Hashing

- Ziel: Daten mit durchschn. 1-2 Seitenzugriffen finden
- Hashfunktion bildet Schlüssel auf Behälter (Bucket) ab: $\boldsymbol{h : S \rightarrow B}$
  - S=beliebig große Schlüsselmenge
  - B=Nummerierung von *n* Behälter → [0..n)
  - |S| >> |B|, d.h. *h* ist im allg. nicht injektiv
- Es soll eine möglichst gleichmäßige Verteilung erreicht werden, ansonsten entsteht Aufwand durch Überlauf eines Behälters.
- Häufig eingesetzt: modulo – Funktion (mit Primzahl)

Statisches Hashing

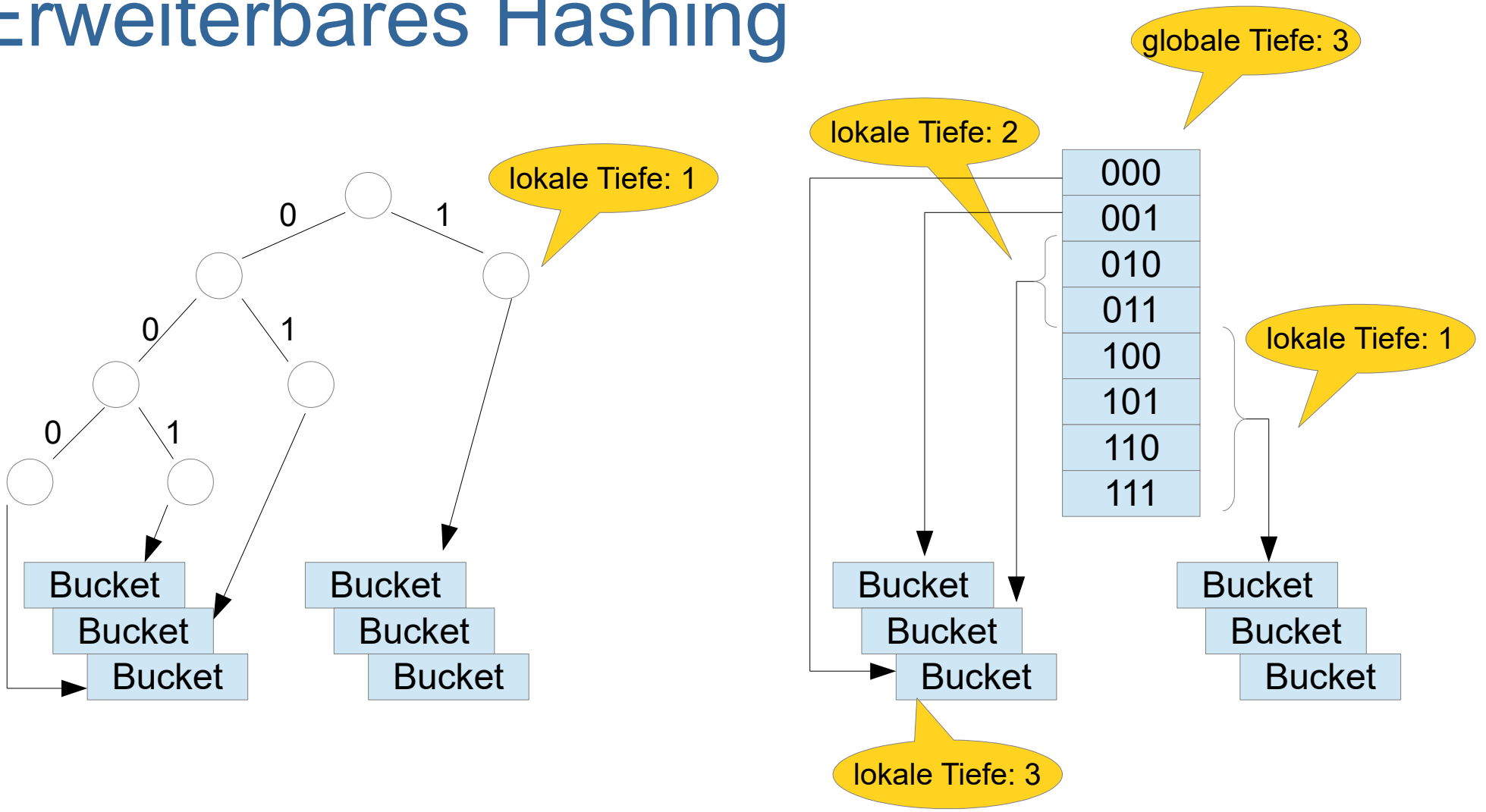
- Jeder Behälter besteht anfangs aus exakt 1 Seite (=primäre Seite)
- Bei einem Überlauf wird eine weitere Seite zum Bucket hinzugefügt (verlinkt)
- Sind mehrere Überlaufseiten vorhanden, wird die Suche teuer, da mehrere Seiten gelesen werden müssen. Werden die Behälter zu groß gewählt, wird anfangs Speicherplatz verschwendet.

- Lösung 1: nachträgliche Vergrößerung der Hashtabelle
  - Rehashing der Einträge → Aufteilung auf mehr Behältersehr teuer bei großen Datenmengen
- Lösung 2: erweiterbares Hashing

Erweiterbares Hashing

- Hashfunktion wird dahingehend verändert, dass sie auf einen wesentlich größeren Bereich abbildet als der tatsächlichen Anzahl an Behälter.
- Das Ergebnis der Hashfunktion wird binär dargestellt, wobei nur ein Präfix dieser Darstellung verwendet wird: ***h(x) = dp***, wobei d=Verzeichnisposition, p=unbenutzt
- Zugriff auf die Buckets kann als (im allgemeinen nicht ausbalancierter) binärer Entscheidungsbaum dargestellt werden.

Erweiterbares Hashing



Erweiterbares Hashing - Hashfunktion

- h: Schlüsselmenge → {0,1}*
- Der Bitstring muss lang genug sein, um alle Objekte auf ihre Buckets abbilden zu können
- Begonnen wird mit einem kurzen Präfix (wenige Bits)
- Wächst die Hashtabelle, wird sukzessive ein längeres Präfix benötigt → Verzeichnis jeweils verdoppelt
- Globale Tiefe: Aktuell verwendete Anzahl an Bits in den Hashwerten
- Lokale Tiefe: Länge des Pfades der auf dieses Bucket zeigt