Datenbankkonzepte für Unternehmensanwendungen: Spaltenorientierte Hauptspeicherdatenbanksysteme

Grundlagen von Unternehmensanwendungen

Stefan Halfpap, Ralf Teusner, Michael Perscheid, Werner Sinzig

Enterprise Platform and Integration Concepts

Hasso-Plattner-Institut

Vorlesungsinhalte

Grundlagen von Unternehmensanwendungen

- Einführung zu Unternehmensanwendungen
- Enterprise Resource Planning
  - Rechnungswesen und Planung
  - Kundenauftragsabwicklung
  - Materialwirtschaft: Einkauf und Produktionsplanung
  - Personalwesen
- Kundenbeziehungsmanagement (Customer Relationship Management; Gast: Prof. Carsten Hahn)
- **Datenbankkonzepte für Unternehmensanwendungen**
  - **Spaltenorientierte Hauptspeicherdatenbanksysteme**
  - Anfragebearbeitung auf komprimierten Daten
- Enterprise Cloud Plattformen zur Erweiterung und Integration von Unternehmensanwendungen

Überblick

- Relationale Datenbanksysteme
- Festplattenbasierte zeilenorientierte Datenbanksysteme
  - Buffermanagement
  - Tabellenrepräsentation
- Spaltenorientierte Hauptspeicherdatenbanksysteme
  - Optimierungen von Hauptspeicherdatenbanksystemen
  - Zeilenorientiertes vs. spaltenorientiertes vs. hybrides Layout
- Zusammenfassung
- Übungsblatt 5

Relationale Datenbanksysteme
Motivation

- „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)
  - Große komplexe Datenmengen, die in der realen Welt existierende Entitäten abbilden
  - Unterstützung/Automatisierung von Geschäftsprozessen auf Basis dieser Daten
  - Nutzen üblicherweise **relationale Datenbanken**

Kundennummer	Name	Land	PLZ	Stadt	...
1	Ostseerad	Deutschland	18724	Zingst	:
2	Silicon Valley Bikes	USA	94304	Palo Alto	:
:	:	:	:	:	:

Ausschnitt einer Kundentabelle

Relationale Datenbanksysteme
Datenbanksystemlandschaft - Historischer Hintergrund

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

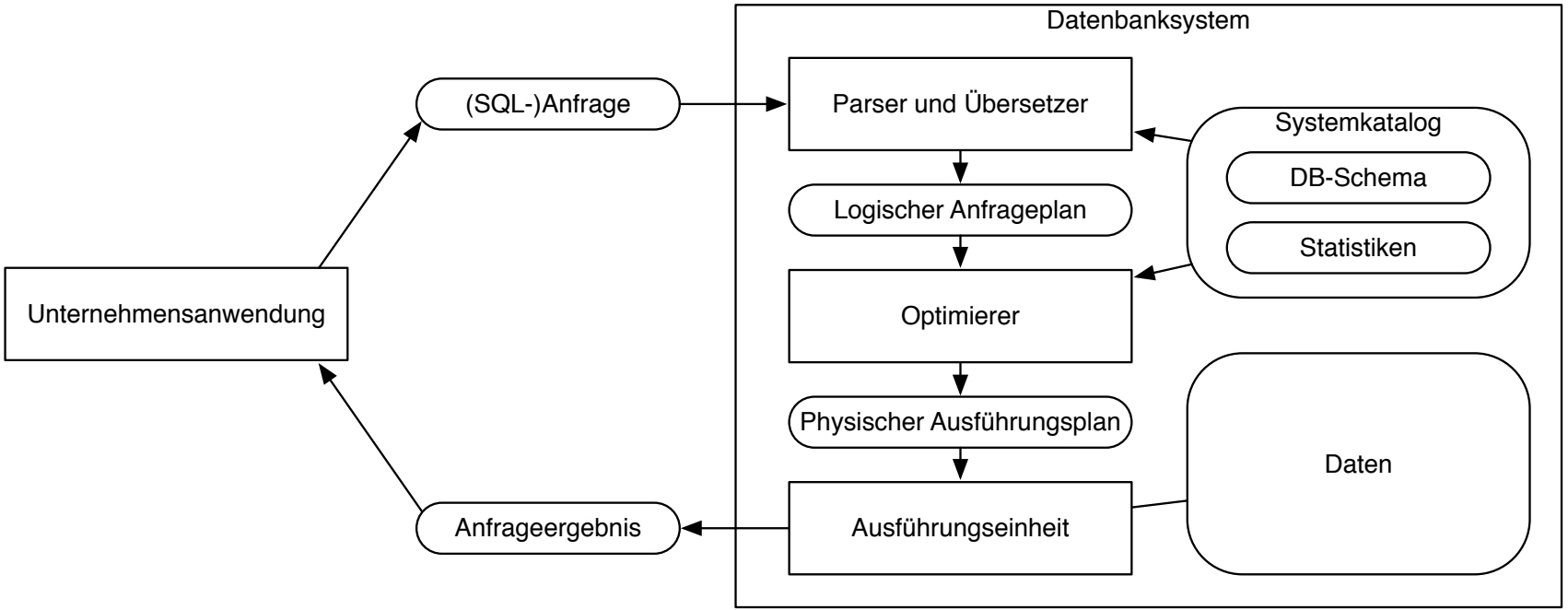
- Relationale DBMSs entstanden in den 1970er Jahren als Forschungs-Prototypen
  - System R
  - INGRES

Genealogy der relationalen Datenbanken
https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/projekte/RDBMSGenealogy/RDBMS_Genealogy_V6.pdf

- Beide Systeme wurden für die Verarbeitung von Unternehmensdaten (OLTP) geschaffen
- **Zeilenorientierte Festplattenbasierte Datenbanksysteme**
- Seit Anfang der 1980er: „**one size fits all**“ Strategie
  Gründe: Kosten, Kompatibilität, Verkauf, Marketing
- Seit 1990er Jahre: Unternehmen setzen zunehmend auf spezialisierte **Data-Warehouses** für **performante und flexible analytische Anfragen** (ohne zusätzliche Indexe und materialisierte Sichten) und **Datenintegration**
- Seit 2000er Jahre: 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)”

Spezialisierte und universelle Hauptspeicherdatenbanksysteme

Relationale Datenbanksysteme
Von der SQL-Anfrage zum Ergebnis



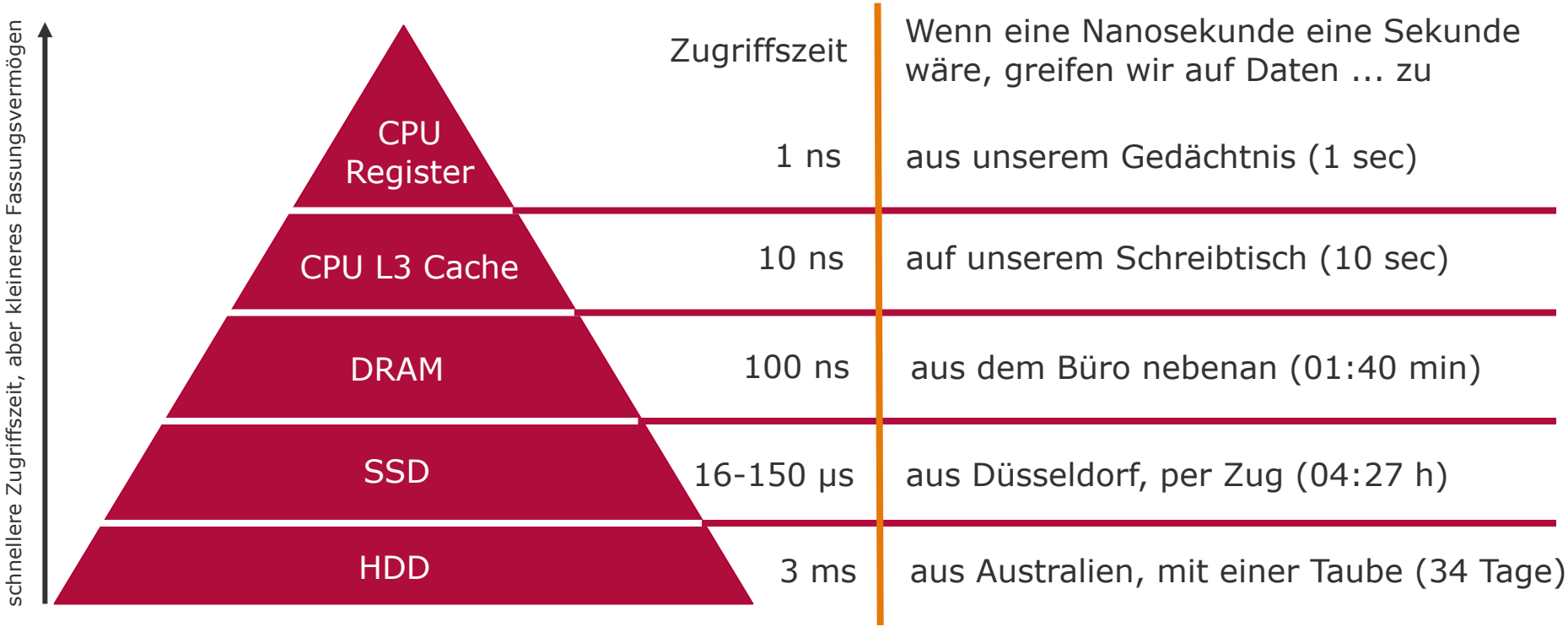
Relationale Datenbanksysteme
Tabellenrepräsentation im Speicher

- Relationales Modell als konzeptionelles/logisches Datenmodell mit abstrakten Operationen

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

- Wie können Daten im Speicher repräsentiert werden?
- Wie können Anfragen **effizient** verarbeitet werden?
  (enge Kopplung, da Speicherzugriff in der Regel der Performanz-Engpass ist)

Speicherhierarchie
„Latency Numbers Every Programmer Should Know“

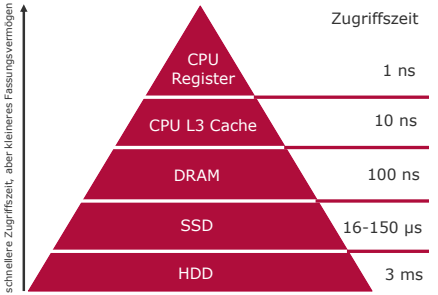


http://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html (Zahlen vom April 2018)

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

Speicherhierarchie
Bedeutung für Programmierung

- Speicher nutzten kleinere und schnellere Ebenen als Caches
- Daten werden in Blöcken in den Speicher geladen
  - CPU-Caches: Cacheline (z.B. 64 Byte bei aktuellen PC-Prozessoren)
  - DRAM: Speicherseite/”Page” (häufig standardmäßig 4 KiB)
    (1 kB = 1000 Bytes 1 KiB = 1024 Bytes)



“All programming is an exercise in caching.”
Terje Mathisen

Festplattenbasierte zeilenorientierte Datenbanksysteme
Motivation

- Festplattenbasierung und Zeilenorientierung sind historisch gewachsen
  - Historische Entwicklung von Datenbanksystemen ist durch den Umstand begrenzter Hardwareressourcen geprägt z.B. Uniprozessoren, kleiner Hauptspeicher

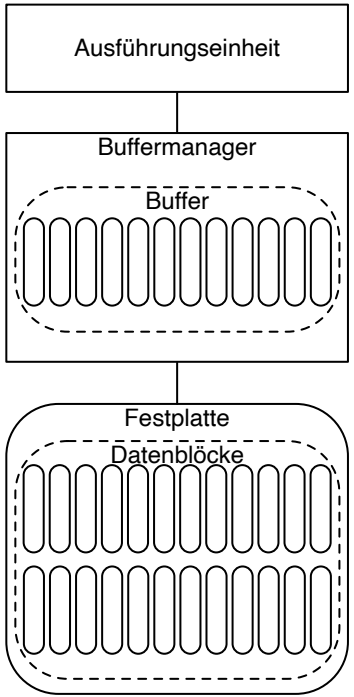
    → Datenbanken mussten auf Festplatte gespeichert werden
  - Zeilenorientierung besser für OLTP als historisch wichtigere Anfrageklasse

Festplattenbasierte zeilenorientierte Datenbanksysteme
Motivation - Festplattenbasierung

- Primärer Speicherplatz der Daten ist die Festplatte
- Daten werden zur Anfragebearbeitung in den Hauptspeicher geladen
- Herausforderungen:
  - Hauptspeicher ist begrenzt,
    aber Performanz ist wichtig (Ziel: Anzahl der Festplattenzugriffe minimieren)
  - Datenänderungen müssen persistiert werden
  - Nebenläufigkeitskontrolle (mehrerer Ausführungseinheiten (Threads))

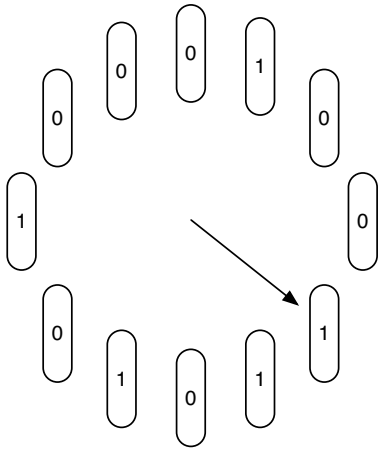
Festplattenbasierte zeilenorientierte Datenbanksysteme
Buffermanager

- Datenbanksysteme verwenden einen **Buffermanager:**
  Alle Speicherzugriffe der Ausführungseinheit gehen über den Buffermanager
- Daten sind (üblicherweise) in **Blöcke** fester Größe (meist 4 – 16 KiB) unterteilt
- Das Datenbanksystem nutzt (in der Regel) einen „Buffer"/Puffer um Blöcke im Hauptspeicher zu cachen
- Buffermanager prüft, ob Block bereits im Speicher ist
  - Falls nicht, liest der Buffermanager den Block in der Buffer
    - Falls kein Platz, muss ein anderer Block ersetzt werden
    - Falls der zu ersetzende Block verändert wurde, muss er zurück auf die Festplatte geschrieben werden



Datenbank hat mehr Informationen als Betriebssystem

- Blockpinning
- Gebündeltes und forciertes Schreiben auf die Festplatte
- Blockprefetching
- Ersetzungsstrategien (Overhead vs. Effektivität):
  - FIFO (First-In-First-Out)
  - LRU (Least Recently Used)
  - Clock („Second Chance“)
  - Wenn letzter Eintrag verarbeitet wurde (Toss-immediate)
  - MRU (Most recently used)



nützlich für Nested-Loop-Join

Tabellenrepräsentation

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

Tabellenrepräsentation
Datentypen

- SQL-Datentypen:
  - Zeichenketten/Strings mit fester und variabler Größe
  - Zahlen mit exaktem Wert
  - Zahlen mit approximiertem Wert
  - Zeitpunkte und –intervalle
  - Große Objekte
  - Benutzerdefinierte Typen

Tabellenrepräsentation
Datentyprepräsentation

- Einzelner Attributwert ist irgendwann eine Bitsequenz im Speicher
  - Bitsequenz ist implementierungsabhängig, meist basierend auf „nativen“ C/C++ Typen
  - Komprimierung möglich

- Datenbank ist eine Aneinanderreihung von Attributwerten (Bitsequenzen); zusätzlich Schema und Metainformationen (welche Komprimierung, welches Datenlayout); + Alignment, Padding

- Datentypen mit fester vs. variabler Größe
  - Felder fester Länge ermöglichen direkten Zugriff im Falle einer kontinuierlichen Speicherung
  - Felder variabler Größe ermöglichen Speicherersparnis

Datentypprepräsentation
Repräsentation von NULL

- Variante 1: Spezieller Wert
  - Festlegung eines speziellen Wertes für einzelne Datentypen (z.B. INT32_MIN)

    → verkleinert den Wertebereich

- **Variante 2: Bitmap pro Spalte/Zeile**

- Variante 3: Markierung direkt am Attributwert

  → benötigt mit Alignment 1 Byte statt 1 Bit (oder verkleinert den Wertebereich)

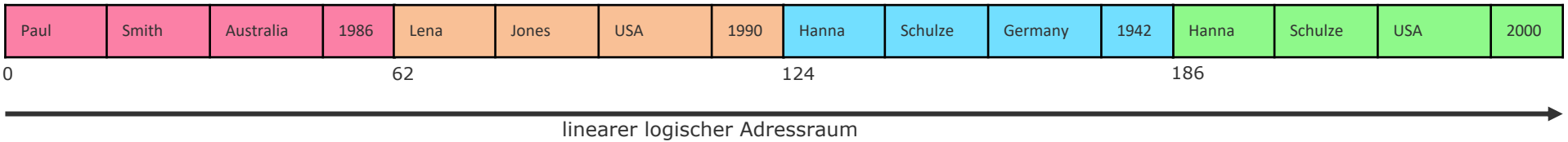
Festplattenbasierte zeilenorientierte Datenbanksysteme
Blocklayout - Datentypen fester Länge

- Sollen Einträge über Blockgrenzen hinausgehen? In der Regel nicht.
- Größe eines Eintrags:

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

Gesamtgröße	62 Bytes

→ 132 Einträge pro 8 KiB Block



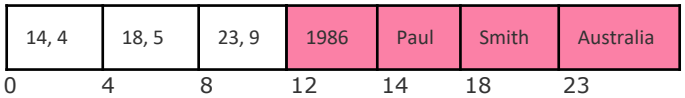
Festplattenbasierte zeilenorientierte Datenbanksysteme
Blocklayout - Datentypen variabler Länge

Hasso Plattner Institut

- Datentypen variabler Länge und NULL-Werte → Einträge variabler Länge
- Zwei Herausforderungen:
  - **Einzelne Attributwerte** eines Eintrags müssen effizient extrahierbar sein (ohne das komplette Tupel zu lesen)
  - **Einzelne Einträge** innerhalb des Blockes müssen effizient extrahierbar sein (ohne den kompletten Block zu lesen)

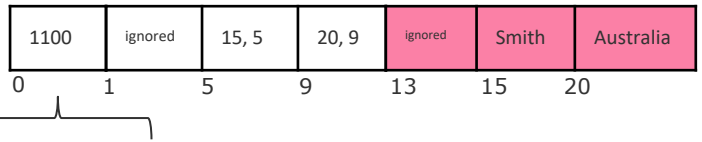
Festplattenbasierte zeilenorientierte Datenbanksysteme
Blocklayout - Datentypen variabler Länge und NULL-Werte

- **Einzelne Attributwerte** eines Eintrags müssen effizient extrahierbar sein (verschiedene Implementierungsmöglichkeiten, hier Lehrbuchbeispiel)
  - Datentypen variabler Länge: Tupel (Offset, Länge) verweisen auf den Attributwert
  - (Datentypen fester Länge ohne zusätzliche Metadaten und am Anfang für direkten Zugriff)



In der Praxis Alignment möglich.

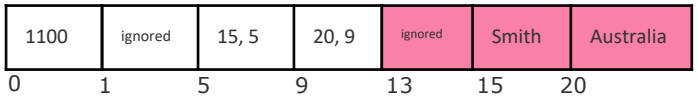
  - Verschiedene Kompromisse zwischen Effizienz und Speicherverbrauch für NULL-Bitmap:
    1. pro Speicherverbrauch: keine Daten bei gesetztem NULL-Bit
    2. pro Effizienz: Daten bei gesetztem NULL-Bit ignorieren



NULL-Bitmap in einem Byte gespeichert. (Reihenfolge entspricht hier der physischen Speicherung.)

Festplattenbasierte zeilenorientierte Datenbanksysteme
Blocklayout - Datentypen variabler Länge und NULL-Werte

Beispiel einer Tupelrepräsentation, die effizienten Zugriff auf alle Attributwerte zulässt



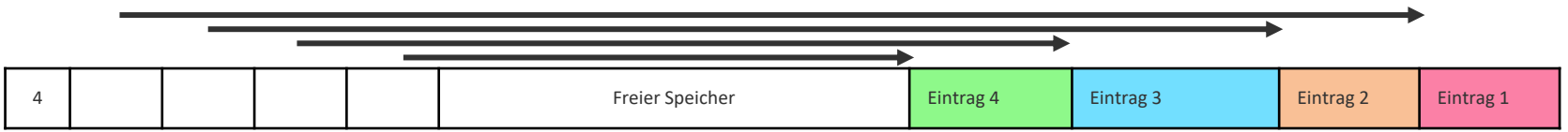
- Die vier NULL-Bits 1, 1, 0 und 0 stehen für die Attribute YearOfBirth, FirstName, LastName und Country
- Datentypen variabler Länge: Tupel (Offset, Länge) verweisen auf den Attributwert
  - gesetztes NULL-Bit für FirstName → Meta-Daten (hier: Bytes an Offset 1 - 4) werden ignoriert
    → erlaubt direkten Zugriff auf Meta-Daten für nachfolgende Attributwerte variabler Länge
  - Attributwert kann bei gesetztem NULL-Bit hingegen weggelassen werden
- Datentypen fester Länge (hier: YearOfBirth)
  - ohne zusätzliche Metadaten (Anzahl der Bytes (hier 2) durch Schema bekannt)
  - am Anfang für direkten Zugriff
  - gesetztes NULL-Bit → Daten (hier: Bytes an Offset 13 und 14) werden ignoriert
    → erlaubt direkten Zugriff für potentiell nachfolgende Attributwerte fester Länge

Festplattenbasierte zeilenorientierte Datenbanksysteme
Blocklayout - Einträge variabler Länge

- **Einzelne Einträge** innerhalb des Blockes müssen effizient extrahierbar sein

  Slotted-Page Struktur: Block-Header speichert:

  - Anzahl der Einträge (oder Beginn des freien Speichers im Block)
  - (Das Ende des freien Speichers im Block)
  - Verweise auf alle Einträge
    - Indexe verweisen auf Verweise im Block-Header und nicht auf die Einträge selbst; diese Indirektion erlaubt es Einträge zu verschieben um Fragmentierung zu verhindern
    - Können auch gelöschte Einträge abbilden



Festplattenbasierte zeilenorientierte Datenbanksysteme
Blocklayout - PostgreSQL

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

- PostgreSQL 11 Documentation Chapter 68. Database Physical Storage
  https://www.postgresql.org/docs/11/storage.html

(Spaltenorientierte) Hauptspeicherdatenbanksysteme
Motivation

- Hauptspeicher ist größer und günstiger geworden
- Durch steigende DRAM-Kapazitäten passen die meisten Datenbanken in den Hauptspeicher
  - **Strukturierte (z.B. relationale) Datensätze** sind kleiner ← unser Fokus
  - Unstrukturierte oder nur teilweise strukturierte Datensätze sind größer
- Festplattenbasierte Datenbanksysteme funktionieren, aber Architektur limitiert Performanz

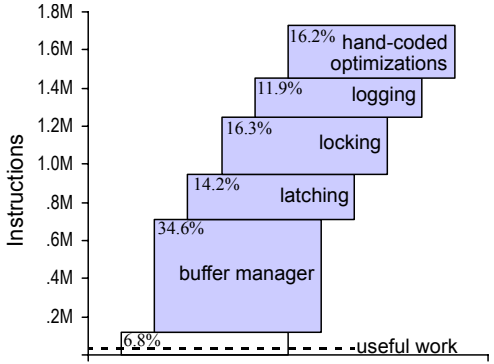


Figure 1. Breakdown of instruction count for various DBMS components for the New Order transaction from TPC-C. The top of the bar-graph is the original Shore performance with a main memory resident database and no thread contention. The bottom dashed line is the useful work, measured by executing the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

(Spaltenorientierte) Hauptspeicherdatenbanksysteme
Hauptspeicher- vs. Festplattenbasierung

- **Datenrepräsentation und -zugriff für dauerhafte Speicherung im Hauptspeicher optimieren:**
  - kein Buffermanager(-overhead)
  - Locking und Latching (limitieren die Skalierbarkeit) vermeiden, z.B. durch optimistische Nebenläufigkeitsverfahren

    Latch vs. Lock (siehe Graefe „A Survey of B-Tree Locking Techniques“)
    https://15721.courses.cs.cmu.edu/spring2016/papers/a16-graefe.pdf
  - Ausnutzung der CPU-Caches:
    auch im Hauptspeicher ist sequenzieller Zugriff ist schneller als zufälliger Zugriff

    Ulrich Drepper „What Every Programmer Should Know About Memory“ (2007)
    https://people.freebsd.org/~lstewart/articles/cpumemory.pdf
  - Hauptspeicherdatenbanken benötigen weiterhin persistenten Speicher für das Log und zur Wiederherstellung, aber Verfahren können optimiert werden, z.B. durch Gruppencommits

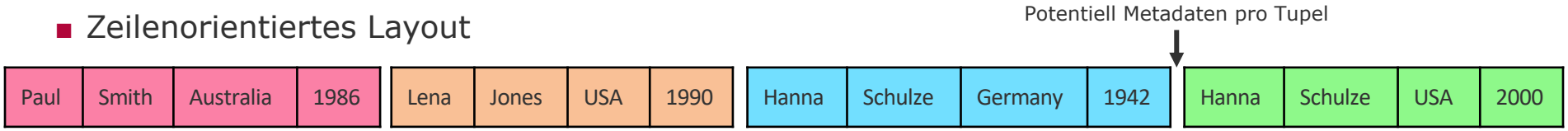
Tabellenrepräsentation
Zeilenorientiertes vs. spaltenorientiertes vs. hybrides Layout

- Wie sind zwei-dimensionale Tabellen im linearen (ein-dimensionalen) Adressraum abgebildet (um möglichst gut Caches zu nutzen)?
- Alternativ: physische Umsetzung des konzeptionellen relationalen Datenmodells

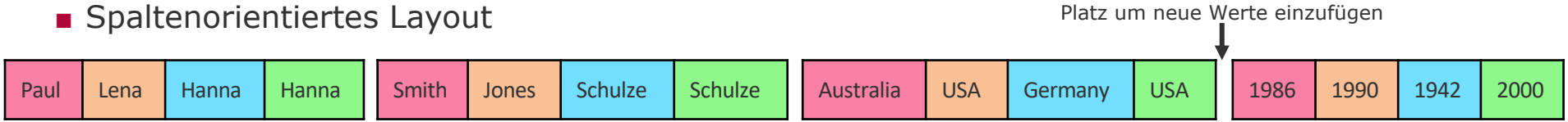
- 3 Varianten:
  - Zeilenorientiert
  - Spaltenorientiert
  - Hybrid (in Attributgruppen)

Tabellenrepräsentation
Zeilenorientiertes vs. spaltenorientiertes vs. hybrides Layout

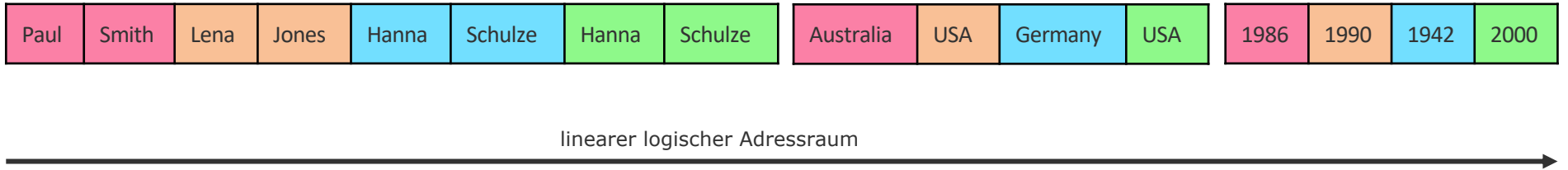
- Zeilenorientiertes Layout



- Spaltenorientiertes Layout

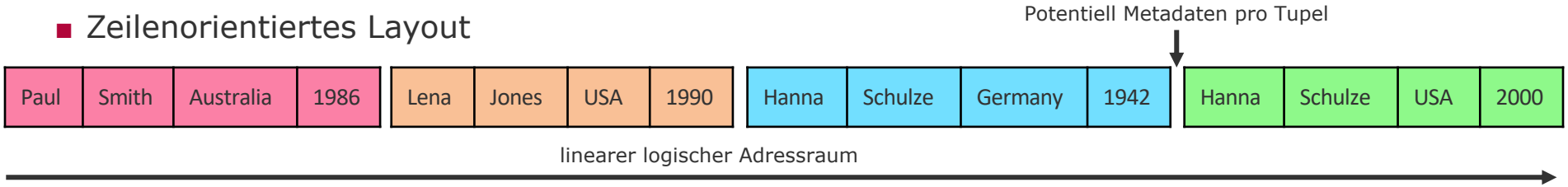


- Hybrides Layout (in Attributgruppen)



Tabellenrepräsentation
Zeilenorientiertes Layout

- Zeilenorientiertes Layout



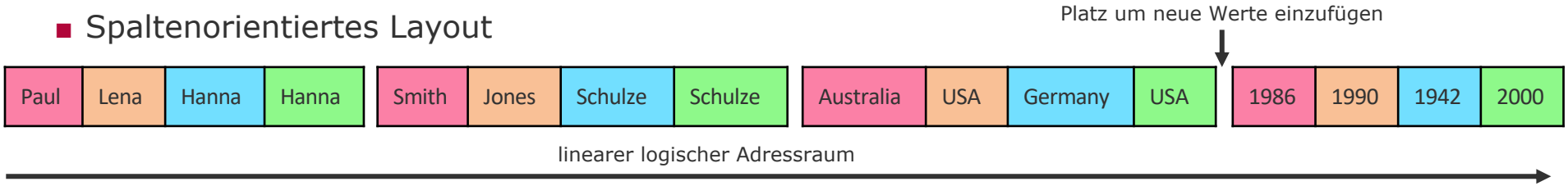
- Speichert alle Attributwerte eines Tupels zusammenhängend

  (bei variabel langen Werten oder sehr großen Objekten zum Teil Verweise/Zeiger)

- Gut geeignet für OLTP-Lasten, in denen Transaktionen auf einzelnen Tupeln arbeiten (Zum effizienten Suchen wird ein Index benötigt!) oder viele neue Tupel einfügen

Tabellenrepräsentation
Spaltenorientiertes Layout

- Spaltenorientiertes Layout



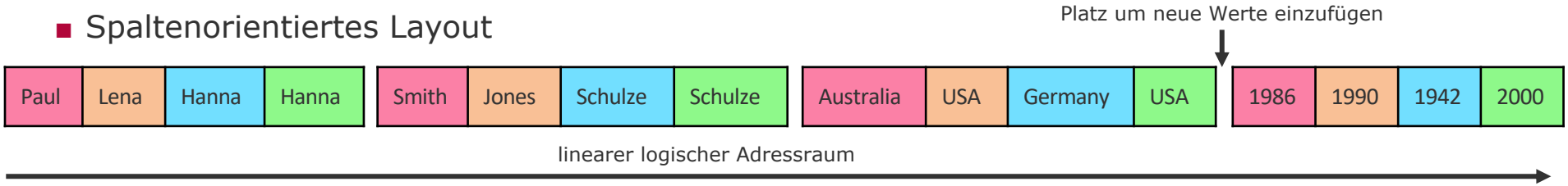
- Speichert alle Attributwerte eines Attributs zusammenhängend

  (bei variabel langen Werten oder sehr großen Objekten zum Teil Verweise/Zeiger)

- Gut geeignet für OLAP-Lasten, in denen Leseanfragen große Teile weniger Attribute durchsuchen

Tabellenrepräsentation
Spaltenorientiertes Layout

- Spaltenorientiertes Layout



- Tupelrekonstruktion
  - **Variante 1: Offset fester Länge**
  - Variante 2: Explizit gespeichert ID

- **Bessere Komprimierung als zeilenorientiertes Layout** (nächste Vorlesung)

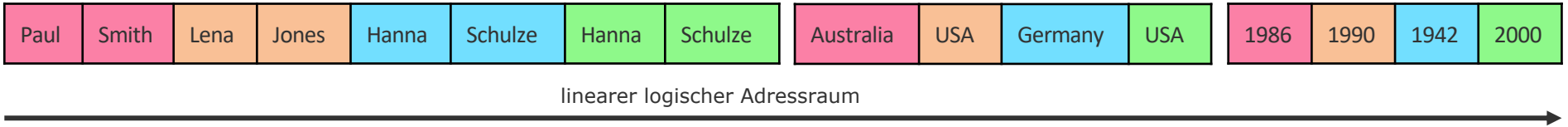
Tabellenrepräsentation
Spaltenorientiertes Layout - Geschichte

- 1970er: Cantor DBMS
- 1980er: Setrag Khoshafian „A Query Processing Strategy for the Decomposed Storage Model“
- 1990er: SybaseIQ (Heute: SAP IQ)
- 2000er: Vertica, MonetDB
- 2010er: **Hyrise** (neue 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

Tabellenrepräsentation
Hybrides Layout

- Hybrides Layout

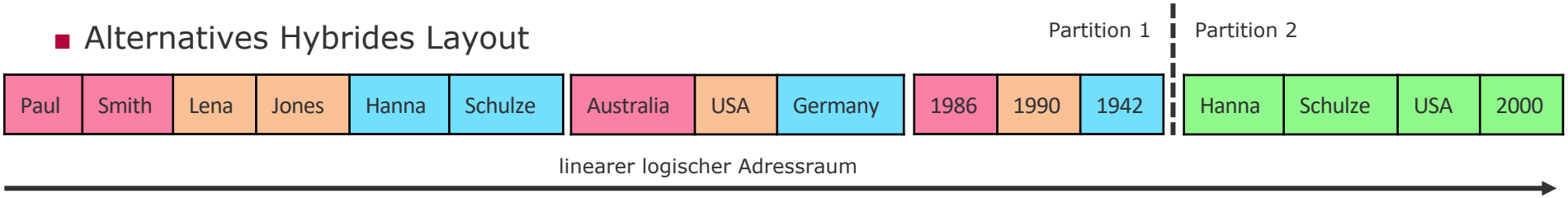


- Speichert alle Attributwerte von Attributgruppen zusammenhängend
  (bei variabel langen Werten oder sehr großen Objekten zum Teil Verweise/Zeiger)

- Attributgruppen einer Tabelle können sich pro Partition unterscheiden
- Idee: optimierte Speicherung der Attribute in Abhängigkeit des Zugriffsmusters
  (Aber: Erhöhte Komplexität bei Query-Optimierung und Ausführungs-Engine)

Tabellenrepräsentation
Hybrides Layout – verschiedene Attributgruppen

- Alternatives Hybrides Layout



- Idee:
  - Sich nicht mehr ändernde Tupel werden für Analysen optimiert gespeichert (Partition 1)
  - Neue Tupel werden zeilenorientiert gespeichert (Partition 2)

- Änderung des Layouts im Laufe der Zeit in Abhängigkeit des Zugriffsmusters (Adaptive/Autonomous/„Self-Tuning"/„Self-Driving" DBSs)

Tabellenrepräsentation
Hybrides Layout – Hyrise

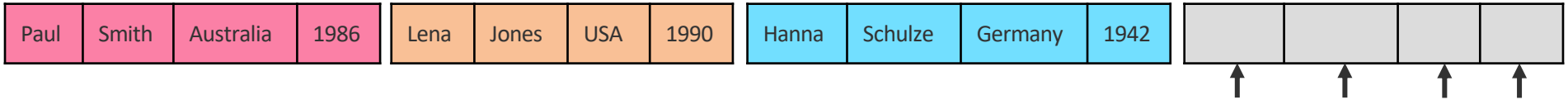
- Martin Grund „HYRISE - A Main Memory Hybrid Storage Engine“ (2010)
  - http://www.vldb.org/pvldb/vol4/p105-grund.pdf
  - Am HPI für Forschungszwecke entwickelte Hauptspeicherdatenbank mit hybridem Speicherformat https://github.com/hyrise/hyrise-v1

Zeilenoperation
Beispiel – Neues Tupel

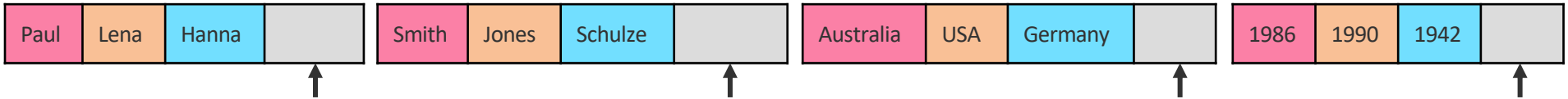
Hanna	Schulze	USA	2000

einfügen

- Zeilenorientiertes Layout



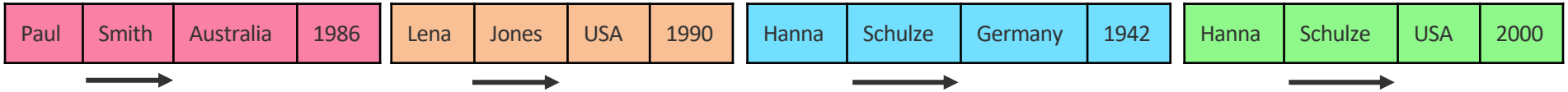
- Spaltenorientiertes Layout



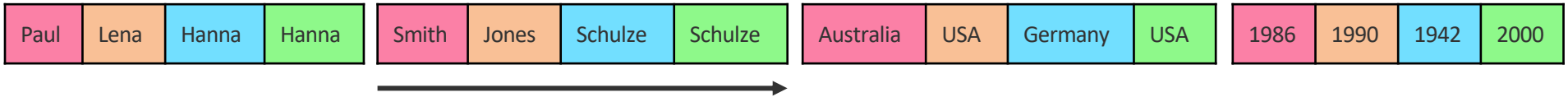
- Hybrides Layout

Spaltenoperation
Beispiel – Tabelle nach Nachname filtern

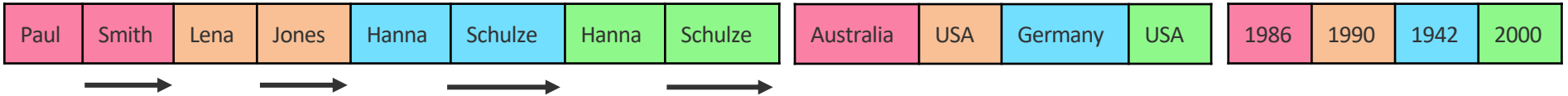
- Zeilenorientiertes Layout



- Spaltenorientiertes Layout

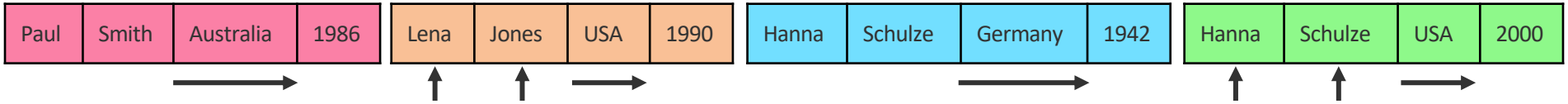


- Hybrides Layout

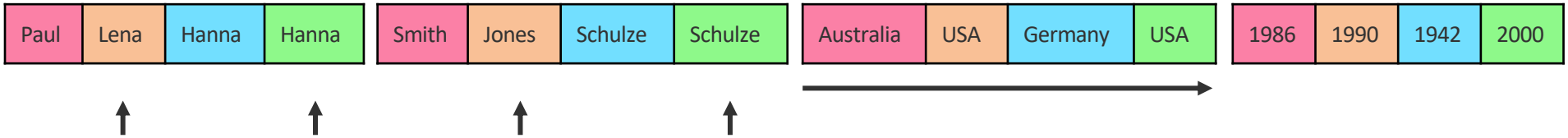


Kombinierte Operationen
Beispiel – Namen aller Personen aus den USA

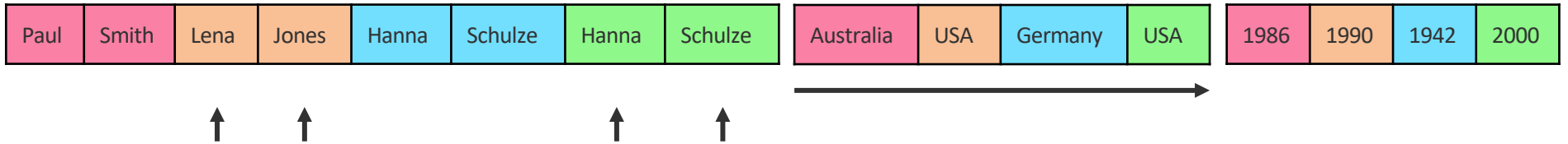
- Zeilenorientiertes Layout



- Spaltenorientiertes Layout



- Hybrides Layout

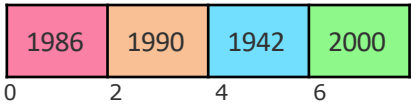


Tabellenrepäsentation
Bestes Layout hängt vom Workload ab

- Insbesondere analytische Anforderungen an Datenbanksysteme sind gewachsen
- Heute: spezialisierte (Hauptspeicher-)Datenbanksysteme
  - Zeilenorientierung besser für OLTP
  - **Spaltenorientierung** (und hybride Layouts) besser für OLAP und **Mixed Workloads** ← unser Fokus (Unternehmensanwendungen)
- Hybride Layouts können dem Workload angepasst werden, aber deren Unterstützung erhöht die Engineering-Komplexität
- Pure Spaltenorientierung ist in der Praxis gut genug für Bearbeitung von OLTP und OLAP in einem (Hauptspeicher-)Datenbanksystem
  (ermöglicht durch Hardwareentwickling: schnelle Multicore-CPUs + größere und günstigerer Hauptspeicher)

Spaltenorientierte Hauptspeicherdatenbanken
Tabellenrepräsentation

- Ziel: (schneller Scan und) **direkter/effizienter** (konstante Zeit) **Zugriff** auf Attributwerte
- Datenblöcke/Spalten speichern Attributwerte fester Länge direkt oder Verweise (mit fester Länge) auf Attributwerte bei variabler Länge
  - Spalte wird als Vektor/Array implementiert



  - Spalten variabler Länge speichern logische Verweise (Offset) oder Pointer

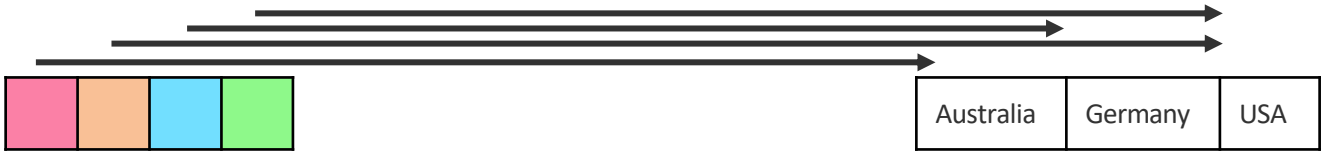
→ Konstante Komplexität beim Punktzugriff (Offset lesen und zur Speicherstelle springen)

→ Aber Indirektion verursacht zusätzlichen Speicherzugriff (Speicher-Effizienz-Kompromiss)

(Maximale Attributwertgröße und „Small/Short String Optimization“ (SSO) als Alternativen)

Spaltenorientierte Hauptspeicherdatenbanken
Tabellenrepräsentation

- Dictionary-Kodierung erzeugt Attributvektoren fester Länge
  (siehe Vorlesung „Datenkompression“)



- Zusätzliche NULL-Bitmap pro Spalte, falls die Spalte NULL-Werte erlaubt

- Löschen von Einträgen in der Regel durch Invalidierung; zusätzlicher Vektor markiert, ob Eintrag gültig ist oder nicht

Spaltenorientierte Hauptspeicherdatenbanken

Primärer Speicherplatz der Daten und Tabellenlayout haben großen Einfluss auf Implementierung des Datenbanksystems:

- Architektur von klassischen festplattenbasierten Datenbanksystemen limitieren die Performanz, selbst wenn die Datenbank in den Hauptspeicher passt
- Zwei-dimensionale Tabellen müssen auf ein-dimensionalen Speicheradressraum abgebildet werden (Ziel: Speicherhierarchie während der Anfragebearbeitung bestmöglich ausnutzen)
  - Das bestes Datenlayout hängt vom Workload (Menge aller Anfragen) ab
  - Spaltenorientuerung ist für analytische und gemischte Workloads besser als Zeilenorientierung
  - Datenkompression ist ein zusätzlicher Faktor bei der Wahl des Datenlayouts

Unternehmensanwendungen
Übungsblatt 5

Unternehmensanwendungen

Sommersemester 2021: Übung 5

Enterprise Platform and Integration Concepts
Fachgebiet | Hasso-Plattner-Institut Universität Potsdam

Aufgabe 1 (20 Punkte)

Beantworten Sie die folgenden Fragen:

a) Was sind Gründe für die historische Trennung von OLTP und OLAP in verschiedene Systeme? (6 Punkte)

- ☐ analytische Anfragen verlangsamen das transaktionale System
- ☐ verschiedene Datenbankschemata notwendig
- ☐ inkompatible SQL-Dialekte
- ☐ kein Bedarf an Vereinigung
- ☐ performante und flexible analytische Anfragen möglich
- ☐ unterschiedliche Workload-Eigenschaften

b) Warum und wie ist es heutzutage möglich OLTP- und OLAP-Anfragen in einem System effizient zu bearbeiten? (6 Punkte)

- ☐ Schnellere Hardware (Moore's Law)
- ☐ Entwicklung spaltenbasierter Hauptspeicher-DBSs
- ☐ Entwicklung von Quantencomputer
- ☐ größerer und günstigerer Hauptspeicher
- ☐ Entwicklung der Blockchain
- ☐ Multicore-CPUs