# Non-Standard-Datenbanken

## Approximationstechniken

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

# Übersicht

- Semistrukturierte Datenbanken (JSON, XML) und Volltextsuche
- Information Retrieval
- Mehrdimensionale Indexstrukturen
- Cluster-Bildung
- Einbettungstechniken
- First-n-, Top-k-, und Skyline-Anfragen
- Probabilistische Datenbanken, Anfragebeantwortung, Top-k-Anfragen und Open-World-Annahme
- Probabilistische Modellierung, Bayes-Netze, Anfragebeantwortungsalgorithmen, Lernverfahren,
- Temporale Datenbanken und das relationale Modell,
- Probabilistische Temporale Datenbanken
- SQL: neue Entwicklungen (z.B. JSON-Strukturen und Arrays), Zeitreihen (z.B. TimeScaleDB)
- Stromdatenbanken, Prinzipien der Fenster-orientierten inkrementellen Verarbeitung
- Approximationstechniken für Stromdatenverarbeitung, Stream-Mining
- Probabilistische raum-zeitliche Datenbanken und Stromdatenverarbeitungsssysteme: Anfragen und Indexstrukturen, Raum-zeitliches Data Mining
- Von NoSQL- zu NewSQL-Datenbanken, CAP-Theorem, CALM-Theorem
- Blockchain-Datenbanken
- Analyse von Graphdaten

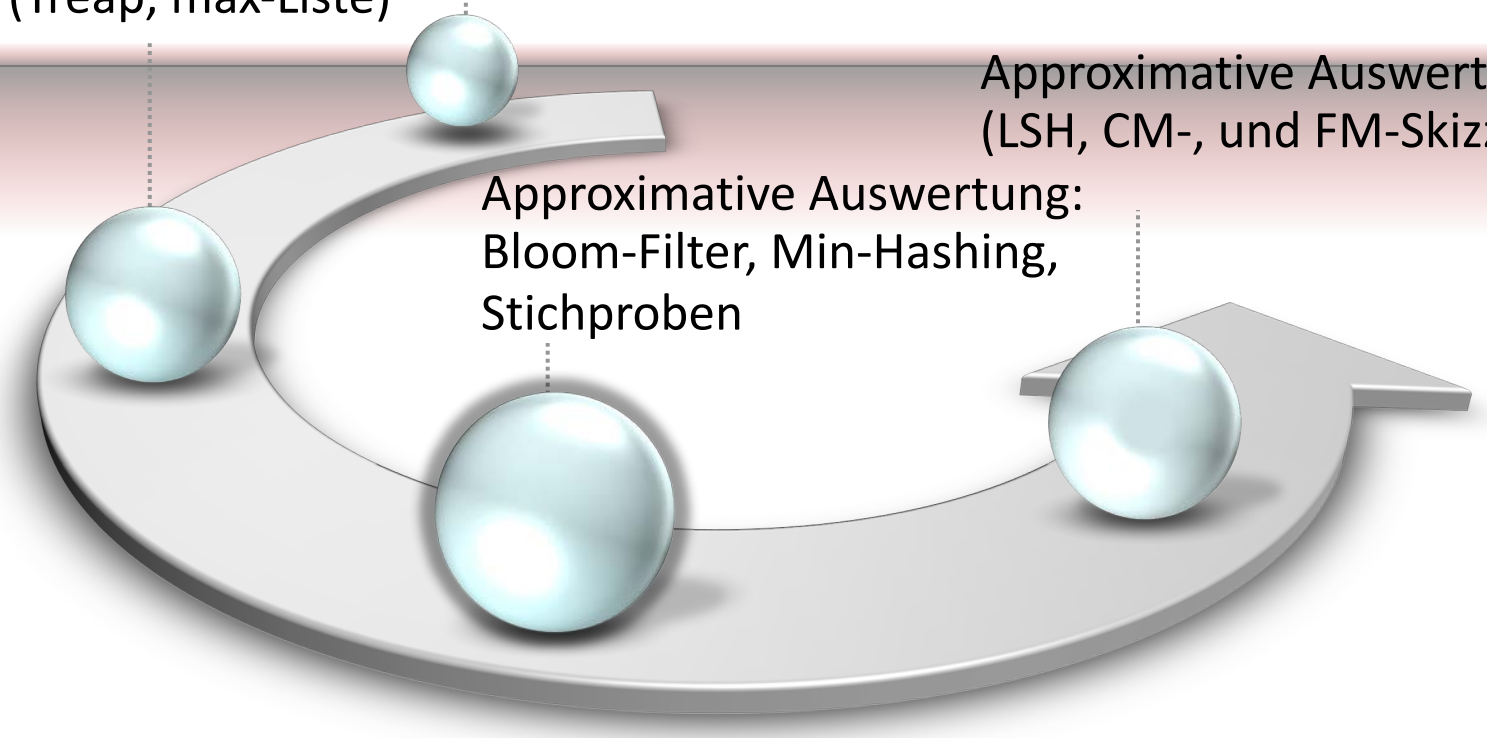
# Übersicht

Intervall-Skyline, inkrementelle Auswertungstechniken Skizzen (Treap, max-Liste)

Stromkonzept, relationale Anfragesprachen, exakte Auswertung, QoS-Angaben

Approximative Auswertung (LSH, CM-, und FM-Skizzen)

Approximative Auswertung: Bloom-Filter, Min-Hashing, Stichproben



# Filterung von Datenströmen

- **Jedes Element eines Datenstroms ist ein Tupel**
- Gegeben sei eine Liste von Schlüsseln **S**
- **Bestimme, welche Tupel eines Strom in S sind**
- **Offensichtliche Lösung: Hashtabelle**
  - Aber was machen wir, wenn **nicht genügend Hauptspeicher bereitsteht**, um alle Elemente von ***S*** in einer Hashtabelle zu platzieren
    - Z.B. könnten wir Millionen von Filtern auf einem Strom evaluieren

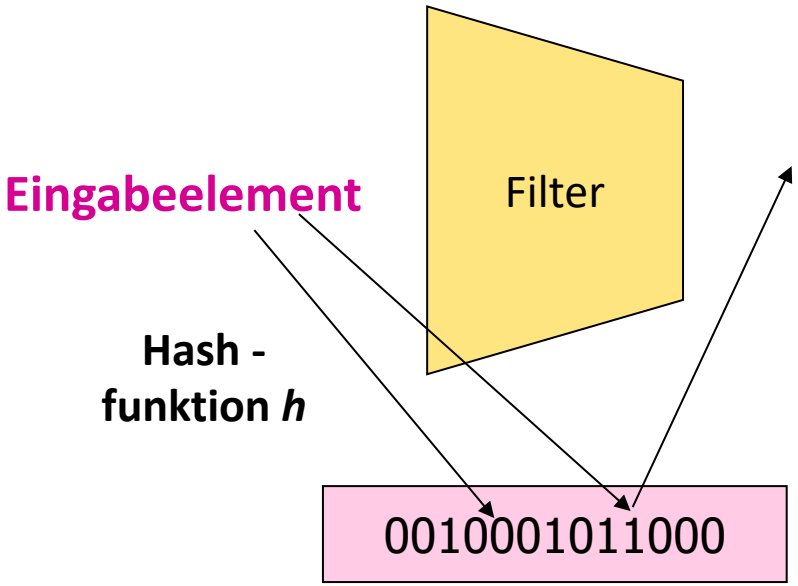
# Anwendungen

- **Email-Spamfilterung**
  - Nehmen wir an, es sind $10^9$ “gute” E-Mail-Adressen bekannt
  - Falls eine E-Mail von einer dieser Adressen kommt, ist sie kein Spam
- **Publish-subscribe-Systeme**
  - Nehmen wir an, jemand möchte Nachrichten zu bestimmten Themen zugesandt bekommen
  - Interessen werden durch Mengen von Schlüsselworten repräsentiert
  - Schlüsselworte von Interessenten müssen mit Nachrichtenstromelementen abgeglichen werden

# Erster Ansatz (1)

- **Gegeben eine Menge von Schlüsseln S, die zur Filterung dienen**
- Erzeuge **Bitfeld *B*** aus ***n*** Bits, anfangs alle ***0***
- Wähle eine **Hash-Funktion *h*** mit Bereich **[*0,n*)**
- Hashe jedes Mitglied von $s \in S$ auf eines der ***n*** Bitfeldelemente und setze das Bit auf **1**: ***B[h(s)]:=1***
- Hashe jedes Element ***a*** des Stroms und gebe nur diejenigen aus, die auf ein Bit hashen, das auf **1** gesetzt ist
  - **Ausgabe *a*** falls **B[h(a)] = 1**

# Erster Ansatz (2)




**Gebe Element aus, es könnte in *S* sein.**
Es hasht auf ein Bitfeld, das auch Hash-Ziel eines Elements in S ist.

**Lasse Element fallen**
Es hasht auf ein Bitelement mit **0**, und es ist daher sicher nicht in ***S***

**Verfahren erzeugt falsch-positive aber keine falsch-negativen Ausgaben**

- Falls Element in ***S***, wird es sicher ausgegeben, ansonsten möglicherweise auch

# Erster Ansatz (3)

- **|S| = 1 Milliarde E-Mail-Adressen**
- **|B|= 1GB = 8 Milliarden Bits**

- Falls die E-Mail-Adresse in ***S*** ist, wird sie sicher auf ein Bitfeldelement gehasht, das mit **1** besetzt ist, so wird sie immer durchgelassen (***keine falsch-negativen Resultate***)

- Ungefähr **1/8** der Bits sind mit **1** besetzt, also werden ca. **1/8** der Addressen nicht in ***S*** durchgelassen (***falsch-positive Resultate***)
  - Tatsächliche sind es weniger als **1/8**, weil mehr als eine Adresse auf das gleiche Bit gehasht wird

# Analyse: Werfen von Pfeilen (1)

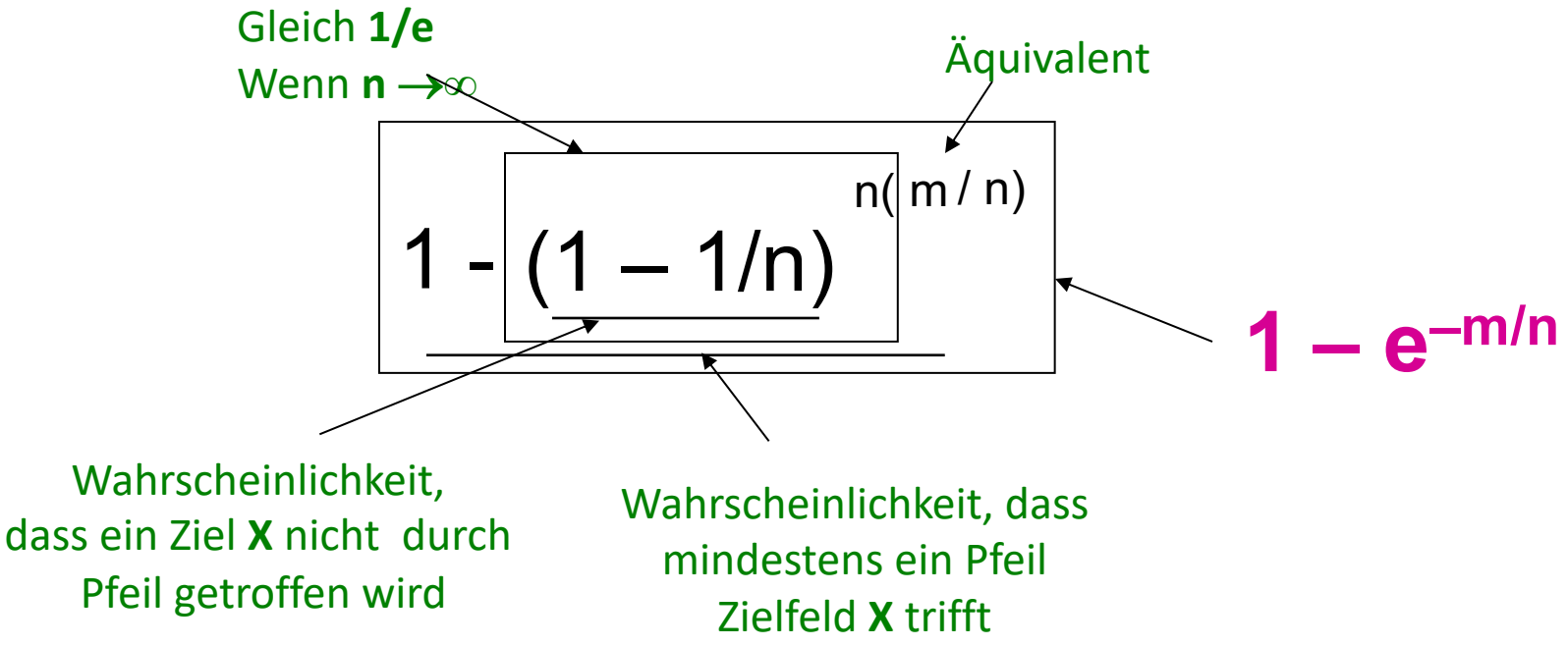
- **Genauere Analyse der Anzahl falsch positiver Ausgaben**

- **Betrachtung:** Wenn wir ***m*** Pfeile in ***n*** gleichermaßen wahrscheinliche Zielfelder werfen, **was ist die Wahrscheinlichkeit, dass ein Zielfeld mindestens einen Pfeil abbekommen?**

- **In unserem Fall:**
  - **Zielfelder** = Bits/Elemente des Hashfeldes
  - **Pfeile** = Hashwerte der Eingabeelemente

# Analyse: Werfen von Pfeilen(2)

- Wir haben ***m*** Pfeile, ***n*** Zielfelder
- **Was ist die Wahrscheinlichkeit, dass ein Zielfeld mindestens einen Pfeilabbekommt?**



# Analyse: Werfen von Pfeilen(3)

- **Anteil der 1en im Bitfeld B =**
  **= Wahrscheinlichkeit eines falsch-positiven Resultats**
  **= $1 - e^{-m/n}$**

- **Beispiel:** **$10^9$** Pfeile, **$8 \cdot 10^9$** Zielfelder
  - Anteil der **1en** in **B = $1 - e^{-1/8}$ = 0.1175**
    - Vergleiche mit Schätzung: **1/8 = 0.125**

# Bloom-Filter

- Betrachte: **|S| = *m*, |B| = *n***
- Verwende ***k*** unabhängige Hash-Funktionen $h_1, \ldots, h_k$
- **Initialisierung:**
  - Besetze **B** mit **0en**
  - Hashe jedes Element $s \in S$ mit jeder Hash-Funktion $h_i$, setze $B[h_i(s)] = 1$ (für jedes $i = 1,.., k$)

Es gibt nur *ein* Bitfeld B!

- **Laufzeit:**
  - Wenn ein Stromelement mit Schlüssel ***x*** erscheint
    - Falls $B[h_i(x)] = 1$ für alle $i = 1,\ldots, k$ dann deklariere, dass ***x*** in ***S***
      - Also, ***x*** hasht auf ein Hashfeld mit **1** für jede Hash-Funktion $h_i(x)$
    - Sonst, ignoriere Element ***x***

# Bloom Filter – Analyse (1)

- **Welcher Anteil des Bitfeldes ist mit 1en gefüllt?**
  - Werfe $k \cdot m$ Pfeile auf $n$ Zielfelder
  - Der Anteil an **1**en ist also $(1 - e^{-km/n})$

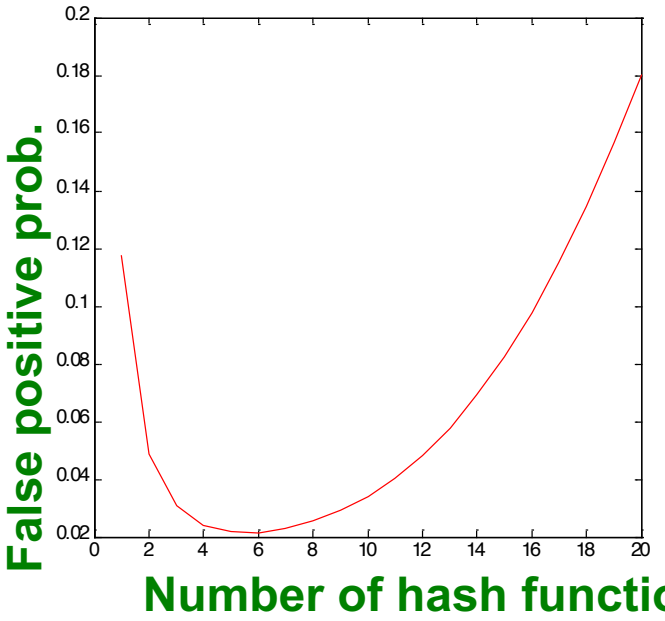
- Aber wir haben $k$ unabhängige Hash-Funktionen und wir lassen $x$ nur durch, falls **alle** $k$ Hashvorgängen das Element $x$ auf eine Bitfeld mit Wert **1** abbilden

- Also ist die **Wahrscheinlichkeit für ein falsch-positives** Resultat $= (1 - e^{-km/n})^k$

Burton H. Bloom: *Space/Time Trade-offs in Hash Coding with Allowable Errors*. In: *Communications of the ACM*. Band 13, Nr. 7, Juli **1970**

# Bloom-Filter – Analyse (2)

- ***m* = 1 billion, *n* = 8 billion**
  - **k = 1**: $(1 - e^{-1/8}) =$ **0.1175**
  - **k = 2**: $(1 - e^{-1/4})^2 =$ **0.0493**




- **Was passiert, wenn *k* steigt?**

- "Optimaler" Wert für ***k***: ***n/m*** **ln(2)**
  - **In unserem Fall:** Optimal ist **k = 8 ln(2) = 5.54 ≈ 6**
    - **Fehler bei k = 6**: $(1 - e^{-6/8})^6 =$ **0.0215**

# Bloom Filter: Zusammenfassung

- **Bloom-Filter garantieren, dass falsch-positive Ausgaben selten sind und verwenden begrenzten Speicher**
  - Gut zur Vorverarbeitung bevor teure Tests durchgeführt werden
- **Geeignet für Hardware-Implementation**
  - Hash-Vorgänge können parallelisiert werden

- Ist es besser, **1** großes **B** zu haben oder ***k*** kleine **B**s?
  - **Es ist gleich:** $(1 - e^{-km/n})^k$ vs. $(1 - e^{-m/(n/k)})^k$
  - Aber **1 großes B** ist einfacher

# Filterung von Datenströmen

- **Fenster-weise Verarbeitung eines Datenstroms**
- Gegeben sei eine Liste von Schlüsselworten **S**
- **Bestimme Ähnlichkeit von Tupeln im Fenster mit Schlüsseln in S**
- **Anwendung: Erkennen von bestimmten Mustern**
  - **Z.B. bei Data Mining**

Andrei Broder, Michael Mitzenmacher: *Network applications of bloom filters: A survey*. In: *Internet Mathematics*. **2002**, S. 636–646

# Min-Hashing

- Gegeben zwei Mengen $A$ und $B$ von Objekten
- Zum Beispiel sind $A$ und $B$ Text-Dokumente mit Wörtern.
- Ein oft benutztes Ähnlichkeitsmaß ist der **Jaccard Koeffizient**:

$$J(A,B) := \frac{|A \cap B|}{|A \cup B|}$$

*Siehe Teil Information Retrieval in dieser Vorlesung*

## Idee hinter Min-Hashing

- Berechne für alle Elemente einer Menge $A$ eine Hashfunktion, die $a \in A$ einem Integer-Wert zuweist.

*Siehe Wörterbucher in AuD*

- Betrachte den kleinsten dieser Werte $h_{min}(A)$
- Wie groß ist die Wahrscheinlichkeit, dass für zwei Mengen A und B dieser min-Wert identisch ist?

Andrei Broder, On the resemblance and containment of documents. In SEQUENCES '97 Proceedings of the Compression and Complexity of Sequences, **1997**

# Beispiel

$S_1 = \{\text{Apfel, Birne, Tomate, Orange}\}$
$S_2 = \{\text{Tomate, Zitrone, Orange, Gurke}\}$
$S_3 = \{\text{Kokosnuss, Aprikose, Banane}\}$

| Objekt | Hashwert |
|---|---|
| Apfel | 263228505 |
| Birne | 1512322680 |
| Tomate | 2655545330 |
| Orange | 2426202636 |
| Zitrone | 4196103473 |
| Gurke | 3877529293 |
| Kokosnuss | 2846776306 |
| Aprikose | 41486361 |
| Banane | 4138599105 |

# Beispiel

$S_1$ ={Apfel, Birne, Tomate, Orange}
$S_2$ ={Tomate, Zitrone, Orange, Gurke}
$S_3$ ={Kokosnuss, Aprikose, Banane}

| Objekt | Hashwert |
|---|---|
| Apfel | 263228505 |
| Birne | 1512322680 |
| Tomate | 2655545330 |
| Orange | 2426202636 |
| Zitrone | 4196103473 |
| Gurke | 3877529293 |
| Kokosnuss | 2846776306 |
| Aprikose | 41486361 |
| Banane | 4138599105 |

Wir haben also

$$h_{min}(S_1) = 263228505$$
$$h_{min}(S_2) = 2426202636$$
$$h_{min}(S_3) = 41486361$$

# Min-Hashing (2)

- Die Wahrscheinlichkeit $Pr[h_{min}(A) = h_{min}(B)]$ steht in direkter Verbindung zum Jaccard-Koeffizienten:

$$Pr[h_{min}(A) = h_{min}(B)] = \frac{|A \cap B|}{|A \cup B|}$$

## Anwendung

- Suche nach ähnlichen Mengen: Betrachte nur Paare von Mengen, deren min-Wert identisch ist
- Bzw., nehme $Pr[h_{min}(A) = h_{min}(B)]$ als Näherung für den Jaccard-Koeffizienten
- Wie gut funktioniert das?

# Beispiel

$S_1$ ={Apfel, Birne, Tomate, Orange}
$S_2$ ={Tomate, Zitrone, Orange, Gurke}
$S_3$ ={Kokosnuss, Aprikose, Banane}

| Objekt | Hashwert |
|---|---|
| Apfel | 263228505 |
| Birne | 1512322680 |
| Tomate | 2655545330 |
| Orange | 2426202636 |
| Zitrone | 4196103473 |
| Gurke | 3877529293 |
| Kokosnuss | 2846776306 |
| Aprikose | 41486361 |
| Banane | 4138599105 |

Wir haben also

$$h_{min}(S_1) = 263228505$$
$$h_{min}(S_2) = 2426202636$$
$$h_{min}(S_3) = 41486361$$

Die Min-Werte sind für alle drei Mengen unterschiedlich. D.h. wir bekommen eine **Schätzung von** $0$ für den Jaccard-Koeffizienten, obwohl dieser für $S_1$ $S_2$ nicht 0 ist.

# Beispiel: zweite Hashfunktion

$S_1$ ={Apfel, Birne, Tomate, Orange}
$S_2$ ={Tomate, Zitrone, Orange, Gurke}
$S_3$ ={Kokosnuss, Aprikose, Banane}

| Objekt | Hashwert 1 | Hashwert 2 |
|---|---|---|
| Apfel | 263228505 | 3747123490 |
| Birne | 1512322680 | 2691314150 |
| Tomate | 2655545330 | 618073562 |
| Orange | 2426202636 | 167471787 |
| Zitrone | 4196103473 | 3040259855 |
| Gurke | 3877529293 | 2452364051 |
| Kokosnuss | 2846776306 | 2613259214 |
| Aprikose | 41486361 | 2319295075 |
| Banane | 4138599105 | 765635320 |

# Beispiel: zweite Hashfunktion

$S_1$ ={Apfel, Birne, Tomate, Orange}
$S_2$ ={Tomate, Zitrone, Orange, Gurke}
$S_3$ ={Kokosnuss, Aprikose, Banane}

| Objekt | Hashwert 1 | Hashwert 2 |
|---|---|---|
| Apfel | 263228505 | 3747123490 |
| Birne | 1512322680 | 2691314150 |
| Tomate | 2655545330 | 618073562 |
| Orange | 2426202636 | 167471787 |
| Zitrone | 4196103473 | 3040259855 |
| Gurke | 3877529293 | 2452364051 |
| Kokosnuss | 2846776306 | 2613259214 |
| Aprikose | 41486361 | 2319295075 |
| Banane | 4138599105 | 765635320 |

Wir haben also

$h1_{min}(S_1) = 263228505$
$h1_{min}(S_2) = 2426202636$
$h1_{min}(S_3) = 41486361$

$h2_{min}(S_1) = 167471787$
$h2_{min}(S_2) = 167471787$
$h2_{min}(S_3) = 765635320$

# Beispiel 2

$S_1$ ={Apfel, Birne, **Tomate**, **Orange**}
$S_2$ ={**Tomate**, Zitrone, **Orange**, Gurke}
$S_3$ ={Kokosnuss, Aprikose, Banane}

$$h1_{min}(S_1) = 263228505$$
$$h1_{min}(S_2) = 2426202636$$
$$h1_{min}(S_3) = 41486361$$

$$h2_{min}(S_1) = 167471787$$
$$h2_{min}(S_2) = 167471787$$
$$h2_{min}(S_3) = 765635320$$

| | Schätzung | Echt |
|---|---|---|
| $J(S_1, S_2)$ | $1/2$ | $2/6$ |
| $J(S_2, S_3)$ | $0$ | $0$ |
| $J(S_1, S_3)$ | $0$ | $0$ |

# Min-Hashing - Mehrere min-Werte bzw. Hashfunktionen

## Mehrere (k) Hash-Funktionen mit je einem min-Wert

- Betrachte $k$ (unabhängige) Hashfunktionen, die jeweils einen min-Wert liefern.
- Approximiere $J(A,B)$ mit Anteil der übereinstimmenden min-Werte.
- Wie im Beispiel zuvor.

## Mehrere (k) min-Werte & eine Hashfunktion

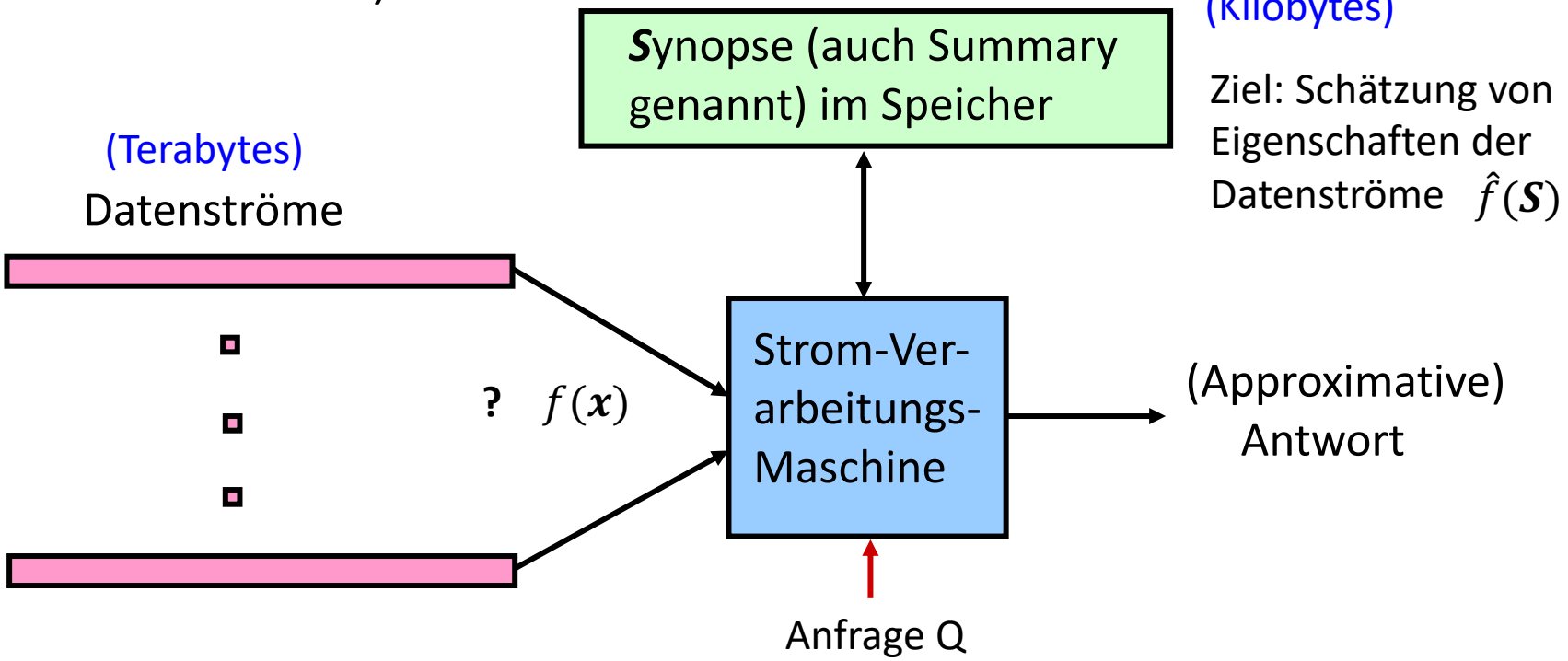
- Betrachte nur eine Hashfunktion, aber nehme von dieser die $k$ kleinsten Werte.

**Der Fehler ist bei beiden Schemata $O(1/\sqrt{k})$.** Ohne Beweis

Approximation mit
Stichproben und Skizzen
Probabilistische Garantien

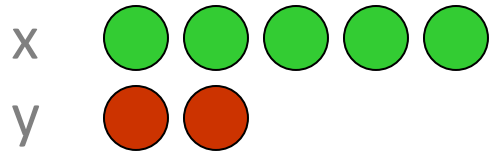
# Approximation

- Daten werden nur einmal betrachtet und
- Speicher für Zustand (stark) begrenzt: O( poly( log( |Strm| ))
- Rechenzeit pro Tupel möglichst klein (auch um Zustand zu modifizieren)

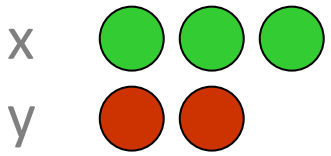


# Stromverarbeitungsmodelle – interner Speicher A[·]

- Absolute Werte (Zeitreihenmodell)
  - Jeder neue Wert pro Zeiteinheit x wird gespeichert (x aus Menge von Zeitpunkten)
- Akkumulierte Werte
  - Nur Eingänge (Akkumulation von Zählern)
    - Zähler immer ≥ 0
    - Multimengenmodell
    - (x, y sind Objekte bzw. Objekttypen)




  - Eingänge und Abgänge (Drehkreuzmodell)
    - Multimengenmodell
    - (x, y sind Objekte bzw. Objekttypen)



# Synopsen

Eine kleine Zusammenfassung großer Datenmengen, die ziemlich gut die interessierenden statistischen Größen abschätzen

**Nützliche Eigenschaften**:

- Es ist leicht, ein Element hinzuzufügen
- Unterstützung für Löschungen ("undo")
- Zusammenführung mehrerer Synopsen sollte leicht sein
  Wichtig für horizontale Skalierung
- Flexibler Einsatz:  Unterstützung mehrerer Anfragetypen

# Was kann über einem Strom berechnet werden?

Einige Funktionen sind leicht:
min, max, sum, …

Es wird jeweils nur eine einzige Variable benötigt:

- Maximum: **Initialisiere $s \leftarrow 0$**

  **For element $x$ do $s \leftarrow \max s, x$**

- Sum: **Initialisiere $s \leftarrow 0$**

  **For element $x$ do $s \leftarrow s + x$**

Die "Synopse" ist ein einzelner Wert
Eine solche Synopse kann zusammengeführt werden

# Approximation und Randomisierung

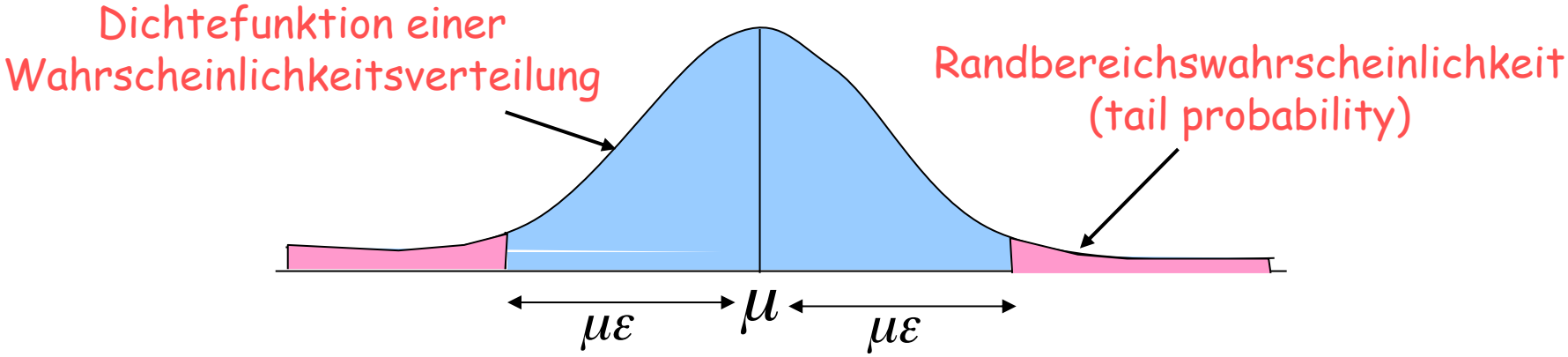
- Viele Probleme sind schwierig, exakt zu berechnen
  - Anzahl der Elemente einer gegebenen Menge von Elementklassen identisch in zwei verschiedenen Strömen?
  - Platzbedarf linear bezogen auf Anzahl der Elementklassen
- **Approximation**: Finde eine Antwort, die korrekt ist bezogen auf einen gegebenen Faktor
  - Finde Antwort im Bereich von $\pm$10% des korrekten Ergebnisses
  - **Genereller**: finde $(1\pm \varepsilon)$-Faktor-Approximation
- **Randomisierung**: Erlaube Fehler, wenn er mit kleiner Wahrscheinlichkeit vorkommt
  - Eine von 10000 Antworten darf falsch sein
  - **Genereller**: Erfolgswahrscheinlichkeit $(1-\delta)$
- *Approximation **und** Randomisierung*: $(\varepsilon, \delta)$-Approximationen

# Probabilistische Garantien

- Benutzerbestimmte $(\varepsilon,\delta)$-Approximationen
  - Beispiel: Tatsächliche Antworten innerhalb von Faktor 1.1 mit Wahrscheinlichkeit $\geq 0.8$ ($\varepsilon = 0.1, \delta = 0.2$)
- Wie können wir prüfen, ob durch ein spezielles Verfahren die Gütekriterien eingehalten werden?
  - Verwendung von Randbereichsabschätzungen

# Exkurs: Randbereichsabschätzungen

- Generelle Grenzen der Randbereichswahrscheinlichkeit einer Zufallsvariable (Wahrscheinlichkeit, dass Wert weit vom Erwartungswert abweicht)




- <u>Basisungleichungen:</u> Sei X eine Zufallsvariable mit Erwartungswert $\mu$ und Varianz Var[X]. Dann gilt für alle $\varepsilon > 0$

Markov-Ungleichung:

$$\Pr(X \geq (1+\varepsilon)\mu) \leq \frac{1}{1+\varepsilon}$$

Chebyshev-Ungleichung:

$$\Pr(|X-\mu| \geq \mu\varepsilon) \leq \frac{\mathrm{Var}[X]}{\mu^2\varepsilon^2}$$

# Stichproben (Abtastung, sampling): Grundlagen

Idee: Ein kleiner Ausschnitt (Stichprobe) *S* einer Datenmenge repräsentiert die Eigenschaften aller Daten

- Zur schnellen Approximierung, wende "modifizierte" Anfrage auf *S* an
- Beispiel: select agg from R where R.e is odd n=12

Datenstrom: 9 3 5 2 7 1 6 5 8 4 9 1

Ausschnitt S: 9 5 1 8

- agg = avg → Mittel der ungeraden Elemente in *S*

Antwort: 5

- agg = count → Mittel über Summanden abgel. aus e in S mit
  - *n* falls *e* ungerade
  - 0 falls *e* gerade

Antwort: 12*3/4 =9

# Randbereichsschätzungen für Summen

- Hoeffding-Ungleichung: Seien $X_1$, ..., $X_m$ unabhängige Zufallsvariablen mit $0 \leq X_i \leq r$

Sei $\bar{X} = \frac{1}{m}\sum_i X_i$ und $\mu$ der Erwartungswert von $\bar{X}$

Dann gilt für alle $\varepsilon > 0$

$$\Pr(|\bar{X} - \mu| \geq \varepsilon) \leq 2\exp^{\frac{-2m\varepsilon^2}{r^2}}$$

- Anwendung für Anfragen mit Mittelwert-Operation (avg) :
  - m ist Größe der Untermenge der Stichprobe S, die das Prädikat erfüllt (3 im Beispiel)
  - r ist Bereich der Elemente in der Stichprobe (9 im Beispiel)
- Anwendung auf Zählanfragen (count):
  - m ist Größe der Stichprobe S (4 im Beispiel)
  - r ist Anzahl der Elemente n im Strom (12 im Beispiel)

J. M. Hellerstein, P. J. Haas, and H. J. Wang. "Online Aggregation". ACM SIGMOD 1997

# Randbereichsschätzungen für Summen (2)

Für Bernoulli-Versuche (zwei mögliche Erg.) sogar stärke Eingrenzung möglich:

- Chernoff Abschätzung: Seien $X_1, \ldots, X_m$ unabhängige Zufallsvariable für Bernoulli-Versuche, so dass $Pr[X_i=1] = p$ ($Pr[X_i=0] = 1-p$)
- Sei $X = \sum_i X_i$ und $\mu = mp$ der Erwartungswert von $X$ dann gilt, für jedes $\varepsilon > 0$

$$\Pr(|X - \mu| \geq \mu\varepsilon) \leq 2\exp^{\frac{-\mu\varepsilon^2}{2}}$$

- Verwendung für Anzahlanfrage (count queries):
  - m ist Größe der Stichprobe S (4 im Beispiel)
  - p ist Anteil der ungeraden Elemente im Strom (2/3 im Beispiel)
- Chernoff-Abschätzung liefert engere Grenzen für Anzahlanfragen als die Hoeffding-Ungleichung

J. M. Hellerstein, P. J. Haas, and H. J. Wang. "Online Aggregation". ACM SIGMOD 1997

# Stichproben für Datenströme

- **(1)** Betrachtung einer **festgelegten Teilmenge** der Elemente, die im Datenstrom auftreten (z.B. 1 von 10)
- **(2)** Extraktion einer **Teilmenge fester Größe** über einem potentiell unendlichen Strom
  - Zu jedem Zeitpunkt k liegt eine Teilmenge der Größe s vor
  - **Was sind die Eigenschaften der Teilmenge, die verwaltet wird?**
    Für alle Zeitpunkte k, soll jedes der n Elemente, die betrachtet wurden, die gleiche Wahrscheinlichkeit haben, in die Teilmenge übernommen zu werden

# Verwendung einer festgelegten Teilmenge

- **Szenario:** Auswertung von Anfragen an eine Suchmaschine
  - **Strom von Tupeln:** (user, query, time)
  - **Stromanfrage:** **How often did a user run the same query in a single day**
  - Ann: Platz für **1/10** des Stroms
- **Naive Lösung:**
  - Generiere ganzzahlige Zufallszahl **[0..9]** für jede Anfrage
  - Speichere Anfrage falls Zahl = **0**, sonst ignoriere Stromelement (eine Anfrage)

# Probleme des naiven Ansatzes

- **Auswertung von: Welcher Bruchteil von Anfragen sind Duplikate?**
  - Nehme an, jeder Benutzer sendet ***x*** Anfragen einmal und ***d*** Anfragen zweimal (insgesamt also ***x*+2*d*** Anfragen)
    - **Korrekte Antwort: *d*/(*x*+*d*)**
  - **Vorgeschlagene Lösung: Verwende 10% der auftretenden Anfragen aus dem Strom**
    - **Problem: Nicht jede Mehrfachanfrage tritt mehrfach in der Stichprobe auf!**
    - **Ergebnis wird verfälscht.**

# Probleme des naiven Ansatzes

- **Auswertung von: Welcher Bruchteil von Anfragen sind Duplikate?**
  - Nehme an, jeder Benutzer sendet ***x*** Anfragen einmal und ***d*** Anfragen zweimal (insgesamt also ***x*+2*d*** Anfragen)
    - **Korrekte Antwort: *d*/(*x*+*d*)**
  - **Vorgeschlagene Lösung: Verwende 10% der auftretenden Anfragen aus dem Strom**
    - Stichprobe enthält ***x*/10** der Einfachanfragen und mindestens **2*d*/10** der Mehrfachanfragen
    - Aber es werden nur ***d*/100** der Paare wirklich als Duplikate erkannt
      - **d/100 = 1/10 · 1/10 · d**
    - Von ***d*** "Duplikaten" kommen ***18d/100*** nur genau einmal in der Stichprobe vor
      - **18d/100 = ((1/10 · 9/10)+(9/10 · 1/10)) · d**
      - **Genau einmal: (P(erstes ja, zweites nicht) + P(erstes nicht, zweitest ja)) · d**
  - **Also wäre die Antwort mit der naiven Stichprobe: (d/100)/(x/10) + d/100 + 18d/100 = d/10x + 19d/100**

# Lösung: Stichprobe mit Benutzern

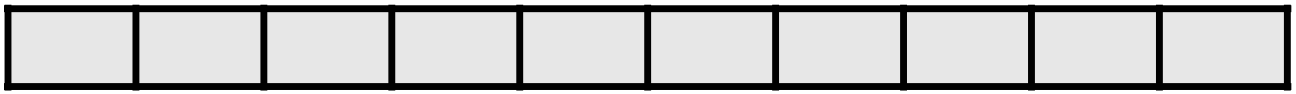
- Wähle **1/10** der **Benutzer** und werte alle ihre Anfagen aus

- Verwende Hash-Funktion, mit der (name, user id) gleichverteilt auf 10 Werte abgebildet wird, speichere Anfragen in Hashtabelle

# Generalisierte Lösung

- **Strom von Tupeln mit Schlüsseln:**
  - Schlüssel = Teilmenge der Tupelkomponenten
    - Z.B.: Tupel = (user, query, time); Schlüssel ist **user**
  - Wahl der Schlüssel hängt von der Anwendung ab

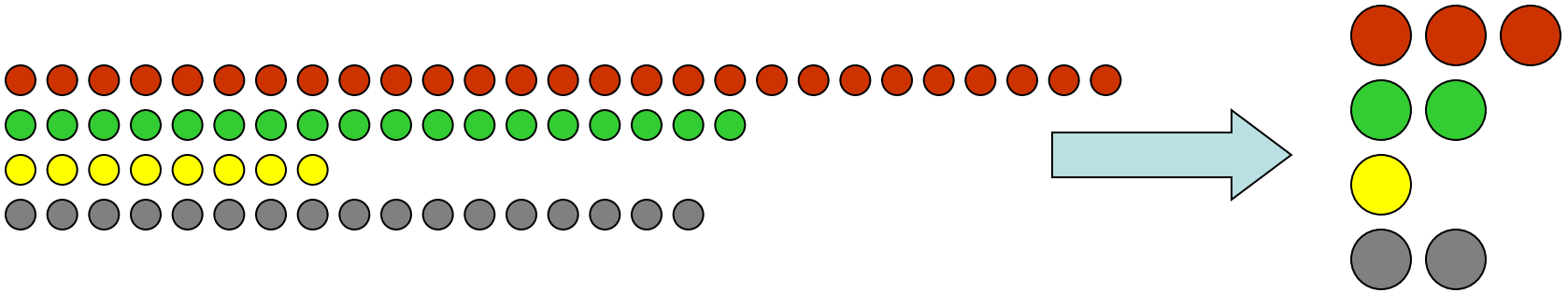
- **Bestimmung einer Stichprobe als *a/b* Bruchteil:**
  - Hashen der Schlüssel auf ***b*** Hashwerte (Eimer)
  - Wähle das Tupel, falls Hashwert höchstens ***a***



**Wie 30% Stichproben generieren?**

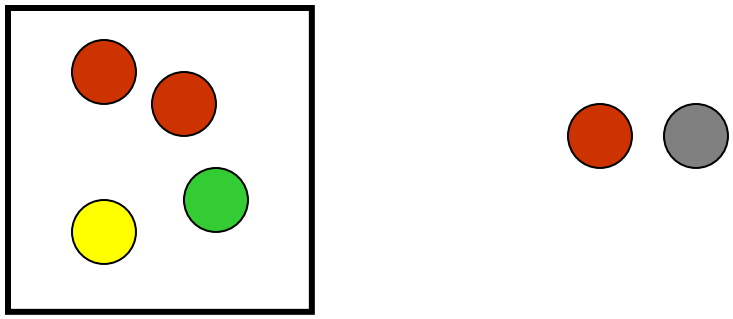
Hash auf b=10 Eimer, wähle Tupel, falls in einen der ersten drei Eimer gehasht

# Stichproben für Datenströme



- Kernproblem: Wähle Stichprobe von m Elementen gleichverteilt aus dem Strom
- Herausforderung: Länge des Stroms nicht bekannt
  - Wann/wie oft Stichprobe entnehmen?
- Zwei Lösungsvorschläge, für verschiedene Anwendungen:
  - Reservoir Sampling (aus den 80ern?)
  - Min-wise Sampling (aus den 90ern?)

# Reservoir Sampling (Reservoir bildet Synopse)



- Übernehme erste m Elemente
- Wähle i-tes Element (i>m) mit Wahrscheinlichkeit m/i
- Wenn neues Element gewählt, ersetze beliebiges Element im Reservoir
- Optimierung: Wenn i groß, berechne jeweils nächstes Element (überspringe Zwischenelemente)



J. S. Vitter. "Random Sampling with a Reservoir". ACM TOMS, **1985**

# Reservoir Sampling – Analyse

- Analysieren wir einen einfachen Fall: m = 1
- Wahrscheinlichkeit, dass i-tes Element nach n Elementen die Stichprobe bildet:
  - Wahrscheinlichkeit, dass i bei Ankunft gewählt wird
    × Wahrscheinlichkeit dass i "überlebt"

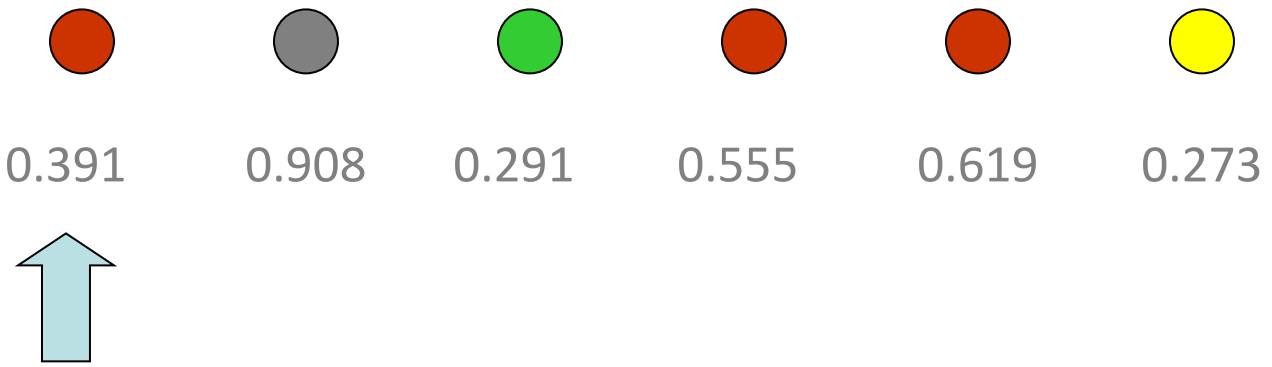
$$\frac{1}{i} \times \left(1 - \frac{1}{i+1}\right) \times \left(1 - \frac{1}{i+2}\right) \ \ldots \ \left(1 - \frac{1}{n-1}\right) \times \left(1 - \frac{1}{n}\right)$$

$$= \frac{1}{\cancel{i}} \times \frac{\cancel{i}}{\cancel{i+1}} \times \frac{\cancel{i+1}}{\cancel{i+2}} \ \ldots \ \frac{\cancel{n-2}}{\cancel{n-1}} \times \frac{\cancel{n-1}}{n} = 1/n$$

- Analyse für m > 1 ähnlich, Gleichverteilung leicht zu zeigen
- Nachteil: Nicht einfach auf mehrere Ströme parallelisierbar

# Min-wise Sampling

- Für jedes Element: Wähle Anteil zwischen 0 und 1
- Wähle Element mit kleinstem Anteilswert



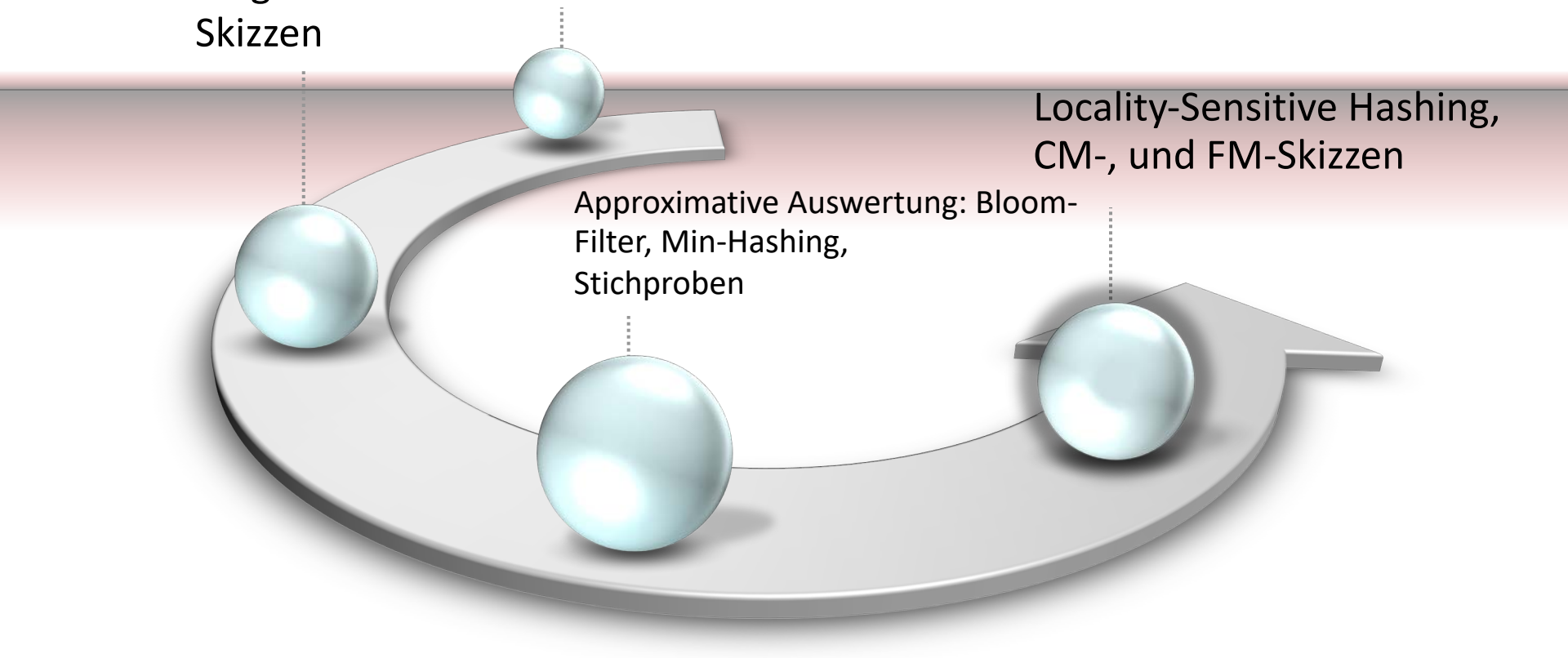

- Jedes Element hat gleiche Chance, kleinstes Element zu werden, also gleichverteilt
- Anwendung auf mehrere Ströme, dann Zusammenführung

S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In ACM SenSys, **2004**

# Übersicht

Intervall-Skyline, inkrementelle Auswertungstechniken Skizzen

Stromkonzept, relationale Anfragesprachen, exakte Auswertung, QoS-Angaben



Locality-Sensitive Hashing, CM-, und FM-Skizzen

Approximative Auswertung: Bloom-Filter, Min-Hashing, Stichproben

# Approximative Ähnlichkeitssuche in hohen Dimensionen

- Vergleiche Daten im Strom mit Daten aus großer Kollektion
- Suche k ähnlichste Dateneinheiten in Kollektion

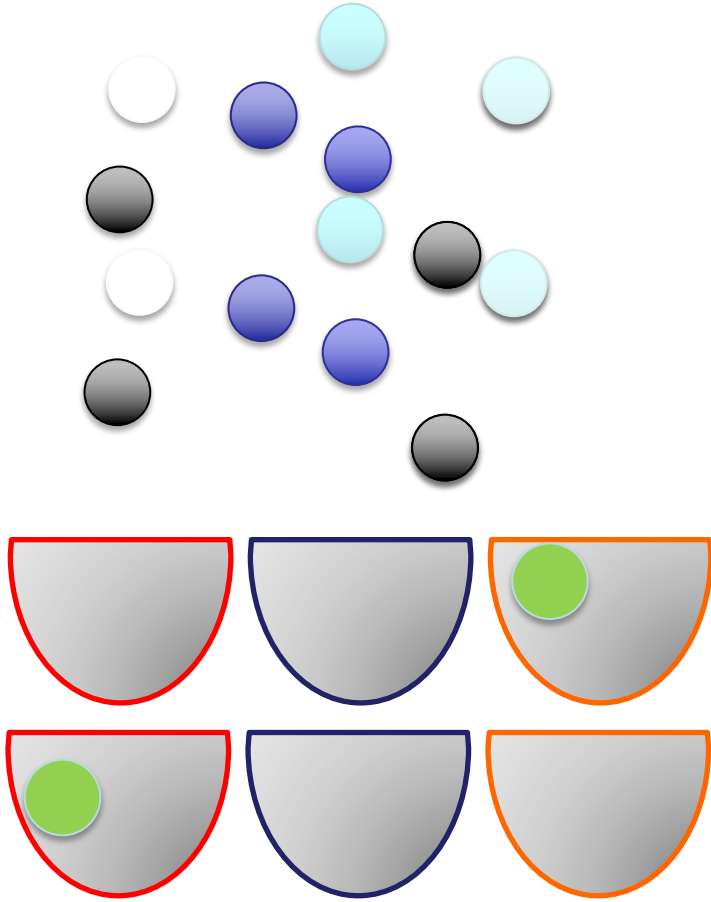
- Anzahl der Vergleichsmerkmale kann sehr groß werden

Gionis, A.; Indyk, P.; Motwani, R. (**1999**). "Similarity Search in High Dimensions via Hashing". Proceedings of the 25th Very Large Database (VLDB) Conference

# Approximative Reduktion der Vergleiche

**Locality Sensitive Hashing** (LSH):

- Jedes Objekt wird in Partition gehasht
- Objekte in der gleichen Partition (Kollision) häufig ähnlich
- Es gibt falsch-negative Ergebnisse (zwei ähnliche Objekten in verschiedenen Partitionen)
- Fehler kann klein gehalten werden
- Um die Wahrscheinlichkeit von Kollisionen zu erhöhen, werden mehrere Hashtabellen aufgebaut
- Vergleiche Objekte mit Objekten aus gleicher Partition in jeder Hashtabelle



Indyk, Piotr.; Motwani, Rajeev. (**1998**). "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality.". Proceedings of 30th Symposium on Theory of Computing.

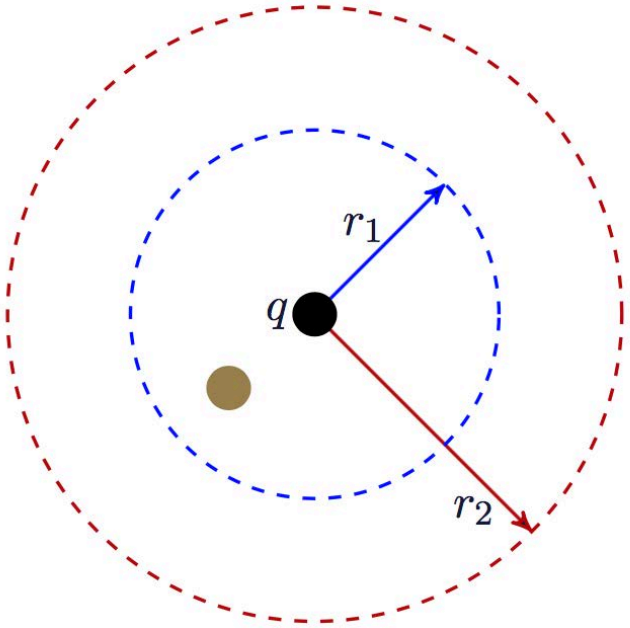
# Was bedeutet "locality-sensitive"

- Eine Familie von Hashfunktionen $H = \{h : S \rightarrow U\}$ heißt $(r_1,r_2,p_1,p_2)$-sensitiv falls die folgenden beiden Bedingungen für beliebige Punkte $q,v \in S$ gelten:

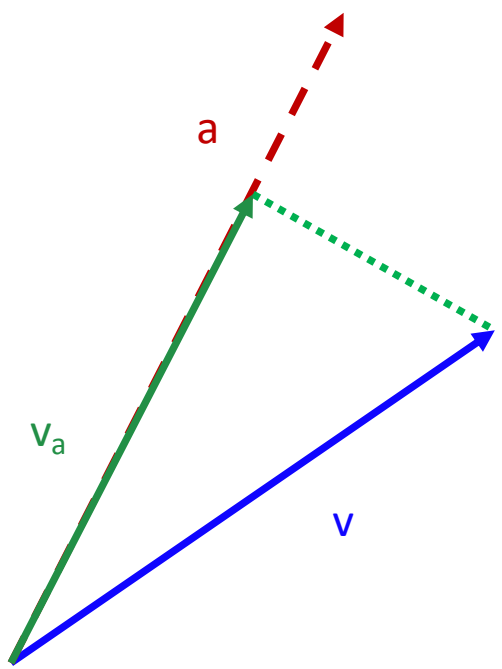
- Falls $dist(q, v) \leq r_1$
  dann $Pr_H(h(q) = h(v)) \geq p_1$
- Falls $dist(q, v) > r_2$
  dann $Pr_H(h(q) = h(v)) \leq p_2$

- Analyse:

  Falls $r_1 < r_2$ und $p_1 > p_2$ gilt:
  Ähnliche Objekte bekommen häufiger den gleichen Hash-Wert als weniger ähnliche

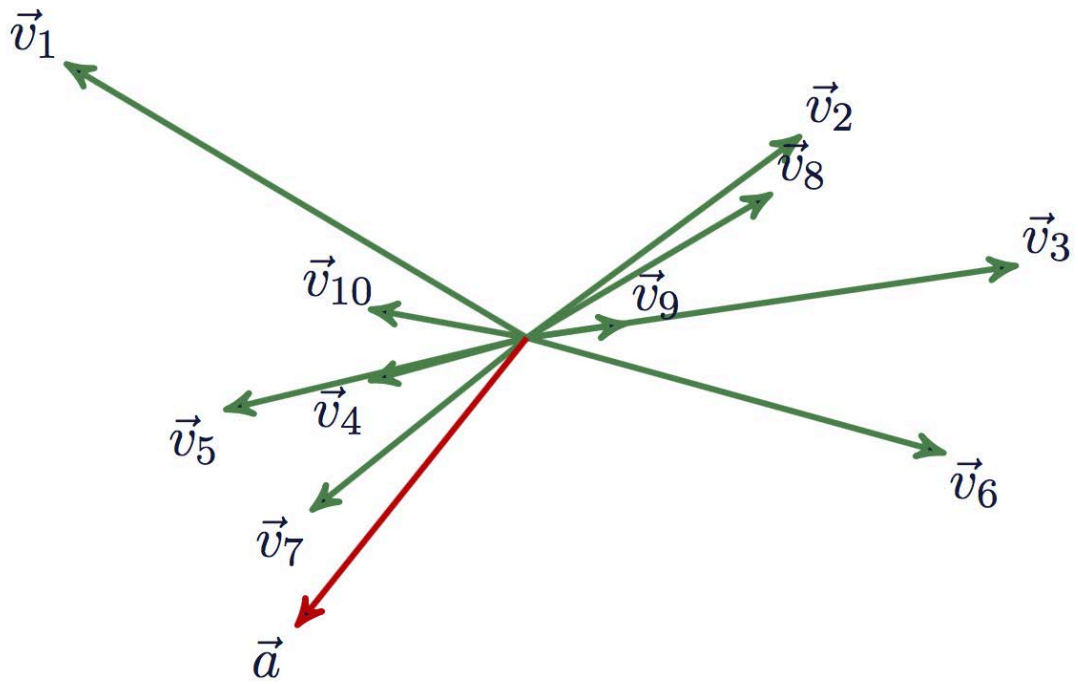


# D-dimensionaler Vektorraum: Idee für Hashfunktion




- Gegeben Vektor v im d-dimensionalen Raum
- Wie kann man v mit weniger als d Werten beschreiben?
- Nehme weiteren Vektor a
- Betrachte Skalarprodukt $a \cdot v$
- Ähnliche Vektoren v haben ähnliche Werte $a \cdot v$
- Also, erste Idee für Hashfunktion: $h(v) := \lfloor a \cdot v \rfloor$

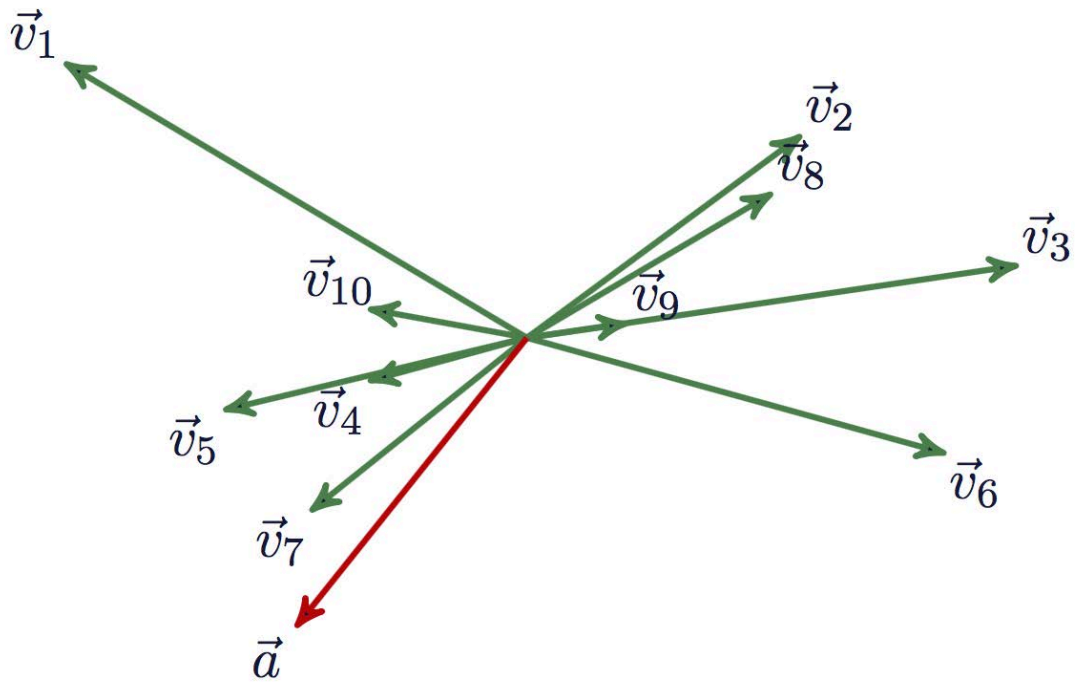
# LSH: Beispiel




$$\vec{v_1} = (-2.1, -1.1)$$
$$\vec{v_2} = (2.4, -1.0)$$
$$\vec{v_3} = (-1.6, -0.5)$$
$$\vec{v_4} = (-3.9, -0.9)$$
$$\vec{v_5} = (-1.0, -0.2)$$
$$\vec{v_6} = (-2.6, 0.0)$$
$$\vec{v_7} = (-0.7, 1.2)$$
$$\vec{v_8} = (1.2, -0.7)$$
$$\vec{v_9} = (-2.6, 0.5)$$
$$\vec{v_{10}} = (-0.8, 1.1)$$

$$\vec{a} = (-1.6, -2)$$

# LSH: Beispiel




Mit Hashfunktion

$$h(v) = \lfloor \vec{a} \cdot \vec{v} \rfloor$$

erhalten wir

$$h(\vec{v}_1) = 1$$
$$h(\vec{v}_2) = -6$$
$$h(\vec{v}_3) = -7$$
$$h(\vec{v}_4) = 2$$
$$h(\vec{v}_5) = 4$$
$$h(\vec{v}_6) = -4$$
$$h(\vec{v}_7) = 4$$
$$h(\vec{v}_8) = -5$$
$$h(\vec{v}_9) = -2$$
$$h(\vec{v}_{10}) = 1$$

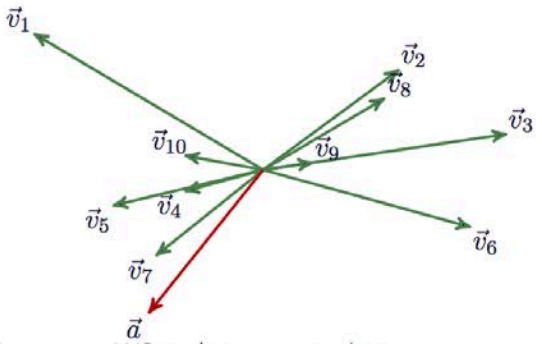
Wir interpretieren die Hashwerte als Label der Buckets, in die wir die Objekte (Vektoren) stecken....

# LSH: Beispiel

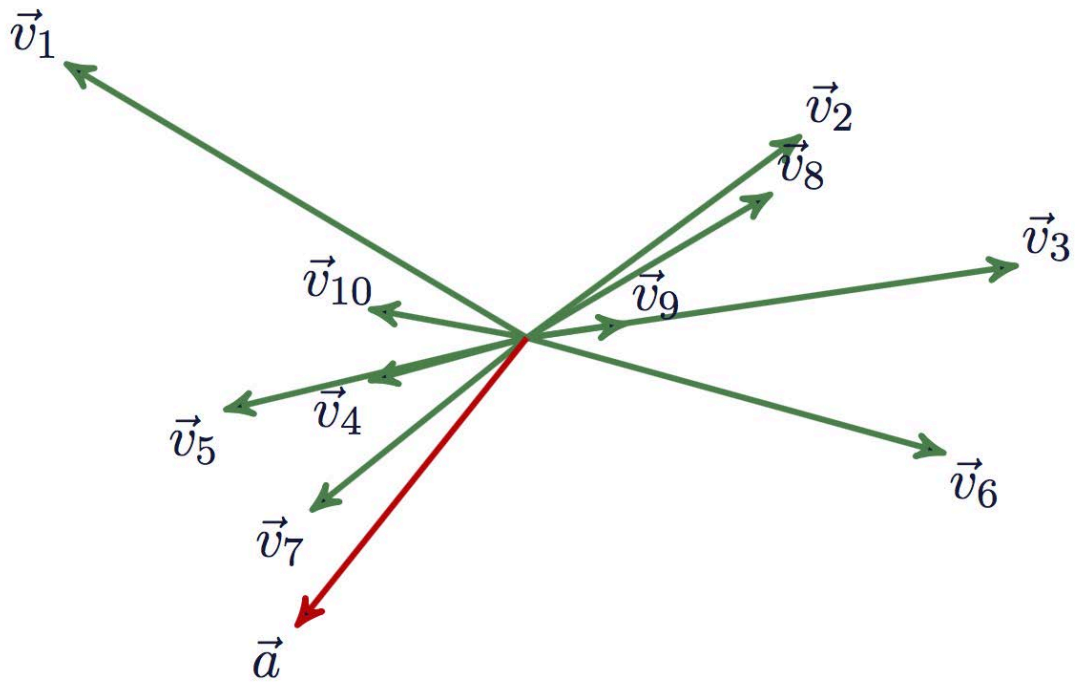
| Bucket -7 | Bucket -6 | Bucket -5 |
|---|---|---|
| $\vec{v}_3$ | $\vec{v}_2$ | $\vec{v}_8$ |
| **Bucket -4** | **Bucket -3** | **Bucket -2** |
| $\vec{v}_6$ | | $\vec{v}_9$ |
| **Bucket -1** | **Bucket 0** | **Bucket 1** |
| | | $\vec{v}_1$ $\vec{v}_{10}$ |
| **Bucket 2** | **Bucket 3** | **Bucket 4** |
| $\vec{v}_4$ | | $\vec{v}_5$ $\vec{v}_7$ |
| **Bucket 5** | **Bucket 6** | **Bucket 7** |
| | | |

**Kaum Kollisionen, viele leere Buckets. Was können wir tun?**

$$h(\vec{v}_1) = 1$$
$$h(\vec{v}_2) = -6$$
$$h(\vec{v}_3) = -7$$
$$h(\vec{v}_4) = 2$$
$$h(\vec{v}_5) = 4$$
$$h(\vec{v}_6) = -4$$
$$h(\vec{v}_7) = 4$$
$$h(\vec{v}_8) = -5$$
$$h(\vec{v}_9) = -2$$
$$h(\vec{v}_{10}) = 1$$



# LSH: Beispiel




Betrachte nun Hashfunktion

$$h(v) = \lfloor \frac{\vec{a} \cdot \vec{v}}{2} \rfloor$$

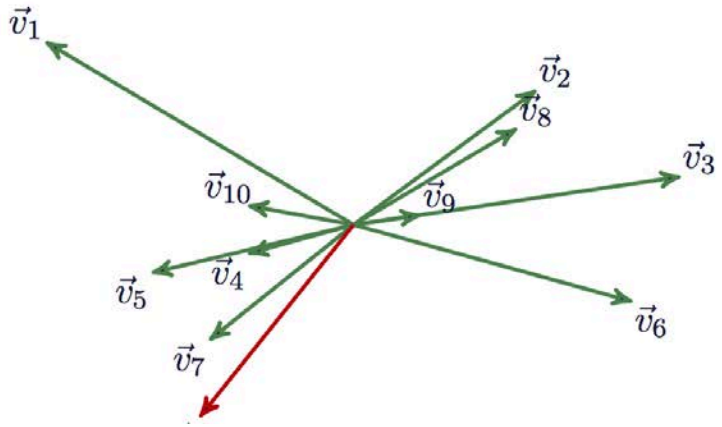
$$h(\vec{v}_1) = 0$$
$$h(\vec{v}_2) = -3$$
$$h(\vec{v}_3) = -4$$
$$h(\vec{v}_4) = 1$$
$$h(\vec{v}_5) = 2$$
$$h(\vec{v}_6) = -2$$
$$h(\vec{v}_7) = 2$$
$$h(\vec{v}_8) = -3$$
$$h(\vec{v}_9) = -1$$
$$h(\vec{v}_{10}) = 0$$

# LSH: Beispiel

| Bucket -4 | Bucket -3 | Bucket -2 |
|---|---|---|
| $\vec{v}_3$ | $\vec{v}_2$<br>$\vec{v}_8$ | $\vec{v}_6$ |
| **Bucket -1** | **Bucket 0** | **Bucket 1** |
| $\vec{v}_9$ | $\vec{v}_1$<br>$\vec{v}_{10}$ | $\vec{v}_4$ |
| **Bucket 2** | **Bucket 3** | **Bucket 4** |
| $\vec{v}_5$<br>$\vec{v}_7$ | | |

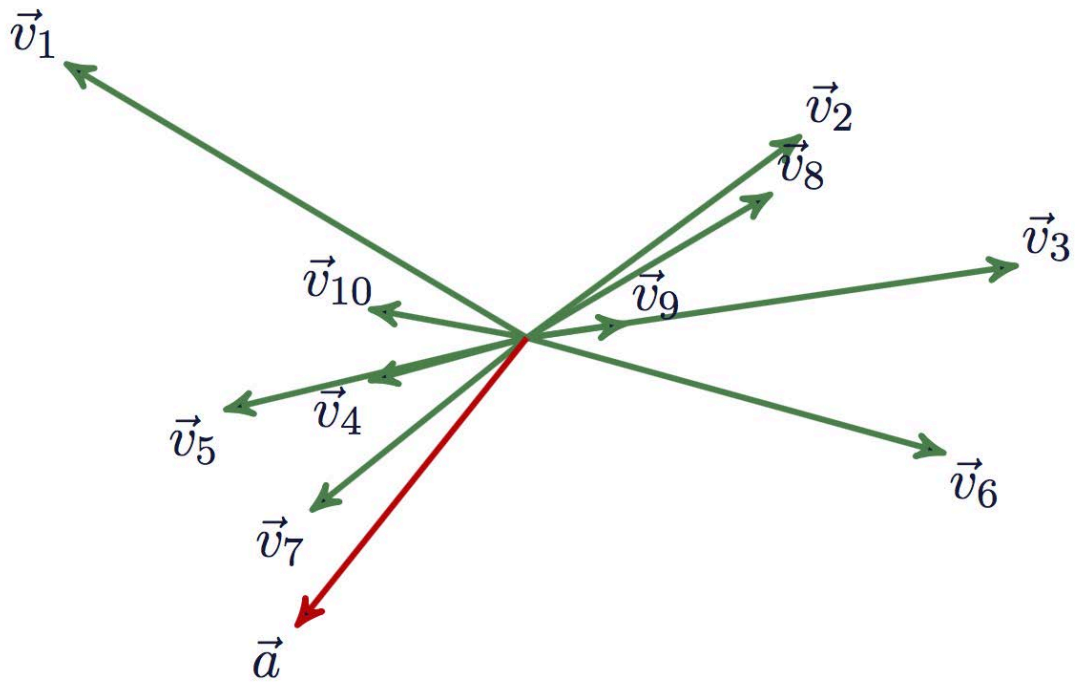
Hashwerte:

$h(\vec{v}_1) = 0$
$h(\vec{v}_2) = -3$
$h(\vec{v}_3) = -4$
$h(\vec{v}_4) = 1$
$h(\vec{v}_5) = 2$
$h(\vec{v}_6) = -2$
$h(\vec{v}_7) = 2$
$h(\vec{v}_8) = -3$
$h(\vec{v}_9) = -1$
$h(\vec{v}_{10}) = 0$




**Wir sehen den Effekt: Weniger Buckets, mehr Kollisionen.**

# LSH: Beispiel




Mit Hashfunktion

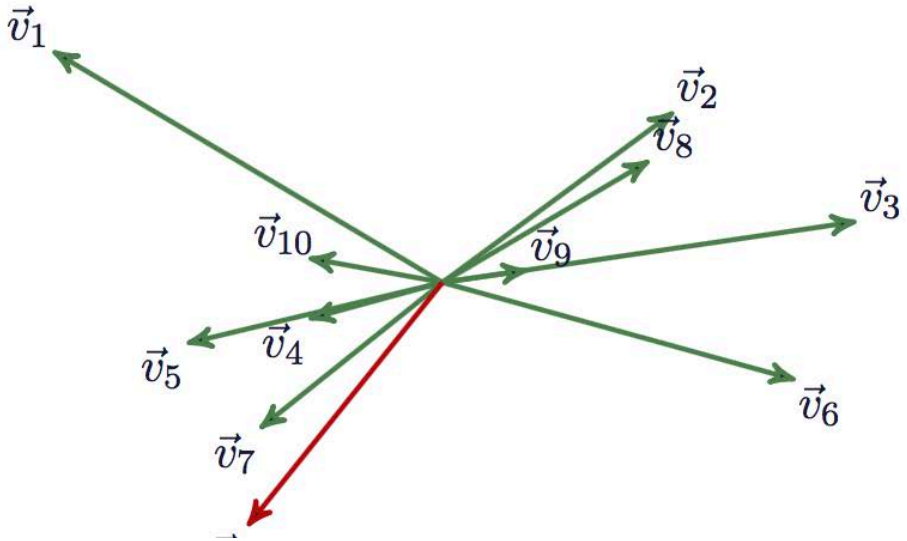
$$h(v) = \lfloor \frac{\vec{a} \cdot \vec{v}}{4} \rfloor$$

$$h(\vec{v}_1) = 0$$
$$h(\vec{v}_2) = -2$$
$$h(\vec{v}_3) = -2$$
$$h(\vec{v}_4) = 0$$
$$h(\vec{v}_5) = 1$$
$$h(\vec{v}_6) = -1$$
$$h(\vec{v}_7) = 1$$
$$h(\vec{v}_8) = -2$$
$$h(\vec{v}_9) = -1$$
$$h(\vec{v}_{10}) = 0$$

# LSH: Beispiel

| Bucket -2 | Bucket -1 | Bucket 0 |
|---|---|---|
| $\vec{v}_2$<br>$\vec{v}_3$<br>$\vec{v}_8$ | $\vec{v}_6$<br>$\vec{v}_9$ | $\vec{v}_1$<br>$\vec{v}_4$<br>$\vec{v}_{10}$ |

| Bucket 1 | Bucket 2 |
|---|---|
| $\vec{v}_5$<br>$\vec{v}_7$ | |




Hashwerte:

$h(\vec{v}_1) = 0$
$h(\vec{v}_2) = -2$
$h(\vec{v}_3) = -2$
$h(\vec{v}_4) = 0$
$h(\vec{v}_5) = 1$
$h(\vec{v}_6) = -1$
$h(\vec{v}_7) = 1$
$h(\vec{v}_8) = -2$
$h(\vec{v}_9) = -1$
$h(\vec{v}_{10}) = 0$

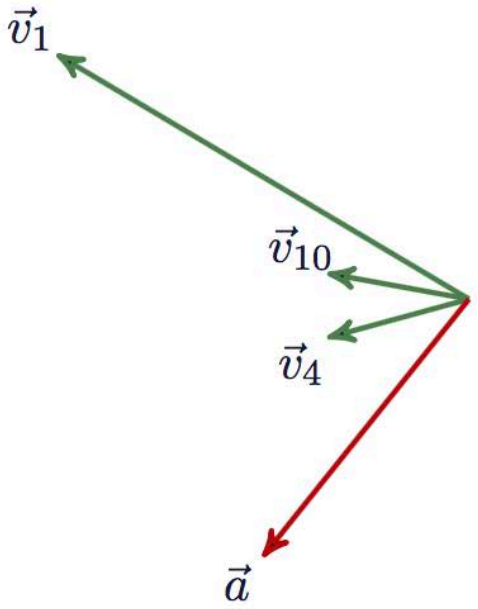
**Anfrage:** Finde Nachbarn von $\vec{v}_8$.

**Vorgehensweise:**
Schaue in Bucket mit Label $-2$ $(= h(\vec{v}_8))$.
Finde zwei Objekte.
Evaluiere diese.

# LSH: Beispiel

**Beobachtung:** $\vec{v}_{10}$, $\vec{v}_4$ und $\vec{v}_1$ fallen in einen Bucket. Ok, $\vec{v}_1$ und $\vec{v}_4$ sind in der Tat ähnlich, aber $\vec{v}_{10}$ passt nicht gut.

- **Wo ist das Problem?**
  Vektor $\vec{a}$ ist nicht in der Lage, gut zwischen diesen drei Vektoren zu unterscheiden.




| Bucket 0 |
| --- |
| $\vec{v}_1$ |
| $\vec{v}_4$ |
| $\vec{v}_{10}$ |

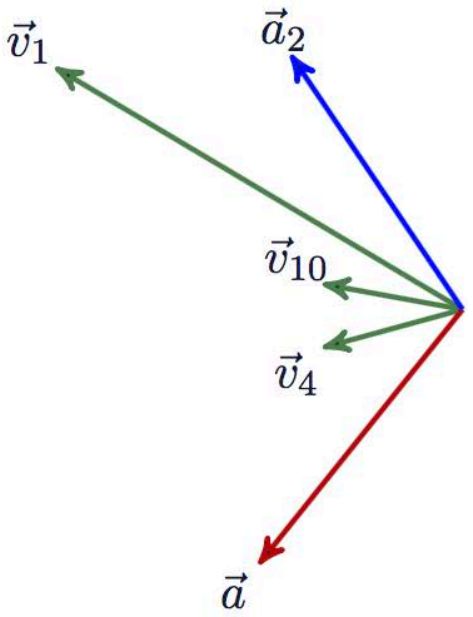
# LSH: Beispiel

**Beobachtung:** $\vec{v}_{10}$, $\vec{v}_4$ und $\vec{v}_1$ fallen in einen Bucket. Ok, $\vec{v}_1$ und $\vec{v}_4$ sind in der Tat ähnlich, aber $\vec{v}_{10}$ passt nicht gut.

- **Wo ist das Problem?**
  Vektor $\vec{a}$ ist nicht in der Lage,gut zwischen diesen drei Vektoren zu unterscheiden.
- **Lösung:** Wir nehmen einen zweiten Vektor $\vec{a}_2$ hinzu.

$$h_{a_2}(\vec{v}) := \lfloor \frac{\vec{a}_2 \cdot \vec{v}}{4} \rfloor$$

Mit $h_{a_2}(\vec{v}_{10}) = 0$, $h_{a_2}(\vec{v}_4) = 0$, $h_{a_2}(\vec{v}_1) = 2$



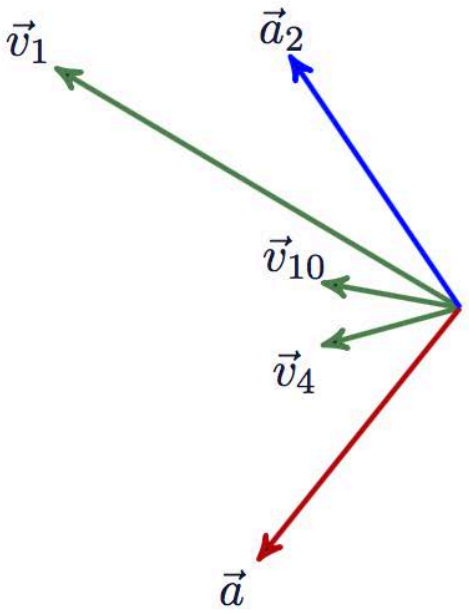
# LSH: Beispiel

**Beobachtung:** $\vec{v}_{10}$, $\vec{v}_4$ und $\vec{v}_1$ fallen in einen Bucket. Ok, $\vec{v}_{10}$und $\vec{v}_4$ sind in der Tat ähnlich, aber $\vec{v}_1$ passt nicht gut.

- **Wo ist das Problem?**
  Vektor $\vec{a}$ ist nicht in der Lage,gut zwischen diesen drei Vektoren zu unterscheiden.
- **Lösung:** Wir nehmen einen zweiten Vektor $\vec{a}_2$ hinzu.

$$h_{a_2}(\vec{v}) := \lfloor \frac{\vec{a}_2 \cdot \vec{v}}{4} \rfloor$$

  Mit $h_{a_2}(\vec{v}_{10}) = 0$, $h_{a_2}(\vec{v}_4) = 0$, $h_{a_2}(\vec{v}_1) = 2$
- Wie sehen dann die Labels der Buckets aus? Konkatenation der einzelnen Hashwerte.
- Wir haben also die "Genauigkeit" der Buckets erhöht.
- Was ist der potentielle Nachteil?




| **Bucket (0,0)** |
| --- |
| $\vec{v}_4$<br>$\vec{v}_{10}$ |

...

# Eine bekannte LSH-Variante

Für jeden $d$-dimensionalen Punkt $\vec{v}$ betrachten wir $k$ unabhängige Hashfunktionen der Form:

$$h_{\vec{a},B}(\vec{v}) = \left\lfloor \frac{\vec{a} \cdot \vec{v} + B}{W} \right\rfloor$$

$\vec{a}$: $d$-dimensionaler Vektor, zufällig anhand Wahrscheinlichkeitsverteilung ausgewählt.
$W \in \mathbb{R}$, und $B \in [0,W]$. $\vec{v}$ wird auf $\vec{a}$ "projiziert" (Skalarprodukt).

**"Beschriftung" der LSH Buckets**

Mit $k$ Hashfunktionen ist die Beschriftung des Buckets ein Integer-Vektor der Länge $k$:

$$g(\vec{v}) = (h_{\vec{a_1},B_1}(\vec{v}), ..., h_{\vec{a_k},B_k}(\vec{v}))$$

Welchen Einfluss hat $k$ auf die Suche?



M. Datar et al.: Locality-sensitive hashing scheme based on p-stable distributions. Symposium on Computational Geometry, **2004**.

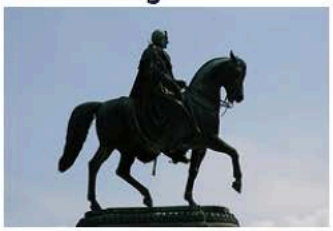
# Objekte den Hash-Buckets zuweisen

## LSH Bucket "Labels"

Mit $k$ Hashfunktionen ist ein Label ein Integer-Vektor der Länge $k$:

$$g(\vec{v}) = (h_{\vec{a_1},B_1}(\vec{v}), ..., h_{\vec{a_k},B_k}(\vec{v}))$$

Objekt:



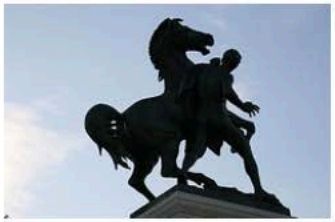
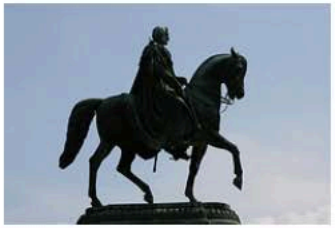
Anwendung von 4 Hashfunktionen:

$$h_1(...) = 0$$
$$h_2(...) = 1$$
$$h_3(...) = 1$$
$$h_4(...) = 1$$

Ergibt das Label: g(...)=(0,1,1,1)

Mehrdimensionale Hashtabelle

# Suche nach ähnlichen Objekten: Beispiel

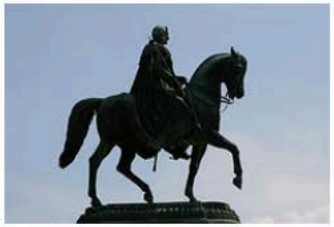
| | | Bucket-Label |
|---|---|---|
| Anfrage: |  | 0,1,0,1 |
| Potentieller Treffer: |  | 0,1,1,1 |

Nicht gefunden!

- Um Trefferwahrscheinlichkeit zu erhöhen, werden Objekte nicht nur in einen Bucket gesteckt anhand von $g$ indexiert, sondern anhand von verschiedenen $g$ in mehrere Buckets in verschiedenen Hashtabellen.

# Mehrere Hashtabellen

Benutze mehrere mehrdimensionale Hashtabellen,
um die Wahrscheinlichkeit von Kollisionen zu vergrößern

| | Hashtabelle 1 | Hashtabelle 2 |
|---|---|---|
|  | 0,1,0,1 | **1,0,0,1** |
|  | 0,1,1,1 | **1,0,0,1** |
| Ergebnis: | Kein Treffer | Treffer! |

# Anfrageverarbeitung: Suche der K nächsten Nachbarn

**Gegeben ein Anfrage-Punkt $q$**

- Berechne Bucket-Labels $g_i(q)$ für jede Hashtabelle $i$.
- Hole Objekte aus diesen Buckets
- Berechne echte Distanz und ordne Objekte entsprechend

**Trotzdem:** LSH ist eine approximative Technik

- Keine harte Garantie, dass alle Treffer (und nur diese) gefunden werden ...
- ... das ist oftmals aber akzeptabel.
- Es gibt theoretische Ansätze die Ergebnisgüte voraus zu sagen

# Beobachtungen

## Tuning

- Tradeoff zwischen Größe der Hash-Buckets und der Effektivität
- Mehrere Hashtabellen: Höhere Trefferwahrscheinlichkeit aber größerer Platzverbrauch
- Achtung: auch hier gilt wieder: Ab einem bestimmen Punkt kann ein Full-Scan günstiger sein (und dieser ist sogar noch exakt!)

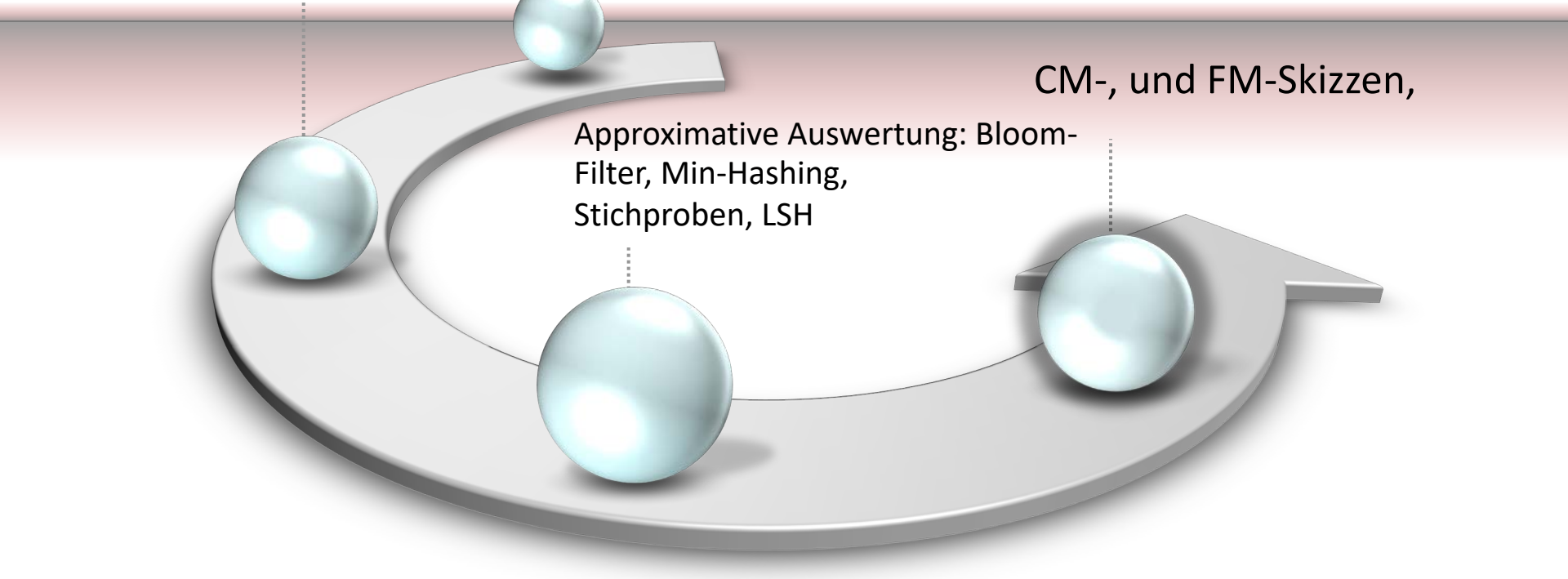
## Erweiterungen

- Schaue in mehrere Hash-Buckets *per* Hashtabelle (aka. multi-probe LSH)
- Auch: verteilte Implementierungen von LSH (z.B. in Peer-to-Peer-Systemen) oder in MapReduce

# Übersicht

Intervall-Skyline, inkrementelle Auswertungstechniken Skizzen

Stromkonzept, relationale Anfragesprachen, exakte Auswertung, QoS-Angaben

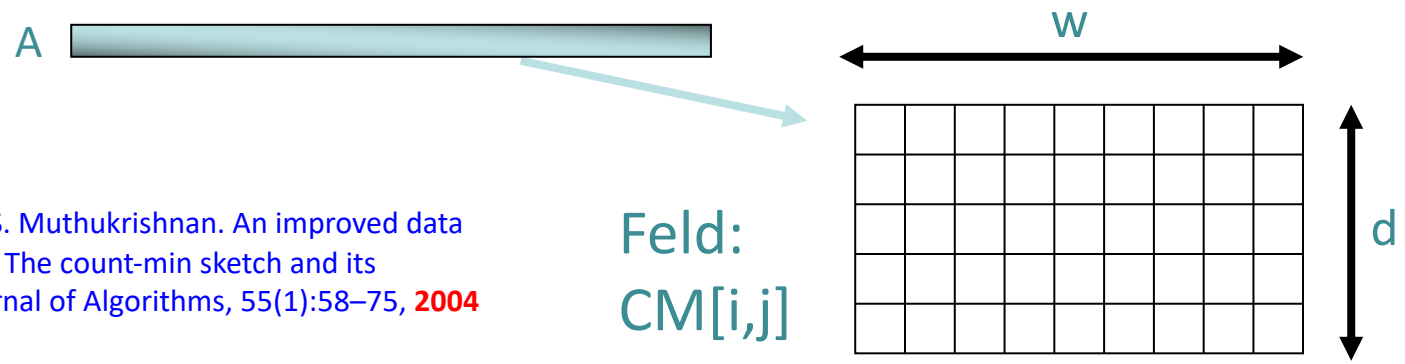


CM-, und FM-Skizzen,

Approximative Auswertung: Bloom-Filter, Min-Hashing, Stichproben, LSH

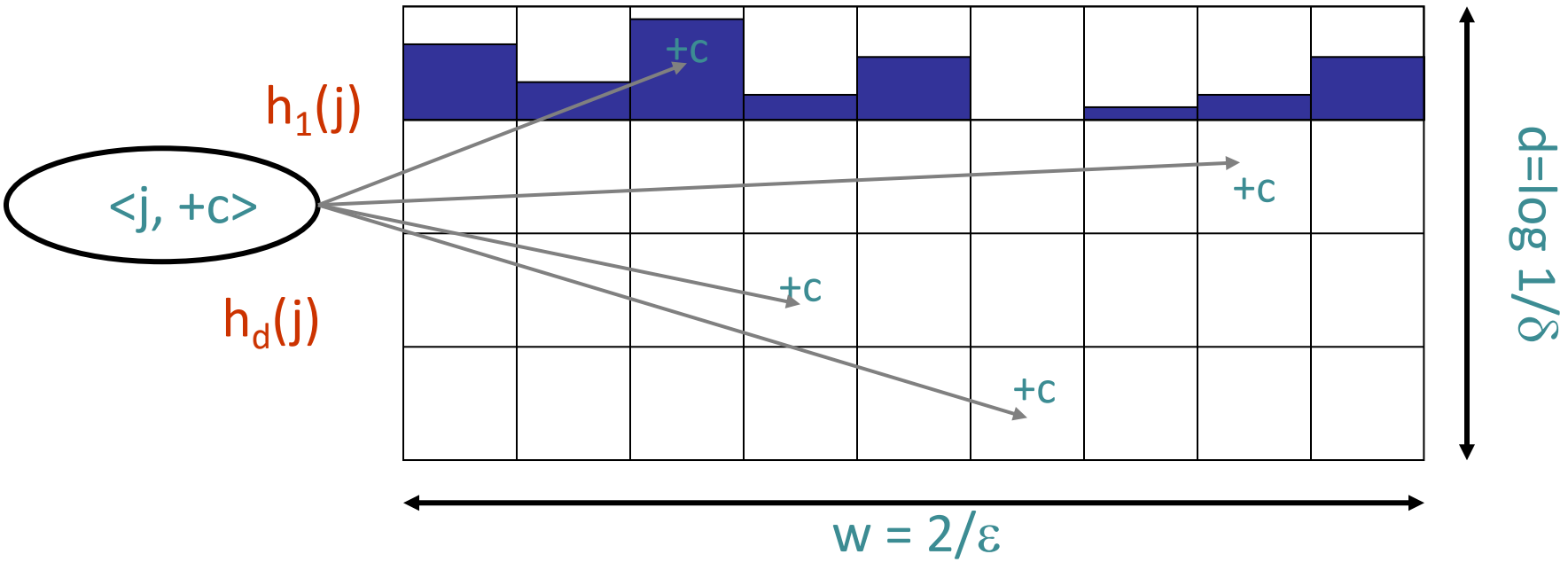
# Count-Min-Skizzen (CM-Skizzen)

- Einfache Synopse (Skizze), Basis für viele Strom-Untersuchungsaufgaben (stream mining tasks)
  - hauptsächlich für "Anzahl Elemente pro Typ" (item frequencies)
  - für Zählerakkumulierung und Drehkreuzmodell
  - Stromrepräsentation als Multimenge (Bag-of-Words-Modell)
- Vektor A der Dimension N: Zähler für jeden Typ
- Skizze als kleines Feld CM der Größe $w \times d$ dargestellt
- Verwende d Hashfunktionen zur Abbildung der A-Elemente auf Intervall [1..w]




G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. Journal of Algorithms, 55(1):58–75, **2004**

# CM-Skizzen




- Jedes A[] wird auf CM-Eintrag abgebildet
  - h()'s *paarweise unabhängig*
- Schätze A[j] über den Ausdruck $\min_k \{ CM[k,h_k(j)] \}$
- Parallelisierung: Verschmelzung eintragsweise durch Summierung verschiedener CM-Matrizen möglich

# CM-Skizzen – (ε, δ)-Approximationen

- CM-Approximierungsfehler bei Punktanfragen kleiner als $\varepsilon\|A\|_1$ mit Platzbedarf $O(1/\varepsilon \log 1/\delta)$ *[Cormode, Muthukrishnan'04]*
  - Wahrscheinlichkeit eines größeren Fehlers kleiner als $1-\delta$
  - Ähnliche Garantien für Bereichsanfragen, Quantile, Verbundgrößen (join size), …
- Hinweise
  - Zähler überschätzen durch Hash-Kollisionen
    - Wie begrenzbar in jedem Feld?
  - Nutze Unabhängigkeit über Zeilen, um Konfidenz für min{}-Schätzung zu erhöhen

G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. LATIN 2004, J. Algorithm 55(1): 58-75, **2004**

# CM-Skizzen – Analyse

Schätze $A'[j] = \min_k \{ CM[k,h_k(j)] \}$

# Tafelbild der Abschätzung A‘[j] für A[j]

$$A'[j] = \min_k \{ CM[k, h_k(j)] \}$$

Analyse der k-ten Zeile

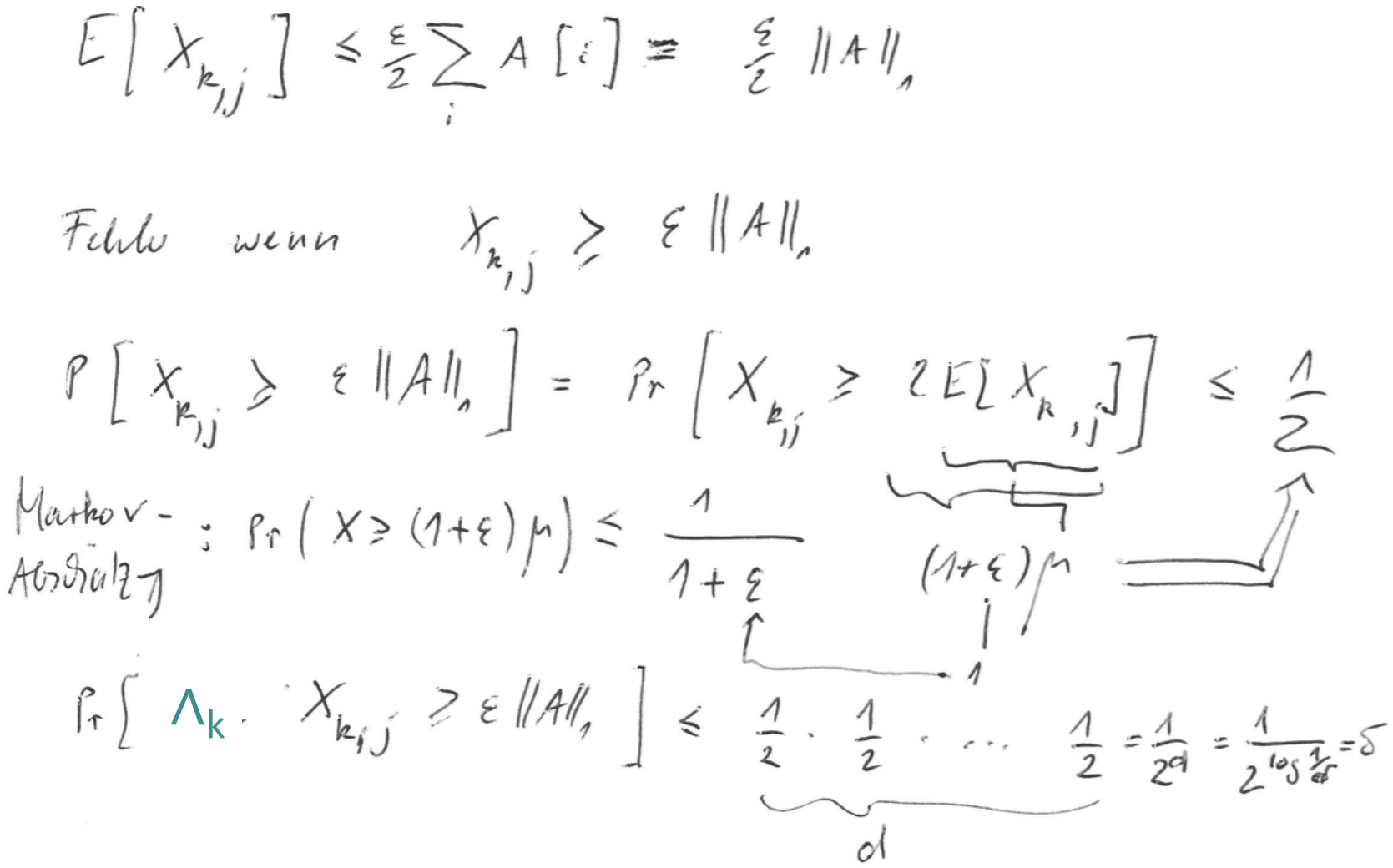
$$CM[k, h_k(j)] = A[j] + X_{k,j}$$

$$X_{k,j} := \sum_{h_k(i) = h_k(j)} A[i]$$

$$E[X_{k,j}] = \sum_i A[i] \cdot Pr(h_k(i) = h_k(j))$$

$$Pr(h_k(i) = h_k(j)) \leq \frac{1}{w} = \frac{\varepsilon}{2} \qquad (h \text{ paarw. unabh.})$$

# Tafelbild der Abschätzung A‘[j] für A[j] (2)



# Stromdatenverarbeitung

# Approximation

# Zählen der Anzahl der verschiedenen Elemente

- **Problem:**
  - Datenstrom enthält Elemente aus Grundmenge der Größe N
  - Gesucht ist Anzahl der verschiedenen Elemente in einem gesamten bisherigen Strom zu einem Zeitpunkt, zu dem Fenster ausgewertet wird

- **Naiver Ansatz:**
  Speichere gesehene Elemente in Hashtabelle als Synopse

# Anzahl **verschiedener** Werte abschätzen

- Aufgabe: Finde Anzahl der verschiedenen Werte in einem Strom von Werten aus [1,…,N] (`count distinct`)
  - Statistik: Anzahl von Arten oder Klassen in Population
  - Datenbanken: Selektivitätsschätzung in Anfrageoptimierung
  - Netzwerkbeobachtung: IP-Adressen mit unterschiedlichem Ziel, Quelle/Ziel-Paare, angeforderte URLs usw.

- Beispiel (N=64)

Datenstrom: 3 2 5 3 2 1 7 5 1 2 3 7

*Anzahl der verschiedene Werte: 5*

- Naiver Ansatz: Hash-Tabelle für alle gesehenen Elemente
- Schwierig auch für CM (gedacht für Multimengen)

# Flajolet-Martin-Ansatz

- Wähle Hashfunktion ***h*** zur Abbildung von ***N*** Elementen auf mindestens $\log_2 N$ Bits

- Für jedes Stromelement ***a***, sei ***r(a)*** die Anzahl der Nullen am Ende von ***h(a)***
  - **r(a)** = Position der ersten 1 von rechts
    - Z.B. sei ***h(a) = 12***, dann ist ***12*** gleich ***1100*** binär, also ***r(a) = 2***
- Speichere ***R = maximales r(a) bisher***
  - $\mathbf{R = max_a\ r(a)}$, über alle Stromelemente ***a*** bisher

- **Geschätzte Anzahl unterschiedlicher Elemente: $\approx 2^R$**

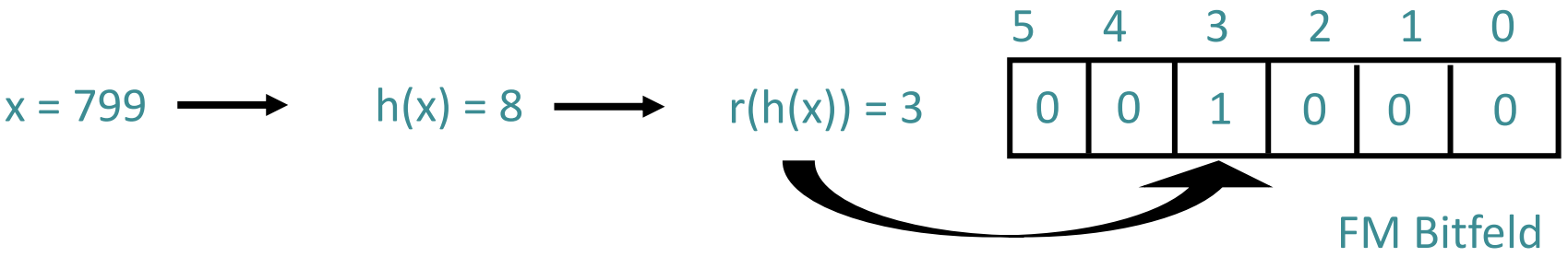
P. Flajolet and G. N. Martin. Probabilistic counting. In IEEE Conference on Foundations of Computer Science, pages 76–82, 1983. Journal version in Journal of Computer and System Sciences, 31:182–209, **1985**.

# Warum funktioniert das? Intuition

- **h(a)** bildet **a** mit **gleicher Wahrscheinlichkeit** auf jeden von **N** möglichen Werten ab
- Dann ist **h(a)** eine Sequenz von $\log_2 N$ Bits, wobei $2^{-r}$ der Anteil aller **a**s mit **r** Nullen am Ende ist
  - Ca. 50% der **a**s hashen auf ***0**
  - Ca. 25% der **a**s hashen auf **00**
  - Wenn also **r(a)=2** Nullen hinten stehen (i.e., Hash ergibt ***100**), dann haben wir wahrscheinlich **ca. 4** verschiedene Elemente gesehen
- **Also braucht es ein Hash auf $2^r$ Elementen bevor man eines mit 0-Suffix der Länge r sieht**

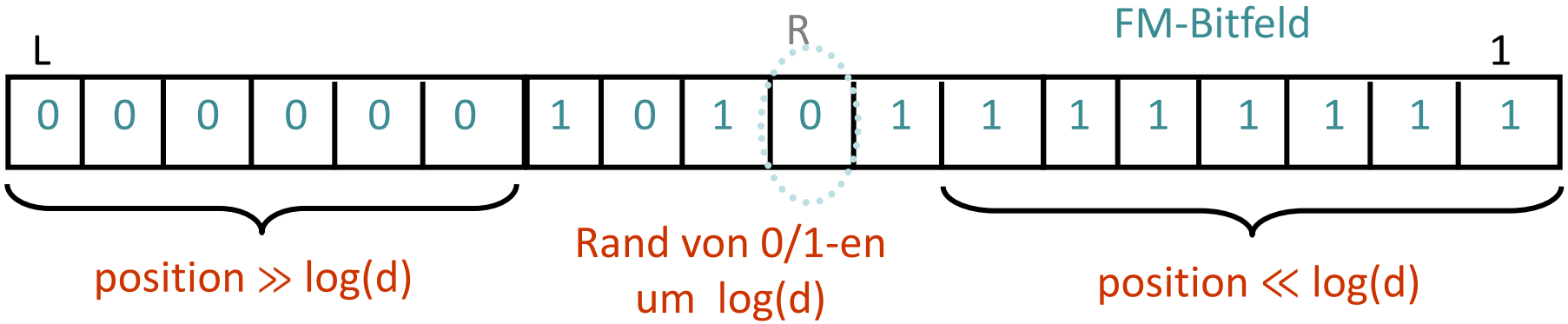
# FM-Skizzen [Flajolet, Martin 85]

- Verwende Hashfunktion h' zur Abbildung von Eingabeelementen auf i mit Wahrscheinlichkeit $2^{-r}$
  - also Pr[h'(x) = 1] = ½, Pr[h'(x) = 2] = ¼, Pr[h'(x)=3] = 1/8 …
  - Konstruiere h'() aus gleichverteilter Hashfunktion und anschließendendem "Zählen der Nullen am Ende" durch r
- Aufbau FM-Skizze= Bitfeld von L = log N Bits
  - Initialisiere Bitfelder of 0
  - Für jeden neuen Wert x, setze FM[r(x)] = 1



# FM-Skizzen – Analyse

- Bei d verschiedenen Werten, erwarte Abbildung von d/2 Werten nach FM[1], d/4 nach FM[2]…

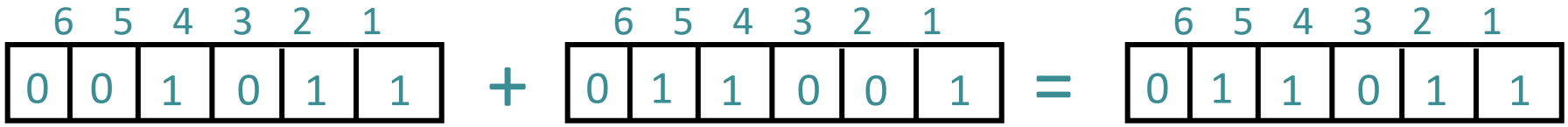



  - Sei R = Position der rechtesten 0 in FM, Indikator für log(d)
  - [FM85] zeigen, dass E[R] = $\log(\phi d)$ , mit $\phi = .7735$
  - Schätzung $d = c2^R$ für Skalierungskonstante $c \approx 1.3$
  - Mittelung mit verschiedenen Hashfunktionen dient zur Verbesserung des Ergebnisses
  - Wieviele Hashfunktionen für best. Anforderung benötigt?

# FM-Skizzen – Eigenschaften

- Mit $O(1/\varepsilon^2 \log 1/\delta)$ Hashfunktionen, $(1 \pm \varepsilon)$ Genauigkeit mit Wahrscheinlichkeit mindestens $1-\delta$
  Ohne Beweis: siehe [Bar-Yossef et al'02], [Ganguly et al.'04]
  - 10 Funktionen ≈ 30% Fehler, 100 Funkt. < 10% Fehler

- Bei Löschung: Verwende Zähler statt Bits
  - +1 für Einfügungen, -1 für Löschungen
- Komposition: komponentenweise OR bzw. +




  - Schätze $|S_1 \cup \cdots \cup S_k|$ = *Kardinalität der Vereinigungsmenge*

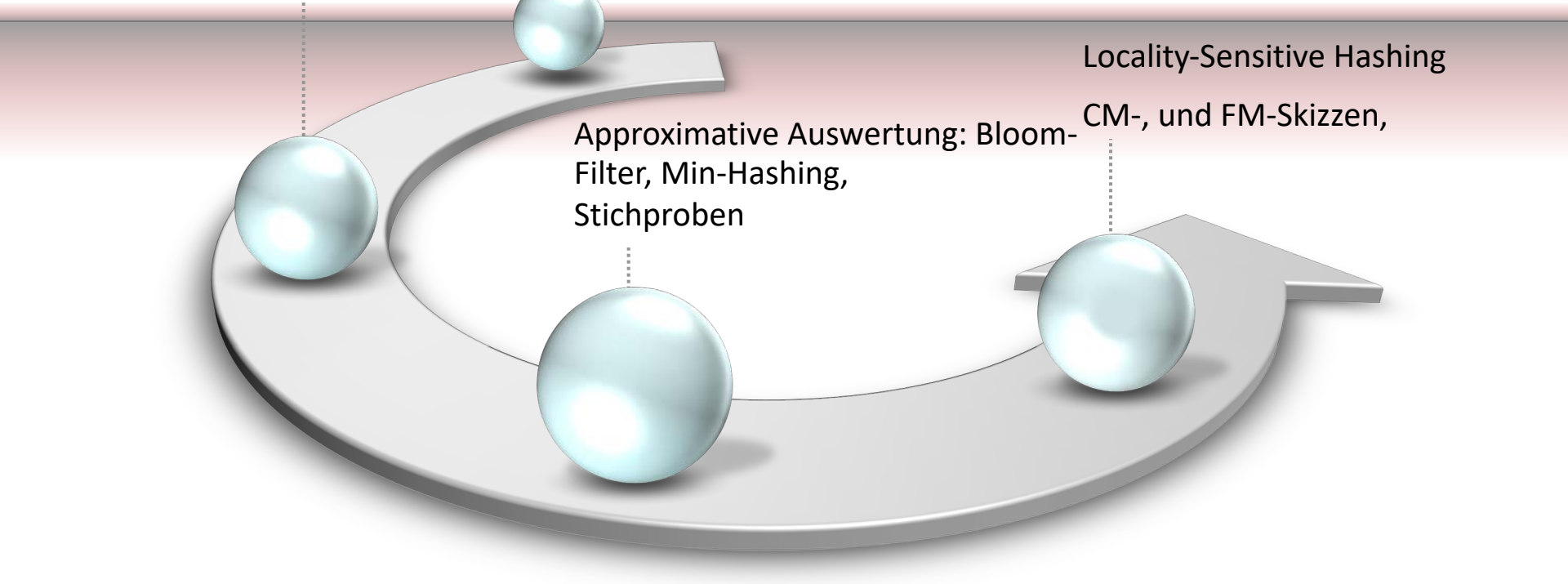
# Stichproben und Skizzierung: Zusammenfassung

- Zentrale Probleme für viele Stromanalyseverfahren
  - Momente/Verbundaggregate, Histogramme, top-k, Meistfrequentierte Elemente, andere Analyseprobleme, …
- Stichproben eher generelle Repräsentation eines Datensatzes
  - Einfache Stichproben (sampling) kaum für Strommodelle geeignet
- Skizzierung eher für speziellen Zweck
  - FM-Skizze für “Anzahl der Typen”,
  - CM-Skizze für “Anzahl Elemente pro Typ” (auch: Verbundgröße bzw. Momentenschätzung …)

# Zusammenfassung

Intervall-Skyline, inkrementelle Auswertungstechniken Skizzen

Stromkonzept, relationale Anfragesprachen, exakte Auswertung, QoS-Angaben



Approximative Auswertung: Bloom-Filter, Min-Hashing, Stichproben

Locality-Sensitive Hashing

CM-, und FM-Skizzen,

# Frequency Estimation with Predictions

- Many frequency estimation algorithms based on count-based sketches
  - Count Min, Count Sketch
  - Algorithms approximate counts for items

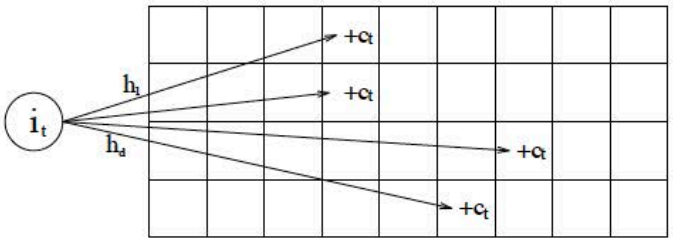



Fig. 1. Each item $i$ is mapped to one cell in each row of the array of counts: when an update of $c_t$ to item $i_t$ arrives, $c_t$ is added to each of these cells

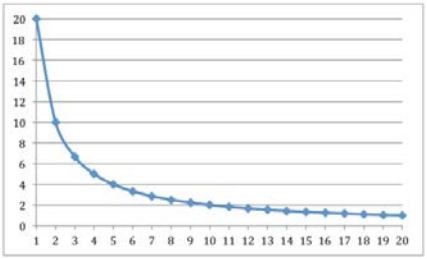
Chen-Yu Hsu, Piotr Indyk, Dina Katabi, Ali Vakilian, Learning-Based Frequency Estimation Algorithms, In Proc. ICLR **2019**.

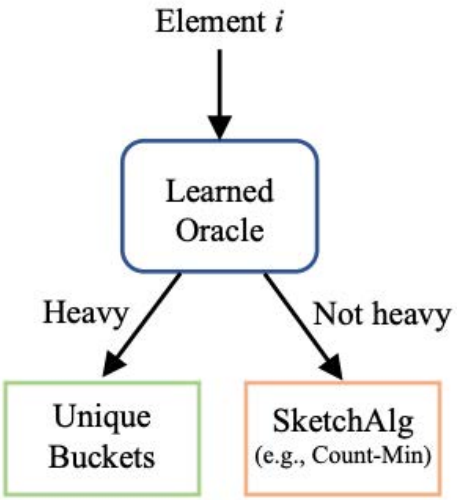
# Streaming Algorithms:

## Frequency Estimation with Predictions

- Problem with sketches:  collisions
  - Low-count items return high counts if they collide with some high-count item in each row.
  - Not frequent, but leads to large error.
- If we could remove high-count items from consideration, better results all around.
  - Do explicit counts for some items, sketch the rest.
- If we have a predictor for high-count items, use the predictor to decide what to remove for explicit counts.

# Frequency Estimation with Predictions



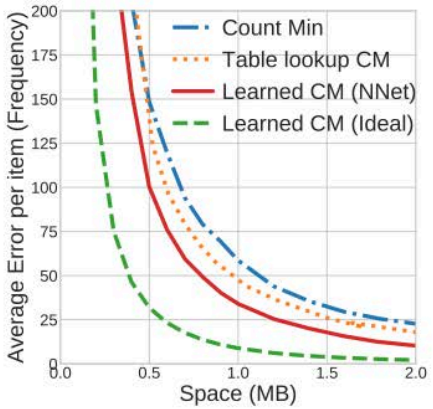

## Theoretical Results for Zipfian Distributions

| | $k = 1$ | $k > 1$ |
|---|---|---|
| **Count-Min (CM)** | $\Theta\left(\frac{\log n}{B}\right)$ [HIKV19] | $\Theta\left(\frac{k \cdot \log(\frac{kn}{B})}{B}\right)$ |
| **Learned Count-Min (L-CM)** | $\Theta\left(\frac{\log^2(\frac{n}{B})}{B \log n}\right)$ [HIKV19] | $\Omega\left(\frac{\log^2(\frac{n}{B})}{B \log n}\right)$ [HIKV19] |

*k* is number of hash functions
*B* is number of buckets

Chen-Yu Hsu, Piotr Indyk, Dina Katabi, Ali Vakilian, Learning-Based Frequency Estimation Algorithms, In Proc. ICLR **2019**.

# Empirical Results on Internet Traffic Data




Chen-Yu Hsu, Piotr Indyk, Dina Katabi, Ali Vakilian, Learning-Based Frequency Estimation Algorithms, In Proc. ICLR **2019**.

# Learned Bloom Filters

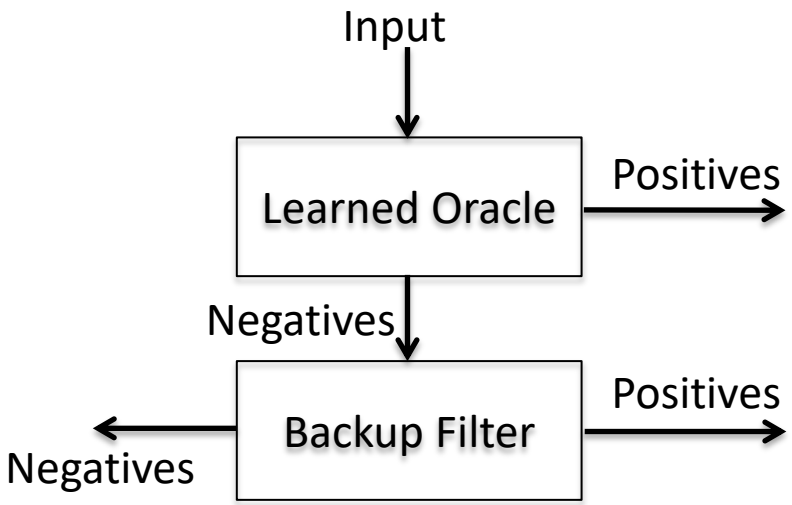
- [Kraska et al. 18] suggests one can do better than standard Bloom filters
  - In a data dependent way
  - Assuming you can "learn" the set from the Bloom filter
- Use machine learning to develop a small-size oracle that provides probability an element is in a set
  - Oracle should (hopefully) give few false positives
  - And you need a backup to catch any false negatives.

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean and Neoklis Polyzotis. The Case for Learned Index Structures. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18), **2018**.

# Learned Bloom Filters: Setup

- Bloom filters as binary classification problem
- Use set of elements as a collection of positive examples.
- Use set of non-elements as a collection of negative examples.
  - Can be chosen non-elements, or random non-elements.
- Derive neural network with sigmoid activation to produce a "probability estimate" that an input is in the set, minimizing log loss.
- Choose a threshold
  - Elements evaluated above the threshold are assumed positive; below the threshold are assumed negative.
- May be false positives, and false negatives.

# Learned Bloom Filter: Improved Setup




Items that the oracle says are likely positives are treated as positives. Causes false positives.

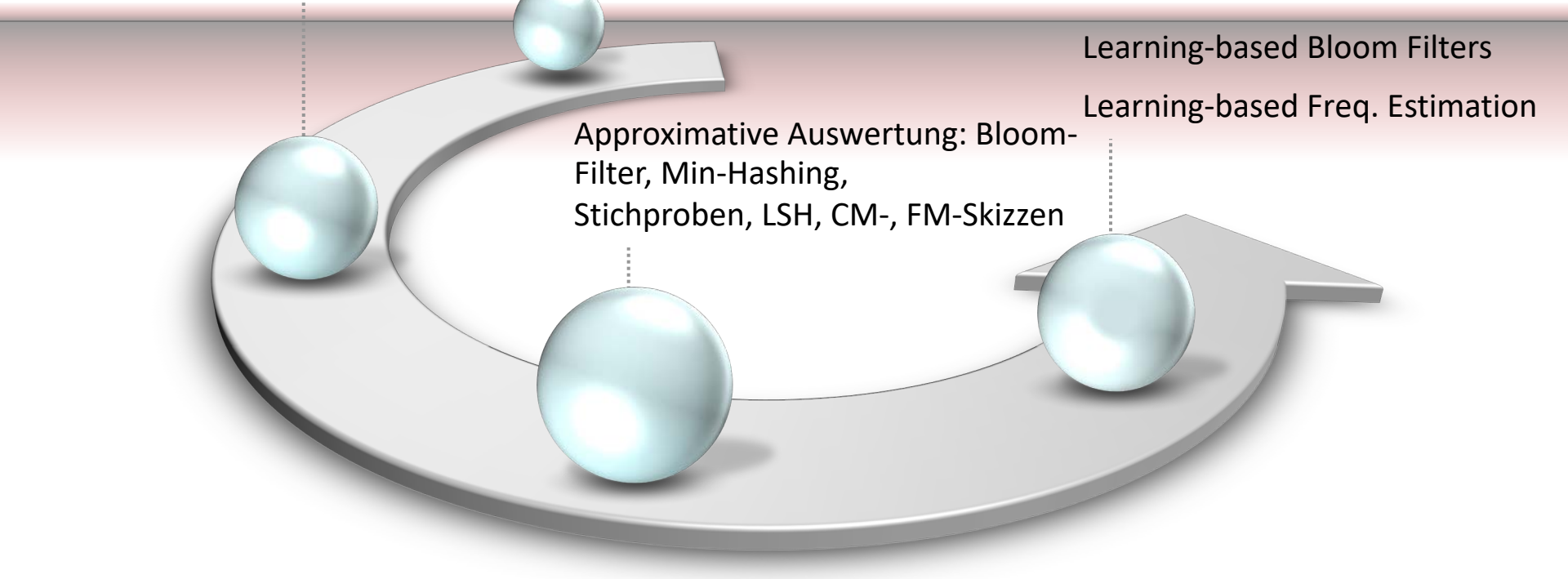
Items that oracles says are negative might include false negatives! So we have a backup Bloom filter to catch false negatives in the set. Causes additional false positives, avoids false negatives.

Michael Mitzenmacher. 2018. A model for learned bloom filters, and optimizing by sandwiching. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18), **2018**.

# Zusammenfassung

Intervall-Skyline, inkrementelle Auswertungstechniken Skizzen

Stromkonzept, relationale Anfragesprachen, exakte Auswertung, QoS-Angaben



Learning-based Bloom Filters

Learning-based Freq. Estimation

Approximative Auswertung: Bloom-Filter, Min-Hashing, Stichproben, LSH, CM-, FM-Skizzen