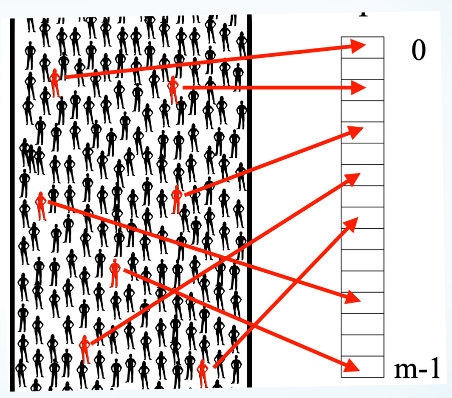


7 Hashing

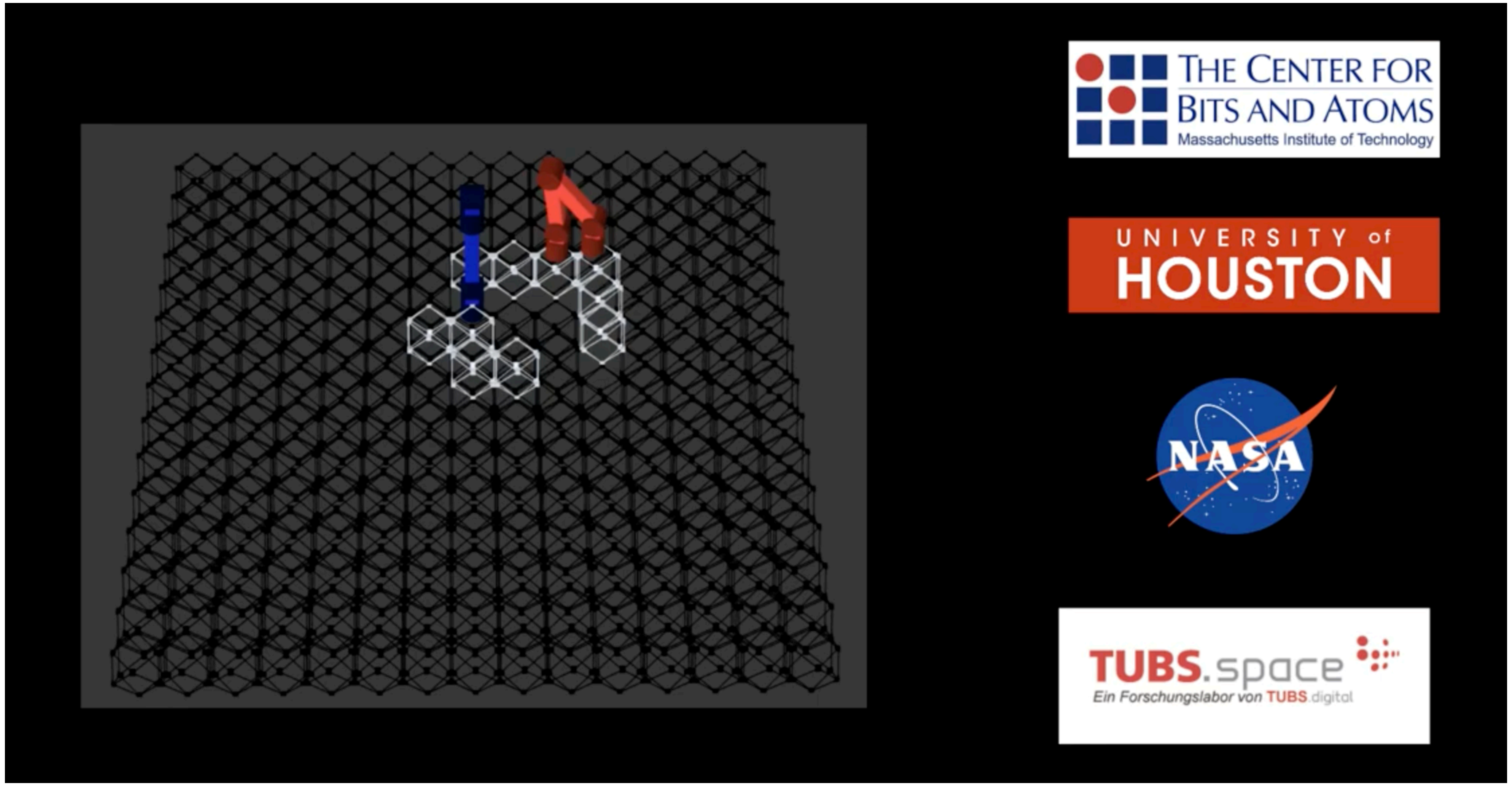
Algorithmen und Datenstrukturen 2
Sommer 2020

Prof. Dr. Sándor Fekete

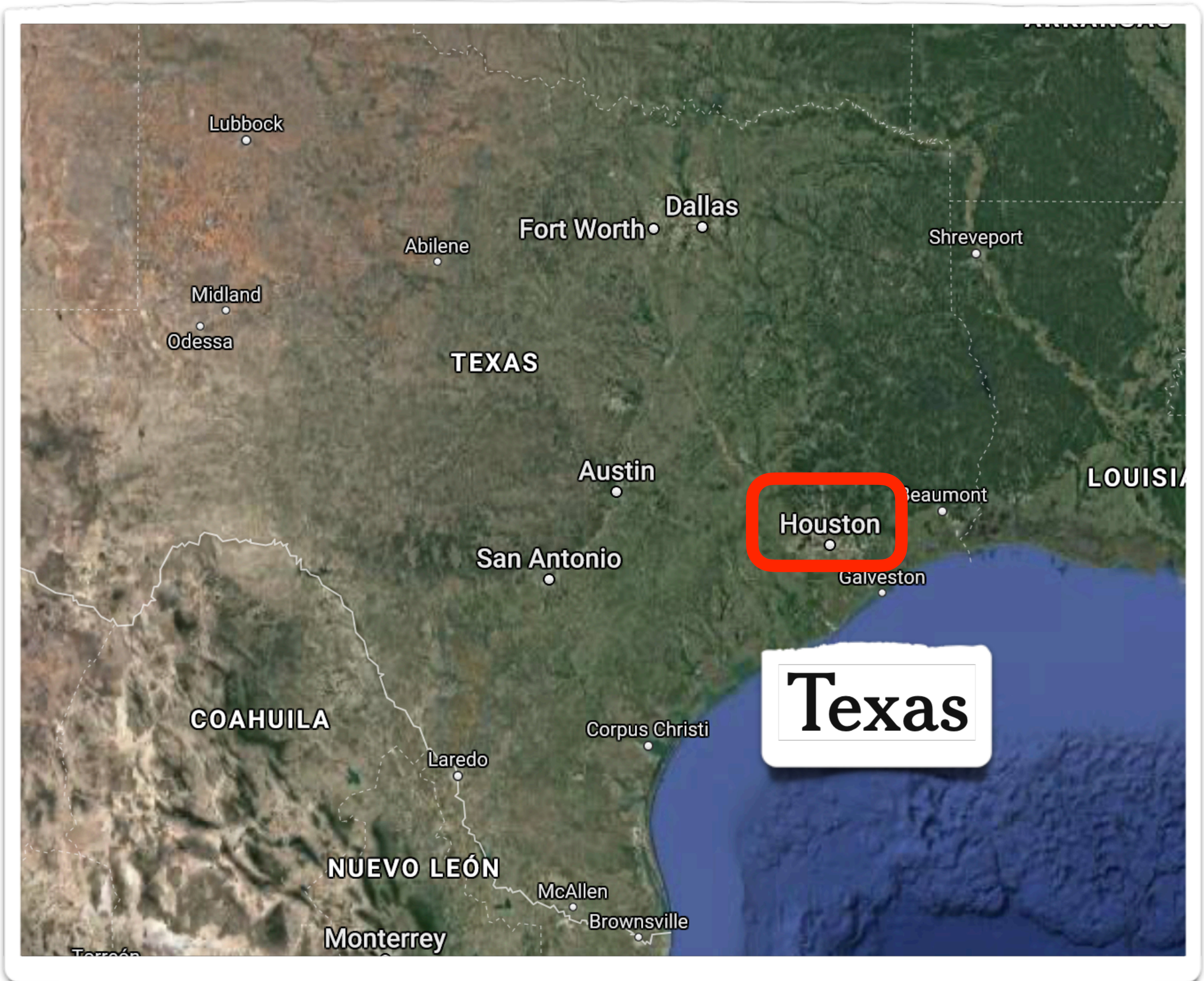
7.1 Motivation

Aufgabenstellung

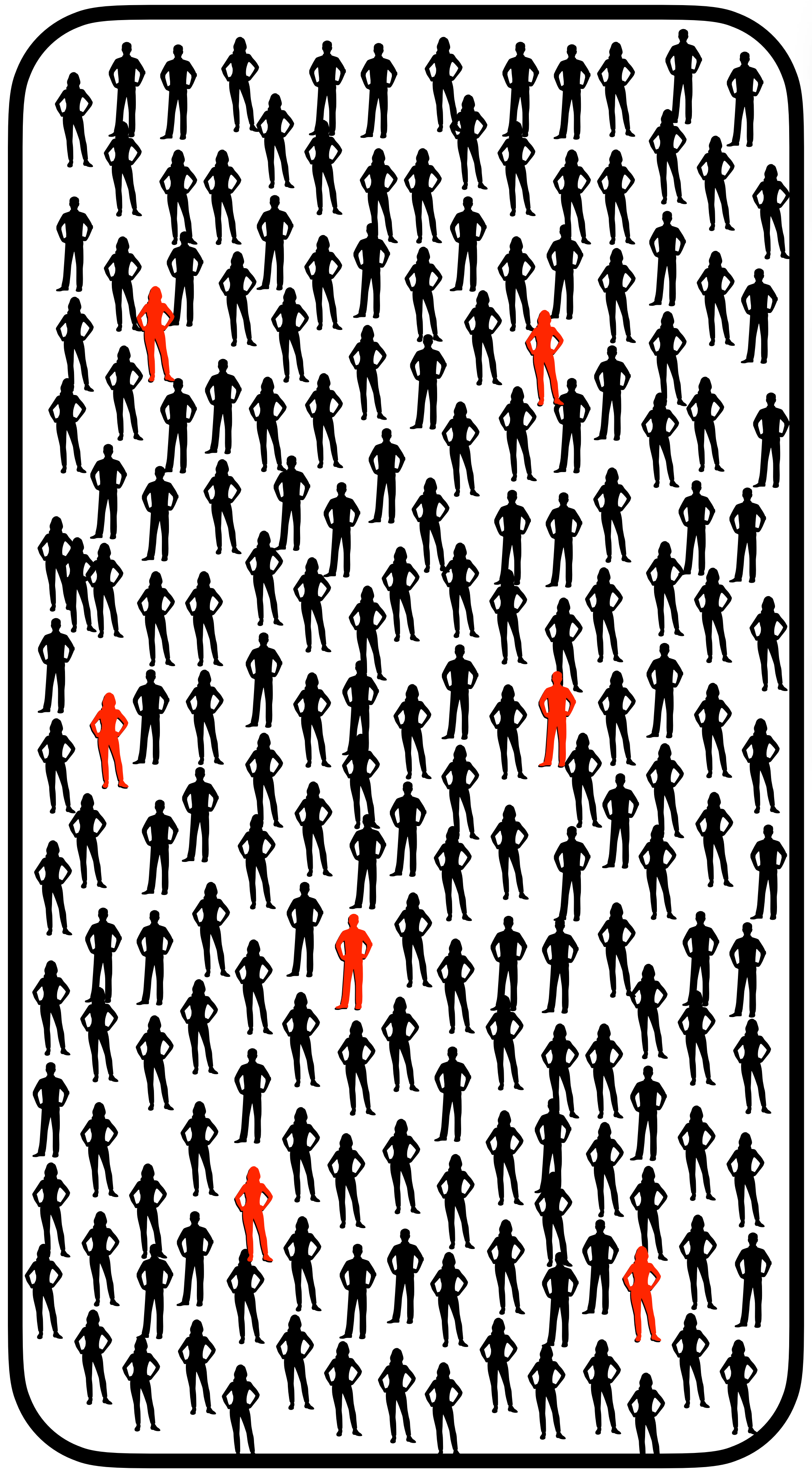
Aufgabenstellung



Aufgabenstellung



Aufgabenstellung

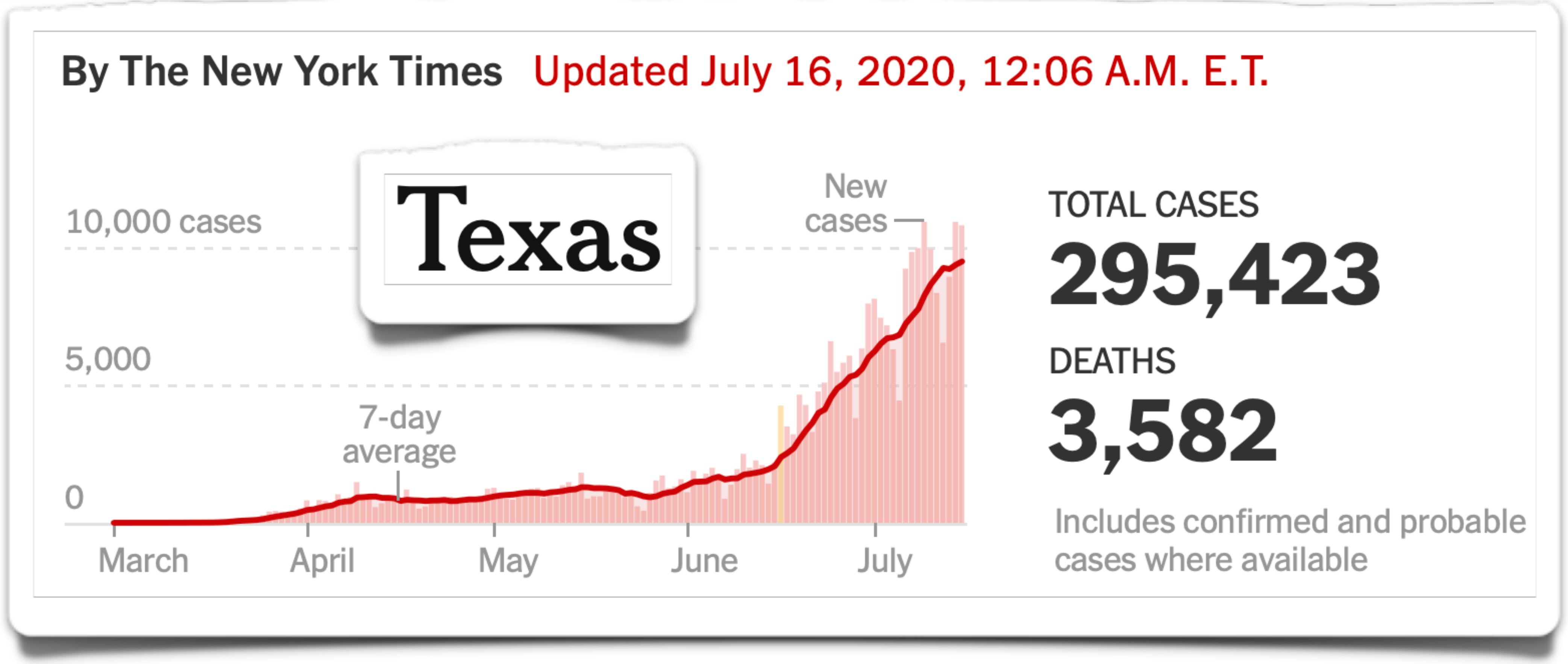


The New York Times

July 13, 2020

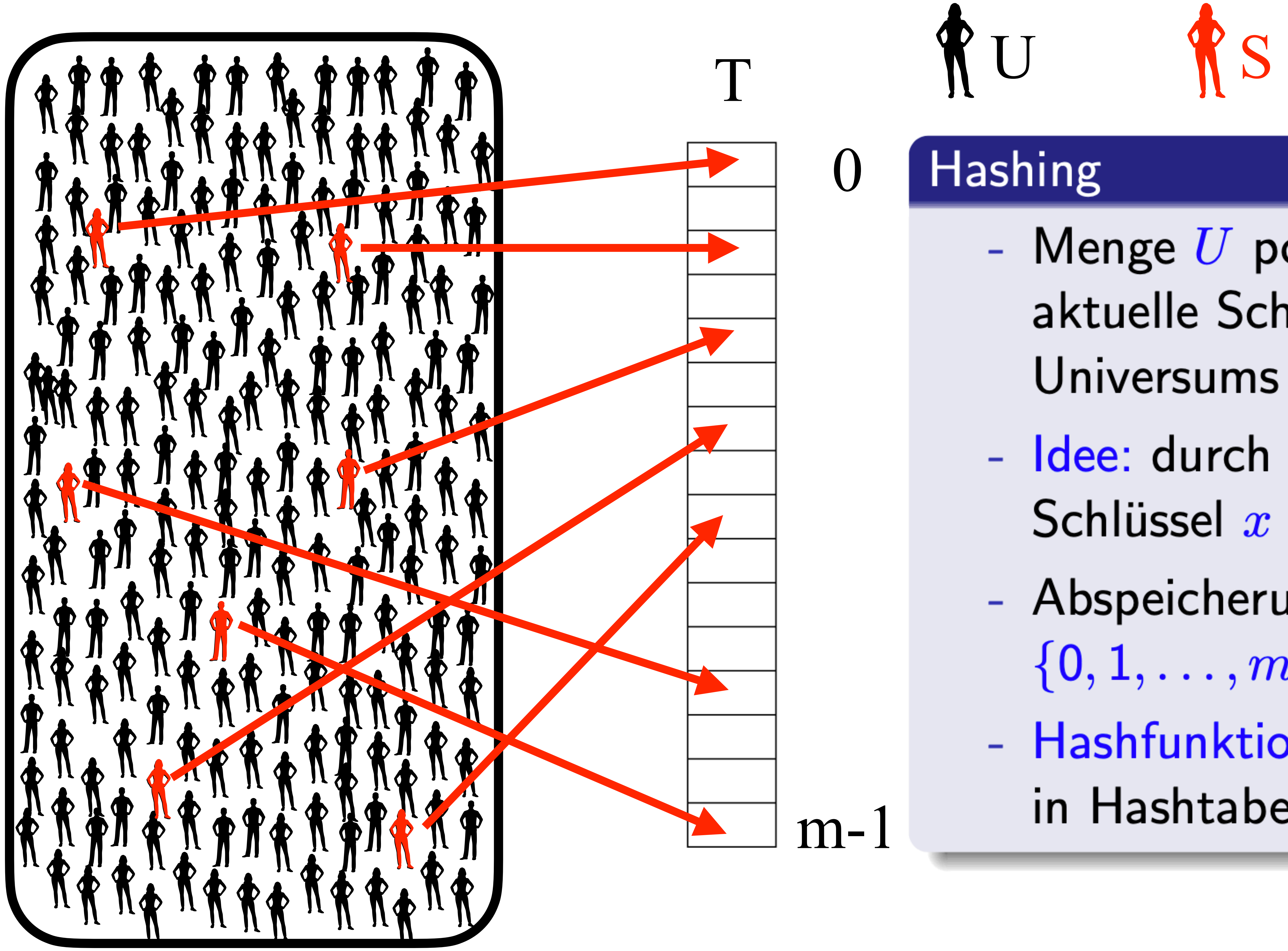
Bottleneck for U.S. Coronavirus Response: The Fax Machine

Before public health officials can manage the pandemic, they must deal with a broken data system that sends incomplete results in formats they can't easily use.



Stacks of coronavirus test results at the Harris County Public Health department in Houston. They arrived by mistake and had to be sent to other jurisdictions.

Aufgabenstellung



Hashing

- Menge U potentieller Schlüssel sehr groß, aktuelle Schlüsselmenge S jeweils nur kleine Teilmenge des Universums (im allgemeinen S nicht bekannt)
- Idee: durch Berechnung feststellen, wo Datensatz mit Schlüssel x gespeichert
- Abspeicherung der Datensätze in einem Array T mit Indizes $\{0, 1, \ldots, m-1\}$: Hashtabelle
- Hashfunktion h liefert für jeden Schlüssel $x \in U$ eine Adresse in Hashtabelle, d.h. $h : U \to \{0, 1, \ldots, m-1\}$.

7.2 Aufgabenstellung

Aufgabenstellung

Aufgabe

Dynamische Verwaltung von Daten, wobei jeder Datensatz eindeutig durch einen Schlüssel charakterisiert ist

Viele Anwendungen benötigen nur einfache Daten-Zugriffsmechanismen (dictionary operations):

- Suche nach Datensatz bei gegebenem Schlüssel x
  search(x)
- Einfügen eines neuen Datensatzes d mit Schlüssel x
  insert(x, d) (abgekürzt insert(x))
- Entfernen eines Datensatzes bei gegebenem Schlüssel x
  delete(x)

Menge potentieller Schlüssel (Universum) kann sehr groß sein!

Alternativen

Sei $n := |S|$.

- Balancierte Suchbäume (AVL-Bäume, B-Bäume):
  dictionary operations haben Komplexität $O(\log n)$
- Hashing:
  für alle Operationen mittlere Komplexität $O(1)$

Belegungsfaktor

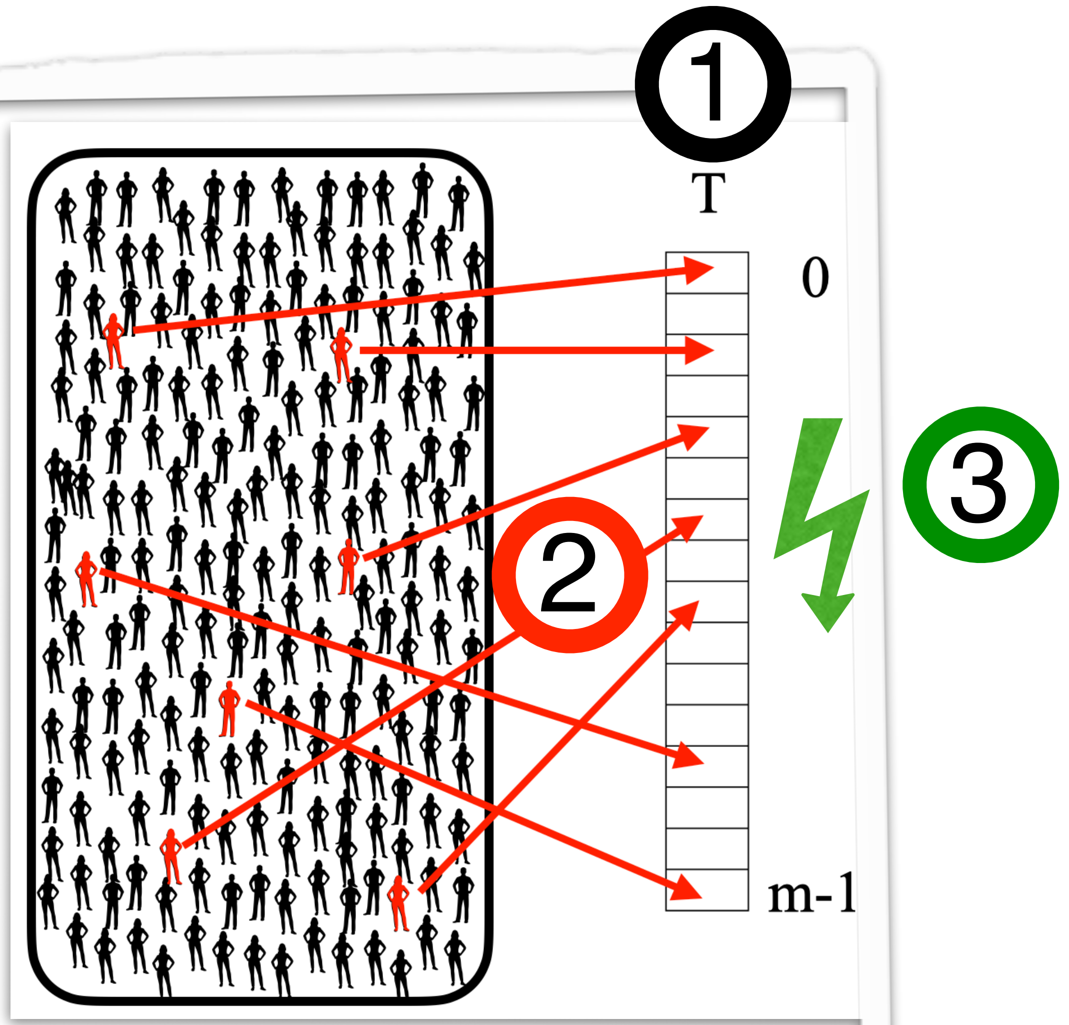
Quotient $\beta := n/m$ heißt Belegungsfaktor oder Auslastungsfaktor einer Hashtabelle der Größe m.

Mittlerer Aufwand für dictionary operations als Funktion in β abschätzbar
$\longrightarrow$ Anzahl aktueller Schlüssel geht nur indirekt in Aufwand ein

Herausforderungen

Anzahl möglicher Schlüssel viel größer als Hashtabelle, also $|U| >> m$

- Hashfunktion muss verschiedene Schlüssel x_1 und x_2 auf gleiche Adresse abbilden.
- x_1 und x_2 beide in aktueller Schlüsselmenge $\longrightarrow$ Adresskollision



Hashverfahren gegeben durch:

1 eine Hashtabelle,

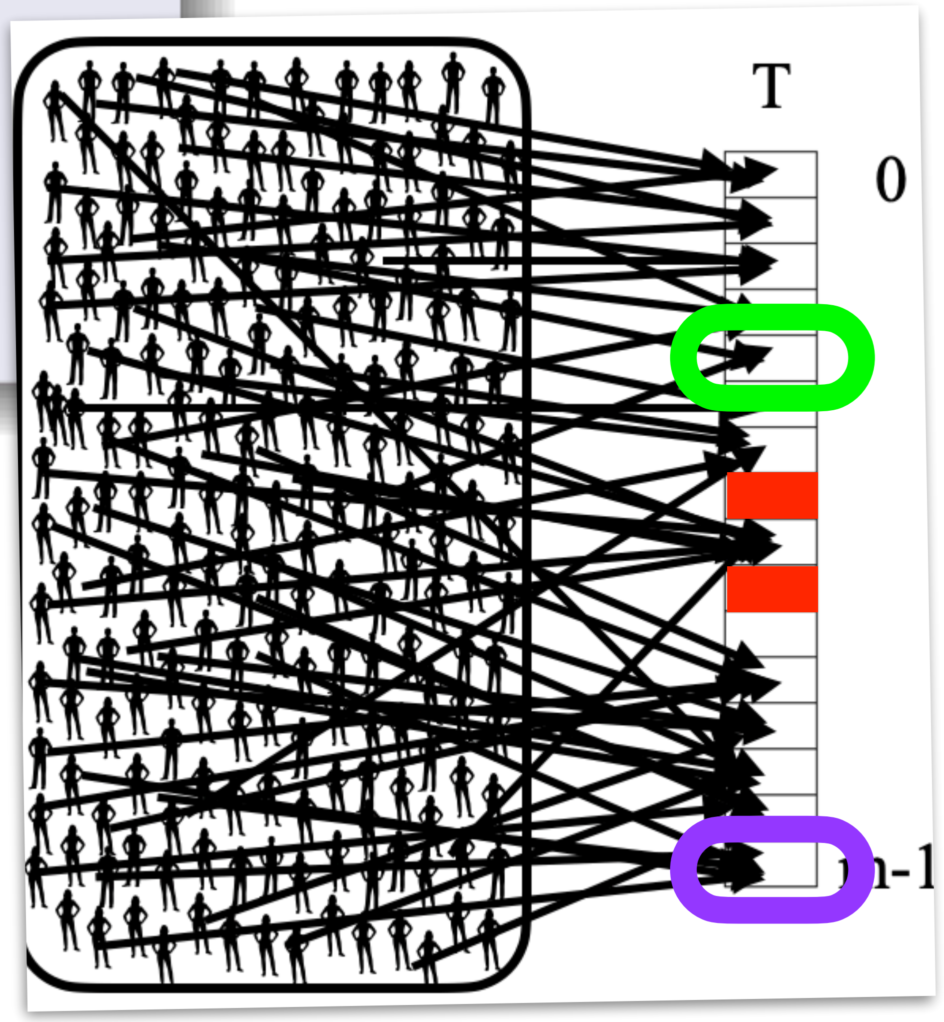
2 eine Hashfunktion, die Universum der möglichen Schlüssel auf Adressen einer Hashtabelle abbildet,

3 eine Strategie zur Auflösung möglicher Adresskollisionen.

Anforderungen

Gute Hashfunktionen sollten:

- surjektiv sein, d.h. den ganzen Wertebereich umfassen,
- die zu speichernden Schlüssel (möglichst) gleichmäßig verteilen, d.h. für alle Speicherplätze i und j sollte gelten $|h^{-1}(i)| \approx |h^{-1}(j)|$,
- effizient berechenbar sein.



7.3 Hashfunktionen

Divisions-Rest-Methode

Divisions-Rest-Methode

$$h(x) = x \bmod m := x - \left\lfloor \frac{x}{m} \right\rfloor \cdot m$$

Beispiel: m=11, S={49, 22, 6, 52, 76, 34, 13, 29}

Hashwerte:

$$\begin{aligned}
h(49) &= 5 \\
h(22) &= 0 \\
h(6) &= 6 \\
h(52) &= 8 \\
h(76) &= 10 \\
h(34) &= 1 \\
h(13) &= 2 \\
h(29) &= 7
\end{aligned}$$

Wahl von m

Problem: Daten oft nicht gleichverteilt!

Beispiel: Texte in Zahlen übertragen, oft viele Leerzeichen, bestimmte Wörter häufiger etc.

Wichtig: geeignete Wahl von m

- m Zweierpotenz: $x \bmod m$ wählt nur die letzten $\log m$ Bits
- m Primzahl: $x \bmod m$ *beeinflusst* alle Bits

7.4 Kollisionen

Auftreten von Kollisionen

Zur Erinnerung:

Im Allgemeinen unvermeidbar, dass Kollisionen auftreten, denn aus $N >> m$ folgt Existenz eines Speicherplatzes i mit $|h^{-1}(i)| \geq N/m$.

Frage:

Sei $n := |S|$. Wie wahrscheinlich sind Kollisionen bei $n << m$?

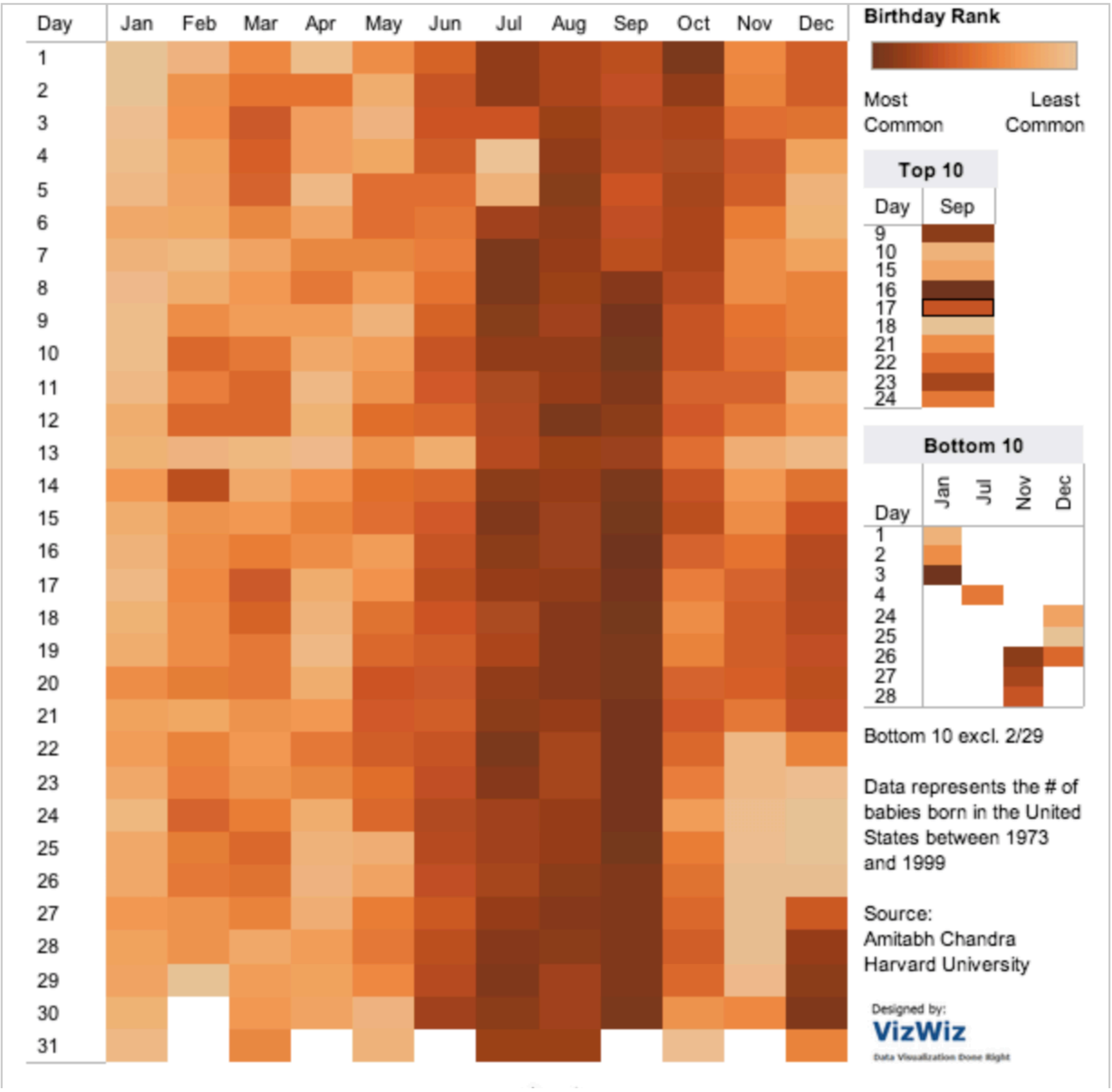
Geburtstagsparadoxon

Bei wie vielen (zufällig gewählten) Personen ist es *wahrscheinlich*, dass hiervon zwei am selben Datum (Tag und Monat) Geburtstag haben?

Geburtstagsparadoxon

Annahme:

- Daten unabhängig
- Prob($h(x) = j$)=$1/m$



Geburtstagsparadoxon

Annahme:

- Daten unabhängig
- Prob($h(x) = j$)=$1/m$

Prob(i-tes Datum kollidiert nicht mit den ersten $i-1$ Daten, wenn diese kollisionsfrei sind)= $\frac{m-(i-1)}{m}$

Intuition:

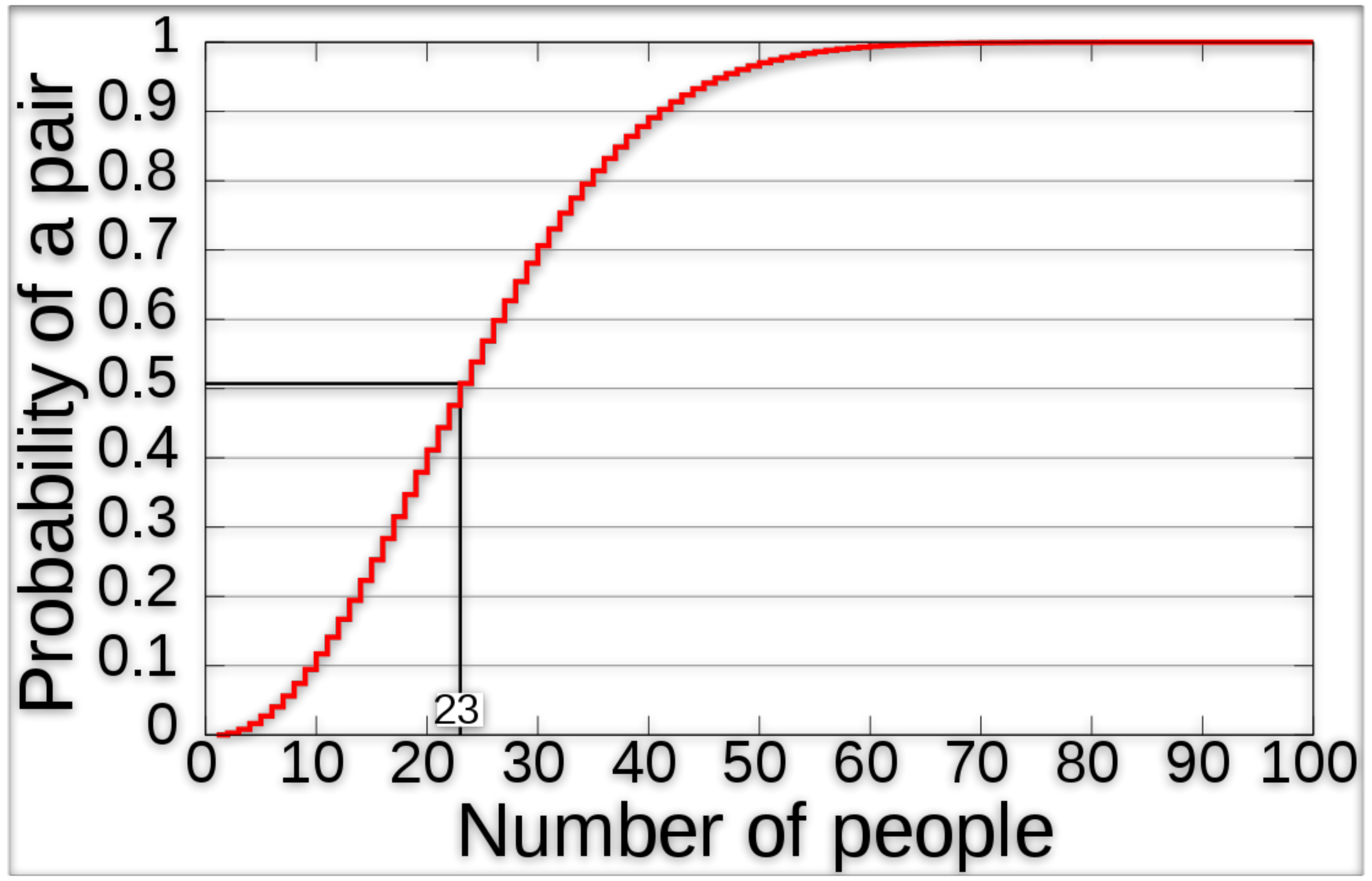
Egal welche Speicherplätze die ersten $i-1$ Daten belegen, $m-i+1$ der m Möglichkeiten sind *gut*.

Prob(n Daten kollisionsfrei) $= \frac{m-1}{m} \cdot \frac{m-2}{m} \cdots \frac{m-n+1}{m}$

Beispiel: $m = 365$

Prob(23 Daten kollisionsfrei) ≈ 0.49

Prob(50 Daten kollisionsfrei) ≈ 0.03



Geburtstagsparadoxon (verallgemeinert)

$$\text{Prob}(2m^{1/2} \text{ Daten kollisionsfrei}) =$$

$$\underbrace{\frac{m-1}{m}\cdots}_{\leq\ 1}\ \cdot\ \underbrace{\frac{m-m^{1/2}}{m}\cdots\frac{m-2m^{1/2}+1}{m}}_{\left(\frac{m-m^{1/2}}{m}\right)^{m^{1/2}}}$$

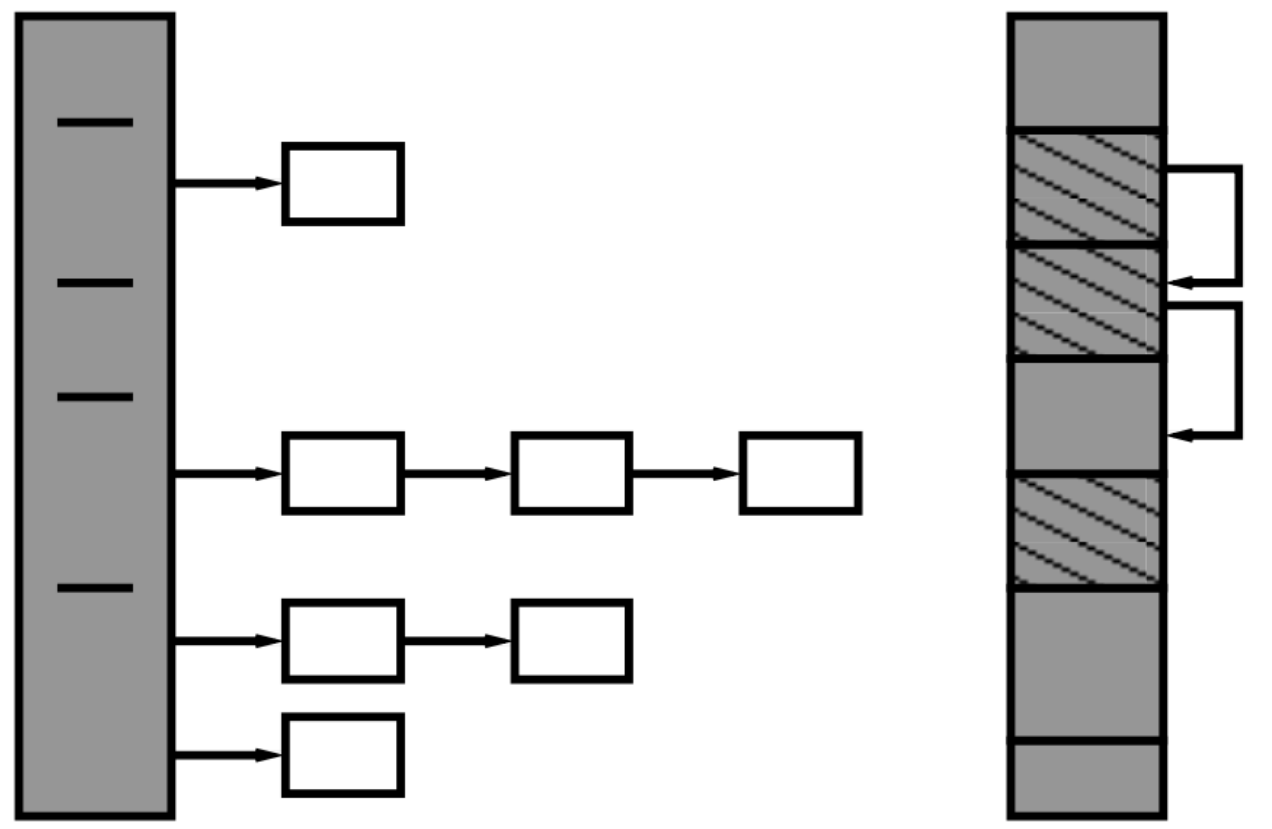
$$\leq 1 \cdot \left(\frac{m-m^{1/2}}{m}\right)^{m^{1/2}} = \left(1-\frac{1}{m^{1/2}}\right)^{m^{1/2}} \approx \frac{1}{e}$$

Hashing muss mit Kollisionen leben und benötigt Strategien zur Kollisionsbehandlung!

Kollisionbehandlung

Verschiedene Arten der Kollisionsbehandlung:

- mittels verketteter Listen (links)
- mittels offener Adressierung (rechts)



7.5 Verkettung von Überläufern

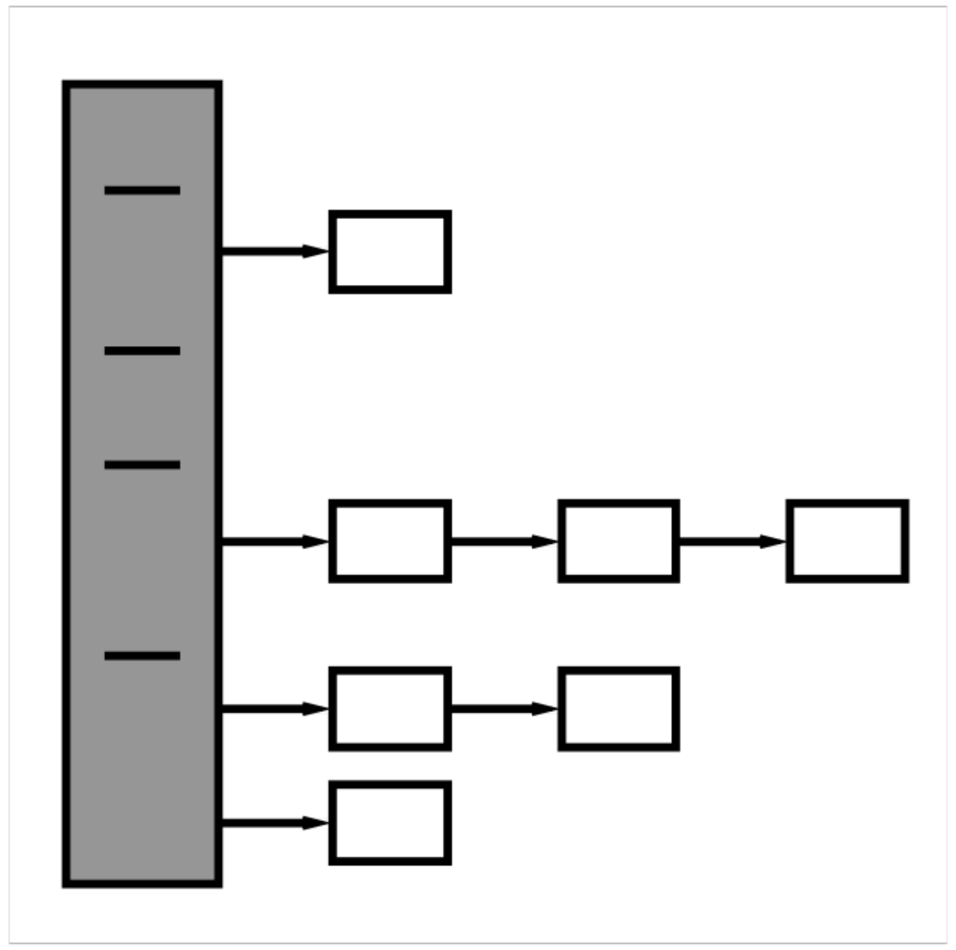
Verkettetung: Realisierung

Realisierung:

Jede Komponente der Hashtabelle enthält Zeiger auf paarweise disjunkte lineare Listen. Die i-te Liste $L(i)$ enthält alle Schlüssel $x \in S$ mit $h(x) = i$.

Vorteil: Alle Operationen werden unterstützt und $n > m$ ist möglich. (Für $n >> m$ jedoch Rehashing ratsam) Nachteil: Speicherplatzbedarf für Zeiger

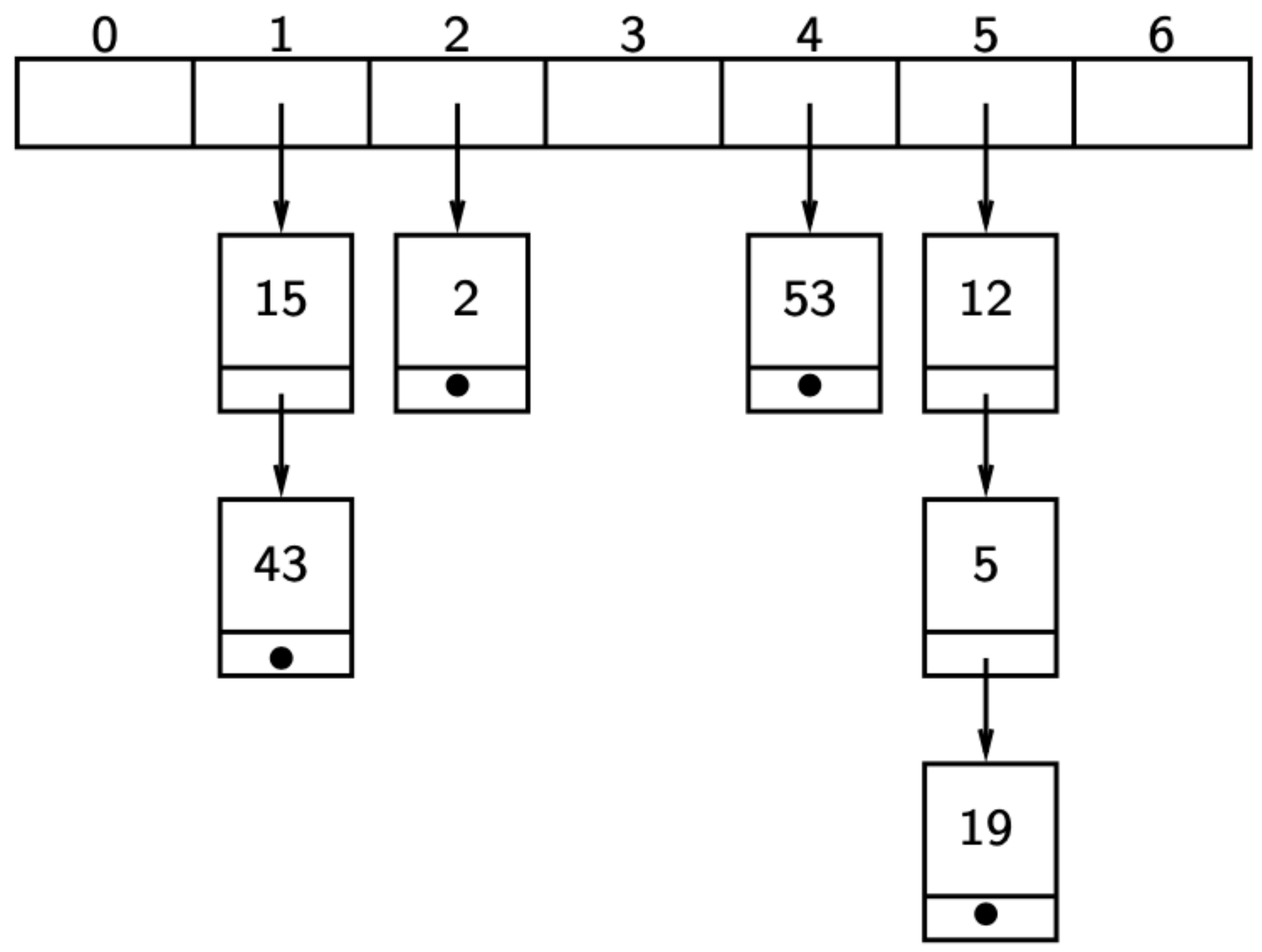
- search(x): Berechne $h(x)$ und suche in Liste $L(h(x))$.
- insert(x) (nach erfolgloser Suche): Berechne $h(x)$ und füge x in Liste $L(h(x))$ ein.
- delete(x) (nach erfolgreicher Suche): Berechne $h(x)$, suche x in Liste $L(h(x))$ und entferne x.



Beispiel

Beispiel: $m = 7$ und $h(x) = x \bmod m$

$S = \{2, 12, 5, 15, 19, 43, 53\}$



Analyse

Bei zufälligen Daten und ideal streuenden Hashfunktion gilt für

$$X_{ij} := \begin{cases} 1 & i\text{-tes Datum kommt in Liste } L(j) \\ 0 & \text{sonst} \end{cases}$$

$\text{Prob}(X_{ij} = 1) = \frac{1}{m}$

$\rightarrow \text{E}(X_{ij}) = 1 \cdot \frac{1}{m} + 0 \cdot \frac{m-1}{m} = \frac{1}{m}$

$X_j = X_{1j} + \cdots + X_{nj}$ zählt Anzahl Daten in Liste $L(j)$.

$\text{E}(X_j) = \text{E}(X_{1j} + \cdots + X_{nj}) = \text{E}(X_{1j}) + \cdots + \text{E}(X_{nj}) = \frac{n}{m}$

Analyse (2)

- Erfolglose Suche in Liste $L(j)$:

  Inklusive nil-Zeiger durchschnittlich $1 + \frac{n}{m} = 1 + \beta$ Objekte betrachten

  Beispiel: Für $n \approx 0.95 \cdot m$ ist dies ≈ 1.95.

- Erfolgreiche Suche in Liste $L(j)$ der Länge ℓ:

  Jede Position in der Liste hat Wahrscheinlichkeit $1/\ell$, also $\frac{1}{\ell}(1 + 2 + \cdots + \ell) = \frac{\ell+1}{2}$.
  Durchschnittliche Listenlänge hier: $1 + \frac{n-1}{m}$
  (Liste enthält sicher das gesuchte Datum, und die anderen $n - 1$ Daten sind zufällig verteilt.)
  Also erwartete Suchdauer $\frac{1}{2}(1 + \frac{n-1}{m} + 1) = 1 + \frac{n-1}{2m} \approx 1 + \frac{\beta}{2}$
  Beispiel: Für $n \approx 0.95 \cdot m$ ist dies ≈ 1.475.

7.6 Offene Adressierung

Demnächst!

Vielen Dank!

s.fekete@tu-bs.de