HPI Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Evaluating Lightweight Integer Compression Algorithms in Column-Oriented In-Memory DBMS

Linus Heinzl Ben Hurdelhey Martin Boissier Michael Perscheid Hasso Plattner

Copenhagen, 16 August 2021

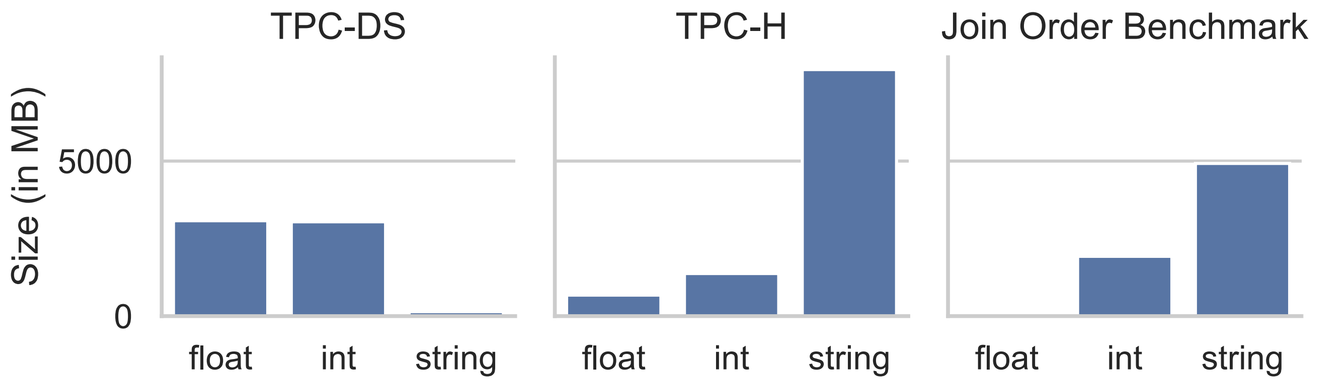
Agenda

Agenda

- **Processing integers in database systems**
  - Standard benchmarks
  - SAP ERP system
- **Hyrise**
- **Integer encodings**
  - Sequential and non-sequential decoding
- **End-to-end benchmarks**
  - TPC-DS, TPC-H, and Join Order Benchmark
  - Intel Xeon Gold 6240L, AMD EPYC 7742, and Apple M1

Processing Integers

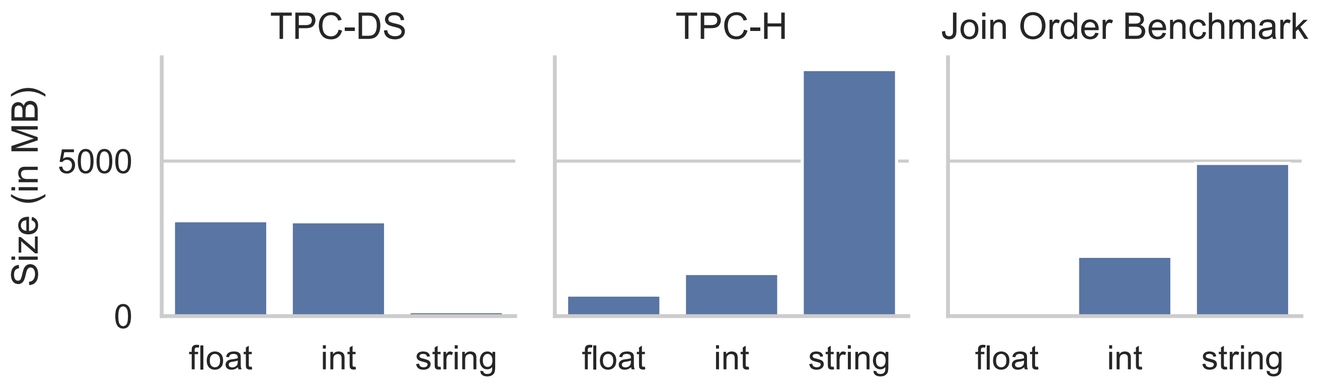
Processing Integers



In Hyrise, decimals are stored using floats, dates as strings.

Processing Integers

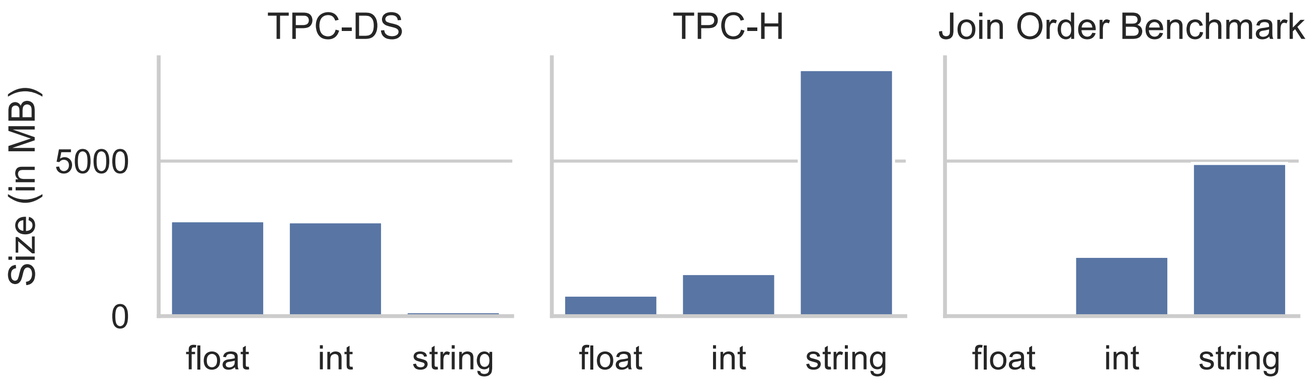
For most benchmarks, integers contribute only a small part to the footprint.



In Hyrise, decimals are stored using floats, dates as strings.

Processing Integers

For most benchmarks, integers contribute only a small part to the footprint.



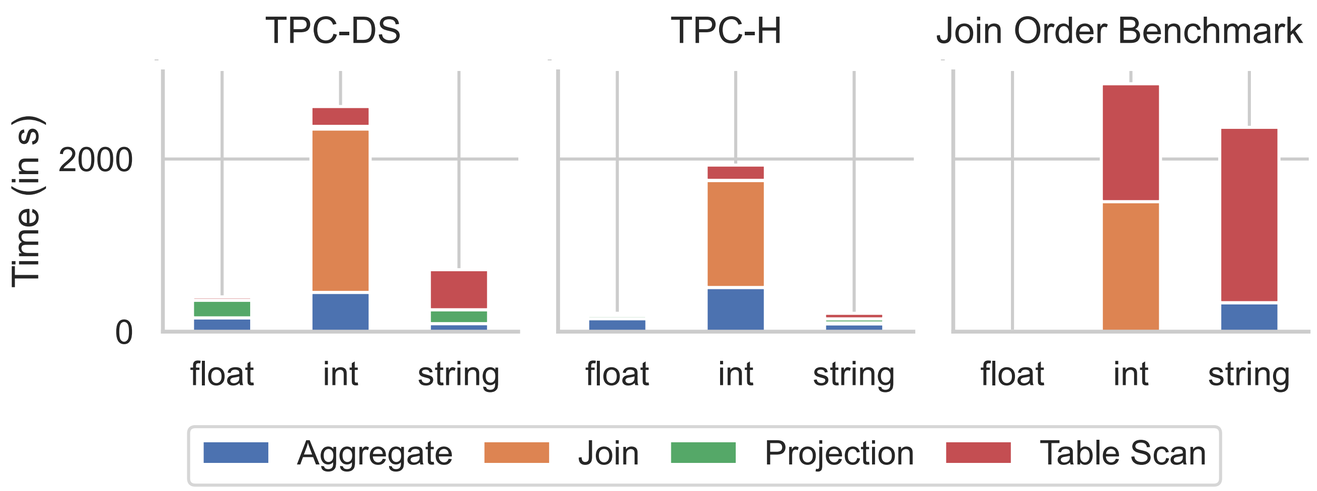
In Hyrise, decimals are stored using floats, dates as strings.

At least as long as strings are present. Warehouses can be a different case.

Processing Integers

However ...

integers are highly relevant when it comes to processing time.



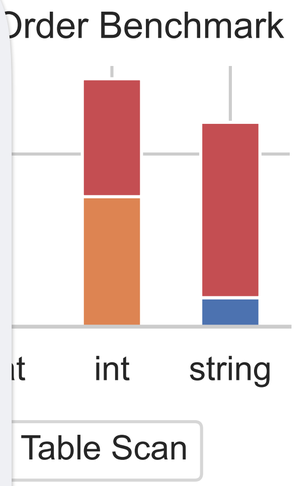
While join ordering is the toughest part of the JOB, most time in Hyrise goes to `LIKE` and `IN()` scans.

Processing Integers

Howev

intege
comes

Assuming the database **does not store** all the data all the time in DRAM:



est part of the JOB,
E and IN( ) scans.

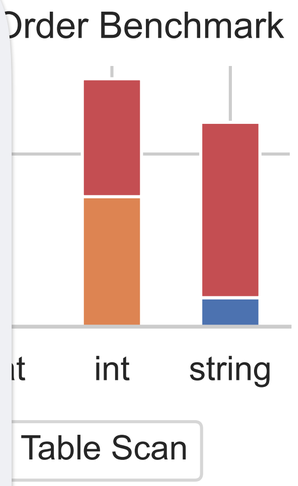
Processing Integers

Howev

intege
comes

Assuming the database **does not store** all the data all the time in DRAM:

- most cached/buffered data in DRAM will be integer data



est part of the JOB,
E and IN( ) scans.

Processing Integers

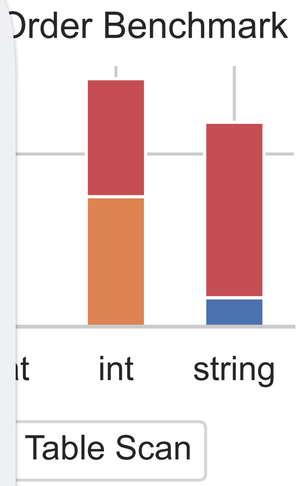
Howev

intege

comes

Assuming the database **does not store** all the data all the time in DRAM:

- most cached/buffered data in DRAM will be integer data
- even when data is tiered: DRAM is still a scarce resource



est part of the JOB,
E and IN( ) scans.

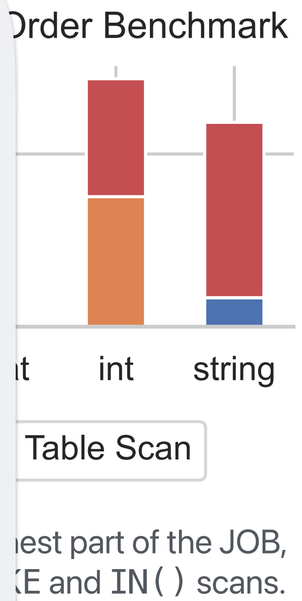
Processing Integers

Howev

intege
comes

Assuming the database **does not store** all the data all the time in DRAM:

- most cached/buffered data in DRAM will be integer data
- even when data is tiered: DRAM is still a scarce resource
- optimizing integers' performance/size can have a vast impact



Real-World Systems: SAP ERP

Real-World Systems: SAP ERP

What about real-world systems?

Real-World Systems: SAP ERP

What about real-world systems?

Vogelsgesang et al. found that ~50% of data is stored in strings (Tableau) [1].

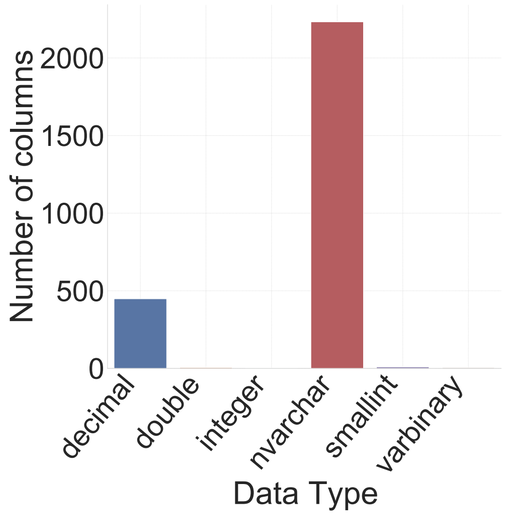
[1] Vogelsgesang et al.: Get Real: **How Benchmarks Fail to Represent the Real World**. DBTest@SIGMOD 2018: 1:1-1:6

Real-World Systems: SAP ERP

What about real-world systems?

Vogelsgesang et al. found that ~50% of data is stored in strings (Tableau) [1].

Query Plan Cache Analyses of an SAP ERP system of a Fortune 500 company



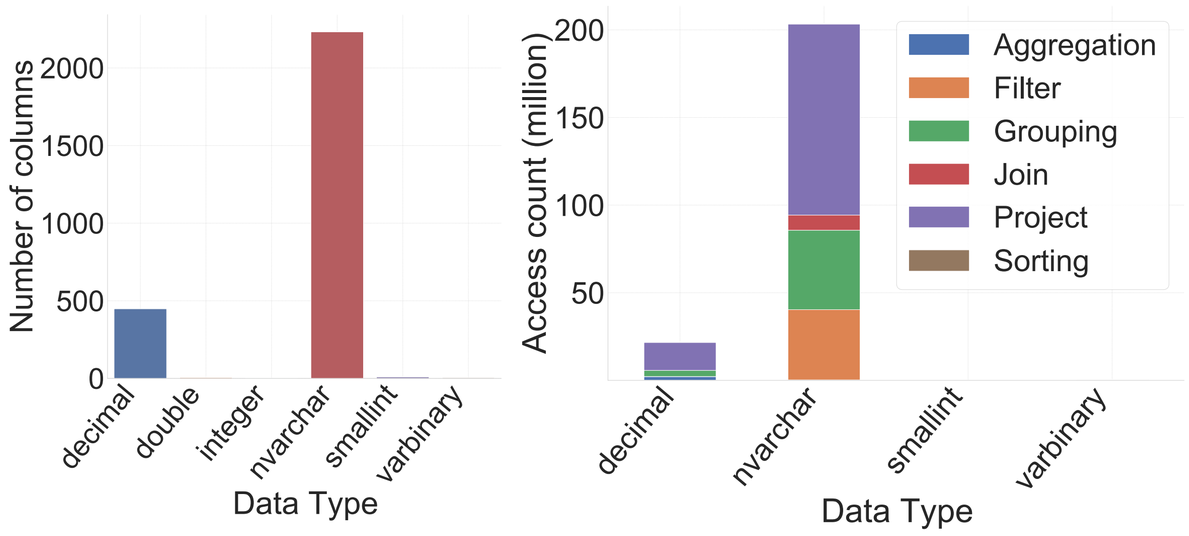
[1] Vogelsgesang et al.: Get Real: **How Benchmarks Fail to Represent the Real World**. DBTest@SIGMOD 2018: 1:1-1:6

Real-World Systems: SAP ERP

What about real-world systems?

Vogelsgesang et al. found that ~50% of data is stored in strings (Tableau) [1].

Query Plan Cache Analyses of an SAP ERP system of a Fortune 500 company



[1] Vogelsgesang et al.: Get Real: **How Benchmarks Fail to Represent the Real World**. DBTest@SIGMOD 2018: 1:1-1:6

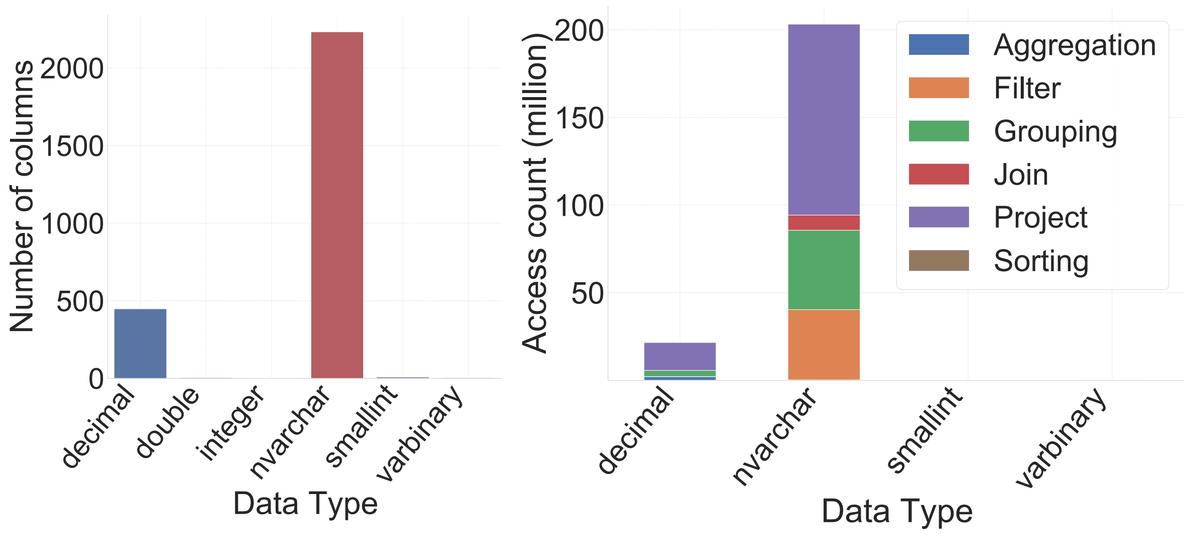
Real-World Systems: SAP ERP

What about real-world systems?

Vogelsgesang et al. found that ~50% of data is stored in strings (Tableau) [1].

6 of the 10 most frequently accessed columns in the SAP ERP are actually integer columns.

Query Plan Cache Analyses of an SAP ERP system of a Fortune 500 company



[1] Vogelsgesang et al.: Get Real: **How Benchmarks Fail to Represent the Real World**. DBTest@SIGMOD 2018: 1:1-1:6

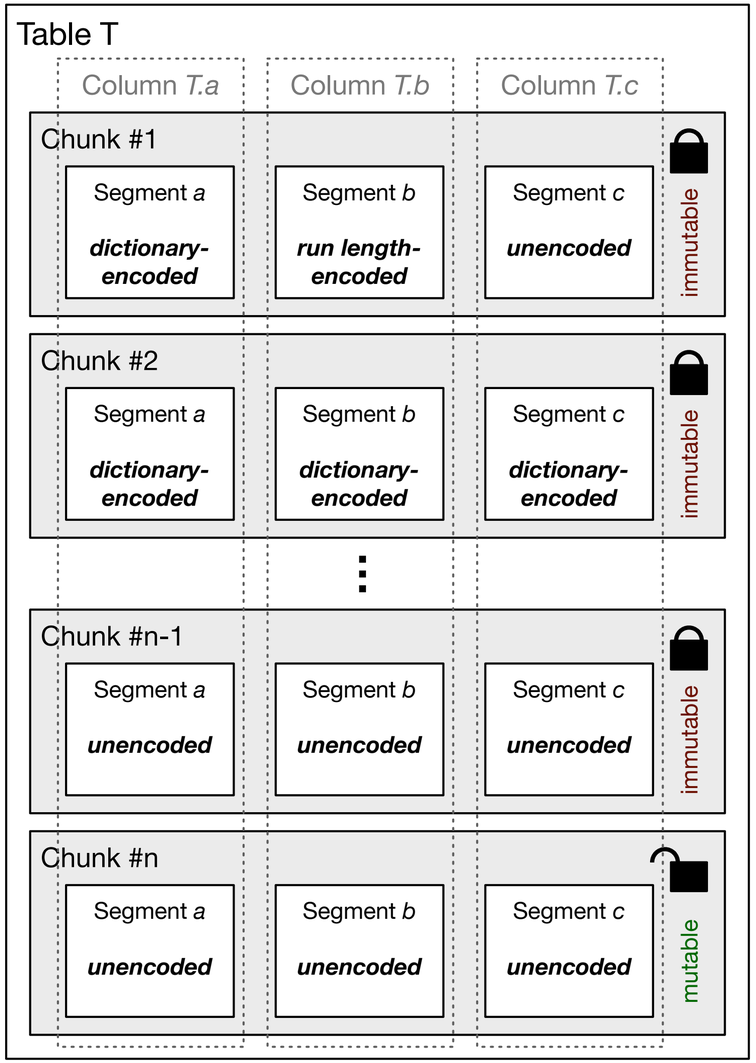
Hyrise

Hyrise

- An columnar in-memory database
  - Speaks SQL
  - Open source[1]
  - HTAP
  - Research & teaching
- Relevant characteristics for this talk
  - chunk-based table layout
  - intermediate query results passed via position lists

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

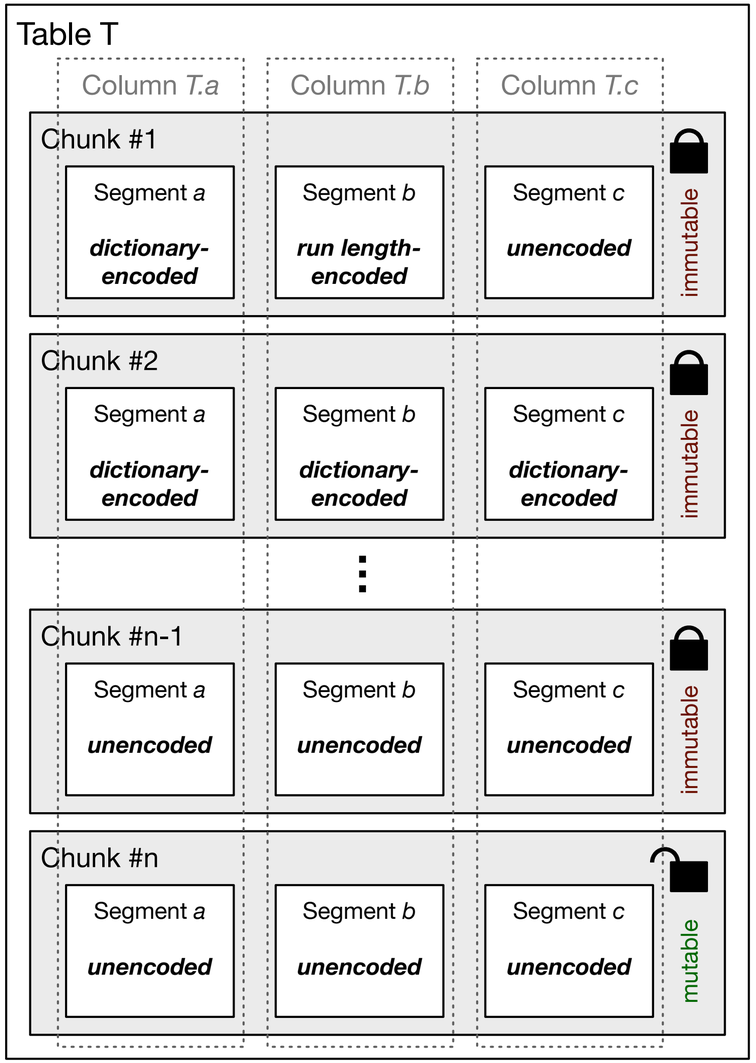


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

- Every table is automatically horizontally partitioned

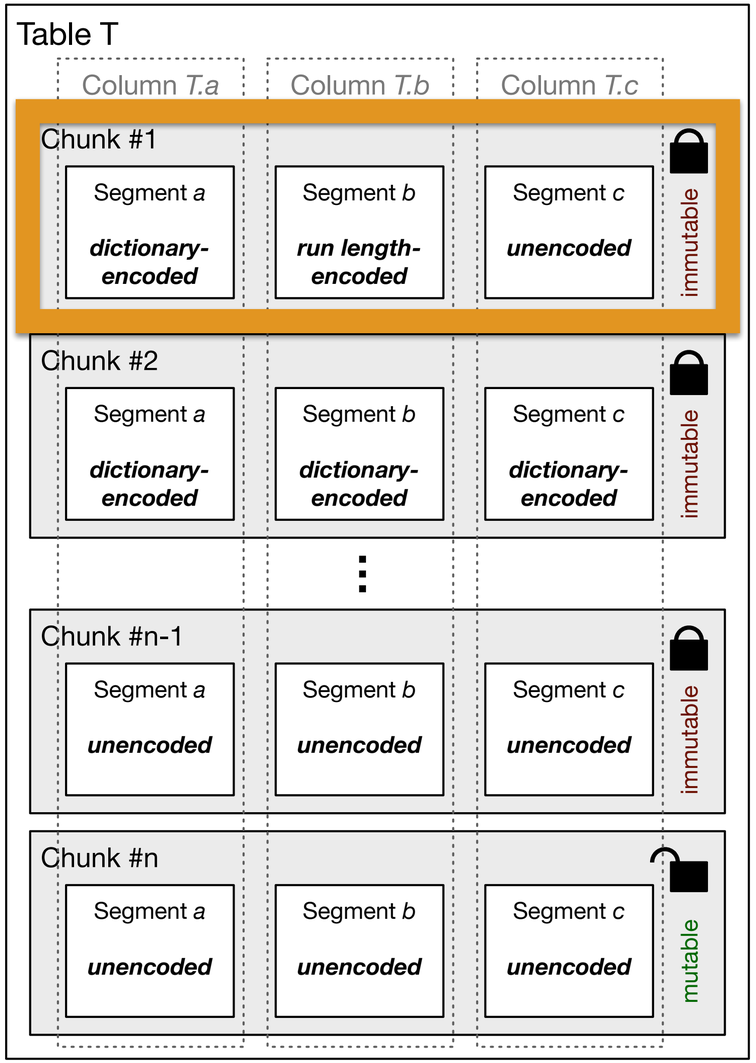


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

- Every table is automatically horizontally partitioned
  - so-called **chunks**

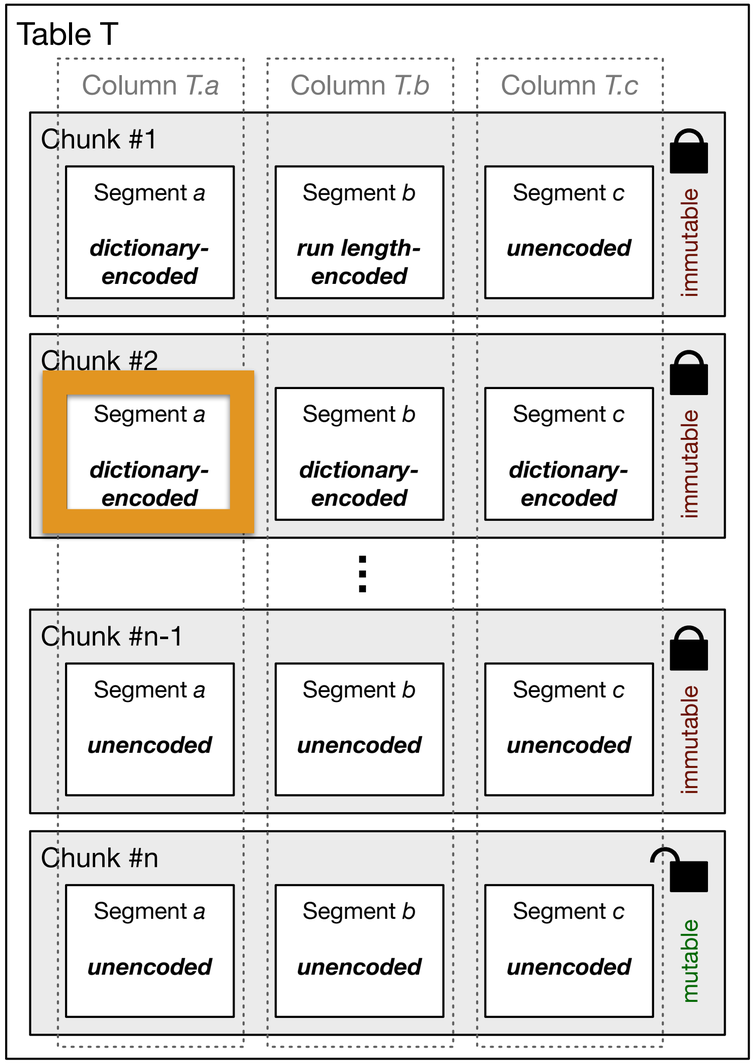


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

- Every table is automatically horizontally partitioned
  - so-called **chunks**
  - each attribute in the chunk is called a **segment**

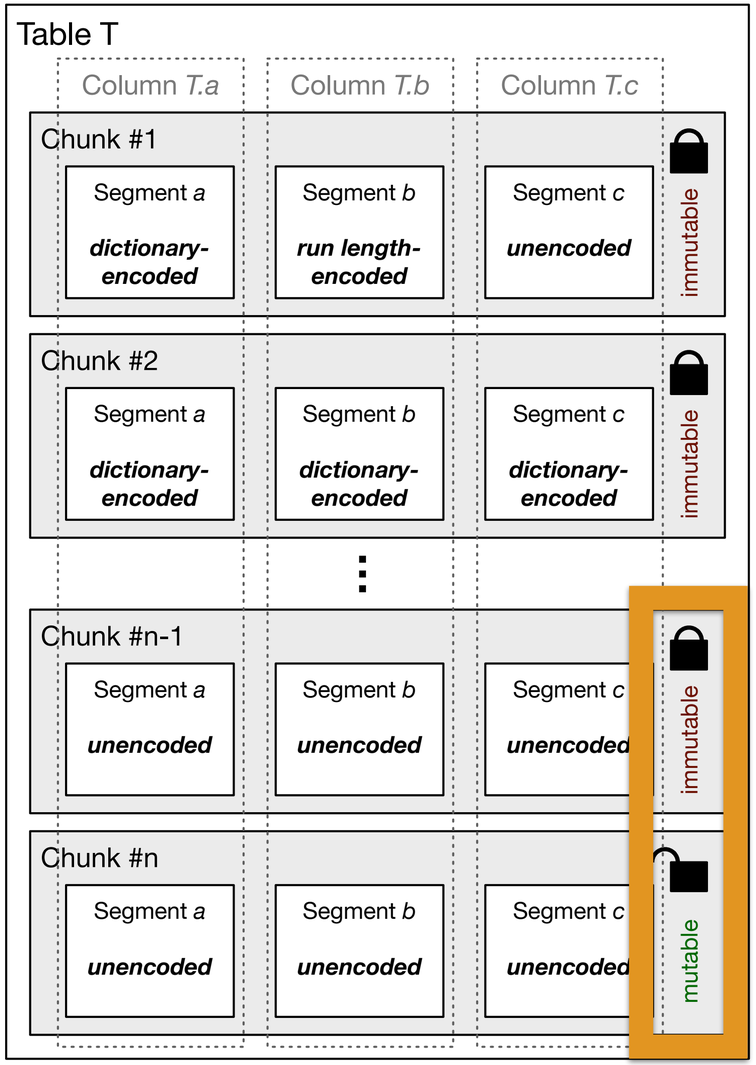


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

- Every table is automatically horizontally partitioned
  - so-called **chunks**
  - each attribute in the chunk is called a **segment**
- Once a chunk is full, it becomes **immutable** and a new mutable chunk is created

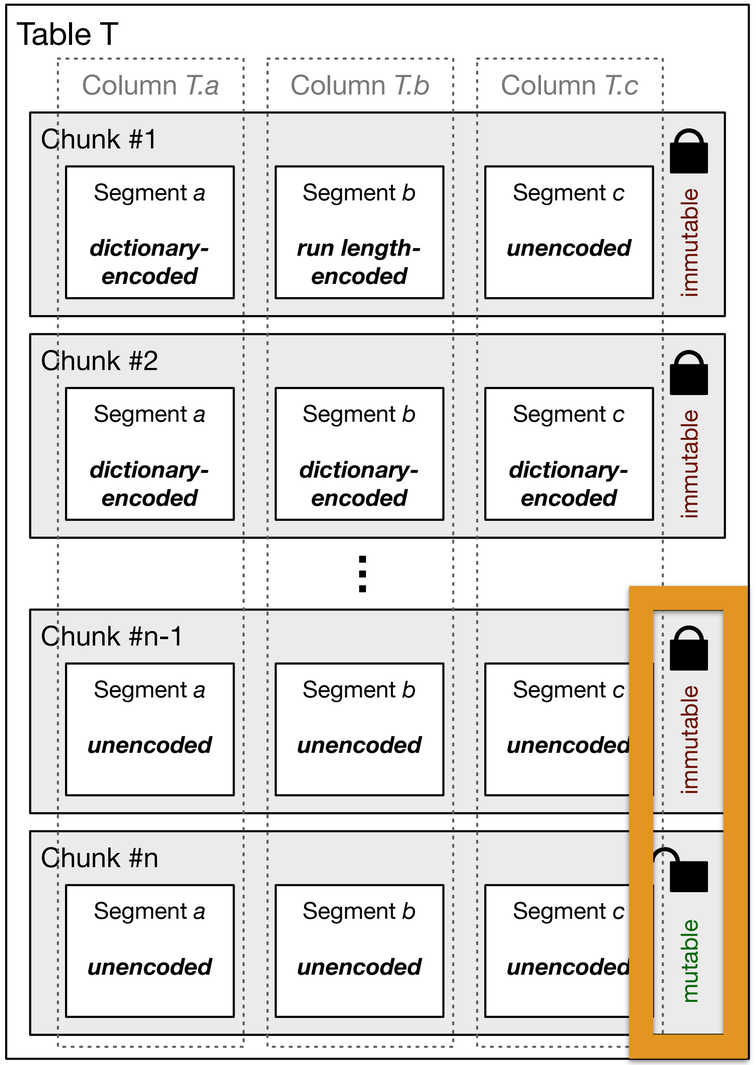


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

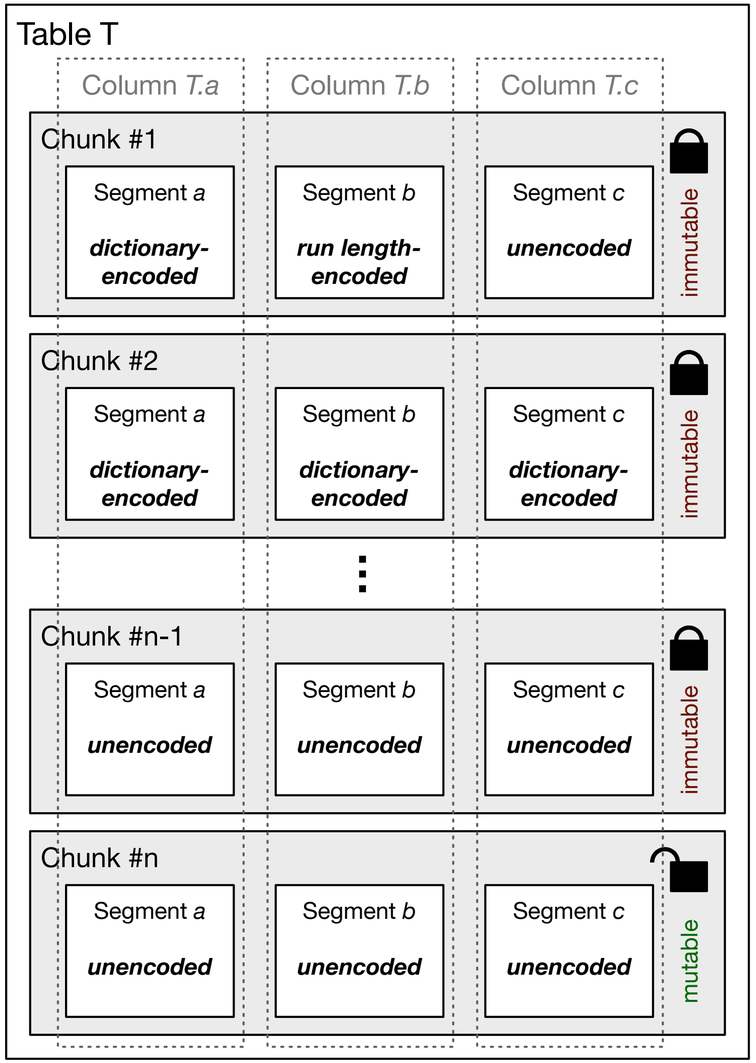
- Every table is automatically horizontally partitioned
  - so-called **chunks**
  - each attribute in the chunk is called a **segment**
- Once a chunk is full, it becomes **immutable** and a new mutable chunk is created
- **Insert-only**: `UPDATE` marks the old row as deleted and inserts updated row to the most recent chunk



Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

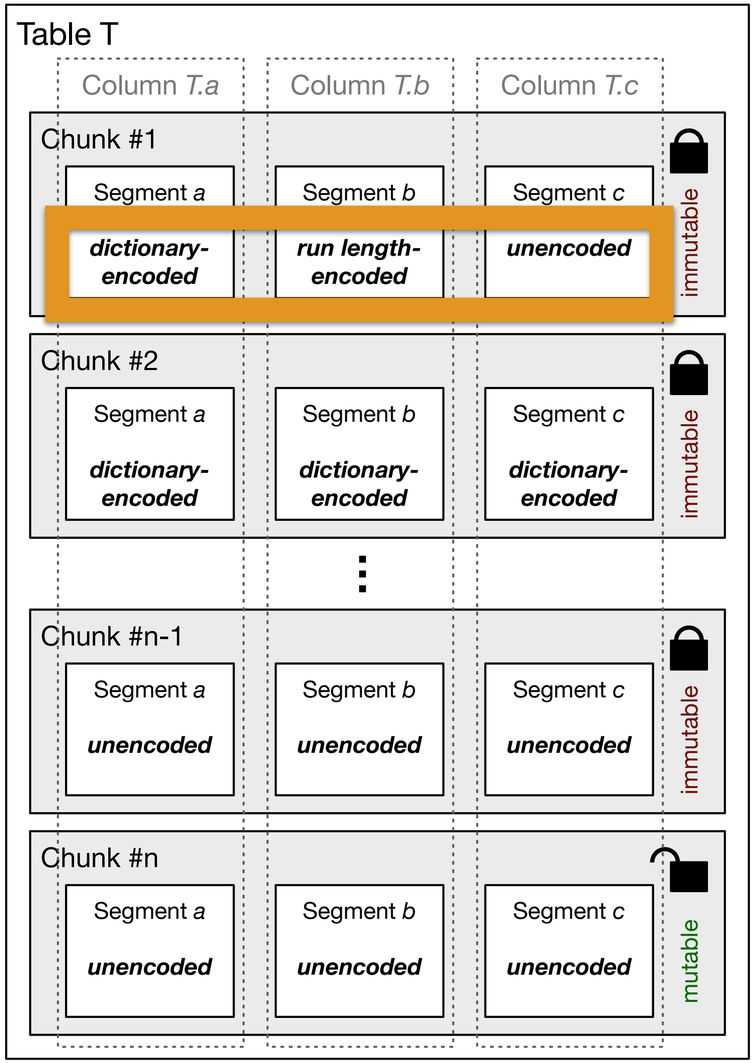


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

- Segments can be **encoded independently**

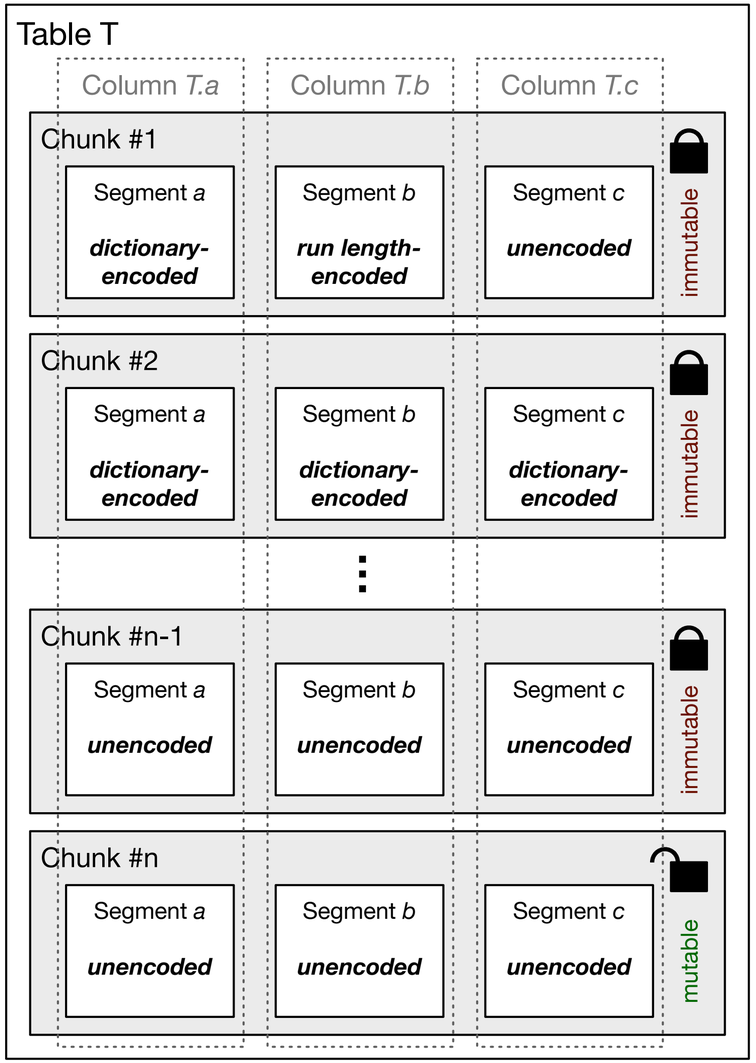


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

- Segments can be **encoded independently**
- On top of the segment encoding type, internal **integer vectors can be further compressed**

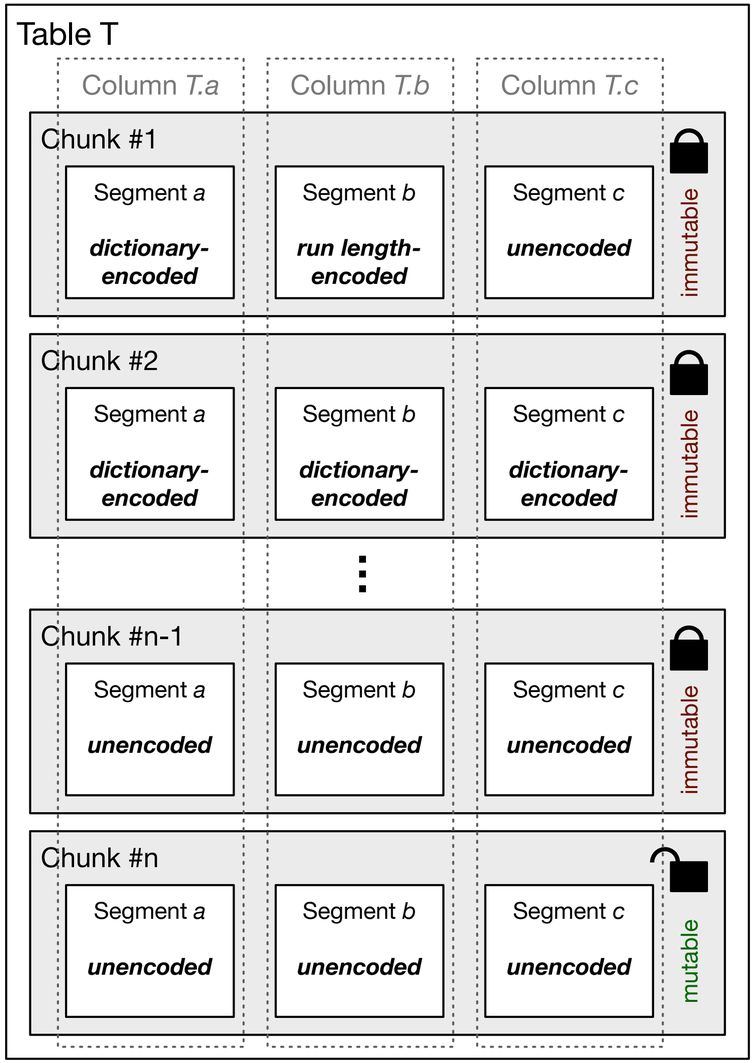


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

- Segments can be **encoded independently**
- On top of the segment encoding type, internal **integer vectors can be further compressed**
  - e.g., the offset vector of a dictionary encoded segment

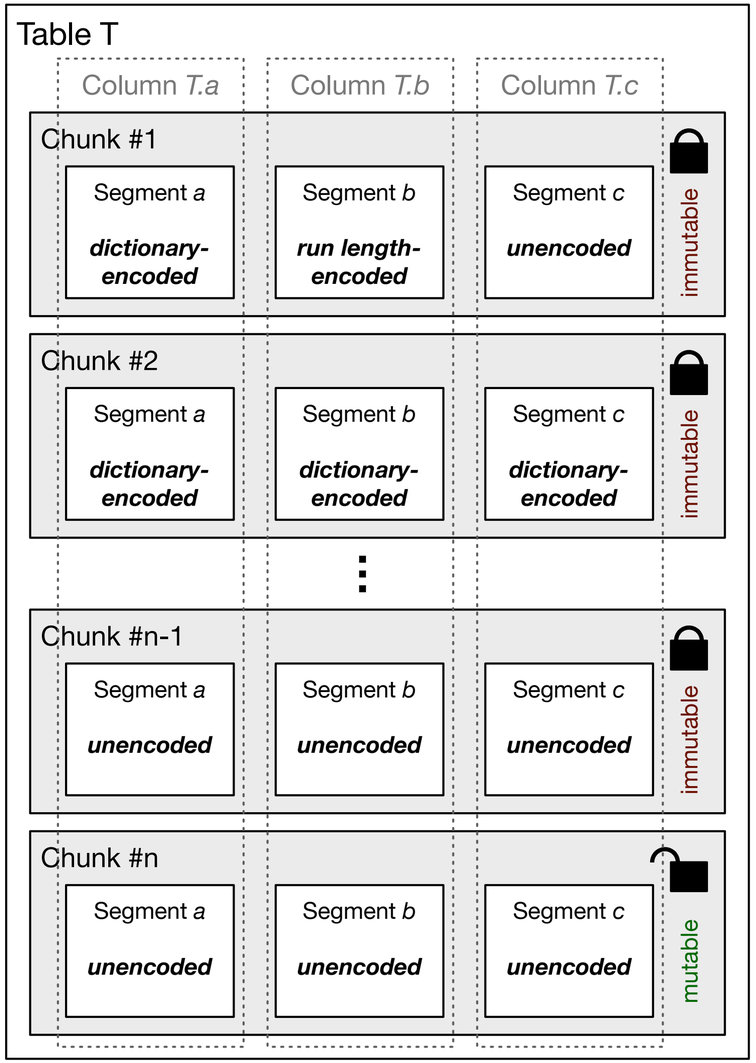


Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Storage Layout

- Segments can be **encoded independently**
- On top of the segment encoding type, internal **integer vectors can be further compressed**
  - e.g., the offset vector of a dictionary encoded segment
  - storing offsets as `uint8_t` for small dictionaries



Storage figure: Dreseler et al.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. EDBT 2019: 313-324

[1] Hyrise on GitHub: https://git.io/hyrise

Hyrise - Intermediate Results

- Hyrise tries to materialize results as late as possible
- Intermediate results are passed to following operators in form of **position lists**:
  - store pairs of `<chunk_id, row_id>` that reference the initial data table
- Effect:
  - small intermediates are passed (small payloads for joins etc.)
  - potentially expensive materialization when position lists is scrambled (e.g. after a join)

[1] Hyrise on GitHub: https://git.io/hyrise

Integer Encodings

Integer Encoding Libraries

Integer Encoding Libraries

TurboPFor: Fastest Integer Compression build passing

- **TurboPFor: The new synonym for "integer compression"**
  - 🆕 (2019.11) **ALL** functions now available for **64 bits ARMv8 NEON** & **Power9 Altivec**

Integer Encoding Libraries

TurboPFor: Fastest Integer Compression build passing

- **TurboPFor: The new synonym for "integer compression"**
  - 🆕 (2019.11) **ALL** functions now available for **64 bits ARMv8 NEON** & **Power9 Altivec**

The FastPFOR C++ library : Fast integer compression

build passing build failing Ubuntu-CI passing

Integer Encoding Libraries

TurboPFor: Fastest Integer Compression

build passing

- TurboPFor: The new synonym for "integer compression"
  - 🆕 (2019.11) **ALL** functions now available for **64 bits ARMv8 NEON** & **Power9 Altivec**

The FastPFOR C++ library : Fast integer compression

build passing | build failing | Ubuntu-CI passing

streamvbyte

Ubuntu 20.04 CI (GCC 9) passing | VS16-CI passing

Integer Encoding Libraries

TurboPFor: Fastest Integer Compression

build passing

- TurboPFor: The new synonym for "integer compression"
  - NEW (2019.11) **ALL** functions now available for **64 bits ARMv8 NEON** & **Power9 Altivec**

The FastPFOR C++ library : Fast integer compression

build passing | build failing | Ubuntu-CI passing

streamvbyte

Ubuntu 20.04 CI (GCC 9) passing | VS16-CI passing

Oroch

A C++ library for integer array compression.

SIMDCompressionAndIntersection

build passing | code quality: c/c++ A+

MaskedVByte

build passing | code quality: c/c++ A+

dictionary

High-performance dictionary coding

Compact vector

This library provide a bit-packed vector like data structure for storing integer types.

Integer Encoding Libraries

- From libraries stem from the field of information retrieval
  - optimized for set operations
  - optimized for sequential operations
- Only few support efficient random access to single positions
  - relevant for Hyrise's processing of position lists

Integer Encoding Libraries

Name	License	Technique	Language	Random Access	Docu-mented	Tested & Matured	Platforms
FastPFor	Apache 2.0	FoR	C++	○	●	●	x64/SSE3
TurboPFor	GPL	FoR, NS	C	◐	○	●	x64/AVX2, ARM/NEON, Power9
SIMDCompressionAnd-Intersection (SIMDCAI)	Apache 2.0	FoR, NS	C++	◐	●	●	x64/SSE4.1
MaskedVByte	Apache 2.0	NS	C	●	●	●	x64/SSE4.1
StreamVByte	Apache 2.0	NS	C	○	●	●	x64/SSE4, ARM
Oroch	MIT	FoR, NS	C++	●	●	●	cross-platform
dictionary	Apache 2.0	DICT	C++	●	●	○	x64/AVX512
compact_vector	MIT	NS	C++	●	●	●	cross-platform

Table 1: Open-source Lightweight Integer Encoding Implementations

Integer Encoding Libraries

Random Access	<technique>_<library> [_<specialization>]	Sequential Decoding	PosList Decoding by Length				Compression Rate
			1	10^3	10^4	10^6	
●	*for_SIMDCAI_simd*	10.1	0.007	6.0	60.1	6044	0.26
	pfor_turboPFOR	54.3	0.006	5.0	55.1	5494	0.34
	bitpacking_turboPFOR	29.0	0.007	2.5	40.0	3053	0.38
	bitpacking_compactvector	161.3	0.005	3.0	32.7	6976	0.39
	bitpacking_Oroch_intArray	14 820.3	0.29	245.4	2308.0	204 491	0.57
	Unencoded (`std::vector<T>`)	9.3	0.002	0.8	15.5	1470	1.0
○	*pfor_fastPFOR*	10.7	10.5	11.3	16.2	854	0.34
	bytepacking_MaskedVByte	87.1	124.7	73.9	87.5	1742	0.57
	bytepacking_StreamVByte	17.4	14.8	20.9	22.4	1991	0.60

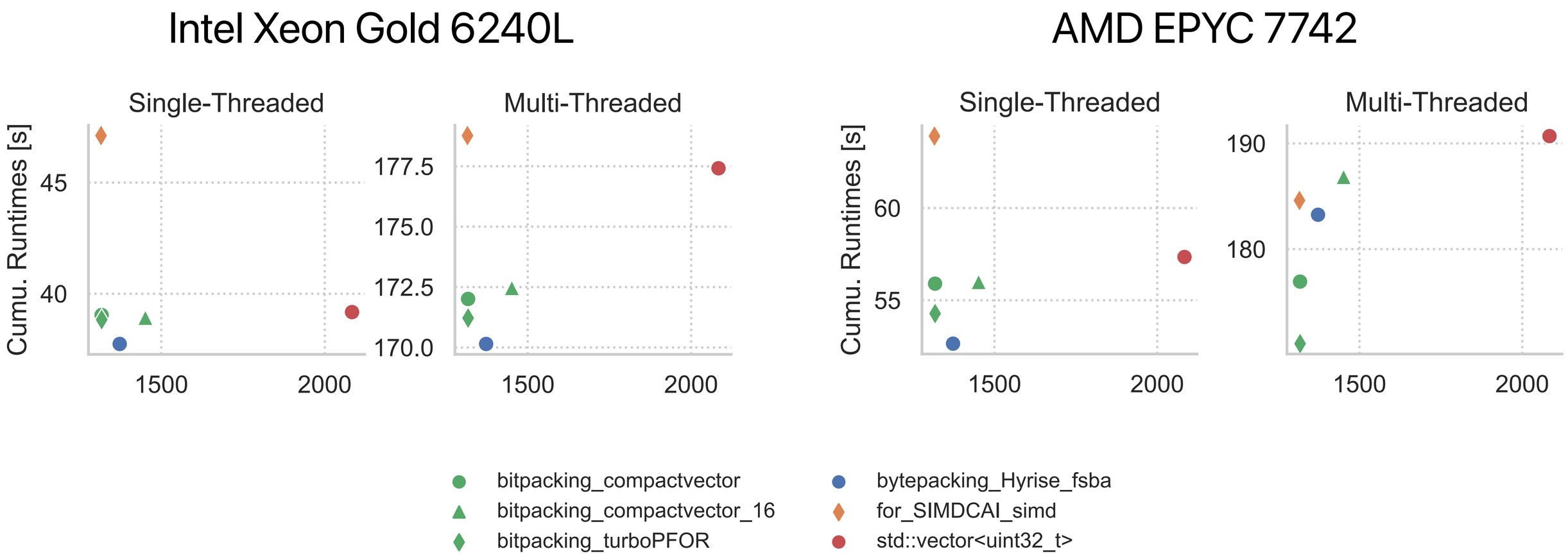
Table 2: CPU time in microseconds for decoding the whole vector, per library. For sequential decoding, runtime measured as mean across all data distributions. Compression rate as average across data distributions.

End-to-End Evaluation

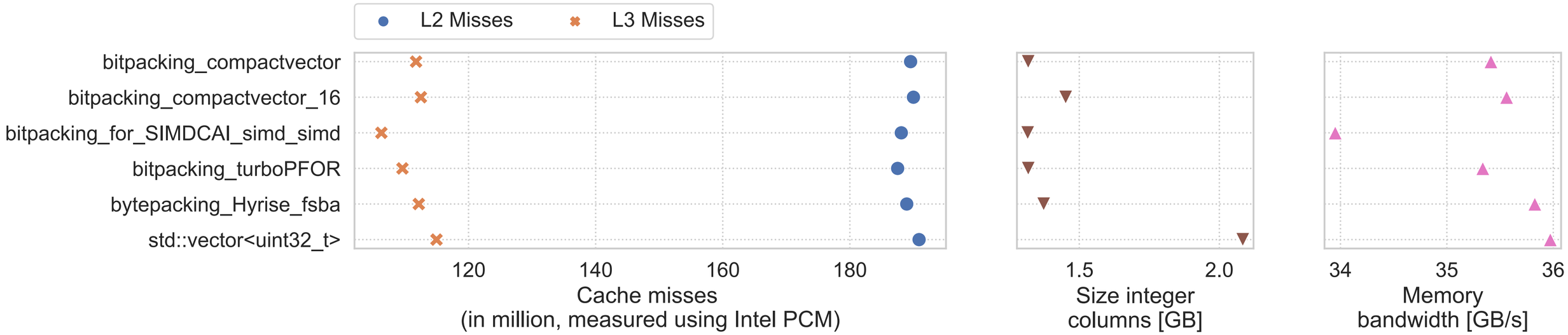
We only show the vector compression evaluations and skip the segment encoding analysis in this presentation. See the paper for all analyses.

TPC-H and DS use a scale factor of 10
Multi-Threaded: slightly more concurrent clients than threads to maximize load

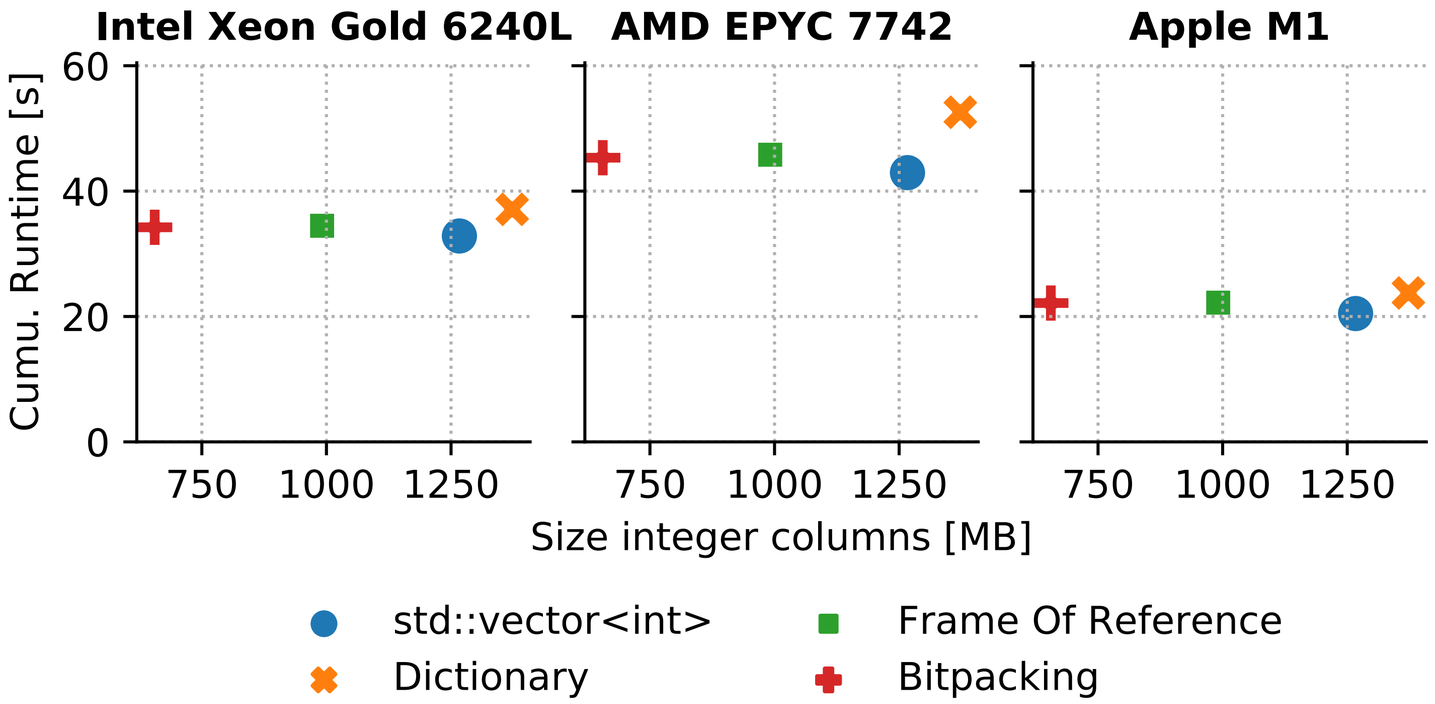
TPC-H - Vector Compression



TPC-H - Performance Counters



TPC-H - Single-Threaded



Lessons Learned

Lessons Learned

- Integers often contribute only a small share to the overall memory consumption
  - different for processing time
  - even more so, when data is tiered/buffered

Lessons Learned

- Integers often contribute only a small share to the overall memory consumption
  - different for processing time
  - even more so, when data is tiered/buffered
- Balancing performance and memory footprint is tough
  - single-threaded performance can differ significantly from multi-threaded
  - highly depends on the load (50% load is more comparable to ST than MT)

Lessons Learned

- Integers often contribute only a small share to the overall memory consumption
  - different for processing time
  - even more so, when data is tiered/buffered
- Balancing performance and memory footprint is tough
  - single-threaded performance can differ significantly from multi-threaded
  - highly depends on the load (50% load is more comparable to ST than MT)
- Hard to generalize performance insights
  - Calibration/learned prediction models needed for selection

Lessons Learned

- Integers often contribute only a small share to the overall memory consumption
  - different for processing time
  - even more so, when data is tiered/buffered
- Balancing performance and memory footprint is tough
  - single-threaded performance can differ significantly from multi-threaded
  - highly depends on the load (50% load is more comparable to ST than MT)
- Hard to generalize performance insights
  - Calibration/learned prediction models needed for selection

Thank you