Lecture #09 – Storage Models & Data Layout

Andy Pavlo // Carnegie Mellon University // Spring 2016

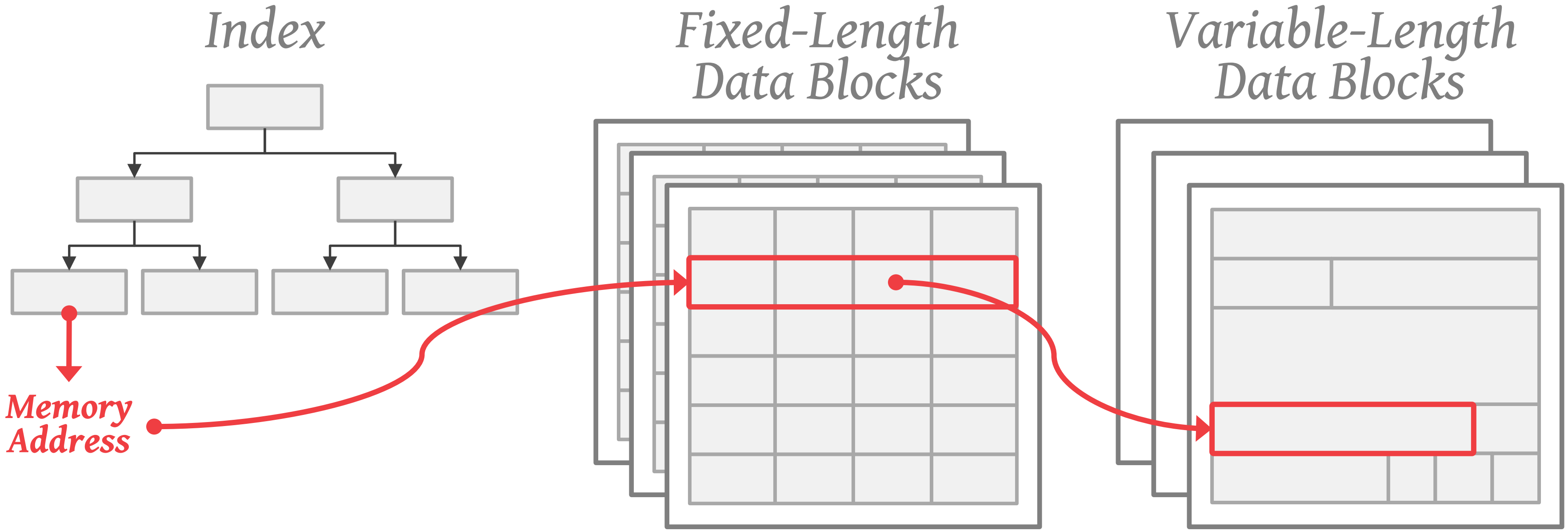
TODAY'S AGENDA

In-Memory Data Layout

Storage Models

Project #2: Performance Profiling

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 just their fixed-length data makes it easy to compute the starting point of any tuple.

DATA REPRESENTATION

INTEGER/BIGINT/SMALLINT/TINYINT

→ C/C++ Representation

NUMERIC

→ IEEE-754 Standard

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

DATA REPRESENTATION

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

char[]

DATA REPRESENTATION

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

DATA REPRESENTATION

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

DATA REPRESENTATION

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

DATA REPRESENTATION

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


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

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

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

this messes up with word alignment.

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.

NOTICE

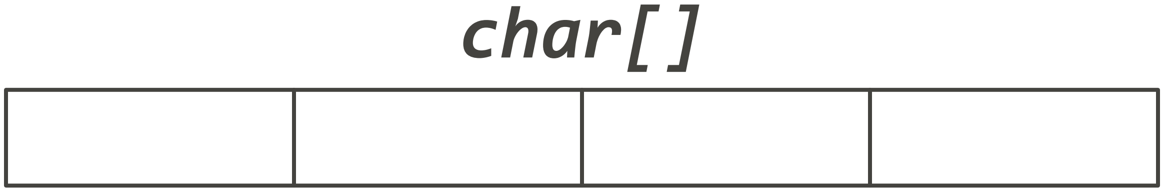
The truth is that you only need to worry about word-alignment for cache lines (e.g., 64 bytes).

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.

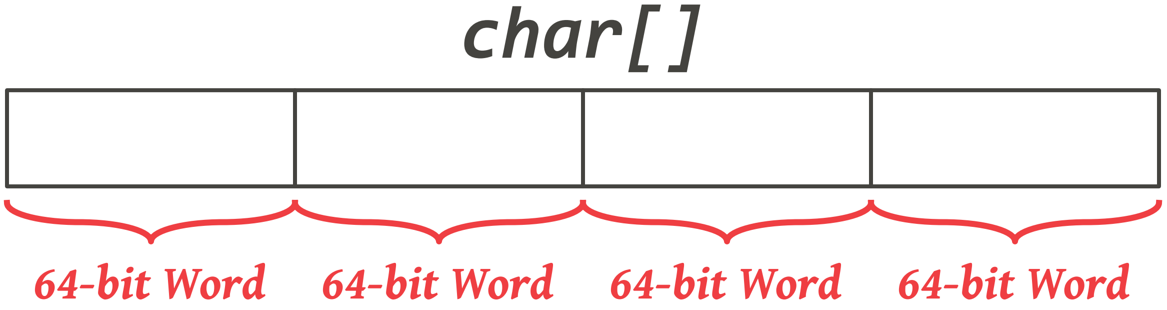
```
CREATE TABLE JoySux (
  id INT PRIMARY KEY,
  cdate TIMESTAMP,
  color CHAR(2),
  zipcode INT
);
```



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 JoySux (
  id INT PRIMARY KEY,
  cdate TIMESTAMP,
  color CHAR(2),
  zipcode INT
);
```

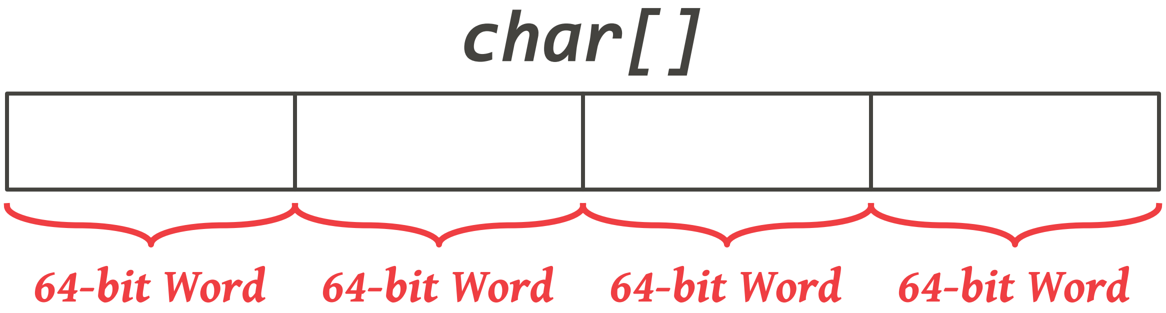


WORD-ALIGNED TUPLES

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 JoySux (
  id INT PRIMARY KEY,
  cdate TIMESTAMP,
  color CHAR(2),
  zipcode INT
);
```

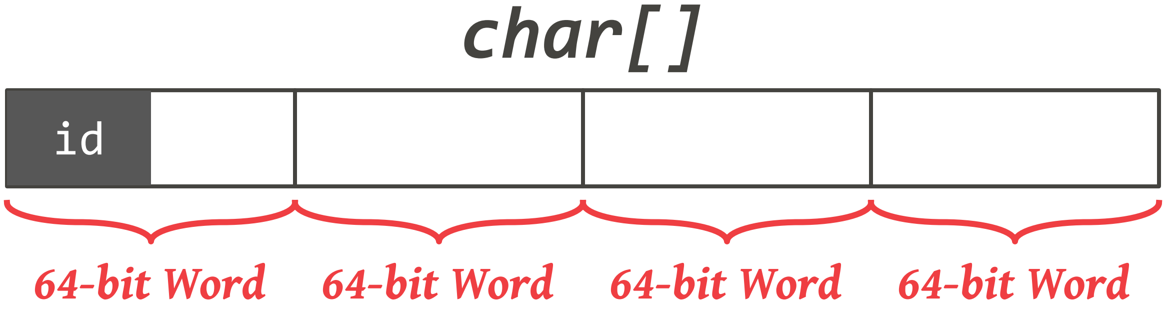


WORD-ALIGNED TUPLES

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

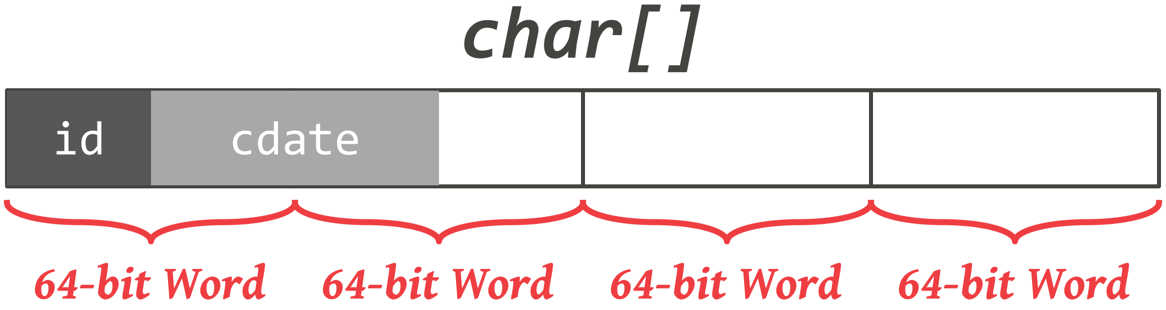
```
CREATE TABLE JoySux (
  id INT PRIMARY KEY,
  cdate TIMESTAMP,
  color CHAR(2),
  zipcode INT
);
```



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.

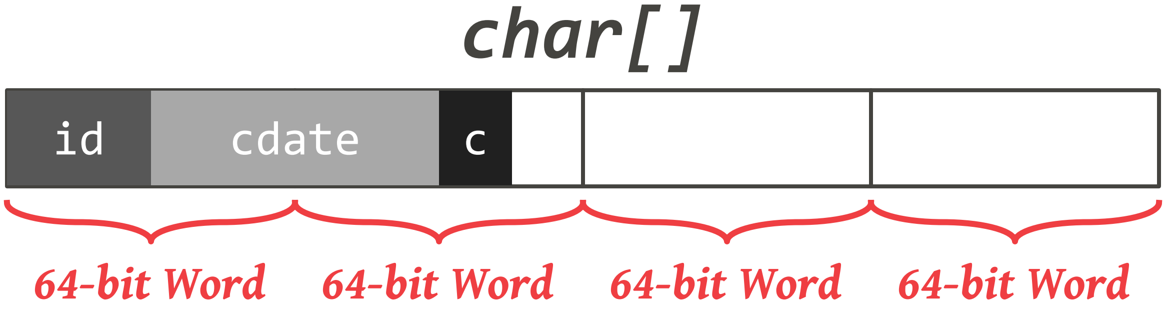
```
CREATE TABLE JoySux (
  id INT PRIMARY KEY,      -- 32-bits
  cdate TIMESTAMP,         -- 64-bits
  color CHAR(2),
  zipcode INT
);
```



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.

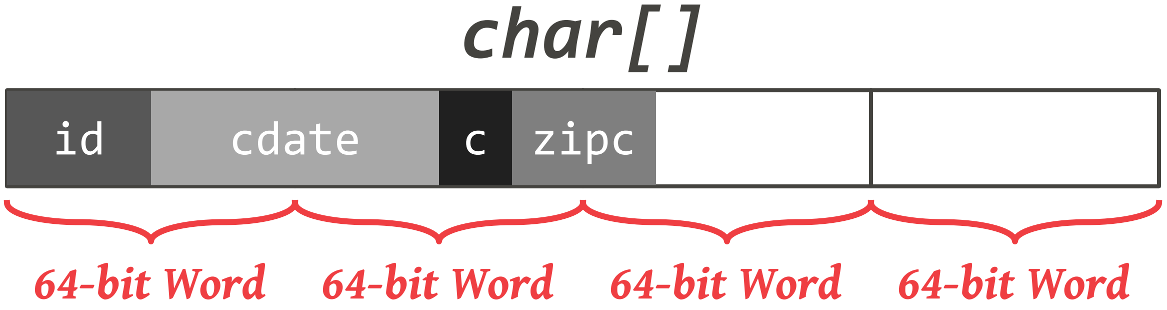
```
         CREATE TABLE JoySux (
32-bits    id INT PRIMARY KEY,
64-bits    cdate TIMESTAMP,
16-bits    color CHAR(2),
           zipcode INT
         );
```



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

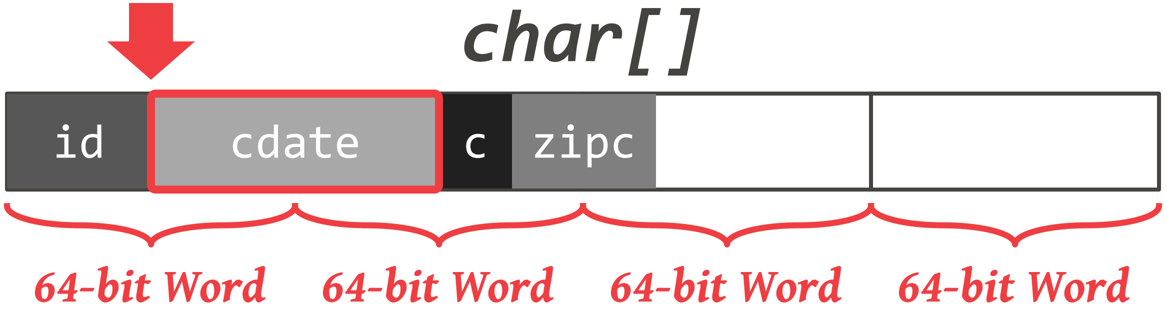


WORD-ALIGNED TUPLES

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

32-bits
64-bits
16-bits
32-bits

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



WORD-ALIGNED TUPLES

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

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

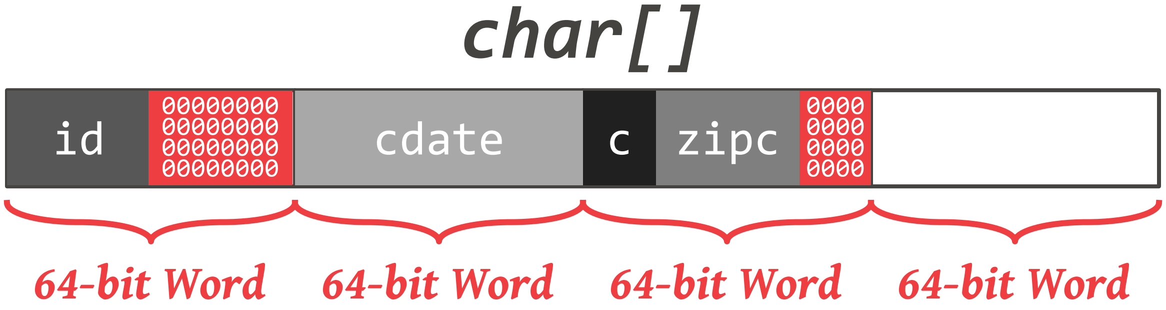
→ Read some unexpected combination of bytes assembled into a 64-bit word.

→ Throw an exception

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.

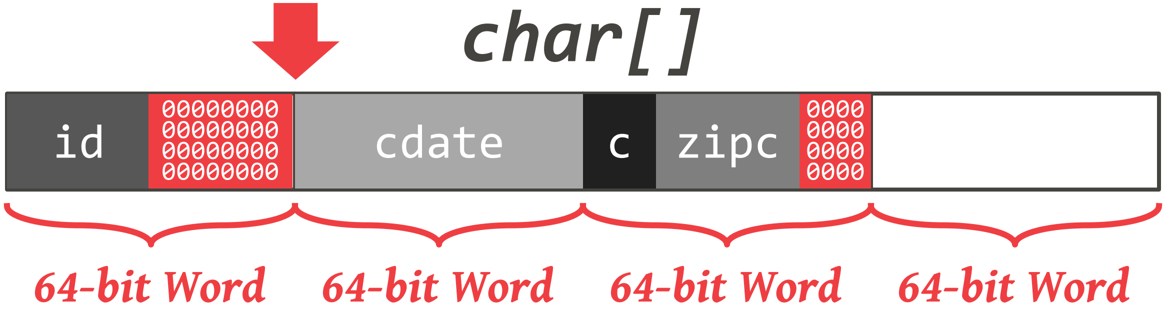
```
          CREATE TABLE JoySux (
32-bits     id INT PRIMARY KEY,
64-bits     cdate TIMESTAMP,
16-bits     color CHAR(2),
32-bits     zipcode INT
          );
```



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 JoySux (
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 index itself.

→ Not quite the same as a clustered index.

CLUSTERED INDEXES

The table is stored in the sort order specified by the primary key.

→ Can be either heap- or index-organized storage.

Some DBMSs always use a clustered index.

→ If a table doesn't include a pkey, the DBMS will automatically make a hidden row id pkey.

Other DBMSs cannot use them at all.

→ A clustered index is non-practical in a MVCC DBMS using the **Insert Method**.

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

CLUSTERED INDEXES

Some columnar DBMSs store data in sorted order to maximize compression.

→ Bitmap indexes with RLE from last class

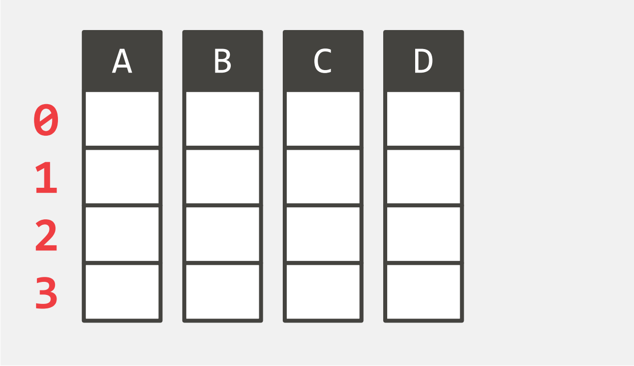
Vertica does not even use indexes because all columns are sorted.

TUPLE IDENTIFICATION

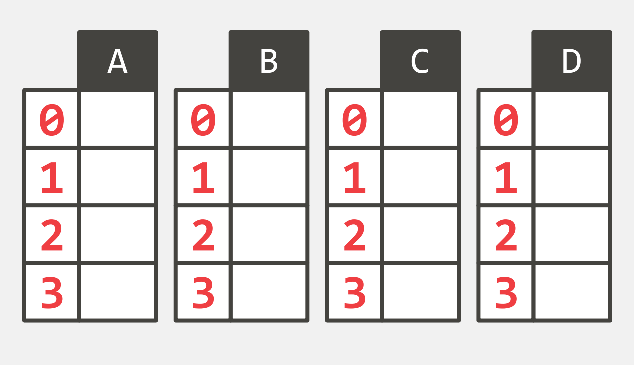
Choice #1: Fixed-length Offsets
→ Each value is the same length for an attribute.

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 (last lecture).

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.

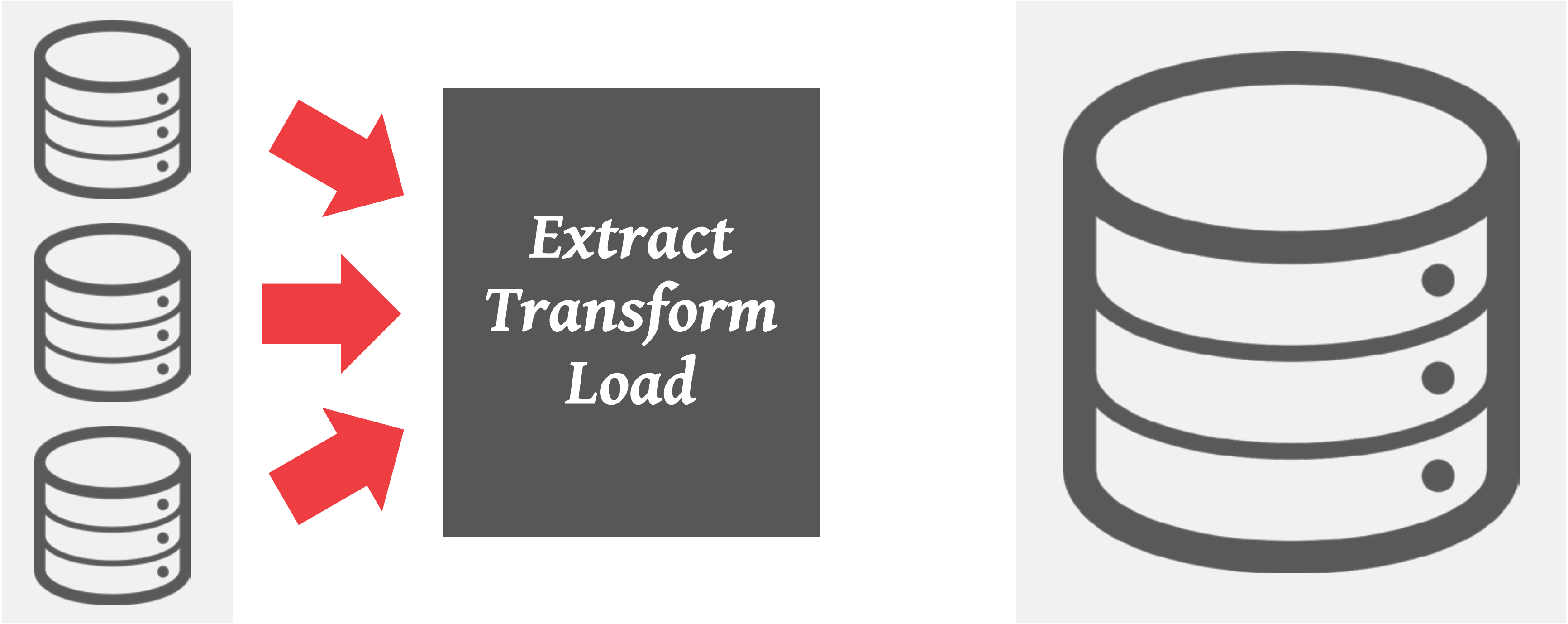
What if we want to use this data to make decisions that affect new txns?

BIFURCATED ENVIRONMENT

OLTP Data Silos

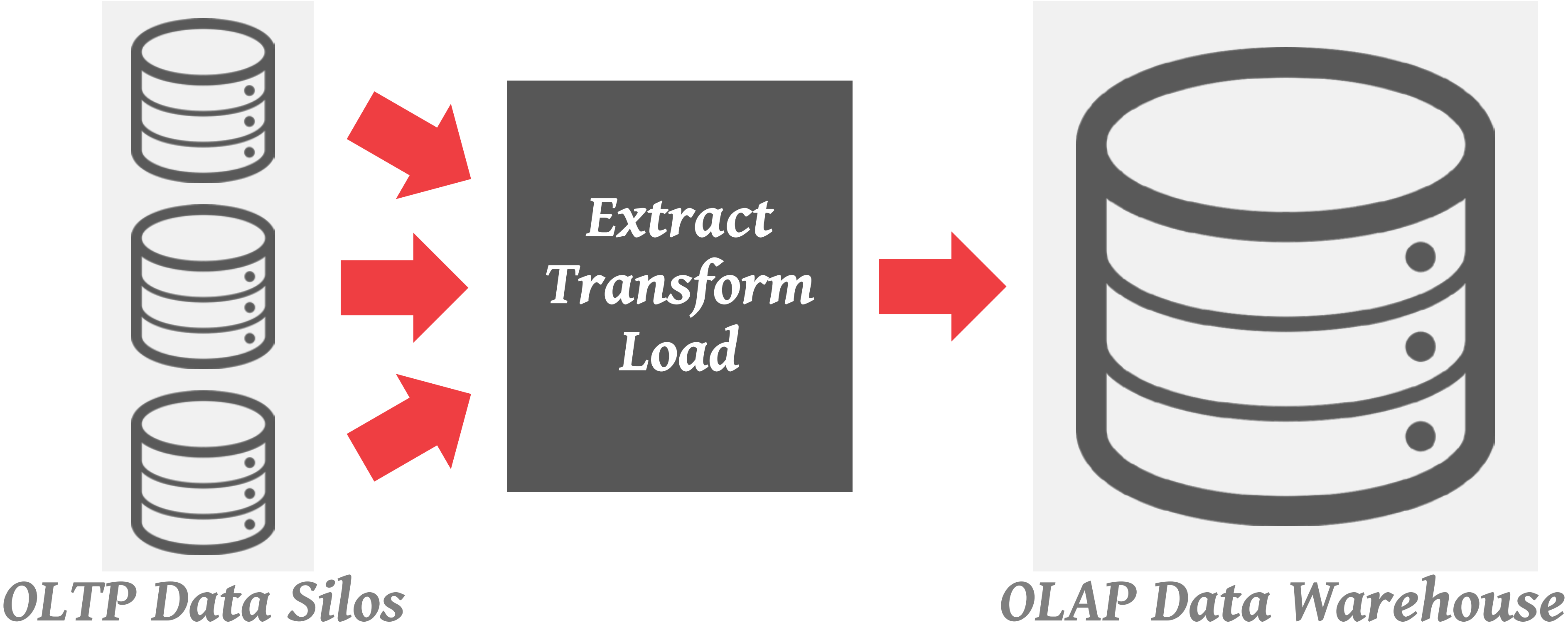
OLAP Data Warehouse

BIFURCATED ENVIRONMENT

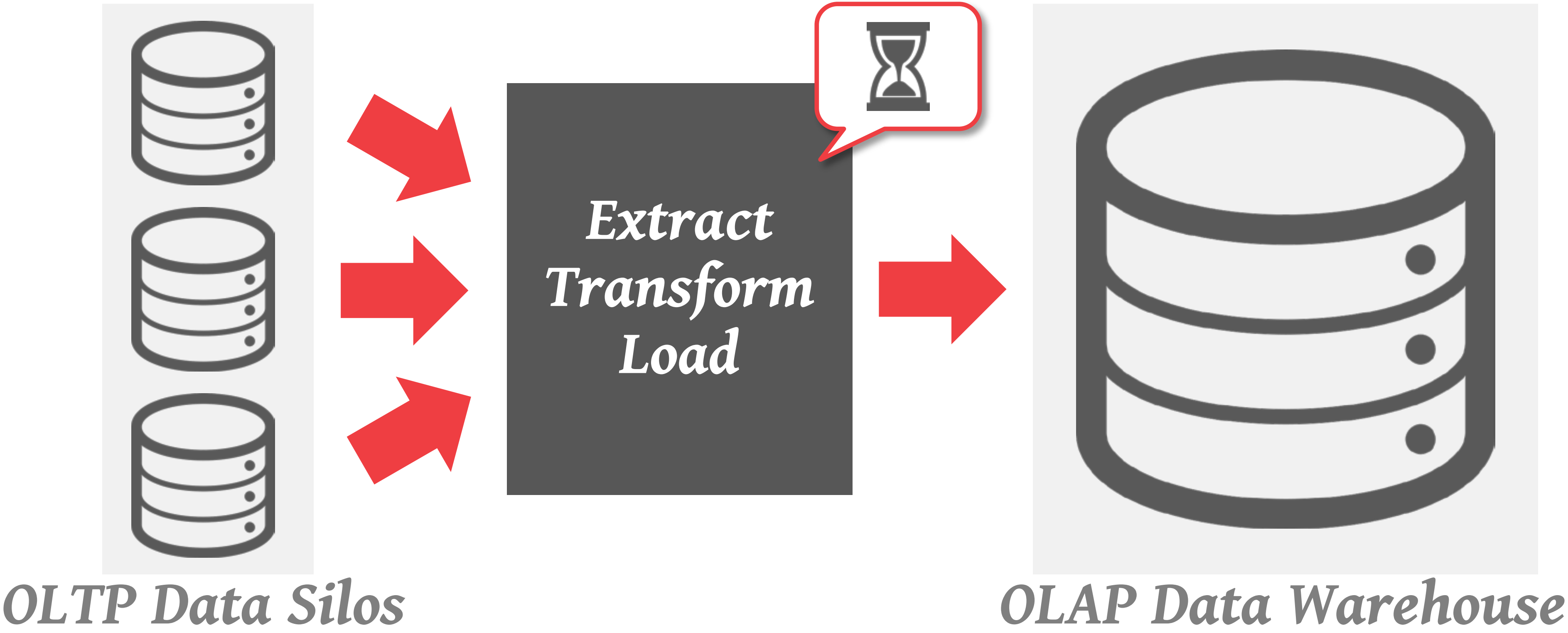


OLTP Data Silos ***OLAP Data Warehouse***

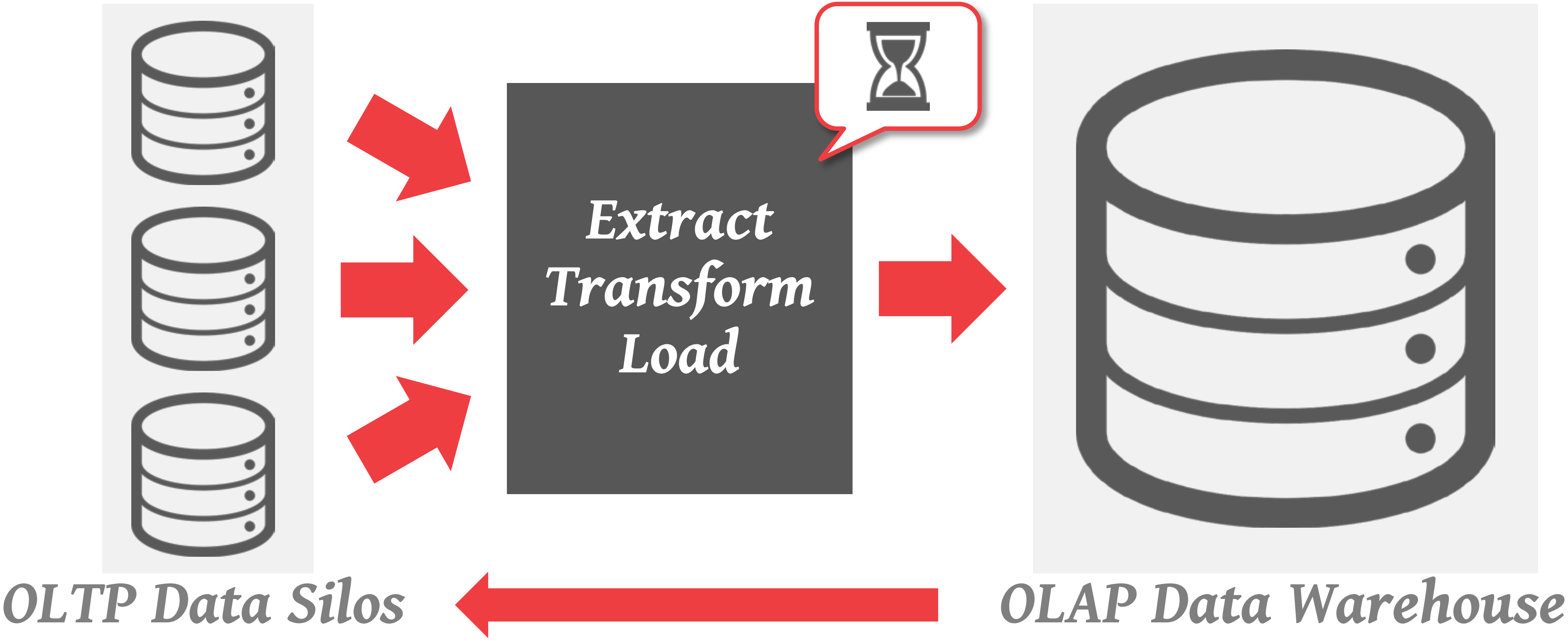
BIFURCATED ENVIRONMENT



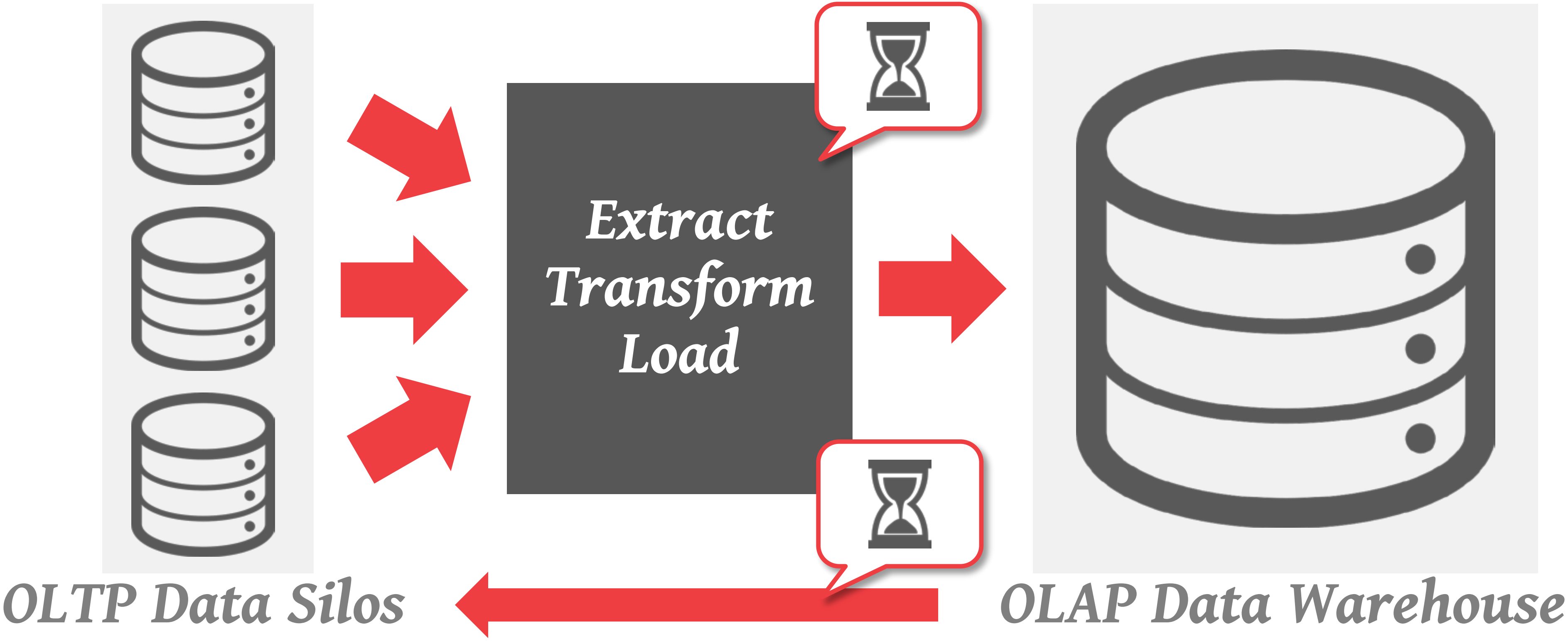
BIFURCATED ENVIRONMENT



BIFURCATED ENVIRONMENT



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.

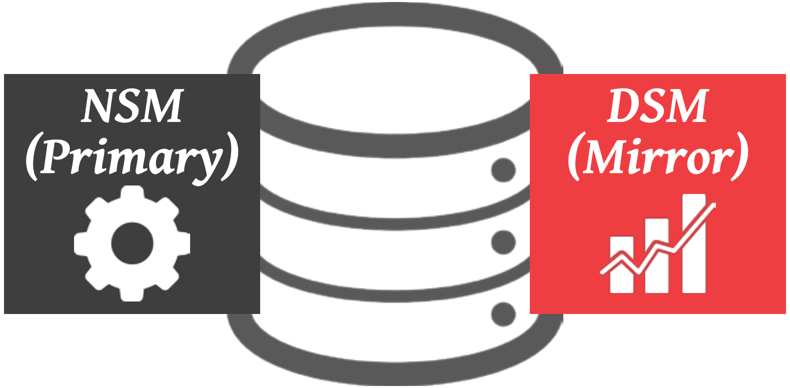
→ All updates are first entered in NSM then eventually copied into DSM mirror.

A CASE FOR FRACTURED MIRRORS
VLDB 2002

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.

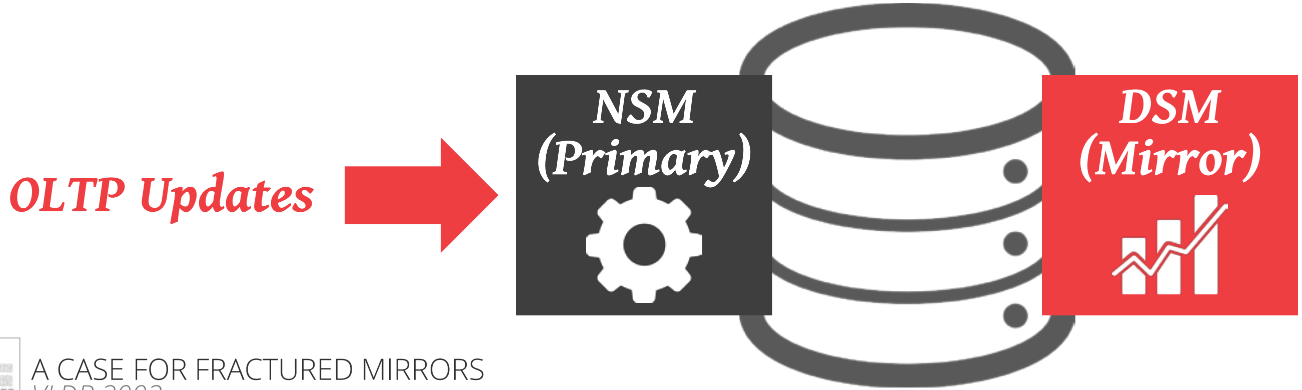


A CASE FOR FRACTURED MIRRORS
VLDB 2002

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.



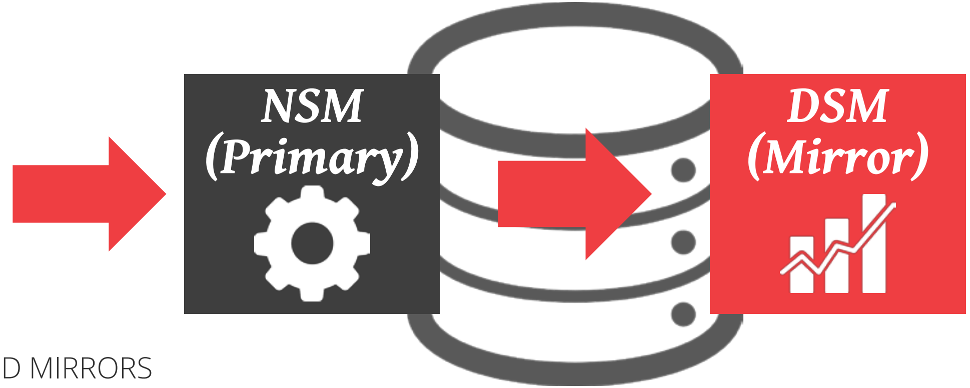
A CASE FOR FRACTURED MIRRORS
VLDB 2002

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.

OLTP Updates

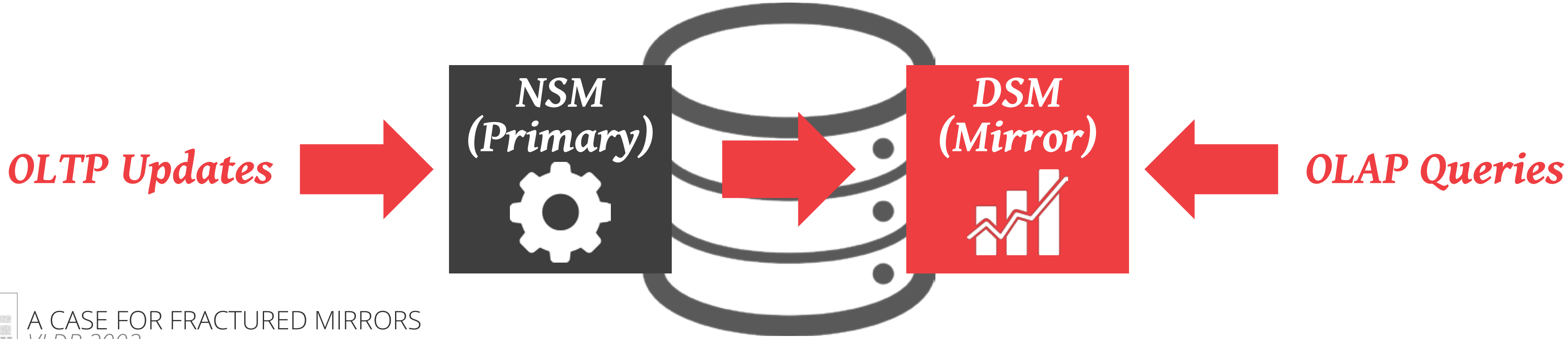


A CASE FOR FRACTURED MIRRORS
VLDB 2002

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

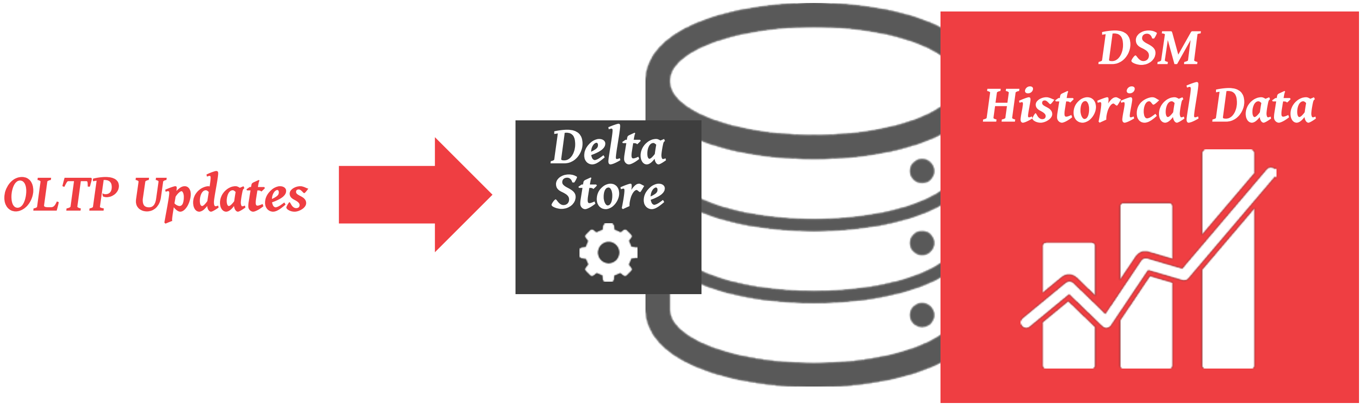
Stage updates to the database in an NSM table.

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

DELTA STORE

Stage updates to the database in an NSM table.

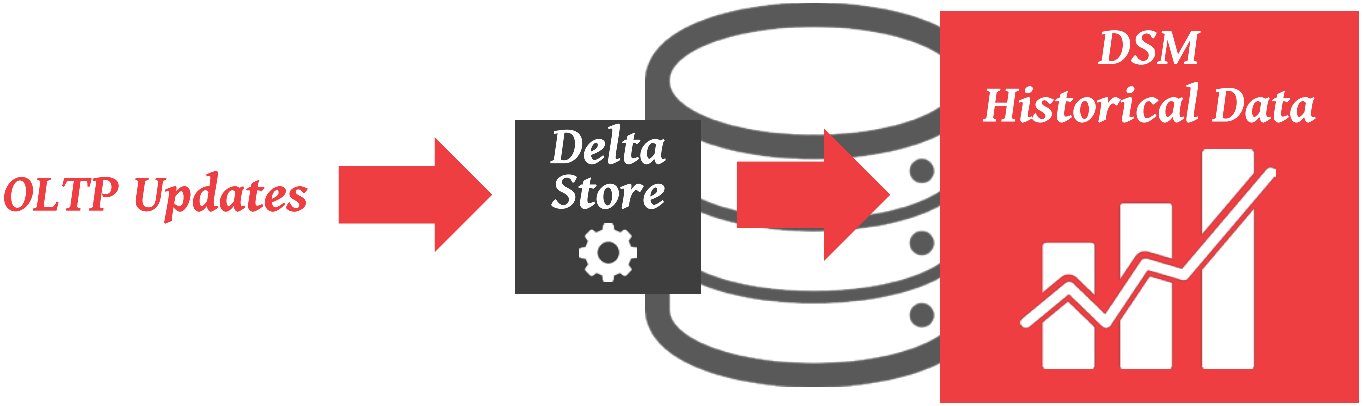
A background thread migrates updates from delta store and applies them to DSM data.



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.



SINGLE, FLEXIBLE ARCHITECTURE

Use 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 use the delta-store for NSM data with a single architecture.

H_2O ADAPTIVE STORAGE

Examine the access patterns of queries and then dynamically reconfigure the database to optimize decomposition and layout.

Copies columns into a new layout that is optimized for each query.

→ Think of it like a mini fractured mirror.

→ Use query compilation to speed up operations.

H2O: A HANDS-FREE ADAPTIVE STORE
SIGMOD 2014

H_2O ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = “xxx”
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```

Original Data

A	B	C	D

H_2O ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = “xxx”
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```

Original Data

A	B	C	D

H_2O ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = “xxx”
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```

Original Data

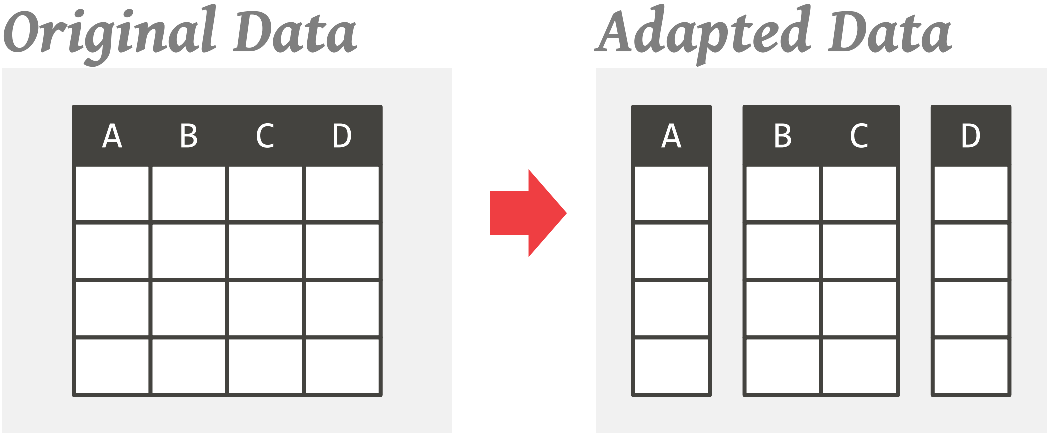
A	B	C	D

H_2O ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = “xxx”
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```



H_2O ADAPTIVE STORAGE

This approach is unable to handle updates to the database.
It also unable to store tuples in the same table in a different layout.

This is because they are missing the ability to categorize whether data is hot or cold...

PELOTON ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = "xxx"
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```

Original Data

A	B	C	D

PELOTON ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = “xxx”
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```

Original Data

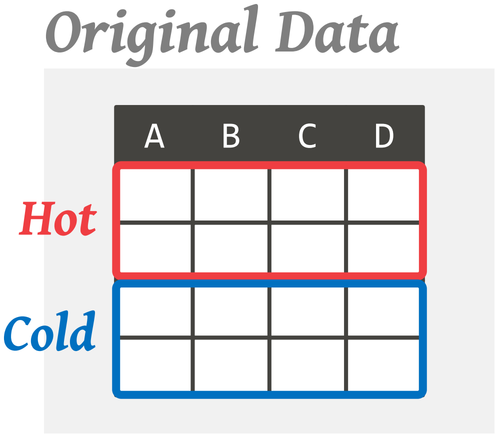
A	B	C	D

PELOTON ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = “xxx”
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```

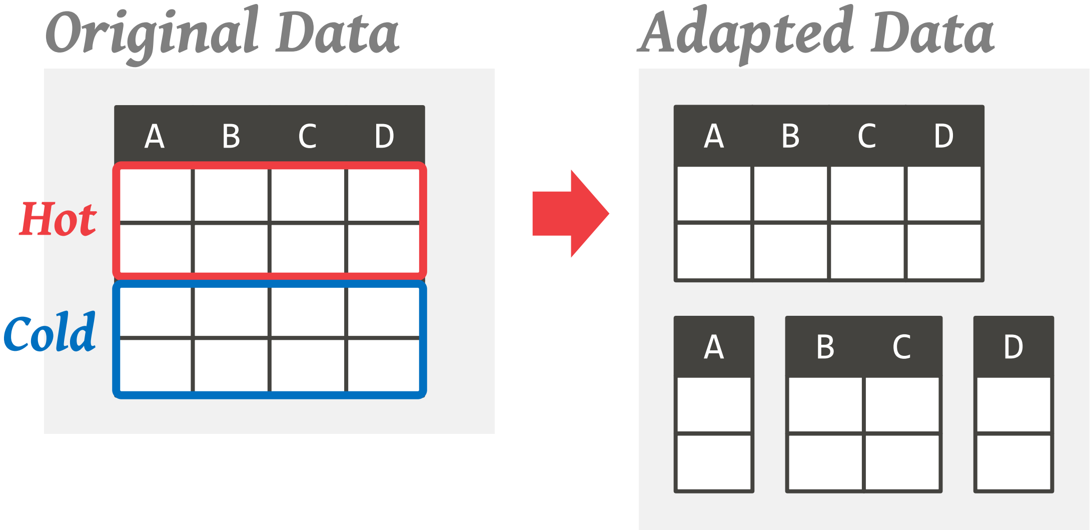


PELOTON ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = "xxx"
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```

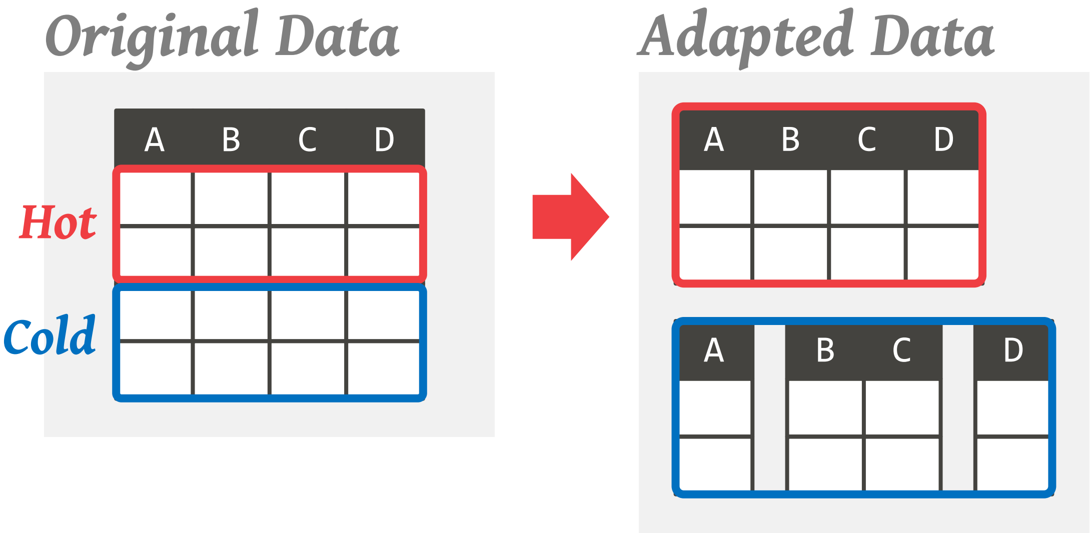


PELOTON ADAPTIVE STORAGE

```
UPDATE JoyStillSux
   SET B = 1234
 WHERE C = “xxx”
```

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

```
SELECT SUM(A)
  FROM JoyStillSux
```



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.

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 known database workloads except for matrices in machine learning.

JOY's DANK

TIPS FOR PROFILING

MOTIVATION

Consider a hot program **Z** with two functions **foo** and **bar**.

How can we speed up **Z** with only a debugger ?
→ Randomly pause it during execution
→ Collect the function call stack

RANDOM PAUSE METHOD

Consider this scenario

→ Collected 10 call stack samples

→ Say 6 out of the 10 samples were in **`foo`**

What percentage of time was spent in **`foo`**?

→ Roughly 60% of the time was spent in **`foo`**

→ Accuracy increases with # of samples

AMDAHL'S LAW

Say we optimized **foo** to run 2 times faster
What's the expected overall speedup ?

→ p = percentage of time spent in optimized task
→ s = speed up for the optimized task
→ Overall speedup = ——— = 1.4 times faster

AMDAHL'S LAW

Say we optimized **foo** to run 2 times faster

What's the expected overall speedup ?

→ 60% of time spent in **foo** drops in half

→ 40% of time spent in **bar** unaffected

→ ***p*** = percentage of time spent in optimized task

→ ***s*** = speed up for the optimized task

→ Overall speedup = ——— = 1.4 times faster

AMDAHL'S LAW

1 0.6 2 +0.4 1 1 0.6 2 +0.4 0.6 2 0.6 0.6 2 2 0.6
2 +0.4 1 0.6 2 +0.4 = 1.4 times faster

1 0.6 2 +0.4 1 1 0.6 2 +0.4 0.6 2 0.6 0.6 2 2 0.6
2 +0.4 1 0.6 2 +0.4 = 1.4 times faster

1 +(1-) +(*1*−) *1*
+(1−)

Say we optimized **foo** to run 2 times faster

What's the expected overall speedup ?
→ 60% of time spent in **foo** drops in half
→ 40% of time spent in **bar** unaffected

PROFILING TOOLS FOR REAL

Choice #1: Valgrind

→ Heavyweight instrumentation framework with a lot of tools
→ Sophisticated visualization tools

Choice #2: Perf

→ Lightweight tool that can record different kinds of events
→ Console-oriented visualization tools

CHOICE #1: VALGRIND

Instrumentation framework for building dynamic analysis tools

→ **`memcheck`**: a memory error detector

→ **`callgrind`**: a call-graph generating profiler

CHOICE #1: VALGRIND

Instrumentation framework for building dynamic analysis tools

→ **memcheck**: a memory error detector

→ **callgrind**: a call-graph generating profiler

Using **callgrind** to profile the index test and Peloton in general:

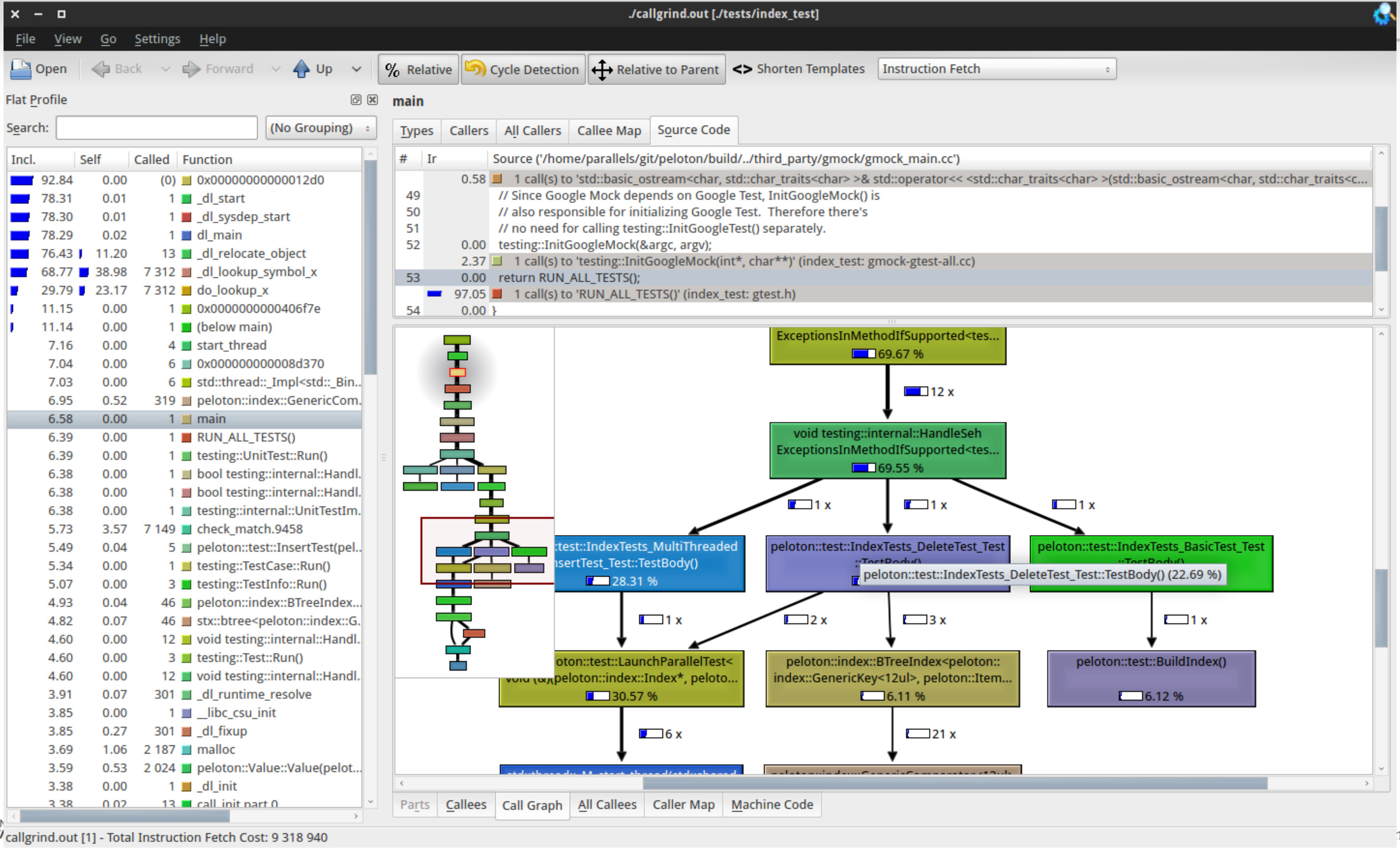
```
$ valgrind --tool=callgrind --trace-children=yes
./tests/index_test

$ valgrind --tool=callgrind --trace-children=yes
./build/src/peloton -D data &> /dev/null&
```

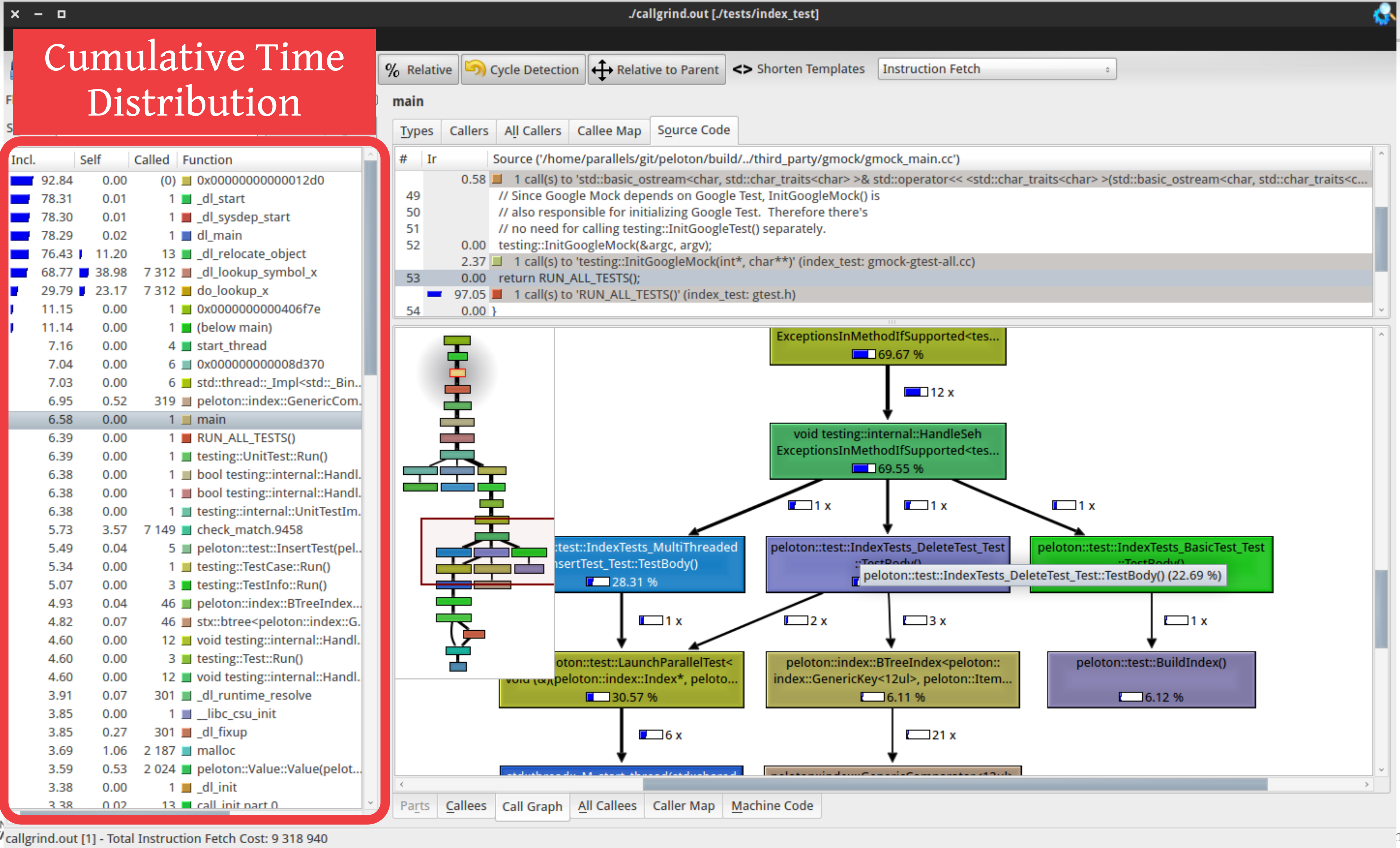
KCACHEGRIND

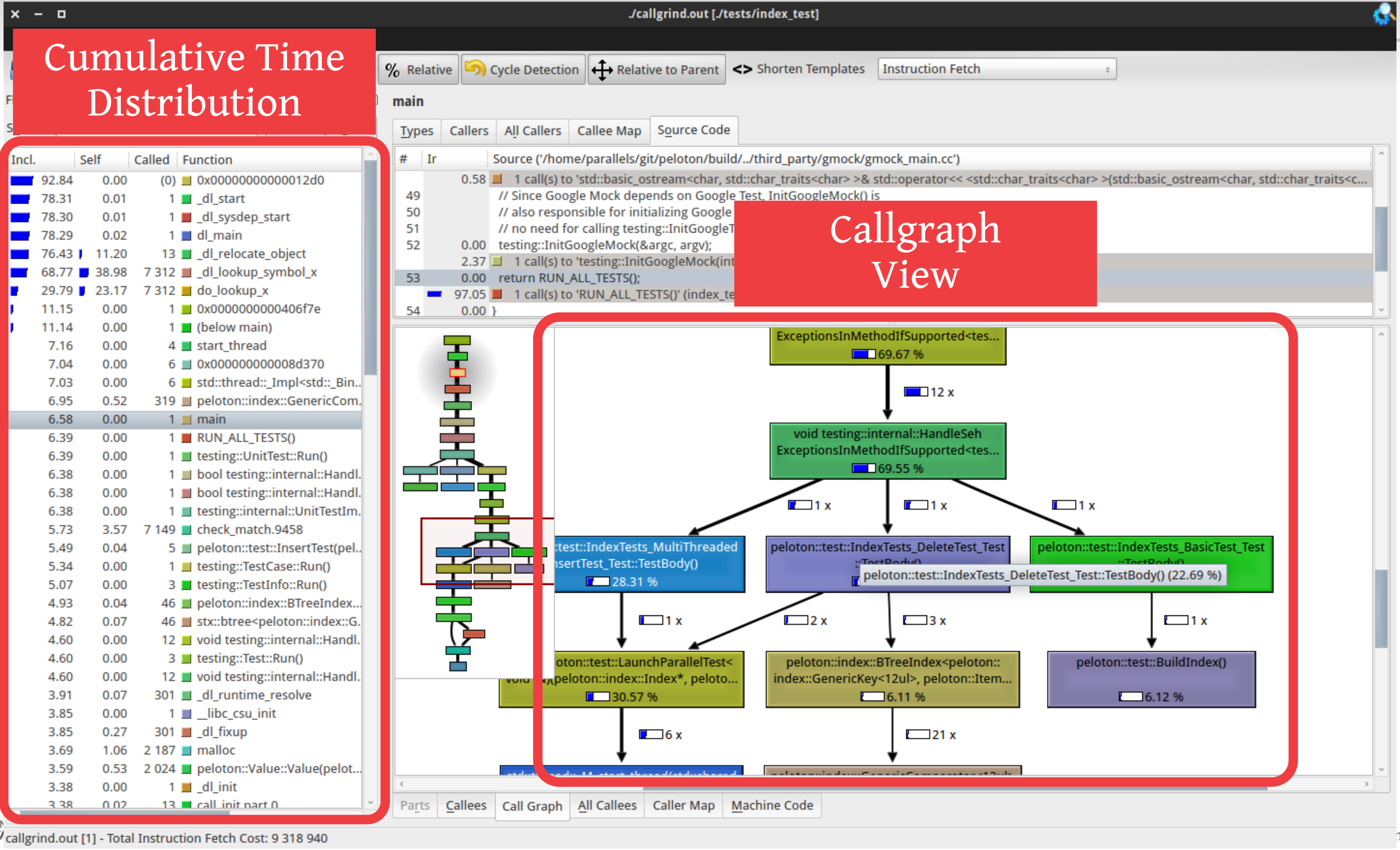
Profile data visualization tool

```
$ kcachegrind callgrind.out.12345
```


./callgrind.out [./tests/index_test]
File View Go Settings Help
Open Back Forward Up
Relative Cycle Detection Relative to Parent Shorten Templates Instruction Fetch
Flat Profile
Search: (No Grouping)
Incl. Self Called Function
92.84 0.00 (0) 0x00000000000012d0
78.31 0.01 1 _dl_start
78.30 0.01 1 _dl_sysdep_start
78.29 0.02 1 dl_main
76.43 11.20 13 _dl_relocate_object
68.77 38.98 7 312 _dl_lookup_symbol_x
29.79 23.17 7 312 do_lookup_x
11.15 0.00 1 0x0000000000406f7e
11.14 0.00 1 (below main)
7.16 0.00 4 start_thread
7.04 0.00 6 0x000000000008d370
7.03 0.00 6 std::thread::_Impl<std::_Bin...
6.95 0.52 319 peloton::index::GenericCom...
6.58 0.00 1 main
6.39 0.00 1 RUN_ALL_TESTS()
6.39 0.00 1 testing::UnitTest::Run()
6.38 0.00 1 bool testing::internal::Handl...
6.38 0.00 1 bool testing::internal::Handl...
6.38 0.00 1 testing::internal::UnitTestIm...
5.73 3.57 7 149 check_match.9458
5.49 0.04 5 peloton::test::InsertTest(pel...
5.34 0.00 1 testing::TestCase::Run()
5.07 0.00 3 testing::TestInfo::Run()
4.93 0.04 46 peloton::index::BTreeIndex...
4.82 0.07 46 stx::btree<peloton::index::G...
4.60 0.00 12 void testing::internal::Handl...
4.60 0.00 3 testing::Test::Run()
4.60 0.00 12 void testing::internal::Handl...
3.91 0.07 301 _dl_runtime_resolve
3.85 0.00 1 __libc_csu_init
3.85 0.27 301 _dl_fixup
3.69 1.06 2 187 malloc
3.59 0.53 2 024 peloton::Value::Value(pelot...
3.38 0.00 1 _dl_init
main
Types Callers All Callers Callee Map Source Code
Ir Source ('/home/parallels/git/peloton/build/../third_party/gmock/gmock_main.cc')
0.58 1 call(s) to 'std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, std::char_traits<c...
49 // Since Google Mock depends on Google Test, InitGoogleMock() is
50 // also responsible for initializing Google Test. Therefore there's
51 // no need for calling testing::InitGoogleTest() separately.
52 0.00 testing::InitGoogleMock(&argc, argv);
2.37 1 call(s) to 'testing::InitGoogleMock(int*, char**)' (index_test: gmock-gtest-all.cc)
53 0.00 return RUN_ALL_TESTS();
97.05 1 call(s) to 'RUN_ALL_TESTS()' (index_test: gtest.h)
54 0.00 }
ExceptionsInMethodIfSupported<tes...
69.67 %
12 x
void testing::internal::HandleSeh
ExceptionsInMethodIfSupported<tes...
69.55 %
1 x
1 x
1 x
test::IndexTests_MultiThreaded
nsertTest_Test::TestBody()
28.31 %
peloton::test::IndexTests_DeleteTest_Test
peloton::test::IndexTests_DeleteTest_Test::TestBody() (22.69 %)
peloton::test::IndexTests_BasicTest_Test
1 x
2 x
3 x
1 x
oton::test::LaunchParallelTest<
peloton::index::Index*, peloto...
30.57 %
peloton::index::BTreeIndex<peloton::
index::GenericKey<12ul>, peloton::Item...
6.11 %
peloton::test::BuildIndex()
6.12 %
6 x
21 x
Parts Callees Call Graph All Callees Caller Map Machine Code
callgrind.out [1] - Total Instruction Fetch Cost: 9 318 940

Cumulative Time Distribution




./callgrind.out [./tests/index_test]
Cumulative Time Distribution
Callgraph View
Relative
Cycle Detection
Relative to Parent
Shorten Templates
Instruction Fetch
main
Types
Callers
All Callers
Callee Map
Source Code
Ir Source ('/home/parallels/git/peloton/build/../third_party/gmock/gmock_main.cc')
0.58 1 call(s) to 'std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, std::char_traits<c...
49 // Since Google Mock depends on Google Test, InitGoogleMock() is
50 // also responsible for initializing Google
51 // no need for calling testing::InitGoogleT
52 0.00 testing::InitGoogleMock(&argc, argv);
2.37 1 call(s) to 'testing::InitGoogleMock(int
53 0.00 return RUN_ALL_TESTS();
97.05 1 call(s) to 'RUN_ALL_TESTS()' (index_te
54 0.00 }
Incl. Self Called Function
92.84 0.00 (0) 0x00000000000012d0
78.31 0.01 1 _dl_start
78.30 0.01 1 _dl_sysdep_start
78.29 0.02 1 dl_main
76.43 11.20 13 _dl_relocate_object
68.77 38.98 7 312 _dl_lookup_symbol_x
29.79 23.17 7 312 do_lookup_x
11.15 0.00 1 0x0000000000406f7e
11.14 0.00 1 (below main)
7.16 0.00 4 start_thread
7.04 0.00 6 0x000000000008d370
7.03 0.00 6 std::thread::_Impl<std::_Bin...
6.95 0.52 319 peloton::index::GenericCom...
6.58 0.00 1 main
6.39 0.00 1 RUN_ALL_TESTS()
6.39 0.00 1 testing::UnitTest::Run()
6.38 0.00 1 bool testing::internal::Handl...
6.38 0.00 1 bool testing::internal::Handl...
6.38 0.00 1 testing::internal::UnitTestIm...
5.73 3.57 7 149 check_match.9458
5.49 0.04 5 peloton::test::InsertTest(pel...
5.34 0.00 1 testing::TestCase::Run()
5.07 0.00 3 testing::TestInfo::Run()
4.93 0.04 46 peloton::index::BTreeIndex...
4.82 0.07 46 stx::btree<peloton::index::G...
4.60 0.00 12 void testing::internal::Handl...
4.60 0.00 3 testing::Test::Run()
4.60 0.00 12 void testing::internal::Handl...
3.91 0.07 301 _dl_runtime_resolve
3.85 0.00 1 __libc_csu_init
3.85 0.27 301 _dl_fixup
3.69 1.06 2 187 malloc
3.59 0.53 2 024 peloton::Value::Value(pelot...
3.38 0.00 1 _dl_init
3.38 0.02 13 call_init.part.0
ExceptionsInMethodIfSupported<tes...
69.67 %
12 x
void testing::internal::HandleSeh
ExceptionsInMethodIfSupported<tes...
69.55 %
1 x
1 x
1 x
:test::IndexTests_MultiThreaded
nsertTest_Test::TestBody()
28.31 %
peloton::test::IndexTests_DeleteTest_Test
peloton::test::IndexTests_DeleteTest_Test::TestBody() (22.69 %)
peloton::test::IndexTests_BasicTest_Test
1 x
2 x
3 x
1 x
oton::test::LaunchParallelTest<
peloton::index::Index*, peloto...
30.57 %
peloton::index::BTreeIndex<peloton::
index::GenericKey<12ul>, peloton::Item...
6.11 %
peloton::test::BuildIndex()
6.12 %
6 x
21 x
Parts
Callees
Call Graph
All Callees
Caller Map
Machine Code
callgrind.out [1] - Total Instruction Fetch Cost: 9 318 940

CHOICE #2: PERF

Tool for using the performance counters subsystem in Linux.
→ **-e** = sample the event `cycles` at the user level only
→ **-c** = collect a sample every 2000 occurrences of event

```
$ perf record -e cycles:u -c 2000
./tests/index_test
```

Uses counters for tracking events
→ On counter overflow, the kernel records a sample
→ Sample contains info about program execution

PERF VISUALIZATION

We can also use **perf** to visualize the generated profile for our application.

```
$ perf report
```

PERF VISUALIZATION

```
perf report
File  Edit  View  Search  Terminal  Help
Samples: 56  of event 'cpu-clock:u', Event count (approx.): 56
 25.00%  index_test  ld-2.19.so             [.] do_lookup_x
 25.00%  index_test  ld-2.19.so             [.] _dl_lookup_symbol_x
 21.43%  index_test  ld-2.19.so             [.] _dl_relocate_object
  7.14%  index_test  ld-2.19.so             [.] check_match.9458
  3.57%  index_test  libstdc++.so.6.0.21    [.] operator delete(void*)
  1.79%  index_test  libstdc++.so.6.0.21    [.] __dynamic_cast
  1.79%  index_test  libstdc++.so.6.0.21    [.] operator new(unsigned long)
  1.79%  index_test  libpelotonpg.so.0.0.0  [.] Json::Value::~Value()
  1.79%  index_test  libpeloton.so.0.0.0    [.] peloton::Value::CompareWithoutNull(peloton::Value) const
  1.79%  index_test  libc-2.19.so           [.] _int_free
  1.79%  index_test  libc-2.19.so           [.] __memcpy_sse2_unaligned
  1.79%  index_test  libc-2.19.so           [.] _dl_addr
  1.79%  index_test  libc-2.19.so           [.] __libc_dl_error_tsd
  1.79%  index_test  ld-2.19.so             [.] strcmp
  1.79%  index_test  index_test             [.] testing::TestEventListeners::TestEventListeners()
```

PERF VISUALIZATION

perf report

File Edit View Search Terminal Help

```
Samples: 56  of event 'cpu-clock:u', Event count (approx.): 56
 25.00%  index_test  ld-2.19.so             [.] do_lookup_x
 25.00%  index_test  ld-2.19.so             [.] _dl_lookup_symbol_x
 21.43%  index_test  ld-2.19.so             [.] _dl_relocate_object
  7.14%  index_test  ld-2.19.so             [.] check_match.9458
  3.57%  index_test  libstdc++.so.6.0.21    [.] operator delete(void*)
  1.79%  index_test  libstdc++.so.6.0.21    [.] __dynamic_cast
  1.79%  index_test  libstdc++.so.6.0.21    [.] operator new(unsigned long)
  1.79%  index_test  libpelotonpg.so.0.0.0  [.] Json::Value::~Value()
  1.79%  index_test  libpeloton.so.0.0.0    [.] peloton::Value::CompareWithoutNull(peloton::Value) const
  1.79%  index_test  libc-2.19.so           [.] _int_free
  1.79%  index_test  libc-2.19.so           [.] __memcpy_sse2_unaligned
  1.79%  index_test  libc-2.19.so           [.] _dl_addr
  1.79%  index_test  libc-2.19.so           [.] __libc_dl_error_tsd
  1.79%  index_test  ld-2.19.so             [.] strcmp
  1.79%  index_test  index_test             [.] testing::TestEventListeners::TestEventListeners()
```

Cumulative Time Distribution

PERF EVENTS

Supports several other events like:
→ L1-dcache-load-misses
→ branch-misses

To see a list of events:

```
$ perf list
```

Another usage example:

```
$ perf record -e cycles,LLC-load-misses -c 2000
./tests/index_test
```

REFERENCES

Valgrind

→ The Valgrind Quick Start Guide
→ Callgrind
→ Kcachegrind
→ Tips for the Profiling/Optimization process

Perf

→ Perf Tutorial
→ Perf Examples
→ Perf Analysis Tools