Most of the Time it Works Every Time

The Mindset Behind Using Probabilistic Data Structures

Techcamp 2024, Hamburg, Germany

Wolfram „Wolle“ Wingerath

June 19, 2024

Slides Available at https://wolle.science

Research Overview: Data Engineering for Data Science

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Baqend

Carl von Ossietzky Universität Oldenburg

OLTP Benchmarking

NoSQL Stores

Stream Processing

Real-Time Databases

Continuous Web Performance Analytics

Ongoing Research: Real-Time Data Science

1. anomaly & bot detection, click path prediction
2. Realtime-as-a-Service, HPC storage optimization
3. data quality & data synthesis, data science for quality of life …

Uni Hamburg — Baqend — Uni Oldenburg

2015

DMC 2014
caching, OLTP benchmarking
(2 workshop papers)

BTW 2015
caching, OLTP benchmarking
(2 papers)

2016

itit 2016
data stream processing survey
(journal article)

CSRD 2016
NoSQL survey
(journal article)

2017

SCDM 2017
real-time databases, OLTP benchmarking
(2 workshop papers)

VLDB 2017
query caching
(industry paper)

BTW 2017
NoSQL & cloud data management
(tutorial)

2018

EDBT 2018
real-time data management
(tutorial)

2019

BTW 2019
NoSQL, data management,
Realtime databases
(tutorial & demo)

Springer Book
Real-Time & Stream Data Management

Nomination: German Future Prize 2020
for a Web without loading times

2020

ICDE 2020
real-time databases, caching
(industry & demo papers)

GI Junior Fellow 2020
Winner

Springer Book
Fast & Scalable Cloud Data Management

Nomination: GI-Dissertationspreis 2019
real-time databases

VLDB 2020
real-time databases
(industry paper)

2021

EDBT 2021
data quality
(application paper)

WWW 2021
Web performance, data management (tutorial)

GI Initiatives
startup platform &
handsfree coding

2022

EDBT 2022
schema transformation
(short paper)

ICWE 2022
delta encoding for Web perf.
(research & demo papers)

VLDB 2022
A/B-testing at scale,
polyglot data management
(industry paper & tutorial)

2023

JWE 2023
delta encoding for Web perf.
(extended research papers)

Mini Lecture Outline

1 Skip Lists: Challenge & Basic Idea

Which problem do skip lists solve and in what ways are they superior to other list variants?

2 Chance, Efficiency & Complexity Analysis

What is the probabilistic element in skip lists, how do they scale, and when should you use them?

3 Trade-Offs in Other Probabilistic Data Structures

What are advantages of other probabilistic data structures like Bloom filters or Count-Min Sketches?

Helpful Basic Knowledge

Data Structures

Linked Lists, Arrays & Array Lists,

Self-Balancing Trees, Hash Maps

Algorithms & Performance Analysis

Binary Search, Tree Traversal, Sorting,

Probability Theory basics

Learning Goal!

Sorted List Applications

Database Sorting & Indexes, Dynamic

Collections, (Streaming) Aggregation

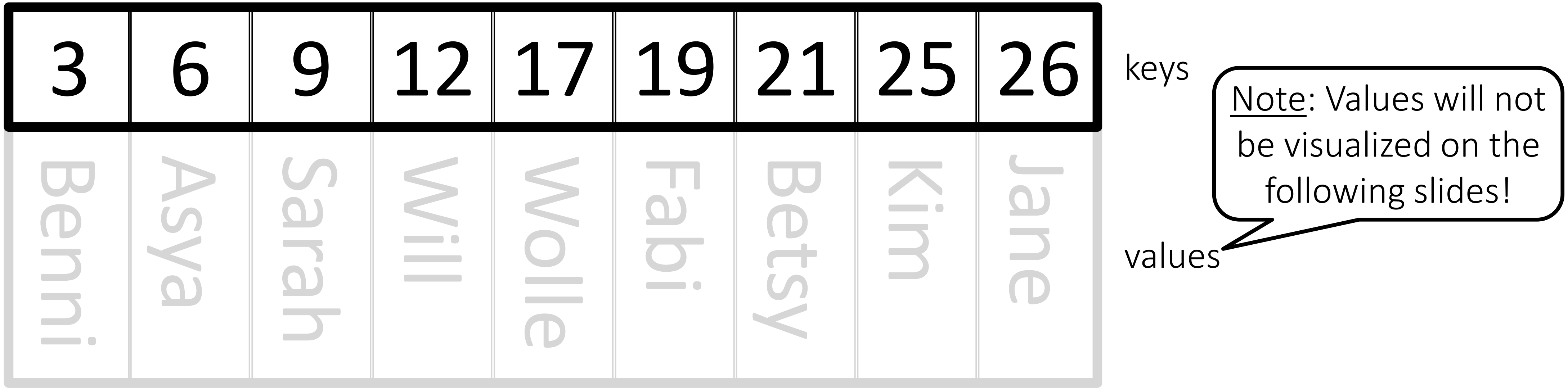
Probabilistic Data Structures

Skip Lists & Coin Flipping, Bloom Filters,

Count-Min Sketch, Trade-Offs & Use Cases

Application Scenario: Working With a Sorted List

- Imagine **sorted list** of key-value pairs, e.g. ...
  - a sorted set in Redis
  - Member list on your Discord server
  - a list of running medians over a large sliding window



Raymond Hettinger. Regaining Lost Knowledge, Deep Thoughts by Raymond Hettinger (2010).

Matt Nowack. Using Rust to Scale Elixir for 11 Million Concurrent Users, Discord Blog (2019).

Redis Ltd. Redis Sorted Sets, Redis Glossary (2023).

Challenge: Maintaining Order

- Why not just use standard list implementations?

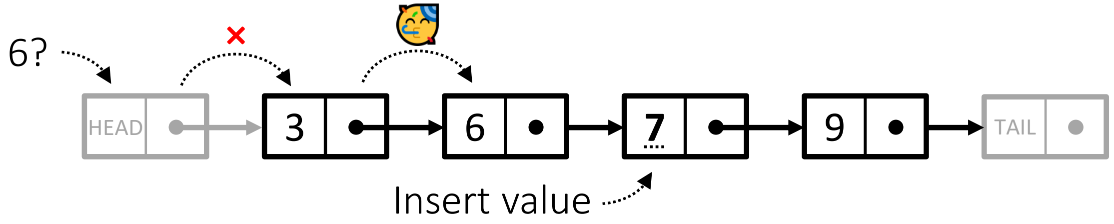
Sorted Linked List:

- Search: $O(n)$
- Update: $O(1)$ (after search)

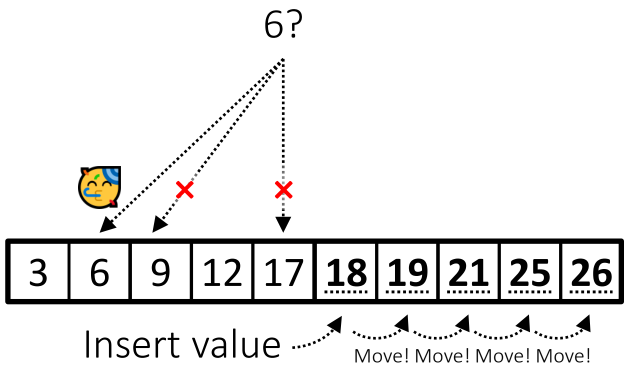
VS.

Sorted Array List:

- Search: $O(log\, n)$
- Update: $O(n)$



+ Fast updates



+ Binary Search

Can‘t we have a list that gives us both?

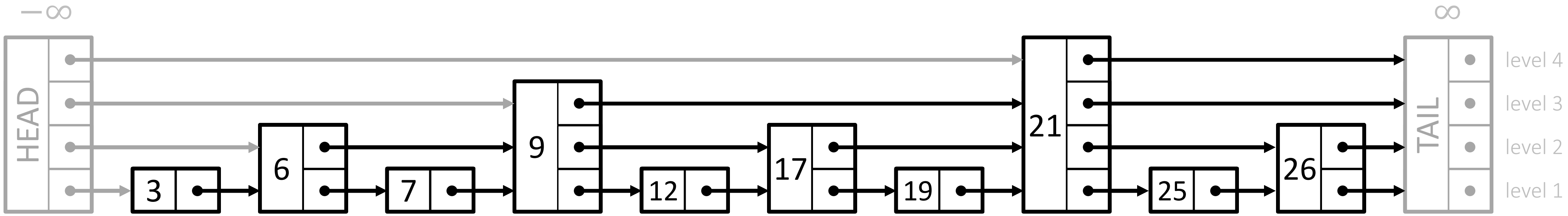
(Yes! Yes, we can!)

Idea for list comparison inspired by:
Kevin Buchin. Skip Lists, YouTube (2021).

Skip List Idea: A Sorted Linked List Tuned For Binary Search

The perfect skip list is a sorted linked list with **shortcuts** for skipping item subsequences during traversal

- *Normal Lane* (level 1): standard sorted linked list where every node is connected to its successor
- **Express Lanes** (levels above): Only half of all nodes are promoted to the next level
  - Level 2: add pointers that connect only every 2nd node
  - Level 3: add pointers that connect only every 4th node
  - ...
  - Level $log\ n$: only 1 node that connects to HEAD and TAIL

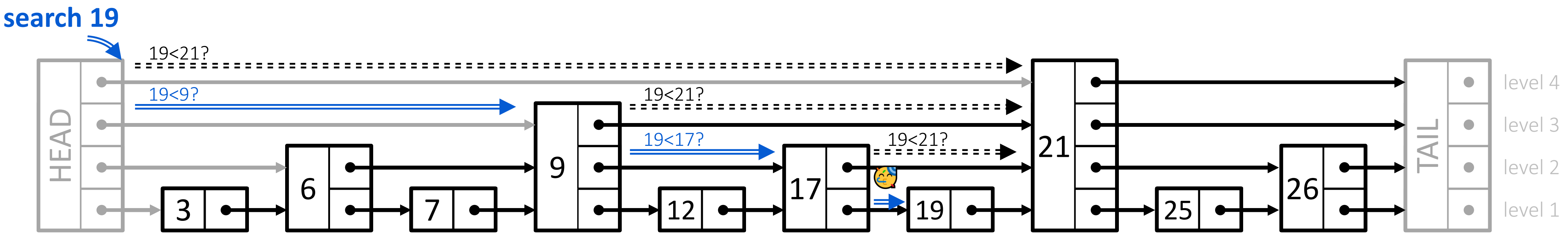


Searching the Perfect Skip List

- Basic search algorithm:
  (1) Start with the fastest express lane (top level)
  (2) Keep advancing until the next step would overshoot, then climb down one level
  (3) Repeat until you either find the target or reach the normal lane and find that it‘s not in the list

🥳 Success! (find the target)

× Failure! (not in the list)



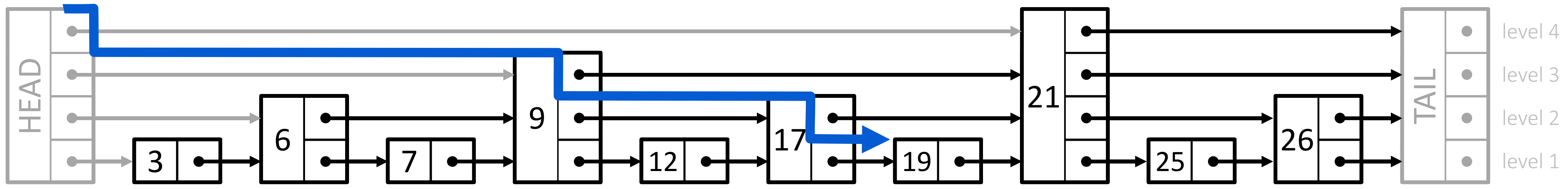
Searching the Perfect Skip List

- Basic search algorithm:
  1. (1) Start with the fastest express lane (top level)
  2. (2) Keep advancing until the next step would overshoot, then climb down one level
  3. (3) Repeat until you either find the target or reach the normal lane and find that it‘s not in the list

🥳 Success! ❌ Failure!

- $\boldsymbol{O(log\, n)}$ **Time Complexity**: Search paths no longer than $2\, log\, n$ nodes
  - There are $log\, n$ levels
  - Search will visit no more than 2 nodes per level!

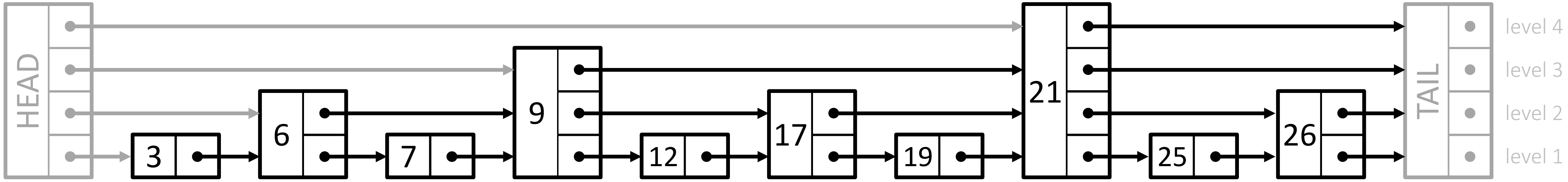
search 19



The Perfect Skip List: Space Efficiency

$\boldsymbol{O(n)}$ **Space Complexity**: The list has no more than $2n$ pointers

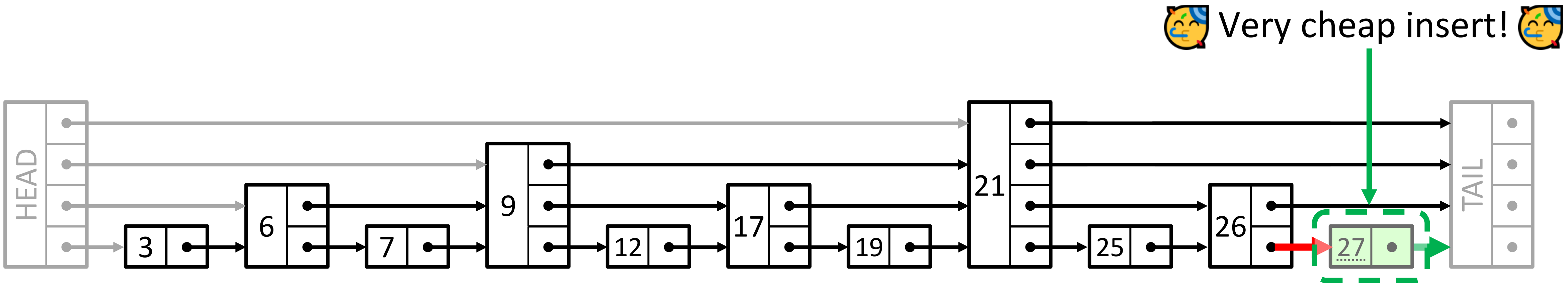
- The number of nodes across all levels can be used as an upper bound:
  - n nodes on level 1 (all nodes)
  - $\frac{n}{2}$ nodes on level 2 (every 2nd node)
  - $\frac{n}{4}$ nodes on level 3 (every 4th node)
  - …
  - Entire list: $$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots = n + n \cdot \underbrace{\sum_{k=1}^{\infty} \left(\frac{1}{k}\right)^k}_{\text{geometric series}} = n + n = \underline{2n}$$



The Perfect Skip List: But What About Updates?

- Value updates are always efficient (search + replace node value)
- Insert and delete operations **can be efficient**!
  - Example: Inserting 27

    → Structure remains intact with only minor changes (and removing it would be easy as well)

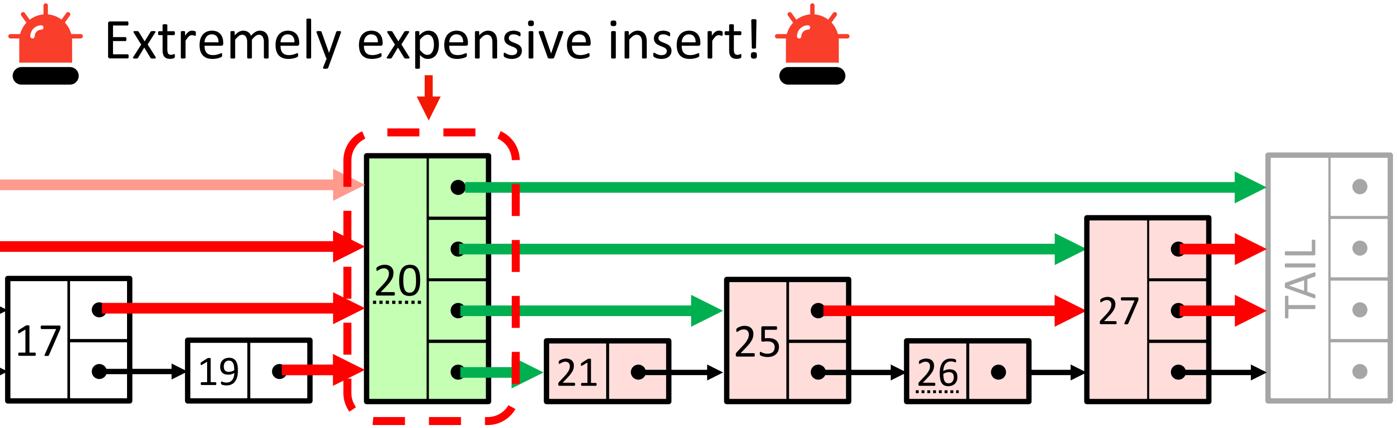


The Perfect Skip List: But What About Updates?

- Value updates are always efficient (search + node value change)
- Insert and delete operations **can be efficient**!
  - Example: Inserting 27

    → Structure remains intact with only minor changes (and removing it would be easy as well)
- But they **can also require (prohibitively) expensive restructuring** to keep the perfect structure!
  - Example: Inserting 20

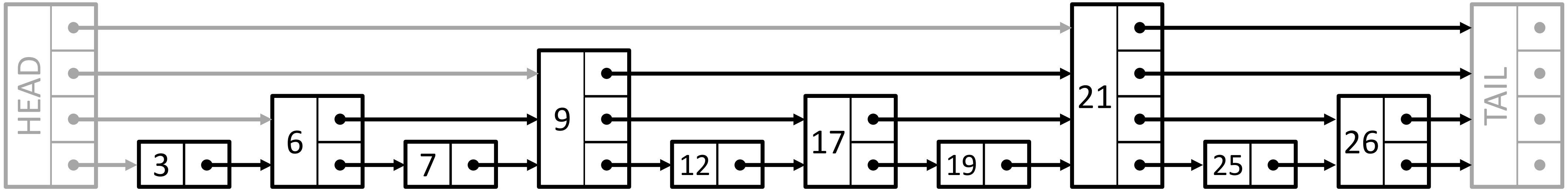
    → Keeping the structure intact is not possible without rearranging many nodes



Probabilistic Structure for Increased Robustness

- **Problem**: efficient updates are not possible while maintaining the perfect skip list structure
- **Approach**: Requirement relaxation!
  - → Exactly half of all nodes are promoted to the next level

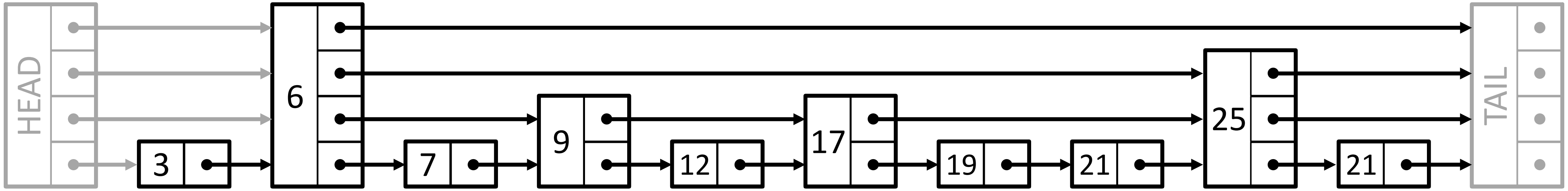
Perfect Skip List (strict requirements)



Probabilistic Structure for Increased Robustness

- **Problem**: efficient updates are not possible while maintaining the perfect skip lis structure
- **Approach**: Requirement relaxation!
  On average,
  → ~~Exactly~~ half of all nodes are promoted to the next level
  → Expected performance remains the same as with perfect skip lists!
- **Coin Flipping**: When inserting a new node, we flip a coin for every promotion decision:
  Heads: The node gets promoted to the next level and we flip again ...
  Tails: No further promotion!

~~Perfect~~ Skip List



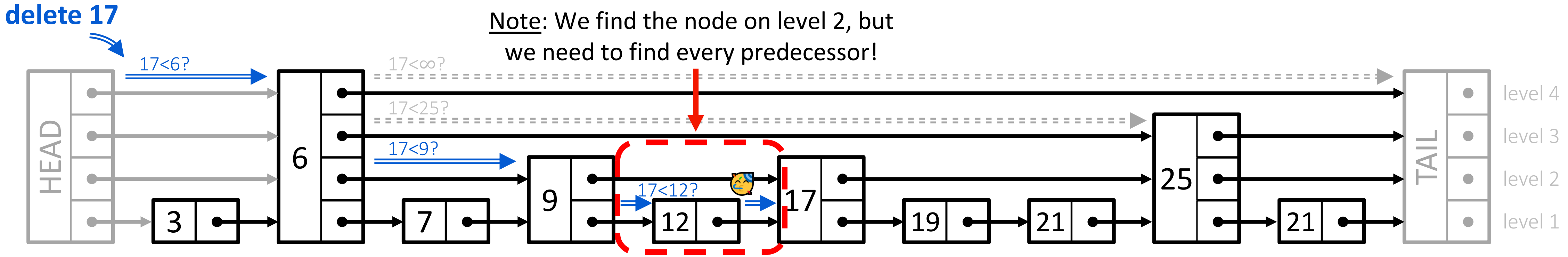
Deleting From a Skip List

- Basic **delete algorithm** for removing a node X (e.g. 17):

(1) Perform search for the to-be-deleted node X until you find the node on the normal lane

(2) On your way down, remember X's predecessor on every level → $predecessors = \begin{pmatrix} 12 \\ 9 \\ 6 \\ 6 \end{pmatrix}$ level 1, level 2, level 3, level 4

(3) …



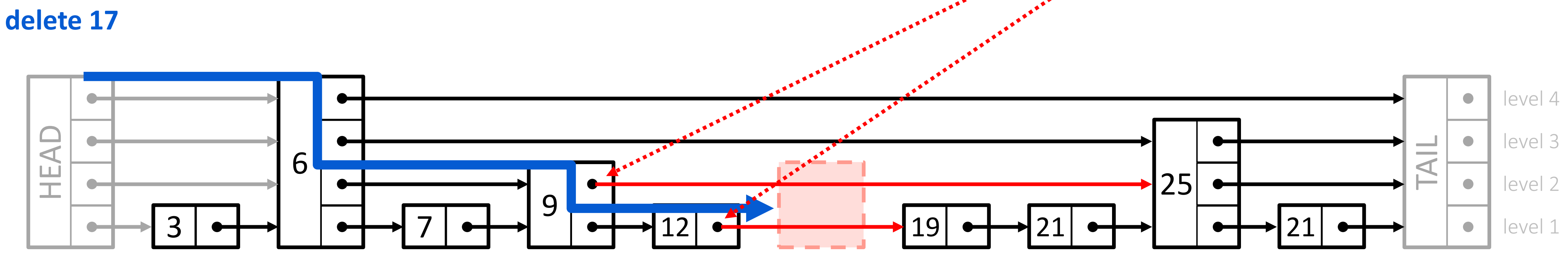
Deleting From a Skip List

- Basic **delete algorithm** for removing a node X (e.g. 17):

(1) Perform search for the to-be-deleted node X until you find the node on the normal lane

(2) On your way down, remember X‘s predecessor on every level → $predecessors = \begin{pmatrix} 12 \\ 9 \\ 6 \\ 6 \end{pmatrix}$ level 1 level 2 level 3 level 4

(3) Connect X‘s predecessors with X‘s successors and remove X



Inserting Into a Skip List

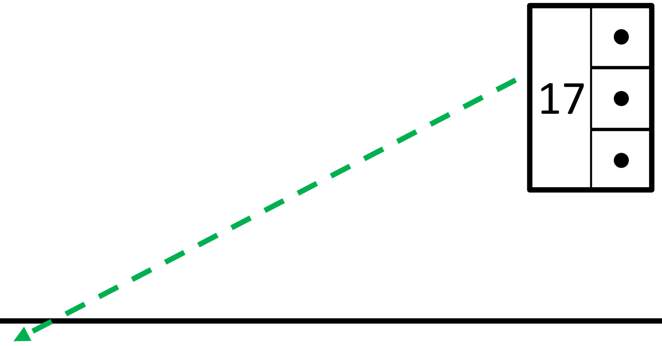
- Basic **insert algorithm** for adding a node X (e.g. 17) is very similar to the deletion algorithm:

  (1) Perform search for the to-be-*inserted* node X until you find the position on the normal lane

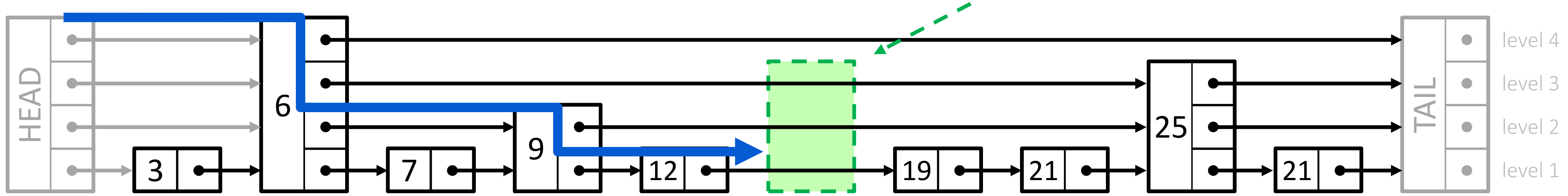
  (2) On your way down, remember X‘s predecessor on every level → $predecessors = \cdots$ (as before)

  (3) Coin flips to choose a level between 1 and max. level →

  (4) Insert the node ...



insert 17



Inserting Into a Skip List

- Basic **insert algorithm** for adding a node X (e.g. 17) is very similar to the deletion algorithm:

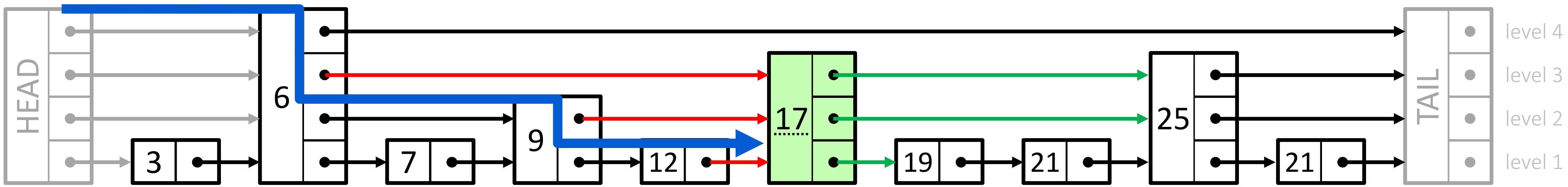
  (1) Perform search for the to-be-*inserted* node X until you find the position on the normal lane

  (2) On your way down, remember X‘s predecessor on every level → $predecessors = \cdots$ (as before)

  (3) Coin flips to choose a level between 1 and max. level →

  (4) Insert the node and update pointers on chosen levels

insert 17



About Fair & Unfair Coins: Choosing the Optimal p-Value

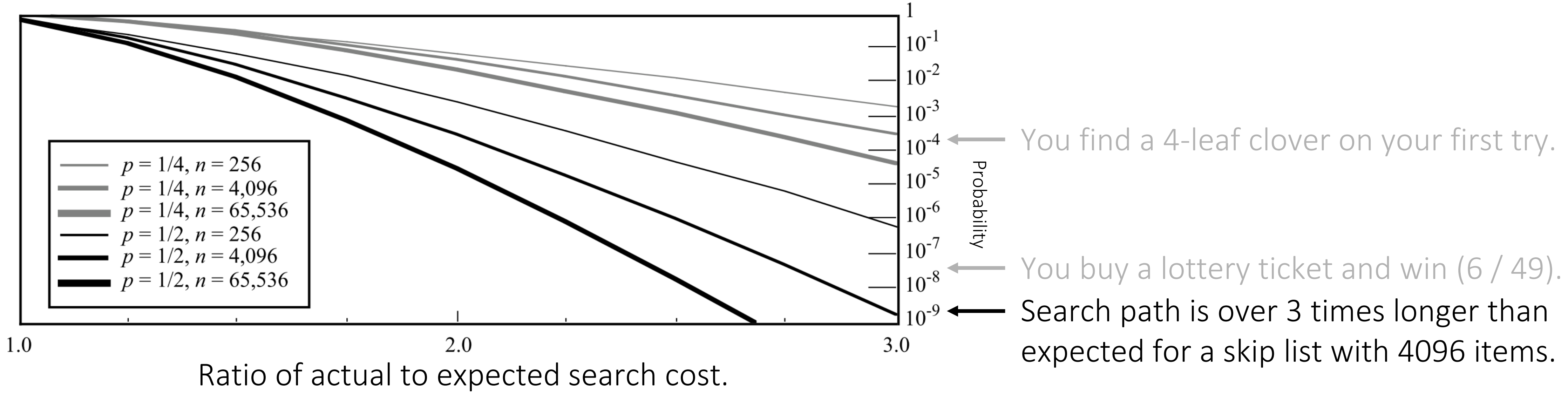
p	**Time Complexity** (Normalized $\frac{\log_{1/p} n}{p}$)	Example ($\frac{log_{1/p}\, n}{p}$ for $n = 128$)	**Space Complexity** ($\frac{1}{1-p}$, i.e. Avg. Pointers Per Node)
$\frac{1}{2} = 0.5$	1	$\frac{log_2 128}{1/2} = 7 \cdot 2 = 14$	2
$\frac{1}{e} \approx 0.368$	0.942…	$\frac{log_e 128}{1/e} \approx 4.852 \cdot e \approx 13.189$	1.582…
$\frac{1}{4} = 0.25$	1	$\frac{log_4 128}{1/4} = 3.5 \cdot 4 = 14$	1.333…
$\frac{1}{8} = 0.125$	1.333…	$\frac{log_8 128}{1/8} \approx 2.333 \cdot 8 \approx 18.666$	1.143…
$\frac{1}{16} = 0.0625$	2	$\frac{log_{16} 128}{1/16} = 1.75 \cdot 16 = 28$	1.067…

So **decreasing the p-value** (promotion probability) ...

- ... means **better storage efficiency** (i.e. fewer levels and thus fewer pointers) ...
- ... but also generally **slower searches** (i.e. more steps on avg. search path)!

William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees, CACM (1990).

Probabilistic Analysis: How Likely is a Slow Search ?



But $O(log\, n)$ with high probability (w.h.p.) does not give you any strict upper bound, so ...

- ... *with some probability*, **search might still be slow**!
- ... *in the worst case*, a **skip list can degrade to a linked list** with $log\, n$ times the normal pointers!

→ Search taking much longer than expected is **extremely rare** for lists large enough for it to matter!

Cheryl Santa Maria. What Are Your Odds of Finding a Four-Leaf Clover?, yahoo!sports (2023).

William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees, CACM (1990).

The Skip List: A Probabilistic Alternative to Balanced Trees?

„***From a theoretical point of view, there is no need for skip lists***.
Balanced trees can do everything that can be done with skip lists and have good worst-case time bounds (unlike skip lists).“

— William Pugh (1990)

Both provide $O(log\ n)$ time and $O(n)$ space complexity, so why should you choose one over the other?

→ **Skip Lists**

- *Easy to Build*: Simple operations without need for rebalance → typically easier to implement
- *Robustness*: performance is unaffected by the order of insertions → no „bad“ input sequences

→ **Balanced Trees**

- *Predictability*: Strict worst-case guarantees → no unexpected execution time spikes
- *Efficiency*: Constants are often favorable, e.g. high branching factor → shallower structure

William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees, CACM (1990).

Topics for Upcoming Lectures

Advanced Skip List Variations

Optimizations, Layering Strategies,

Complexity Analysis

Applications & Benchmarking

Implementation & Performance Shoot-Out,

In-Memory vs. Persistent Storage, Tuning

Other Probabilistic Data Structures

Bloom Filters, Count-Min Sketch, HyperLogLog,

Trade-Offs & Optimization Goals

Topics for Upcoming Lectures

Advanced Skip List Variations

Optimizations, Layering Strategies, Complexity Analysis

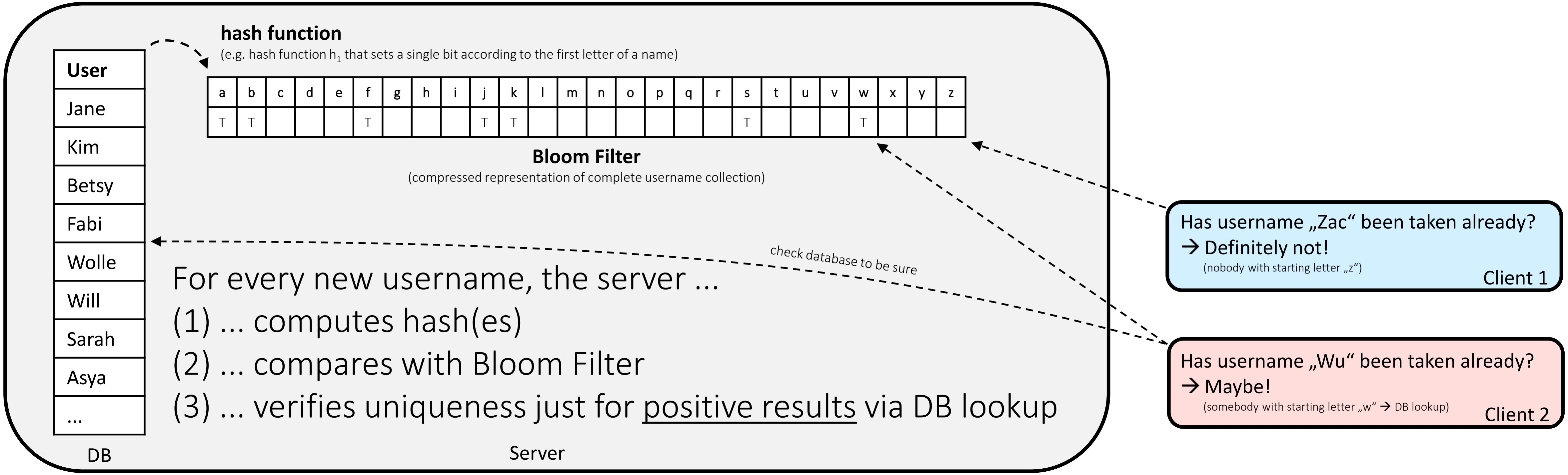
Applications & Benchmarking

Implementation & Performance Shoot-Out, In-Memory vs. Persistent Storage, Tuning

Other Probabilistic Data Structures

Bloom Filters, Count-Min Sketch, HyperLogLog, Trade-Offs & Optimization Goals

Bloom Filter Challenge: Checking for Membership



- **Problem**: Checking the DB for username availability on every registration is expensive!
- **Optimization**: Only ask DB on positive Bloom Filter check!
  - Trade-off: memory efficiency vs. false-positive rate
  - Tuning parameters: number of bits & number of hash functions
- Hash collisions only produce false positives, but never false negatives!

Niema Moshiri: Advanced Data Structures: Bloom Filters, YouTube (2020).

Bloom Filter Challenge: Checking for Membership

User
Jane
Kim
Betsy
Fabi
Wolle
Will
Sarah
Asya
...

DB

hash function
(e.g. hash function h_1 that sets a single bit according to the first letter of a name)

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
T	T				T				T	T								T				T			

Bloom Filter
...llection)

Speed Kit's web acceleration is only possible because of the **Cache Sketch**, a probabilistic data structure based on Bloom Filters!

For every new username...
(1) ... computes hash(es)
(2) ... compares with Blo...
(3) ... verifies uniqueness

Se...

...as username „Zac" been taken already?
→ Definitely not!
(nobody with starting letter „z")
Client 1

...as username „Wu" been taken already?
→ Maybe!
(somebody with starting letter „w" → DB lookup)
Client 2

Wolfram Wingerath. Speed Kit: A Polyglot & GDPR-Compliant Approach For Caching Personalized Content, ICDE (2020)

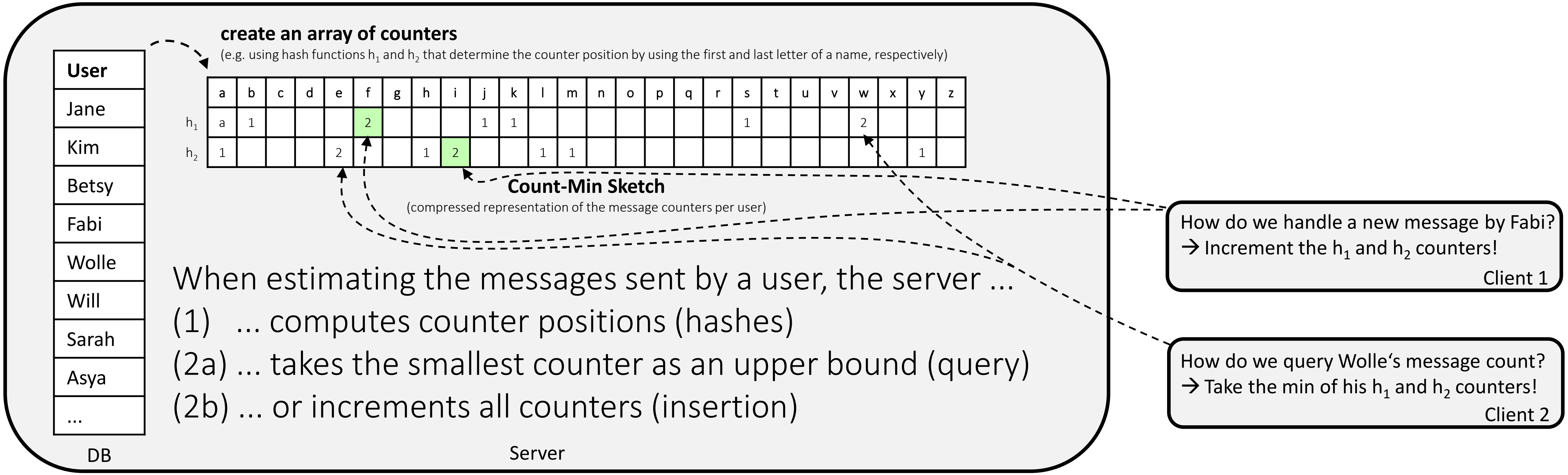
- **Problem**: Checking the DB fo... ...ion is expensive!
- **Optimization**: Only ask DB on po...
  - Trade-off: memory efficiency vs. false-positive rate
  - ...f bits & number of hash functions
  - ...ositives, but never false negatives!

F. Gessert, M. Schaarschmidt, W. Wingerath, S. Friedrich, N. Ritter: The Cache Sketch: Revisiting Expiration-based Caching in the Age of Cloud Data Management, BTW 2025

W. Wingerath, F. Gessert, E. Witt, H. Kuhlmann, F. Bücklers, B. Wollmer, N. Ritter. Speed Kit: A Polyglot & GDPR-Compliant Approach For Caching Personalized Content, ICDE 2020

Niema Moshiri: Advanced Data Structures: Bloom Filters, YouTube (2020).

Count-Min Sketch Challenge: Estimating Item Frequencies



- **Problem**: The space for keeping one message counter per user grows linearly with your user base!
- **Optimization**: Count hashes instead of users!
  - Trade-off: memory efficiency vs. overcounting error
  - Tuning parameters: number of counters & number of hash functions
- Counts are upper bounds, since hash collisions only lead to overcounting!

Niema Moshiri: Advanced Data Structures: Count-Min Sketches, YouTube (2020).

Summing up: Probabilistic Data Structures Are Awesome!

- **Skip Lists** combine elements from sorted linked lists and array lists to achieve
  - *Simplicity: straightforward implementation, extension & modification*
  - *Efficiency:* $O(log\, n)$ *Time Complexity* for inserts, deletes & search with high probability
  - Robustness: no „bad“ sequences, no rebalancing, no sophisticated tuning required!
- **Probabilistic Data Structures** in general are used across a variety of **Applications** including
  - *Order-Preserving Dynamic Collections (Skip Lists)*
  - *Efficient Membership Tests Without False Negatives (Bloom Filters)*
  - *Estimating Upper Bounds for Item Counts (Count-Min Sketch)*
  - *Many More, e.g. Counting Unique Visitors (HyperLogLog)*

Thanks! Questions?

https://uni-oldenburg.de/awesome-job

Looking for a **job in research**?
Apply as a research assistant and
do your PhD in my group NOW!