Hauptspeicherindexstrukturen

Stefan Sprenger

Semesterprojekt "Verteilte Echtzeitrecherche in Genomdaten"

10. November 2015

Agenda

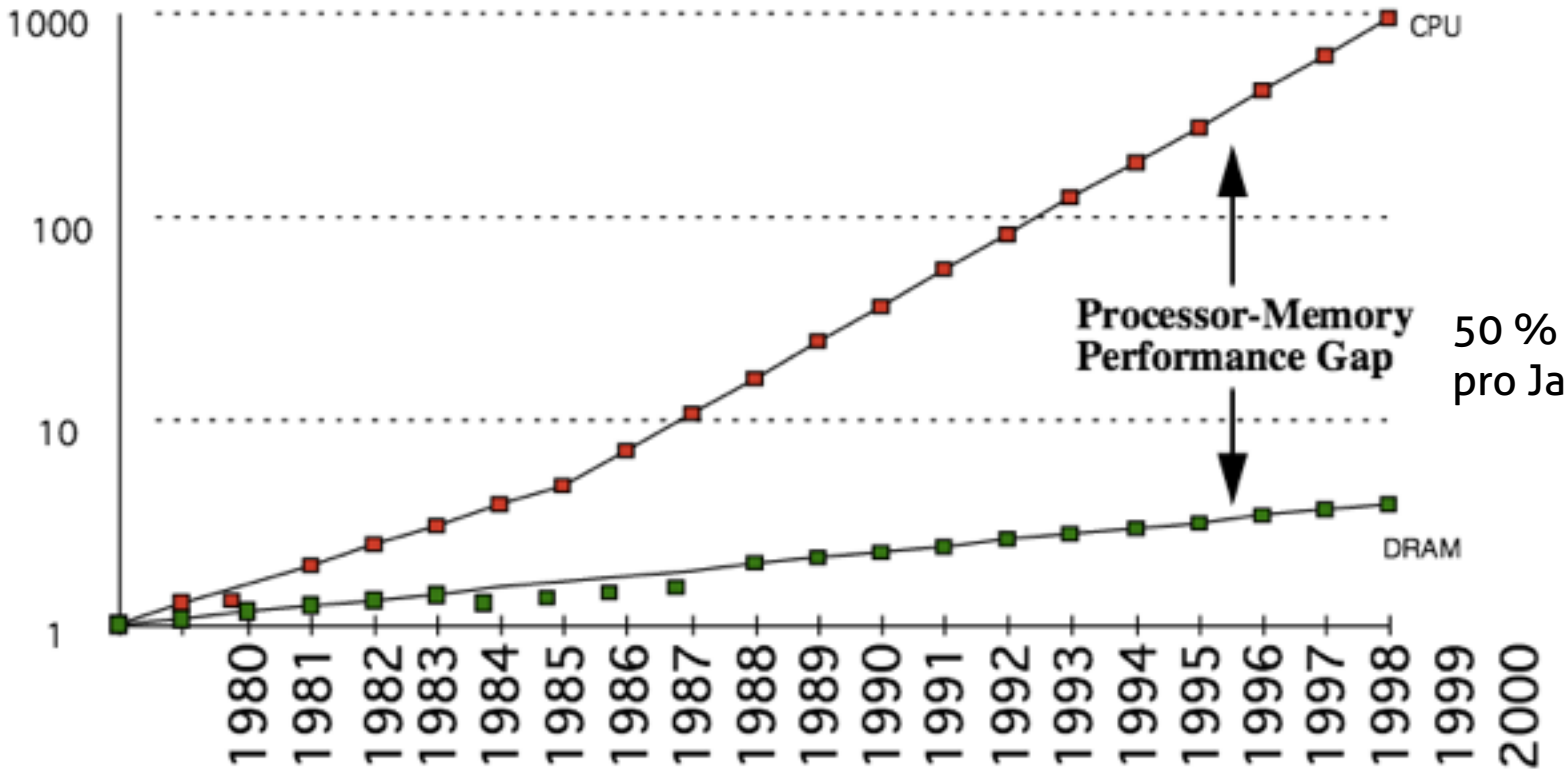
- Einführung in Hauptspeichertechnologien
- Indexstrukturen in relationalen Datenbanken
- Indexstrukturen für den Hauptspeicher
- Tipps für das Projekt

Einführung

Motivation

- Verfügbarer Hauptspeicher steigt an
- Heute existieren Systeme mit TBs an Hauptspeicher
- Hauptspeicher kann zum Speichern von Indexstrukturen und ganzen Datenbanken genutzt werden
- Hauptspeicher bietet “schnellen” Zugriff auf große Datenmenge

Processor-Memory Performance Gap



50 % Zuwachs pro Jahr

D. Patterson et al.: "A case for intelligent RAM", 1997

Motivation

runtime = compute time + wait time

minimieren

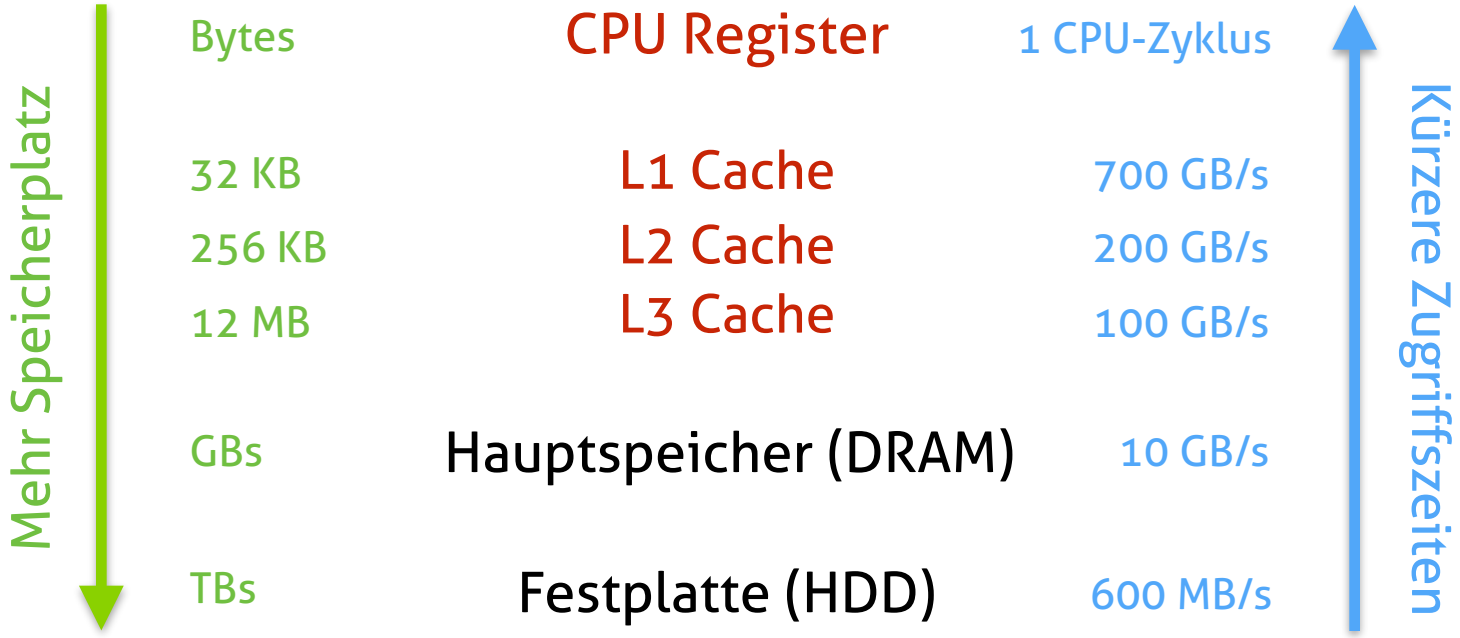
Motivation

- Hauptspeicher wird zum Flaschenhals
- Es müssen diverse Besonderheiten beachtet werden, um der CPU einen möglichst effizienten Zugriff auf den Hauptspeicher zu ermöglichen
- Die Implementierung von Datenstrukturen beeinflusst die Performanz einer Anwendung wesentlich

Speicherhierarchie

- Speicherhierarchie besteht aus mehreren Ebenen
- höhere Ebenen sind kleiner, teurer und schneller
- tiefere Ebenen sind größer, günstiger und langsamer
- höhere Ebenen speichern Kopien von Daten aus tieferen Ebenen

Speicherhierarchie



Zugriff auf Daten

- Daten werden mit dem LOAD-Befehl in die CPU-Register übertragen
- Falls die angefragten Daten im Cache existieren, werden sie direkt in das Register geladen
- Falls die angefragten Daten nicht im Cache existieren, werden sie über den Cache in das Register geladen

Cache Lines

- zentrale Größeneinheit für Daten im Hauptspeicher und den Caches
- Daten werden immer als Cache Line (CL) übertragen, auch wenn weniger angefragt werden
- eignet sich sehr gut für Streaming
- typische CL-Größe: 64 oder 128 Bytes

Memory Locality

- Hauptspeicher = Random Access Memory?
- Da CL verwendet werden, kann ein sequentieller Zugriff auf benachbarte Daten vorteilhaft sein
- Daten im Hauptspeicher abhängig von den Zugriffsmustern ablegen

Wichtige Parameter in Linux bestimmen

L1 Cache: /sys/devices/system/cpu/cpu0/cache/index{0,1}/..
L2 Cache: /sys/devices/system/cpu/cpu0/cache/index2/..
L3 Cache: /sys/devices/system/cpu/cpu0/cache/index3/..

Cache-Größe:

```
$ more /sys/devices/system/cpu/cpu0/cache/index0/size
32K
```

Cache Line-Größe:

```
$ more /sys/devices/system/cpu/cpu0/cache/index0/coherency_line_size
64
```

Cache-Level:

```
$ more /sys/devices/system/cpu/cpu0/cache/index0/level
1
```

Flüchtigkeit

- Hauptspeicher ist flüchtig
- Bei Hardwaredefekten oder Systemneustarts muss der Hauptspeicherindex neu aufgebaut werden
- Neue Technologien: Non-Volatile (Main) Memory

Indexstrukturen in relationalen Datenbanken

Indexstrukturen in relationalen Datenbanken

- Relationale Datenbankmanagementsysteme (RDBMS) stammen aus einer Zeit, in der Hauptspeicher teuer und klein war
- Erste RDBMS in den 70er Jahren
- Indexstrukturen werden auf der Festplatte gespeichert
- Ausnutzen der blockweisen Speicherung

Indexstrukturen in relationalen Datenbanken

- Indexstrukturen werden üblicherweise spaltenweise genutzt
- Vermeiden von teuren Scan-Operationen
- Ziel: Suchen und Sortieren beschleunigen

Beispiel:

```
-- create table
CREATE TABLE employees (
  name varchar(100) NOT NULL,
  salary integer NOT NULL,
  gender varchar(100) NOT NULL,
  country varchar(100) NOT NULL
);

-- create index (uses B tree by default)
CREATE INDEX ON employees(salary);

-- select all employees from UK
SELECT * FROM employees WHERE country = 'UK';

-- select all employees with a salary over 40000
SELECT * FROM employees WHERE salary > 40000;
```

Indexstrukturen in relationalen Datenbanken

1.) B+ Baum

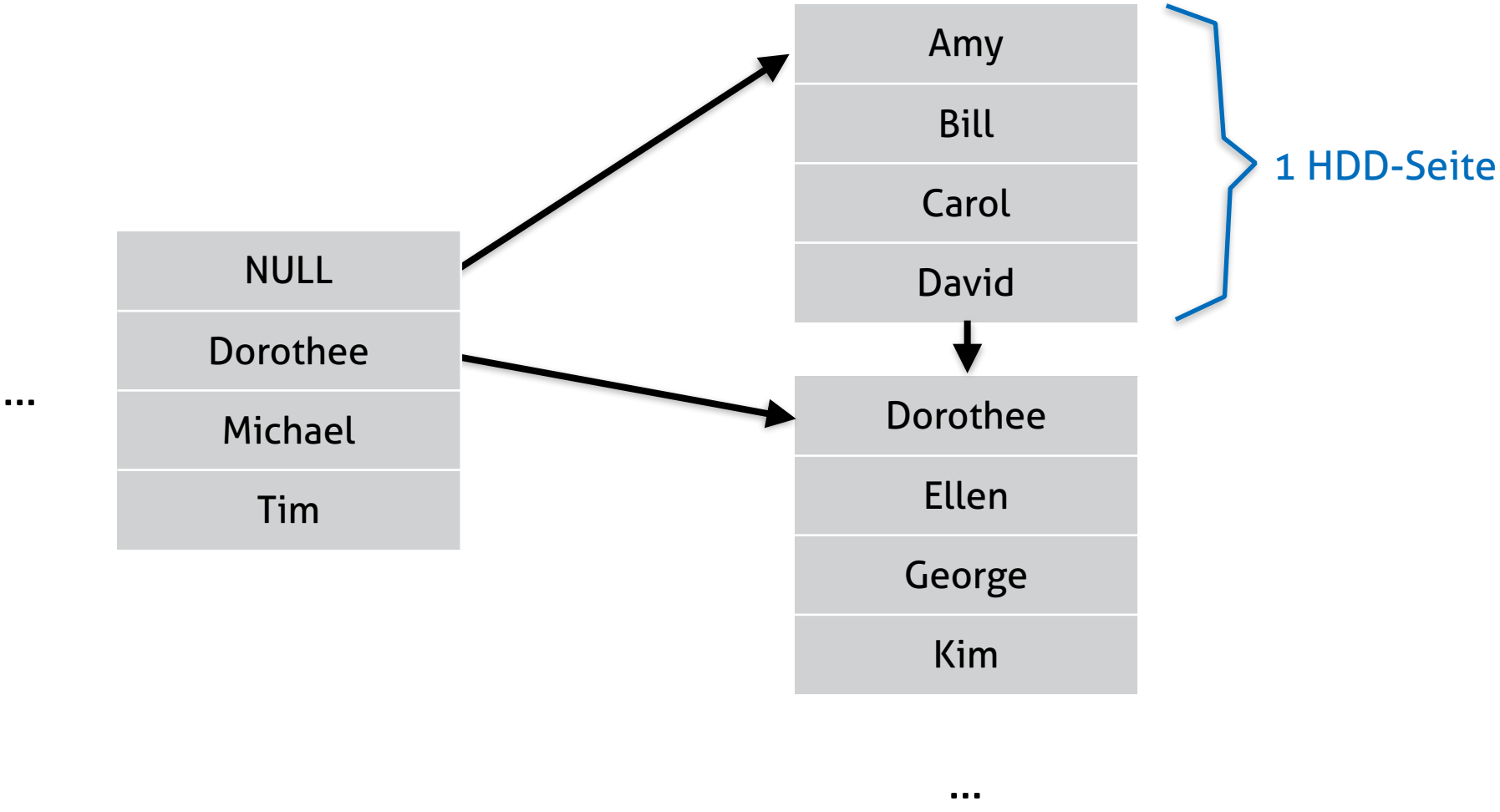
2.) Bitmap-Index

B+-Baum

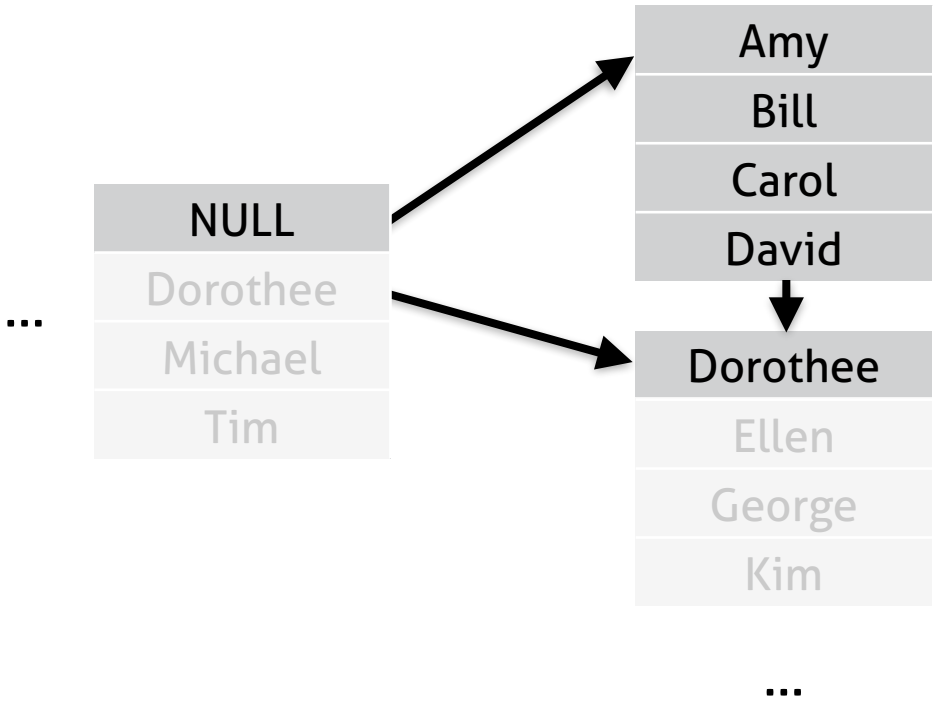
- Erweiterung des B-Baums: Verknüpfung aller Blätter
- sehr verbreitet bei RDBMS
- effizient für Bereichsabfragen (log(n)+k)
- In RDBMS speichern Blätter des Baums HDD-Seiten (je Blatt eine Seite)

D. Comer: "Ubiquitous B-tree", 1979

B+-Baum: Beispiel



B+-Baum: Beispiel




```
SELECT *
  FROM employees
  WHERE name < 'Ellen';
```

Indexstrukturen in relationalen Datenbanken

1.) B+ Baum

2.) Bitmap-Index

Bitmap-Index

- effizientes Indexieren von (mehreren) Spalten mit geringer Kardinalität
- speichert für jede Kombination aus Zeile, indexierter Spalte und vorkommendem Spaltenwert, ob Zeile x in der Spalte y den Wert z besitzt
- Durchsuchen mit booleschen Operatoren möglich

C.-Y. Chan und Y. Ioannidis: “Bitmap index design and evaluation”, 1998

Bitmap-Index: Beispiel

Name	gender: male	gender: female	country: DE	country: UK
Amy	0	1	0	1
Barbara	0	1	1	0
Peter	1	0	0	1

Bitmap-Index: Beispiel

Name	gender: male	gender: female	country: DE	country: UK
Amy	0	1	0	1
Barbara	0	1	1	0
Peter	1	0	0	1

Suche nach allen Frauen aus Großbritannien: Suchmaske = 0101

Indexstrukturen für den Hauptspeicher

Main-Memory-Datenbanken

redis

memsql

HANA

... und viele mehr.

Indexstrukturen für den Hauptspeicher

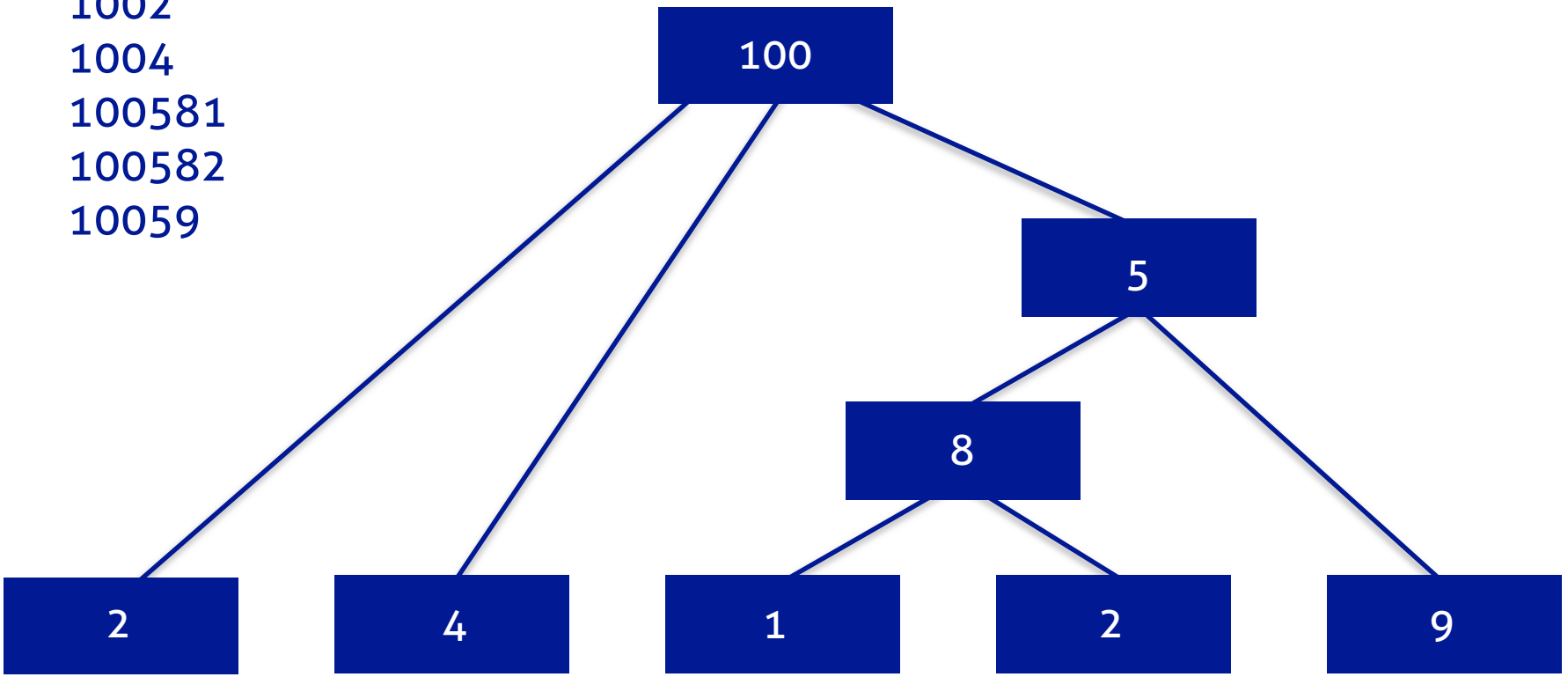
1.) Adaptive Radix Tree

2.) Skip List

3.) CSB+-Baum

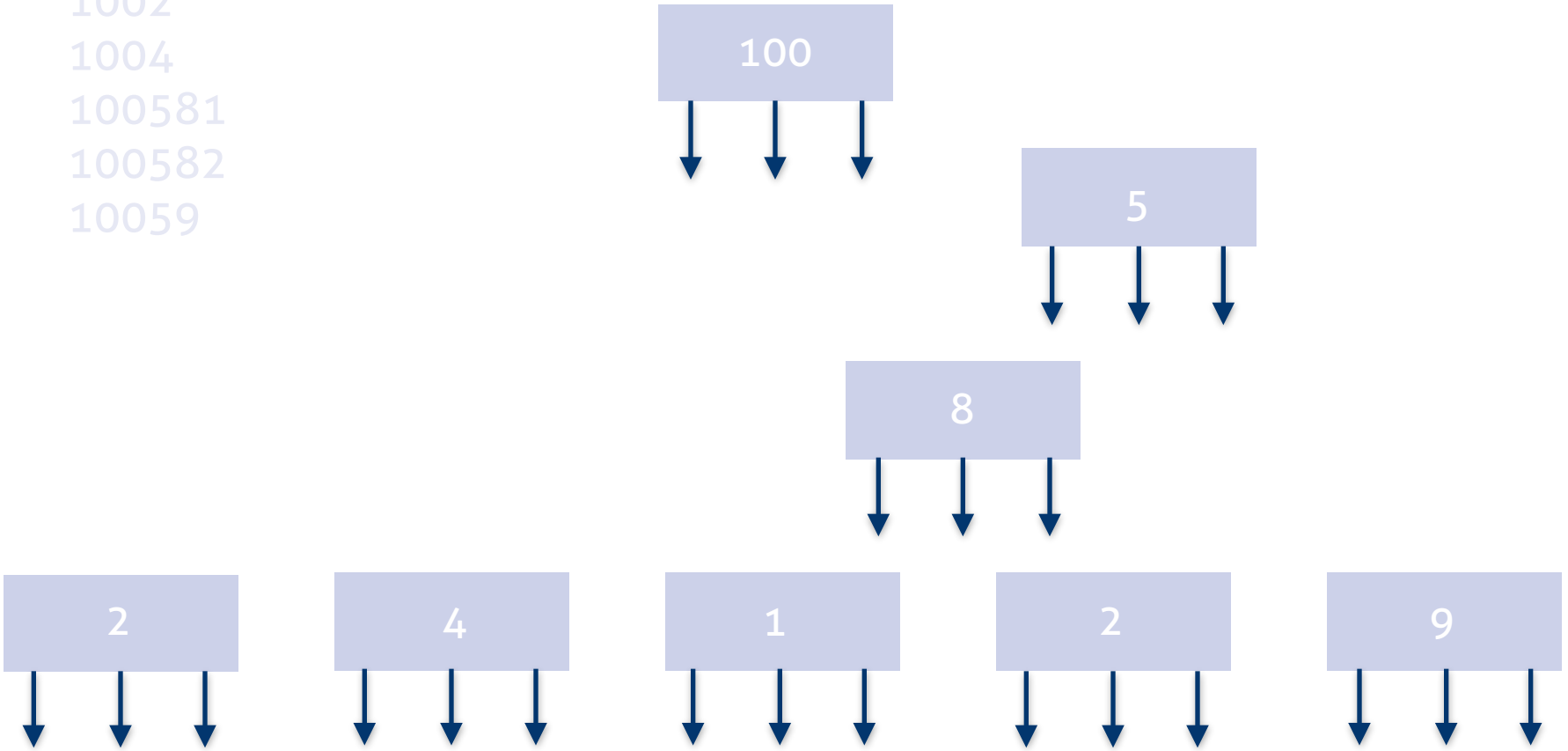
Radix Tree

1002
1004
100581
100582
10059



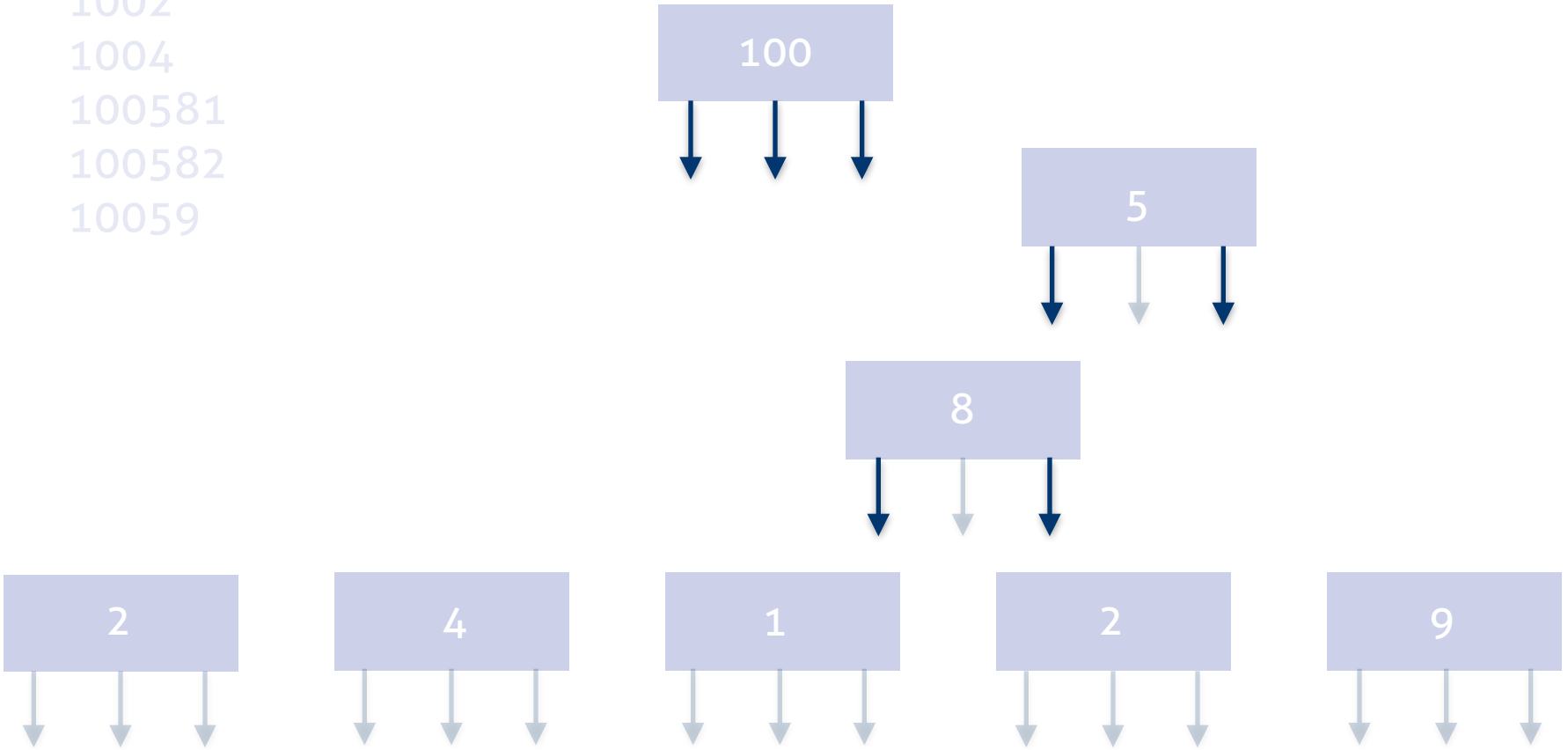
Radix Tree

1002
1004
100581
100582
10059



Radix Tree

1002
1004
100581
100582
10059



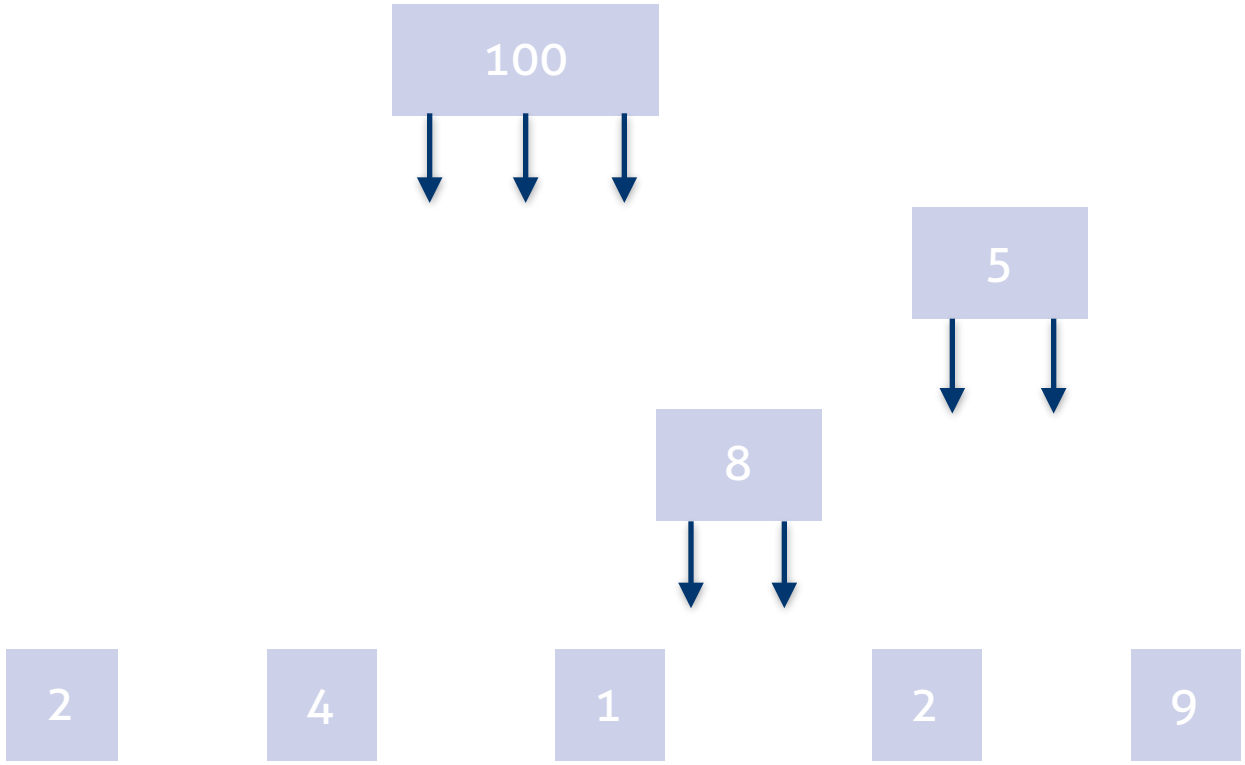
Adaptive Radix Tree

- Erweiterung des Radix Tree
- Adaptive Knotengrößen (4 bis 256 Pointer-Arrays)
- Geringerer Speicherverbrauch
- Bessere Performanz

V. Leis et al.: “The adaptive radix tree: ARTful indexing for main-memory databases”, 2013

Adaptive Radix Tree

1002
1004
100581
100582
10059



Indexstrukturen für den Hauptspeicher

1.) Adaptive Radix Tree

2.) Skip List

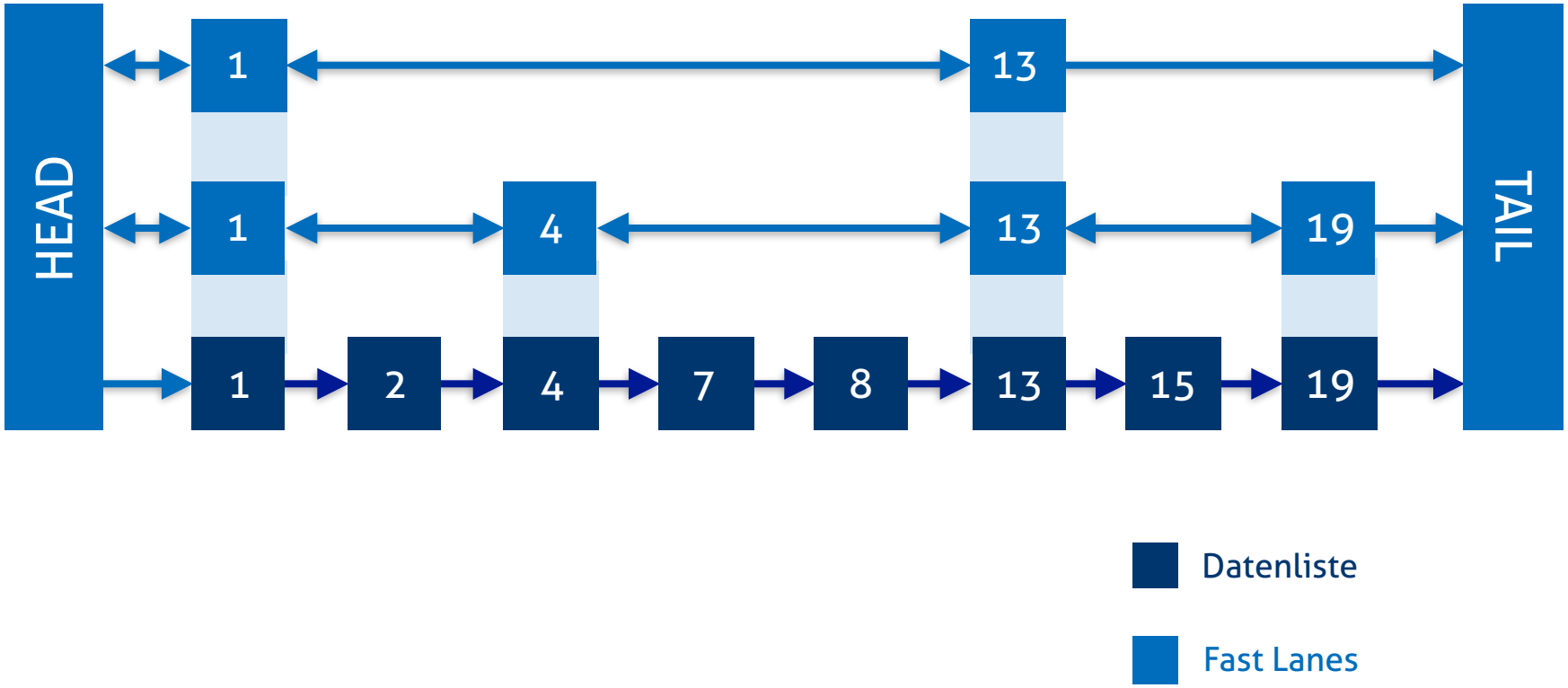
3.) CSB+-Baum

Skip List

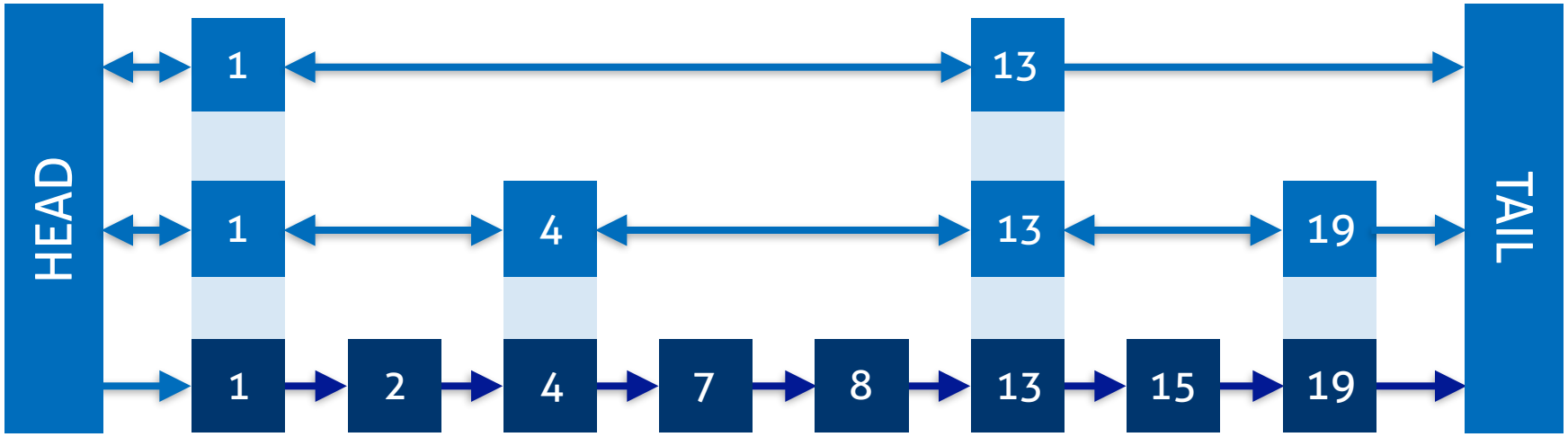
- schnelle Suche über Elemente einer verlinkten Liste
- mehrere Listen (Fast Lanes) zusätzlich zur eigentlichen Liste
  - hierarchisch angeordnet
  - Subsequenzen unterschiedlicher Dichte
  - zum Überspringen von Elementen
- gleiche zeitliche Komplexitäten wie B-Baum
- probabilistische Datenstruktur

W. Pugh: “Skip lists: a probabilistic alternative to balanced trees”, 1990

Skip List: Beispiel



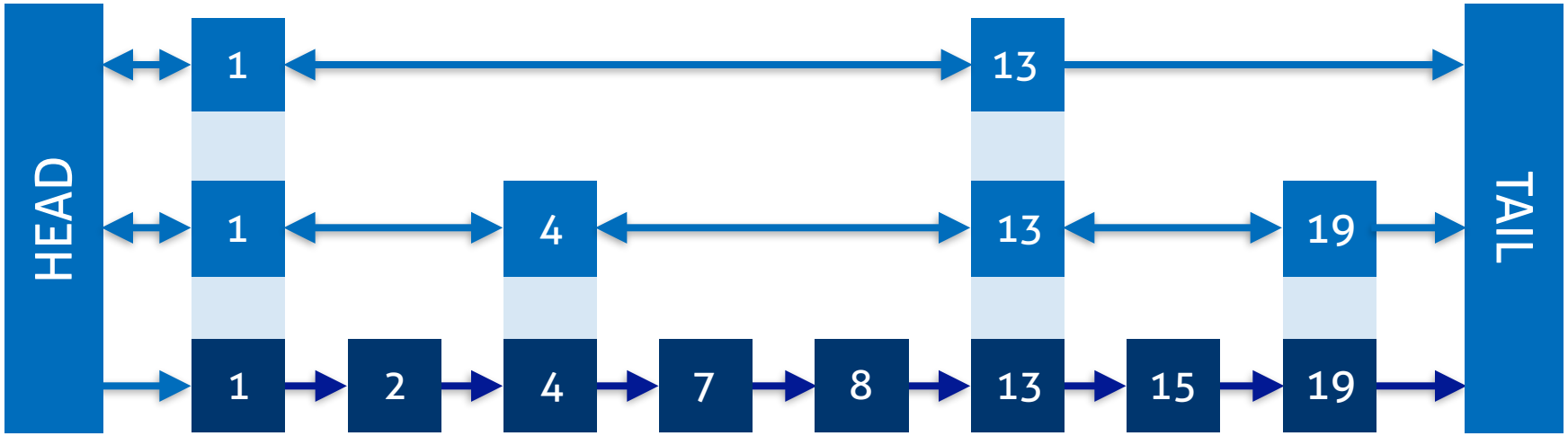
Skip List: Suchen



Schritt 1:
Mit Hilfe der Fast Lanes den Suchbereich eingrenzen

Schritt 2:
Gesuchtes Element in der Datenliste suchen

Skip List: Einfügen



Schritt 1:
Suche mit Skip List die Position in der Datenliste, in die das neue Element eingefügt wird

Schritt 2:
Berechne für jede Fast Lane, ob ein Fast Lane-Eintrag für das neue Element eingefügt wird

Schritt 3:
Neusetzen der Pointer in der Datenliste, ggf. Neusetzen der Pointer in Fast Lanes

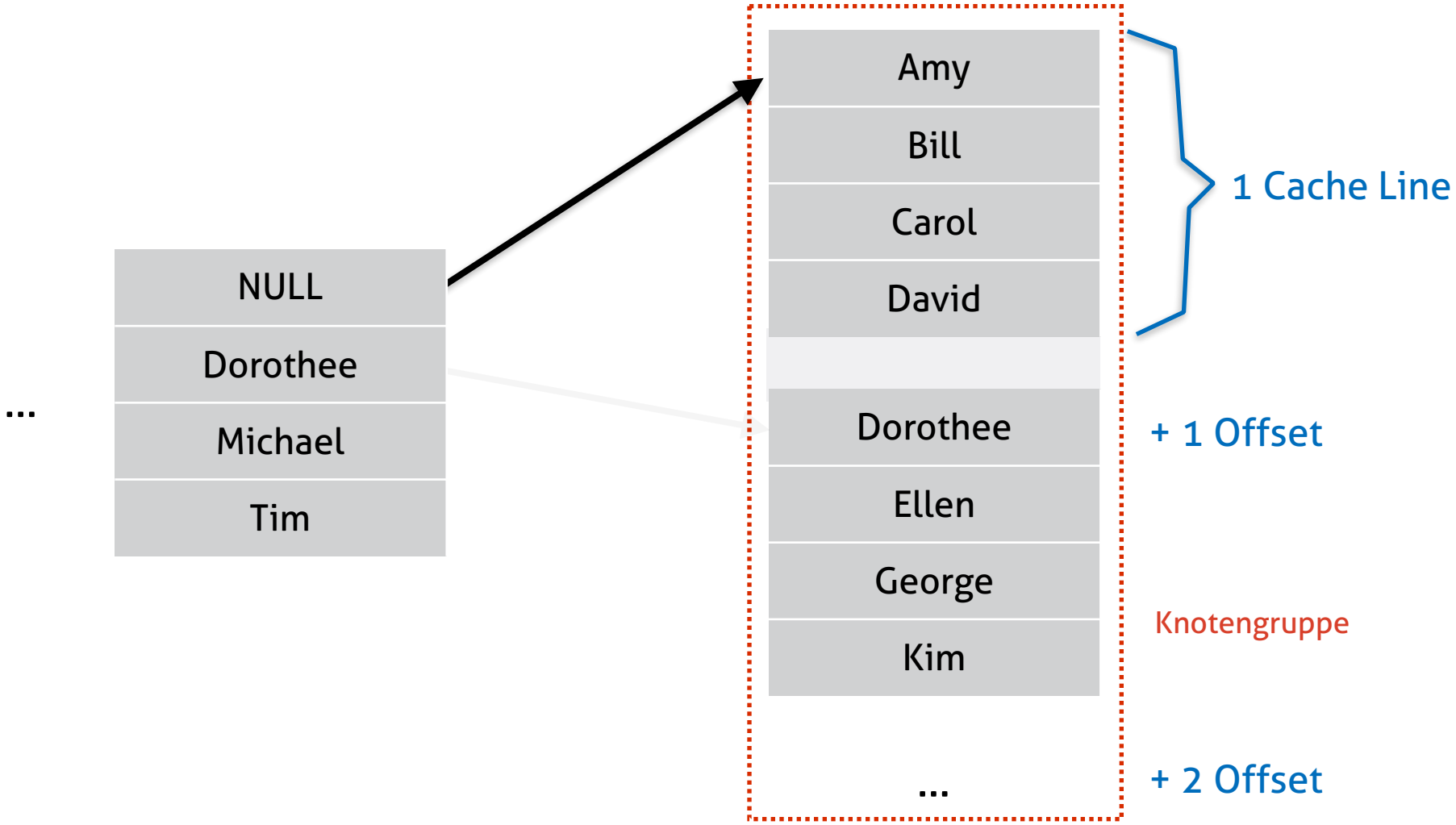
Indexstrukturen für den Hauptspeicher

1.) Adaptive Radix Tree
2.) Skip List
3.) CSB+-Baum

CSB+-Baum

- Cache Sensitive B+-Baum
- sequentielles Speichern der Kindknoten in Array
- jeder Knoten verweist mit Pointer auf ersten Kindknoten aus einer Gruppe von Kindknoten (Knotengruppe)
- Kindknoten werden über den Pointer und ein Offset bestimmt
- jeder Knoten ist exakt so groß wie eine Cache Line

CSB+-Baum: Beispiel



Tipps für das Projekt

Tipps für das Projekt

- möglichst platzsparend arbeiten und nur wichtige Daten in den Index laden
- bei der Implementierung beachten:
  - wenige Pointer verwenden
  - auf primitive Datentypen zurückgreifen
  - Arrays statt Listen
  - Cache Lines
- auch im Hauptspeicher möglichst viel sequentiell lesen

Tipps für das Projekt

- möglichst früh die Performanz analysieren
- Code-Reviews
- an Flüchtigkeit des Hauptspeichers denken
- Fragen stellen :-)

Weiterführende Literatur

- "What every programmer should know about memory" von U. Drepper
  http://www.akkadia.org/drepper/cpumemory.pdf
- "Memory hierarchy: Caches " (Vorlesung) von G. Wellein et al.
  http://www3.informatik.uni-erlangen.de/Lehre/CAMA/SS2014/caches.pdf
- "Efficient In-Memory Indexing with Generalized Prefix Trees" von M. Boehm et al.
  https://wwwdb.inf.tu-dresden.de/misc/team/boehm/pubs/btw2011.pdf
- "A study of index structures for main memory database management systems" von T. Lehman und M. Carey
  http://www.vldb.org/conf/1986/P294.PDF