Columnar In-Memory Database Systems

Trends and Concepts of Business Application Architecture

Stefan Halfpap, Michael Perscheid, Ralf Teusner, Werner Sinzig

Enterprise Platform and Integration Concepts

Hasso-Plattner-Institut

Course Overview

Trends and Concepts of Business Application Architecture

- Digitalization of Business Processes
- Enterprise Resource Planning
- **Columnar In-Memory Database Systems for Business Applications**
  - **Columnar In-Memory Database Systems**
  - Query Execution on Compressed Data
- Customer Relationship Management
- Enterprise Cloud Platforms for Integration and Extensions
- Block Week: Architecture Deep Dives

Agenda

- Relational Database Systems (Motivation/Recap)
- Database System Landscape for Business Applications
- Preliminary Knowledge for Database Table Layouts
  - Memory Hierarchy
  - Table Representation
- Row-Oriented Disk-Based Database Systems
- Columnar In-Memory Database Systems
- Summary

Relational Database Systems for Business Applications
Motivation (Recap)

- „Enterprise applications are about the **display, manipulation, and storage of large amounts of often complex data** and the **support or automation of business processes with that data**.“ Martin Fowler „Patterns of Enterprise Application Architecture Patterns“ (2002)
  - Large and complex data sets, that model real-world entities, e.g., journal entries or customers
  - Support and automate business processes on basis of this data
  - Typically use **relational database systems**

customer_id	name	country	zip_code	city	...
1	Ostseerad	Germany	18724	Zingst	:
2	Silicon Valley Bikes	USA	94304	Palo Alto	:
:	:	:	:	:	:

Extract of an exemplary customer table

Relational Database Systems for Business Applications
Motivation (Recap)

- Business applications use **relational database systems**
  - Flexible/powerful relational model (tables, operations, constraints)
  - Performant queries due to many optimizations
  - Consistent data and query results

- Business applications issue **transactional and analytical queries**;
  In particular, analytical demand has increased and is increasing
  For both workload types, **read queries dominate** (SQL SELECTs);

- Tables for business applications are **wide and sparse**

Relational Database Systems
Database System Landscape – Historical Background

Michael Stonebraker „One Size Fits All: An Idea Whose Time Has Come and Gone“ (2005)

- First Relational Database systems were developed in the 1970ies as research prototypes
  - System R
  - INGRES

Genealogy of relational database systems
https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/projekte/RDBMSGenealogy/RDBMS_Genealogy_V6.pdf

- Both systems were developed for transactional data processing (OLTP) and **row-oriented disk-based database systems**
- Since beginning of the 1980ies: „**one size fits all**“ strategy
  Reasons: Costs, Compatibility, Sales, Marketing
- Since the 1990ies: Enterprises increasingly deploy specialized analytical database systems (so called **data warehouses)** for performant and flexible analytical queries (without additional indices and materialized views) and data integration
- Since the 2000ies: Michael Stonebraker „One Size Fits All: An Idea Whose Time Has Come and Gone“
  Michael Stonebraker „The End of an Architectural Era (It's Time for a Complete Rewrite)”

Specialized and universal main memory database systems enabled by HW advances

Database System Landscape for Business Applications
Data-Warehouse: Motivation and Creation

S. Chaudhuri et U. Dayal „An Overview of Data Warehousing and OLAP Technology " (1997)

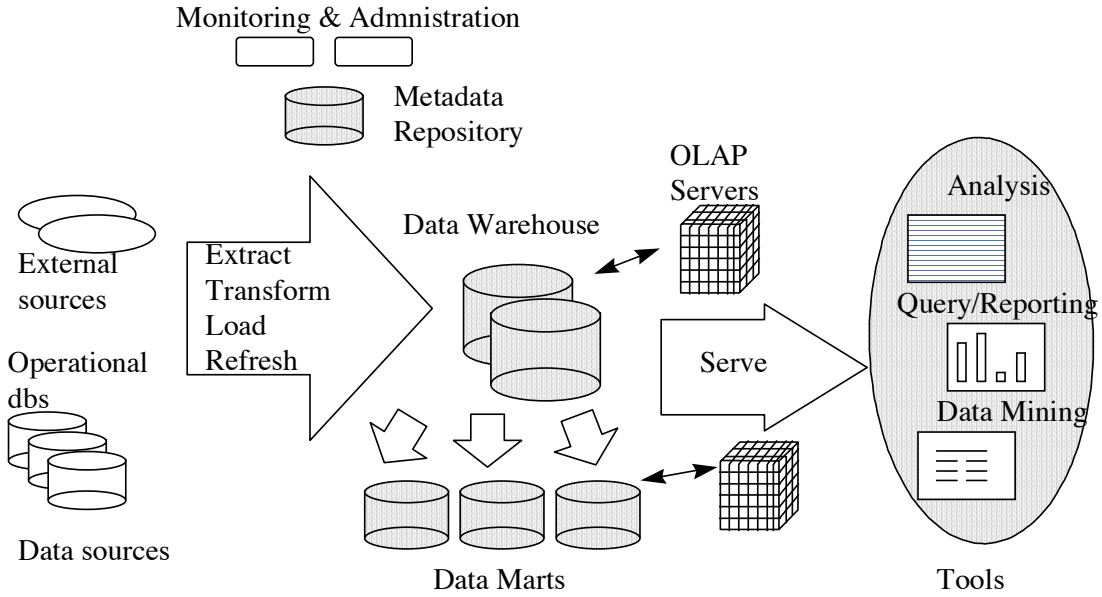
- Reasons for development and usage: **performance** and **data integration**
  - Historically, relational database systems were developed/optimized for OLTP:
    Fear of performance degradations for transactions due to complex ad-hoc queries
  - Keep historical data + additional information from external sources, e.g., stock prices

- The deployment of a data warehouse (data modelling and integration) is for enterprises a long and complex process
- Requires a comprehensive business process modelling
  (Coverage of entire enterprise)
- Data-Marts: integrate data subsets that focus on single organizational units, e.g., marketing

Database System Landscape for Business Applications
Data-Warehouse: Overview

S. Chaudhuri et U. Dayal „An Overview of Data Warehousing and OLAP Technology " (1997)

- Data integration
- Data modelling
- Operations
- Data storage
- Auxiliary structures



Database System Landscape for Business Applications
Separation of OLTP and OLAP

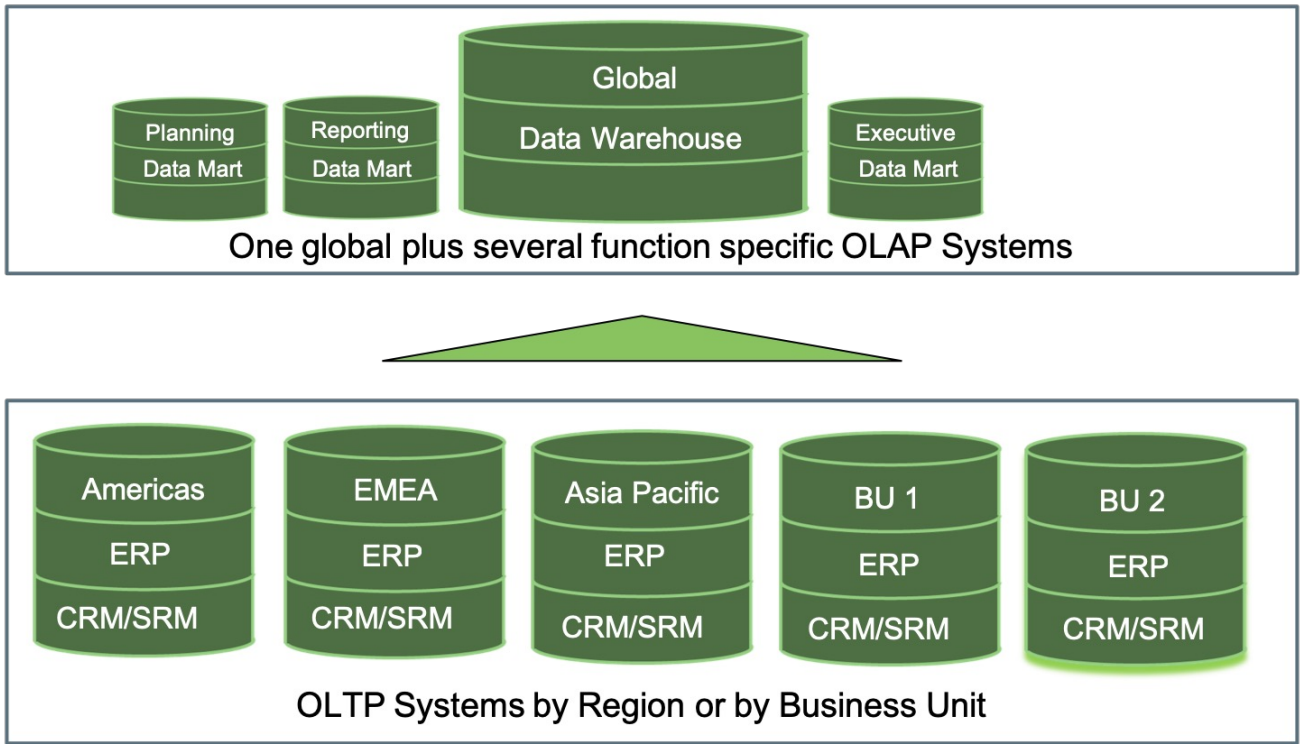
- Business applications use transactional (OLTP) and analytical (OLAP) database queries
- But: Requirements for OLAP and OLTP database systems differ

(Historical) Approach:

Separation in distinct database systems (tuning of the system (e.g., storage layout) and schema)

- **Row-oriented disk-based database system**
  with many indices and materialized views for frequent reports
- Data warehouse for OLAP performance and data integration

Database System Landscape for Business Applications
Separation of OLTP and OLAP: Distinct Database Systems



Database System Landscape for Business Applications
Separation of OLTP and OLAP: Complex OLTP Schema

Hasso Plattner „The Impact of Columnar In-Memory Databases on Enterprise Systems " (2014)

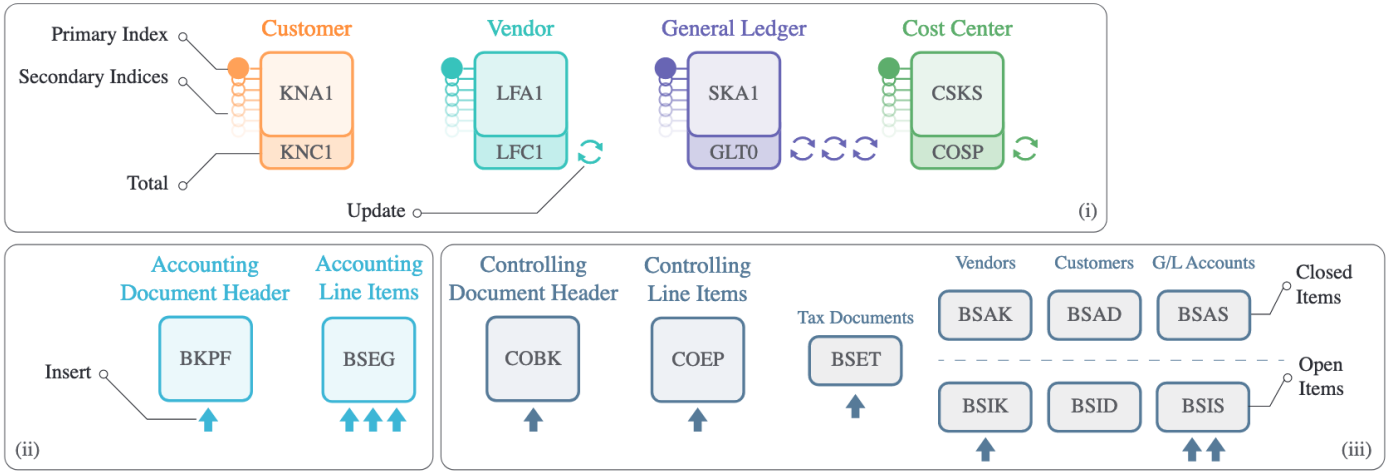


Figure 3: Selected tables of the SAP Financials data model, illustrating inserts and updates for a vendor invoice posting: (i) Master data for customers, vendors, general ledger and cost centers with transaction-maintained totals. (ii) Accounting documents. (iii) Replicated accounting line items as materialized views.

Database System Landscape for Business Applications
Separation of OLTP and OLAP: Disadvantages

- Data redundancy
- Data consistency is difficult to maintain
- Synchronization of the OLAP systems with ETL (extract, transform, load) process
  - High costs
  - Time delay
- No flexible reports/analytics on transactional schema
- Differing data schemata of the OLTP and OLAP system increase the complexity of applications that use both systems

Database System Landscape for Business Applications
Combination of OLTP and OLAP in a Single System

- Business applications use transactional (OLTP) and analytical (OLAP) database queries
- But: Requirements for OLAP and OLTP database systems differ

(Modern) Approach:

Optimize a system for both workload types (with compromises)

- **Columnar in-memory database system**
- (Data warehouse for data integration)

Database System Landscape for Business Applications Combination of OLTP and OLAP in a Single System

- A single columnar in-memory database system as "single source of truth"
  - No data redundancy
  - No consistency and synchronization issues between different systems
  - Enables ad-hoc analyses based on the transactional data without materialized views (e.g., precomputed aggregates) – "mixed workloads"

→ Simplified applications (against a single database system with a single schema) and database system maintenance (fewer indices, no materialized views)

- Enabled by hardware advancements: fast multicore CPUs + larger and cheaper main memory
- Required a complete redesign and rewrite of the database system

Database System Landscape for Business Applications

Combination of OLTP and OLAP in a Single System: Simplified Schema

Hasso Plattner „The Impact of Columnar In-Memory Databases on Enterprise Systems " (2014)

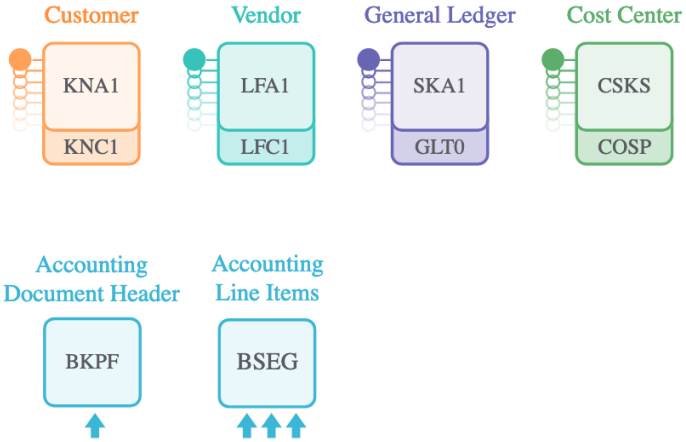


Figure 4: Simplified SAP Financials data model on the example of a vendor invoice posting illustrating the remaining inserts for the accounting document.

Database System Landscape for Business Applications
Combination of OLTP and OLAP in a Single System

Tradeoffs:

- For more restricted workload characteristics (i.e., OLTP or OLAP), more specialized/optimized systems can be built

  Michael Stonebraker „One Size Fits All: An Idea Whose Time Has Come and Gone" (2005)

- Storing all data permanently in main memory is a cost-efficiency trade-off: idea: offloading cold data to cheaper storage

  David Lomet „Caching Data Stores: High Performance at Low Cost" (2018)

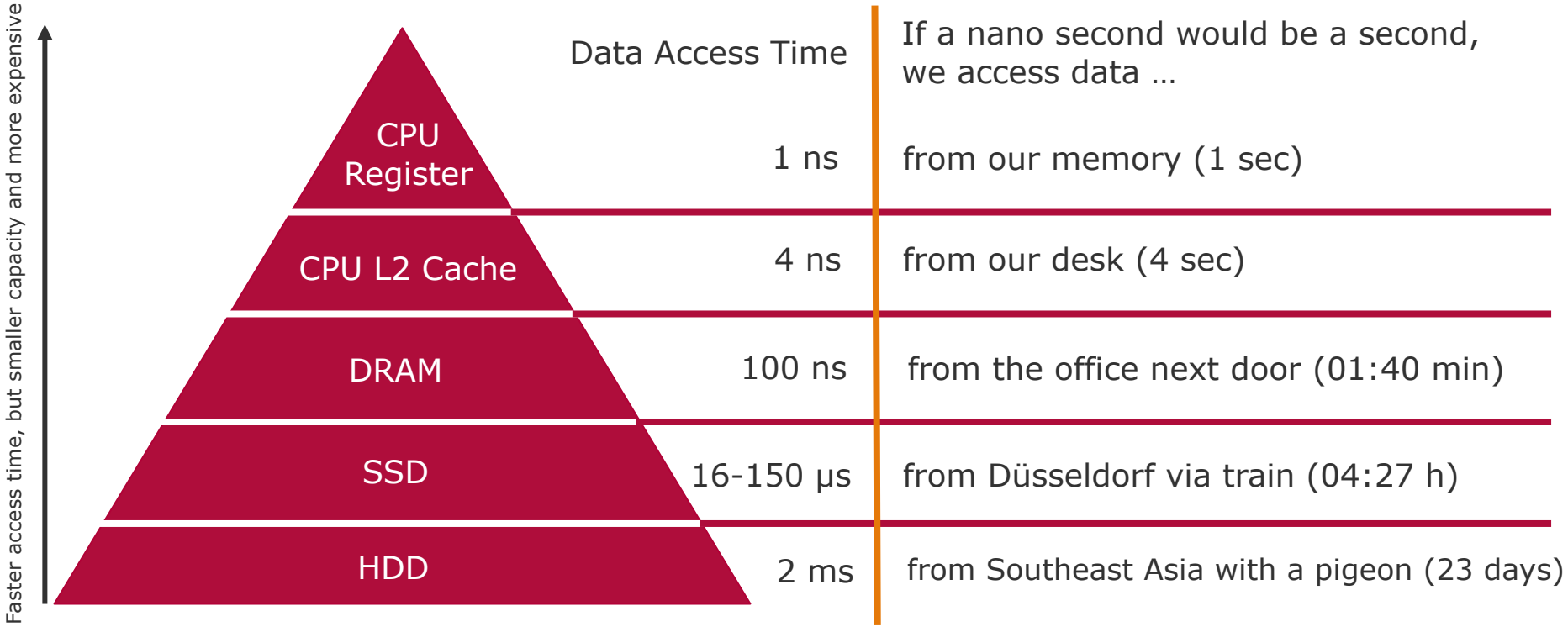
Database System Research:

- Cloud promises (cost-effectiveness, scalability, elasticity), auto tuning
- Disk-based systems with near in-memory performance at lower costs, e.g., using multiple fast SSDs and light-weight buffer management

Agenda

- Relational Database Systems (Motivation/Recap)
- Database System Landscape for Business Applications
- **Preliminary Knowledge for Database Table Layouts**
  - Memory Hierarchy
  - Table Representation
- Row-Oriented Disk-Based Database Systems
- Columnar In-Memory Database Systems
- Summary

Preliminary Knowledge: Memory Hierarchy „Latency Numbers Every Programmer Should Know“

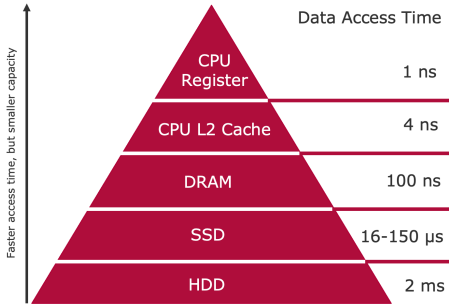


http://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html (Numbers from 2020)

http://www.montana.edu/cpa/news/wwwpb-archives/yuth/pigeon.html

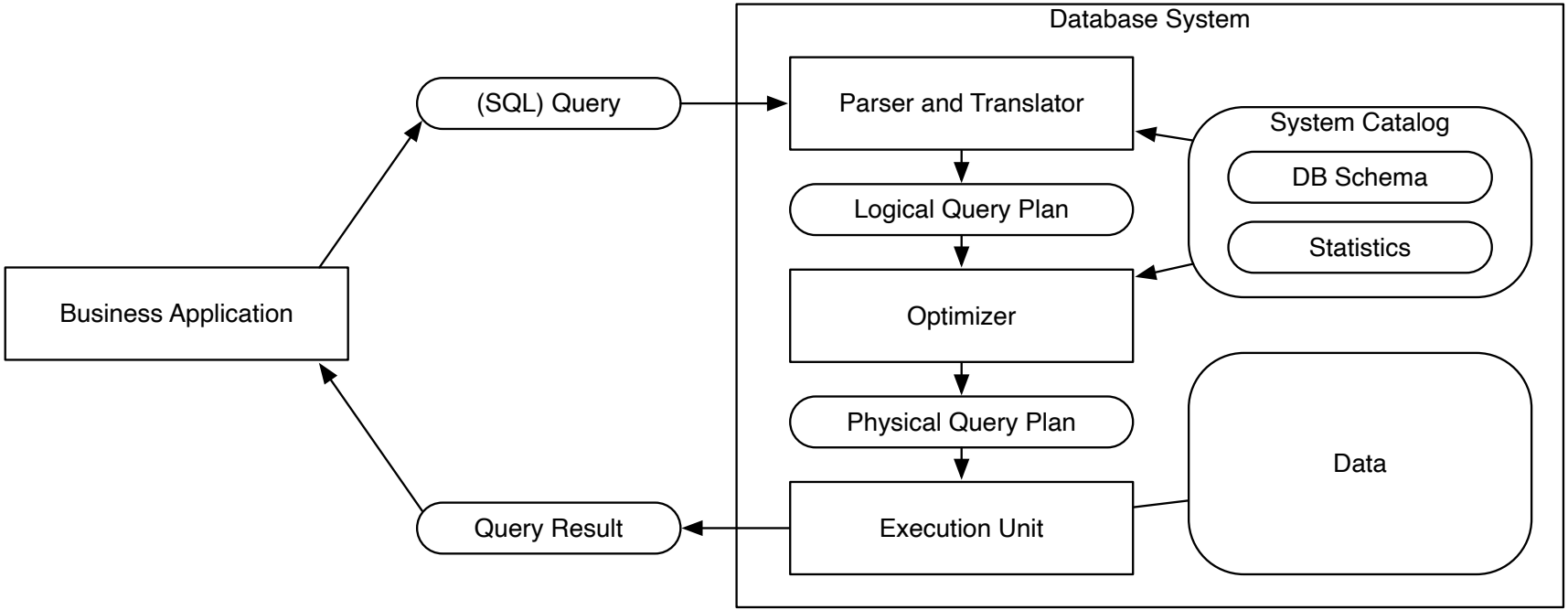
Preliminary Knowledge: Memory Hierarchy Implications for Programming

- Memory levels use smaller and faster levels as caches
- Data is transferred between levels in chunks
  - CPU-Caches: cache line (e.g., 64 bytes for current CPUs)
  - DRAM: page (often 4 KiB as default)
    (1 kB = 1000 bytes 1 KiB = 1024 bytes)



"All programming is an exercise in caching."
Terje Mathisen

Preliminary Knowledge
Query Processing – Overview (Recap)



Preliminary Knowledge
Table Representation

- Relational model as conceptional/logical data model with abstract operations

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

- How to represent/store data in memory?
- How to efficiently process queries in memory?
  (tight coupling, because memory/storage access is usually a performance bottleneck)

Table Representation
Data Types

- SQL Data Types:
  - Character types with fixed or variable length
  - Numeric types with exact values
  - Numeric types with approximated values
  - Boolean
  - Date and time types
  - Large objects
  - …
  - User defined types

Table Representation
Data Type Representation

- Single attribute values are eventually bit sequences in memory
  - Specific bit sequence of values depends on implementation, usually based on native programming language (e.g. C/C++) types
  - Encodings/compression possible

- The Database is a organized collection of attribute values (bit sequences); + schema and meta information (which encoding, which data layout); alignment and padding possible

- Data types with fixed vs. variable length
  - Bit sequences with a fixed length enable direct access
  - Bit sequences with a variable length are more space efficient

Table Representation
Data Type Representation: NULL

- **Variant 1: Bitmap per column or row**

- Variant 2: Indication next to the attribute value

  → May require 1 byte (or more) with alignment instead of 1 bit (or reduces the domain)

- Variant 3: Special value
  - Use a special value for individual data types or attributes (e.g., INT32_MIN)

    → reduces the domain

Table Representation
Conceptual Model vs. Physical Storage

- How do we map **two-dimensional tables** to the linear (**one-dimensional) address space** (to exploit the memory hierarchy as good as possible)?
- i.e., the physical implementation of the conceptual relational data model

Conceptual Model

First Name	Last Name	Country	Year of Birth
Paul	Smith	Australia	1986
Lena	Jones	USA	1990
Hanna	Schulze	Germany	1942
Hanna	Schulze	USA	2000

Physical Storage

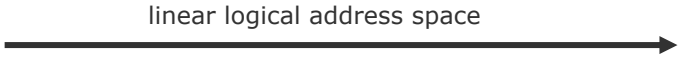
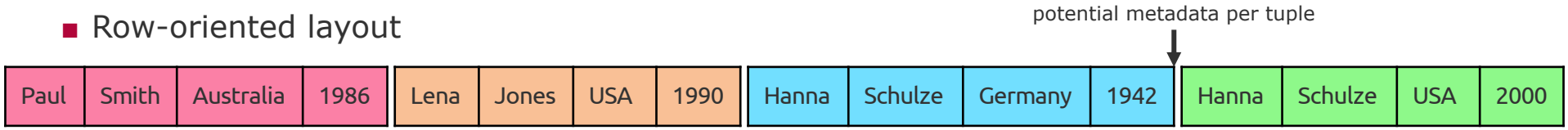
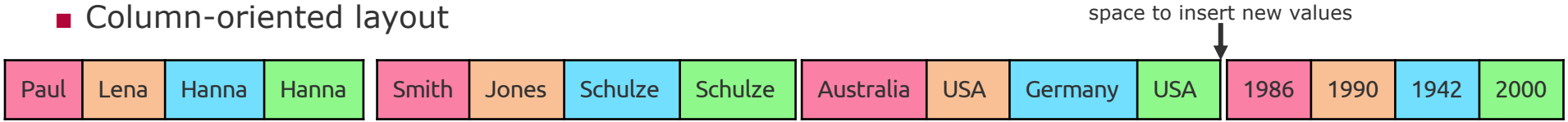


Table Representation (Physical Storage)
Row-Oriented vs. Column-Oriented vs. Hybrid Layout

- Row-oriented layout



- Column-oriented layout



- Hybrid layout (using attribute groups)

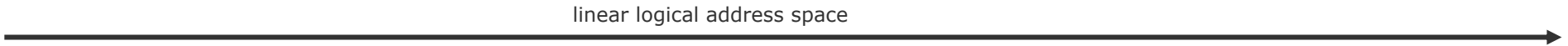
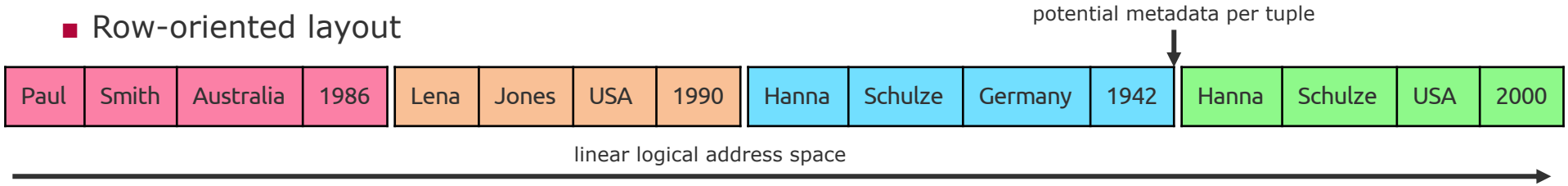


Table Representation
Row-Oriented Layout

- Row-oriented layout



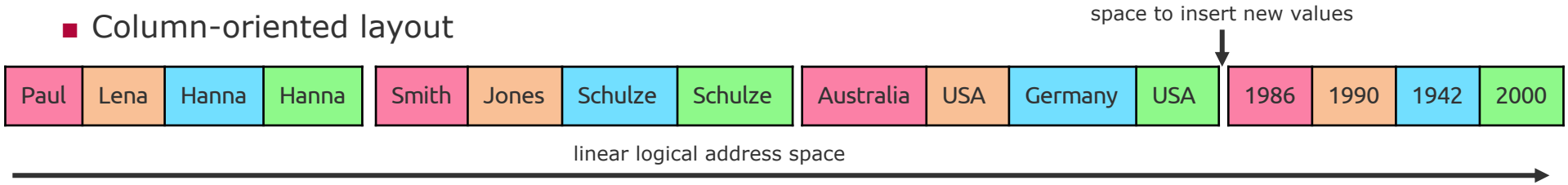
- Stores all attribute values of the same tuple consecutively

  (for variable-length values or large objects, references may be useful)

- Suitable for transaction processing, for which queries process single or only a few tuples (Note, an **index is required for efficient tuple retrievals**) or insert many tuples

Table Representation
Column-Oriented Layout

- Column-oriented layout

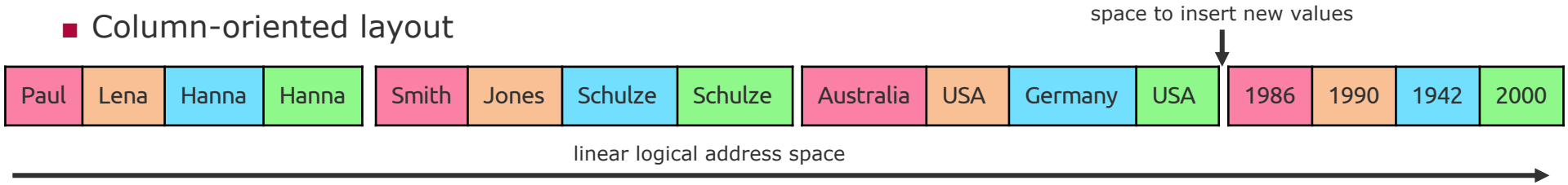


- Stores all attribute values of the same attribute consecutively
  (for variable-length values or large objects, references may be useful)

- Suitable for analytical queries that read large data parts of few attributes

Table Representation
Column-Oriented Layout

- Column-oriented layout



- Tuple reconstruction
  - **Variant 1: Fixed-width offsets**
  - Variant 2: Explicitly stored ID

- **Better compression possible compared to row-oriented layout**

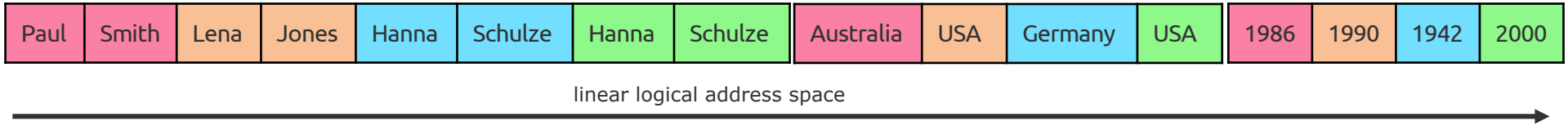
Table Representation
Column-Oriented Layout - History

- 1970ies: Cantor DBMS
- 1980ies: Setrag Khoshafian „A Query Processing Strategy for the Decomposed Storage Model“
- 1990ies: SybaseIQ (Heute: SAP IQ)
- 2000ies: Vertica, MonetDB
- 2010er: **Hyrise** (new version), SAP HANA, Cloudera Impala, Amazon Redshift, MemSQL, ...

Andy Pavlo: „Advanced Database Systems: Storage Models & Data Layout (Spring 2019)“
https://15721.courses.cs.cmu.edu/spring2019/schedule.html

Table Representation
Hybrid Layout

- Hybrid layout

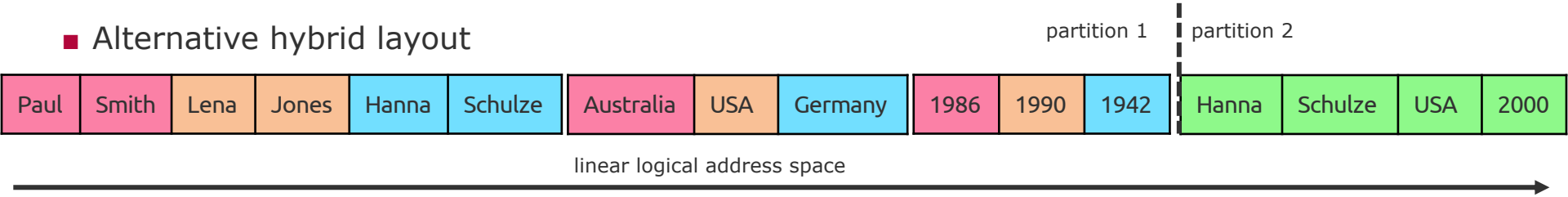


- Stores all attribute values of the same attribute group consecutively
  (for variable-length values or large objects, references may be useful)

- Attribute groups of a single table may differ per partition
- Idea: **optimized layout** depending on the data **access pattern**
  But: **Increased complexity**, e.g., for implementation, query-optimization, execution engine

Table Representation
Hybrid Layout – Different Attribute Groups

- Alternative hybrid layout



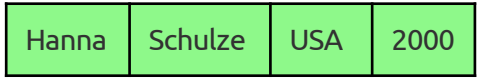
- Exemplary idea:
  - Layout of older tuples is optimized for analytical workloads (partition 1)
  - New tuples are stored in row format for fast inserts (partition 2)

- Dynamic layout changes over time possible

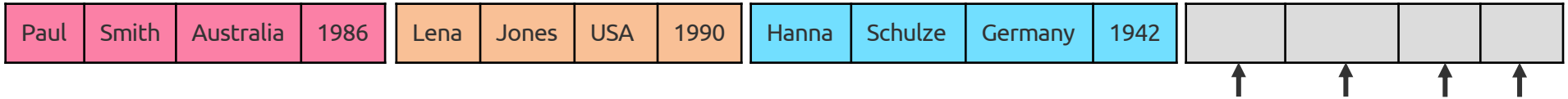
Table Representation
Hybrid Layout– Hyrise

- Grund et al.: "HYRISE - A Main Memory Hybrid Storage Engine" (2010)
  - http://www.vldb.org/pvldb/vol4/p105-grund.pdf
  - At HPI developed research in-memory database system, which supports flexible **hybrid table layouts** https://github.com/hyrise/hyrise-v1
- Dreseler et al.: "Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management."
  - https://doi.org/10.5441/002/edbt.2019.28
  - **Columnar** research in-memory database system https://github.com/hyrise/hyrise
  - Reasons for rewrite:
    - Resolution of data layout abstractions at runtime too expensive
    - Many prototypical components that do not interact well
    - Accumulated technical deb
    - Lack of SQL support

Table Representation: Row Operation
Example – Insert a new Entry



- Row-oriented layout

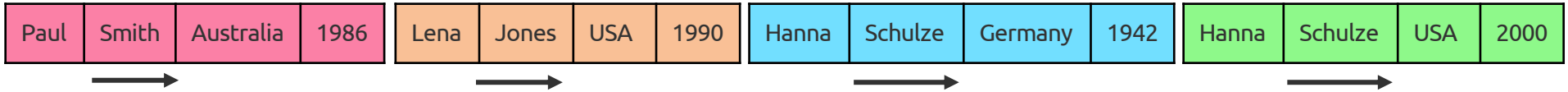


- Column-oriented layout

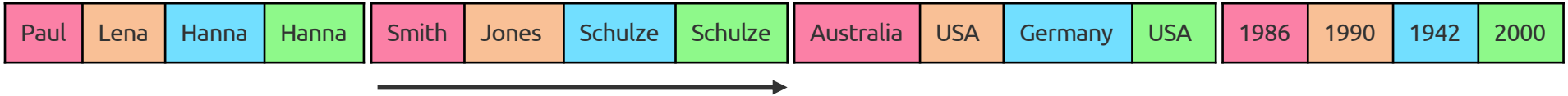
- Hybrid layout

Table Representation: Column Operation Example – Filter a Table by Last Name

- Row-oriented layout



- Column-oriented layout



- Hybrid layout

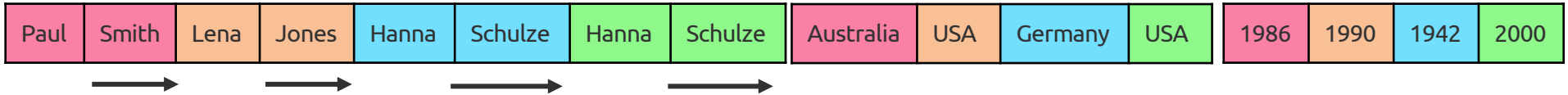
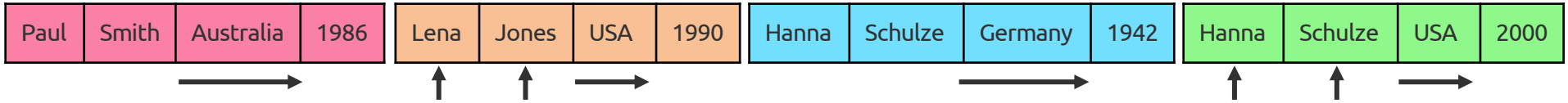
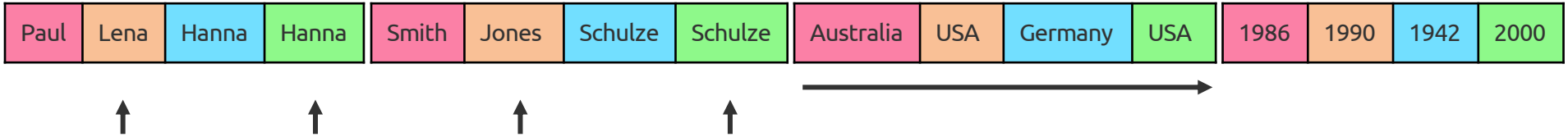


Table Representation: Combined Operation Example – Names of all Persons from the USA

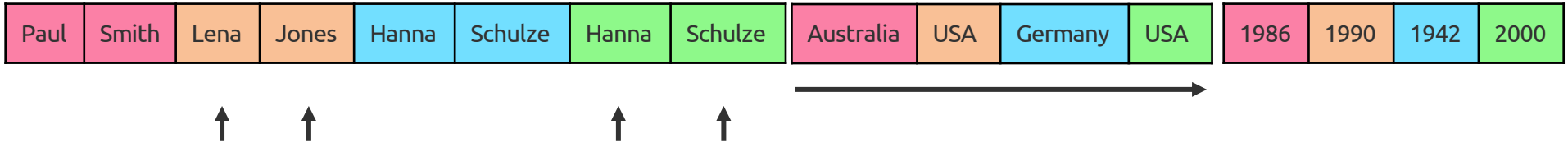
- Row-oriented layout



- Column-oriented layout



- Hybrid layout



Agenda

- Relational Database Systems (Motivation/Recap)
- Database System Landscape for Business Applications
- Preliminary Knowledge for Database Table Layouts
  - Memory Hierarchy
  - Table Representation
- **Row-Oriented Disk-Based Database Systems**
- Columnar In-Memory Database Systems
- Summary

Row-Oriented Disk-Based Database Systems
Motivation

- **Disks as primary storage** and **row orientation** originate from historic/earlier systems/requirements
  - Limited hardware resources, e.g., single CPU, single-core CPU, small and expensive main memory

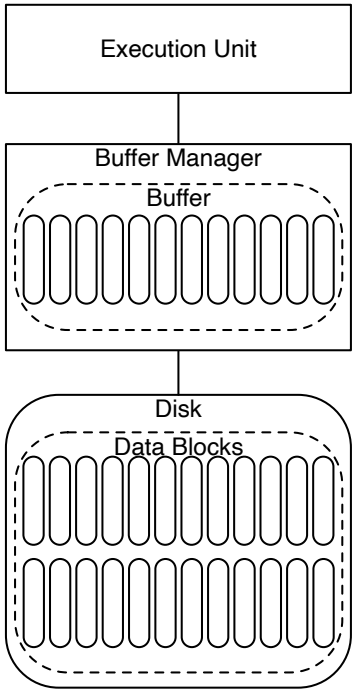
    → Databases must be stored on disk
  - Row orientation (in combination with indices) is better suitable for OLTP, which is the historically more important workload class

Row-Oriented Disk-Based Database Systems
Motivation - Disks as Primary Storage

- Disks is the primary memory level for storing data
- Data is loaded into higher levels (e.g., main memory) for processing
- Challenges:
  - Main memory is limited, but performance is important (goal: minimize disk accesses)
  - Data modification must be stored on persistently
  - Concurrency control (multiple execution units (threads))

Row-Oriented Disk-Based Database Systems
Buffer Manager

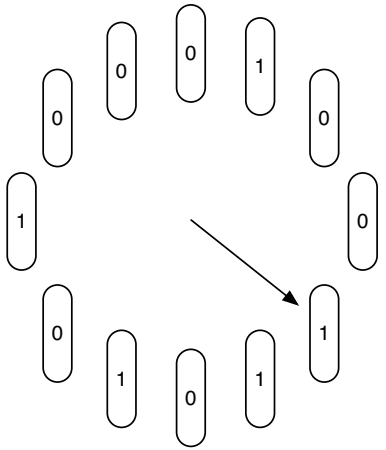
- Database systems use (some kind of) a **buffer management** for all memory accesses of the execution unit
- Data is (usually) divided into **blocks** of a fixed size (often 4 – 16 KiB)
- The database system uses a buffer to cache the blocks in main memory
- The buffer manager checks if a block is already in main memory
  - If not, the buffer manager reads the block into main memory
    - If there is no space (in the buffer or main memory), another block must be replaced
    - If the block that will be replaced was changed, it must be written to disk



The database system has more information than the operating system

- Block pinning
- Batched and forced writes to disk
- Block prefetching
- Block replacement strategies (Overhead vs. effectivity):
  - FIFO (first-in-first-out)
  - LRU (least recently used)
  - Clock („second chance“)
  - Toss-immediate
  - MRU (Most recently used)
  - Random

useful for nested-loop joins: blocks of outer loop; blocks of inner loop



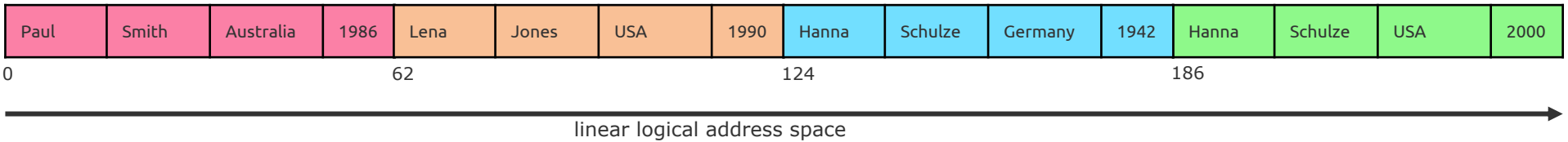
Row-Oriented Disk-Based Database Systems
Block Layout – Data Types with a Fixed Length

- Should tuples/entries span multiple blocks? Usually not.
- Size of an entry:

First Name	20 bytes
Last Name	20 bytes
Country	20 bytes
Year Of Birth	2 bytes

Overall size	62 bytes

→ 132 entries per 8 KiB block



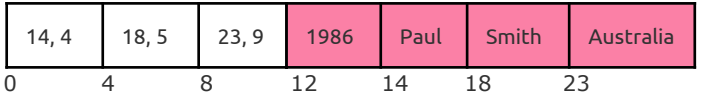
Row-Oriented Disk-Based Database Systems
Block Layout – Data Types with a Variable Length

- Data types with variable lengths and NULL values → variable-length entries
- **Besides: Fixed-size values waste space (disk-access is often the performance bottleneck)**
- Two challenges:
  - **Single attribute values** of entries should be retrievable efficiently (without reading/decoding the entire entry/tuple)
  - **Single entries** of blocks should be retrievable efficiently (without reading/decoding the entire block)

Row-Oriented Disk-Based Database Systems
Block Layout – Data Types with a Variable Length and NULL Values

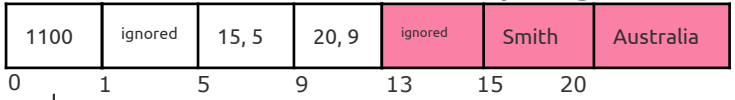
- **Single attribute values** of entries should be retrievable efficiently (different implementation possibilities; here an educational example)
  - Data types with a variable length: tuple (offset, length) references the attribute value
  - Data types with a fixed length: no additional Meta data (except schema information) and at the begin for direct access

bit representation of tuple



conceptual view of tuple

  - Different tradeoffs between retrieval-efficiency and memory consumption for NULL bitmap:
    1. for lower memory consumption: no data data for NULL values (set NULL bit)
    2. for retrieval-efficiency : ignore data for NULL values (set NULL bit)



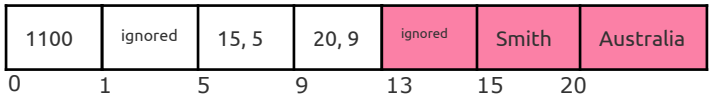
NULL-Bitmap in einem Byte gespeichert. (Here: Order corresponds to the physical storage order.)

conceptual view of tuple

Row-Oriented Disk-Based Database Systems
Block Layout – Data Types with a Variable Length and NULL Values

Exemplary (educational) tuple representation that allows an efficient access of all attribute values

bit representation of tuple



conceptual view of tuple

- The four NULL bits 1, 1, 0 and 0 belong to the attributes YearOfBirth, FirstName, LastName and Country
- Variable-length data types: Tuple (offset, length) reference the attribute value
  - set NULL bit for FirstName → metadata (here: bytes at offset 1 - 4) are ignored
    → enables direct access of metadata for the following variable-length attribute values
  - Attribute value (e.g., for FirstName) can be omitted if the NULL-Bit is set
- Fixed-length data types (i.e., YearOfBirth)
  - Without additional metadata (number of bytes (2 for YearOfBirth) is known/specified via the schema)
  - At the beginning for direct access
  - set NULL bit → data (here: bytes at offset 13 and 14) is ignored
    → enables direct access for the potentially following fixed-length attribute values

Row-Oriented Disk-Based Database Systems
Block Layout – Entries with Variable Length

- **Single entries** of blocks should be retrievable efficiently
  Slotted page structure: Use block header, which stores meta data:
  - Number of stored entries (or begin the "free space" block)
  - (The end of the "free space" block)
  - References to data of all stored entries
    - Indexes reference to these references in the block header and not to the entry data; this indirection allows moving entry data to avoid fragmentation
    - Can also indicate deleted values

Row-Oriented Disk-Based Database Systems
Block Layout - PostgreSQL

- The Internals of PostgreSQL
  http://www.interdb.jp/pg/pgsql01.html

- PostgreSQL 14 Documentation Chapter 70. Database Physical Storage
  https://www.postgresql.org/docs/14/storage.html

Agenda

- Relational Database Systems (Motivation/Recap)
- Database System Landscape for Business Applications
- Preliminary Knowledge for Database Table Layouts
  - Memory Hierarchy
  - Table Representation
- Row-Oriented Disk-Based Database Systems
- **Columnar In-Memory Database Systems**
- Summary

Columnar **In-Memory** Database Systems
Motivation

- Main memory has become larger and cheaper
- Many databases fit entirely into main memory
  - **Structured (e.g., relational) database** are smaller ← our focus
  - Unstructured or semi-structured are larger
- Disk-based database systems work,
  but their architecture brings overhead

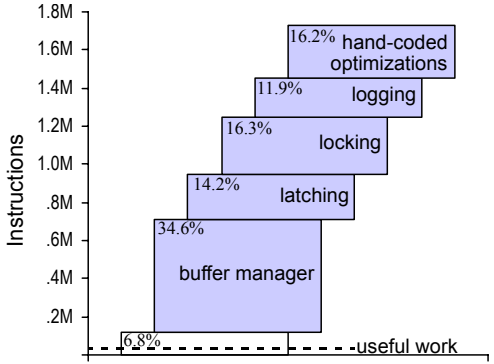


Figure 1. Breakdown of instruction count for various DBMS components for the New Order transaction from TPC-C. The top of the bar-graph is the original Shore performance with a main memory resident database and no thread contention. The bottom dashed line is the useful work, measured by executing the transaction on a no-overhead kernel.

Stavros Harizopoulos et al. „OLTP Through the Looking Glass, and What We Found There" (2008)

Hector Garcia-Molina et al. „Main Memory Database Systems: An Overview" (1992)

Columnar **In-Memory** Database Systems
Main Memory vs. Disk as Primary Storage

- **Main memory (or "in-memory") database systems optimize the data representation and access for a permanent data residence in memory:**
  - No buffer manager (overhead)
  - Avoid/reduce further locking and latching (which limit the scalability), e.g., by optimistic concurrency control
  - Leverage CPU caches:
    also for main memory, sequential access is faster than random access

Latch vs. Lock (siehe Graefe „A Survey of B-Tree Locking Techniques")
https://15721.courses.cs.cmu.edu/spring2016/papers/a16-graefe.pdf

Ulrich Drepper „What Every Programmer Should Know About Memory" (2007)
https://people.freebsd.org/~lstewart/articles/cpumemory.pdf

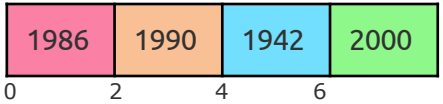
  - In-memory database systems still require persistent memory for logging and recovery, but approaches can be optimized, e.g., by group commits

Columnar In-Memory Database Systems
Best choice for Business Applications

- In particular, analytical processing demand for enterprise database systems is increasing
- Today: specialized (main memory) database systems
  - Row orientation better suited for OLTP
  - **Column orientation** (and hybrid layouts) **better suited for OLAP and mixed workloads** ← our focus (business applications)
    - Faster processing of set operation on few attributes
    - Better compression for wide and sparse data
- Hybrid layouts can be adapted to the workload, but their support increases the complexity significantly
- Pure column orientation is well-suited for processing OLTP and OLAP in a single (main memory) database systems and the primary choice in practice

Columnar In-Memory Database Systems
Table Representation (1/2)

- Goal: (fast scans and) **direct/efficient** (constant-time) **access** of attribute values
- Data blocks/columns store attribute values with a fixed length directly or references (with a fixed length) to attribute values in case of variable lengths
  - Column is implemented as vector/array



  - Columns with variable-length values use logical (offset) or physical references/pointers

→ Constant complexity for point accesses (Read offset and follow the reference)

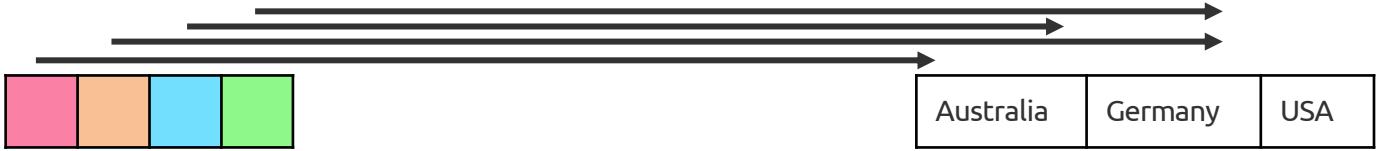
→ But indirection increases memory consumption and requires an additional memory access (Storage-efficiency-tradeoff)

(Usage of maximum value length and „small/short string optimization" (SSO) as alternatives)

Columnar In-Memory Database Systems
Table Representation (2/2)

- Additional NULL bitmap per column, if column allows NULL values

- Dictionary encoding produces attribute vectors with fixed-length values (see lecture „data compression“)



- Deletion of entries is usually implemented as invalidation, i.e., an additional vector indicates, whether an entry is valid or not

Columnar In-Memory Database Systems Summary

Primary data storage layer and table layout influence the implementation of a database system significantly:

- Architecture of traditional disk-based database systems limit the performance, even if the database fits entirely in memory
- Two-dimensional tables must be mapped to one-dimensional memory (address space) (Goal: Exploit the memory hierarchy and caching during query processing as good as possible)
  - Best data layout depends on the workload (set/sequence of all queries)
  - Column-oriented layouts are better suited for analytical and mixed workloads than row-oriented layouts
  - Data encodings/compression is a further influence factor for the choice of the data layout