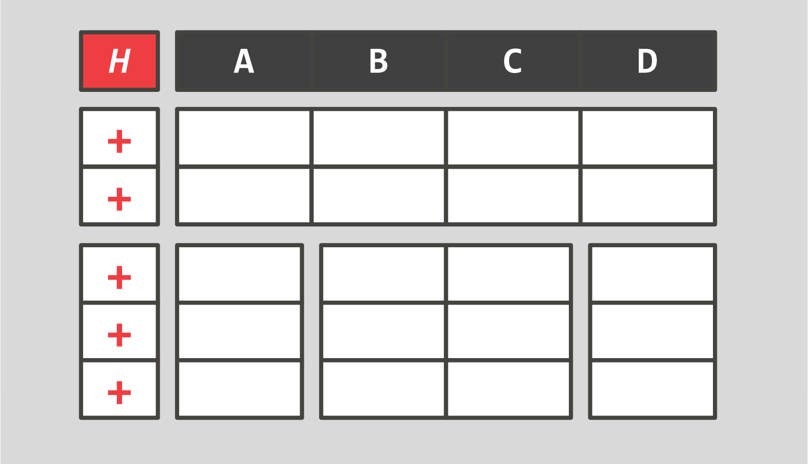
TILE ARCHITECTURE

Introduce an indirection layer that abstracts the true layout of tuples from query operators.

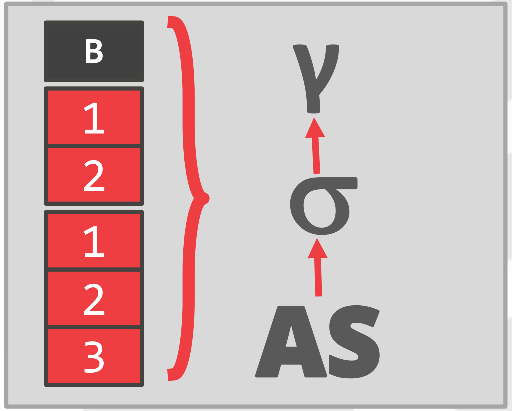


TILE ARCHITECTURE

Introduce an indirection layer that abstracts the true layout of tuples from query operators.



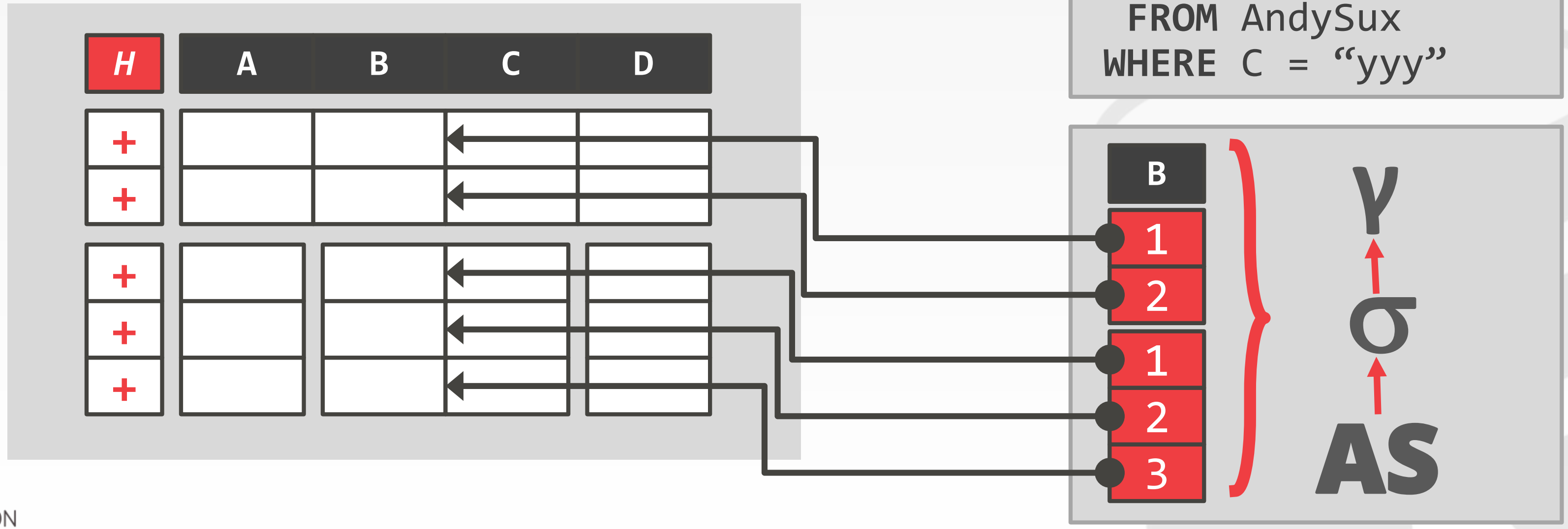

```
SELECT AVG(B)
  FROM AndySux
 WHERE C = "yyy"
```



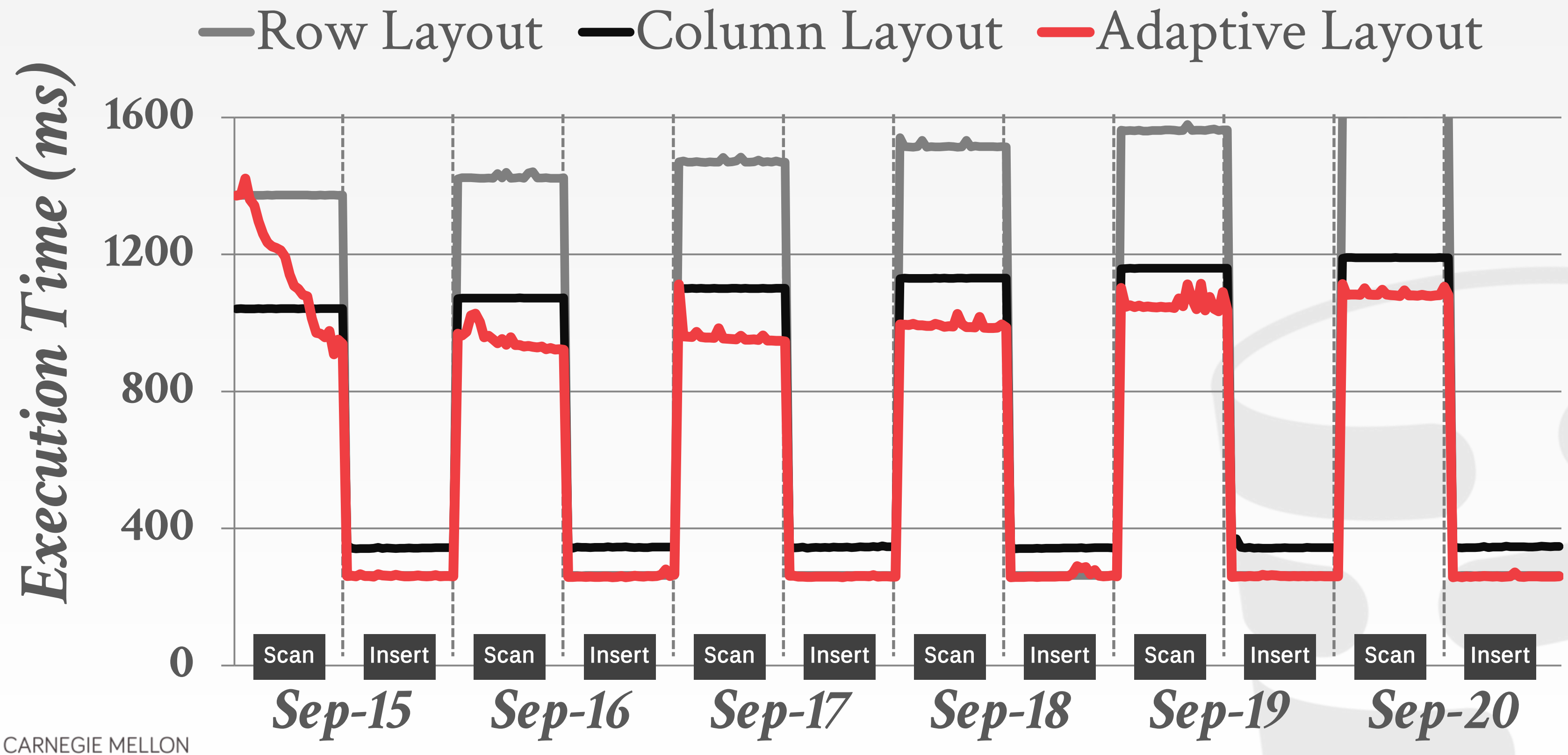
TILE ARCHITECTURE

Introduce an indirection layer that abstracts the true layout of tuples from query operators.

```
SELECT AVG(B)
  FROM AndySux
 WHERE C = “yyy”
```



PELOTON ADAPTIVE STORAGE



PARTING THOUGHTS

A flexible architecture that supports a hybrid storage model is the next major trend in DBMSs

→ This will enable relational DBMSs to support all database workloads except for matrices in machine learning.