# Indexstrukturen für Zeichendaten und Texte

Felix Hain

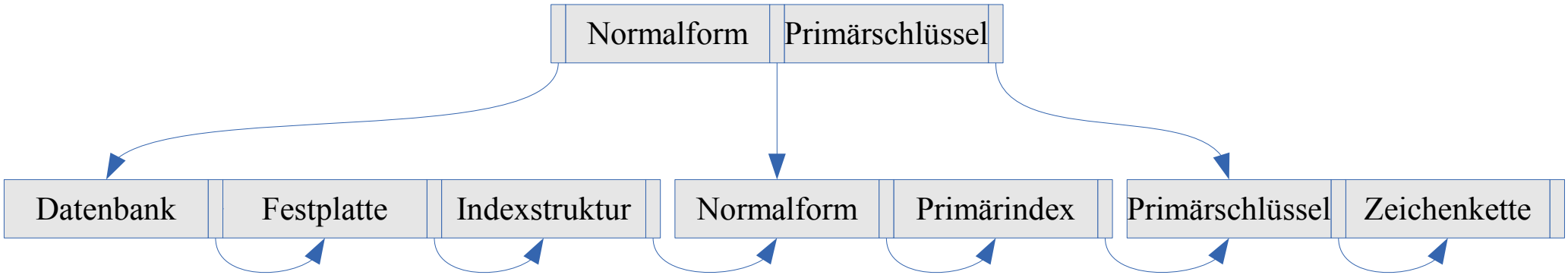
HTWK Leipzig

29.06.15

# Gliederung

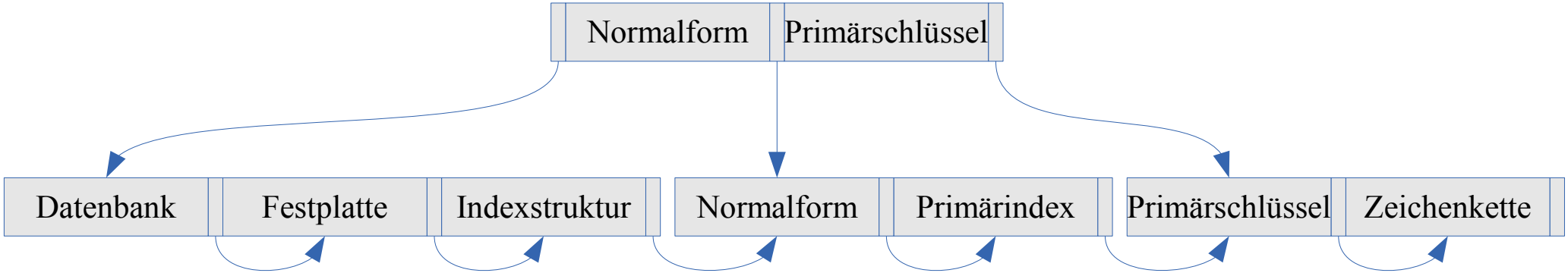
# B+-Baum

- balancierter Suchbaum
- Schlüssel mit dazugehörigen Datensätzen in den Blattknoten
- Referenzschlüssel mit Wegweiserfunktion in den inneren Knoten



# $B^+$-Baum

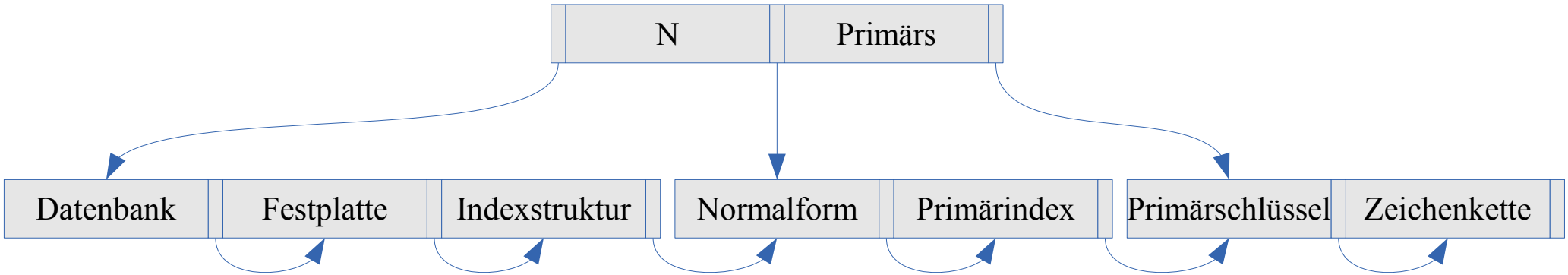
- lange Schlüssel wie z. B. Zeichenketten haben hohen Platzbedarf
- für B-Bäume ungünstig, da Anzahl von Einträgen pro Knoten beschränkt
- aber: im $B^+$-Baum haben innere Knoten nur eine Wegweiserfunktion



# Präfix-B$^+$-Baum

Idee:

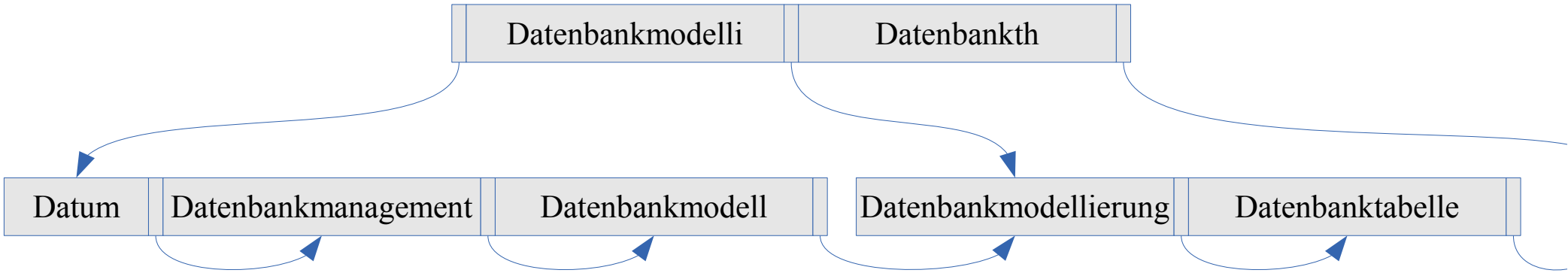
- Speicherung von Schlüsselpräfixen statt ganzer Schlüssel
- nur so lang wie benötigt



# Präfix-B$^+$-Baum

Idee:

- Speicherung von Schlüsselpräfixen statt ganzer Schlüssel
- nur so lang wie benötigt




- Nachteil: bei Schlüsseln mit langen gemeinsamen Präfixen degeneriert der Baum annähernd zum gewöhnlichen B$^+$-Baum

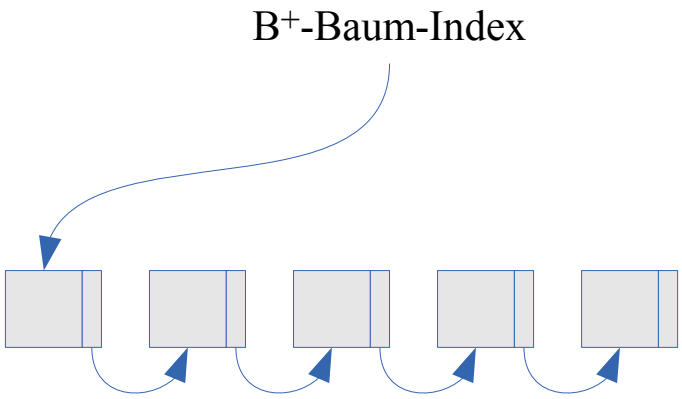
# $B^+$-Baum für BLOBs

- BLOB = Binary Large Object, d. h. große unstrukturierte Datenmenge
- in Allgemeinen größer als eine Seite
- unmittelbare Speicherung in den Knoten eines Baumes daher nicht sinnvoll

# $B^+$-Baum für BLOBs

Möglichkeit:

- jedes Objekt wird auf einer verketteten Liste von Seiten gespeichert
- B-Baum- oder $B^+$-Baum-Index enthält Zeiger auf die jeweilige Anfangsseite des Objekts

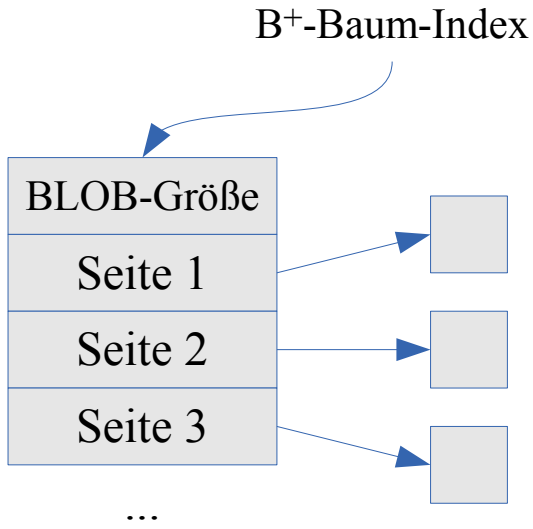



- Nachteil: kein wahlfreier Zugriff im BLOB

# $B^+$-Baum für BLOBs

alternative Möglichkeit:

- Index enthält für jedes Objekt ein BLOB-Directory
- Directory enthält Gesamtgröße des BLOBS und Zeiger auf seine einzelnen Seiten

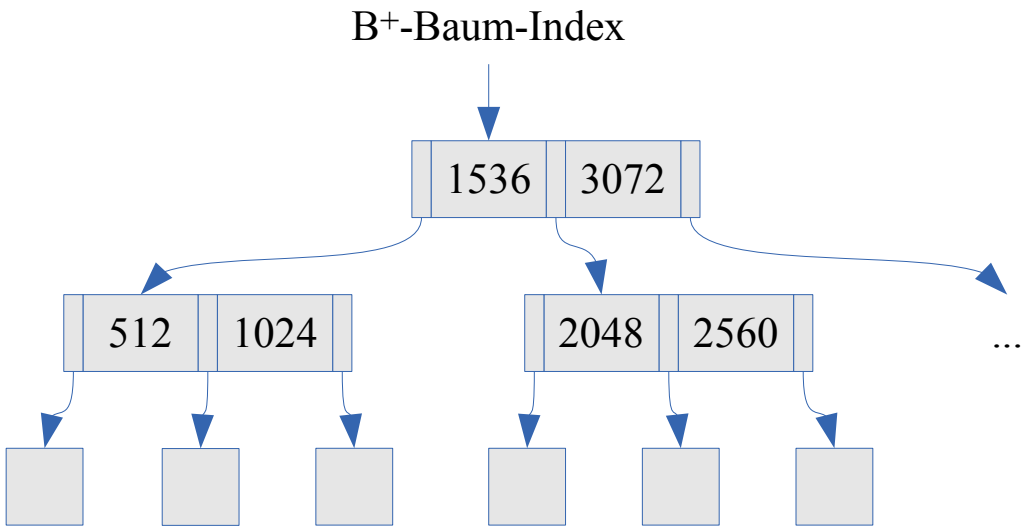



- Vorteil: wahlfreier Zugriff möglich
- Nachteil: Directory kann selbst wiederum sehr groß sein

# B+-Baum für BLOBs

Positions-B+-Baum:

- Datenseiten jedes BLOBs werden in einen jeweils eigenen B+-Baum eingetragen
- Schlüssel = Offset der entsprechenden Seite im Objekt
- der ursprüngliche Index enthält dann nur noch Zeiger auf diese Positionsbäume



# $B^+$-Baum

- Vorteil von B-Bäumen:
  bei fester Suchschlüsselgröße kann man eine optimale Ordnung für den Baum bestimmen, sodass Knoten jeweils eine Seite belegen
- funktioniert nicht bei Schlüsseln variabler Länge, wie Zeichenketten
- Abhilfe:
  - Präfix-$B^+$-Bäume mit festgelegter maximaler Präfixlänge
  - B-Bäume mit Knoten, die eine feste Anzahl von mehreren Seiten belegen
  - digitale Bäume

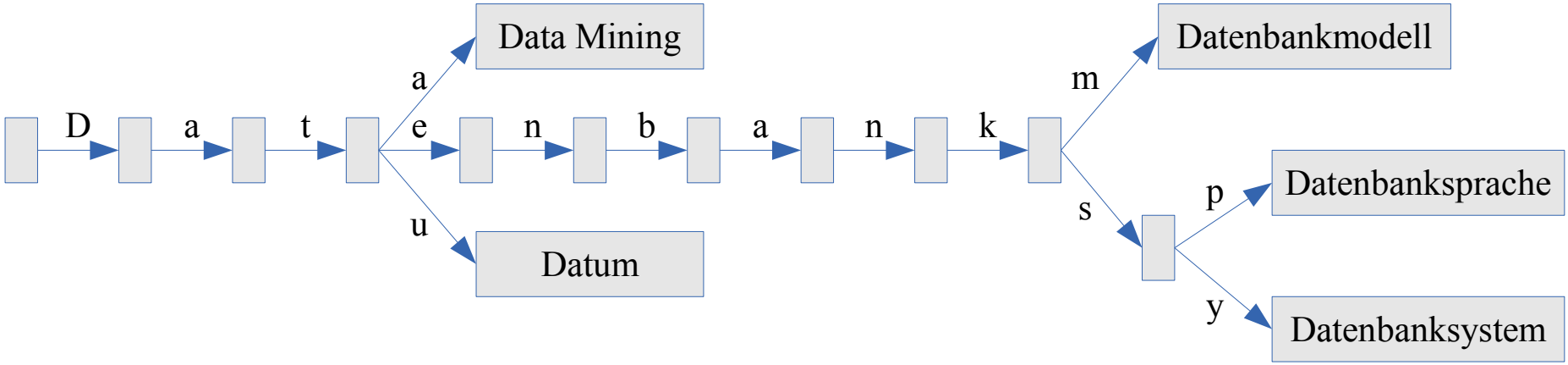
# Digitale Bäume

- “Digitale Bäume” ist ein Sammelbegriff
  - Suchbäume
  - Zeichenketten über einem festen Alphabet als Schlüssel
  - nicht banlanciert

- auch genannt “Tries” (von engl. “information retrieval”)

# (grundlegender) Trie

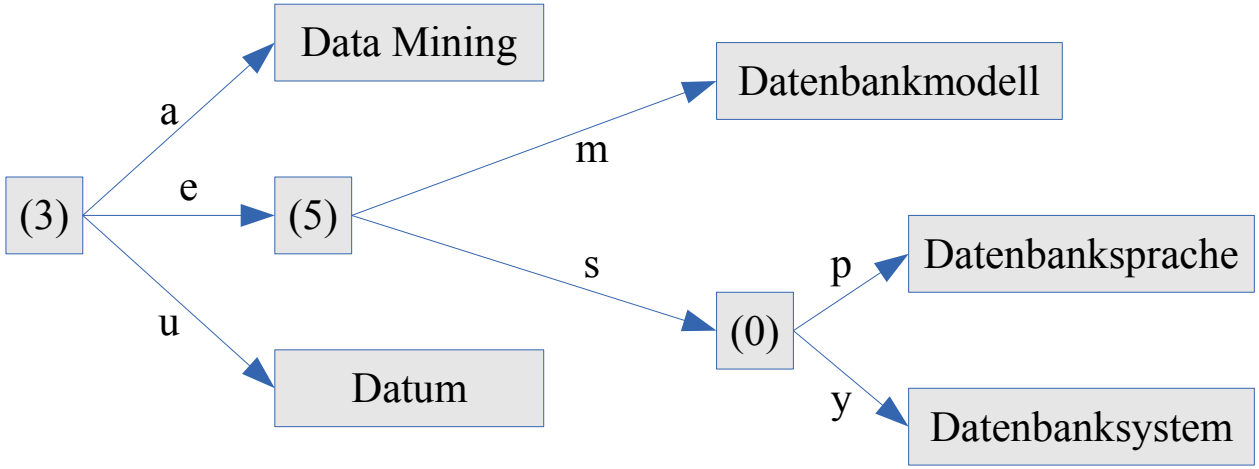
- jeder innere Knoten
  - enthält genau ein Zeichen
  - verweist auf min. einen weiteren inneren Knoten oder auf ein Blatt
- jeder Blattknoten
  - enthält genau einen Datensatz, die indizierte Zeichenkette




- Nachteil: bei Zeichenketten mit einem langen identischen Präfix entartet der entprechende Teil des Baumes zu einer Liste

# Patricia Trie

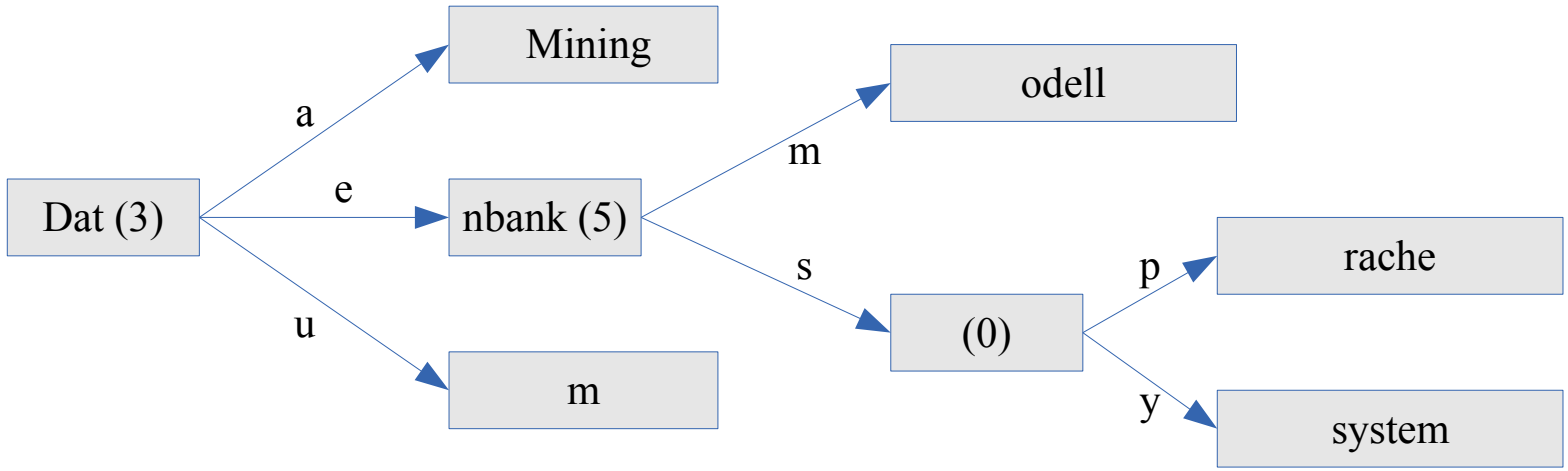
- “Practical Algorithm to Retrieve Information Coded in Alphanumeric”
- Verbesserung des Trie-Prinzips durch “Überspringen” irrelevanter Teilwörtern
- Listenstrukturen werden zu einem Knoten zusammengefasst, der nur die Anzahl der zu überspringenden Knoten enthält




- Nachteil: nicht enthaltene Datensätze werden bei einer Suche u. U. erst spät erkannt (hier z. B.: “Triebwerksperre”)

# Prefix Trie

- Verbesserung des Patricia-Baumes durch Speichern des “übersprungenen“ Präfixes in den inneren Knoten zusätzlich zur Zeichenanzahl
- Blätter müssen dann nur noch die Restzeichenkette enthalten



# Digitale Bäume: Zusammenfassung

Vorteile:

- keine Speicherung kompletter Schlüssel in den inneren Knoten
- beim Standard- und Patricia-Trie: innere Knoten haben feste Größe
- beim Präfix-Trie: Speicherung gemeinsamer Präfixe in den inneren Knoten, dafür kleinere Blattknoten

Nachteile:

- hohe Bäume
- mangelnde Adaptivität an ungünstige Datenverteilungen

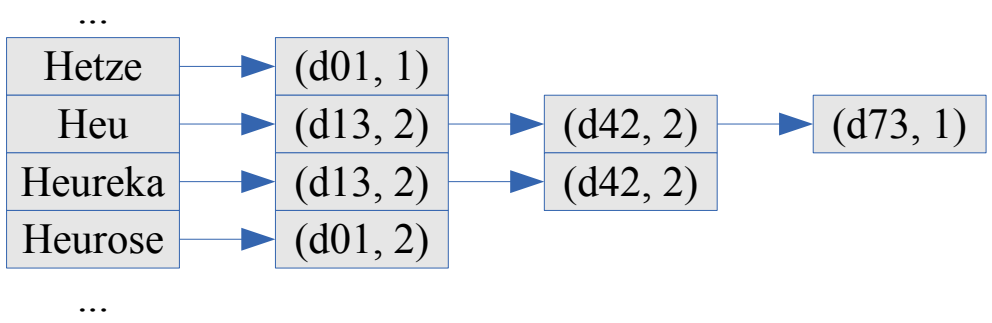
# Digitale Bäume: Zusammenfassung

Digitale Bäume sind geeignet:

- bei annähernd gleichverteilten Präfixen
  - insbesondere bei künstlich erzeugten, z. B. für Hashverfahren
- bei ungleich verteilten Präfixen, wenn sich die Komprimierung lohnt
- für Textindizes
- für andere Anwendungen im Textbereich, z. B. Rechtschreibkorrektur

# Invertierte Liste

- sehr alte Indexstruktur für Zeichenketten in Dokumenten
- Anordnung der indizierten Worte in einer lex. sortierten Liste
- jeder Knoten enthält eine Liste mit Identifikatoren der Dokumente, in denen das Wort vorkommt
  - zusätzlich optional die Position des ersten Vorkommens und die Gesamtzahl der Vorkommen im jeweiligen Dokument




- Suchanfragen über Vorkommen und Relevanz können allein mittels der Liste beantwortet werden

# Quellen


- Gunter Saake, Andreas Heuer, Kai-Uwe Sattler; Datenbanken - Implementierungstechniken, mitp, 2011 (3. Auflage)
- Thomas Kudraß: Dateiorganisation und Indexe, in: Taschenbuch Datenbanken, Hanser, 2007
- https://de.wikipedia.org/wiki/Trie, Stand: 29.06.15